

Games + Adversarial Search

- Ways to play
- Mini max
 - scoring function
 - $\alpha \beta$
 - Progressive deepening
- Deep Blue

Using computers to play Chess

- later

Quick Review

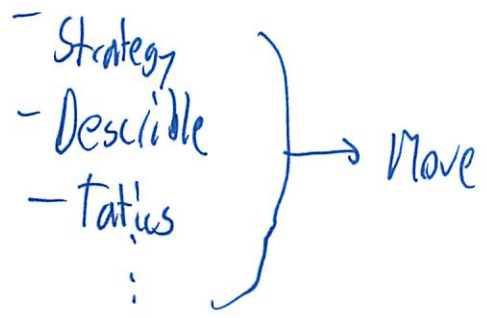
- Extended \leftarrow
- ~~Expanded~~ required for some searches
- Expanded \leftarrow but works for all, so just use it
- Enqueue \leftarrow look at nearby \leftarrow extended on that
 - never use
- added to queue, does not work for some searches

List of Presidents

- all played chess on sight
- need to think several steps ahead of your opponent
- so not just next step, but more

②

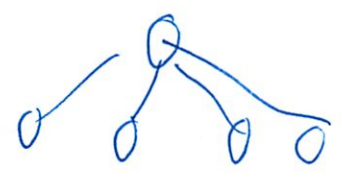
① How humans do it



② If-then

Chess is too complicated

③ Rank possible positions

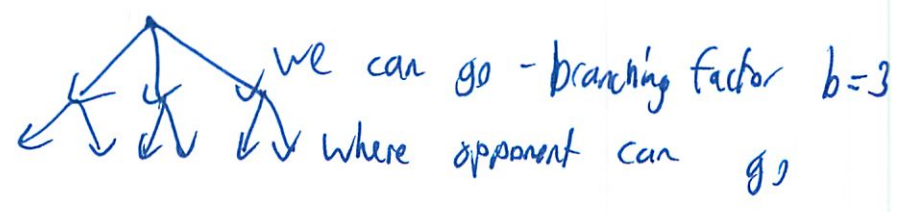


but no evaluator which tells you which board is best

④ British Museum

Is that practical?

depth
here
 $d=2$



So # of places $b^d = 3^2$

Claude Shannon - Chess 10^{120} end points

3

Note chess terms, but we will just call 1 ply = 1 move

If it took 10^4 seconds to evaluate a possibility then we can do 10^7 sec/year. Even w/ 10^{80} atoms in the universe - have to wait a while

5 Can we look forward + hope situation clarifies like look at piece count, etc

So reduce game down to single #

↳ from a static evaluator

- evals w/o look ahead

$$S = F(f_1, f_2, f_3, \dots)$$

↳ a whole bunch of features

- piece count

- king safety

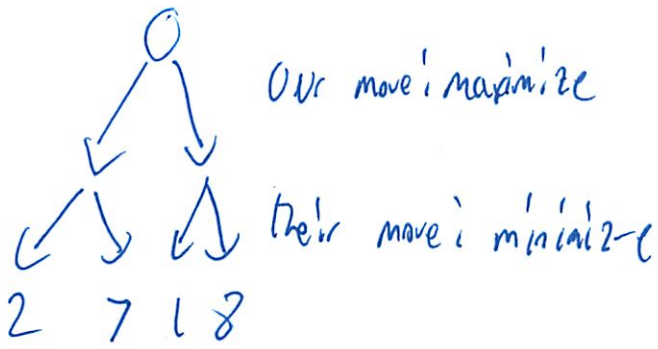
- pawn structure

- general case: scoring polynomial

- linear

$$S = C_1 f_1 + C_2 f_2 + \dots + C_n f_n$$

9

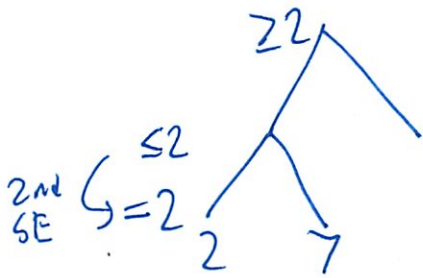


Minimax

Evaluate many levels down to see where best possible outcome is

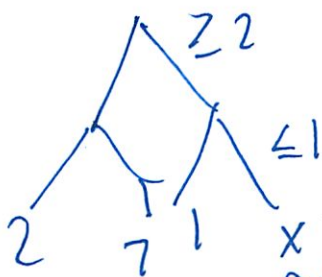
This is what deep blue does. - goes 14-16 ply moves
 b^d can be painfully large though

When doing static eval, store what is best he can do



2 steps SE α, β add on

then



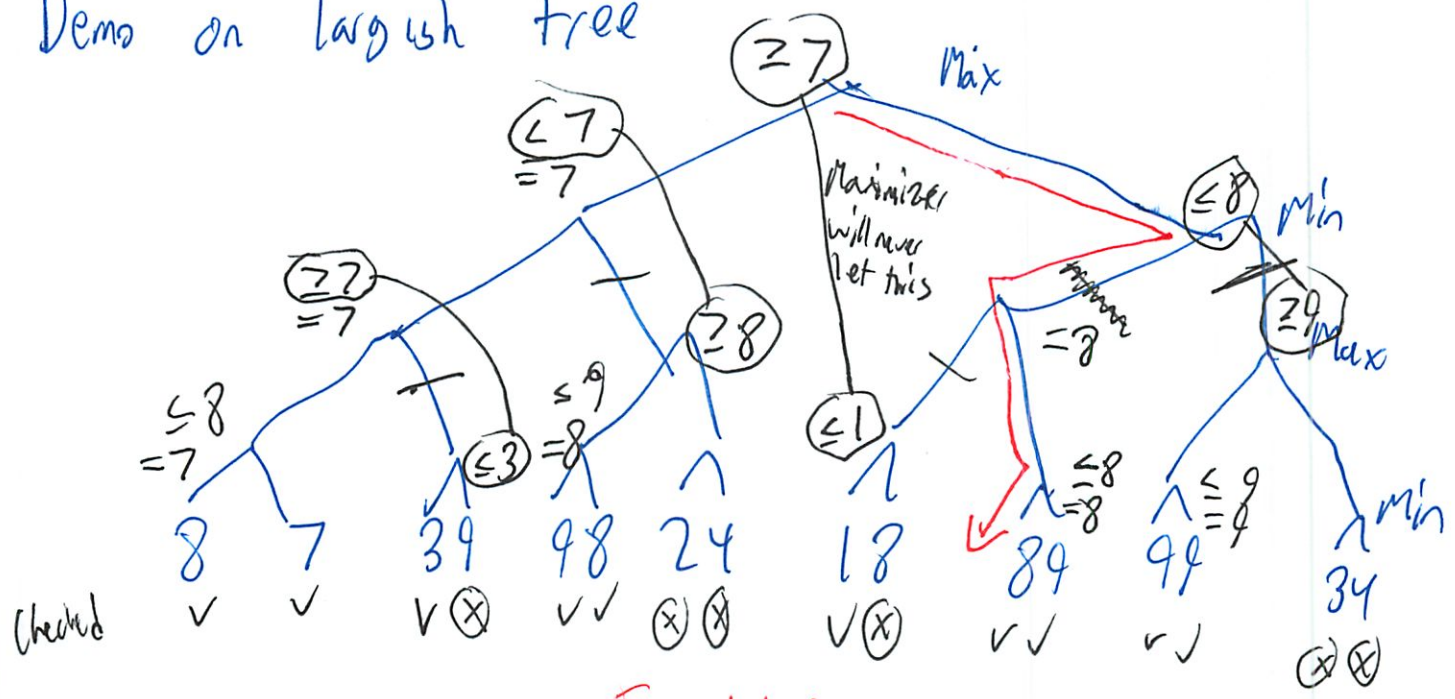
α, β add on

↑ don't care what this is
bc minimize will go to 1

5

So if assume opponent has same scoring polynomial (is rational, etc) then it will give you exactly same answer \uparrow α, β

Demo on largish tree



- Expected Path

Not necessarily the path it would take
- but this is best possible at M's pt
- Reevaluate later,

α = Bound as low as can go
 β = " " high " " ") Record in boxes
 α/β

6

Static evaluation best case

$$S_{\text{Best}} = b^{\frac{d+1}{2}} + b^{\frac{d-1}{2}} - 1 \quad d \text{ odd}$$
$$= 2b^{\frac{d}{2}} - 1 \quad d \text{ even}$$

- must be extremely likely
- proportional to $b^{d/2}$
- So if use α, β will cut list in half
 $\frac{d}{2}$

Every game program uses α, β

All this assumes tree is uniform

So hard to decide how deep to go

- leave computation on table
- or take too long

2 parts: generate tree and static evaluation

If not sure will get there want an insurance policy
- what it times up, what will you say

7

Insurance costs - need since don't know how many branches



Can take insurance on insurance $S = b^{d-2}$

So how far up to buy insurance for?

If bought all

$$S = 1 + b + b^2 + \dots + b^{d-1}$$

$$bS = b + \dots + b^d$$

$$S(b-1) = b^d - 1$$

$$S = \frac{b^d - 1}{b - 1}$$

So ratio = ~~$\frac{b^d}{b^d}$~~ $\frac{b^d}{b^d(b-1)}$ = $\frac{1}{b-1}$

(amt of work in excess of what I had to do)

So all insurance policies basically, cost of doing exponential
Will always have more when out of time

L = any time algorithm

- want one exp if takes various time

⑦ Called progressive deeping - gives you same answer to certain depth

So you do minimax, α/β , progressive deepening

"Game of century"

Exam wed ?

Test: ~~list~~ of Rule w/ OR() which do
list

OR part 1
→ Through assertions

OR part 2
→ etc

Or
Through assertions
- whatever OR

Prof. Bob Brwick does not know - can do ~~at~~ any way

Quiz back at end

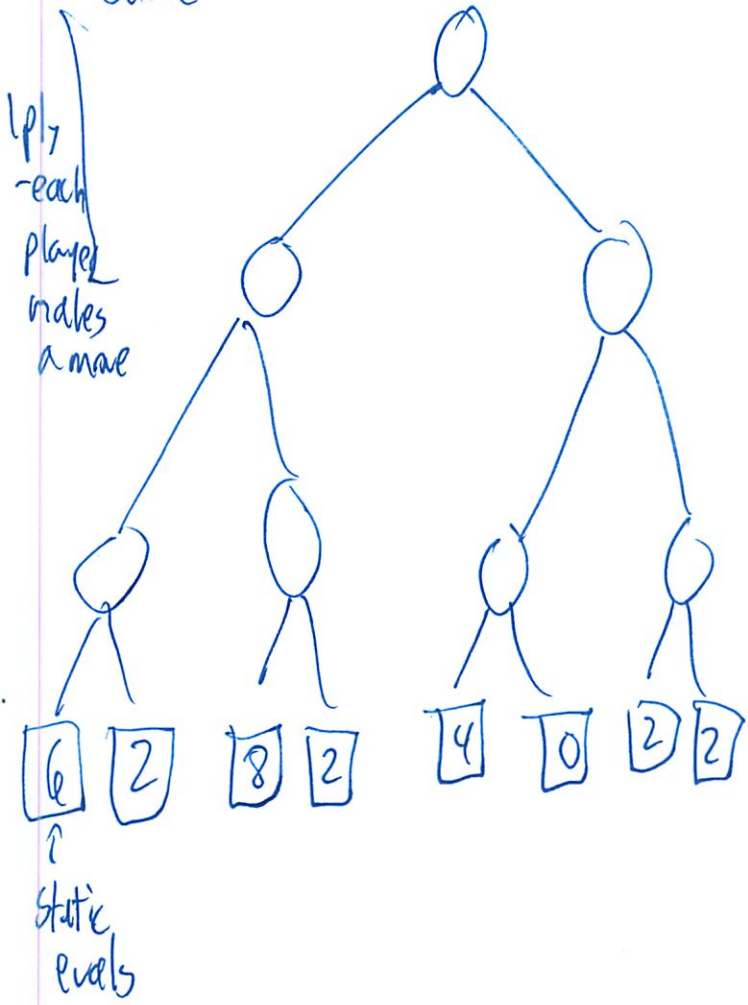
1	84-100
2	75-88
3	0-75

Today: Games Minimax
 α, β search

①

α - β gets same answer as minimax
— but less work

Game



Max

Min

Max

branching factor
= fanout
= here 2

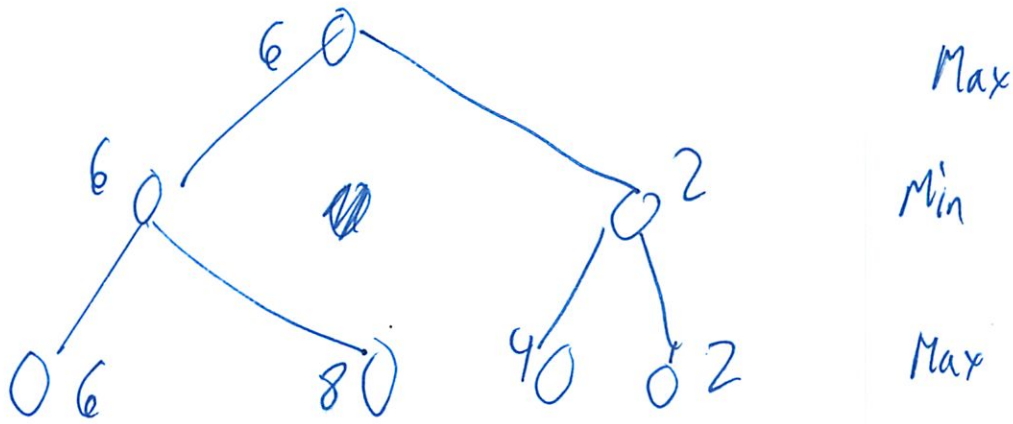
Q: What's the maximal guaranteed value I can get

So at max, I take (6) vs 2

Bubble up

Then minimizer picks up earlier

(3)



↑
So pick
this path

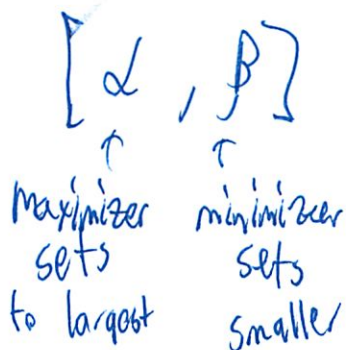
In α - β search lots of branch cut
- esp if going 40-ply in chess

So at this node



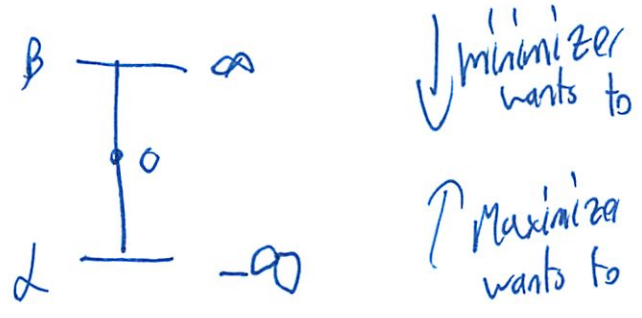
why care about 8 here anymore
(rational) opponent will never get it

So carry around an interval



Q

So before you have any info



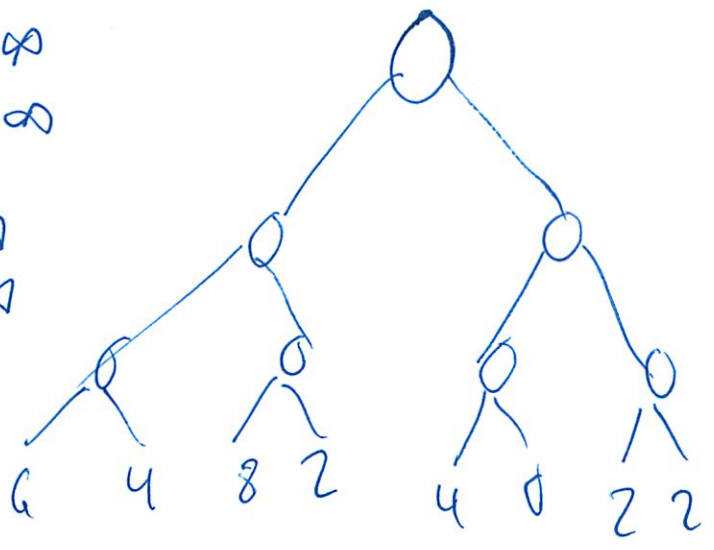
Ultimately should get to $[\epsilon, \epsilon]$

α = Maximizer guaranteed to get at least α

Can you ever have $\alpha \geq \beta$?

No! Immediately exit out of loop
Can't have in rational play

$\alpha - \infty$
 $\beta + \infty$
 $\alpha - \infty$
 $\beta + \infty$
 $\alpha - \infty$
 $\beta + \infty$



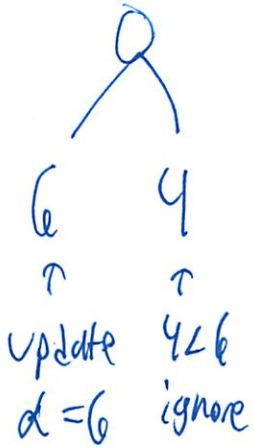
Maximizer - Max α , only update α

Min - Min β , only update β

Max - α

5

So move →



So $d=6$

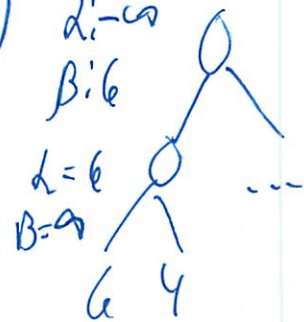
Done w/ subnode

Return $d=6$ upwards

Now at next level up (since DFS)

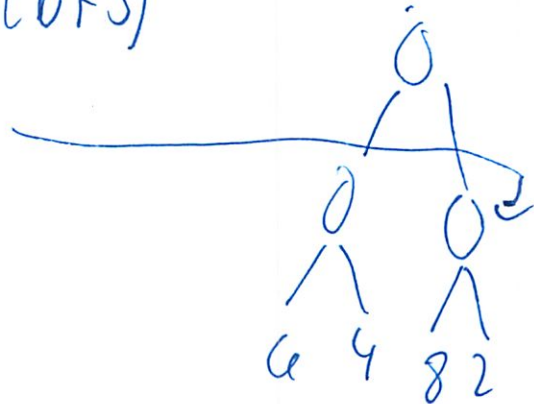
I have $+\infty$

But now I also have 6



Now go back down (DFS)

We have $d = -\infty$
 $B = +6$



See ~~8~~ 8,
better than ∞
So $d=8$

So now
~~now look at next~~

before looking at 2

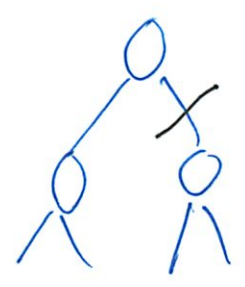
We see now $8 \geq 6$ which is

wrong!
Return

immediately
Don't look at 2 cutoff

6

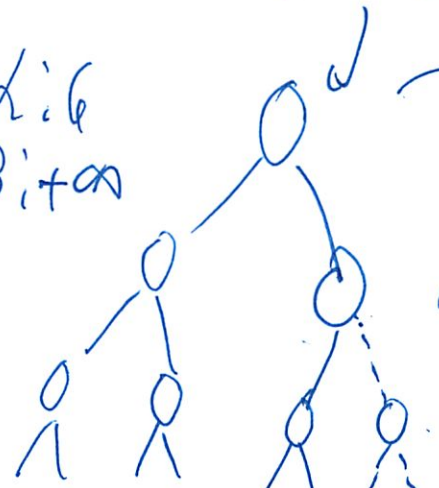
Now



$d: -\infty$
 $\beta: 6$ ✓ confirmed
 guaranteed

So return β up one level
 - at top level

$d: 6$
 $\beta: +\infty$



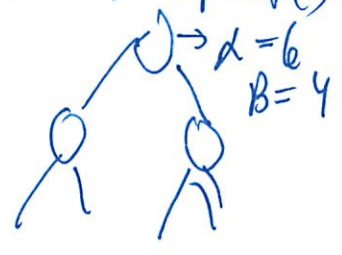
Now do recursive calls down

$d = 6$
 $\beta = +\infty$

$4 \uparrow 0$
 now look here

$d = 4$
 $\beta = +\infty$ look at 0
 - not bigger

Return 4 upwards



But now $d \geq \beta$
 So can stop evaluating anything below
 this node



①

Return ~~α~~ ~~β~~ $\alpha = \beta$ to parent node

So parent node $\alpha = \beta$
 $\beta = +\infty$

So we ~~evaluated~~ saved evaluating $\frac{3}{8}$ nodes

which is best we can do w/ ~~α~~ α, β

α, β is optimal when go highest + lowest right to left

If lowest \rightarrow highest it would be no good
- well it would not help

So want to achieve maximal α, β cutoff

Could try sorting

- but then need to evaluate

So try sorting w/ heuristic

Or remember key paths that worked before
transposition hashing

p 8 of notes projective analysis of games solving
Chess grandmasters ~~just~~ evaluate far fewer methods

②
A State of AI: only way is brute force

Is there a better way?

Not really being researched

Recitation 3, Thursday, September 29

Game trees, minimax, & alpha-beta search

Prof. Bob Berwick, 32D-728

1. Minimax search in a game tree.

Notation: Let SV denote the 'static value' or 'worth' of a particular game state/board position. $SV(\text{state})$ returns the static value of the game state. These are the numbers we give you as the leaf nodes in the game search tree. They are generally in the perspective of the current player. Here is the minimax algorithm:

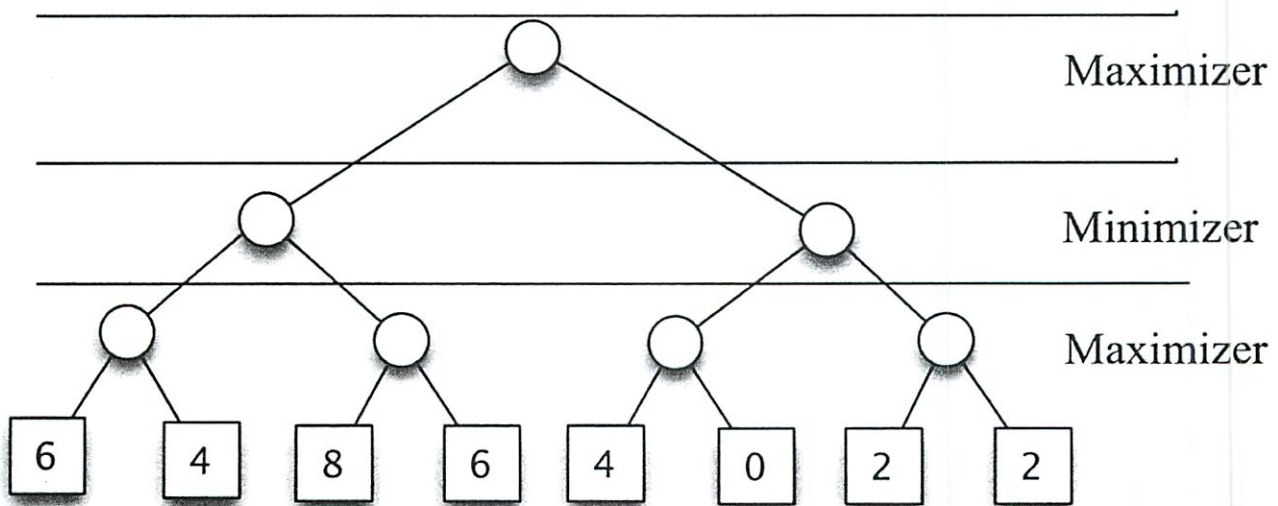
- If the limit of search has been reached, return the static value of the current position relative to the appropriate player.
- Otherwise, if level is a minimizing level, call Minimax on the children of the current position, and report the MINIMUM of the results.
- Otherwise, if level is a maximizing level, call Minimax on the children of the current position and report the MAXIMUM of the results.

Or in (somewhat wordy) pseudo-code:

```
function max-value(state, depth)
  1. if state is an end-state (end of game) or depth is 0
    return SV(state)
  2. v = -infinity
  3. for s in get-all-next-moves(state)
    v = max(v, min-value(state, depth-1))
  4. return v
```

```
function min-value(state, depth)
  1. if state is an end-state (end of game) or depth is 0
    return SV(state)
  2. v = +infinity
  3. for s in get-all-next-moves(state)
    v = min(v, max-value(state, depth-1))
  4. return v
```

Let's try minimax out on the following game tree, and figure out the line of play as well for the top-most player, the **Maximizer**:



What is the value that minimax returns here at the topmost, maximizer node? What is the line of play?

2. Alpha-beta search. Now let's think about this game tree some more. Consider the static value at the left-most **minimizer** node in the tree above. It is 6. Now, how was that computed? What happens if we change the fourth static evaluation from the left, containing a 6, to something lower, like 0? What if we make it something much higher, like 1000? Does the minimax value at the left-most minimizer node change as a result?

This result suggests that we can **avoid** ever computing the static evaluations of some game positions – a good thing, since this is expensive, in general. The upshot what is called **alpha-beta search**.

Alpha-beta search is simply an **efficient** form of minimax that gets the **same** result (evaluation at top and line of play) but with often (far) **fewer** static evaluations.

- Basic idea: track best move's value so far during minimax search, from perspective of both the maximizer and the minimizer
- For the **Maximizer**, that value is **Alpha** (goal: make it LARGE) – this is the **guaranteed best** that Maximizer can get (a **guaranteed lower bound** on the maximum value)
- For **Minimizer**, the value is **Beta** (goal: make it SMALL) – this is the **guaranteed best** that Minimizer can get (i.e., lose, a **guaranteed upper bound** on the minimum value)
- Initially, $\alpha = -\infty$; $\beta = +\infty$; so this is initially a range $[\alpha, \beta] = [-\infty, +\infty]$. As we search the tree, this range should narrow such that $\alpha = \beta$, and the values **should never cross** (see pruning below)
- Pruning:
 - When the Minimizer is examining its moves, determining **beta**, if any are **less than or equal to alpha** (**worse** for Maximizer), then the Minimizer's parent, the Maximizer, would **never** make that move because the move that yielded alpha is already better – so the Minimizer can abandon this node! (In other words: cut-off if ever $\beta < \alpha$)
 - When the Maximizer is examining its moves, determining **alpha**, if any are **greater than beta** (**worse** for Minimizer) then the Maximizer's parent, the Minimizer, would never make that move because the move that yielded beta is already better – so Maximizer can abandon this node! (In other words: cut-off if ever $\alpha < \beta$)

Remember this: **Maximizer calculates alpha; Minimizer calculates beta; cut-off if alpha ever crosses beta or vice-versa.**

Pseudo-code (here SV = 'static value' of a node)

```
function max-value(state, depth, alpha, beta):
  1. if state is an end-state (end of game) or depth is 0 then
    return SV(state)
  2.  $v = -\infty$  # initial ALPHA value for node
  3. for s in get-all-next-moves(state)
     $v = \max(v, \text{min-value}(state, depth-1, alpha, beta))$ 
     $\alpha = \max(\alpha, v)$ 
    if  $\alpha \geq \beta$  then # alpha-beta cutoff!
      return alpha # exit from loop immediately 4.
return v
```

$\alpha \geq \beta$ cutoff!

```
function min-value(state, depth, alpha, beta):
  1. if state is an end-state (end of game) or depth is 0 then
    return SV(state)
  2.  $v = +\infty$  # initial BETA value for node
  3. for s in get-all-next-moves(state)
     $v = \min(v, \text{max-value}(state, depth-1, alpha, beta))$ 
     $\beta = \min(\beta, v)$ 
    if  $\alpha \geq \beta$  then # alpha-beta cutoff!
      return beta # exit from loop immediately
  4. return v
```

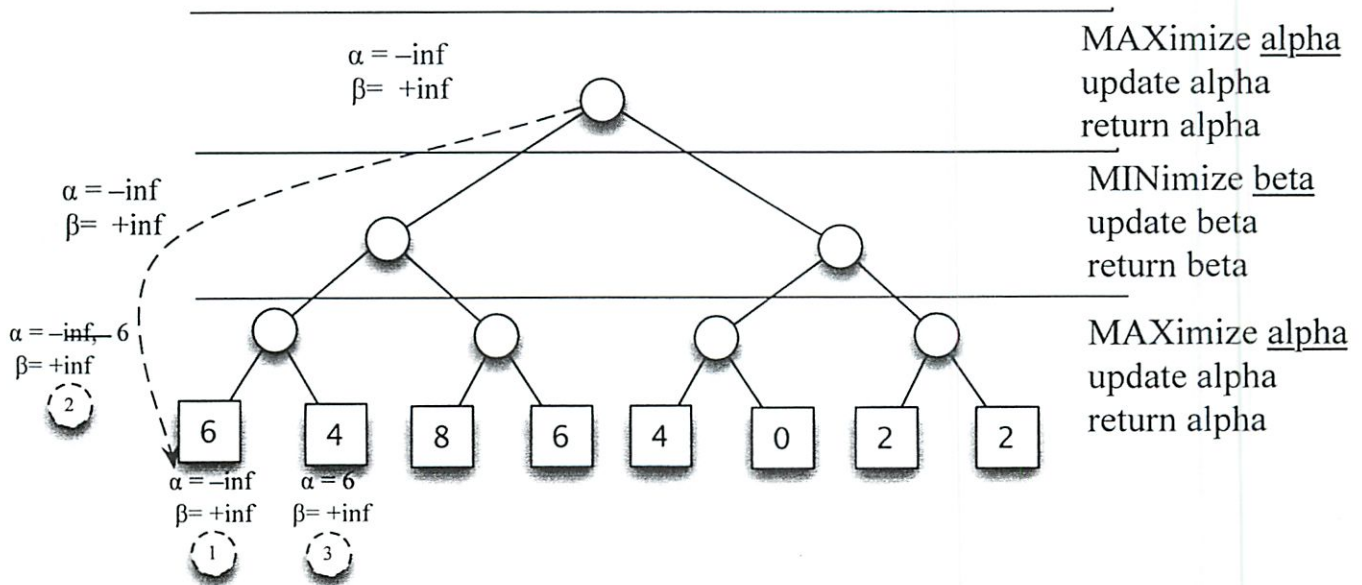
Initially we set Alpha and Beta to their worst-case values: $\alpha = -\infty$ (or some very large negative value), and $\beta = +\infty$.

Note 1. In the code the **Minimizer** loop returns the value for **Beta** (remember, it's setting the upper bound, the **worst case** for the minimizer), while the **Maximizer** loop returns the value for **Alpha** (it is setting the lower bound, the **worst case** for the maximizer).

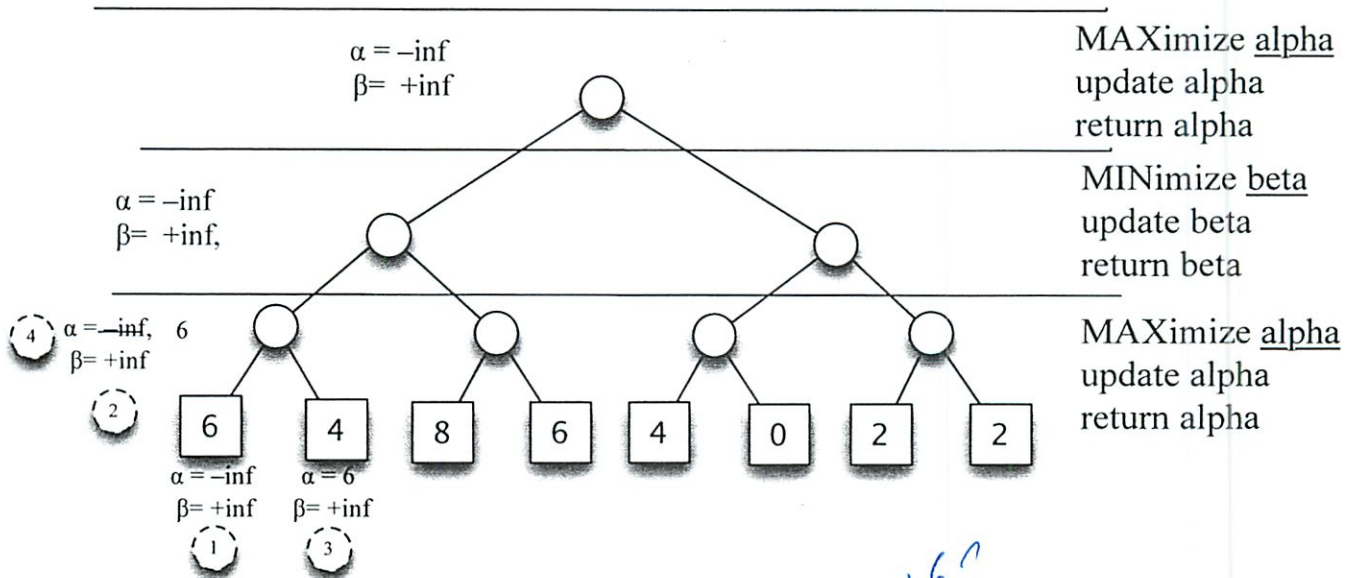
Note 2. The loops over daughters of a node are cut-off if ever alpha exceeds beta or vice-versa. **Repeat the mantra after me: ALPHA CAN NEVER BE GREATER THAN BETA (OR VICE-VERSA) IN ALPHA-BETA SEARCH.**

Note 3. The Alpha (conversely Beta) values are related as we move from level to level in the game tree: the Alpha (resp. Beta) values are passed *up* to be used at the next level in the minimax tree to become the refined Beta (resp. Alpha) bounds at the next level. What is the reasoning here? For example, assuming initial Alpha, Beta values of, say, $-/+infinity$, then if the Maximizer can set Alpha to be equal to, e.g., 100, we know that from the Minimizer's point of view one level above, the Minimizer **can do no worse than this**, so Beta at this level may now be decreased from $+infinity$ to, tentatively, 100 (thought their loss might wind up being even less). In other words, if I (the maximizer) am guaranteed to *win* \$100, then my opponent (the minimizer) cannot *lose* more than this amount so far (they might wind up losing less, as other parts of the tree are explored). One person's ceiling is another person's floor (as the song goes).

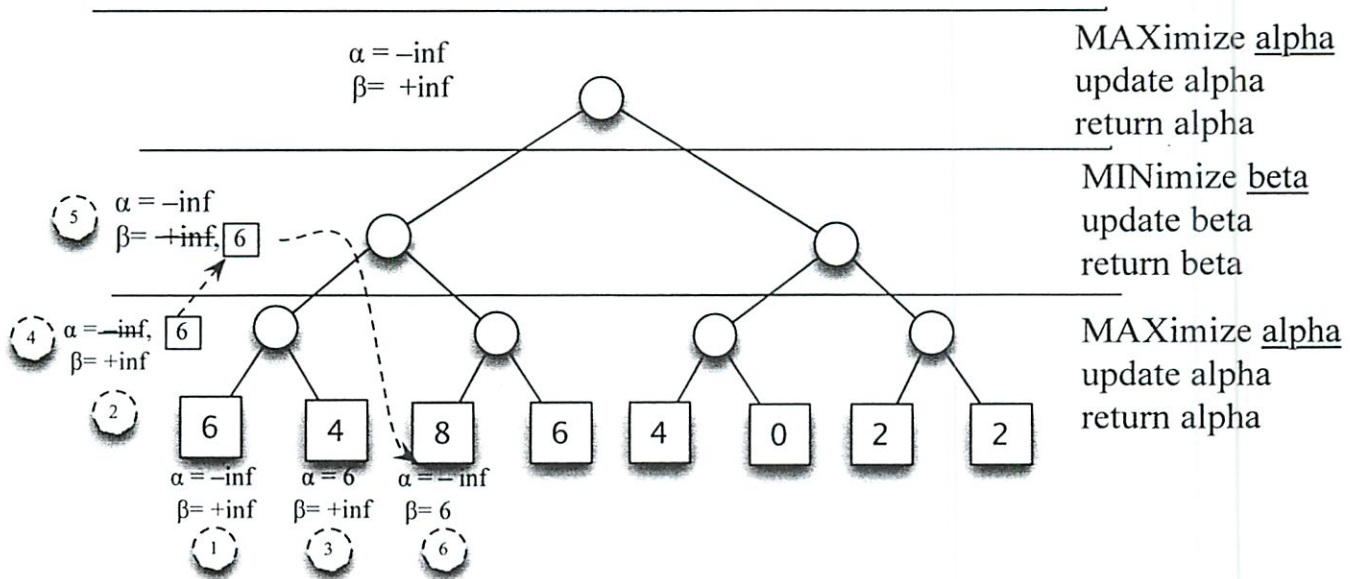
So now let's see how all this works and now alpha-beta search can save us effort with static evaluations in some cases. Let's go through the *same* minimax tree as before (so we should get the same answer!), but using alpha-beta pruning. Let's pay careful attention to how the values of alpha and beta are narrowed down, as well as how they are passed both up the tree (as return values) and down the tree. The first snapshot shows just the initialization of alpha, beta and diving down to the first static evaluation node on the left, at the bottom. Step numbers are shown inside dotted circles. Since this is a maximization node, we (tentatively) set Alpha to be 6 (it might get larger, since we are maximizing):



Steps 3-4: Carrying on, as shown just below, we evaluate the right-most sister of the bottom node, with a static value of 4. Since at this level we are *maximizing* (setting Alpha), this is smaller than the alpha value we have already, of 6. So we now can set Alpha=6 for this entire node, shown as Step #4.

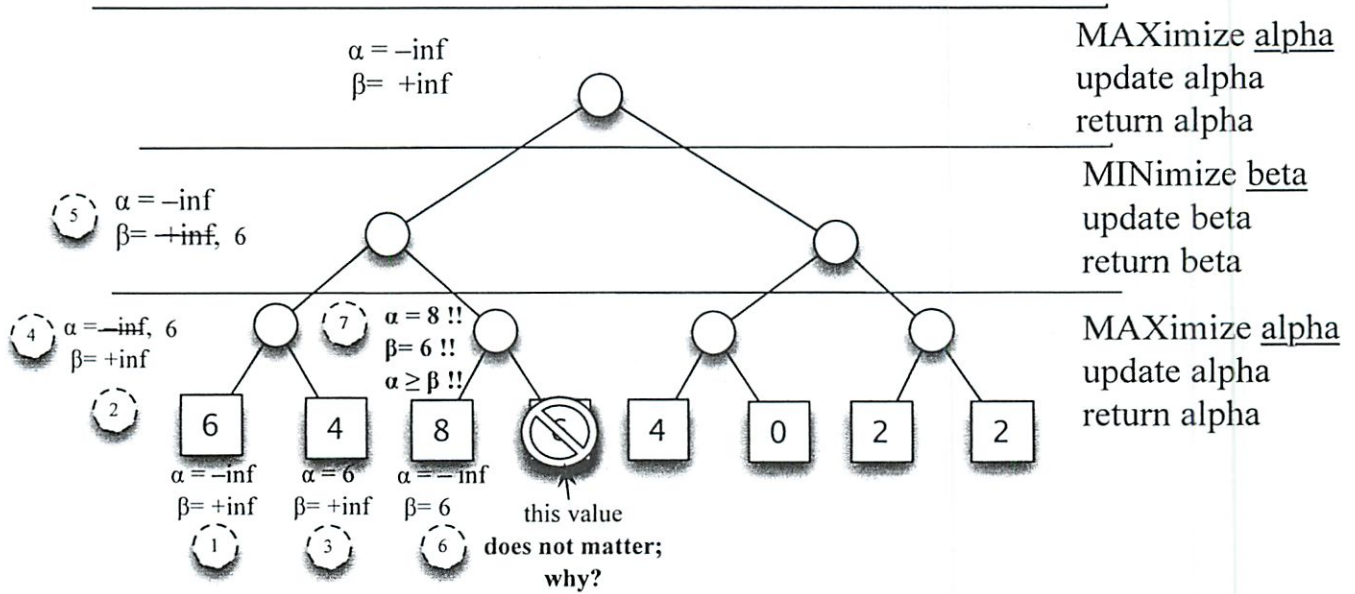


Steps 5-6: Now we are done with this MAXimizing subnode, so we can *return* from the MAXimizing level at this point, with Alpha=4, to the MINimizing level right above it. Since Alpha=4, recall this means that one level up, at the MINimizing level, we can (tentatively) set Beta to 4, and this is shown as Step #5 below, along with arrow indicating how the Alpha value has been transferred to its Beta value counterpart. (Note that the Alpha value at this level is still -Infinity, because this value can only be updated by running minimax on the left *above* this one). Finally, we can pass these values of α , β of $(-\text{inf}, 6)$ to the down to the leaf nodes at the Maximizer level again, shown as Step #6:

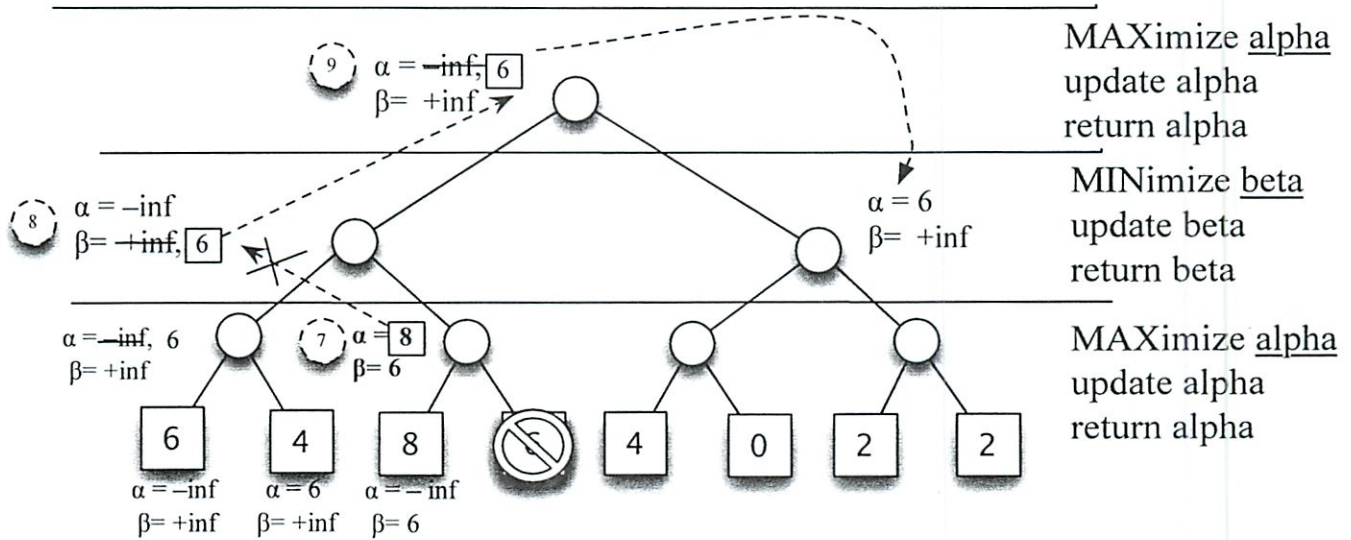


Step 7, alpha-beta cut-off: OK, since we're at a MAXimizing level, the static evaluation value of 8 at the left-most node tells us that we can set α tentatively to 8. If we do that, we arrive at Step #7 as shown in the figure immediately below. **Note now that we have set $\alpha=8$ and $\beta=6$.**

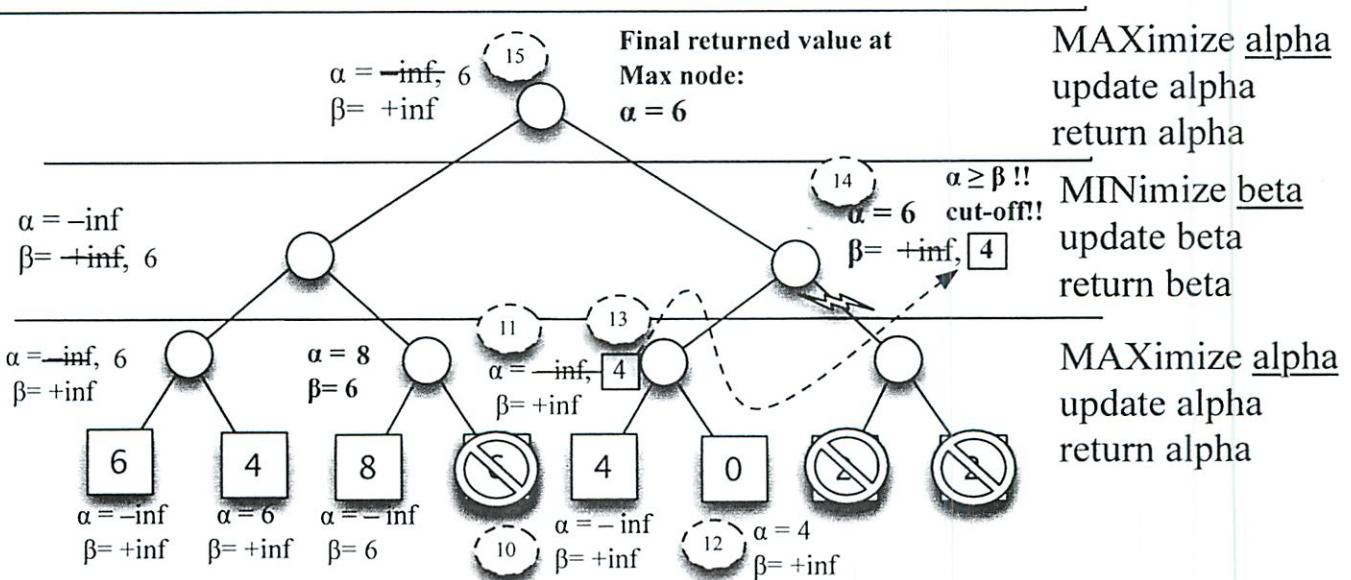
DANGER DANGER WILL ROBINSON!! α is NOW GREATER than β !! Therefore, we can EXIT from this MAXimizer call immediately **without ever having to evaluate the node on the right with its static evaluation of 6.** (We have drawn a big "NO" symbol through this node. This static evaluation, and that of any other static evaluation to the right underneath this maximizing node CANNOT NOT MATTER to the final result returned here. Can you remember WHY? If you ever forget, just trying making the crossed out static value either very small or very large, and you'll see it makes no difference to the α value.)



Steps 8-9: Note that the value returned from the MAXimizer search then is $\alpha = 8$. Now we're back at the MINimizer level, and have the job of setting β , so $\alpha = 8$ at the MAXimizer level implies $\beta = 8$ for the MINimizer, using the same reasoning as before. Taking the minimum of the two β values, 6 and 8, we can now fix $\beta=6$, at Step #8. Since $\beta=6$ at this MINimizer level, this implies that the most that can be **lost** from this node is 6; therefore, the minimum that can be **gained** at the next level up, the higher MAXimizer level, is at least 6, that is, (tentatively) $\alpha = 6$, marked as Step #9, with β still at +infinity. We then pass the $\alpha = 6, \beta = +\text{inf}$ values down one level.



Steps 10-15: And now we can quickly roar through the rest of the tree, noting a final alpha-beta cutoff....note that when the dust settles, we get the *same* returned value 6 for the game tree (and the same line of play) as we did before with pure minimax, just as claimed – the line of play is always to the *left* down from the root node:



How effective is alpha-beta pruning? In this case, in all, we **saved 3 static evaluations** via **two cut-offs**, out of 8 possible static evaluations, about $\frac{1}{2}$ of them. This turns out to be the theoretical best savings. It depends on the order in which children are visited. If children of a node are visited in the worst possible order, it may be that no pruning occurs. For max nodes, we want to visit the best child first so that time is not wasted in the rest of the children exploring worse scenarios. For min nodes, we want to visit the worst child first (from our perspective, not the opponent's.) When the optimal child is selected at every opportunity, alpha-beta pruning causes all the rest of the children to be pruned away at every other level of the tree; only that one child is explored. This means that on average the tree can be searched twice as deeply as before—a huge increase in searching performance.

Note that the best case holds for the game tree above: at each of the last MAXimizer nodes, the leaves below are ordered from maximum to minimum values, i.e., (6, 4) for the children of the first node; (8, 6) for the second; (4, 0) for the next, and so on. More generally, we assume that the *leftmost* alternative is always the best move, for *all* levels. Suppose the tree is d levels deep (excluding the static evaluation layer), and has branching factor b . Then one can show that in general the number of static evaluations s needed is:

$$s = 2b^{d/2} - 1 \text{ for } d \text{ even}$$

$$s = b^{(d+1)/2} + b^{(d-1)/2} - 1 \text{ for } d \text{ odd}$$

In our case, $d=2$ and $b=2$, so the formula yields $2 \times 2^1 - 1 = 4 - 1 = 3$ as confirmed. Note that the value in both cases is reduced by the *square root* of the number of nodes in a full static evaluation, b^d .

Note that we can't in general ensure that the static evaluations in a game tree are 'optimistically' ordered in this way, because that would involve evaluating the nodes, exactly what we want to avoid. For example, suppose the order of the leaves of the tree above were the reverse: (2, 2); (0, 4); (6, 8); (4, 6). You should see that alpha-beta search on this reversed order gives zero cut-offs, because at the bottom, maximizer level the leaves are in *ascending* order under each maximizer node, so we have to explore each of them. This is the worst case when maximizing. (The converse is true when minimizing.)

There are several ways to obtain the information about how to order nodes. First, one can develop heuristic methods that seem to do a good job of approximating such an ordering, sometimes thinking about the particular game itself. Second, what is sometimes done is to use a 'history heuristic' to remember what previous searches of the game tree, from previous moves (in the current game or in the past) have led to strong alpha-beta pruning, and/or strong evaluations at the 'root' node, and then 'hash' code these so that if they arise again, they can be efficiently re-used.

Chess has seen the development of many, many other heuristic methods. Some of these are: (1) transposition tables, or a hash memory of common, previously encountered board positions with associated scores, regardless of how we got there; (2) refutation moves, that is, alpha-beta search until we get at least a (sufficiently good) alpha value in response to the opponent's play, but not necessarily the best alpha-beta score; and (3) principal-variation moves (used with progressive deepening, see just below).

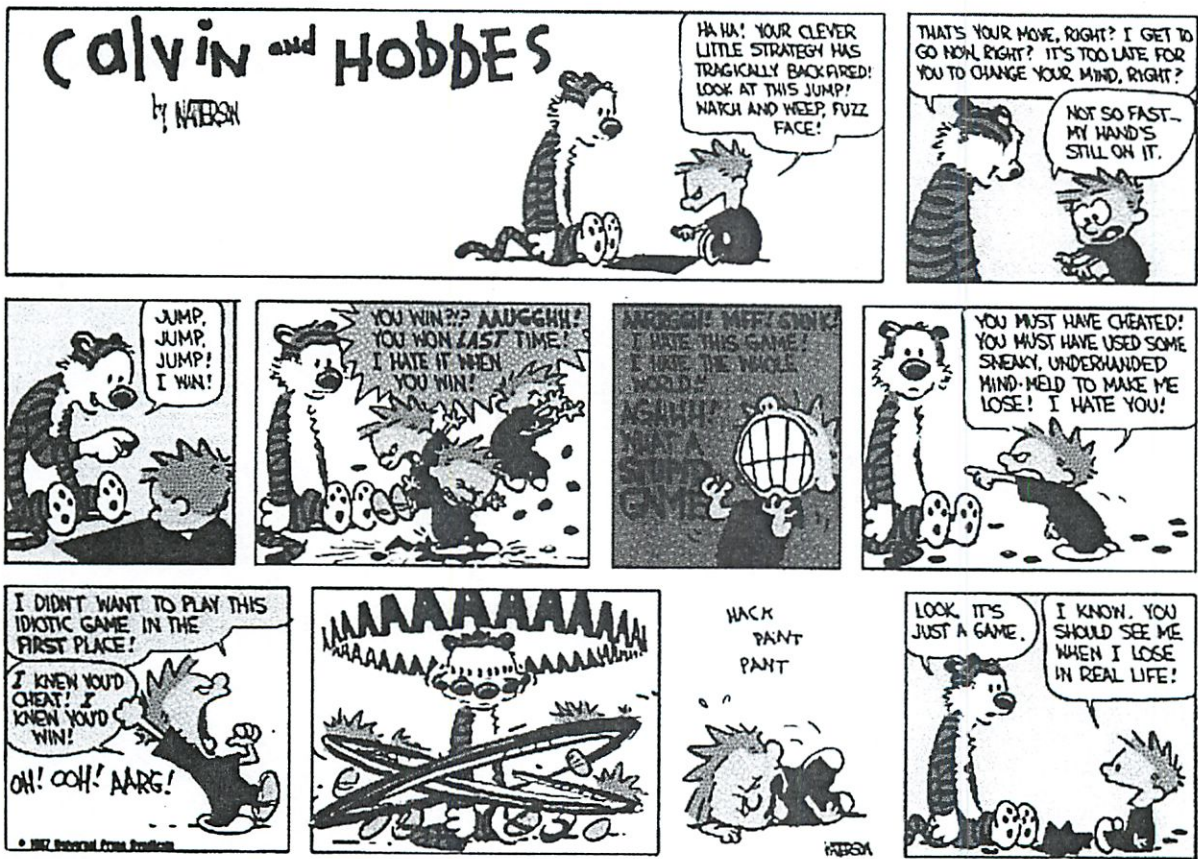
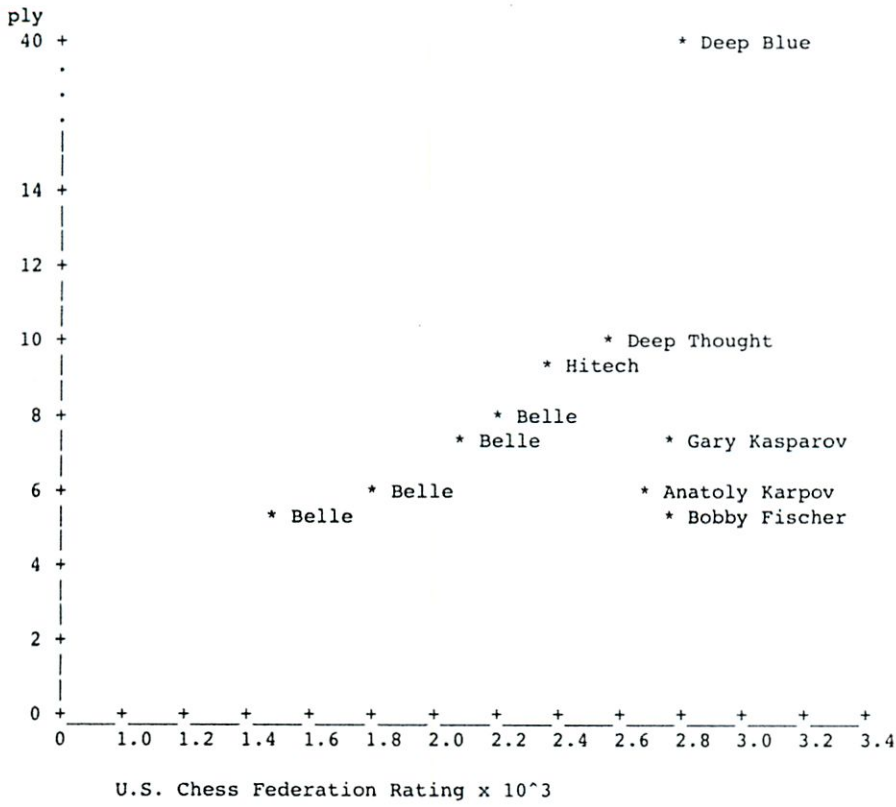
Note that alpha-beta is basically depth-first search. So, apart from alpha-beta, a second method that is commonly used in game tree searching is called *progressive deepening*, which is a way of embedding this depth-first search into a breadth-first framework. (Otherwise, we might run out of time diving far down into the tree and not have come up with a good sequence of moves to any level!) The way to do this is to use the notion of *iterative deepening*: first set $depth = 1$ and solve this tree by alpha-beta. This

gives a *principal variation* set of moves, down to this fixed depth. Then we increment the *depth* by 1, repeating the alpha-beta search *starting from the line of play already established*, and so on, until one runs out of time, making sure that the move we first search on the next iteration at *depth+1* is the best move we got from the search at the previous, shallower *depth*. Much more could be said about all of this – see, e.g., Adriaan de Groot, 1965, *Thought and Choice in Chess*. Mouton & Co Publishers, The Hague, The Netherlands. Second edition 1978. ISBN 90-279-7914-6, or, more recently, Chess Metaphors: Artificial Intelligence and the Human Mind by Diego Rasskin-Gutman, MIT Press.

Some ‘state space’ sizes for game trees. Note the huge difference between checkers, chess, and Go. Reference: H. Jaap van den Herik, Jos W.H.M. Uiterwijk, Jack van Rijswijck, Games solved: Now and in the future, *Artificial Intelligence* 134 (2002) 277–311

Table 6
State-space complexities and game-tree complexities of various games

Id.	Game	State-space compl.	Game-tree compl.	Reference
1	Awari	10^{12}	10^{32}	[3.7]
2	Checkers	10^{21}	10^{31}	[7.94]
3	Chess	10^{46}	10^{123}	[7.29]
4	Chinese Chess	10^{48}	10^{150}	[7.113]
5	Connect-Four	10^{14}	10^{21}	[2.7]
6	Dakon-6	10^{15}	10^{33}	[62]
7	Domineering (8×8)	10^{15}	10^{27}	[20]
8	Draughts	10^{30}	10^{54}	[7]
9	Go (19×19)	10^{172}	10^{360}	[7]
10	Go-Moku (15×15)	10^{105}	10^{70}	[7]
11	Hex (11×11)	10^{57}	10^{98}	[90]
12	Kalah(6,4)	10^{13}	10^{18}	[62]
13	Nine Men's Morris	10^{10}	10^{50}	[7.44]
14	Othello	10^{28}	10^{58}	[7]
15	Pentominoes	10^{12}	10^{18}	[85]
16	Qubic	10^{30}	10^{34}	[7]
17	Renju (15×15)	10^{105}	10^{70}	[7]
18	Shogi	10^{71}	10^{226}	[76]



And finally, a quote to read and think about:

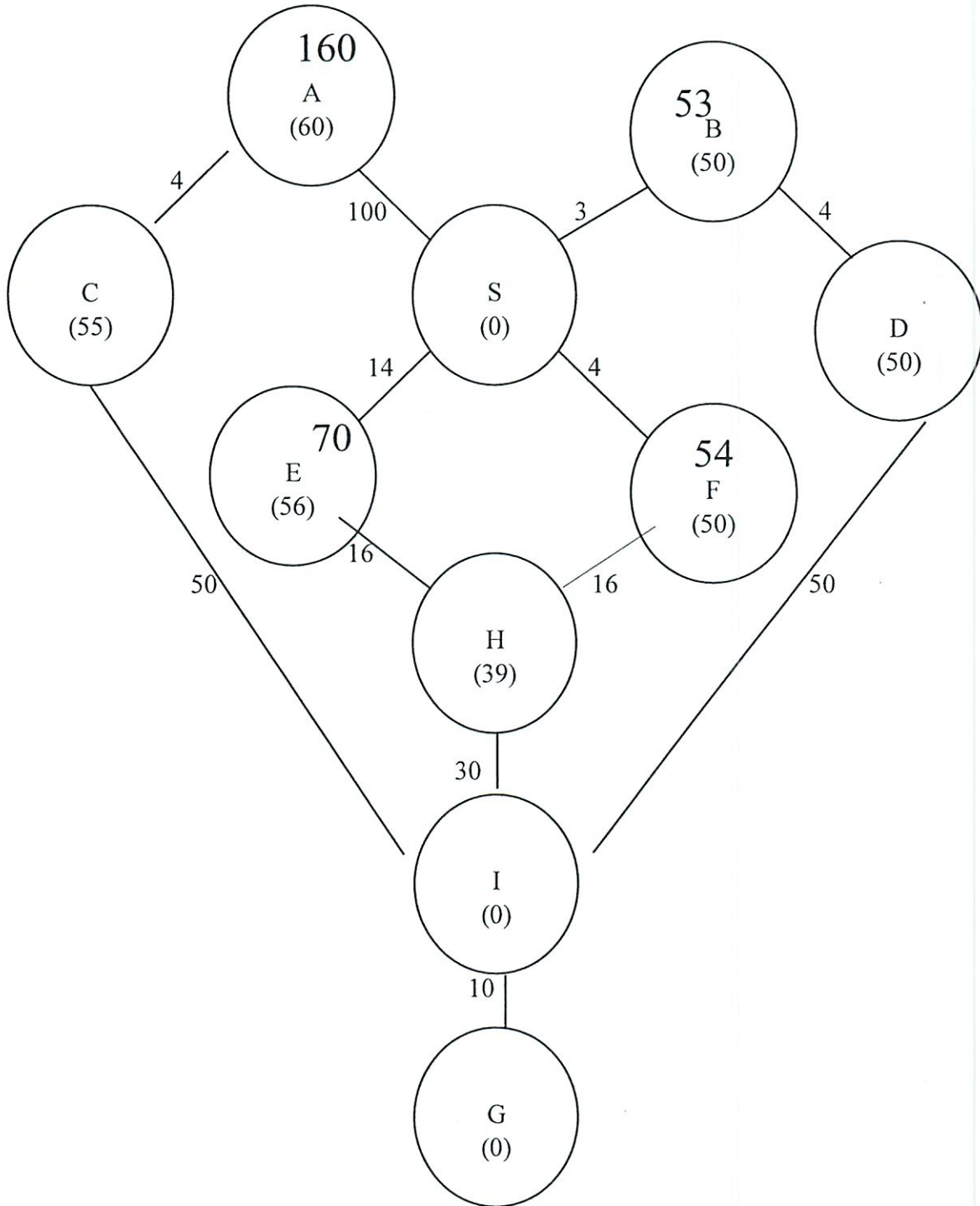
“With the supremacy of the chess machines now apparent and the contest of “Man vs. Machine” a thing of the past, perhaps it is time to return to the goals that made computer chess so attractive to many of the finest minds of the twentieth century. Playing better chess was a problem they wanted to solve, yes, and it has been solved. But there were other goals as well: to develop a program that played chess by thinking like a human, perhaps even by learning the game as a human does. Surely this would be a far more fruitful avenue of investigation than creating, as we are doing, ever-faster algorithms to run on ever-faster hardware.

“This is our last chess metaphor, then—a metaphor for how we have discarded innovation and creativity in exchange for a steady supply of marketable products. The dreams of creating an artificial intelligence that would engage in an ancient game symbolic of human thought have been abandoned. Instead, every year we have new chess programs, and new versions of old ones, that are all based on the same basic programming concepts for picking a move by searching through millions of possibilities that were developed in the 1960s and 1970s.

Like so much else in our technology-rich and innovation-poor modern world, chess computing has fallen prey to incrementalism and the demands of the market. Brute-force programs play the best chess, so why bother with anything else? Why waste time and money experimenting with new and innovative ideas when we already know what works? Such thinking should horrify anyone worthy of the name of scientist, but it seems, tragically, to be the norm. Our best minds have gone into financial engineering instead of real engineering, with catastrophic results for both sectors.” – Gary Kasparov, 2010 *New York Review of Books*.

Optimal Search

Now that Mark has his new stronghold, he wants to invade parallel universes. So Mark programs his evil supercomputer to find the shortest path of jumps from his starting universe S to his goal universe G.



Part B1 Branch & Bound search

First, Mark programs a simple **branch-and-bound search with an extended list**. As usual, he breaks ties of equal length in lexicographic order. List the nodes Mark's computer adds to the extended list, in order. Distances are shown next to edges. Ignore the numbers in parentheses for this part of the problem. Extra space is provided below in case you want to show your work.

Remember: for B&B, $f(n)=g(n)$ (total path length so far), and we sort the agenda by total path length.

(3 S B)(4 S F)(14 S E)(100 S A)

(4 S F)(7 S B D)(14 S E)(100 S A)

etc.

The answer is:

S B F D E H I G

The path found is:

S F H I G with cost 60

What path does Mark's computer find?

Part B2

Frustrated by branch-and-bound's speed, Mark reprograms his computer to use A^* . Mark counts the number of subspace anomalies between each universe and the goal and uses this count as the **heuristic** for A^* (**these are the numbers in parentheses**). List the nodes Mark's computer adds to the extended list, in order. Extra space is provided below in case you want to show your work. REMEMBER: For A^* , $f(n)=g(n)+h(n)$ = total path length + heuristic value at that node.

We have done the calculation of f for the first nodes away from S on the first page.

What path does Mark's computer find now?

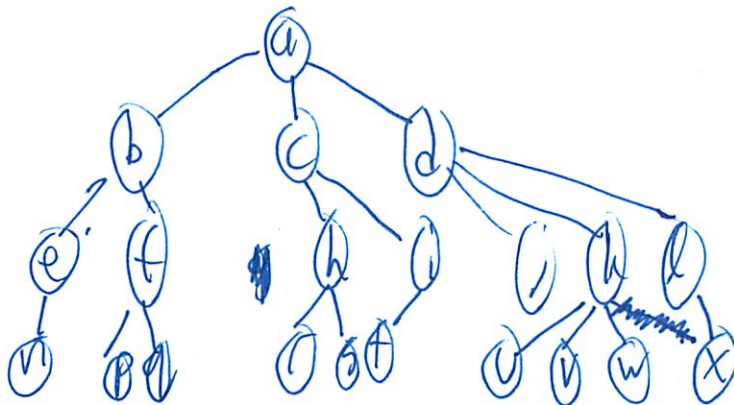
Part B3

Mark is confused. Give a **brief** but **specific** explanation of what happened and why.

Silver Star Idea

- Nuclear Options α and β
 - people have trouble visualizing
 - α is abs worst case max α
 - if maximizer gets less than α , just take α
 - Starts at $-\infty$ ie ~~max~~ will lose
 - then we see a way to win
 - updates α
 - Snow White
 - Deep Cutoff \approx Cake
 - Static is Expensive!
-

People voted for hard example



← static where maximizer does not want to continue

②

On first leg of tree, never prune anything

Snow White principle: Can take beauty from parents
or steal from children (???)

① Can't take value till eval ⑨

First value

$$\textcircled{e} \alpha = n$$

Then ① - sees $\alpha = n$, has $\beta = +\infty$
put $\beta = n$ since better
but still need to eval down
So $\beta \leq n$

What is α ⑥ = know nothing from children'
So from parents $\alpha = -\infty$

Cutoff when
 $\alpha \geq \beta$

Need to at least look at (p)

Then at (f) $\alpha \geq p$

Want to do cutoff, so might not want to look at (q)

But $p \geq n$, so α at (f) $>$ β at (b)

So minimizer cuts off (f), w/o evaluating (q)

(q) is completely irrelevant, whatever it is

Now set (b) $\beta = n$

Now look up (a) $d = n$

Now (c) β can only inherit higher from parent higher than you
 $\beta \leq \infty$

Need to look at (r) - not enough info to cutoff after looking
- maximizer is greedy, wants better solution

If $\max(\text{node } (r), (s)) < n$

So now (h) $d < n$

Then (c) $\beta \leq n$

4

But $\alpha \geq \beta$

So completely prove (c) - w/o looking (i) (t)

(d) β is $-\infty$, no parent to inherit from

Look at (j) $j < n$

So $\beta \leq j$

Now $\beta < \alpha$ so cut off (d)

In lab - players of different levels

In some games could *build* strategies where

assume opponent is stupid

- but usually, bad strategy

Another Question: Progressive Deepening

↑ do each level one at a time

- need to trust static evaluator

- Costs a bit more to do more levels

- but always return a value

5

Are you guaranteed to do at least as well?

- it can make a few errors - possible
- but on average it helps you

When you calculate it for real on PC its the
status eval that is a pain in the ass

- for us on the test its the α, β thats hard

If $\cdot x$ by a \oplus # - all nodes

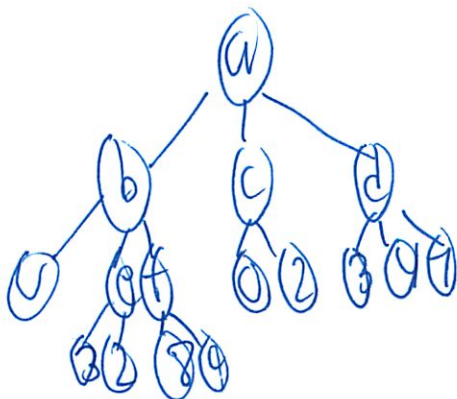
- No change

If x by \ominus # - ~~the~~

- that is reverse. took good nodes + made bad

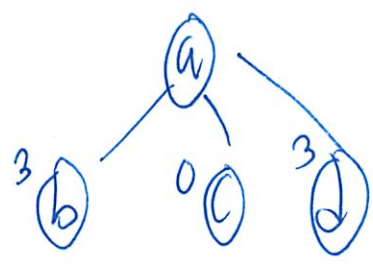
Evil progressive deepening (2004 quiz)

- Sabotage the order so screws stuff up

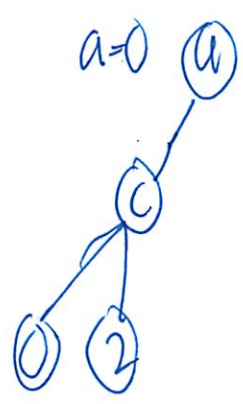


6

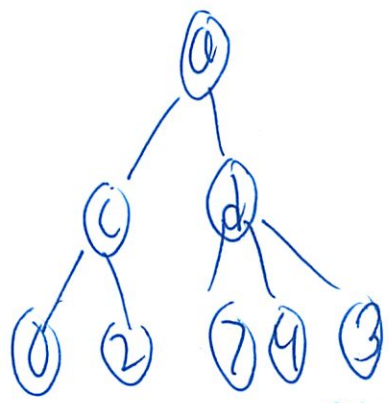
So



So new tree



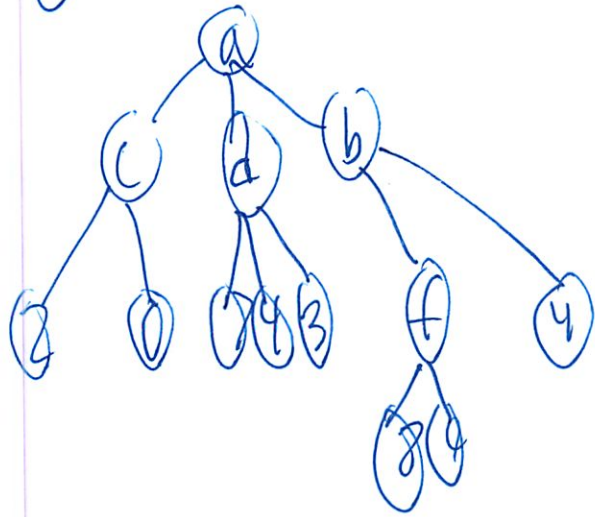
Can do b, d in any order



Does not matter $B=3$
make minimizer work for this

Then for (b) put (f) first since it is worse

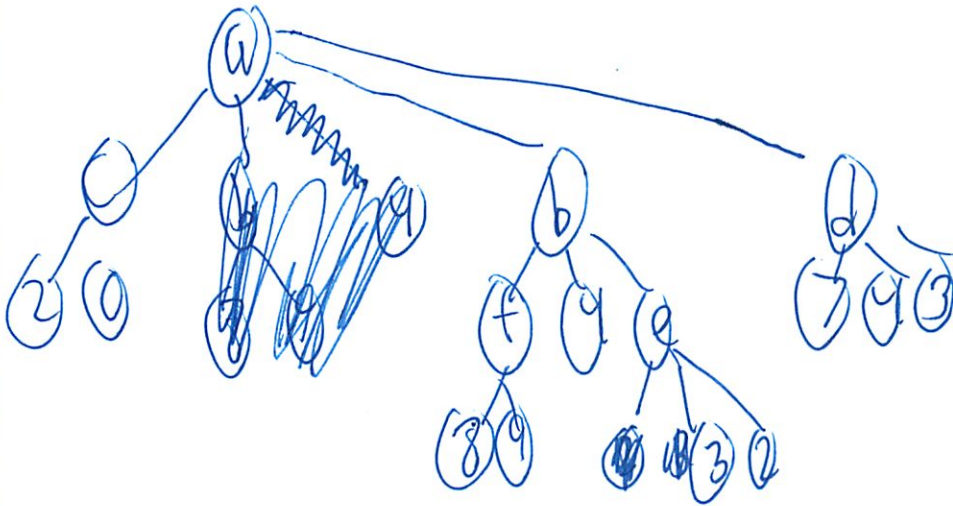
7



Again, trying put in worst order

we minimize's appetite

Mistake: b must come before d



Or could do it if switch 2, 3

□ Let C be a constraint set

□ Guzman \rightarrow Huffman \rightarrow Waltz

Gold star ideas

* Criteria for success

Problem

~~Problem~~ Program

Principle

* More description

\rightarrow More constraint

\rightarrow More speed

* Intellectual Diversity Good

This might be most important lecture of semester

Most classes teach 100+ year old stuff

Here you can hear how the figured stuff out.

- Human visuals

- How to schedule w/ lots of resource constraints

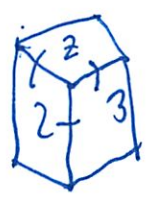
Showing those visual puzzles

- ambiguity

- does computer have same problem - yeah

2

A drawing has junctions
A world has vertexes



Needed some abstract principle

It worked a priori → Same object 

Works w/ abduction

deduction



deduction
many to
few



induction -
few to
many



abduction -
Usually true,
so assume will be true

~~Other~~ ^{Huffman} ~~person~~ abstracted problem

1. Three Faces

~~Tri~~ Tri - Medial Vertexes

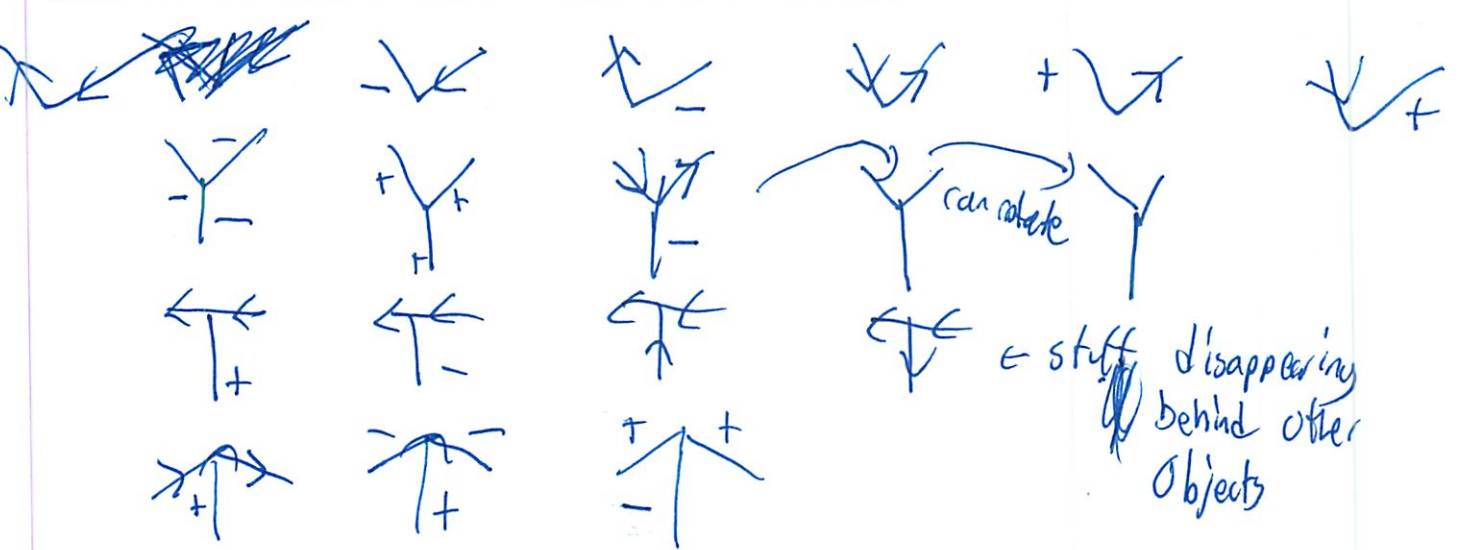
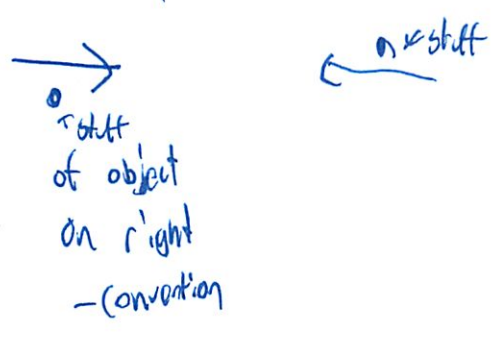
- 3 face
- what usually happens w/ cubes
- rules some stuff out - like pyramids

2. General position

- can rotate any 3D objects

③ - Can you rotate w/o changing it

3. Assume can see convex edge or concave edge
 Or boundary -table + -table -



Only drew 18, ~~was~~ but may be 100s

Looking at objects, what do we see?

I didn't do ~~color~~ colors

Considered all the possibilities

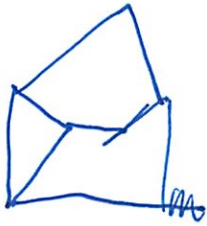
2, 4, 6 octaves are not tri-hedral

1, 3, 5, 7 considered

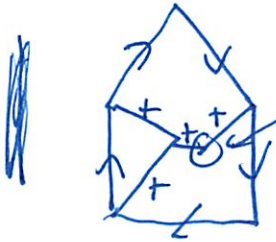
4

Want every possibility
- British museum - like

Try some examples

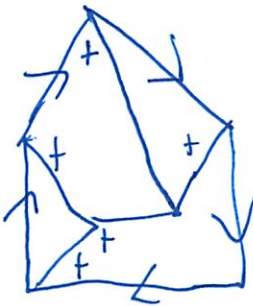


Does not look like something you
can make



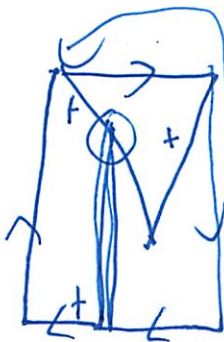
No such L in this uni
So can not exist in world
we defined

Could do?



No

How about



No - but can be made
Since more than 3 faces

are 4 faces here (one behind)

5

Huffman's solution not a program

- and in real life you can have more than 3 faces

Waltz tried to be more complete in theoretical

Goes to 11 interpretations

- shadows - up from Υ + - \Rightarrow \Leftarrow

- cracks \uparrow \downarrow

\overline{c} \overleftarrow{c} \overrightarrow{c} \overline{c}

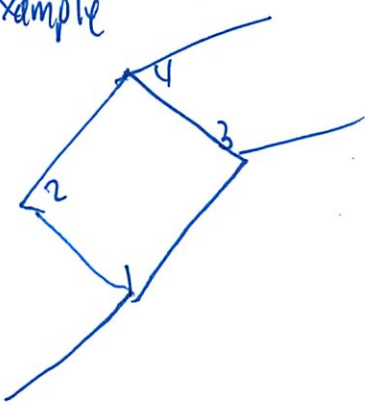
- lighting, so 99 total

- then down to 50

10,000s of junction types

- by looking at blocks

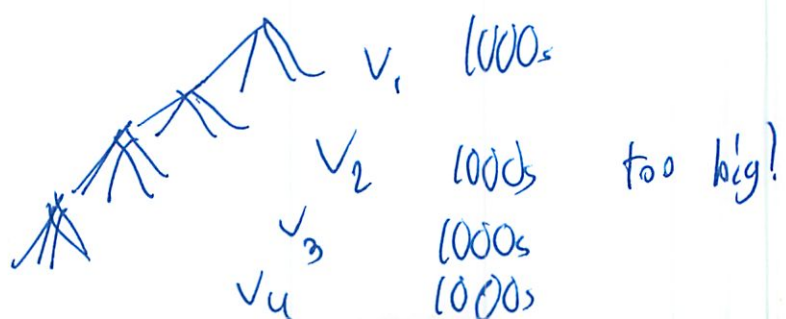
Example



How to label/interpret:

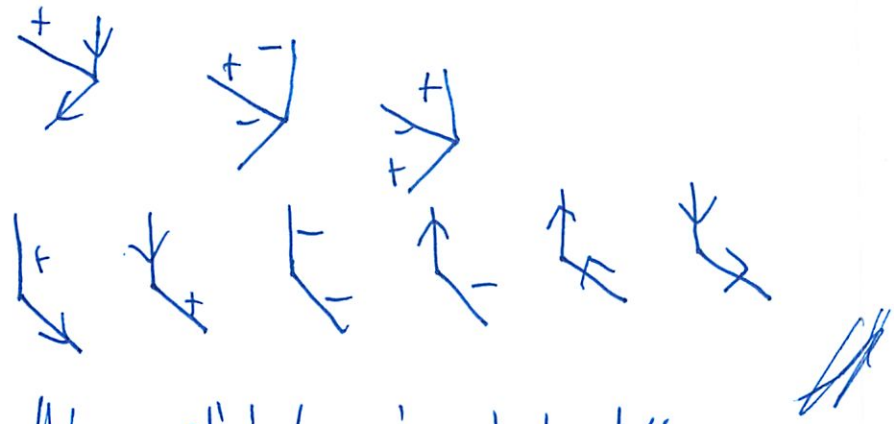
- so don't duplicate

Can do tree



6

Needs to invent new type of search



Now eliminate inconstant stuff

Propagate Constraints from 1st group
 - can eliminate 1, 3, 5, 6

Now another group



- Eliminate 1, 2 from this

Can propagate down

- eliminate 2

Propagate to first

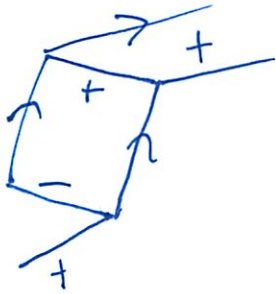
- eliminate 1, 2

So ~~one more~~ group now one in each group

(7)

So

Runs in n^2
↑
Junctions



Line labling is NP complete

Can have others w/ multiple interpretations

- does not always converge
- So P vs NP not solved!

Ambiguous

Is it on a table?

Stuck against a wall?

~~Experiment~~ Experiment

Problem	G	H	W	← One guy
Program	G		W	
Principle		M	W	

- Went to different one at someone's suggestion
Erek

PS2 review

- knowing the difference
- beam search had most changes

PS3

- Connect 4
- Hardest part of lab is knowing Connect 4 is good
- Hardest lab of year
- Half people don't get 5
- Need to build good static evaluator
 - know how to win Connect 4 yourself
 - Pick those things out
- Static Evaluator
 - accurate
 - fast
- Often 2x better to go 1 more ply than better SE

②
Chess: think ~~more~~ ahead more than 1 move

Have a good representation of moves

- here everyone same

Hint: know chances to win

Or # of threats

- fast enough?

Connect 4 Wikipedia pg

- link to Masters thesis

- that red will always win

- but must play perfectly

- Or another shorter article

Now how to implement α, β search

- here always maximize

- but also negate + switch values

Make a helper function

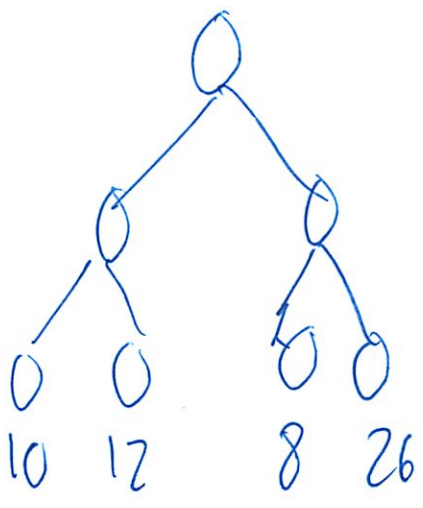
- takes in α, β as well

\downarrow \downarrow
- β - α as go up tree

③

α, β is 2nd hardest part of lab

Doing this switch simplification to make code simple



α = lower bound
 β = upper bound

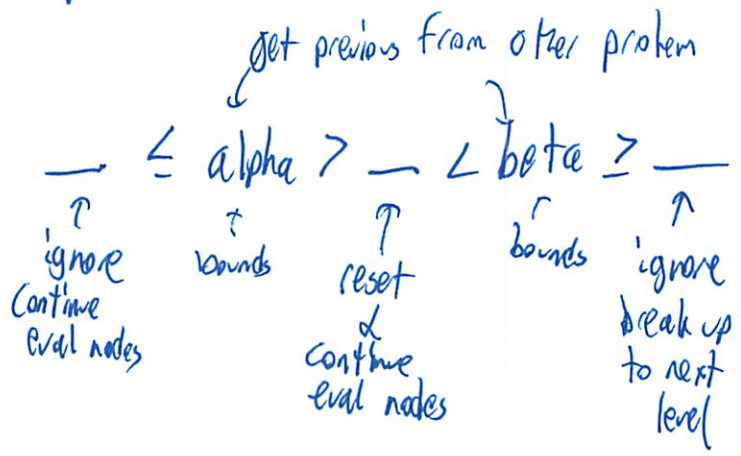
?

becomes lower bound

but our simplification ... -10

Compare it to other things

- $\leq \alpha$
- $> \alpha$
- $< \beta$
- $\geq \beta$



If $< \alpha$, ignore, continue evaluating nodes

So we need to be ≥ -10

Then compare w/ -12

-ignore, continue

4

So now say node = -10

Now at root node set $\alpha = 10$
negate

- can at least as well as 10
- β is still ∞ - we have made no upper bound

2ND level right

- propagate α, β down
- Call fn ($-\infty, -10$)
 α β

At γ , since $\gamma > \beta$ so we break/return

What do we return?

- a tuple ($\#, \text{move}$)
- or
- take β , return it whatever it is

~~to α~~ up to next level
- then this will propagate up

we Pruned 26

- since best min can do is -10
- if $-\gamma$, min will want to do
- But Max won't let so $\beta = 10$

5

So if violate β , then α would go other way

This will be hard if don't see intuition behind this method

Previous Game

0	0	0	0

Goal - get \downarrow _{white} person to other side

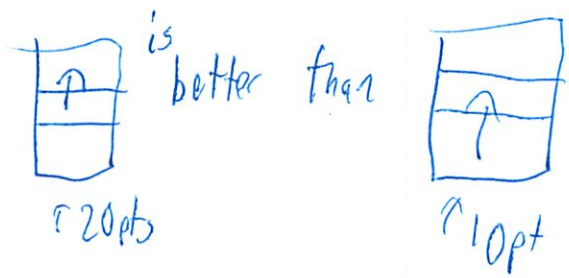
Static evaluator: possibilities

1. How far up ~~or down~~?

UP = 10pt each

Or have a multiple for 2nd row

- since



- Work out constants

- Shortcut: ~~use~~ use exponent $(column)^2$

Correct 4: use $(chains)^2$

6

Could count moves

White has 4 possible moves

(Bad in CF4 - always $>$ possible moves)

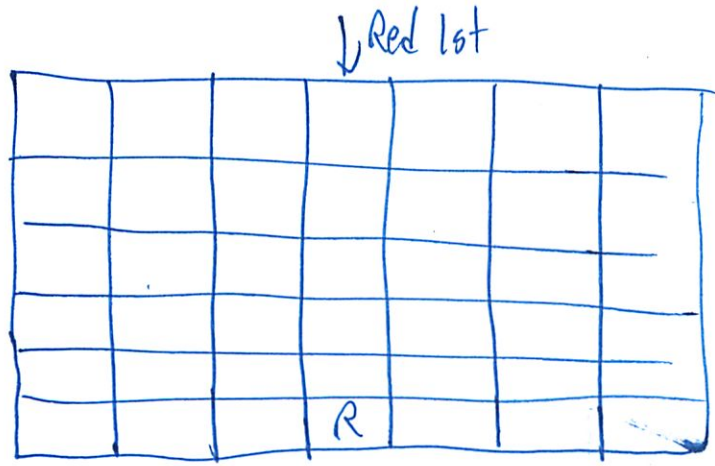
Look at basic player AI

- puts pieces in middle
- Checks if won or loss
- ↑ draws considered losses

* - would want to weight diff

- just looks at longest chain

- does not even aware it



↑ think about odd rows - somehow special

- Want a lot of threats

if

x x

x x odd

⑦

Then player shall be able to win

If 3 odd minor threats - would win

Threat - minor 2 away from win) for you
major 1 away from win

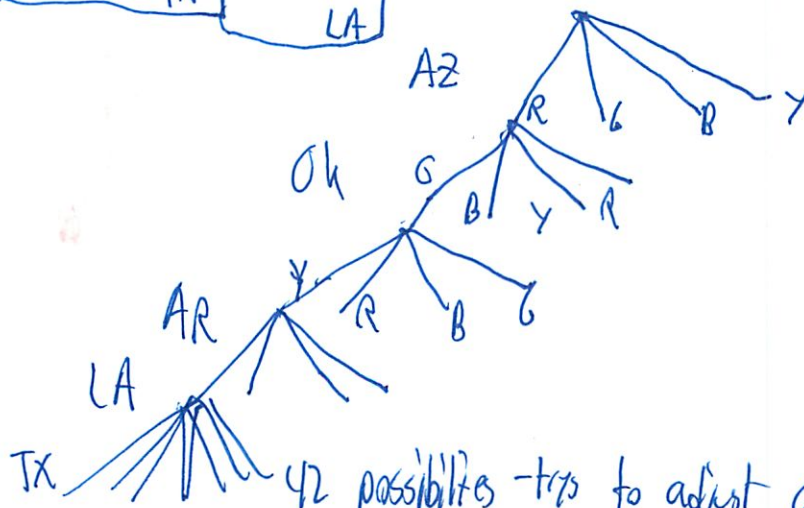
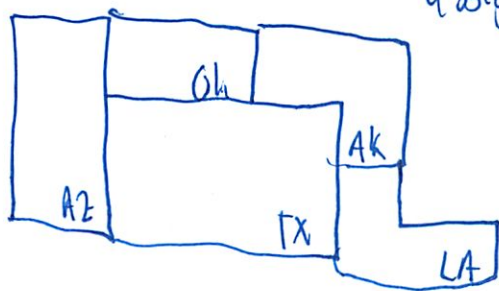
↳ ~~the~~ don't care whose move it is
 T X X X

Lecture

- The trouble with Texas
- Domain Reduction Algorithm
- Propagation Options
- Ordering Options
- Resources
- Anytime Schedule
- * It pays to work together
- * It pays to organize the work
- * Anytime algorithm

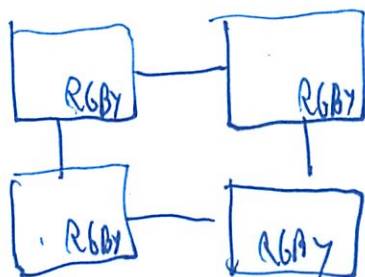
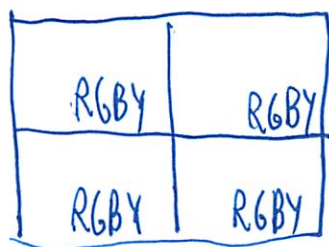
Some searching algorithm (missed description)

Assigning colors to states



②

Constraint propagation



No place to get started

This coloring problem is like any scheduling program

He knows Texas is the problem - 4 states touch

So keep track of what colors Texas is

Can see problem before going down 42 levels of search

When goes to 0, backup

Called domain-reduction algorithm

Terms

Variable V can have assignments

Values x, y can be assignments

Domain D bag of values

Constraint C limit values in pair of domains
typical

3

So the algorithm is
- combines DFS w/ constraint propagation

For each DFS assignment

For each variable V we considered

For each value X in D

For each constraint $C(x, Y)$

If $\nexists y$ such that $C(x, y)$ is satisfied

Then remove X from D
reduces # items in domain

If D empty, backup

← here Texas

← what does this mean
Just look at neighbors

↑ other domain

All the ways to consider.

Constraints checked

1. No check

2. Assignments

3. Neighbors only

u. Domains reduced to 2 value

5. Neighbors of neighbors when something does not work

(Propage through neighbors w/ reduced domain)

6. Check everything

480 ← best

2623

(4)

But best depends on how assigned

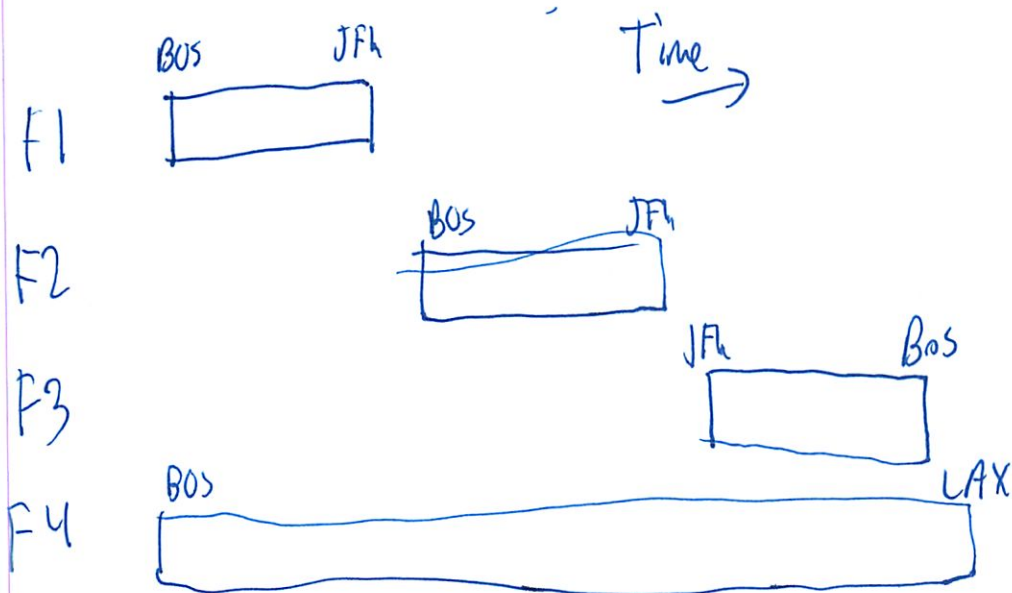
Picking Arizona lot was carefully picked to show it down
What if worked alphabetically?

Most constrained lot: ← far better

Least constrained lot:

Can work w/o constraint checking
gold star idea #2

Airline scheduling



Constraint: No 2 flights at same time

Can do map coloring - each plane is a color

5

Heuristic: Draw a line straight down

The most boxes you cross are min planes you need

Most constrained is often best

The fewer the resources, the harder it is

Can keep running a plane algorithm till it gives up
With 1 less resource each time

Find min # of resources

Time to search gets longer in the middle

Some intelligence in here!

How about A*?

We just followed a recipe

Which we were able to learn

Just like we can learn stories

1. Thinking about α, β cut off

2. Constraints - 3 types of constraint propagation

- Back tracking (BT)

- BT w/ forward checking (FC) through singleton domains

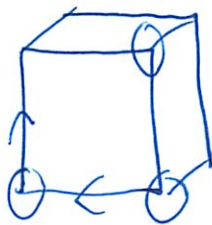
- BT w/ FC through reduced domains

- waltz labeling

- AC-2

- arc consistency 2

Domain



Junctions \equiv Variables
(Domains)

\swarrow Junction Labels
(Values)

Will review last lecture later.

②

But first another look at d, β

- let me see if I can do it

- ~~in~~ in the packet

- He is confusing me by writing $d \geq \dots$

before wrote $d = \dots$

the $>$ is implicit

Constraint propagation

- giving talks

- Constraints

(I remember doing these manually)

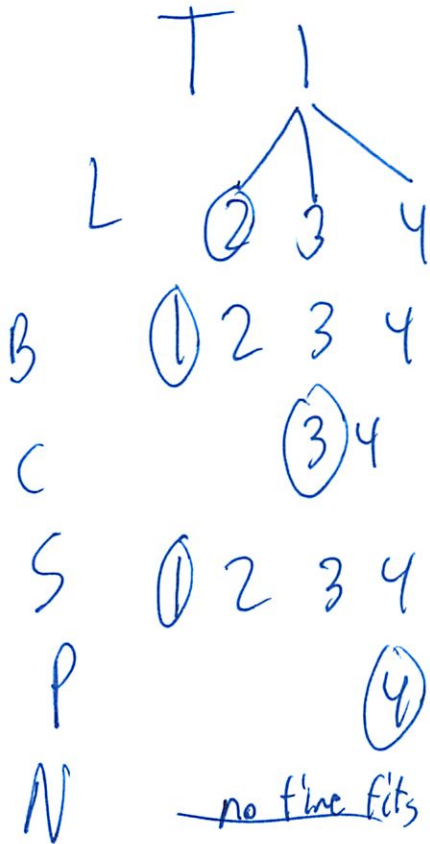
Think

Pure back tracking - really slow.

Just try to fill in
if problem, back track up tree

③

Turing is fixed
- put him in



Can see conflicts on
the graph/web/diagram
Then kinda fill it in

← can only be 4

None work so back track to first free choice

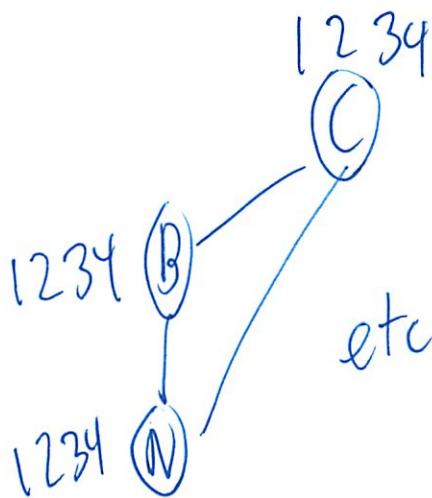
So to S



And try again

4

Now forward checking w/ single for domains
 ↙ When I item left, propagate it
 ↘ check
 L checking neighbors
 So write possible times next to each bubble



So T gets 1

Then circle 1 for T

And for all people T is connected to, cross out

So N is ~~2 3 4~~

Then next one, L gets 2

- Then cross all his neighbor's 2s out

Next B - give him 1 since all free

Cross out C, N 2s

- N already crossed out - ok, ignore

⑤

Now C. Give it 3

But now N has X & X & 4

So N only has 1 left

- could assign 4 (optional)

- but cross all N's neighbors are 4's out

~~BT~~ Now P has only 1 left 1 X & X

So. ~~pro~~ optionally assign and propagate

If that was not available then we would have to back track like before.

Now boxes on last pg handout

BT would be horrible

Worst possible example

4¹⁵ at least

FC w/ Singleton gets it instantly

(6)

AC-2 looks at pairs of junctions
- not all colors in map

May still need to backup

If start at most constrained it is easier
- connected to most other regions

- will solve pg 2 example w/o backtracking

Takes computation to maintain Acc consistency

Nothing is sure fire

- Depends on topology

Constraint satisfaction algorithms that we have learned about:

1. **DFS with Backtracking (BT).** No constraint checks are propagated ["Assignments only"]. One variable is assigned a single value at a time, and then the next variable, and the next, etc. Check to see if the current partial solution violates any constraint. Whenever this check produces a zero-value domain, backup occurs.
2. **DFS with Backtracking + Forward-checking.** A variable is assigned a unique value, and then *only its neighbors* are checked to see if their values are consistent with this assignment. That is: assume the current partial assignment, apply constraints and reduce domain of neighboring variables. ["Check neighbors of current variable only"].
3. **DFS with Backtracking + Forward-checking + propagation through singleton domains.** If during forward checking you reduce a domain to size 1 (singleton domain), then assume the assignment of the singleton domain and repeat forward checking from this variable. (Note that reduction to a singleton domain and repeated forward checking might lead to yet another singleton domain in some cases.)
4. **DFS with Backtracking + Forward checking + propagation through reduced domains (this is also known as: Arc-consistency, AC-2, since it ensures that all pairs of variables have consistent values).** Constraints are propagated through all **reduced domains** of variable values, possibly beyond immediate neighbors of current variable, until closure. ["Propagate through reduced domains"; also called "Waltz labeling"]

1. Map coloring – comparing Backtracking (BT) with Forward-checking+propagation through singleton domains.

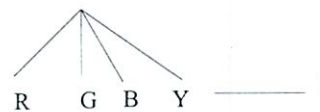
(a) How long will it take to color (two adjacent states cannot have the same color) the following map with 4 colors with just Backtracking (BT)?

19																	
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18

Assume that:

- Color the states **in the order they are numbered.**
- Use R(ed), G(reen), B(lue) and Y(ellow) in **the given order & then in rotation**, starting with R; i.e., when you finish assigning Y, start again with R
- Assume that coloring a single location takes 1/100 second

Draw the shape of the search tree for BT: (we have started it for you):

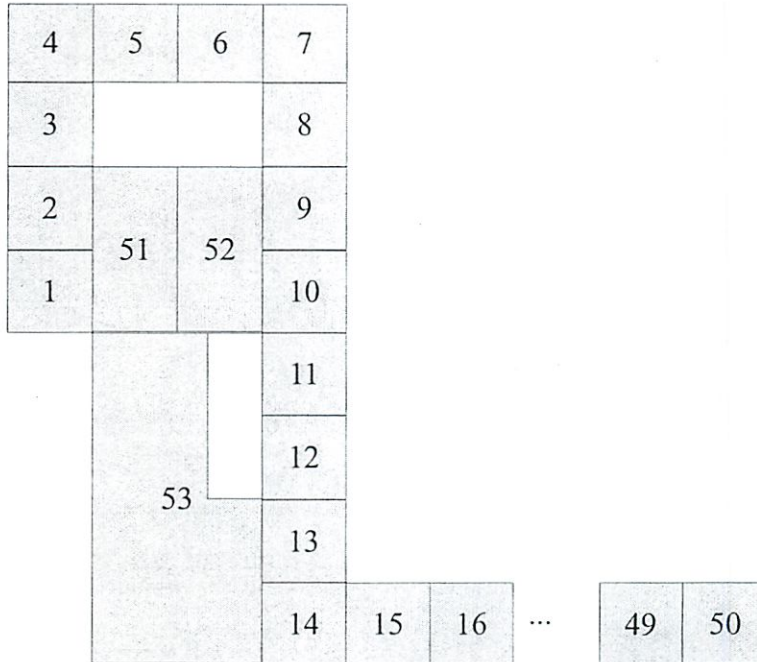


BT will therefore take _____ seconds

(b) How many seconds will **Forward checking+propagation through singleton domains** take on the *same* map?

FC+prop through singletons will take _____ seconds.

(c) Can **FC+propagation through singletons** finish the map below quickly? Why or why not? What about AC-2? (Propagation through reduced domains.)



Suppose we proceeded not from square #1 to #53 but from #53 to #1. Would our result now change? What principle of efficient constraint propagation does this illustrate? Are there any other principles of efficient constraint propagation that you can think of?

2. Converting problems into constraint propagation form

“Paul, John, and Ringo are musicians. One of them plays bass, another plays guitar, and the third plays drums. As it happens, one of them is afraid of things associated with the number 13, another is afraid of Apple Computers, and the third is afraid of heights. Paul and the guitar player skydive; John and the bass player enjoy Apple Computers; and the drummer lives in an open penthouse apartment 13 on the thirteenth floor. What instrument does Paul play?”

How can we solve this problem? Try it yourself first, by any means you care, then we’ll see how to do it by – ta-da! – the magic of constraint propagation! (You might want to think about constraints when you solve it, and what the constraints are.)

What are the constraints? How might they be represented? We want to use the facts in the story to determine whether certain identity relations hold or are eXcluded. Here is our notation: assume $X(\text{Peter, Guitar Player})$ means “the person who is John is not the person who plays the guitar.” Further, this relation is symmetrical, so that if we have $X(\text{Peter, Guitar Player})$ then we also have, $X(\text{Guitar Player, Peter})$. In this notation, the facts become the following (of course all the symmetrical facts hold as well):

1. $X(\text{Paul, Guitar Player})$
2. $X(\text{Paul, Fears Heights})$
3. $X(\text{Guitar Player, Fears Heights})$
4. $X(\text{John, Fears Apple Computers})$
5. $X(\text{John, Bass Player})$
6. $X(\text{Bass Player, Fears Apple Computers})$
7. $X(\text{Drummer, Fears 13})$
8. $X(\text{Drummer, Fears Heights})$

Now we can represent the possible relations implicitly by means of entries in tables, and use constraint propagation. An X entry in a table denotes that the identity relation is excluded, and an I denotes that the identity relation actually holds. We can then use three tables, one to represent the possible identities between people and instrument players; one to represent the possible identities between people and fears; and a third to represent the possible relationships between instrument players and fears.

1. $X(\text{Paul}, \text{Guitar Player})$
2. $X(\text{Paul}, \text{Fears Heights})$
3. $X(\text{Guitar Player}, \text{Fears Heights})$
4. $X(\text{John}, \text{Fears Apple Computers})$
5. $X(\text{John}, \text{Bass Player})$
6. $X(\text{Bass Player}, \text{Fears Apple Computers})$
7. $X(\text{Drummer}, \text{Fears 13})$
8. $X(\text{Drummer}, \text{Fears Heights})$

People	Instrument Player		
	Bass Player	Guitar Player	Drum Player
Paul			
John			
Ringo			

People	Fears		
	13	Apple Computers	Heights
Paul			
John			
Ringo			

Instrument Player	Fears		
	13	Apple Computers	Heights
Bass player			
Guitar player			
Drum Player			

How do we do constraint propagation in this system? Note that we can deduce rules like the following to fill in the three tables:

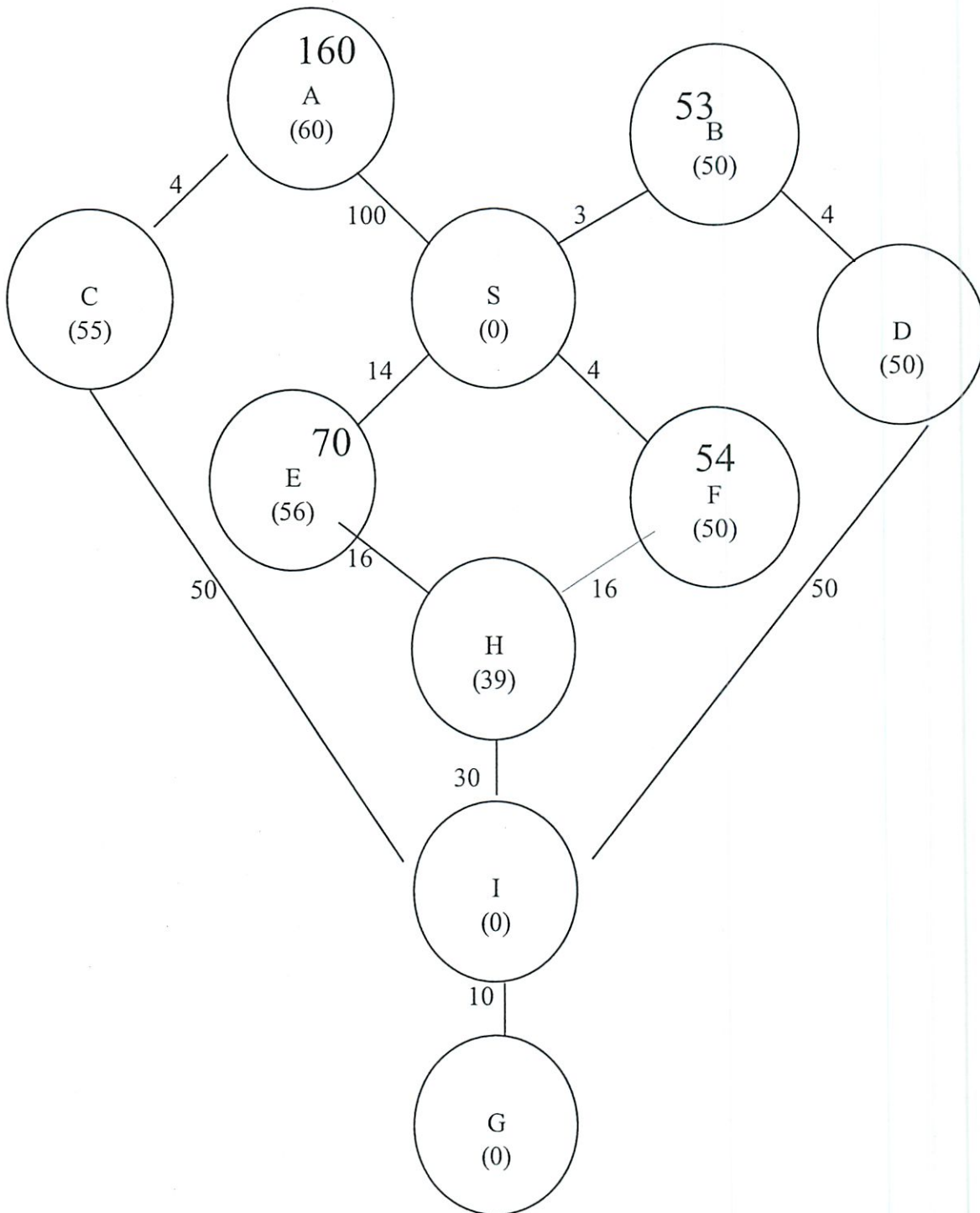
1. If all but one entry in a row are X , then the remaining entry is I .
2. If you know $I(x,y)$ and $X(y,z)$ then you may conclude $X(x,z)$.

Question: what property of the world does this last constraint rule capture?

What other rules do we need?

Optimal Search

Now that Mark has his new stronghold, he wants to invade parallel universes. So Mark programs his evil supercomputer to find the shortest path of jumps from his starting universe S to his goal universe G.



Part B1 Branch & Bound search

First, Mark programs a simple **branch-and-bound search with an extended list**. As usual, he breaks ties of equal length in lexicographic order. List the nodes Mark's computer adds to the extended list, in order. Distances are shown next to edges. Ignore the numbers in parentheses for this part of the problem. Extra space is provided below in case you want to show your work.

Remember: for B&B, $f(n)=g(n)$ (total path length so far), and we sort the agenda by total path length.

(3 S B)(4 S F)(14 S E)(100 S A)

(4 S F)(7 S B D)(14 S E)(100 S A)

etc.

The answer is:

S B F D E H I G

The path found is:

S F H I G with cost 60

What path does Mark's computer find?

Part B2

Frustrated by branch-and-bound's speed, Mark reprograms his computer to use A^* . Mark counts the number of subspace anomalies between each universe and the goal and uses this count as the **heuristic** for A^* (**these are the numbers in parentheses**). List the nodes Mark's computer adds to the extended list, in order. Extra space is provided below in case you want to show your work. REMEMBER: For A^* , $f(n)=g(n)+h(n)=$ total path length + heuristic value at that node.

We have done the calculation of f for the first nodes away from S on the first page.

Remember: We sort the agenda by total path length.

(53 S B)(54 S F)(70 S E)(160 S A)

(54 S F)(57 S B D)(70 S E)(160 S A)

(57 S B D)(59 S F H)(70 S E)(160 S A)

(57 S B D I)(59 S F H)(70 S E)(160 S A)

(59 S F H)(67 S B D I G)(70 S E)(160 S A)

~~(59 S F H I)~~(67 S B D I G)(70 S E)(160 S A) [Why???

(67 S B D I G)(70 S E)(160 S A)

Path: S B D I G, length= 67, which is non-optimal.

Compute the f values in F, H, I and let's see why this happens... remember, f values should be monotonically increasing on each path, e.g., S-F-H-I, are they?

What is the term for this?

What path does Mark's computer find now?

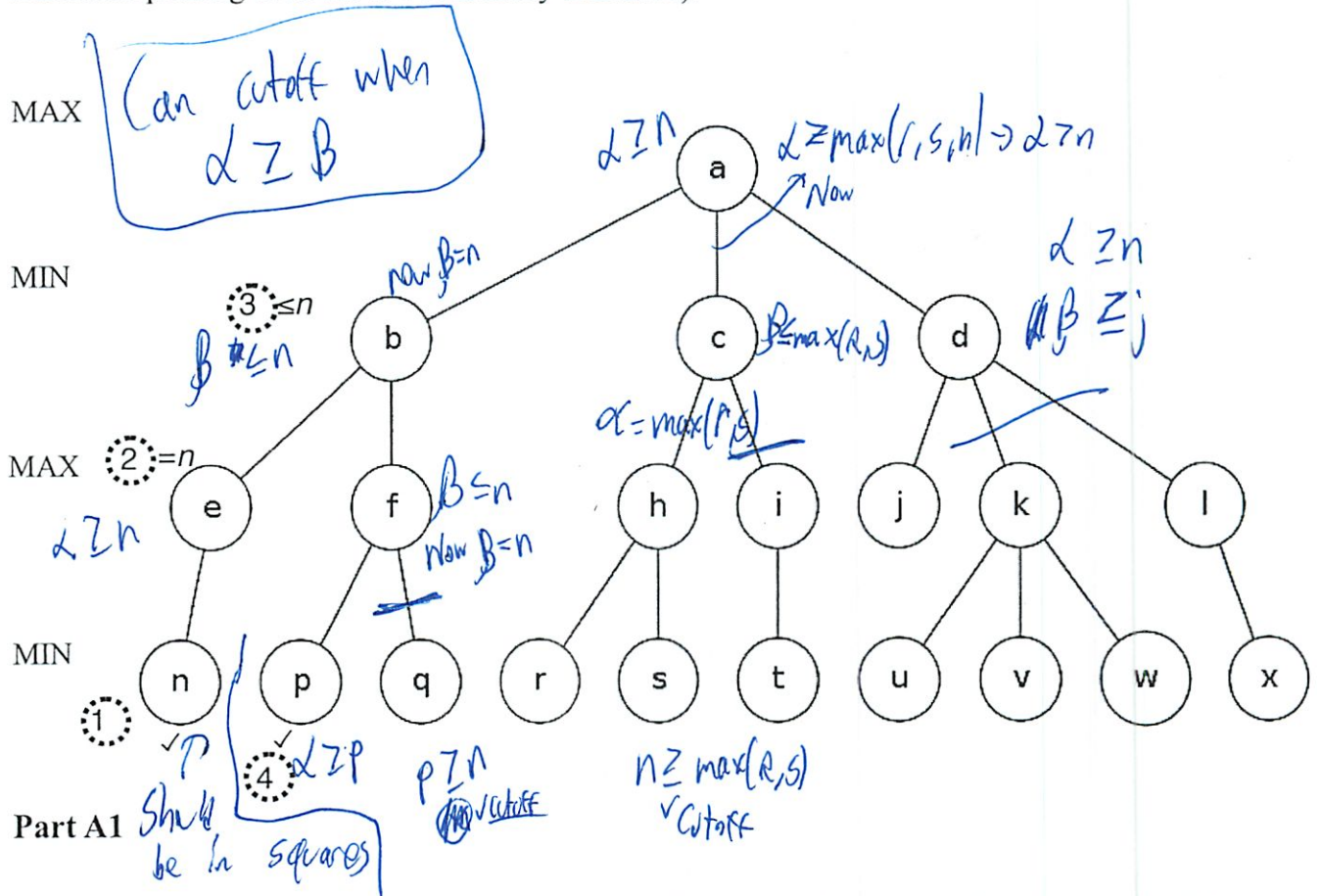
Part B3

Mark is confused. Give a **brief** but **specific** explanation of what happened and why.

Problem 1: Games

Part A: Working with a maximally pruned tree (25 points)

For the following min-max tree, cross out those leaf nodes for which alpha-beta search would **not do static evaluations** in the best case possible (minimum number of static evaluations, maximum pruning of nodes to be statically evaluated).



Now, list the leaf nodes at which alpha-beta would do static evaluations in the best case possible.

n p r s j

Part A2

What is the final value returned by the alpha beta search in the best case possible for the given tree? Express your answer as the simplest function of the static values of the leaf nodes (e.g. take n to be the static value at the leaf node labeled n). Your function may contain operations such as **max** and **min**.

Part A3

What constraints ensure best case possible (minimum static evaluation) for the given tree? State your constraints as inequalities on the static values of the leaf nodes.

Part A4

Suppose your static evaluation function, $S(\text{node})$, is modified as follows:

$$S'(\text{node}) = 42 \times S(\text{node}) + 1000. \quad (\text{If } S(\text{node}) = 1, S'(\text{node}) = 1042)$$

Would your answer for Part A1 be the same for all possible $S(\text{node})$ values?

Yes

No

didn't read wright!

Suppose your function were

$$S'(\text{node}) = -42 \times S(\text{node}) + 1000.$$

Would your answer for Part A1 be the same for all possible $S(\text{node})$ values?

Yes

No

Now inequalities flipped

would actually give no diff

Problem 2: Time Travelers' Convention

The MIT Time Travel Society (MITTTS) has invited seven famous historical figures to each give a lecture at the annual MITTTS convention, and you've been asked to create a schedule for them. Unfortunately, there are only four time slots available, and you discover that there are some restrictions on how you can schedule the lectures and keep all the convention attendees happy. For instance, physics students will be disappointed if you schedule Niels Bohr and Isaac Newton to speak during the same time slot, because those students were hoping to attend both of those lectures.

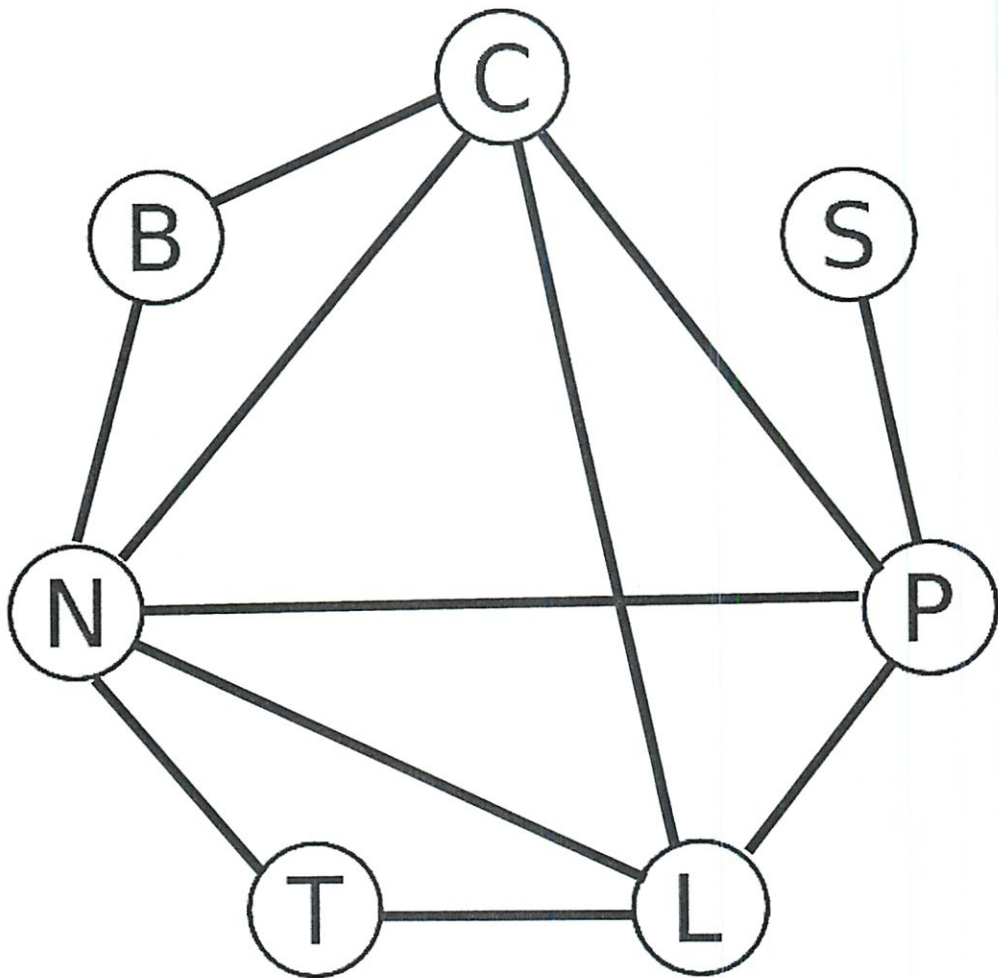
After talking to some students who are planning to attend this year's convention, you determine that they fall into certain groups, each of which wants to be able to see some subset of the time-traveling speakers. (Fortunately, each student identifies with at most one of the groups.) You write down everything you know:

The list of guest lecturers consists of Alan Turing, Ada Lovelace, Niels Bohr, Marie Curie, Socrates, Pythagoras, and Isaac Newton.

- 1) Turing has to get home early to help win World War II, so he can only be assigned to the 1pm slot.
- 2) The Course VIII students want to see the physicists: Bohr, Curie, and Newton.
- 3) The Course XVIII students want to see the mathematicians: Lovelace, Pythagoras, and Newton.
- 4) The members of the Ancient Greece Club wants to see the ancient Greeks: Socrates and Pythagoras.
- 5) The visiting Wellesley students want to see the female speakers: Lovelace and Curie.
- 6) The CME students want to see the British speakers: Turing, Lovelace, and Newton.
- 7) Finally, you decide that you will be happy if and only if you get to see both Curie and Pythagoras. (Yes, even if you belong to one or more of the groups above.)

Part A (5 points)

That's a lot of preferences to keep track of, so you decide to draw a diagram to help make sense of it all. Draw a line between the initials of each pair of guests who must not



Part B (15 points)

You decide to first assign the time slots (which conveniently happen to be 1, 2, 3, and 4 pm) by using a **depth-first search with no constraint propagation**. The only check is to be sure each new assignment violates no constraint with any previous assignment. As a tiebreaker, assign a lecturer to the earliest available time slot (so as to get them back to their own historical eras as soon as possible).

In the tree below, Alan Turing has already been scheduled to speak at 1 pm, in accordance with constraint #1. Continue filling in the search tree up to the first time you try (and fail) to assign a time slot to Isaac Newton, at which point you give up in frustration and move on to Part C in search of a more sophisticated approach.

T			1		
L		1	2	3	4
B					
C					
S					
P					
N					

Part C (20 points)

You're not fond of backtracking, so rather than wait and see just how much backtracking you'll have to do, you decide to start over and use **depth-first search with forward checking (constraint propagation through domains reduced to size 1)**. As before, your tiebreaker is to assign the earliest available time slot.

T 1

L

B

C

S

P

N

What is the final lecture schedule you hand in to MITTTS?

1 pm:

2 pm:

3 pm:

4 pm:

Part D (10 points)

Now, rather than backtracking, you're concerned about the amount of time it takes to keep track of all those domains and propagate constraints through them. You decide that the problem lies in the ordering of the guest list. Just then, you get a call from the MITTTS president, who informs you that **Alan Turing's schedule has opened up and he is now free to speak during any one of the four time slots.**

Armed with this new information, you reorder the guest list to maximize your chances of quickly finding a solution. In particular, which lecturer do you now assign a time slot to first, and why?

Constraint Satisfaction

Constraint satisfaction algorithms that we have learned about:

1. **DFS with Backtracking (BT).** No constraint checks are propagated ["Assignments only"]. One variable is assigned a single value at a time, and then the next variable, and the next, etc. Check to see if the current partial solution violates any constraint. Whenever this check produces a zero-value domain, backup occurs.
2. **DFS with Backtracking + Forward-checking.** A variable is assigned a unique value, and then *only its neighbors* are checked to see if their values are consistent with this assignment. That is: assume the current partial assignment, apply constraints and reduce domain of neighboring variables. ["Check neighbors of current variable only"].
3. **DFS with Backtracking + Forward-checking + propagation through singleton domains.** If during forward checking you reduce a domain to size 1 (singleton domain), then assume the assignment of the singleton domain and repeat forward checking from this variable. (Note that reduction to a singleton domain and repeated forward checking might lead to yet another singleton domain in some cases.)
4. **DFS with Backtracking + Forward checking + propagation through reduced domains (this is also known as: Arc-consistency, AC-2, since it ensures that all pairs of variables have consistent values).** Constraints are propagated through all **reduced domains** of variable values, possibly beyond immediate neighbors of current variable, until closure. ["Propagate through reduced domains"; also called "Waltz labeling"]

1. Map coloring – comparing Backtracking (BT) with Forward-checking+propagation through singleton domains.

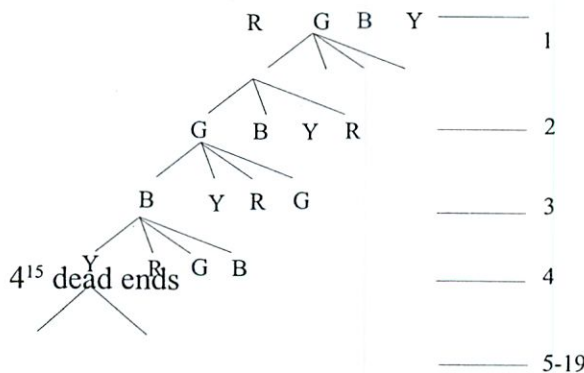
(a) How long will it take to color (two adjacent states cannot have the same color) the following map with 4 colors with just Backtracking (BT)?

19																	
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18

Assume that:

- Color the states in the order they are numbered.
- Use R(ed), G(reen), B(lue) and Y(ellow) in the given order & then in rotation, starting with R; i.e., when you finish assigning Y, start again with R
- coloring a location takes 1/100 sec

Draw the shape of the search tree for BT:



BT will take on the order of $4^{15} / 100$ seconds.

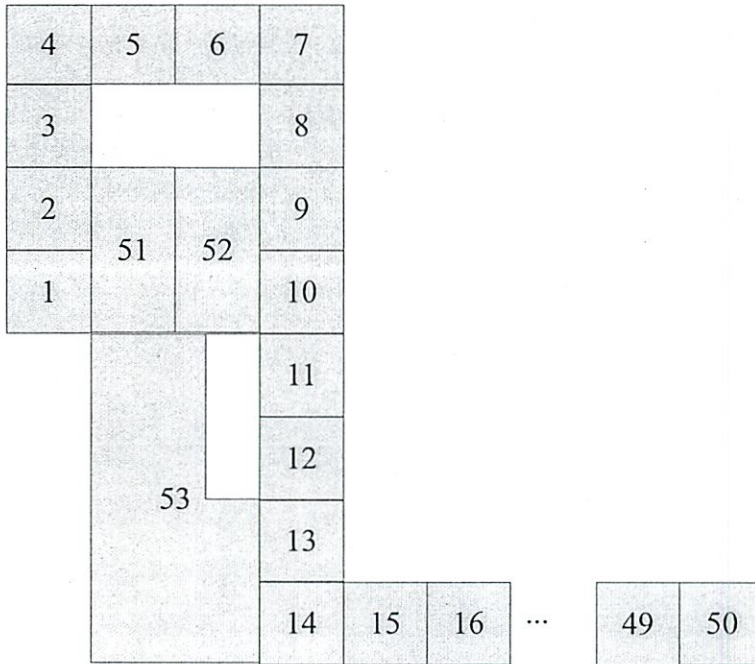
The search has a branching factor of 4, and for the above map, a depth of 15, since blocks 1-4 are assigned R-G-B-Y.

(b) How many seconds will **Forward checking+propagation through singleton domains** take on the *same* map?

FC+prop through singletons will take 0.2 seconds.

“R” is selected for square #1; this reduces the possible values for squares #2 and #19 to {G,B,Y}. Then square #2 is selected, and the next color in the rotation, G, is picked. This reduces #3 to the colors {R,B,Y} AND #19 to {B,Y}. #3 is then assigned B, and so #4 is reduced to {R,G,Y} and #19 to just {Y}. #19 is now a singleton, so Y has been in fact assigned to 19 so that this constrains square #4 to just {R,G}. Next, #4 is assigned R (the next color in the rotation after Y- remember, Y was just used), and so #5 is constrained to be {G,B,Y}, etc. (the rest continues down the line to square #18). About 20 colors are assigned, which takes about $20/100 = 0.2$ sec.

(c) Can FC+propagation through singletons finish the map below quickly? Why or why not? What about AC-2? (Propagation through reduced domains.)



Forward checking + propagation through singletons takes years (one must back up to square 14); because the colors available for 51, 52, and 53 are not reduced to 1, there is no propagation to their neighbors when they are checked.

Forward checking + propagation through reduced domains (AC-2) takes years as well; there is propagation, but the pairwise checking allows both yellow and blue for 51, 52, and 53, so there is no early backup, but then, at the end of the search, there are only 2 colors available for three states.

Suppose we proceeded not from square #1 to #53 but from #53 to #1. Would our result now change? What principle of efficient constraint propagation does this illustrate? Are there any other principles of efficient constraint propagation that you can think of?

Yes, both would finish in under a second. These examples are instances of the “use the most constraint first” idea.

2. Converting problems into constraint propagation form

“Paul, John, and Ringo are musicians. One of them plays bass, another plays guitar, and the third plays drums. As it happens, one of them is afraid of things associated with the number 13, another is afraid of Apple Computers, and the third is afraid of heights. Paul and the guitar player skydive; John and the bass player enjoy Apple Computers; and the drummer lives in an open penthouse apartment 13 on the thirteenth floor. What instrument does Paul play?”

How can we solve this problem? Try it yourself first, by any means you care, then we’ll see how to do it by – ta-da! – the magic of constraint propagation! (You might want to think about constraints when you solve it, and what the constraints are.)

What are the constraints? How might they be represented? We want to use the facts in the story to determine whether certain identity relations hold or are eXcluded. Here is our notation: assume $X(\text{Peter, Guitar Player})$ means “the person who is John is not the person who plays the guitar.” Further, this relation is symmetrical, so that if we have $X(\text{Peter, Guitar Player})$ then we also have, $X(\text{Guitar Player, Peter})$. In this notation, the facts become the following (of course all the symmetrical facts hold as well):

1. $X(\text{Paul, Guitar Player})$
2. $X(\text{Paul, Fears Heights})$
3. $X(\text{Guitar Player, Fears Heights})$
4. $X(\text{John, Fears Apple Computers})$
5. $X(\text{John, Bass Player})$
6. $X(\text{Bass Player, Fears Apple Computers})$
7. $X(\text{Drummer, Fears 13})$
8. $X(\text{Drummer, Fears Heights})$

Now we can represent the possible relations implicitly by means of entries in tables, and use constraint propagation. An X entry in a table denotes that the identity relation is excluded, and an I denotes that the identity relation actually holds. We can then use three tables, one to represent the possible identities between people and instrument players; one to represent the possible identities between people and fears; and a third to represent the possible relationships between instrument players and fears.

1. $X(\text{Paul}, \text{Guitar Player})$
2. $X(\text{Paul}, \text{Fears Heights})$
3. $X(\text{Guitar Player}, \text{Fears Heights})$
4. $X(\text{John}, \text{Fears Apple Computers})$
5. $X(\text{John}, \text{Bass Player})$
6. $X(\text{Bass Player}, \text{Fears Apple Computers})$
7. $X(\text{Drummer}, \text{Fears 13})$
8. $X(\text{Drummer}, \text{Fears Heights})$

	Instrument Player		
	Bass Player	Guitar Player	Drum Player
Paul	X	X	I
John	X	I	X
Ringo	I	X	X

	Fears		
	13	Apple Computers	Heights
Paul	X	I	X
John	I	X	X
Ringo	X	X	I

	Fears		
	13	Apple Computers	Heights
Bass player	X	X	I
Guitar player	I	X	X
Drum Player	X	I	X

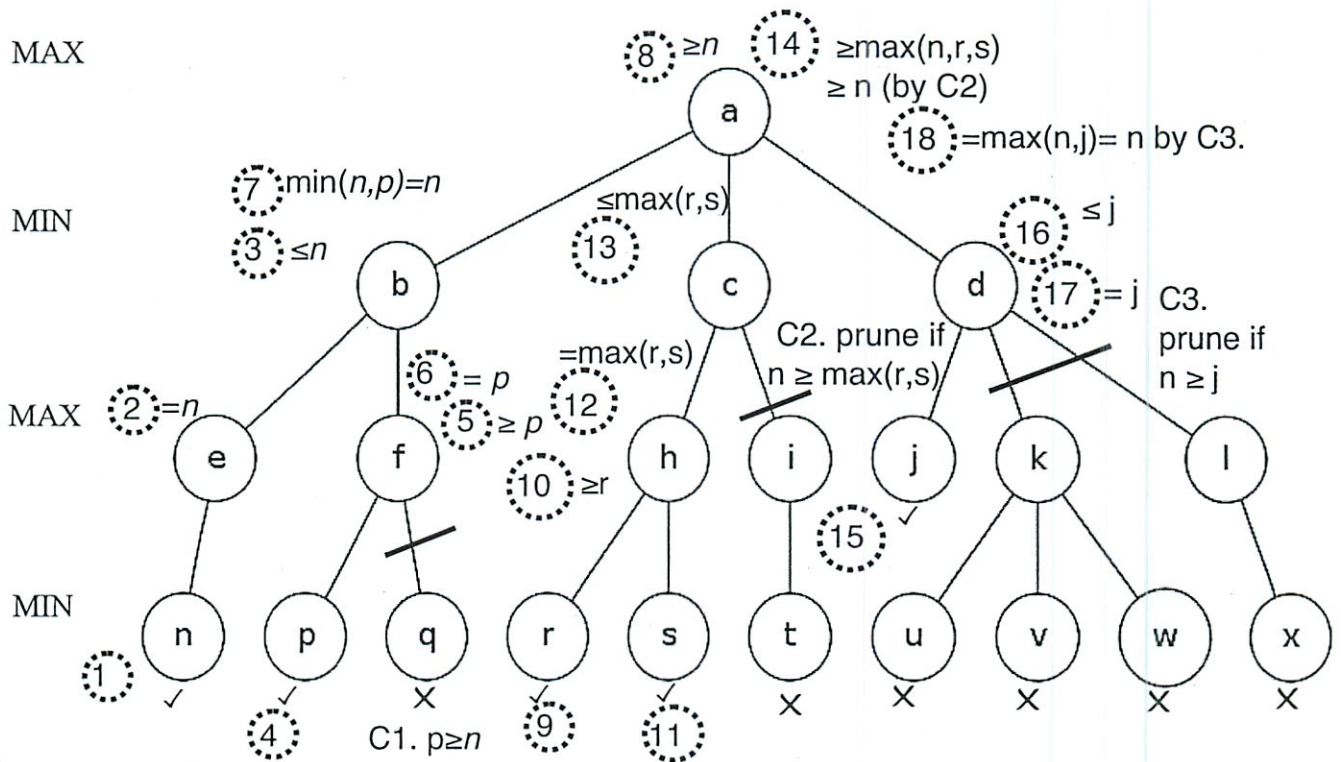
How do we do constraint propagation in this system? Note that we can deduce rules like the following to fill in the three tables:

1. If all but one entry in a row are X , then the remaining entry is I .
 2. If you know $I(x,y)$ and $X(y,z)$ then you may conclude $X(x,z)$.
- If two names pick out the same thing (are identical), then they must share all the same properties.
What other rules do we need?
3. If you know $X(x,y)$ & $X(z,y)$ then you may conclude $X(x,z)$
 4. If you know $X(x,y)$ & $X(x,z)$ then you may conclude $X(y,z)$

Problem 1: Games

Part A: Working with a maximally pruned tree (25 points)

For the following min-max tree, cross out those leaf nodes for which alpha-beta search would **not do static evaluations** in the best case possible (minimum number of static evaluations, maximum pruning of nodes to be statically evaluated).



Part A1

Now, list the leaf nodes at which alpha-beta would do static evaluations in the best case possible.

n, p, r, s, j

Part A2

What is the final value returned by the alpha beta search in the best case possible for the given tree? Express your answer as the simplest function of the static values of the leaf nodes (e.g. take n to be the static value at the leaf node labeled n). Your function may contain operations such as **max** and **min**.

n

Part A3

What constraints ensure best case possible (minimum static evaluation) for the given tree? State your constraints as inequalities on the static values of the leaf nodes.

C1. $p \geq n$
C2. $n \geq \max(r, s)$ or: $n \geq s$ AND $n \geq r$
C3. $n \geq j$

Part A4

Suppose your static evaluation function, $S(\text{node})$, is modified as follows:

$$S'(\text{node}) = 42 \times S(\text{node}) + 1000. \quad (\text{If } S(\text{node}) = 1, S'(\text{node}) = 1042)$$

Would your answer for Part A1 be the same for all possible $S(\text{node})$ values?

Yes No

Suppose your function were

$$S'(\text{node}) = -42 \times S(\text{node}) + 1000.$$

Would your answer for Part A1 be the same for all possible $S(\text{node})$ values?

Yes No

Problem 2: Time Travelers' Convention

The MIT Time Travel Society (MITTTS) has invited seven famous historical figures to each give a lecture at the annual MITTTS convention, and you've been asked to create a schedule for them. Unfortunately, there are only four time slots available, and you discover that there are some restrictions on how you can schedule the lectures and keep all the convention attendees happy. For instance, physics students will be disappointed if you schedule Niels Bohr and Isaac Newton to speak during the same time slot, because those students were hoping to attend both of those lectures.

After talking to some students who are planning to attend this year's convention, you determine that they fall into certain groups, each of which wants to be able to see some subset of the time-traveling speakers. (Fortunately, each student identifies with at most one of the groups.) You write down everything you know:

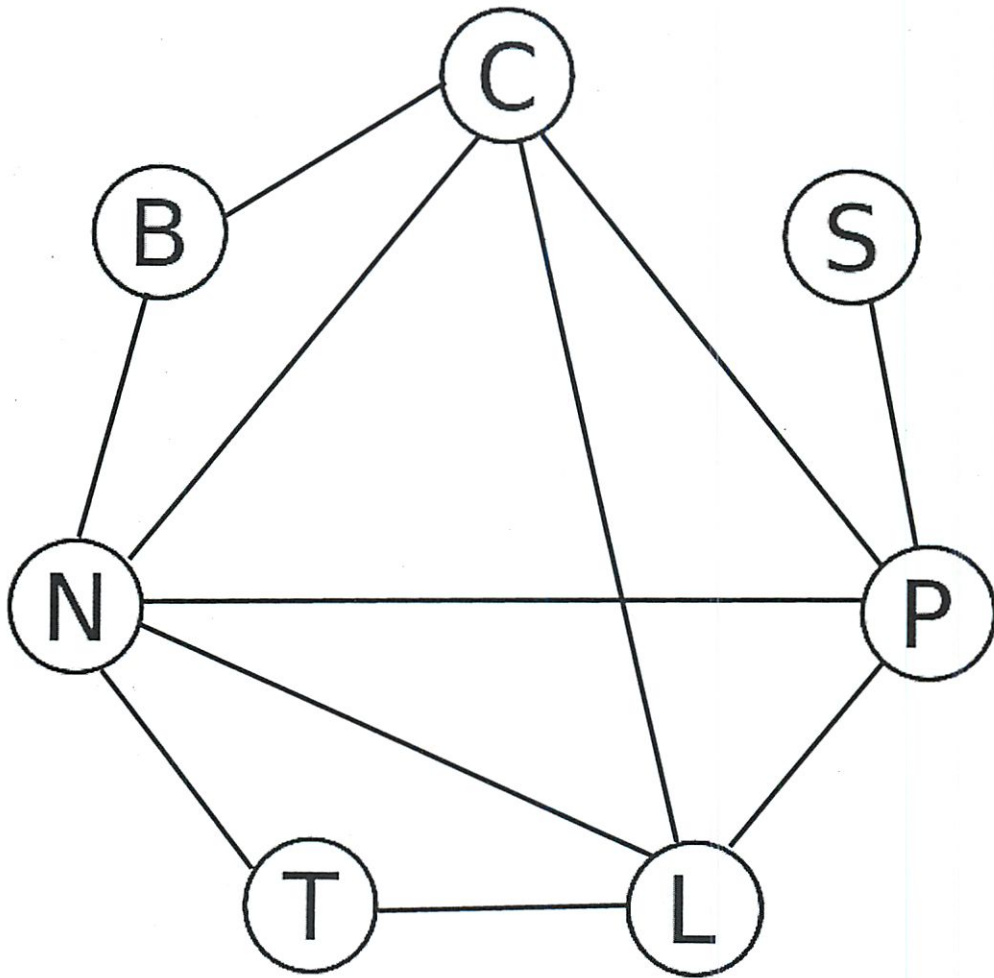
The list of guest lecturers consists of Alan Turing, Ada Lovelace, Niels Bohr, Marie Curie, Socrates, Pythagoras, and Isaac Newton.

- 1) Turing has to get home early to help win World War II, so he can only be assigned to the 1pm slot.
- 2) The Course VIII students want to see the physicists: Bohr, Curie, and Newton.
- 3) The Course XVIII students want to see the mathematicians: Lovelace, Pythagoras, and Newton.
- 4) The members of the Ancient Greece Club wants to see the ancient Greeks: Socrates and Pythagoras.
- 5) The visiting Wellesley students want to see the female speakers: Lovelace and Curie.
- 6) The CME students want to see the British speakers: Turing, Lovelace, and Newton.
- 7) Finally, you decide that you will be happy if and only if you get to see both Curie and Pythagoras. (Yes, even if you belong to one or more of the groups above.)

Part A (5 points)

That's a lot of preferences to keep track of, so you decide to draw a diagram to help make sense of it all. Draw a line between the initials of each pair of guests who must not share the same time slot.

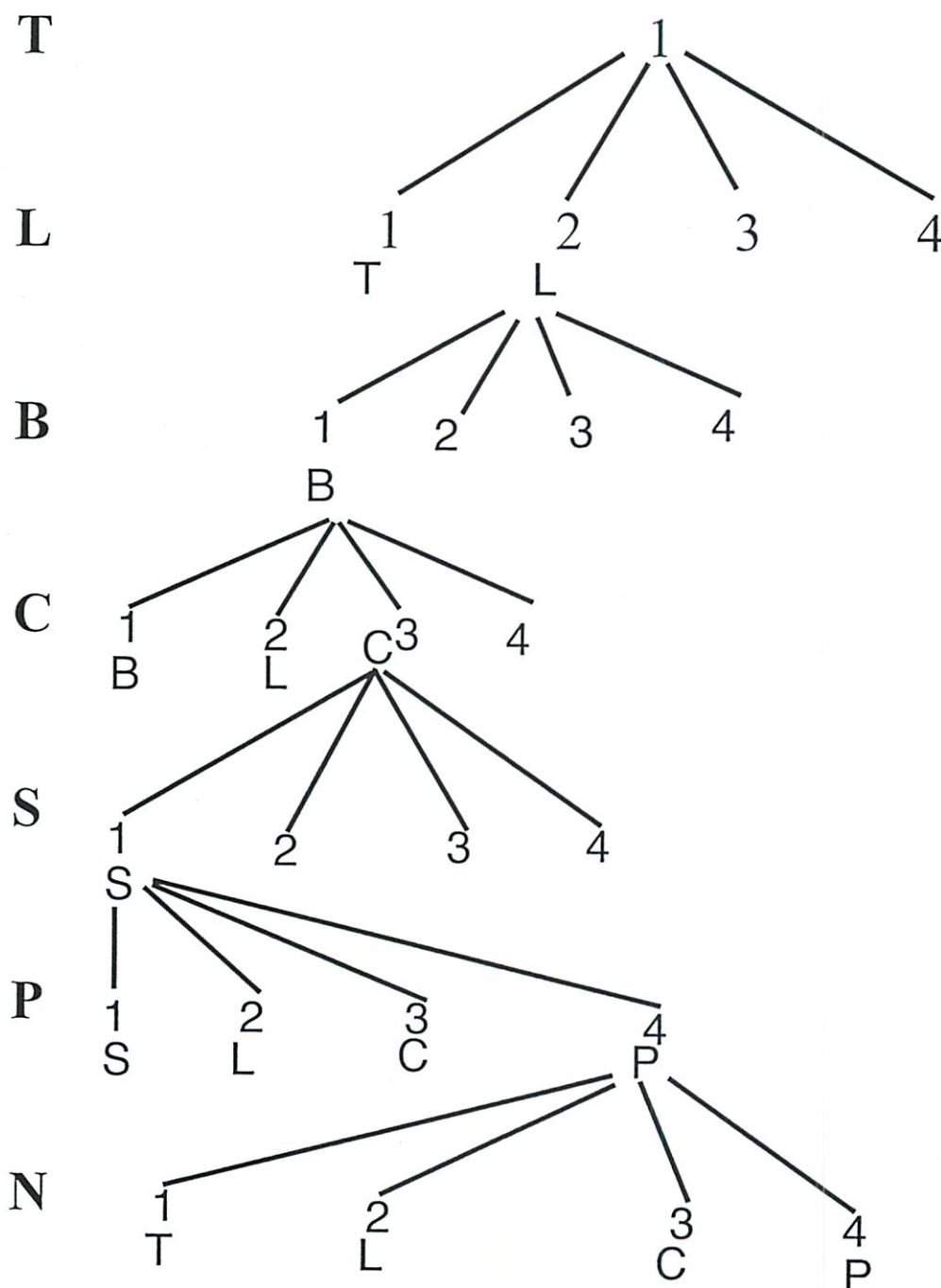
This diagram is repeated on the tear off sheet.



Part B (15 points)

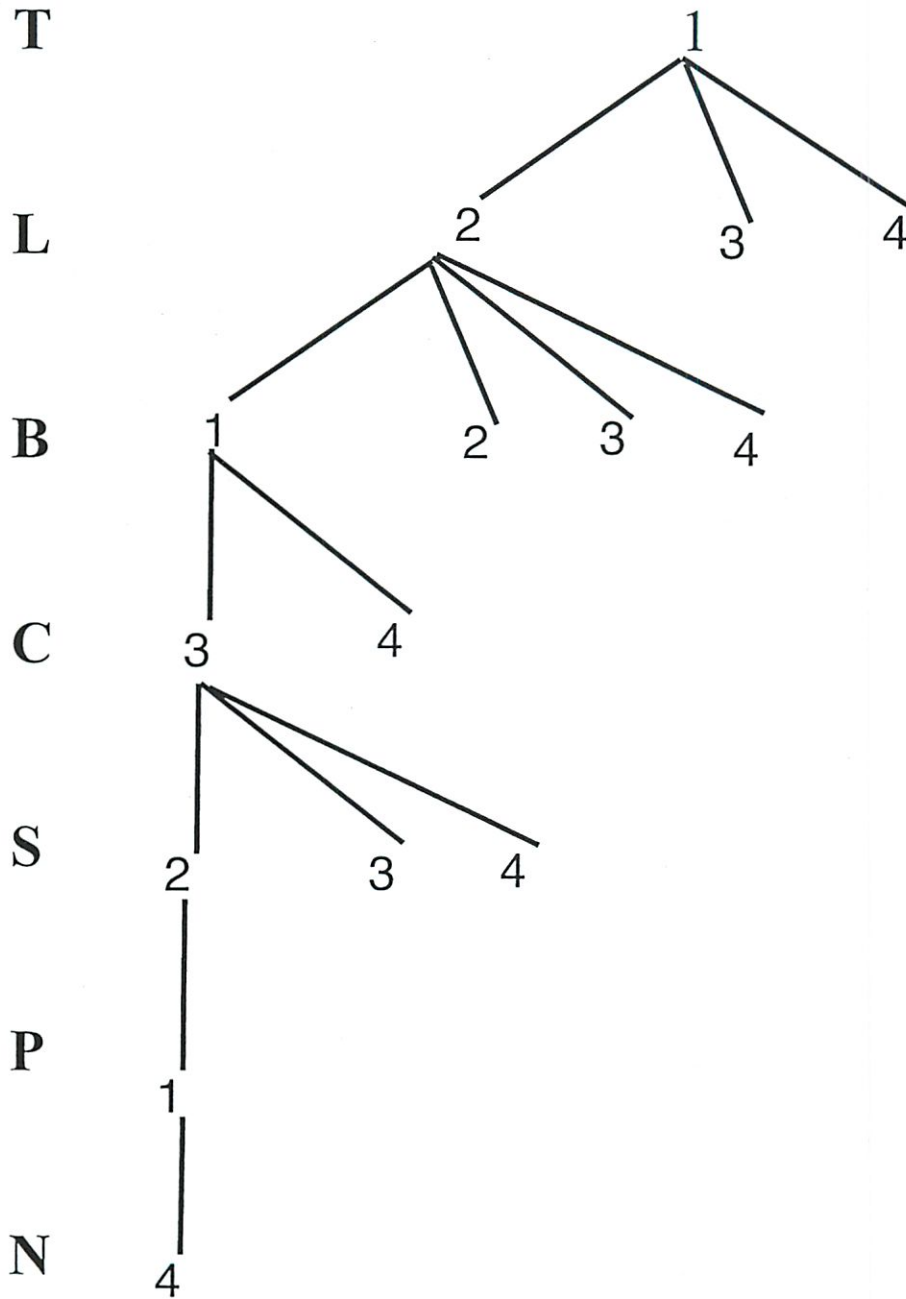
You decide to first assign the time slots (which conveniently happen to be 1, 2, 3, and 4 pm) by using a **depth-first search with no constraint propagation**. The only check is to be sure each new assignment violates no constraint with any previous assignment. As a tiebreaker, assign a lecturer to the earliest available time slot (so as to get them back to their own historical eras as soon as possible).

In the tree below, Alan Turing has already been scheduled to speak at 1 pm, in accordance with constraint #1. Continue filling in the search tree up to the first time you try (and fail) to assign a time slot to Isaac Newton, at which point you give up in frustration and move on to Part C in search of a more sophisticated approach.



Part C (20 points)

You're not fond of backtracking, so rather than wait and see just how much backtracking you'll have to do, you decide to start over and use **depth-first search with forward checking** (constraint propagation through domains reduced to size 1). As before, your tiebreaker is to assign the earliest available time slot.



What is the final lecture schedule you hand in to MITTTS?

1 pm: T, P, B

2 pm: S, L

3 pm: C

4 pm: N

Part D (10 points)

Now, rather than backtracking, you're concerned about the amount of time it takes to keep track of all those domains and propagate constraints through them. You decide that the problem lies in the ordering of the guest list. Just then, you get a call from the MITTTS president, who informs you that **Alan Turing's schedule has opened up and he is now free to speak during any one of the four time slots.**

Armed with this new information, you reorder the guest list to maximize your chances of quickly finding a solution. In particular, which lecturer do you now assign a time slot to first, and why?

Newton - has the most constraints

Silver Star Ideas

- * Adjacent, search tree \rightarrow Neighbor \subset must be constrained together
- * Do the right thing ^{visually}
- * Keep Domains in the margin

Hard problem

- Lets you pick domain and variables
- But pushes you to wrong thing

Variables \rightarrow seat or who ? ?

domain \rightarrow who or seat ? ?

Could do either way.

How star

Do the right thing

- there are 4 possible things

Pure constraint propagation - like Sudoku

- no search/guess a value
- is there anything that can possibly work

2)

Like if say Rep only even # seats
L have already lost

1. McCain insists on seat 1

Mark on both charts

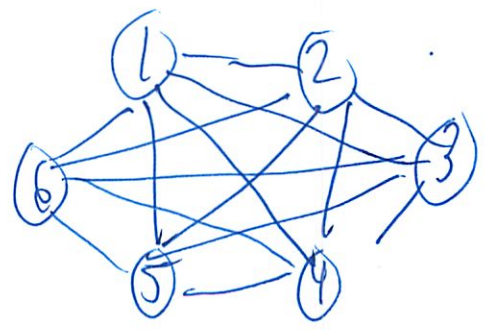
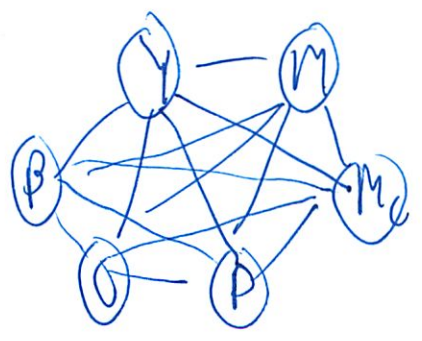
You	1	2	3	4	5	6
Moose	1	2	3	4	5	6
McCain	1	2	3	4	5	6
Palin						
Obama						
Biden						

1	X	M	Mc	R	O	B
2	Y	M	Mc	P	O	B
3			?			
4						
5						
6						

crossing this off
only when
run pre
constraint
propagation

2. Each person 1 seat

- So can make graph
- will be complete graph



3

3. Moose afraid of Palin

4. Political opponents won't sit next to each other

N_p & next to D

5. Political allies must be next to each other

So put Palin near in for 2 or 6

Can cross out O, B in 2, 6 ^{-cross out elsewhere}

Do some stuff

As much as you can do

Constraint Prop does not do more

Yor	X	2	3	4	5	6
Moose	X	2	3	4	5	6
ML	1	2	3	4	5	6
P	X	2	3	4	5	6
O	2	2	3	4	5	6
B	1	2	3	4	5	6

2nd one...

4
Weird ternary constraints on doing the last one

- hard to backtrack immediately

- hard / ~~to~~ impossible to know ahead of time

4 possible things

1. Pure Constraint

2. Basic Search

- DFS, no constraint propagation

- check value when done it

- if wrong, backtrack

- dumb

3. ~~Basic~~ Basic Search w/ Forward Checking

4. Basic Search propagation thru singletons

- assign variables that only have 1 choice

5

Now do DFS (no constraint propagation) from where

we are after pure constraints

↑ don't always do this 1st

~~Backtrack~~ Mc

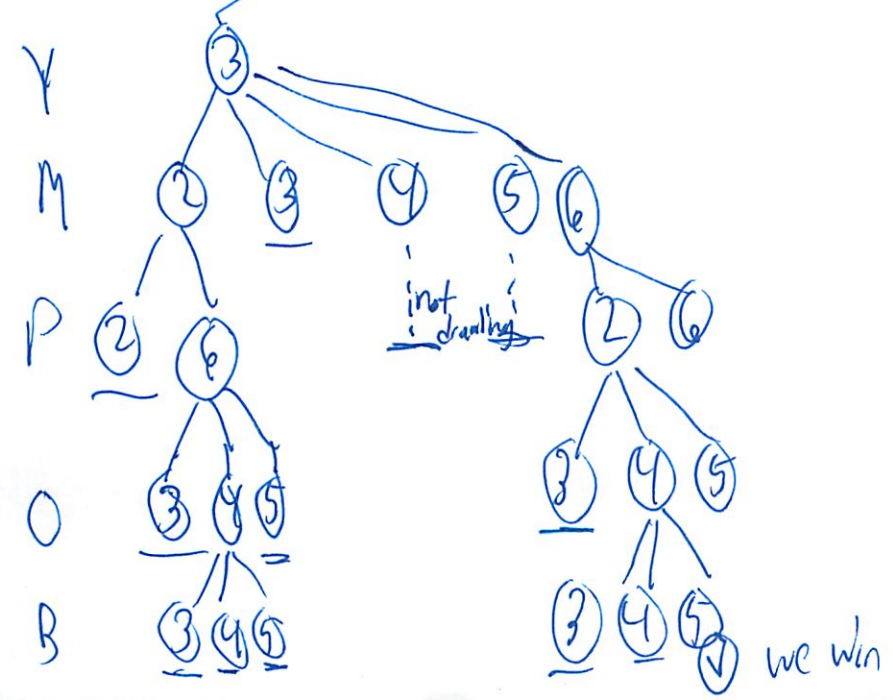
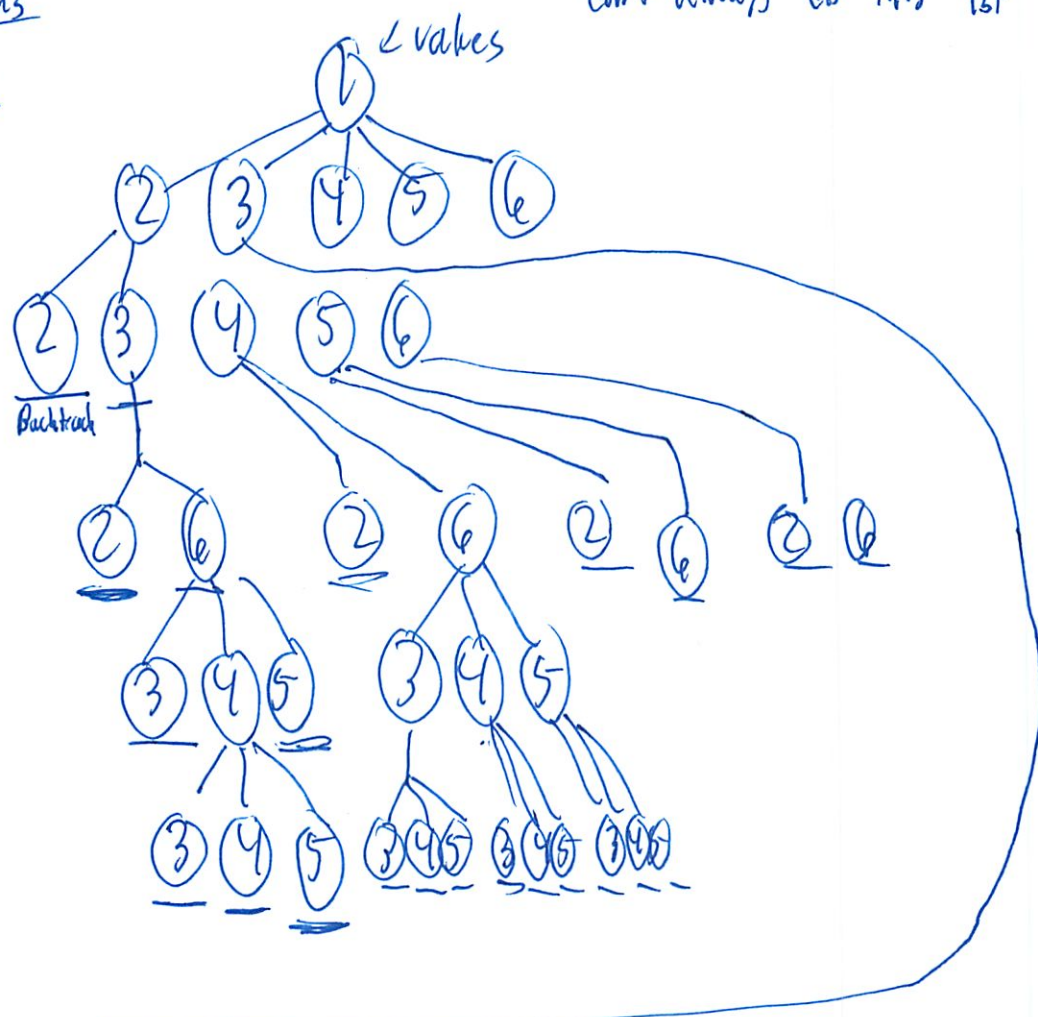
Assign in this order ↓ Y

No pre constraint checking → M

P

O

B



6

Skip to Forward Checking w/ Singletons

M/C

1

Y
D(P) = 6 2 3 4 5 6

+ possible assign to 6
do actually do
work to BT

possibles → P(M) = 3, 4
P(O) = 3, 4
P(B) = 3, 4
→ sudoku solver
knows bad
but our
program does
not

margin
notes
only
applies
for
Y=2

M

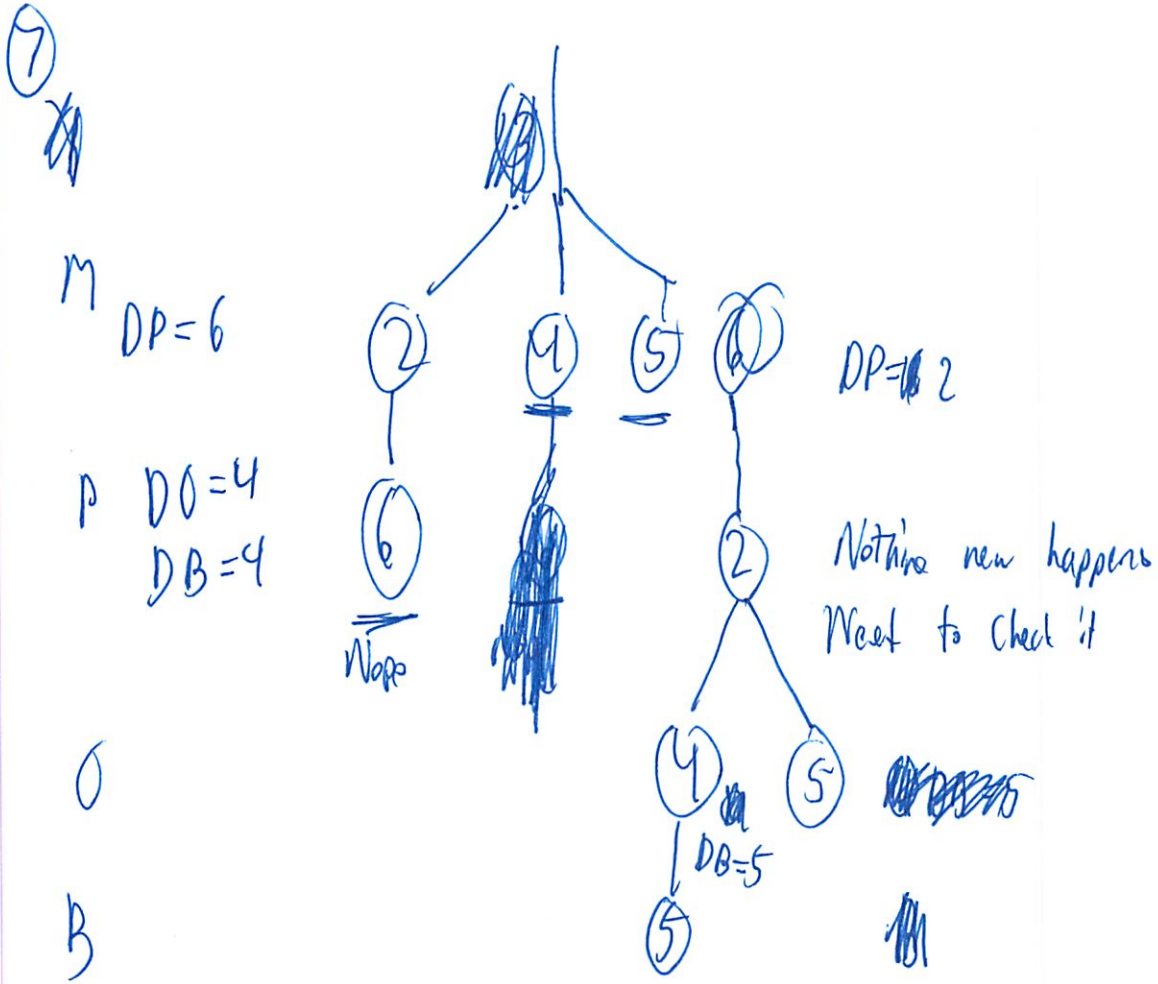
3 4

will fail through O, B
singleton propagation

Y

DM = 2, 3, 5, 6
DO = 4, 5
DB = 4, 5

3



If you can sort variables before hand it makes it a lot easier

Most constrained lst is better

Method
For #3 - same domain checking
But don't propagate downward
So just write it

Don't always ask for constraint propagation

6.034 Lecture
Recognition

10/12

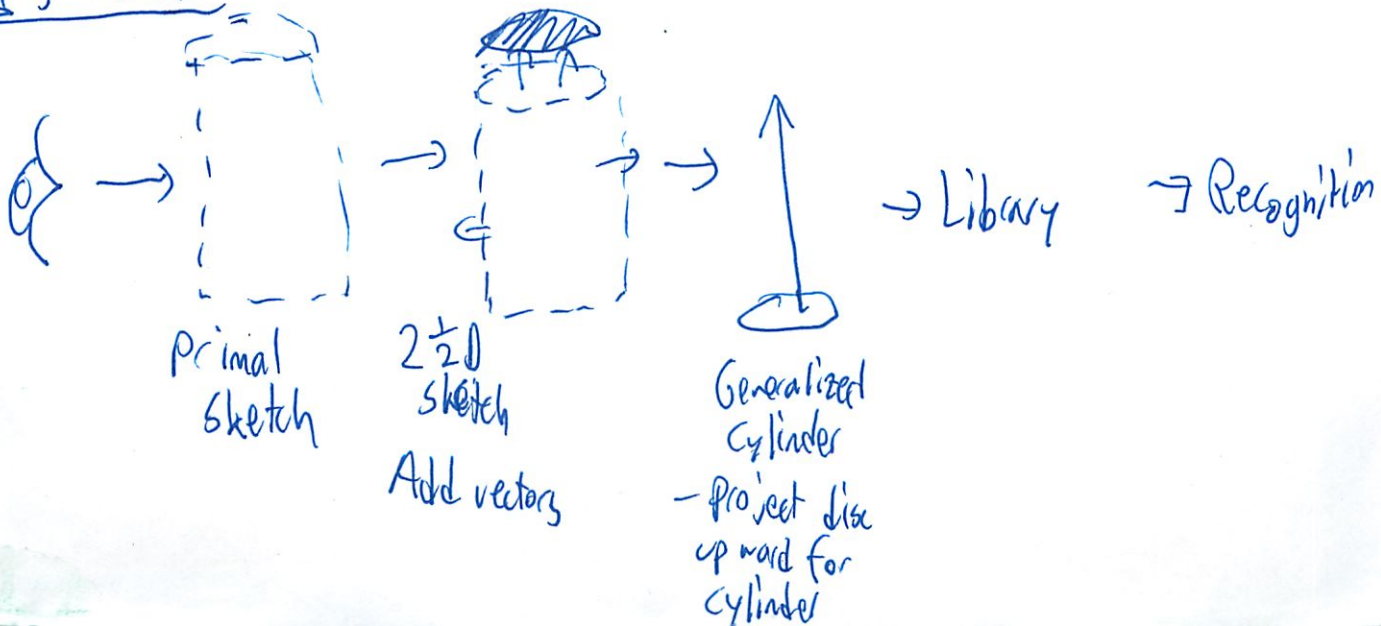
- Gospel according to Marr
- Ullman's alignment theory
- Ullman's intermediate feature story
- * Rempelstilskin principle
- * Power of correlation
- * Goldilocks principle
- * Vision as stories

Musi: the first time I ever saw you Elvis Presley

Big Concept lecture
Humans very good at recognizing stuff
How???

~~Primal sketch~~

Marr's Model



②

Or for bottle



Marr was very good at naming things
- got power over them (rumplestiltskin)

No one could get this to work

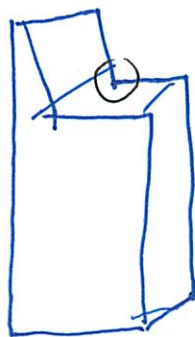
If have a ~~all these~~ few line drawings
- could you get a sense of the figure?



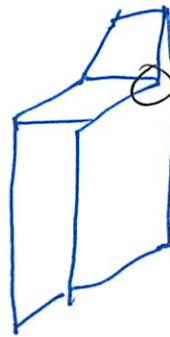
A



B



C



D

$$X_D = \alpha X_A + \beta X_B + \gamma X_C + T$$

trivially true - set to 0

3

So do another pt

see black

And do 2 more pts

So have 4 eq

$$\begin{array}{l}
 X_u = \alpha X_A + \beta X_B + \gamma X_C + T \\
 X_v = \alpha X_A + \beta X_B + \gamma X_C + T \\
 X_w = \alpha X_A + \beta X_B + \gamma X_C + T \\
 X_z = \alpha X_A + \beta X_B + \gamma X_C + T
 \end{array}$$

↑ all the same!

4 eq 4 unknowns (α, β, γ, T)

Since know X_A, X_B, X_C for each
- off the diagram for each pt

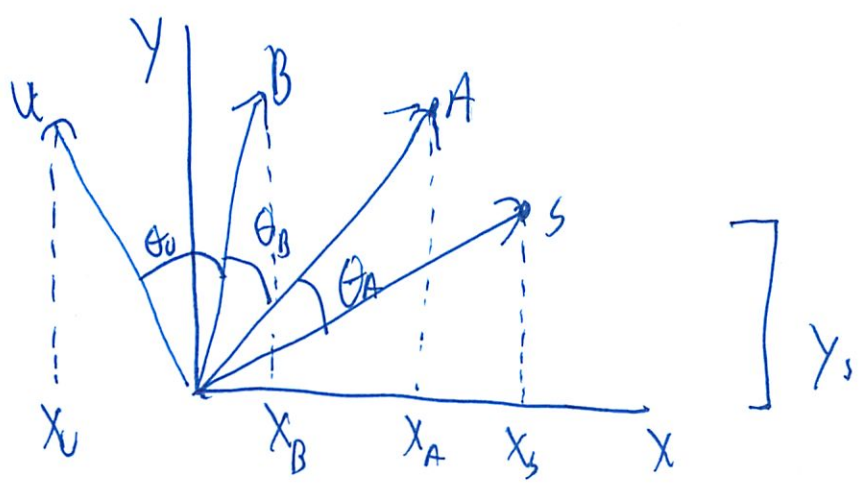
Now can use same #s for any pt in image

↳ can predict where pt will be

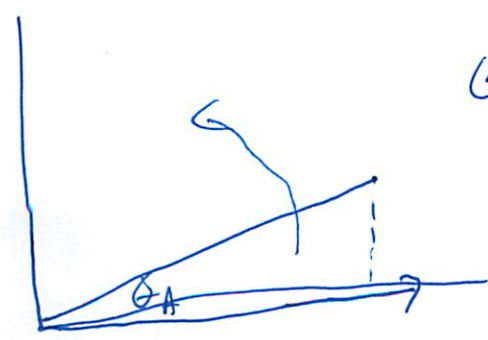
Demo: Pick 3 pts in each of 3 figures

4

Not true for ~~projects~~ perspective
Only orthographic projection
Look down on object from top
and rotate it

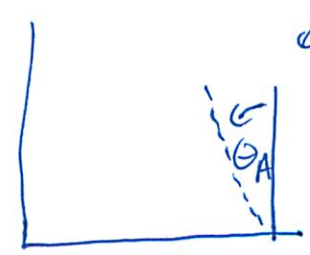


Write expression for X_s, Y_s
 then rotate components individually
 - then add
 - So have X_A, Y_A



$$X_A = X_s \cos \theta_A$$

$$Y_A = Y_s \sin \theta_A$$



$$X_A = X_s \cos \theta_A - Y_s \sin \theta_A$$

5

$$x_B = x_s \cos \theta_B - y_s \sin \theta_B$$

$$x_{all} = x_s \cos \theta_u - y_s \sin \theta_u$$

Can rewrite lot eq to get x_s

- find

- then sub in for x_s in those things

Same for y_s

So

$$x_u = f_1(\sin s + \cos s) x_A + f_2(\sin s + \cos s) x_B$$

So taking a vertex and rotating it

- f_n is same for all vertices

- So can rotate an object through space

Rewrite

$$x_u = \alpha x_A + \beta x_B$$

Works well on manufactured objects

Not good at recognizing trees ^{-all the same}

- each one is different

(6)

Same for facial recognition

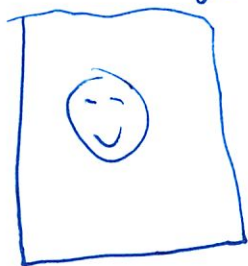
- lots of things can change - messing up system

3rd approach

Correlation

↳ common in 6-1

Have an image 



↳ want to find it
in the picture

So scan over image looking for correlation

$$\text{Max} \int \int_x \int_y f(x, y) g(x + X, y + Y)$$

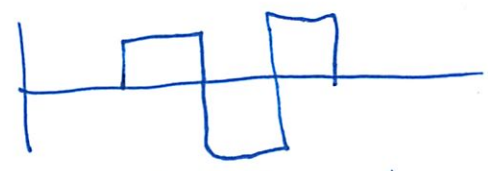
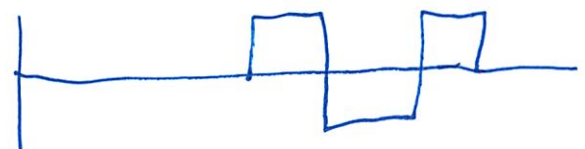
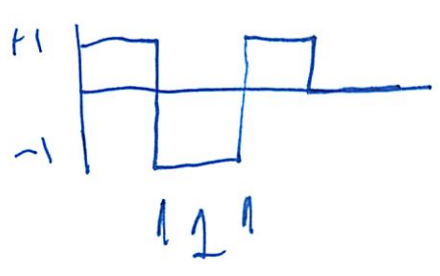
Brighten image where strong correlation

- to see visually

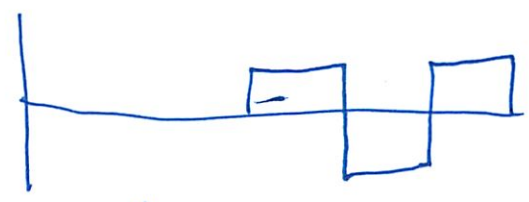
Works w/ a lot of noise

7

Signal



- only 1 section
superimpose



- all sections super
imposed



Need large lib of faces to do this

A face

Another one

training
data



Inputs



Which one
is the
face?

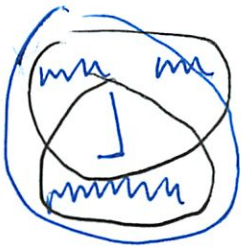
(8)

Clearly #1

But what does PC do?

Chap faces up into intermediate sized pieces
Find closest correlation

Find 2 pieces ~~that~~ from 2 faces that overlap



But pieces can't be too small

So



Works!
But it shouldn't

Intermediate size = goldilocks principle

Shakespeare

Macbeth_{only} = too broad

king = too small - in every play

So look for intermediate thing

revenge = just right → Hamlet

9

Need to find corresponding parts

Marker for human to see upside down

1. Finish Progressive Deepening
- finish ordering nodes

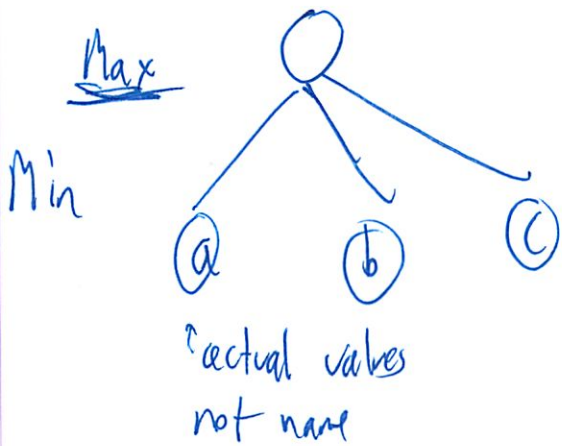
2. Constraint propagation finish



Future

1. Simplest classification
- function approximates
2. Info theory
3. Neural networks
4. Support vector machines

3. Learning + Objects



* get max of B cutoff,
want best line of play
to be on left

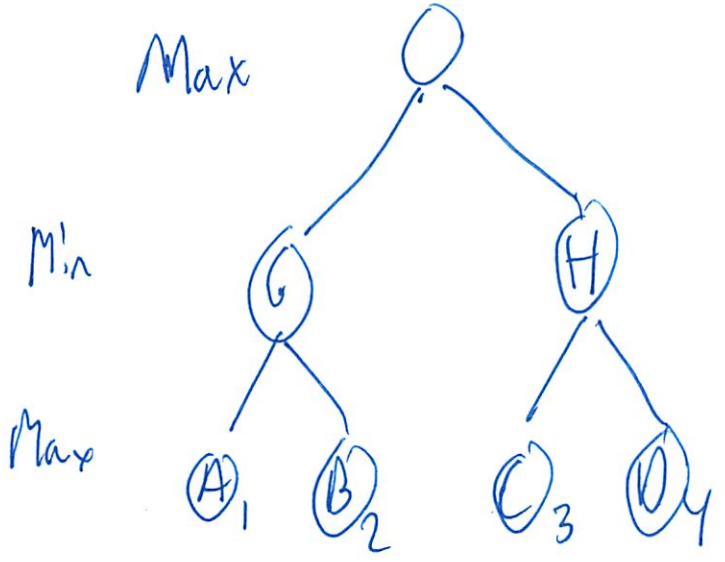
$$a > b \quad a > c$$

At Min level want values to descend

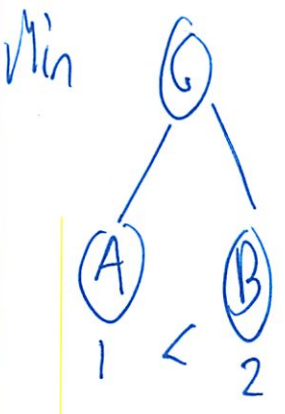
$$a < b < c$$

②

Pseudo code will reorder tree

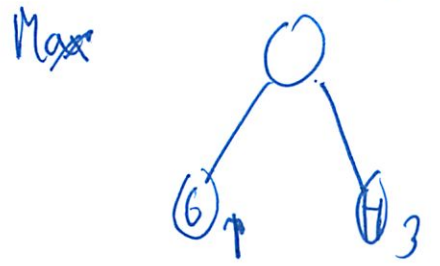


Progressive deepening goes down to a certain level
and then starts looking



'is goal for minimizer
will get cutoff

Then ~~it~~ do other + propagate up



3

This one should be switched!



So most optimal original is



Is best original case

#2 Same qv, diff tree



#3 Similar qv on pg 4

- Do minimax
- Do α, β pruning
- Do ordering

(4)

So best way to do this is to run
 minimax and then write # next to nodes
 Then work from top down to order the nodes
 Expand nodes and then sort inside for their branch

2. Constraint Propagation

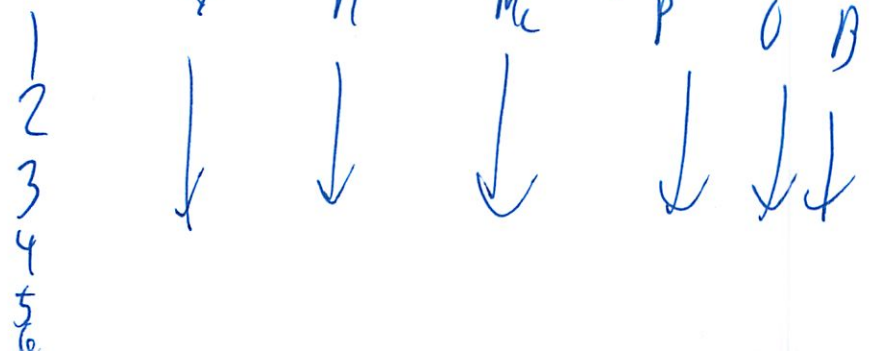
Same problem in mega recitation
 Will do dual version

This is the Palin / Moore / Obama etc table thing

Domain: seat # $\{1, \dots, 6\}$
 Variables: people $\{Y, M, Mc, P, O, B\}$

↪ Can also do it the other way!

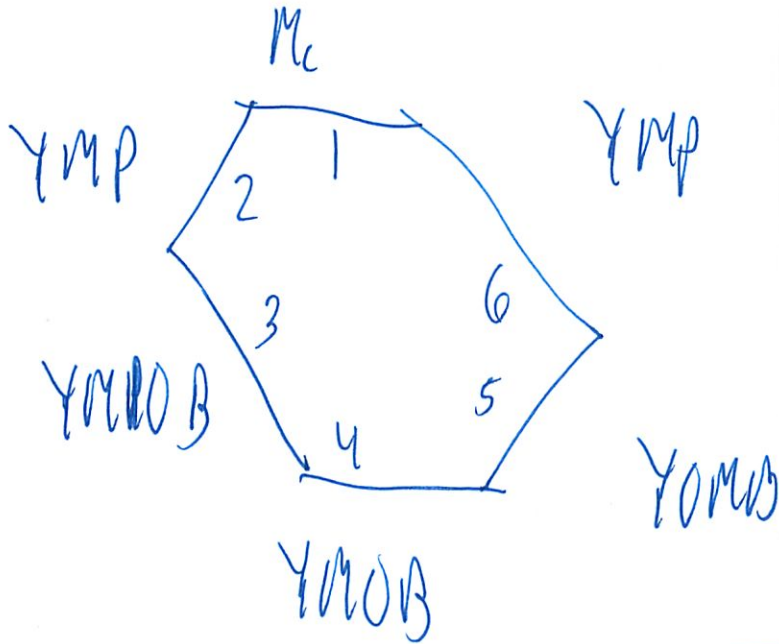
Go through chart and impose the static constraints



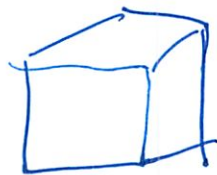
5

Compare 1. Depth First Search

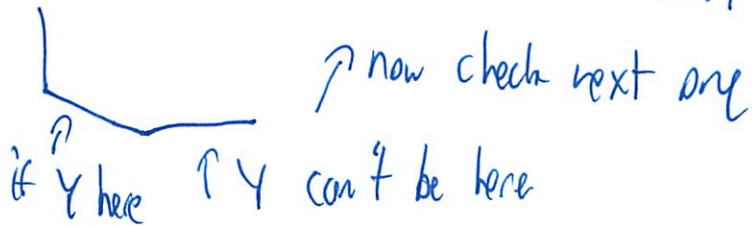
2. Forward checking and propagate domains w/ n



Arc consistency is best on square-line labling

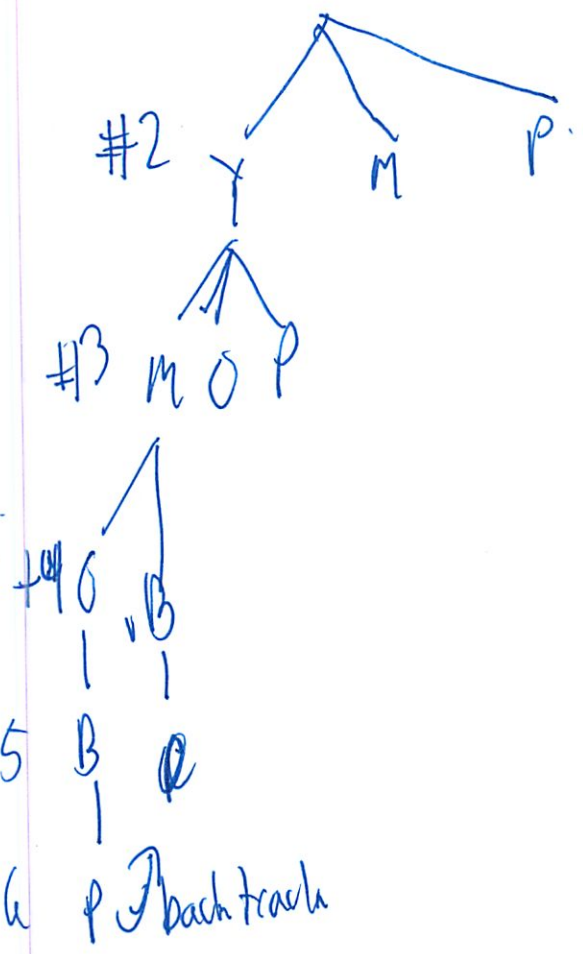


- Won't do much here
- Best on line labling
- Very hard to on board + exam



6

If wanted to do DFS



The static checking we did was an option
- can skip

Singleton domain - means if only 1 left
pursue that

(7)



↑ try one and see if we get to sol or we get to impossible ~~problem~~ area

- like 0 people possible in a seat

- or two people who can't sit next to each other

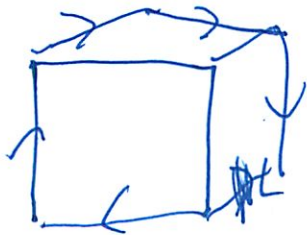
- or no seat for person

if so go back up tree

else if it works continue down tree

Etc - ~~it~~ did not write at

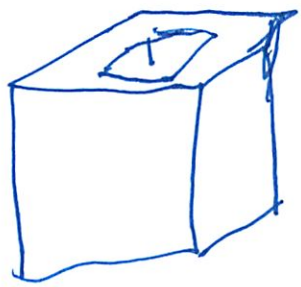
Library of line labels available for download



Labels Patrick gave

8

Implicit constraint is can have implicitly small holes



will ~~fail~~ fail

How you put stuff into machine is key

3. Next 3 weeks: Learning

Simplest: classification task

Take object x

Measure a # of attributes

Classify as something

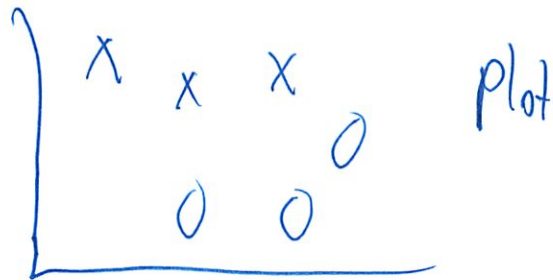
$$f(x_1, \dots, x_n) \begin{matrix} \rightarrow \text{Spam} \\ \rightarrow \text{Mail} \end{matrix}$$

attributes



9

Nearest neighbor



You are whatever you are closest to
Actually a lot of human learning does this
- memorization

But in middle/borderline
- equal distance b/w them

So can move from a monarchy \rightarrow democracy

So use 3 nearest neighbors

Can't really go beyond 5 for accuracy

Works well on human genome

3 is also more robust

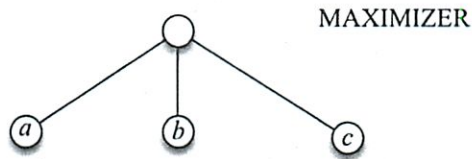
Recitation 5, Thursday, October 13

Game trees & progressive deepening; constraint prop revisited

Prof. Bob Berwick, 32D-728

1. Games and progressive deepening.

To ensure the **maximum** amount of pruning in alpha-beta search, one must ensure that the **best** line of play (for either the minimizer or the maximizer) is **always on the left**. This implies that for the **maximizer**, the best (let us say **biggest**) evaluations are always on the left, which implies that at any one maximizer level, for best alpha-beta pruning the evaluations at the leaves or nodes just below should be ordered from **largest to smallest**. That is, for the maximizer, in the following tree, the nodes a, b, c should have values such that $a \geq b \geq c$.



For the **minimizer**, the best move should also be always on the left, and this implies that the best (or **smallest**) evaluations are always on the left, which implies that at any one level, for best alpha-beta pruning the evaluations of the leaves or nodes just below should be ordered from **smallest to largest**. This suggests the following general tree rotation optimizing function:

```
function tree-rotation-optimizer
  From level in [1 ... n]
  if level is MAX then
    sort nodes at this level from smallest to largest
  else if level is MIN then
    sort nodes at this level from largest to smallest
```

We can use this function in conjunction with the idea of **progressive deepening** to heuristically sort the static evaluations of a game tree, in breadth first order:

Step 1. Run alpha-beta-search up to depth d .

This will yield some static values for (un-pruned) leaf nodes at depth d .

Step 2. Using these computed static values compute the values associated with intermediate nodes by running minimax from leaf up to root.

Step 3. Run the tree-rotation-optimizer on this (possibly-pruned) tree.

Step 4. Record at each node, the ordering of its children.

Step 5. Run alpha-beta search at depth $d + 1$.

But when computing the next-moves for any node, lookup the node-ordering from step 4.

Evaluate alpha-beta with nodes sorted using this ordering. NOTE: Any nodes not in the ordering will automatically receive the lowest priority; such nodes come from having been pruned in alpha beta search.

Note: The results of the tree rotation optimizer only **influences** alpha-beta search evaluation order. It will not actually guarantee maximum pruning at every stage. This is because:

1. We are using the rotation algorithm with static values at d to **approximate** the values of the tree at $d+1$.
2. Pruned nodes from alpha-beta-search will not be used by the rotation algorithm. While the tree-rotation doesn't yield maximum pruning, it does enhance pruning, so we still get a worthwhile speed-up.

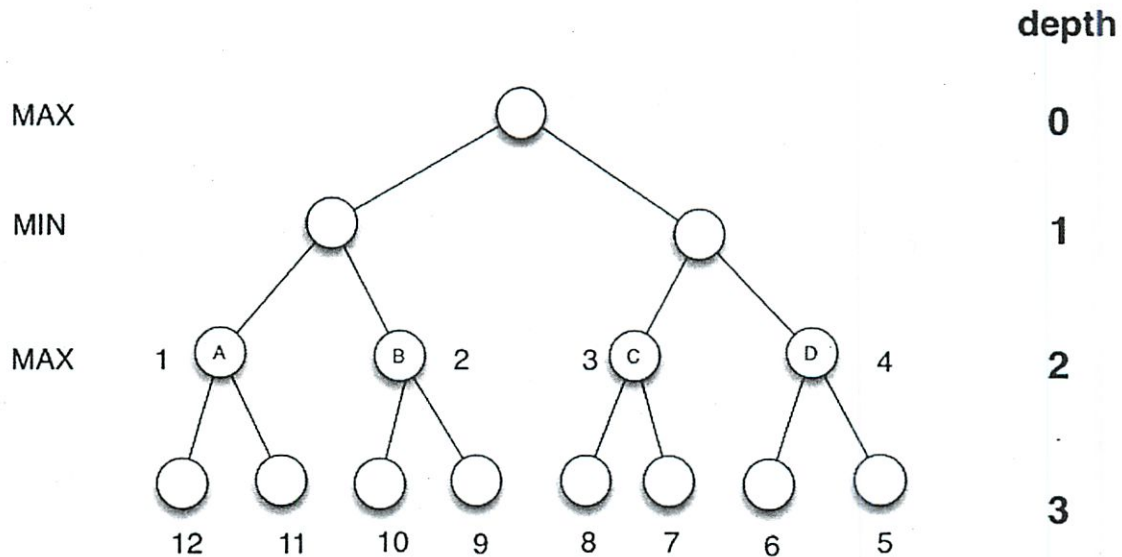
Let's test our ideas on how this works on a couple sample exam problems.

10/13/11 Minimax with Alpha-Beta Pruning and Progressive Deepening

When answering the question in Parts C.1 and C.2 below, assume you have already applied minimax with alpha-beta pruning and progressive deepening on the corresponding game tree up to **depth 2**. The value shown next to each node of the tree at depth 2 is the respective node's static-evaluation value. Assume the procedure uses the information it has acquired up to a given depth to try to improve the order of evaluations later. In particular, the procedure reorders the nodes based on the evaluations found up to depth 2 in an attempt to improve the effectiveness of alpha-beta pruning when running up to depth 3.

We want to know in which order the nodes/states A, B, C, and D in the game tree are evaluated when the procedure runs up to depth 3, after running up to depth 2 and reordering.

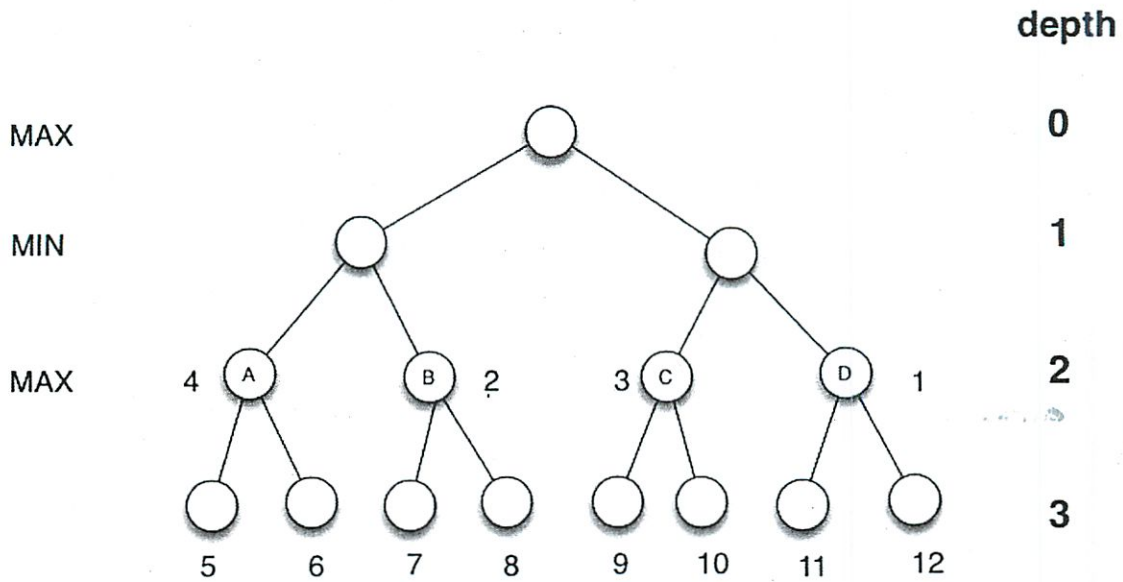
Part C.1: Game Tree I (5 points)



Choose the order in which the nodes/states A, B, C and D in game tree I above are evaluated when running minimax with alpha-beta pruning and progressive deepening after running up to depth 2 and reordering. (Circle your answer)

- a. A B C D
- b. D A B C
- c. B A D C
- d. C D A B
- e. D C B A

Part C.2: Game Tree II (5 points)

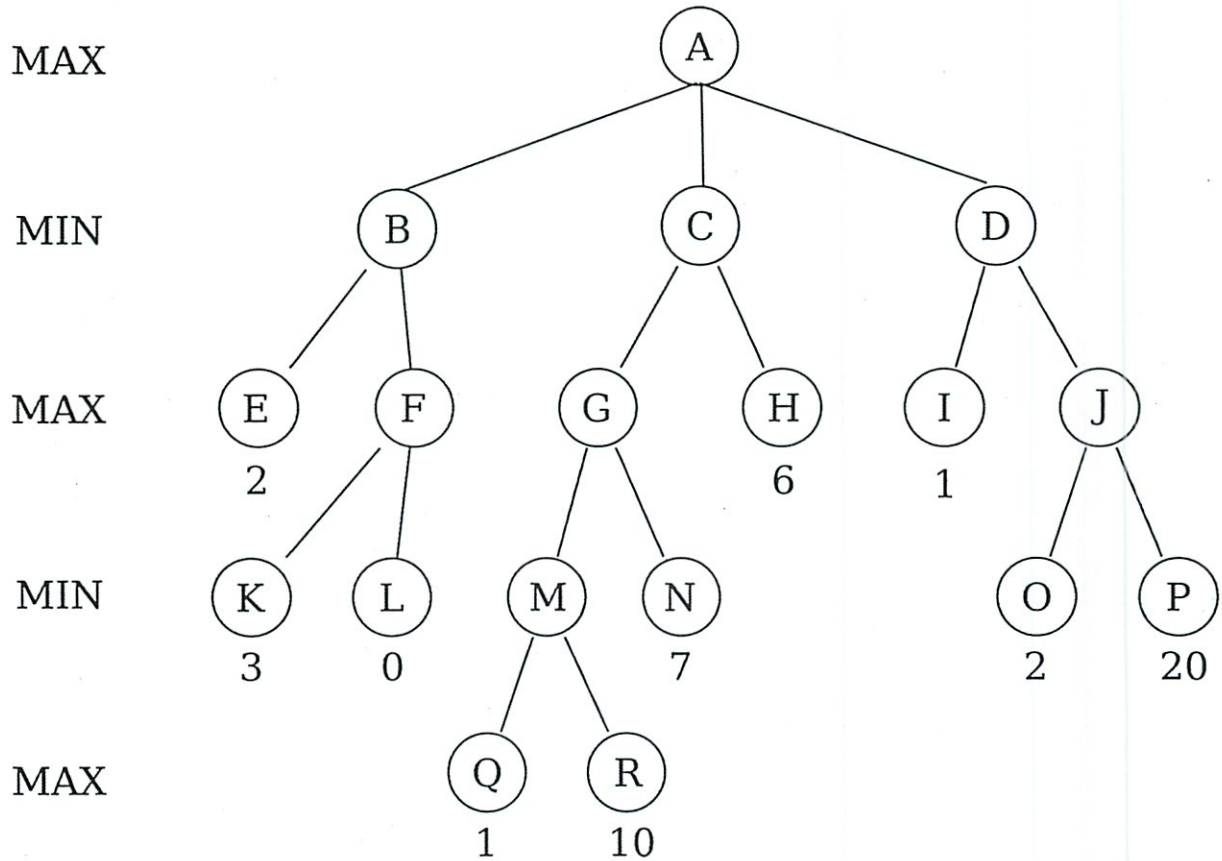


Choose the order in which the nodes/states A, B, C and D in game tree II above are evaluated when running minimax with alpha-beta pruning and progressive deepening after running up to depth 2 and reordering (Circle your answer)

- f. A B C D
- g. D A B C
- h. B A D C
- i. C D A B
- j. D C B A

10/13/11 Problem 2: Games

In the game tree below, the value below each node is the static evaluation at that node. MAX next to a horizontal line of nodes means that the maximizer is choosing on that turn, and MIN means that the minimizer is choosing on that turn.



Part A (10 points)

Using minimax without Alpha-Beta pruning, which of the three possible moves should the maximizer take at node A?

What will be the final minimax value of node A?

Part B (10 points)

Mark suggests that Alpha-Beta pruning might help speed things up. Perform a minimax search with alpha-beta pruning, traversing the tree, and list the order in which you **statically evaluate** the nodes (that is, you would start with E). Write your answer below. **Note that there is, at the end of this quiz, a tear-off sheet with copies of the tree.**

Part C (10 points)

Tom thinks that he might save some trouble by calculating static values at depth 2 and then using those static values to reorder the tree for the alpha-beta search that goes all the way to the leaf nodes. Thus, Tom is attempting to deploy alpha-beta search in a way that will improve pruning.

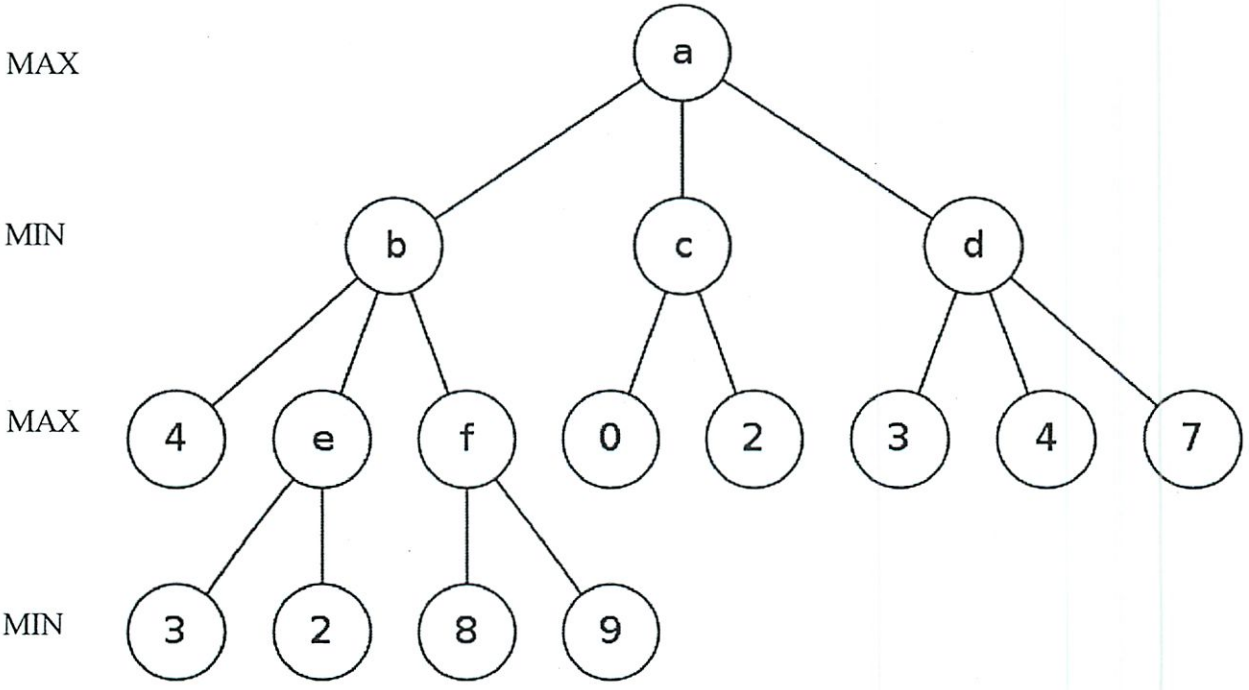
Suppose the static evaluator Tom uses at depth 2 produces exactly the minimax values you found in Part A. Tom uses those numbers to reorder BC&D, EF, GH, and IJ. Note that Tom knows nothing about how the tree branches below depth 2 at this point. Draw the reordered tree down to depth 2, the EFGHIJ level.

Explain the reasoning behind your choices:

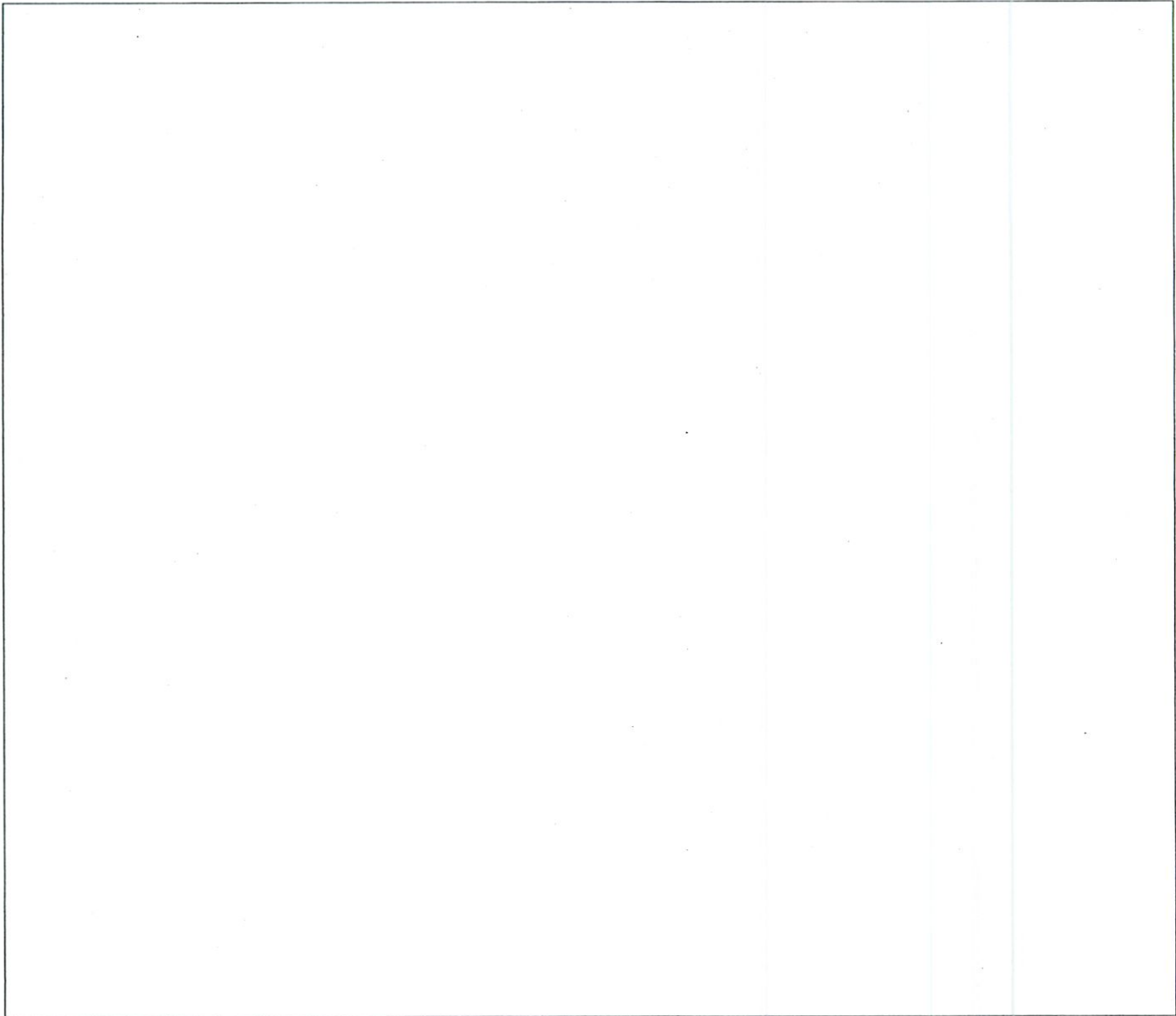
Explain your reasoning in less than 4 meaningful sentences or give a short proof.

Part B1: Digging Progressively Deeper (10 points)

You decide to put your 6.034 knowledge to good use by entering an adversarial programming contest. In this contest, one player (your opponent) is tasked with optimizing the running time of **alpha beta search with progressive deepening**. Your goal, as her adversary, is to slow down her algorithm. Remembering a key insight from 6.034: node order affects the amount of alpha-beta pruning, you decide to do the exact opposite: you decide to reorder your opponent's search tree so as to **eliminate** alpha beta pruning. To practice, you perform your anti-optimization on the following tree.



Reorder the nodes of the tree at every level such that alpha-beta search does **no pruning**. You may only reorder nodes (you cannot reattach a node to a different parent). Show your reordered tree below:



2. Constraint propagation revisited: definitions, constraint propagation types, algorithms

Constraint Types

- **Unary Constraints** – constraint on single variables (often used to reduce the initial domain)
- **Binary Constraints** – constraints on two variables, $C(X, Y)$
- **n-ary Constraints** – constraints involving n variables. Any n-ary variables $n > 2$
(Can always be expressed in terms of Binary constraints + Auxiliary variables)

Search Algorithm types:

1. DFS w/ BT + basic constraint checking: Check current partial solution see if you violated any constraint.
2. DFS w/ BT + forward checking: Assume the current partial assignment, apply constraints and reduce domain of other variables.
3. DFS w/ BT + forward checking + propagation through singleton domains: If during forward checking, you reduce a domain to size 1 (singleton domain), then assume the assignment of singleton domain, repeat forward checking from singleton variable.
4. DFS w/ BT with propagation through reduced domains (AC-2): see below.

Note: You can replace DFS with Best-first Search or Beam-search if variable assignments have “scores,” or if you are interested in the best assignments.

Definitions.

- k -consistency. If you set values for $k-1$ variables, then the k th variable can still have some non-zero domain.
- 2-consistency = arc-consistency.
 - For all variable pairs (X, Y)
 - for any settings of X there is an assignment available for Y
- 1-consistency = node-consistency
 - For all variables there is *some* assignment available.
- Forward Checking is only checking consistency of the current variable with all neighbors.
- Arc-consistency = propagation through reduced domains
- Arc-consistency ensures consistency is maintained for all *pairs* of variables, also known as AC-2
- Variable domain (VD) table – bookkeeping for what values are allowed for each variable.

Variable Selection Strategies:

1. Minimum Remaining Values Heuristic (MRV): Pick the variable with the smallest domain to start.
2. Degree Heuristic – usually the tie breaker after running MRV: When domains sizes are same (for all variables), choose the variable with the most constraints on other variables (greatest number of edges in the constraint graph)

Forward Checking Pseudo Code:

Assume all constraints are Binary on variables X, and Y.

constraint.get_X gives the X variable

constraint.get_Y gives the Y variable

X.domain gives the current domain of variable X.

forward_checking(state)

1. run basic constraint checking fail if basic check fails
2. Let X be the current variable being assigned,
3. Let x be the current value being assigned to X.
4. **check_and_reduce**(state, X=x)

check_and_reduce(state, X=x)

1. neighbor_constraints = get_constraints(X)
2. foreach constraint in neighbor_constraints:
3. Y = constraint.get_Y()
4. skip if Y is already assigned
5. foreach y in Y.get_domain()
6. if check_constraint(constraint, X=x, Y=y) fails
7. reduce Y's domain by removing y.
8. if Y's domain is empty return fail
9. return success

forward_checking_with_prop_thru_singletons(state)

1. run forward checking
2. queue = find all unassigned variables that have domain size = 1
3. while queue is not empty
4. X = pop off first variable in queue
5. **check_and_reduce**(state, X = x.DOMAIN[0])
6. singletons' = find all new unvisited unqueued singletons
7. add singletons' to queue

forward_checking_with_prop_thru_reduced_domains(state)

1. run arc-consistency(state)

arc-consistency(state)

1. queue = all (X, Y) variable pairs from binary constraints
2. while queue not empty
3. (X, Y) get the first pair in queue
4. if **remove-inconsistent-values**(X, Y) then
5. for each Y' neighbors(X)
6. do add (Y', X) to queue if not already there

remove-inconsistent-values(X, Y)

1. reduced = false
2. for each x in X.domain
3. if **no** y in Y.domain satisfies constraints(X=x, Y=y)
4. remove x from X.domain
5. reduced = true
6. return reduced

Let's try some examination problems with constraint-propagation.

2. Converting problems into constraint propagation form

“Paul, John, and Ringo are musicians. One of them plays bass, another plays guitar, and the third plays drums. As it happens, one of them is afraid of things associated with the number 13, another is afraid of Apple Computers, and the third is afraid of heights. Paul and the guitar player skydive; John and the bass player enjoy Apple Computers; and the drummer lives in an open penthouse apartment 13 on the thirteenth floor. What instrument does Paul play?”

How can we solve this problem? Try it yourself first, by any means you care, then we’ll see how to do it by – ta-da! – the magic of constraint propagation! (You might want to think about constraints when you solve it, and what the constraints are.)

What are the constraints? How might they be represented? We want to use the facts in the story to determine whether certain identity relations hold or are eXcluded. Here is our notation: assume $X(\text{Peter, Guitar Player})$ means “the person who is John is not the person who plays the guitar.” Further, this relation is symmetrical, so that if we have $X(\text{Peter, Guitar Player})$ then we also have, $X(\text{Guitar Player, Peter})$. In this notation, the facts become the following (of course all the symmetrical facts hold as well):

1. $X(\text{Paul, Guitar Player})$
2. $X(\text{Paul, Fears Heights})$
3. $X(\text{Guitar Player, Fears Heights})$
4. $X(\text{John, Fears Apple Computers})$
5. $X(\text{John, Bass Player})$
6. $X(\text{Bass Player, Fears Apple Computers})$
7. $X(\text{Drummer, Fears 13})$
8. $X(\text{Drummer, Fears Heights})$

Now we can represent the possible relations implicitly by means of entries in tables, and use constraint propagation. An X entry in a table denotes that the identity relation is excluded, and an I denotes that the identity relation actually holds. We can then use three tables, one to represent the possible identities between people and instrument players; one to represent the possible identities between people and fears; and a third to represent the possible relationships between instrument players and fears.

1. X(Paul, Guitar Player)
2. X(Paul, Fears Heights)
3. X(Guitar Player, Fears Heights)
4. X(John, Fears Apple Computers)
5. X(John, Bass Player)
6. X(Bass Player, Fears Apple Computers)
7. X(Drummer, Fears 13)
8. X(Drummer, Fears Heights)

People	Instrument Player		
	Bass Player	Guitar Player	Drum Player
Paul			
John			
Ringo			

People	Fears		
	13	Apple Computers	Heights
Paul			
John			
Ringo			

Instrument Player	Fears		
	13	Apple Computers	Heights
Bass player			
Guitar player			
Drum Player			

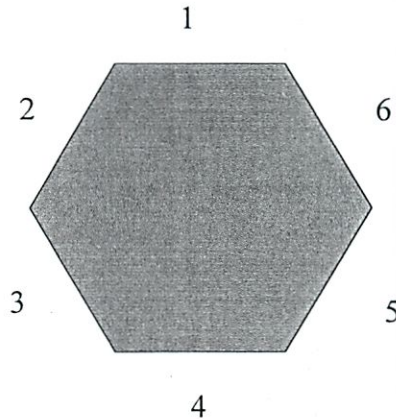
How do we do constraint propagation in this system? Note that we can deduce rules like the following to fill in the three tables:

1. If all but one entry in a row are X , then the remaining entry is I .
 2. If you know $I(x,y)$ and $X(y,z)$ then you may conclude $X(x,z)$.
- Question: what property of the world does this last constraint rule capture?

What other rules do we need?

10/13/11 Constraint Propagation

You just bought a 6-sided table (because it looks like a benzene ring) and want to hold a dinner party. You invite your 4 best friends: McCain, Obama, Biden and Palin. Luckily a moose wanders by and also accepts your invitation. Counting yourself, you have 6 guests for seats labeled 1-6.



Your guests have seven seating demands:

- Palin wants to sit next to McCain
- Biden wants to sit next to Obama
- Neither McCain nor Palin will sit next to Obama or Biden
- Neither Obama nor Biden will sit next to McCain or Palin
- The moose is afraid to sit next to Palin
- No two people can sit in the same seat, and no one can sit in 2 seats.
- McCain insists on sitting in seat 1

Part A (10 points)

You realize there are 2 ways to represent this problem as a constraint problem. For each below, run the domain reduction algorithm and continue to propagate through domains reduced to one value. That is, **cross out all the impossible values in each domain without using any search.**

Variables: You, Moose, McCain, Palin, Obama, Biden

Domains: Seats 1-6

Constraints: I-VII

You:	1	2	3	4	5	6
Moose:	1	2	3	4	5	6
McCain:	1	2	3	4	5	6
Palin:	1	2	3	4	5	6
Obama:	1	2	3	4	5	6
Biden:	1	2	3	4	5	6

Variables: Seats 1-6

Domains: You, Moose, McCain, Palin, Obama, Biden

Constraints: I-VII

1:	You	Moose	McCain	Palin	Obama	Biden
2:	You	Moose	McCain	Palin	Obama	Biden
3:	You	Moose	McCain	Palin	Obama	Biden
4:	You	Moose	McCain	Palin	Obama	Biden
5:	You	Moose	McCain	Palin	Obama	Biden
6:	You	Moose	McCain	Palin	Obama	Biden

Part B (15 points)

For now, you decide to continue using seats as variables and guests as domains (the 2nd representation). You decide to see how a depth-first search with no constraint propagation works, but as you run the search you see you are doing a lot of backtracking.

You break ties by choosing the left-most in this order:

You (Y), Moose (M), McCain(Mc), Palin (P), Obama (O), Biden (B)

Show the partial search tree *only* up to the first time you try to **assign seat 6**.

Begin with the reduced domains from the previous

page. Use only the constraints supplied; use no commonsense beyond what you see in the constraints. You might want to work this out on the tear off sheet at the end of the quiz first.

1			Mc	
2	Y		M	P
3				
4				
5				
6				

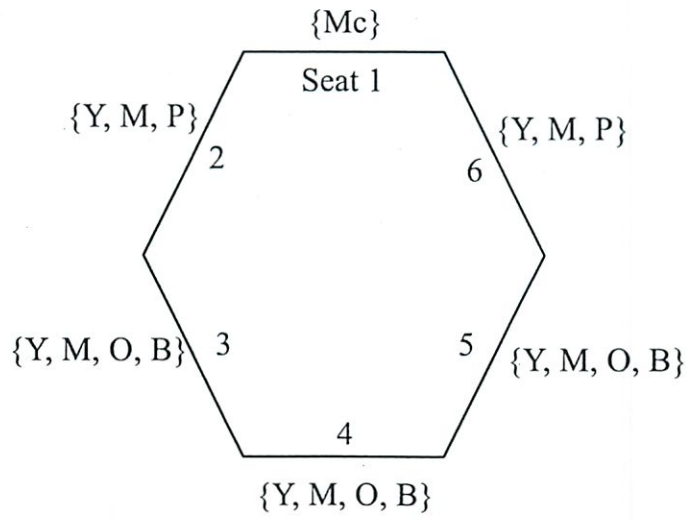
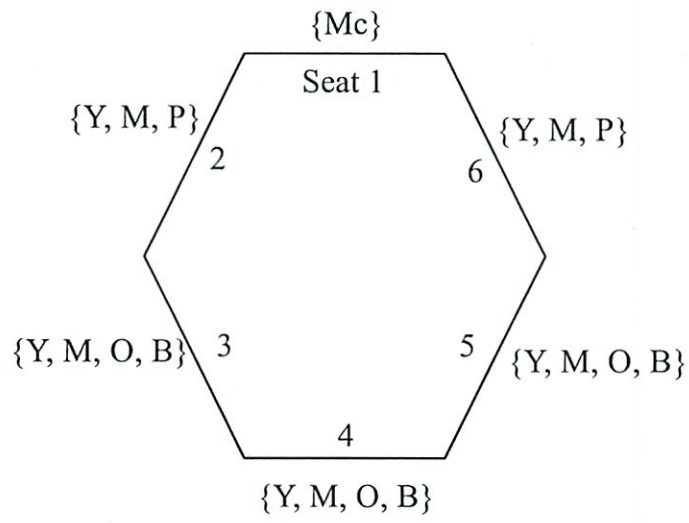
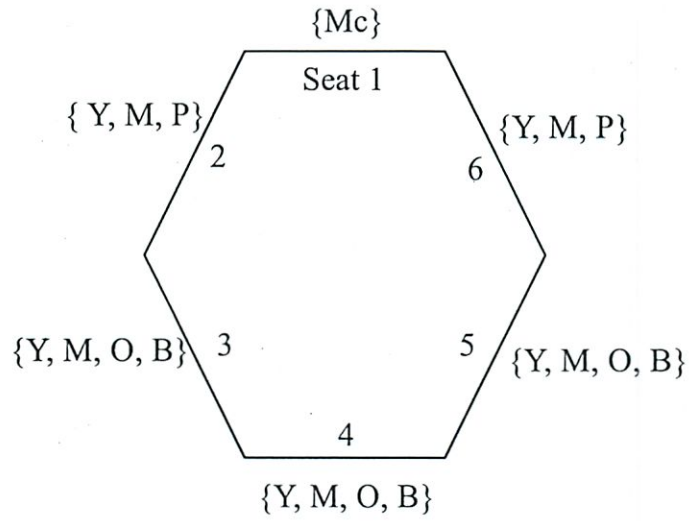
Part C (15 points)

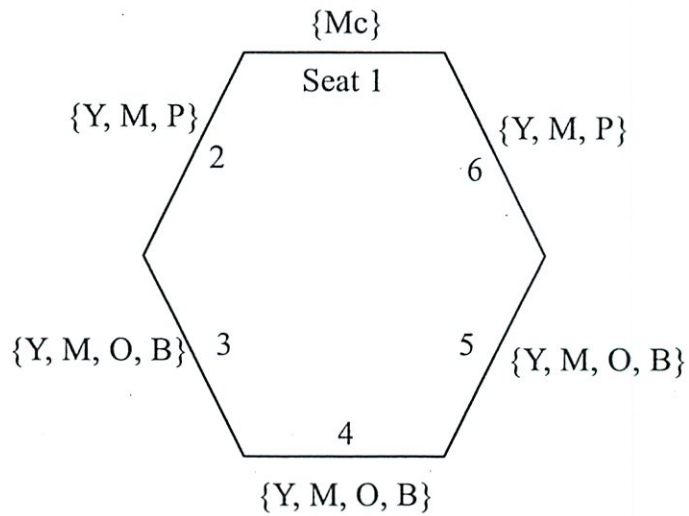
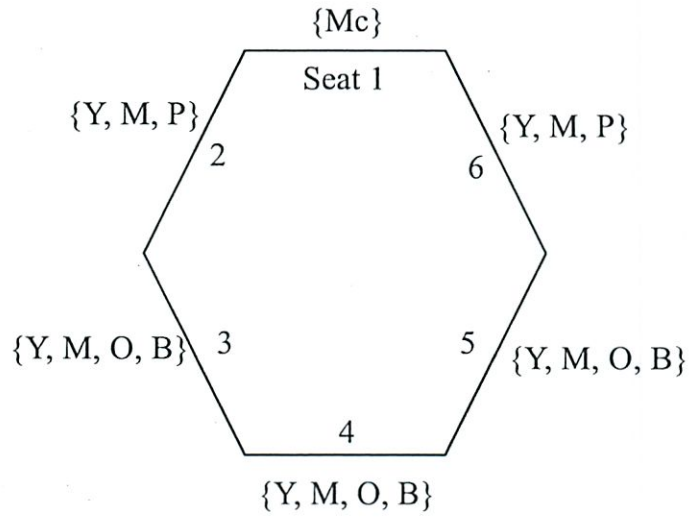
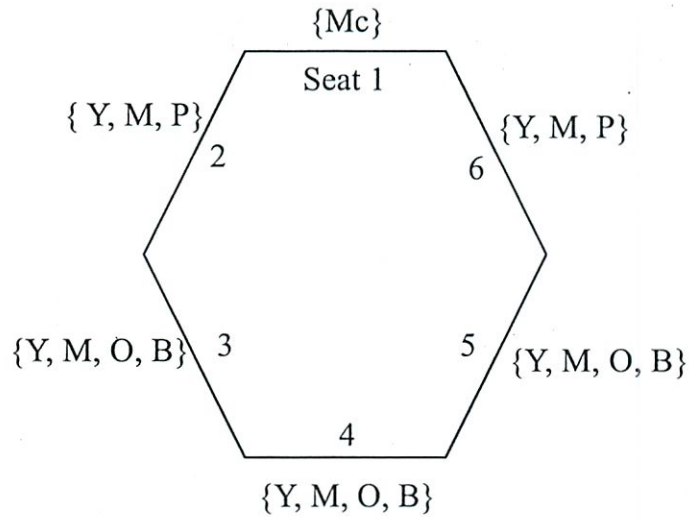
Now you try to use depth-first search *and constraint propagation through domains reduced to size 1*. You break ties by choosing the left-most in this order:

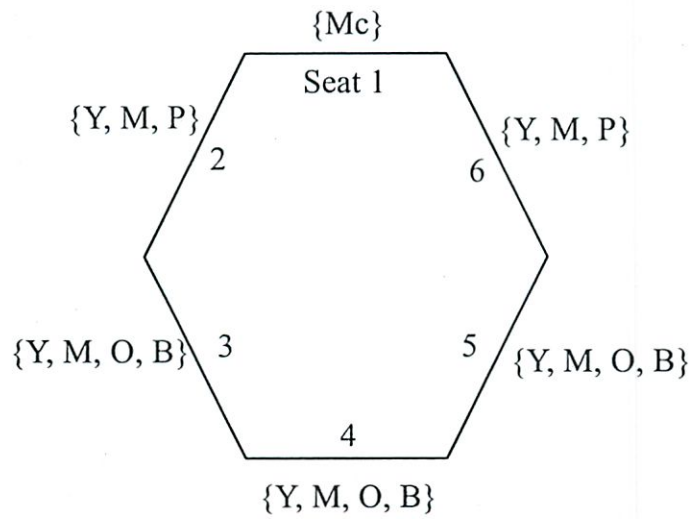
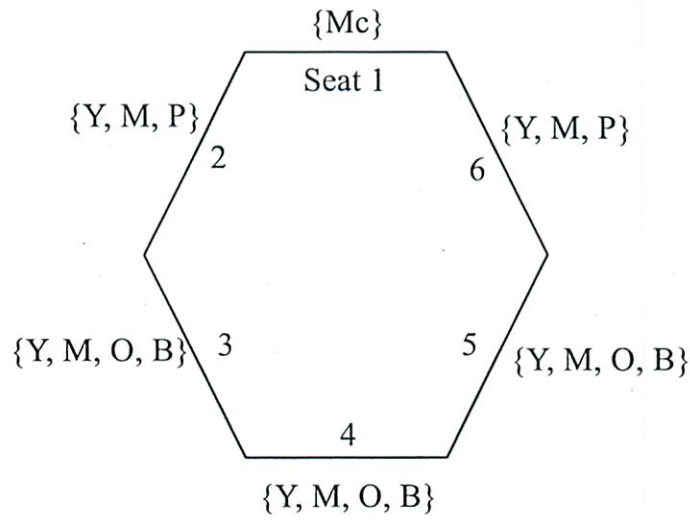
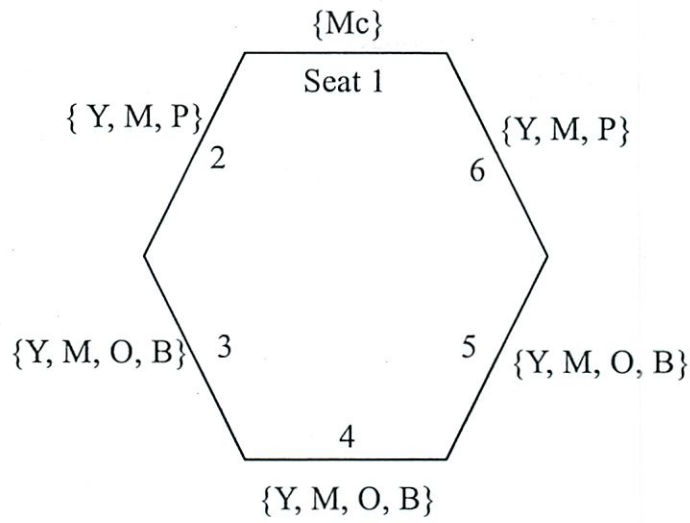
You (Y), Moose (M), McCain(Mc), Palin (P), Obama (O), Biden (B)

Show the the *full* search tree (up until you find a solution), starting with the **same pre-reduced domains**. You might want to work this out on a copy at the end of the quiz..

1	Mc
2	
3	
4	
5	
6	







Lab 3

From 6.034 Fall 2011

Contents

- 1 Game search
 - 1.1 Playing the game
 - 1.2 The code
 - 1.2.1 ConnectFourBoard
 - 1.2.2 Other Useful Functions
 - 1.2.3 Writing your Search Algorithm
 - 1.2.3.1 "evaluate" Functions
 - 1.2.3.2 Search Functions
 - 1.2.4 Creating a Player
 - 1.3 Just win already!
 - 1.4 Alpha-beta search
 - 1.4.1 Hints
 - 1.5 A better evaluation function
 - 1.6 Advice
 - 1.7 IMPORTANT NOTES
 - 1.8 Tournament
- 2 Survey

This problem set is due Friday, October 14th at 11:59 PM. If you have questions about it, ask the list 6034tas@csail.mit.edu.

To work on this problem set, you will need to get the code, much like you did for earlier problem sets.

- You can view it at: <http://web.mit.edu/6.034/www/labs/lab3/>
- Download it as a ZIP file: <http://web.mit.edu/6.034/www/labs/lab3/lab3.zip>
- Or, on Athena, attach 6.034 and copy it from `/mit/6.034/www/labs/lab3/`.

Your answers for the problem set belong in the main file `lab3.py`.

Game search

This problem set is about game search, and it will focus on the game "Connect Four (http://en.wikipedia.org/wiki/Connect_Four)". This game has been around for a very long time, though it has been known by different names; it was most recently released commercially by Milton Bradley.

A board is a 7x6 grid of possible positions. The board is arranged vertically: 7 columns, 6 cells per column, as follows:

	0	1	2	3	4	5	6
0	*	*	*	*	*	*	*
1	*	*	*	*	*	*	*
2	*	*	*	*	*	*	*
3	*	*	*	*	*	*	*
4	*	*	*	*	*	*	*
5	*	*	*	*	*	*	*

Two players take turns alternately adding tokens to the board. Tokens can be added to any column that is not full (ie., does not already contain 6 tokens). When a token is added, it immediately falls to the lowest unoccupied cell in the column.

The game is won by the first player to have four tokens lined up in a row, either vertically:

Oh yeah - I was thinking about doing this
- Simple game


```

0 1 2 3 4 5 6
0
1
2
3  O   X
4  O   X
5  O   X

```

horizontally:

```

0 1 2 3 4 5 6
0
1
2
3
4
5 X X X O O O O

```

or along a diagonal:

```

0 1 2 3 4 5 6
0
1
2  O   O
3  O   X X
4  O   X X
5  O   X X X

```

```

0 1 2 3 4 5 6
0
1
2  X   O O
3  X   X X
4  O   X X
5  X   O O X

```

Playing the game

You can get a feel for how the game works by playing it against the computer. For example, by uncommenting this line in lab3.py, you can play white, while a computer player that does minimax search to depth 4 plays black.

```
run_game(basic_player, human_player)
```

For each move, the program will prompt you to make a choice, by choosing what column to add a token to.

The prompt may look like this:

```

Player 1 (O) puts a token in column 0
0 1 2 3 4 5 6
0
1
2
3
4
5 O
Pick a column #: -->

```

In this game, Player 1 just added a token to Column 0. The game is prompting you, as Player 2, for the number of the column that you want to add a token to. Say that you wanted to add a token to Column 1. You would then type '1' and press Enter.

The computer, meanwhile, is making the best move it can while looking ahead to depth 4 (two moves for itself and two for you). If you read down a bit farther in the lab3.py file (or farther into this lab writeup), we'll explain how to create new players that search to

arbitrary depths.

The code

Here's an overview of the code in the various lab files that you may need to work with. The code contains inline documentation as well; feel free to read it.

ConnectFourBoard

`connectfour.py` contains a class entitled `ConnectFourBoard`. As you might imagine, the class encapsulates the notion of a Connect Four board.

`ConnectFourBoard` objects are immutable. If you haven't studied mutability, don't worry: This just means that any given `ConnectFourBoard` instance, including the locations of the pieces on it, will never change after it's created. To make a move on a board, you (or, rather, the support code that we provide for you) create a new `ConnectFourBoard` object with your new token in its correct position. This makes it much easier to make a search tree of boards: You can take your initial board and try several different moves from it without modifying it, before deciding what you actually want to do. The provided `minimax` search takes advantage of this; see the `get_all_next_moves`, `minimax`, and `minimax_find_board_value` functions in `basicplayer.py`.

6.005 this week!

So, to make a move on a board, you could do the following:

```

>>> myBoard = ConnectFourBoard()
>>> myBoard
0 1 2 3 4 5 6
0
1
2
3
4
5

>>> myNextBoard = myBoard.do_move(1)
>>> myNextBoard
0 1 2 3 4 5 6
0
1
2
3
4
5
X

>>> myBoard # Just to show that the original board hasn't changed
0 1 2 3 4 5 6
0
1
2
3
4
5

```

row, col
→ ↑

There are quite a few methods on the `ConnectFourBoard` object. You are welcome to call any of them. However, many of them are helpers or are used by our tester or support code; we only expect you to need some of the following methods:

- `ConnectFourBoard()` (the constructor) -- Creates a new `ConnectFourBoard` instance. You can call it without any arguments, and it will create a new blank board for you.
- `get_current_player_id()` -- Returns the player ID number of the player whose turn it currently is.
- `get_other_player_id()` -- Returns the player ID number of the player whose turn it currently isn't.
- `get_cell(row, col)` -- Returns the player ID number of the player who has a token in the specified cell, or 0 if the cell is currently empty.
- `get_top_elt_in_column(column)` -- Gets the player ID of the player whose token is the topmost token in the specified column. Returns 0 if the column is empty.
- `get_height_of_column(column)` -- Returns the row number for the highest-numbered unoccupied row in the specified column. Returns -1 if the column is full, returns 6 if the column is empty. NOTE: this is the row index number not the actual "height" of the column, and that row indices count from 0 at the top-most row down to 5 at the bottom-most row.
- `do_move(column)` -- Returns a new board with the current player's token added to `column`. The new board will indicate that it's

now the other player's turn.

- `longest_chain(playerid)` -- Returns the length of the longest contiguous chain of tokens held by the player with the specified player ID. A 'chain' is as defined by the Connect Four rules, meaning that the first player to build a chain of length 4 wins the game. *Clever - how to calculate!*
- `chain_cells(playerid)` -- Returns a Python set containing tuples for each distinct chain (of length 1 or greater) of tokens controlled by the current player on the board.
- `num_tokens_on_board()` -- Returns the total number of tokens on the board (for either player). This can be used as a game progress meter of sorts, since the number increases by exactly one each turn.
- `is_win()` -- Returns the *player ID number* of the player who has won, or 0.
- `is_game_over()` -- Returns true if the game has come to a conclusion. Use `is_win` to determine the winner.

Note also that, because ConnectFourBoards are immutable, they can be used as dictionary keys and they can be inserted into Python `set()` objects.

Other Useful Functions

There are a number of other useful functions in this lab, that are not members of a class. They include the following:

- `get_all_next_moves(board)` (`basicplayer.py`) -- Returns a generator of all moves that could be made on the current board
- `is_terminal(depth, board)` (`basicplayer.py`) -- Returns true if either a depth of 0 is reached or the board is in the game over state.
- `run_search_function(board, search_fn, eval_fn, timeout)` (`util.py`) -- Runs the specified search function with iterative deepening, for the specified amount of time. Described in more detail below.
- `human_player` (`connectfour.py`) -- A special player, that prompts the user for where to place its token. See below for documentation on "players".
- `count_runs()` (`util.py`) -- This is a Python Decorator callable that counts how many times the function that it decorates, has been called. See the decorator's definition for usage instructions. Can be useful for confirming that you have implemented alpha-beta pruning properly: you can decorate your evaluator function and verify that it's being called the correct number of times.
- `run_game(player1, player2, board = ConnectFourBoard())` (`connectfour.py`) -- Runs a game of Connect Four using the two specified players. 'board' can be specified if you want to start off on a board with some initial state.

they do a lot of the work for

Writing your Search Algorithm

"evaluate" Functions

Static Evaluators

In this lab, you will implement two evaluation functions for the minimax and alpha-beta searches: `focused_evaluate`, and `better_evaluate`. Evaluate functions take one argument, an instance of `ConnectFourBoard`. They return an integer that indicates how favorable the board is to the current player.

The intent of `focus_evaluate` is to simply get your feet wet in the wild-and-crazy world of evaluation functions. So the function is really meant to be very simple. You just want to make your player win more quickly, or lose more slowly.

The intent of `better_evaluate` is to go beyond simple static evaluation functions, and getting your function to beat the default evaluation function: `basic_evaluate`. There are multiple ways to do this but the most-common solutions involve knowing how far you are into the game at any given time. Also note that, each turn, one token is added to the board. There are a few functions on `ConnectFourBoard` objects that tell you about the tokens on the board; you may be able to use one of these. You can look at the source for the evaluation function you are trying to beat by looking at `def basic_evaluate(board)` in `basicplayer.py`.

Search Functions

As part of this lab, you must implement an alpha-beta search algorithm. Feel free to follow the model set by `minimax`, in `basicplayer.py`.

Your `alpha_beta_search` function must take the following arguments:

- `board` -- The `ConnectFourBoard` instance representing the current state of the game
- `depth` -- The maximum depth of the search tree to scan.
- `eval_fn` -- The "evaluate" function to use to evaluate board positions

And optionally it takes two more function arguments:

- `get_next_moves_fn` -- a function that given a board/state, returns the successor board/states. By default `get_next_moves_fn` takes on the value of `basicplayer.get_all_next_moves`
- `is_terminal_fn` -- a function that given a depth and board/state, returns True or False. True if the board/state is a terminal node, and that static evaluation should take place. By default `is_terminal_fn` takes on the value of `basicplayer.is_terminal`

You should use these functions in your implementation to find next board/states and check termination conditions.

The search should return the *column number* that you want to add a token to. If you are experiencing massive tester errors, make sure that you're returning the column number and not the entire board!

TIP: We've added a file called `tree_searcher.py` to help you debug problems with your `alpha_beta_search` implementation. It contains code that will test your alpha-beta-search on static game trees of the kind that you can work out by hand. To debug your alpha-beta-search, you should run: `python tree_searcher.py`; and visually check the output to see if your code return the correct expected next moves on simple game trees. Only after you've passed `tree_searcher` then should you go on and run the full tester.

Creating a Player

In order to play a game, you have to turn your search algorithm into a *player*. A player is a function that takes a board as its sole argument, and returns a number, the column that you want to add a piece to.

Note that these requirements are quite similar to the requirements for a search function. So, you can define a basic player as follows:

```
def my_player(board):
    return minimax(board, depth=3, eval_fn=focused_evaluate, timeout=5)
```

or, more succinctly (but equivalently):

```
my_player = lambda board: minimax(board, depth=3, eval_fn=focused_evaluate)
```

However, this assumes you want to evaluate only to a specific depth. In class, we discussed the concept of *iterative deepening*. We have provided the `run_search_function` helper function to create a player that does iterative deepening, based on a generic search function. You can create an iterative-deepening player as follows:

```
my_player = lambda board: run_search_function(board, search_fn=minimax, eval_fn=focused_evaluate)
```

Just win already!

You may notice, when playing against the computer, that it seems to make plainly "stupid" moves. If you gain an advantage against the computer so that you are certain to win if you make the right moves, the computer may just roll over and let you win. Or, if the computer is certain to win, it may seem to "toy" with you by making irrelevant moves that don't change the outcome.

This isn't "stupid" from the point of view of a minimax search. If all moves lead to the same outcome, why does it matter which move the computer makes?

This isn't how people generally play games, though. People want to win as quickly as possible when they can win, and lose slowly so that their opponent has several opportunities to mess up. A small change to the basic player's static evaluation function will make the computer play this way too.

Our player currently does not

- In `lab3.py`, write a new evaluation function, `focused_evaluate`, which prefers winning positions when they happen sooner and losing positions when they happen later.

It will help to follow these guidelines:

- Leave the "normal" values (the ones that are like 1 or -2, not 1000 or -1000) alone. You don't need to change how the procedure evaluates positions that aren't guaranteed wins or losses.
- Indicate a certain win with a value that is greater than or equal to 1000, and a certain loss with a value that is less than or equal to -1000.
- Remember, `focus_evaluate` should be very simple. So don't introduce any fancy heuristics in here, save your ideas for when you implement `better_evaluate` later on.

Alpha-beta search

The computerized players you've used so far would fit in well in the British Museum - they're evaluating all the positions down to a certain depth, even the useless ones. You can make them search much more efficiently by using alpha-beta search.

It may help to read the appropriate chapter of the text (<http://courses.csail.mit.edu/6.034f/ai3/ch6.pdf>) if you're unclear on the details of alpha-beta search.

- Write a procedure `alpha_beta_search`, which works like `minimax`, except it does alpha-beta pruning to reduce the search space.
- You should always return a result for a particular level as soon as alpha is greater than or equal to beta.
- `lab3.py` defines two values `INFINITY` and `NEG_INFINITY`, equal to positive and negative infinity; you should use these for the initial values of alpha and beta, as discussed in class.

This procedure is called by the player `alphabeta_player`, defined in `lab3.py`.

Your procedure will be tested with games and trees so don't be surprised if the input doesn't always look like a connect 4 board.

Hints

In class, we describe alpha-beta in terms of one player maximizing a value and the other person minimizing it. However, it will probably be easiest to write your code so that each player is trying to *maximize* the value from their own point of view. Whenever they look forward a step to the other player's move, they *negate* the resulting value to get a value for their own point of view. This is how the minimax function we provide works.

You will need to keep track of your range for alpha-beta search carefully, because you need to negate this range as well when you propagate it down the tree. If you have determined that valid scores for a move must be in the range `[3, 5]` -- in other words, `alpha=3` and `beta=5` -- then valid scores for the other player will be in the range `[-5, -3]`. So if you use this negation trick, you'll need to propagate alpha and beta like this in your recursive step:

```
newalpha = -beta
newbeta = -alpha
```

This is more compact than the representation we were using on the board in lecture, and it captures the insight that the player evaluates its opponent's choices just as if the player was making those choices itself.

Ask a TA if you are confused by this.

A common pitfall is to allow the search to go beyond the end of the game. So be sure to use `is_terminal_fn` to determine the end.

A better evaluation function

This problem is going to be a bit different. There's no single right way to do it - it will take a bit of creativity and thought about the game.

Your goal is to write a new procedure for static evaluation that outperforms the `basic_player` one we gave you. It is evaluated by the test case `run_test_game_1`, which plays `your_player` against `basic_player` in a tournament of 4 games. Clearly, if you just play `basic_player` against itself, each player will win about as often as it loses. We want you to do better, and design a player that wins at least 2 times more than it loses in the four games.

We should hardly need to state that your solutions should win legitimately, not by somehow interfering with the other player. This isn't Spassky vs. Fischer (http://en.wikipedia.org/wiki/World_Chess_Championship_1972).

An additional warning, regarding this particular subject matter: There is a fair bit of published research and online information about "Connect Four", and about search algorithms in general in Python. You're welcome to read any documents that you'd like, as long as you cite them by including a comment in the relevant part of your code. However, you may not re-use complete third-party implementations of search algorithms, and you may not write code that access online resources (other than the 6.034 test server) while it is executing. The tester can't easily test for these, so it doesn't. The TA's will be looking over your code, though, and they will not be amused if they find a student using someone else's work in their lab.

Relatedly, while it would be possible to pass the lab using `human_player`, we can't give you credit for doing so. The TA's will be

amused if they find out that you did this, but they'll have to revoke credit for the relevant portions of the lab.

Advice

In an evaluation function, simpler can be better! It is much more useful to have time to search deeper in the tree than to perfectly express the value of a position. This is why in many games (not Connect Four) it takes some effort to beat the simple heuristic of "number of available moves"; you can't get much simpler than that and still have something to do with winning the game.

If you write a very complicated evaluation function, you won't have time to search as deep in the tree as `basic_evaluate`. Keep this in mind.

IMPORTANT NOTES

After you've implemented `better_evaluate`, please change the line in your `lab3.py` From

```
better_evaluate = memoize(basic_evaluate)
```

to

```
better_evaluate = memoize(better_evaluate)
```

The original setting was set to allow you play the game earlier on in the lab, but become unnecessary and incorrect once you've implemented your version of `better_evaluate`.

TIP: To save time, change the `if True` to `if False` on line 308 of `tests.py`. Disabling this test will skip the game test that usually takes a few minutes to finish. But be sure to remember to turn it back on when you have passed all your other offline tests!

```
# Set this if-guard to False temporarily disable this test.
if True:
    make_test(type = 'MULTIFUNCTION',
              getargs = run_test_game_1_getargs,
              testanswer = run_test_game_1_testanswer,
              expected_val = "You must win at least 2 more games than you lose to pass this test",
              name = 'run_test_game'
              )
```

Tournament

Please set "COMPETE" to True if you would like to participate in a tournament. If there is enough interest/capable AIs then we will face you off against your fellow classmates and perhaps some undisclosed prize. The first round will cull entries which can't beat the `basic_player` enough times so please set "COMPETE" to False if you haven't managed to pass that test.

Survey

Please answer these questions at the bottom of your `lab3.py` file:

- How many hours did this problem set take?
- Which parts of this problem set, if any, did you find interesting?
- Which parts of this problem set, if any, did you find boring or tedious?

(We'd ask which parts you find confusing, but if you're confused you should really ask a TA.)

When you're done, run the tester as usual to submit your code.

Retrieved from "http://ai6034.mit.edu/fall11/index.php?title=Lab_3"

- This page was last modified on 30 September 2011, at 20:52.
- *Forsan et haec olim meminisse iuvabit.*

Reading Connect 4 Book

Playing

- I'm not very strategic

- Don't think a lead

Minimax pre built

Do the α, β part now

- static eval later

Where is static eval? for pre-built one
Hard to see what is going on...

Oh search in that file

Do MC \checkmark 2/2

So forced eval is the static eval

- get board object

Based Basic is pre built - oh I see now

- return -1000 if lost

~~oh~~

- looks at the whole board - not a move makes sense.

②

Putting print board makes it really print!

Take longest chain = 10

And want pieces in center

Returns score

keep for now

do α, β

Copy min max into α, β to convert

Alpha-beta player uses α, β search w/ focused

So iterates over next moves

Why does minimax not work when I copy + paste it

and run it - diff functions!

Get-next-moves is a generator fn

- tries all 7 moves, if not invalid

- but how does new board change?

Is the actual move - but don't know where it's generated

Oh "yield"

(3)

Oh returns "Yields" ; and do move ;

So the column

So it returns a board w/ new move

(so much time is figuring out their code....)

So it then runs recursively on next depth

If val is better than current best val, save it

Return the best next move

So gets chain

- No just reports best upward.

Oh ^{-1 guess} ~~and~~ -1 to swap min/max levels
Oh now look how α, β supposed to work

- lecture notes no help - did all at once

And start at bottom - not top

Need to somehow get to bottom

- ~~the~~ they have progressive deepening?

Oh pseudo code in recitation notes

4

They don't check ^{max} depth

I need that

Or do I?

I think I need it depth decrements to 0

Then recurse down

No d/B

α should be large - starts $-\infty$
Maximizer - guaranteed lower bound

β Min - should be small - starts ∞
- guaranteed upper bound

$d \geq \beta$ ~~always~~ ^{cutoff!}

Minimizer determines β if $\leq d$ than min's parent (Max) than would never go there so min skip

Basicall $\beta < \alpha$ cut!

If Max sees $d > \beta$, cut

~~if so cut when $d > \beta$~~

5

Get max of min-value
or α, β - swapped

Then α is new max of alpha, v
max element

So $v = \max$ of all children

alpha is then max of parents' alpha or v result of children

So what is cutoff

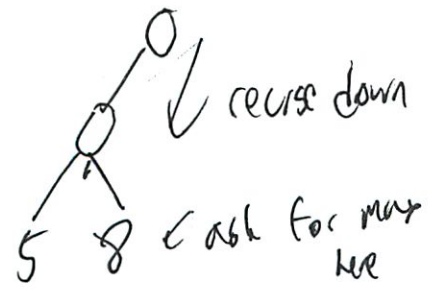
$\alpha \geq \beta$ is what we want always - right?
No this is cutoff condition

So when cutoff we say nothing better than α

Otherwise return best of children
new alpha

(The swapping thing is confusing)

Think through finding max



~~8~~ - now compare to pre ~~and~~ $(-\infty)$

New $\alpha = 8$

Now is $8 \geq \infty$ - No
So return 8 - that is best value

⑥

Just like normal

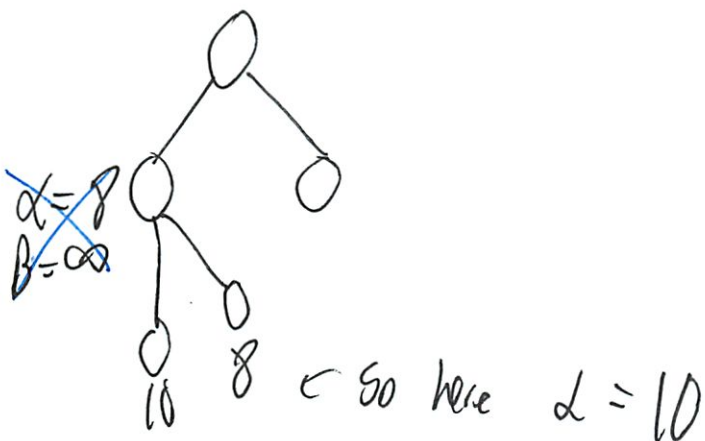
Now one with cutoff α

But need to do rest

Look through examples

Well now

Here put
 $\alpha = -\infty$
 $\beta = 8$
Since min picks here



Then returns 10 upward

So it has 8, 10

well - 8, -10

picks max - 8

Saves ~~at~~ $\alpha = \beta$

Bubbles up and switches $\alpha = 8$ ← since min picks first
 $\beta = \infty$

7

Oh missed cutoff.

When l_0 was evaled we had

$$\alpha = -\infty$$

$$B = 6 \quad \leftarrow \text{so this was passed along}$$

When $l_0 = \alpha$

$$\text{Now } l_0 \geq 6$$

so returned alpha $\rightarrow l_0$

Oh cut it off earlier than I had in my code

Good find

So cut since Min will not want to get l_0
When can get 8

(I'm getting min/max switch)

Test - getting a lot of pruning, good

But am I saving α, B handoff right?

- return α as best if best

But do I update this value? on return

- Yes pass along

- but does it do this or instance values?

⑧
So return tuple and update
Get int not iterable error...
Oh did not change all
Can't do that because end return...
Minimax should only return at end once!
I return all the time...

There is no progressive deepening
Starts at max depth
We did not do depth check last time...
Why not???

Oh 2 minimax fns!

- a master and recursive fn

Oops...

- but that is just for printing... and check for depth
Can do master

9

Ok I am evaluating properly

Do I need to pass α, β along?

Yeah since they are always \rightarrow or \leftarrow

Need to be passed along

I passed some tests
— the right ones!

Tester ~~failed~~ crashed

But I think I passed

Don't pass α, β along

Try other tests

— tree search

(But still not convinced did α, β right)

[It does lots of cutoffs

— try other way ...

Seems to be going faster!

But just proving all on the last level

(10)

I think proper text never returns

Email in

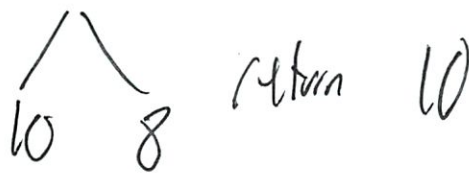
So its propagate down always ✓

But do you propagate up when don't return value

No - its just return value

We don't care about local state

like my



Then it picks between it

Yeah propagate up is stupid

But w/o it my code freezes or looks like it

Was it before I was returning wrong results

Returning too fast

Then what is wrong w/ what I have?

Email return is just prop down

11

Seems to crash in special case

Test 12

X X 0

- Others pass

Let me come back to it

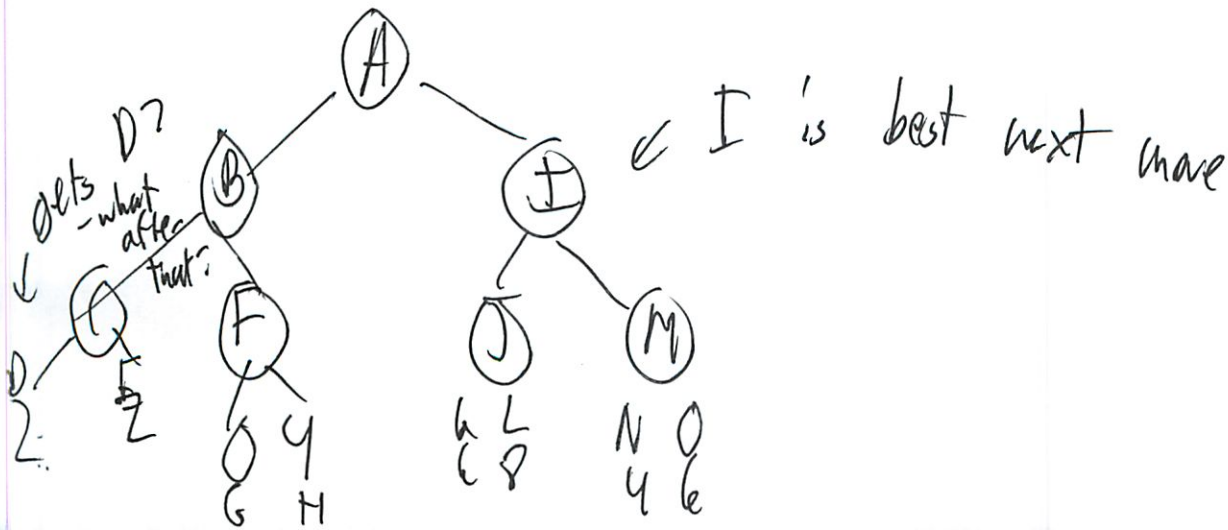
So now try want me to do better - evaluate

min - test - tree - search

Node has no attribute 'is_game_over' in basic player
is terminal

Ok - tech error

Now I fast fails



(12)

It proves for some reason

So $D \geq \infty$? yes

$0 \geq \infty$ No

Convert D to y

Ok that worked

Now for column it wants a #

Why did not error before?

Oh convert move!

No ~~in~~ compared wrong thing whole time!

~~We~~ don't care node name!

Why did val return blank? for C

Should be the d

Why did I not realize that long before!

Fixed!

That could be why that test had failed

-test 10 I think

Still crashing on # 13 now - or hanging x x 0

13

Online test crashing/hanging

The bum part of this p-set

Think that is enough for now

Still need 2 SEs

And see tests

Run it visually to see if hung

Crashes on test 13

X X O

So this is win 2 games

So its just test 13 - if disable we are fine

So do rest tomorrow

Some player fails

I think needs your player

That is all there - works manually ...

(14)

So its progressive deepening does not work
Emailed in

So it was needed Py 2.7

10/10

Ok so back to problem 1 week later
Revert the changes I made to core files...

So progressive deepening works

Now just need my static eval

And then I think we are good!

Reviewed diff - did not change much...

I think all I need is a better static eval

better eval

- Should test online too
- Python does not let you type input in?
- Oh ya have to add it
- and need to comment out before test
- Online tester (In Idle taking forever!)

tie - is Win = 0?

So online tests all pass except winning 2x on your player
So focused eval not tested at all
↳ better-eval

(15)

Basically all that is left is beat the basic player with my ~~the~~ better - player twice ...

Connect 4 tactics

- make your move more valuable than opponent
 - gives them no strong cell or wastes (not static)
- Forcing combos
- avoid threats (I could build this)
- 2 center cells in 2nd, 4th, 6th row bad for red
 - red 1st, black 2nd
- don't care about strategy, - just count threats and assign them a point value
- could do many things faster
 - Red wants threats on 3rd, 5th row

16

Vertical threats bad

Red odd horizontal threats
if 1st player diagonal
Even row

Threats better low on board

Central column are better

Diagonals toward center are better

Chap 4/1

Red wants odd row threats
Black even

So static eval + your threats - other threats

Adjacent threats twice as good

2nd + 5th car threats same as two odd-thread columns

Minor threats - cells that if captured would be major threats

(I don't need to see ahead as much - PC will do - but only to certain depth)

17

I only need a good enough accuracy anyway

So build a threat center ...

- Ens - longest chain
- no chain cells command
- nice abstraction Ens ☺

high scores good

how find first mover

Odd rows are as in book

5	0
4	1
3	2
2	3
1	4
book	5
	Python

Oh it does not say where Threat is - need to do more calc

Test one first

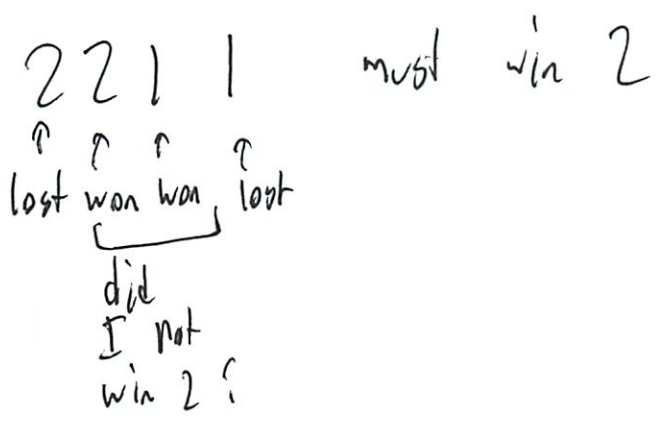
Calculating ~~the~~ where threats are is not easy!

- but might have to do - or at least horiz or diag

(18)

Calc row or horiz threat

Still lost on basic



Run another test w/ scoring diff threats differently

Next - diagonal threats towards center are better

- adj threats good

- odd rows have already right?

- is odd threat actually better - or just leading up to?

- for threat + MT should actually calc where it is so don't just take boxed in 3 chains

↑ actually next item

2 2 1 1 again

(19)

No win 2 more games than you lose

$$\text{So } 2-2 = 0$$

$$3-1 = 2$$

So win one more

Do that other part

Actually look for threat

perhaps make sub chap

So vertical chain - can only check up
- either open or other candidate

If black even can go good

Does chain always return left to right?

It always goes $c \rightarrow l$

Damn I switched row and col!

Lots of Cuss - got tired around
Print board!

2211

(20)

Still no!

Why not! I am now only looking for actual threats!
So I did not copy it over correctly?

Checked it out - not returning correctly?

We're 0

lots diagonal

(Could go back even + add needed to fill - might not be right though...)

Left + right not working?

No works fine!

Perhaps close + reopen files?

Nope 22 11

More heuristics to get started

Add back the two in a row
- test if open next to it

22 11

(21)

What next?

Could do diagonal

Or test the 2s to see if next space is empty or is _{over}

Why is it no good?

Oh my memorization has been bad!

But returning wrong ans for player

- when I mem

So mixing current player?

Retest w/o memorize

(Takes forever to test)

I emailed in to TAs

Oh got diff result!

1	0	2	1
↑	↑	↑	↑
won	tie	lost	lost

Worse !?!

Or are my scorings wrong

I am getting confused X, O
max? min?

22

Is it always trying to min?

Am I going wrong way?

Reversed static eval

Or max of abs values?

- no then other one would be wrong?

Why does it pick lowest one?

Minimax does that as well - did not write

2 1 0 1 ← what did I change?

↑ ↑ ↑ ↑
lost lost tie lost

Worse - oh flipping the pts

Hmm - score makes tests fail

But ± on threats does not fail tests

- no threats here: basic boards

i pump up scores

Anything else easy?
(Ties can't poorly!)

2 1 0 1

(23)

But ∇ Fixed score:

5% bias to opponent's rows - to be a bit more defensive?
Rejiggered pts.

1 2 2 1
↑ ↑ ↑ ↑
won won won won

Wait - that should do it!

No 1 2 1 2 is goal

So won won lost lost

I think min is good - other wise it would lose earlier

No when player 1 it maximizes correctly...

No still minimizes or

No always minimizes 1 or 2

No clue what to do next!

The min thing - is it a problem?

- since test passes!

(29)

Can mess w/ params -15 sec fine cut!

1020

worse! I'm doing something wrong!

So should min when player 2
- since we do - values

Emailed Issue

Got email from Erik

10/14

- first player id is always 1
- remember time matters!
- turn memorize back on

~~Focus want ASAP~~

Focused want ASAP

We are trying to do evaluate better

What is exponent not linear constants

i does he mean 4^5 is faster than $4 \cdot 4 \cdot 4 \cdot 4 \cdot 4$?

since can do exponentiate

But where do I do that

~~Copy~~

Do they mean instead of adding points:

1 0 2 | ~~pts~~ wins

one more win, than tie or one more loss

1 2 1 2 is goal

So fixing stuff broke it more

So 1 win
1 tie
2 losses

Worse than basic!

Try a new approach:

Cassandra's mirroring?

- no fn for getting last move

My fns want more pts for making chains

- it picks longest!

Try simple chain^{that} count - all same pts

26

What else besides threats:

i fix 2 in a row.

- check 2 in a row - not just 1?

What else?

1 0 2 1

Everyone says should be simple

+ fast

Get chain cells prob slow!

i only do for 1 player

and get rid of put tokens in center

So only do chains

2 2 2 2

Only your chains + no pieces in center

2 1 1 2

Darn! 2/2 each time

Goal 1 2 1 2

So don't do this on row 2!

(27)

Issue; take chains

Compute length.

Sum f^{length}

So try this \pm

Not filter out the 1s

- causes it to win fast

Should be same speed as my other one

So running 2

back to regular check threat

- just one
- no columns

2 1 1 2

Issue's for \pm

2 1 1 2

Unless it's not working:

Test just for me Isaac's

2 1 2 2

↑ lost one more

28

Then Erik wrote me back

recommended the squared value too

So test my method by ² all values
→ 2 2 2 2

Eval as expect on test cases

- do you mean it passes test cases

And how would I make it faster?

How about cubed? And bigger win/lose values

Oh win!

2222 means 2nd player always wins

- any weakness here

Perhaps look at games

It misses an opponent 3 in a row



Very weird, emailed Erik

Not learning much w/ G+V

29

0222 online

~~me~~ ^{win}
1 2 1 2 goal darn

Erick says its b/c my eval sees it

lost 8 turns ahead ~~and~~ and gave up
? So no win/loss thing?

- if take that out loses a lot of best cases

Instructions say to use # of tokens on board

? Subtract from score?

Now getting 2222 - what was that for?

- was before current move

- no pieces in center but 3ed?

On moves

0222

So did I have a win and then erase it?

The win faster go on the initial return

02222

30

- Chase

A bunch of people helping me

Can't figure it out - lots of discussion

Trying min max

Win!!

And Issues w/ Win earlier

0 1 1 2

Trying Chase d, B Eclipse

0 1 1 2

evaluate better evaluate

2 2 1 0

i 2 i 2 goal

very different!

but both 2 wins

Together the file wins + passes test!

Both my d, B and eval suck!

So basically my bad A, B or invalidates all my eval work

31

Review my a, B

- appears to work
- changed to return a , not ~~best~~ best val

So minimax online gives - w/ Issas scoring

2	2	2	1
1	2	1	2

- don't know if matters
- can make an occasional diff

So that fails

Test with new a, B

0	1	1	2
---	---	---	---

Eliza
← same as w/ Chase

And w/ my old scoring ^{minimax} _{TABLE}

- (should hopefully be same as above

- time matters ...)

I think my a, B was wrong - but via cheating - hopefully w/ fixed a, B this will work too

But then Chase's eval should work
Take parts of Chases eval

So testing w/ diff center values (.5)

and add row (low better) = 3

0	2	2	1
1	2	1	2

← so ~~same~~ ^{Similar} as his eval - but don't want

(32)

Will see that and then look if more needed

But then why does his L, B search work with it
as a package

And why is the difference so meaningful

I'm not really thinking - more trying

Test cedue line / row vales

$$\begin{array}{cccc} 0 & 2 & 2 & 2 \\ \substack{1 \\ ?} & \substack{1 \\ ?} & \substack{1 \\ ?} & \substack{2 \\ ?} \end{array} \text{ better!}$$

I think give up

But reason wins w/ minimax is L, B is wrong
or takes diff time!

Michael E Plasmeier

From: Erek Speed <espeed@MIT.EDU>
Sent: Thursday, October 13, 2011 2:36 AM
To: Michael E Plasmeier
Subject: Re: FW: Having trouble coming up with a good static eval

Follow Up Flag: Follow up
Flag Status: Flagged

Right, I actually forgot I was going to get back to you on that.

The only odd thing is that your `focused_evaluate` is your `better_evaluate` but the tests don't do a good job of isolating the wanted behavior of `focused_evaluate` so it's not really a big deal.

Your alpha-beta seems fine though on first glance, so I assume you just need to work out your static evaluator.

I know you were hoping for some help but there's not much to say without giving too much away when it comes to strategies.

Things to consider:

Speed (yours went 5 or 6 ply deep on turn one which is pretty good) using exponents over linear constants ?

Also, the first player has id 1 if you want something a bit more robust than checking a common placement. I think this actually made you draw instead of lose when I made the change. nice

Also, don't forget to turn memoize back on for real games, it let's you search deeper. fine matter

I don't know what your process has been so far in building your evaluator, but I definitely recommend trying to distill down concepts as much as possible.

good luck.

What does this mean?

Erek

2011/10/13 Michael E Plasmeier <theplaz@mit.edu>:

> Any luck finding something I did wrong??? Or suggestions for what else I can do for a good static eval?

>

> Thanks -Plaz

>

> -----Original Message-----

> From: Erek Speed [mailto:espeed@MIT.EDU]

> Sent: Tuesday, October 11, 2011 11:31 AM

> To: Michael E Plasmeier

> Subject: Re: FW: Having trouble coming up with a good static eval

>

> I'll look over your code in general later to see if it has any flaws beyond the specific evaluator. Other than that, ends up doing that due to the hash on boards not taking into account who's turn it is.

> It doesn't effect actual games but it comes up in testing like that.

>

> I should add an faq.

>

> 2011/10/11 Michael E Plasmeier <theplaz@mit.edu>:

>> I need some static evaluator help! Why can't I win more than 2 games?

>>

>>

>>

>> Thanks so much!

>>

>>

>>

>> -Plaz

>>

>>

>>

>> From: Michael E Plasmeier

>> Sent: Tuesday, October 11, 2011 1:20 AM

>>

>> To: 6034tas@csail.mit.edu

>> Subject: RE: Having trouble coming up with a good static eval

>>

>>

>>

>> Are we supposed to be picking the lowest static value? Only for player 2?

>> It seems to do that for when I am player 1 or 2.

>>

>>

>>

>> But all my other tests are right. (16/17)

>>

>>

>>

>> -Michael

>>

>>

>>

>> From: Michael E Plasmeier

>> Sent: Tuesday, October 11, 2011 12:41 AM

>> To: 6034tas@csail.mit.edu

>> Subject: RE: Having trouble coming up with a good static eval

>>

>>

>>

>> Turning off memorize got me a different result 1 0 2 1, so at least I

>> know it is working (changing) now..

>>

>>

>>

>> -Michael

>>


```
>>
>>
>> From: Michael E Plasmeier
>> Sent: Tuesday, October 11, 2011 12:38 AM
>> To: 6034tas@csail.mit.edu
>> Subject: Having trouble coming up with a good static eval
>>
>>
>> So I have spent a few hrs building a static evaluator based around threats.
>> But the result has always been the same: 2, 2, 1, 1.
>>
>>
>>
>> Am I doing something wrong - like forgetting something so my
>> evaluator is not working right?
>>
>>
>> Also about memorize. I just realized we have to change it. But when
>> I turned it on and ran this:
>>
>> board_tuples = (( 0,0,0,0,0,0,0 ),
>>
>>     ( 0,0,0,0,0,0,0 ),
>>
>>     ( 0,0,0,0,0,0,0 ),
>>
>>     ( 0,2,2,1,1,2,0 ),
>>
>>     ( 0,2,1,2,1,2,0 ),
>>
>>     ( 2,1,2,1,1,1,0 ),
>>
>> )
>>
>> test_board_1 = ConnectFourBoard(board_array = board_tuples,
>>
>>     current_player = 1)
>>
>> test_board_2 = ConnectFourBoard(board_array = board_tuples,
>>
>>     current_player = 2)
>>
>> # better evaluate from player 1
>>
>> print "%s => %s" %(test_board_1, better_evaluate(test_board_1))
>>
>> # better evaluate from player 2
>>
>> print "%s => %s" %(test_board_2, better_evaluate(test_board_2))
>>
```

>>
>>
>> I got the same result for both player 1 and 2. When I turned it off,
>> I got the negative of the result for player 2 - which is what I want, right?
>>
>>
>>
>> -Michael
>

Michael E Plasmeier

From: Erek Speed <espeed@MIT.EDU>
Sent: Friday, October 14, 2011 8:56 PM
To: Michael E Plasmeier
Subject: Re: FW: Having trouble coming up with a good static eval

I mean that instead of using linear combinations you should use a combinations of exponents. So when you get your value back from your threat analysis, square it or similar instead of multiplying it by a constant. It should make better moves stand out. It might not be enough but is usually a correct addition to any other algorithm.

Because the tests don't start from a blank board it could certainly effect it that way. It really means that your better_evaluate isn't good enough. And whether you win or lose is a coin flip. (Although, if you can pass the test due to bugs, that's technically allowed.)

Your threat analysis code is pretty complicated so it's hard to say whether it's too slow or possibly has a bug.

Although, when you run it on test boards does it evaluate how you expect? I'll tweak it a bit as well.

It might also be a problem with not looking far enough ahead to make correct beginning moves so even if it is good it might not matter mid game. I'll spend some time going through your code though.

Erek

2011/10/14 Michael E Plasmeier <theplaz@mit.edu>:

> Hi,
>
> What do you mean by using "exponents over linear constants". I am guessing that means 4^5 is faster than $4*4*4*4*4$ since you can do fast exponentiation. But where would that help me?

>
> I fixed the player ID and memorize. But I am getting 1 0 2 1 now, instead of 1 0 2 0. I believe the goal is 1 2 1 2 which means that changed cost me to lose.

>
> Are there any specific concepts I should rethink?

>
> Thanks -Plaz

> -----Original Message-----

> From: Erek Speed [mailto:espeed@MIT.EDU]
> Sent: Thursday, October 13, 2011 2:36 AM
> To: Michael E Plasmeier
> Subject: Re: FW: Having trouble coming up with a good static eval

>
> Right, I actually forgot I was going to get back to you on that.

>
> The only odd thing is that your focused_evaluate is your better_evaluate but the tests don't do a good job of isolating the wanted behavior of focused_evaluate so it's not really a big deal.

>

G.034 Tutorial

10/17

Could do α, β very slowly

Some people have old key files

Don't change pre-filled variables

Did anyone win all 4?

Some people tied all 4

Good AIs track # pieces on board

- not true for Connect 4

Lot of confusion on focused evaluate

- tests always passed

- So when pc lost it will just put in slot 0

- or multiple ways to win, it will kill time to that predetermined win - even if other ways ~~passed~~

- So have win/loss score +/- the # of tokens left on the board

- So it wins as ASAP, loses as long as possible

②

But only works if your opponent plays perfect game
Could an AI take advantage of the opportunity to win

Constraint Propagation

CP w/ Backtracking

- brute first
- DFS

+ Forward Checking

- apply constraints to neighbors
- remove stuff that violates constraints

FC w/ Single Domain

- if only one left, check that one next

Arc

Does not know much about
One of most common?
Outside scope of class

3

Exams

Can make up criteria about propagating constraints
- so read carefully!

Quiz is next wed

L Games + Constraint propagation

Question: Make a jingle for a commercial

Rules

1. Must start w/ A

^{giving lot, result}
not a rule in general
but makes answers
more standard

2. Must end w/ A or F only

3. M7 \rightarrow E

4. A ^{can only be} ~~must~~ ~~have~~ ~~either~~ first or last

5. M4 \rightarrow E

Choards A \rightarrow F

8 Measures

④

So put these in a table

1	A				
2	B	C	D	E	F
3	B	C	D	E	F
4	E				
5	B	C	D	E	F
6	B	C	D	E	F
7	E				
8	A	F			

New rules

A $\xrightarrow{\text{transitions}}$ any

B \rightarrow E

C \rightarrow F

D \rightarrow A or E

E \rightarrow A or F

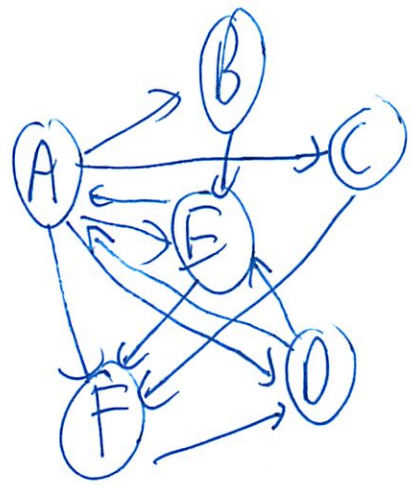
F \rightarrow D

No consecutive cards

5

So build DAG

No self loops

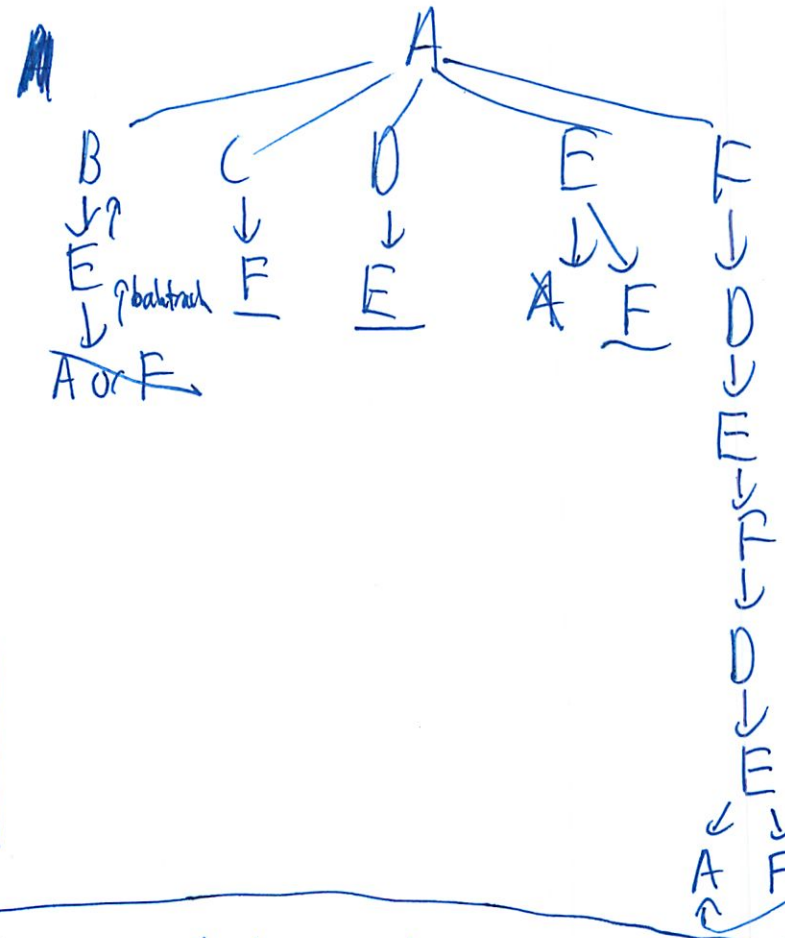


Backtracking w/ Forward Checking

- No singleton domains

↓
Order to check in

- 1
- 2
- 3
- 4
- 5
- 6
- 7
- 8



just pick 1

pick this one since lex. first

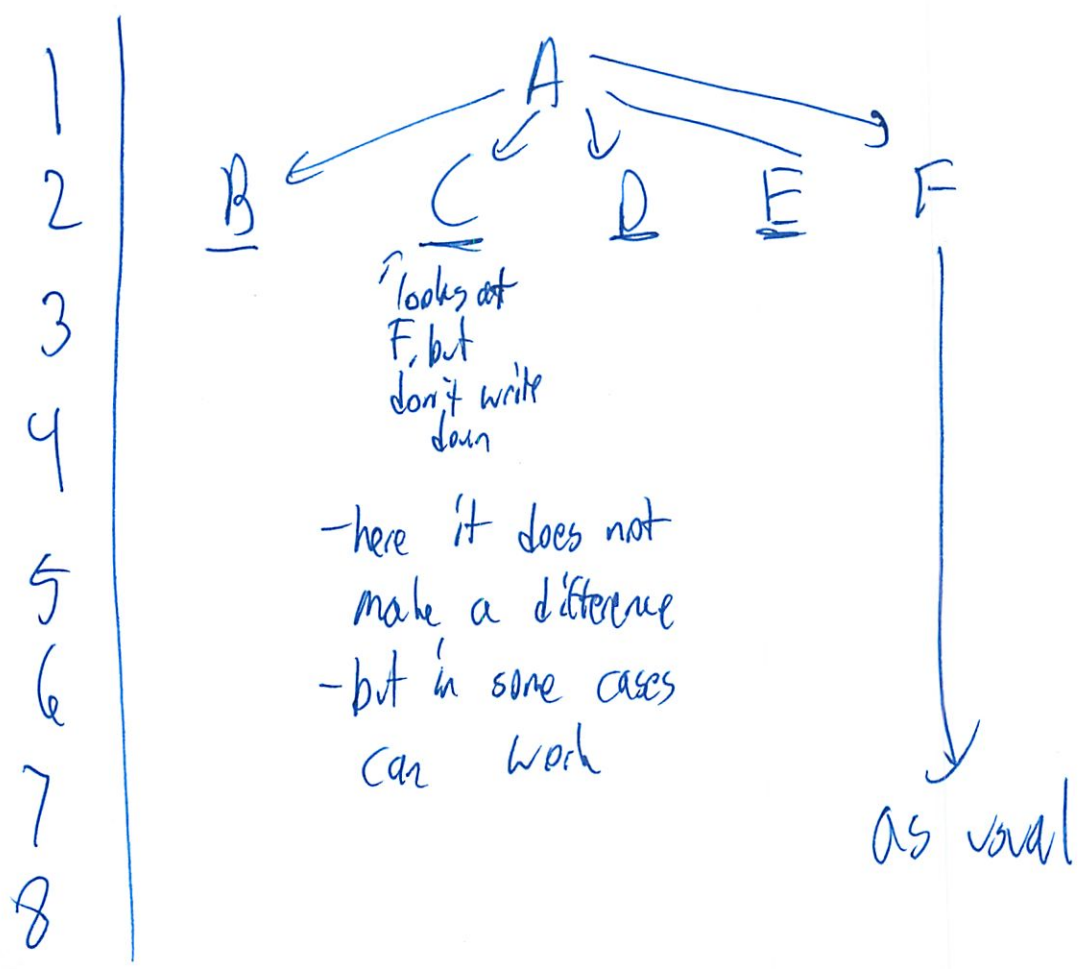
ones shall not even write here - should not have been added in 1st place

6

So like 2 types of constraints

- can do pre - about domain
- or transitions like next to
- can be given all at once
- or in two parts like here

Forward Checking w/ ~~the~~ Singleton domains



If did brute force, you would have all listed letters of domain (I disagree that we have that - and for FC didn't cross out)

Genetic algorithm

- Complicated
- but need to tune weights for parameters
- often used to tune weights for neural networks
- It's searching search space for a solution
- how much it exploits current solution
 - ↳ solution pressure
- or keep best 2 or 3
 - ↳ elitism
- how much it explores new solutions

* exploits vs explores *

Blocks i can use applet online: car

↳ tests on rocky road as fitness mechanisms
Can splice them together to see a bunch

②

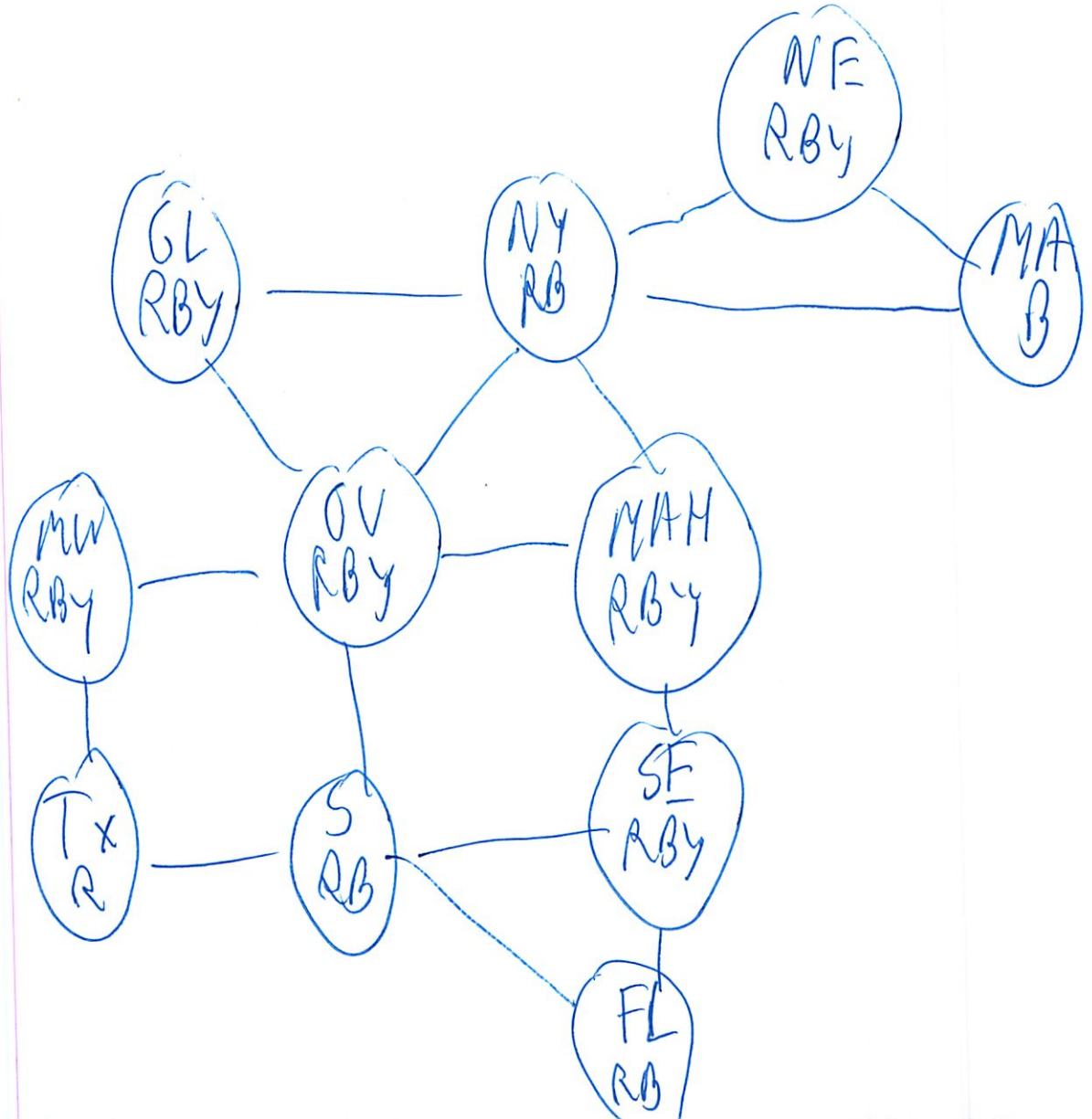
However if space of possible solutions is too large
might not work well

You can tune it to the game

But a more basic algorithm might be just as good

Last week: Constraint satisfaction

This week: More of same



③

Forward checking w/o constraint propagation

Banned RB, BR, Y₀Y

Political map

Trying to ~~color~~ color

Top is state name

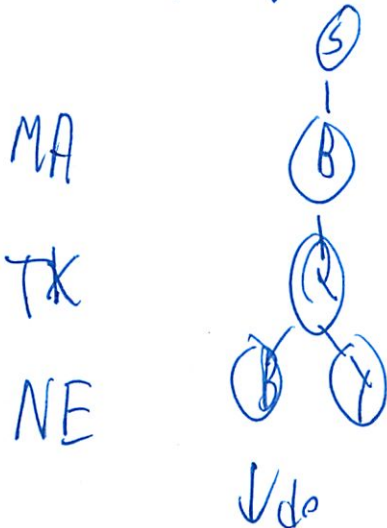
Go in order specified

Forward checking

- So cross out R in NE and NY

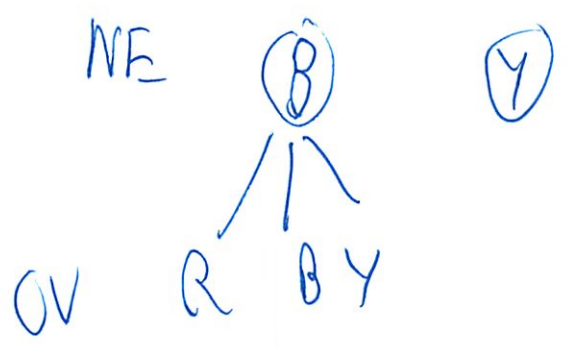
Then look at TX next

And ~~the~~ NE



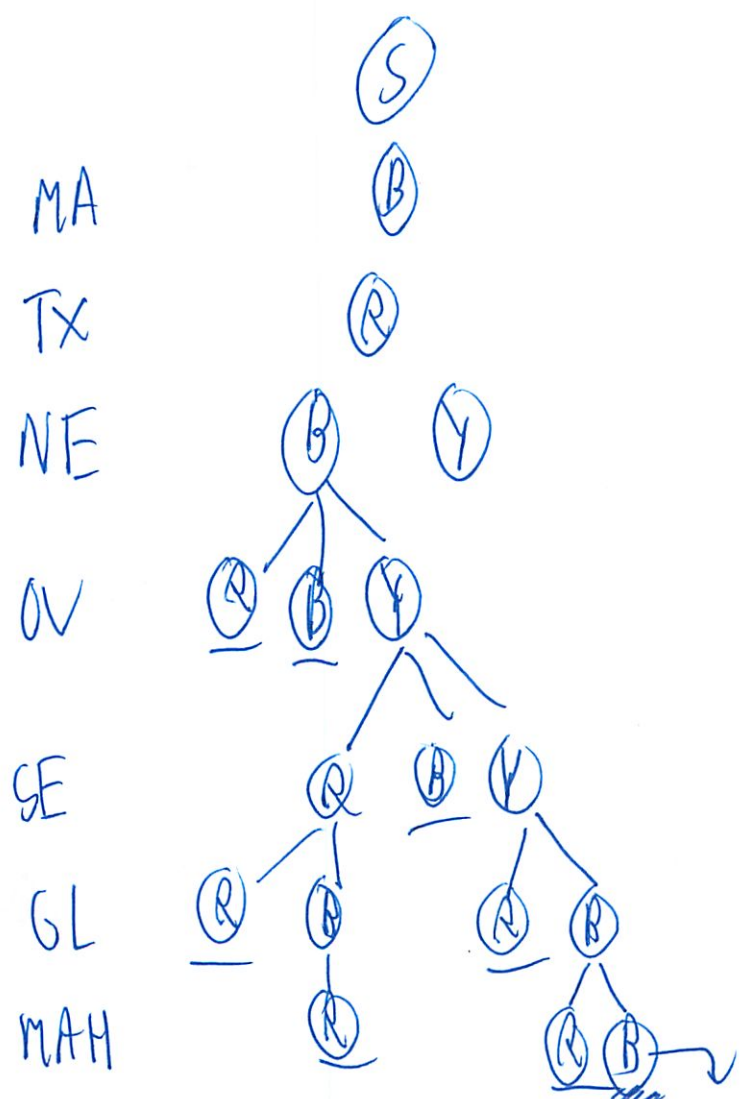
Look in order R → B → Y

4



So at each position try to cross out
 If something is reduced to no possibilities - backtrack

Etc so tree becomes



5

When assigning - you don't have to cross out other one a state to a color

MAH

MW

S

NY

F



That is our solution

Next part Forward checking w/ constraint propagation
Through singleton domains

So restart problem



This is the one when you only have 1 left -> you check that

As before

when assigning one, run constraint propagation

pick next ^{singleton} one at random

6

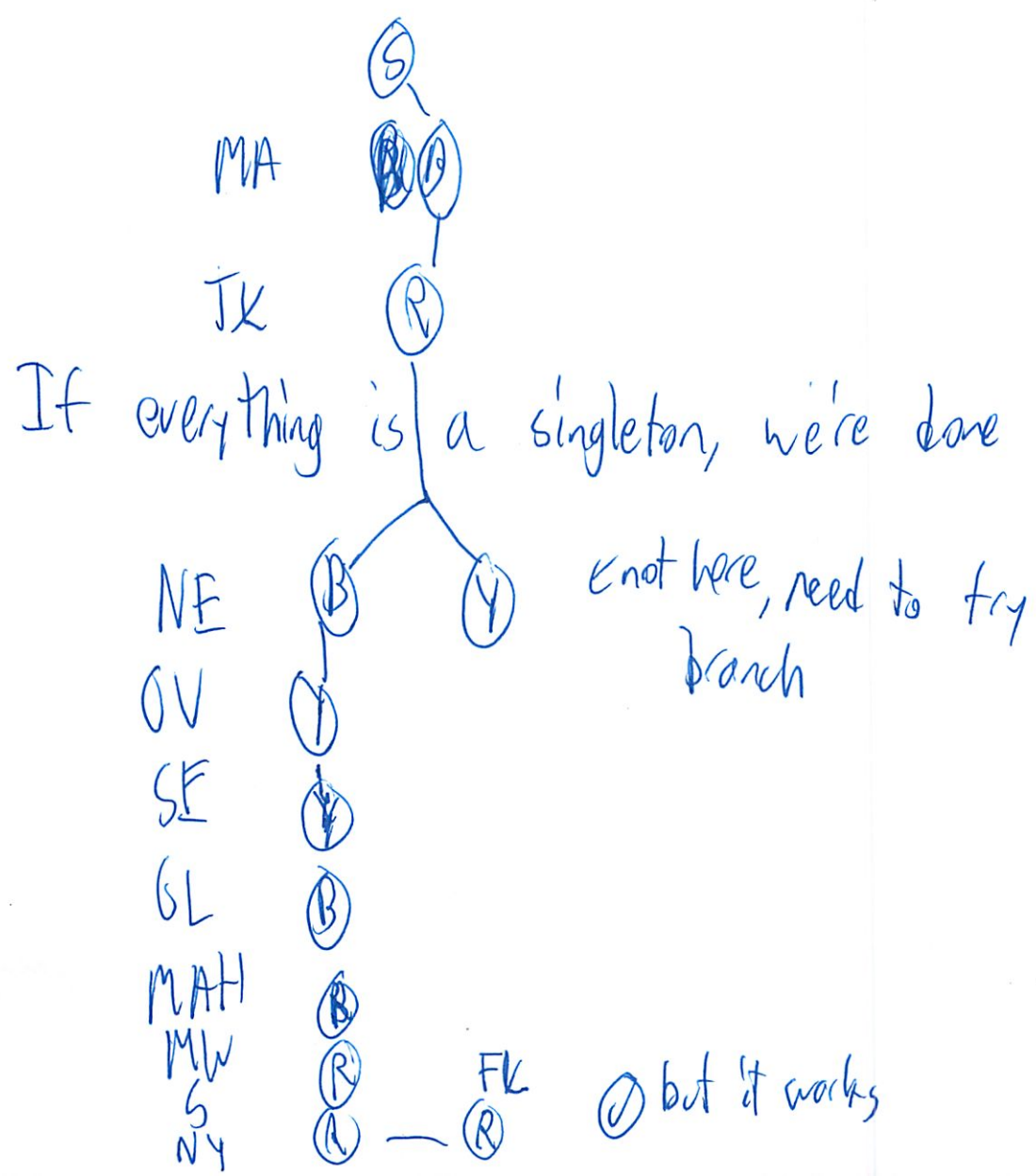
What

We said we would pick that ^{next singleton} at random

L -> Specified sometimes
- if not specified could be random

) comes out the same either way

Then On this problem a lot of constraint props,
then do tree



7

If it was brute force it would be a ton more work

- huge mess
- good luck writing
- (I'm not going to try)

Test Wed

- Games
 - ~~A~~ Minimax
 - α, β
 - static eval
 - reordering nodes
- Constraint propagation
 - today
- ~~||~~ Things from lecture
 - could be multiple choice or not

Ce.034 Topic
Review
Exam 2

10/25

Games + Adversarial Search

- Minimax
- Scoping
- L, B
- Progressive Deepening

From before

Extended - works for all

Expanded - look at nearby

Enqueued - added to queue

Chess think ahead of opponent

So have static evaluator

$$S = F(f_1, f_2, f_3, \dots)$$

- can be linear
- or anything

②

Then min max

- Our move: max
- Their move: minimize

Alpha, beta ~~don't~~ not visit certain paths

(this I need to practice)

(hard to describe and description is different)

α as low as can go

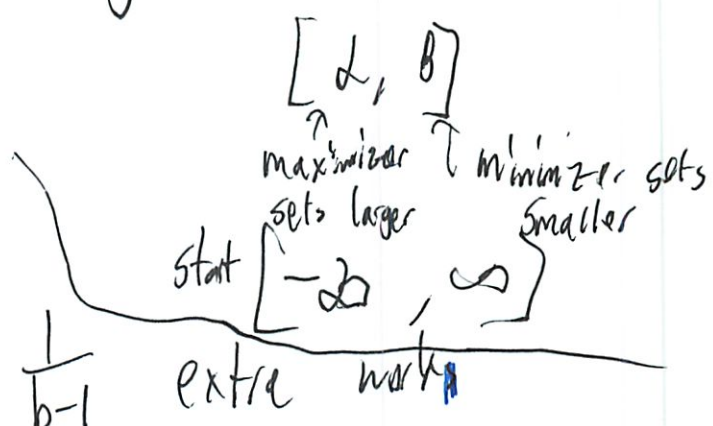
β as high

Progressive deepening

- insurance

- cost of doing $\frac{1}{b-1}$ exponential

- but any time



Best for α, β , put best option 1st

(For nearest neighbor its most constrained)

3

α : absolute worst case for maximizer

- if maximizer get less than α , take a

static is expensive

Show White: Take beauty from parents or steal from children

(Cutoff $\alpha \geq \beta$)

Progressive deepening i need to first static eval

- are you guaranteed to do as least as well

May not have values at each node

Letters can have ~~the~~ values to order

- need to read up

How stuff was figured out

Visual projects

Vertices



4

10/25
Later

deduction - many to few

induction - few to many

Abjunction - usually true, so assume true

Convex edge (+)

Concave " (-)

> on boundaries

→ on the right
• c-stuff of object

←

A bunch of possibilities

Looking for Tri-hedral

- 3 faces

- usually w/ cubes

Start labeling and see if it will work

①

Static

Constraint Propagation

- Coloring a map
- like any scheduling problems
- domain reduction : what does this exactly refer to?

↳ no good results online

Is it writing terms?

Variables - are or can be assigned

Domains D - bag of values

Constraint C - limit values in typical

pair of domains

Or backing up when 0 possibilities

Based on DFS

For each Variable V

Try each value X

Through every constraint

⊘ If no possibilities back up

⑦

Different things to consider

↳ and when to consider
before assigning something

but we only have our 3 big methods

Can have a heuristic → most constrained

Our 3 types

- Backtracking / Brute force

- BT w/ Forward checking (FC)

- BT w/ FC through singleton domains

- AC-2 (↳ When have only 1 item left → do that
BT w/ FC through reduced domains)

AC-2 looks at pairs of junctions

Nothing is sure-fire - depends on topology

The list of
methods was
↓ always confusing

↳ Yes it's a method *of* *the*

Sometimes can switch variable and domain
Do right thing

↳ in order → order given

keep domains in margin to see

8

Image Recognition

Marr's model

Primal Sketch \rightarrow $2\frac{1}{2}D$ sketch w/ vectors \rightarrow Generalized cylinder \rightarrow Library \rightarrow Recognition

If had few Marr's Pictures, could you get a sense?

- 4 eq 4 unknowns!

Can look down on top of object + rotate

3rd Correlation

Max SS $f(x,y) \sim \sim$

works well w/ lot of noise

It matters a lot how much you describe the blobs

- want just right Goldilox principle

9

Learning

Nearest
neighbor

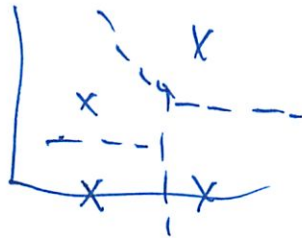
Feature
Computer

Lib features
Distance detector

↓
recognition

So for ~~the~~ look for k nearest pts
given add

Whatever is ~~closest~~ most - say that
for 1 nearest neighbor can also draw lines



Can have sections by themselves

Don't need to be connected with a line ---

Could also do min θ

Can teach robot to throw ball
w/ gigantic table
↳ might be too big

10

Sleep

- important
 - after 36 hrs awake, bad at firing
 - after 20 days w/ 6 hrs at 80% capacity
 - 30 min or more naps help
-

Constraint propagation

- some static constraints need to apply 1st AC best on square \rightarrow line labeling

Love never did a single example of this

Line labeling - how put stuff in machine is key

Can't go be 5 for accurate nearest neighbor

6.034 is good at teaching

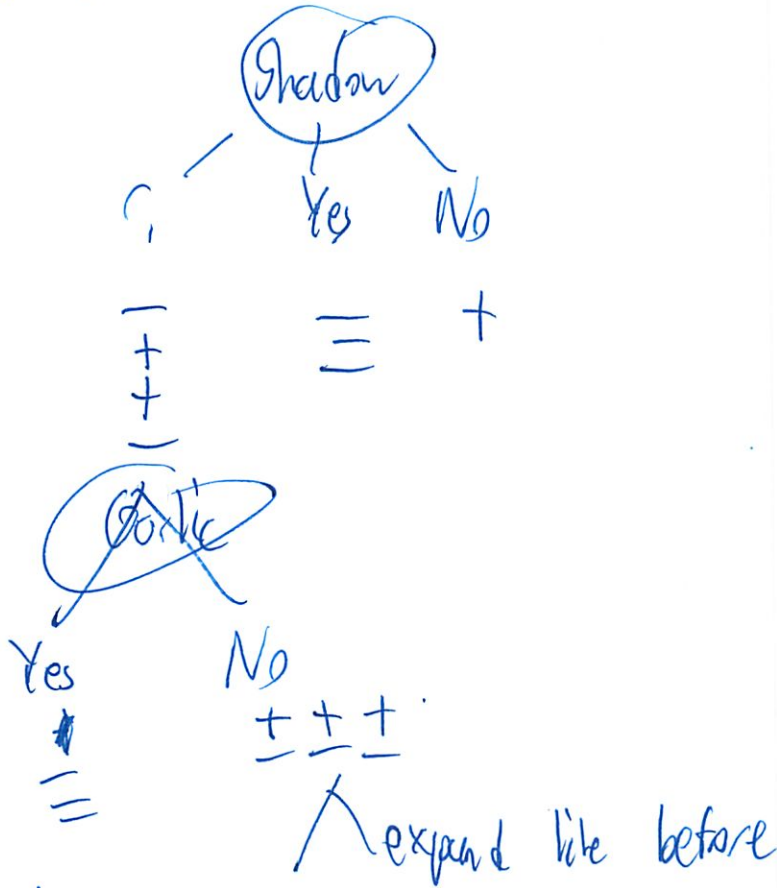
What is coverage? Learning / classification trees not on?

(11)

Classification Trees

Symptoms of vampiress

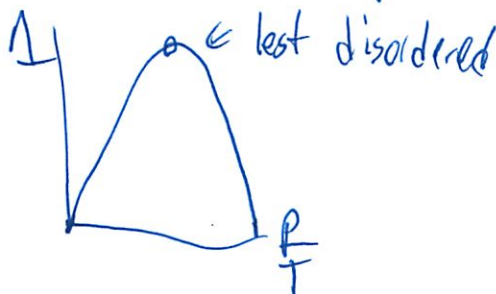
Trees of tests



Need a measure of disorder

↳ Since we pick one w/ least disorder

$$D(\text{set}) = -\frac{P}{T} \log_2 \frac{P}{T} - \frac{N}{T} \log_2 \frac{N}{T}$$



(12)

Do as weighted avg of all items in set

Split each branch up further by recurring algorithm

Then we can assign rules

Use sources to combine like rules
Like k-maps

Neural Networks

I know is NOT on it

This is open book, right

I'm pretty sure k-Nearest Neighbor and ~~Decision~~ trees is not on here

? No are not

α, β Cheat Sheet

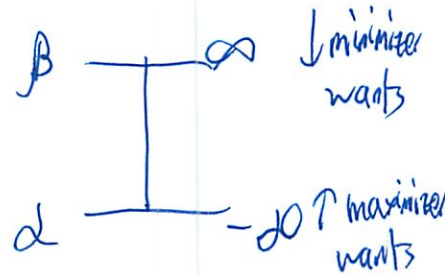
10/25

α

- Starts $-\infty$
- maximizer sets when (no)
- as low as can go
- absolute worst case for maximizer
- ~~if maximizer gets $< \alpha$, then take α~~
don't think about

β

- starts ∞
- minimizer sets when (no)
- as high as can go



Cutoff $\alpha \geq \beta$

Snow White Take beauty from parents or steal from children



α, β Practice

10/25

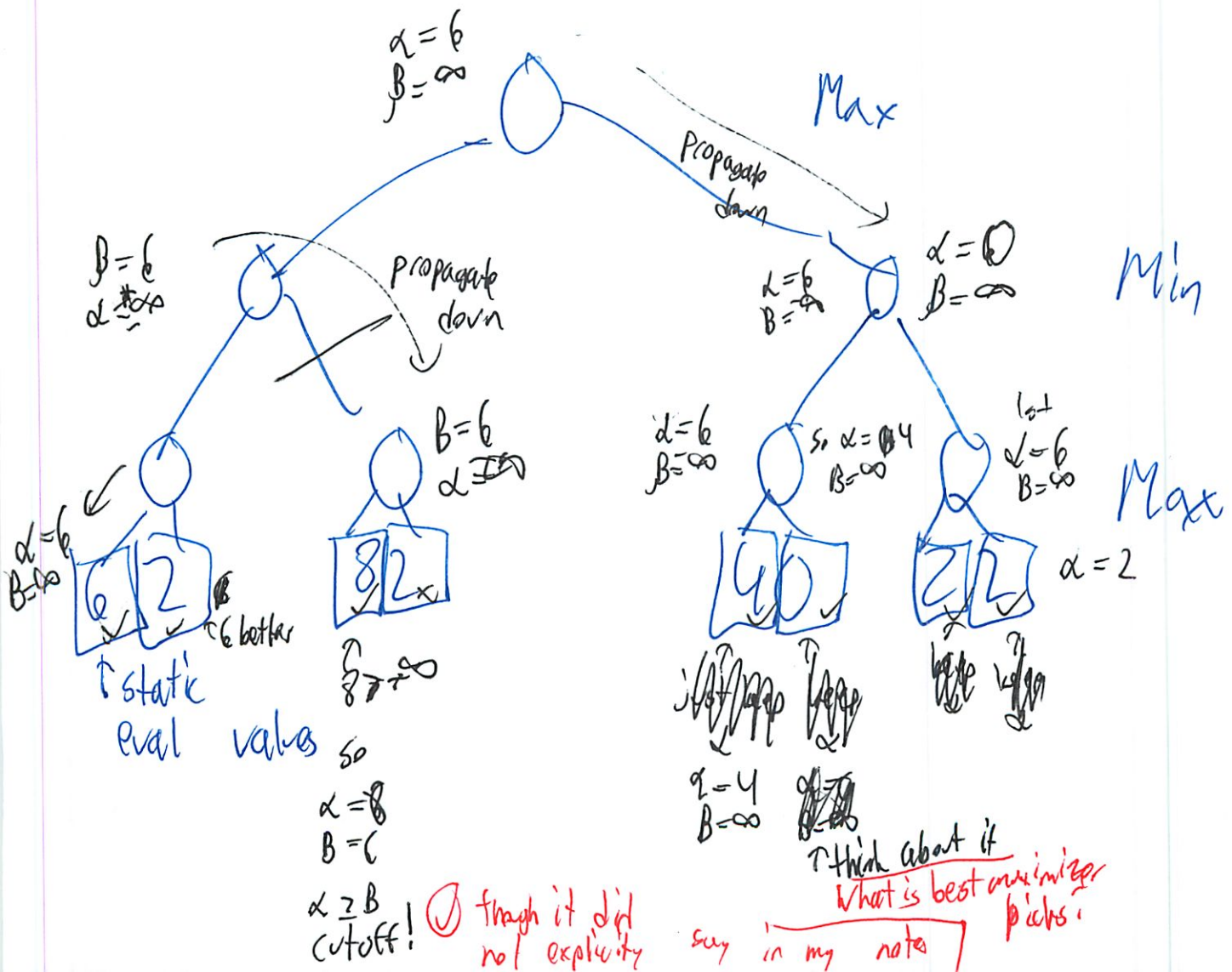
α, β

Make ~~down~~ lsls part understanding

And first bubble down

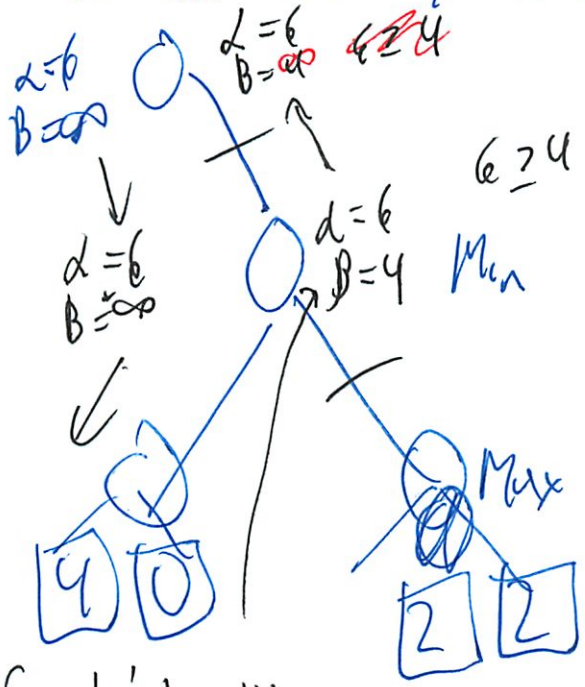
Then bring current values - w/ α as ~~α~~ ∞
See

Let me try myself



2

So let me try this again



eval independtly

$k=4$
 $B = \infty$

0
 not
 bigger

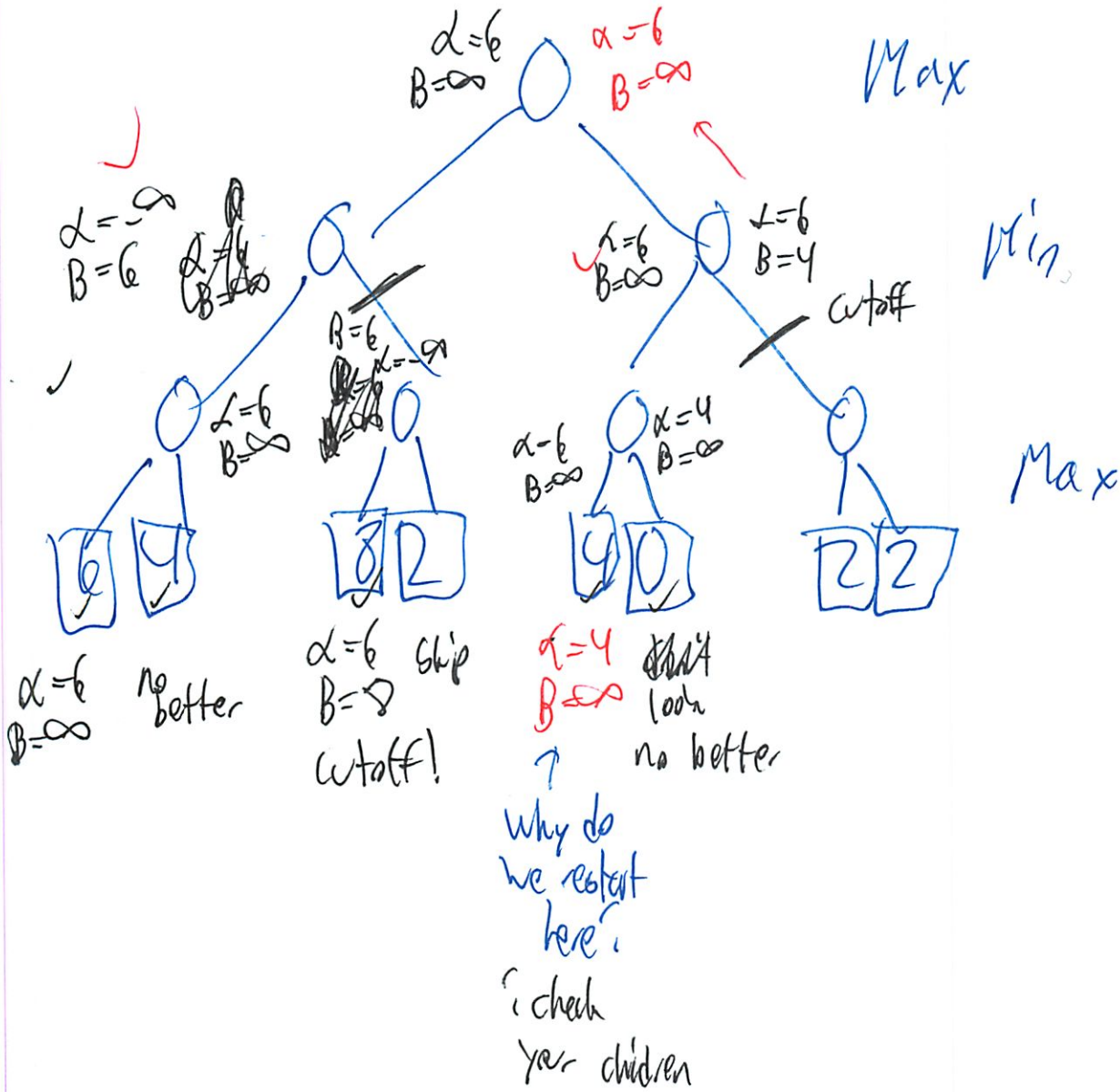
Evalled 3 nodes

LI had 4 6, 2, 8, 4, 0

No saved 3 nodes so (✓) what I had

③

Redo same problem myself



I'm not convinced did it right in original work

5

So still confused

Each has a different convention to write

- Found slides online

keys Maximizer sets \downarrow
 Minimizer sets β

down \downarrow When transferring down use α of intel record
 while we work on Beta

\uparrow up update Beta from previous α

Think about min/max
Check for pruning

I think I have to make an

I'm also confused by write once when propagating \downarrow
then updating it

Not all sols do that
but written differently!

6.034 Quiz 2 October 21, 2009

*10/25
Practice*

Name	
EMail	

Circle your TA and recitation time, if any, so that we can more easily enter your score in our records and return your quiz to you promptly.

TAs
Erica Cooper
Matthew Peairs
Charles Watts
Mark Seifter
Yuan Shen
Jeremy Smith
Olga Wichrowska

Thu	
Time	Instructor
11-12	Gregory Marton
12-1	Gregory Marton
1-2	Bob Berwick
2-3	Bob Berwick
3-4	Bob Berwick

Fri	
Time	Instructor
1-2	Randall Davis
2-3	Randall Davis
3-4	Randall Davis

Problem number	Maximum	Score	Grader
1	50		
2	50		
Total	100		

There are 11 pages in this quiz, including this one. In addition, tear-off sheets are provided at the end with duplicate drawings and data.

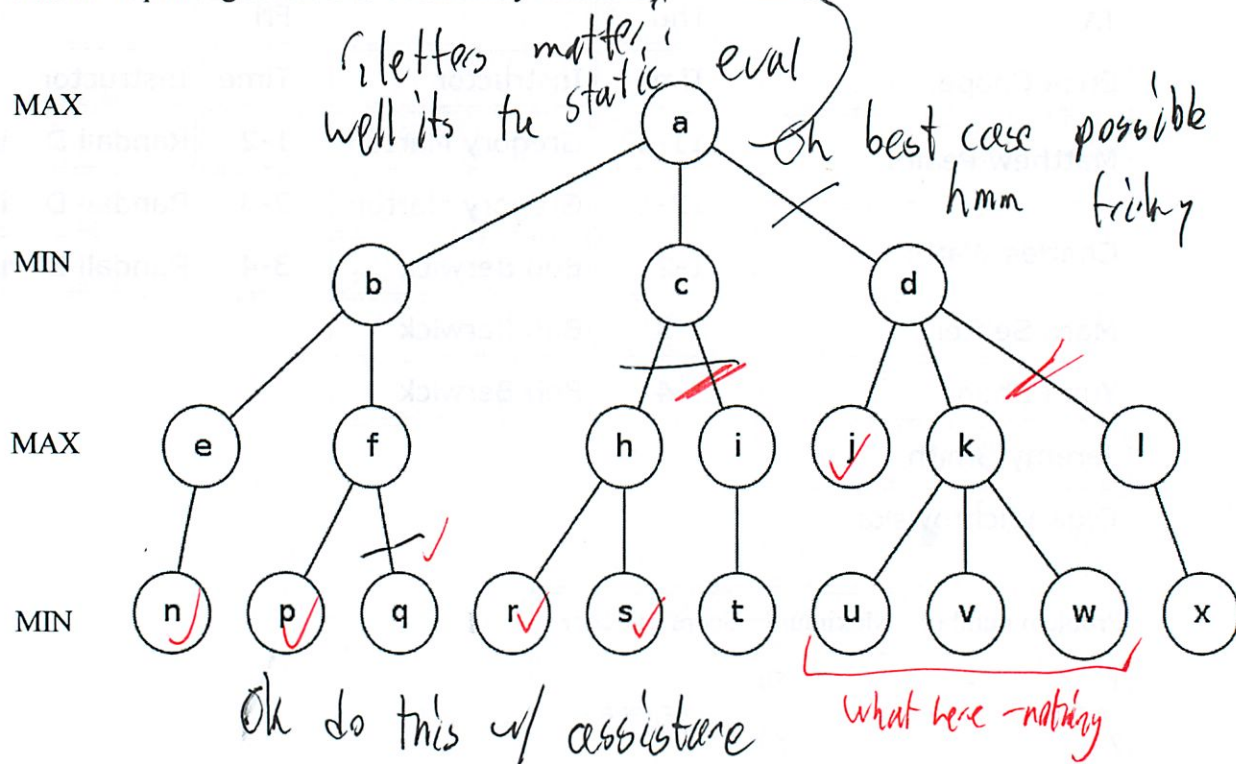
As always, open book, open notes, open just about everything.

Problem 1: Games (50 points)

For your reference in working this problem, pseudo code for the standard version of minimax with alpha beta is given on the tear off sheet at the end.

Part A: Working with a maximally pruned tree (25 points)

For the following min-max tree, cross out those leaf nodes for which alpha-beta search would **not do static evaluations** in the best case possible (minimum number of static evaluations, maximum pruning of nodes to be statically evaluated).



Part A1

No, did this all wrong

Now, list the leaf nodes at which alpha-beta would do static evaluations in the best case possible.

n p r s j

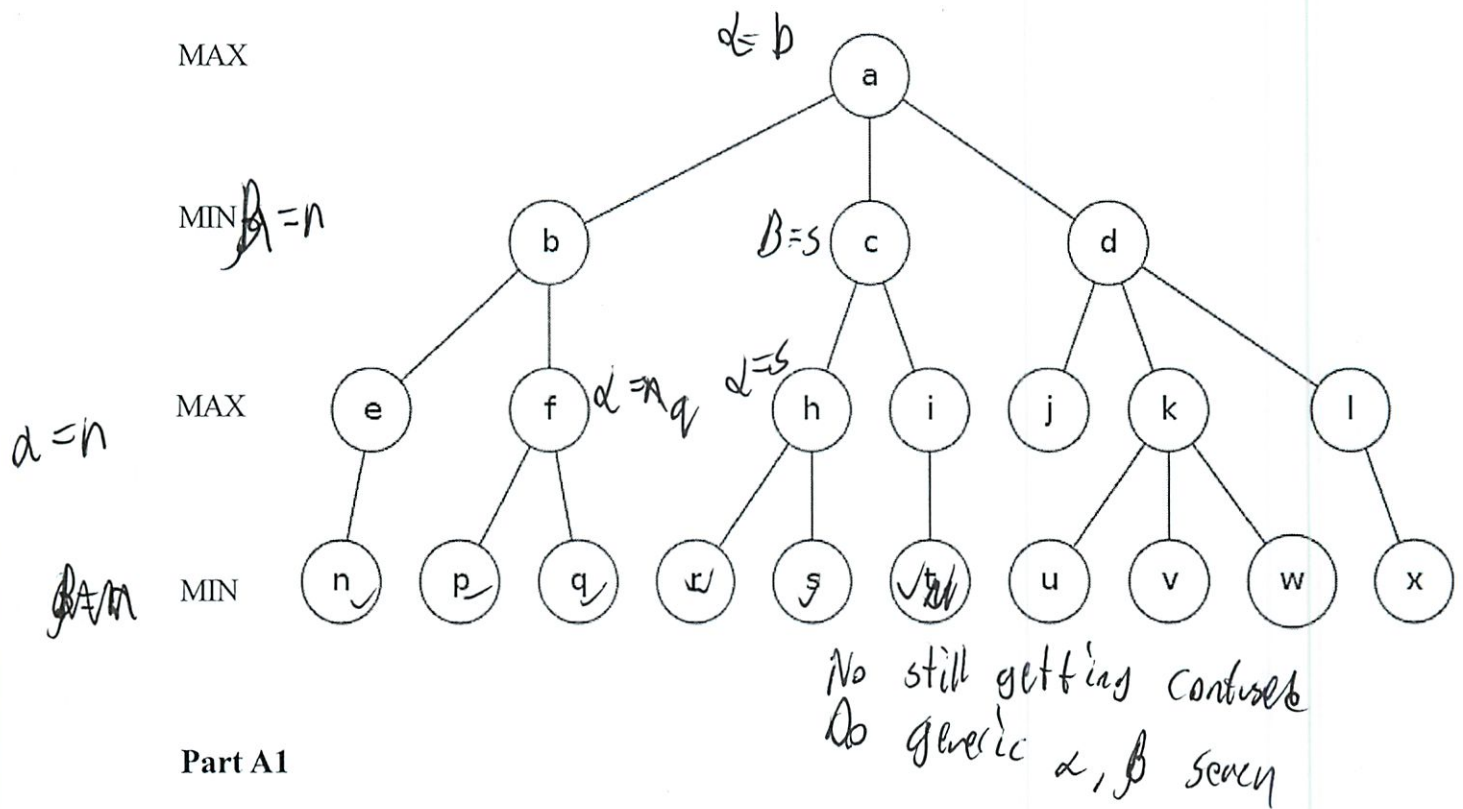
So letters did matter?
Try again w/ doing letters

Problem 1: Games (50 points)

For your reference in working this problem, pseudo code for the standard version of minimax with alpha beta is given on the tear off sheet at the end.

Part A: Working with a maximally pruned tree (25 points)

For the following min-max tree, cross out those leaf nodes for which alpha-beta search would **not do static evaluations** in the best case possible (minimum number of static evaluations, maximum pruning of nodes to be statically evaluated).

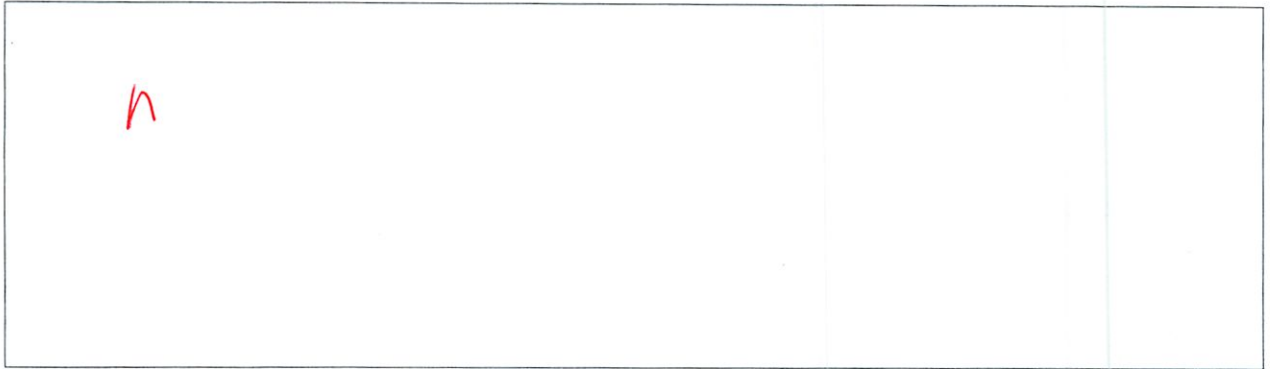


Part A1

Now, list the leaf nodes at which alpha-beta would do static evaluations in the best case possible.

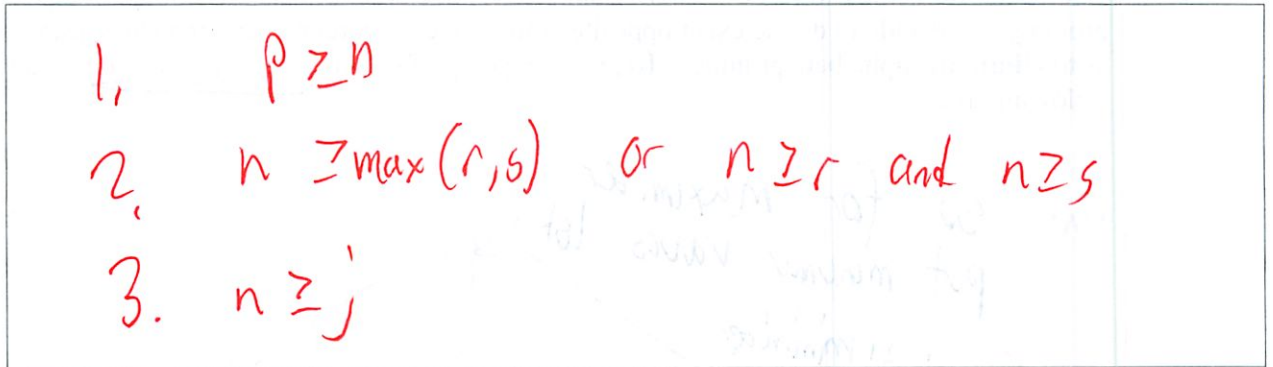
Part A2

What is the final value returned by the alpha beta search in the best case possible for the given tree? Express your answer as the simplest function of the static values of the leaf nodes (e.g. take n to be the static value at the leaf node labeled n). Your function may contain operations such as **max** and **min**.



Part A3

What constraints ensure best case possible (minimum static evaluation) for the given tree? State your constraints as inequalities on the static values of the leaf nodes.



Part A4

Suppose your static evaluation function, $S(\text{node})$, is modified as follows:

$$S'(\text{node}) = 42 \times S(\text{node}) + 1000. \quad (\text{If } S(\text{node}) = 1, S'(\text{node}) = 1042)$$

Would your answer for Part A1 be the same for all possible $S(\text{node})$ values? Yes No

Suppose your function were

$$S'(\text{node}) = -42 \times S(\text{node}) + 1000.$$

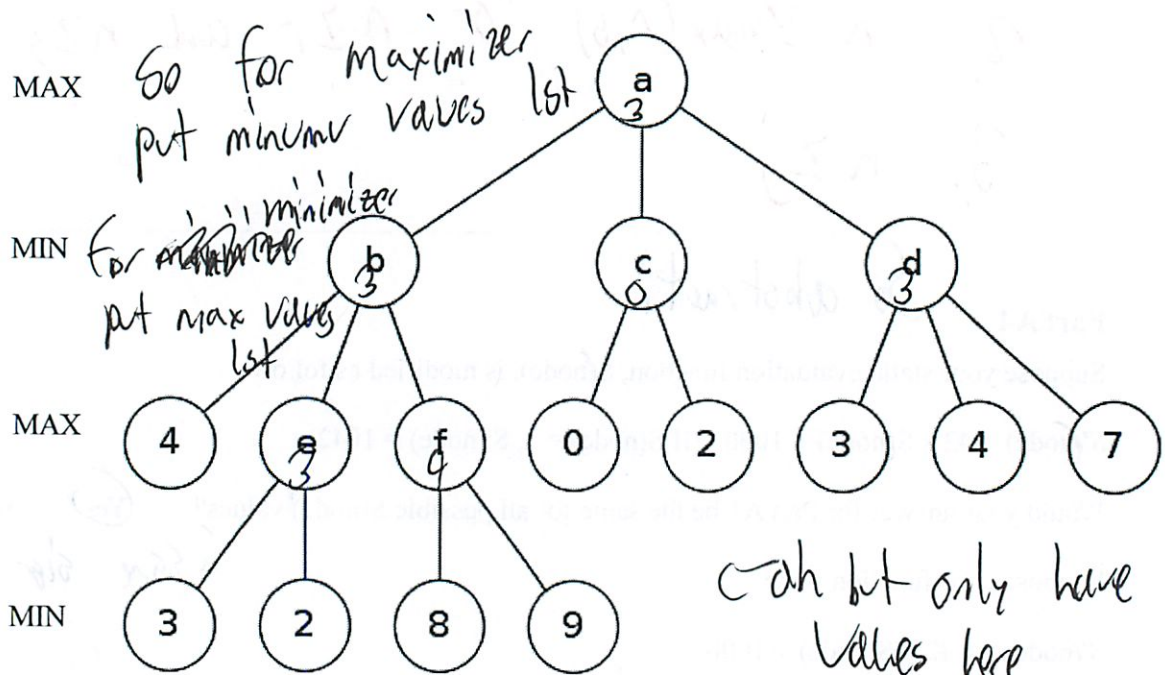
Would your answer for Part A1 be the same for all possible $S(\text{node})$ values?

Yes No
Signs change ✓

Explain your reasoning in less than 4 meaningful sentences or give a short proof.

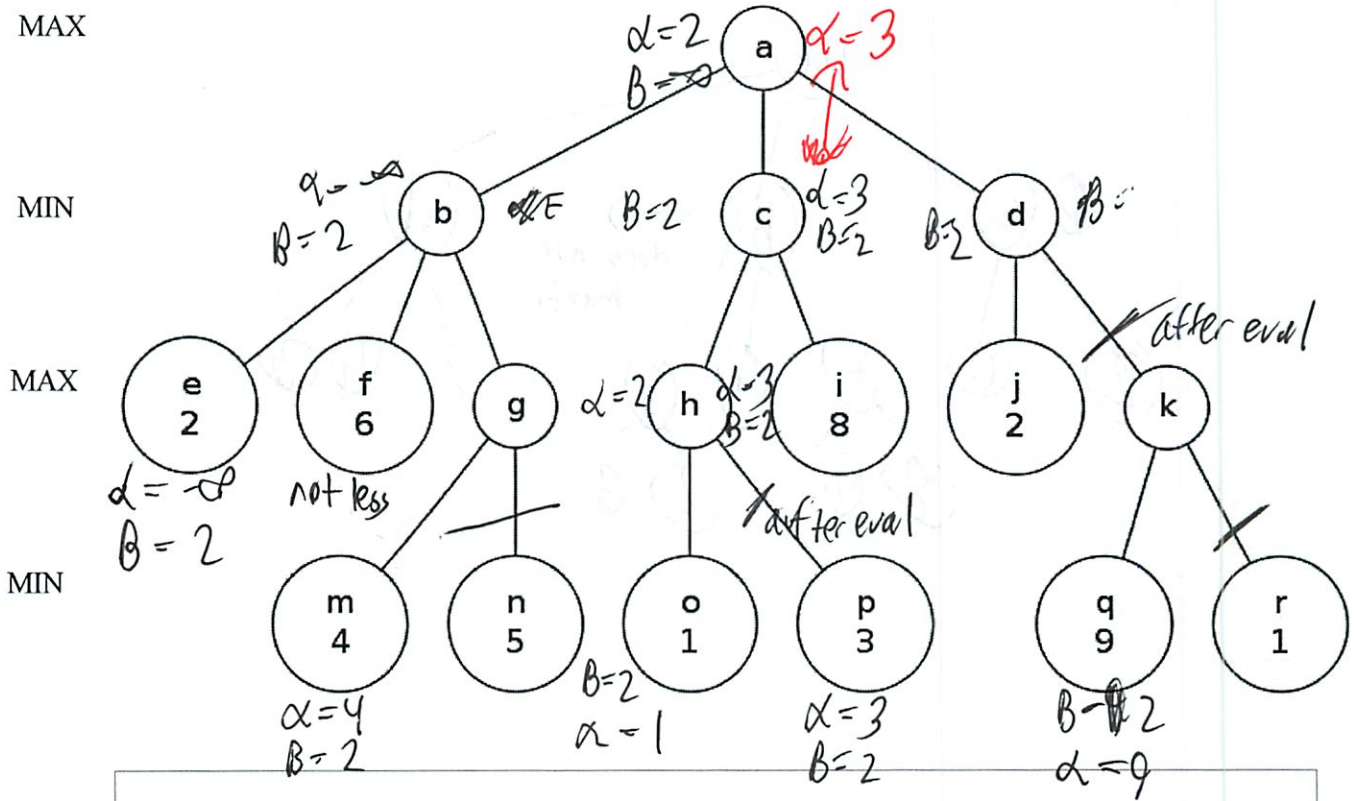
Part B1: Digging Progressively Deeper (10 points)

You decide to put your 6.034 knowledge to good use by entering an adversarial programming contest. In this contest, one player (your opponent) is tasked with optimizing the running time of alpha beta search with progressive deepening. Your goal, as her adversary, is to slow down her algorithm. Remembering a key insight from 6.034: node order affects the amount of alpha-beta pruning, you decide to do the exact opposite: you decide to reorder your opponent's search tree so as to eliminate alpha beta pruning. To practice, you perform your anti-optimization on the following tree.



Part B2 (5 points) Another α, β ! One last try

For the tree given below (not the same tree as in B1), perform alpha beta search. Do your work on the tree as given; **do no reordering**. List the leaf nodes (letter) in the order that they are statically evaluated in the box below the tree.

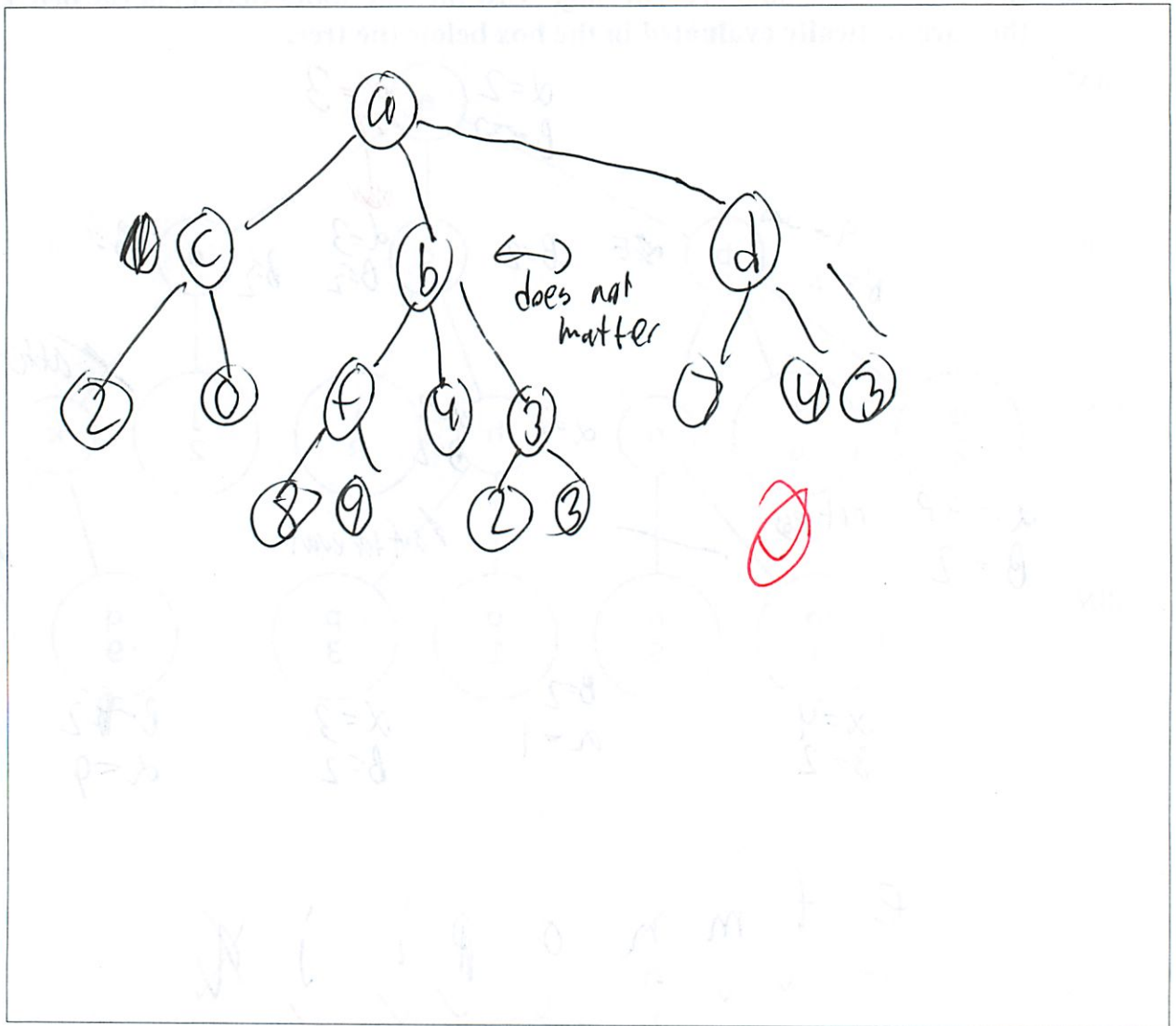


e f m n o p i j

✓ ✓ ✓ ✓ ✓ ✓ ✓ ✓

I crossed it out too

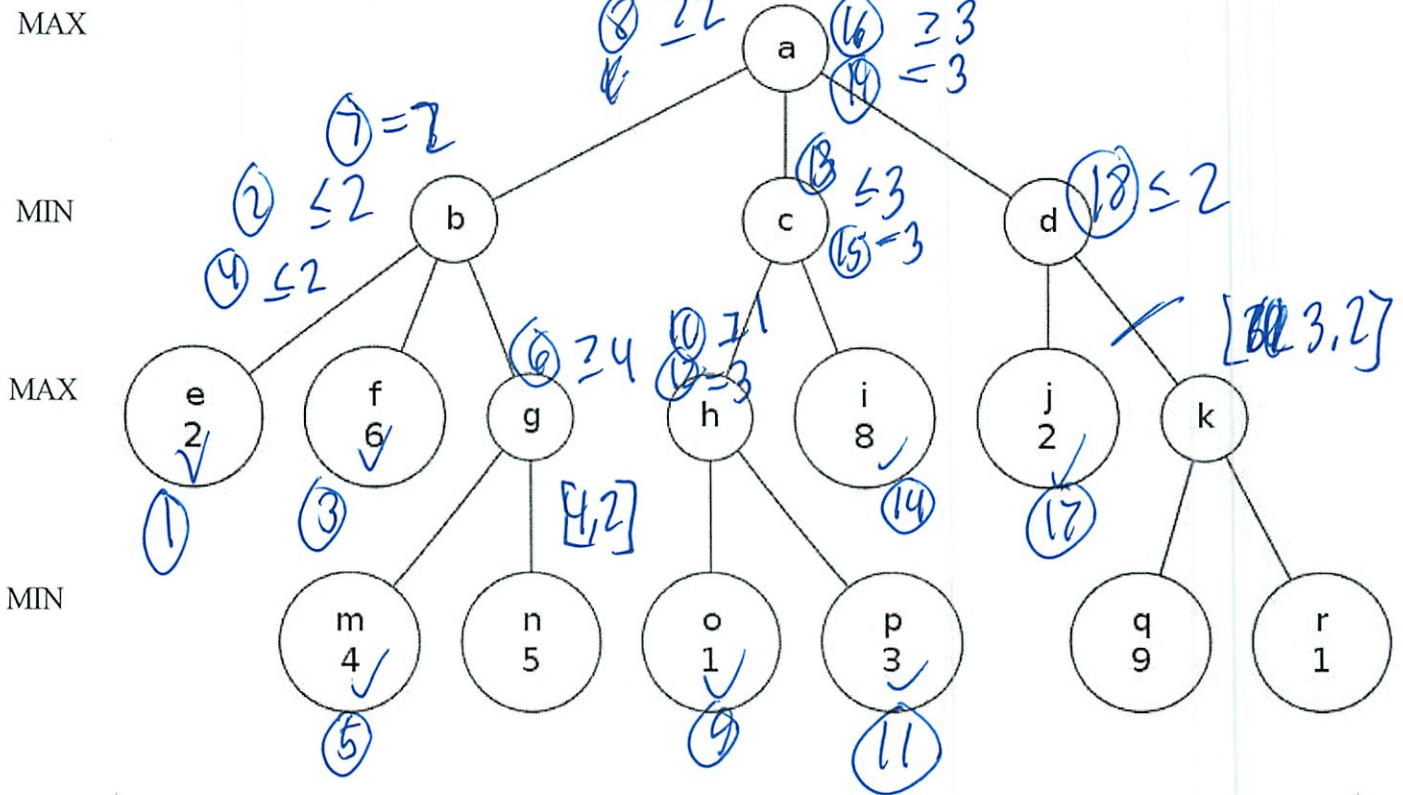
Reorder the nodes of the tree at every level such that alpha-beta search does **no pruning**. You may only reorder nodes (you cannot reattach a node to a different parent). Show your reordered tree below:



Part B2 (5 points)

Use \geq framework

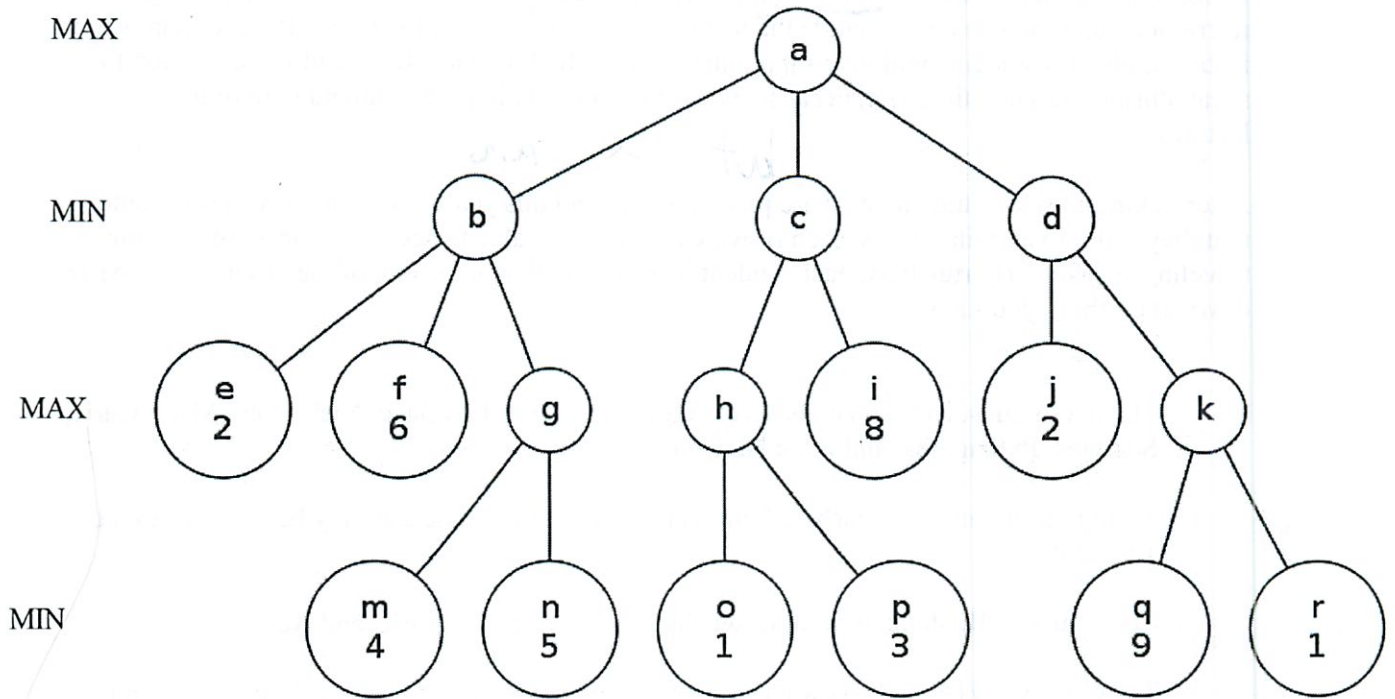
For the tree given below (not the same tree as in B1), perform alpha beta search. Do your work on the tree as given; **do no reordering**. List the leaf nodes (letter) in the order that they are statically evaluated in the box below the tree.



It's so different from the other way
 I used to do it
 So this tells us what values
 But does not help on cut-off

Part B3 (10 points)

Now, you are to repeat your alpha beta search, but this time with **initial values for alpha = 2 and beta = 7**. List, in the box below the tree, the leaf nodes in the order that they are statically evaluated, given initial alpha = 2, and beta = 7,



What is the best move according to the search with alpha = 2 and beta = 7?

Problem 2: Time Travelers' Convention (50 points)

The MIT Time Travel Society (MITTTS) has invited seven famous historical figures to each give a lecture at the annual MITTTS convention, and you've been asked to create a schedule for them. Unfortunately, there are only four time slots available, and you discover that there are some restrictions on how you can schedule the lectures and keep all the convention attendees happy. For instance, physics students will be disappointed if you schedule Niels Bohr and Isaac Newton to speak during the same time slot, because those students were hoping to attend both of those lectures.

but ∞ rooms

After talking to some students who are planning to attend this year's convention, you determine that they fall into certain groups, each of which wants to be able to see some subset of the time-traveling speakers. (Fortunately, each student identifies with at most one of the groups.) You write down everything you know:

The list of guest lecturers consists of Alan Turing, Ada Lovelace, Niels Bohr, Marie Curie, Socrates, Pythagoras, and Isaac Newton.

- 1) Turing has to get home early to help win World War II, so he can only be assigned to the 1pm slot.
- 2) The Course VIII students want to see the physicists: Bohr, Curie, and Newton.
- 3) The Course XVIII students want to see the mathematicians: Lovelace, Pythagoras, and Newton.
- 4) The members of the Ancient Greece Club wants to see the ancient Greeks: Socrates and Pythagoras.
- 5) The visiting Wellesley students want to see the female speakers: Lovelace and Curie.
- 6) The CME students want to see the British speakers: Turing, Lovelace, and Newton.
- 7) Finally, you decide that you will be happy if and only if you get to see both Curie and Pythagoras. (Yes, even if you belong to one or more of the groups above.)

Oh already did this

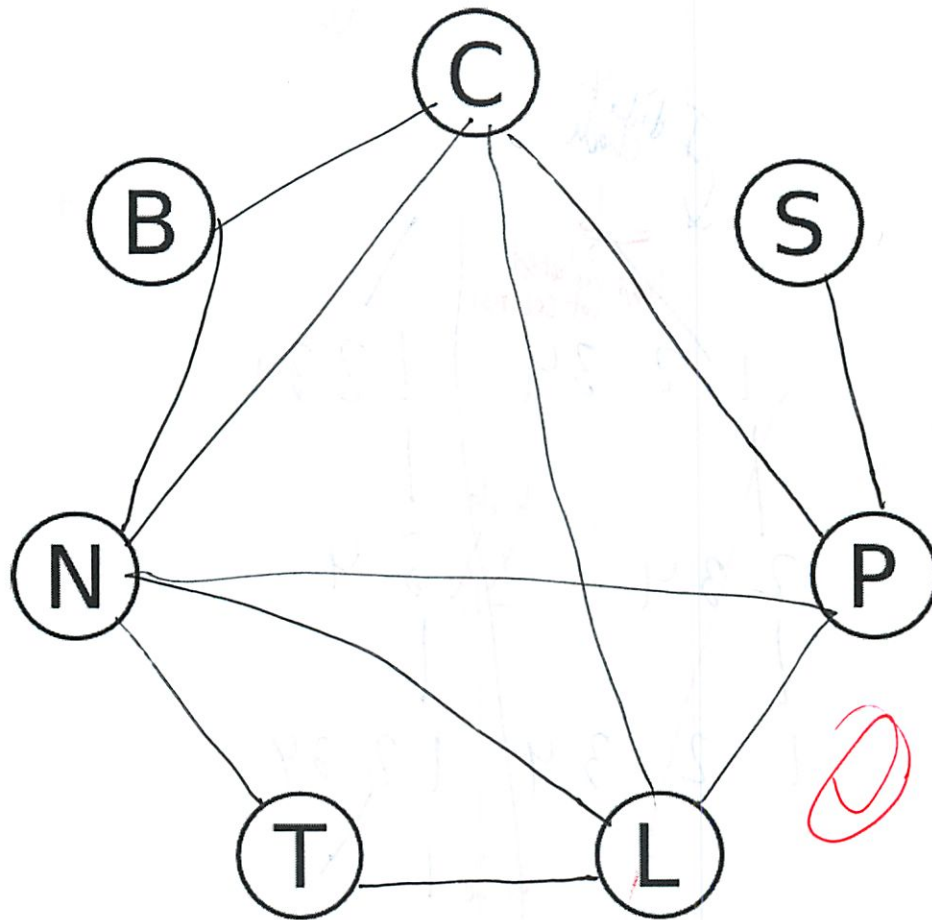
- or saw it done

forget it though

Part A (5 points)

That's a lot of preferences to keep track of, so you decide to draw a diagram to help make sense of it all. Draw a line between the initials of each pair of guests who must not share the same time slot.

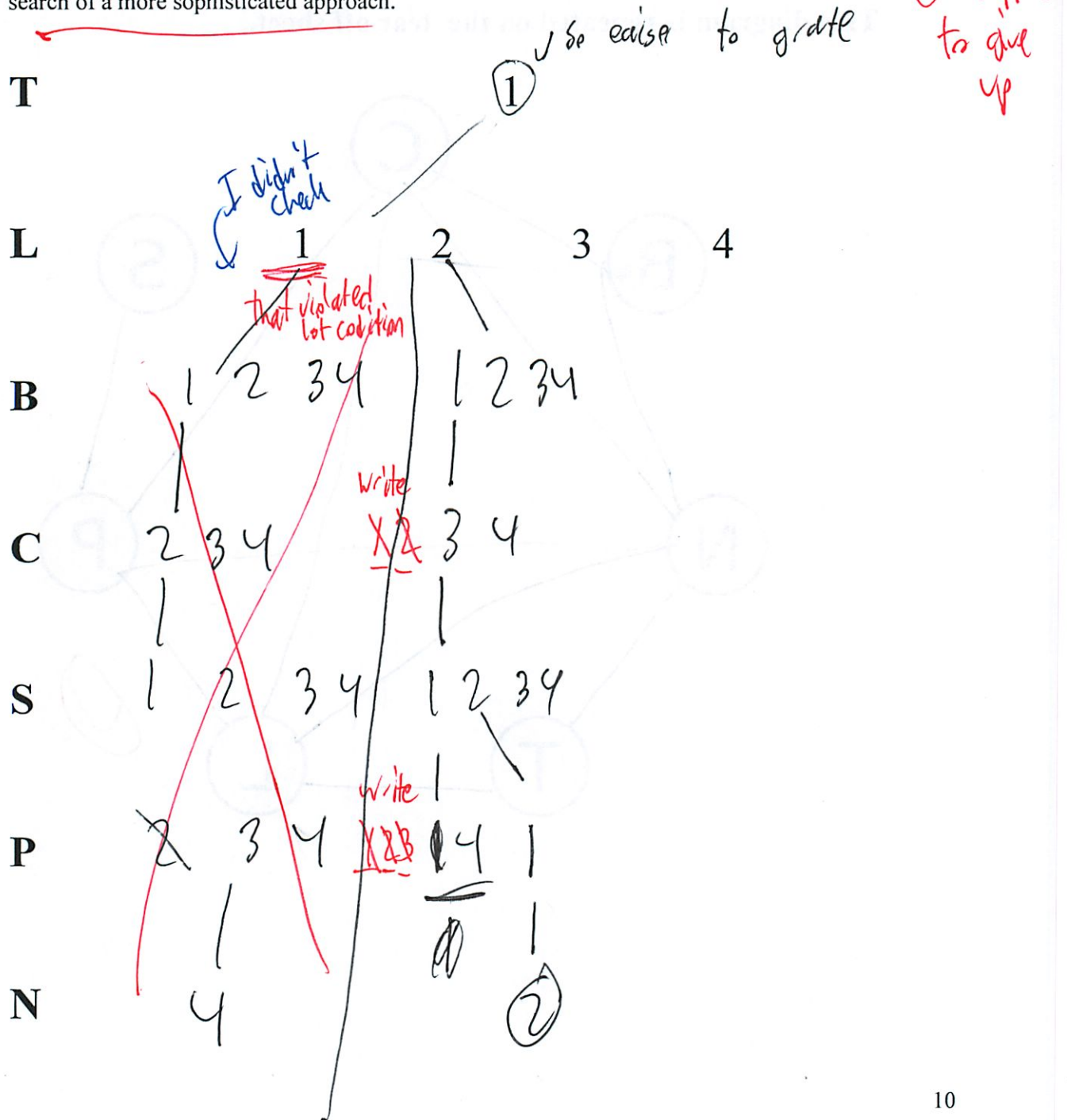
This diagram is repeated on the tear off sheet.



Part B (15 points)

You decide to first assign the time slots (which conveniently happen to be 1, 2, 3, and 4 pm) by using a **depth-first search with no constraint propagation**. The only check is to be sure each new assignment violates no constraint with any previous assignment. As a tiebreaker, assign a lecturer to the earliest available time slot (so as to get them back to their own historical eras as soon as possible).

In the tree below, Alan Turing has already been scheduled to speak at 1 pm, in accordance with constraint #1. Continue filling in the search tree up to the first time you try (and fail) to assign a time slot to Isaac Newton, at which point you give up in frustration and move on to Part C in search of a more sophisticated approach.



What is the final lecture schedule you hand in to MITTTS?

1 pm: Turing B P

2 pm: L S


3 pm: C 

4 pm: N

Part D (10 points)

Now, rather than backtracking, you're concerned about the amount of time it takes to keep track of all those domains and propagate constraints through them. You decide that the problem lies in the ordering of the guest list. Just then, you get a call from the MITTTS president, who informs you that **Alan Turing's schedule has opened up and he is now free to speak during any one of the four time slots.**

Armed with this new information, you reorder the guest list to maximize your chances of quickly finding a solution. In particular, which lecturer do you now assign a time slot to first, and why?

Most constrained (Newton) 1st 

Tear off sheet, you need not hand this sheet in.

You may use the following alpha-beta mini-max pseudo code as a reference:

```
alpha_beta_search(node, alpha = -infinity, beta = +infinity)
  v = max_value(node, alpha, beta)
  return the action associated with v
```

where start

```
max_value(node, alpha, beta)
  if is_leaf(node)
    return static_value(node)
  v = -infinity
  for child in children(node):
    v = MAX(v, min_value(child, alpha, beta))
    if v >= beta
      return v
  alpha = MAX(alpha, v)
  return v
```

```
min_value(node, alpha, beta)
  if is_leaf(node)
    return static_value(node)
  v = +infinity
  for child in children(node):
    v = MIN(v, max_value(child, alpha, beta))
    if v <= alpha
      return v
  beta = MIN(beta, v)
  return v
```

Try Pseudo Code

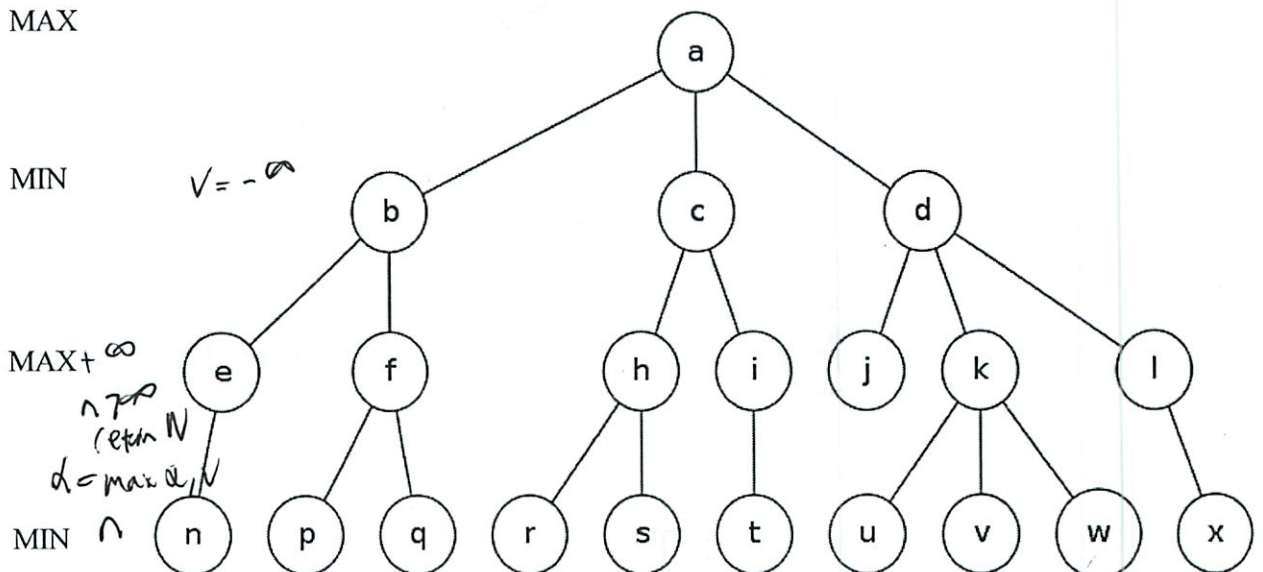
Problem 1, part A

MAX

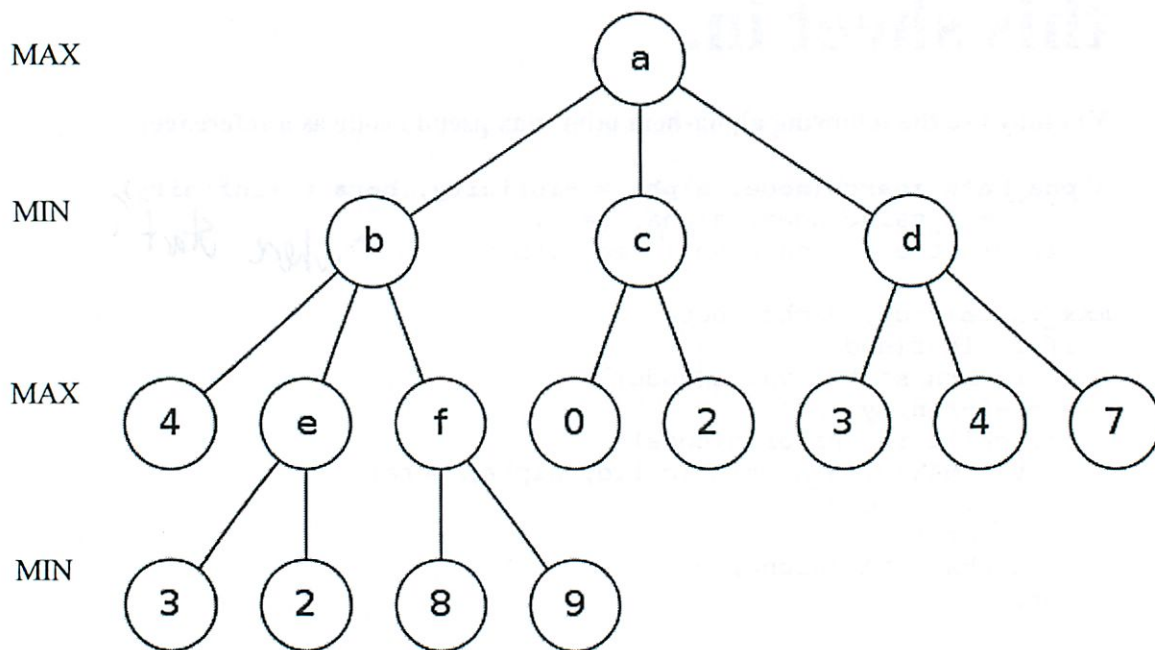
MIN

MAX + ∞

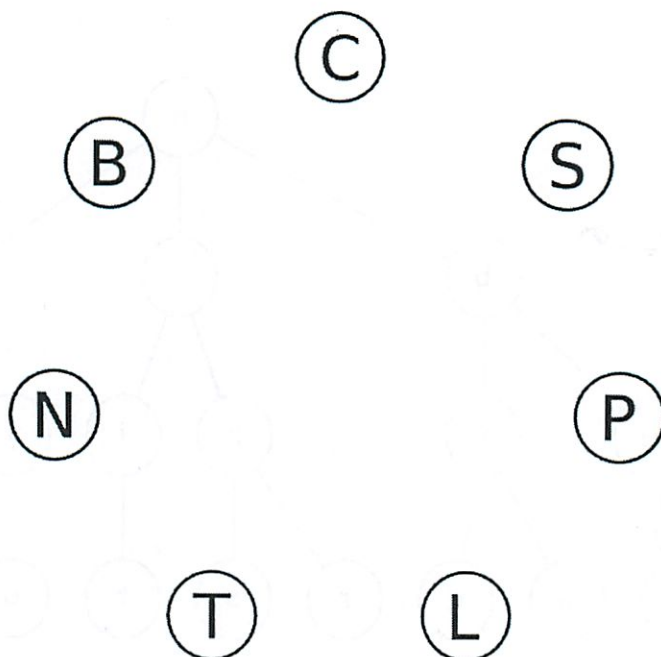
MIN



Problem 1, part B1



Problem 2, part A



6.034 Quiz 2
October 21, 2009

Name CHARLES WATTS
EMail

Circle your TA and recitation time, if any, so that we can more easily enter your score in our records and return your quiz to you promptly.

TAs	Thu	Fri
	Time Instructor	Time Instructor
Erica Cooper		
Matthew Peairs	11-12 Gregory Marton	1-2 Randall Davis
	12-1 Gregory Marton	2-3 Randall Davis
Charles Watts	1-2 Bob Berwick	3-4 Randall Davis
Mark Seifter	2-3 Bob Berwick	
Yuan Shen	3-4 Bob Berwick	
Jeremy Smith		
Olga Wichrowska		

Problem number	Maximum	Score	Grader
1	50		
2	50		
Total	100		

There are 11 pages in this quiz, including this one. In addition, tear-off sheets are provided at the end with duplicate drawings and data.

As always, open book, open notes, open just about everything.

6.034 Quiz 2
October 21, 2009

Name CHARLES WATTS
EMail

Circle your TA and recitation time, if any, so that we can more easily enter your score in our records and return your quiz to you promptly.

TAs	Thu	Fri
	Time Instructor	Time Instructor
Erica Cooper		
Matthew Peairs	11-12 Gregory Marton	1-2 Randall Davis
	12-1 Gregory Marton	2-3 Randall Davis
Charles Watts	1-2 Bob Berwick	3-4 Randall Davis
Mark Seifter	2-3 Bob Berwick	
Yuan Shen	3-4 Bob Berwick	
Jeremy Smith		
Olga Wichrowska		

Problem number	Maximum	Score	Grader
1	50		
2	50		
Total	100		

There are 11 pages in this quiz, including this one. In addition, tear-off sheets are provided at the end with duplicate drawings and data.

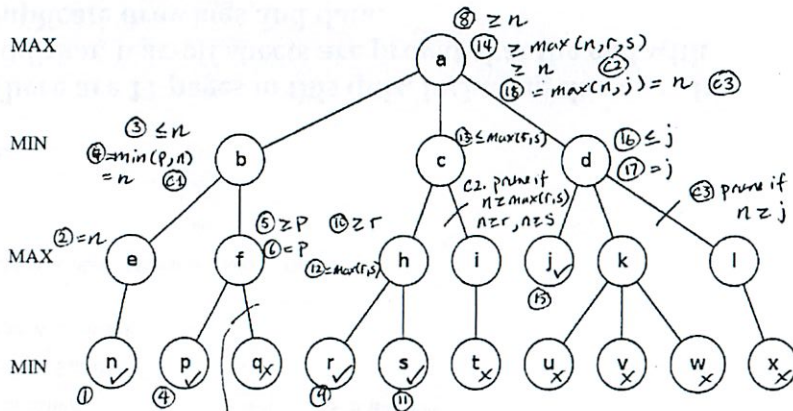
As always, open book, open notes, open just about everything.

Problem 1: Games (50 points)

For your reference in working this problem, pseudo code for the standard version of minimax with alpha beta is given on the tear off sheet at the end.

Part A: Working with a maximally pruned tree (25 points)

For the following min-max tree, cross out those leaf nodes for which alpha-beta search would **NOT** do static evaluations in the best case possible (minimum number of static evaluations, maximum pruning of nodes to be statically evaluated).



Part A1
 • constraints (c1)
 • prune if $p \geq n$

Now, list the leaf nodes at which alpha-beta would do static evaluations in the best case possible.

$n \ p \ r \ s \ j$

Part A2 4 pts

What is the final value returned by the alpha beta search in the best case possible for the given tree? Express your answer as the simplest function of the static values of the leaf nodes (e.g. take n to be the static value at the leaf node labeled n). Your function may contain operations such as \max and \min .

n

Part A3 8 pts

What constraints ensure best case possible (minimum static evaluation) for the given tree? State your constraints as inequalities on the static values of the leaf nodes.

(c1) $p \geq n$
 (c2) $n \geq \max(r, s)$ OR $n \geq r$ and $n \geq s$
 (c3) $n \geq j$

Part A4 4 pts each

Suppose your static evaluation function, $S(\text{node})$, is modified as follows:

$S'(\text{node}) = 42 \times S(\text{node}) + 1000$. (If $S(\text{node}) = 1$, $S'(\text{node}) = 1042$)

Would your answer for Part A1 be the same for all possible $S(\text{node})$ values? Yes No

Suppose your function were

$S'(\text{node}) = -42 \times S(\text{node}) + 1000$.

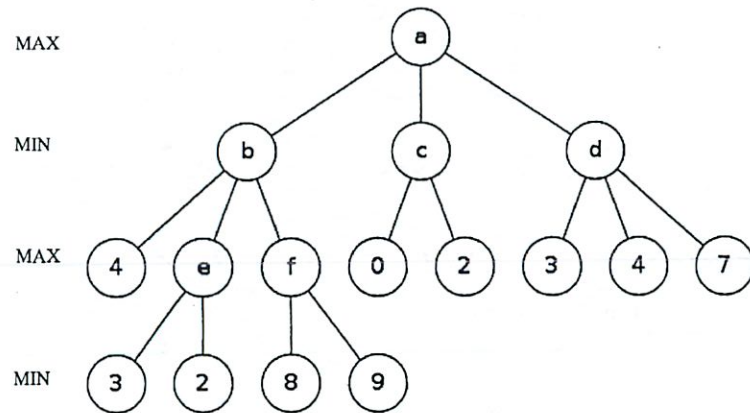
Would your answer for Part A1 be the same for all possible $S(\text{node})$ values? Yes No

Explain your reasoning in less than 4 meaningful sentences or give a short proof.

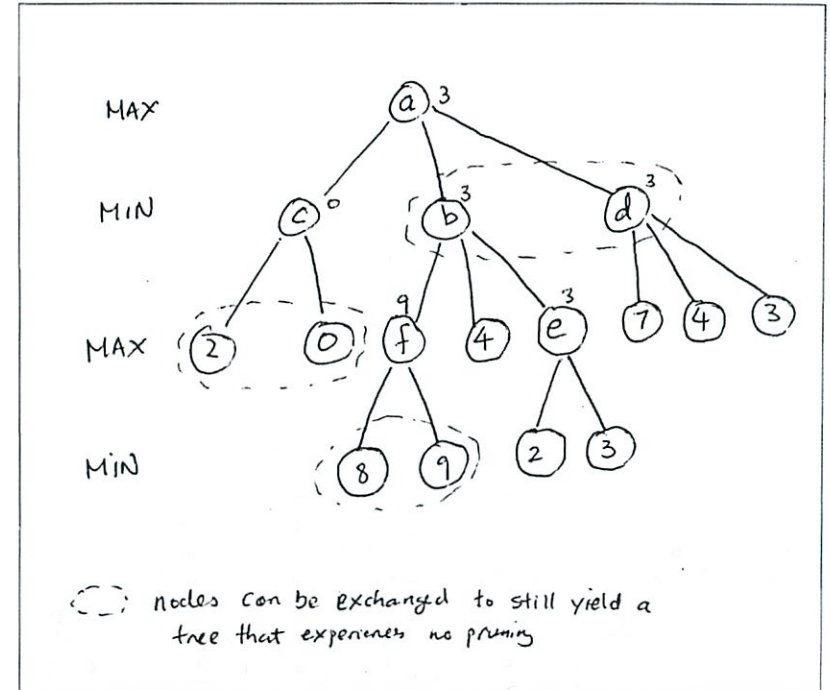
Applying $S(x) = 42 \cdot x + 1000$ to all 3 constraints in A3 does not change the direction of the inequalities. Hence preserving the optimal primal constraints.
 But $S(x) = -42 \cdot x + 1000$, multiplies -1 to both sides of the inequalities resulting in a flip of the direction, which breaks the pruning conditions.

Part B1: Digging Progressively Deeper (10 points)

You decide to put your 6.034 knowledge to good use by entering an adversarial programming contest. In this contest, one player (your opponent) is tasked with optimizing the running time of alpha beta search with progressive deepening. Your goal, as her adversary, is to slow down her algorithm. Remembering a key insight from 6.034: node order affects the amount of alpha-beta pruning, you decide to do the exact opposite: you decide to reorder your opponent's search tree so as to eliminate alpha beta pruning. To practice, you perform your anti-optimization on the following tree.

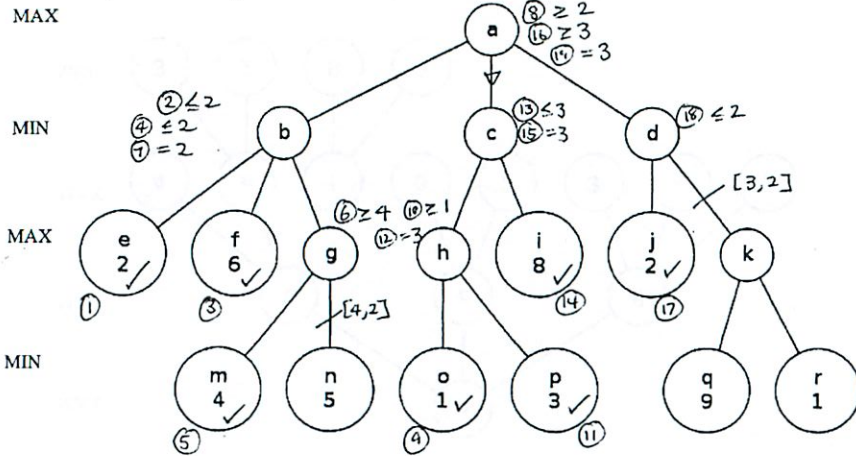


Reorder the nodes of the tree at every level such that alpha-beta search does **no pruning**. You may only reorder nodes (you cannot reattach a node to a different parent). Show your reordered tree below:



Part B2 (5 points)

For the tree given below (not the same tree as in B1), perform alpha beta search. Do your work on the tree as given; do no reordering. List the leaf nodes (letter) in the order that they are statically evaluated in the box below the tree.

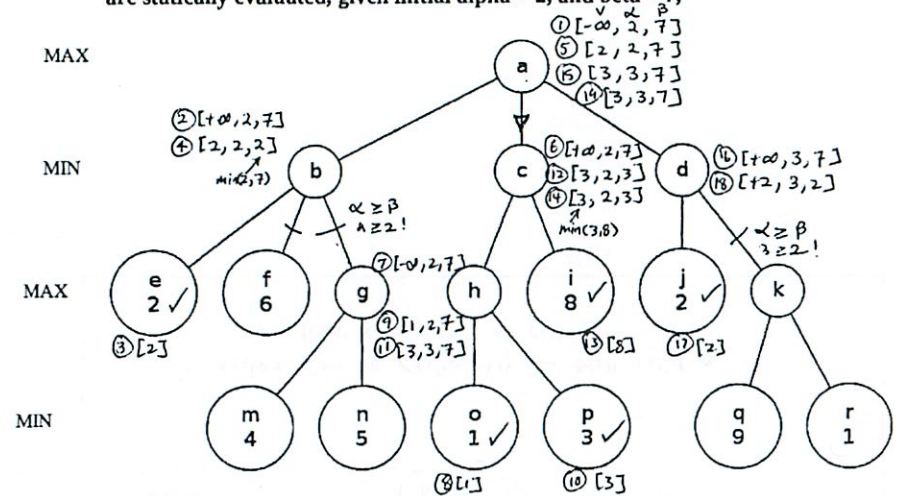


e f m o p i j

-2 pts for each error

Part B3 (10 points)

Now, you are to repeat your alpha beta search, but this time with initial values for alpha = 2 and beta = 7. List, in the box below the tree, the leaf nodes in the order that they are statically evaluated, given initial alpha = 2, and beta = 7,



7 pts (-2 for each error)

e o p i j

What is the best move according to the search with alpha = 2 and beta = 7? 3 pts

c

They don't write α, β !
Grr

I could also use the \leq framework

Problem 2: Time Travelers' Convention (50 points)

The MIT Time Travel Society (MITTS) has invited seven famous historical figures to each give a lecture at the annual MITTS convention, and you've been asked to create a schedule for them. Unfortunately, there are only four time slots available, and you discover that there are some restrictions on how you can schedule the lectures and keep all the convention attendees happy. For instance, physics students will be disappointed if you schedule Niels Bohr and Isaac Newton to speak during the same time slot, because those students were hoping to attend both of those lectures.

After talking to some students who are planning to attend this year's convention, you determine that they fall into certain groups, each of which wants to be able to see some subset of the time-traveling speakers. (Fortunately, each student identifies with at most one of the groups.) You write down everything you know:

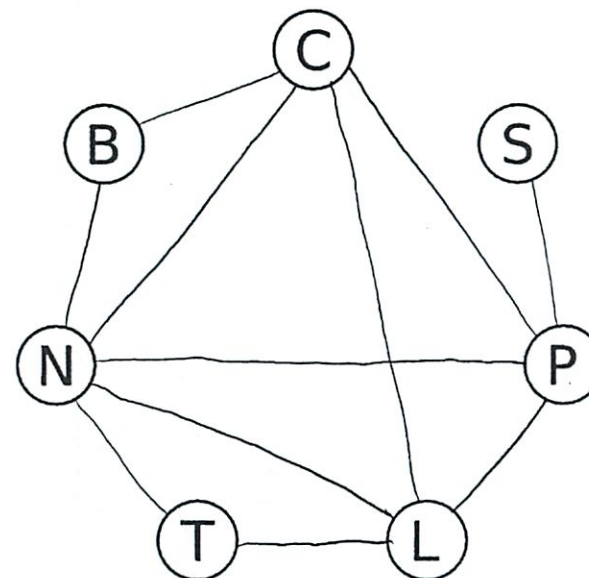
The list of guest lecturers consists of Alan Turing, Ada Lovelace, Niels Bohr, Marie Curie, Socrates, Pythagoras, and Isaac Newton.

- 1) Turing has to get home early to help win World War II, so he can only be assigned to the 1pm slot.
- 2) The Course VIII students want to see the physicists: Bohr, Curie, and Newton.
- 3) The Course XVIII students want to see the mathematicians: Lovelace, Pythagoras, and Newton.
- 4) The members of the Ancient Greece Club wants to see the ancient Greeks: Socrates and Pythagoras.
- 5) The visiting Wellesley students want to see the female speakers: Lovelace and Curie.
- 6) The CME students want to see the British speakers: Turing, Lovelace, and Newton.
- 7) Finally, you decide that you will be happy if and only if you get to see both Curie and Pythagoras. (Yes, even if you belong to one or more of the groups above.)

Part A (5 points)

That's a lot of preferences to keep track of, so you decide to draw a diagram to help make sense of it all. Draw a line between the initials of each pair of guests who must not share the same time slot.

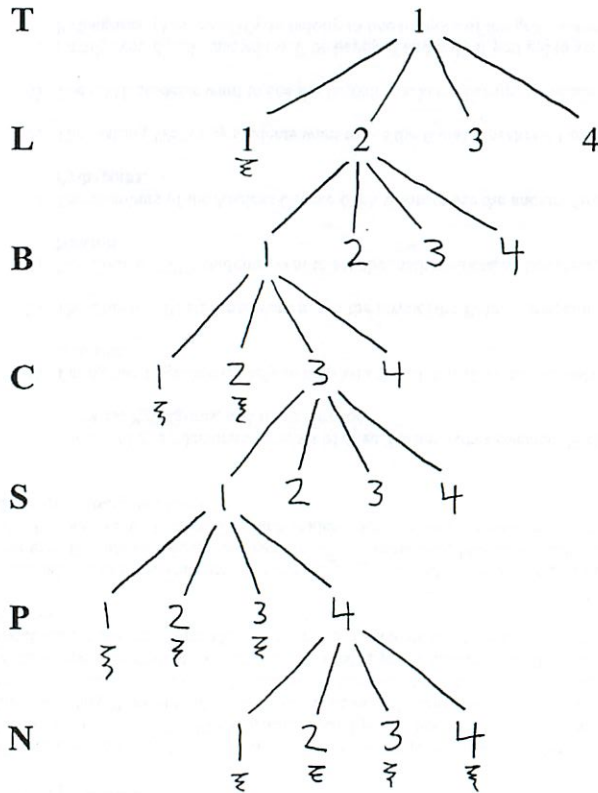
This diagram is repeated on the tear off sheet.



Part B (15 points)

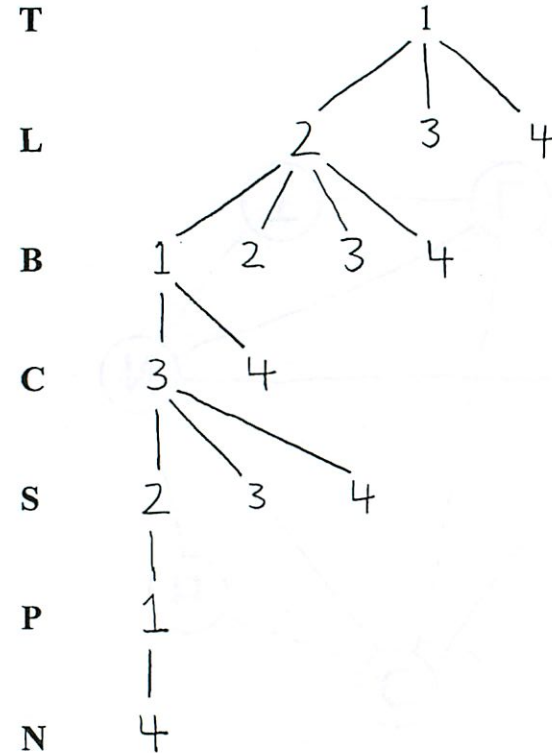
You decide to first assign the time slots (which conveniently happen to be 1, 2, 3, and 4 pm) by using a **depth-first search with no constraint propagation**. The only check is to be sure each new assignment violates no constraint with any previous assignment. As a tiebreaker, assign a lecturer to the earliest available time slot (so as to get them back to their own historical eras as soon as possible).

In the tree below, Alan Turing has already been scheduled to speak at 1 pm, in accordance with constraint #1. Continue filling in the search tree up to the first time you try (and fail) to assign a time slot to Isaac Newton, at which point you give up in frustration and move on to Part C in search of a more sophisticated approach.



Part C (20 points)

You're not fond of backtracking, so rather than wait and see just how much backtracking you'll have to do, you decide to start over and use **depth-first search with forward checking** (constraint propagation through domains reduced to size 1). As before, your tiebreaker is to assign the earliest available time slot.



What is the final lecture schedule you hand in to MITTTS?

1 pm: T, P, B

2 pm: S, L,

3 pm: C

4 pm: N

Part D (10 points)

Now, rather than backtracking, you're concerned about the amount of time it takes to keep track of all those domains and propagate constraints through them. You decide that the problem lies in the ordering of the guest list. Just then, you get a call from the MITTTS president, who informs you that Alan Turing's schedule has opened up and he is now free to speak during any one of the four time slots.

Armed with this new information, you reorder the guest list to maximize your chances of quickly finding a solution. In particular, which lecturer do you now assign a time slot to first, and why?

Newton, because he has the most constraints.

10/25
Practice

6.034 Quiz 2 20 October 2010

Name	
email	

Circle your TA and recitation time (**for 1 point**), so that we can more easily enter your score in our records and return your quiz to you promptly.

TAs
Martin Couturier
Kenny Donahue
Charles Watts
Gleb Kuznetsov
Kendra Pugh
Mark Seifter
Yuan Shen

Thu	
Time	Instructor
1-2	Bob Berwick
2-3	Bob Berwick
3-4	Bob Berwick

Fri	
Time	Instructor
1-2	Randall Davis
2-3	Randall Davis
3-4	Randall Davis

Problem number	Maximum	Score	Grader
1	50		
2	50		
Total	100		

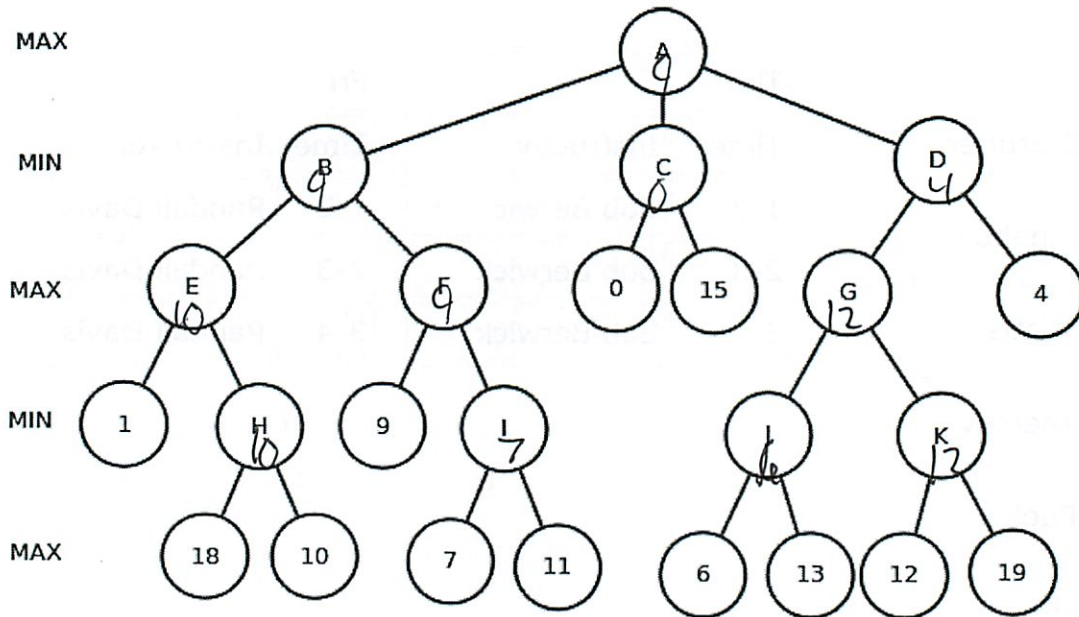
There are 15 pages in this quiz, including this one, but not including blank pages and tear-off sheets. Tear-off sheets are provided at the end with duplicate drawings and data. As always, open book, open notes, open just about everything, including a calculator, but no computers.

Problem 1: Games (50 points)

Part A: Minimax (10 points)

A1: Perform Minimax (5 points)

Perform the minimax algorithm, without alpha-beta, on this tree. Write the minimax value at each node.



A2: Rotate the Tree (5 points)

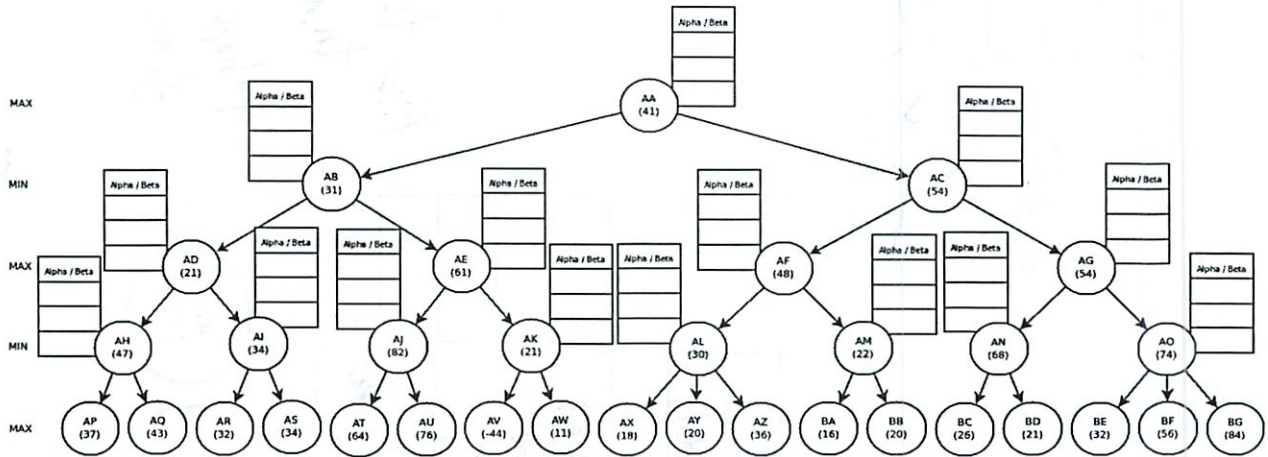
Using the minimax calculations from part A1, without performing any alpha-beta calculation, rotate the children of each node in the above tree at every level to ensure maximum alpha-beta pruning.

For max put max values 1st
 - " min " min " "
 - actually do

could do
 this for
 A, B
 then confirm

Part B: Alpha Beta (30 points)

While playing a game, you find yourself in the situation indicated by the following tree.



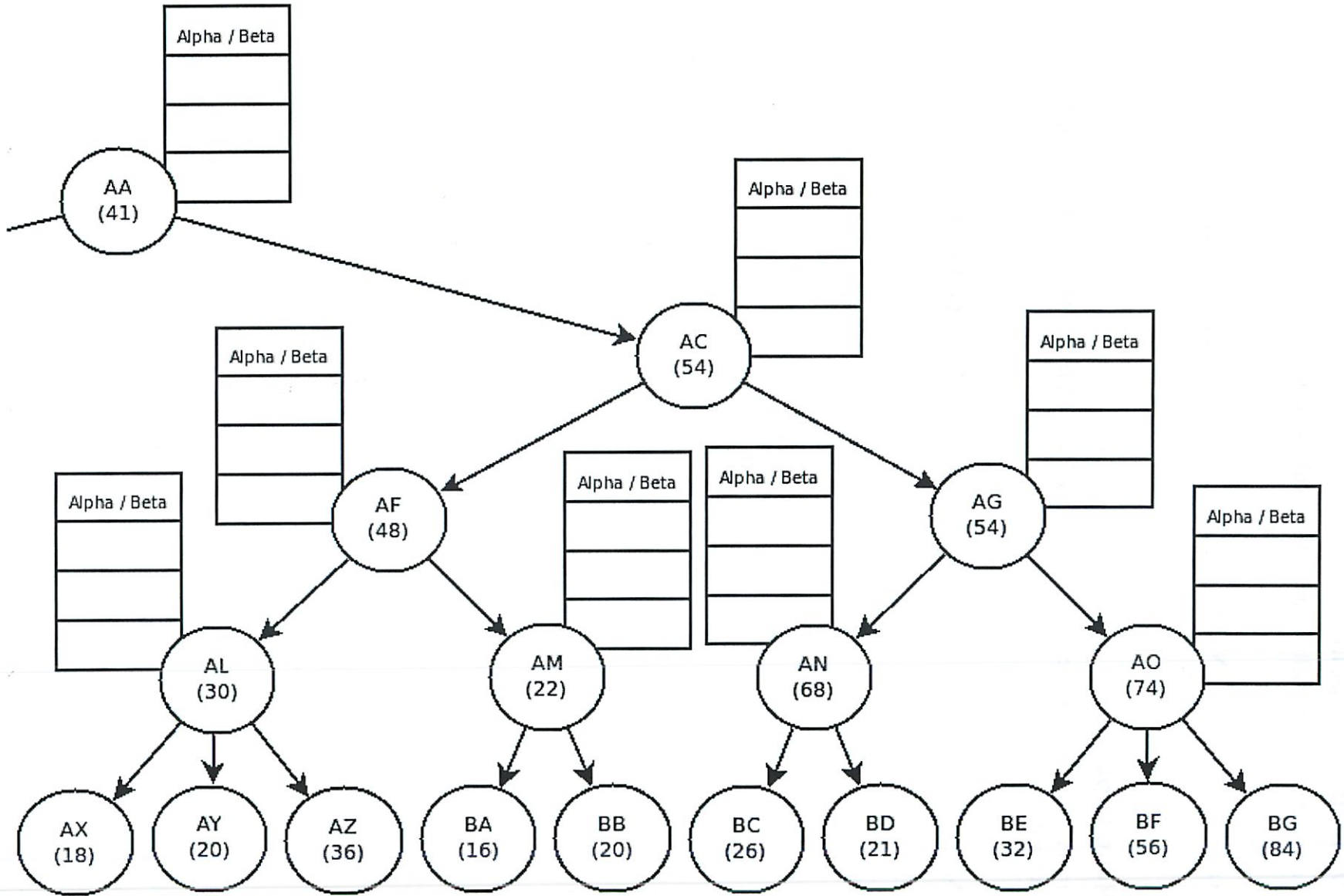
Because this tree is difficult to read, the the tree has been broken into two subtrees, shown on the next two pages. Do all your work on those subtrees, not the tree shown above. The tree above is merely meant to show how the two subtrees connect. Note that the root node AA is shown in both halves.

B1 The Game Tree (7 points)

You are provided with static values at all the nodes. **In this part of the problem, you are to ignore the static values at intermediate notes.** You are to use only the values on the leaf nodes.

Perform the alpha-beta algorithm on the graph tree on the next two pages.

- 1) Use the boxes to help you perform your alpha-beta calculations as needed.
- 2) **Draw an 'X' through any branch** that is blocked from further consideration by an alpha-beta calculation.
- 3) Assume no progressive deepening



B2 Evaluations (20 points)

How many nodes were statically evaluated and which were they?

# Evals _____	Nodes: _____
---------------	--------------

B3 Move (3 points)

What move is chosen as the best? What node in the deepest level is the maximizing player trying to move towards?

AA--> _____	Moving Toward: _____
-------------	----------------------

Part C: Progressive Deepening (10 points)

Assume you have the same setup as in Part B, except this time, you have an **5-second time-limit for each move**. Each static evaluation takes **1 second**. Assume there is no time associated with building the alpha-beta tree and no time associated with performing the alpha beta search. You are to use progressive deepening and alpha-beta together.

Before performing alpha-beta at any level, use all you know from previous calculations to reorder the nodes at **ALL higher levels** as much as possible. Naturally, for any reordering, a value produced by search from static values lower down is better than a static value or a value produced by a less-deep search. Assume reordering takes zero time.

Assume that the static values calculated are those shown in the diagram. You are to show us what work would be done by the maximizer at AA during the 5 seconds available to move.

C1 The tree (4 points)

Draw the reordered tree of evaluated nodes.

C2 Evaluations (4 points)

How many nodes were statically evaluated during the 5 seconds available to the maximizer at AA. In which order were they evaluated?

Evals _____ Nodes: _____

C3 Move (2 points)

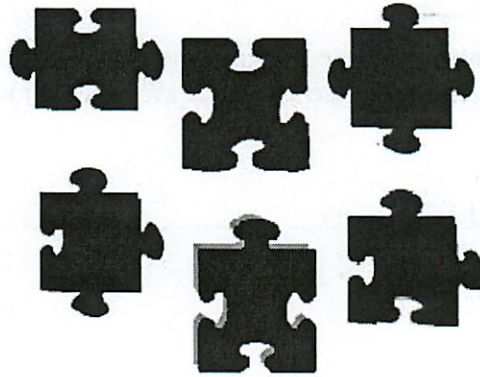
What move is chosen as the best during the available 5 seconds? What is the deepest node that we are trying to move toward? Use all the static evaluation values that the maximizer is able to calculate to make your decision.

AA--> _____ Moving Toward: _____

Problem 2: Constraint Propagation(50 points)

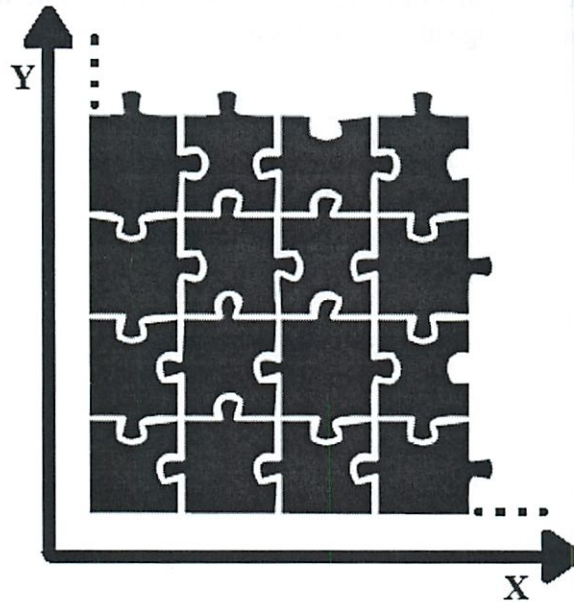
Two of your TA's, Martin and Kenny were given the task of putting together two 20x10, 200 piece jigsaw puzzles. Unfortunately, a child has painted all the pieces black.

The following shapes are **representative** of the shapes of interior pieces (thus, the shapes shown are not all of the shapes involved). Note that each side of the interior pieces has a concavity or a protrusion. In spite of our limited drawing skills, there is just one protrusion shape and one concavity shape, so any protrusion will fit into any concavity.



Edge pieces are like interior pieces except that they have one flat side. Corner pieces are like interior pieces except that they have two adjacent flat sides.

Because all pieces are the same size, the final puzzle can be represented as a grid of (X,Y) coordinates, where each set of coordinates is a location of a puzzle piece. The puzzle is 20 pieces wide by ten pieces tall.



Part A: Setting Constraints (15 points)

While Kenny sets out to solve the puzzle in the typical fashion, Martin sees that this is just another constraint problem and sets out to solve it using what he learned while taking 6.034:

1. He decides that the puzzle locations are the problem variables.
2. He decides to treat all 200 pieces as values (thus, **the pieces, not their shapes, are values**).
3. **He notes that an oracle has put all 200 pieces in their correct orientation** (thus, each physical piece is one value, not four. Rotation is not allowed to simplify your work).
4. He devises this list of constraints:
 - a) Piece locations around the edge of the puzzle must contain a piece with one flat edge that is facing outward from the center of the puzzle.
 - b) Piece locations at the corners must contain a piece with two flat edges that are facing outward from the center of the puzzle.
 - c) Piece locations not around the edge of the puzzle must not contain a puzzle piece with any flat edges
 - d) All four sides of a puzzle piece must be compatible with all of the pieces around it.
 - e) No two locations can contain the same piece.

A1 (6 points)

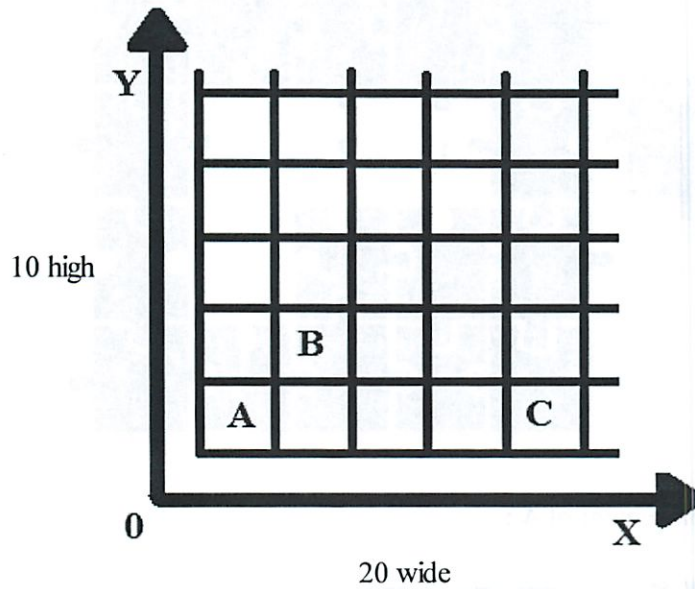
Draw a graph of the variable nodes for the subset of the puzzle containing the 9 piece locations shown below. Indicate constraints between pairs of variable nodes by drawing **one line for each** such pairwise constraint. Suggestion: arrange all your nodes in a circle.

1	2	3
4	5	6
7	8	9



A2 (9 points)

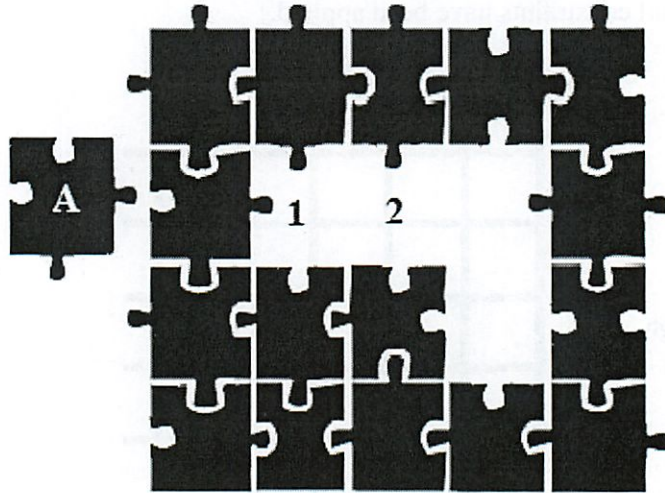
Thinking that constraints 1-4 can serve as a good starting point to solve any puzzle, and remembering that oracle has spoken, so **all pieces are provided in the correct orientation**, Martin decides to use the constraints to simplify the starting domains for each of the piece locations. Describe the domains for A, B, C after the initial constraints have been applied.



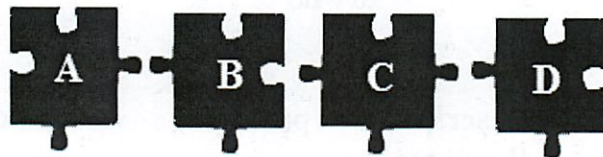
Piece Location	Description of pieces in the domain	Size of domain
A		
B		
C		

Part B: Checking Neighbors (8 points)

While solving the puzzle, Martin places piece A in location 1. He is running the Domain Reduction Algorithm using **forward checking**. The domain of location 2 is shown as it was before A was placed.



Domain of location 2 before placement of A :

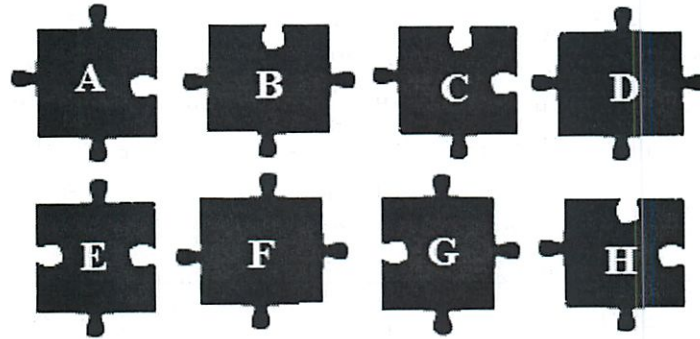
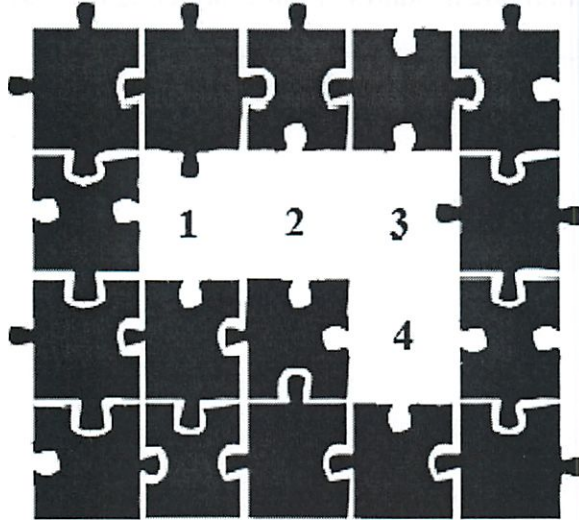


Again, noting that no rotation is allowed, and that, all protrusions fit successfully into all concavities in spite of our limited drawing skills, what happens after he places piece A and why?

Part C: Neighbors of Neighbors (27 points)

C1 (5 points)

Nearing completion of the puzzle, Martin encounters a cluster of 4 empty piece locations. (there are 4 other empty spaces elsewhere). He still has a total of 8 pieces. Considering only constraints imposed by the neighbors of the 4 empty piece locations shown, fill in the domain table for each of the piece locations. Once again note that **all pieces are shown in the only orientation allowed**,



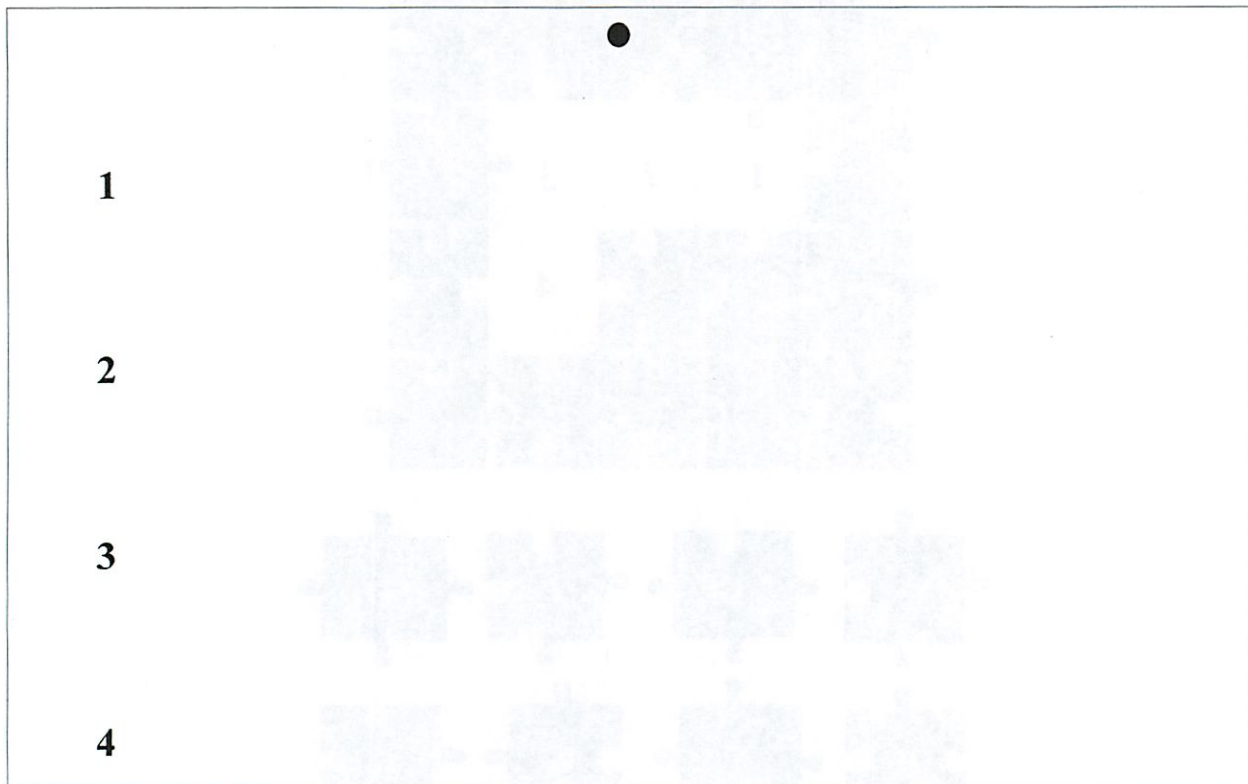
Variable	Domain
1	
2	
3	
4	

C2 (10 points)

Once again noting that all pieces are shown in the only orientation allowed, complete this section of the puzzle using **depth-first search with forward checking and propagation through singleton domains**. Again assume that, all protrusions fit successfully into all concavities.

Assigning the variables, 1, 2, 3, 4, in numerical order, draw the search tree that is evaluated using forward checking with **singleton propagation**. **Be sure to try assignments in lexicographic order.**

Work carefully, because we will award points only for the fraction of the search done right, so early mistakes will be worse than late mistakes or not completing the search.



C3 (4 points)

What is the final solution to this region of the puzzle?

Variable	Value
1	
2	
3	
4	

C4 (3 points)

List the partial paths in your tree where constraint checking fails.

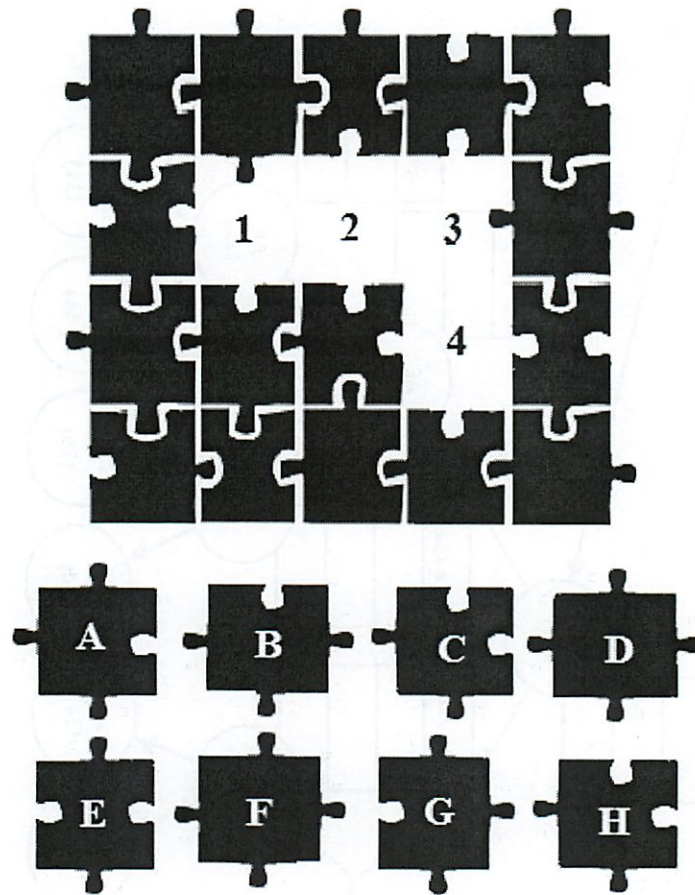
C5 (5 points)

Looking at your answer to C1, how might we adjust the search to find solutions with less backtracking?

Blank page

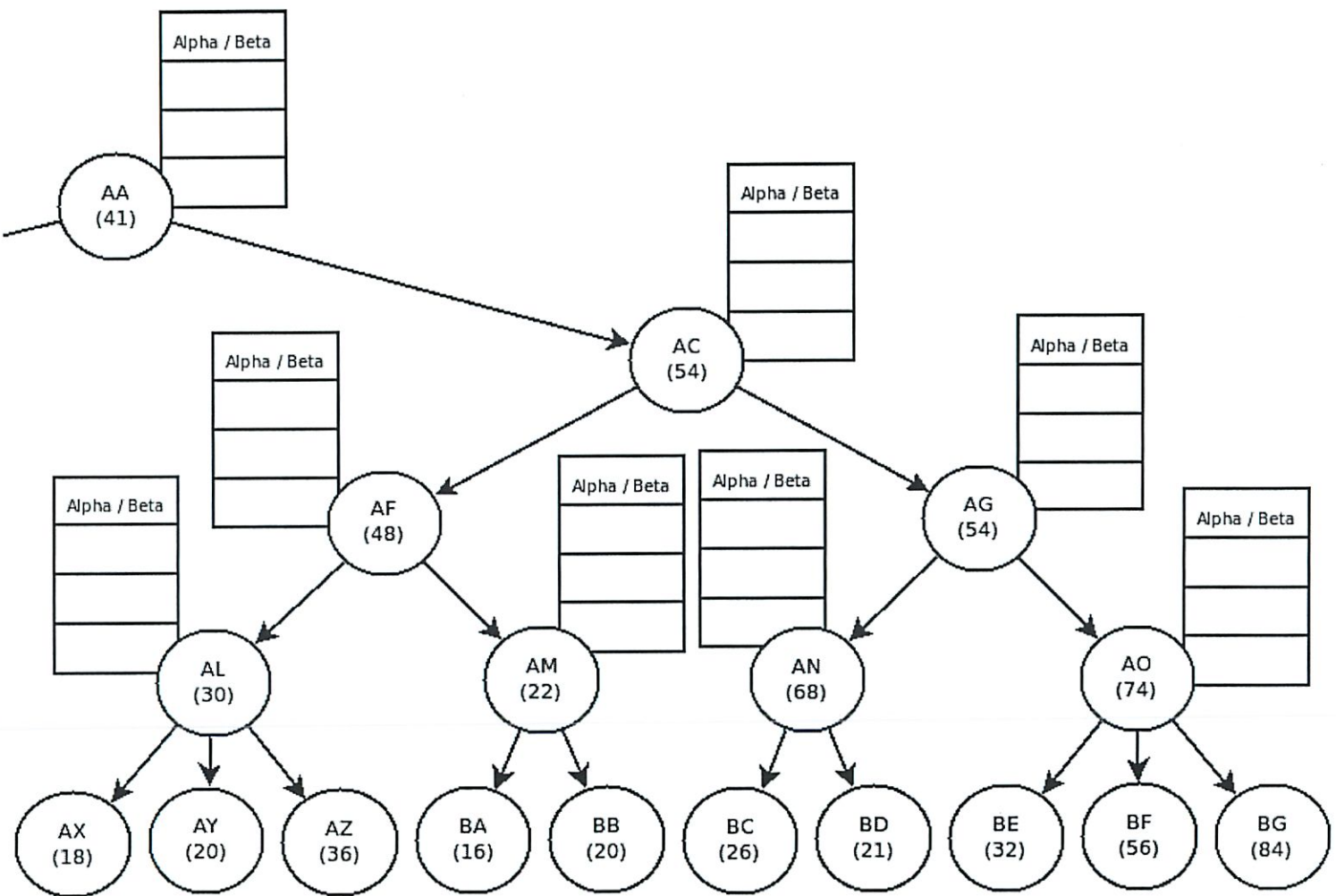
Tear off sheet, you need not hand this in

Problem 2, Part B



Tear off sheet, you need not hand this in

Problem 1, Part B, Right Side



Blank, final page

of this board has been only available to the

of this board has been only available to the



6.034 Quiz 2
20 October 2010

Name NORBERT WIENER
email _____

Circle your TA and recitation time (for 1 point), so that we can more easily enter your score in our records and return your quiz to you promptly.

TAs	Thu	Fri
Martin Couturier	Time Instructor	Time Instructor
Kenny Donahue	1-2 Bob Berwick	1-2 Randall Davis
Charles Watts	2-3 Bob Berwick	2-3 Randall Davis
Gleb Kuznetsov	3-4 Bob Berwick	3-4 Randall Davis
Kendra Pugh		
Mark Seifter		
Yuan Shen		

Problem number	Maximum	Score	Grader
1	50		
2	50		
Total	100		

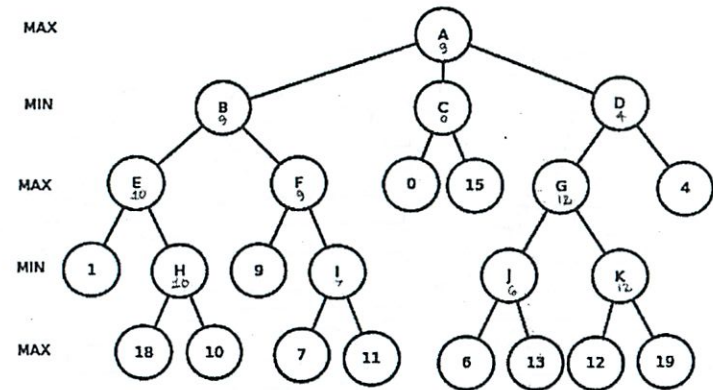
There are 15 pages in this quiz, including this one, but not including blank pages and tear-off sheets. Tear-off sheets are provided at the end with duplicate drawings and data. As always, open book, open notes, open just about everything, including a calculator, but no computers.

Problem 1: Games (50 points)

Part A: Minimax (10 points)

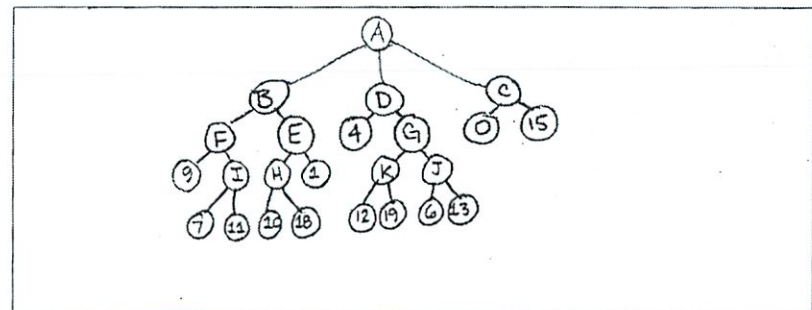
A1: Perform Minimax (5 points)

Perform the minimax algorithm, without alpha-beta, on this tree. Write the minimax value at each node.



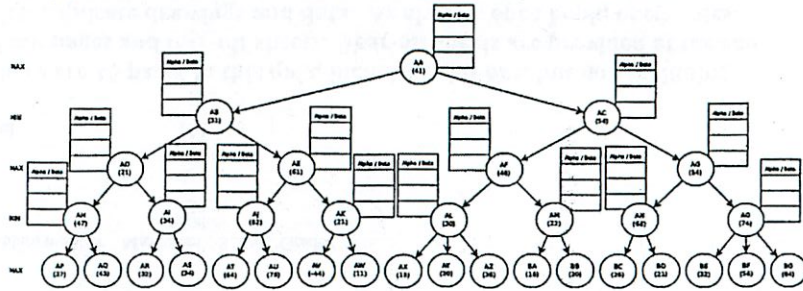
A2: Rotate the Tree (5 points)

Using the minimax calculations from part A1, without performing any alpha-beta calculation, rotate the children of each node in the above tree at every level to ensure maximum alpha-beta pruning.



Part B: Alpha Beta (30 points)

While playing a game, you find yourself in the situation indicated by the following tree.



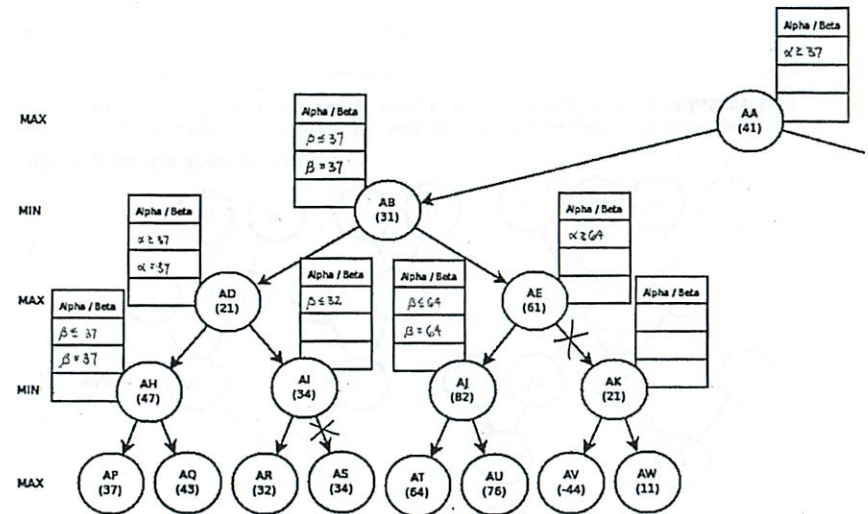
Because this tree is difficult to read, the tree has been broken into two subtrees, shown on the next two pages. Do all your work on those subtrees, not the tree shown above. The tree above is merely meant to show how the two subtrees connect. Note that the root node AA is shown in both halves.

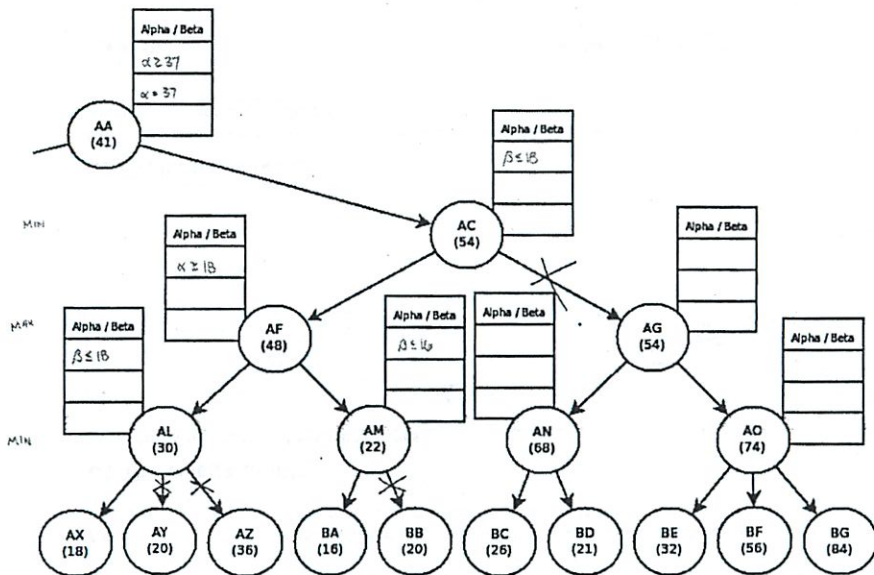
B1 The Game Tree (7 points)

You are provided with static values at all the nodes. In this part of the problem, you are to ignore the static values at intermediate nodes. You are to use only the values on the leaf nodes.

Perform the alpha-beta algorithm on the graph tree on the next two pages.

- 1) Use the boxes to help you perform your alpha-beta calculations as needed.
- 2) Draw an 'X' through any branch that is blocked from further consideration by an alpha-beta calculation.
- 3) Assume no progressive deepening





B2 Evaluations (20 points)

How many nodes were statically evaluated and which were they?

Evals 7 Nodes: AP, AQ, AR, AT, AU, AX, BA

B3 Move (3 points)

What move is chosen as the best? What node in the deepest level is the maximizing player trying to move towards?

AA \rightarrow AB Moving Toward: AP

50/50

Part C: Progressive Deepening (10 points)

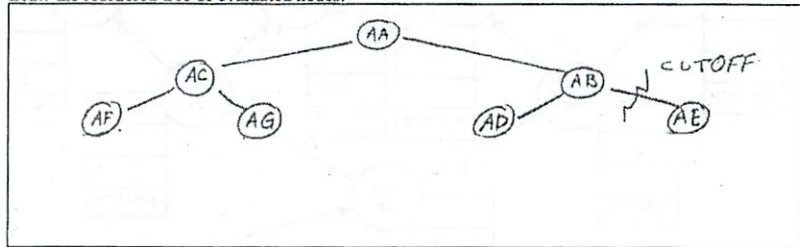
Assume you have the same setup as in Part B, except this time, you have an 5-second time-limit for each move. Each static evaluation takes 1 second. Assume there is no time associated with building the alpha-beta tree and no time associated with performing the alpha beta search. You are to use progressive deepening and alpha-beta together.

Before performing alpha-beta at any level, use all you know from previous calculations to reorder the nodes at ALL higher levels as much as possible. Naturally, for any reordering, a value produced by search from static values lower down is better than a static value or a value produced by a less-deep search. Assume reordering takes zero time.

Assume that the static values calculated are those shown in the diagram. You are to show us what work would be done by the maximizer at AA during the 5 seconds available to move.

C1 The tree (4 points)

Draw the reordered tree of evaluated nodes.



C2 Evaluations (4 points)

How many nodes were statically evaluated during the 5 seconds available to the maximizer at AA. In which order were they evaluated?

Evals 5 Nodes: AB, AC, AF, AG, AD

C3 Move (2 points)

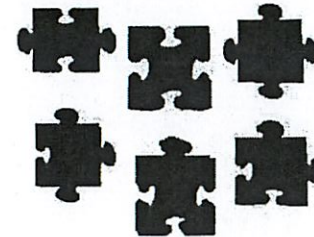
What move is chosen as the best during the available 5 seconds? What is the deepest node that we are trying to move toward? Use all the static evaluation values that the maximizer is able to calculate to make your decision.

AA → AC Moving Toward: AF

Problem 2: Constraint Propagation(50 points)

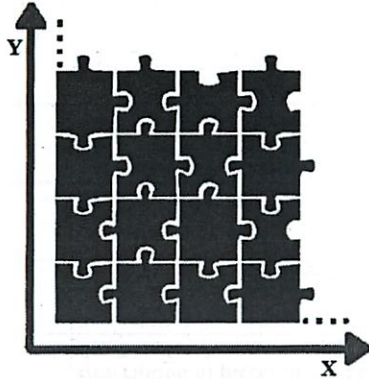
Two of your TA's, Martin and Kenny were given the task of putting together two 20x10, 200 piece jigsaw puzzles. Unfortunately, a child has painted all the pieces black.

The following shapes are **representative** of the shapes of interior pieces (thus, the shapes shown are not all of the shapes involved). Note that each side of the interior pieces has a concavity or a protrusion. In spite of our limited drawing skills, there is just one protrusion shape and one concavity shape, so any protrusion will fit into any concavity.



Edge pieces are like interior pieces except that they have one flat side. Corner pieces are like interior pieces except that they have two adjacent flat sides.

Because all pieces are the same size, the final puzzle can be represented as a grid of (X,Y) coordinates, where each set of coordinates is a location of a puzzle piece. The puzzle is 20 pieces wide by ten pieces tall.



Part A: Setting Constraints (15 points)

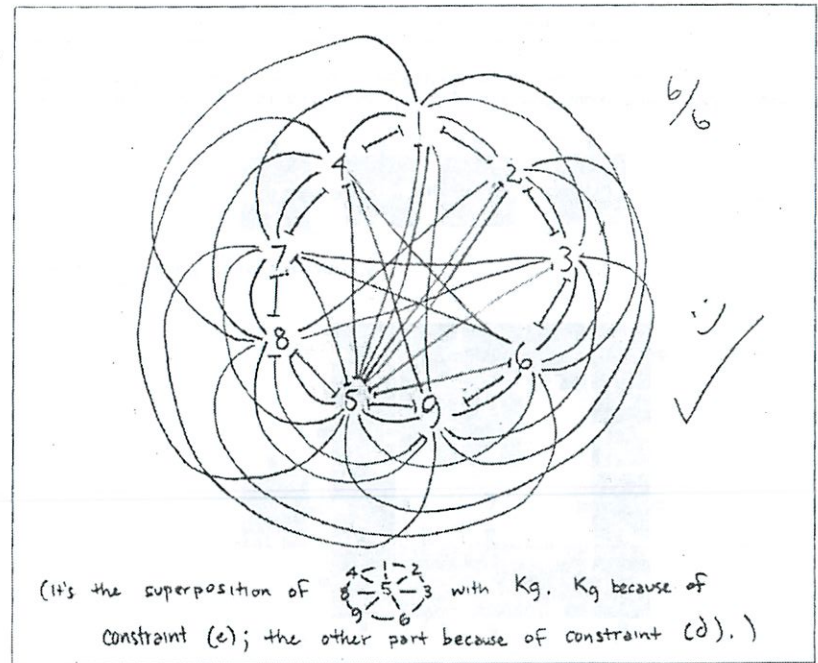
While Kenny sets out to solve the puzzle in the typical fashion, Martin sees that this is just another constraint problem and sets out to solve it using what he learned while taking 6.034:

1. He decides that the puzzle locations are the problem variables.
2. He decides to treat all 200 pieces as values (thus, the pieces, not their shapes, are values).
3. He notes that an oracle has put all 200 pieces in their correct orientation (thus, each physical piece is one value, not four. Rotation is **NOT** allowed to simplify your work).
4. He devises this list of constraints:
 - a) Piece locations around the edge of the puzzle must contain a piece with one flat edge that is facing outward from the center of the puzzle.
 - b) Piece locations at the corners must contain a piece with two flat edges that are facing outward from the center of the puzzle.
 - c) Piece locations not around the edge of the puzzle must not contain a puzzle piece with any flat edges
 - d) All four sides of a puzzle piece must be compatible with all of the pieces around it.
 - e) No two locations can contain the same piece.

A1 (6 points)

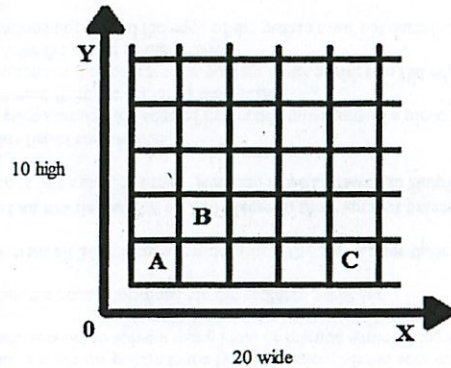
Draw a graph of the variable nodes for the subset of the puzzle containing the 9 piece locations shown below. Indicate constraints between pairs of variable nodes by drawing one line for each such pairwise constraint. Suggestion: arrange all your nodes in a circle.

1	2	3
4	5	6
7	8	9





A2 (9 points)

Thinking that constraints 1-4 can serve as a good starting point to solve any puzzle, and remembering that oracle has spoken, so all pieces are provided in the correct orientation, Martin decides to use the constraints to simplify the starting domains for each of the piece locations. Describe the domains for A, B, C after the initial constraints have been applied.



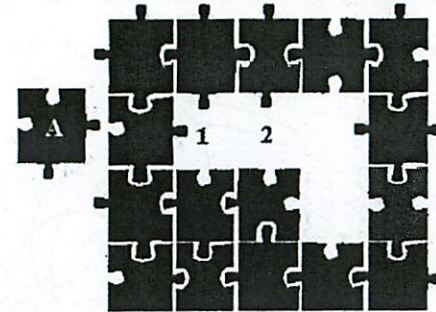
9/9

$18 \cdot 8 = 144$

Piece Location	Description of pieces in the domain	Size of domain
A	the corner piece oriented like  (first edges left & bottom)	1 ✓
B	All interior pieces ✓	144 ✓
C	the edge pieces oriented like  (first edge bottom)	18 ✓

Part B: Checking Neighbors (8 points)

While solving the puzzle, Martin places piece A in location 1. He is running the Domain Reduction Algorithm using forward checking. The domain of location 2 is shown as it was before A was placed.



Domain of location 2 before placement of A :



8/8

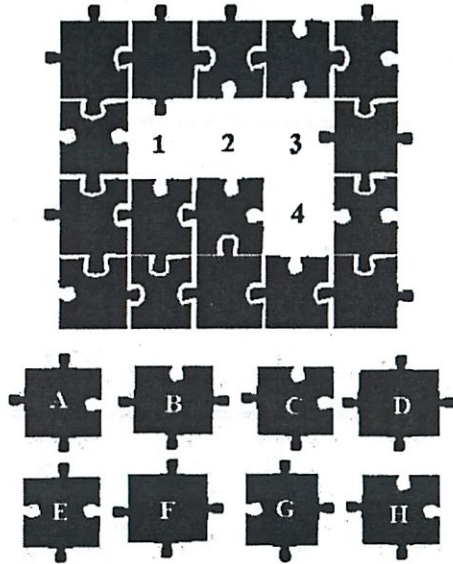
Again, noting that no rotation is allowed, and that, all protrusions fit successfully into all concavities in spite of our limited drawing skills, what happens after he places piece A and why?

The domain of location 2 is reduced to \emptyset , ✓ because none of $\{B, C, D\}$ can attach to the right side of A. So he must take piece A back out. ✓

Part C: Neighbors of Neighbors (27 points)

C1 (5 points)

Nearing completion of the puzzle, Martin encounters a cluster of 4 empty piece locations. (there are 4 other empty spaces elsewhere). He still has a total of 8 pieces. Considering only constraints imposed by the neighbors of the 4 empty piece locations shown, fill in the domain table for each of the piece locations. Once again note that all pieces are shown in the only orientation allowed,



5/5

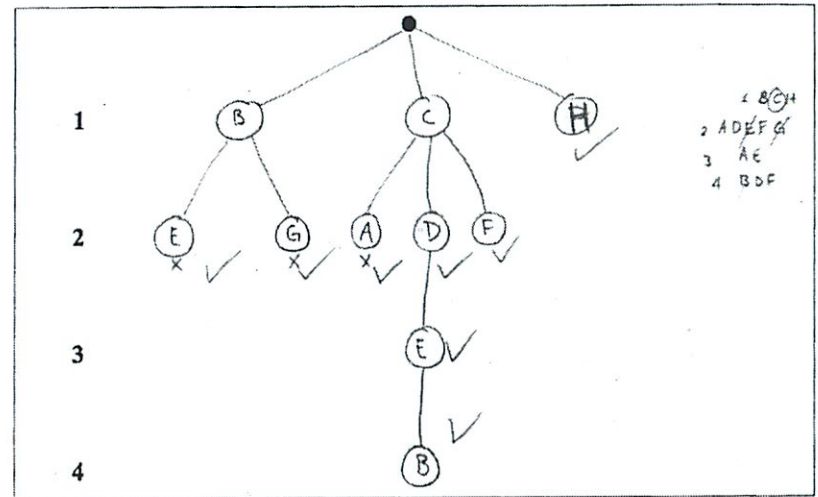
Variable	Domain
1	B, C, H ✓
2	A, D, E, F, G ✓
3	A, E ✓
4	B, D, F ✓

C2 (10 points)

Once again noting that all pieces are shown in the only orientation allowed, complete this section of the puzzle using depth-first search with forward checking and propagation through singleton domains. Again assume that, all protrusions fit successfully into all concavities.

Assigning the variables, 1, 2, 3, 4, in numerical order, draw the search tree that is evaluated using forward checking with singleton propagation. Be sure to try assignments in lexicographic order.

Work carefully, because we will award points only for the fraction of the search done right, so early mistakes will be worse than late mistakes or not completing the search.



C3 (4 points)

What is the final solution to this region of the puzzle?

Variable	Value
1	C ✓
2	D ✓
3	E ✓
4	B ✓

C4 (3 points)

List the partial paths in your tree where constraint checking fails.

B E ✓
B G ✓
C A ✓

C5 (5 points)

Looking at your answer to C1, how might we adjust the search to find solutions with less backtracking?

Order the positions by how constrained they are,
most to least:
3, 1, 4, 2. ✓

6.034 Quiz 2

26 October 2011

Name	Michael Plasencia
email	theplaz@mit.edu

Circle your TA and recitation time (**for 1 extra credit point**), so that we can more easily enter your score in our records and return your quiz to you promptly.

TAs		Thu	Fri
		Time	Instructor
Avril Kenney	Adam Mustafa	1-2	Bob Berwick
Blondie Chaplin	Erek Speed	2-3	Bob Berwick
Gary Planthaber	Caryn Krakauer	3-4	Bob Berwick
Peter Brin	Tanya Kortz		

+1

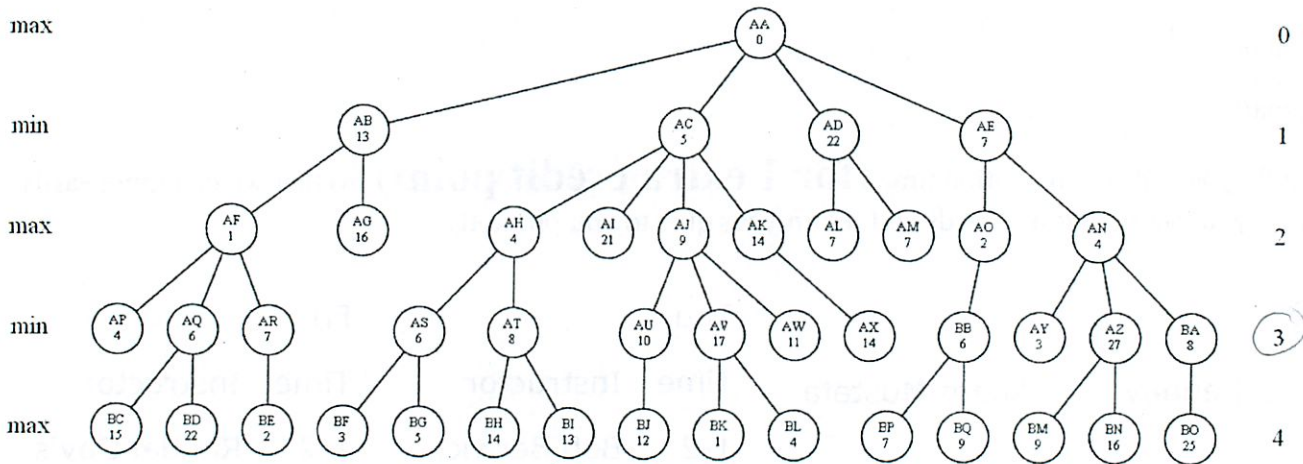
Correction

Problem number	Maximum	Score	Grader
1	40	20	CEK
2	40	21	ES
3	20	0	P
Total	100	42	AFK

There are 11 pages in this quiz, including this one, but not including blank pages and tear-off sheets. A tear-off sheet is provided at the end with duplicate drawings and data. As always, open book, open notes, open just about everything, including a calculator, but no computers.

Problem 1: Games (40 points)

This part makes reference to the following tree. Each node has a label, and a number which represents the value returned by the static evaluator at that node. The numbers on the right indicate ply.



The tree is reprinted **in larger scale** for your convenience at the end of the test.

Part A: It's Your Move (15 points)

It's your move as the maximizing player. You plan to use the minimax algorithm without alpha-beta pruning to find your move, but you only have time to go at most three plies deep, so you do static evaluation at level 3 in the diagram (or at higher levels in parts of the tree that do not go all the way to level 3). What move do you choose?

next move only full tree AC → AH → AT

What is the value associated with that move?

How many nodes needed to be statically evaluated during the application of the minimax algorithm?

Part B: Your opponent responds (10 points)

Given the move you chose in A1, what is your opponent's move in response? Assume he also has time to go at most three plies beyond the move you have chosen in Part A, the move that takes you to ply 1.

AH

AT

What is the value of that move?

8

12

Part C: Improve your search (15 points)

You realize that you can use alpha-beta to improve the efficiency of your search. You start over on the problem, ignoring all the computation you did in Part A and Part B. Using the minimax algorithm with alpha-beta pruning you decide you can go to the bottom of the game tree, performing static evaluation at level 4 (or at higher levels in parts of the tree that do not go all the way to level 4). What is your best move now?

AC

Always the same

4 levels here
3 levels before

What is the value of that move?

8

List the nodes you statically evaluated in the order they were evaluated.

AP BC BD BE AG BF BH ~~BI~~ AL BP ~~AX~~ ~~AM~~ ~~AN~~ ~~AO~~

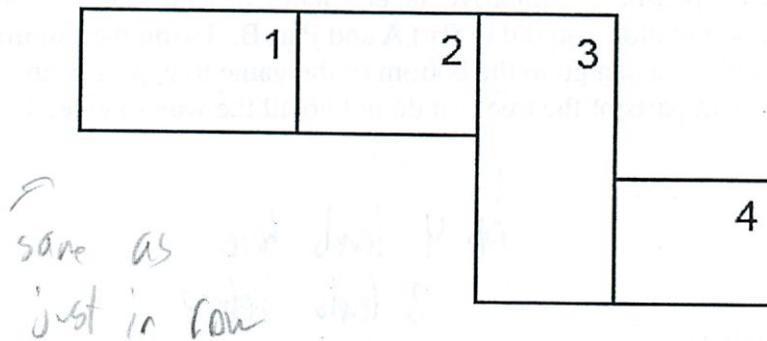
-10

Problem 2: Constraint Propagation (40 points)

In honor of MIT 150, MIT has decided to open a new zoo in Killian Court. They have obtained 7 animals and 4 enclosures. Because there are more animals than enclosures, some animals have to be in the same enclosure as others. However, the animals are very picky about who they live with. The MIT administration is having trouble assigning animals to enclosures, just as they often have trouble assigning students to residences. As you have taken 6.034, they have asked you to plan where each animal goes.

The animals chosen are a LION, ANTELOPE, HYENA, EVIL LION, HORNBILL, MEERKAT, and BOAR.

They have given you the plans of the zoo layout:



Each numbered area is a zoo enclosure. Multiple animals can go into the same enclosure, and not all enclosures have to be filled.

Each animal has restrictions about where it can be placed.

- The **LION** and the **EVIL LION** hate each other, and do not want to be in the same enclosure.
- The **MEERKAT** and **BOAR** are best friends, and have to be in the same enclosure.
- The **HYENA** smells bad. Only the EVIL LION will share his enclosure.
- The **EVIL LION** wants to eat the **MEERKAT, BOAR, and HORNBILL.**
- The **LION** and the **EVIL LION** want to eat the **ANTELOPE** so badly that the **ANTELOPE** cannot be in either the same enclosure with or in an enclosure adjacent to either the **LION** or **EVIL LION.**
- The **LION** annoys the **HORNBILL**, so the **HORNBILL** doesn't want to be in the **LION's** enclosure.
- The **LION** is the king, so he wants to be in enclosure 1.

Part A (5 points)

Ben is trying to solve the problem, but he didn't take 6.034 and refuses to believe that constraint propagation works. Because of this, he is determined to do a search with absolutely no domain reduction, constraint checking, or constraint propagation (he's not very smart). Ben only checks to see if an assignment is ok once all animals have been assigned to enclosures. What is the size of the tree that Ben will have to search through to find all solutions?

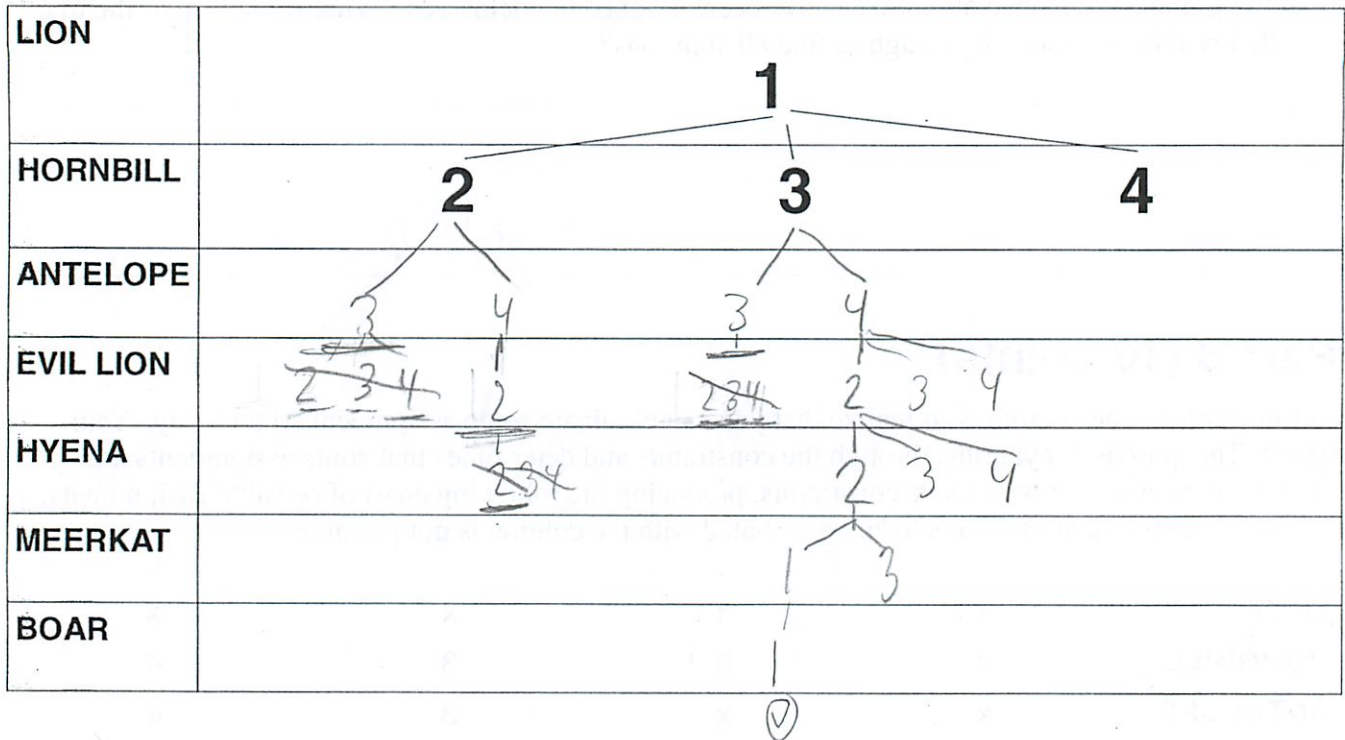
if he a
 y? if he assigns animals to enclosures, like
 but w/o cross ats

Part B (10 points)

Before starting your search, you realize that you can eliminate some assignments right away. Your friend, Theophilus Teeya, runs through the constraints and determines that some assignments are obviously inconsistent with those constraints, producing the following chart of possible assignments; x means the assignment of the enclosure associated with the column is not possible:

LION	1	x	x	x
HORNBILL	x	2	3	4
ANTELOPE	x	x	3	4
EVIL LION	x	2	3	4
HYENA	x	2	3	4
MEERKAT	1	2	3	4
BOAR	1	2	3	4

Assume Theophilus conclusions are correct; then, use the reduced domains found by Theophilus to find one solution by doing **forward checking, with propagation through domains reduced by any number of values**. Break ties in numerical order (1,2,3,4).



I am not writing single values where are correct on their level

-10

TA: It depends on method

They didn't draw t/B 1 - so clue

- well that was static eval

- this isn't

So draw w/ cross at

write all that are not statically eliminated

Part C (10 points)

Mercy Lestier, another friend, believes Theophilus could do more to reduce the domains **before** starting with forward checking. She is right. Demonstrate by crossing out **all** additional domain values that can be eliminated by repeated use the domain reduction algorithm.

LION	①	x	x	x
HORNBILL	x	2	3	4
ANTELOPE	x	x	3	④
EVIL LION	x	②	3	4
HYENA	x	②	3 - 2	4
MEERKAT	1	2	3	4
BOAR	1	2	3	4

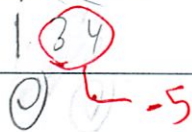
④ must be two and, for EL, who can't be in 1

Part D (10 points)

Using the reduced domains found in part C, find one solution by doing **forward checking**,

with propagation through domains reduced by **any** number of values. Break ties in numerical order (1,2,3,4).

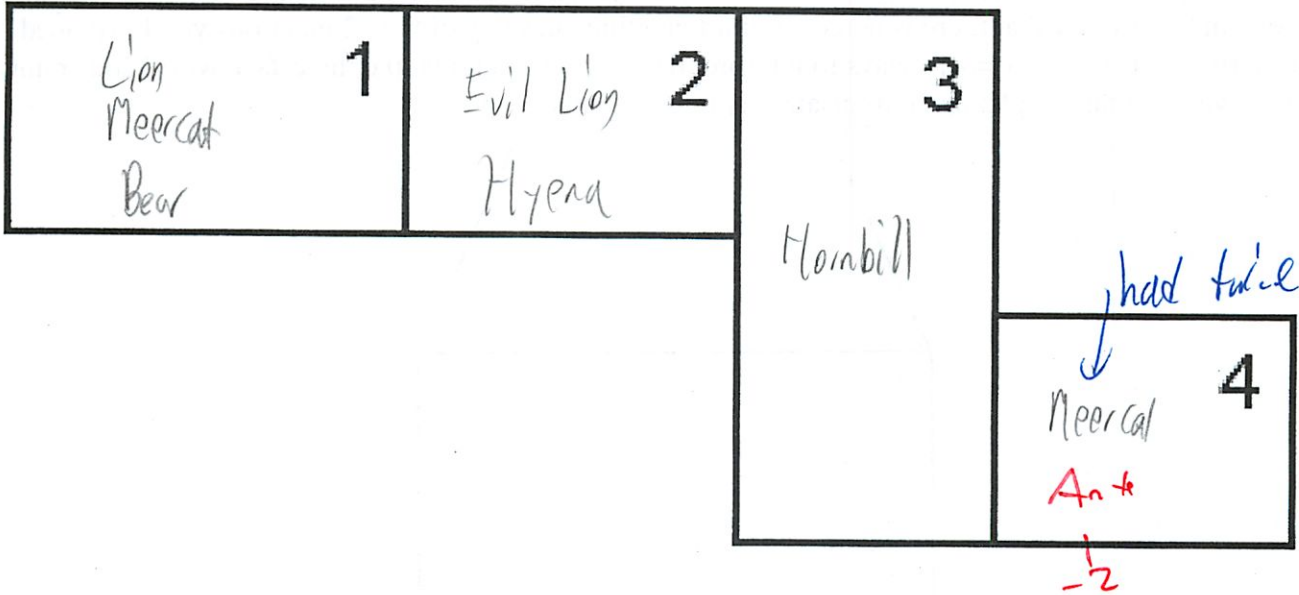
LION	
HORNBILL	1
ANTELOPE	3 4
EVIL LION	4
HYENA	2
MEERKAT	2
BOAR	1 3 4



T did right

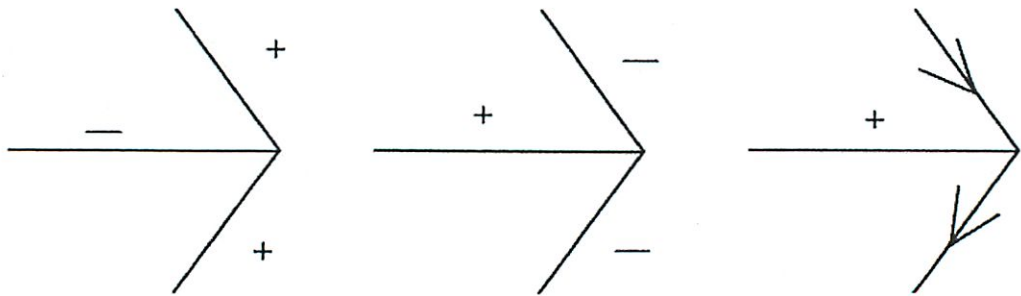
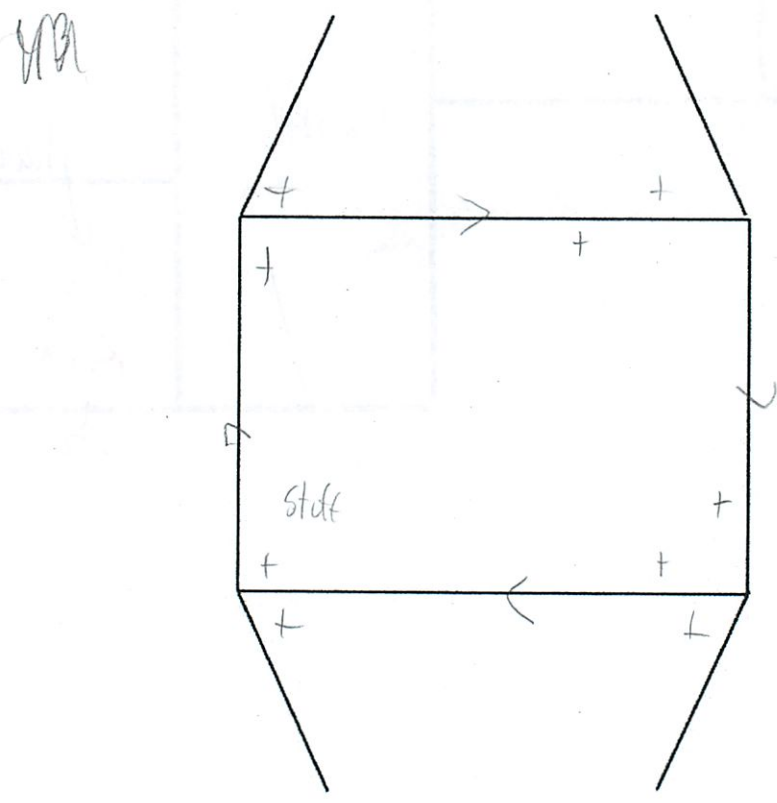
Part E (5 points)

Write the animal names in their final locations in the zoo.



Problem 3: Constraints in Drawings (20 points)

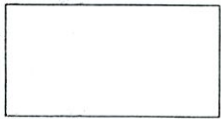
You are working on a problem set with various friends. In one question, you are presented with the drawing below, consisting of arrow junctions and junctions with just one line attached. You are to find junction labellings consistent with each other in the three-face world, which means there are just three ways to label the arrows, as shown. Junctions with just one line attached provide no constraint. You pile up all three labels on each of the four arrow junctions and start to work. Your friends Joshua, Jason, and Justin say that there is no constraint that eliminates any of the 12 junctions you have piled up, so there must be $3^4 = 81$ ways to interpret the drawing such that the three-face world constraints are obeyed. Sarah, Stephanie, and Susana say they are nuts.



If you think the Js are correct, explain why:

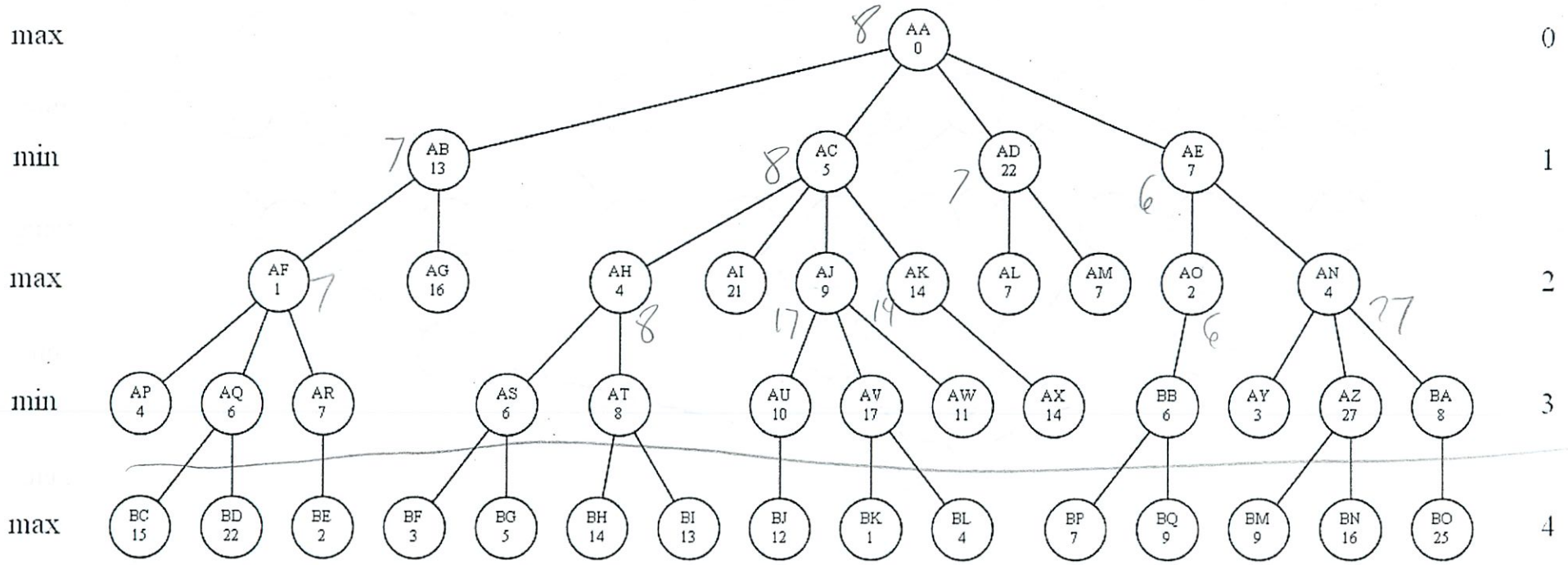
Yes there are 4 possible junctions, each with 3
Combos when you first start $3 \cdot 3 \cdot 3 \cdot 3$

If you think the Ss are correct, determine the number of ways the drawing can be interpreted (show your work).



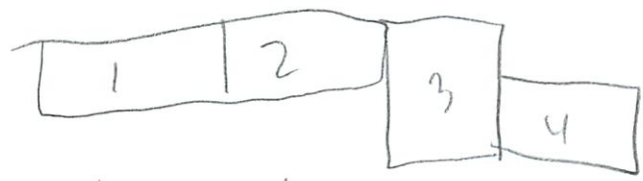
X

Tear off sheet



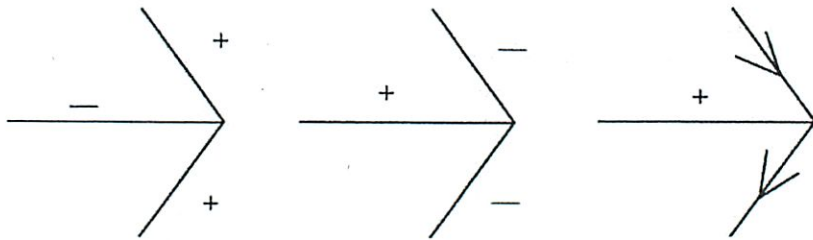
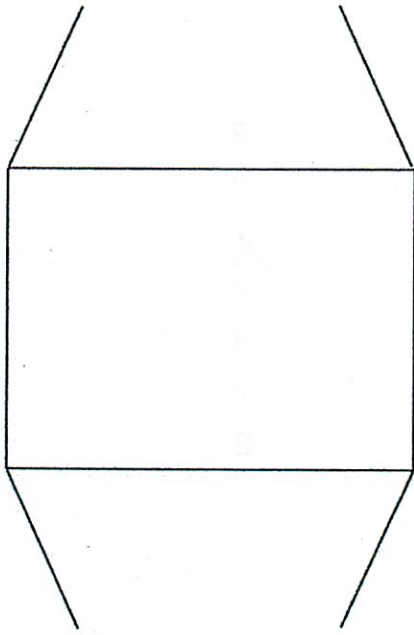
- The **LION** and the **EVIL LION** hate each other, and do not want to be in the same enclosure.
- The **MEERKAT** and **BOAR** are best friends, and have to be in the same enclosure.
- The **HYENA** smells bad. Only the **EVIL LION** will share his enclosure.
- The **EVIL LION** wants to eat the **MEERKAT**, **BOAR**, and **HORNBILL**.
- The **LION** and the **EVIL LION** want to eat the **ANTELOPE** so badly that the **ANTELOPE** cannot be in either the same enclosure with or in an enclosure adjacent to either the **LION** or **EVIL LION**.
- The **LION** annoys the **HORNBILL**, so the **HORNBILL** doesn't want to be in the **LION's** enclosure.
- The **LION** is the king, so he wants to be in enclosure 1.

LION	①	2	3	4
HORNBILL	1	2	3	4
ANTELOPE	1	2	3	④
EVIL LION	1	②	3	4
HYENA	1	2	3	4
MEERKAT	1	2	3	4
BOAR	1	2	3	4



Same as in lion





6.034 Quiz 2

26 October 2011

Name Alan Turing

email _____

Circle your TA and recitation time (**for 1 extra credit point**), so that we can more easily enter your score in our records and return your quiz to you promptly.

TAs		Thu		Fri	
		Time	Instructor	Time	Instructor
Avril Kenney	<u>Adam Mustafa</u>	1-2	Bob Berwick	1-2	<u>Randal Davis</u>
Blondie Chaplin	Erek Speed	2-3	Bob Berwick	2-3	Randall Davis
Gary Planthaber	Caryn Krakauer	3-4	Bob Berwick	3-4	Randall Davis
Peter Brin	Tanya Kortz				

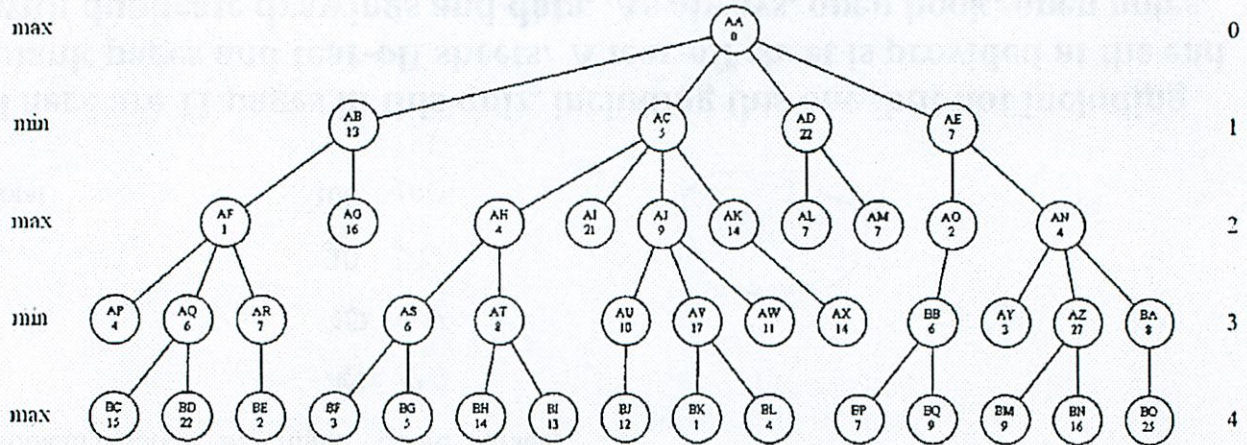
Problem number	Maximum	Score	Grader
1	40	40	
2	40	40	
3	20	20	
Total	100	100	

101

There are 11 pages in this quiz, including this one, but not including blank pages and tear-off sheets. A tear-off sheet is provided at the end with duplicate drawings and data. As always, open book, open notes, open just about everything, including a calculator, but no computers.

Problem 1: Games (40 points)

This part makes reference to the following tree. Each node has a label, and a number which represents the value returned by the static evaluator at that node. The numbers on the right indicate ply.



The tree is reprinted **in larger scale** for your convenience at the end of the test.

Part A: It's Your Move (15 points)

It's your move as the maximizing player. You plan to use the minimax algorithm **without** alpha-beta pruning to find your move, but you only have time to go at most three plies deep, so you do static evaluation at level 3 in the diagram (or at higher levels in parts of the tree that do not go all the way to level 3). What move do you choose?

AC

What is the value associated with that move?

8

How many nodes needed to be statically evaluated during the application of the minimax algorithm?

17

Part B: Your opponent responds (10 points)

Given the move you chose in A1, what is your opponent's move in response? Assume he also has time to go at most three plies beyond the move you have chosen in Part A, the move that takes you to ply 1.

AJ

What is the value of that move?

12

If you chose

for Part A

AB \rightarrow AF, 15

AC \rightarrow AJ, 12

AD \rightarrow AL, 12

AE \rightarrow AD, 7

Part C: Improve your search (15 points)

You realize that you can use alpha-beta to improve the efficiency of your search. You start over on the problem, ignoring all the computation you did in Part A and Part B. Using the minimax algorithm with alpha-beta pruning you decide you can go to the bottom of the game tree, performing static evaluation at level 4 (or at higher levels in parts of the tree that do not go all the way to level 4). What is your best move now?

AB

What is the value of that move?

15

List the nodes you statically evaluated in the order they were evaluated.

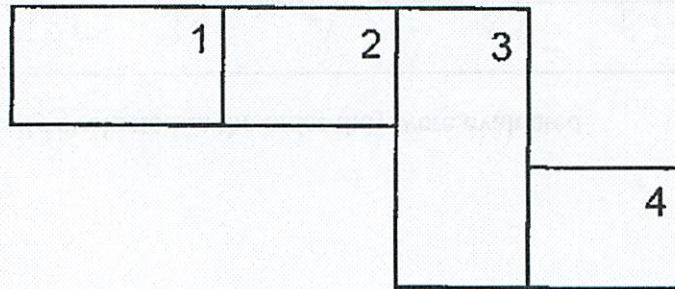
AP BC BD BE AL BF BH AL BP

Problem 2: Constraint Propagation (40 points)

In honor of MIT 150, MIT has decided to open a new zoo in Killian Court. They have obtained 7 animals and built 4 enclosures. Because there are more animals than enclosures, some animals have to be in the same enclosure as others. However, the animals are very picky about who they live with. The MIT administration is having trouble assigning animals to enclosures, just as they often have trouble assigning students to residences. As you have taken 6.034, they have asked you to plan where each animal goes.

The animals chosen are a **LION**, **ANTELOPE**, **HYENA**, **EVIL LION**, **HORNBILL**, **MEERKAT**, and **BOAR**.

They have given you the plans of the zoo layout:



Each numbered area is a zoo enclosure. Multiple animals can go into the same enclosure, and not all enclosures have to be filled.

Each animal has restrictions about where it can be placed.

- The **LION** and the **EVIL LION** hate each other, and do not want to be in the same enclosure.
- The **MEERKAT** and **BOAR** are best friends, and have to be in the same enclosure.
- The **HYENA** smells bad. Only the **EVIL LION** will share his enclosure.
- The **EVIL LION** wants to eat the **MEERKAT**, **BOAR**, and **HORNBILL**.
- The **LION** and the **EVIL LION** want to eat the **ANTELOPE** so badly that the **ANTELOPE** cannot be in either the same enclosure with or in an enclosure adjacent to either the **LION** or **EVIL LION**.
- The **LION** annoys the **HORNBILL**, so the **HORNBILL** doesn't want to be in the **LION**'s enclosure.
- The **LION** is the king, so he wants to be in enclosure 1.

Part A (5 points)

Ben is trying to solve the problem, but he didn't take 6.034 and refuses to believe that constraint propagation works. Because of this, he is determined to do a search with absolutely no domain reduction, constraint checking, or constraint propagation (he's not very smart). Ben only checks to see if an assignment is ok once all animals have been assigned to enclosures. What is the size of the tree that Ben will have to search through to find all solutions?

47

Part B (10 points)

Before starting your search, you realize that you can eliminate some assignments right away. Your friend, Theophilus Teeya, runs through the constraints and determines that some assignments are obviously inconsistent with those constraints, producing the following chart of possible assignments; x means the assignment of the enclosure associated with the column is not possible:

LION	1	x	x	x
HORNBILL	x	2	3	4
ANTELOPE	x	x	3	4
EVIL LION	x	2	3	4
HYENA	x	2	3	4
MEERKAT	1	2	3	4
BOAR	1	2	3	4

Assume Theophilus conclusions are correct; then, use the reduced domains found by Theophilus to find one solution by doing forward checking, **with** propagation through domains reduced by **any** number of values. Break ties in numerical order (1,2,3,4).

LION	
HORNBILL	
ANTELOPE	
EVIL LION	
HYENA	
MEERKAT	
BOAR	

Part C (10 points)

Mercy Lestier, another friend, believes Theophilus could do more to reduce the domains **before** starting with forward checking. She is right. Demonstrate by crossing out **all** additional domain values that can be eliminated by repeated use the domain reduction algorithm.

LION	1	x	x	x
HORNBILL	x	x	3	4
ANTELOPE	x	x	3	4
EVIL LION	x	2	3	4
HYENA	x	2	3	4
MEERKAT	1	x	3	4
BOAR	1	x	3	4

Part D (10 points)

Using the reduced domains found in part C, find one solution by doing **forward checking**, **with** propagation through domains reduced by **any** number of values. Break ties in numerical order (1,2,3,4).

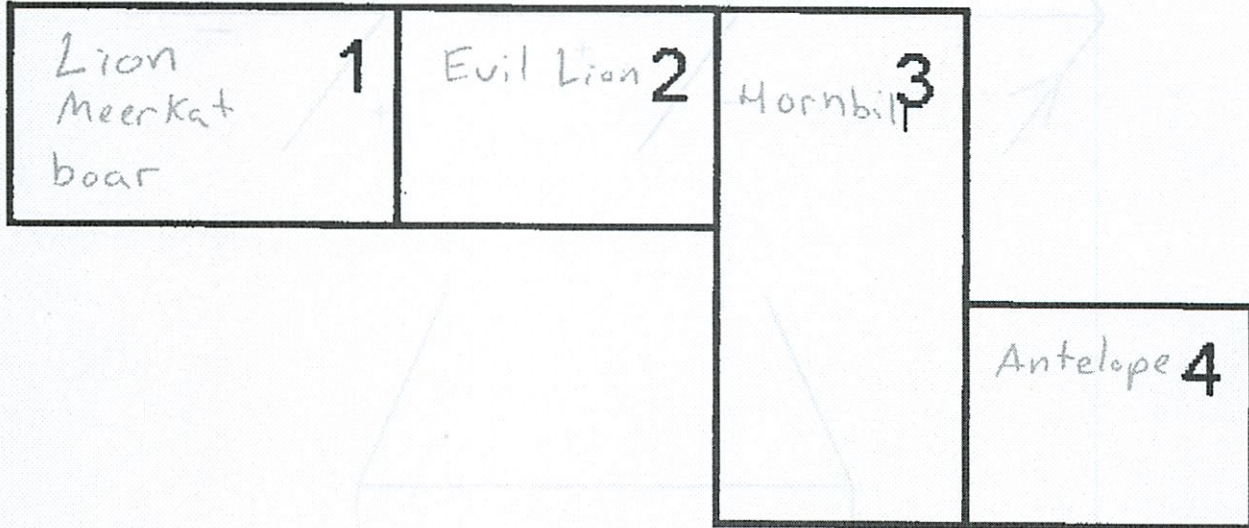
LION	
HORNBILL	1
ANTELOPE	3 4
EVIL LION	4
HYENA	2
MEERKAT	2
MEERKAT	1 3 4
BOAR	1

BOAR	1	2	3	4
MEERKAT	1	2	3	4
HYENA	X	2	3	4
EVIL LION	X	2	X	4
ANTELOPE	X	X	3	4
HORNBILL	X	X	X	4
LION	1	X	X	X

Part E (10 points)

Part E (5 points)

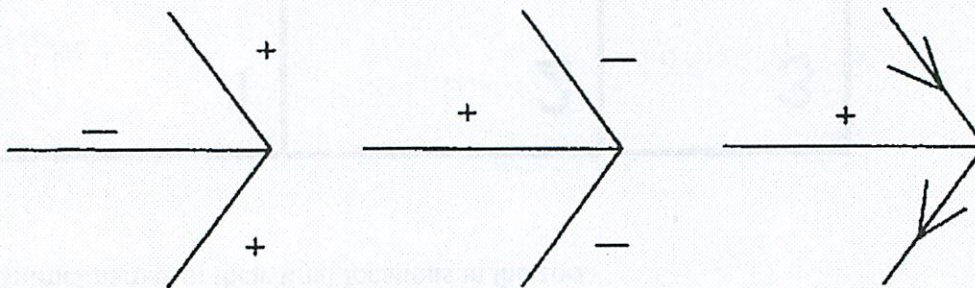
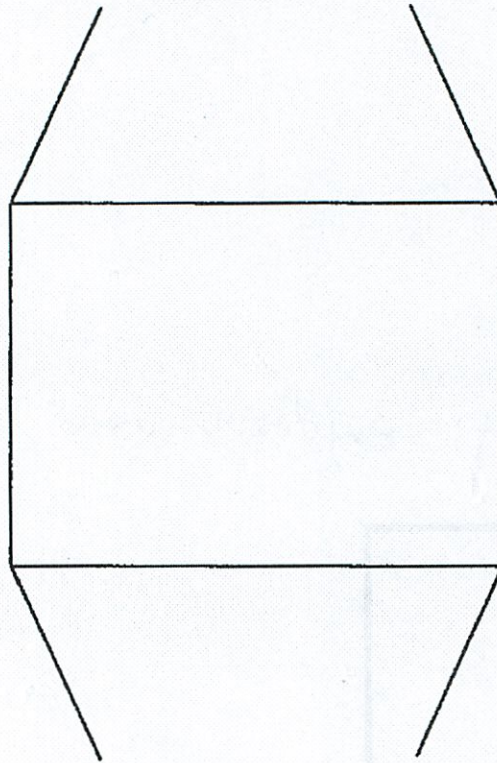
Write the animal names in their final locations in the zoo.



Other answers were also accepted if it made sense.

Problem 3: Constraints in Drawings (20 points)

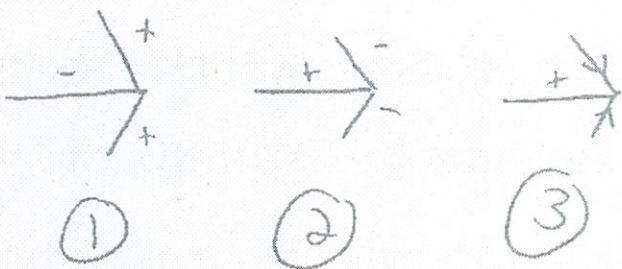
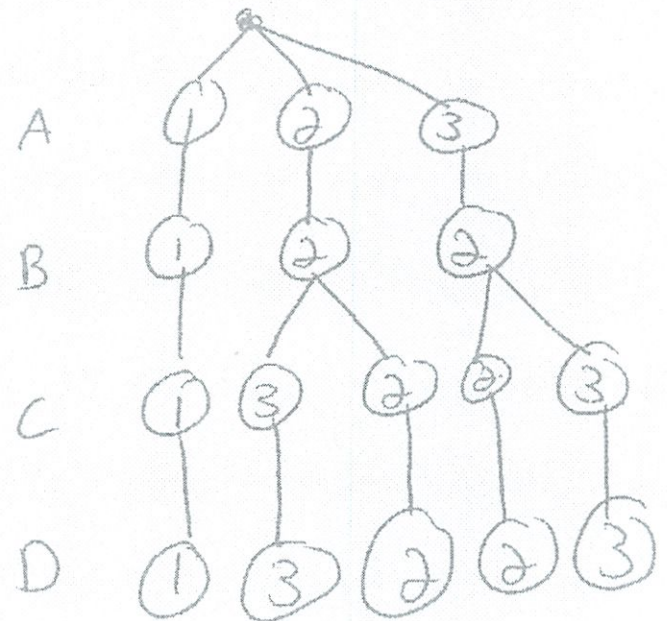
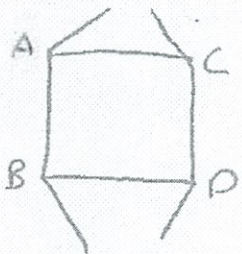
You are working on a problem set with various friends. In one question, you are presented *with the* drawing below, consisting of arrow junctions and junctions with just one line attached. You are to find junction labellings consistent with each other in the three-face world, which means there are just three ways to label the arrows, as shown. Junctions with just one line attached provide no constraint. You pile up all three labels on each of the four arrow junctions and start to work. Your friends Joshua, Jason, and Justin say that there is no constraint that eliminates any of the 12 junctions you have piled up, so there must be $3^4=81$ ways to interpret the drawing such that the three-face world constraints are obeyed. Sarah, Stephanie, and Susana say they are nuts.



If you think the Js are correct, explain why:

If you think the Ss are correct, determine the number of ways the drawing can be interpreted (show your work).

5



Quiz 2 Debit

10/26

α, β only small part of exam

- not scored:

- and which pick is always same

- focus on doing best possible

- so its just what expanded

Not good at all w/ line labling

- never reviewed ...

Constraint prog I think I did Ok

But didn't know it to write stuff

I'm predicting 4

- But cured? - so if no one else had line labling

I don't think could have done better

Except if asked TA on α, β
line labling