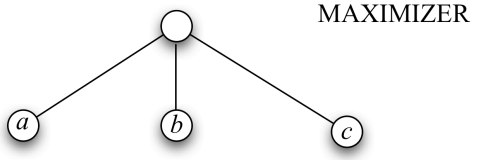
MASSACHUSETTS INSTITUTE OF TECHNOLOGY
Department of Electrical Engineering and Computer Science
6.034 Artificial Intelligence
Fall, 2011

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.

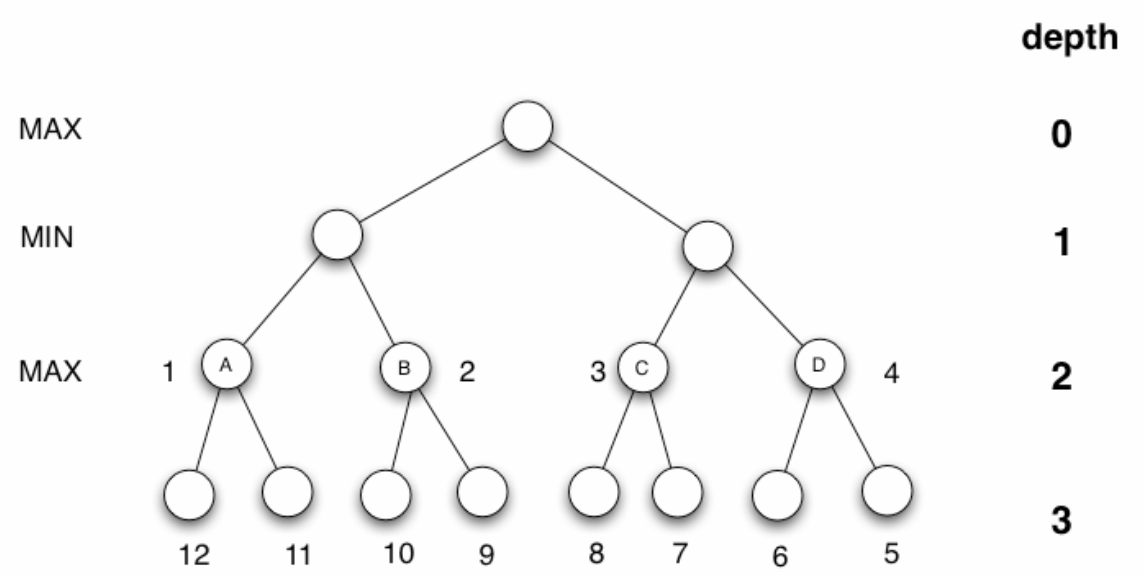
Let's test our ideas on how this works on a couple sample exam problems.

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.

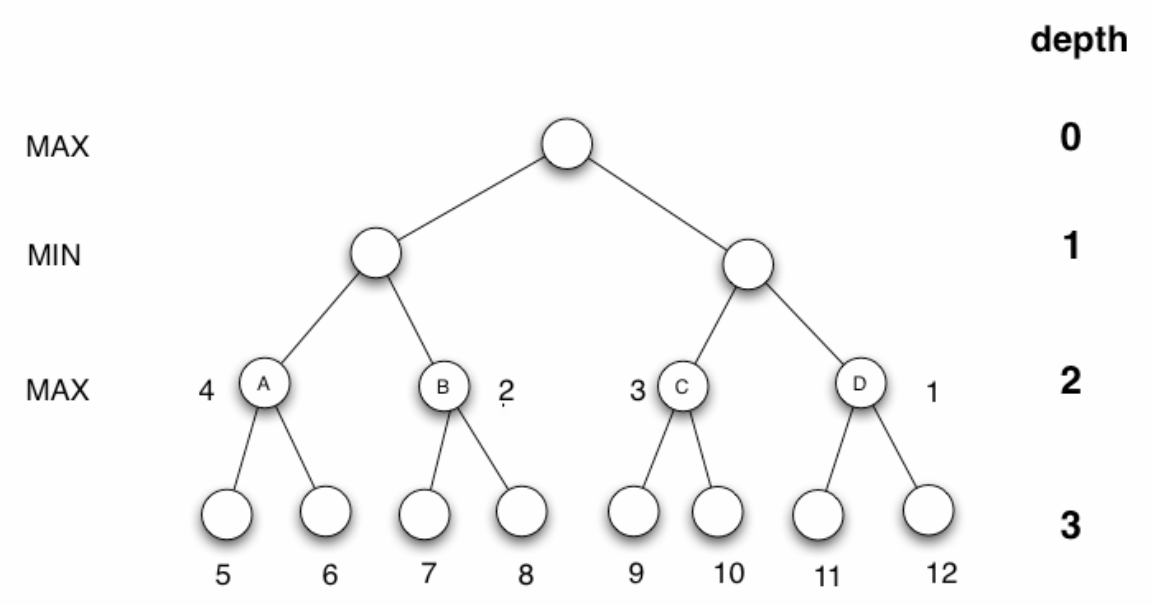
Part C.1: Game Tree I (5 points)



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)

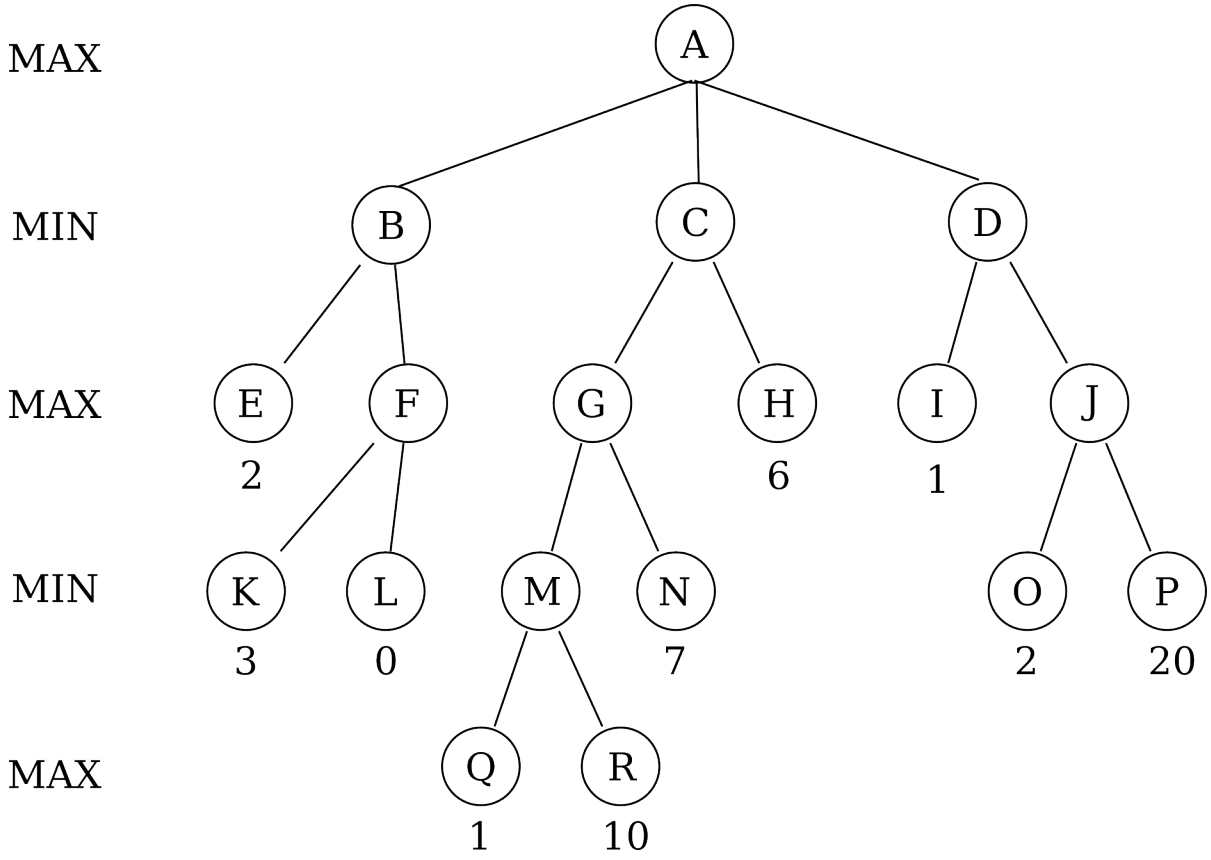


Choose the order in which the nodes/states A, B, C and D in game tree II above are evaluated when running minimax with alpha-beta pruning and progressive deepening after running up to depth 2 and reordering (Circle your answer)

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. N**ote 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.

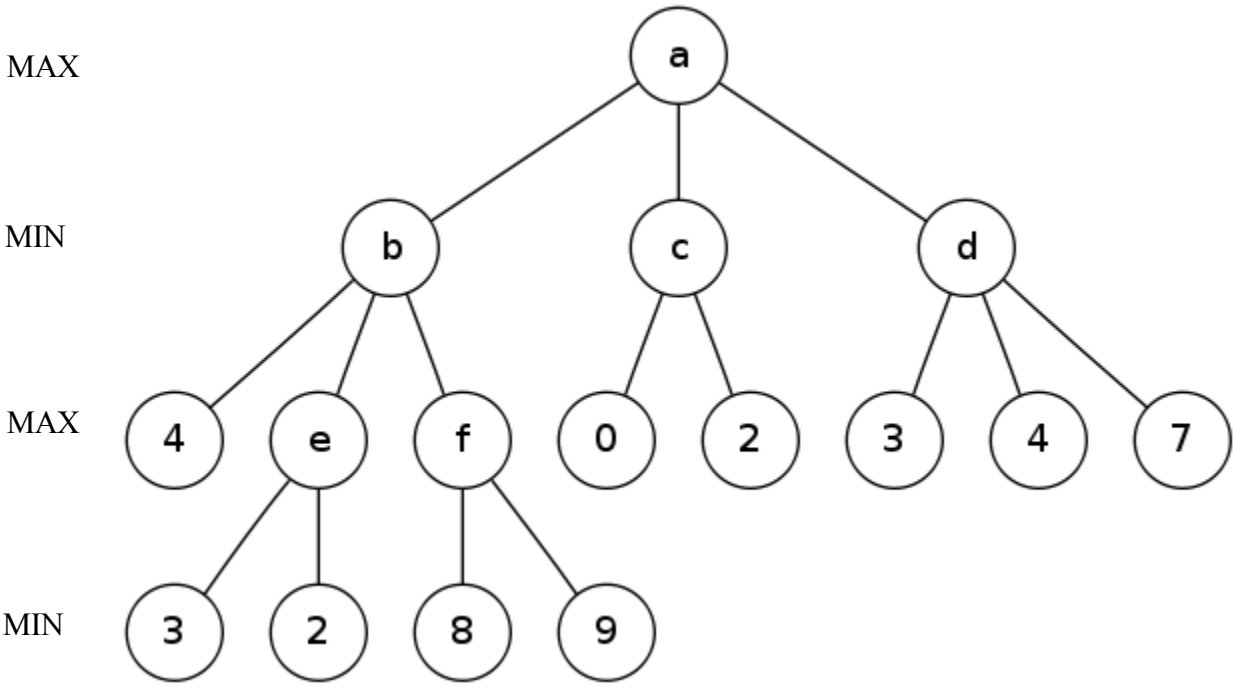
Suppose the static evaluator Tom uses at depth 2 produces exactly the minimax values you found in Part A. Tom uses those numbers to reorder BC&D, EF, GH, and IJ. Note that Tom knows nothing about how the tree branches below depth 2 at this point. Draw the reordered tree down to depth 2, the EFGHIJ level.

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 ore are eXcluded. Here is our notation: assume X(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(Peter, Guitar Player) then we also have, X(Guitar Player, Peter). In this notation, the facts become the following (of course all the symmetrical facts hold as well):

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)

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:

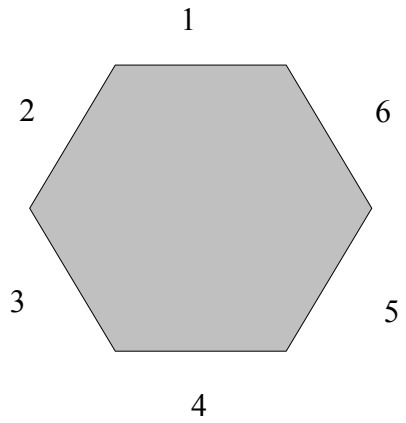
1. If all but one entry in a row are *X,* then the remaining entry is *I.*
2. If you know *I*(*x,y*) and *X*(*y,z*) then you may conclude *X*(*x,z*).

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

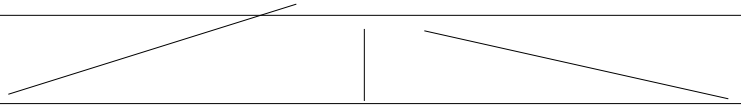
Part B (15 points)

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 commensense 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.

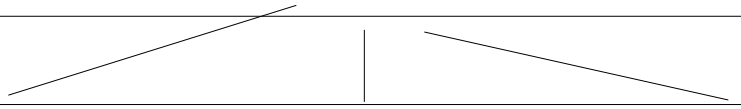
1	Mc
	
2	Y M P
3	
4	
5	
6	

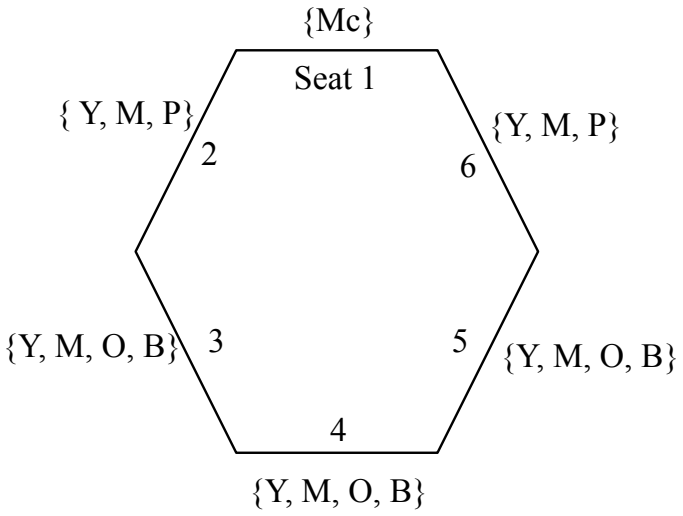
Part C (15 points)

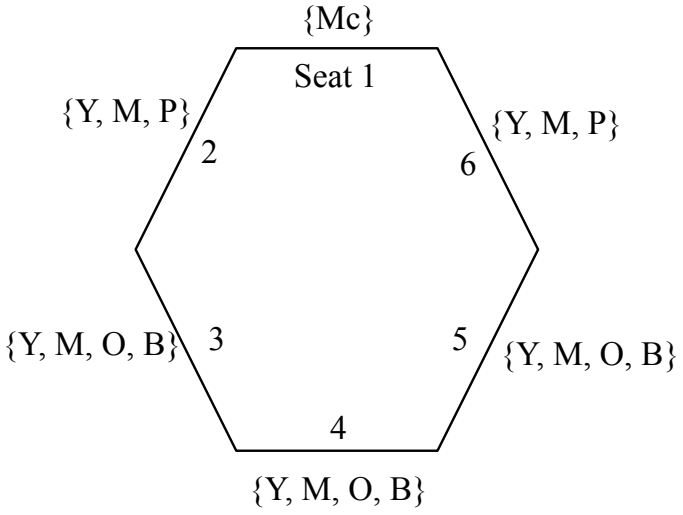
Now you try to use depth-first search *and* ***constraint propagation through domains reduced to size 1.*** **Y**ou break ties by choosing the left-most in this order:

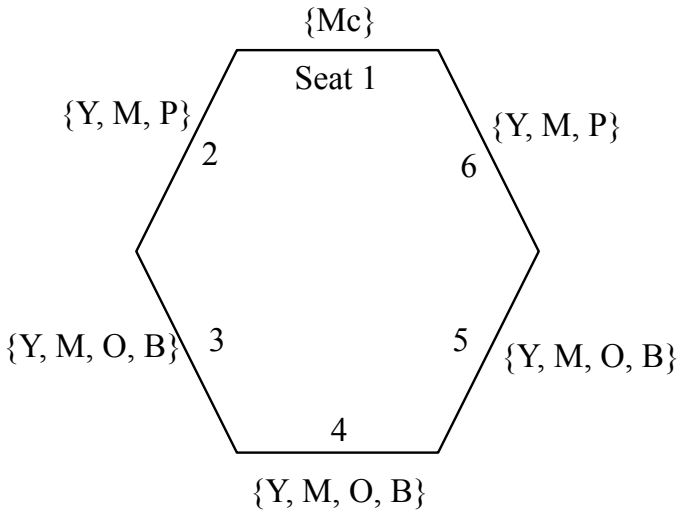
You (Y), Moose (M), McCain(Mc), Palin (P), Obama (O), Biden (B)

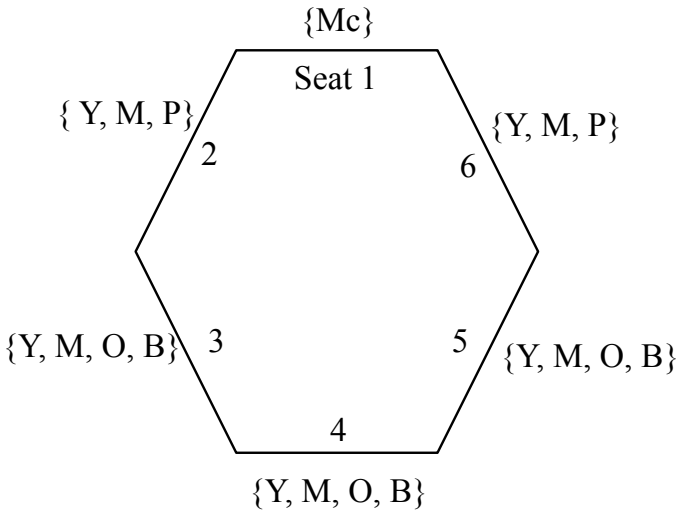
Show the the *full* search tree (up until you find a solution), starting with the same pre-reduced domains. You might want to work this out on a copy at the end of the quiz..

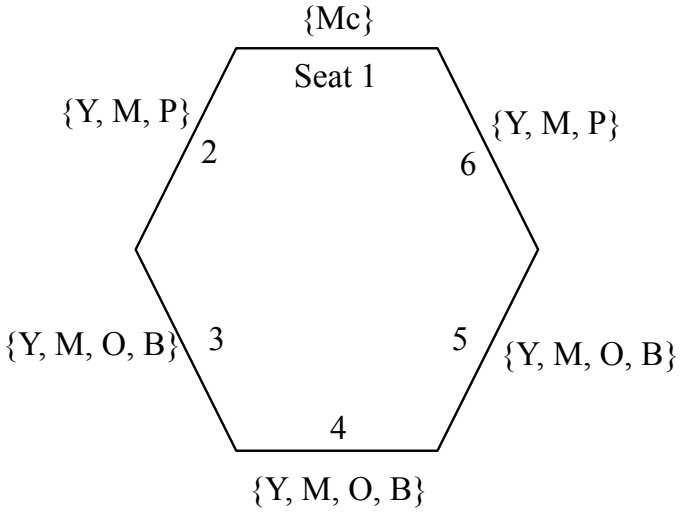
1	Mc
	
2	Y M P
3	
4	
5	
6	

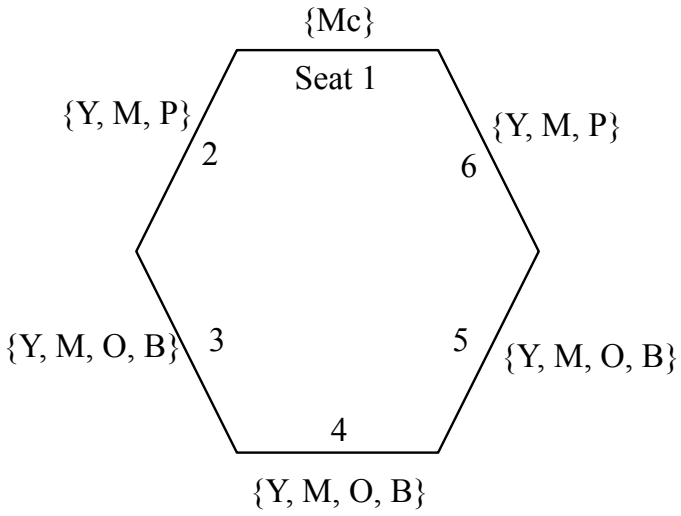

{Mc}
Seat 1
{ Y, M, P}
{Y, M, P}
2
6
{Y, M, O, B}
3
5
{Y, M, O, B}
4
{Y, M, O, B}

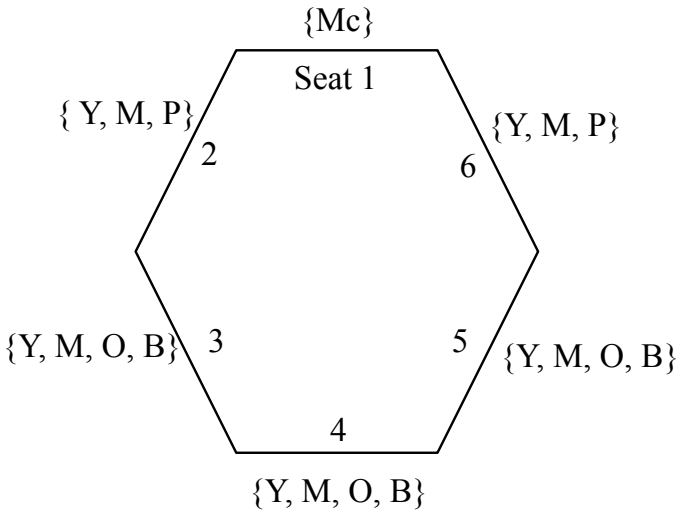

{Mc}
Seat 1
{Y, M, P}
{Y, M, P}
2
6
{Y, M, O, B}
3
5
{Y, M, O, B}
4
{Y, M, O, B}

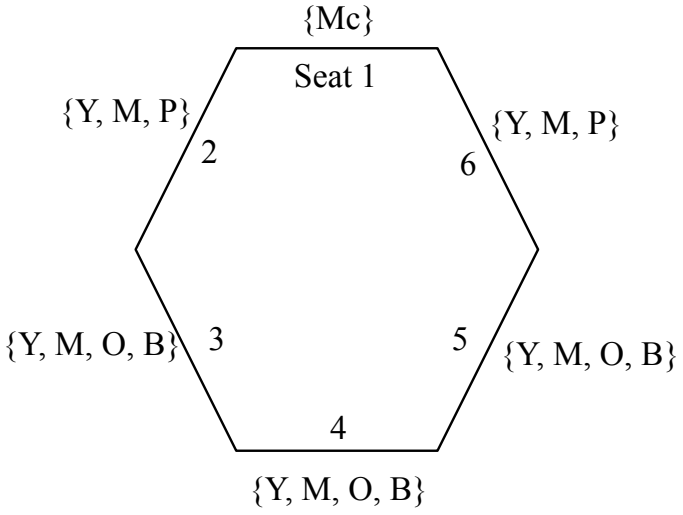

{Mc}
Seat 1
{Y, M, P}
{Y, M, P}
2
6
{Y, M, O, B}
3
5
{Y, M, O, B}
4
{Y, M, O, B}

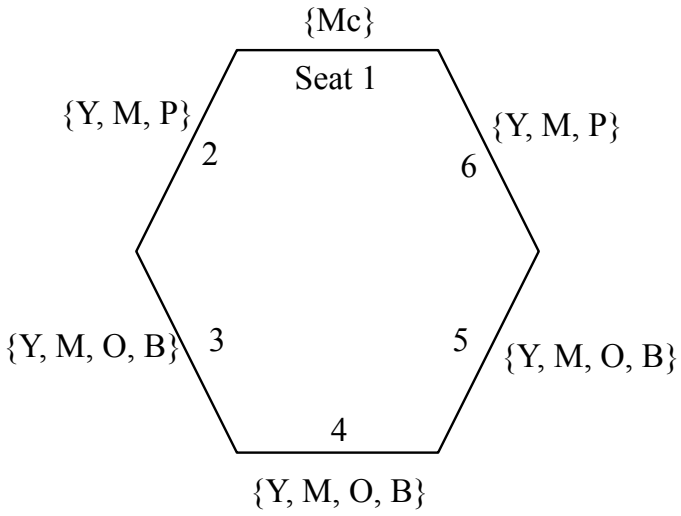

{Mc}
Seat 1
{ Y, M, P}
{Y, M, P}
2
6
{Y, M, O, B}
3
5
{Y, M, O, B}
4
{Y, M, O, B}

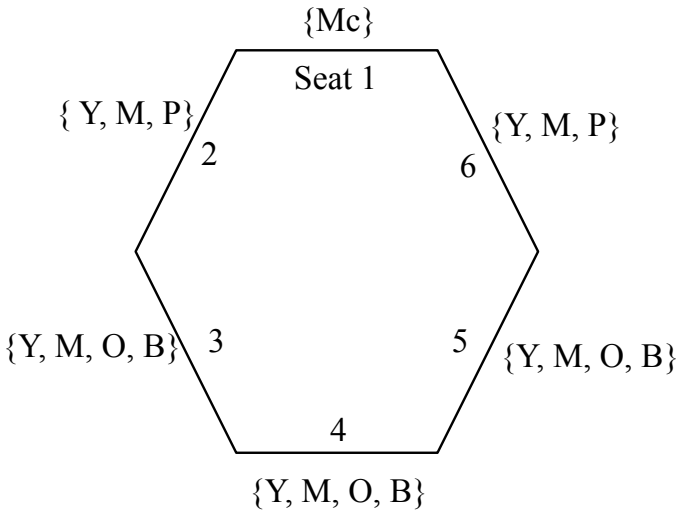

{Mc}
Seat 1
{Y, M, P}
{Y, M, P}
2
6
{Y, M, O, B}
3
5
{Y, M, O, B}
4
{Y, M, O, B}

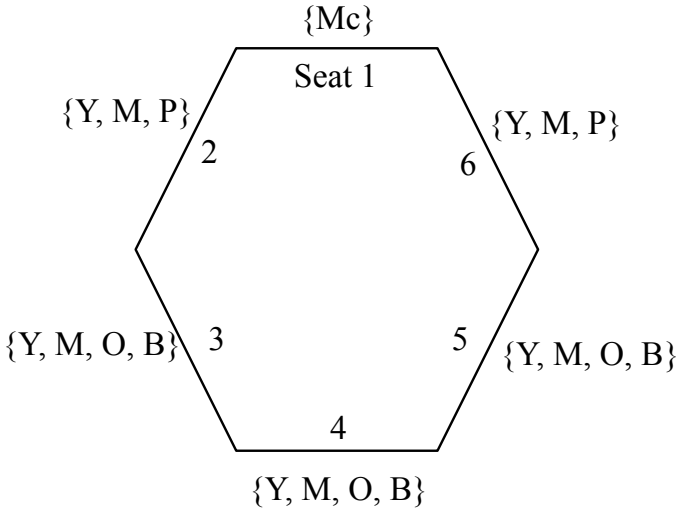

{Mc}
Seat 1
{Y, M, P}
{Y, M, P}
2
6
{Y, M, O, B}
3
5
{Y, M, O, B}
4
{Y, M, O, B}

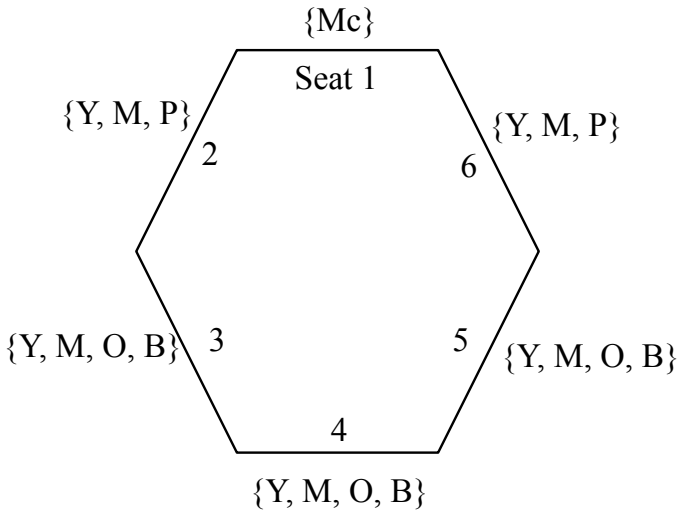

{Mc}
Seat 1
{ Y, M, P}
{Y, M, P}
2
6
{Y, M, O, B}
3
5
{Y, M, O, B}
4
{Y, M, O, B}


{Mc}
Seat 1
{Y, M, P}
{Y, M, P}
2
6
{Y, M, O, B}
3
5
{Y, M, O, B}
4
{Y, M, O, B}


{Mc}
Seat 1
{Y, M, P}
{Y, M, P}
2
6
{Y, M, O, B}
3
5
{Y, M, O, B}
4
{Y, M, O, B}


{Mc}
Seat 1
{ Y, M, P}
{Y, M, P}
2
6
{Y, M, O, B}
3
5
{Y, M, O, B}
4
{Y, M, O, B}


{Mc}
Seat 1
{Y, M, P}
{Y, M, P}
2
6
{Y, M, O, B}
3
5
{Y, M, O, B}
4
{Y, M, O, B}


{Mc}
Seat 1
{Y, M, P}
{Y, M, P}
2
6
{Y, M, O, B}
3
5
{Y, M, O, B}
4
{Y, M, O, B}