W20 CS 317 Exam

This is an open-book and notes untimed exam.

You must work alone on this exam.

Submit your answers as a single Google Doc or PDF with all of the questions in the correct order and clearly numbered.

You can use Google to look up reasonable resources, such as Wikipedia, Youtube videos, slides, and websites, including previous exams from Berkeley (http://ai.berkeley.edu/exams.html). The one thing you cannot use Google for is to see if other schools have assigned these problems and posted solutions (I don't think they have but don't look).

Sign your name to indicate that you have read these instructions, and have upheld the highest ideals of Knox College's Honor Code.

________________Nrepesh Joshi______________

Corrections in this color

1. **Game tree search.** Suppose we are searching a tree to solve a puzzle problem (i.e. a puzzle like 8-puzzle). We have 64 GB of memory for storing nodes, and nodes are 8 KB each. The branching factor is 32. Let's make the simplifying assumption that we are *only* storing the nodes in memory; no other overhead is required.

   a. How deep can we search using BFS? Remember that you fill the queue with everything from depth d+1 while searching depth d.

      Given,
      64 GB of memory for storing nodes.
      8 KB for each node ~ $8 * 10^{-6}$ GB for each node.
      The branching factor is 32.

      Solution,
      We can see here that the branching factor is 32 which means that the average number of children of each node is 32. If we are searching using BFS it means that BFS will expand nodes prioritizing breath for each level of depth. This process is exponential and will exhaust memory in minutes.

      We are limited with memory of 64 GB. In the worst case, we search through the entire frontier to depth d. While searching for depth d, we fill the queue with everything from depth d+1.

      So if the worst case space complexity is $O(b^d)$, but as we are filling the queue with one more depth the space complexity is $O(b^{d+1})$.

      If we were to exhaust the 64 GB by filling it with $8 * 10^{-6}$ GB for each node, then the total number of nodes would be

      No. of nodes = $\frac{64\ GB}{8*10^{-6}\ GB}$ Nodes
      No. of nodes = $8 * 10^6$ Nodes

      The no. of nodes we can fill 64 GB is the worst case so,

      $b^{d+1} = 8 * 10^6$ Nodes
      $32^{d+1} = 8 * 10^6$
      $Log[32^{d+1}] = Log[8 * 10^6]$
      $(d+1)\, Log[32\ ] = Log[8 * 10^6]$
      $(d+1) \ = \frac{Log[8*10^6]}{Log[32]}$
      $(d+1) \ = 4.5863$
      $d \ = 3.5863$

This means that we can only search for depth 3 with branching factor 32 as we fill the queue with depth d+1 while searching for depth d.

b. How deep can we search using DFS? (Hint: Deeper than BFS!)

Given,
64 GB of memory for storing nodes.
8 KB for each node ~ $8 * 10^{-6}$ GB for each node.
The branching factor is 32.

Solution,
We can see here that the branching factor is 32 which means that the average number of children of each node is 32. If we are searching using DFS it means that DFS will expand nodes prioritizing depth.

In the worst case, we store a whole subtree down to depth m. At most there will be b*m nodes on the stack because we only store a single path and unexplored nodes. We will not be limited with memory of 64 GB but will be limited by time.

If we were to exhaust the 64 GB by filling it with $8 * 10^{-6}$ GB for each node, then the total number of nodes would be

No. of nodes = $\frac{64\ GB}{8*10^{-6}\ GB}$ Nodes

No. of nodes = $8 * 10^{6}$ Nodes

The no. of nodes we can fill 64 GB is the worst case so,

$b * d = 8 * 10^{6}$ Nodes

$32 * d = 8 * 10^{6}$

$d = 8 * 10^{6}/32$

$d = 250,000$

This means that we can search for depth very large with just 64 GB of memory with branching factor 32. This might not find the optimal solution first and space is not the limiting factor but time definitely is.

2. **Chess vs Go.** Let's compare the branching factors of Chess and Go, to get a sense of why AlphaGo used Monte Carlo rollouts, convolutional neural nets, and reinforcement learning rather than minimax with alpha-beta pruning.

   a. Chess has a branching factor of about 35, and games are estimated at 80 ply. (The last question should have proved that we are going to be limited by time, not space). Assume that a modern CPU can process 100 million nodes per second, and tournament chess allows 3 minutes per move. Assume we have an effective way to order the moves, which allows alpha-beta to evaluate only $b^{d/2}$ moves instead of b^d moves through pruning. About how many ply can we search in our 3 minutes?

      Given,
      Branching factor is 35.
      100 million nodes per second.
      3 mins per move = 3*60 sec = 180 sec per move (limiting agent)

      Solution,
      In 180 sec, CPU can process 180*100* 10^6 nodes.
      This is the maximum number of nodes we can evaluate using alpha beta pruning.

$$b^{d/2} = 18000 * 10^6 \text{ Nodes}$$
$$35^{d/2} = 18000 * 10^6 \text{ Nodes}$$
$$\tfrac{d}{2} Log[35] = Log[18000 * 10^6]$$
$$d/2 = 6.6417$$
$$d = 13.28$$

We can evaluate 13 plies in 3 minute out of 80 total ply.

   b. Go has a branching factor of 250, and games are estimated at 150 ply. Again, assume we can handle 100 million nodes per second, and again assume we have 3 minutes to make each move (Tournament Go timing rules are different than chess but assume they are the same for this problem). About how many ply can we search in our 3 minutes? Assume that we can also search 100 million nodes per second, and that we are using alpha-beta pruning the same as in the previous part of the question (i.e. we evaluate $b^{d/2}$ actions instead of b^d).

      Given,
      Branching factor is 250.
      100 million nodes per second.
      3 mins per move = 3*60 sec = 180 sec per move (limiting agent)

Solution,

In 180 sec, CPU can process 180*100* 10^6 nodes.

This is the maximum number of nodes we can evaluate using alpha beta pruning.

$$b^{d/2} = 18000 * 10^6 \text{ Nodes}$$

$$250^{d/2} = 18000 * 10^6 \text{ Nodes}$$

$$\frac{d}{2} Log[250] = Log[18000 * 10^6]$$

$$d/2 = 4.276$$

$$d = 8.55$$

We can evaluate 8 plies in 3 minutes out of 150 total ply.

c. What conclusions can you draw from these results? The numbers are not that different.

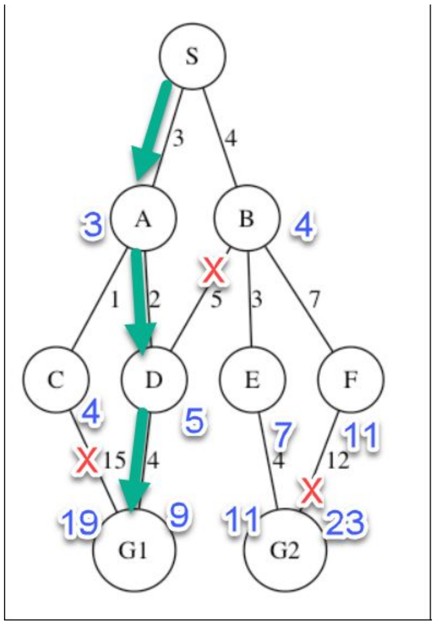
= For both games we were limited by 3 minutes. The branching factor was 35 in Chess and 250 in alpha go. We see that when the branching factor is high then minimax search with alpha beta pruning does not scale well because it needs to evaluate each node. For chess, we can evaluate 13 plies in 3 minute out of 80 total ply. That is only about 16% of the total tree. This would not give us good results when playing with human players. For Alpha go, we can evaluate 8 plies in 3 minutes out of 150 total ply. That is only about 5% of the total tree. This would definitely not give us good results when playing with human players.

The solution to this is reinforcement learning, which only takes optimal strategy using the best action that has been discovered. It uses exploration and exploitation which keeps searching for new paths but exploits the best path found so far.

The reason monte carlo is effective is because it repeatedly samples a problem space randomly to get more optimal paths. It only parses a couple layers of depth at a time. It prioritizes what path to explore nexxt and it simulates outcomes by committing but to expanding all nodes.

3. **UCS and A* with an inadmissible heuristic**. Consider the search tree below, where S is the *start state*, and G1 and G2 are the two possible *goal states*.

    a. What is the optimal path that would be found by UCS?



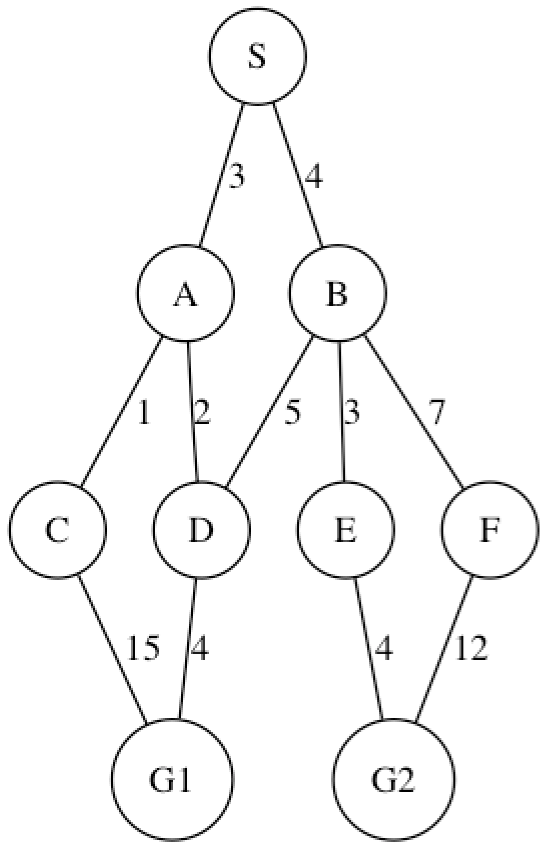
The optimal path that UCS would find is S,A,D,G1.

Explanation:
First S would expand to A and B. A will be expanded next as 3 < 4. The cost to get to C and D would be 4 and 5 respectively. B will be expanded next but B->D would not be optimal because there was another path which got to D at a better cost. The cost to get to E and F would be 7 and 11. The path from C to G1 would cost 19, cost from D to G1 would be 9, cost from E to G2 would be 11 and cost from F to G2 would be 23. We would cross out C-> G1 and F-> G2 because there is already a more optimal path. Comparing D->G1 and E->G2 we see that D->G1 gets us to a goal state with less cost. So, the optimal path that UCS would find is S,A,D,G1.

    b. Give an *inadmissible* heuristic (i.e. one that may over-estimate the true cost to a goal state) that guarantees that A* will find a *suboptimal* goal, but uses as many zeros as possible. So, basically, make up estimates for each node (S, G1, and G2 are already done for you) that would be returned by a heuristic that ensure that A* will find the suboptimal solution. Use as few non-zero values as possible, and make every other estimate zero.

= An admissible heuristic would always underestimate. If a heuristic is 0 then the node assumes that the next node is the goal state. If all the heuristics are 0 then A* will work as Uniform Cost Search. Here we are working with an inadmissible heuristic which over-estimates the true cost to a goal state.

The two terminal nodes are goal states so we are confident that the search will find a goal state. We want the heuristic to find a sub-optimal goal state, so as long as the heuristic does not follow the optimal path of S,A,D,G1 then it is bound to find a sub-optimal goal state.



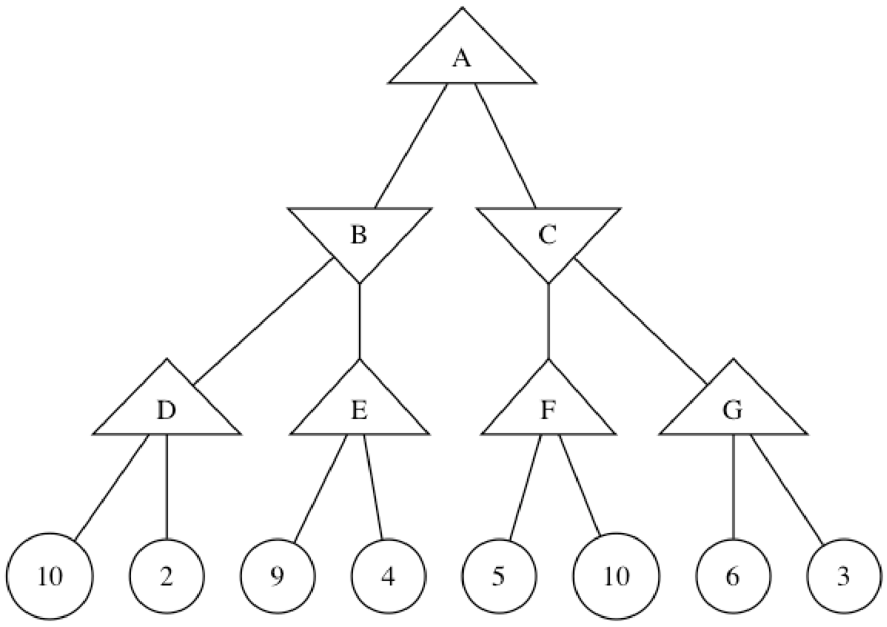
Heuristic	
S	0
A	30
B	0
C	0
D	30
E	0
F	0
G1	0
G2	0

If we said that the heuristic for all nodes were high then it would find a sub-optimal goal state. However, we are using few non-zero heuristic numbers so if the heuristic for the optimal path is high then the search would not proceed in that direction.

When at A the heuristic would guess that the total cost would be 33 where the actual cost would be 9. When at D the heuristic would guess that the total cost would be 35 where the actual cost would be 9. This overestimates act as blockers to get to the actual optimal state.

All other heuristics are 0 which are underestimates but these underestimates act as UCS and will never take us to the optimal goal state. It will lead us to one of the goal states which will be suboptimal.

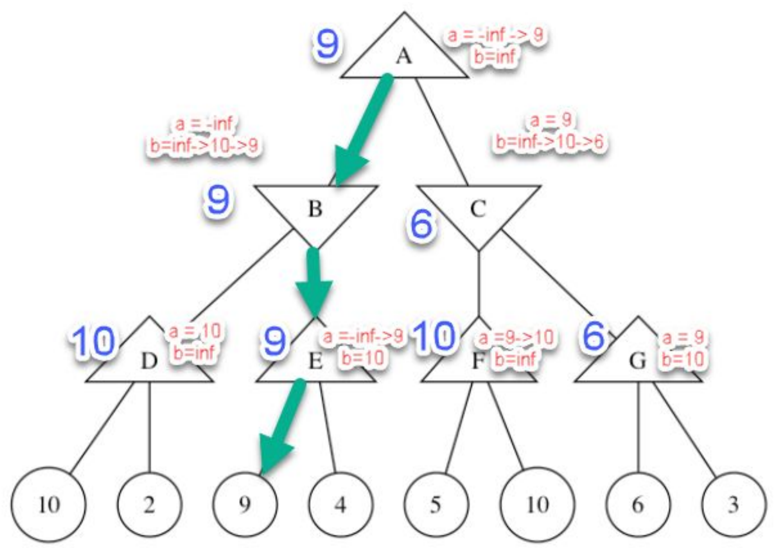
4. **Minimax.** Consider the following multiagent game tree. Assume that triangles are max nodes, upside-down triangles are min nodes, and circled numbers are the result of the evaluation function.



a. Perform Minimax on this tree. Fill out the table below.

A	9
B	9
C	6
D	10
E	9
F	10
G	6

b. Which nodes would be pruned if we were to perform Alpha-Beta pruning? Indicate prunings with an X or a line, or just explain what gets pruned.



No Pruning occurs as the condition Alpha >= Beta condition for pruning never occurs. We go through D and it picks the max node 10. This changes the alpha value to 10 which is the beta value for B. At E, alpha changes from -inf to 9 and beta is 10. Alpha is not greater than beta so no pruning occurs. The alpha value at B remains -inf but the beta value changes from -inf to 10 to 9. The alpha value at A changes from -inf to 9 and beta remains the same.

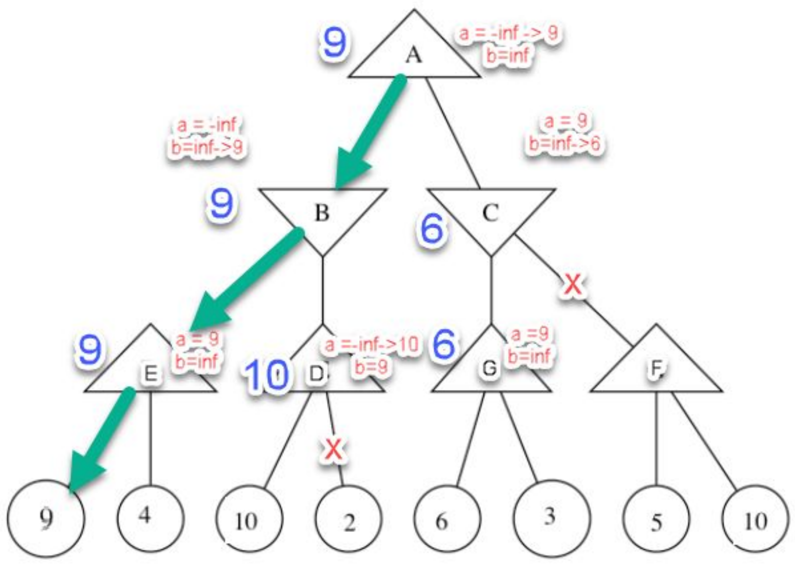
At C, the alpha value remains 9 but the beta value changes from inf to 10 to 6. Alpha is greater than beta here but it is already late to prune. This is because at F the alpha value changes from 9 to 10 and beta remains inf. At G, the alpha value is 9 and beta is 10. This minimax results to the route ABE9 without pruning.

c. Rearrange the order in which we evaluate actions in the tree to maximize the number of nodes pruned by Alpha-Beta pruning. Remember, we can visit the nodes in whatever order we would like (i.e. we can visit C before B, and once we visit C, we can visit F and G in any order, and once we visit F, we can visit 5 and 10 in any order), but we cannot move nodes to a different subtree (i.e. we cannot move F over to be below B).

Nodes		**which node to evaluate first? If it doesn't matter, say so.**
B	C	B should be first
D	E	E should be first

F	G	G should be first
10	2	10 should be first
9	4	Doesn't matter
5	10	Doesn’t matter (pruned)
6	3	Doesn’t matter

Also write the new tree:

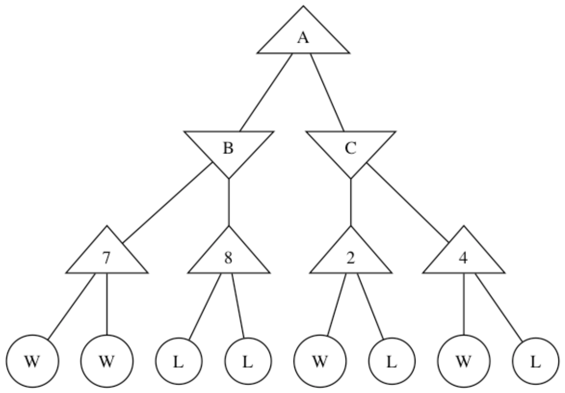


Pruning occurs after rearranging as the condition Alpha >= Beta condition for pruning occurs. We go through E and it picks the max node 9. This changes the alpha value to 9 which is the beta value for B. At D, alpha changes from -inf to 10 and beta is 9. Alpha is greater than beta so pruning occurs and we do not evaluate the second branch. The alpha value at B remains -inf but the beta value changes from -inf to 9. The alpha value at A changes from -inf to 9 and beta remains the same.

At C, the alpha value remains 9 but the beta value changes from inf to 6. Alpha is greater than beta here so we prune evaluating branch F. This is because at G, the alpha value is 9 and beta is inf but at C, beta changes from inf to 6 and pruning occurs. This minimax results to the route ABE9 with alpha-beta pruning.

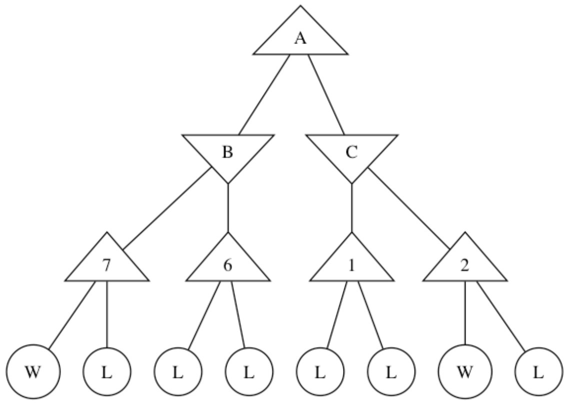
5. **Multiagent search evaluation functions**. Each of the following two minimax game trees has a problem with the evaluation function. Node A is a Max node, while B and C are Min nodes, D E F G are also Max nodes. If we were to stop the search at D E F G, the evaluation function would yield the given numerical values. The W and L nodes indicate terminal nodes; W means that Max wins, and L means that Min wins (and Max loses).
   Each minimax game tree has a problem related to the values returned by the evaluation function. Explain the problem, and fix it by changing only ONE number. What number would you change, and why?

Minimax Game Tree #1:



The evaluation function is used by the algorithm to best estimate the optimal path. If the evaluation function is high then the player would benefit from taking that path. It is not the case in this tree. Even though B is a min node and will take 7 which leads to a win, we see that both lose cases have an evaluation function value greater than the optimal. This is incorrect and I would change 8 to -100 so that the player absolutely does not take that route.

Minimax Game Tree #2:



The evaluation function is used by the algorithm to best estimate the optimal path. If the evaluation function is high then the player would benefit from taking that path. It is not the case in this tree. We see that both lose cases have an evaluation function value greater than the cases with win. This is incorrect and I would change 6 to -100 so that the player absolutely does not take that route. Both lose cases evaluation functions are now less than the ones which have win routes in them.

6. **A* for a really large problem**. Suppose we are solving a puzzle with a state space that is too large to fit into memory, so we cannot search all the way to a goal state *G* (assume there is more than one goal state). We will use A* with an admissible heuristic to search until we are about to run out of memory in our priority queue; at that point we will choose the sequence of actions *A* that brings us to the unvisited node *N* that our heuristic tells us is closest to the goal state (according to the actual distance to *N*, plus our heuristic's estimate to *G*). Then we will re-start the search with node *N* as the root of a new search tree, and run A* again from *N*.

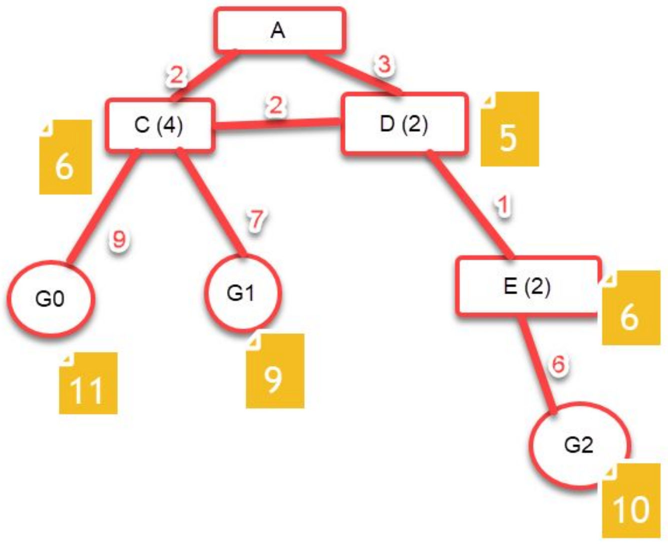
   a. Is this approach guaranteed to find a goal state *G*? If so, explain why; if not, give a *specific example* of a search tree where the algorithm will not find it.

= Yes this approach is guaranteed to find a goal state because the heuristic in A* leads to one of the goal states no matter where we restart the A* search. If all terminal nodes are goal states then the approach always moves to a goal state.

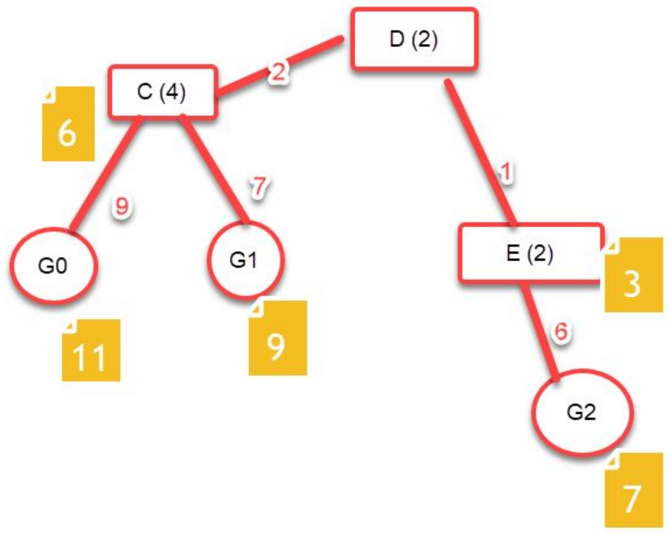
Furthermore, as the memory runs out, we are brought to the unvisited node N that our heuristic tells us is closest to the goal state. So restarting A* might get us to the optimal place but is not guaranteed.

   b. Is this approach still optimal (i.e. guaranteed to find the goal state closest to the root)? If it is optimal, explain *why you cannot* find a suboptimal solution; if it is not optimal, show a *specific example* of a search tree where it would not be optimal.

= The approach does not guarantee us an optimal solution. The counterexample is below.



Here we see that the optimal path considering heuristic would be ACG1 with a cost of 9



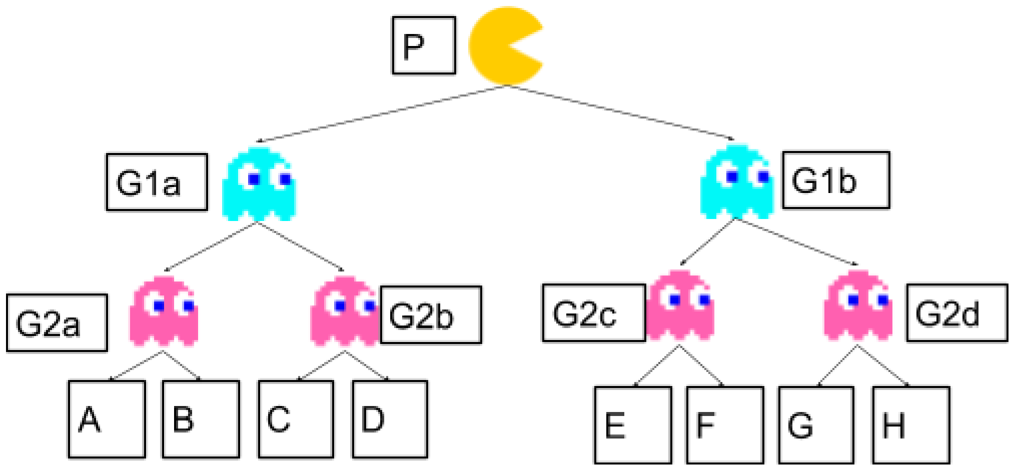
If A is dropped then the C branch would still be the same however, DEG2 would now be the optimal.

7. **A* search.** Suppose we want to use A* to solve a Rubik's cube (wikipedia) puzzle. An action is a twist or turn of the cube.

   a. Suppose you just want to get one particular face of the cube to be blue, but you don't care about the other 5 faces of the cube. Is the number of squares that are not blue on the face you are trying to make blue an admissible heuristic? Why or why not?

= An action is a twist or turn of the cube. A heuristic is admissible if it always underestimates. We do not care about the other 5 faces of the cube. The number of squares that are not blue on the face you are trying to make blue is an admissible heuristic. This is because it is the worst case with no blue squares at all. This only happens if it is a major underestimate. Any move that we make will either bring a blue square or make the state better.

8. **Minimax vs Expectimax.** For the following game tree, assume that nodes A-H are terminal states (i.e. either end of the game, or the result of calling our evaluation function). Assume that Pacman is a Max node, and will choose the actions left or right (i.e. either the left or the right branch).

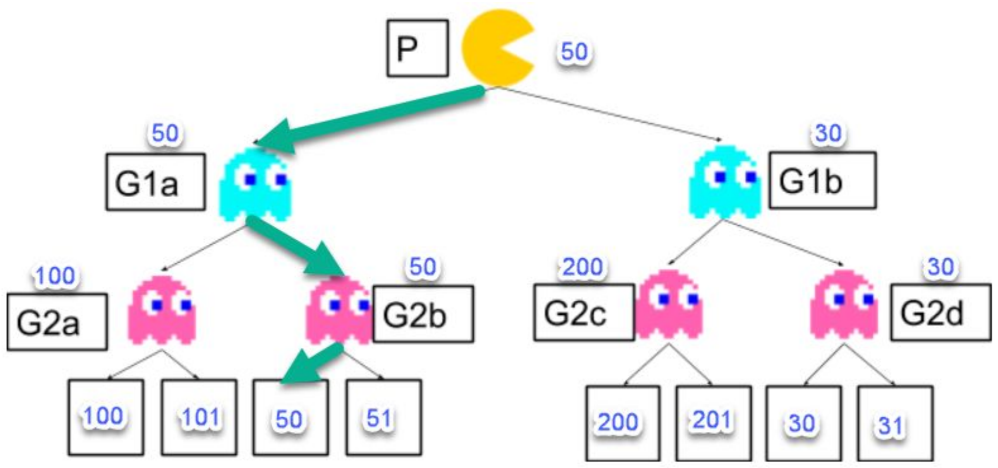


a. Choose values for A-H such that Pacman would choose a different action using Expectimax than it would with Minimax. For Expectimax, assume the ghosts move randomly (i.e. they are 50% likely to choose the left branch and 50% likely to choose the right branch), while for Minimax, the ghosts are Min nodes. Also fill out the values for P, G1a, G1b, and G2a to G2d for both minimax and expectimax.

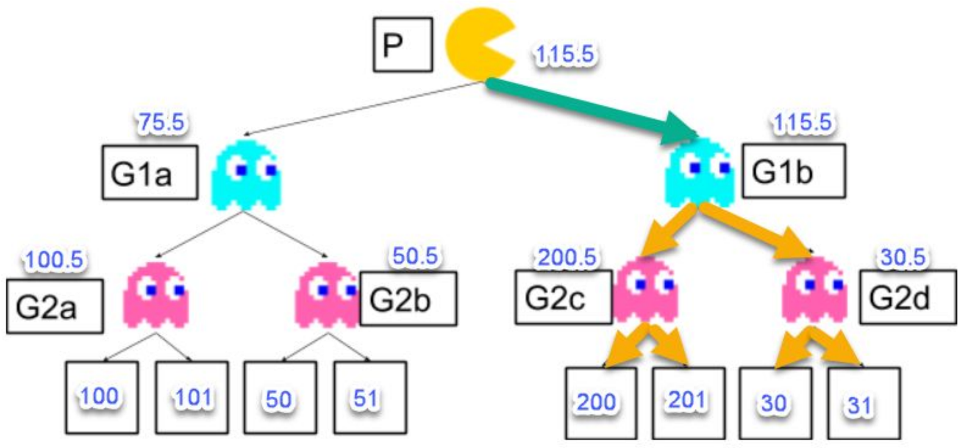
	A	B	C	D	E	F	G	H
Values	100	101	50	51	200	201	30	31

	P	G1a	G1b	G2a	G2b	G2c	G2d
Minimax	50	50	30	100	50	200	30
Expectimax	115.5	75.5	115.5	100.5	50.5	200.5	30.5

For Minimax:



For Minimax, Pacman is a max player and the ghosts are min players (deterministic). G2a and G2b pick 100 and 50 respectively as those are the minimum number. G2c and G2d pick 200 and 30 respectively as those are the minimum number. G1a and G1b pick 50 and 30 respectively as those are the minimum number. Now, Pacman is a max player so it picks 50 instead of 30.



For Expectimax, Pacman is a max player and the ghosts are non-deterministic. The ghosts move randomly (50% to choose left or right). G2a and G2b have an average value of 100.5 and 50.5 respectively. G2c and G2d pick 200.5 and 30.5 respectively. G1a and G1b have an average of value of 200.5 and 30.5 respectively. These numbers are not minimum and they mean that the ghosts can take either path but on average their score is going to average to the average number. However, Pacman is a max player so it picks 115.5 instead of 75.5. So, minimax and expectimax take different actions.

9. **Constraint Satisfaction Problems (CSPs)**. We are scheduling MWF classes in the Computer Science department at Knox College. We have to schedule 6 classes (cs141, cs142, cs214, cs292, cs314, cs399) and each course will be taught for one period during periods 1 to 6. So there are 6 variables, and 6 domain values per variable. (For this question, ignore lab periods, and pretend that cs292 is taught on a MWF schedule). We have the following constraints:
    i. cs141 and cs317 cannot be scheduled at the same time
    ii. cs214 and cs142 cannot be scheduled at the same time
    iii. cs292 and cs399 cannot be scheduled at the same time
    iv. cs317 and cs141 cannot be scheduled 4th period
    v. cs214 and cs142 cannot be scheduled 4th period
    vi. cs141 and cs317 cannot be scheduled 1st period
    vii. cs292 and cs399 cannot be scheduled 6th period
    viii. cs317 and cs399 and cs214 cannot be taught at the same time
    ix. cs214 and cs142 should be taught back-to-back
    x. cs317 must be taught later in the day than cs399 (for some reason)

a. Which are the binary constraints?

I. cs141 and cs317 cannot be scheduled at the same time
= Two variables involved in the same equation (same time).

II. cs214 and cs142 cannot be scheduled at the same time
= Two variables involved in the same equation (same time).

III. cs292 and cs399 cannot be scheduled at the same time
= Two variables involved in the same equation (same time).

VIII. cs317 and cs399 and cs214 cannot be taught at the same time
= Three variables involved in the same equation (same time). Could compare 2 variables but it would still be in the same equation.

XI. cs214 and cs142 should be taught back-to-back
= Two variables involved in the same equation (back-to-back).

X. cs317 must be taught later in the day than cs399 (for some reason)
= Two variables involved in the same equation (later in the day).

b. Which are the unary constraints?

IV. cs317 and cs141 cannot be scheduled 4th period
= Can be divided into two unary equations. Variable is equal to constant.
    cs317 cannot be scheduled 4th period

cs141 cannot be scheduled 4th period

V. cs214 and cs142 cannot be scheduled 4th period
= Can be divided into two unary equations. Variable is equal to constant.
cs214 cannot be scheduled 4th period
cs142 cannot be scheduled 4th period

VI. cs141 and cs317 cannot be scheduled 1st period
= Can be divided into two unary equations. Variable is equal to constant.
cs141 cannot be scheduled 1st period
cs317 cannot be scheduled 1st period

VII. cs292 and cs399 cannot be scheduled 6th period
= Can be divided into two unary equations. Variable is equal to constant.
cs292 cannot be scheduled 6th period
cs399 cannot be scheduled 6th period

c. **Suppose we have the following table of possible remaining domain values for variables cs399 and cs317. We perform AC-3 to compute Arc Consistency. Which values (if any) from the domain of cs399 would need to be removed?**

= To be arc consistent, every binary constraint c(x,y), every value in domain should satisfy c(x,y) using some variable from the domain of y.
The constraints having cs 399 and cs317 are:

1. Cs317 and cs399 cannot be taught at the same time
2. Cs 317 must be taught later than 399

Arcs:
317 != 399 (cannot be in the same time)
399 != 317 (cannot be in the same time)
399 < 317 (must be taught later than)
317 > 399 (must be taught later than)

Agenda:
399 != 317 : except for 1 all 2,3,4,5 are crossed as they are not unique
317 != 399 : After cancellation 1 is unique
399 < 317 : Satisfied
317 > 399 : Satisfied

399 = (1,2X, 3X, 5X)
317 = (2,3,4,5)

So we cancel 2nd, 3rd and 5th from cs399. This means we can only teach cs 399 1st period which will not be at the same time as cs317 and Cs 317 will always be taught later than 399

	cs399	**cs317**
1	X	
2	X	X
3	X	X
4		X
5	X	X
6		

10. **K-fold cross validation for Machine Learning**. We want to build a classifier that can classify a book into a genre. Each book has exactly one genre label. We train a machine learning classifier using 10,000 pre-classified books, extracting a variety of features from the books. We measure the precision of the classifier by the number of books that it classifies correctly (i.e. the classifier picks a label and that label is correct) divided by the total number of books we tried to classify. We perform K-fold cross validation by repeatedly splitting our data set of 10,000 books into 3/4 that we use to train the classifier, and 1/4 we use to test the classifier. (For this problem, it does not actually matter what machine learning algorithm we used, or what particular features we used; just look at the data, and think about what it means. You don't even really need to know anything about ML to answer this).

    Suppose we perform 10-fold cross validation, and get these results:

Fold	**precision**
1	90
2	90
3	67

4	88
5	91
6	89
7	86
8	91
9	92
10	86
Total	87

a. What can we conclude about these results in general?

= In 10 fold cross validation, we train the model with 9 datasets and we test it in the respective data set. We do that 10 times for each dataset. The precision value is the number correct divided by the total number of books we classified. A precision value of 90 is very good. It means that it predicted 90% of the books correctly. Set 3 has about 67% precision but on average all the sets have more than 80% precision. We can conclude that this model would do a good job correctly classifying on a new data set.

b. Suppose that the 10,000 training books are classified into 2 genres with 9,000 books in one genre and 1,000 in the other genre (there are only the 2 genres). What can we conclude about these results under those circumstances?

= This model may overestimate heavily and will be biased on the training data set. If 9000 books were trained in one genre then a test book will be biased towards that genre. A good training set would be divided equally and not 90/10 split. This would result in reduced precision as the model would predict incorrect values biased towards one genre.

c. Suppose that the 10,000 books are classified into 10 genres with 1,000 books in each genre. What can we conclude about these results?

= For 10 genres, the 1000 data points might be very less data to train a good model. The model would not overfit but the accuracy of the prediction might not be correct. Regarding precision, the average would be the same but each individual precision value would reduce.

11. **Debugging**.
    a. How has your process for debugging changed during this course?

= My debugging skills have surely advanced during this course. The skill I learned is to simulate the debug in my mind. Sometimes print statements were very hard to add between statements because of the other methods involved. For that I tried to simulate the workflow of my code and understand what it is really trying to execute. This skill was very important for me to understand what was really going on with the code.

    b. Explain your process for debugging for a future CS 317 student.

= The first advice would be to add print statements and try to figure out where the code works and where it doesn't. Making sure you understand the data structures is key for debugging. Since we use tuple unpacking and list comprehension it is better to see if the list is being parsed correctly. The autograder is your friend and it will give you great insights on what's going wrong with the code. Sometimes you get stuck on the score threshold and adding new conditions could really make the code have a better score.

12. **What was your most important take-home message from the AISOC readings and viewings this term? In other words, what is most likely to stick with you?**

= I really agree with Nick Bostrom when he said that we need to make artificial intelligence understand the values of humankind before making it super intelligent. Bostrom wants to prevent super intelligence AI from creating a King Midas situation by using the intelligence to learn what we value first. I do agree that we might reach a point where there will be no off switch such as "where is the off switch to the internet?".

Bostrom wants the AI motivation system to be constructed in such a way that it is motivated to peruse human value. I again disagree with this statement because humankind is restricted by mortality. I believe that unless AI is also motivated by mortality like humans they will not be able to fully understand our values. I hope to experience AI making me smile in the year 2040-2050 in a humane way.