Question 1 - 20 points

Consider a search tree for which we know the following:
- Each node has 0, 1, or 2 children, never more than 2.
- The depth of the tree is 100.
- There is one and only one goal node in the entire tree, and it is located at level 20.
- The entire tree contains exactly 500 million nodes.
- The tree contains exactly 1000 nodes with depth less than or equal to 20.
- We define the depth of the root to be 1 (not 0).

1a (10 points). For each of breadth-first search (BFS), depth-first search (DFS), and iterative deepening search (IDS), what is the maximum size (MEASURED IN NUMBER OF NODES) that can be reached by the list of nodes to visit, given the information above? In other words, what is the maximum number of nodes that the list can possibly contain at the same time? For question 1a, reasonable upper bounds that are no more than 5 times the correct answer, will get full credit.

BFS: 1000 nodes or 2^20 nodes.

DFS: 200 nodes (b*m)

IDS: 40 nodes (b*d)

1b (4 points). What is the maximum number of nodes that BFS may visit?
1000 nodes

1c (3 points). Is it possible for DFS to visit more than 1000 times more nodes than the maximum possible number for BFS? Justify your answer.
Yes. DFS can visit almost all of the 500 million nodes

1d (3 points). Is it possible for IDS to visit more than 1000 times more nodes than the maximum possible number for BFS? Justify your answer.
Here we assume that if IDS visits a node multiple times, we count each visit separately. Then, the answer is no. If BFS visits a node A, IDS will not visit that node more than 20 times, since the goal is at level 20. So, IDS can at most visit 20 times more nodes than BFS.

Question 2 – 20 points

Consider a search problem for which we have both an A* implementation and a uniform cost search (UCS) implementation. Remember that, in A*, the order in which nodes n are visited is increasing in f(n), where f(n) is an estimate of the cost of a solution going through node n. Remember that f(n) = g(n) + h(n), where g(n) is the actual cost of the path from the root to n (i.e., g(n) is the sum of the costs of all edges traversed to get from the root to n), and h(n) is a (not necessarily accurate) heuristic estimate of the cost of the path from n to the nearest goal node.

In this question, the heuristic function h(n) that we use for A* is:

h(n) = 10.

In other words, according to this heuristic, the estimated cost of the path from EVERY node n to the goal is 10. We also know that every search tree in which we apply these implementations will have one and only one goal node.

2a (10 points). Is the heuristic function h(n) = 10 admissible? Justify your answer.

No. For the goal node g, the true cost of getting from g to the goal is 0, but h(n) = 10 > 0.

2b (5 points). Will A*, using this heuristic function h(n) = 10, always find the smallest-cost solution? Justify your answer. THROUGHT THIS EXAM YOU CAN ASSUME THAT ALL EDGES OF THE SEARCH TREE HAVE NON-ZERO POSITIVE COSTS.

Yes, because A* will visit the exact same nodes and in the exact same order as UCS. UCS visits nodes in an order determined by f(n) = true cost from root to n. A* with h(n) = 10 will visit nodes in an order determined by g(n) = f(n) + 10, so the order will not change compared to the order imposed by f(n).

2c (5 points). Is it possible that A*, using this heuristic function h(n) = 10, will ever visit more nodes than uniform cost search (UCS)? Is it possible that A*, using this heuristic function h(n), will ever visit fewer nodes than UCS? Justify your answer. A node n is considered to be visited at the point where the program checks if n is the goal node (if n is the goal node, the search finishes).

No, A* will always visit the exact same number of nodes as UCS for the same reason as in question 2b.

Question 3 – 20 points

We want to find optimal paths in a maze, which is a grid of size 100x100. At each step we can move to an immediately adjacent square that is up, down, left, or right from the current square. However, some squares are blocked, and it is not possible to be at or move to those squares. We want to find an optimal path (i.e., a path with the smallest possible number of moves) that leads to square (92,28), where there is a pot of gold. In addition to the above, we also know the following:

- the starting position,
- the location of the goal square (which is (92,28)).
- the location of all blocked squares.
- that there exists at least one path to the goal.

However, because of limited memory, the search algorithm cannot keep track of all positions already visited during search.

3a. (10 points) Given the above information, define a maximally admissible heuristic to be used in conjunction with A* search.

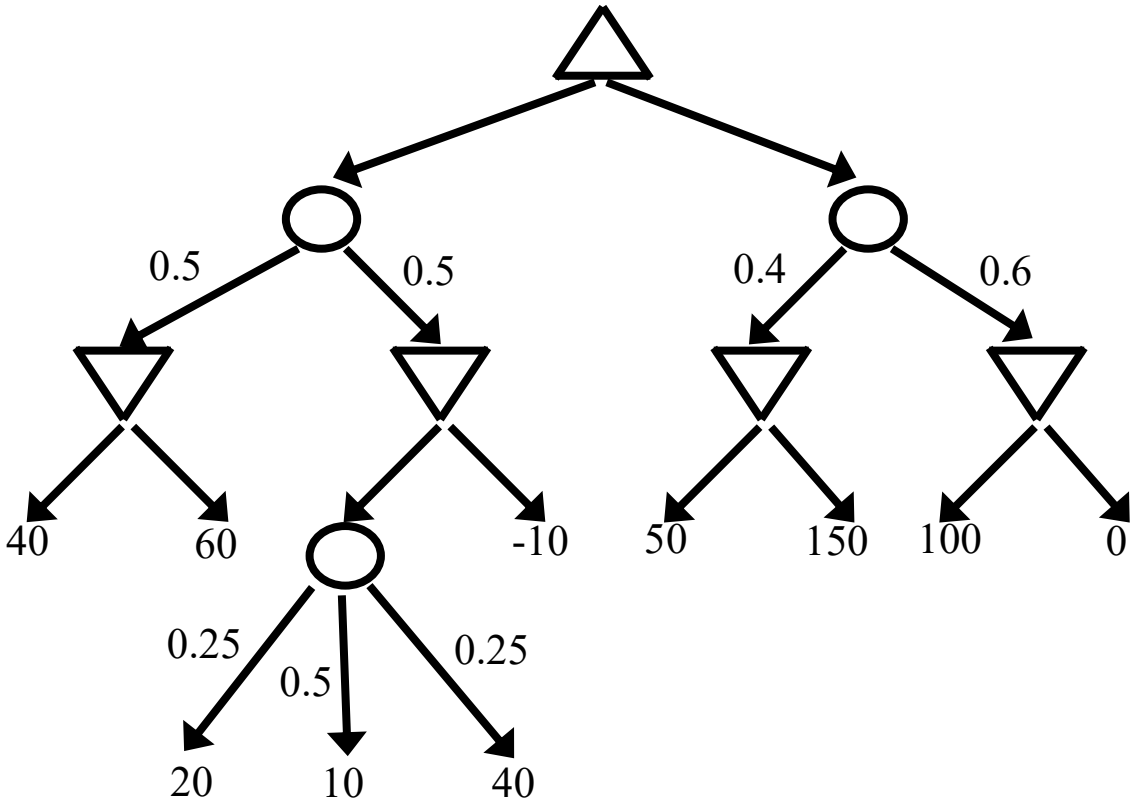
Since we know the starting position, we can know the current position, by updating our position every time we move. Then, a maximally admissible heuristic is the Manhattan distance.

3b. (10 points) Suppose that A* uses an admissible heuristic h such that $h(n) > 0$ for at least one node n in the search tree. Are there scenarios where A* will visit fewer nodes than uniform cost search (UCS)? Are there scenarios where A* will visit more nodes than UCS? Justify your answer.

A* will never visit more nodes, because h(n) is admissible and dominates the zero heuristic which is used by UCS. Since h(n) dominates the zero heuristic, there are cases where A* will visit fewer nodes. An example is given in the textbook for the 8-puzzle, where using A* a very small number of nodes is visited.

Question 4 - 20 points

In the search tree below, indicate the EXPECTIMINIMAX value of each non-terminal node. Assume the MAX player plays first. Triangles pointing up indicate a MAX move, triangles pointing down indicate a MIN move, and circles indicate chance nodes.



Question 5 - 20 points

5a. (10 points) Suppose the computer is playing a game against an opponent, and it is the computer's turn to make a move. The computer runs MINIMAX to figure out the utility of the current game state and to figure out the best move to make. Suppose that the search tree is finite, and that the computer is the MAX player. MINIMAX returns that the utility of the current board state is 5.

When the game finally ends by reaching a terminal game state, what is the range of possible utility values for that game state? Assume that the computer keeps using MINIMAX to decide all its moves, but DO NOT ASSUME ANYTHING about the opponent's strategy, or about the set of possible utility values used for terminal nodes in this game. (You can assume, however, that the opponent will always make legal moves in finite time, and that the opponent will not quit the game before it is over.) A range of possible utility values should be indicated as one of the following options:

- [-infinity, +infinity]
- [lower_bound, +infinity]
- [-infinity, upper_bound]
- [lower_bound, upper_bound]

If your answer includes a lower bound or an upper bound, YOU SHOULD SPECIFY THE EXACT NUMERICAL VALUE of that lower bound or upper bound. Your answer should be as specific as possible (utility values will always be in the [-infinity, +infinity] range, but [-infinity, +infinity] may not always be the most specific answer).

The possible range of results is [5, +infinity]. If the opponent does not follow an optimal strategy, the opponent can make a mistake that leads to a better result than +5.

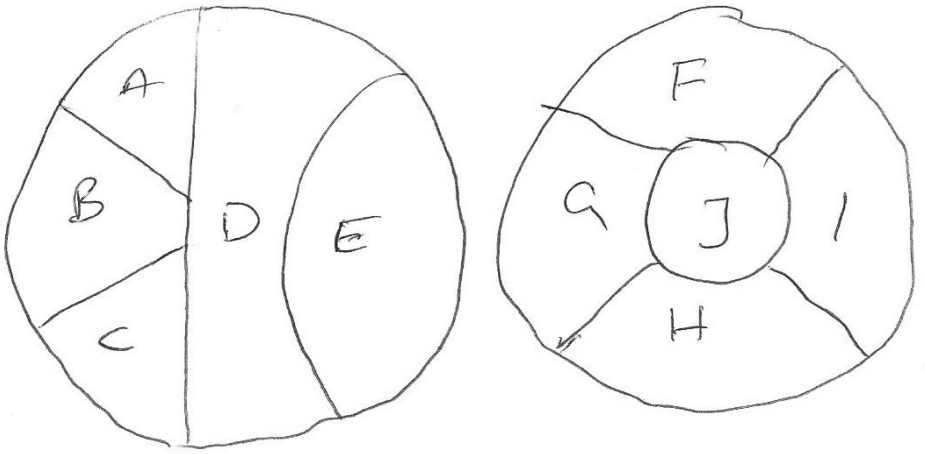
5b. (10 points) Now, suppose that the setting is as in question 5a, except that the game is a game of chance, and the computer runs EXPECTIMINIMAX to figure out the utility of the current game state. EXPECTIMINIMAX returns that the utility of the current board state is 5. When the game finally ends by reaching a terminal game state, what is the range of possible utility values for that game state? A range should be indicated as specified in question 5a above.

The possible range of results is [-infinity, +infinity]

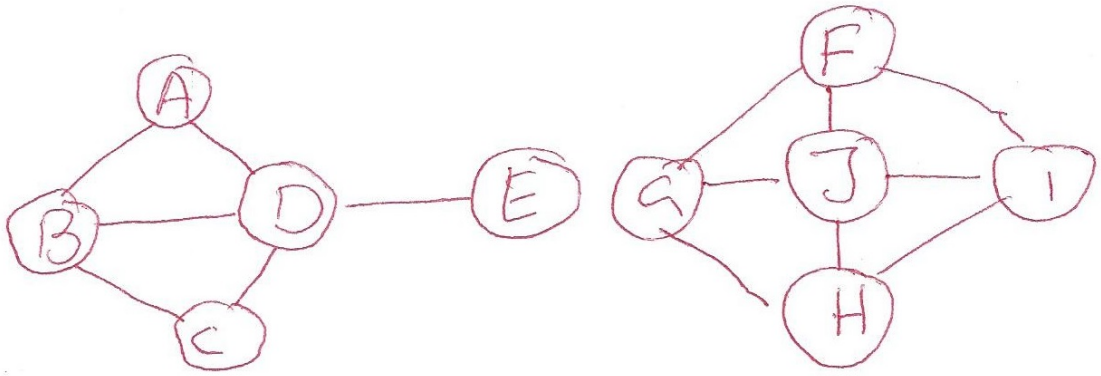
Question 6

40 points.

Your job is to color the various sections such that no two sections sharing a border have the same color. You can use the colors (Red, Green, Blue).



6a. Draw the Constraint Graph for this problem. Use it to separate the problem into subproblems



The 2 connected subgraphs shown here correspond to 2 separate sub-problems. Each of these can be solved separately and the solutions concatenated to solve the entire problem.

6b. Assuming you are using Backtracking search to solve this problem and that you are using both MRV and Degree heuristic to select the variable, Which variable will be selected at each level of the search tree of each subproblems.

Subproblem 1:

1. Unassigned variables are {A,B,C,D,E}
   a. A: RV: 3 Deg: 2, B: RV: 3 Deg: 3, C: RV: 3 Deg: 2, D: RV: 3 Deg: 4, E: RV: 3 Deg: 1
   b. Chosen Variable: D.
2. Unassigned variables are {A,B,C,E}
   a. A: RV: 2 Deg: 1, B: RV: 2 Deg: 2, C: RV: 2 Deg: 1, E: RV: 2 Deg: 0
   b. Chosen Variable: B.
3. Unassigned variables are {A,C,E}
   a. A: RV: 1 Deg: 0, C: RV: 1 Deg: 0, E: RV: 2 Deg: 0
   b. Chosen Variable: A or C (Chose A).
4. Unassigned variables are {C,E}
   a. C: RV: 1 Deg: 0, E: RV: 2 Deg: 0
   b. Chosen Variable: C.
5. Unassigned variables are {E}
   a. E: RV: 2 Deg: 0
   b. Chosen Variable: E.

Subproblem 2:

1. Unassigned variables are {F,G,H,I,J}
   a. F: RV: 3 Deg: 3, G: RV: 3 Deg: 3, H: RV: 3 Deg: 3, I: RV: 3 Deg: 3, J: RV: 3 Deg: 4
   b. Chosen Variable: J.
2. Unassigned variables are {F,G,H,I}
   a. F: RV: 2 Deg: 2, G: RV: 2 Deg: 2, H: RV: 2 Deg: 2, I: RV: 2 Deg: 2
   b. Chosen Variable: F or G or H or I (Chose F).
3. Unassigned variables are {G,H,I}
   a. G: RV: 1 Deg: 1, H: RV: 2 Deg: 2, I: RV: 1 Deg: 1
   b. Chosen Variable: G or I (Chose G).
4. Unassigned variables are {H,I}
   a. H: RV: 1 Deg: 1, I: RV: 1 Deg: 1
   b. Chosen Variable: H or I (Chose H).
5. Unassigned variables are {I}
   a. I: RV: 1 Deg: 0
   b. Chosen Variable: I.

6c. For each of the subproblems, show how forward chaining is used to select values.

Subproblem 1:

1. A: RGB, B: RGB, C: RGB, D: RGB, E: RGB
2. A: GB, B: GB, C: GB, D: R, E: GB
3. A: B, B: G, C: B, D: R, E: GB

4. A: B, B: G, C: B, D: R, E: G

Subproblem 2:

1. F: RGB, G: RGB, H: RGB, I: RGB, J: RGB
2. F: GB, G: GB, H: GB, I: GB, J: R
3. F: G, G: B, H: GB, I: B, J: R
4. F: G, G: B, H: G, I: B, J: R

6d. Assuming you have already assigned A = Red, D = Blue check the arc consistency.

A: R, B: RGB, C: RGB, D: B, E: RGB

A-B
A: R, B: ~~R~~GB, C: RGB, D: B, E: RGB
A-D
A: R, B: ~~R~~GB, C: RGB, D: B, E: RGB
B-D
A: R, B: ~~RG~~B, C: RGB, D: B, E: RGB
B-C
A: R, B: ~~RG~~B, C: R~~G~~B, D: B, E: RGB
C-D
A: R, B: ~~RG~~B, C: R~~GB~~, D: B, E: RGB
D-E
A: R, B: ~~RG~~B, C: R~~GB~~, D: B, E: RG~~B~~

Remaining Legal values
A: R, B: ~~RG~~B, C: R~~GB~~, D: B, E: RG~~B~~