your name

1	2	3	4	5	6	total
45	15	30	30	10	10	**140**

UMBC CMSC 471 Midterm Exam 2022-03-17

Write your answers on this exam, which is closed book and consists of six problems, summing to 140 points. You have the entire class period, seventy-five minutes, to work on the exam. There is a blank page at the end you can use for working out problems, please hand in with your exam. Good luck.

1. True/False [45 points; 3 pts each]

Circle T or F in the space before each statement to indicate whether the statement is **true** or **false**.

T F In fully observable, deterministic environments agents need not deal with uncertainty. TRUE

T F There is generally no difference in the performance of a **depth-first graph search** and a **breadth-first graph search** algorithm for a problem with a finite state-space graph. FALSE

T F If a solution exists and all actions have unit cost, the **iterative deepening** approach to search will always find a solution with the shortest length. TRUE

T F A **depth-first tree-search or graph-search** has an advantage over breadth-first searches in that it will eventually find a solution if one exists. FALSE

T F Using the **min-conflicts local search** technique for solving a search problem will always find a solution, given enough time. FALSE

T F A **local search** algorithm typically has the advantage of needing less memory than a tree-search or graph-search algorithm. TRUE

T F In a constraint satisfaction problem's **constraint graph**, each node represents a constraint and each arc represents a variable. FALSE

T F In solving constraint satisfaction problems, if you use the **arc consistency** algorithm, you do not also have to use the forward checking algorithm to get the best results. TRUE

T F In constraint satisfaction problems variables must have a finite set of possible values. FALSE

T F The **alphabeta algorithm** can only be used for zero-sum games. TRUE

T F The **alphabeta algorithm** is preferred to **minimax** because it gives a more accurate prediction of which move is best for long look-ahead distances. FALSE

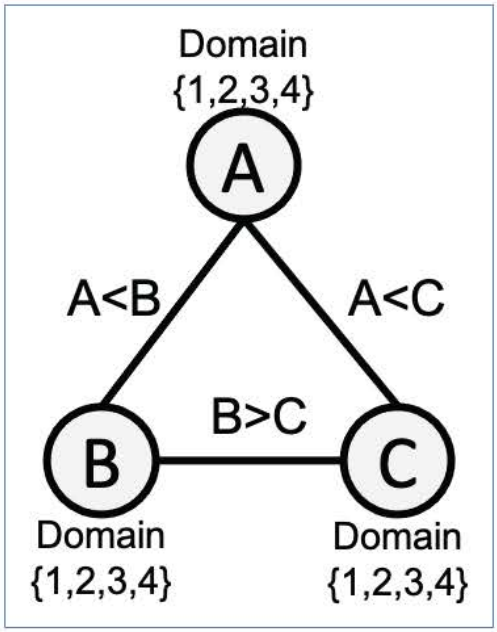
T F If the payoffs for a two-person game make it an example of the prisoner's dilemma, it means that neither player has a **dominant strategy** in a single encounter. FALSE

T F The **min-conflicts algorithm** for solving a constraint satisfaction problem is sound. TRUE

T F The **hill climbing** search algorithms cannot be used to solve problems that have only one dimension. FALSE

T F The **prisoner's dilemma** shows that rational agents should not use game theory. FALSE

2. Constraints [15 points]



Consider a constraint satisfaction problem with three variables: A, B, and C, each with the initial domain {1,2,3,4}. There are three constraints: A<B, A<C, and B>C.

2.1 If we set the value of **B** to be **3** and run the **forward-checking algorithm**, what are the new domains of variables A, B, and C? Express each domain as a set of integers, e.g., {1,2,3,4}, {1,2,3}, {}, … [6 pts]

- A domain: **{1,2}**
- B domain: **{3}**
- C domain: **{1,2}**

2.2 Suppose we have the same constraint problem in the previous question: three variables: A, B, and C, each with the initial domain {1,2,3,4} and three constraints, A<B, A<C, and B>C.

Show each variable's domains after running the **arc consistency algorithm** before assigning a value to any one of them. [6 pts]

- A domain: **{1,2}**
- B domain: **{3,4}**
- C domain: **{2,3}**

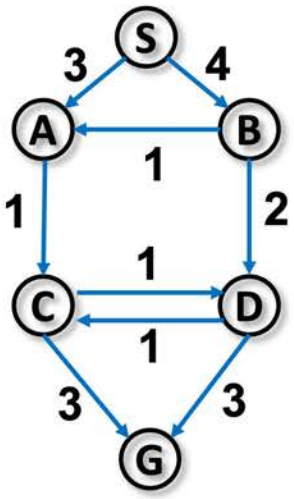
2.3 How many solutions are there? [3 pts]

Four solutions:

- **{A:1, B:3, C:2};**
- **{A:1, B:4, C:2};**
- **{A:1, B:4, C:3};**
- **{A:2, B:4, C:3}**

3. Search [30 points]

Consider this **problem-space graph**, where S is the start node, G is the goal node, and the number next to each edge is the cost to go from the state at its tail to the state at its head.



n	h(n)	h*(n)
S	5	7
A	3	4
B	1	5
C	1	3
D	2	3
G	0	0

The table shows the values of a **heuristic function h** for each state, which estimates the cost of the best path from that state to a goal. The **h*(n) column** will represent the true minimal cost of a path from node n to the goal node G.

We run **algorithm A** on this graph using the heuristic function **h(n)** whose values are given in the table.

3.1 Enter a *string* that is the sequence of the **states expanded** (not just noticed and added to the graph) by the algorithm in the order that they are expanded, e.g., (e.g., SACG). In choosing which node to expand next, assume ties are broken by selecting the one whose name appears first in an alphabetic ordering. [5 pts]

SBACD

3.2 Enter a *string* which is the sequence of the states that represent the final **solution path found** by algorithm A using the given heuristic h(n) from S to G (e.g., SBDG): [5 pts]

SACG

3.3 Enter a *number* that is the **cost of the path** found (e.g., 9): [5 pts]

7

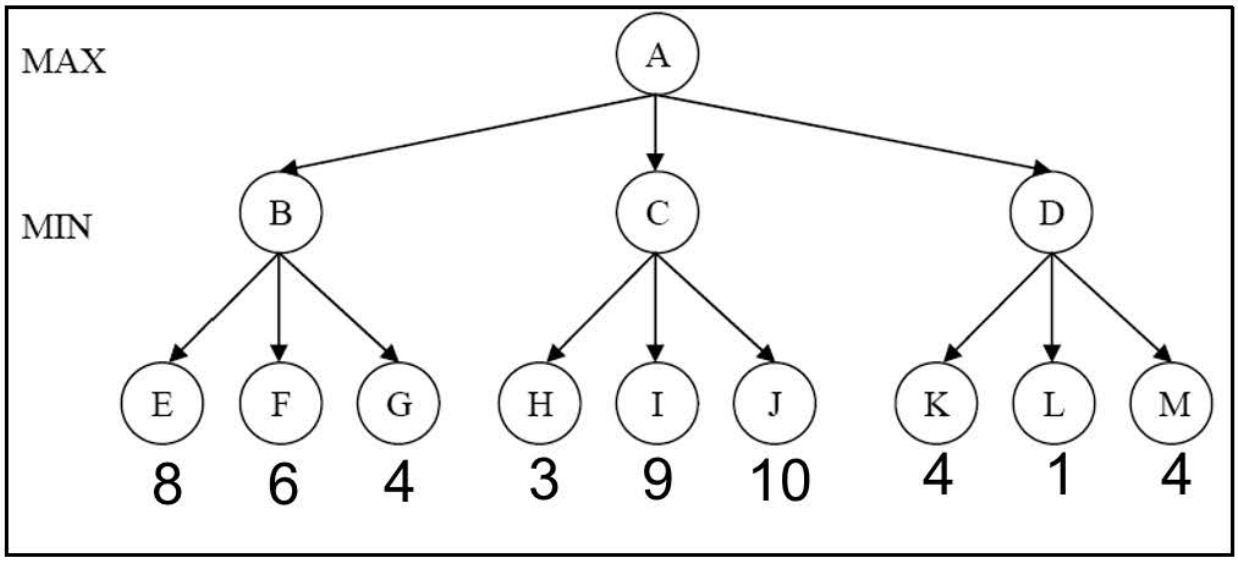
3.4 Is the heuristic function h **admissible**? Enter *yes* or *no*: [5 pts]

YES

3.5 In the table above, fill in the five missing **h*(n) values** (i.e., the true cost of the shortest path from node n to the goal node G for the five non-goal nodes: S; A; B; C; and D. [10 pts]

4. Game Trees [30 pts]

In this game tree, the root is the maximizing player. The numbers under the tree's leaf nodes are the values assigned to the position by a static evaluation function.



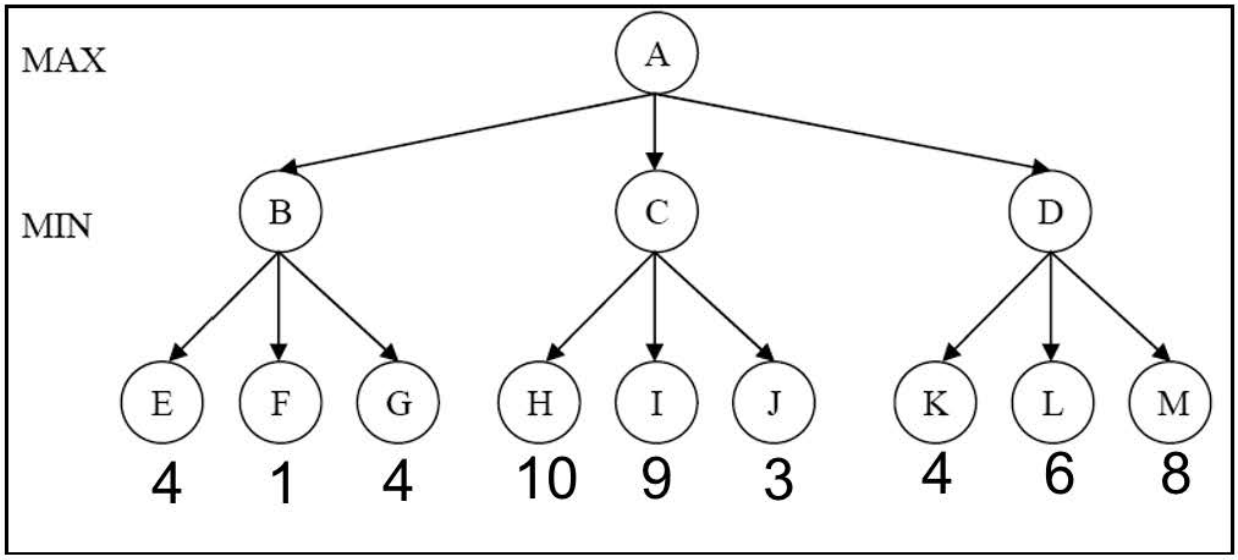
4.1 Using the standard **minimax algorithm**, show the values that are assigned to each of the non-leaf nodes and the move selected by the maximizing player. [10 pts]

- **A:** 4
- **B:** 4
- **C:** 3
- **D:** 1
- **Move selected (i.e., B, C or D):** B

4.2 If the **alpha-beta** algorithm is used, some nodes will be pruned, meaning that their values need not be computed. Enter the letters of the nodes pruned in this tree. If no nodes can be pruned, enter **NONE**. [5 pts]

I J L M

In this game tree, the root is the maximizing player. The numbers under the tree's leaf nodes are the values assigned to the position by a static evaluation function.



4.3 Using the standard **minimax algorithm**, show the values that are assigned to each of the non-leaf nodes. [10 pts]

- **A:** 4
- **B:** 1
- **C:** 3
- **D:** 4
- **Move selected (i.e., B, C or D):** D

4.4 If the **alpha-beta** algorithm is used, some nodes will be pruned, meaning that their values need not be computed. Enter the letters of the nodes pruned in this tree. If no nodes can be pruned, enter **NONE.** [5 pts]

NONE

5. Short answer (10 pts)

In a few sentences, describe a situation where a **depth-first search GRAPH search** algorithm may not find a solution to a search problem even when one exists.

If the search space has an infinite number of states, a depth-first search algorithm might follow an infinite path that never contains a solution. It's possible that another path might lead to a solution in a finite number of steps. For example, consider a problem of finding the first prime larger than one million. A path that explores all of the even numbers larger than 1M will continue forever without finding a solution.

Note: Since we have only asked about graph search, we do not need to be worried about loops. Graph search does not have a problem with loops.

6. Short answer (10 pts)

In the problem solving as search paradigm, explain in a few sentences why it does or does not make sense to allow an **action whose result is the same state as the one it is performed in**. Mention issues for both tree-based and graph-based search algorithms.

For both tree and graph search, such self-loops can never be part of a shortest solution. If they have positive cost, performing the action will add to the solution cost without getting us closer to a goal. If they have a zero cost, there is still no need to perform the action, since it does not get us closer to a goal.

For tree search, there is an additional and more serious problem. The loop will make it see that there are an infinite number of states even if there are not, since tree search does not recognize a state as being one it has seen before. This can greatly increase the amount of time and space needed to solve the problem or, worse yet, produce infinitely long paths that will prevent some algorithms from finding a solution when one exists.

Note: take off 5 points if an answer did not mention the key issues for both tree search and graph search algorithms.