Name

1	2	3	4	5	total
10	10	35	35	35	125

UMBC CMSC 471 Midterm Exam 21 March 2016

Write your answers on this exam, which is closed book and consists of five problems, summing to 125 points. You have the entire class period, seventy-five minutes, to work on this exam. Good luck.

1. True/False [10 points]

Circle either **T** or **F** in the space before each statement to indicate whether the statement is true or false. If you think the answer is simultaneously true and false, quit while you are ahead.

T F Alan Turing proposed his famous Turing test as a technique for deciding whether or not a problem was "Turing computable".
FALSE. The Turing test measures "intelligence" and has no direct relation to computability.

T F A greedy, best-first search algorithm is always complete.
FALSE. Infinite loops can cause it to fail to find a solution

T F A simple breadth-first search always finds a shortest solution if one exists that is of finite length.
TRUE

T F For a search problem, the path returned by uniform cost search may change if we add a positive constant C to every step cost.
TRUE. Given two paths from S to G: S→A→G and S→G where cost(S,A)=1, cost(A,G)=1, and cost(S,G)=3, he optimal path is through A. Adding two to each arc cost makes the optimal path S→G. Since uniform cost search finds the optimal path, its path will change.

T F The alpha-beta algorithm is preferred to minimax because it provides a better estimation of which move is best for a given lookahead distance.
FALSE. Alpha-beta and minimax always return the same values.

T F In a two-player, zero-sum game there is always a winner and a loser.
FALSE. Many such games can result in a draw

T F In a prisoners' dilemma game, each player chooses a dominant strategy, but each could do better if both chose different strategies.
TRUE.

T F Game theory predicts that players will always have a dominant strategy.
FALSE. Many games do not have a dominant strategy

T F Tit-for-tat is a strategy that cannot be applied in repeated games.
FALSE. Tit-for-tat is a popular strategy in games in with repeated encounters

T F If h1(s) and h2(s) are admissible A* heuristics, then their average, (h1(s) + h2(s))/2 must also be admissible.
TRUE. Let h(s) be the true distance from s. We know that h1(s) ≤ h(s) and h2(s) ≤ h(s), thus f(s) = ½ h1(s) + ½ h2(s) ≤ 1/2 h(s) + ½ h(s) = h(s)

2. Multiple Choice [10 points]

Circle *all* of the correct answers.

2.1 [2] Which of the following search algorithms finds an optimal solution?

a. Breadth first
b. Hill climbing
c. Depth first
d. Greedy search
e. None of the above

a

2.2 [2] A search algorithm is *complete* if it…

a. Always finds the optimal solution
b. Always finds a solution if there is one
c. Finds all possible solutions
d. Never finds a solution

b, c

2.3 [2] Which of these techniques uses randomness to avoiding getting trapped in local maxima?

a. Best first search
b. Local beam search
c. Simulated annealing
d. Gradient descent
e. None of the above

c

2.4 [2] Which of the following is a problem that occurs in hill climbing search?

a. Cliffs
b. Ridges
c. Valleys
d. "Slippery slopes"
e. Local maxima and minima
f. Cycles in the graph
g. None of the above

b, e

2.5 [2] A greedy search uses a heuristic function to expand the node that

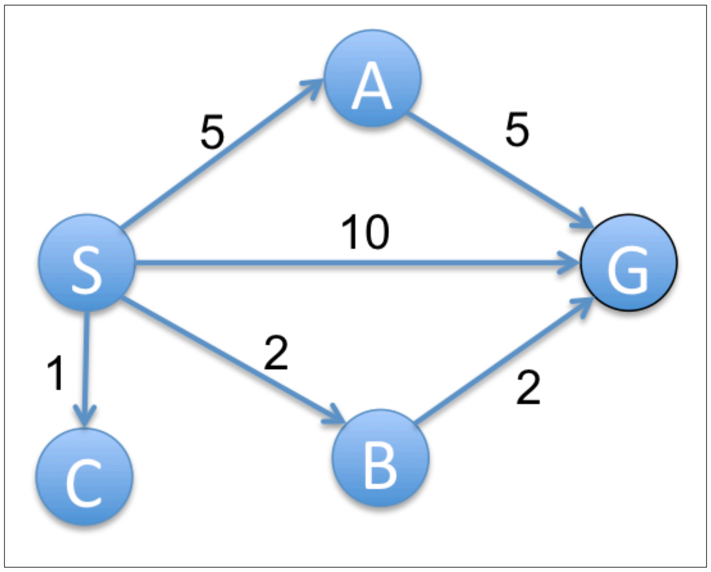
a. Appears to be closest to the goal
b. Is closest to a goal
c. Is closest to the start state
d. Is the leftmost node in the search tree
e. None of the above

a

3. Search I [35 points]

Consider the search space with this graph, where S is the start state and G is the goal. Assume a heuristic is available for the A* algorithm that has the following values:

{S:4, A:3, B:2, C:100, G:0}



For questions a-e specify the path as a sequence of nodes starting at S and ending at G that each algorithm returns. Assume the successor functions work so that nodes are explored in alphabetical order whenever possible.

(a) [5] Breadth-first search

$S \rightarrow G$

(b) [5] Depth-first search

$S \rightarrow A \rightarrow G$

(c) [5] Uniform-cost search

$S \rightarrow B \rightarrow G$

(d) [5] A* search

$S \rightarrow B \rightarrow G$

(e) [5] Greedy search

$S \rightarrow G$

(f) [5] Name a node that uniform-cost search will expand, but A* will not.

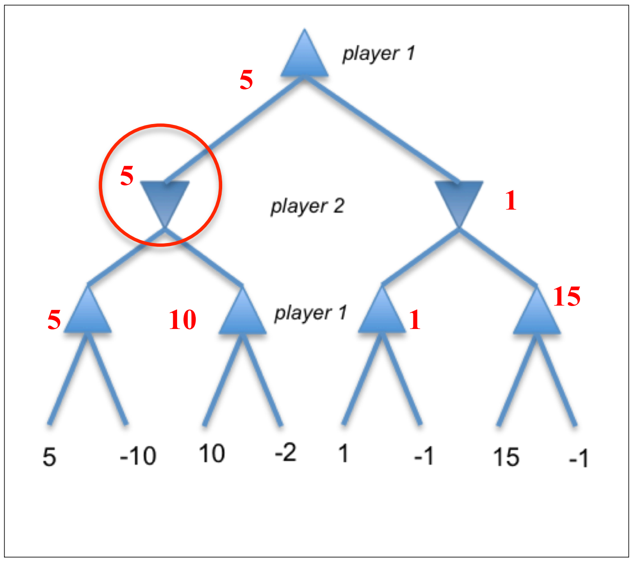
C

(g) [5] Judging just from the heuristic function for these nodes, is it an admissible heuristic?

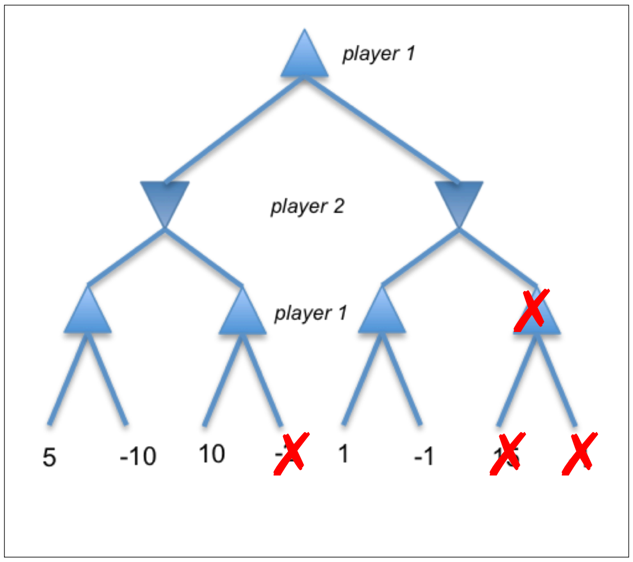
Yes

4. Minimax and Expectimax [35]

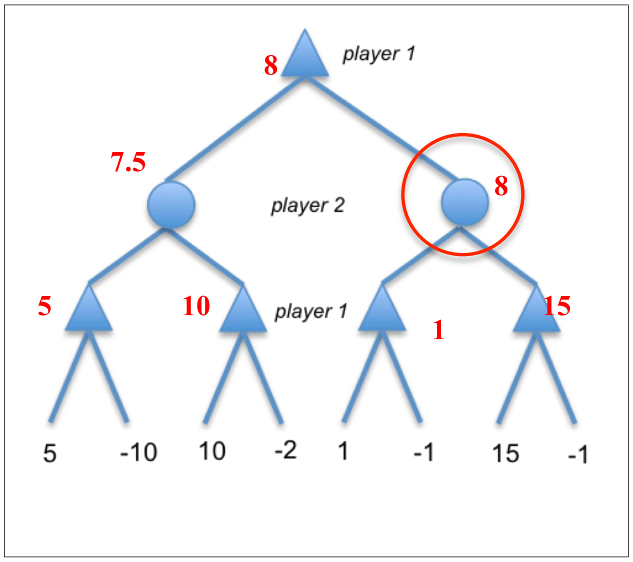
4.1 [6] Consider a zero-sum game with two players. Each leaf is labeled with the payoff player 1 receives. It is player 1's turn to move. Assume both play optimally at every ply (i.e. player 1 seeks to maximize payoff while Player 2 seeks to minimize it). Circle player 1's best next move on the graph, and show the minimax values for each node in the tree.



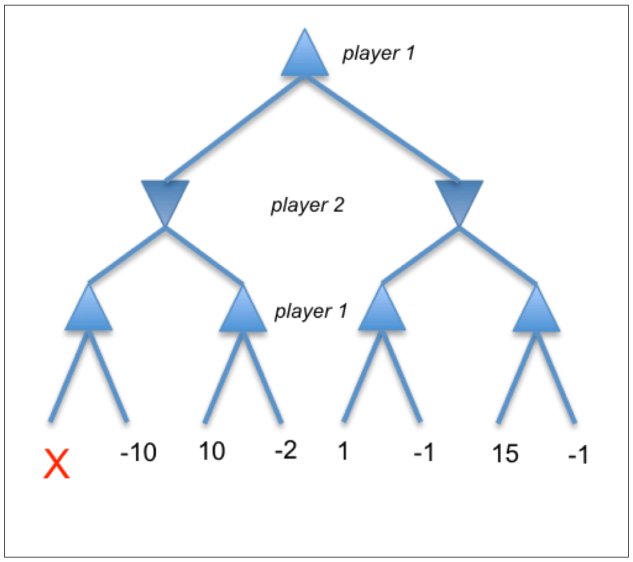
4.2 [6] Consider this game tree. Player 1 moves first, and attempts to maximize the expected payoff. Player 2 moves second, and attempts to minimize the expected payoff. Expand nodes left to right. Cross out both internal and leaf nodes that would be pruned by the alpha-beta algorithm



4.3 [6] Assume that player 2 chooses an action uniformly *at random* every turn and that player 1 knows this. Player 1 still seeks to maximize her payoff. Circle player 1's optimal next move, and give her expected payoff. Show the expected value at each node in the tree.



4.4 [6] Now consider a modified version of the game tree, where the leftmost leaf node has an unknown payoff **X**. Player 1 moves first, and attempts to maximize the value of the game.



4.5 [3] Assume player 2 is a minimizing agent (and Player 1 knows this). For what values of **X** does player 1 choose the left action?

X > 1 or X ≥ 1, depending on how an implementation breaks ties.

4.6 [3] Assume Player 2 chooses actions at random (and Player 1 knows this). For what values of **X** does Player 1 choose the left action?

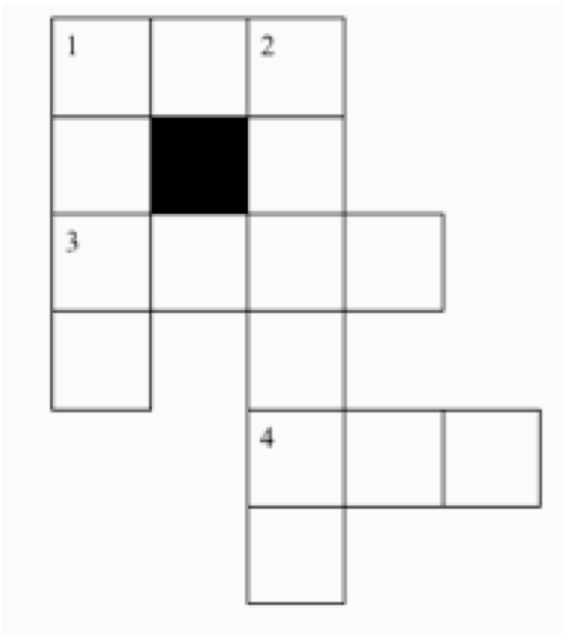
X > 6 or X ≥ 6, depending on how an implementation breaks ties.

4.7 [5] For what values of **X** is the minimax value backed up (i.e., when player2 chooses rationally) to the the tree's root worth more than the expectimax value at the root (i.e., when player 2 chooses randomly)?

The tree's minimax value can never exceed it's expectimax value. For X=10, the two cases are equal. For the rational/minimax case, the value at the root is max(min(x,10), 1), so it can never exceed 10. For the expectimax/random case it is max((x+10)/2, 8).

5. Constraint Satisfaction [35]

Consider the crossword puzzle shown on the right. Suppose we have the following words in our dictionary: *ant, ape, big, bus, car, has, bard, book, buys, hold, lane, year, rank, browns, ginger, symbol, syntax*. The goal is to fill the puzzle with words from the dictionary.

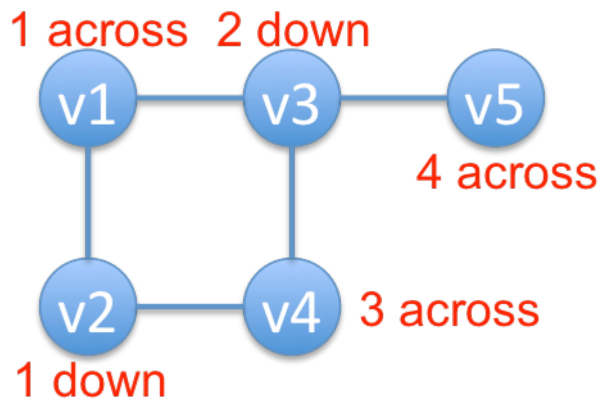


This problem asks you to set the problem up for solution as a constraint satisfaction by specifying the variables, their domains and constraints.

5.1 [10] Describe a set of variable for this problem and give the initial domain for each. If you need more table rows, use the back of this sheet.

var	description	Initial domain
V1	**1 across**	***3-letter words: ant, ape, big, bus, car, has***
V2	**1 down**	**4 letter words:** ***bard, book, buys, hold, lane, year, ran***
V3	**2 down**	**6 letter words:** ***browns, ginger, symbol, syntax***
V4	**3 across**	**4 letter words:** ***bard, book, buys, hold, lane, year, ran***
V5	**4 across**	***3-letter words: ant, ape, big, bus, car, has***
V6		
V7		
V8		
V9		
V10		

5.2 [10] Draw a constraint graph for the problem, e.g., a graph with variables as nodes and an edge between variables when they have a constraint between their values.



5.3 [5] Describe how you could represent a variable value in Python, e.g., as an integer, float, character, string, tuple, list, dictionary, set, class instance, etc.

Values can be simple strings representing the dictionary words, e.g., v1=”ape”

5.3 [10] For this representation describe each of the constraints. Feel free to use Python to describe it or a simple sentence. If you need more table rows, use the back of this sheet.

Var1	Var2	Constraint
v1	**v2**	**v1[0] == v2[0] # first char of 1 across must equal first char of 1 down**
v1	**v3**	**v1[-1] == v3[0] # last char of 1 across must equal first char of 2 down**
v1	**v4**	**v1[2] == v4[0] # third char in 1 down must equal first char 3 across**
v3	**V5**	**v3[4] == v5[0] # fifth char of 2 down must equal first char of 4 across**
V3	**V4**	**v3[2] == v4[2] # third chars of 2 down and 3 across must equal**