CS 109: Introduction to Computer Science

Goodney

Fall 2017

Homework Assignment 4

Assigned: 11/13/17 via Blackboard

Due: 2359 (i.e. 11:59:00 pm) on 1/27/17 via Blackboard

Notes:

a. This is the fourth homework assignment of CS 109. It is worth 7.5% of your overall grade for this class. You must solve all problems correctly to obtain full credit.

b. You are free to discuss the problems on this assignment with your classmates. However, to receive credit, you must write up and submit solutions completely on your own. You are responsible for understanding your answers. The purpose of discussing the questions with your classmates is to deepen your understanding of the material – not simply to obtain answers from them. To get the most out of the class, you are strongly encouraged to make a serious effort to understand and solve the problems on your own before discussing them with others.

c. On the work you turn in, you must list the names of everyone with whom you discussed the assignment.

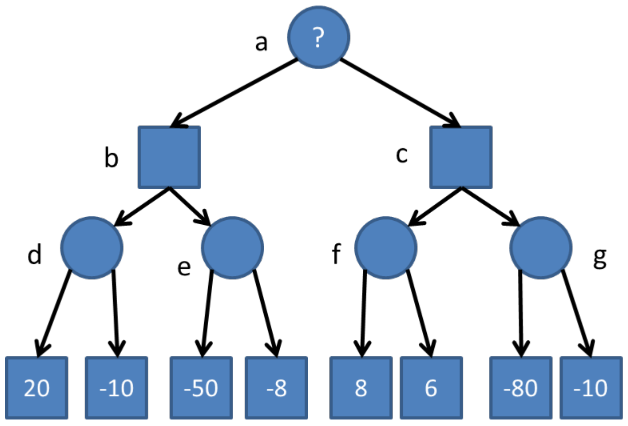
d. All answers must be typed, not handwritten. All answers must be typed, not handwritten (other than pictures of hand drawn figures that you may embed in your PDF). The homework must be turned in as a single **PDF** document on Blackboard. Other formats will not be graded.

Question 1: Minimax and game trees [2.5 points]

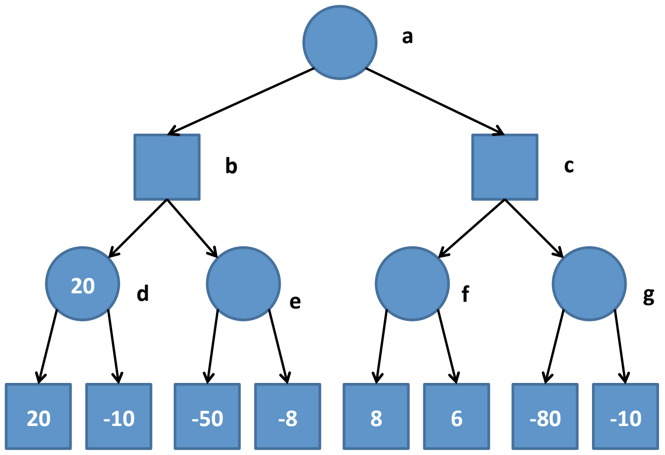
One way to program a computer to play a game against an opponent is to use the Minimax algorithm. In this question, we will first work through a helpful example using the algorithm, and then ask you to apply it to the game of Tic-Tac-Toe.

Each state of the board in a game is given a score, with high score states assumed to be 'good' for the computer, and low score states 'good' for the opponent. When it is the computer's turn to play, it picks the move that leads to a board state with the highest score. When it is the opponent's turn to play, we assume that the opponent picks its best move (i.e. that leads to a board state with the lowest score).

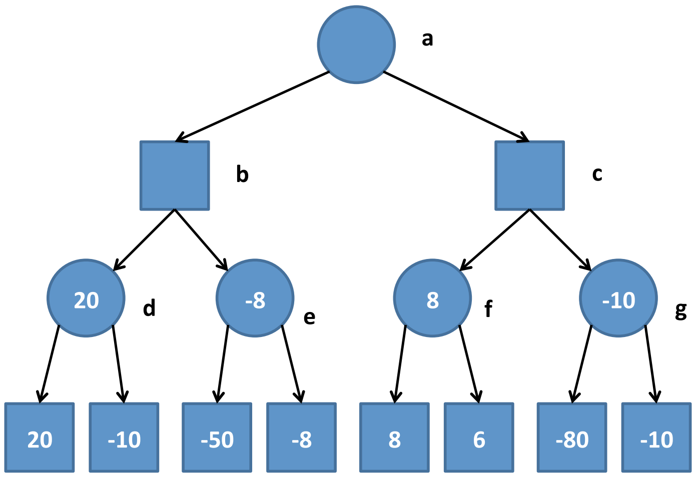
Assume the leaves of the following tree represent outcomes of the game. Assume it is the computer's turn to make a move in board state **a**. Let's use Minimax to determine the score of the current position and choose whether to make a move leading to state **b** or **c.**



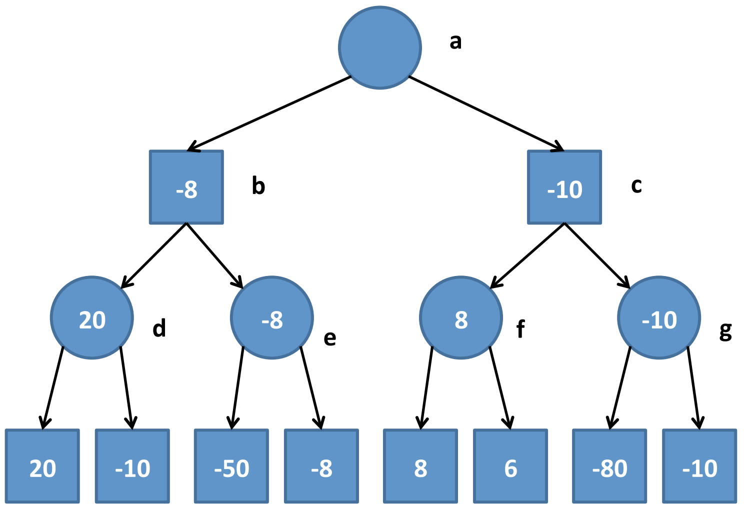
Working backwards, we first compute the scores for the game-ending board states **d**, **e**, **f**, and **g**. Since it is the computer's turn in state **a**, states **b** and **c** are the opponent's turn, and it will be the computer's turn in **d**, **e**, **f**, and **g**. In each case, the computer has two choices. For example, in state **d**, the computer can choose a move that leads to a state with score 20 (good for the computer) or a move that leads to a state with score -10 (bad for the computer). According to the Minimax algorithm, the computer should choose the move that leads to a "max" score state, i.e. score 20. This is shown below.



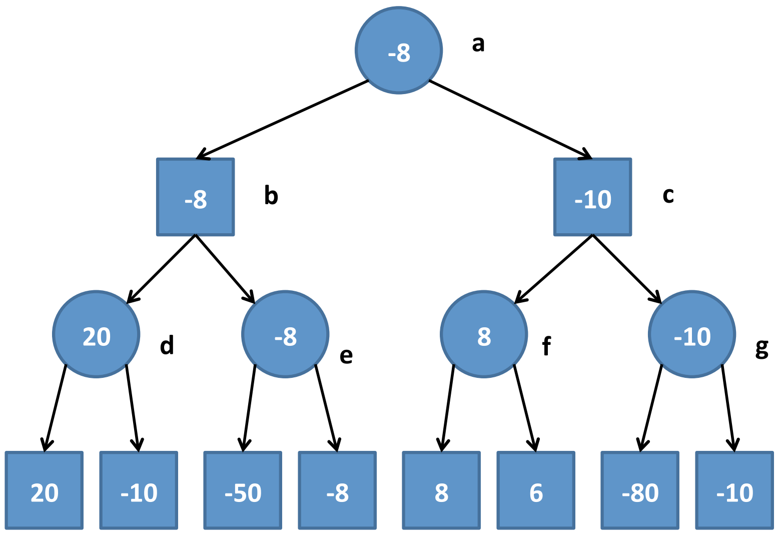
Similarly, the computer would choose the move that maximizes the score in states **e, f,** g. These three are shown filled in below.



Now consider state **b** and state **c**, where it is the opponent's move. For example, in state **b**, the opponent can choose a move leading to state **d** or a move leading to state **e**. In state **c**, one choice leads to state **f**, and the other to state **g**. Low scores are good for the opponent, so it will pick the move leading to the minimum score (the "Mini" in the Minimax algorithm). So in state **b** the opponent will pick the move that leads to state **e** with score -8, and in state **c** the opponent will pick the move that leads to state **f** with score -10.

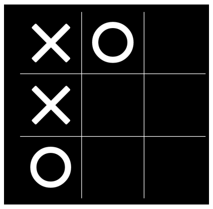


In state **a**, there are two choices for the computer's move . One leads to a state whose score is -8 and other leads to a state whose score is -10. The computer will take the "max" score move, leading to state **b.** By working backwards from the leaf nodes, we have shown that this move will lead to the best end-game outcome for the computer, assuming the opponent plays optimally.



Application to tic-tac-toe

Suppose you (X) are playing the game of tic-tac-toe with your friend (O). You are in the middle of the game where the position is the following and it is your turn (X to play).



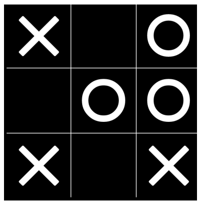
If both players play with the best possible strategy, this game will end in a win for X.

1.1 Draw a decision tree starting from this position to show all possible moves. Use +1 to score your victory, 0 for a draw, and -1 for your loss. (1.5 points)

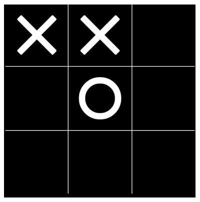
1.2 Use the Minimax algorithm to score the positions and find the winning strategy (winning set of moves). (1 point)

Note the following comments before drawing the tree.

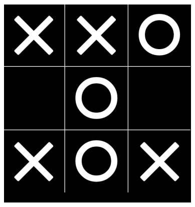
1. You may assume that both players are smart enough so that if they can win in a single move, they will. Therefore the state to the right with X to play has a score of +1. When you draw the game tree you need not expand this position further.

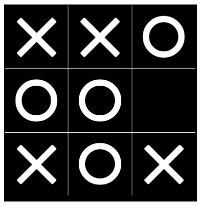


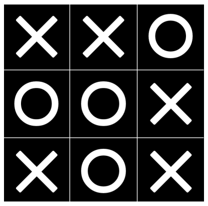
2. In our model of tic-tac-toe game play, a player must block the opponent's immediate threat of making 3 in a row. Any other choice is considered illegal. For instance, in this position, the only legal move for O is the top-right corner, and must be played (a forced move).



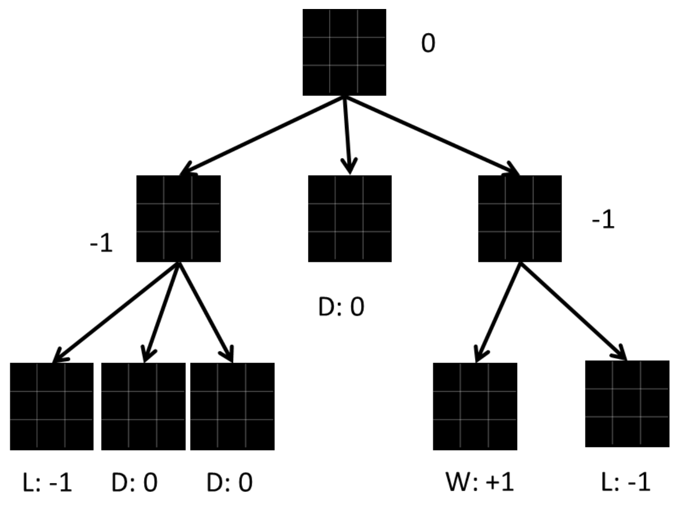
3. Similarly if a draw is forced (there is exactly one way to play for each player), you do not have to play it out to the end, and you can assign the position a score of 0. For instance the position shown on the right is a forced draw position (O is to play next) because the only legal continuation is a draw (as shown in the two figures below).







4. A sample minimax tree may look like the one below (though when you draw yours you will need to fill in the Xs and Os in each of the states). The root node represents the current position of a game and it is your turn to play.



5. Note – it's perfectly OK to draw the tree by hand, take a picture and embed the picture in your HW response.

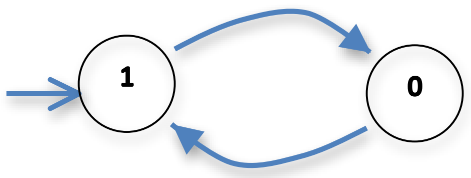
Question 2: Finite state machines [1.5 points]

As we saw in class, finite state machines are limited in what they can do but they are nevertheless useful to model some kinds of computation. We will show several examples illustrating ways to use FSMs, and then ask you to apply these modeling techniques on your own.

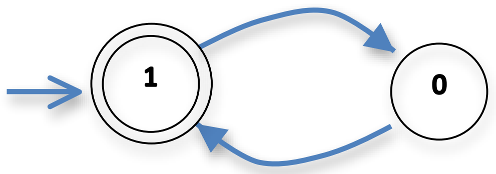
For example, suppose you want model the following computation:

> Input: a string of letters (e.g., *abcd* or *weerycbn* or *lfjkurbtnc*)
> Output: 1 if the number of letters is even

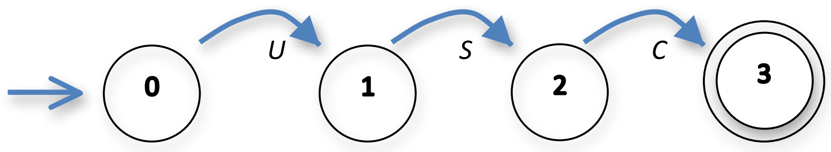
A finite state machine that models this computation is shown below:



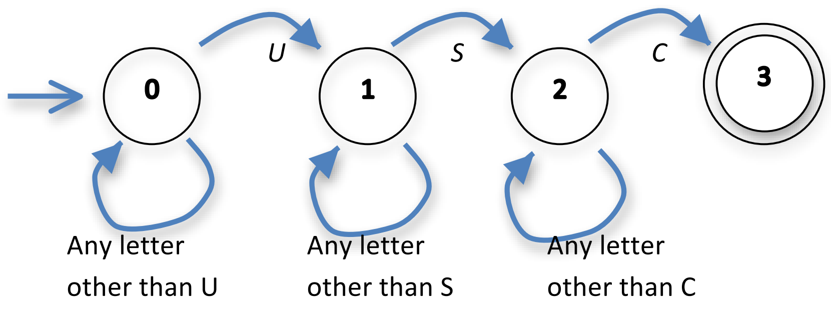
The way you interpret this picture is as follows. Let's assume the input is the string **abcd**. The first input is the letter '***a***'. The machine begins in state 1 (that's what the leftmost arrow pointing to state 1 means), takes the letter ***a*** as input and transitions to state 0. The next input is the letter ***b***. The machine takes the letter ***b*** as input and transitions to state 1. The next input is the letter ***c***. The machine takes the letter ***c*** as input and transitions to state 0. The final input is the letter ***d***. The machine takes the letter ***d*** as input and transitions to state 1. Since there is no more input, the machine outputs the state it is in, which is 1. The state 1 is called the terminal state and is usually shown as below (with two circles). If the machine finds itself in this state and there is no more input, it prints the state it is in (in this case, 1)



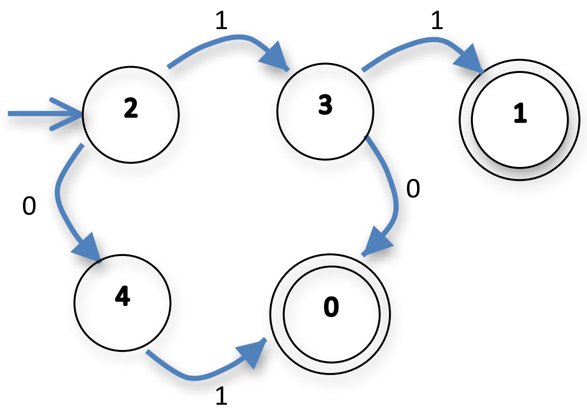
Here is another example. This machine prints 3 if it is given the input string *USC*.



We may modify this example to print 3 if the input string contains *USC* within it (not necessarily contiguous). For example, it will print 3 for any input that contains a *U* followed by an *S* followed by a *C* (even if there are letters in between, e.g. *UASC* or *UASBC* or *XYZULMNSC* or *UUSSC*).



As a final illustration consider the following. Suppose we want a finite state machine that prints the result of the logical AND operation on exactly two bits. This means that if the input bit string is 10, we want the machine to print 0 (since the logical AND of 1 and 0 is 0). If the input bit string is 01, we want the machine to print 0 (since the logical AND of 0 and 1 is 0). If the input bit string is 00, we want the machine to print 0 (since the logical AND of 0 and 0 is 0). If the input bit string is 11, we want the machine to print 01 (since the logical AND of 1 and 1 is 1). Here is what the machine looks like:



Note, we start in state 2 by evaluating the left-most bit. If it is 1, we move to state 3, if it is 0 we move to state 4. The next arrows evaluate the second bit, leading to our desired result of 0 or 1 for the logical AND.

2.1. Draw a finite state machine that prints the result of the logical OR operation on a 2 bit input string made up of 0s and 1s. (0.5 points)

2.2. Draw a finite state machine that takes arbitrary length strings of letters as input and prints a 1 if the string contains the letter S exactly three times. Assume the alphabet is limited and only the letters S and C can be used as input letters. (0.5 points)

2.3. Draw a finite state machine that that takes arbitrary length strings of letters as input and prints a 1 if the string contains the letter S exactly as many times as the letter C. Assume the alphabet is limited and only the letters S and C can be used as input letters. (0.5 points)

Question 3: Turing Machines [3.5 points]

As we discussed in class, a Turing machine can be represented either as a Table or as a State Diagram. Below you will find a table for a Turing machine.

State / Input	**a**	**b**	**_**
0	*_, Right, 1*	*_, Right, 2*	*_, Left, 5*
1	*a, Right, 1*	*b, Right, 1*	*_, Left, 3*
2	*a, Right, 2*	*b, Right, 2*	*_, Left, 4*
3	*_, Left, 7*	*b, Left, 6*	*_, Left, 5*
4	*a, Left, 6*	*_, Left, 7*	*_, Left, 5*
5	*Accept*	*Accept*	*Accept*
6	*Reject*	*Reject*	*Reject*
7	*a, Left, 7*	*b, Left, 7*	*_, Right, 0*

Note that states 5 and 6 are special states. In both of them, the machine halts on any input. In state 5, the machine halts in an 'accept' state. In state 6 the machine halts in a 'reject' state. This means that if the machine ends up in state 5, it 'accepts' the input string it has processed. If the machine ends up in state 6, it 'rejects' the input string it has processed. As you will see, this machine accepts certain kinds of strings and rejects others.

3.1. Draw the equivalent State Diagram. (1 point)

Here is example of the step-by-step execution for an initial tape input of *aa*. Blank cells are equivalent to the _ symbol; the asterisk (*) indicates the location of the head at each step. The machine always begins in state 0 with the head on the leftmost non-blank square. The machine stops when it is in state 5 or state 6.

Step	**State**	**Tape**								**Transition**	**Note**
1	*0*		*a	a						*_, Right, 1*	Found an a. Write _, switch to 1.
2	*1*			*a						*a, Right, 1*	Found an a. Write a, stay in 1
3	*1*			a	*					*_, Left, 3*	Found a _.Write _, switch to 3
4	*3*			*a						*_, Left, 7*	Found an a. Write _, switch to 7
5	*7*		*							*_, Right, 0*	Found a _. Write _, switch to 0
6	*0*			*						*_, Left, 5*	Found a _. Write _, switch to 5
7	*5*		*							*Accept*	

3.2. Give the step-by-step execution of the machine for an initial tape input of *ababa*. In what state does the machine end? Use formatting similar to the above. Keep in mind that the machine begins in State 0 and the head's initial position is on the leftmost non-blank square. The machine stops when it is in state 5 or state 6. (1 point)

3.3. Give the step-by-step execution of the machine for an initial tape input of *abb*. In what state does the machine end? Use formatting similar to the above. Keep in mind that the machine begins in State 0 and the head's initial position is on the leftmost non-blank square. The machine stops when it is in state 5 or state 6. (1 point)

3.4. Explain in a single sentence what this Turing machine is doing, i.e., what kinds of strings does it accept and what kinds of strings does it reject. (0.5 points)