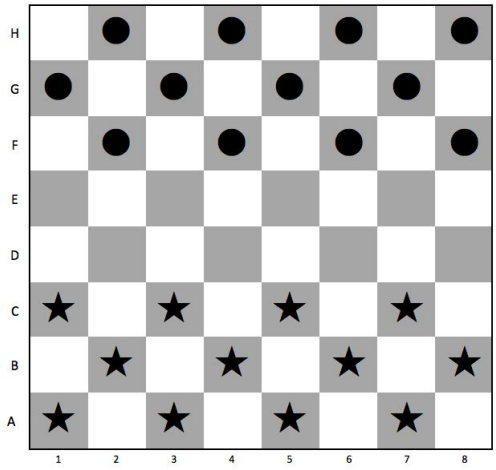
CSCI-561 - Spring 2018 - Foundations of Artificial Intelligence
Homework 1

Due February 5, 2018 23:59:59

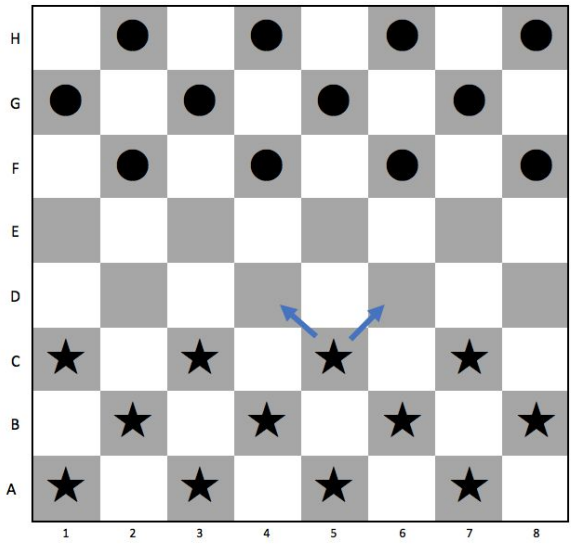
Problem Description (Rules of the game)

This game is played by two players, on opposite sides of the game board. The game is played on a board of fixed size 8x8. One player has the Star pieces; the other has the Circle pieces. Player Star plays from the bottom of the board to the top (from A to H) and player Circle plays from the top of the board to the bottom (from H to A). The initial state of the board is shown in the figure below.

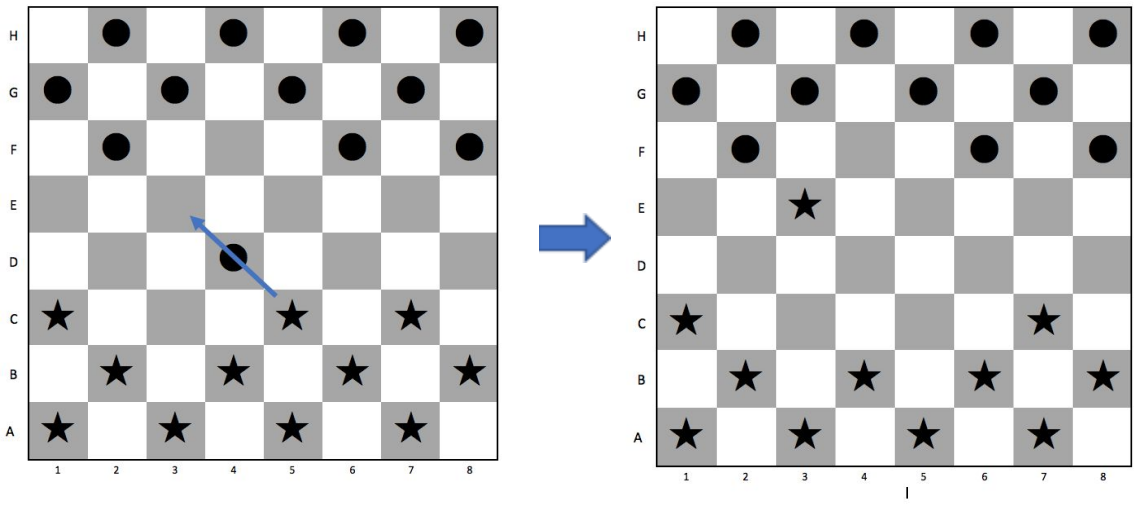


Only the dark squares of the board are used. Players alternate turns. When it is a player's turn (say player Star's turn), it can do one of the following:

1- Move one of its pieces diagonally to an adjacent unoccupied square on the next row. See figure below.



2- If the adjacent square contains an opponent's piece, and the square immediately beyond it (diagonally) is vacant, the opponent's piece can be captured (and removed from the game) by jumping over it. See figures below.



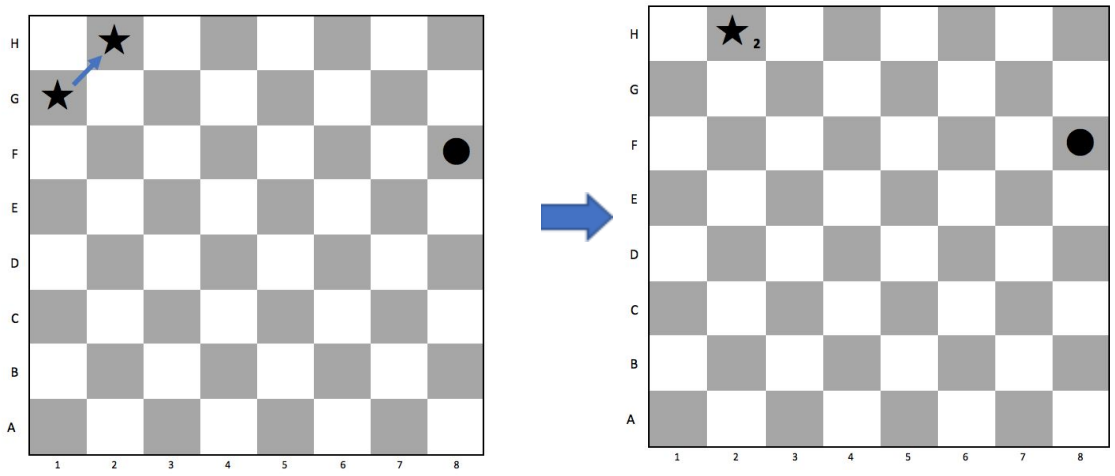
3- Pass the move to the other player. A player can only pass if it cannot move any of its pieces.

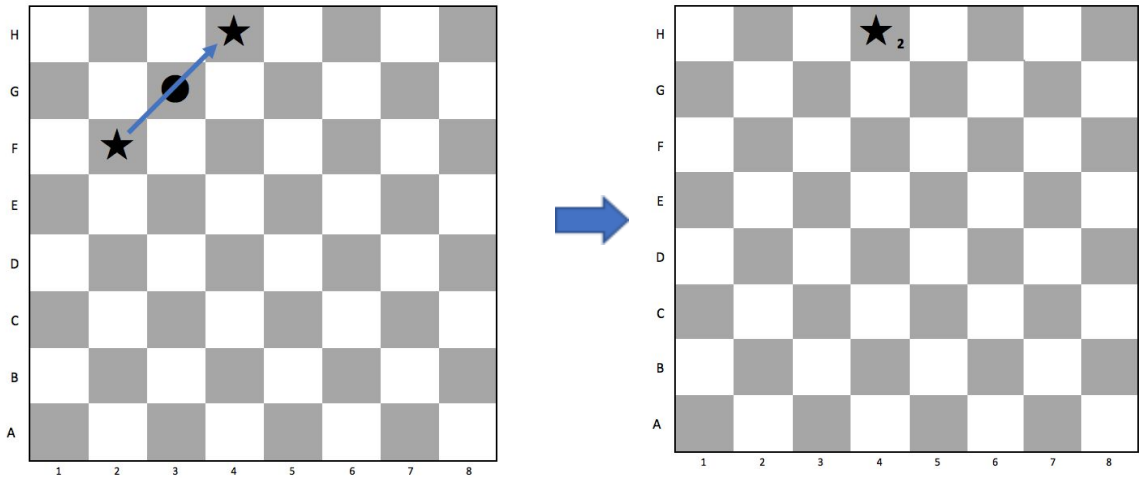
Note that no backward movement is allowed in this game. According to the description above, any non-pass move in this game is specified by *"{row letter of initial position}{ column number of initial position}-{row letter of final position}{ column number of final position}"* of the piece the player chooses to move. For example, "C5-D4" describes a

move where a piece has been moved from square C5 to square D4. The pass move is denoted by “pass”.

Last Row Properties

Multiple pieces can be placed on each dark square of the last row (row H for player Star and row A for player Circle), as no valid move is available for pieces located in the last row. See figures below.





Pass Move and End of Game

If one player cannot make a valid move, it must play a “pass” move, which makes no change on the board, and the game goes to the other player’s turn. A player cannot pass when it has at least one valid move.

The game ends in two cases:

1- When neither player can make a valid move, the game ends. In other words, when a player plays a “pass”, and the other player in response also plays a “pass”, the game ends.

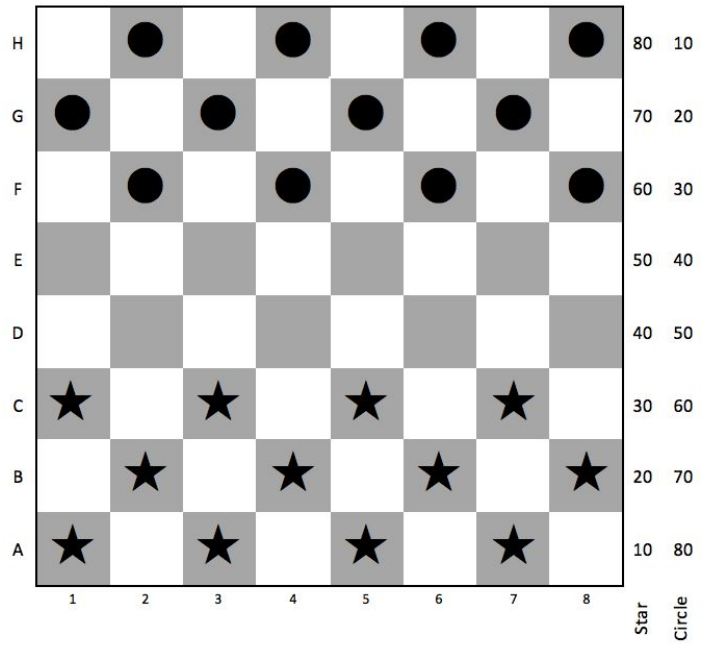
2- When the game reaches a state where only one player has pieces on the board.

Your Task

In this assignment, you will implement the Minimax and Alpha-Beta algorithms to determine the depth-bounded minimax values of given game positions. Your program will take the input from the file “input.txt” and print out its output to the file “output.txt”. Each input file contains a game position (including the board state and the player to move), a search cut-off depth *D*, the values of board rows, and the name of the algorithm to be used (Minimax or Alpha-Beta). Your program should output the corresponding information after running the specified search algorithm to depth *D*. That is, the leaf nodes of the corresponding game tree should be either a game position after exactly *D* moves (alternating between Star and Circle) or an end-game position after no more than *D* moves.

Evaluation function: positional weights

Each square of the board has a certain strategic value, W_i. We assume all squares of a row have the same value. The figure below shows a possible set of values for the rows.



Given the weights of the squares, the evaluation function of a given game position, *s,* (with respect to a specific player) can be computed by

$$E^p(s) = \sum_{i \in all\ squares} \alpha_i^p W_i^p - \sum_{j \in all\ squares} \alpha_j^{p'} W_j^{p'}$$

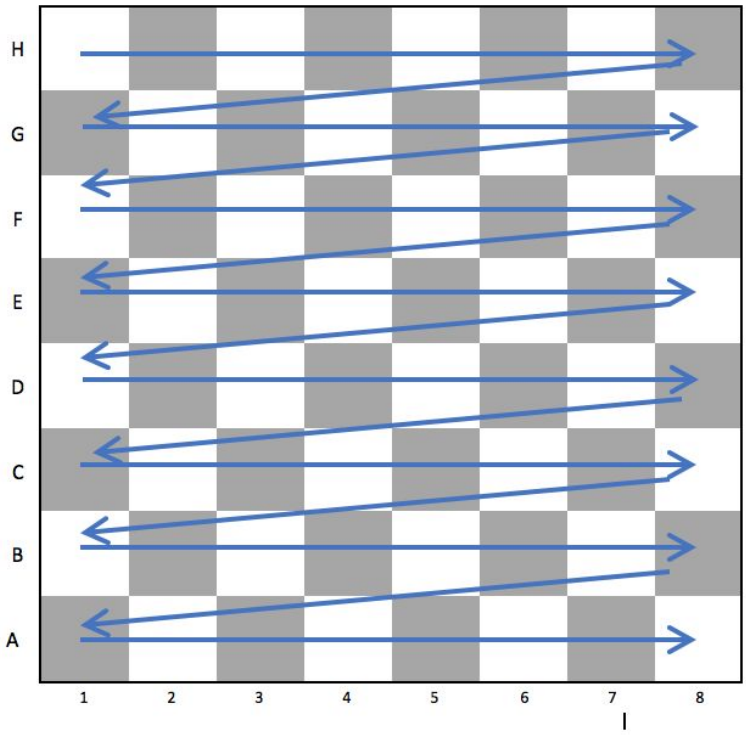
where p denotes the player (Star or Circle) and p' denotes its opponent; α_i^p denotes the number of pieces of the player p on square i; and W_i^p denotes the weight of square i for player p.

Expansion order and Tie breaking

When it is a player's turn (either Star player or Circle player) to play and it has more than one valid move, you **must** follow this rule for expanding the moves:
First, sort the moves based on the initial position according to the figure below. If two moves have the same initial position, sort them based on the final position according to the figure below. For example, if the valid moves are "C3-D4", "C3-D2", "E5-F4", you should expand them in this order: "E5-F4", "C3-D2", "C3-D4".

Ties between the legal moves are also broken by handling the moves in positional order shown in the figure below (that is, first favor squares in upper rows, and in the same row favor squares in the left side). For example, if a player has the three possible moves

“C3-D4”, “C3-D2”, and “E5-F4”, and all of them will end up in the same utility for this player, it should choose “E5-F4”.



File Formats

Input format:

<PLAYER> which will be either “Star” or “Circle”.
<ALGORITHM> which will either be “MINIMAX” or “ALPHABETA” and determines the algorithm that you must use to come up with next move
<DEPTH LIMIT> which is a number up to 10 and determines the maximum depth of your minimax or alpha/beta pruning tree
<CURRENT BOARD STATE> contains 8 lines where each line includes 8 square indicators separated by a comma “,”. These 8 lines represent the rows of the board from H to A. A square indicator could be “S{i}”, “C{i}”, or “0” where {i} indicates the number of pieces (Star or Circle) at that square, and “0” indicates that no pieces are on that square. Note that only squares of row H (for Star) and row A (for Circle) can have more than one Star or Circle piece, respectively. The initial board state will contain at least one piece for each player.

Further, the initial board state may not necessarily happen as a result of playing a game from the beginning.
<ROW VALUES> contains one line including 8 increasing numbers separated by a comma “,’. This line indicates the values of rows A to H for player Star and the values of rows H to A for player Circle.

Output format:

<NEXT MOVE> the next move played by the player <PLAYER>. Remember that the pass move should be denoted by “pass” and any non-pass move in this game should be denoted by *“{row letter of initial position}{ column number of initial position}-{row letter of final position}{ column number of final position}”* of the piece the player chooses to move. For example, “C5-D4” describes a move where a piece has been moved from square C5 to square D4.
<MYOPIC_UTILITY> the instant utility of the player <PLAYER> after playing <NEXT MOVE>.
<FARSIGHTED_UTILITY> the utility of the player <PLAYER> at the terminal node of the search tree (considering the maximum depth of <DEPTH LIMIT>) for the game under the assumption that both players select moves maximizing their utilities.
<NUMBER OF NODES> number of nodes visited during the execution of the algorithm <ALGORITHM>. Note that when you implement either “MINIMAX” or “ALPHABETA” algorithm, it takes a state as an input. Here, you need to count the number of states (or nodes of the tree) your algorithm visits during the execution.

- Note: there should not be any empty new line after <NUMBER OF NODES> line in the “output.txt” file generated by your code.

Grading

Your homework submission will be tested as follows: Your program should take no command-line arguments. It should read a text file called “input.txt” in the current directory that contains a problem definition. It should write a file “output.txt” with your solution. The format for files “input.txt” and “output.txt” is specified below. End-of-line convention is Unix (because Vocareum is a Unix system).

During grading, 50 test cases will be used. If your homework submission passes all 50 test cases, you receive 100 points. For each test case that fails, you will lose 2 points.

Note that if your code does not compile, or somehow fails to load and parse input.txt, or writes an incorrectly formatted output.txt, or no output.txt at all, or OuTpUt.TxT, you will get zero points.

Sample Test Cases

You are provided with sample test cases and outputs (see below). The goal of the samples is to help you check if you have correctly parsed the input and generated a correctly formatted output. You should not assume that if your program works on the sample test cases it will work on all test cases.

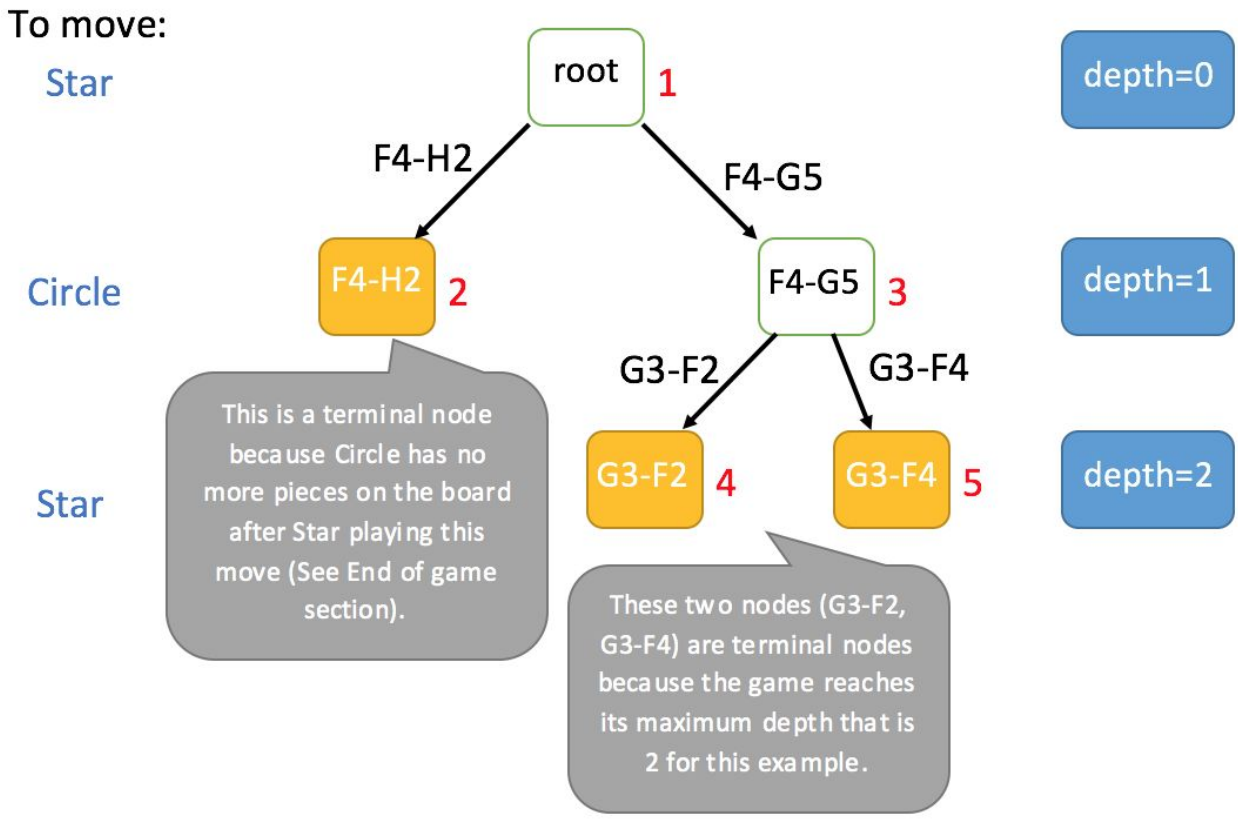
Example 1:
Input:

```
Star
MINIMAX
2
0,S1,0,0,0,0,0,0
0,0,C1,0,0,0,0,0
0,0,0,S1,0,0,0,0
0,0,0,0,0,0,0,0
0,0,0,0,0,0,0,0
0,0,0,0,0,0,0,0
0,0,0,0,0,0,0,0
0,0,0,0,0,0,0,0
10,20,30,40,50,60,70,80
```

Output:

F4-H2
160
160
5

- Clarification (search tree): In the search tree below, the initial state of the board is denoted by a node named “root”. The name of a move is denoted beside the arrows, and the name of the node resulting from a move is denoted by that move.



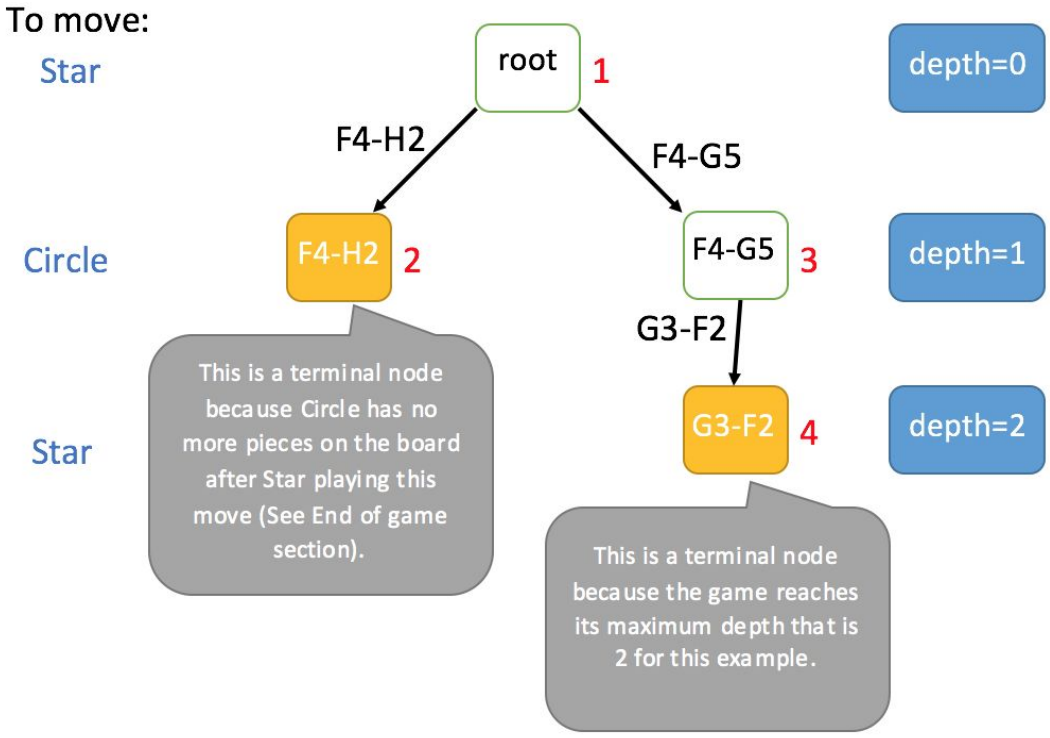
Example 2:
Input:

```
Star
ALPHABETA
2
0,S1,0,0,0,0,0,0
0,0,C1,0,0,0,0,0
0,0,0,S1,0,0,0,0
0,0,0,0,0,0,0,0
0,0,0,0,0,0,0,0
0,0,0,0,0,0,0,0
0,0,0,0,0,0,0,0
0,0,0,0,0,0,0,0
10,20,30,40,50,60,70,80
```

Output:

```
F4-H2
160
160
4
```

- Clarification (search tree):



Note that Examples 1 and 2 are the same except for their search algorithms which is MINIMAX for the former and ALPHABETA for the latter. Compare these two examples and observe how alpha-beta pruning eliminated one node from the search tree compared to the search tree of Example 1.

Example 3:

Input:

```
Star
MINIMAX
9
0,S1,0,0,0,0,0,0
S1,0,0,0,0,0,0,0
0,0,0,0,0,0,0,C1
0,0,0,0,0,0,0,0
0,0,0,0,0,0,0,0
0,0,0,0,0,0,0,0
0,0,0,0,0,0,0,0
0,0,0,0,0,0,0,0
10,20,30,40,50,60,70,80
```

Output:

```
G1-H2
130
90
26
```

Example 4:
Input:

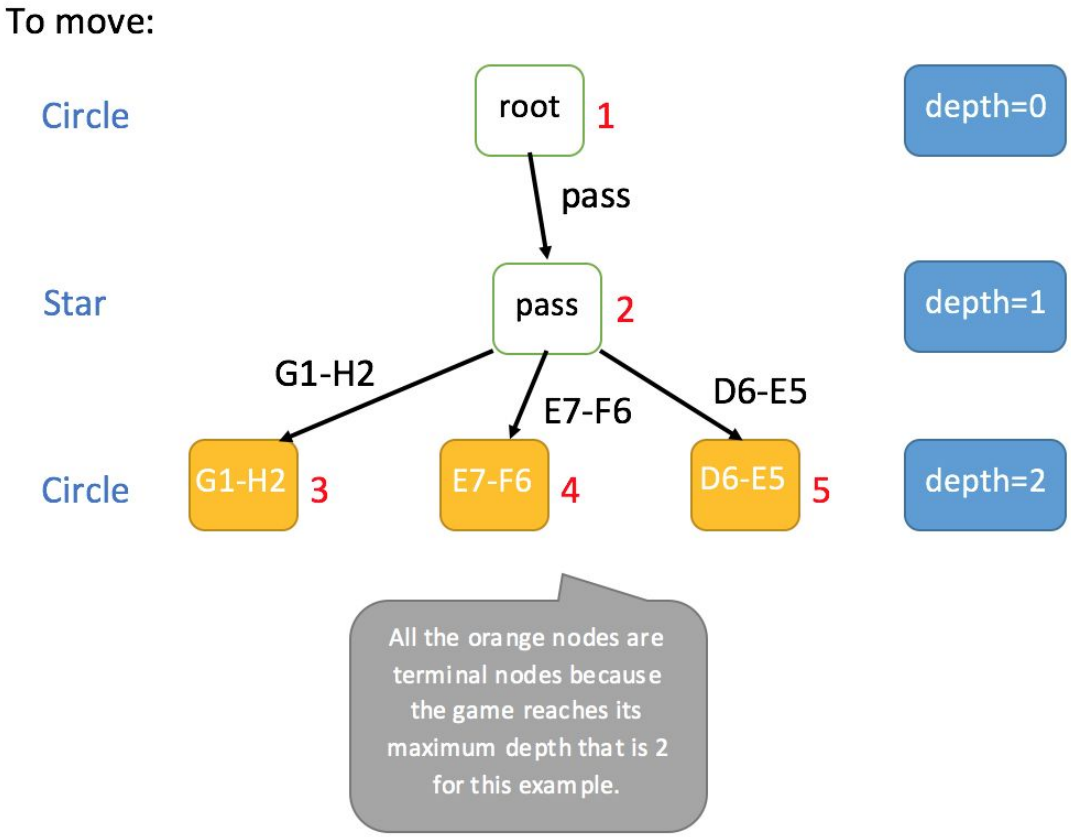
```
Circle
ALPHABETA
2
0,S2,0,0,0,0,0,0
S1,0,0,0,0,0,0,0
0,0,0,0,0,0,0,C1
0,0,0,0,0,0,S1,0
0,0,0,0,0,S1,0,0
0,0,0,0,0,0,0,0
0,0,0,0,0,0,0,0
0,0,0,0,0,0,0,0
10,20,30,40,50,60,70,80
```

Output:

```
pass
-290
-300
5
```

Note that in this case, Circle has no moves available and it has to play "pass". Even if it had ***only one*** *valid move, it had to play that one. Hence, in these cases, doing the search seems unnecessary, but you still need to do the search to calculate <FARSIGHTED_UTILITY>, that is, the utility of this player at the terminal node of the search tree.*

- Clarification (search tree):



Example 5:
Input:

```
Star
ALPHABETA
7
0,C1,0,C1,0,C1,0,C1
C1,0,C1,0,C1,0,C1,0
0,S1,0,S1,0,S1,0,S1
S1,0,S1,0,S1,0,S1,0
0,0,0,0,0,0,0,0
0,0,0,0,0,0,0,0
0,0,0,0,0,0,0,0
0,0,0,0,0,0,0,0
10,20,30,40,52,70,90,1000
```

Output:

```
pass
368
368
3
```

- Clarification (search tree)