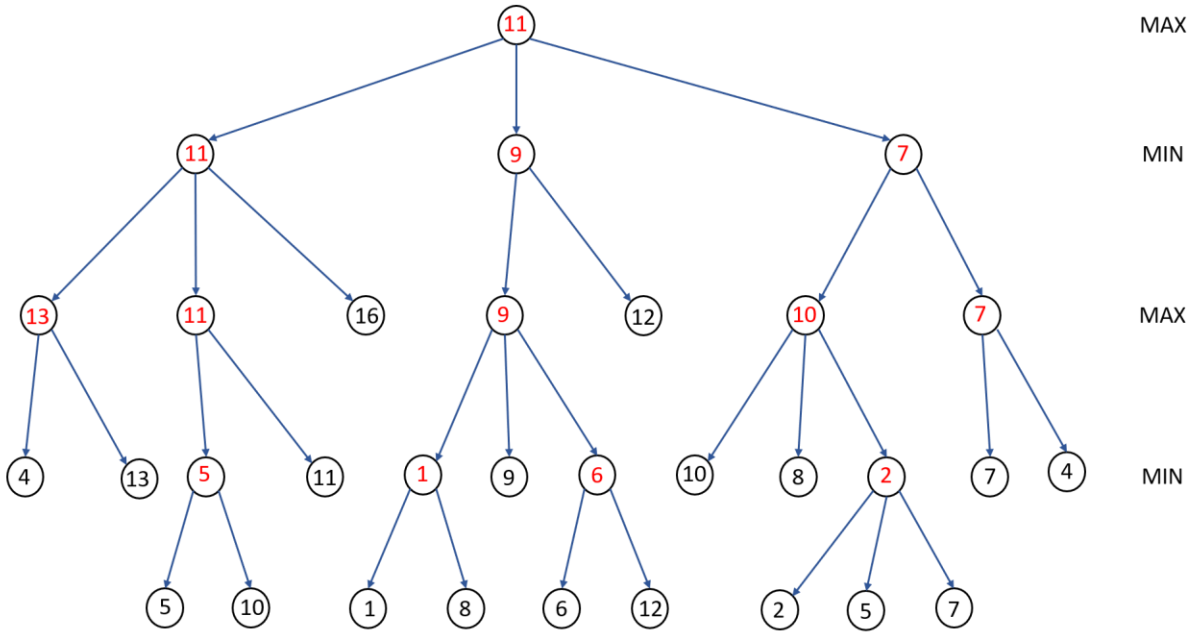
Problem 1: [15] Minimax and Alpha-Beta

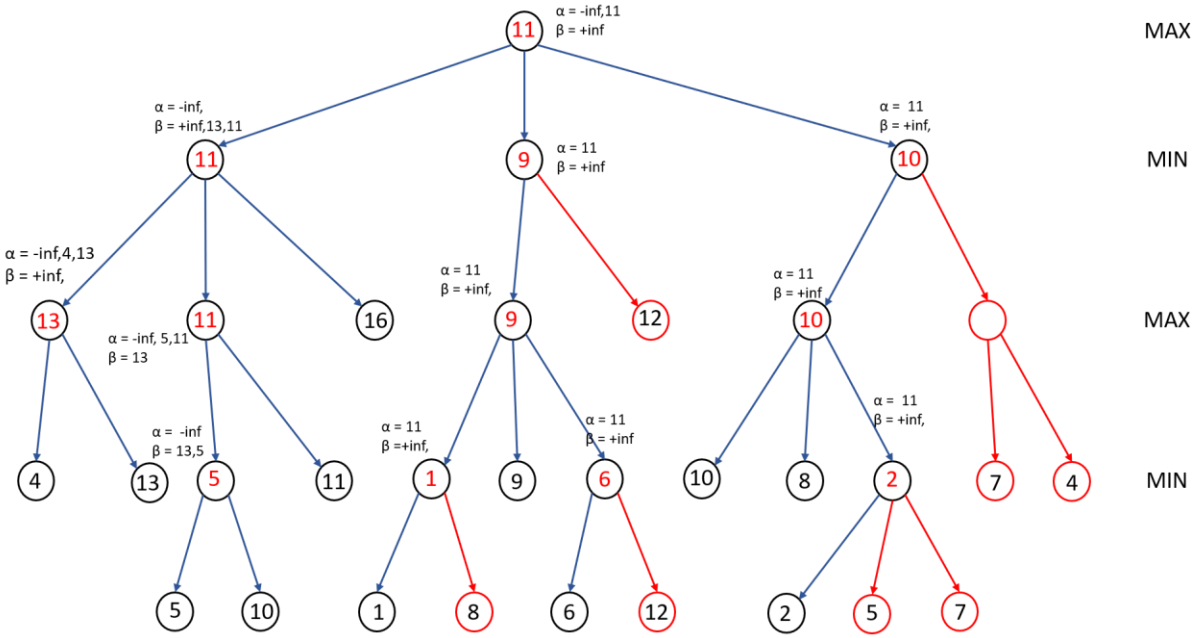
(a) [4] Use the Minimax algorithm to compute the minimax value at each node for the game tree below.

[Ans]:



[1] for having the correct value at the root node
[-0.25] for each incorrect value at the inner nodes (deduct a maximum of 3 points)

(b) [9] Use the Alpha-Beta pruning algorithm to prune the game tree in Problem 1 (a) assuming child nodes are visited from left to right. Show all final alpha and beta values computed at root, each internal node explored, and at the top of pruned branches. Note: Follow the algorithm in Figure 5.7 in the textbook [edition 3]. Also show the pruned branches.



[2] for having the correct alpha and beta values at the root node

- **[1]** for the alpha value
- **[1]** for the beta value

[5] for having the correct alpha and beta values at the inner nodes right before the pruned nodes (5 of them in total)

- **[-1]** for each incorrectly marked pruned node
- **[0.5]** for each of the alpha values
- **[0.5]** for each of the beta values

[2] for having the correct alpha and beta values at the other inner nodes [**deduct maximum of 2 points**]

- **[0.25]** for each of the alpha values
- **[0.25]** for each of the beta values

(c) [2] For a general game tree (i.e. not limited to the above tree), are there any cases that the Alpha-Beta algorithm gives a *different value at the root node* than the Minimax algorithm? If yes, show an example.

[Ans] :

[2] *No*

Problem 2: [20] Hill-Climbing

Given a set of locations and distances between them, the goal of the Traveling Salesperson Problem (TSP) is to find a shortest tour that visits each location exactly once. Assume that you do not return to the start location after visiting the last location in a tour. We would like to solve the TSP problem using a greedy hill-climbing algorithm. Each state corresponds to a permutation of all the locations (called a *tour*). The operator *neighbors*(*s*) generates all neighboring states of state *s* by swapping two locations. For example, if *s* = <*A-B-C*> is a tour, then <*B-A-C*>, <*C-B-A*> and <*A-C-B*> are the three neighbors generated by *neighbors*(*s*). We can set the evaluation function for a state to be the total distance of the tour where each pairwise distance is looked up from a distance matrix. For example, if *s* = <*A-B-C*> is a tour, then total distance is *d(A, B) + d(B,C).* Where *d(A, B)* is the distance between location *A* and *B* .

(a) [3] If there are *n* locations how many neighboring states does the neighbors(s) function produce?

[Ans] :
[3] ${}^{n}C_2 = n*(n-1)/2$

(b) [3] What is the total size of the search space, i.e., the total number of states in the sample space? There are n locations.

[Ans] : **[3]** points
[3] ${}^{n}P_n = n!$

(c) [14] Imagine that a student wants to hand out fliers about an upcoming programming contest. The student wants to visit the Memorial Union (*M*), Wisconsin Institute of Discovery (*W*), Computer Sciences Building (*S*), and Engineering Hall (*E*) to deliver the fliers. The goal is to find a tour as short as possible. The distance matrix between these locations is given as follows:

	M	W	E	S
M	0	1.1	1.4	0.9
W	1.1	0	0.6	0.7
E	1.4	0.6	0	0.5
S	0.9	0.7	0.5	0

The student starts applying hill-climbing algorithm from the initial state: <*W-M-E-S*>.

1) [2] Compute total distance of current state.

[Ans]
[2] $d(W,M) + d(M,E) + d(E,S) = \ 1.1 \ + 1.4 + 0.5 = 3.0$

2) [4] What are the possible neighboring states of <*W-M-E-S*> and the total distances?

[Ans] : Total **[4]** points

<M-W-E-S> : 1.1 + 0.6 + 0.5 = 2.2
<E-M-W-S> : 1.4 + 1.1 + 0.7 = 3.2
<S-M-E-W> : 0.9 + 1.4 + 0.6 = 2.9
<W-E-M-S> : 0.6 +1.4 + 0.9 = 2.9
<W-S-E-M> : 0.7 + 0.5 + 1.4 = 2.6
<W-M-S-E> : 1.1 + 0.9 + 0.5 = 2.5
[-0.5] for each wrong state. **[-0.25]** for each wrong distance. Deduct maximum of 4 points

3) [2] What is the *next* state reached by hill-climbing? Or explain why there is no next state, if there isn't one?

[Ans] : Total **[2]** point
[2] *<M-W-E-S>*

4) [6] If 3) had a next state iterate further until the algorithm terminate and list out the set of states from initial state to final state. What is total distance of the final state?

[Ans] [6] points
[4] *<W-M-E-S> -> <M-W-E-S> -> <M-S-E-W>*
[2] distance = 2.0