AI Assignment 1

Sourabh Shenoy
UIN: 225009050
Date: 09/25/2016

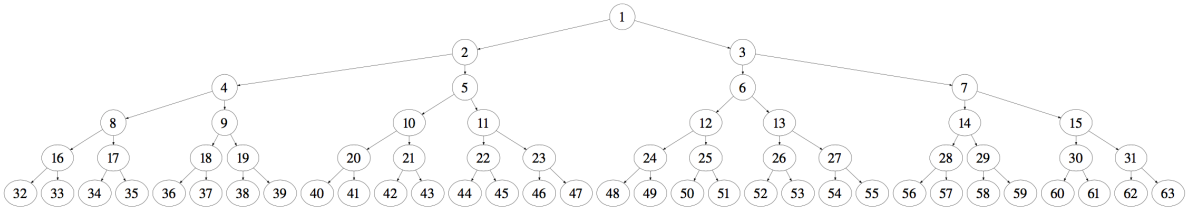


Figure 1: **Search Trees.**

Question 1 (5 pts): Give an example of when breadth-first search (BFS) and depth-first search (DFS) have the same time complexity. Pick one goal node number from the tree above where such a case happens, with depth > 2.

Solution:

BFS and DFS both work from top to bottom, left to right and hence, will have to exhaust and expand all nodes to reach the last node, i.e; **63** in this case. Hence, for node 63, both BFS and DFS have the same time complexity.

Question 2 (5 pts): Give an example when depth first search is suboptimal. Pick two node numbers as goal nodes as an example.

Solution:

Let us consider the goal nodes 3 and 33. DFS, traverses from top to down, left to right. Hence, DFS chooses the path 1 -> 2 -> 4 -> 8 -> 16 -> 32 -> 16 -> 33. and reaches goal node 33, which is at depth 5. However, had we chosen BFS, we would have reached 1 -> 2 -> 3, which is a more optimal goal node at level 1. Thus, we can see that at times, DFS is suboptimal.

Question 3 (5 pts): What limitation in BFS does iterative deepening search overcome?

Solution: Iterative deepening search overcomes the problem of BFS as it has **better space complexity**. In BFS, we have to hold all the nodes in memory at the same time, which leads to an exponential space complexity of O(b^d), where b is branching factor and d is the depth. However, for IDS,

we only need O(d) space complexity, since it just needs to store the d nodes in the current path where the search is performed.

IDS also overcomes the problem of DFS where it does not reach an infinite depth, by iteratively increasing the depth at which it searches.

Question 4 (5 pts): Why is the space complexity of BFS $O(b^{d+1})$, not $O(b^{d})$, where b is the branching factor and d is the goal depth?

Solution:

In the worst case scenario, suppose the goal node is located at level d, then we have to store nodes at all levels, including the root. The number of nodes at level 0 will be 1 (root). Nodes at level 1 will be b, level 2 will be b^2 and so on, till b^d, which expands the last level to locate the node.

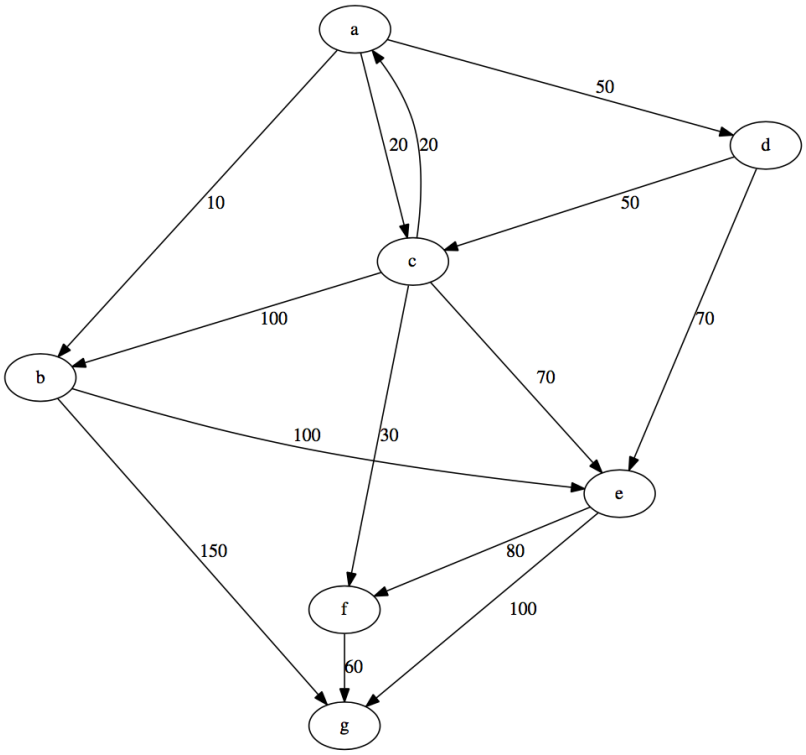
We know that $1+b+b^2+\ldots.+b^d=b^{d+1}-2 = O(b^{(d+1)})$.

Question 5 (5 pts): Can depth limited search become incomplete in the case of the finite search tree above? If so, give an example. If not, explain why not.

Solution:

Yes, Depth limited search **can become incomplete** in the case of the finite search tree given in the figure. This is because, DLS limits the search based on the depth limit. Suppose our depth limit is 4 and the goal node is 40, then DLS will not discover the goal node, since it will assume that the nodes 16-31 are the leaf nodes and will not traverse further down those paths. To avoid such a situation, we should set the depth limit as 5 (Assuming that root is at level 0).

Question 6 (10 pts): For the problem shown in Fig. 2, show that the heuristic is admissible ($h(n) \leq h^*(n)$ for all n). Note: You have to compute $h^*(n)$ for each n and compare to the h(n) table.



Node	h(n)
a	78
b	71
c	72
d	57
e	28
f	51
g	0

Figure 2: **Informed Search.**

Solution:

Node	Paths with Cost	h(node)	h*(node)
G	G : 0	0	0
F	F -> G : 60	51	60
E	E -> G : 100 E -> F -> G : 140	28	100
B	B -> G : 150 B -> E —-> G : 200	71	150
C	C -> F -> G : 90 C -> B —-> G : 250 C -> E —-> G : 170 C -> A -> D -> E -—> G : 240 C -> A -> B —-> G : 180	72	90
D	D -> C —-> G : 140 D -> E —-> G : 170	57	140

Node	Paths with Cost	h(node)	h*(node)
A	A -> C —-> G : 110 A -> D —-> G : 190 A -> B —-> G : 160	78	110

Thus, we see that for each node, h(n) < h*(n). Hence, the **heuristic chosen is admissible.**

Question 7 (15 pts): Manually conduct greedy best-first search on the graph below (Fig. 2), with initial node a and goal node g. Actual cost from node to node are shown as edge labels. The heuristic function value for each node is shown in a separate table to the right. Show:

1. Node list content at each step
2. Node visit order
3. Solution path
4. Cost of the final solution.

Solution:

For greedy best first search, we will maintain 2 lists. Ones which are currently discovered, which will be sorted according to the heuristic value in the ascending order(L) and another list of selected nodes (V).

Every time, we visit the children of the first node, with the least heuristic value and add the node to the selected list. Since we need to go from A to G, the lists will be:

Step 1:

L= {<A,78>}, V= {}

Step 2:

L= {<D,57>,<B,71>,<C,72>} , V ={A}

Step 3:

L={<E,28>,<B,71>,<C,72>}, V={A,D}

Step 4:

L={<G,0>,<F,51>,<B,71>,<C,72>}, V={A,D,E}

Since the first element of list L, ‘G’, is our goal node, we stop the algorithm. The path chosen by the greedy best first search algorithm is **A -> D -> E -> G**, and the cost of the path is **50 + 70 + 100 = 220**, which is not the optimal cost.

Question 8 (15 pts): (1) Repeat the problem right above with A* search. (2) In addition, show the f(n) value for all nodes expanded (you need this to sort them in the node list). (3)Which one gives a shorter solution: Greedy best-first or A*? Note: Note that the same node can appear in the node list with a different f(n) value, depending on the path taken. For example, f(e) will be different if you followed a different path to reach the node: a → b → e vs. a → c → b → e. Due to this, you may need to track which path you followed to reach node n and calculate the f(n) value accordingly. It helps to write the node f as $_{ac}$f to indicate the path in the subscript (a → c → f).

Solution:

The elements are represented as a triplet, like <node, cost, chosen path to node>

Step 1:

<A, 78, - >

Possible paths from first node A, are B, C and D

f(b) = 10 + 71 = 81
f(c) = 20 + 72 = 92
f(d) = 50 + 57 = 107

Step 2:

<B, 81, A>	<C, 92, A>	<D, 107, A>

Possible paths from first node B, are E and G

f(e) = 10 + 100 + 28 = 138

f(g) = 10 + 150 + 0 = 160

Step 3:

<C, 92, A>	<D, 107, A>	<E,138,AB>	<G,160,AB>

Possible paths from first node C, are A, B, F and E

f(a) = 20 + 20 + 78 = 118
f(b) = 20 + 100 + 71 = 191
f(f) = 20 + 30 + 51 = 101
f(e) = 20 + 70 + 28 = 118

Step 4:

<F, 101, AC>	<D, 107, A>	<A, 118, AC>	<E,118,AC>	<E,138,AB>	<G,160,AB>	<B,191,AC>

Possible path from first node F is G

f(g) = 20 + 30 + 60 + 0 = 110

Step 5:

<D, 107, A>	<G, 110, ACF>	<A,118,AC>	<E,118,AC>	<E,138,AB>	<G,160,AB>	<B,191,AC>

Possible paths from first node D, are C and E

f(c) = 50 + 50 + 72 = 172
f(e) = 50 + 70 + 28 = 148

Step 5:

<G, 110, ACF>	<A,118,AC>	<E,118,AC>	<E,138,AB>	<E,148,AD>	<G,160,AB>	<C,172,AD>	<B,191,AC>

Since the first node in the list is the goal node, we stop the algorithm. The path chosen is **A -> C -> F -> G** and the cost of the path is **110**.

As we can see, **A* search gives a more optimal path than greedy best first search algorithm.**

Question 9 (5 pts): Using the following figure 3, use minmax search to assign utility values for each internal node (i.e., non-leaf node) and indicate which path is the optimal solution for the MAX node at the root of the tree.

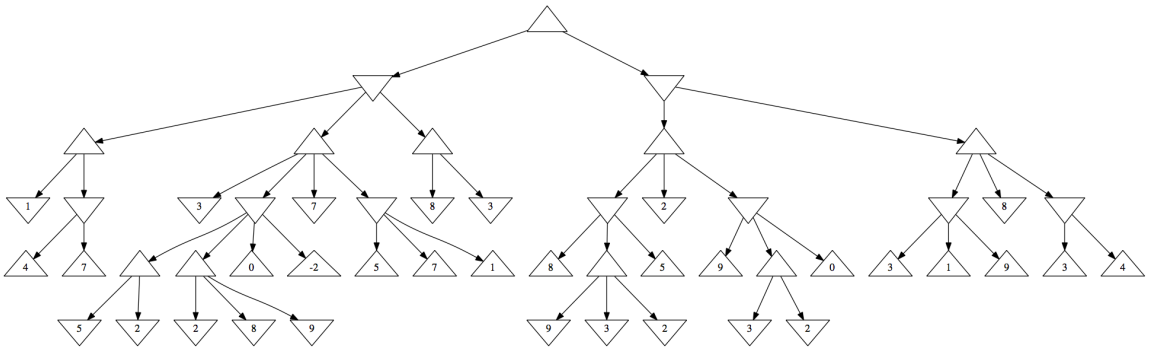


Figure 3: **Game Tree.** Solve using minmax search.

Assume you explore the successors from left to right.

Solution:

The assigned values are in color red and the optimal path is highlighted in blue in the figure below.

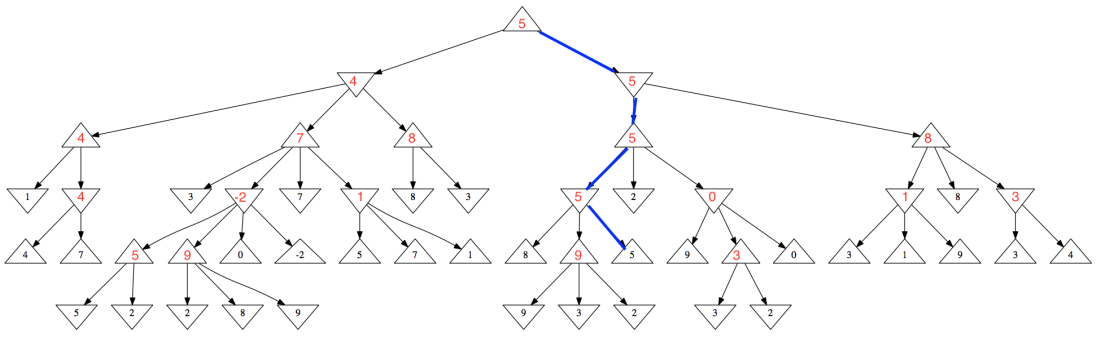


Figure 4: **Game Tree.** Solve using $\alpha - \beta$ pruning. This tree is the same as figure 3.

Question 10 (20 pts): Using the following figure 4, use α − β pruning to (1) assign utility values for each internal node (i.e., non-leaf node) and indicate which path is the optimal solution for the MAX node at the root of the tree. (2) For each node, indicate the final α and β values. (Note that initial values

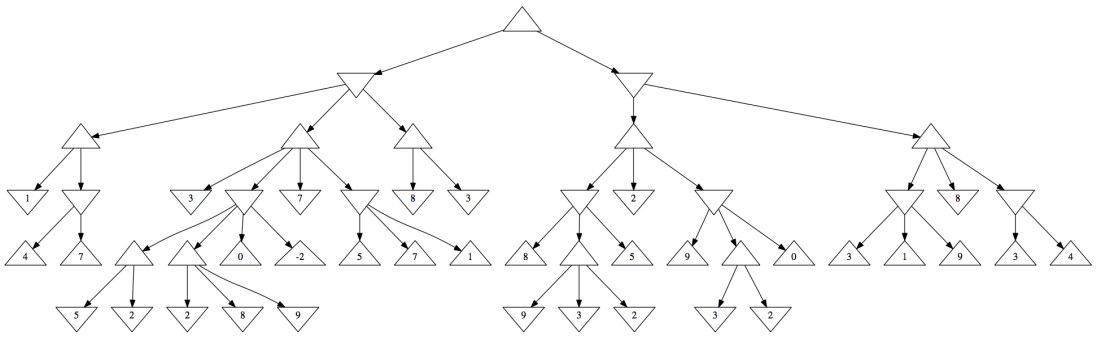


Figure 4: **Game Tree.** Solve using $\alpha - \beta$ pruning. This tree is the same as figure 3.

at the root are α = −∞, β = ∞.) (3) For each cut that happens, draw a line to cross out that subtree.

Solution:
The assigned values are in color red and the optimal path is highlighted in blue in the figure below. Also, the alpha and beta values are mentioned in the brackets

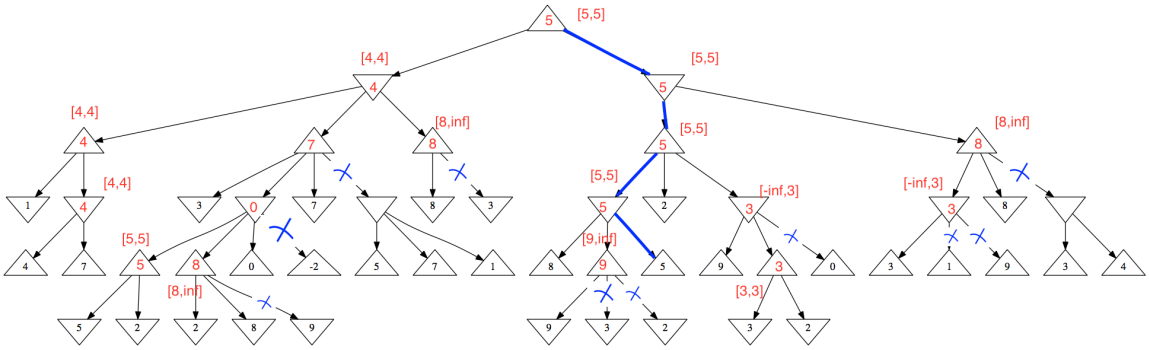


Figure 3: **Game Tree.** Solve using minmax search.

Question 11 (10 pts): In Minmax search, we used a depth-first exploration through the use of recursion. We know that Minmax gives an optimal solution, however, we also know that depth-first search is suboptimal.

Explain why Minmax gives an optimal solution even when it is using a depth-first exploration.

Solution:

In a minimal search, we consider the maximum value for our turn/level and minimum value for the opponent's turn/level. The root is the current state and the children represent all the possible moves that could be made. The leaf nodes are the final states of the game. Thus, we use a depth first search approach to explore the entire game tree. We check for all possible paths, in a brute force manner. However, in the case of DFS, we return whenever a goal node is found and do not traverse any further. This may lead to cases where optimal path is not found.