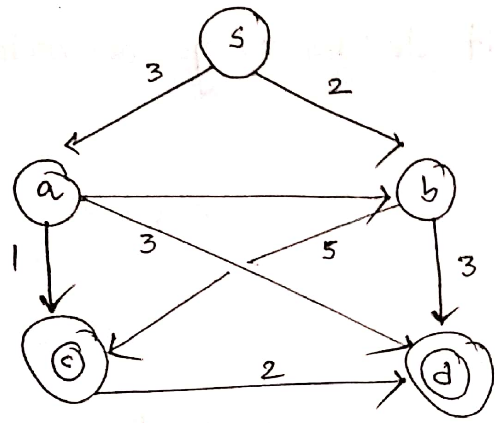
AI Th DA-1

Fall 2021

by Sharadindu Adhikari | 19BCE2105

Q1.

Using A* algorithm, find the optimal path from the node S for the graph given in the diagram. Two goal nodes are given in the diagram. Explain how the algorithm can/can not find the optimal solution for both goal nodes.



h(s)	h(a)	h(b)	h(c)	h(d)
1	3	3	0	0

Soln:

A* algorithm finds optimal path using $f(n) = g(n) + h(n)$ function, where

n is the last node

$g(n)$ is the cost from start to node n.

$h(n)$ is heuristic value of node n.

Step-1 :

We start with node 'S'. Node 'a' and 'b' can be reached from node 'S'.

A* algorithm calculates f(a) and f(b) as:

$f(a) = 3+3 = 6$

$f(b) = 2+3 = 5$

$\because f(b) < f(a)$, so it decides to go to node b.

$\boxed{\text{Path: } S \longrightarrow b}$

Step-2 :

Node 'c' and node 'd' can be reached from node 'b'.

A* algorithm calculates f(c) and f(d) as:

$f(c) = (2+5)+0 = 7$

$f(d) = (2+3)+0 = 5$

$\because f(d) < f(a)$ and $f(d) < f(c)$, so it decides to go to node 'd'.

$\boxed{\text{Path: } S \rightarrow b \rightarrow d}$

So, it's cost is 5 that is less among all; it is optimal path.

So, optimal solution for node 'd' is $S \rightarrow b \rightarrow d$ with cost of 5.

step-3 :

There are no nodes that can be reached from ~~node 'D'~~ node 'd'.

So from node 'a' and node 'c', $f(a) < f(c)$.

So it decides to go to node 'a'.

path: $S \rightarrow a$

step-4 :

Node 'b', 'c' and 'd' can be reached from node 'a'.

A* algorithm calculates f(b), f(c) and f(d) as:

$$f(b) = (3+3)+3 = 9$$
$$f(c) = (3+1)+0 = 4$$
$$f(d) = (3+3)+0 = 6$$

$\because f(c) < f(b)$ and $f(c) < f(d)$, so it decides to go to node c.

path: $S \rightarrow a \rightarrow c$

Step- 5 :

As 4 is the minimum cost among others, there aren't any other optimal path possible.

$\boxed{\text{path: } S \rightarrow a \rightarrow c}$

So, it's cost is 4, that is less among all; it is optimal path.

So, Optimal solution for node 'c' is $S \rightarrow a \rightarrow c$ with the cost of 4.

Final answer :

∴ Optimal solution for node 'c' is $S \rightarrow a \rightarrow c$ with cost of 4;

Optimal solution for node 'd' is $S \rightarrow b \rightarrow d$ with cost of 5.

Q2.

If a program prunes away a move in chess that looks bad because it sacrifices valuable material that may just be the sacrificial move that would have checkmated the opponent in another few additional ply. How can alpha-beta pruning be made safe for a chess program? Discuss whether this pruning algorithm can be used to achieve the best solution space for the given problem.

Soln:

We use alpha-beta pruning with a minimax algorithm for evaluating a large ~~numbe~~ game tree in AI. Generally, when the game tree is very large, i.e., the possible positions or states are of huge numbers, for a computing system, it gets unrealistic and impractical to search the whole game tree for the required move in a game like Chess. Hence it searches in a small portion of the game tree making fixed-depth or level.

The tendency of fixed-depth minimax searches to badly underestimate or overestimate positional scores in dynamic situations known as the horizon effect. The problem is due to static evaluation being used on positions unsuited to such evaluation. Static evaluation scores a position based on things like material, pawn structure, and king mobility, properties that can be determined without extending the search tree. But some positions are transitional, needing more search to identify their nature.

The standard way to deal with this problem is to "see beyond the horizon" and recursively apply a quiescence function to all terminal nodes that extends the search an extra ply if there are checks, forced captures or promotions available at that node (and finding the best possible move). If no such moves are available, then the position is considered quiet and the usual static evaluation function is called.