National University of Computer & Emerging Sciences

Lab Manual
CS461: Artificial Intelligence Lab

Course Instructor	Dr. Hafeez-Ur-Rehman
Lab Instructor	Muhammad Hamza
Semester	Spring 2022

Uninformed Search Algorithms

- Uniform-cost Search
- Bidirectional Search

Uniform Cost Search Algorithm

Uniform-cost search is a searching algorithm used for traversing a weighted tree or graph. This algorithm comes into play when a different cost is available for each edge. The primary goal of the uniform-cost search is to find a path to the goal node which has the lowest cumulative cost. Uniform- cost search expands nodes according to their path costs form the root node. It can be used to solve any graph/tree where the optimal cost is in demand. A uniform-cost search algorithm is implemented by the priority queue. It gives maximum priority to the lowest cumulative cost. Uniform cost search is equivalent to BFS algorithm if the path cost of all edges is the same.

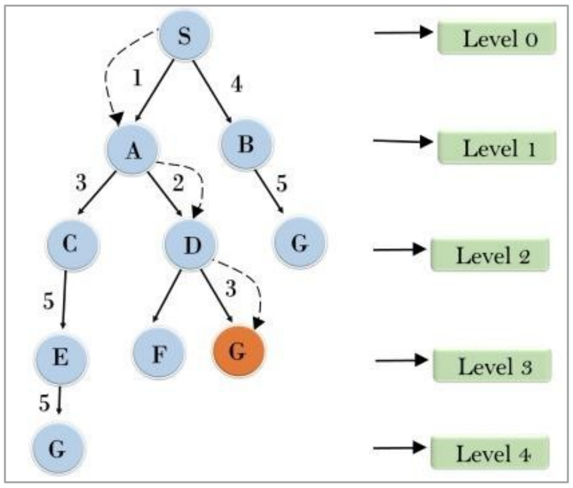
Advantages:

- Uniform cost search is optimal because at every state the path with the least cost is chosen.

Disadvantages:

- It does not care about the number of steps involve in searching and only concerned about path cost. Due to which this algorithm may be stuck in an infinite loop.

Example:



Completeness: Uniform-cost search is complete, such as if there is a solution, Uniform Cost Search will find it.

Time Complexity:

If the branching factor is b, every time you expand out a node, you will encounter **k** more nodes. Therefore, there are

1 node at level **0**,

b nodes at level **1**,

b_2 nodes at level **2**,

b_3 nodes at level **3**,

...

b_k nodes at level **k**.

So let's suppose that the search stops after you reach level **k**. When this happens, the total number of nodes you'll have visited will be

$$1 + b + b_2 + \ldots + b_k = (b^{k+1} - 1) / (b - 1)$$

That equality follows from the sum of a geometric series. It happens to be the case that $b^{k+1} / (b - 1) = O(b^k)$, so if your goal node is in layer **k**, then you have to expand out $O(b^k)$ total nodes to find the one you want.

If **C** is your destination cost and each step gets you ε closer to the goal, the number of steps you need to take is given by $C / \varepsilon + 1$. The reason for the **+1** is that you start at distance 0 and end at C / ε, so you take steps at distances

And there are $1 + C / \varepsilon$ total steps here. Therefore, there are $1 + C / \varepsilon$ layers, and so the total number of states you need to expand is $O(b^{(1 + C / \varepsilon)})$.

Space Complexity: The same logic is for space complexity so, the worst-case space complexity of Uniform-cost search is $O(b^{(1 + C / \varepsilon)})$.

Optimal: Uniform-cost search is always optimal as it only selects a path with the lowest path cost.

Bidirectional Search Algorithm

Bidirectional search algorithm runs two simultaneous searches, one form initial state called as forward- search and other from goal node called as backward-search, to find the goal node. Bidirectional search replaces one single search graph with two small subgraphs in which one starts the search from an initial vertex and other starts from goal vertex. The search stops when these two graphs intersect each other.

Bidirectional search can use search techniques such as BFS, DFS, DLS, etc.

Advantages:

- Bidirectional search is fast.
- Bidirectional search requires less memory

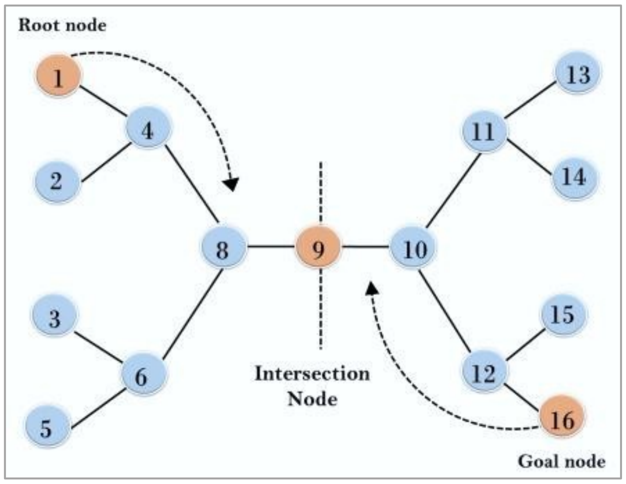
Disadvantages:

- Implementation of the bidirectional search tree is difficult.
- In bidirectional search, one should know the goal state in advance.

Example:

In the below search tree, bidirectional search algorithm is applied. This algorithm divides one graph/tree into two sub-graphs. It starts traversing from node 1 in the forward direction and starts from goal node 16 in the backward direction.

The algorithm terminates at node 9 where two searches meet.



Completeness: Bidirectional Search is complete if we use BFS in both searches.

Time Complexity: Time complexity of bidirectional search using BFS and DFS is **$O(b^d)$**.

Space Complexity: Space complexity of bidirectional search using BFS and DFS is **$O(b^d)$**.

Optimal: Bidirectional search is Optimal.