



MIDTERM PROJECT

Introduction to Artificial Intelligence

MEMBER

PHAN ANH TUẤN - 521H0516

NGÔ TRỌNG NHÂN - 520H0393

TRƯƠNG HUỲNH ĐĂNG KHOA - 521H0503

NGUYỄN THÀNH THẮNG - 519H0348

TRẦN HOÀNG PHÚC - 521H0288

Email:

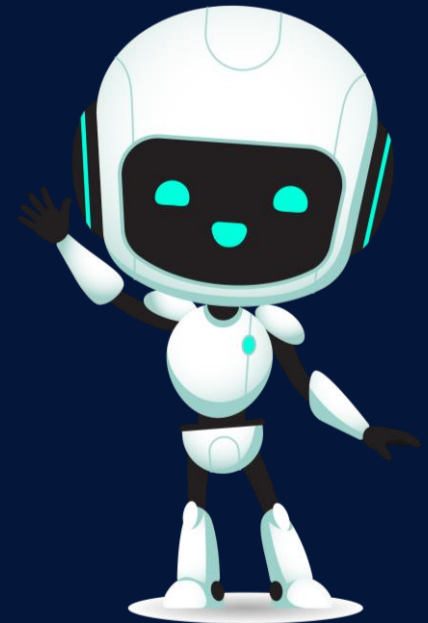
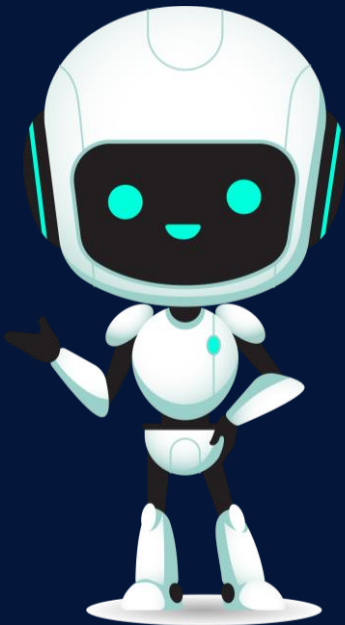
520H0393@student.tdtu.edu.vn

521H0503@student.tdtu.edu.vn

521H0288@student.tdtu.edu.vn

519H0348@student.tdtu.edu.vn

521H0516@student.tdtu.edu.vn



TASK 1

Uninformed search:

is a class of general-purpose search algorithms which operates in brute force-way. Uninformed search algorithms do not have additional information about state or search space other than how to traverse the tree, so it is also called blind search.

Following are the various types of uninformed search algorithms:

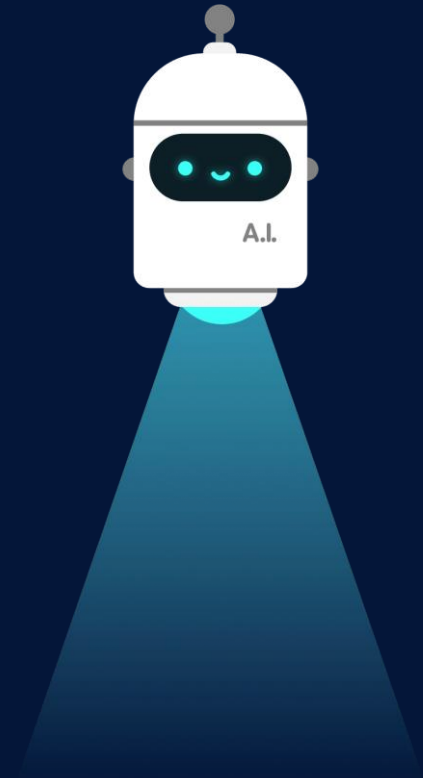
- Breadth-first Search
- Depth-first Search
- Depth-limited Search
- Iterative deepening depth-first search
- Uniform cost search
- Bidirectional Search



TASK 1

Breadth-first Search:

- Breadth-first search is the most common search strategy for traversing a tree or graph. This algorithm searches breadthwise in a tree or graph, so it is called breadth-first search.
- BFS algorithm starts searching from the root node of the tree and expands all successor node at the current level before moving to nodes of next level.
- The breadth-first search algorithm is an example of a general-graph search algorithm.
- Breadth-first search implemented using FIFO queue data structure

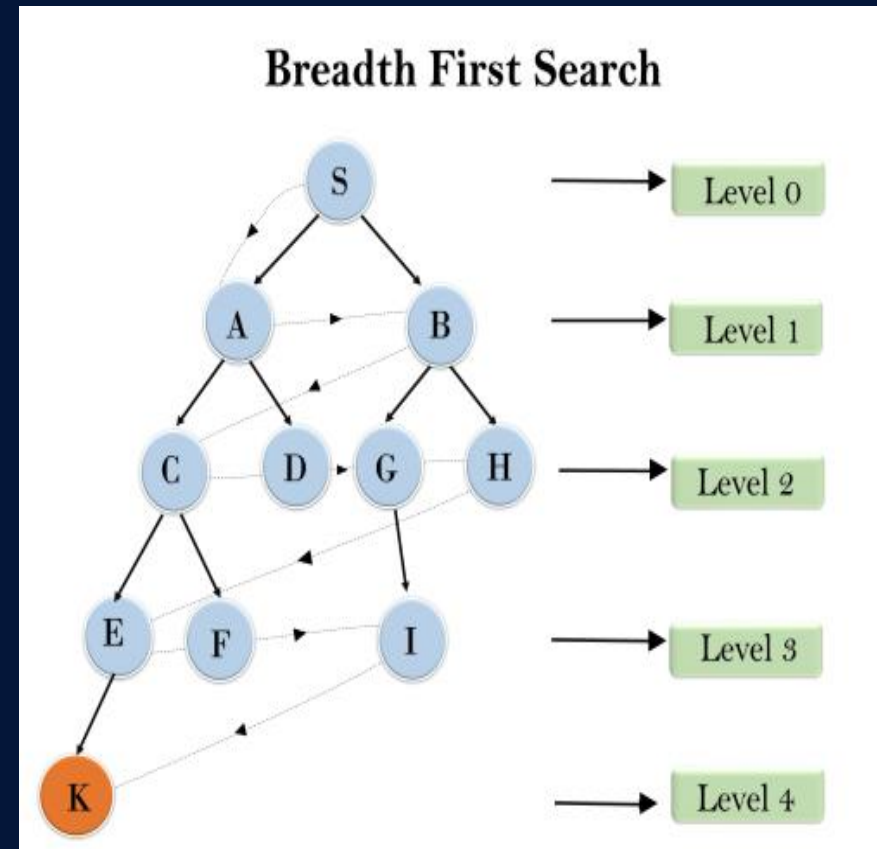


BREADTH-FIRST SEARCH:

Example:

In the tree structure, we have shown the traversing of the tree using BFS algorithm from the root node S to goal node K. BFS search algorithm traverse in layers, so it will follow the path which is shown by the dotted arrow, and the traversed path will be:

$S \rightarrow A \rightarrow B \rightarrow C \rightarrow D \rightarrow G \rightarrow H \rightarrow E \rightarrow F \rightarrow I \rightarrow K$



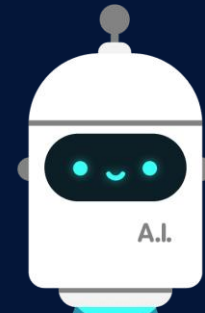
BREADTH-FIRST SEARCH:

Advantages:

- BFS will provide a solution if any solution exists.
- If there are more than one solutions for a given problem, then BFS will provide the minimal solution which requires the least number of steps.

Disadvantages:

- It requires lots of memory since each level of the tree must be saved into memory to expand the next level.
- BFS needs lots of time if the solution is far away from the root node.



TASK 1

Depth-first Search:

- Depth-first search is a recursive algorithm for traversing a tree or graph data structure.
- It is called the depth-first search because it starts from the root node and follows each path to its greatest depth node before moving to the next path.
- DFS uses a stack data structure for its implementation.
- The process of the DFS algorithm is similar to the BFS algorithm.



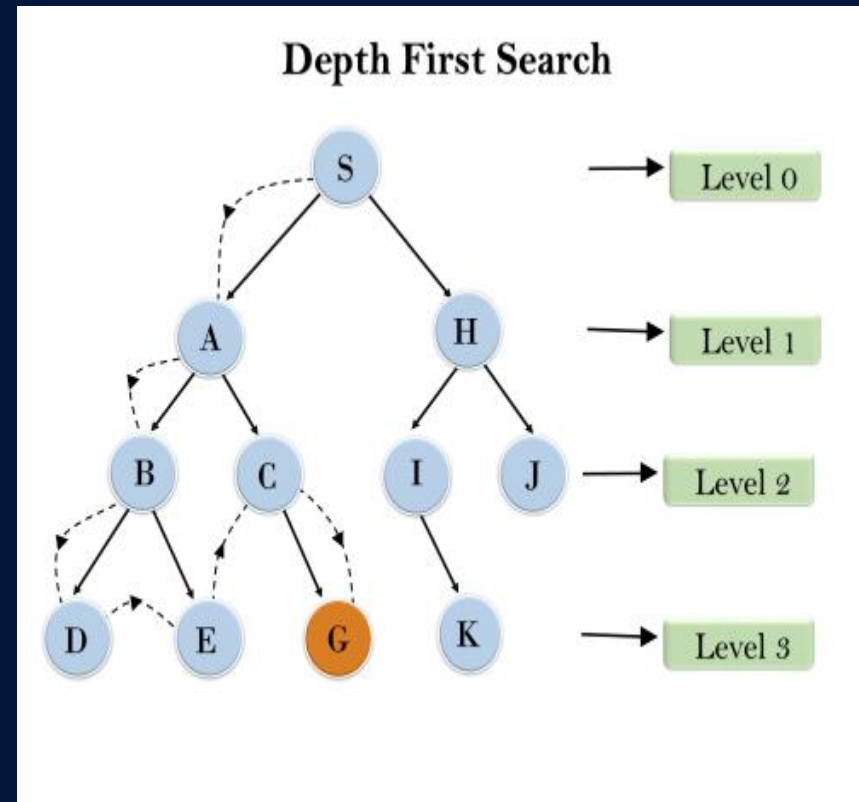
DEPTH-FIRST SEARCH:

Example:

In the below search tree, we have shown the flow of depth-first search, and it will follow the order as:

Root node--->Left node ----> right node.

It will start searching from root node S, and traverse A, then B, then D and E, after traversing E, it will backtrack the tree as E has no other successor and still goal node is not found. After backtracking it will traverse node C and then G, and here it will terminate as it found goal node.



DEPTH-FIRST SEARCH:

Advantage:

- DFS requires very less memory as it only needs to store a stack of the nodes on the path from root node to the current node.
- It takes less time to reach to the goal node than BFS algorithm (if it traverses in the right path).

Disadvantage:

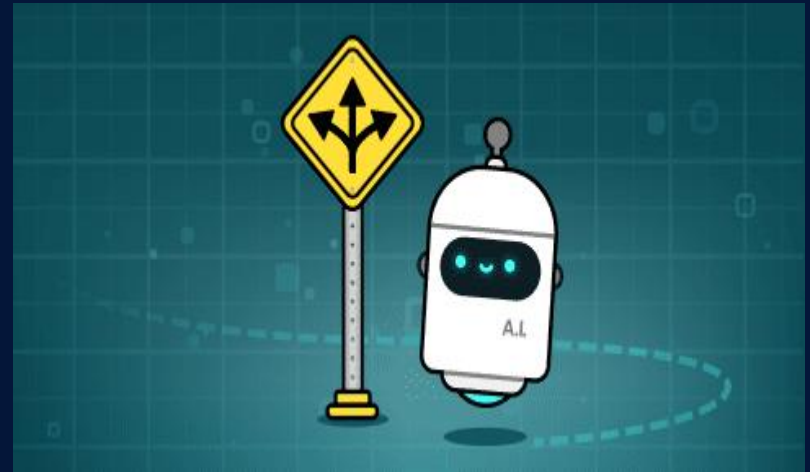
- There is the possibility that many states keep re-occurring, and there is no guarantee of finding the solution.
- DFS algorithm goes for deep down searching and sometime it may go to the infinite loop.



TASK 1

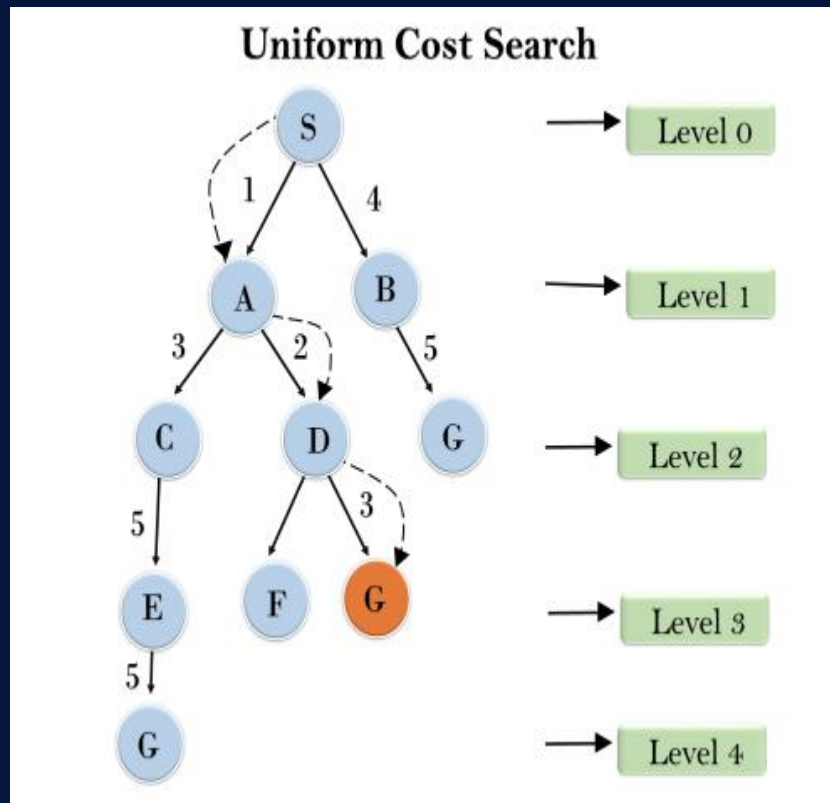
Uniform-cost Search Algorithm:

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 from 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..



UNIFORM-COST SEARCH ALGORITHM:

Example:



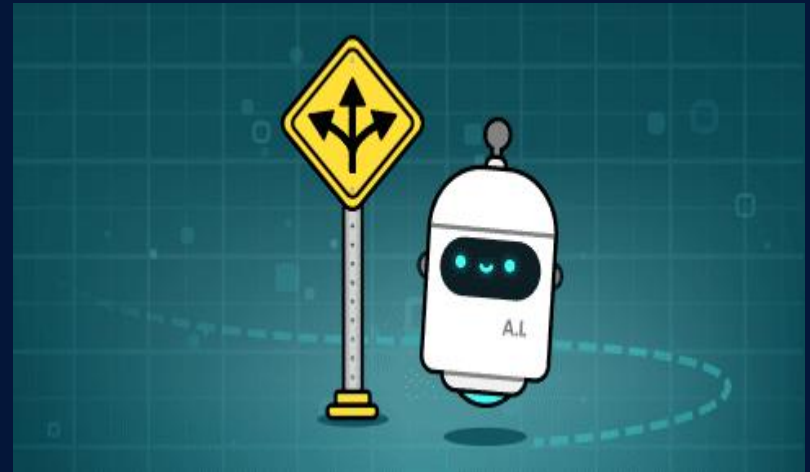
UNIFORM-COST SEARCH ALGORITHM:

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 involved in searching and only concerned about path cost. Due to which this algorithm may be stuck in an infinite loop.



TASK 2

Informed Search Algorithms:

So far we have talked about the uninformed search algorithms which looked through search space for all possible solutions of the problem without having any additional knowledge about search space. But informed search algorithm contains an array of knowledge such as how far we are from the goal, path cost, how to reach to goal node, etc. This knowledge help agents to explore less to the search space and find more efficiently the goal node.

The informed search algorithm is more useful for large search space. Informed search algorithm uses the idea of heuristic, so it is also called **Heuristic search**.



TASK 2

Heuristics function:

Heuristic is a function which is used in Informed Search, and it finds the most promising path. It takes the current state of the agent as its input and produces the estimation of how close agent is from the goal. The heuristic method, however, might not always give the best solution, but it guaranteed to find a good solution in reasonable time. Heuristic function estimates how close a state is to the goal.

It is represented by $h(n)$, and it calculates the cost of an optimal path between the pair of states. The value of the heuristic function is always positive.

Admissibility of the heuristic function is given as:

$$h(n) \leq h^*(n)$$

Here $h(n)$ is heuristic cost, and $h^*(n)$ is the estimated cost. Hence heuristic cost should be less than or equal to the estimated cost.



TASK 2

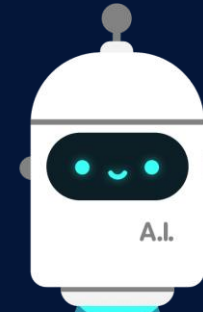
Best-first Search Algorithm (Greedy Search):

Greedy best-first search algorithm always selects the path which appears best at that moment. It is the combination of depth-first search and breadth-first search algorithms. It uses the heuristic function and search. Best-first search allows us to take the advantages of both algorithms. With the help of best-first search, at each step, we can choose the most promising node. In the best first search algorithm, we expand the node which is closest to the goal node and the closest cost is estimated by heuristic function, i.e.

$$f(n) = g(n) + h(n)$$

Where, $h(n)$ = estimated cost from node n to the goal.

The greedy best first algorithm is implemented by the priority queue.



TASK 2

Best first search algorithm:

Step 1: Place the starting node into the OPEN list.

Step 2: If the OPEN list is empty, Stop and return failure.

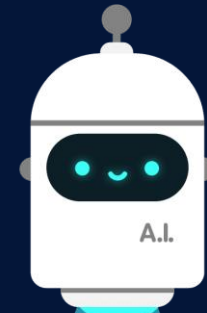
Step 3: Remove the node n , from the OPEN list which has the lowest value of $h(n)$, and places it in the CLOSED list.

Step 4: Expand the node n , and generate the successors of node n .

Step 5: Check each successor of node n , and find whether any node is a goal node or not. If any successor node is goal node, then return success and terminate the search, else proceed to Step 6.

Step 6: For each successor node, algorithm checks for evaluation function $f(n)$, and then check if the node has been in either OPEN or CLOSED list. If the node has not been in both list, then add it to the OPEN list.

Step 7: Return to Step 2



Example:

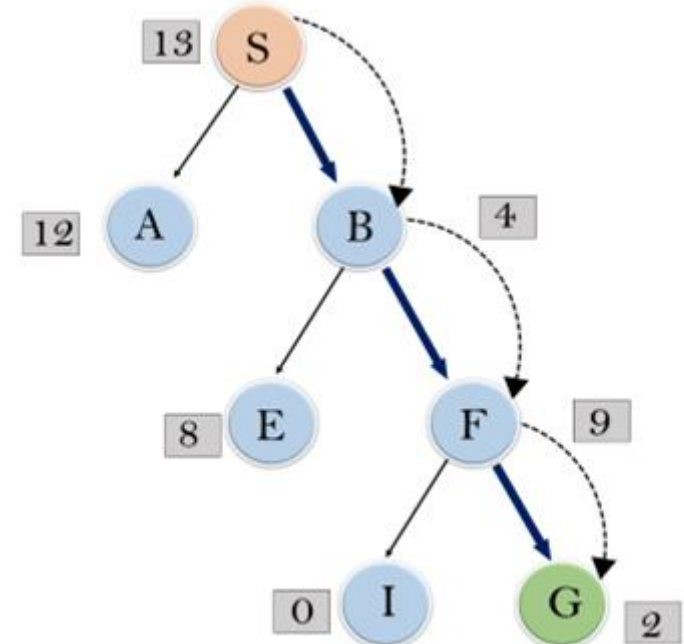
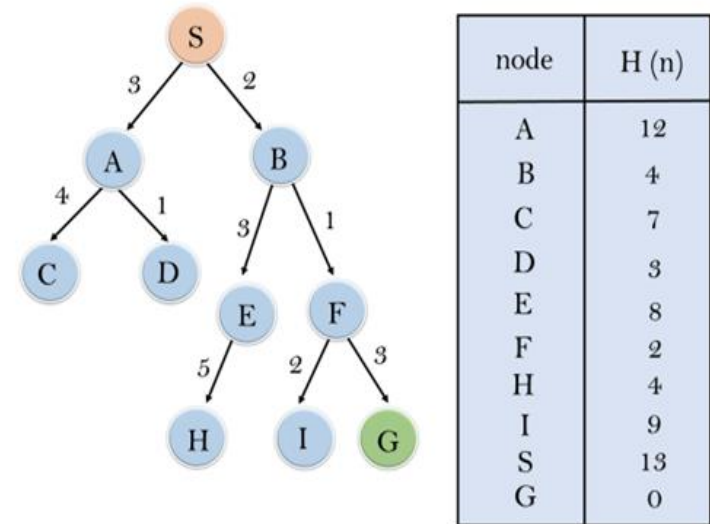
Consider the search problem, and we will traverse it using greedy best-first search. At each iteration, each node is expanded using evaluation function $f(n)=h(n)$, which is given in the table.

In this search example, we are using two lists which are OPEN and CLOSED Lists. Following are the iteration for traversing the example.

Expand the nodes of S and put in the CLOSED list

- Initialization: Open [A, B], Closed [S]
- Iteration 1: Open [A], Closed [S, B]
- Iteration 2: Open [E, F, A], Closed [S, B]
- : Open [E, A], Closed [S, B, F]
- Iteration 3: Open [I, G, E, A], Closed [S, B, F]
- : Open [I, E, A], Closed [S, B, F, G]

Hence the final solution path will be: **S--> B-->F--> G**

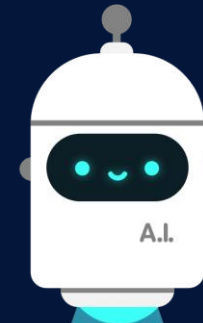


Advantages:

- Best first search can switch between BFS and DFS by gaining the advantages of both the algorithms.
- This algorithm is more efficient than BFS and DFS algorithms.

Disadvantages:

- It can behave as an unguided depth-first search in the worst case scenario.
- It can get stuck in a loop as DFS.
- This algorithm is not optimal.

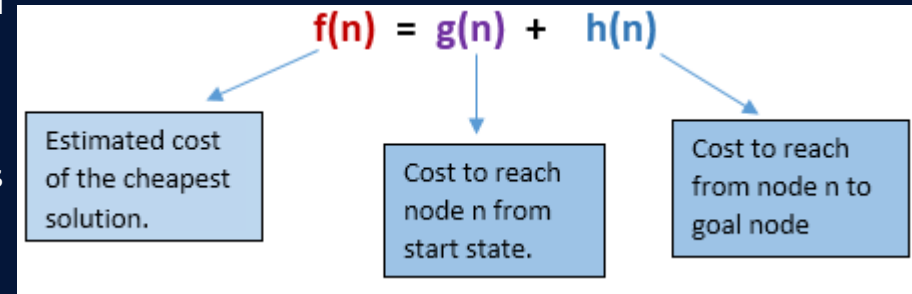


TASK 2

A* Search Algorithm:

A* search is the most commonly known form of best-first search. It uses heuristic function $h(n)$, and cost to reach the node n from the start state $g(n)$. It has combined features of UCS and greedy best-first search, by which it solve the problem efficiently. A* search algorithm finds the shortest path through the search space using the heuristic function. This search algorithm expands less search tree and provides optimal result faster. A* algorithm is similar to UCS except that it uses $g(n)+h(n)$ instead of $g(n)$.

In A* search algorithm, we use search heuristic as well as the cost to reach the node. Hence we can combine both costs as following, and this sum is called as a **fitness number**.



TASK 2

Algorithm of A* search:

Step 1: Place the starting node in the OPEN list.

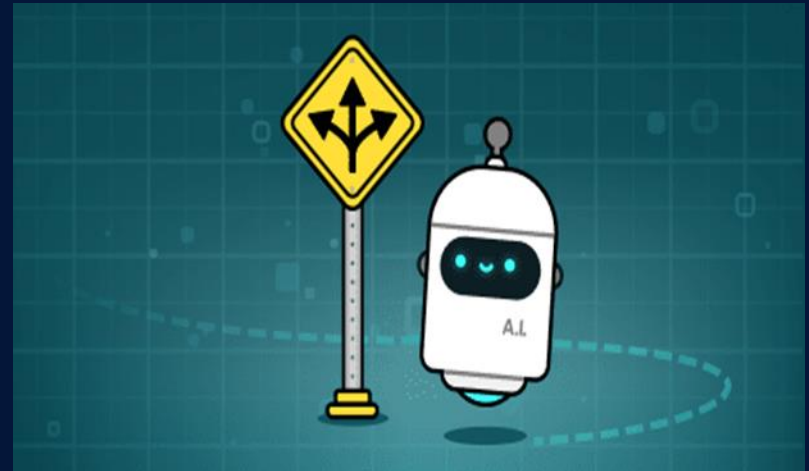
Step 2: Check if the OPEN list is empty or not, if the list is empty then return failure and stops.

Step 3: Select the node from the OPEN list which has the smallest value of evaluation function ($g+h$), if node n is goal node then return success and stop, otherwise.

Step 4: Expand node n and generate all of its successors, and put n into the closed list. For each successor n' , check whether n' is already in the OPEN or CLOSED list, if not then compute evaluation function for n' and place into Open list.

Step 5: Else if node n' is already in OPEN and CLOSED, then it should be attached to the back pointer which reflects the lowest $g(n')$ value.

Step 6: Return to Step 2.



ALGORITHM OF A* SEARCH:

Example:

In this example, we will traverse the given graph using the A* algorithm. The heuristic value of all states is given in the below table so we will calculate the $f(n)$ of each state using the formula $f(n) = g(n) + h(n)$, where $g(n)$ is the cost to reach any node from start state.

Here we will use OPEN and CLOSED list.

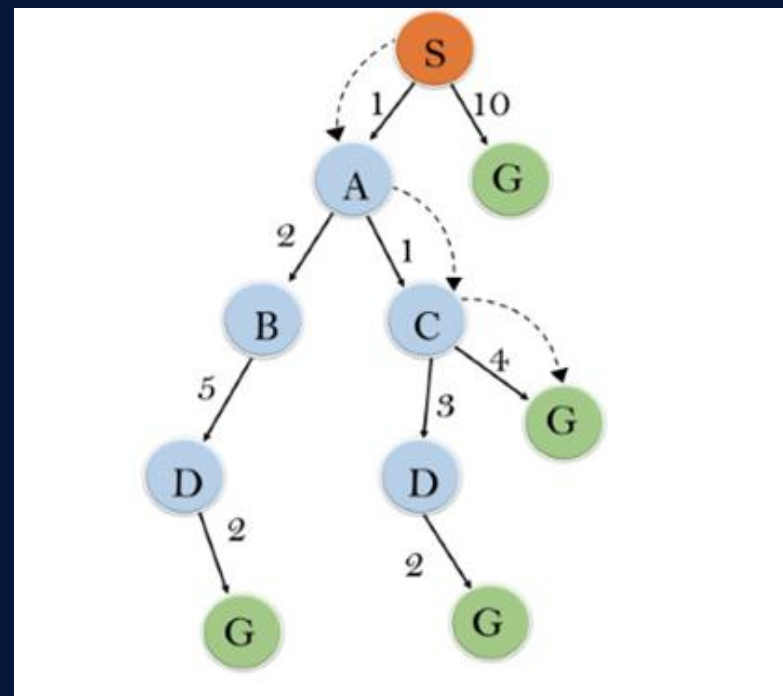
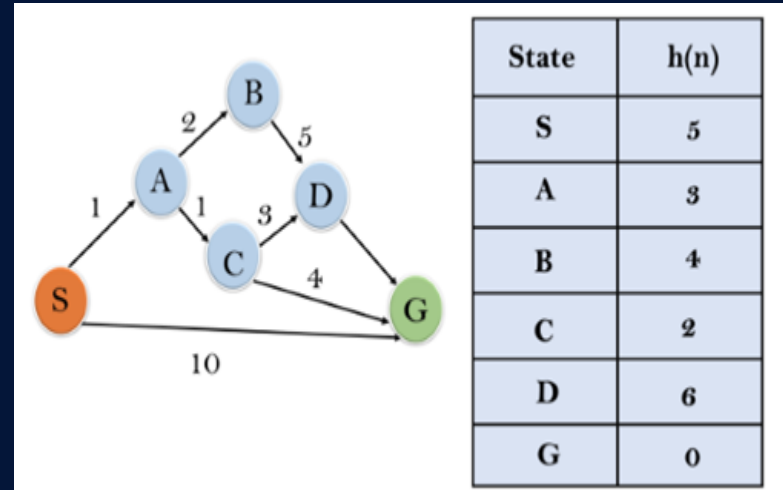
Initialization: $\{(S, 5)\}$

Iteration1: $\{(S \rightarrow A, 4), (S \rightarrow G, 10)\}$

Iteration2: $\{(S \rightarrow A \rightarrow C, 4), (S \rightarrow A \rightarrow B, 7), (S \rightarrow G, 10)\}$

Iteration3: $\{(S \rightarrow A \rightarrow C \rightarrow G, 6), (S \rightarrow A \rightarrow C \rightarrow D, 11), (S \rightarrow A \rightarrow B, 7), (S \rightarrow G, 10)\}$

Iteration 4 will give the final result, as $S \rightarrow A \rightarrow C \rightarrow G$ it provides the optimal path with cost 6.



ALGORITHM OF A* SEARCH:

Advantages:

- A* search algorithm is the best algorithm than other search algorithms.
- A* search algorithm is optimal and complete.
- This algorithm can solve very complex problems.

Disadvantages:

- It does not always produce the shortest path as it mostly based on heuristics and approximation.
- A* search algorithm has some complexity issues.
- The main drawback of A* is memory requirement as it keeps all generated nodes in the memory, so it is not practical for various large-scale problems





COMPLETE PERCENTAGES FOR EACH TASK:

Task 1	Task 2	Task 3
YC1-1: 100%	YC2-1: 100%	YC3-1: 0%
YC1-2: 100%	YC2-2: 100%	YC3-2: 0%
YC1-3: 100%	YC2-3: 100%	
YC1-4: 100%	YC2-4: 100%	
YC1-5: 100%	YC2-5: 100%	
YC1-6: 100%		

REFERENCE

1. <https://www.w3schools.com/python/default.asp>
2. http://ai.berkeley.edu/project_overview.html
3. <https://www.geeksforgeeks.org/a-search-algorithm/amp/>

THANK YOU

010010011011
011101010010