ASSIGNMENT-2

10/10

Pemmalla Dharan
AP21110010201

① Discuss the working principle of the following searching algorithms. Give a suitable example of one application where these searching algorithms can be used.

(i) Breadth First Search
(ii) Depth First Search
(iii) Iterative Deepening Search
(iv) Best First Search
(v) A* Search
(vi) Hill Climbing Search

(A) (i) Breadth First Search

Breadth first search algorithm uses a queue data structure technique to store the vertices, and the queue follows FIFO (first in first out) principle, which means that the neighbors of the node will be displayed, beginning with the node that was put first.

Generally the transverse of the BFS algorithm is approaching the nodes in two ways

- visited node
- not visited node

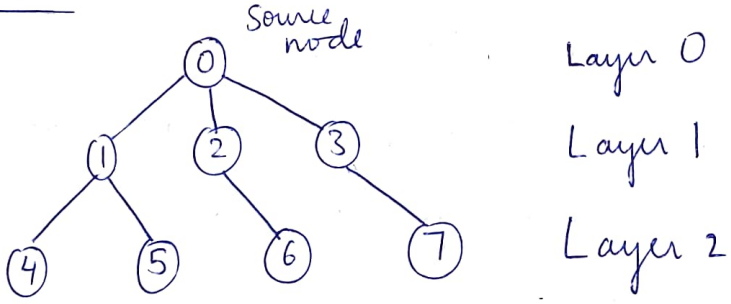
How does the algorithm operate?

- start with the source node
- add that node at the front of the queue to the visited list.
- make a list of the nodes as visited that are close to that vertex.

- and dequeue the nodes once they are visited.
- repeat the actions until the queue is empty.

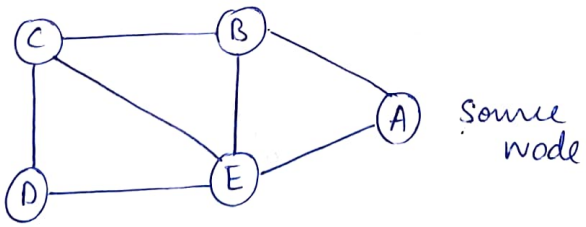
The BFS algorithm has a simple and reliable architecture.
The BFS algorithm helps evaluate nodes in a graph and determines the shortest path to traverse nodes.
The BFS algorithm can traverse a graph in the fewest number of iterations possible.
The iterations in the BFS algorithm are smooth; and there is no way for this method to get stuck in an infinite loop. It also has high level of accuracy.

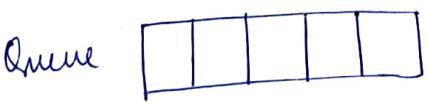
Architecture



Example

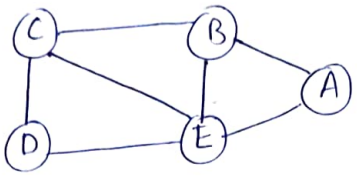
Step 1 -





Here source node is A

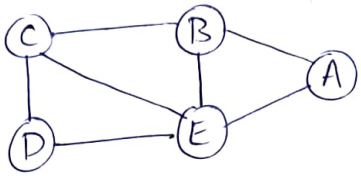
Step 2 -



Start from A and mark it as visited.

Queue - | A | | | | |

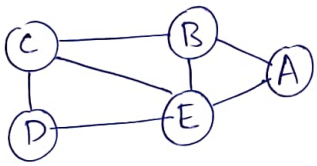
Step 3 -



Here we can choose B or E, then put it in visited.

Queue - | ~~A~~ ~~B~~ | | | |

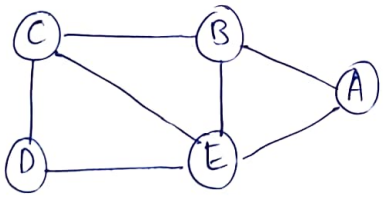
Step 4 -



Now enqueue E.

Queue - | A | B | ~~E~~ | | |

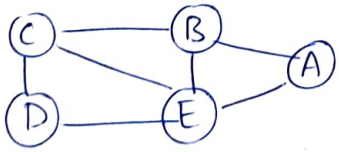
Step 5 -



Now, A doesnot have any unvisited nodes so dequeue it from the queue.

Queue - | ~~B~~ | ~~E~~ | | | |

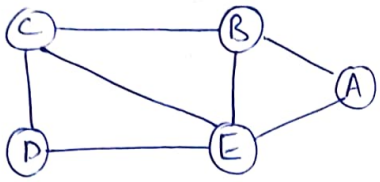
Step 6 -



Now C is unvisited node from B so enqueue C.

Queue - | B | E | C | | | |

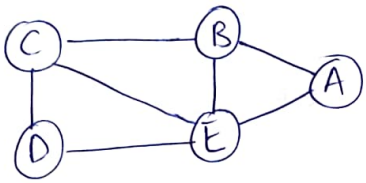
Step 7



since all nodes from B are visited then dequeue B

Queue - E C

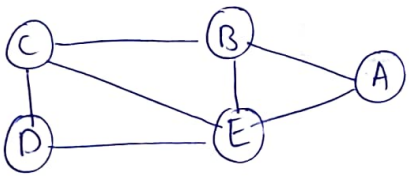
Step 8



Since D is unvisited enqueue D to queue.

Queue - E C D

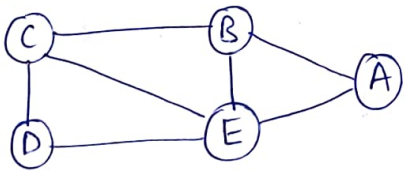
Step 9



All nodes visited from E so dequeue

Queue - C D

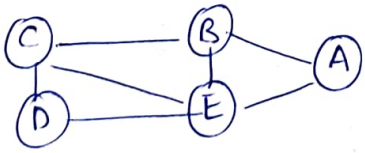
Step 10



All nodes visited from C so dequeue

Queue - D

Step 11



All nodes visited from D so dequeue

Queue -

Queue is empty, so our BFS traversal has ended.

Applications

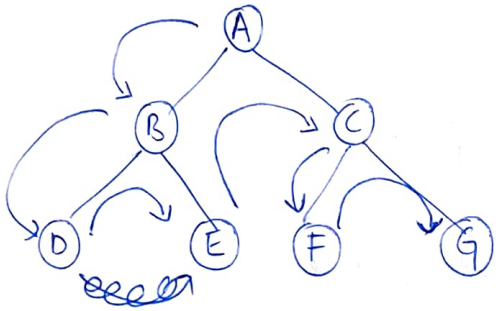
(1) For unweighted graphs, you must use the shortest path and minimum spacing tree.
(2) Peer to peer network (like bittorrent)
(3) Crawlers in search engine
(4) Social networking websites
(5) GPS navigation system
(6) Broadcasting network
(7) Cheney's Technique
(8) Cycle detection in graphs
(9) Identifying Routes
(10) Finding all nodes within one connected component.

(ii) Depth First Search

The depth first search algorithm traverses or explores data structures, such as trees and graphs. The algorithm starts at the root node and examines each branch as far as possible before backtracking.
When a dead end occurs in any iteration, the DFS method traverses a network in a deathward motion and uses a stack data structure to remember to acquire the next vertex to start a search.

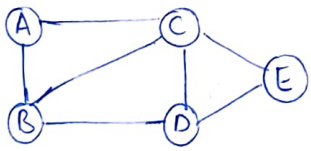
Implementation
- create a stack with the total no of vertices in the graph as the size.

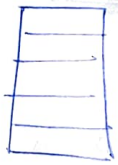
- choose any vertex as the traversal's beginning point. Push a visit to that vertex and add it to the stack.
- push any non-visited adjacent vertices of a vertex at the top of the stack to the top of the stack.
- repeat above steps until there are no more vertices to visit from the vertex at the top of the stack.
- if there are no new vertices to visit, go back and pop one from the stack using backtracking.
- continue until stack is empty.
- when the stack is entirely unoccupied, create the final spanning tree by deleting the graph's unused edges



Example -

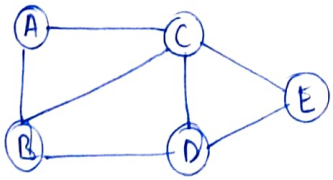
Step 1 -





Stack

Mark A as visited.
Push A to top of stack



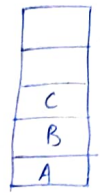
Stack

Step 2- Nearby unvisited vertex of A, say B should be visited.
Push B into stack.

Stack

Step 3- From C and D, visit any adjacent unvisited vertices of B. Lets take C.

Push C



Stack

Step 4- Now push D

Stack

Step 5- Vertex E is remaining so push E

Stack

Step 6- Vertex E's nearby vertices are visited, so pop E

Stack

Now pop D

Stack

⇒

then C

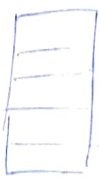
Stack

~~then~~ then B

Stack

⇒

then A


Stack

Time complexity is O(v)
Space complexity is O(v)

Applications –

1. Detecting a graph's cycle
2. Topological Sorting (mainly used to schedule jobs based on dependencies between them)
3. To determine if a graph is bipartite.
4. Finding strongly connected components in a graph.
5. Solving mazes and other puzzles with only one solution
6. Path finding

(iii) Iterative Deepening Search

IDS is an integral component. They are used to solve a variety of issues. It is a search algorithm that combines the benefits of BFS and DFS, which means it continually runs DFS, raising the depth limit each time, until desired result is obtained. This makes the search performed is thorough and efficient.

Working

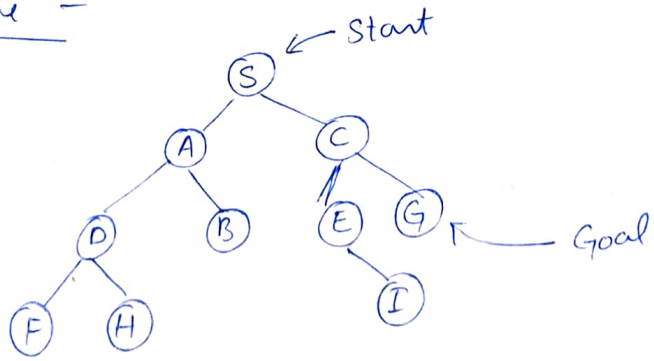
The IDS function performs IDS on the graph using a root node and a goal node as inputs until the goal is attained or the search space is used up. This is accomplished by regularly using the ~~depth Limited Search~~ function which applies a depth restriction to DFS. The search ends and returns the goal node if the goal is located at any depth. The result is none if the search space is used up.

The function conducts DFS on the graph with the specified depth limit by taking as inputs a node, a distination node and a depth limit. The search results found if the desired node is located at the current depth. The search returns not found if the dep limit is reached but the goal node cannot be located. If neither criterion is true, the search iteratively moves on to the node offspring

Advantages

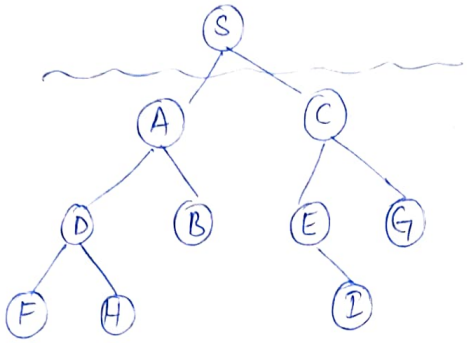
- IDS is superior to other search algos in a no of ways. The first benefit is that it is comprehensive, which ensures that a solution will be found if one is there in the search space. This is so that all nodes under a specific depth limit are investigated before the depth limit is raised by IDS, which does a depth-limited DFS.
- IDS is memory efficient, which is its second benefit. This is because IDS decreases the algorithm's memory needs by not storing every node in the search area in memory. IDS minimises the algorithm's memory footprint by only storing the nodes up ~~to~~ the current depth limit.
- IDS's ability to be utilised for both tree and graph search is its third benefit. This is due to the fact that IDS is a generic search algorithm that works on any search space, including a tree or a graph.

Example -



Step 1

1st iteration, depth = 0

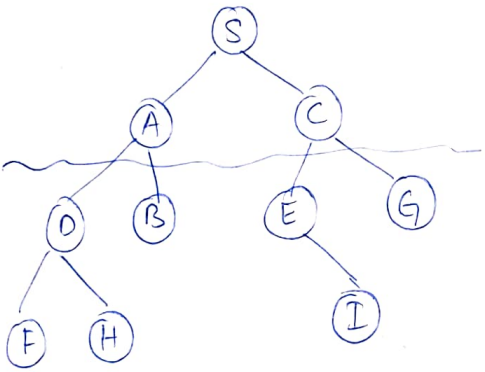


Level 0

Path = [S]

Step 2

2nd iteration, depth = 1

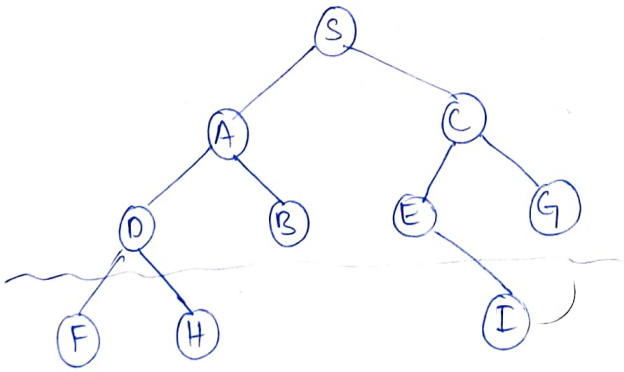


Level 1

Path = [S, A, C]

Step 3

3rd iteration, depth = 2



Level 2

Path = [S, A, D, B, C, E, G]

Here goal node is reached so we stop the process.

Time Complexity is $O(b^d)$ where b is branching factor and d is the depth of the goal node.
Space complexity is $O(bd)$

Applications

1. Analyzing networks in AI and data science.
2. Solving puzzles with a unique solution, such as sudoku.
3. Detecting cycles in graphs.
4. Sorting directed acyclic graphs.
5. Solving the N-Queens-Problem.
6. Playing games like chess and checkers.
7. Locating the shortest route on a map.

(iii) Best First Search

It is a popular search algorithm used for solving problems such as pathfinding, puzzle solving and other optimization tasks. It is a heuristic search algorithm that selects the most promising node for expansion based on an evaluation function or heuristic. The choice of the best node is made according to the estimated cost of reaching the goal from that node.

Working

The BFS method that operates by maintaing two lists - open and closed. It assigns each node a combined value (f-value) that reflects both the cost incurred so far (g-value) and an estimate of the remaining cost to reach the goal. It selects nodes from the open list with the lowest f-value for expansion, prioritizing paths that seem most likely to lead to the goal efficiently. It generates

successor nodes, calculates their f-values, and adds or updates them in the open list as appropriate. If the goal is reached during the process, the algorithm terminates, having found an optimal solution. If the open list becomes empty without reaching the goal, the search terminates, indicating no solution. It is complete and optimal when the heuristic is admissible, making it widely used algorithm for finding the shortest paths and solving optimization problems.

Advantages

(1) It is complete, meaning it will find a solution if one exists. It explores all nodes at a given depth level before moving on to the next level, ensuring that all searches all possible paths in a graph or tree.

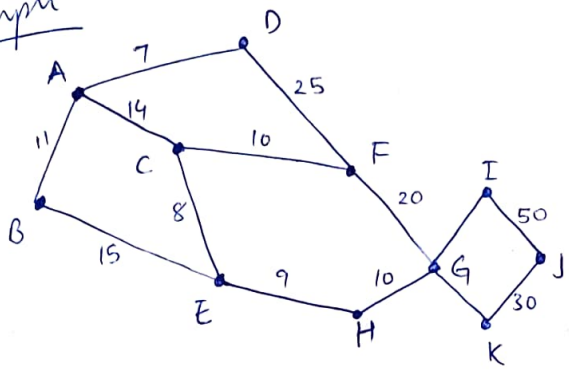
(2) ~~Often used to find the shortest path in unweighted graphs,~~ BFS

(2) Effective while dealing with large search spaces or state spaces. By selecting the most promising node based on heuristic values, it can quickly navigate through the space and reach a solution without exhaustively exploring all possibilities.

(3) Relies on heuristics that provide informed estimates of the cost to reach the goal from a given state. This heuristic guidance allows the algorithm to focus its search on the most likely paths, making it highly efficient in finding solutions faster than uninformed search algorithms.

(4) When an admissible heuristic is used, BFS is guaranteed to find optimal solution. This optimality is a significant advantage in optimization problems.

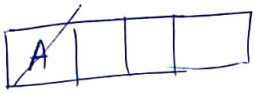
Example -



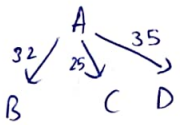
straight line distance

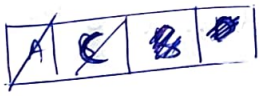
$A \rightarrow G = 40$
$B \rightarrow G = 32$
$C \rightarrow G = 25$
$D \rightarrow G = 35$
$E \rightarrow G = 19$
$F \rightarrow G = 17$
$H \rightarrow G = 10$
$G \rightarrow G = 0$

Step 1

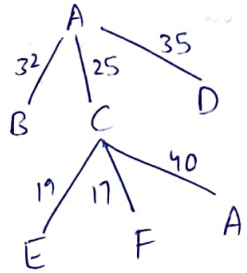


Step 2

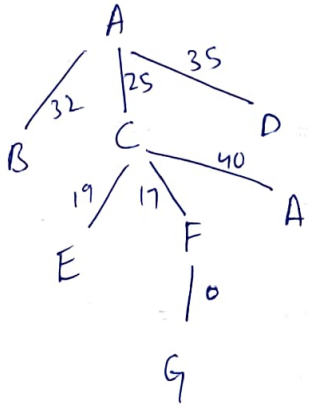




Step 3



Step 4



Applications

(1) Commonly used in route planning and navigation systems, helping find the shortest or most efficient path from a starting location to a destination. Widely used in GPS navigation, map applications and autonomous vehicles.

(2) Plays a crutial role in computer networks, where it assists in routing data packets through a network to reach their destination efficiently. Network routers use BFS to determine the best path for data transmission.

(3) Used in some natural language processing tasks, such as parsing and language generation, where it helps generate sentences or analyze syntax trees efficiently.

(4) Robots and autonomous system use BFS to plan their movements and avoid obstacles.

(5) Used in puzzle solving like 8-puzzle.

(V) A* Search Algorithm

A* search algorithm is a widely used graph traversal and pathfinding algorithm. It is used to find the shortest path from a starting node to a goal node in a graph, where each edge has a non-negative cost or weight associated with it. A* is espcially popular for applications like pathfinding in games, route planning and AI

The A* algorithm combines the benefits of two other popular search algorithms: dijkstra's algorithm and best first search. It uses heuristics to estimate the cost of reaching the goal from the current node. A* is informed, meaning it takes into account both the actual cost from the starting node & an estimate of the cost to the goal and it selects nodes to explore based on their ~~heuristics~~ $f(n)$ values, where $f(n) = g(n) + h(n)$.

Working

1. Initialize an open list & add the starting node to it with a $g(n)$ value of 0 and an estimated $h(n)$ value for the cost to reach the goal from the starting node.
2. Initialize a closed list to keep track of nodes that have been explored.
3. While the open list is not empty
   a) select the node with the lowest $f(n)$ value from the open list. This node will be the next one to explore.
   b) If the selected node is the goal node, you have found the optimal path, and the search can terminate.
   c) If the selected node is not the goal node, remove it from the open list and add it to the closed list to mark it as explored.
   d) Expand the selected node by generating its neighbouring nodes.
   e) For each successors node:
   - calculate the cost to reach the successor node from the current node.
   - calculate an estimate of the cost to the goal from the successor node.
   - calculate the total cost $f(n) = g(n) + h(n)$
   - if the successor node is in the closed list and has a lower $f(n)$ value, skip it.

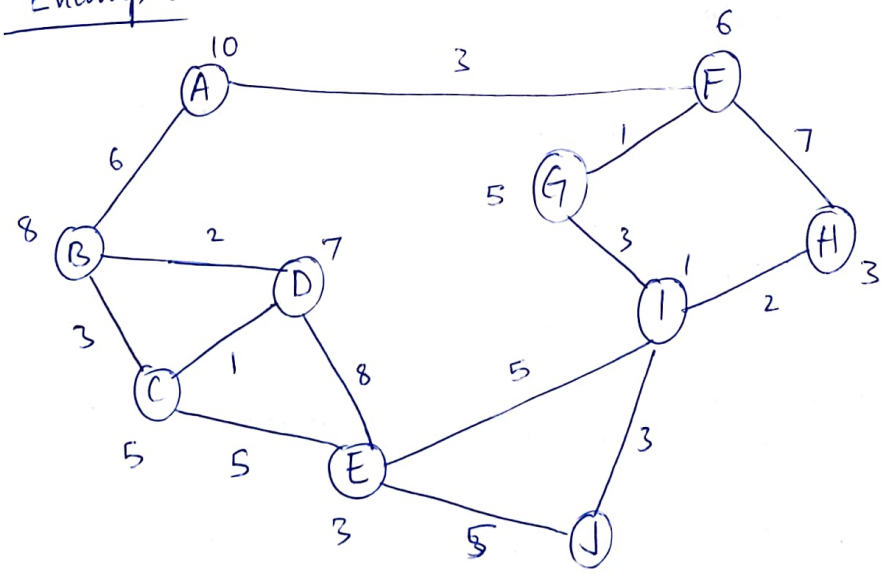
- if the successor node is not in the open list or has a higher $f(n)$ value, add it to the open list with the updated $f(n)$ value.

4. If the open list becomes empty and the goal node has not been reached, then there is no path from the start to the goal, and the search terminates.

Advantages

- It guarantees the shortest path
- It is faster than many other search algorithms due to informed exploration.
- You can tailor A* to specific problems by adjusting the heuristic.
- A* will find a solution if it exists.
- Techniques can be used to manage memory.
- A* produces high-quality paths and allows for path customization.

Example -



Step 1

Start with node A

Node B and Node F can be reached from node A

A* algorithm calculates $f(B)$ and $f(F)$

$f(B) = 6+8 = 14$

$f(F) = 3+6 = 9$

Since $f(F) < f(B)$ so it decides to go to node F

Path = $A \rightarrow F$

Step 2

Node G and Node H can be reached from node F.

A* algorithm calculates $f(G)$ and $f(H)$

$f(G) = (3+1)+5 = 9$

$f(H) = (3+7)+3 = 13$

since $f(G) < f(H)$, so it decides to go to node G.

Path = $A \rightarrow F \rightarrow G$

Step 3

Node I can be reached from node G.

A* algorithm calculates $f(I)$

$f(I) = (3+1+3)+1 = 8$

It decides to go to node I

Path = $A \rightarrow F \rightarrow G \rightarrow I$

Step 4

Node E, Node H and Node J can be reached from node I.

A* algorithm calculates $f(E)$, $f(H)$ and $f(J)$

$f(E) = (3+1+3+5)+3 = 15$

$f(H) = (3+1+3+2)+3 = 12$

$f(J) = (3+1+3+3)+0 = 10$

Since $f(J)$ is least, so it decides to go to node J.

Path = $A \rightarrow F \rightarrow G \rightarrow I \rightarrow J$

(vii) Hill Climbing Search

Hill climbing is a local search algorithm that starts from an initial state and repeatedly moves to a neighboring state that improves the objective function. It terminates when no better neighbor can be found. Hill climbing can be used in optimization problems, such as finding the optimal configuration for a machine or tuning parameters in machine learning models.

Working

The algorithm works on the following steps in order to find an optimal solution.

- It tries to define the current state as the state of starting or the initial state.
- It generalizes the solution to the current state and tries to find an optimal solution. The solution obtained may not be the best.
- It compares the solution which is generated to the final state also known as the goal state.
- It will check whether the final state is achieved or not, if not achieved, it will try to find another solution.

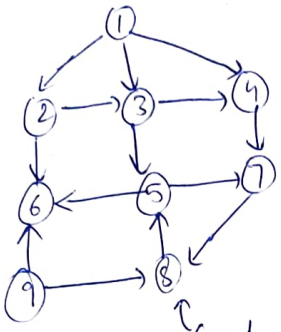
Advantages

- It is a very useful technique while solving problems like job searching, salesman technique, chip design and management.
- The agent moves in the direction of the goal which optimizes our cost.

(2) Draw a comparative analysis of different graph searching algorithms considering their time complexity, space complexity, completeness, optimal nature and information required.

(1) BFS (Breadth First Search) → Time Complexity $O(b^d)$
Space Complexity $O(b^d)$

b → branching factor, d → solution depth.



Open list	Closed list
1	1
2, 3, 4	2
3, 4, 6	3
4, 6, 5	4
6, 5, 7	6
5, 7	5
7	7
8	

Completeness – BFS is complete if the graph is infinite. It will find the goal state if one exists.

Optimality – BFS guarantees the shortest path if edge weights are uniform.

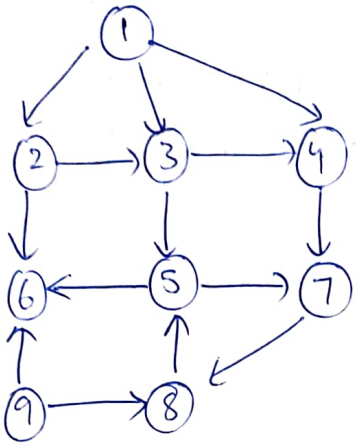
Information Required – No additional information is required, BFS explores all neighbours before moving to the next level.

(2) DFS (Depth First Search) → Time Complexity $O(b^m)$
Space Complexity $O(bm)$

Completeness – DFS may not be complete if there are infinite paths or cycles in the graph.

Optimality – DFS doesnot guarantee an optimal soln

Information Required – It requires less memory than BFS, but it can get stuck in infinite loops.



Open list	Closed list
1	1
4, 3, 2	2
4, 3, 6	6
4, 3	3
4, 5	5
4, 7	7
4, 8	–

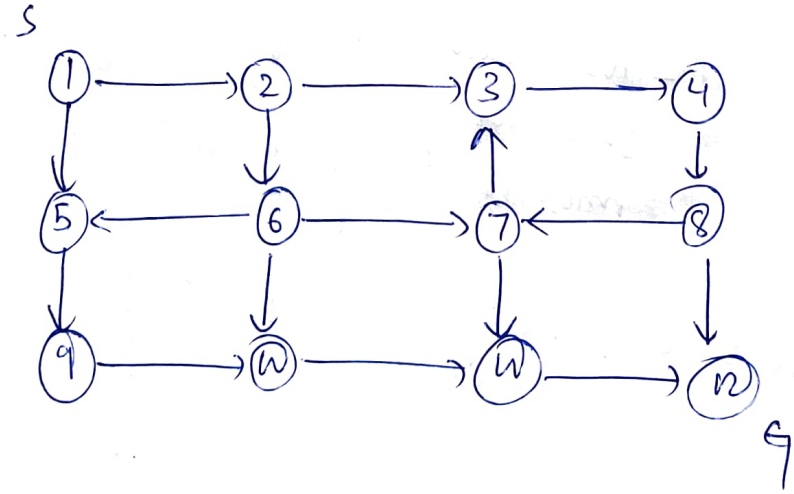
③ Uniform Cost Search → Time Complexity $O(b^{1+C^*/\epsilon})$

Space Complexity $O(b^{1+C^*/\epsilon})$

Completeness – UCS is complete if there are no cycles with negative edge weight

Optimality – UCS is optimal for non-negative edge weight.

Information Required – It doesn't require any additional information beyond the edge weights.



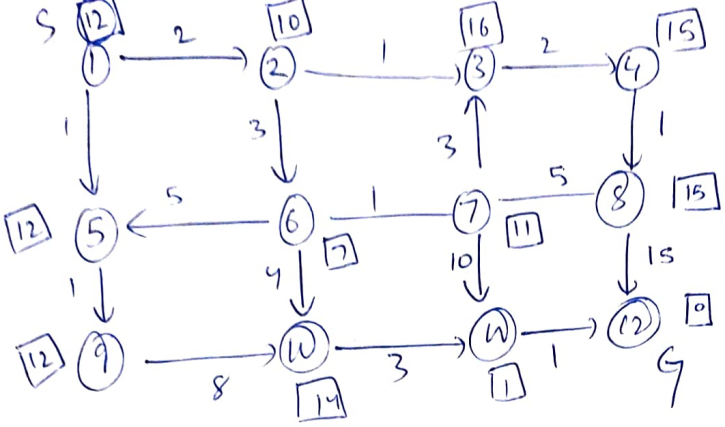
Open list	Closed list
$1^{(0)}$	$1^{(0)}$
$2^{(2)}, 5^{(1)}$	$5^{(1)}$
$2^{(2)}, 9^{(2)}$	$2^{(2)}$
$9^{(2)}, 6^{(5)}, 3^{(3)}$	$9^{(2)}$
$6^{(5)}, 3^{(3)}, 80^{(10)}$	$3^{(3)}$
$6^{(5)}, 10^{(10)}, 4^{(5)}$	$6^{(5)}$
$10^{(10)}, 4^{(5)}, 7^{(6)}, 10^{(9)}$	$4^{(5)}$
$7^{(8)}, 8^{(6)}, 11^{(16)}$	$7^{(6)}$
$10^{(9)}, 8^{(6)}, 11^{(16)}$	$8^{(6)}$
$10^{(9)}, 11^{(16)}, 12^{(21)}$	$10^{(9)}$
$12^{(21)}, 11^{(12)}$	$11^{(12)}$
$12^{(21)}, 12^{(13)}$	–

④ A* algorithm → Time complexity $O(b^d)$
Space complexity $O(b^d)$

Completeness – A* is complete if the search space is finite.

Optimality – A* is optimal if the heuristic is admissible (never overestimates the cost to reach the goal)

Information required – It requires a heuristic function that estimates the cost to reach the goal.



Open list	Closed list
$1^{(12)}$	$1^{(12)}$
$2^{(12)}, 5^{(13)}$	$2^{(12)}$
$5^{(13)}, 3^{(19)}, 6^{(12)}$	$6^{(12)}$
$5^{(13)}, 3^{(19)}$	$5^{(13)}$
$3^{(19)}, 7^{(17)}, 10^{(13)}, 9^{(17)}$	$10^{(13)}$
$3^{(19)}, 7^{(17)}, 9^{(14)}, 11^{(13)}$	$11^{(13)}$
$3^{(19)}, 7^{(17)}, 9^{(14)}, 12^{(13)}$	—

(5) **Best First Search** → Time complexity $O(b^d)$
Space complexity $O(b^d)$

Completeness – BFS may not be complete
Optimality – It is not guaranteed to be optimal.
Information required – It requires a heuristic that estimates the cost from the current node to goal.