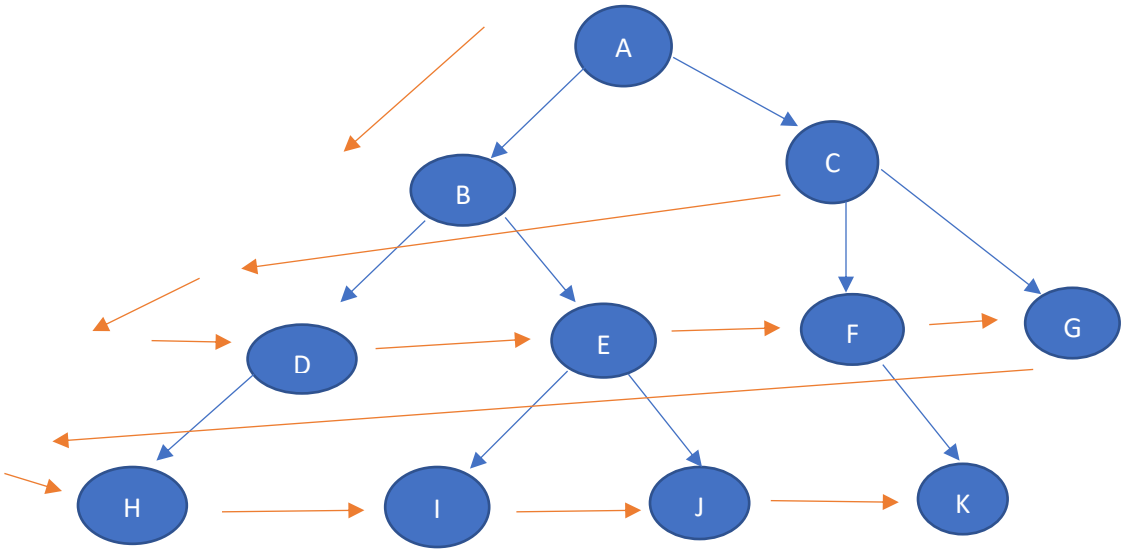
Assignment

Q1. Describe and differentiate BFS, DFS, and UCS with an example. Show necessary simulations.

Ans:

Breadth-First Search (BFS): Breadth-first search is a simple strategy in which the root node is expanded first, then all the successors of the root node are expanded next, then *their* successors, and so on. It is another search algorithm in AI which traverses breadthwise to search the goal in a tree. It begins searching from the root node and expands the successor node before going expanding it further expands along breadthwise and traverses those nodes rather than searching depth-wise.



The above figure is an example of a BFS Algorithm. It starts from the root node A and then traverses node B. Till this step it is the same as DFS. But here instead of expanding the children of B as in the case of DFS we expand the other child of A node C because of BFS and then move to the next level and traverse from D to G and then from H to K in this typical example. To traverse here we have only taken into consideration the lexicographical order. This is how the BFS Algorithm is implemented.

Advantages:

1. BFS will provide a solution if any solution exists.

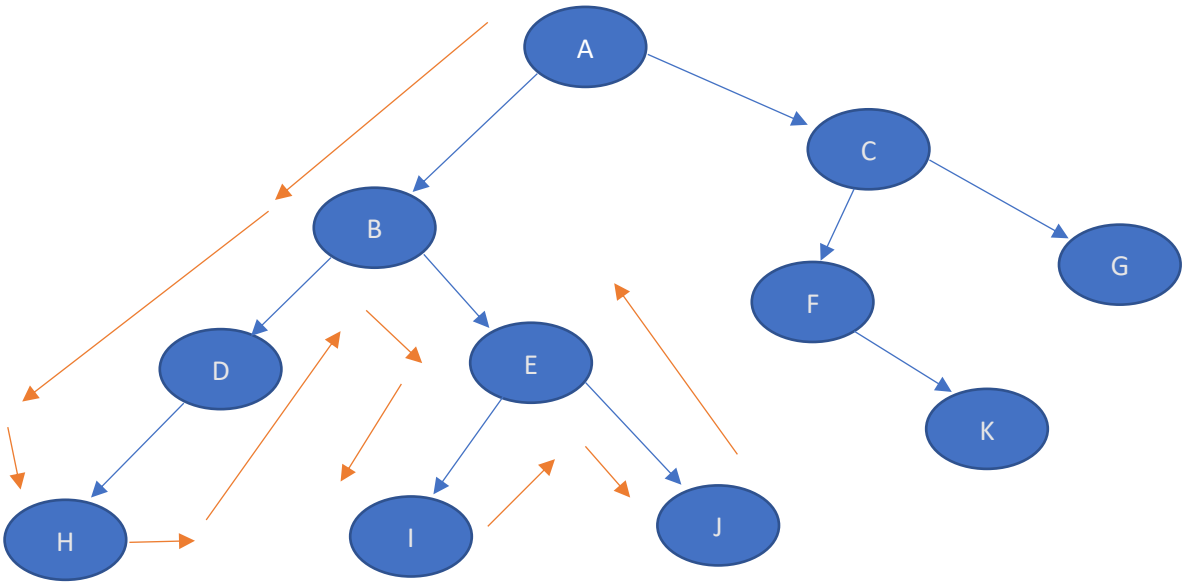
2. If there is more than one solution for a given problem, then BFS will provide the minimal solution which requires the least number of steps.

Disadvantages:

1. It requires lots of memory since each level of the tree must be saved into memory to expand the next level.

2. BFS needs lots of time if the solution is far away from the root node.

Depth First Search (DFS): It is a search algorithm where the search tree will be traversed from the root node. It will be traversing, searching for a key at the leaf of a particular branch. If the key is not found the searching retraces its steps back to the point from where the other branch was left unexplored and the same procedure is repeated for that other branch.



The above figure is an example of a BFS Algorithm. It starts from the root node A and then traverses node B. Till this step it is the same as DFS. But here instead of expanding the children of B as in the case of DFS we expand the other child of A node C because of BFS and then move to the next level and traverse from D to G and then from H to K in this typical example. To traverse here we have only taken into consideration the lexicographical order. This is how the BFS Algorithm is implemented.

Advantages:

1.BFS will provide a solution if any solution exists.

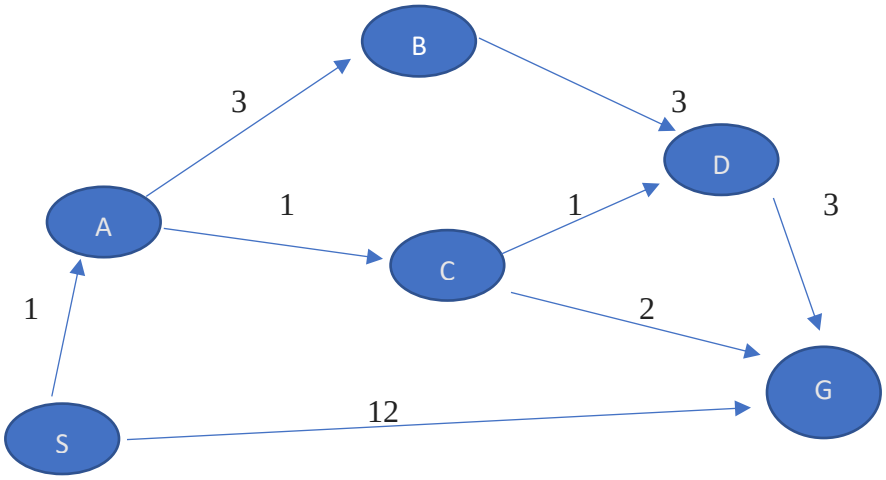
2. If there is more than one solution for a given problem, then BFS will provide the minimal solution which requires the least number of steps.

Disadvantages:

1. It requires lots of memory since each level of the tree must be saved into memory to expand the next level.

2.BFS needs lots of time if the solution is far away from the root node.

Uniform Cost Search (UCS): This algorithm is mainly used when the step costs are not the same but we need the optimal solution to the goal state. In such cases, we use Uniform Cost Search to find the goal and the path including the cumulative cost to expand each node from the root node to the goal node. It does not go depth or breadth, it searches for the next node with the lowest cost and in the case of the same path cost, let's consider lexicographical order in our case.



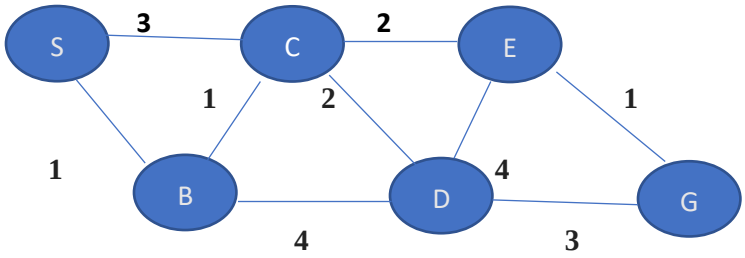
In the above figure consider S to be the start node and G to be the goal state. From node S we look for a node to expand and we have nodes A and G but since it's a uniform cost search it's expanding the node with the lowest step cost so node A becomes the successor rather than our required goal node G. From A we look at its children nodes B and C. So since C has the lowest step cost it traverses through node C and then we look at successors of C i.e. D and G since the cost to D is low we expand along with the node D. Since D has only one child G which is our required goal state we finally reach the goal state D by implementing UFS Algorithm. If we have traversed this way definitely our total path cost from S to G is just 6 even after traversing through many nodes rather than going to G directly where the cost is 12 and 6<<12(in terms of step cost). But this may not work with all cases.

Advantages:

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

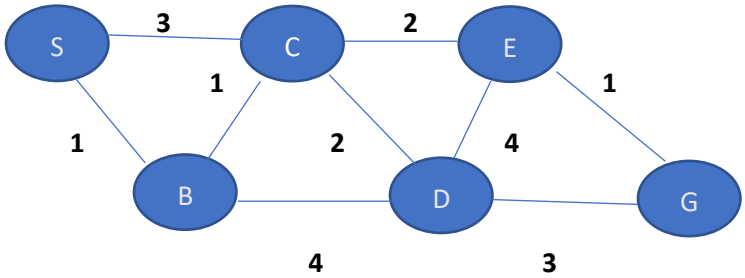
Disadvantages:

- 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.

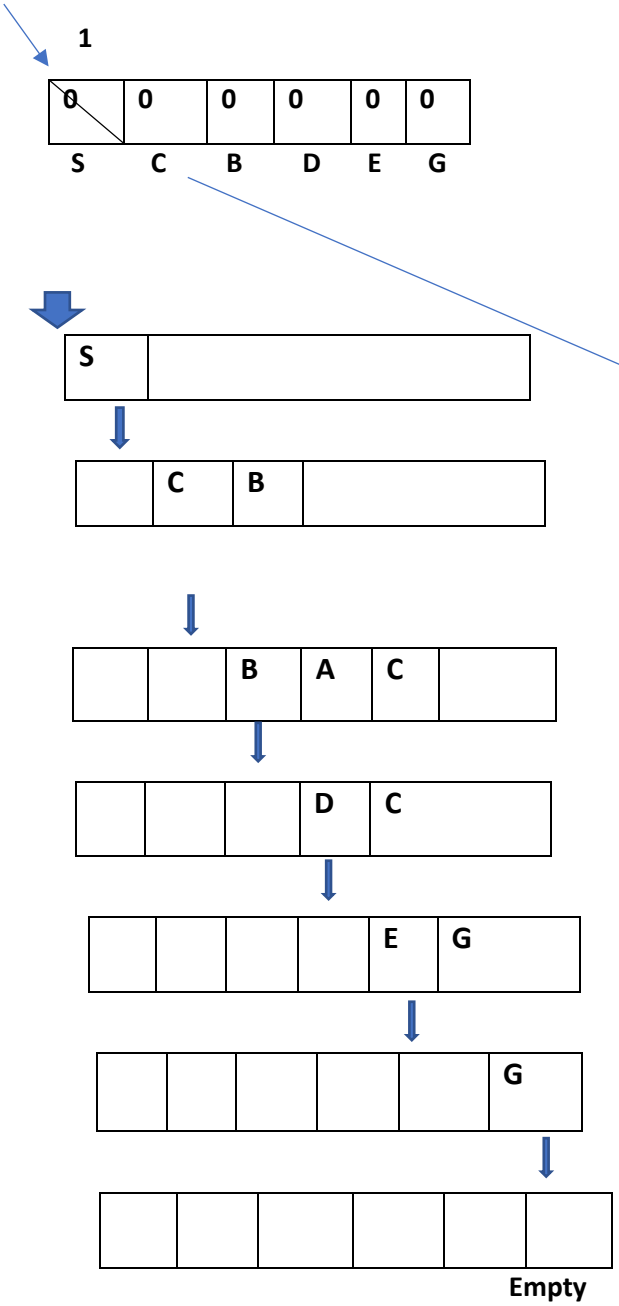


	BFS		DFS		UCS	
Here	Queue	Current. node	Stack	Current. node	Queue	Current. node
1	S	S	S	S	S	S
2	C	C	B	C	B	C
3	B	B	D	B	C	B
4	D	D	Q	D	D	D
5	E	E	E	E	E	C
6	G	G	C	G	G	G

BFS :

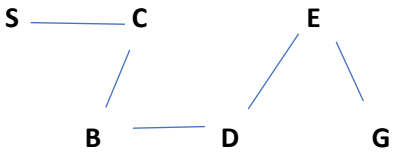


Agent start at node S.



Here we selected S and take the adjacent of S i.e C B then take the adjacent of C and So on.

BFS order is S, C, B, D, C, G

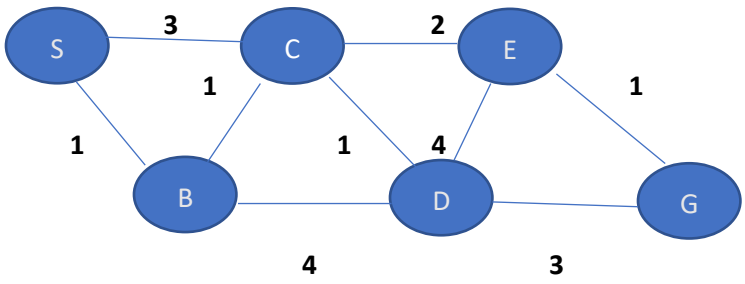


BFS graph

BFS [Breadth First Search]

```
BFS(V)
{
Visited (V)=1
Add (V, Q)
While (Q is not empty)
{
X=delete (Q)
Print(x)
For all w adj to x
{
If (w is not visited)
{
Visited(w)=1
Add (W, Q);
}
}
}
}
```

DFS :



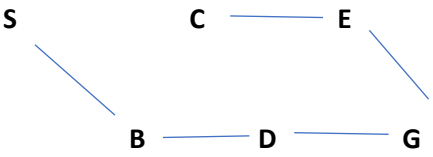
In this B stack is used because is middle backtrack is used

1

0	0	0	0	0	0
S	C	B	D	E	G

Agent start from S. S → B → D → G → E → C

DFS order : S, B, D, G, E, C

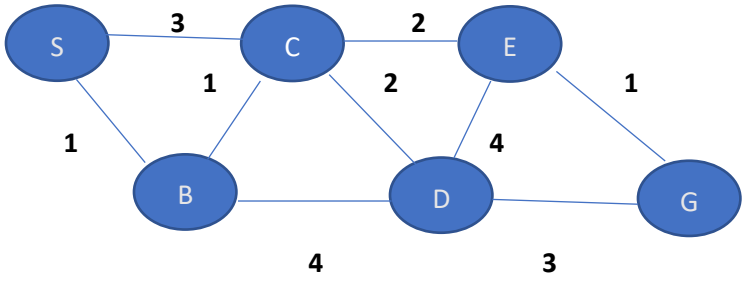


5	C
4	E
3	G
2	D
1	B
0	S

DFT (depth first traversal)

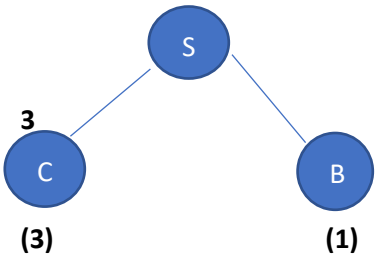
```
DFT(V)
{
Visited(V)=1;
If(V);
For all w adj to v
{
  If (w is not visited)
DFT(w);
}
}
```

UCS (Uniform Cost Search)



Step 1 - S {start with search agent}

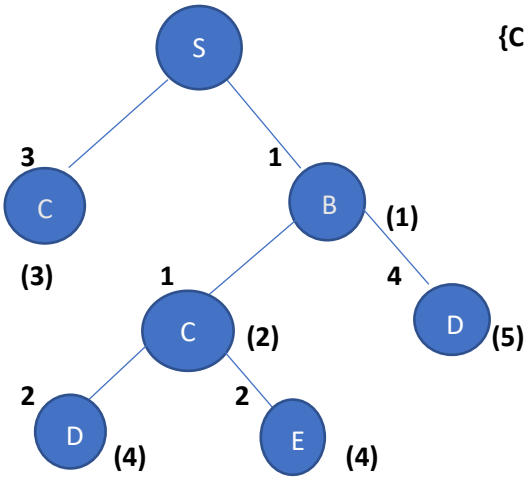
Step 2 -



{.adjacent of node S ie C,B with their cost 3,1

Respectively, now we will go to minimum cost}

Step 3 -

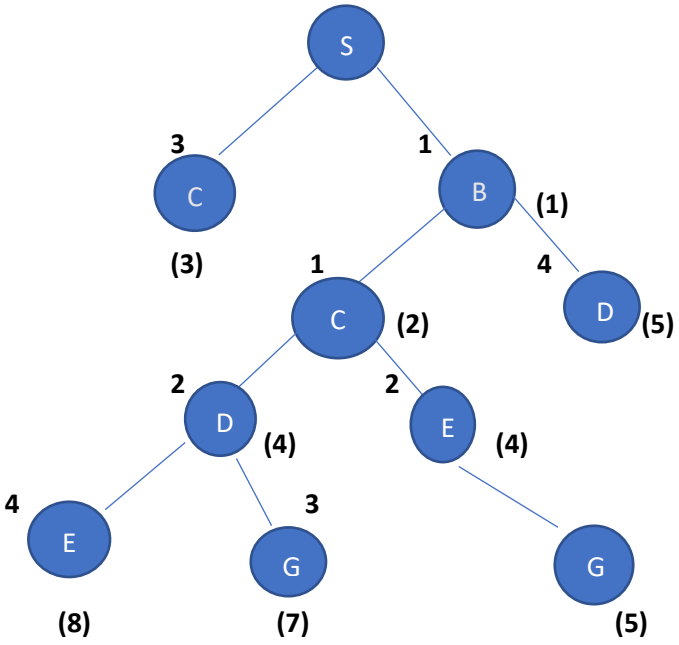


{C has minimum cost among all other node}

{and so on check the cost put the minimum

Cost node in the queue}

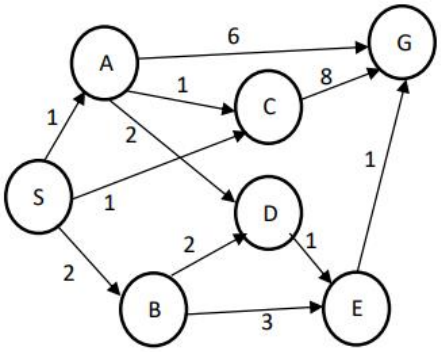
Step 4 -



UCS order: S, B, C, D, E, G

Q2.

Consider the following search problem, represented as a graph. The start state is S and the only goal state is G. The heuristic values are given in the table below.



S	A	B	C	D	E	G
10	8	5	5	4	2	0

Perform Greedy Best First and A* search on the graph.

Show the sequence of the expanded nodes for both algorithms. Explain which algorithm is faster?

Discuss the optimality of these two algorithms with respect to the above search problem.

Ans: Greedy Best Search Algorithm works by selecting the minimum heuristic functions and tries to reach the goal, ignoring the cost involved in reaching the goal. This is why it has the word Greedy in it as it does not care about the cost that is involved in reaching the goal. Greedy Best First Search uses the following formula:

f(n) =h(n)

where: f(n) is the desired value, and

h(n) is the heuristic value.

As we see in the graph, starting from S, we have three options, A, B and C. We will choose C as it seems the best option seeing that it has the least heuristic value and closest to goal G.

After that, we will directly reach goal G from C.

Using a greedy algorithm, expand the first successor of the parent. After a successor is generated:

1. If the successor's heuristic is better than its parent, the successor is set at the front of the queue (with the parent reinserted directly behind it), and the loop restarts.
2. Else, the successor is inserted into the queue (in a location determined by its heuristic value). The procedure will evaluate the remaining successors (if any) of the parent.

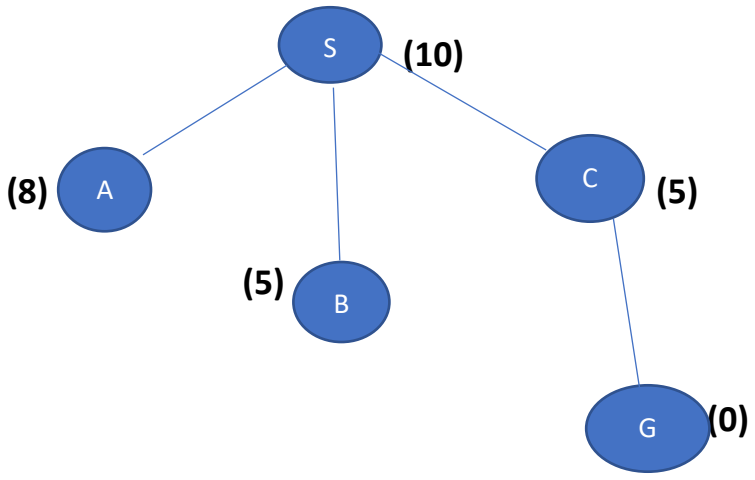
A * algorithm is a searching algorithm that searches for the shortest path between the *initial and the final state.* It is used in various applications, such as *maps*. In *maps* the A* algorithm is used to calculate the shortest distance between the source (initial state) and the destination (final state).

A* algorithm has 3 parameters:

1. **g:** the cost of moving from the initial cell to the current cell. Basically, it is the sum of all the cells that have been visited since leaving the first cell.
2. **h:** also known as the *heuristic value*, it is the **estimated** cost of moving from the current cell to the final cell. The actual cost cannot be calculated until the final cell is reached. Hence, h is the estimated cost. We **must** make sure that there is **never** an over estimation of the cost.
3. **f:** it is the sum of g and h. So, **f = g + h**

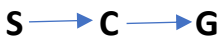
The way that the algorithm makes its decisions is by taking the f-value into account. The algorithm selects the *smallest f-valued cell* and moves to that cell. This process continues until the algorithm reaches its goal cell.

Refer to diagram below for sequence of expanded nodes below:



Greedy Best First

f(n) = h(n)



Hence, from this diagram we can see that the Greedy Best First Search Algorithm path is: S-C-G with a f(n) value of 15.

A* Search Algorithm works by calculating both the cost and the heuristic value to determine the path to reach the goal. It selects the path which has the lowest total value.

A* Search Algorithm use the following formula:

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

Where:

F(n) is the desired value,

G(n) is the cost, and

H(n) is the heuristic value.

In this graph, we see that there are total 6 paths that we can take. They are as follows:

- **S-A-G**
- **S-A-C-G**
- **S-A-D-E-G**
- **S-C-G**
- **S-B-D-E-G**
- **S-B-E-G**

**** S-A-G**

f(n)= (1+6) + (8+0)

f(n)=15

****S-A-C-G**

f(n)= (1+1+8) + (8+4+2+0)

f(n)=23

****S-A-D-E-G**

f(n)= (1+2+1+1) + (8+4+2+0)

f(n)= 19

****S-C-G**

f(n)= (1+8) + (5+0)

f(n)=14

****S-B-D-E-G**

f(n)= (2+2+1+1) + (5+4+2+0)

f(n)=17

****S-B-E-G**

f(n)= (2+3+1) + (5+2+0)

f(n)=13

A* = S-B-E-G

Hence, from calculations, we can see that A* Search Algorithm path is :

S-B-E-G with a f(n) value of 13.