CHRIS SUNNY THALIYATH

AI Assignmen #2 (MIDSEM)

Question 1> Design an Algorithm to search an Infinite graph which follows DFS & BFS. Explain time & space Complexity of the proposed algorithm

Table 1

Entity	Units	Cost (in lacs)
Classroom (C)	1	5
Hostel (H)	1	18
Residence (R)	1	30
Labs (L)	1	6

Answer

Pseudo Code :
depth First Search (G, S) : Graph G, Stack S

~~for each v in V do~~

```
DFS (G, S) :
    u.visited = true
    For each v ∈ G.Adj [u]
        if v.visited == false
            DFS (G, v)

init {
    For each u ∈ G
        u.visited = false
    For each u ∈ G
        DFS (G, u)
}
```

→ Recursive Algorithm for searching all the vertices of a graph or tree data structure

→ Traversal means visiting all graph nodes

→ Puts all vertex of graph in to two categories
→ visited
→ Not visited

Complexity of DFS : $O(V+E)$ where [Time complexity]

V – number of Node

E – number of Edges

: $O(V)$ where [Space complexity]

Applications of DFS

→ Finding the path → To test if Graph is biparite

→ Finding strongly connected componentss of Graph

→ For detecting cycles in graph

BFS (Bredth First Search)

Pseudo Code :

Create a queue Q

mark v as visited and put v into Q

While Q is non-empty :

remove the head u of Q

mark and enqueue all (unvisited) neighbours of u

Complexity of BFS [V – number of Nodes ,
E – number of Edges]

$O(V+E)$: Time Complexity

$O(V)$: Space Complexity

Algorithm : to Search for Goal State with minimal Cost :

1) Vector representing the current state <C, H, R, L>

2) Define a Goal State Example <6C, 3H, 1R, 6L>

3) Create a priority Queue [initialize with Start State & a cost of 0]

4) Create a "Closed Set" to keep track of explored States

5) While the open Set is not empty

→ Select the node with the lowest total cost $(f = g + h)$ where

→ g : is the cost of the start node to the current node

→ h : heuristic cost estimation from current state to goal state

→ if the selected node is Goal state, return the Solution path & cost

→ Otherwise, expand the selected node by generating its child node [possible next states]

Answer To Question 2

Current State : <3C, 2H, 0R, 3L>

Goal State : <6C, 3H, 1R, 6L>

We can use A* along with BFS, DFS to get the most optimal Solution.

Heuristic selection :

1) The difference between Cost of Goal state & Current state

2) The difference between the number of entities in the Goal state & current State

The algorithm will find the fastest & most economical Solution to achive the Goal state while considering the cost of Construction

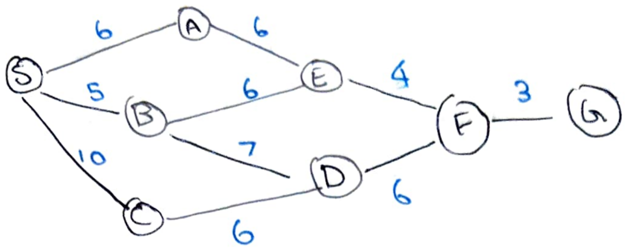
Answer Question 3

UCS : Uniform Cost Search

'S' → 'G'

Heuristic Values

S	A	B	C	D	E	F	G
16	10	12	4	2	4	1	0



- UCS is a varient of Dijkstra's Algorithm

Pseudo Code

function UCS (Graph , Start , target) :

Add the starting node to the opened list

The node has zero distance value to it self.

~~While True~~

Visited ← [ ]

actions ← [ ]

fringe ← Priority Queue ( )

fringe . add (Cost, (start_state, action, actions)

While fringe is not empty :

Curr Path ← fetch element from Queue

Curr State ← Curr path [0]

action ← Curr Path [1]

actions ← Curr path [2]

if (reached goal state) :

Return actions

End IF

```
IF CurrState is not visited :
    visited . append (CurrState)
    branches ← get_children (CurrState)
    for route in branches
        IF route is not visited :
            moves ← list (actions)
            moves . append (route)
            fringe . add ((route [state], ....
            route [action], moves) ....
            Calc_Cost (moves)))
        END IF
    END IF
```

a) UCS for $S \rightarrow G$

Flow	Visited
S	S
S — A	S A

Answer
Question 4

To measure 4 L water using 5 L & 3 L

a) Possible Actions

i) <Fill 5 liter , >

ii) <pour 5 liter into, Fill 3 liter >

iii) < 5liter lef with 2 liter >

~~iv)~~ (0,0)

(0, 3)

(5, 0)

(5, 3)

(3, 0)

(2, 3)

(3, 3)

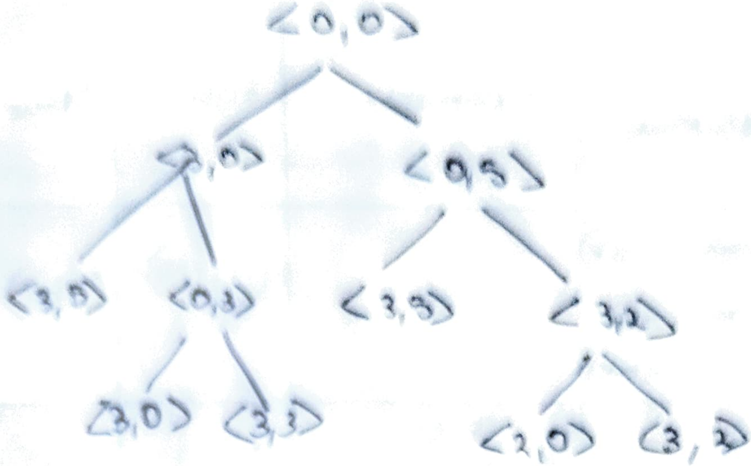
(5, 1)

(0, 1) ← now we have 1 liter here

(1, 0) pour it into 5 liter

(1, 3)

(4, 0) ← Now we have 4 liter in the 5 liter vessel



<0,0>
<0,5>
<3,0>
<2,0>

Search	Frontier	Completeness	Optimality	Time	Space
DFS	Stack	tree search - no (cycle) graph - yes (finite) No (infinite)	no	$O(b^m)$	$O(bm)$
BFS	Queue	Yes	no (except when all edge costs same)	$O(b^s)$	$O(b^s)$
Iterative Deepening	Stack	Yes (same as BFS)	no (same as BFS)	$O(b^s)$	$O(bs)$
UCS	heap-based PQ	Yes (assuming positive edge costs & $\epsilon > 0$)	Yes (assuming positive edge costs & $\epsilon > 0$)	$O(b^{C/\epsilon})$	$O(b^{C/\epsilon})$

b - branching factor

m - max depth of search

s - smallest depth of solution

C - cost of optimal solution

ϵ - minimum cost between 2 nodes

Answer
Question 5

DLS (Depth Limited Search)

1) Time complexity : depth limit "i"
branching factor "b"

Worst case $O(b^i)$ which means it can be exponential if "i" is deep.

2) Space complexity : $O(i)$

3) Completeness : Not complete.
It may fail to find a solution if depth limit "i" is too shallow.
It might terminate prematurely

4) Quality of Solution :

(Iterative Deepening Search) IDS

1) Time complexity : b – branching factor
d – depth of the goal

$O(b^d)$

2) Space complexity : $O(bd)$

3) Completeness : is Complete.
Guarantees finding a solution if one exists

4) Quality of Solution :