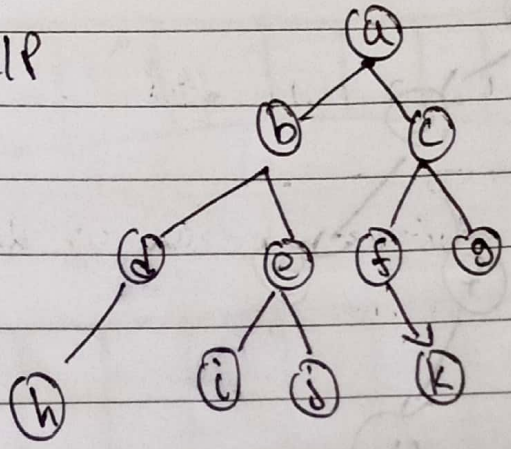
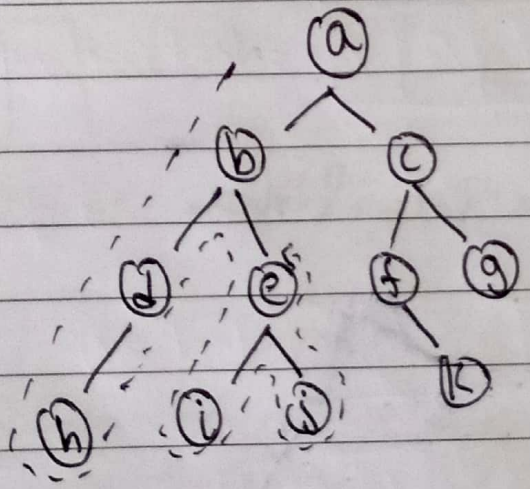
K. Barath
192011105
AI/CSA1771 Day-2

7. BFS Problem:

I/P



O/P



Solution:

⇒ The nodes are visited level by level

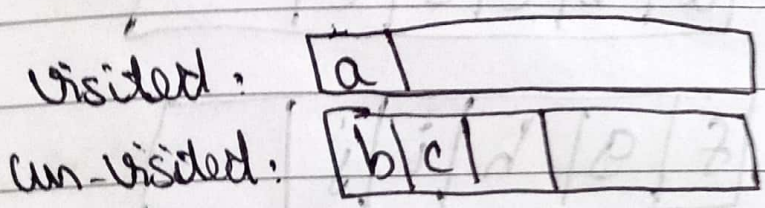
⇒ Two Queue are used.

1. visited Queue: keep track of the nodes that are visited

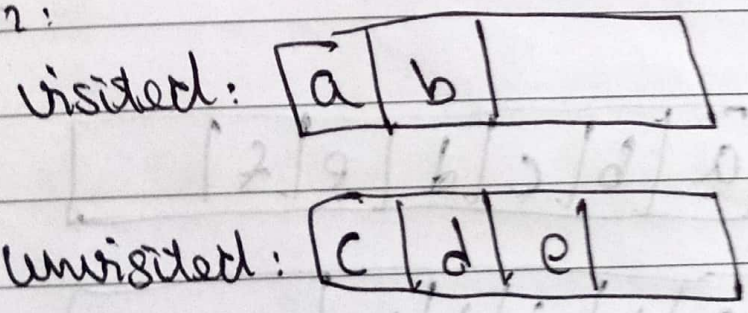
2. Unvisited Queue: keeps a track of the nodes that are yet to be visited.

=> Initially, both queues are empty.
=> The searching stops when -
    => The desired node is found
    => all nodes are visited

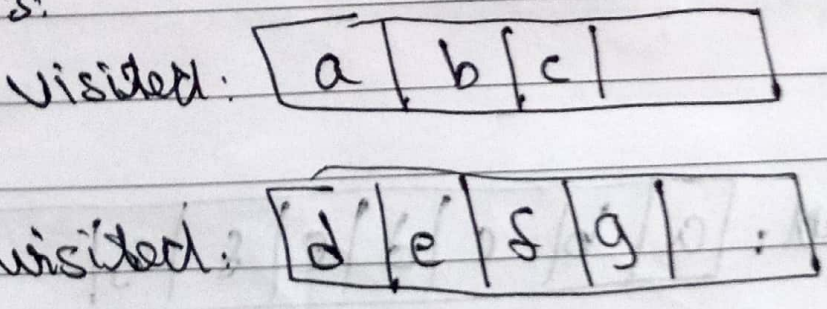
Step 1:



Step 2:



Step 3:



Step 4:

visited: | a | b | c | d |

unvisited: | e | f | g | h |

Step 5:

visited: | a | b | c | d | e |

unvisited: | f | g | h | i | j |

Step 6:

visited: | a | b | c | d | e | f |

unvisited: | g | h | i | j | k |

Step 7:

visited: | a | b | c | d | e | f | g |

unvisited: | h | i | j | k |

Step 8:

visited: | a | b | c | d | e | f | g | h |

unvisited: | i | j | k |

Step 9:

visited: | a | b | c | d | e | f | g | h | i |

unvisited: | j | k |

Step 10:

visited: | a | b | c | d | e | f | g | h | i | j |

unvisited: | k |

Step 11:

visited: | a | b | c | d | e | f | g | h | i | j | k |

unvisited:

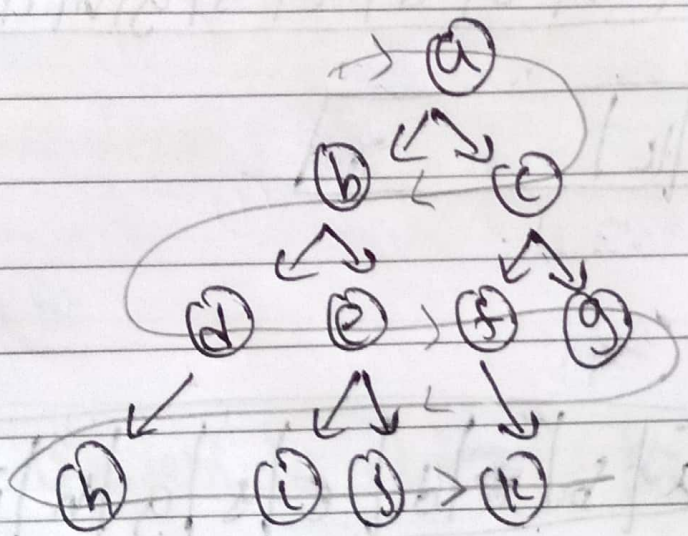
No node are presented in the unvisited queue

∴ Search stops here

BFS Traversal of the given tree →

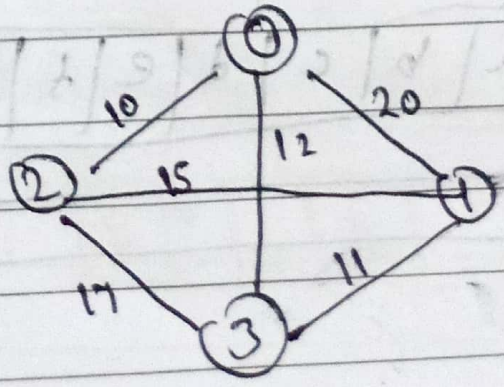
a → b → c → d → e → f → g → h → i → j → k

OUTPUT:

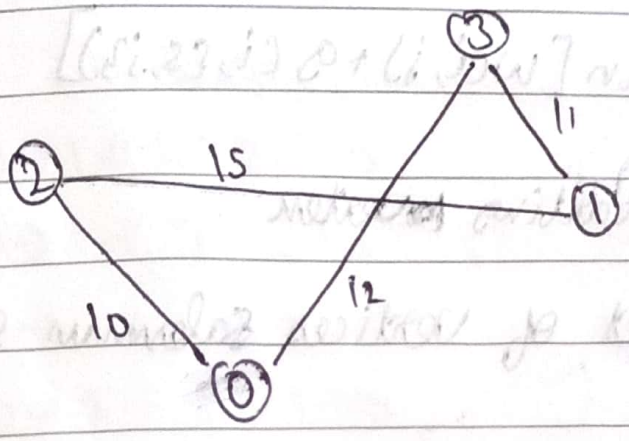


(9.) Solve the Travelling Sales Mans Problem using the following input and output.

Input:



OUTPUT:



Solution:

The Salesman has to visit every city only once and returns back to the city he/she started.

The most optimal that covers all the nodes must be found.

Adjacency Matrix:

	0	1	2	3
0	0	20	10	12
1	20	0	15	11
2	10	15	0	17
3	12	11	17	0

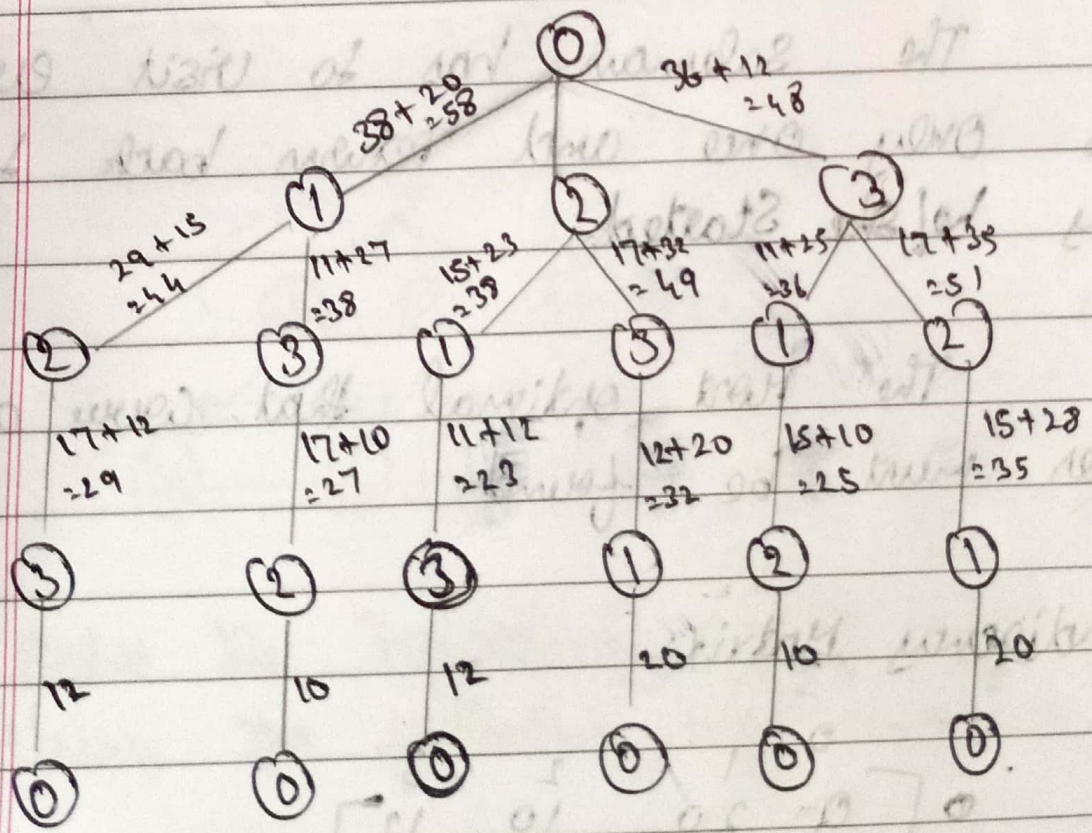
Formula used:

$$g(i,S) = \min[w(i,j) + g(i,\{S,j\})]$$

i: Starting vertex

S: set of vertices salesman should visit

Solution:



The Possible Paths are->

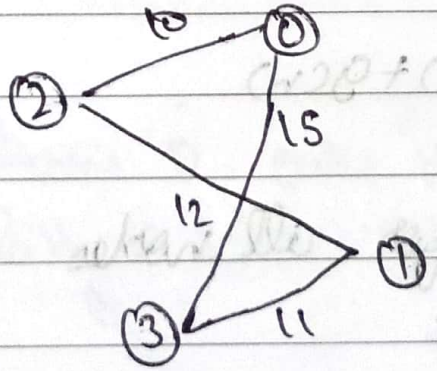
Path	Cost
0->1->2->3->0	58
0->1->3->2->0	64
0->2->1->3->0	48
0->2->3->1->0	59
0->3->1->2->0	48
0->3->2->1->0	64

The Most optimal Path:

0->2->1->3->0, Cost = 48

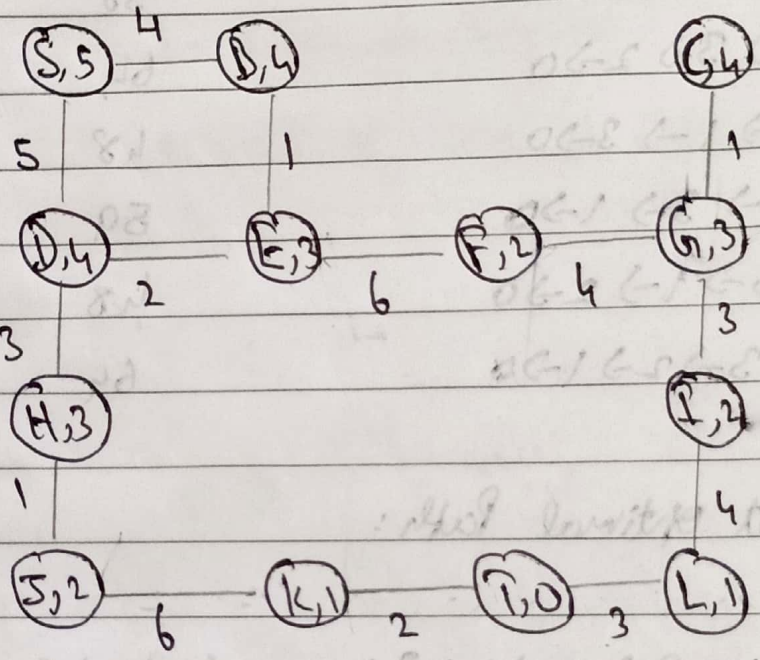
0->3->1->2->0, Cost = 48

OUTPUT:



10 Solve A* algorithm using following i/p & o/p

I/P



SOLUTION:

Formula used:-

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

Calculating $f(n)$ for all nodes

Step 1:

'S' is the source

$f(B) = 4+4 = 8$

$f(D) = 8+4 = 9$

Since $f(B) < f(D)$, we choose B

$\therefore$ Path : S → B

Step 2:

From B only F can be traversed

$\therefore$ Path ⇒ S → B → E

Step 3:

From E, D and F can be traversed

$f(D) = 2+4 = 6$

$f(F) = 6+2 = 8$

Since $f(D) < f(F)$, we choose 'D'

$\therefore$ Path ⇒ S → B → E → D

Step 4:

From D only H can be traversed

$\therefore$ Path → S → B → E → D → H

Step 5:

From H only J can be traversed

$\therefore$ Path ⇒ S → B → E → D → H → J

Step 6:

From J only K can be traversed

∴ Path ⇒ S→B→E→D→H→J→K

Step 7:

From K only 'T' can be Traversed

∴ Path ⇒ S→B→E→D→H→J→K→T

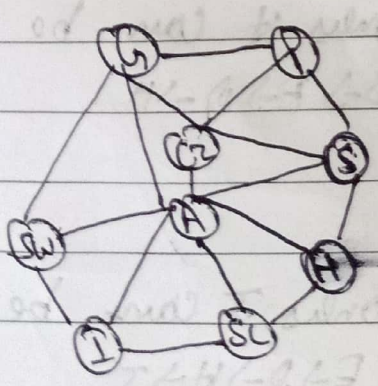
∴ The Path given by A* algorithm to reach to T from S is

S→B→E→D→H→J→K→T

Total Cost = 4+1+2+3+1+6+2

= 19.

11. Solve the Map coloring Problem using following i/p & o/p



G → Germany
C2 → Czech Republic
S → Slovakia
H → Hungary
SL → Slovenia
I → Italy
Sw → Switzerland

A → Austria
P → Poland

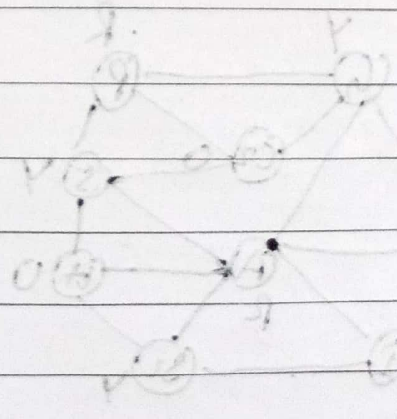
Solution:

Domain: { Red, Orange, Yellow, Purple }

Variables: { G, CZ, S, H, SL, I, SW, A, P }

Adjacency Matrix:

	G	CZ	S	H	SL	I	SW	A	P
G	0	1	0	0	0	0	1	1	1
CZ	1	0	1	0	0	0	0	1	1
S	0	1	0	1	0	0	0	1	1
H	0	0	1	0	1	0	0	1	0
SL	0	0	0	1	0	1	0	1	0
I	0	0	0	0	1	0	1	1	0
SW	1	0	0	0	0	1	0	1	0
A	1	1	1	1	1	1	1	0	0
P	1	1	1	0	0	0	0	0	0

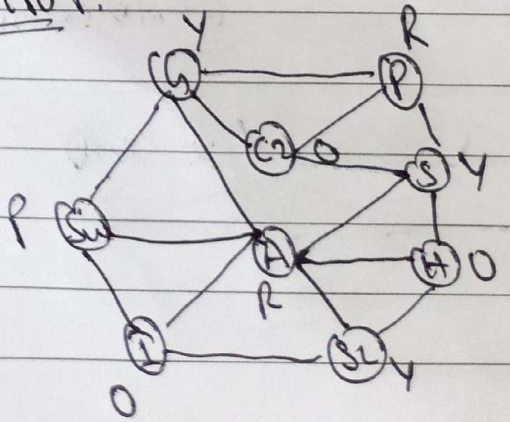


	Red	yellow	Orange	Purple
A	✓	–	–	–
G	–	✓	–	–
C2	–	–	✓	–
S	–	✓	–	–
H	–	–	✓	–
SL	–	✓	–	–
I	–	–	✓	–
P	✓	–	–	–
SW	–	–	–	✓

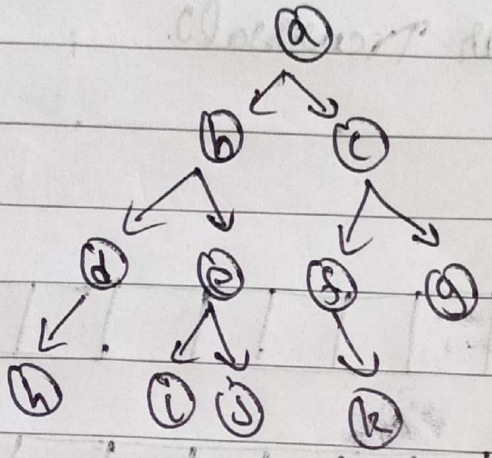
Conclusion:

Red: A and P
yellow: G, S & SL
Orange: C2, H & I
Purple: SW

OUTPUT:



8. Solve the DFS Problem using the following i/p & o/p.



=) Here, the nodes of a tree are visited from bottom to top (i.e; from leaves to root).

=) It can be done in 3 ways.

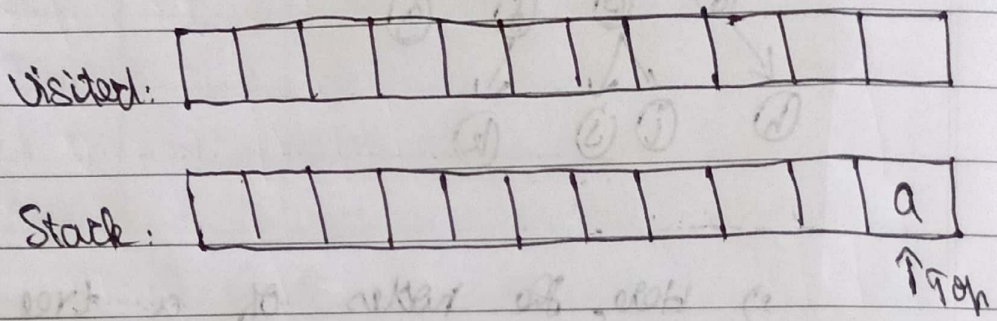
i) In-order : [left child - Root -> Right child]
ii) Pre-order : [Root -> left child -> Right child]
iii) Post order : [Left child -> Right child -> Root]

=) A stack is used. (to keep track to the next node to be visited)

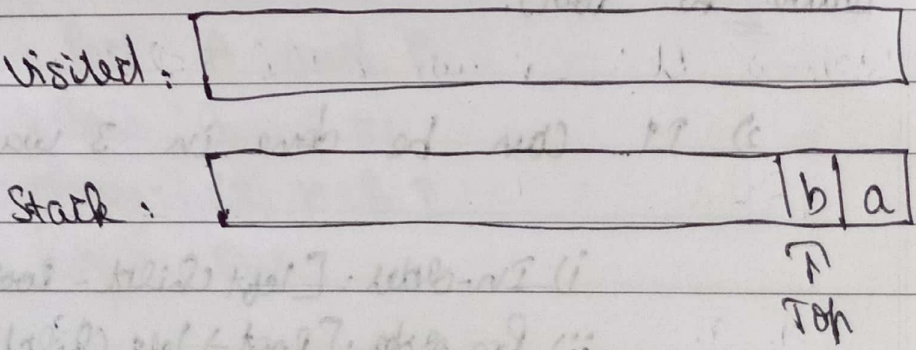
- an array 'VISITED' is used to keep track of the visited nodes.

Solution: (In-order Traversal).

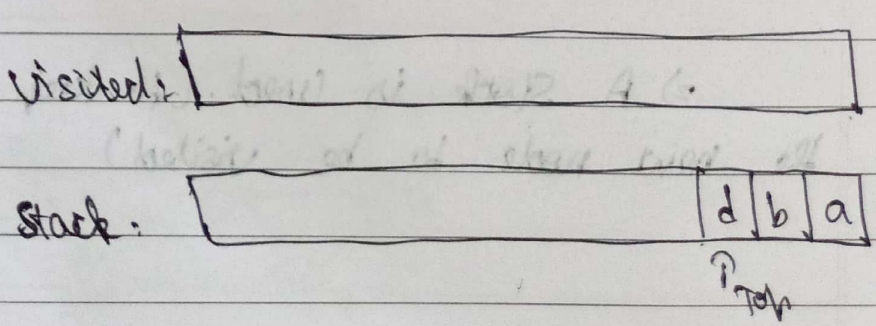
Step 1 => PUSH A



Step 2 => Push B



Step 3 => Push D



Step 4: ⇒ Push H

Visited:

Stack: h d b a

Top

Step 5 ⇒ Pop H (as H has no children)

Visited: h

Stack: d b a

Top

Step 6 ⇒ Pop D (D has no right-child)

Visited: h d

Stack: b a

Top

Step 7 ⇒ Pop B

Visited: h d b

Stack: a

Top

Step 8 => Push E

Visited: | h | d | b |

Stack: | e | a |

Top

Step 9 => Push I

Visited: | h | d | b |

Stack: | i | e | a |

Top

Step 10 => Pop I (I has no left or right child)

Visited: | h | d | b | i |

Stack: | e | a |

Top

Step 11 -> Pop E

Visited :

h	d	b	i	k

Stack :

	a

↑ Top

Step 12 2) Push J

Visited :

h	d	b	i	e

Stack :

j	a

Top

Step 13 : Pop J (J has no children)

Visited :

h	d	b	i	e	j

Stack :

a

Top

Step 14 : Pop A

Visited :

h	d	b	i	e	j	a

Stack : -1

Top

Step 15: Push C:

visited: | h | d | b | i | e | j | a |

stack: | c |

↑ Top

Step 16: Push F

visited: | h | d | b | i | e | j | a |

stack: | f | c |

↑ Top

Step 17: Pop

visited: | h | d | b | i | e | j | a | f |

stack: | | | c |

Top

Step 18: PUSH k

visited: | h | d | b | i | e | j | a | s |

stack: | k | c |

Top

Step 19: Pop k

visited: | h | d | b | i | e | j | a | s | k |

stack: | c |

Top

Step 20: Pop c

visited: | h | d | b | i | e | j | a | s | k | c |

stack:

Step 21: PUSH G

visited: | h | d | b | i | e | j | a | s | k | c |

stack: | g |

Step 22: Pop h

Visited: | h | d | b | i | e | j | a | f | k | c | g |

Stack: [ ]

No more nodes are left to be traversed
∴ The search stops here.

Order of Traversal:

h → d → b → i → e → j → a → f → k → c → g.

Output: