SOLVED QUESTION BANK

[Sequence given as per syllabus]

INTRODUCTION

Graphs and trees required searching for a node with some specific properly. When the search requires necessarily visiting every node in the structure, then it is called a traversal.

To solve the problems on graphs various approaches are used. One among them is based on the notion of systematically visiting all the vertices and edges of a graph.

In order to traverse a graph we have to process every node exactly once. While processing a particular node we should be very careful as various path lead to that node. So, we have to process the node exactly once.

BASIC TRAVERSAL AND SEARCH TECHNIQUES

Q.1. What are the basic traversal and search techniques?

Ans.

- One of the most fundamental graph problem is to traverse every edge and vertex in a graph.

Application of traversing include :

(i) Counting the number of edges.

(ii) Identifying connected components of a graph.

(iii) Printing out the contents of each edge and vertex.

- For efficiency we must make sure we visit each edge at most twice.

Marking vertices :

- The idea in graph traversal is that we must mark each vertex when we first visit it and keep track of what have not yet completely explored.
- For each vertex, we can maintain two flags :

(1) **Discovered :** Have we ever encountered this vertex before?

(2) **Completely explored :** Have we finished exploring this vertex yet?

- To completely explore a vertex, we look at each edge going out of it.
- For each edge which goes to an undiscovered vertex, we mark it discovered and add it to the list of work to do.

Traversal orders :

- The order we explored the vertices depends upon what kind of data structure is to be used.

(1) **Queue :**

- By storing the vertices in a first in first out queue, we explore the oldest unexplored vertices first.
- Thus our explorations radiate out slowly from the starting vertex, this is called as "Breadth first search".

(2) **Stack :**

- By storing the vertices in a last in first out (LIFO) stack we explore the vertices by searching along a path, constantly visiting a new neighbour if one is available and previously discovered vertices.
- Thus our explorations quickly wonder away from our starting point, defining a so called "Depth first search".
- The three possible colors of each node reflect if it is :

(a) Unvisited-white.

(b) Visited but-unexplored-grey.

(c) Completely explored-black.

- There are two methods for traversals of graph :

(1) Breadth first search.

(2) Depth first search.

BREADTH FIRST SEARCH

Q.2. Give an algorithm to obtain a BFS tree. Discuss its complexity when the graph is implemented using adjacency list.

CT : S-09(7M)

OR Give an algorithm for BFS with example.

CT : S-11(3M), S-14(6M), W-08(4M)

OR What is breadth first search?

Ans. Breadth first search :

- Breadth first search traverse a connected compo-nent of a graph and defines a spanning tree.
- Breadth first search algorithm used in :

(a) Prim's MST algorithm.

(b) Dijkstra's single source shortest path algorithm.

- BFS starts at a given vertex which is at level 0.
- In the first stage, we visit all vertices at level 1.
- In second stage, we visit all vertices at level second. The new vertices, which are adjacent to level 1 vertices and so on.
- The BFS traversal terminates when every vertex has been visited.

Algorithm :

BFS (G, S)

(1) For each vertex u ∈ v [G] {s} do

(2) color [u] ← WHITE

(3) d [u] ← ∞

(4) P [u] = NIL i.e. parent in the BFS tree

(5) color [s] = gray

(6) d [s] ← 0

(7) Q = {s}

(8) while Q ≠ ϕ do

(9) u = head [Q]

(10) For each v ∈ Adj[u] do

(11) if color [u] = white then

(12) color [v] = Gray

(13) d [v] = d [u] + 1

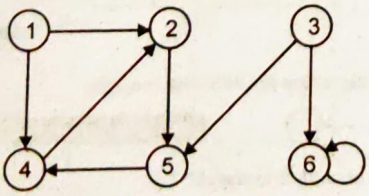
(14) P [v] = u

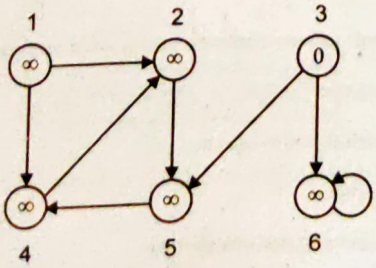
(15) Enqueue [Q, v]

(16) Dequeue [Q]

(17) color [u] = black

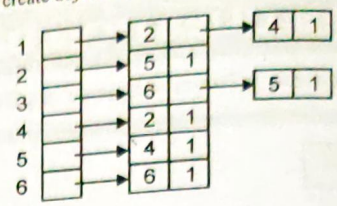
Example : Consider the graph G in figure. We describe the whole process of breadth first search using vertex 3 as the source.





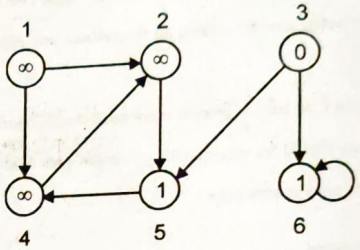
Q | 3 |
0

First, we create adjacency list representation,

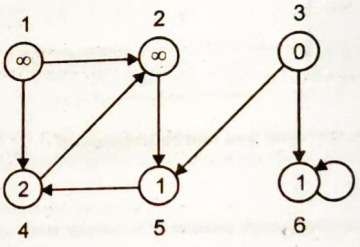


So, adj [4] = {6, 5} so, color [5] = GRAY,
color [6] = GRAY
And d [6] = 1 d [5] = 1
P [6] ← 3 P [5] ← 3

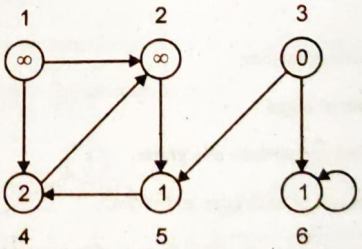
Now,



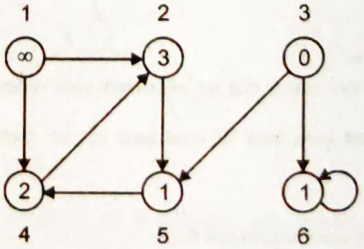
Q | 5 | 6 |
1 1



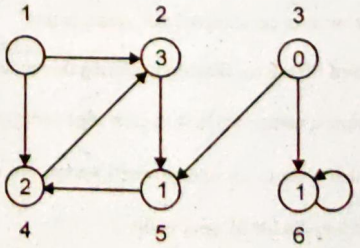
Q | 6 | 4 |
1 2



Q | 4 | |
2



Q | 2 | |
3



Q | ϕ | |
1

Thus, vertex 1 cannot be reachable from source.

Complexity : The total running time of BFS is O (V + E).

If adjacency list is used the complexity O (n +e)

Q.3. Give the applications of breadth first search.

Ans. Applications of BFS are as follows :

(i) Testing whether graph is connected.

(ii) Computing a spanning forest of graph.

(iii) Computing for every vertex in graph a path with the minimum number of edges between start vertex and current vertex or reporting that no such path exists.

DEPTH FIRST SEARCH

Q.4. Give the algorithm for depth first search. Explain with example.

CT : W-06,10(5M)

OR Another way to search a graph is to explore the vertex most recently added to the list of unexplored vertices. Write an algorithm for such graph search. What is the time and space requirement of the algorithm? **CT : W-07,11(7M)**

OR Give an algorithm to obtain the DFS tree.

CT : W-09(7M), W-13(3M)

Ans. Depth first search :

- In depth first search edges are explored out of the most recently discovered vertex v that still has unexplored edges leaving the vertex from which v was discovered.
- This process continues until we have discovered all the vertices that are reachable from the original source vertex.
- If any undiscovered vertices remain then one of them is selected as a new source and the search is repeated from that source. This entire process is repeated until all vertices are discovered.
- DFS uses the backtracking technique.

Algorithm :

BFS (G, S)

(1) For each vertex u ∈ v [G]

(2) do color [u] = white

(3) parent [u] = NIL

(4) time = 0

(5) for each vertex u ∈ v [G]

(6) do if color [u] = white

(7) then DFS-VIST [u].

DFS-VISIT [u]

(1) color [u] = gray

(2) discover [u] = time

(3) time = time + 1

(4) for each v ∈ adj [u]

(5) if color [v] = white

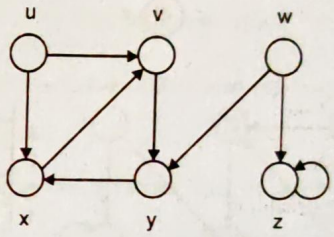
(6) then parent [v] = u

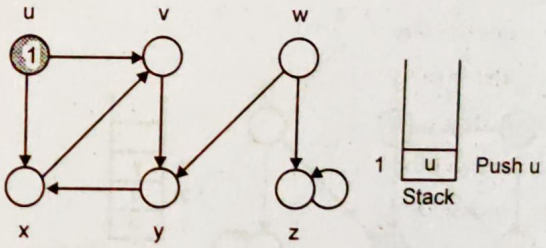
(7) DFS-VISIT [u].

(8) color [u] = black

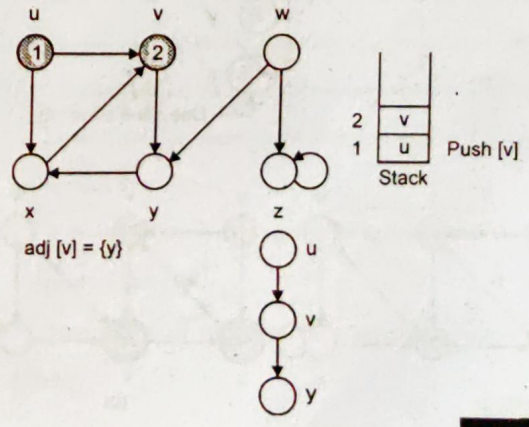
(9) finish [u] where time = time + 1

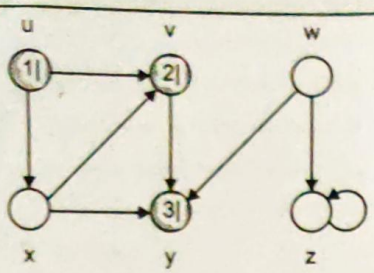
Example : Consider the following graph G. Depth First search is done as follows :

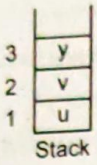




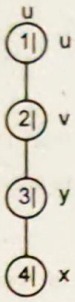
adj [u] = [v, x]

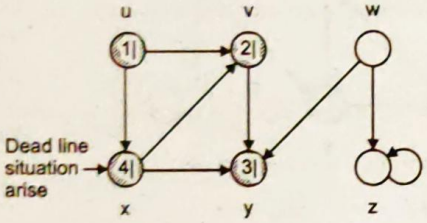


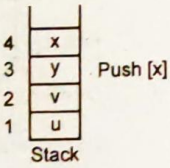


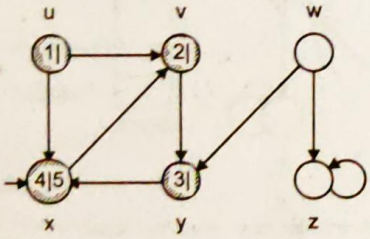


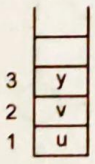
adj [y] = {x}

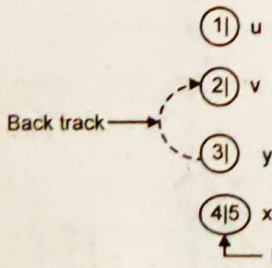


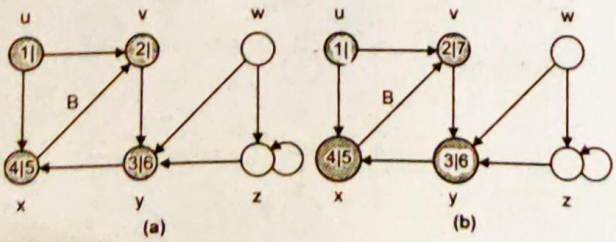


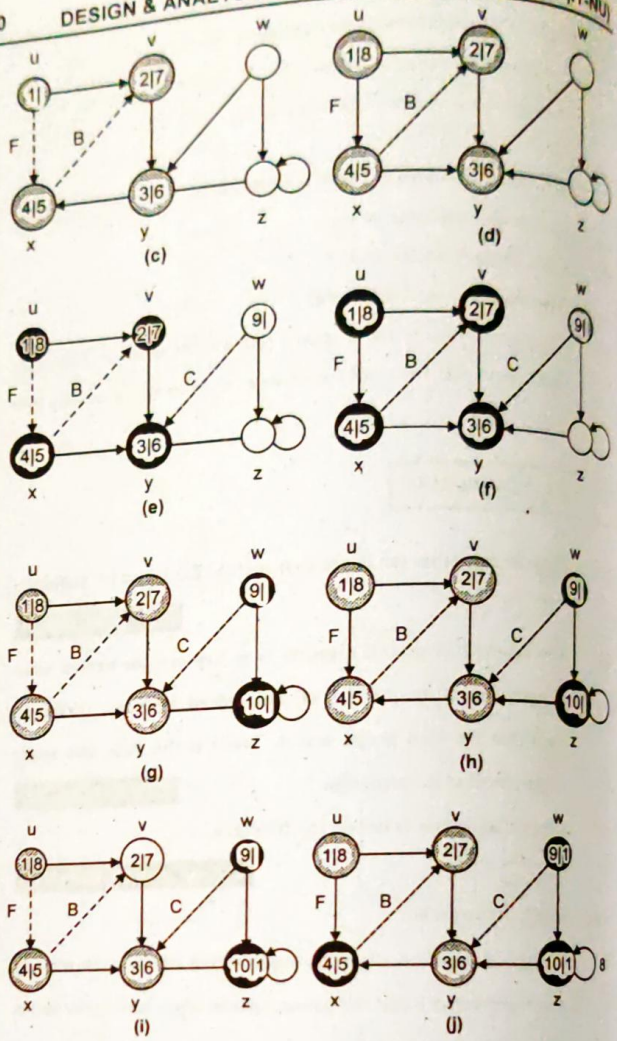












Solid Edge = discovery or tree edge

Dashed Edge = back Edge

○ → white vertex

○ → Gray vertex

● → Black vertex

Complexity :

The running time of DSF is $\theta (V + E)$.

Q.5. Give the applications of depth first search.

Ans. Application of DFS algorithm are as follows :

(i) Testing whether the graph is connected.

(ii) Computing a spanning forest of graph.

(iii) Computing a path between two vertices of graph or equivalently reporting that no such path exist.

(iv) Computing a cycle in graph or equivalently reporting that no such cycle exists.

Q.6. With respect to DFS, define Tree edge, Back edge, Forward edge, Cross edge. In DFS of an undirected graph G. Prove that every edge of G is either a tree edge or a back edge.

CS : S-13(6M)

Ans.

(1) Tree edge :

Tree edge in depth first forest G. Edge (u, v) is a free edge if v was first discovered by exploring edge (u, v).

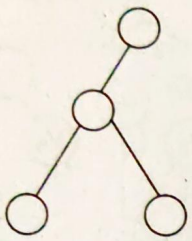


Fig. (a) Tree edge

(2) Back edge :

Back edges are those edges (u, v) connecting a vertex u to an ancestor v in a depth first tree, self loops, which may occur in directed graphs are considered to be back edges.

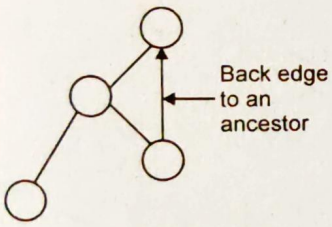


Fig. (b) Back edge

(3) Forward edge :

Forward edges are those non tree edges (u, v) connecting a vertex u to a descendent v in a depth first tree.

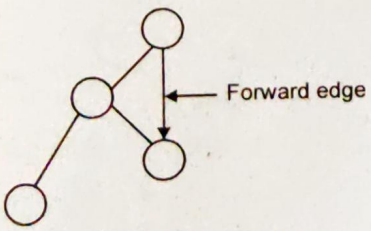


Fig. (c) Forward edge

(4) Cross edge :

Cross edges are other edges. They can go between vertices in the same depth first tree, as long as one vertex is not an ancestor of the other, or they can go between vertices in different depth first trees.

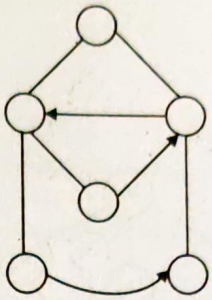
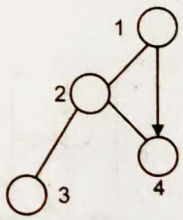


Fig. (d) Cross edges

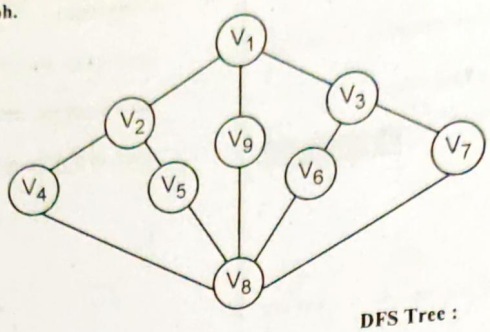
- In a DFS of an undirected graph, every edge is either a tree edge or a back edge.
- Suppose we have a forward edge. We should have encountered (4, 1) when expanding 4, so this is a back edge.



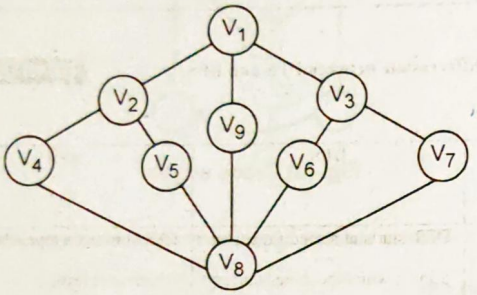
Proof :

- Let (u, v) be an arbitrary edge of G, and suppose without loss of generality that u.d < v.d.
- Then the search must discover and finishes v before it finishes u.
- Since v is on u's adjaceney list. If the first time that the search explores edge (u, v) it is in the direction from u to v, then v is undiscovered until that time, for otherwise the search would have explored this edge already in the direction from v to u. Thus (u, v) becomes a tree edge.
- If the search explores (u, v) first in the direction from v to u, then (u, v) is a back edge.
- Hence proved that every edge of G is either a tree edge or a back edge.

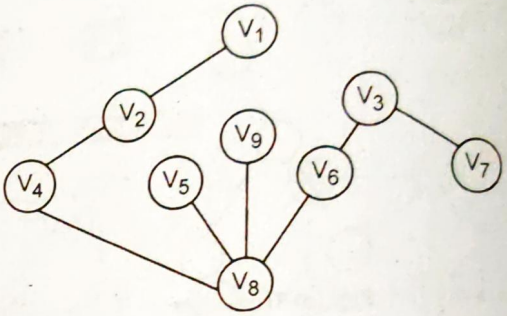
Q.7. Draw DFS and BFS for following graph.



Ans. The given graph is :

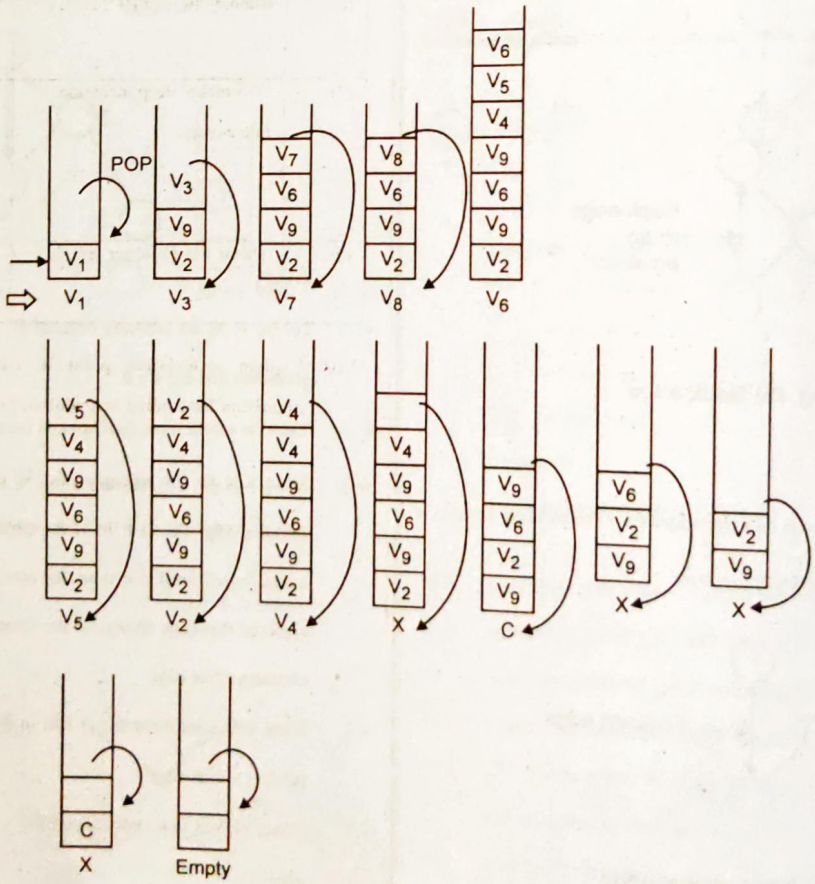


DFS Tree :

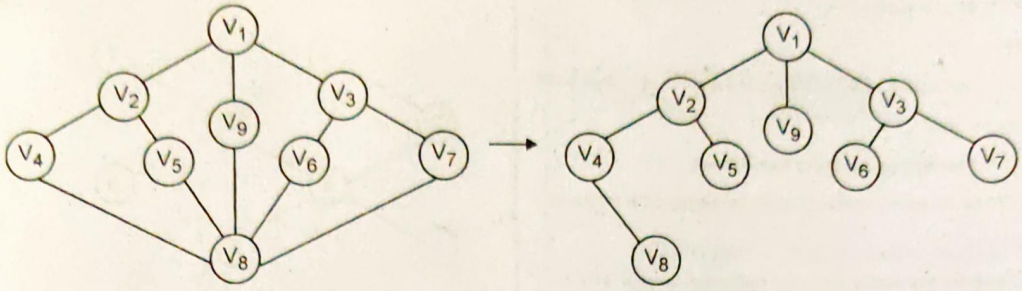


Sequence is :

$V_1 - V_3 - V_7 - V_8 - V_6 - V_5 - V_2 - V_4 - V_9$

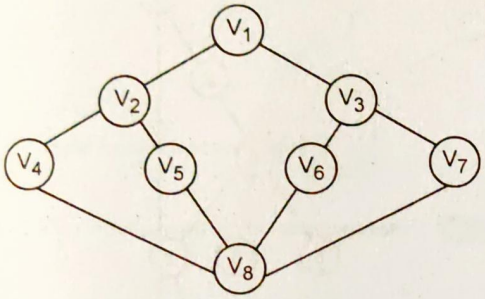


Breadth first search tree :

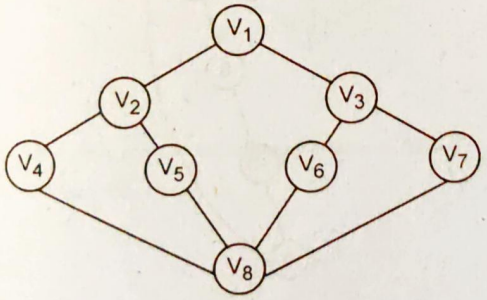


Q.8. Obtain the DFS tree for the given graph.

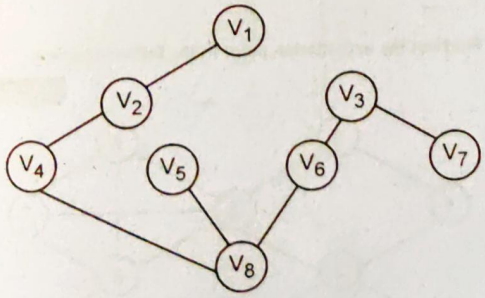
CT : W-09,(6M), W-13(4M)



Ans. The given graph is :



DFS Tree :



Sequence is :

$V_1, -V_2, -V_4, -V_8, -V_5, -V_6, -V_3, -V_7$

Q.9. Differentiate between DFS and BFS. **CT : W-12(5M)**

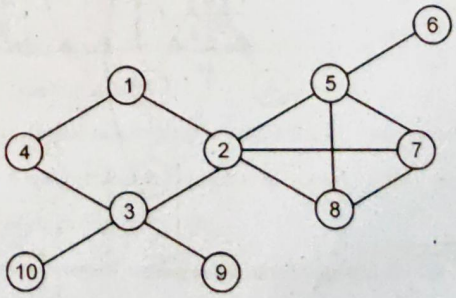
Ans.

Sr. No.	DFS	BFS
(1)	DFS starts at a specific vertex S in G, which becomes current vertex.	BFS starts at a given vertex. which is at level 0.
(2)	DFS uses stack.	BFS uses queue.
(3)	Running time of DFS is O(V+ E).	Running time of DFS is θ (V + E)
(4)	DFS makes deep incursion into a graph.	BFS makes sure to visit vertices in increasing order of their distance from the starting point.
(5)	This is not a broader search.	This is broader, shallower search.

Q.10. Explain articulation point in DFS and give the complete algorithm for finding articulation point of undirected graph.

CS : W-13(7M)

OR Find out the articulation point in the following graph. Explain the procedure to find the articulation point. **CS : W-12(6M)**



Ans. Articulation point :

A node of given graph on which maximum edges and vertices of the graph are connected is known as articulation point.

Algorithm :

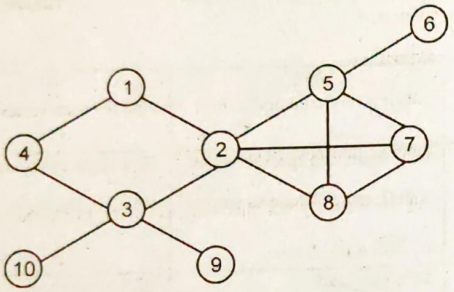
Step 1 : For a given graph perform DFS and find out the sequence in which the vertices are visited.

Step 2 : For a sequence generated in step draw DFS tree.

Step 3 : Find out the edges present in the graph but not present in the tree (n – 1) i.e. back edge,

Step 4 : Find a edge representing connecting child-parent-ancestor in the tree. The parent node is called as articulation point.

Example : Consider the following graph :

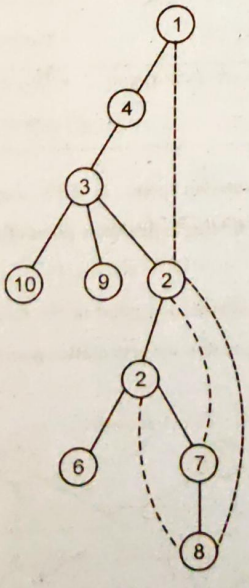


Assume vertex-1 as source vertex

Sequence generated using DFS is :

1 - 4 - 3 - 10 - 9 - 2 - 5 - 6 - 7 - 8

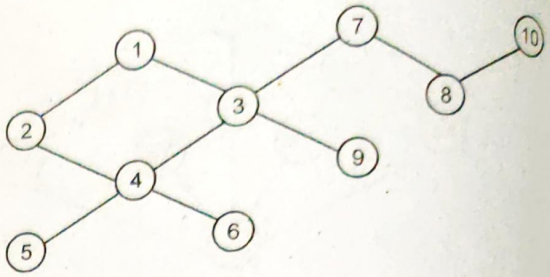
DFS tree :



1 - 2 - 7
1 - 2 - 8
→ Vertex 2 articulation point.

Q.11. Explain articulation point for the graph given below.

CT : S-14(6M)

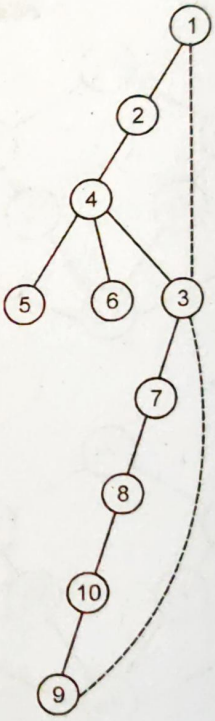


Ans. Assume vertex-1 as source vertex.

Sequence generated using DFS is :

1 - 2 - 4 - 5 - 6 - 3 - 7 - 8 - 10 - 9

DFS Tree :

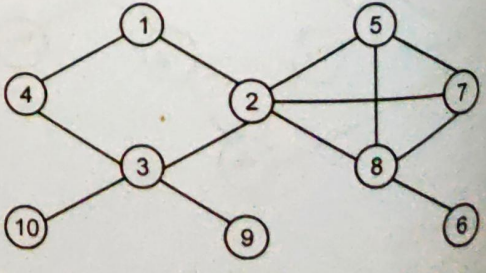


Vertex 3 is articulation point

1 - 3 - 9

Q.12. Find out the articulation point in the following graph.

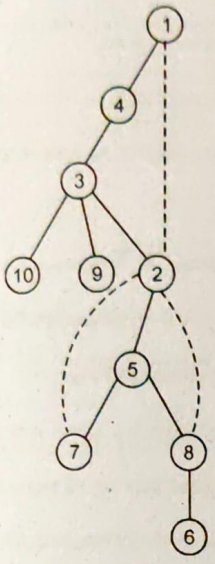
CS : S-14(6M)



Ans. Consider source vertex as vertex -1

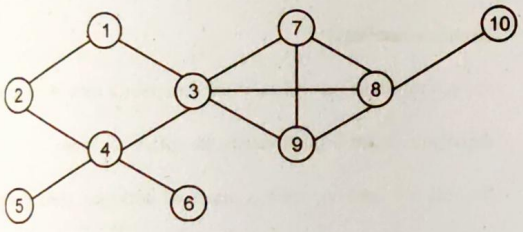
Sequence generated using DFS :

1-4-3-10-9-2-5-8-7-6



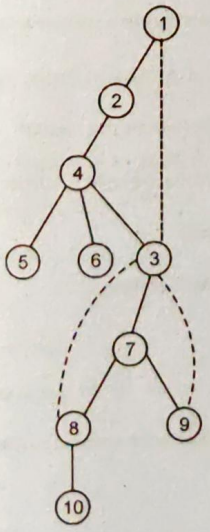
Articulation point is vertex 2 i.e. 1-2-7, 1-2-8

Q.13. **Find articulation point for the following graph :** *CS : W-11(5M)*



Ans. Articulation point for the following graph is as follows :

Assume vertex 1 as source.



Articulation point is Vertex 3 i.e. 1-3-8, 1-3-9

CONNECTED COMPONENTS

Q.14. What is connected component? Explain

Ans. Connected components :

- A directed graph is strongly connected if and only if there is a directed path between any two vertices.
- The strongly connected components of a graph is a partition of the vertices in to subsets(maximal) such that each subset is strongly connected.

Algorithm :

(1) Call DFS (G) to compute finishing time for each vertex.

(2) Compute transpose of G i.e. G^T

(3) Call DFS (G^T) but this time consider the vertices in order of decreasing finish time.

(4) Out the vertices of each tree in DFS-forest.

Q.15. Write an algorithm to find out whether the graph has any unconnected component. *CT : S-07(7M)*

Ans. Unconnected components (G) :

(1) For each vertex v ∈ v [G]

(2) do MAKE-SET (v)

(3) for each edge (u, v) ∈ E [G]

(4) do if FIND-SET (u) ≠ FIND-SET (v)

(5) then UNION (u, v)

SAME COMPONENT (u, v)

(1) if FIND.SET (u) = FIND SET (v)

(2) then return TRUE

(3) else return FALSE

Q.16. What do you mean by topological sort?

Ans. Topological sort :

- A directed acyclic graph is a directed graph with no directed cycles.
- A topological sort of a graph is an ordering on the vertices so that all edges go from left to right.
- A topological sort is an ordering such that any directed path in DAG, G traverses vertices in increasing order.

Algorithm :

(1) For each vertex find the finish time by calling DFS (G)

(2) Insert each finish vertex into the front of a linked list.

(3) Return the linked list.

Complexity : Total running time of topological sort is $\theta(V+E)$.

Q.17. What is biconnected graph? *CT : S-14(2M), CS : W-11(2M)*

Ans. Biconnected graph :

- A biconnected graph has edges that are "two way" i.e. we can go from one vertex to another and vice-versa.
- A biconnected graph is a connected and "nonseparable" graph meaning that if any vertex were to be removed the graph will remain connected.
- Biconnected graph has no articulation vertices.

Diagram :

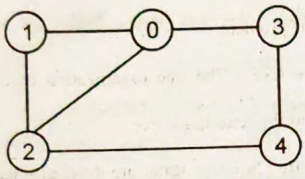


Fig. Bi-connected graph

BACKTRACKING

Q.18. What is backtracking? Explain the application in which backtracking principle can be used to design solution. *CS : S-11(6M)*

OR Explain the backtracking method for designing algorithm. Give a generalized recursive algo-rithm. *CT : W-09(7M)*

OR Give the general schema for backtracking algorithm and explain in brief. *CT : S-07(7M), W-07(6M)*

OR What types of problem can be solved using principle of backtracking? Also discuss backtracking in detail with suitable example. *CT : S-06(13M)*

OR Discuss with suitable example the theory of backtracking algorithm. Your answer should clearly bring out the terms involved. *CT : W-06, S-08(7M)*

Ans. Backtracking :

- Backtracking is a systematic way to go through all possible configuration of a search space.
- The basic idea of backtracking is to build up a vector one component at a time and to test whether the vector being formed has any chance of success.
- The major advantage or backtracking algorithm is that if it is realized that the partial vector generated does not lead to an optimal solution then that vector may be ignored.
- Backtracking is a depth first search with some bounding function. All solution using backtracking are required to satisfy a complex set of constraints. The constraint may be explicit or implicit.

Explicit constraint :

Explicit constraints are rules, which restrict each vector element to be chosen from the given set.

Implicit constraints :

- Implicit constraints are rules, which determine which of the tuples in the solution space, actually satisfy the criteria function.
- Suppose we have to make a series of decisions, among various choices, where we don't have enough information to know what to choose and each decision leads to a new set of choices. Some sequence of choices may be a solution to our problem.
- Backtracking is a method of trying out various sequences of decisions, until we find one that "works".
- The time required for generating solution is not fixed because the process is not fixed.

Application of Backtracking :

(1) Game Theory :

- Recursize maze is one of the good example for backtracking algorithm. Recursize maze is one of the most available solutions for solving maze.

- Maze is an area surrounded by walls, in between we have a path from a starting position to ending position.
- We have to start from the starting point and travel towards the ending point.

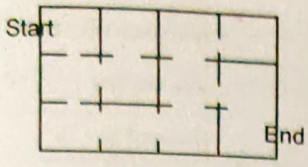


Fig. Maze

- In maze, we have to travel from starting point to ending point. The problem is to choose the path. If we find any dead-end before ending point we have to backtrack and change the direction.
- We have to continue "move and backtrack" until we reach the ending point.

Algorithm :

Backtrack (a, k)

(i) If a is solution

(ii) then print (a)

(iii) else

(iv) $k \leftarrow k + 1$

(v) compute S_k

(vi) while $S_k \neq 0$ do

(vii) $a_k \leftarrow$ an element in S_k

(viii) $S_k \leftarrow S_k - a_k$

(ix) Backtrack (a, k)

(2) Backtracking can be easily used to iterate through all subsets or permutations of a set.

(3) Backtracking ensures correctness by enumerating all possibilities.

Q.19. Define the following terms as applied to design of algorithm using backtracking. *CT : W-08(13M)*

(i) Problem state.

(ii) State space.

(iii) State space tree.

(iv) Solution state.

(v) Answer state.

(vi) Dead node.

(vii) Live node.

(viii) E-node.

(ix) Bounding functions.

(x) Explicit constraints.

(xi) Implicit constraints.

Ans.

(i) **Problem state :** Each node in this tree defines a problem state as shown in the fig.

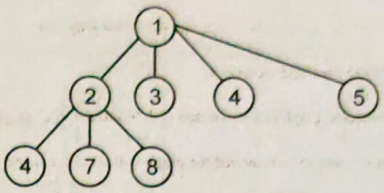


Fig. Problem state

(ii) **State space :** All paths from the root to other nodes define the state space of the problem.

(iii) **State space tree :** The tree organization of the solution space is referred to as the state space tree.

(iv) **Solution state :** Solution states are those problem states S for which the path from the root to S defines a tuple in the solution space.

(v) **Answer states :** Answer states are those solution states S for which the path from the root S defines a tuple that is member of the set of solutions.

(vi) **Dead node :** A dead node is a generated node which is not to be expanded further or all of whose children have been generated.

(vii) **Live node :** A node which has been generated and all of whose children have not yet been generated is called Live node.

(viii) **E-node :** The live node whose children are currently being generated is called E-node.

(ix) **Bounding functions :** The functions that are used to kill live nodes without generating all their children are known as bounding functions.

(x) **Explicit constraints :** The rules that restrict each X_i to take on values only from a given set is known as explicit constraints.

(xi) **Implicit constraints :** The rules that determines which of the tuples in the solution space of I satisfy the criterion function is known as implicit constraints.

Q.20. Explain the difference between explicit constraint and implicit constraint. **CT : S-14(2M)**

Ans.

Sr. no.	Explicit constraint	Implicit constraint
(1)	Explicit constraints are rules, which restrict each vector element to be chosen from the given set.	Implicit constraints are rules, which determine which of the tuples in the solution space actually satisfy the creation function.
(2)	**Example :** x = 1.....9 x y y = 1.....9 ↓ ↓ Rule : 6 4 (1) x + y = 10 4 6 (2) x * y = 24 If x = 1 and y = 9 only rule (1) satisfy . If x = 4 and y = 6 or x = 6 and y = 4 satisfy both the rules.	**Example :** x = 1.....9, y = 1.....4 } explicit x + y = 10, x * y = 24 } criteria x y 6 4 4 6 Implicit is x = 6 and y = 4 or x = 4 and y = 6.

Q.21. What is the significance of backtracking?

Ans. Significance of backtracking :

- Backtracking is an important tool for solving constraint satisfaction problem.
- Backtracking can be applied only for problems which admit the concept of a 'partial candidate solution' and a relatively quick test of whether it can possibly be completed to a valid solution.
- Backtracking algorithm enumerates a set of partial candidates that in principle could be completed in various ways. It traverses search tree recursively.

Q.22. Give the recursive and non-recursive skeleton of backtracking algorithm. **CT : W-06(6M), S-08(7M)**

Ans. Recursive algorithm :

```
Algorithm Backtrack (k)
{
    for (each x [k] ∈ T (x [1], ..... x [k – 1])        do
    {
        if (Bk (x [1], x [2] ....x [k] ≠ 0) then
        {
            if (x [1], x[2], ..... x[k] is a path
                                        to answer node)
                then write (x [1 : k]) ;
            if (k < n) then
            Backtrack (k + 1);
        }
    }
}
```

Iterative Backtracking :

```
Algorithm Backtrack (n)
{
    k : = 1;
    while (k ≠ 0) do
    { if (there remains an untied x [k] ∈ T (x [1],
    x[2] ..... x[k–1]) and Bk (x[1] .... x[k] is true) then
        { if ( x [1] ..... x[k] is a path to an answer node)
        write (x [ 1 : k]);
        k : = k + 1;
        }
        else
            k = k – 1;
    }
}
```

Q.23. Distinguish between iterative and recursive backtracking algorithm. **CT : S-11(4M)**

Ans.

Sr. no.	Recursive algorithm	Iterative algorithm
(1)	Recursive backtracking is a process of executing certain set of instructions repeatedly by calling self method repeatedly.	Iterative backtracking is a process of execute certain set of instruction repeatedly without calling self method.

(2)	This algorithm is less efficient.	This algorithm is more efficient because of their execution speed.
(3)	More memory is utilized.	Less memory is utilized.
(4)	Quite complex to implement.	Less complex to implement.

Q.24. Explain branch and bound versus back-tracking search.

CT : W-11(6M)

Ans.

- In backtracking algorithm a preorder depth first search of the state space tree is done.
- A branch and bound algorithm is not restricted to a depth-first search of the state space tree, it may use a DFS if it need so.
- A branch and bound technique generates nodes according to several rules, the most natural one is the best first rule where in the most promising node at the present moment is expanded first.
- The backtracking algorithm is effective for decision problem but it is not designed for optimization problem.
- A branch and bound technique is designed for optimization problem.

Q.25. Write down an algorithm using backtracking to enumerates all sequences of states. Starting from initial state go up to final state if on consuming each and every symbol of input string 'w'.

CT : S-07(13M)

Ans. BACKTRACK (n)

(1) Local integer x [0 : n – 1], K;

(2) k ← 1;

(3) while k > 0 do

(4) if there remains untried x [k] such that x [k] in T (x [1], ... , x [k – 1]) and B_k (x [1], ... x [k–1]), ... is true then

(5) if (x [1], ..., x [k]) is a path to the answer node then

(6) print (x [1], ... x [k]);

(7) end

(8) k ← k + 1; [consider the next set]

(9) else

(10) k ← k – 1; [backtrack to previous set]

(11) end

(12) end.

RBACKTRACK (K)

(1) global integer [x [0 : n – 1], n ;

(2) k ← 1;

(3) for each x [k] such that x [k] in T (x [1], x [k – 1]) and B_k (x [1], ..., x [k – 1]) is true do.

(4) if [x [1], ..., x [k]) is a path to the answer node then

(5) print (x [1], ..., x [k]);

(6) end

(7) RBACKTRACK (K + 1);

(8) end

4 QUEEN'S PROBLEM

Q.26. Discuss the four queen problem and give its algorithm using backtracking method. **CT : S-09(7M), S-13, W-13(6M)**

OR State 4-Queen problem and write its backtracking algorithm. Discuss its time complexity. **CT : S-11(9M)**

OR Explain how backtracking technique can be applied to solve 4-queen problems. **CT : W-10(6M), W-12(7M)**

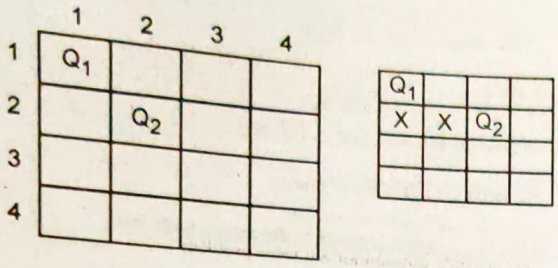
Ans. **4-Queen problem :**

- Given a 4 × 4 chess board and 4 queens to be placed on 4 × 4 chess board so that no two queens are on the same row, column or diagonal.
- Here the solution to the problem is 4-tuple where queens are placed with the above constraints. Now we show the step by step solution of 4 queen problem.

Step 1 : Placed first queen Q_1 in the first column.

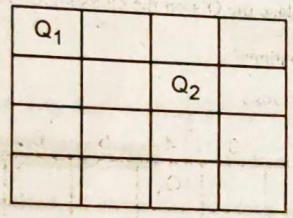
	1	2	3	4
1	Q_1			
2				
3				
4				

Step 2 : After placing first queen in the first column we cannot placed the second queen in the first or second column. So, we place Q_2 in the third column



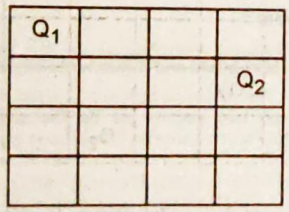
cannot be placed Q_2 in second column diagonal formed.

Step 3 : After placing the first and the second queen we cannot place Q_3 anywhere.

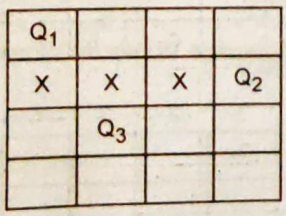


We cannot place Q_3 any where.

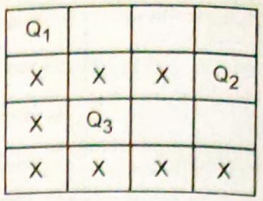
So, we apply another placement for Q_2 as shown below.



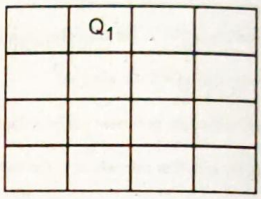
Now we only have option to place Q_3 in the column two.



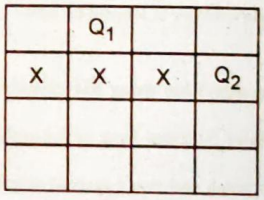
Step 4 : Here, after placing Q_1 in column 1, Q_2 in column 4 and Q_3 in column 2 again. We have no any option to place Q_4.



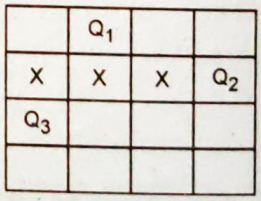
Step 5 : From step 4 it is noticeable that we have no any option to change the position of Q_1, Q_2 and Q_3 for the placement of Q_4. Hence, we now have to make backtracking and examine whether there is any option for placement of the queen. We now start with placement of the queen one in the column two.



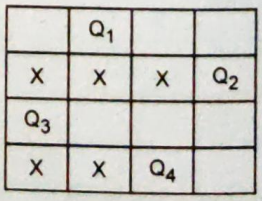
Step 6 : Having placed the queen one in the column two. We can place the queen two in the column four only.



Step 7 : Now, we place queen Q_3 in the column one.



Step 8 : Now, after placing queen one in column two, queen two in column four, queen three in column one. We can place queen four in column three.



Time complexity = O (K).

8 QUEEN'S PROBLEM

Q.27. State an eight queens problem and write its backtracking algorithm. Discuss its time complexity.

CT : S-07(6M), S-12(5M)

OR Write an algorithm to solve "8-Queens" problem. Explain the explicit and implicit constraints associated with this problem. Give at least two solutions for this problem.

CS : S-11(6M), W-11(8M), W-13(7M)

OR Give any two solutions for 8-Queen problem. Explain the constraint implementation formula. **CS : W-12, S-14(6M)**

OR State two possible solutions in 8-Queen problem. **CS : S-12(6M)**

OR Explain 8 queen's problem in detail. Give at least two solutions for this problem. **CT : S-14(7M)**

Ans. **8 - Queen problem :**

- 8-Queen problem is a classic combinational problem. This problem can be solved by applying backtracking we can define 8-queens to be placed on an 8 × 8 chessboard so that no two queens are on the same row, column or diagonal (both major and minor).
- A chess board of size 8 × 8 is given along with 8-Q's. It is required to place the Q's such that

(i) No two Q's should be in same row.

(ii) No two Q's should be in same column.

(iii) No two Q's should be diagonally opposite.

```
Algorithm NQueen (K, n)
{
    For i = 1 to n do
    {
        if (place [K, i]) then
        {
            x [K] = i
            if (K = n) then
            write (x [1....n])
            else
            NQueen (K + 1, n)
        }
    }
}
```

- ```
  Algorithm place (K, i)
  {
      For j = 1 to K – 1 do
      {
          if (x [j] = i) or
          (abs (x [j] – i)) = (abs (j – K)) then
              return false
      }
      return true
  }
  ```

- One possible solution for 8 queens problem

Implicit Constraint :

(1) No two Q's should be present in same row.

(2) No two Q's should be present in same column.

(3) No two Q's should be diagonally opposite.

Explicit constraint :

Required to place the Q's on the chessboard. Solution space consist of 8^8 combinations.

Possible solutions :

	1	2	3	4	5	6	7	8
1				Q_1				
2						Q_2		
3								Q_3
4		Q_4						
5							Q_5	
6	Q_6							
7			Q_7					
8					Q_8			

The sequence of Queen's are : [4, 6, 8, 2, 7, 1, 3, 5]

	1	2	3	4	5	6	7	8
1		Q_1						
2				Q_2				
3						Q_3		
4								Q_4
5					Q_5			
6							Q_6	
7	Q_7							
8			Q_8					
  ```

The sequence of the Queen's are : [2, 4, 6, 8, 5, 7, 1, 3]

Formula : abs (i – K) = abs (K – L)

abs (K – i) = abs (j – i)

Complexity : O (K).

Q.28. State an explain the N-Queens problem. *CT : S-12(2M)*

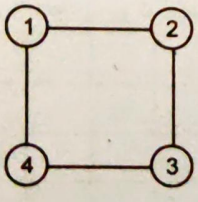
Ans. **N-Queens problem :**

- N-Queens problem is to place n-Queens in such a manner on an n * n chess board that no two queens attack each other by being in the same row, column or diagonal.
- It can be seen that for n = 1, the problem has a trivial solution and no solution exists for n = 2 and n = 3, so first we consider 4-queen problem and the generalized it to n-queen problem.
- N-queen problem is a classic problem that is often used in terms of standard 8 * 8 chessboard and can be solvable for $N \geq 4$.
- In general, problem is to place the queen on the board so that no two queens attack each other.
- In an N-by-N board, each queen is located on exactly one row, one column and two diagonals.

GRAPH COLORING

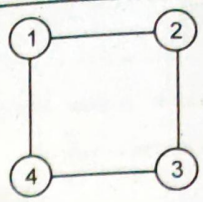
Q.29. Explain graph coloring method with a suitable example. Give algorithm for it. *CT : S-06,08,13 W-10(7M), S-11, W-12(6M)*

OR **Implement graph coloring on following graph and generate solution space tree of number permitted colors = 3. Write algorithm for graph coloring.** *CS : W-10, S-11(5M), S-12(7M)*



OR **Explain graph coloring problem. Explain how this problem can be solved by backtracking method. Draw state space free for given graph. How many colors are required?**

CT : W-09(6M), CS : W-10(5M)



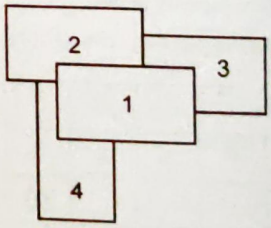
OR **Give a backtracking algorithm for graph colouring problem. Discuss its time complexity.** *CT : W-07(7M)*

Ans. **Graph coloring :**

- The main objective of graph coloring problem is to colour the given graph with minimum number of colors. Such that two adjacent vertices will be in different colors.
- The number of colors to be used are fixed.
- It is required to color the graph such that no two adjacent vertices will be in same color.
- Number of colors used should not exceed the given value.

  Color (A) ≠ Color (B)
- If d = degree of graph the number of colors required = d + 1.
- The number of colors required to color the graph is called as "chromatic number.

Example : Coloring maps.



Explicit constraints :

Fixed number of vertices and colors.

Implicit constraint :

Adjacent vertices should be in different colors.

Algorithm :

(1) The graph is represented in the form of matrix. In this matrix if the edge is present then G <I, j> = 1, otherwise = 0.

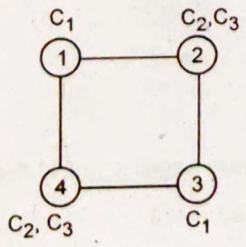
(2) Let variable 'm' represent number of colours.

(3) The algorithm will generate an array x [1...n] which will provide information about one solution initially all the values in x are 0.

(4) The algorithm uses a function NEXT VALUE (K) where K represents a vertex to be colored.

```
Algorithm Geolor (K)
{
    do
      {
      NEXTVALUE (K)
      if (x [K] = 0) then
          Return
      if (K = n) then
          Write (x [1....n])
      else
          Geolor (K + 1)
      }
    while (false)
}
Algorithm NEXTVALUE (K)
{
    do
      {
        x [K] = (x [K] + 1) mod (m + 1)
        if (x [K] = 0) then
            return
        for j = 1 + 0 n do
        {
            if (G [K, j] ≠ 0) and (x [K] = x [j])
                                                    then
                return
        }
            if (j = n +1) then
                return
      }
        while (false)
}
```

Example :

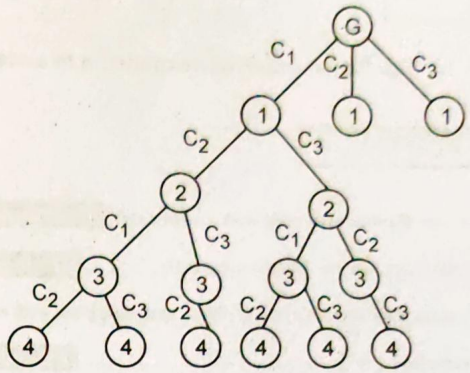


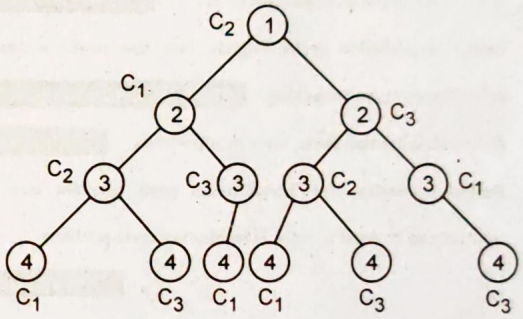
Graph of 4 vertices

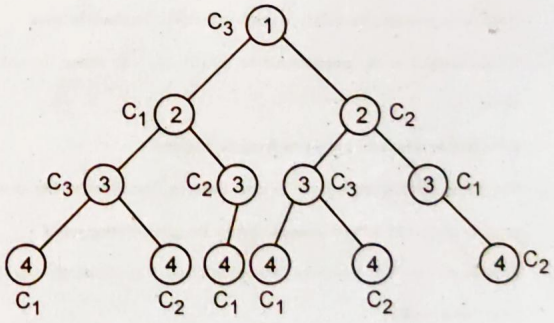
Number of colors = 3

Solution space tree :

This tree will define all possible solutions which can exist for the given graph and number of colors.







Total solution = 6 × 3 = 18

Number of colors = 3 (i.e. d + 1)

Hence, degree of graph = 2

Q.30. What is plannar graph? **CS : W-10, S-11(2M)**

Ans. Plannar graph :

A graph is plannar if it can be drawn such that no edges cross when drawn on a plane surface. A map, as used by geographers can be easily converted to plannar graph.

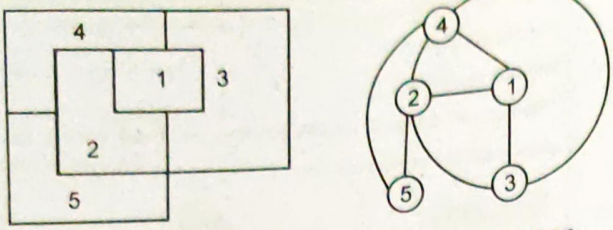


Fig. Planar graph corresponding to a map

HAMILTONIAN CYCLE

Q.31. Explain Hamiltonian cycle with an example. ***CT : S-06,08(6M)***

OR Write short note on Hamiltonian cycle. ***CT : W-10(4M)***

OR Explain Hamiltonian cycle problem and algorithm with suitable example. ***CT : S-10(6M)***

OR What is the use of Hamiltonian cycle? ***CS : S-11, W-11(2M)***

OR Define Hamiltonian cycle. Explain how this problem can be solved by using backtracking. ***CT : S-09, W-12(6M), S-13(2M)***

OR Define Hamiltonian cycle. Give its algorithm. ***CT : W-08(6M)***

OR Design a solution for Hamiltonian path. Explain how the solution can be used to solve Hamiltonian cycle problem. ***CS : S-13,14(6M)***

Ans. Hamiltonian cycle :

- It is a cycle generated for the given graph with following characteristics.

(1) All the vertices of the graph should be visited only one except the source vertex.

(2) If the edge is used in the cycle then it can be repeated.

- The solution for this problem can be more than one. The algorithm design will generate any one of the valid sequences from the given starting vertex.

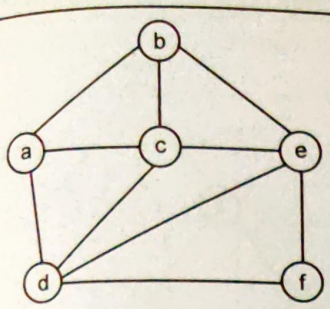
Explicit concern : If n is given vertices in graph, it is not possible to visit more than n vertices.

Implicit concern : If the vertex is already visited then it will be available in previously generated sequence and it cannot be used again.

Algorithm :

```
Algorithm Hcycle (K)
{
    repeat
    {
    next value (K)
    if (x [K] > = 0)
        if [ K = n] then
            Write x [ 1.....n]
    else
        Hcycle (K + 1)
    }
        until (false)
}
Algorithm nextvalue (K)
{
repeat
    {
        x [K] = [ (x [K] + 1) mod (n + 1)]
        if (x [K] =0) then return
        if (G [K– 1, K] ≠ 0) then
        for j = 1 to K – 1do
        {
            if (x [K] = x [j]) then
                break;
        }
        if (j = k) then return
            {
                if (K < n) then
                    return
                if (K =  n) then
                {
                    if ( ∈ [n, 1] ≠ 0 ) then
                }
                    return
                    until (false)
            }
    }
}
```

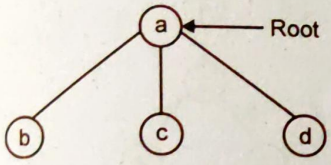
Example : Consider a graph G = (V, E) find a Hamiltonian circuit using Backtracking method.



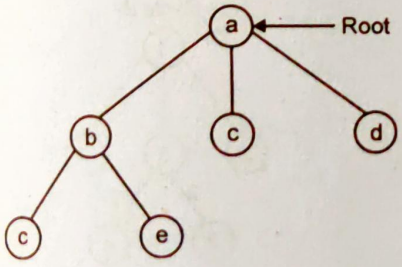
Firstly, we start our search with vertex 'a' this vertex 'a' becomes the root of our implicit tree.

(a) ← Root

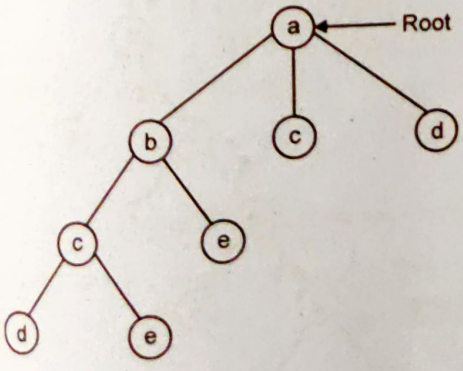
Next, we choose vertex 'b' adjacent to 'a' as it comes first in lexicographical order b, c, d.



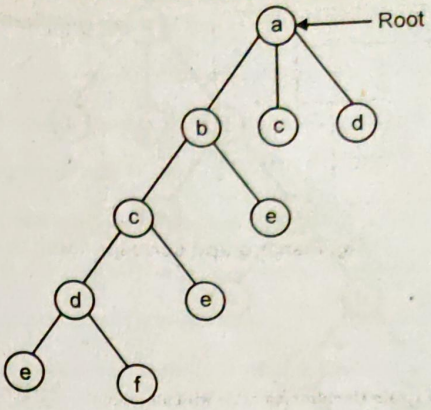
Next, we select 'c' adjacent to 'b'.



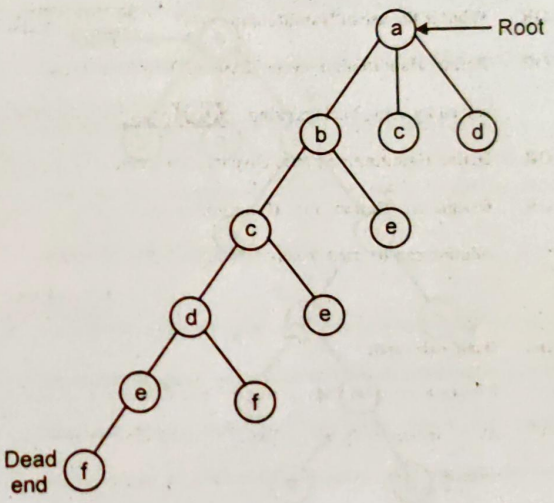
Next, we select 'd' adjacent to 'c'.

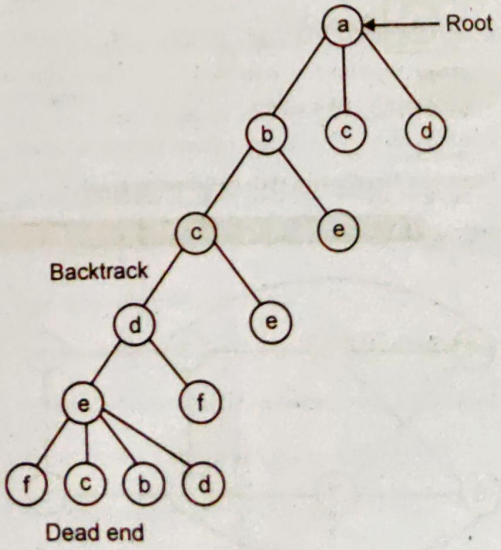


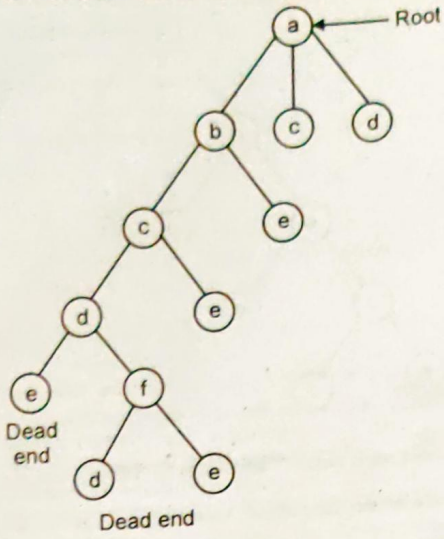
Next, we select 'e' adjacent to 'd'.

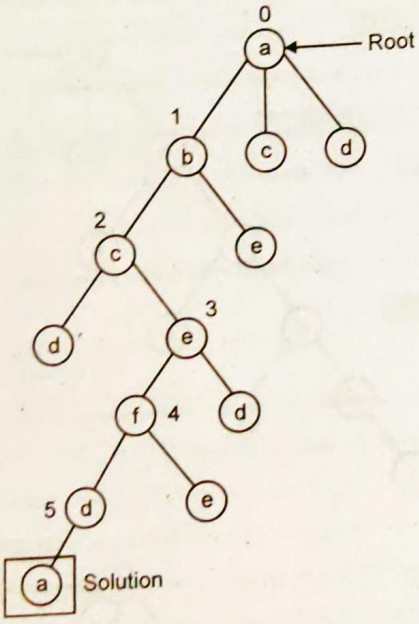


- Next, we select vertex 'f' adjacent to 'e'. The vertex adjacent to f are d and e but they have already visited. Thus, we get the dead end and we backtrack one step and remove vertex 'f' from partial solution.





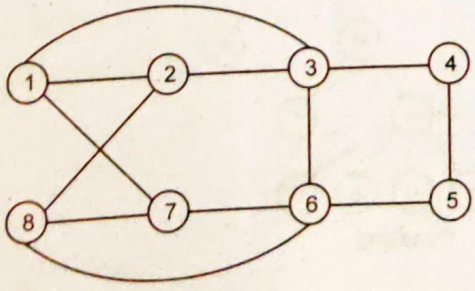




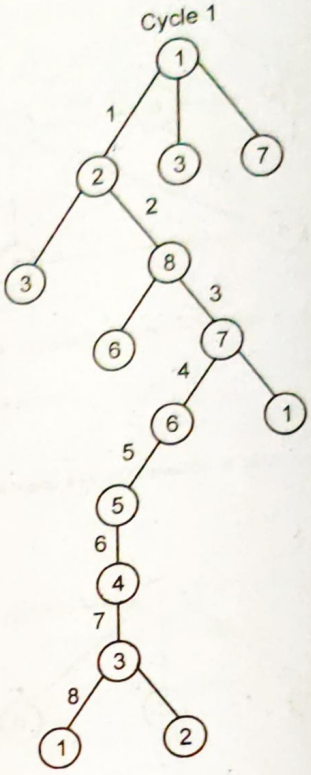
The sequence is : a-b-c-e-f-d-a.

Q.32. Implement Hamiltonian cycle on following graph.

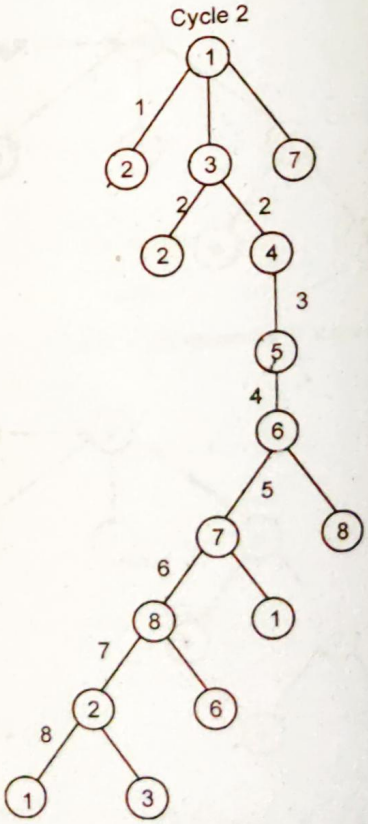
CS : S-11(5M), W-11, S-13(3M), W-12(8M), W-13(6M)



Ans.

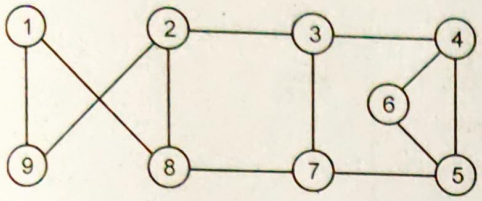


Path = 1-2-8-7-6-5-4-3-1

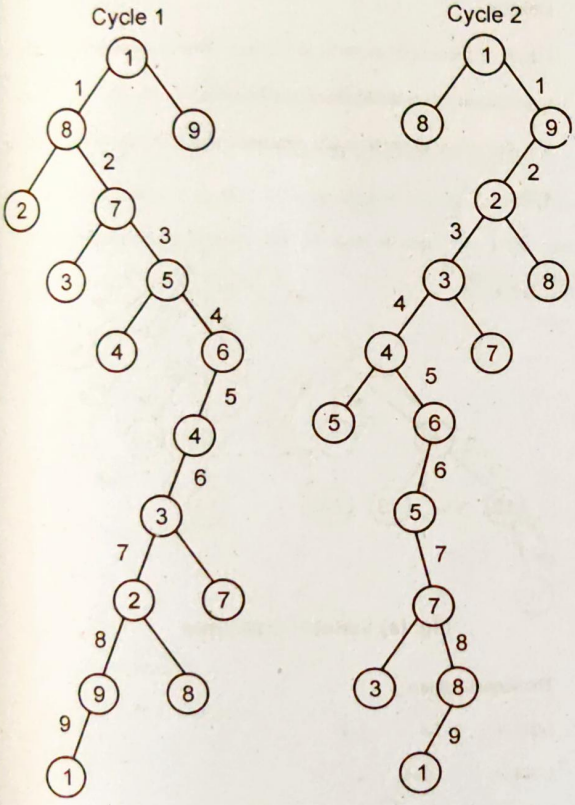


Path = 1-3-4-5-6-7-8-2-1

Q.33. For the following graph design Hamiltonian cycle. Write the algorithm for same. Explain how one solution may result in multiple solution. **CS : W-10(7M)**



Ans.



Path for cycle 1 : 1-8-7-5-6-4-3-2-9-1

Path for cycle 2 : 1-9-2-3-4-6-5-7-8-1

Q.34. Determine the order of magnitude of the worst case computing time for the back-tracking procedure that finds Hamiltonian cycles. **CT : W-11(5M)**

Ans.

- Let G = <V, E> be a connected graph with n nodes.
- A Hamiltonian circuit or cycle is a round trip path along n edges, which visits each node exactly once and returns to the starting node.
- The solution vector is x [0 : n – 1] such that x [i] represents the ith visited node in the circuit.
- We have to select a set of possible vertices for x [k], once x [0], x [1], ..., x [K –1] are already chosen. It should of course be connected by an edge to x [K–1].
- The function next value ( ) in the algorithm finds the next node which is connected by an edge to x [K – 1] and is distinct.
- The top level function Hamilton( )follows the recursive backtrack algorithm closely and prints the result complete circuit is built up and then repeats to check for the presence of any more circuits.
- The time complexity next value take $O(n^2)$ time for selecting next node and Hamilton( ) calls it $O(n^n)$ times and the worst case.
- Thus, the time complex is $O(n^{2+2})$ but in actual we reduces drastically, due to the backtracking.

Q.35. Consider the sum of subset problem in which n distinct positive numbers are given and desire to find all combinations of these numbers whose sum are m. Write down the recursive backtracking algorithm for the above said problem by sum of subsets. **CT : W-11(10M)**

OR Explain backtracking algorithm with the help of sum of subset problem. **CT : S-10(4M)**

OR Given 'n' distinct positive numbers. Write an algorithm to find all combination of these number whose sum are m, using backtracking technique. **CT : W-07(7M)**

Ans. **Sum of subset problem :**

Given an array a [1....n] of positive integers and a variable 'm', sum of subset problem will find a sequence from the array consisting of distinct element such that sum of elements = m.

Explicit constraint :

(1) Only positive values can be used.

(2) Only elements from the array can be used.

Implicit constraint :

(1) Element can be used only once.

(2) If sequence starts with a[i] then it can contain elements from index a [i + 1].

```
Algorithm sumofsub (s, K, r)
{
    x [K] = 1;
    if (s + W [K] = m) then
       write (x [1 : K]);
    else if (s + w [K] + w [K + 1] ≤ m )
     then sumofsub (s + w [K], K + 1, r – w [K]);
     if ((s + r – w [K] ≥ m) and (s + w [K +1] ≤ m )) then
     {
        x [K] = 0;
        sumofsub (s, K + 1, r – w [K]);
     }
}
```

Q.36. Let W = {11, 13, 7, 24} and m = 31. Find all possible subset of 'W' that generates sum = m and draw fixed size tuple and variable size tuple solution space tree for same. ***CS : W-10(6M)***

Ans.

- There are two types of method to desire the sequences. Both the methods based on trees :

  (1) Variable size tuple.

  (2) Fixed size tuple.

(1) Variable size tuple :

- In this method a solution space tree is generated which represents all the valid sequences for the given arrays.
- Out of the valid sequences the sequence resulting in sum = 31 i.e. sum = m will be selected.

$W = \begin{bmatrix} x_1 & x_2 & x_3 & x_4 \\ 11 & 13 & 7 & 24 \end{bmatrix}$

m = 31

[11 13 7] i. e. $[x_1, x_2, x_3]$

[7 24] $[x_3\ x_4]$

(1) [1 1 1 0]

(2) [0 0 1 1]

$x_1\ x_2\ x_3\ x_4$

1 = used, 0 = not used

Positive sequences :

(1) 1-2-3-4
(2) 1-2-4
(3) 1-3-4
(4) 1-4
(5) 2-3-4
(6) 2-4
(7) 3-4
(8) 4.

} 8 Sequences

- As shown in the sequences all the sequence terminates with last element.
- Hence, if there is an element in the array whose value is equal to the required sum then the element will be selected only if.
- It is the single element in the sequence and it is last element of the array.

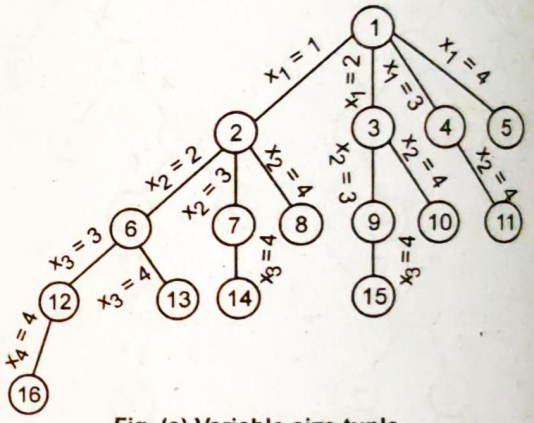


Fig. (a) Variable size tuple

The sequences are :

1-2-3-4 2-3-4 3-4

1-2-4 2-4 4

1-3-4

1-4

∴ solution nodes :

1-2-3

3-4

(2) Fixed size tuple :

- In this method all the solutions will terminate at leaf node.
- The edge from lable i to i + 1 is labelled with xi which can be either 1 or 0.

- The solution is defined by all the path from root node terminating at leaf node.

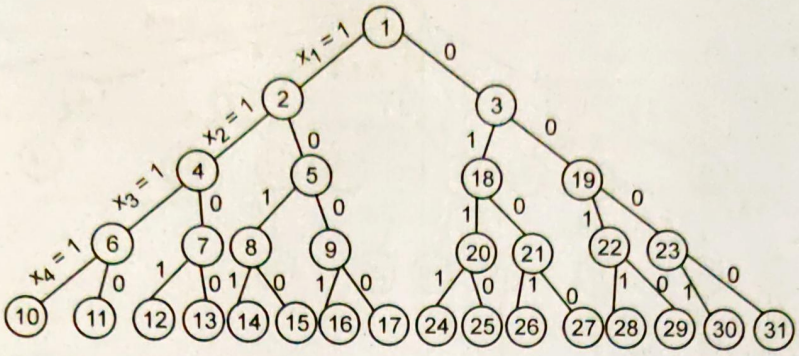


Fig. (b) Fixed size tuple

Solution node :

1-2-3

3-4

Q.37. Given a set W = {3, 4, 5, 6} and m = 9. Find all possible subsets of W that sum to m. Do this using sum of subset algorithm based on backtracking? Draw the portion of state space tree that is generated. **CT : S-12(6M)**

Ans. $x_1\ x_2\ x_3\ x_4$

W = {3, 4, 5, 6}

m = 9.

$[x_1 - x_4] = 9$. i.e. [3 – 6]

$[x_2 - x_3] = 9$ i.e. [4 – 5]

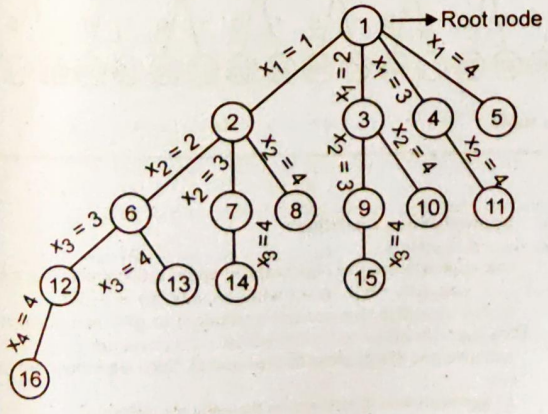
[1 0 0 1]

[0 1 1 0]

Solution space tree for

W = (3, 4, 5, 6) and m = 9

No. of nodes = 4.



Positive sequences :

1-2-3-4 2-3-4 3-4

1-2-4 2-4 4

1-3-4

1-4

Solution node :

1-2-3

3-4

Q.38. Draw the solution space tree for fixed and variable size tuple for the following array and positive integer.

a = [12, 16, 9, 25, 10] sum = 35.

Find out the solution nodes. **CS : W-12(7M)**

Ans. a = [12, 16, 9, 25, 10]

Sum = 35

Sequence = [4-5] i.e. 00011

[2-3-5] i.e. 01101

Positive sequence :

1-2-3-4-5

1-2-3-5

1-2-4-5

1-3-4-5

1-5

2-3-4-5

2-3-5

2-4-5

2-5

3-4-5

3-5

4-5

5

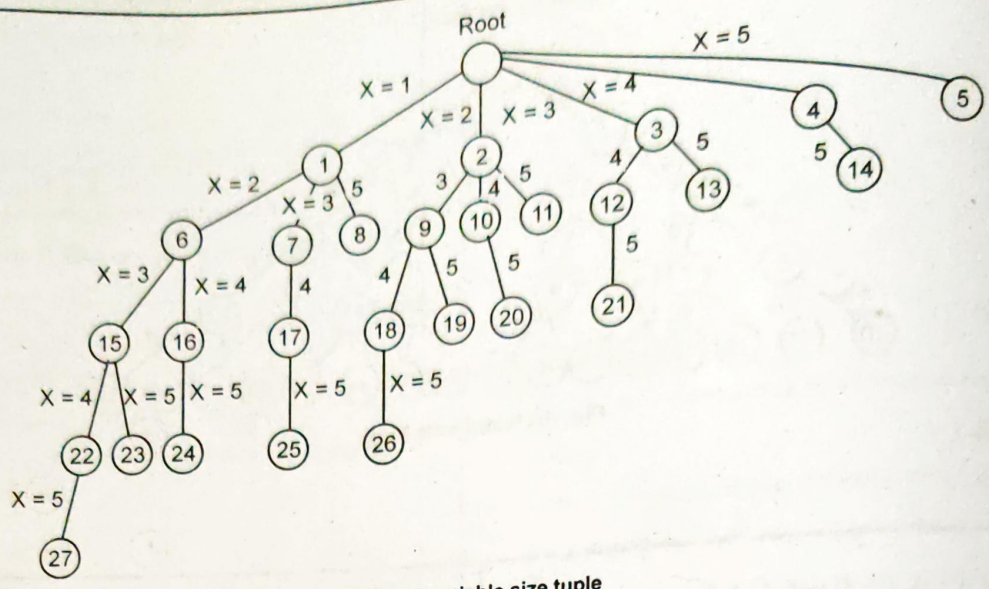


Fig. (a) variable size tuple

Solution node = (14) [4-5]

= (19) [2-3-5]

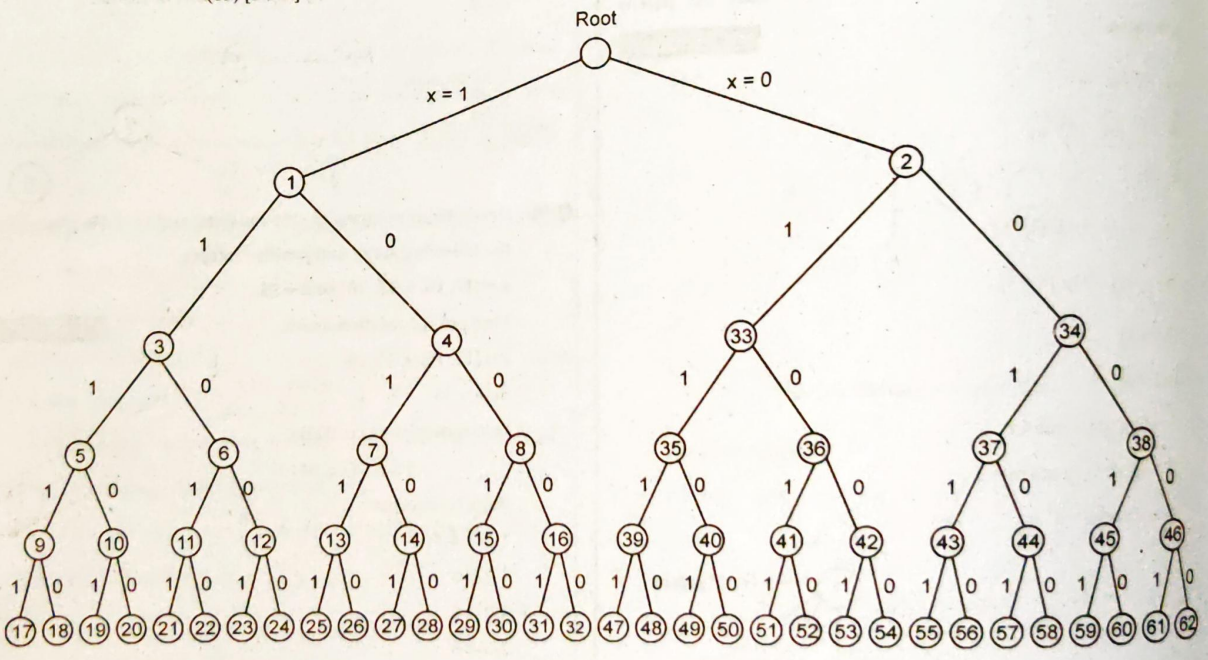


Fig. (b) Fixed size tuple

INTRODUCTION TO APPROXIMATION ALGORITHM

Q.39. Explain approximation algorithm. Write an algorithm for vertex cover problem using approximation principle. *CS : W-13(6M)*

OR What is optimal vertex cover problem? Write algorithm. *CS : S-13(4M)*

Ans. Approximation algorithm :

- One approach to solve Np-complete optimization problem is the use of fast algorithm that are not guaranteed to give best solution but will give one that is close to the optimal. Such algorithms are called as approximation algorithms or heuristic algorithms.

VBD

In approximation algorithm, we remove the requirement that the algorithm solves the optimi-zation problem p must always generate an opti-mal solution.

A feasible solution with value close to the value of an optimal solution is called an approximate solution.

An approximation algorithm for P is an algorithm that generates approximate solutions for P.

Examples of Approximate algorithms are as follows :

(1) Vertex cover problem.

(2) Travelling salesman problem.

(3) Set covering problem.

(1) **Vertex cover problem :**

A vertex cover of an undirected graph G = (V, E) is a subset W of V ($W \subseteq V$) such that, for every (a, b) in E, 0 is in W or b is in W. The size of vertex cover is the number of vertices in it.

Algorithm :

Vertex : cover-Approx (G)

Input : An undirected graph G

Output : A vertex-cover C for G

(1) C ← Empty set

(2) H ← G

(3) while H has edges

(4) e ← H. remove edge (H. anedge ( ))

(5) v ← H. origin (e)

(6) w ← H. destination (e)

(7) C . add (v)

(8) C . add (w)

(9) for each f incident to v or w

(10) H. remove Edge ( )

(11) return C

Analysis : The running time of this algorithm is O (v + e) using adjacency list to represent H.

(2) **Travelling salesman problem :**

Imagine we are salesman and we need to visit n cities. We want to start a tour at a city and visit every city exactly one time and finish the tour at the city from where we start. There is a non-negative cost (i, j) to travel from city i to city j. The goal is to find a tour of minimum cost. Such problem is called traveling salesman problem (TSP).

Approx-TSP (G = V, E)

{

(1) Compute a MST T of G;

(2) Select any vertex r be the root of the tree;

(3) Let L be the list of vertices visited in a preorder tree walk of T;

(4) Return the Hamiltonian cycle H that visits the vertices in the order L;

}

Example :

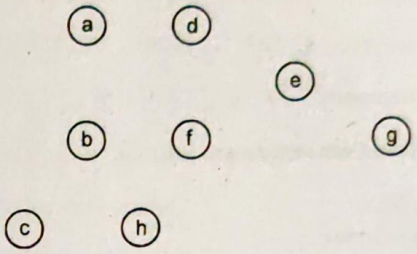


Fig. (a) Given set of points

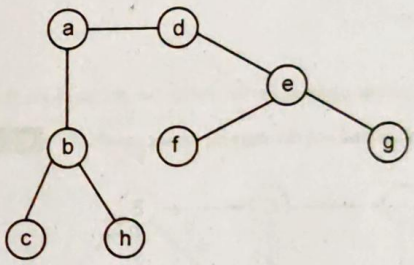


Fig. (b) MST

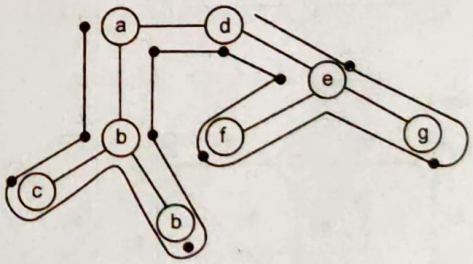


Fig. (c) Full tree walk on T

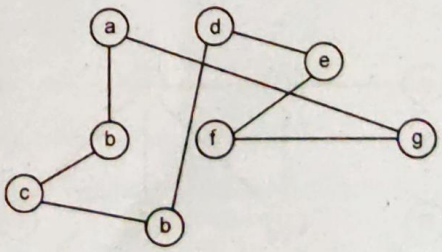


Fig. (d) Preorder sequence gives a tour H

(3) Set covering problem :

- The set covering problem is an optimization problem that models many resource selection problems.
- We can define OPT-SET-COVER problem as follows, Given a collection of m sets, find the smallest number of them whose union is the same as the whole collection of m sets? POT-SET-COVER is NP-hard.

Algorithm :

Set cover-Approx (G)

(1) $F \leftarrow \{S_1, S_2, \ldots S_m\}$

(2) C ← Empty set.

(3) U ← Union of $S_1 \ldots S_m$.

(4) While U is not empty.

(5) S_i ← Set in F with most elements in U.

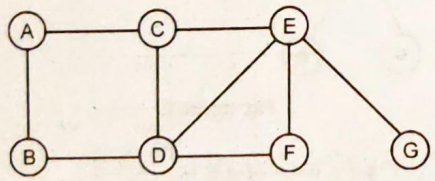
(6) F.remove (Si).

(7) C. add (Si).

(8) Remove all elements in Si from U.

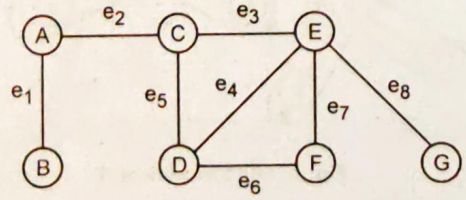
(9) Return C.

Q.40. Implement optimal vertex cover the problem on the following graph to find out the suitable vertex cover. ***CS : S-13(3M)***

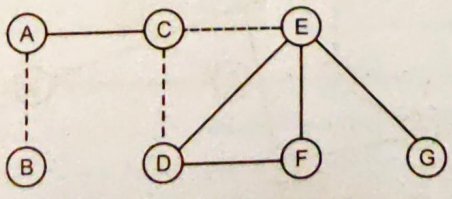


Ans.

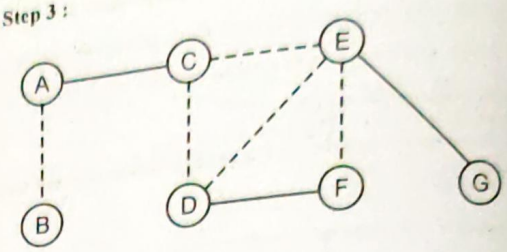
Step 1 :



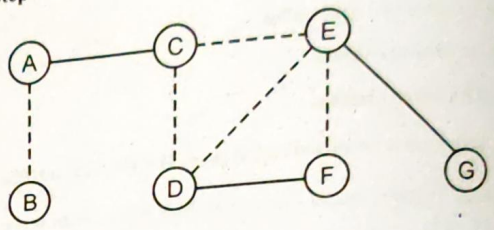
Step 2 :



Step 3 :



Step 4 :



Vertex cover are : { A, C, D, E, F, G }

Q.41. Discuss various graph representation scheme with suitable example. Write advantages of each scheme. ***CT : W-08(6M)***

Ans.

- There are three representations of graphs in memory. They are :

  (1) adjacency matrices.

  (2) adjacency lists.

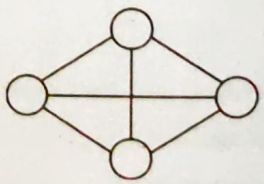
  (3) adjacency multilists.

(1) Adjacency matrices :

- For a graph with n vertices, the adjacency matrix of 2D with n×n elements is used.

  A[i, j] = 1, if the edge <Vi, Vj> or (Vi, Vj) is in the graph, otherwise, A[i, j] = 0, if there is no such edge.

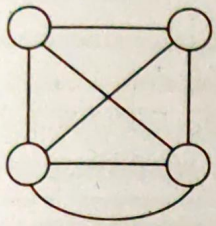
- Consider, the following undirected graph :



$$\begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \end{array} \begin{array}{c} \begin{array}{cccc} 1 & 2 & 3 & 4 \end{array} \\ \begin{bmatrix} 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 0 \end{bmatrix} \end{array}$$

- The adjacency matrix for undirected graph is symmetrical, i.e., the diagonal elements are zero. The total memory required will be n×n location.

- If the upper or lower triangular matrix elements are kept in the memory, instead of saving the complete matrix, as A[i,j]=A[j, i] and the diagonal elements are zero.
- The degree of any vertex is equal to the sum of the row of the matrix.
- Consider the directed graph.



$$\begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \end{array} \begin{array}{c} \begin{array}{cccc} 1 & 2 & 3 & 4 \end{array} \\ \begin{bmatrix} 0 & 0 & 0 & 1 \\ 1 & 0 & 1 & 1 \\ 1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix} \end{array}$$

- For directed graph, the row sum is the outdegree and the column sum is indegree of the vertex.
- The advantage of this representation is that, it is possible to determine whether an edge is connecting two vertices or not.
- The drawback is that, it requires n×n memory locations. If the graph is not complete, there will be a number of zero entries and this will waste the memory area. To avoid this, the adjacency list method is used.

(2) **Adjacency list :**

- The concept of list is used to represent the graph.
- The 'n' rows of adjacency matrix are represented using n linked lists.
- The list is used for one row.
- Each list has a head node.
- The head nodes are sequential and they provide easy and random access to the adjacency list for any particular vertex.
- For undirected graph with 'n' vertices and 'e' edges, this representation requires 'n' head nodes and '2e' list nodes.

(3) **Adjacency multilist :**

- From the adjacency list representation of undirected graph, it is clear that each edge (Vi, Vj) is represented by two entries, one in list Vi and another in list Vj. This is unnecessary wastage of the memory as the information present is same at both the nodes (1, 2) = (2, 1).
- To avoid this, the adjacency multilist is used. Using this representation, the nodes are shared amongst the several lists.
- In this method for each edge of the graph, there will be exactly one node. But this node will provide the information about two more nodes to which it is incident .
- The node structure with this representation will be :

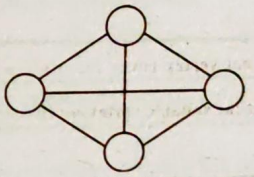
m	V1	V2	P1	P2

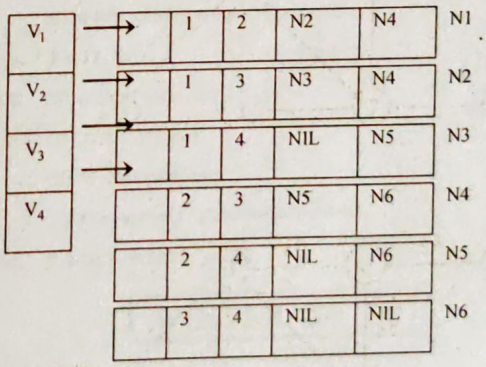
m - tag field; V1, V2 – vertex; P1, P2 - incident paths.

m is a one bit mark field that is used to indicate whether or not the edge has been examined.

- Consider the following graph :

Total edges are, (1, 2), (1, 3), (1, 4), (2, 3), (2, 4), (3, 4)





- The lists are :

Vertex 1 : N1 → N2 → N3

Vertex 2 : N1 → N4 → N5

Vertex 3 : N2 → N4 → N6

Vertex 4 : N3 → N5 → N6

- The storage requirement of adjacency multilist is the same as adjacency lists except for the addition of the mark list m.

POINTS TO REMEMBER :

(1) The predecessor subgraph G_π is a breadth-first tree if it consist of the vertices reachable from s and for all $v \in V_\pi$.

(2) Depth first search edges are explored out of the most recently discovered vertex v that still has unexpected edges leaving the vertex from which v was discovered.

(3) Articulation point is a node of given graph on which maximum edges and vertices of the graph are connected.

(4) A topological sort of a graph is an ordering of the vertices so that all edges go from left to right.

(5) Backtracking is a systematic way to go through all the possible configuration of a search space.

(6) The main objective of graph coloring problem is to colour the given graph with minimum number of colors such that two adjacent vertices will be in different color.

(7) A graph that contains a Hamiltonian cycle is said to be Hamiltonian otherwise it is non Hamiltonian.

(8) A directed graph G = (V, E) is a simple Hamiltonian cycle that contains each vertex in V.

(9) In a graph is a simple path that visits every vertex exactly once is known as Hamiltonian path.

(10) Given a finite set $S \subset N$ and a target $t \in N$. We ask whether there is a subset $S' \subseteq S$ whose elements sum to t. This is known as the subset sum problem.