GRAPHS

Chapter at a Glance

- A graph is a mathematical tool used to represent a physical problem. It is also used to model networks, data structures, scheduling, computation and a variety of other systems where the relationship between the objects in the system plays a dominant role.
- **Types of graph:** A graph often symbolized as G can be of two types:
  1. **Undirected graph:** where a pair of vertices representing an edge is unordered.
  2. **Directed graph:** where a pair of vertices representing an edge is ordered.
- **Graph Traversal:** A graph can be traversed in two ways:
  Depth first search traversal: often analogous to pre-order traversal of an ordered tree.
  Breadth first search traversal: often analogous to level-by-level traversal of an ordered tree.
- **Spanning Tree:** Given a connected, undirected graph, a spanning tree of that graph is a sub graph which is a tree and connects all the vertices together. A single graph can have many different spanning trees. We can also assign a *weight* to each edge, which is a number representing how unfavourable it is, and use this to assign a weight to a spanning tree by computing the sum of the weights of the edges in that spanning tree.
- **Shortest Path:** In graph theory, the shortest path problem is the problem of finding a path between two vertices in a weighted graph such that the sum of the weights of its constituent edges is minimized.
  The most important algorithms for solving this problem are:
  Dijkstra's algorithm solves the single-source shortest path problems.
  Bellman-Ford algorithm solves the single source problem if edge weights may be negative.
  A* search algorithm solves for single pair shortest path using heuristics to try to speed up the search.

Multiple Choice Type Questions

1. The vertex, removal of which makes a graph disconnected is called [WBUT 2007]
a) Pendant vertex
b) bridge
c) articulation point
d) colored vertex
Answer: (c)

2. A vertex of in-degree zero in a directed graph is called [WBUT 2007, 2018]
a) articulation point
b) sink
c) isolated vertex
d) root vertex
Answer: (c)

3. Adjacency matrix of a digraph is [WBUT 2007, 2012, 2016]
a) identity
b) symmetric
c) asymmetric
d) none of these
Answer: (b)

4. Which data structure is used for breadth first traversal of a graph? [WBUT 2008]
   a) Stack
   b) Queue
   c) Both stack and queue
   d) None of these

Answer: (b)

5. The adjacency matrix of an undirected graph is [WBUT 2008, 2010]
   a) Unit matrix
   b) Asymmetric matrix
   c) Symmetric matrix
   d) None of these

Answer: (c)

6. BFS [WBUT 2008]
   a) scans all incident edges before moving to the other vertex
   b) scans adjacent unvisited vertex as soon as possible
   c) is same as backtracking
   d) none of these

Answer: (b)

7. A non-planar graph with minimum number of vertices has [WBUT 2008, 2018]
   a) 9 edges, 6 vertices
   b) 6 edges, 4 vertices
   c) 10 edges, 5 vertices
   d) 9 edges, 5 vertices

Answer: (c)

8. Any connected graph with x vertices must have at least [WBUT 2009]
   a) x+ 1 edges
   b) x-1 edges
   c) x edges
   d) x/2 edges

Answer: (b)

9. Maximum number of edges in a n-node undirected graph without self loop is [WBUT 2010]

   a) n^2
   b) $\frac{n(n-1)}{2}$
   c) $n-2$
   d) $\frac{(n+1)(n)}{2}$

Answer: (b)

10. BFS constructs [WBUT 2010, 2014, 2018]
   a) a minimal cost spanning tree of a graph
   b) a depth first spanning tree of a graph
   c) a breath first spanning tree of a graph
   d) none of these

Answer: (a)

11. A complete directed graph of 5 nodes has..... [WBUT 2011]
   a) 5
   b) 10
   c) 20
   d) 25

Answer: (b)

12. A vertex with degree one in a graph is called [WBUT 2011]
   a) leaf
   b) pendant vertex
   c) end vertex
   d) none of the

Answer: (b)

13. To implement
   a) Stack

Answer: (a)

14. Adjacency mat
   a) unit matrix
   c) asymmetric

Answer: (b)

15. What is the su
   a) 19
   b) 20
   c) 5
   d) none of thes

Answer: (a)

16. The adjacency
   a) Unit matrix
   c) Symmetric

Answer: (c)

17. A path is
   a) a closed wal
   b) an open wal
   c) an open wal
   d) a closed wal

Answer: (c)

1. Describe Kruska

Answer:

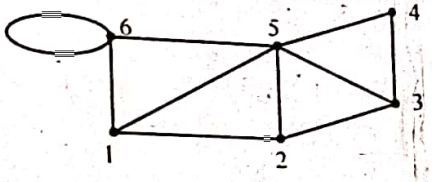
Kruskal's algorithm
for a connected wei
tree that includes e
minimized. If the g
minimum spanning
example of a greedy
It works as follows:

- create a forest F
- create a set S cor
- while S is nonem
- remove an edge

13. To implement DFS which data structure is generally used? [WBUT 2013]
a) Stack b) Queue c) Both (a) & (b) d) None of these
Answer: (a)

14. Adjacency matrix for a digraph is – [WBUT 2014]
a) unit matrix b) symmetric matrix
c) asymmetric matrix d) none of these
Answer: (b)

15. What is the sum of the degrees of all the vertices in the following graph? [WBUT 2017]



a) 19
b) 20
c) 5
d) none of these
Answer: (a)

16. The adjacency matrix of an undirected graph is [WBUT 2017]
a) Unit matrix b) Asymmetric matrix
c) Symmetric matrix d) none of these
Answer: (c)

17. A path is [WBUT 2017]
a) a closed walk with no vertex repetition
b) an open walk with no vertex repetition
c) an open walk with no edge repetition
d) a closed walk with no edge repetition
Answer: (c)

Short Answer Type Questions

1. Describe Kruskal's minimal spanning tree algorithm. [WBUT 2008, 2018]

Answer:
Kruskal's algorithm is an algorithm in graph theory that finds a minimum spanning tree for a connected weighted graph. This means it finds a subset of the edges that forms a tree that includes every vertex, where the total weight of all the edges in the tree is minimized. If the graph is not connected, then it finds a minimum spanning forest (a minimum spanning tree for each connected component). Kruskal's algorithm is an example of a greedy algorithm.

It works as follows:

- create a forest F (a set of trees), where each vertex in the graph is a separate tree
- create a set S containing all the edges in the graph
- while S is nonempty
- remove an edge with minimum weight from S

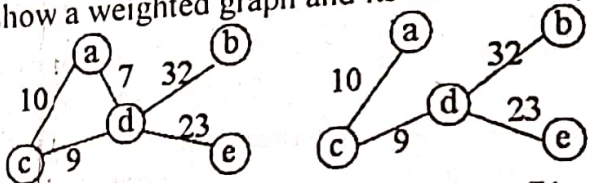
- if that edge connects two different trees, then add it to the forest, combining two trees into a single tree
- Otherwise discard that edge.
- At the termination of the algorithm, the forest has only one component and forms a minimum spanning tree of the graph.

2. What is a minimum spanning tree? Describe Huffman's Algorithm. [WBUT 2012]

Answer:

1st Part:

A Minimum Spanning Tree in an undirected connected weighted graph is a spanning tree of minimum weight (among all spanning trees). The minimum spanning tree may not be unique. However, if the weights of all the edges are pairwise distinct, it is indeed unique. The following figures show a weighted graph and its minimum spanning tree.



weighted graph Tree 1. w=74

2nd Part:

Huffman's algorithm is a method for building an extended binary tree with a minimum weighted path length from a set of given weights. Initially construct a forest of singleton trees, one associated with each weight. If there are at least two trees, choose the two trees with the least weight associated with their roots and replace them with a new tree constructed by creating a root node whose weight is the sum of the weights of the roots of the two trees removed, and setting the two trees just removed as this new node's children. This process is repeated until the forest consists of one tree.

Pseudocode

Huffman(W, n)
Input: A list W of n (positive) weights.
Output: An extended binary tree T with weights
taken from W that gives the minimum weighted path length.
Procedure:
Create list F from singleton trees formed from elements of W
WHILE (F has more than one element) DO
Find T1, T2 in F that have minimum values associated with their roots
Construct new tree T by creating a new node and setting T1 and T2 as its children
Let the sum of the values associated with the roots of T1 and T2 be associated with the root of T
Add T to F
End DO
Huffman := tree stored in F

3. What is the adjacency matrix representation of a graph? **[WBUT 2013]**

Answer:

The adjacency matrix of a graph is

	1	2	3	4	5	6	7
1	0	1	1	0	0	1	0
2	0	0	0	1	0	0	0
3	0	0	0	0	0	0	1
4	0	0	0	0	1	1	0
5	0	0	0	0	0	0	0
6	0	1	0	0	1	0	0
7	0	0	0	0	0	1	0

4. Write the Prim's algorithm for finding MST from a graph. **[WBUT 2014, 2016]**

Answer:

Prim's algorithm is a Greedy algorithm. It starts with an empty spanning tree. The idea is to maintain two sets of vertices. The first set contains the vertices already included in the MST, the other set contains the vertices not yet included. At every step, it considers all the edges that connect the two sets, and picks the minimum weight edge from these edges. After picking the edge, it moves the other endpoint of the edge to the set containing MST.

Algorithm

1) Create a set ***mstSet*** that keeps track of vertices already included in MST.

2) Assign a key value to all vertices in the input graph. Initialize all key values as INFINITE. Assign key value as 0 for the first vertex so that it is picked first.

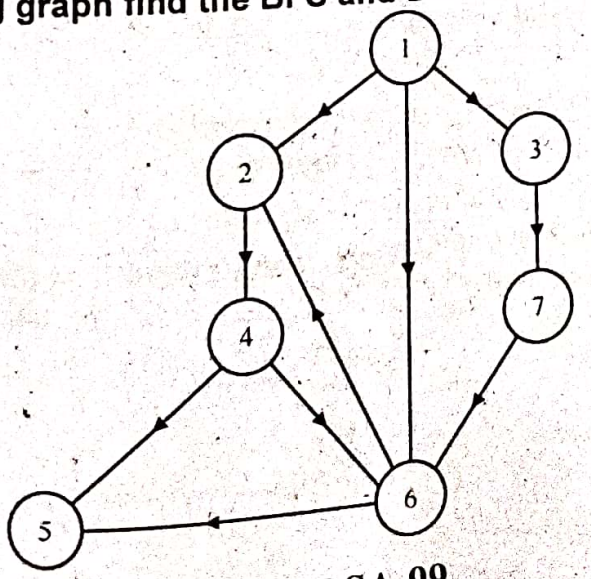
3) While mstSet doesn't include all vertices

....a) Pick a vertex ***u*** which is not there in ***mstSet*** and has minimum key value.

....b) Include ***u*** to mstSet.

....c) Update key value of all adjacent vertices of ***u***. To update the key values, iterate through all adjacent vertices. For every adjacent vertex ***v***, if weight of edge ***u-v*** is less than the previous key value of ***v***, update the key value as weight of ***u-v***

5. For the following graph find the BFS and DFS traversal with proper algorithm. **[WBUT 2015]**



Answer:

a) (i) The BFS traversal of the graph is:

Event	*Queue (Front to Rear)*	*BFS*
Visit 1		1
Visit 2	2	12
Visit 3	23	123
Visit 6	236	1236
Remove 2	36	
Visit 4	364	12364
Remove 3	64	
Visit 7	647	123647
Remove 6	47	
Visit 5	475	1236475
Remove 4	75	
Remove 7	5	
Remove 5		
Done		

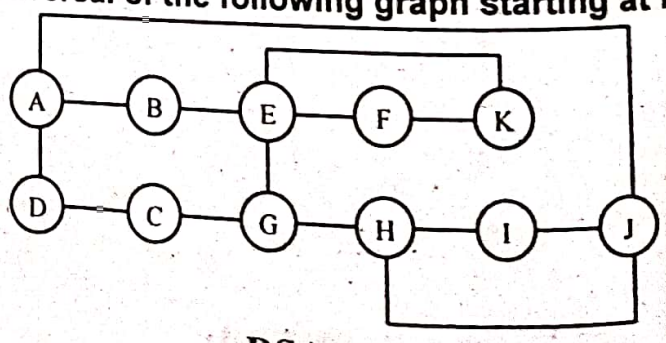
Required BFS is 1 2 3 6 4 7 5

(ii) The DFS traversal of the graph is:

Event	*Stack*	*DFS*
Visit 1	1	1
Visit 2	12	12
Visit 4	124	124
Visit 5	1245	1245
Pop 5	124	
Pop 4	12	
Visit 6	126	12456
Pop 6	12	
Pop 2	1	
Pop 1	-	
Visit 3	3	124563
Visit 7	37	1245637
Remove 7	3	
Remove 3	-	
Done		

Required DFS is 1 2 4 5 6 3 7.

6. Find out the DFS traversal of the following graph starting at node A. [WBUT 201



Answer:

For DFS traversa

- Rule 1 –
  Push it i
- Rule 2 –
  will pop
  vertices.
- Rule 3 –

Using the above
A B E G H I J F

7. Apply BFS/DF

Answer:

The steps involve

| **Search Steps**
| ----------
| **Step 1 -> Node**
| **Step 2 -> Node**
| **Step 3 -> Node**
| **Step 4 -> Node**
| **Step 5 -> Node**
| **Step 6 -> Node**
| **Step 7 -> Node**

The steps involve

| ----------
| Step 1 -> Node A
| Step 2 -> Node E
| Step 3 -> Node G
| Step 4 -> Node F
| Step 5 -> Node D
| Step 6 -> Node C
| Step 7 -> Node B

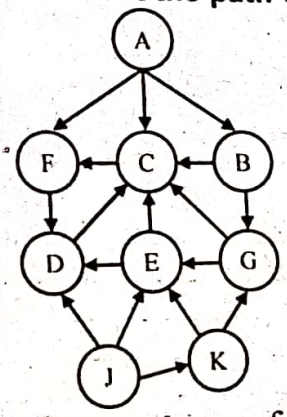
Answer:

For DFS traversal we employ the following rules using stack

- Rule 1 – Visit the adjacent unvisited vertex. Mark it as visited. Display it. Push it in a stack.
- Rule 2 – If no adjacent vertex is found, pop up a vertex from the stack. (It will pop up all the vertices from the stack, which do not have adjacent vertices.)
- Rule 3 – Repeat Rule 1 and Rule 2 until the stack is empty.

Using the above rules we get can get the following traversal when starting with A:
A B E G H I J F K C D

7. Apply BFS/DFS Algorithms and find out the path of the given graph: **[WBUT 2018]**



Answer:

The steps involved in **breadth first traversal** are as follows:

|Search Steps
|--------
|Step 1 -> Node A
|Step 2 -> Node B
|Step 3 -> Node C
|Step 4 -> Node F
|Step 5 -> Node G
|Step 6 -> Node D
|Step 7 -> Node E

The steps involved in **depth first traversal** are as follows:

|Step 1 -> Node A
|Step 2 -> Node B
|Step 3 -> Node C
|Step 4 -> Node F
|Step 5 -> Node D
|Step 6 -> Node G
|Step 7 -> Node E

UT 2016]

8. Describe the Edge classification of DFS algorithm. [MODEL QUESTION]

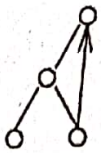
Answer:

Consider a directed graph G = (V, E). After a DFS of graph G we can put each edge into one of four classes:

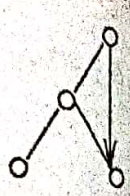
1. A ***tree edge*** is an edge in a DFS-tree.
2. A ***back edge*** connects a vertex to an ancestor in a DFS-tree. Note that a self-loop is a back edge.
3. A ***forward edge*** is a non-tree edge that connects a vertex to a descendent in a DFS-tree.
4. A ***cross edge*** is any other edge in graph G. It connects vertices in two different DFS-tree or two vertices in the same DFS-tree neither of which is the ancestor of the other.

1. A Tree Edge

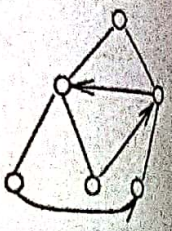
2. A Back Edge to an ancestor



3. A Forward Edge to a descendant



4. A Cross Edge to a different node

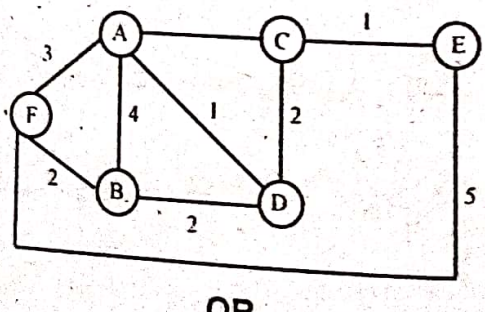


Any particular DFS or BFS of a directed or un-directed graph, each edge gets classified as one of the above.

In a DFS of an undirected graph, every edge is either a tree edge or back edge. Here no cross edge goes to a higher numbered or rightward vertex. The reason behind the DFS algorithm is so important is that it defines a very nice ordering of edges of the graph.

Long Answer Type Questions

1. a) Compare BFS and DFS. Discuss the two different ways of representing a graph. [WBUT 2009]



Describe BFS algorithm. OR, [WBUT 2016]

Answer:

1st Part:

Depth First Se

ordered tree. For

- **Rule 1:** If p on the stack.
- **Rule 2:** If R from it.
- **Rule 3:** Repe

Breadth First S

ordered tree.

For a given graph

Rule 1: Visit al vertex, and insert

Rule 2: If there i current vertex and

Rule 3: Repeat R Breadth-first searc explores all the ne unexplored neighb The time complex edge will be explo Depth-first search or graph. Intuitive case) and explores complexity is also

2nd Part:

There are three wa

i) Adjacency ma

ii) Adjacency list

iii) Adjacency mu

i) ***Adjacency ma***

array with n x

For undirected gra

A [i, j] = 1, if there

A [i, j] = 0, if there

Answer:

1st Part:

Depth First Search: DFS of a graph is analogous to the per-order traversal of an ordered tree. For a given graph the DFS will have the following rules.

- **Rule 1:** If possible, visit an adjacent unvisited vertex, mark it visited, and push it on the stack.
- **Rule 2:** If Rule 1 fails, then if possible pop a vertex off the stack follow Rule 1 from it.
- **Rule 3:** Repeat Rule 1 and Rule 2 until the all the vertices are visited.

Breadth First Search: BFS of a graph is analogous to level-by-level traversal of an ordered tree.

For a given graph the BFS will have the following rules.

Rule 1: Visit all the **next** unvisited vertices (if any) that is adjacent to the current vertex, and insert them into a queue one at a time on every visit.

Rule 2: If there is no unvisited vertex, remove a vertex from the Queue and make it the current vertex and then follow Rule 1.

Rule 3: Repeat Rule 1 and Rule 2 until the all the vertices visited.

Breadth-first search (BFS) is a graph search algorithm that begins at the root node and explores all the neighboring nodes. Then for each of those nearest nodes, it explores their unexplored neighbor nodes, and so on, until it finds the goal.

The time complexity can also be expressed as $O(|E| + |V|)$ since every vertex and every edge will be explored in the worst case.

Depth-first search (DFS) is an algorithm for traversing or searching a tree, tree structure, or graph. Intuitively, one starts at the root (selecting some node as the root in the graph case) and explores as far as possible along each branch before backtracking. The time complexity is also given by $O(|E| + |V|)$.

2nd Part:

There are three ways by which graphs can be represented in memory.

i) Adjacency matrices.
ii) Adjacency list.
iii) Adjacency multilists

i) ***Adjacency matrices:*** For a graph G with n vertices the adjacency matrix is a 2D array with n x n element.

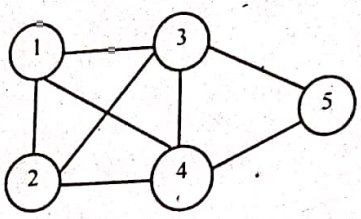
For undirected graphs

A [i, j] = 1, if there is an edge between i & j
A [i, j] = 0, if there is no edge between i & j.

For directed graphs

A [i, j] = 1, if there is an edge directed i & j

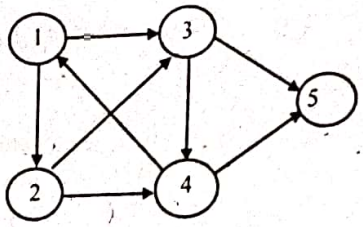
A [i, j] = 0, otherwise



Undirected graph

Adjacency Matrix

	1	2	3	4	5
1	0	1	1	1	0
2	1	0	1	1	0
3	1	1	0	1	1
4	1	1	1	0	1
5	0	0	1	1	0



Directed graph

	1	2	3	4	5
1	0	1	1	0	0
2	0	0	1	1	0
3	0	0	0	1	1
4	1	0	0	0	1
5	0	0	0	0	0

The adjacency matrix for an undirected graph is symmetrical i.e., diagonal elements are zero.

Advantage: It is possible to determine whether an edge is connecting two vertices or not.

Disadvantage:

- It requires n x n memory locations. If the graph is not complete, there will be a number of zero entries & this will waste the memory area. To avoid this, the adjacency list method is used.
- There will be two entries for the same edge. This information is redundant.

ii) ***Adjacency List:*** It is a linked list representation of a graph where each vertex represents a head node. Each head node has a list associated with it which represents the edges. The nodes of the list form a path between the head node and all other reachable vertices in the graph from that head node.

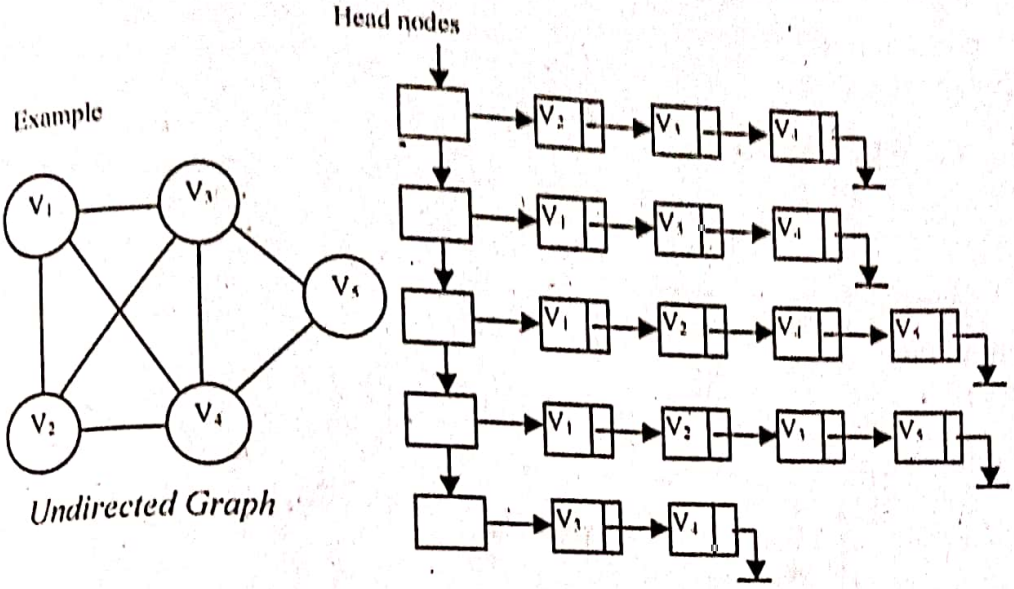
Examp

Undir

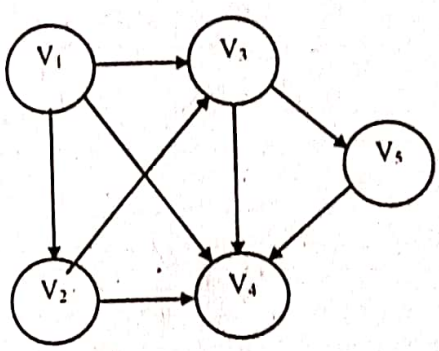
Drawbacks: Whe
pointers, which ca

iii) ***Adjacency M***
graph, it is cle
and another
information pr
To avoid this, the
shared amongst the
In this method for
will provide the inf
The node structure

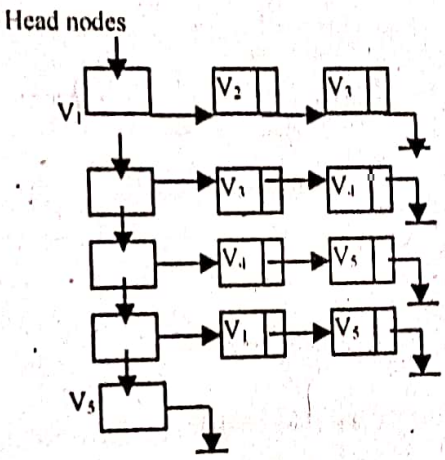
m --- tagfield
V_1, V_2 --- Vertex
P_1, P_2 ----- Incident



Undirected Graph



Directed graph



Drawbacks: When a graph is large, it is necessary to handle to many numbers of pointers, which can be complex.

iii) ***Adjacency Multilist:*** From the adjacency list representation of an undirected graph, it is clear that each edge (Vi, Vj) is represented by two entries, one is list Vi and another is 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	V_1	V_2	P_1	P_2

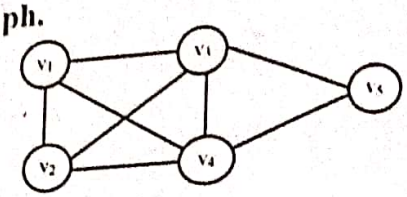
m --- tagfield

V_1, V_2 ---Vertex

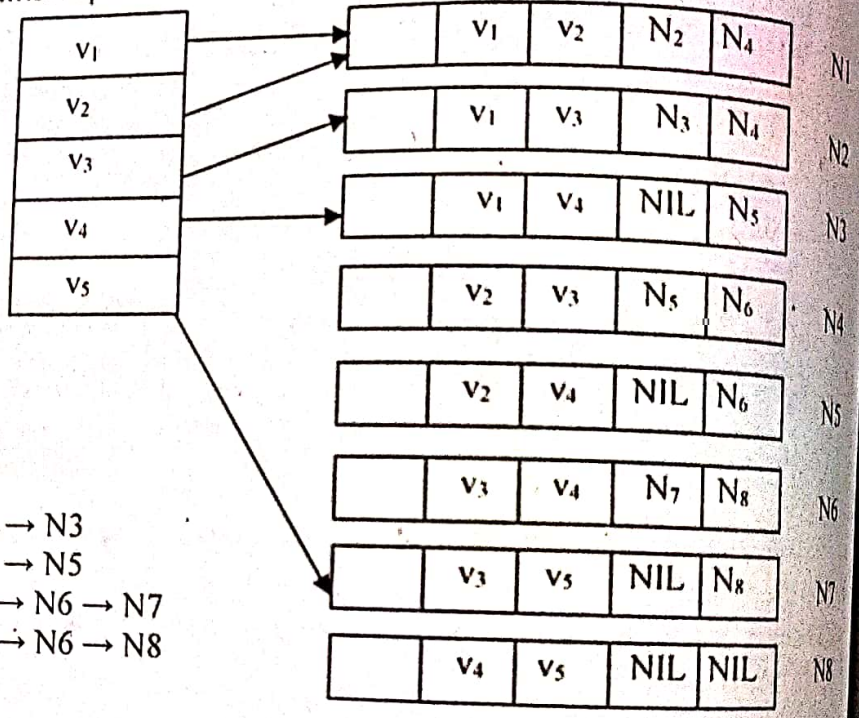
P_1, P_2----- Incident paths

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

Consider the following graph.



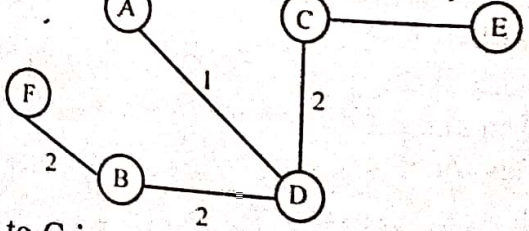
Total distinct edges are:
$\{v_1, v_2\}$, $\{v_1, v_3\}$, $\{v_1, v_4\}$, $\{v_2, v_3\}$, $\{v_2, v_4\}$, $\{v_3, v_4\}$, $\{v_3, v_5\}$ & $\{v_4, v_5\}$. All other edges represent the same information, because this is an undirected graph.
The adjacency multilist representation is as follows



The lists are:
Vertex 1 : N1 → N2 → N3
Vertex 2 : N1 → N4 → N5
Vertex 3 : N2 → N4 → N6 → N7
Vertex 4 : N3 → N5 → N6 → N8
Vertex 5 : N7 → N8

b) Draw the minimum cost spanning tree for the graph given below and also find its cost. [WBUT 2009]

Answer:
The tree has edges AD, BD, CD, CE and BF.
Cost is 8.



Note: Since the cost of A to C is not given in the question it is assumed that the cost is ≥5.

c) What is Comp
Show that the s

Show that the nu

Prove that the nu

Answer:
1st **Part:** A graph
vertices.
2nd **Part:** The num
Degree of a vertex
loop counted twic
summation is an
calculate degree o

that is nothing but

2. A rat has ente
boxes are open
which the rat can
representation of
algorithm.
Answer:
Let the checkerboar
4×4 checkerboard.

The maze can be re
obstacles, the rat wil
the diagonal.
The algorithm will be
Steps:
i←0; j←0
1. Start with A(i, j)
2. While (i < n, j < n)
i = i + 1
j = j + 1
3. Final position A(n,

c) What is Complete Graph?
Show that the sum of degree of all the vertices in a graph is always even. **[WBUT 2009]**

OR,

Show that the number of vertices of odd degree in a finite graph is even. **[WBUT 2009]**

OR,

Prove that the number of odd degree vertices in a graph is always even. **[WBUT 2017]**

[WBUT 2018]

Answer:

1st Part: A graph is said to be complete if there exist an edge between every pair of vertices.

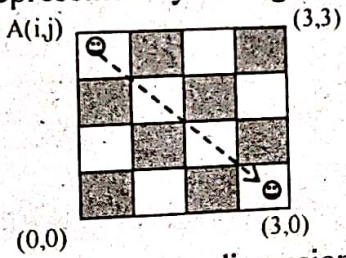
2nd Part: The number of odd degree vertices in a graph is always even:

Degree of a vertex is defined as no. of edges incident on a particular vertex with the self loop counted twice. Now, let us take the sum of degree of all the vertices. Now, this summation is an even number, because each edge contributes twice when we are calculate degree of different vertices. Now, if we take the summation of all the degrees that is nothing but the twice the number of edges $\sum_{i=1}^{n} d(v_i) = 2e \Rightarrow even\ number$

2. A rat has entered a checkerboard maze through one corner, whose the white boxes are open and black boxes represent obstacles. Develop an algorithm by which the rat can exit the maze through the opposite corner. Clearly explain the representation of the maze and any specific data structure you have used for the algorithm. **[WBUT 2011]**

Answer:

Let the checkerboard maze be represented by the figure below. For simplicity we show a 4×4 checkerboard.



The maze can be represented by a $n \times n$ two dimensional array. Since black boxes are obstacles, the rat will only travel through white boxes. So the best path to follow will be the diagonal.

The algorithm will be as follow:

Steps:

i←0; j←0

1. Start with A(i, j)
2. While (i < n, j < n)
   i = i + 1
   j = j + 1
3. Final position A(n, n)

3. Write short notes on the following: [WBUT 2010]
a) BFS vs DFS [WBUT 2012]
b) BFS [WBUT 2013]
c) Dijkstra's Algorithm [WBUT 2014, 2016]
d) DFS in graph

Answer:

a) BFS vs. DFS: *Refer to Question No. 1(a) of Long Answer Type Questions.*

b) BFS: *Refer to Question No. 1(a) of Long Answer Type Questions.*

c) Dijkstra's Algorithm:

Dijkstra's algorithm works on the principle that the shortest possible path from the source has to come from one of the shortest paths already discovered. A way to think about this is the "explorer" model--starting from the source, we can send out explorers each travelling at a constant speed and crossing each edge in time proportional to the weight of the edge being traversed. Whenever an explorer reaches a vertex, it checks to see if it was the first visitor to that vertex: if so, it marks down the path it took to get to that vertex. This explorer must have taken the shortest path possible to reach the vertex. Then it sends out explorers along each edge connecting the vertex to its neighbors.

It is useful for each vertex of the graph to store a "prev" pointer that stores the vertice from which the "explorer" came from. This is the vertex that directly precedes the current vertex on the path from the source to the current vertex.

The pseudocode for Dijkstra's algorithm is fairly simple and reveals a bit more about what extra information needs to be maintained. Vertices will be numbered starting from 0 to simplify the pseudocode.

```
Given a graph, G, with edges E of the form (v1, v2) and vertices
V, and a
source vertex, s
dist : array of distances from the source to each vertex
prev : array of pointers to preceding vertices
i    : loop index
F    : list of finished vertices
U    : list or heap unfinished vertices

/* Initialization: set every distance to INFINITY until we
discover a path */
for i = 0 to |V| - 1
   dist[i] = INFINITY
   prev[i] = NULL
end
/* The distance from the source to the source is defined to be
zero */
dist[s] = 0
/* This loop corresponds to sending out the explorers walking the
paths, where
```



```
* the step of
s" corresponds
* to an explor

while(F is mis
   pick the ve
   add v to F
   for each ed
      /* The
      "relaxat
         if(di
         dist[
         prev[
         possi
            en
         end f
end while
```

d) DFS in graph:

```
* the step of picking "the vertex, v, with the shortest path to
s" corresponds
* to an explorer arriving at an unexplored vertex */

while(F is missing a vertex)
  pick the vertex, v, in U with the shortest path to s
  add v to F
  for each edge of v, (v1, v2)
    /* The next step is sometimes given the confusing name
    "relaxation"
      if(dist[v1] + length(v1, v2) < dist[v2])
      dist[v2] = dist[v1] + length(v1, v2)
      prev[v2] = v1
      possibly update U, depending on implementation
         end if
      end for
end while
```

d) DFS in graph: ***Refer to Question No. 1(a) of Long Answer Type Questions.***