MODULE - IV

TREES

We will study in this chapter a class of graphs called trees.

Let us start our topic by taking an example.

Ex A group of boxers contending for the title of heavyweight champion of the world. Suppose that each boxer has one chance to challenge the champion, and the loser of any match will be eliminated from the competition.

Let $G = (V, E)$ be an undirected graph.

where V represents the boxers (vertices) & E represents the matches (Edges).

i.e. for a, b in V, $\{a, b\}$ is in E if there was a match between a & b.

Let $V = \{a, b, c, d, e, f, g, h, i\}$.

where $a \rightarrow$ be the champion,

& in the subsequent challenge matches

a beat b, c & d.

& then lost the title to e.

e beat f & g & then lost the title to h.

& finally h lost the title to i.

The following graph shows all the matches that took place

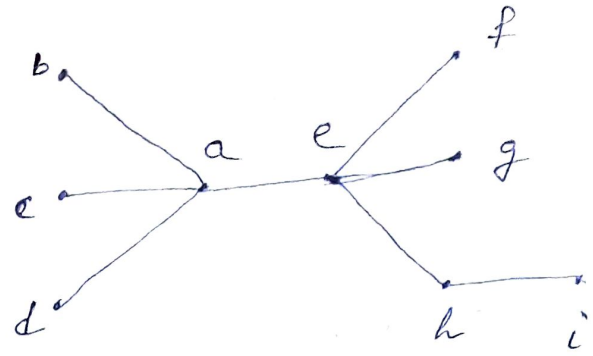


Fig - (a)

Let us consider another example

Consider four gossipy couples

$\{a, A, b, B, c, C, d, D\}$

where $\{a, b, c, d\}$ are husbands & $\{A, B, C, D\}$ are their wives respectively.

Suppose a calls her wife to tell her some gossip & she then calls the other wives to spread the gossip. & each of them in turn calls her husband about it. The following graph shows how the gossip spread. where the edges represent the telephone calls.

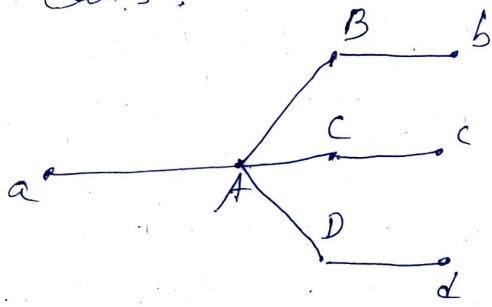


Fig - (b)

A tree is said to be connected (undirected) that contains no simple circuit.

The graphs in fig (a) & (b) are trees.

A collection of disjoint trees is called a forest.

A vertex of degree 1 in a tree is called a leaf or a terminal node, and a vertex of degree larger than 1 is called a branch node or an internal node.

In Fig (a) b, c, d, f, g, & i are leaves & the vertices a, e & h are branch nodes.

Let us note first some properties of trees.

(1) There is a unique path between every two vertices in a tree.

(2) The number of vertices is one more than the number of edges in a tree.

(3) A tree with two or more vertices has at least two leaves.

Property (1) follows directly from the definition of a tree. Because a tree is a connected graph, there is at least one path between every two vertices.

If there were two or more paths between a pair of vertices there would be a circuit in the tree.

To prove property (2) we can use induction method which is quite easy

property (3) follows from property (2)

"The sum of the degrees of the vertices in a graph is equal to $2e = 2(v-1)$ in a tree.

A tree with more than one vertex can't have any isolated vertex, there must be at least two vertices of degree 1 in the tree.

Thus we can say the following :

(1) A graph in which there is a unique path between every pair of vertices is a tree.

(2) A connected graph with $e = v-1$ is a tree.

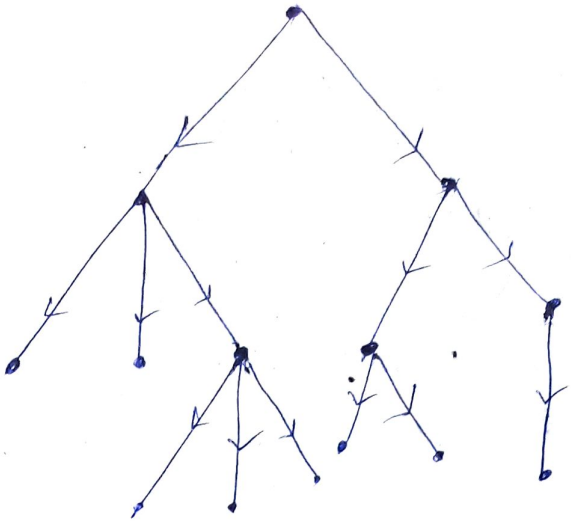
(3) A graph with $e = v-1$ that has no circuit is a tree.

ROOTED TREES

A directed graph is said to be a a directed tree if it becomes a tree when the directions of the edges are ignored.

A directed tree is called a rooted tree if there is exactly one vertex whose incoming degree is 0 & the incoming degrees of all other vertices are 1.

The vertex with incoming degree 0 is called the root of the rooted tree.



(A Rooted tree)

In a rooted tree, a vertex whose outgoing degree is 0 is called a leaf or a terminal node & a vertex whose outgoing degree is nonzero is called a branch node or an internal node.

Let a be a branch node on the tree T. By the subtree with a as the root we mean the subgraph $T' = (V', E')$ of T such that V' contains a & all of its descendants & E' contains the edges on all directed paths from a. By a subtree of a we mean a subtree that has a son of a as root.

An ordered tree is a rooted tree with the edges incident from each branch node labeled with integers $1, 2, \ldots, i, \ldots$

Two ordered trees are said to be isomorphic if there is one to one correspondence between their vertices & edges that preserves the incidence relation & if the labels of correspondent edges match.

An ordered tree in which every branch node has at most m sons is called an m-ary tree.

An m-ary tree is said to be regular if everyone of its branch node has exactly m sons.

An important class of m-ary is binary tree.

Binary search Trees

Binary search tree is a tree in which each child is either a left or right child, no vertex has more than one left child & one right child and the data are associated with vertices.

Ex:→ Suppose a friend has picked a number between 1 & 99, and we want to determine what this number is by asking questions of the form.

"Is the number 50?"

If we are told that the number is 50, we are done.

If we are told that the number is less than 50, we have narrowed down the range. Centering the number to 1 through 49.

Our next question is "Is the number 25?"

If the number lies between 51 & 99.

Our next question is,

"Is the number 75?"

In this way we can find out the number by asking a sequence of questions & reducing the range.

Decision Trees

A rooted trees in which each internal vertex is assigned to a decision with a sub-tree at the vertices, then each possible outcome of the decision is called a decision tree.

Traversal of a Tree

A systematic method for visiting every vertex of an ordered rooted tree is called as a Traversal algorithm.

MINIMUM SPANNING TREES

A minimum spanning tree is one with minimum weights.

The weight of a spanning tree is defined to be the sum of the weights of the branches of the tree.

There is a technique to find out minimum spanning tree proposed by Joseph Kruskal in 1956. So this is known as Kruskal's Algorithm.

KRUSKAL'S ALGORITHM

1→ List all the edges of graph G in the increasing order of weights.

2→ select the smallest edge (having minimum weight) from the list & add it to the spanning tree, (which is initially empty), if the inclusion of the edge doesn't make a circuit.

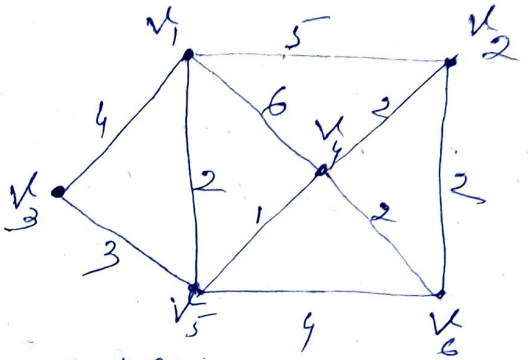
If the selected edge makes a circuit, then remove it from the list.

(3) Repeat the steps 1 & 2 until the tree contains $v-1$ edges (where v is the number of vertices) or the list is empty.

(4) Now if the tree contains less than $v-1$ edges & the list happened to be empty then no spanning tree is possible, else it gives the minimum spanning tree.

Now let us take following example to understand Kruskal's Algorithm better.

Ex



This is a weighted graph

Step-1 :- We have to list all the edges in the increasing order of weights.

Edges in the increasing order of their weights	weights of edges
$v_4 - v_5$	1
$v_1 - v_5$	2
$v_2 - v_4$	2
$v_4 - v_6$	2
$v_2 - v_6$	2
$v_3 - v_5$	3
$v_1 - v_3$	4

$v_5 - v_6$	4
$v_1 - v_2$	5
$v_1 - v_4$	6

Step-2 :⇒

Iteration-1 ⇒ Consider the edge (v_4, v_5) having smallest weight of 1 from the table & add it to the spanning tree.

Iteration-2 :⇒

Now consider the edges $(v_1 - v_5)$, $(v_2 - v_4)$, (v_4, v_6), & $(v_2 - v_6)$ from the table & add it to the spanning tree by taking into consideration that it must not form a loop.

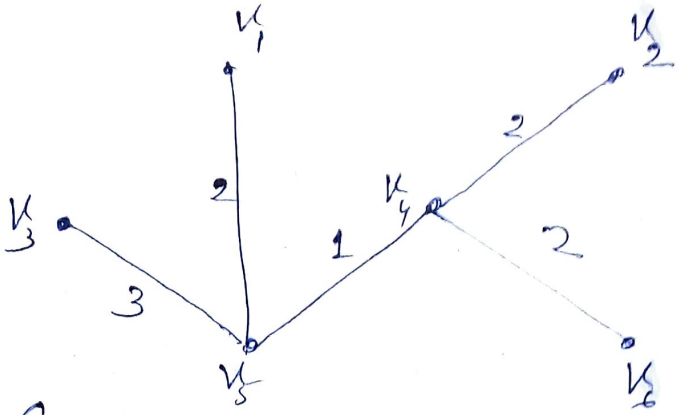
But considering $(v_2 - v_6)$ we would get a loop. Hence we must not take this edge. & delete it from our table

Iteration-3 ⇒

Consider the edge (v_3, v_5), having smallest weight 3.

Iteration-4 :⇒

The algorithm stops as the tree already contains $v-1$ number of edges where v is the number of vertices.



(Minimum spanning tree)

Edges	Weights	selection of the edges
$v_4 - v_5$	1	Yes
$v_1 - v_5$	2	Yes
$v_2 - v_4$	2	Yes
$v_4 - v_6$	2	Yes
$v_2 - v_6$	2	No
$v_3 - v_5$	3	Yes
$v_1 - v_3$	4	
$v_5 - v_6$	4	
$v_1 - v_2$	5	
$v_1 - v_4$	6	

PRIM's ALGORITHM

(1) Draw n isolated vertices and label them as $v_1, v_2 \ldots v_n$ where n is the number of vertices.

(2) Represent the given weights of the edges of graph G in an $n \times n$ matrix.

(3) Assign the weights of non-existent edges (e.g. corresponding to pairs of towns between which no direct road can be constructed) as very high.

(4) Start from vertex v_1 & connect it to its nearest neighbours (i.e. to the vertex which has the smallest entry in row 1 of the matrix), say v_k. If more than one smallest entry is there, then arbitrarily select one of them.

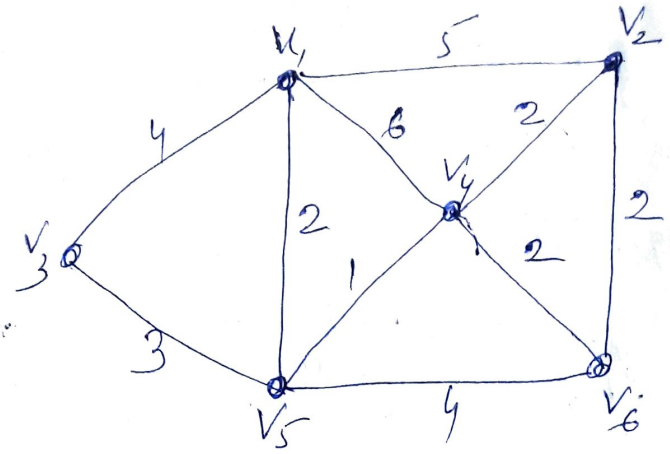
(5) Consider v_1 & v_k as one subgraph and connect the subgraph to its closest neighbours (i.e. to a vertex other than v_1 and v_k which has the smallest entry in rows 1 & k). Suppose this new vertex be v_i.

(6) Consider the tree with vertices v_1, v_k & v_i as one subgraph & continue the process until all the n vertices have been connected by $(n-1)$ edges.

(7) Exit.

Now let us take an example to explain the working of Prim's Algorithm.

Ex ⇒



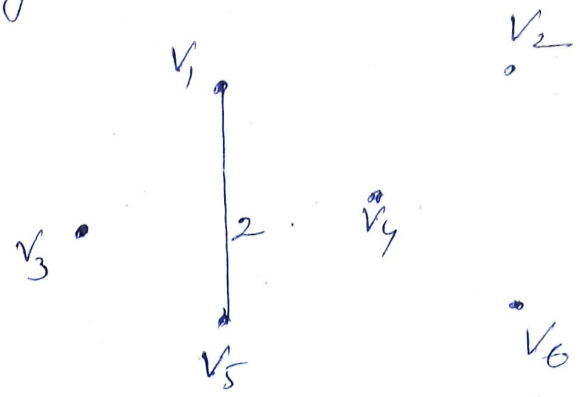
(A weighted graph)

Step-1

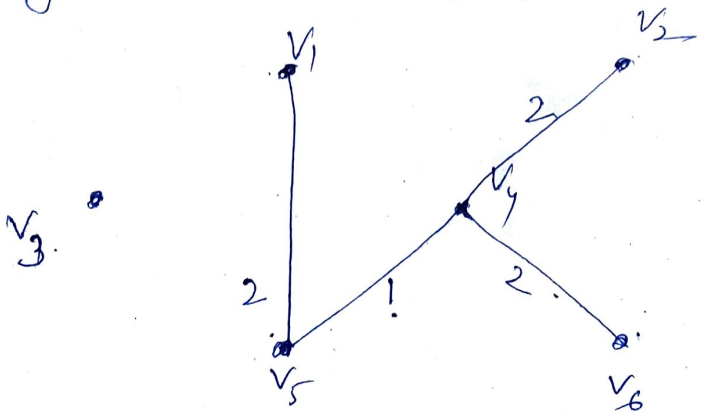
	V_1	V_2	V_3	V_4	V_5	V_6
V_1	—	5	4	6	2	—
V_2	5	—	—	2	—	2
V_3	4	—	—	—	3	—
V_4	6	2	—	—	1	2
V_5	2	—	3	1	—	4
V_6	—	2	—	2	4	—

(Adjacency matrix)

Step-2 → We have to start from vertex V_1 & connect it to the vertex having lowest weight, which is V_5.

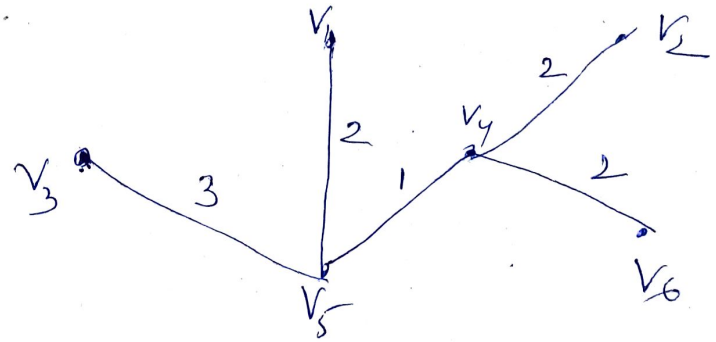


Step-3 → Then consider the next edges having lowest weights.



But we can't connect (V_2, V_6) as it will make a loop.

Step-4 → Now consider the vertices V_1, V_5, V_4, V_2, V_6 as one subgraph & connect it to the vertex V_3 as it has the smallest entry other than the vertices V_1, V_5, V_4, V_2, V_6.



Now the algorithm exists as there are already 5 edges connecting the 6 vertices.