GRAPH THEORY

Module 3

Rijin IK

Assistant Professor
Department of Computer Science and Engineering
Vimal Jyothi Engineering College
Chemperi

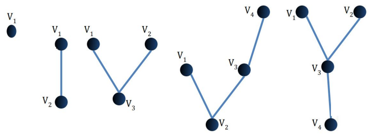
April 8, 2023

Outline

Trees

TREES

- A tree is a connected graph without any circuits.
- A tree has to be simple graph, that is having neither a self-loop nor parallel edges.
- A single vertex can also be considered as a tree



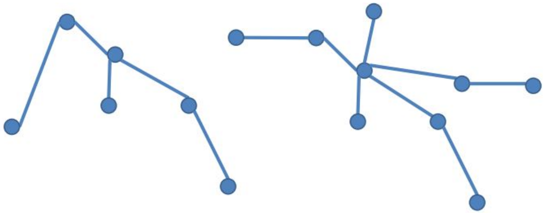
Trees can be used to represent and manipulate real life instances like

- The genealogy of a family
- A river with its tributaries and sub tributaries.
- Sorting of mail according to zip code.

Trees

Minimally connected Graph

- A graph G is said to be minimally connected if removal of any one edge from G, leaves it disconnected



Trees

PROPERTIES OF TREES

1. A tree is a connected graph & circuit-less
2. There is exactly one path between every pair of vertices of a tree
3. A tree with n vertices will have (n-1) edges
4. A tree is a minimally connected graph
5. A tree will have at least 2 pendant vertices
6. A tree can have only 1 or 2 centers

Trees

THEOREM 3.1

There is one and only one path between every pair of vertices in a tree, T.

Proof:

- Since T is a connected graph, there must exist at least one path between every pair of vertices in T.
- Now, suppose that between two vertices, say Vi & Vj , there exists two distinct paths.
- Then union of these two paths will result in a circuit. But since T is a tree, it cannot contain any circuits.
- Hence there could be no two distinct paths between any pair of vertices in T.
- Hence the theorem.

Trees

THEOREM 3.2

If, in a graph G, there is one and only one path between every pair of vertices, G is a tree.

Proof:

- Since there is at least one path between every pair of vertices in G, we can say that G is connected.
- Now, if the graph has a circuit in it, there must be two vertices vi & vj such that there exists two distinct paths between vi & vj
- But since there is only one path between every pair of vertices, G cannot contain any circuit
- Since G is connected & contains no circuit, we can say that G is a tree

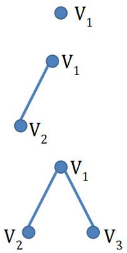
Trees

THEOREM 3.3

A tree with n vertices has n-1 edges

Proof:

- **By induction we prove that the theorem is true.**
- For n=1
  - No. of edges = n-1= 1-1= 0
  - Which is true as we can see in the figure
- For n=2
  - No. of edges = n-1 = 2-1=1
  - Which is true
- For n=3
  - No. of edges = n-1 = 3-1=2
  - true
- Now we assume that the theorem is true for all trees with no. of vertices up to (n-1)



Trees

Proof cont..:

- We need to prove that the theorem holds for a tree with n vertices
  - Consider a tree with n vertices; Remove an edge e_k from the tree; Let v_i & v_j be the end vertices of e_k
  - Removal of e_k will disconnect the graph into 2 components say T1 and T2 each of which is again a tree
  - Let the component T1 contain n1 vertices; then no. of edges in $T1 = (n1 - 1)$
  - Let the component T2 contain n2 vertices; then no. of edges in $T2 = (n2 - 1)$
  - Since n1 and n2 $<$n, the theorem holds, as per our assumption
  - Also n1 $+$n2 $=$ n (total no. of vertices in the graph)
  - Now, add back the deleted edge ek so that the graph again becomes connected
  - The resulting no. of edges in the graph is given by $(n1 - 1) + (n2 - 1) + 1 = n1 + n2 - 1 - 1 + 1 = n - 1$
  - Hence a tree with n vertices will have (n-1) edges

Trees

THEOREM 3.4

Any connected graph with n vertices and (n-1) edges is a tree

Proof:

- The minimum no. of edges required to make a connected graph with n vertices is (n-1)
- In such a graph, removal of any 1 edge leaves the graph disconnected
- If the graph had a circuit, there would be at least one edge, removal of which do not make the graph disconnected.
- But there is no such edge at all. Hence the graph is circuit-less
- Since the graph is connected & circuit-less, it is a tree

Trees

THEOREM 3.5

A graph is a tree if and only if it is minimally connected

Proof:

- **Suppose the graph is minimally connected.**
- A minimally connected graph, cannot contain any circuits because if it contained a circuit, then there must be at least one edge removal of which do not make the disconnected.
- But all the edges of the graph are such that, removal of any edge, leaves the graph disconnected
- Hence the graph has no circuits & it is connected. So it must be a tree

Trees

Proof Cont..:

- **Conversely: Let the graph be a tree**
- Since it is a tree, it is connected & circuit-less
- And also there is only & only one path b/w every pair of vertices
- Removal of any edge from a path leaves the graph disconnected, hence we can say that the graph which is a tree, is always minimally connected

Trees

THEOREM 3.6

A graph G with n vertices , (n-1) edges & no circuits is connected

Proof:

- Suppose there exists a graph with n vertices, n-1 edges & no circuits, but it is disconnected.
- Then G may contain 2 or more circuit-less components
- Now our disconnected graph G has total n vertices & n-1 edges
- Now add an edge e_k between vertex v_i from one component & vertex v_j from another component
- Adding of this edge will not create circuit as v_i & v_j were from different components & there wasn't any path between them
- Now, our graph must contain n vertices & n edges, which is not possible for a tree.
- Hence the graph G must be connected

Pendant vertices in a tree

THEOREM 3.7

In any tree(with two or more vertices), there must be at least 2 pendant vertices

Proof:

- A tree with n vertices will have (n-1) edges
- As each edge contributes 2, to the total degree of the graph, total degree of the tree is 2(n-1) = 2n-2
- Which shows all n vertices may take degree 2 each, except 2.
- Two of the vertices will get only degree 1 each
- Hence there will be at least 2 pendant vertices for any tree

Trees

AN APPLICATION OF TREES

Problem:

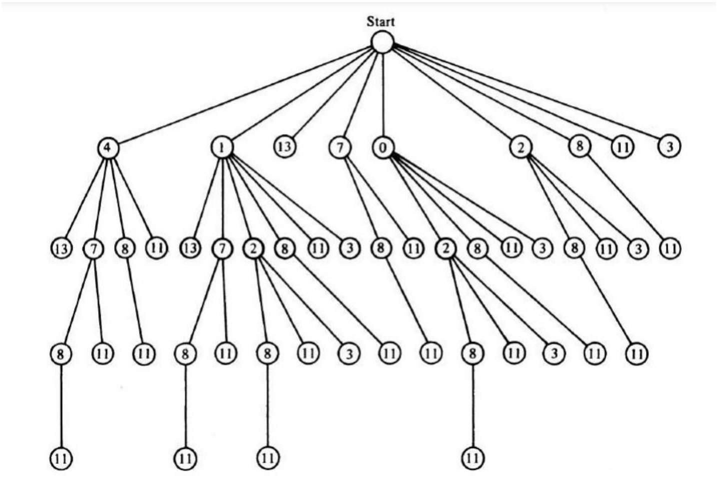
Given a sequence of integers, no two of which are the same, find the largest monotonically increasing subsequence in it.

Solution:

- Suppose if the given sequence is 4,1,13,7,0,2,8,11,3.
- Represent it by a tree where each sequence number in the sequence is a descendant vertex of a root vertex.
- Now, the path from the start vertex to a particular vertex, v gives the monotonically increasing subsequence ending in v.

Trees

AN APPLICATION OF TREES



Trees

AN APPLICATION OF TREES

- For the given sequence, the length of the largest such subsequence is four
- And there are 4 different subsequence, each of length four
- They are (4,7,8,11) , (1,7,8,11) , (1,2,8,11) & (0,2,8,11)
- Such trees are called data trees by computer programmers

Trees

Q. Find the largest monotonically increasing subsequence of the given sequence 5, 0, 1, 7, 6, 8, 2, 9, 4, 3

Path length

Length of a path:

- Length of the path between any two vertices of a graph is simply the no. of edges included in the path

Shortest path:

- If more than one path is present between any 2 vertices, then the path with the minimum no. of edges is the shortest path.

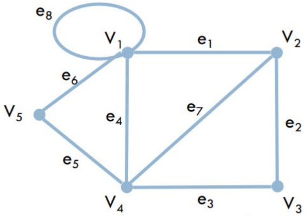
Distance:

- Distance between any 2 vertices, $v_i \& v_j$ is represented as d$(v_i \& v_j)$ and is given by the length of the shortest path between them
  - * In a normal connected graph, to find the distance b/w any 2 vertices, we have to enumerate all available paths & pick the one with the shortest length.
  - * However **in a tree, there is only path b/w every pair of vertices; so distance between any 2 vertices is just the length of the path b/w them**

Path length

Example: distance b/w two vertices In graph G, available paths between $v_5 \& v_2$ are

- $v_5\ e_6\ v_1\ e_1\ v_2$
- $v_5\ e_5\ v_4\ e_7\ v_2$
- $v_5\ e_6\ v_1\ e_8\ v_1\ e_1\ v_2$
- $v_5\ e_5\ v_4\ e_3\ v_3\ e_2\ v_2$
- $v_5\ e_5\ v_4\ e_4\ v_1\ e_1\ v_2$
- $v_5\ e_5\ v_4\ e_4\ v_1\ e_8\ v_1\ e_1\ v_2$
- $v_5\ e_6\ v_1\ e_4\ v_4\ e_7\ v_2$
- $v_5\ e_6\ v_1\ e_4\ v_4\ e_3\ v_3\ e_2\ v_2$



- Two shortest paths are available between $v_5 \& v_2$
- Each contain two edges
- Hence $d(v_5, v_2) = 2$

Trees

Metric

A function f(x,y) of two variables is called a metric if it satisfies the following 3 conditions

1. Non-negativity
   - $f(x, y) \geq 0 \ \forall x, y$
   - $f(x, y) = 0 \textit{ iff } x = y$
2. Symmetry
   - $f(x, y) = f(y, x)$
3. Triangle inequality
   - $f(x, y) \leq f(x, z) + f(z, y) \textit{ for any } z$

Trees

Theorem

The distance between the vertices of a connected graph is a metric

Proof:

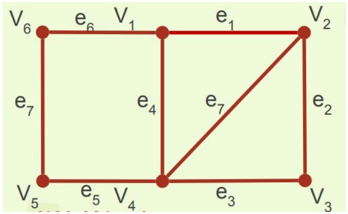
- Checking the conditions of a metric for the distance between any 2 vertices of a connected graph
- Non negativity: Distance b/w any 2 vertices is never negative. Distance from one vertex to itself is zero (simple graph)
- Symmetry: Distance from one vertex to another vertex will be the same in the reverse direction (undirected graph)
- Triangle inequality: Since distance b/w any 2 vertices $d(v_i \& v_j)$ is the shortest path between them, there cannot be any shorter path that goes through some vertex vk such that $d(v_i \& v_j) \leq d(v_i \& v_k) + d(v_k \& v_j)$
- Hence distance between the vertices of a connected graph is a metric

ECCENTRICITY OF A VERTEX

Eccentricity E(v) of a vertex v in a graph G is the distance from v to the farthest available vertex in G
E(v) $=$ max d(v&v_i) for every vertex v_i of G

Eg:

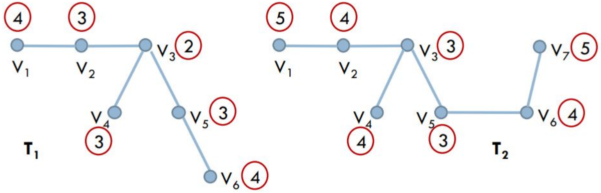
E(v2)=2	**E(v5)=2**	**E(v1)=2**
d(V2,V1) $= 1$	d(V5,V1) $= 2$	d(V1,V2) $= 1$
d(V2,V3) $= 1$	d(V5,V2) $= 2$	d(V1,V3) $= 2$
d(V2,V4) $= 1$	d(V5,V3) $= 2$	d(V1,V4) $= 1$
d(V2,V5) $= 2$	d(V5,V4) $= 1$	d(V1,V5) $= 2$
d(V2,V6) $= 2$	d(V5,V6) $= 1$	d(V1,V6) $= 1$



Trees

Center of a tree

- **The vertex with the minimum eccentricity is considered as the center of the tree**
- Some trees may have 2 centers. They are called bi-centered
- Every tree will have either one or two centers

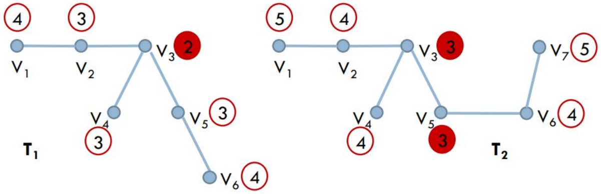


Trees

Radius of a tree

The eccentricity of the center of the tree is regarded as the radius of the tree

- In tree T1, radius is 2
- In tree T2, radius is 3



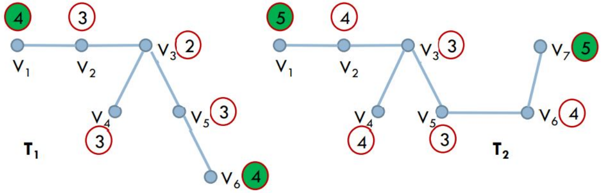
Trees

Diameter of a tree

The length of the longest path in the tree is the diameter of the tree

Note: It is not necessary that the diameter be twice its radius

- In tree T1, diameter is 4 (radius is 2)
- In tree T2, diameter is 5 (radius is 3)



Trees

Theorem

Every tree has either one or two centers

Proof:

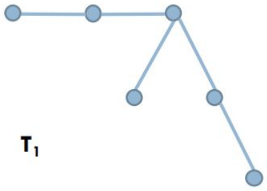
- Consider a tree with n vertices where $n > 2$
- A tree must have 2 or more pendant vertices
- Delete all pendant vertices from T.
  - The resulting graph is still a tree, say T'.
  - Deletion of the pendant vertices, will reduce the eccentricities of all remaining vertices by one.
  - Hence the center of the graph will remain the same.
  - From T' again remove all pendant vertices, to obtain another tree T". Again T" has the same vertex as its center
- continue this process until the remaining tree has either one vertex or one edge. (2 vertices)
  - So in the end, if one vertex is there this implies tree T has one center. If one edge is there then tree T has two centers.
- Hence any tree can have only 1 or 2 centers

Trees

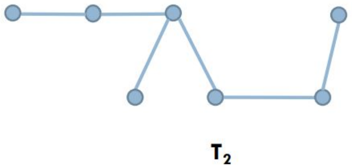
Proof Cont..

Corollary:

If a tree has 2 centers, then both of them must be adjacent



1 vertex is left,
which is the center
of the tree

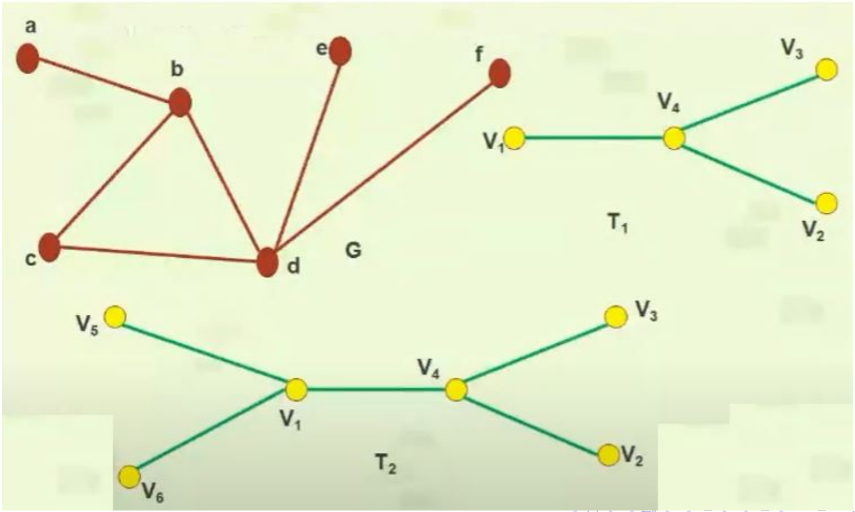


1 edge is left,
end vertices of
which are the
centers of the tree

Trees

Problem

- Find the eccentricity,Radius and Diameter

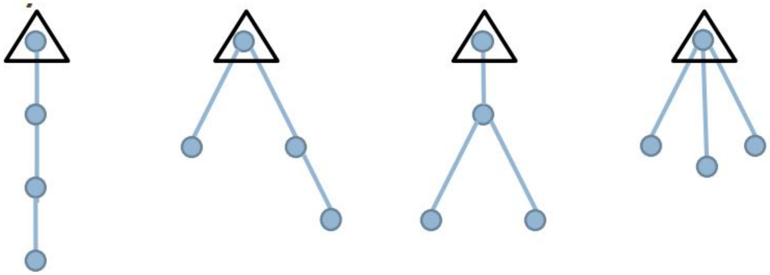


Rooted and binary tree

Rooted trees

- A tree in which one vertex (called the root) is distinguished from all others is called a rooted tree
- In a rooted tree, the root vertex is usually marked within a Δ
-

Eg: All possible rooted trees with four vertices

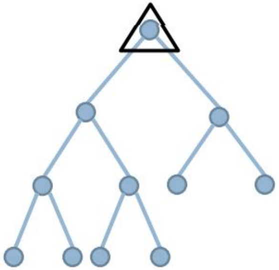


Rooted and binary tree

Binary trees

They are a special class of rooted trees where the root vertex is of degree 2 & all remaining vertices are of degree either 1 or 3

- Type of vertices in a binary tree:
  - **Root vertex:** The only vertex with degree 2
  - **Pendant vertices:** The vertices that are of degree 1
  - **Internal vertices:** The vertices that are of degree 3 (Non-pendant vertices)



Rooted and binary tree

Levels of a binary tree

In a binary tree, a vertex v_i is said to be at level L_i if v_i is at a distance of L_i

- The root is assumed to be at level 0 from the root
- Hence at level 0, there could be only 1 vertex
    - at level 1, at most $2^1 = 2$ vertices (either 0 or 2)
    - at level 2, at most $2^2 = 4$ vertices & so on
    - at level k, at most 2^k vertices
- Hence maximum no. of vertices possible in a k-level binary tree is

$$Vmax = 2^0 + 2^1 + 2^2 + \ldots\ldots + 2^k$$

- Minimum no. of vertices possible in a k-level binary tree is

$$Vmin = 1 + 2 + 2 + \ldots\ldots + 2(ktimes) = 2k + 1$$

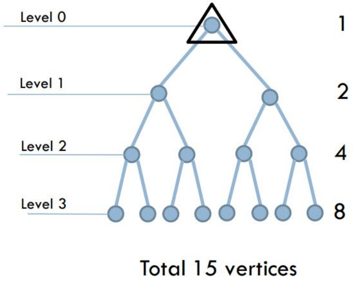
Rooted and binary tree

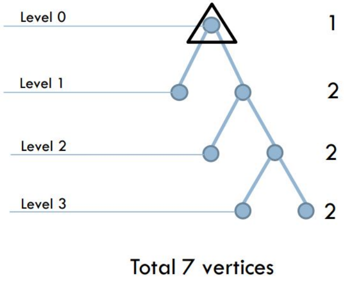
Example

- Here k=3

$$Vmax = 2^0 + 2^1 + 2^2 + 2^3 = 1 + 2 + 4 + 8 = 15 \ vertices$$

$$Vmin = 2 * 3 + 1 = 7 \ vertices$$





Rooted and binary tree

Height of a binary tree

Number of edges in longest path from root to a leaf node

or

The maximum level,Lmax of any vertex in a binary tree is called the height of the tree

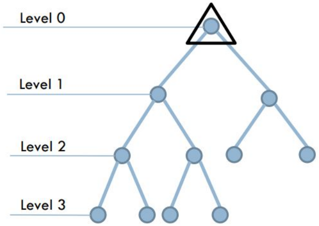
No. of levels possible in an ‘n vertex’ binary tree

- $Lmin = \lceil log_2(n+1) - 1 \rceil$
  - This is obtained when all levels are filled with maximum possible vertices (tightly packed)
- $Lmax = \frac{n-1}{2}$
  - This is obtained when the farthest vertex is made as far as possible from the root; all levels contain just 2 vertices (loosely packed)

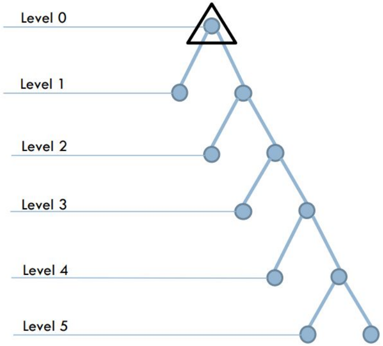
Rooted and binary tree

Consider a binary tree with n=11

- $L_{min} = \lceil \log_2(n+1)-1 \rceil = \lceil \log_2(12)-1 \rceil = 3$
- $L_{max} = \frac{(n-1)}{2} = 5$



Min level binary tree for n=11

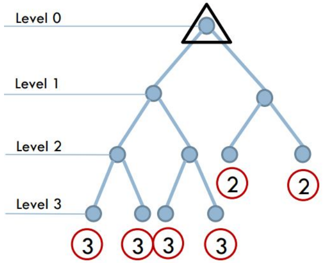


Max level binary tree for n=11

Rooted and binary tree

Path length of a binary tree

Path length of a tree is defined as the sum of the path lengths of the pendant vertices from the root (Sum of the levels of all pendant vertices)



$$P = 3 + 3 + 3 + 3 + 2 + 2 = 16$$

Rooted and binary tree

Weighted Path Length

Weighted path length of the binary tree is the sum of the weighted path lengths of the pendant vertices

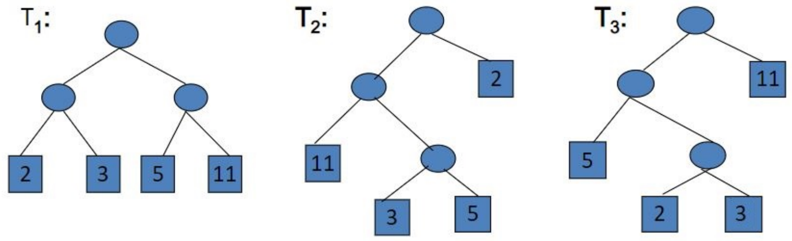
- All pendant vertices are associated with a positive real number, called weight w
- Path length of each pendant vertex is multiplied by the corresponding weight to obtain weighted path length of the pendant vertex

$$P_w = \sum w_i l_i$$

l_i *is the lavel of* pendant vertex V_i

Rooted and binary tree

Weighted Path Length



$P_1 = 2.2 + 3.2 + 5.2 + 11.2$

$= 42$

$P_2 = 2.1 + 11.2 + 3.3 + 5.3$

$= 48$

$P_3 = 11.1 + 5.2 + 2.3 + 3.3$

$= 36$

Rooted and binary tree

Problem:

Consider a coke machine that accepts coins. Suppose the machine lets only 4 types of coins to go through the slot, say pennies, nickels, dimes & quarters. The machine performs a sequence of tests to identify the coin. Probability of a coin being a penny, nickel, dime & a quarter are 0.05, 0.15, 0.5 & 0.30 respectively. For each type of coin, 1 test is done. Suppose all 4 tests take the same amount of time. The result of each test is yes or no. Then in what order must the tests be done, so as to minimize the overall coin determination time.

Rooted and binary tree

Solution

- Construct a binary tree with 4 pendant vertices, with their corresponding weight, as w1=0.05, w2= 0.15, w3= 0.5, w4 =0.3, such that the weighted path length is the minimum

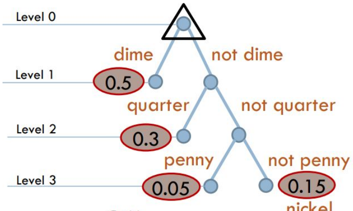


fig (a)

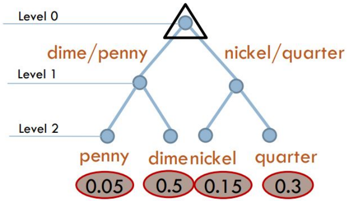


fig (b)

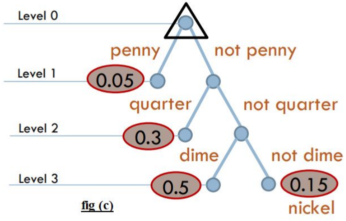


fig (c)

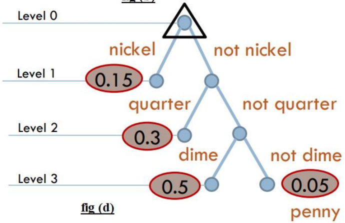


fig (d)

Rooted and binary tree

Solution Cont..

fig a

- $\sum w_i L_i = 1x0.5 + 2x0.3 + 3x0.05 + 3x0.15 = 1.7t$

fig b

- $\sum w_i L_i = 2x0.5 + 2x0.3 + 2x0.05 + 2x0.15 = 2t$

fig c

- $\sum w_i L_i = 1x0.05 + 2x0.3 + 3x0.5 + 3x0.15 = 2.6t$

fig d

- $\sum w_i L_i = 1x0.15 + 2x0.3 + 3x0.5 + 3x0.05 = 2.4t$

Hence the first binary tree would be our solution

Rooted and binary tree

Theorem

In a binary tree with n vertices, the number of pendant vertices, $P = \frac{(n+1)}{2}$

Proof:

- In a binary tree,
  - no. of root vertex = 1 & the degree of root vertex is 2
  - Degree of each pendant vertex is 1
  - Degree of each internal vertex is 3
  - In a binary tree with n vertices, the no. of edges = n-1
    - We know that the sum of the degrees of all vertices in a graph is twice the no. of edges

$$\begin{aligned} 2x1 + 1xp + 3x(n-p-1) &= 2(n-1) \\ 2 + p + 3n - 3p - 3 &= 2n - 2 \\ 2 - 3 + 2 + 3n - 2n &= 3p - p \\ 1 + n = 2p \rightarrow p &= \frac{(n+1)}{2} \end{aligned}$$

Rooted and binary tree

Theorem

The number of vertices 'n' in a binary tree is always odd

Proof:

- In a binary tree, only the root node has even degree
- In an 'n' vertex binary tree, remaining (n-1) vertices are of odd degree
- The no. of odd degree vertices in any graph is even
- Therefore the quantity (n-1) must always be even
- Then n is odd

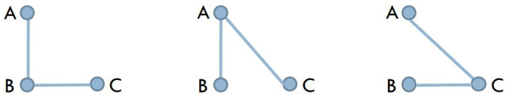
Counting trees

Counting trees

- The total number of trees that one can construct with a given number of vertices
- The count is different for labeled and unlabeled trees

Labeled trees

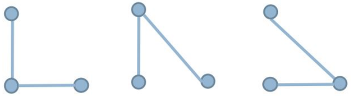
- A tree in which each vertex is assigned a unique label
  - According to Cayley's theorem, the no. of different labeled trees possible with n vertices is n^{n-2}
    - Eg: no. of labeled trees possible for n=3 is $3^{3-2} = 3$



Counting trees

Unlabeled trees

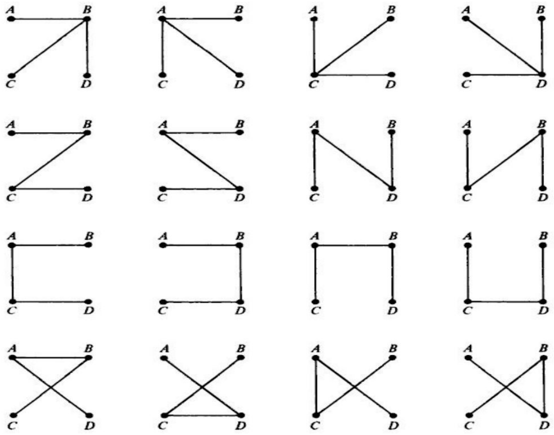
- A tree in which there is no distinction between the vertices by their name. However vertices differ in their degree
  - Eg: Count all unlabeled trees for n=3
  - There is only 1 tree possible with n=3



all are same

Counting trees

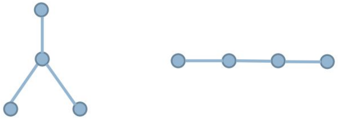
Eg: Count all labeled trees for n=4
no. of labeled trees possible for n=4 is $4^{4-2} = 16$



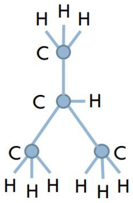
Counting trees

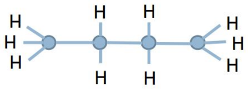
Q:How many structural isomers are there for Butane (C_4H_{10})?

- Possible unlabeled trees for n=4 are:



- Hence 2 isomers are possible for butane:


iso Butane


n Butane

Counting trees

Cayley's theorem

The number of labelled trees with 'n' vertices ($n \geq 2$) is n^{n-2}

Proof:

- Suppose we have a tree with n uniquely labeled vertices
- Number all vertices with numbers from 1 to n
- Find the pendant vertex with the least number. Let it be P1 . Let a1 be the vertex adjacent to p1
- Delete vertex p1 along with the incident edge from the tree
- Start a prufer sequence; add a1 to the prufer sequence (a1)

Counting trees

Proof cont..

- From the remaining tree with (n-1) vertices, find the pendant vertex with the least number. Let it be p2 . Let a2 be the vertex adjacent to p2
- Delete vertex p2 along with the incident edge from the tree
- Add a2 to the prufer sequence (a1 , a2 ,)
- Continue the process until only 2 vertices are left in the tree
- Now the prufer sequence (a1 , a2 , . . . , an-2)uniquely defines the tree
- Thus with n vertices, we can get n^{n-2} different labeled trees

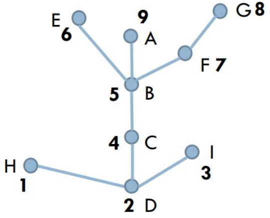
Counting trees

Example

Consider the uniquely labelled tree given

Number the vertices

- The pendant vertex with least number is H; hence p1=vertex H
  - a1=vertex D(2) (adjacent to p1)
  - Prufer seq = (2)
  - Delete H
- Next pendant vertex with least number p2= I
  - a2=vertex D(2) (adjacent to I)
  - Prufer seq = (2, 2)
  - Delete I
- Next pendant vertex is p3= D
  - a1=vertex C(4) (adjacent to D)
  - Prufer seq = (2, 2, 4)
  - Delete D



Counting trees

Example Cont..

- Next is p4= C
    - a4=vertex B(5) (adjacent to C)
    - Prufer seq = (2, 2, 4, 5)
    - Delete C
- Next is p5= E
    - a5=vertex B(5) (adjacent to E)
    - Prufer seq = (2, 2, 4, 5, 5)
    - Delete E
- Next is p6= G
    - a6=vertex F(7) (adjacent to G)
    - Prufer seq = (2, 2, 4, 5, 5, 7)
    - Delete G
- Next is p7= F
    - a7=vertex B(5) (adjacent to F)
    - Prufer seq = (2, 2, 4, 5, 5, 7, 5)
    - Delete

Counting trees

Example Cont..

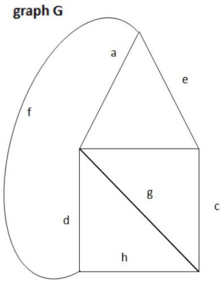
- Now the sequence is unique for the labelled tree
- Since the sequence contains 7 elements, and n=9, we can get $n^{n-2} = 9^7$ such unique sequences

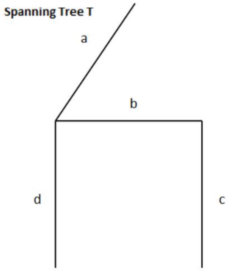
Spanning trees

Spanning trees

A tree T is said to be a spanning tree of a connected graph G if

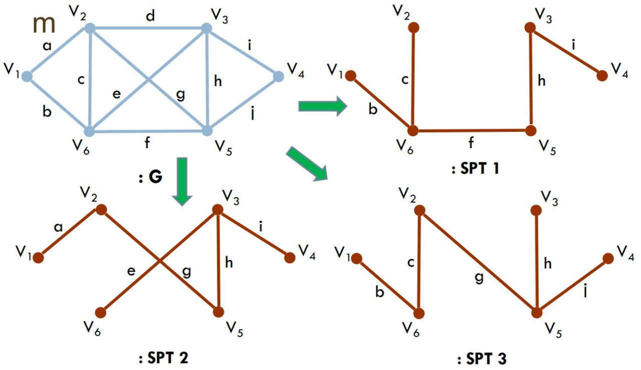
- T is a sub graph of G and
- contains all vertices of G.





Spanning trees

- **Note:**
  - Since the spanning tree depicts the skeleton of the graph G, T is also known as the **skeleton/scaffolding of G**
  - Since spanning trees are the largest trees among all possible trees of G, it is also known as **maximal tree subgraph or maximal tree of G**

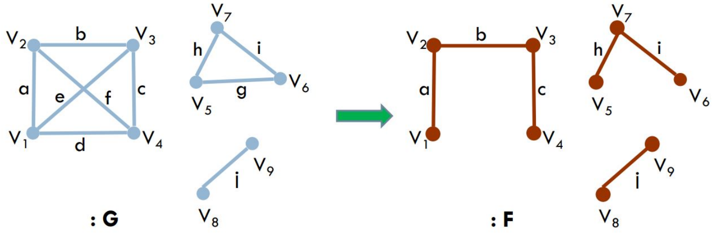


Spanning trees

Spanning forest

In a disconnected graph, a spanning tree can be found for each connected component, together which is known as a spanning forest

Hence a disconnect graph with k components will have a spanning forest with k spanning trees



Spanning trees

Theorem

Every connected graph has atleast one spanning tree.

- For a given connected graph G,
  1. If G has no circuits, G itself is the SPT
  2. If G has a circuit, remove one edge from the circuit such that G still remains connected
  3. Repeat step 2 until G is circuit-less
  4. The remaining graph is a SPT

Spanning trees

Branch

- An edge in a spanning tree T is called Branch of T

Chord

- An edge in G which is not in a spanning tree T is called Chord of T

Chord set (Co tree)

- If T is a spanning Tree, then complement of T is $\bar{T}$ or T'. T' is called chord set or co-tree.

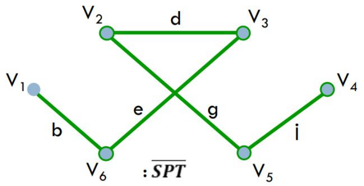
Note:

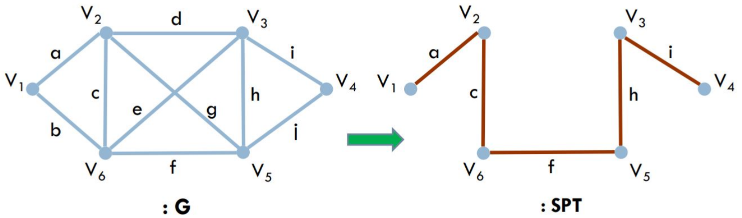
- In a connected graph of n vertices and e edges, w.r.t any of its spanning tree.
  - No of branches = n-1
  - No of chords = e-(n-1) = e-n+1.
  - (i.e) To eliminate all circuits,
    - the number of edges to be deleted is = e-n+1.

Spanning trees

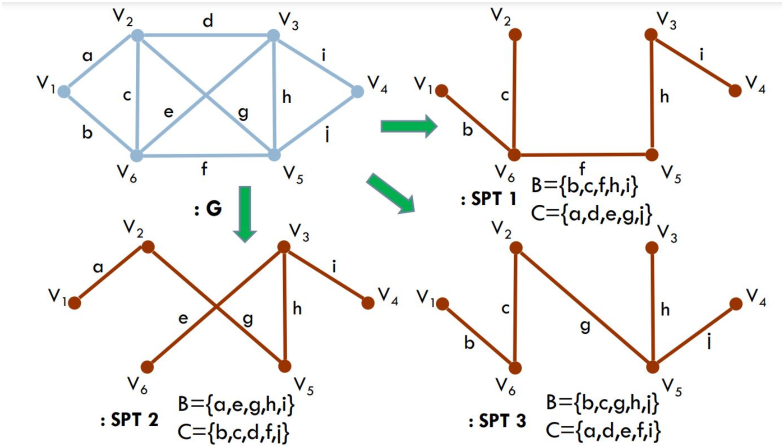
For the SPT chosen,

- Branches={a,c,f,h,i}
- Chords={b,d,e,g,j}





Spanning trees



Spanning trees

Theorem

Any SPT of a connected graph with n vertices and e edges has (n-1) tree branches and (e-n+1) chords

Proof:

- The connected graph has n vertices and e edges
- Any SPT of the graph will contain n vertices and (n-1) edges
- The remaining edges of the graph are chords, i.e. e-(n-1)= e-n+1

Spanning trees

Rank:

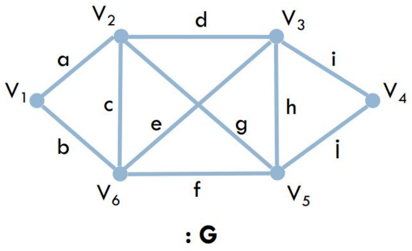
- Rank of G = no of branches in any spanning tree of G.
- Let n be the no of vertices , e be the no of edges k be the no of components then
    - Rank, r= n-k
- If the graph is connected (k=1)
    - Rank, r= n-1

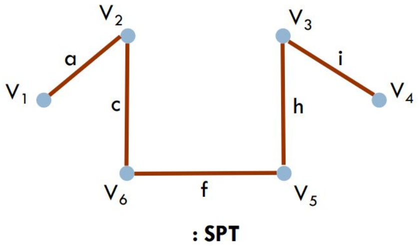
Nullity:

- Nullity of G = no of chords in G.
- Nullity $\mu = e - n + k$
- If the graph is connected (k=1)
    - Nullity $\mu = e - n + 1$

Rank + Nullity = no of edges in G

Spanning trees





B={a,c,f,h,i}
C={b,d,e,g,j}

- Rank r = n-1 = 6-1 = 5
- Nullity, μ = e-n+1 = 10-6+1 = 5

Spanning trees

Applications

Problem

An electric network contains e elements and n nodes. What is the minimum no. of elements that must be removed so as to make the network circuit-less?

Graph theoretic problem:

- Represent the network as a connected graph.
  - Let the edges represent the elements and vertices the nodes.
  - How many edges need to be removed so as make the graph circuit-less?

Solution

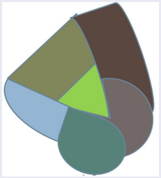
- No. of branches in the SPT = n-1
- No. of chords = e-n+1
- Hence removal of e-n+1 elements can make the network circuit-less

Spanning trees

Applications

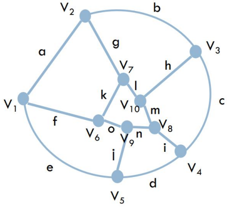
Problem

Consider a farm consisting of 6 walled plots of land and these plots are filled with water, then how many walls need to be broken so as to drain out the water?



Graph theoretic problem:

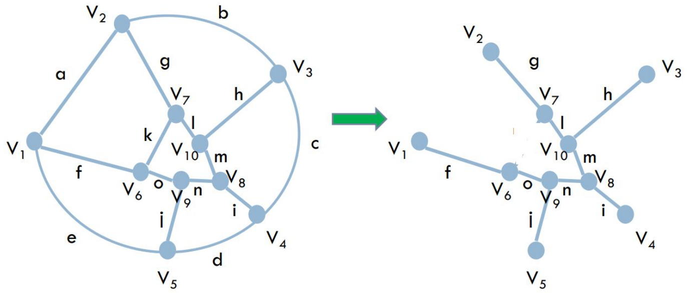
- Represent the farm as connected graph with the walls as edges and corners as vertices.
  - How many edges need to be removed so as make the graph circuit-less?



Spanning trees

Solution

- Since n=10 & e=15,
- No. of branches= n-1=9
- No. of chords = e-n+1 =6
- Hence, removing 6 chords can make the graph a tree; then water can flow out easily



Spanning trees

Properties of spanning trees

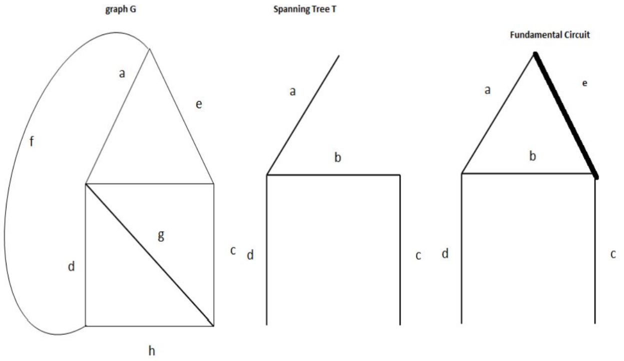
- A connected graph G may have any number of spanning trees
- The no. of vertices in any spanning tree is n; these are the vertices of the graph G
- The no. of edges in any spanning tree is n-1; these are some of the edges of the graph G
- The edges of a SPT are known as its branches
- The edges of the graph not included in a SPT are known as chords of the graph wrt the SPT
- Every SPT will contain (n-1) branches; but the set of (n-1) branches will be different for different SPTs
- Every SPT will leave (e-n+1) chords in the graph; the set of chords will be different for different SPTs

Spanning trees

Fundamental Circuits

A circuit formed by adding a chord to a spanning tree, is called a fundamental circuit.

- A circuit is a fundamental circuit only w.r.t a given spanning tree.



Spanning trees

Theorem

A connected graph is a tree iff adding an edge between any two vertices in G creates exactly one circuit

Proof:

- **suppose that the connected graph G is a tree**
- Add an edge between any two vertices of the tree say $v_i \& v_j$
- Since $v_i \& v_j$ are vertices of the tree, an edge between them creates a circuit, as there was already a path between $v_i \& v_j$ in the tree
- Since there could be only one path between every pair of vertices in a tree, adding an edge can create only one circuit

Spanning trees

Proof Cont..:

- **Conversely:**
- **suppose that adding an edge between any two vertices of G creates exactly one circuit**
- That means there was only path between every pair of vertices in G
- Which in turn implies that G was a tree

Spanning trees

Finding all spanning trees of a graph

Definition: (Cyclic Interchange or Elementary tree transformation.)

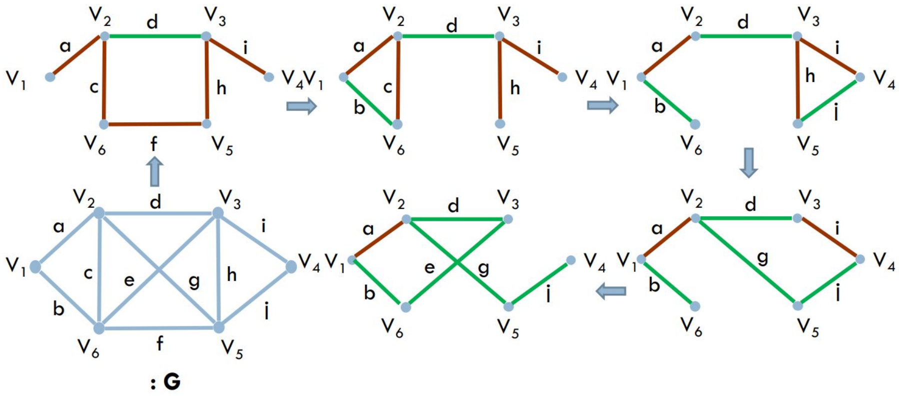
The generation of one spanning trees from another, through addition of a chord and deletion of an appropriate branch is called cyclic interchange.

How do get all spanning trees of a graph?

1. Start with a given SPT
2. Add a chord to the SPT so that a fundamental circuit is formed
3. Remove 1 branch from the circuit so formed. This will break the circuit and generate a new SPT
4. Repeat steps 2 & 3 until all SPTs are obtained

Spanning trees

Example

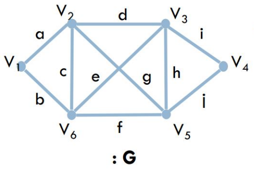


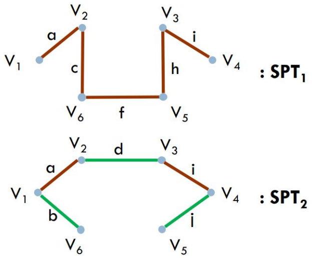
Spanning trees

Distance between two spanning trees

The distance between two spanning trees $T_i \& T_j$ of a graph G is defined as the number of edges of G present in one tree but not in the other.

- This is written as $d(T_i \& T_j)$
- $d(T_i \& T_j) = \frac{1}{2} N(T_i \oplus T_j)$
- where N(g) means Number of edges in g.





- Here, $N(SPT1 \oplus SPT2) = 6$
- d(SPT1,SPT2)=6/2=3

Spanning trees

Maximum distance b/w two SPTs

- Two SPTs are at maximum distance, when they are edge disjoint;
  - i.e they have no edge in common
- Since maximum no. of edges in a SPT is (n-1), the maximum distance possible b/w any 2 SPTs trees of a graph = n-1
- But if there aren't enough chords left, the maximum distance possible b/w any 2 SPTs get limited to the no. of chords available

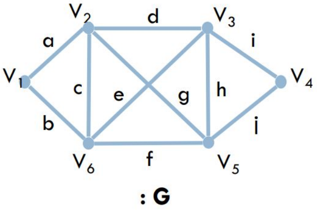
Spanning trees

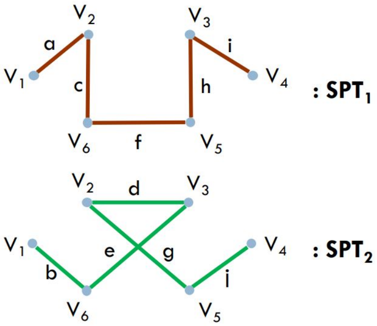
- For n vertices and e edges in G,
  - If e=2(n-1) (exactly n-1 chords available)
    - $d(SPT_i, SPT_j) = n - 1 = r$
  - If $e > 2(n-1)$ (more chords left)
    - $d(SPT_i, SPT_j) = n - 1 = r$
  - If $e < 2(n-1)$ (not enough chords)
    - $d(SPT_i, SPT_j) = e - n + 1 = \mu$
- Hence we can conclude
  - if $e \geq 2(n-1) \implies d(SPT_i, SPT_j) = r$
  - if $e < 2(n-1) \implies d(SPT_i, SPT_j) = \mu$
- More precisely,
  - $d(SPT_i, SPT_j) = min(r, \mu)$

Spanning trees

Example : when r = μ

- Here n=6 & e=10; r $= 5$ & $\mu = 5$
- Hence $d(SPT_1, SPT_2) = 5$

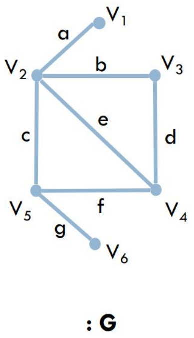


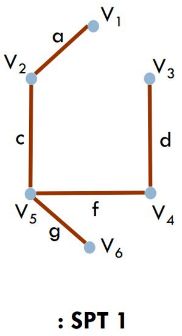


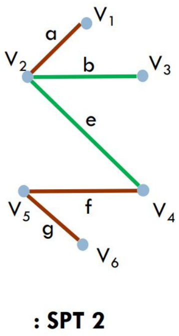
Spanning trees

Example : when r $> \mu$

- Here n=6 & e=7; r $= 5$ & $\mu = 2$
- Hence $d(SPT_1, SPT_2) = 2$



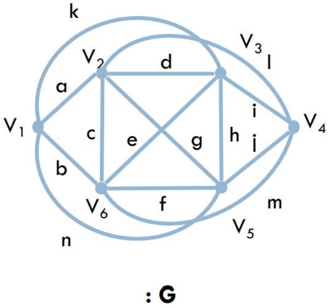


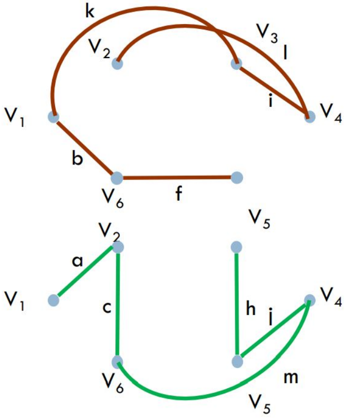


Spanning trees

Example : when r $< \mu$

- Here n=6 & e=15; r $= 5$ & $\mu = 9$
- Hence $d(SPT_1, SPT_2) = 5$





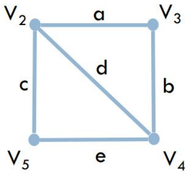
Spanning trees

- Hence maximum distance b/w any two SPTs of a graph is either
  - n-1 (rank,r) or
  - $e - n + 1(nullity, \mu)$
    - Whichever is smaller
- max $d(SPT_i, SPT_j) = min(r, \mu)$

Spanning trees

Problem

- Find the max distance b/w the SPTs of the graph G

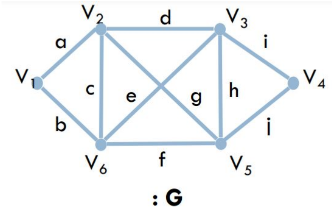


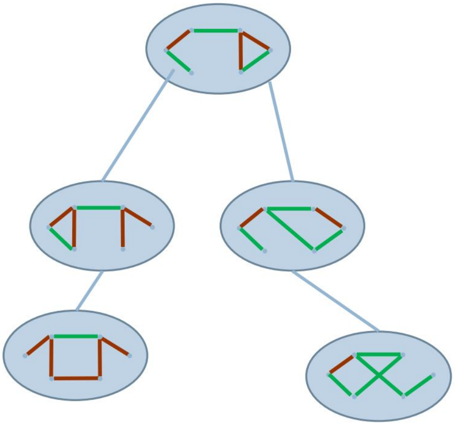
: G

Spanning trees

Tree graph

- All SPTs of a graph are represented as vertices of a tree
- The cyclic interchange from one SPT to another is represented as an edge between them





Spanning trees

Central spanning tree

- The SPT of a graph which has minimum distance with all other spanning trees of the same graph
- Same concept as center of a tree

Spanning trees

Theorem

The distance between the spanning trees of a graph is a metric

Proof:

- b b/w any two spanning trees is always positive or zero
  - $d(SPT_i, SPT_j) \geq 0$
  - $d(SPT_i, SPT_j) = 0 \; if \; i = j$
- The number of branches by which SPT_i differs from SPT_j is the same as that SPT_j differs from SPT_i
  - $d(SPT_i, SPT_j) = d(SPT_j, SPT_i)$
- $d(SPT_i, SPT_j) \leq d(SPT_i, SPT_k) + d(SPT_k, SPT_j)$

Spanning trees

Theorem

Starting from any one SPT, we can obtain every other SPT of G by successive cyclic interchanges

Spanning trees

Shortest Spanning trees

- In the context of weighted graphs, where a numerical value (weight) is associated with each edge of the graph
- The shortest SPT of the graph is the one with minimum sum of weights
- Similar to lightest Ham circuit (travelling salesman problem)
- Also known as
    - Shortest distance SPT
    - Minimal SPT

Spanning trees

Application

- **Real life problem:** Suppose if we need to construct roads to connect ‘n’ cities. Which are the cities that need to directly connected by roads so that construction cost can be minimized?
- **Graph theoretic problem:** Draw a complete graph with ‘n’ vertices. On each edge note down the construction cost. Find the minimum SPT of the graph. The branches of which represents the roads that are to be constructed

Spanning trees

Finding the min SPT

- Many algorithms are available to find the minimum SPT of weighted graphs
    - Kruskal's algorithm
    - Prim's algorithm

Kruskal's algorithm

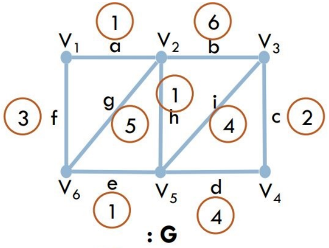
Kruskal's algorithm to find the minimum cost spanning tree uses the greedy approach.

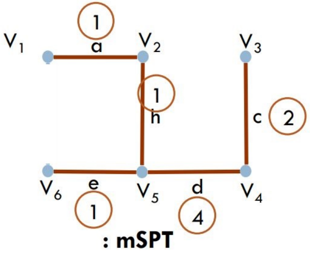
1. Remove all self loop and parallel edges
2. List all edges of the graph in the increasing order of their weights
3. Choose the edge with the smallest weight to be the first branch of the SPT
4. Choose the next smallest-weight edge such that it doesn't make a circuit with the preciously selected edges
5. Continue the process until (n-1) edges have been chosen

Kruskal's algorithm

Edge listing:

wt	edge	
1	a	✓
1	h	✓
1	e	✓
2	c	✓
3	f	×
4	d	✓
4	i	
5	g	
6	b	





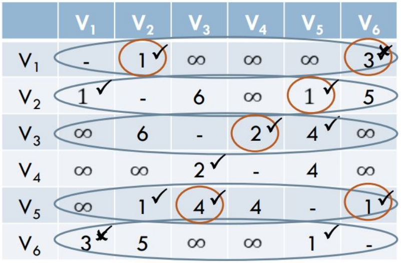
Weight of the mSPT=9

Prim's algorithm

Prim's Algorithm is a greedy algorithm that is used to find the minimum spanning tree from a graph

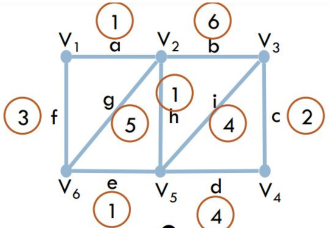
1. Remove all self loop and parallel edges
2. Draw 'n' isolated vertices and label them as $v_1, v_2, v_3, \dots v_n$
3. Tabulate the weights of the edges in an nxn matrix
4. Set the weights of non existent edges as α
5. Start from vertex v_1 and find its nearest neighbor (edge with minimum distance). Nearest neighbor is found by choosing the one which has least value in row1, say v_i . Draw the edge b/w v_1 and v_i in the null graph. Consider the edge as a subgraph
6. Now find the nearest neighbor of the subgraph. It is found by choosing the one with least value in rows 1 & i, say v_j . Add this edge to the subgraph.
7. Repeat step 3 until (n-1) edges have been chosen

Prim's algorithm

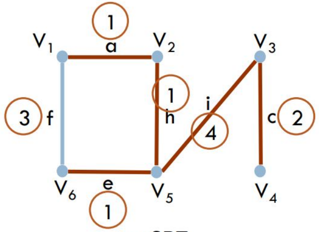


	V_1	V_2	V_3	V_4	V_5	V_6
V_1	-	1 ✓	∞	∞	∞	3 ✗
V_2	1 ✓	-	6	∞	1 ✓	5
V_3	∞	6	-	2 ✓	4 ✓	∞
V_4	∞	∞	2 ✓	-	4	∞
V_5	∞	1 ✓	4 ✓	4	-	1 ✓
V_6	3 ✗	5	∞	∞	1 ✓	-

Weight of the mSPT= 1+1+1+4+2=9



: G



: mSPT

Dijkstra's shortest path algorithm

Dijkstra's shortest path algorithm

- Dijkstra's shortest path algorithm is a solution to the single-source Shortest path problem in graph theory
- Works on both directed and undirected graphs. However, all edges must have nonnegative weights.

Approach: Greedy

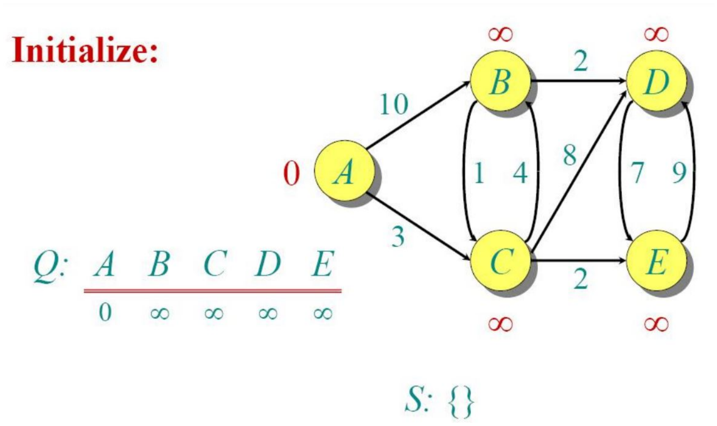
- **Input:**Weighted graph $G = \{E, V\}$ and source vertex v∈V, such that all edge weights are nonnegative
- **Output:**Lengths of shortest paths (or the shortest paths themselves) from a given source vertex v∈V to all other vertices

Dijkstra's shortest path algorithm

1. First of all, we will mark all vertex as unvisited vertex
2. Then, we will mark the source vertex as 0 and all other vertices as infinity
3. Consider source vertex as current vertex
4. Calculate the path length of all the neighboring vertex from the current vertex by adding the weight of the edge in the current vertex
5. Now, if the new path length is smaller than the previous path length then replace it otherwise ignore it
6. Mark the current vertex as visited after visiting the neighbor vertex of the current vertex
7. Select the vertex with the smallest path length as the new current vertex and go back to step 4.
8. Repeat this process until all the vertex are marked as visited.

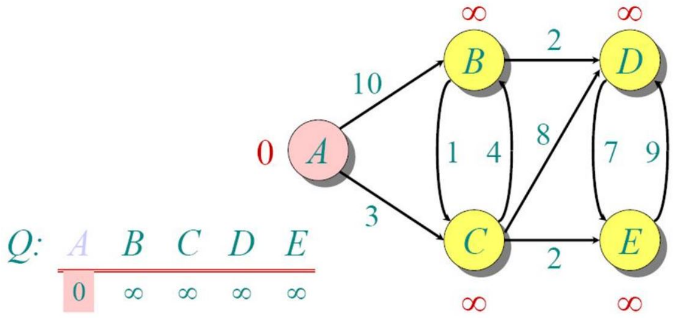
Once we go through the algorithm, we can backtrack the source vertex and find our shortest path.

Dijkstra's shortest path algorithm

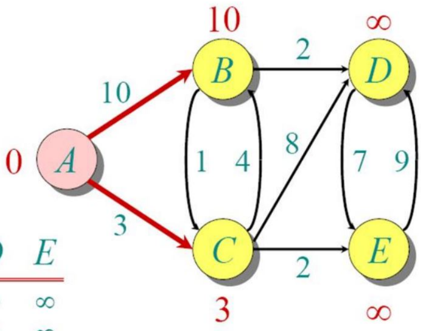


Q:	A	B	C	D	E
	0	∞	∞	∞	∞

Dijkstra's shortest path algorithm



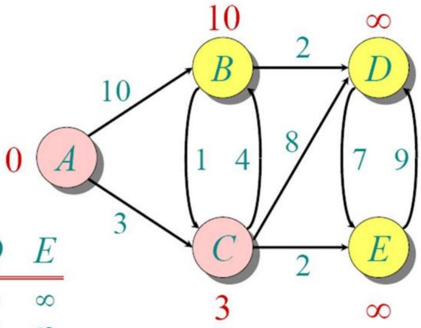
Dijkstra's shortest path algorithm



Q:	A	B	C	D	E
	0	∞	∞	∞	∞
		10	3	∞	∞

S: { A }

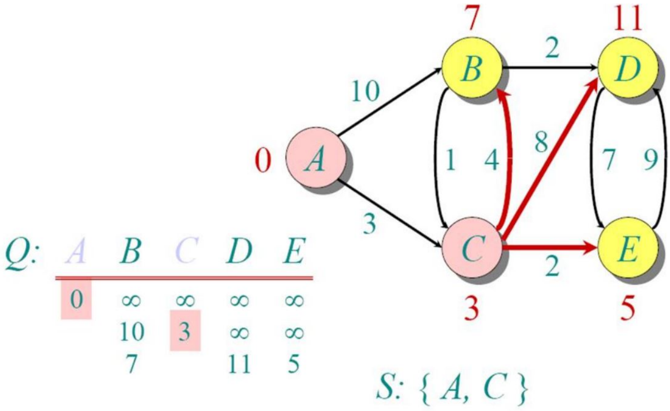
Dijkstra's shortest path algorithm



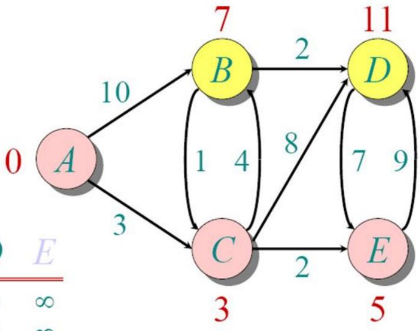
Q:	A	B	C	D	E
	0	∞	∞	∞	∞
		10	3	∞	∞

S: $\{ A, C \}$

Dijkstra's shortest path algorithm



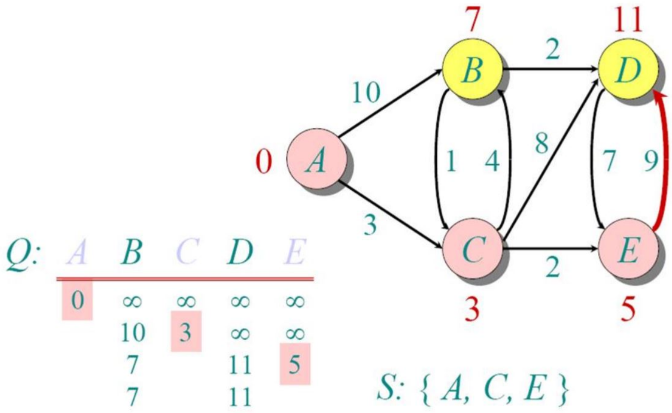
Dijkstra’s shortest path algorithm



Q:	A	B	C	D	E
	0	∞	∞	∞	∞
		10	3	∞	∞
		7		11	5

S: $\{ A, C, E \}$

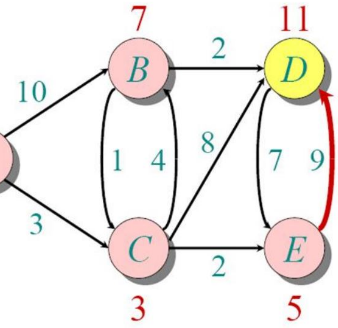
Dijkstra's shortest path algorithm



Q:	A	B	C	D	E
	0	∞	∞	∞	∞
		10	3	∞	∞
		7		11	5
		7		11	

S: { A, C, E }

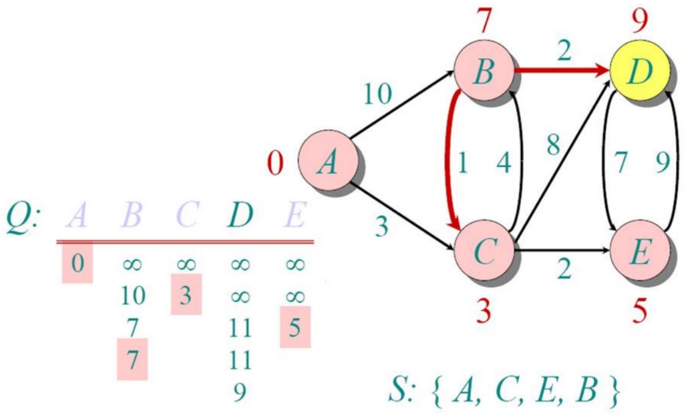
Dijkstra's shortest path algorithm



Q:	A	B	C	D	E
	0	∞	∞	∞	∞
		10	3	∞	∞
		7		11	5
		7		11	

S: { A, C, E, B }

Dijkstra's shortest path algorithm

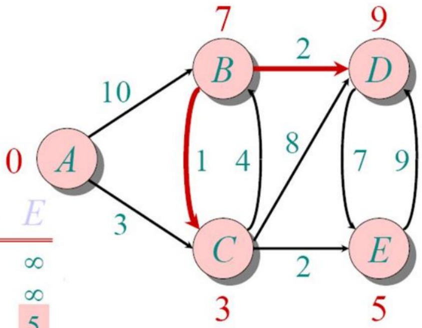


Q:

A	B	C	D	E
0	∞	∞	∞	∞
	10	3	∞	∞
	7		11	5
	7		11	
			9	

S: { A, C, E, B }

Dijkstra's shortest path algorithm



Q:	A	B	C	D	E
	0	∞	∞	∞	∞
		10	3	∞	∞
		7		11	5
		7		11	
				9	

S: $\{ A, C, E, B, D \}$

Floyd-Warshall shortest path algorithm

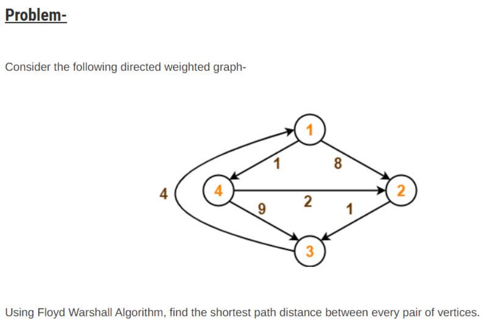
- Floyd-Warshall Algorithm is an algorithm for finding the shortest path between all the pairs of vertices in a weighted graph.
- This algorithm works for both the directed and undirected weighted graphs.

Floyd-Warshall shortest path algorithm

Steps

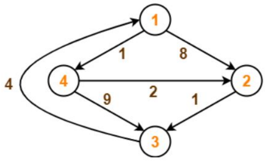
1. Remove all the self loops and parallel edges (keeping the lowest weight edge) from the graph.
2. Create a matrix A^0 of dimension $n * n$ where n is the number of vertices. The row and the column are indexed as i and j respectively. i and j are the vertices of the graph.
   - Each cell A[i][j] is filled with the distance from the ith vertex to the jth vertex. If there is no path from ith vertex to jth vertex, the cell is left as infinity.
3. Now, create a matrix A^1 using matrix A^0 . The elements in the first column and the first row are left as they are. The remaining cells are filled in the following way.
   - Let k be the intermediate vertex in the shortest path from source to destination. In this step, k is the first vertex. A[i][j] is filled with $(A[i][k] + A[k][j]) if (A[i][j] > A[i][k] + A[k][j])$
4. Similarly, A^2 is created using A^1. The elements in the second column and the second row are left as they are.
5. Similarly find the matrix A^n

Floyd-Warshall shortest path algorithm



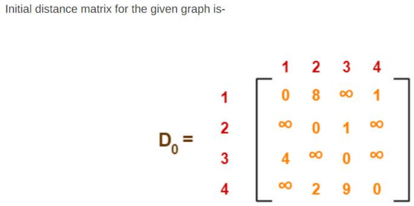
Problem-

Consider the following directed weighted graph-



Using Floyd Warshall Algorithm, find the shortest path distance between every pair of vertices.

Floyd-Warshall shortest path algorithm



Initial distance matrix for the given graph is-

$$D_0 = \begin{array}{c|cccc} & 1 & 2 & 3 & 4 \\ \hline 1 & 0 & 8 & \infty & 1 \\ 2 & \infty & 0 & 1 & \infty \\ 3 & 4 & \infty & 0 & \infty \\ 4 & \infty & 2 & 9 & 0 \end{array}$$

Floyd-Warshall shortest path algorithm

$$D_1 = \begin{array}{c|cccc} & 1 & 2 & 3 & 4 \\ \hline 1 & 0 & 8 & \infty & 1 \\ 2 & \infty & 0 & 1 & \infty \\ 3 & 4 & 12 & 0 & 5 \\ 4 & \infty & 2 & 9 & 0 \end{array}$$

$$D_3 = \begin{array}{c|cccc} & 1 & 2 & 3 & 4 \\ \hline 1 & 0 & 8 & 9 & 1 \\ 2 & 5 & 0 & 1 & 6 \\ 3 & 4 & 12 & 0 & 5 \\ 4 & 7 & 2 & 3 & 0 \end{array}$$

$$D_2 = \begin{array}{c|cccc} & 1 & 2 & 3 & 4 \\ \hline 1 & 0 & 8 & 9 & 1 \\ 2 & \infty & 0 & 1 & \infty \\ 3 & 4 & 12 & 0 & 5 \\ 4 & \infty & 2 & 3 & 0 \end{array}$$

$$D_4 = \begin{array}{c|cccc} & 1 & 2 & 3 & 4 \\ \hline 1 & 0 & 3 & 4 & 1 \\ 2 & 5 & 0 & 1 & 6 \\ 3 & 4 & 7 & 0 & 5 \\ 4 & 7 & 2 & 3 & 0 \end{array}$$

The last matrix D4 represents the shortest path distance between every pair of vertices.