SCSI 1013: Discrete Structure

CHAPTER 4

GRAPH THEORY

Part 2

2018/2019 – SEM. 1 : nzah@utm.my

Tree

Introduction

Definition 1. A tree is a connected undirected graph with no simple circuits.

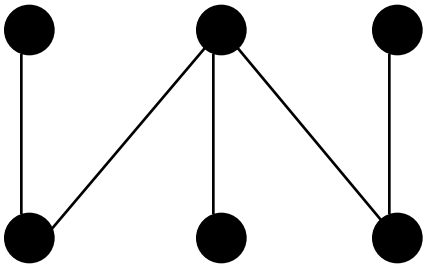
Theorem 1. An undirected graph is a tree if and only if there is a unique simple path between any two of its vertices.

Theorem 2 . A tree with m-vertices has m-1 edges

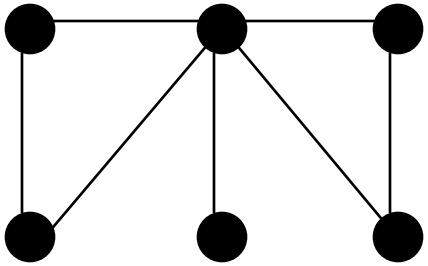
Example

Which of the graphs (a, b and c) are trees?

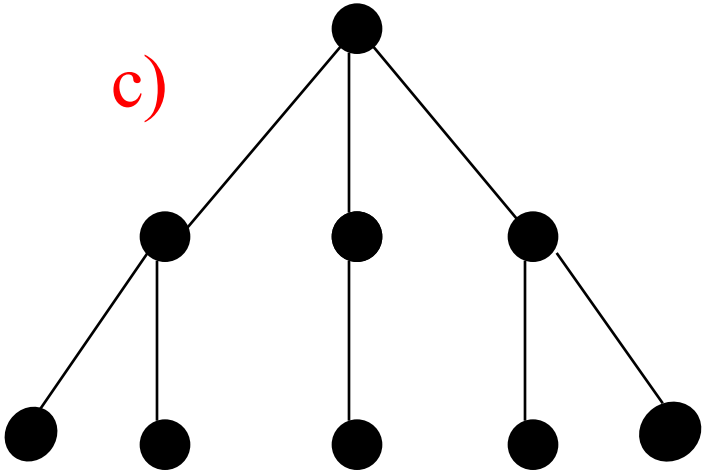
a)



b)

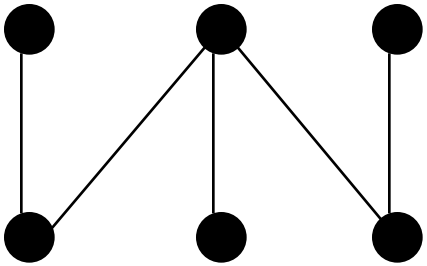


c)



Solution

a)

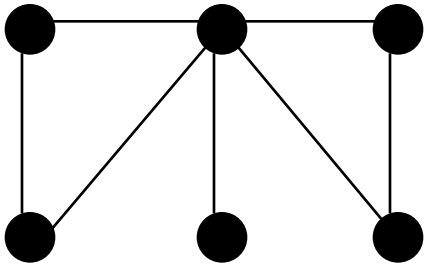


tree

vertices = 6

edges = 5

b)

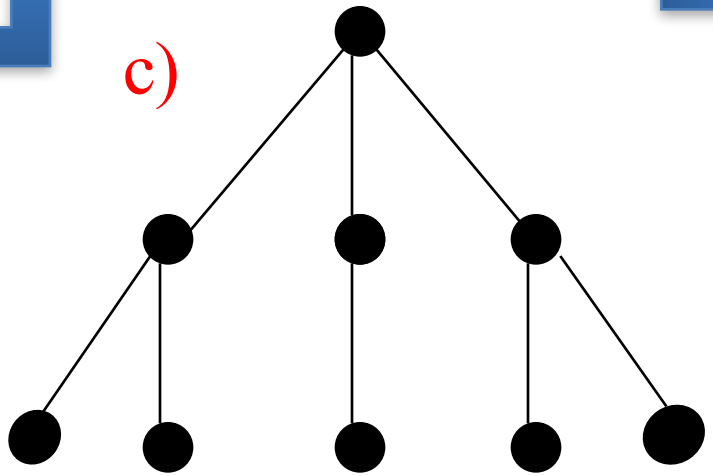


Not a tree

vertices = 6

edges = 6

c)



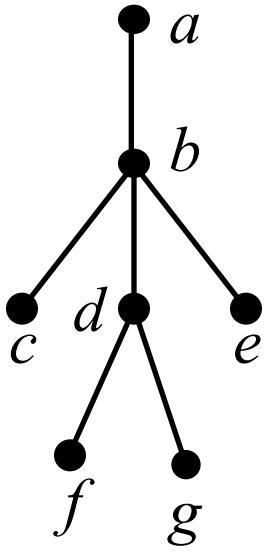
tree

vertices = 9

edges = 8

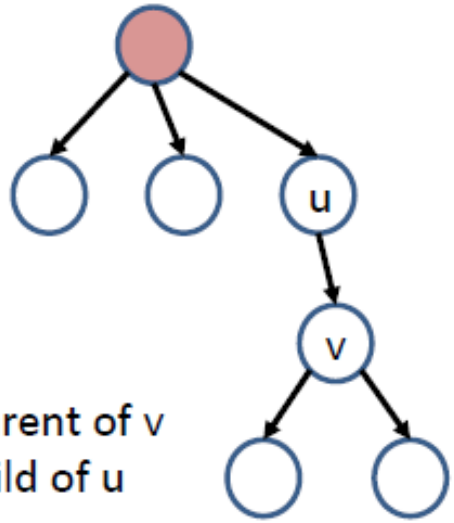
Rooted Tree

Definition 2. A **rooted tree** is a tree in which one vertex has been designed as the **root** and every edge is directed away from the root.

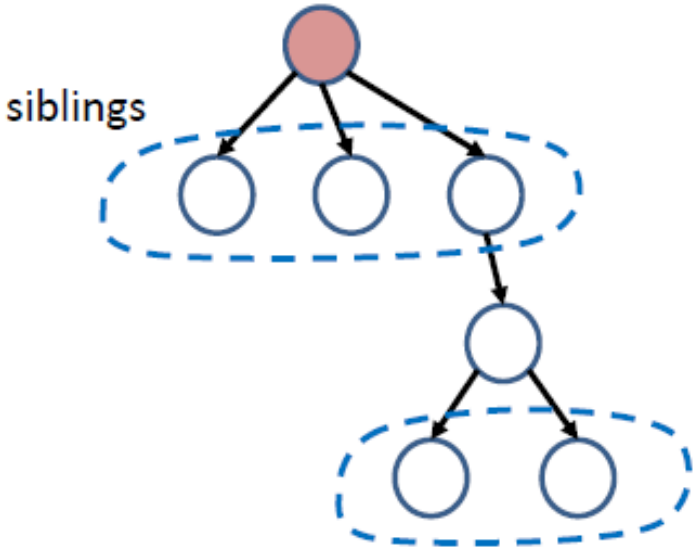


Rooted Tree - Terminologies

- Each edge is from a parent to a child
- Vertices with the same parent are siblings

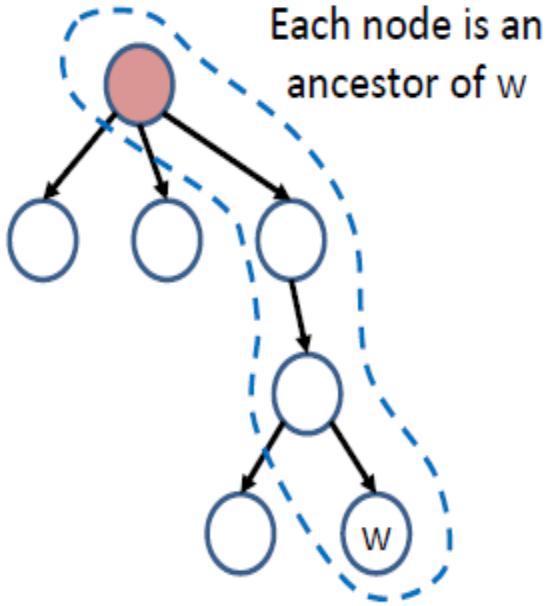


u is the parent of v
v is a child of u



..cont'd

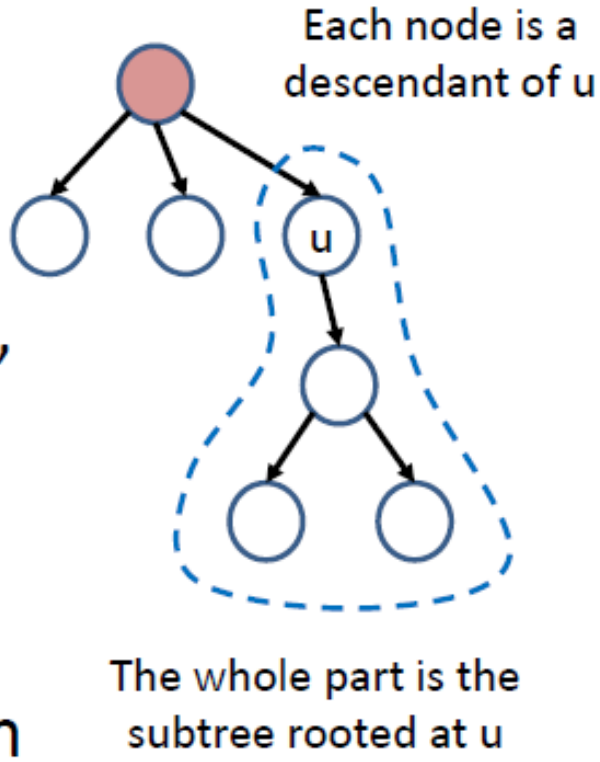
- The ancestors of a vertex w include all the nodes in the path from the root to w
- The proper ancestors of a vertex w are the ancestors of w, but excluding w



The whole part forms a path from root to w

..cont'd

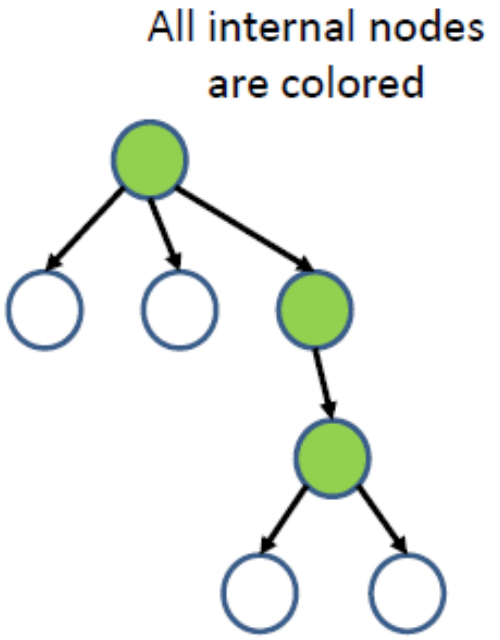
- The descendants of a vertex u include all the nodes that have u as its ancestor
- The proper descendants of a vertex u are the descendants of u, but excluding u
- The subtree rooted at u includes all the descendants of u, and all edges that connect between them



..cont'd

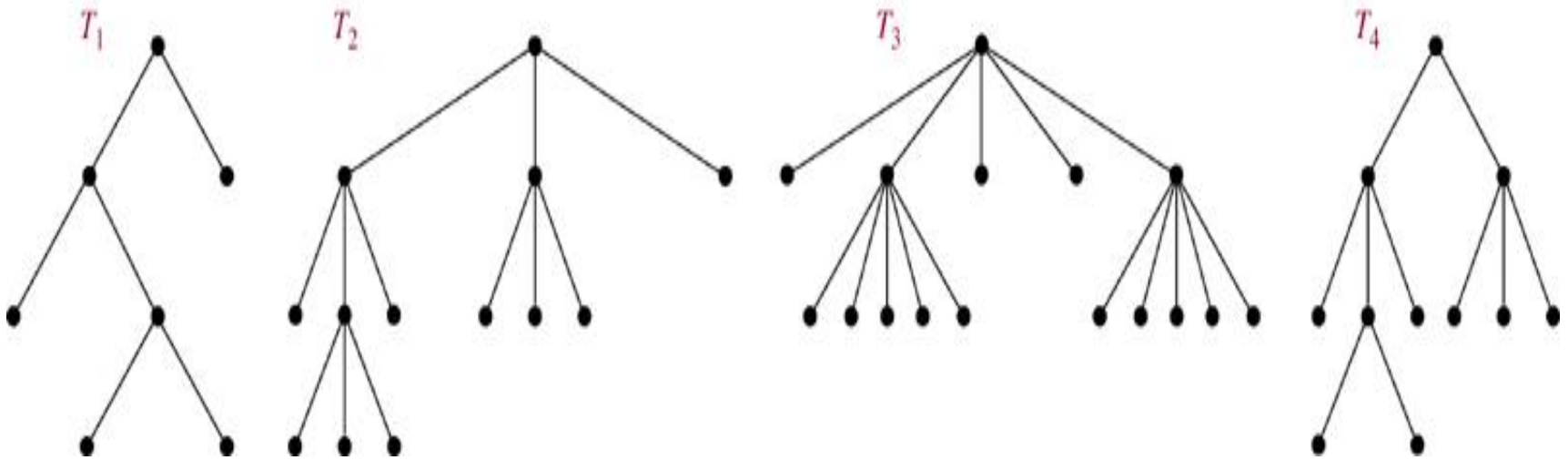
- Vertices with no children are called leaves ;

  Otherwise, they are called internal nodes
- If every internal node has no more than m children, the tree is called an m-ary tree
  - Further, if every internal node has exactly m children, the tree is a full m-ary tree



The tree is ternary (3-ary), but not full

Examples



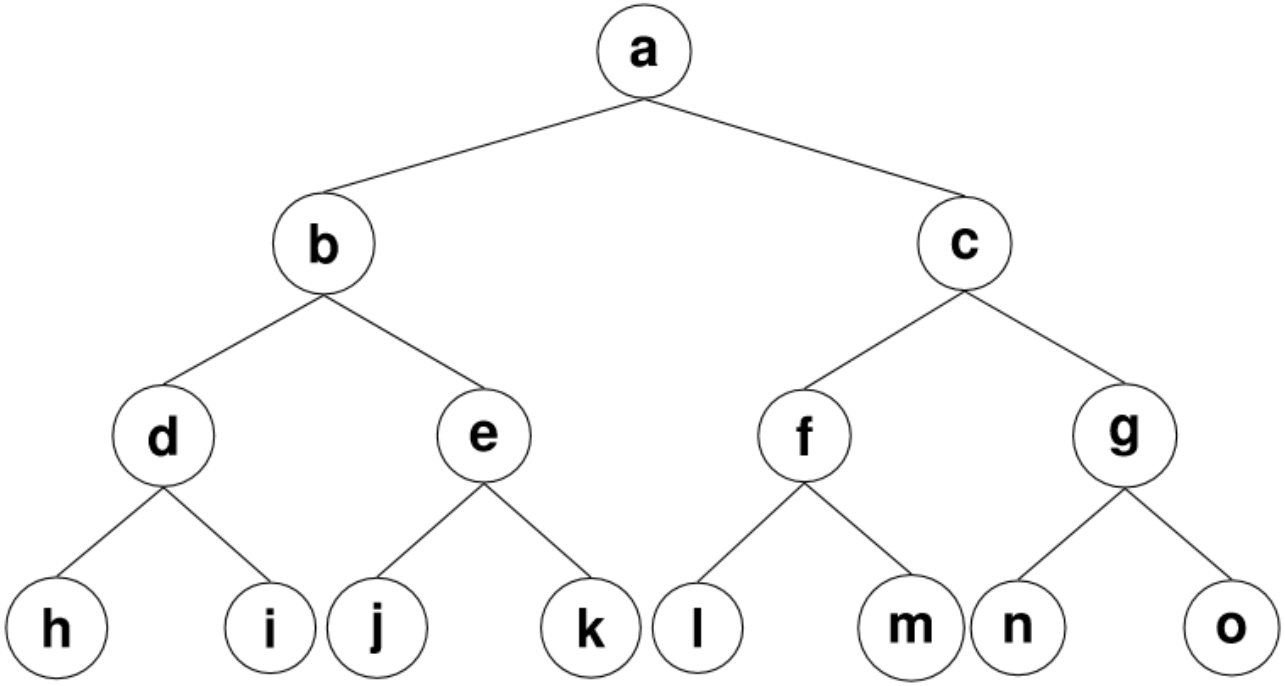
full binary tree

full 3-ary tree

full 5-ary tree

not full 3-ary tree

Exercise



Find:

- Ancestors of *g*
- Descendents of *g*
- Parent of *e*
- Children of *e*
- Sibling of *h*

Properties of Trees

- **Theorem 2** : A tree with n nodes has $n-1$ edges
- **Theorem 3** : A full m-ary tree with i internal vertices contains $n = mi + 1$ vertices.

Corollary: A full m-ary tree with n vertices contains $(n-1)/m$ internal vertices, and hence $n - (n-1)/m = ((m-1)\ n+1)/m$ leaves

Corollary is a result in which the (usually short) proof relies heavily on a given theorem.

Theorem 4 – A full ***m*-ary** tree with

1. n vertices has $i = (n\text{-}1)/m$ internal vertices and $l = [(m\text{-}1)n+1]/m$ leaves.
2. i internal vertices has $n = mi + 1$ vertices and $l = (m\text{-}1)i + 1$ leaves.
3. l leaves has $n = (ml\text{-}1)/(m\text{-}1)$ vertices and $i = (l\text{-}1)/(m\text{-}1)$ internal vertices.

Example

Peter starts out a chain mail. Each person receiving the mails is asked to send it to four other people. Some people do this, and some don't. Now, there are 100 people who received the letter but did not send it out. Assuming no one receives more than one mail. How many people have sent the letter?

Exercise

How many matches are played in a tennis tournament of 27 players.

Solution:

- A leaf for each player, l=27
- An internal node for each matches: m=2
- Number of matches: $\frac{l \quad 1}{m \quad 1} = \frac{27 \quad 1}{2 \quad 1} = 26$

Exercise

Suppose 1000 people enter a chess tournament. Use a rooted tree model of the tournament to determine how many games must be played to determine a champion, if a player is eliminated after one loss and games are played until only one entrant has not lost. (Assume there are no ties.)

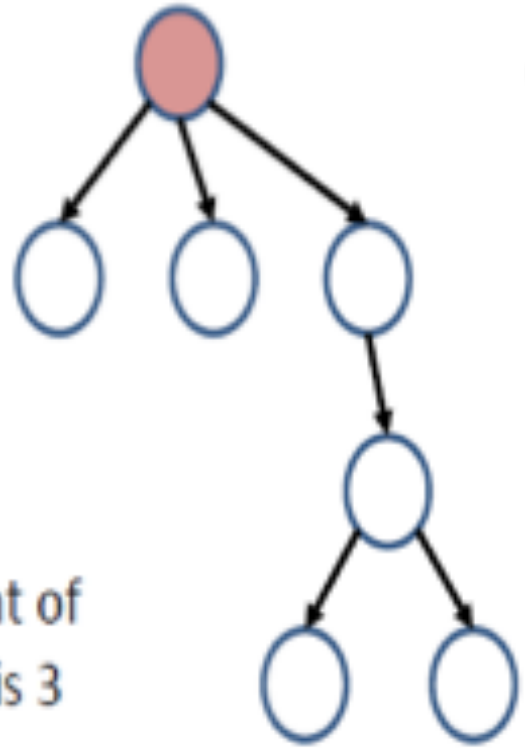
Properties of Rooted Trees

- The level of a vertex v in a rooted tree is the length of the unique path from the root to this vertex.

- The level of the root is defined to be zero.

- The height of a rooted tree is the maximum of the levels of vertices.

Example



Level 0

Level 1

Level 2

Level 3

This height of the tree is 3

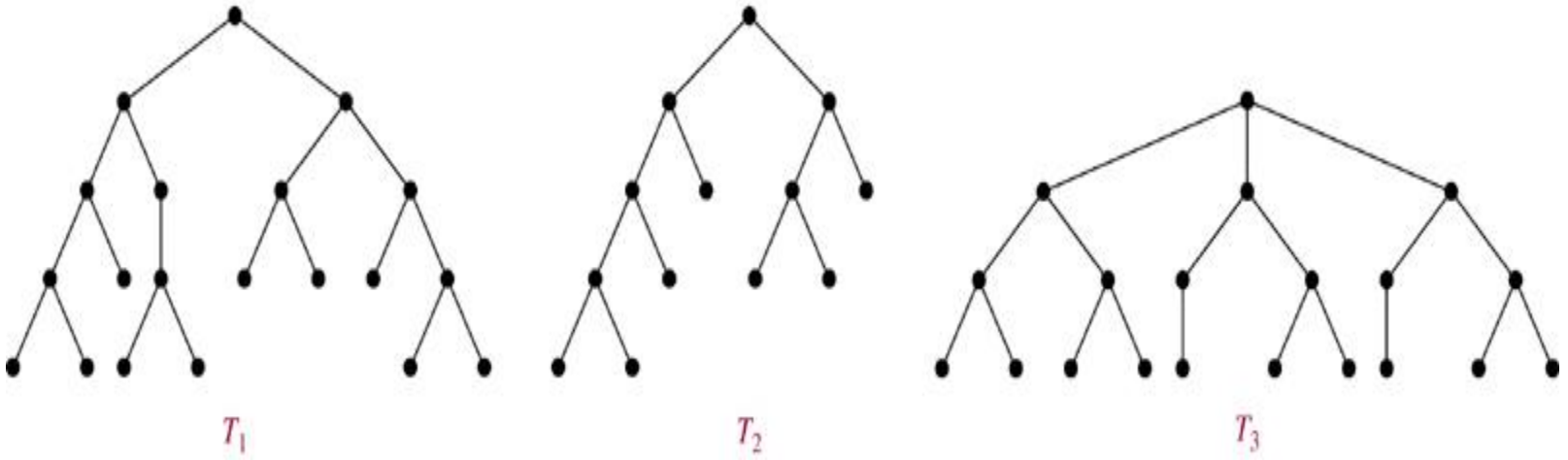
..cont'd

- **Definition:** A rooted m-ary tree of height h is balanced if all leaves are at levels h or $h-1$.

- **Theorem.** There are at most m^h leaves in an m-arry tree of height h.

Example

Which of the rooted trees shown below are balanced?



Solution: T_1, T_3

Tree Traversal

Universal Address Systems

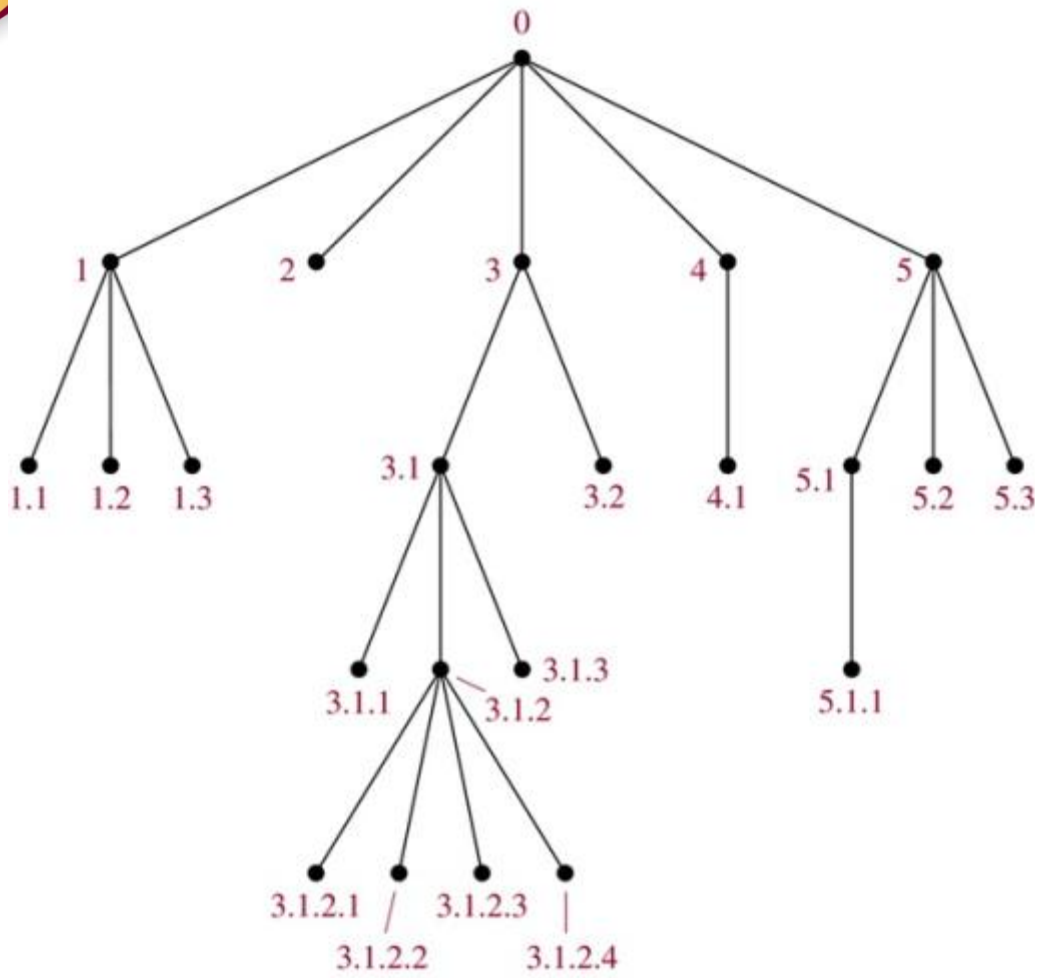
Label vertices:

1.root $\rightarrow$ 0, its k children $\rightarrow$ $1, 2, \ldots, k$ (from left to right)

2.For each vertex v at level n with label A,
its r children $\rightarrow$ $A.1, A.2, \ldots, A.r$ (from left to right).

We can totally order the vertices using the lexicographic ordering of their labels in the universal address system.

$x_1.x_2.....x_n < y_1.y_2.....y_m$
if there is an i, $0 \leq i \leq n$, with $x_1 = y_1$, $x_2 = y_2$, …, $x_{i-1} = y_{i-1}$, and $x_i < y_i$; or if $n < m$ and $x_i = y_i$ for $i = 1, 2, \ldots, n$.



The lexicographic ordering is:

$0 < 1 < 1.1 < 1.2 < 1.3 < 2 < 3 < 3.1 < 3.1.1 < 3.1.2 < 3.1.2.1 < 3.1.2.2 < 3.1.2.3 < 3.1.2.4 < 3.1.3 < 3.2 < 4 < 4.1 < 5 < 5.1 < 5.1.1 < 5.2 < 5.3$

..cont'd

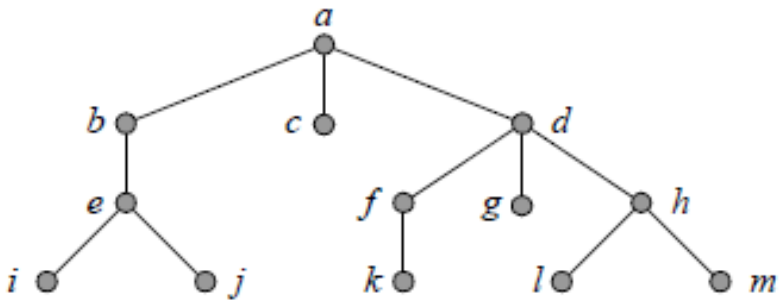
Procedures for systematically visiting every vertex of an ordered rooted tree are called traversal algorithm. There are three most commonly used such algorithms:

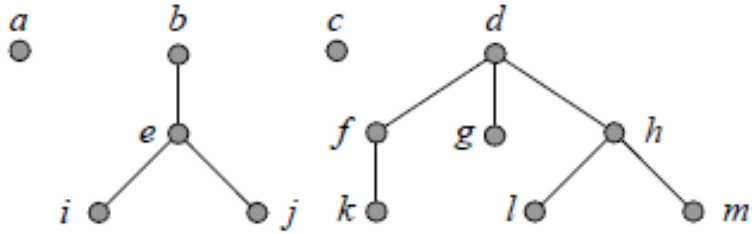
1) Preorder: root, left-subtree, right subtree
2) Inorder: left subtree, root, right sub-tree
3) Postorder: left subtree, right sub-tree, root

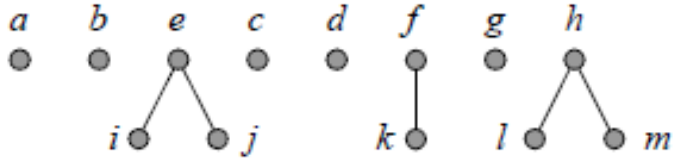
Preorder Traversal

Procedure *preorder*(*T*: ordered rooted tree)
r := root of *T*
list *r*
for each child *c* of *r* from left to right
begin
    T(*c*) := subtree with *c* as its root
    preorder(*T*(*c*))
end

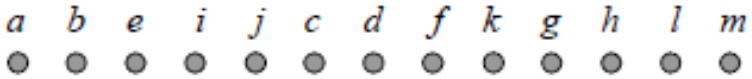
Example







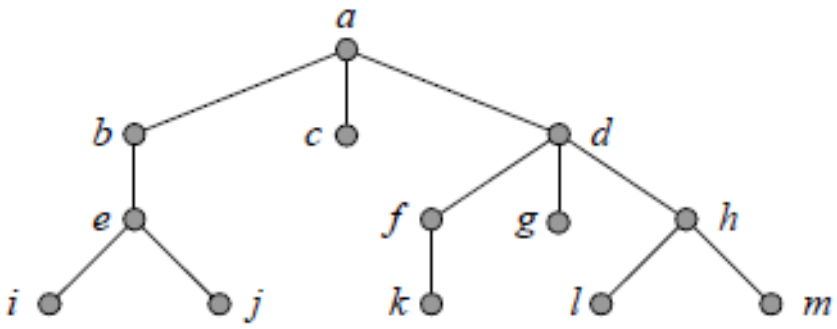
Preorder traversal:

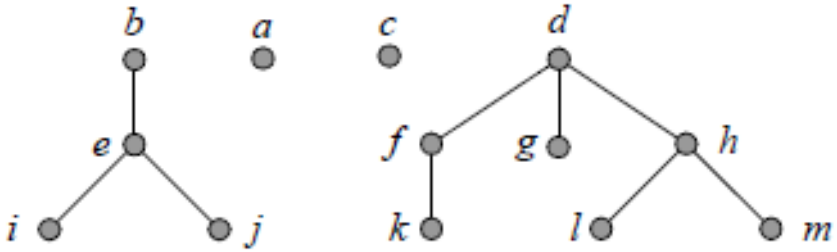


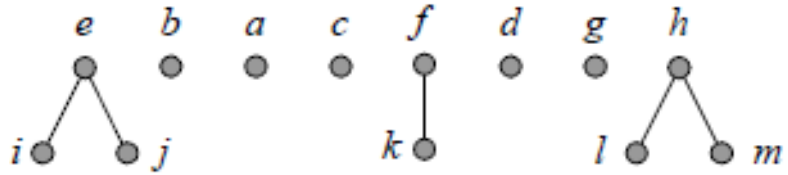
Inorder Traversal

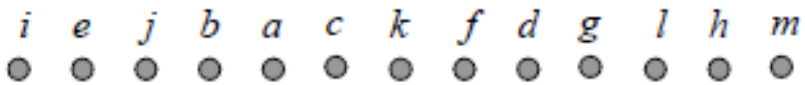
Procedure *inorder*(*T*: ordered rooted tree)
r := root of *T*
If *r* is a leaf **then** list *r*
else
begin
    l := first child of *r* from left to right
    T(*l*) := subtree with *l* as its root
    inorder(*T*(*l*))
    list *r*
    for each child *c* of *r* except for *l* from left to right
        T(*c*) := subtree with *c* as its root
        inorder(*T*(*c*))
end

Example







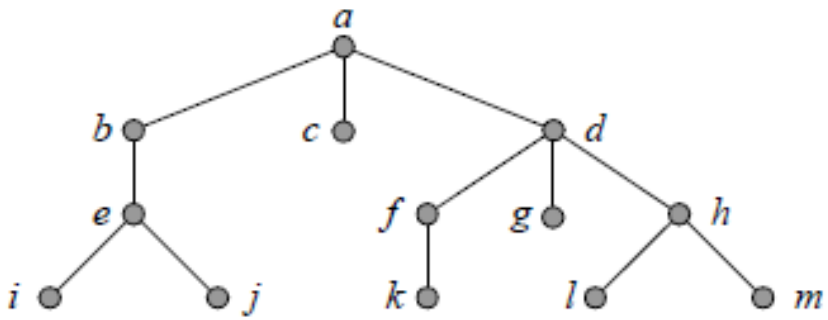


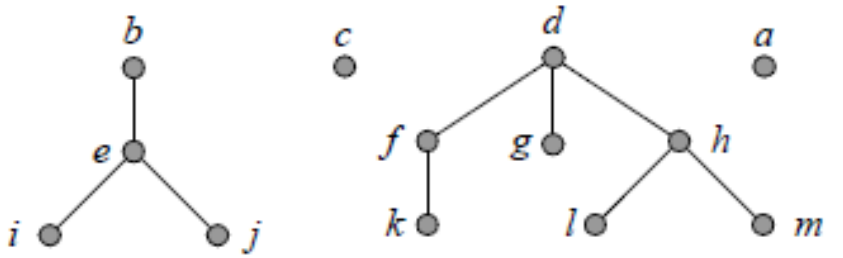
Inorder traversal

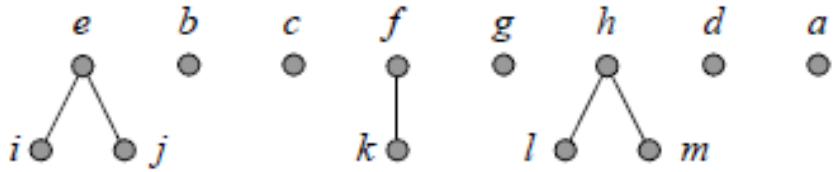
Postorder Traversal

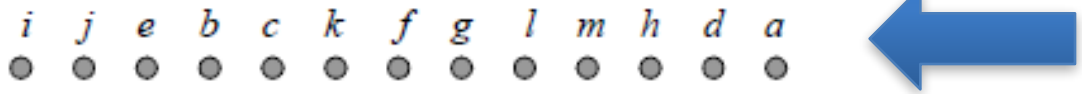
Procedure *postorder*(*T*: ordered rooted tree)
r := root of *T*
for each child *c* of *r* from left to right
begin
    T(*c*) := subtree with *c* as its root
    postorder(*T*(*c*))
end
list *r*

Example



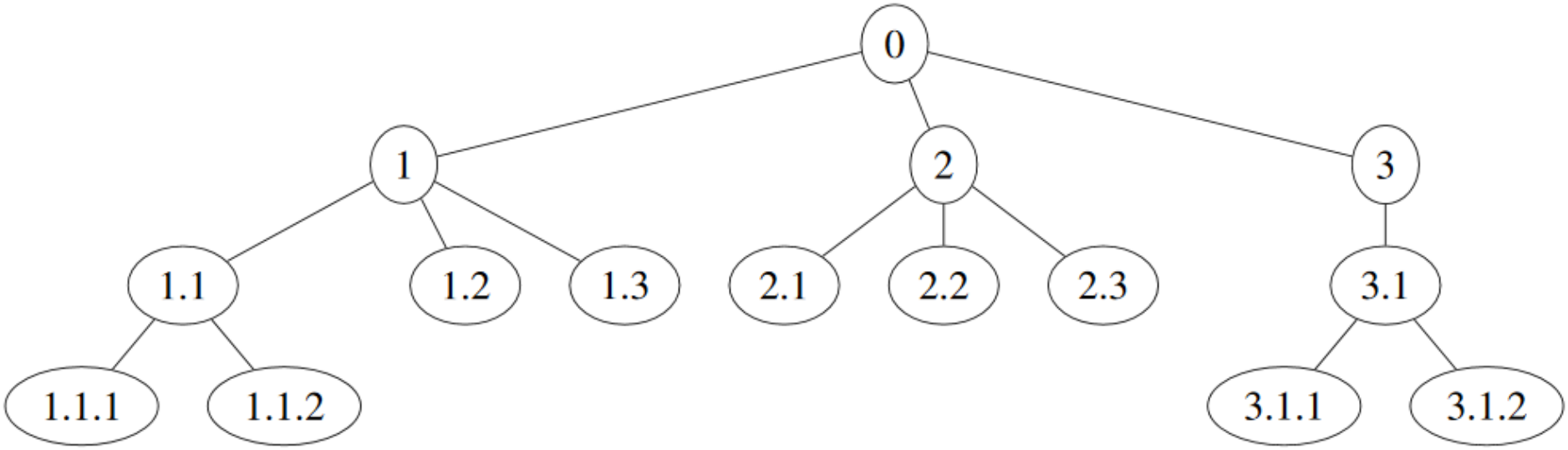






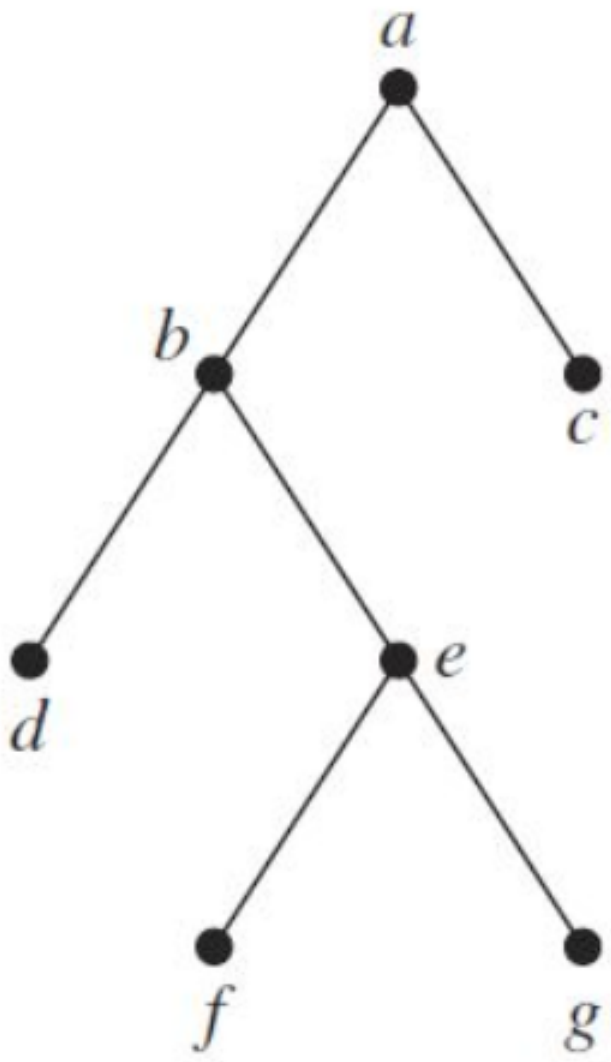
Postorder traversal

Exercise



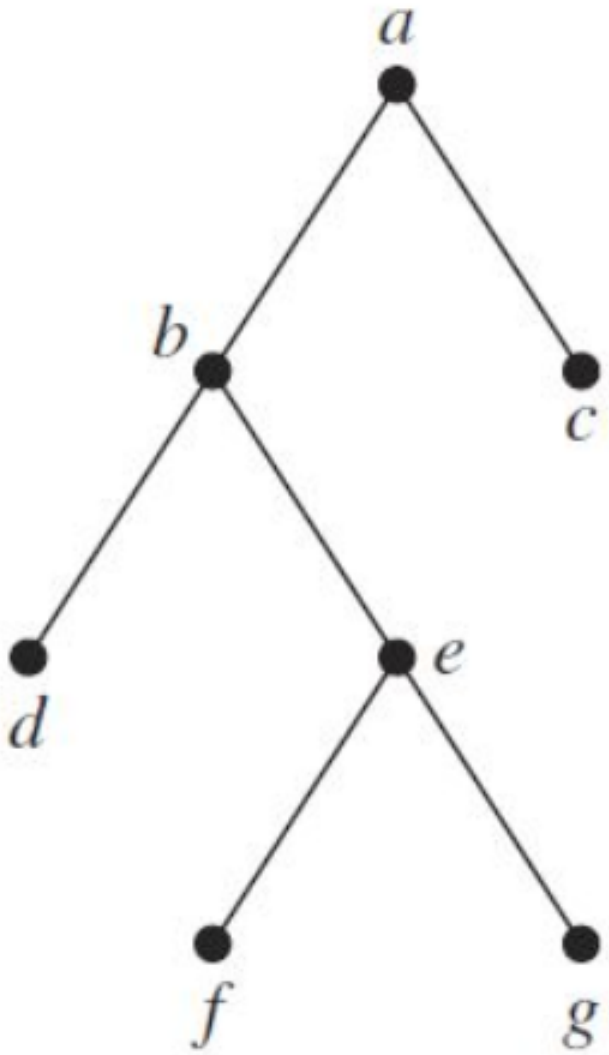
Find the lexicographic ordering of the above tree.

Exercise



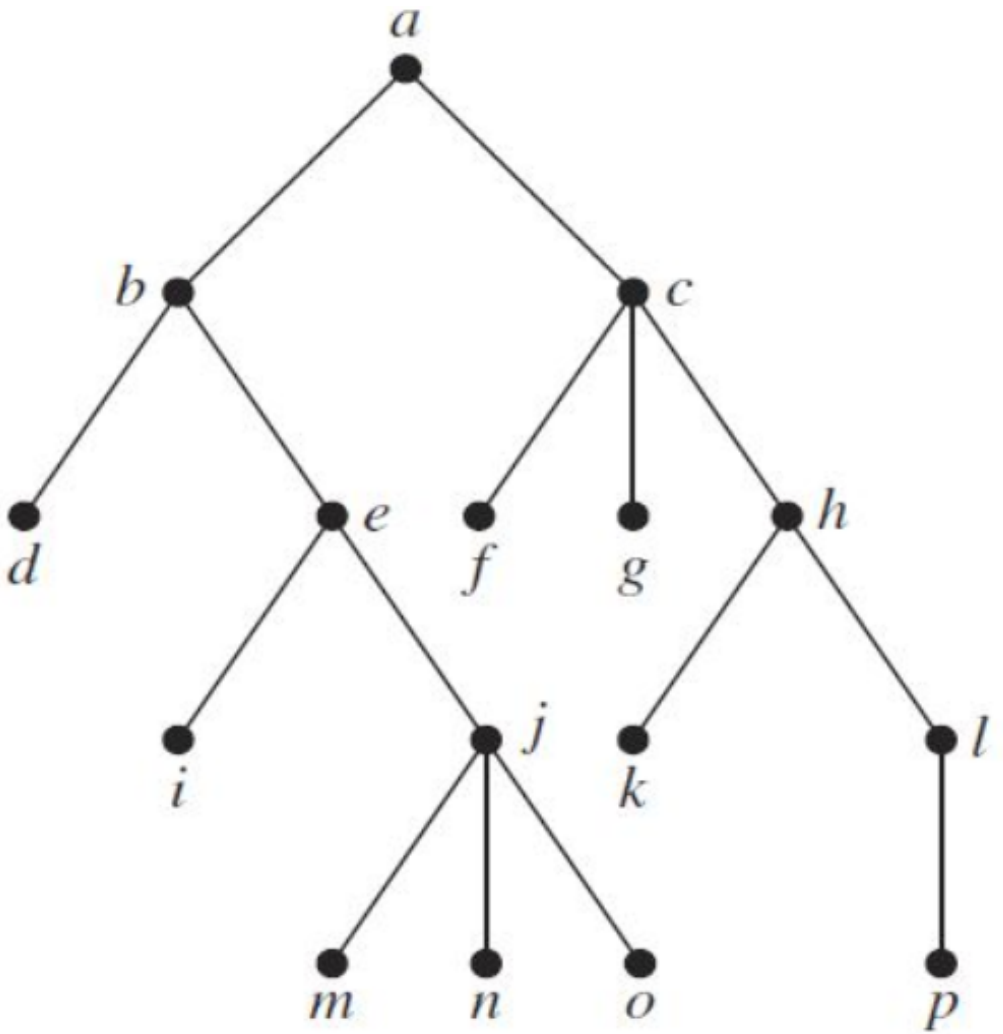
Determine the order in which a preorder traversal visits the vertices of the given ordered rooted tree.

Exercise



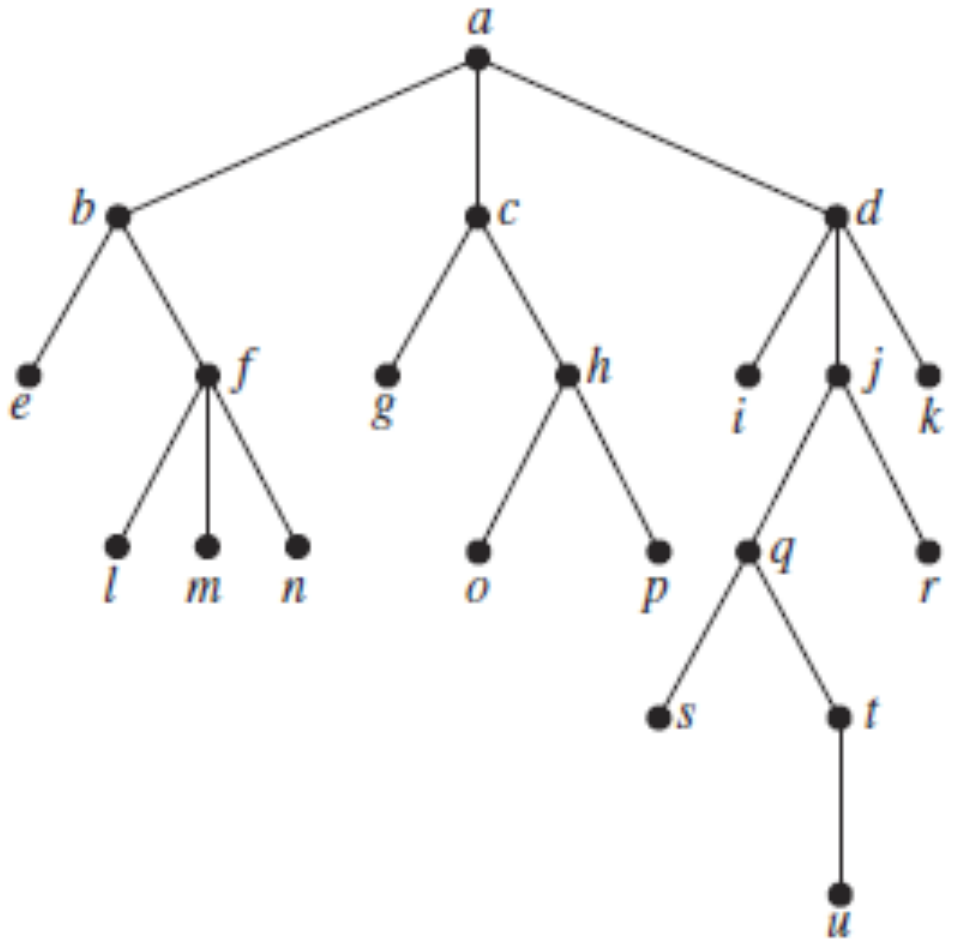
Determine the order in which a inorder traversal visits the vertices of the given ordered rooted tree.

Exercise



Determine the order of preorder, inorder and postorder of the given rooted tree.

Exercise

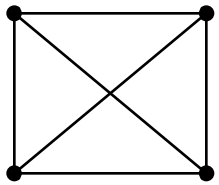


Determine the order of preorder, inorder and postorder of the given rooted tree.

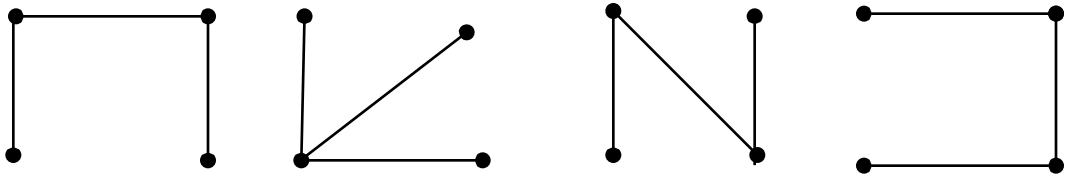
Spanning Tree

Spanning Trees

- A spanning tree is a simple graph that is a subgraph of ***G*** and contains every vertex of ***G*** and is a tree.



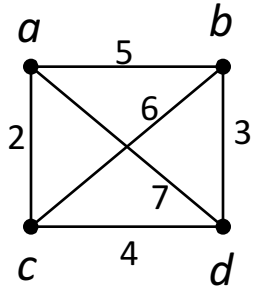
A connected undirected graph

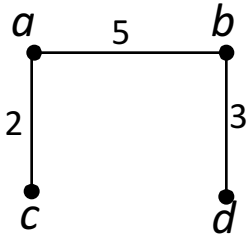


Four spanning trees of the graph

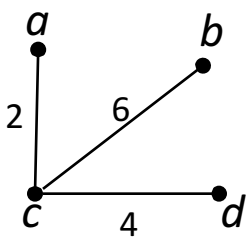
Minimum Spanning Tree (MST)

- A Minimum Spanning Tree is a spanning tree on a weighted graph that has minimum total weight.
- Example:

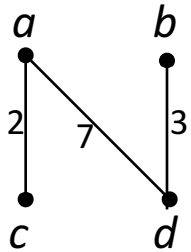




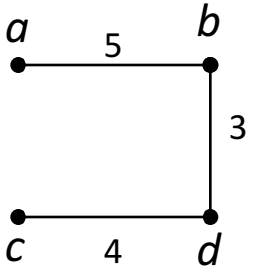
$T_1 = 10$



$T_2 = 12$

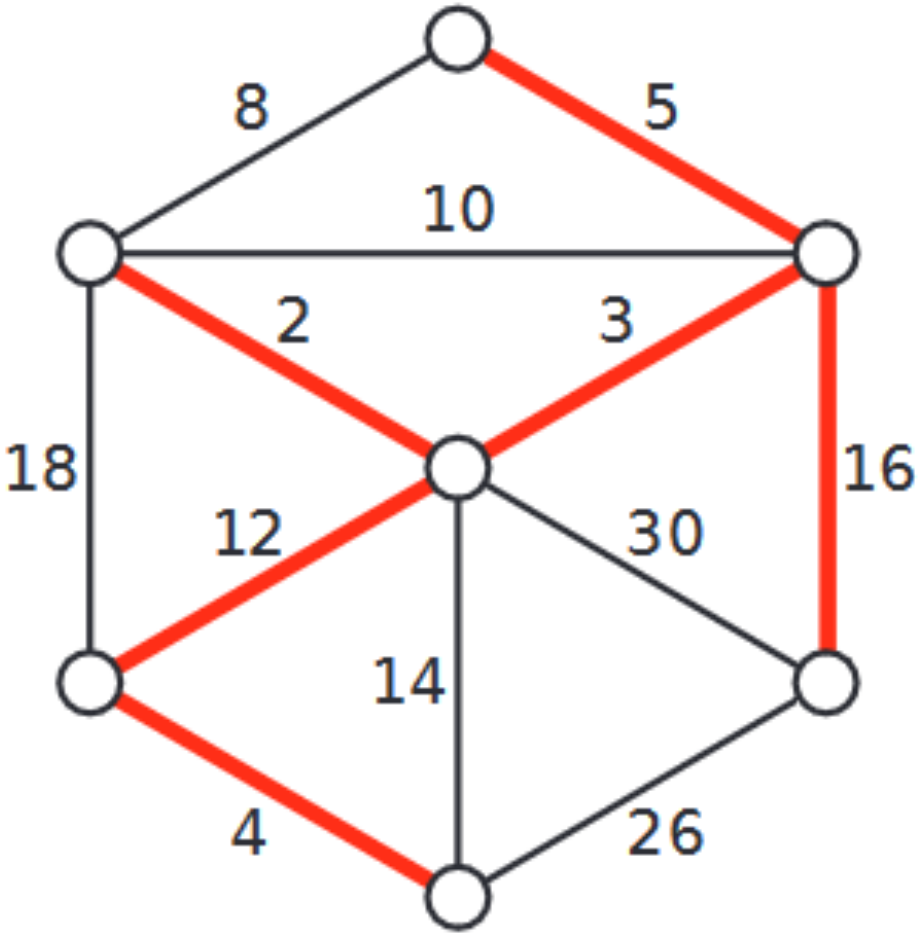


$T_3 = 12$



$T_4 = 12$

Minimum Spanning Tree (MST)



A weighted graph and its minimum spanning tree.

Muddy City Problem

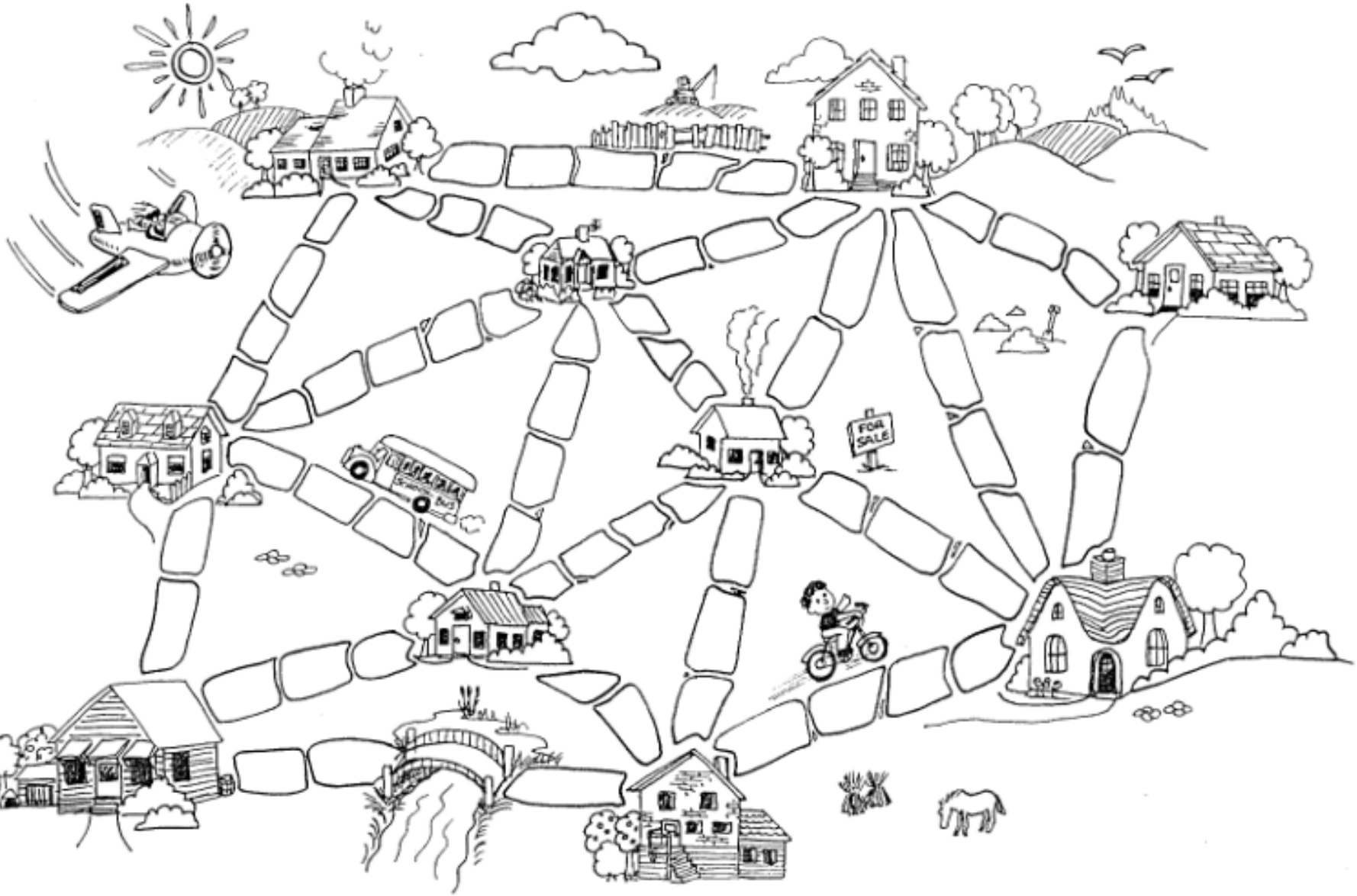
Once upon a time there was a city that had no roads. Getting around the city was particularly difficult after rainstorms because the ground became very muddy. Cars got stuck in the mud and people got their boots dirty. The mayor of the city decided that some of the streets must be paved, but didn't want to spend more money than necessary because the city also wanted to build a swimming pool.

Muddy City Problem

The mayor therefore specified two conditions:

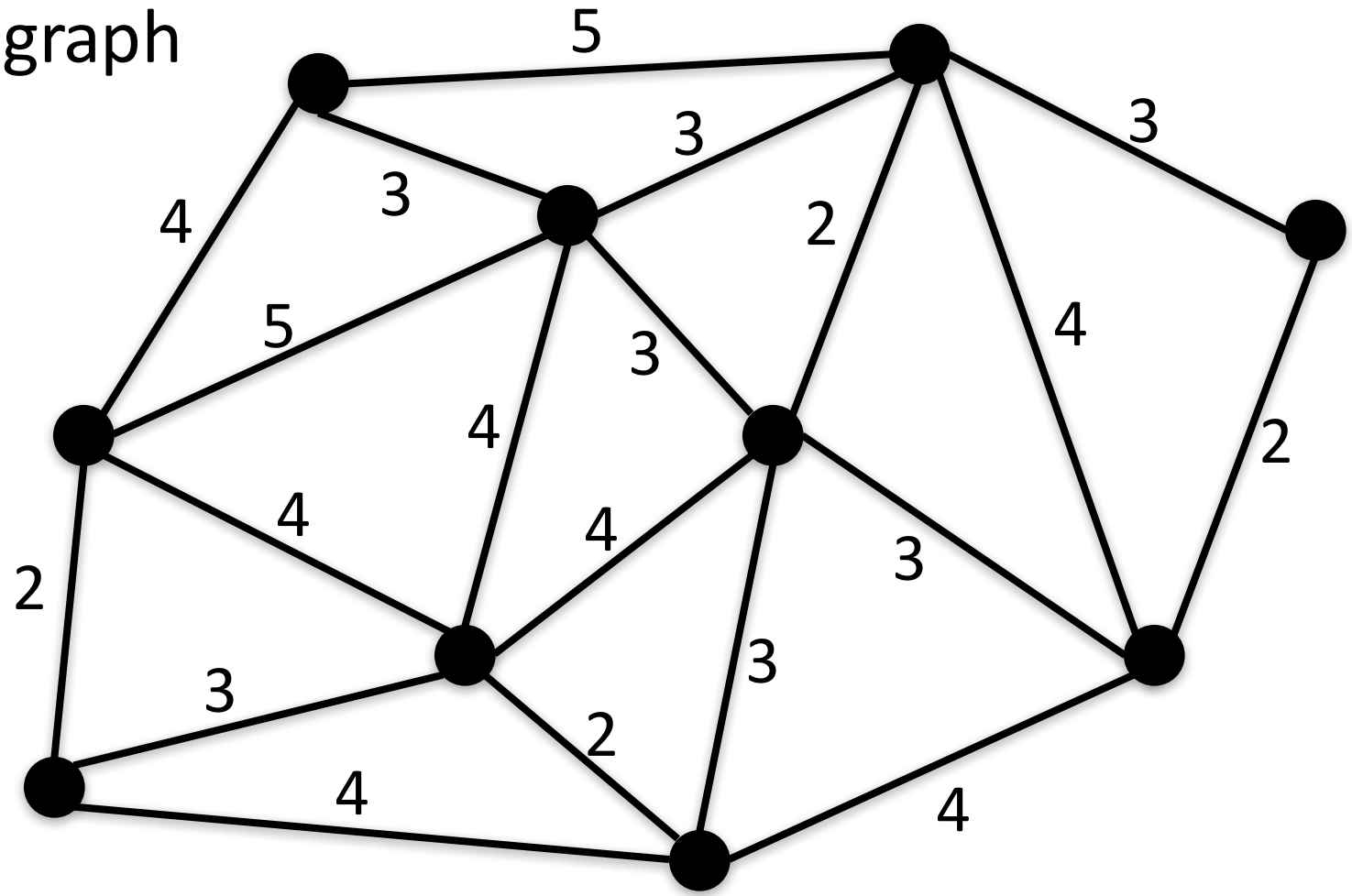
1. Enough streets must be paved so that it is possible for everyone to travel from their house to anyone else's house only along paved roads, and
2. The paving should cost as little as possible.

Here is the layout of the city. The number of paving stones between each house represents the cost of paving that route. Find the best route that connects all the houses, but uses as few counters (paving stones) as possible.


FOR SALE

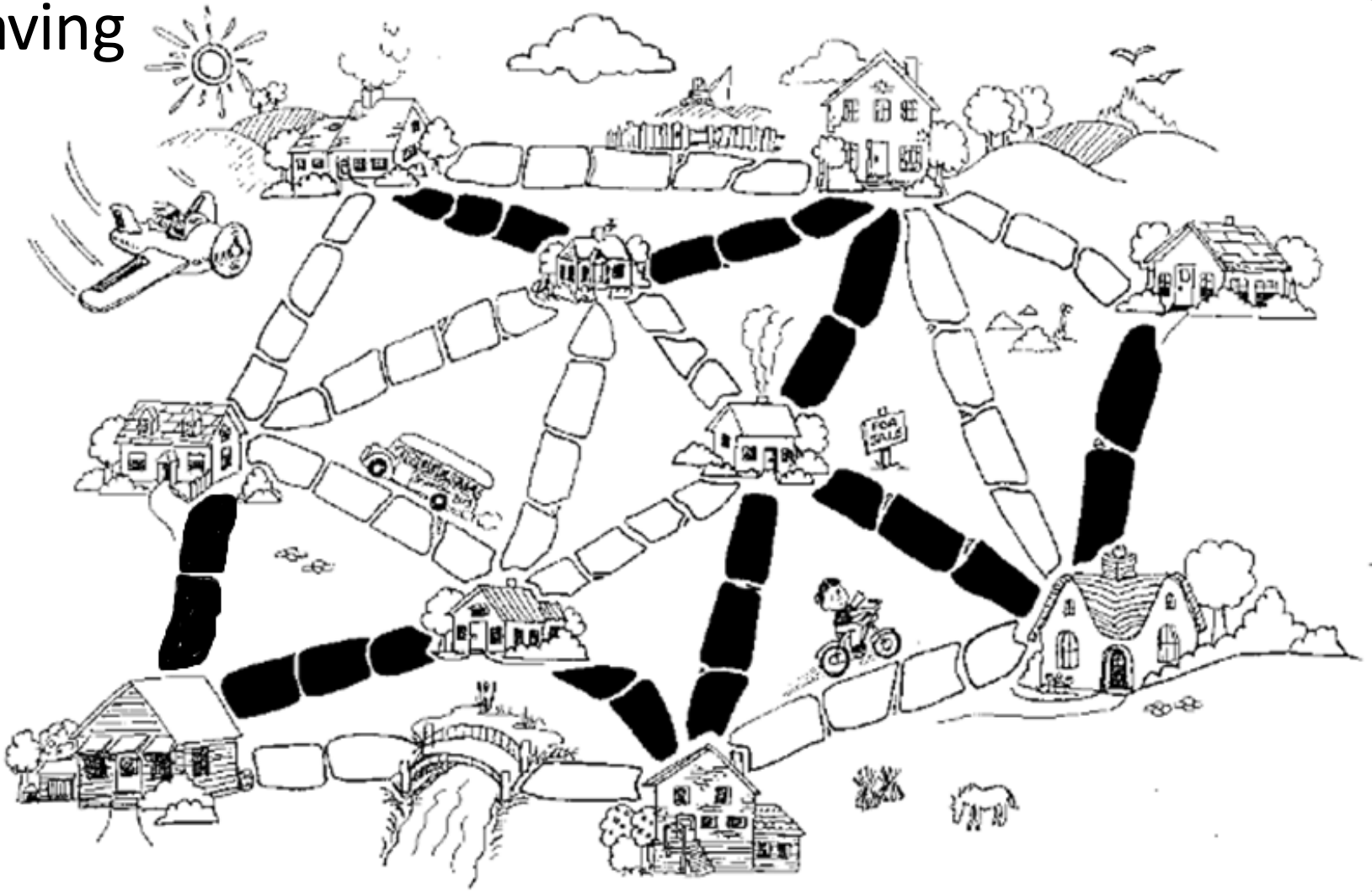
Muddy City Problem

The graph



Muddy City Problem

The paving



Application of MST: Examples

- In the design of electronic circuitry, it is often necessary to make a set of pins electrically equivalent by wiring them together.
- Running cable TV to a set of houses. What's the least amount of cable needed to still connect all the houses?

Finding MST

- Kruskal's Algorithm: start with no nodes or edges in the spanning tree and repeatedly add the cheapest edge that does not create a cycle.

Kruskal Algorithm

Procedure Kruskal (*G*: weighted connected undirected graph with *n* vertices)

T:= empty graph

for i := 1 to *n*-1

begin

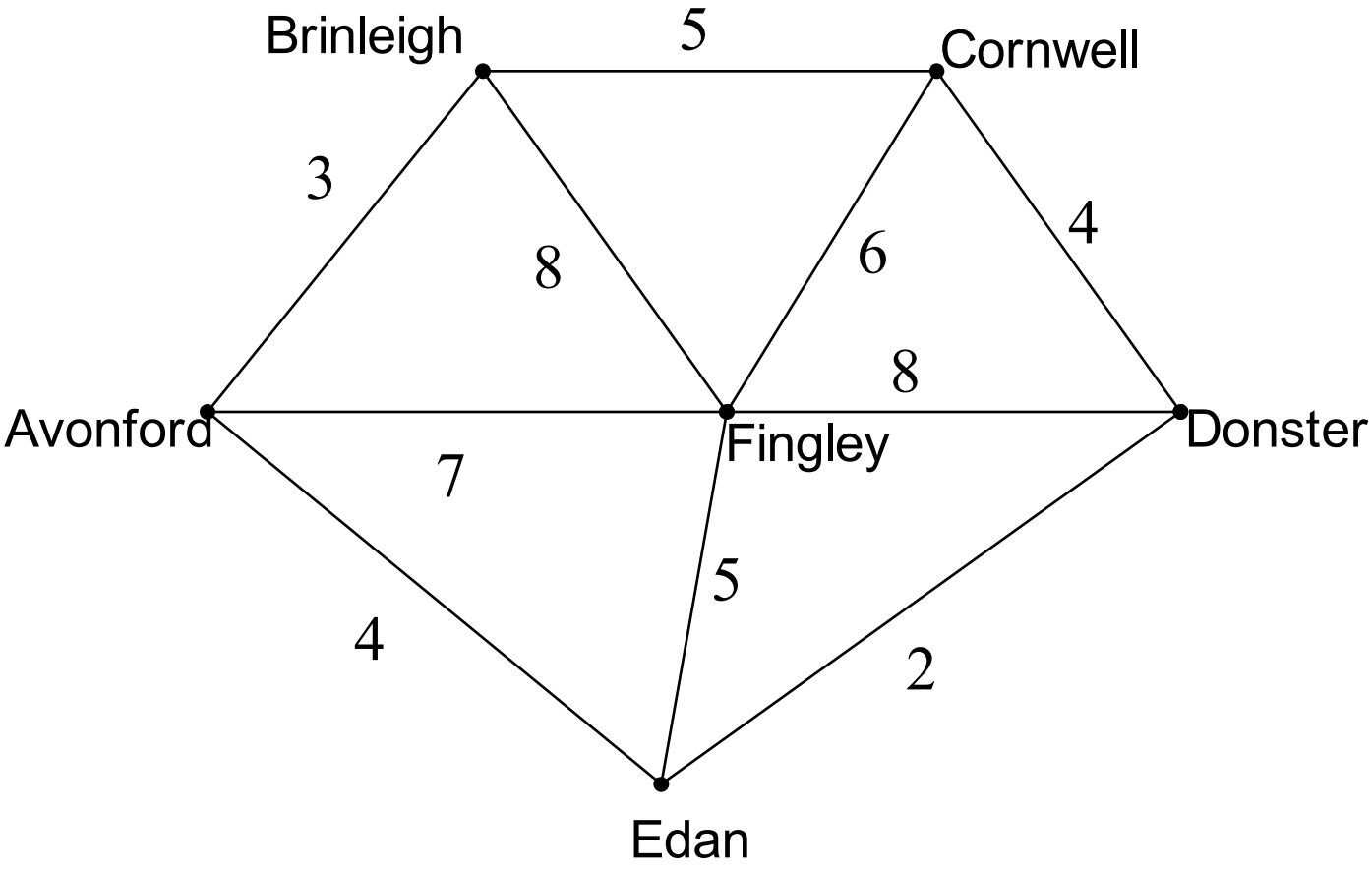
e:= any edge in *G* with smallest weight that does not form a simple circuit when added to *T*

T:= *T* with e added

end (*T* is a minimum spanning tree of *G*)

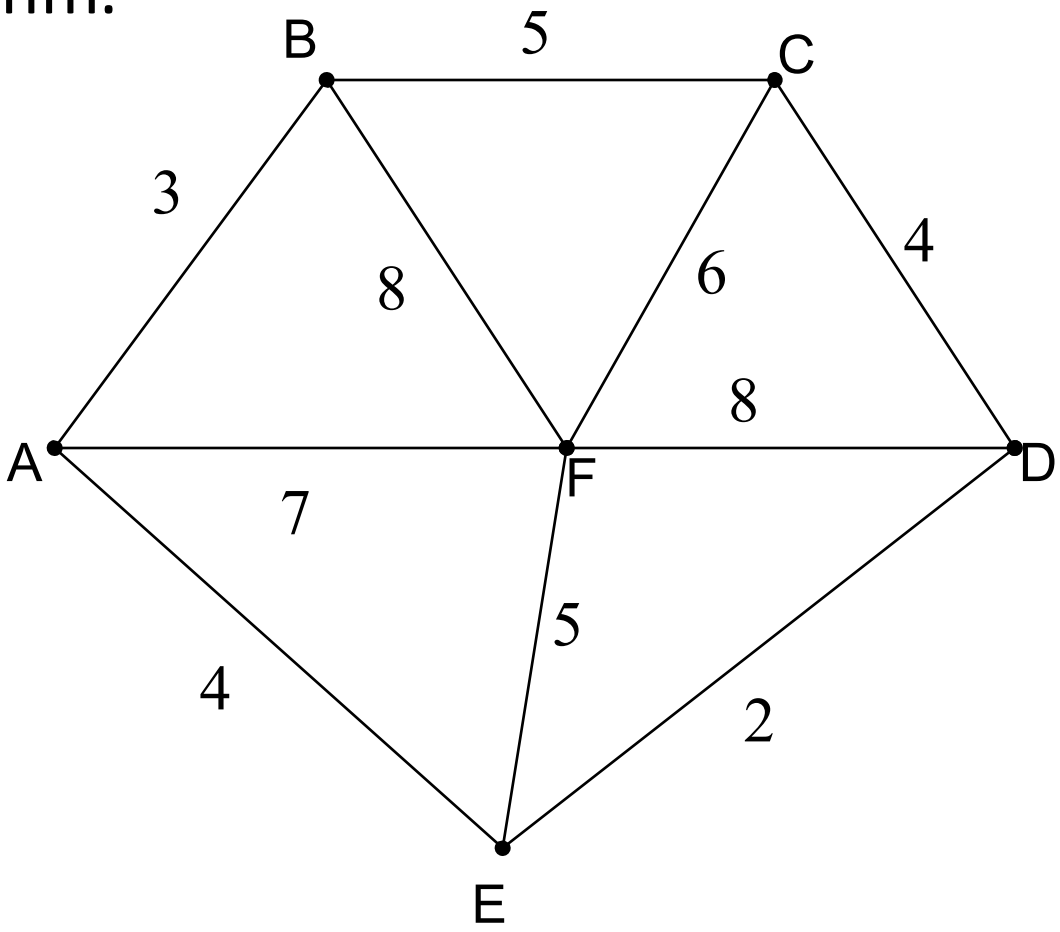
Example

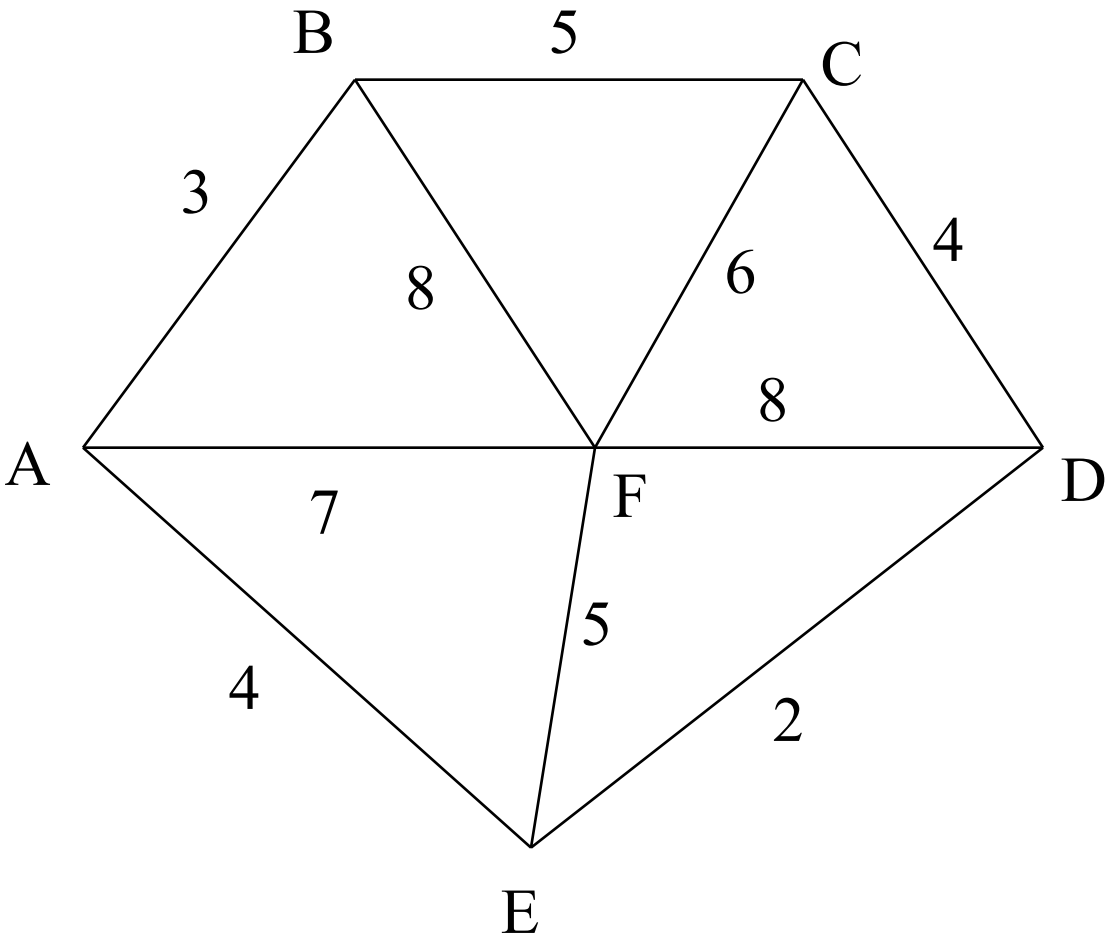
A cable company want to connect five villages to their network which currently extends to the market town of Avonford. What is the minimum length of cable needed?



Example - Solution

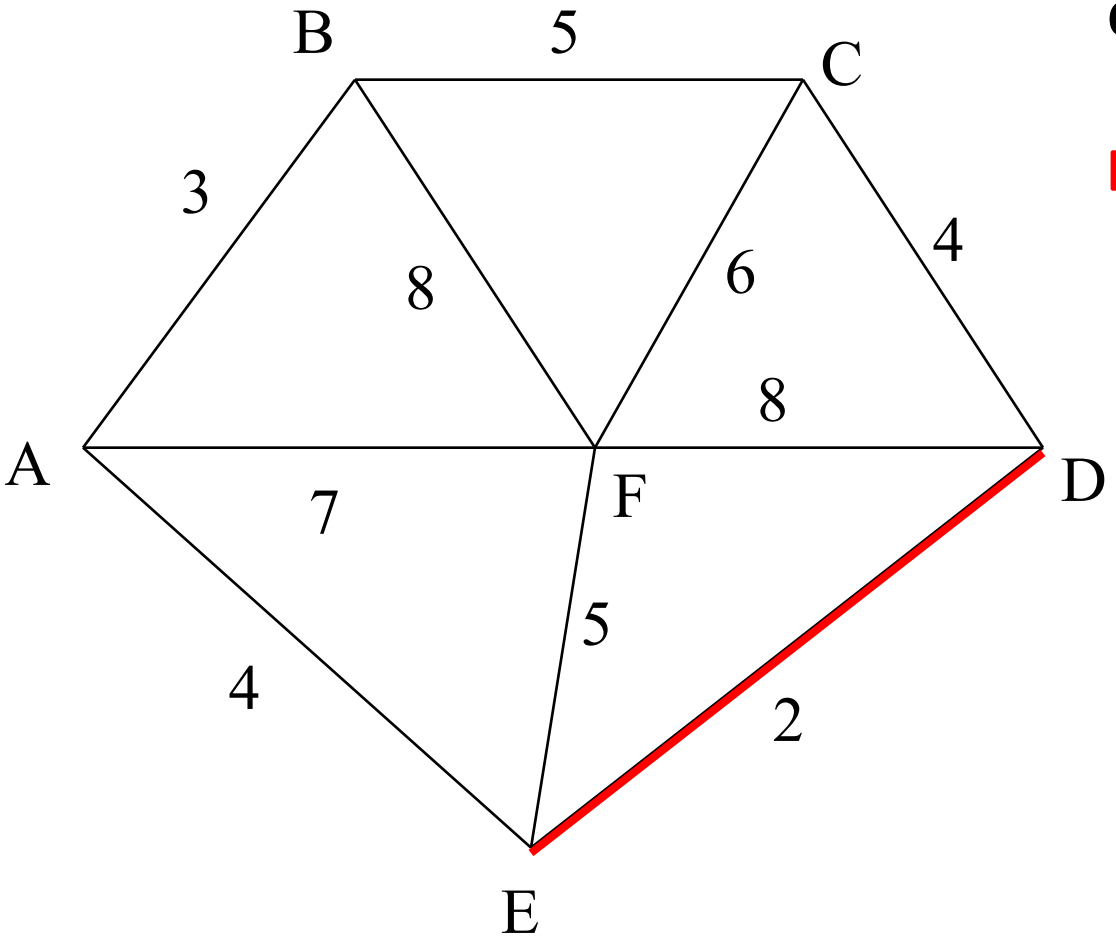
We model the situation as a network, then the problem is to find the minimum connector for the network. Use Kruskal's Algorithm.





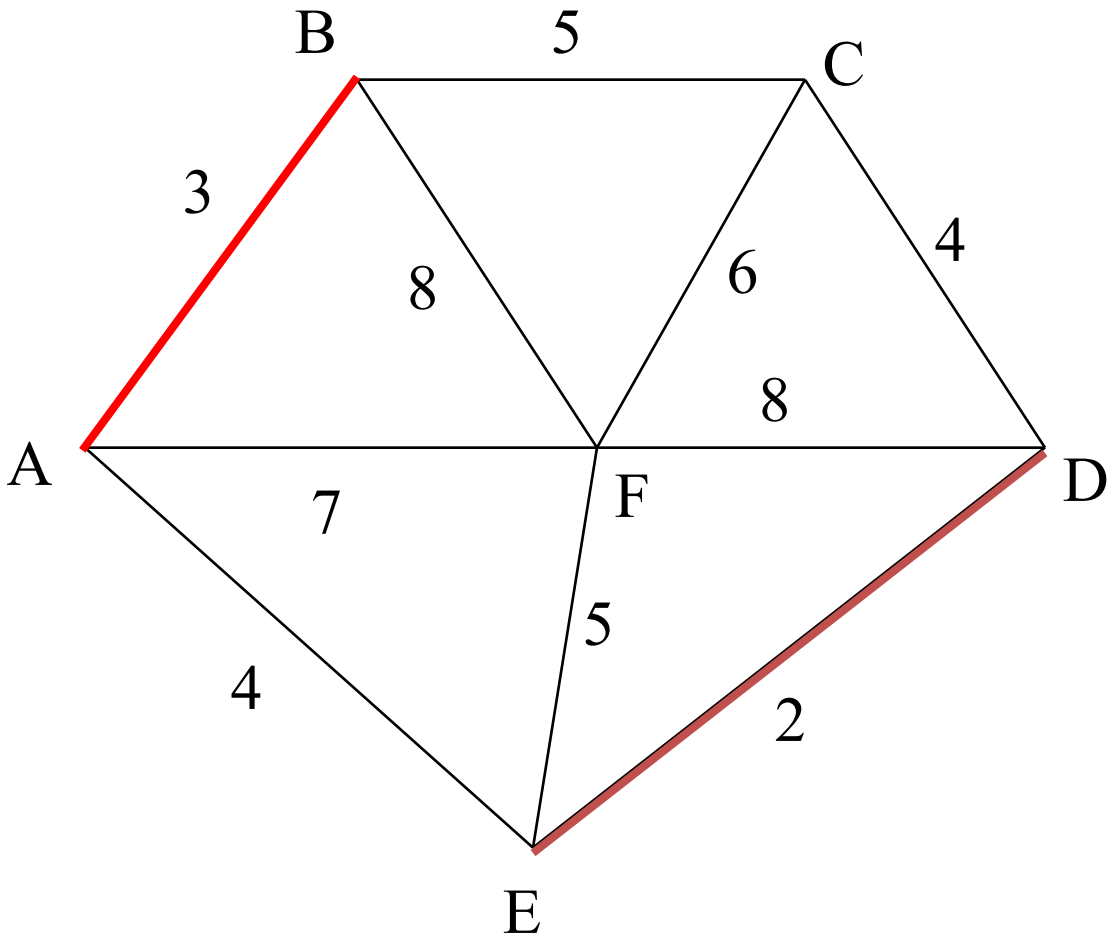
List the edges in order of size:

ED	2
AB	3
AE	4
CD	4
BC	5
EF	5
CF	6
AF	7
BF	8
CF	8



Select the shortest edge in the network

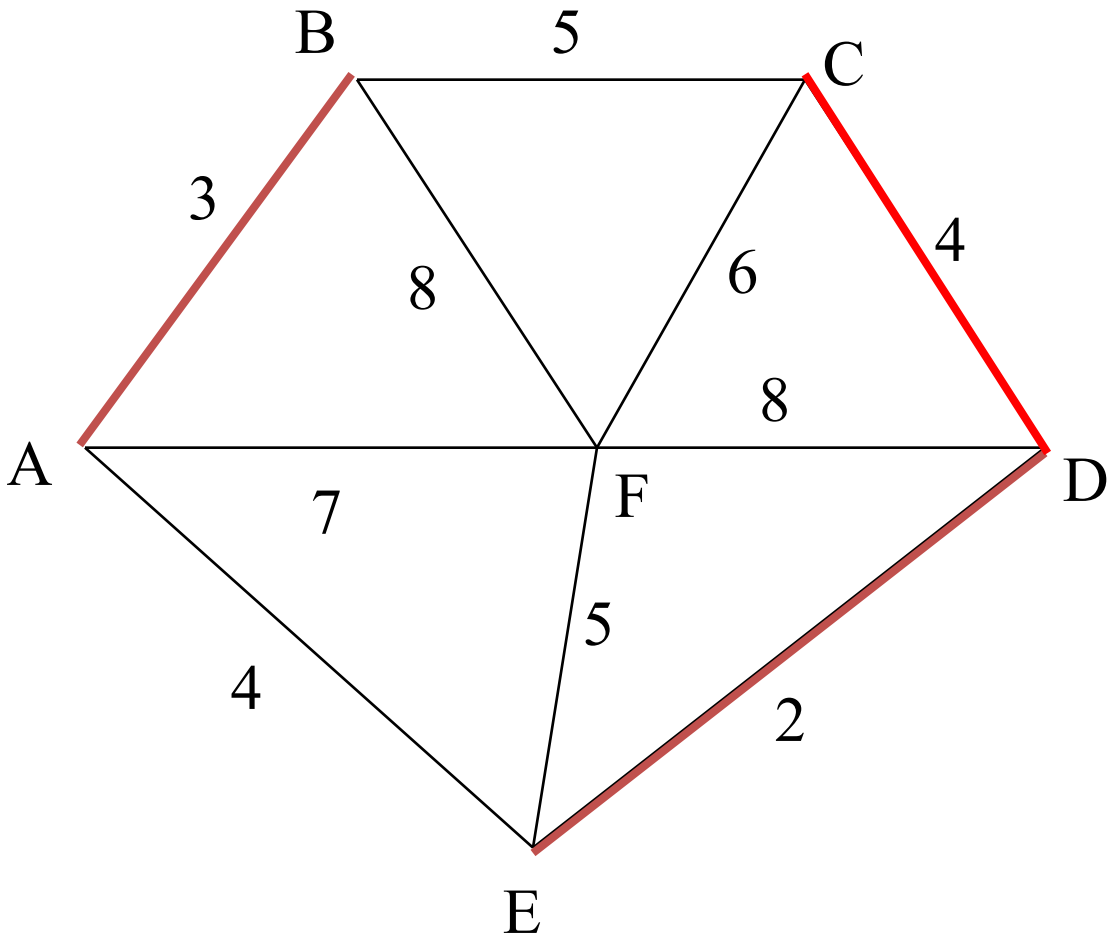
ED 2



Select the next shortest edge which does not create a cycle

ED 2

AB 3

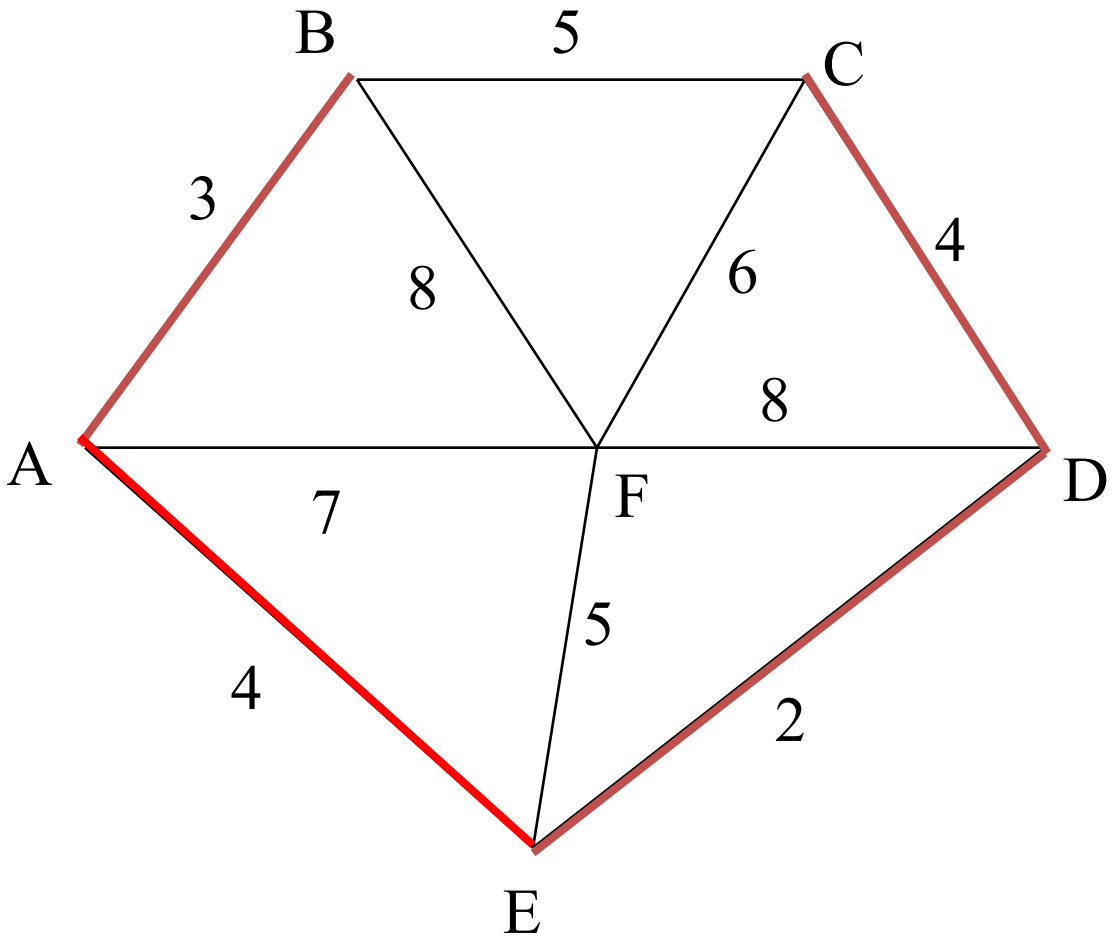


Select the next shortest edge which does not create a cycle

ED 2

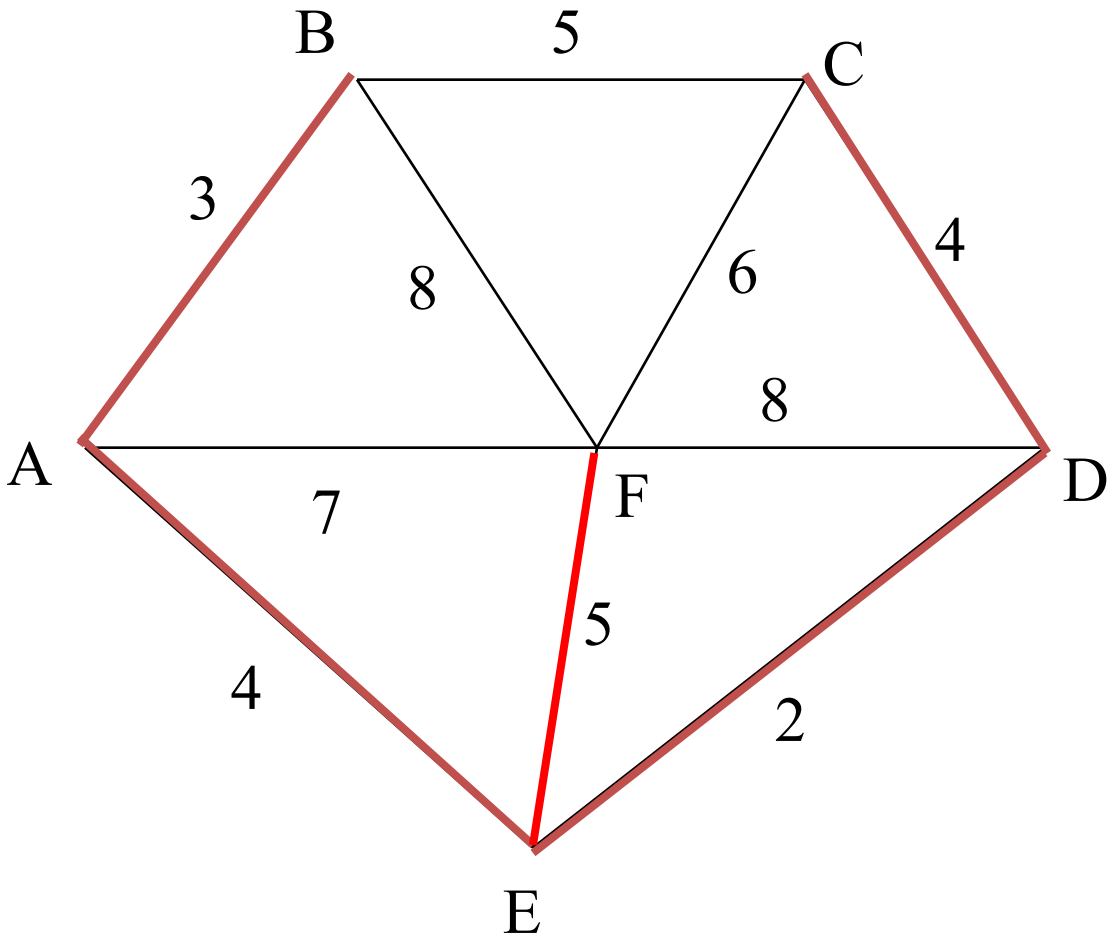
AB 3

CD 4 (or AE 4)



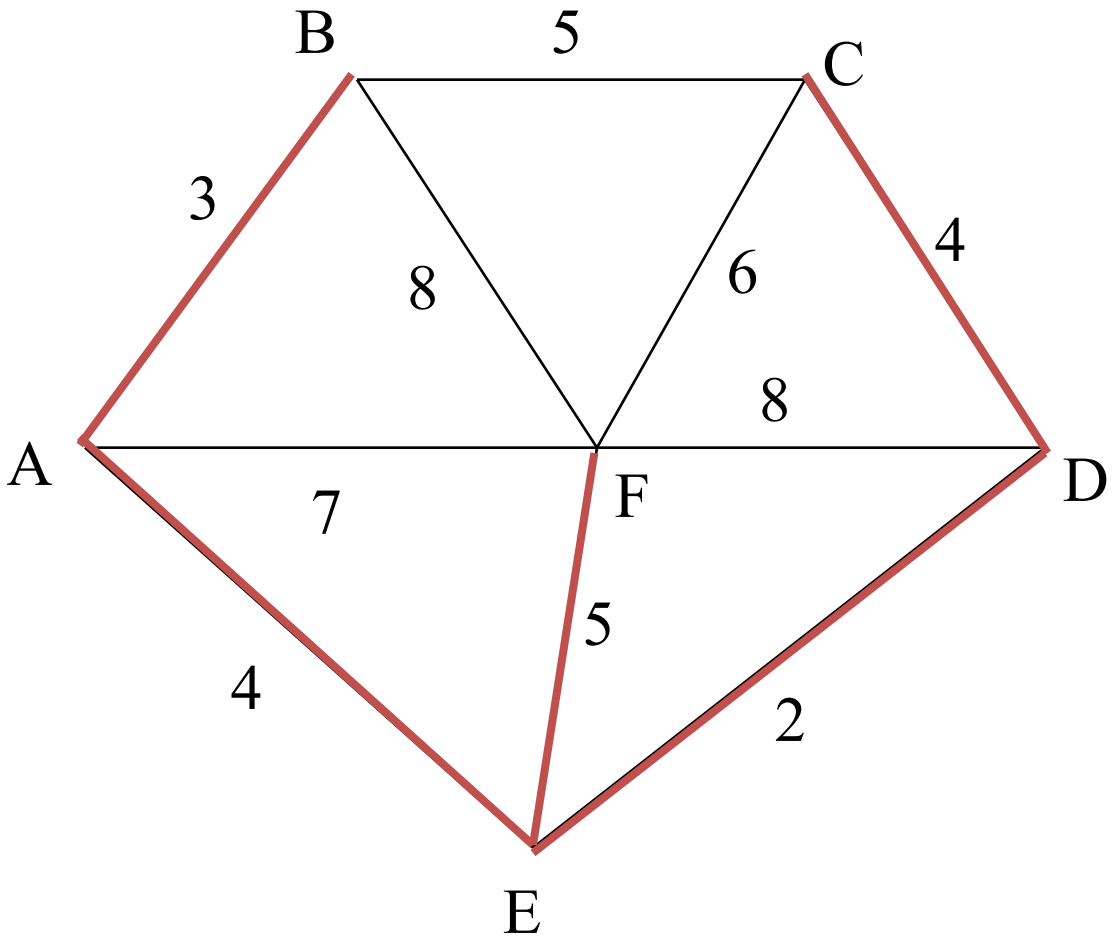
Select the next shortest edge which does not create a cycle

ED 2
AB 3
CD 4
AE 4



Select the next shortest edge which does not create a cycle

ED 2
AB 3
CD 4
AE 4
~~**BC 5**~~ **– forms a cycle**
EF 5



All vertices have been connected.

The solution is

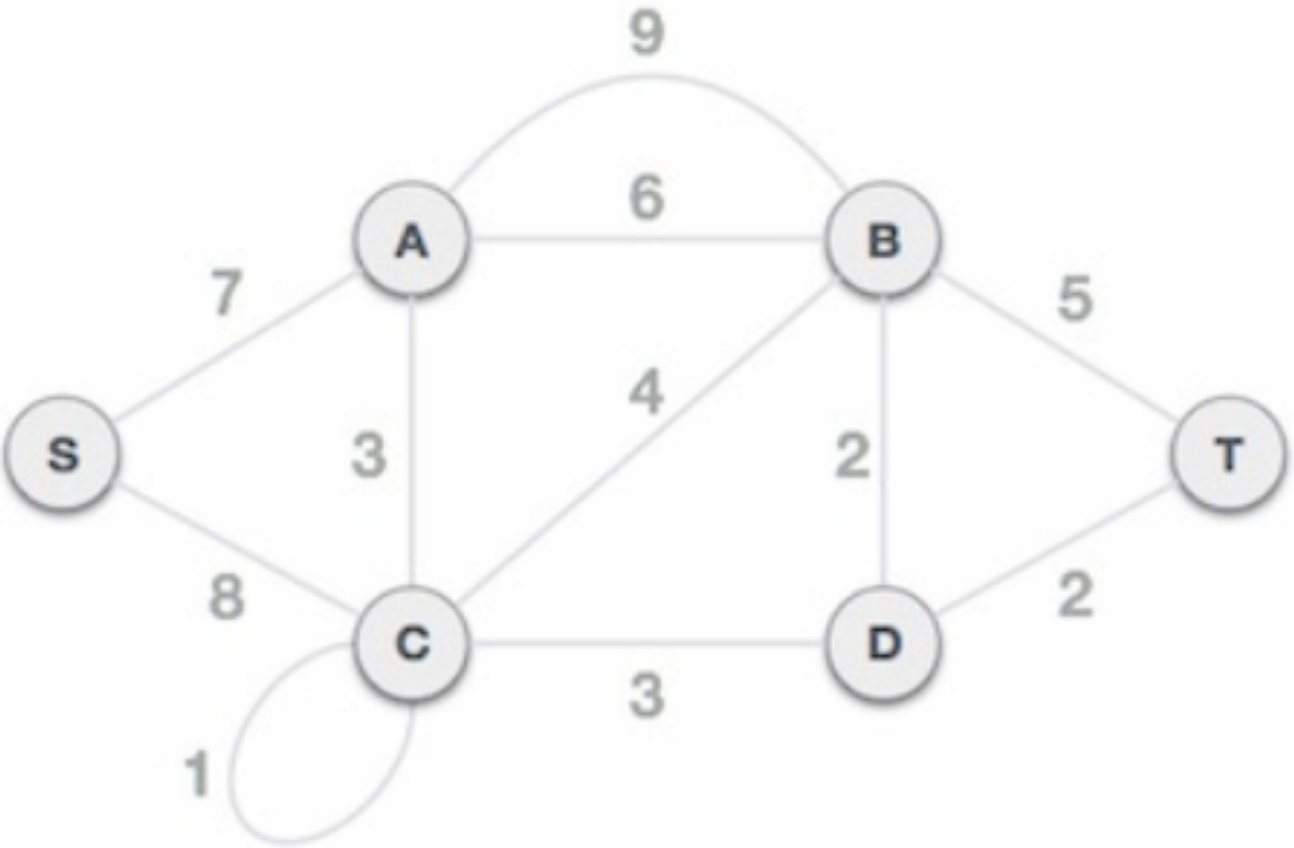
ED 2
AB 3
CD 4
AE 4
EF 5

Total weight of tree: 18

Important notes:

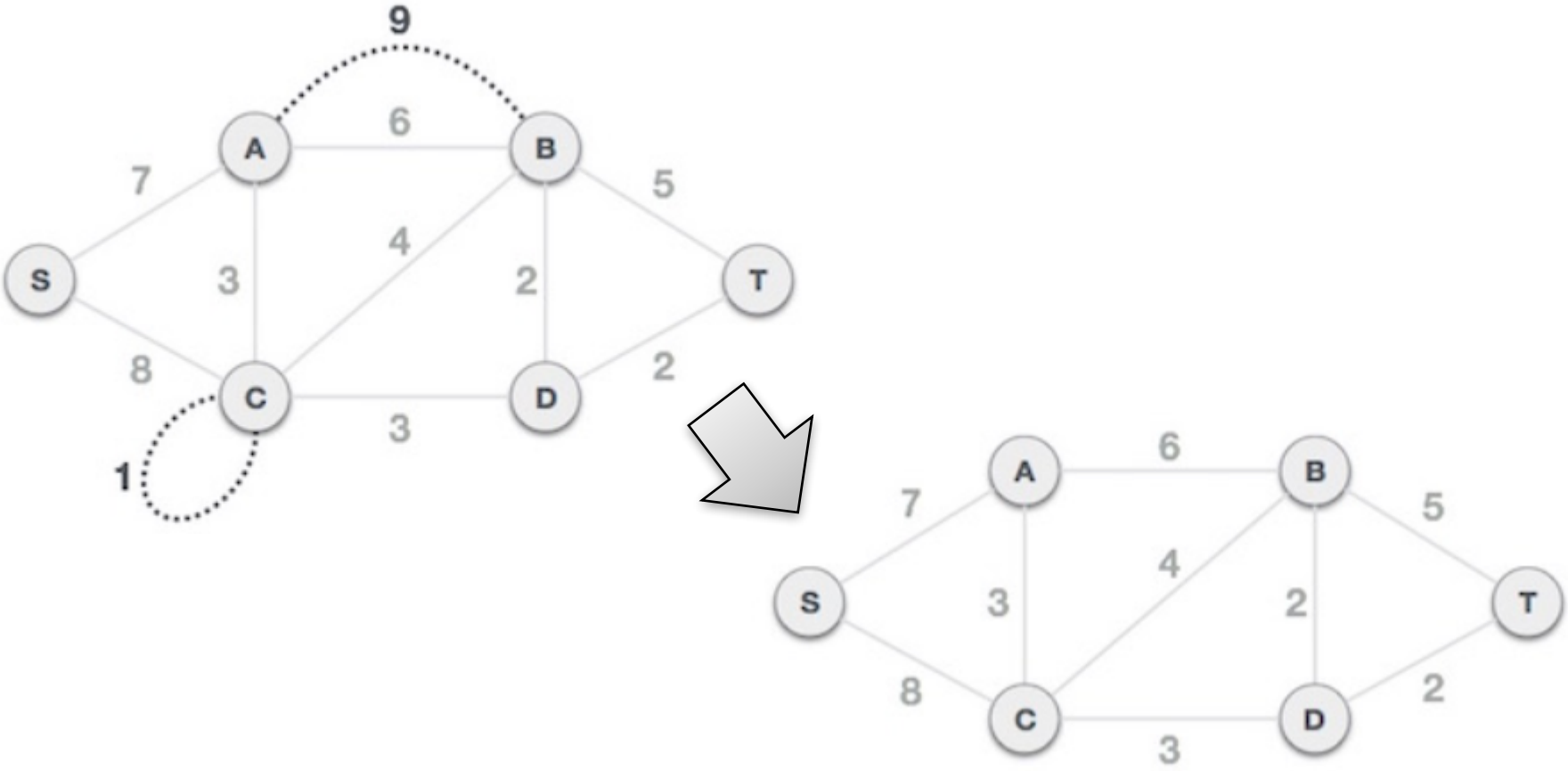
- The graph given should be a tree
  - Remove all loops (if any)
  - Remove all parallel edges
    - keep the one which has the least weight associated and remove all others.

Example



Example

- Remove all loops and parallel edges



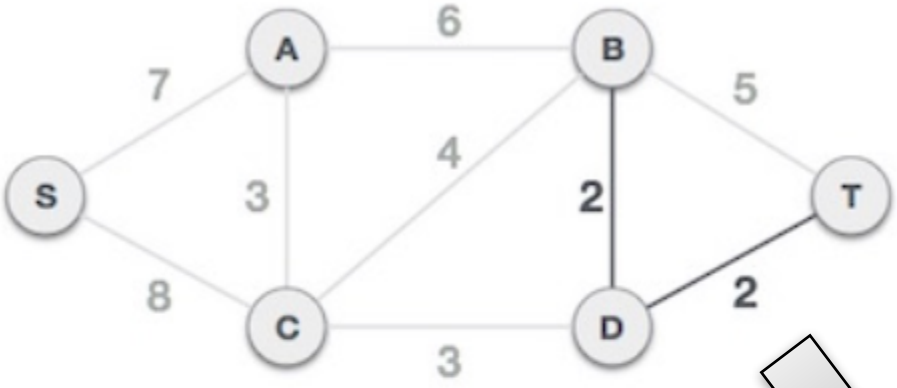
Example

- Arrange all edges in their increasing order of weight

BD	DT	AC	CD	CB	BT	AB	SA	SC
2	2	3	3	4	5	6	7	8

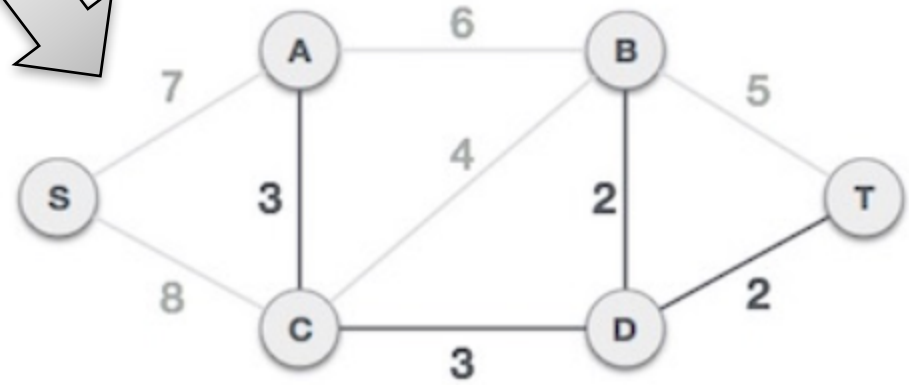
Example

- Add the edge which has the least weightage

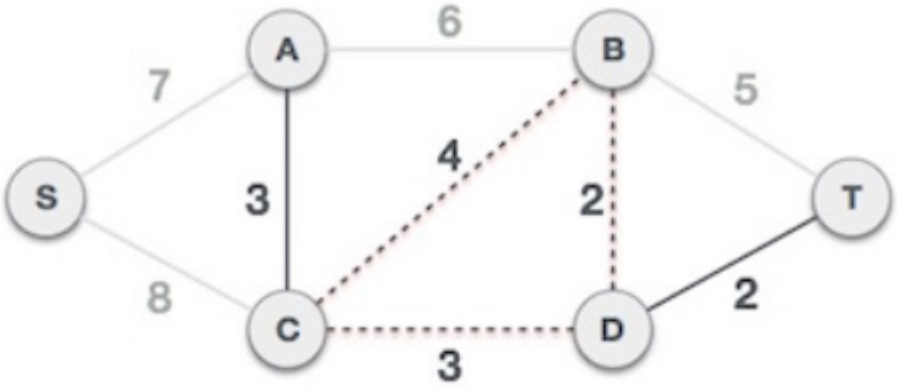


The least weight is 2 and edges involved are BD and DT

Next weight is 3, and associated edges are AC and CD

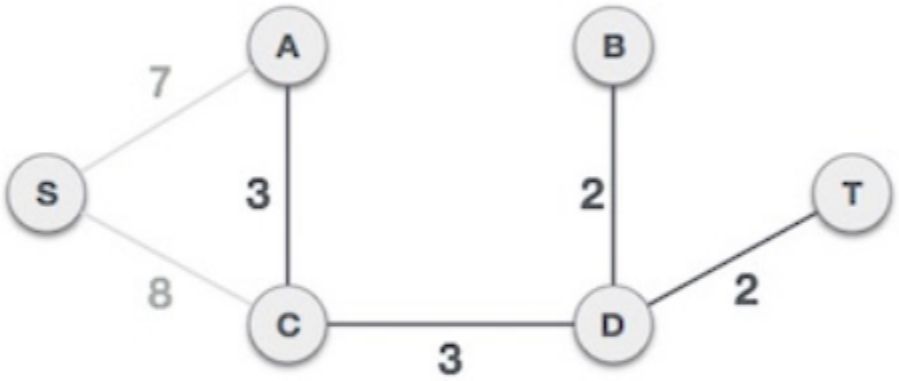


Example



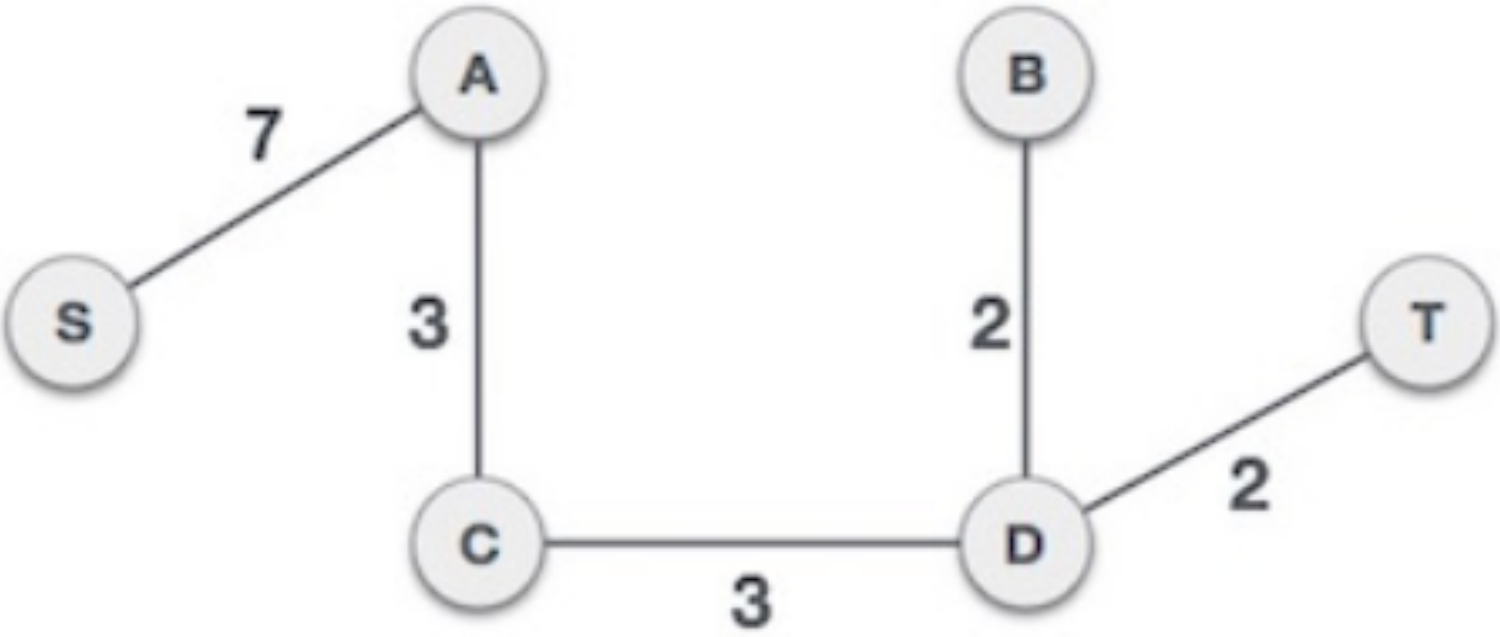
Next weight is 4, and we observe that adding it will create a circuit in the graph. Thus, we ignore it.

We observe that edges with weight 5 and 6 also create circuits. We ignore them and move on.



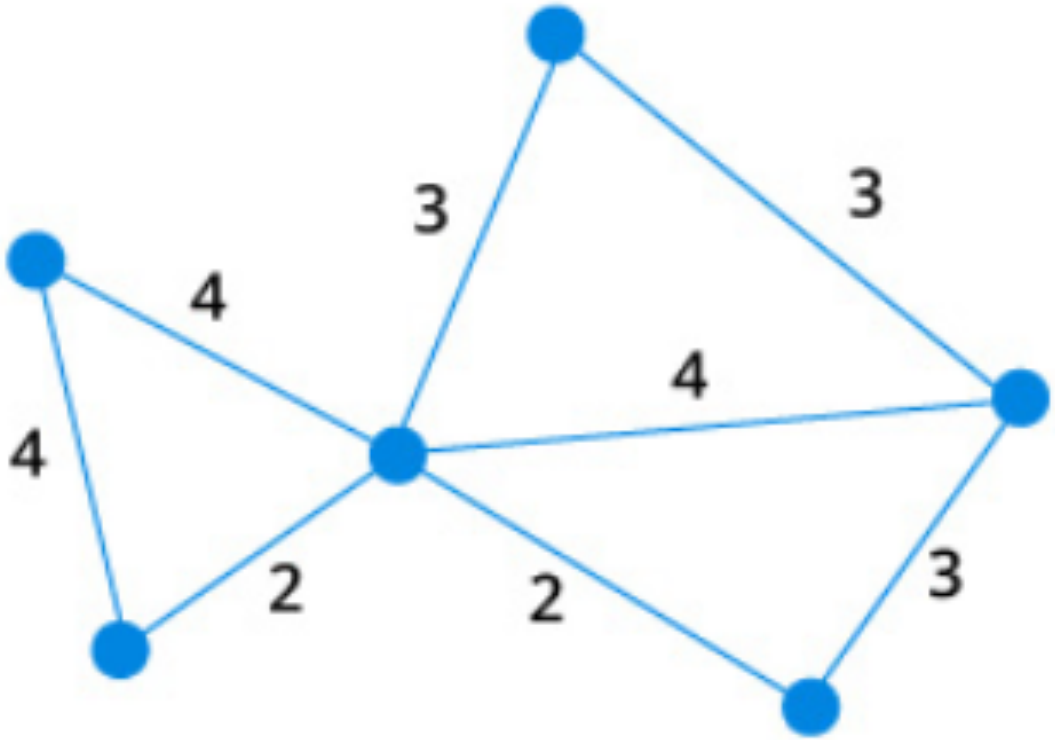
Now we are left with only one node to be added. Between the two least weighted edges available 7 and 8, we shall add the edge with weight 7.

Now we have minimum spanning tree with total weight is 17.



Exercise

Find the minimum spanning tree using Kruskal's Algorithm and give the total weight for the minimum spanning tree.



Exercise

Find the minimum spanning tree using Kruskal's Algorithm and give the total weight for the minimum spanning tree.