SCSI 1013 DISCRETE STRUCTURE

CHAPTER 4 – PART 2

TREE

Trees

- Trees were used as long ago as 1857, when the English mathematician Arthur Cayley used them to count certain types of chemical compounds.
- Trees are used wide variety of applications such as family trees, organizational charts, biology and computer file structures.
- Useful in computer science, where they are employed in a wide range of algorithms.

Introduction

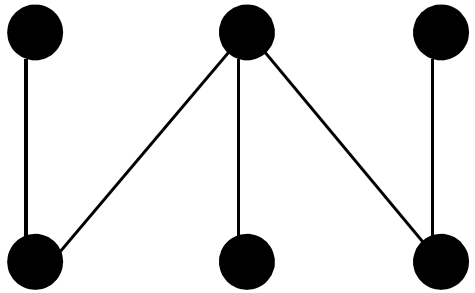
Definition 1. A tree is a connected undirected graph with no simple circuits(acyclic).

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

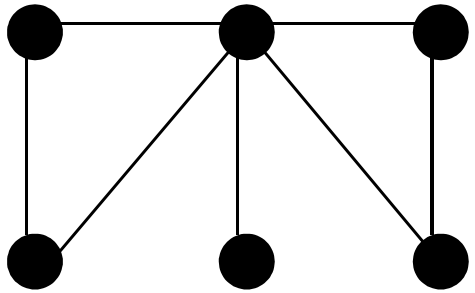
Theorem 2 . A tree with m-vertices has m-1 edges

Example 1 :Which graphs are trees?

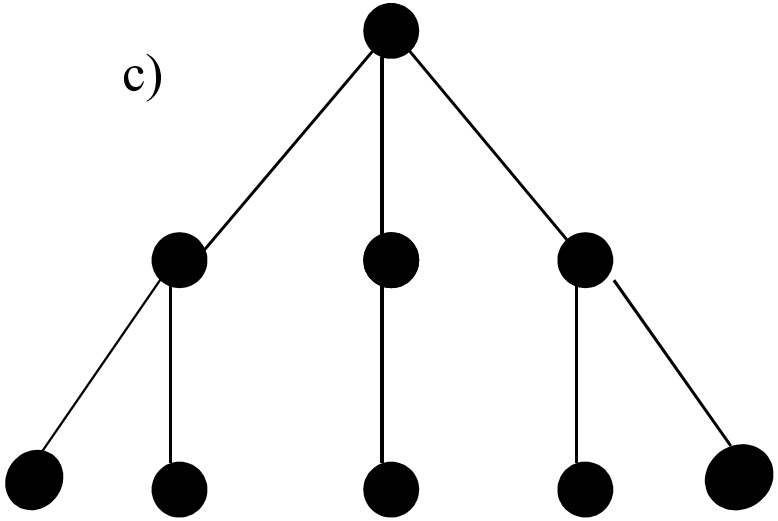
a)



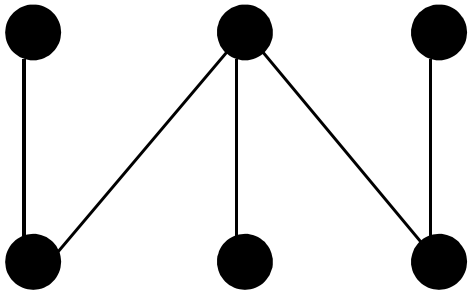
b)



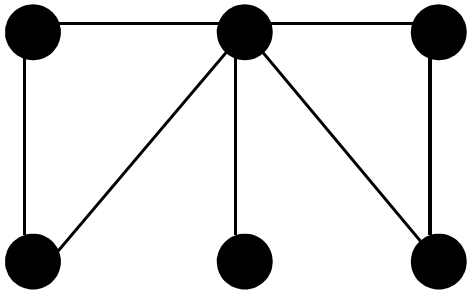
c)



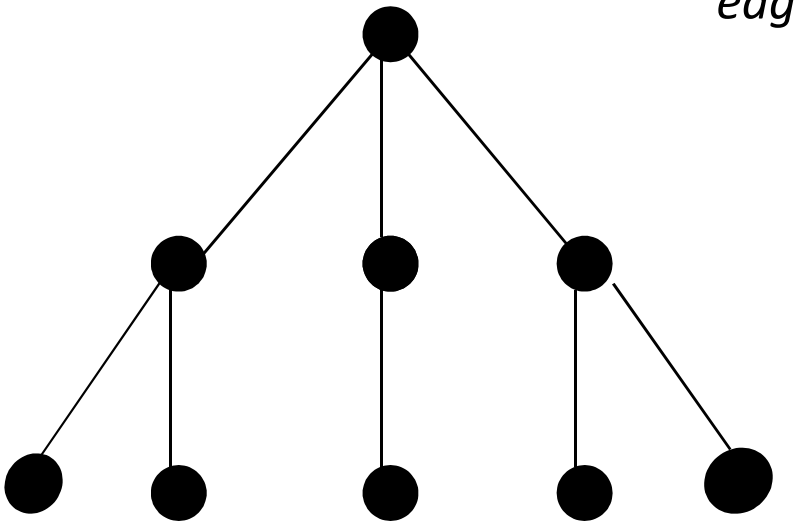
Example 1 : Solution



tree
vertices = 6
edges = 5



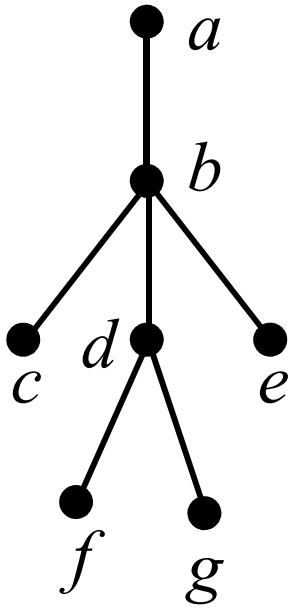
Not a tree
vertices = 6
edges = 7



tree
vertices = 9
edges = 8

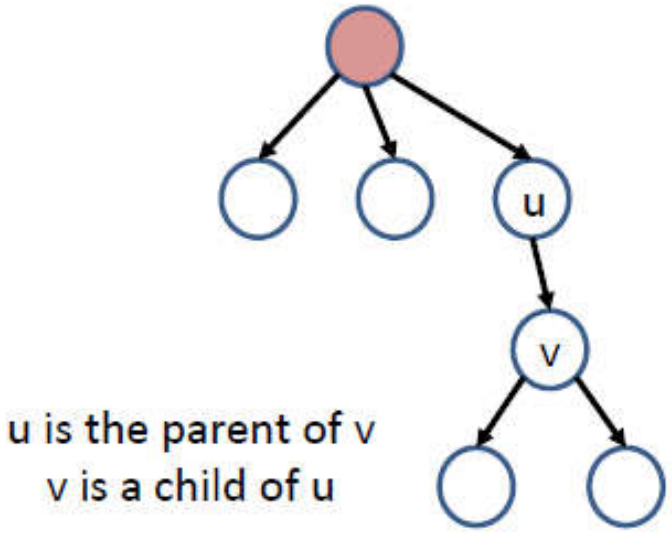
Rooted tree

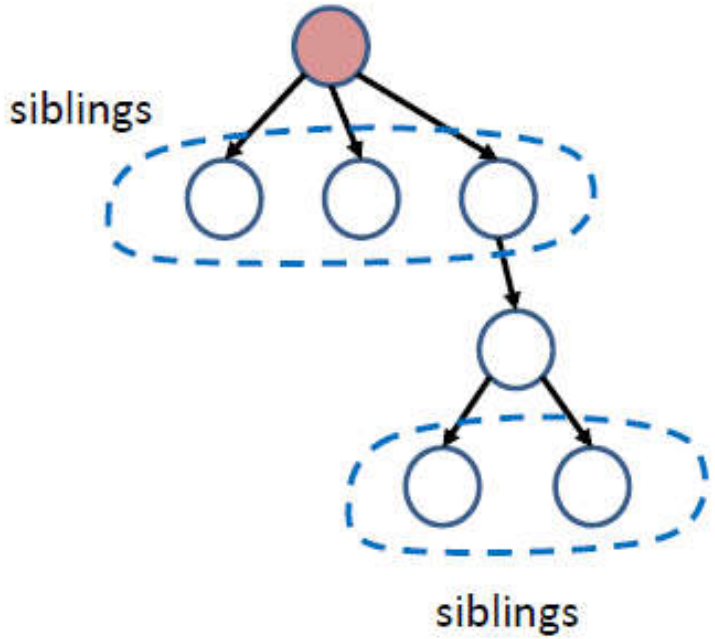
Definition 2. A **rooted tree** is a tree in which one vertex has been designed as the **root** and every edge is directed away from the root.



Rooted Tree - Terminologies

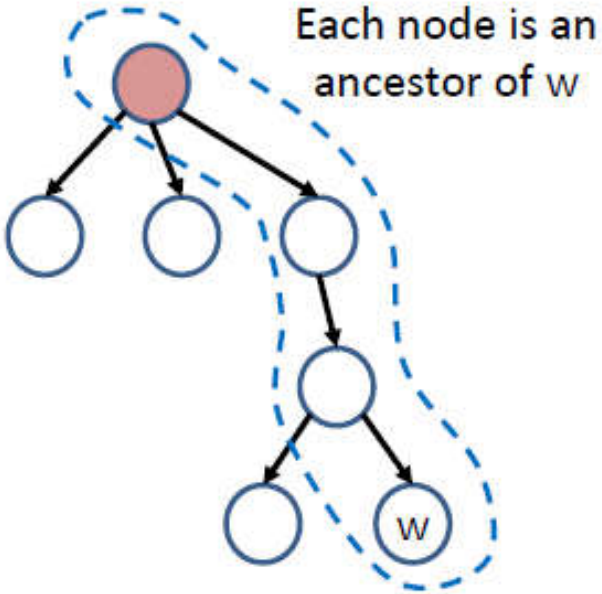
- Each edge is from a parent to a child
- Vertices with the same parent are siblings





Rooted Tree - Terminologies

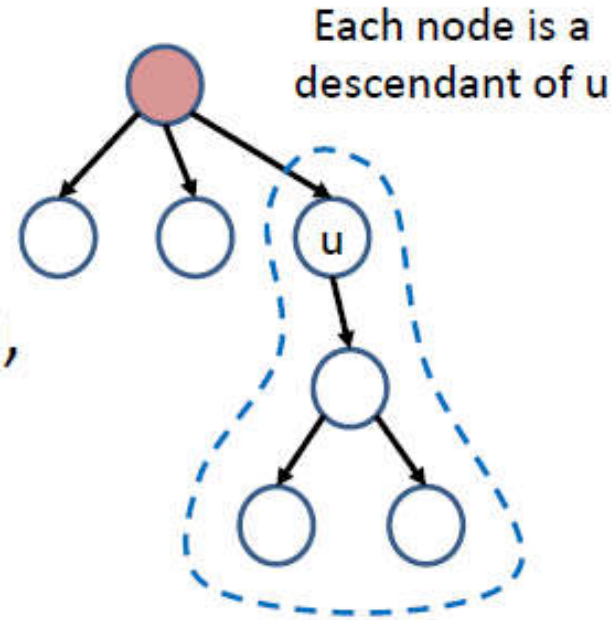
- The ancestors of a vertex w include all the nodes in the path from the root to w
- The proper ancestors of a vertex w are the ancestors of w, but excluding w



The whole part forms a path from root to w

Rooted Tree - Terminologies

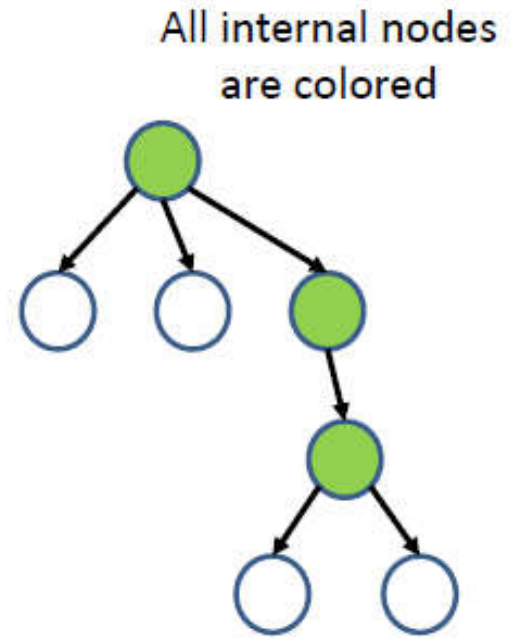
- The descendants of a vertex u include all the nodes that have u as its ancestor
- The proper descendants of a vertex u are the descendants of u, but excluding u
- The subtree rooted at u includes all the descendants of u, and all edges that connect between them



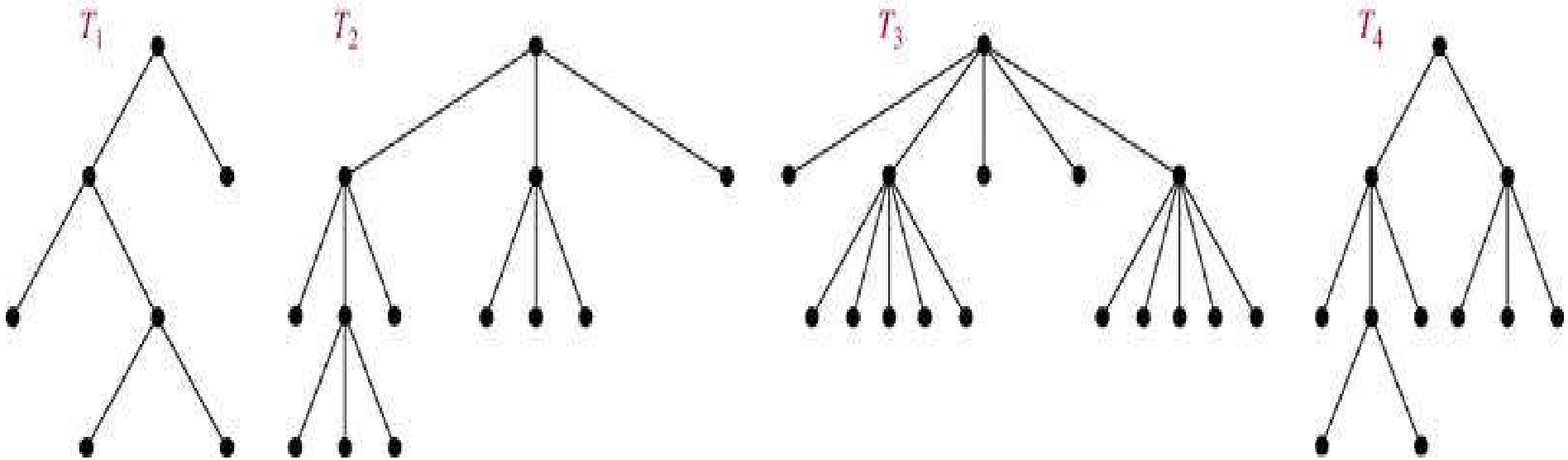
Rooted Tree - Terminologies

- 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



Examples 2



full binary tree

full 3-ary tree

full 5-ary tree

not full 3-ary tree

Suppose that T is a full m-ary tree.Let:

- *n* - the number of vertices in a tree.
- *i* - the number of internal vertices
- *l* - the number of leaves in this tree.

Once one of *n*, *i*, and *l* is known, the other two quantities are determined

Properties of Trees

Theorem 3(i) – A full **m-ary** tree with ***n*** vertices has

$$i = \frac{n \quad 1}{m} \qquad l = \frac{(m \quad 1)n + 1}{m}$$

Properties of Trees

Theorem 3ii)– A full **m-ary** tree with ***i*** internal vertices has

$$n = mi + 1 \qquad l = (m \quad 1)i + 1$$

Properties of Trees

Theorem 3iii)– A full **m-ary** tree with l leaves has

$$n = \frac{ml \quad 1}{m \quad 1} \qquad i = \frac{l \quad 1}{m \quad 1}$$

Example 3

Ex : Peter starts out a chain mail. Each person receiving the mail 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 ?

Example 3 :Solution

- The chain letter can be represented using **4-ary** tree. The internal vertices correspond to people who sent out the letter, and the leaves correspond to people who did not send it out. Since 100 people did not send out the letter, the number of leaves in this rooted tree is, *l*=100. The number of people have seen the letter is

$i = \frac{100-1}{4-1} = \frac{99}{3} = 33$ or *n=(4x100-1)/(4-1)=133*

133-100=33

33 people sent the letter.

Example 4

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.)

Example 4 :Solution

We can model this tournament with a full binary tree. We know we have 1000 leaves for this tree because we have 1000 people. Each internal vertex will represent the winner of the game played by its children. The root will be the winner of the tournament.

Example 4 :Solution

With $m = 2$ and $l = 1000$. We know that:

$$i = \frac{(1000 - 1)}{(2 - 1)}$$

$$= \frac{999}{1}$$

$$= 999$$

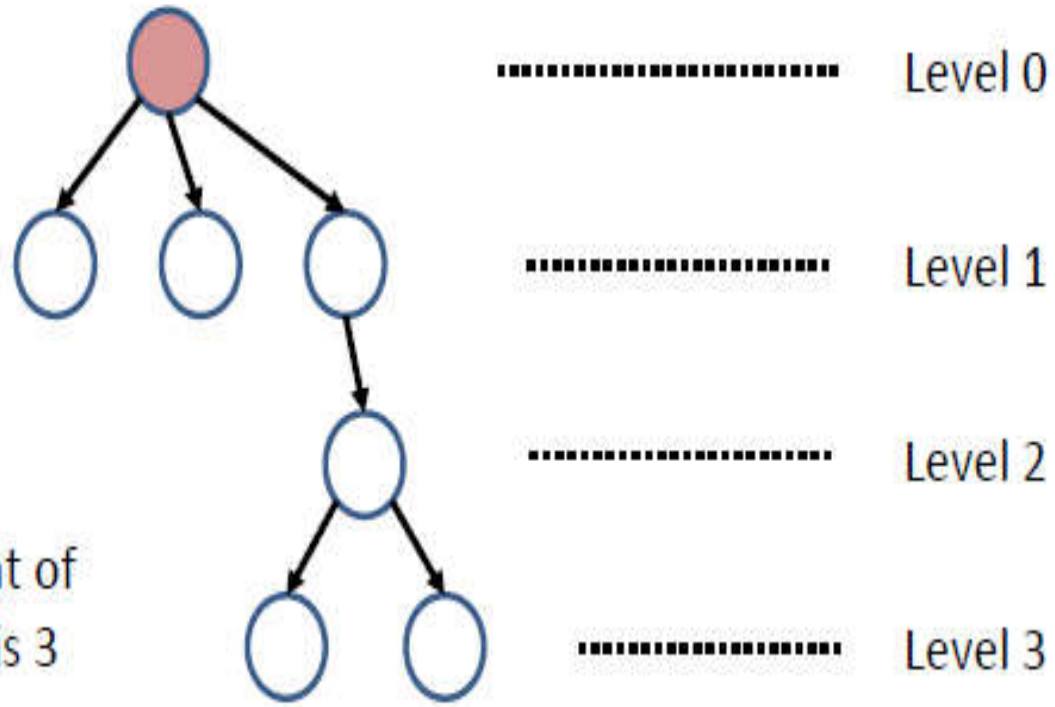
We have 999 internal vertices, so we know 999 games must be played to determine the champion.

Properties of 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 5

- Ex :



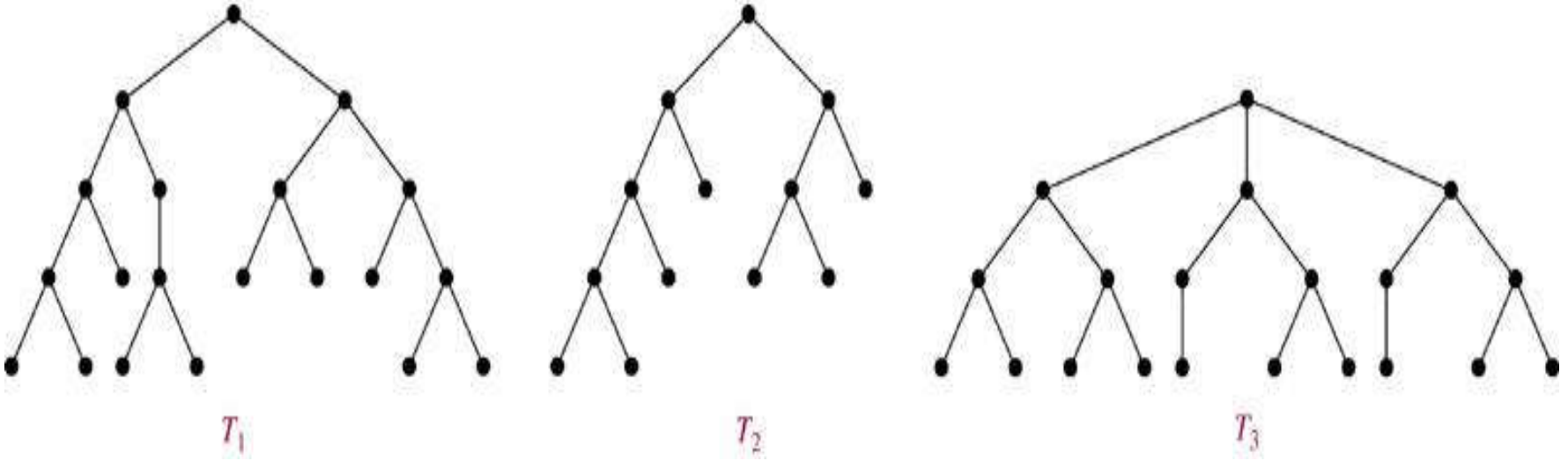
Properties of Trees

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

- **Theorem 4.** There are at most m^h leaves in an m-ary tree of height h.

Example 6

Which of the rooted trees shown below are balanced?



Sol.

Tree Traversal

Universal Address Systems

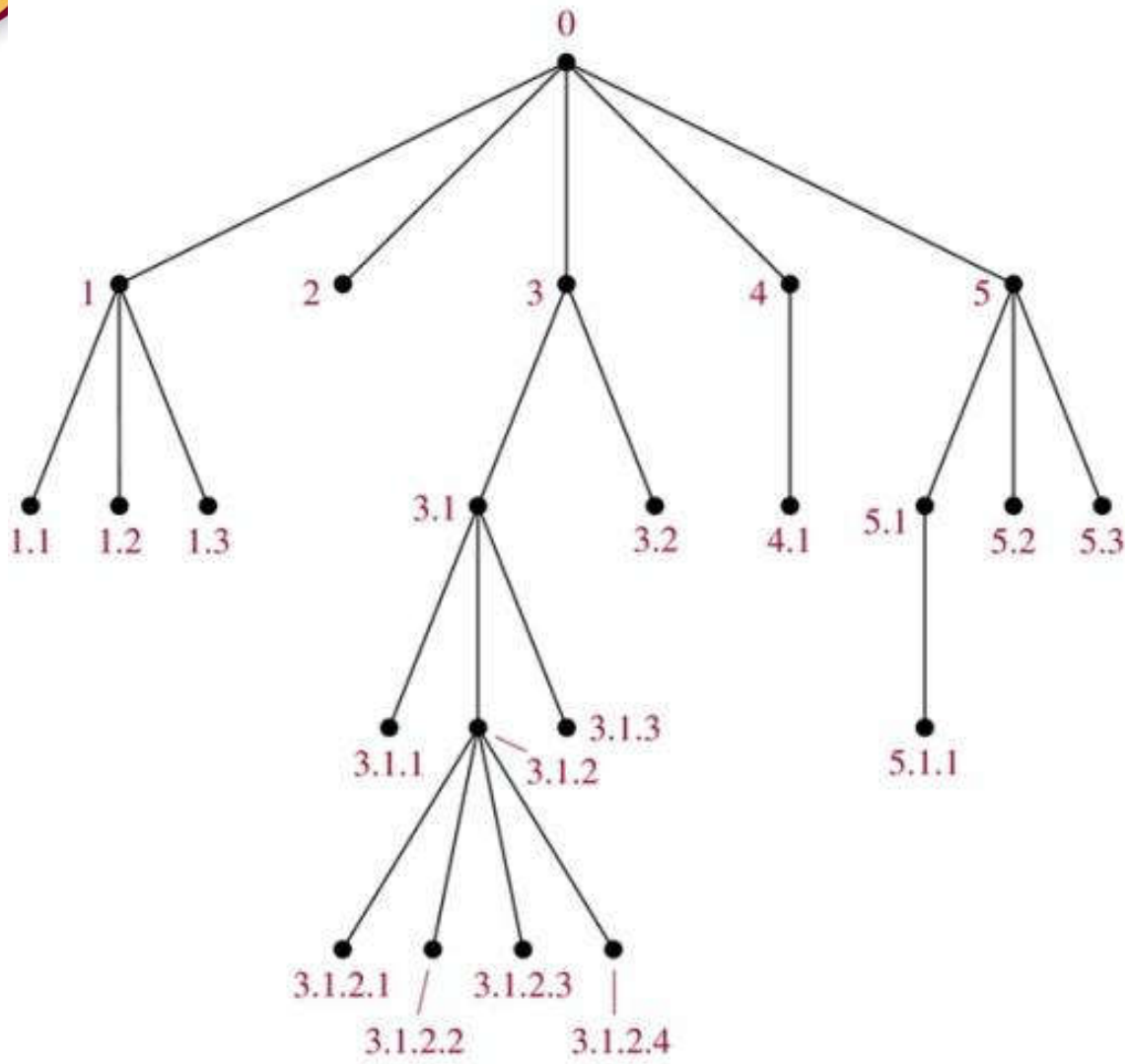
Label vertices:

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.

$x_1.x_2.\ldots.x_n < y_1.y_2.\ldots.y_m$
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:

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

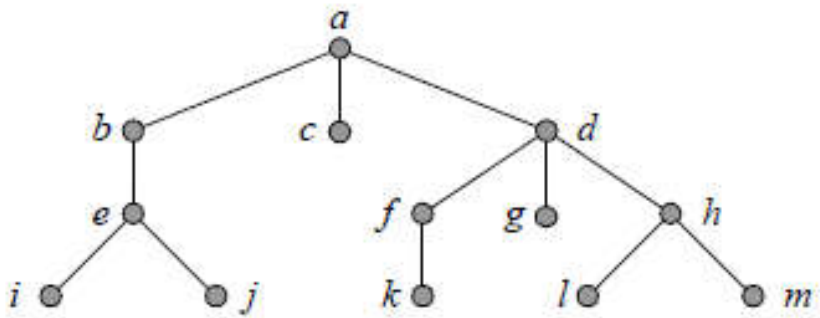
Tree Traversal

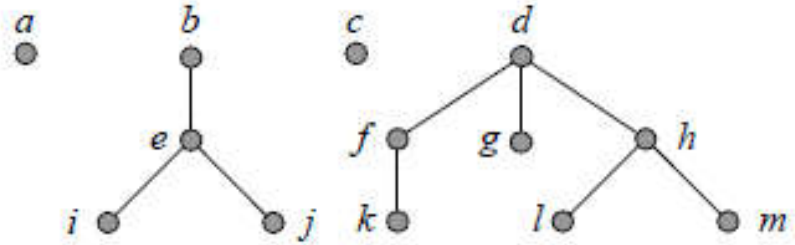
- Preorder: **root**, **left**-subtree, **right** subtree
- Inorder – **left** subtree, **root**, **right** sub-tree
- Post-order : **left** subtree, **right** sub-tree, **root**

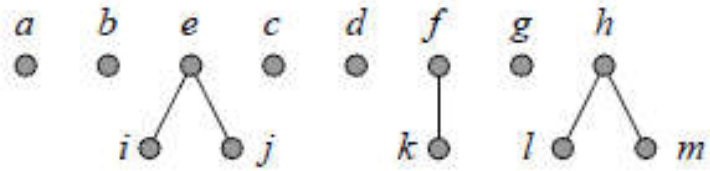
Preorder Traversal

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

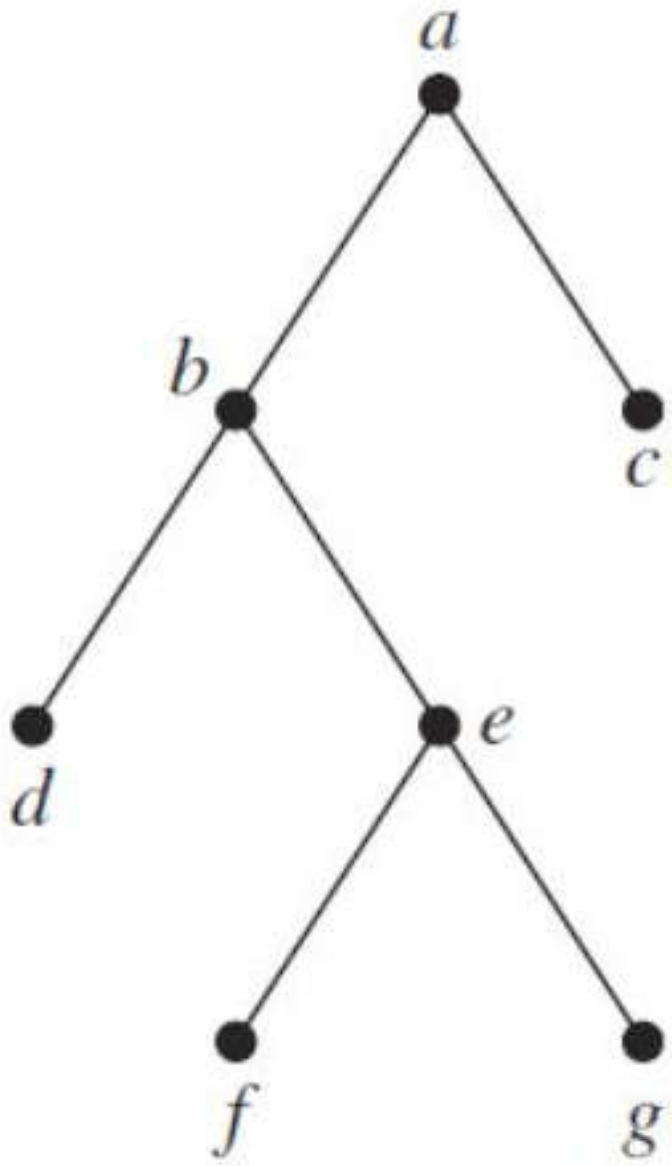
Example 7
Preorder Traversal







Example 8

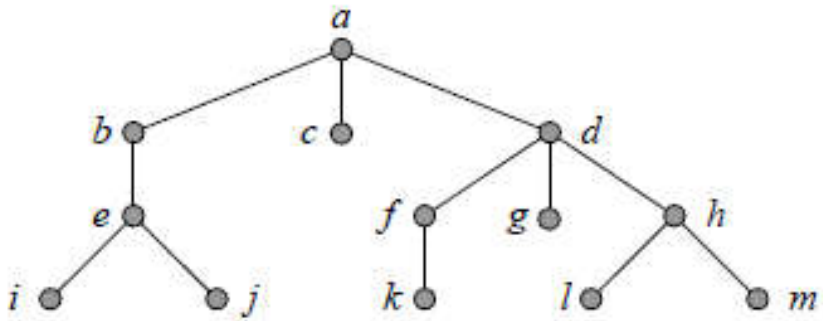


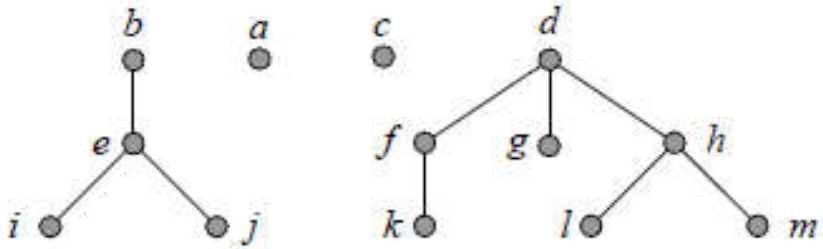
Determine the order in which a preorder traversal visits the vertices of the given ordered rooted tree.

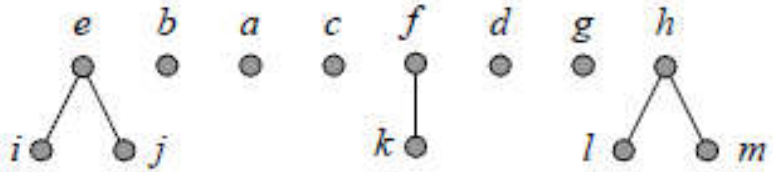
Inorder Traversal

```
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
```

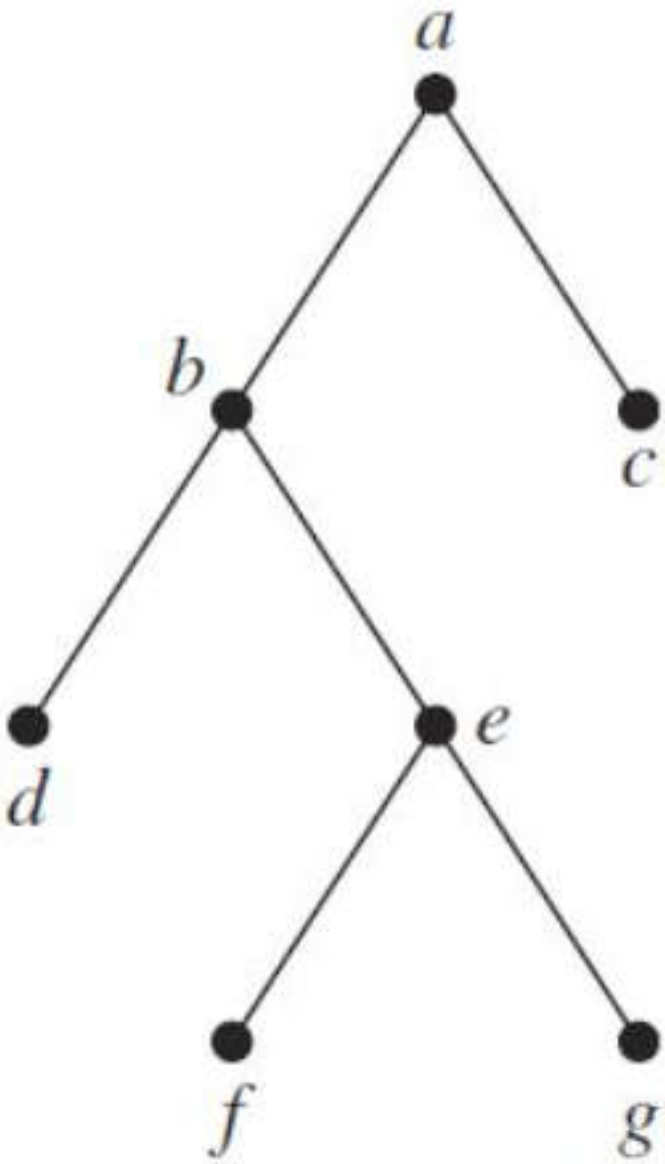
Example 9
Inorder Traversal







Example 10

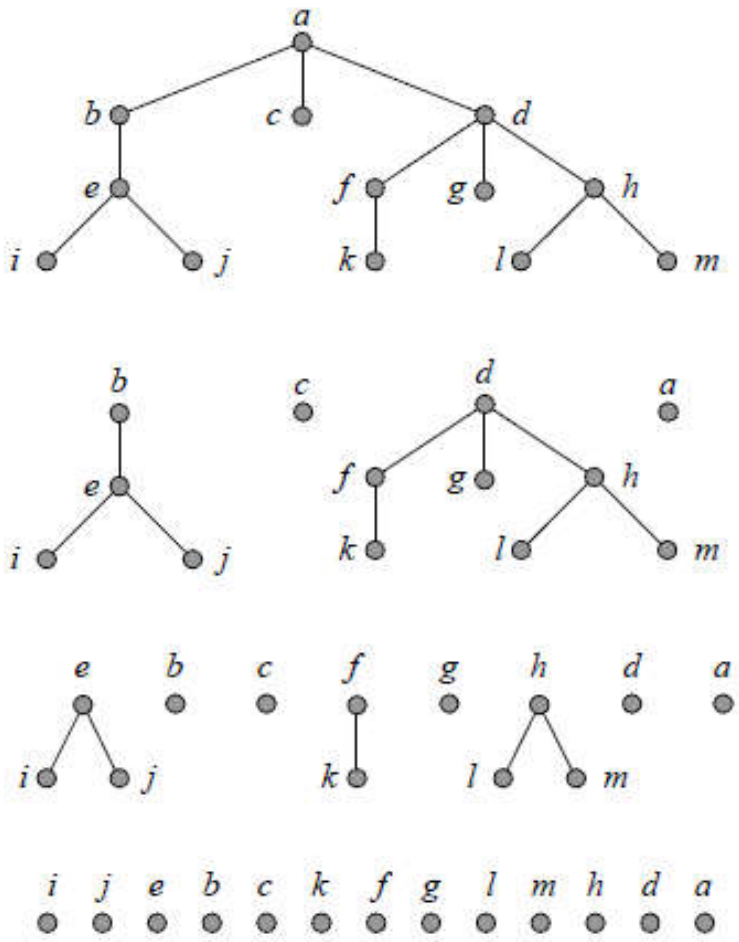


Determine the order in which a inorder traversal visits the vertices of the given ordered rooted tree.

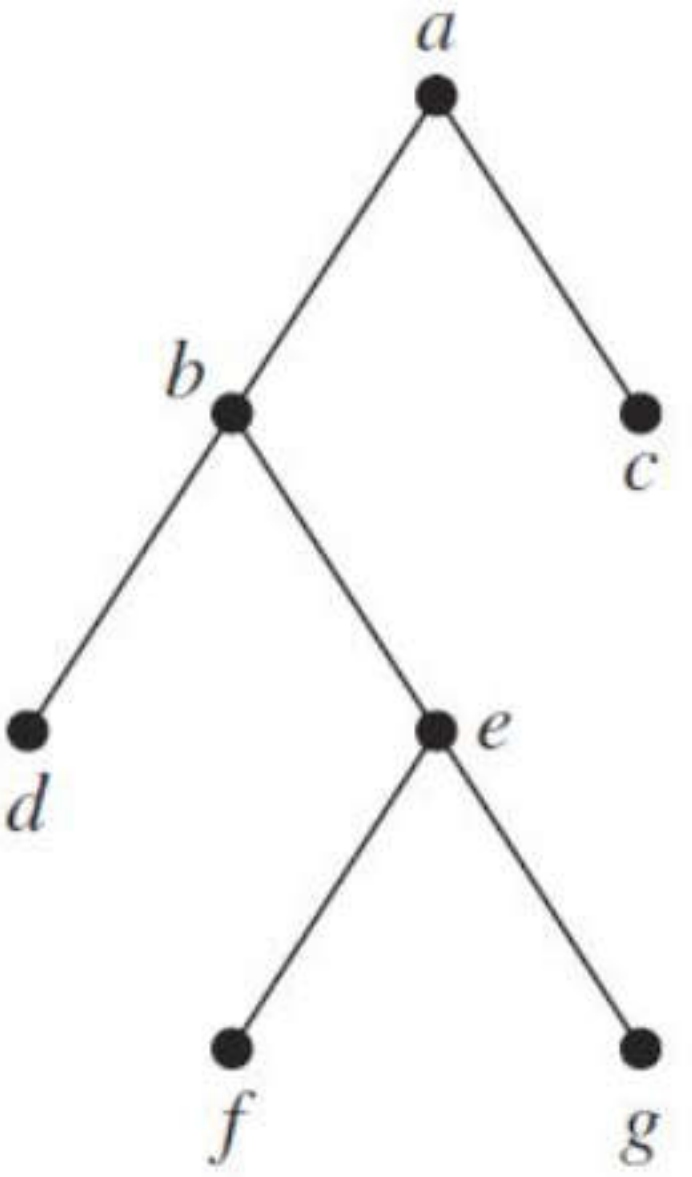
Postorder Traversal

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 11
Postorder Traversal



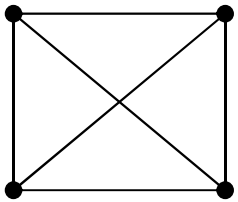
Example 12



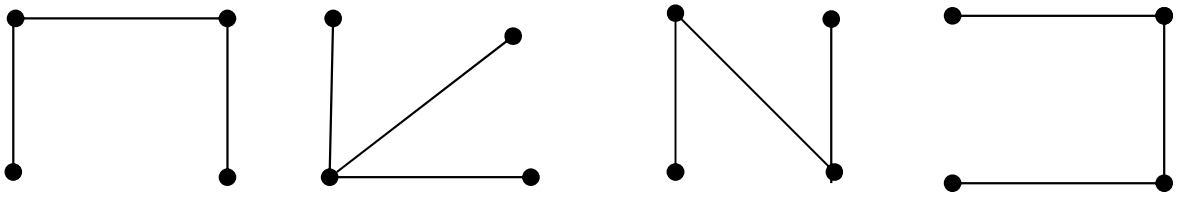
Determine the order in which a postorder traversal visits the vertices of the given ordered rooted tree.

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.



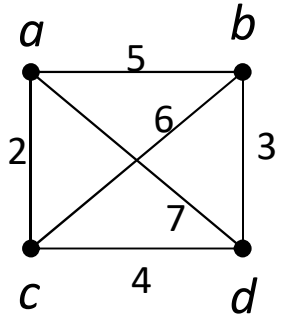
A connected undirected graph

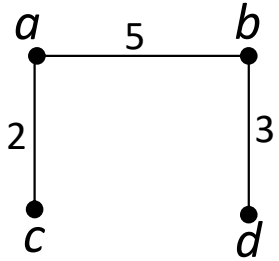


Four spanning trees of the graph

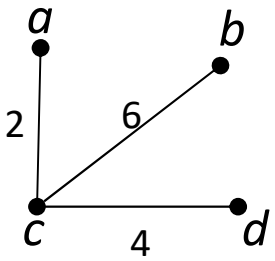
Minimum Spanning Tree (MST)

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

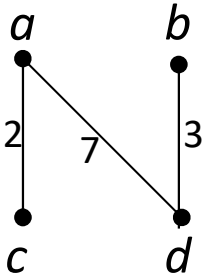




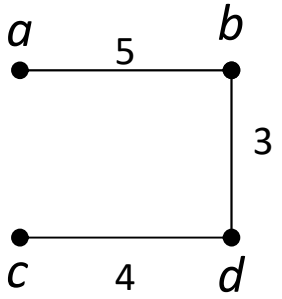
$T_1 = 10$



$T_2 = 12$

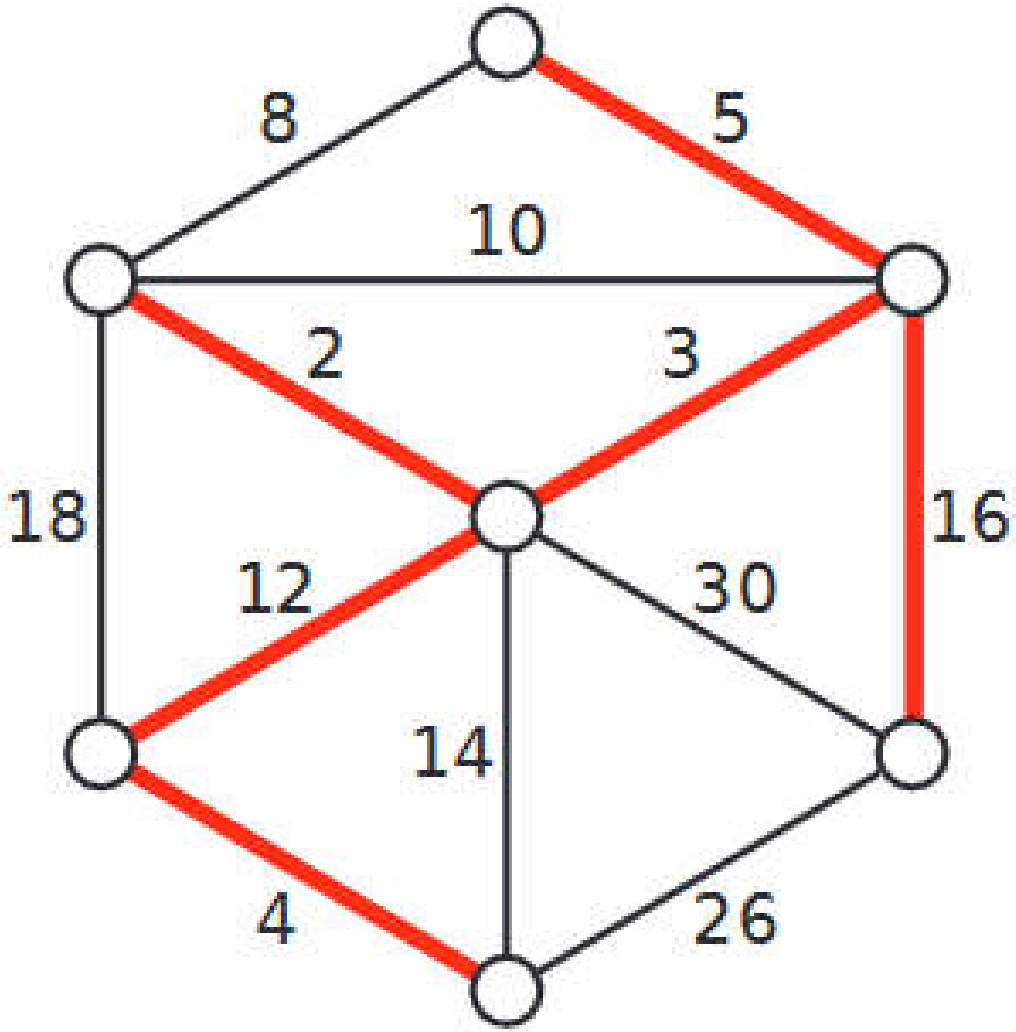


$T_3 = 12$



$T_4 = 12$

Minimum Spanning Tree (MST)



A weighted graph and its minimum spanning tree.

Muddy City Problem

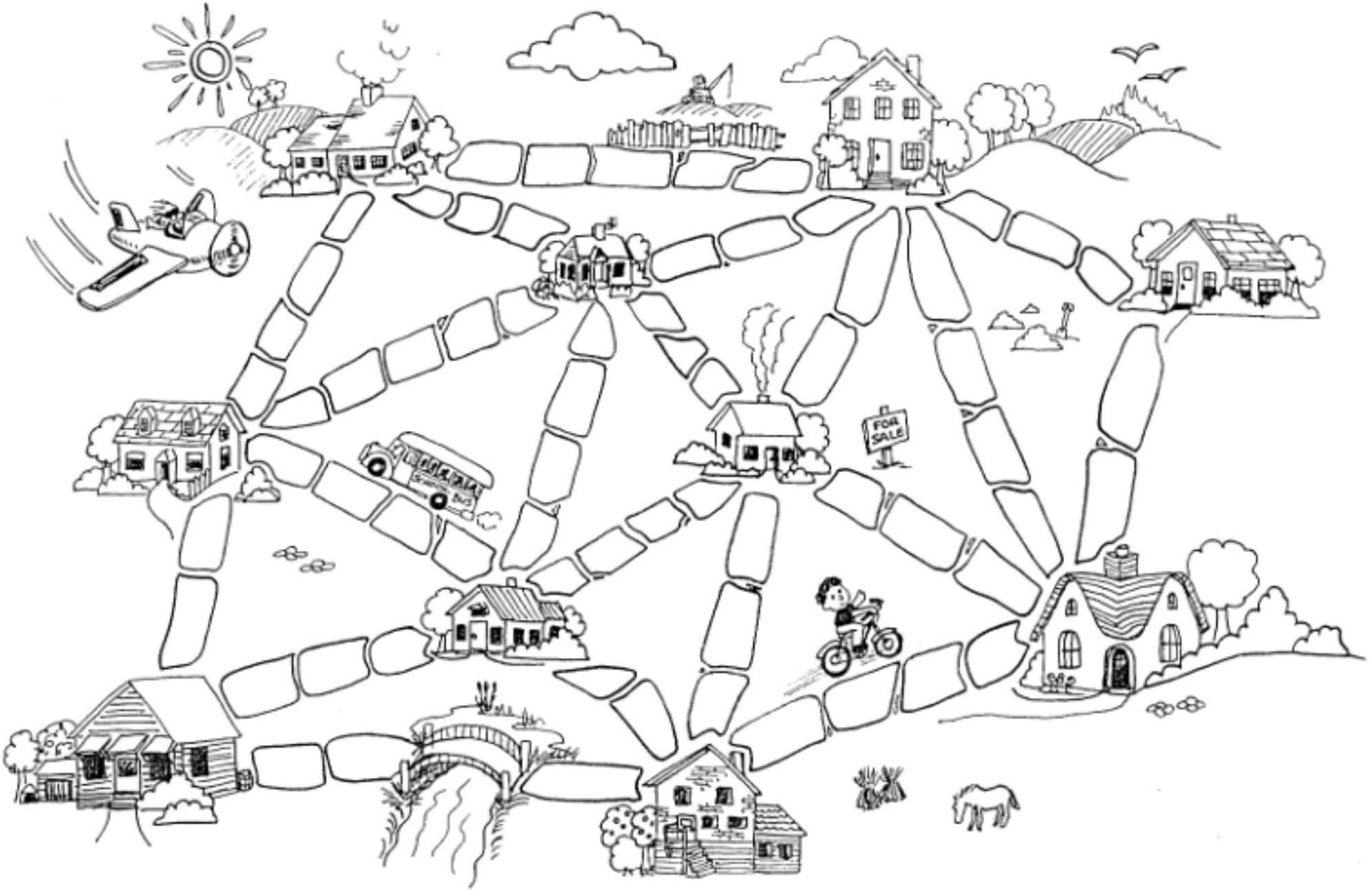
Once upon a time there was a city that had no roads. Getting around the city was particularly difficult after rainstorms because the ground became very muddy. Cars got stuck in the mud and people got their boots dirty. The mayor of the city decided that some of the streets must be paved, but didn't want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

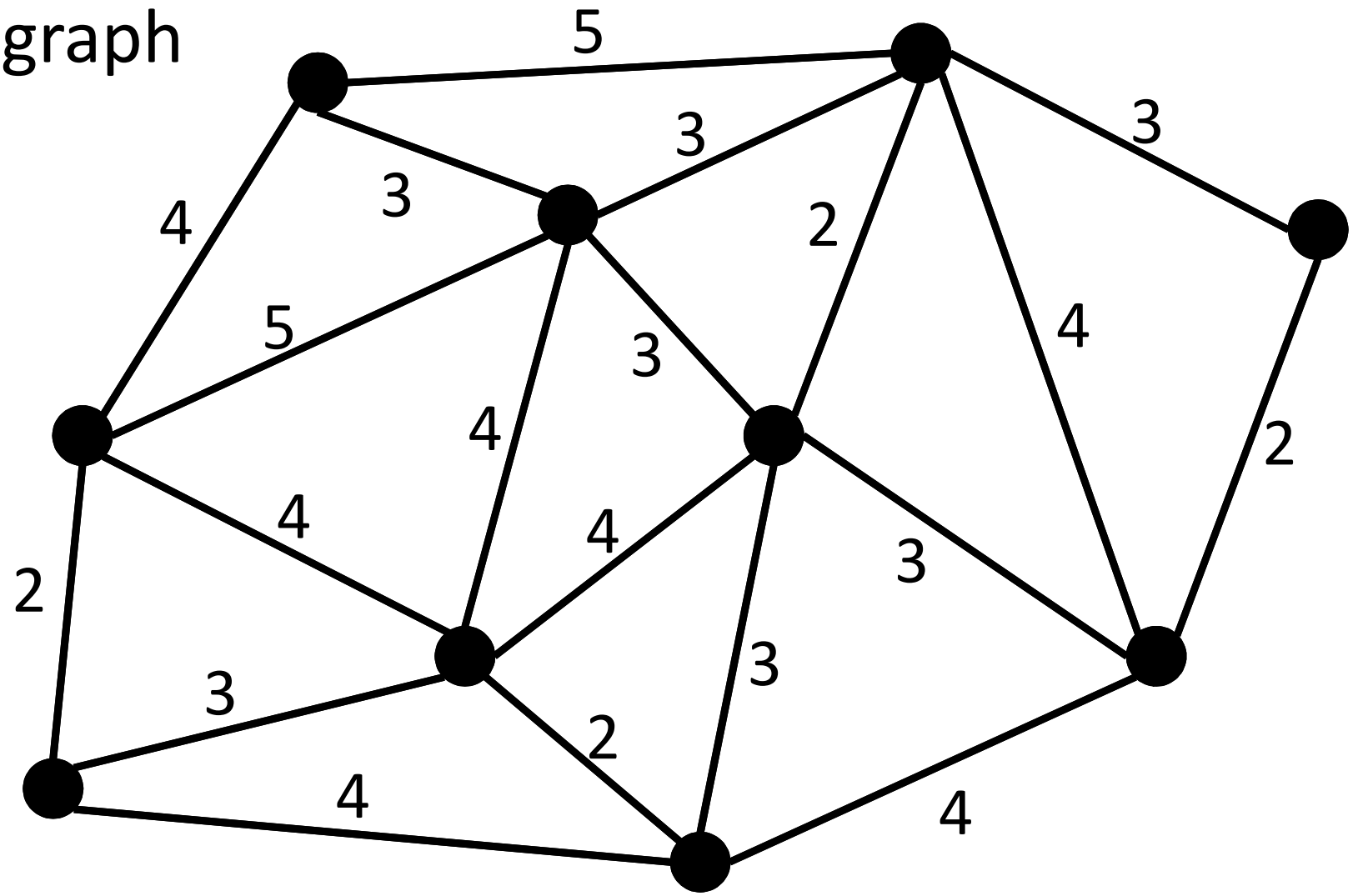
1. Enough streets must be paved so that it is possible for everyone to travel from their house to anyone else's house only along paved roads, and
2. The paving should cost as little as possible.

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

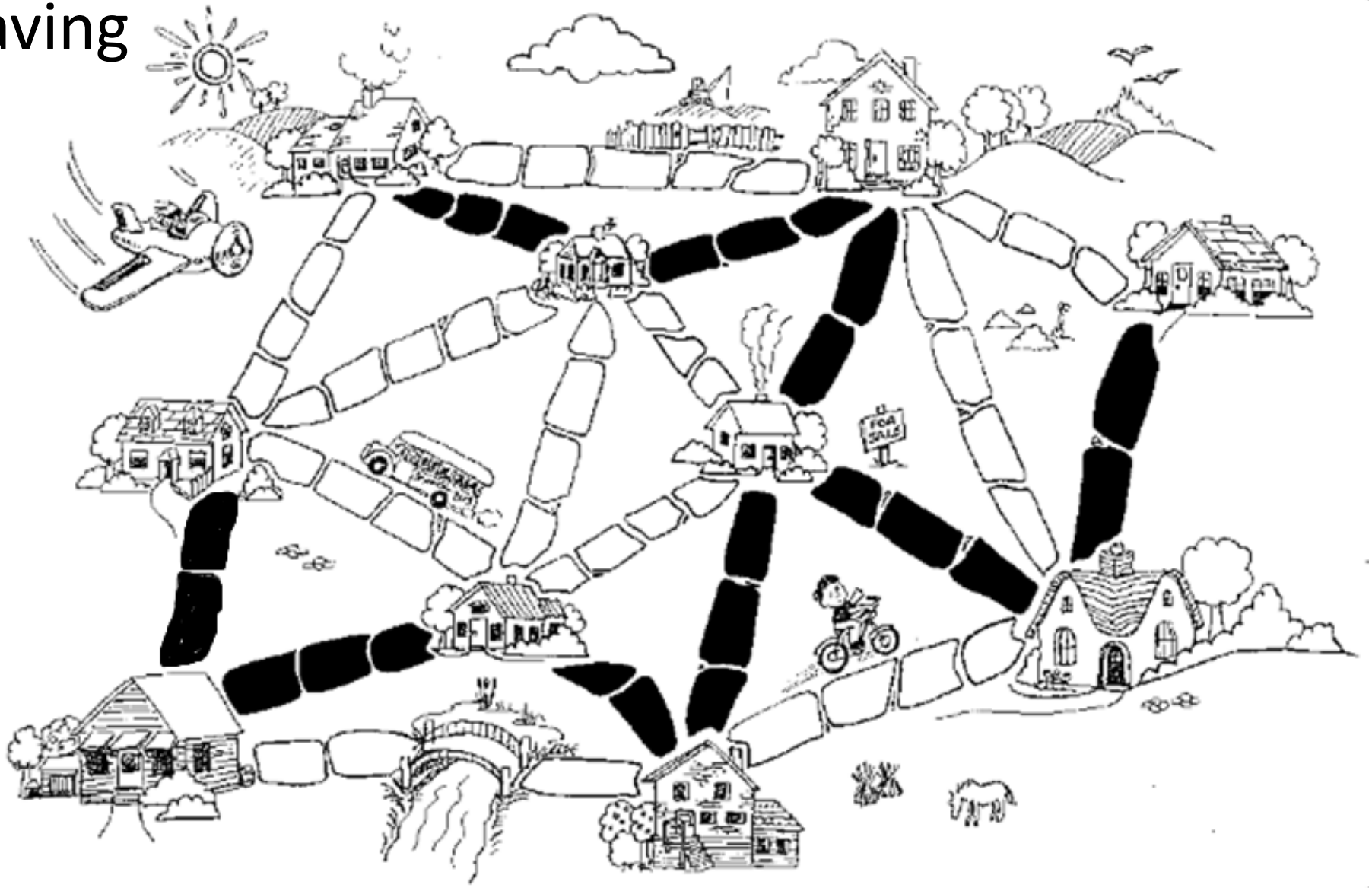
Muddy City Problem

The graph



Muddy City Problem

The paving



Application of MST: an example

- 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

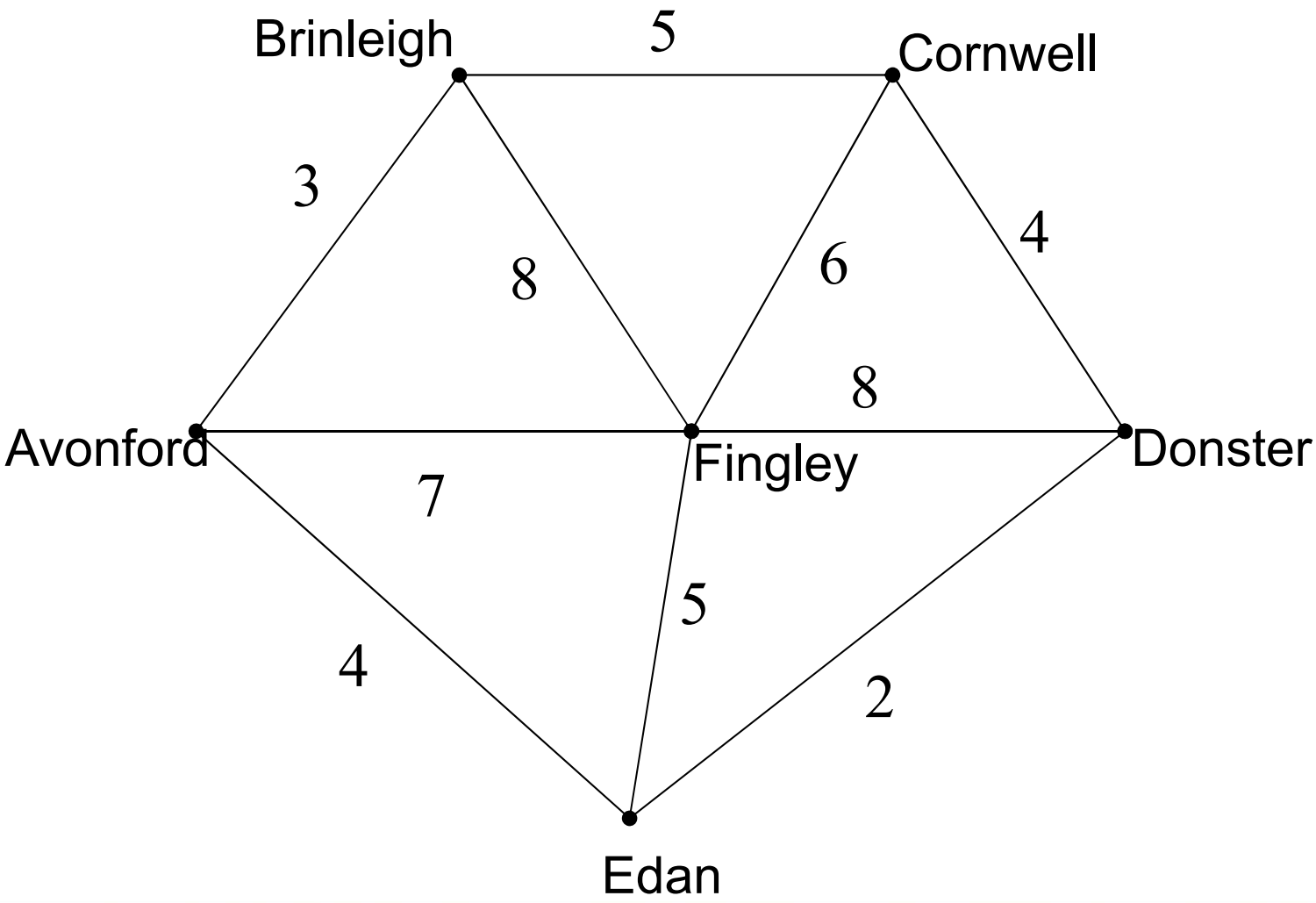
e:= any edge in *G* with smallest weight that does not form a simple circuit when added to *T*

T:= *T* with e added

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

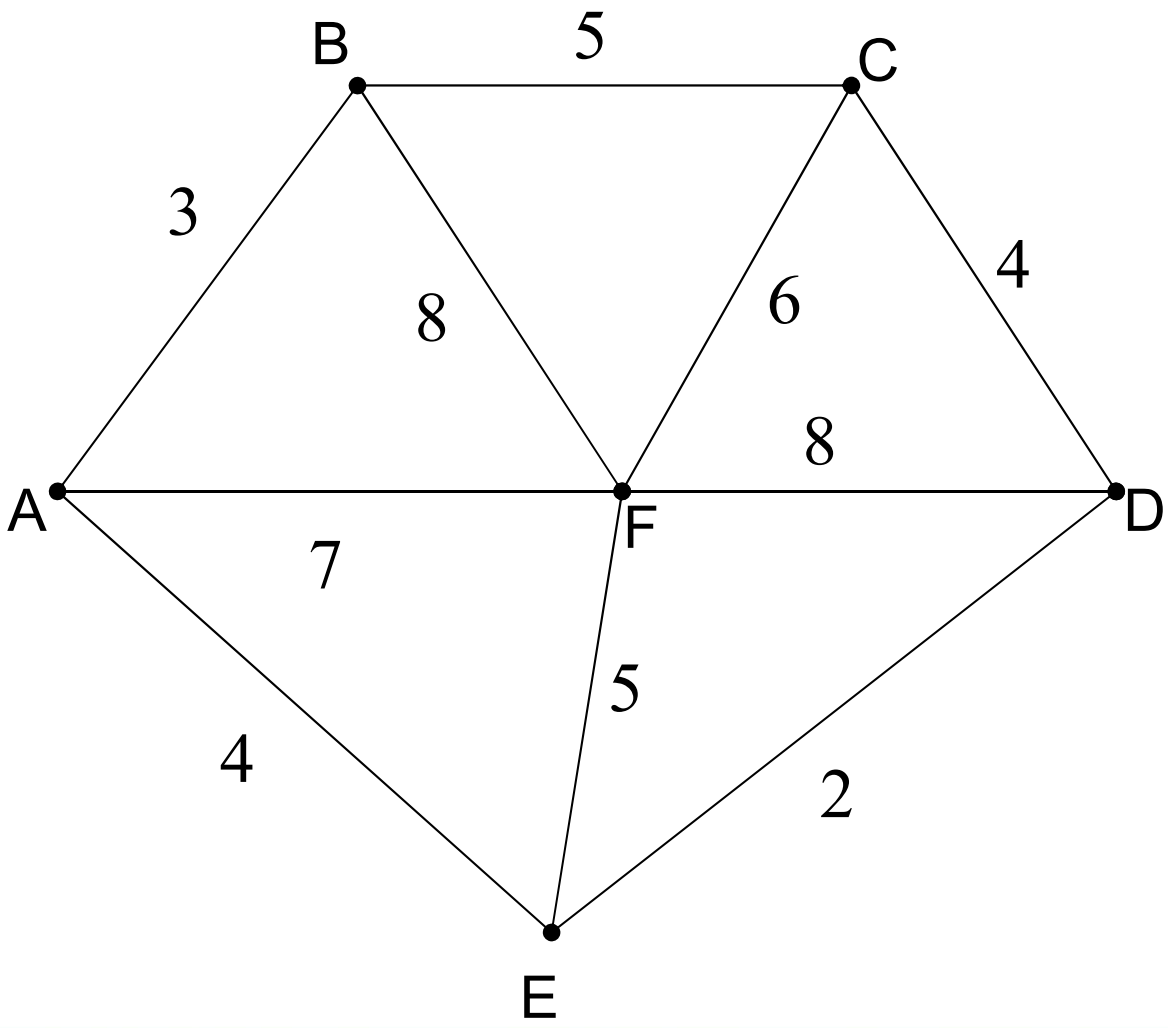
Example 13

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?



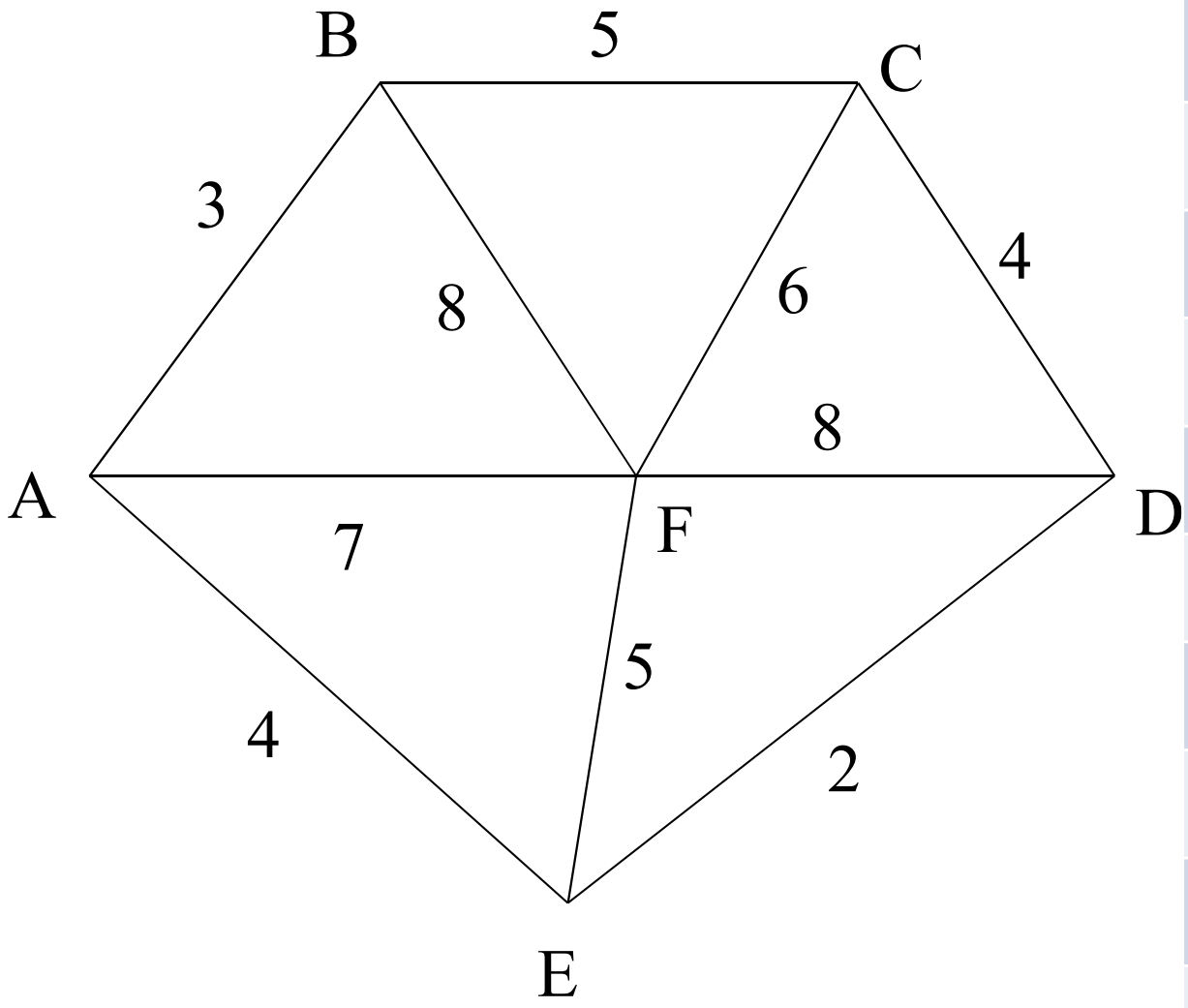
Example 13

We model the situation as a network, then the problem is to find the minimum connector for the network



Kruskal's Algorithm

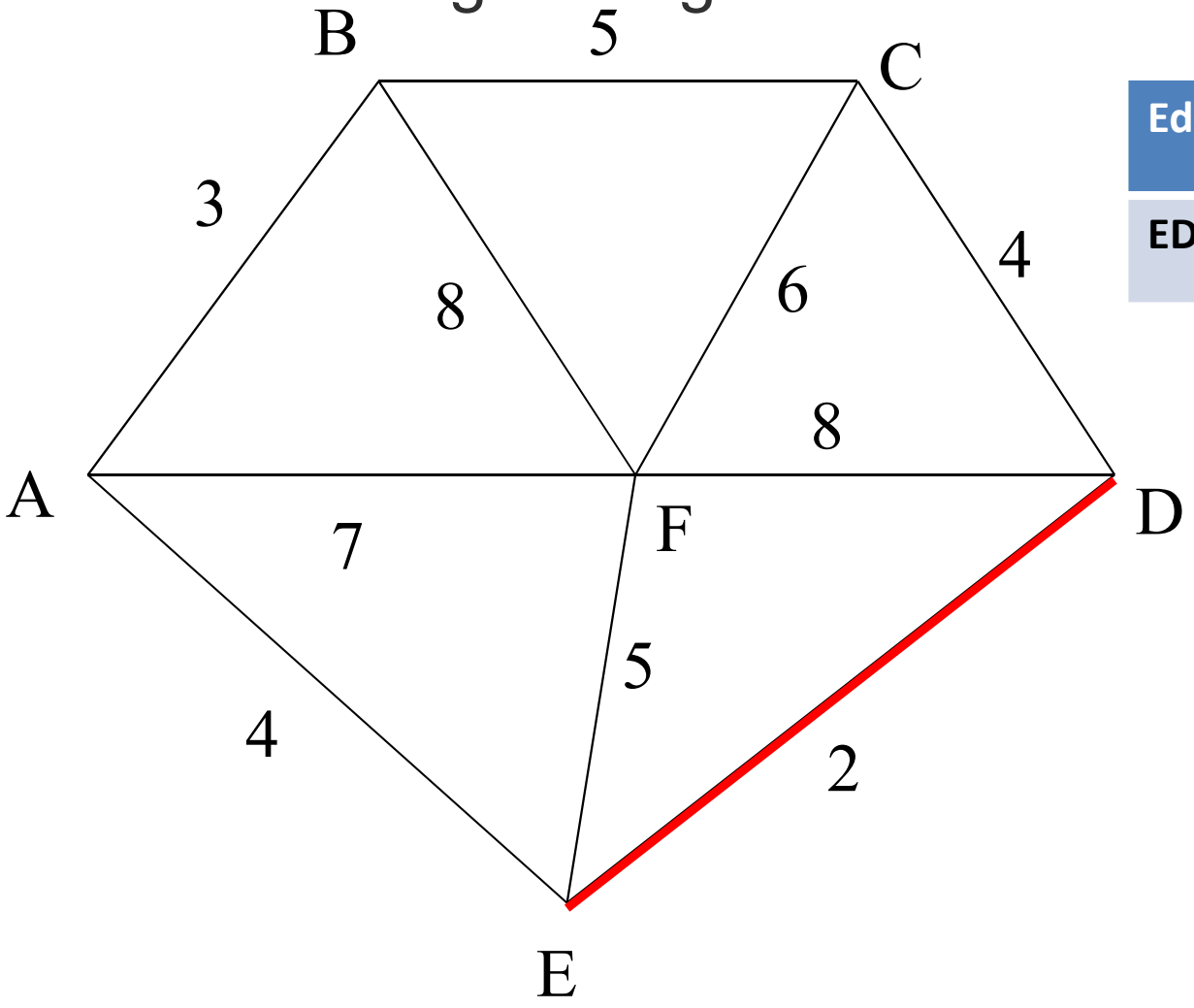
1)Sort edges – Arrange all edges in ascending order of weights



Edges	Weight	cycle (Yes/No)	Select (Yes/No)
ED	2		
AB	3		
CD	4		
AE	4		
BC	5		
EF	5		
CF	6		
AF	7		
BF	8		
DF	8		

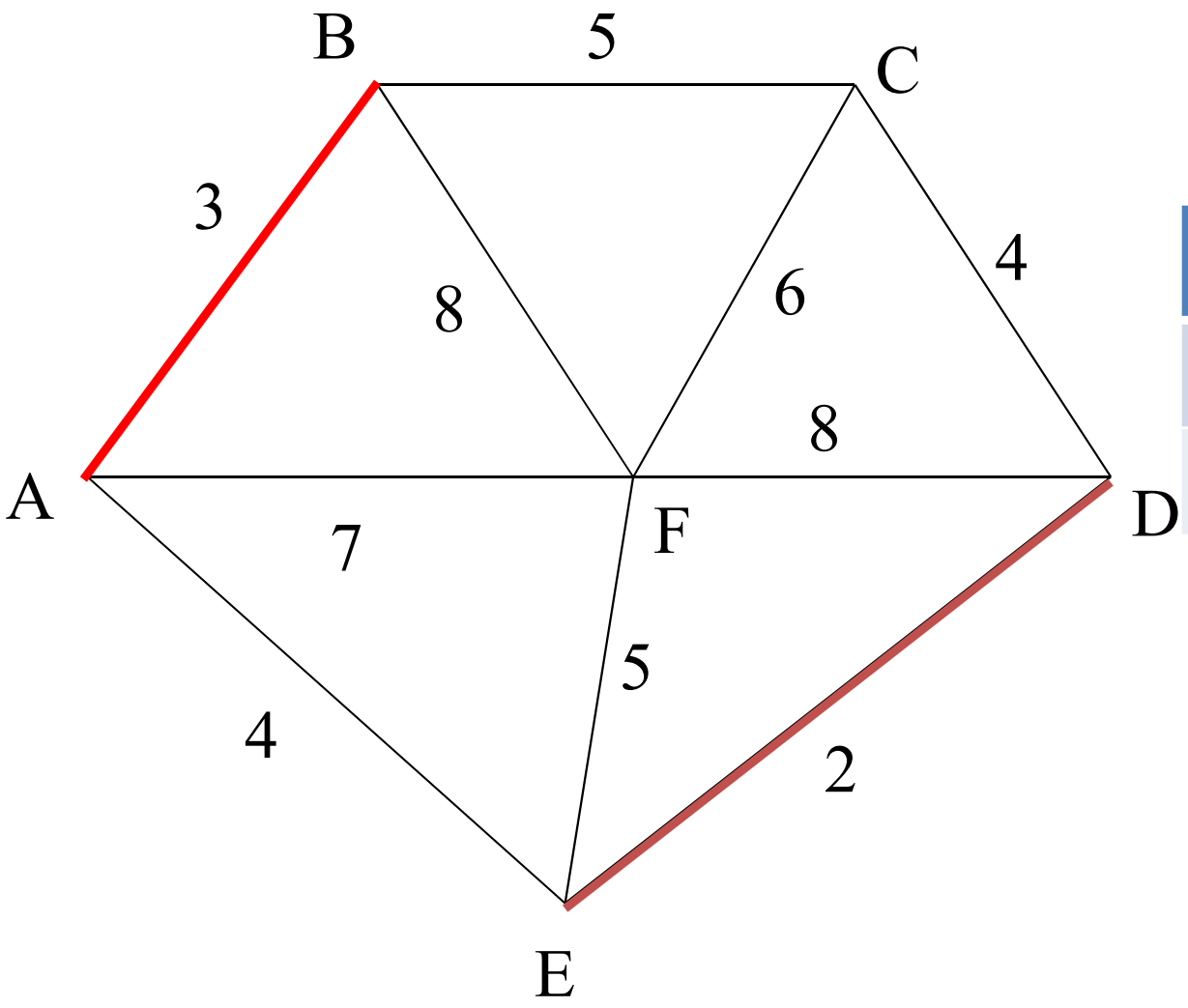
Kruskal's Algorithm

Select the edge with the smallest weight to connect the vertices of graph. If adding an edge creates a cycle, then discard that edge and go for the next least weight edge.



Edges	Weight	cycle (Yes/No)	Select (Yes/No)
ED	2	No	Yes

Kruskal's Algorithm

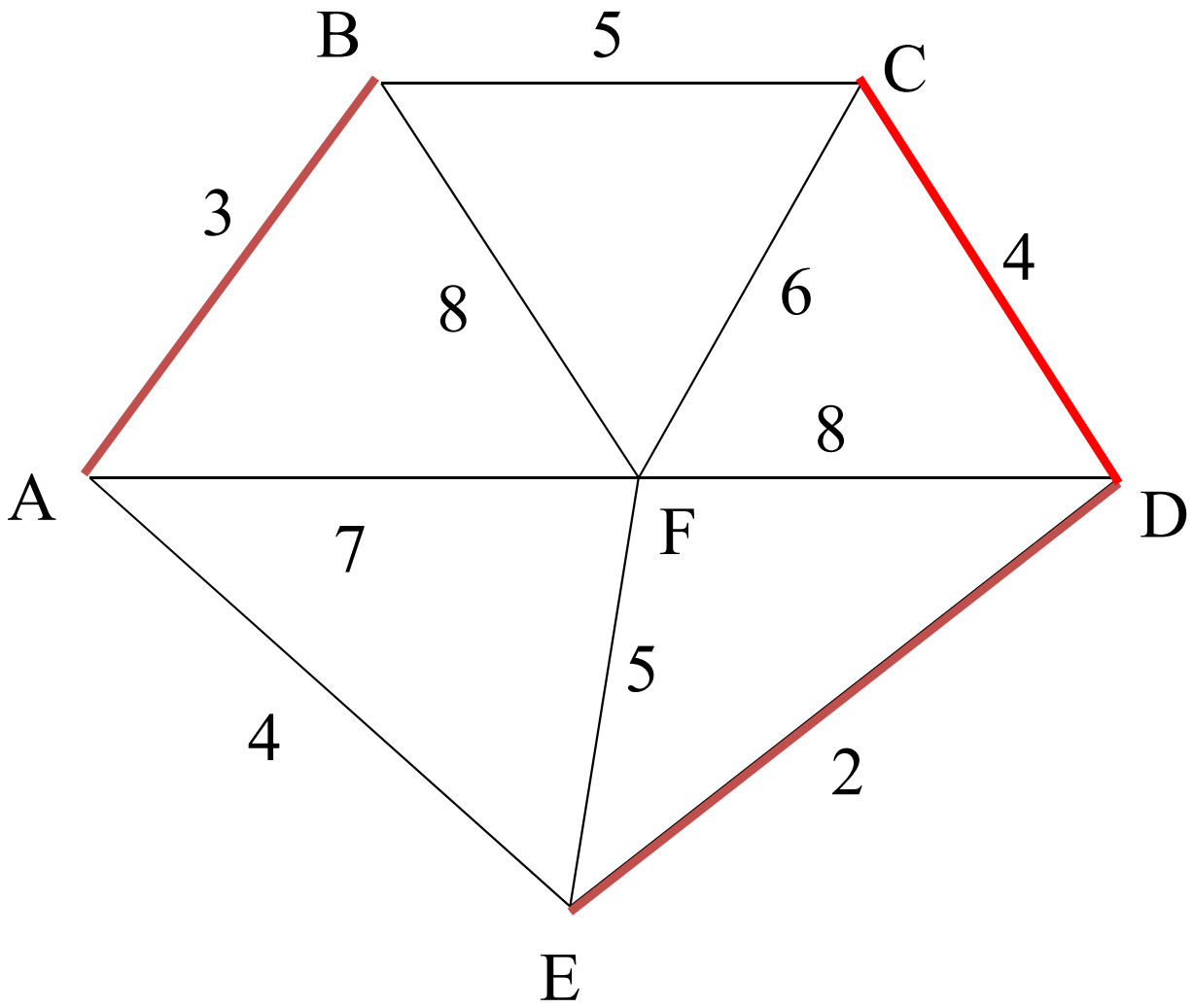


Select the next shortest edge which does not create a cycle

Edges	Weight	cycle (Yes/No)	Select (Yes/No)
ED	2	No	Yes
AB	3	No	Yes

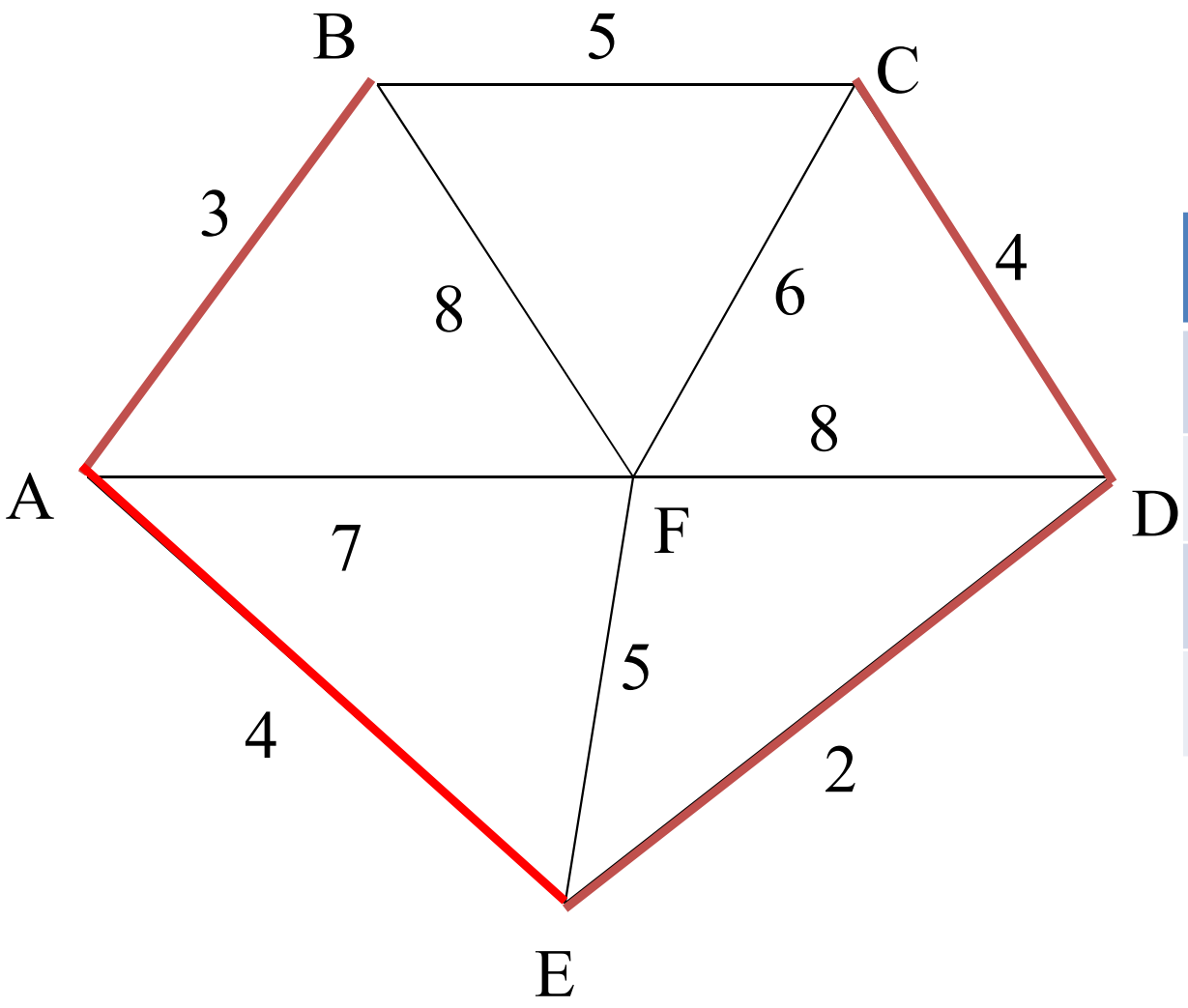
Kruskal's Algorithm

Select the next shortest edge which does not create a cycle



Edges	Weight	cycle (Yes/No)	Select (Yes/No)
ED	2	No	Yes
AB	3	No	Yes
CD	4	No	Yes

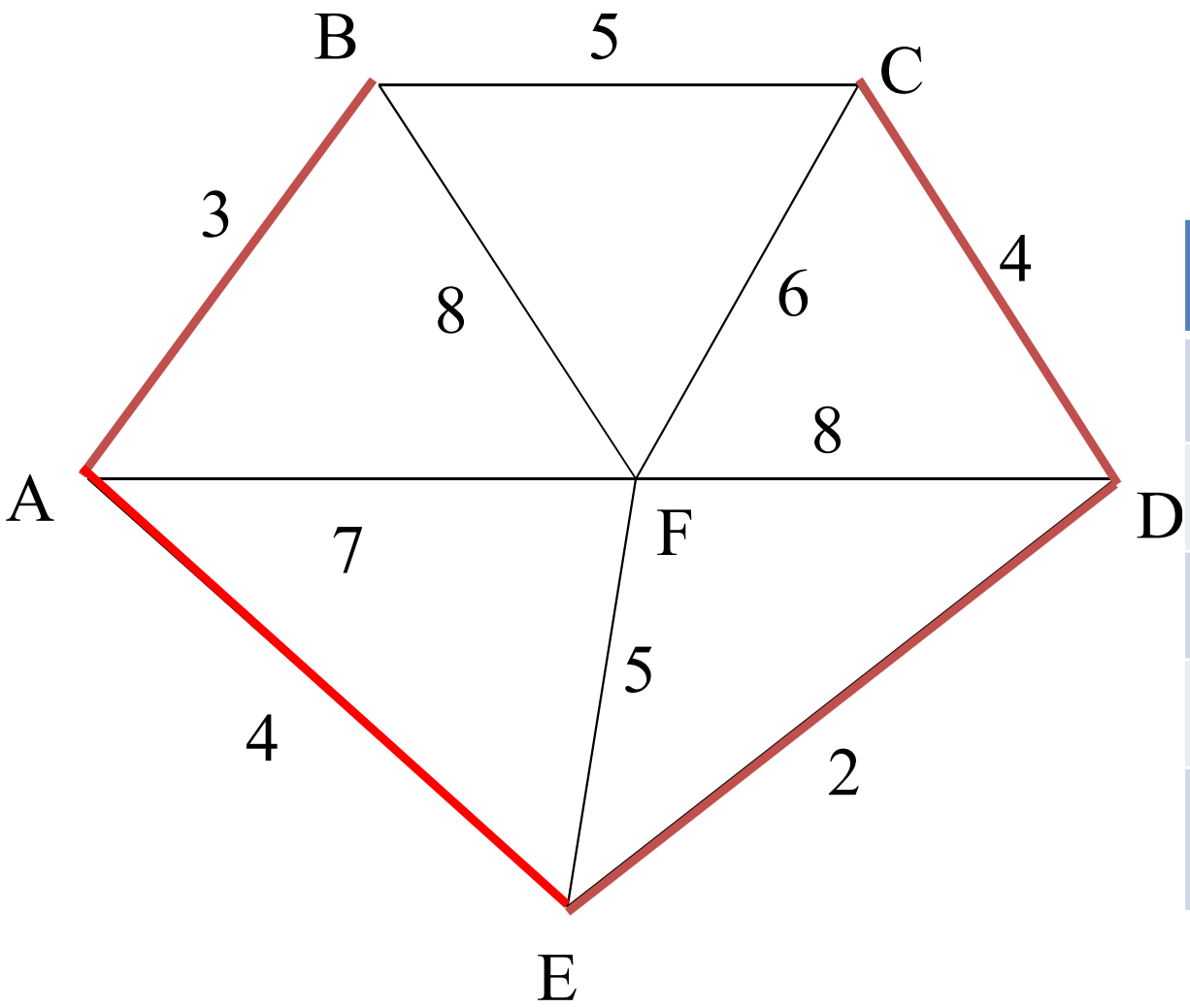
Kruskal's Algorithm



Select the next shortest edge which does not create a cycle

Edges	Weight	cycle (Yes/No)	Select (Yes/No)
ED	2	No	Yes
AB	3	No	Yes
CD	4	No	Yes
AE	4	No	Yes

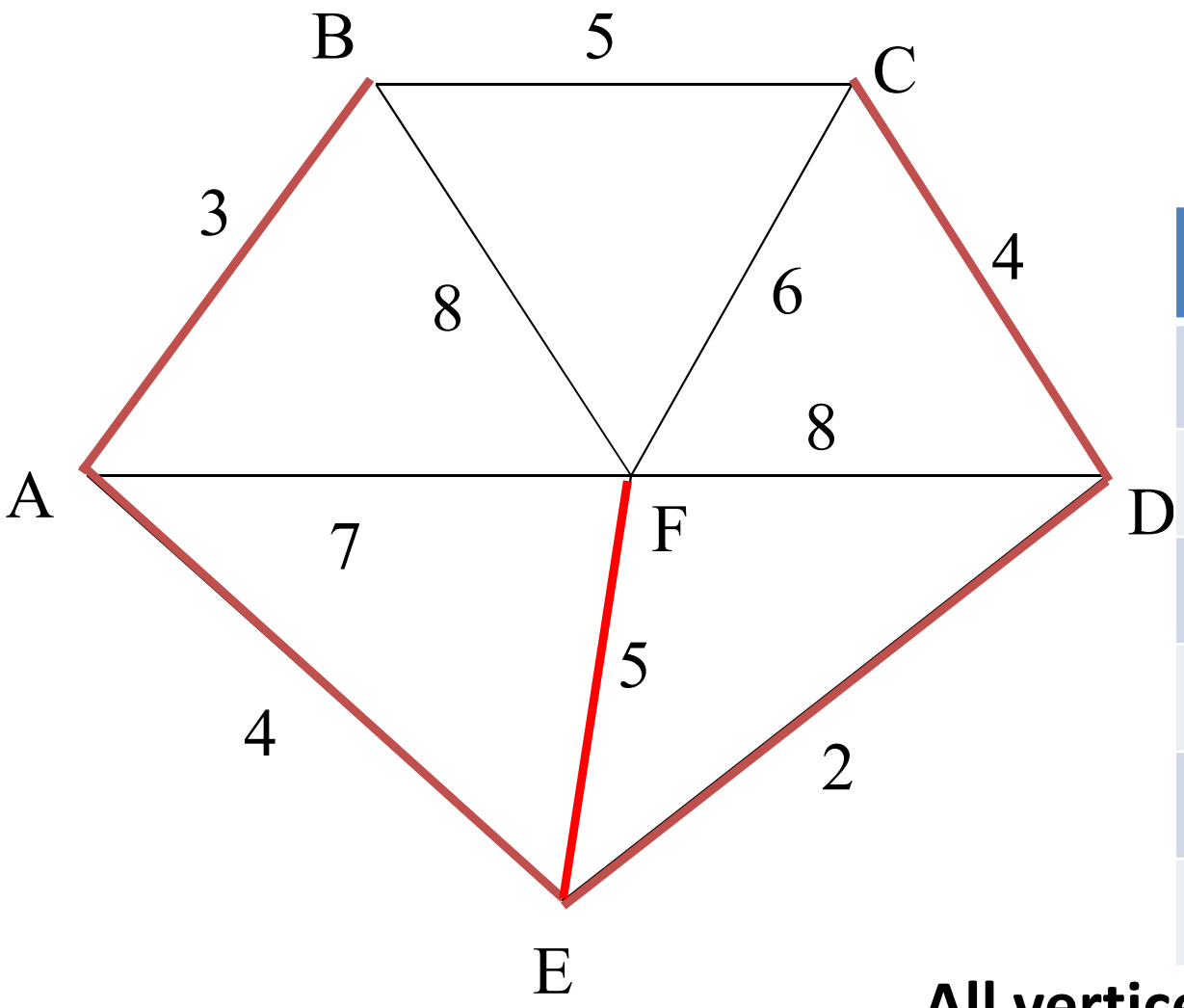
Kruskal's Algorithm



Select the next shortest edge which does not create a cycle

Edges	Weight	cycle (Yes/No)	Select (Yes/No)
ED	2	No	Yes
AB	3	No	Yes
CD	4	No	Yes
AE	4	No	Yes
BC	5	Yes	No

Kruskal's Algorithm



Select the next shortest edge which does not create a cycle

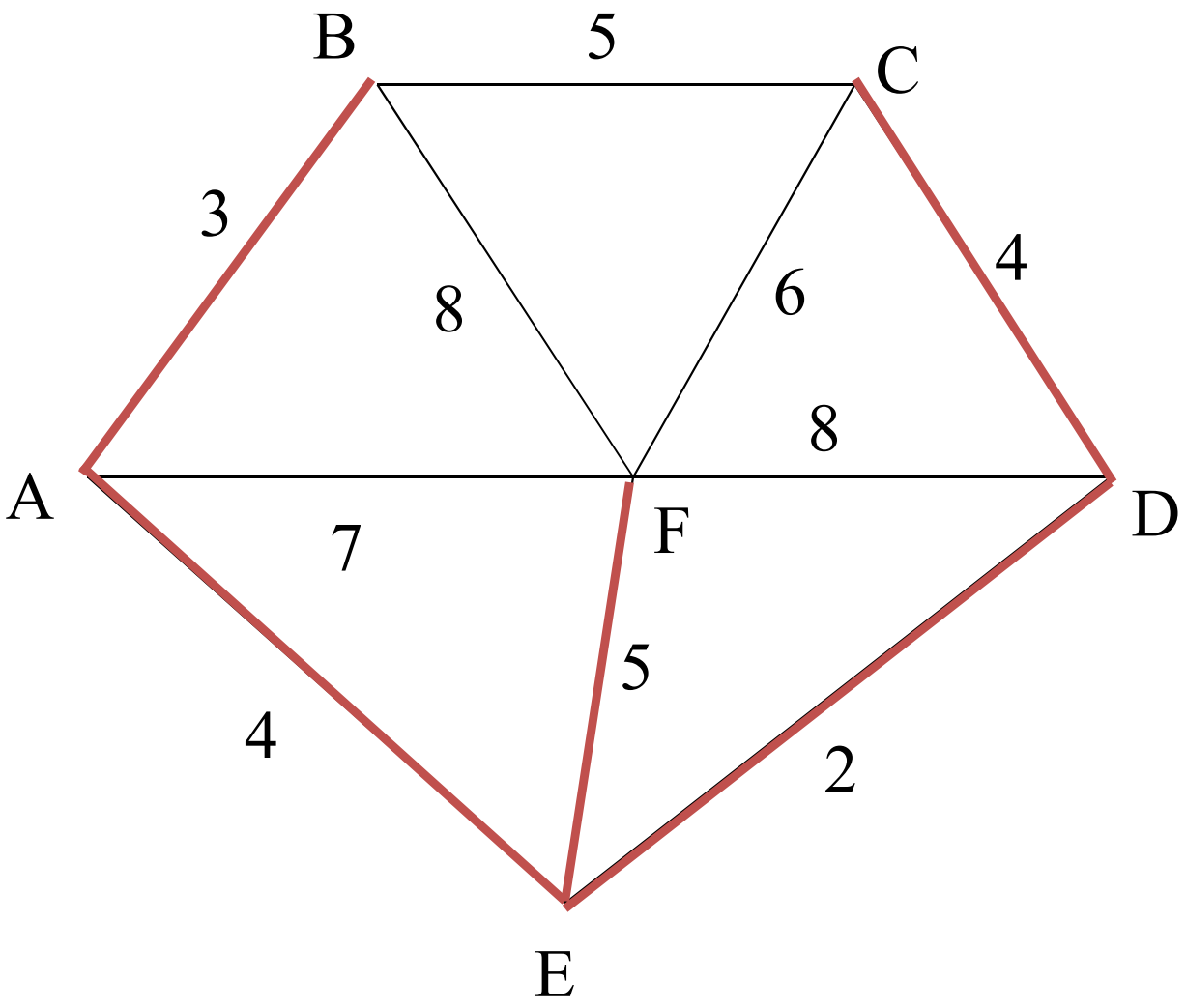
Edges	Weight	cycle (Yes/No)	Select (Yes/No)
ED	2	No	Yes
AB	3	No	Yes
CD	4	No	Yes
AE	4	No	Yes
BC	5	Yes	No
EF	5	No	Yes

All vertices have been connected.

Kruskal's Algorithm

Edges	Wight	cycle (Yes/No)	Select (Yes/No)
ED	2	No	Yes
AB	3	No	Yes
CD	4	No	Yes
AE	4	No	Yes
BC	5	Yes	No
EF	5	No	Yes
CF	6	Yes	No
AF	7	Yes	No
BF	8	Yes	No
CF	8	Yes	No

Kruskal's Algorithm



The solution is

ED 2
AB 3
CD 4
AE 4
EF 5

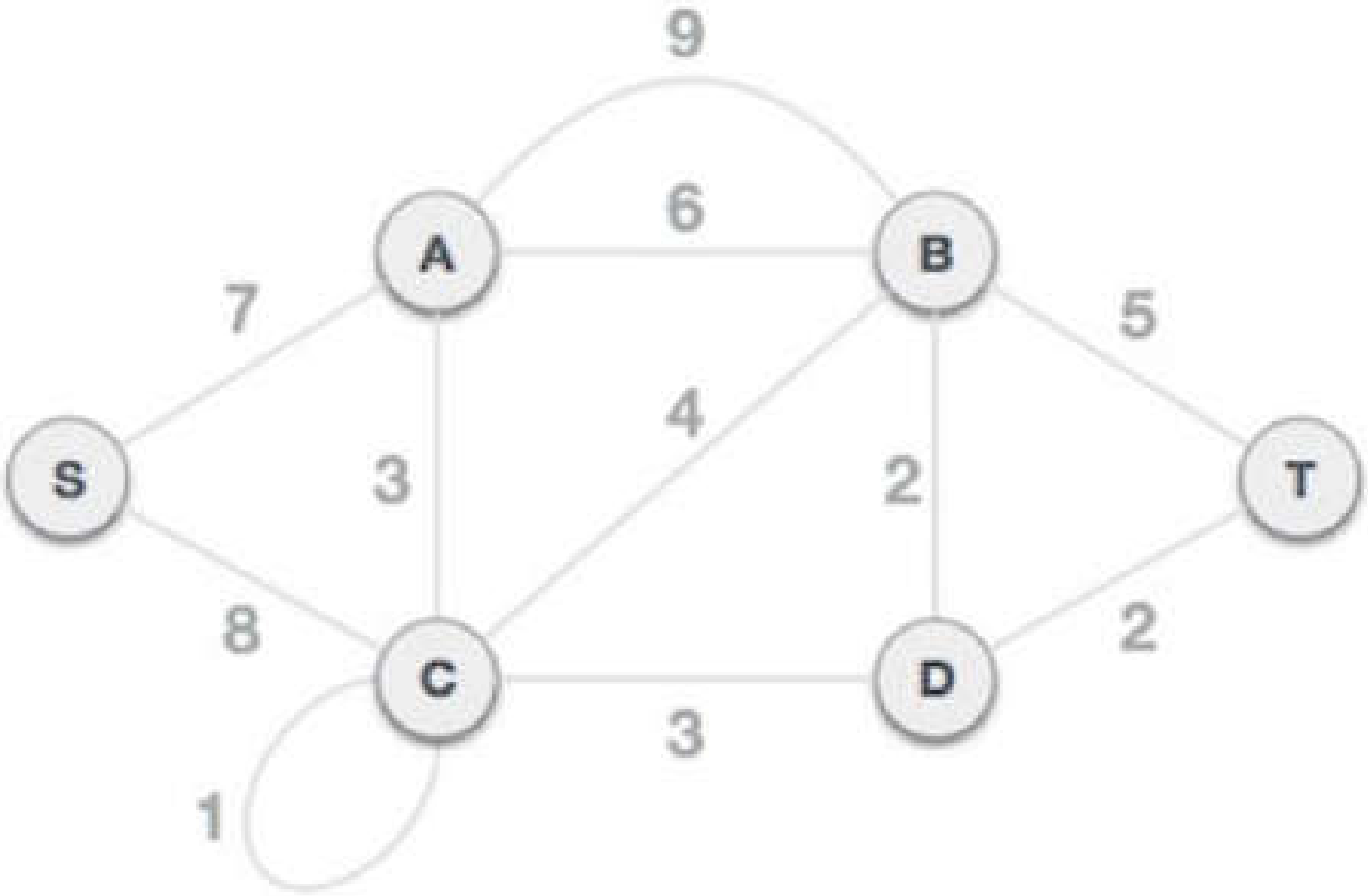
Total weight of tree:
18

Kruskal's Algorithm

Important notes:

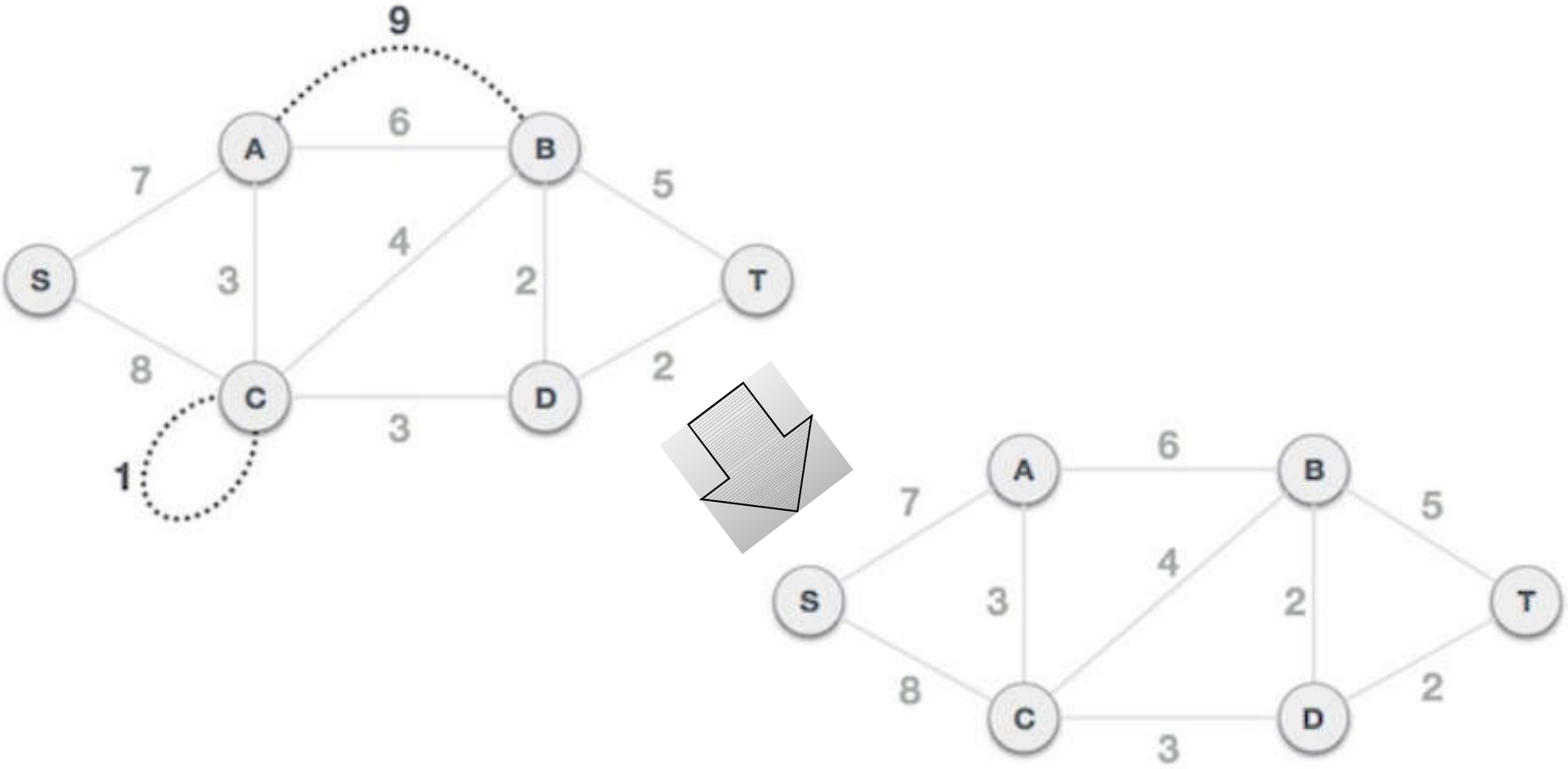
- The graph given should be a tree
  - Remove all loops (if any)
  - Remove all parallel edges
    - keep the one which has the least weight associated and remove all others

Example 14



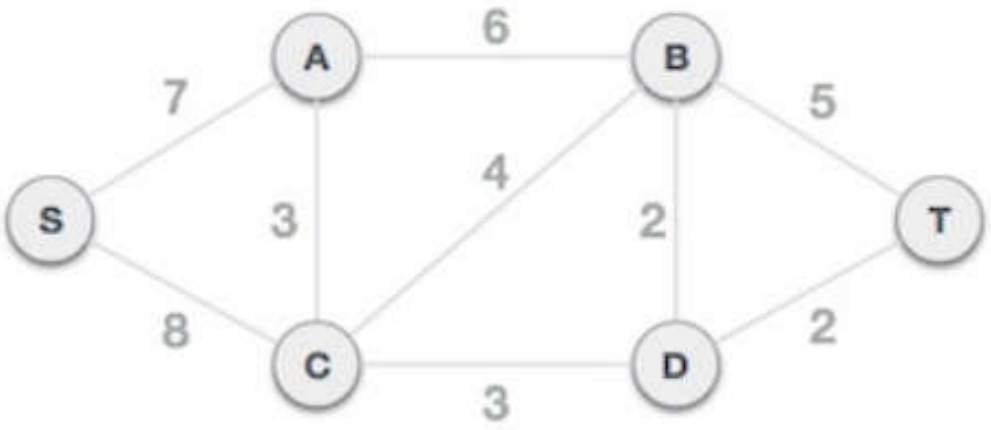
Example 14

- Remove all loops and parallel edges



Example 14

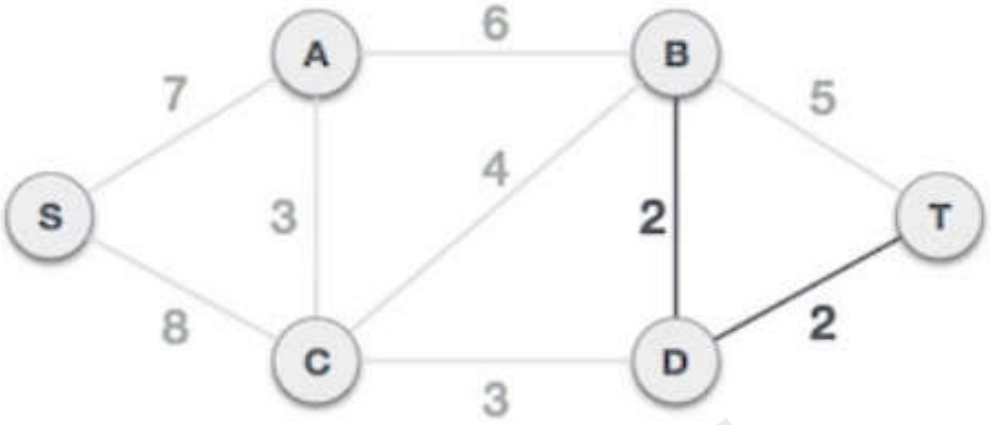
- Arrange all edges in ascending order of weight



Edges	Weight	cycle (Yes/No)	Select (Yes/No)
BD	2		
DT	2		
AC	3		
CD	3		
CB	4		
BT	5		
AB	6		
SA	7		
SC	8		

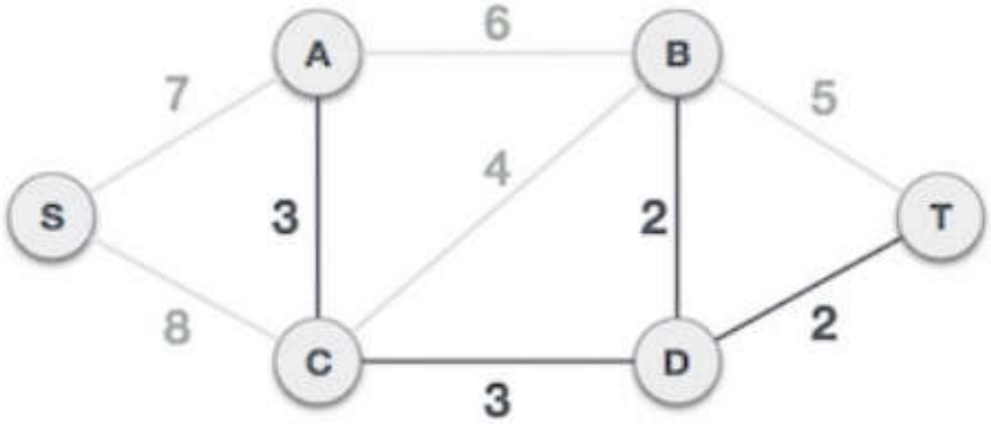
Example 14

- Add the edge which has the least weightage



Edges	Weight	cycle (Yes/No)	Select (Yes/No)
BD	2	No	Yes
DT	2	No	Yes

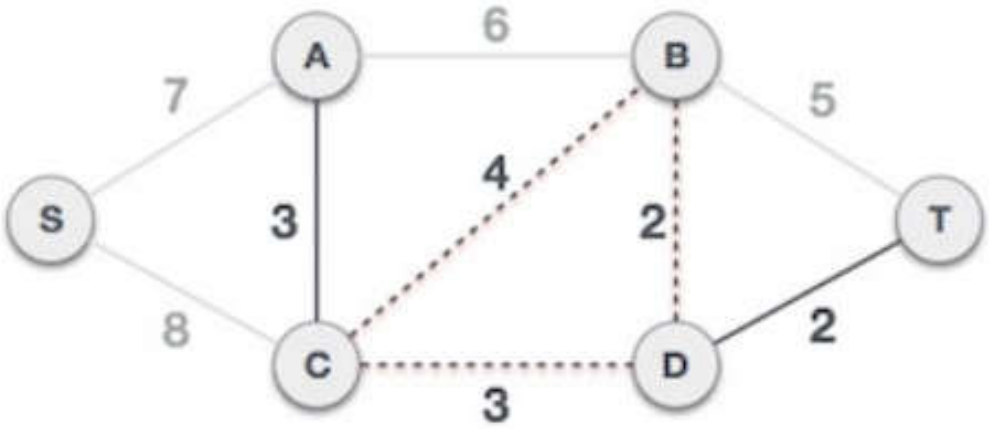
Next weight is 3, and associated edges are AC and CD



Edges	Weight	cycle (Yes/No)	Select (Yes/No)
BD	2	No	Yes
DT	2	No	Yes
AC	3	No	Yes
CD	3	No	Yes

Example 14

The next weight is 4, and adding it would create a circuit in the graph. Thus, it is discarded.

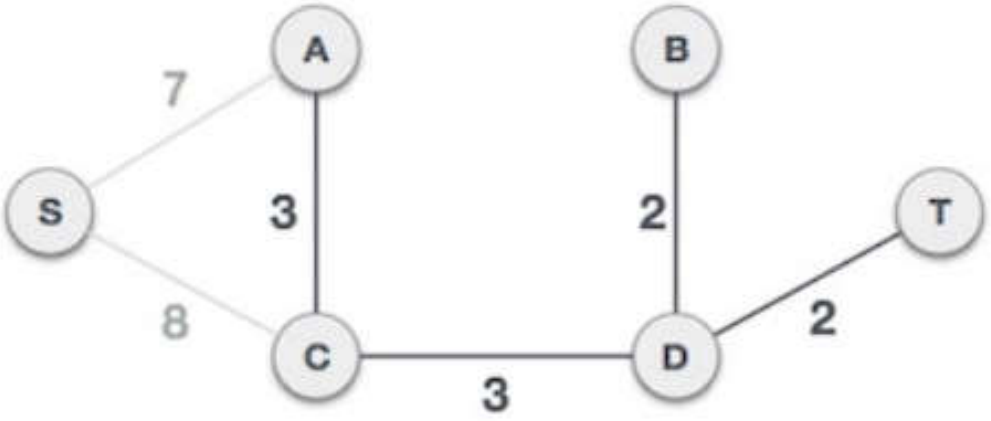


Edges	Weight	cycle (Yes/No)	Select (Yes/No)
BD	2	No	Yes
DT	2	No	Yes
AC	3	No	Yes
CD	3	No	Yes
CB	4	Yes	No
BT	5	Yes	No
AB	6	Yes	No

Edges with weight 5 and 6 also create circuits and are therefore discarded.

Example 14

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



Edges	Wight	cycle (Yes/No)	Select (Yes/No)
BD	2	No	Yes
DT	2	No	Yes
AC	3	No	Yes
CD	3	No	Yes
CB	4	Yes	No
BT	5	Yes	No
AB	6	Yes	No
SA	7	No	Yes
SC	8	Yes	No

Example 14

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