UTM

UNIVERSITI TEKNOLOGI MALAYSIA

INSPIRING CREATIVE AND INNOVATIVE MINDS

www.utm.my

# Chapter 4

# Graph Theory

## (Part 1)

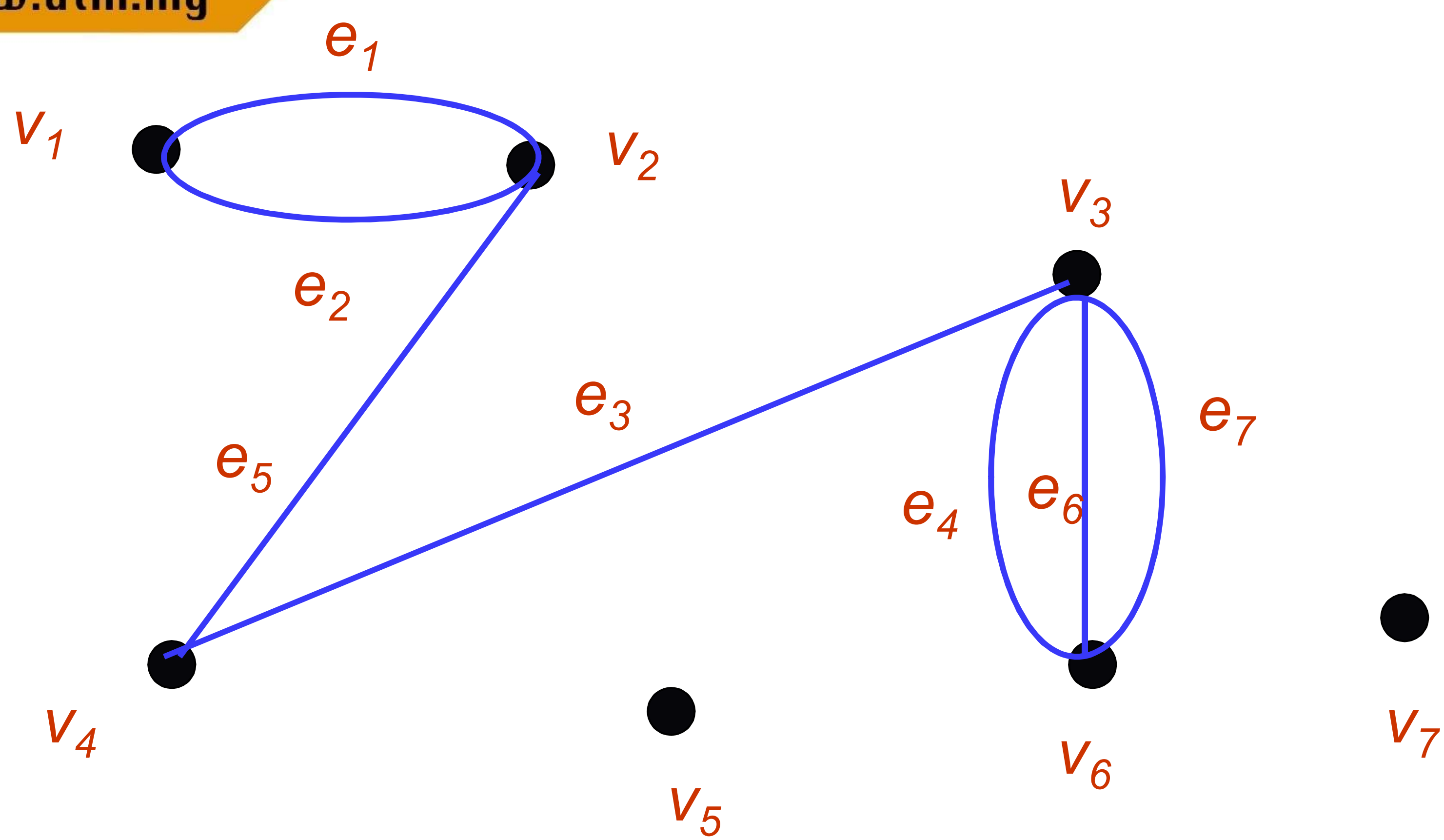
# Definition

- A graph $G$ is a triple $(V, E, f)$, where
  - $V$ is a finite nonempty set, called the set of vertices
  - $E$ is a finite set (may be empty), called the set of edges
  - $f$ is a function, called an incidence function, that assign to each edge, $e \in E$, a one-element subset $\{v\}$ or a two-element subset $\{v, w\}$, where $v$ and $w$ are vertices.
- We can write $G$ as $(V, E, f)$ or $(V, E)$ or simply as $G$.

# example

- Let,
  - $V=\{v_1, v_2, v_3, v_4, v_5, v_6, v_7\}$
  - $E=\{e_1, e_2, e_3, e_4, e_5, e_6, e_7\}$
- and $f$ be defined by
  - $f(e_1)= f(e_2)=\{v_1, v_2\}$
  - $f(e_3)= \{v_4, v_3\}$
  - $f(e_4)= f(e_6)= f(e_7)=\{v_6, v_3\}$
  - $f(e_5)= \{v_2, v_4\}$
- Then $G=(V,E,f)$ is a graph

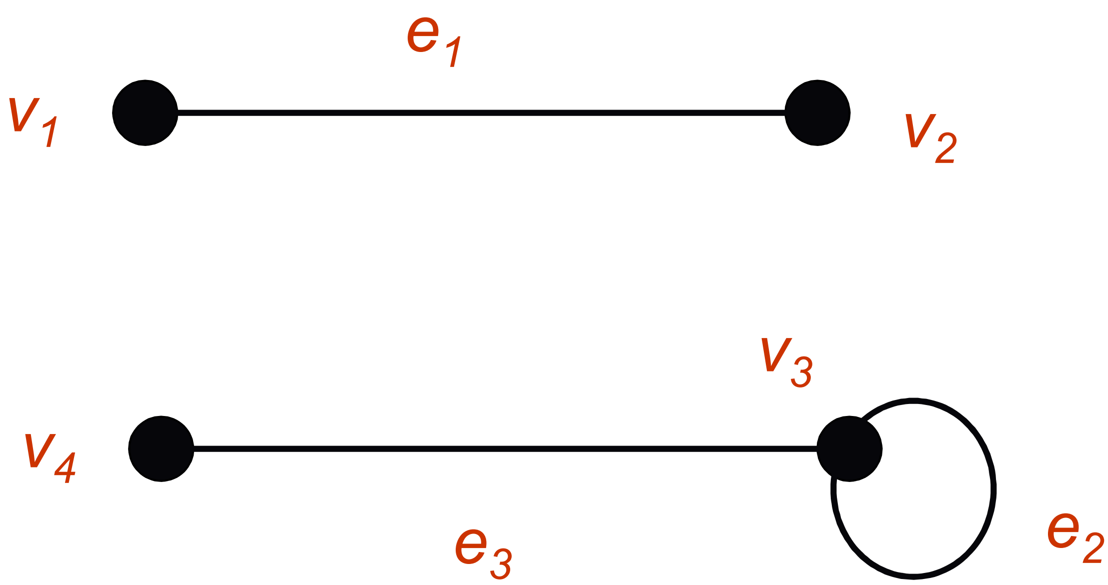
# example



# example

- Let $V=\{v_1, v_2, v_3, v_4\}$, $E=\{e_1, e_2, e_3\}$ and
  - $f(e_1)=\{v_1, v_2\}$
  - $f(e_2)=\{v_3, v_3\}$
  - $f(e_3)=\{v_3, v_4\}$
- Then $G=(V,E,f)$ is a graph

# example




prepared by Razana Alwee

# Characteristics of Graph

# Adjacent Vertices

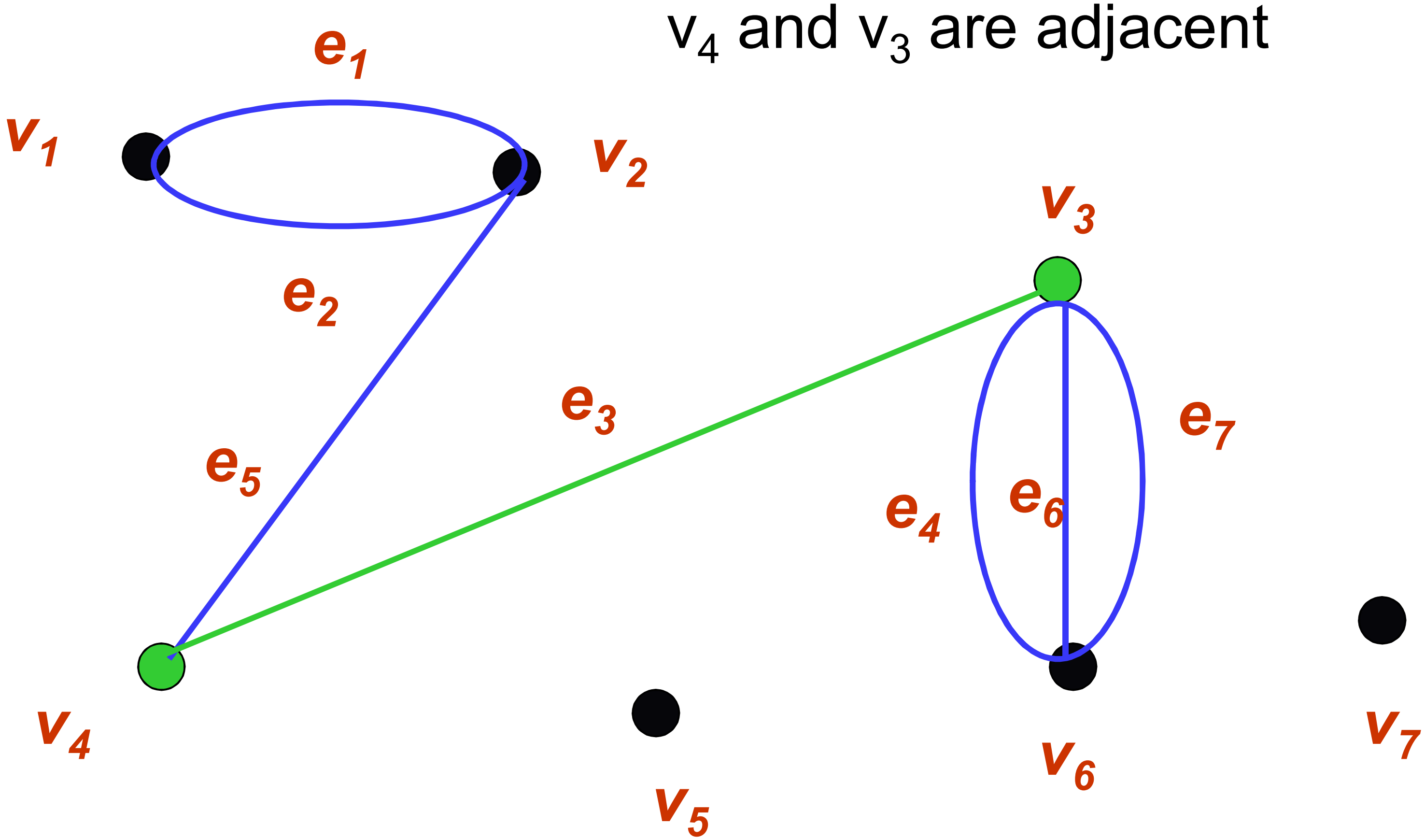
- An edge *e* in a graph that is associated with the pair of vertices *v* and *w* is said to be incident on *v* and *w*, and *v* and *w* are said to be incident on *e* and to be adjacent vertices.

- A vertex that is an endpoint of a loop is said to be adjacent to itself.

prepared by Razana Alwee

# example

$v_4$ and $v_3$ are adjacent



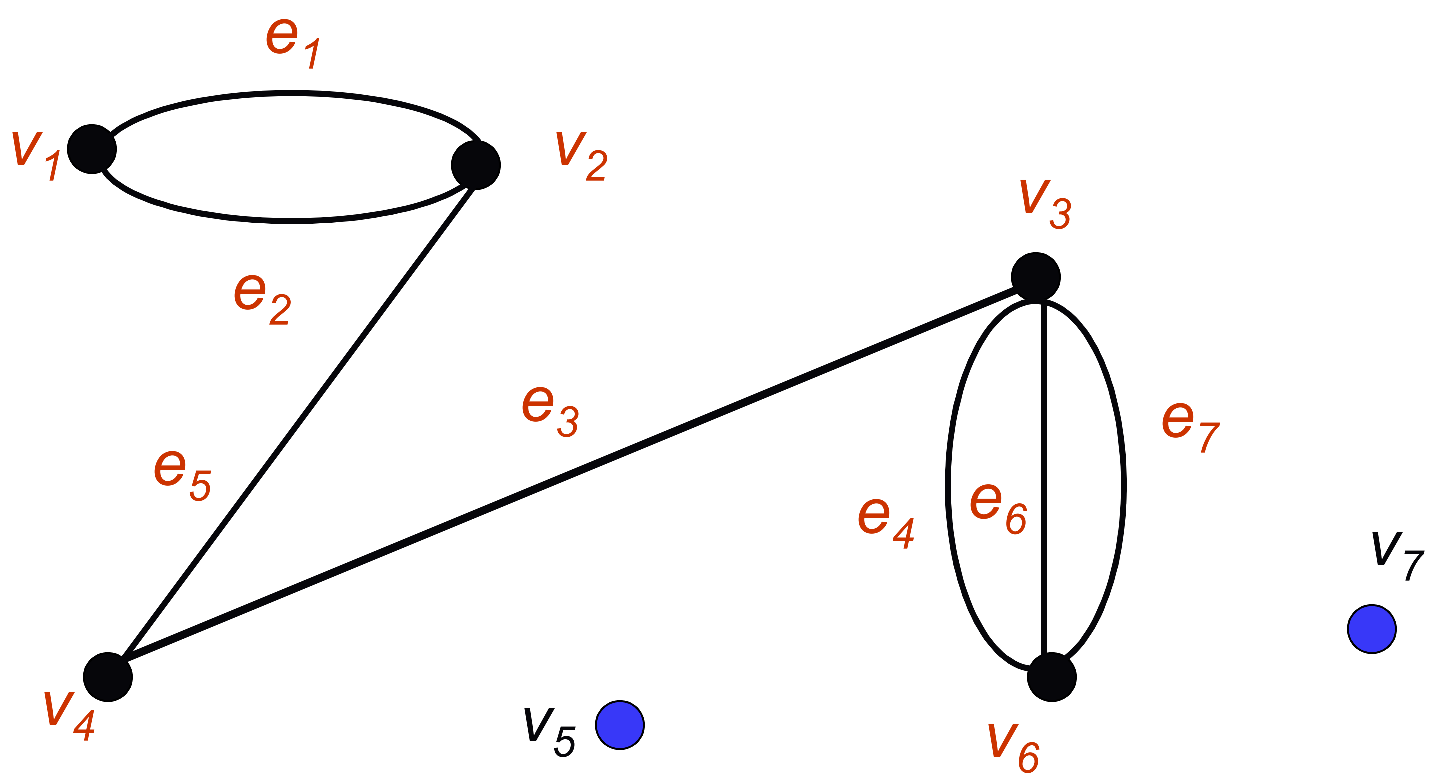

prepared by Razana Alwee

# Isolated Vertex

- Let $G$ be a graph and $v$ be a vertex in $G$.
- We say that $v$ is an isolated vertex if it is not incident with any edge.

# example

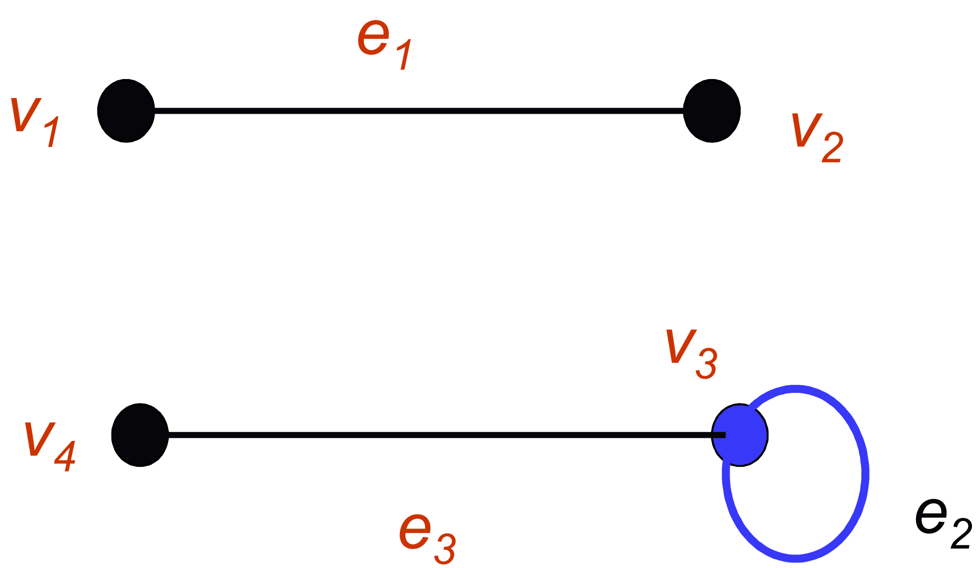
- *v*5 and *v*7 are isolated vertices.



# Loop

- An edge incident on a single vertex is called a loop.

  **Example**: $e_2$ is a loop



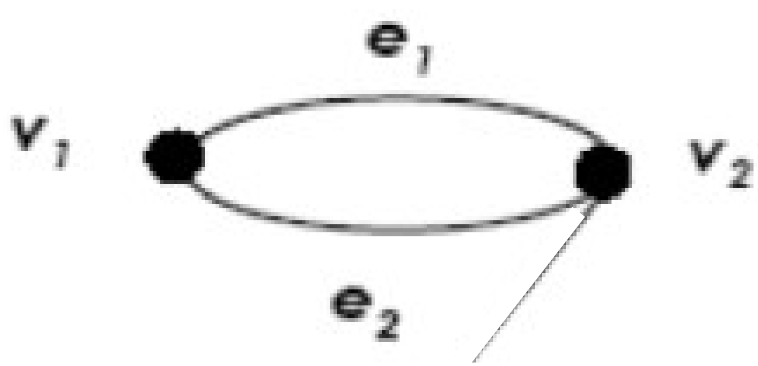

prepared by Razana Alwee

# Parallel Edges

Two or more distinct edges with the same set of endpoints are said to be parallel.

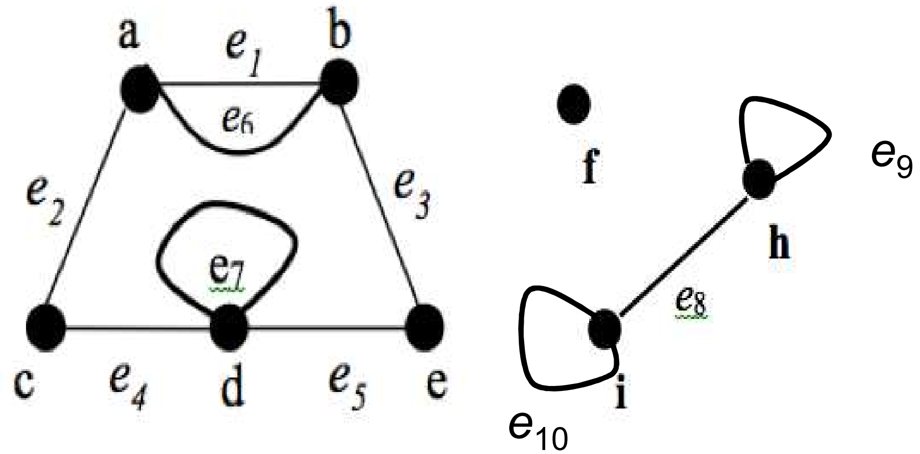
- $e_1$ and $e_2$ are parallel.

**Example**



# Example

Given a graph as shown below,




a) Write a vertex set and the edge set, and give a table showing the edge-endpoint function.

b) Find all edges that are incident on a, all vertices that are adjacent to a, all edges that are adjacent to $e_2$, all loops, all parallel edges, all vertices that are adjacent to themselves and all isolated vertices.

# Example 1 - Solution

**Solution:**

a) Vertex set, V = {a, b, c, d, e, f, i, h} and
the set of edges, E = {$e_1$, $e_2$, $e_3$, $e_4$, $e_5$, $e_6$, $e_7$, $e_8$, $e_9$, $e_{10}$}

| Edge | Endpoints |
|---|---|
| $e_1$ | {a, b} |
| $e_2$ | {a, c} |
| $e_3$ | {b, e} |
| $e_4$ | {c, d} |
| $e_5$ | {d, e} |
| $e_6$ | {a, b} |
| $e_7$ | {d} |
| $e_8$ | {i, h} |
| $e_9$ | {h} |
| $e_{10}$ | {i} |

b)

| | |
|---|---|
| incident on a, | e1, e2, e6 |
| adjacent to a, | c, b |
| adjacent to $e_2$, | e1, e4, e6 |
| loops, | e7, e9, e10 |
| parallel edges, | e1, e6 |
| adjacent to themselves, | i , h, d |
| isolated vertices, | f |

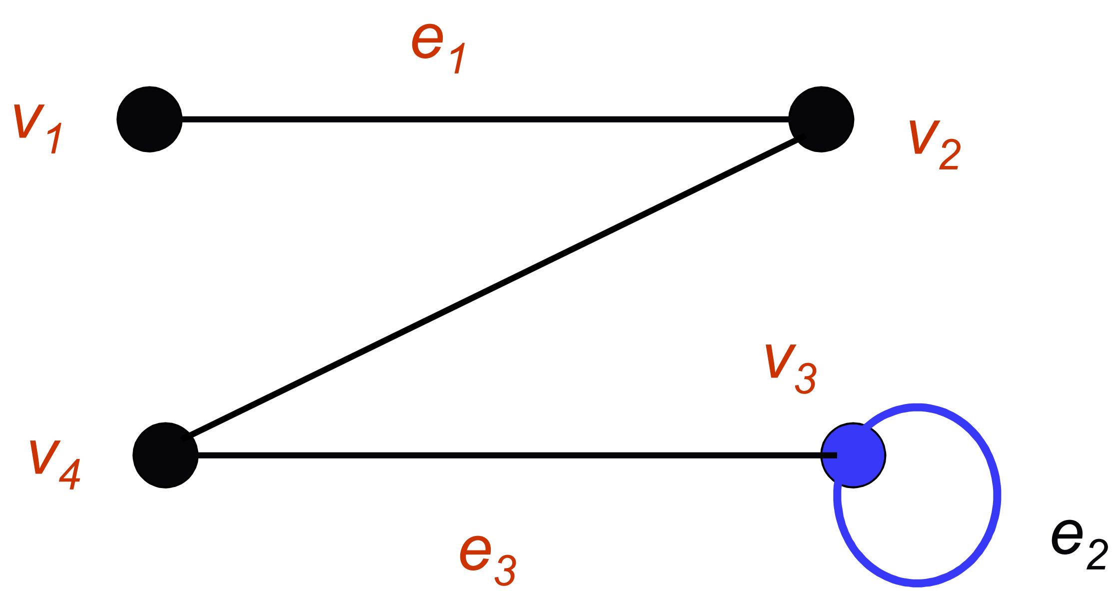
# The Concept of Degree

# Degree of a vertex

- Let *G* be a graph and *v* be a vertex of *G*.
- The degree of *v*, written **deg(*v*)** or **d(*v*)** is the number of edges incident with *v*.
- Each loop on a vertex *v* contributes **2** to the degree of *v*.

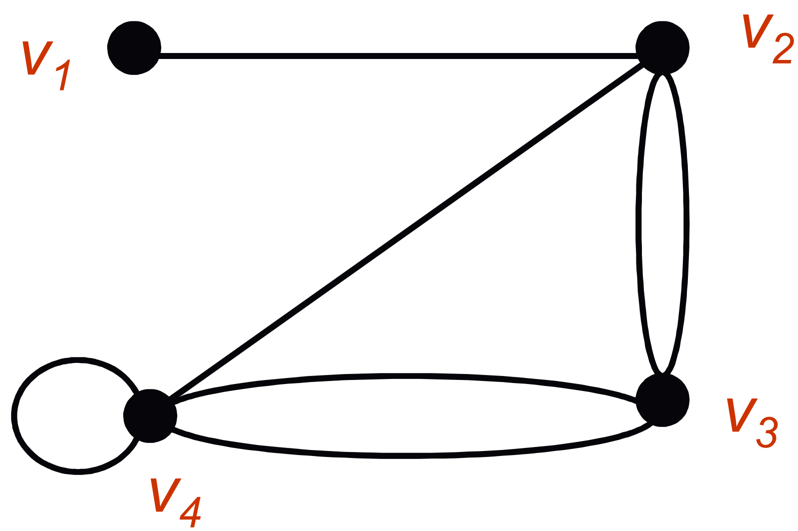
# example

- deg($v_1$ )= 1; deg($v_2$)= 2; deg($v_3$)= 3; deg($v_4$)= 2



# exercise

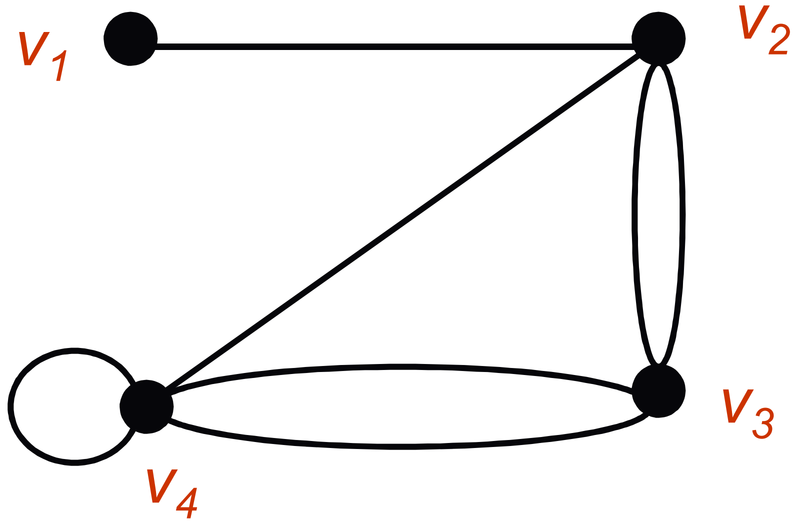
- Find the degree of each vertex in the graph.




prepared by Razana Alwee

# solution

- Find the degree of each vertex in the graph.

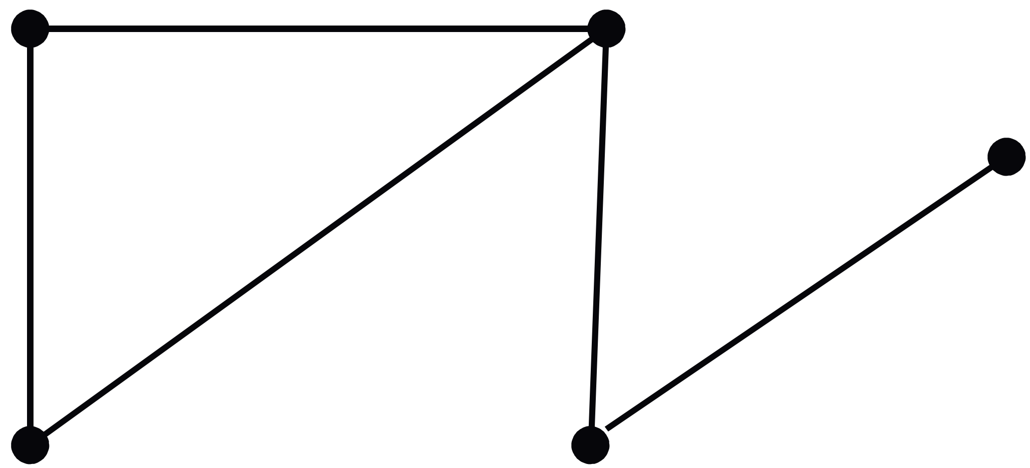



**Solution: deg($v_1$ )= 1; deg($v_2$)= 4; deg($v_3$)= 4; deg($v_4$)= 5**

# Types of Graphs

# Simple Graphs

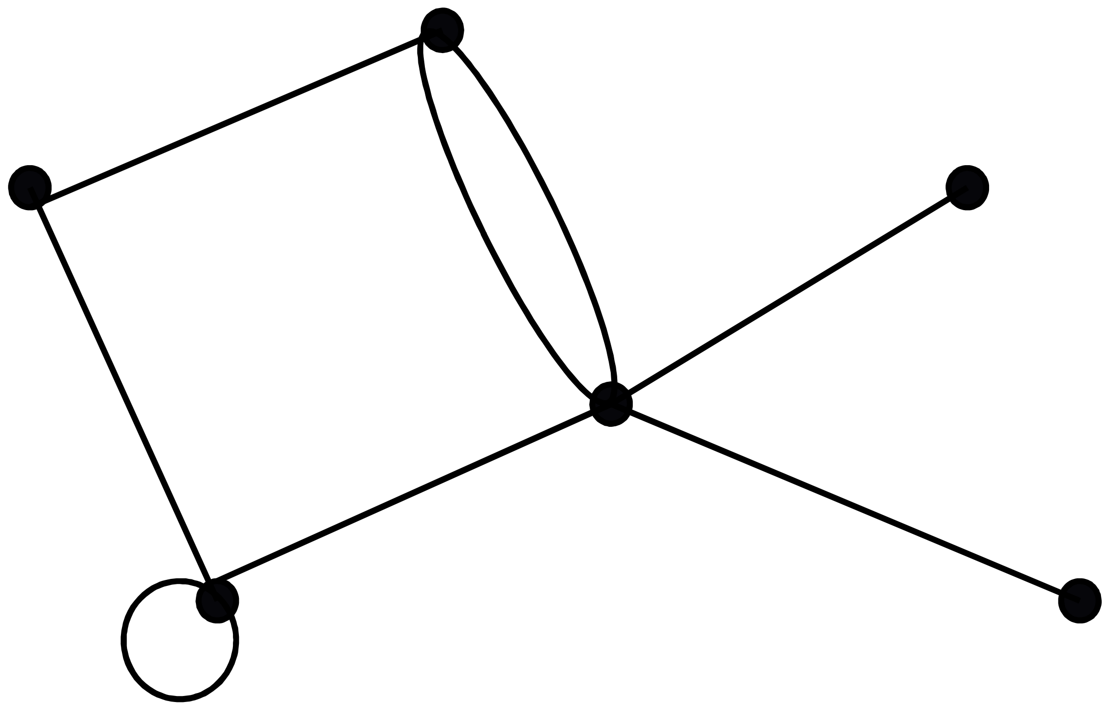
- A graph *G* is called a simple graph if *G* does not contain any parallel edges and any loops.
- **Example**



prepared by Razana Alwee

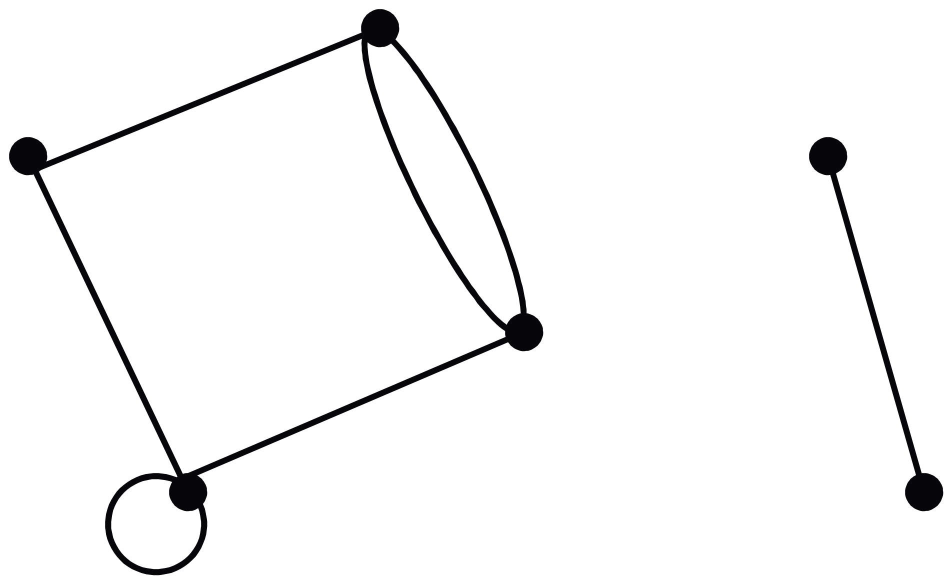
# Connected Graph

- A graph *G* is connected if given any vertices *v* and *w* in *G*, there is a path from *v* to *w*.
- **Example:**



# example

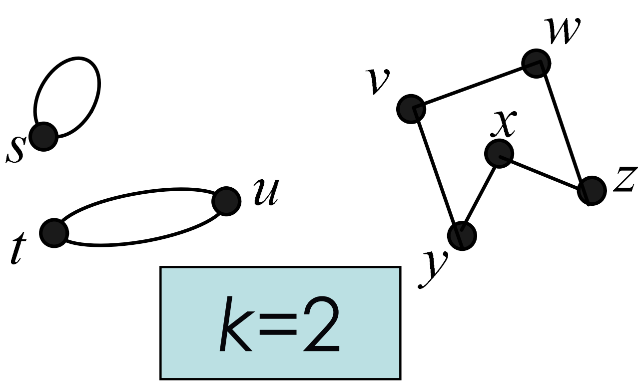
- not connected

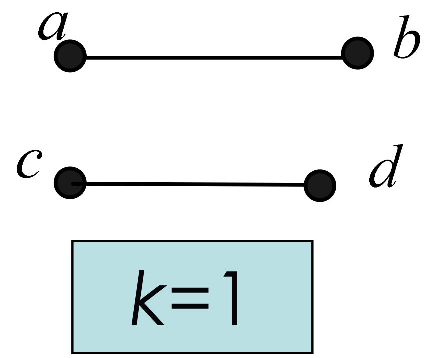


# Regular Graphs

- Let $G$ be a graph and $k$ be a nonnegative integer.
- $G$ is called a $k$-regular graph if the degree of each vertex of $G$ is $k$.

# example

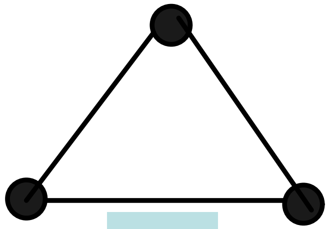


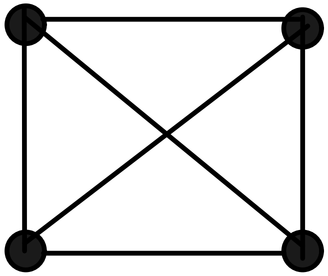
# Complete Graph

- A simple graph with *n* vertices in which there is an edge between every pair of distinct vertices is called a complete graph on *n* vertices.
- This is denoted by $K_n$.
- **Example**

$K_2$



$K_3$

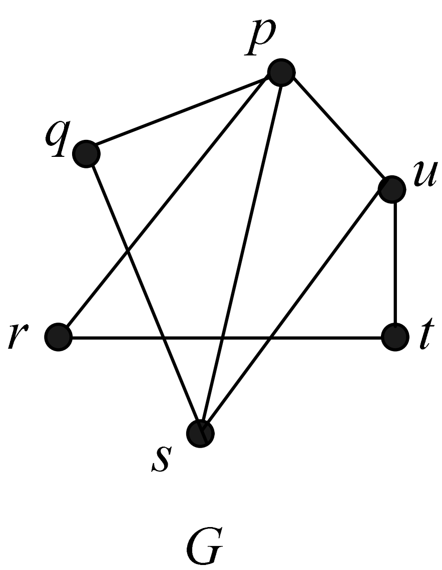


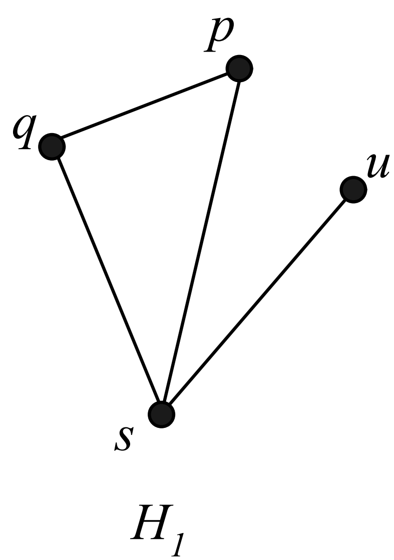
$K_4$

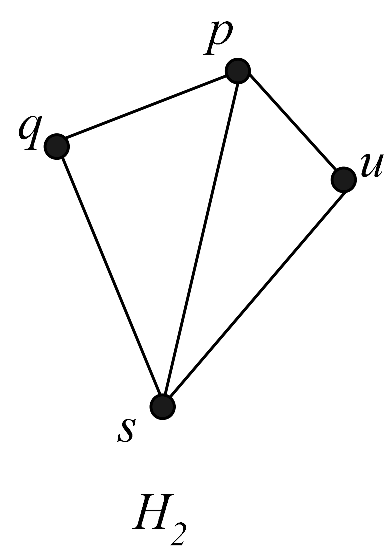
# Subgraph

- Let $G=(V,E)$ be a graph.
- $H=(U,D)$ is a subgraph of G if
  - $U \subseteq V$ and $D \subseteq E$
  - for every edge $e \in D$, if $e$ is incident on $v$ and $w$, then $v,w \in V$.

# example



# Graph Representation

# Matrix Representation of a Graph

- To write programs that process and manipulate graphs, the graphs must be stored, that is, represented in computer memory.
- A graph can be represented (in computer memory) in several ways.
- 2-dimensional array: adjacency matrix and incidence matrix.

# Adjacency Matrices

- Let $G$ be a graph with $n$ vertices.

- The adjacency matrix, $A_G$ is an $n \times n$ matrix $[a_{ij}]$ such that,

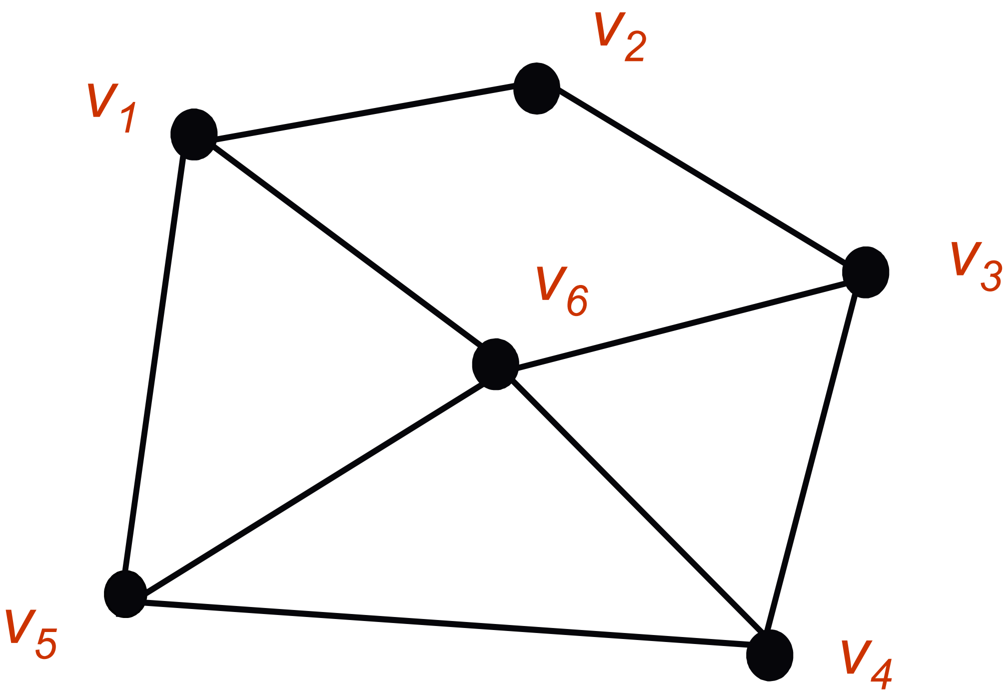
  $a_{ij}$= the number of edges from $v_i$ to $v_j$, {undirected G}

  or,

  $a_{ij}$= the number of arrows from $v_i$ to $v_j$, {directed G}

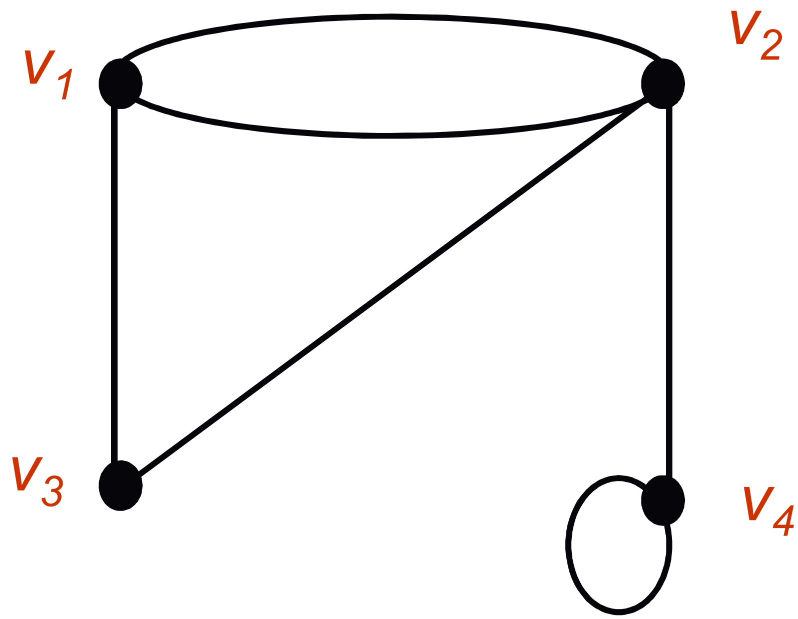
  for all $i$, $j$ = 1,2,….., n.

# example




$$A_G = \begin{bmatrix} 0 & 1 & 0 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 & 1 \\ 1 & 0 & 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 1 & 1 & 0 \end{bmatrix}$$

# example

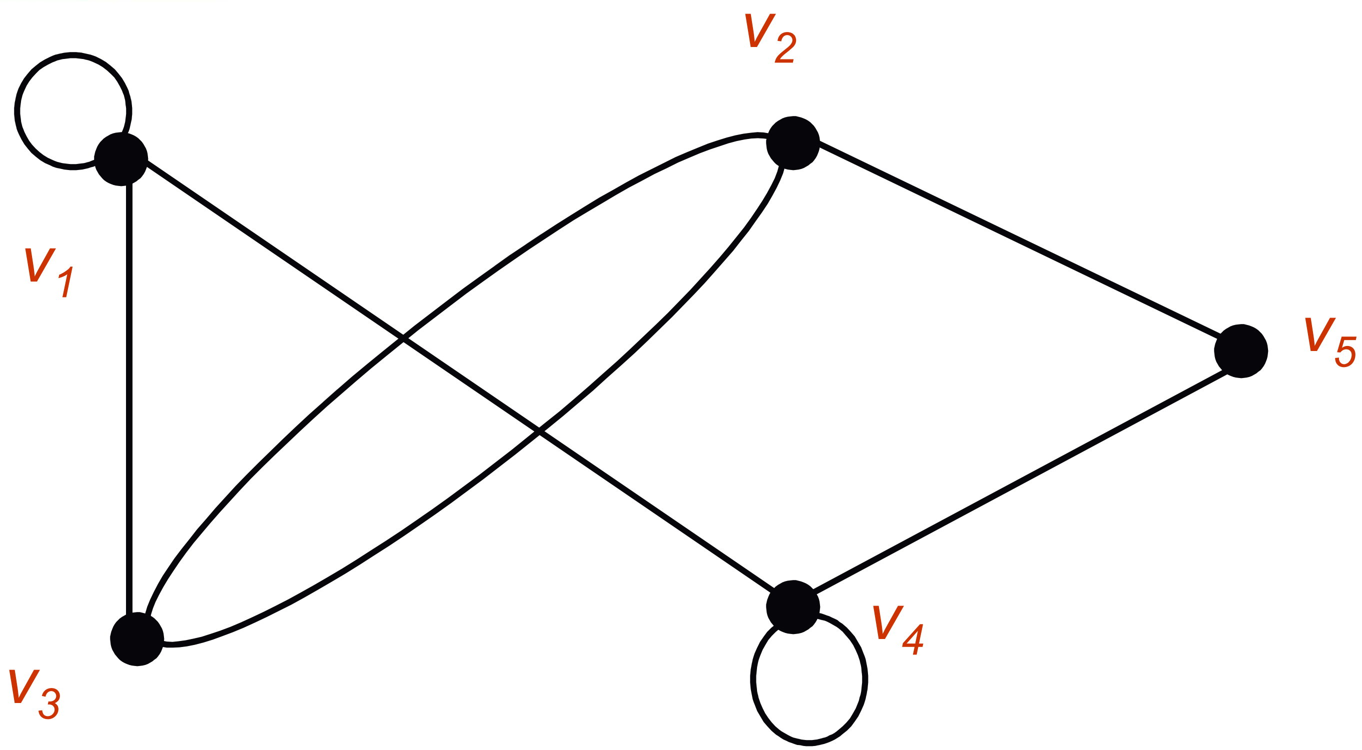



$$A_G = \begin{bmatrix} 0 & 2 & 1 & 0 \\ 2 & 0 & 1 & 1 \\ 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 \end{bmatrix}$$

# example

$$A_G = \begin{bmatrix} 1 & 0 & 1 & 1 & 0 \\ 0 & 0 & 2 & 0 & 1 \\ 1 & 2 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 & 0 \end{bmatrix}$$

# example



# Adjacency Matrices

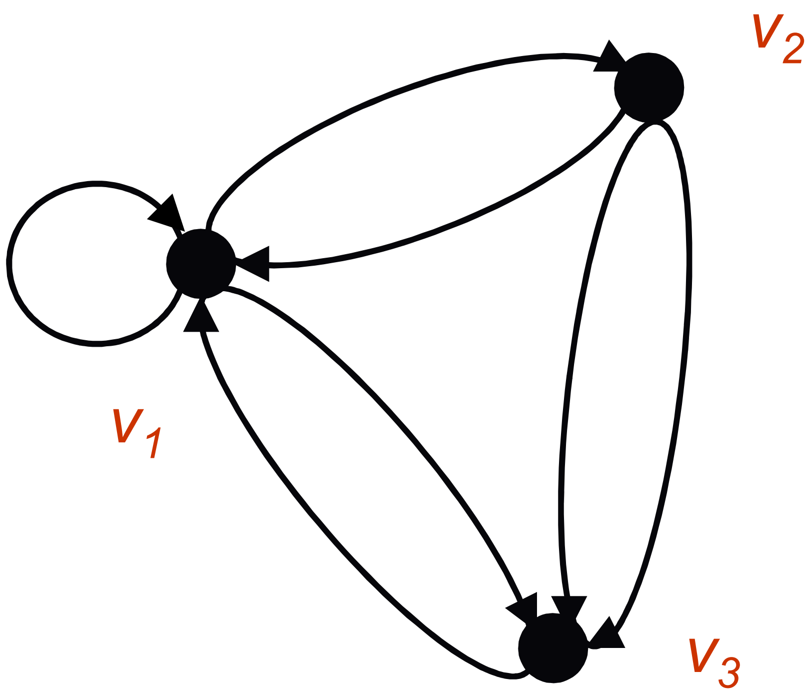
- Notice that the matrix $A_G$ is a **symmetric matrix** if it is representing an undirected graph, where

$$a_{ij} = a_{ji}$$

- If G is a directed graph (**digraph**), then $A_G$ need not be a symmetric matrix.

prepared by Razana Alwee

# example




$$\begin{bmatrix} 1 & 1 & 1 \\ 1 & 0 & 2 \\ 1 & 0 & 0 \end{bmatrix}$$

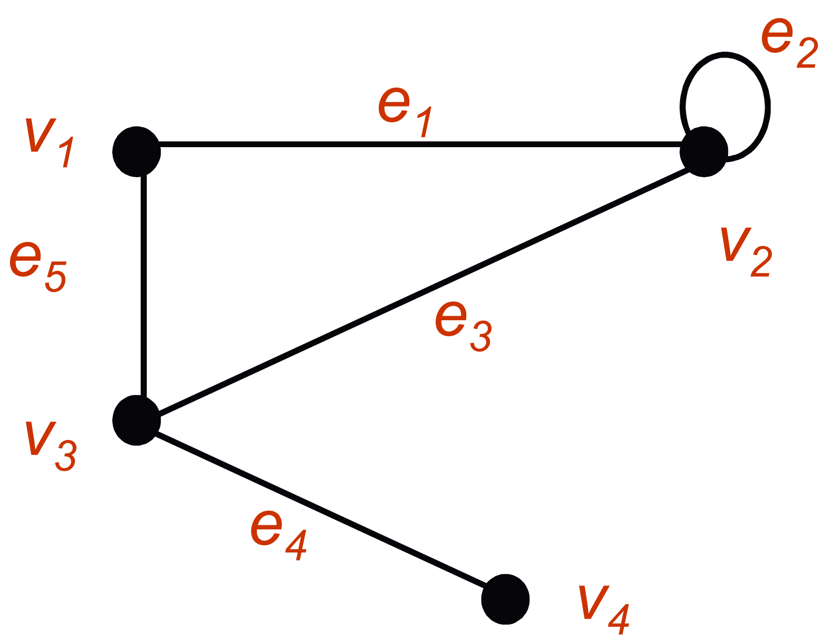
# Incidence Matrices

- Let *G* be a graph with *n* vertices and *m* edges.

- The incidence matrix $I_G$ is an $n \times m$ matrix $[a_{ij}]$ such that,

$$a_{ij} = \begin{cases} 0 & \text{if } v_i \text{ is not an end vertex of } e_j, \\ 1 & \text{if } v_i \text{ is an end vertex of } e_j, \text{but } e_j \text{ is not a loop} \\ 2 & \text{if } e_j \text{ is a loop at } v_i \end{cases}$$

# example



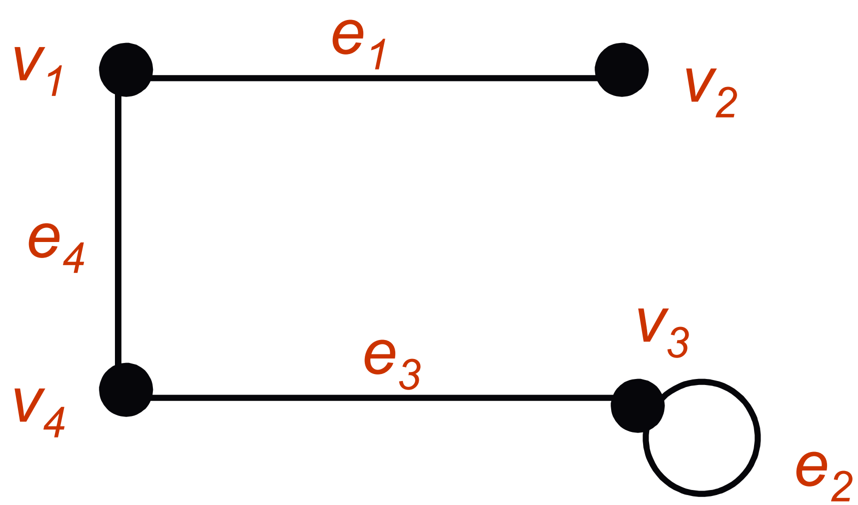

$$
\begin{array}{c}
\begin{array}{ccccc} \quad e_1 & e_2 & e_3 & e_4 & e_5 \end{array} \\
\begin{array}{c} v_1 \\ v_2 \\ v_3 \\ v_4 \end{array}
\begin{bmatrix}
1 & 0 & 0 & 0 & 1 \\
1 & 2 & 1 & 0 & 0 \\
0 & 0 & 1 & 1 & 1 \\
0 & 0 & 0 & 1 & 0
\end{bmatrix}
\end{array}
$$

Notice that the sum of the *i*th row is the degree of $v_i$

prepared by Razana Alwee

# exercise

- Find the adjacency matrix and the incidence matrix of the graph.




prepared by Razana Alwee

# Exercise
# Past Year 2015/2016

A cat show is being judged from pictures of the cats. The judges would like to see pictures of the following pairs of cats next to each other for their final decision: Fifi and Putih, Fifi and Suri, Fifi and Bob, Bob and Cheta, Bob and Didi, Bob and Suri, Cheta and Didi, Didi and Suri, Didi and Putih, Suri and Putih, Putih and Jeep, Jeep and Didi.

Draw a graph modeling this situation. (3 marks)

# Exercise
# Past Year 2015/2016

Given a graph as shown in Figure 1.

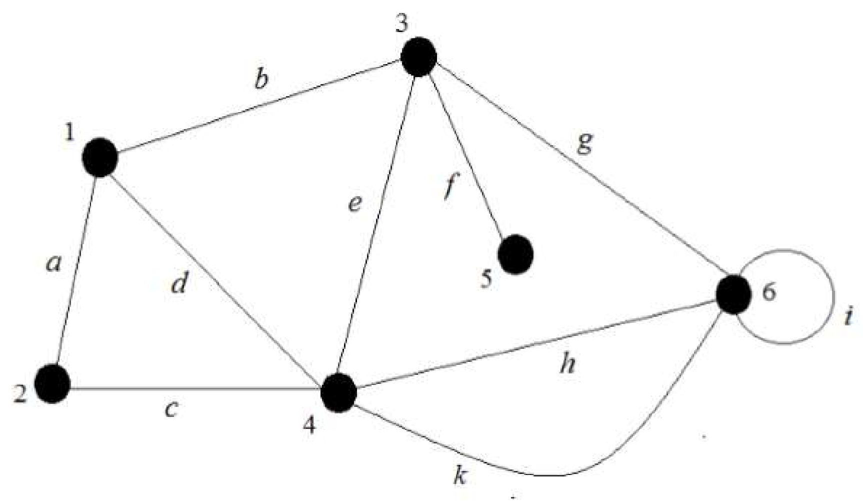


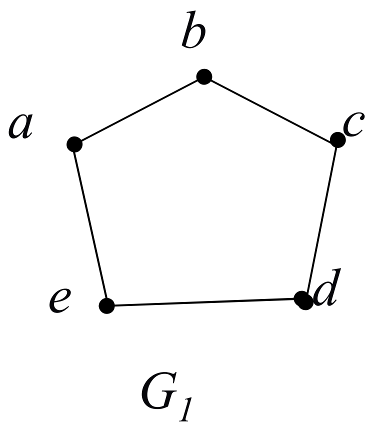

**Figure 1**

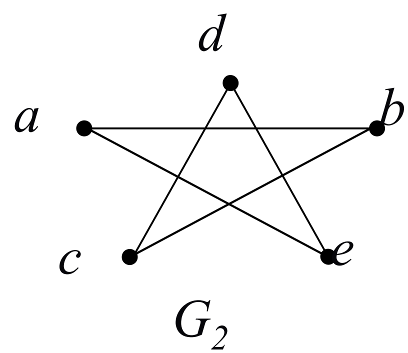
i. Find the incidence matrix of the graph. (4 marks)

ii. Find the adjacency matrix of the graph. (3 marks)

# Isomorphisms

# Isomorphism




- Are these 2 graphs the same?
- When we say that 2 graphs are the same mean they are isomorphic to each other.

# Isomorphism

- Graphs $G_1$ and $G_2$ are isomorphic if there is
  a one-to-one, onto function $f$ from the vertices of $G_1$ to the vertices of $G_2$
  and
  a one-to-one, onto function $g$ from the edges of $G_1$ to the edges of $G_2$

# Isomorphism

- An edge *e* is incident on *v* and *w* in $G_1$ if and only if the edge *g(e)* is incident on *f(v)* and *f(w)* in $G_2$.

- The pair of functions *f* and *g* is called an isomorphism of $G_1$ onto $G_2$.

- Graphs $G_1$ and $G_2$ are isomorphic if and only if for some ordering of their vertices, their adjacency matrices are equal.

# Definition

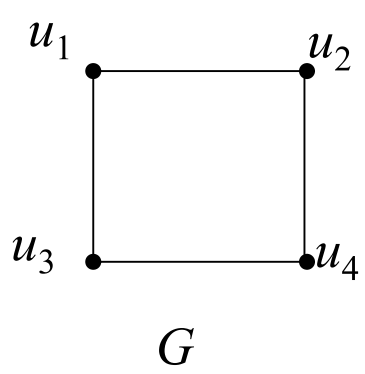
Let $G = \{V, E\}$ and $G' = \{V', E'\}$ be graphs. $G$ and $G'$ are said to be isomorphic if there exist a pair of functions $f : V \to V'$ and $g : E \to E'$ such that $f$ associates each element in $V$ with exactly one element in $V'$ and vice versa; $g$ associates each element in $E$ with exactly one element in $E'$ and vice versa, and for each $v \in V$, and each $e \in E$, if $v$ is an endpoint of the edge $e$, then $f(v)$ is an endpoint of the edge $g(e)$.

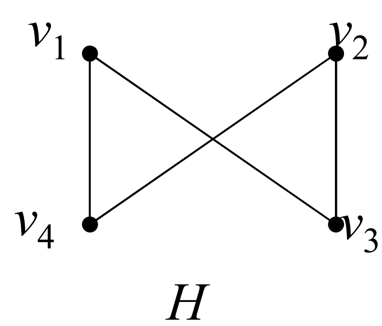
## Isomorphism

- If two graphs is isomorphic, they must have:
  - the same number of vertices and edges,
  - the same degrees for corresponding vertices,
  - the same number of connected components,
  - the same number of loops and parallel edges,
  - both graphs are connected or both graph are not connected,
  - pairs of connected vertices must have the corresponding pair of vertices connected.
- In general, it is easier to prove two graphs are not isomorphic by proving that one of the above properties fails.

# example

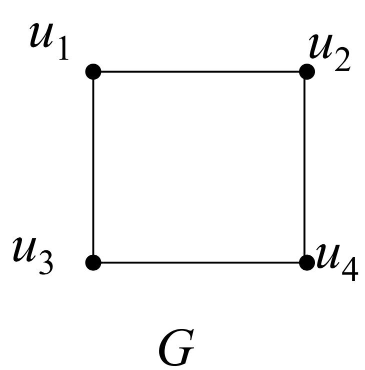
- Determine whether *G* is isomorphic to *H.*

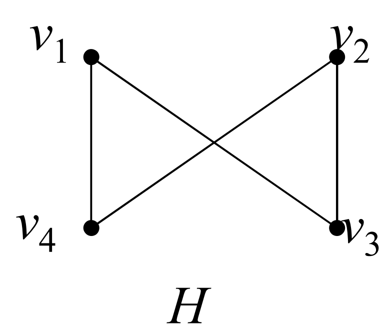


# example

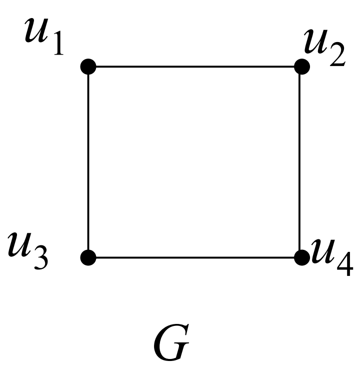
- Both graphs are simple and have the same number of vertices and the same number of edges.

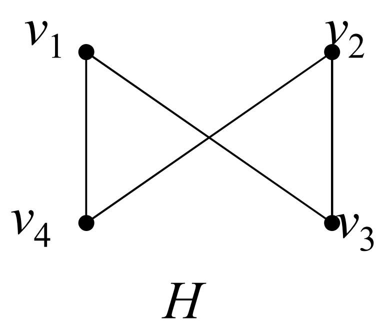


# example

- All the vertices of both graphs have degree 2.



# example

- Define $f$: $U \to V$, where $U=\{u_1, u_2, u_3, u_4\}$ and $V=\{v_1, v_2, v_3, v_4\}$

$f(u_1)=v_1$, $f(u_2)=v_4$, $f(u_3)=v_3$, $f(u_4)=v_2$


prepared by Razana Alwee

# example

- To verify whether $G$ and $H$ are isomorphic, we examine the adjacency matrix $A_G$ with rows and columns labeled in the order $u_1, u_2, u_3, u_4$

  and

  the adjacency matrix $A_H$ with rows and columns labeled in the order $v_1, v_4, v_3, v_2$.

# example

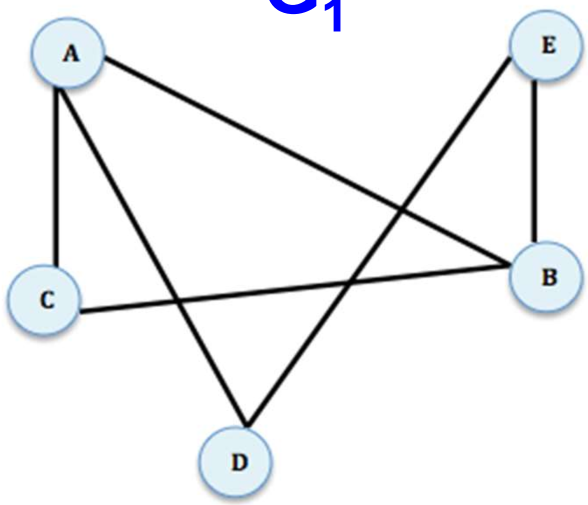
- $A_G$ and $A_H$ are the same, *G* and *H* are isomorphic.

$$A_G = \begin{array}{c} \\ u_1 \\ u_2 \\ u_3 \\ u_4 \end{array} \begin{array}{c} \begin{array}{cccc} u_1 & u_2 & u_3 & u_4 \end{array} \\ \begin{pmatrix} 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 \end{pmatrix} \end{array} \qquad A_H = \begin{array}{c} \\ v_1 \\ v_4 \\ v_3 \\ v_2 \end{array} \begin{array}{c} \begin{array}{cccc} v_1 & v_4 & v_3 & v_2 \end{array} \\ \begin{pmatrix} 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 \end{pmatrix} \end{array}$$

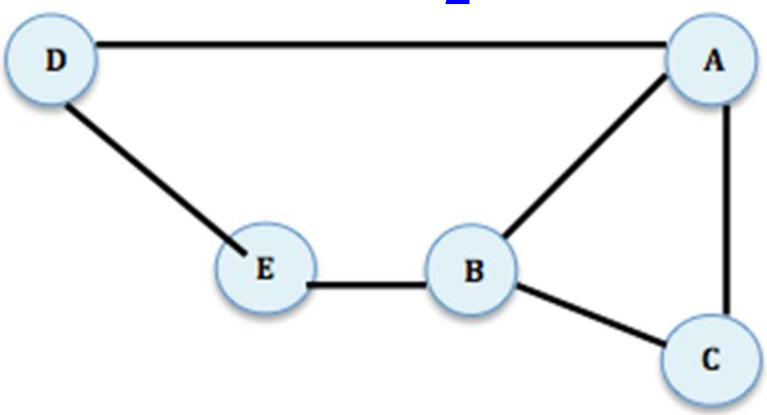
# Exercise

**Q: Show that the following two graphs are isomorphic.**

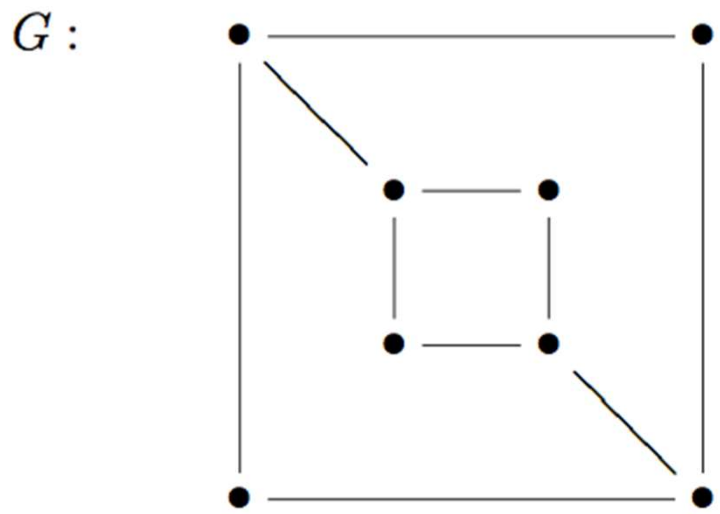
$G_1$

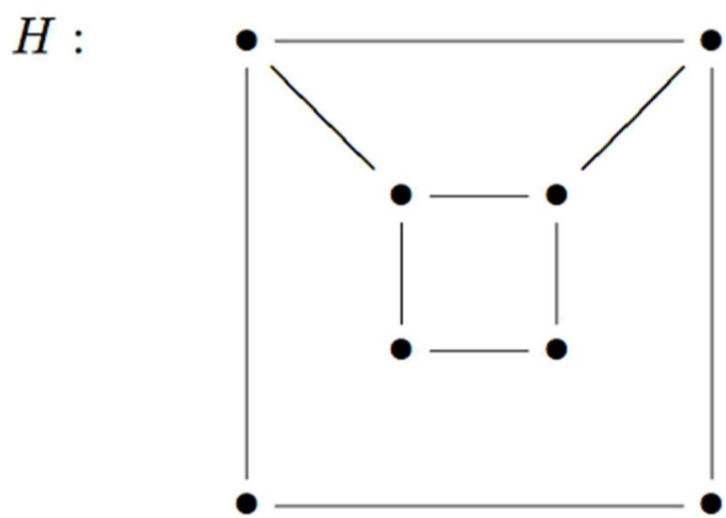



$G_2$



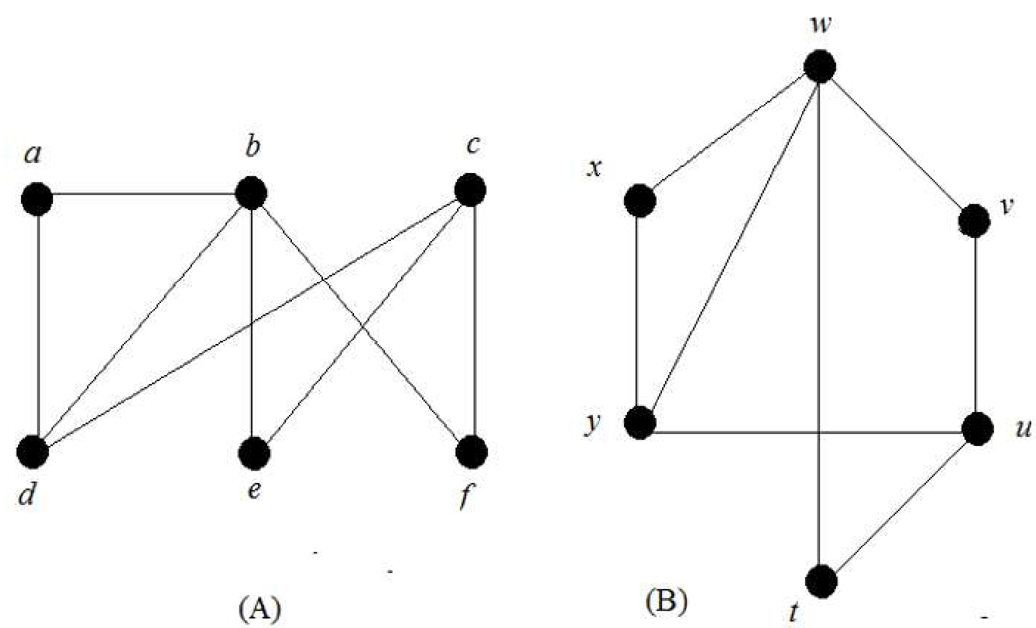
# Q: Is these two graphs are isomorphic?



# Exercise
# Past Year 2015/2016

Determine whether the graphs in Figure 2 ($A$ and $B$) are isomorphic. If the graphs are isomorphic, find their adjacency matrices; otherwise, give an invariant that the graphs do not share. (6 marks)



# Trails, Paths & Circuits

# Term and Description

- A walk from $v$ to $w$ is a finite alternating sequence of adjacent vertices and edges of G. Thus a walk has the form

  $$(v_0, e_1, v_1, e_2, v_2, \ldots\ldots, v_{n-1}, e_n, v_n)$$

  where the $v$'s represent vertices, the $e$'s represent edges, $v = v_0$, $w = v_n$, and for $i$ = 1,2,…,$n$. $v_{i-1}$ and $v_i$ are the endpoints of $e_i$.

- A trivial walk from $v$ to $w$ consist of the single vertex $v$. The walk contains zero edges (has length zero)

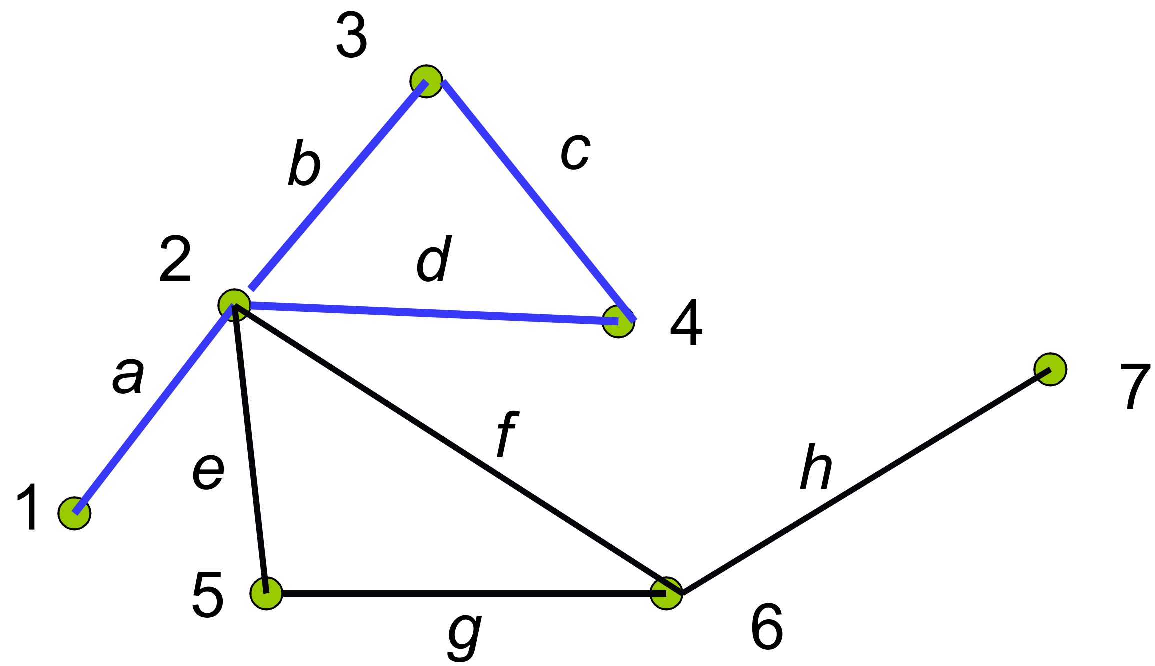
- The length of a walk is the number of edges it has.

# Term and Description (cont.)

- A trail from $v$ to $w$ is a walk from $v$ to $w$ that does not contain a repeated edge.
- A path from $v$ to $w$ is a trail from $v$ to $w$ that does not contain a repeated vertex.
- A closed walk is a walk that start and ends at the same vertex.
- A circuit/cycle is a closed walk that contains at least one edge and does not contain a repeated edge.
- A simple circuit is a circuit that does not have any other repeated vertex except the first and the last.

# example

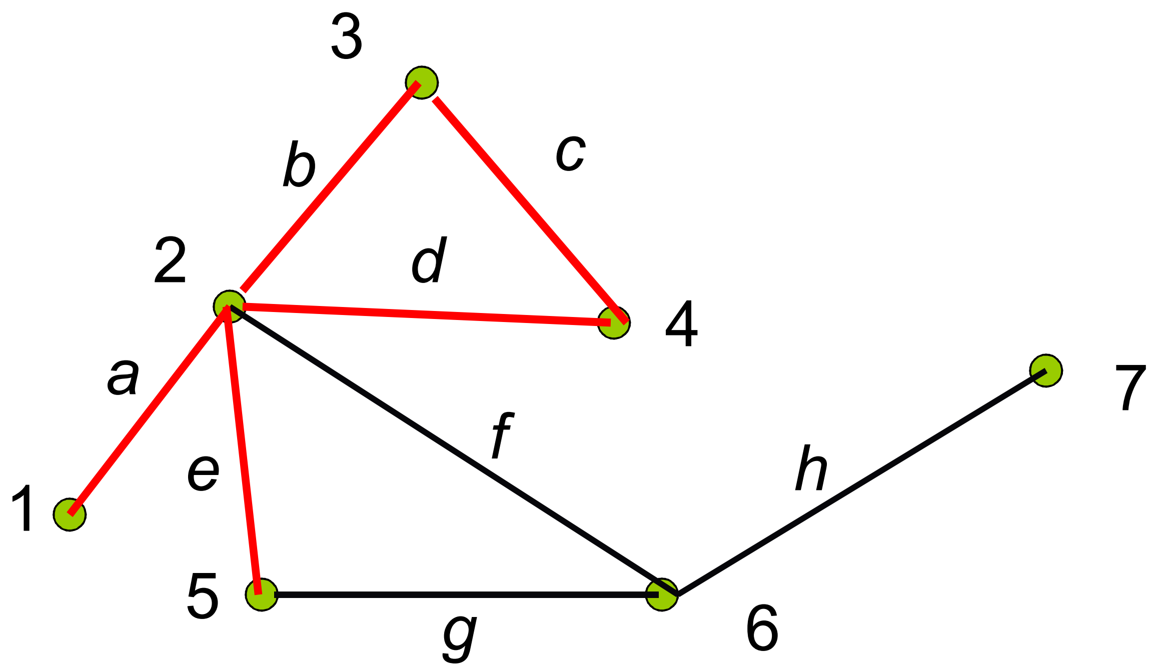
- (1, *a*, 2, *b*, 3, *c*, 4, *d*, 2) is a walk of length 4 from vertex 1 to vertex 2.




prepared by Razana Alwee

# example

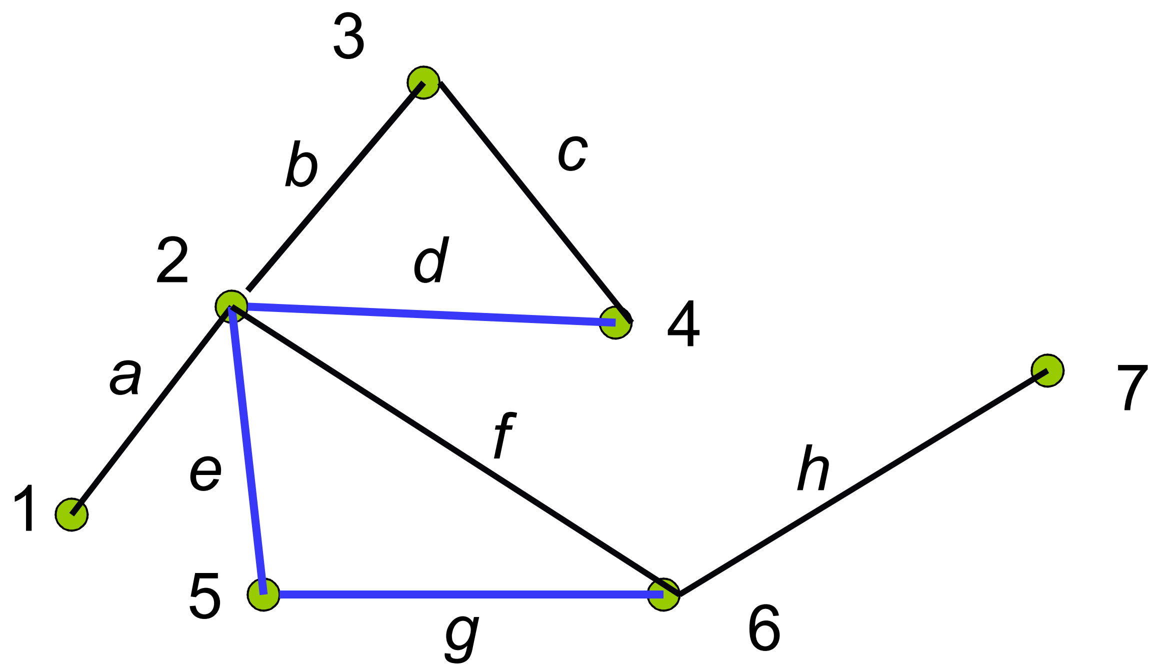
- (1, a, 2, b, 3, c, 4, d, 2, e, 5) is a trail.




Note:
Trail: No repeated edge (can repeat vertex).

# example

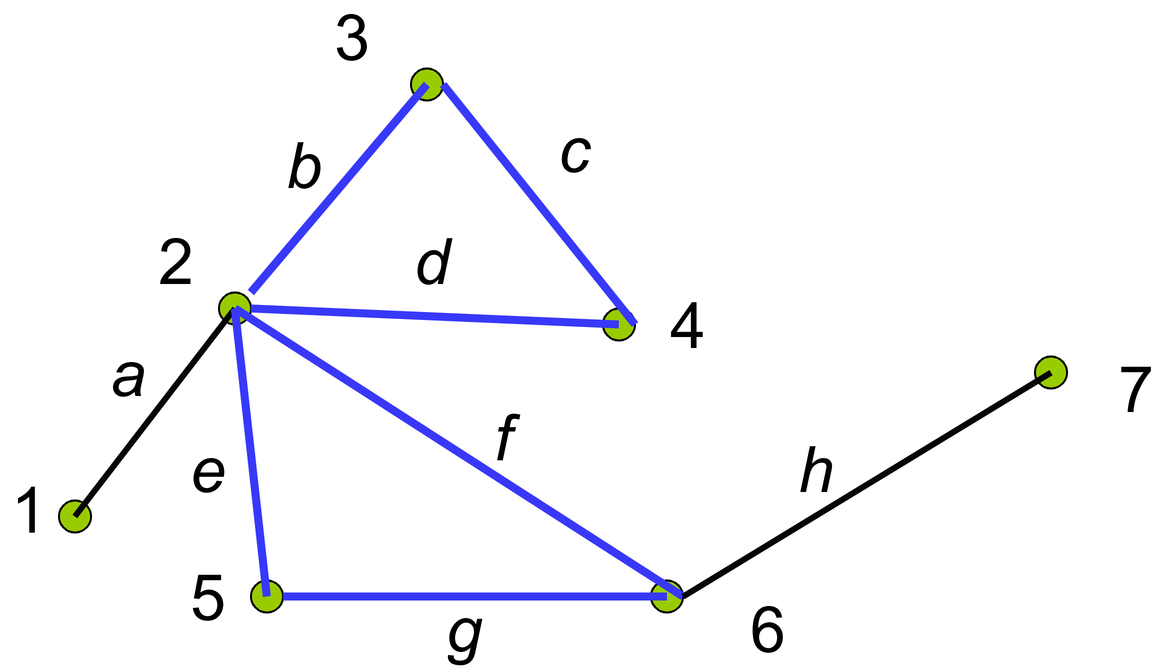
- (6, g, 5, e, 2, d, 4) is a path.




Note:
Path: No repeated vertex and edge.

# example

- (2, *f*, 6, *g*, 5, *e*, 2, *d*, 4, *c*, 3, *b*, 2) is a circuit/cycle.

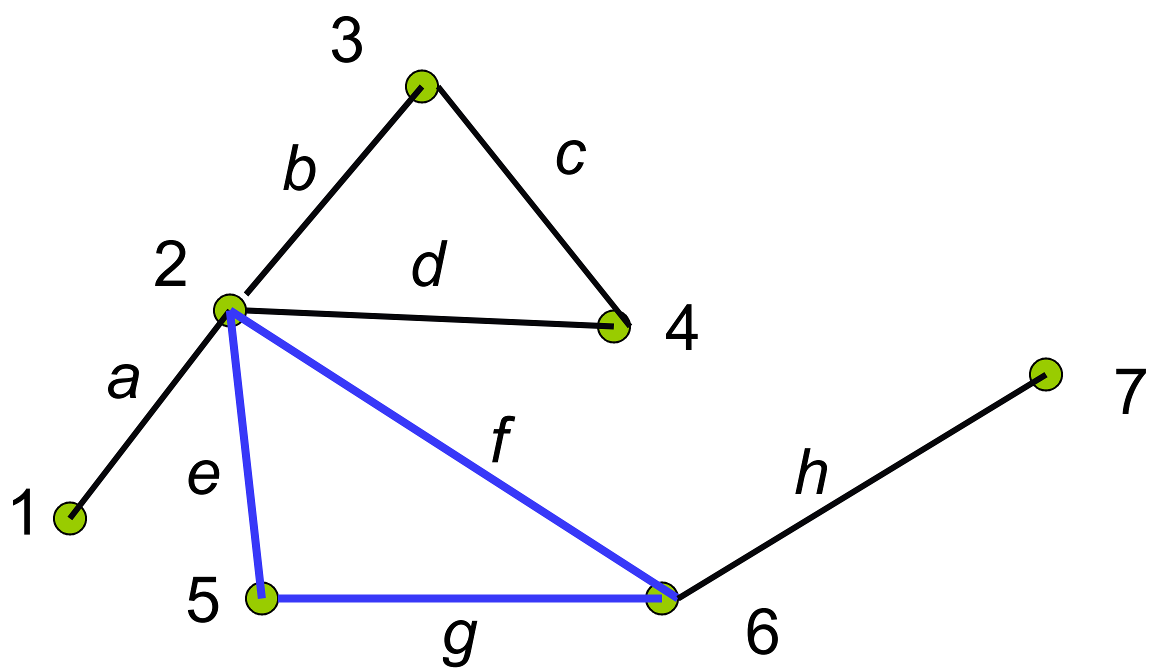



Note: circuit → start and end at same vertex, no repeated edge.

prepared by Razana Alwee

# example

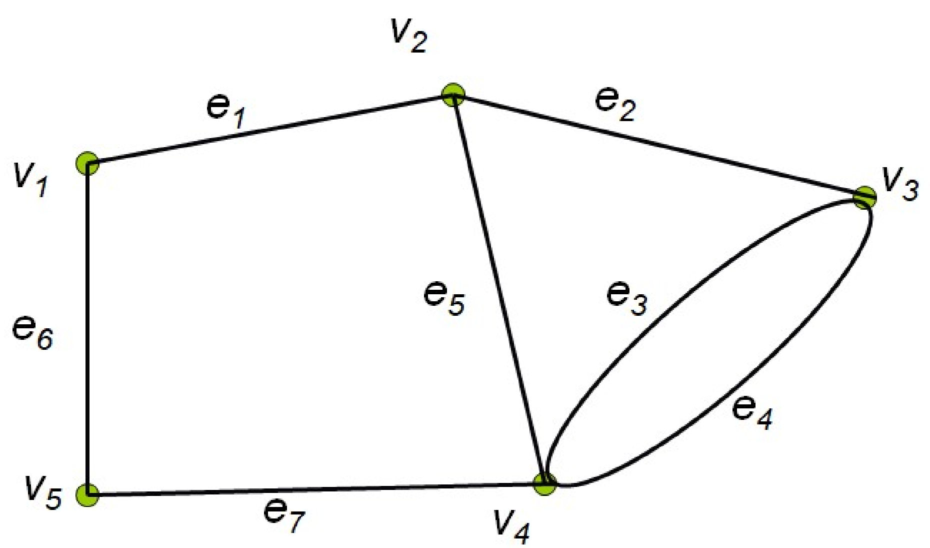
- (5, *g*, 6, *f*, 2, *e*, 5) is a simple circuit.




Note: Simple circuit →start and end at same vertex, no repeated edge or vertex except for the start and end vertex.

# exercise

- Tell whether the following is either a walk, trail, path, circuit, simple circuit, closed walk or none of these.
  - $(v_1, e_1, v_2)$
  - $(v_2, e_2, v_3, e_3, v_4, e_4, v_3)$
  - $(v_4, e_7, v_5, e_6, v_1, e_1, v_2, e_2, v_3, e_3, v_4)$
  - $(v_4, e_4, v_3, e_3, v_4, e_5, v_2, e_1, v_1, e_6, v_5, e_7, v_4)$



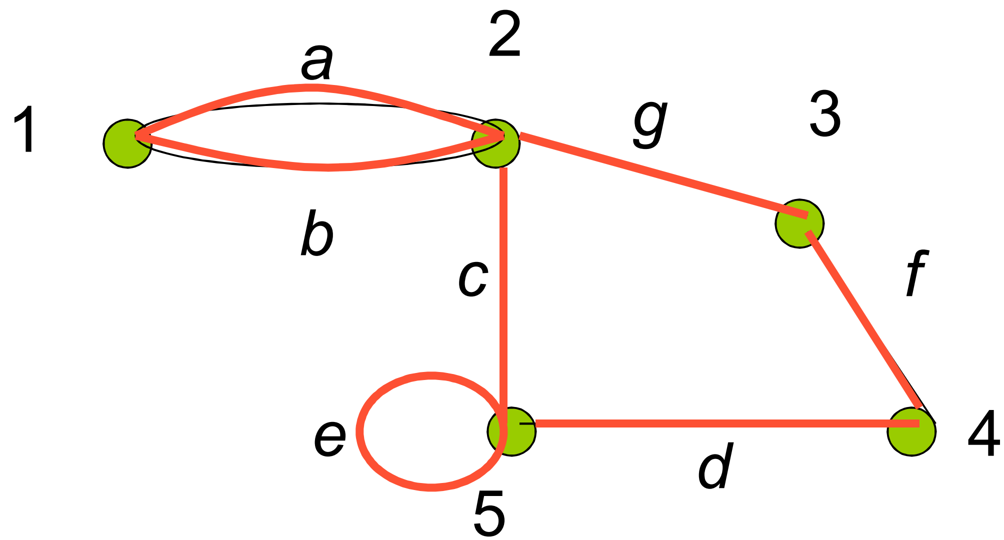
# Euler Trail & Circuit

# Euler Circuits

- A circuit in a graph that includes all the edges of the graph is called an Euler circuit.

- Let G be a graph. An Euler circuit for G is **a circuit that contains every vertex and every edges of G**. That is, an Euler circuit for G is a sequence of adjacent vertices and edges in G that has at least one edges, starts and ends at the same vertex, **uses every vertex of G at least once**, and **uses every edge of G exactly once**.

# example




(**1**, *a*, 2, *c*, 5, *e*, 5, *d*, 4, *f*, 3, *g*, 2, *b*, **1**)
is an Euler circuit

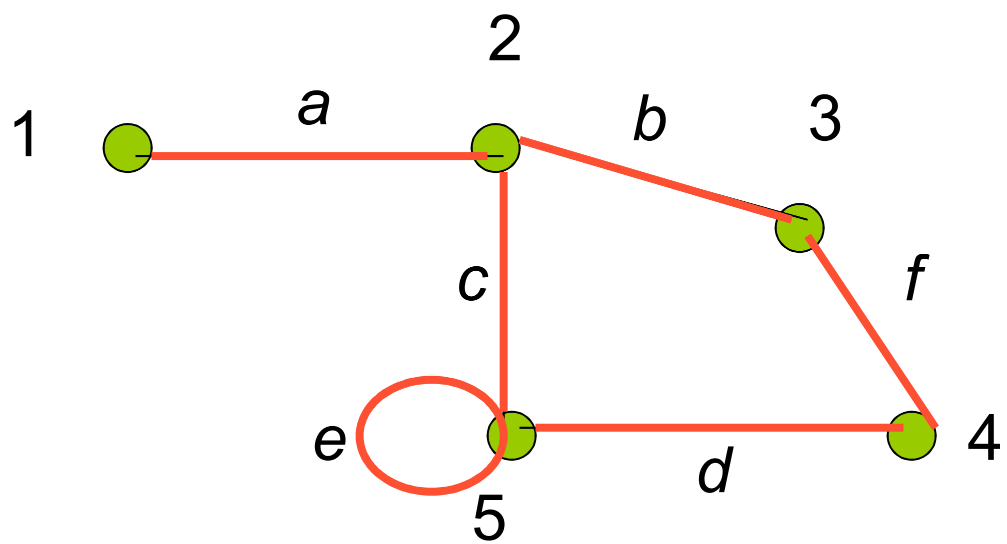
# Euler Trail

- A trail from *v* to *w* ($v \neq w$) with no repeated edges is called an Euler trail if it **contains all the edges and all the vertices**.

- Let G be a graph, and let $v$ and $w$ be two distinct vertices of G. An Euler trail from $v$ to $w$ is a sequence of adjacent vertices and edges that starts at $v$ and ends at $w$, passes through every **vertex** of G at least once, and traverses every **edge** of G exactly once.

prepared by Razana Alwee

# example




(1, *a*, 2, *c*, 5, *e*, 5, *d*, 4, *f*, 3, *b*, 2)
is an Euler trail

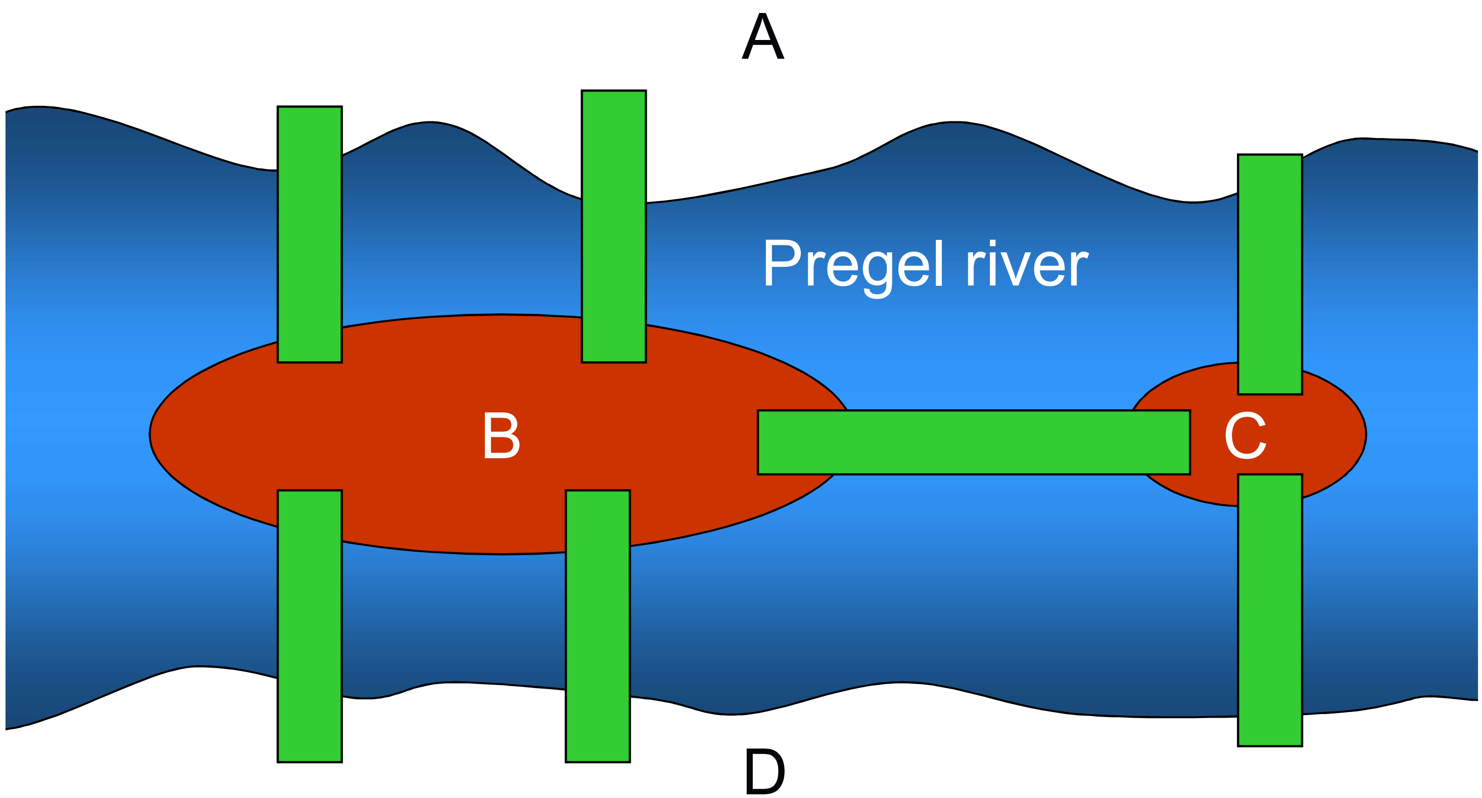
# Theorem

- If *G* is a connected graph and **every vertex has even degree**, then *G* has an **Euler circuit**.

- A graph has an **Euler trail** from *v* to *w* ($v \neq w$) if and only if it is connected and ***v* and *w* are the only vertices having odd degree**.

prepared by Razana Alwee

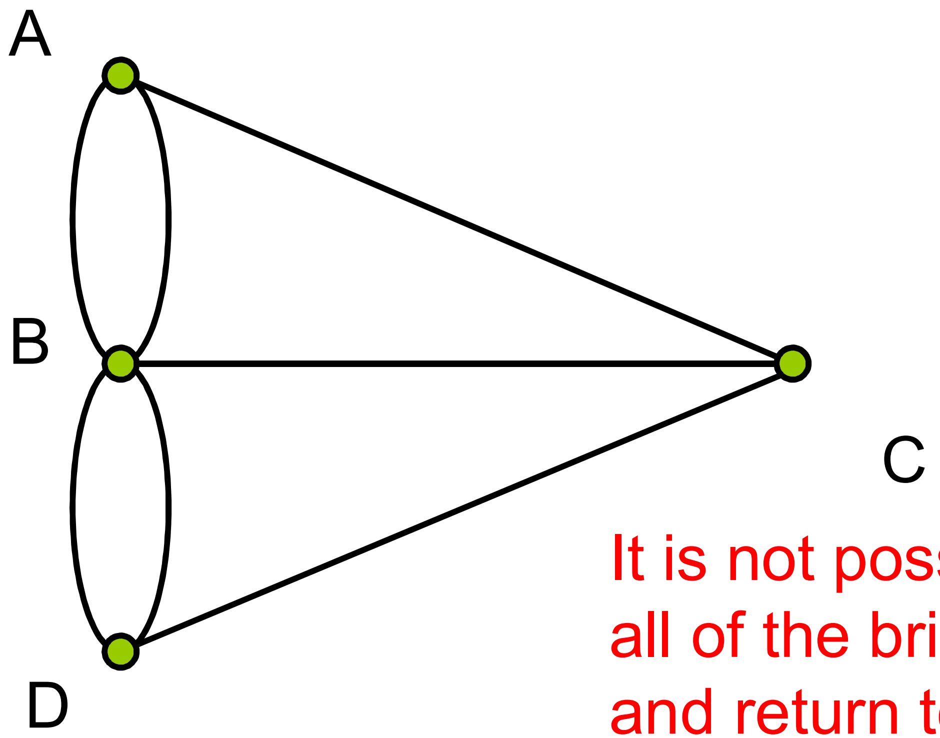
# Königsberg Bridge Problem




Starting at one land area, is it possible to walk across all of the bridges exactly once and return to the starting land area?

# Königsberg Bridge Problem

- Graph of the Königsberg Bridge Problem

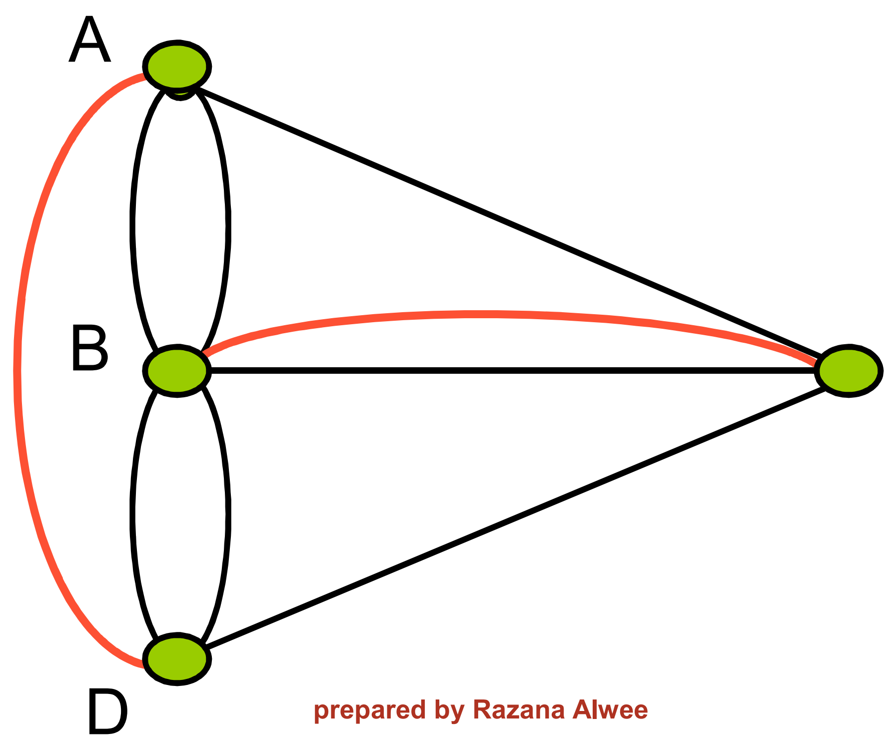



no Euler circuit

It is not possible to walk across all of the bridges exactly once and return to the starting land area

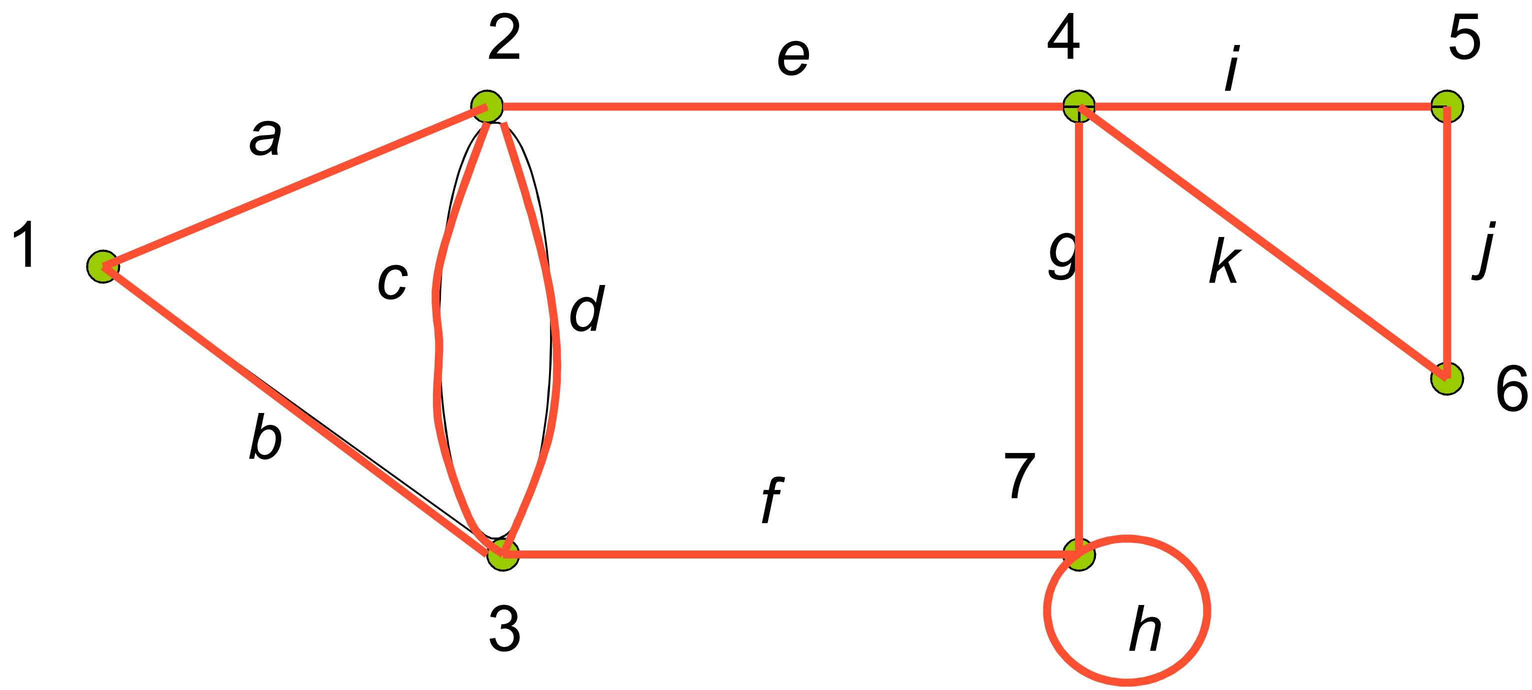
# Königsberg Bridge Problem

- Since 1736, two additional bridges have been constructed on the Pregel river.



# example

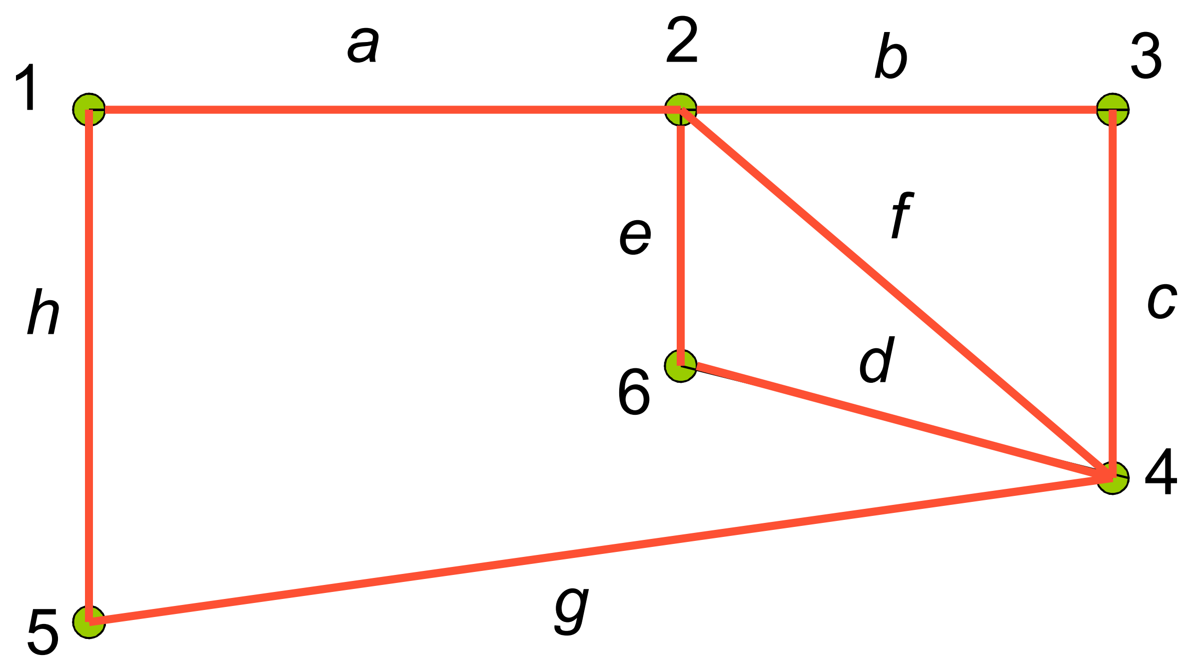
| Vertex | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|
| Degree | 2 | 4 | 4 | 4 | 2 | 2 | 4 |




This graph has an Euler circuit

# example

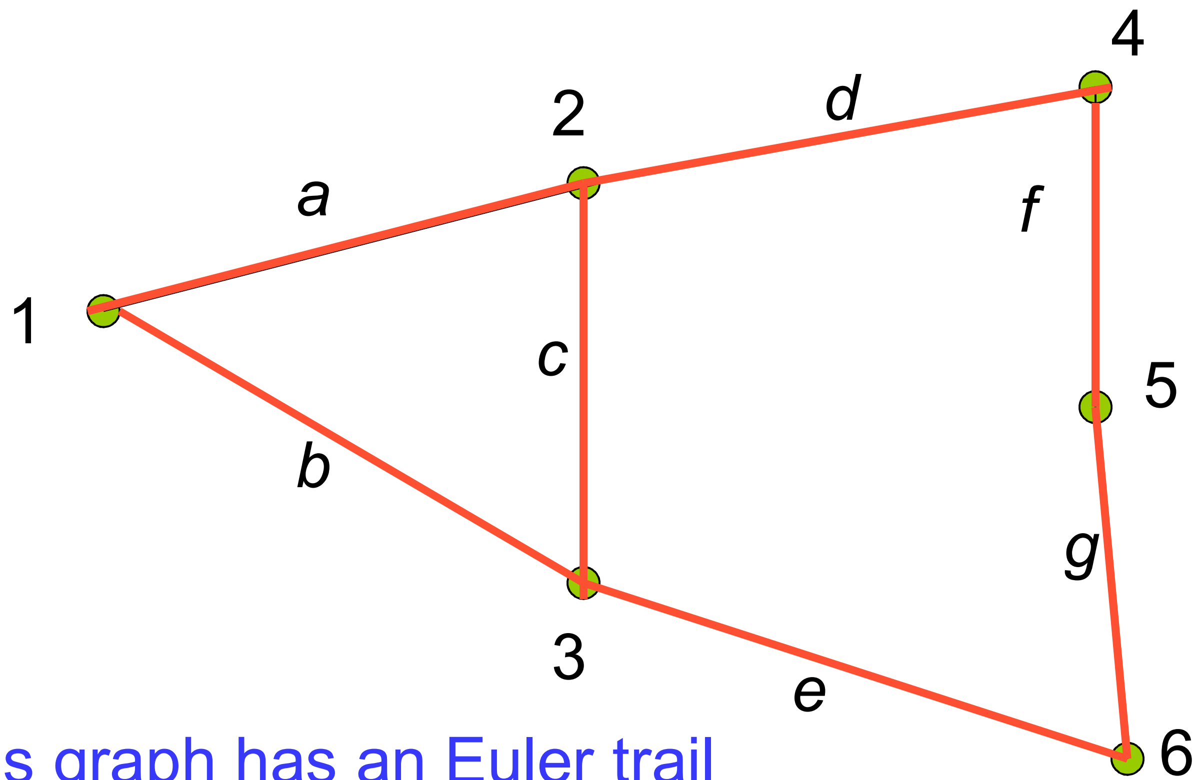
| Vertex | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|
| **Degree** | 2 | 4 | 2 | 4 | 2 | 2 |




This graph has an Euler circuit

# example

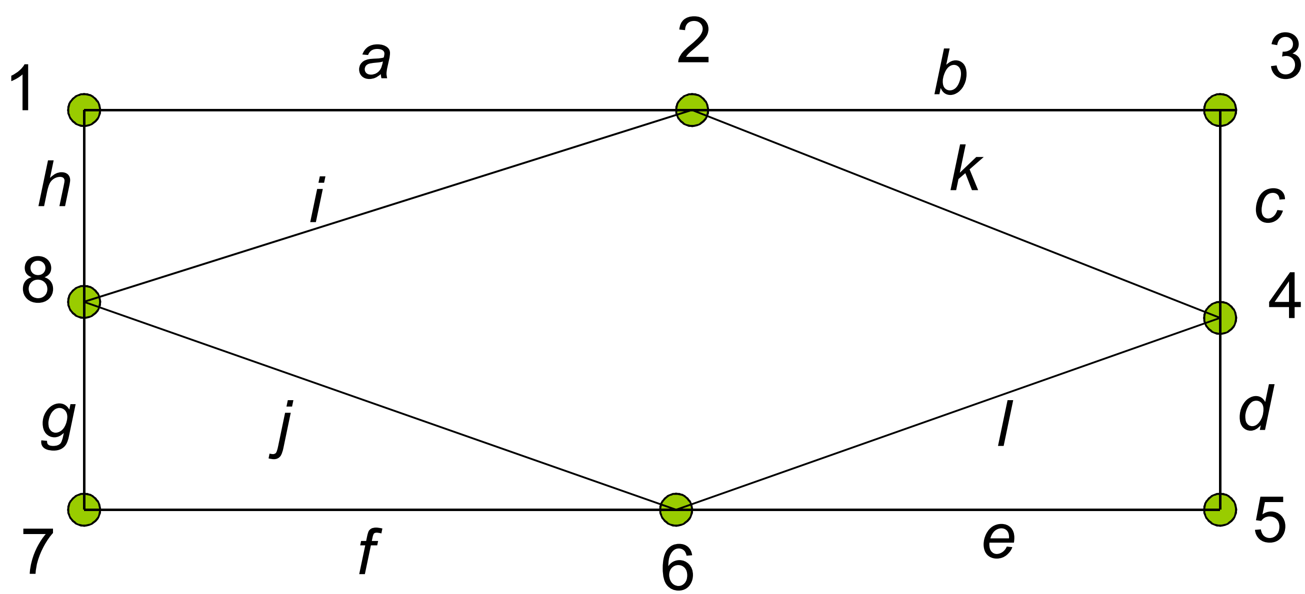
| Vertex | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|
| Degree | 2 | 3 | 3 | 2 | 2 | 2 |




This graph has an Euler trail

# exercise

- Decide whether the graph has an Euler circuit. If the graph has an Euler circuit, exhibit one.

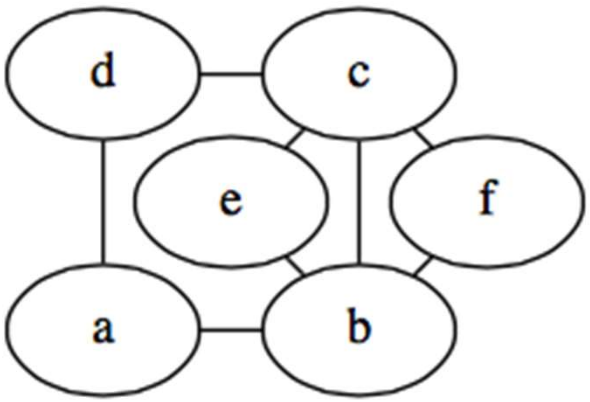



prepared by Razana Alwee

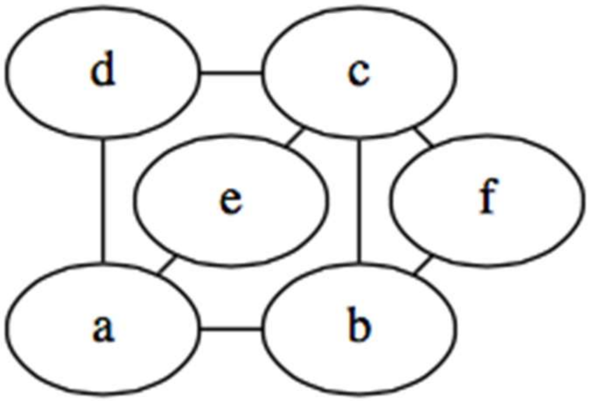
# exercise

Q: Which of the following graphs has Euler circuit? Justify your answer.

**$G_1$**




**$G_2$**



# Exercise
# Past Year 2015/2016

Determine whether the graph in Figure 3 has an Euler cycle or Euler path. If the graph has an Euler cycle or Euler path, exhibit one; otherwise, give an argument that shows there is no Euler path. (4 marks)

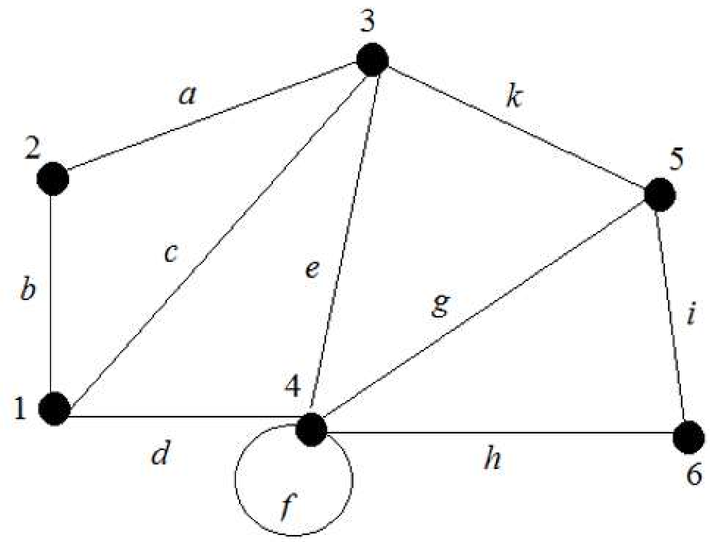



**Figure 3**

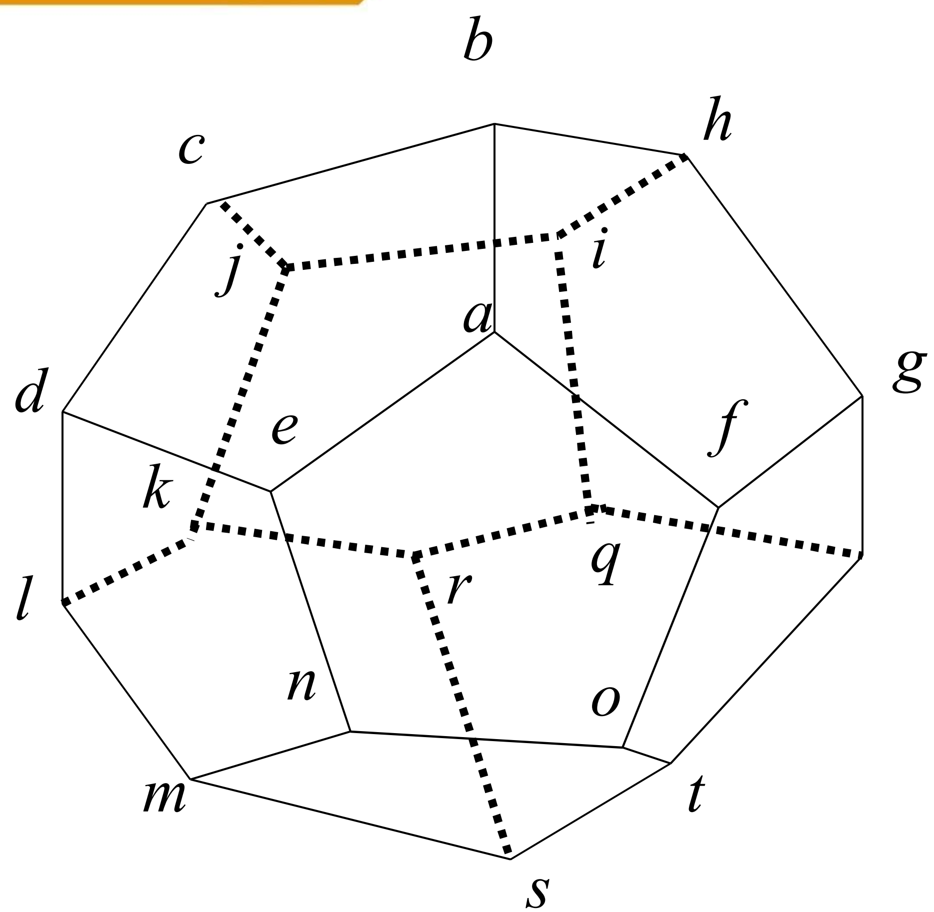
# Hamilton Circuits

# Hamiltonian Circuit

- A circuit in a graph *G* is called a Hamiltonian circuit if it contains each vertex of *G*.

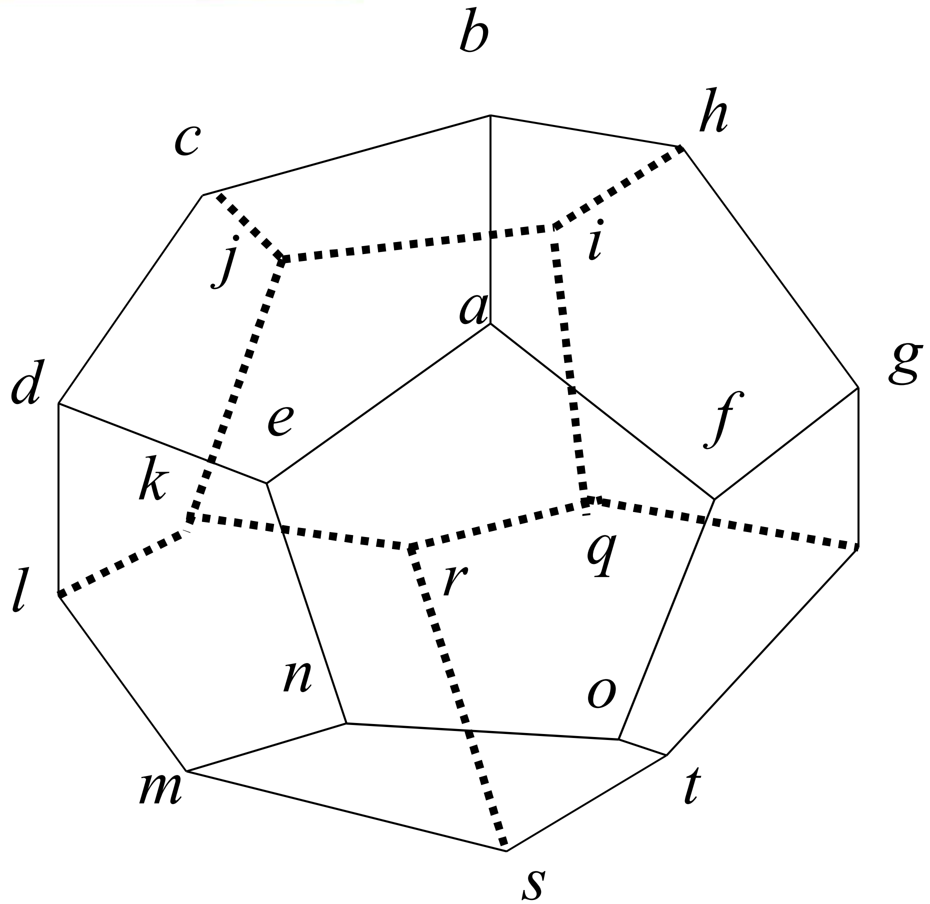
- Given a graph G, a Hamiltonian circuit for G is **a simple circuit that includes every vertex of G** (but doesn't need to include all edges). That is, a Hamiltonian circuit for G is a sequence of adjacent vertices and distinct edges in which every vertex of G appears exactly once, except for the first and the last, which are the same.

# Around the world game




Sir William Rowan Hamilton marketed a puzzle in the mid-1800s in the form of dedocahedron

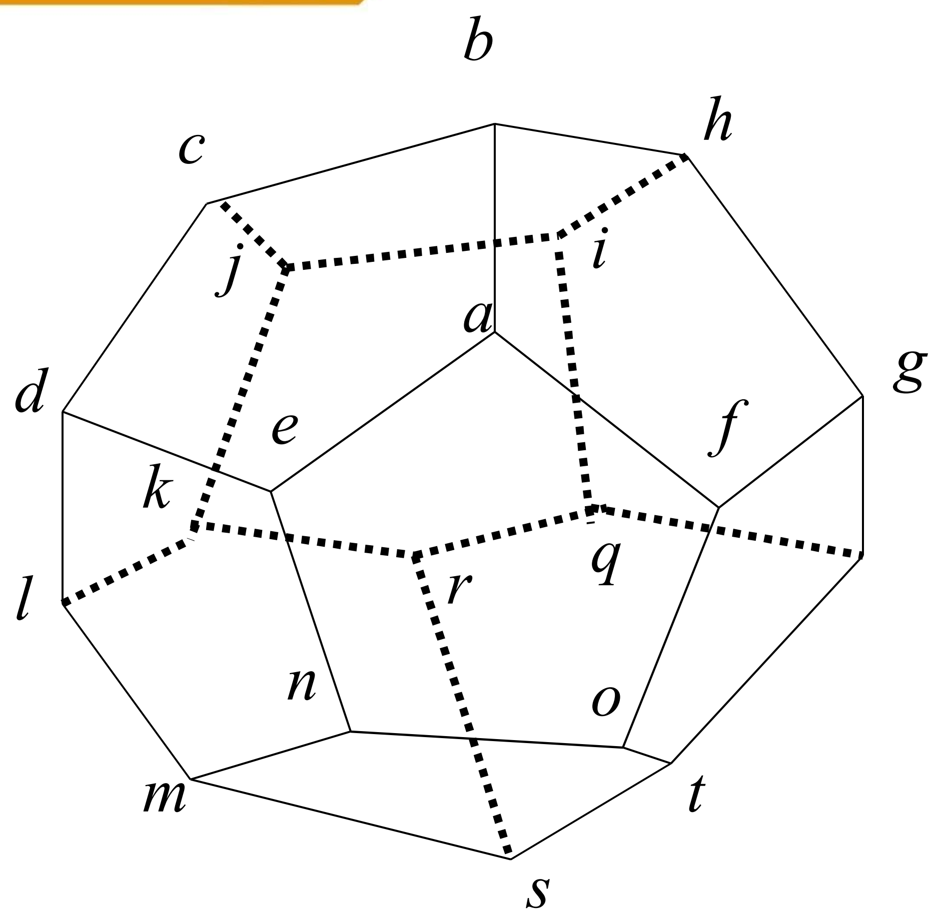
# Around the world game




Each corner bore the name of a city

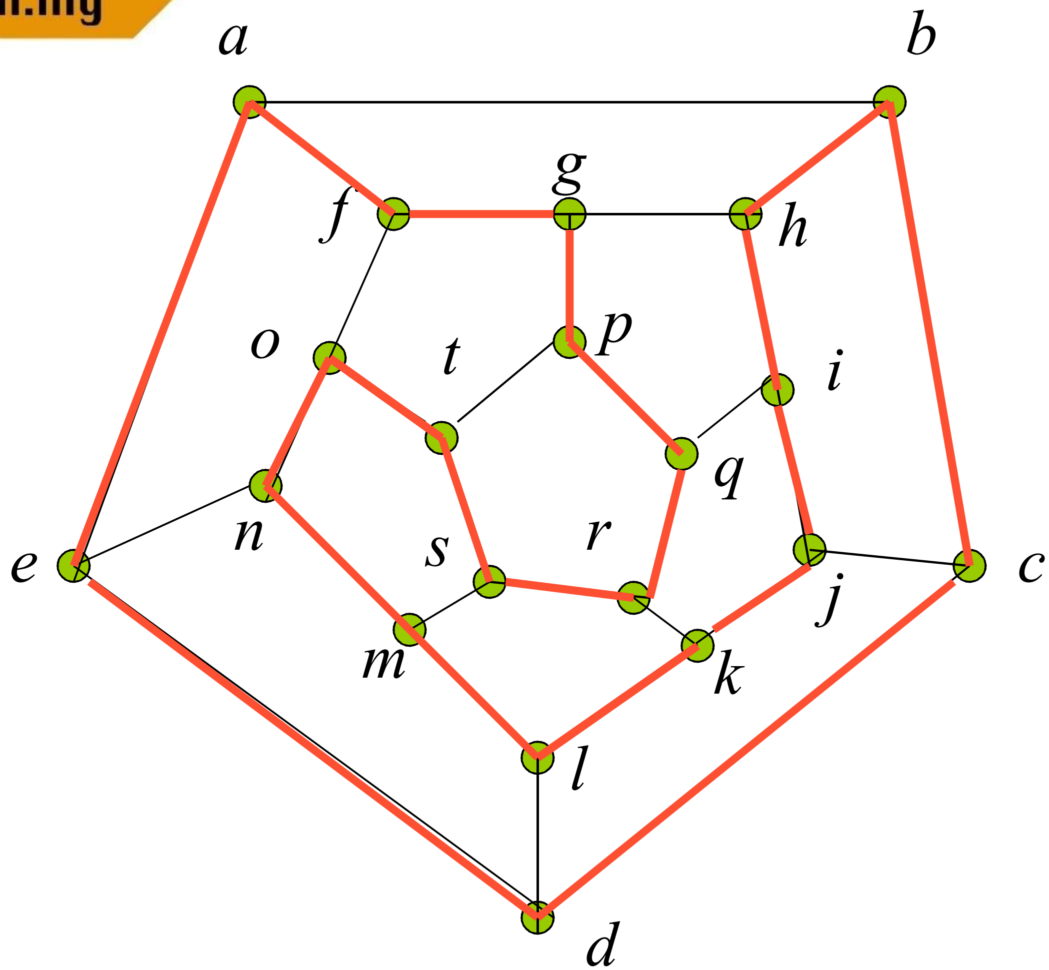
prepared by Razana Alwee

# Around the world game



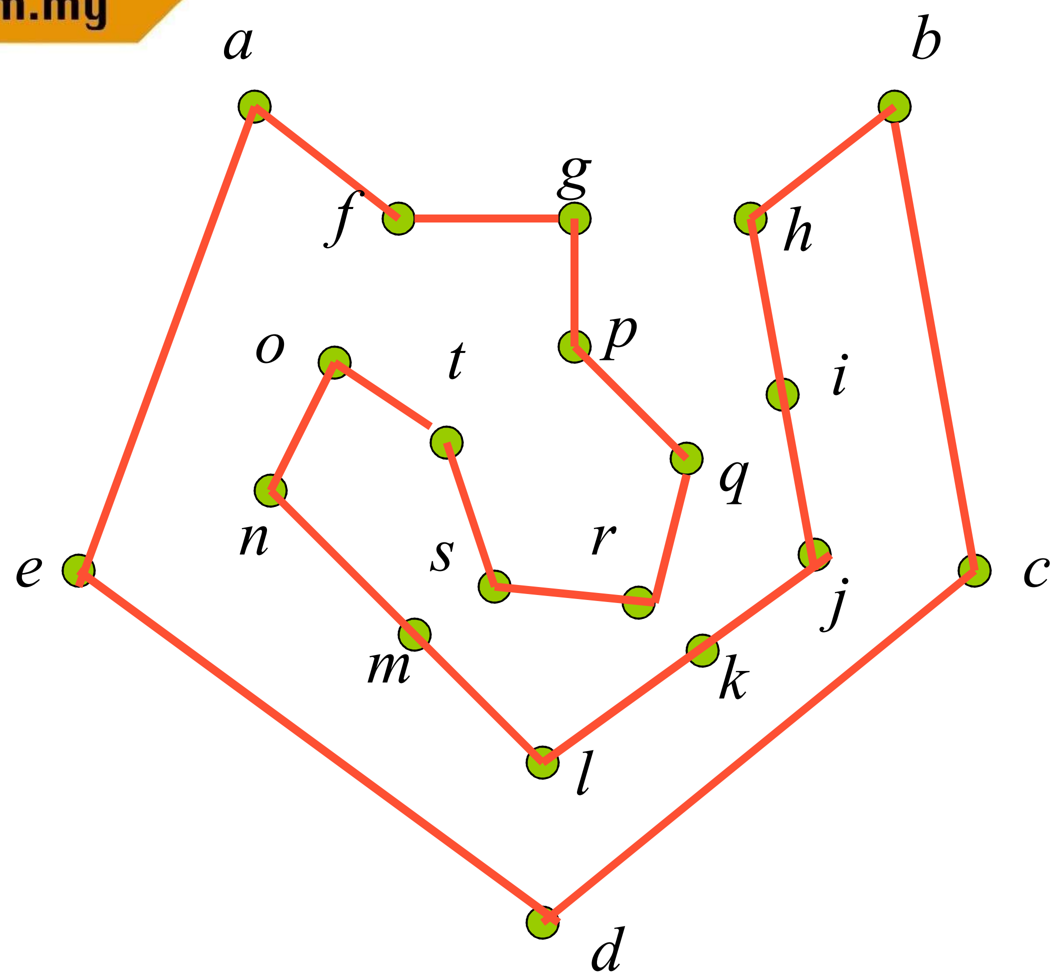

The problem was to start at any city, travel along the edges, visit each city exactly one time and return to the initial city

# The graph




prepared by Razana Alwee

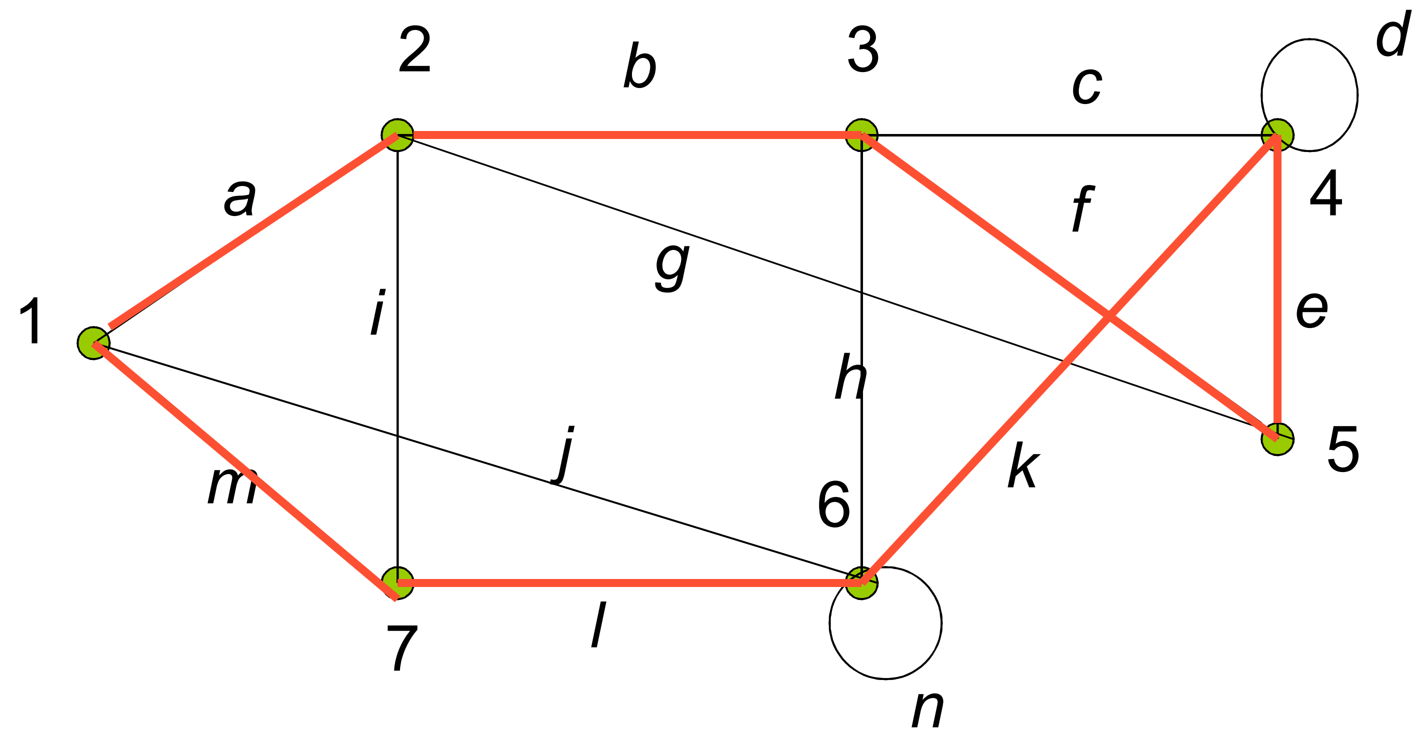
# Hamiltonian Circuit




**a-f-g-p-q-r-s-t-o-n-m-l-k-j-i-h-b-c-d-e-a**

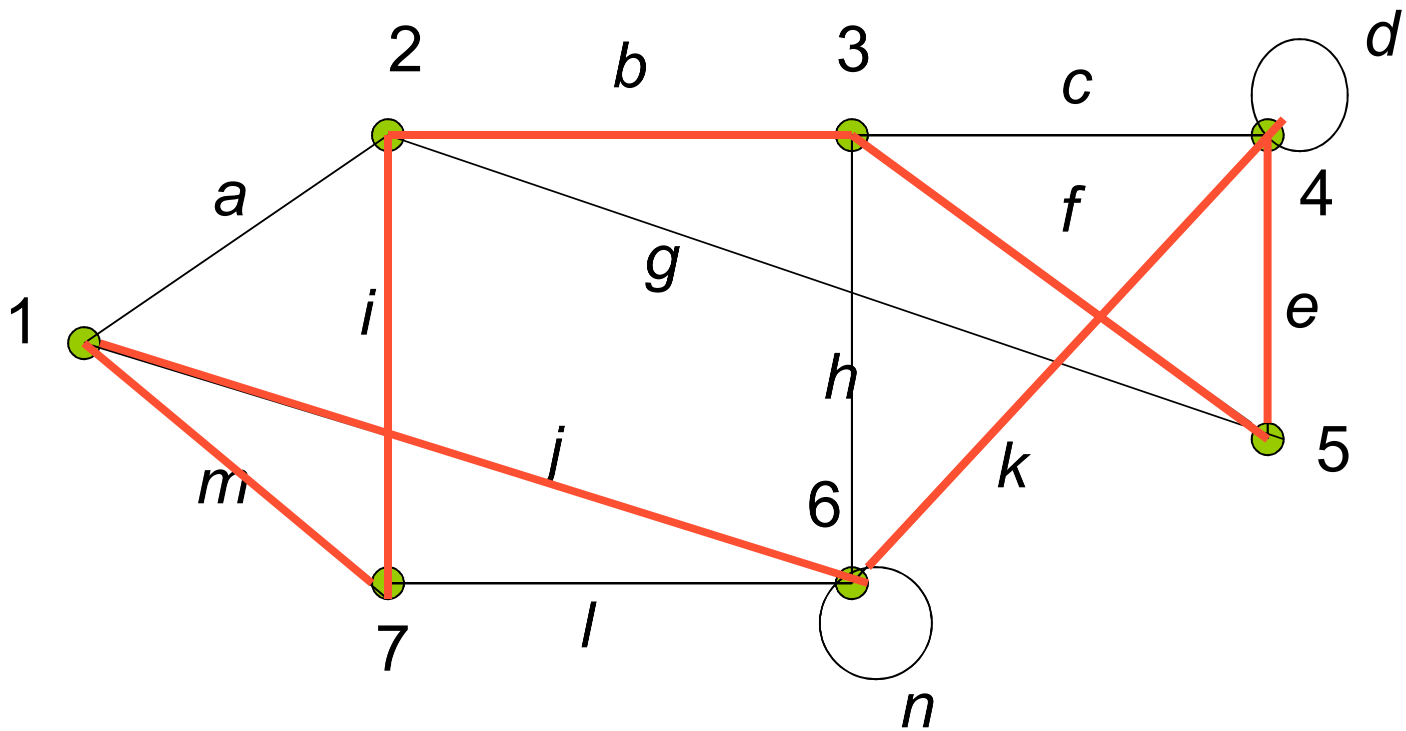
# example

This graph has a Hamiltonian circuit




**1-a-2-b-3-f-5-e-4-k-6-l-7-m-1**

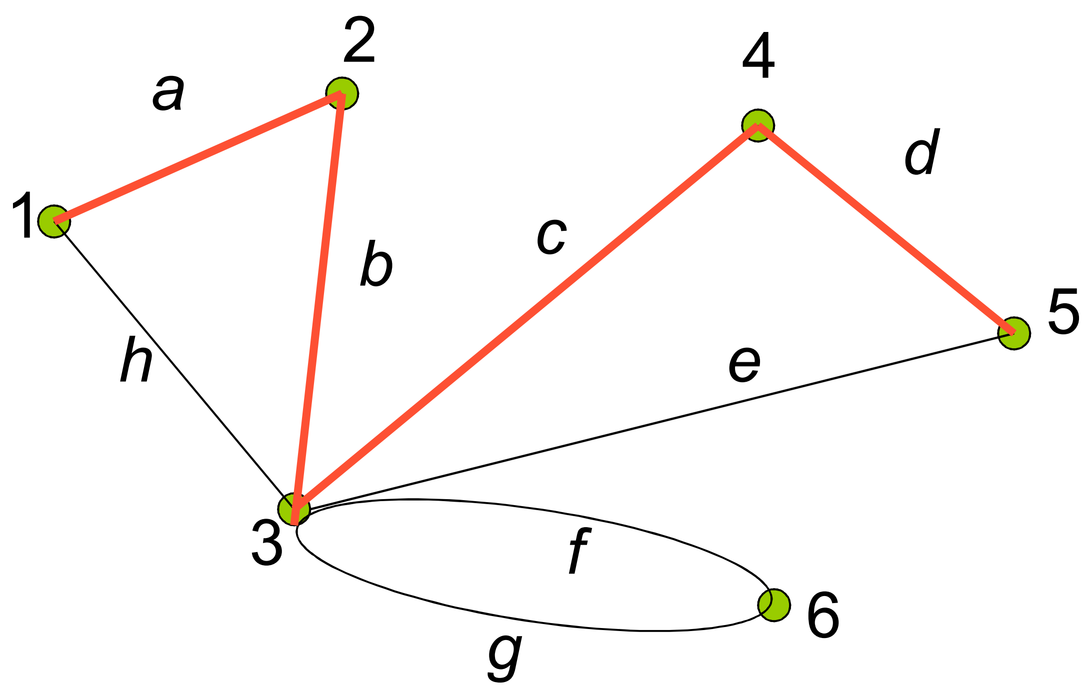
# example




**1-j-6-k-4-e-5-f-3-b-2-i-7-m-1**

prepared by Razana Alwee

# example

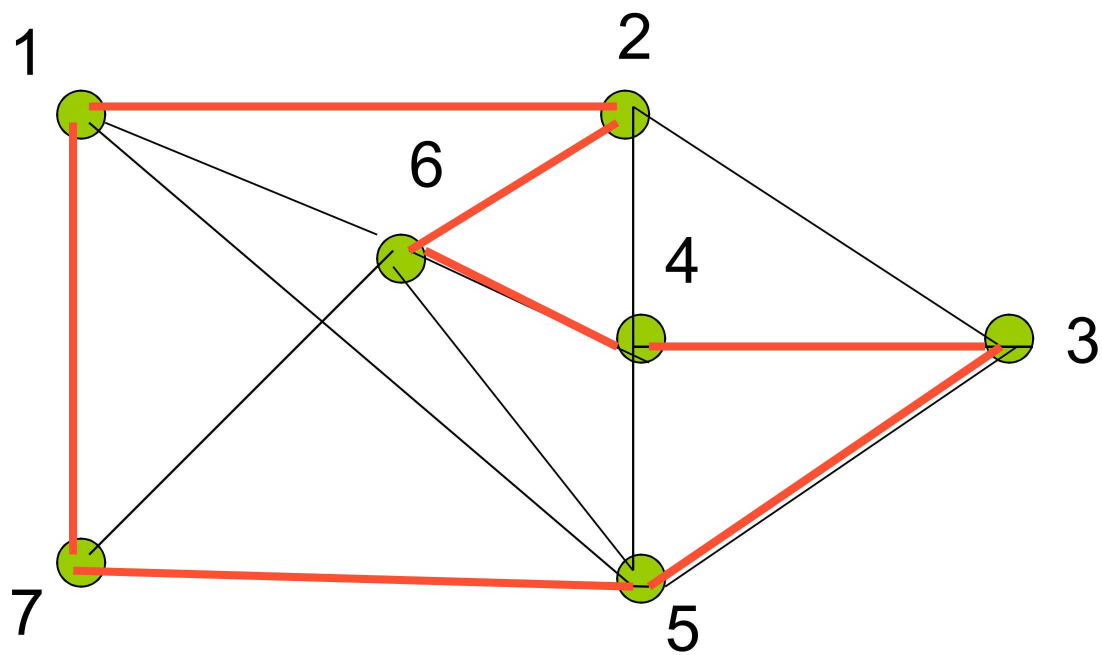



no Hamiltonion circuit

prepared by Razana Alwee

# example

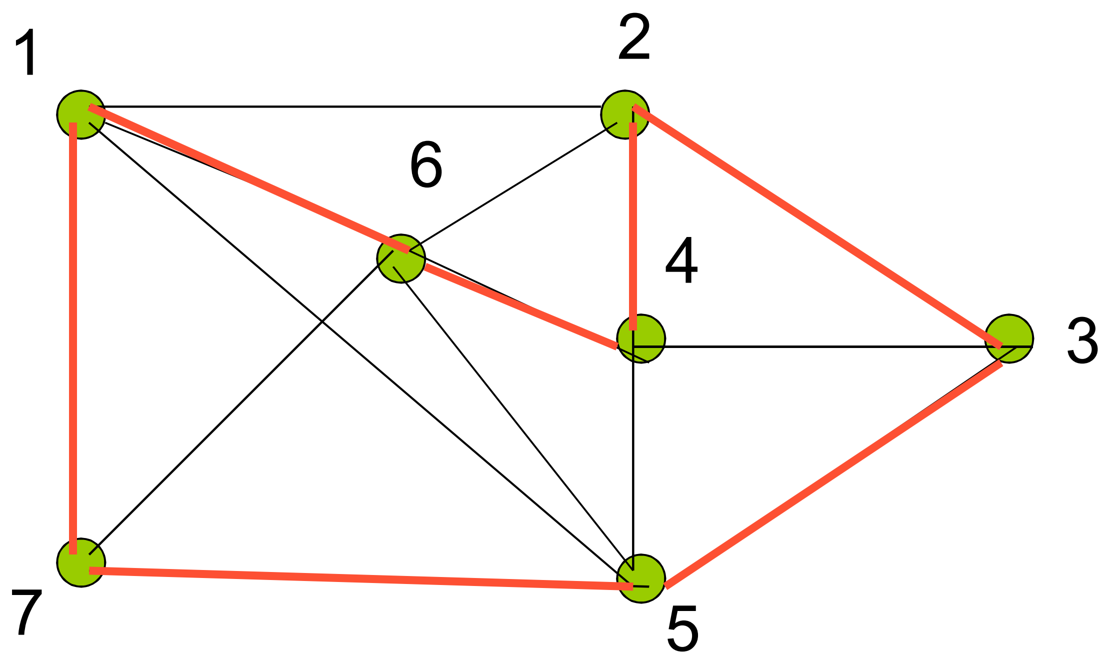
This graph has a Hamiltonian circuit




**1-2-6-4-3-5-7-1**

prepared by Razana Alwee

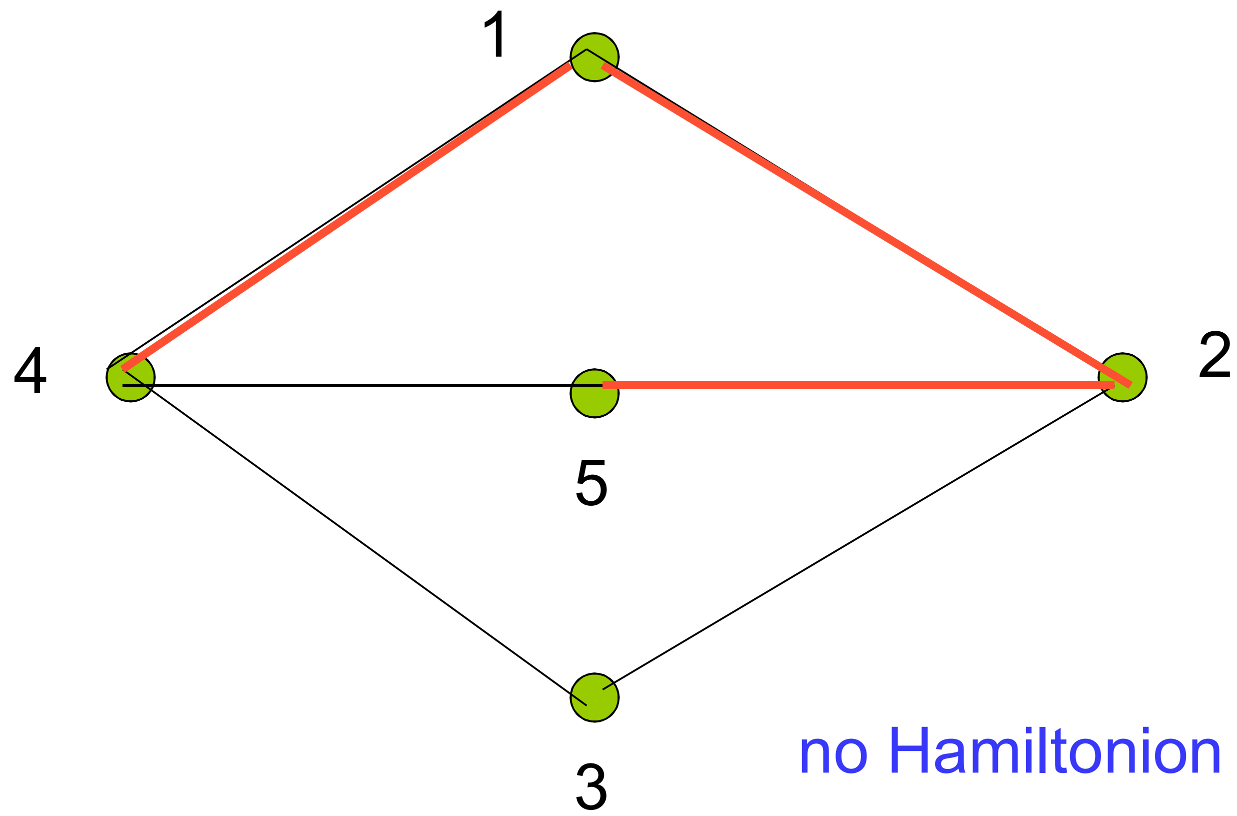
# example




**1-6-4-2-3-5-7-1**

prepared by Razana Alwee

# example

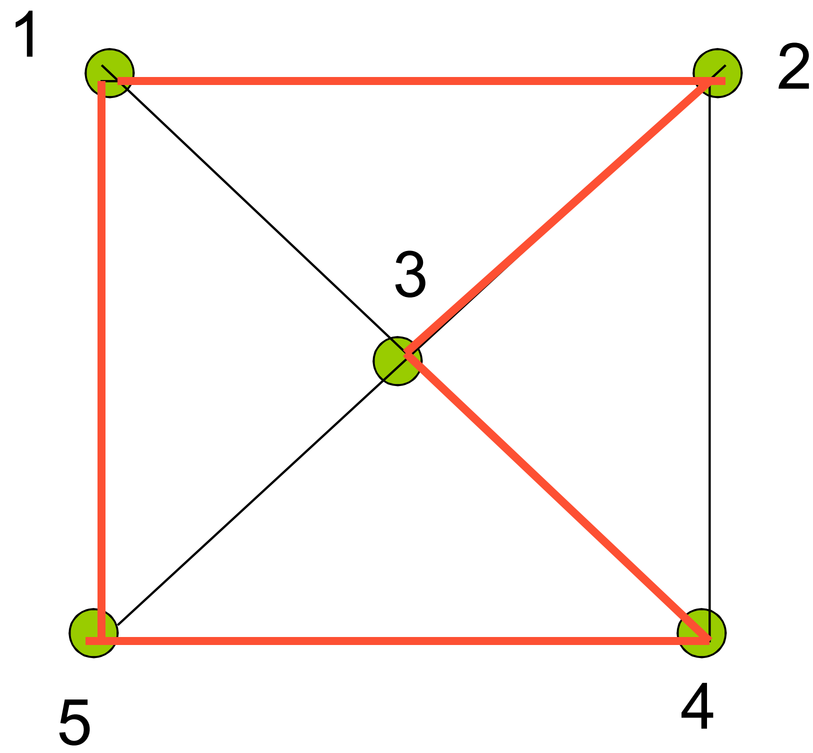



no Hamiltonion circuit

prepared by Razana Alwee

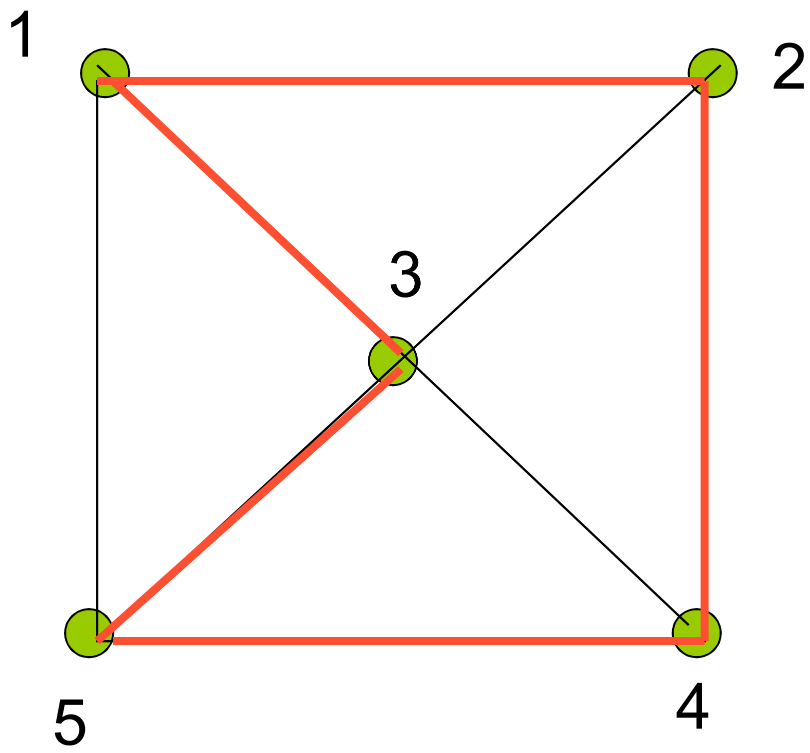
# example

This graph has a Hamiltonian circuit




**1-2-3-4-5-1**

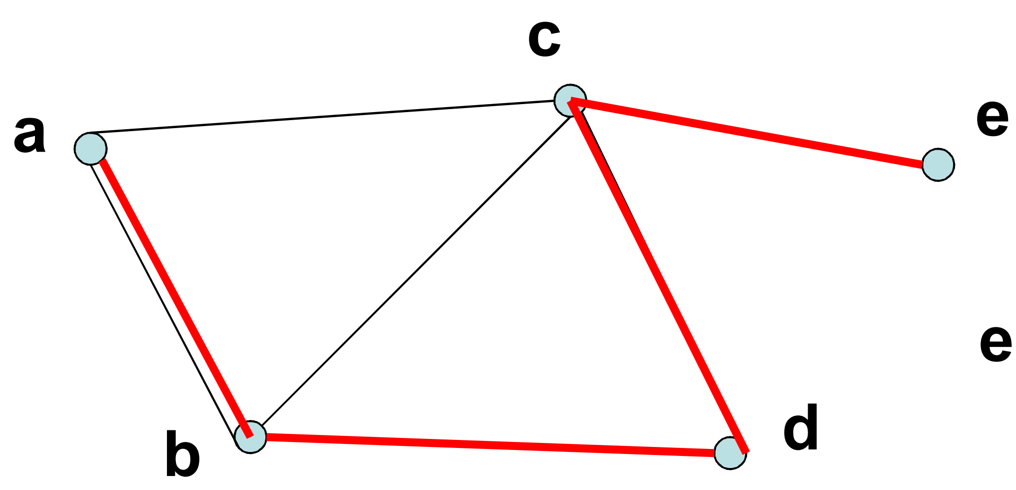
# example




**3-5-4-2-1-3**

# Hamiltonian Path

- A path in a graph *G* is called a Hamiltonian path if it contains each vertex of *G.*
- Example:

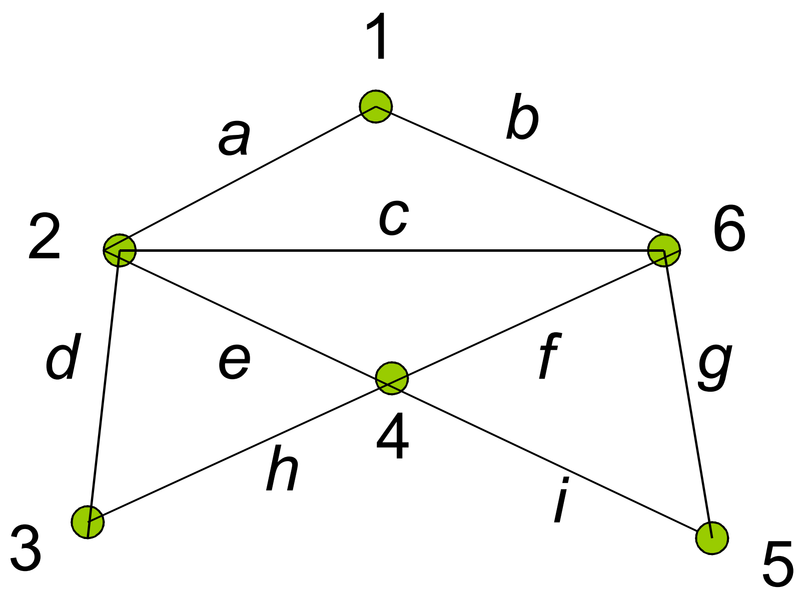



**e-c-d-b-a**

prepared by Razana Alwee

# exercise

- Find a Hamiltonian circuit in this graph.




prepared by Razana Alwee

# Exercise
# Past Year 2015/2016

Determine whether the graph in Figure 4 has an Hamiltonian cycle. If yes, exhibit one.

(3 marks)

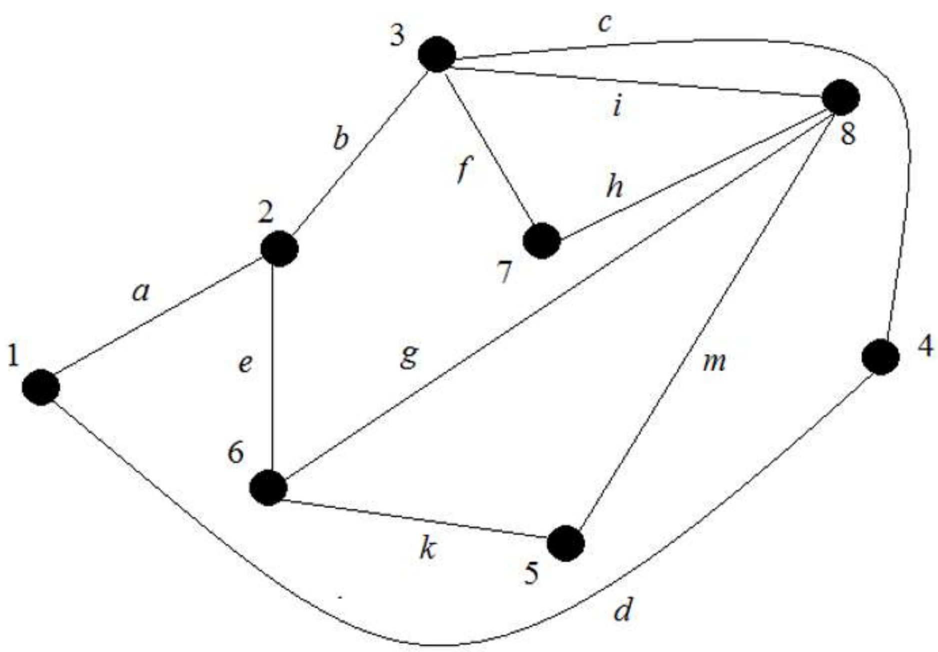



**Figure 4**

# Shortest Path Problem

# Shortest Path

- **Let *G* be a weighted graph.**
- **Let *u* and *v* be two vertices in *G*, and let *P* be a path in *G* from *u* to *v*.**
- **The length of path *P*, written *L*(*P*), is the sum of the weights of all the edges on path *P*.**
- **A shortest path from a vertex to another vertex is a path with the shortest length between the vertices.**

# Dijkstra's Shortest Path Algorithm

- 1. $S := \emptyset$
- 2. $N := V$
- 3. For all vertices, $u \in V$, $u \neq a$, $L(u) := \infty$
- 4. $L(a) := 0$
- 5. While $z \notin S$ do,

  5.a :Let $v \in N$ be such that

  $L(v) = \min\{L(u) \mid u \in N\}$

  5.b : $S := S \in \{v\}$

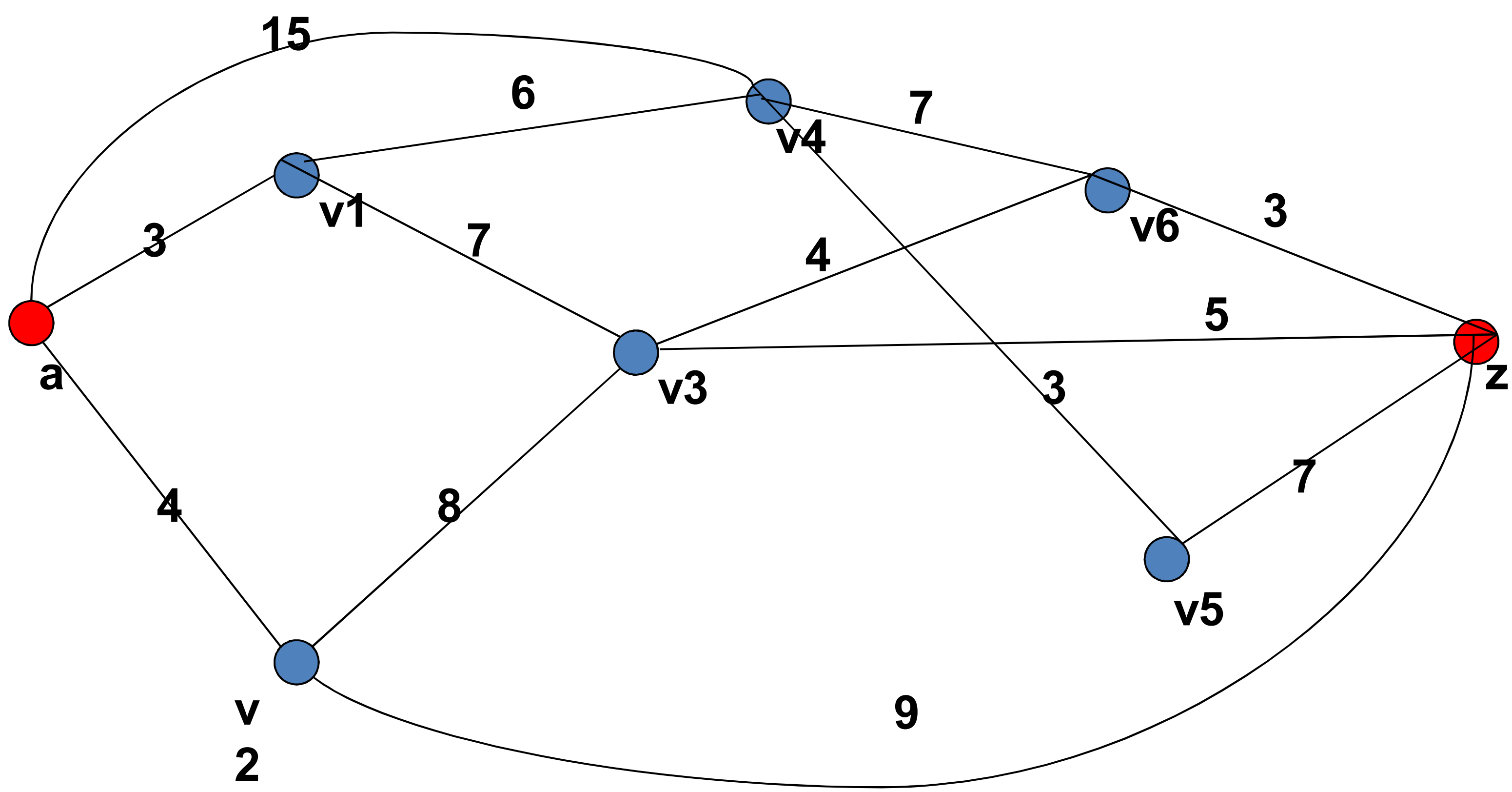
  5.c : $N := N - \{v\}$

  5.d : For all $w \in N$ such that there is an edge from $v$ to $w$

  5.d.1: If $L(v) + W[v,w] < L(w)$ then $L(w) = L(v) + W[v,w]$

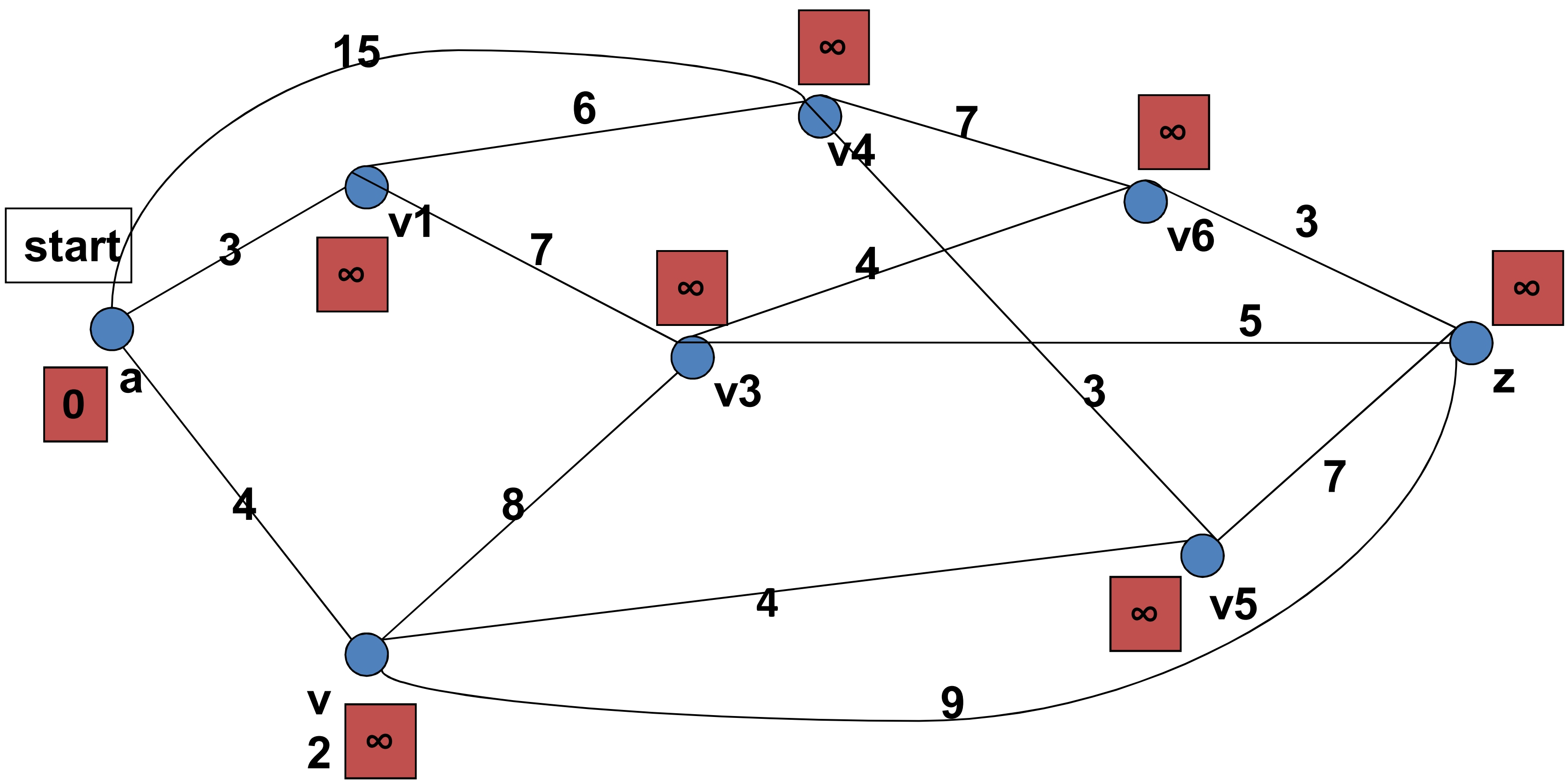
# Example

## What is the shortest path from a to z?



$S = \emptyset$

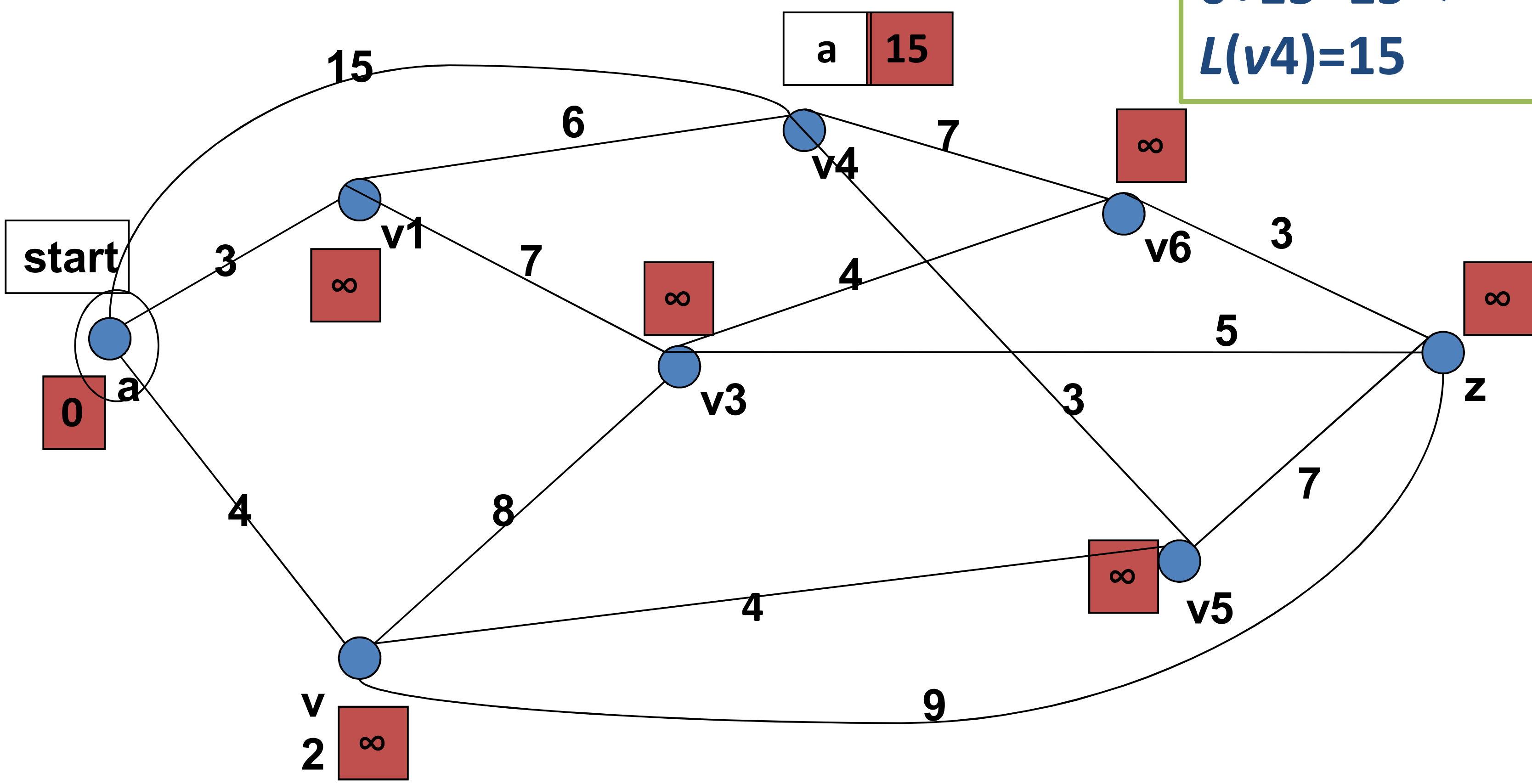
$N = \{a, v1, v2, v3, v4, v5, v6, z\}$



$S=\{a\}$

$N= \{v1,v2,v3,v4,v5,v6,z\}$

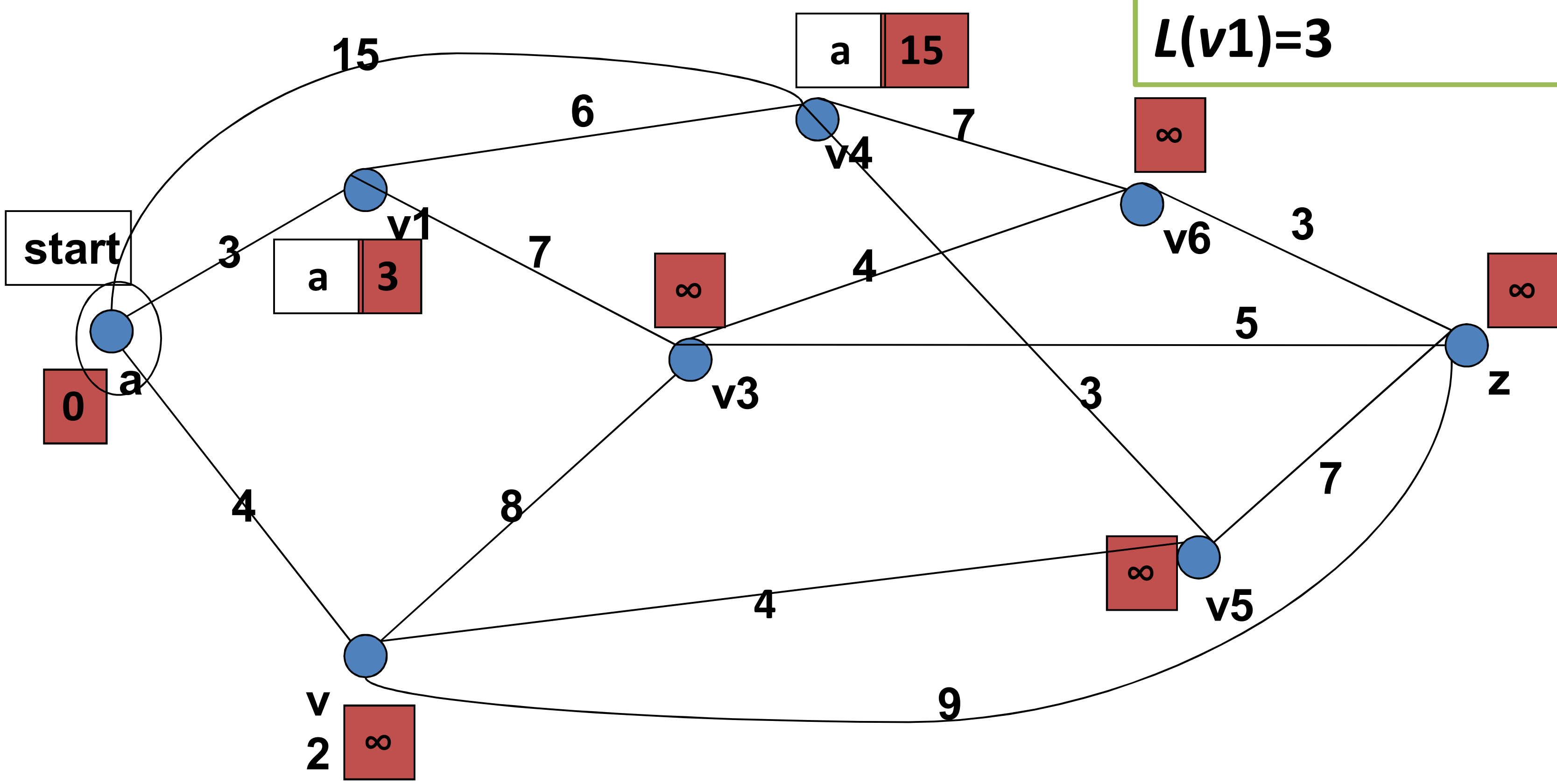
**$L(a)+W[a,v4]$**
**$<L(v4)$**
**$0+15=15 < \infty$**
**$L(v4)=15$**



# $S=\{a\}$

# $N= \{v1,v2,v3,v4,v5,v6,z\}$

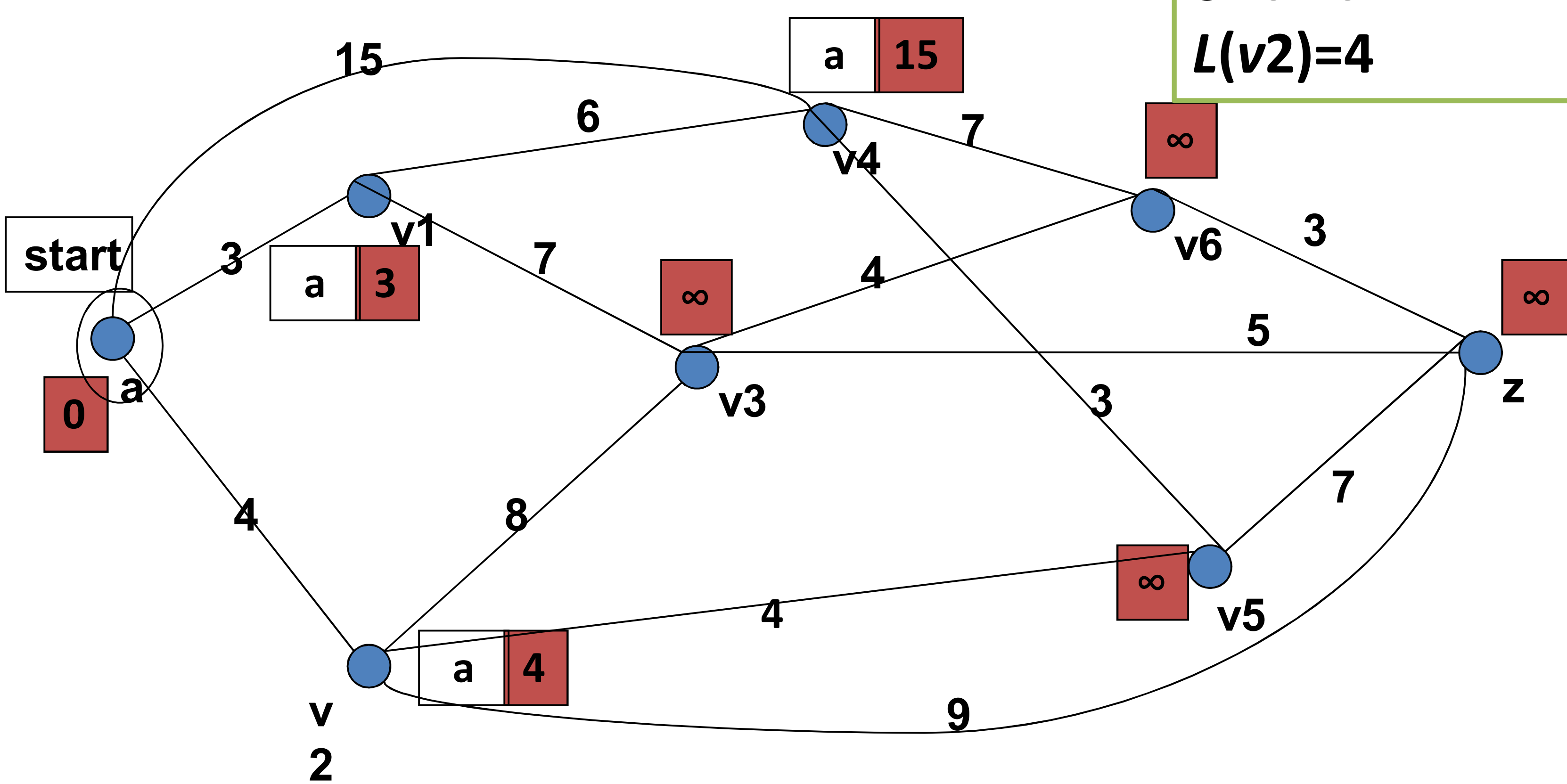
$L(a)+W[a,v1]$
$<L(v1)$
$0+3= 3 < \infty$
$L(v1)=3$



$S=\{a\}$

$N= \{v1,v2,v3,v4,v5,v6,z\}$

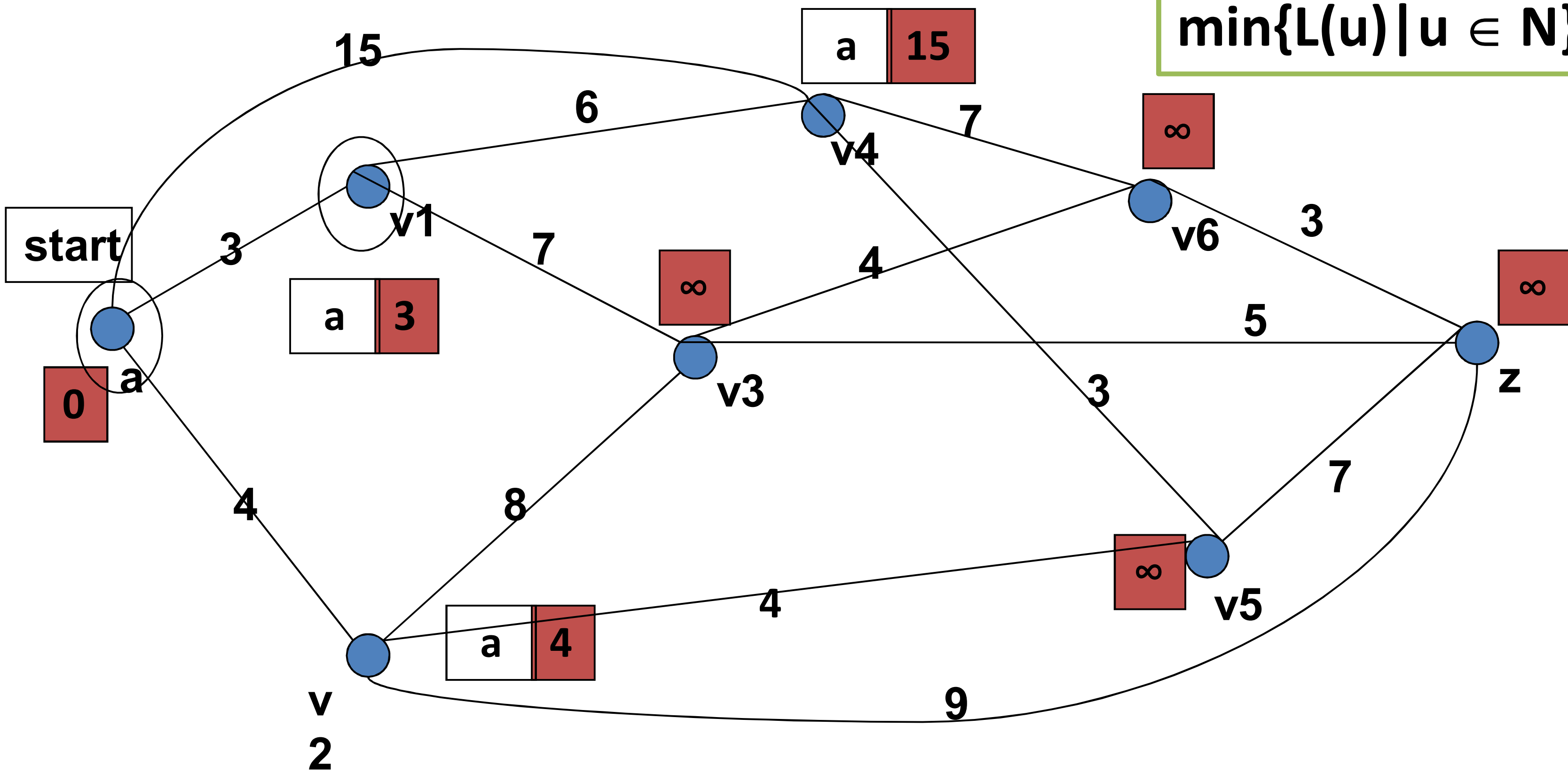
$L(a)+W[a,v2]$
$<L(v2)$
$0+4= 4 < \infty$
$L(v2)=4$



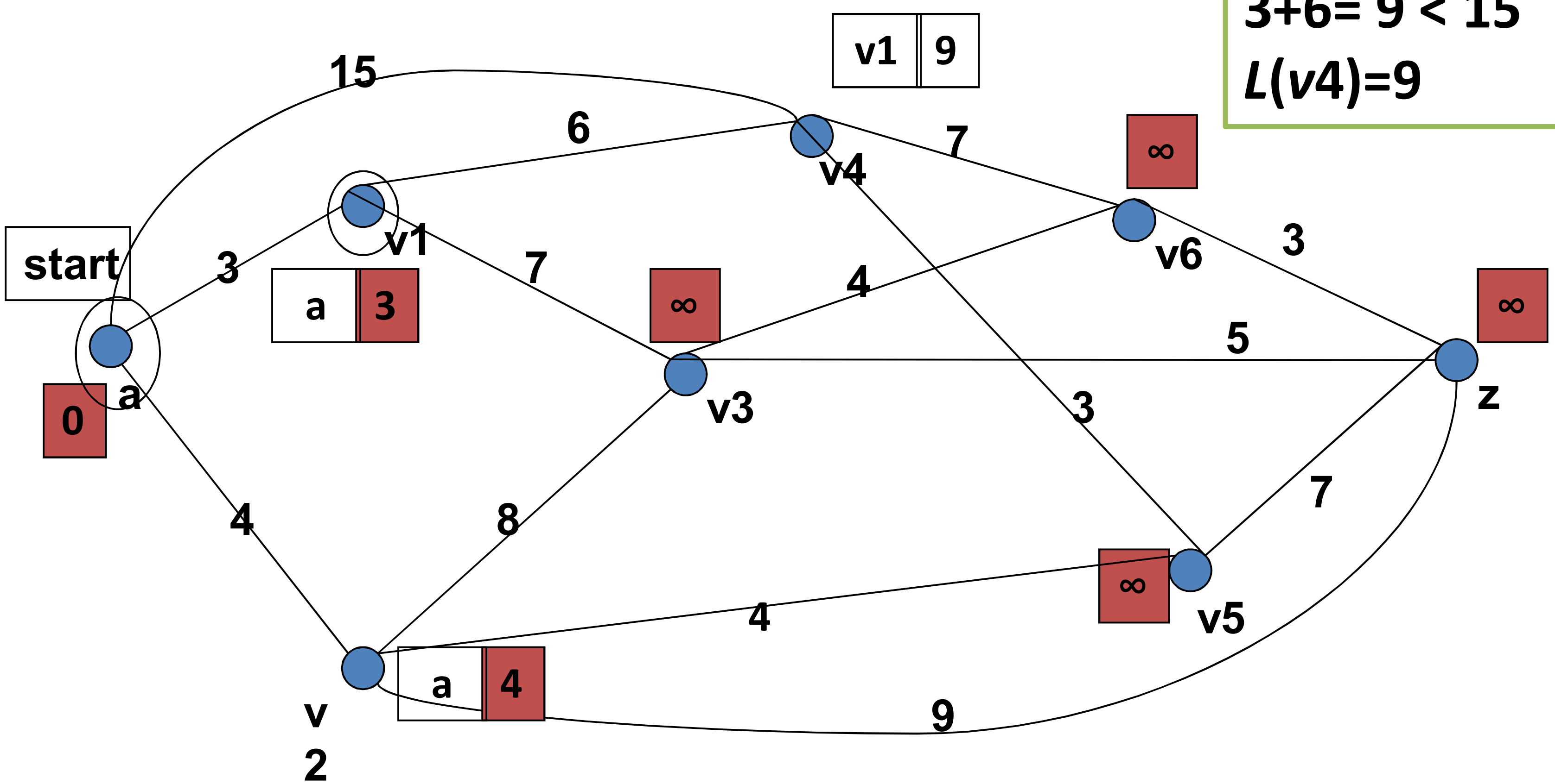
# *S*={*a*}

# *N*= {*v*1,*v*2,*v*3,*v*4,*v*5,*v*6,*z*}

**choose *v*1**
**because**
***L*(*v*1)=3 =**
**min{L(u)|u $\in$ N}**



# *S={a, v1}*
# *N= {v2,v3,v4,v5,v6,z}*



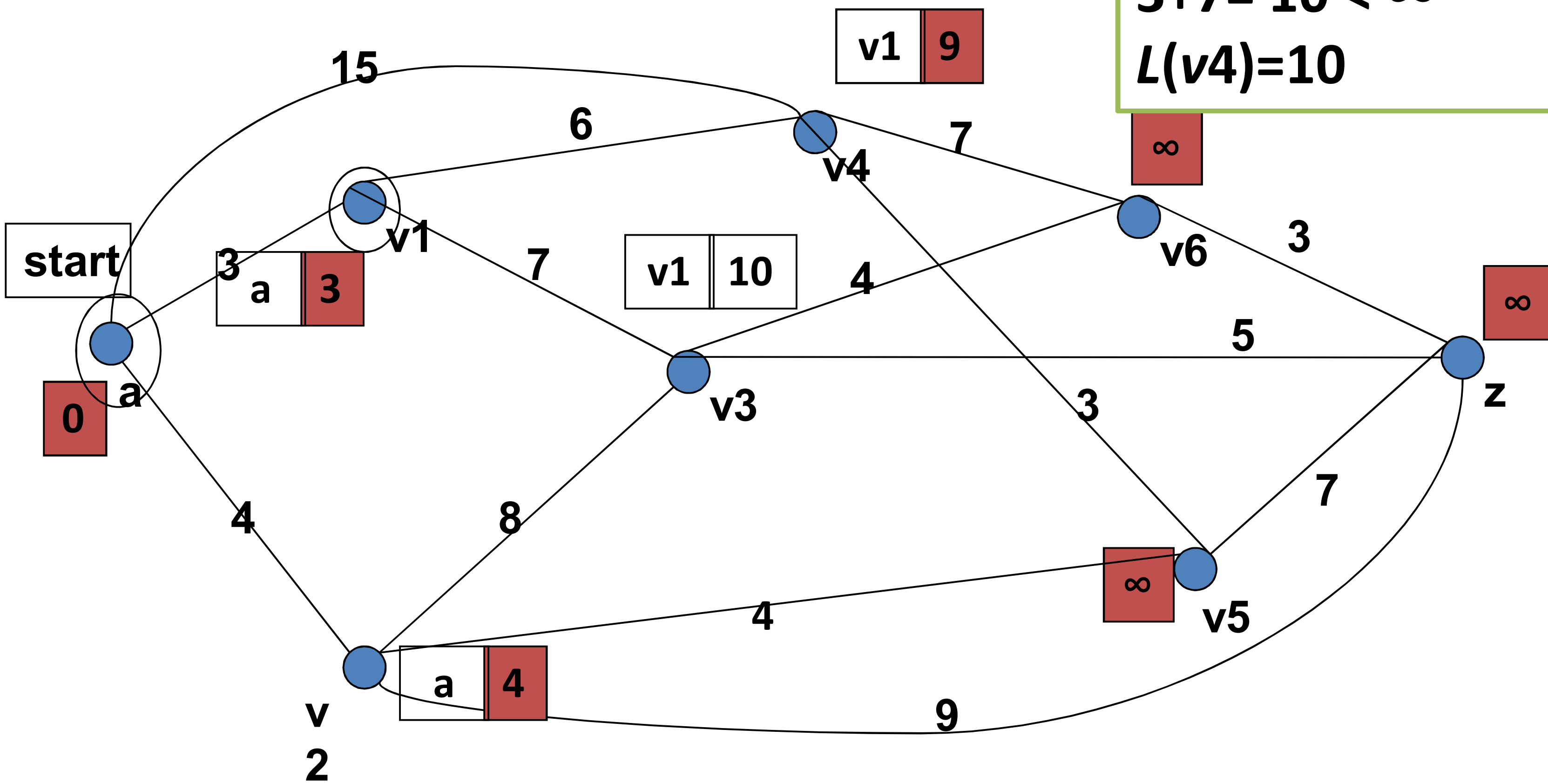

*L*(*v*1)+*W*[*v*1,*v*4]< *L*(*v*4)

3+6= 9 < 15

*L*(*v*4)=9

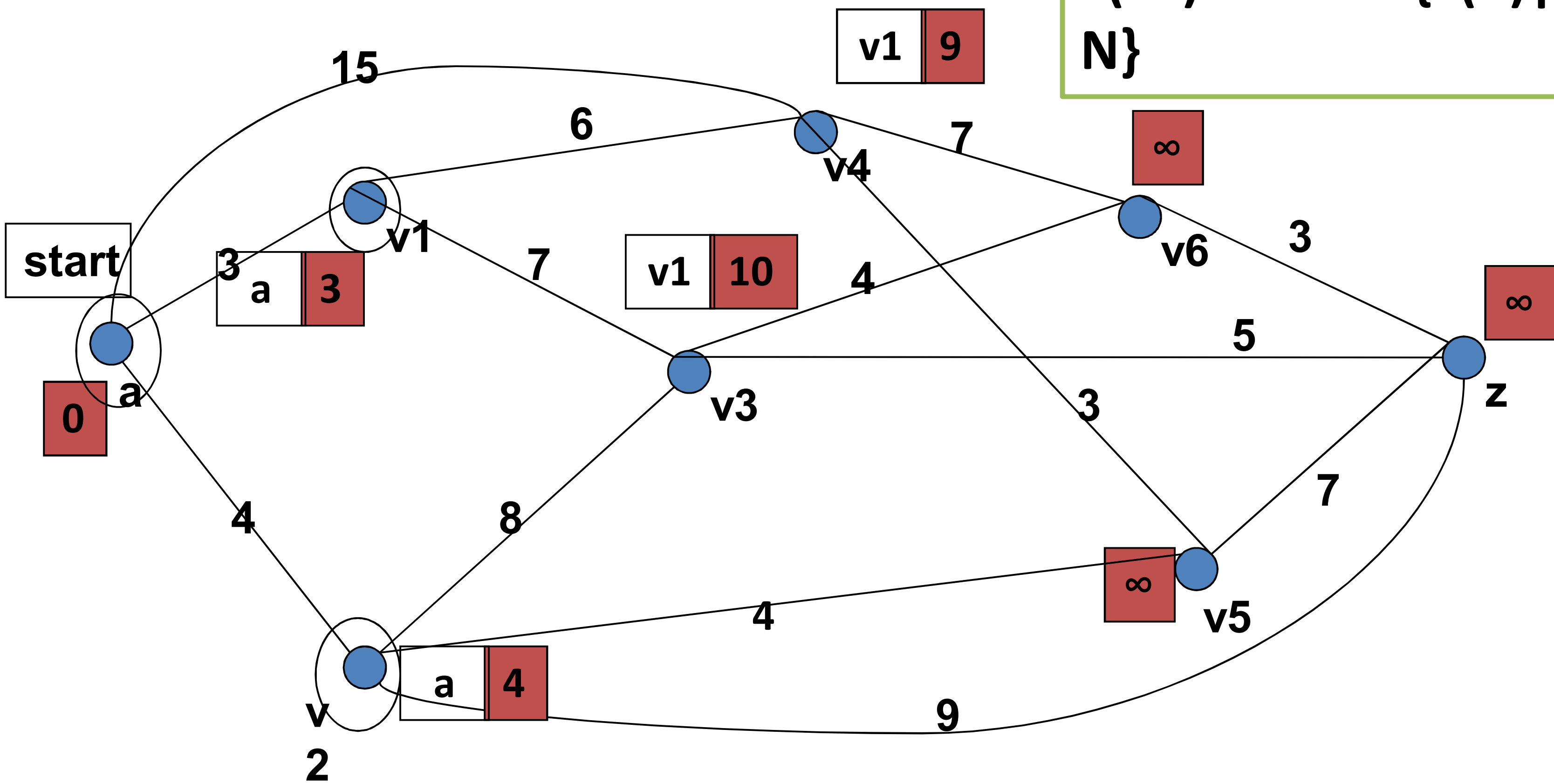
# *S={a, v1}*
# *N= {v2,v3,v4,v5,v6,z}*

*L*(*v*1)+*W*[*v*1,*v*3]<*L*(*v*3)
3+7= 10 < ∞
*L*(*v*4)=10



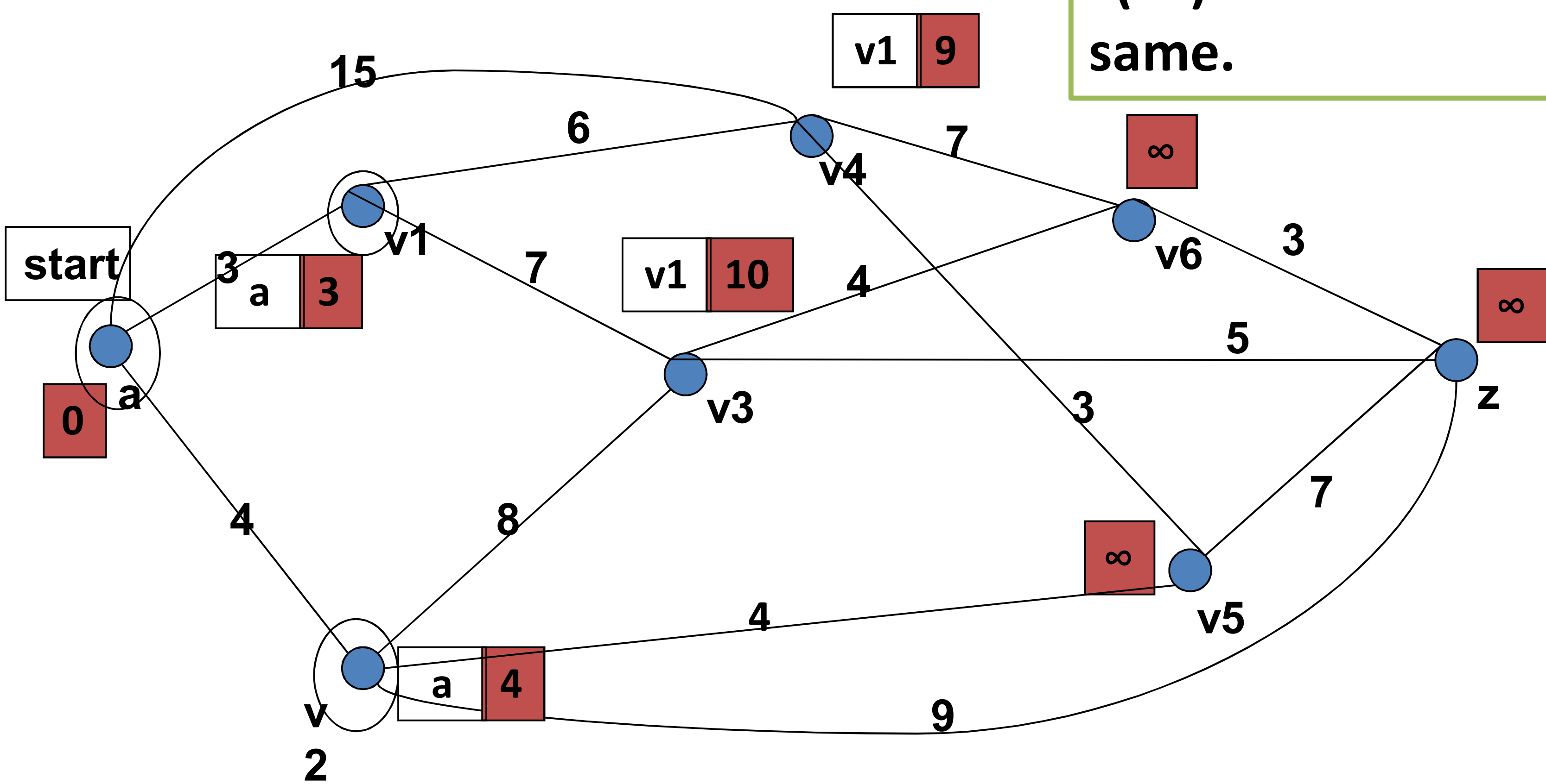
# *S={a, v1}*
# *N= {v2,v3,v4,v5,v6,z}*

choose *v2* because *L*(*v2*)=4 = min{L(u)|u ∈ N}



# *S*={*a*, *v1*, *v2*}
# *N*= {*v3*,*v4*,*v5*,*v6*,*z*}

**$L(v2)+W[v2,v3]<L(v3)$**
**4+8= 12 > 10**
**$L(v3)$ remains the same.**

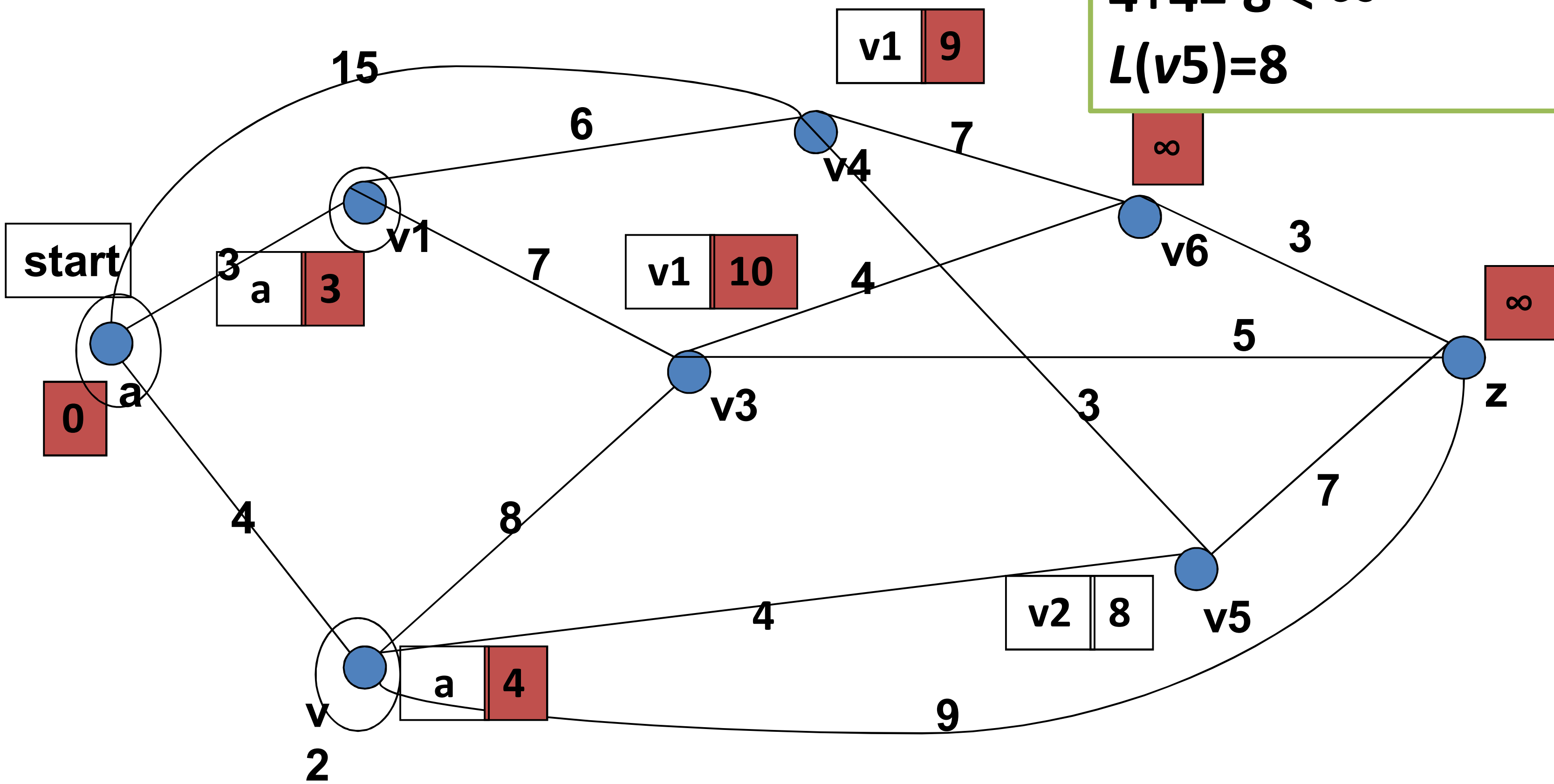


# $S=\{a, v1, v2\}$
# $N= \{v3,v4,v5,v6,z\}$

$L(v2)+W[v2,v5]<L(v5)$

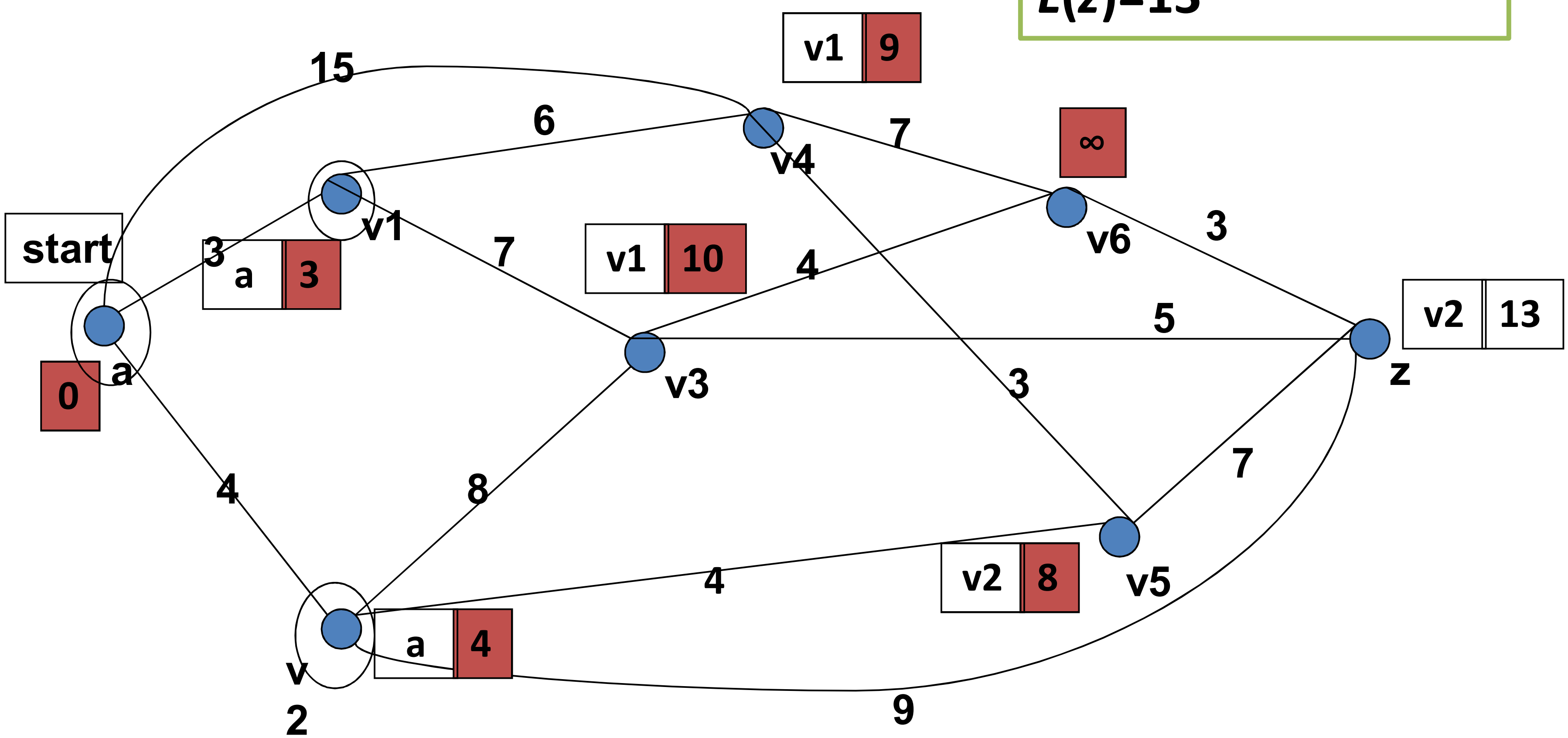
$4+4= 8 < \infty$

$L(v5)=8$



# $S=\{a, v1, v2\}$
# $N= \{v3,v4,v5,v6,z\}$

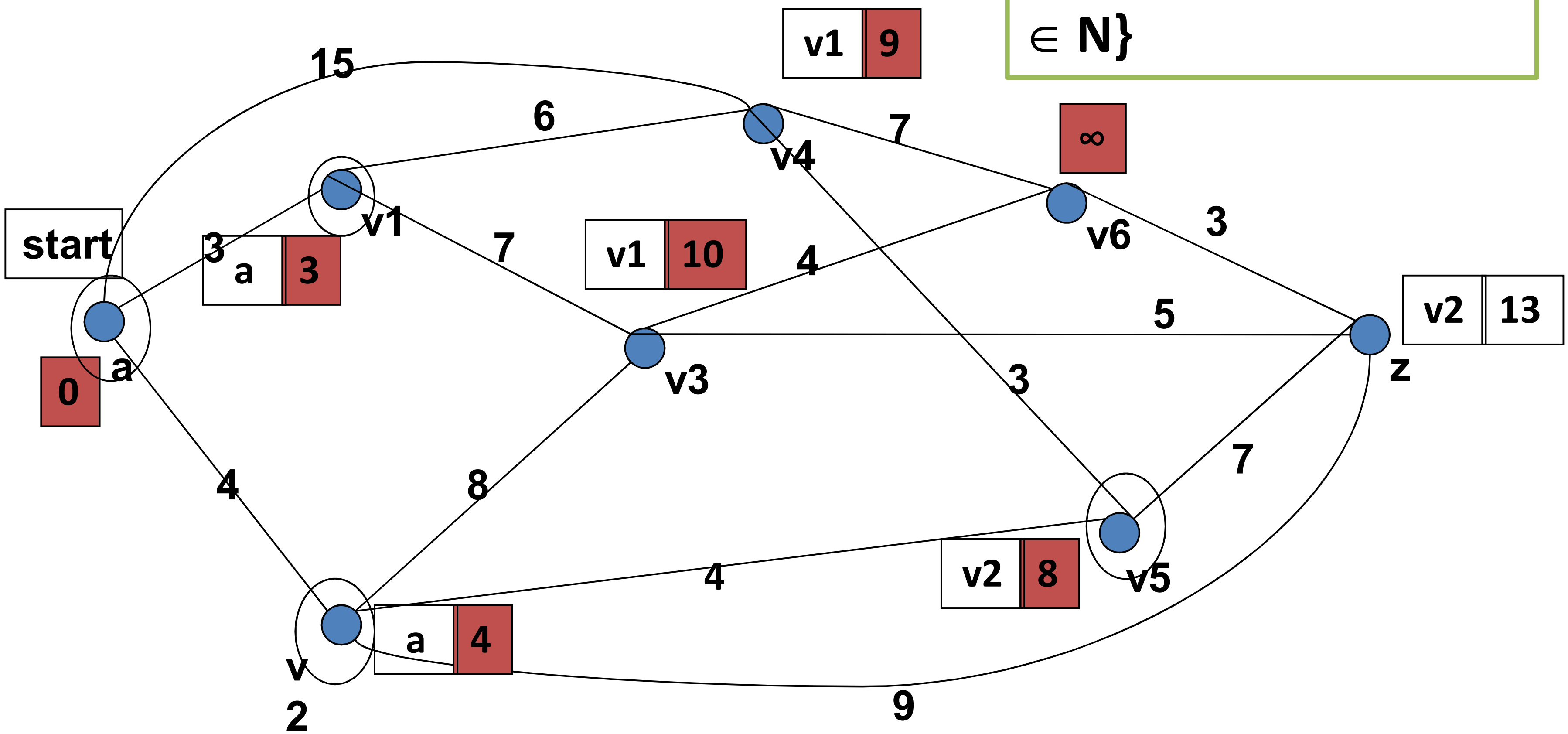
$L(v2)+W[v2, z]<L(z)$
$4+9= 13 < \infty$
$L(z)=13$



# $S=\{a, v1, v2\}$

# $N=\{v3,v4,v5,v6,z\}$

choose *v*5 because $L(v5)=8 = \min\{L(u) \mid u \in N\}$



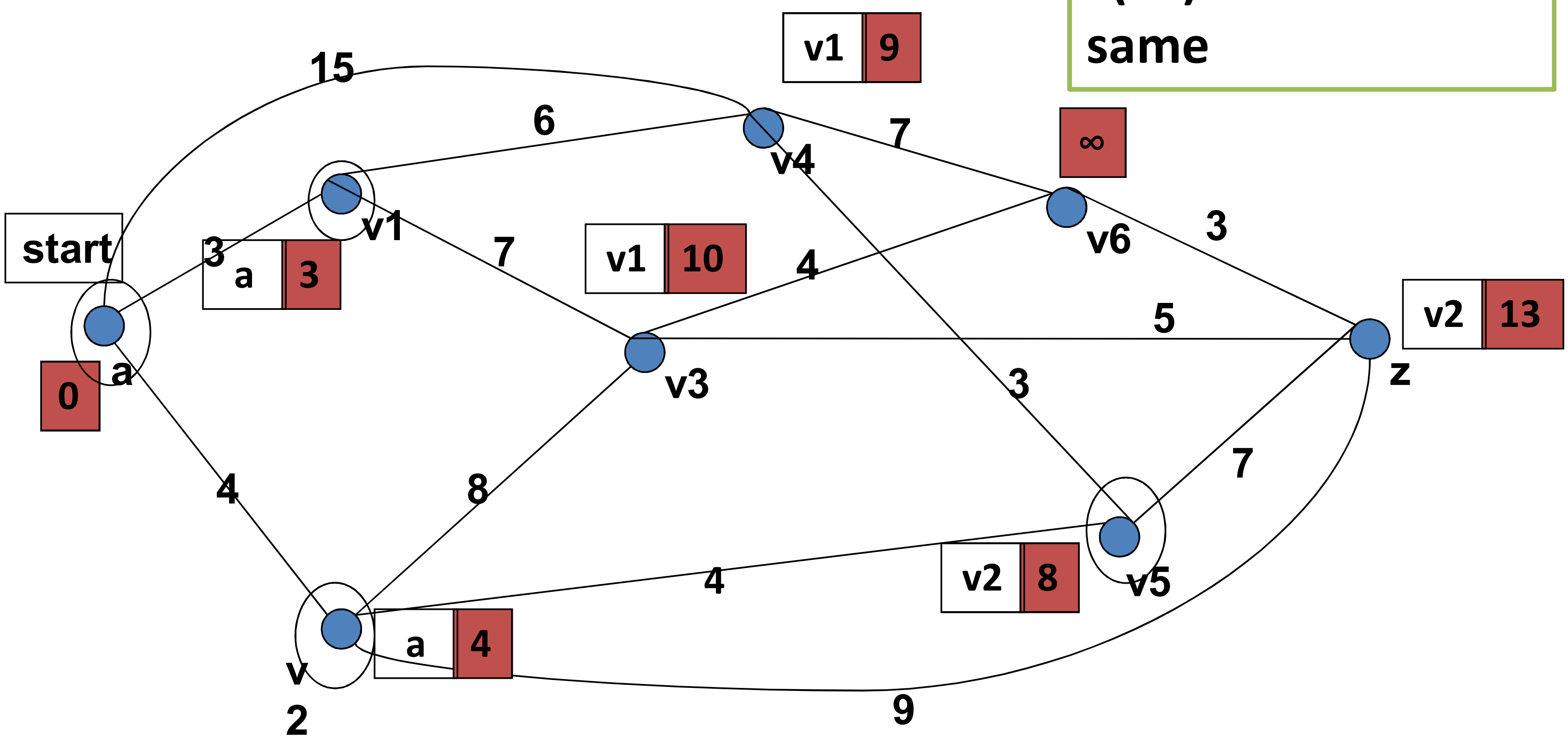
# $S=\{a, v1, v2, v5\}$

# $N=\{v3, v4, v6, z\}$

$L(v5)+W[v5, v4]<L(v4)$

8+3= 11 > 9

$L(v4)$ remains the same



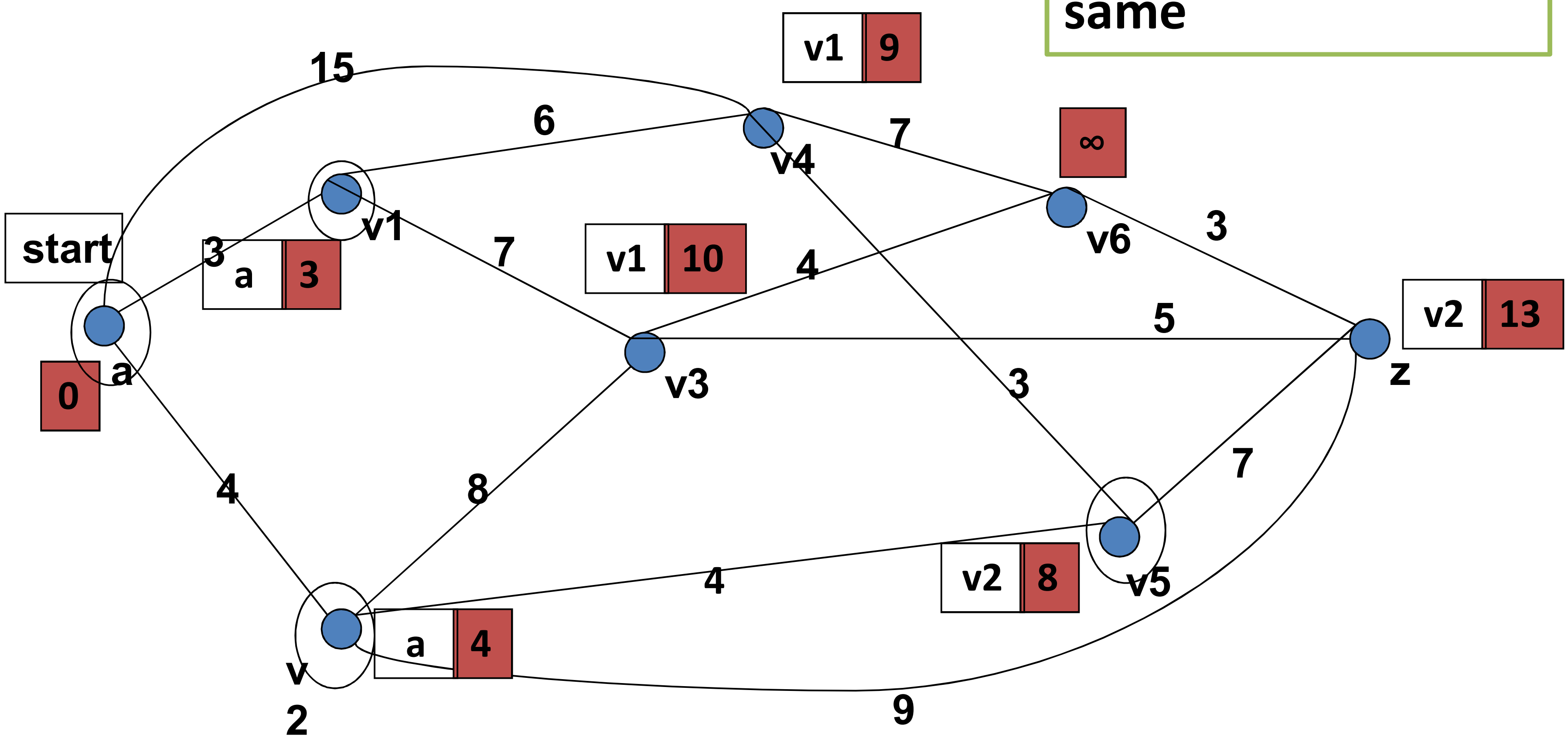
# $S=\{a, v1, v2, v5\}$

# $N= \{v3,v4, v6,z\}$

$L(v5)+W[v5, z] < L(z)$

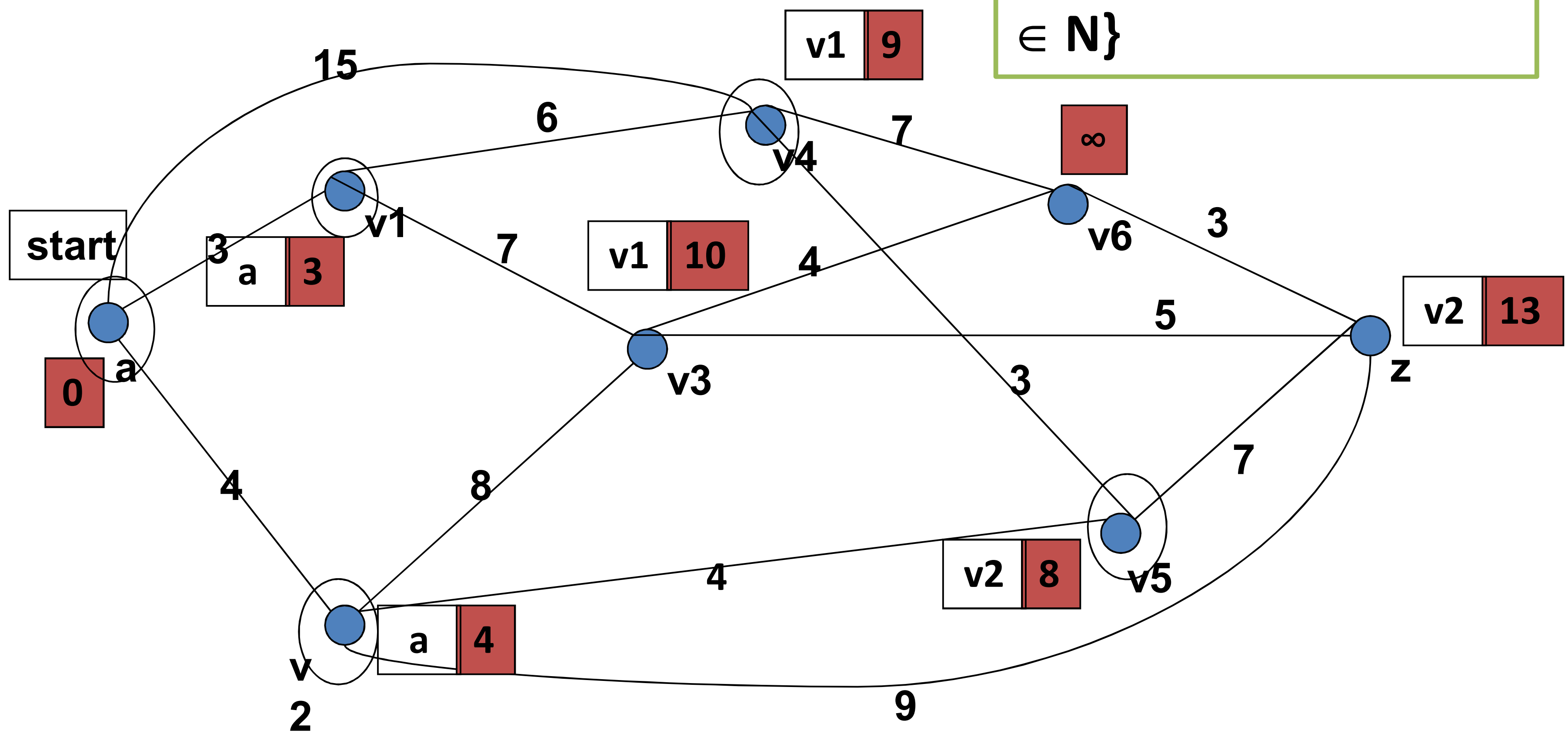
$8+7= 15 > 13$

$L(z)$ remains the same



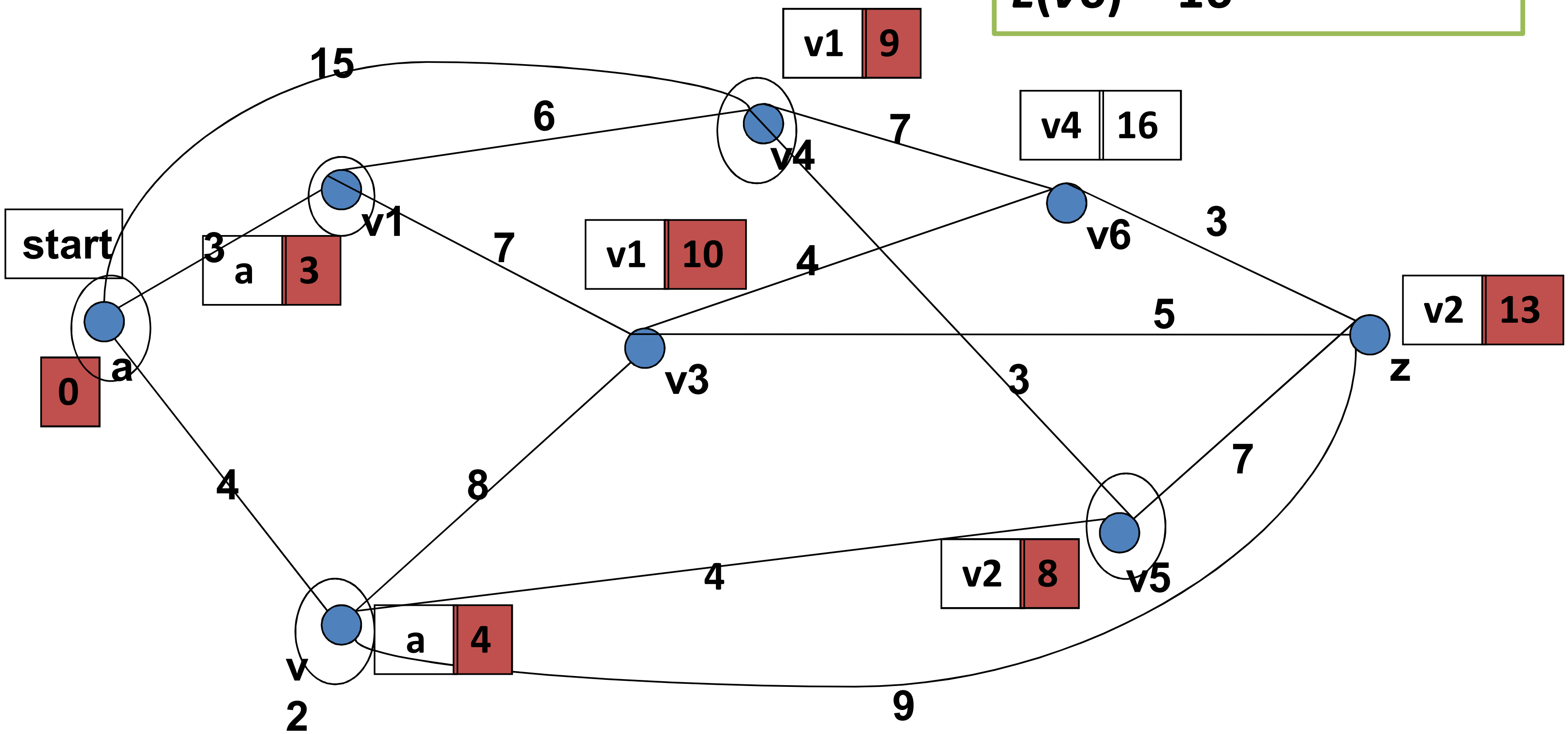
# *S*={*a*, *v1*, *v2*, *v5*}
# *N*= {*v3*,*v4*, *v6*,*z*}

choose *v4* because *L*(*v4*)=9 = min{L(u)|u ∈ N}



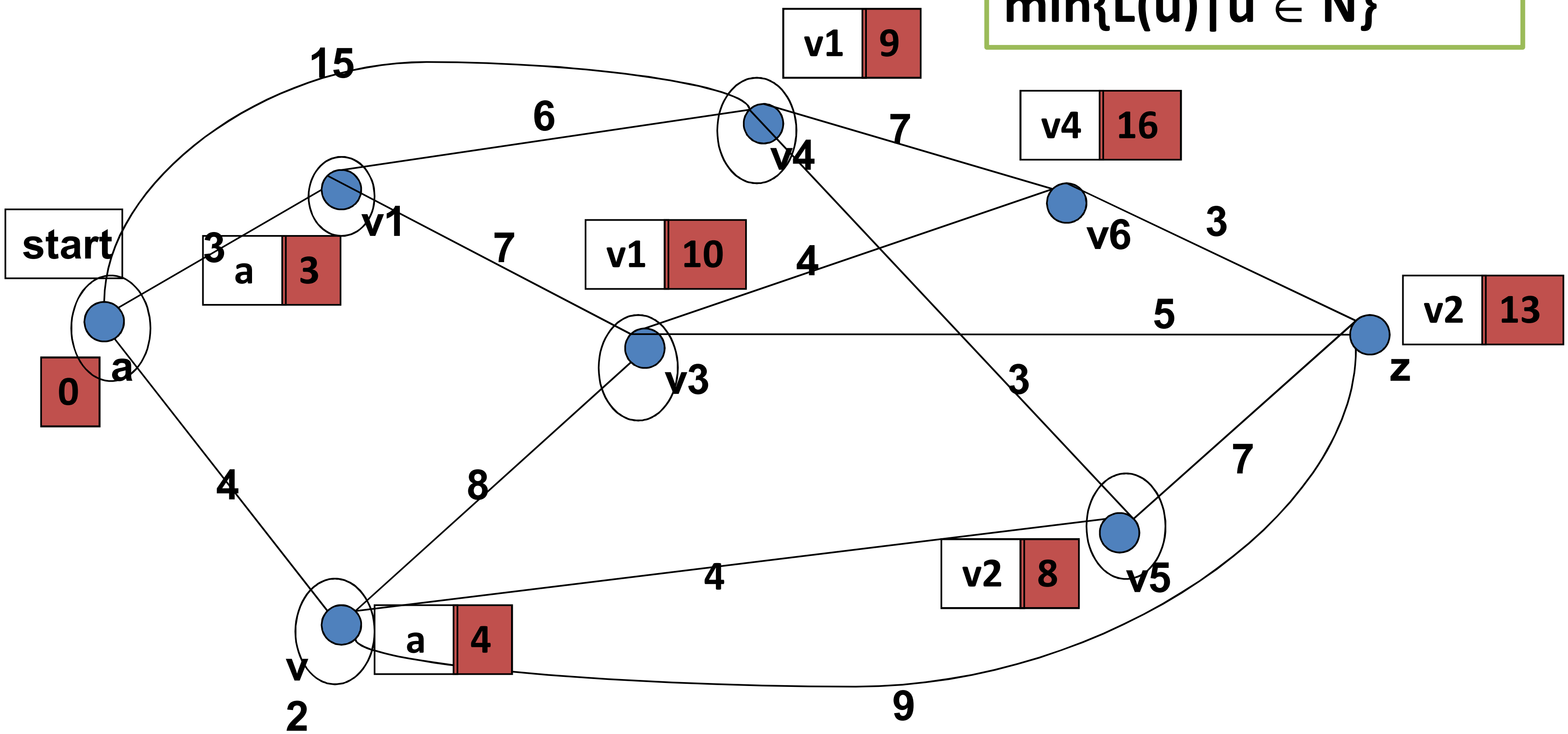
# S={a, v1, v2, v5,v4}
# N= {v3,v6,z}

*L*(v4)+*W*[v4, v6]<*L*(v6)
9+7= 16 < ∞
*L*(v6) = 16



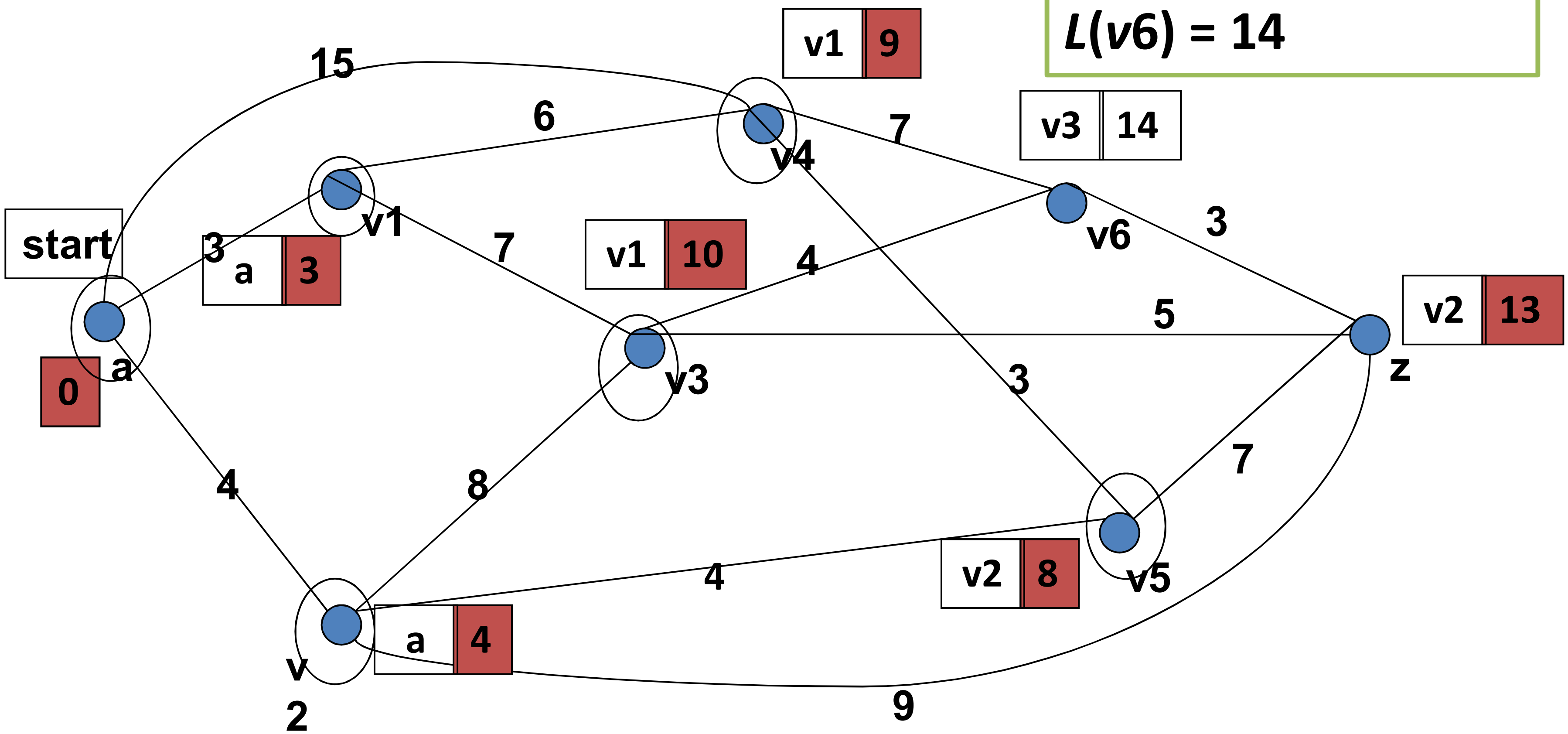
## *S*={*a*, *v1*, *v2*, *v5*,*v4*}
## *N*= {v3, v6, *z*}

choose *v3*
because
*L*(*v3*)=10 =
min{L(u)|u ∈ N}



## $S=\{a, v1, v2, v5, v4, v3\}$
## $N= \{v6, z\}$

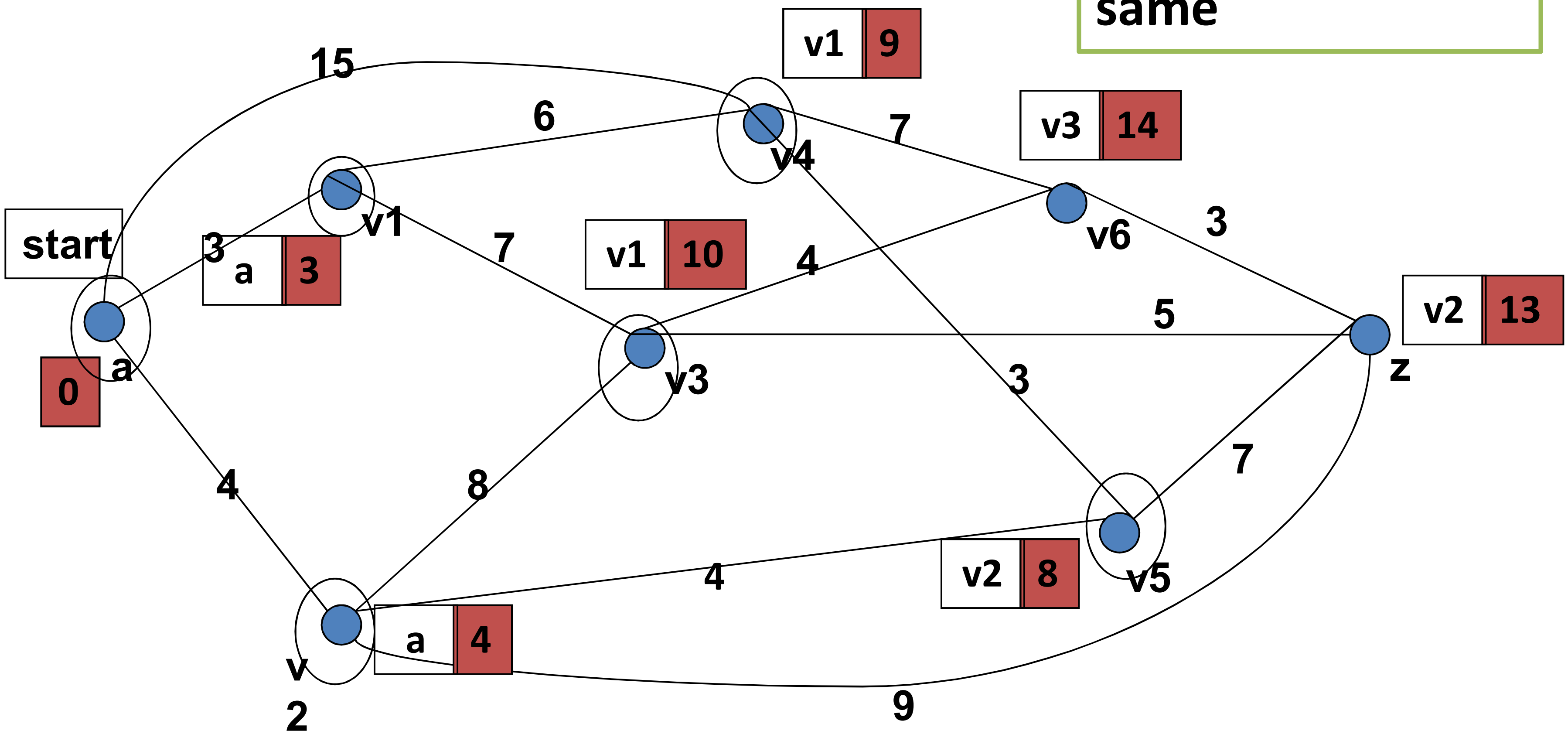
$L(v3)+W[v3, v6]<L(v6)$
10+4= 14 < 16
$L(v6) = 14$



# *S*={*a*, *v*1, *v*2, *v*5,*v*4,*v*3}

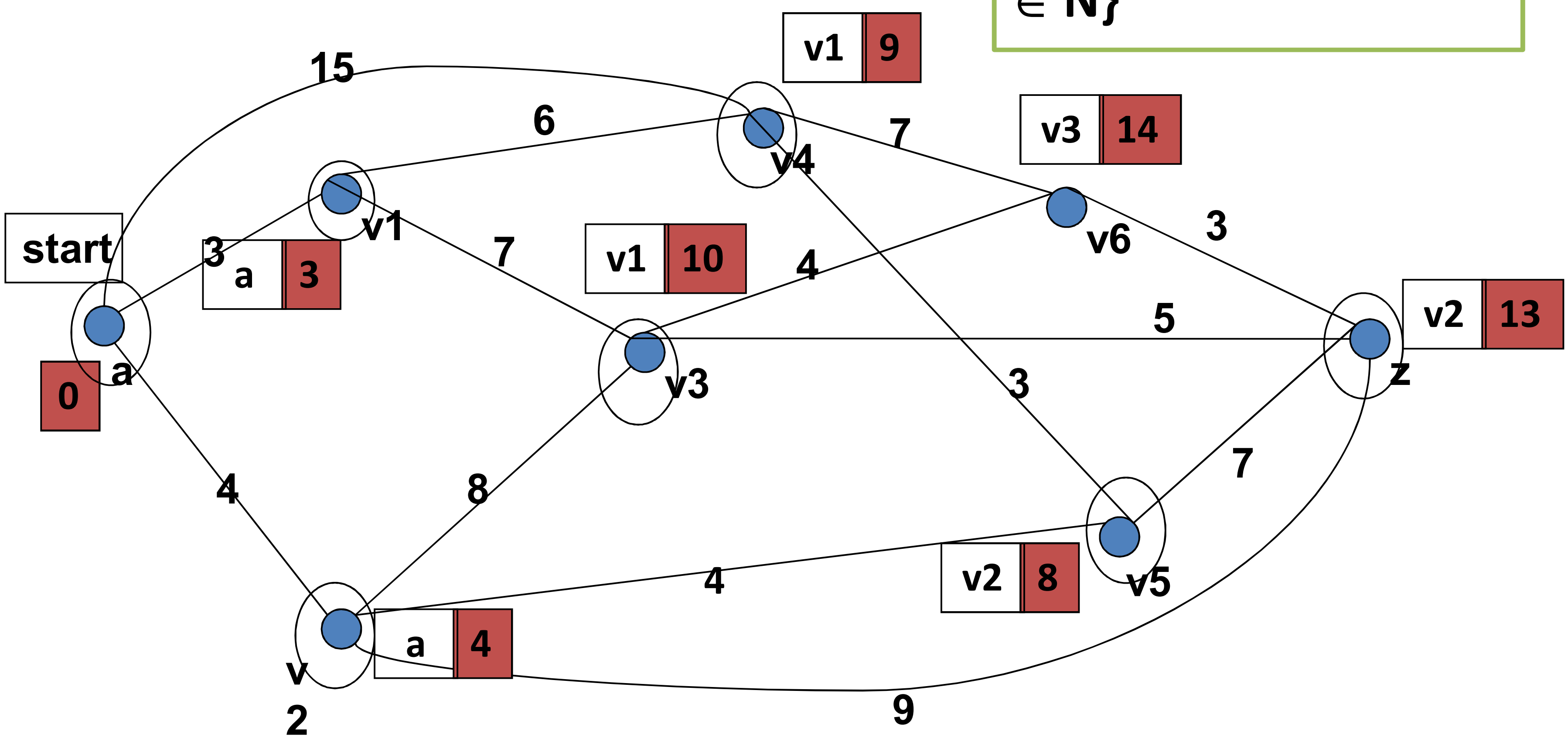
# *N*= {v6,*z*}

$L(v3)+W[v3, z]$
$< L(z)$
$10+5= 15 > 13$
$L(z)$ remains the same



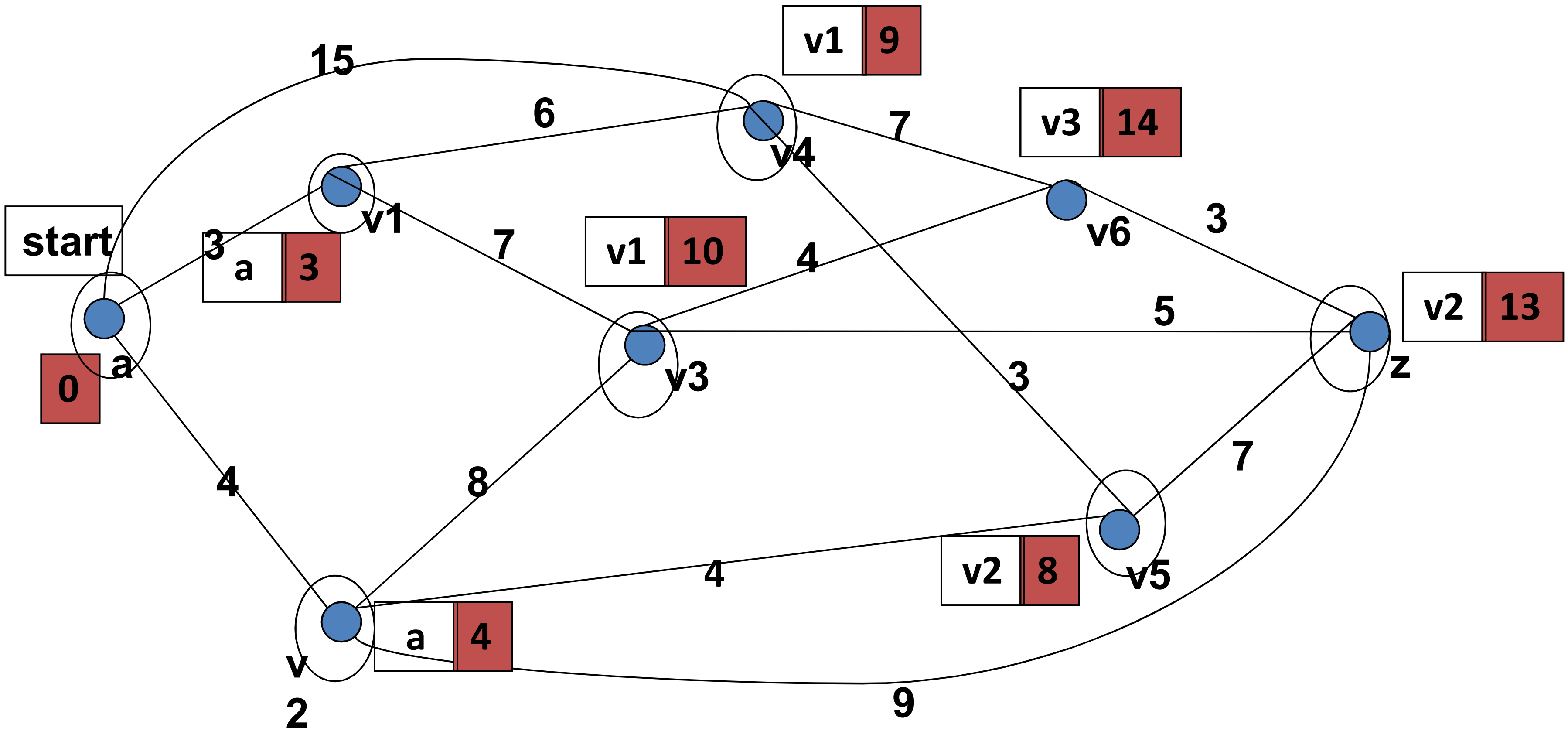
# *S={a, v1, v2, v5,v4,v3}*
# *N= {v6,z}*

**choose *z* because *L*(*z*)=13 = min{L(u)|u ∈ N}**

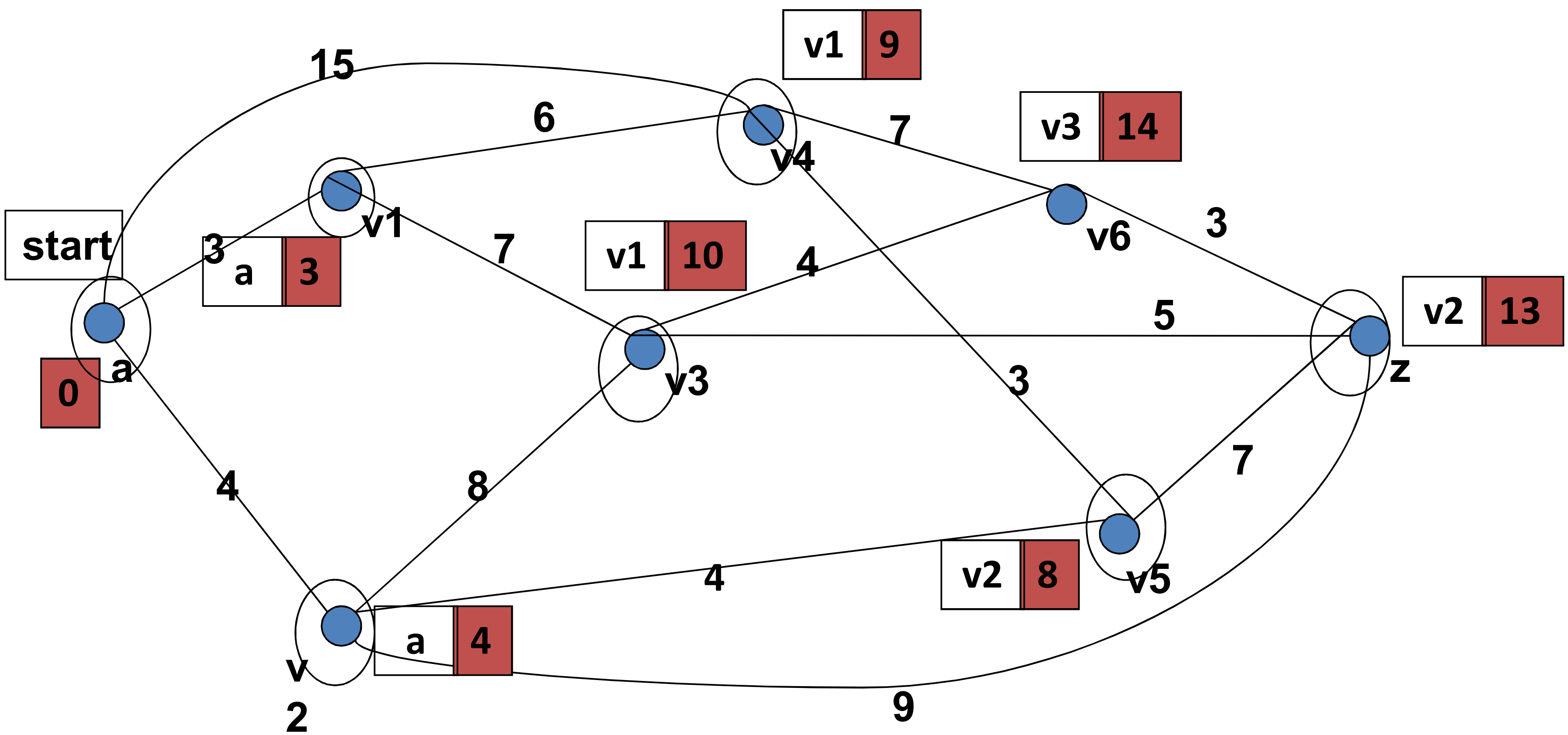


# $S=\{a, v1, v2, v5, v4, v3, z\}$
# $N=\{v6\}$

The loop terminates because $z \in S$



Shortest path from *a* to z is a -> v2-> z, with the length 13.

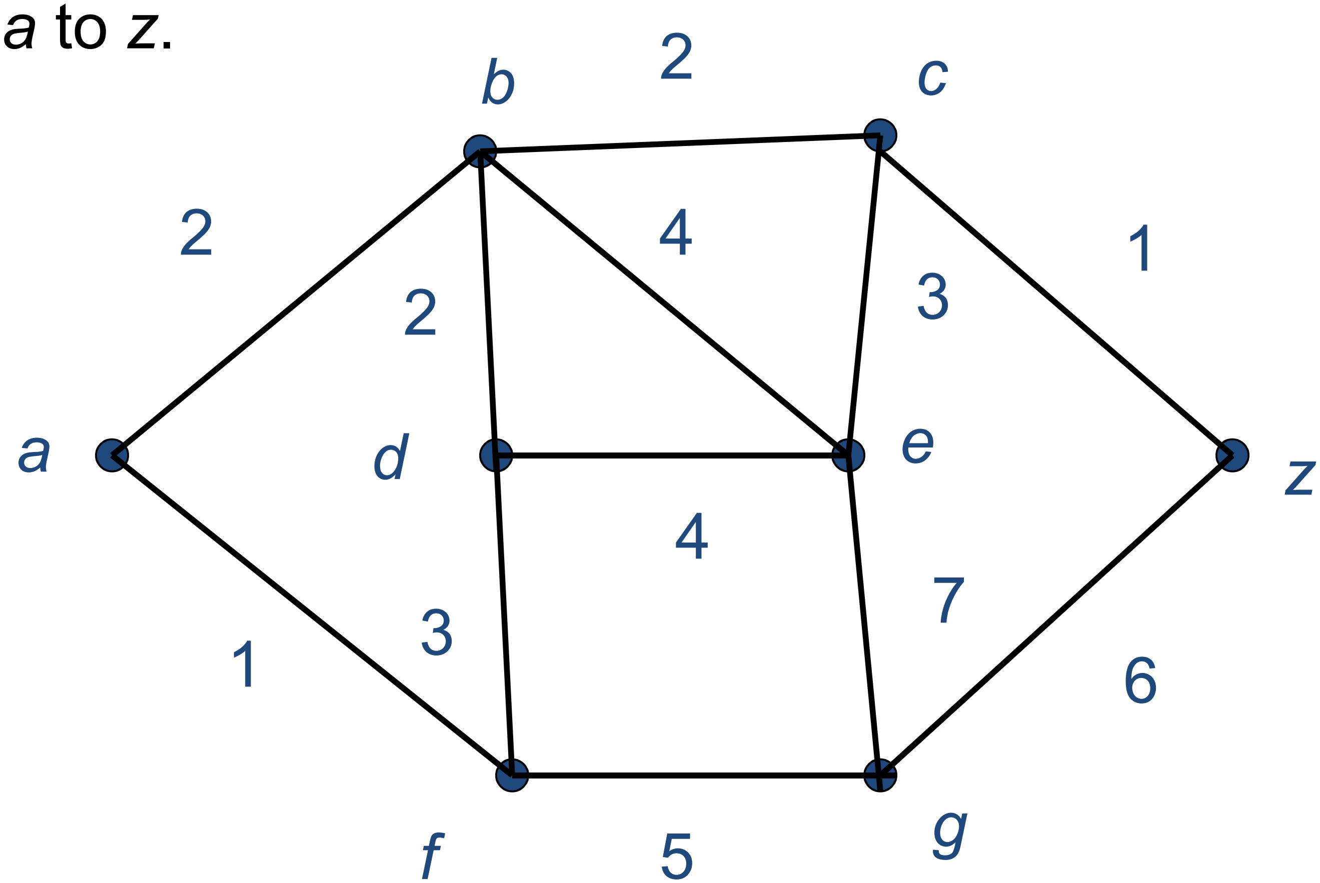


# Table – Djikstra Algorithm

| No. | S | N | L(a) | $L(V_1)$ | $L(V_2)$ | $L(V_3)$ | $L(V_4)$ | $L(V_5)$ | $L(V_6)$ | L(z) |
|---|---|---|---|---|---|---|---|---|---|---|
| 0 | { } | $\{a, V_1, V_2, V_3, V_4, V_5, V_6, z\}$ | 0 | ∞ | ∞ | ∞ | ∞ | ∞ | ∞ | ∞ |
| 1 | $\{a\}$ | $\{V_1, V_2, V_3, V_4, V_5, V_6, z\}$ | | 3 | 4 | ∞ | 15 | ∞ | ∞ | ∞ |
| 2 | $\{a, V_1\}$ | $\{V_2, V_3, V_4, V_5, V_6, z\}$ | | **3** | 4 | 10 | 9 | ∞ | ∞ | ∞ |
| 3 | $\{a, V_1, V_2\}$ | $\{V_3, V_4, V_5, V_6, z\}$ | | | **4** | 10 | 9 | 8 | ∞ | 13 |
| 4 | $\{a, V_1, V_2, V_5\}$ | $\{V_3, V_4, V_6, z\}$ | | | | 10 | 9 | **8** | ∞ | 13 |
| 5 | $\{a, V_1, V_2, V_5, V_4\}$ | $\{V_6, z\}$ | | | | 10 | **9** | | 16 | 13 |
| 6 | $\{a, V_1, V_2, V_5, V_4, V_3,\}$ | $\{V_6, z\}$ | | | | **10** | | | 14 | 13 |
| 7 | $\{a, V_1, V_2, V_5, V_4, V_3, z\}$ | $\{V_6\}$ | | | | | | | 14 | **13** |

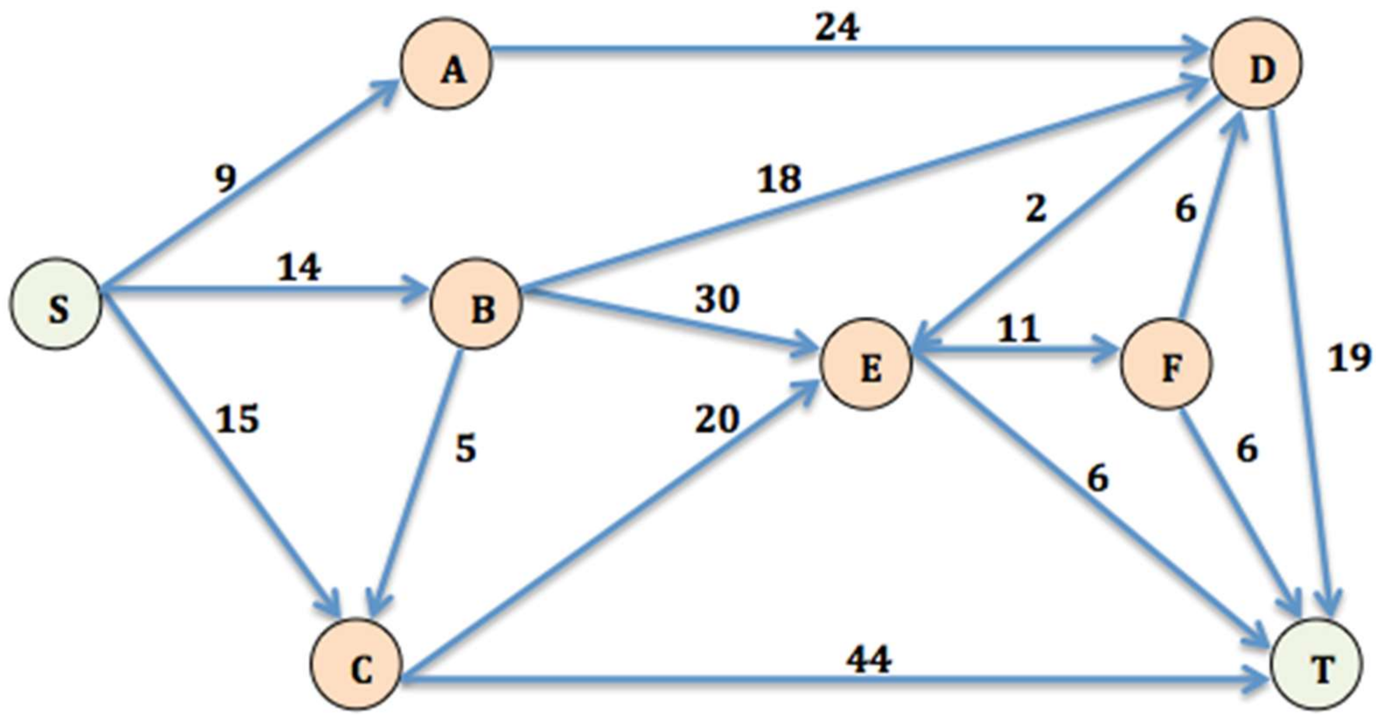
# exercise

Use Dijkstra’s algorithm to find the length of a shortest path from *a* to *z*.



# exercise

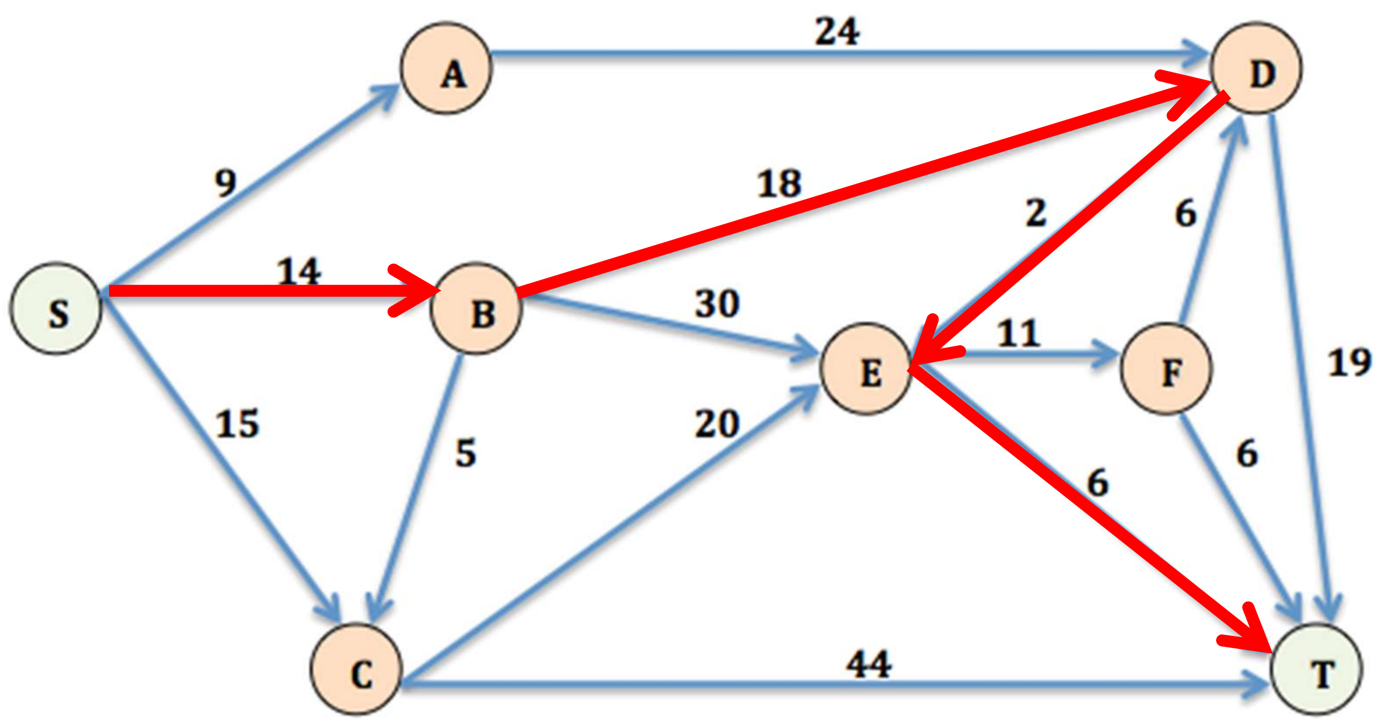
**Q: Given a weighted digraph, find the shortest path from S to T, using Djik**




Note: Weights are arbitrary numbers (i.e., not necessarily distances).

# Exercise Solution

The shortest path from **S** to **T**, having weight 40, is S – B – D – E – T




Note: Weights are arbitrary numbers (i.e., not necessarily distances).

Exercise

Past Year

2015/2016

The network in Figure 5 gives the distances in miles between pairs of cities A, B, …, and H.

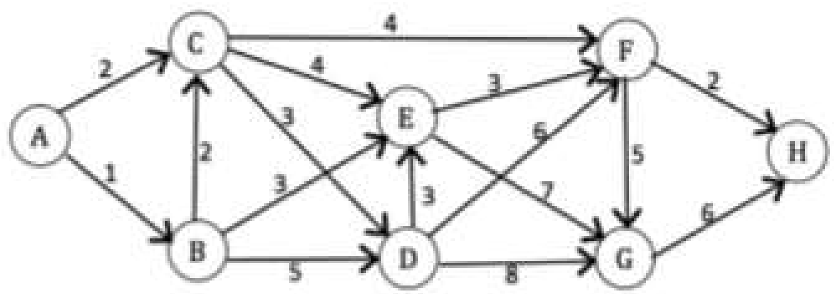



**Figure 5**

a) Based on Dijkstra's algorithm, complete Table 1 to find the shortest path from city A to city H. (Note: Copy Table 1 into your answer booklet).

(8 marks)

**Table 1**

| Iteration | S | N | $L$(A) | $L$(B) | $L$(C) | $L$(D) | $L$(E) | $L$(F) | $L$(G) | $L$(H) |
|---|---|---|---|---|---|---|---|---|---|---|
| 0 | | | | | | | | | | |
| 1 | | | | | | | | | | |
| 2 | | | | | | | | | | |
| 3 | | | | | | | | | | |
| 4 | | | | | | | | | | |
| 5 | | | | | | | | | | |
| 6 | | | | | | | | | | |
| 7 | | | | | | | | | | |

b) State the minimum distance and the shortest path from city 1 to city 8.

(2 marks)

# SCSI 1013 DISCRETE STRUCTURE

## CHAPTER 4 – PART 2

# TREE

Semester 1 – 2017/2018

# Introduction

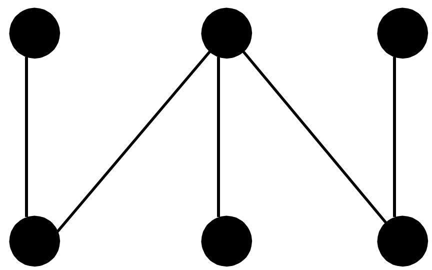
**Definition 1.** A tree is a connected undirected graph with no simple circuits.

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

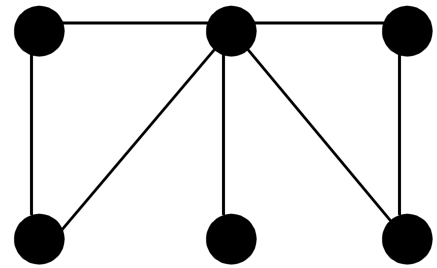
**Theorem 2 .** A tree with m-vertices has m-1 edges

# Which graphs are trees?

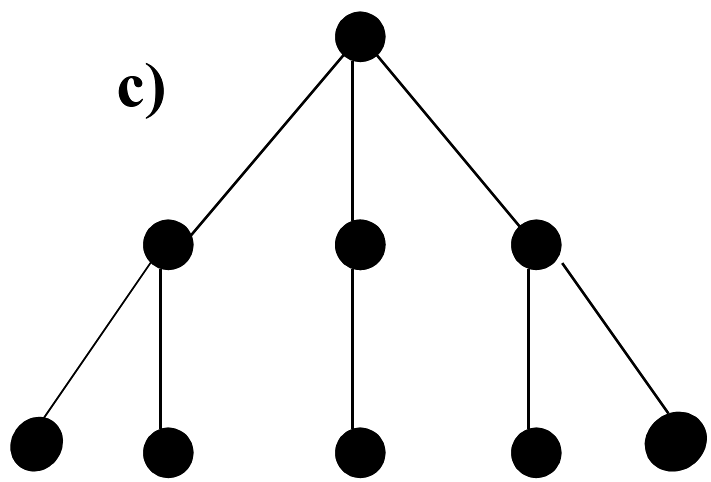
a)



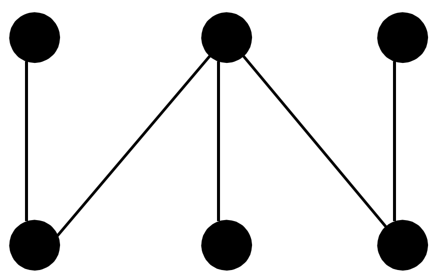
b)



c)



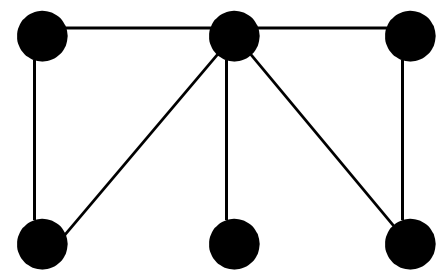
# Solution



**tree**

***vertices = 6***

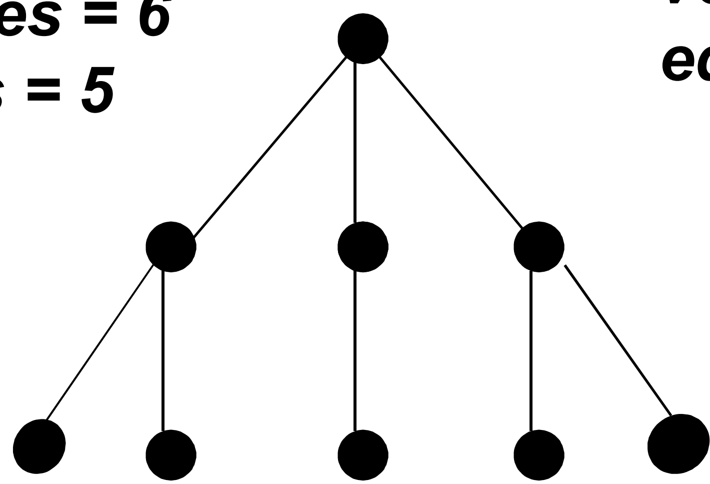
***edges = 5***



**Not a tree**

***vertices = 6***

***edges = 7***



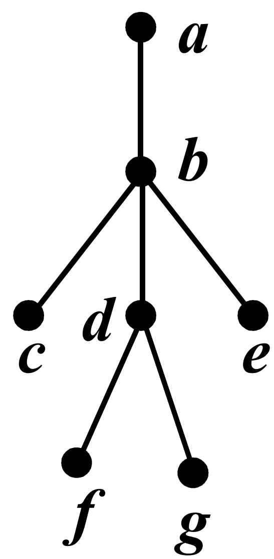
**tree**

***vertices = 9***

***edges = 8***

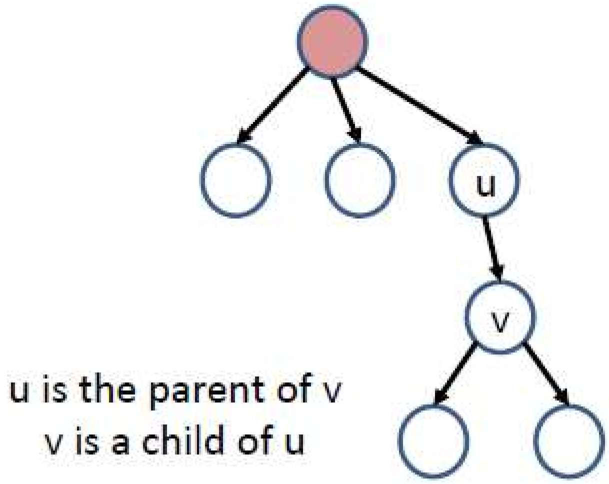
# Rooted tree

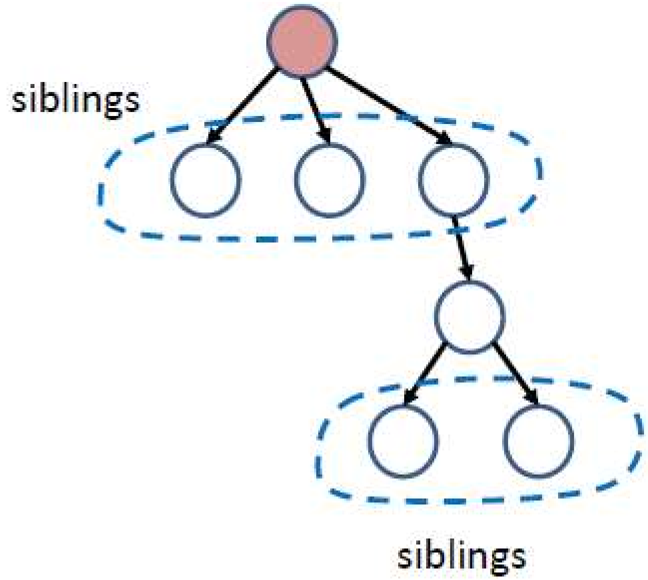
**Definition 2**. A **rooted tree** is a tree in which one vertex has been designed as the **root** and every edge is directed away from the root.



# Rooted Tree - Terminologies

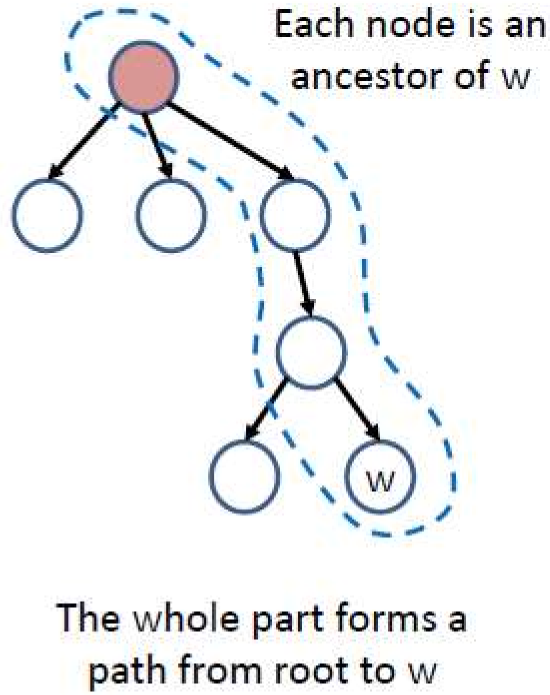
- Each edge is from a parent to a child
- Vertices with the same parent are siblings



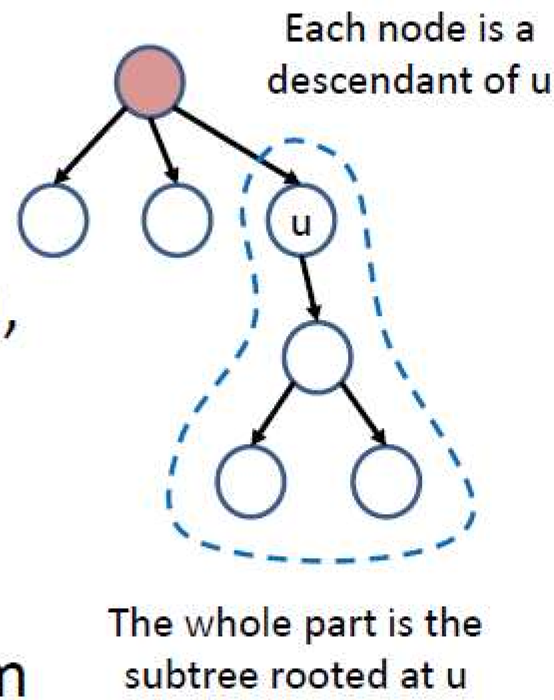
# Rooted Tree - Terminologies

- 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



# 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

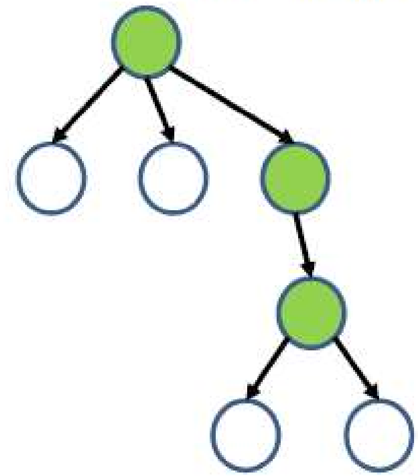


# Rooted Tree - Terminologies

- Vertices with no children are called leaves ;

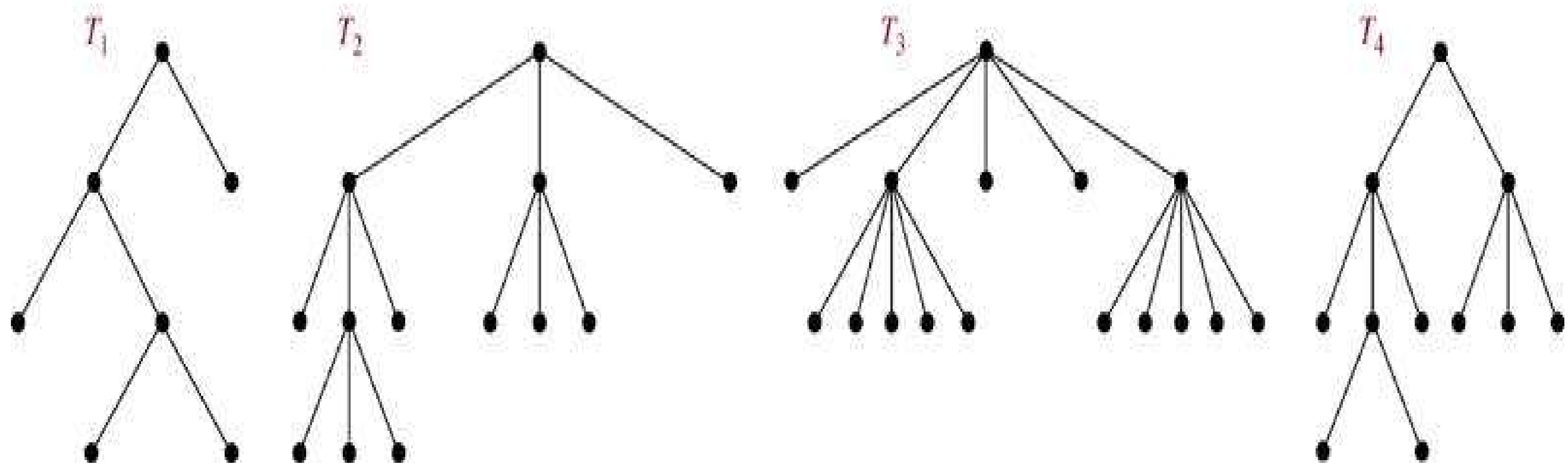
  Otherwise, they are called internal nodes
- If every internal node has no more than m children, the tree is called an m-ary tree
  - Further, if every internal node has exactly m children, the tree is a full m-ary tree

All internal nodes are colored



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

# Properties of Trees

- Theorem : A tree with n nodes has n-1 edges
- Theorem : 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**

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

# Properties of Trees

Theorem – A full **m-ary** tree with

- ***n*** vertices has ***i = (n-1)/m*** internal vertices and ***l = [(m-1)n+1]/m*** leaves.

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

# Properties of Trees

Theorem – A full **m-ary** tree with

- ***i*** internal vertices has ***n =mi+1*** vertices and ***l = (m-1)i + 1*** leaves

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

# Properties of Trees

Theorem – A full **m-ary** tree with

- $l$ leaves has $n = (ml - 1)/(m-1)$ vertices and $i = (l - 1)/(m-1)$ internal vertices

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

UTM UNIVERSITI TEKNOLOGI MALAYSIA

# Example

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 ?

# Exercise

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

# Exercise

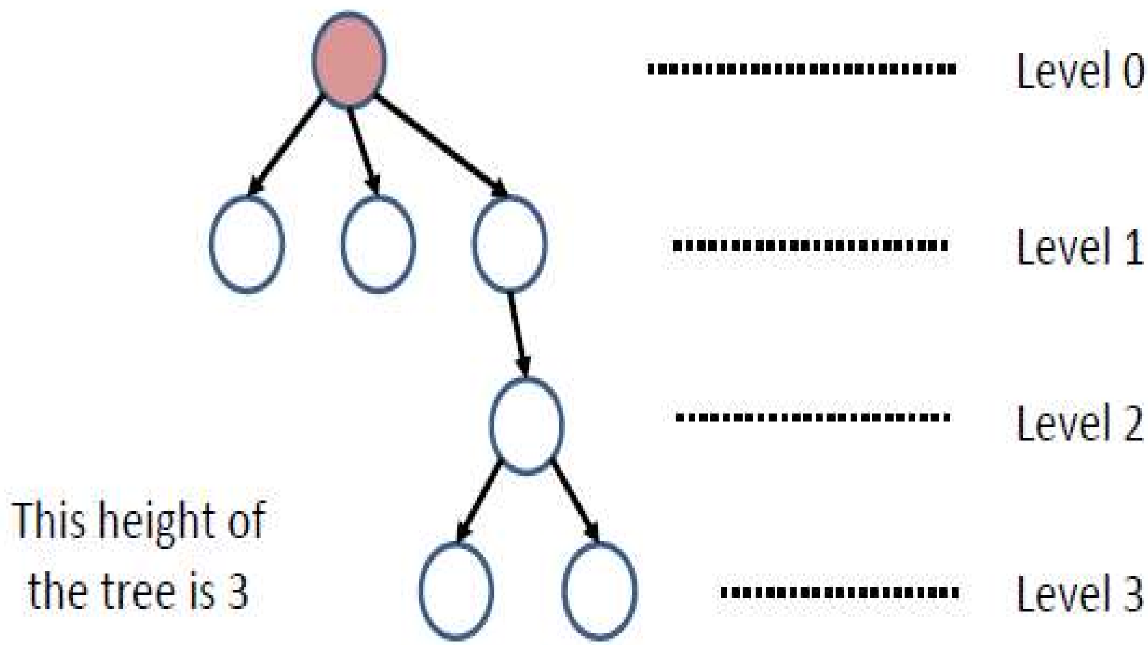
Suppose 1000 people enter a chess tournament. Use a rooted tree model of the tournament to determine how many games must be played to determine a champion, if a player is eliminated after one loss and games are played until only one entrant has not lost. (Assume there are no ties.)

# Properties of 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

- Ex :



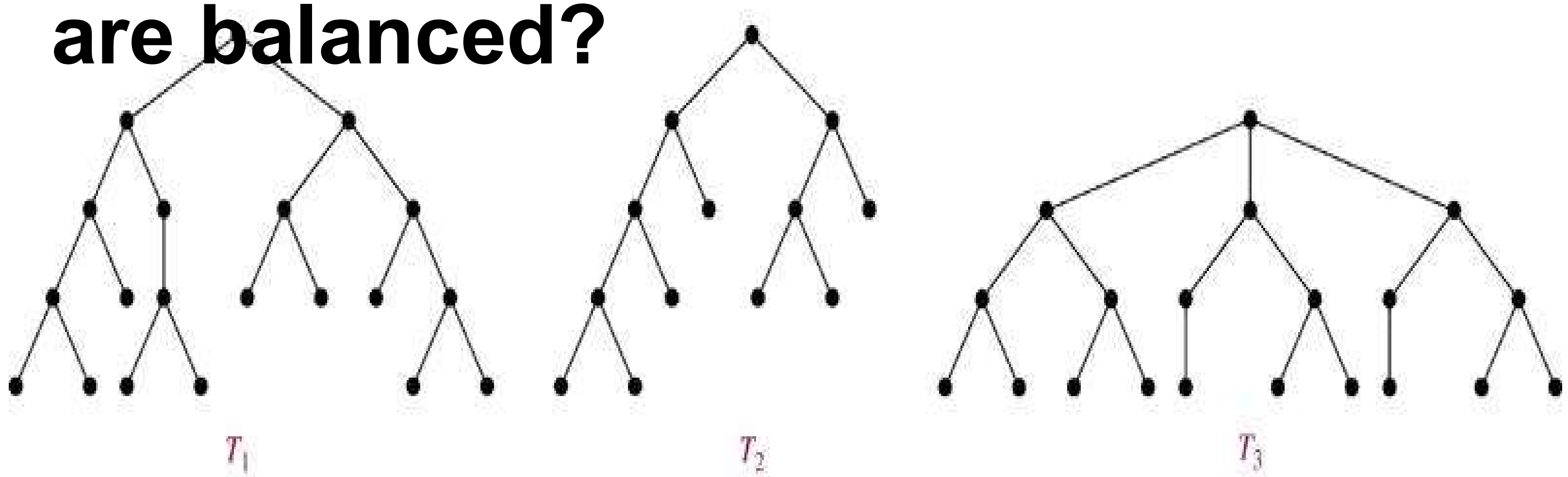
# Properties of Trees

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

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

# Example

**Which of the rooted trees shown below are balanced?**




**Sol.** $T_1$, $T_3$

# Tree Traversal

## Universal Address Systems

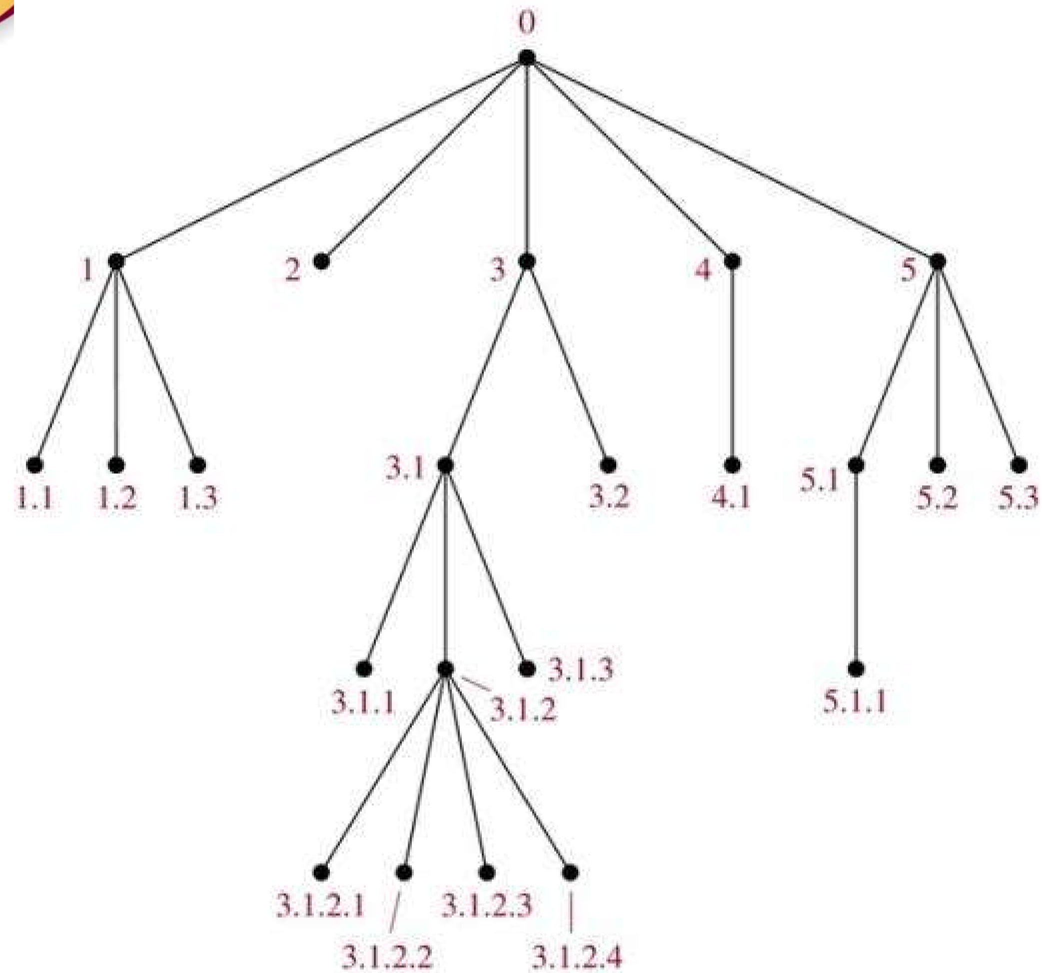
**Label vertices:**

**1.root $\rightarrow$ 0, its $k$ children $\rightarrow$ 1, 2, …, $k$ (from left to right)**
**2.For each vertex $v$ at level $n$ with label $A$,**
**its $r$ children $\rightarrow$ $A$.1, $A$.2, …, $A.r$ (from left to right).**

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

$x_1.x_2.\dots.x_n < y_1.y_2.\dots.y_m$

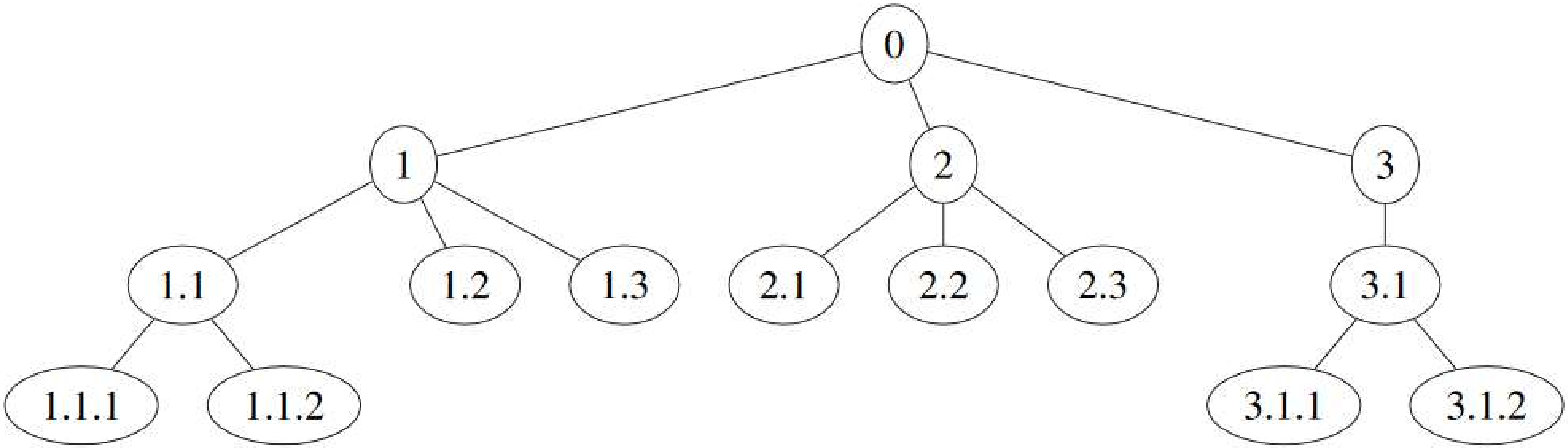
**if there is an $i$, $0 \leq i \leq n$, with $x_1=y_1$, $x_2=y_2$, …, $x_{i-1}=y_{i-1}$, and $x_i<y_i$; or if $n<m$ and $x_i=y_i$ for $i=1, 2, \dots, n$.**




# 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**

# Exercise




Find the lexicographic ordering of the above tree.

# Tree Traversal

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

# Tree Traversal

- Pre-Order
  - Visit the root
  - Traverse the left sub-tree in pre-order
  - Traverse the right sub-tree in pre-order
- In-Order
  - Traverse the left sub-tree in in-order
  - Visit the root
  - Traverse the right sub-tree in in-order
- Post-Order
  - Traverse the left sub-tree in post-order
  - Traverse the right sub-tree in post-order
  - Visit the root

# Preorder Traversal algorithm

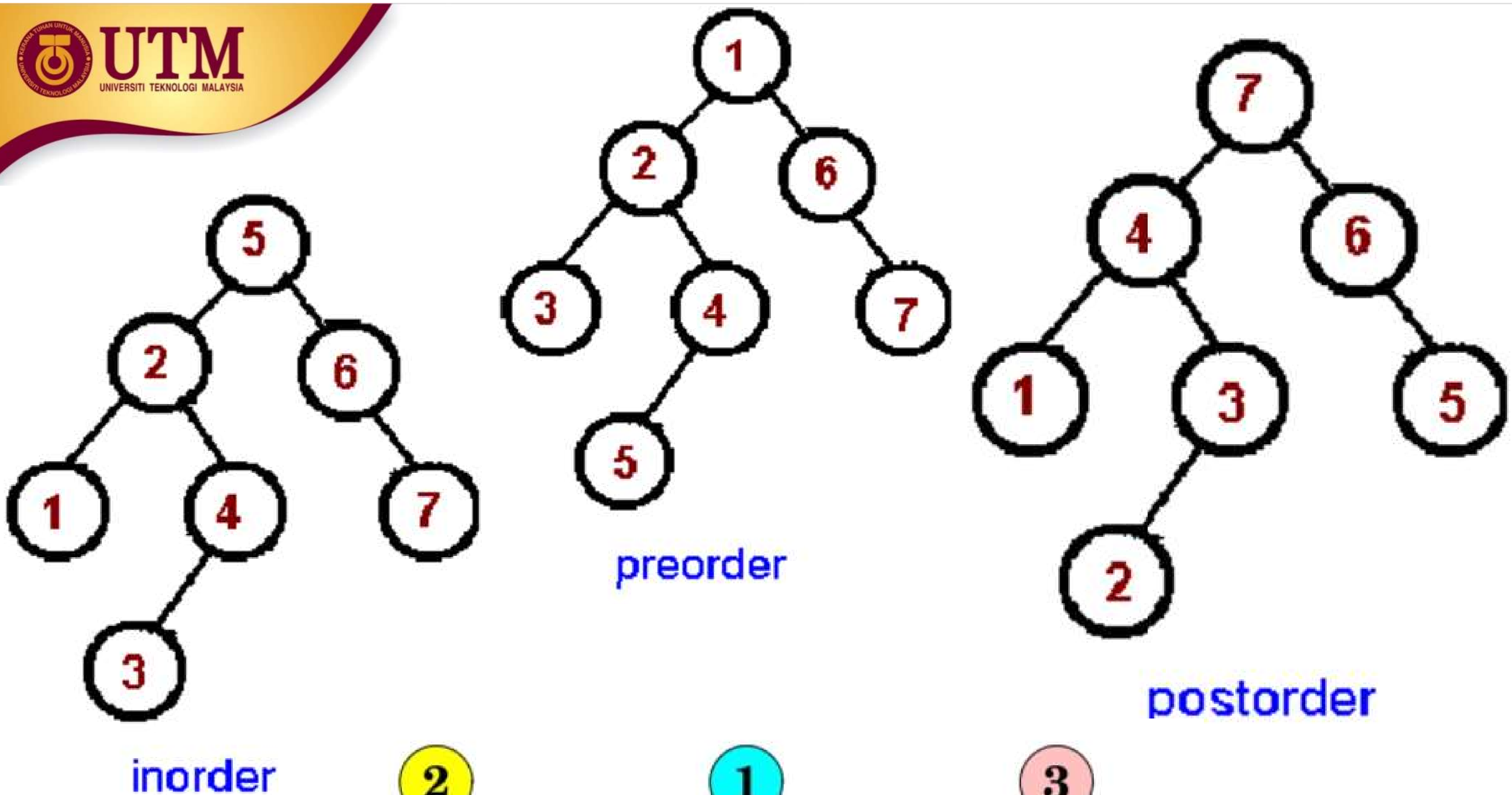
```
Procedure preorder(T: ordered rooted tree)
r := root of T
list r
for each child c of r from left to right
begin
    T(c) := subtree with c as its root
    preorder(T(c))
end
```

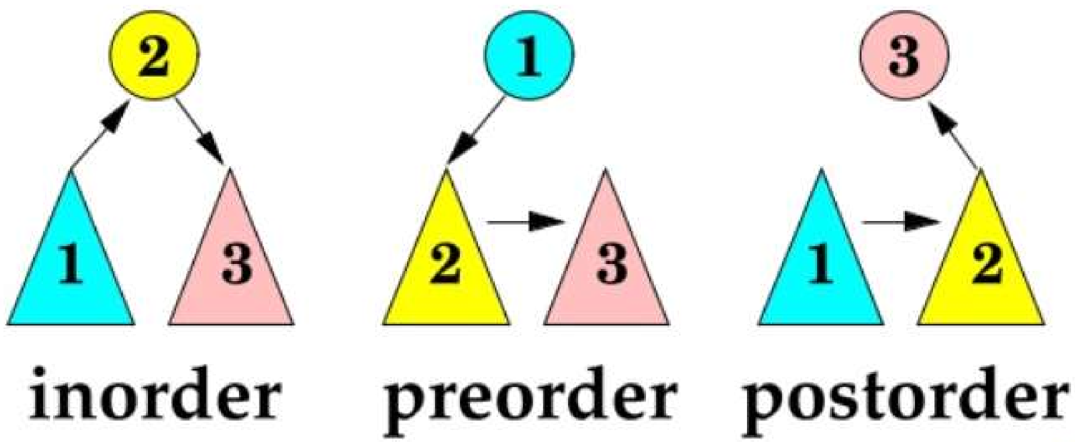
# Inorder Traversal algorithm

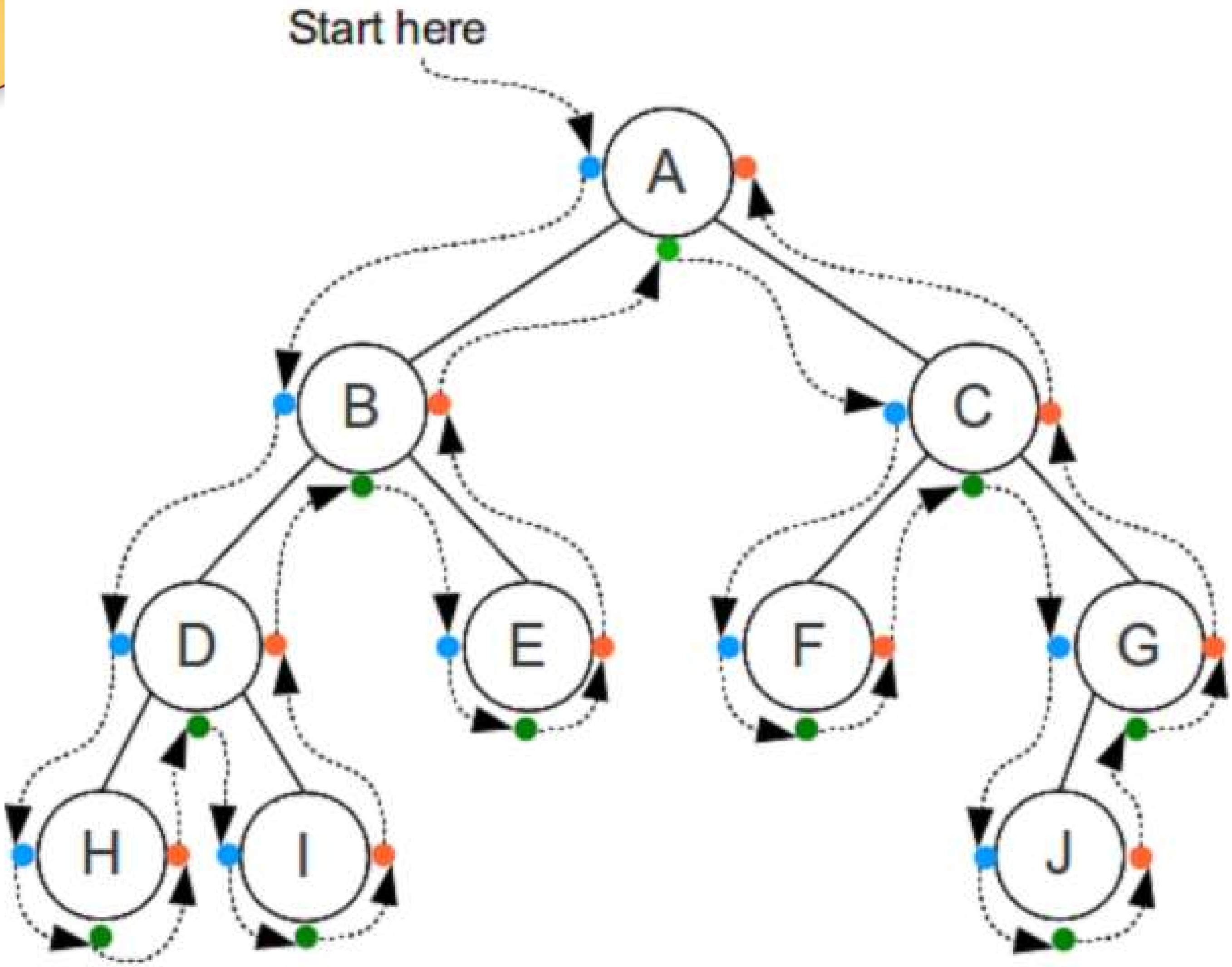
**Procedure** *inorder*(*T*: ordered rooted tree)
*r* := root of *T*
**If** *r* is a leaf **then** list *r*
**else**
**begin**
    *l* := first child of *r* from left to right
    *T*(*l*) := subtree with *l* as its root
    *inorder*(*T*(*l*))
    list *r*
    **for** each child *c* of *r* except for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

# Postorder Traversal algorithm

**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*


inorder
preorder
postorder

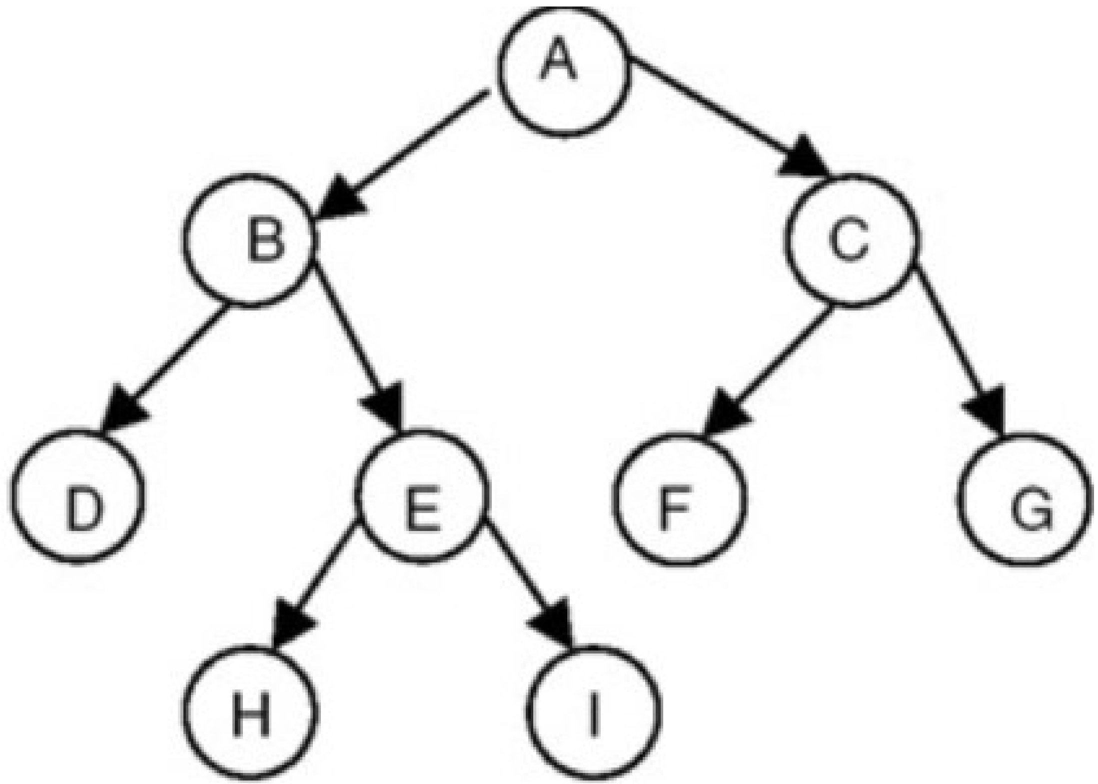

inorder
preorder
postorder




| | |
|---|---|
| Pre-Order | ABDHIECFGJ |
| In-Order | HDIBEAFCJG |
| Post-Order | HIDEBFJGCA |

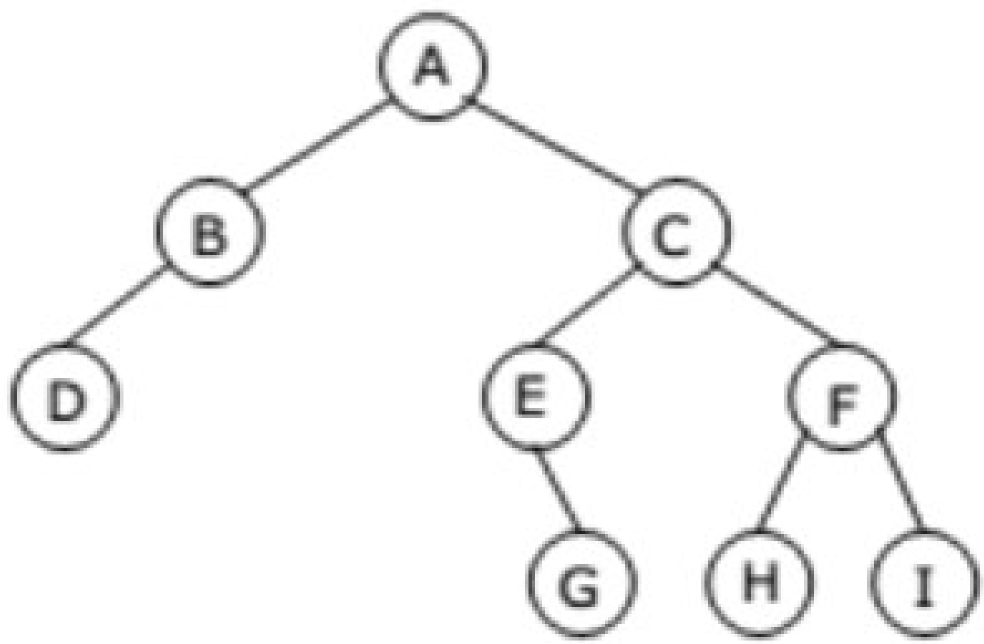
# Exercise

**Give the inorder, preorder, and postorder traversals for the following tree.**



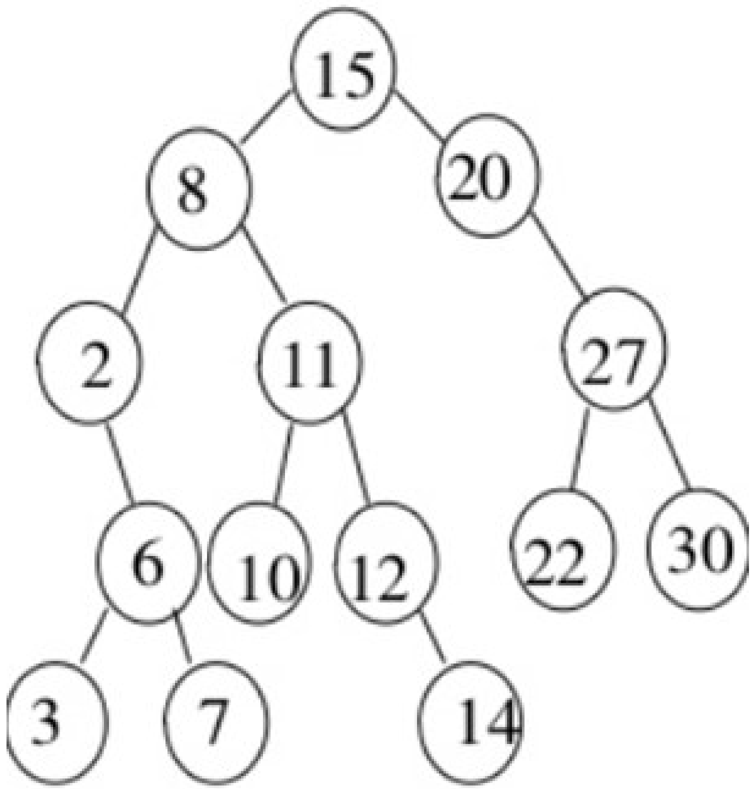
# Exercise

**Trace the inorder, preorder, and postorder traversals for the following tree.**



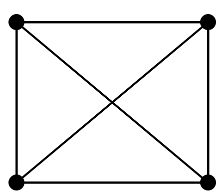
# Exercise

**Find the inorder, preorder, and postorder traversals for the following tree.**

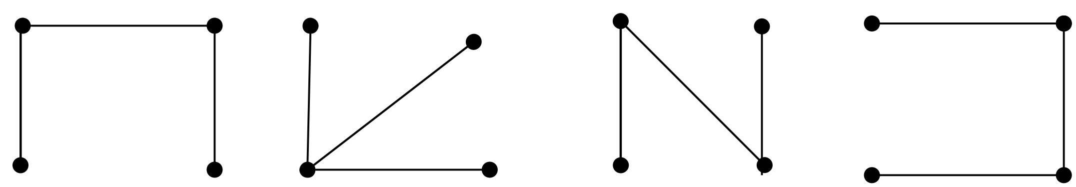


# Spanning Trees

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



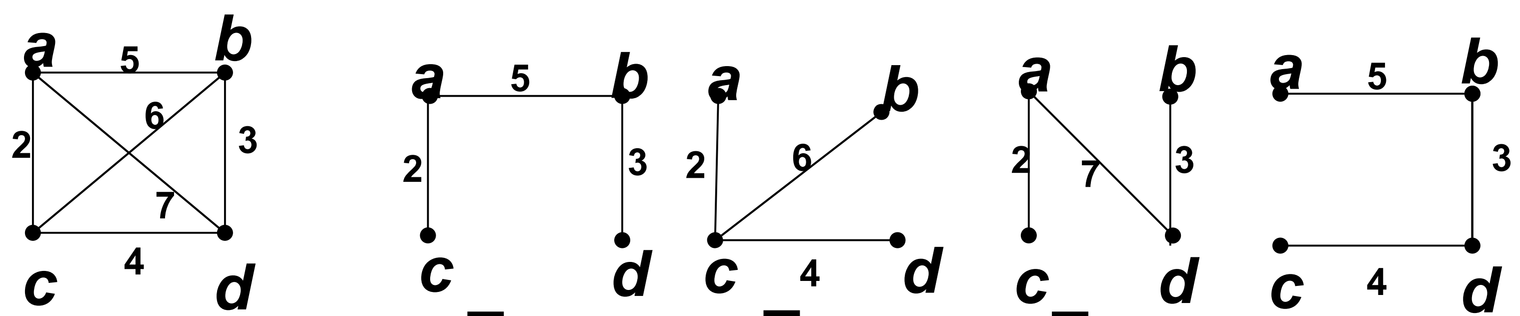
**A connected undirected graph**



**Four spanning trees of the graph**

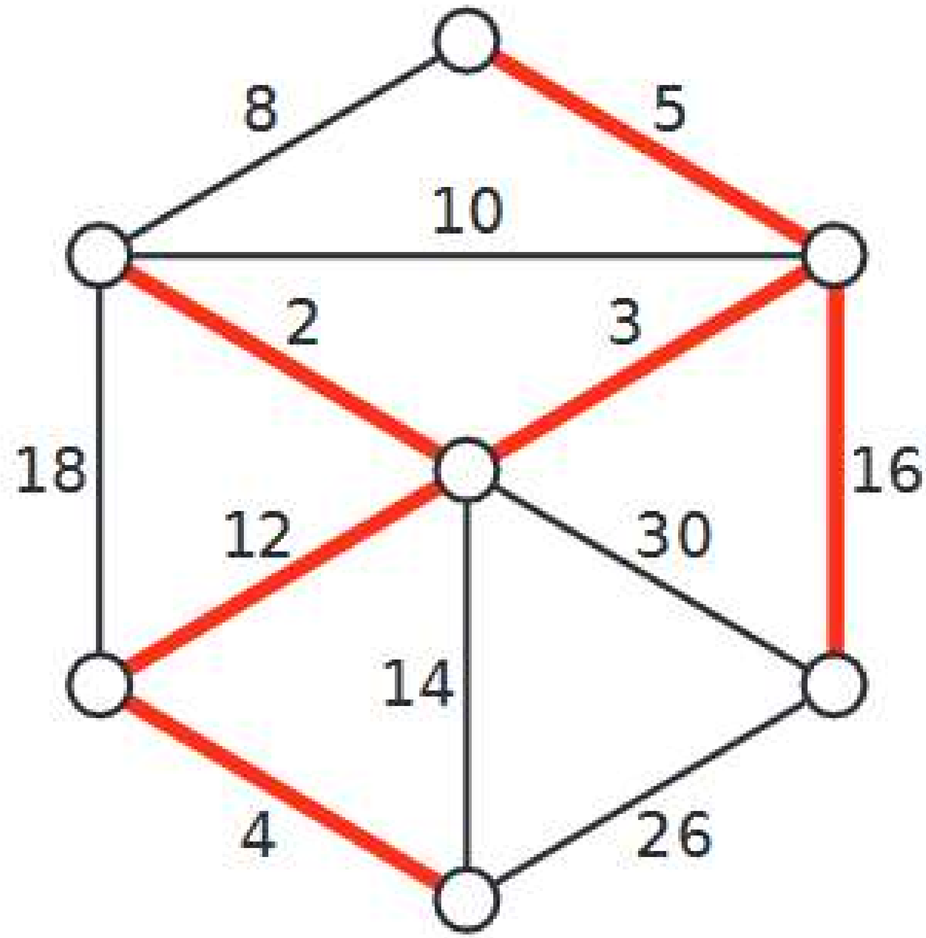
# Minimum Spanning Tree (MST)

- 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

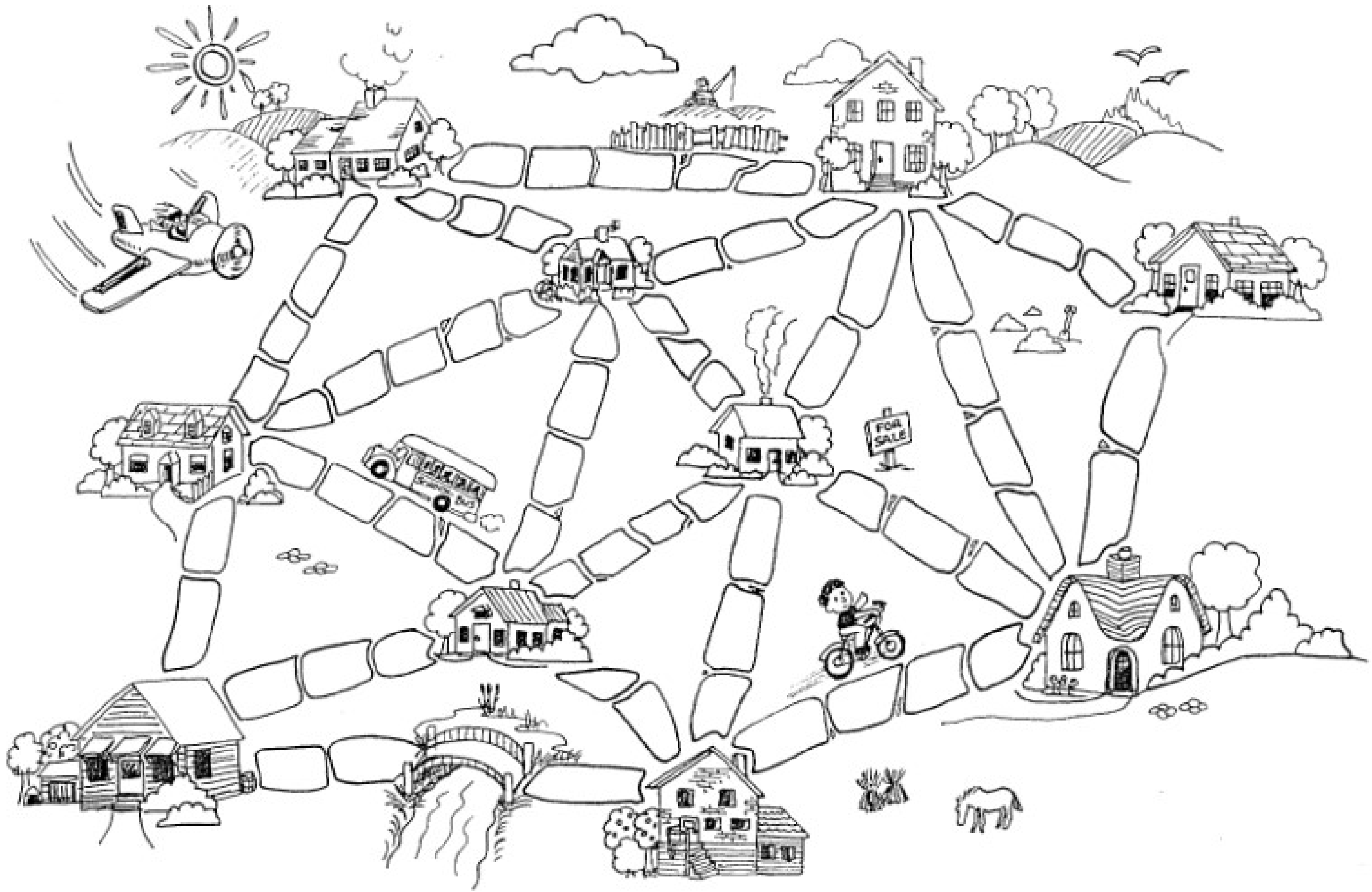
Once upon a time there was a city that had no roads. Getting around the city was particularly difficult after rainstorms because the ground became very muddy. Cars got stuck in the mud and people got their boots dirty. The mayor of the city decided that some of the streets must be paved, but didn't want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

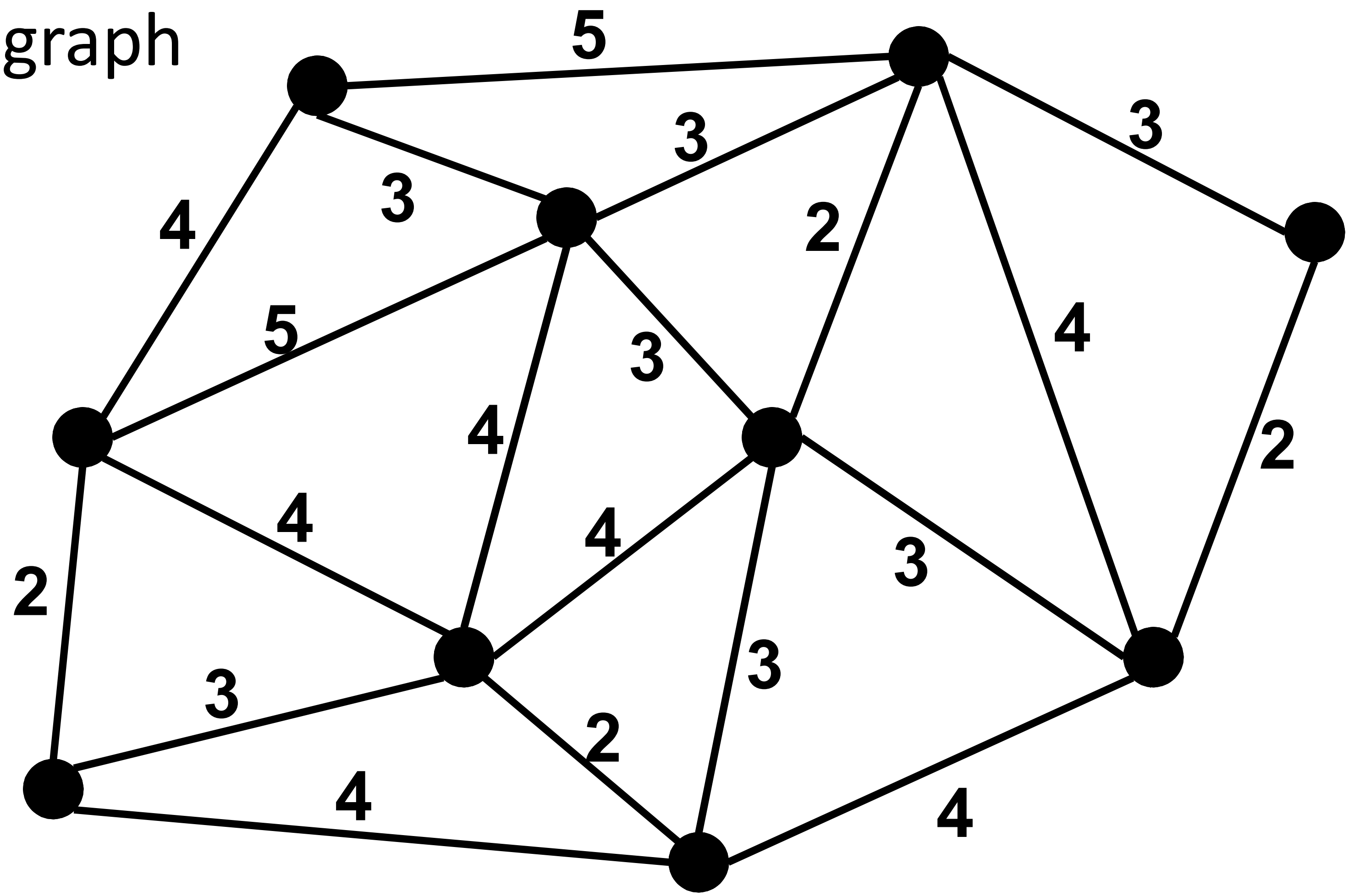
Here is the layout of the city. The number of paving stones between each house represents the cost of paving that route. Find the best route that connects all the houses, but uses as few counters (paving stones) as possible.


FOR SALE

UTM UNIVERSITI TEKNOLOGI MALAYSIA

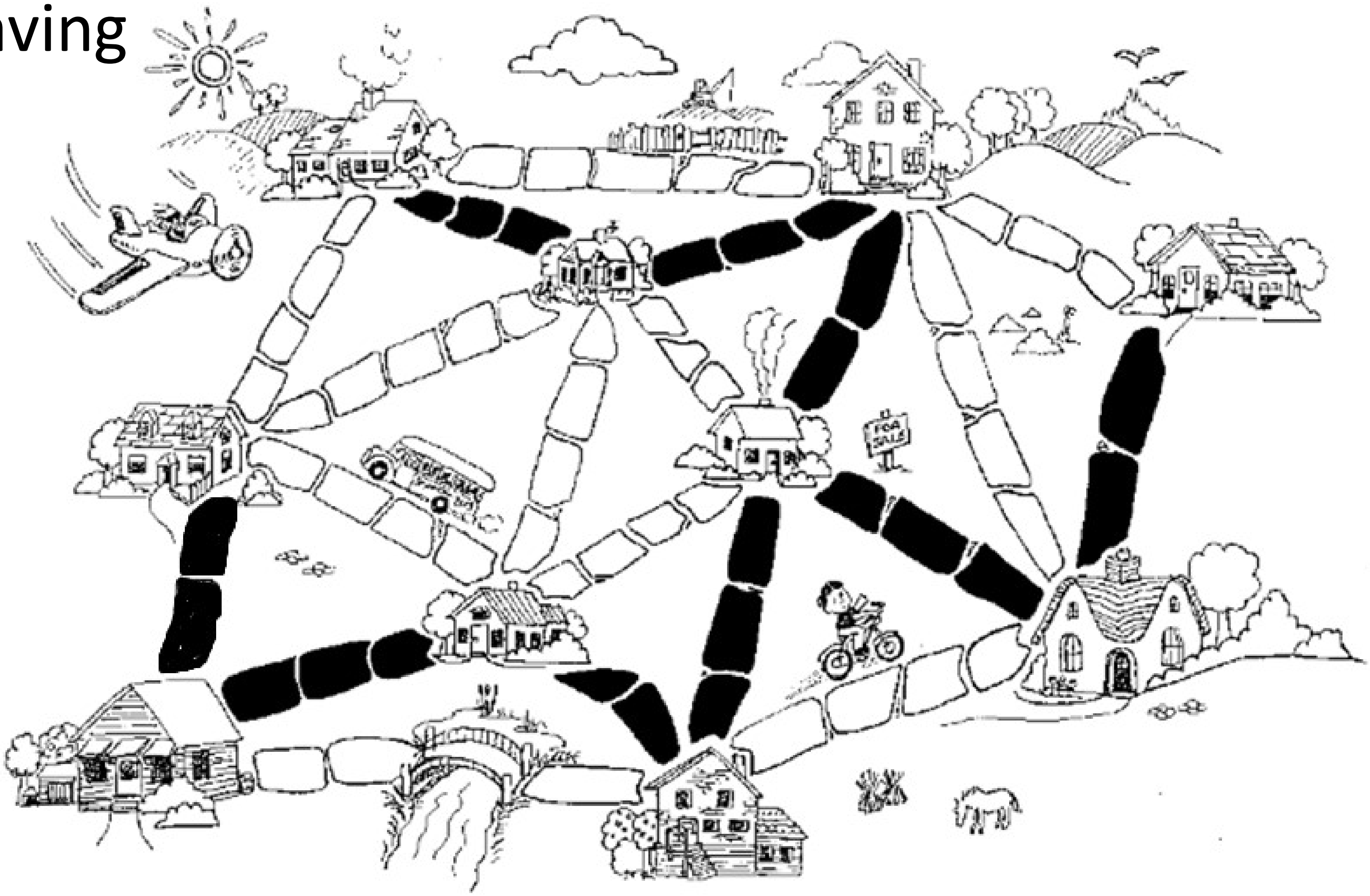
# Muddy City Problem

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

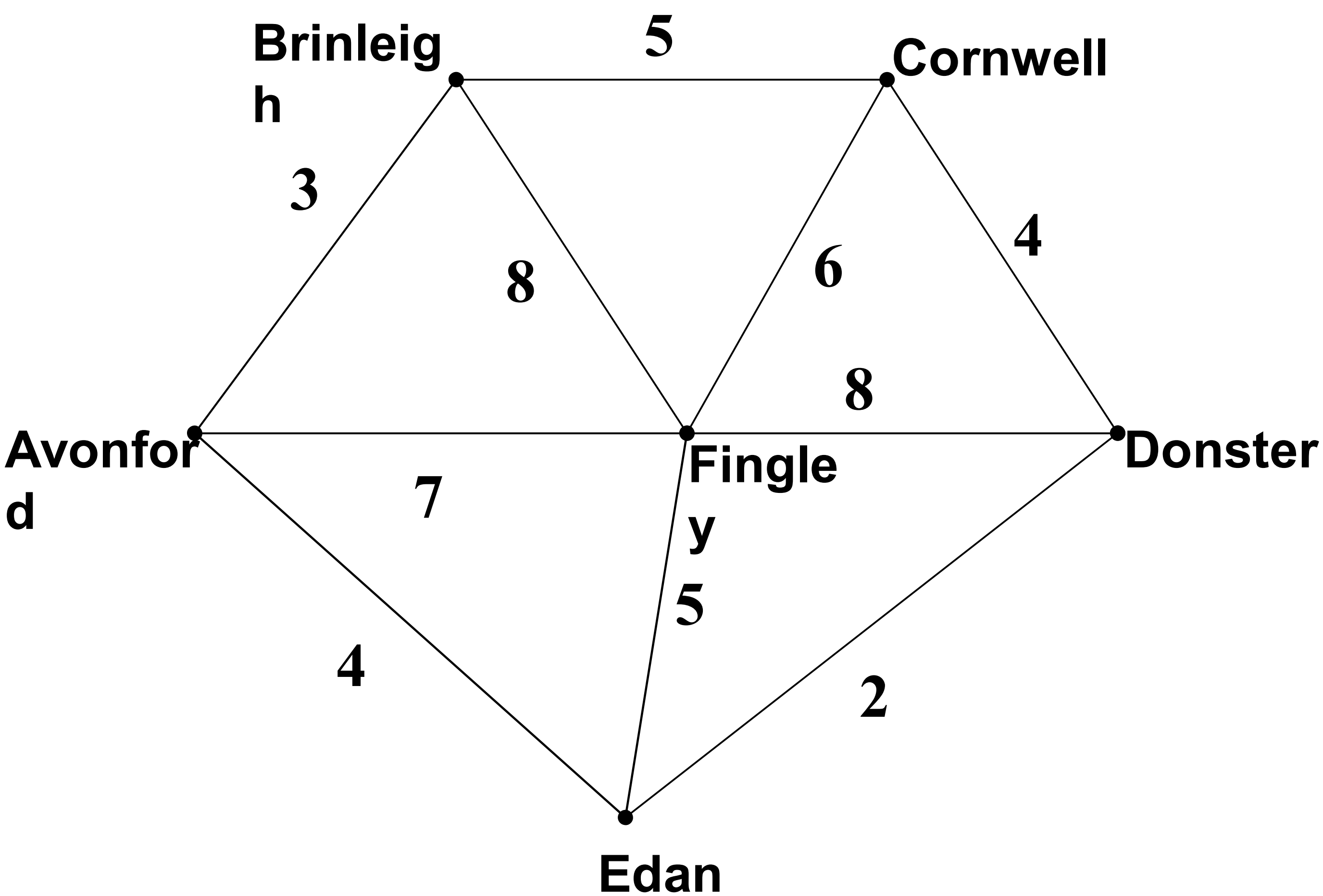
e:= any edge in *G* with smallest weight that does not form a simple circuit when added to *T*

*T*:= *T* with e added

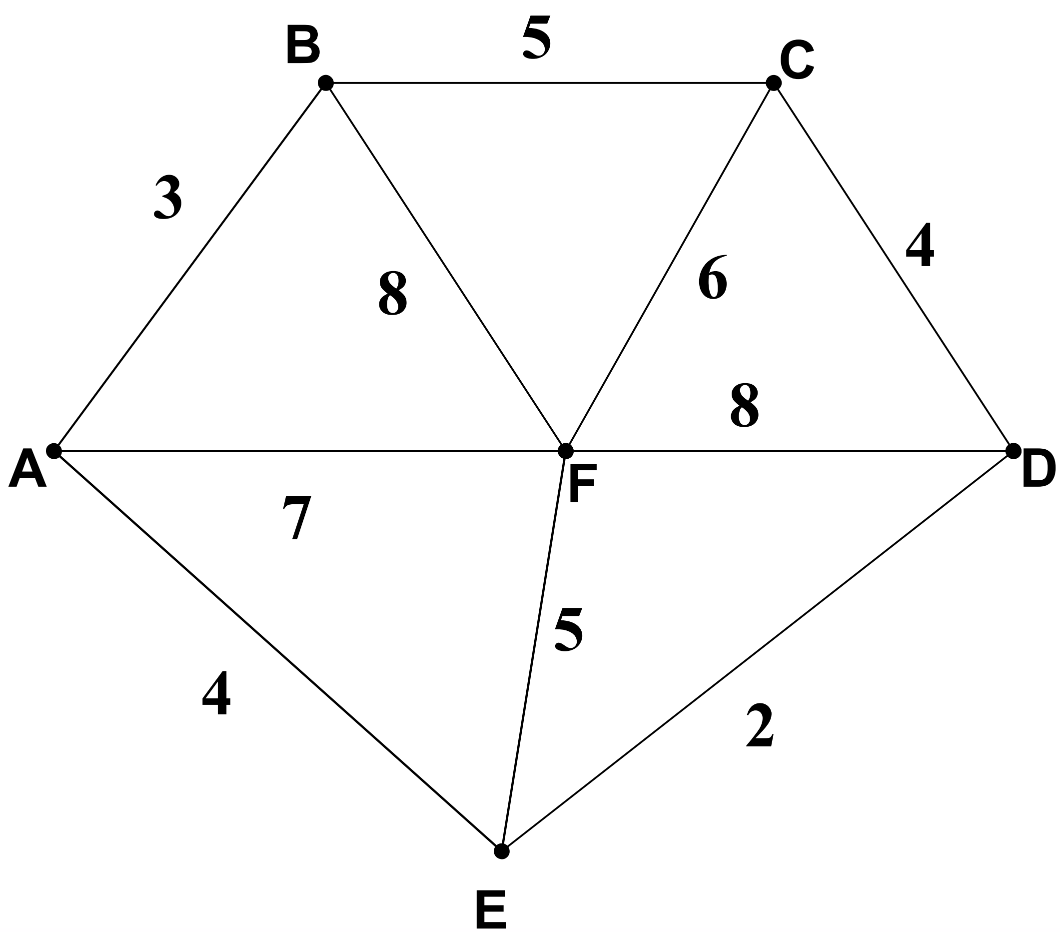
end (*T* is a minimum spanning tree of *G*)

# Example

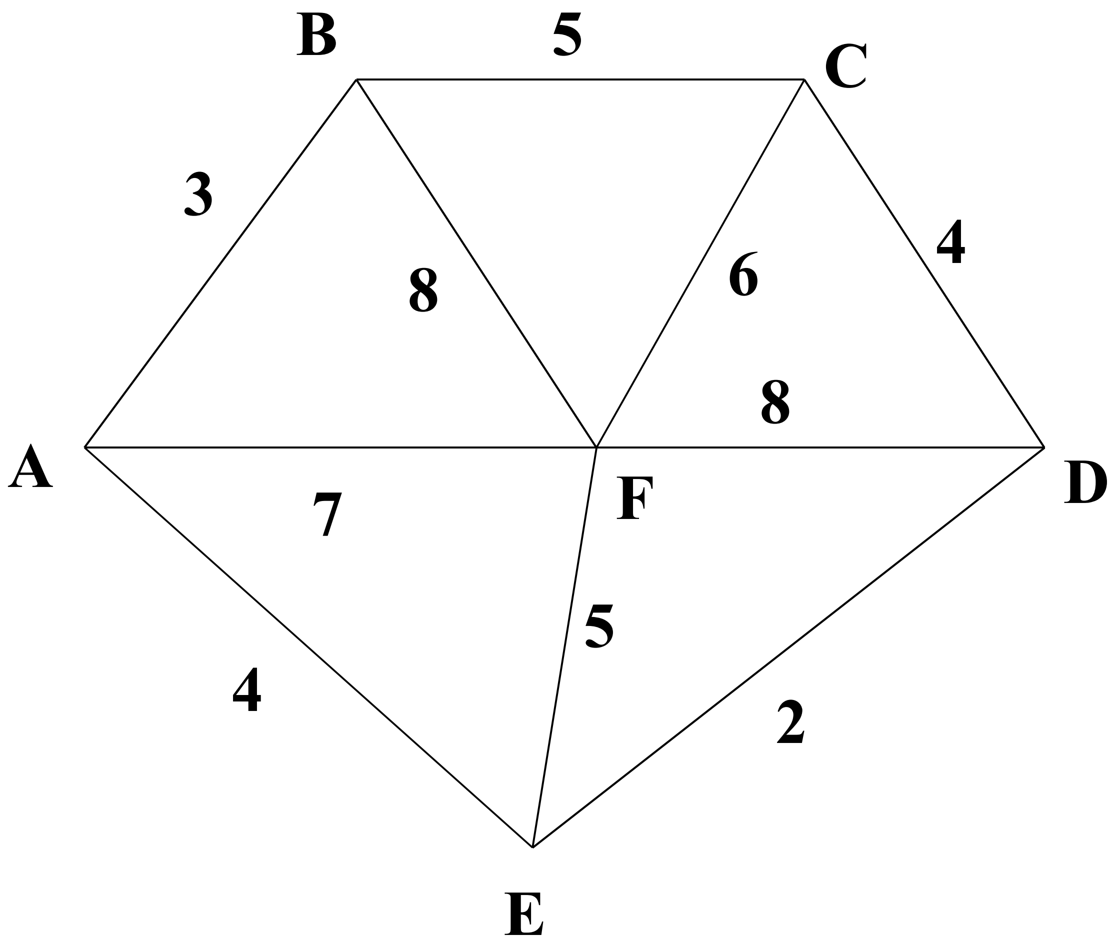
A cable company want to connect five villages to their network which currently extends to the market town of Avonford. What is the minimum length of cable needed?



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



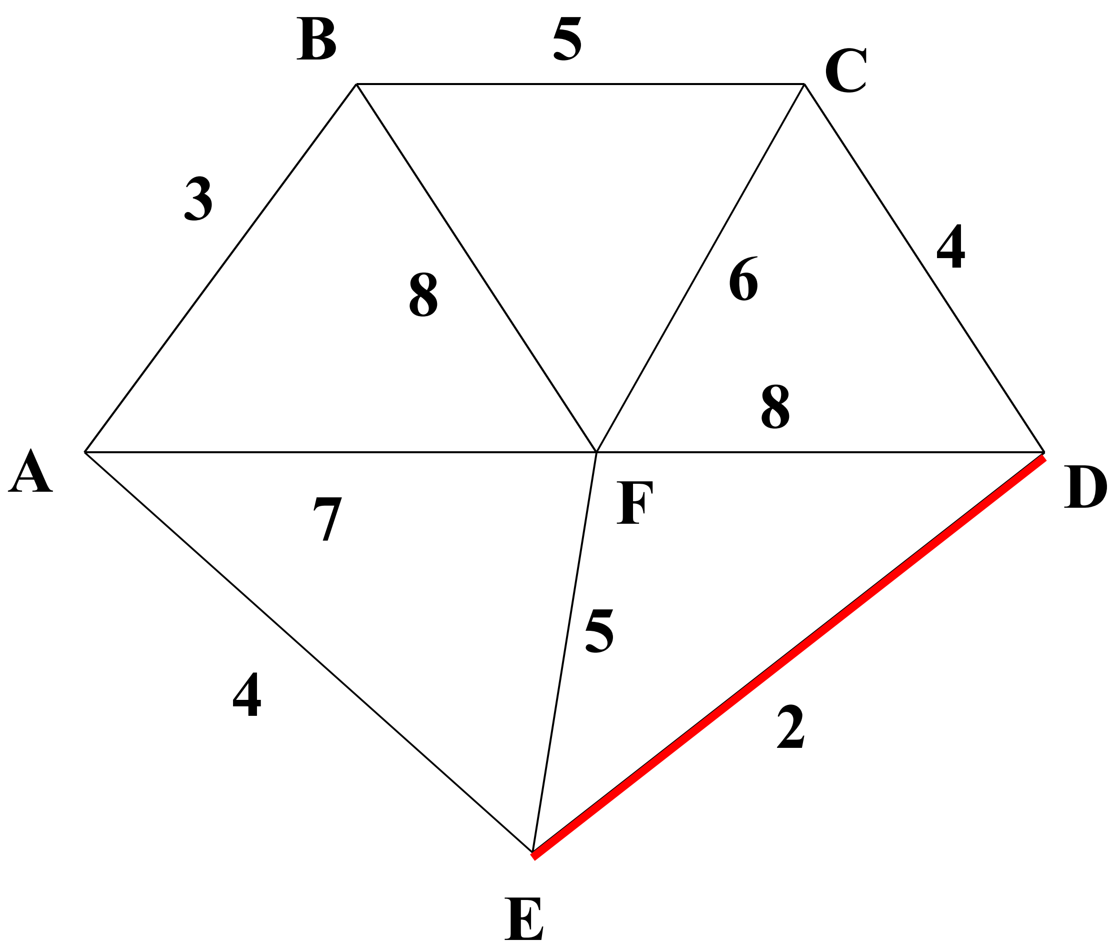
# Kruskal's Algorithm




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

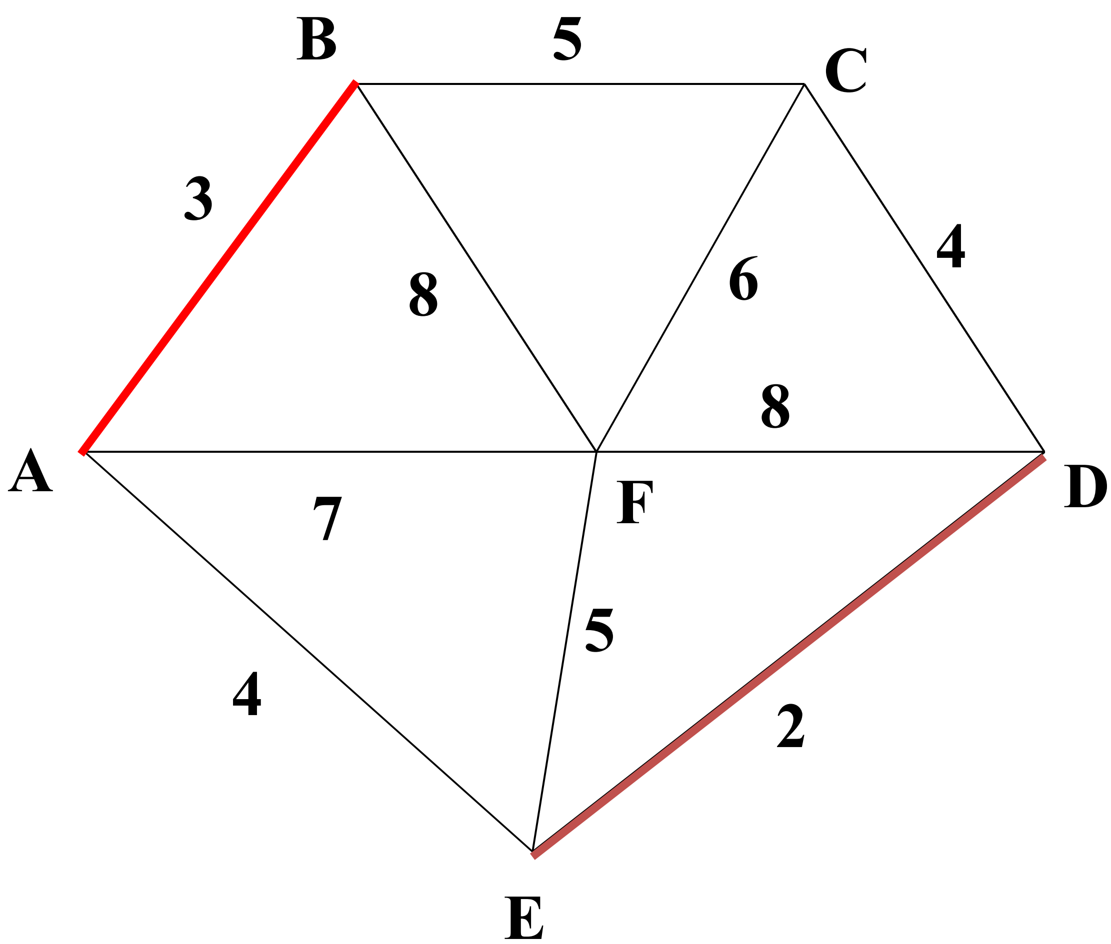
# Kruskal's Algorithm




**Select the shortest edge in the network**

**ED 2**

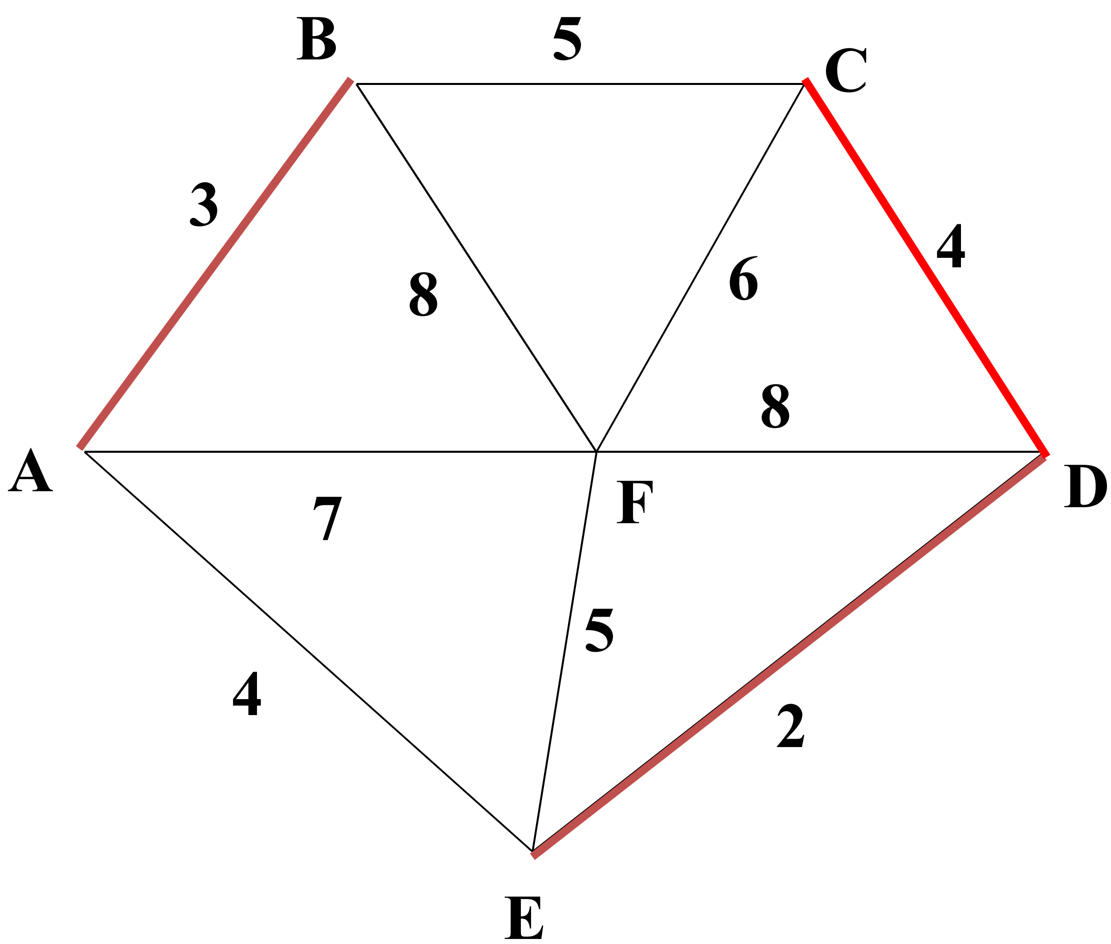
# Kruskal's Algorithm




**Select the next shortest edge which does not create a cycle**

**ED 2**
**AB 3**

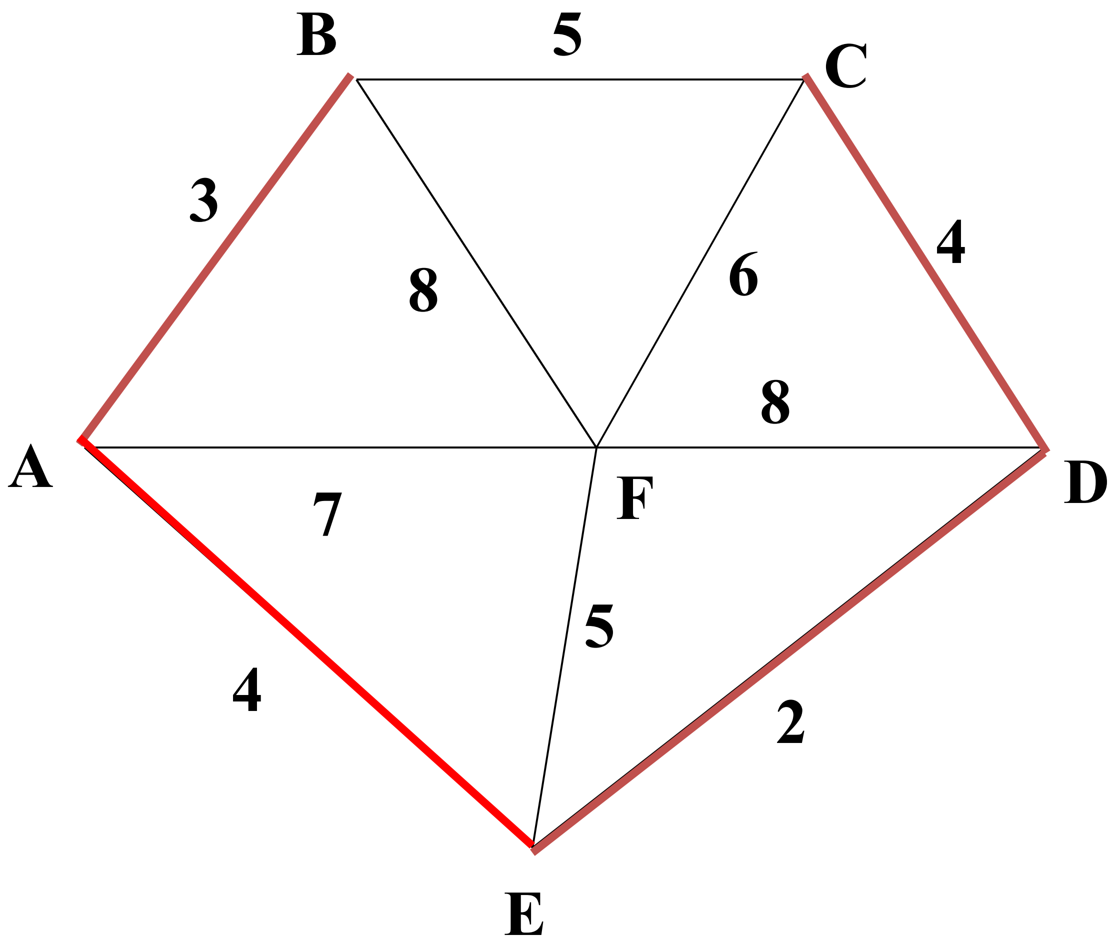
# Kruskal's Algorithm




**Select the next shortest edge which does not create a cycle**

**ED 2**
**AB 3**
**CD 4 (or AE 4)**

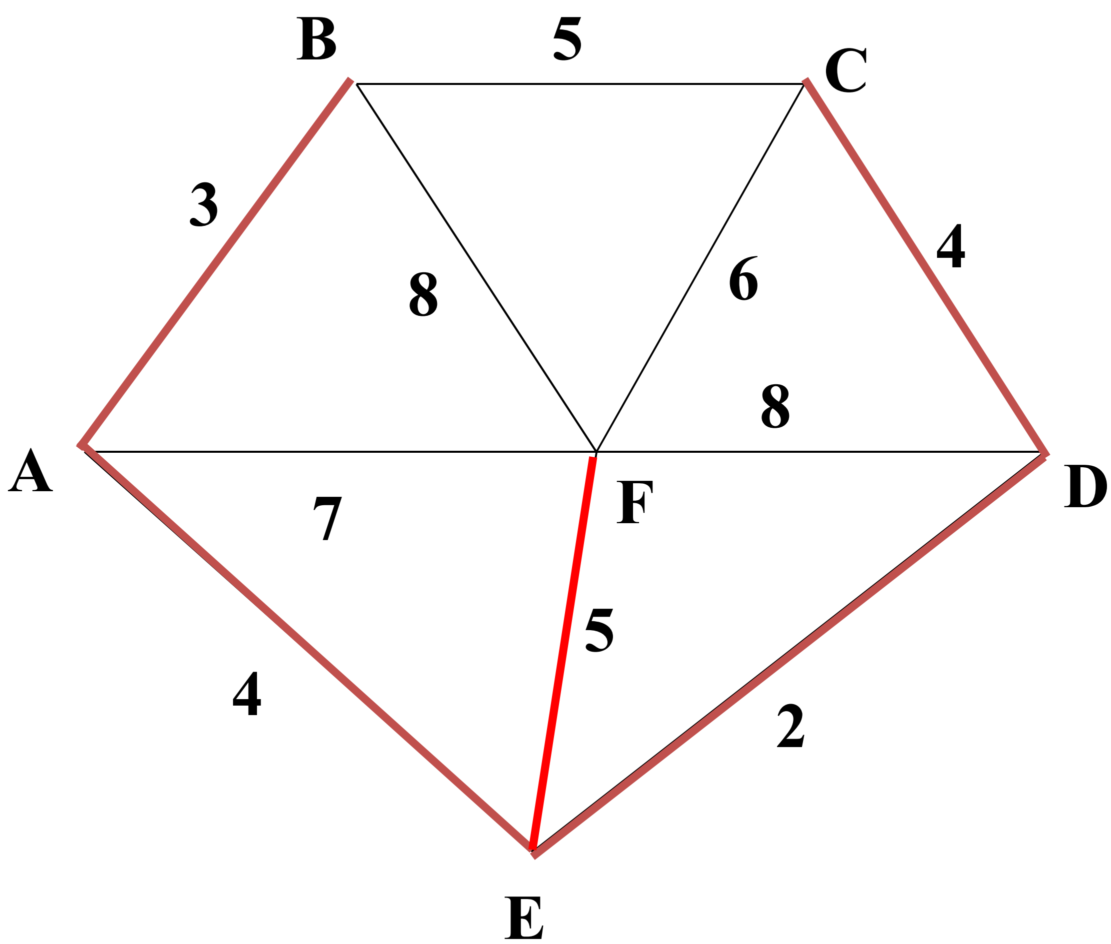
# Kruskal's Algorithm




Select the next shortest
edge which does not
create a cycle

ED 2
AB 3
CD 4
AE 4

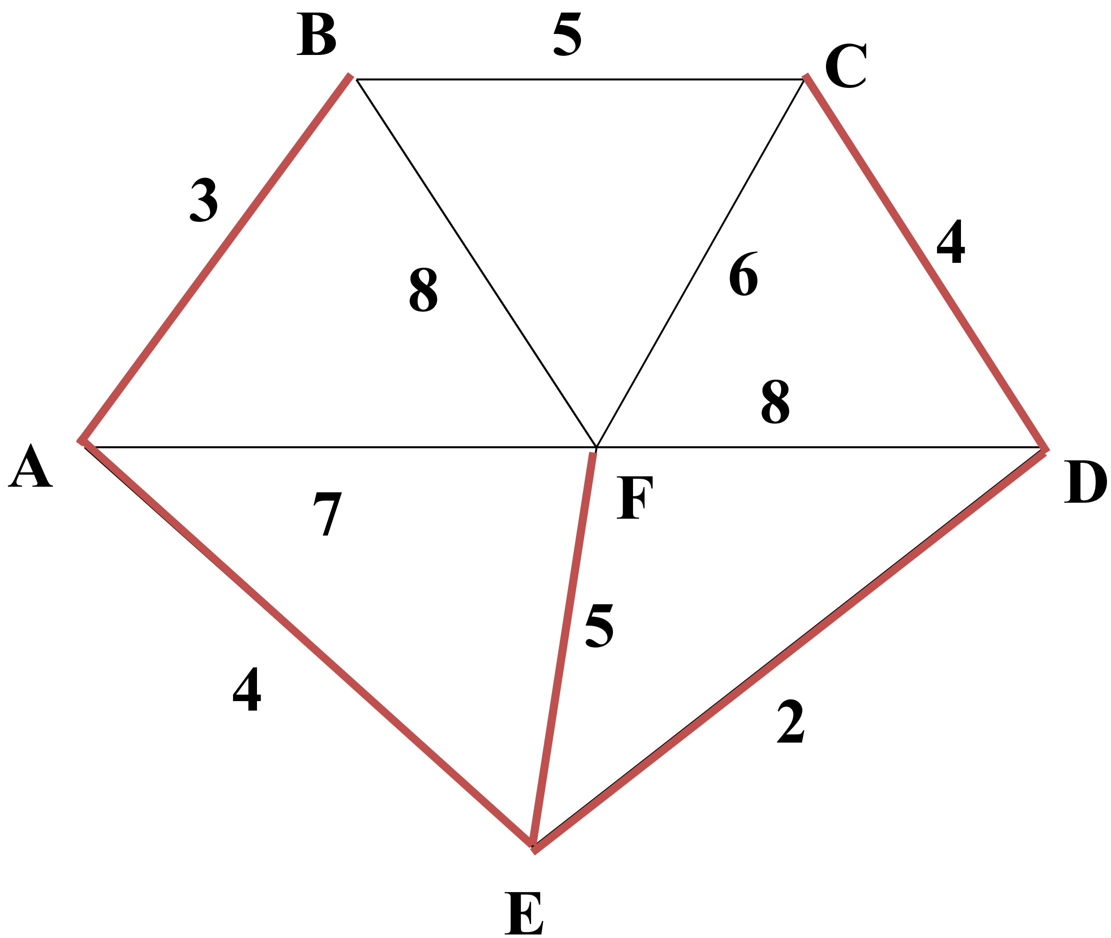
# Kruskal's Algorithm




**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**

# Kruskal's Algorithm




All vertices have been connected.

The solution is

ED 2
AB 3
CD 4
AE 4
EF 5

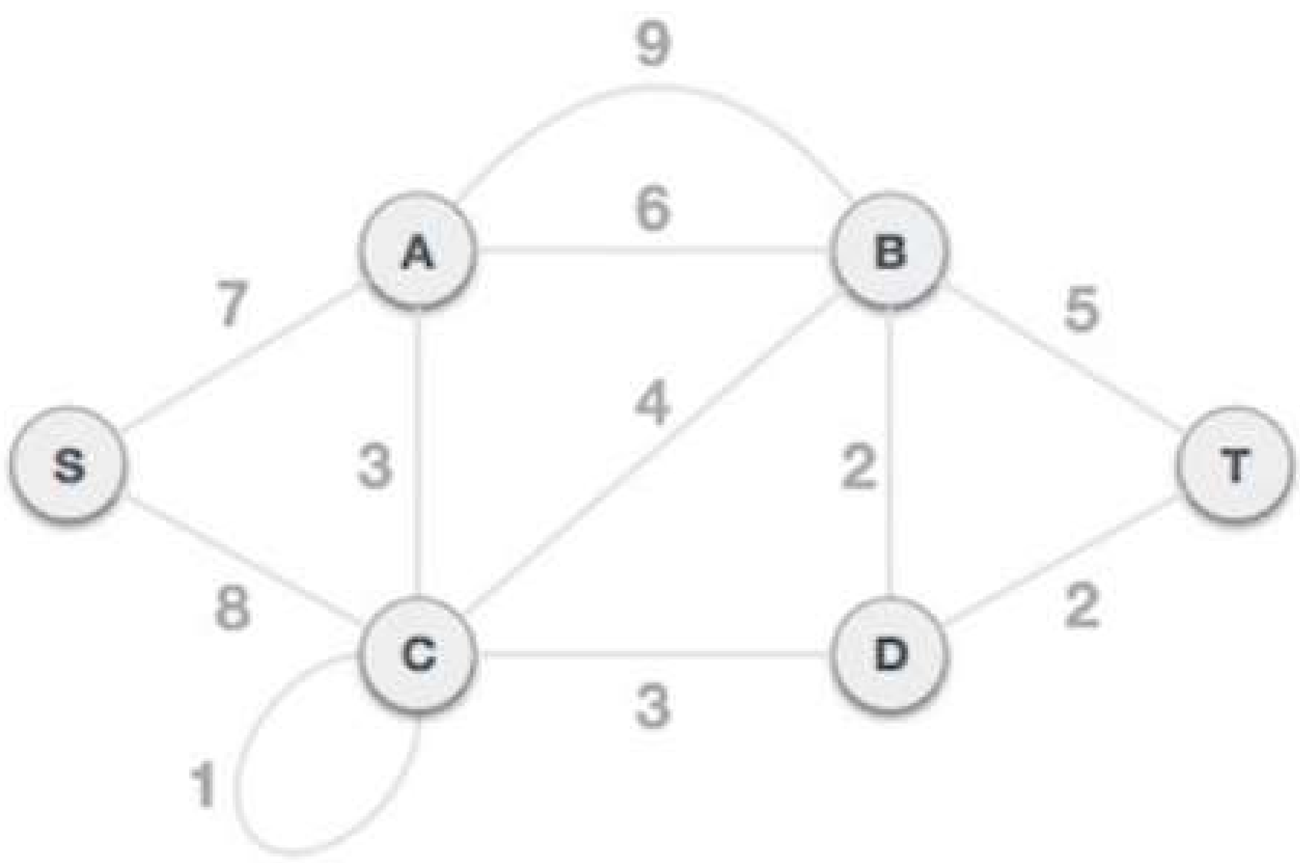
Total weight of tree: 18

# Kruskal’s Algorithm

Important notes:

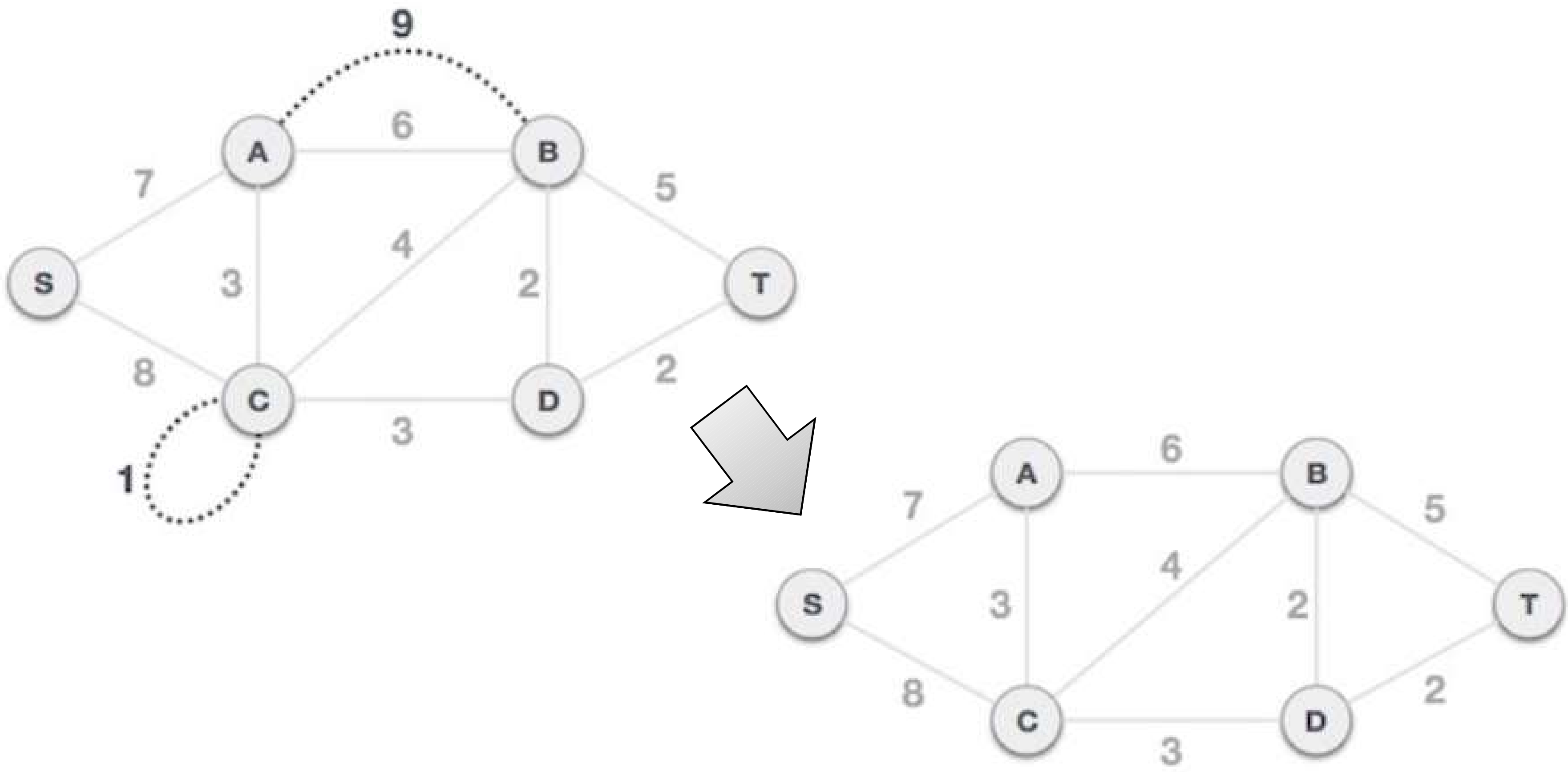
- The graph given should be a tree
  - Remove all loops (if any)
  - Remove all parallel edges
    - keep the one which has the least weight associated and remove all others

# Example



# Example

- Remove all loops and parallel edges



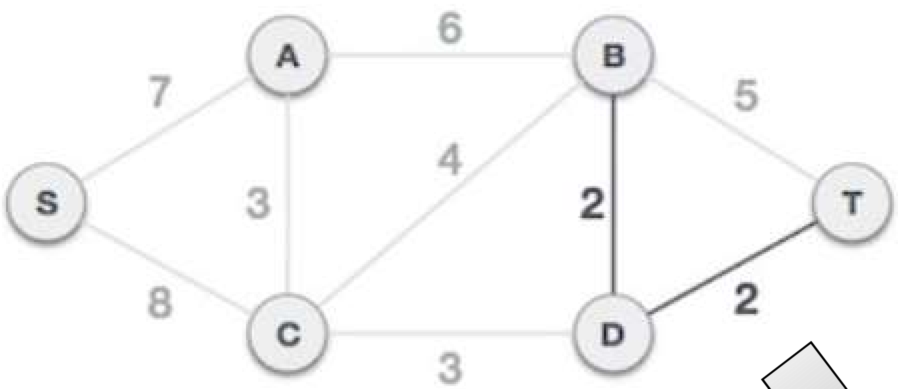
# Example

- Arrange all edges in their increasing order of weight

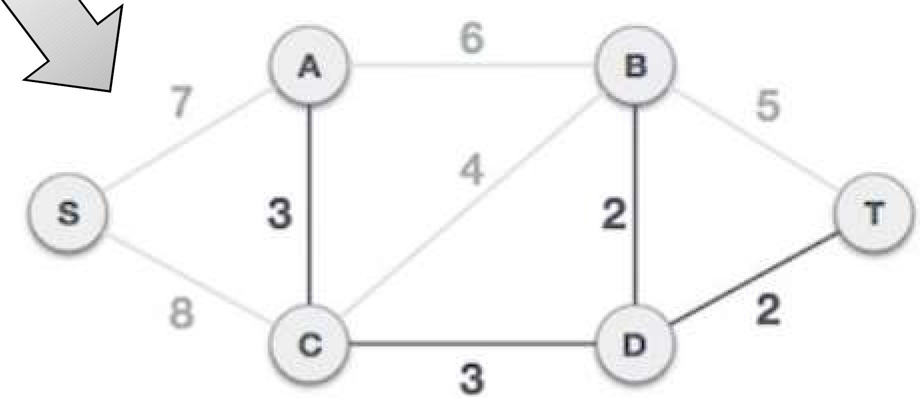
| BD | DT | AC | CD | CB | BT | AB | SA | SC |
|---|---|---|---|---|---|---|---|---|
| 2 | 2 | 3 | 3 | 4 | 5 | 6 | 7 | 8 |

# Example

- Add the edge which has the least weightage

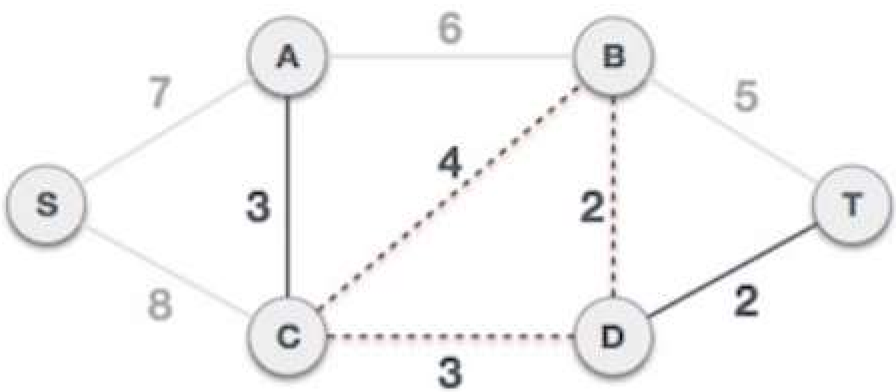



**The least weight is 2 and edges involved are BD and DT**



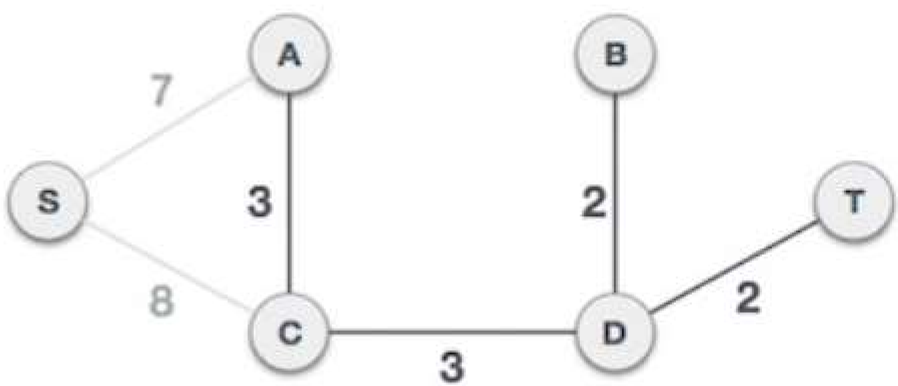

**Next weight is 3, and associated edges are AC and CD**

# Example




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

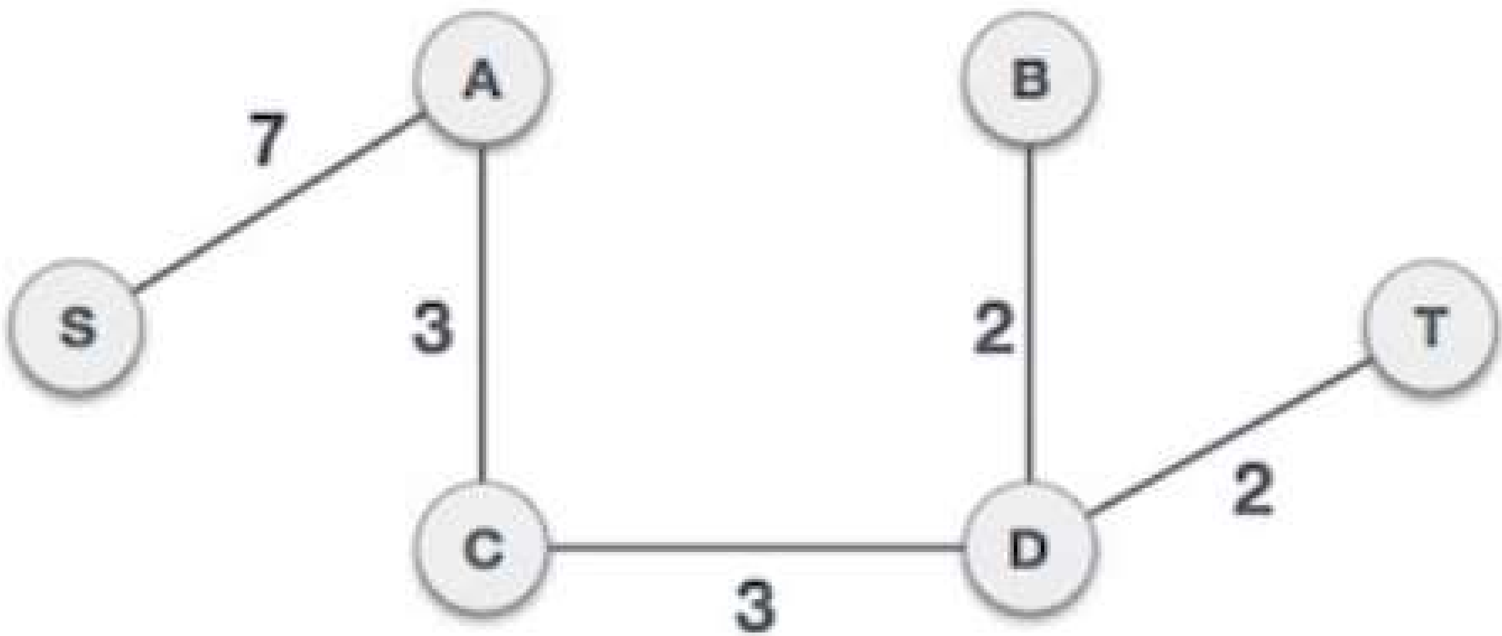
**We observe that edges with weight 5 and 6 also create circuits. We ignore them and move on.**




**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.**

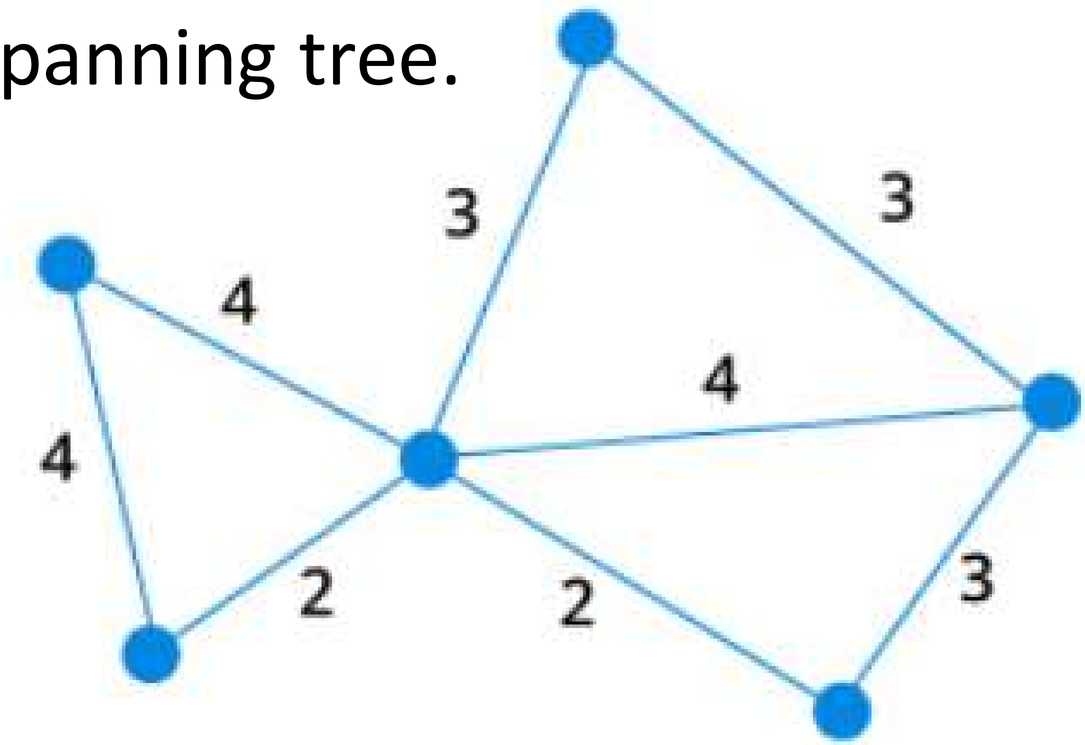
# Example

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



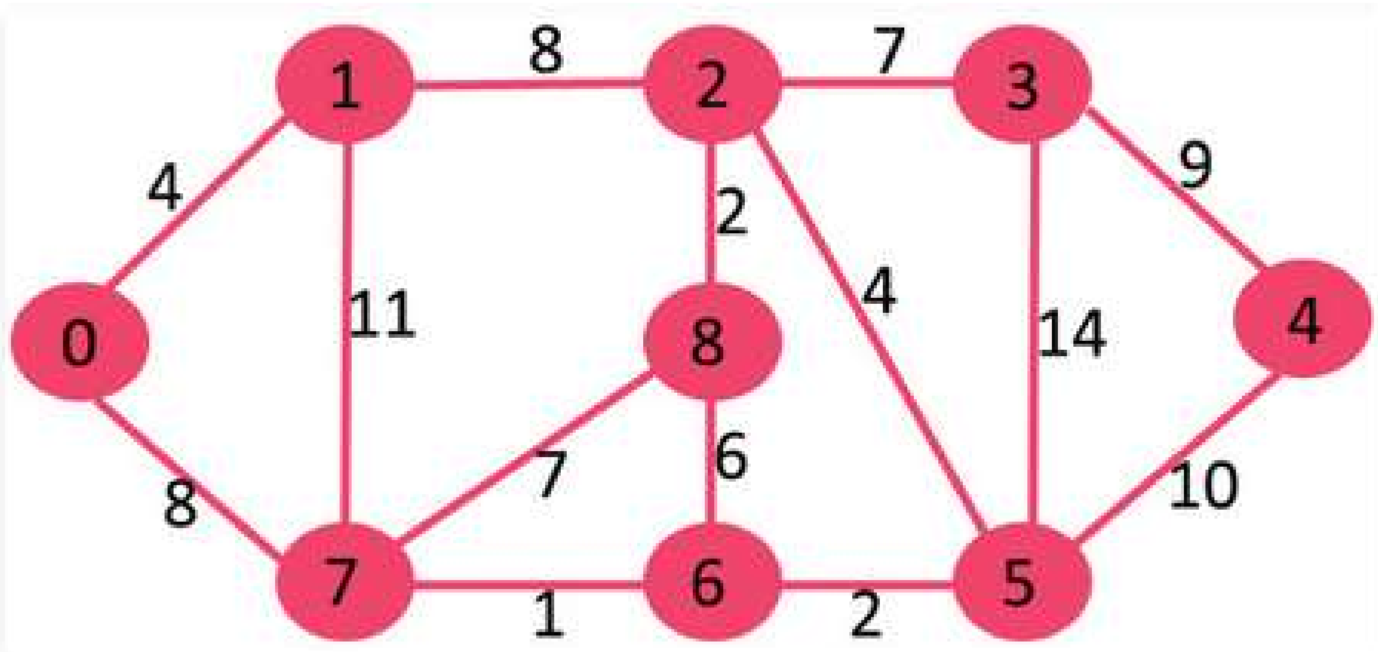
# Exercise

Find the minimum spanning tree using Kraskal’s Algorithm and give the total weight for the minimum spanning tree.



# Exercise

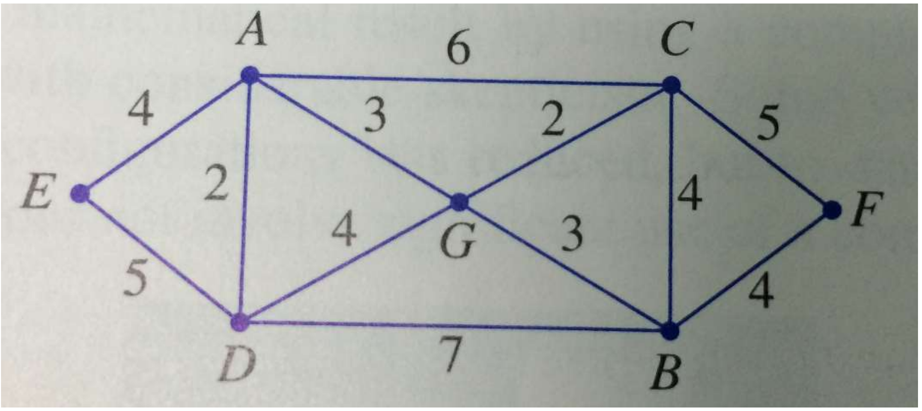
Find the minimum spanning tree using Kraskal's Algorithm and give the total weight for the minimum spanning tree.



# Exercise

**Use Kruskal's algorithm to find a minimal spanning tree for the following graphs.**

**a)**




**b)**