Graph Theory

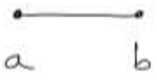
Def: A *graph* is a pair (V, E) of sets, where V is a non-empty set whose elements are called *vertices* and E is a set of unordered pairs of vertices from E, whose elements are called *edges*.

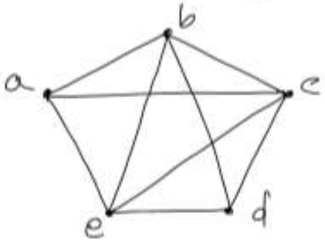
Def: A graph is *finite* if V is finite.

We'll only consider finite graphs.

Notes:

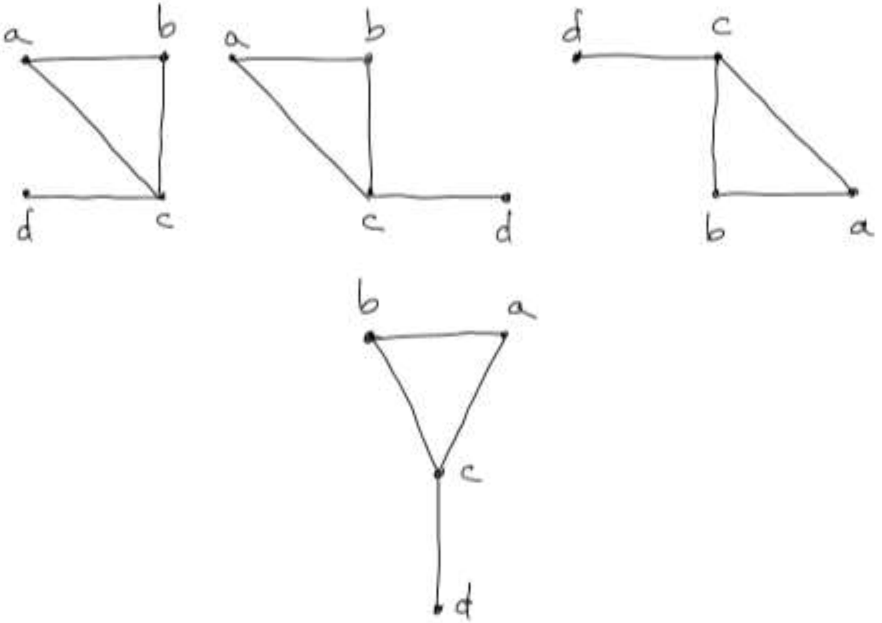
1. An edge consists of a pair of distinct vertices. Graphs which allow loops, ie. edges with two vertices the same are multi-graphs which we won't consider.
2. Vertices are represented by points, edges by lines (may be curved) joining these points. It's important to note that edges only meet at vertices, not where they cross.



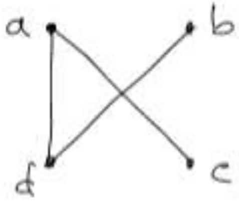


Examples: above

The way a graph in drawn is somewhat arbitrary. For eg. the following are the same:



Every edge joins two vertices called it's endpoints. An edge connects it's vertices or is incident upon them. Vertices where share an edge are adjacent.

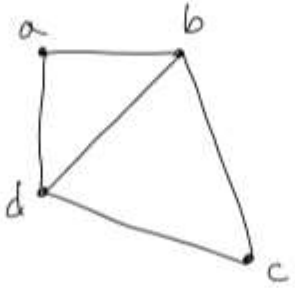


a and c are adjacent but b and c are not.

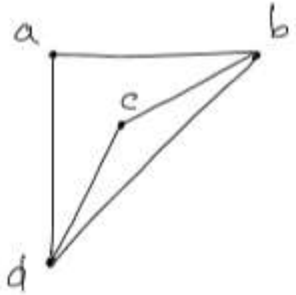
An eg. of a graph in set notation is

$$V = \{a, b, c, d\}, \quad E = \{\{a, b\}, \{a, d\}, \{c, d\}, \{c, b\}, \{b, d\}\}.$$

This graph has 4 vertices and 5 edges. We can represent it as



or



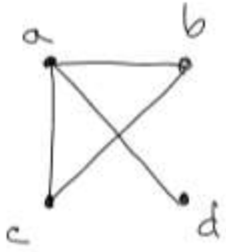
Def: The *degree* of a vertex in a graph G is the no. of edges incident upon it. If a is a vertex in G we denote it's degree by deg(a).

Def: A vertex is *even* or *odd* according to whether it's degree is even or odd.

Def: A grap G is simple if at most one edge connects any pair of vertices.

We'll only consider simple undirected graphs. In a simple graph an edge in uniquely identified by it's vertices.

Eg.



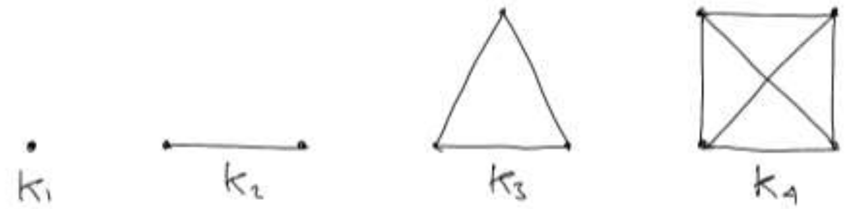
$\deg(a) = 3$ $\deg(c) = 2$

$\deg(b) = 2$ $\deg(d) = 1$

Theorem (Handshaking theorem): In a simple undirected graph, the sum of the degrees of all the vertices is twice the no. of edges.

Proof: Each edge contributes twice to the sum of the degrees, once for each endpoint.

Def: A simple grap is complete if every pair of vertices has an edge incident on them. For each $n \in \mathbb{Z}^+$ there is only one complete graph, denoted K_n.

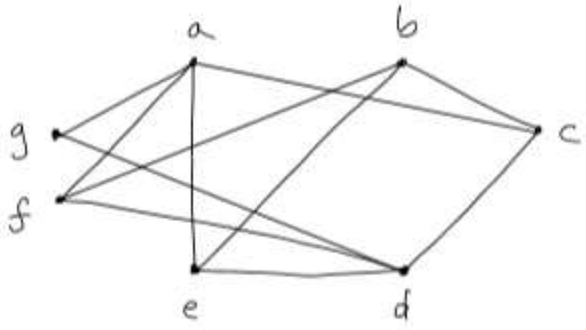


Def: A grap $G = (V, E)$ is <u>bipartite</u> if V can be partitioned into two disjoint sets V_1 and V_2 in such a way that every edge is incident on a vertex in V_1 and a vertex in V_2. That is no edge connects two vertices in V_1 or V_2.

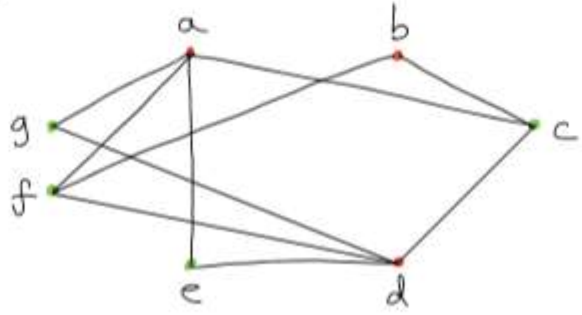
K_3 is not bipartite, but K_2 is.

Theorem: A simple graph is bipartite if and only if it's possible to colour the vertices, using exactly two colours in such a way that no adjacent vertices have the same colour.

Eg. Is

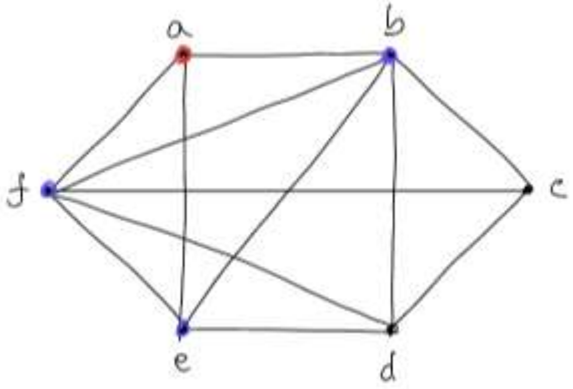


bipartite?



Yes, it is bipartite.

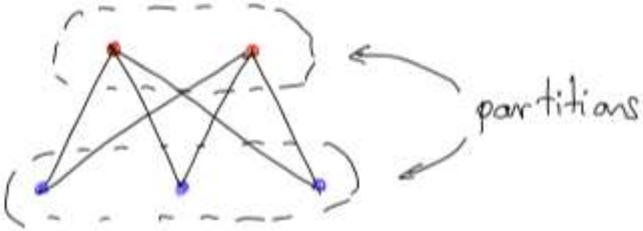
Eg. Is



bipartite? No, since for eg. b and f above have the same colour.

A graph is complete bipartite if it is bipartite and each vertex in one partition is adjacent to every vertex in the second. If m, n are the no. of vertices in the two partitions then $K_{m,n}$ is the unique complete bipartite graph with those partitions.

Eg. $K_{2,3}$ is



Paths in Graphs

A path in a graph is a sequence of edges that connect two vertices. Each edge in the sequence must start at the endpoint of the previous edge. If the start and end vertex are the same we say it's a closed path, a circuit or a cycle.

Note that a path can visit the same vertex more than once.

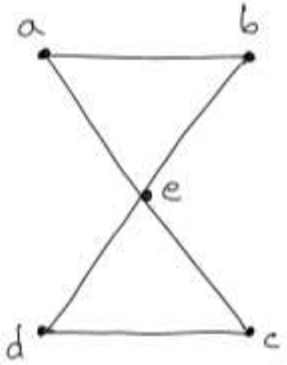
A path is simple if it does not contain the same edge more than once.

A graph is connected if there is a path joining every pair of distinct vertices.

Theorem: In a connected graph (undirected) there is a simple path between every pair of distinct vertices.

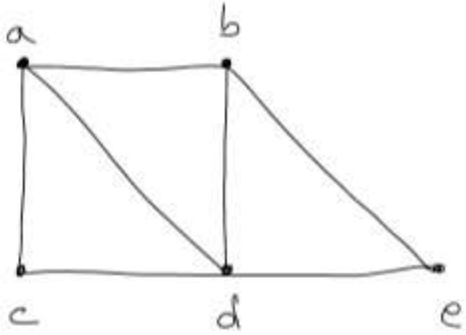
An *Euler path* in a graph is a simple path containing every edge in the graph. An Euler path is an *Euler cycle* or *circuit* if it starts and ends at the same vertex.

Eg.



a, e, c, d, e, b, a is an Euler cycle.

Eg.



a, c, d, e, b, d, a, b is an Euler path but there is no Euler cycle.

Suppose we start at a. Three edges are incident on a. We leave a twice but only return once so we never return to a. If instead we start at another vertex then we arrive at a twice but only leave once so we never return to the starting vertex.

Theorem: A connected, undirected graph has an Euler cycle if and only if every vertex has even degree.

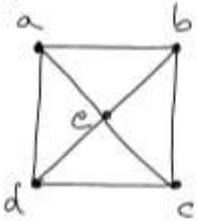
Theorem: A connected, undirected graph has an Euler path if and only if there are exactly two vertices of odd degree.

For the previous example we have

	a	b	c	d	e
deg	3	3	2	4	2

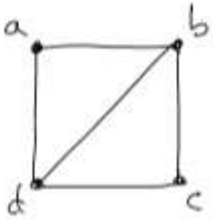
Exactly two vertices with odd degree so this graph has an Euler path but no Euler cycle.

Examples:



	a	b	c	d	e
deg	3	3	3	3	4

No Euler path or cycle.



	a	b	c	d
deg	2	3	2	3

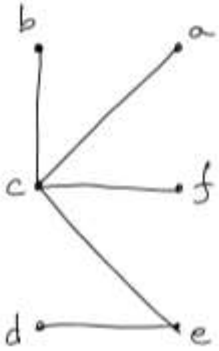
Exactly two vertices with odd degree. has an Euler path but no Euler cycle.

Weighted Graphs

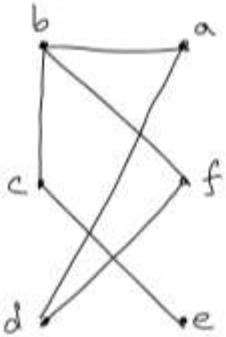
In a *weighted graph* each edge is assigned a number called a weight. In a graph representing a computer network this could be traffic, leasing cost, capacity etc.

We're often interested in paths of *lowest weight* ie where the combined weight of the edges in the path are a min. Another item of interest is a *minimal weight spanning tree*.

if and only if
An undirected graph is a *tree* iff there is a unique simple path between any two vertices.



A tree



Not a tree
(multiple paths between
a and b).

A spanning tree of a graph G is a subgraph of G (ie a subset of G which is itself a graph) which is a tree containing every vertex of G.

There are various algorithms for producing a minimal weight spanning tree for a graph. We'll look at kruskal's algorithm.

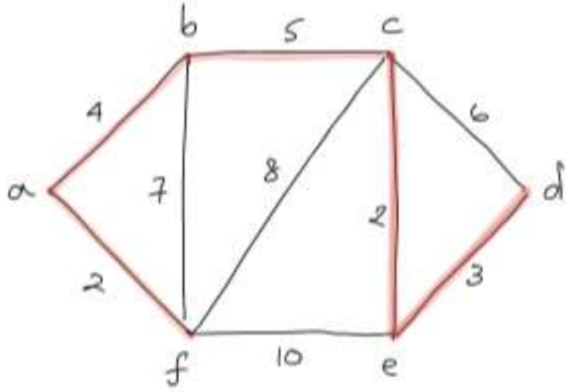
Kruskal's Algorithm:

Step 1: Pick any edge in the graph with lowest weight and mark it in some way to indicate it's in the tree.

Step 2: Choose any edge in the graph, not already in the tree which has smallest weight and which does not close a cycle with edges already in the graph. Add it to the tree by marking it as in step one.

Step 3: Repeat step 2 until every vertex is in the tree. The tree now spans the graph and has minimal weight.

Example: Find a minimal weight spanning tree for the weighted graph:



The minimal weight spanning tree has weight 16.

Dijkstra's Shortest Path Algorithm:

Given a starting and ending vertex, find the shortest (ie minimal weight) path between them.

Algorithm is as follows:

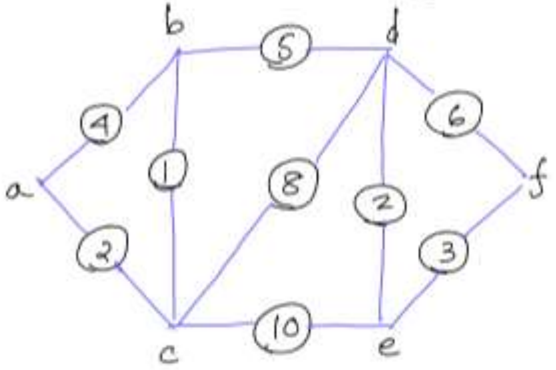
① Assign the starting vertex distance 0.

② Mark all other vertices as having distance ∞ (ie currently unknown).

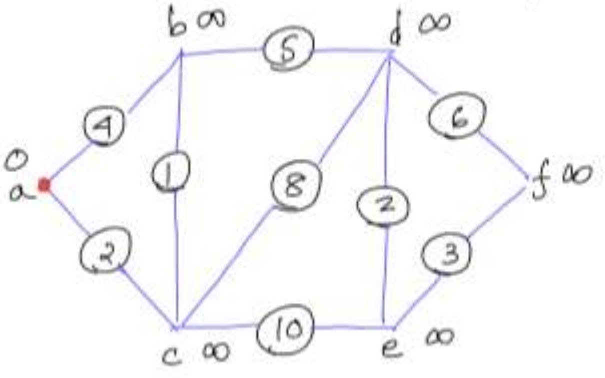
③ Mark the starting vertex as being in the set of vertices whose shortest distance from it has been calculated. Call the set of vertices whose shortest distance from the starting vertex has been calculated, D.

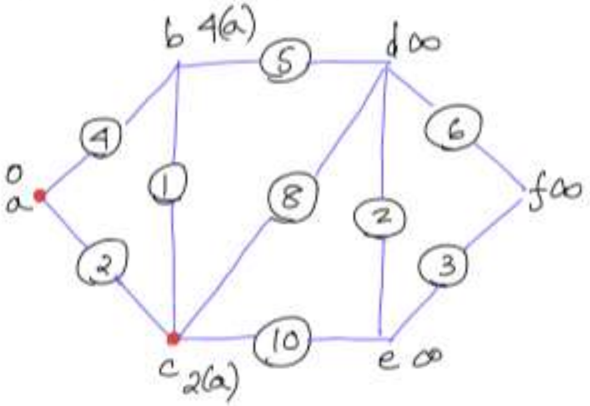
④ For each vertex adjacent to D calculate it's shortest distance to the starting vertex using only the vertices in D. We only need to calculate the distances from members of D adjacent to the vertex.

Example: Given the weighted graph



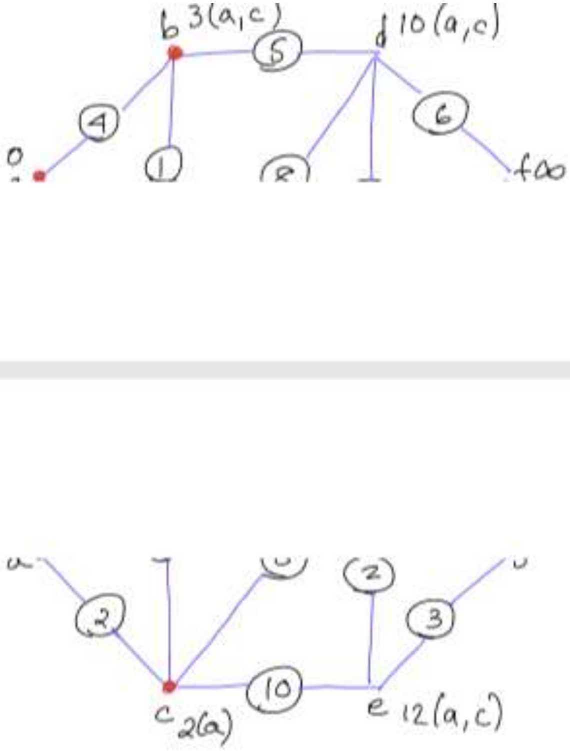
find a minimum weight path from a to f.



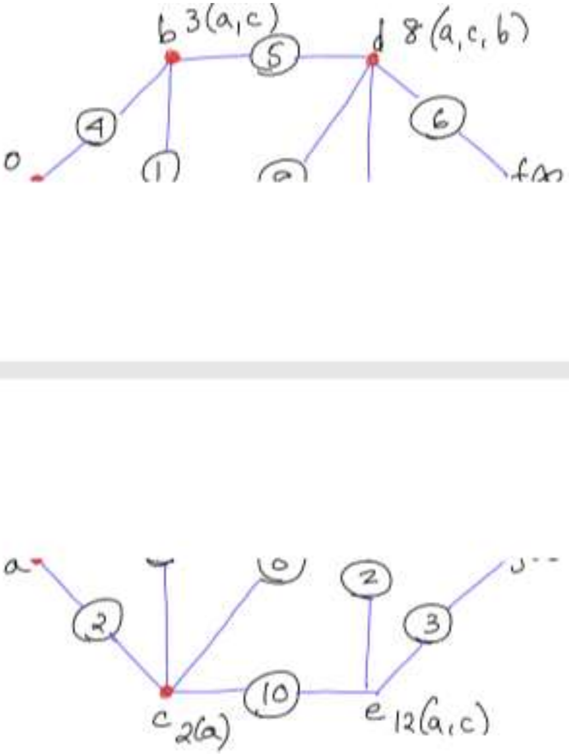


D = {a}

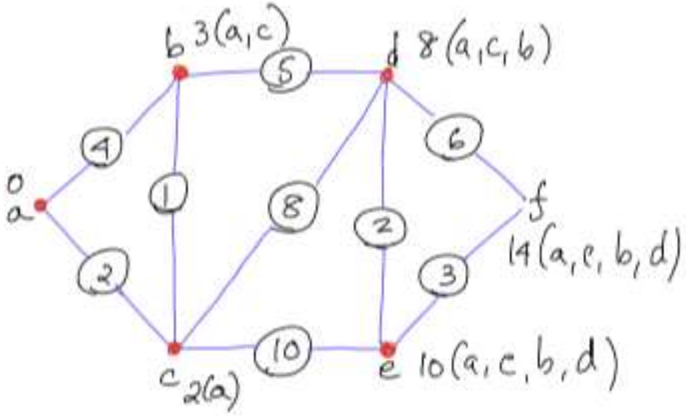
D = {a, c}


b 3(a,c)
5
d 10(a,c)
4
6
0
1
8
f∞
a
2
2
3
c 2(a)
10
e 12(a,c)

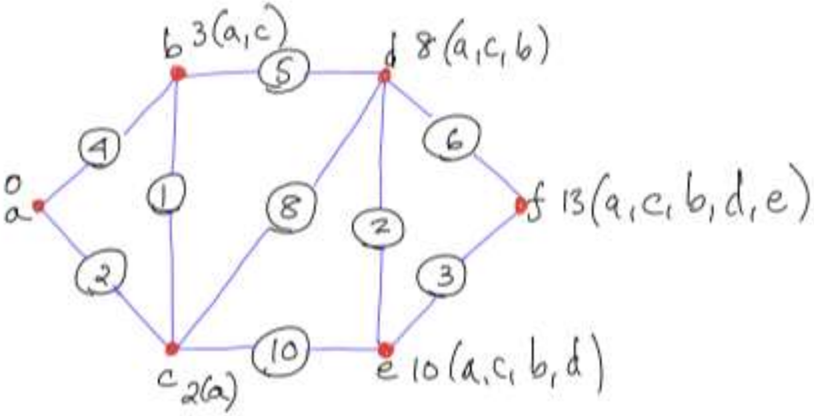
D = {a, c, b}


b 3(a,c)
5
d 8(a,c,b)
4
6
0
1
f∞
a
2
2
3
c 2(a)
10
e 12(a,c)

D = {a, c, b, d}



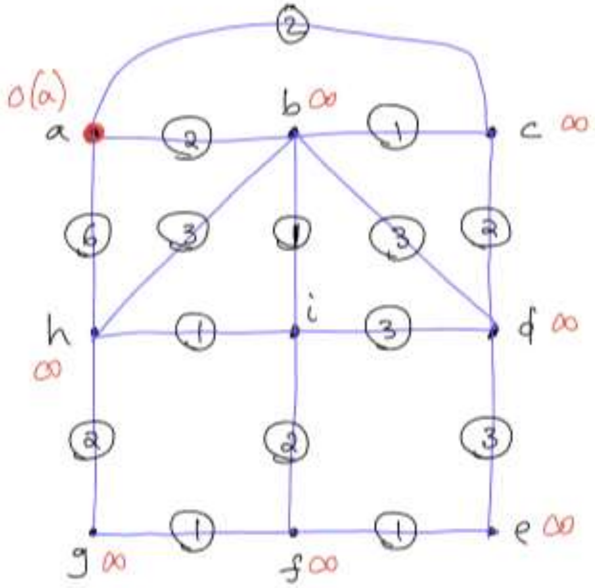
D = {a, c, b, d, e}



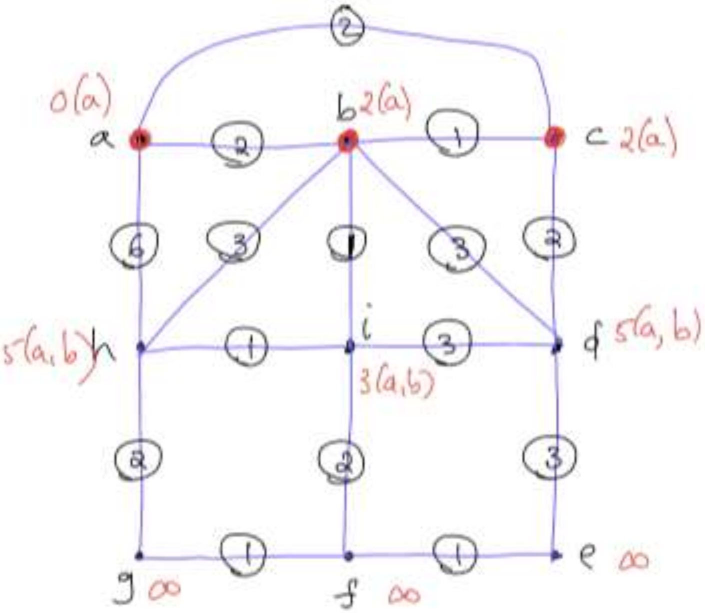
D = {a, c, b, d, e, f}

Min. path in 13 via a, c, d, e, f.

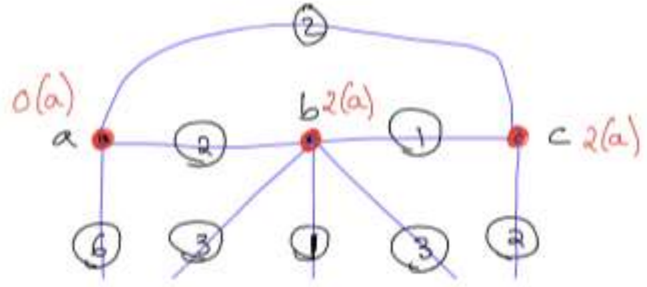
Example Using Dijkstra's algorithm find a shortest path from a to e in the following graph:



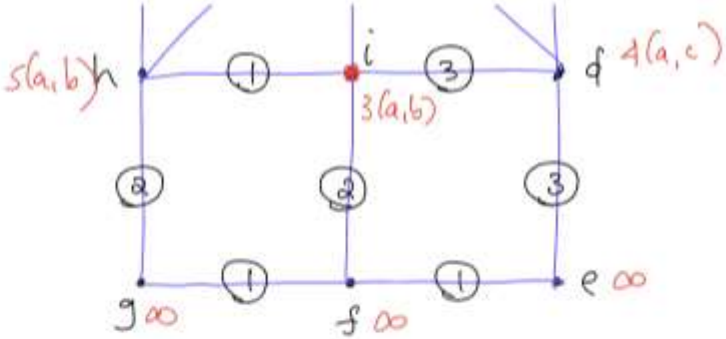
$D = \{a\}$

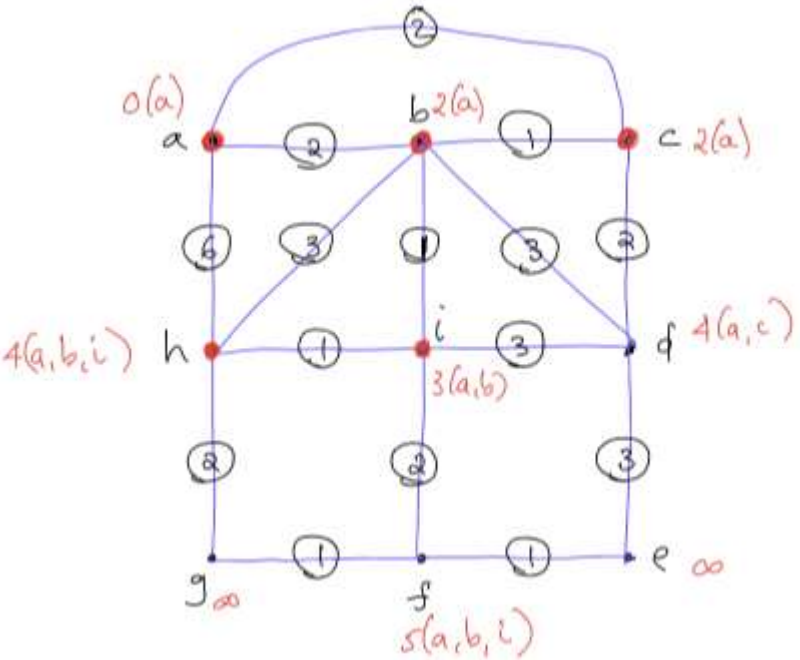


D={a,b}

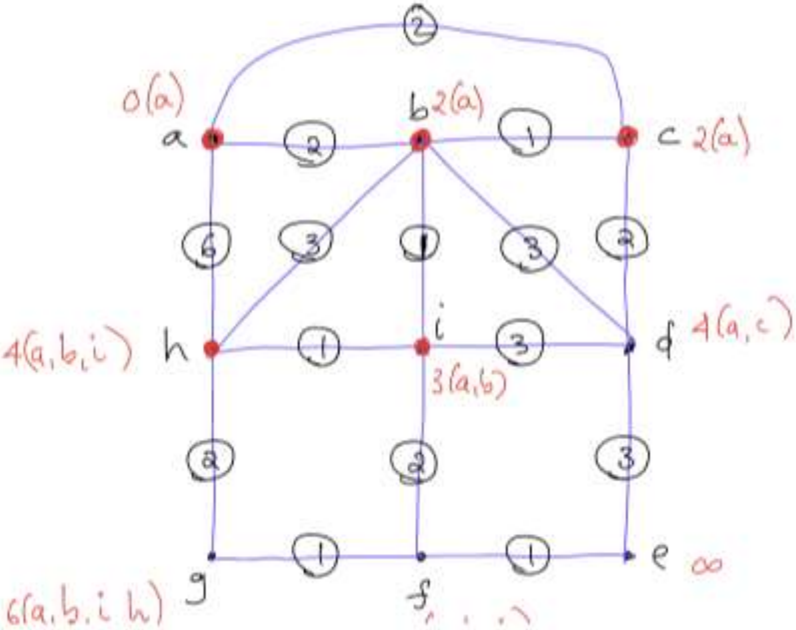


D= {a,b,c}





D = {a, b, c, i}



D={a,b,c,i,h}

Graphs & Matrices

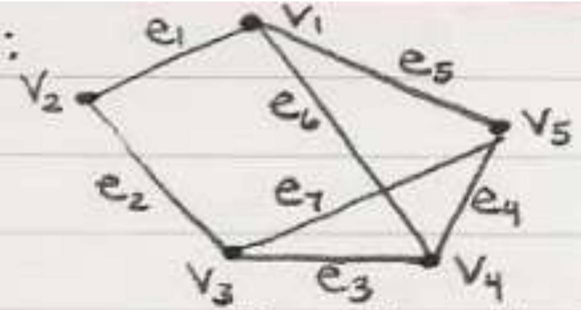
- **Adjacency Matrix:** the $n \times n$ matrix where,

$$a_{ij} = \begin{cases} 1 \; if \; v_i \; is \; adjacent \; to \; v_j \\ 0 \qquad\qquad\qquad\qquad otherwise \end{cases}$$

- **Incidence Matrix:** the $n \times m$ matrix where,

$$b_{ij} = \begin{cases} 1 \; if \; v_i \; is \; adjacent \; to \; e_j \\ 0 \qquad\qquad\qquad\qquad otherwise \end{cases}$$

G:



$$A(G): \begin{array}{c|ccccc} & v_1 & v_2 & v_3 & v_4 & v_5 \\ \hline v_1 & 0 & 1 & 0 & 1 & 1 \\ v_2 & 1 & 0 & 1 & 0 & 0 \\ v_3 & 0 & 1 & 0 & 1 & 1 \\ v_4 & 1 & 0 & 1 & 0 & 1 \\ v_5 & 1 & 0 & 1 & 1 & 0 \end{array}$$

A(G) is the adjacency matrix of G

$$B(G): \begin{array}{c|ccccccc} & e_1 & e_2 & e_3 & e_4 & e_5 & e_6 & e_7 \\ \hline v_1 & 1 & 0 & 0 & 0 & 1 & 1 & 0 \\ v_2 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ v_3 & 0 & 1 & 1 & 0 & 0 & 0 & 1 \\ v_4 & 0 & 0 & 1 & 1 & 0 & 1 & 0 \\ v_5 & 0 & 0 & 0 & 1 & 1 & 0 & 1 \end{array}$$

B(G) is the incidence matrix of G