* Chapter 06 *

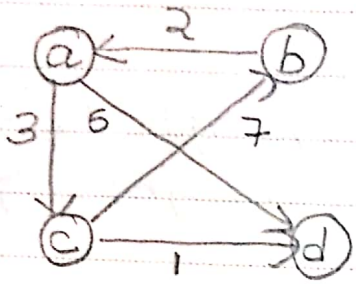
Graph Algorithms

Prakash Hegade

Examples and Tracing Only.

PAGE NO. :
DATE : / /

Floyd's algorithm:



$D^{(0)} =$

	a	b	c	d
a	0	∞	3	∞
b	2	0	∞	∞
c	∞	7	0	1
d	6	∞	∞	0

$D^{(1)} =$

	a	b	c	d
a	0	∞	3	∞
b	2	0	5	∞
c	∞	7	0	1
d	6	∞	9	0

$D^{(2)} =$

	a	b	c	d
a	0	∞	3	∞
b	2	0	5	∞
c	9	7	0	1
d	6	∞	9	0

$D^{(3)} =$

	a	b	c	d
a	0	10	3	4
b	2	0	5	6
c	9	7	0	1
d	6	16	9	0

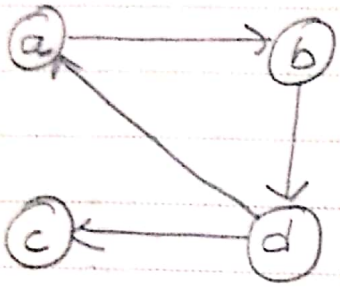
$D^{(4)} =$

	a	b	c	d
a	0	10	3	4
b	2	0	5	6
c	7	7	0	1
d	6	16	9	0

Presents the all pair shortest path matrix

PAGE NO :
DATE : / /

* Warshall's Algorithm *



$R^{(0)}$ =

	a	b	c	d
a	0	1	0	0
b	0	0	0	1
c	0	0	0	0
d	1	0	1	0

$R^{(1)}$ =

	a	b	c	d
a	0	1	0	0
b	0	0	0	1
c	0	0	0	0
d	1	1	1	0

$R^{(2)}$ =

	a	b	c	d
a	0	1	0	1
b	0	0	0	1
c	0	0	0	0
d	1	1	1	1

$R^{(3)}$ =

	a	b	c	d
a	0	1	0	1
b	0	0	0	1
c	0	0	0	0
d	1	1	1	1

$R^{(4)}$ =

	a	b	c	d
a	1	1	1	1
b	1	1	1	1
c	0	0	0	0
d	1	1	1	1

We can make the algorithm run faster by treating matrix row as bit strings and employ the bitwise or operation available in most modern computer languages.

PAGE NO. :
DATE : / /

Union-find : (Disjoint set union)

If we have a set of N elements which are partitioned into further subsets, and if we have to keep track of connectivity of each element in a particular subset or connectivity of subsets with each other, we use union-find.

Union(A,B) - connect two elements A and B
Find(A,B) - find, if there is any path connecting two elements A and B.

Example :

Arr

0	1	2	3	4	5	6	7	8	9
0	1	2	3	4	5	6	7	8	9

Union(2,1)

Arr

0	1	1	3	4	5	6	7	8	9
0	1	2	3	4	5	6	7	8	9

Union(4,3)
Union(8,4)
Union(9,3)
Union(6,5)

Arr

0	1	1	3	3	5	6	7	8	9
0	1	2	3	4	5	6	7	8	9

5 3 3

DATE : / /

We now have 5 subsets.

{0}
{1,2}
{3,4,8,9}
{5, 6}
{7}

Each subset is a connected component.

find(0,7) → false
find(8,9) → true

We have,

Arr	0	1	1	3	3	5	5	7	3	3
	0	1	2	3	4	5	6	7	8	9

Union(5,2)

Arr	0	1	1	3	3	1	1	7	3	3
	0	1	2	3	4	5	6	7	8	9

Code snippets:

```
int find(int arr[], int a, int b)
{
    if(arr[a] == arr[b])
        return 1;
    else
        return 0;
}
```

PAGE NO. :
DATE : / /

```
void union( int arr[], int n, int a, int b)
{
    int temp = arr[a];
    int i;
    for(i=0; i<n; i++)
    {
        if(arr[i]==temp)
            arr[i]=arr[b];
    }
}
```

Kruskal's Algorithm

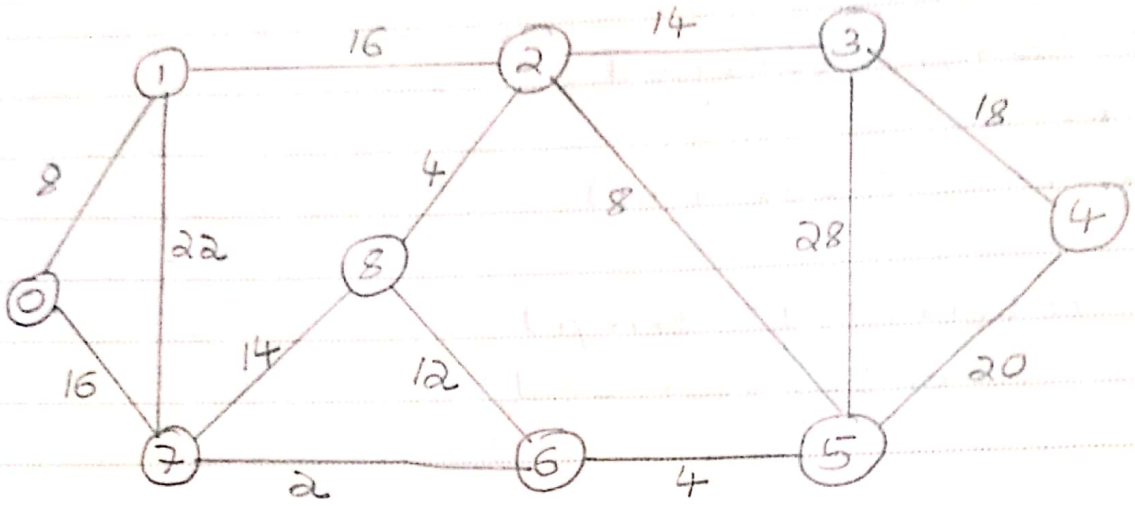
- Used to find the minimum spanning tree

Method :

- Sort all the edges
- Keep adding edges one by one until
  - There is no cycle
  - all the nodes are connected

Every spanning tree of n vertices will have $n-1$ edges.

Example:



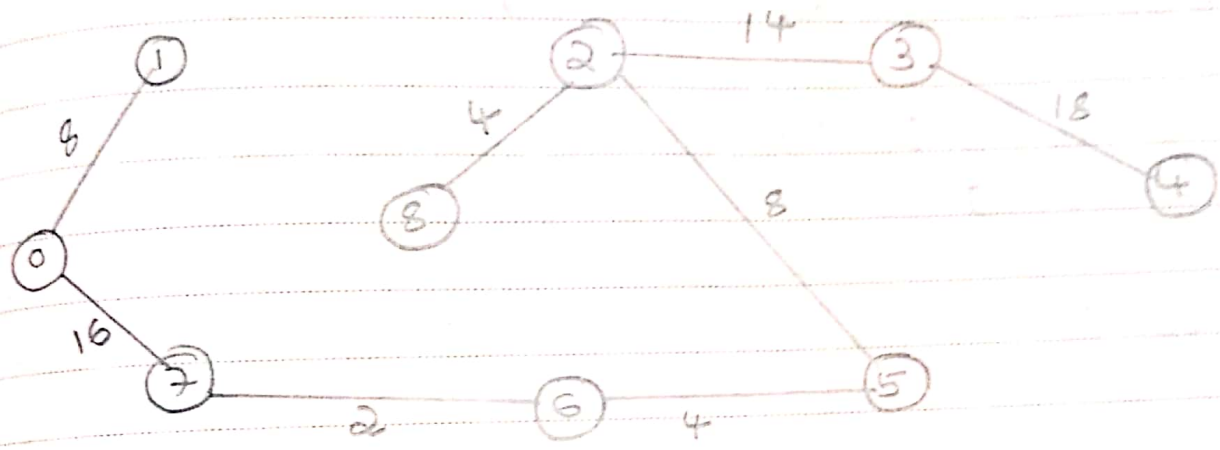
Sorted Edges

(0,1) → 8	(6,7) → 2	(2,8) → 4	(5,6) → 4
(7,8) → 14	(2,5) → 8	(6,8) → 12	(2,3) → 14
(4,5) → 20	(0,7) → 16	(1,2) → 16	(3,4) → 18
	(1,7) → 22	(3,5) → 28	

Steps	(u,v)	i = find(u) j = find(v)	o/p (u,v)	Union(i,j) array 0 1 2 3 4 5 6 7 8
init	~~8,7~~	~~6,7~~		0 1 2 3 4 5 6 7 8
1	(6,7)	6, 7	(6,7)	0 1 2 3 4 5 7 7 8
2	(2,8)	2, 8	(2,8)	0 1 8 3 4 5 7 7 8
3	(5,6)	5, 7	(5,6)	0 1 8 3 4 7 7 7 8
4	(0,1)	0, 1	(0,1)	1 1 8 3 4 7 7 7 8
5	(2,5)	8 7	(2,5)	1 1 7 3 4 7 7 7 7
6	(6,8)	7 7	discard	
7	(2,3)	7 3	(2,3)	1 1 3 3 4 3 3 3 3
8	(7,~~8~~)	3 3	discard	
9	(0,7)	1 3	(0,7)	3 3 3 3 4 3 3 3 3
10	(1,2)	3 3	discard	
11	(3,4)	3 4		4 4 4 4 4 4 4 4 4

PAGE NO
DATE : / /

Spanning Tree:



Minimum Cost = 8+16+2+4+8+4+14+18
= 74.

Union-find : Root Method

	0	1	2	3	4	5
Arr	~~0~~ 2	~~0~~ 1	2	3	4	5
	0	1	2	3	4	5

copy

Union (0,2)

Union (1,0)

↳ root of 0

root of 1

Union (1,4)

	0	1	2	3	4	5
Arr	2	0	4	~~3~~ 4	4	5
	0	1	2	3	4	5

Union (3,4)

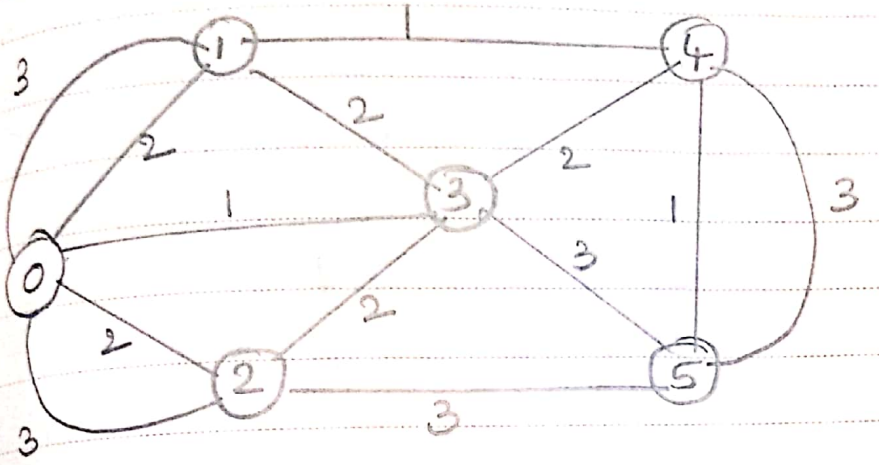
PAGE NO. :
DATE : / /

Code Snippets :

```
int root (int arr[], int i)
{
    while (arr[i] != i)
        i = arr[i];

    return i;
}

int find (int u, int v, int arr[])
{
    if((root (arr, u)) == (root (arr, v)))
        return 1;
    else
        return 0;
}

int union ( int arr[], int u, int v)
{
    int rootu = root (arr, u)
    int rootv = root (arr, v)
    arr[rootu] = rootv;
}
```

PAGE NO. :
DATE : / /

Kauskals Example: Compute MAX Spanning Tree.



Sorted Edges: (descending)

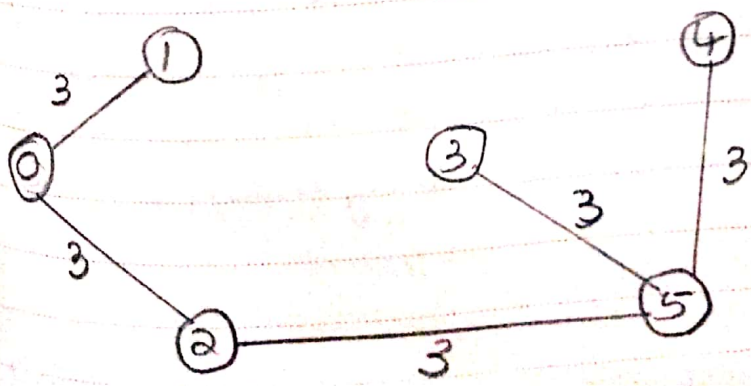
(2,5) → 3

(0,1) → 3 (0,2) → 3 (3,5) → 3 (4,5) → 3

(0,1) → 2 (0,2) → 2 (1,3) → 2 (2,3) → 2

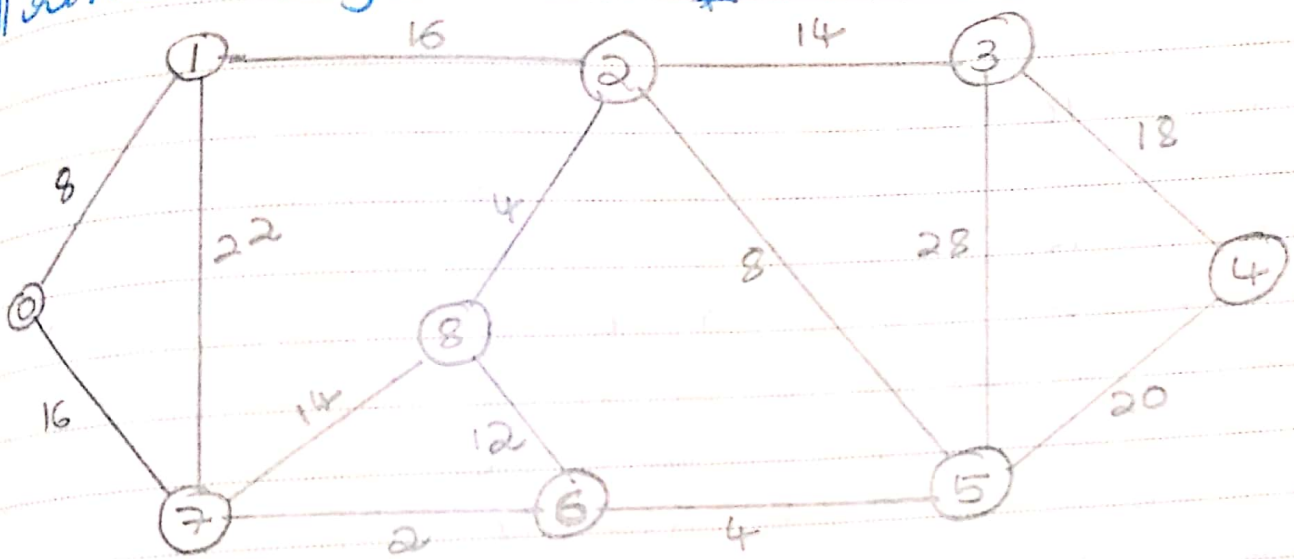
(3,4) → 2 (0,3) → 1 (1,4) → 1 (4,5) → 1

Steps	(u, v)	i = find(u) j = find(v)	Output	Union(i,j) 0 1 2 3 4 5
init				0 1 2 3 4 5
1	(0,1)	0 1	(0,1)	1 1 2 3 4 5
2	(0,2)	1 2	(0,2)	2 2 2 3 4 5
3	(2,5)	2 5	(2,5)	5 5 5 3 4 5
4	(3,5)	3 5	(3,5)	5 5 5 5 4 5
5	(4,5)	4 5	(4,5)	5 5 5 5 5 5



Tree Cost = 15.

Prim's Algorithm *



Initialization: Step 0

	dist	path
0	0	0
1	8	0
2	∞	0
3	∞	0
4	∞	0
5	∞	0
6	∞	0
7	16	0
8	∞	0

u = 1

dist[u] = 8

o/p: (0, 1)

p[u], u

Cost matrix is written in next page.

PAGE NO. :
DATE : / /

Cost [ ] [ ] =

	0	1	2	3	4	5	6	7	8
0	0	8	∞	∞	∞	∞	∞	16	∞
1	8	0	16	∞	∞	∞	∞	22	∞
2	∞	16	0	14	∞	8	∞	∞	4
3	∞	∞	14	0	18	28	∞	∞	∞
4	∞	∞	∞	18	0	20	∞	∞	∞
5	∞	∞	8	28	20	0	4	∞	∞
6	∞	∞	∞	∞	∞	4	0	2	12
7	16	22	∞	∞	∞	∞	2	0	14
8	∞	∞	4	∞	∞	∞	12	14	0

Step 1 :

$S = \{0, 1\}$

$V - S = \{2, 3, 4, 5, 6, 7, 8\}$

	dist	path
0	0	0
1	8	0
2	16	1
3	∞	0
4	∞	0
5	∞	0
6	∞	0
7	16	0
8	∞	0

$\min(\infty, 16) = 16$

$\min(\infty, \infty) = \infty$

$\min(\infty, \infty) = \infty$

$\min(\infty, \infty) = \infty$

$\min(\infty, \infty) = \infty$

$\min(16, 22) = 16$

$\min(\infty, \infty) = \infty$

$u = 2$

$d[u] = 16$
ist

O/p:
(1, 2)

PAGE NO. :
DATE : / /

Step 2:

$S = \{0, 1, 2\}$ $\quad V - S = \{3, 4, 5, 6, 7, 8\}$

	dist	path	
0	0	0	
1	8	0	
2	16	1	
3	14	2	$\min(\infty, 14) = 14$
4	∞	0	$\min(\infty, \infty) = \infty$
5	8	2	$\min(\infty, 8) = 8$
6	∞	0	$\min(\infty, \infty) = \infty$
7	16	0	$\min(16, \infty) = 16$
8	4	2	$\min(\infty, 4) = 4$

$u = 8$
$dist[u] = 4$

o/p : (2, 8)

Step 3:

$S = \{0, 1, 2, 8\}$ $\quad V - S = \{3, 4, 5, 6, 7\}$

	dist	path	
0	0	0	
1	8	0	
2	16	1	
3	14	2	$\min(14, \infty) = 14$
4	∞	0	$\min(\infty, \infty) = \infty$
5	8	2	$\min(8, \infty) = 8$
6	12	8	$\min(\infty, 12) = 12$
7	14	8	$\min(16, 14) = 14$
8	4	2	

$u = 5$
$dist[u] = 8$
o/p : (2, 5)

DATE : / /

Step 4:

$S = \{0, 1, 2, 5, 8\}$ $\quad V - S = \{3, 4, 6, 7\}$

	dist	path
0	0	0
1	8	0
2	16	1
3	14	2
4	20	5
5	8	2
6	4	5
7	14	8
8	4	2

$\min(14, 28) = 14$

$\min(20, 20) = 20$

$\min(12, 4) = 4$

$\min(14, 6) = 14$

$u = 6$

$dist[u] = 4$

O/p : (5, 6)

Step 5:

$S = \{0, 1, 2, 5, 6, 8\}$ $\quad V - S = \{3, 4, 7\}$

	dist	path
0	0	0
1	8	0
2	16	1
3	14	2
4	20	5
5	8	2
6	4	5
7	2	6
8	4	2

$\min(14, 6) = 14$

$\min(20, 4) = 20$

$\min(14, 2) = 2$

$u = 7$

$dist[u] = 2$

O/p : (6, 7)

PAGE NO. :
DATE : / /

Step 6:

$S = \{0, 1, 2, 5, 6, 7, 8\}$ $\quad V - S = \{3, 4\}$

	dist	path
0	0	0
1	8	0
2	16	1
3	14	2
4	20	5
5	8	2
6	4	5
7	2	6
8	4	2

$\min(14, a) = 14$

$\min(20, a) = 20$

$u = 3$

$dist[3] = 14$

o/p : (2, 3)

Step 7:

$S = \{0, 1, 2, 3, 5, 6, 7, 8\}$ $\quad V - S = \{4\}$

	dist	path
0	0	0
1	8	0
2	16	1
3	14	2
4	18	3
5	8	2
6	4	5
7	2	6
8	4	2

$\min(20, 18) = 18$

$u = 4$

$dist[4] = 18$

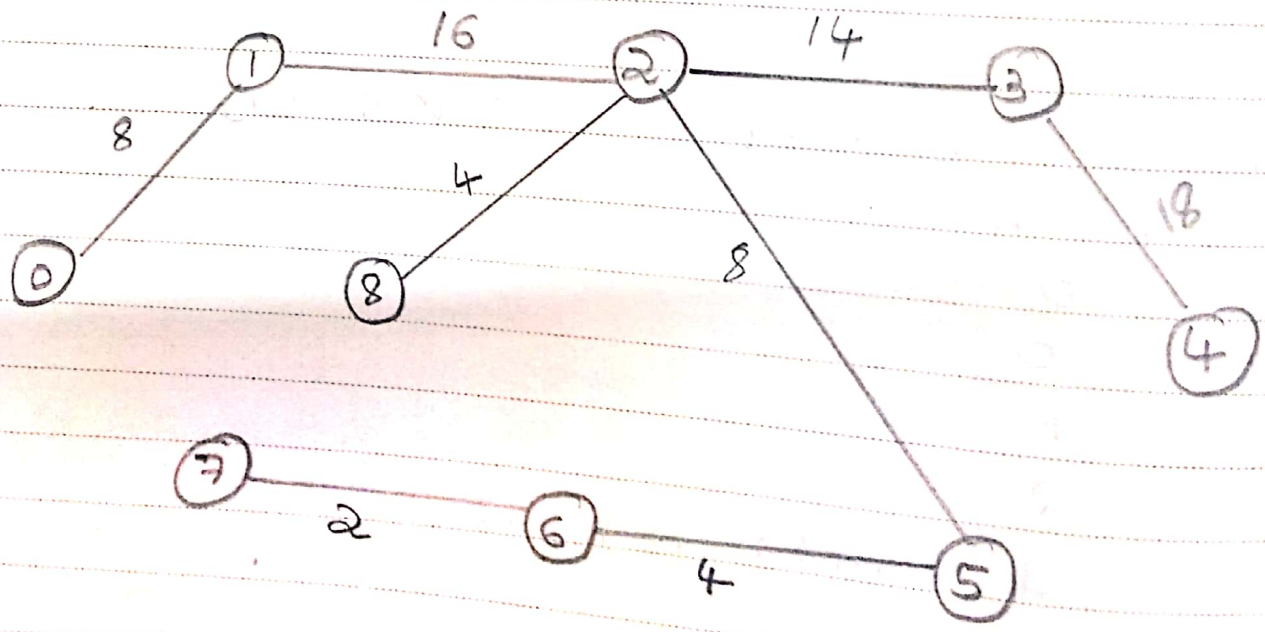
o/p : (3, 4)

Step 8:

$V - S = \{\}$

$S = \{0, 1, 2, 3, 4, 5, 6, 7, 8\}$

	dist	path
0	0	0
1	8	0
2	16	1
3	14	2
4	18	3
5	8	2
6	4	5
7	2	6
8	4	2



MST = 8 + 16 + 14 + 18 + 8 + 4 + 2 + 4

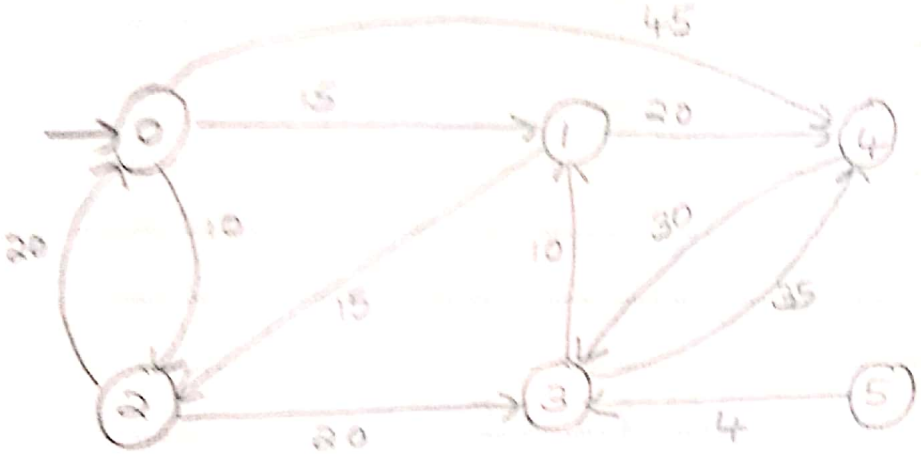
= 74

DATE : / /

Kruskal's	Prim's
1) Grows with minimum cost edge.	1) Grows with minimum cost vertex.
2) If we stop algorithm in the middle, kruskal can give a disconnected tree or forest.	2) If we stop algorithm in the middle, prim always generates a connected tree.
3) Need to give attention on cycle check.	3) Need not give attention on cycle check.
4) Can function on the disconnected graphs too.	4) Graphs must be connected.
5) Edge selection is not based on previous step.	5) Spans from one vertex to another
6) Allows new to new & old and old to get connected.	6) Joins new vertex to old vertex.
7) With an efficient Union-find algorithm, running time is dominated by time needed for sorting edges. $O(\|E\| \log \|E\|)$	7) Weight matrix & priority queue as unordered array is $O(\|V\|^2)$. Adjacency list & priority queue as min heap is $O(\|E\| \log \|V\|)$

Dijkstra's Algorithm

Trace for Dijkstra's algorithm using 5 as the source vertex.



Cost matrix[6][6] =

	0	1	2	3	4	5
0	0	15	10	∞	45	∞
1	∞	0	15	∞	20	∞
2	20	∞	0	20	∞	∞
3	∞	10	∞	0	35	∞
4	∞	∞	∞	30	0	∞
5	∞	∞	∞	4	∞	0

Source = 5

PAGE NO. :
DATE : / /

PH

Initialization:

	dist
0	∞
1	∞
2	∞
3	4
4	∞
5	0

	path
0	5
1	5
2	5
3	5
4	5
5	5

$V = \{0, 1, 2, 3, 4, 5\}$
$S = \{5\}$
$V - S = \{0, 1, 2, 3, 4\}$

Iteration 01:

$u = 3$
$dist[u] = 4$

$S = \{5, 3\}$
$V - S = \{0, 1, 2, 4\}$

	dist
0	∞
1	14
2	∞
3	4
4	39
5	0

	path
0	5
1	3
2	5
3	5
4	3
5	5

$\min(\infty, 4+\infty) = \infty$
$\min(\infty, 4+10) = 14$
$\min(\infty, 4+\infty) = \infty$
$\min(\infty, 4+35) = 39$

Find the min for all V-S set.

Eg:

$\min(\infty, 4+10)$

∞ → dist[1]

$4+10$ → dist[3] + cost[3][1]

Iteration 02:

$u = 1$

$dist[u] = 14$

$S = \{1, 3, 5\}$

$V - S = \{0, 2, 4\}$

	dist
0	∞
1	14
2	29
3	4
4	34
5	0

	path
0	5
1	3
2	1
3	5
4	1
5	5

$\min(\infty, 14 + \infty) = \infty$

$\min(\infty, 14 + 15) = 29$

$\min(\infty 39, 14 + 20) = 34$

Iteration 03:

$u = 2$

$dist[u] = 29$

$S = \{1, 2, 3, 5\}$

$V - S = \{0, 4\}$

	dist
0	49
1	14
2	29
3	4
4	34
5	0

	path
0	2
1	3
2	1
3	5
4	1
5	5

$\min(\infty, \infty + 29 + 20) = 49$

$\min(34, 29 + \infty)$

DATE: / /

PH

Iteration 04:

$S = \{1, 2, 3, 4, 5\}$

$V - S = \{0\}$

$u = 4$

$dist[u] = 34$

	dist
0	49
1	14
2	29
3	4
4	34
5	0

	path
0	2
1	3
2	1
3	5
4	1
5	5

$\min(^{49}, 34 + \infty) = 49$

Iteration 05:

$S = \{0, 1, 2, 3, 4, 5\}$

$V - S = \{\}$

$u = 0$

$dist[u] = 49$

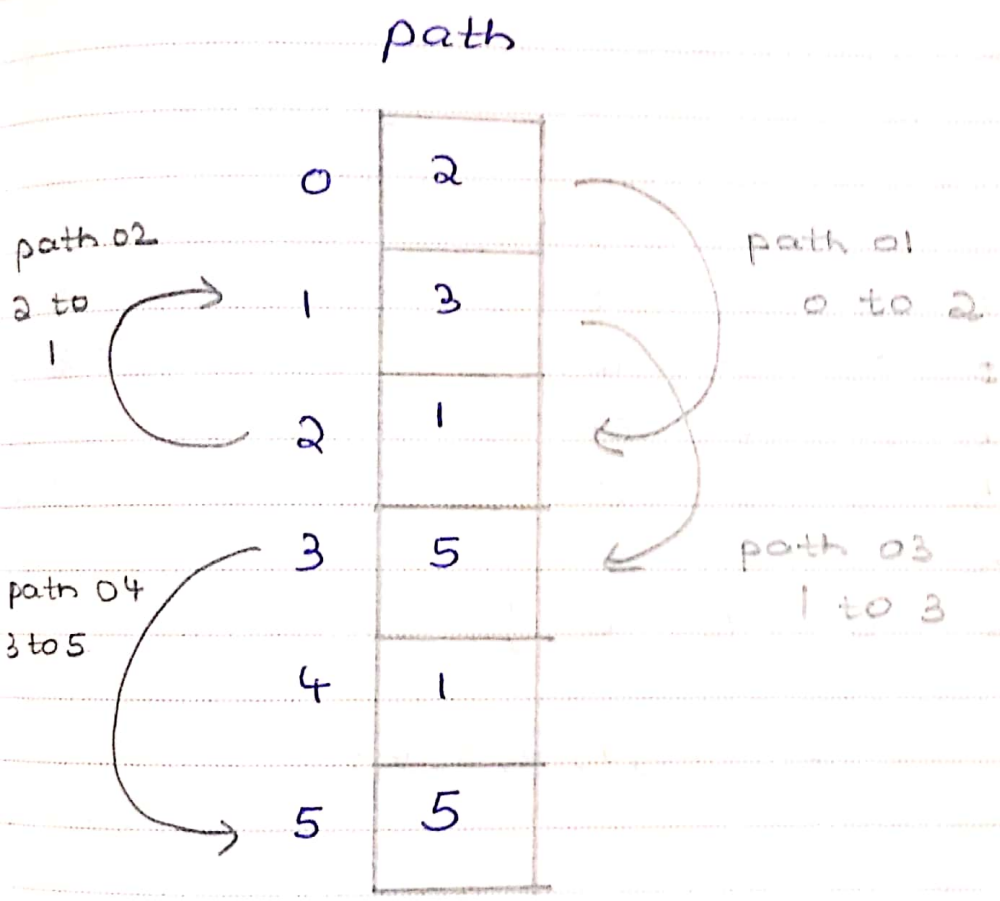
	dist
0	49
1	14
2	29
3	4
4	34
5	0

	path
0	2
1	3
2	1
3	5
4	1
5	5

PAGE NO :
DATE : / /

To Traverse Path:

To traverse path from 5 to 0, go in reverse order following the vertex path.

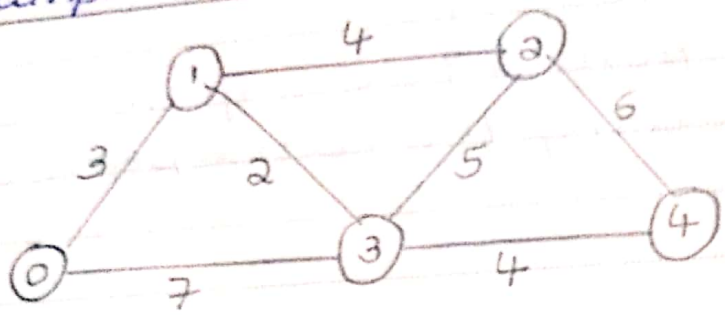


$0 \rightarrow 2 \rightarrow 1 \rightarrow 3 \rightarrow 5$

↓ Now reverse this & we have path from 5 to 0.

The algorithm is single source shortest path.

Example 02:



cost[s][5] =

	0	1	2	3	4
0	0	3	∞	7	∞
1	3	0	4	2	∞
2	∞	4	0	5	6
3	7	2	5	0	4
4	∞	∞	6	4	0

$S = \{0\}$ $V = \{0, 1, 2, 3, 4\}$

Initialization:

$S = \{0\}$

$V - S = \{1, 2, 3, 4\}$

	dist	path
0	0	0
1	3	0
2	∞	0
3	7	0
4	∞	0

PAGE NO. :
DATE : / /

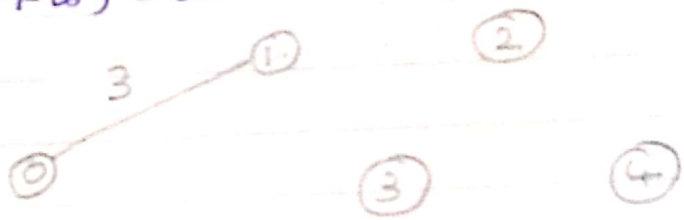
Ituation 01:

$u = 1$
$dist[u] = 3$

$S = \{0, 1\}$
$V - S = \{2, 3, 4\}$

	dist	path
0	0	0
1	3	0
2	7	1
3	5	1
4	∞	0

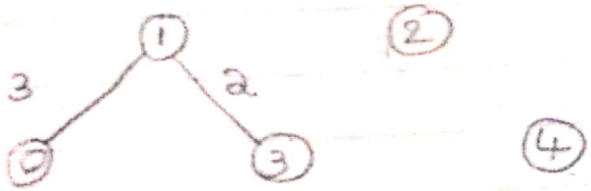
$\min(\infty, 3 + 4) = 7$
$\min(7, 3 + 2) = 5$
$\min(\infty, 3 + \infty) = \infty$



Ituation 02:

$u = 3$
$dist[u] = 5$

$S = \{0, 1, 3\}$
$V - S = \{2, 4\}$



	dist	path
0	0	0
1	3	0
2	7	1
3	5	1
4	9	3

$\min(7, 5 + 5) = 7$

$\min(\infty, 5 + 4) = 9$

DATE : / /

Itration 03:

$S = \{0, 1, 2, 3\}$

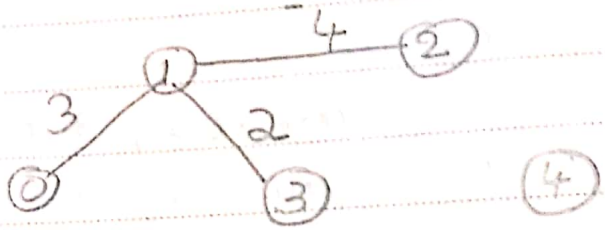
$V - S = \{4\}$

$u = 2$

$dist[u] = 7$

dist path

0	0	0
1	3	0
2	7	1
3	5	1
4	9	3



$\min(9, 7 + 6) = 13$

Itration 04:

$S = \{0, 1, 2, 3, 4\}$

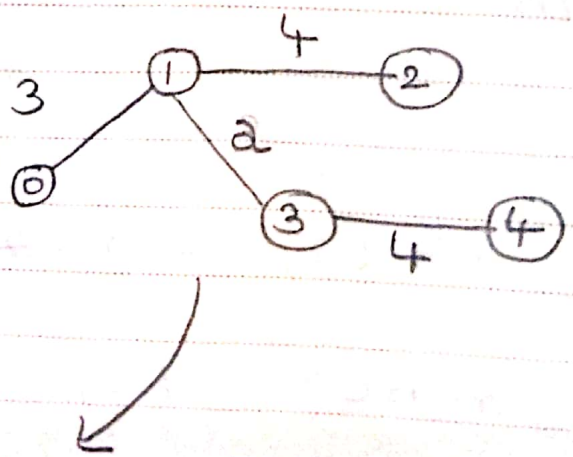
$V - S = \{\}$

$u = 4$

$dist[u] = 9$

dist path

0	0	0
1	3	0
2	7	1
3	5	1
4	9	3

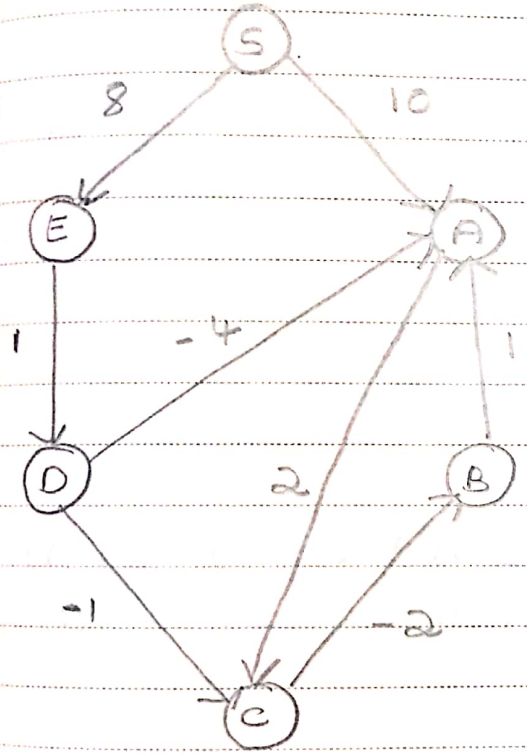


Final Single Source Shortest Path from 0.

PAGE NO. :
DATE : / /

* Bellman Ford Algorithm *

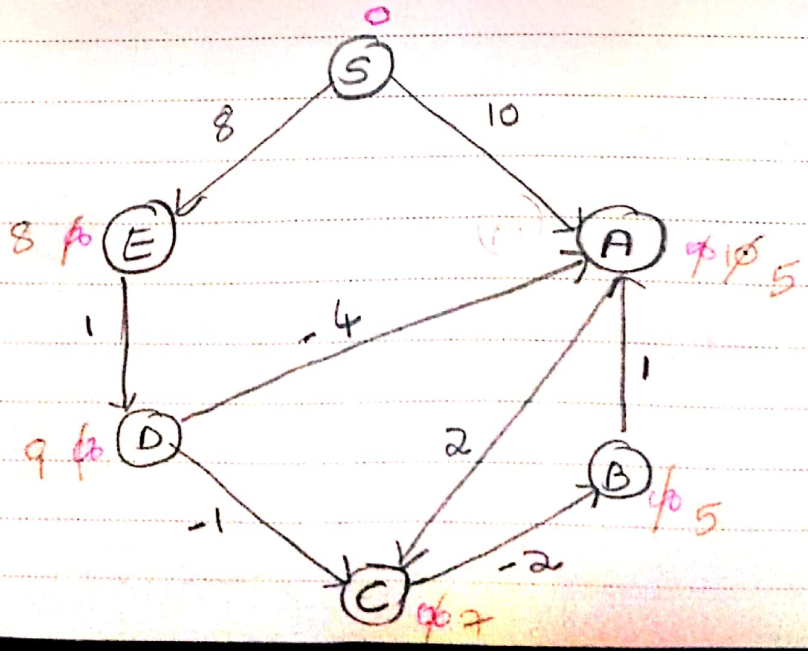
Example 01 Case 01



Write edges :

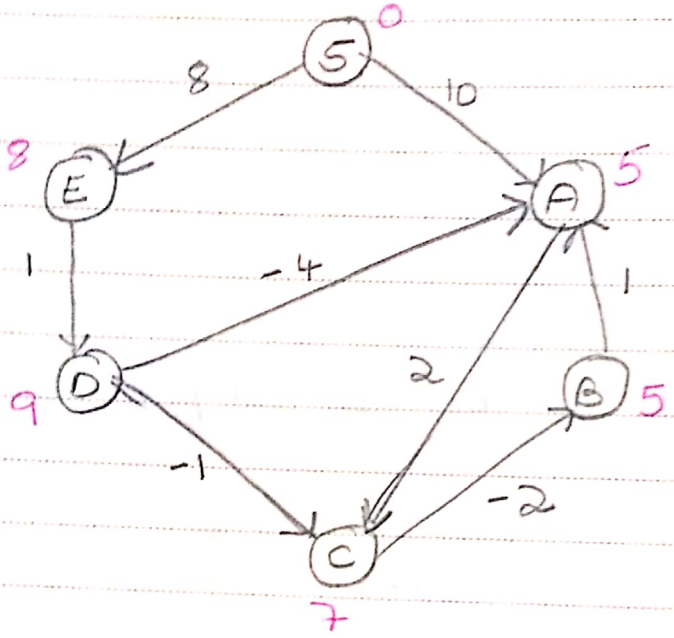
(S, E)
(S, A)
(E, D)
(D, A)
(A, C)
(B, A)
(D, C)
(C, B)

Iteration 01 :



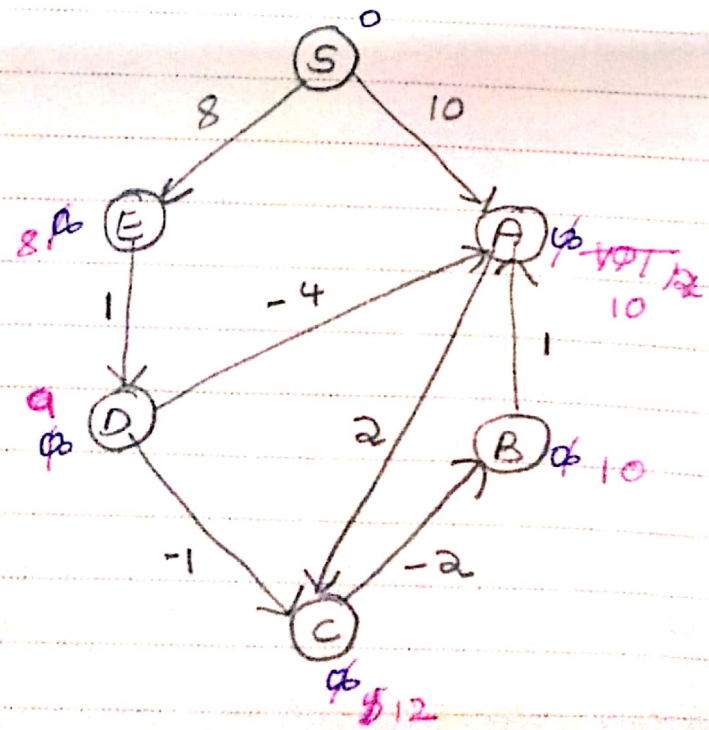
Iteration 02:

no changes, we stop.



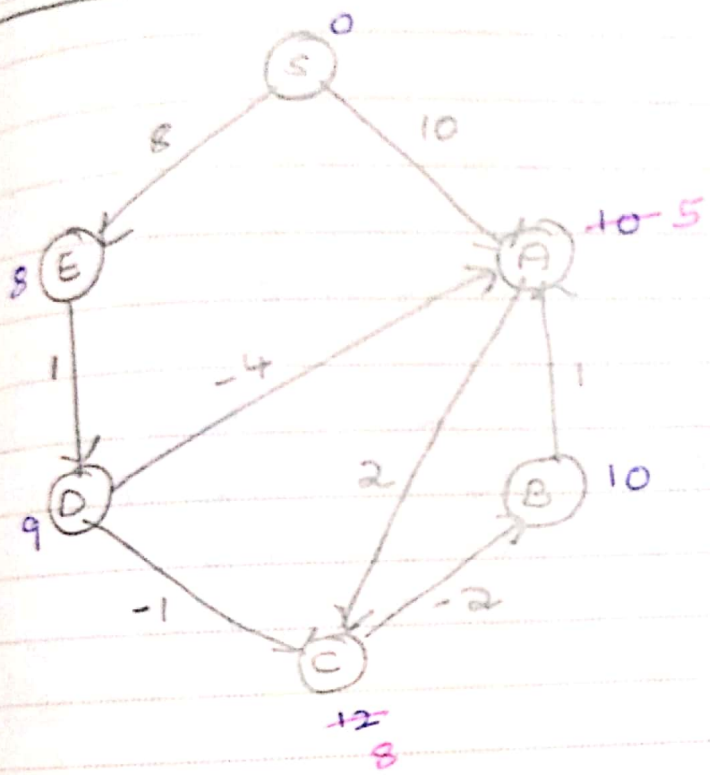
Case 02: Edge list: (S, E) (S, A) (A, C) (B, A) (C, B) (D, A) (D, C) (E, D)

Iteration 01:

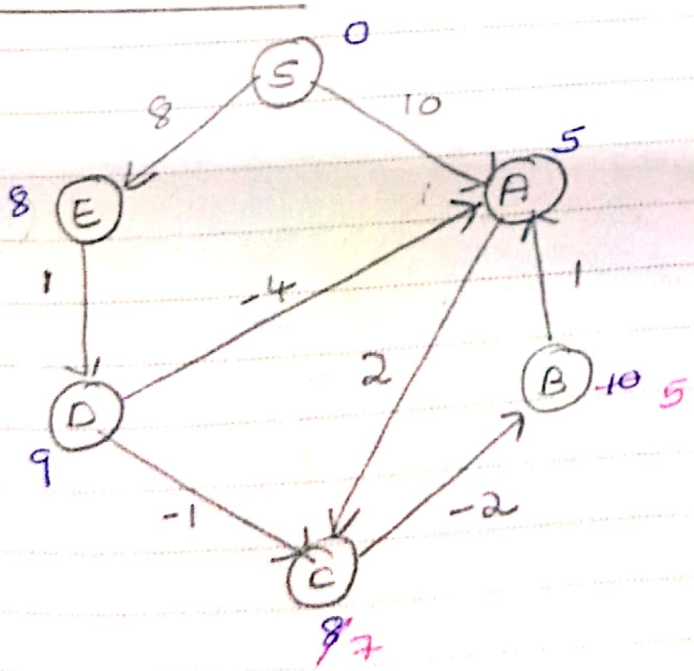


PAGE NO :
DATE : / /

Iteration 02:



Iteration 03:



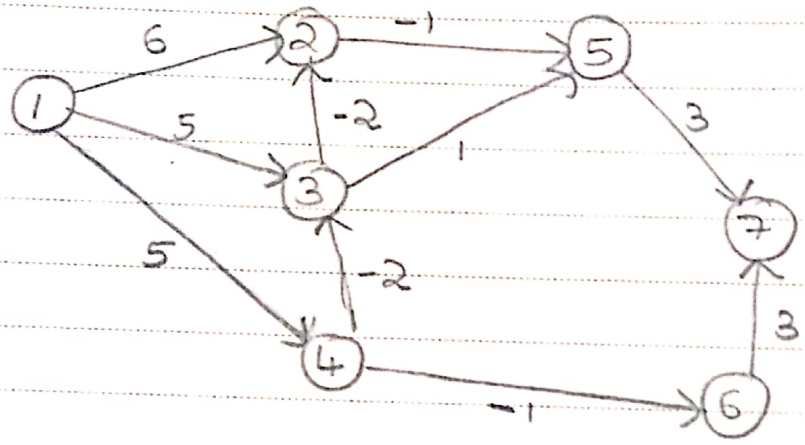
Iteration 04:

Nothing changes. We stop.

PAGE NO. :
DATE : / /

Note: With n vertices, we will have atmost $n-1$ iterations.

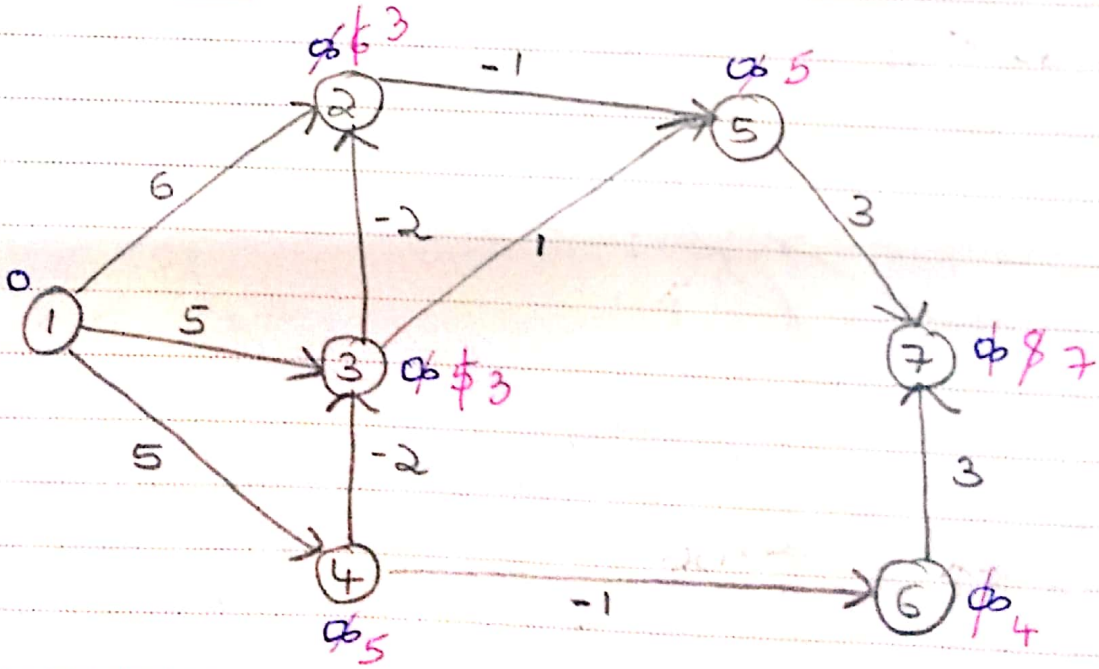
Example 02:



Edge list:

(1,2) (1,3) (1,4)
(2,5) (3,2), (3,5)
(4,3) (4,6)
(5,7) (6,7)

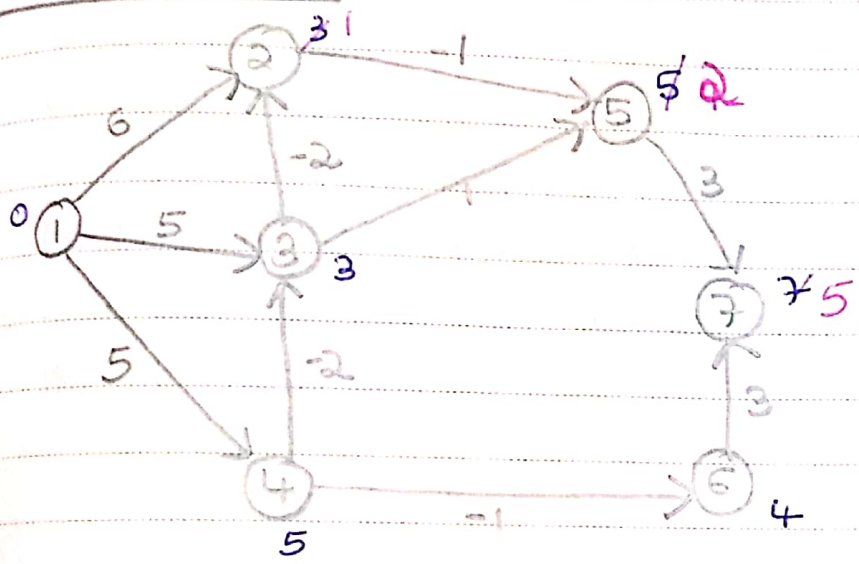
Iteration 01:



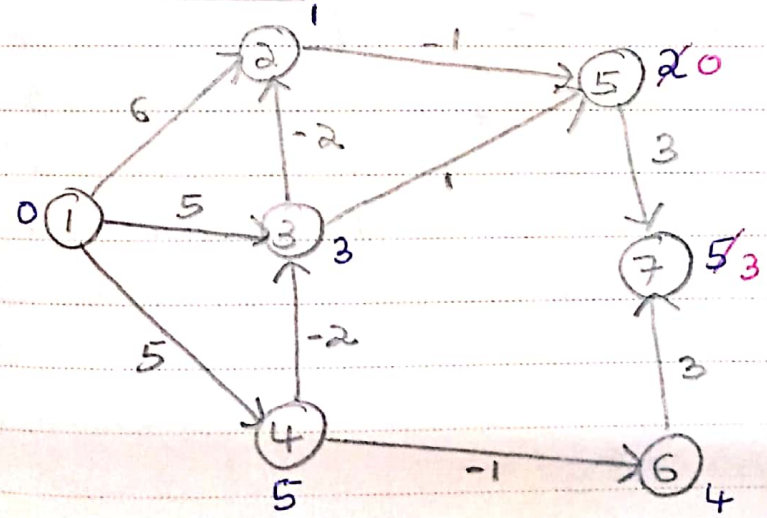
PAGE NO. :

DATE : / /

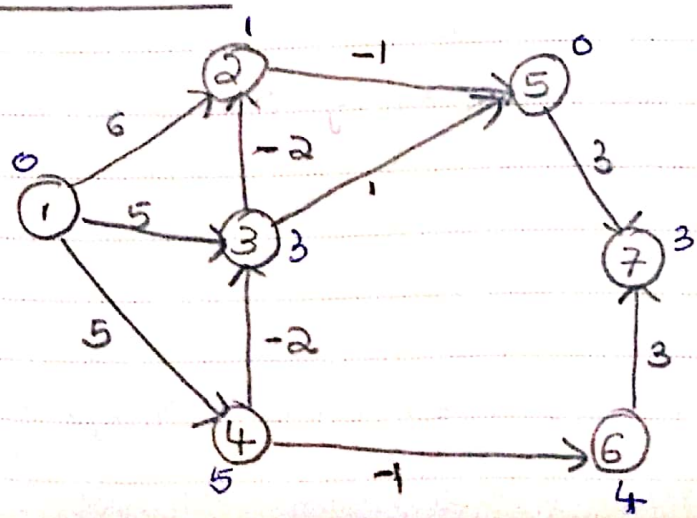
Iteration 02:



Iteration 03:

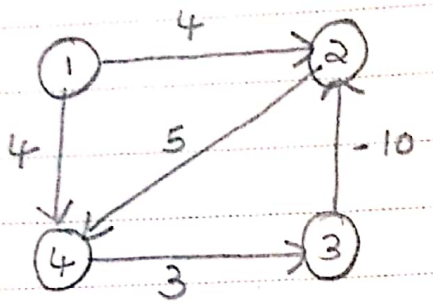


Iteration 04:



NO changes, we stop.

Example 03:

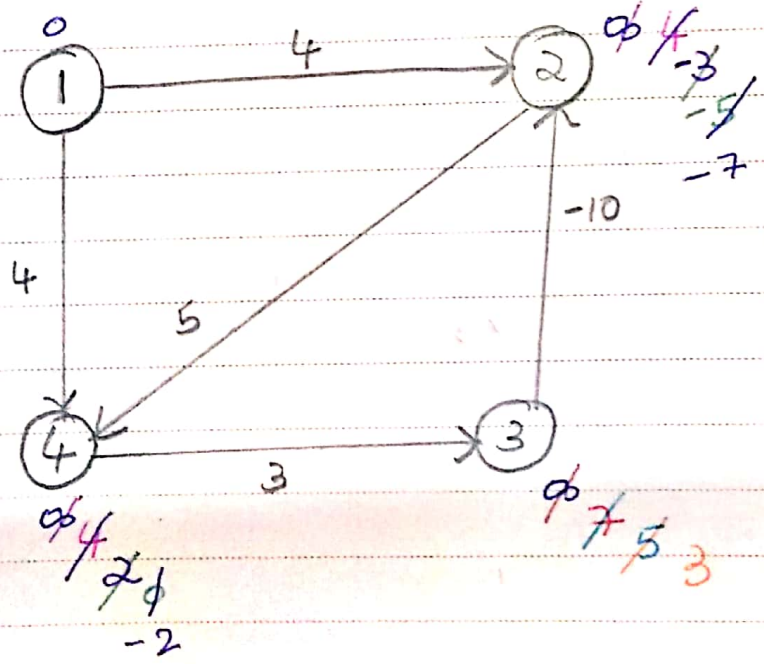


Edge list:

(3,2) (4,3) (1,4) (1,2)
(2,4)

Iterations

Iteration 01
Iteration 02
Iteration 03
Iteration 04
Iteration 05
Iteration 06
Iteration 07
&
So on.



Bellman-ford will not work if there is a negative weight cycle.

In the graph:

4 → 3 → 2 → 5 → 4 is a negative weight Cycle.

PAGE NO. :
DATE : / /

Dijkstras v/s Bellman-Ford:

1. Bellman can handle negative weights & can detect negative weight cycles

2. Dijkstras does edge relaxation once & in Bellman ford it is done atmost n-1 times where n is number of vertices

3. Bellman visits vertices more than once

4. Dijkstra's has better implementation efficiency

5. Both algorithms work towards single source shortest path.

6. Bellman algorithm uses only information only from its neighbors and knowledge of its link costs. Dijkstras requires that each node must have complete topological information about the network.

7. Bellman works well for distributed systems

8. Dijkstras is greedy technique and Bellman is dynamic programming technique.