Module-2

Greedy Methods

BCSC0012

Designing Techniques of Algorithms

Design and Analysis of Algorithms

Dr. Gaurav Kumar
Asst. Prof, CEA, GLAU, Mathura

Designing Techniques of Algorithms

01

Divide and Conquer Strategy

02

Greedy Technique

03

Dynamic Programming

04

Branch and Bound

~Dr Gaurav Kumar, Asst. Prof, CEA, GLAU

02

Greedy Technique

Example of Counting Coins

We have coins of ₹ 1, 2, 5 and 10 and we are asked to count ₹ 18. This problem is to count to a desired value by choosing the least possible coins

The greedy procedure will be −

1 − Select one ₹ 10 coin, the remaining count is 8

2 − Then select one ₹ 5 coin, the remaining count is 3

3 − Then select one ₹ 2 coin, the remaining count is 1

4 − And finally, the selection of one ₹ 1 coins solves the problem

Though, it seems to be working fine, for this count we need to pick only 4 coins

~Dr Gaurav Kumar, Asst. Prof, CEA, GLAU

Example of Counting Coins

This problem is to count to a desired value by choosing the least possible coins

We have coins of ₹ 1, 7, 10 and we are asked to count ₹ 18.

Counting coins for value 18 will be absolutely optimum

but for count like 15?

For example, the greedy approach will use 10 + 1 + 1 + 1 + 1 + 1, total 6 coins.

Whereas the same problem could be solved by using only 3 coins (7 + 7 + 1)

Note- Greedy approach picks an immediate optimized solution and may fail where global optimization is a major concern.

Greedy Technique

01

Always makes the choice that looks best at the moment

02

Solve the **optimization problem**

03

Doesn't always guarantee the optimal solution

04

It generally produces a solution that is very close in value to the optimal

~Dr Gaurav Kumar, Asst. Prof, CEA, GLAU

Optimization Problem

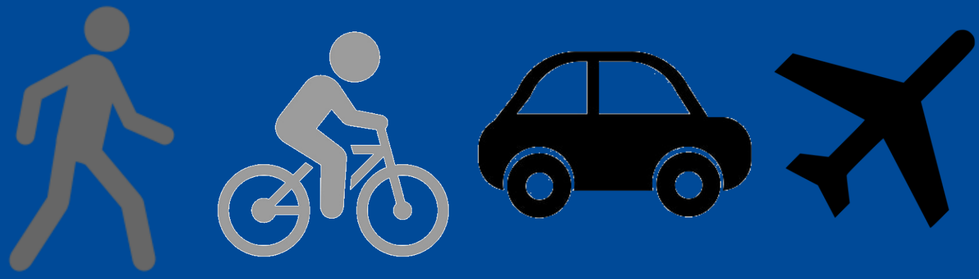
A problem that requires a maximum or minimum result is the optimization problem.

~Dr Gaurav Kumar, Asst. Prof, CEA, GLAU

More Examples for the Better Understanding

Problem- A person wants to travel from Delhi to Mumbai

Delhi



GATEWAY OF INDIA

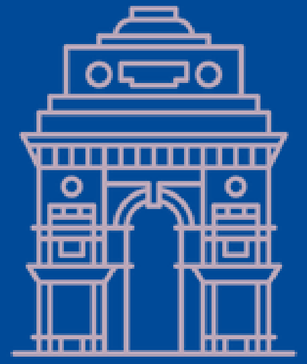
Mumbai

4 Solutions

~Dr Gaurav Kumar, Asst. Prof, CEA, GLAU

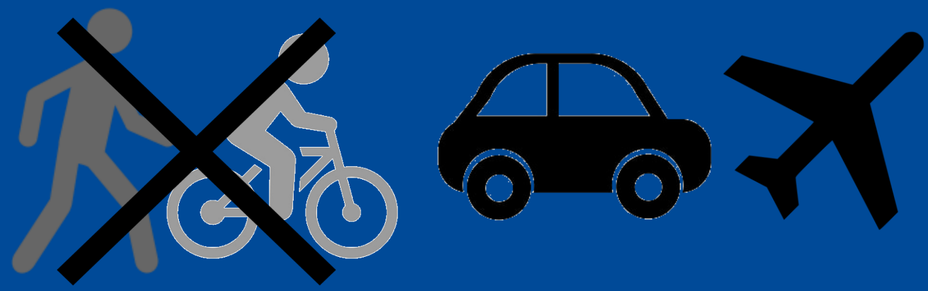
More Examples for the Better Understanding

Problem- A person wants to travel from Delhi to Mumbai



Within 16 hours

Delhi



GATEWAY OF INDIA

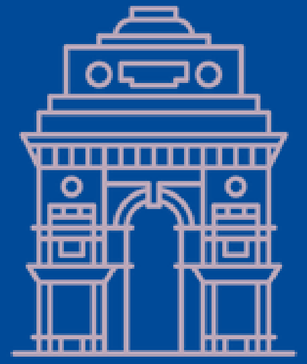
Mumbai

Two Feasible Solutions

~Dr Gaurav Kumar, Asst. Prof, CEA, GLAU

More Examples for the Better Understanding

Problem- A person wants to travel from Delhi to Mumbai

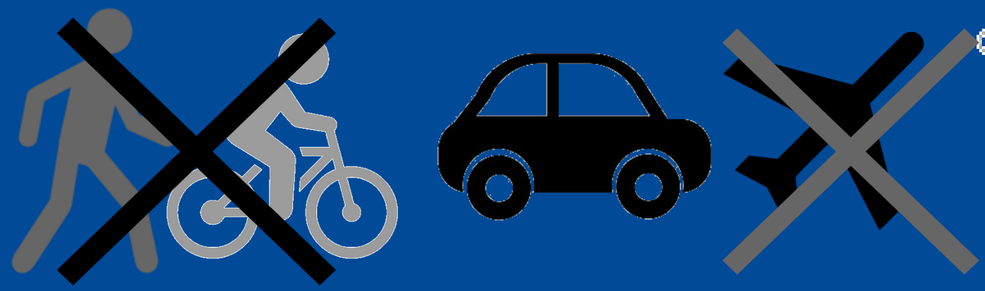


Delhi

Within 16 hours, with minimum cost

GATEWAY OF INDIA

Mumbai



One Optimal Solution

~Dr Gaurav Kumar, Asst. Prof, CEA, GLAU

Terms Used in Greedy Methods

Solution Space – All possible solution of a given problems

Feasible Solutions – Possible solutions of a given problem with certain constraints (some conditions)

Optimal Solution – Minimum or Maximum Cost/Profit Solution of a given problem

Note: Each problem can have only one OPTIMAL SOLUTION

~Dr Gaurav Kumar, Asst. Prof, CEA, GLAU

Algorithms of Greedy Methods

```
Algorithm Greedy Method (a, n)
{
   for i -> n
    {
       x= select (a)
       if x is feasible
       then solution = solution + x
    }
}
```

n=6

a

a1	a2	a3	a4	a5	a6

Note: We solve the problems in stage wise manner.

~Dr Gaurav Kumar, Asst. Prof, CEA, GLAU

Application of Greedy Technique

01 Fractional Knapsack Problem

02 Prim's Algorithm

03 Dijkstra's Algorithm

04 Kruskal's Algorithm

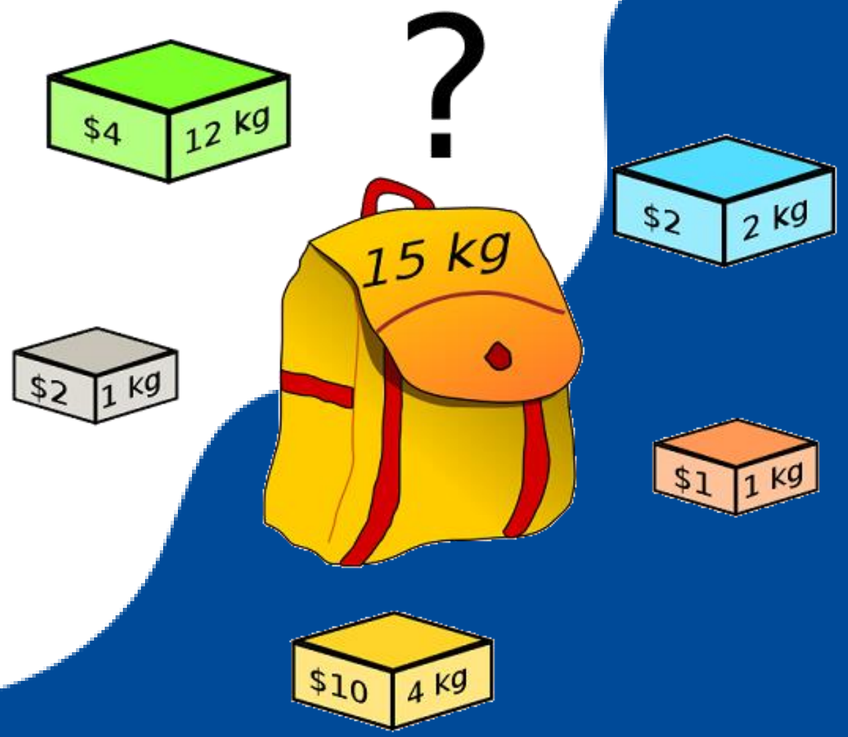
05 Activity Search Problem

06 Huffman Coding

07 Travelling Salesperson Problem

08 Ford- Fulkerson Algorithm

09 Boruvka's Algorithm



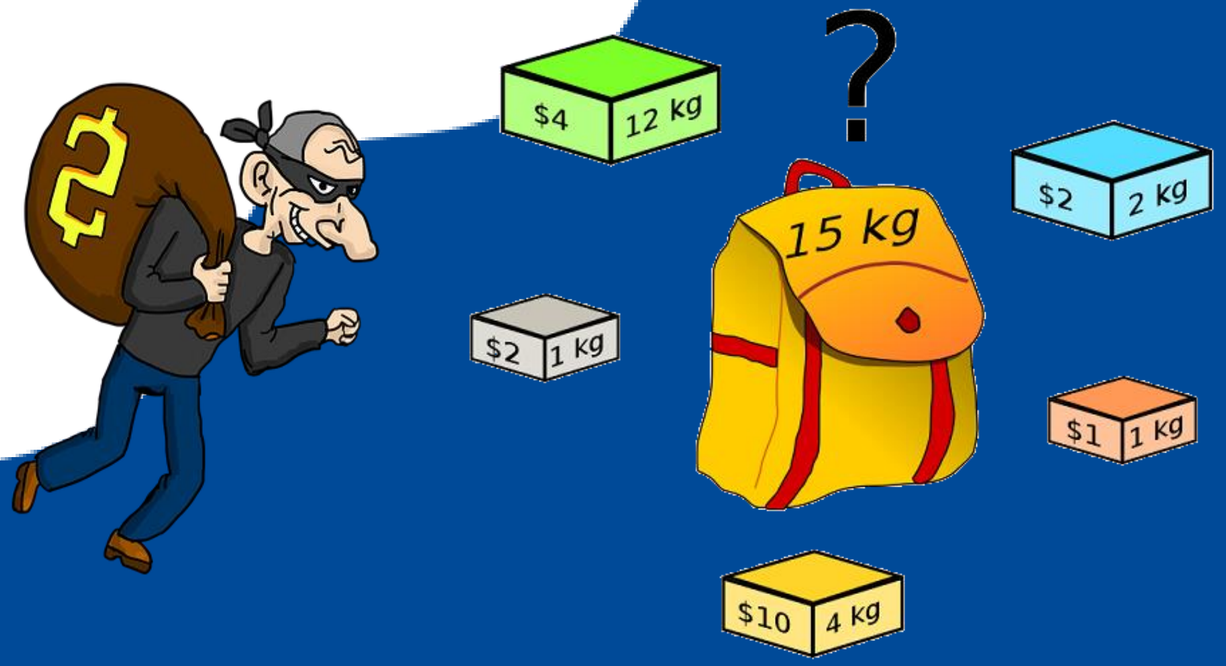
Greedy Methods Application

1. Knapsack Problem

~Dr Gaurav Kumar, Asst. Prof, CEA, GLAU

1. Knapsack Problem

Problem- Given weights and values of n items, we need to put these items in a knapsack of capacity W to get the maximum total value in the knapsack.



a. Fractional Knapsack Problem

we can break items for maximizing the total value of knapsack. This problem in which we can break an item is also called the fractional knapsack problem. It is solved using Greedy Method.

b. 0/1 Knapsack Problem

We are not allowed to break items. We either take the whole item or don't take it. Use Dynamic Programming method to solve it.

~Dr Gaurav Kumar, Asst. Prof, CEA, GLAU

Example of Fractional Knapsack Problem

Problem Statement

A thief enters a house for robbing it. He can carry a maximal weight of 60 kg into his bag. There are 5 items in the house with the following weights and cost values. What items should thief take if he can even take the fraction of any item with him?

Item	Weight	Cost Value
1	5	30
2	10	40
3	15	45
4	22	77
5	25	90

~Dr Gaurav Kumar, Asst. Prof, CEA, GLAU

Example of Fractional Knapsack Problem

Greedy Algorithm

Step-01: Compute the value / weight ratio for each item

Step-02: Sort all the items in decreasing order of their value / weight ratio

Step-03: Start filling the knapsack by putting the items into it one by one till bag is full.

Note: In fractional knapsack problem,
even the fraction of any item can be taken

~Dr Gaurav Kumar, Asst. Prof, CEA, GLAU

Example of Fractional Knapsack Problem

Step-01: Compute the value / weight ratio for each item

Items	Weight	Value	Ratio
1	5	30	6
2	10	40	4
3	15	45	3
4	22	77	3.5
5	25	90	3.6

Step-02: Sort all the items in decreasing order of their value / weight ratio-

I1	I2	I5	I4	I3
(6)	(4)	(3.6)	(3.5)	(3)

Example of Fractional Knapsack Problem

Step-03: Start filling the knapsack by putting the items into it one by one.

Knapsack Weight	Items in Knapsack	Cost
60	Ø	0
55	I1	30
45	I1, I2	70
20	I1, I2, I5	160

Now,

Knapsack weight left to be filled is 20 kg but item-4 has a weight of 22 kg.

~Dr Gaurav Kumar, Asst. Prof, CEA, GLAU

Example of Fractional Knapsack Problem

Since in fractional knapsack problem, even the fraction of any item can be taken.

So, knapsack will contain the following items-

< I1 , I2 , I5 , (20/22) I4 >

Total cost of the knapsack

= 160 + (20/22) x 77

= 160 + 70

= 230 units

Practice Questions on Fractional Knapsack Problem

Problem Statement

The capacity of the knapsack ***W* = 60** is given and the list of provided items are shown in the following table. Determine the list of items a person can carry in the knapsack with maximum profits?

Item	**A**	**B**	**C**	**D**
Profit	280	100	120	120
Weight	40	10	20	24

~Dr Gaurav Kumar, Asst. Prof, CEA, GLAU

Practice Questions on Fractional Knapsack Problem

Answer

Step-1, Calculate Profit and Weight Ration

Item	A	B	C	D
Profit (P)	280	100	120	120
Weight (W)	40	10	20	24
Ratio (P_i/W_i)	7	10	6	5

Step-2, After sorting, the items are as shown in the following table.

Item	B	A	C	D
Profit (P)	100	280	120	120
Weight (W)	10	40	20	24
Ratio (P_i/W_i)	10	7	6	5

~Dr Gaurav Kumar, Asst. Prof, CEA, GLAU

Practice Questions on Fractional Knapsack Problem

Step 3

After sorting all the items according to (P_i/W_i)

- First all of ***B*** is chosen as weight of ***B*** is less than the capacity of the knapsack.
- Next, item ***A*** is chosen, as the available capacity of the knapsack is greater than the weight of ***A***.
- Now, ***C*** is chosen as the next item.

However, the whole item cannot be chosen as the remaining capacity of the knapsack is less than the weight of ***C***.

Hence, fraction of ***C*** (i.e. (60 − 50)/20) is chosen.

Now, the capacity of the Knapsack is equal to the selected items. <B, A, (10/20)C>

Hence, no more item can be selected.

The total weight of the selected items is **10 + 40 + 20 * (10/20) = 60**

And the total profit is **100 + 280 + 120 * (10/20) = 380 + 60 = 440**

This is the optimal solution.

We cannot gain more profit selecting any different combination of items.

~Dr Gaurav Kumar, Asst. Prof, CEA, GLAU

Design and Analysis of Algorithm of Fractional Knapsack Problem

P= Profit, W= Weight of an Item, M= Capacity of Knapsack,
n= number of items

Knapsack Algorithm

Step 1- For i=1 to n
calculate P_i/W_i (Profit/Weight) ratio → n

Step 2- Sort the items in decreasing order of P_i/W_i → nlogn

Step 3- For i=1 to n
Step 3a if M>0 and W_i <=M then
M=M- W
P= P + P_i
else
break
Step 3 b If M >0
P=P+ P_i (M/W_i) → n

Total Time= n + nlogn +n

TC= O(nlogn)

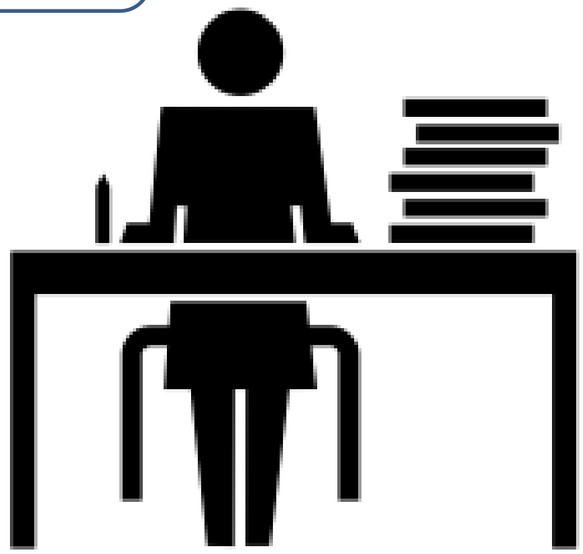
~Dr Gaurav Kumar, Asst. Prof, CEA, GLAU

02

Job Sequencing with Deadlines

~Dr Gaurav Kumar, Asst. Prof, CEA, GLAU

2. Job Sequencing with Deadlines



The sequencing of jobs on a single processor with deadline constraints is called as Job Sequencing with Deadlines.

Jobs	J1	J2	J3	J4	J5	J6	J7
Profits	35	30	25	20	15	12	5
Deadlines	3	4	4	2	3	1	2

Problem Statement

How can the total profit be maximized if only one job can be completed at a time?

~Dr Gaurav Kumar, Asst. Prof, CEA, GLAU

2. Job Sequencing with Deadlines

Here, For a given set of jobs

❖The profit of a job is given only when that job is completed within its deadline.

❖ Only one processor is available for processing all the jobs.

❖ Processor takes one unit of time to complete a job.

Jobs	J1	J2	J3	J4	J5	J6	J7
Profits	35	30	25	20	15	12	5
Deadlines	3	4	4	2	3	1	2

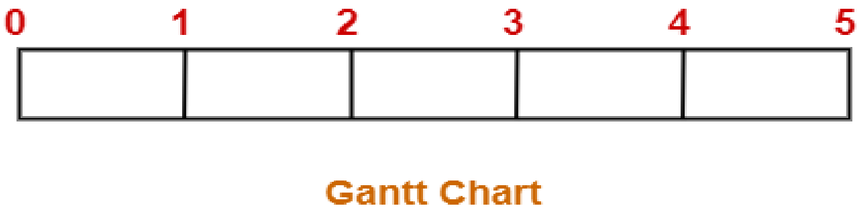
Job Sequencing Greedy Algorithm

Greedy Algorithm for Job Sequencing with Deadlines

Step-01: Sort all the given jobs in decreasing order of their profit.

Step-02:

- Check the value of maximum deadline.
- Draw a **Gantt chart** where maximum time on Gantt chart is the value of maximum deadline.



Gantt Chart

Step-03:

- Pick up the jobs one by one.
- Put the job on Gantt chart as far as possible from 0 ensuring that the job gets completed before its deadline.

~Dr Gaurav Kumar, Asst. Prof, CEA, GLAU

Practice Problem Based On Job Sequencing With Deadlines

Given the jobs, their deadlines and associated profits as shown

Jobs	J1	J2	J3	J4	J5	J6
Deadlines	5	3	3	2	4	2
Profits	200	180	190	300	120	100

Answer the following questions-

1. Write the optimal schedule that gives maximum profit.
2. Are all the jobs completed in the optimal schedule?
3. What is the maximum earned profit?

~Dr Gaurav Kumar, Asst. Prof, CEA, GLAU

Practice Problem Based On Job Sequencing With Deadlines

Jobs	J1	J2	J3	J4	J5	J6
Deadlines	5	3	3	2	4	2
Profits	200	180	190	300	120	100

Step-01: Sort all the given jobs in decreasing order of their profit

Jobs	J4	J1	J3	J2	J5	J6
Deadlines	2	5	3	3	4	2
Profits	300	200	190	180	120	100

~Dr Gaurav Kumar, Asst. Prof, CEA, GLAU

Practice Problem Based On Job Sequencing With Deadlines

Jobs	J4	J1	J3	J2	J5	J6
Deadlines	2	5	3	3	4	2
Profits	300	200	190	180	120	100

Step-02:

- Value of maximum deadline = 5.
- So, draw a Gantt chart with maximum time on Gantt chart = 5 units as shown

Gantt Chart

Now,

We take each job one by one in the order they appear in Step-01.

We place the job on Gantt chart as far as possible from 0.

~Dr Gaurav Kumar, Asst. Prof, CEA, GLAU

Practice Problem Based On Job Sequencing With Deadlines

Jobs	J4	J1	J3	J2	J5	J6
Deadlines	2	5	3	3	4	2
Profits	300	200	190	180	120	100

Step-03: We take job J4.

Since its deadline is 2, so we place it in the first empty cell before deadline 2 as-

~Dr Gaurav Kumar, Asst. Prof, CEA, GLAU

Practice Problem Based On Job Sequencing With Deadlines

Jobs	J4	J1	J3	J2	J5	J6
Deadlines	2	5	3	3	4	2
Profits	300	200	190	180	120	100

Step-04: We take job J1.

Since its deadline is 5, so we place it in the first empty cell before deadline 5 as-

~Dr Gaurav Kumar, Asst. Prof, CEA, GLAU

Practice Problem Based On Job Sequencing With Deadlines

Jobs	J4	J1	J3	J2	J5	J6
Deadlines	2	5	3	3	4	2
Profits	300	200	190	180	120	100

Step-05 We take job J3.

Since its deadline is 3, so we place it in the first empty cell before deadline 3 as-

~Dr Gaurav Kumar, Asst. Prof, CEA, GLAU

Practice Problem Based On Job Sequencing With Deadlines

Jobs	J4	J1	J3	J2	J5	J6
Deadlines	2	5	3	3	4	2
Profits	300	200	190	180	120	100

Step-06: We take job J2.

- Since its deadline is 3, so we place it in the first empty cell before deadline 3.
- Since the second and third cells are already filled, so we place job J2 in the first cell as-

~Dr Gaurav Kumar, Asst. Prof, CEA, GLAU

Practice Problem Based On Job Sequencing With Deadlines

Jobs	J4	J1	J3	J2	J5	J6
Deadlines	2	5	3	3	4	2
Profits	300	200	190	180	120	100

Step-07 Now, we take job J5.

Since its deadline is 4, so we place it in the first empty cell before deadline 4 as-

Now,

The only job left is job J6 whose deadline is 2.

All the slots before deadline 2 are already occupied. Thus, job J6 can not be completed.

~Dr Gaurav Kumar, Asst. Prof, CEA, GLAU

Practice Problem Based On Job Sequencing With Deadlines

Jobs	J4	J1	J3	J2	J5	J6
Deadlines	2	5	3	3	4	2
Profits	300	200	190	180	120	100

Now, we can answer the following questions-

1. **Write the optimal schedule that gives maximum profit?**

The optimal schedule is-

J2 , J4 , J3 , J5 , J1

This is the required order in which the jobs must be completed in order to obtain the maximum profit.

~Dr Gaurav Kumar, Asst. Prof, CEA, GLAU

Practice Problem Based On Job Sequencing With Deadlines

Jobs	J4	J1	J3	J2	J5	J6
Deadlines	2	5	3	3	4	2
Profits	300	200	190	180	120	100

Now, we can answer the following questions-.

2. Are all the jobs completed in the optimal schedule?

All the jobs are not completed in optimal schedule. This is because job J6 could not be completed within its deadline.

~Dr Gaurav Kumar, Asst. Prof, CEA, GLAU

Practice Problem Based On Job Sequencing With Deadlines

Jobs	J4	J1	J3	J2	J5	J6
Deadlines	2	5	3	3	4	2
Profits	300	200	190	180	120	100

Now, we can answer the following questions-.

3. What is the maximum earned profit?

Maximum earned profit = Sum of profit of all the jobs in optimal schedule

= Profit of job J2 + Profit of job J4 + Profit of job J3 + Profit of job J5

+ Profit of job J1

= 180 + 300 + 190 + 120 + 200 = **990 units**

~Dr Gaurav Kumar, Asst. Prof, CEA, GLAU

Practice Problem Based On Job Sequencing With Deadlines

Job	J_1	J_2	J_3	J_4	J_5
Deadline	2	1	3	2	1
Profit	60	100	20	40	20

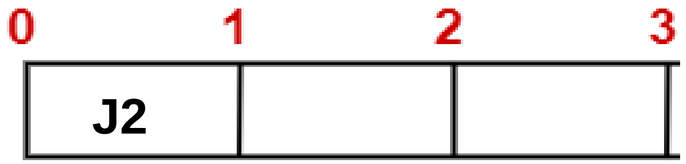
Find a sequence of jobs, which will be completed within their deadlines and will give maximum profit ?

~Dr Gaurav Kumar, Asst. Prof, CEA, GLAU

Practice Problem Based On Job Sequencing With Deadlines

Solution

Job	J_2	J_1	J_4	J_3	J_5
Deadline	1	2	2	3	1
Profit	100	60	40	20	20



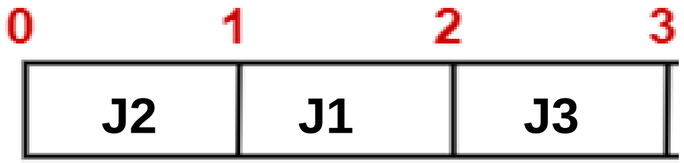
From this set of jobs, first we select J_2, as it can be completed within its deadline and contributes maximum profit.

~Dr Gaurav Kumar, Asst. Prof, CEA, GLAU

Practice Problem Based On Job Sequencing With Deadlines

Solution

Job	J_2	J_1	J_4	J_3	J_5
Deadline	1	2	2	3	1
Profit	100	60	40	20	20



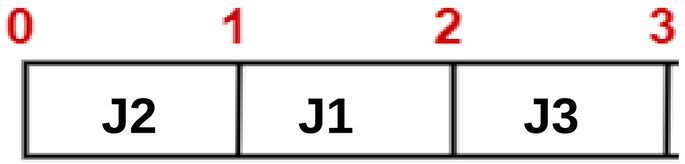
- Next, J_1 is selected as it gives more profit compared to J_4.
- In the next clock, J_4 cannot be selected as its deadline is over, hence J_3 is selected as it executes within its deadline.
- The job J_5 is discarded as it cannot be executed within its deadline.

~Dr Gaurav Kumar, Asst. Prof, CEA, GLAU

Practice Problem Based On Job Sequencing With Deadlines

Solution

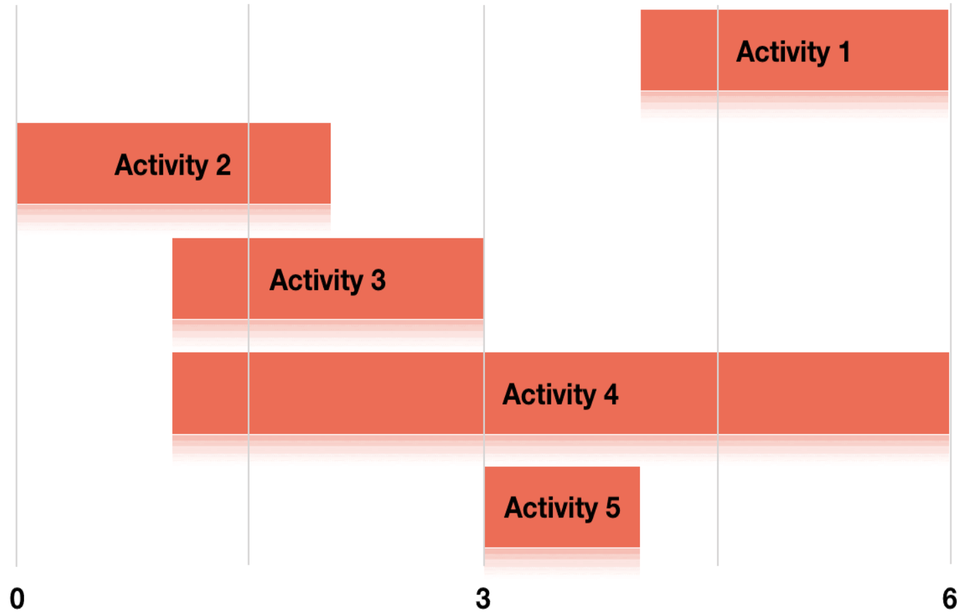
Job	J_2	J_1	J_4	J_3	J_5
Deadline	1	2	2	3	1
Profit	100	60	40	20	20



Thus, the solution is the sequence of jobs (J_2, J_1, J_3), which are being executed within their deadline and gives maximum profit.

Total profit of this sequence is **100 + 60 + 20 = 180**.

~Dr Gaurav Kumar, Asst. Prof, CEA, GLAU



03

Activity Selection Problem

~Dr Gaurav Kumar, Asst. Prof, CEA, GLAU

Understanding the Activity Selection Problem

Let's consider that you have n activities with their start and finish times, the objective is to find solution set having maximum number of non-conflicting activities that can be executed in a single time frame, assuming that only one person or machine or resource is available for execution.

Class Room

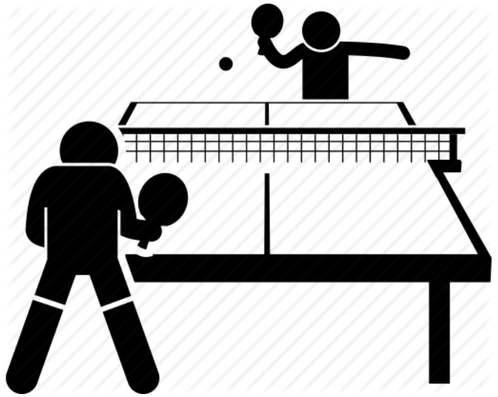


Table Tennis

Note: Only one activity can be executed in a Class Room or a team can play a game at a time.

~Dr Gaurav Kumar, Asst. Prof, CEA, GLAU

Understanding the Activity Selection Problem

Activity (a_i)	1	2	3	4	5
s_i	4	0	1	1	3
f_i	6	2	3	6	4

a3 and a5 are non conflicting activities

a3 and a4 are conflicting activities

Some points to note here

- It might not be possible to complete all the activities, since their timings can collapse.
- Two activities, say i and j are said to be **non-conflicting** if $s_j >= f_i$ where s_j denote the starting time of activities j and f_i refer to the finishing time of the activity i.

Or (Start time of second activity should be greater or equal to the finish time of first activity then only first and second activity are said to be non conflicting activities.

~Dr Gaurav Kumar, Asst. Prof, CEA, GLAU

Solving Activity Selection Problem

Problem Statement- We have 6 activities with corresponding start and end time, the objective is to compute an execution schedule having maximum number of non-conflicting activities.

Start Time (s)	Finish Time (f)	Activity Name
5	9	a1
1	2	a2
3	4	a3
0	6	a4
5	7	a5
8	9	a6

~Dr Gaurav Kumar, Asst. Prof, CEA, GLAU

Solving Activity Selection Problem

Problem Statement- We have 6 activities with corresponding start and end time, the objective is to compute an execution schedule having maximum number of non-conflicting activities.

Input Data for the Algorithm:

- act[] array containing all the activities.
- s[] array containing the starting time of all the activities.
- f[] array containing the finishing time of all the activities.

Output Data from the Algorithm:

- sol[] array referring to the solution set containing the maximum number of non-conflicting activities.

Start Time (s)	Finish Time (f)	Activity Name
5	9	a1
1	2	a2
3	4	a3
0	6	a4
5	7	a5
8	9	a6

~Dr Gaurav Kumar, Asst. Prof, CEA, GLAU

Algorithm for Solving Activity Selection Problem

Following are the steps to solve the activity selection problem,

Step 1: Sort the given activities in ascending order according to their finishing time.

Step 2: Select the first activity from sorted array act[] and add it to sol[] array.

Step 3: Repeat steps 4 and 5 for the remaining activities in act[].

Step 4: If the start time of the currently selected activity is greater than or equal to the finish time of previously selected activity, then add it to the sol[] array otherwise reject it.

Step 5: Select the next activity in act[] array.

Step 6: Print the sol[] array.

~Dr Gaurav Kumar, Asst. Prof, CEA, GLAU

Solving Activity Selection Problem

Step 1: Sort the given activities in ascending order according to their finishing time.

Start Time (s)	Finish Time (f)	Activity Name
5	9	a1
1	2	a2
3	4	a3
0	6	a4
5	7	a5
8	9	a6

Start Time (s)	Finish Time (f)	Activity Name
1	2	a2
3	4	a3
0	6	a4
5	7	a5
5	9	a1
8	9	a6

~Dr Gaurav Kumar, Asst. Prof, CEA, GLAU

Solving Activity Selection Problem

Start Time (s)	Finish Time (f)	Activity Name
1	2	a2
3	4	a3
0	6	a4
5	7	a5
5	9	a1
8	9	a6

Step 2: Select the first activity from sorted array act[] and add it to the sol[] array, thus **sol = {a2}**.

Solving Activity Selection Problem

Start Time (s)	Finish Time (f)	Activity Name
1	2	a2
3	4	a3
0	6	a4
5	7	a5
5	9	a1
8	9	a6

Step 3: Repeat the steps 4 and 5 for the remaining activities in act[].

Step 4: If the start time of the currently selected activity is greater than or equal to the finish time of the previously selected activity, then add it to sol[].

Step 5: Select the next activity in act[]

Solving Activity Selection Problem

$S_{a3} > f_{a2}$ is true, select it.

Start Time (s)	Finish Time (f)	
1	2	a2
3	4	a3
0	6	a4
5	7	a5
5	9	a1
8	9	a6

Step 4: Select the a3 activity, its starting time is greater than ending time of previous selected activity, hence, it is added into the sol[] array, thus **sol = {a2, a3}**.

Step 5: Select the next activity in act[]

Solving Activity Selection Problem

Start Time (s)	Finish Time (f)	
1	2	
3	4	a3
0	6	a4
5	7	a5
5	9	a1
8	9	a6

$S_{a4} > f_{a3}$ is false, reject it

Step 4: Select the a4 activity, its starting time is not greater than ending time of previous selected activity, hence, it is rejected, thus **sol = {a2, a3} is remain same as earlier.**

Step 5: Select the next activity in act[]

~Dr Gaurav Kumar, Asst. Prof, CEA, GLAU

Solving Activity Selection Problem

Start Time (s)	Finish Time (f)	Activity
1	2	
3	4	
0	6	a4
5	7	a5
5	9	a1
8	9	a6

$S_{a5} > f_{a3}$

is true, add it

tep 4: Select the a5 activity, its starting time is greater than ending time of previous selected activity, hence, it is selected, and added in to the solution, thus **sol = {a2, a3, a5}.**

Step 5: Select the next activity in act[]

Solving Activity Selection Problem

Start Time (s)	Finish Time (f)	Activity Name
1	2	a2
3	4	
0	6	
5	7	a5
5	9	a1
8	9	a6

$S_{a1} > f_{a5}$ is false, reject it

Step 4: Select the a1 activity, its starting time is not greater than ending time of previous selected activity, hence, it is rejected, thus **sol = {a2, a3, a5} is remain same as earlier.**

Step 5: Select the next activity in act[]

Solving Activity Selection Problem

Start Time (s)	Finish Time (f)	Activity Name
1	2	a2
3	4	a3
0	6	
5	7	
5	9	a1
8	9	a6

$S_{a6} > f_{a5}$ is true, add it

Step 4: Select the a6 activity, its starting time is greater than ending time of previous selected activity, hence, it is selected and it is added in the solution,

Sol = {a2, a3, a5, a6}.

Step 5: Select the next activity in act[].

Solving Activity Selection Problem

Start Time (s)	Finish Time (f)	Activity Name
1	2	a2
3	4	a3
0	6	a4
5	7	a5
5	9	a1
8	9	a6

There is no more activity left.

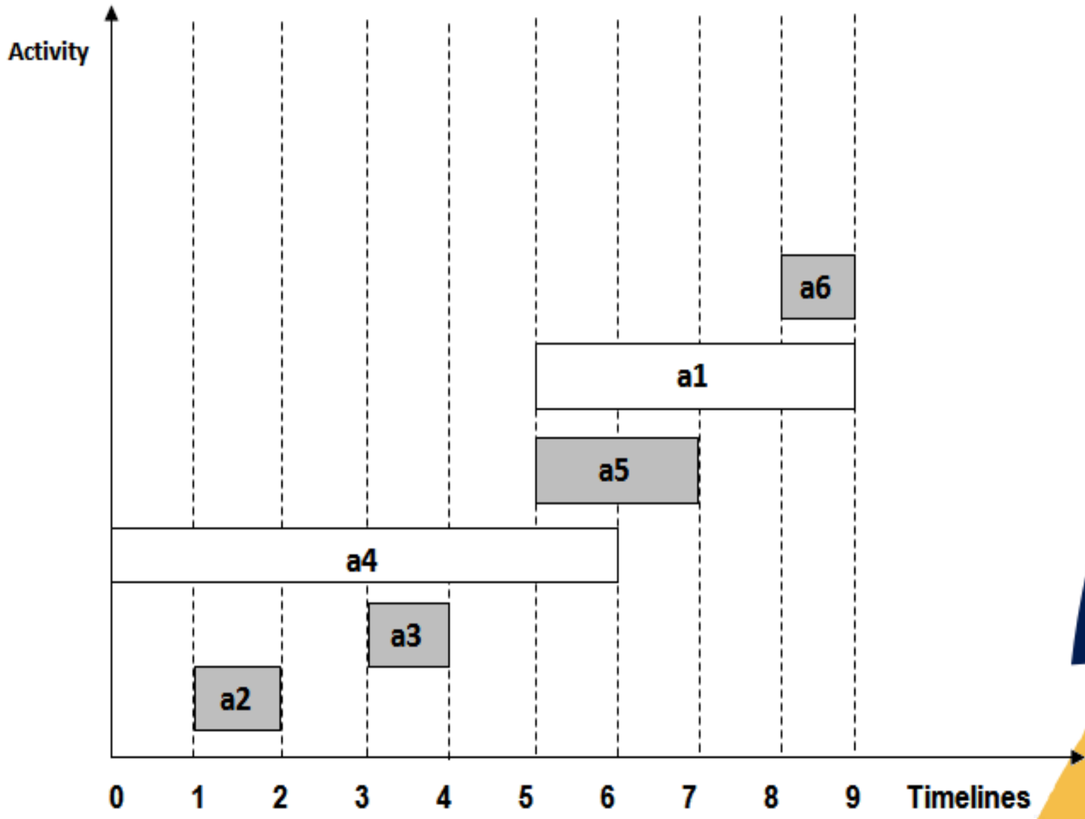
Step 6 : Print the Sol array.

The execution schedule of maximum number of non-conflicting activities will be:

= {a2, a3, a5, a6}

Through Graph Solving Activity Selection Problem

Start Time (s)	Finish Time (f)	Activity Name
1	2	a2
3	4	a3
0	6	a4
5	7	a5
5	9	a1
8	9	a6



In the above diagram, the selected activities have been highlighted in grey.

~Dr Gaurav Kumar, Asst. Prof, CEA, GLAU

Practice Question on Activity Selection Problem

Compute a schedule where the greatest number of activities takes place

$S = (A_1\ A_2\ A_3\ A_4\ A_5\ A_6\ A_7\ A_8$ A9 A10)

$S_i = (1,2,3,4,7,8,9,9,11,12)$

$f_i = (3,5,4,7,10,9,11,13,12,14)$

The final Activity Schedule is: $(A_1\ A_3\ A_4\ A_6\ A_7\ A_9\ A_{10})$

~Dr Gaurav Kumar, Asst. Prof, CEA, GLAU

Time Complexity Analysis for Solving Activity Selection Problem

Step 1: Sort the given activities in ascending order according to their finishing time. → **nlogn**

Step 2: Select the first activity from sorted array act[] and add it to sol[] array. → 1

Step 3: Repeat steps 4 and 5 for the remaining activities in act[].

Step 4: If the start time of the currently selected activity is greater than or equal to the finish time of previously selected activity, then add it to the sol[] array otherwise reject it. → 1

Step 5: Select the next activity in act[] array. → 1

n-1 times repeat

Step 6: Print the sol[] array.

Total Time= 2* (n-1) + nlogn + 1 = 2n + nlogn -1

TC= O(nlogn)

~Dr Gaurav Kumar, Asst. Prof, CEA, GLAU

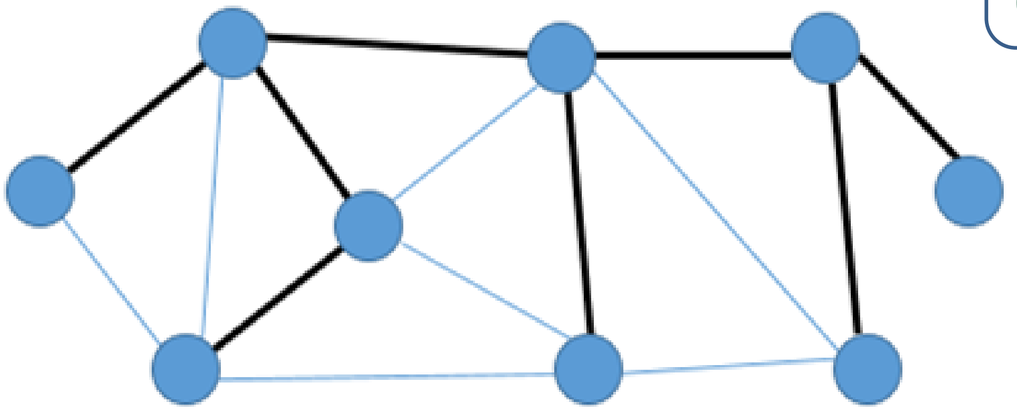
Time Complexity Analysis for Solving Activity Selection Problem

Some more scenarios of computing the time complexity of Activity Selection Algorithm:

Case 1: When a given set of activities is unsorted, then we will have to use the sort() method for sorting the activities list. The time complexity of this method will be **O(nlogn)**, which also defines complexity of the algorithm.

Case 2: When a given set of activities are already sorted according to their finishing time, then there is no sorting mechanism involved, in such a case the complexity of the algorithm will be **O(n)**

~Dr Gaurav Kumar, Asst. Prof, CEA, GLAU

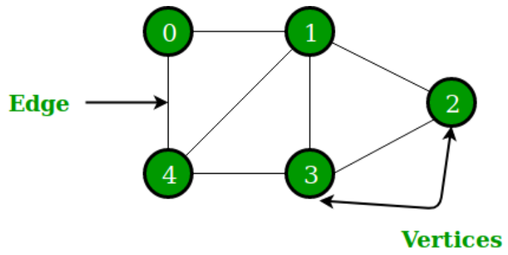


04

Minimum Spanning Trees

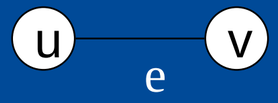
~Dr Gaurav Kumar, Asst. Prof, CEA, GLAU

Introduction to Graph



A graph G(V, E) consists of two sets

- a finite, nonempty set of vertices V(G) (points or vertices)
- a set of edges E, pair of nodes [u, v] in V, denoted by e=[u, v].

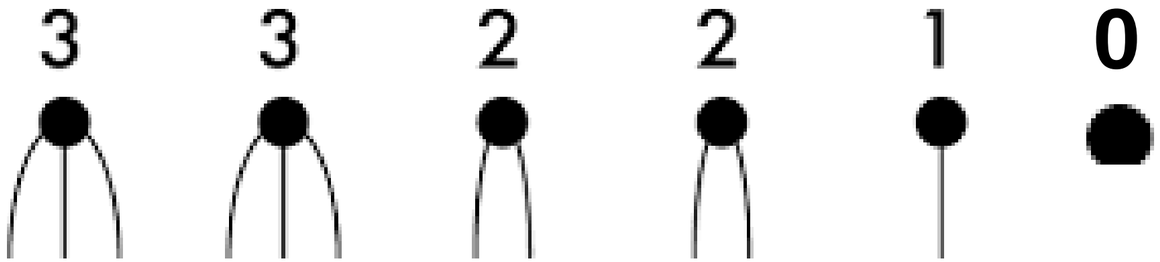


U and V are called endpoints of e, and also said to be adjacent or neighbors

Introduction of Graph

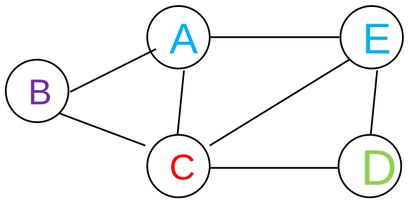
degree(u) is no. of edges containing u.

degree(u)=0 means if u does not belong to any edge (isolated node)



Introduction of Graph

A graph G is said to be connected if there is a path between any two of nodes.



There are 2 simple paths of length 2 from B to E-

(B, A, E) & (B, C, E).

There is only 1 simple path of length 2 from B to D : (B, C, D).

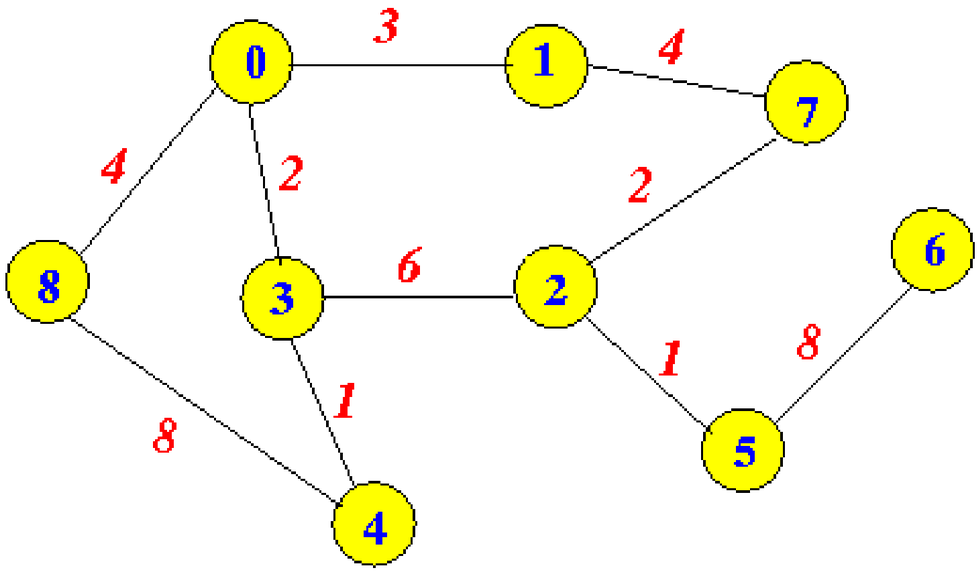
Deg(A)=3

Deg(C) = 4

Deg(D)=2

Introduction of Graph

A graph G is said to be labeled or weighted if its edges are assigned data. .

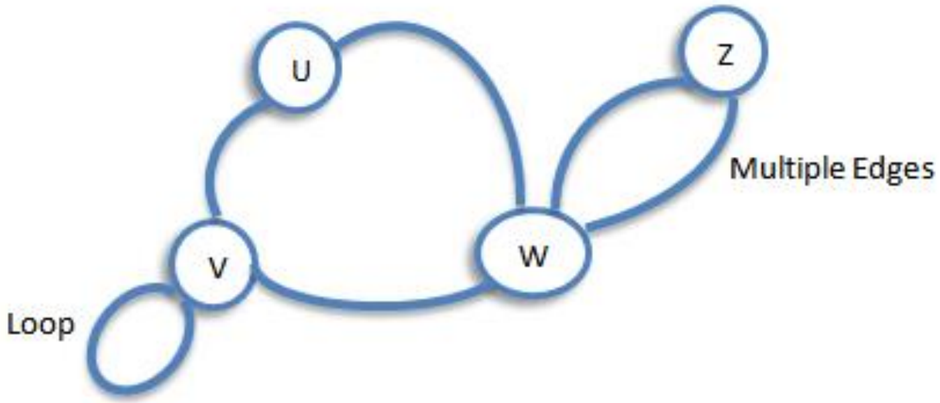


~Dr Gaurav Kumar, Asst. Prof, CEA, GLAU

Introduction of Graph

Multiple edges connect the same endpoints (e4 and e5 connects C and D).

An edge is called Loop if it has identical endpoints i.e. e6 connects (D, D)



Introduction of Graph

A Directed Graph G, also called a digraph or graph in which each edge is assigned a direction.

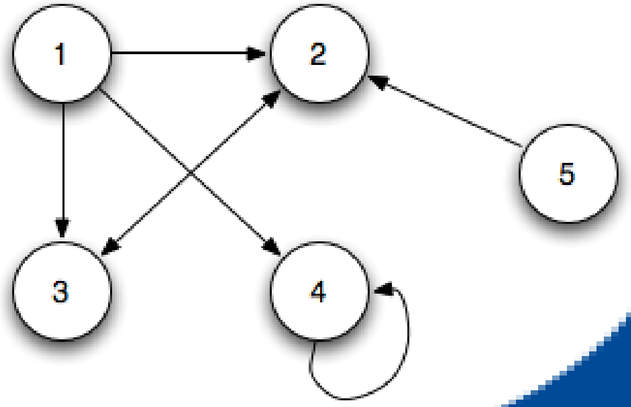
Suppose G is directed graph with directed edge e=(u, v).

e begins at u and ends at v. (example 1 and 2)

u is origin and v is destination.

u is predecessor of v and v is successor of u.

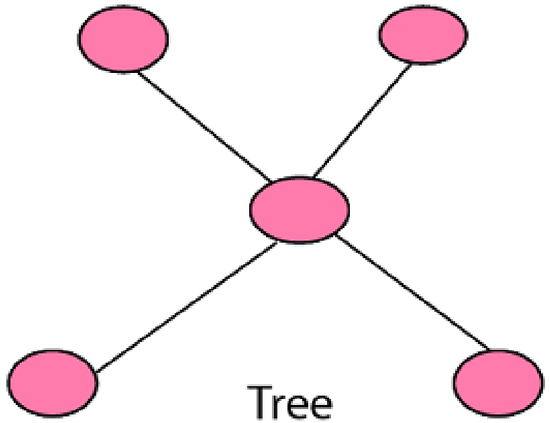
u is adjacent to v.



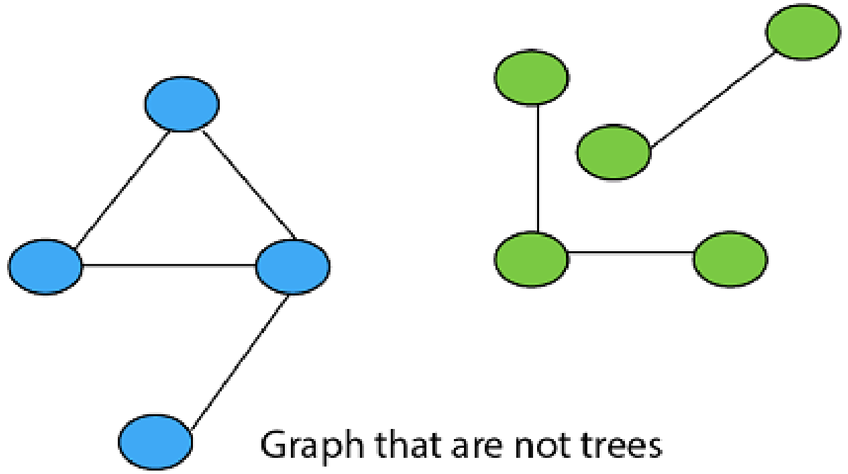
Introduction of Tree Graph

A tree is a graph with the following properties:

- The graph is connected (can go from anywhere to anywhere)
- There are no cyclic (Acyclic)



Tree



Graph that are not trees

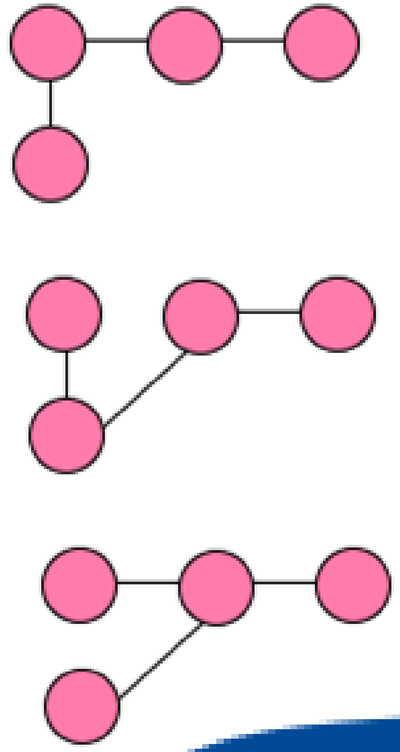
~Dr Gaurav Kumar, Asst. Prof, CEA, GLAU

Introduction of Spanning Tree

A spanning tree is a subset of an undirected Graph that has all the vertices connected by minimum number of edges.

Properties

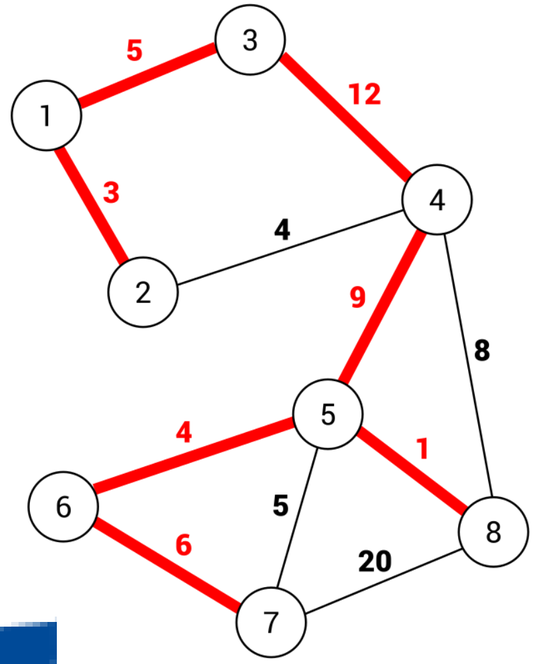
- A spanning tree does not have any cycle.
- Any vertex can be reached from any other vertex.

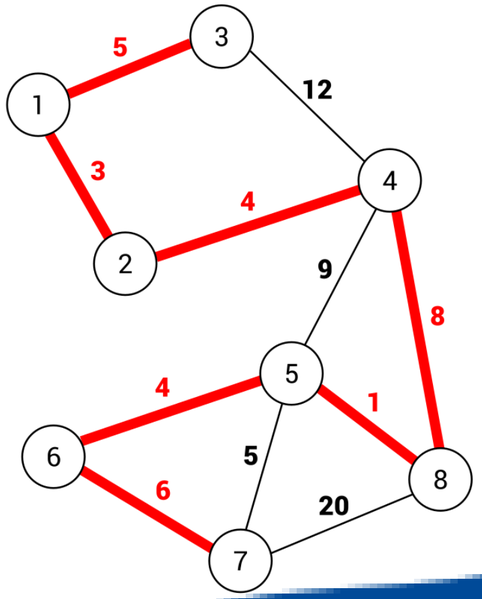


~Dr Gaurav Kumar, Asst. Prof, CEA, GLAU

Introduction of Minimum Cost Spanning Tree

- Minimum Spanning Tree is a Spanning Tree which has minimum total cost.
- If we have a linked undirected graph with a weight (or cost) combine with each edge. Then the cost of spanning tree would be the sum of the cost of its edges.

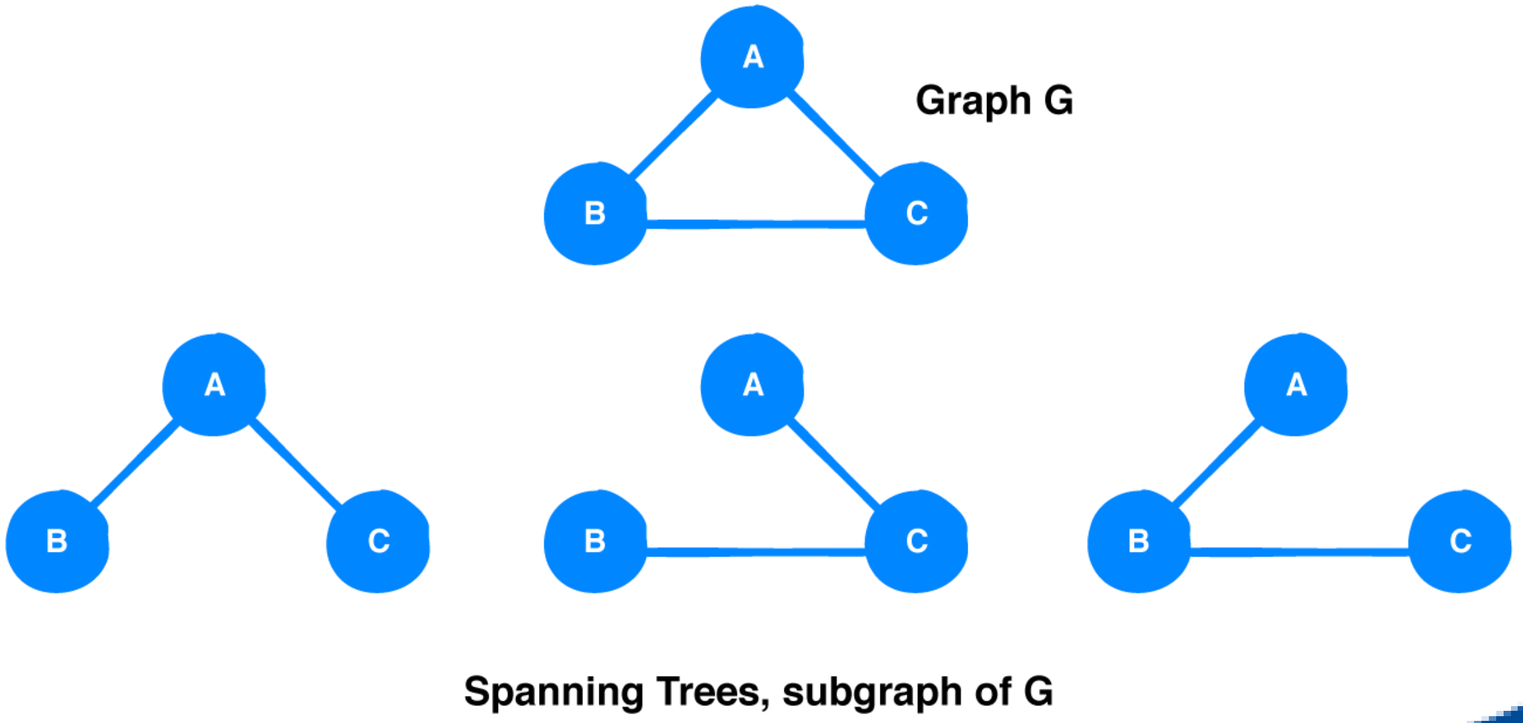




~Dr Gaurav Kumar, Asst. Prof, CEA, GLAU

More Properties of Spanning Tree

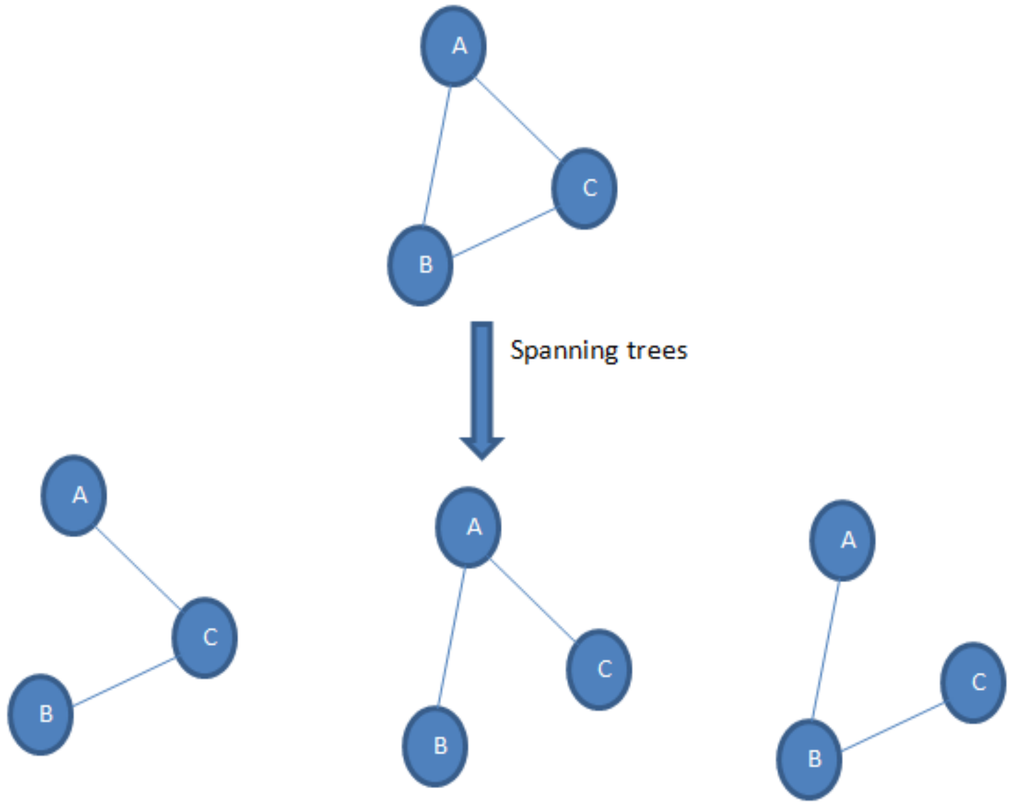
1. There may be several minimum spanning trees of the same weight having the minimum number of edges.



~Dr Gaurav Kumar, Asst. Prof, CEA, GLAU

More Properties of Spanning Tree

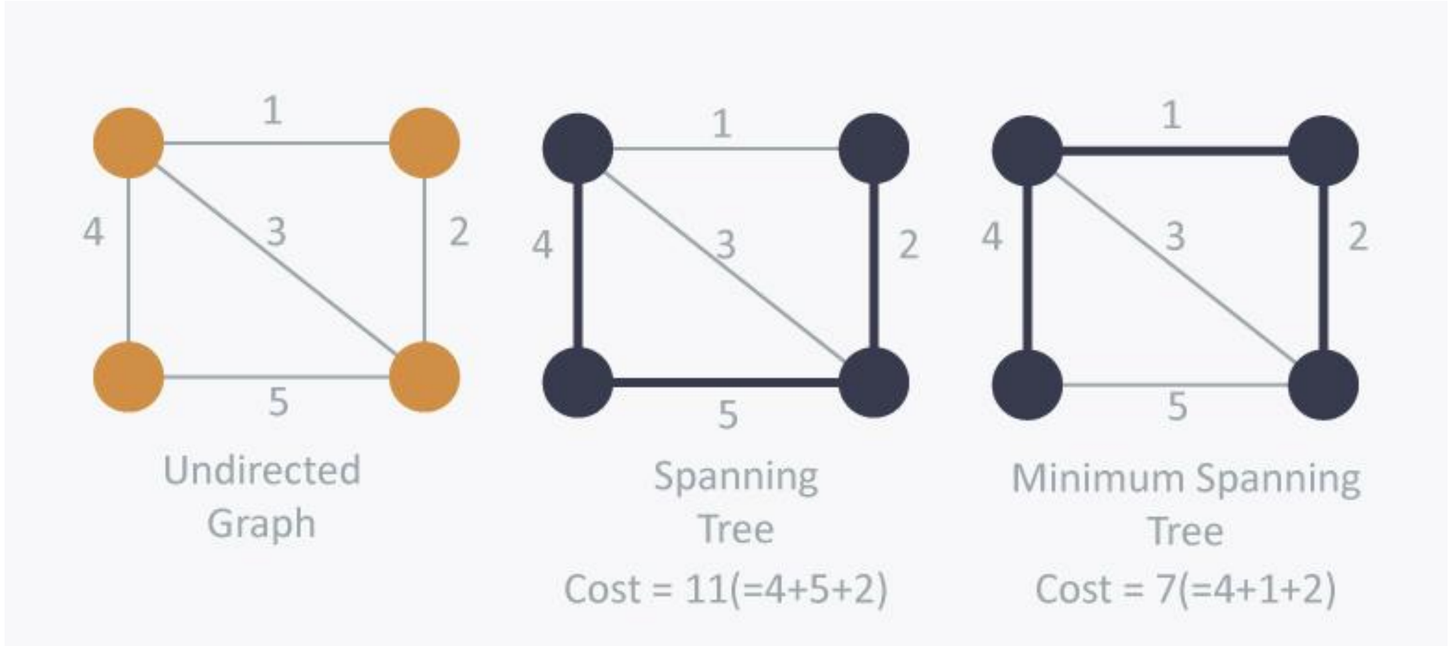
2. If all the edge weights of a given graph are the same, then every spanning tree of that graph is minimum.



~Dr Gaurav Kumar, Asst. Prof, CEA, GLAU

More Properties of Spanning Tree

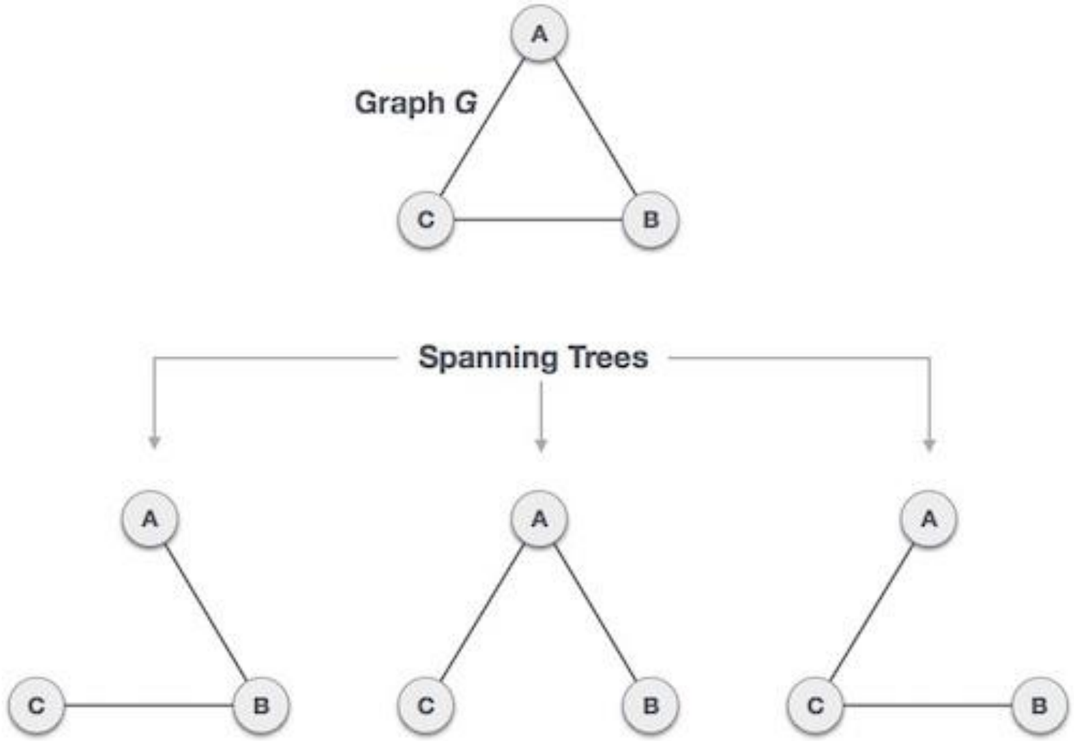
3. If each edge has a distinct weight, then there will be only one, unique minimum spanning tree.



~Dr Gaurav Kumar, Asst. Prof, CEA, GLAU

More Properties of Spanning Tree

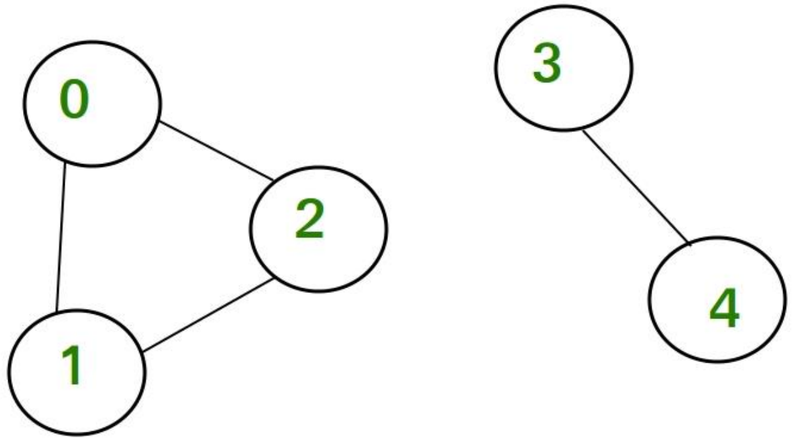
4. A connected graph G can have more than one spanning trees.

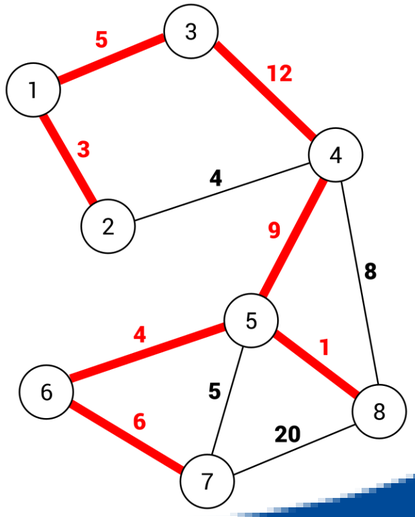


~Dr Gaurav Kumar, Asst. Prof, CEA, GLAU

More Properties of Spanning Tree

5. A disconnected graph can't have to span the tree, or it can't span all the vertices.





6. Spanning Tree doesn't contain cycles.

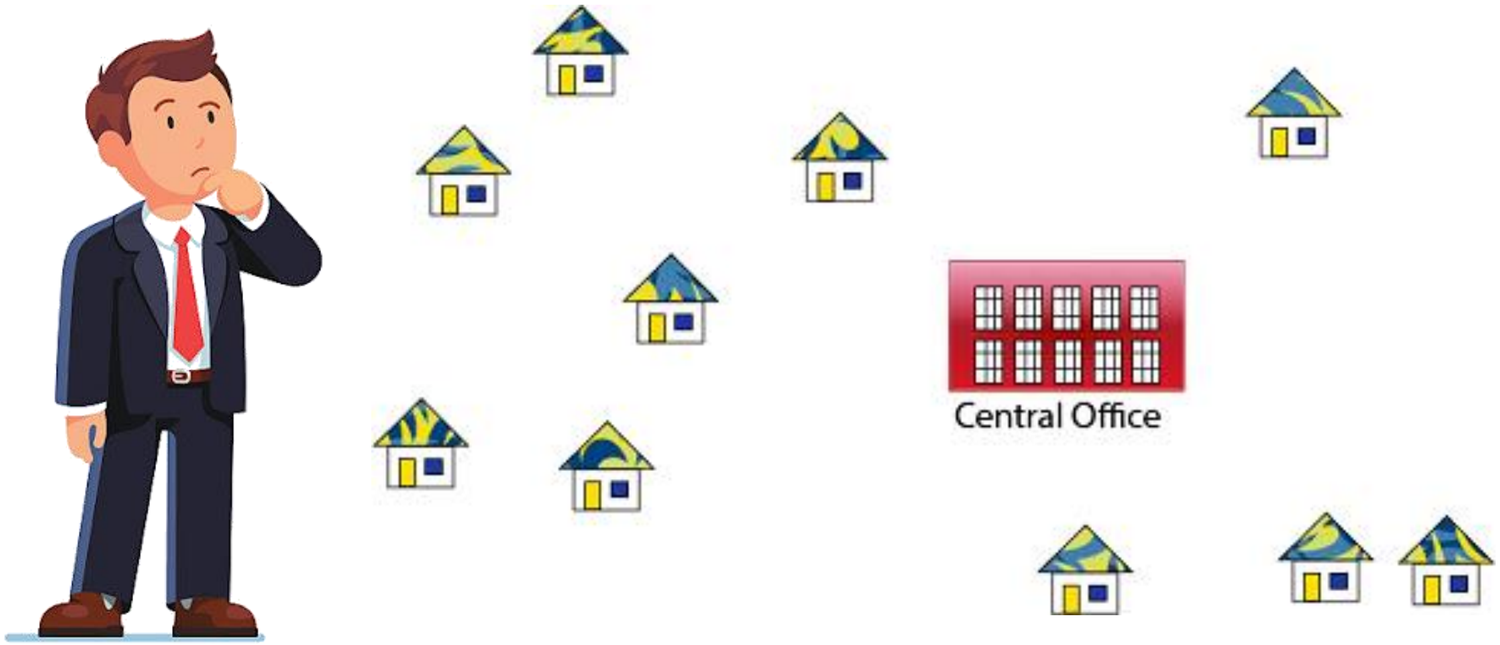
7. Spanning Tree has **(n-1) edges** where n is the number of vertices.

~Dr Gaurav Kumar, Asst. Prof, CEA, GLAU

Application of Minimum Spanning Tree

Consider n stations **are to be linked using a communication network** &

laying of communication links between any two stations involves a cost.

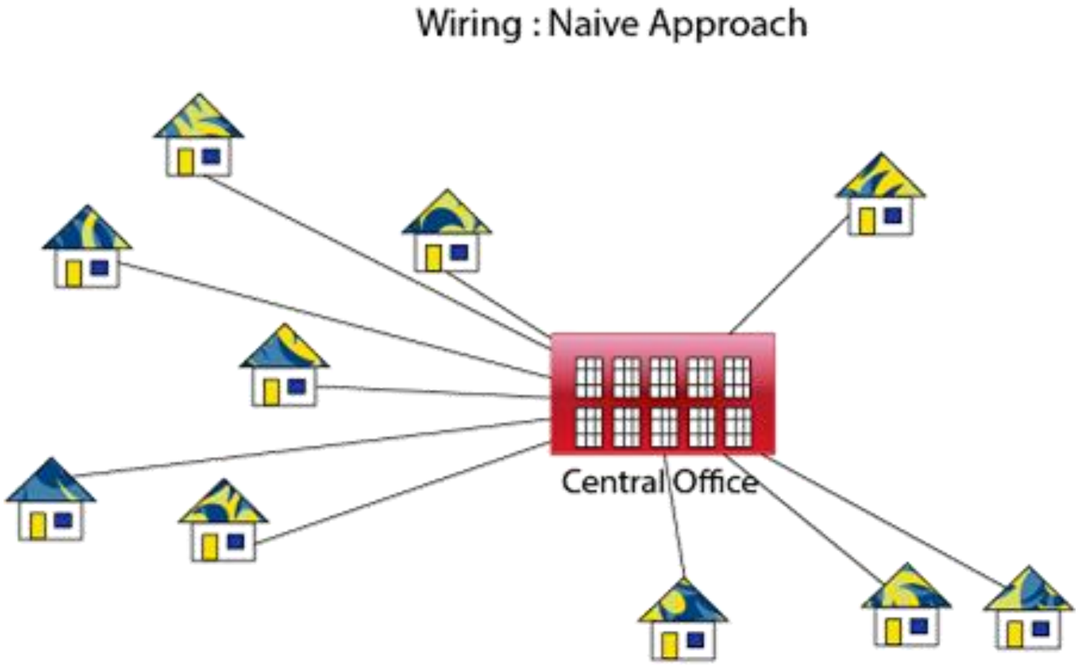


Application of Minimum Spanning Tree

1. Consider n stations **are to be linked using a communication network** &

laying of communication links between any two stations involves a cost.

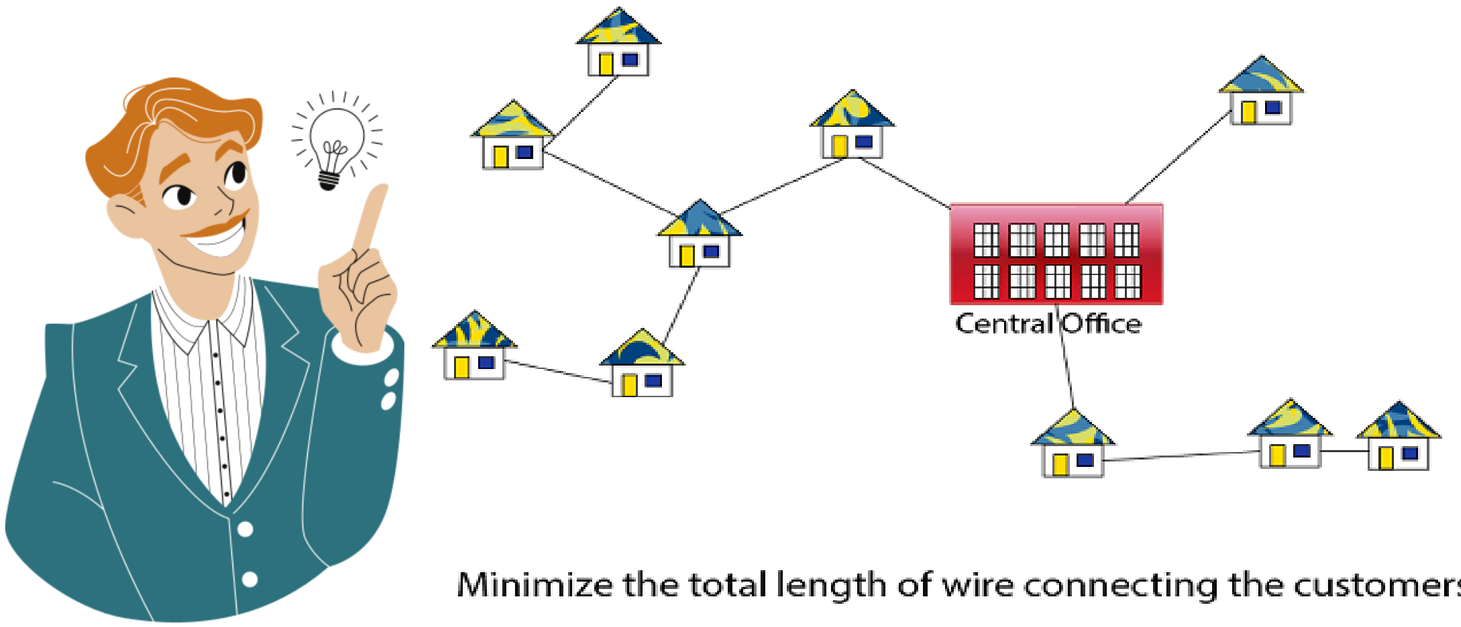
It is very Expensive



Application of Minimum Spanning Tree

Consider n stations **are to be linked using a communication network** & laying of communication links between any two stations involves a cost.

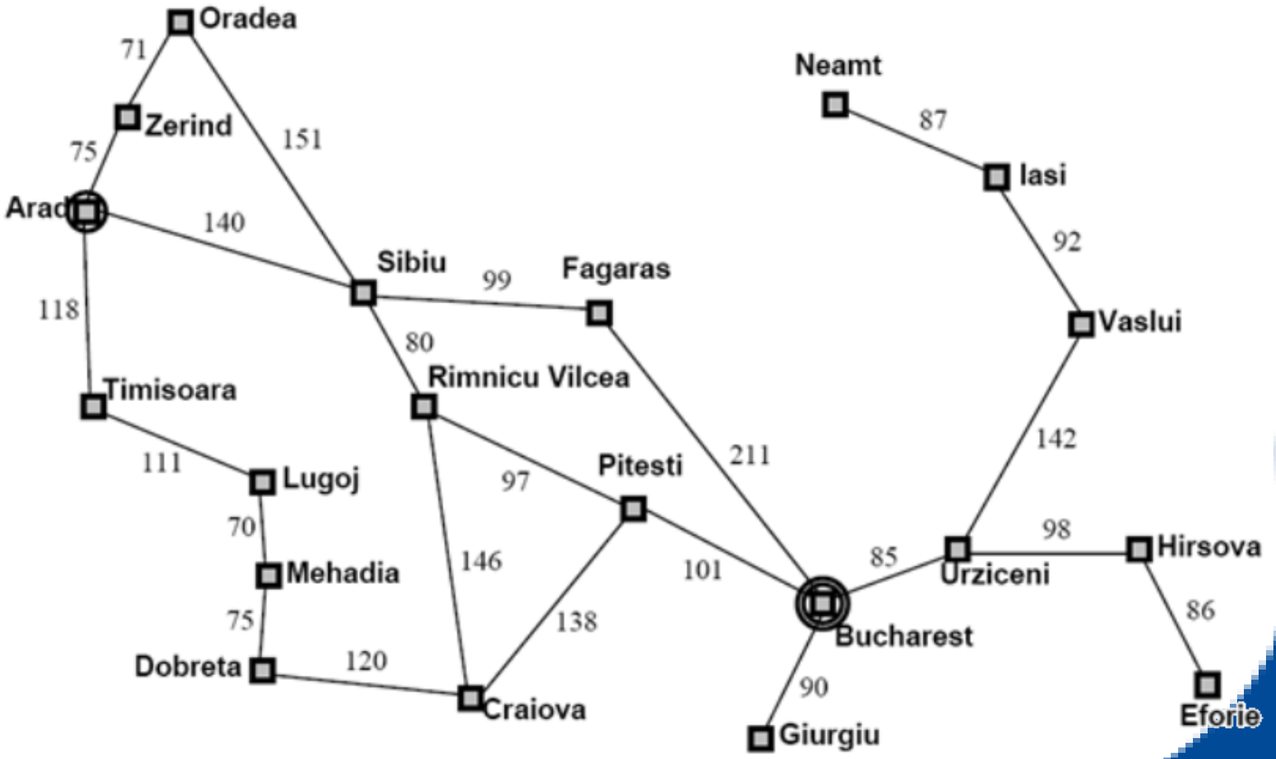
The ideal solution would be to extract a subgraph termed as minimum cost spanning tree.



~Dr Gaurav Kumar, Asst. Prof, CEA, GLAU

Application of Minimum Spanning Tree

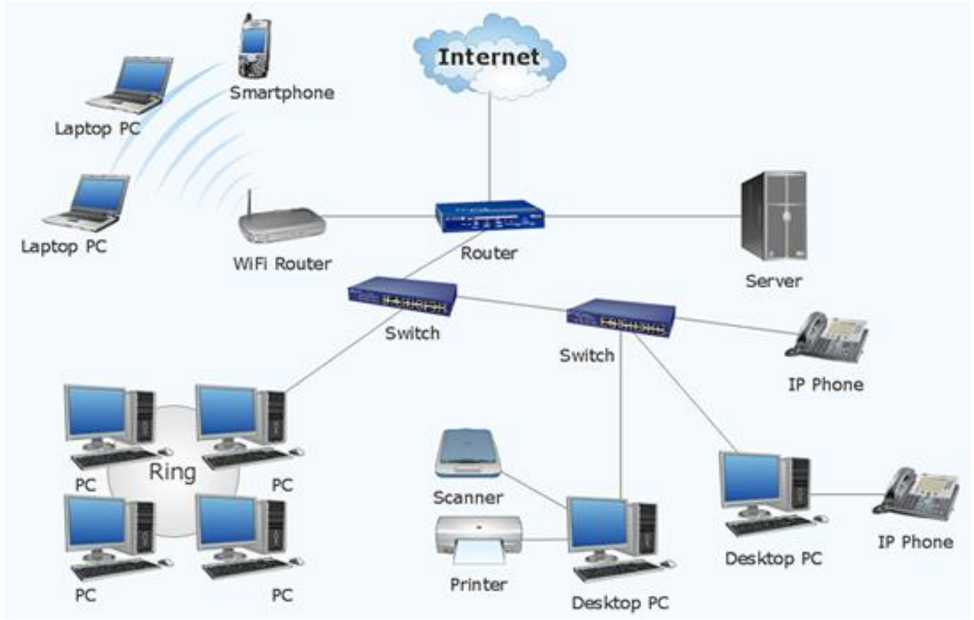
2. Suppose you want to construct highways or railroads spanning several cities then we can use the concept of minimum spanning trees



~Dr Gaurav Kumar, Asst. Prof, CEA, GLAU

Application of Minimum Spanning Tree

3. Designing Local Area Networks.

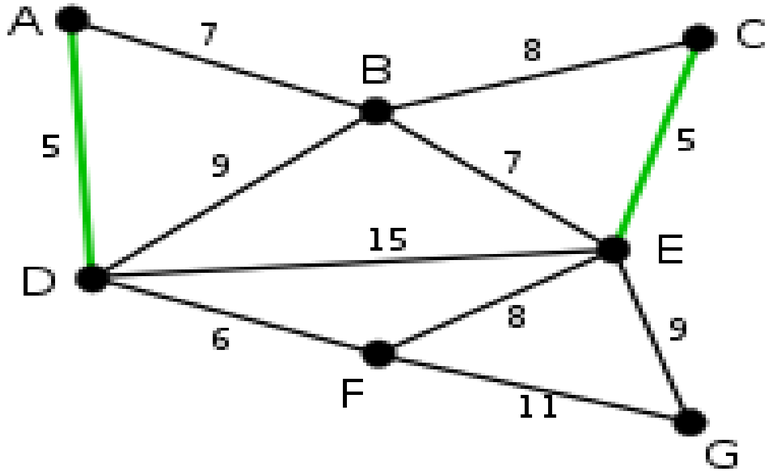


4. Laying pipelines connecting offshore drilling sites, refineries and consumer markets.

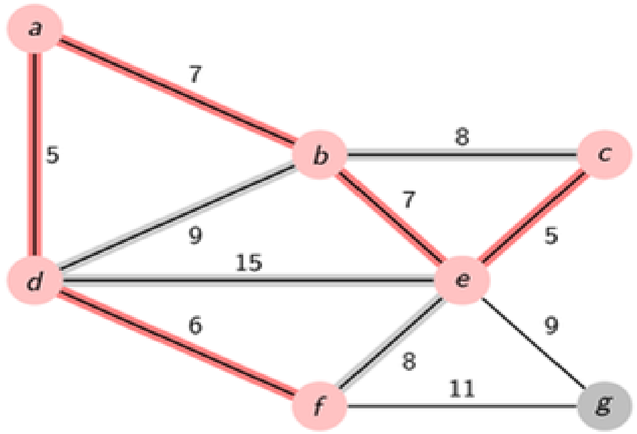
Methods to Estimate Minimum Cost Spanning Tree

There are two methods to find Minimum Spanning Tree

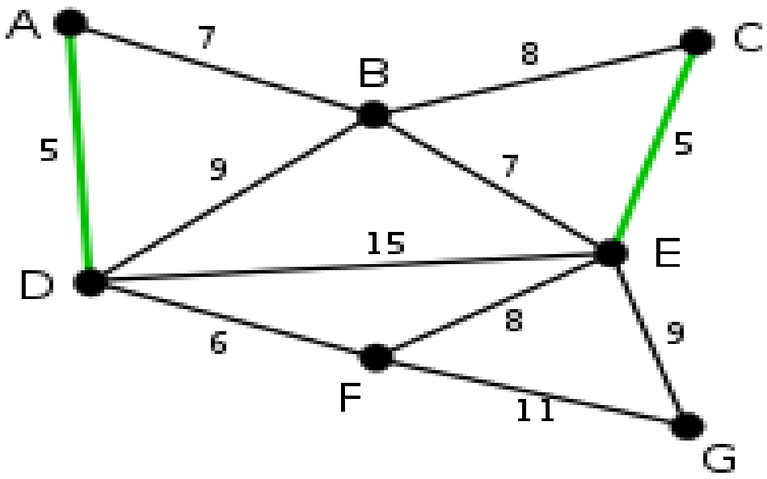
1. **Kruskal's Algorithm**



2. **Prim's Algorithm**



~Dr Gaurav Kumar, Asst. Prof, CEA, GLAU

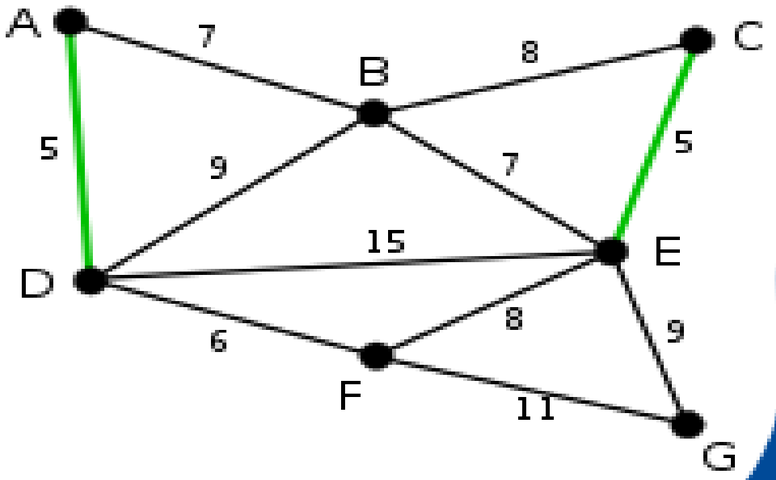


04.a

Kruskal's Algorithm

~Dr Gaurav Kumar, Asst. Prof, CEA, GLAU

What is Kruskal's Method

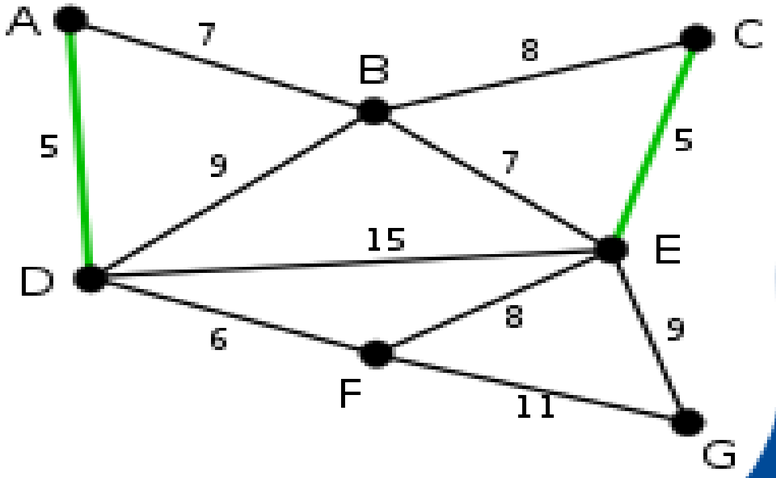


Kruskal's algorithm is a minimum spanning tree algorithm that takes a graph as input and finds the subset of the edges of that graph which

- **form a tree that includes every vertex**
- **has the minimum sum of weights among all the trees that can be formed from the graph**

~Dr Gaurav Kumar, Asst. Prof, CEA, GLAU

Kruskal's Algorithm



1. Sort all the edges from low weight to high

2. Take the edge with the lowest weight and add it to the spanning tree. If adding the edge created a cycle, then reject this edge.

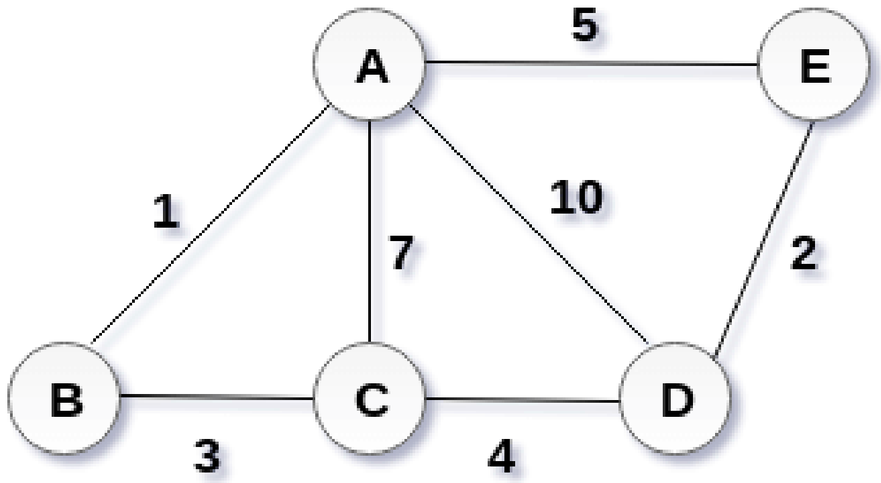
3. Keep adding edges until we reach all vertices.

Or *Repeat step#2 until there are (V-1) edges in the spanning tree.*

~Dr Gaurav Kumar, Asst. Prof, CEA, GLAU

Practice Example of Kruskal's Algorithm

Problem



The weight of the edges given as

Edge	AE	AD	AC	AB	BC	CD	DE
Weight	5	10	7	1	3	4	2

Practice Example of Kruskal's Algorithm

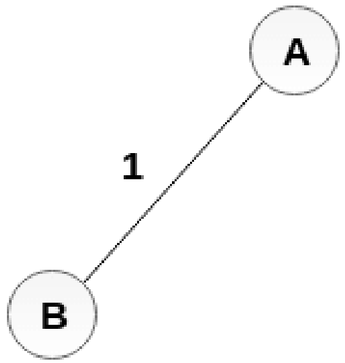
Step-1 Sort the edges according to their weights.

Edge	AB	DE	BC	CD	AE	AC	AD
Weight	1	2	3	4	5	7	10

Start constructing the tree

Step-2 Select the lowest weight edge one by one (according to the table) and add to the MST, If it forms a cycle, then reject it otherwise continue…

Add AB to the MST

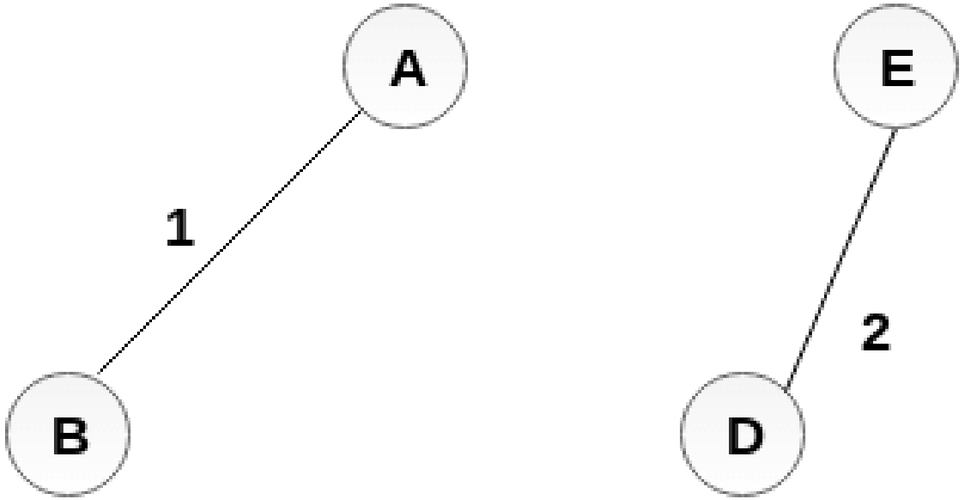


~Dr Gaurav Kumar, Asst. Prof, CEA, GLAU

Practice Example of Kruskal's Algorithm

Edge	AB	DE	BC	CD	AE	AC	AD
Weight	1	2	3	4	5	7	10

Add DE to the MST

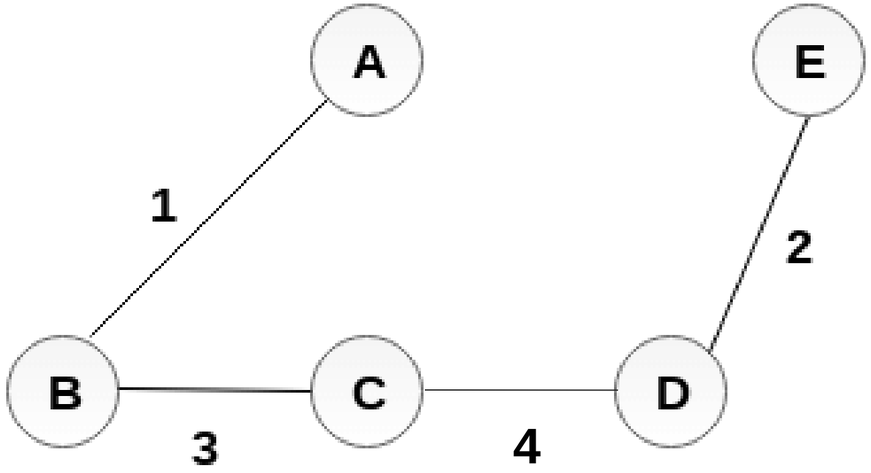


~Dr Gaurav Kumar, Asst. Prof, CEA, GLAU

Practice Example of Kruskal's Algorithm

Edge	AB	DE	BC	CD	AE	AC	AD
Weight	1	2	3	4	5	7	10

Add BC to the MST

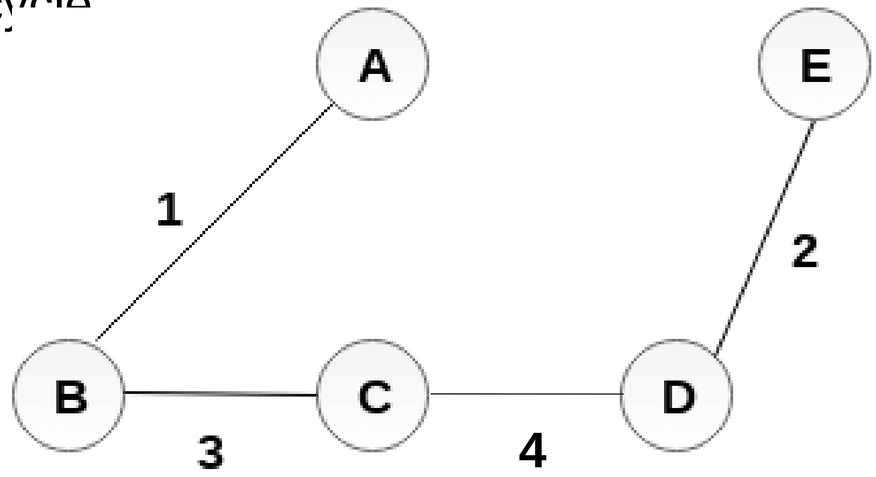


~Dr Gaurav Kumar, Asst. Prof, CEA, GLAU

Practice Example of Kruskal's Algorithm

Edge	AB	DE	BC	CD	AE	AC	AD
Weight	1	2	3	4	5	7	10

The next step is to **add AE, but we can't add** that as it will cause a cycle

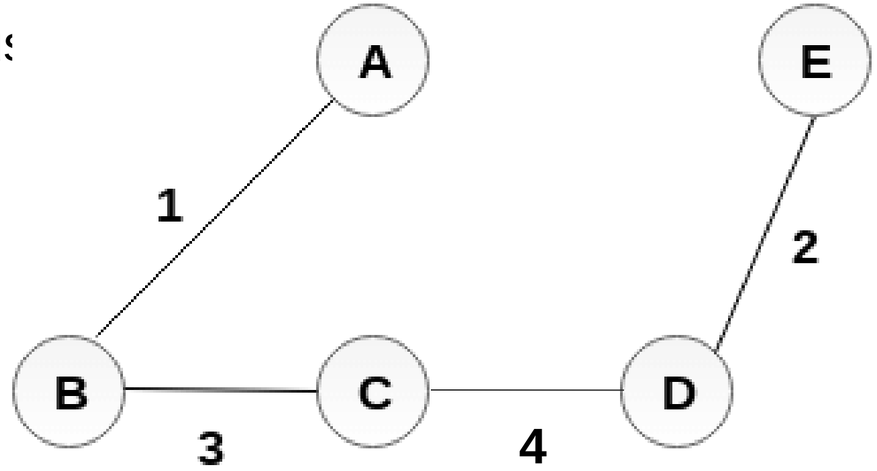


~Dr Gaurav Kumar, Asst. Prof, CEA, GLAU

Practice Example of Kruskal's Algorithm

Edge	AB	DE	BC	CD	AE	AC	AD
Weight	1	2	3	4	5	7	10

The next edge to be **added is AC, but it can't be added** as it will caus

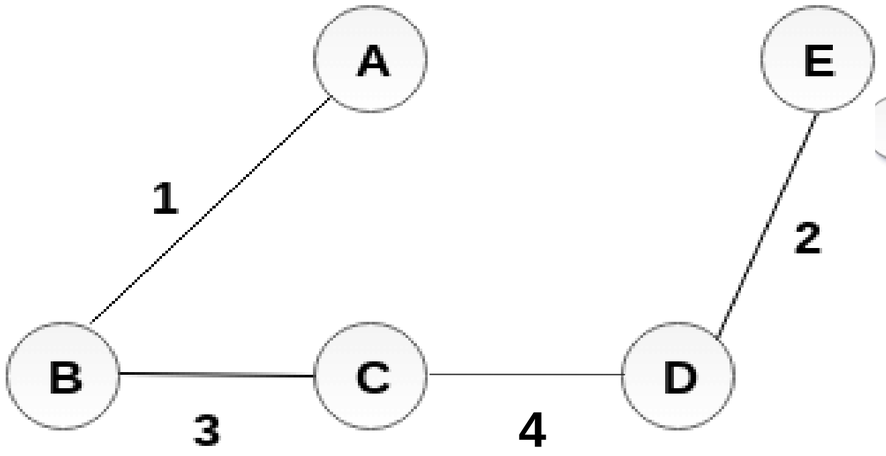


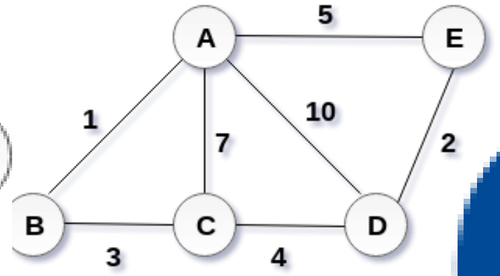
~Dr Gaurav Kumar, Asst. Prof, CEA, GLAU

Practice Example of Kruskal's Algorithm

Edge	AB	DE	BC	CD	AE	AC	AD
Weight	1	2	3	4	5	7	10

The next edge to be **added is AD, but it can't be added** as it will contain a cycle.



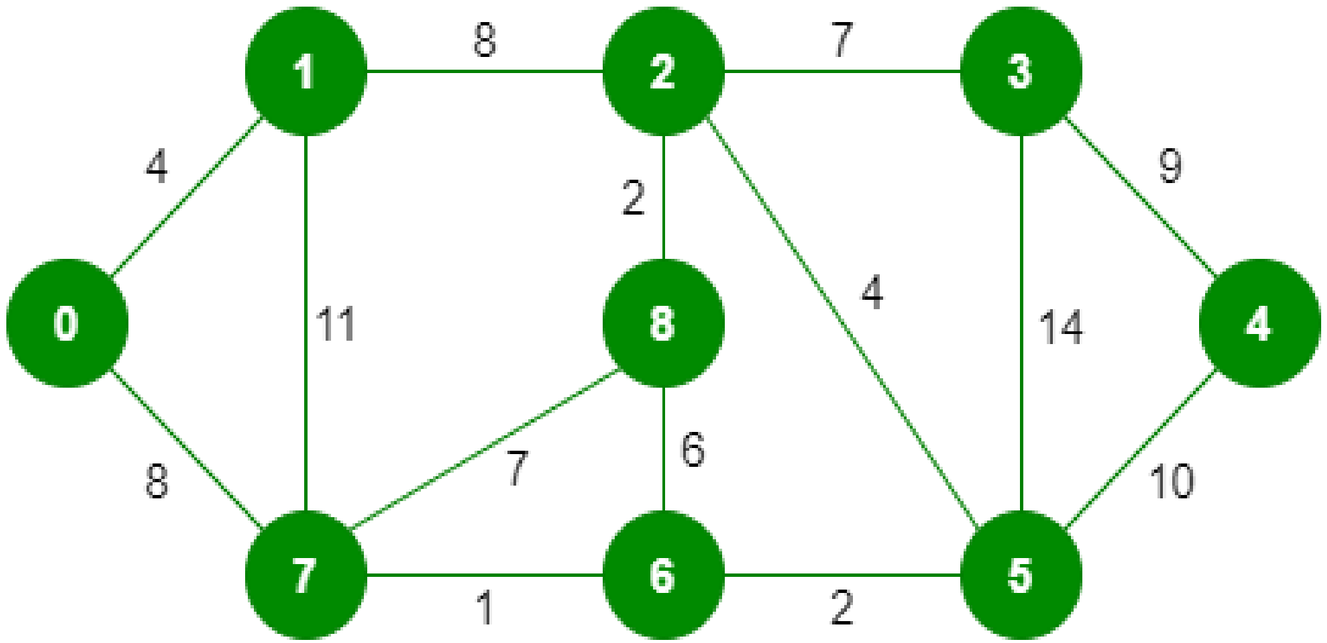


Hence, the final MST is the one which is shown above.

the cost of MST = 1 + 2 + 3 + 4 = 10.

~Dr Gaurav Kumar, Asst. Prof, CEA, GLAU

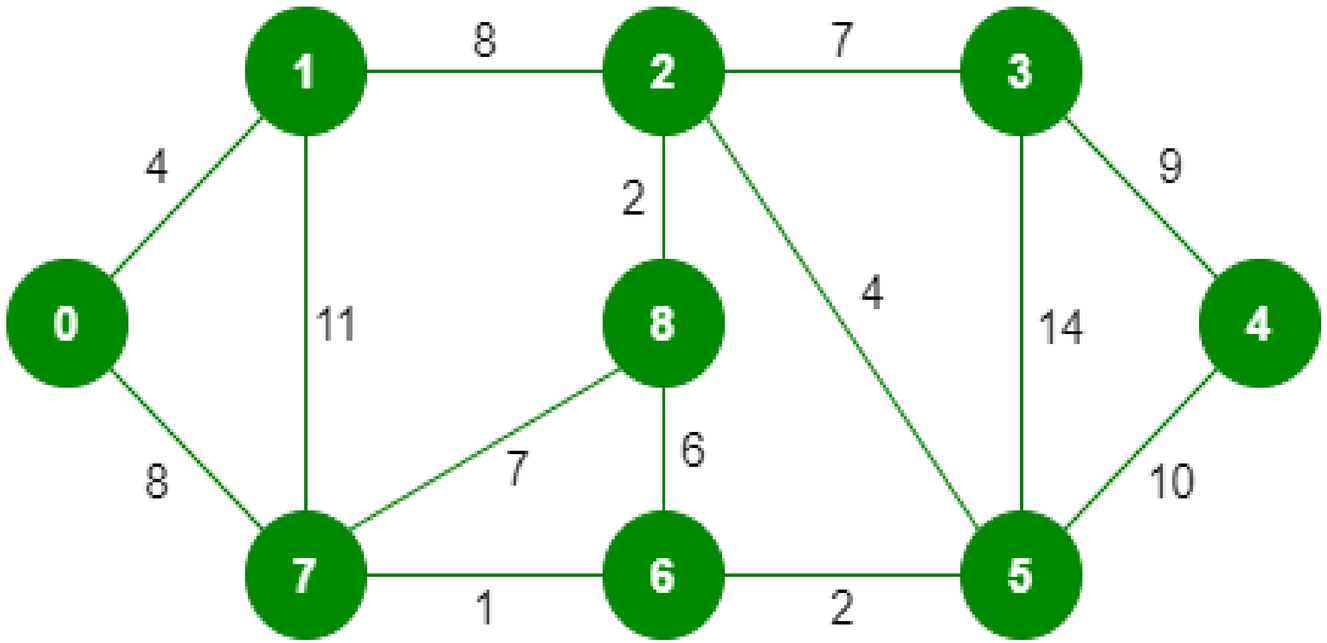
Practice Example of Kruskal's Algorithm



The graph contains 9 vertices and 14 edges. So, the minimum spanning tree formed will be having (9 – 1) = 8 edges.

~Dr Gaurav Kumar, Asst. Prof, CEA, GLAU

Practice Example of Kruskal's Algorithm



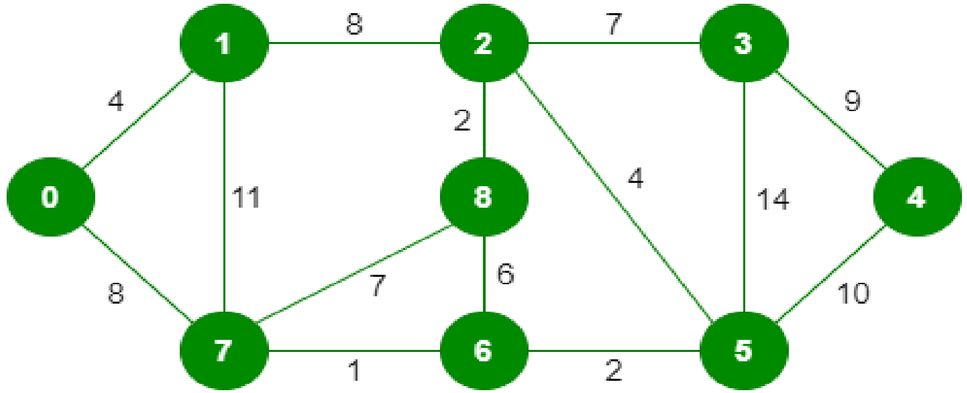
After sorting

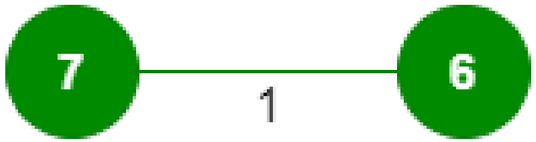
Edge	7-6	2-8	6-5	0-1	2-5	8-6	2-3	7-8	0-7	1-2	3-4	4-5	1-7	3-5
Weight	1	2	2	4	4	6	7	7	8	8	9	10	11	14

9 vertices and 14 edges

~Dr Gaurav Kumar, Asst. Prof, CEA, GLAU

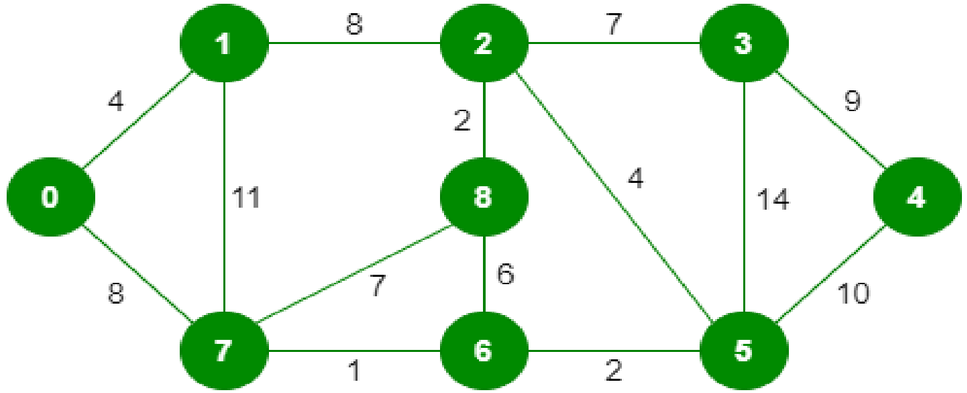
Practice Example of Kruskal's Algorithm

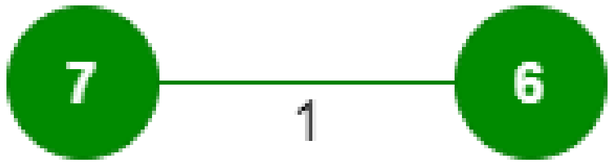




Edge	7-6	2-8	6-5	0-1	2-5	8-6	2-3	7-8	0-7	1-2	3-4	4-5	1-7	3-5
Weight	1	2	2	4	4	6	7	7	8	8	9	10	11	14

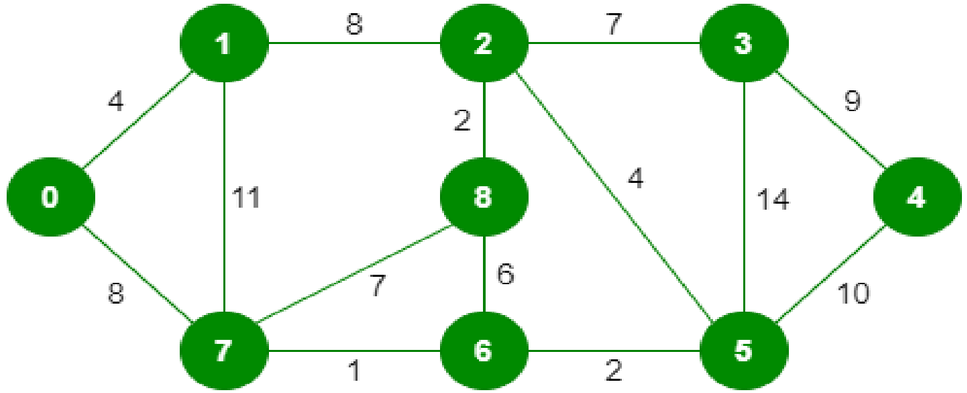
Practice Example of Kruskal's Algorithm

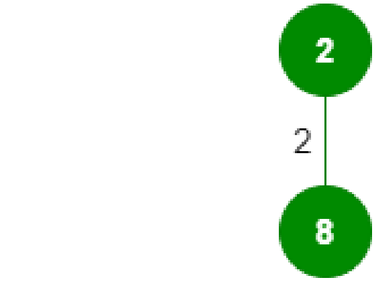


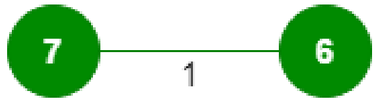


Edge	7-6	2-8	6-5	0-1	2-5	8-6	2-3	7-8	0-7	1-2	3-4	4-5	1-7	3-5
Weight	1	2	2	4	4	6	7	7	8	8	9	10	11	14

Practice Example of Kruskal's Algorithm

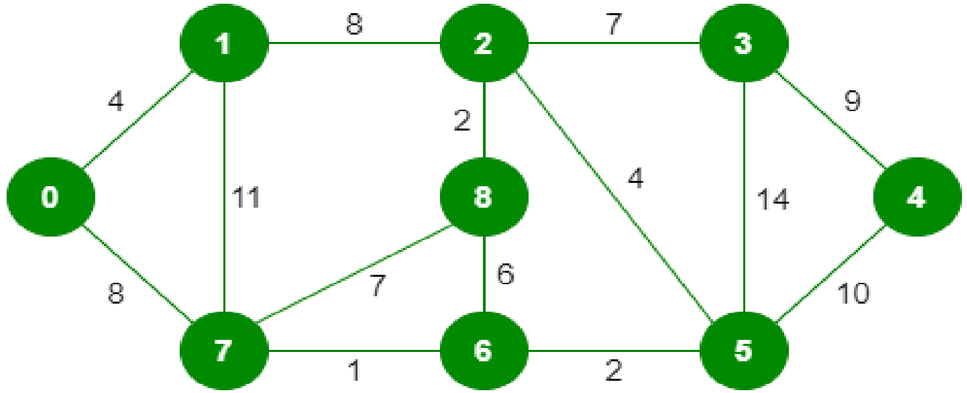


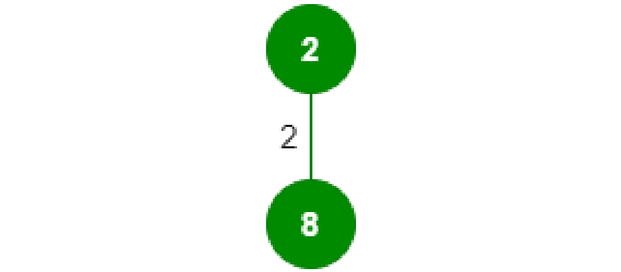




Edge	7-6	2-8	6-5	0-1	2-5	8-6	2-3	7-8	0-7	1-2	3-4	4-5	1-7	3-5
Weight	1	2	2	4	4	6	7	7	8	8	9	10	11	14

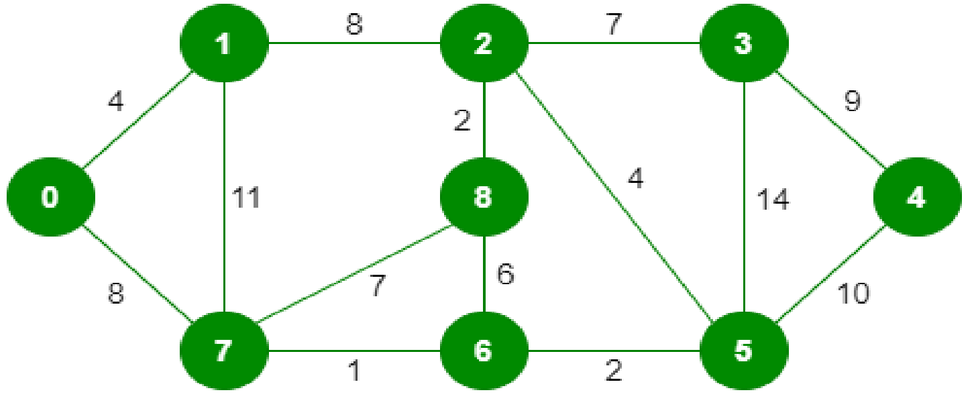
Practice Example of Kruskal's Algorithm

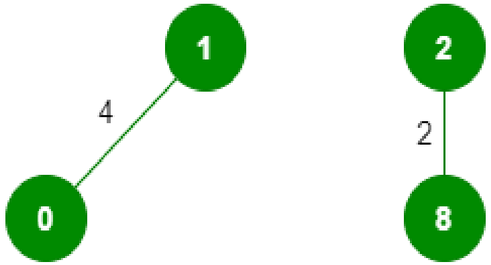




Edge	7-6	2-8	6-5	0-1	2-5	8-6	2-3	7-8	0-7	1-2	3-4	4-5	1-7	3-5
Weight	1	2	2	4	4	6	7	7	8	8	9	10	11	14

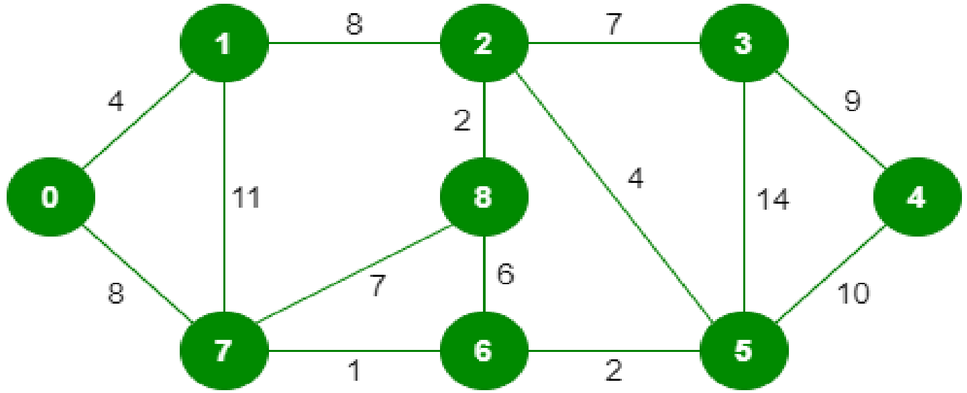
Practice Example of Kruskal's Algorithm

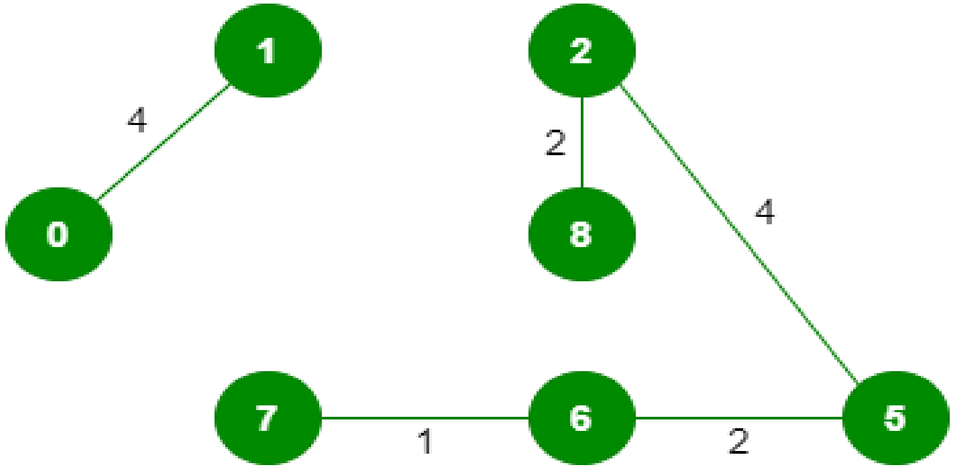




Edge	7-6	2-8	6-5	0-1	2-5	8-6	2-3	7-8	0-7	1-2	3-4	4-5	1-7	3-5
Weight	1	2	2	4	4	6	7	7	8	8	9	10	11	14

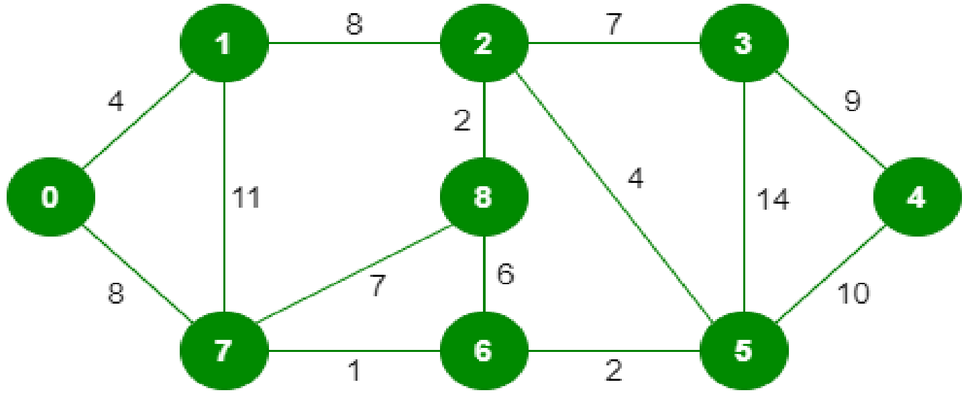
Practice Example of Kruskal's Algorithm

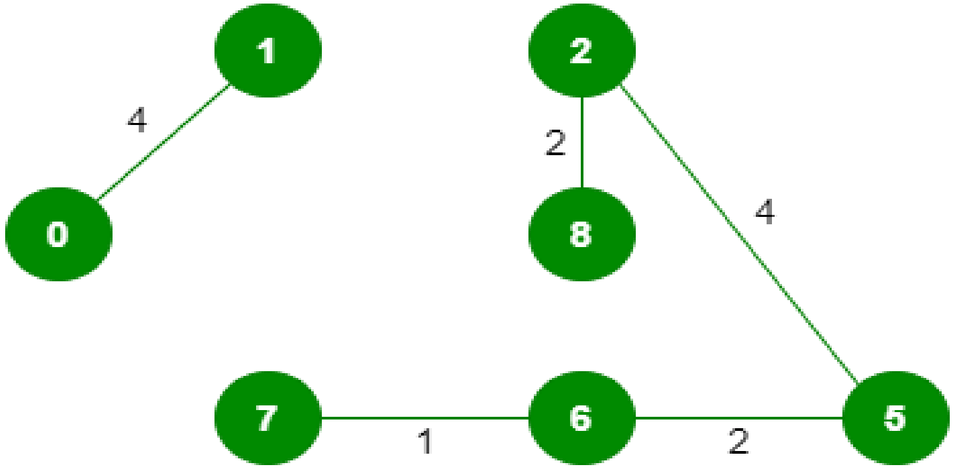




Edge	7-6	2-8	6-5	0-1	2-5	8-6	2-3	7-8	0-7	1-2	3-4	4-5	1-7	3-5
Weight	1	2	2	4	4	6	7	7	8	8	9	10	11	14

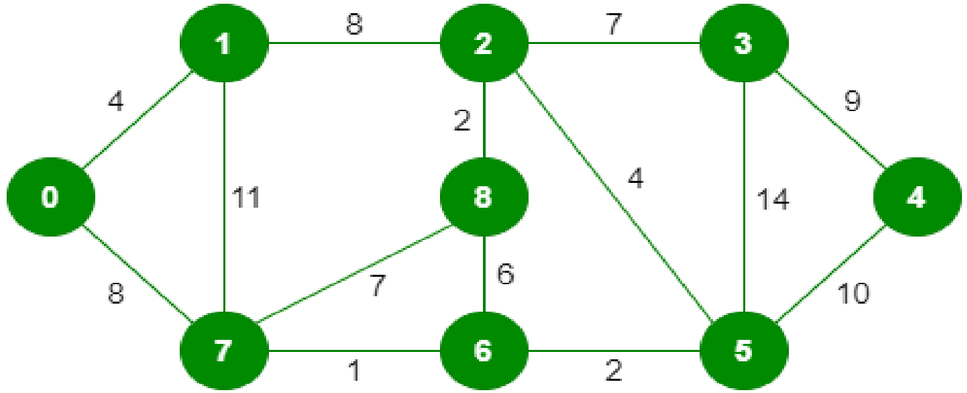
Practice Example of Kruskal's Algorithm

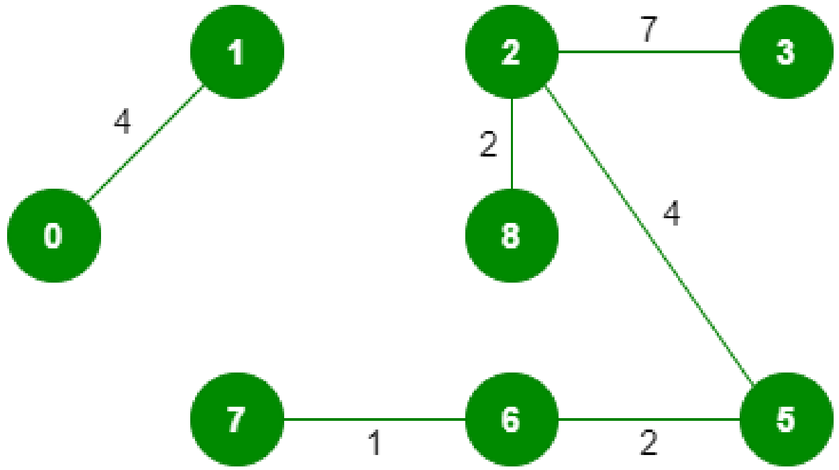




Edge	7-6	2-8	6-5	0-1	2-5	8-6	2-3	7-8	0-7	1-2	3-4	4-5	1-7	3-5
Weight	1	2	2	4	4	6	7	7	8	8	9	10	11	14

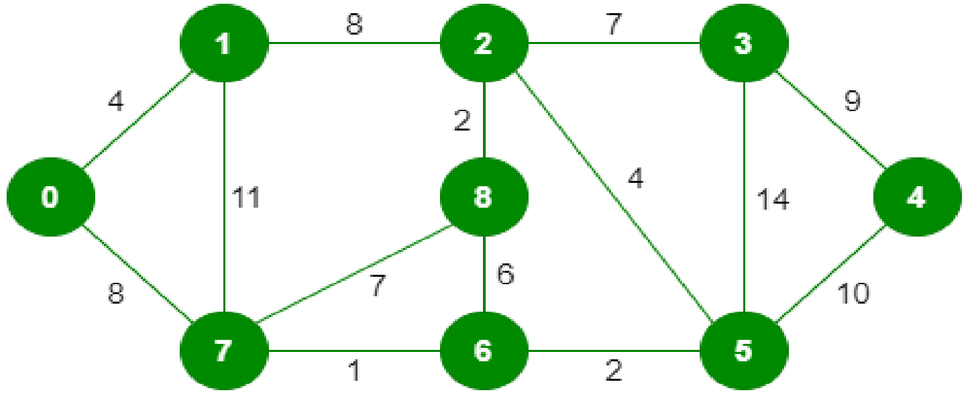
Practice Example of Kruskal's Algorithm

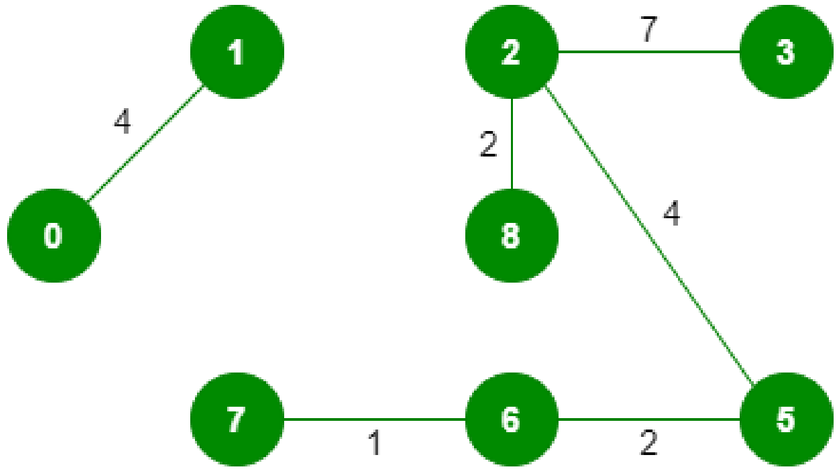




Edge	7-6	2-8	6-5	0-1	2-5	8-6	2-3	7-8	0-7	1-2	3-4	4-5	1-7	3-5
Weight	1	2	2	4	4	6	7	7	8	8	9	10	11	14

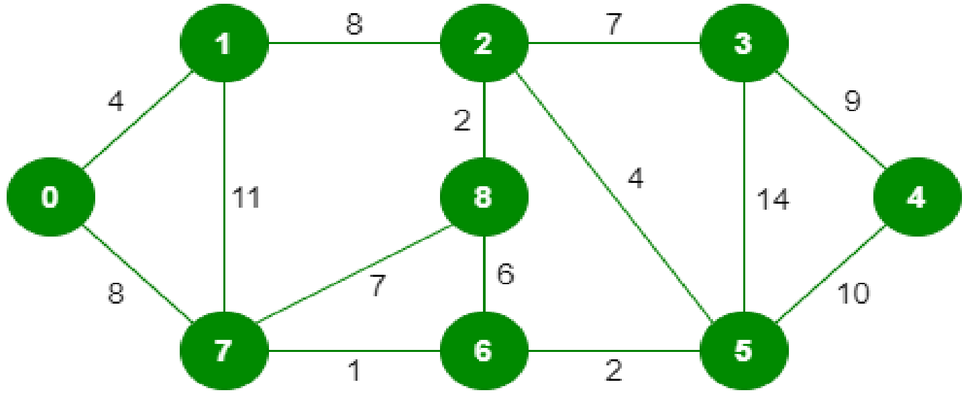
Practice Example of Kruskal's Algorithm

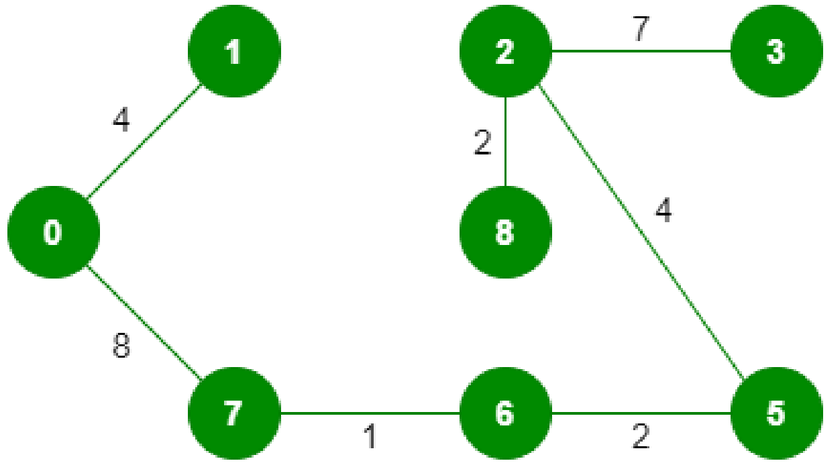




Edge	7-6	2-8	6-5	0-1	2-5	8-6	2-3	7-8	0-7	1-2	3-4	4-5	1-7	3-5
Weight	1	2	2	4	4	6	7	7	8	8	9	10	11	14

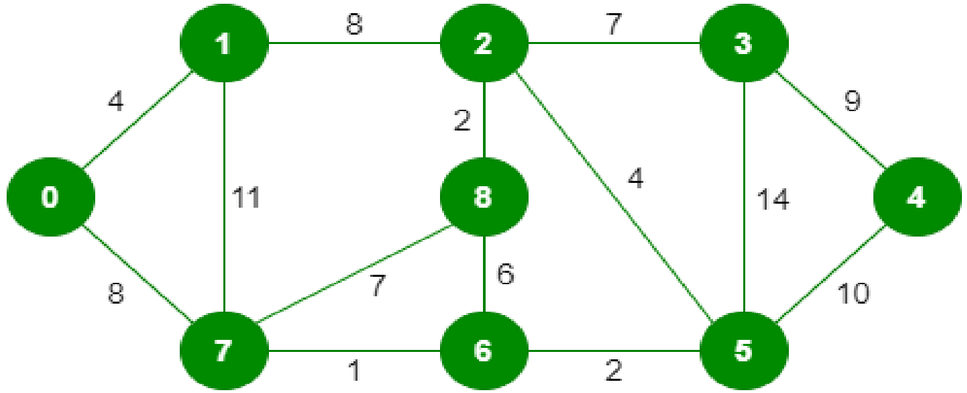
Practice Example of Kruskal's Algorithm

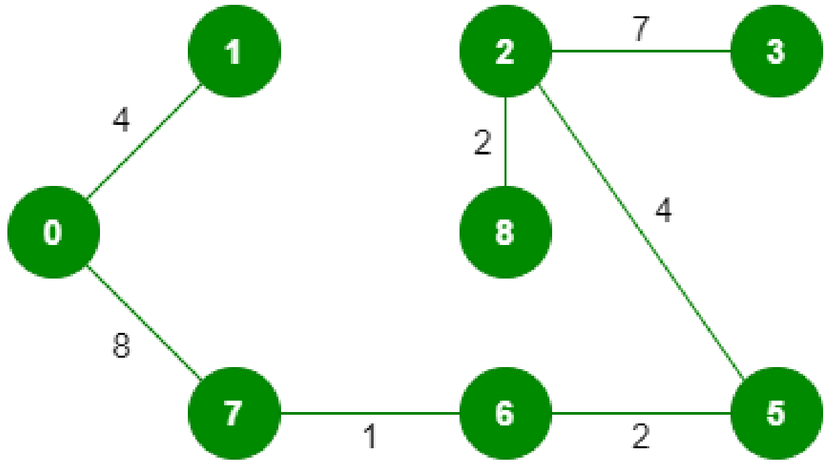




Edge	7-6	2-8	6-5	0-1	2-5	8-6	2-3	7-8	0-7	1-2	3-4	4-5	1-7	3-5
Weight	1	2	2	4	4	6	7	7	8	8	9	10	11	14

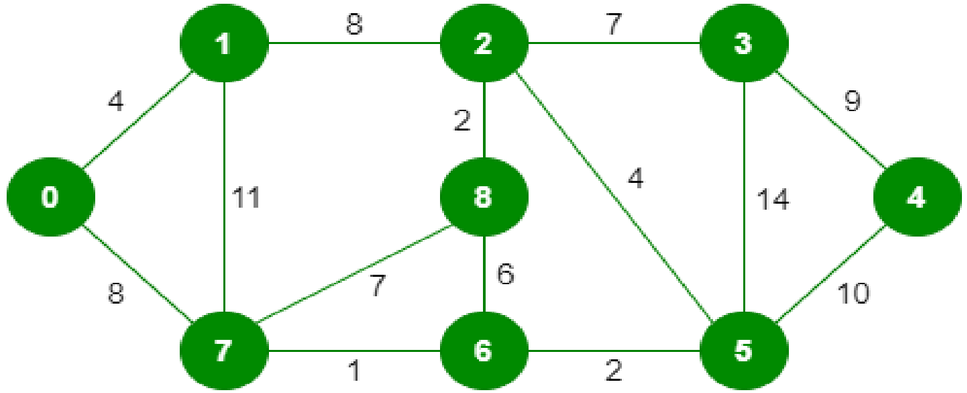
Practice Example of Kruskal's Algorithm

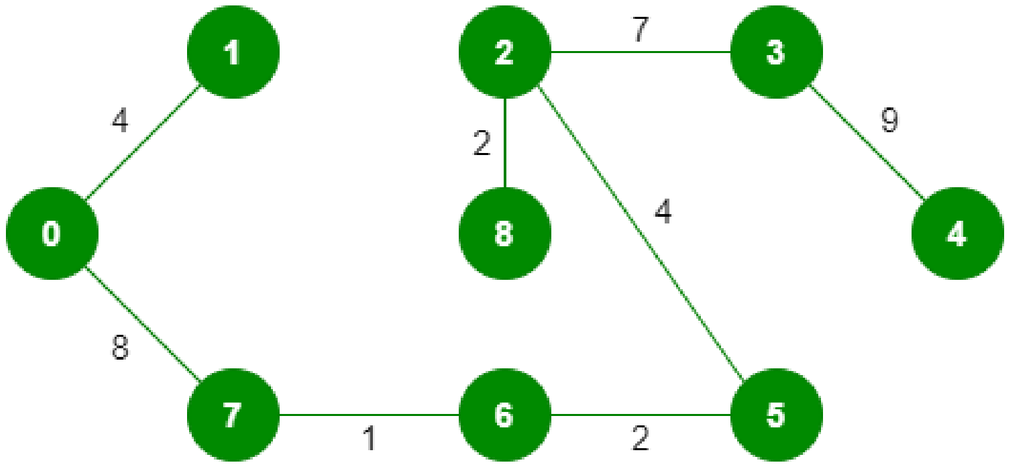




Edge	7-6	2-8	6-5	0-1	2-5	8-6	2-3	7-8	0-7	1-2	3-4	4-5	1-7	3-5
Weight	1	2	2	4	4	6	7	7	8	8	9	10	11	14

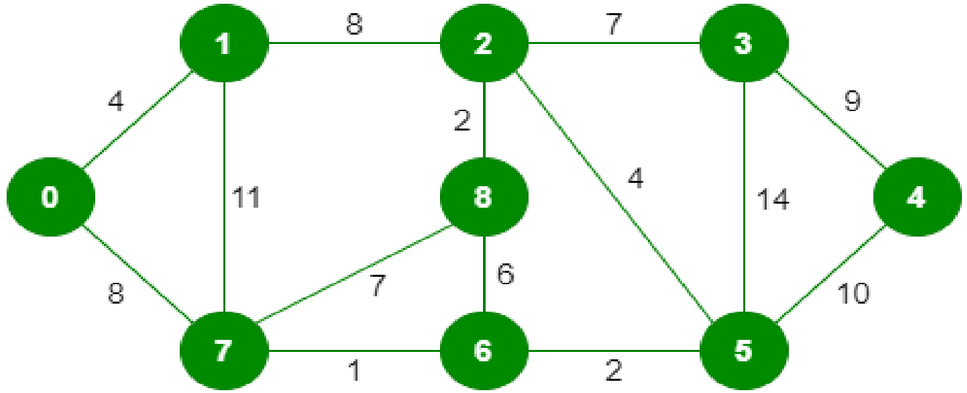
Practice Example of Kruskal's Algorithm

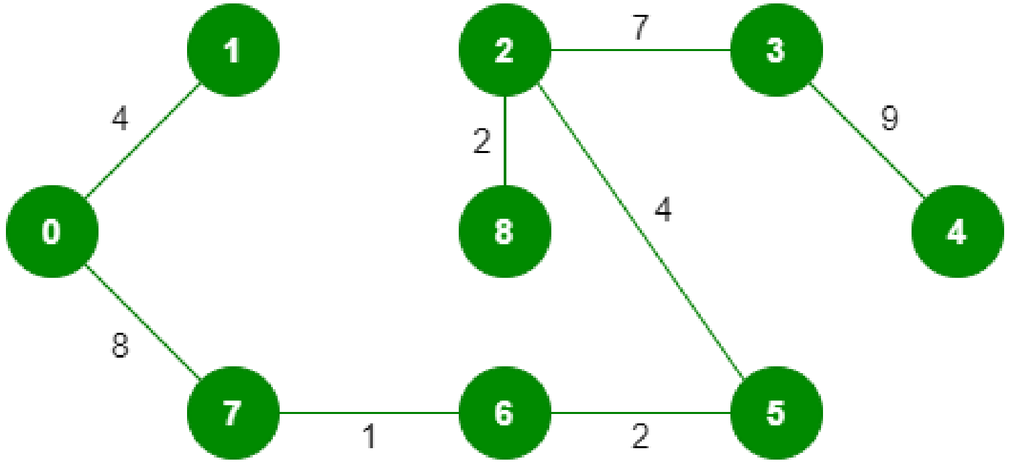




Edge	7-6	2-8	6-5	0-1	2-5	8-6	2-3	7-8	0-7	1-2	3-4	4-5	1-7	3-5
Weight	1	2	2	4	4	6	7	7	8	8	9	10	11	14

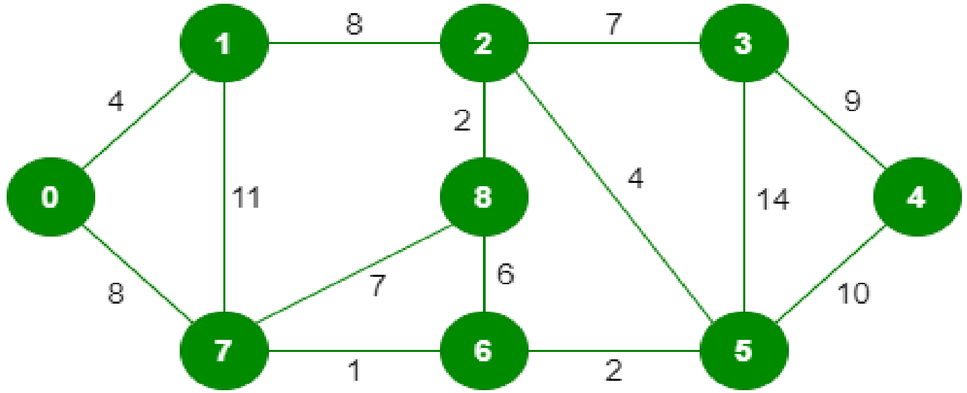
Practice Example of Kruskal's Algorithm

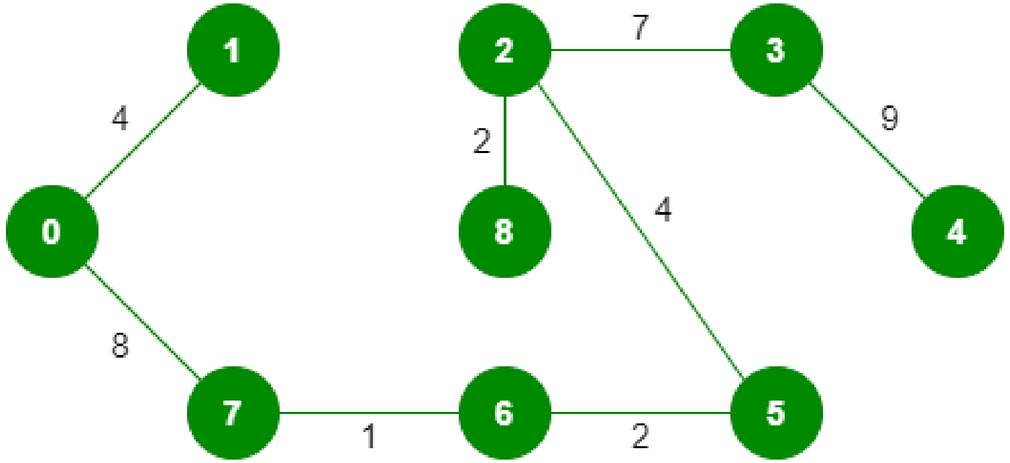




Edge	7-6	2-8	6-5	0-1	2-5	8-6	2-3	7-8	0-7	1-2	3-4	4-5	1-7	3-5
Weight	1	2	2	4	4	6	7	7	8	8	9	10	11	14

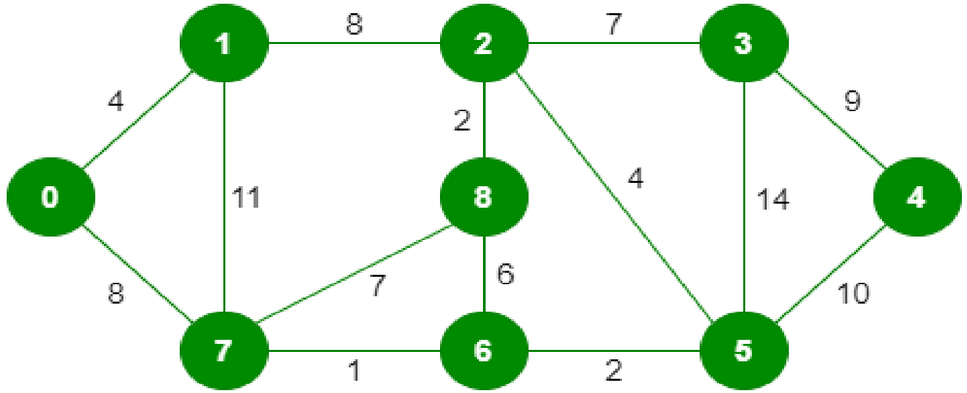
Practice Example of Kruskal's Algorithm

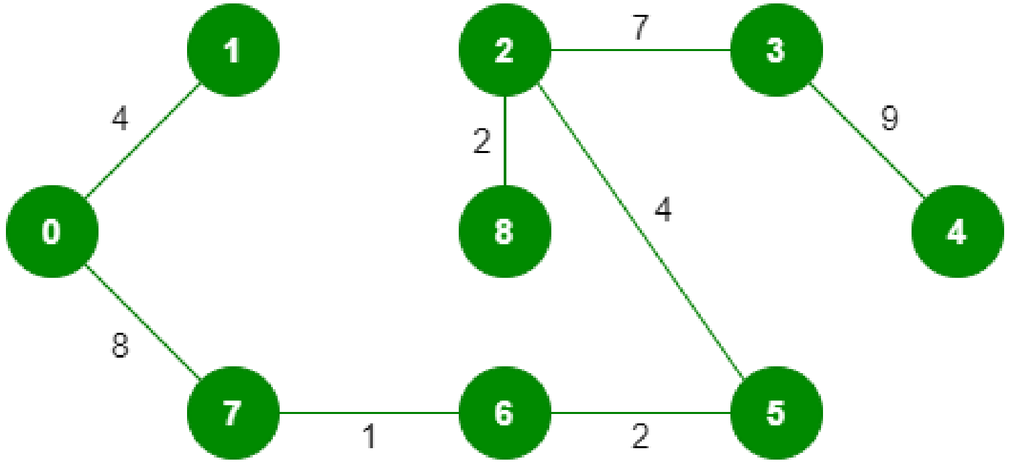




Edge	7-6	2-8	6-5	0-1	2-5	8-6	2-3	7-8	0-7	1-2	3-4	4-5	1-7	3-5
Weight	1	2	2	4	4	6	7	7	8	8	9	10	11	14

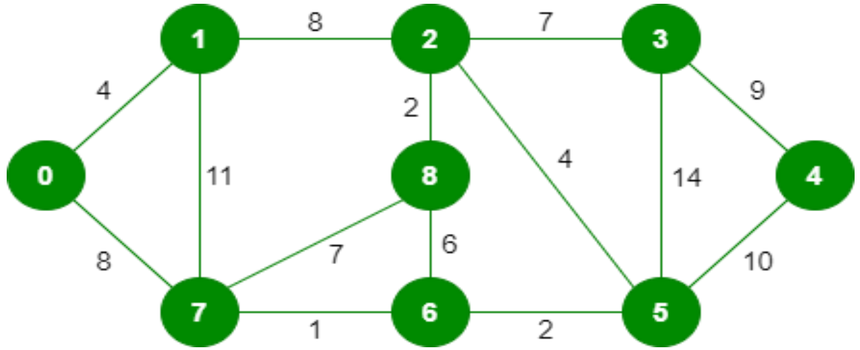
Practice Example of Kruskal's Algorithm

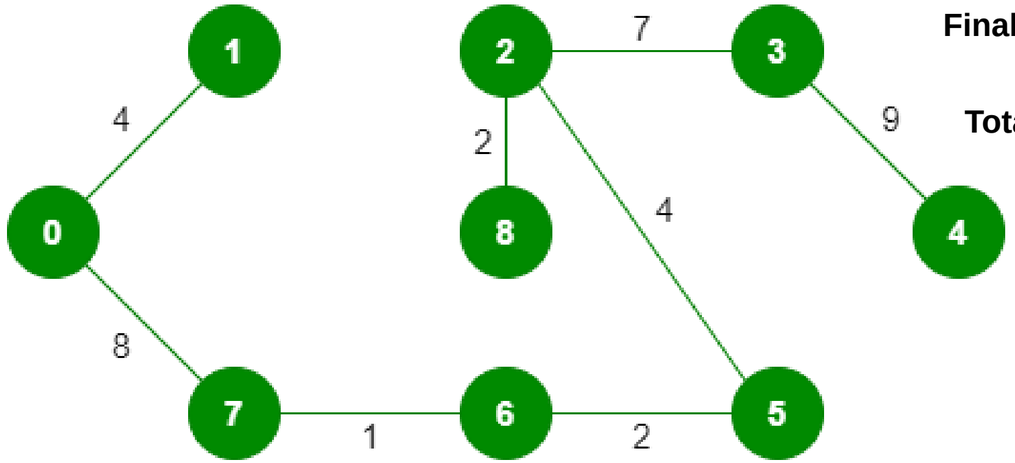




Edge	7-6	2-8	6-5	0-1	2-5	8-6	2-3	7-8	0-7	1-2	3-4	4-5	1-7	3-5
Weight	1	2	2	4	4	6	7	7	8	8	9	10	11	14

Practice Example of Kruskal's Algorithm



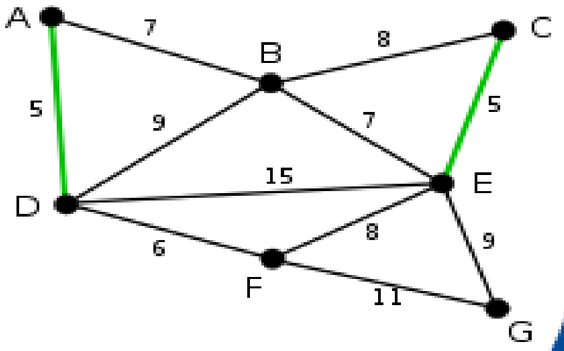


Final Minimum Cost Spanning Tree

Total Cost- 37

Edge	7-6	2-8	6-5	0-1	2-5	8-6	2-3	7-8	0-7	1-2	3-4	4-5	1-7	3-5
Weight	1	2	2	4	4	6	7	7	8	8	9	10	11	14

Kruskal's Algorithm



1. Sort all the edges from low weight to high → O(ElogE)

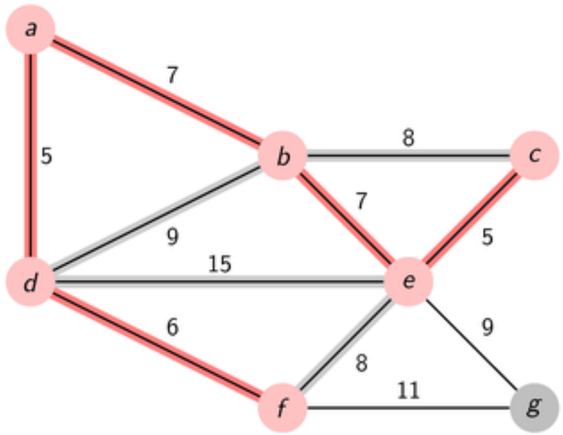
2. Take the edge with the lowest weight and add it to the spanning tree. If adding the edge create a cycle, then reject this edge. → logV

3. Keep adding edges until we reach all vertices. → O(ElogV)

Or *Repeat step#2 until there are (V-1) edges in the spanning tree.*

Total Time= ElogE + logV + ElogV

The time complexity of Kruskal's Algorithm is: O(E log E).

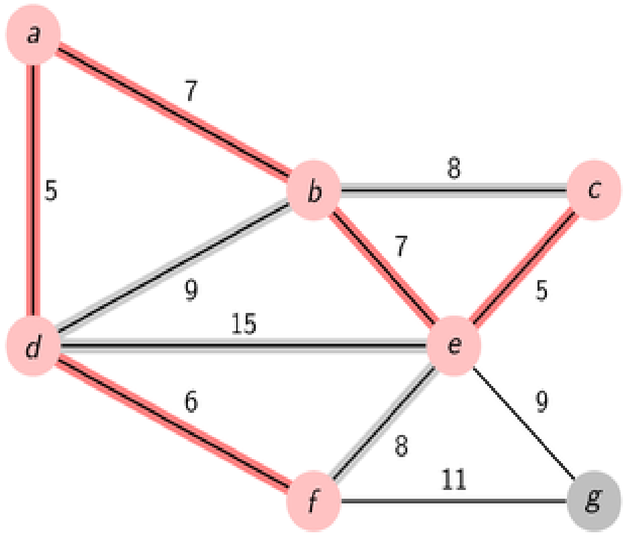


04.b

Prim's Algorithm

~Dr Gaurav Kumar, Asst. Prof, CEA, GLAU

What is Prim’s Method



- Prim's Algorithm is used to find the minimum spanning tree from a graph.

- Prim's algorithm finds the subset of edges that includes every vertex of the graph such that the sum of the weights of the edges can be minimized.

- Prim's algorithm starts with the single node and explore all the adjacent nodes with all the connecting edges at every step. The edges with the minimal weights causing no cycles in the graph got selected.

Prim's Algorithm

Step 1: Select a starting vertex

Step 2: Repeat Steps 3 until all vertices are connected without forming any cycle

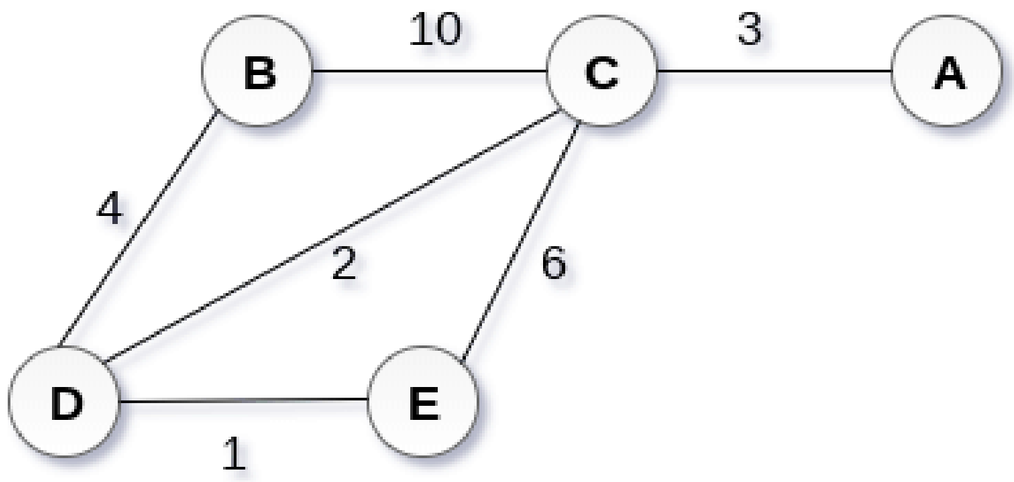
Step 3: Select an edge **e** connecting to the adjacent vertex that has the minimum weight among all adjacent vertices of visited nodes, if it is not forming a cycle then add it otherwise reject it.

Step 4: EXIT

~Dr Gaurav Kumar, Asst. Prof, CEA, GLAU

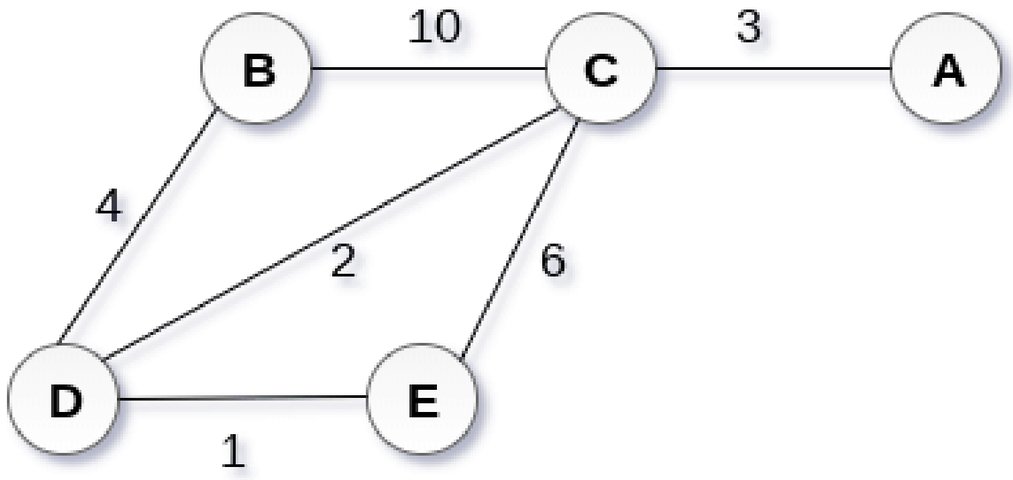
Practice Example of Prim's Algorithm

Construct a minimum spanning tree of the graph given in the following figure by using prim's algorithm.



~Dr Gaurav Kumar, Asst. Prof, CEA, GLAU

Practice Example of Prim's Algorithm

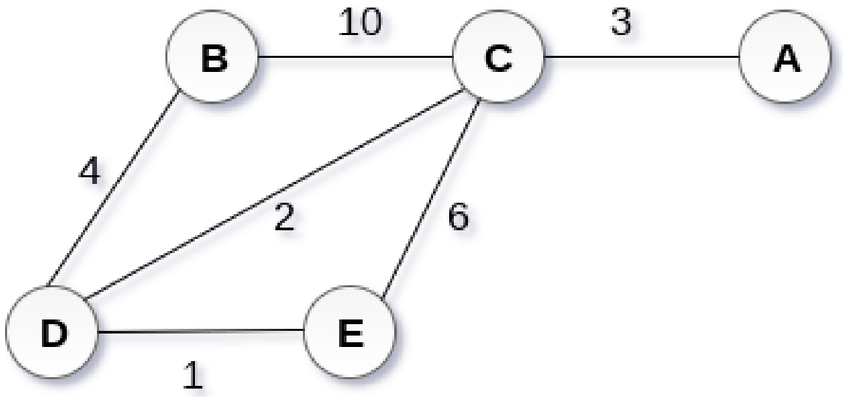


Step 1 : Choose a starting vertex B.

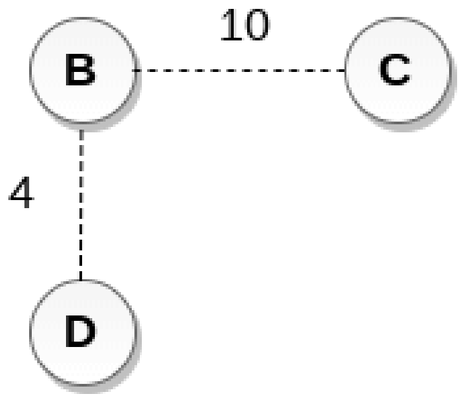
Step 1

~Dr Gaurav Kumar, Asst. Prof, CEA, GLAU

Practice Example of Prim's Algorithm



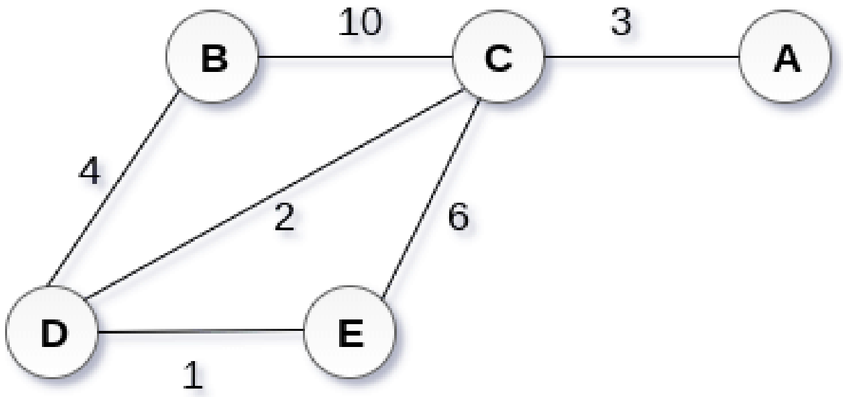
Step 2: Add the vertices that are adjacent to B. the edges that connecting the vertices are shown by dotted lines.



Step 2

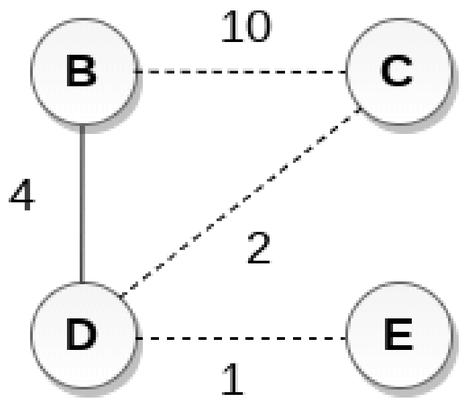
~Dr Gaurav Kumar, Asst. Prof, CEA, GLAU

Practice Example of Prim's Algorithm



Step 3: Choose the edge with the minimum weight among all.

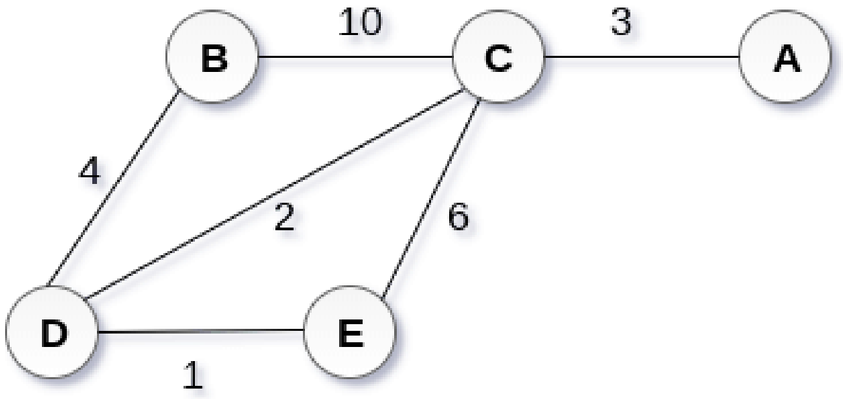
i.e. BD and add it to MST. Add the adjacent vertices of D i.e. C and E.



Step 3

~Dr Gaurav Kumar, Asst. Prof, CEA, GLAU

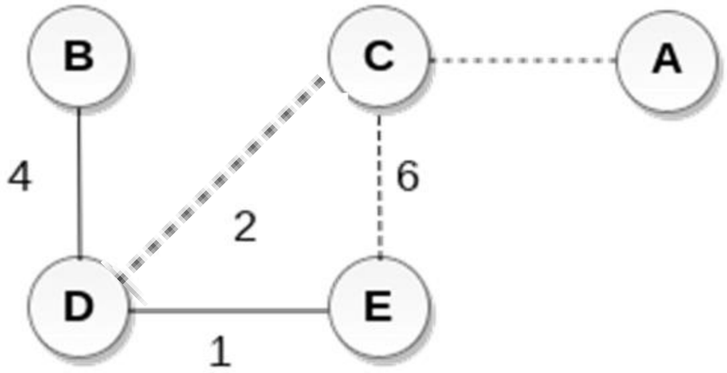
Practice Example of Prim's Algorithm



Step 5: Choose the edge with the minimum weight among all.

In this case, the edges DC is the minimum weighted edge.

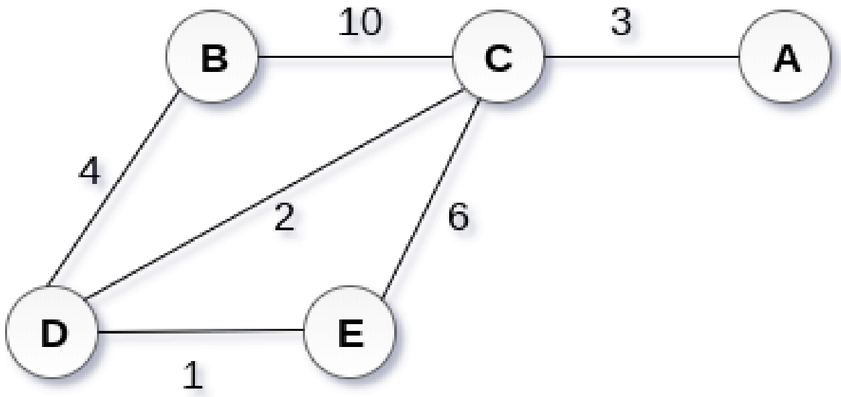
Add DC to MST and add the adjacent of C i.e. A



Step 5

~Dr Gaurav Kumar, Asst. Prof, CEA, GLAU

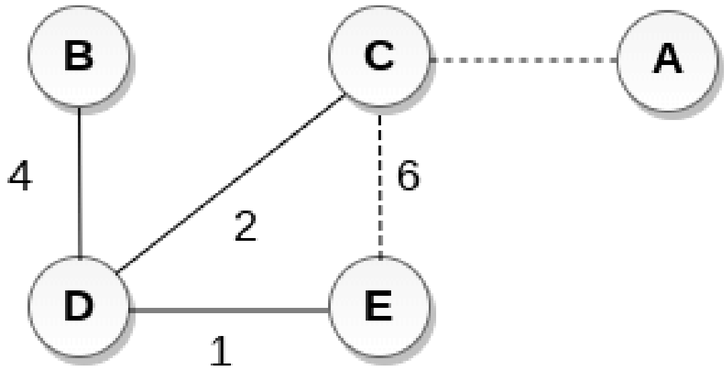
Practice Example of Prim's Algorithm



Step 4: Choose the edge with the minimum weight among all.

In this case, the edges DE is the minimum weighted edge.

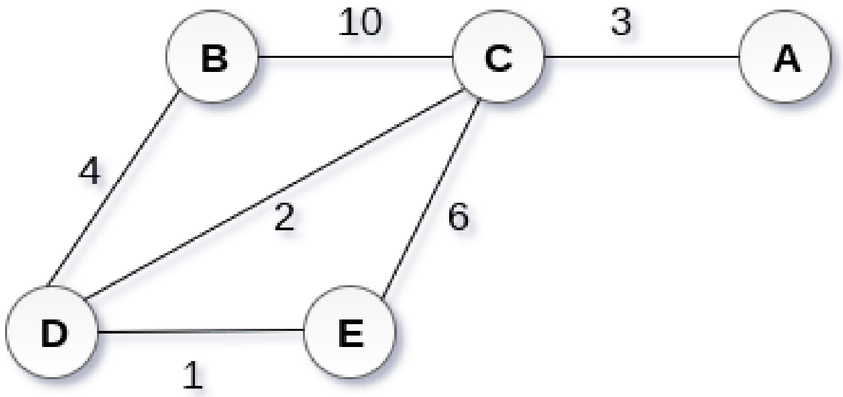
Add DE to MST and explore the adjacent of E i.e. C



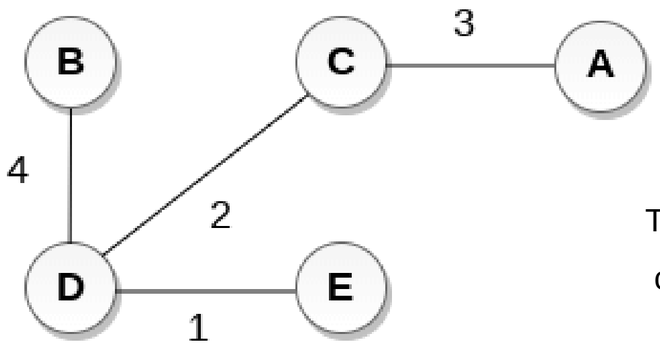
Step 4

~Dr Gaurav Kumar, Asst. Prof, CEA, GLAU

Practice Example of Prim's Algorithm



Step 6: Choose the edge with the minimum weight among all i.e. CA. We can't choose CE as it would cause cycle in the graph.



Step 6

The cost of MST will be calculated as;

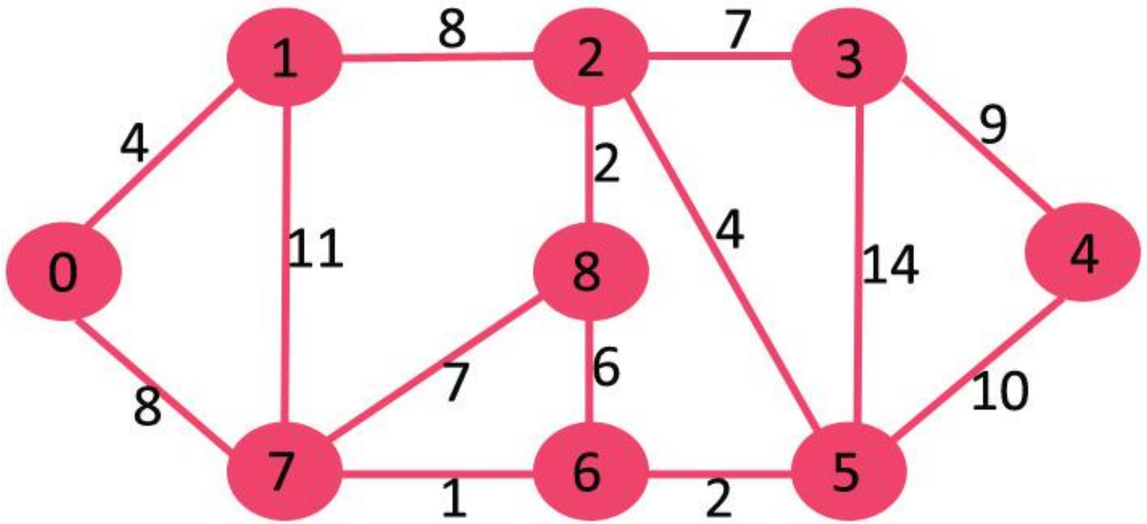
cost(MST) = 4 + 2 + 1 + 3 = 10 units.

The graph produces in the step 6 is the minimum spanning tree of the graph.

~Dr Gaurav Kumar, Asst. Prof, CEA, GLAU

Practice Example of Prim's Algorithm

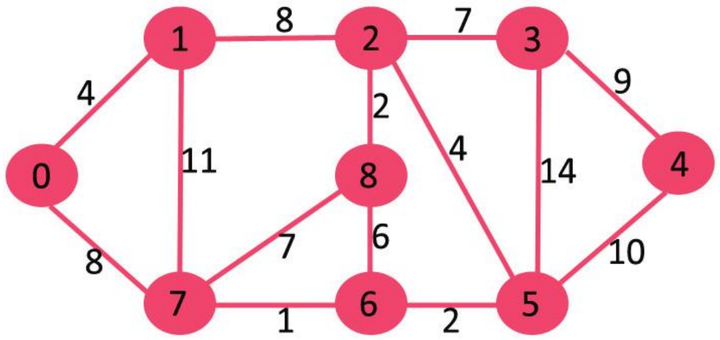
Construct a minimum spanning tree of the graph given in the following figure by using prim's algorithm.



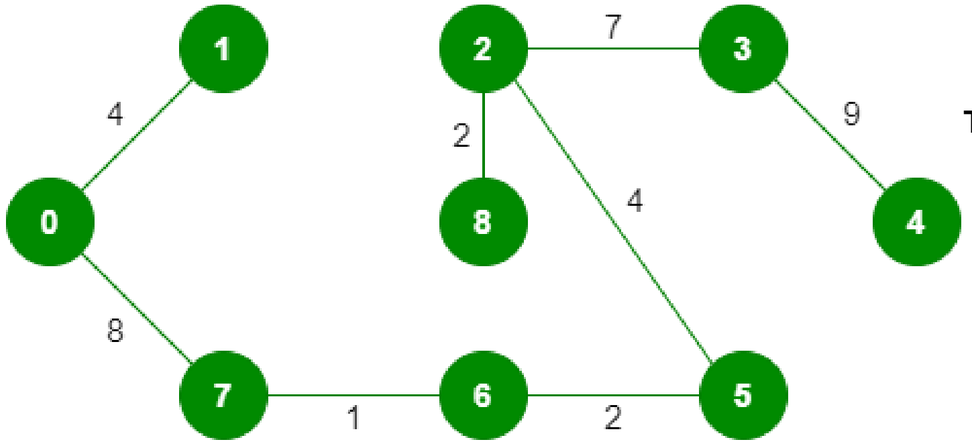
~Dr Gaurav Kumar, Asst. Prof, CEA, GLAU

Practice Example of Prim's Algorithm

Construct a minimum spanning tree of the graph given in the following figure by using prim's algorithm.



Final Minimum Cost Spanning Tree

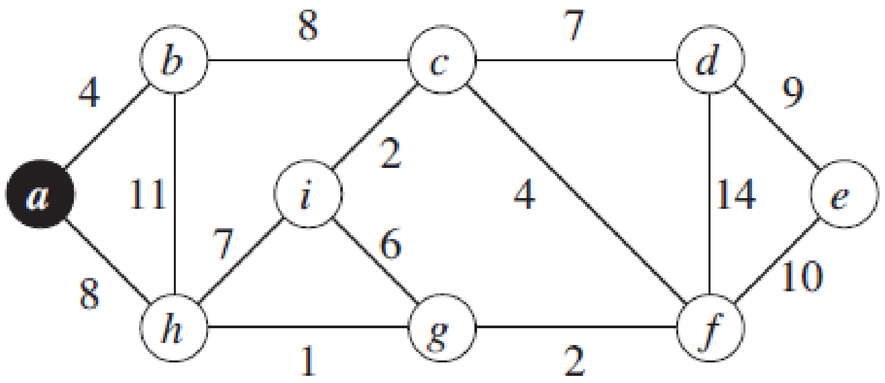


Total Cost- 37

~Dr Gaurav Kumar, Asst. Prof, CEA, GLAU

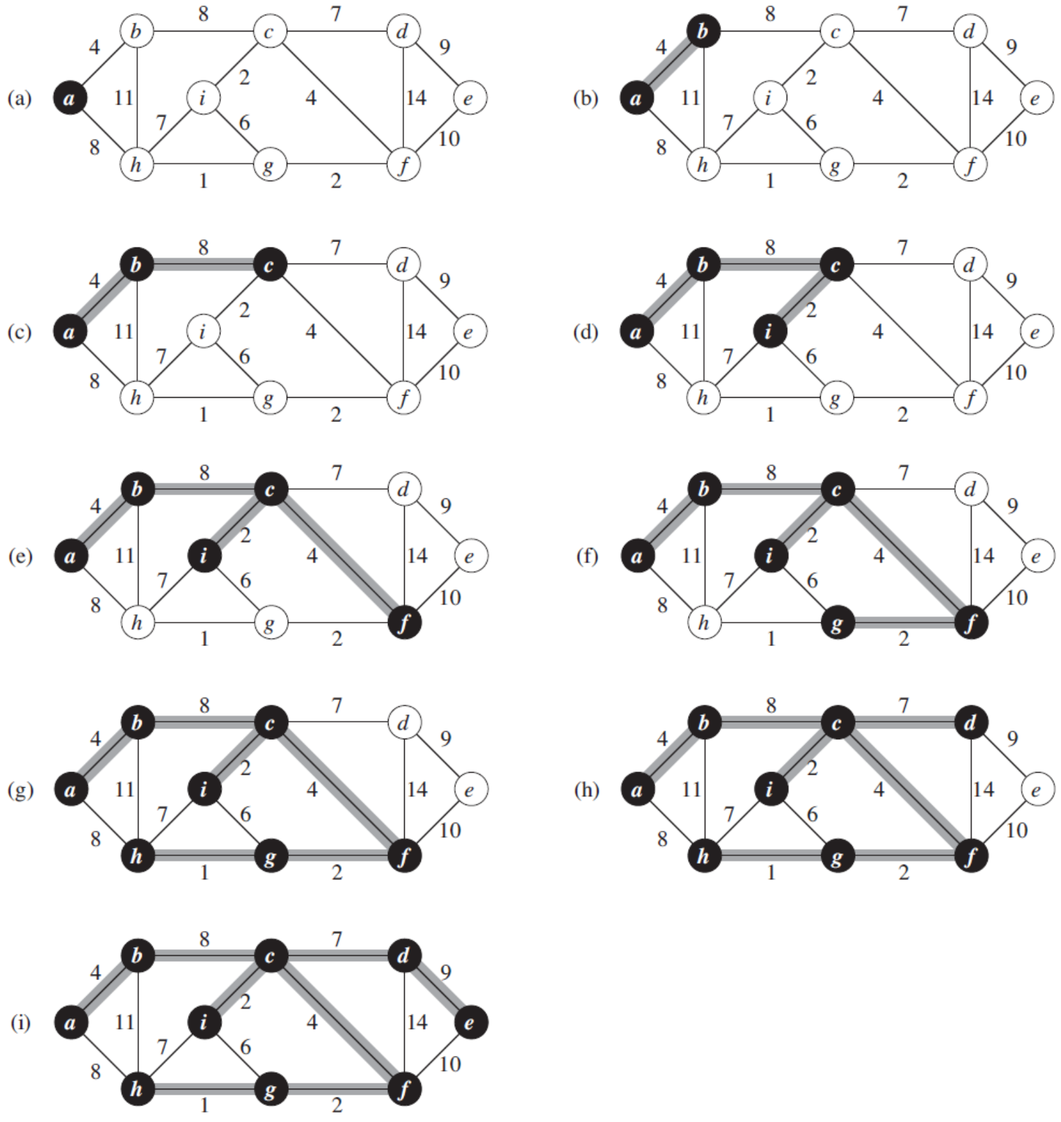
Self Practice Example of Prim's Algorithm

Construct a minimum spanning tree of the graph given in the following figure by using prim's algorithm.



~Dr Gaurav Kumar, Asst. Prof, CEA, GLAU

Self Practice Example of Prim's Algorithm



~Dr Gaurav Kumar, Asst. Prof, CEA, GLAU

Time Complexity of Prim's Algorithm

Step 1: Select a starting vertex

Step 2: Repeat Steps 3 until all vertices are connected without forming any cycle

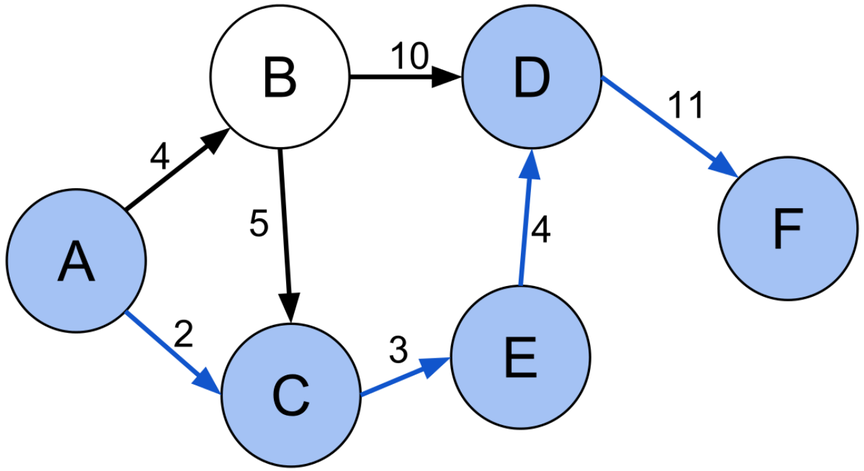
Step 3: Select an edge **e** connecting to the adjacent vertex that has the minimum weight among all adjacent vertices of visited nodes, if it is not forming a cycle then add it otherwise reject it.

Step 4: EXIT

The time complexity is = $O(V^2)$

Difference Between Prim's & Kruskal's Algorithm

Prim's Algorithm	Kruskal's Algorithm
It starts to build the Minimum Spanning Tree from any vertex in the graph.	It starts to build the Minimum Spanning Tree from the Edge carrying minimum weight in the graph.
Prim's algorithm gives connected component as well as it works only on connected graph.	Kruskal's algorithm can generate forest(disconnected components) at any instant as well as it can work on disconnected components
Prim's algorithm runs faster in dense graphs.	Kruskal's algorithm runs faster in sparse graphs.
Prim's algorithm uses List Data Structure.	Kruskal's algorithm uses Heap Data Structure.
Prim's algorithm has a time complexity of $O(V^2)$, V being the number of vertices and can be improved up to O(E log V) using Fibonacci heaps.	Kruskal's algorithm's time complexity is O(E log V), V being the number of vertices.



05

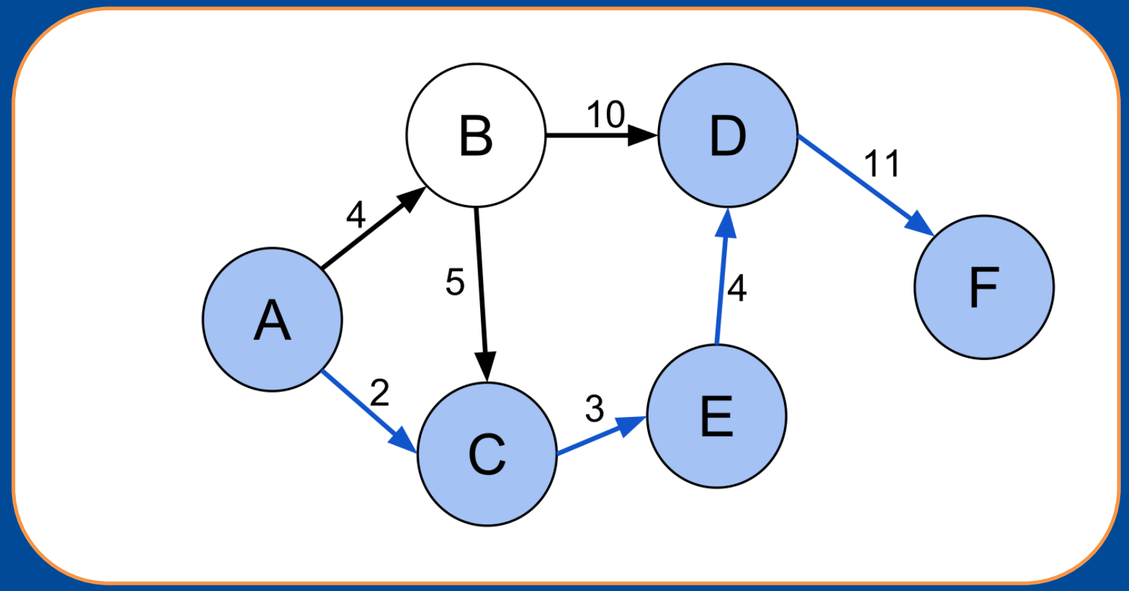
Single Source Shortest Path Problem

~Dr Gaurav Kumar, Asst. Prof, CEA, GLAU

Single Source Shortest Path Problem

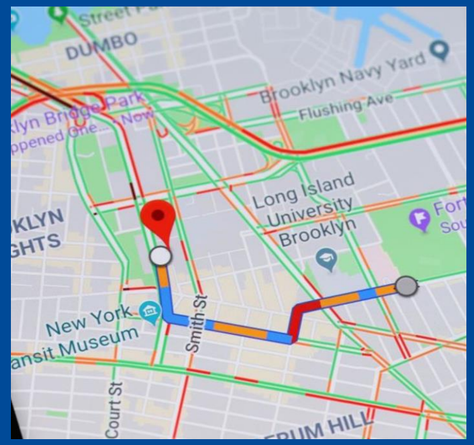
Find minimum distance from source vertex to any other vertex.

In a **Single Source Shortest Paths Problem**, we are given a **directed Graph** G = (V, E), we want to find the shortest path from a given source vertex s ∈ V to every vertex v ∈ V.



Application of Single Source Shortest Path Problem

Google Maps



Road Networks

Logistics Research

Single Source Shortest Path Problem

1. Dijkstra's Algorithm

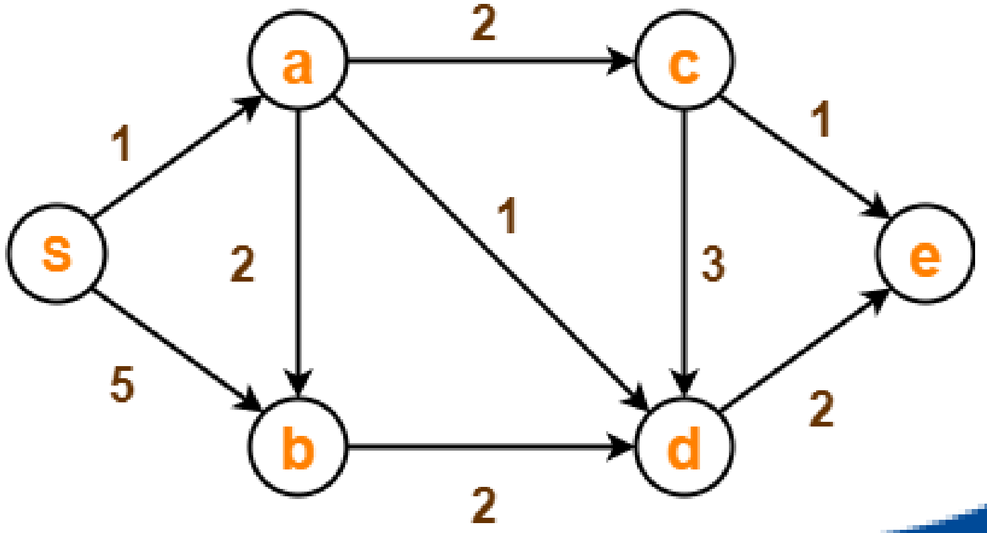
2. Bellman-Ford Algorithm

~Dr Gaurav Kumar, Asst. Prof, CEA, GLAU

1. Dijkstra's Algorithm

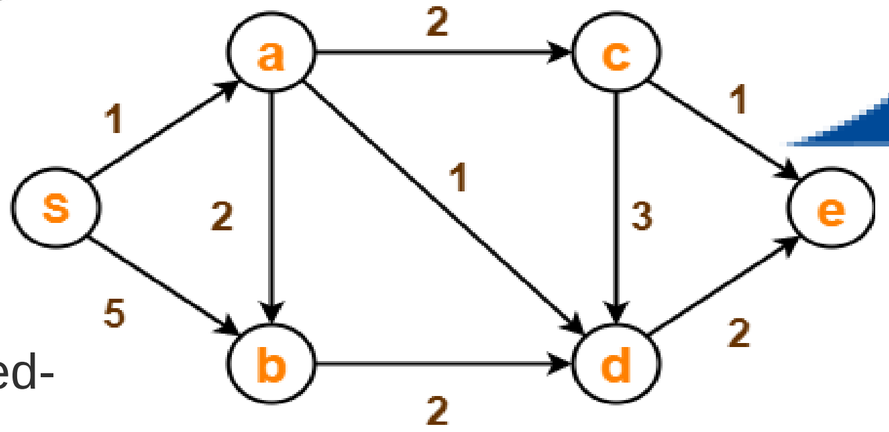
It is a **greedy algorithm** that solves the **single-source shortest path problem** for a **directed graph** G = (V, E) with **nonnegative edge weights**, i.e., w (u, v) ≥ 0 for each edge (u, v) ∈ E.

Problem: Using Dijkstra's Algorithm, find the shortest distance from source vertex 'S' to remaining vertices in the following graph. Also, write the order in which the vertices are visited.



~Dr Gaurav Kumar, Asst. Prof, CEA, GLAU

1. Dijkstra's Algorithm



Step-01:

In the first step, two sets are defined-

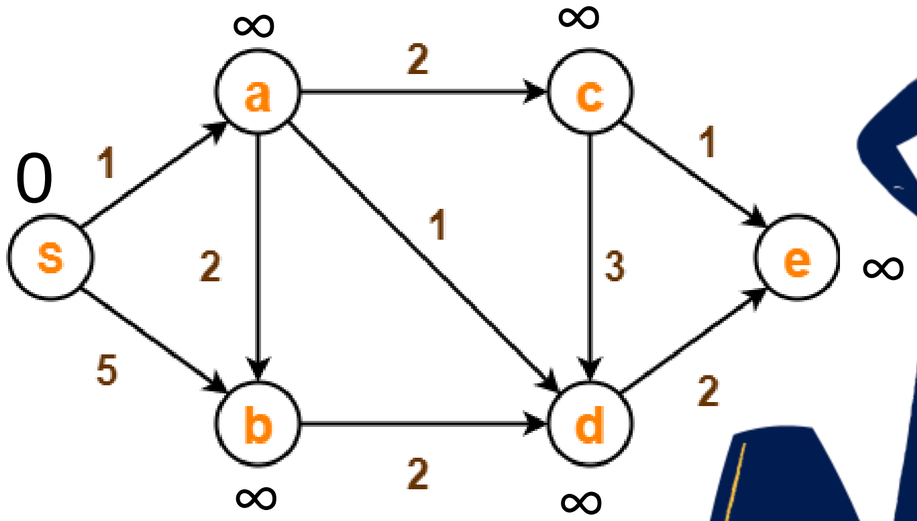
• One set contains all those vertices which have been included in the shortest path tree. (In the beginning, this set is empty).

• Other set contains all those vertices which are still left to be included in the shortest path tree. (In the beginning, this set contains all the vertices of the given graph)

1. Unvisited set : {S , a , b , c , d , e}
2. Visited set : { }

~Dr Gaurav Kumar, Asst. Prof, CEA, GLAU

1. Dijkstra's Algorithm



Step-02:

For each vertex of the given graph, two variables are defined as-

1. **Π[v]** which denotes the predecessor of vertex ‘v’

2. **d[v]** which denotes the shortest path estimate of vertex ‘v’ from the source vertex.

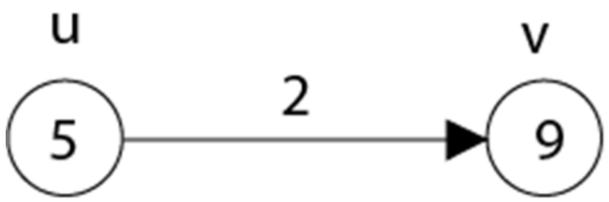
In this case,

- The value of variable ‘Π’ for each vertex is set to NIL i.e. Π[v] = NIL
- The value of variable ‘d’ for source vertex is set to 0 i.e. d[S] = 0
- The value of variable ‘d’ for remaining vertices is set to ∞ i.e. d[v] = ∞

~Dr Gaurav Kumar, Asst. Prof, CEA, GLAU

1. Dijkstra's Algorithm

Meaning of **π[v]** which denotes the predecessor of vertex ‘v’



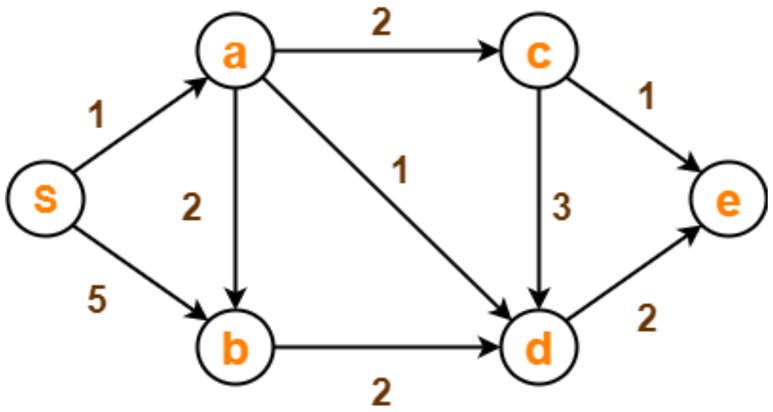
Π[v] = u

Meaning of **d[u], d[v]** which denotes the shortest path estimate of vertex u and v from the source vertex.

d[u] = 5 , d[v] = 9

~Dr Gaurav Kumar, Asst. Prof, CEA, GLAU

1. Dijkstra's Algorithm



Step-03:

The following procedure is repeated until all the vertices of the graph are processed-

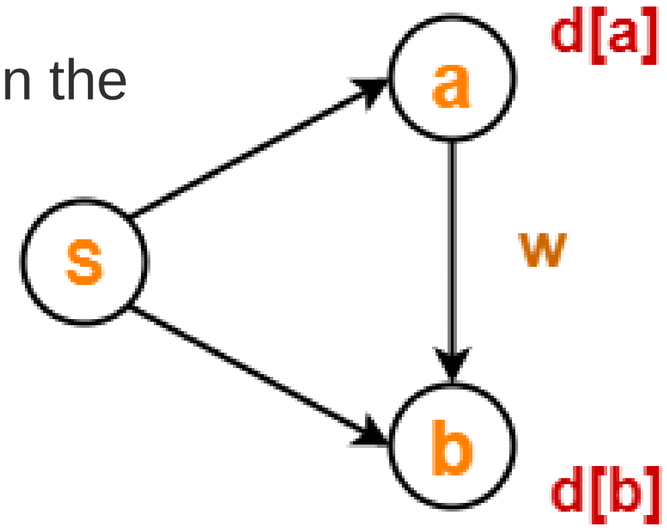
- Among unprocessed vertices, a vertex with minimum value of variable ‘d’ is chosen.
- Its outgoing edges are **relaxed**.
- After relaxing the edges for that vertex, the sets created in step-01 are updated.

~Dr Gaurav Kumar, Asst. Prof, CEA, GLAU

1. Dijkstra's Algorithm

What is Edge Relaxation?

Consider the edge (a , b) in the following graph-

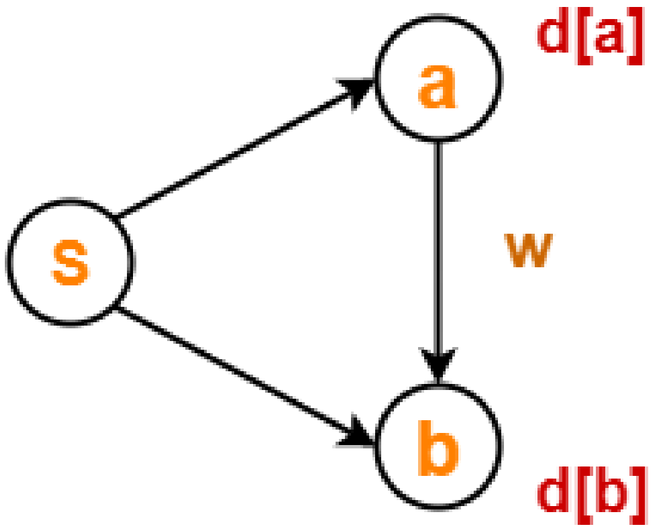


Here, d[a] and d[b] denotes the shortest path estimate for vertices a and b respectively from the source vertex ‘S’.

~Dr Gaurav Kumar, Asst. Prof, CEA, GLAU

1. Dijkstra's Algorithm

What is Edge Relaxation?



Now,

If d[a] + w < d[b]

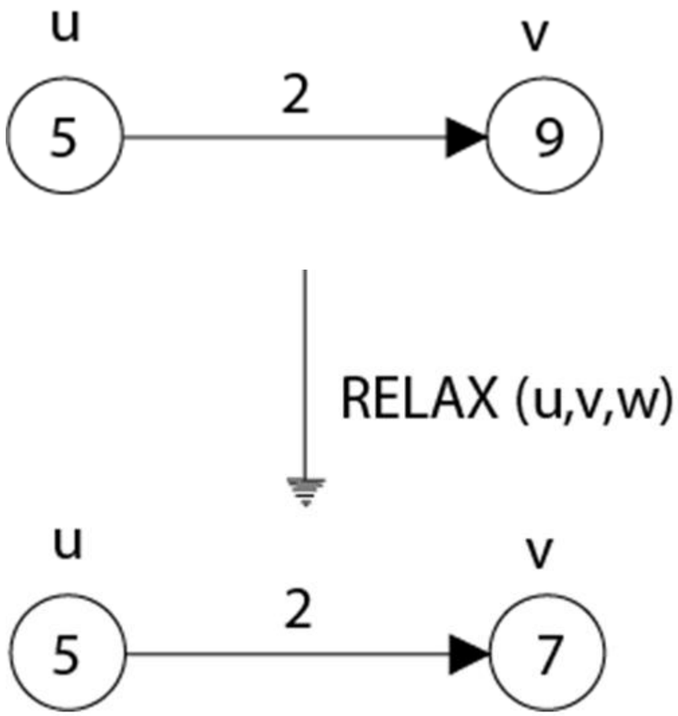
then d[b] = d[a] + w and Π[b] = a

This is called as edge relaxation.

~Dr Gaurav Kumar, Asst. Prof, CEA, GLAU

1. Dijkstra's Algorithm

Example of Edge Relaxation



If d[a] + w < d[b]

then d[b] = d[a] + w and Π[b] = a

This is called as edge relaxation.

If d[u] + w < d[v]

then d[v] = d[u] + w and Π[v] = u

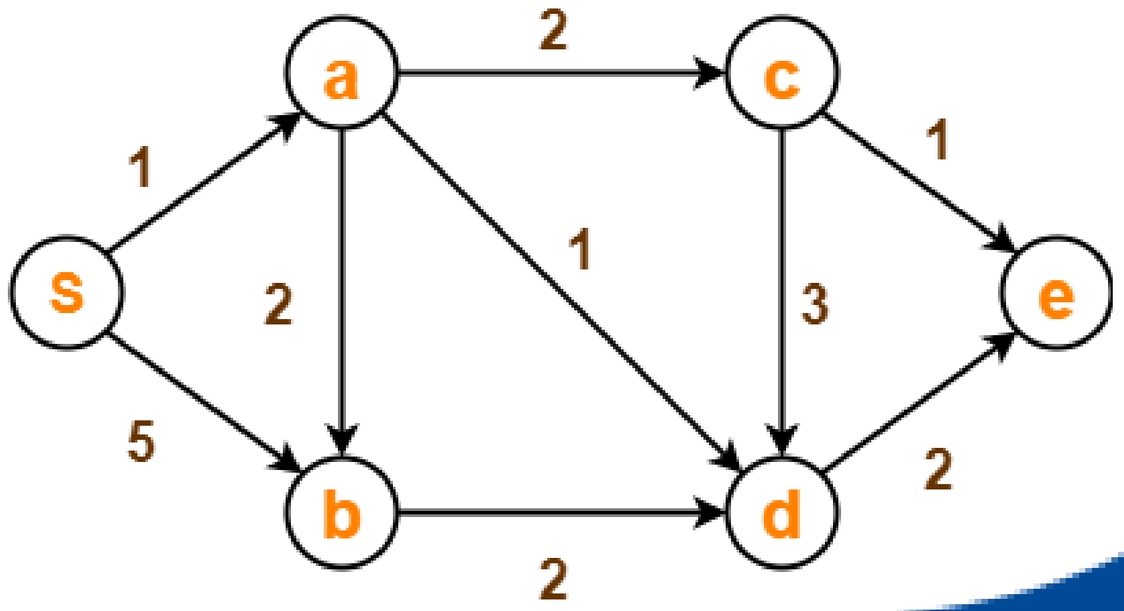
If 5 + 2 < 9

then d[v] = 5+2 and Π[v] = u

~Dr Gaurav Kumar, Asst. Prof, CEA, GLAU

Practice Question on Dijkstra's Algorithm

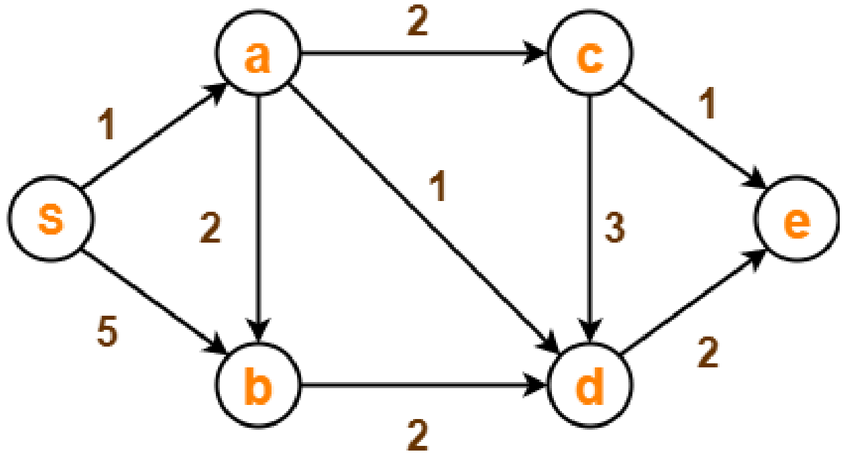
Problem: Using Dijkstra's Algorithm, find the shortest distance from source vertex 'S' to remaining vertices in the following graph. Also, write the order in which the vertices are visited.



~Dr Gaurav Kumar, Asst. Prof, CEA, GLAU

Practice Question on Dijkstra's Algorithm

Step 1



The following two sets are created-

- Unvisited set : {S , a , b , c , d , e}
- Visited set : { }

~Dr Gaurav Kumar, Asst. Prof, CEA, GLAU

Practice Question on Dijkstra's Algorithm

Step 2

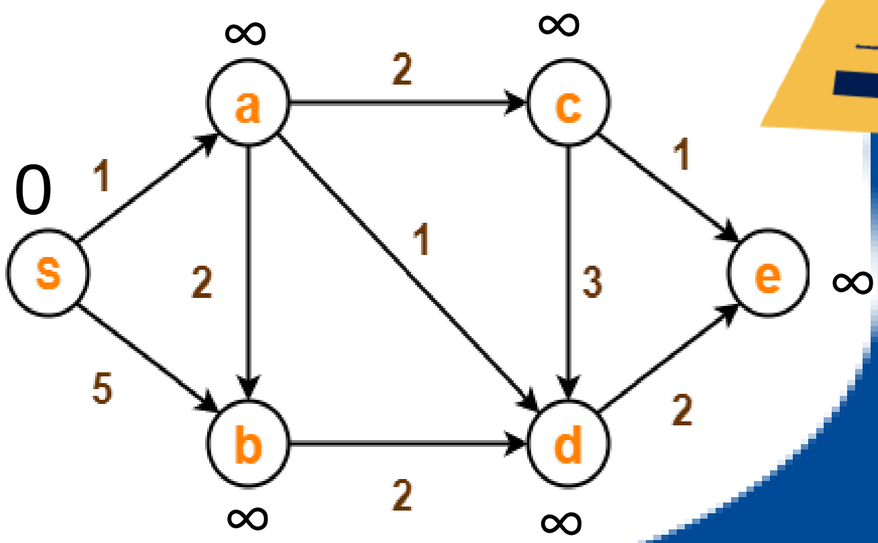
The two variables Π and d are created for each

vertex and initialized as-

Π[S] = Π[a] = Π[b] = Π[c] = Π[d] = Π[e] = NIL

d[S] = 0

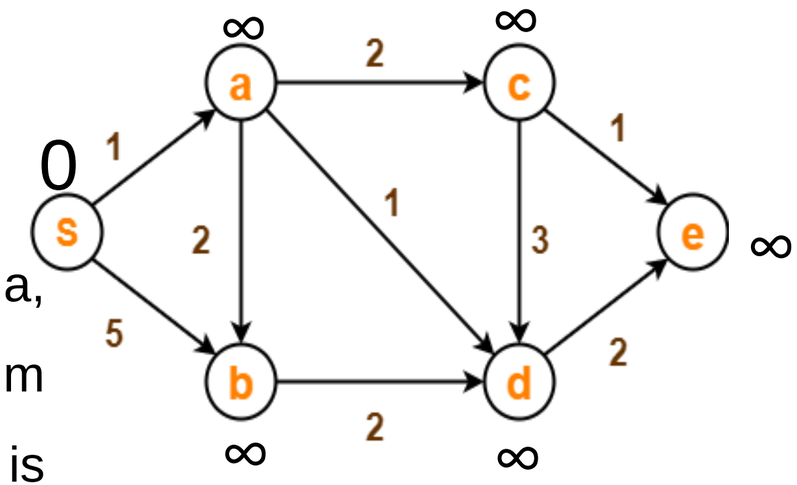
d[a] = d[b] = d[c] = d[d] = d[e] = ∞



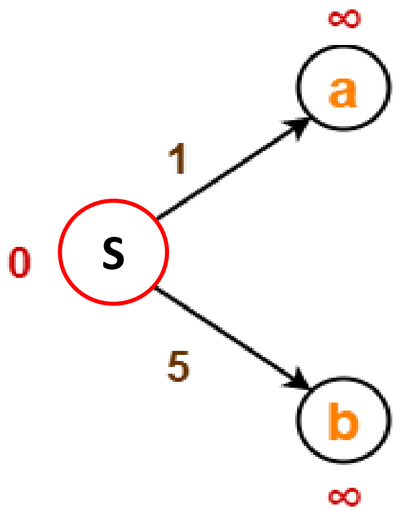
~Dr Gaurav Kumar, Asst. Prof, CEA, GLAU

Practice Question on Dijkstra's Algorithm

Step 3



- Among unprocessed vertices (s, a, b, c, d, e) a vertex with minimum value of variable distance ‘d’ is chosen



- Vertex ‘S’ is chosen. This is because shortest path estimate for vertex ‘S’ is least. (d(s)= 0)

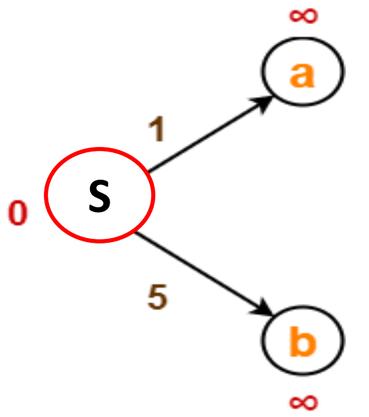
- The **outgoing edges of vertex ‘S’** are relaxed.

~Dr Gaurav Kumar, Asst. Prof, CEA, GLAU

Practice Question on Dijkstra's Algorithm

Step 3

The outgoing edges of vertex ‘S’ are relaxed.



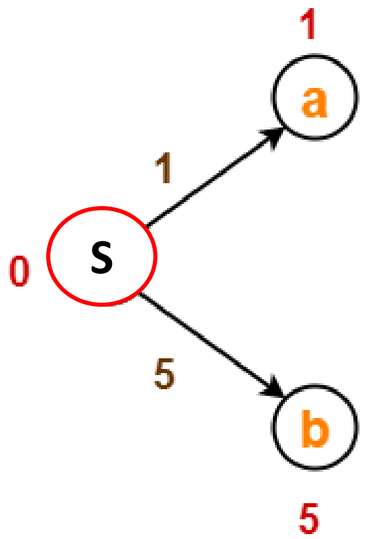
Now, if d[S] + 1 = 0 + 1 = 1 < ∞

Then d[a] = 1 and Π[a] = S

Here, d[S] + 5 = 0 + 5 = 5 < ∞

Then d[b] = 5 and Π[b] = S

After edge relaxation, our shortest path tree is-

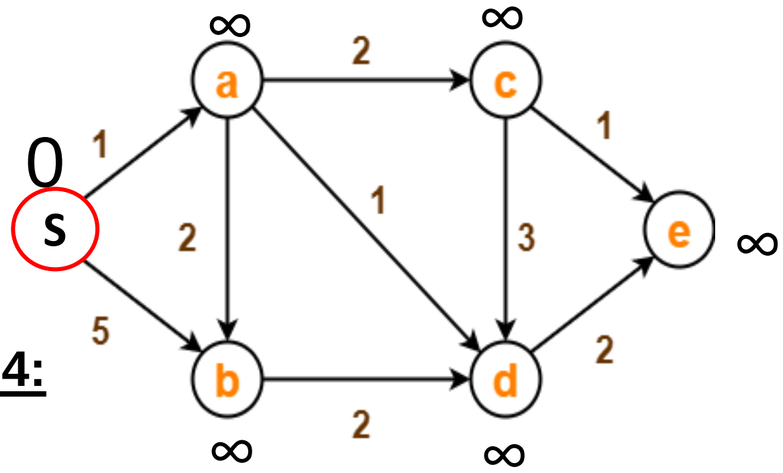


Now, the sets are updated as-

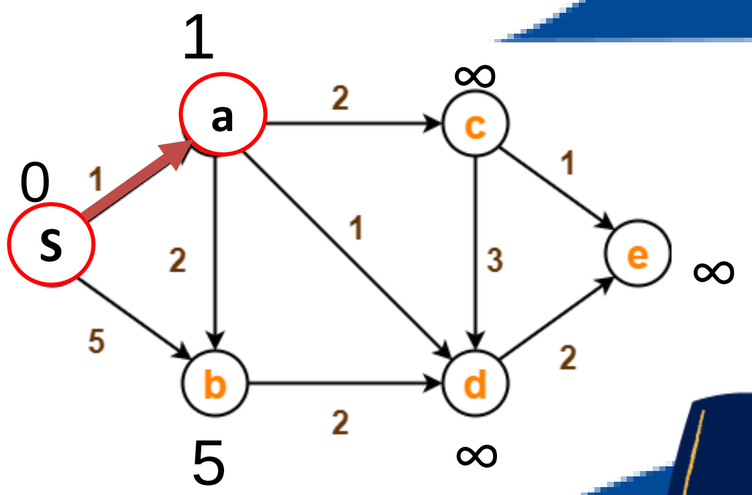
Unvisited set : {a , b , c , d , e} **Visited set : {S}**

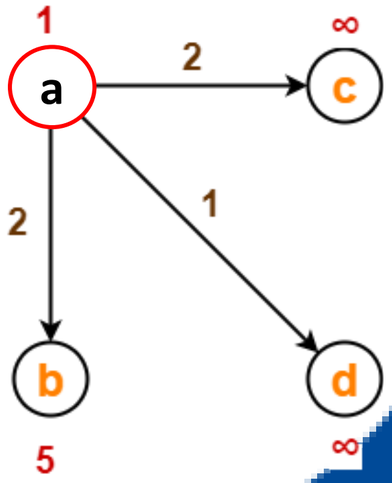
~Dr Gaurav Kumar, Asst. Prof, CEA, GLAU

Practice Question on Dijkstra's Algorithm



Step-04:





- Among unprocessed vertices (a,b,c,d,e), a vertex with minimum value of variable ‘d’ (distance) is chosen
- Vertex ‘a’ is chosen. This is because shortest path [d(a)] estimate for vertex ‘a’ is least.
- Then, the outgoing edges of vertex ‘a’

~Dr Gaurav Kumar, Asst. Prof, CEA, GLAU

Practice Question on Dijkstra's Algorithm

Step-04:

Now,

d[a] + 2 = 1 + 2 = 3 < ∞

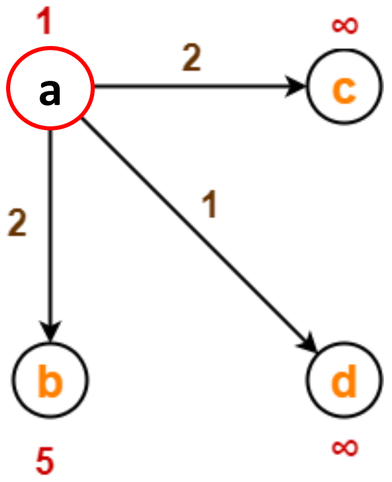
∴ d[c] = 3 and Π[c] = a

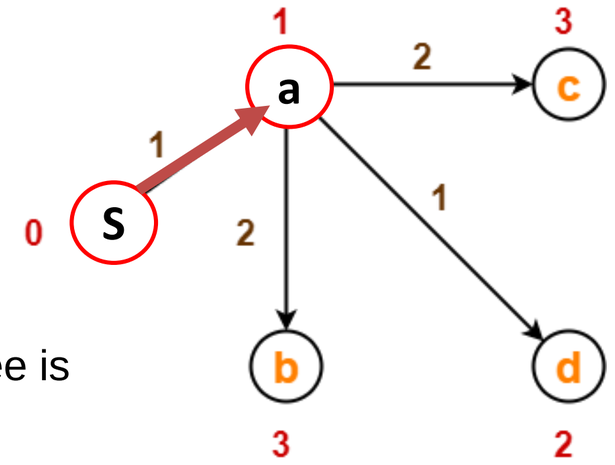
d[a] + 1 = 1 + 1 = 2 < ∞

∴ d[d] = 2 and Π[d] = a

d[b] + 2 = 1 + 2 = 3 < 5

∴ d[b] = 3 and Π[b] = a





After edge relaxation, our shortest path tree is

Now, the sets are updated as-

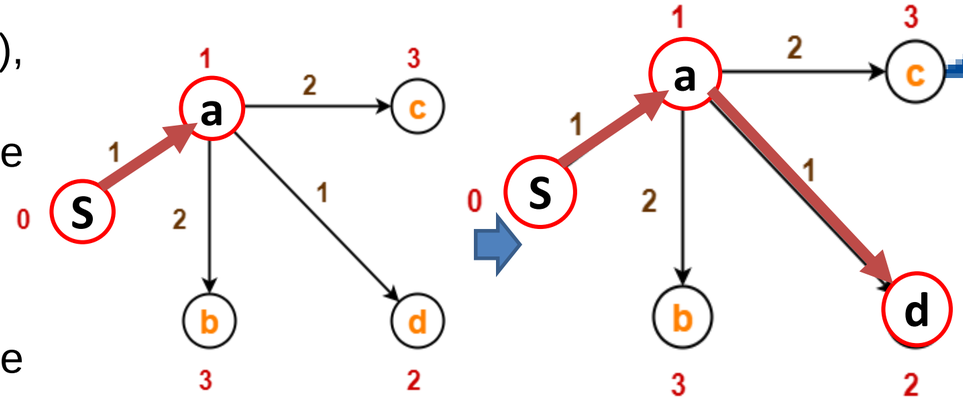
Unvisited set : {b , c , d , e} Visited set : {S , a}

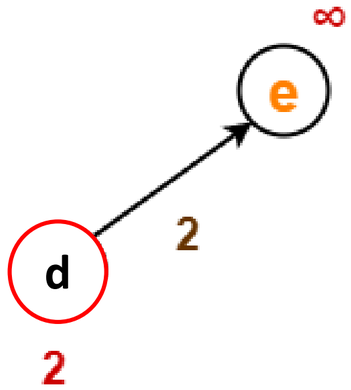
~Dr Gaurav Kumar, Asst. Prof, CEA, GLAU

Practice Question on Dijkstra's Algorithm

Step-05:

- Among unprocessed vertices (b,c,d,e), a vertex with minimum value of variable distance (d) is chosen
- Vertex ‘d’ is chosen. This is because shortest path estimate for vertex ‘d’ is least.
- The outgoing edges of vertex ‘d’ are relaxed.





~Dr Gaurav Kumar, Asst. Prof, CEA, GLAU

Practice Question on Dijkstra's Algorithm

Step-05:

The outgoing edges of vertex ‘d’ are relaxed.

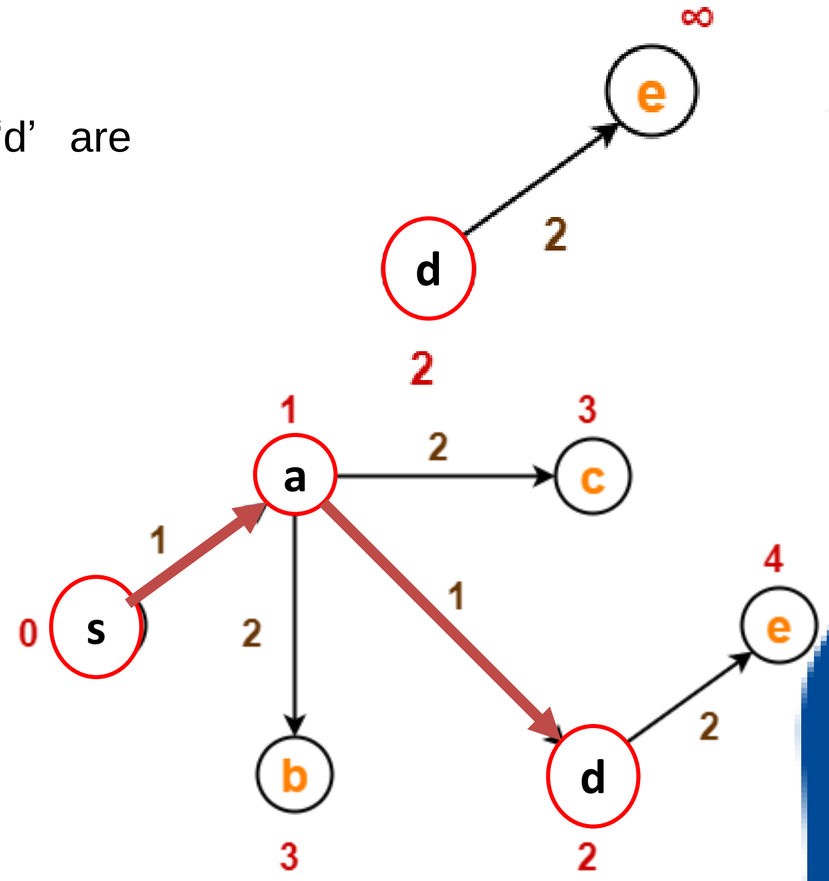
Now, d[d] + 2 = 2 + 2 = 4 < ∞

∴ d[e] = 4 and Π[e] = d

After edge relaxation, our shortest path tree is

Now, the sets are updated as-

Unvisited set : {b , c , e} Visited set : {S , a , d}

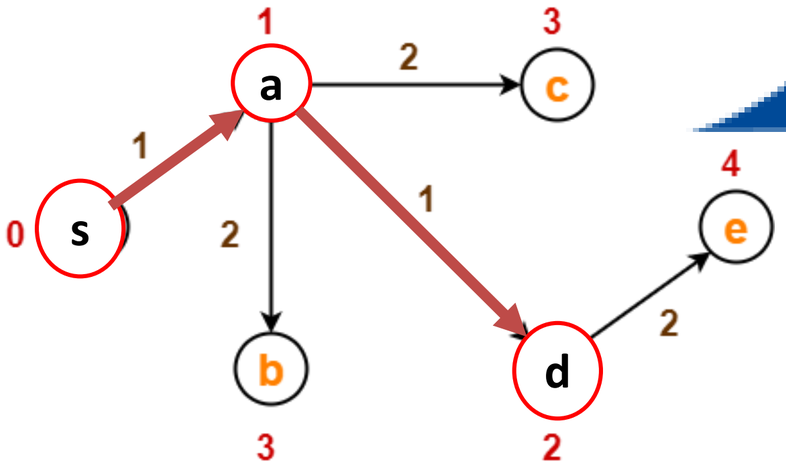


~Dr Gaurav Kumar, Asst. Prof, CEA, GLAU

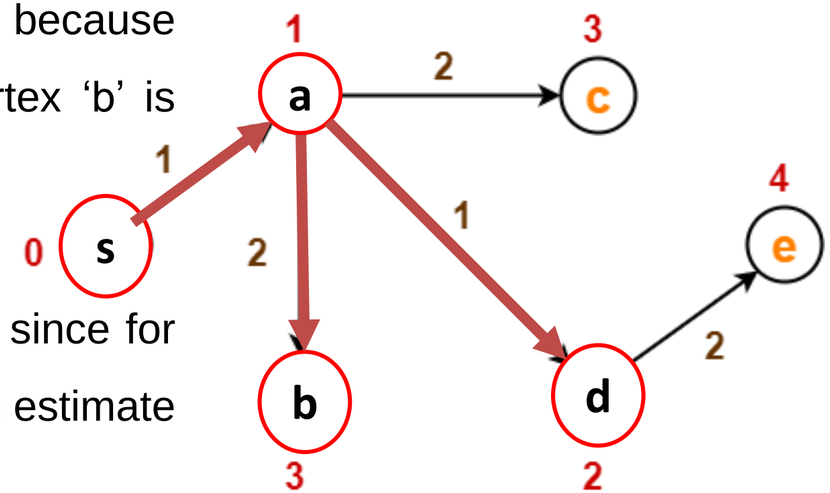
Practice Question on Dijkstra's Algorithm

Step-06:

- Among unprocessed vertices {b,c,e), a vertex with minimum value of variable distance (d) is chosen



- Vertex ‘b’ is chosen. This is because shortest path estimate for vertex ‘b’ is least.



- Vertex ‘c’ may also be chosen since for both the vertices, shortest path estimate is least.

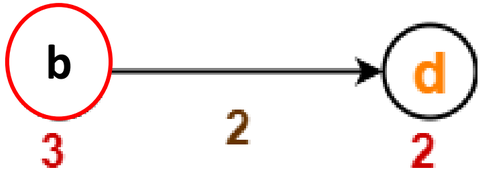
- The outgoing edges of vertex ‘b’ are relaxed.

~Dr Gaurav Kumar, Asst. Prof, CEA, GLAU

Practice Question on Dijkstra's Algorithm

Step-06:

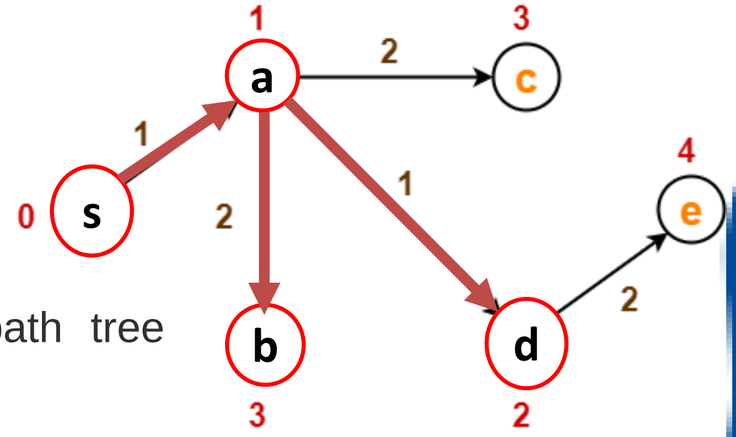
The outgoing edges of vertex ‘b’ are relaxed.



Now,

d[b] + 2 = 3 + 2 = 5 > 2

∴ No change



After edge relaxation, our shortest path tree remains the same as in Step-05.

Now, the sets are updated as-

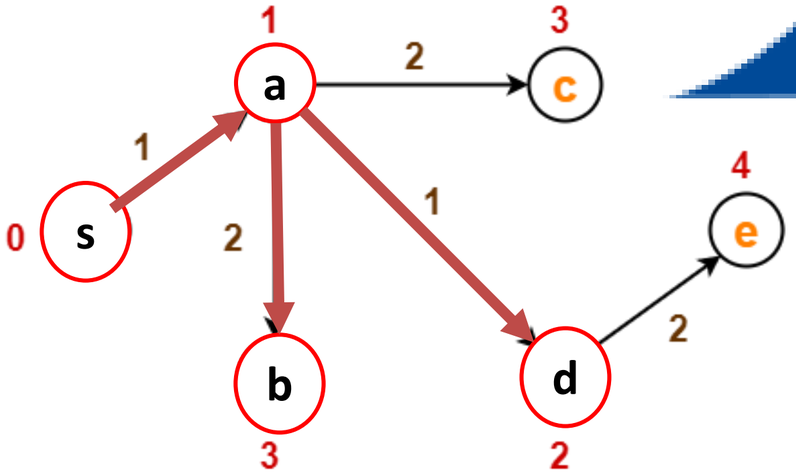
Unvisited set : {c , e} Visited set : {S , a , d , b}

~Dr Gaurav Kumar, Asst. Prof, CEA, GLAU

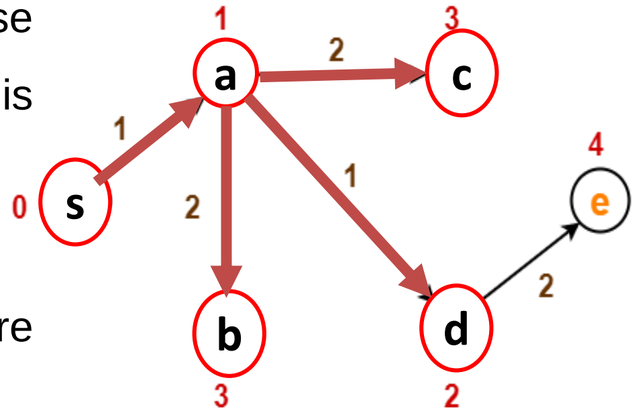
Practice Question on Dijkstra's Algorithm

Step-07:

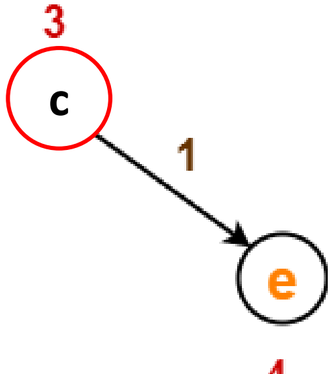
- Among unprocessed vertices (c, e), a vertex with minimum value of variable distance (d) is chosen



- Vertex ‘c’ is chosen. This is because shortest path estimate for vertex ‘c’ is least.



- The outgoing edges of vertex ‘c’ are relaxed.

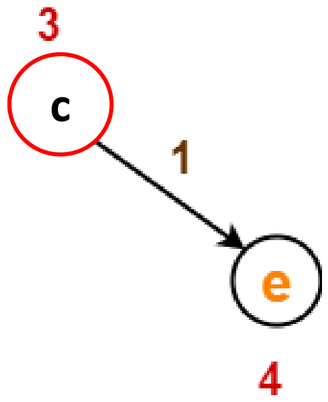


~Dr Gaurav Kumar, Asst. Prof, CEA, GLAU

Practice Question on Dijkstra's Algorithm

Step-07:

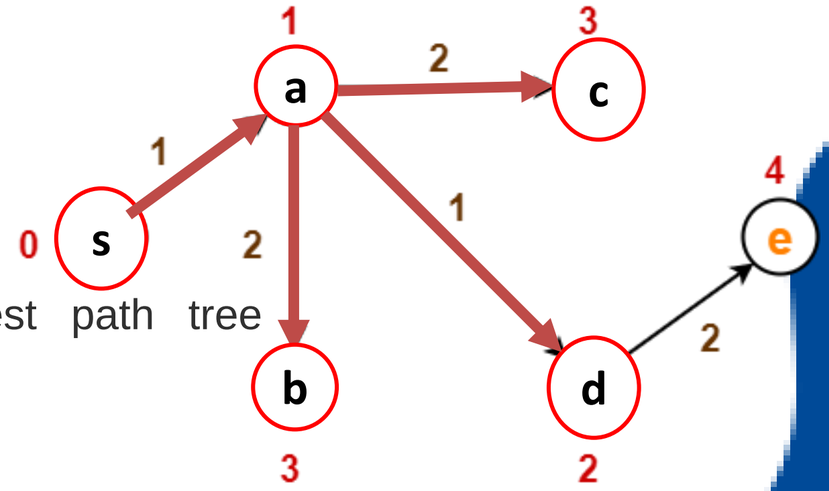
The outgoing edges of vertex ‘c’ are relaxed.



Now,

d[c] + 1 = 3 + 1 = 4 = 4

∴ No change



After edge relaxation, our shortest path tree remains the same as in Step-06.

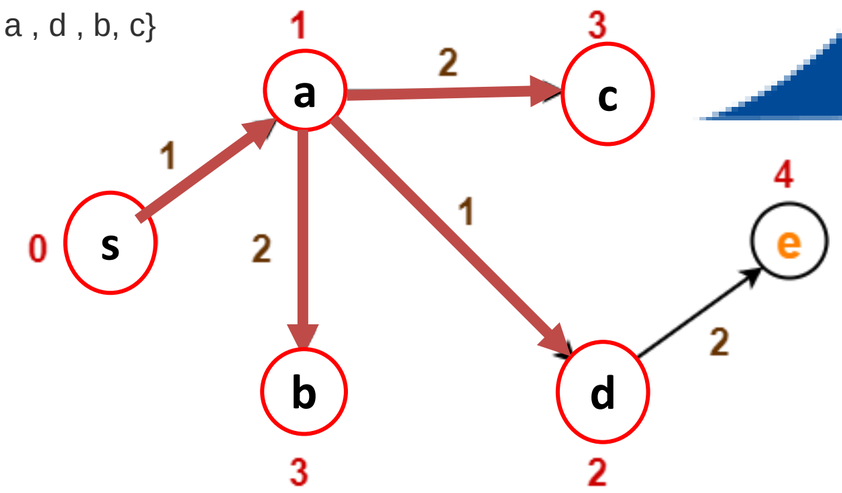
Now, the sets are updated as-

Unvisited set : {e} Visited set : {S , a , d , b, c}

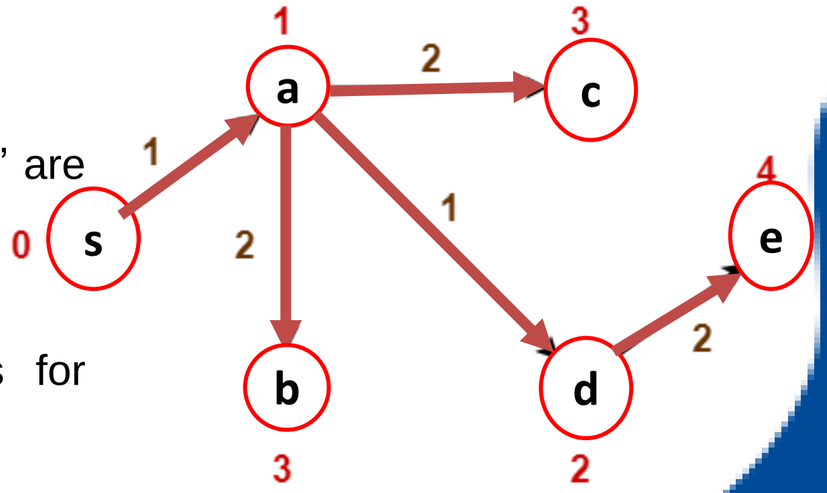
~Dr Gaurav Kumar, Asst. Prof, CEA, GLAU

Practice Question on Dijkstra's Algorithm

Step-08: Unvisited set : {e} Visited set : {S , a , d , b, c}



- Among unprocessed vertices (e), a vertex with minimum value of variable distance (d) is chosen
- Vertex ‘e’ is chosen. This is because shortest path estimate for vertex ‘e’ is least.
- The outgoing edges of vertex ‘e’ are relaxed.
- There are no outgoing edges for vertex ‘e’.



Shortest Path Tree

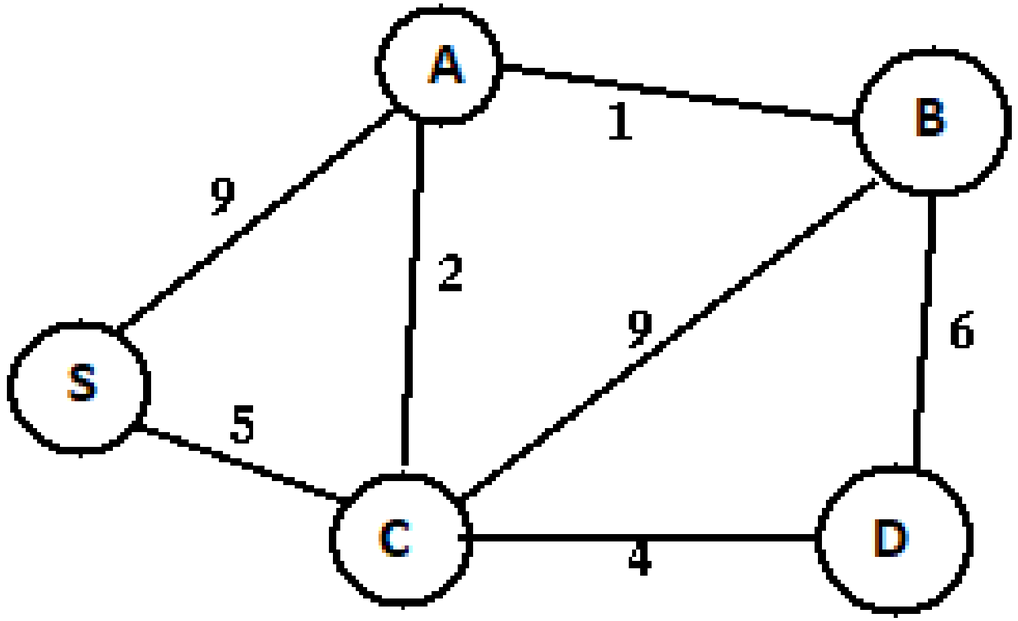
Now, the sets are updated as-

- Unvisited set : { }
- Visited set : {S , a , d , b , c , e}

The order in which all the vertices are processed is : **S , a , d , b , c , e**.

Self Practice Question on Dijkstra's Algorithm

Given the weighted and connected graph and source vertex s, find the tree representing shortest path between s and all other vertices.

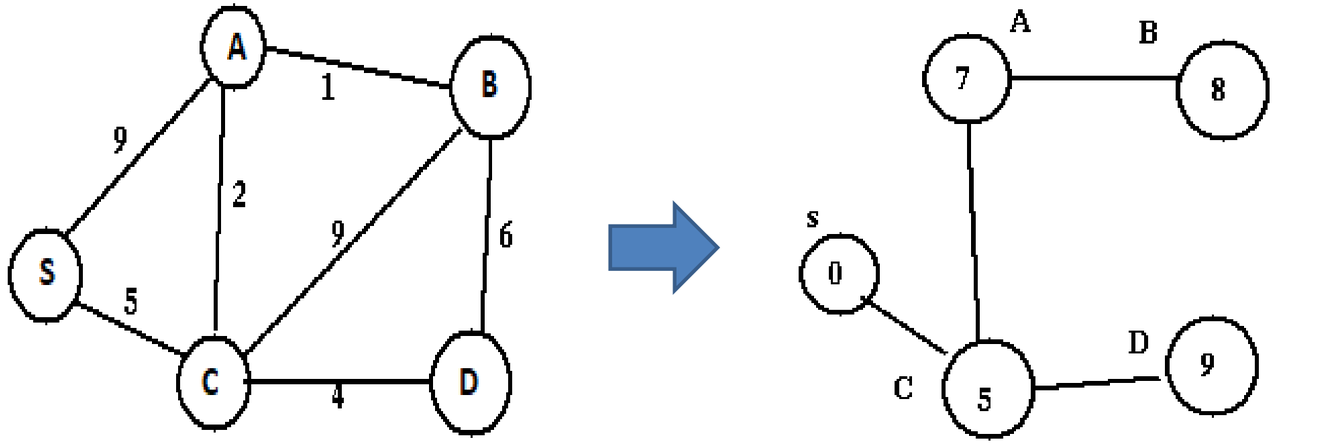


~Dr Gaurav Kumar, Asst. Prof, CEA, GLAU

Self Practice Question on Dijkstra's Algorithm

Given the weighted and connected graph and source vertex s, find the tree representing shortest path between s and all other vertices.

[s, C, A, B, D]



Shortest path spanning tree

Note: There can be multiple shortest path spanning trees for the same graph depending on the source vertex

~Dr Gaurav Kumar, Asst. Prof, CEA, GLAU

Another Way of Writing Algorithm of Dijkstra's Algorithm

Dijkstra's Algorithm (G, W, S)

1.INITIALIZE - SINGLE - SOURCE (G, S)

2. S←∅ //Visited Set

3. Q←V [G] //Unvisited Set

→ V

4.while Q ≠ ∅

5. do u ← EXTRACT - MIN (Q) → V*V

// Finding minimum on each vertices
(V + V-1+ V-2 + V-3 +…3+2+1)

6.S ← S ∪ {u} → 1

7.for each vertex v ∈ Adj [u]

8.do RELAX (u, v, w) → V*V

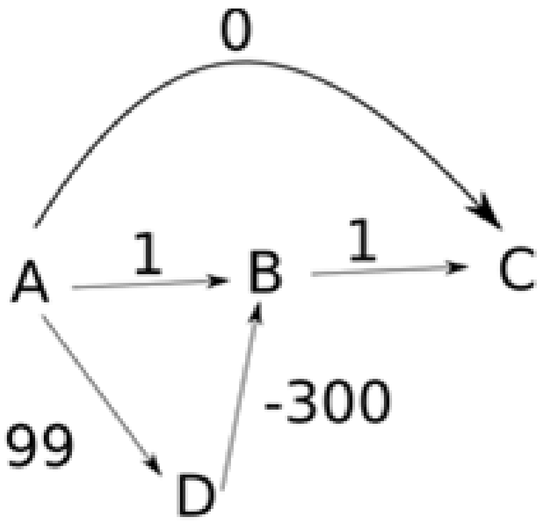
// Relaxing edges from one vertices will take max v times
For V vertices, it will take V*V times

Total Time = $V + V^2 + V^2 + 1$

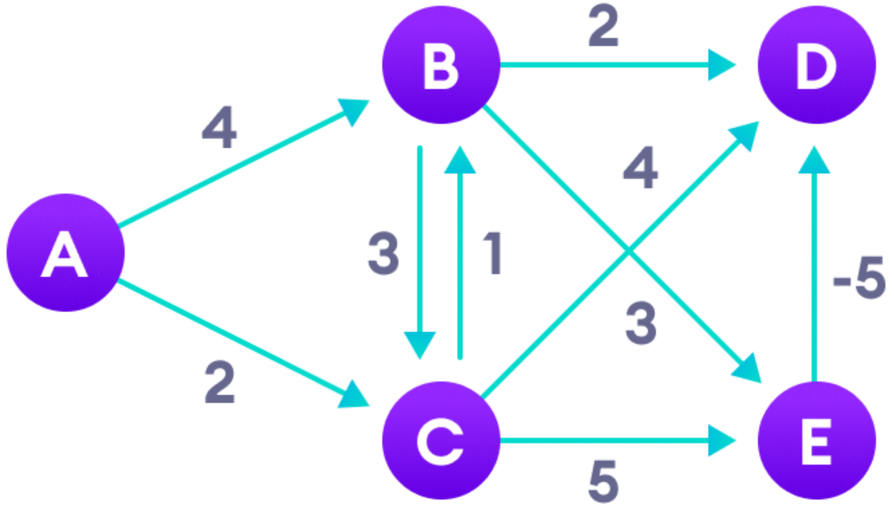
TC= $O(V^2)$

Drawback of Dijkstra's Algorithm

Dijkstra’s Algorithm cannot obtain correct shortest path(s)with weighted graphs having negative edges.



~Dr Gaurav Kumar, Asst. Prof, CEA, GLAU



05.b

Bellman-Ford Algorithm

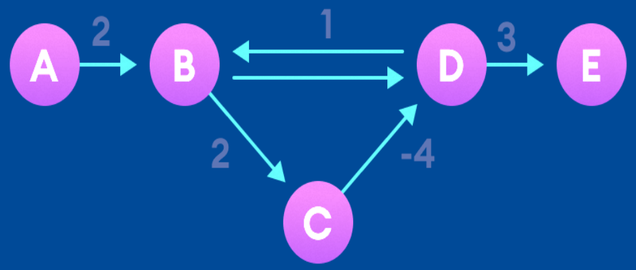
Suitable for Negative Edge Weight Graph

Dr Gaurav Kumar, Asst. Prof. CEA, GLAU

Bellman-Ford Algorithm

- Bellman Ford algorithm helps us find the shortest path from a vertex to all other vertices of a weighted graph.

- It is similar to Dijkstra's algorithm but it can work with graphs in which **edges can have negative weights**.



- It is used to **detect Negative Weight Cycle** in a Graph.

Algorithm for Bellman-Ford

BELLMAN -FORD (G, w, s)

1. INITIALIZE - SINGLE - SOURCE (G, s)
2. for i ← 1 to |V[G]| - 1 // Loop will repeat V-1 times
3. do **for each edge (u, v) ∈ E [G]**
4. do **RELAX (u, v, w)**
5. for each edge (u, v) ∈ E [G]
6. do if d [v] > d [u] + w (u, v)
7. then return FALSE //Cycle Found
8. return TRUE. //Shortest Path

Algorithm for Bellman-Ford

OR

Step 1: Initializes distances from the source to all vertices as infinite and distance to the source itself as 0.

Step 2: This step calculates shortest distances. Do following |V|-1 times where |V| is the number of vertices in given graph.

a) Do following for each edge u-v

If dist[v] > dist[u] + weight of edge uv,

then update dist[v] = dist[u] + weight of edge uv

(This step is called Edge Relaxation)

Step 3: This step reports if there is a negative weight cycle in graph. Do following for each edge u-v

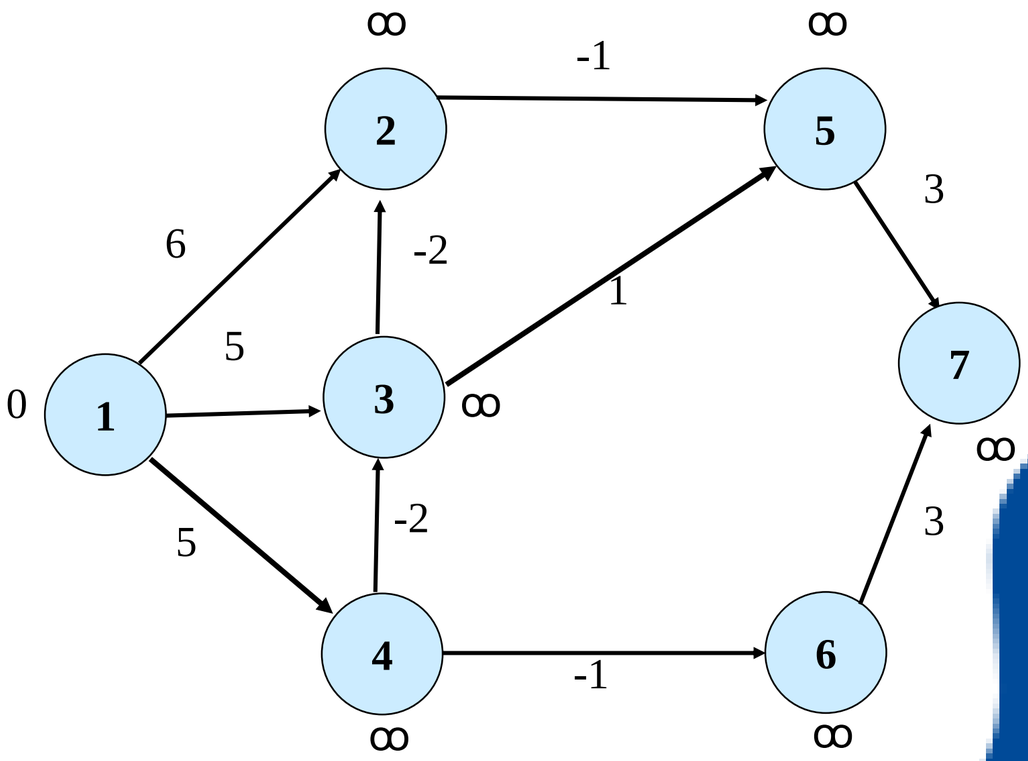
a) If dist[v] > dist[u] + weight of edge uv,

then “Graph contains negative weight cycle”

Practice Problem using Bellman-Ford Algorithm

Find the shortest path from a vertex “1” to all other vertices of a weighted graph

Step 1: Initializes distances from the source to all vertices as infinite and distance to the source itself as 0.



~Dr Gaurav Kumar, Asst. Prof, CEA, GLAU

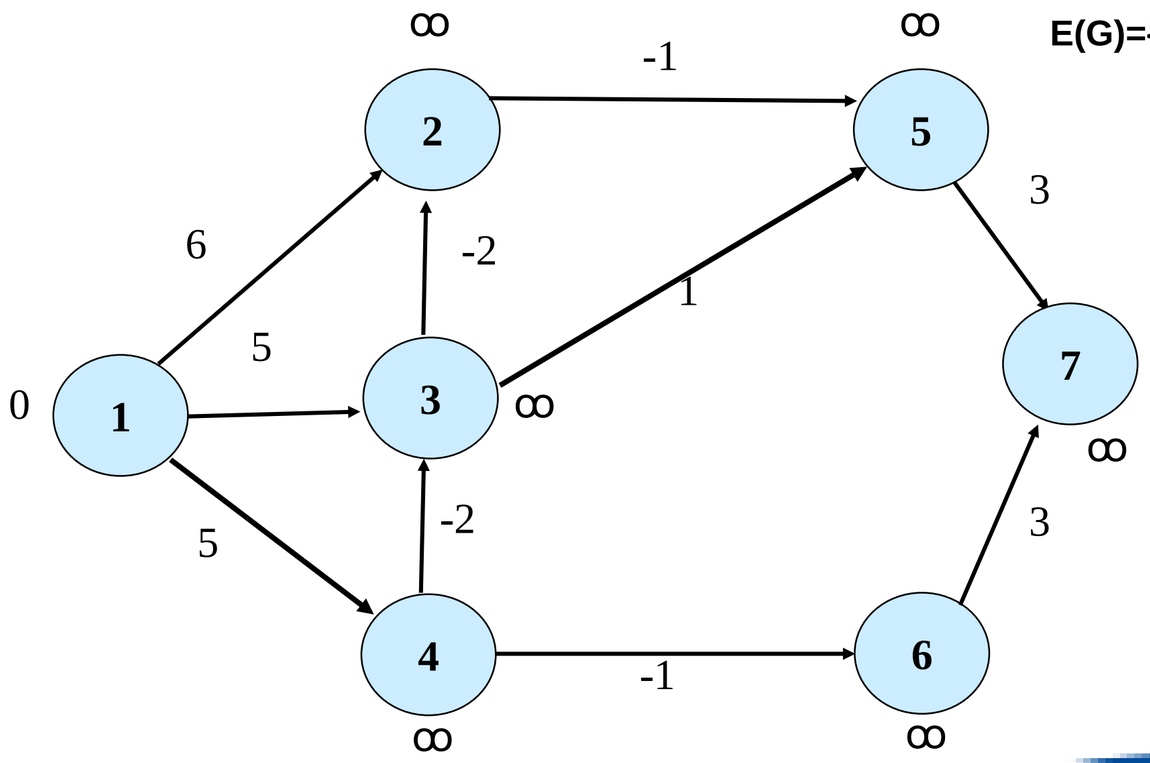
Practice Problem using Bellman-Ford Algorithm

Step 2:

for each edge u-v

If dist[v] > dist[u] + weight of edge

then update dist[v] = dist[u] + weight of edge uv



Iteration 1

E(G)={(1,2), (1,3),(1,4), (2,5), (3,5),(3,2) ,(4,3),(4,6),(5,7), (6,7)}

~Dr Gaurav Kumar, Asst. Prof, CEA, GLAU

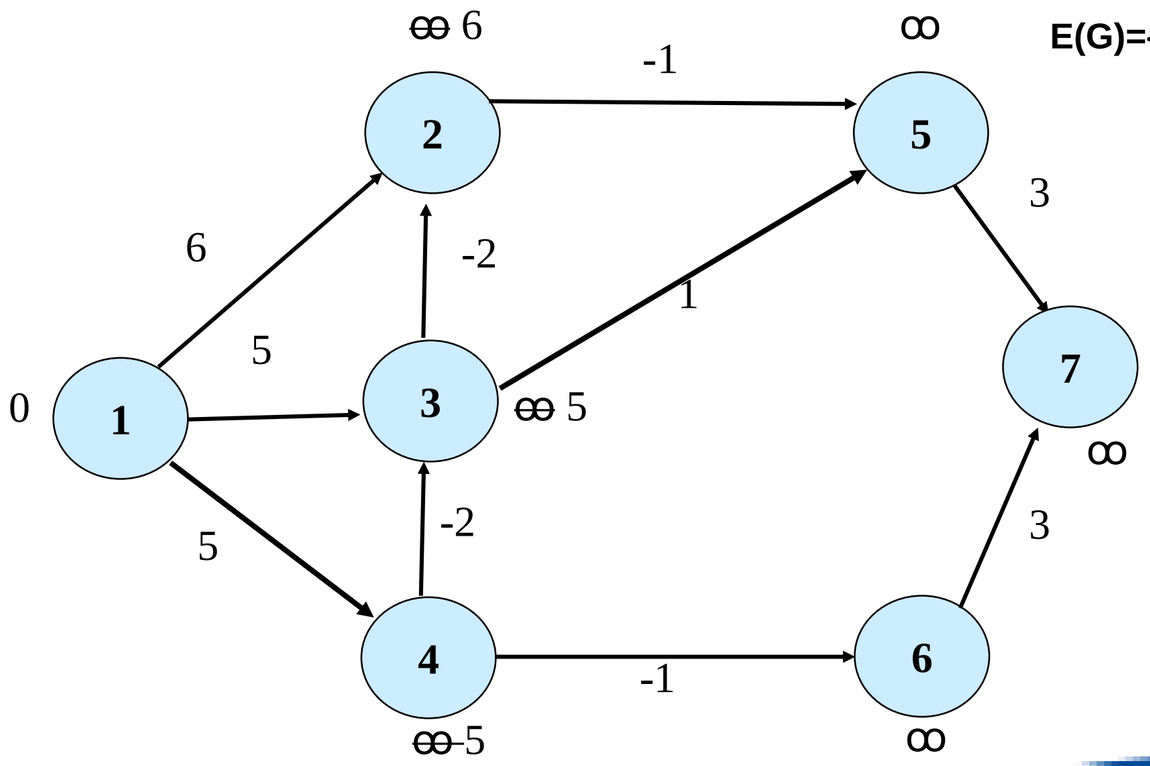
Practice Problem using Bellman-Ford Algorithm

Step 2:

for each edge u-v

If dist[v] > dist[u] + weight of edge

then update dist[v] = dist[u] + weight of edge uv



Iteration 1

E(G)={(1,2), (1,3),(1,4), (2,5), (3,5),(3,2) ,(4,3),(4,6),(5,7), (6,7)}

~Dr Gaurav Kumar, Asst. Prof, CEA, GLAU

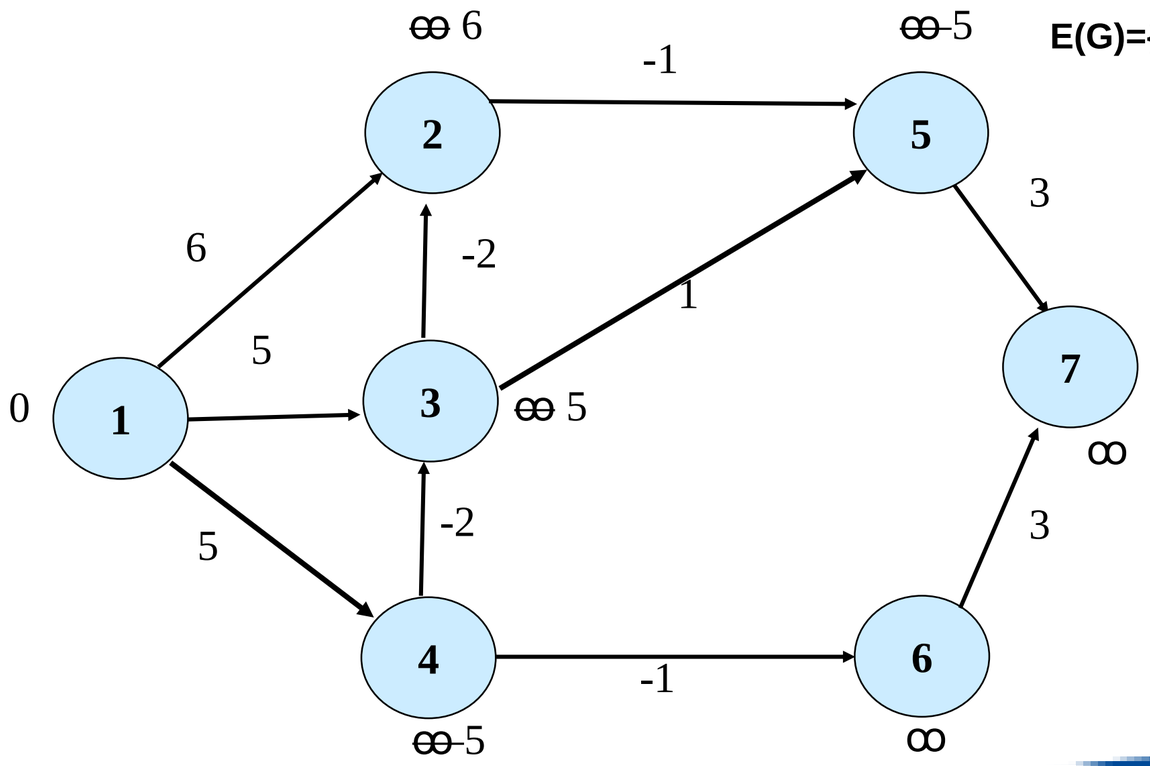
Practice Problem using Bellman-Ford Algorithm

Step 2:

for each edge u-v

If dist[v] > dist[u] + weight of edge

then update dist[v] = dist[u] + weight of edge uv



Iteration 1

E(G)={(1,2), (1,3),(1,4), (2,5), (3,5),(3,2) ,(4,3),(4,6),(5,7),(6,7)}

~Dr Gaurav Kumar, Asst. Prof, CEA, GLAU

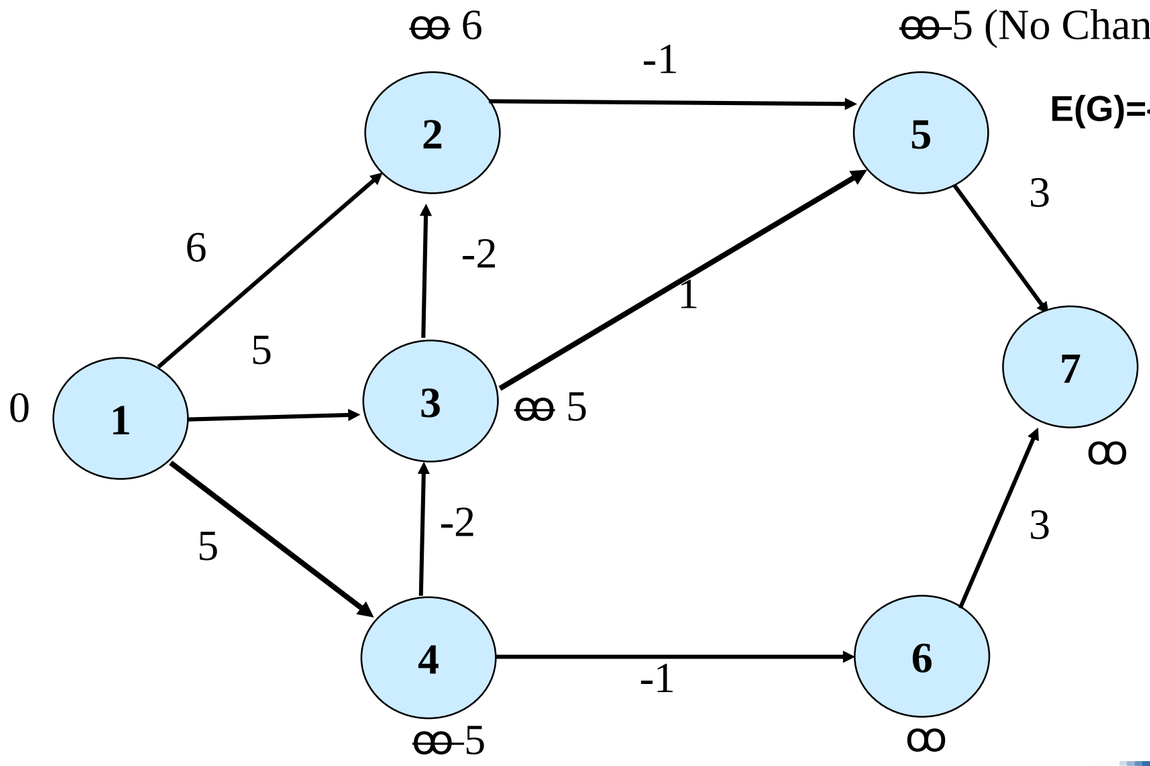
Practice Problem using Bellman-Ford Algorithm

Step 2:

for each edge u-v

If dist[v] > dist[u] + weight of edge

then update dist[v] = dist[u] + weight of edge uv



Iteration 1

E(G)={(1,2), (1,3),(1,4), (2,5), (3,5),(3,2) ,(4,3),(4,6),(5,7),(6,7)}

~Dr Gaurav Kumar, Asst. Prof, CEA, GLAU

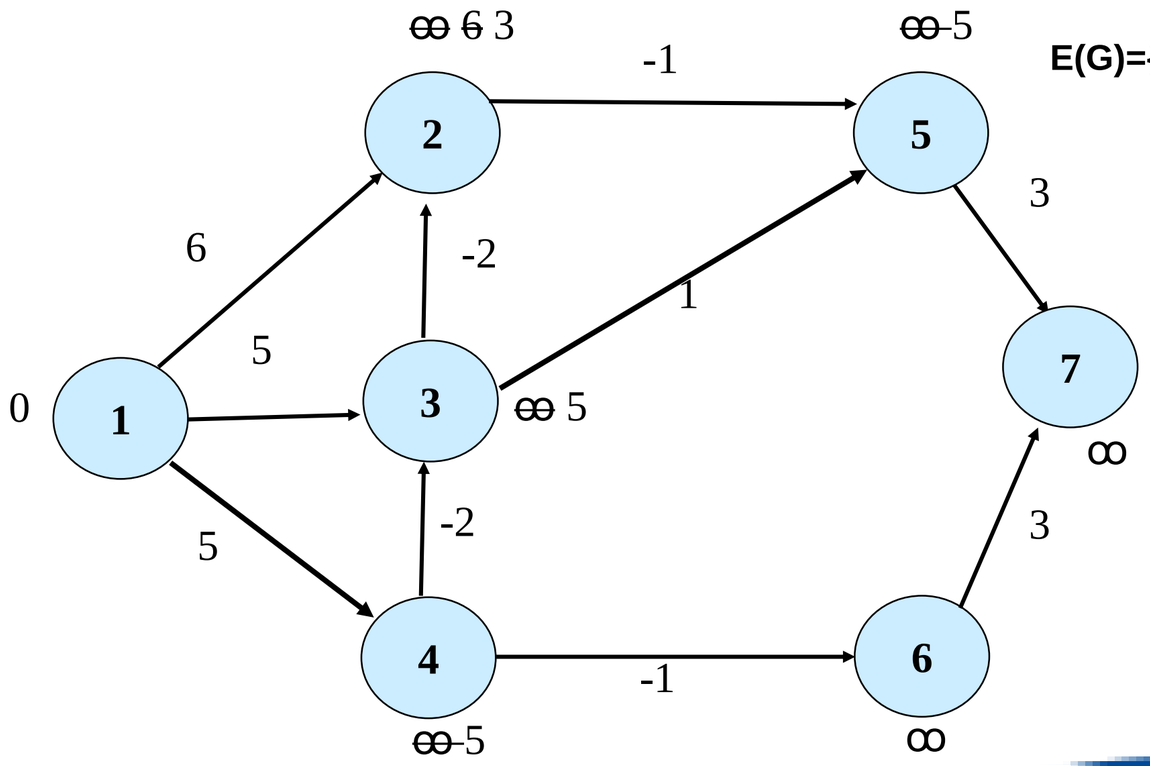
Practice Problem using Bellman-Ford Algorithm

Step 2:

for each edge u-v

If dist[v] > dist[u] + weight of edge

then update dist[v] = dist[u] + weight of edge uv



Iteration 1

E(G)={(1,2), (1,3),(1,4), (2,5), (3,5), (3,2) ,(4,3),(4,6),(5,7),(6,7)}

~Dr Gaurav Kumar, Asst. Prof, CEA, GLAU

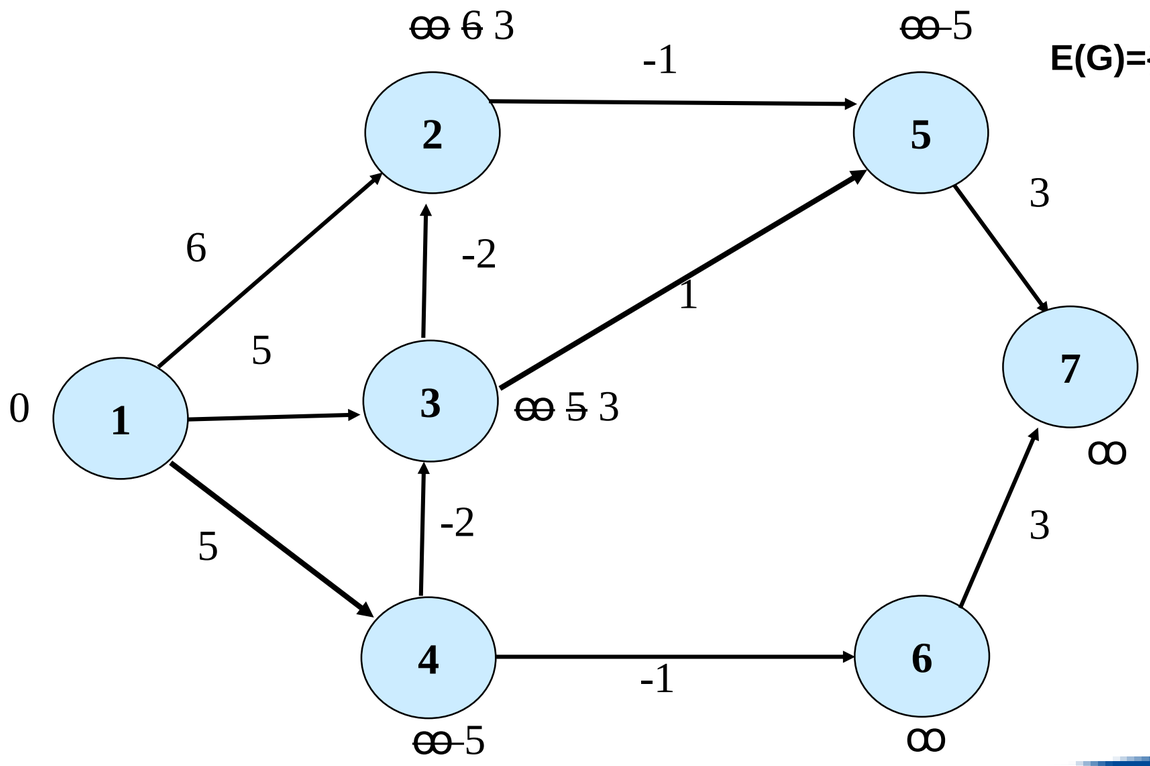
Practice Problem using Bellman-Ford Algorithm

Step 2:

for each edge u-v

If dist[v] > dist[u] + weight of edge

then update dist[v] = dist[u] + weight of edge uv



Iteration 1

E(G)={(1,2), (1,3),(1,4), (2,5), (3,5), (3,2) ,(4,3),(4,6),(5,7),(6,7)}

~Dr Gaurav Kumar, Asst. Prof, CEA, GLAU

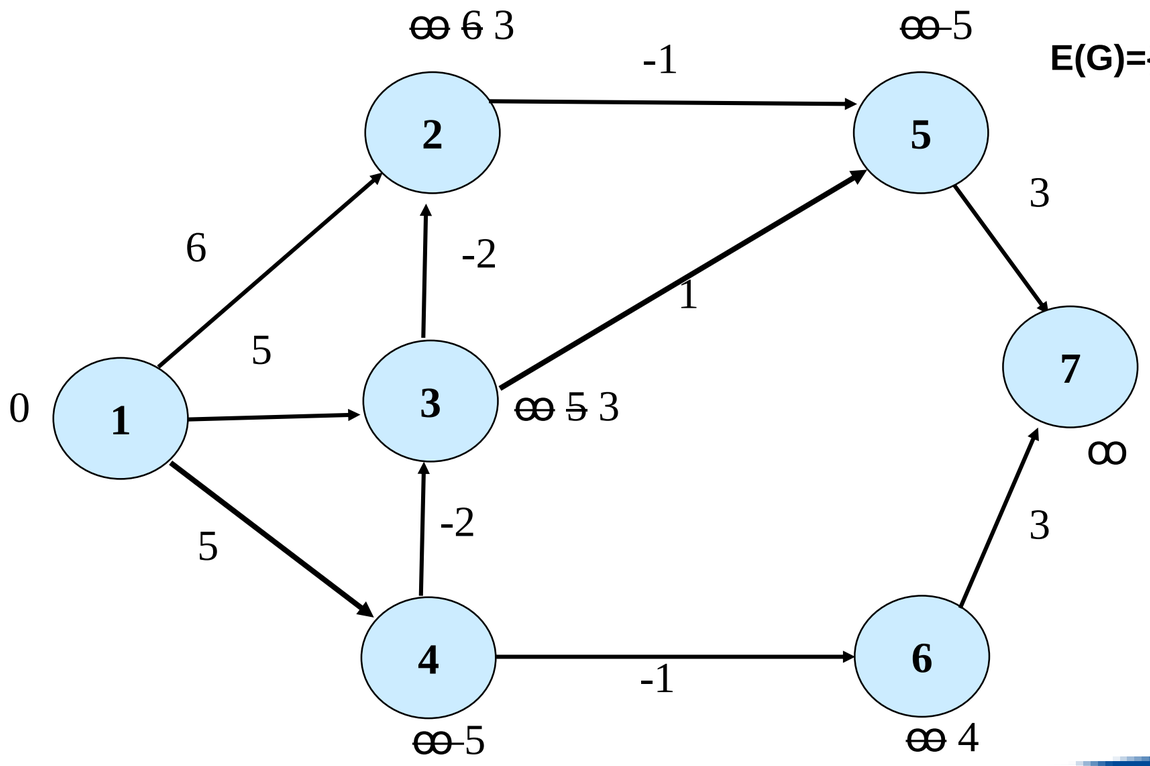
Practice Problem using Bellman-Ford Algorithm

Step 2:

for each edge u-v

If dist[v] > dist[u] + weight of edge

then update dist[v] = dist[u] + weight of edge uv



Iteration 1

E(G)={(1,2), (1,3),(1,4), (2,5), (3,5), (3,2) ,(4,3),(4,6),(5,7),(6,7)}

~Dr Gaurav Kumar, Asst. Prof, CEA, GLAU

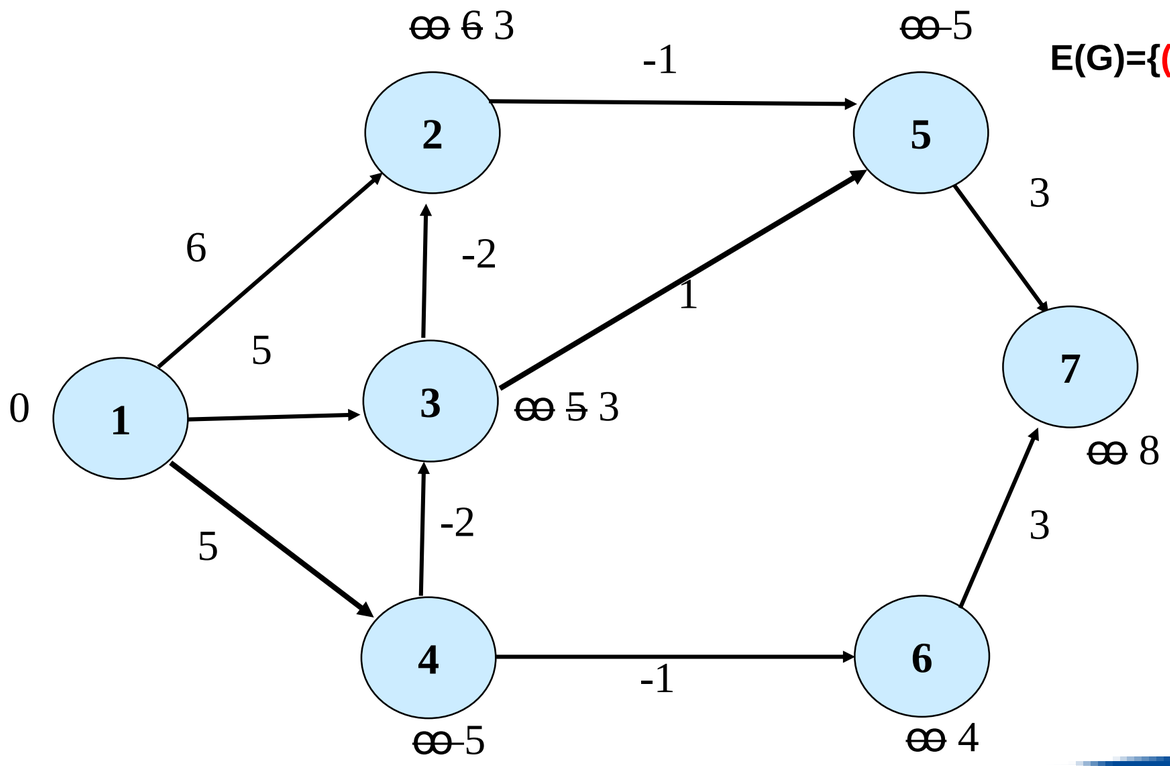
Practice Problem using Bellman-Ford Algorithm

Step 2:

for each edge u-v

If dist[v] > dist[u] + weight of edge

then update dist[v] = dist[u] + weight of edge uv



Iteration 1

E(G)={(1,2), (1,3),(1,4), (2,5), (3,5), (3,2) ,(4,3),(4,6),(5,7),(6,7)}

~Dr Gaurav Kumar, Asst. Prof, CEA, GLAU

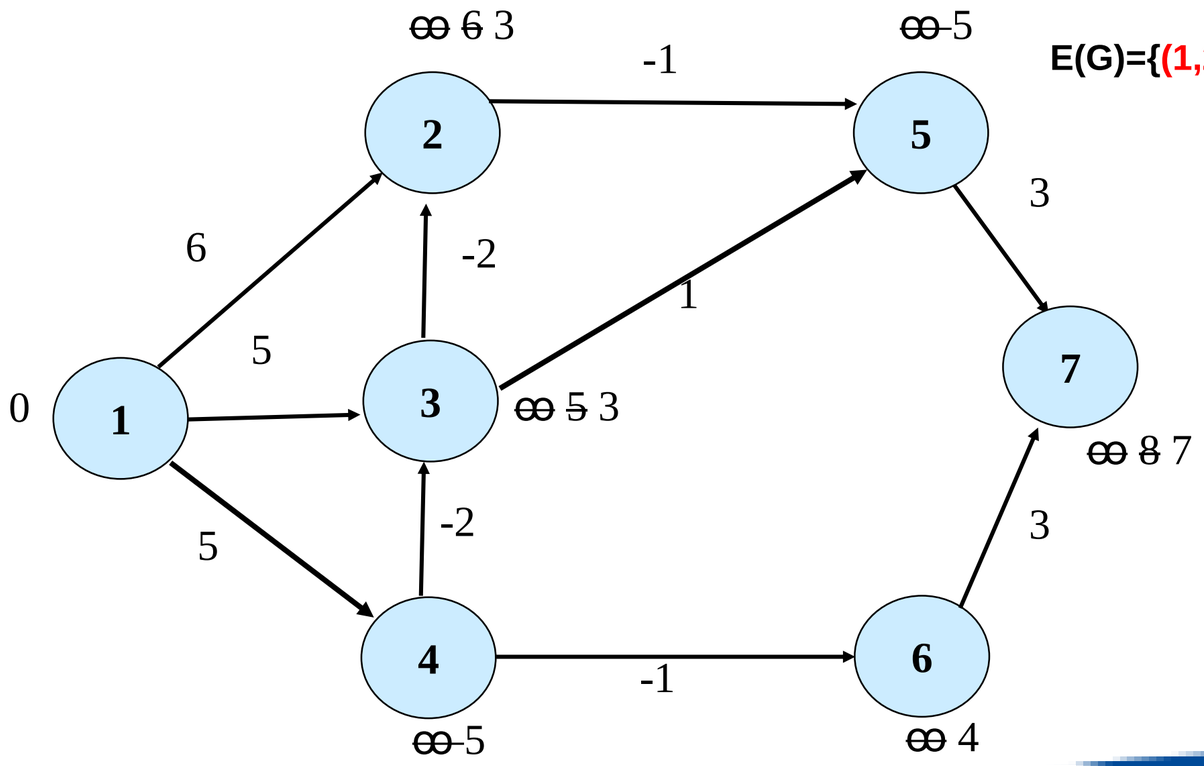
Practice Problem using Bellman-Ford Algorithm

Step 2:

for each edge u-v

If dist[v] > dist[u] + weight of edge

then update dist[v] = dist[u] + weight of edge uv



Iteration 1

E(G)={(1,2), (1,3),(1,4), (2,5), (3,5), (3,2) ,(4,3),(4,6),(5,7),(6,7)}

~Dr Gaurav Kumar, Asst. Prof, CEA, GLAU

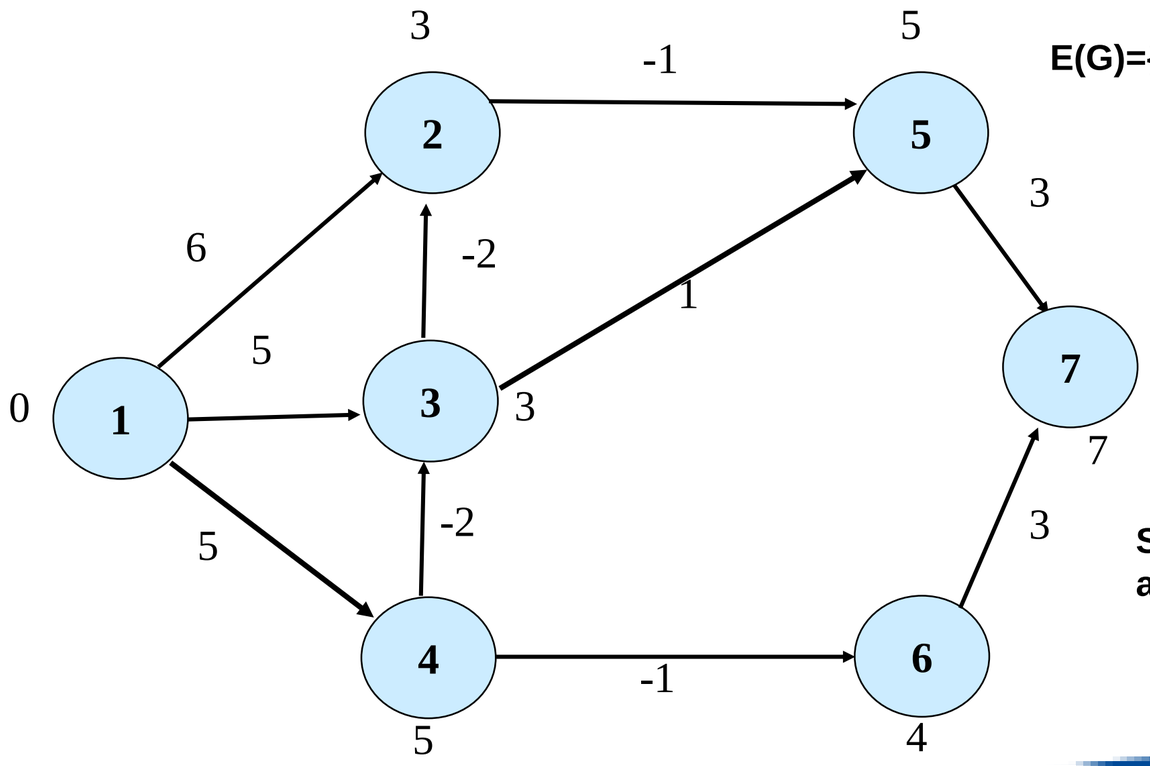
Practice Problem using Bellman-Ford Algorithm

Step 2:

for each edge u-v

If dist[v] > dist[u] + weight of edge

then update dist[v] = dist[u] + weight of edge uv



Iteration 1

E(G)={(1,2), (1,3),(1,4), (2,5), (3,5), (3,2) ,(4,3),(4,6),(5,7),(6,7)}

Shortest Spanning Tree after Iteration 1

~Dr Gaurav Kumar, Asst. Prof, CEA, GLAU

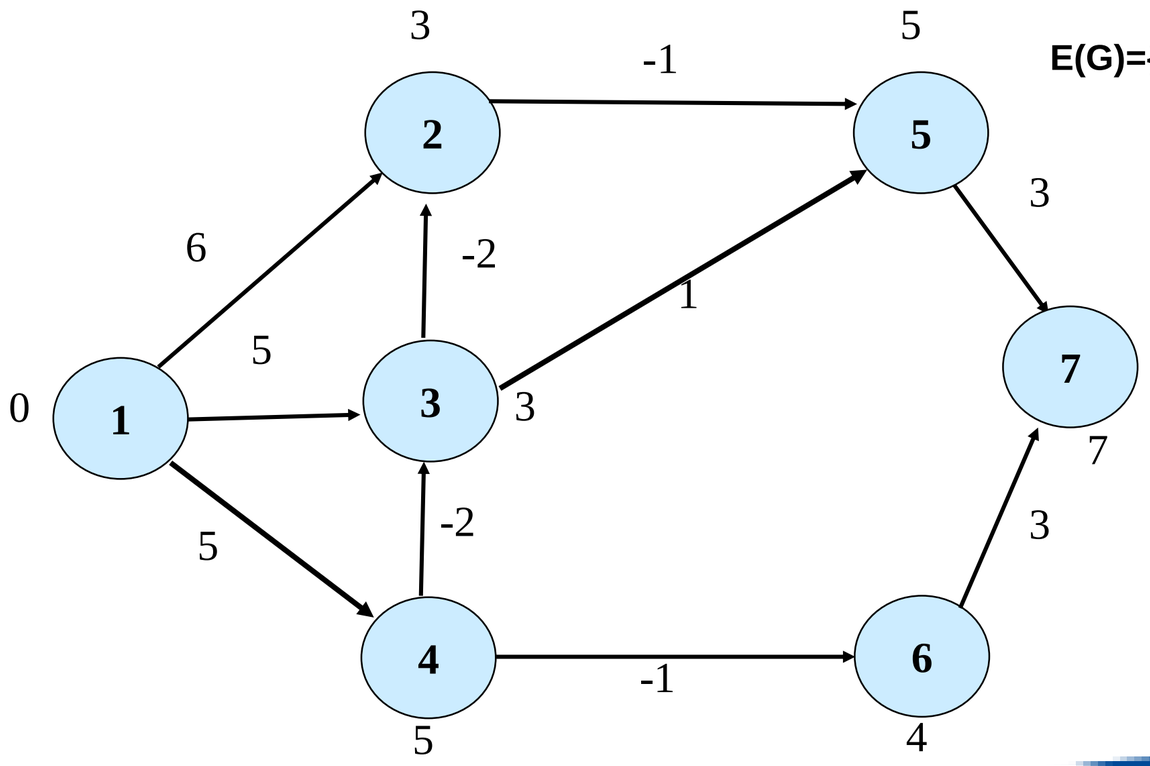
Practice Problem using Bellman-Ford Algorithm

Step 3: Step 2 will repeat for V-1 times

for each edge u-v

If dist[v] > dist[u] + weight of edge

then update dist[v] = dist[u] + weight of edge uv



Iteration 2

E(G)={(1,2), (1,3),(1,4), (2,5), (3,5), (3,2) ,(4,3),(4,6),(5,7),(6,7)}

~Dr Gaurav Kumar, Asst. Prof, CEA, GLAU

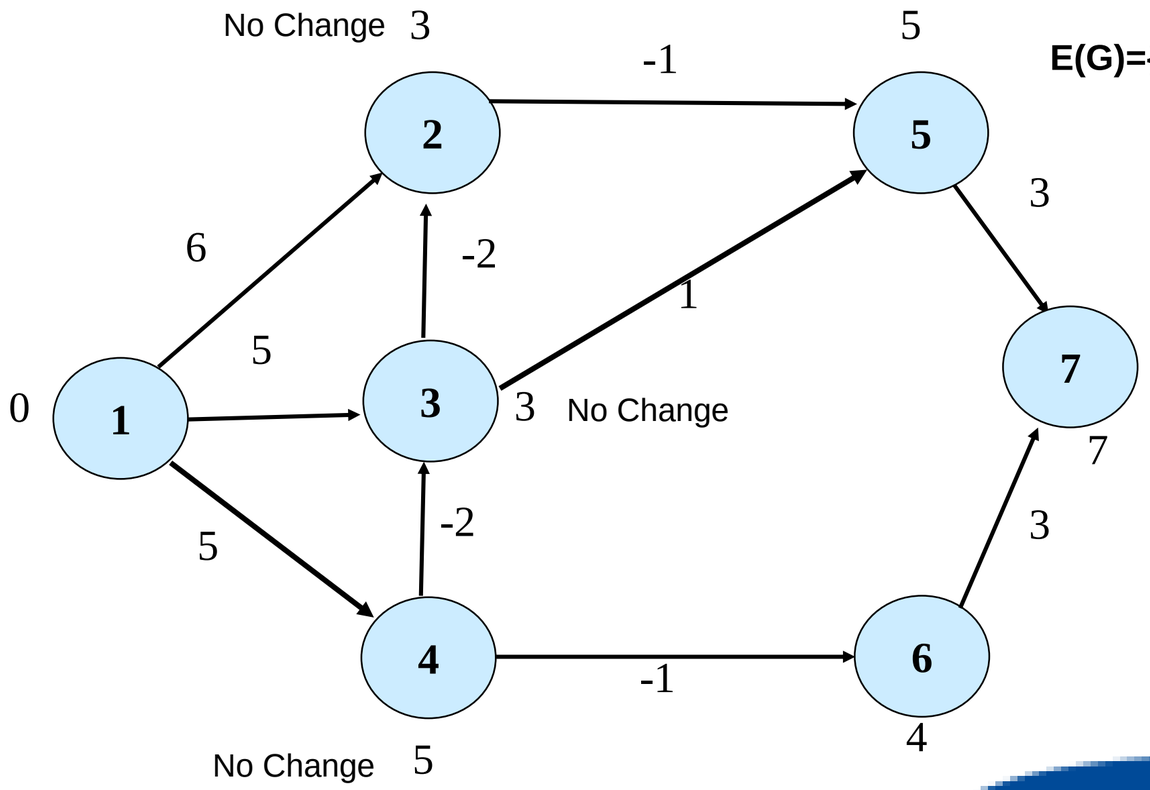
Practice Problem using Bellman-Ford Algorithm

Step 3: Step 2 will repeat for V-1 times

for each edge u-v

If dist[v] > dist[u] + weight of edge

then update dist[v] = dist[u] + weight of edge uv



Iteration 2

E(G)={(1,2), (1,3),(1,4), (2,5), (3,5), (3,2) ,(4,3),(4,6),(5,7),(6,7)}

~Dr Gaurav Kumar, Asst. Prof, CEA, GLAU

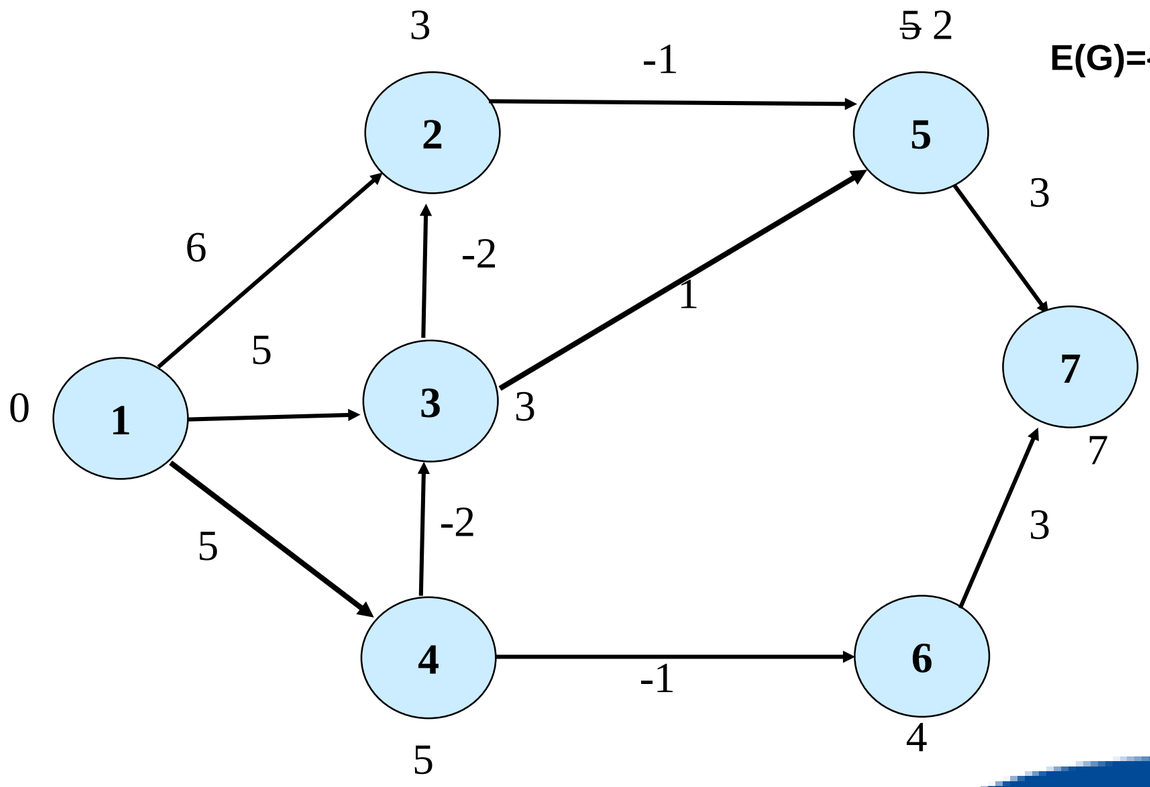
Practice Problem using Bellman-Ford Algorithm

Step 3: Step 2 will repeat for V-1 times

for each edge u-v

If dist[v] > dist[u] + weight of edge

then update dist[v] = dist[u] + weight of edge uv



Iteration 2

E(G)={(1,2), (1,3),(1,4), (2,5), (3,5), (3,2) ,(4,3),(4,6),(5,7),(6,7)}

~Dr Gaurav Kumar, Asst. Prof, CEA, GLAU

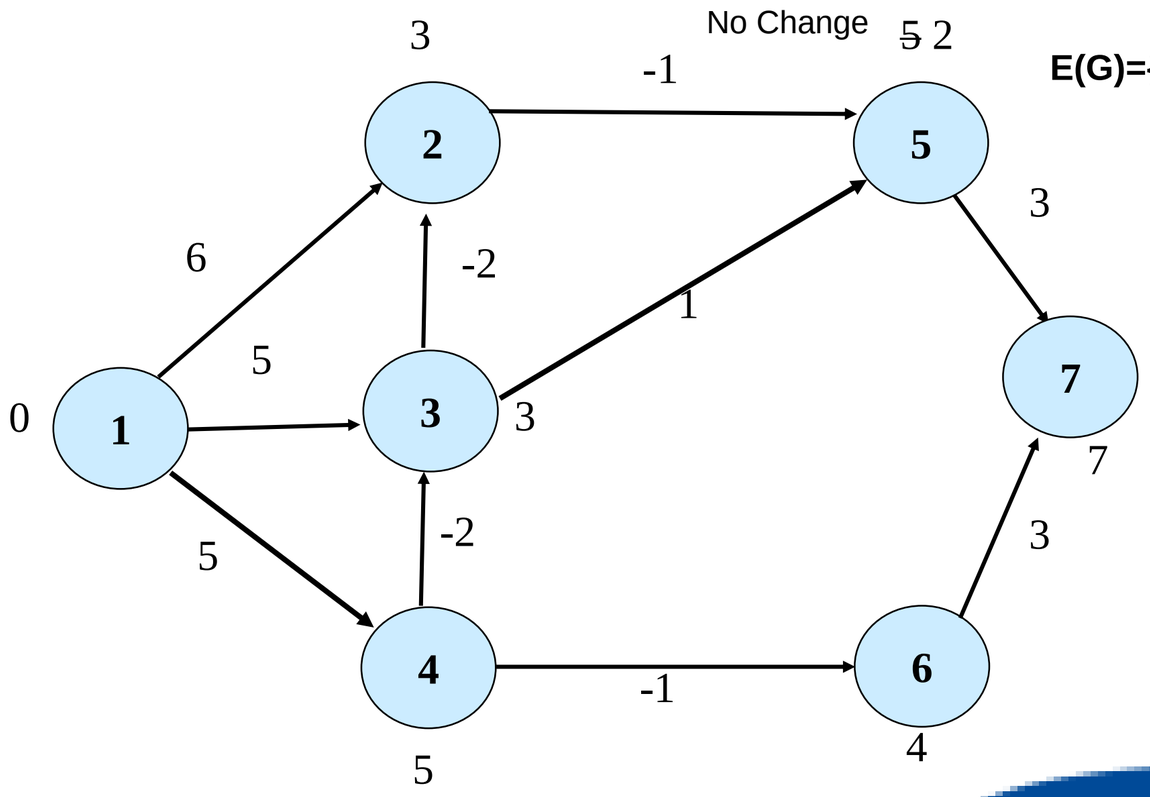
Practice Problem using Bellman-Ford Algorithm

Step 3: Step 2 will repeat for V-1 times

for each edge u-v

If dist[v] > dist[u] + weight of edge

then update dist[v] = dist[u] + weight of edge uv



Iteration 2

E(G)={(1,2), (1,3),(1,4), (2,5), (3,5), (3,2) ,(4,3),(4,6),(5,7),(6,7)}

~Dr Gaurav Kumar, Asst. Prof, CEA, GLAU

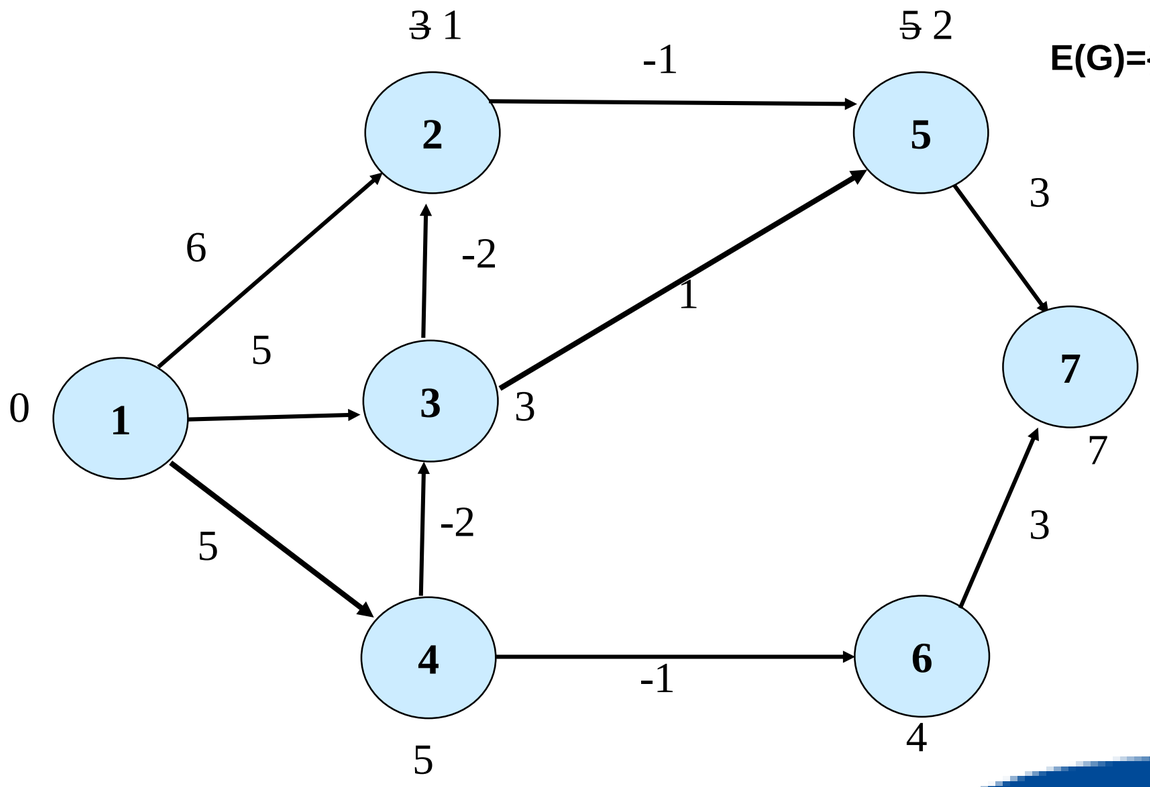
Practice Problem using Bellman-Ford Algorithm

Step 3: Step 2 will repeat for V-1 times

for each edge u-v

If dist[v] > dist[u] + weight of edge

then update dist[v] = dist[u] + weight of edge uv



Iteration 2

E(G)={(1,2), (1,3),(1,4), (2,5), (3,5), (3,2) ,(4,3),(4,6),(5,7),(6,7)}

~Dr Gaurav Kumar, Asst. Prof, CEA, GLAU

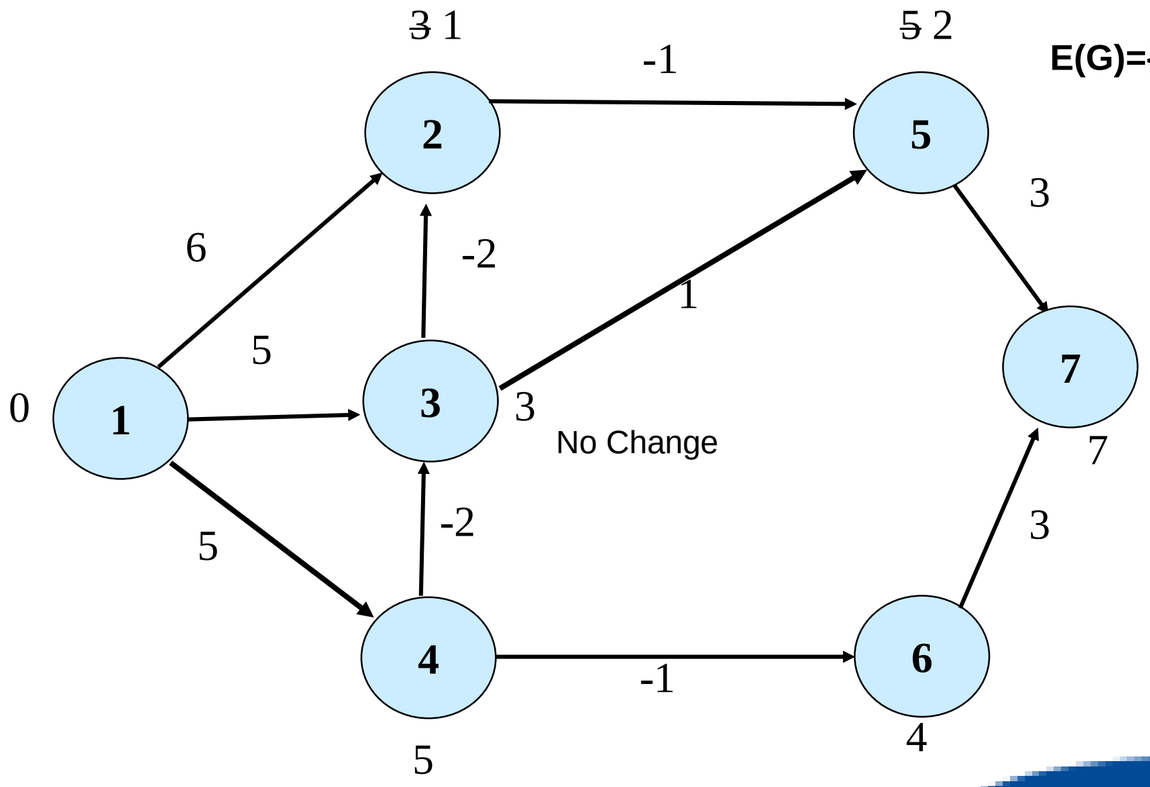
Practice Problem using Bellman-Ford Algorithm

Step 3: Step 2 will repeat for V-1 times

for each edge u-v

If dist[v] > dist[u] + weight of edge

then update dist[v] = dist[u] + weight of edge uv



Iteration 2

E(G)={(1,2), (1,3),(1,4), (2,5), (3,5), (3,2) ,(4,3),(4,6),(5,7),(6,7)}

~Dr Gaurav Kumar, Asst. Prof, CEA, GLAU

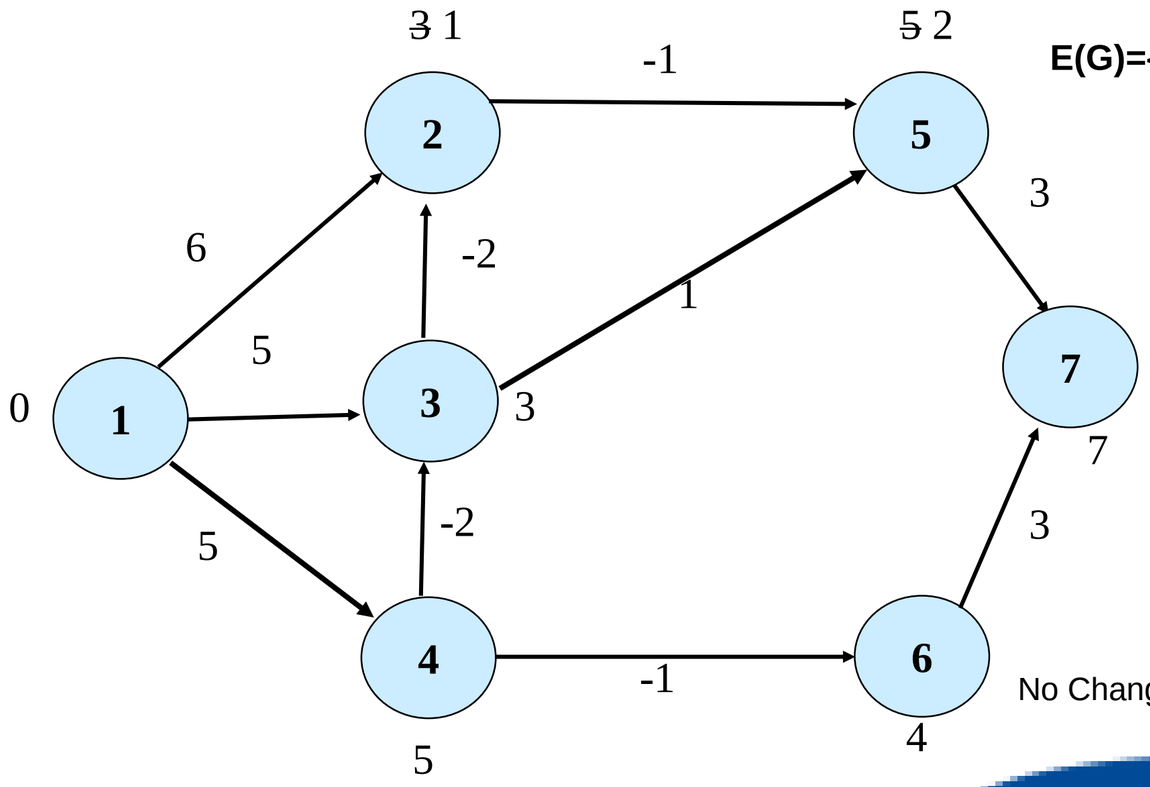
Practice Problem using Bellman-Ford Algorithm

Step 3: Step 2 will repeat for V-1 times

for each edge u-v

If dist[v] > dist[u] + weight of edge

then update dist[v] = dist[u] + weight of edge uv



Iteration 2

E(G)={(1,2), (1,3),(1,4), (2,5), (3,5), (3,2) ,(4,3),(4,6),(5,7),(6,7)}

~Dr Gaurav Kumar, Asst. Prof, CEA, GLAU

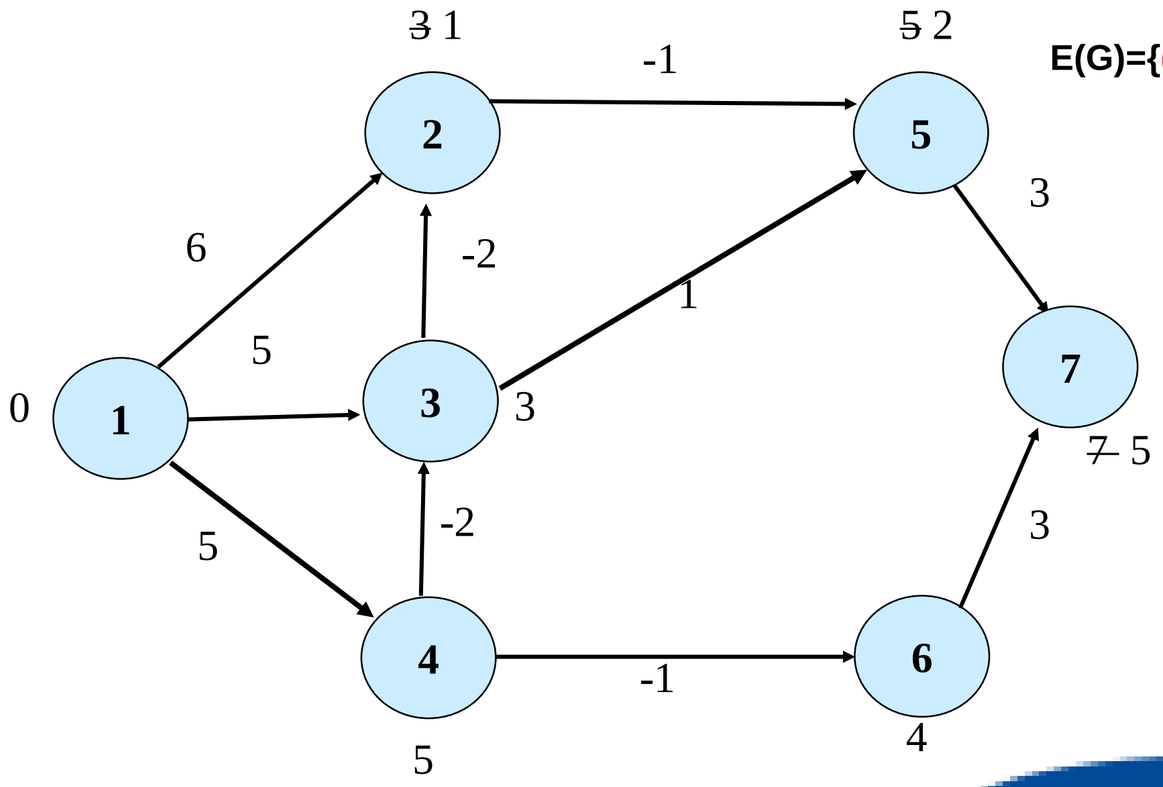
Practice Problem using Bellman-Ford Algorithm

Step 3: Step 2 will repeat for V-1 times

for each edge u-v

If dist[v] > dist[u] + weight of edge

then update dist[v] = dist[u] + weight of edge uv



Iteration 2

E(G)={(1,2), (1,3),(1,4), (2,5), (3,5), (3,2) ,(4,3),(4,6),(5,7),(6,7)}

~Dr Gaurav Kumar, Asst. Prof, CEA, GLAU

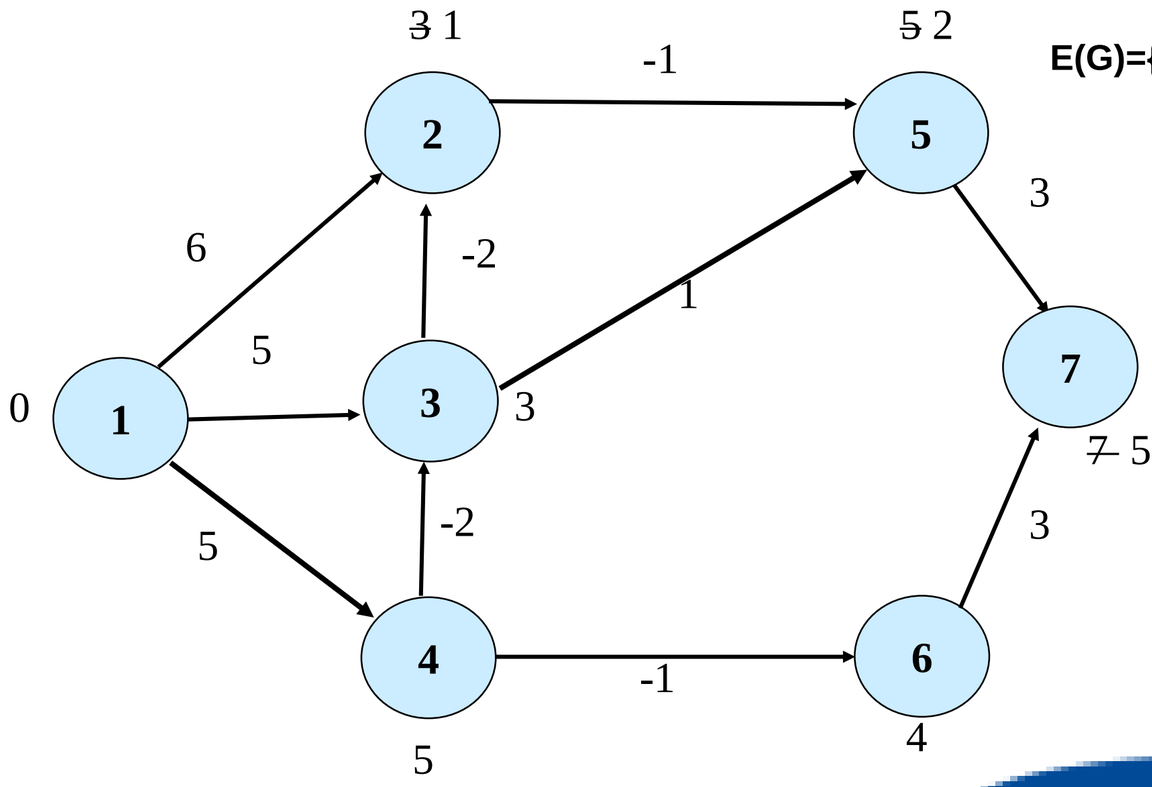
Practice Problem using Bellman-Ford Algorithm

Step 3: Step 2 will repeat for V-1 times

for each edge u-v

If dist[v] > dist[u] + weight of edge

then update dist[v] = dist[u] + weight of edge uv



Iteration 2

E(G)={(1,2), (1,3),(1,4), (2,5), (3,5), (3,2) ,(4,3),(4,6),(5,7),(6,7)}

~Dr Gaurav Kumar, Asst. Prof, CEA, GLAU

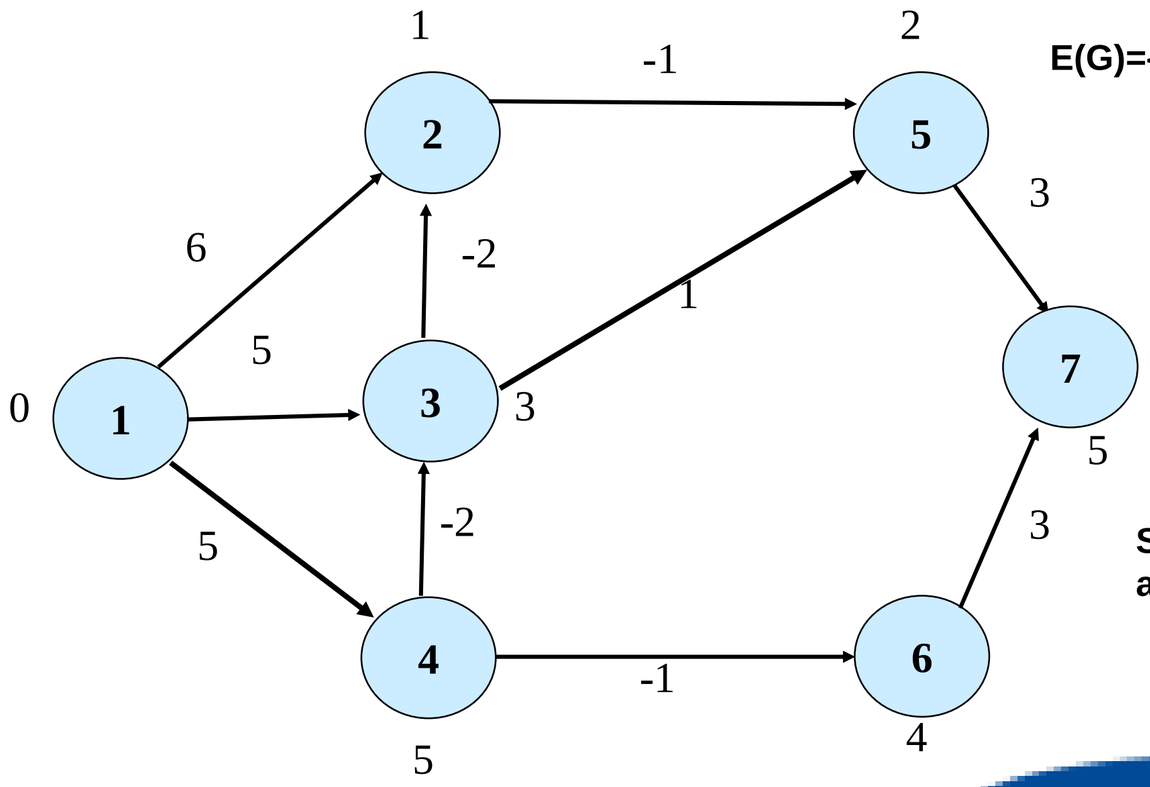
Practice Problem using Bellman-Ford Algorithm

Step 3: Step 2 will repeat for V-1 times

for each edge u-v

If dist[v] > dist[u] + weight of edge

then update dist[v] = dist[u] + weight of edge uv



Iteration 2

E(G)={(1,2), (1,3),(1,4), (2,5), (3,5), (3,2) ,(4,3),(4,6),(5,7),(6,7)}

Shortest Spanning Tree after Iteration 2

~Dr Gaurav Kumar, Asst. Prof, CEA, GLAU

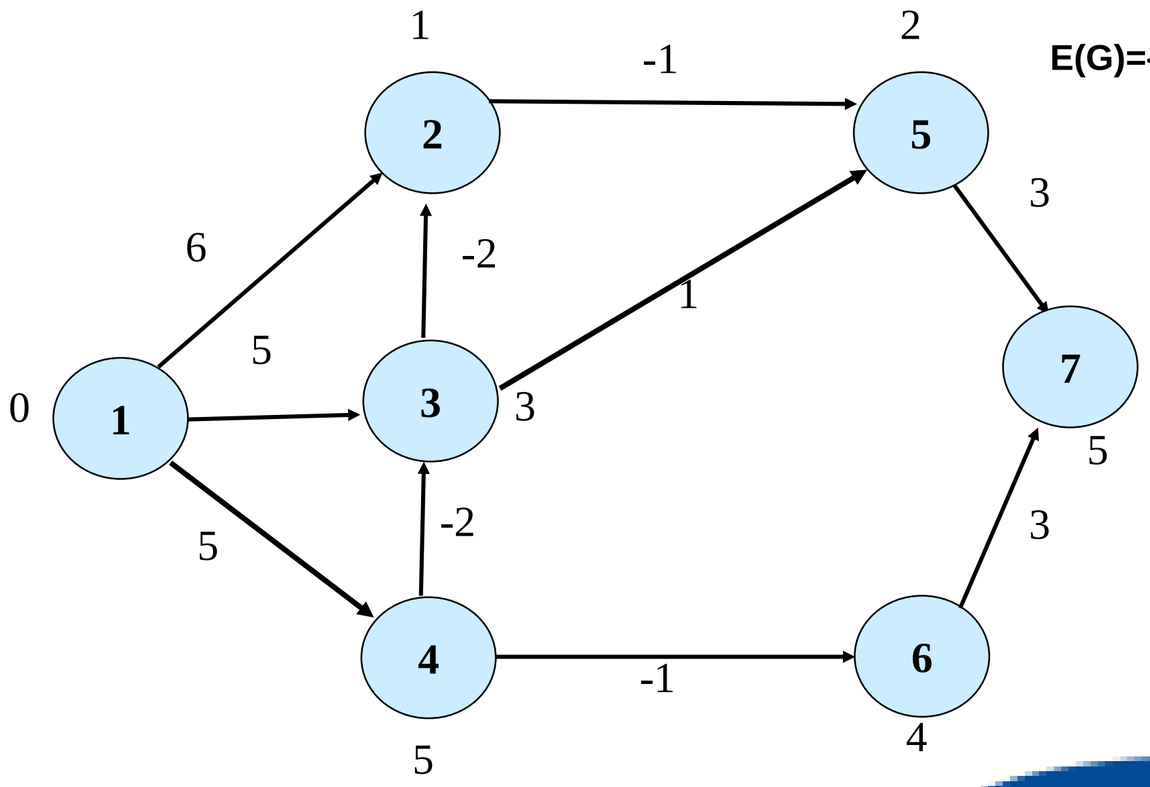
Practice Problem using Bellman-Ford Algorithm

Step 3: Step 2 will repeat for V-1 times

for each edge u-v

If dist[v] > dist[u] + weight of edge

then update dist[v] = dist[u] + weight of edge uv



Iteration 3

E(G)={(1,2), (1,3),(1,4), (2,5), (3,5), (3,2) ,(4,3),(4,6),(5,7),(6,7)}

~Dr Gaurav Kumar, Asst. Prof, CEA, GLAU

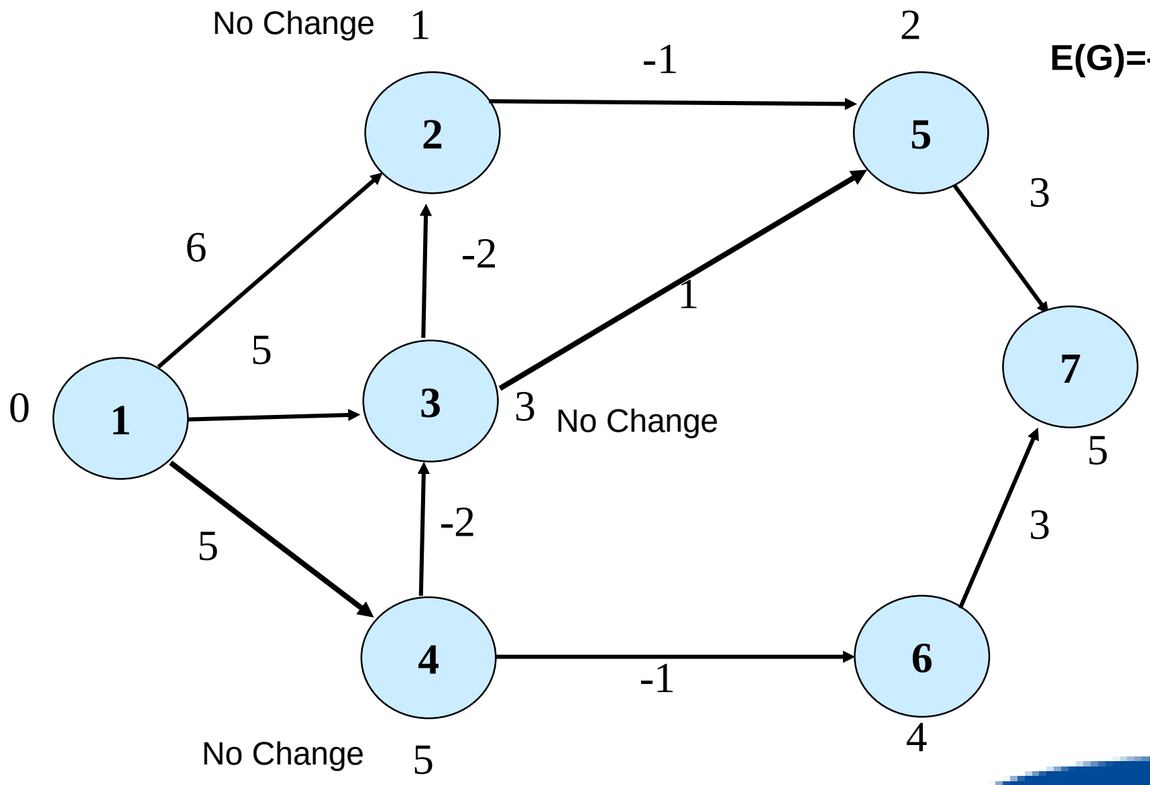
Practice Problem using Bellman-Ford Algorithm

Step 3: Step 2 will repeat for V-1 times

for each edge u-v

If dist[v] > dist[u] + weight of edge

then update dist[v] = dist[u] + weight of edge uv



Iteration 3

E(G)={(1,2), (1,3),(1,4), (2,5), (3,5), (3,2) ,(4,3),(4,6),(5,7),(6,7)}

~Dr Gaurav Kumar, Asst. Prof, CEA, GLAU

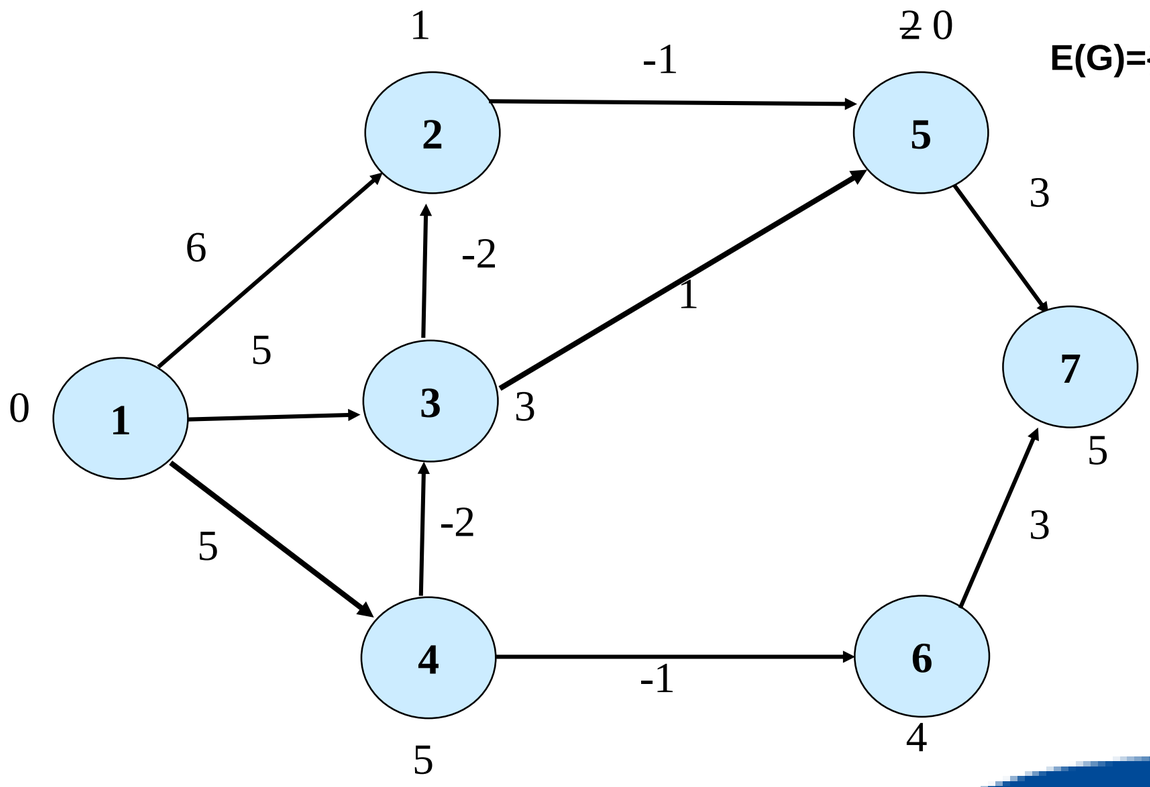
Practice Problem using Bellman-Ford Algorithm

Step 3: Step 2 will repeat for V-1 times

for each edge u-v

If dist[v] > dist[u] + weight of edge

then update dist[v] = dist[u] + weight of edge uv



Iteration 3

E(G)={(1,2), (1,3),(1,4), (2,5), (3,5), (3,2) ,(4,3),(4,6),(5,7),(6,7)}

~Dr Gaurav Kumar, Asst. Prof, CEA, GLAU

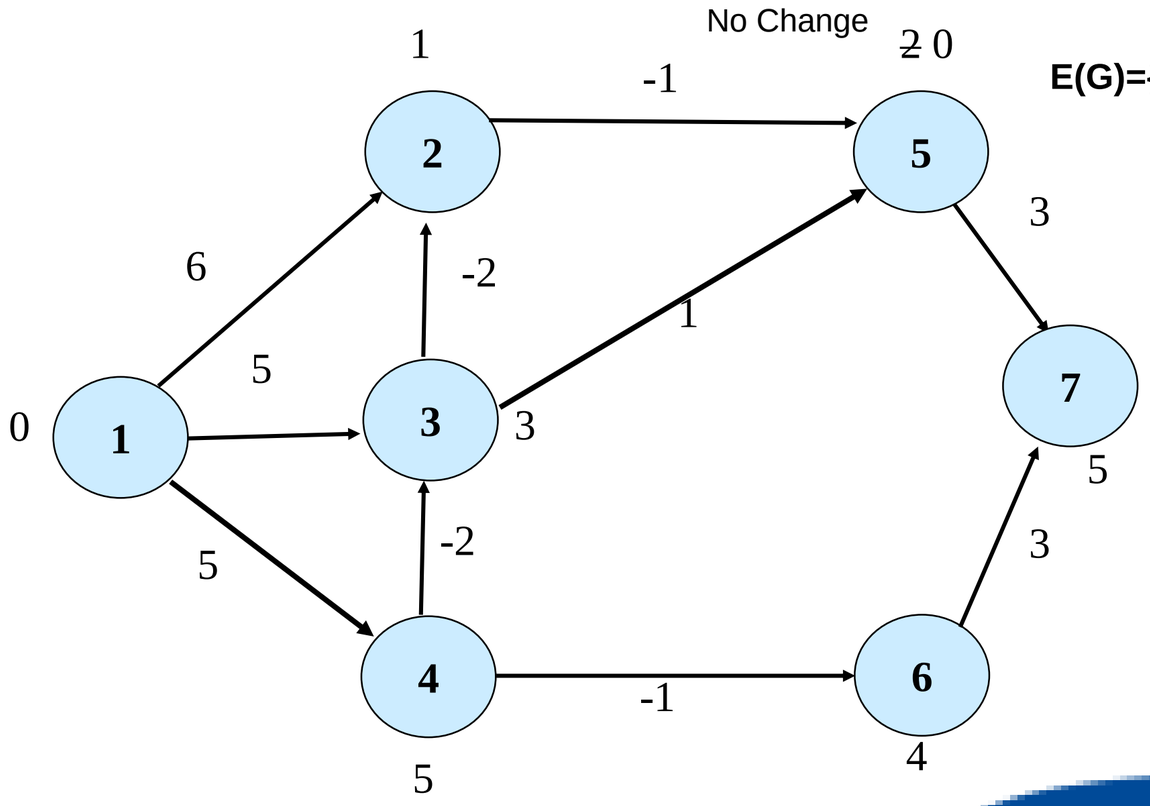
Practice Problem using Bellman-Ford Algorithm

Step 3: Step 2 will repeat for V-1 times

for each edge u-v

If dist[v] > dist[u] + weight of edge

then update dist[v] = dist[u] + weight of edge uv



Iteration 3

E(G)={(1,2), (1,3),(1,4), (2,5), (3,5), (3,2) ,(4,3),(4,6),(5,7),(6,7)}

~Dr Gaurav Kumar, Asst. Prof, CEA, GLAU

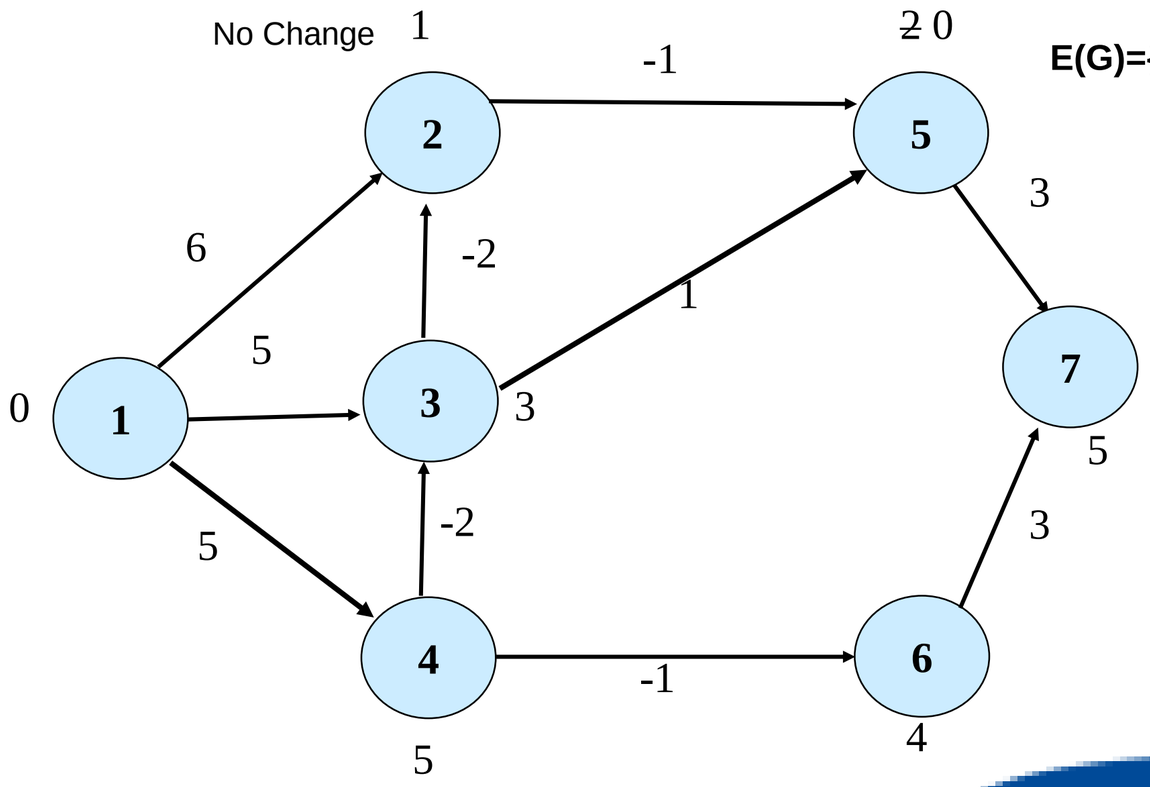
Practice Problem using Bellman-Ford Algorithm

Step 3: Step 2 will repeat for V-1 times

for each edge u-v

If dist[v] > dist[u] + weight of edge

then update dist[v] = dist[u] + weight of edge uv



Iteration 3

E(G)={(1,2), (1,3),(1,4), (2,5), (3,5), (3,2) ,(4,3),(4,6),(5,7),(6,7)}

~Dr Gaurav Kumar, Asst. Prof, CEA, GLAU

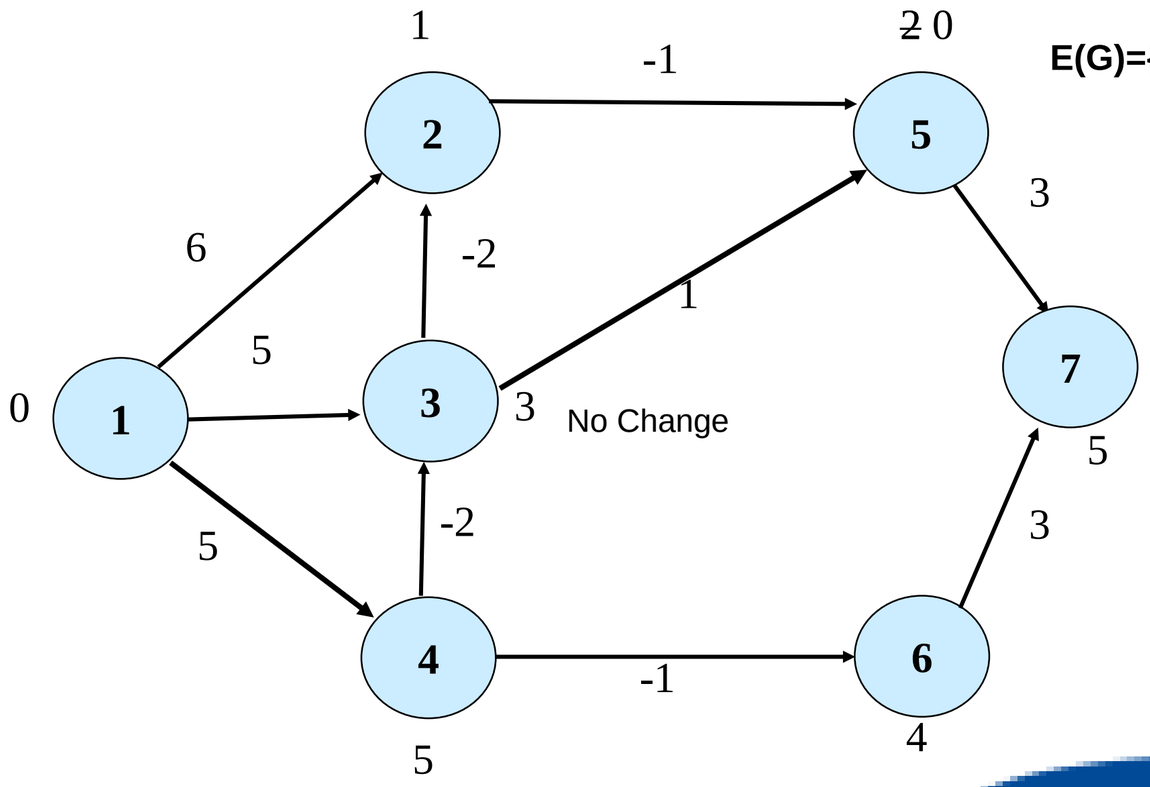
Practice Problem using Bellman-Ford Algorithm

Step 3: Step 2 will repeat for V-1 times

for each edge u-v

If dist[v] > dist[u] + weight of edge

then update dist[v] = dist[u] + weight of edge uv



Iteration 3

E(G)={(1,2), (1,3),(1,4), (2,5), (3,5), (3,2) ,(4,3),(4,6),(5,7),(6,7)}

~Dr Gaurav Kumar, Asst. Prof, CEA, GLAU

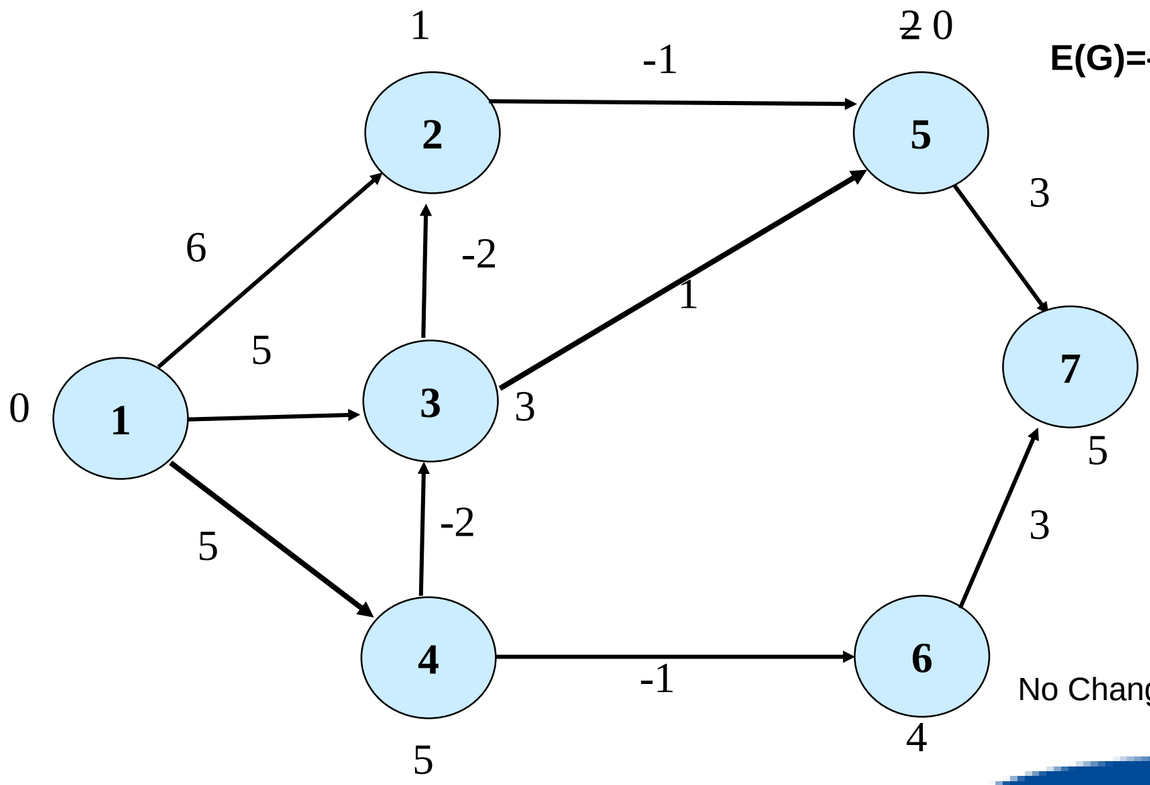
Practice Problem using Bellman-Ford Algorithm

Step 3: Step 2 will repeat for V-1 times

for each edge u-v

If dist[v] > dist[u] + weight of edge

then update dist[v] = dist[u] + weight of edge uv



Iteration 3

E(G)={(1,2), (1,3),(1,4), (2,5), (3,5), (3,2) ,(4,3),(4,6),(5,7),(6,7)}

~Dr Gaurav Kumar, Asst. Prof, CEA, GLAU

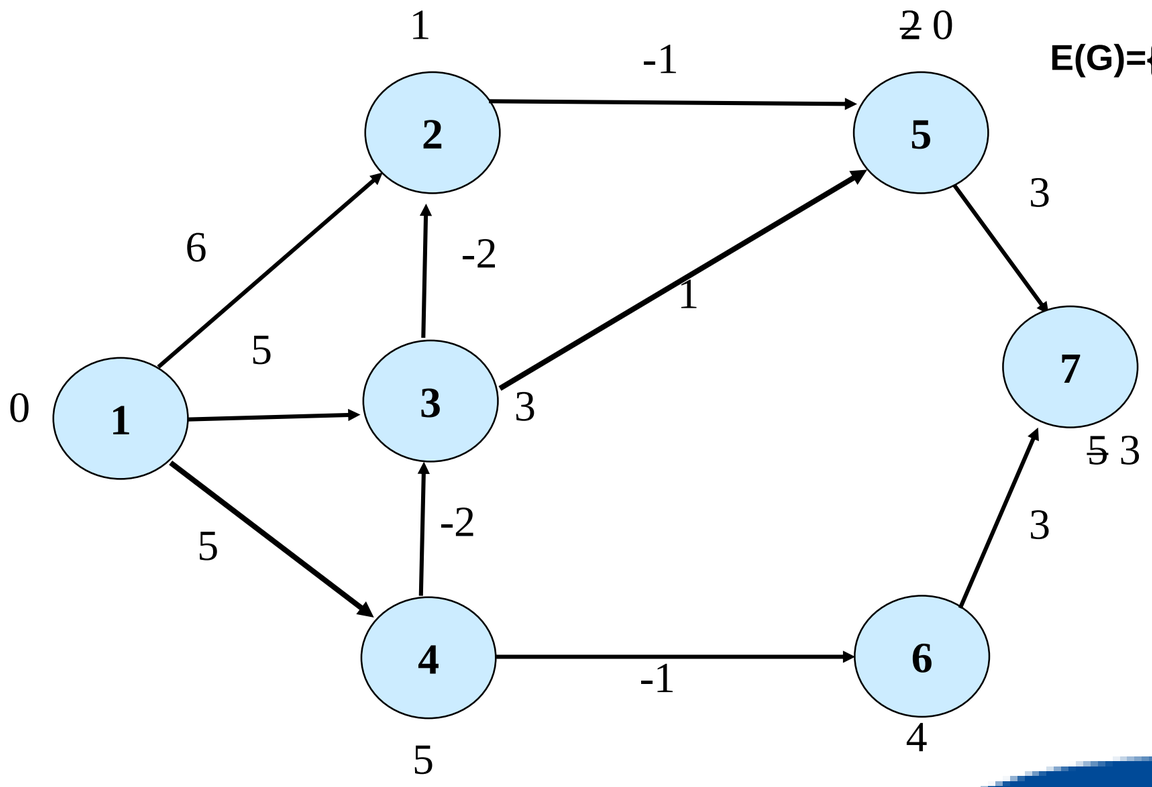
Practice Problem using Bellman-Ford Algorithm

Step 3: Step 2 will repeat for V-1 times

for each edge u-v

If dist[v] > dist[u] + weight of edge

then update dist[v] = dist[u] + weight of edge uv



Iteration 3

E(G)={(1,2), (1,3),(1,4), (2,5), (3,5), (3,2) ,(4,3),(4,6),(5,7),(6,7)}

~Dr Gaurav Kumar, Asst. Prof, CEA, GLAU

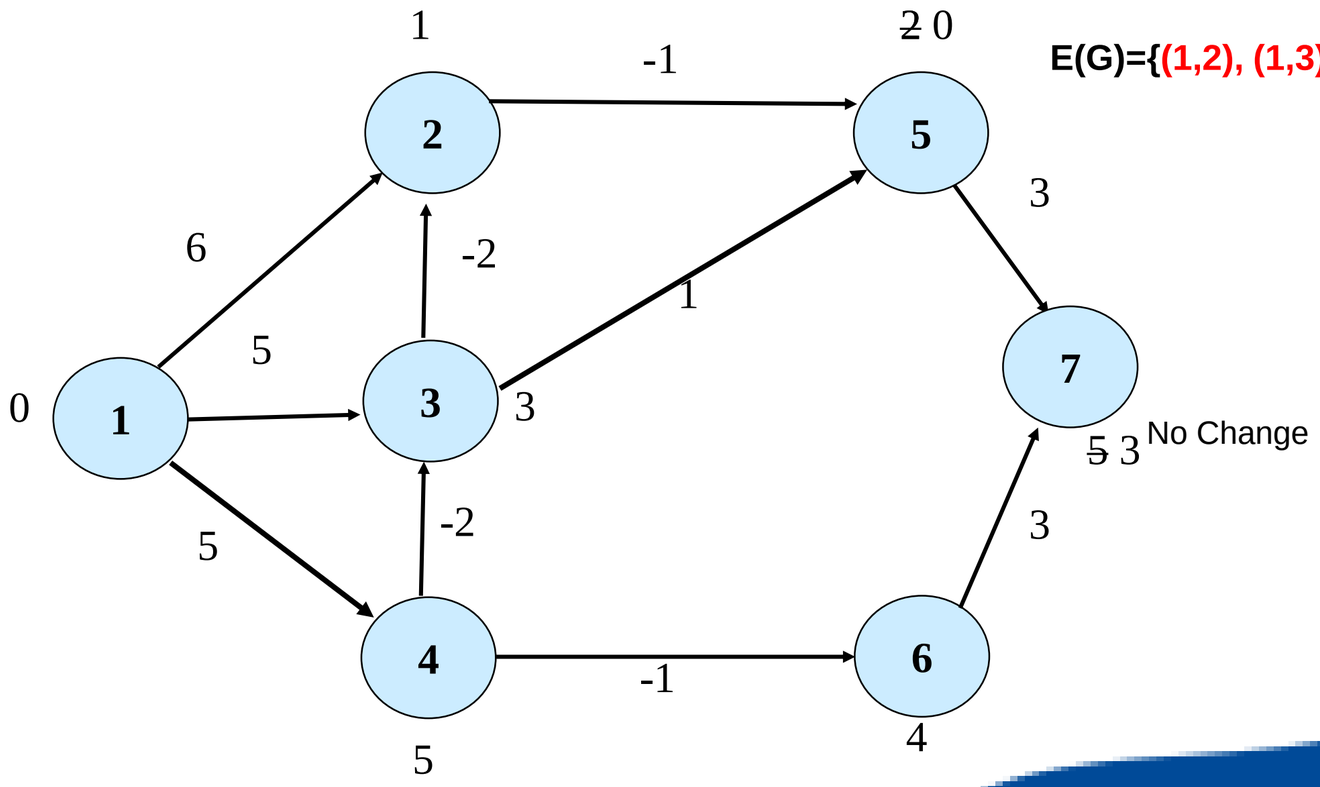
Practice Problem using Bellman-Ford Algorithm

Step 3: Step 2 will repeat for V-1 times

for each edge u-v

If dist[v] > dist[u] + weight of edge

then update dist[v] = dist[u] + weight of edge uv



Iteration 3

E(G)={(1,2), (1,3),(1,4), (2,5), (3,5), (3,2) ,(4,3),(4,6),(5,7),(6,7)}

~Dr Gaurav Kumar, Asst. Prof, CEA, GLAU

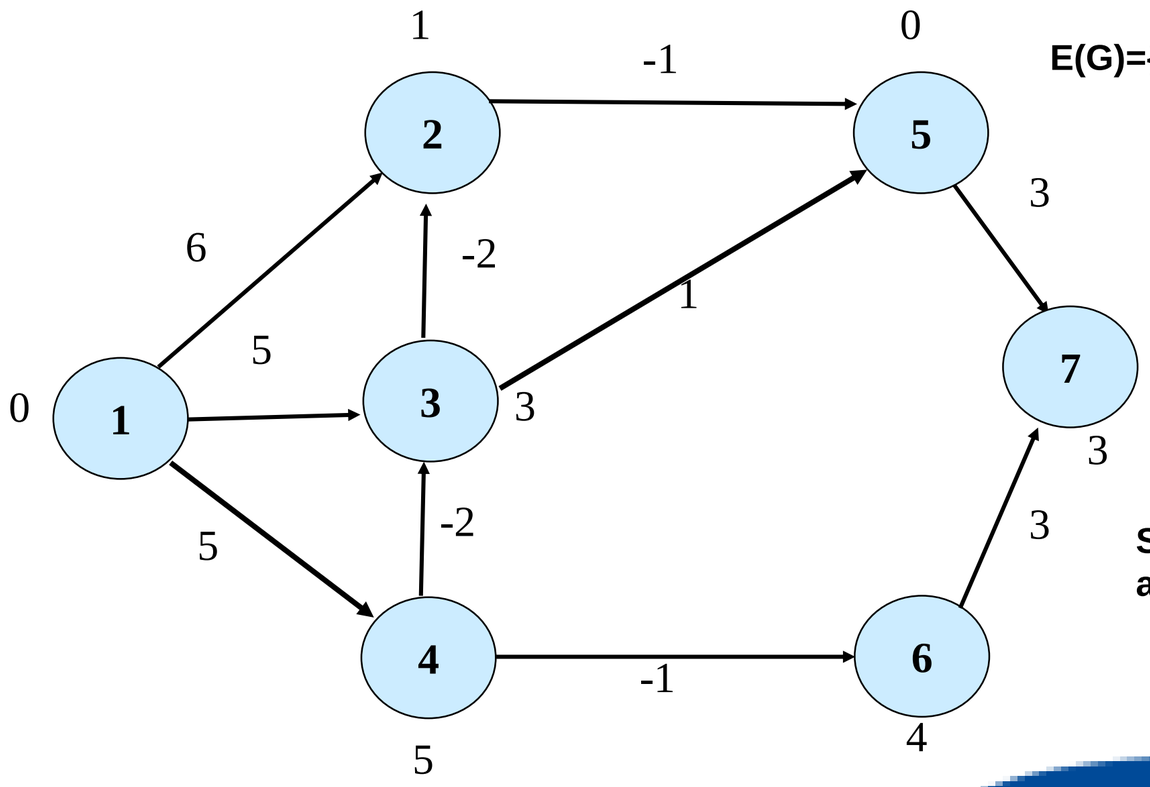
Practice Problem using Bellman-Ford Algorithm

Step 3: Step 2 will repeat for V-1 times

for each edge u-v

If dist[v] > dist[u] + weight of edge

then update dist[v] = dist[u] + weight of edge uv



Iteration 3

E(G)={(1,2), (1,3),(1,4), (2,5), (3,5), (3,2) ,(4,3),(4,6),(5,7),(6,7)}

Shortest Spanning Tree after Iteration 3

~Dr Gaurav Kumar, Asst. Prof, CEA, GLAU

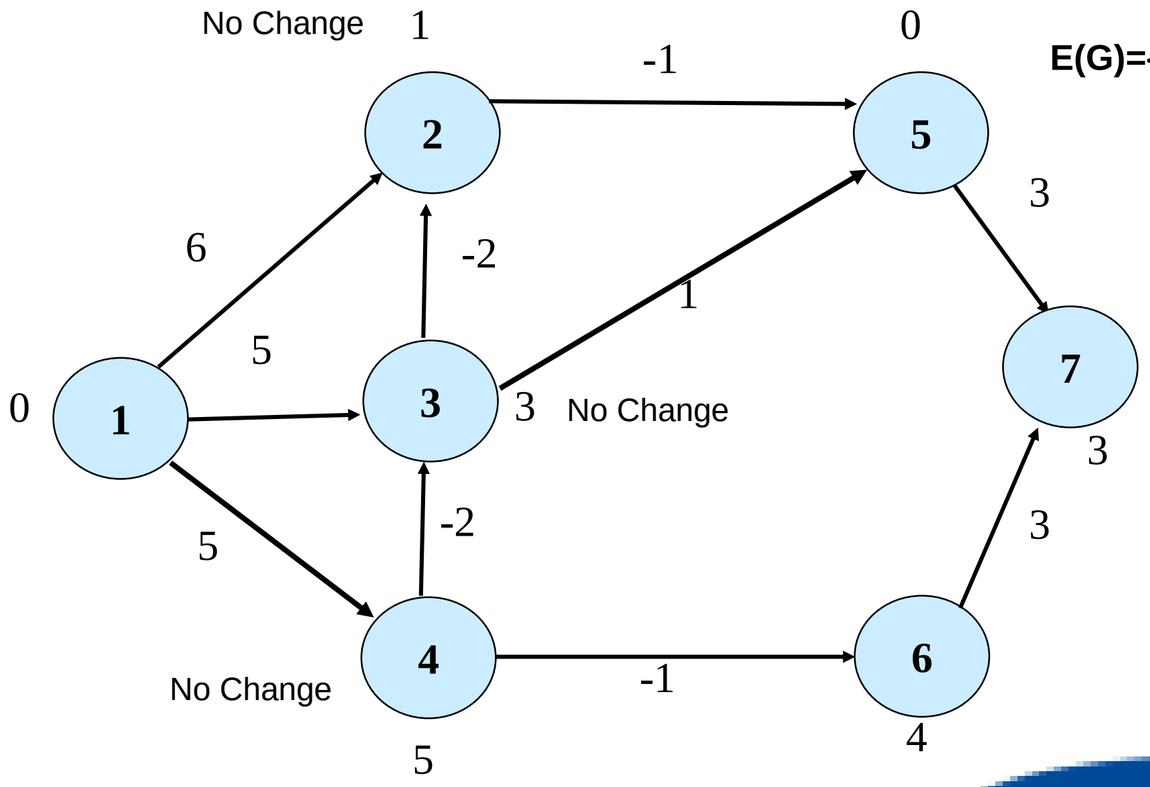
Practice Problem using Bellman-Ford Algorithm

Step 3: Step 2 will repeat for V-1 times

for each edge u-v

If dist[v] > dist[u] + weight of edge

then update dist[v] = dist[u] + weight of edge uv



Iteration 4

E(G)={(1,2), (1,3),(1,4), (2,5), (3,5), (3,2) ,(4,3),(4,6),(5,7),(6,7)}

~Dr Gaurav Kumar, Asst. Prof, CEA, GLAU

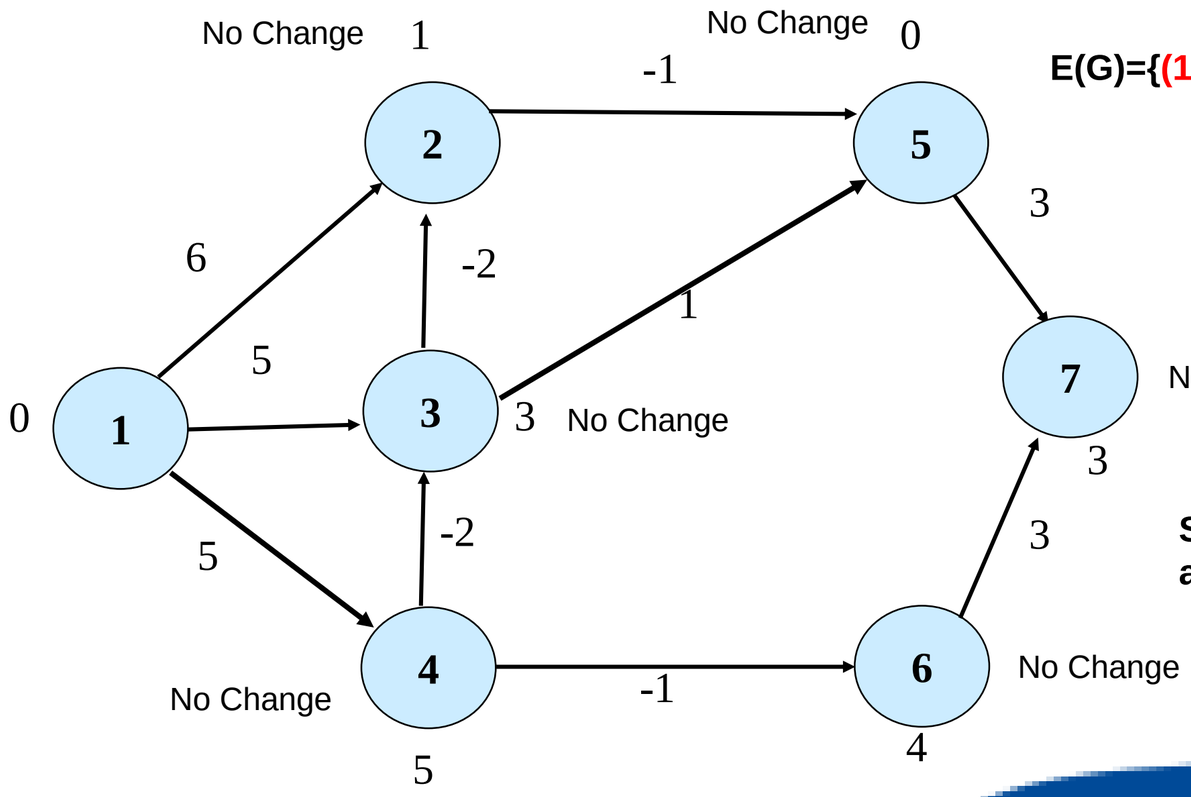
Practice Problem using Bellman-Ford Algorithm

Step 3: Step 2 will repeat for V-1 times

for each edge u-v

If dist[v] > dist[u] + weight of edge

then update dist[v] = dist[u] + weight of edge uv



Iteration 4

E(G)={(1,2), (1,3),(1,4), (2,5), (3,5), (3,2) ,(4,3),(4,6),(5,7),(6,7)}

Shortest Spanning Tree after Iteration 4

~Dr Gaurav Kumar, Asst. Prof, CEA, GLAU

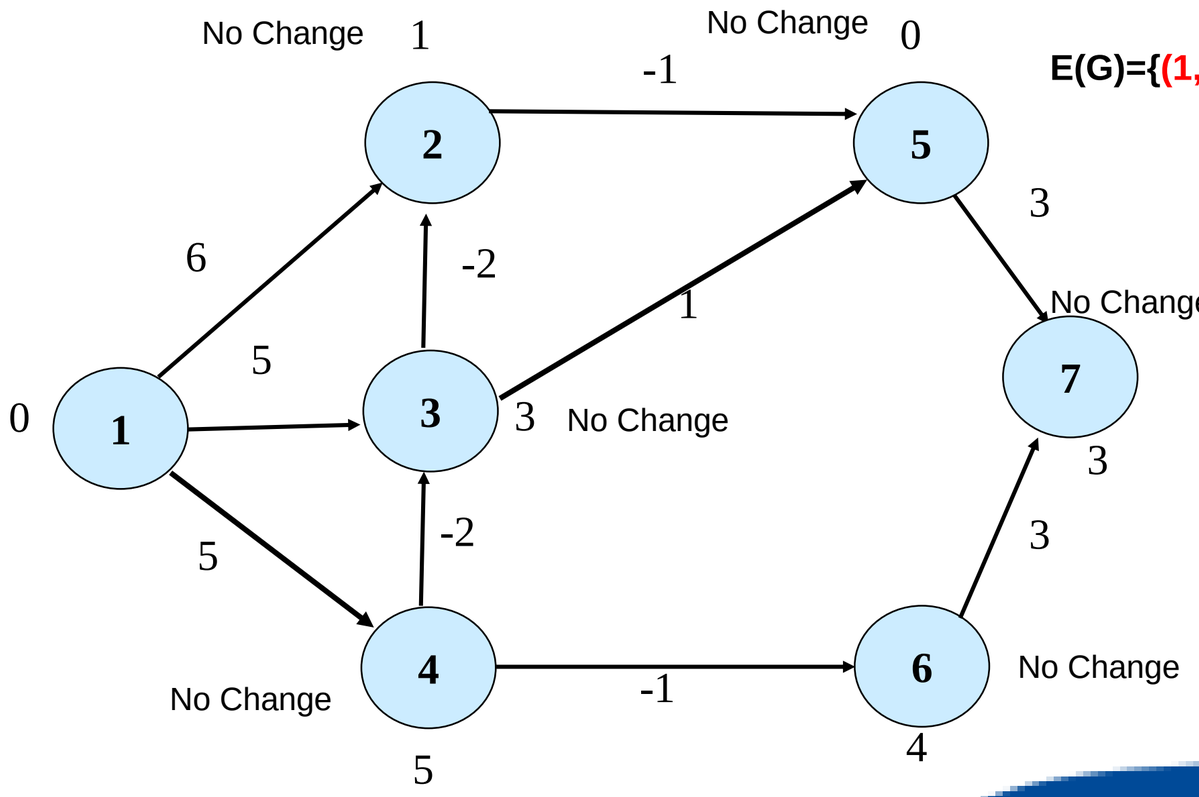
Practice Problem using Bellman-Ford Algorithm

Step 3: Step 2 will repeat for V-1 times

for each edge u-v

If dist[v] > dist[u] + weight of edge

then update dist[v] = dist[u] + weight of edge uv



Iteration 6

E(G)={(1,2), (1,3),(1,4), (2,5), (3,5), (3,2) ,(4,3),(4,6),(5,7),(6,7)}

Shortest Spanning Tree after Iteration 4

Shortest Spanning Tree after Iteration 5, 6 will remain same

~Dr Gaurav Kumar, Asst. Prof, CEA, GLAU

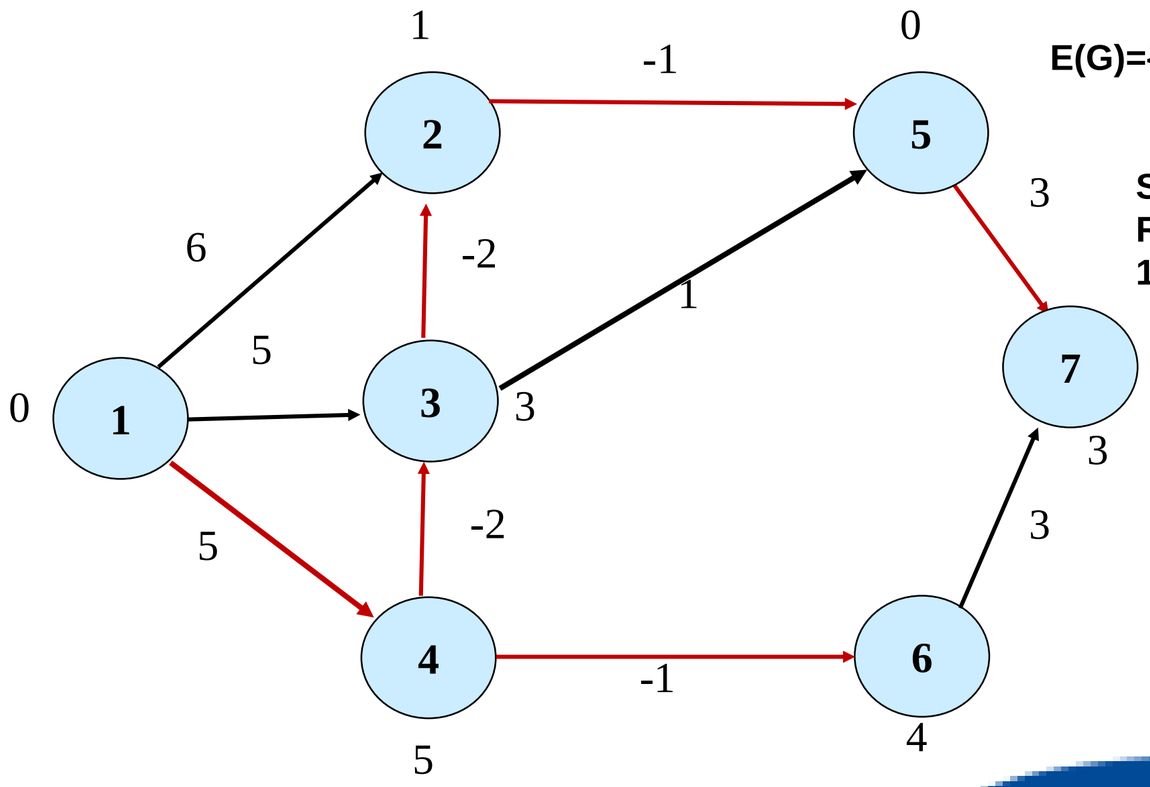
Practice Problem using Bellman-Ford Algorithm

Step 3: Step 2 will repeat for V-1 times

for each edge u-v

If dist[v] > dist[u] + weight of edge

then update dist[v] = dist[u] + weight of edge uv



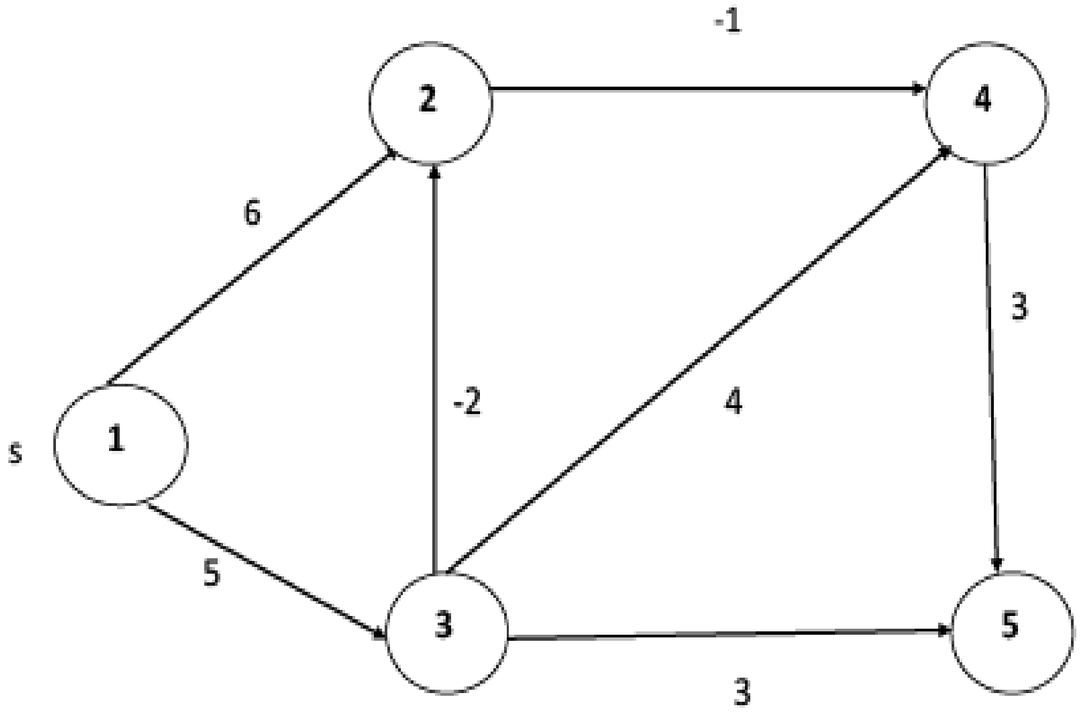
E(G)={(1,2), (1,3),(1,4), (2,5), (3,5), (3,2) ,(4,3),(4,6),(5,7),(6,7)}

Single Source Shortest Past in Red Color Spanning Tree
1, 4, 3, 2, 5, 6, 7

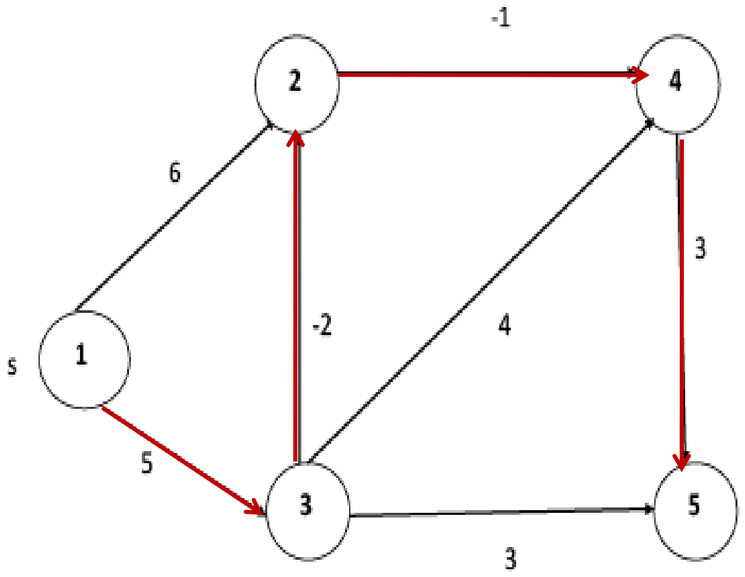
~Dr Gaurav Kumar, Asst. Prof, CEA, GLAU

Self Practice Problem using Bellman-Ford Algorithm

Find the shortest path from a vertex “A” to all other vertices of a weighted graph



Self Practice Problem using Bellman-Ford Algorithm



Vertices

No of Edges Traversed	1	2	3	4	5
1	0	6	5	∞	∞
2	0	3	5	5	8
3	0	3	5	2	8
4	0	3	5	2	5

~Dr Gaurav Kumar, Asst. Prof, CEA, GLAU

Time Complexity of Bellman-Ford Algorithm

BELLMAN -FORD (G, w, s)

1. INITIALIZE - SINGLE - SOURCE (G, s) → **V**
2. for i ← 1 to |V[G]| - 1 // Loop will repeat V-1 times → **V-1**
3. do **for each edge (u, v) ∈ E [G]**
4. do **RELAX (u, v, w)** → **E**

   → **(V-1)*E**
5. for each edge (u, v) ∈ E [G] → **E**
6. do if d [v] > d [u] + w (u, v)
7. then return FALSE //Cycle Found
8. return TRUE. //Shortest Path

Total Time= V+ (V-1)*E + E +E
TC= O(VE) or
for complete graph, it is $O(V^3)$

Happy Learning!

If you have any doubts, or queries , can be discussed in the C-11, Room 310, AB-1.

or share it on WhatsApp 8586968801

if there is any suggestion or feedback about slides, please write it on gaurav.kumar@gla.ac.in

~Dr Gaurav Kumar, Asst. Prof, CEA, GLAU