Unit 3
Greedy Algorithms

Prof. Fenil Khatiwala
Department of Computer Engineering
CGPIT, UTU

Outline

- General Characteristics of greedy algorithms
- Elements of Greedy Strategy
- Make change Problem
- Minimum Spanning trees (Kruskal’s algorithm, Prim’s algorithm)
- The Knapsack Problem
- Job Scheduling Problem

Characteristics of Greedy Algorithms

Characteristics of Greedy Algorithms

- Greedy algorithms are characterized by the following features.
  1. Greedy approach forms **a set or list of candidates *C***.
  2. Once a candidate is **selected in the solution**, it is there forever: once a candidate is **excluded from the solution**, it is never reconsidered.
  3. To construct the solution in an optimal way, Greedy Algorithm maintains **two sets**.
  4. One set contains candidates that have already been **considered and chosen**, while the other set contains candidates that have been **considered but rejected**.

Elements of Greedy Strategy

The greedy algorithm consists of **four functions**.

1. **Solution Function**:- A function that checks whether chosen set of items provides a solution.
2. **Feasible Function**:- A function that checks the feasibility of a set.
3. **Selection Function**:- The selection function tells which of the candidates is the most promising.
4. **Objective Function**:- An objective function, which does not appear explicitly, but gives the value of a solution.

Making Change Problem

Making Change Problem

- Suppose following coins are available **with unlimited quantity**:
  1. ₹ 10
  2. ₹ 5
  3. ₹ 2
  4. ₹ 1
  5. 50 paisa

- Our problem is to devise an algorithm for paying a given amount to a customer using **the smallest possible number of coins**.

Making Change Problem

- If suppose, we need to pay an amount of ₹ 28/- using the available coins.
- Here we have a candidate (coins) set $\boldsymbol{C} = \{\mathbf{10}, \mathbf{5}, \mathbf{2}, \mathbf{1}, \mathbf{.5}\}$
- The greedy solution is,

Selected coins are,

coins	quantity
10	2
5	1
2	1
1	1

Amount

28

Total required coins = 5

Selected coins = {10, 5, 2, 1}

Making Change Problem

```
# Input: C = {10, 5, 2, 1, 0.5} //C is a candidate set
# Output: S: set of selected coins
Function make-change(n): set of coins
S ← Ø {S is a set that will hold the solution}
sum ← 0 {sum of the items in solution set S}
while sum ≠ n do
   x ← the largest item in C such that sum + x ≤ n
   if there is no such item then
      return "no solution found"
   S ← S U {a coin of value x)
   sum ← sum + x
return S
```

Making Change – The Greedy Property

- The algorithm is **greedy** because,
  - At every step it chooses **the largest available coin**, without worrying whether this will prove to be a **correct** decision later.
  - It **never changes** the decision, i.e., once a coin has been included in the solution, it is there **forever**.
- Examples: Some coin denominations 50, 20, 10, 5, 1 are available.
  1. How many minimum coins required to make change for 37 cents?

     5
  2. How many minimum coins required to make change for 91 cents?

     4
  3. Denominations: $d_1=6$, $d_2=4$, $d_3=1$. Make a change of ₹ 8. ~~3~~

The minimum coins required are 2

Fractional Knapsack Problem

Fractional Knapsack Problem

- We are given n objects and a knapsack.
- Object $\boldsymbol{i}$ has a positive weight $\boldsymbol{w_i}$ and a positive value $\boldsymbol{v_i}$ for $\boldsymbol{i} = \mathbf{1}, \mathbf{2} \dots \boldsymbol{n}$.
- The knapsack can carry a weight not exceeding $\boldsymbol{W}$.
- Our aim is to fill the knapsack in a way that **maximizes** the value of the included objects, while respecting the capacity constraint.

Fractional Knapsack Problem

- In a fractional knapsack problem, we assume that the objects **can be broken into smaller pieces**.
- So we may decide to carry only a fraction x_i of object i, where $0 \le x_i \le 1$.
- In this case, object i contribute $x_i w_i$ to the total weight in the knapsack, and $x_i v_i$ to the value of the load.
- Symbolic Representation of the problem can be given as follows:

$$\textbf{maximize } \boxed{\sum_{i=1}^{n} x_i v_i} \textbf{ subject to } \underline{\sum_{i=1}^{n} x_i w_i \le W}$$

$$\text{Where, } v_i > 0,\ w_i > 0 \text{ and } 0 \le x_i \le 1 \text{ for } 1 \le i \le n.$$

Fractional Knapsack Problem – Example

- Example: We are given 5 objects and the weight carrying capacity of knapsack is $W = 100.$
- For each object, weight w_i and value v_i are given in the following table.

Object i	1	2	3	4	5
v_i	20	30	66	40	60
w_i	10	20	30	40	50

- Fill the knapsack with given objects such that the total value of knapsack is **maximized.**

Greedy Solution

- Three **selection functions** can be defined as,
    1. Sort the items in **descending order of their values** and select the items till weight criteria is satisfied.
    2. Sort the items in **ascending order of their weight** and select the items till weight criteria is satisfied.
    3. To calculate the **ratio value/weight** for each item and sort the item on basis of this ratio. Then take the item with the highest ratio and add it.

Fractional Knapsack Problem – Solution

Object i	1	2	3	4	5
v_i	20	30	66	40	60
w_i	10	20	30	40	50

Selection	Objects					Value
	1	2	3	4	5	
Max v_i						
Min w_i						
Max $^{v_i}/_{w_i}$						

Weight Capacity 100

30	50	20	
10	20	30	40
30	10	20	40

Profit = 66 + 20 + 30 + 48 = 164

Fractional Knapsack Problem – Example

- Example: We are given 7 objects and the weight carrying capacity of knapsack is $W = 15.$
- For each object, weight w_i and value v_i are given in the following table.

Object i	1	2	3	4	5	6	7
v_i	10	5	15	7	6	18	3
w_i	2	3	5	7	1	4	1

- Fill the knapsack with given objects such that the total value of knapsack is **maximized.**

Fractional Knapsack Problem – Solution

Object i	1	2	3	4	5	6	7
v_i	10	5	15	7	6	18	3
w_i	2	3	5	7	1	4	1

Selection	**Objects**							**Value**
	1	**2**	**3**	**4**	**5**	**6**	**7**	
Max v_i								
Min w_i								
Max v_i/w_i								

Weight Capacity 15

4	5	2	4		
1	1	2	3	4	4
1	2	4	5	1	2

Profit = 6 + 10 + 18 + 15 + 3 + 3.3 = 55.3

Fractional Knapsack Problem – Algorithm

```
Algorithm: Greedy-Fractional-Knapsack (w[1..n], p[1..n], W)
for i = 1 to n do
      x[i] ← 0
      weight ← 0
While weight < W do
      i ← the best remaining object
      if weight + w[i] ≤ W then
            x[i] ←1
            weight ← weight + w[i]
      else
            x[i] ← (W - weight) / w[i]
            weight ← W
return x
```

(100 – 60) / 50 = 0.8

Examples

1. Consider Knapsack capacity $W = 50$, $w = (10, 20, 40)$ and $v = (60, 80, 100)$ find the maximum profit using greedy approach.
2. Consider Knapsack capacity $W = 10$, $w = (4, 8, 2, 6, 1)$ and $v = (12, 32, 40, 30, 50)$. Find the maximum profit using greedy approach.

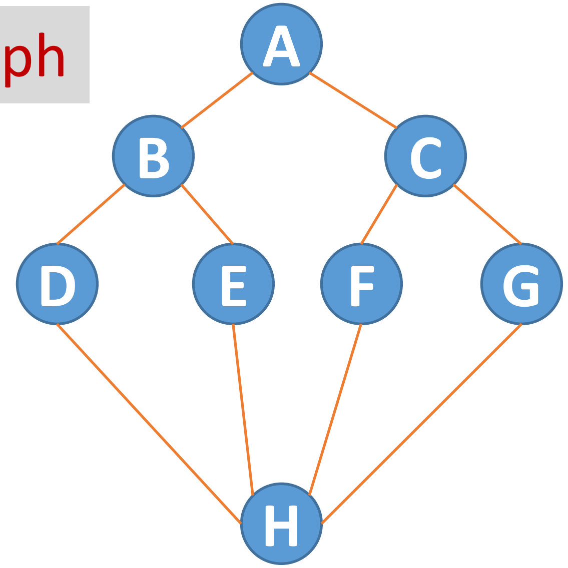
Minimum Spanning Tree

Minimum Spanning Tree – MST

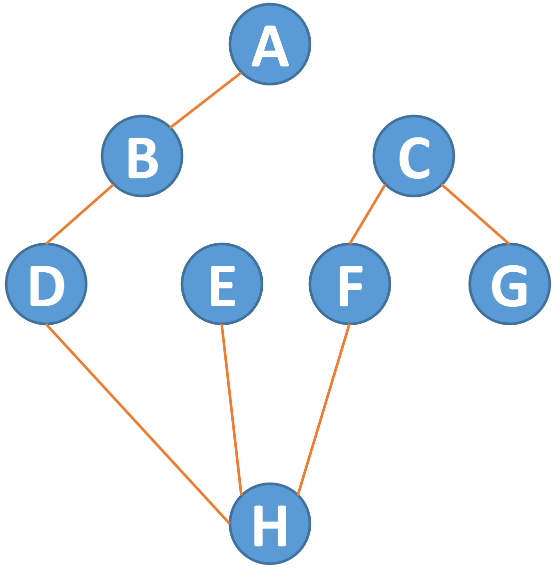
- Let $G = \langle N, A \rangle$ be a **connected, undirected graph** where,
  1. N is the set of nodes and
  2. A is the set of edges.
- Each edge has a given **positive length or weight**.
- A spanning tree of a graph G **is a sub-graph** which is basically a tree and it contains all the vertices of G but does not contain cycle.
- A minimum spanning tree (MST) of a **weighted connected graph** G is a spanning tree with minimum or smallest weight of edges.
- Two Algorithms for **constructing** minimum spanning tree are,
  1. Kruskal's Algorithm
  2. Prim's Algorithm

Spanning Tree

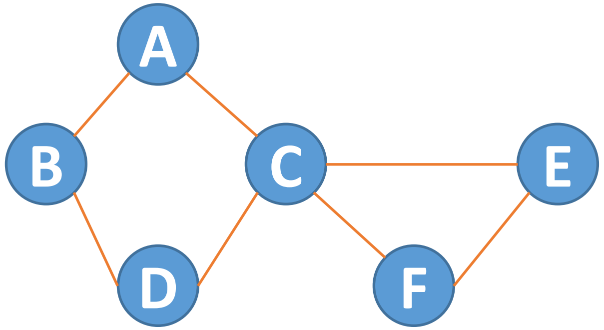
Graph



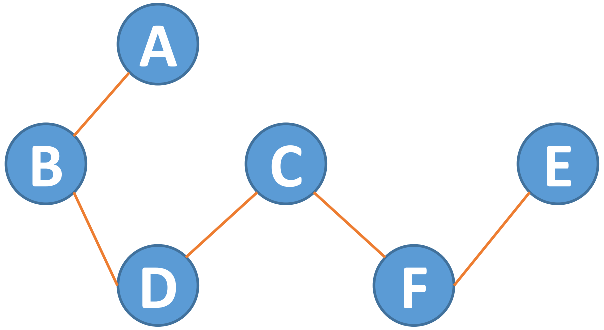
Spanning Tree



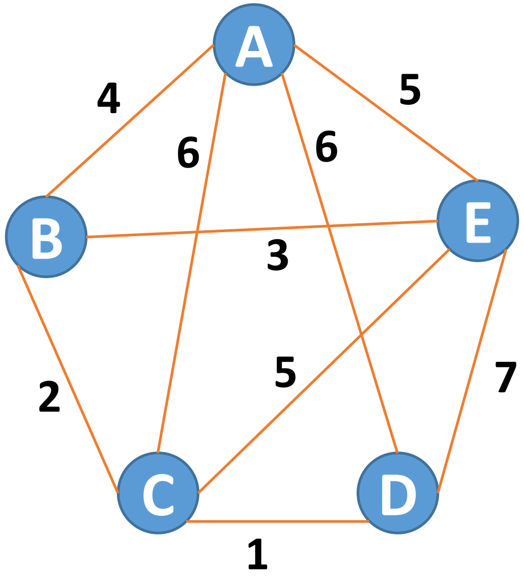
Graph



Spanning Tree

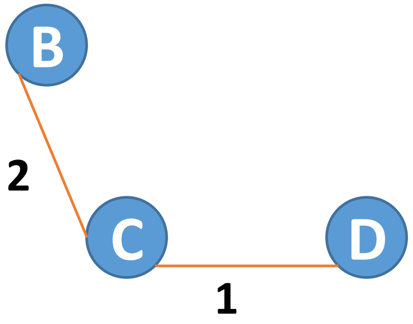


MST – Kruskal's Algorithm – Example

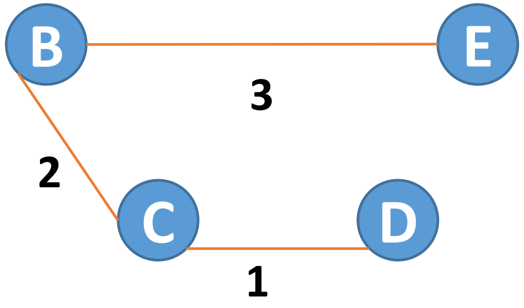


Step 1: Taking min edge (C,D)

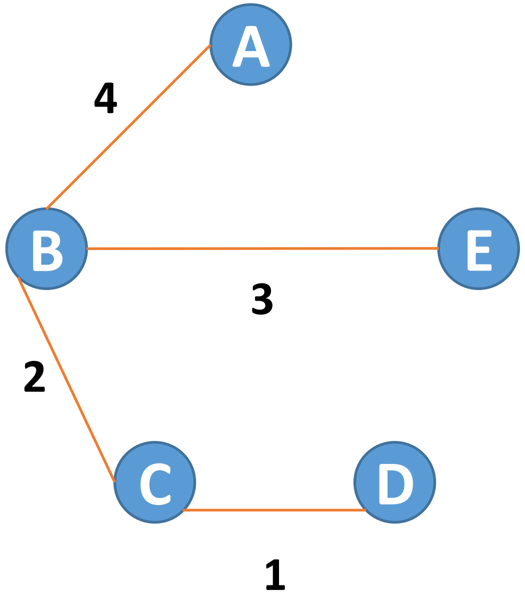
Step 2: Taking next min edge (B,C)



Step 3: Taking next min edge (B,E)



Step 4: Taking next min edge (A,B)



So, we obtained a minimum spanning tree of cost:
4 + 2 + 1 + 3 = 10

Kruskal's Algorithm

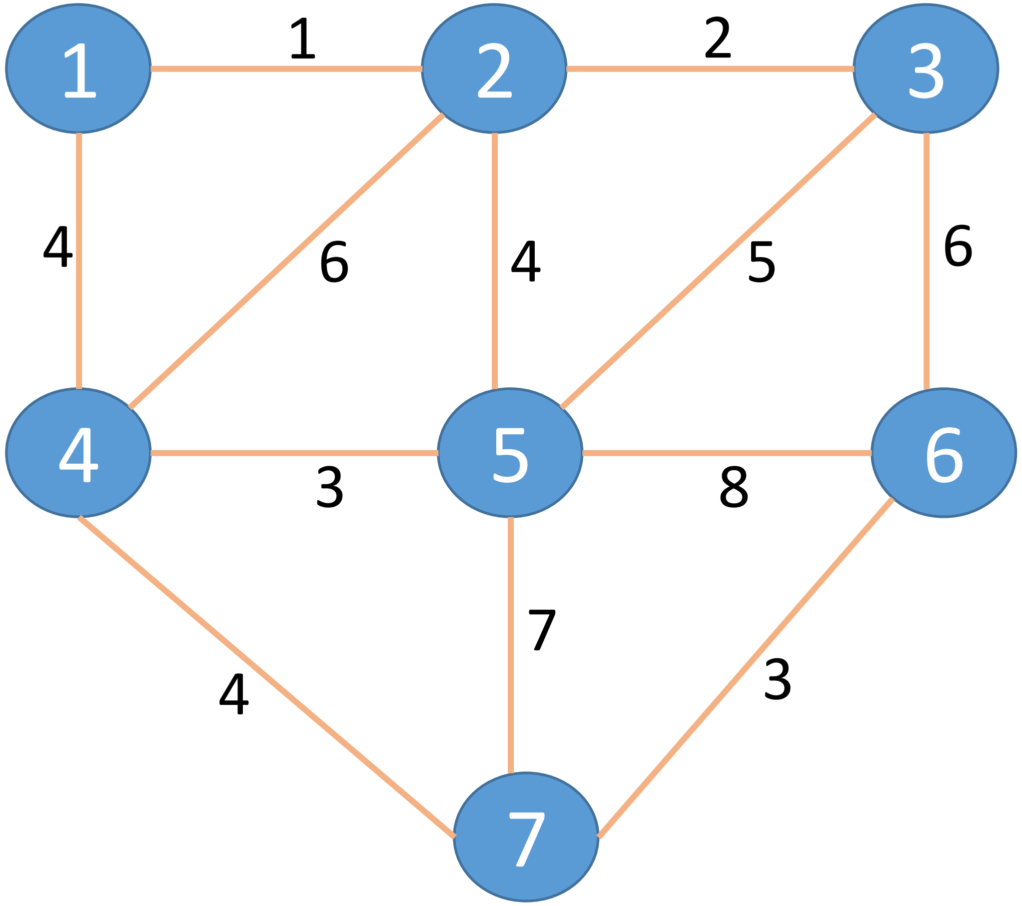
```
Function Kruskal(G = (N, A))
Sort A by increasing length
n ← the number of nodes in N
T ← Ø {edges of the minimum spanning tree}
Define n sets, containing a different element of set N
repeat
    e ← {u, v} //e is the shortest edge not yet considered
    ucomp ← find(u)
    vcomp ← find(v)
    if ucomp ≠ vcomp then merge(ucomp, vcomp)
    T ← T U {e}
until T contains n - 1 edges
return T
```

find(u) tells in which connected component a node u is found

merge(ucomp, vcomp) is used to merge two connected components.

Kruskal's Algorithm – Example

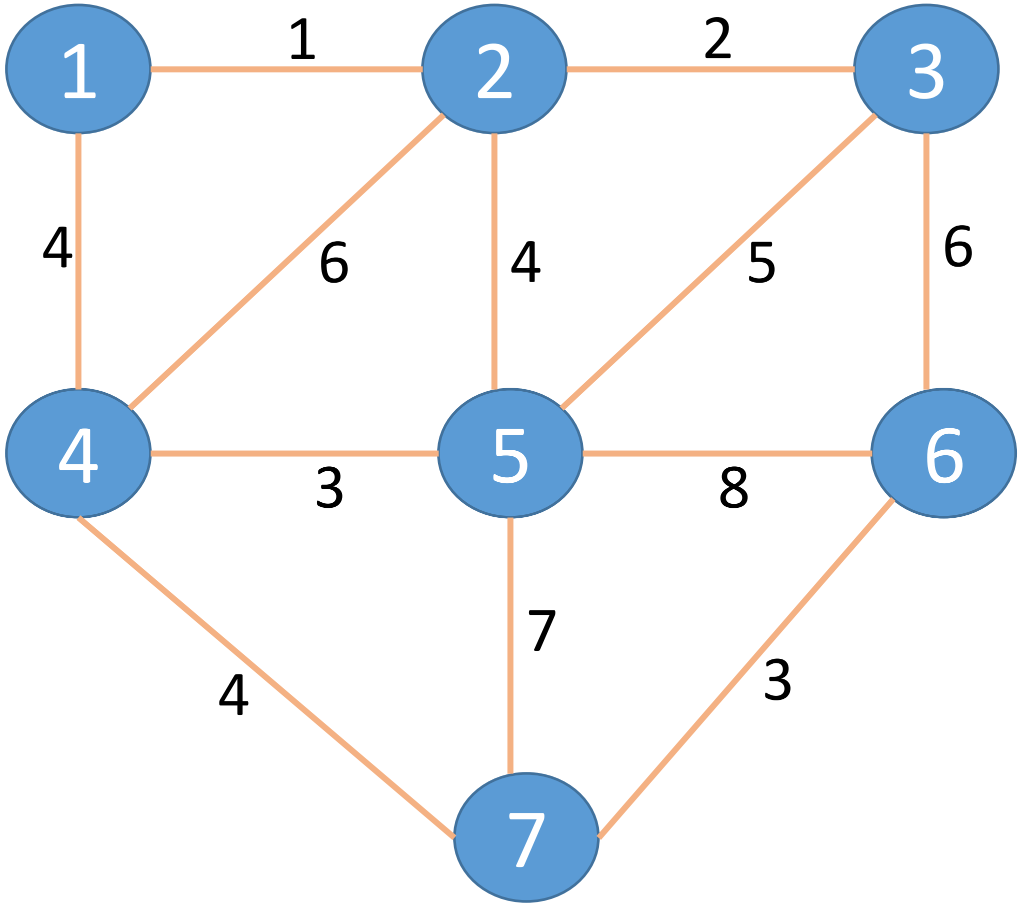
- Find the minimum spanning tree for the following graph using Kruskal's Algorithm.



Kruskal's Algorithm – Example

Step:1

Sort the edges in increasing order of their weight.

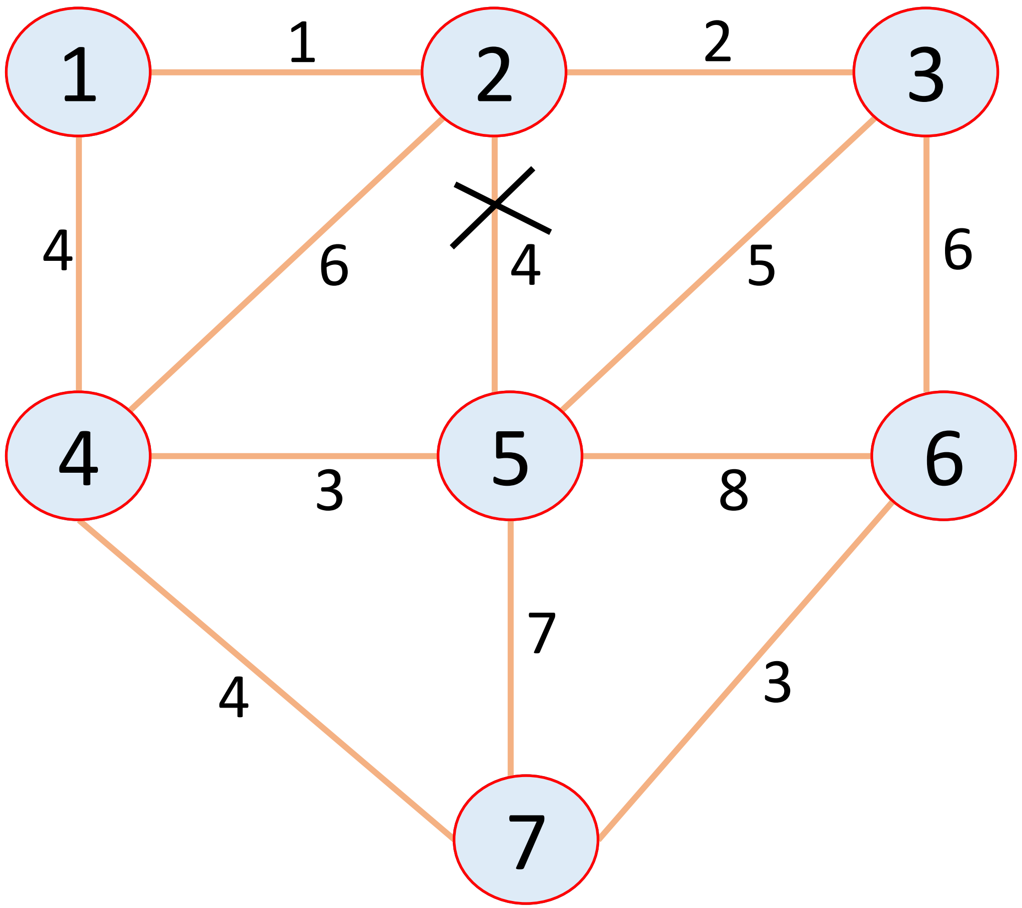


Edges	Weight	

Kruskal’s Algorithm – Example

Step:2

Select the minimum weight edge but no cycle.



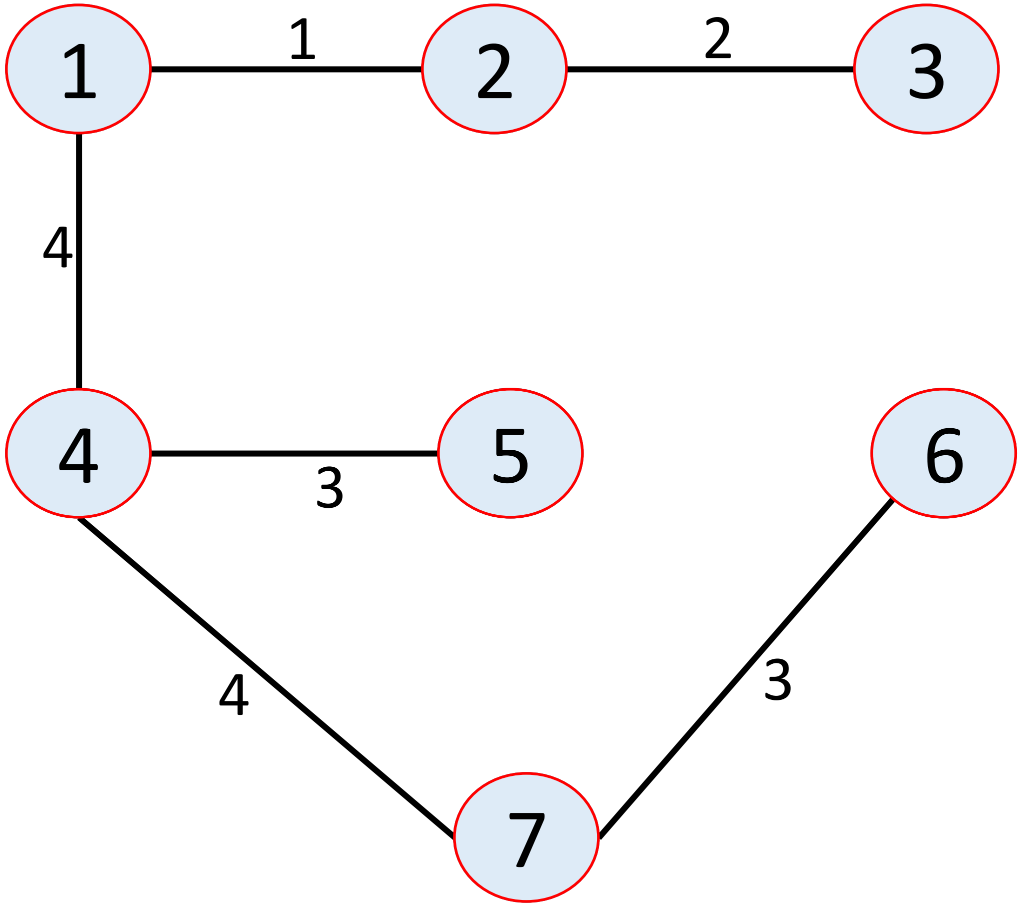
Edges	Weight	
{1, 2}	1	
{2, 3}	2	
{4, 5}	3	
{6, 7}	3	
{1, 4}	4	
~~{2, 5}~~	~~4~~	
{4, 7}	4	
{3, 5)	5	
{2, 4}	6	
{3, 6}	6	
{5, 7}	7	
{5, 6}	8	

Kruskal's Algorithm – Example

Step:3

The minimum spanning tree for the given graph

Total Cost = 17



Kruskal's Algorithm – Example

Step	Edges considered - {u, v}	Connected Components

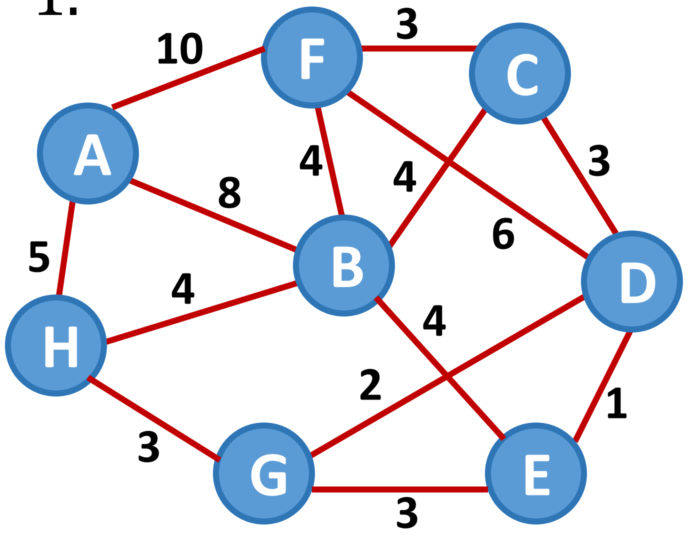
Edges	Weight
{1, 2}	1
{2, 3}	2
{4, 5}	3
{6, 7}	3
{1, 4}	4
~~{2, 5}~~	~~4~~
{4, 7}	4

Total Cost = 17

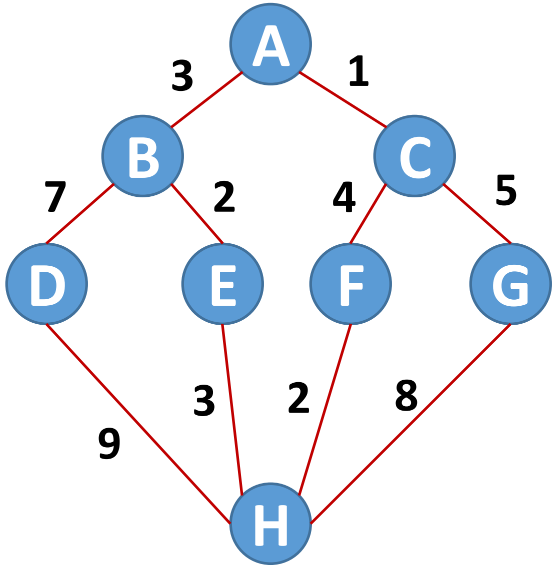
Exercises – Home Work

- Write the kruskal's Algorithm to find out Minimum Spanning Tree. Apply the same and find MST for the graph given below.

1.



2.



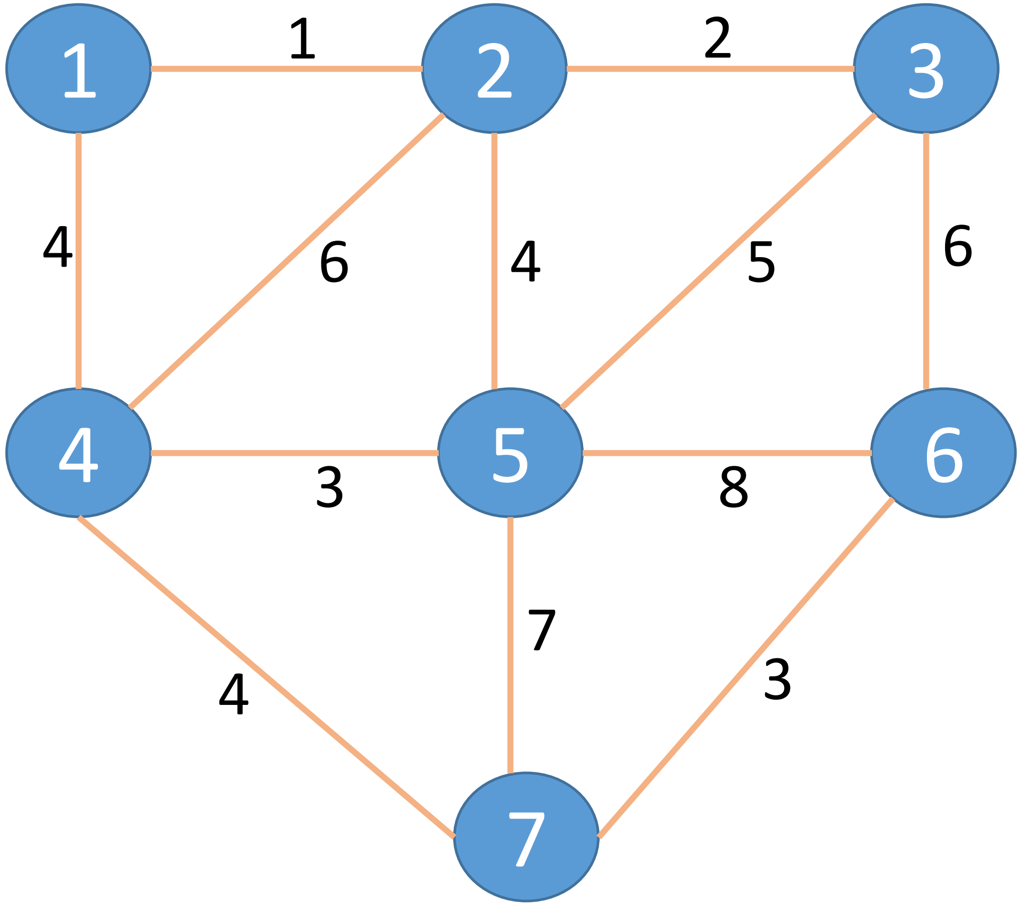
- The complexity for the Kruskal's algorithm is in $\theta(\boldsymbol{a}\log\boldsymbol{n})$ where a is total number of edges and n is the total number of nodes in the graph G.

Prim’s Algorithm

- In Prim's algorithm, the minimum spanning tree grows in a natural way, starting from an arbitrary root.
- At each stage we add a new branch to the tree already constructed; the algorithm stops when all the nodes have been reached.
- The complexity for the Prim’s algorithm is $\boldsymbol{\theta(n^2)}$ where n is the total number of nodes in the graph G.

Prim’s Algorithm – Example

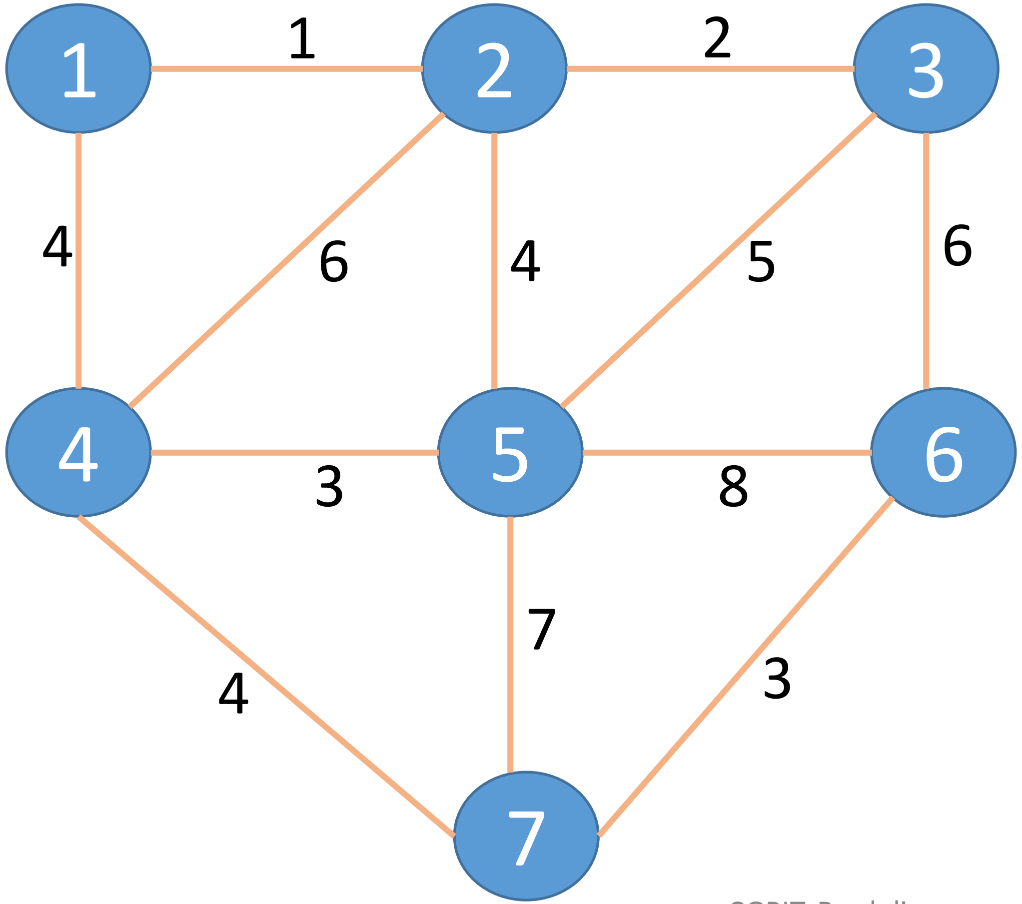
- Find the minimum spanning tree for the following graph using Prim’s Algorithm.



Prim's Algorithm – Example

Step:2

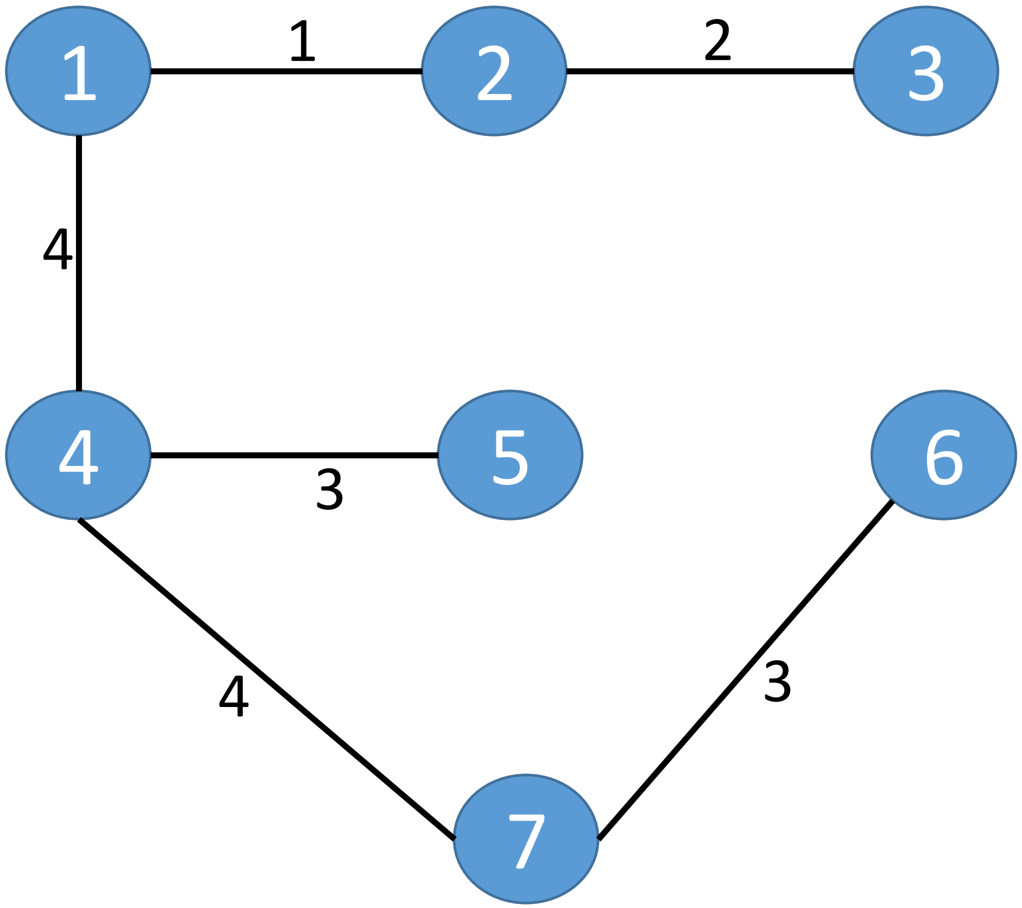
Find an edge with minimum weight



Node	Edges

Prim's Algorithm – Example

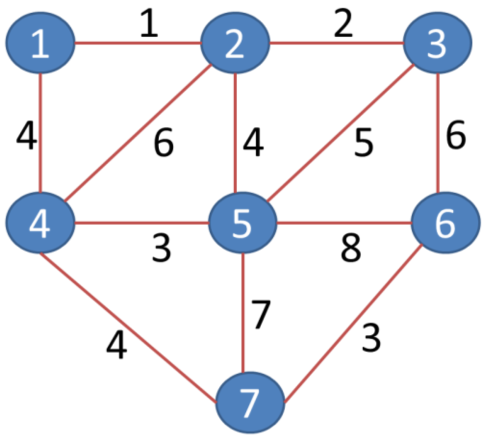
The minimum spanning tree for the given graph

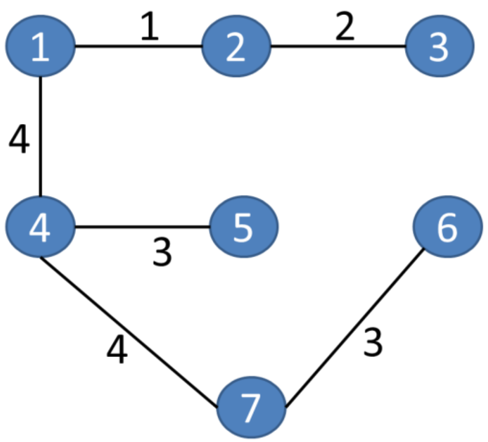


Node	Edges
1	{1, 2}
1, 2	{2, 3}
1, 2, 3	{1, 4}
1, 2, 3, 4	{4, 5}
1, 2, 3, 4, 5	{4, 7}
1, 2, 3, 4, 5, 6	{6, 7}

Total Cost = 17

Prim's Algorithm – Example





Cost = 17

Step	Edge Selected {u, v}	Set B	Edges Considered
Init.	-	{1}	--
1	{1, 2}	{1,2}	**{1,2}** {1,4}
2	{2, 3}	{1,2,3}	{1,4} **{2,3}** {2,4} {2,5}
3	{1, 4}	{1,2,3,4}	**{1,4}** {2,4} {2,5} {3,5} {3,6}
4	{4, 5}	{1,2,3,4,5}	{2,4} {2,5} {3,5} {3,6} **{4,5}** {4,7}
5	{4, 7}	{1,2,3,4,5,7}	{2,4} {2,5} {3,5} {3,6} **{4,7}** {5,6} {5,7}
6	{6,7}	{1,2,3,4,5,6,7}	{2,4} {2,5} {3,5} {3,6} {5,6} {5,7} **{6,7}**

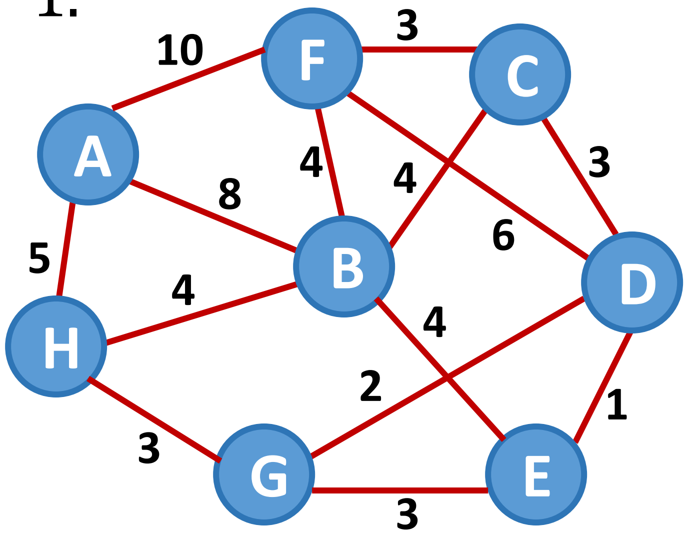
Prim's Algorithm

```
Function Prim(G = (N, A): graph; length: A — R+): set of
edges
T ← Ø
B ← {an arbitrary member of N}
while B ≠ N do
     find e = {u, v} of minimum length such that
          u ∈ B and v ∈ N \ B
     T ← T U {e}
     B ← B U {v}
return T
```

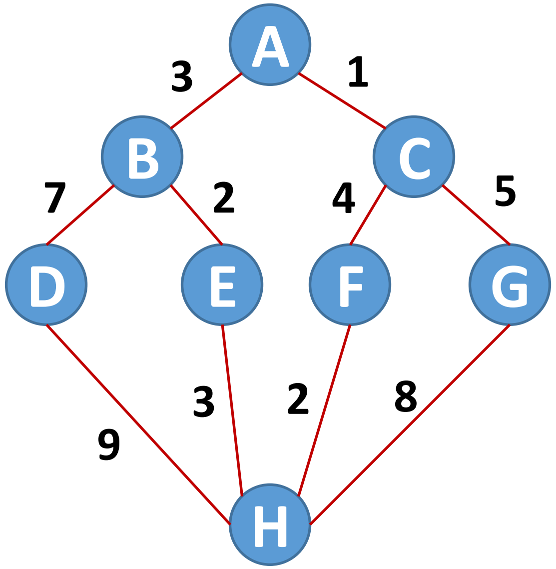
Exercises – Home Work

- Write the Prim's Algorithm to find out Minimum Spanning Tree. Apply the same and find MST for the graph given below.

1.



2.



Single Source Shortest Path – Dijkstra’s Algorithm

Dijkstra's Algorithm

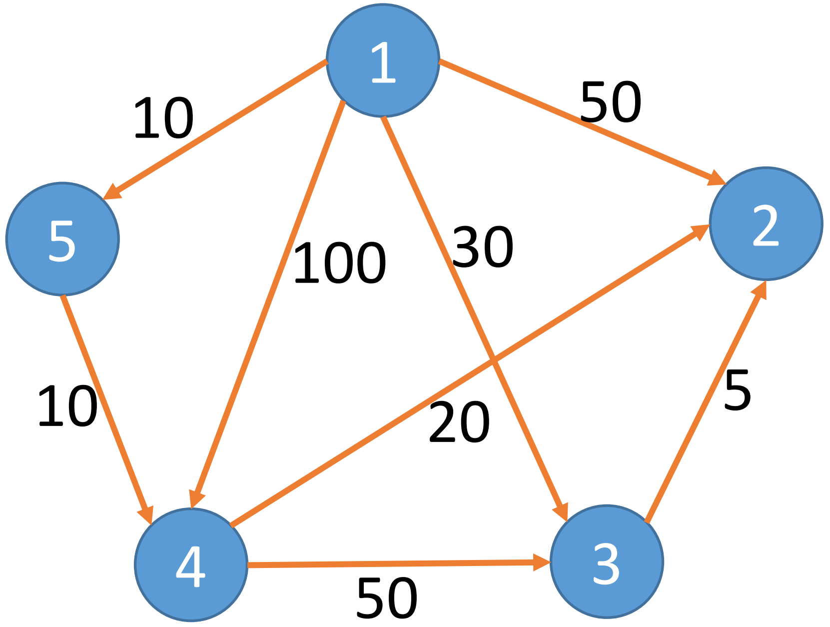
- Consider now **a directed graph** $\boldsymbol{G = (N, A)}$ where N is the set of nodes and A is the set of directed edges of graph G.
- Each edge has a **positive length**.
- One of the nodes is designated as the **source node**.
- The problem is **to determine the length of the shortest path** from the source to each of the other nodes of the graph.
- The **algorithm maintains a matrix** $\boldsymbol{L}$ which gives the length of each directed edge:

$$L[i, j] \geq 0 \textit{ if the edge } (i, j) \in A, \textit{and}$$
$$L[i, j] = \infty \textit{ otherwise.}$$

Dijkstra's Algorithm

```
Function Dijkstra(L[1 .. n, 1 .. n]): array [2..n]
array D[2.. n]
C ← {2,3,…, n}
{S = N \ C exists only implicitly}
for i ← 2 to n do
   D[i] ← L[1, i]
repeat n - 2 times
   v ← some element of C minimizing D[v]
   C ← C \ {v} {and implicitly S ← S U {v}}
   for each w ∈ C do
      D[w] ← min(D[w], D[v] + L[v, w])
return D
```

Dijkstra's Algorithm – Example



Is there path from 1 - 5 - 4

No | Yes

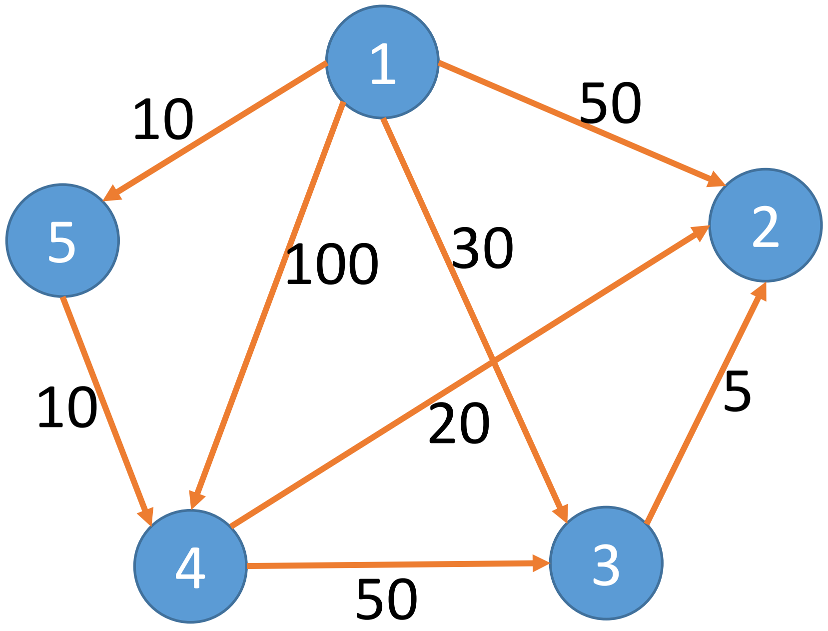
Compare cost of 1 – 5 – 4 and 1- 4

Single source shortest path algorithm

Source node = 1

Step	v	C	2	3	4	5
Init.	-	{2, 3, 4, 5}	50	30	100	10
1	5	{2, 3, 4}	50	30	20	10

Dijkstra's Algorithm – Example



Is there path from 1 - 4 - 5

No

Yes

Compare cost of 1 – 4 – 3 and 1-3

Single source shortest path algorithm

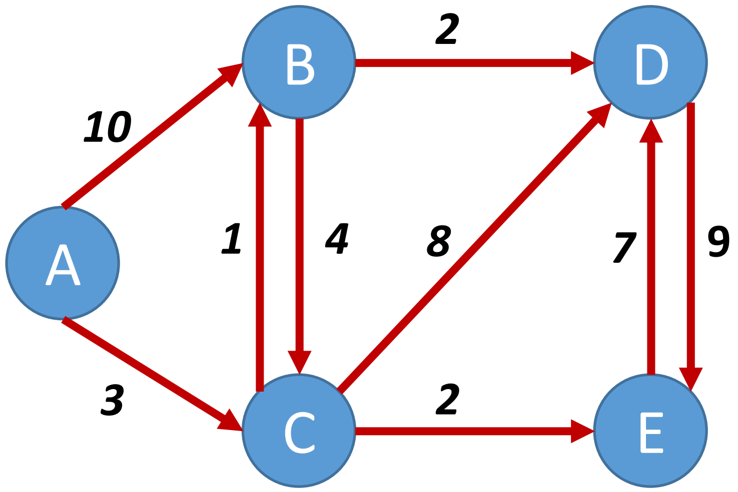
Source node = 1

Step	v	C	2	3	4	5
Init.	-	{2, 3, 4, 5}	50	30	100	10
1	5	{2, 3, 4}	50	30	20	10
2	4	{2, 3}	40	30	20	10
3	3	{2}	35	30	20	10

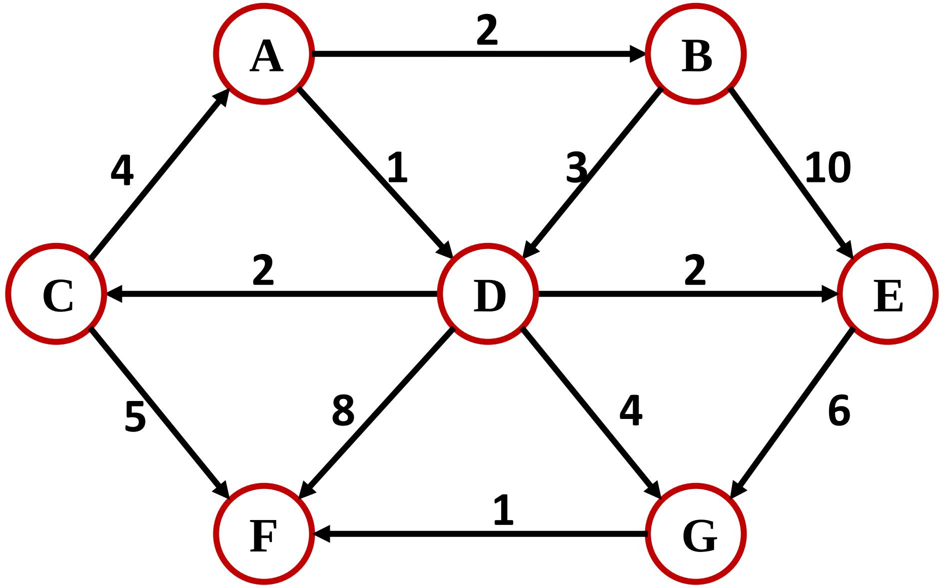
Exercises

- Write Dijkstra's Algorithm for shortest path. Use the algorithm to find the shortest path from the following graph.

1.



2.



Job Scheduling Problem

Job Scheduling with Deadlines

- We have set of n jobs to execute, each of which **takes unit time**.
- At any point of time we can **execute only one job.**
- Job i earns profit $g_i > 0$ if and only if it is executed **no later than** time d_i.
- We have to find an optimal sequence of jobs such that our total **profit is maximized**.
- *Feasible jobs: A set of job is feasible if there exits* ***at least one sequence*** *that allows all the jobs in the set to be executed no later than their respective deadlines.*

Job Scheduling with Deadlines – Algorithm

```
Algorithm:
let total position P = min(n, max(di))
for i=1 to n do (for total P slots)
      set k = min(dmax, deadline[i])
      while k>=1 do
            if timeslot[k] is empty then
                  timeslot[k] = job[i]
            break
            endif
                  set k = k-1
      end while
end for
```

Job Scheduling with Deadlines – Examples

- Using greedy algorithm find an optimal schedule for following jobs with $\boldsymbol{n} = \mathbf{6}$.
- Profits: $(P_1, P_2, P_3, P_4, P_5, P_6) = (15,20,10,7,5,3)$ &
- Deadline: $(d_1, d_2, d_3, d_4, d_5, d_6) = (1, 3, 1, 3, 1, 3)$

Solution:

$\boxed{\text{Step 1:}}$ Sort the jobs in **decreasing order** of their profit.

Job i	**1**	**2**	**3**	**4**	**5**	**6**
Profit g_i	**20**	**15**	**10**	**7**	**5**	**3**
Deadline d_i.	**3**	**1**	**1**	**3**	**1**	**3**

Job Scheduling with Deadlines – Examples

Job i	1	2	3	4	5	6
Profit g_i	20	15	10	7	5	3
Deadline d_i.	3	1	1	3	1	3

Step 2: Find total position $\boxed{P = \min(n, \max(di))}$

Here, $P = \min(6, 3) = \mathbf{3}$

P	1	2	3
Job selected	0	0	0

Step 3: $d_1 = 3$: assign job 1 to position 3

P	1	2	3
Job selected	0	0	J1

Job Scheduling with Deadlines – Examples

Job i	1	2	3	4	5	6
Profit g_i	20	15	10	7	5	3
Deadline d_i.	3	1	1	3	1	3

Step 4: $d_2 = 1$: assign job 2 to position 1

P	1	2	3
Job selected	J2	0	J1

Step 5: $d_3 = 1$: assign job 3 to position 1

Position 1 is already occupied, so reject job 3

Job Scheduling with Deadlines – Examples

Job i	1	2	3	4	5	6
Profit g_i	20	15	10	7	5	3
Deadline d_i.	3	1	1	3	1	3

$\boxed{\text{Step 6:}}$ $d_4 = 3$: assign job 4 to position 2 as, position 3 is not free but position 2 is free.

P	1	2	3
Job selected	J2	J4	J1

- Now **no more free position** is left so no more jobs can be scheduled.
- The final optimal sequence:

Execute the job in order 2, 4, 1 with total profit value 42.

Exercises

- Using greedy algorithm find an optimal schedule for following jobs with n=4.

  Profits: (a, b, c, d) = (20,10,40,30) &

  Deadline: $(d_1, d_2, d_3, d_4) = (4, 1, 1, 1)$

- Using greedy algorithm find an optimal schedule for following jobs with n=5.

  Profits: (a, b, c, d, e) = (100,19,27,25,15) &

  Deadline: $(d_1, d_2, d_3, d_4, d_5) = (2, 1, 2, 1, 3)$

Thank You!