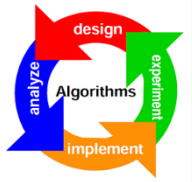
DESIGN AND ANALYSIS OF ALGORITHMS (DAA) (A34EC)

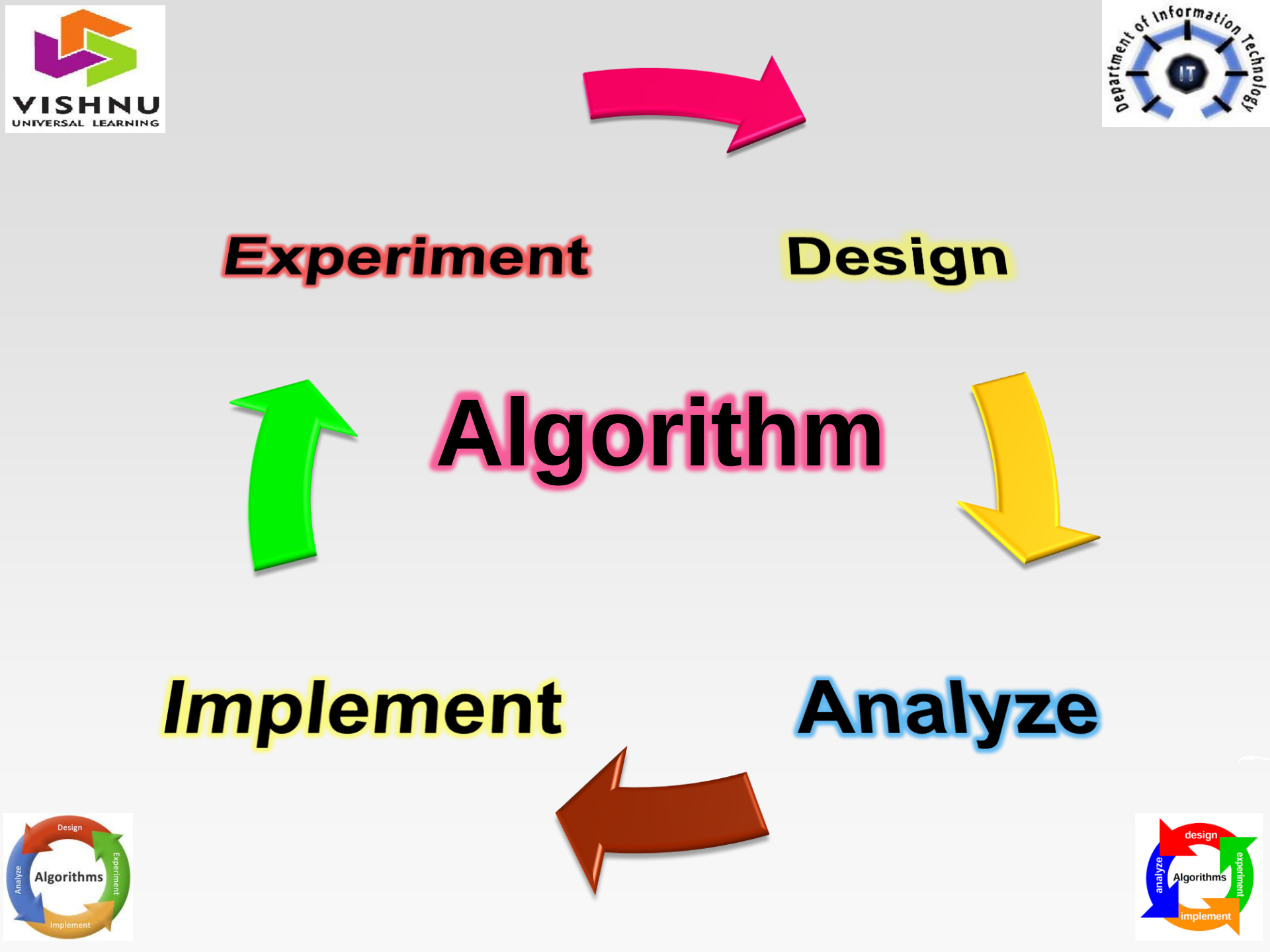
By :-

VIJAYKUMAR MANTRI,

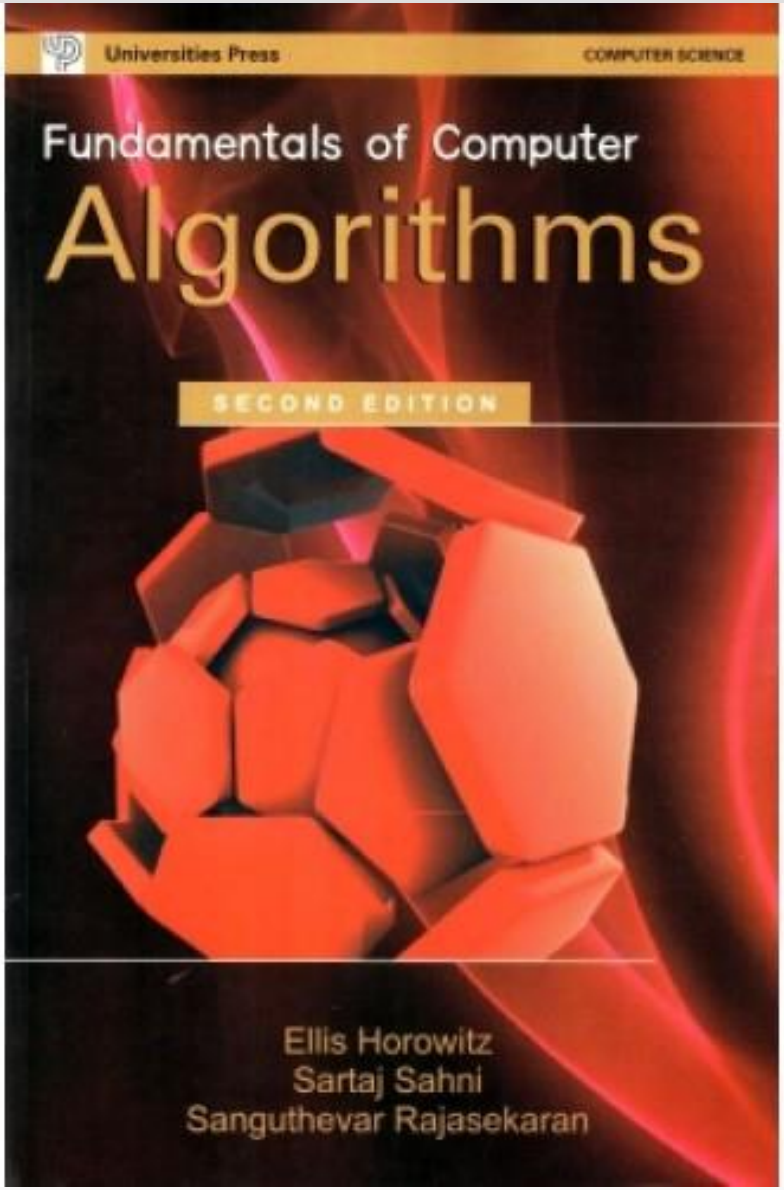
ASSOCIATE PROFESSOR.

vijay_mantri.it@bvrit.ac.in



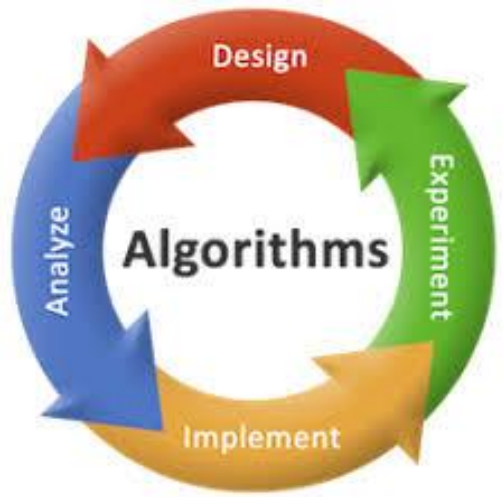

Experiment
Design
Algorithm
Implement
Analyze

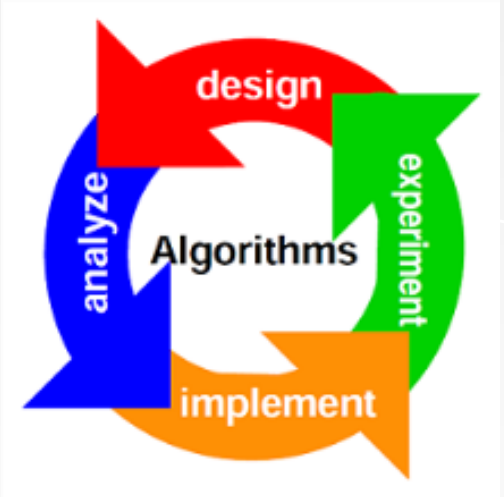
Textbook


Universities Press
COMPUTER SCIENCE
Fundamentals of Computer
Algorithms
SECOND EDITION
Ellis Horowitz
Sartaj Sahni
Sanguthevar Rajasekaran

DAA Unit III

Greedy Method

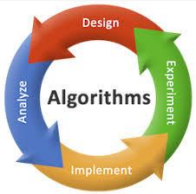


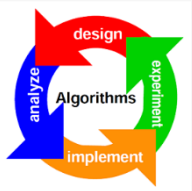


Unit III Syllabus

Greedy Method:

- General Method
- Applications –
  - Job Sequencing with Deadlines
  - Knapsack Problem
  - Minimum Cost Spanning Trees
    - Prim's Algorithm.
    - Kruskal's Algorithm
  - Single Source Shortest Path Problem.





Greedy Method – General Method

- Greedy method is the most straightforward designed technique.
- As the name suggest they are short sighted in their approach taking decision on the basis of the information immediately at the hand without worrying about the effect these decision may have in the future.

- **Greedy Method :** A problem have n inputs and require us to obtain a subset that that satisfy some constraints.
- Any subset that satisfy these constraints is called a **Feasible Solution**.
- A feasible solution that either **maximizes** or **minimizes** a given objective function is called an **Optimal Solution**.

Greedy Method – General Method

- A Greedy method suggests that one can devise an algorithm that works in stages, considering one input at a time for getting Optimal solution.
- This is done by considering the inputs in an order determined by some selection procedure.
- This version of greedy technique is called **Subset Paradigm**.
- For problems that do not call for the selection of an optimal subset in the greedy method we made decisions by considering the inputs in some order.
- Each decision is made using Optimization Criteria that can be computed using decisions already made.
- This version of greedy technique is called **Ordering Paradigm**.

Control algorithm for Greedy Method

```
1.  Algorithm Greedy (a, n)
2.  // a[ 1 : n ] contain the 'n' inputs
3.  {
4.    solution := ϕ ; // Initialize the solution.
5.    for i := 1 to n do
6.    {
7.      x := Select (a);
8.      if Feasible(solution, x) then
9.        solution := Union(solution, x);
10.   }
11.  return solution;
12.  }
```

Greedy Method – Examples

- Cash Denominations – Bank Cashier clearing a Cheque of Rs. 123,878/- & giving Cash Notes/Coins to Customer.
- Machine Scheduling – Complete given number tasks (with Start & Finish) time on number of Machines.

Knapsack Problem

- We are given n objects and an knapsack or bag.
- Object i had a weight w_i and knapsack capacity m.
- If a fraction $x_i, 0 \leq x_i \leq 1$, of object i placed into knapsack, then profit of $p_i x_i$ is earned.
- The objective is to obtain a filling of knapsack that maximizes the total profit earned.
- As knapsack capacity is m, we require the total weight of all chosen objects to be at most m.
- Formally the problem can be stated as

$$Maximize \sum_{1 \leq i \leq n} p_i x_i$$

$$Subject\ to \sum_{1 \leq i \leq n} w_i x_i \leq m$$

$$and\ 0 \leq x_i \leq 1, \qquad 1 \leq i \leq n$$

Algorithm for Knapsack Problem

```
1.  Algorithm GreedyKnapsack (m, n)
2.  // p[1:n] and w[1:n] contain the profits & weights of
3.  // the n object ordered such that
4.  // p[i] / w[i]  >=  p[i + 1] / w[i + 1]
5.  // m is the Knapsack size and x[1:n] is solution vector.
6.  {
7.     for i := 1 to n do x[ i ] :=  0.0; // Initialize x.
8.     U :=  m;
9.     for i :=  1 to n do
10.  {
11.    if (w[i] > U) then break;
12.    x[i] := 1.0; U:=  U  −  w[i]
13.  }
14.  if(i <= n) then x[ i ] :=  U/w[i];
15. }
```

Knapsack Problem – Examples

- Find an optimal solution to the knapsack instance(s).

Wi	Pi
m=750	n=15
70	135
73	139
77	149
80	150
82	156
87	163
90	173
94	184
98	192
106	201
110	210
113	214
115	221
118	229
120	240

(1)

Wi	Pi
m=165	n=10
23	92
31	57
29	49
44	68
53	60
38	43
63	67
85	84
89	87
82	72

(2)

Wi	Pi
m=26	n=5
12	24
7	13
11	23
8	15
9	16

(3)

Wi	Pi
m=104	n=8
25	350
35	400
45	450
5	20
25	70
3	8
2	5
2	5

(4)

Wi	Pi
m=190	n=6
56	50
59	50
80	64
64	46
75	50
17	5

(5)

Wi	Pi
m=50	n=7
31	70
10	20
20	39
19	37
4	7
3	5
6	10

(6)

Knapsack Problem – Solution

(1)	m = 750						n = 15								
Wi	70	73	77	80	82	87	90	94	98	106	110	113	115	118	120
Pi	135	139	149	150	156	163	173	184	192	201	210	214	221	229	240
Pi/Wi	1.93	1.90	1.94	1.88	1.90	1.87	1.92	1.96	1.96	1.90	1.91	1.89	1.92	1.94	2.00
Xi	**1**	**0**	**1**	**0**	**0**	**0**	**1**	**1**	**1**	**0**	**0**	**0**	**0.72**	**1**	**1**
Profit Pi*Wi	**135**	**0**	**149**	**0**	**0**	**0**	**173**	**184**	**192**	**0**	**0**	**0**	**159**	**229**	**240**

1) Total Profit = 1461

(2)	m = 165						n = 10			
Wi	23	31	29	44	38	53	63	85	89	82
Pi	92	57	49	68	43	60	67	84	87	72
Pi/Wi	4.00	1.84	1.69	1.55	1.13	1.13	1.06	0.99	0.98	0.88
Xi	**1**	**1**	**1**	**1**	**1**	**0**	**0**	**0**	**0**	**0**
Profit (Pi*Wi)	**92**	**57**	**49**	**68**	**43**	**0**	**0**	**0**	**0**	**0**

(3)	m = 26			n = 5	
Wi	12	7	11	8	9
Pi	24	13	23	15	16
Pi/Wi	2.00	1.86	2.09	1.88	1.78
Xi	**1**	**0**	**1**	**0.38**	**0**
Profit (Pi*Wi)	**24**	**0**	**23**	**6**	**0**

2) Total Profit = 309

3) Total Profit = 53

Knapsack Problem – Solution

(4)	m = 104						n = 8	
Wi	25	35	45	5	25	3	2	2
Pi	350	400	450	20	70	8	5	5
Pi/Wi	14.00	11.43	10.00	4.00	2.80	2.67	2.50	2.50
Xi	**1**	**1**	**0.98**	**0**	**0**	**0**	**0**	**0**
Profit Pi*Wi	**350**	**400**	**441**	**0**	**0**	**0**	**0**	**0**

(5)	m = 190				n = 6	
Wi	56	59	80	64	75	17
Pi	50	50	64	46	50	5
Pi/Wi	0.89	0.85	0.80	0.72	0.67	0.29
Xi	**1**	**1**	**0.94**	**0**	**0**	**0**
Profit Pi*Wi	**50**	**50**	**60**	**0**	**0**	**0**

4) Total Profit = 1191

5) Total Profit = 160

6) Total Profit = 108

(6)	m = 50				n = 7		
Wi	31	10	20	19	4	3	6
Pi	70	20	39	37	7	5	10
Pi/Wi	2.26	2.00	1.95	1.95	1.75	1.67	1.67
Xi	**1**	**1**	**0.45**	**0**	**0**	**0**	**0**
Profit Pi*Wi	**70**	**20**	**18**	**0**	**0**	**0**	**0**

Job Sequencing with Deadlines

- We are given a set of n jobs.
- Associated with job i is an integer deadline $d_i \geq 0$ and a profit $p_i > 0$.
- If any job i the profit p_i is earned iff the job is completed by its deadline.
- To complete a job, one has to process the job on a machine for one unit of time.
- Only one machine is available for processing jobs.
- A feasible solution for this problem is a subset J of jobs such that each job in this subject can be completed by this deadline.
- The value of feasible solution J is the sum of the profits of the jobs in J i.e. $\sum_{i \in J} p_i$
- An optimal solution is a feasible solution with maximum value.
- Since the problem involves the identification of a subset, it fits the Subset Paradigm.

- **Example 1.** $n = 5$ $(p_1, p_2, p_3, p_4, p_5) = (20, 15, 10, 5, 1)$
  $(d_1, d_2, d_3, d_4, d_5) = (2, 2, 1, 3, 3)$
- Feasible solution — Processing Sequence — Value

1.	(1)	1	20
2.	(2)	2	15
3.	(3)	3	10
4.	(4)	4	5
5.	(5)	5	1
6.	(1, 2)	1, 2 or 2, 1	35
7.	(1, 3)	3, 1	30
8.	(1, 4)	1, 4 or 4, 1	25
9.	(1, 5)	1, 5 or 5, 1	21
10.	(2, 3)	3, 2	25
11.	(2, 4)	2, 4 or 4, 2	20
12.	(2, 5)	2, 5 or 5, 2	16
13.	(1, 2, 4)	1, 2, 4	40
14.	(1, 3, 5)	3, 1, 5	31

- The Solution 13 is optimal

- **Example 2.** $n = 4$ $(p_1, p_2, p_3, p_4) = (100, 10, 15, 27)$

$(d_1, d_2, d_3, d_4) = (2, 1, 2, 1)$

- Feasible solution

		Processing Sequence	Value
1.	(1, 2)	2, 1	110
2.	(1, 3)	1, 3 or 3, 1	115
3.	(1, 4)	4, 1	127
4.	(2, 3)	2, 3	25
5.	(3, 4)	4, 3	42
6.	(1)	1	100
7.	(2)	2	10
8.	(3)	3	15
9.	(4)	4	27

- The Solution 3 is optimal

Job Sequencing with Deadlines

```
1.  Algorithm JS(d, J, n)
2.  // d[i] ≥ 1, 1 ≤ i ≤ n are the deadlines, n ≥ 1.
3.  // The job are ordered such that p[1] ≥ p[2] …. ≥ p[n]
4.  // J[i] is the ith job in the optimal solution, 1 ≤ i ≤ k.
5.  // Also at termination d[J[i]] ≤ d[J[i + 1]], 1 ≤ i < k.
6.  {
7.    d[0] := J[0] := 0; // Initialize
8.    J[1] = 1; // Include job 1
9.    k := 1;
10.   for i := 2 to n do
11.   {
12.   // Consider jobs in non increasing order of P[i]; Find
13.  // the position for i and check feasibility of insertion
14.  r := k;
```

Job Sequencing with Deadlines

15. $while\left(\left(d\left[J[r]\right] > d[i]\right)and\left(d\left[J[r]\right] \neq r\right)\right)do\ \ r := r - 1;$
16. $if\ \left(\left(d\left[J[r]\right] \leq d[i]\right)and\ (d[i] > r)\right)then$
17. $\{$
18. $//\ Insert\ I\ into\ J[\ ]$
19. $for\ q := k\ to\ (r + 1)step - 1\ do\ J\ [q + 1] := J[q];$
20. $J[r + 1] := i;$
21. $k := k + 1;$
22. $\}$
23. $\}$
24. $return\ k;$
25. $\}$

High Level description of Job Sequencing algorithm

```
1.  Algorithm GreedyJob(d, J, n)
2.  // J is a set of jobs that can be completed by thier
3.  // deadlines
4.  {
5.    J := {1};
6.    for i := 2 to n do
7.    {
8.      if (all jobs in J ∪ {i} can be completed by their
9.      deadlines) then
10.       J := J ∪ {i}
11.   }
12. }
```

- *Let's revisit example 1 to know how Optimal Solution is found.*

$$n = 5 \quad (p_1, p_2, p_3, p_4, p_5) = (20, 15, 10, 5, 1)$$

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

- *The array P is sorted as per Profits, lets see how it works as per algorithm we seen.*

J	*Assigned Slots*	*Job Considered*	*Action*	*Profit*
Φ	none	1	Assign to [1, 2]	0
{1}	[1, 2]	2	Assign to [0, 1]	20
{1, 2}	[0, 1], [1, 2]	3	Cannot fit, reject	35
{1, 2}	[0, 1], [1, 2]	4	Assign to [2, 3]	35
{1, 2, 4}	[0, 1], [1, 2], [2, 3]	5	Reject	40

- **The Optimal Solution is J = {1, 2, 4} with a profit of 40.**

Job Sequencing with Deadlines

- **Example 3 : Find Feasible solutions & Optimal Solution for given jobs with deadlines for $n = 7$**

$$(p_1, p_2, p_3, p_4, p_5, p_6, p_7) = (3, 5, 20, 18, 1, 6, 30)$$

$$(d_1, d_2, d_3, d_4, d_5, d_6, d_7) = (1, 3, 4, 3, 2, 1, 2)$$

- **Also show the Optimal Solution generated by function (algorithm) JS. (Hint : Answer is J6, J7, J4 & J3)**
- **Example 4 : Find Feasible solutions & Optimal Solution for given jobs with deadlines for $n = 7$**

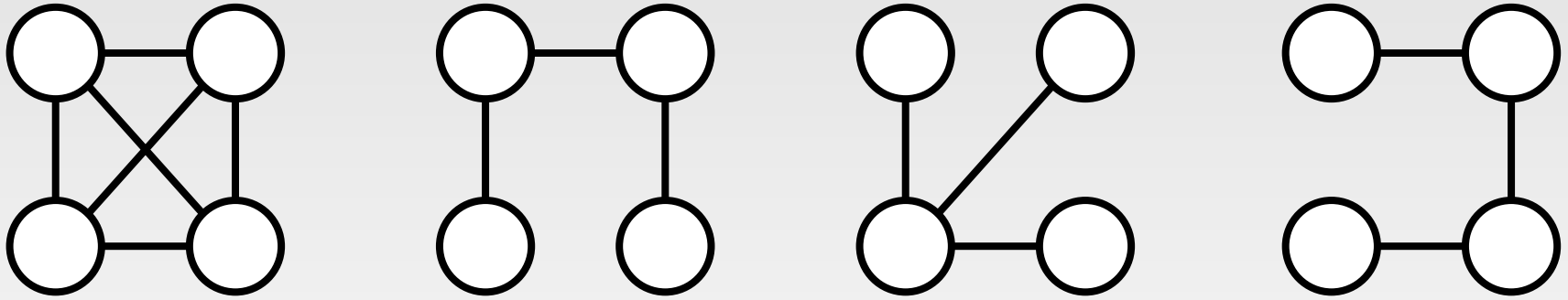
$$(p_1, p_2, p_3, p_4, p_5, p_6, p_7) = (3, 5, 20, 18, 1, 6, 30)$$

$$(d_1, d_2, d_3, d_4, d_5, d_6, d_7) = (1, 2, 4, 2, 2, 1, 2)$$

- **Also show the Optimal Solution generated by function (algorithm) JS. (Hint : Be careful with deadlines)**

Minimum-Cost Spanning Trees

- Let $G = (V, E)$ be an undirected connected graph with vertices 'V' and edges 'E'.
- A sub-graph $\mathbf{T} = (V, E')$ of the G is a Spanning tree of G iff 'T' is a tree.

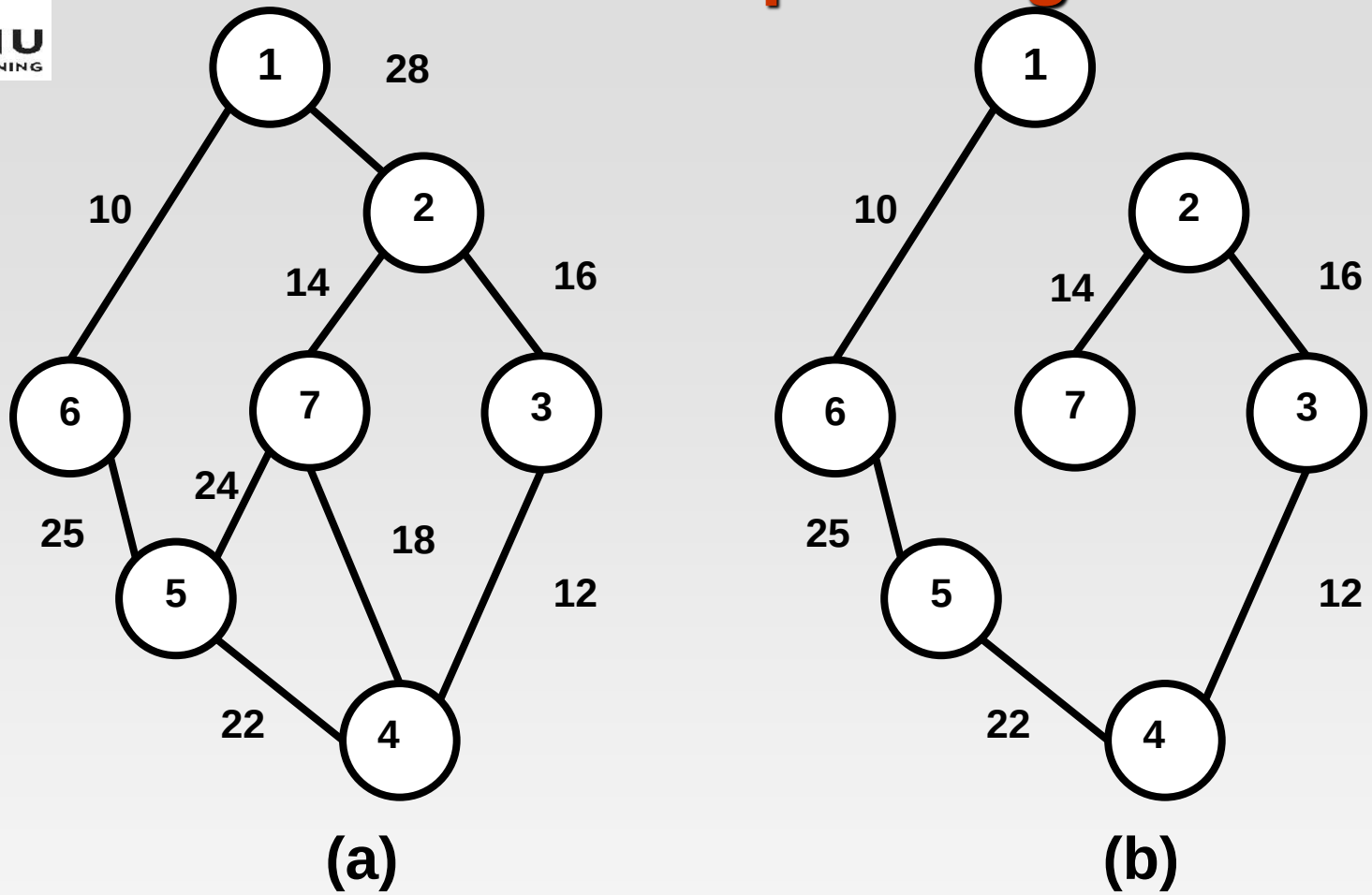


- The figure shows the Complete Graph with 4 Nodes and three spanning trees of the same.
- The spanning trees have many applications like Analysis of Electrical Circuits, Shortest route problems, etc.

Minimum-Cost Spanning Trees

- The problem is to generate a minimal subgraph G' of G such that $G' = (V, E')$ where E' is the subset of E, & G' is a Minimal Subgraph (minimum spanning tree).
- A minimal subgraph is one with the fewest number of edges.
- Any connected graph with n vertices must have at least $n - 1$ edges and all connected graphs with $n - 1$ edges are trees.
- The spanning trees of G represent all feasible choices
- Each and every edge will contain the given non-negative length.
- Connect all the nodes with edge present in set E' and weight has to be minimum i.e. we are interested in finding a spanning tree of G with minimum cost.
- The cost of a spanning tree is the sum of the costs of the edges in that tree.

Minimum-Cost Spanning Trees



- The figure shows a Graph & its minimum-cost spanning tree.
- There are 2 method to determine a minimum-cost spanning tree namely **Kruskal's Algorithm and Prim's Algorithm.**

Kruskal's Algorithm

- In Kruskal's algorithm the selection function chooses edges in increasing order of length without worrying too much about their connection to previously chosen edges, except that never to form a cycle.
- The result is a forest of trees that grows until all the trees in a forest (all the components) merge in a single tree.
- In this algorithm, a minimum cost-spanning tree '***T***' is built edge by edge.
- Edge are considered for inclusion in '***T***' in increasing order of their cost.
- An edge is included in '***T***' if it doesn't form a cycle with edge already in ***T***.
- To find the minimum-cost spanning tree the edge are inserted to tree in increasing order of their cost.

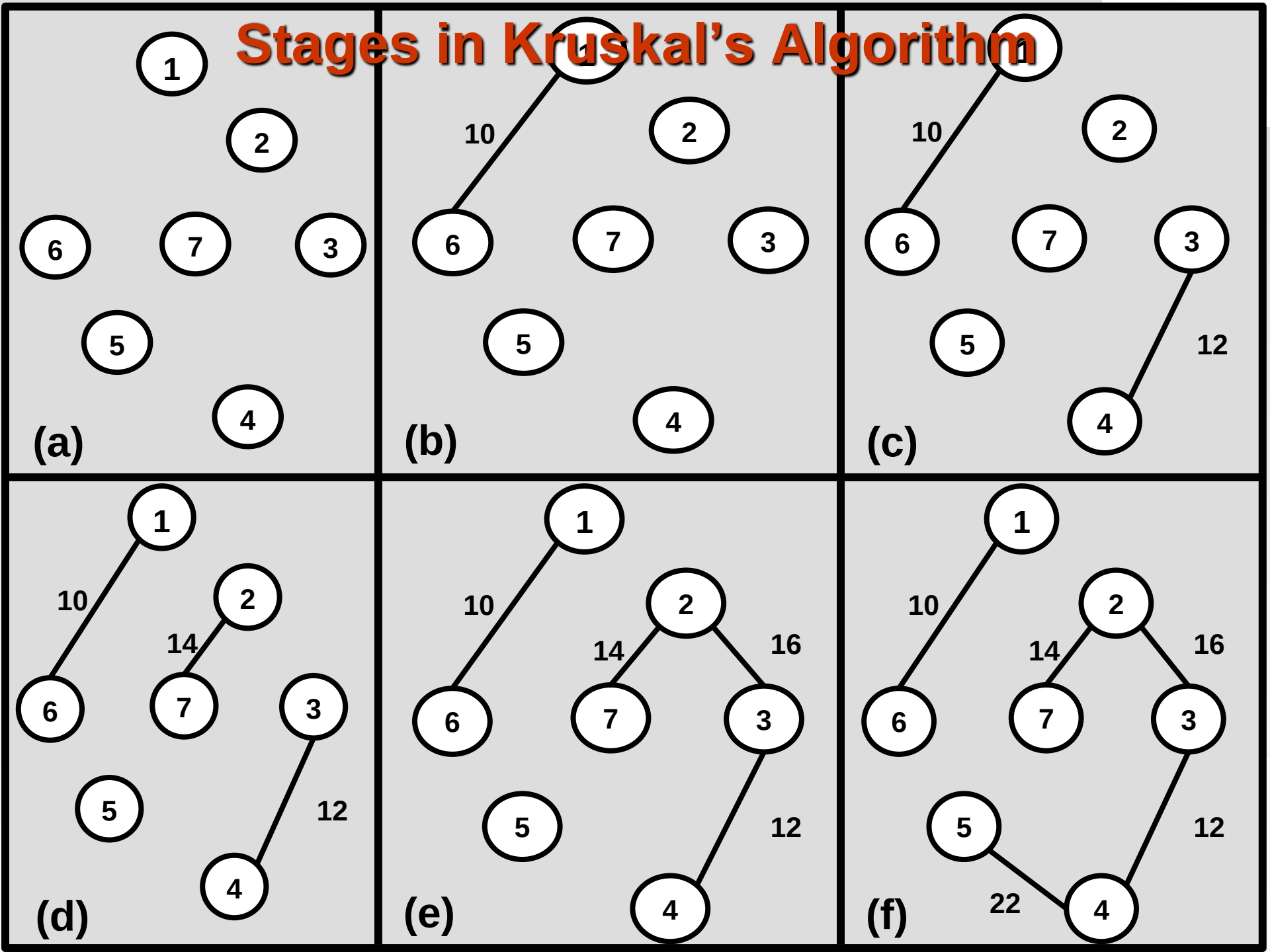
Kruskal's Algorithm

1. Let $\boldsymbol{G} = (\boldsymbol{V}, \boldsymbol{E})$ be a connected graph with weights assigned to each edge.
2. Select any edge of minimum value of $\boldsymbol{G}$. This is the first edge of minimal spanning tree $\boldsymbol{T}$.
3. Select any edge $(\boldsymbol{v}, \boldsymbol{w})$ of $\boldsymbol{E}$ from remaining edges of $\boldsymbol{G}$ having minimum value, which will not form a closed path with the edges already included in $\boldsymbol{T}$.
4. Step **3** is repeated until $\boldsymbol{T}$ contains $\boldsymbol{n} - \boldsymbol{1}$ edges where $\boldsymbol{n}$ is number of vertices of $\boldsymbol{G}$.
5. Now the tree $\boldsymbol{T}$ becomes Minimal-Cost Spanning Tree of $\boldsymbol{G}$.

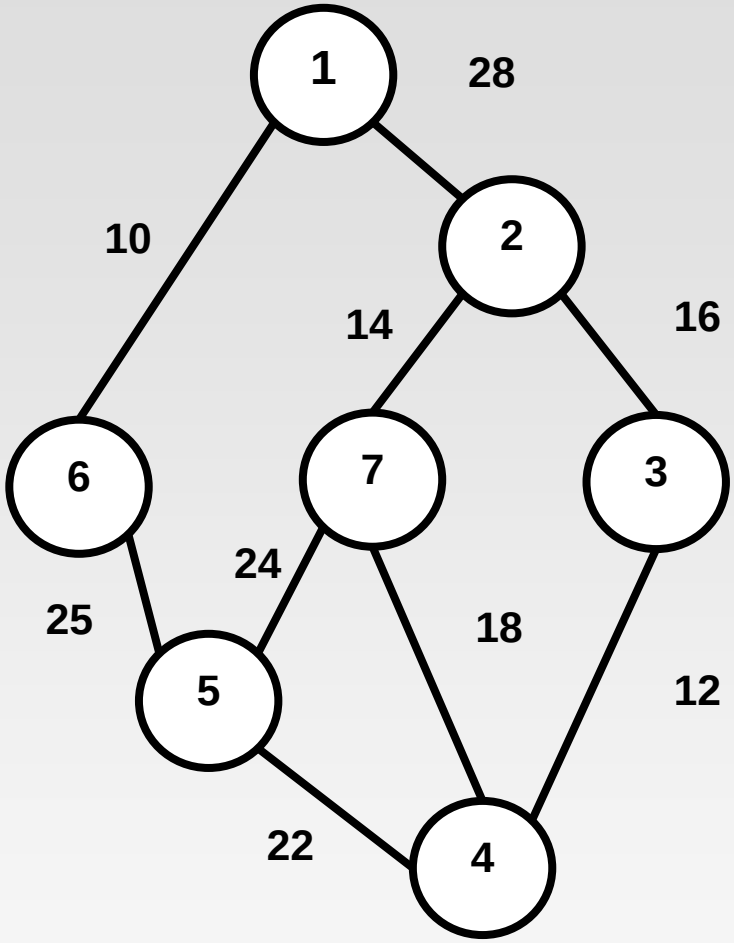
Kruskal's Algorithm – Early form of Minimum-Cost Spanning Tree algorithm

1. $t := \emptyset$;
2. $while\ ((t\ has\ less\ than\ n-1\ edges)\ and\ (E \neq \emptyset))do$
3. {
4. $\quad Choose\ an\ edge\ (u,w)\ from\ E\ of\ lowest\ cost$;
5. $\quad Delete\ (u,w)\ from\ E$;
6. $\quad if\ (u,w)\ does\ not\ create\ a\ cycle\ in\ t\ then\ add\ (u,w)\ to\ t$;
7. $\quad else\ discard\ (u,w)$
8. }

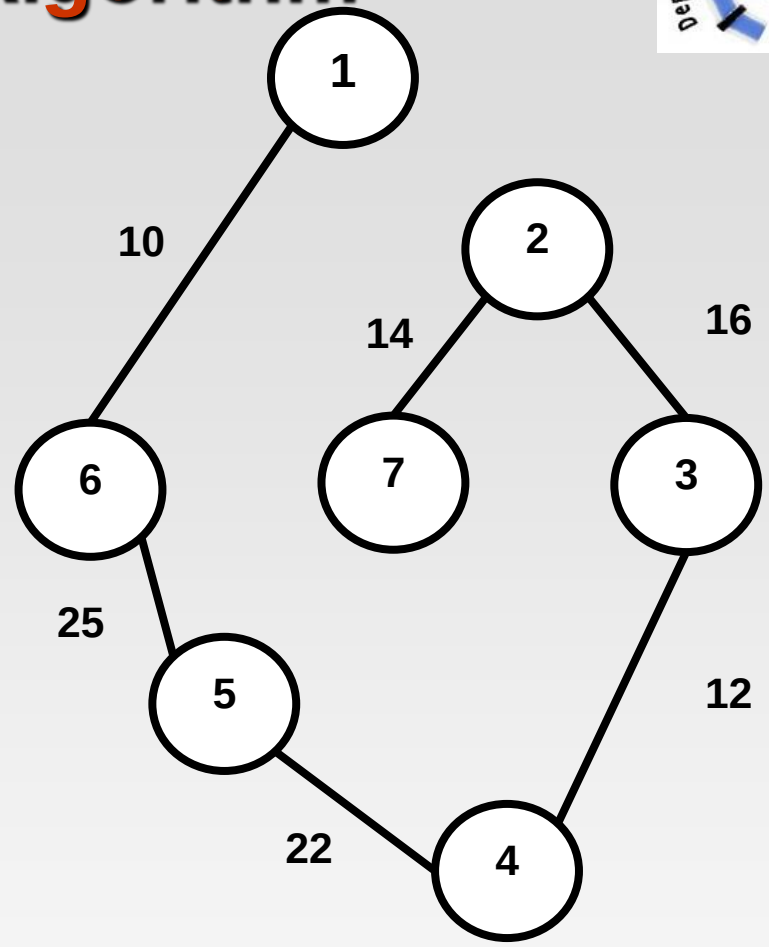
Stages in Kruskal's Algorithm



Kruskal's Algorithm

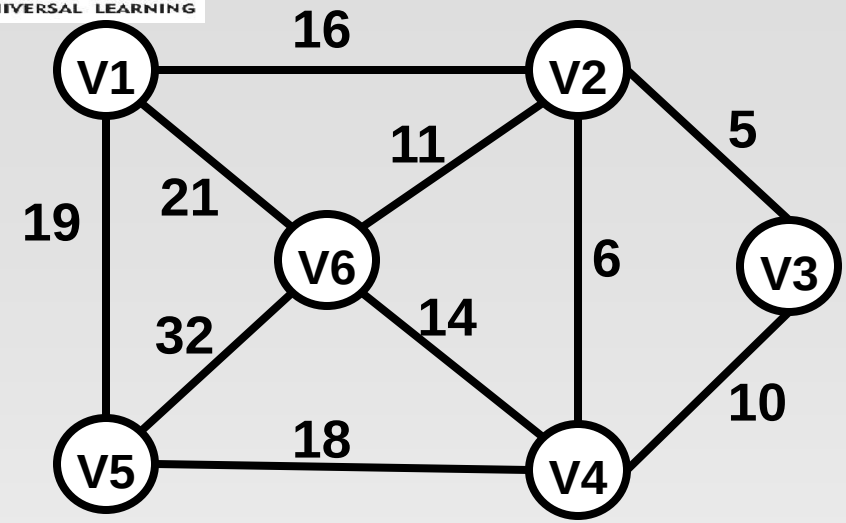


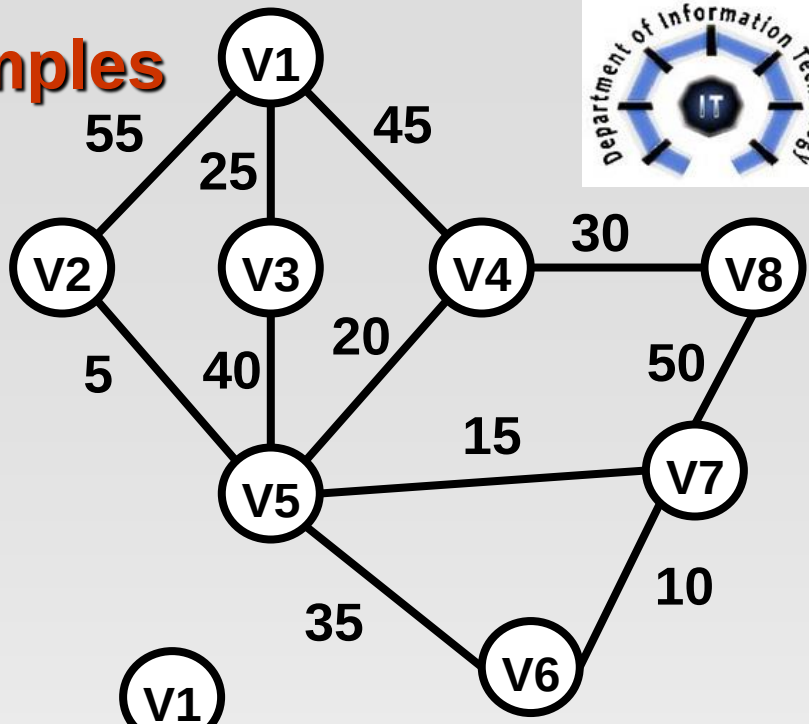
Original Graph G

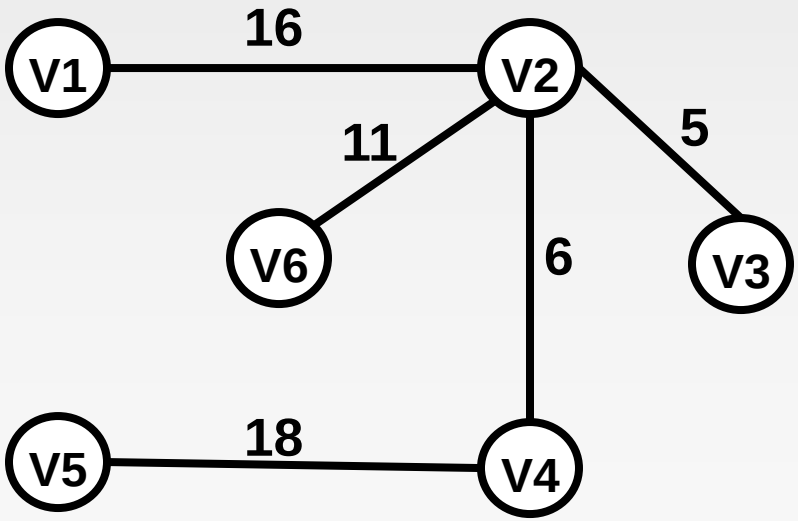


Minimum-Cost Spanning Tree T of G.
Total Cost = 99

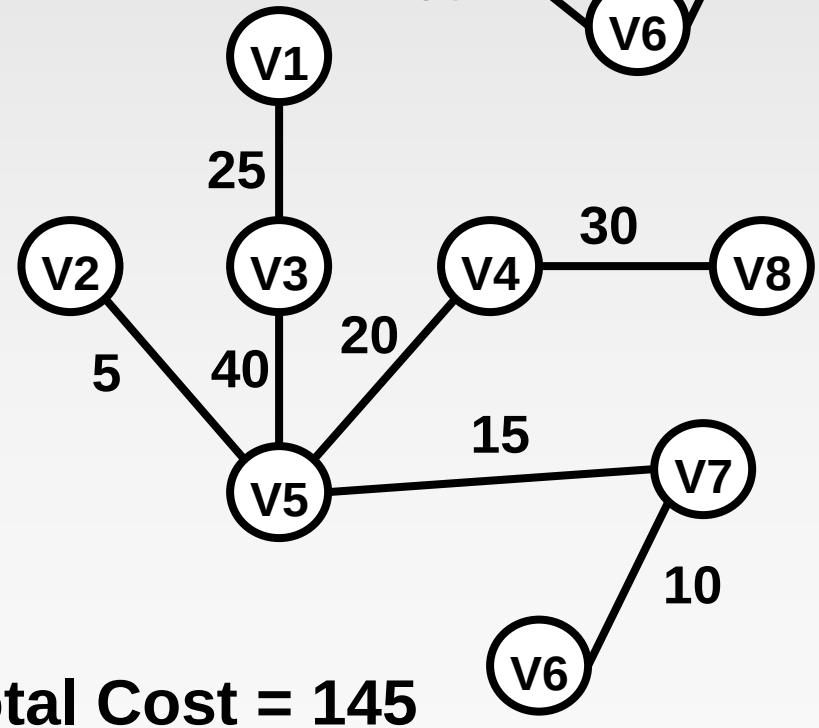
Kruskal's Algorithm Examples







Total Cost = 56



Total Cost = 145

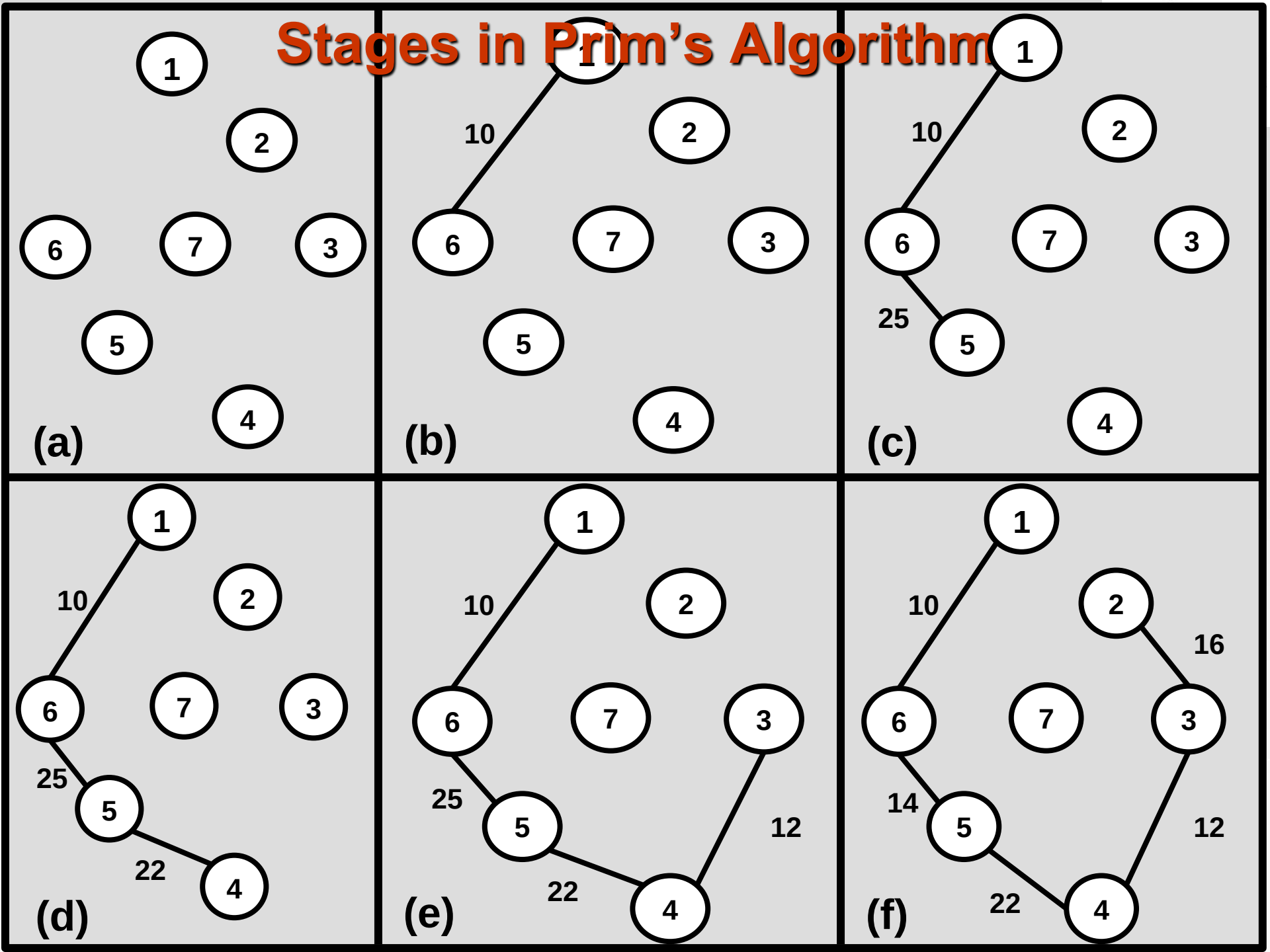
Prim's Algorithm

- Let G be a connected graph with weights assigned to each edge.
- In Prim's algorithm, we start from an arbitrary vertex (root).
- At each stage, add a new branch (edge) to the tree already constructed; the algorithm halts when all the vertices in the graph have been reached.
- The Prims algorithm will start with a tree that includes only a minimum cost edge of G.
- Then, edges are added to the tree one by one.
- The next edge (i, j) to be added in such that i is a vertex included in the tree, j is a vertex not yet included, and cost of $(i, j), cost[i, j]$ is minimum among all the edges.

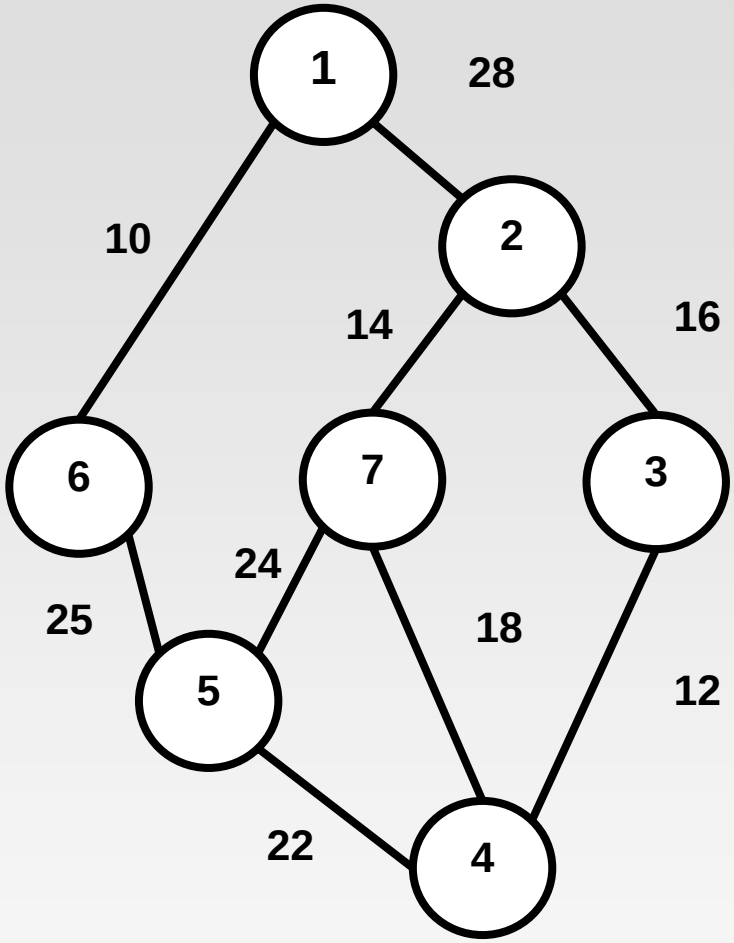
Prim's Algorithm

1. Let G be a connected graph with weights assigned to each edge.
2. First let T be the minimal spanning tree consists of any vertex V of G.
3. Among all edges not in T, which are incident on a vertex (neighborhood of vertex) in T, and not forming a closed path when added to T, select the minimum cost edge and add it to T.
4. The step 3 is repeated until we select $n - 1$ edges which covers n vertices in G resulting in Minimal-Cost Spanning Tree T.
5. Now the tree T becomes Minimal-Cost Spanning Tree of G.

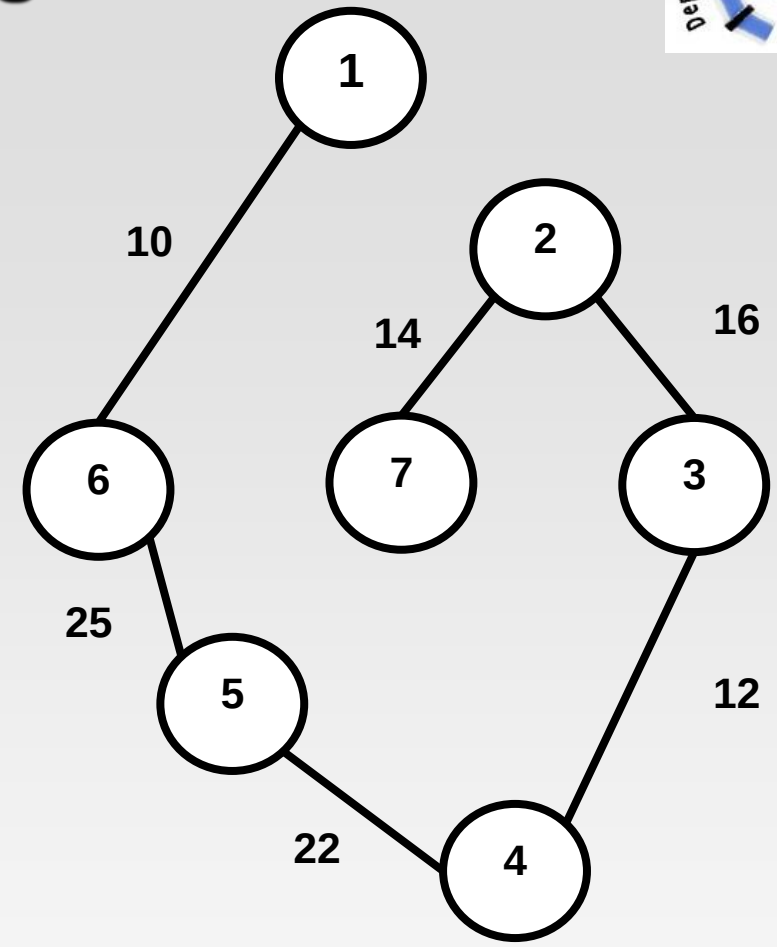
Stages in Prim's Algorithm



Prim's Algorithm

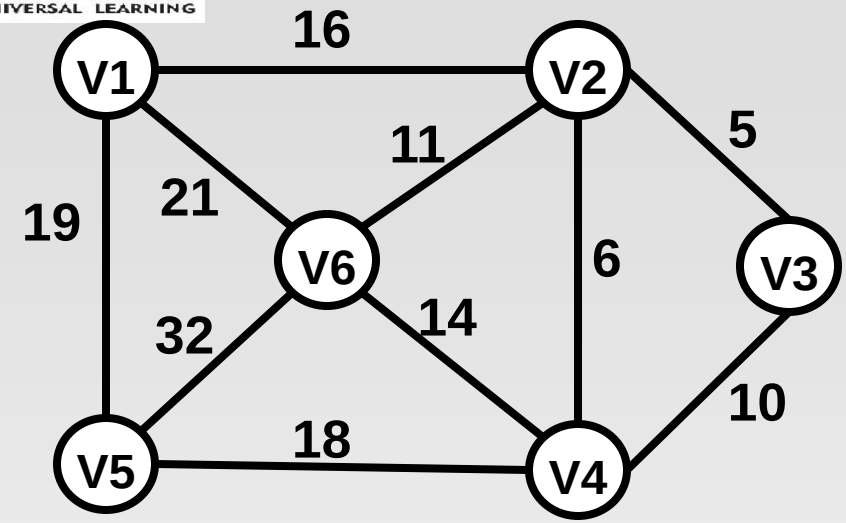


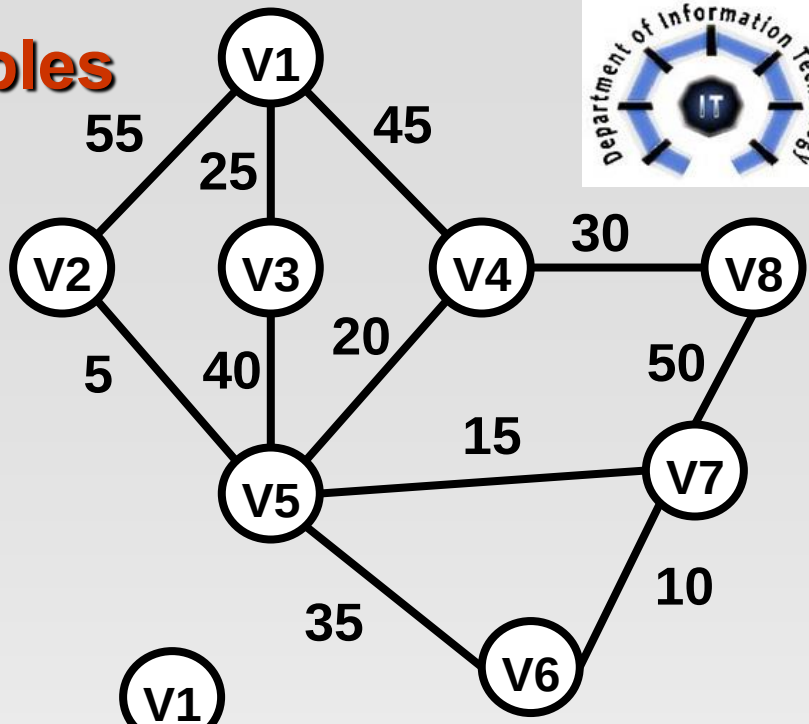
Original Graph G

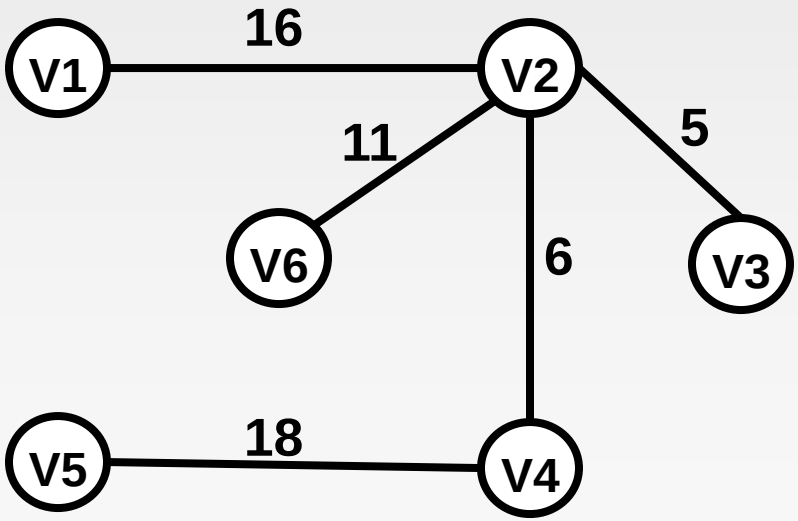


Minimum-Cost Spanning Tree T of G.
Total Cost = 99

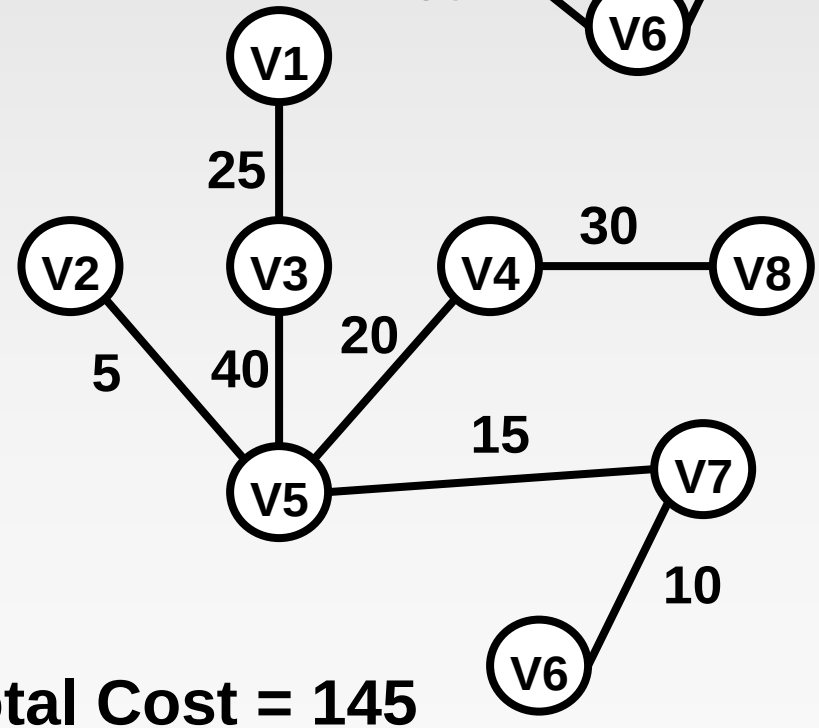
Prim's Algorithm Examples



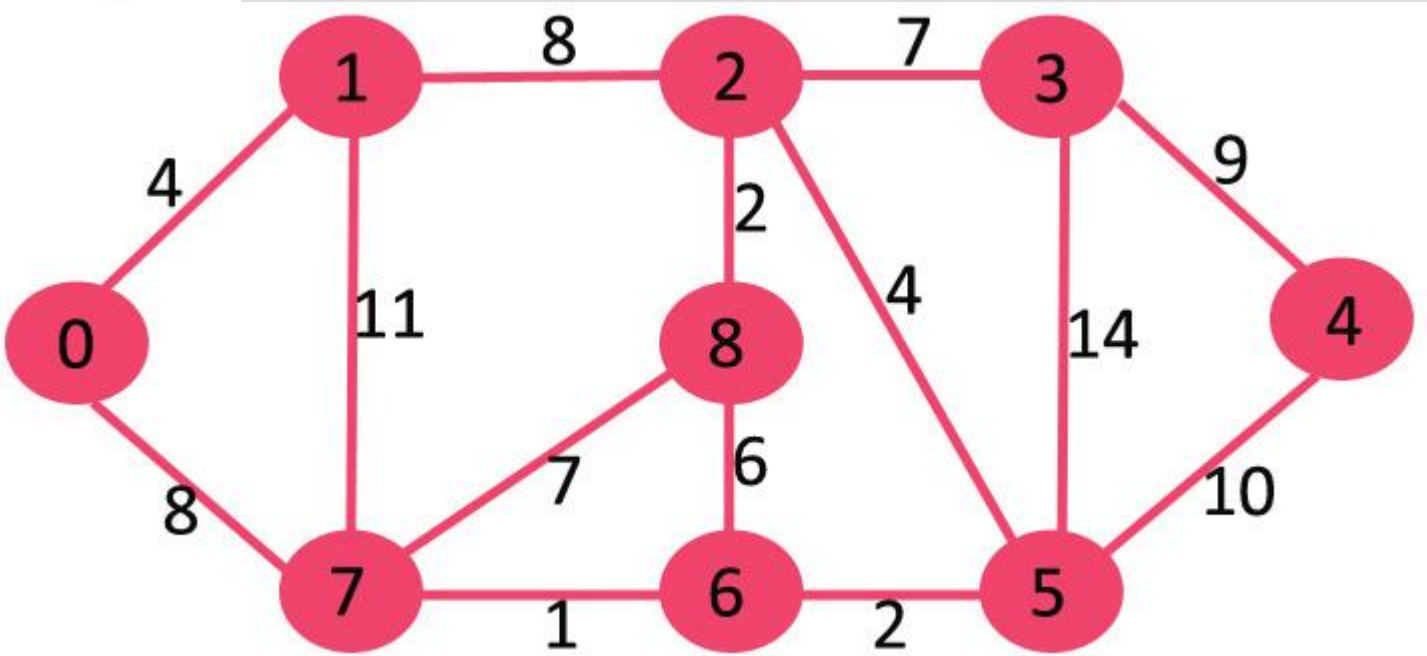




Total Cost = 56



Total Cost = 145



Compute a minimum cost spanning tree using
a) Kruskal's Algorithm
&
b) Prim's Algorithm