

PDFZilla – Unregistered

PDFZilla - Unregistered

PDFZilla - Unregistered

Knapsack problem:

"Given 'n' objects and a knapsack (bag) with capacity 'm', where each object i has a weight " w_i " and profit associated with it " p_i "

- The objective is to obtain a filling of knapsack that maximizes the total profit earned.

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

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

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

$$\text{where } 0 \leq x_i \leq 1 \quad \& \quad 1 \leq i \leq n$$

A feasible solution is any set of x_1, \dots, x_n satisfying ② and ③

- An optimal solution is a feasible solution for which ① is maximized.

$$\begin{aligned} \text{Maximize } & \sum_{i=1}^n p_i x_i + 1.725 \\ \text{Subject to } & \sum_{i=1}^n w_i x_i \leq 100 \\ & 0 \leq x_i \leq 1 \quad \& \quad 1 \leq i \leq 10 \end{aligned}$$

Optimal solution = (1, 0, 1, 0, 1, 0, 1, 0, 1, 0)

① Consider the following instance of Knapsack problem & find all feasible and hence an optimal solution

$$n=3, \quad m=20 \quad (P_1, P_2, P_3) = \{25, 24, 15\}$$

$$(w_1, w_2, w_3) = \{18, 15, 10\}$$

I feasible solution:

- consider decreasing order of profit P_i :
i.e P_1 has highest value = 25,
so place it in knapsack first

Set $x_1 = 1$ & profit $P_1 x_1 = 25 \times 1 = \frac{25}{15}$
is earned.
So now, Knapsack capacity = $20 - 18 = \underline{02}$

Next largest profit is $P_2 = 24$

$$\text{but } w_2 = 15$$

Knapsack capacity = 02, but $w_2 = 15$
which doesn't fit into knapsack.

Hence $\frac{2}{15}$ can fill Knapsack

$$\sum_{i=1 \text{ to } 3} w_i x_i = 18 \times 1 + 15 \times \frac{2}{15} = 20$$

$$\sum_{i=1 \text{ to } 3} P_i x_i = 25 \times 1 + 24 \times \frac{2}{15} = 28.2$$

Solution vector $(x_1, x_2, x_3) = (1, \frac{2}{15}, 0)$

Feasible Solution = 28.2

II feasible solution:

- Consider "increasing order of weight w_i "

i.e object 3 has least weight $w_3 = 10$

Hence place object 3 into knapsack 8

Set $x_3 = 1$ & profit $P_i x_i = P_3 x_3 = 15 \times 1 = 15$

Knapsack capacity reduces to

$$20 - 10 = 10$$

- Next least weight is $w_2 = 15$, but capacity now is 10, so it does not fit

Hence use fraction $\frac{10}{15} = \frac{2}{3}$ to fill knapsack.

$$\text{Now } \sum_{i=1 \text{ to } 3} w_i x_i = 10x_1 + 15 \times \frac{2}{3} = 20$$

$$\sum_{i=1 \text{ to } 3} P_i x_i = 15x_1 + 24 \times \frac{2}{3} = 31$$

Solution Vector $(x_1, x_2, x_3) = (0, \frac{2}{3}, 1)$

and feasible solution = 31

III feasible solution:

Consider decreasing order of ratio P_i/w_i

$$P_1/w_1 = \frac{25}{18} = 1.38$$

$$P_2/w_2 = \frac{24}{15} = 1.6$$

$$P_3/w_3 = 15/10 = 1.5$$

- According to decreasing order of P_i/w_i first consider object 2 with weight 15 & place it in knapsack & set $x_2 = 1$

& profit $P_i x_i = 24 x_1 = 24$ is earned
capacity reduces to $20 - 15 = 5$

- Next object 3 is considered i.e. $w_3 = 10$.
which does not fit into knapsack and hence a fraction $5/10 = 1/2$ is used to fill knapsack. $\therefore x_3 = 1/2$

$$\text{Now } \sum_{i=1 \text{ to } 3} w_i x_i = 15x_1 + 10x_{1/2} = 20$$

$$\sum_{i=1 \text{ to } 3} P_i x_i = 24x_1 + 15x_{1/2} = 31.5$$

Solution Vector $(x_1, x_2, x_3) = (0, 1, 1/2)$
Feasible Solution = 31.5

- Now comparing all three feasible solution, it can be concluded that the 3rd feasible solution i.e. 31.5 is maximum

- Hence the optimal solution is 31.5 with solution vector (x_1, x_2, x_3)

$$= \underline{(0, 1, 1/2)}$$

* obtain the optimal solution for the following instance of knapsack problem.

① $m = 40$ $n = 3$

$$w_i = \{20, 25, 10\} \quad P_i = \{30, 40, 35\}$$

$$1 \leq i \leq 3$$

$$82.5$$

② $n = 3, m = 20, w_i = \{18, 15, 10\} P_i = \{30, 21, 18\}$

$$32.8$$

③ $n = 7, m = 15, w_i = \{2, 3, 5, 7, 1, 4, 1\}$

$$P_i = \{10, 5, 15, 7, 6, 18, 3\}$$

$$55.25$$

Algorithm Greedy-knapsack (m, n)

// finds the solution vector

// Input: m , the capacity of knapsack,
 n , number of objects
 w_i the weights of n objects

// Output: the solution vector x

{

for $i=1$ to n do $x[i] = 0.0$;

$u = m$;

for $i=1$ to n do

{

if ($w[i] > u$) then break;

$x[i] = 1.0$;

$u = u - w[i]$;

}

if ($i \leq n$) then

$x[i] = u / w[i]$;

}

Solve the following Strassen's Multiplication:

$$A = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix}$$

$$B = \begin{bmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{bmatrix}$$

$$P = (A_{11} + A_{22})(B_{11} + B_{22}) = (2+5)(1+7) = 7*8 = 56$$

$$Q = (A_{21} + A_{22})B_{11} = (3+5)1 = 8$$

$$R = A_{11}(B_{12} - B_{22}) = 2(3-7) = 2 * (-4) = -8$$

$$S = A_{22}(B_{21} - B_{11}) = 5(4-1) = 5*3 = 15$$

$$T = (A_{11} + A_{12})B_{22} = (2+4)7 = 6+7 = 42$$

$$U = (A_{21} - A_{11})(B_{11} + B_{12}) = (3-2)(1+3) = 1*4 = 4$$

$$V = (A_{12} - A_{22})(B_{21} + B_{22}) = (4-5)(4+7) = -1*11 = -11$$

$$C_{11} = P + S - T + V = 56 + 15 - 42 - 11 = 18$$

$$C_{12} = R + T = -8 + 42 = 34$$

$$C_{21} = Q + S = 8 + 15 = 23$$

$$C_{22} = P + R - Q + U = 56 - 8 - 8 + 4 = 44$$

$$C = \begin{bmatrix} 18 & 34 \\ 23 & 44 \end{bmatrix}$$

Solve the Strassen's Multiplication:

$$A = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} = \begin{bmatrix} 1 & 0 & 2 & 1 \\ 4 & 1 & 1 & 0 \\ \hline 0 & 1 & 3 & 0 \\ 5 & 0 & 2 & 1 \end{bmatrix}$$

$$B = \begin{bmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 & 1 \\ 2 & 1 & 0 & 4 \\ \hline 2 & 0 & 1 & 1 \\ 1 & 3 & 5 & 0 \end{bmatrix}$$

$$P = (A_{11} + A_{22}) * (B_{11} + B_{22})$$

$$= \left[\begin{bmatrix} 1 & 0 \\ 4 & 1 \end{bmatrix} + \begin{bmatrix} 3 & 0 \\ 2 & 1 \end{bmatrix} \right] * \left[\begin{bmatrix} 0 & 1 \\ 2 & 1 \end{bmatrix} + \begin{bmatrix} 1 & 1 \\ 5 & 0 \end{bmatrix} \right]$$

$$= \begin{bmatrix} 4 & 0 \\ 6 & 2 \end{bmatrix} * \begin{bmatrix} 1 & 2 \\ 7 & 1 \end{bmatrix} = \begin{bmatrix} 4 & 8 \\ 20 & 14 \end{bmatrix}$$

$$Q = (A_{21} + A_{22}) * B_{11}$$

$$= \left[\begin{bmatrix} 0 & 1 \\ 5 & 0 \end{bmatrix} + \begin{bmatrix} 3 & 0 \\ 2 & 1 \end{bmatrix} \right] * \begin{bmatrix} 0 & 1 \\ 2 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} 3 & 1 \\ 7 & 1 \end{bmatrix} * \begin{bmatrix} 0 & 1 \\ 2 & 1 \end{bmatrix} = \begin{bmatrix} 2 & 4 \\ 2 & 8 \end{bmatrix}$$

$$R = A_{11} * [B_{12} - B_{22}]$$

$$= \begin{bmatrix} 1 & 0 \\ 4 & 1 \end{bmatrix} * \begin{bmatrix} -1 & 0 \\ -5 & 4 \end{bmatrix} = \begin{bmatrix} -1 & 0 \\ -9 & 4 \end{bmatrix}$$

$$S = A_{22} * [B_{21} - B_{11}]$$

$$= \begin{bmatrix} 3 & 0 \\ 2 & 1 \end{bmatrix} * \begin{bmatrix} 2 & -1 \\ -1 & 2 \end{bmatrix} = \begin{bmatrix} 6 & -3 \\ 3 & 0 \end{bmatrix}$$

$$T = [A_{11} + A_{12}] B_{22}$$

$$= \begin{bmatrix} 3 & 1 \\ 5 & 1 \end{bmatrix} * \begin{bmatrix} 1 & 1 \\ 5 & 0 \end{bmatrix} = \begin{bmatrix} 8 & 3 \\ 10 & 5 \end{bmatrix}$$

$$U = (A_{21} - A_{11}) * (B_{11} + B_{12})$$

$$= \begin{bmatrix} -1 & 1 \\ 1 & -1 \end{bmatrix} * \begin{bmatrix} 0 & 2 \\ 2 & 5 \end{bmatrix} = \begin{bmatrix} 2 & 3 \\ -2 & -3 \end{bmatrix}$$

$$V = (A_{12} - A_{22}) * (B_{21} + B_{22}) =$$

$$= \begin{bmatrix} -1 & 1 \\ -1 & -1 \end{bmatrix} * \begin{bmatrix} 3 & 1 \\ 6 & 3 \end{bmatrix} = \begin{bmatrix} 3 & 2 \\ 9 & -4 \end{bmatrix}$$

$$C_{11} = P + S - T + V$$

$$= \underbrace{\begin{bmatrix} 4 & 8 \\ 20 & 14 \end{bmatrix}}_{=} + \underbrace{\begin{bmatrix} 6 & -3 \\ 3 & 0 \end{bmatrix}}_{=} - \underbrace{\begin{bmatrix} 8 & 3 \\ 10 & 5 \end{bmatrix}}_{=} + \underbrace{\begin{bmatrix} 3 & 2 \\ -9 & -4 \end{bmatrix}}_{=}$$

$$= \begin{bmatrix} 10 & 5 \\ 23 & 14 \end{bmatrix} - \begin{bmatrix} 8 & 3 \\ 10 & 5 \end{bmatrix} + \begin{bmatrix} 3 & 2 \\ -9 & -4 \end{bmatrix}$$

$$= \begin{bmatrix} 02 & 2 \\ 13 & 9 \end{bmatrix} + \begin{bmatrix} 3 & 2 \\ -9 & -4 \end{bmatrix}$$

$$= \begin{bmatrix} 5 & 4 \\ 4 & 5 \end{bmatrix}$$

$$C_{12} = R + T = \begin{bmatrix} -1 & 0 \\ -9 & 4 \end{bmatrix} + \begin{bmatrix} 8 & 3 \\ 10 & 5 \end{bmatrix} = \begin{bmatrix} 7 & 3 \\ 1 & 9 \end{bmatrix}$$

$$C_{21} = Q + S = \begin{bmatrix} 2 & 4 \\ 2 & 8 \end{bmatrix} + \begin{bmatrix} 6 & -3 \\ 3 & 0 \end{bmatrix} = \begin{bmatrix} 8 & 1 \\ 5 & 8 \end{bmatrix}$$

$$C_{22} = P + R - Q + U = \underbrace{\begin{bmatrix} 4 & 8 \\ 20 & 14 \end{bmatrix} + \begin{bmatrix} -1 & 0 \\ -9 & 4 \end{bmatrix}}_{\begin{bmatrix} 3 & 8 \\ 19 & 18 \end{bmatrix}} - \begin{bmatrix} 2 & 4 \\ 2 & 8 \end{bmatrix} + \begin{bmatrix} 2 & 3 \\ -2 & -3 \end{bmatrix}$$

$$= \begin{bmatrix} 1 & 4 \\ 9 & 10 \end{bmatrix} + \begin{bmatrix} 2 & 3 \\ -2 & -3 \end{bmatrix}$$

$$= \begin{bmatrix} 3 & 7 \\ 7 & 7 \end{bmatrix}$$

$$C = \begin{bmatrix} 5 & 4 & 7 & 3 \\ 4 & 5 & 1 & 9 \\ 8 & 1 & 3 & 7 \\ 5 & 8 & 7 & 7 \end{bmatrix}$$

Advantages & Disadvantages of Divide & Conquer Paradigm:

Advantages:

1. Solves difficult problems.
2. It reduces the degree of difficulty, since it divides the problem into sub problems that are easily solvable and usually runs faster than other algorithms would.
3. It also uses memory caches effectively. As when sub problems become simple, it can be solved within cache, without having to access the slower main memory, which saves time and makes the algorithm more efficient.
4. parallelism - It is suitable for multiprocessor machines.

Disadvantages:

1. Divide and conquer strategy uses recursion that makes it a slower and if a small/little error occurs in the code, the program may enter into an infinite loop.
2. Usage of explicit stacks make use of extra space.

Decrease and Conquer:

- Decrease and conquer technique is based on exploiting the relationship between a solution to a given instance of a problem and solution to a smaller instance of same problem.
- Once such relationship is established, it can be exploited either top down (recursively) or bottom up (without a recursion).
- There are three major variations of decrease-and-conquer:
 - Decrease by constant
 - Decrease by constant factor
 - Variable size decrease

Decrease by constant:

The size of instance is reduced by the same constant on each iteration of the algorithm.

- Consider an example, exponential problem of computing a^n for positive integer ~~const~~ exponents.

$$a^n = a^{n-1} \times a$$

$$\text{So } f(n) = a^n$$

$$f(n) = \begin{cases} f(n-1) \cdot a & \text{if } n \geq 1 \\ a & \text{if } n = 1 \end{cases}$$

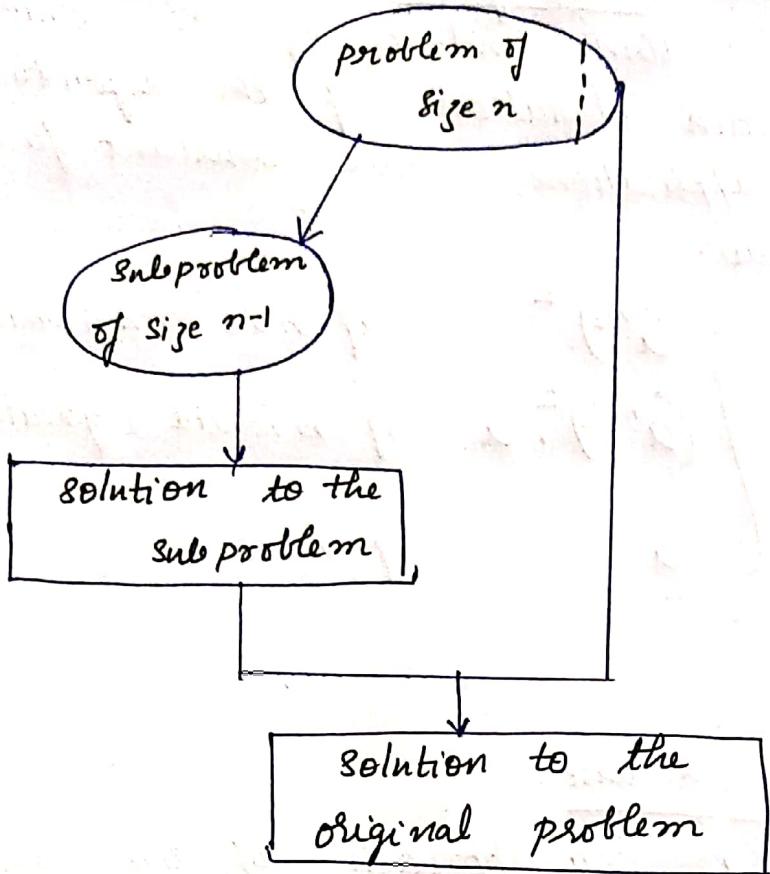
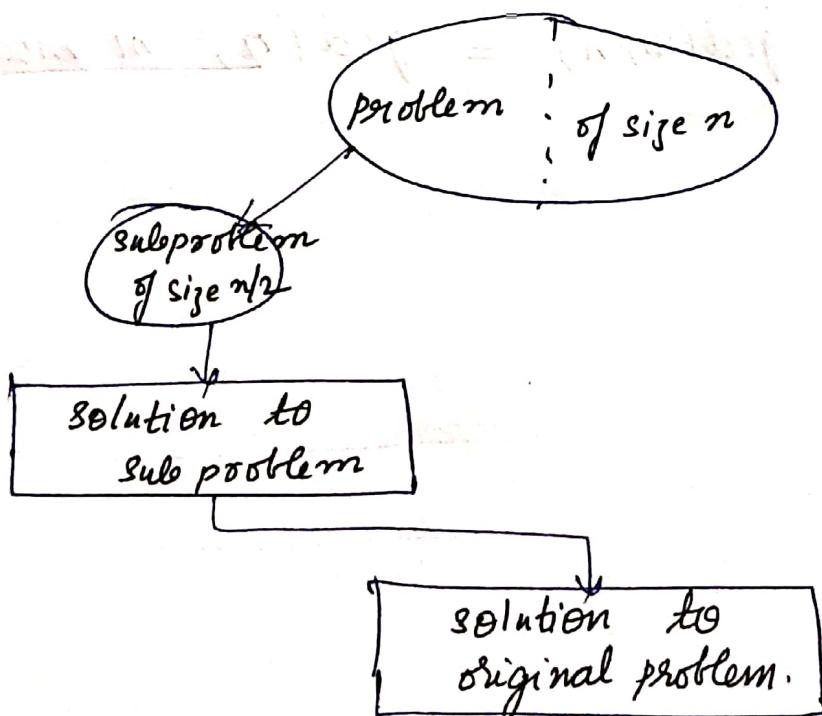


Figure: Decrease & conquer technique.

$f(n) = a^n$ can be computed either "top-down" by using its recursive definition or 'bottom-up' by multiplying a by itself $n-1$ times.

Decrease by constant factor:



- Decrease by a constant factor technique suggests reducing a problem instance by the same constant factor on each iteration of the algorithm. In most applications, this constant factor is equal to two.

Ex:

$$a^n = \begin{cases} (a^{n/2})^2 & \text{if } n \text{ is even \& positive} \\ (a^{(n-1)/2})^2 \cdot a & \text{if } n \text{ is odd \& greater than 1} \\ a & \text{if } n=1 \end{cases}$$

variable size decrease:

- It is a approach of size reduction of instance input 'n'.
- Reduction pattern varies from one iteration of an algorithm to another

Ex:

Euclid compute of GCD

$$\gcd(m, n) = \gcd(n, m \bmod n)$$

Topological Sorting:

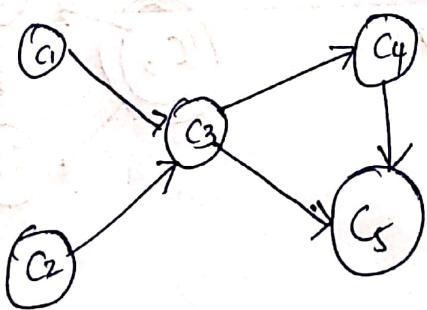
4

Problem statement:

Given a digraph $G = (V, E)$, the task is to find a linear ordering of vertices such that for any edge (u, v) in E , u precedes v in the ordering.

For topological sorting, the graph should be a DAG (Directed Acyclic Graph).

- Consider an example of set of five required courses $\{c_1, c_2, c_3, c_4, c_5\}$ a part time student has to take in some degree program. The course can be taken in any order as long as following course pre-requisites are met: c_1, c_2 has no prerequisites, c_3 requires c_1, c_2 , c_4 requires c_3 , c_5 requires $c_3 \& c_4$. The student can take only one course per term. In which order should the student take the courses?



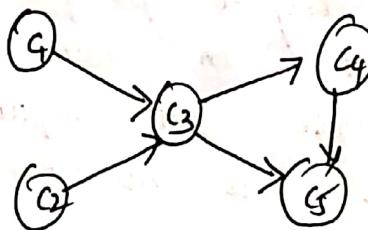
The situation can be modeled by a digraph in which vertices represent courses and directed edges indicate prerequisite requirements.

- This problem is solved by topological sorting.
- Two techniques to solve are:
 - Source Removal Algorithm
 - DFS

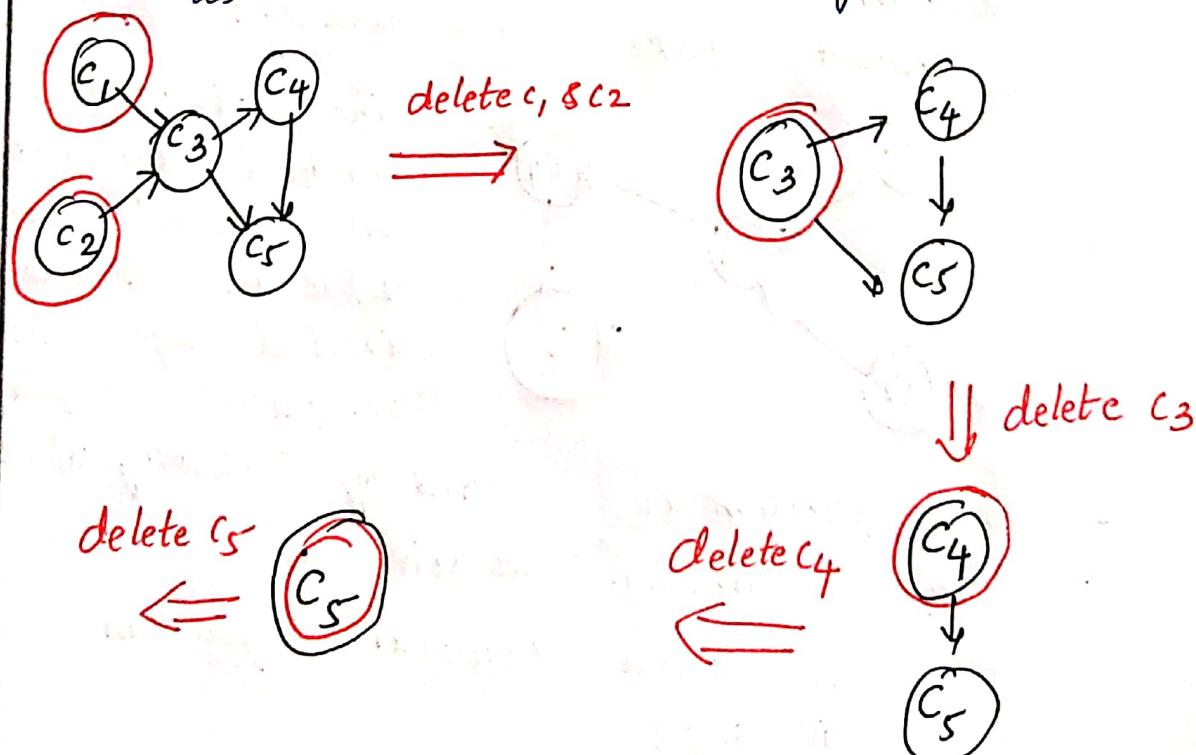
Source-Removal Algorithm:

- In a given digraph, identify a "source", a vertex with no incoming edges and delete it along with all the edges outgoing from it.
[if there are no source vertex, then problem cannot be solved]
- The order in which the vertices are deleted yields a solution to the topological sorting problem.

Ex:



Sol In a Graph given, the source is c_1 , as it has no incoming edge!

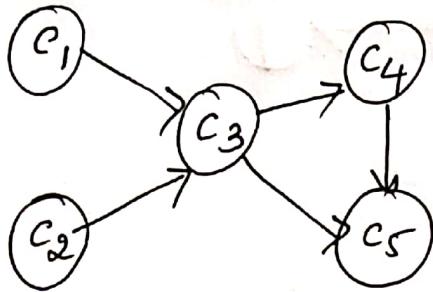


Topological order is $\langle c_1, c_2, c_3, c_4, c_5 \rangle$

(ii)

DFS Based Method:

For a given digraph, perform DFS traversal & note the order in which vertices become deadends. Reversing this order results a topological ordering.



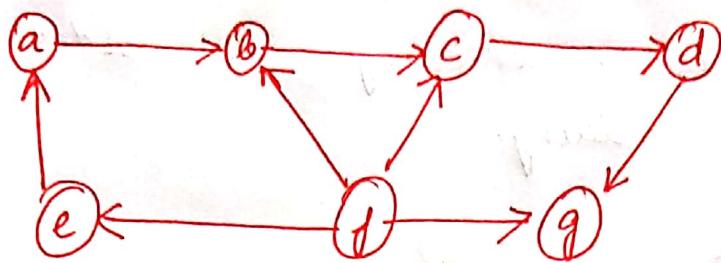
DFS traversal is performed as

C₅ 1
C₄ 2
C₃ 3
C₁ 4
C₂ 5

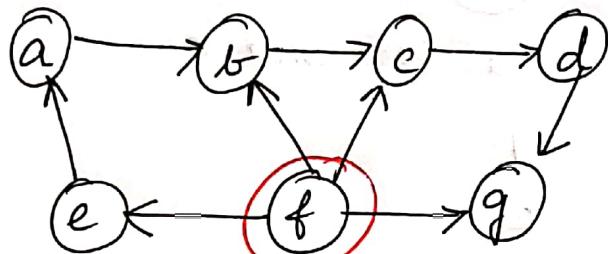
popping off order is C₅, C₄, C₃, C₁, C₂

the topological order is < C₂, C₁, C₃, C₄, C₅ >

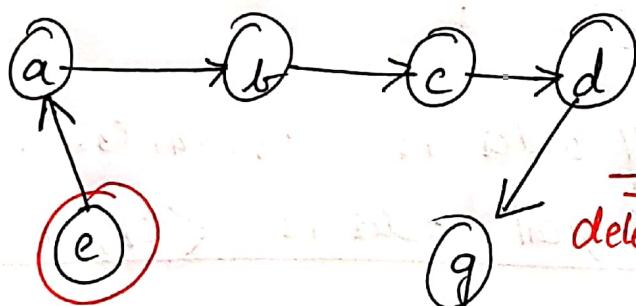
obtain the topological ordering for graph given below:



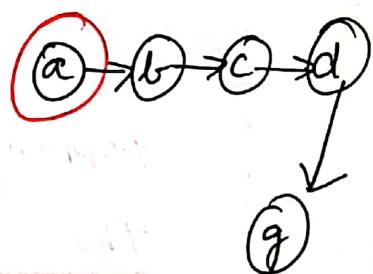
Source removal Algo:



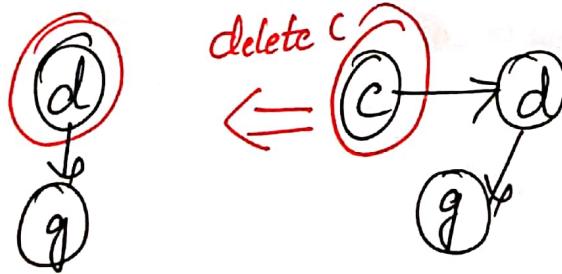
↓ delete f



↓ delete e



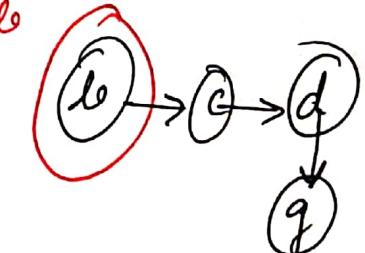
↓ delete a



↓ delete d

← delete b

← delete c



→ delete g

topological sorting = $\{f, e, a, b, c, d, g\}$

3.2

Job sequencing with deadlines:

Given a set of 'n' jobs,

deadline d_i for each job i ($d_i \geq 0$)

profit $p_i > 0$ for each job

- For any job i , the profit p_i is earned iff the job is completed by its deadline.
- only one machine is available for processing
- A job is completed, when it gets executed on a machine for one unit of time.
- A feasible solution for this problem is a subset T of jobs, such that each job in this subset can be completed by its deadline.
- An optimal solution is a feasible solution with maximum value.

Greedy Approach

1. Sort all jobs in decreasing order of profit.
2. Initialize the result sequence as first in sorted jobs, as late as possible
3. Do the following for remaining $n-1$ jobs:
 - If the current job can fit in the current result sequence without missing the deadline, add current job to the

-result, else ignore the current job.

Ex: solve the below instance of job sequencing with deadlines problem and obtain the optimal solution:

Let $n = 4$

$$P_i = \{100, 10, 15, 27\} \quad 1 \leq i \leq 4$$

$$d_i = \{2, 1, 2, 1\} \quad 1 \leq i \leq 4$$

Step 1

Sort all jobs in decreasing order of profit

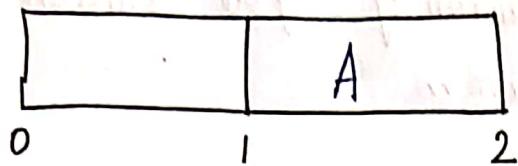
job	A	B	C	D
P_i	100	10	15	27
d_i	2	1	2	1

↓ After sorting.

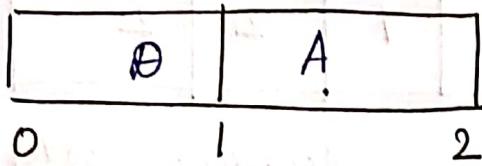
job	A	D	C	B
P_i	100	27	15	10
d_i	2	1	2	1

Step 2 - Initialize Resultant as 1st Job and schedule as late as possible

Draw Gant chart



Step 3 : Repeat the same for others:



- other jobs are Rejected as they won't fit

∴ the selected job set

$$J = \{ D, A \}$$

$$\text{profit earned} = 27 + 100 = 127$$



2. solve the following instance of jobs

Let $n=5$, profits $(10, 3, 33, 11, 40)$ and deadlines $(3, 1, 1, 2, 2)$. Find the optimal sequence of execution of job solution using greedy algorithm

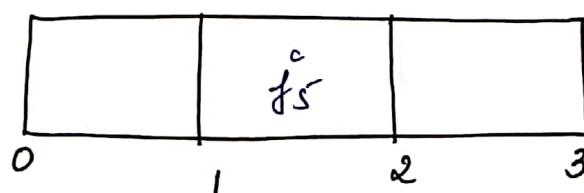
i	j_1	j_2	j_3	j_4	j_5
p_i	10	3	33	11	40
d_i	3	1	1	2	2

Step 1: Sort it in Ascending order

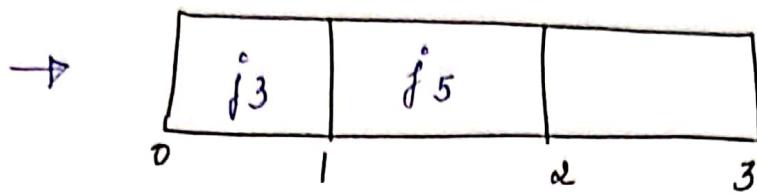
	j_5	j_3	j_4	j_1	j_2
p_i	40	33	11	10	3
d_i	2	1	2	3	1

Step 2: Initialize Result as 1st job and schedule as late as possible.

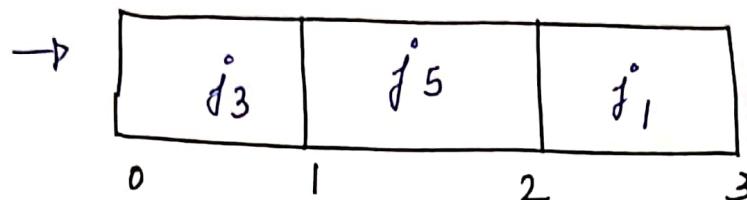
Draw Gantt chart



Step 3 : Repeat the same for others:



j_4 is rejected as its deadline is 2, consider j_1



- j_2 is rejected as the sequence is full.

∴ the selected job set:

$$J = \{j_3, j_5, j_1\}$$

$$\text{profit earned} = 33 + 40 + 10 = \boxed{83}$$

3. solve the following instance of job sequencing with deadlines problem:

$$\begin{array}{ccccccc} p_i & 20 & 15 & 10 & 5 & 1 \\ d_i & 2 & 2 & 1 & 3 & 3 \end{array}$$

$$\frac{S_0}{n} = \{J_1, J_2, J_4\}$$

$$20 + 15 + 5 = 40$$

Solve the following Strassen's Multiplication:

$$A = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix}$$

$$B = \begin{bmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{bmatrix}$$

$$P = (A_{11} + A_{22})(B_{11} + B_{22}) = (2+5)(1+7) = 7*8 = 56$$

$$Q = (A_{21} + A_{22})B_{11} = (3+5)1 = 8$$

$$R = A_{11}(B_{12} - B_{22}) = 2(3-7) = 2 * (-4) = -8$$

$$S = A_{22}(B_{21} - B_{11}) = 5(4-1) = 5*3 = 15$$

$$T = (A_{11} + A_{12})B_{22} = (2+4)7 = 6+7 = 42$$

$$U = (A_{21} - A_{11})(B_{11} + B_{12}) = (3-2)(1+3) = 1*4 = 4$$

$$V = (A_{12} - A_{22})(B_{21} + B_{22}) = (4-5)(4+7) = -1*11 = -11$$

$$C_{11} = P + S - T + V = 56 + 15 - 42 - 11 = 18$$

$$C_{12} = R + T = -8 + 42 = 34$$

$$C_{21} = Q + S = 8 + 15 = 23$$

$$C_{22} = P + R - Q + U = 56 - 8 - 8 + 4 = 44$$

$$C = \begin{bmatrix} 18 & 34 \\ 23 & 44 \end{bmatrix}$$

Solve the Strassen's Multiplication:

$$A = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} = \begin{bmatrix} 1 & 0 & 2 & 1 \\ 4 & 1 & 1 & 0 \\ \hline 0 & 1 & 3 & 0 \\ 5 & 0 & 2 & 1 \end{bmatrix}$$

$$B = \begin{bmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 & 1 \\ 2 & 1 & 0 & 4 \\ \hline 2 & 0 & 1 & 1 \\ 1 & 3 & 5 & 0 \end{bmatrix}$$

$$P = (A_{11} + A_{22}) * (B_{11} + B_{22})$$

$$= \left[\begin{bmatrix} 1 & 0 \\ 4 & 1 \end{bmatrix} + \begin{bmatrix} 3 & 0 \\ 2 & 1 \end{bmatrix} \right] * \left[\begin{bmatrix} 0 & 1 \\ 2 & 1 \end{bmatrix} + \begin{bmatrix} 1 & 1 \\ 5 & 0 \end{bmatrix} \right]$$

$$= \begin{bmatrix} 4 & 0 \\ 6 & 2 \end{bmatrix} * \begin{bmatrix} 1 & 2 \\ 7 & 1 \end{bmatrix} = \begin{bmatrix} 4 & 8 \\ 20 & 14 \end{bmatrix}$$

$$Q = (A_{21} + A_{22}) * B_{11}$$

$$= \left[\begin{bmatrix} 0 & 1 \\ 5 & 0 \end{bmatrix} + \begin{bmatrix} 3 & 0 \\ 2 & 1 \end{bmatrix} \right] * \begin{bmatrix} 0 & 1 \\ 2 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} 3 & 1 \\ 7 & 1 \end{bmatrix} * \begin{bmatrix} 0 & 1 \\ 2 & 1 \end{bmatrix} = \begin{bmatrix} 2 & 4 \\ 2 & 8 \end{bmatrix}$$

$$R = A_{11} * [B_{12} - B_{22}]$$

$$= \begin{bmatrix} 1 & 0 \\ 4 & 1 \end{bmatrix} * \begin{bmatrix} -1 & 0 \\ -5 & 4 \end{bmatrix} = \begin{bmatrix} -1 & 0 \\ -9 & 4 \end{bmatrix}$$

$$S = A_{22} * [B_{21} - B_{11}]$$

$$= \begin{bmatrix} 3 & 0 \\ 2 & 1 \end{bmatrix} * \begin{bmatrix} 2 & -1 \\ -1 & 2 \end{bmatrix} = \begin{bmatrix} 6 & -3 \\ 3 & 0 \end{bmatrix}$$

$$T = [A_{11} + A_{12}] B_{22}$$

$$= \begin{bmatrix} 3 & 1 \\ 5 & 1 \end{bmatrix} * \begin{bmatrix} 1 & 1 \\ 5 & 0 \end{bmatrix} = \begin{bmatrix} 8 & 3 \\ 10 & 5 \end{bmatrix}$$

$$U = (A_{21} - A_{11}) * (B_{11} + B_{12})$$

$$= \begin{bmatrix} -1 & 1 \\ 1 & -1 \end{bmatrix} * \begin{bmatrix} 0 & 2 \\ 2 & 5 \end{bmatrix} = \begin{bmatrix} 2 & 3 \\ -2 & -3 \end{bmatrix}$$

$$V = (A_{12} - A_{22}) * (B_{21} + B_{22}) =$$

$$= \begin{bmatrix} -1 & 1 \\ -1 & -1 \end{bmatrix} * \begin{bmatrix} 3 & 1 \\ 6 & 3 \end{bmatrix} = \begin{bmatrix} 3 & 2 \\ 9 & -4 \end{bmatrix}$$

$$C_{11} = P + S - T + V$$

$$= \underbrace{\begin{bmatrix} 4 & 8 \\ 20 & 14 \end{bmatrix}}_{=} + \underbrace{\begin{bmatrix} 6 & -3 \\ 3 & 0 \end{bmatrix}}_{=} - \underbrace{\begin{bmatrix} 8 & 3 \\ 10 & 5 \end{bmatrix}}_{=} + \underbrace{\begin{bmatrix} 3 & 2 \\ -9 & -4 \end{bmatrix}}_{=}$$

$$= \begin{bmatrix} 10 & 5 \\ 23 & 14 \end{bmatrix} - \begin{bmatrix} 8 & 3 \\ 10 & 5 \end{bmatrix} + \begin{bmatrix} 3 & 2 \\ -9 & -4 \end{bmatrix}$$

$$= \begin{bmatrix} 02 & 2 \\ 13 & 9 \end{bmatrix} + \begin{bmatrix} 3 & 2 \\ -9 & -4 \end{bmatrix}$$

$$= \begin{bmatrix} 5 & 4 \\ 4 & 5 \end{bmatrix}$$

$$C_{12} = R + T = \begin{bmatrix} -1 & 0 \\ -9 & 4 \end{bmatrix} + \begin{bmatrix} 8 & 3 \\ 10 & 5 \end{bmatrix} = \begin{bmatrix} 7 & 3 \\ 1 & 9 \end{bmatrix}$$

$$C_{21} = Q + S = \begin{bmatrix} 2 & 4 \\ 2 & 8 \end{bmatrix} + \begin{bmatrix} 6 & -3 \\ 3 & 0 \end{bmatrix} = \begin{bmatrix} 8 & 1 \\ 5 & 8 \end{bmatrix}$$

$$C_{22} = P + R - Q + U = \underbrace{\begin{bmatrix} 4 & 8 \\ 20 & 14 \end{bmatrix} + \begin{bmatrix} -1 & 0 \\ -9 & 4 \end{bmatrix}}_{\begin{bmatrix} 3 & 8 \\ 19 & 18 \end{bmatrix}} - \underbrace{\begin{bmatrix} 2 & 4 \\ 2 & 8 \end{bmatrix}}_{\begin{bmatrix} 2 & 4 \\ 2 & 8 \end{bmatrix}} + \begin{bmatrix} 2 & 3 \\ -2 & -3 \end{bmatrix}$$

$$= \begin{bmatrix} 1 & 4 \\ 9 & 10 \end{bmatrix} + \begin{bmatrix} 2 & 3 \\ -2 & -3 \end{bmatrix}$$

$$= \begin{bmatrix} 3 & 7 \\ 7 & 7 \end{bmatrix}$$

$$C = \begin{bmatrix} 5 & 4 & 7 & 3 \\ 4 & 5 & 1 & 9 \\ 8 & 1 & 3 & 7 \\ 5 & 8 & 7 & 7 \end{bmatrix}$$

Advantages & Disadvantages of Divide & Conquer Paradigm:

Advantages:

1. Solves difficult problems.
2. It reduces the degree of difficulty, since it divides the problem into sub problems that are easily solvable and usually runs faster than other algorithms would.
3. It also uses memory caches effectively. As when sub problems become simple, it can be solved within cache, without having to access the slower main memory, which saves time and makes the algorithm more efficient.
4. parallelism - It is suitable for multiprocessor machines.

Disadvantages:

1. Divide and conquer strategy uses recursion that makes it a slower and if a small/little error occurs in the code, the program may enter into an infinite loop.
2. Usage of explicit stacks make use of extra space.

Decrease and Conquer:

- Decrease and conquer technique is based on exploiting the relationship between a solution to a given instance of a problem and solution to a smaller instance of same problem.
- Once such relationship is established, it can be exploited either top down (recursively) or bottom up (without a recursion).
- There are three major variations of decrease-and-conquer:
 - Decrease by constant
 - Decrease by constant factor
 - Variable size decrease

Decrease by constant:

The size of instance is reduced by the same constant on each iteration of the algorithm.

- Consider an example, exponential problem of computing a^n for positive integer ~~const~~ exponents.

$$a^n = a^{n-1} \times a$$

$$\text{So } f(n) = a^n$$

$$f(n) = \begin{cases} f(n-1) \cdot a & \text{if } n \geq 1 \\ a & \text{if } n = 1 \end{cases}$$

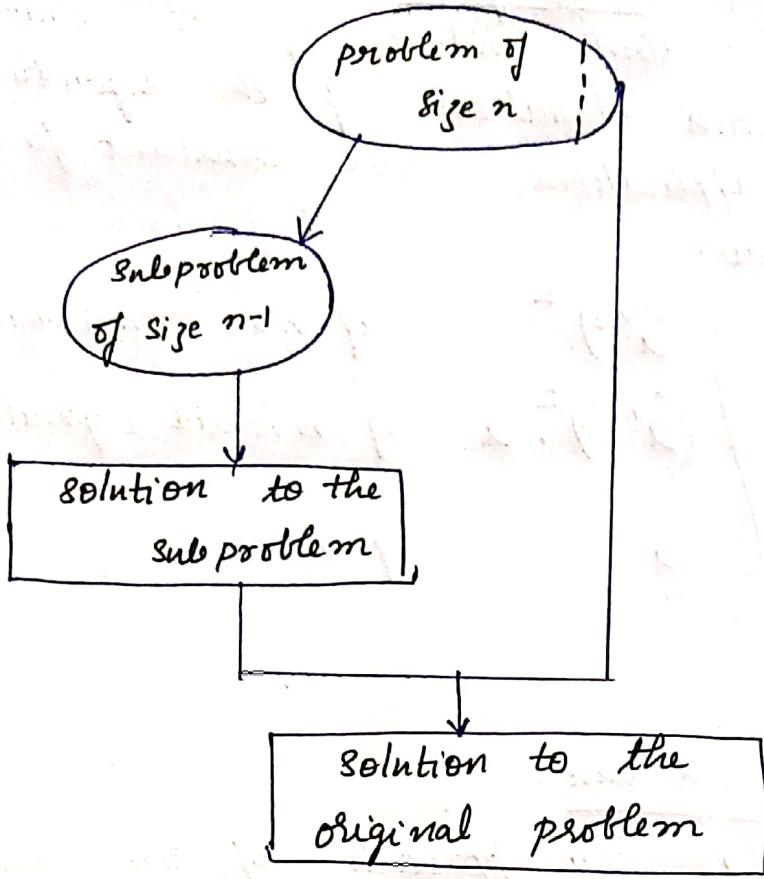
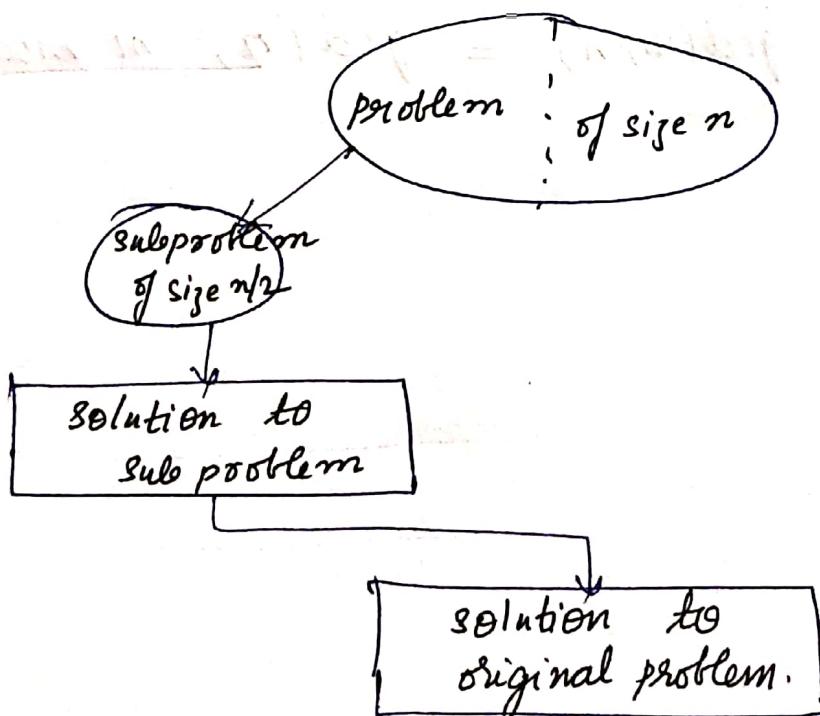


Figure: Decrease & conquer technique.

$f(n) = a^n$ can be computed either "top-down" by using its recursive definition or 'bottom-up' by multiplying a by itself $n-1$ times.

Decrease by constant factor:



- Decrease by a constant factor technique suggests reducing a problem instance by the same constant factor on each iteration of the algorithm. In most applications, this constant factor is equal to two.

Ex:

$$a^n = \begin{cases} (a^{n/2})^2 & \text{if } n \text{ is even \& positive} \\ (a^{n-1/2})^2 \cdot a & \text{if } n \text{ is odd \& greater than 1} \\ a & \text{if } n=1 \end{cases}$$

variable size decrease:

- It is a approach of size reduction of instance input 'n'.
- Reduction pattern varies from one iteration of an algorithm to another

Ex:

Euclid compute of GCD

$$\gcd(m, n) = \gcd(n, m \bmod n)$$

Topological Sorting:

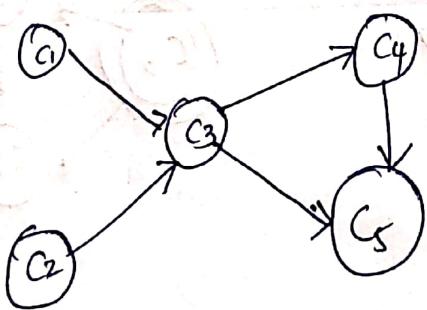
4

Problem statement:

Given a digraph $G = (V, E)$, the task is to find a linear ordering of vertices such that for any edge (u, v) in E , u precedes v in the ordering.

For topological sorting, the graph should be a DAG (Directed Acyclic Graph).

- Consider an example of set of five required courses $\{c_1, c_2, c_3, c_4, c_5\}$ a part time student has to take in some degree program. The course can be taken in any order as long as following course pre-requisites are met: c_1, c_2 has no prerequisites, c_3 requires c_1, c_2 , c_4 requires c_3 , c_5 requires $c_3 \& c_4$. The student can take only one course per term. In which order should the student take the courses?



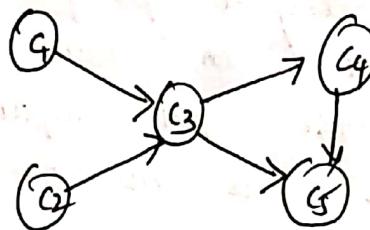
The situation can be modeled by a digraph in which vertices represent courses and directed edges indicate prerequisite requirements.

- This problem is solved by topological sorting.
- Two techniques to solve are:
 - Source Removal Algorithm
 - DFS

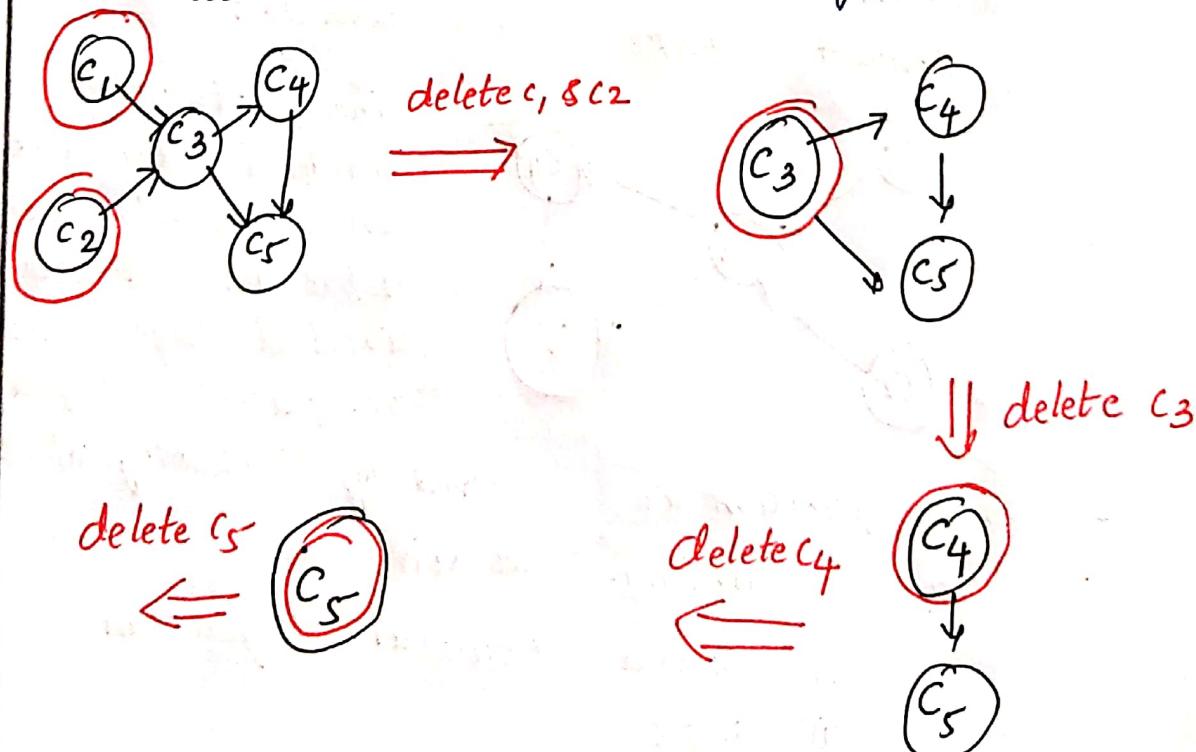
Source-Removal Algorithm:

- In a given digraph, identify a "source", a vertex with no incoming edges and delete it along with all the edges outgoing from it.
[if there are no source vertex, then problem cannot be solved]
- The order in which the vertices are deleted yields a solution to the topological sorting problem.

Ex:



Sol In a Graph given, the source is c_1 , as it has no incoming edge!

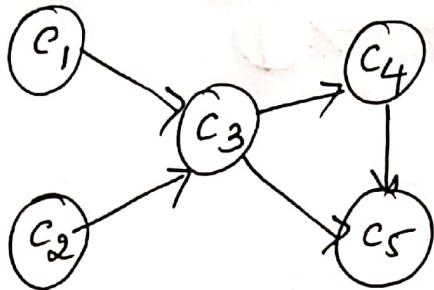


Topological order is $\langle c_1, c_2, c_3, c_4, c_5 \rangle$

(ii)

DFS Based Method:

For a given digraph, perform DFS traversal & note the order in which vertices become deadends. Reversing this order results a topological ordering.



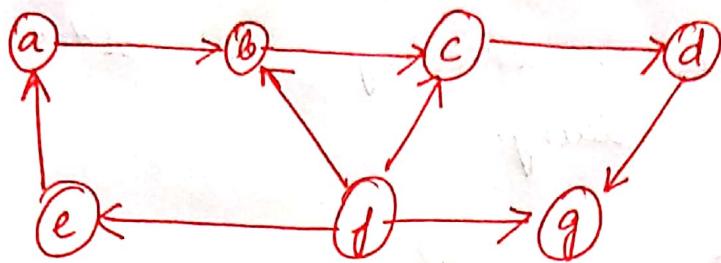
DFS traversal is performed as

C₅ 1
C₄ 2
C₃ 3
C₁ 4
C₂ 5

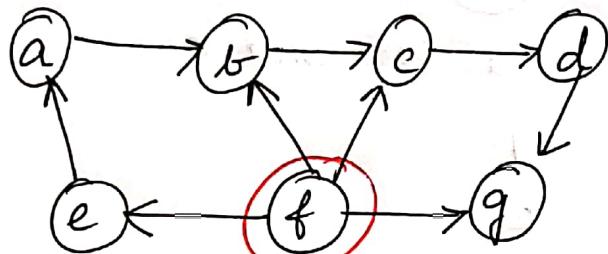
popping off order is C₅, C₄, C₃, C₁, C₂

the topological order is < C₂, C₁, C₃, C₄, C₅ >

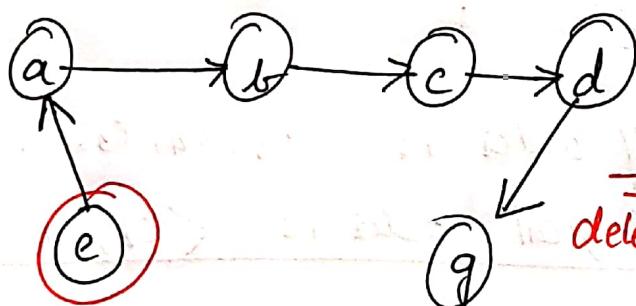
obtain the topological ordering for graph given below:



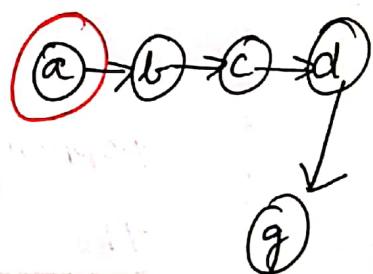
Source removal Algo:



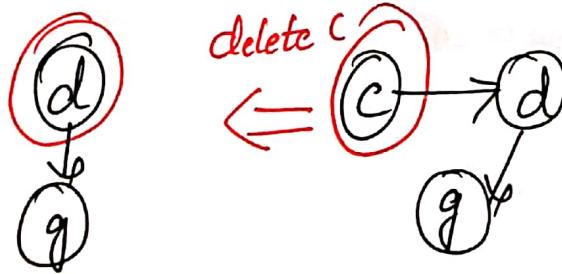
↓ delete f



delete e



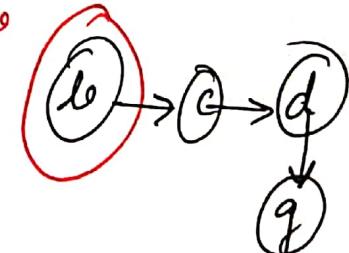
↓ delete a



delete d

delete b

←



←

↓ delete d



→ delete g

topological sorting = {f, e, a, b, c, d, g}

Module - 3

Greedy Approach:

3.1 General Method:

- Greedy technique is one of the straight forward design method which can be applied to a wide variety of problems.
- The problems that are solved using this technique has 'n' inputs and requires us to obtain a subset satisfying some constraints.
- Any subset that satisfy these constraints is called a 'feasible solution'
- The feasible solution that either minimizes or maximizes a given objective function is called 'optimal solution'

Algorithm Greedy (a, n)
// a[1:n] contains 'n' inputs
{

solution = \emptyset

for i=1 to n do

{

x = Select (a);

if Feasible (solution, x) then

solution = union (solution, x);

return solution;

}

- function "select" picks an input from $a[\cdot]$ & assigns to x .
- "Feasible" is a boolean valued function that determines whether " x " can be added to solution vector.
- the function "union" combines ' x ' with the solution & updates the objective function.

* coin-change problem:

It is the problem of finding the number of ways of making change for a particular amount of 'n' cents using a given set of denominations d_1, \dots, d_m .

Eg: Let $n=5$

$$d_i = \{1, 2, 3\}$$

Then following are the different ways of making the change

(i) $\langle 1, 1, 1, 1, 1 \rangle$ (ii) $\langle 1, 1, 1, 2 \rangle$

(iii) $\langle 1, 1, 3 \rangle$ (iv) $\langle 1, 2, 2 \rangle$

(v) $\langle 2, 3 \rangle$

Knapsack problem:

"Given 'n' objects and a knapsack (bag) with capacity 'm', where each object i has a weight " w_i " and profit associated with it " p_i "

- The objective is to obtain a filling of knapsack that maximizes the total profit earned.

i.e., Maximise $\sum p_i x_i$ — (1)

Subject to $\sum w_i x_i \leq m$ — (2)

where $0 \leq x_i \leq 1$ & $1 \leq i \leq n$ — (3)

A feasible solution is any set $\{x_1, \dots, x_n\}$ satisfying (2) and (3)

- An optimal solution is a feasible solution for which (1) is maximized.

① Consider the following instance of Knapsack problem & find all feasible and hence an optimal solution

$$n=3, \quad m=20 \quad (P_1, P_2, P_3) = \{25, 24, 15\}$$

$$(w_1, w_2, w_3) = \{18, 15, 10\}$$

I feasible solution:

- consider decreasing order of profit P_i :
i.e P_1 has highest value = 25,
so place it in knapsack first

set $w_1 x_1 = 1$ & profit $P_1 x_1 = 25 \times 1 = \underline{25}$
so now, knapsack capacity = $20 - 18 = \underline{02}$

Next largest profit is $P_2 = 24$

but $w_2 = 15$

knapsack capacity = 02, but $w_2 = 15$
which doesn't fit into knapsack.

Hence $\frac{2}{15}$ can fill knapsack

$$\sum_{i=1 \text{ to } 3} w_i x_i = 18 \times 1 + 15 \times \frac{2}{15} = 20$$

$$\sum_{i=1 \text{ to } 3} P_i x_i = 25 \times 1 + 24 \times \frac{2}{15} = 28.2$$

Solution vector $(x_1, x_2, x_3) = (1, \frac{2}{15}, 0)$

Feasible Solution = 28.2

II feasible solution:

- Consider "increasing order of weight w_i "

∴ object 3 has least weight $w_3 = 10$

Hence place object 3 into knapsack 8

Set $x_3 = 1$ & profit $P_i x_i = P_3 x_3 = 15 \times 1 = 15$

Knapsack capacity reduces to

$$20 - 10 = 10$$

- Next least weight is $w_2 = 15$, but capacity now is 10, so it does not fit

Hence use fraction $\frac{10}{15} = \frac{2}{3}$ to fill knapsack.

$$\text{Now } \sum_{i=1 \text{ to } 3} w_i x_i = 10 \times 1 + 15 \times \frac{2}{3} = 20$$

$$\sum_{i=1 \text{ to } 3} P_i x_i = 15 \times 1 + 24 \times \frac{2}{3} = 31$$

Solution Vector $(x_1, x_2, x_3) = (0, \frac{2}{3}, 1)$

and feasible solution = 31

III feasible solution:

Consider decreasing order of ratio P_i/w_i

$$P_1/w_1 = \frac{25}{18} = 1.38$$

$$P_2/w_2 = \frac{24}{15} = 1.6$$

$$P_3/w_3 = 15/10 = 1.5$$

- According to decreasing order of P_i/w_i , first consider object 2 with weight 15 & place it in knapsack & set $x_2 = 1$

∴ profit $P_i x_i = 24 x_1 = 24$ is earned
capacity reduces to $20 - 15 = 5$

- Next object 3 is considered i.e. $w_3 = 10$. which does not fit into knapsack and hence a fraction $5/10 = 1/2$ is used to fill knapsack ∴ $x_3 = 1/2$

$$\text{Now } \sum_{i=1 \text{ to } 3} w_i x_i = 15x_1 + 10x_{1/2} = 20$$

$$\sum_{i=1 \text{ to } 3} P_i x_i = 24x_1 + 15x_{1/2} = 31.5$$

Solution vector $(x_1, x_2, x_3) = (0, 1, 1/2)$

Feasible solution = 31.5

- Now comparing all three feasible solution, it can be concluded that the 3rd feasible solution i.e. 31.5 is maximum.

- Hence the optimal solution is 31.5 with solution vector (x_1, x_2, x_3)

$$= \underline{(0, 1, 1/2)}$$

* obtain the optimal solution for the following instance of knapsack problem.

① $m = 40$ $n = 3$

$$w_i = \{20, 25, 10\} \quad P_i = \{30, 40, 35\}$$

$$1 \leq i \leq 3$$

$$82.5$$

② $n = 3, m = 20, w_i = \{18, 15, 10\} \quad P_i = \{30, 21, 18\}$

$$32.8$$

③ $n = 7, m = 15, w_i = \{2, 3, 5, 7, 1, 4, 1\}$

$$P_i = \{10, 5, 15, 7, 6, 18, 3\}$$

$$55.35$$

→ Disregarding the time to initially sort the objects, each of three strategies outline $O(n)$ time.

Theorem: If $\frac{P_1}{W_1} \geq \frac{P_2}{W_2} \geq \frac{P_3}{W_3} \dots \geq \frac{P_n}{W_n}$,

then Greed knapsack generates an optimal solution to the given instance of the knapsack problem.

Algorithm for Greed Knapsack

Greedy Approach

Greedy Approach

Greedy Approach

Algorithm Greedy-Knapsack (m, n)

// finds the solution vector

// Input: m , the capacity of Knapsack,
 n , number of objects &
 w_i the weights of n objects

// Output: the solution vector x

{

// initialize x

for $i=1$ to n do $x[i] = 0.0$;

$u = m$;

for $i=1$ to n do

{

if ($w[i] > u$) then break;

$x[i] = 1.0$;

$u = u - w[i]$;

}

if ($i \leq n$) then

$x[i] = u / w[i]$;

}

3.2

Job sequencing with deadlines:

Given a set of 'n' jobs,

deadline d_i for each job i ($d_i \geq 0$)

profit $p_i > 0$ for each job

- For any job i , the profit p_i is earned iff the job is completed by its deadline.
- only one machine is available for processing
- A job is completed, when it gets executed on a machine for one unit of time.
- A feasible solution for this problem is a subset T of jobs, such that each job in this subset can be completed by its deadline.
- An optimal solution is a feasible solution with maximum value.

Greedy Approach

1. Sort all jobs in decreasing order of profit.
2. Initialize the result sequence as first in sorted jobs, as late as possible
3. Do the following for remaining $n-1$ jobs:
 - If the current job can fit in the current result sequence without missing the deadline, add current job to the

-result, else ignore the current job.

Ex: solve the below instance of job sequencing with deadlines problem and obtain the optimal solution:

Let $n = 4$

$$P_i = \{100, 10, 15, 27\} \quad 1 \leq i \leq 4$$

$$d_i = \{2, 1, 2, 1\} \quad 1 \leq i \leq 4$$

Step 1

Sort all jobs in decreasing order of profit

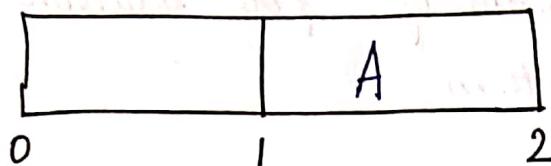
job	A	B	C	D
P_i	100	10	15	27
d_i	2	1	2	1

↓ After sorting.

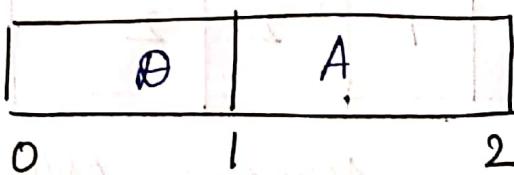
job	A	D	C	B
P_i	100	27	15	10
d_i	2	1	2	1

Step 2 - Initialize Resultant as 1st Job and schedule as late as possible

Draw Gant chart



Step 3 : Repeat the same for others:



- other jobs are Rejected as they won't fit

$$\therefore \text{the selected job set} \\ J = \{D, A\} \\ \text{profit earned} = 27 + 100 = 127$$

2. solve the following instance of job

Let $n=5$, profits $(10, 3, 33, 11, 40)$ and deadlines $(3, 1, 1, 2, 2)$. Find the optimal sequence of execution of job solution using greedy algorithm

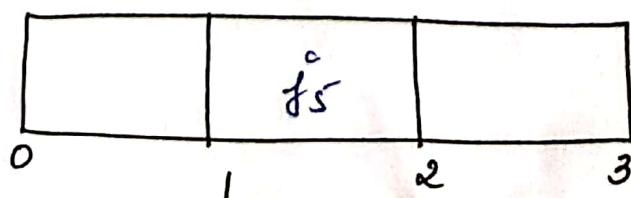
i	j_1	j_2	j_3	j_4	j_5
p_i	10	3	33	11	40
d_i	3	1	1	2	2

Step 1: Sort it in Ascending order

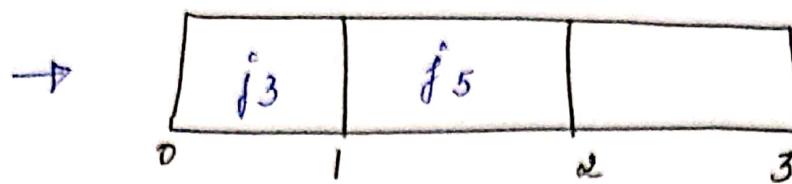
	j_5	j_3	j_4	j_1	j_2
p_i	40	33	11	10	3
d_i	2	1	2	3	1

Step 2: Initialize Result as 1st job and schedule as late as possible.

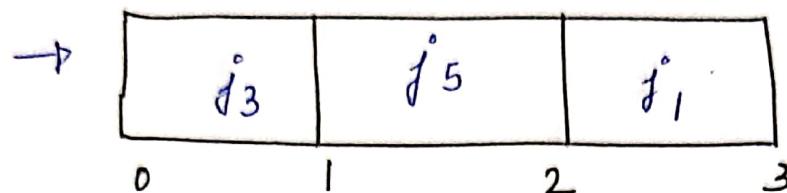
Draw Gantt chart



Step 3: Repeat the same for others.



j_4 is rejected as its deadline is 2, consider j_1



- j_2 is rejected as the sequence is full.

∴ the selected job set:

$$J = \{j_3, j_5, j_1\}$$

$$\text{Profit earned} = 33 + 40 + 10 = 83$$

3. solve the following instance of job sequencing with deadlines problem:

P_i	20	15	10	5	1
d_i	2	2	1	3	3

$$\frac{S_0}{n} = \{J_1, J_2, J_4\}$$

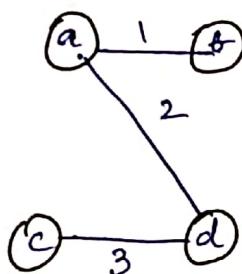
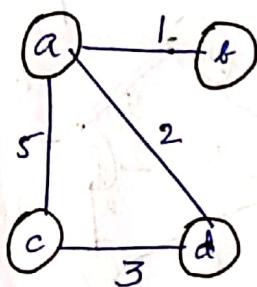
$$20 + 15 + 5 = 40$$

Prims Algorithm:

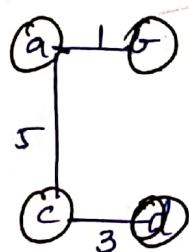
Spanning tree: A spanning tree of a connected graph is its connected acyclic sub-graph that contains all the vertices of the graph.

Minimum Spanning tree:

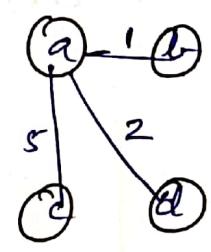
Minimum spanning tree of a weighted connected graph is its spanning tree of the smallest weight, where the weight of a tree is defined as the sum of the weights on all its edges.



$$w(T_1) = 9$$



$$w(T_2) = 8$$



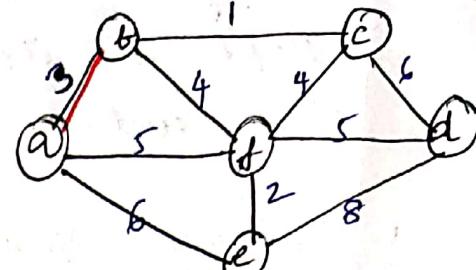
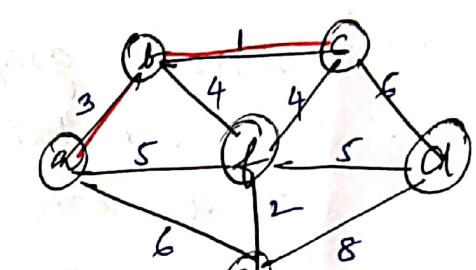
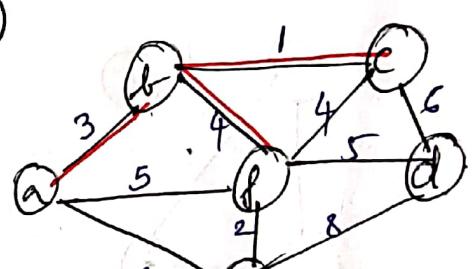
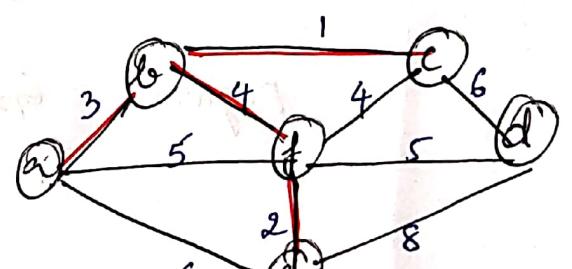
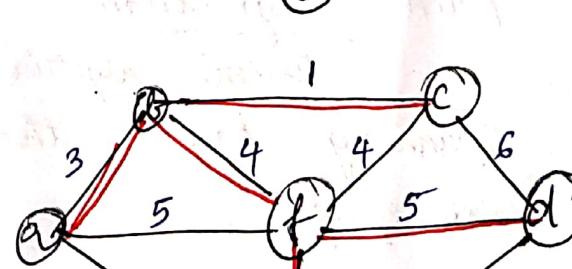
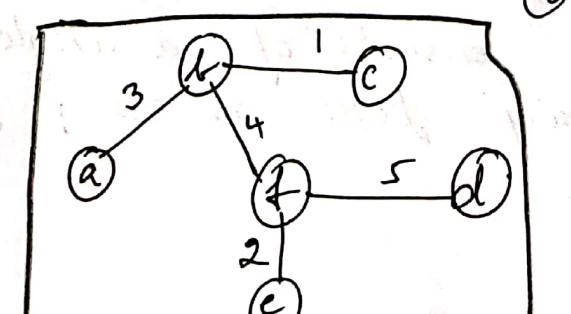
$$w(T_3) = 8$$

Figure: Graph & its spanning trees
 T_1 is the minimum spanning tree.

Prims Algorithm:

Prims algorithm constructs a minimum spanning tree through a sequence of expanding subtrees.

- Initial subtree consists of a single vertex selected arbitrarily from the set V of the graph's vertices.

	Tree vertices	Remaining vertices	Illustration
1.	$a(-, -)$	$b(a, 3) c(-, \infty)$ $d(-, \infty) e(a, 6)$ $f(a, 5)$	
2.	$b(a, 3)$	$c(b, 1) d(-, \infty)$ $e(a, 6) f(b, 4)$	
3.	$c(b, 1)$	$d(c, 6) e(a, 6) f(b, 4)$	
4.	$f(b, 4)$	$d(f, 5) e(f, 2)$	
5.	$e(f, 2)$	$d(f, 5)$	
6.	$d(f, 5)$		

- On each iteration, the tree is expanded in greedy manner by simply attaching to the nearest vertex, that is not already selected.
 - Nearest vertex means vertex which is not already selected; select the edge of smallest weight. Ties can be broken.
 - The algorithm stops after all the graph's vertices have been included in the tree being constructed.
- Since the algorithm expands a tree by exactly one vertex on each of its iterations, total number of such iterations is $n-1$, where n is the number of vertices in the graph.

Algorithm prim(G)

// prim's algorithm for constructing minimum spanning tree
// Input: A weighted connected graph $G = (V, E)$
// Output: E_T , the set of edges composing a minimum spanning tree of G

$$V_T \leftarrow \{v_0\}$$

$$E_T \leftarrow \emptyset$$

for $i \leftarrow 1$ to $|V| - 1$ do

 find a minimum weight edge

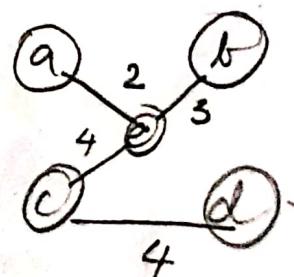
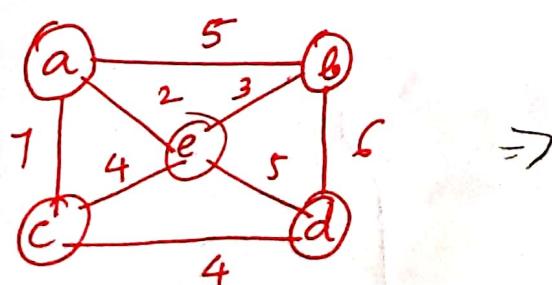
$e^* = (v, u^*)$ among all the edges
(v, u) such that v is in V_T and
 u is in $V - V_T$

$$V_T \leftarrow V_T \cup \{u^*\}$$

$$E_T \leftarrow E_T \cup \{e^*\}$$

return E_T

Find minimum spanning tree using prim's
Algorithm:



How efficient is Prim's Algorithm?

- It depends on data structures chosen for the graph:

(i) If graph is represented by its weight matrix and priority queue is implemented as an unordered array;

- Algorithm running time will be

$$\Theta(N^2)$$

- On each $|V|-1$ iterations, the array implementing priority queue is traversed to find and delete the minimum & then to update.

(ii) If priority queue is implemented as min-heap:

- This results in $O(\log n)$

- Each $|V|-1$ iteration, $\Theta(\log n) + V^2$

(iii) Graph is implemented with adjacency lists & priority queue - min-heap.

- Running time will be $O(|E| \log |V|)$

$$|V|-1 + |E| O \log |V| = \underline{O(|E| \log |V|)}$$

Assi

- ① How can we use prim's algorithm to find a spanning tree of a connected graph with no weight on its edges?

Kruskals Algorithm:

- It is named after Joseph Kruskal.
- This is a greedy algorithm to obtain a minimum spanning tree, that yields (like prims) an optimal solution.
- This algorithm constructs a MST as an expanding sequence of subgraphs, which are always acyclic but are not necessarily connected on intermediate stages of the algorithm.
- Algorithm begins by sorting the edges in increasing order of their weights, then starting with the empty sub-graph. It scans the sorted list and adds the next edge on this list to the current subgraph, if such inclusion does not create a cycle [skip that edge otherwise].

Algorithm: Konskals(G)

// constructs a minimum spanning tree

// Input: weighted connected graph $G = (V, E)$

// Output: E_T , the set of edges composing a minimum spanning tree of G .

Sort E in non-decreasing order of the edge weights $w(e_1) \leq \dots \leq w(e_{|E|})$

$E_T \leftarrow \emptyset$;

encounter $\leftarrow 0$;

$k \leftarrow 0$

while $c\text{counter} < |V| - 1$ do

$k \leftarrow k + 1$

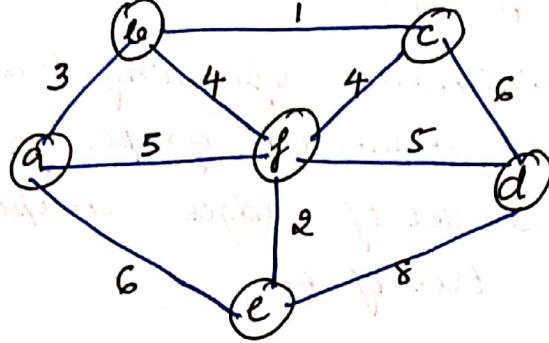
if $E_T \cup \{e_k\}$ is acyclic

$E_T \leftarrow E_T \cup \{e_k\}$;

encounter \leftarrow encounter + 1

return E_T

Ex:

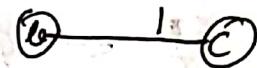
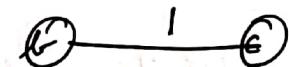
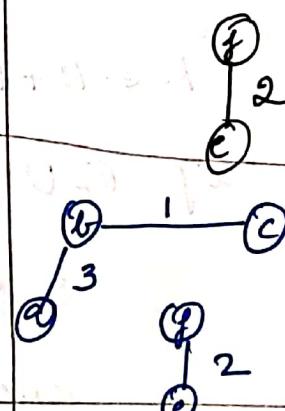
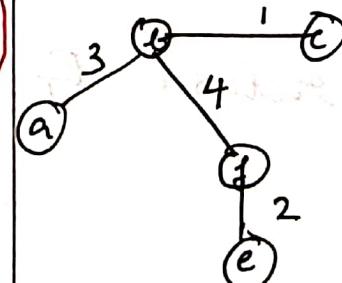
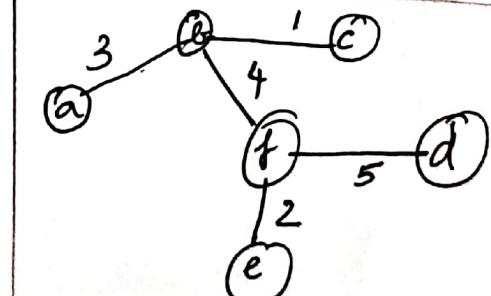


Tree edges

Sorted list of edges

 $\begin{matrix} bc & ef & ab & bf & cf & af & df \\ 1 & 2 & 3 & 4 & 4 & 5 & 5 \end{matrix}$
 $\begin{matrix} ae & cd & de \\ 6 & 6 & 8 \end{matrix}$

Illustration.

bc
1
 $\begin{matrix} bc & ej & ab & bf & cf & af \\ 1 & 2 & 3 & 4 & 4 & 5 \end{matrix}$
 $\begin{matrix} df & ae & cd & de \\ 5 & 6 & 6 & 8 \end{matrix}$
ej
2
 $\begin{matrix} bc & ej & ab & bf & cf & af \\ 1 & 2 & 3 & 4 & 4 & 5 \end{matrix}$
 $\begin{matrix} df & ae & cd & de \\ 5 & 6 & 6 & 8 \end{matrix}$
ab
3
 $\begin{matrix} bc & ej & ab & bf & cf & af \\ 1 & 2 & 3 & 4 & 4 & 5 \end{matrix}$
 $\begin{matrix} df & ae & cd & de \\ 5 & 6 & 6 & 8 \end{matrix}$
bf
4
 $\begin{matrix} bc & ej & ab & bf & cd & af \\ 1 & 2 & 3 & 4 & 4 & 5 \end{matrix}$
 $\begin{matrix} df & ae & cd & de \\ 5 & 6 & 6 & 8 \end{matrix}$
df
5

Disjoint subsets and union-Find Algorithms:

- makeset (x) - creates a one-element set $\{x\}$.
It is assumed that this operation can be applied to each of the elements s of set S only once.

Ex: $S = \{1, 2, 3, 4, 5, 6\}$

makeset (i) creates the set $\{i\}$ and applying this operation six times initializes the structure to the collection of six singleton sets: $\{\{1\}\}, \{\{2\}\}, \{\{3\}\}, \{\{4\}\}, \{\{5\}\}, \{\{6\}\}$.

- find (x)
returns S_x , a subset containing x .
- union (x, y)
constructs the union of the disjoint subsets S_x and S_y containing x and y , respectively and adds it to the collection to replace S_x and S_y which are deleted from it.

Ex: performing $\text{union}(1, 4)$ and $\text{union}(5, 2)$ yields

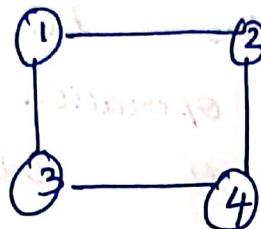
$$\{\{1, 4\}, \{2\}, \{3, 6\}\}$$

and $\text{union}(4, 5)$ and by $\text{union}(3, 6)$ is

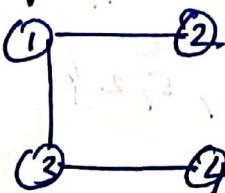
$$\{\{1, 4, 5, 2\}, \{3, 6\}\}$$

- union is used to check whether there exists a cycle or not.
- union $\{1, 2\}$ is not possible as it exists in the same set $\{1, 4, 5, 2\}$ and forms a cycle.

Ex: Check the graph



- makeset(i) - It creates four singleton sets $\{1\}, \{2\}, \{3\}, \{4\}$
- find i - locates the elements
- union $\{1, 2\}$ - it is not possible as they are from different sets: $\{1, 2\}, \{3\}, \{4\}$
- union $\{3, 4\}$ - $\{1, 2\}, \{3, 4\}$
- union $\{1, 3\}$ - $\{1, 2, 3, 4\}$
- union $\{2, 4\}$ - Not possible as they are from same sets and hence forms a cycle.

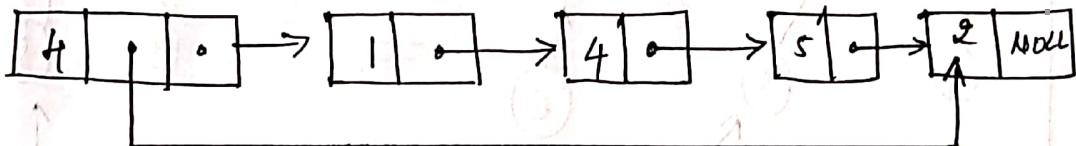


Implementation of find and union operations:

(1) Quick find:

- It uses array ordered by elements.
- Array's values indicate the representatives of the subsets
- Each subset is implemented as linked list whose header contains the pointers to the first and last element along with number of elements in the list.

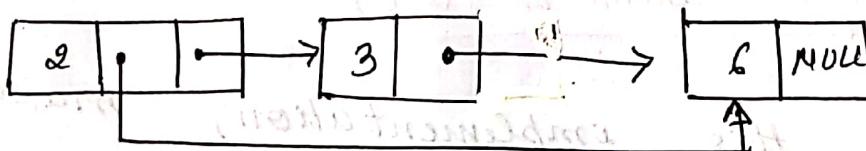
list 1



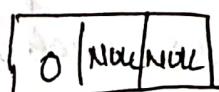
list 2



list 3



list 4



- Executing $\text{union}(x, y)$ takes longer.
- Time efficiency to find is $\Theta(n^2)$
- This can be overcome by using union by size combining shorter list and perform find operation.

- so the time efficiency for union by size is $O(n \log n + m)$

for sequence of $(n-1)$ unions and m finds.

(ii) Quick union:

second principal alternative for implementing disjoint subsets - are represent each subset by a rooted tree.

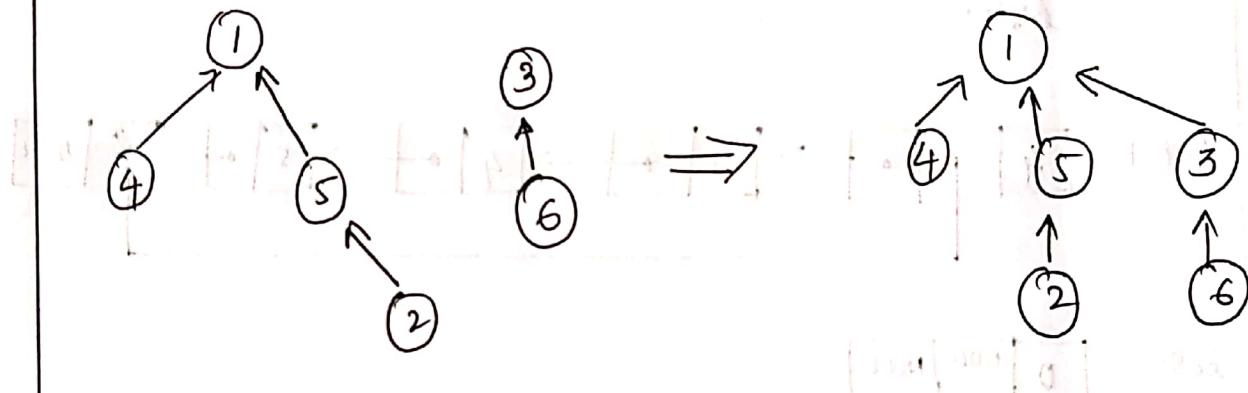
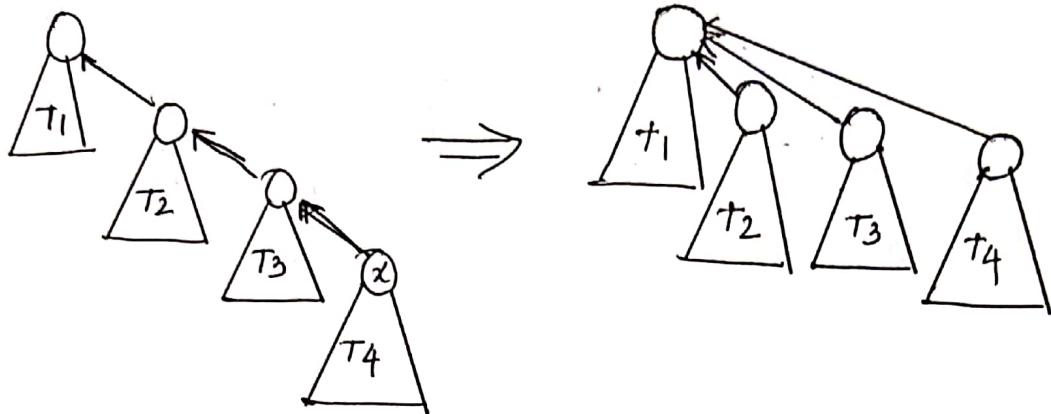


fig \Rightarrow union of $(\{5, 6\})$

- for this implementation, `makeSet(x)` requires the creation of a single-node tree, which is $\Theta(1)$ operation for each, so n singletone is $\Theta(n)$.

for union operation, root element must be checked, if they are same, it forms cycle, if not it can be performed.

Path compression:



- To improve time efficiency, path compression is used, as in union operation, each time it should refer root element. to check for the cycle.

To reduce the access time for root from node descendants, all the nodes point to the tree's root as shown in above figure.

Dijkstra's Algorithm:

The single source shortest path problem states:

"Given a weighted, connected graph and a source vertex, the task is to find the shortest path from source to all other vertices in the given graph

$$G = (V, E)$$

- Dijkstra's Algorithm solves SSSP (single source shortest path problem).
- First the algorithm finds the shortest path from the source to a vertex nearest to it, then to second nearest and so on.
- In general before its i^{th} iteration, the algorithm has commences, already identified the shortest paths to $i-1$ other vertices nearest to the source

Algorithm Dijkstra (n , adj , src , dest)

// computes shortest paths from source vertex

// Input : number of vertices n ,

adjacency matrix adj & source vertex

// output : The shortest paths from source vertex to all other vertices.

{

for $i=0$ to $n-1$ do

{

$\text{dist}[i] = \text{adj}[\text{src}][i]$;

$\text{visited}[i] = 0$

}

$\text{visited}[\text{src}] = 1$

for $i=1$ to $n-1$ do

{

find unvisited and $\text{dist}[\text{unvisited}]$ such

$\text{dist}[\text{unvisited}]$ is minimum

and vertex $\text{unvisited} \in V - \text{visited}[]$

for every vertex $\in V - \text{visited}[]$ do

{

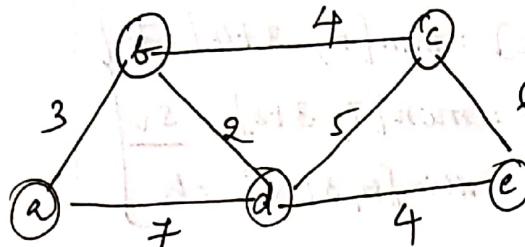
if $\text{dist}[\text{unvisited}] + \text{adj}[\text{unvisited}][\text{vertex}] < \text{dist}[\text{vertex}]$ then

{

$\text{dist}[\text{vertex}] = \text{dist}[\text{unvisited}] + \text{adj}[\text{unvisited}][\text{vertex}]$

} }

problem: consider the following graph & obtain the shortest path from the vertex "a" to all other vertices using Dijkstra's Algorithm.



	a	b	c	d	e
a	0	3	∞	7	∞
b	3	0	4	2	∞
c	∞	4	0	5	6
d	7	2	5	0	4
e	∞	∞	6	4	0

- Initialize distance array i.e $dist[]$ from source "a".

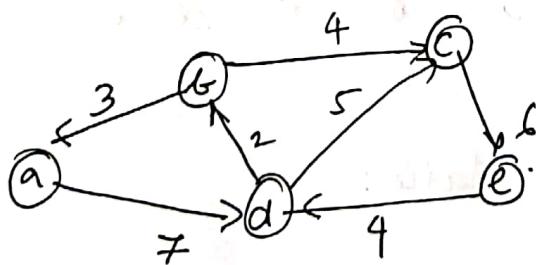
$$dist[] = [0 \ 3 \ \infty \ 7 \ \infty]$$

$$\quad \quad \quad a \ b \ c \ d \ e$$

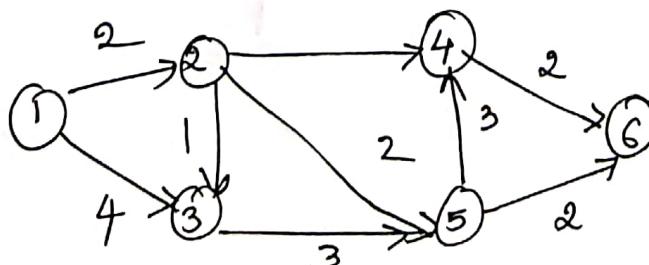
- Build the table:

visited	vertex = v-visited	$\text{dist[vertex]} = \min[\text{dist[vertex}], \text{dist[unvisited]} + \text{adj[unvisited]}$	Find unvisited & dist(unvisited) such that dist[unvisited] is minimum & unv's $\notin V\text{-visited}$.
a	b, c, de	-	b, 3
a, b	c, d, e	$\text{dist}[c] = \min[6, 3+4] = 7$ $\text{dist}[d] = \min[7, 3+2] = 5$ $\text{dist}[e] = \min[6, 3+\infty] = 6$	d, 5
a, b, d	c, e	$\text{dist}[c] = \min[7, 5+5] = 7$ $\text{dist}[e] = \min[6, 5+4] = 9$	c, 7
a, b, d, c	e	$\text{dist}[e] = \min[9, 7+6] = 9$	e, 9
a, b, d, c	-	-	-

solve:



a as source



5 as source

Huffman Trees:

There are mainly two types of compressing data namely

- fixed length - each character represents same number of bits.
- variable length ex: a=00, b=01, c=11

↳ characters having varying number of bits

ex: a=0, b=01, c=111

- Huffman coding [David Huffman] is a popular method for compressing data with variable length codes. The idea behind this approach is:

- (i) It is prefix free
 - (ii) Least occurrences characters have more bit representation compared to more occurrences characters.
- Given a set of data symbols (an alphabets) & their frequencies of occurrence. The huffman coding algorithm constructs a set of variable length codewords with the shortest average length & assigns them to symbols.

- Huffman's Algorithm:-

Step 1 - Initialize n one-node trees and label them with characters of the alphabet. Record the frequency of each character in its tree's root to indicate tree's weight.

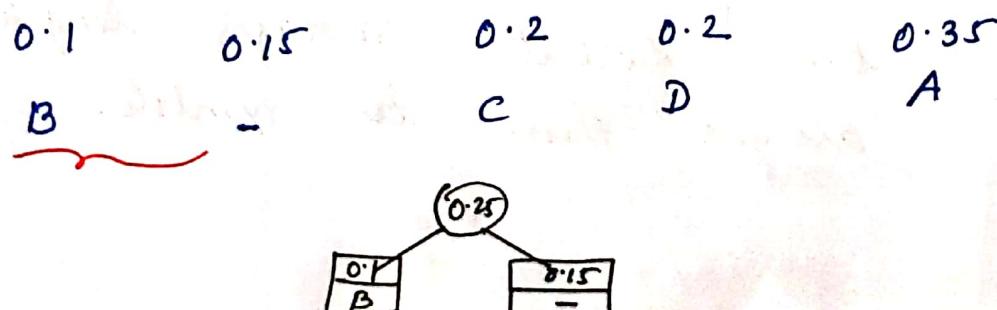
Step 2 - Repeat the following operation until single tree is obtained.

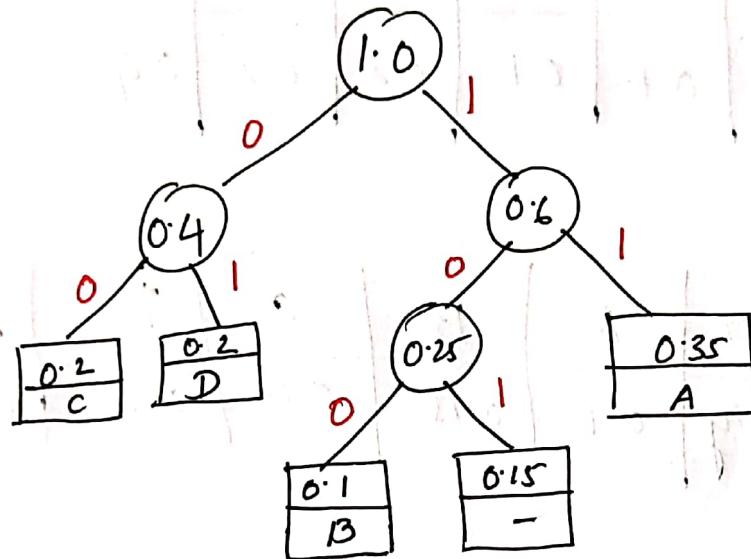
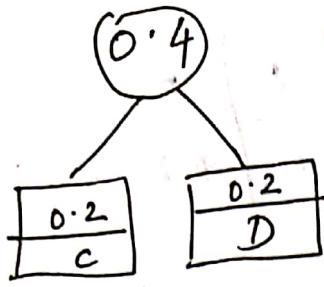
Find two trees with the small weight. Make them left and right subtree of new tree and record the sum of their weight in the root of the new tree as its weight.

P construct Huffman tree for the following data:

character	A	B	C	D	-
probability	0.35	0.1	0.2	0.2	0.15

step 1 \rightarrow Root based on probability.





$C \rightarrow 00 - 2 \text{ bits}$

$D \rightarrow 01 - 2 \text{ bits}$

$A \rightarrow 11 - 2 \text{ bits}$

$B \rightarrow 100 - 3$

$- \rightarrow 101 - 3$

with the occurrence probabilities ; the expected number of bits are:

$$(A * 0.35) + (B * 0.1) + (C * 0.2) +$$

$$(D * 0.2) + (- * 0.15)$$

$$= 2 * 0.35 + 3 * 0.1 + 2 * 0.2 + 2 * 0.2 \\ + 3 * 0.15 = \underline{\underline{2.25}}$$

construct Huffman tree:

a)

character	A	B	C	D	E	-
probability	0.5	0.35	0.5	1	0.4	0.2

b)

character	A	B	C	D	E
probability	0.1	0.1	0.2	0.2	0.4

c)

character	A	B	C	D	-
probability	0.1	0.15	0.15	0.2	0.4

$$2^{k-1} = 3 \Rightarrow k = 3$$

$$2^{k-1} = 3 \Rightarrow k = 3$$

$$2^k = 11 \Rightarrow k = 4$$

$$E = 331 \Rightarrow 3$$

$$E = 131 \Rightarrow -$$

Step 1: Selecting characters according to their probability

+ (0.5 + 0.35) = 0.85 and A

(0.5 + 0.35) = 0.85

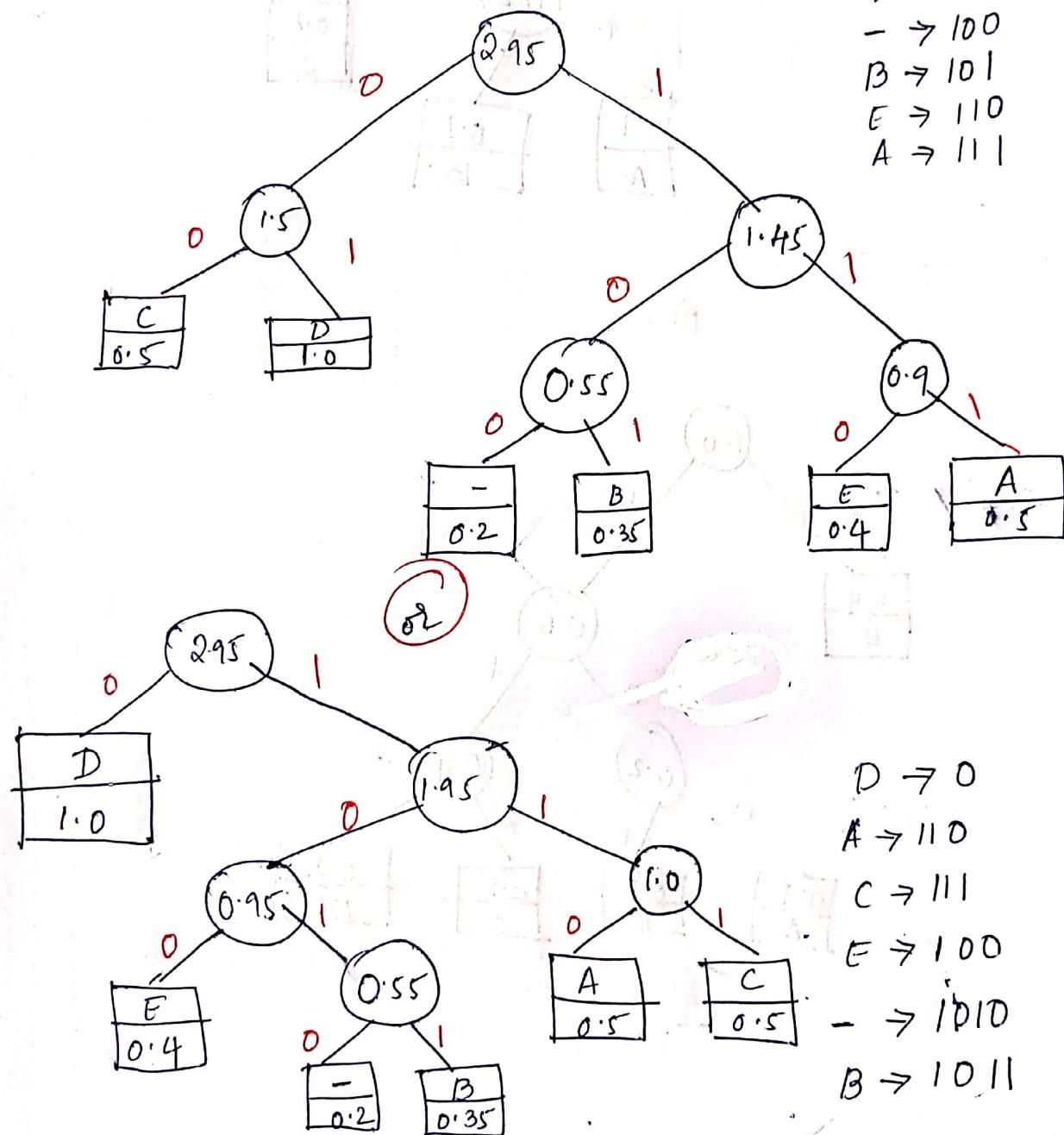
0.85 + 0.85 = 1.70

801

(a) sort

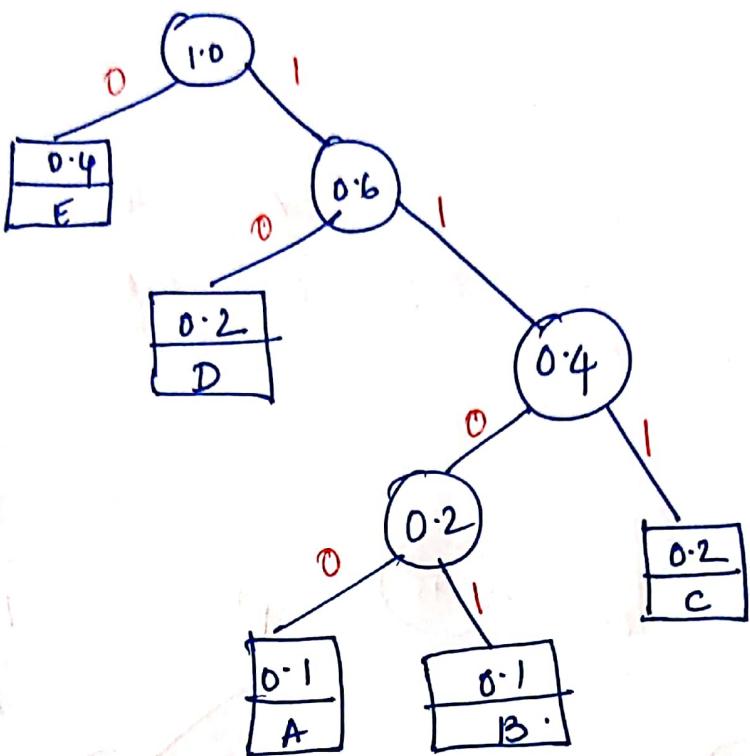
B E A D
 $0.2 \quad 0.35 \quad 0.4 \quad 0.5 \quad 0.65 \quad 1.0$

C \rightarrow 00
 D \rightarrow 01
 - \rightarrow 100
 B \rightarrow 101
 E \rightarrow 110
 A \rightarrow 111

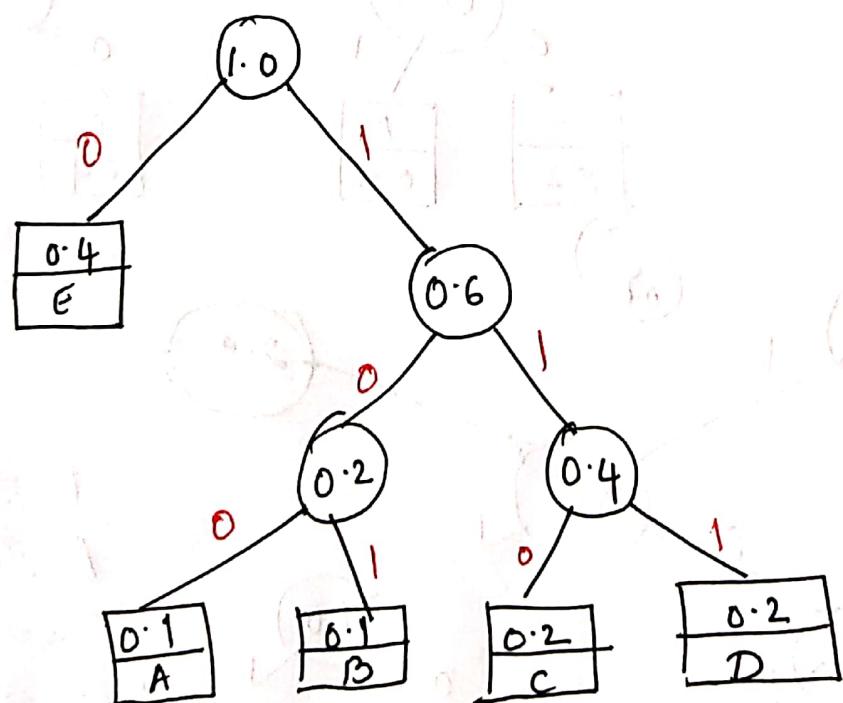


D \rightarrow 0
 A \rightarrow 110
 C \rightarrow 111
 E \rightarrow 100
 - \rightarrow 1010
 B \rightarrow 1011

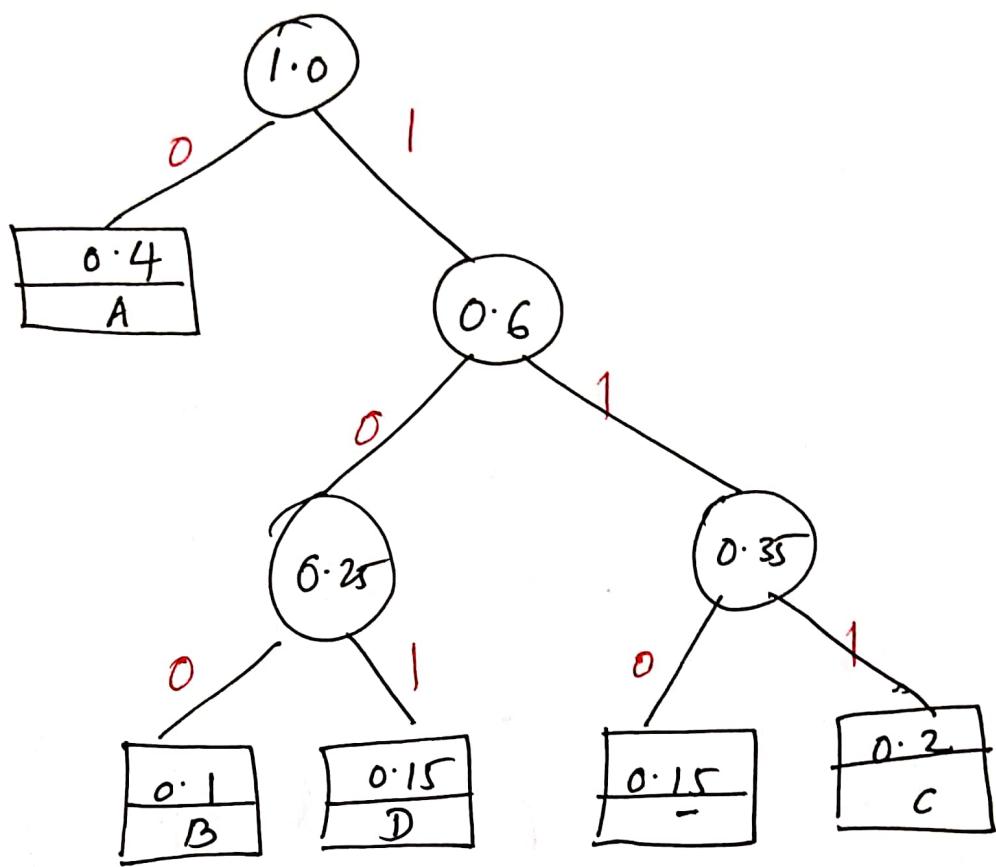
(1)



or



(c)

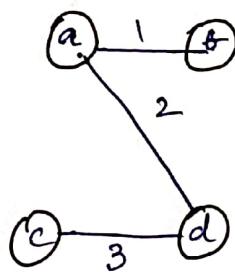
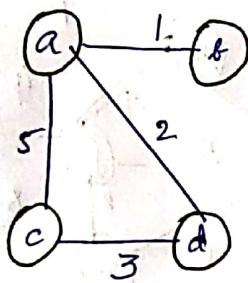


Prim's Algorithm:

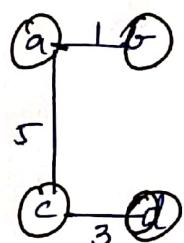
Spanning tree: A spanning tree of a connected graph is its connected acyclic sub-graph that contains all the vertices of the graph.

Minimum spanning tree:

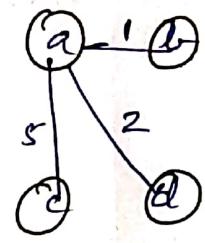
Minimum spanning tree of a weighted connected graph is its spanning tree of the smallest weight, where the weight of a tree is defined as the sum of the weights on all its edges.



$$w(T_1) = 6$$



$$w(T_2) = 9$$



$$w(T_3) = 8$$

Figure: Graph G & its spanning trees
 T_1 is the minimum spanning tree.

Prim's Algorithm:

Prim's algorithm constructs a minimum spanning tree through a sequence of expanding subtrees.

- Initial subtree consists of a single vertex selected arbitrarily from the set V of the graph's vertices.

	Tree vertices	Remaining vertices	Illustration
1.	$a(-, -)$	$b(a, 3) \quad c(-, \infty)$ $d(-, \infty) \quad e(a, 6)$ $f(a, 5)$	
2.	$b(a, 3)$	$c(b, 1) \quad d(-, \infty)$ $e(a, 6) \quad f(b, 4)$	
3.	$c(b, 1)$	$d(c, 6) \quad e(a, 6) \quad f(b, 4)$	
4.	$f(b, 4)$	$d(f, 5) \quad e(f, 2)$	
5.	$e(f, 2)$	$d(f, 5)$	
6.	$d(f, 5)$		

Algorithm prim(G)

// Prim's algorithm for constructing minimum spanning tree
// Input: A weighted connected graph $G = (V, E)$
// Output: E_T , the set of edges composing a minimum spanning tree of G

$$V_T \leftarrow \{v_0\}$$

$$E_T \leftarrow \emptyset$$

for $i \leftarrow 1$ to $|V| - 1$ do

 find a minimum weight edge

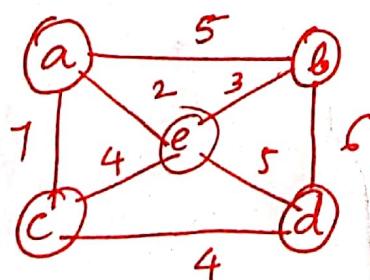
$e^* = (v, u^*)$ among all the edges (u, v) such that v is in V_T and u is in $V - V_T$

$$V_T \leftarrow V_T \cup \{u^*\}$$

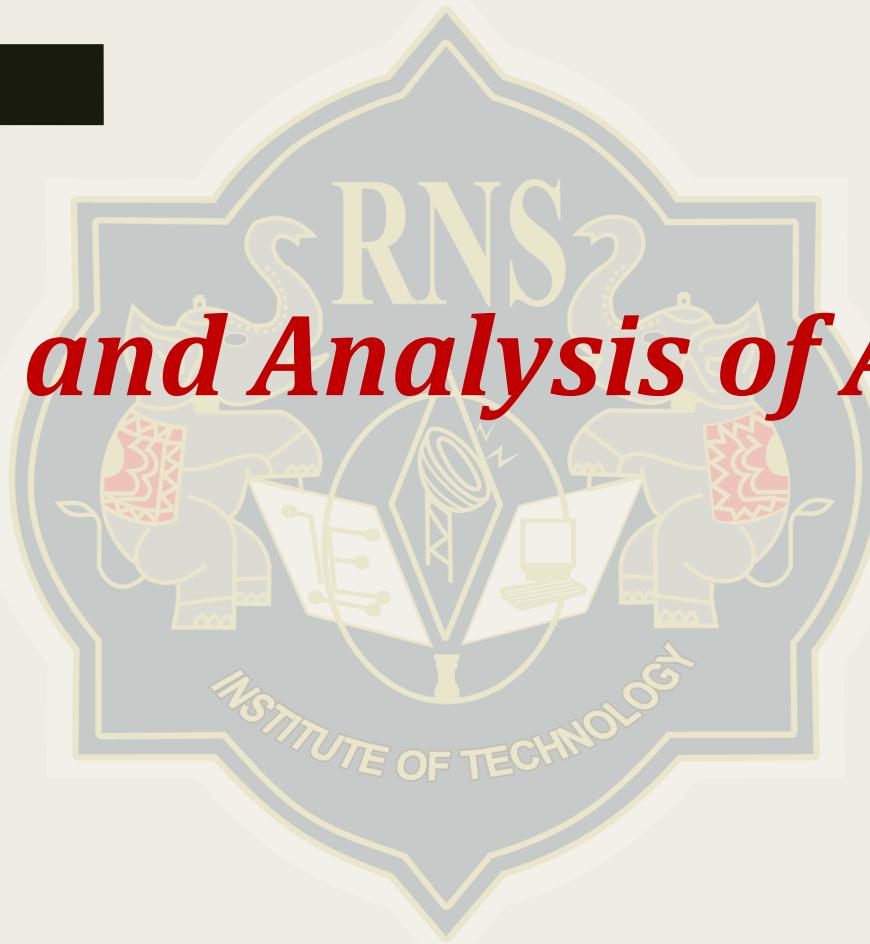
$$E_T \leftarrow E_T \cup \{e^*\}$$

return E_T

Find minimum spanning tree using prim's algorithm:



Design and Analysis of Algorithms



ESTD:2001

An Institute with a Difference

Greedy Method

- It's a straight forward design technique that can be applied to a wide variety of problems
- Coin change problem
 - Let $A_n = \{a_1, a_2, a_3, \dots, a_n\}$ be a finite set of distinct coin types (for example, $a_1 = 5p$, $a_2 = 10p$, $a_3 = 20p$, $a_4 = 25p$, $a_5 = 50p$ and so on.)
 - Assume each a_i is an integer and $a_1 > a_2 > a_3 \dots a_n$.
 - Each type is available in unlimited quantity.
 - The coin-changing problem is to make up an exact amount C using a minimum total number of coins.
 - C is an integer >0 .



Greedy Method

■ Lets take an example

- X goes to a shop to buy an item worth 30p and hands over 1Rs.
- What is the change expected by X? 70p
- Probable set of coins given by shop keeper
 - A={10, 20, 20, 20}
 - B={10, 10, 10, 10 ,10 ,10, 10}
 - C={5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5}
 - D={25 ,25 ,20}
 - E={50, 20}
 - F= {50, 10, 10}

■ What was the constraint?

- The sum of the coins given by the shopkeeper =70p

■ The subset that satisfies this constraint is called **feasible solution**

■ The feasible solution that maximizes/minimizes the objective function is called an **optimal solution**

Greedy Method

■ Greedy algorithms work in stages

- At each stage, a decision is made regarding whether a particular input is in an optimal solution.
- This is done by considering the inputs in an order determined by some selection procedure.
- If the inclusion of the next input into the partially constructed optimal
- Solution will result in an infeasible solution, then this input is not added to the partial solution, otherwise, it is added

Greedy Method

Control Abstraction

```
Algorithm Greedy(a, n)
// a[1:n] contains the n inputs.
{
    solution:=0;// Initialize the solution.
    for i :=1 to n do
    {
        x :=Select(a);
        if Feasible(solution, x) then
            solution:=Union (solution, x);
    }
    return solution;
}
```

Greedy Method

Knapsack problem

- Given n objects, with each object i having a positive weight w_i and the profit p_i
- A knapsack / bag with a capacity m
- If a fraction x_i , $0 < x_i < 1$, of object i is placed into the knapsack, then a profit of $p_i x_i$ is earned
- The objective is to find the set of objects which fills the knapsack and maximizes the total profit
- The total weight of the selected objects can be at most m

Greedy Method

Formal definition of Knapsack problem

$$\text{maximize} \quad \sum_{1 \leq i \leq n} p_i x_i$$

$$\text{subject to} \quad \sum_{1 \leq i \leq n} w_i x_i \leq m$$

$$\text{and } 0 \leq x_i \leq 1, \quad 1 \leq i \leq n$$

$x_i = 1$ when item i is selected and let $x_i = 0$ when item i is not selected.

Greedy Method

Example

Consider the following instance of the knapsack problem

$n = 3$, $m = 20$, $(p_1, p_2, p_3) = (25, 24, 15)$ and $(w_1, w_2, w_3) = (18, 15, 10)$.

Find all the feasible solutions and hence an optimal solution

First feasible solution

- Consider the objects according to their profit (descending order)
- Object with largest profit is considered first and so on
- Object 1 (highest profit) with weight 18 can be put into the knapsack in its entirety . So $x_1=1$ and $\text{RemCap} = 2$
- Object 2 (second highest profit) with weight 15 is considered but cant be put into the bag in its entirety ($\text{RemCap} < w_2$)
- Hence a fraction $2/15$ is put into the knapsack, $x_2=2/15$, $\text{RemCap} = 0$

Greedy Method

-Remaining objects cant be put into the knapsack

$$\sum_{i=1 \text{ to } 3} w_i x_i = 18 \times 1 + 15 \times \frac{2}{15} + 0 = 20$$

$$\sum_{i=1 \text{ to } 3} p_i x_i = 25 \times 1 + 24 \times \frac{2}{15} + 0 = 28.2$$

The solution vector $(x_1, x_2, x_3) = (1, \frac{2}{15}, 0)$

Greedy Method

Second feasible solution

- Consider the objects according to their weights (increasing order)
- Object with smallest weight is considered first and so on
- Object 3 (smallest weight) with weight 10 can be put into the knapsack in its entirety .
So $x_3=1$ and $\text{RemCap} = 10$
- Object 2 (second smallest weight) with weight 15 is considered but cant be put into the bag in its entirety ($\text{RemCap} < w_2$)
- Hence a fraction $10/15$, i.e. $2/3$ is put into the knapsack, $x_2=2/3$, $\text{RemCap} = 0$
- Remaining objects cant be put into the knapsack

$$\sum_{i=1 \text{ to } 3} w_i x_i = 0 + 15 \times \frac{2}{3} + 10 \times 1 = 20$$

$$\sum_{i=1 \text{ to } 3} p_i x_i = 0 + 24 \times \frac{2}{3} + 15 \times 1 = 31$$

The solution vector $(x_1, x_2, x_3) = (0, \frac{2}{3}, 1)$

Greedy Method

Third feasible solution

- Consider the objects according to the ratio p_i/w_i (decreasing order)
- $p_1/w_1 = 1.38 \quad p_2/w_2 = 1.6 \quad p_3/w_3 = 1.5$
- Object with highest ratio is considered first and so on
- Object 2 (highest ratio) with weight 15 can be put into the knapsack in its entirety . So $x_2=1$ and $\text{RemCap} = 5$
- Object 3 (second highest ratio) with weight 10 is considered but cant be put into the bag in its entirety ($\text{RemCap} < w_3$)
- Hence a fraction 5/10, i.e. 1/2 is put into the knapsack, $x_3=1/2$, $\text{RemCap} = 0$
- Remaining objects cant be put into the knapsack

$$\sum_{i=1 \text{ to } 3} w_i x_i = 0 + 15 \times 1 + 10 \times \frac{1}{2} = 20$$

$$\sum_{i=1 \text{ to } 3} p_i x_i = 0 + 24 \times 1 + 15 \times \frac{1}{2} = 31.5$$

The solution vector $(x_1, x_2, x_3) = (0, 1, \frac{1}{2})$

Greedy Method

- First feasible solution =28.2
- Second feasible solution =31
- Third feasible solution =31.5
- Since the objective function is to maximize the profit,
the optimal solution is
 - Profit =31.5
 - The solution vector $(x_1, x_2, x_3) = (0, 1, \frac{1}{2})$

Greedy Method

Homework

Consider the following instance of the knapsack problems.

- Find all the feasible solutions and hence an optimal solution
 $n = 3, m= 20, (p_1, p_2, p_3) = (30, 21, 18)$ and $(w_1, w_2, w_3) = (18, 15, 10)$.
- Find all the feasible solutions and hence an optimal solution
 $N=7, m=15, p_i=(10,5,15,7,6,18,3)$ and $w_i=(2,3,5,7,1,4,1)$

Greedy Method

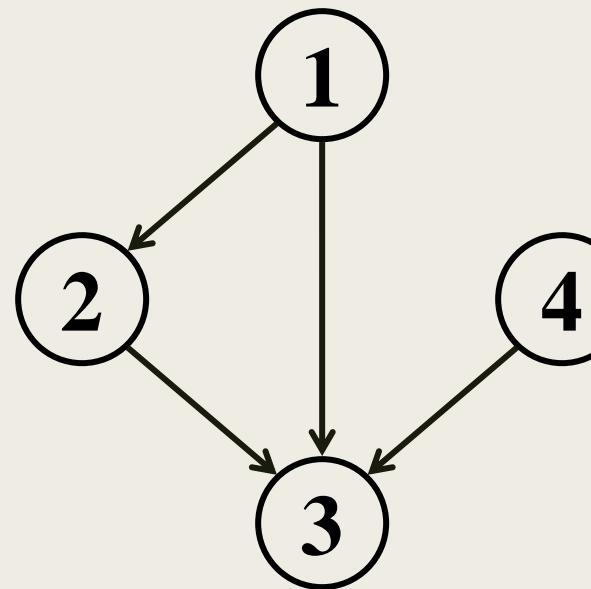
ALGORITHM Greedy_Knapsack(m,n)

```
{  
for( i = 1 to n ) do  x[ i ] = 0.0  
u=m  
for( i = 1 to n ) do  
{  
    if( w[ i ]>u ) then break  
    x[ i ] = 1.0  
    u=u-w[ i ]  
}  
if( i <= n ) then  
x[ i ] = u/w[ i ]  
}
```

Introduction to Graphs

■ A graph $G = (V, E)$

- $V = \text{set of vertices}$
- $E = \text{set of edges} = \text{subset of } V \times V$
- $\text{Thus } |E| = O(|V|^2)$



Vertices: {1, 2, 3, 4}

Edges: {(1, 2), (2, 3), (1, 3), (4, 3)}

Graph – Variations

■ Directed / undirected:

-In an undirected graph:

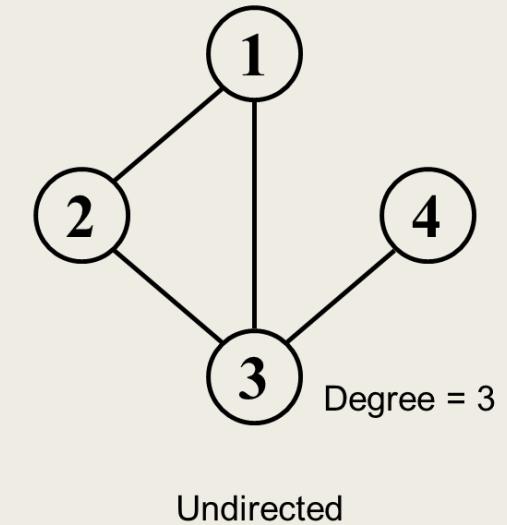
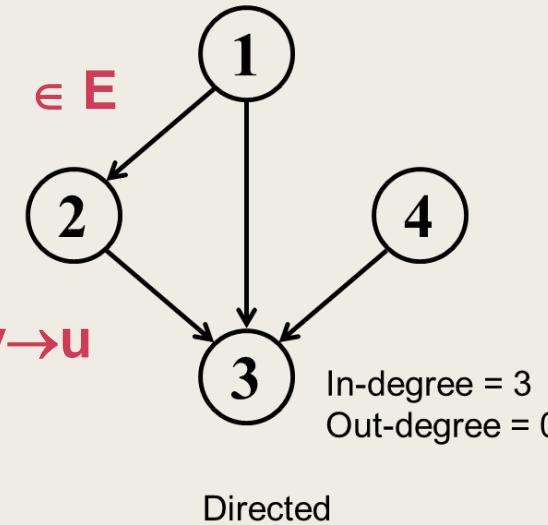
- Edge $(u,v) \in E$ implies edge $(v,u) \in E$
- Road networks between cities

-In a directed graph:

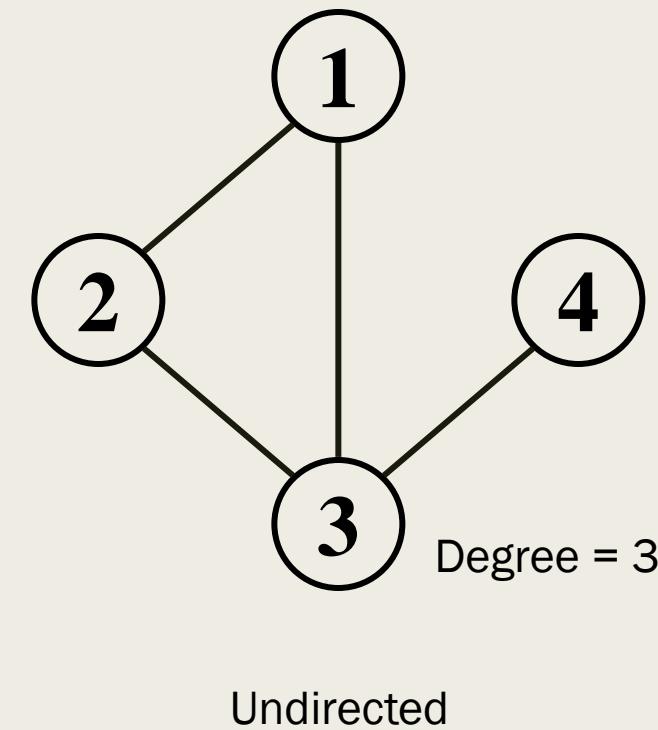
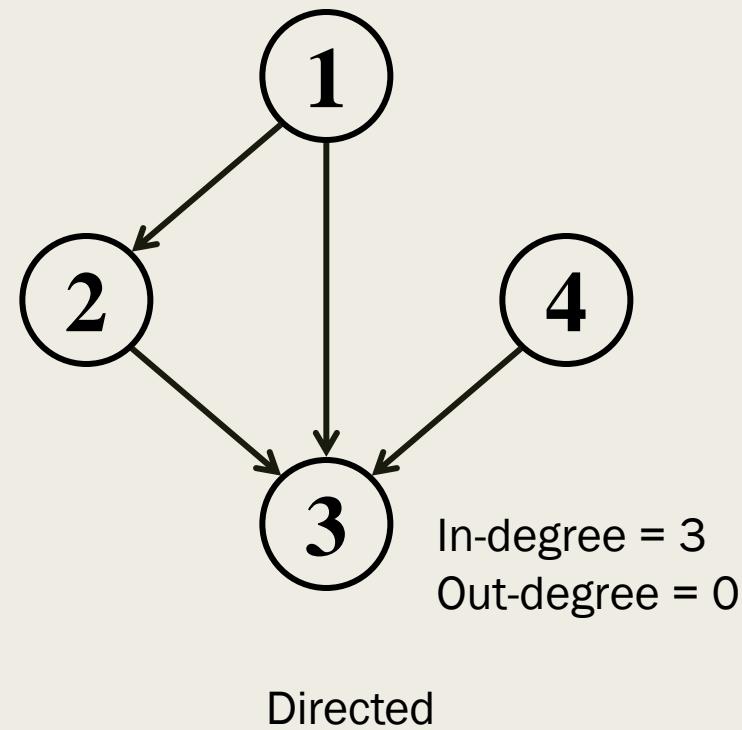
- Edge (u,v) : $u \rightarrow v$ does not imply $v \rightarrow u$
- Street networks in downtown

-Degree of vertex v:

- The number of edges adjacency to v
- For directed graph, there are in-degree and out-degree



Graph – Variations

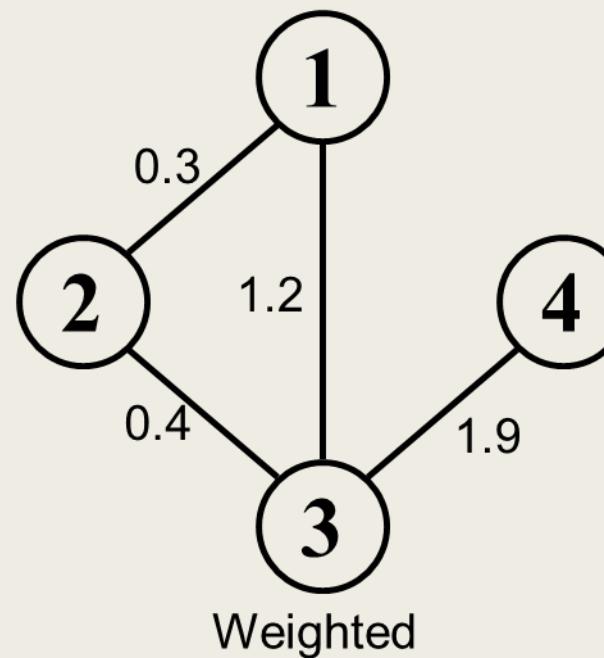
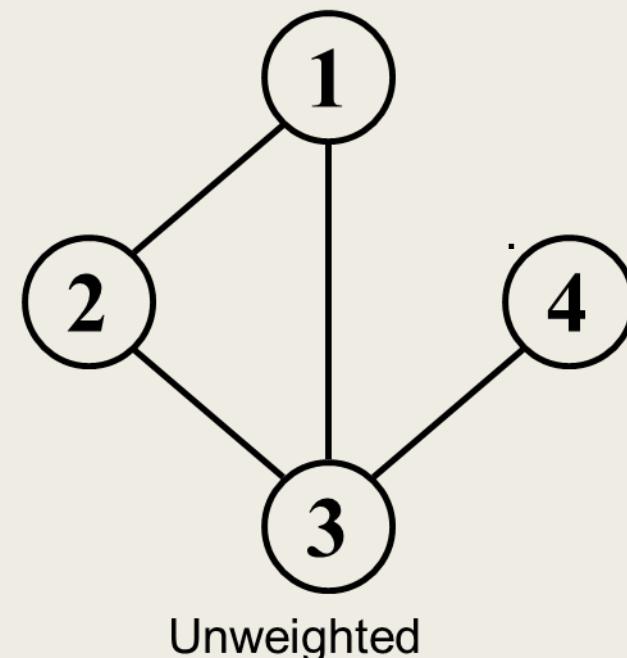


Graph – Variations

■ Weighted / unweighted:

- In a weighted graph, each edge or vertex has an associated weight (numerical value)

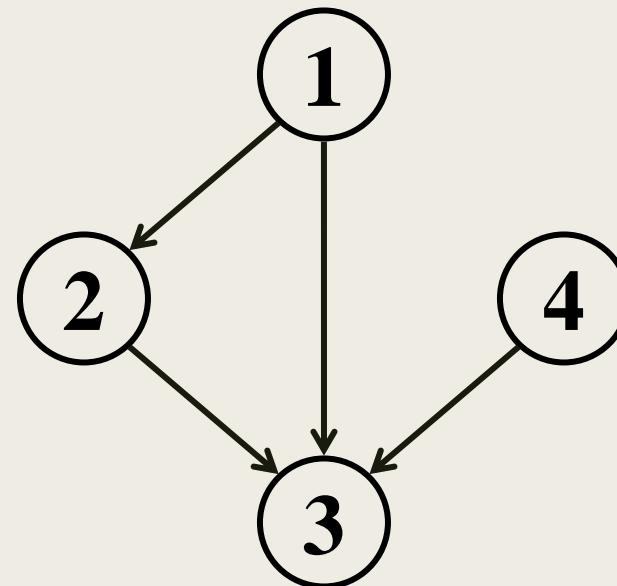
- E.g., a road map: edges might be weighted w/ distance



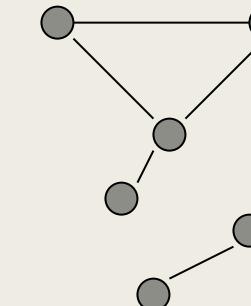
Graph – Variations

■ Connected / disconnected:

- A **connected graph** has a path from every vertex to every other
- A **directed graph is strongly connected if there is a directed path between any two vertices**



Connected but not
strongly connected



Graph – Variations

■ Dense / sparse:

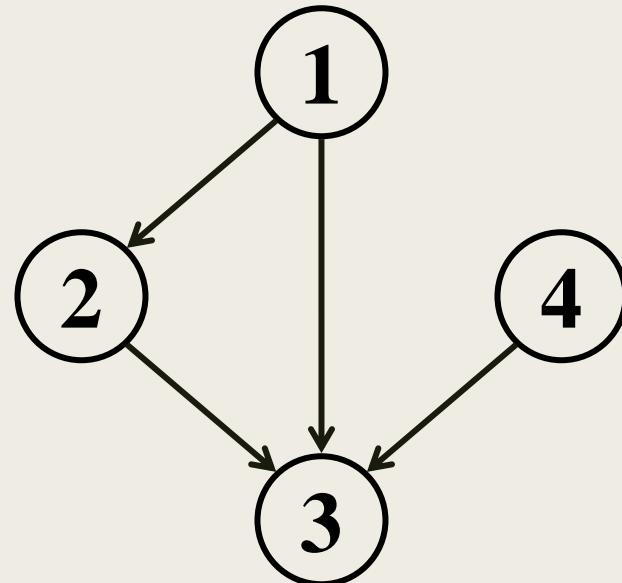
- *Graphs are sparse when the number of edges is linear to the number of vertices*
 - $|E| \in O(|V|)$
- *Graphs are dense when the number of edges is quadratic to the number of vertices*
 - $|E| \in O(|V|^2)$
- *Most graphs of interest are sparse*
- *If you know you are dealing with dense or sparse graphs, different data structures may make sense*

Representing Graphs

- Assume $V = \{1, 2, \dots, n\}$
- An adjacency matrix represents the graph as a $n \times n$ matrix A :
 - $A[i, j] = 1$ if edge $(i, j) \in E$
 - $A[i, j] = 0$ if edge $(i, j) \notin E$
- For weighted graph
 - $A[i, j] = w_{ij}$ if edge $(i, j) \in E$
 - $A[i, j] = 0$ if edge $(i, j) \notin E$
- For undirected graph
 - **Matrix is symmetric:** $A[i, j] = A[j, i]$

Graphs: Adjacency Matrix

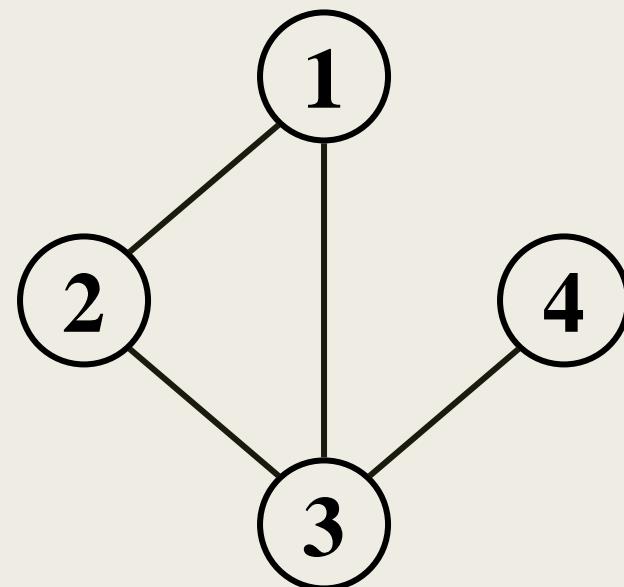
■ Example:



A	1	2	3	4
1				
2				
3				??
4				

Graphs: Adjacency Matrix

■ Example

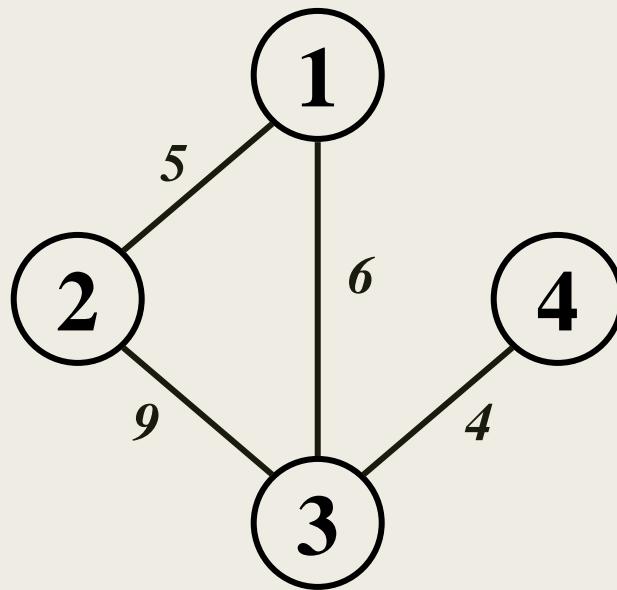


Undirected graph

A	1	2	3	4
1	0	1	1	999
2	1	0	1	0
3	1	1	0	1
4	999	0	1	0

Graphs: Adjacency Matrix

■ Example

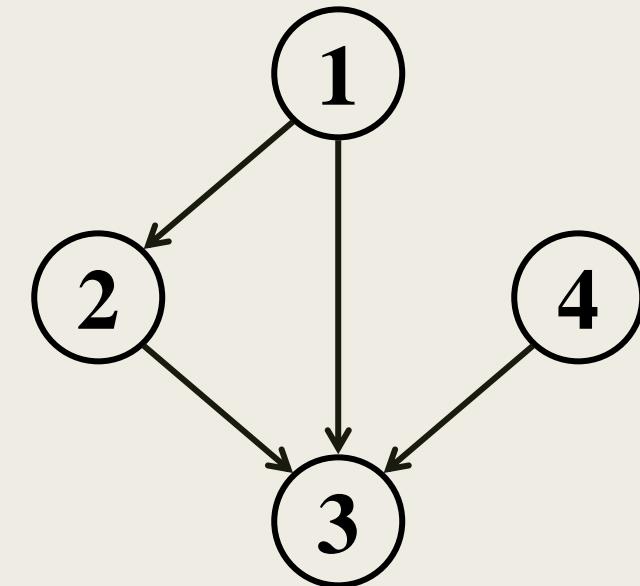


Weighted graph

A	1	2	3	4
1	0	5	6	999
2	5	0	9	0
3	6	9	0	4
4	99	0	4	0
	9			

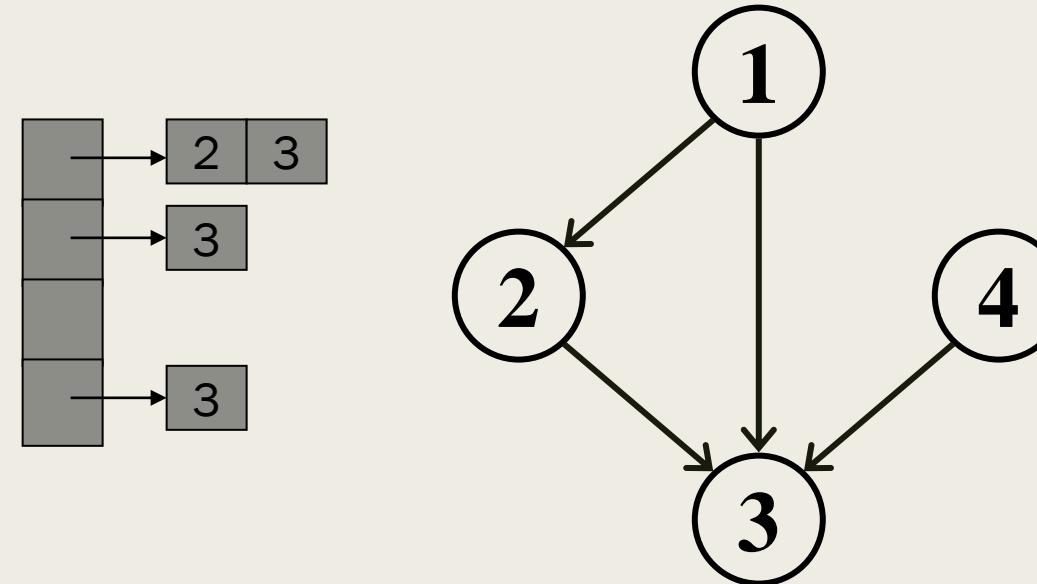
Graphs: Adjacency List

- **Adjacency list:** for each vertex $v \in V$, store a list of vertices adjacent to v
- **Example:**
 - $\text{Adj}[1] = \{2,3\}$
 - $\text{Adj}[2] = \{3\}$
 - $\text{Adj}[3] = \{\}$
 - $\text{Adj}[4] = \{3\}$
- **Variation:** can also keep a list of edges coming into vertex



Graphs: Adjacency List

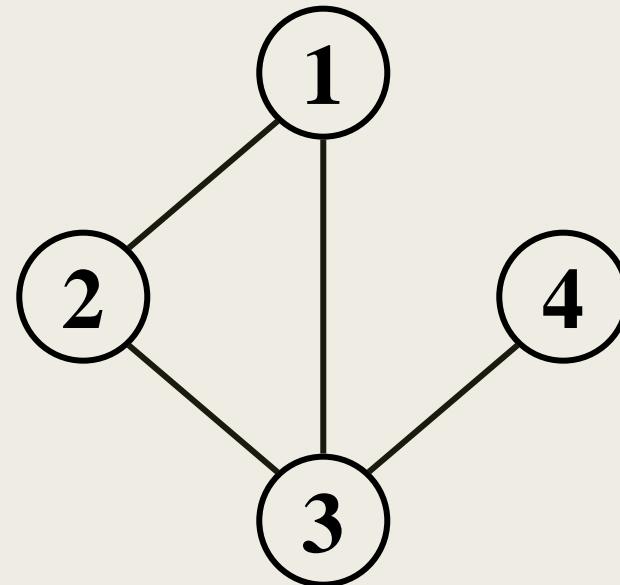
■ Adjacency list



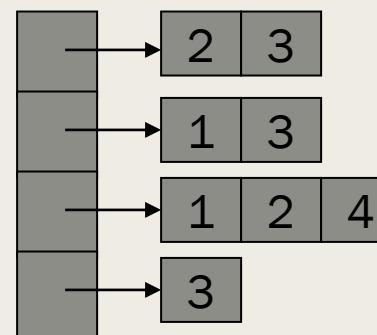
How much storage does the adjacency list require?
A: $O(V+E)$

Graphs: Adjacency List

■ Undirected graph

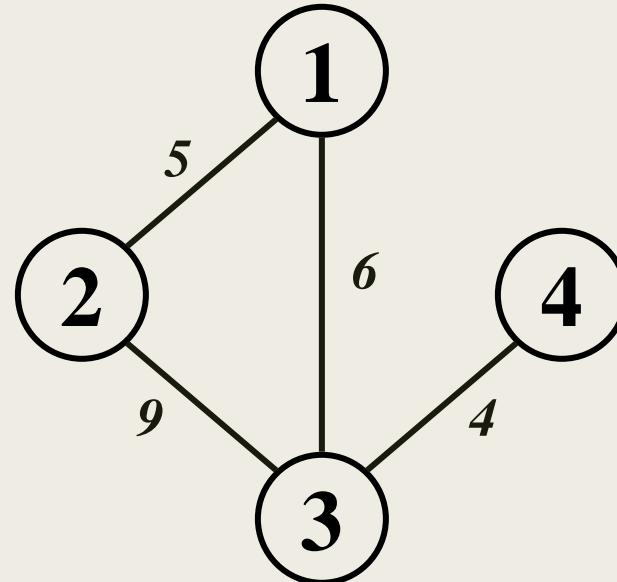


A	1	2	3	4
1	0	1	1	0
2	1	0	1	0
3	1	1	0	1
4	0	0	1	0

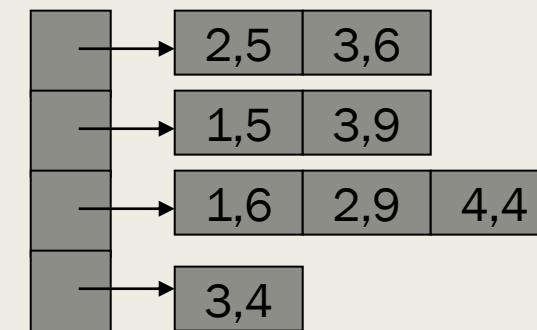


Graphs: Adjacency List

- Weighted graph



A	1	2	3	4
1	0	5	6	0
2	5	0	9	0
3	6	9	0	4
4	0	0	4	0



Trade off between two representations

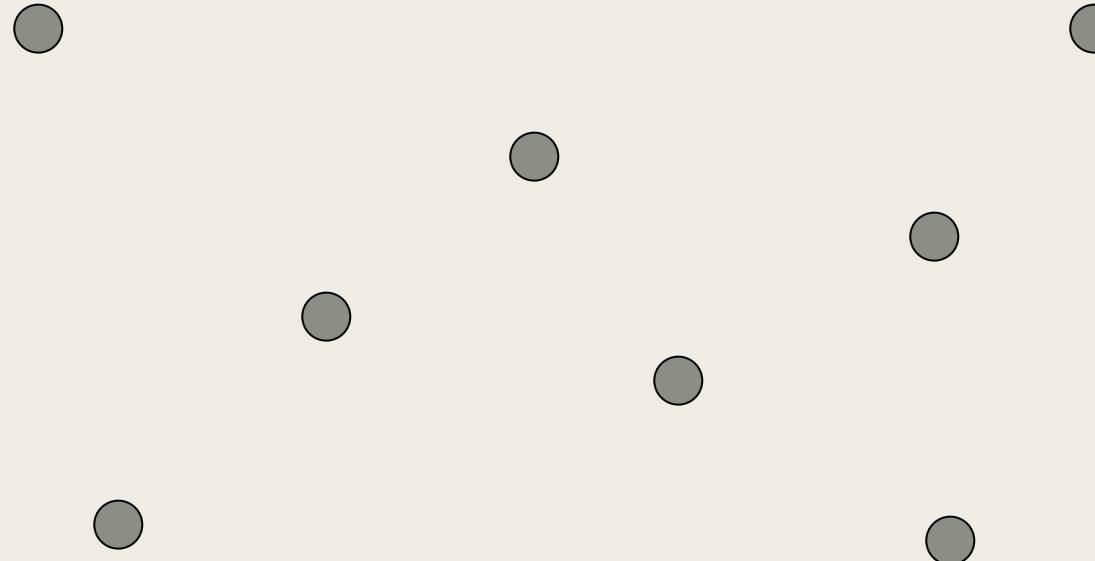
- $|V| = n, |E| = m$

	Adj Matrix	Adj List
test $(u, v) \in E$	$\Theta(1)$	$O(n)$
Degree(u)	$\Theta(n)$	$O(n)$
Memory	$\Theta(n^2)$	$\Theta(n+m)$
Edge insertion	$\Theta(1)$	$\Theta(1)$
Edge deletion	$\Theta(1)$	$O(n)$
Graph traversal	$\Theta(n^2)$	$\Theta(n+m)$

Both representations are very useful and have different properties, although adjacency lists are probably better for more problems

Problems

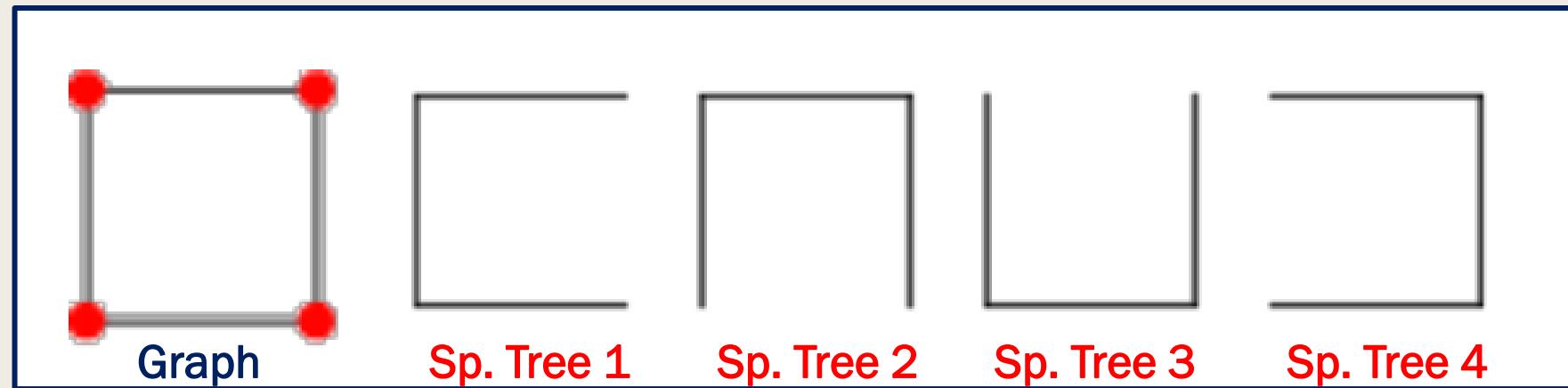
- Given a set of cities, how to construct minimum length of highways to connect the cities so that there are paths between every two cities?



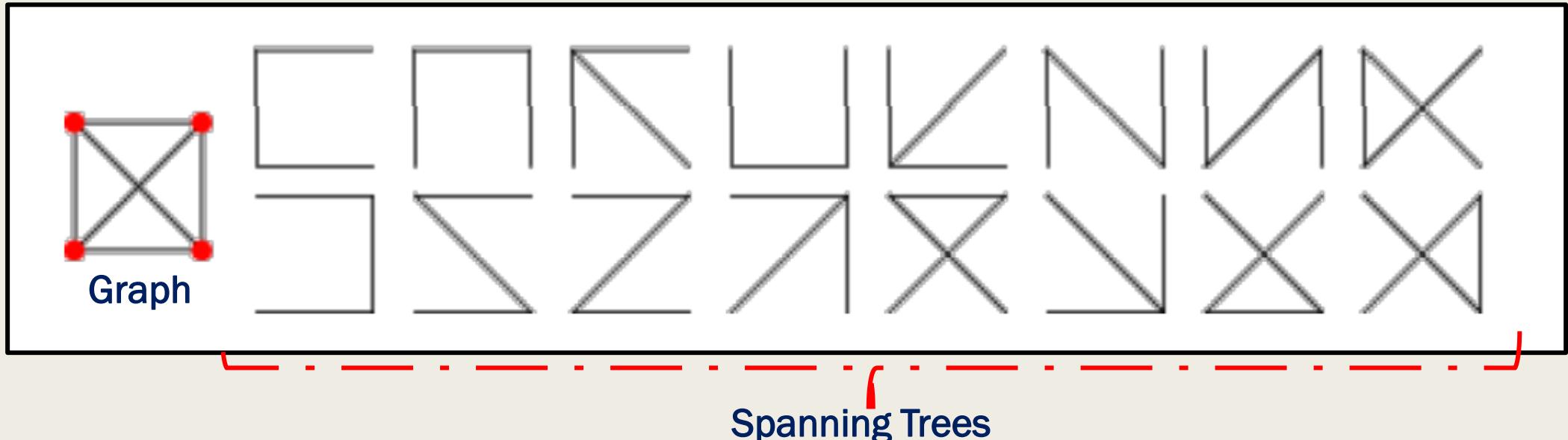
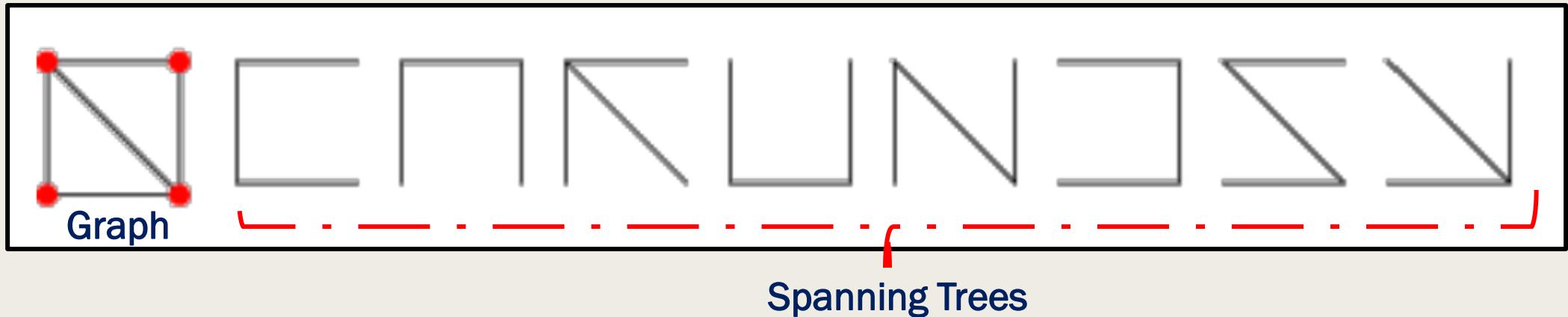
Spanning Tree

Definition

- Let $G = (V, E)$ be an undirected connected graph.
- A subgraph $t = (V, E')$ of G is a spanning tree of G iff t is a tree

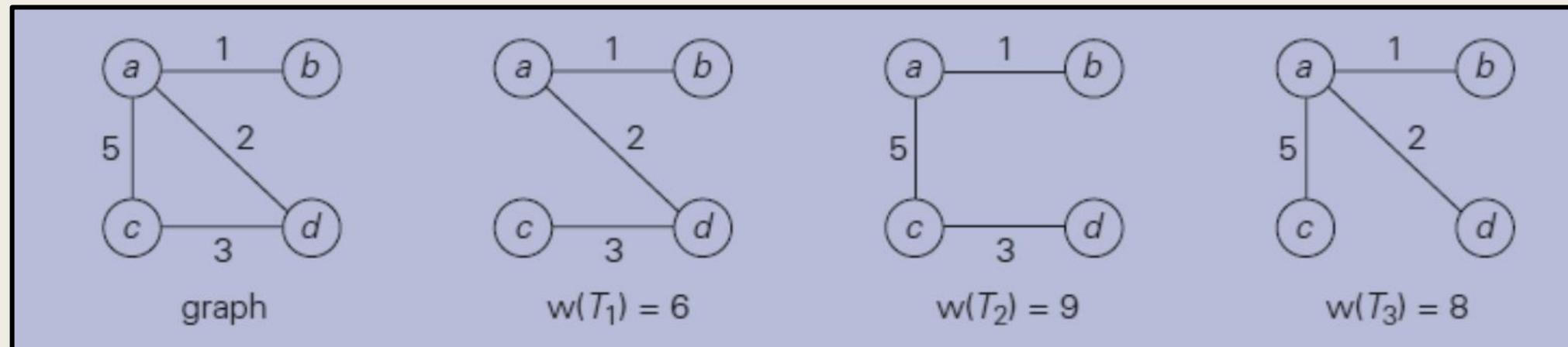


Spanning Tree

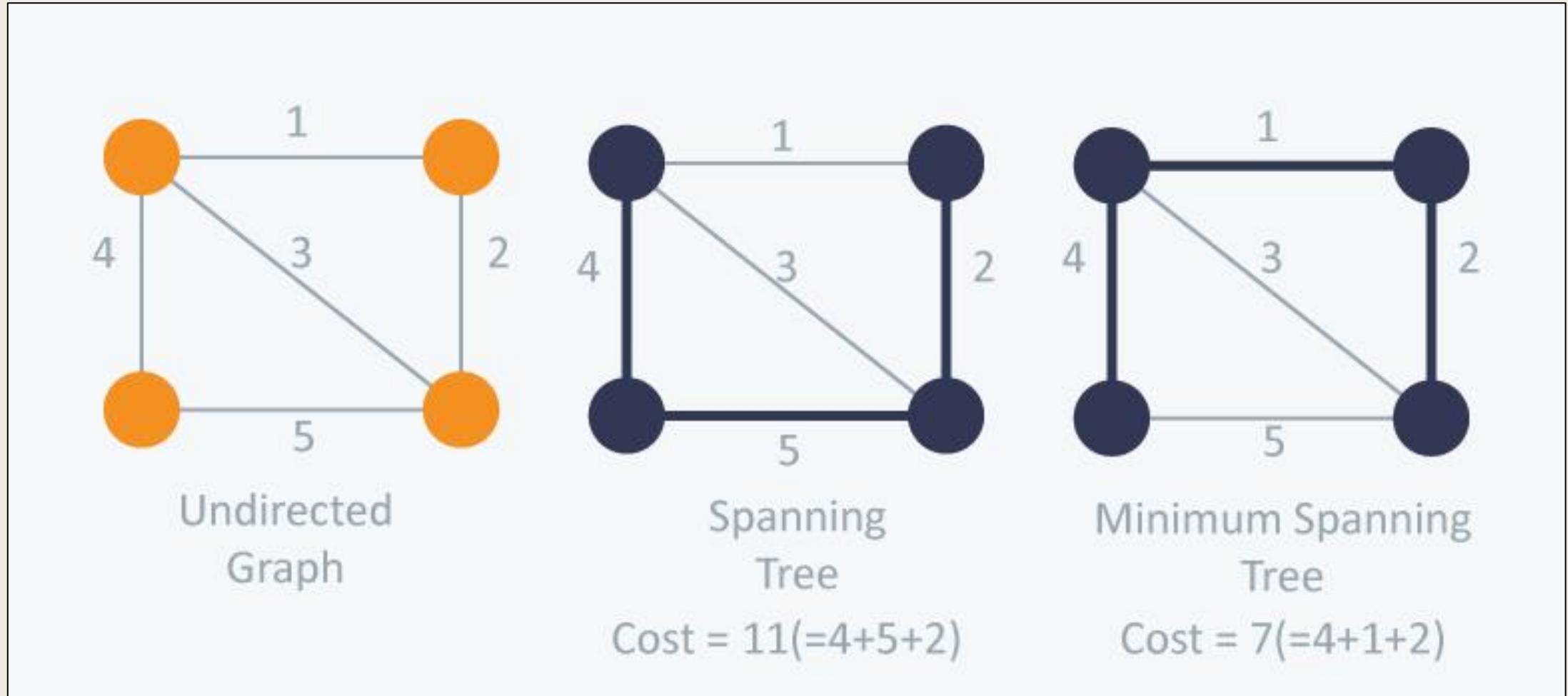


Spanning Tree

- A ***spanning tree*** of an undirected connected graph is its connected acyclic subgraph (i.e., a tree) that contains all the vertices of the graph.
- For a weighted graph, a ***minimum spanning tree*** is its spanning tree of the smallest weight, where the ***weight*** of a tree is defined as the sum of the weights on all its edges.
- The ***minimum spanning tree problem*** is the problem of finding a minimum spanning tree for a given weighted connected graph.



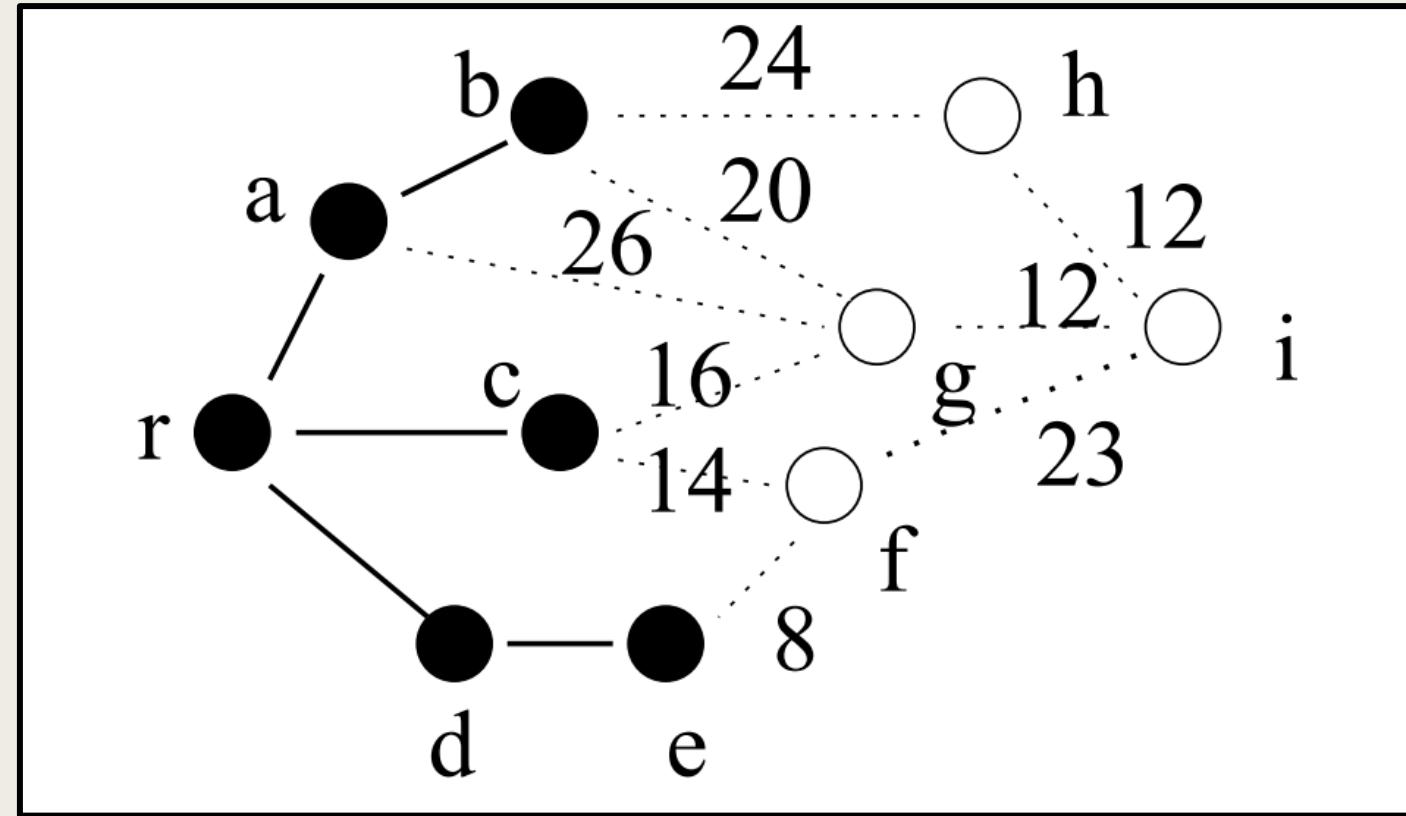
Spanning Tree



Prims Algorithm

- A greedy method to obtain a minimum-cost spanning tree builds this tree **edge by edge**.
- The next edge to include is chosen according to some optimization criterion.
- The simplest such criterion is to choose an edge that results in a **minimum increase** in the sum of the costs of the edges so far included

Prims Algorithm – Illustration



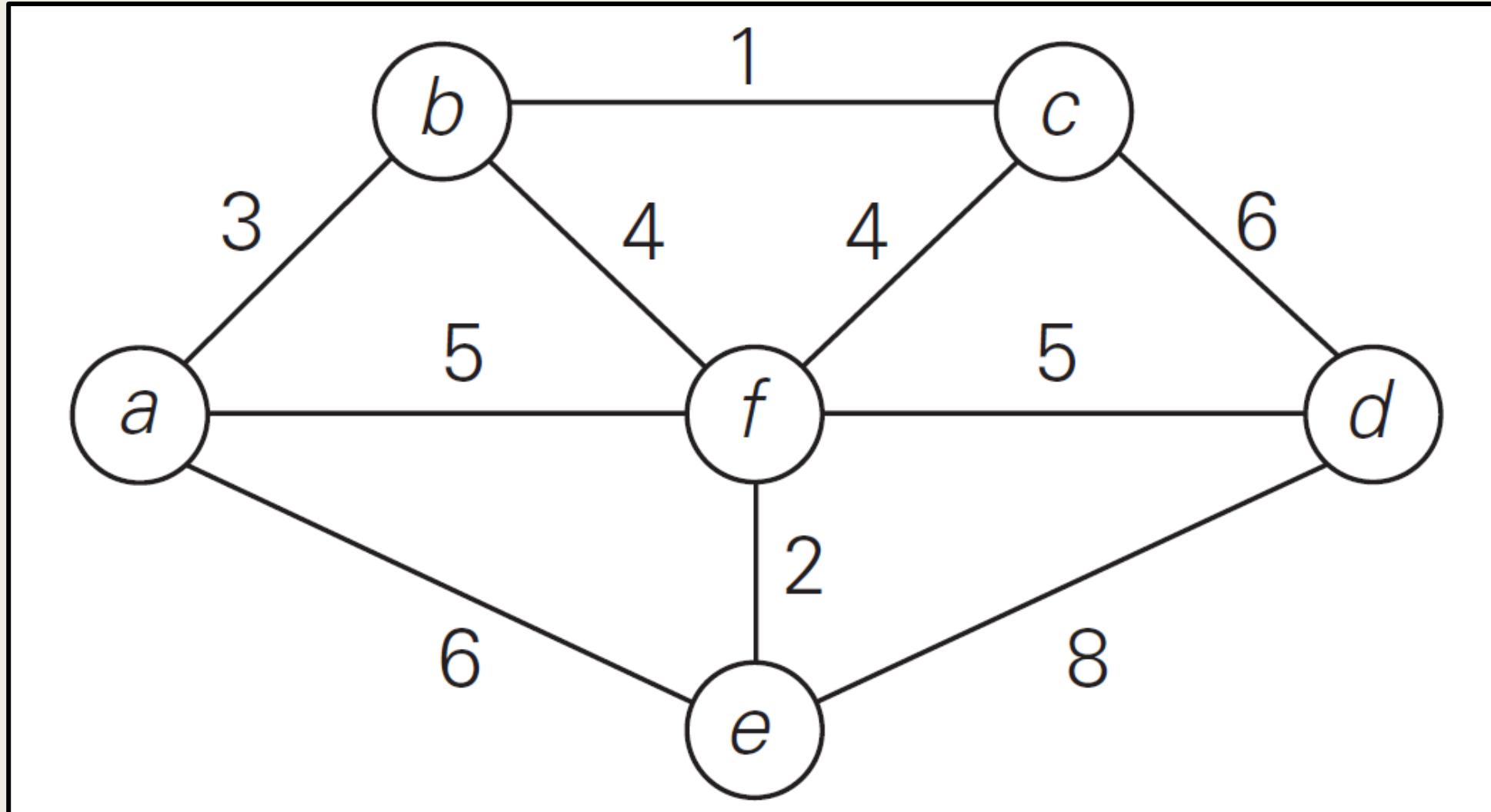
Tree Vertices - { **r, a, b, c, d, e** } – vertices that are part of the spanning tree

Fringe vertices - { **h, g, f** } – vertices not in the tree but adjacent to at least one vertex that is in the tree

Unseen vertices – { **i** } – vertices not yet affected by the algorithm

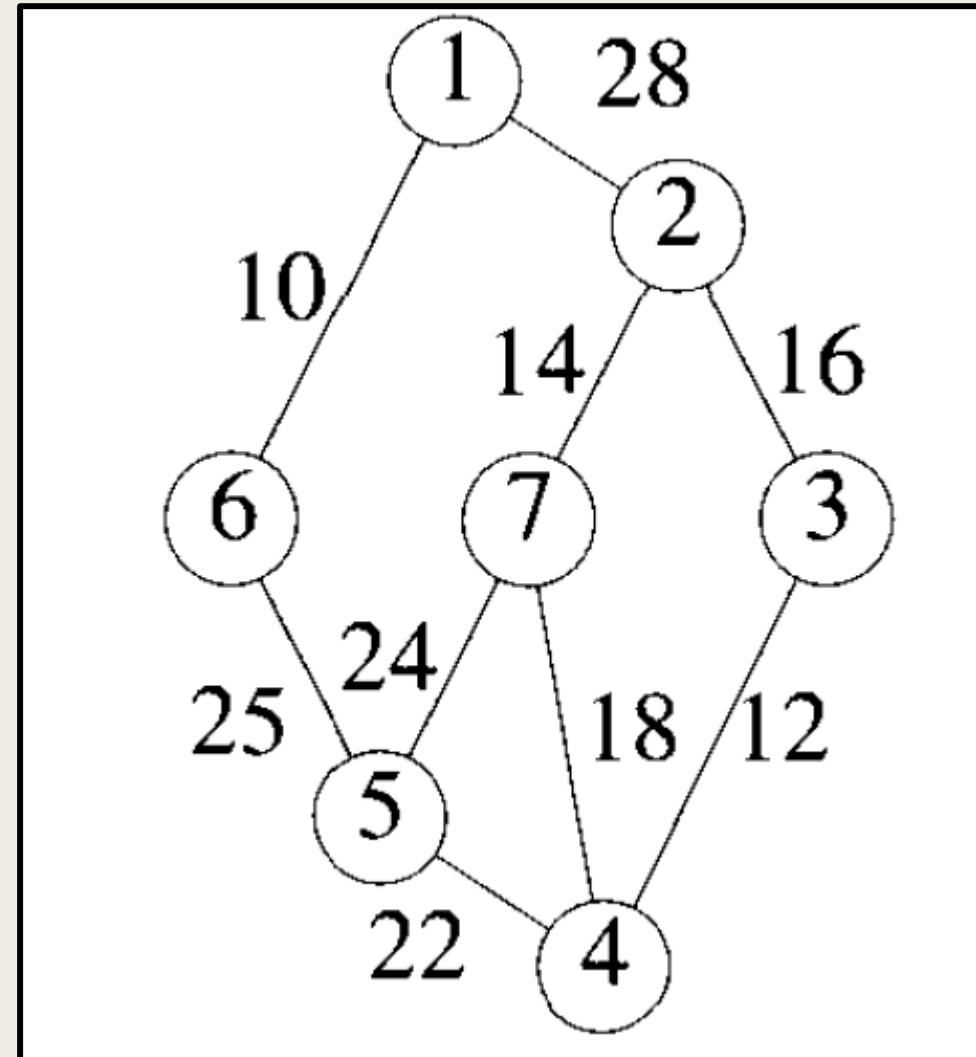
Prims Algorithm – Problem

Find the minimum cost spanning tree for the following graph

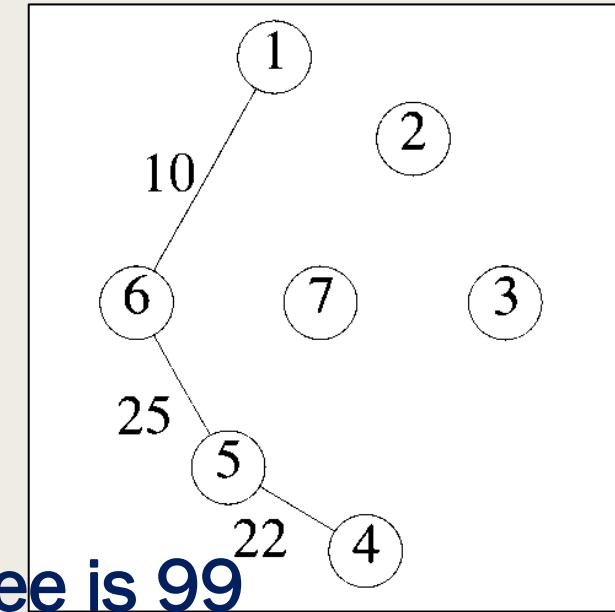
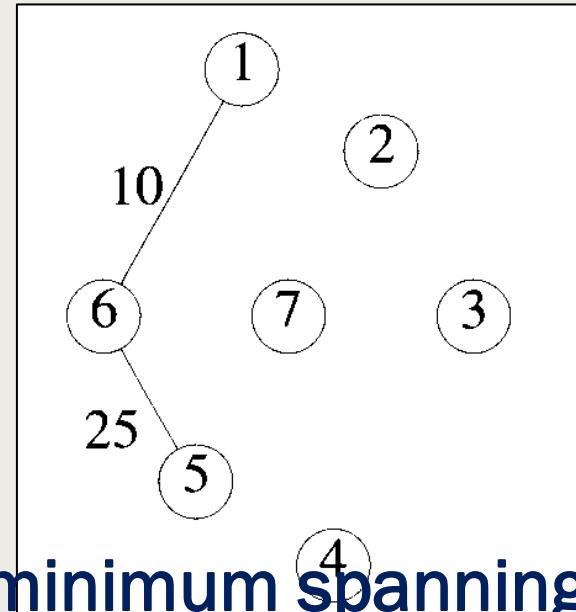
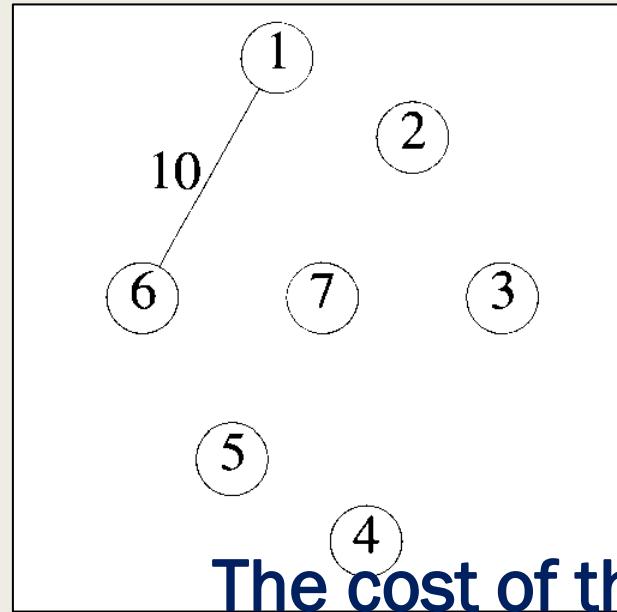


Prims Algorithm – Problem

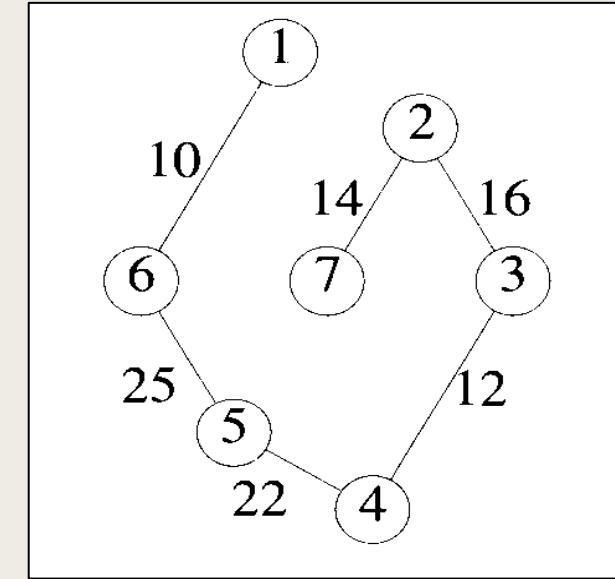
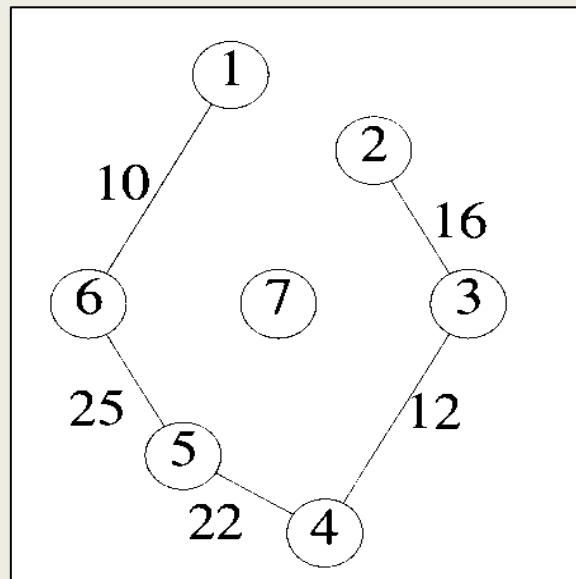
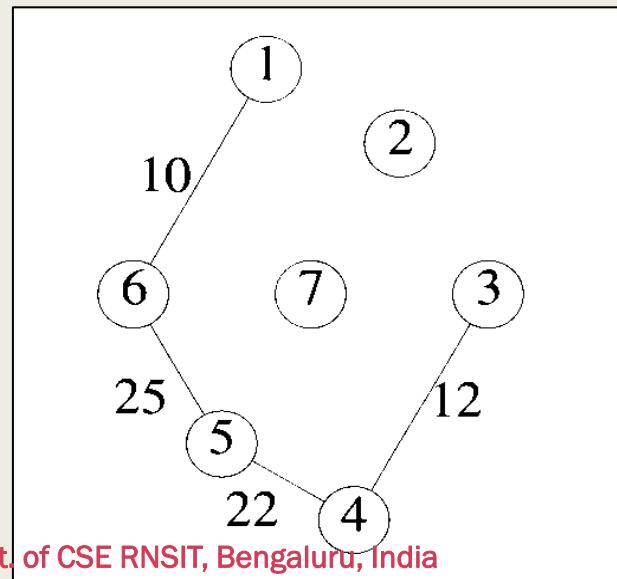
Find the minimum cost spanning tree for the following graph



Prims Algorithm- Example

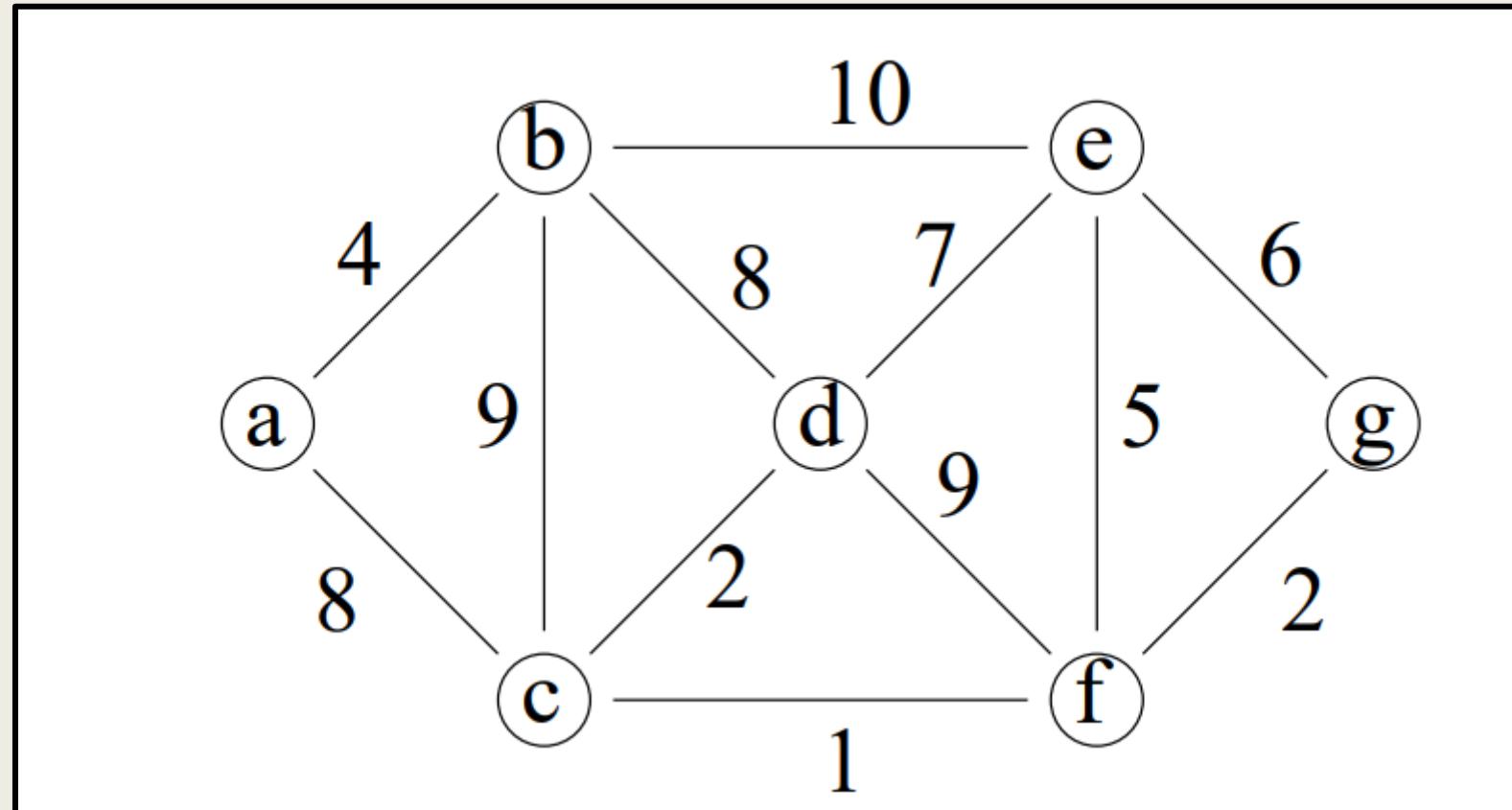


The cost of the minimum spanning tree is 99

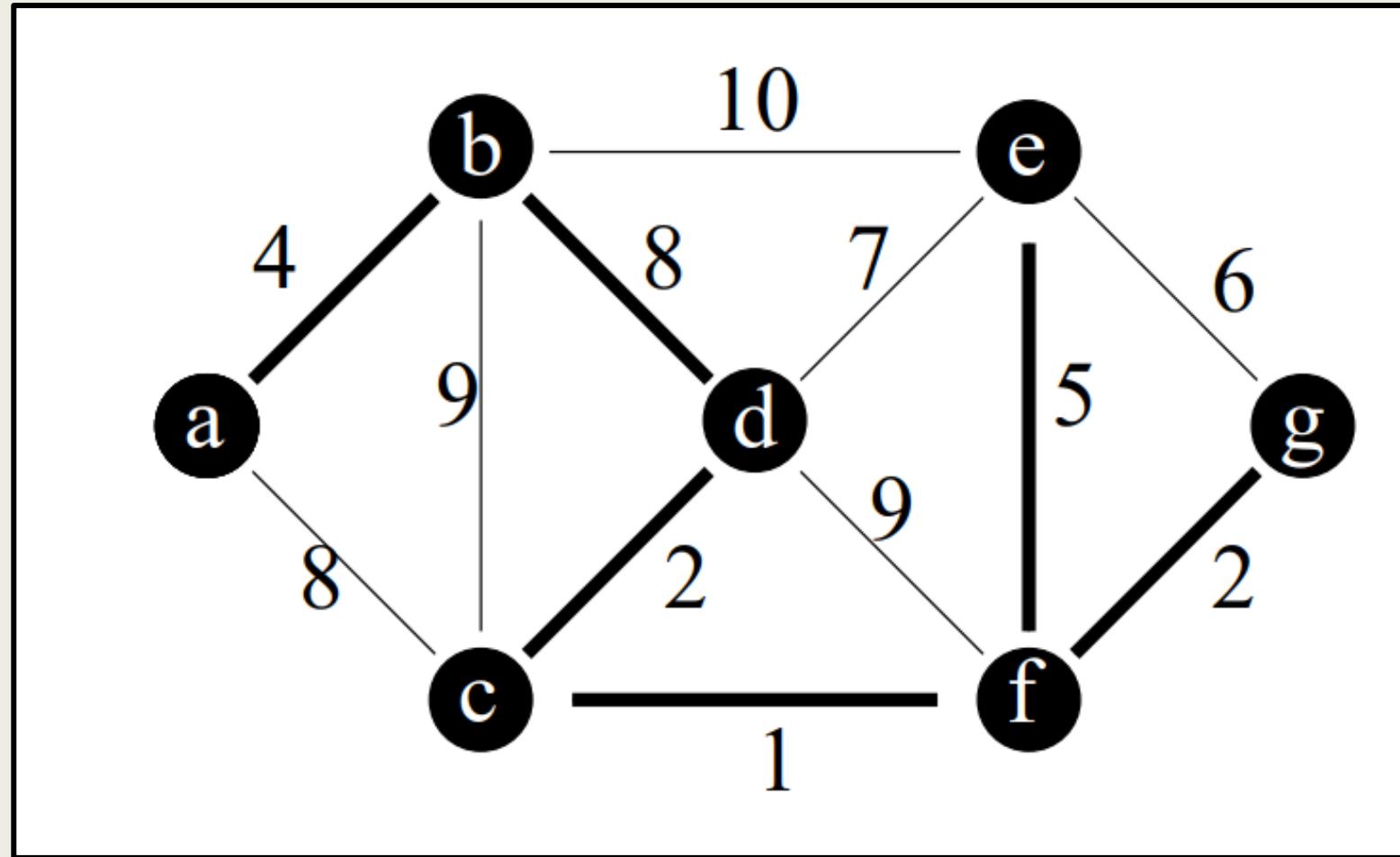


Prims Algorithm – Problem Home work

Find the minimum cost spanning tree for the following graph



Prims Algorithm – Problem Home work – solution



The cost of the minimum spanning tree is 22

Prims Algorithm

ALGORITHM $Prim(G)$

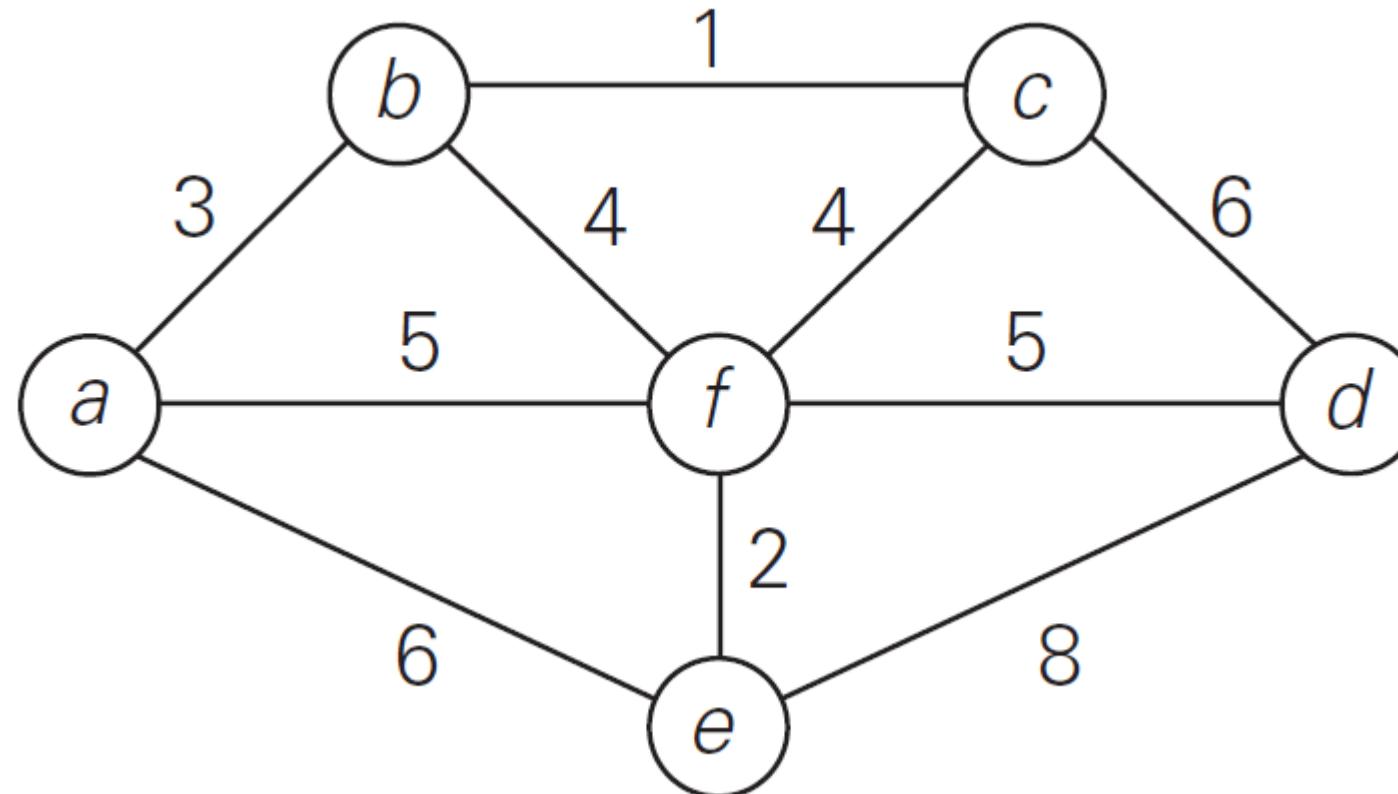
```
//Prim's algorithm for constructing a minimum spanning tree
//Input: A weighted connected graph  $G = \langle V, E \rangle$ 
//Output:  $E_T$ , the set of edges composing a minimum spanning tree of  $G$ 
 $V_T \leftarrow \{v_0\}$  //the set of tree vertices can be initialized with any vertex
 $E_T \leftarrow \emptyset$ 
for  $i \leftarrow 1$  to  $|V| - 1$  do
    find a minimum-weight edge  $e^* = (v^*, u^*)$  among all the edges  $(v, u)$ 
    such that  $v$  is in  $V_T$  and  $u$  is in  $V - V_T$ 
     $V_T \leftarrow V_T \cup \{u^*\}$ 
     $E_T \leftarrow E_T \cup \{e^*\}$ 
return  $E_T$ 
```

Kruskal's Algorithm

- Joseph Kruskal developed an algorithm - **Kruskal's Algorithm** - to find an optimal solution to minimum spanning tree problem when he was a **second year graduate student**
 - Prim's algorithm grew the minimum spanning tree by including nearest vertex to the vertices already in the tree
 - In Prim's algorithm, the subgraphs were always connected in the intermediate stages
- The **Kruskal's algorithm** begins by sorting the edges in **increasing order of their weights**
- Starting with the **empty subgraph**, it scans this sorted list, **adding the next edge** on the list to the current subgraph if such an **inclusion does not create a cycle** and simply **skipping the edge** otherwise

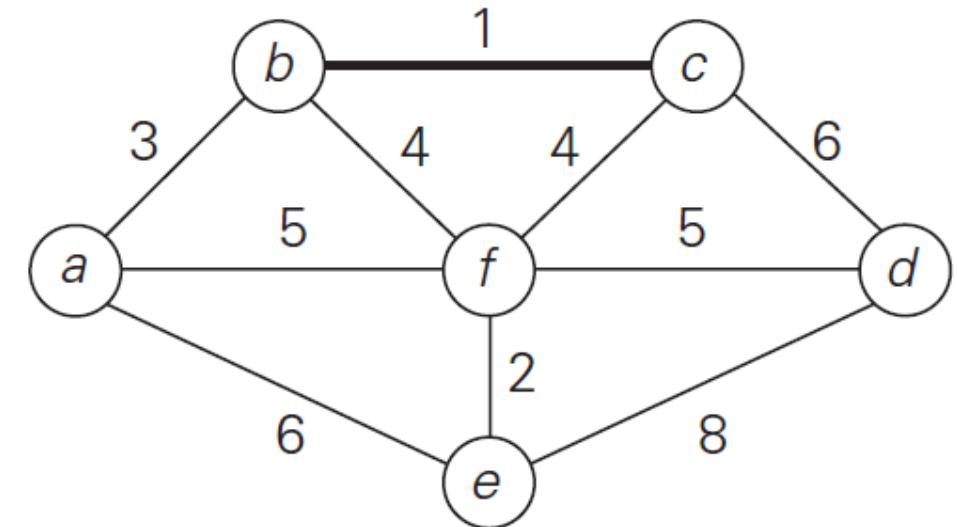
Kruskal's Algorithm - Problem

Find the minimum cost spanning tree for the following graph



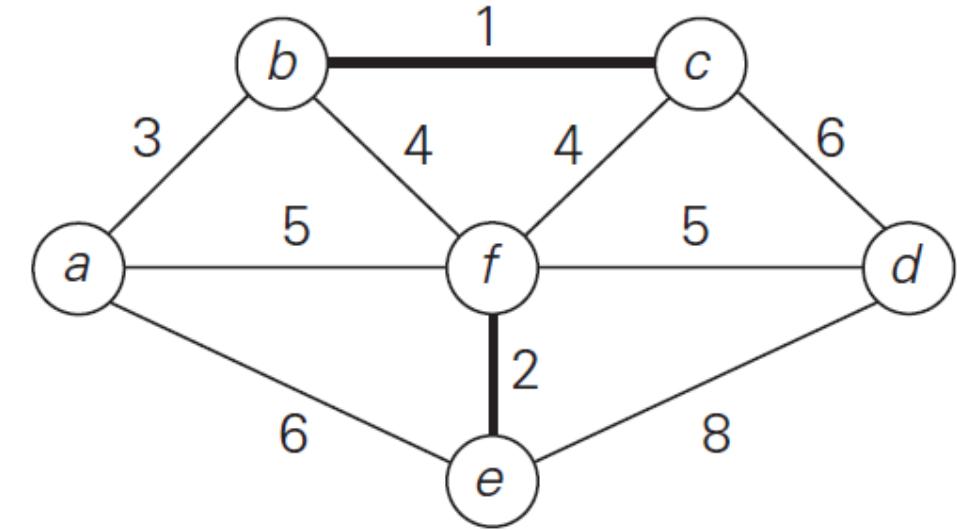
Kruskal's Algorithm - Solution

bc	ef	ab	bf	cf	af	df	ae	cd	de
1	2	3	4	4	5	5	6	6	8



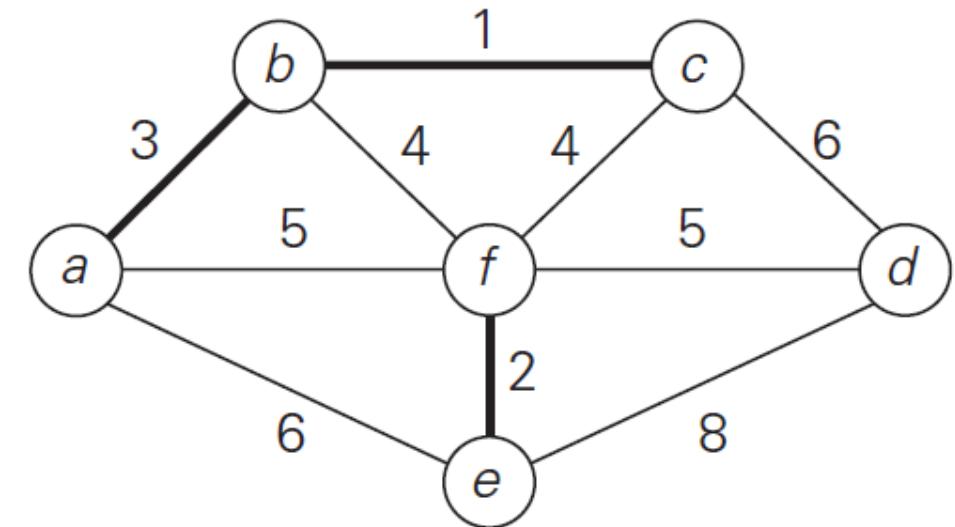
Kruskal's Algorithm - Solution

bc	ef	ab	bf	cf	af	df	ae	cd	de
1	2	3	4	4	5	5	6	6	8



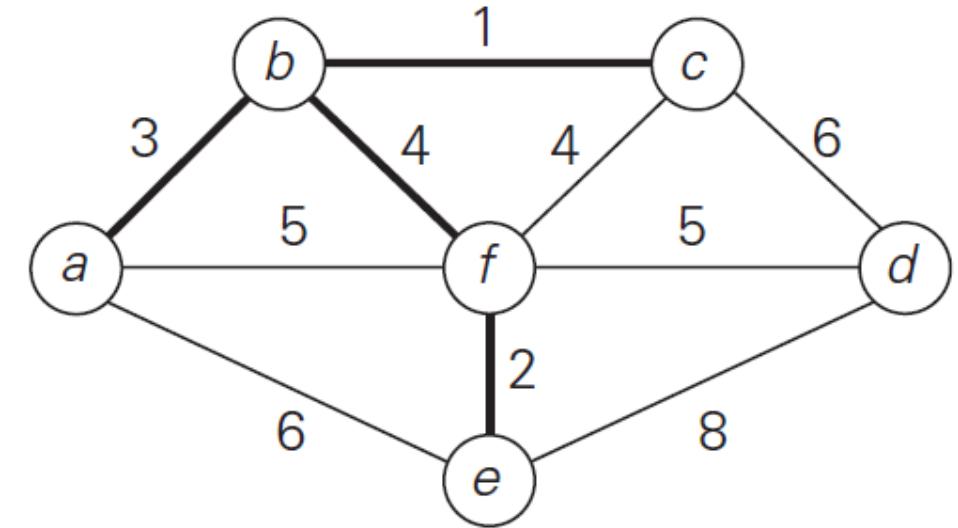
Kruskal's Algorithm - Solution

bc	ef	ab	bf	cf	af	df	ae	cd	de
1	2	3	4	4	5	5	6	6	8



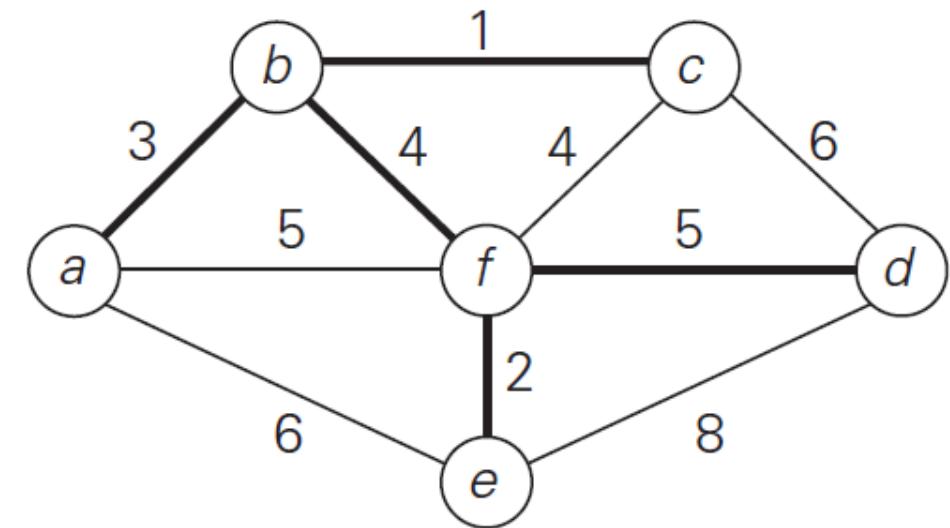
Kruskal's Algorithm - Solution

bc	ef	ab	bf	cf	af	df	ae	cd	de
1	2	3	4	4	5	5	6	6	8



Kruskal's Algorithm - Solution

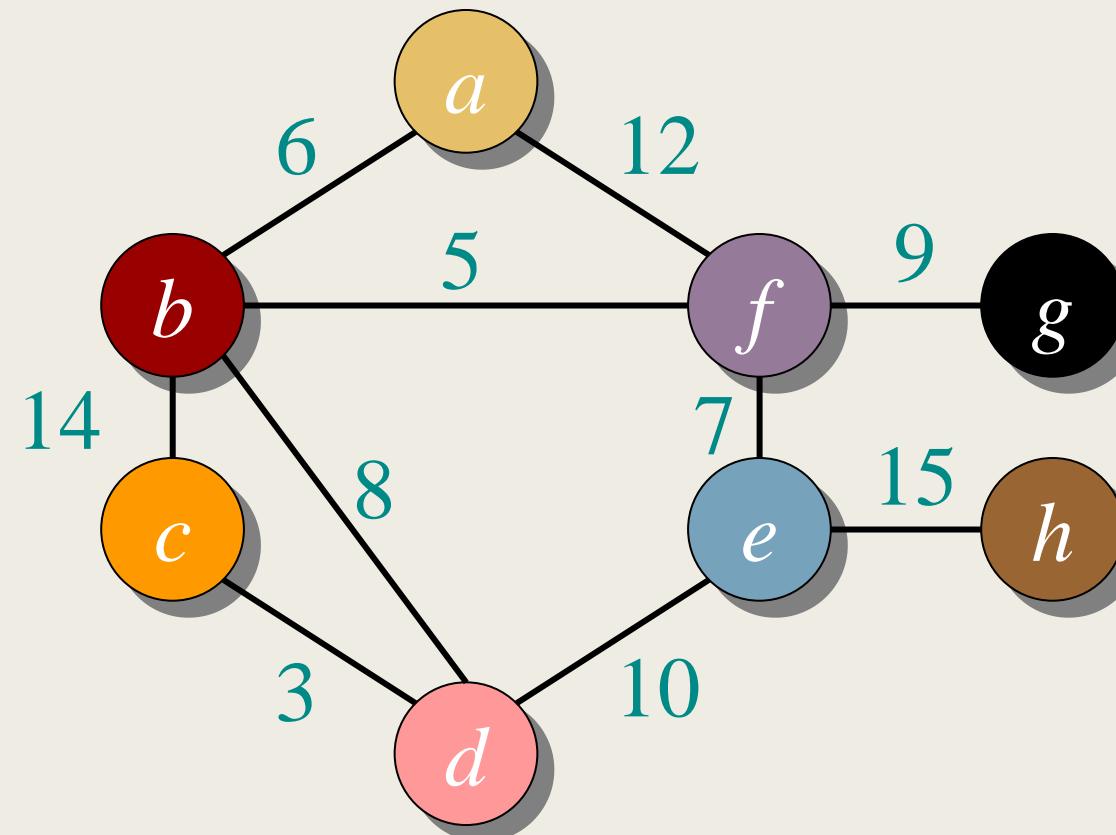
bc ef ab bf cf af **df** ae cd de
1 2 3 4 4 5 5 6 6 8



Total cost of the minimum spanning tree= 15

Kruskal's Algorithm - Problem

- Find the minimum cost spanning tree for the following graph



Kruskal's Algorithm - Solution

c-d: 3

b-f: 5

b-a: 6

f-e: 7

b-d: 8

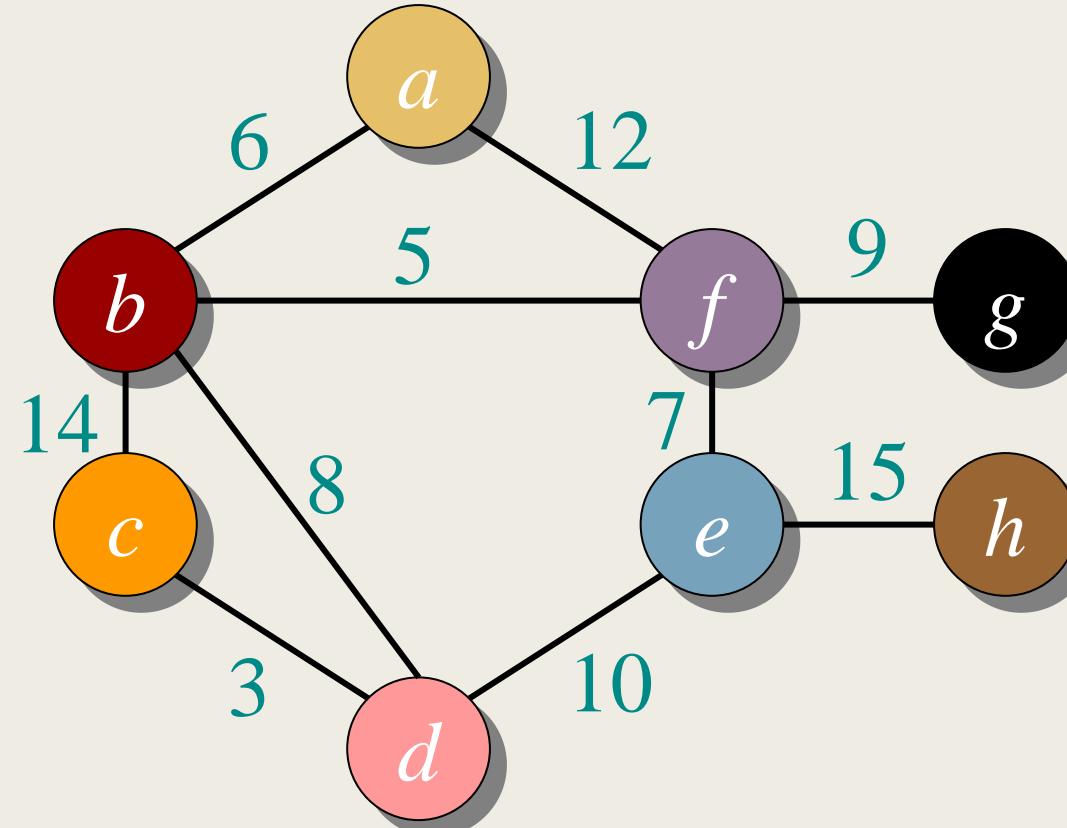
f-g: 9

d-e: 10

a-f: 12

b-c: 14

e-h: 15



Kruskal's Algorithm - Solution

c-d: 3

b-f: 5

b-a: 6

f-e: 7

b-d: 8

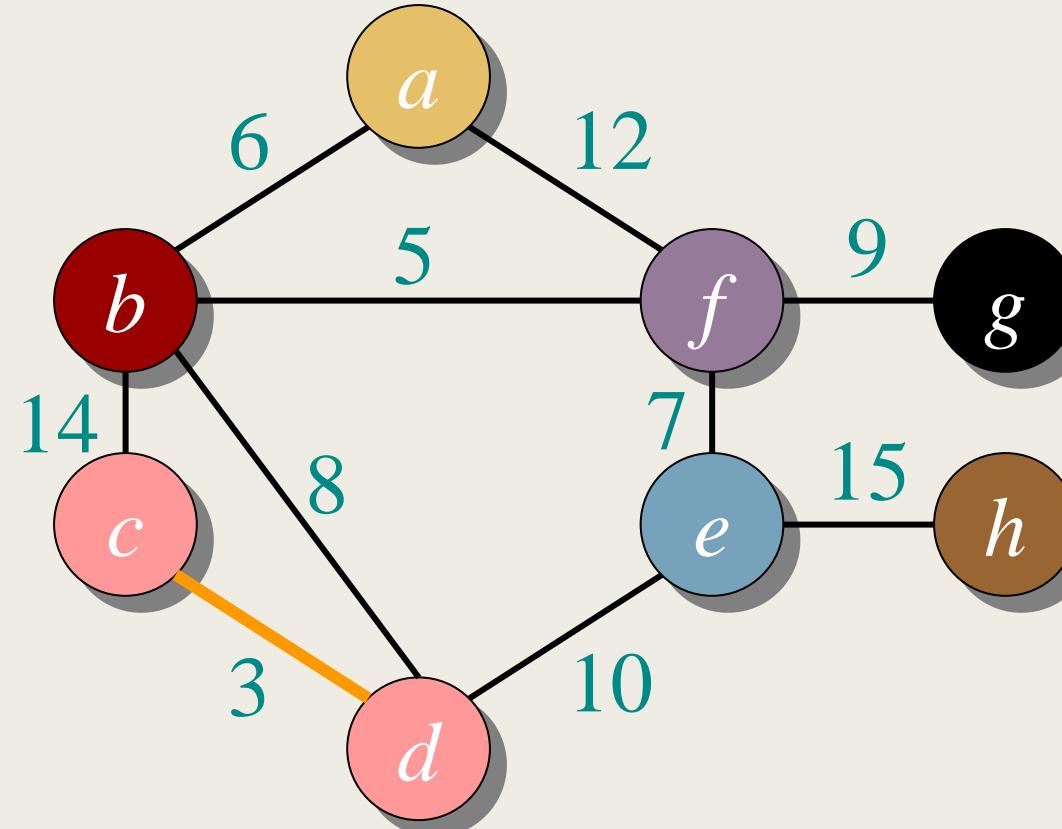
f-g: 9

d-e: 10

a-f: 12

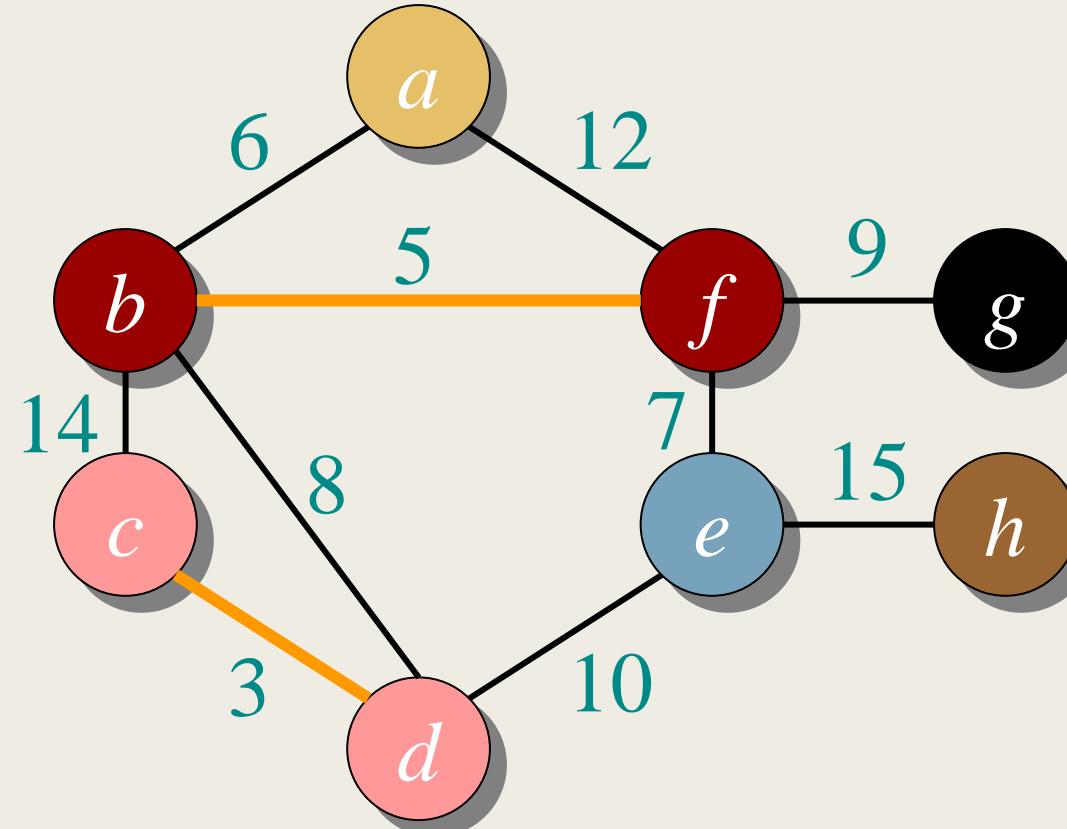
b-c: 14

e-h: 15



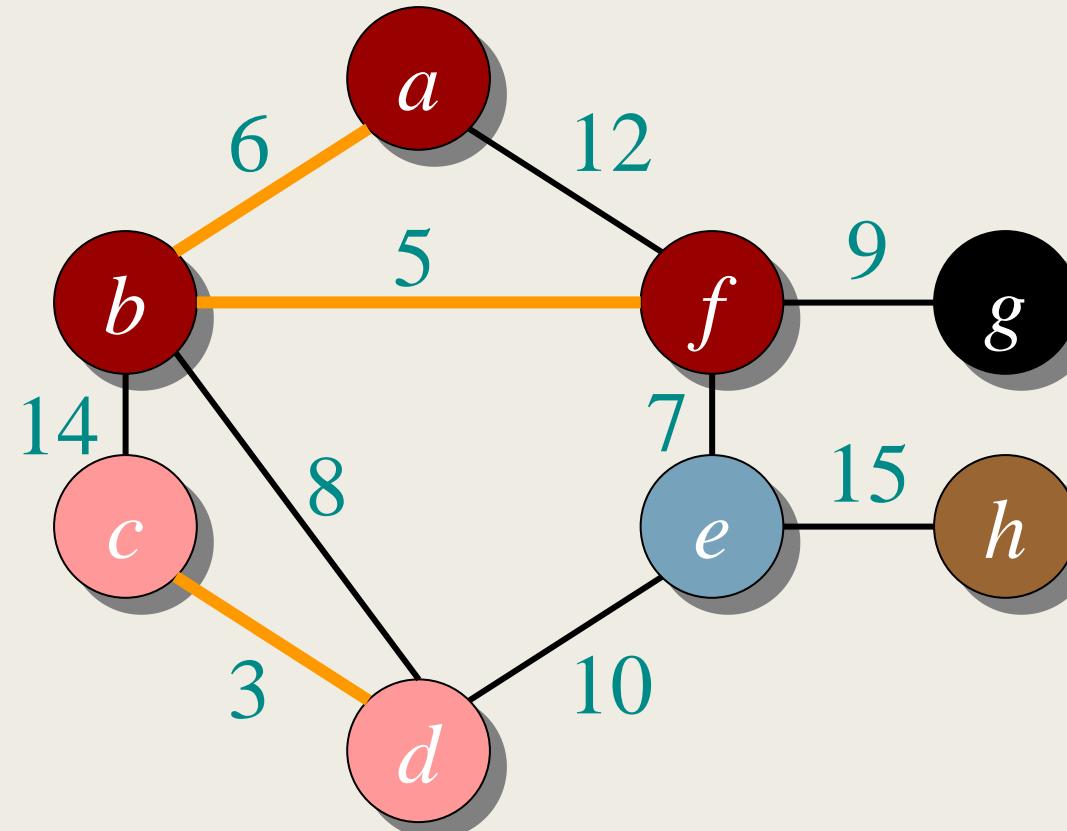
Kruskal's Algorithm - Solution

c-d:	3
b-f:	5
b-a:	6
f-e:	7
b-d:	8
f-g:	9
d-e:	10
a-f:	12
b-c:	14
e-h:	15



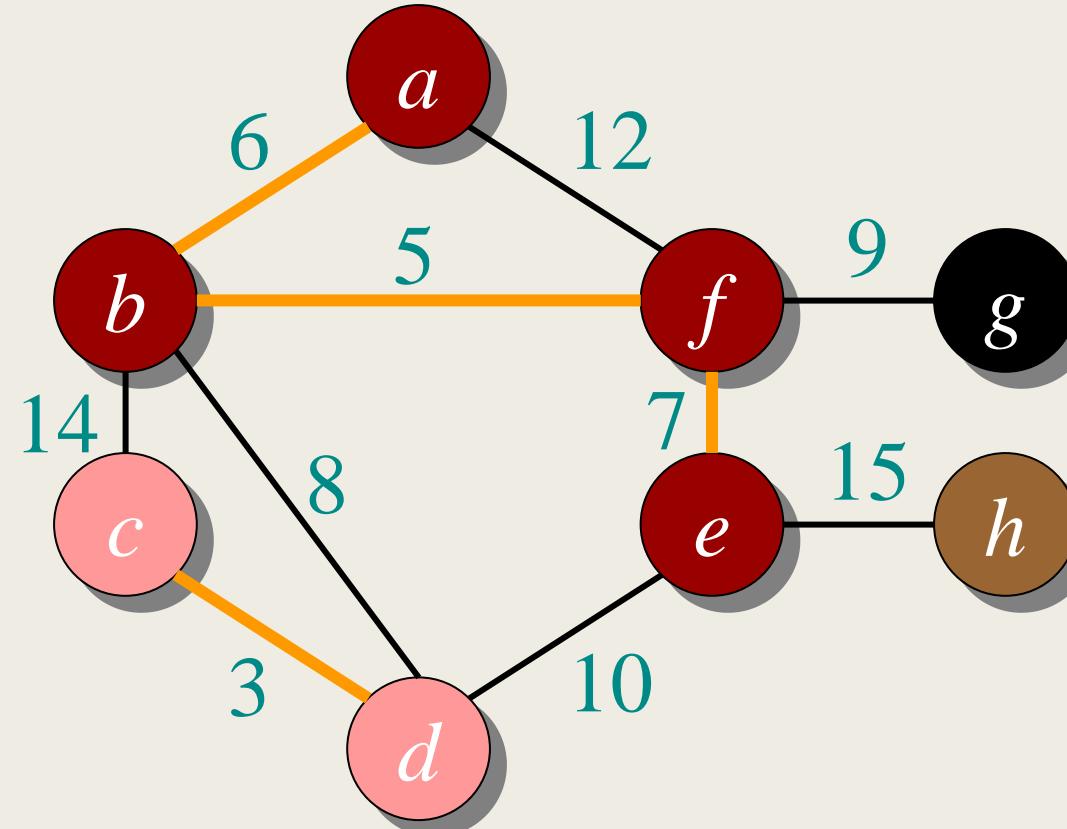
Kruskal's Algorithm - Solution

c-d:	3
b-f:	5
b-a:	6
f-e:	7
b-d:	8
f-g:	9
d-e:	10
a-f:	12
b-c:	14
e-h:	15



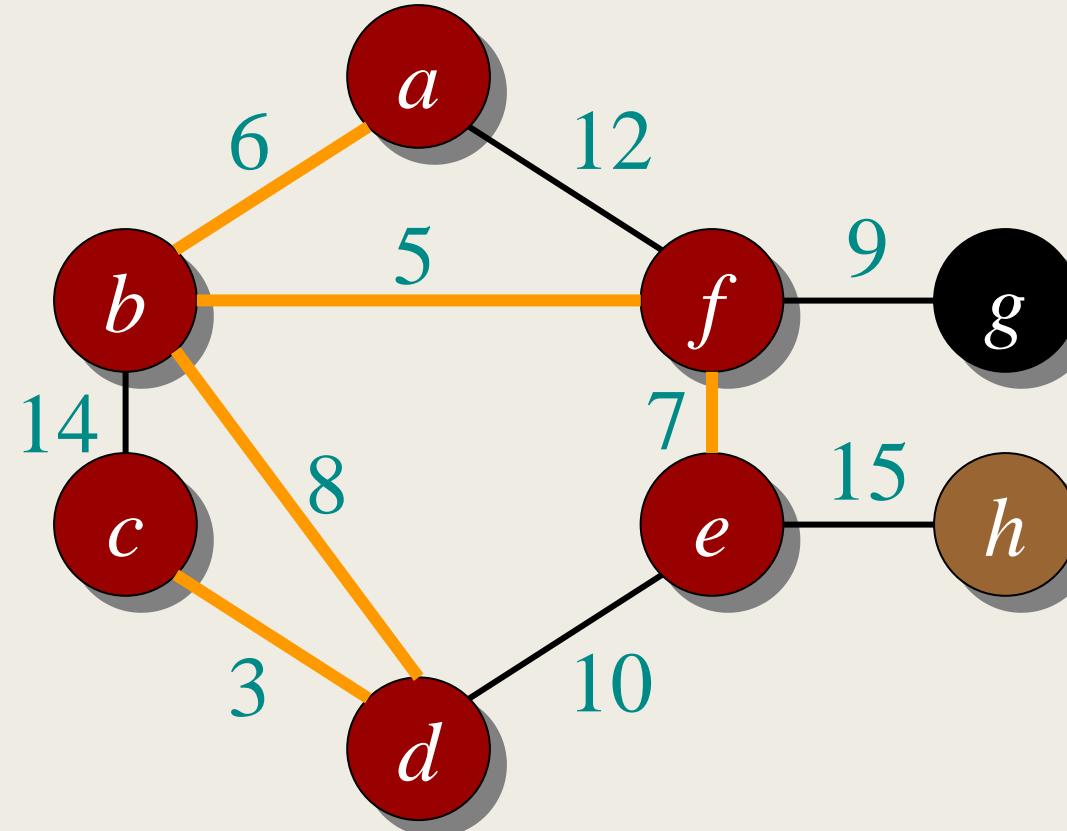
Kruskal's Algorithm - Solution

c-d:	3
b-f:	5
b-a:	6
f-e:	7
b-d:	8
f-g:	9
d-e:	10
a-f:	12
b-c:	14
e-h:	15



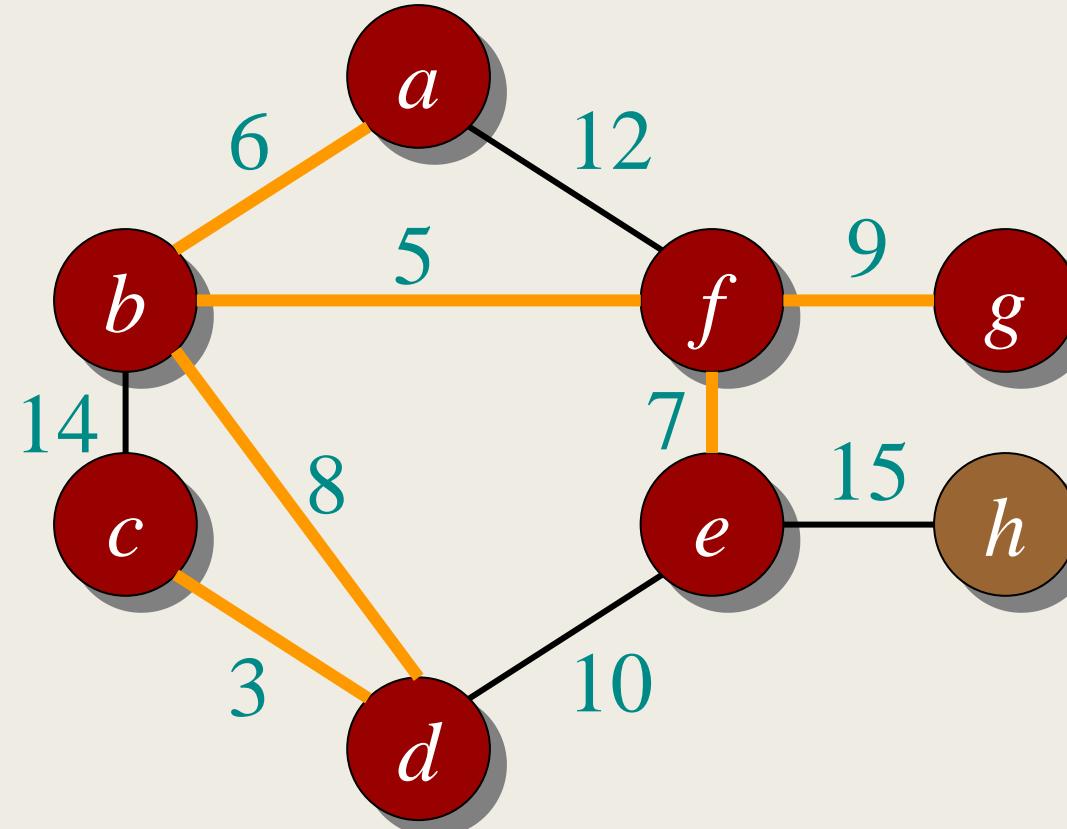
Kruskal's Algorithm - Solution

c-d:	3
b-f:	5
b-a:	6
f-e:	7
b-d:	8
f-g:	9
d-e:	10
a-f:	12
b-c:	14
e-h:	15



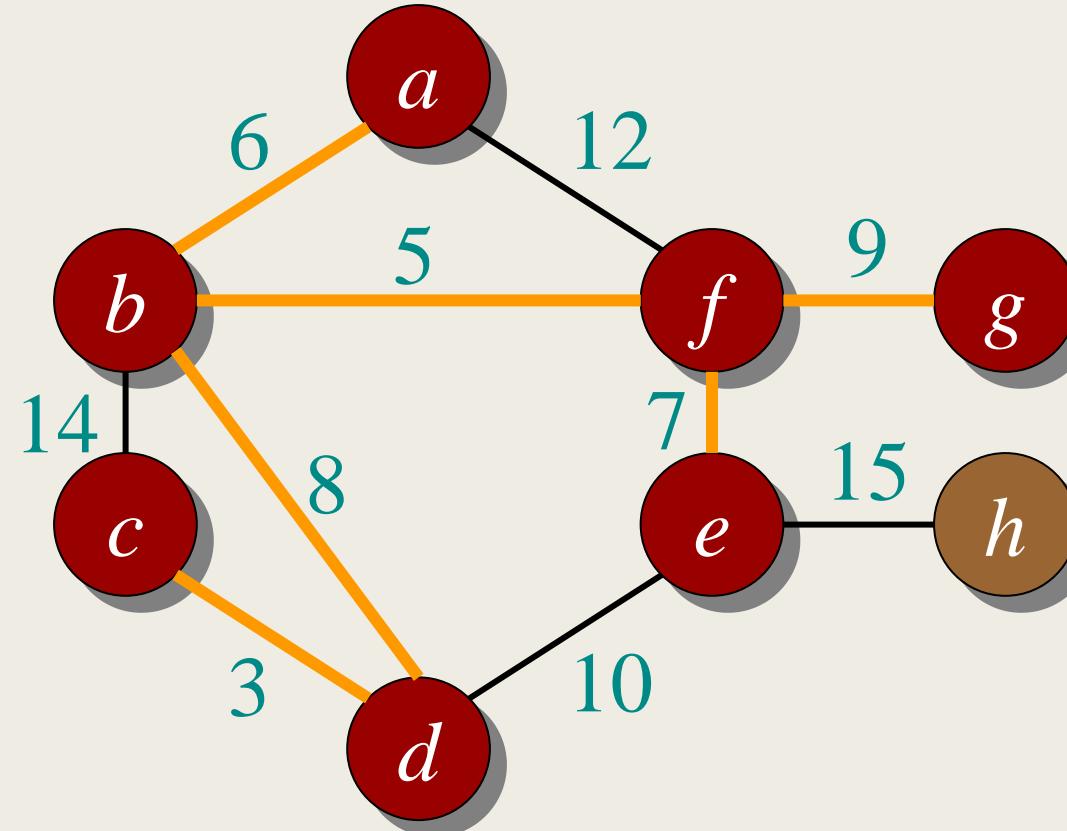
Kruskal's Algorithm - Solution

c-d:	3
b-f:	5
b-a:	6
f-e:	7
b-d:	8
f-g:	9
d-e:	10
a-f:	12
b-c:	14
e-h:	15



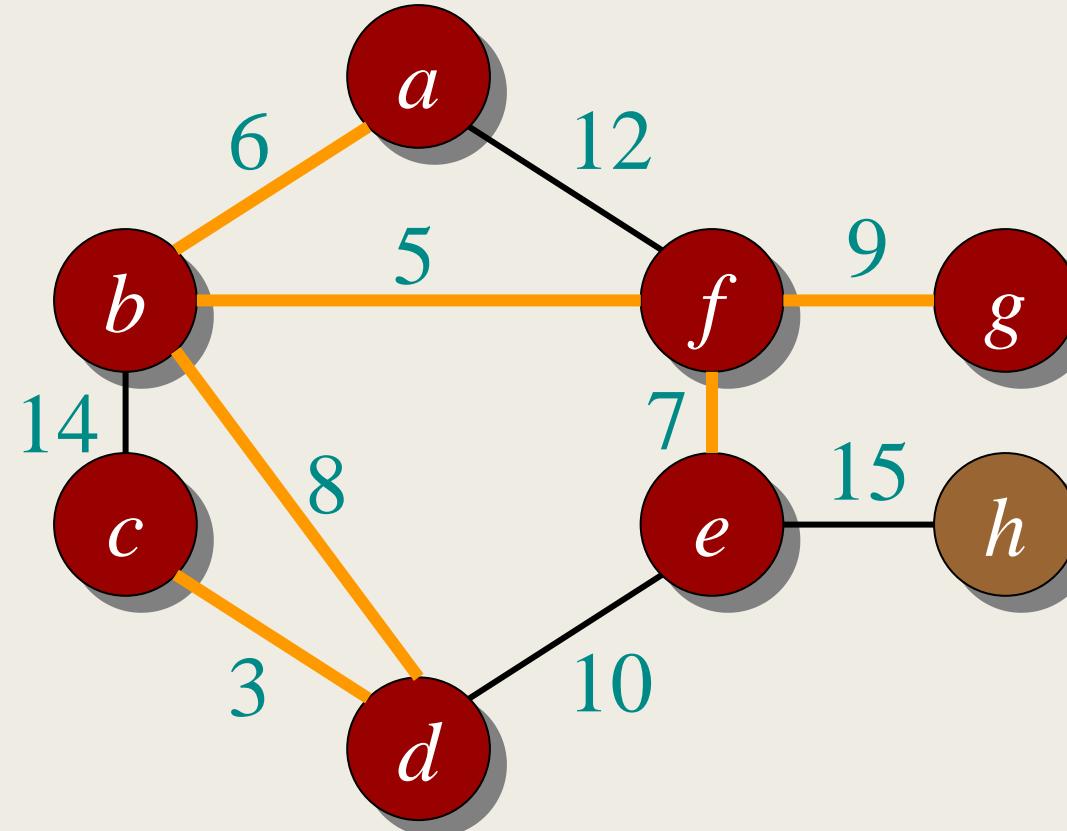
Kruskal's Algorithm - Solution

c-d:	3
b-f:	5
b-a:	6
f-e:	7
b-d:	8
f-g:	9
d-e:	10
a-f:	12
b-c:	14
e-h:	15



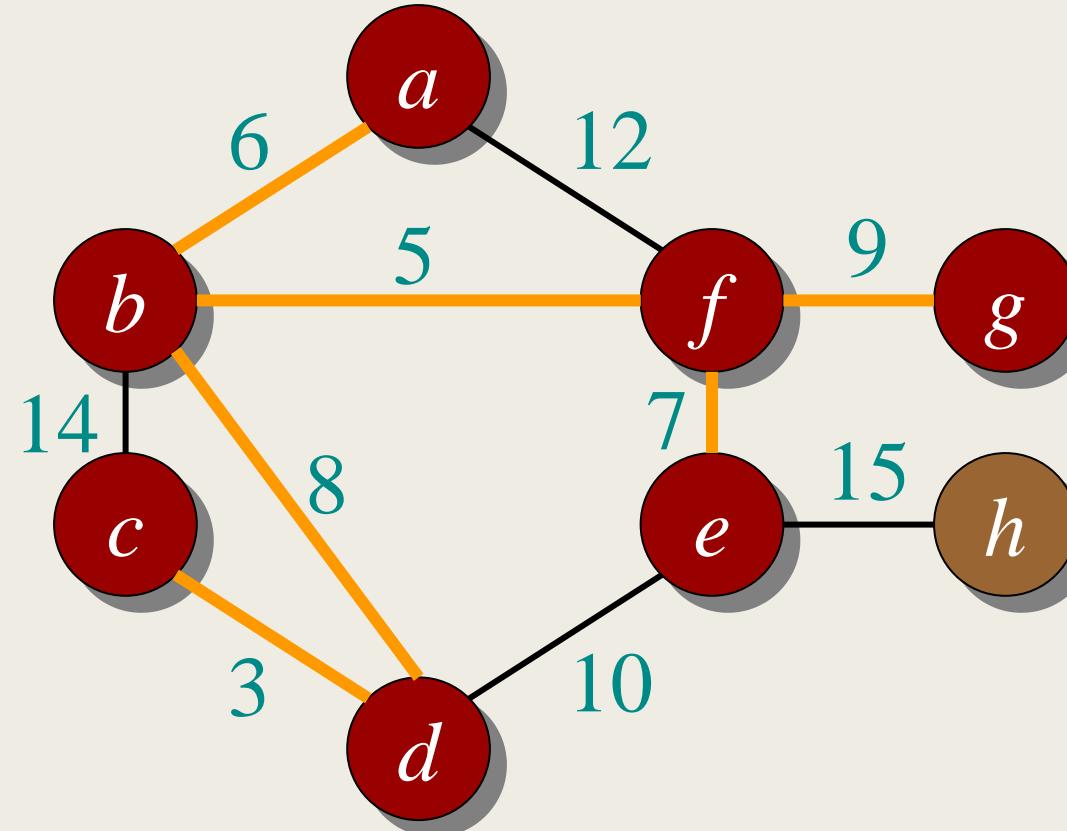
Kruskal's Algorithm - Solution

c-d:	3
b-f:	5
b-a:	6
f-e:	7
b-d:	8
f-g:	9
d-e:	10
a-f:	12
b-c:	14
e-h:	15



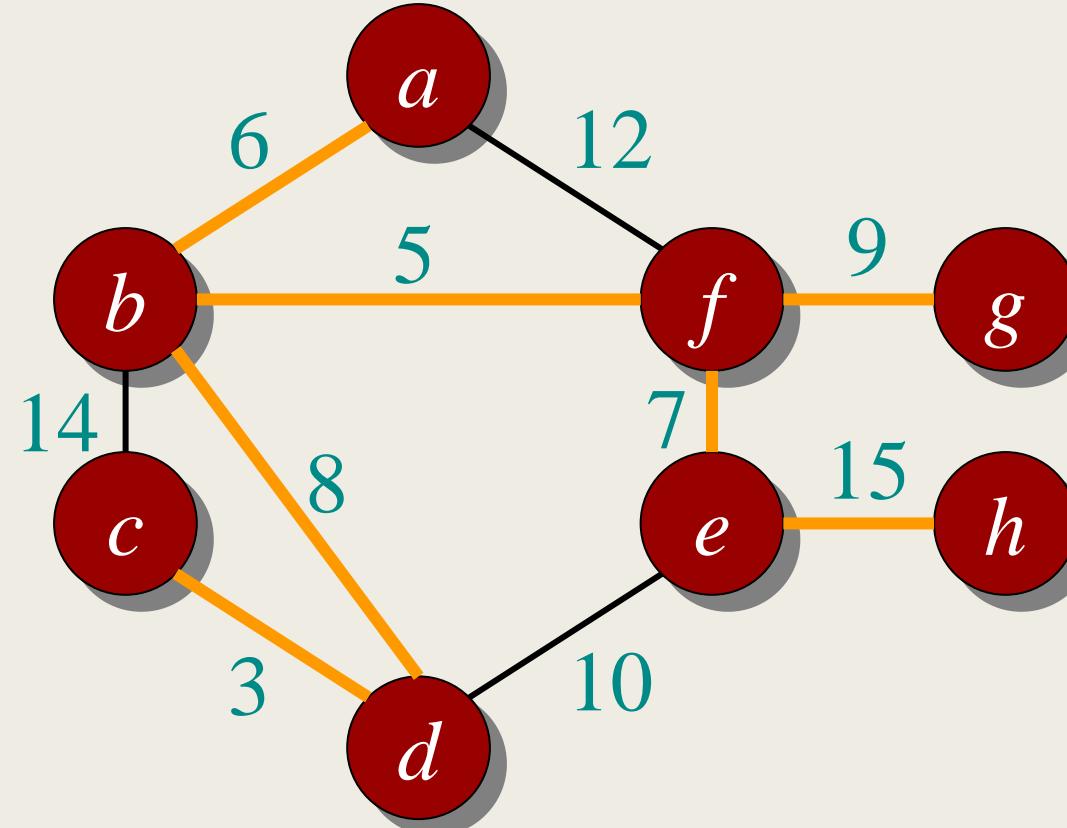
Kruskal's Algorithm - Solution

c-d:	3
b-f:	5
b-a:	6
f-e:	7
b-d:	8
f-g:	9
d-e:	10
a-f:	12
b-c:	14
e-h:	15



Kruskal's Algorithm - Solution

c-d:	3
b-f:	5
b-a:	6
f-e:	7
b-d:	8
f-g:	9
d-e:	10
a-f:	12
b-c:	14
e-h:	15



Total cost of the minimum spanning tree= 53

Kruskal's Algorithm

ALGORITHM Kruskal(G)

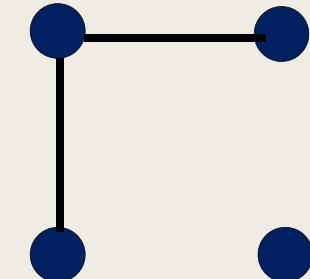
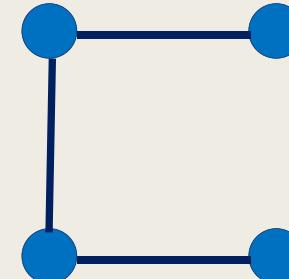
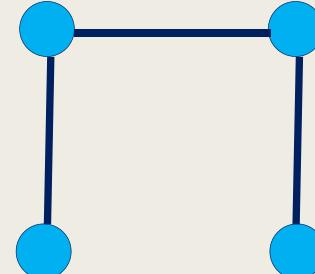
```

//Kruskal's algorithm for constructing a minimum spanning tree
//Input: A weighted connected graph  $G = \langle V, E \rangle$ 
//Output:  $E_T$ , the set of edges composing a minimum spanning tree of  $G$ 
sort  $E$  in nondecreasing order of the edge weights  $w(e_{i_1}) \leq \dots \leq w(e_{i_{|E|}})$ 
 $E_T \leftarrow \emptyset$ ;  $eCounter \leftarrow 0$       //initialize the set of tree edges and its size
 $k \leftarrow 0$                           //initialize the number of processed edges
while  $eCounter < |V| - 1$  do
     $k \leftarrow k + 1$ 
    if  $E_T \cup \{e_{i_k}\}$  is acyclic
         $E_T \leftarrow E_T \cup \{e_{i_k}\}$ ;  $eCounter \leftarrow eCounter + 1$ 
return  $E_T$ 
```

Kruskal's Algorithm

The implementation catch

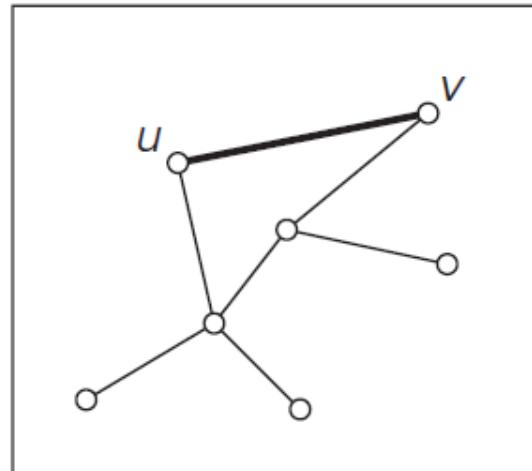
- Kruskal's approach gives away an impression of being very easy to implement. It is really so?
- Each time / iteration the algorithm should check whether adding the next edge to the edges already selected creates a cycle?
- Remember a cycle is created iff the new edge connects two vertices which are already connected by a path



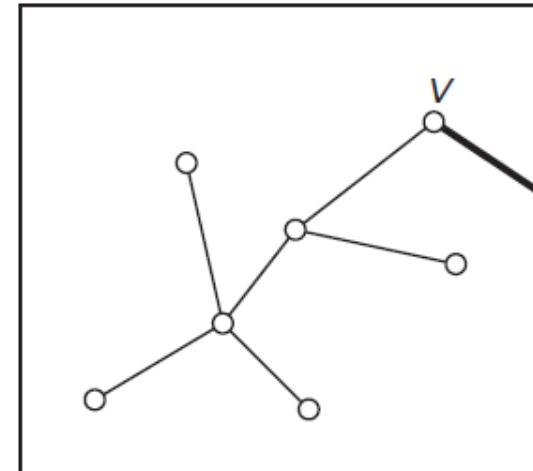
Kruskal's Algorithm

The implementation catch

- New cycle is created if and only if the two vertices belong to the same connected component
- Each connected component of a subgraph generated by Kruskal's algorithm is a tree because it has no cycles.



(a)



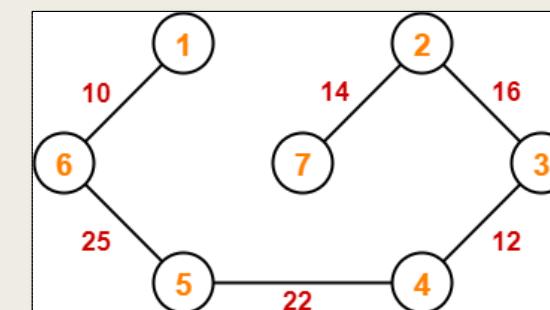
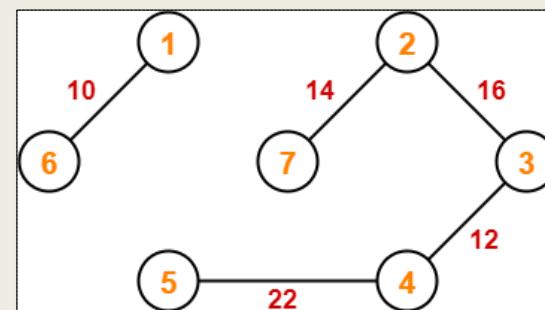
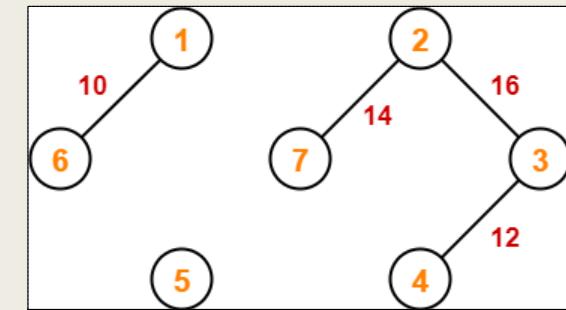
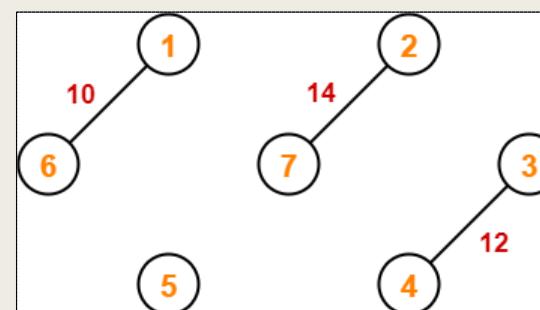
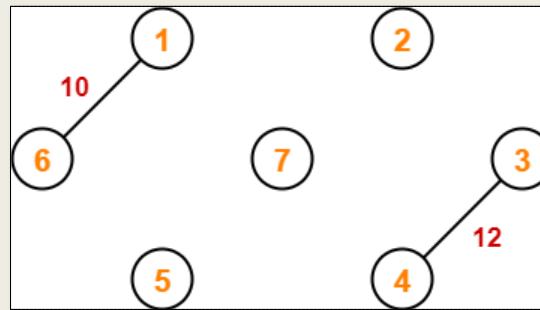
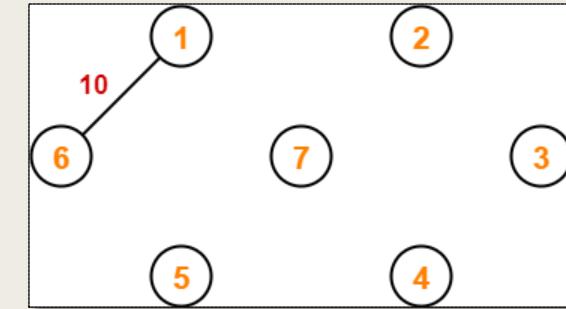
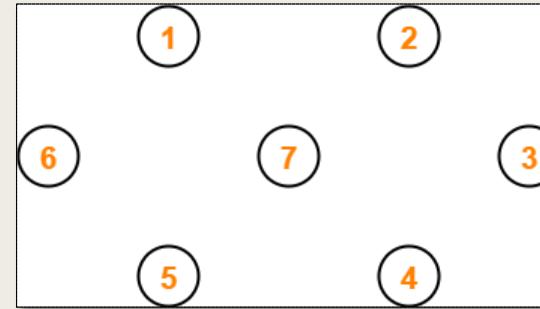
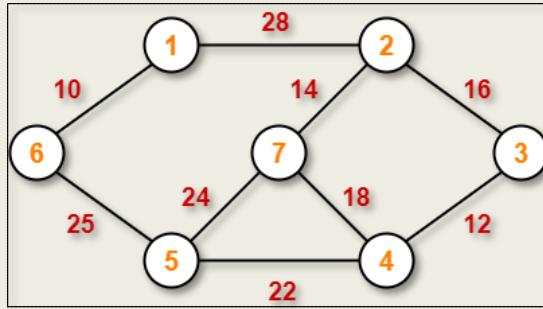
(b)

Kruskal's Algorithm

The implementation catch

- Consider the algorithm's operations as a progression through a series of forests containing all the vertices of a given graph and some of its edges.
- The initial forest consists of $|V|$ trivial trees, each comprising a single vertex of the graph.
- The final forest consists of a single tree, which is a minimum spanning tree of the graph.
- On each iteration, the algorithm takes
 - *the next edge (u, v) from the sorted list of the graph's edges,*
 - *finds the trees containing the vertices u and v , and,*
 - *if these trees are not the same, unites them in a larger tree by adding the edge (u, v) .*

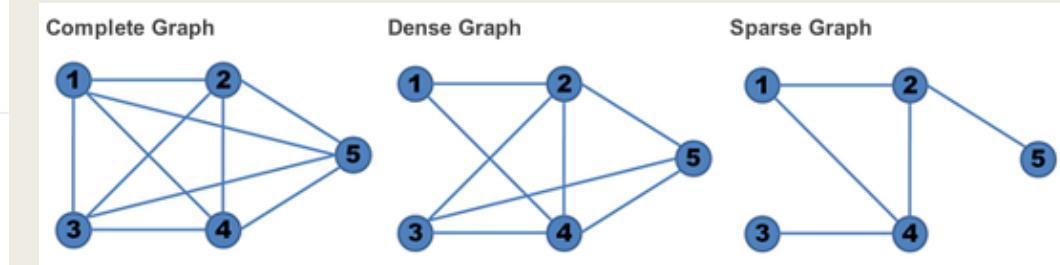
Kruskal's Algorithm



Differences between Prim's and Kruskal's

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 vertex carrying minimum weight in the graph.
It traverses one node more than one time to get the minimum distance.	It traverses one node only once.
Prim's algorithm has a time complexity of $O(V^2)$, V being the number of vertices.	Kruskal's algorithm's time complexity is $O(\log V)$, V being the number of vertices.
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.

A **graph** in which the number of edges is much less than the possible number of edges



A **graph** in which the number of edges is close to the possible number of edges.

Job Sequencing with deadlines

- The sequencing of jobs (for execution) on a single processor with deadline constraints is called as Job Sequencing with Deadlines.
- Problem statement

“

Given,

- **set of n jobs**
- **a deadline associated with each job i , $d_i \geq 0$**
- **profit associated with each job i , $p_i > 0$**
- For 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 job

Obtain the sequence of jobs that yields optimal solution (max profit) ”

Job Sequencing with deadlines



What is the feasible solution ?

- A **feasible solution** for this problem is a subset J of jobs such that each Job in this subset can be **completed by its deadline**
- The value of a **feasible solution J** is the sum of the profits of the jobs in J , or $\sum_{i \in J} p_i$
- An **optimal solution** is a feasible solution with **maximum value**.

Job Sequencing with deadlines

Brute force approach

- Solve the following instance of job sequencing problem

Let $n = 4$, $(p_1, p_2, p_3, p_4) = (100, 10, 15, 27)$, $(d_1, d_2, d_3, d_4) = (2, 1, 2, 1)$

	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

Job Sequencing with deadlines



Greedy Approach

- *Sort all the given jobs in decreasing order of their profit.*
- *Check the value of maximum deadline.*
- *Draw a Gantt chart where maximum time on Gantt chart is the value of maximum deadline.*
- *Select 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.*

Job Sequencing with deadlines

- Solve the following instance of job sequencing with deadline problem.

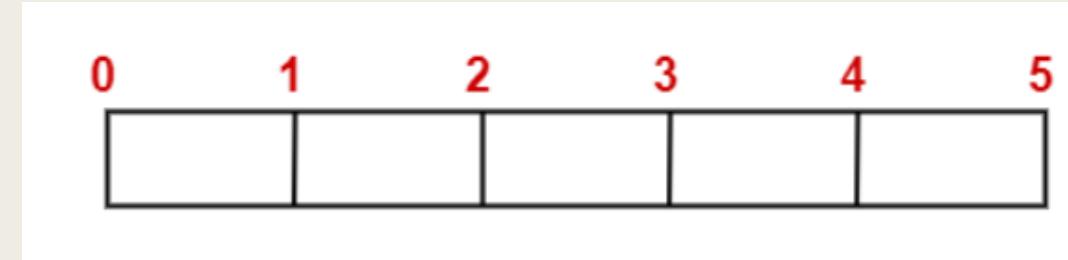
Jobs	J1	J2	J3	J4	J5	J6
Deadlines	5	3	3	2	4	2
Profits	200	180	190	300	120	100

- Solution
 - Sort all the given jobs in decreasing order of their profit.*

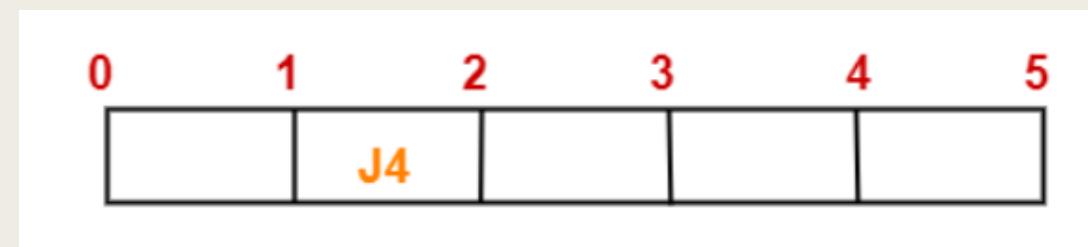
Jobs	J4	J1	J3	J2	J5	J6
Deadlines	2	5	3	3	4	2
Profits	300	200	190	180	120	100

Job Sequencing with deadlines

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

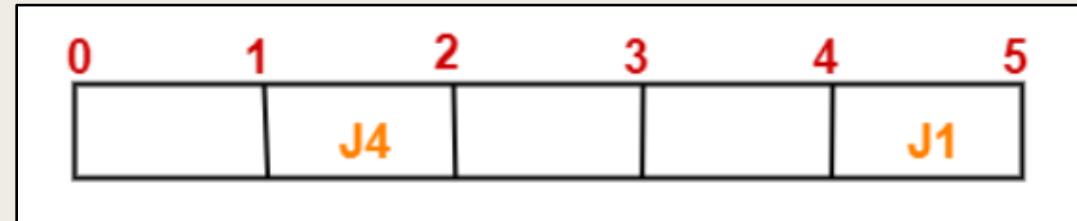


- Select the jobs as per the ordered profits and place them in the chart accordingly
- Select J4 ($d_4=2$)

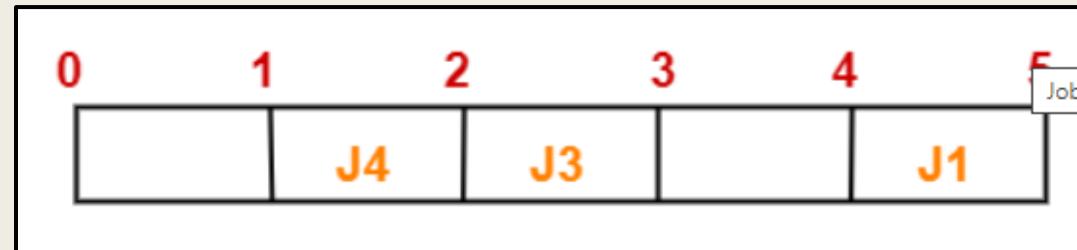


Job Sequencing with deadlines

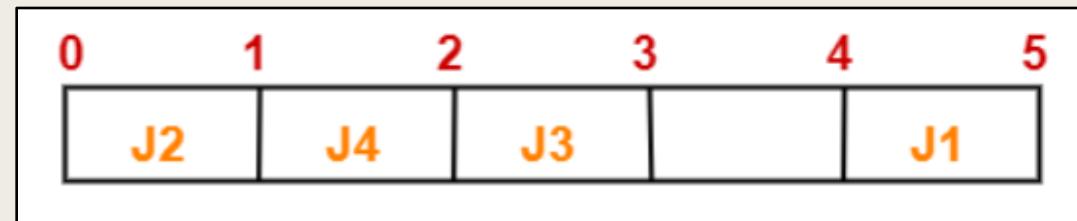
- Select J1 ($d_1=5$)



- Select J3 ($d_3=3$)

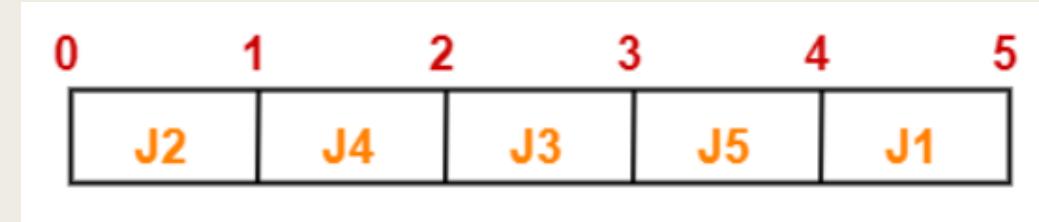


- Select J2 ($d_2=3$)



Job Sequencing with deadlines

- Select J5 (d5=4)



- The left over job is J6 whose deadline is 2.
- All the slots before deadline 2 are already occupied.
- Thus, job J6 can not be completed.
- Hence the optimal sequence of jobs are <J2, J4, J3, J5, J1>
- The profit earned by this sequence is 990 units

Job Sequencing with deadlines

- Solve the following instance of job sequencing problem

Let $n = 4$, $(p_1, p_2, p_3, p_4) = (100, 10, 15, 27)$, $(d_1, d_2, d_3, d_4) = (2, 1, 2, 1)$

Solution ?

Huffman coding

Situation

- How will you compactly store a data file of 100000 characters?
- The file consists of only 6 different characters
- Their occurrences (frequencies) are as follows

	a	b	c	d	e	f
Frequency (in thousands)	45	13	12	16	9	5

- Designing a **binary character code** (in which each character is represented by a unique binary string, which we call a **Codeword**) is one among multiple solutions

Huffman coding

	a	b	c	d	e	f
Frequency (in thousands)	45	13	12	16	9	5
Fixed-length codeword	000	001	010	011	100	101
Variable-length codeword	0	101	100	111	1101	1100

■ Fixed length code (e.g. ASCII)

- 3 bits to represent 6 characters
- **a = 000, b = 001, . . . , f = 101.**
- This method requires 300,000 bits to code the entire file
- **Can you do better ?**

Huffman coding

	a	b	c	d	e	f
Frequency (in thousands)	45	13	12	16	9	5
Fixed-length codeword	000	001	010	011	100	101
Variable-length codeword	0	101	100	111	1101	1100

■ Variable length code (Morse code)

- assign frequent characters short codewords and infrequent characters long codewords
- This code requires $(45.1+13.3+12.3+16.3+9.4+5.4).1,000 = 224,000$ bits
- When compared to fixed length code, the variable length code saved the storage by 25%

Huffman coding

Prefix codes

- A code is called a prefix (free) code if no codeword is a prefix of another one. (typically of variable length)
- Example

A: 00

B: 010

C: 011

D: 10

E: 11

↑
Prefix code

A: 00

B: 010

C: 001 # 011 -> 001

D: 10

E: 11

↑
Not a Prefix code

{3, 11, 22} ← Prefix code

{1, 12, 13} ← Not a Prefix code

Huffman coding

Prefix codes

- Life is easy with prefix code
- Let [1, 2, 33, 34, 50, 61] be the codewords
- Let the sequence of number received be 1611333425012
- Decoding ?

1 61 1 33 34 2 50 1 2

- <https://gist.github.com/joepie91/26579e2f73ad903144dd5d75e2f03d83>
- <https://leimao.github.io/blog/Huffman-Coding/>

Huffman coding

David Huffman invented a greedy algorithm to

“Construct a tree that would assign shorter bit strings to high-frequency symbols and longer ones to low-frequency symbols”

- He invented this when he was a graduate student at MIT
- The two major steps in Huffman algorithm are
 - Building a Huffman Tree from the input characters.
 - Assigning code to the characters by traversing the Huffman Tree.

Huffman coding

1. Create a leaf node for each character of the text.
2. Arrange all the nodes in increasing order of their frequency value.
3. Considering the first two nodes having minimum frequency,
 - a. Create a new internal node.
 - b. The frequency of this new node is the sum of frequency of those two nodes.
 - c. Make the first node as a left child and the other node as a right child of the newly created node.
4. Keep repeating Step-2 and Step-3 until all the nodes form a single tree.

<https://www.gatevidyalay.com/huffman-coding-huffman-encoding/>

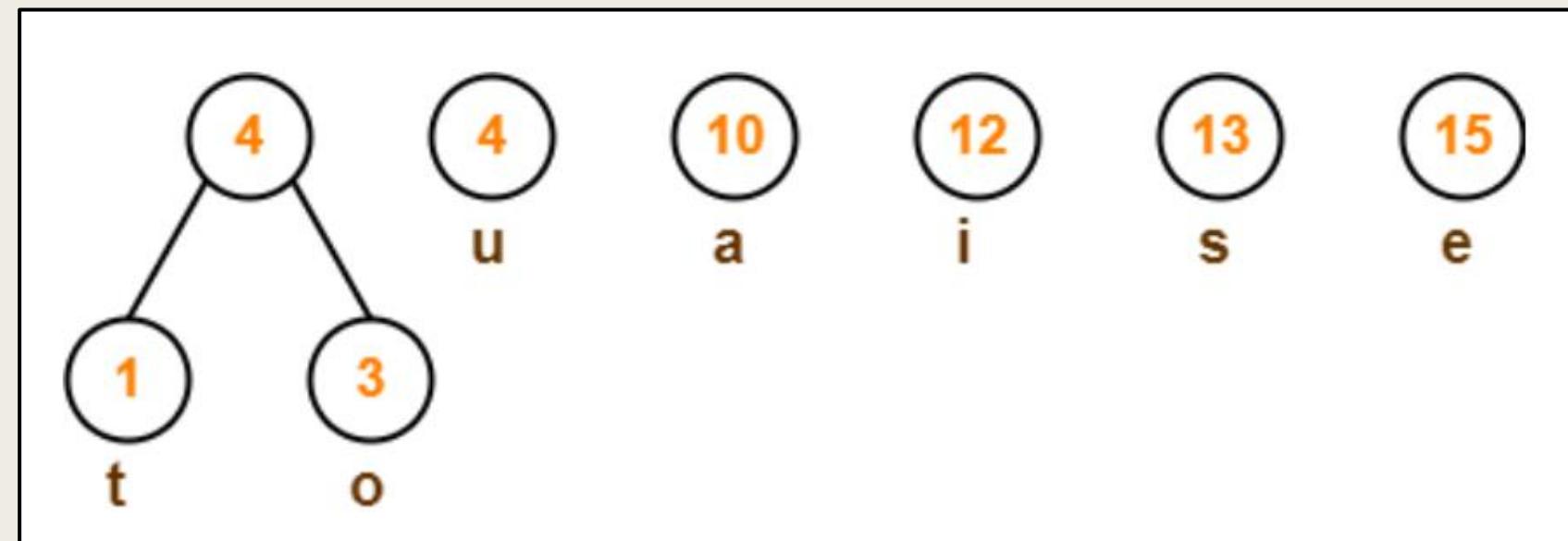
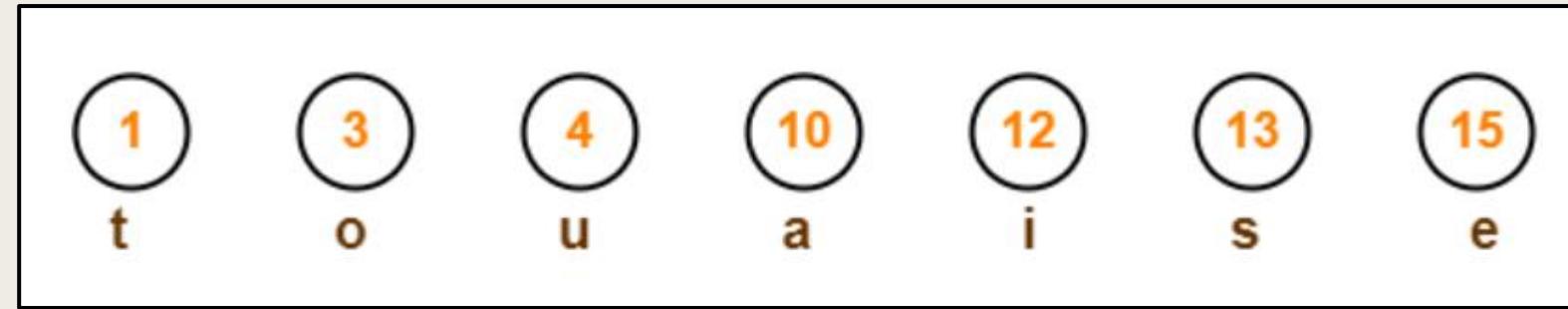
Huffman coding - Problem

- A file contains the following characters with the frequencies as shown. If Huffman Coding is used for data compression, determine-
 - Huffman Code for each character
 - Average code length
 - Length of Huffman encoded message (in bits)

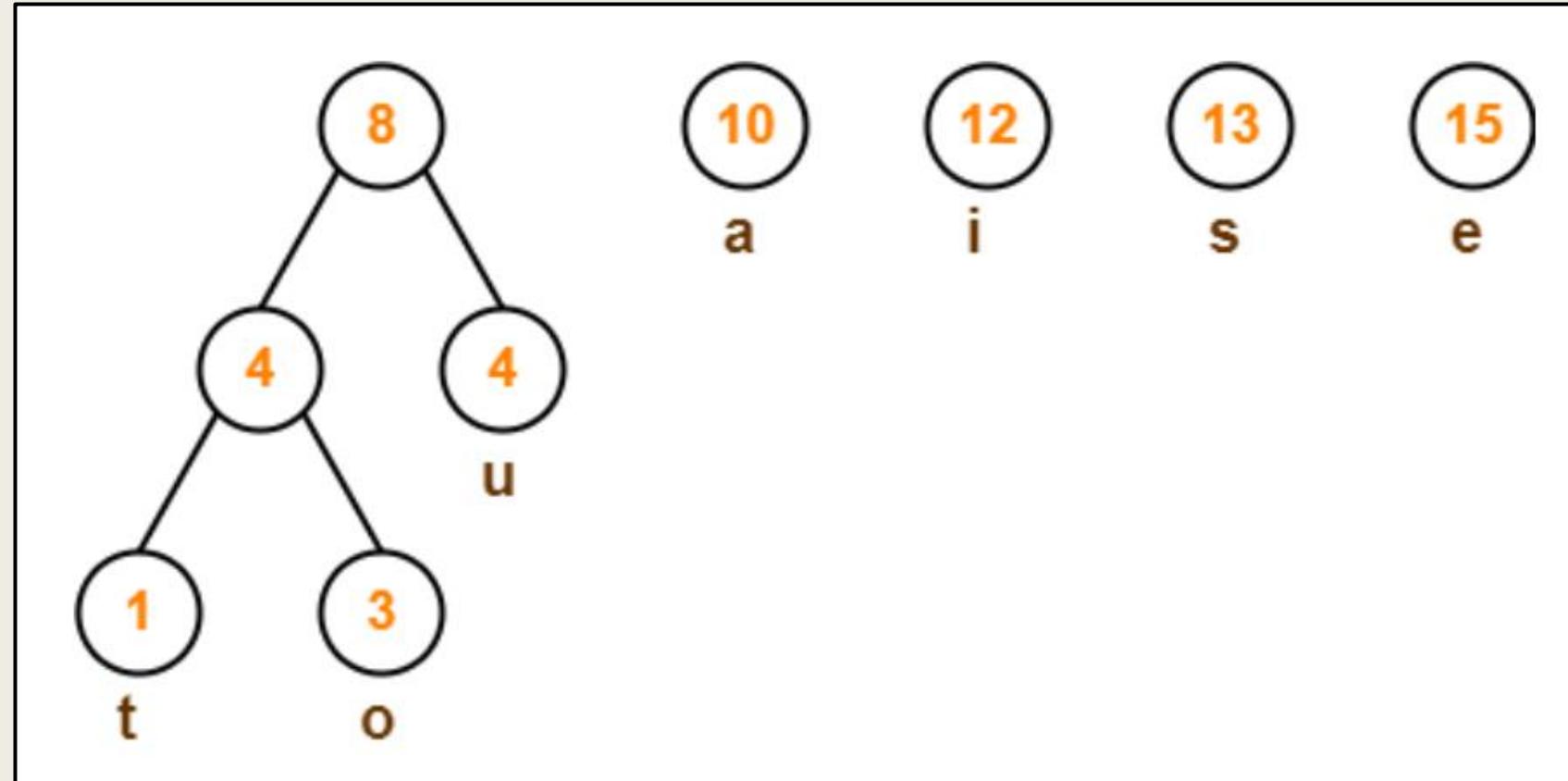
Characters	Frequencies
a	10
e	15
i	12
o	3
u	4
s	13
t	1

Huffman coding

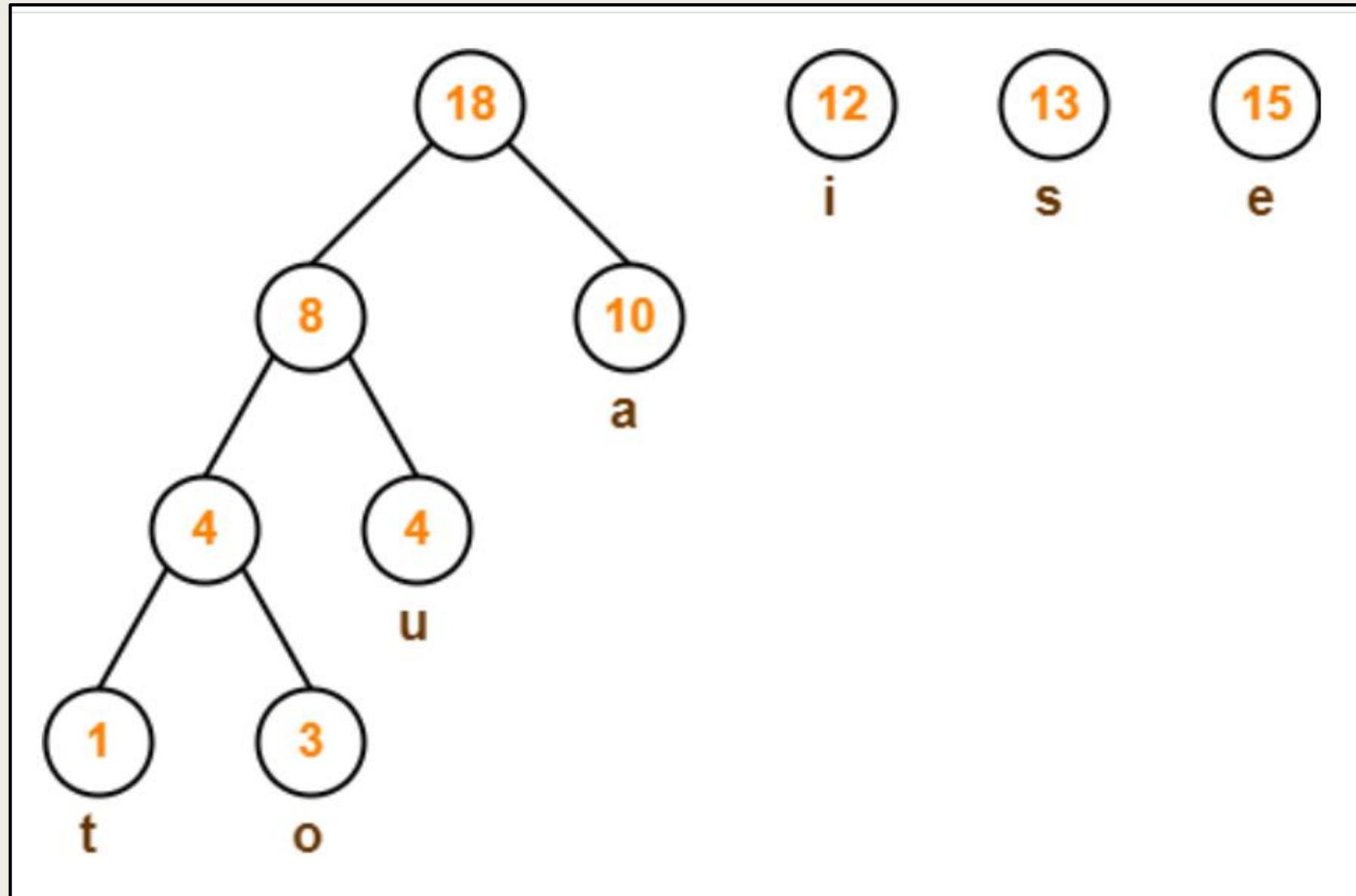
Construction of Huffman tree



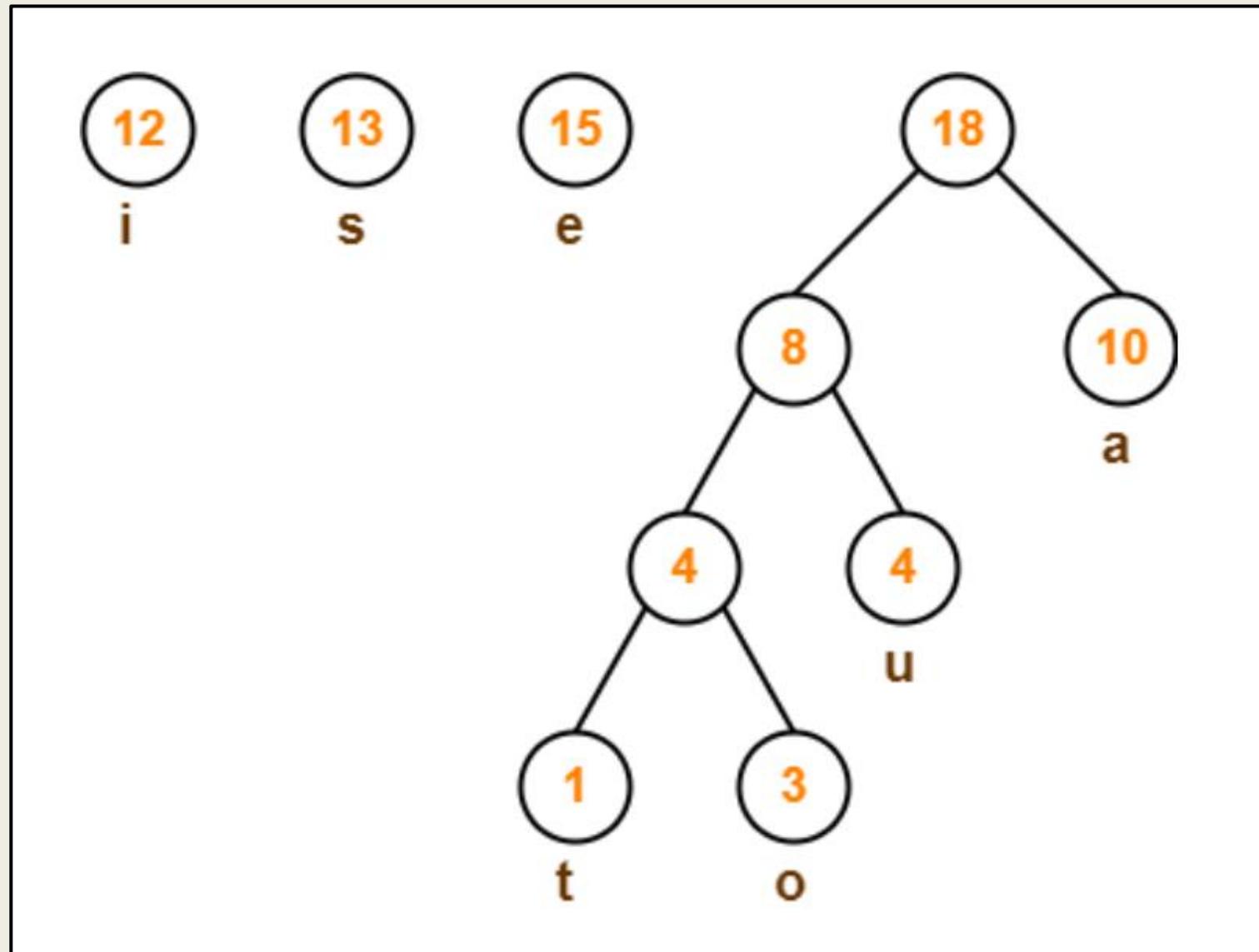
Huffman coding



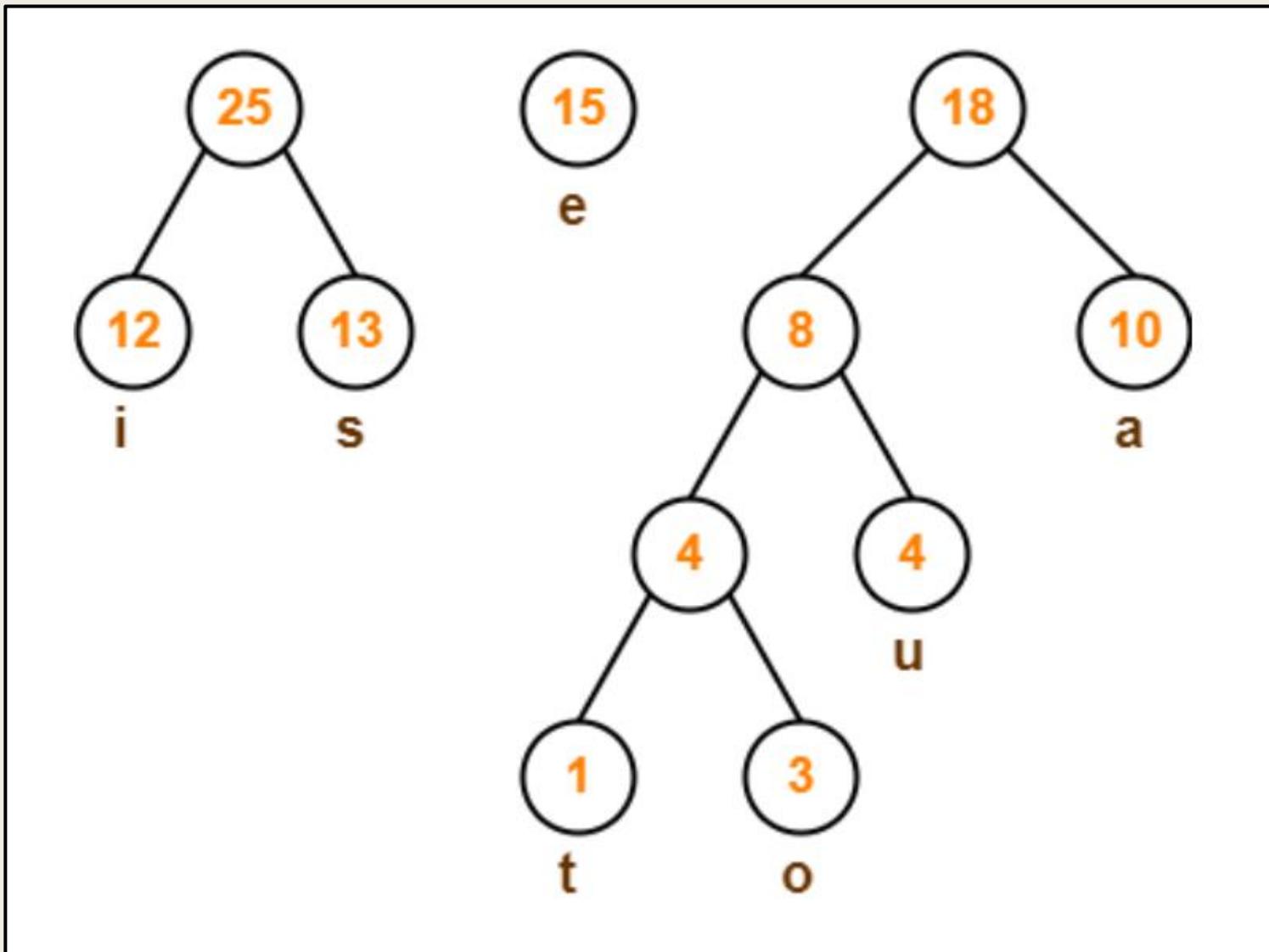
Huffman coding



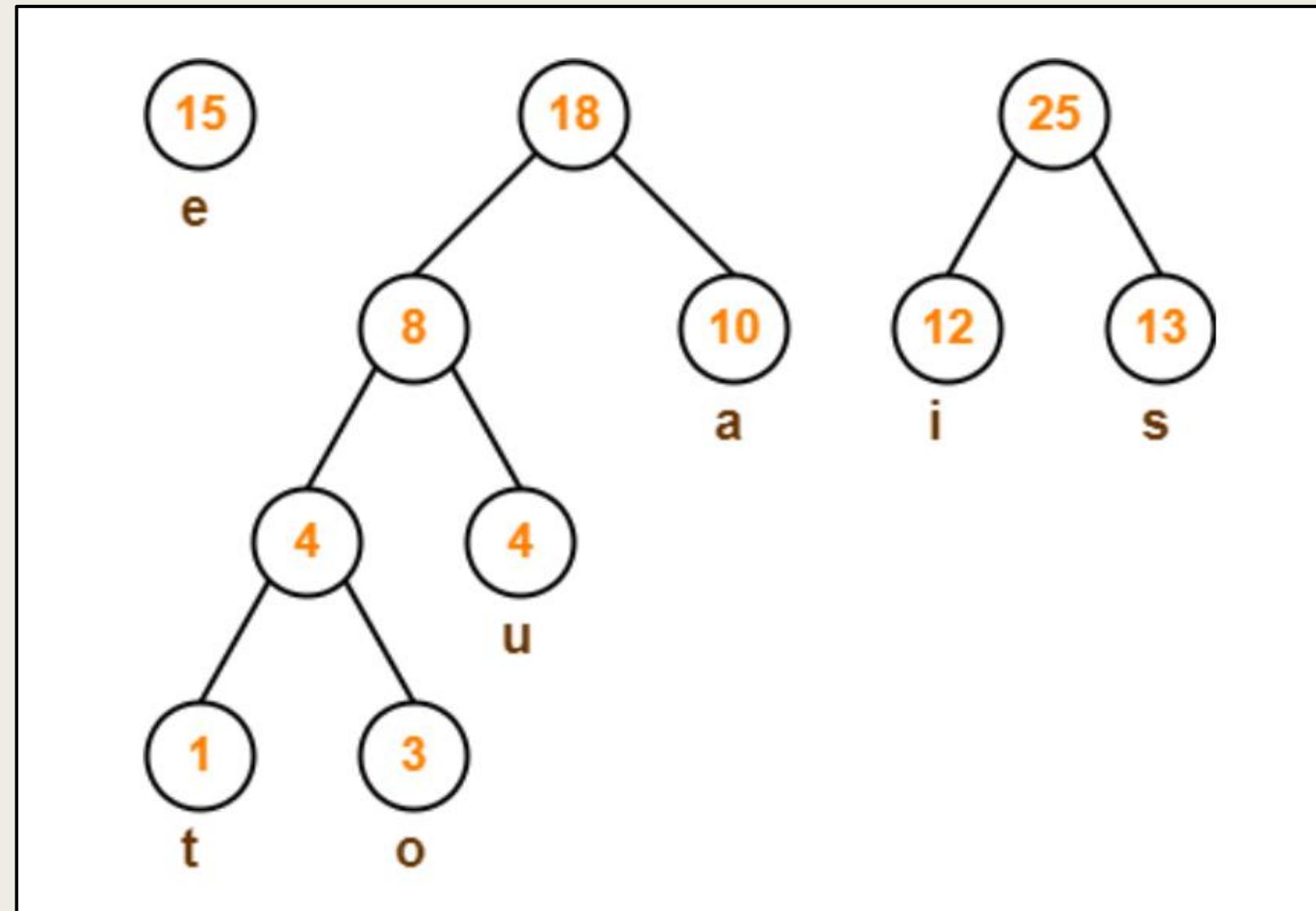
Huffman coding



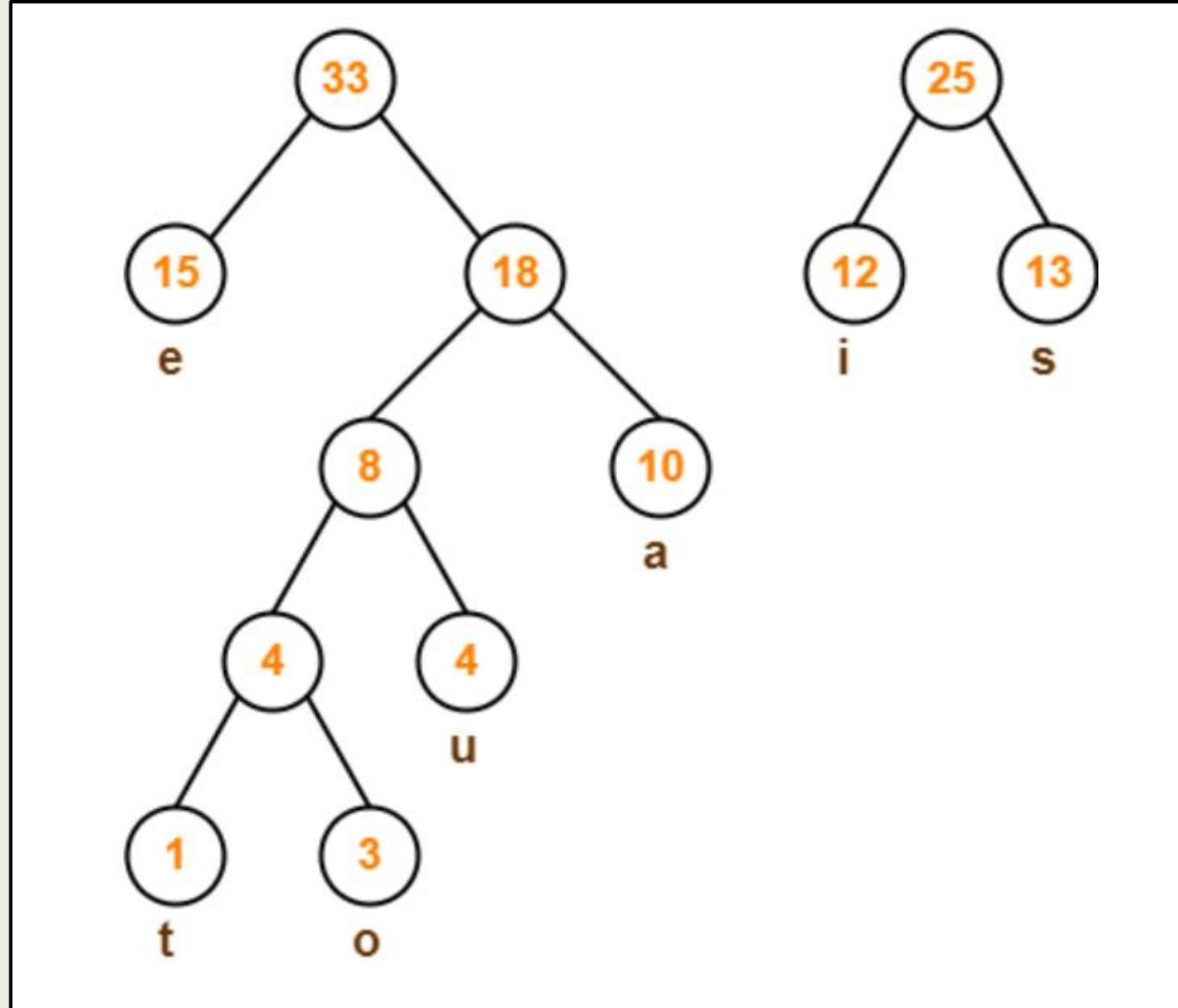
Huffman coding



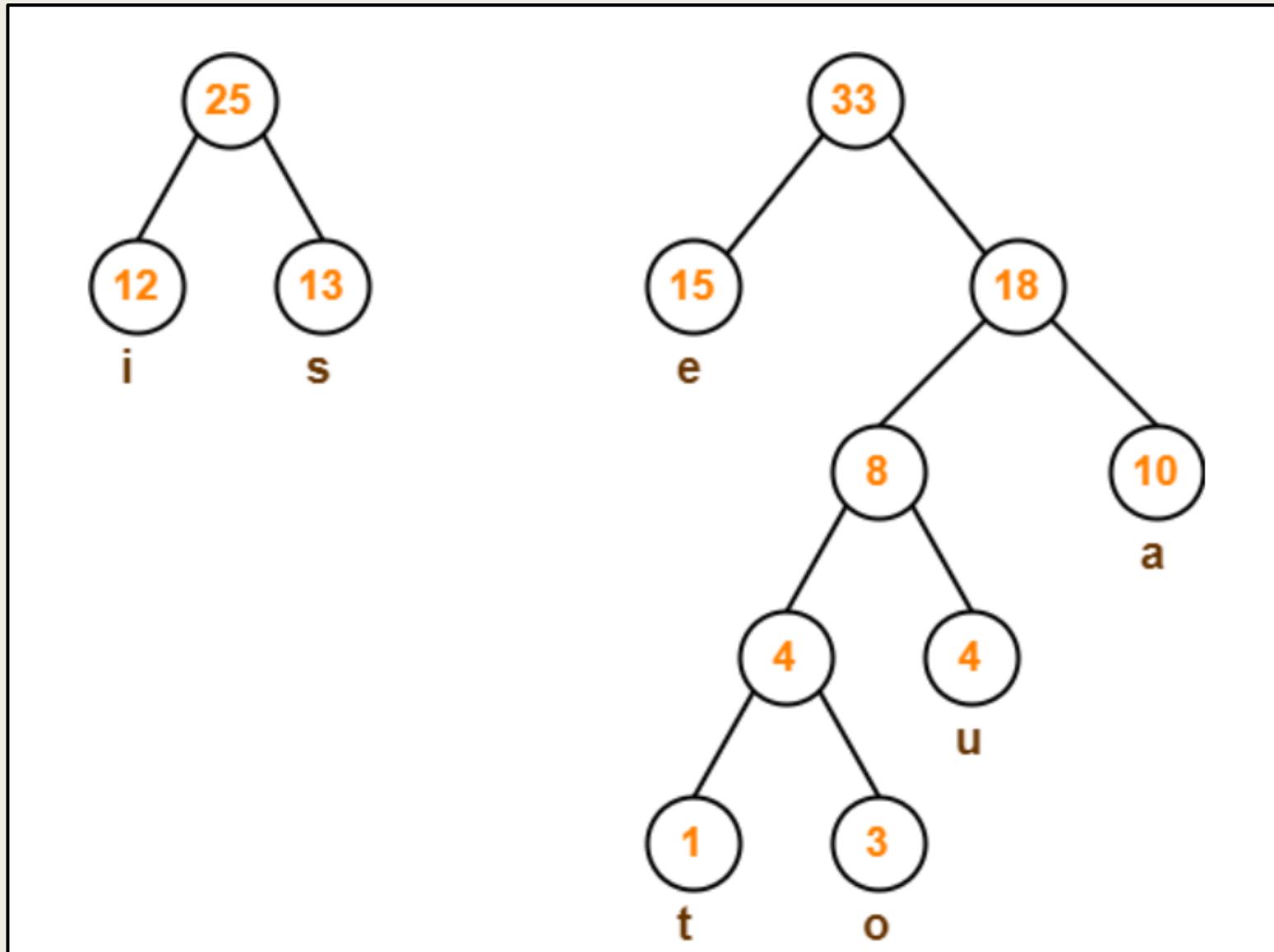
Huffman coding



Huffman coding

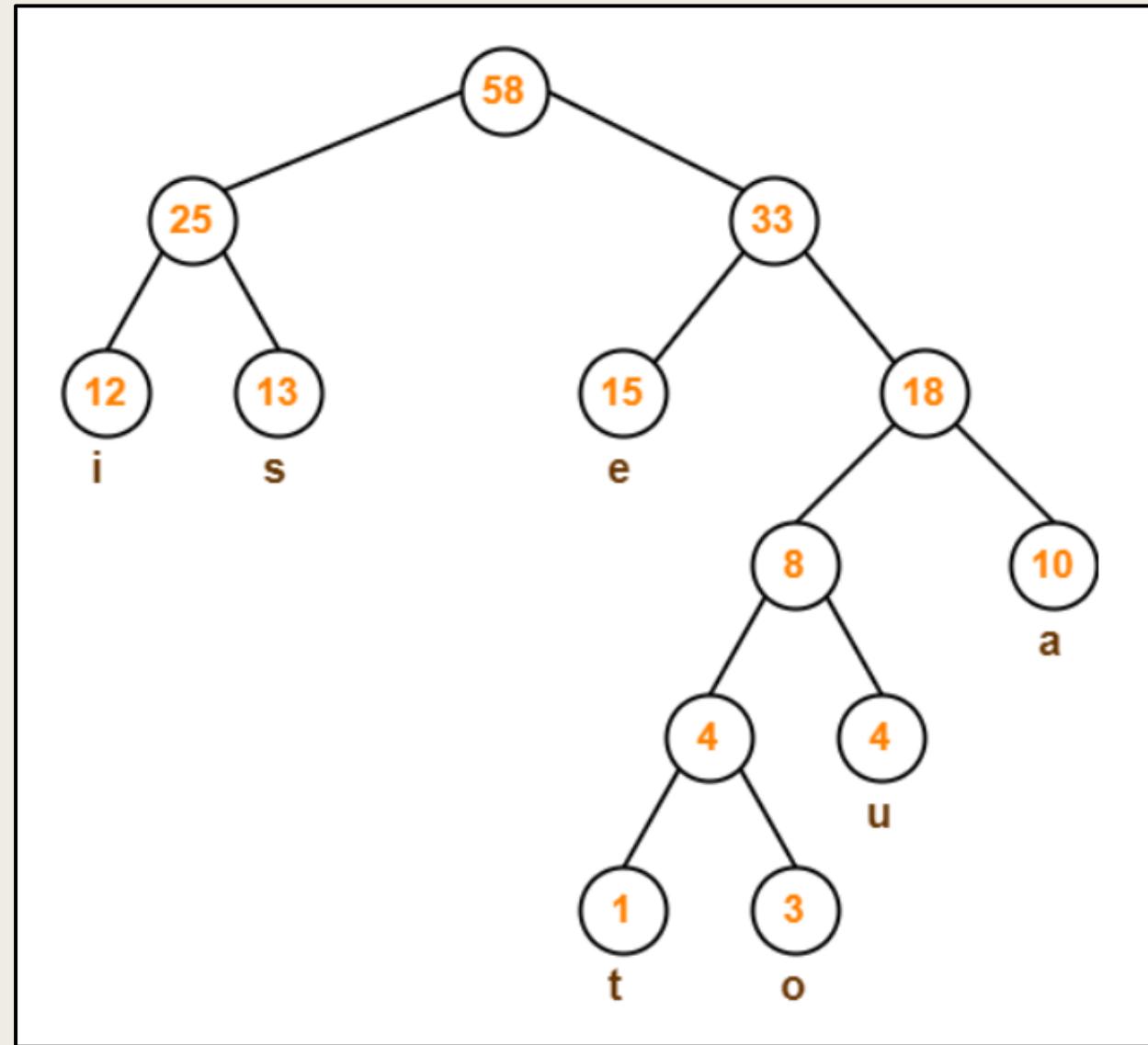


Huffman coding



Huffman coding

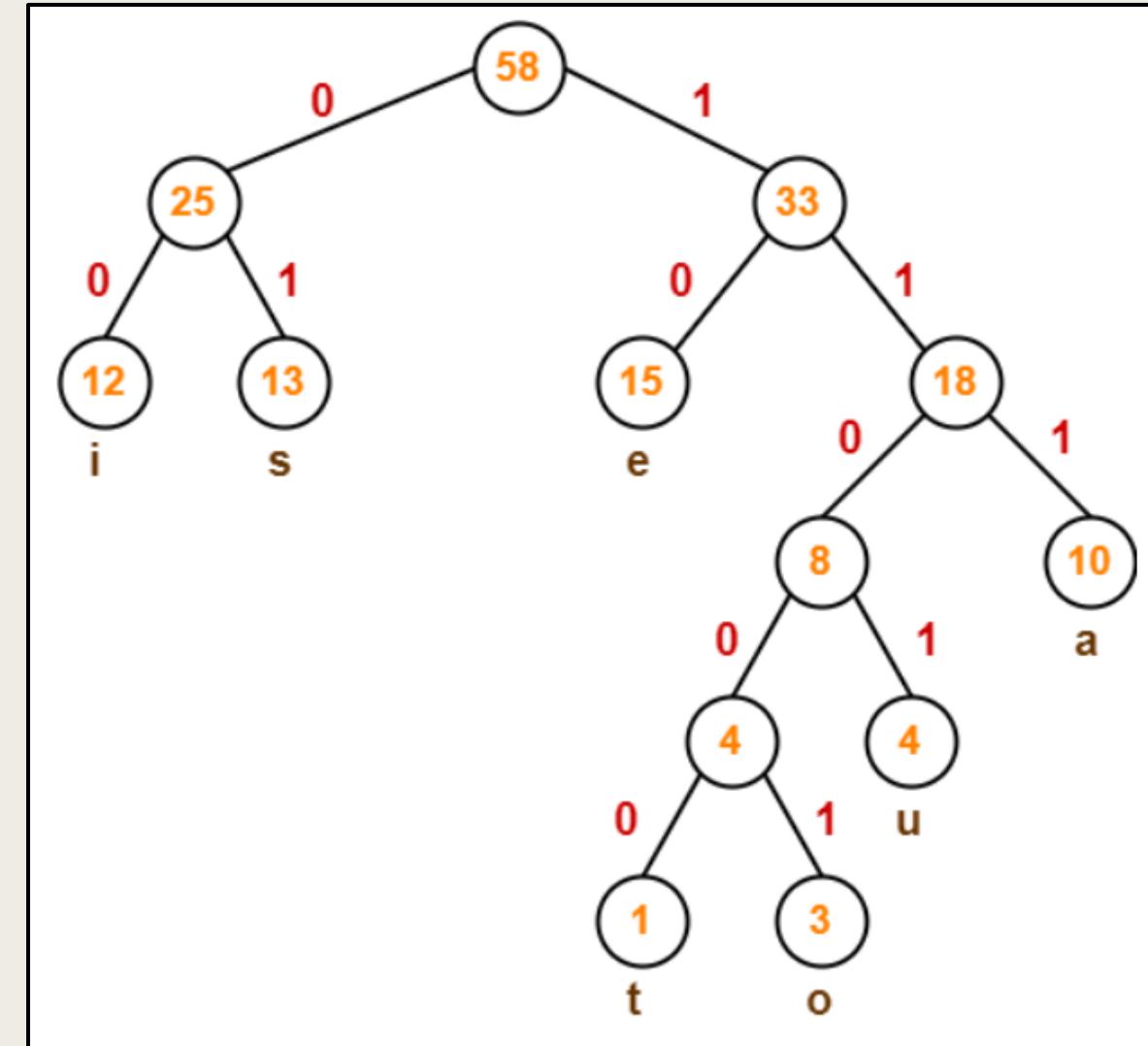
Huffman Tree →



Huffman coding

Assign weight '0' to the left edges and weight '1' to the right edges of the Huffman Tree

a = 111
e = 10
i = 00
o = 11001
u = 1101
s = 01
t = 11000



Huffman coding

- Construct the Huffman tree and hence obtain the codewords for the following population

Value	A	B	C	D	E	F
Frequency	5	25	7	15	4	12

Single source shortest path



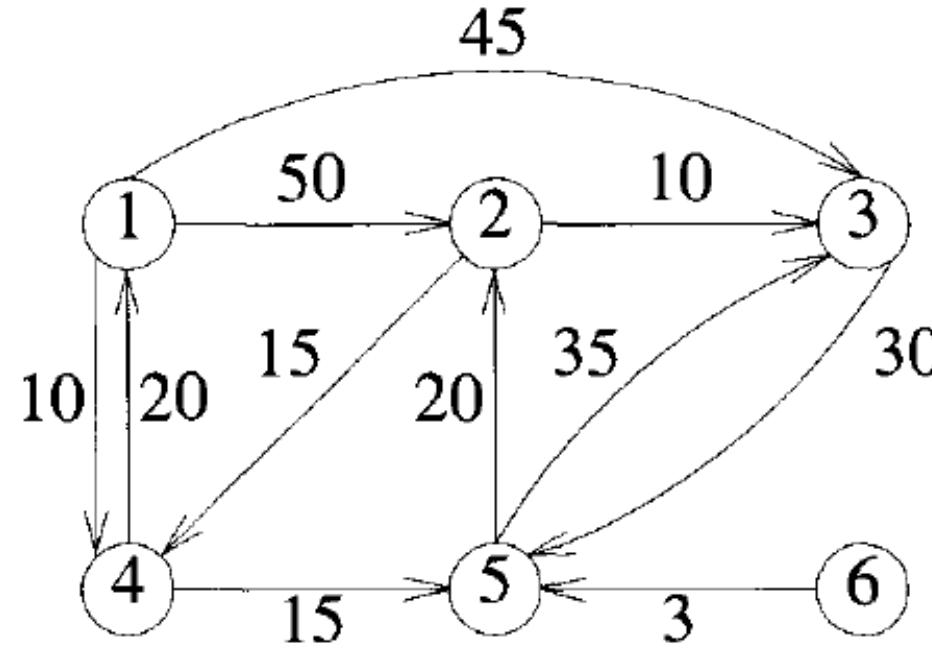
- Graphs can be used to model highway structure of a state/country.
- Vertices representing cities and edges representing the connecting roads
- The edges can be assigned weights which might represent distance/cost/time to drive across the cities
- If Mr. X wants to drive from point A to point B
- Following are the questions raised
- Is there a path from A to B?
- If there are multiple paths then which is the shortest path?

Single source shortest path



- Graphs can be used to model highway structure of a state/country.
- Vertices representing cities and edges representing the connecting roads
- The edges can be assigned weights which might represent distance/cost/time to drive across the cities
- If Mr. X wants to drive from point A to point B
- Following are the questions raised
 - *Is there a path from A to B?*
 - *If there are multiple paths then which is the shortest path?*

Single source shortest path



(a) Graph

<i>Path</i>	<i>Length</i>
1) 1, 4	10
2) 1, 4, 5	25
3) 1, 4, 5, 2	45
4) 1, 3	45

(b) Shortest paths from 1

Single source shortest path

- The problem statement
- Given a weighted connected graph $G=(V, E)$ and a source vertex s , determine the shortest paths from s to all the remaining vertices.
- The single-source shortest-paths problem asks for a family of paths, each leading from the source to a different vertex in the graph, though some paths may, of course, have edges in common.
- Major applications
 - *Transportation planning*
 - *Packet routing in Internet*
 - *Social networks*
 - *Speech recognition*
 - *Path finding video games*

Single source shortest path

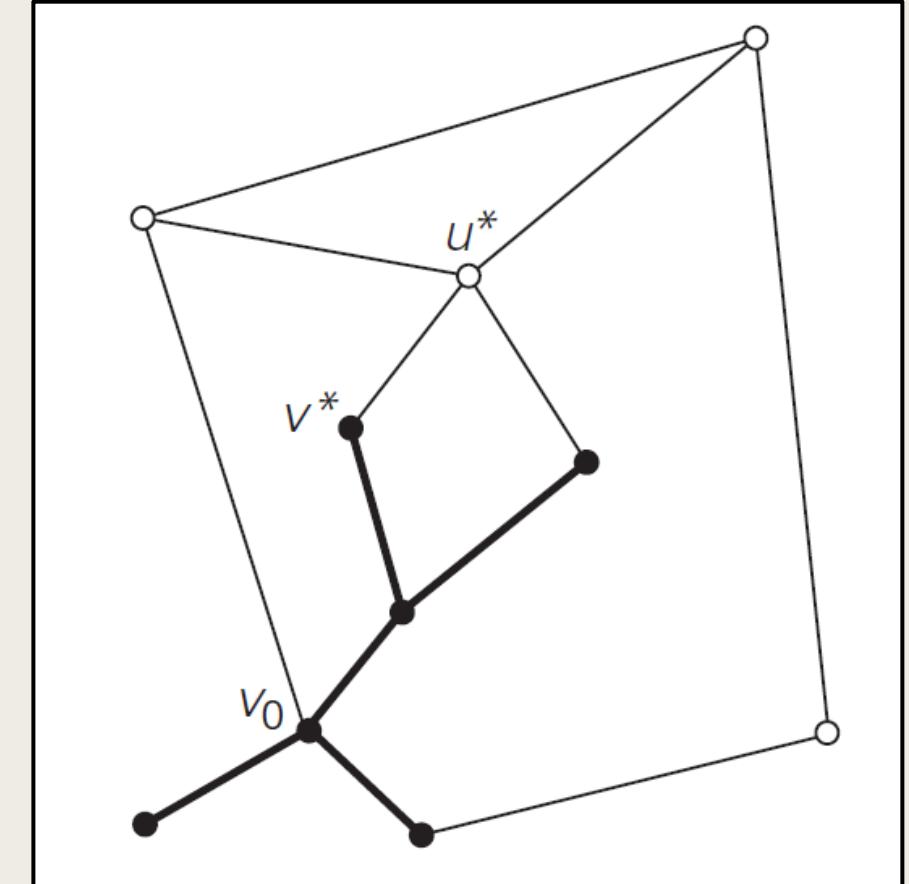


Dijkstra's Algorithm

- **Best-known algorithm for the single-source shortest-paths problem**
- **This algorithm is applicable to undirected and directed graphs with nonnegative weights only**
- Dijkstra's algorithm finds the shortest paths to a graph's vertices in order of their distance from a given source
- First, it finds the shortest path from the source to a vertex nearest to it then to a second nearest, and so on
- In general, before its i th iteration commences, the algorithm has already identified the shortest paths to $i - 1$ other vertices nearest to the source.

Single source shortest path

- Working of Dijkstra's algorithm
- Fringe vertices?
 - *The set of vertices adjacent to the vertices in T_i*
- How to identify next nearest vertex? u^*



Single source shortest path

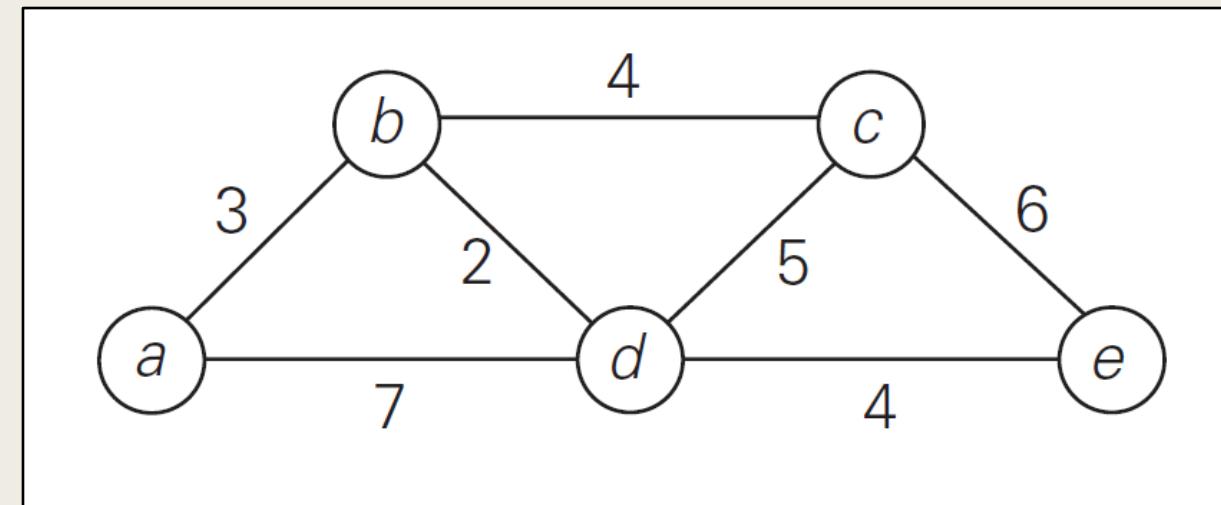
- After identifying do the following
- Move u^* from the fringe to the set of tree vertices.
- For each remaining fringe vertex u that is connected to u^* by an edge of weight $w(u^*, u)$ such that $du^* + w(u^*, u) < du$, update the labels of u by u^* and $du^* + w(u^*, u)$, respectively.

Single source shortest path

- Find the shortest paths from the source vertex **a** to all other vertices

- Adjacency Matrix

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

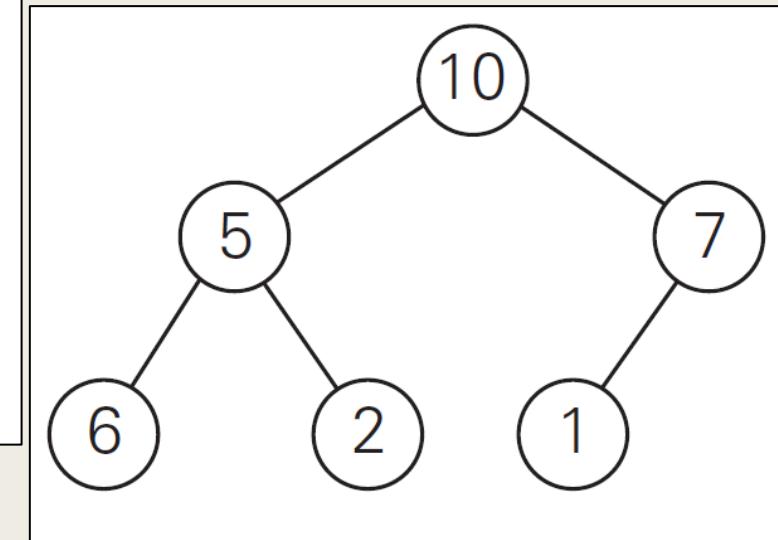
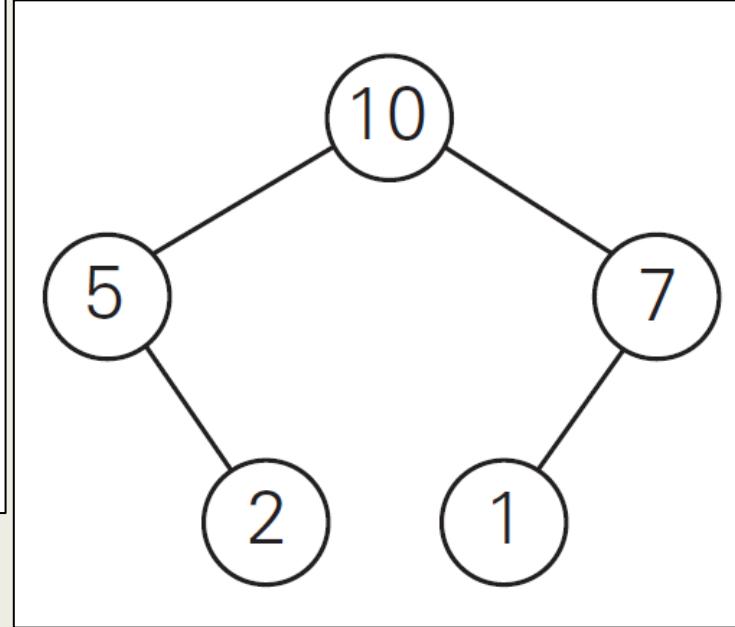
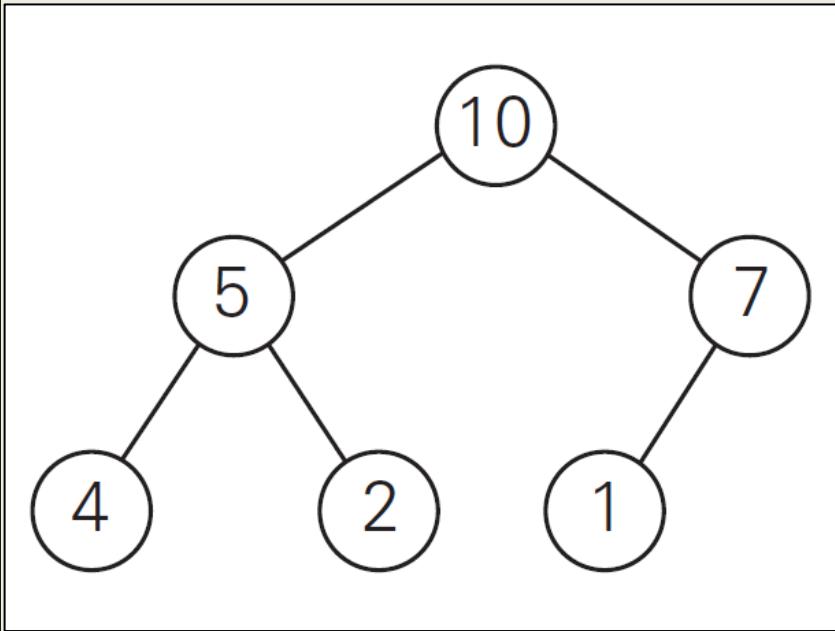


[Solution](#)

Heaps and Heap Sort

- Partially ordered data structure that is especially suitable for implementing priority queues
- A **heap** can be defined as a binary tree with keys assigned to its nodes, one key per node, provided these two conditions are met:
 - **The shape property**
 - The binary tree is essentially complete (or simply complete), i.e., all its levels are full except possibly the last level, where only some rightmost leaves may be missing.
 - **The parental dominance or heap property**
 - The key in each node is greater than or equal to the keys in its children. (This condition is considered automatically satisfied for all leaves.)

Heaps and Heap Sort



- **Note**
 - key values in a heap are ordered top down
 - there is no left-to-right order in key values;

Heaps and Heap Sort

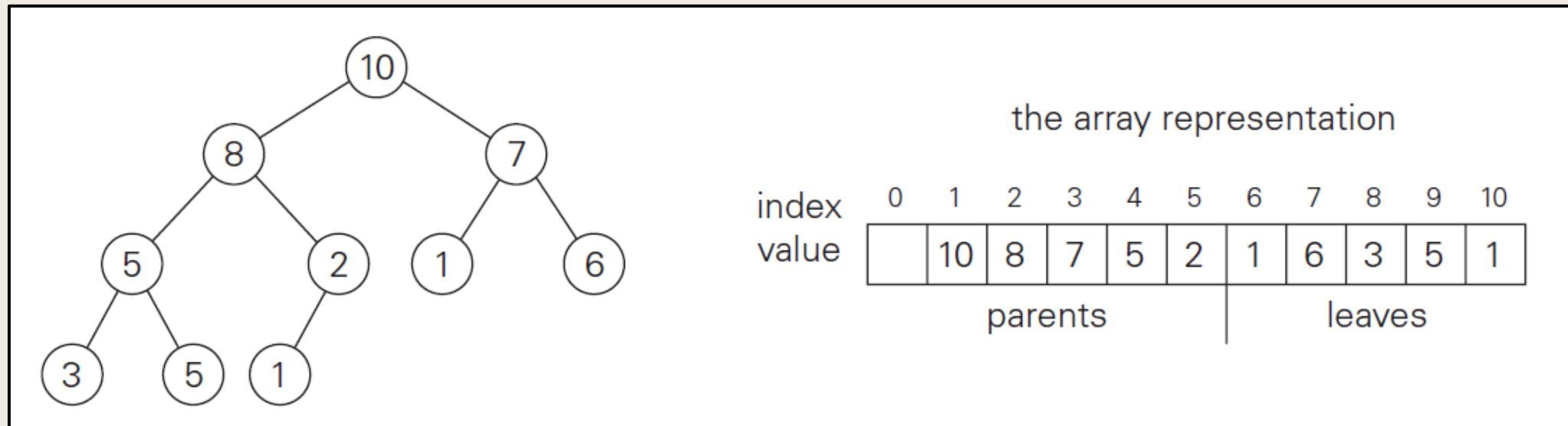
Properties of Heap

- There exists exactly one essentially complete binary tree with n nodes. Its height is equal to $\log_2 n$.
- The **root** of a heap always contains its **largest** element.
- A node of a heap considered with all its descendants is also a heap.
- A heap can be implemented as an **array** by recording its elements in the top down, left-to-right fashion
- It is convenient to store the heap's elements in positions 1 through n of such an array, leaving **H[0]** either unused

Heaps and Heap Sort

■ In an array representation of heaps

- the parental node keys will be in the first $[n/2]$ positions of the array, while the leaf keys will occupy the last $[n/2]$ positions;
- the children of a key in the array's parental position $i (1 \leq i \leq n/2)$ will be in positions $2i$ and $2i + 1$, and, correspondingly, the parent of a key in position $i (2 \leq i \leq n)$ will be in position $[i/2]$



Heaps and Heap Sort

How to construct a heap for a given list of keys?

■ **bottom-up heap construction**

- It initializes the essentially complete binary tree with n nodes by placing keys in the order given
- then “heapifies” the tree

■ **top-down heap construction**

- Successive insertions of a new key into a previously constructed heap

Bottom-up heap construction

Example : construct a heap for the following list of numbers

2, 9, 7, 6, 5, 8.

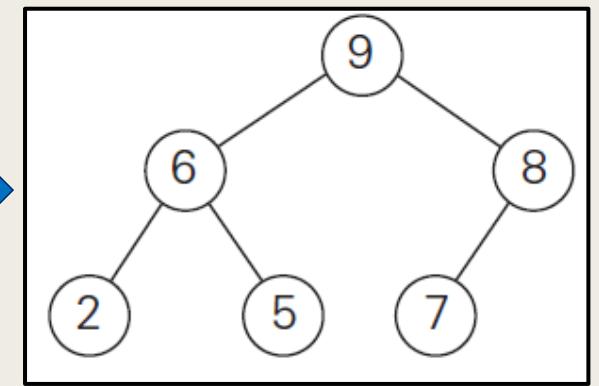
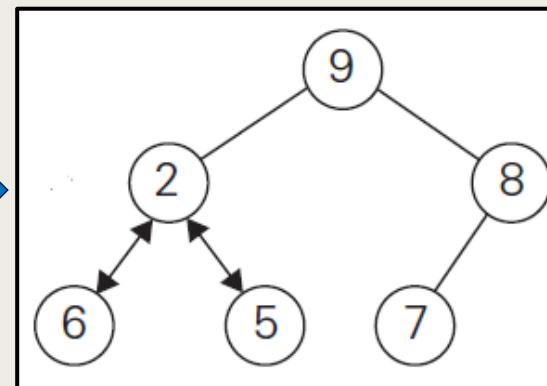
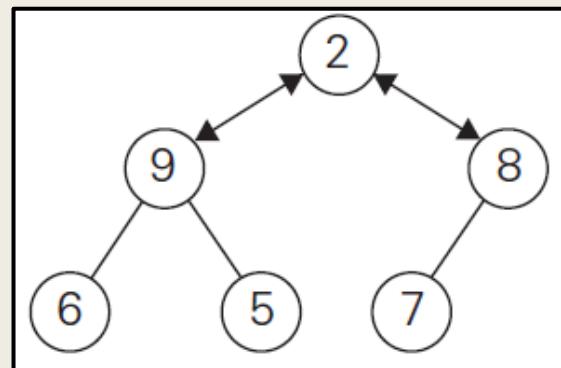
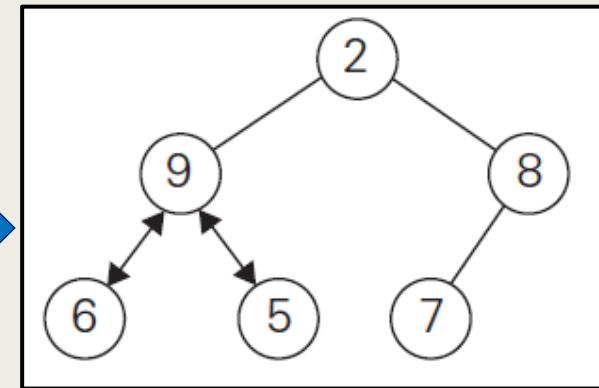
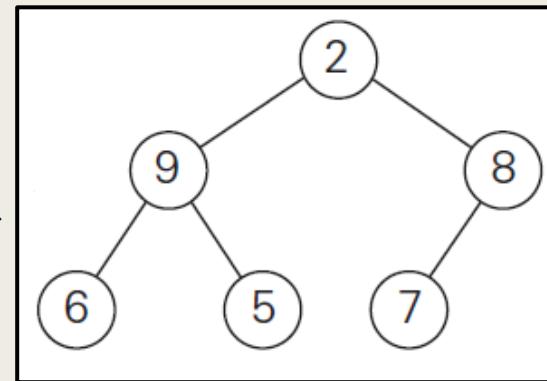
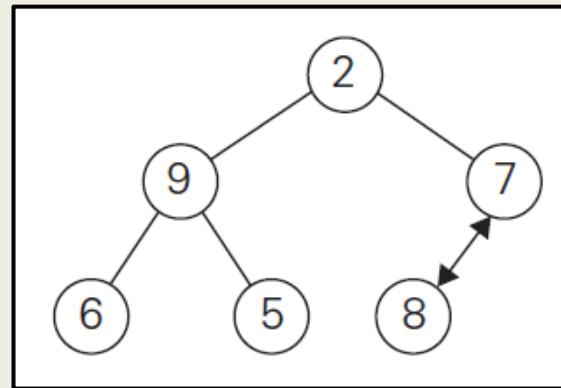
Heaps and Heap Sort

Bottom-up heap construction

- It initializes the essentially complete binary tree with n nodes by placing keys in the order given
- **Heapify**
 - Starting with the last parental node, the algorithm checks whether the parental dominance holds for the key in this node
 - If it does not, the algorithm exchanges the node's key K with the larger key of its children and checks whether the parental dominance holds for K in its new position
 - This process continues until the parental dominance for K is satisfied.
 - After completing the “**heapification**” of the subtree rooted at the current parental node, the algorithm proceeds to do the same for the node's immediate predecessor
 - The algorithm stops after this is done for the root of the tree.

Heaps and Heap Sort

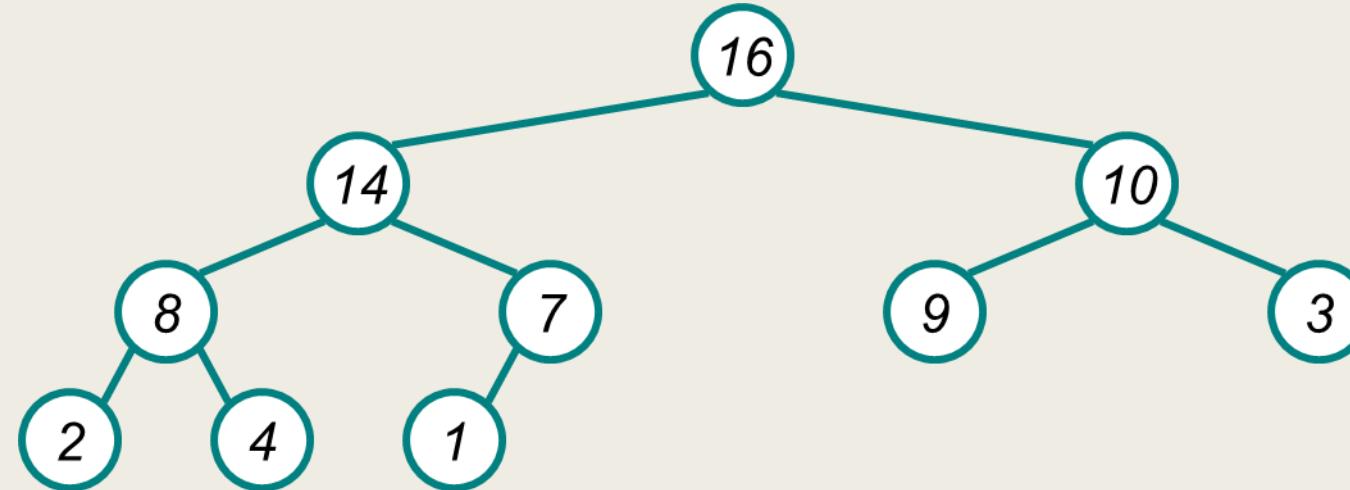
Example : Construct a heap for the following list of numbers
2, 9, 7, 6, 5, 8.



Heaps and Heap Sort

- construct a heap for the following list of numbers

4, 1, 3, 2, 16, 9, 10, 14, 8, 7



Heaps and Heap Sort

ALGORITHM *HeapBottomUp($H[1..n]$)*

```
//Constructs a heap from elements of a given array
// by the bottom-up algorithm
//Input: An array  $H[1..n]$  of orderable items
//Output: A heap  $H[1..n]$ 
for  $i \leftarrow \lfloor n/2 \rfloor$  downto 1 do
     $k \leftarrow i$ ;  $v \leftarrow H[k]$ 
     $heap \leftarrow \text{false}$ 
    while not  $heap$  and  $2 * k \leq n$  do
         $j \leftarrow 2 * k$ 
        if  $j < n$  //there are two children
            if  $H[j] < H[j + 1]$   $j \leftarrow j + 1$ 
        if  $v \geq H[j]$ 
             $heap \leftarrow \text{true}$ 
        else  $H[k] \leftarrow H[j]$ ;  $k \leftarrow j$ 
     $H[k] \leftarrow v$ 
```

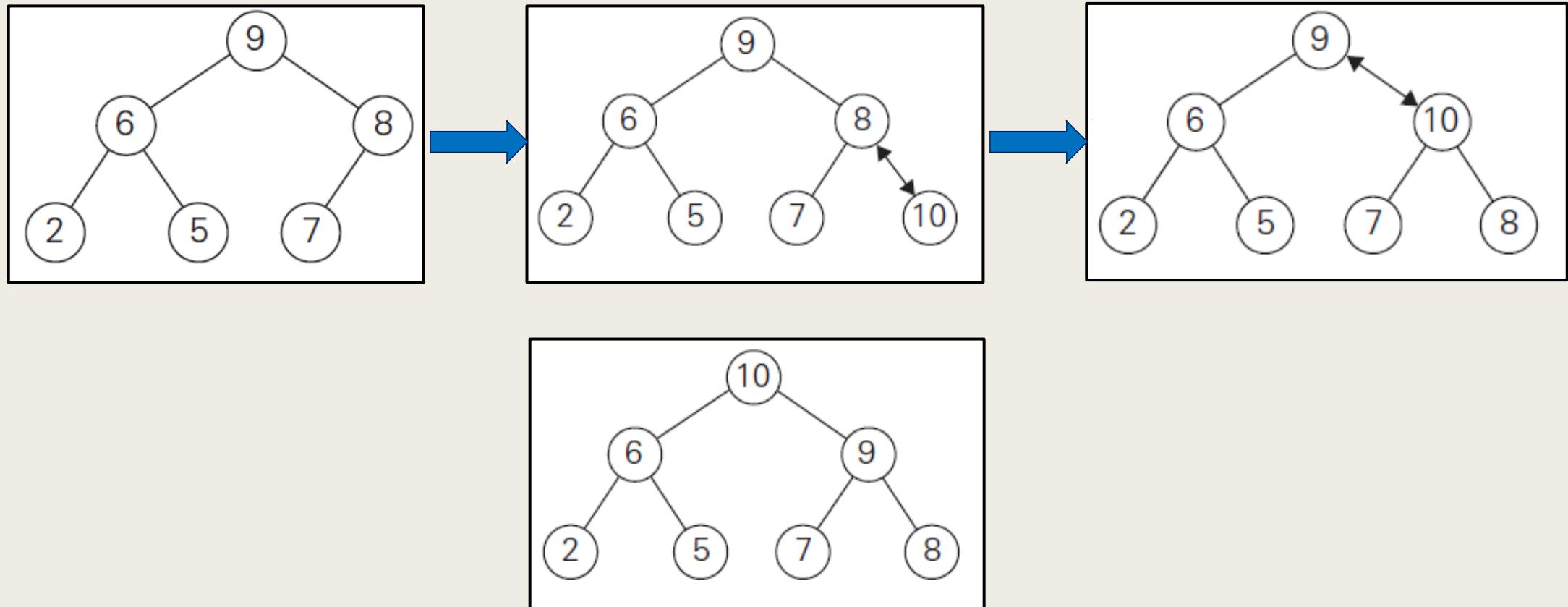
Heaps and Heap Sort

- **Top-down heap construction**
 - Successive insertions of a new key into a previously constructed heap
- **How to insert a new key K into a heap?**
 - First, attach a new node with key K in it after the last leaf of the existing heap.
 - Then shift K up to its appropriate place in the new heap as follows.
 - Compare K with its parent's key: if the latter is greater than or equal to K , stop (the structure is a heap);
 - Otherwise, swap these two keys and compare K with its new parent.
 - This swapping continues until K is not greater than its last parent or it reaches the root

Heaps and Heap Sort

■ top-down heap construction

– Successive insertions of a new key into a previously constructed heap



Heaps and Heap Sort

- How to delete an item form the heap?
- Special case- **Deletion of root**

Maximum Key Deletion from a heap

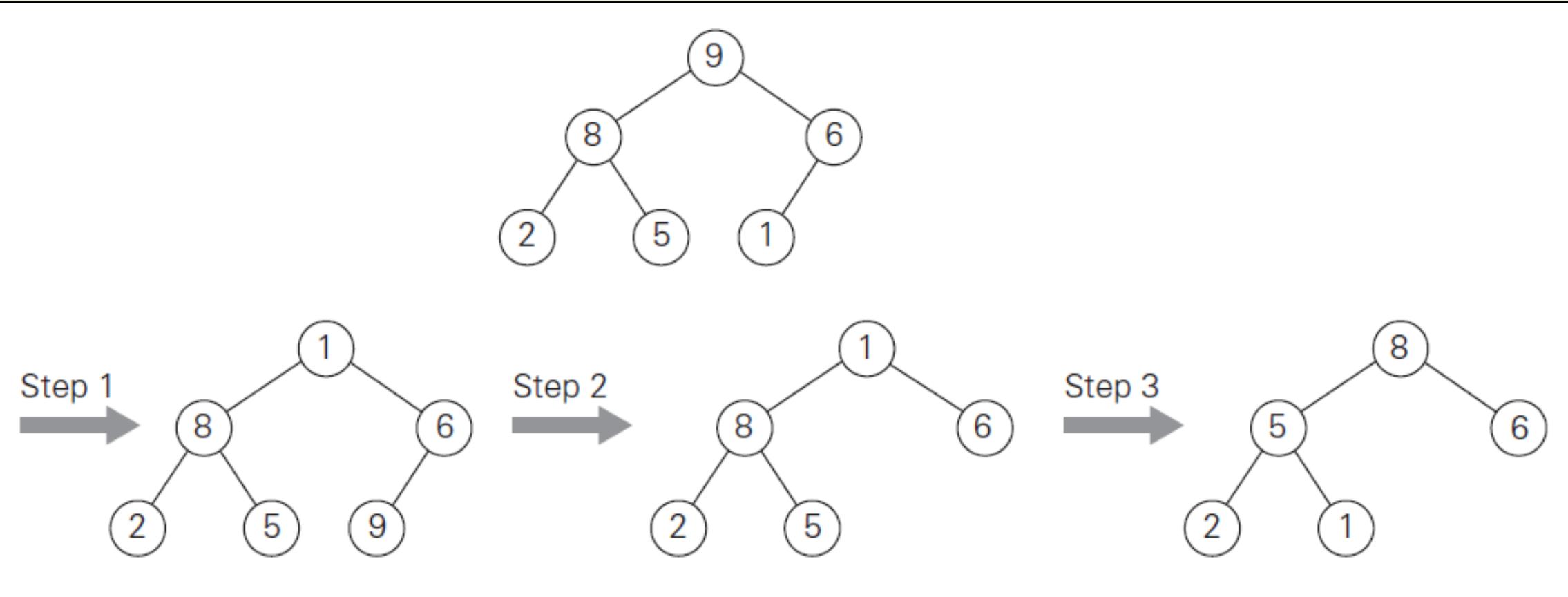
Step 1 Exchange the root's key with the last key K of the heap.

Step 2 Decrease the heap's size by 1.

Step 3 “Heapify” the smaller tree by sifting K down the tree exactly in the same way we did it in the bottom-up heap construction algorithm. That is, verify the parental dominance for K : if it holds, we are done; if not, swap K with the larger of its children and repeat this operation until the parental dominance condition holds for K in its new position.

Heaps and Heap Sort

Steps illustrating the deletion of root element from the heap



Heaps and Heap Sort

Heap Sort

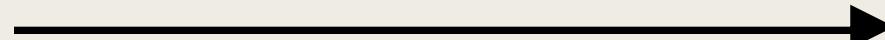
- An interesting sorting algorithm discovered by J. W. J. Williams
- This is a two-stage algorithm that works as follows.
 - Stage 1 (heap construction): Construct a heap for a given array.
 - Stage 2 (maximum deletions): Apply the root-deletion operation $n - 1$ times to the remaining heap.

Heaps and Heap Sort

Sort the following elements using heap sort

2, 9, 7, 6, 5, 8

Stage 1 (heap construction)						
2	9	7	6	5	8	
2	9	8	6	5	7	
2	9	8	6	5	7	
9	2	8	6	5	7	
9	6	8	2	5	7	



Stage 2 (maximum deletions)						
9	6	8	2	5	7	
7	6	8	2	5	1	9
8	6	7	2	5		
5	6	7	2	1	8	
7	6	5	2			
2	6	5	1	7		
6	2	5				
5	2	1	6			
5	2					
2	1	5				
						2



GREEDY METHOD