Parul® University
Vadodara, Gujarat
NAAC GRADE A++

Information and Communication Technology

Greedy Algorithms
Chapter 1

Mrs. Bhumi Shah
Assistant Professor
Computer Science and Engineering

Content

1. Introduction, Elements of Greedy Strategy
2. Minimum Spanning Tree:

- Kruskal’s Algorithm
- Prim’s Algorithm
- Dijkstra’s Algorithm

3. Knapsack Problem, Activity Selection Problem, Huffman Codes

Introduction to Greedy Strategy

- A **Greedy Algorithm** builds up a solution **piece by piece**, always choosing the option that looks best at the moment.
- It **does not reconsider** its choices once made.
- Works well for optimization problems (e.g., minimum, maximum).
- Simpler and more efficient than dynamic programming but doesn't always guarantee optimal solution.

Characteristics of Greedy Algorithms

Greedy Choice Property:

- A global optimum can be arrived at by selecting a local optimum.
- Feasibility: Only choose options that satisfy the problem's constraints.
- Optimal Substructure: A problem has an optimal solution that includes optimal solutions to subproblems.

When to Use Greedy Algorithms

When a problem exhibits:
- Greedy-choice property
- Optimal substructure

If a greedy approach fails to provide the correct solution, consider Dynamic Programming.

Real-Life Examples of Greedy Strategy

- Coin Change Problem (Limited to certain denominations)
- Activity Selection Problem
- Huffman Coding
- Kruskal’s and Prim’s Algorithms for Minimum Spanning Tree
- Dijkstra’s Algorithm for Shortest Path

Elements of Greedy Strategy

1. Candidate Set: A list of possible candidates to be chosen.
2. Selection Function: Chooses the best candidate to add to the solution.
3. Feasibility Function: Determines whether a candidate can be added without violating the problem's constraints.
4. Solution Function: Determines whether a complete solution has been reached.

Greedy Algorithm Structure (Pseudo-code)

```
Greedy(A)
  solution = Ø
  while feasible(solution)
    x = select(A)
    if is_feasible(solution, x)
       solution = solution ∪ {x}
  return solution
```

Parul® University | NAAC GRADE A++

https://paruluniversity.ac.in/

Greedy Algorithms
Chapter 1

Mrs. Bhumi Shah
Assistant Professor
Computer Science and Engineering

Content

1. Introduction, Elements of Greedy Strategy
2. Minimum Spanning Tree:

- Kruskal's Algorithm
- Prim's Algorithm
- Dijkstra's Algorithm

3. Knapsack Problem, Activity Selection Problem, Huffman Codes

Introduction to Greedy Strategy

- A **Greedy Algorithm** builds up a solution **piece by piece**, always choosing the option that looks best at the moment.
- It **does not reconsider** its choices once made.
- Works well for optimization problems (e.g., minimum, maximum).
- Simpler and more efficient than dynamic programming but doesn't always guarantee optimal solution.

Characteristics of Greedy Algorithms

Greedy Choice Property:

- A global optimum can be arrived at by selecting a local optimum.
- Feasibility: Only choose options that satisfy the problem's constraints.
- Optimal Substructure: A problem has an optimal solution that includes optimal solutions to subproblems.

When to Use Greedy Algorithms

When a problem exhibits:

- Greedy-choice property
- Optimal substructure

If a greedy approach fails to provide the correct solution, consider Dynamic Programming.

Real-Life Examples of Greedy Strategy

- Coin Change Problem (Limited to certain denominations)
- Activity Selection Problem
- Huffman Coding
- Kruskal’s and Prim’s Algorithms for Minimum Spanning Tree
- Dijkstra’s Algorithm for Shortest Path

Elements of Greedy Strategy

1. Candidate Set: A list of possible candidates to be chosen.
2. Selection Function: Chooses the best candidate to add to the solution.
3. Feasibility Function: Determines whether a candidate can be added without violating the problem's constraints.
4. Solution Function: Determines whether a complete solution has been reached.

Greedy Algorithm Structure (Pseudo-code)

```
Greedy(A)
  solution = Ø
  while feasible(solution)
    x = select(A)
    if is_feasible(solution, x)
       solution = solution ∪ {x}
  return solution
```

Introduction to Minimum Spanning Tree (MST)

Let $G = \langle N, A \rangle$ be a **connected, undirected graph** where,

1. N is the set of nodes and
2. A is the set of edges.

Each edge has a given **positive length or weight**.

A spanning tree of a graph G **is a sub-graph** which is basically a tree and it contains all the vertices of G but does not contain cycle.

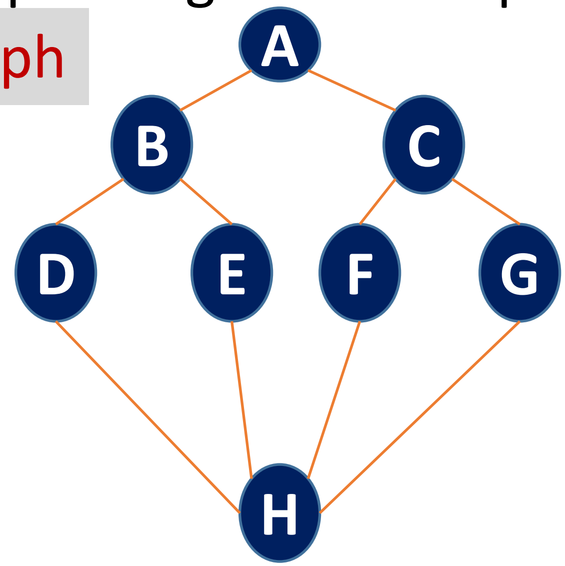
A minimum spanning tree (MST) of a **weighted connected graph** G is a spanning tree with minimum or smallest weight of edges.

Two Algorithms for **constructing** minimum spanning tree are,

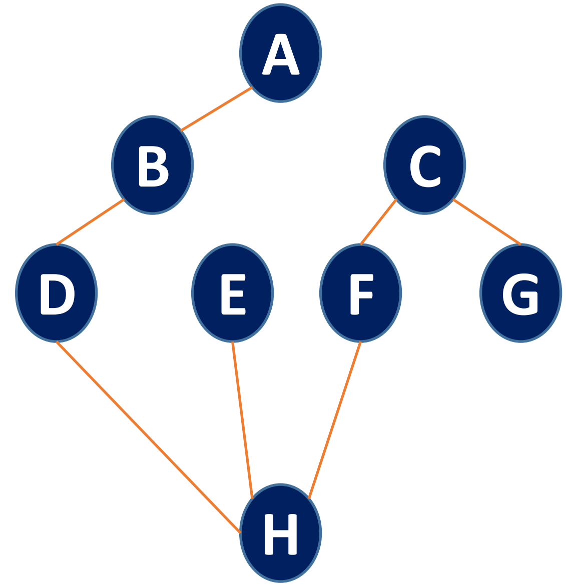
1. Kruskal's Algorithm
2. Prim's Algorithm

Spanning Tree Examples

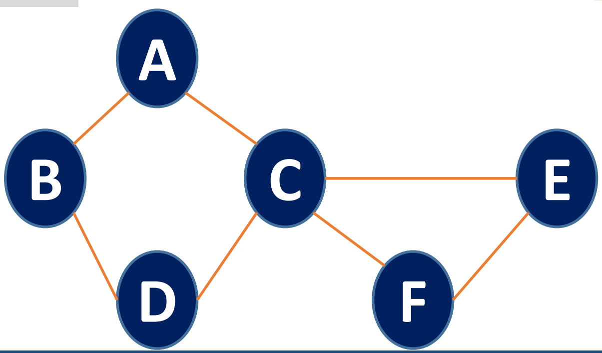
Graph



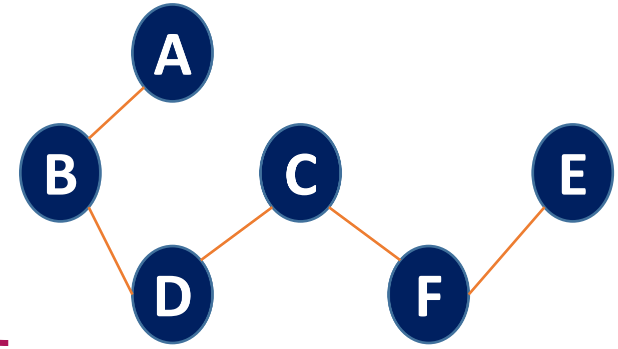
Spanning Tree



Graph

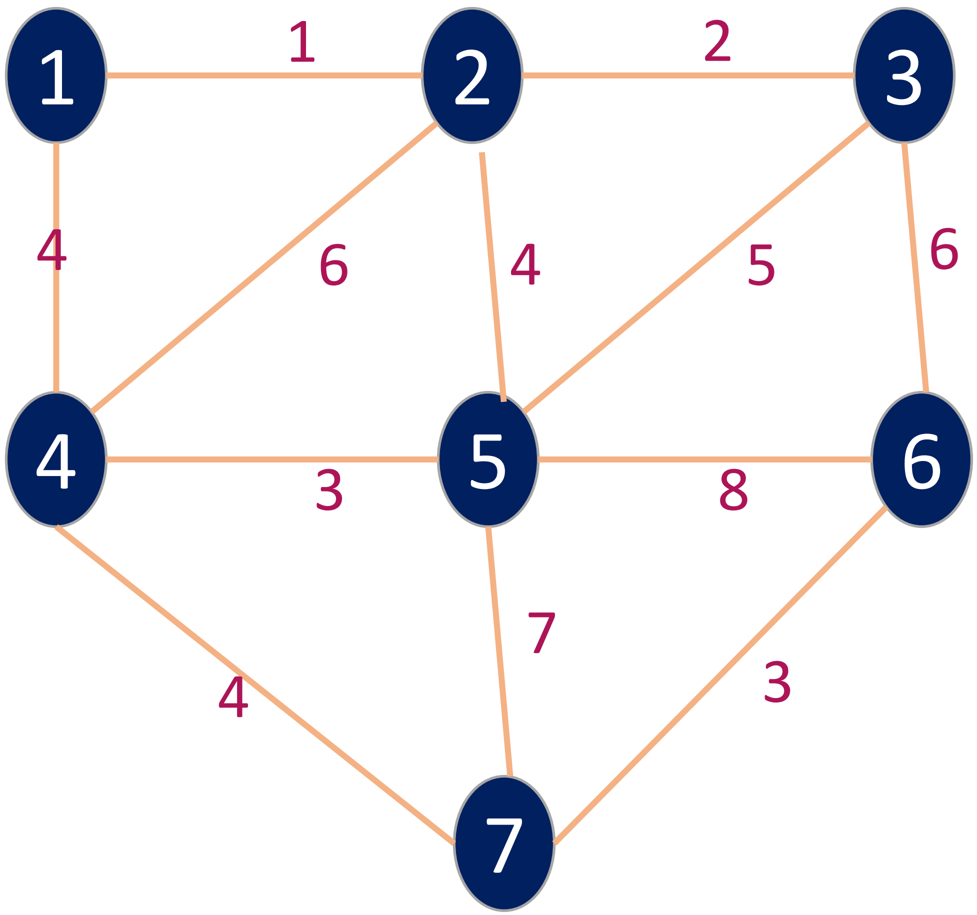


Spanning Tree



Parul® University | NAAC GRADE A++
Vadodara, Gujarat

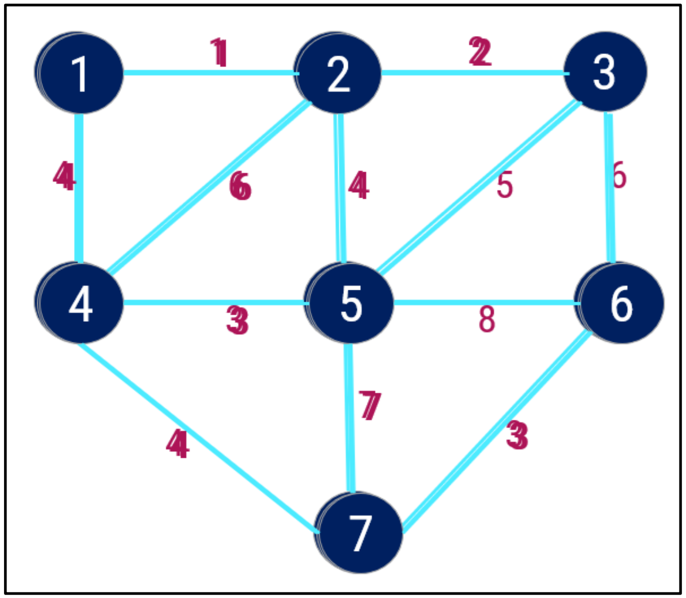
Prim's Algorithm for MST – Example 1



Step:1 Select an arbitrary node.

Node - Set B	Edges
1	

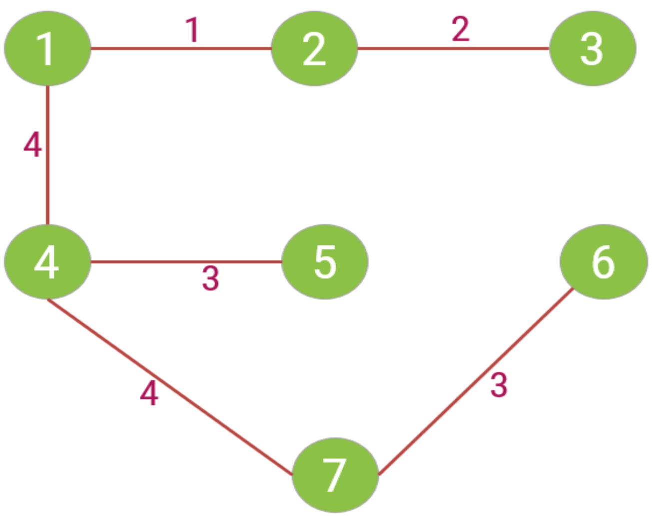
Prim's Algorithm for MST – Example 1



Step:2 Find an edge with minimum weight.

Node - Set B	Edges
1	{1, 2}, ~~{1, 4}~~
1, 2	~~{1, 4}~~, {2, 3} ~~{2, 4}, {2, 5}~~
1, 2, 3	{1,4}, ~~{2,4}, {2,5}, {3,5}, {3,6}~~
1, 2, 3, 4	~~{2,4} {2,5} {3,5} {3,6}~~ {4,5} ~~{4,7}~~
1, 2, 3, 4, 5	~~{2,4} {2,5} {3,5} {3,6}~~ {4,7} ~~{5,6} {5,7}~~
1, 2, 3, 4, 5, 7	~~{2,4} {2,5} {3,5} {3,6} {5,6} {5,7}~~ {6,7}
1, 2, 3, 4, 5, 6, 7	

Prim’s Algorithm for MST – Example 1



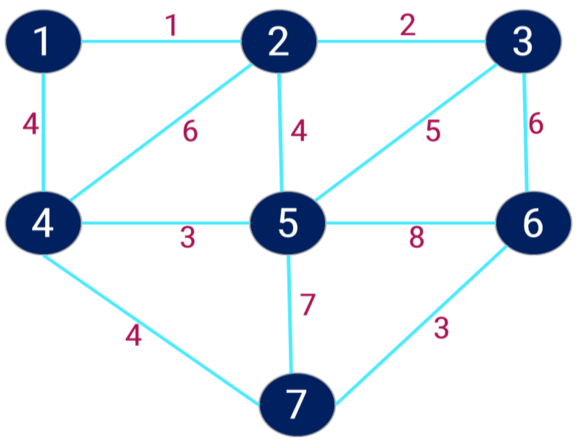
Step:3

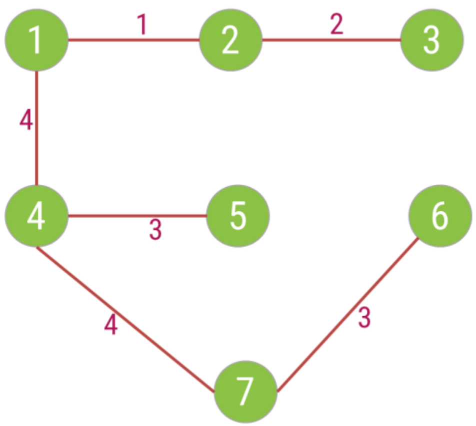
The minimum spanning tree for the given graph.

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

Total Cost = 17

Prim's Algorithm for MST – Example 1





Cost = 17

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

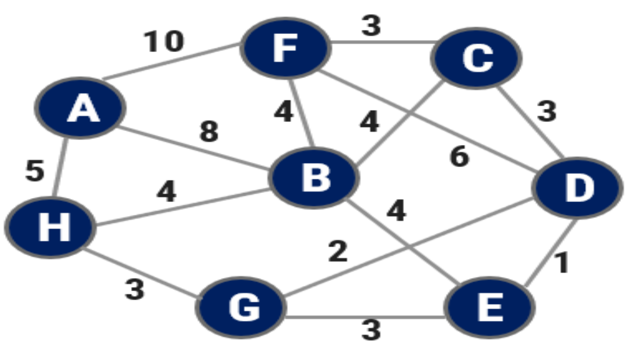
Prim's Algorithm

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

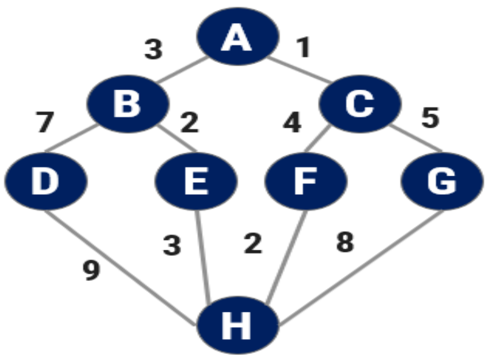
Exercises – Home Work

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

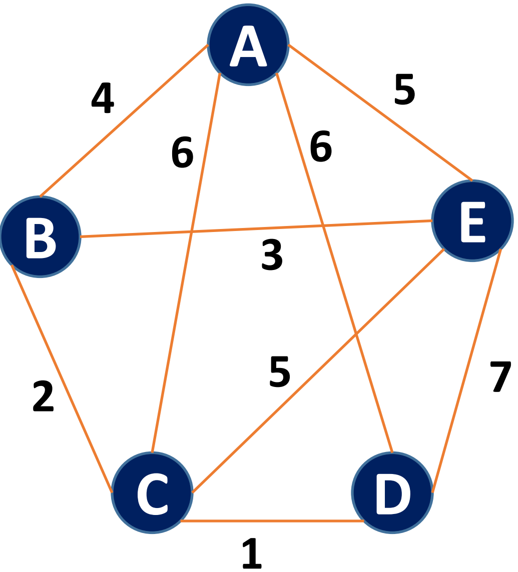
1.



2.

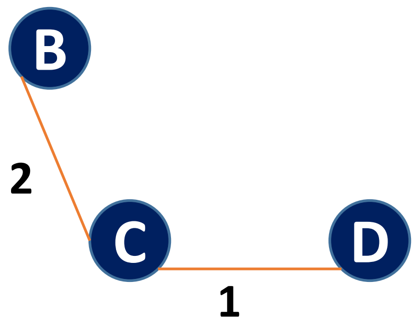


Kruskal's Algorithm for MST – Example 1

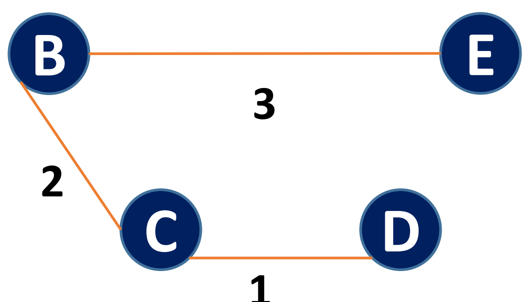


Step 1: Taking min edge (C,D)

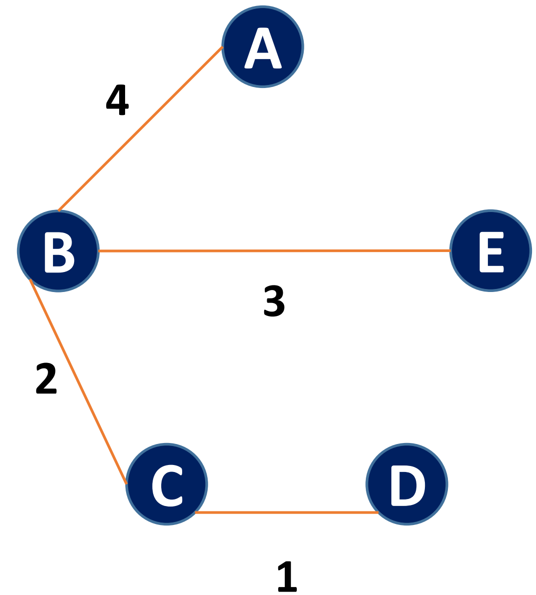
Step 2: Taking next min edge (B,C)



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

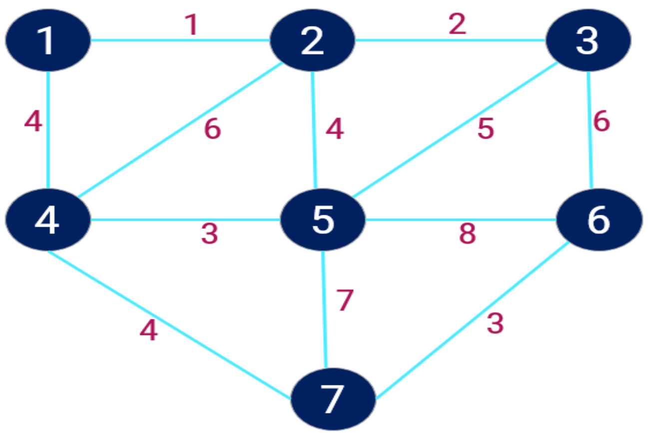


Step 4: Taking next min edge (A,B)



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

Kruskal's Algorithm for MST – Example 2

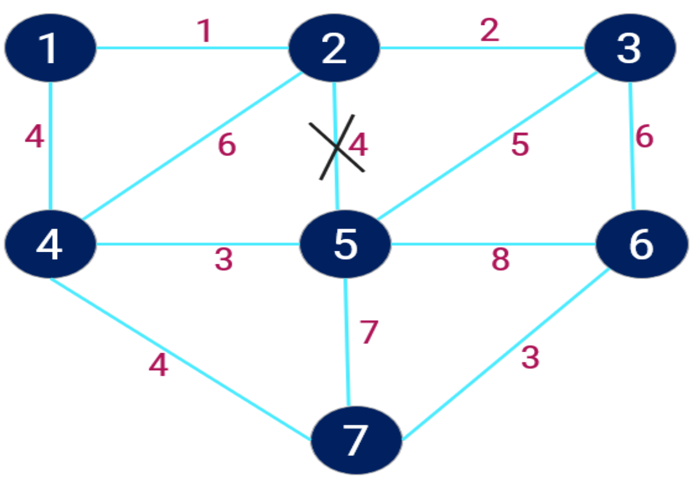


Step:1

Sort the edges in increasing order of their weight.

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

Kruskal’s Algorithm for MST – Example 2

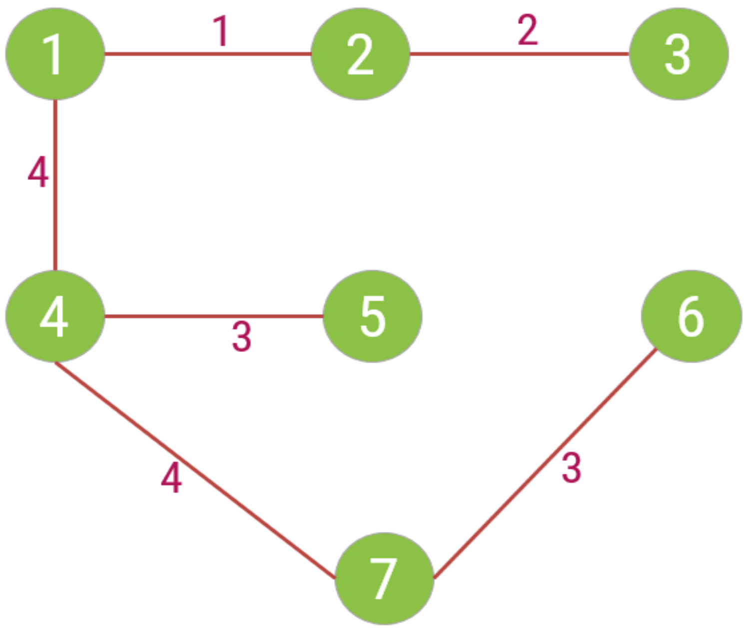


Step:2

Select the minimum weight edge but no cycle.

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

Kruskal's Algorithm for MST – Example 2



Step:3

The minimum spanning tree for the given graph.

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

Total Cost = 17

Kruskal’s Algorithm

Function Kruskal(G = (N, A))

Sort A by increasing length

n ← the number of nodes in N

T ← Ø {edges of the minimum spanning tree}

Define n sets, containing a different element of set N

repeat

e ← {u, v} //e is the shortest edge not yet considered

ucomp ← find(u)

vcomp ← find(v)

if ucomp ≠ vcomp then merge(ucomp, vcomp)

T ← T U {e}

until T contains n - 1 edges

return T

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

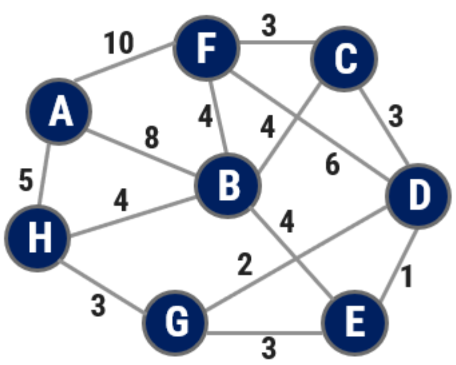
merge(ucomp, vcomp) is used to merge two connected components.

Exercises – Home Work

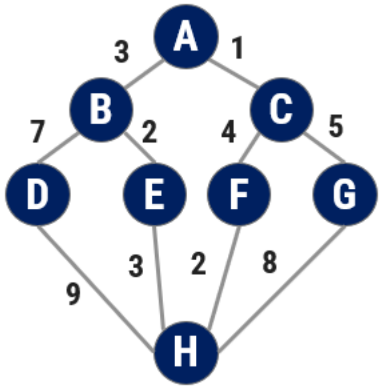
- The complexity for the Kruskal's algorithm is in $\boldsymbol{\theta(a \log n)}$ where $\boldsymbol{a}$ is total number of edges and $\boldsymbol{n}$ is the total number of nodes in the graph G.

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

1.



2.

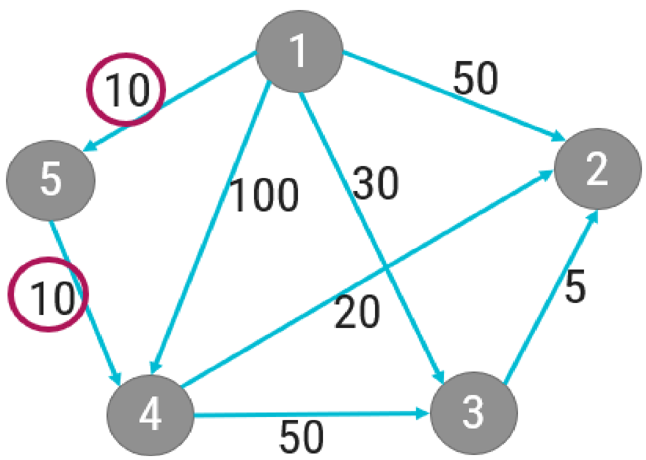


Dijkstra's Algorithm

- Consider now **a** directed graph $G = (N, A)$ where N is the set of nodes and A is the set of directed edges of graph G.
- Each edge has a positive length.
- One of the nodes is designated as the source node.
- The problem is to determine the length of the shortest path from the source to each of the other nodes of the graph.
- Dijkstra's Algorithm is for finding the shortest paths between the nodes in a graph.
- For a given source node, the algorithm finds the shortest path between the source node and every other node.
- The **algorithm maintains a matrix L** which gives the length of each directed edge:

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

Dijkstra's Algorithm - Example



Single source shortest path algorithm

Source node = 1

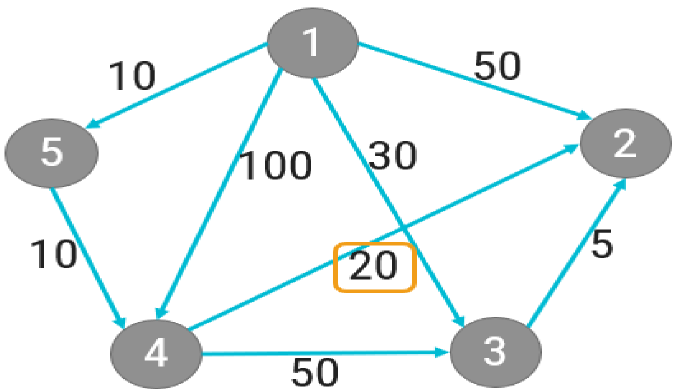
Step	v	C	2	3	4	5
Init.	-	{2, 3, 4, 5}	50	30	100	10
1	5	{2, 3, 4}	50	30	20	10

Is there path from 1 - 5 - 4

Yes

Compare cost of 1–5–4 (20) and 1–4 (100)

Dijkstra's Algorithm - Example



Single source shortest path algorithm

Source node = 1

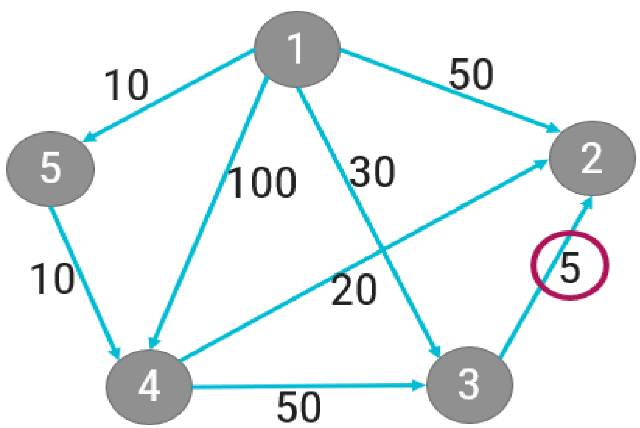
Step	v	C	2	3	4	5
Init.	-	{2, 3, 4, 5}	50	30	100	10
1	5	{2, 3, 4}	50	30	20	10
2	4	{2, 3}	40	30	20	10

Is there path from 1 - 4 - 5

No

Compare cost of 1–4–3 (70) and 1–3 (30)

Dijkstra's Algorithm - Example



Single source shortest path algorithm

Source node = 1

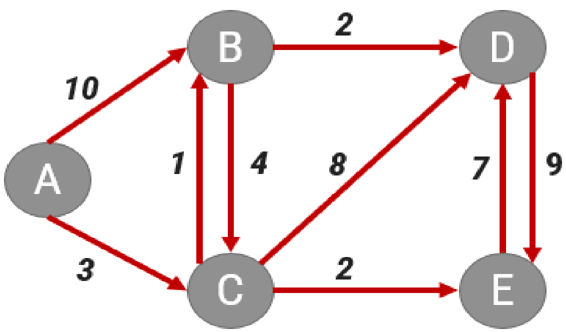
Step	v	C	2	3	4	5
Init.	-	{2, 3, 4, 5}	50	30	100	10
1	5	{2, 3, 4}	50	30	20	10
2	4	{2, 3}	40	30	20	10
3	3	{2}	35	30	20	10

Compare cost of
1–3–2 and 1–2

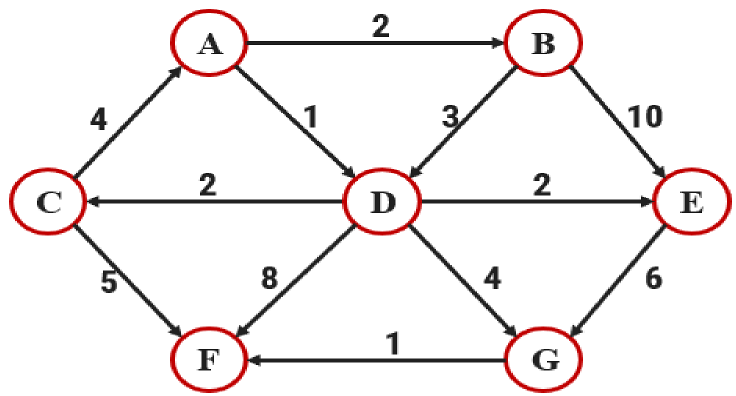
Exercises – Home Work

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

1.



2.



Exercises – Home Work

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

Parul®
University

NAAC GRADE A++

https://paruluniversity.ac.in/

Parul® University NAAC GRADE A++
Vadodara, Gujarat

Information and Communication Technology

Greedy Algorithms
Chapter 1

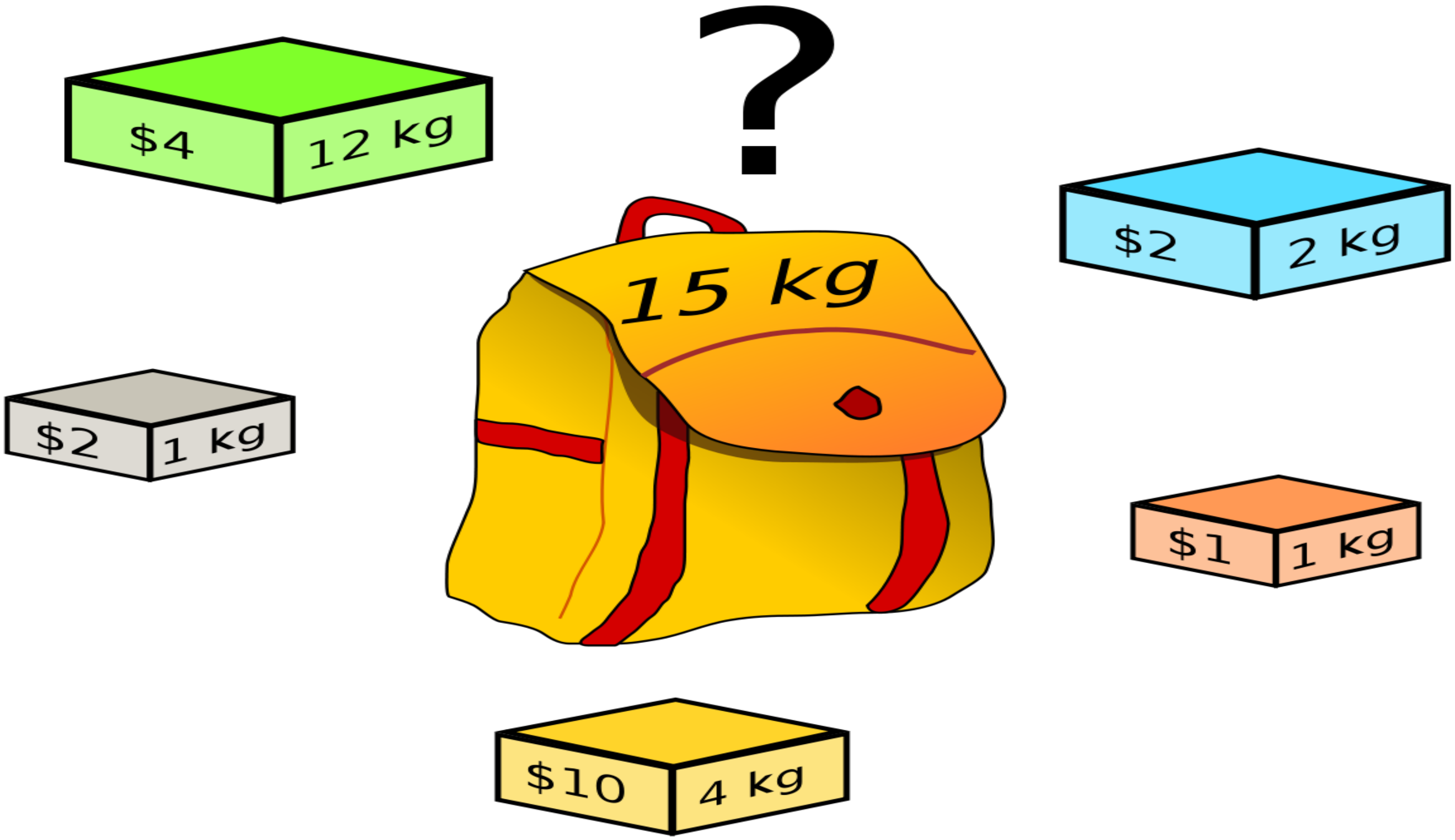
Mrs. Bhumi Shah
Assistant Professor
Computer Science and Engineering

Content

1. Introduction, Elements of Greedy Strategy
2. Minimum Spanning Tree:
- Kruskal’s Algorithm
- Prim’s Algorithm
- Dijkstra’s Algorithm

3. Knapsack Problem, Activity Selection Problem, Huffman Codes

Knapsack Problem



Fractional Knapsack Problem

- We are given n objects and a knapsack.
- Object $\boldsymbol{i}$ has a positive weight $\boldsymbol{w_i}$ and a positive value $\boldsymbol{v_i}$ for $\boldsymbol{i} = \boldsymbol{1}, \boldsymbol{2} \ldots \boldsymbol{n}$.
- The knapsack can carry a weight not exceeding $\boldsymbol{W}$.
- Our aim is to fill the knapsack in a way that **maximizes** the value of the included objects, while respecting the capacity constraint.
- In a fractional knapsack problem, we assume that the objects **can be broken into smaller pieces**.

Fractional Knapsack Problem

- So we may decide to carry only a fraction x_i of object i,

  where $0 \leq x_i \leq 1$.
- In this case, object i contribute $x_i w_i$ to the total weight in the knapsack, and $x_i v_i$ to the value of the load.
- Symbolic Representation of the problem can be given as follows:

maximize $\sum_{i=1}^{n} x_i v_i$ **subject to** $\sum_{i=1}^{n} x_i w_i \leq W$

Where, $v_i > 0$, $w_i > 0$ and $0 \leq x_i \leq 1$ for $1 \leq i \leq n$.

Fractional Knapsack Problem - Example

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

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

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

Fractional Knapsack Problem - Greedy Solution

Three Selection Functions can be defined as,

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

Fractional Knapsack Problem - Greedy Solution

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

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

Weight Capacity 100

30	50	20	
10	20	30	40
30	10	20	40

Profit = 66 + 20 + 30 + 48 = 164

Fractional Knapsack Problem - Algorithm

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

Exercises – Home Work

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

Activity Selection Problem

- The Activity Selection Problem is an optimization problem which deals with the selection of non-overlapping activities that needs to be executed by a single person or a machine in a given time duration.
- An activity-selection can also be applicable for scheduling a resource among several competing activities.
- We are given a set S of n activities with start time s_i and finish time f_i, of an i^{th} activity. Find the maximum size set of mutually compatible activities.
- Activities i and j are compatible if the half-open internal $[s_i, f_i)$ and $[s_j, f_j)$ do not overlap, that is, i and j are compatible if $s_i \geq f_j$ or $s_j \geq f_i$.

Activity Selection Problem-Example

Sr.	Activity	(s_i, f_i)
1	P	(1, 4)
2	Q	(3, 5)
3	R	(0, 6)
4	S	(5, 7)
5	T	(3, 8)
6	U	(5, 9)
7	V	(6, 10)
8	W	(8, 11)
9	X	(8, 12)
10	Y	(2, 13)
11	Z	(12, 14)

Example: 11 activities are given as,

Solution:

Step 1:
Sort the activities of set S as per increasing finish time to directly identify mutually compatible activities by comparing finish time of first activity and start time of next activity

Activity Selection Problem-Example

Sr.	Activity	(s_i, f_i)
1	P	$\boxed{(1, 4)}$
~~2~~	~~Q~~	~~(3, 5)~~
3	R	(0, 6)
4	S	$\boxed{(5, 7)}$
5	T	(3, 8)
6	U	(5, 9)
7	V	(6, 10)
8	W	$\boxed{(8, 11)}$
9	X	(8, 12)
10	Y	(2, 13)
11	Z	$\boxed{(12, 14)}$

Example: 11 activities are given as,

Solution:

Step 2:

1. A = {P}
2. A = {P, S}
3. A = {P, S, W}
4. A = {P, S, W, Z}

Answer: A = {P, S, W, Z}

Activity Selection Problem

Algorithm: Activity Selection

Step I: Sort the input activities by increasing finishing time. $f_1 \leq f_2 \leq \ldots \leq f_n$

Step II: Call GREEDY-ACTIVITY-SELECTOR (s, f)

```
n = length [s]
A = {i}
j = 1
for  i = 2  to  n
  do if   s_i ≥ f_j
     then  A = A U {i}
           j = i
return  set A
```

Huffman Codes

- Prefix code is used for encoding(compression) and Decoding(Decompression).
- Prefix Code: Any code that is not prefix of another code is called prefix code.

Characters	Frequency	Code	Bits
a	45	000	135
b	13	111	39
c	12	101	36
d	16	110	48
e	9	011	27
f	5	001	5
		Total bits	290

Huffman Codes

- Huffman invented a greedy algorithm that constructs an optimal prefix code called a Huffman code.
- Huffman coding is a lossless data compression algorithm.
- It assigns variable-length codes to input characters.
- Lengths of the assigned codes are based on the frequencies of corresponding characters.
- The most frequent character gets the smallest code and the least frequent character gets the largest code.
- The variable-length codes assigned to input characters are Prefix Codes.

Huffman Codes

- In Prefix codes, the codes are assigned in such a way that the code assigned to one character **is not a prefix of code** assigned to any other character.

For example,

a = 01, b = 010 and c = 11 — Not a prefix code

- This is how Huffman Coding makes sure that there is no ambiguity when decoding the generated bit stream.
- There are mainly two major parts in Huffman Coding
- Build a Huffman Tree from input characters.
- Traverse the Huffman Tree and assign codes to characters.

Huffman Codes

- Find the Huffman codes for the following characters.

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

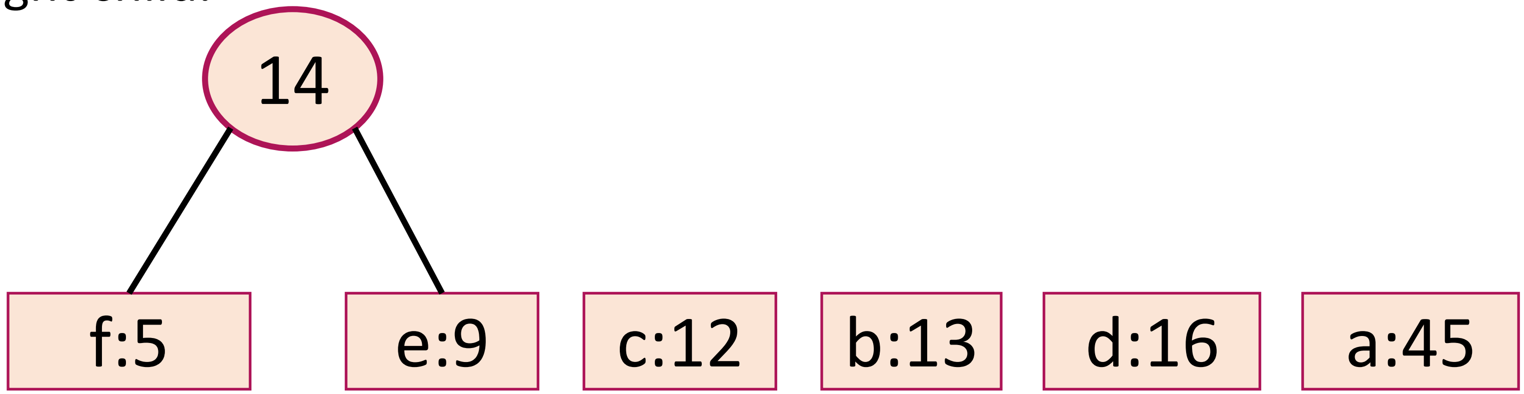
Step 1: Arrange the characters in the Ascending order of their frequency.

f:5	e:9	c:12	b:13	d:16	a:45

Huffman Codes

Step 2:

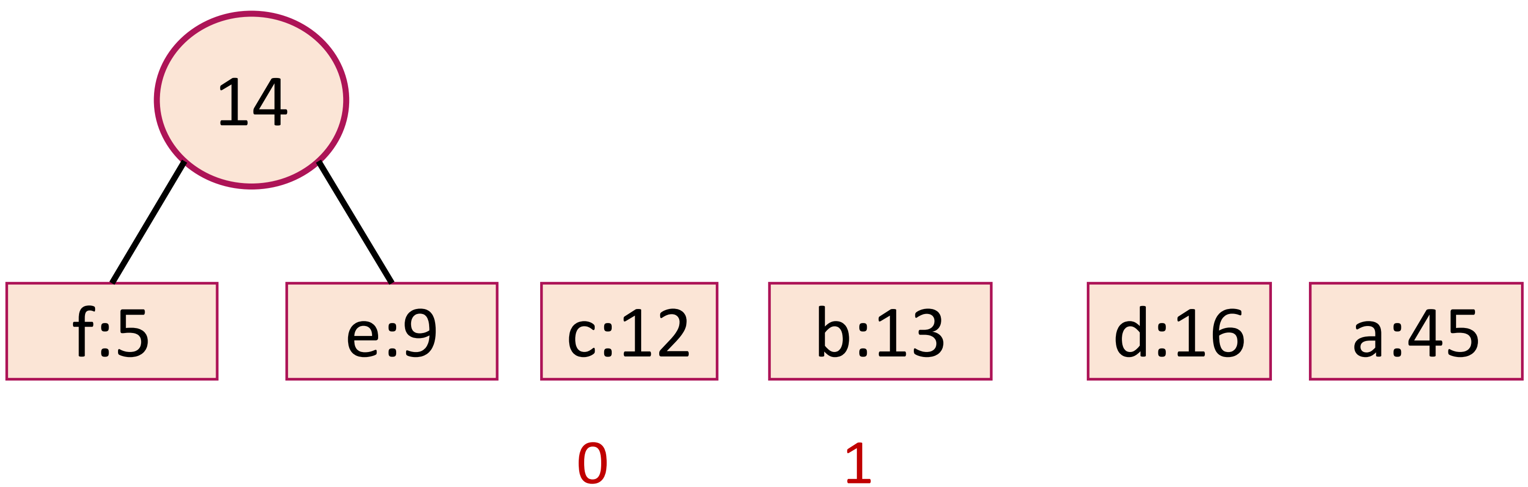
- ✓ Extract two nodes with the minimum frequency.
- ✓ Create a new internal node with frequency equal to the sum of the two nodes frequencies.
- ✓ Make the first extracted node as its left child and the other extracted node as its right child.



Huffman Codes

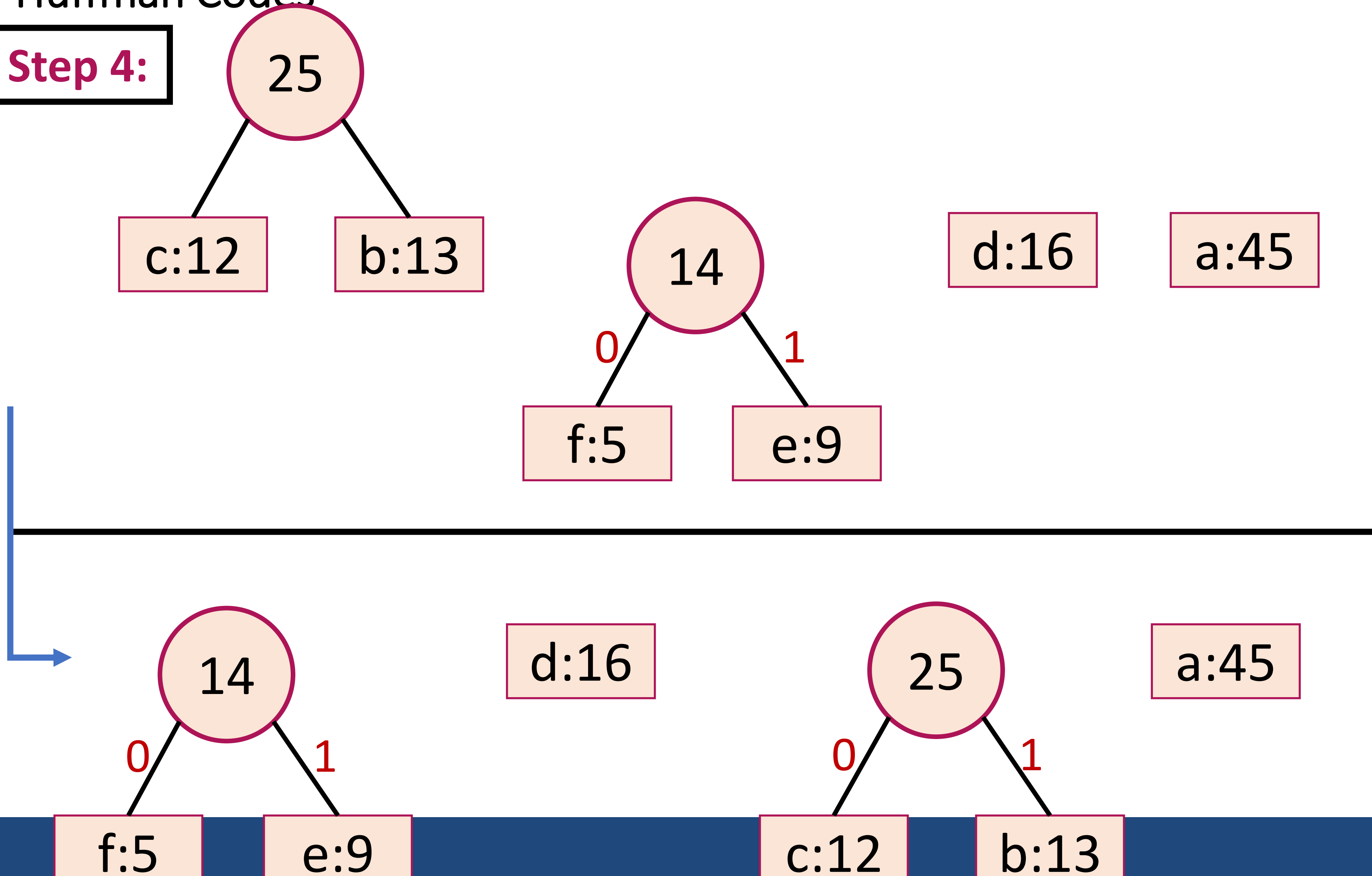
Step 3:

- ✓ Rearrange the tree in ascending order.
- ✓ Assign **0** to the left branch and **1** to the right branch.
- ✓ Repeat the process to complete the tree.



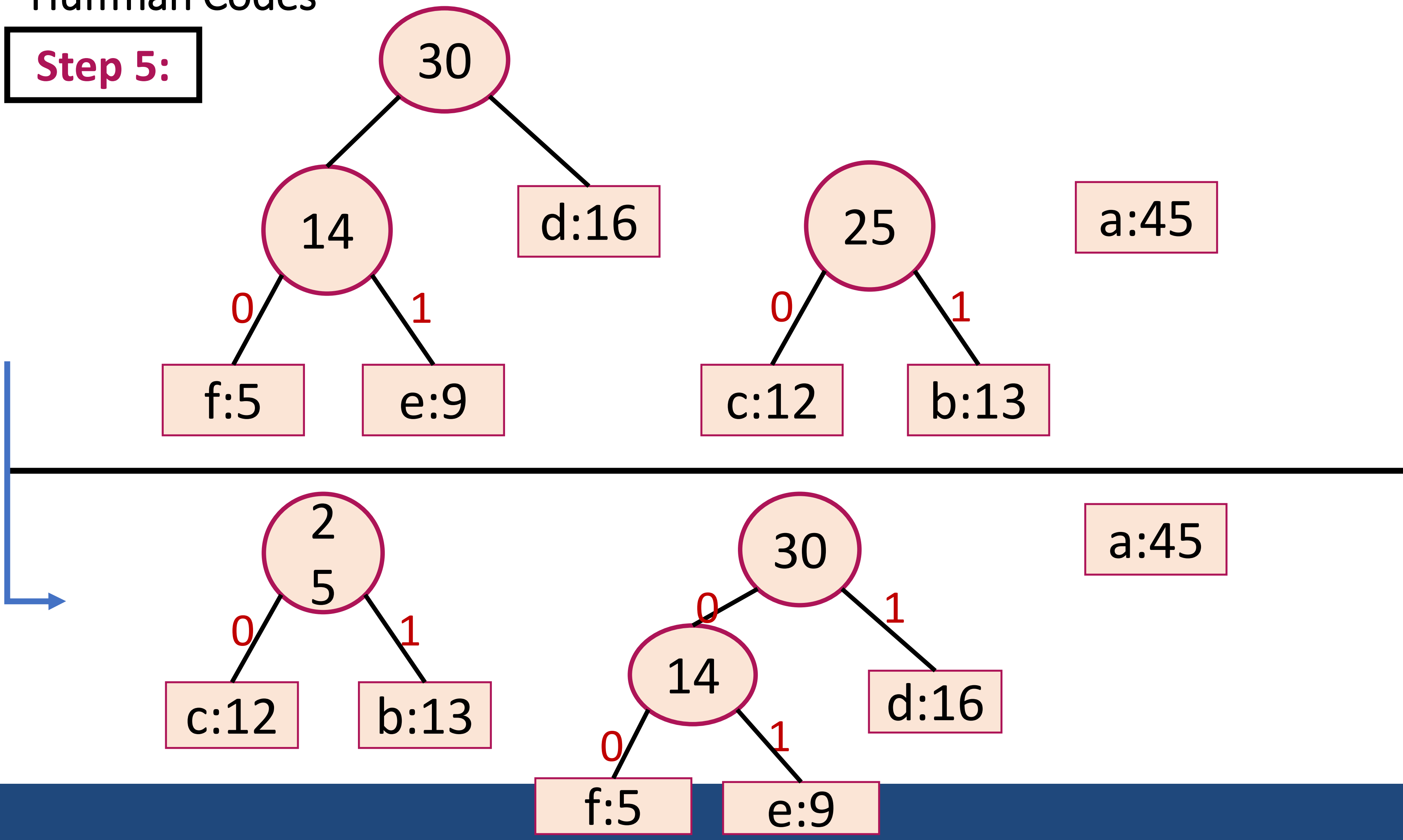
Huffman Codes

Step 4:



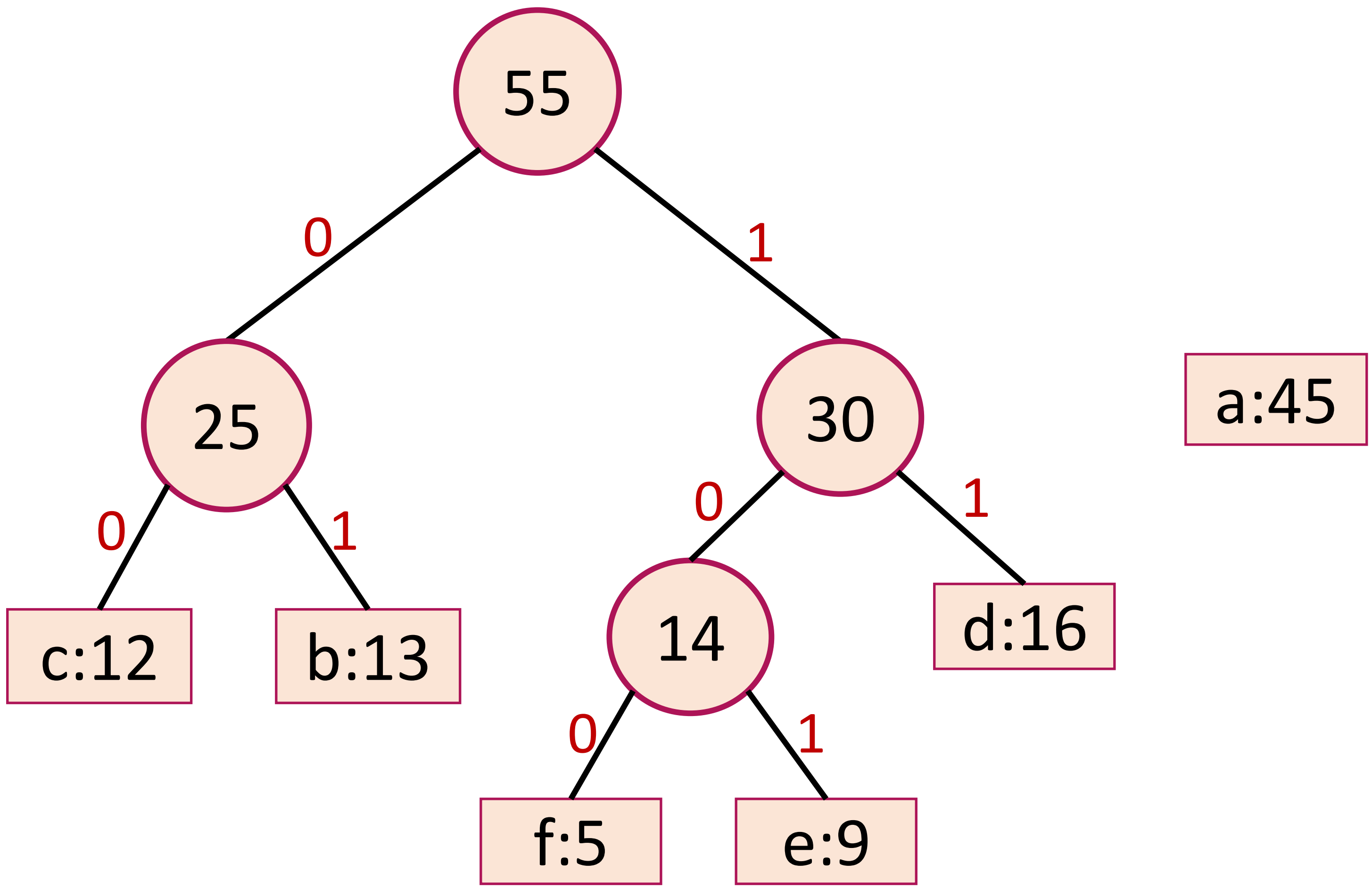
Huffman Codes

Step 5:



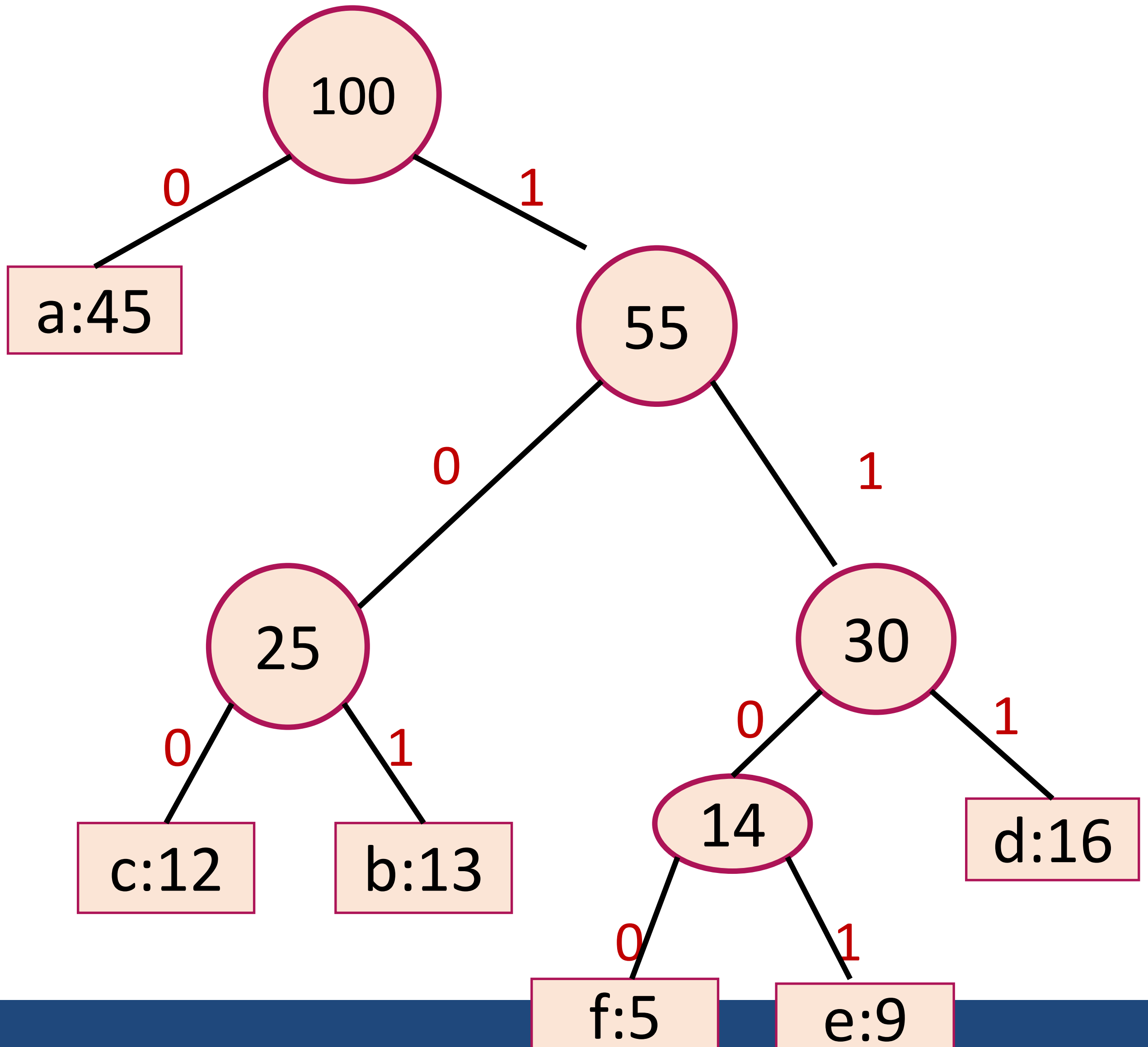
Huffman Codes

Step 6:



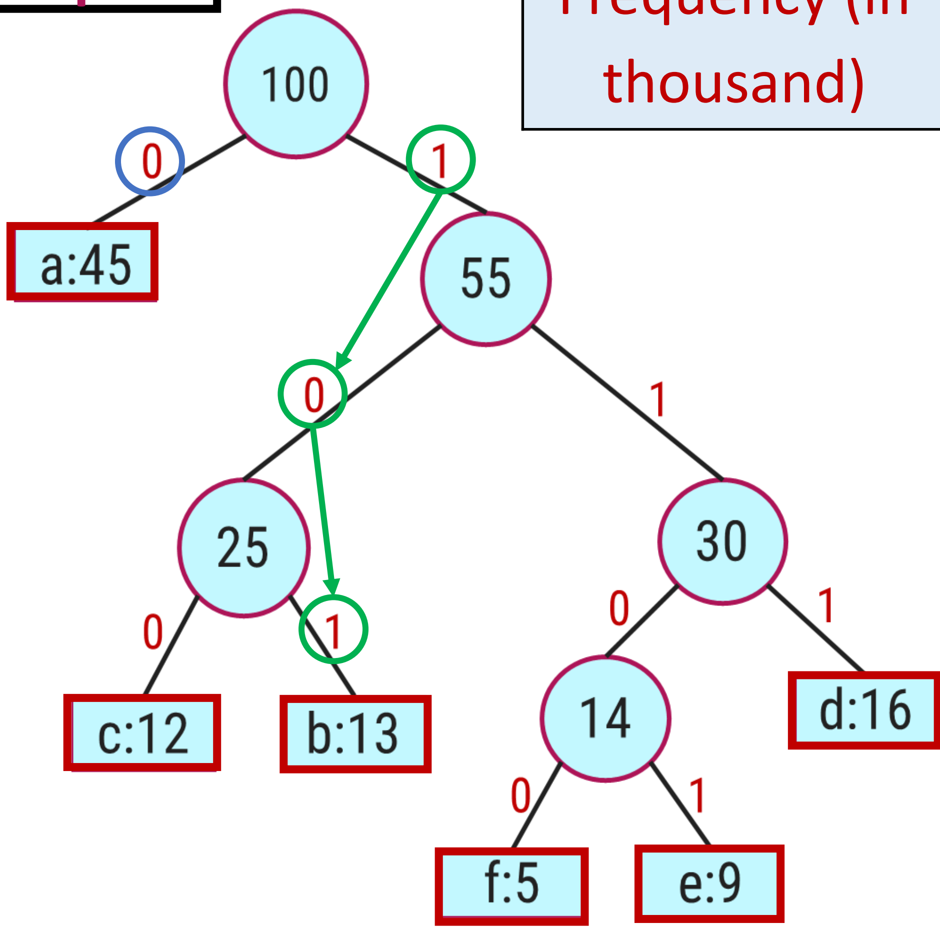
Huffman Codes

Step 7:



Huffman Codes

Step 8:



Characters	a	b	c	d	e	f
Frequency (in thousand)	45	13	12	16	9	5
	0	101	100	111	1101	1100

Huffman Codes

Algorithm: HUFFMAN (C)

```
n = |C|
Q = C
for i = 1 to n-1
      allocate a new node z
      z.left = x = EXTRACT-MIN(Q)
      z.right = y = EXTRACT-MIN(Q)
      z.freq = x.freq + y.freq
      INSERT(Q,z)
return EXTRACT-MIN(Q)  // return the root of the tree
```

Exercises – Home Work

- Find an optimal Huffman code for the following set of frequency.

1. a : 50, b : 20, c : 15, d : 30.
2. Frequency

Characters	A	B	C	D	E	F
Frequency (in thousand)	24	12	10	8	8	5

3. Frequency

Characters	a	b	c	d	e	f	g
Frequency (in thousand)	37	28	29	13	30	17	6

Parul® University | NAAC GRADE A++

https://paruluniversity.ac.in/