Unit-2
Greedy Algorithms

Outline

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

Introduction

Characteristics of Greedy Algorithms

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

- The greedy algorithm consists of four functions.
  i. **Solution Function**:- A function that checks whether chosen set of items provides a solution.
  ii. **Feasible Function**:- A function that checks the feasibility of a set.
  iii. **Selection Function**:- The selection function tells which of the candidates is the most promising.
  iv. **Objective Function**:- An objective function, which does not appear explicitly, but gives the value of a solution.

Make a Change Problem

Problem Definition

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

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

Make Change – Greedy Solution

- If suppose, we need to pay an amount of ₹ 28/- using the available coins.
- Here we have a candidate (coins) set $C = \{\mathbf{10},\ \mathbf{5},\ \mathbf{2},\ \mathbf{1},\ \mathbf{0.5}\}$
- The greedy solution is,

Selected coins are

coins	quantity
10	2
5	1
2	1
1	1

Amount

28

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

Make Change - Algorithm

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

Make Change – The Greedy Property

- The algorithm is **greedy** because,
  - At every step it chooses **the largest available coin**, without worrying whether this will prove to be a **correct** decision later.
  - It **never changes** the decision, i.e., once a coin has been included in the solution, it is there **forever**.
- Examples:
  1. Some coins with denominations 50, 20, 10, 5, 1 are available.
     - How many minimum coins required to make change for 37 cents? $\boxed{5}$
     - How many minimum coins required to make change for 91 cents? $\boxed{4}$
  2. Denominations: $d_1=6$, $d_2=4$, $d_3=1$. Make a change of ₹ 8. ~~3~~

     The minimum coins required are $\boxed{2}$

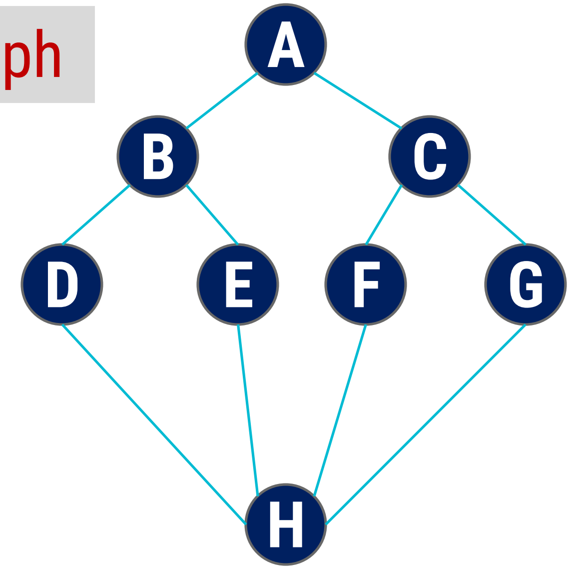
Minimum Spanning Tree

Introduction to Minimum Spanning Tree (MST)

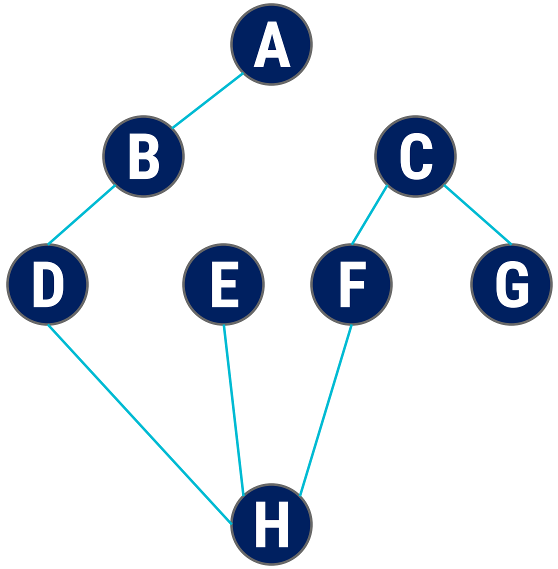
- Let $G = \langle N, A \rangle$ be a **connected, undirected graph** where,
  1. N is the set of nodes and
  2. A is the set of edges.
- Each edge has a given **positive length or weight**.
- A spanning tree of a graph G **is a sub-graph** which is basically a tree and it contains all the vertices of 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

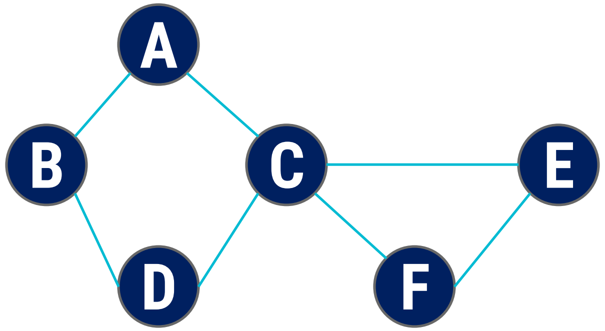
Graph



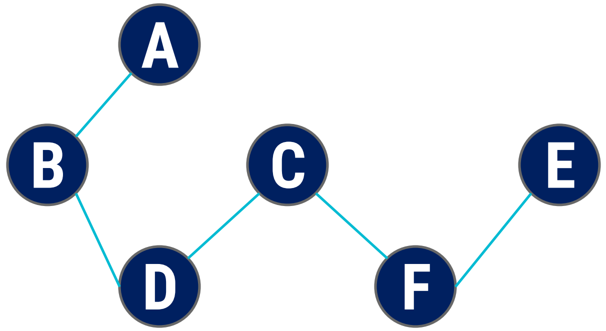
Spanning Tree



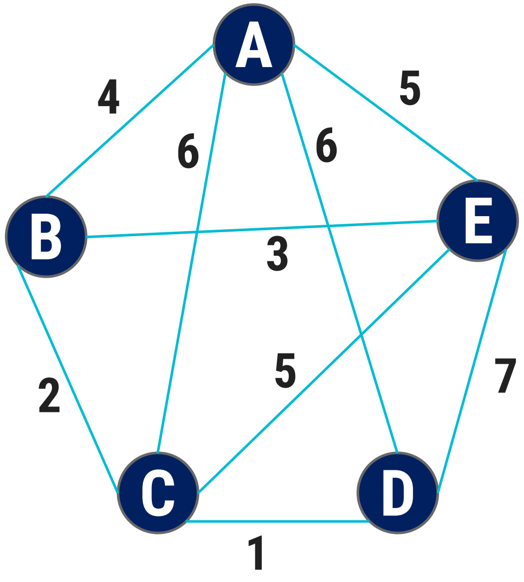
Graph



Spanning Tree

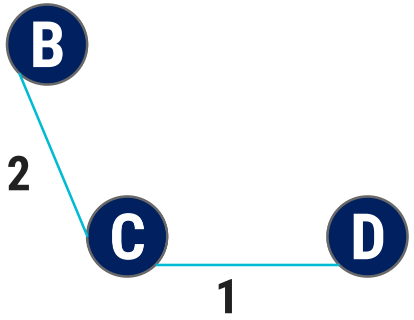


Kruskal’s Algorithm for MST – Example 1

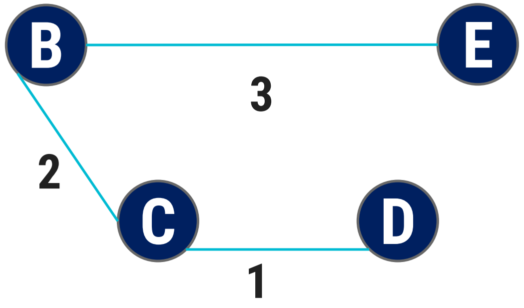


Step 1: Taking min edge (C,D)

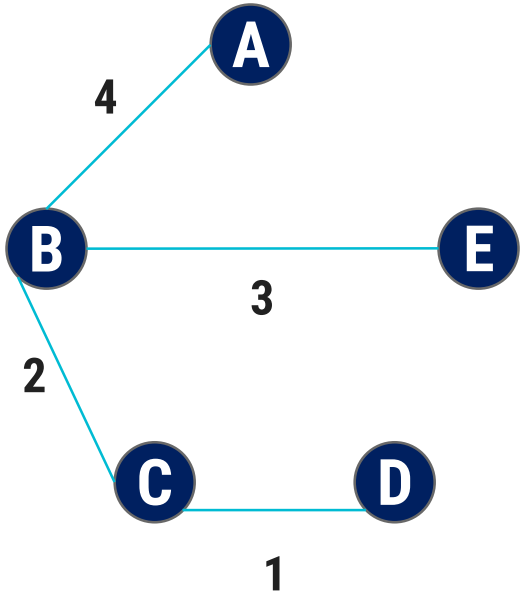
Step 2: Taking next min edge (B,C)



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

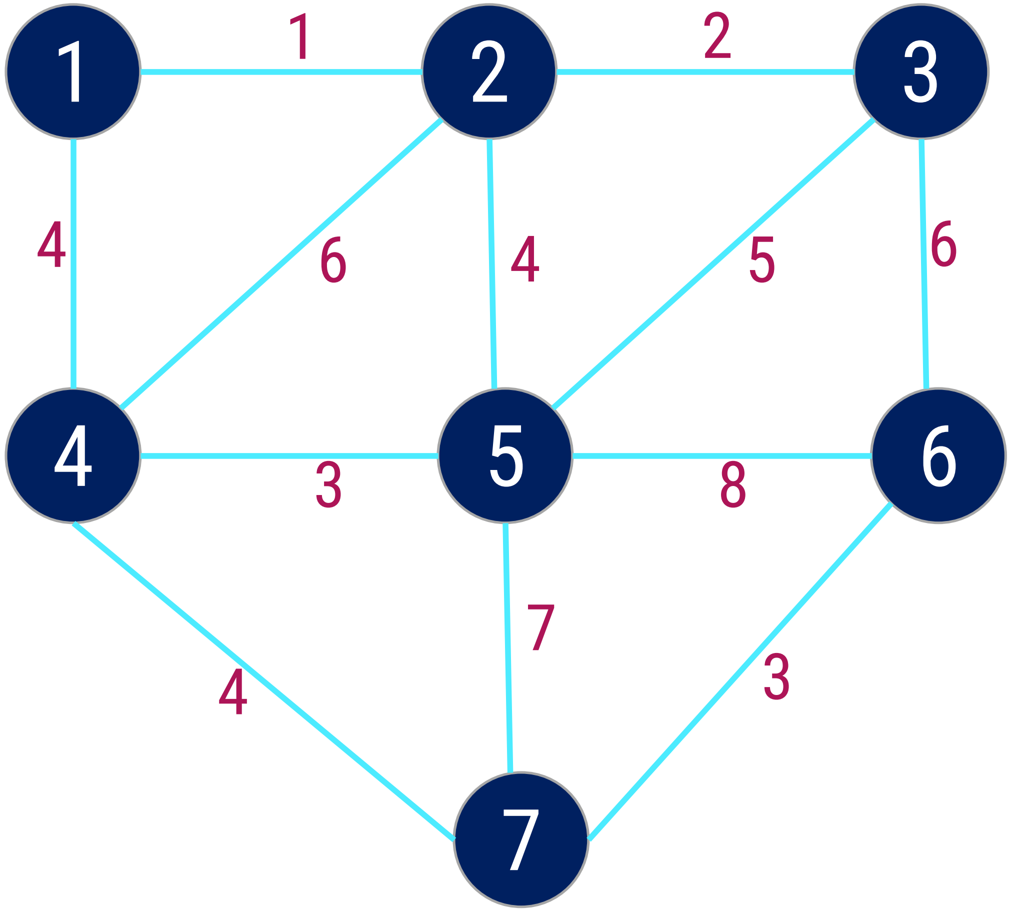


Step 4: Taking next min edge (A,B)



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

Kruskal's Algorithm for MST – Example 2

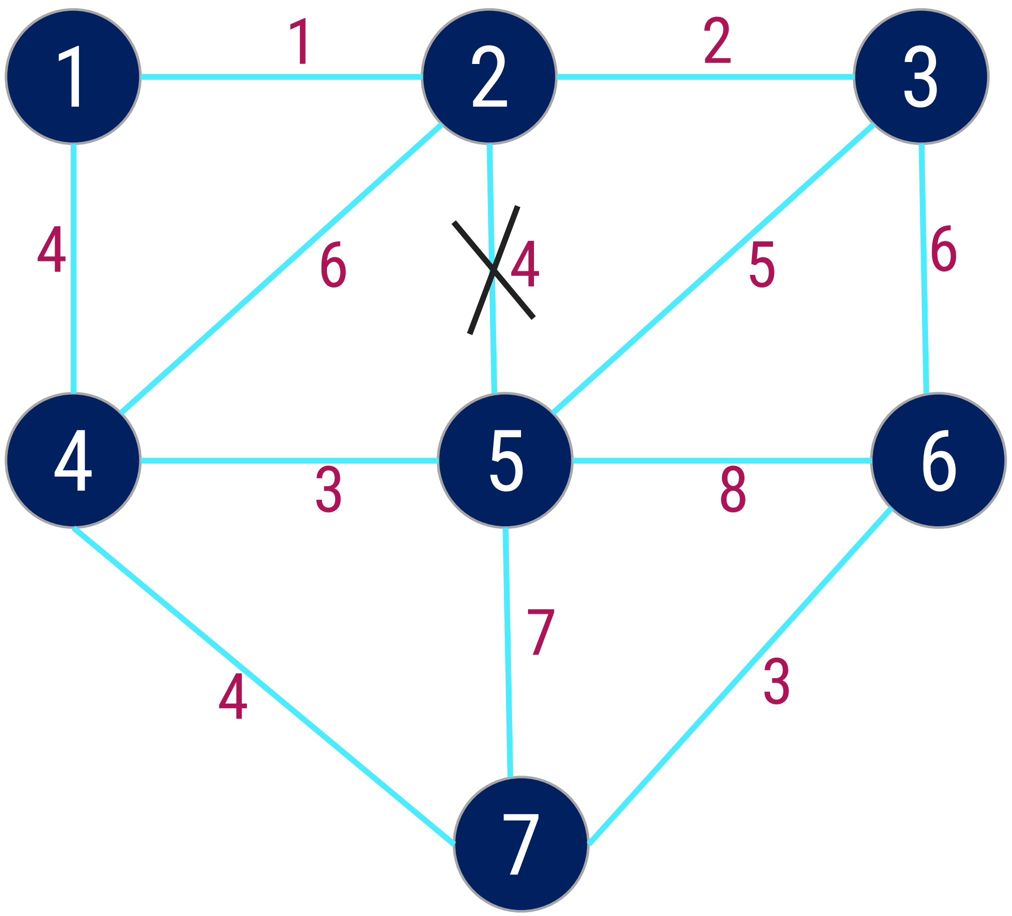


Step:1

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

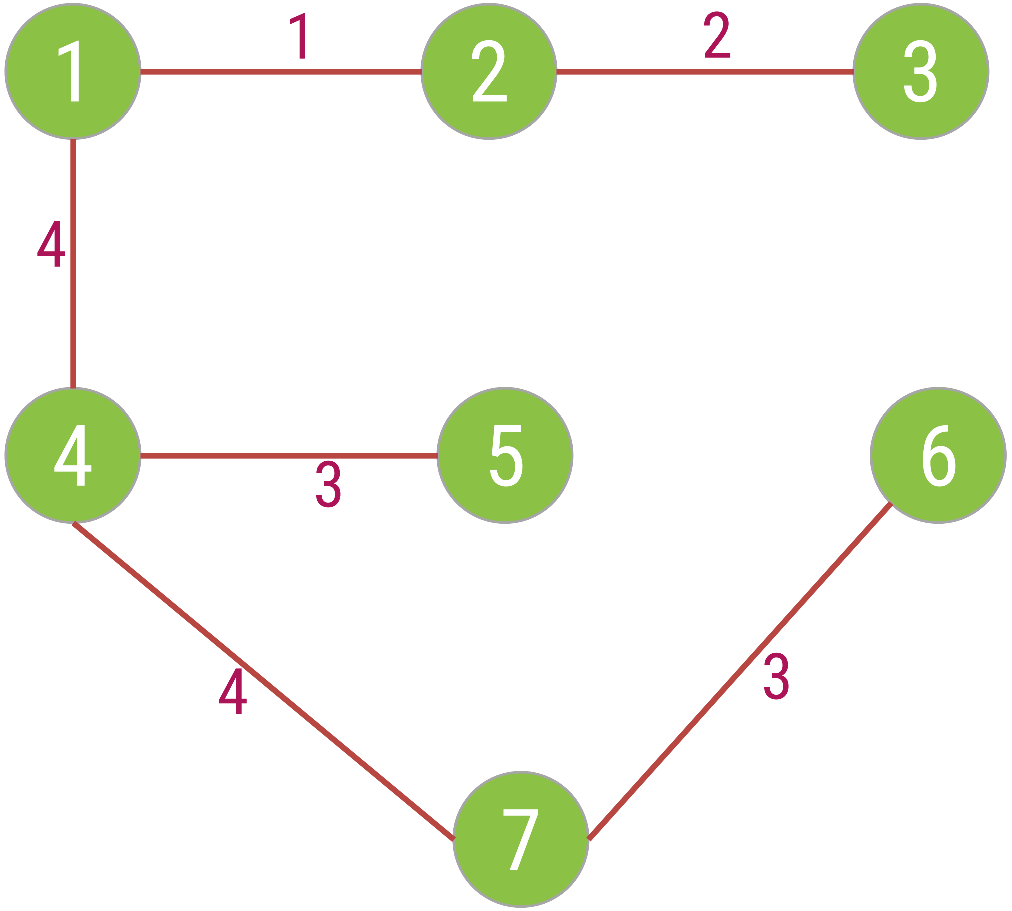


Step:2

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 – Example 2

Step	Edges considered - {u, v}	Connected Components

Edges	Weight

Total Cost = 17

Kruskal's Algorithm for MST

```
Function Kruskal(G = (N, A))
Sort A by increasing length
n ← the number of nodes in N
T ← ∅ {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

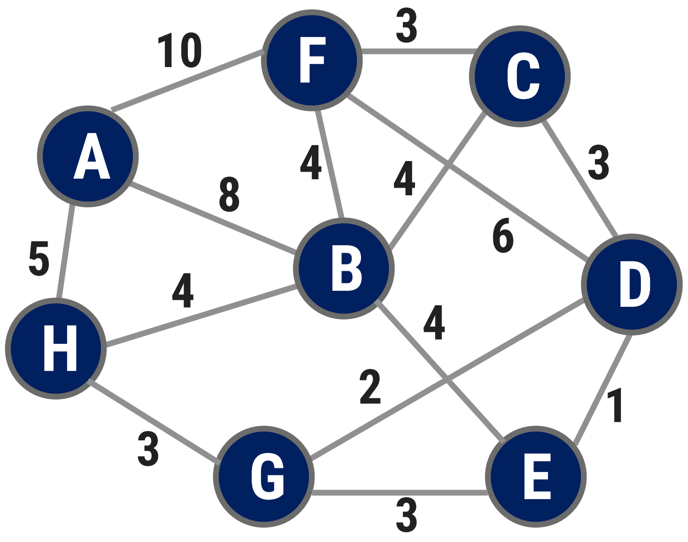
merge(ucomp, vcomp) is used to merge two connected components.

Exercises – Home Work

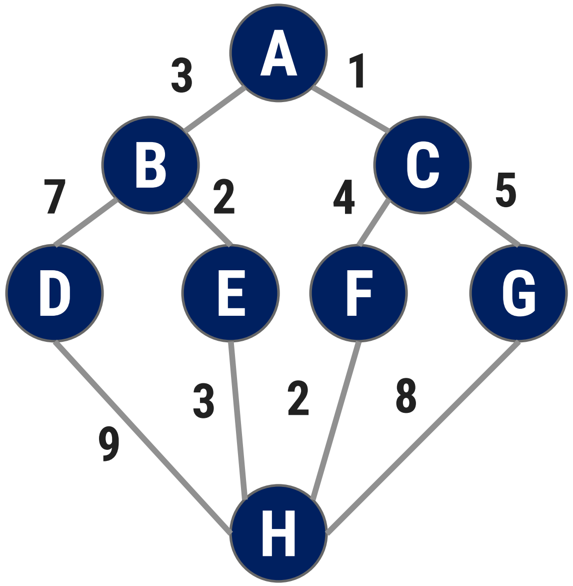
- The complexity for the Kruskal's algorithm is in $\boldsymbol{\theta(a \log n)}$ where $\boldsymbol{a}$ is total number of edges and $\boldsymbol{n}$ is the total number of nodes in the graph G.

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

1.



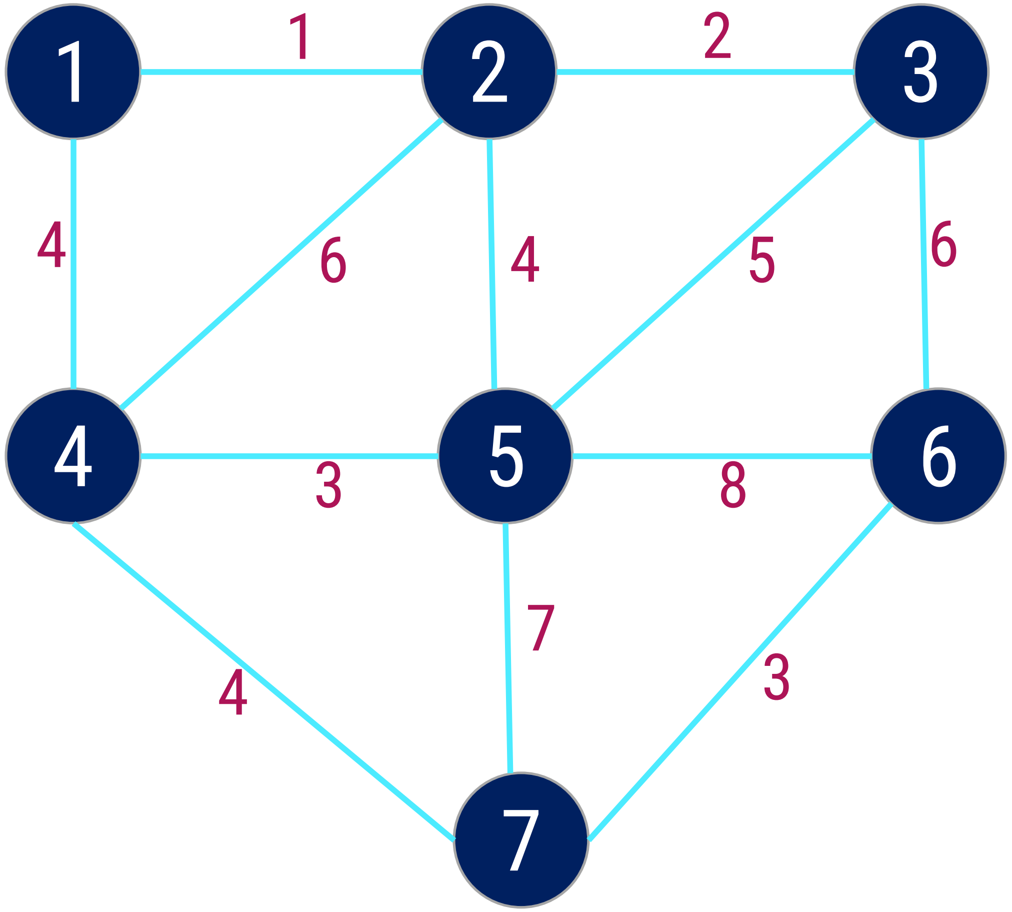
2.



Prim’s Algorithm

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

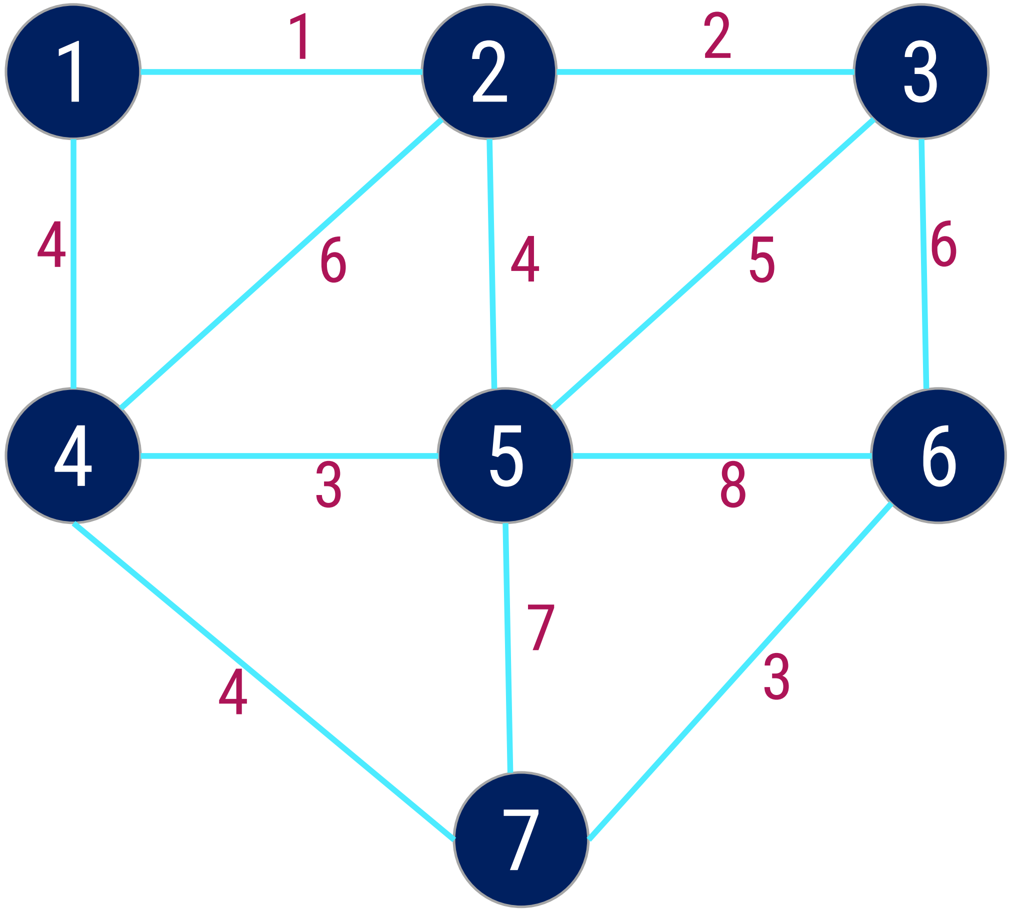
Prim's Algorithm for MST – Example 1



Step:1 Select an arbitrary node.

Node - Set B	Edges
1	

Prim’s Algorithm for MST – Example 1

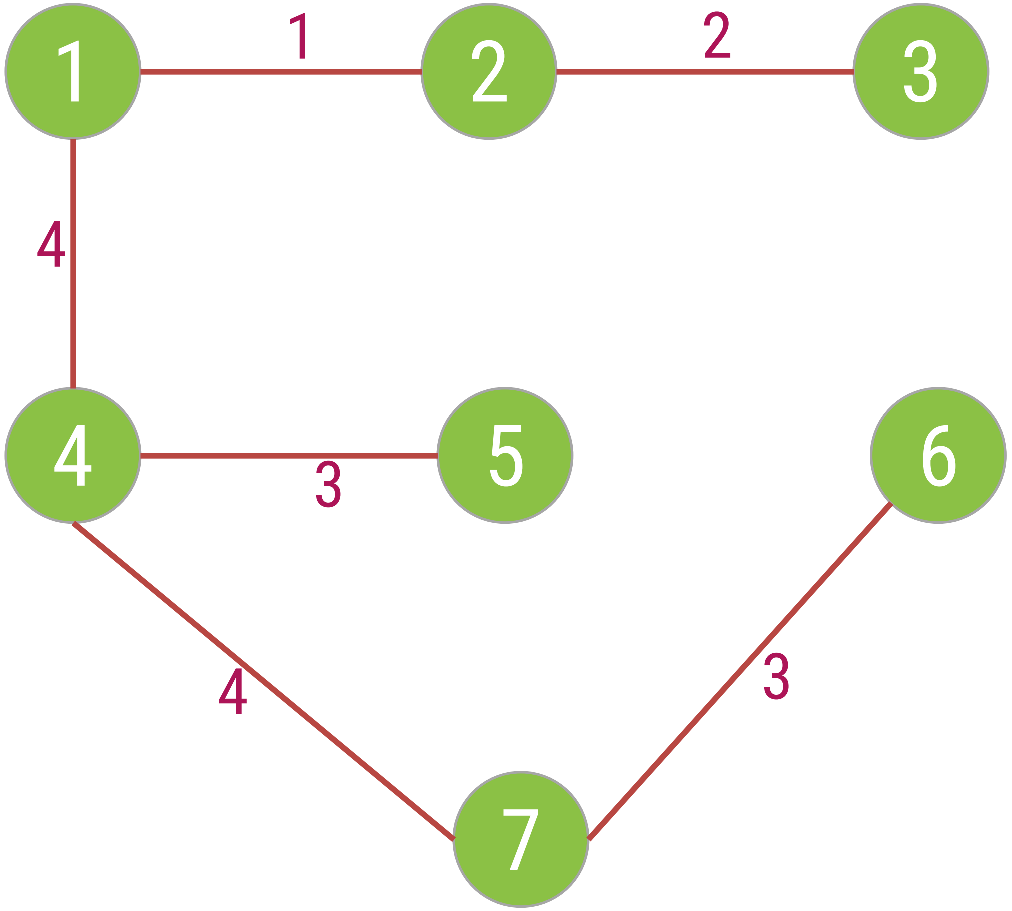


Step:2 Find an edge with minimum weight.

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



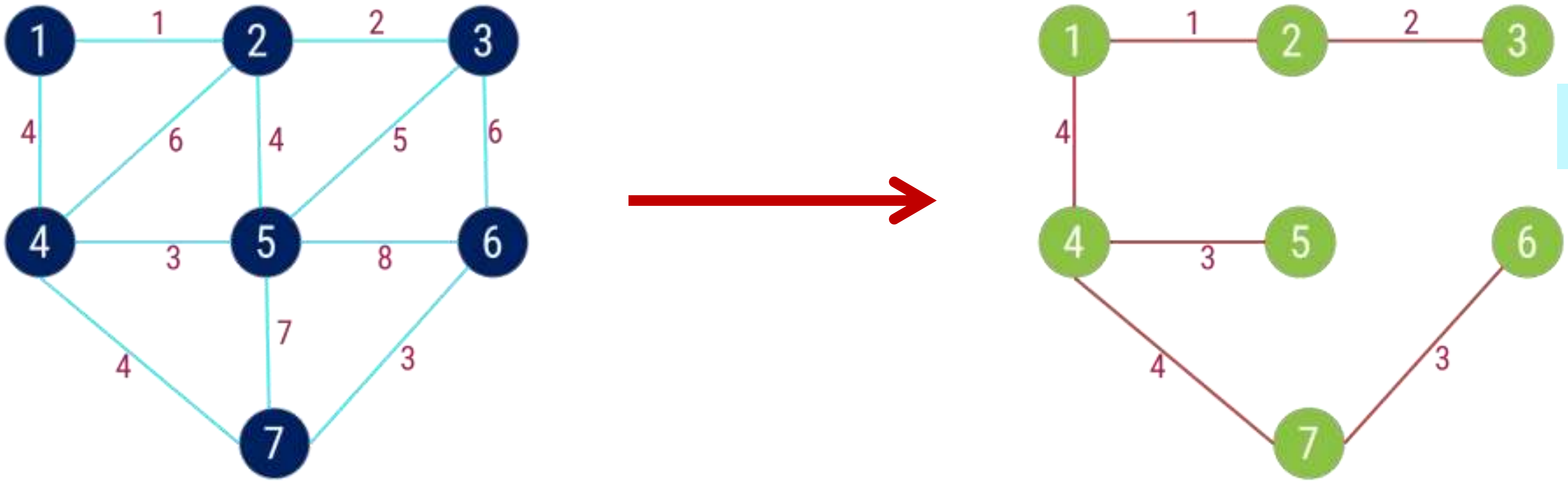
Step:3

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 – 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}**

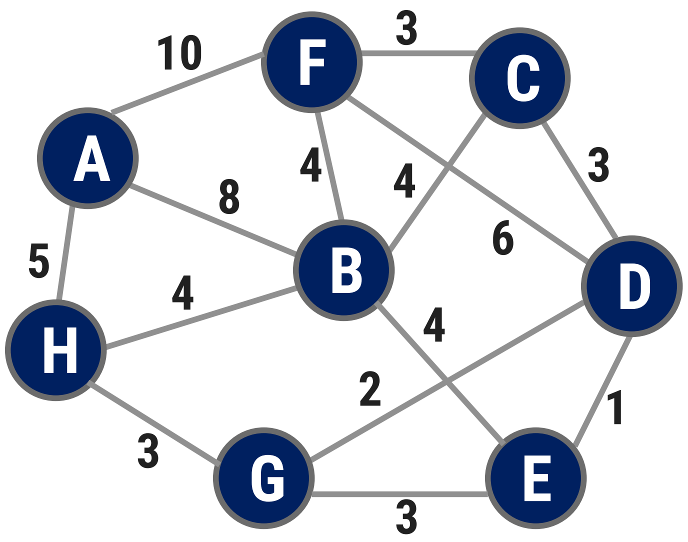
Prim's Algorithm

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

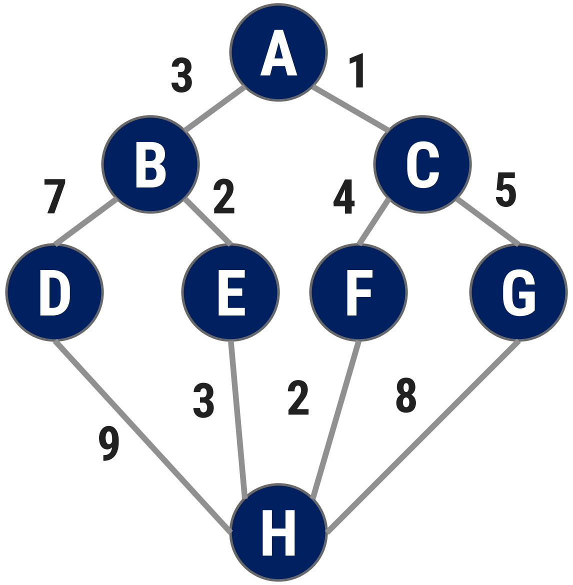
Exercises – Home Work

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

1.



2.



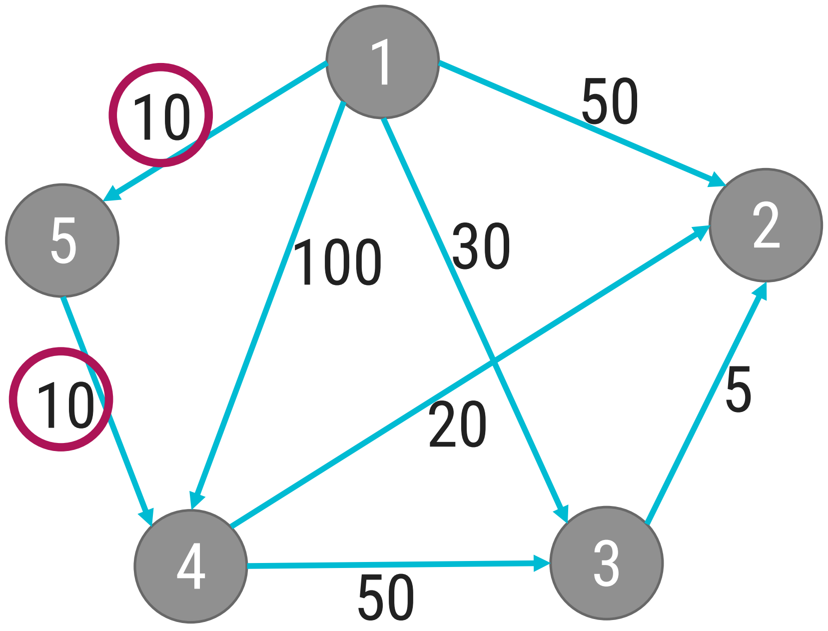
Single Source Shortest Path – Dijkstra's Algorithm

Introduction

- 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

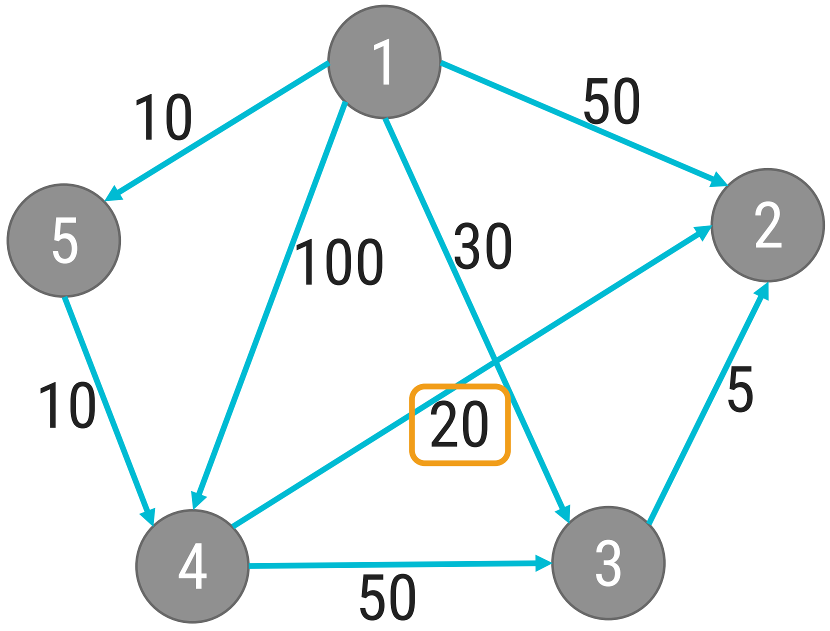
Step	v	C	2	3	4	5
Init.	-	{2, 3, 4, 5}	50	30	100	10
1	5	{2, 3, 4}	50	30	20	10

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

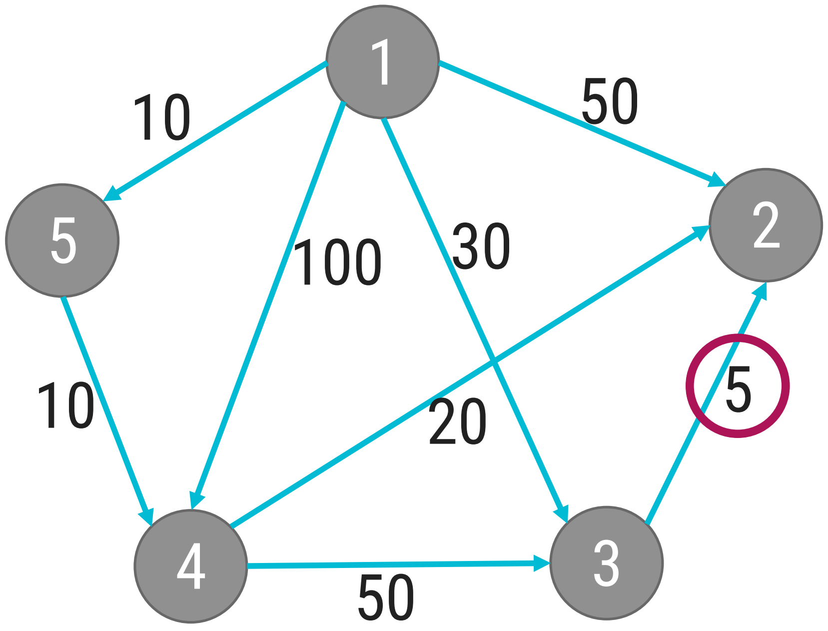
Step	v	C	2	3	4	5
Init.	-	{2, 3, 4, 5}	50	30	100	10
1	5	{2, 3, 4}	50	30	20	10
2	4	{2, 3}	40	30	20	10

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

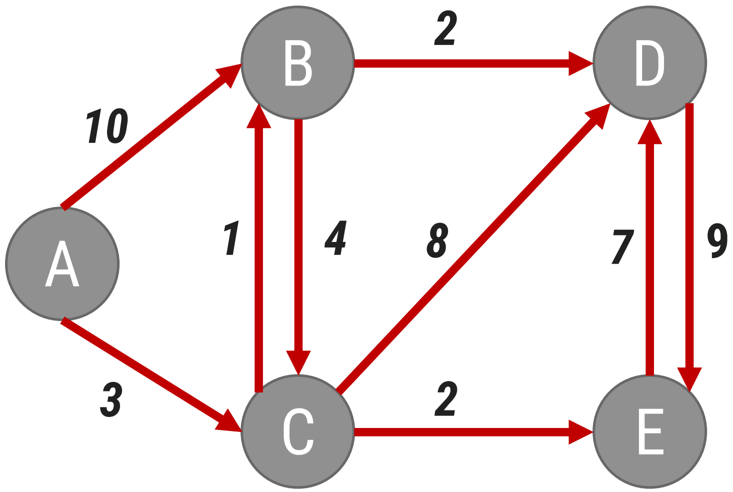
Step	v	C	2	3	4	5
Init.	-	{2, 3, 4, 5}	50	30	100	10
1	5	{2, 3, 4}	50	30	20	10
2	4	{2, 3}	40	30	20	10
3	3	{2}	35	30	20	10

Compare cost of 1–3–2 and 1–2

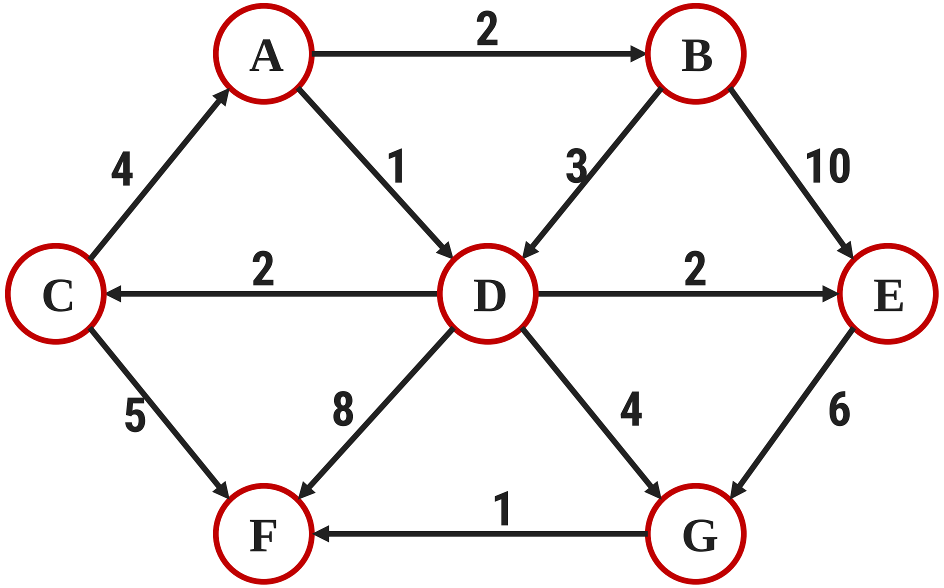
Exercises – Home Work

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

1.



2.

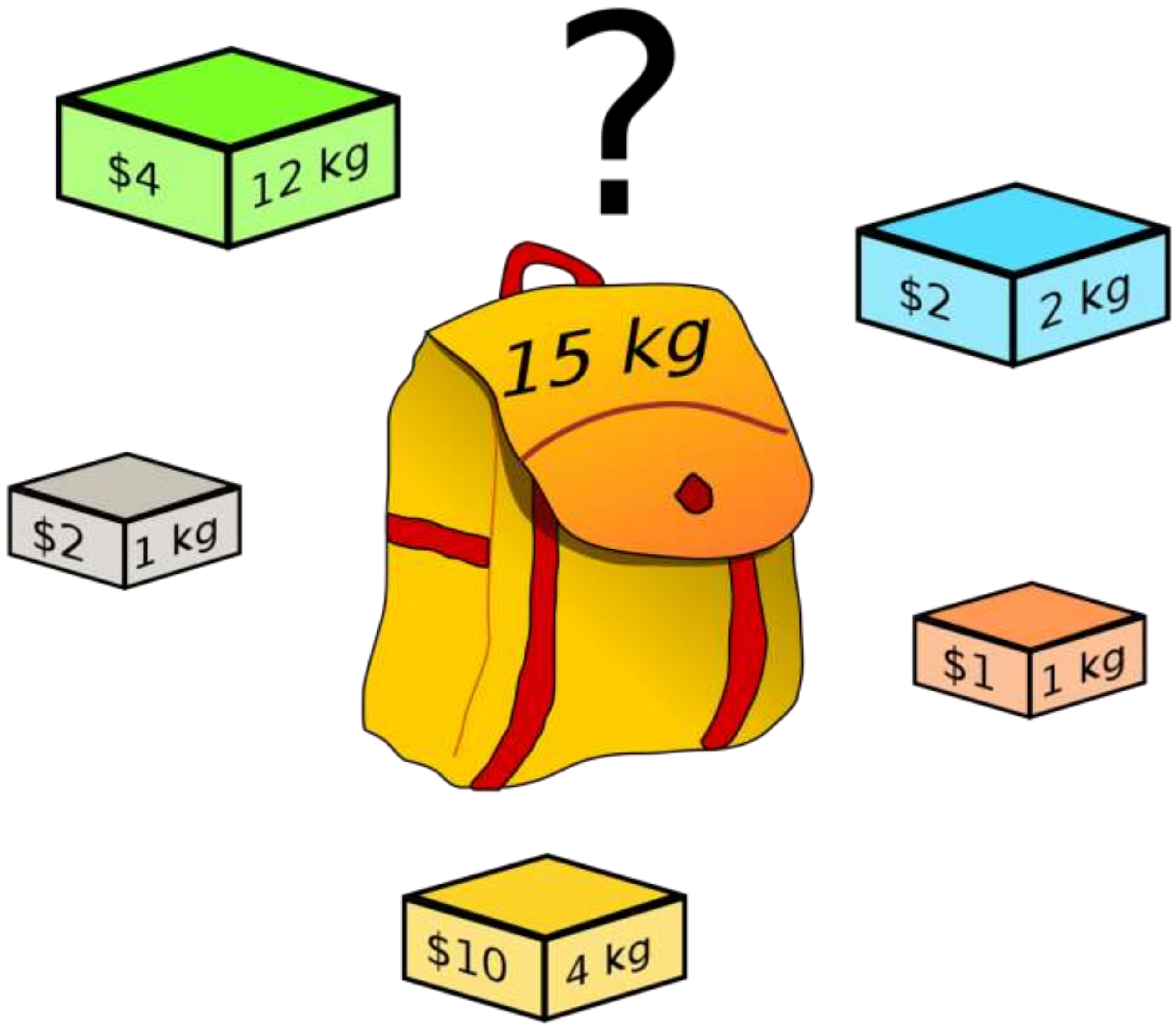


Dijkstra’s Algorithm

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

Knapsack Problem

Knapsack Problem



Fractional Knapsack Problem

Introduction

- 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 $\boldsymbol{i = 1, 2 \ldots n}$.
- 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**.
- So we may decide to carry only a fraction $\boldsymbol{x_i}$ of object $\boldsymbol{i}$, where $\boldsymbol{0 \leq x_i \leq 1}$.
- In this case, object $\boldsymbol{i}$ contribute $\boldsymbol{x_i w_i}$ to the total weight in the knapsack, and $\boldsymbol{x_i v_i}$ 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.

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

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

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
```

W = 100 and Current weight in knapsack= 60
Object weight = 50
The fraction of object to be included will be
(100 − 60) / 50 = 0.8

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.

Job Scheduling with Deadlines

Introduction

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

Job Scheduling with Deadlines - Example

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

Solution:

Step 1: Sort the jobs in **decreasing order** of their profit.

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

Job Scheduling with Deadlines - Example

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

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

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

P	1	2	3
Job selected	0	0	0

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

P	1	2	3
Job selected	0	0	J1

Job Scheduling with Deadlines - Example

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

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

P	1	2	3
Job selected	J2	0	J1

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

But position 1 is already occupied and two jobs can not be executed in parallel, so reject job 3

Job Scheduling with Deadlines - Example

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

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

P	1	2	3
Job selected	J2	J4	J1

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

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

Exercises – Home Work

1. Using greedy algorithm find an optimal schedule for following jobs with $\boldsymbol{n}$=4.
   Profits: (a, b, c, d) = (20,10,40,30) &
   Deadline: $(d_1, d_2, d_3, d_4) = (4, 1, 1, 1)$
2. Using greedy algorithm find an optimal schedule for following jobs with $\boldsymbol{n}$=5.
   Profits: (a, b, c, d, e) = (100,19,27,25,15) &
   Deadline: $(d_1, d_2, d_3, d_4, d_5) = (2, 1, 2, 1, 3)$

Job Scheduling with Deadlines - Algorithm

```
Algorithm: Job-Scheduling (P[1..n], D[1..n])
1. Sort all the n jobs in decreasing order of their profit.
2. Let total position P = min(n, max(d_i))
3. Each position 0, 1, 2…, P is in different set and T({i}) = i, for 0 ≤ i ≤
   P.
4. Find the set that contains d, let this set be K. if T(K) = 0 reject the
   job; otherwise:
   1. Assign the new job to position T(K).
   2. Find the set that contains T(K) – 1. Call this set L.
   3. Merge K and L. the value for this new set is the old value of T(L).
```

Huffman Codes

Prefix Code

- 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 code Introduction

- 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
  1. Build a Huffman Tree from input characters.
  2. Traverse the Huffman Tree and assign codes to characters.

Huffman Codes - Example

- Find the Huffman codes for the following characters.

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

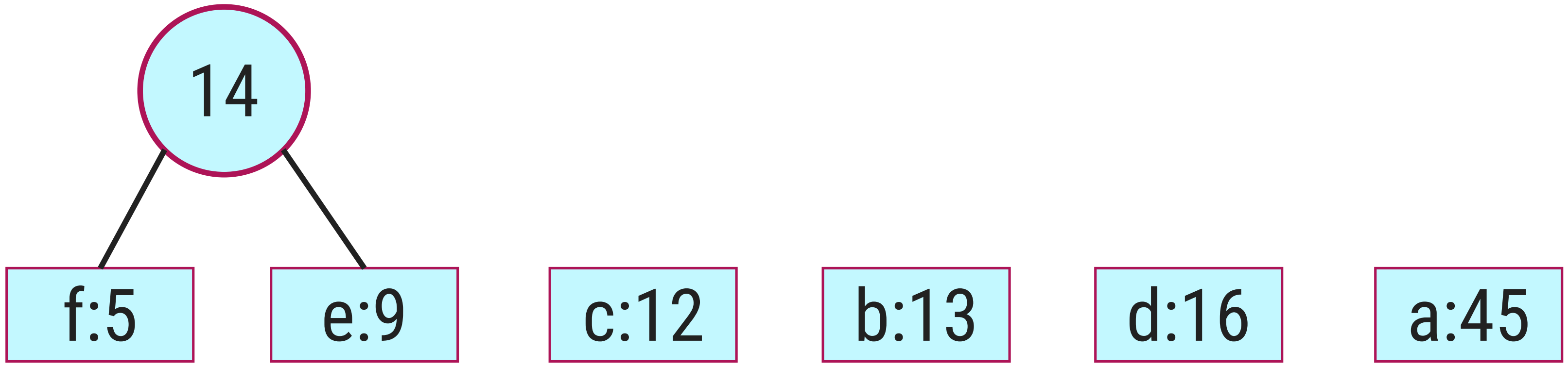
Step 1: Arrange the characters in the Ascending order of their frequency.

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

Huffman Codes - Example

Step 2:

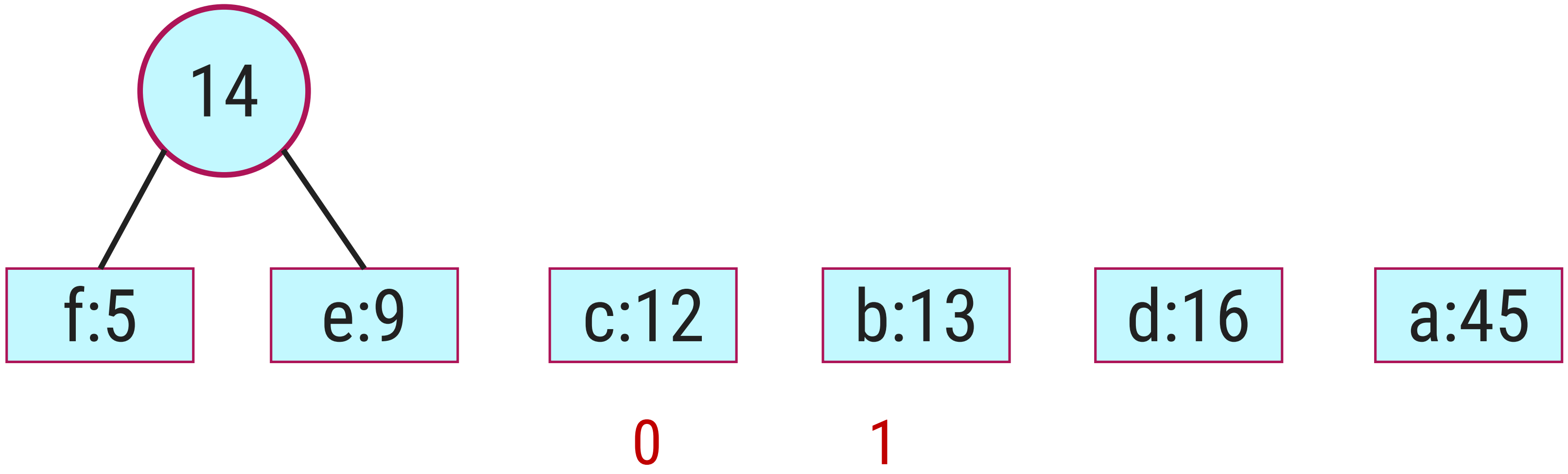
- ✓ Extract two nodes with the minimum frequency.
- ✓ Create a new internal node with frequency equal to the sum of the two nodes frequencies.
- ✓ Make the first extracted node as its left child and the other extracted node as its right child.



Huffman Codes - Example

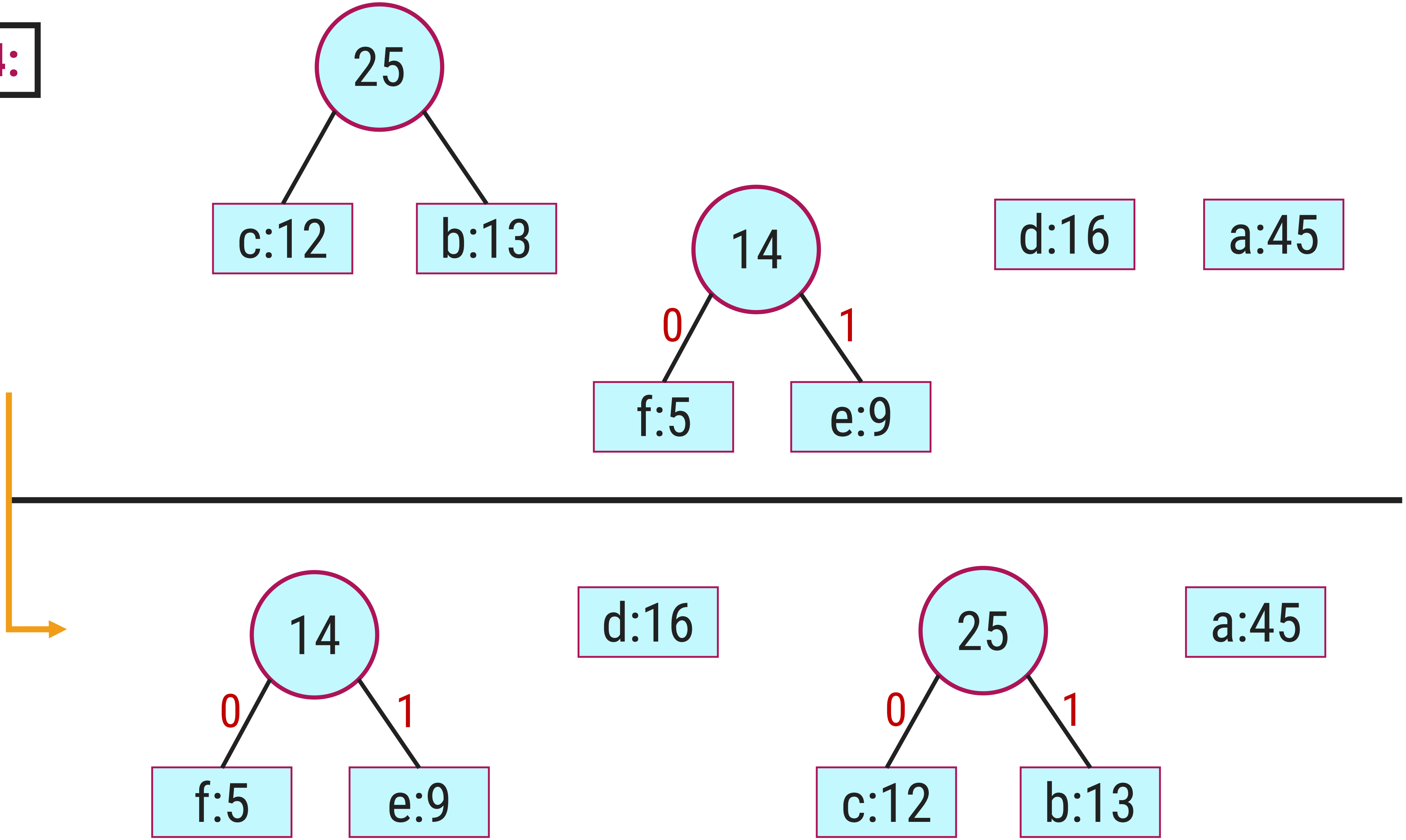
Step 3:

- ✓ 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.



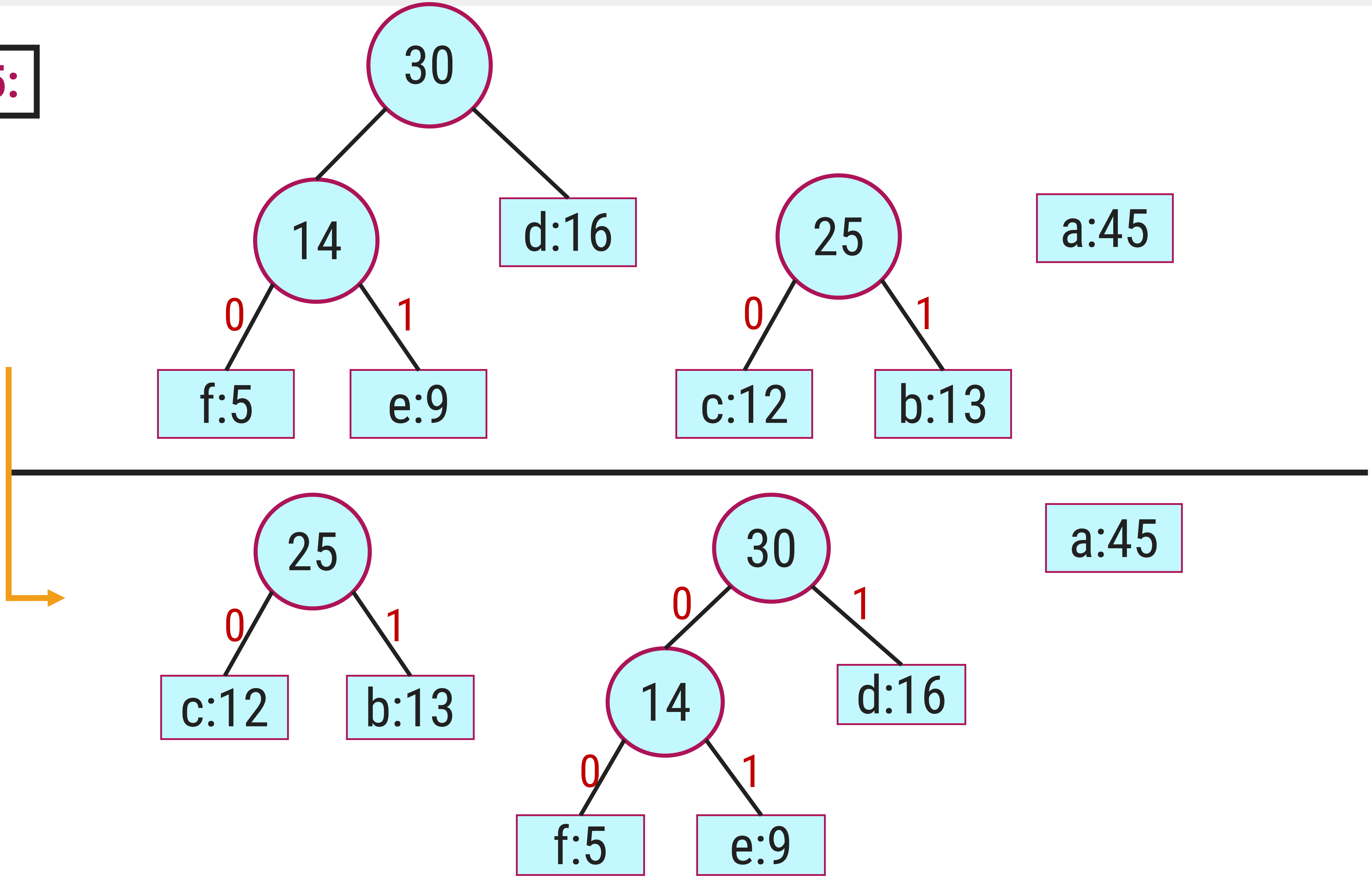
Huffman Codes - Example

Step 4:



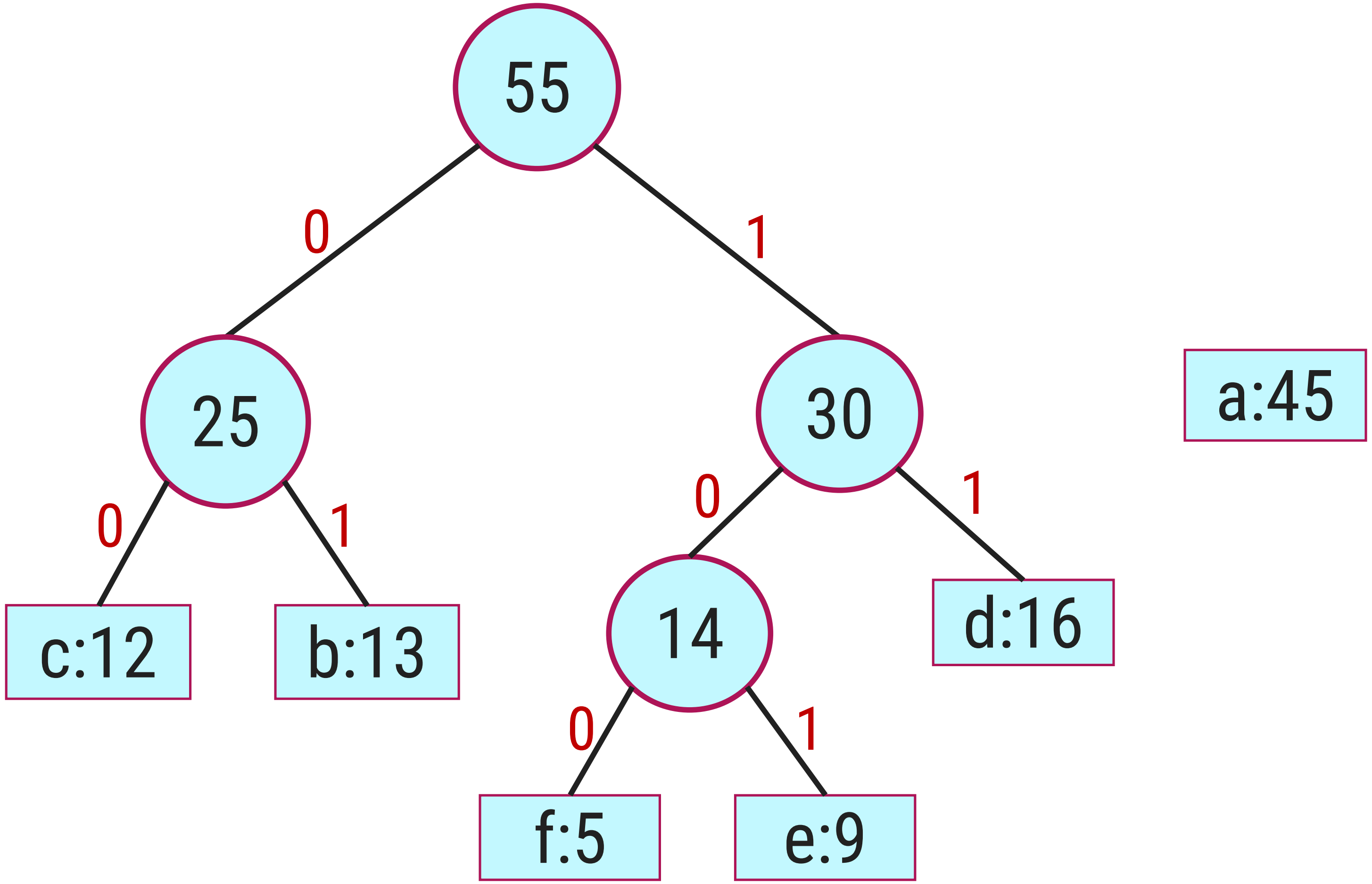
Huffman Codes - Example

Step 5:



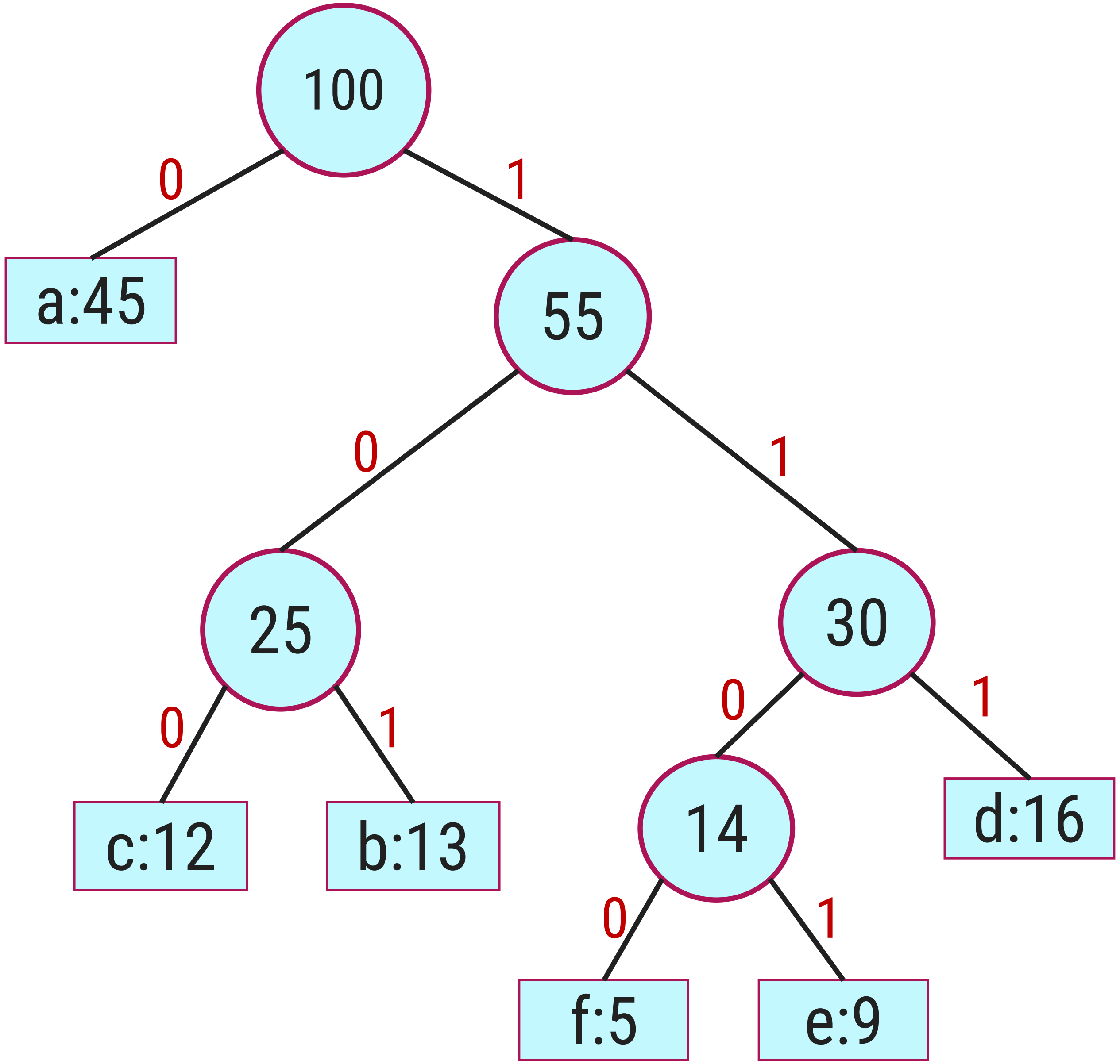
Huffman Codes - Example

Step 6:



Huffman Codes - Example

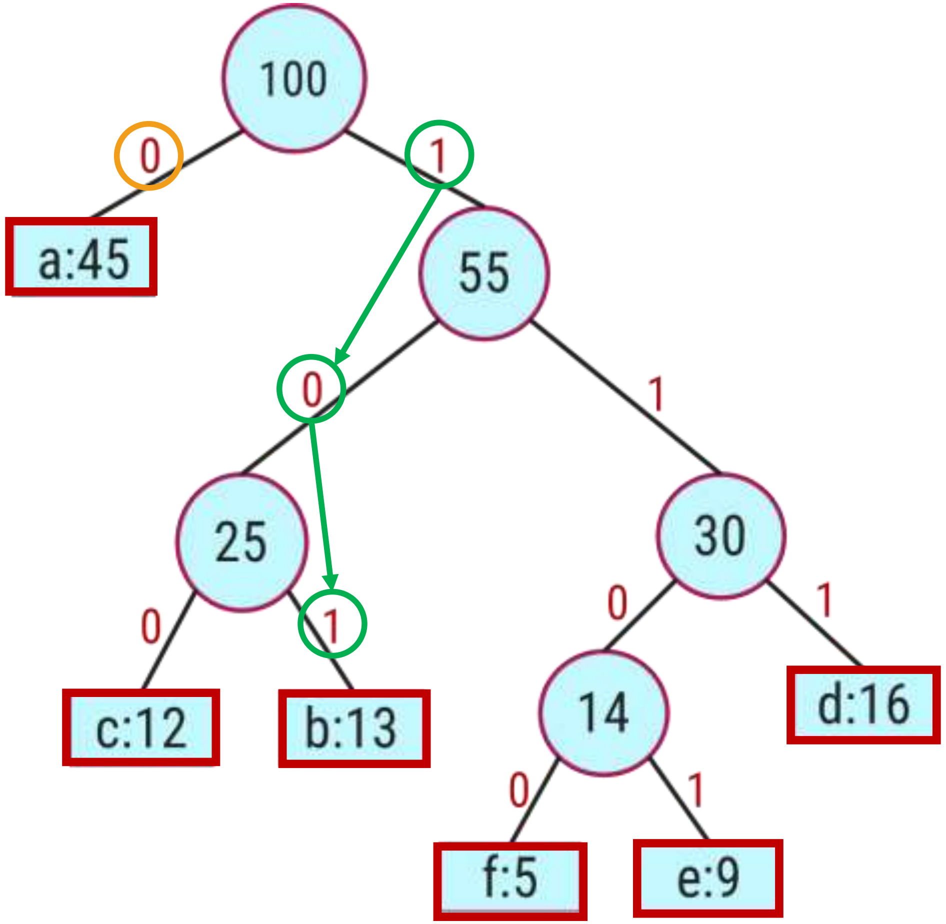
Step 7:



Huffman Codes - Example

Step 8:

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



Total bits: 224

Huffman Codes - Algorithm

```
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

Thank You!