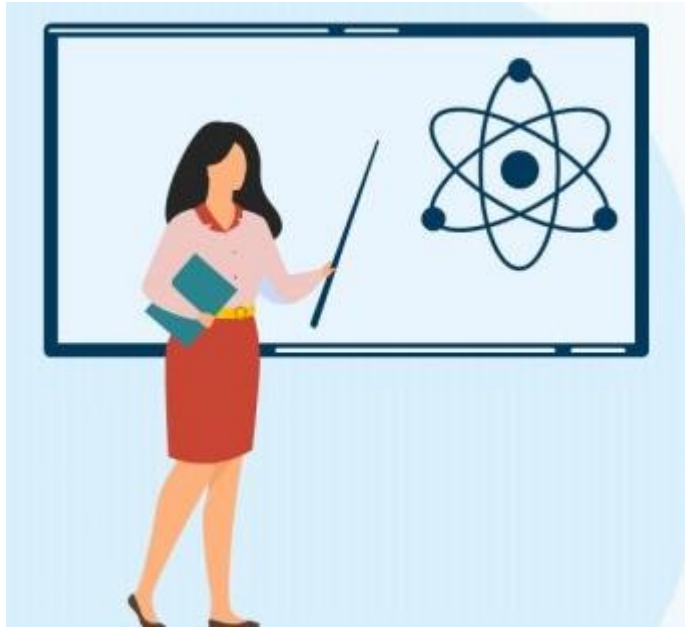
Greedy



Suman Pandey

Greedy Algorithms

- Greedy are used to solve **Optimization** Problems
- Problem that require **minimum** or **maximum** results
- Ex 1 : You travel from location A to location B

  A ⟶ B

  You can travel by walk, car, bus, train, flight
  but u need to cover the journey only in 12 hour. -> **constraints**
  Ans: Train, flight - **Feasible solutions**
  But the journey should cost you **minimum**
  Ans: Train – **Optimal solution**
- Ex 2: You want to buy a best car

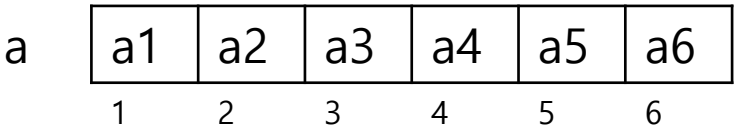
  Check features of all the car. - **constraints**
  We sort the brand first (BMW)
  Then you sort the top models - **Feasible solutions**
  Then you pick up latest car, or the well tested car - **Optimal solution**
- This approach is greedy approach.
- When you **follow the known method** for solving a problem and you quickly solve it that approach is called greedy
- We do not check all the possible solution, rather go with one approach (sometimes based on intuition) and follow that.

Greedy Algorithm

```
Algorithm Greedy ( a , n)
{
        for i =1 to n do
        {
                x = Select (a); //select the input one by one
                if feasible(x) then //if it is feasible, include it in the output
                {
                        solution = solution + x ;
                }
        }

}
```

n=5



- Greedy says that a problem should be solved in stages
- In each stage we consider the input and if the input is **feasible** we will include that input
- If we follow this procedure we will get the optimal solution

Greedy Algorithms

- Both Dynamic Programming and Greedy are used to solve **Optimization** Problems
- Greedy
  - Greedy deals with forming the solution step by step by choosing the **local optimum** at each step and **finally reaching a global optimum**.
  - Greedy does not deal with multiple possible solutions, its just builds the one solution that it believes to be correct
  - Greedy believes that choosing local optimum at each stage will lead to form the global optimum
  - We often require **preprocessing the input** or using an **appropriate data structure** (often **a priority queue**). This way we often can make greedy choices quickly, thus yielding an efficient algorithm
- DP
  - DP does not deal with such uncertain assumptions
  - DP finds a solution to all subproblems and chooses the best ones to form the global optimum
  - DP **guarantees** the correct answer each and every time whereas Greedy is not.
  - DP is much **slower** than Greedy, Greedy deals with only one subproblem, however DP deals with all the subproblems.
  - DP works only when there is overlapping subproblems.

Greedy Algorithms

- Fractional Knapsack problem
- Scheduling unit-time tasks with deadlines
- Huffman Coding – Data Compression
  - Optimal Merge Pattern
  - Huffman Coding
- Minimum Spanning Tree (Chapter 23)
  - Prims and Kruskal's
- Dijkstra Algorithm (Chapter 24)

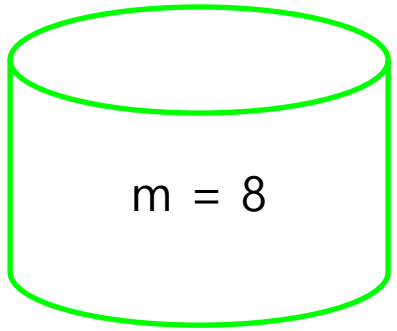
Note: It is a good idea to read Chapter 16, 23 and 24 together.

Fractional Knapsack Problem

There are 4 objects
n = 4

For each object you have some profit and weight
P = {1, 2, 5, 6}
W = {2, 3, 4, 5}

There is a bag of capacity 8
m = 8



Max $\sum_o^i p\ x$

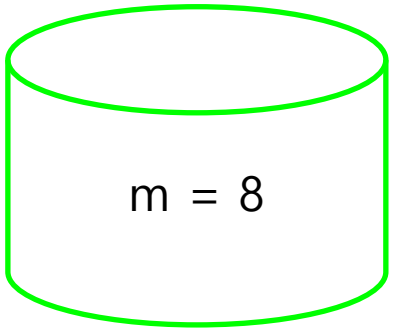
Min $\sum_o^i w\ x \leq m$

Goal is to fill the bag such that, total profit is **maximized**
Output x = **{ 0~1 , 0~1, 0~1, 0~1 }** which means **we can take fractions** of the items

Fractional Knapsack Problem

n = 4
m = 8

For each object you have some profit and weight
P = {1, 2, 5, 6}
W ={2, 3, 4, 5}
x = **{ 0~1 , 0~1, 0~1, 0~1 }** -> output



There could be many possible solution to this
1. To maximize the profit, one can say include item 4 first, as its profit is highest
2. To maximize the profit, one can say include smaller item first, so we can have more items and more profit
3. But the best way will be to have items that have highest **profit / weight**

P/W ={1/2, 2/3, 5/4, 6/5} -> { 0.5, 0.66, 1.25, 1.2 }

Now we can say that select item 3 first and per kg its profit is highest
then select item 4, but you can take only 4 kg of item 4.
Total profit - > 5 + 1.2 * 4 = 9.8
x = **{0, 0, 1, 4/5 }**

How we solved this problem?
There are several approaches, but we decided to go with **profit/weight** app roach.
So in **Greedy** we **first decide our approach** and then **go about followi ng that approach.**

Knapsack Problem (Greedy)

For each object you have some profit and weight

O_{id}= {1, 2, 3, 4, 5, 6, 7}

P = {10, 5, 15, 7, 6, 18, 3}

W = {2, 3, 5, 7, 1, 4, 1}

Fractional Knapsack Vs 0/1 Knapsack

Thief must select the items to maximize its profit.
What is the maximum profit he can get, in following scenarios?

1. He can take fractions of the items
2. He can not take fractions of the items
3. He can take one item multiple times

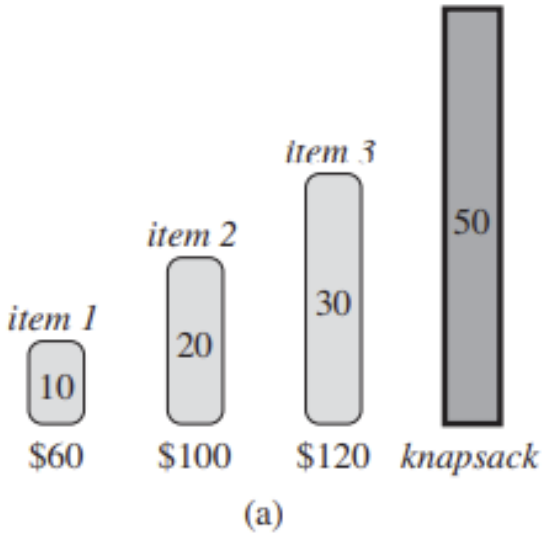


Figure 16.2 An example showing that the greedy strategy does not work for the 0-1 knapsack problem. **(a)** The thief must select a subset of the three items shown whose weight must not exceed 50 pounds. **(b)** The optimal subset includes items 2 and 3. Any solution with item 1 is suboptimal, even though item 1 has the greatest value per pound. **(c)** For the fractional knapsack problem, taking the items in order of greatest value per pound yields an optimal solution.

scheduling unit-time tasks with deadlines

n=5

Tasks	T1	T2	T3	T4	T5
Earning	20	15	10	5	1
Deadlines	2	2	1	3	3

Each printing task will take 1 unit of time
And it can wait for their Deadlines respectively
So which job should be done first.

We want set of those jobs, that can be completed within their deadline in a way that earnings can be maximized – Optimization problem

scheduling unit-time tasks with deadlines

n=5

Tasks	T1	T2	T3	T4	T5
Earning	20	15	10	5	1
Deadlines	2	2	1	3	3

Each task will take 1 unit of time
And it can wait for their Deadlines respectively
So which job should be done first?

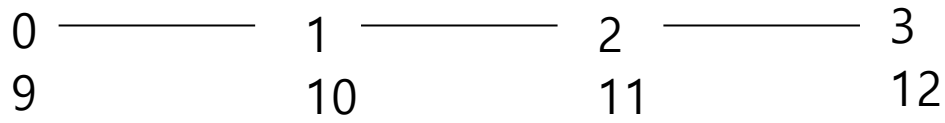
Problem : We want set of those jobs, that can be completed within their deadline in a way that earnings can be maximized

This is an optimization problem (maximize the earning)
Constraints – deadlines to finish the job

We can solve this problem using Greedy method

None of the jobs have deadline beyond 3 hours
Hence the maximum number of jobs should fini sh within 3 hours.

Time slots



Each job takes **1 unit** of time, hence only 3 jobs can be done

Out of 5 he has to **select just 3 job**, so that prof it can be maximized

Obvious choice will be to select the job that giv es **highest earning first**. - **Greedy**

scheduling unit-time tasks with deadlines

n=5

Tasks	T1	T2	T3	T4	T5
Earning	20	15	10	5	1
Deadlines	2	2	1	3	3
			X		X

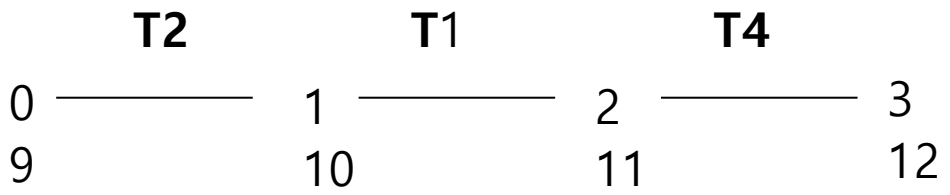
None of the jobs have deadline beyond 3 hours
Hence the maximum number of jobs should fini sh within 3 hours.

Each task takes **1 unit** of time, hence only 3 task s can be done

Out of 5 he has to **select just 3 task**, so that pr ofit can be maximized

Obvious choice will be to select the task that gi ves **highest earning first**. - **Greedy**

Time slots



Select the task that gives **highest earning first - T**1
T1 can be wait for 2 hours, so we can put it in 2nd hour slot

Select the task that gives next highest earnings – **T2**
T2 can also wait for 2 hours, we do have one slot withi n 2 hours left, so put the T2 in that slot.

Select next task that gives the highest earning – **T3**
T3 can wait for 1 hour, can we do this task ?
Ans: no the first hour is already occupied with higher e arning tasks. So we reject T3.

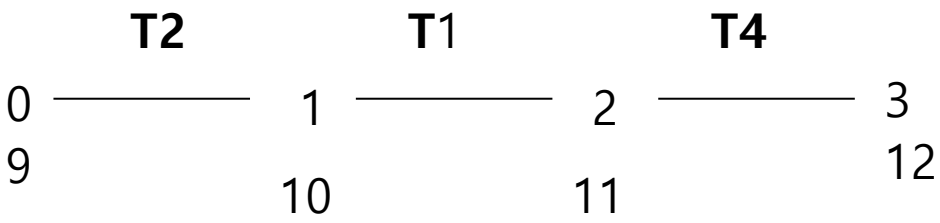
Select next task that gives highest earning – **T4**
T4 can wait for 3 hours, can we do this task ?
Ans: Yes, the 3rd slot is available .

scheduling unit-time tasks with deadlines

n=5

Tasks	T1	T2	T3	T4	T5
Earning	20	15	10	5	1
Deadlines	2	2	1	3	3
			X		X

Time slots



We can do Tasks { T2, T1, T4 }
In these two particular sequences
{T2, T1, T4 }
or
{T1, T2, T4 }
Total Earnings – 20 +15 + 5 = 40

Tasks Assigned	Slot assigned	Solution	Earning
1	[1,2]	T1	20
2	[0,1], [1,2]	T1, T2	20 + 15
3	[0,1], [1,2]	T1, T2	20 + 15
4	[0,1], [1,2] [2,3]	T1, T2, T4	20 + 15 +5
5	**[0,1], [1,2] [2,3]**	**T1, T2, T4**	**20 + 15 +5**

scheduling unit-time tasks with deadlines

n=5

Tasks	T1	T2	T3	T4	T5	T6	T7
Earning	35	30	25	20	15	12	5
Deadlines	3	4	4	2	3	1	2

Complete this table
1. Find the jobs
2. Find the sequence in which the job should be done
3. Maximum earning

Tasks Assigned	Slot assigned	Solution	Earning

With Added Parameter

n=5

Tasks	T1	T2	T3	T4	T5
Earning	20	15	10	5	1
Deadlines	2	2	1	3	3
Time taken	2	1	2	1	1

Challenge !!!!
Can you come up with a approach to solve this question
With Greedy ?

Optimal Merge Pattern

This algorithm will be used in Huffman Coding

What is merging?
What is Optimal Merge pattern ?

Problem : merge two or more list in a way that cost of merging can be minimized.
Minimization -> Optimization problem

Optimal Merge Pattern

What is merging ?

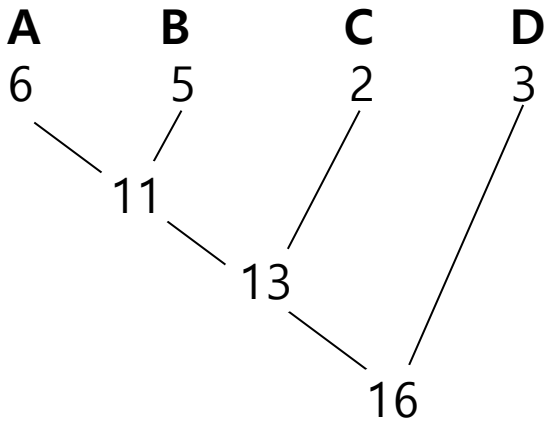
A	B	C
3	5	3
8	9	5
12	11	8
20	16	9
m=4	n=4	11
		12
		16
		20

O(m+n)

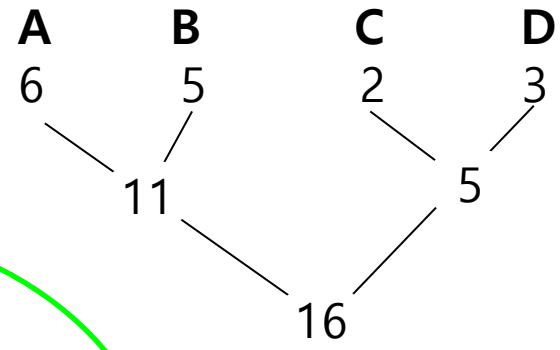
A more complicated problem is when you have more than two l ists to merge

List ->	A	B	C	D
Sizes ->	6	5	2	3

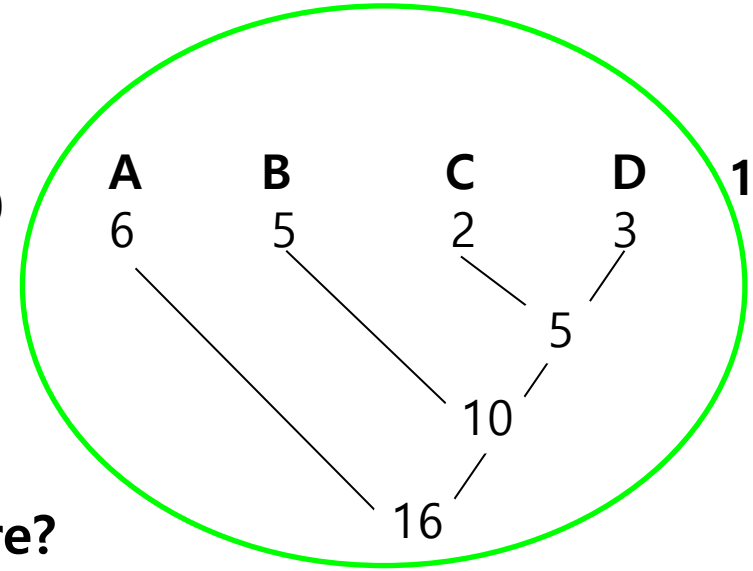
How to go about two-way merging



11 + 13 + 16 = 40



11 + 16 + 5 = 32



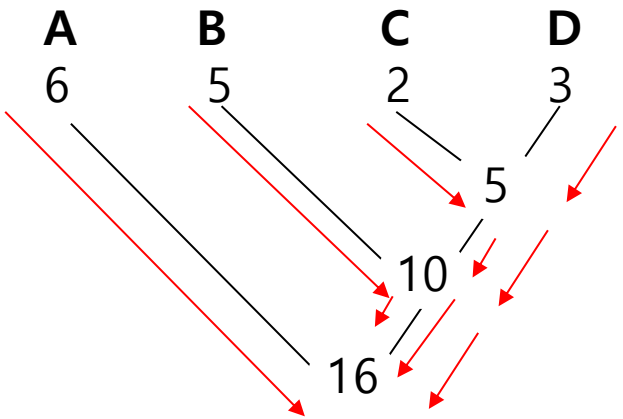
16 + 10 + 5 = 31

Which pattern is giving the best result?
Ans: Patter where we are merging the sma llest lists first.

What is the Greedy method we followed here?
Ans: We merge the smallest list first

Optimal Merge Pattern

List -> A B C D
Sizes -> 6 5 2 3
How to go about two-way merging

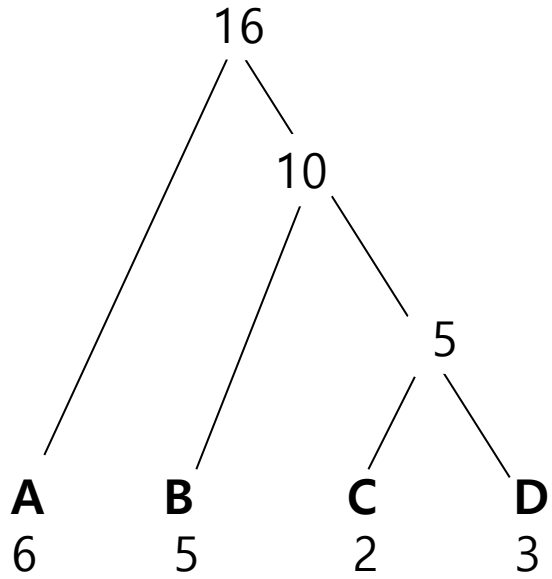


Adding of the internal nodes
16 + 10 + 5 = 31

Finding how many times a list is merged
A is merged one time
B is merged two time
C is merged three time
D is merged three times

6 * 1 + 5 * 2 + 2 * 3 + 3 * 3 = 31

∑ di * xi



Optimal Merge Pattern

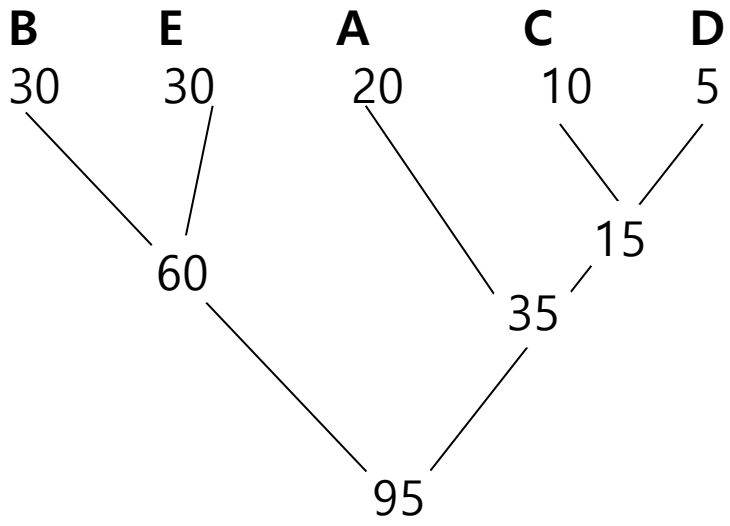
List ->	A	B	C	D	E
Sizes ->	20	30	10	5	30

Q. What will be the minimum cost of merging these lists ?

Optimal Merge Pattern

List ->	A	B	C	D	E
Sizes ->	20	30	10	5	30

Q. What will be the minimum cost of merging these lists ?



60 +95 + 35 + 15 = 205

Huffman Coding

Huffman Coding
Message – BCCABBDDAECCBBAEDDCC

Huffman coding is a compression technique

Cost of sending normal msg without encoding
Cost of sending for fixed-size encoding
Cost of sending variable-size encoding

1010000010101010101010101010010101011101-**160 bits**
101000001010101010101010101-**115 bits**
10100000101010101010 -**94 bits**

Completely Compressed data msg
Encoded data msg

Problem : Send the msg in a way that cost of sending the msg can be minimized - > Optimization Problem

Fixed Size Coding

Message – BCCABBDDAECCBBAEDDCC

This is your msg that is needed to be sent over the net work

What will be the cost of sending this msg?
There are 20 letters in this msg
Length – 20
ASCII code of each (128) character is 8bit

Char	decimal	Binary
A	65	01000001
B	66	01000010
C	67	….
D	68	….

For a msg of 20 length
Total bits – 8 X 20 – 160 bits -> without encoding

Fixed size encoding
Can I use my own codes, instead of 8-bit Bina ry codes?
Ans: Yes, We are not using all the alphabets on the keyboards**.** We are only using a few characte rs. These few characters can be represented with a smaller number of bits. 8 bits are for 128 char acters, but we have only 5 character

Char	codes
A	000
B	001
C	010
D	011
E	100

1 bit - 0 / 1 - 2 char can be represented

2 bit - 01 , 10, 00, 11 – 4 char

3 bit - 000, 001, 010, 100, 011, 110, 111, 101 – 8 char

Fixed Size Coding

Message – BCCABBDDAECCBBAEDDCC
Length of msg - 20

Char	codes
A	000
B	001
C	010
D	011
E	100

Using our own code can reduce the size of the message
What will be the size of the new msg ?
Ans: Size of the msg – 3 X 20 = 60 bits

When the msg reaches the receiver, can the receiver decode this msg?
Ans: No, the Receiver doesn't know 000 corresponds to A.
So, with msg, we need to send a table, that will tell the receiver about each new code and which alphabets it corresponds to.

All computers can understand the original ASCII code for each charact er. Hence we need to send 8 bit ASCII code for each character A, B, C, D, E

8 X 5 = 40 bits

Apart from that, we will also have to send the mapping of each ASCII to our newly generated code

3 X 5 = 15 bit

Total size of the msg – 60 bits + 40 bits + 15 bits = 115 bits
This is still smaller than the original size of msg – 160 bits

Huffman Coding

Message – BCCABBDDAECCBBAEDDCC

Huffman says that we don't need to send the fixed size co de for each character.
Rather we can send a variable size code based on the ch aracter frequency in the msg.

Small size code for frequently appearing character and bigger size code for less frequent character

Hence, Arrange the alphabets in the increasing order of th eir count

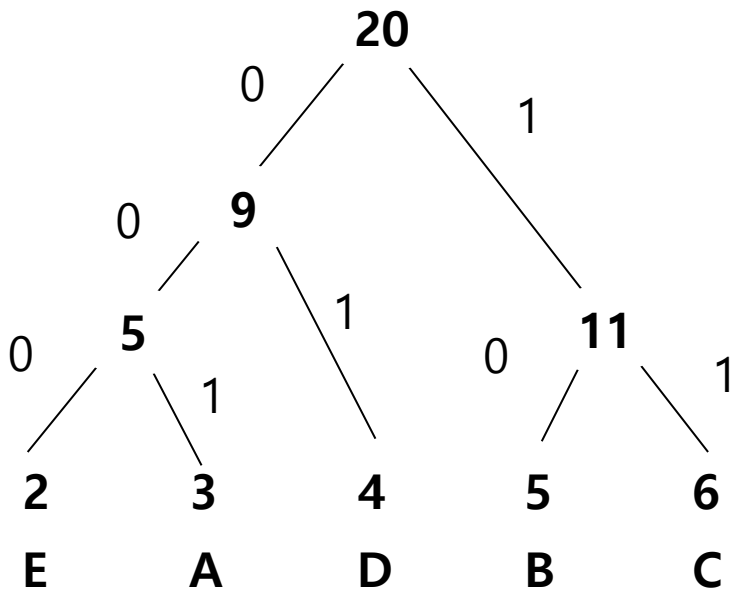
2	3	4	5	6
E	A	D	B	C

utilize **Optimal merger patter to assign code–** **Greedy**
Left-hand side edges 0, right hand side edges 1

Char	frequency	codes
A	3	001
B	5	10
C	6	11
D	4	01
E	2	000

20

Variable size code



Huffman Coding

Message – BCCABBDDAECCBBAEDDCC

Variable size code

Char	frequency	codes	Msg bits
A	3	001	3x3 = 9
B	5	10	5x2 = 10
C	6	11	6x2 = 12
D	4	01	4x2 = 8
E	2	000	3x2 = 6

5 X 8 = 40 bits **12 bits** **45 bits**

1) Sort the items based on the frequency

2) utilize Optimal merger patter to assign code– Greedy

What is an Optimal merger pattern?

Ans: Always select two minimum one first and merge it to mak e a tree.

Why do we use the Optimal merge pattern?

Ans: It helps us define the code. Smaller code for frequent item s. Bigger codes for less frequent items.

Remember that the highest value had the minimum distance fr om the root of the tree, and the smallest had the highest dista nce.

How do we assign code based on the OMP ?

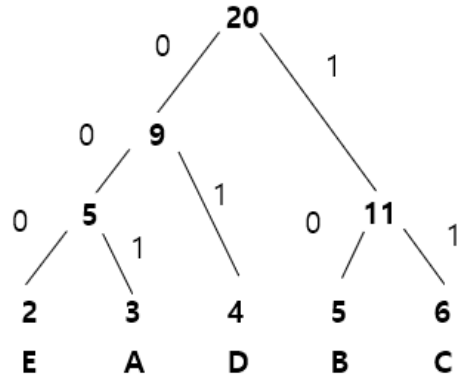
Ans: The left-hand side edges 0, right hand side edges 1

What will be the size of the msg?

Ans: 45 bit

What will be the total size of msg with tree?

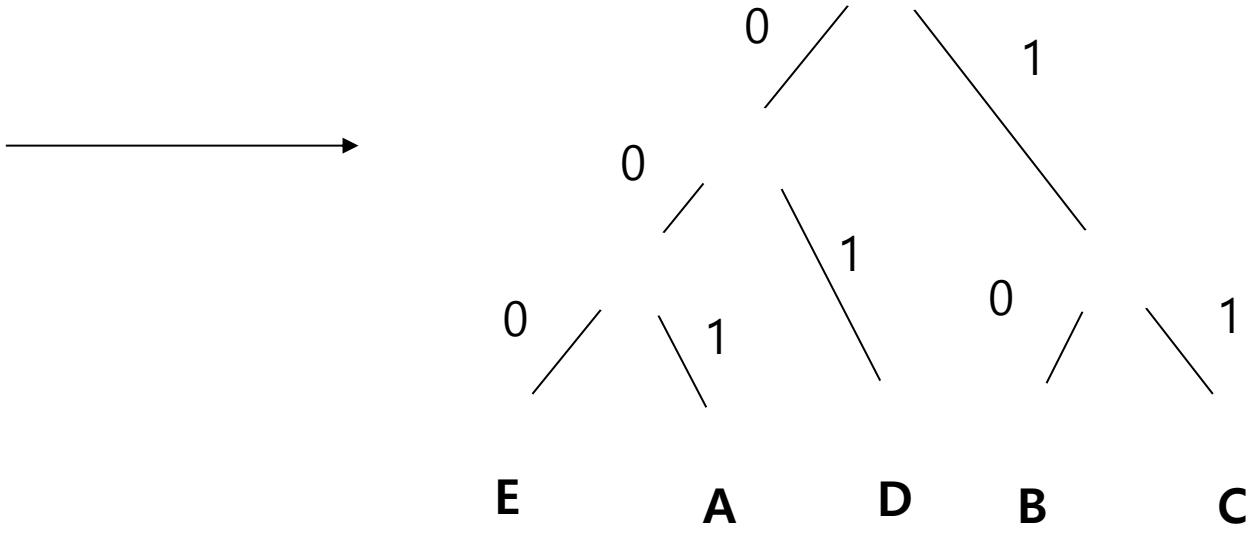
Ans: 40 + 12 + 45 = 97 bits



Msg – 45 bits
Tree/Table – 52 bits
---
97 bits < fixed size code 115 bits < no encoding 160 bits

If you have the Table you can construct the tree

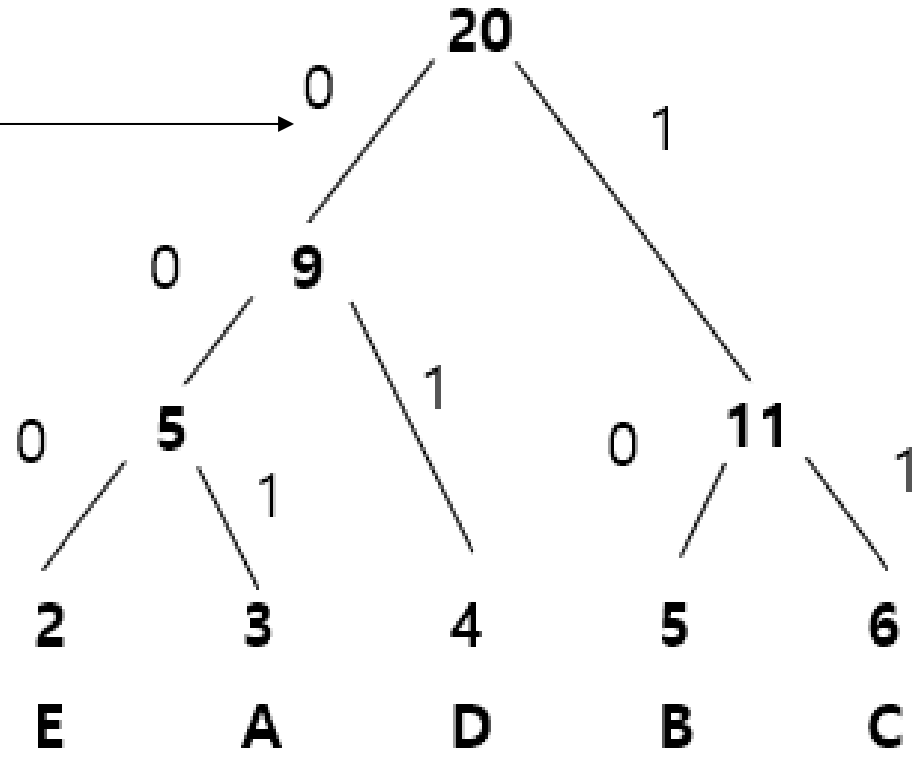
Char	codes
A	001
B	10
C	11
D	01
E	000



How Decoding is done at the Client side

Message – BCCABBDDAECCBBAEDDCC
101111001101001…….

For decoding, we must have the tree with the msg.
Msg - > 101111001101001 + Tree ⟶



Start from root of tree
1 - > right
0 -> left -> u got **B**

Again start from the root
1 -> right
1 -> right -> u got **C**

Again start from root
1 -> right
1 -> right -> u gor **C**
…..
End the end you will have the msg **BCCABBDDAECCBBAEDDCC**

Huffman Coding Questions

Minimize the cost of msg through Huffman coding

Build a Huffman Tree from input characters.

Traverse the Huffman Tree and assign codes to characters

What will be the cost of the msg ?

What is the difference of the Huffman coding and no encoding msg?

Freq	**5**	**9**	**12**	**13**	**16**	**45**
Char	**A**	**B**	**C**	**D**	**E**	**F**

Variable size code

Char	frequency	Msg bits	Msg bits
A	5		
B	9		
C	12		
D	13		
E	16		
F	45		

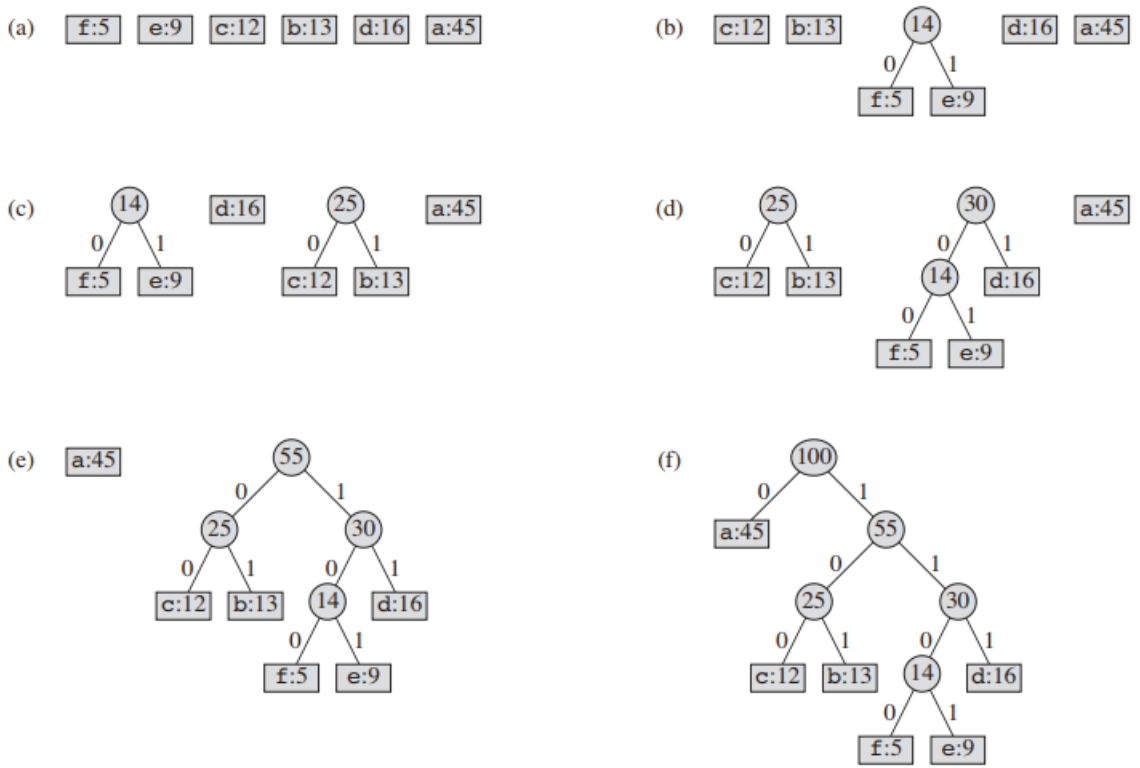
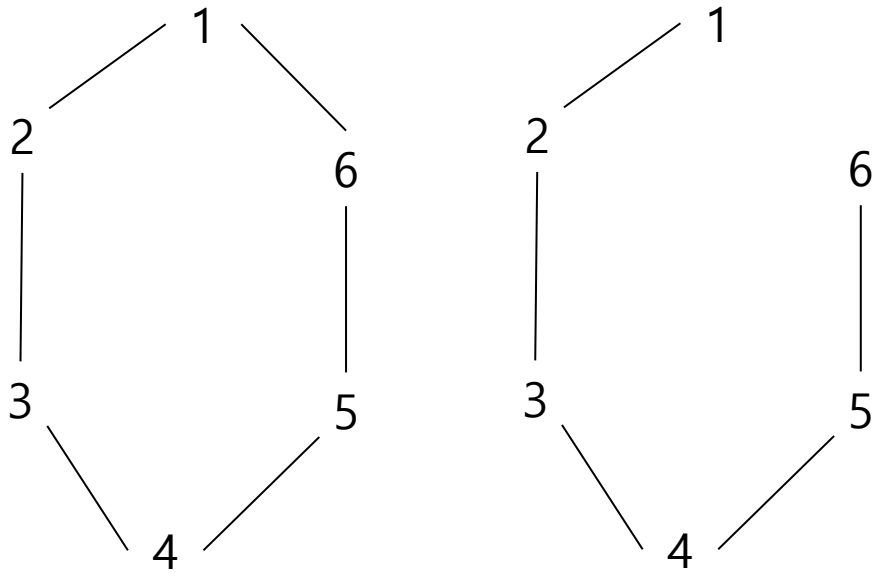


Figure 16.5 The steps of Huffman's algorithm for the frequencies given in Figure 16.3. Each part shows the contents of the queue sorted into increasing order by frequency. At each step, the two trees with lowest frequencies are merged. Leaves are shown as rectangles containing a character and its frequency. Internal nodes are shown as circles containing the sum of the frequencies of their children. An edge connecting an internal node with its children is labeled 0 if it is an edge to a left child and 1 if it is an edge to a right child. The codeword for a letter is the sequence of labels on the edges connecting the root to the leaf for that letter. **(a)** The initial set of $n = 6$ nodes, one for each letter. **(b)–(e)** Intermediate stages. **(f)** The final tree.

How to improve time complexity of Huffman

- Using minheap

Minimum Cost Spanning Tree



What is a minimum spanning tree?
The Minimum Spanning Tree is **the one whose cumulative ed ge weights have the smallest value**

What is a spanning tree?
How many spanning trees is possible for a given graph?
What is the minimum cost spanning tree ?
Greedy method to find minimum cost spanning tree.

1) **Prims**
2) **Kruskals**

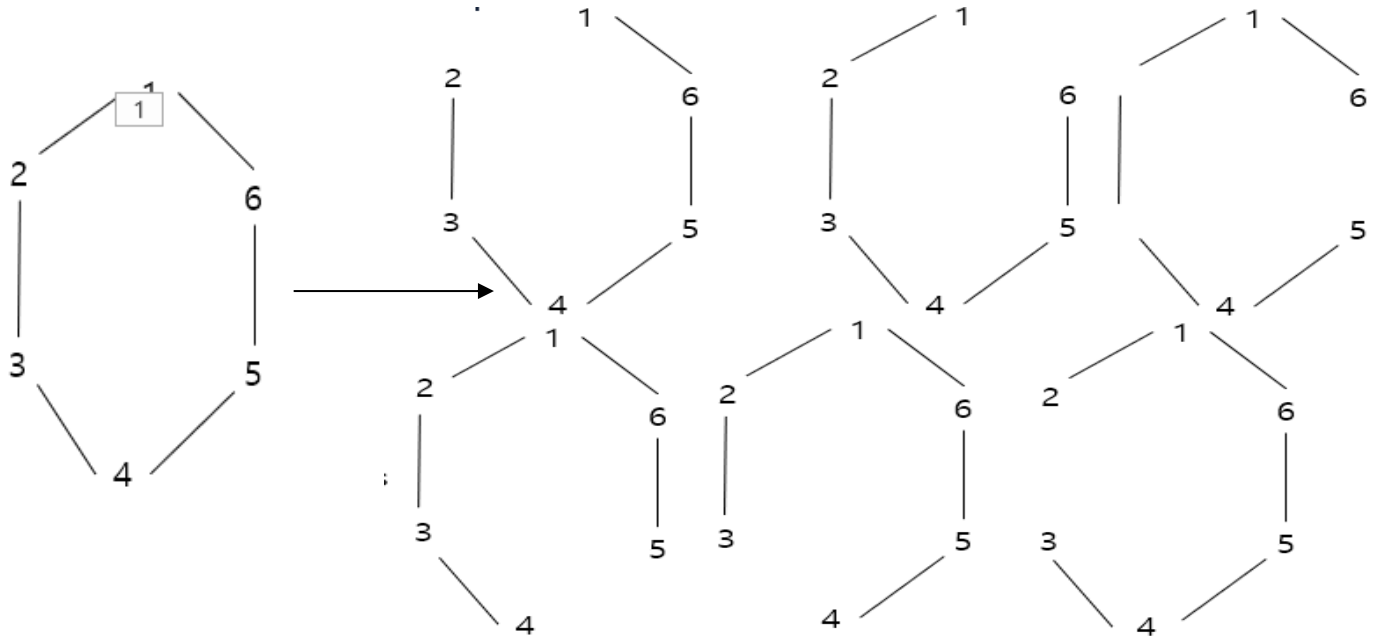
What is Spanning Tree

G = (V, E)
V = { 1, 2, 3, 4, 5, 6 }
E = {(1,2) , (2, 3) , (3, 4), (4, 5) , (5,6), (6, 1)}

What is a spanning tree?
Ans: Spanning tree is a subset of the gra ph, having all the vertices but only a subs et of edges.
The spanning-tree does not have any cycl e (loops).

By definition Spanning Tree
S ⊆ G
S = (V' E')
V' = V -> 6
E' = |V| -1 -> 5

How many ways you can make a spanning tree?

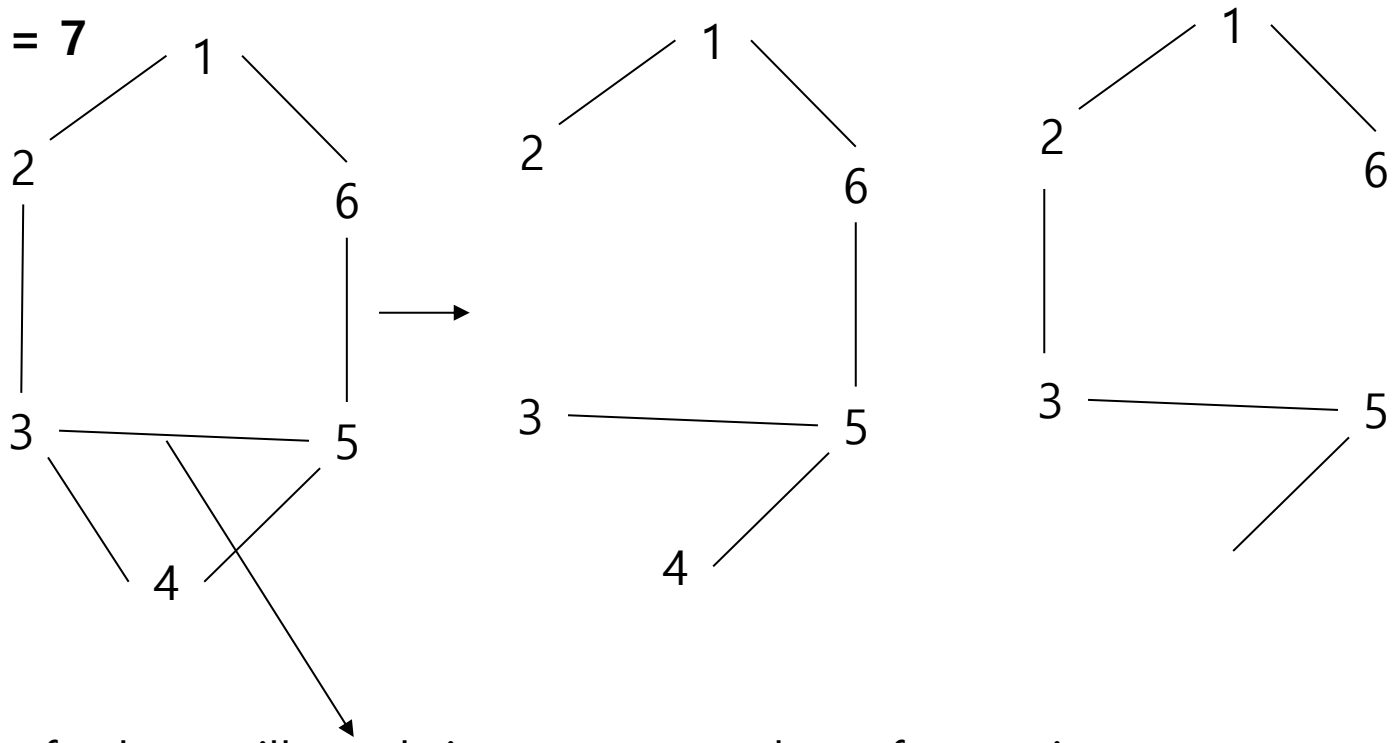


How many Spanning Tree

G = (V, E)
V = { 1, 2, 3, 4, 5, 6 } = 6
E = {(1,2) , (2, 3) , (3, 4), (4, 5) , (5,6), (3,5)} **= 7**

By definition Spanning Tree
S ⊆ G
S = (V' E')
V' = V -> 6
E' = |V| -1 -> 5

Now, how many ways you can make a spanning tree?



Higher number of edges will result into more number of spanning trees.

Minimum cost spanning tree

This is a weighted graph

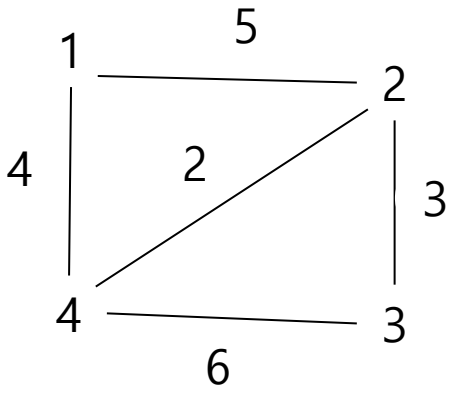
Lets make a spanning tree for this graph

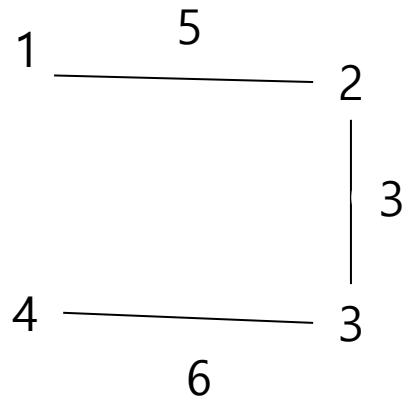
How many Edges are in spanning tree?

Ans: V-1 = 4-1 = 3

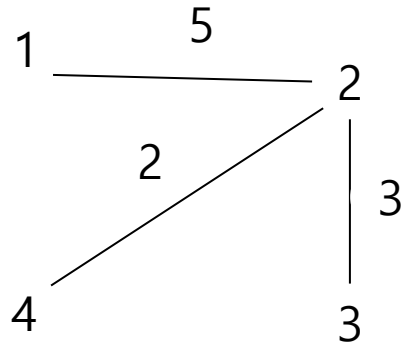
How many spanning trees are possible?

Ans: ${}_5C_{4-1} - 2 = 10 - 2 = 8$

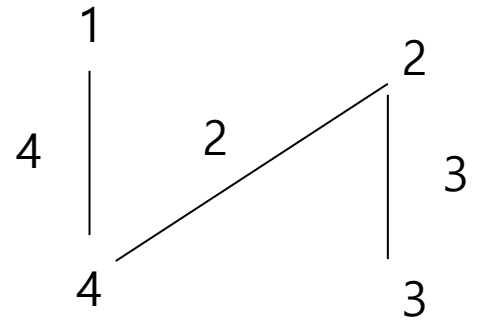




Cost of this Spanning Tree
= 5 +3 + 6 = 1?



Cost of this Spanning Tree
= 5 +2 + 3 = 10



Cost of this Spanning Tree
= 4+2 + 3 = 9

Can I find out the minimum cost spanning tree ?

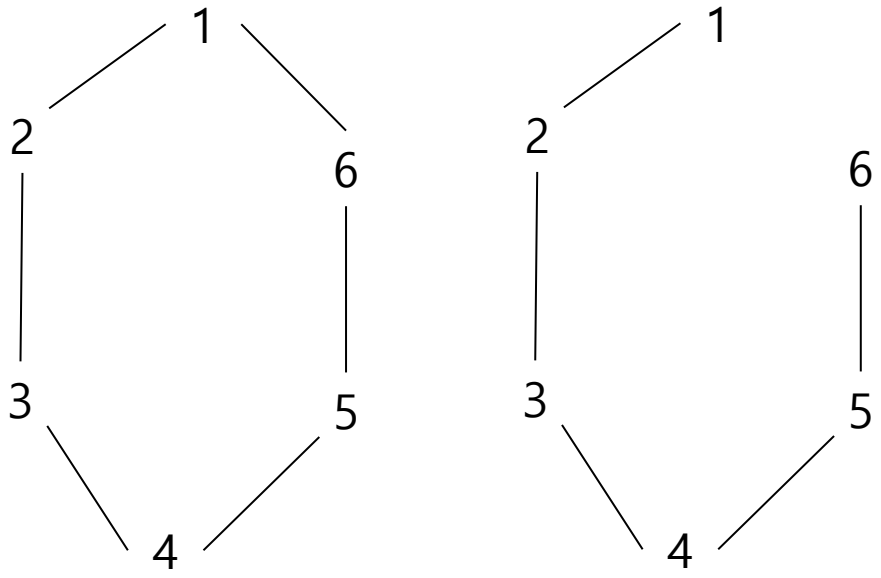
Ans: Yes you can, you can try out all possible spanning trees and choose minimum from all

Is there any Greedy method ?

Ans: Yes, Greedy methods are available. It won't require you to find all the possible spanning trees

1) Prims Algorithm 2) Kruskals Algorithm

Minimum Cost Spanning Tree
Prim's

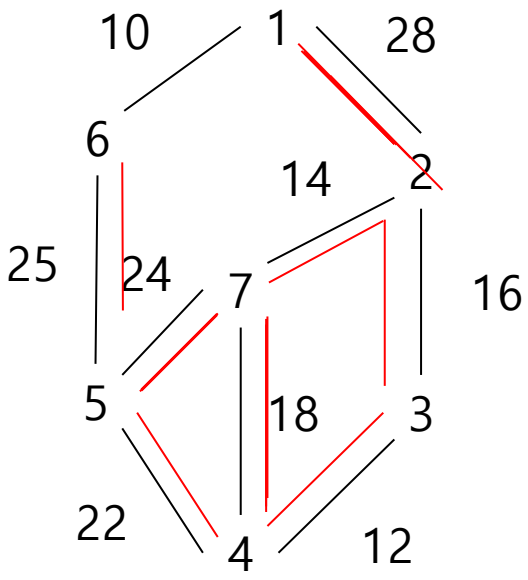


Greedy method to find minimum cost spanning tree.

1) **Prim's**
2) **Kruskal's**

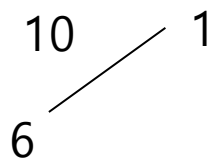
Minimum Cost Spanning Tree Prim's

We have a weighted graph, we have to find MCS following Prims

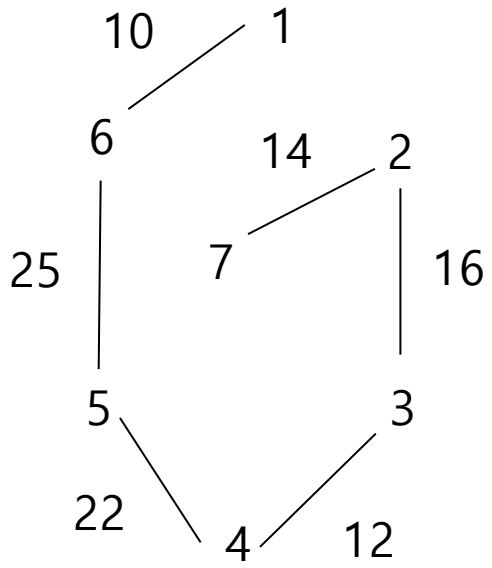


Prim's says

1) **First , select the minimum cost edge**



2) **Next, select the next minimum cost edge in a way that is connected to already selected vertices**
   Maintain the tree

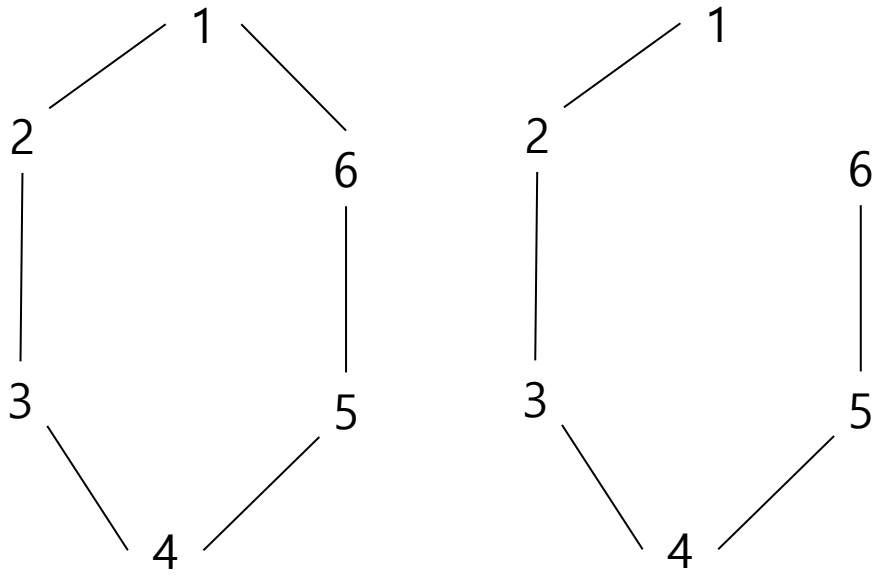


Total cost: 10 + 25 + 22 + 12 + 16 + 14 = 99

First select the minimum
Then always select the next connected minimum from the already selected vertices.

- this is Greedy

Minimum Cost Spanning Tree
Kruskal's

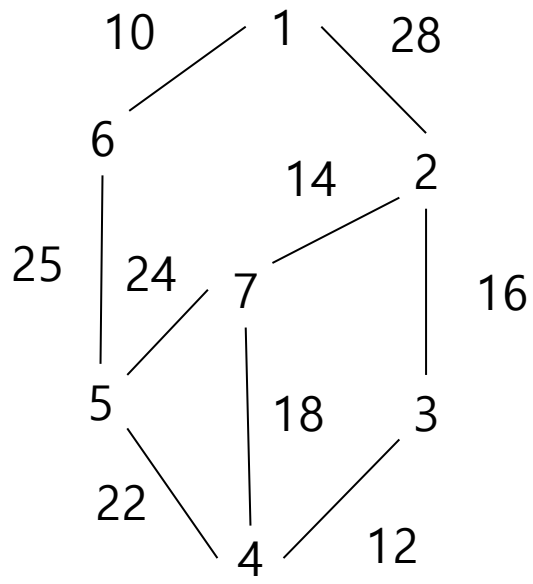


Greedy method to find minimum cost spanning tree.

1) **Prim's**
2) **Kruskal's**

Minimum Cost Spanning Tree Kruskal's

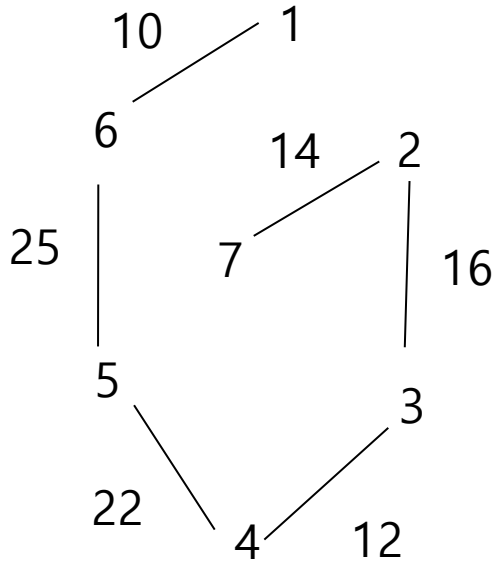
We have a weighted graph, we have to find MCS following Kruskal's



Kurskal's says

1) Always select the minimum cost edge 1-6
2) Next, select the next smallest cost edge 3-4
3) Next, select the next smallest cost edge 2-7
4) Next, select the next smallest cost edge 2-3
5) Next, select the next smallest cost edge 7-4 , but this will form a cycle
kruskals says, if the edge is forming a cycle, do not include that in solution
6) Next, select the next smallest cost edge 5-4
7) Next, select the next smallest cost edge 5-7, this is forming cycle
8) Next, select the next smallest cost edge 6-5
When to stop this loop
Ans: If you have selected V-1 edges then stop. As spanning tree can have V-1 edges only.

Cost of this MCS - 99



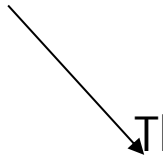
- this is Greedy

How to improve time complexity of Kruskals

- Time Complexity
  - E – no of edges
  - It has to always select the minimum cost edge out of these E edges
  - How many times it will do this task ?
  - So time complexity will be **O(E * E)**

- Using minheap / priorityQueue we can reduce this time complexity

  - What is Minheap -> If you store the edges into a min heap, it gives you the minimum cost edge on the top of the heap. So when you pop an item out of the heap the next minimum will come at the root of the tree



This task takes **O(lon E)** time

  - So getting the next minimum edge will be O(log E) instead of O(E)
  - So the total time complexity will be **O(E log E)**

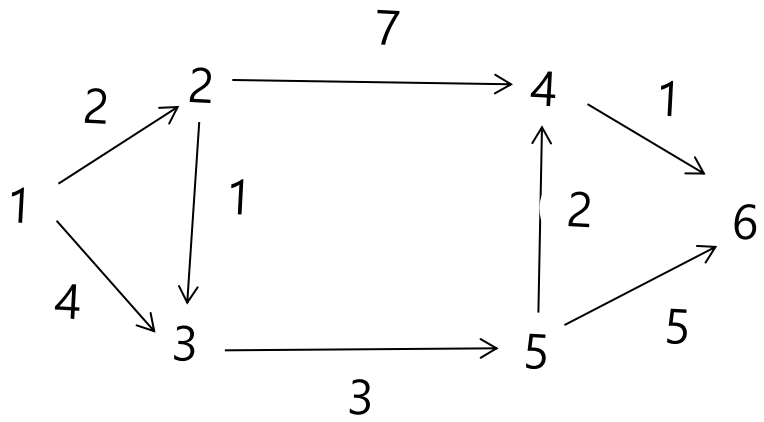
Use case of Non-Connected Graphs

- Which one is better for non – connected graph, prims or kruskals

MCS, can we have different MCS

- Yes, we can have multiple Spanning tree for same graph
- But there will be only one Minimum Cost

Single Source Shortest Path Problem
Dijkstra's Algorithm

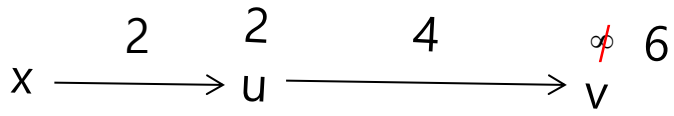


Weighted graph is given ,then finding the shortest path from some starting vertices to all other vertices.
For ex: from vertices 1 to all other vertices.

This is a **minimization** problem
So this is also an **optimization** problem

Dijkstra Algorithm

- Greedy says that problem should be solved in stages by taking one step at a time and talking one input at a time
- In Greedy there are predefined procedures and we follow that procedure to get the optimal solution.
- Dijkstras algorithm gives us the procedure for getting the optimal solution, that is shortest path
- Dijkstras work on directed and also on non-directed graph

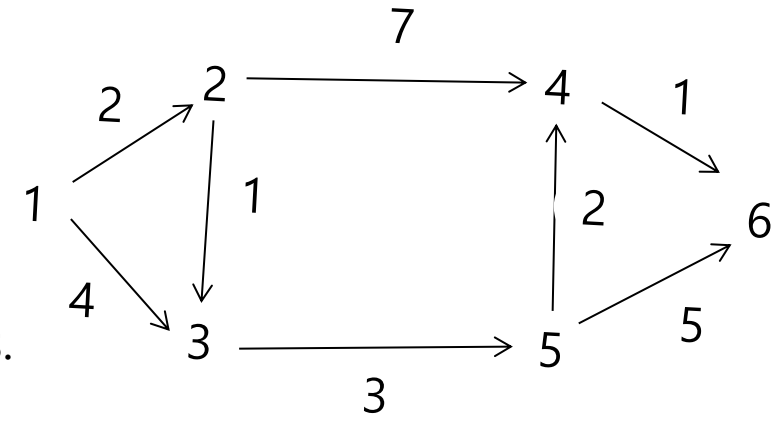


- There is a direct path from 1 -> 2 (cost 2) There is no direct path from 1 -> 3 (cost ∞)
- There is no direct path from 1->3, so it will check from second node if there is shortest path to 3.
- Yes there is a path from 2 to 3, then add 2 with 4, which is 6.
- 6 < infinity, hence that becomes shortest path -> This is called Relaxation
- Relaxation

```
if ( d[u] + c(u,v) < d[v])
      d[v] = d[u] + c[u,v)
```

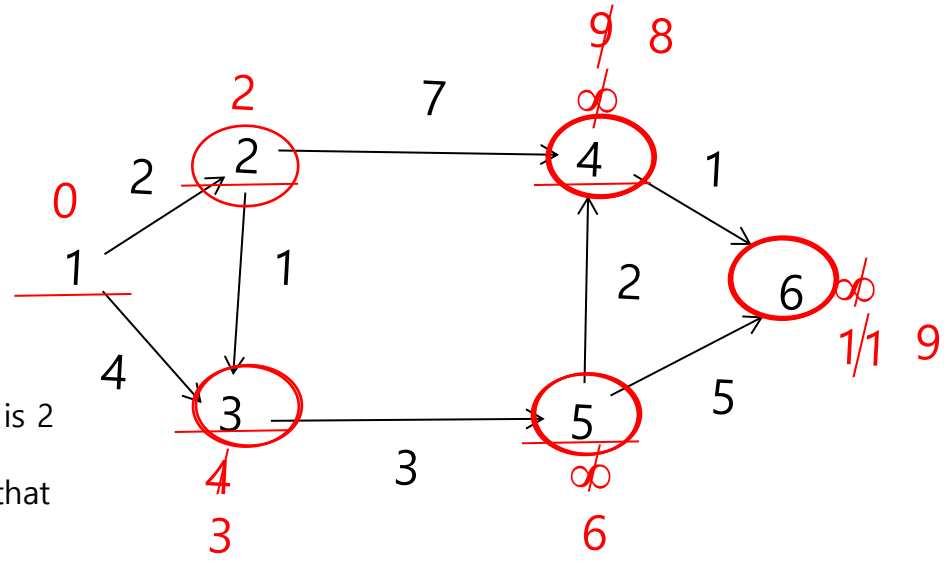
d[u] -> distance from x to u
d[v] -> distance from x to v
c[u,v] -> cost of connecting u to v

- Whenever we select the shortest path, we will try to relax other vertices.



Dijkstra Algorithm

- Relaxation
  if (d[u] + c(u,v) < d[v])
  d[v] = d[u] + c[u,v)
- Whenever we select the shortest path, we will try to relax other vertices.
- Initialization
  - We assign distances to all the nodes from node 1
  - There is direct edge from 1 to 2 and 3, so 2 and 4 are assigned
  - There is no direct edge from 1 to 4, 5 and 6, so we assign infinity
- Repeating phase
- (1)
  - Select the smallest distance node, select node **2**, its distance from origin is 2
  - Check all the connected nodes from the node 2
  - Node 4 is connected to 2 -> check the distance , its infinity, try to relax that vertices
    ∞ > 2 +7, hence modify ∞ to 9.
  - Node 3 is connected to 2 -> check the distance, its 3, try to relax the vertice s 3.
    4 > 2 + 1, hence modify 4 to 3.
- (2)
  - Select the smallest distance node among the non visited nodes, which are n ode 3, 5, 6, and 4. Smallest distance node is **3**, its distance from origin is 3
  - Check all the connected nodes from the node 3
  - Only one node is connected to 3, that is node 5 -> check the distance, its in finity, try to relax the vertices
    ∞ > 3 +3, hence modify ∞ to 6.
- (3)
  - Select the smallest distance node among non visited nodes, which are 5, 6 a nd 4. smallest distance node is **5**. its distance from origin is 6.
  - Check all the connected nodes from the node 5
  - Node 6 is connected to 5 -> check the distance, its infinity, try to relax that vertices

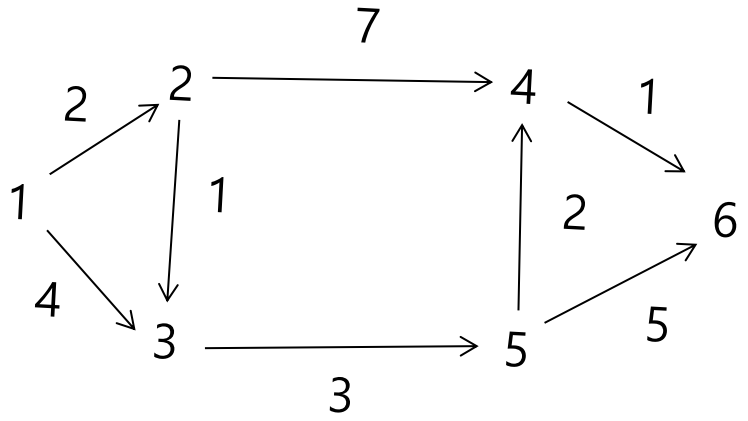


    ∞ > 6+5 , hence modify ∞ to 11.
  - Node 4 is also connected to 5 -> check the distance, its 9, try to relax
    9 > 6+2, hence modify 9 to 8.
- (4)
  - Select the smallest distance node among non visited nodes, w hich are 4 and 6. smallest node is **4**. its distance from origin is 8.
  - Check all the connected nodes from the node 4
  - Node 6 is connected to 4 -> Check the distance, its 11, try to relax that vertices
    11 > 8 + 1, hence modify 11 to 9.

Dijkstra Algorithm

Starting at vertex 1

Selected Vertex	2	3	4	5	6
2	2	4	∞	∞	∞
3	2	3	9	∞	∞
5	2	3	9	6	∞
4	2	3	8	6	11
	2	3	8	6	9



Select the smallest vertex
From 2, 2 nodes are connected, can we relax them? Yes
Rest all copy as it is

Select the next smallest
From 3, 1 node is connected, can we relax them? Yes
Rest all copy as it is

Select the next smallest
From 5, 2 nodes are connected, can we relax them? Yes
Rest all copy as it is

Select the next smallest
From 4, 1 node is connected, can we relax it ? Yes
Rest all copy as it is

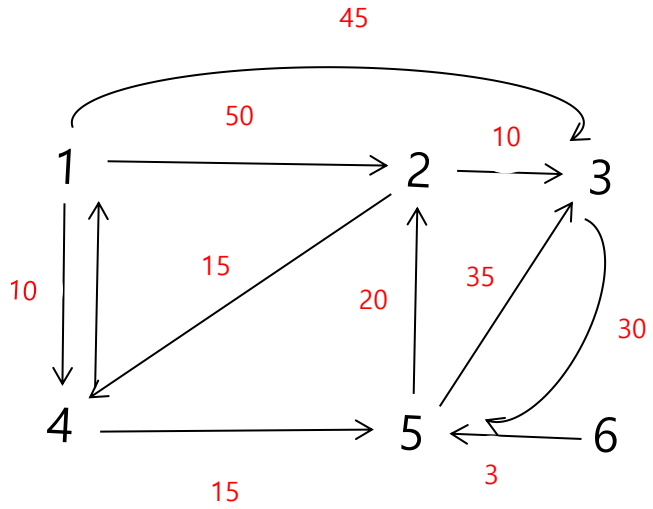
Time Complexity of Dikstras

- Vertex **n**
- For all vertex its finding shortest path
  - While doing so, it is relaxing
  - Total how many vertex its relaxing, in worst case all **n** vertex (when all nodes are connected to each other)
  - So for all the **n** vertex, in worst case its relaxing all **n** vertex.
  - O(**n*n) ->** O(**n^2**)

Question

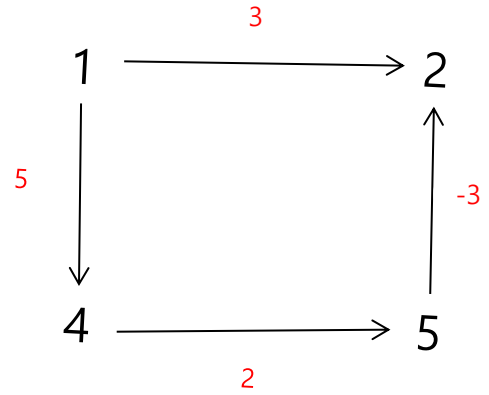
Starting at vertex 1

Selected Vertex	2	3	4	5	6
	50	45	10	∞	∞



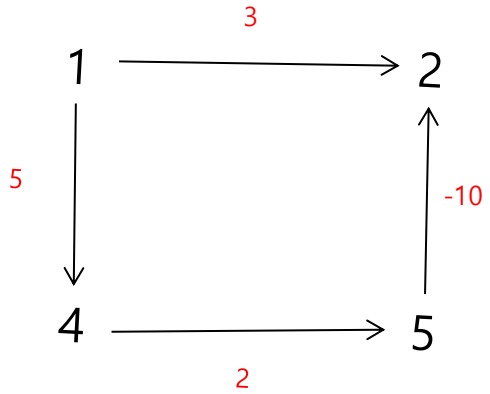
Drawback of Dijkstra algorithm

- What happens when you have –ve weight on the edge? Dijkstra has not considered the –ve weight. However having –ve weight is very common in many scenarios.
- According to dijkastra's we relax only non selected node, but that approach gives us wrong result in the 2nd graph.
- Dijkastra algorithm may or may not work on –ve edges.



For this one it will work

	2	4	5
2	3	5	inf
4	3	5	inf
7	3	5	7
	We don't have any node left to relax		



For this one it will not work

according to dijkstras we relax only non selected node, from node 5 the only connected node is node 2, but that node is already selected node. Highlighted in red, so we don't relax already selected nodes. However if we try to relax, it gives us better results, that is -3 in the ca se of second graph, which is lesser than the existing distance 3. hence dijkstras failed.

Johnson's Algorithm

- This algorithm works by reweighting the edges to make all weights non-negative and then applies Dijkstra's Algorithm

Preserving shortest paths by reweighting

The following lemma shows how easily we can reweight the edges to satisfy the first property above. We use δ to denote shortest-path weights derived from weight function w and $\widehat{\delta}$ to denote shortest-path weights derived from weight function $\widehat{w}$.

- Book - 25.3