CSE 221 Algorithms: Ch4

Instructor: Saraf Anika

CSE, SoSET, EDU

Summer 2021

Greedy Method

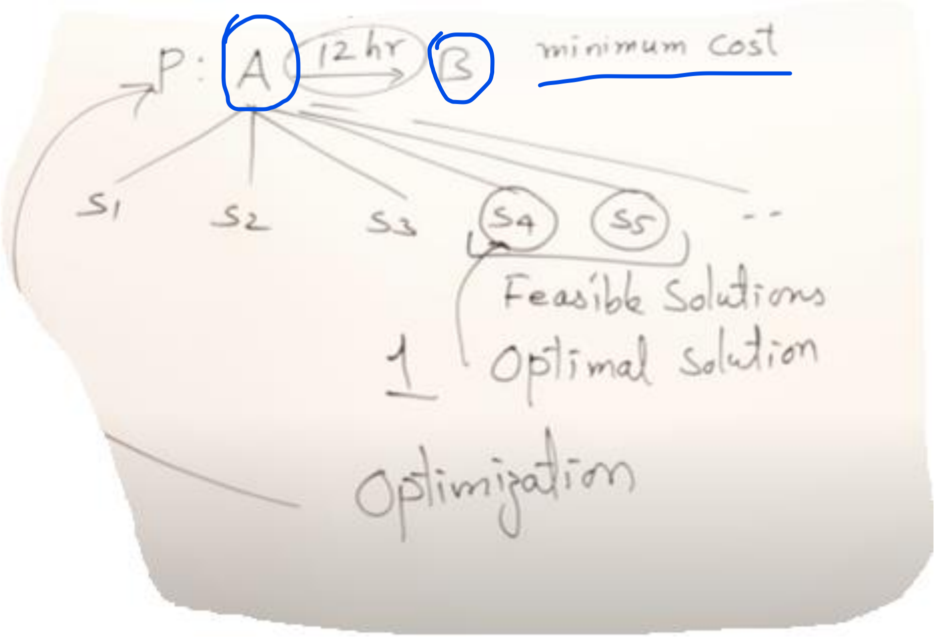
- **What is a 'Greedy Algorithm'?**

A greedy algorithm, as the name suggests, always makes the choice that seems to be the best at that moment. This means that it makes a locally-optimal choice in the hope that this choice will lead to a globally-optimal solution.

- **How do you decide which choice is optimal?**

Assume that you have an objective function that needs to be optimized (either maximized or minimized) at a given point. A Greedy algorithm makes greedy choices at each step to ensure that the objective function is optimized. The Greedy algorithm has only one shot to compute the optimal solution so that it never goes back and reverses the decision.

Greedy Method Cont.

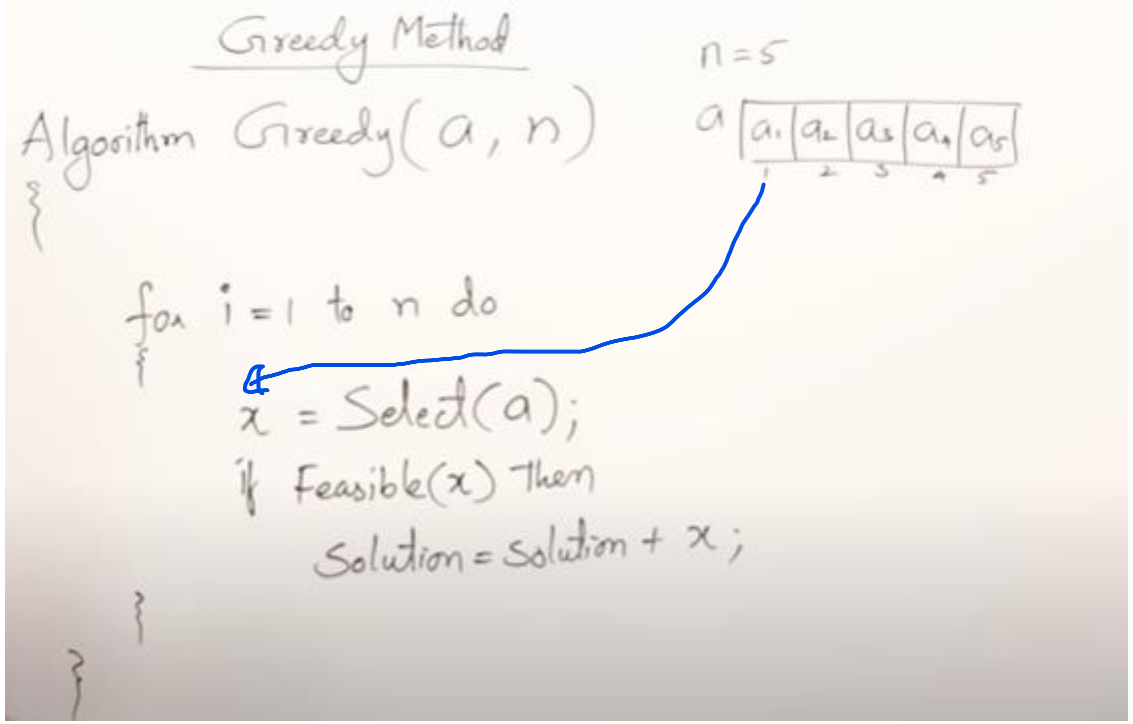


- Components of Greedy Algorithm

Greedy algorithms have the following five components −

- **A candidate set** − A solution is created from this set.
- **A selection function** − Used to choose the best candidate to be added to the solution.
- **A feasibility function** − Used to determine whether a candidate can be used to contribute to the solution.
- **An objective function** − Used to assign a value to a solution or a partial solution.
- **A solution function** − Used to indicate whether a complete solution has been reached.

Greedy Method Cont.



Greedy Method

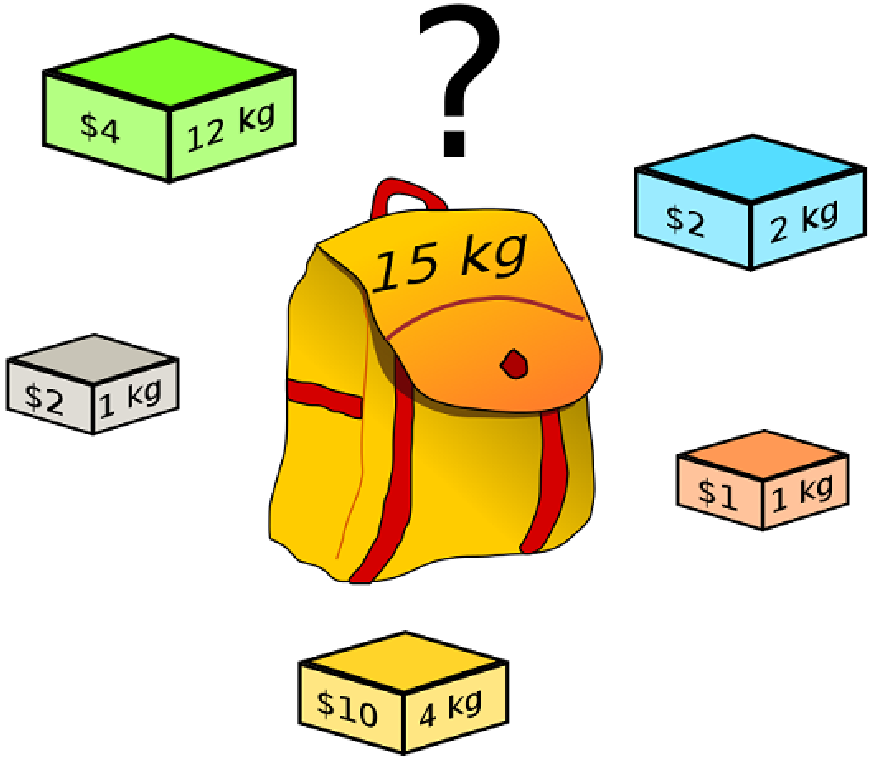
n=5

a	a_1	a_2	a_3	a_4	a_5
	1	2	3	4	5

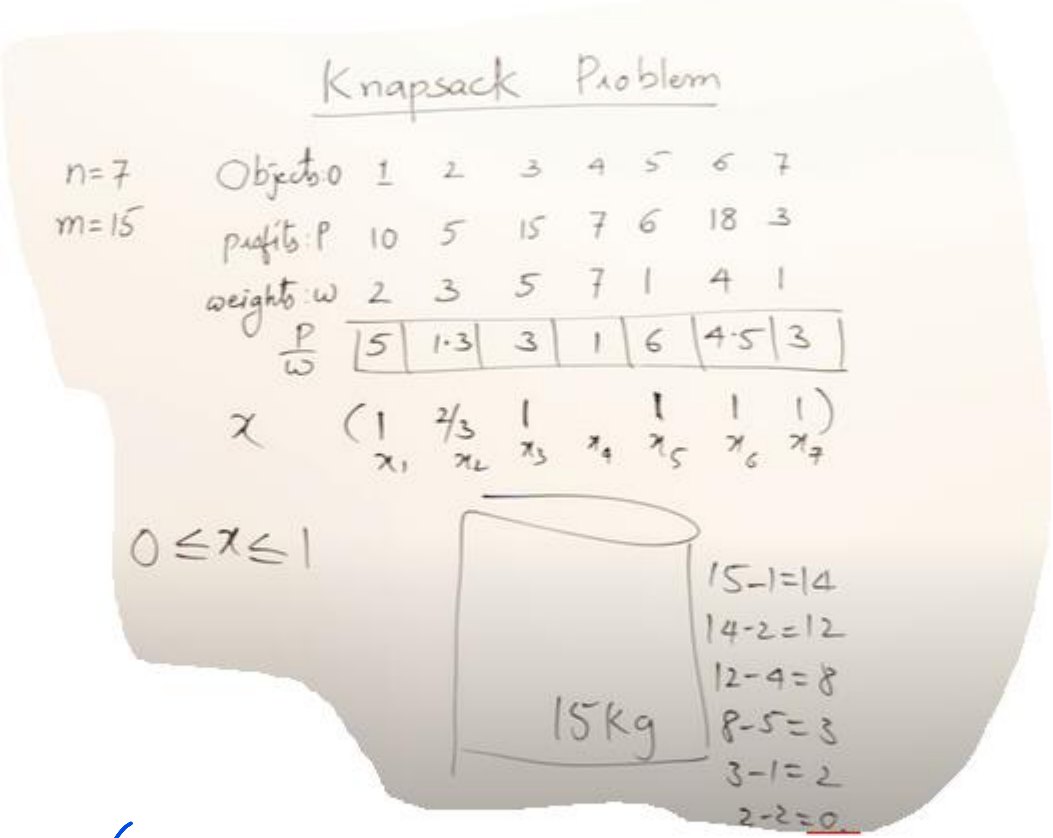
```
Algorithm Greedy(a, n)
{
    for i=1 to n do
    {
        x = Select(a);
        if Feasible(x) Then
            Solution = Solution + x;
    }
}
```

Knapsack Problem

Given a Knapsack of a maximum capacity of W and N items each with its own value and weight, throw in items inside the Knapsack such that the final contents has the maximum value. Yikes !!



Knapsack Problem



Knapsack Problem

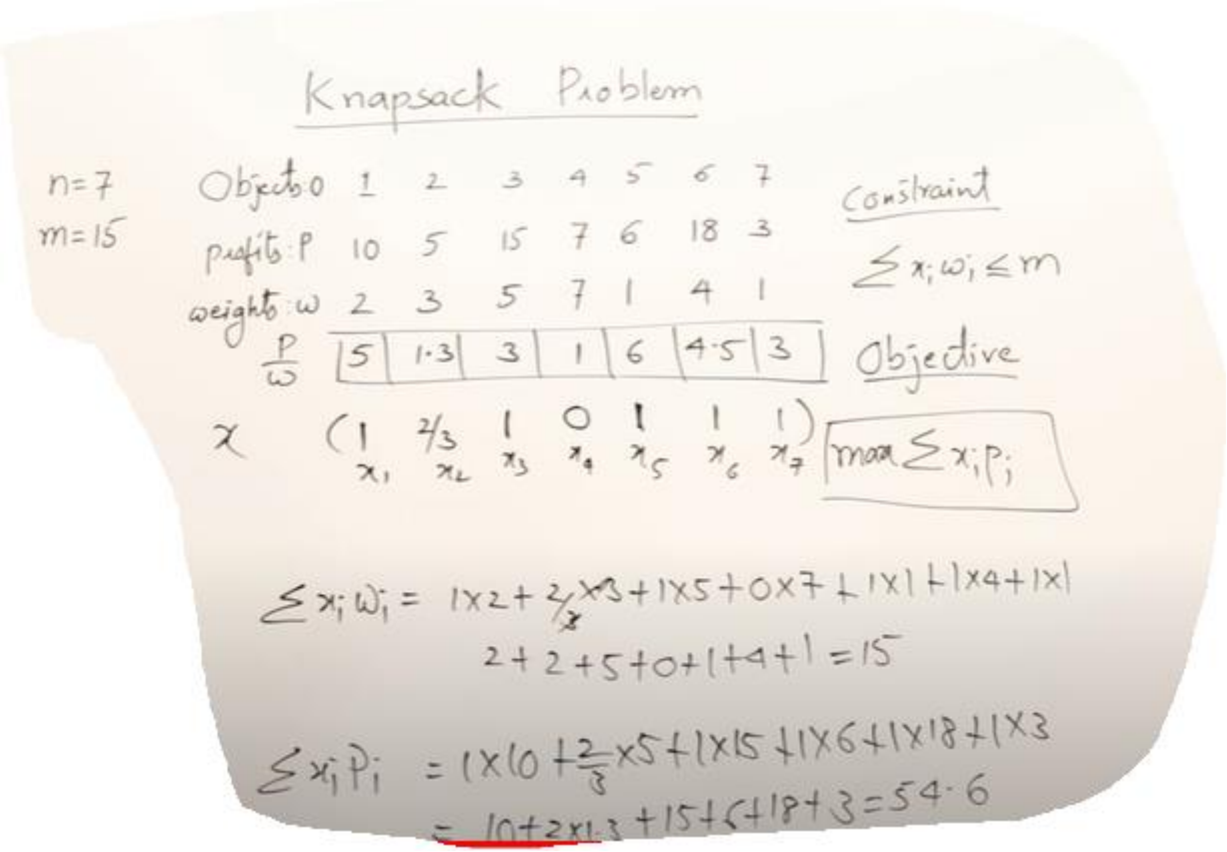
n=7
m=15

Objects:o	1	2	3	4	5	6	7
profits:P	10	5	15	7	6	18	3
weights:w	2	3	5	7	1	4	1
$\frac{P}{\omega}$	5	1.3	3	1	6	4.5	3

$x \quad (\underset{x_1}{1}\ \ \underset{x_2}{2/3}\ \ \underset{x_3}{1}\ \ \underset{x_4}{}\ \ \underset{x_5}{1}\ \ \underset{x_6}{1}\ \ \underset{x_7}{1})$

$0 \leq x \leq 1$

15-1=14
14-2=12
12-4=8
8-5=3
3-1=2
2-2=0



Knapsack Problem

n=7
m=15

Objects:o	1	2	3	4	5	6	7
profits:P	10	5	15	7	6	18	3
weights:w	2	3	5	7	1	4	1
$\frac{P}{\omega}$	5	1.3	3	1	6	4.5	3

$x \quad (\underset{x_1}{1}\ \ \underset{x_2}{2/3}\ \ \underset{x_3}{1}\ \ \underset{x_4}{0}\ \ \underset{x_5}{1}\ \ \underset{x_6}{1}\ \ \underset{x_7}{1})$

Constraint

$\sum x_i w_i \leq m$

Objective

$\boxed{\max \sum x_i p_i}$

$$\sum x_i w_i = 1\times2 + \tfrac{2}{3}\times3 + 1\times5 + 0\times7 + 1\times1 + 1\times4 + 1\times1$$
$$2 + 2 + 5 + 0 + 1 + 4 + 1 = 15$$

$$\sum x_i P_i = 1\times10 + \tfrac{2}{3}\times5 + 1\times15 + 1\times6 + 1\times18 + 1\times3$$
$$= 10 + 2\times1.3 + 15 + 6 + 18 + 3 = 54.6$$

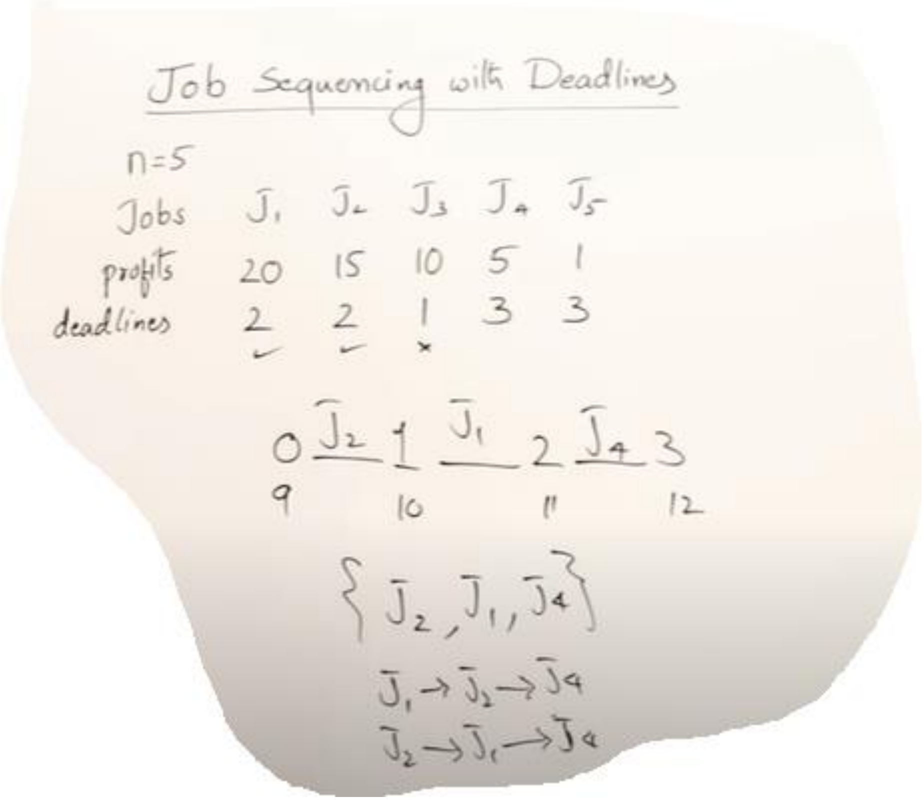
✓ P/W high to low calculate. That means sorted value.
I choose = object 5, 1, 4.5, 3, 7, 2. (set by the P/W)
Then,
15 - (P/W is weight) = ans
....

Job Sequencing with Deadlines

- The sequencing of jobs on a single processor with deadline constraints is called as Job Sequencing with Deadlines.
- Here-
  - You are given a set of jobs.
  - Each job has a defined deadline and some profit associated with it.
  - The profit of a job is given only when that job is completed within its deadline.
  - Only one processor is available for processing all the jobs.
  - Processor takes one unit of time to complete a job.
- Greedy Algorithm is adopted to determine how the next job is selected for an optimal solution.

Job Sequencing with Deadlines

Example 1:



Job Sequencing with Deadlines

$n=5$

Jobs	J_1	J_2	J_3	J_4	J_5
profits	20	15	10	5	1
deadlines	2	2	1	3	3
	✓	✓	×		

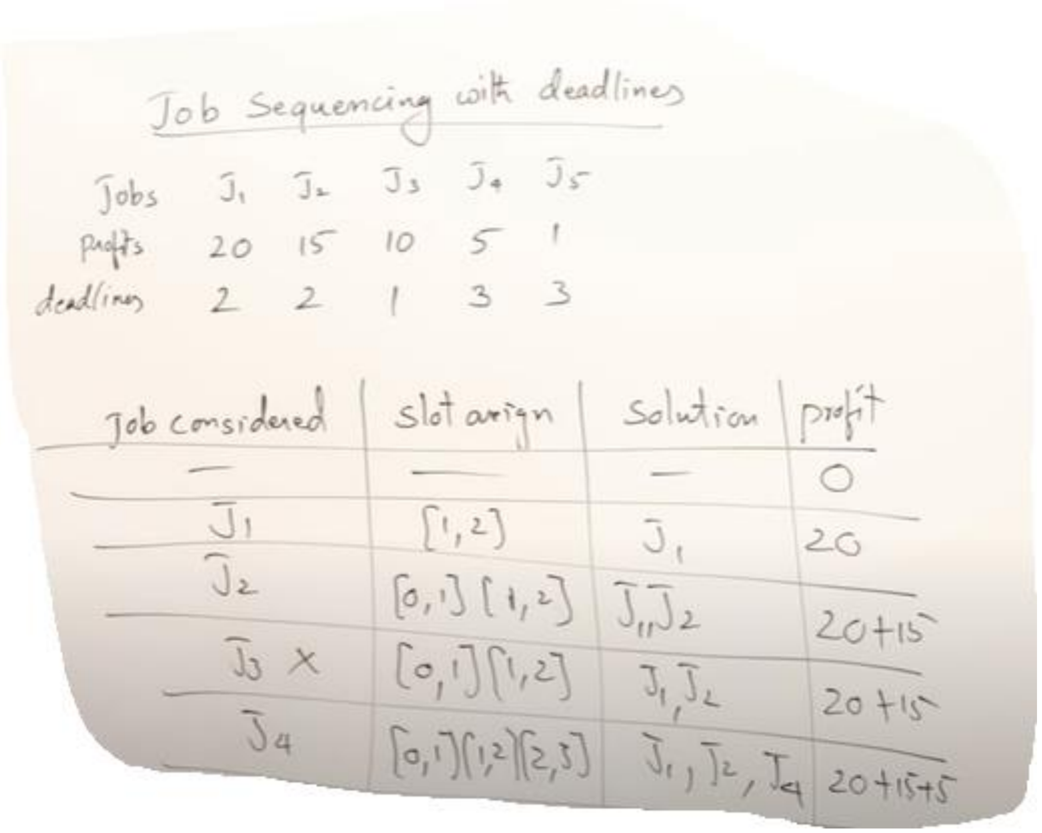
$0\ \underline{J_2}\ 1\ \underline{J_1}\ 2\ \underline{J_4}\ 3$

9 10 11 12

$\{J_2, J_1, J_4\}$

$J_1 \to J_2 \to J_4$

$J_2 \to J_1 \to J_4$



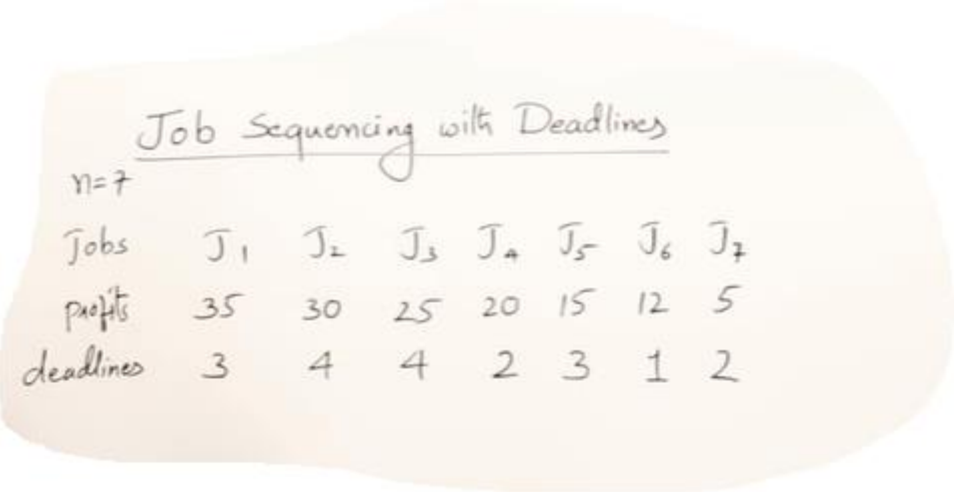
Job Sequencing with deadlines

Jobs	J_1	J_2	J_3	J_4	J_5
Profits	20	15	10	5	1
deadlines	2	2	1	3	3

Job considered	slot assign	Solution	profit
—	—	—	0
J_1	[1,2]	J_1	20
J_2	[0,1] [1,2]	J_1, J_2	20+15
J_3 ×	[0,1] [1,2]	J_1, J_2	20+15
J_4	[0,1] [1,2] [2,3]	J_1, J_2, J_4	20+15+5

Job Sequencing with Deadlines

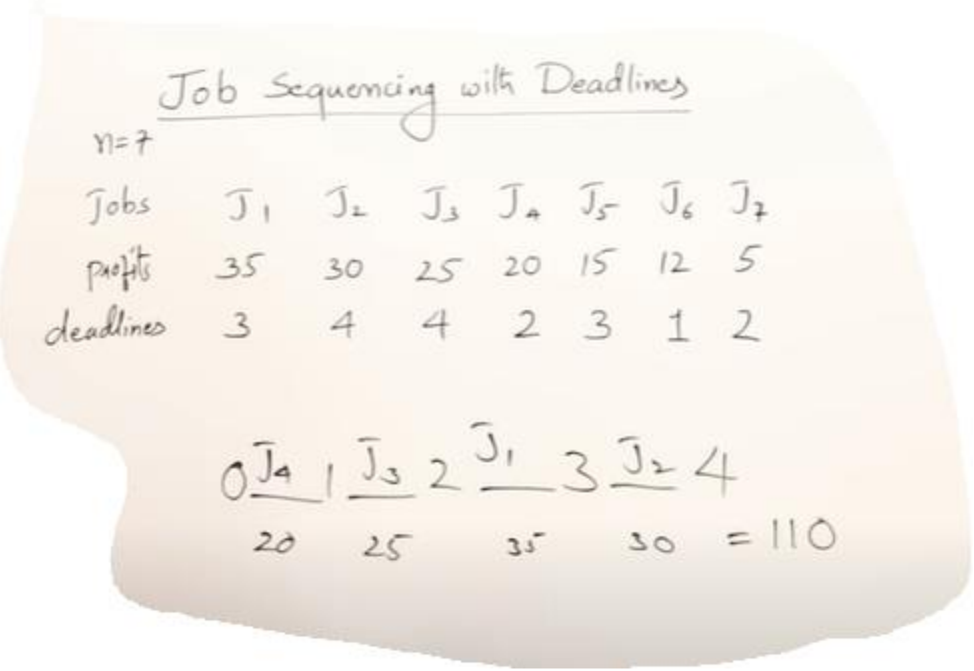
DIY: Practice



Job Sequencing with Deadlines

n=7

Jobs	J_1	J_2	J_3	J_4	J_5	J_6	J_7
profits	35	30	25	20	15	12	5
deadlines	3	4	4	2	3	1	2



Job Sequencing with Deadlines

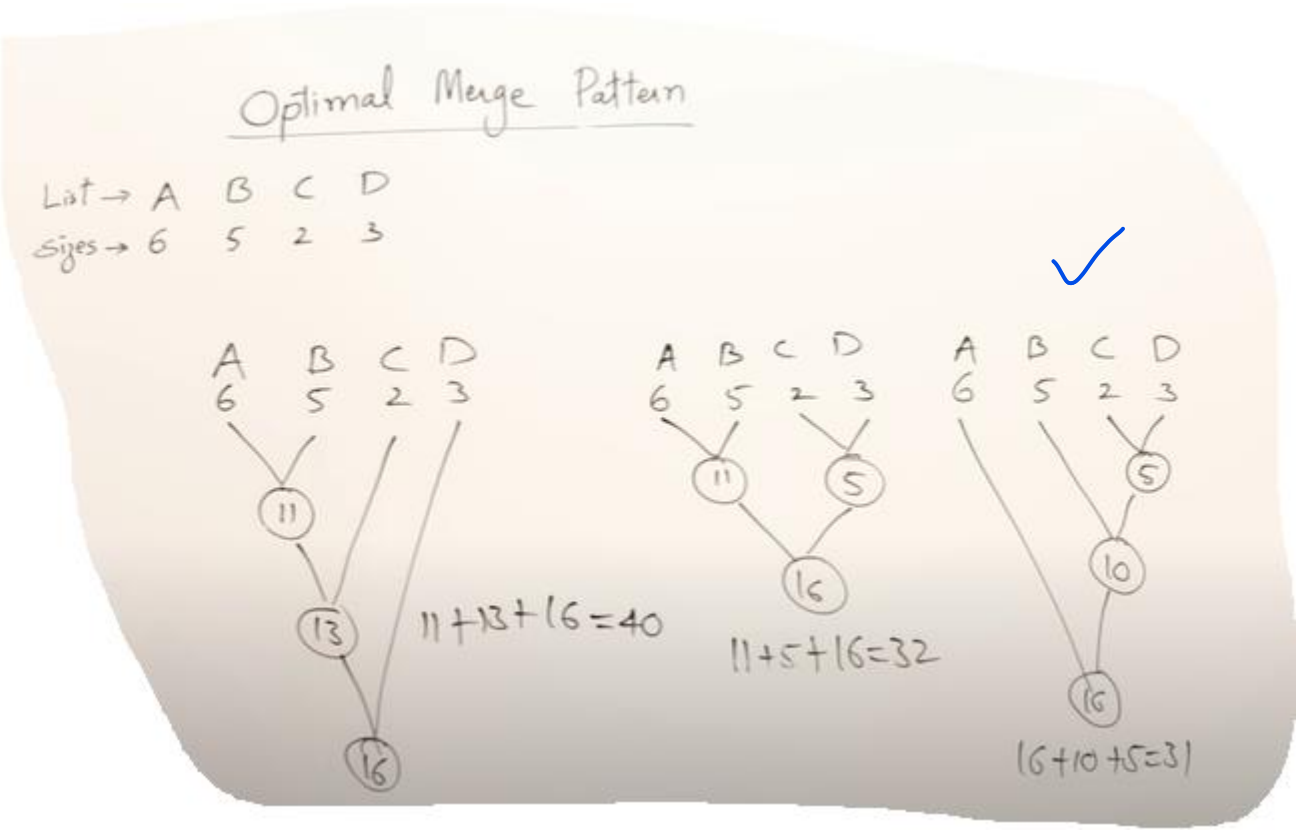
n=7

Jobs	J_1	J_2	J_3	J_4	J_5	J_6	J_7
profits	35	30	25	20	15	12	5
deadlines	3	4	4	2	3	1	2

$0 \overset{J_4}{\_\_} 1 \overset{J_3}{\_\_} 2 \overset{J_1}{\_\_} 3 \overset{J_2}{\_\_} 4$

20 25 35 30 = 110

Optimal Merge Pattern



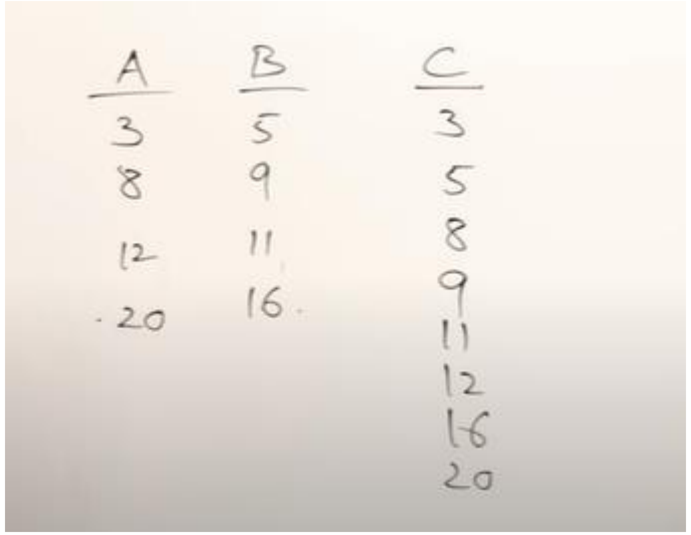
Optimal Merge Pattern

List →	A	B	C	D
sizes →	6	5	2	3

11+13+16=40

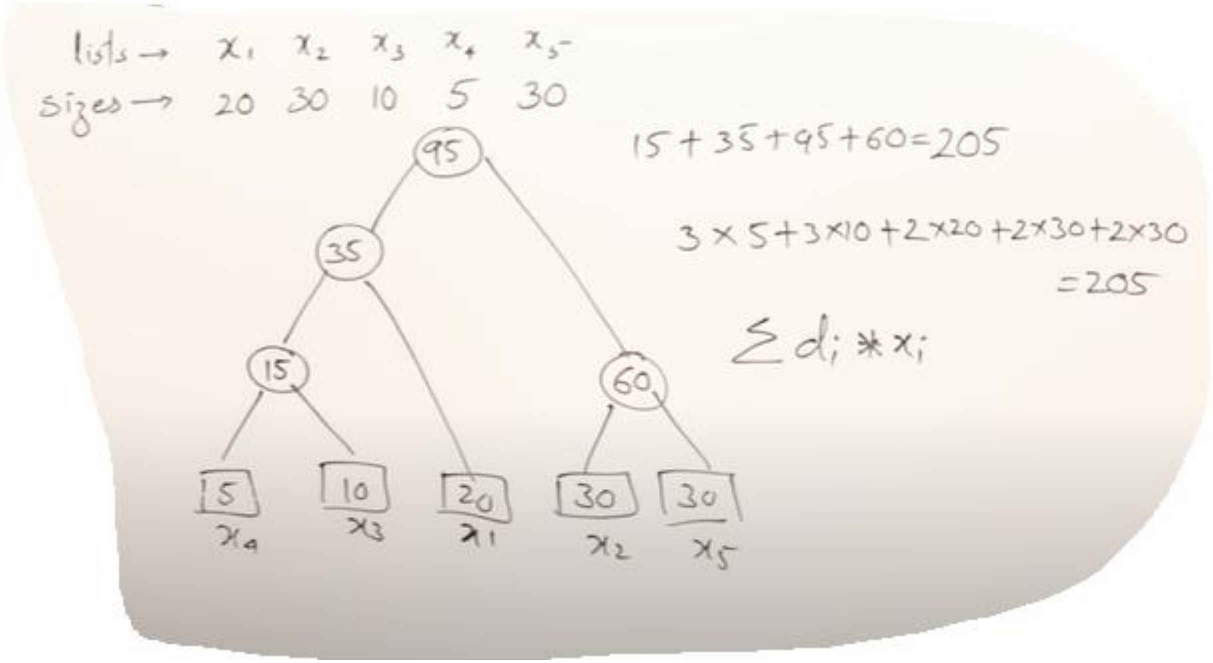
11+5+16=32

16+10+5=31



A	B	C
3	5	3
8	9	5
12	11	8
20	16	9
		11
		12
		16
		20

Optimal Merge Pattern



lists →	x_1	x_2	x_3	x_4	x_5
sizes →	20	30	10	5	30

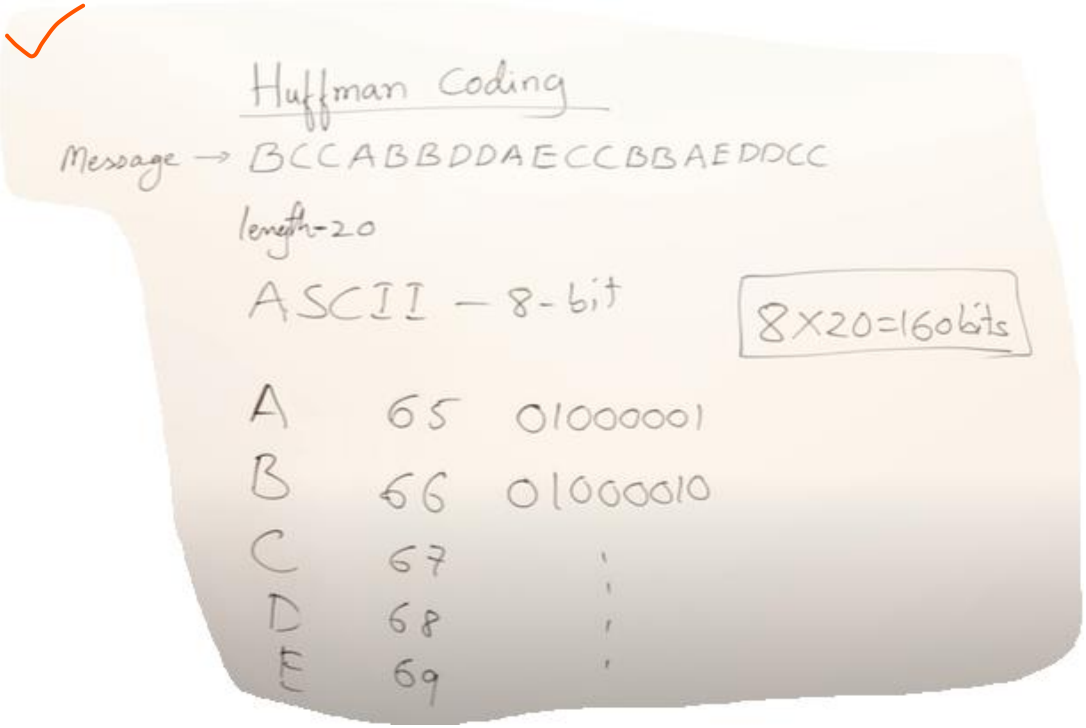
$15+35+95+60=205$

$3\times5+3\times10+2\times20+2\times30+2\times30$
$=205$

$\sum d_i * x_i$

Huffman Coding

- Huffman Coding is a technique of compressing data to reduce its size without losing any of the details. It was first developed by David Huffman.
- Huffman Coding is generally useful to compress the data in which there are frequently occurring characters.
- The most frequent character gets the smallest code and the least frequent character gets the largest code.



Huffman Coding

Message → BCCABBDDAECCBBAEDDCC

length-20

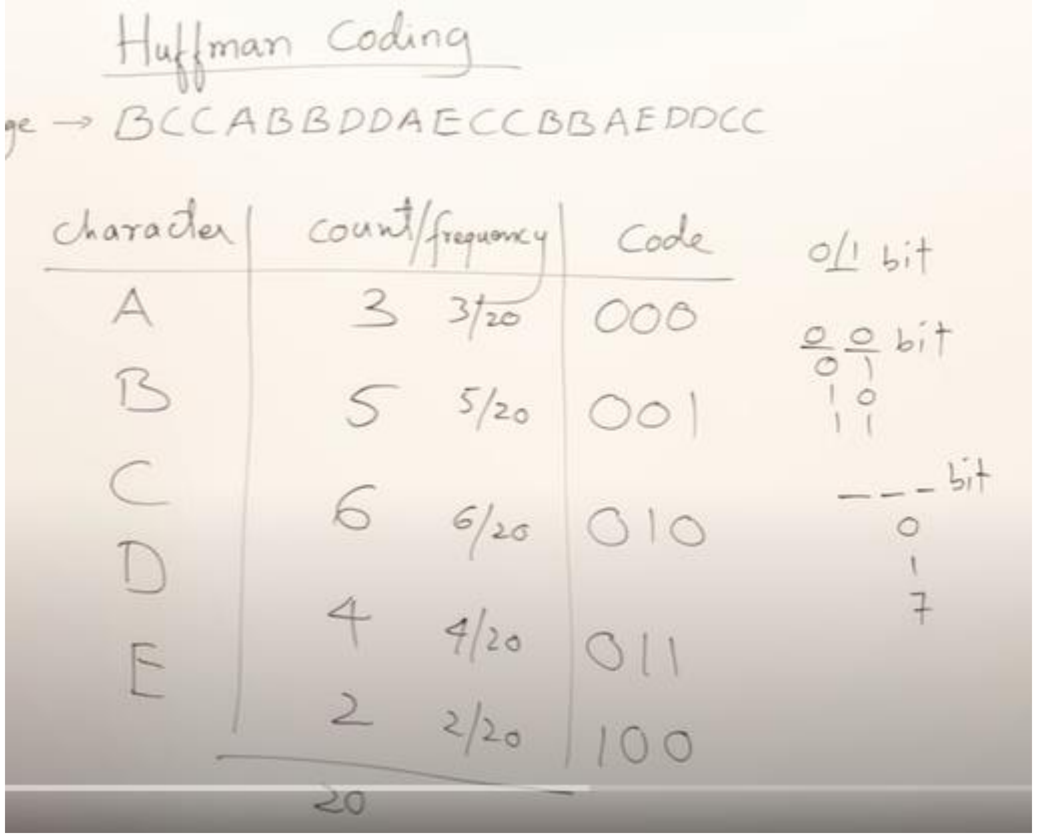
ASCII – 8-bit

$\boxed{8 \times 20 = 160\text{bits}}$

A	65	01000001
B	66	01000010
C	67	⋮
D	68	
E	69	

Huffman Coding Cont.

- For Fixed Sized Codes: (Method 1)



Huffman Coding

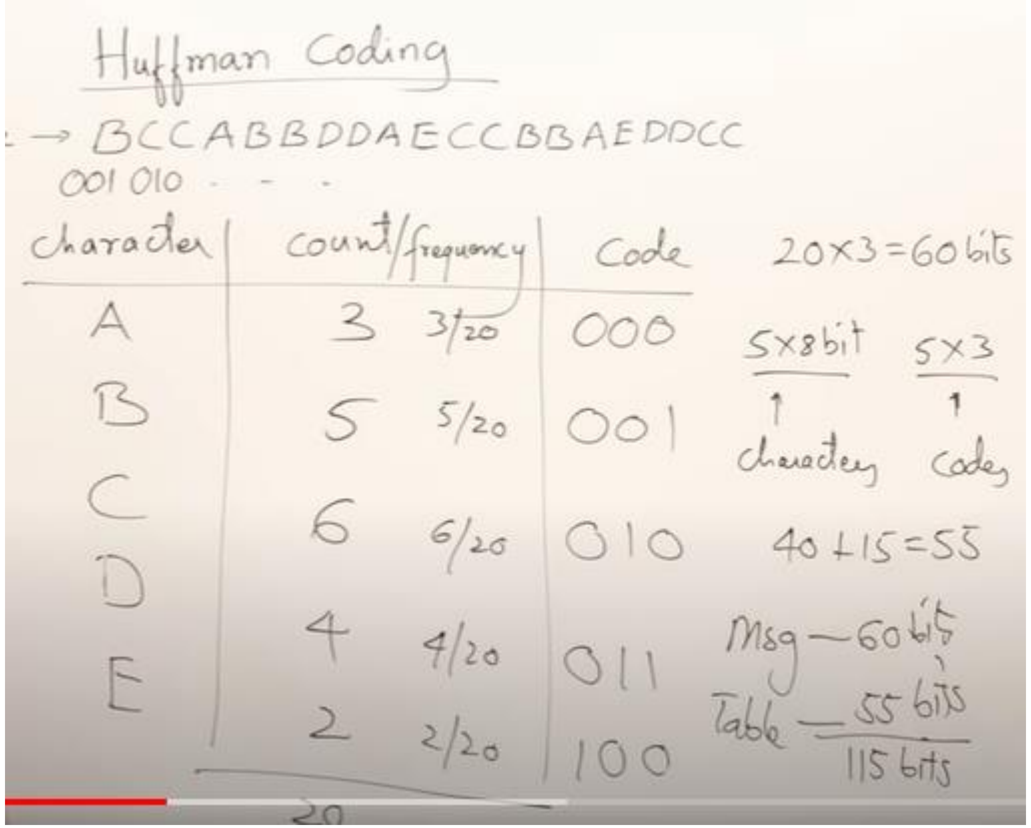
ge → BCCABBBDDAECCBBAEDDCC

character	count/frequency		Code
A	3	3/20	000
B	5	5/20	001
C	6	6/20	010
D	4	4/20	011
E	2	2/20	100
	20		

0/1 bit

$\frac{0}{}\ \frac{0}{}$ bit
0 1
1 0
1 1

_ _ _ bit
0
1
7



Huffman Coding

→ BCCABBBDDAECCBBAEDDCC
001 010 . . .

character	count/frequency		Code
A	3	3/20	000
B	5	5/20	001
C	6	6/20	010
D	4	4/20	011
E	2	2/20	100
	20		

20×3=60 bits

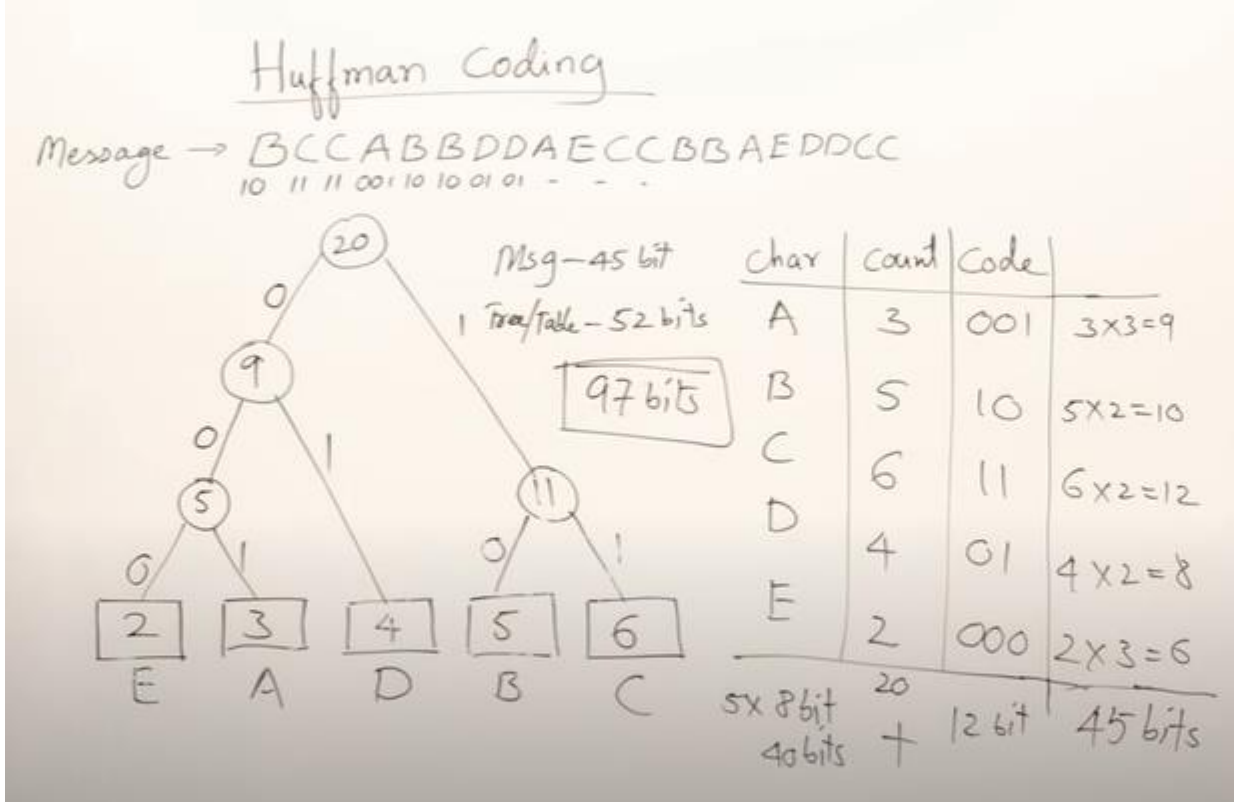
$\underline{5\times 8\text{bit}}$ $\underline{5\times 3}$
↑ ↑
characters codes

40+15=55

Msg — 60 bits
Table — $\underline{55 \text{ bits}}$
115 bits

Huffman Coding Cont.

- For Variable Sized Codes: (Method 2)
- From Root to Leaf



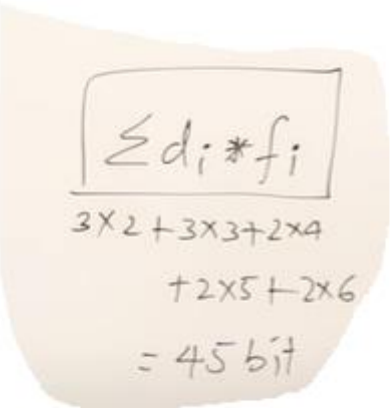
Huffman Coding

Message → BCCABBDDAECCBBAEDDCC
10 11 11 001 10 10 01 01 - - .

Msg—45 bit
Tree/Table—52 bits

$\boxed{97 \text{ bits}}$

Char	Count	Code	
A	3	001	3x3=9
B	5	10	5x2=10
C	6	11	6x2=12
D	4	01	4x2=8
E	2	000	2x3=6
5x 8bit 40bits	20 +	12 bit	45 bits

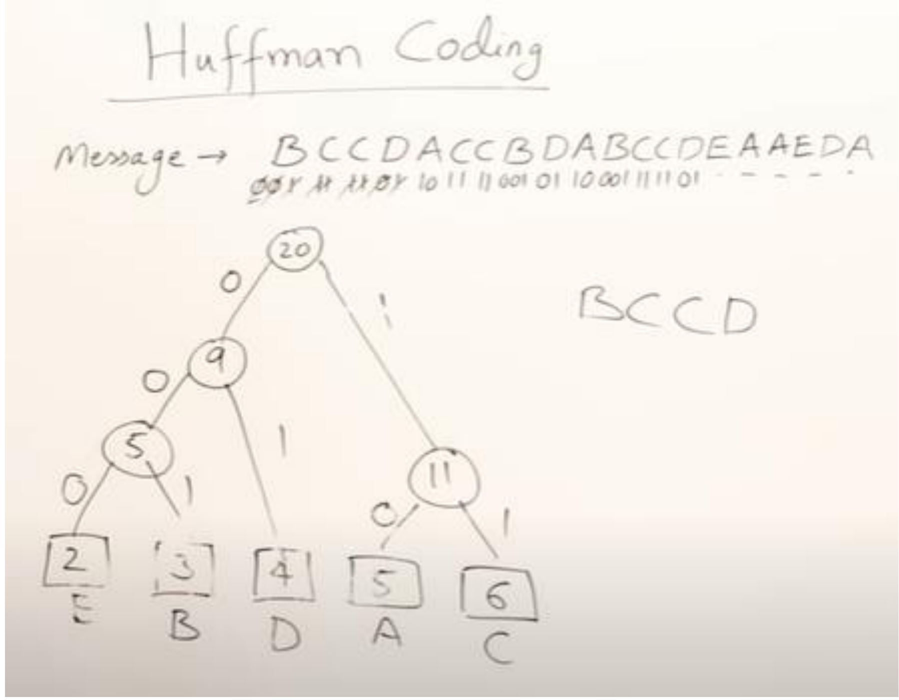


$\boxed{\sum d_i * f_i}$

3x2+3x3+2x4
+2x5+2x6
= 45 bit

Huffman Decoding

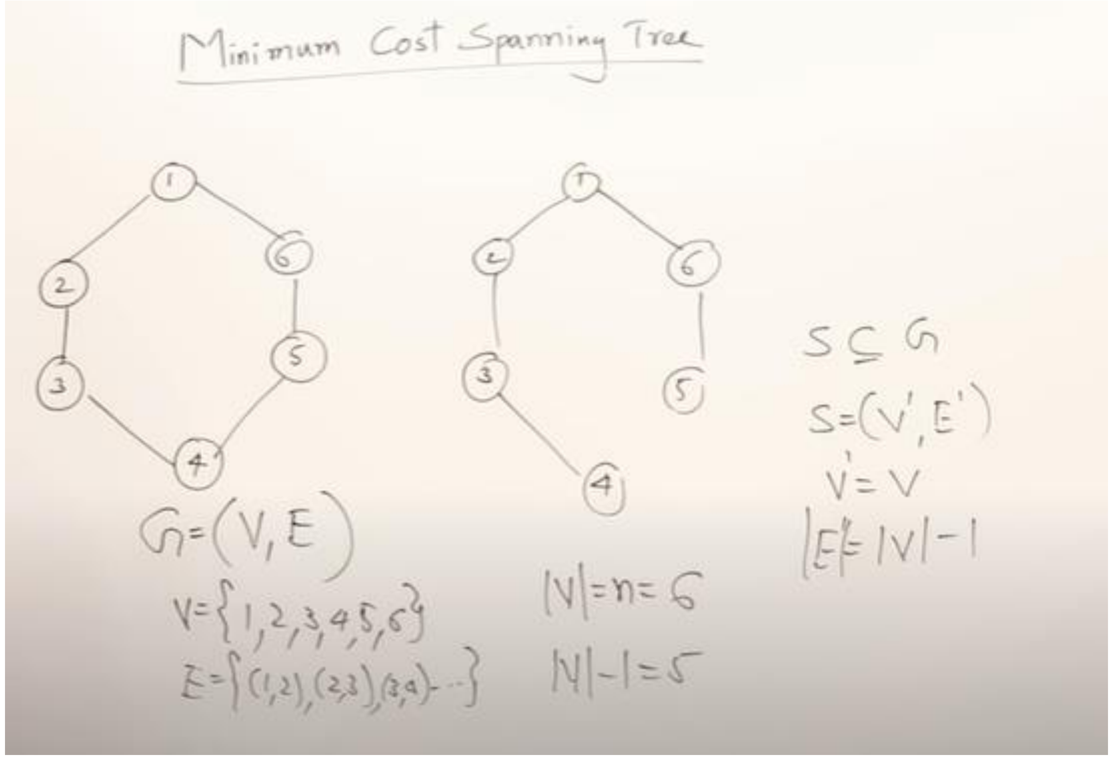
- From root to leaf



Minimum Spanning Tree

- Given an undirected and connected graph *G=(V,E)*, a spanning tree of the graph G is a tree that spans G (that is, it includes every vertex of G) and is a sub-graph of G (every edge in the tree belongs to G)
- The cost of the spanning tree is the sum of the weights of all the edges in the tree.
- Minimum spanning tree is the spanning tree where the cost is minimum among all the spanning trees.
- There are two famous algorithms for finding the Minimum Spanning Tree:
  - ✓ Prim’s Algorithm
  - ✓ Kruskal’s Algorithm

Minimum Spanning Tree



Minimum Cost Spanning Tree

$G=(V,E)$

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

$E=\{(1,2),(2,3),(3,4)\cdots\}$

$|V|=n=6$

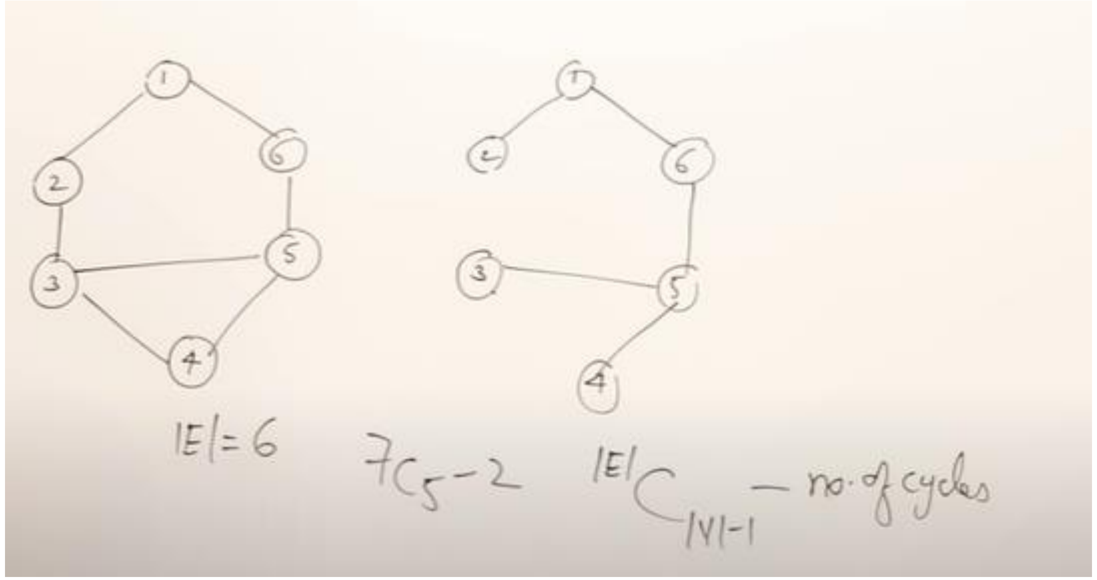
$|V|-1=5$

$S \subseteq G$

$S=(V',E')$

$V'=V$

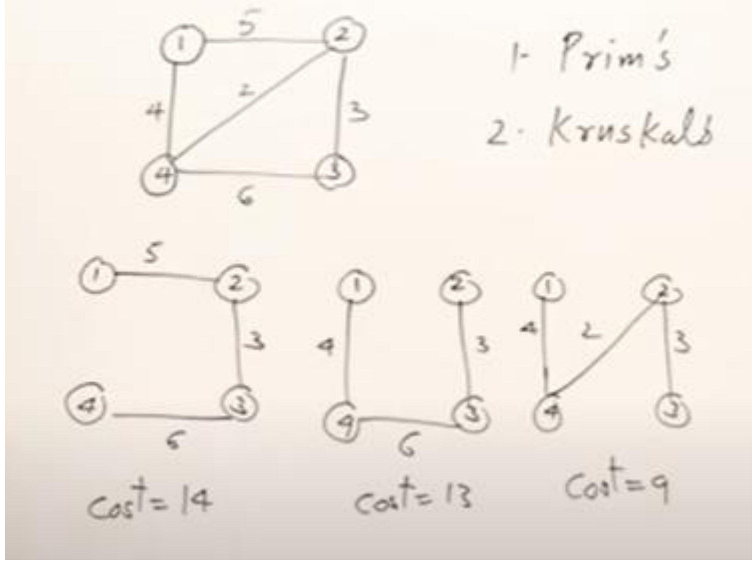
$|E'|=|V|-1$



$|E|=6$

${}^{7}C_{5}-2$

${}^{|E|}C_{|V|-1}$ − no. of cycles



1. Prim's

2. Kruskals

cost = 14

cost = 13

cost = 9

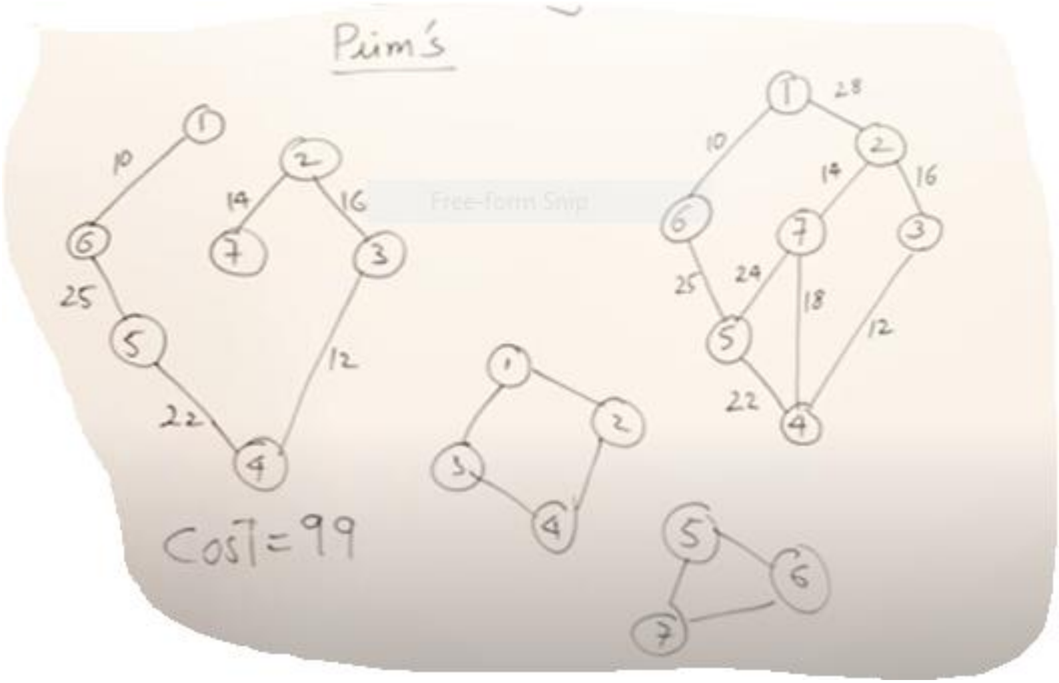
Prim’s Algorithm

- It falls under a class of algorithms called Greedy Algorithms that find the local optimum in the hopes of finding a global optimum.

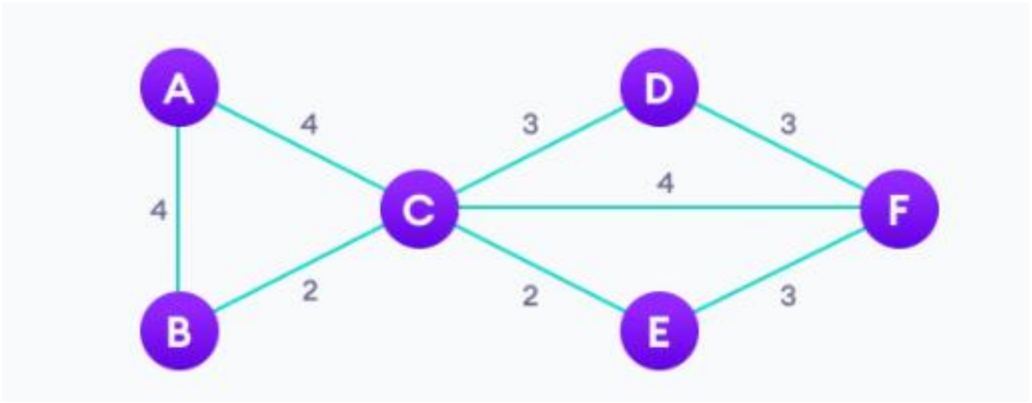
- We start from one vertex and keep adding edges with the lowest weight until we reach our goal.

- The steps for implementing Prim's algorithm are as follows:
  - Initialize the minimum spanning tree with a vertex chosen at random.
  - Find all the edges that connect the tree to new vertices, find the minimum and add it to the tree
  - Keep repeating step 2 until we get a minimum spanning tree

Prim’s Algorithm



DIY:

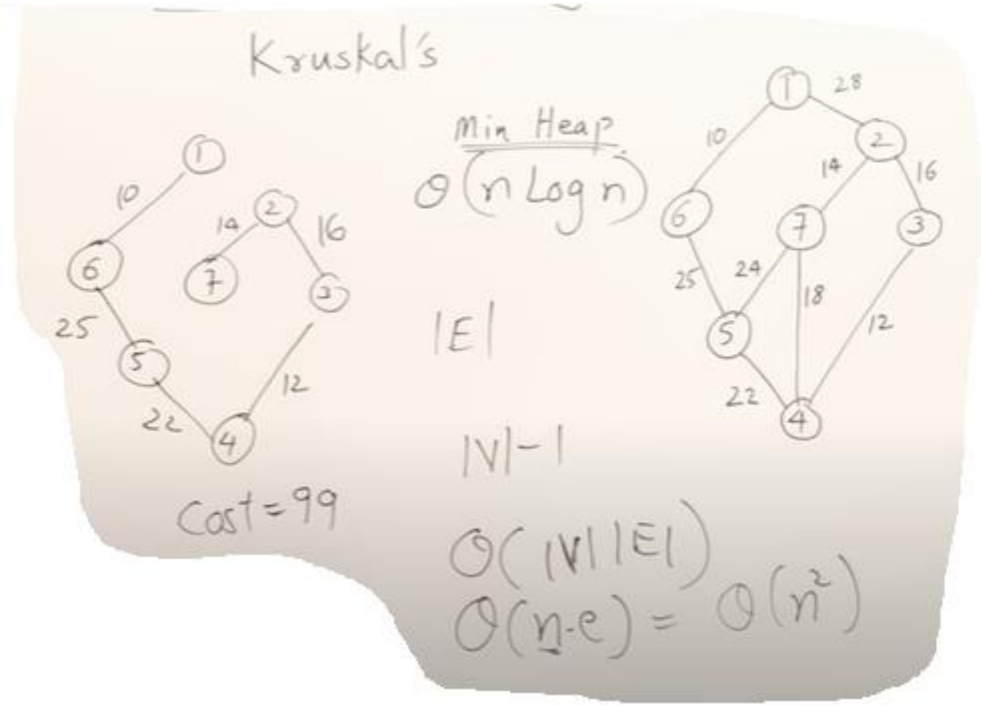


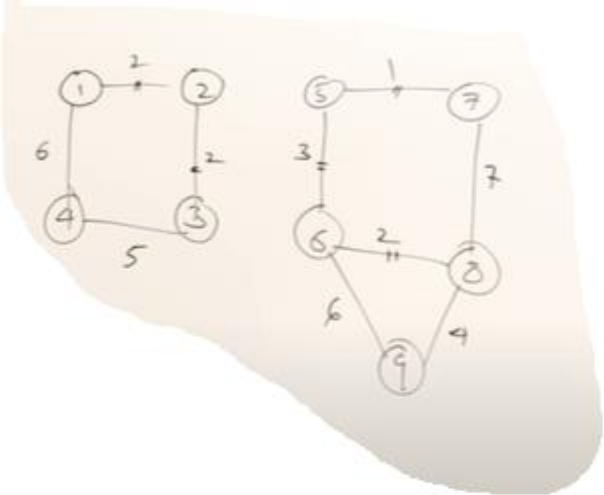
Cost = 14

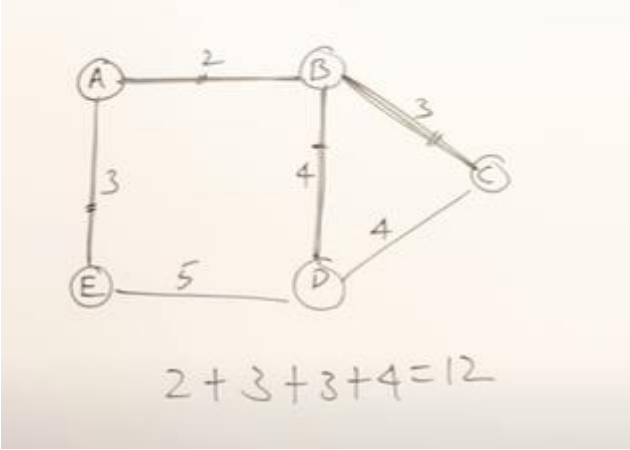
Kruskal’s Algorithm

- It falls under a class of algorithms called Greedy Algorithm that find the local optimum in the hopes of finding a global optimum.
- We start from the edges with the lowest weight and keep adding edges until we reach our goal.
- The steps for implementing Kruskal's algorithm are as follows:
  - Sort all the edges from low weight to high
  - Take the edge with the lowest weight and add it to the spanning tree. If adding the edge created a cycle, then reject this edge.
  - Keep adding edges until we reach all vertices.

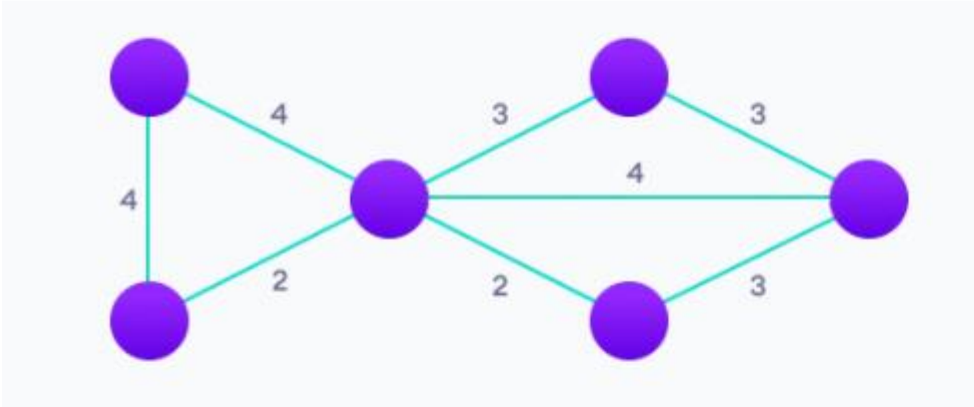
Kruskal’s Algorithm







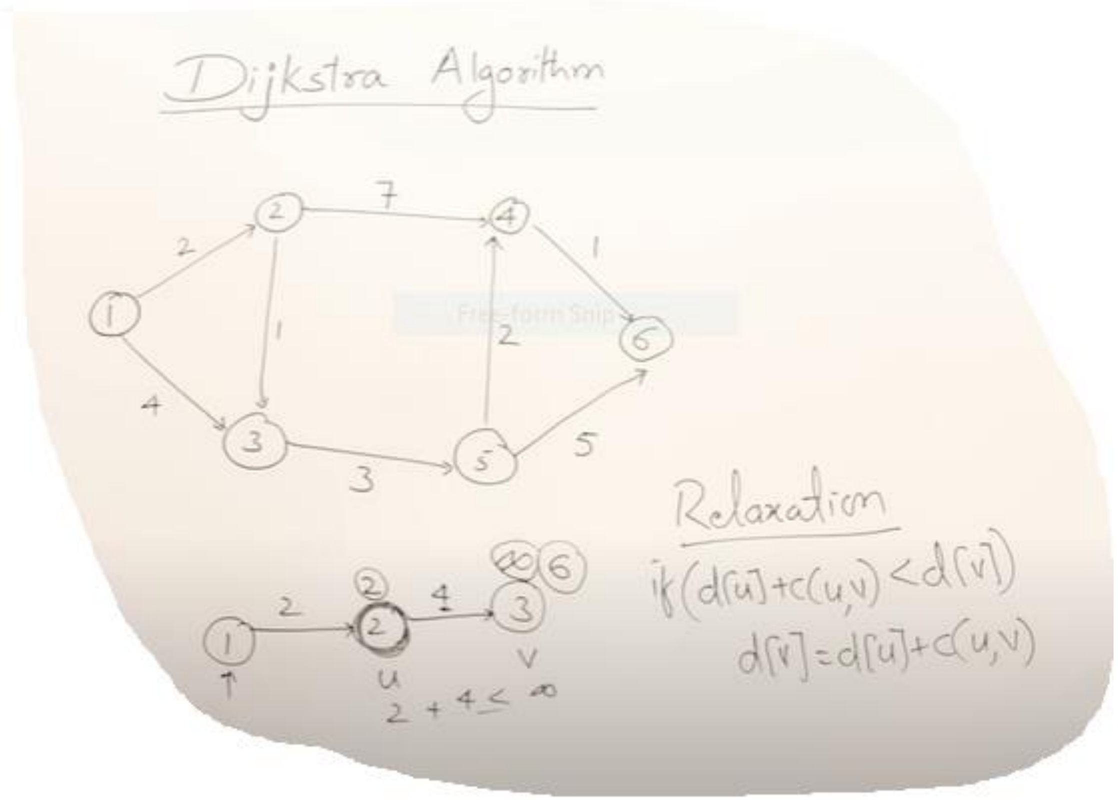
DIY:



Dijkstra Algorithm - Single Source Shortest Path

- It is a greedy algorithm that solves the single-source shortest path problem for a directed graph G = (V, E) with nonnegative edge weights, i.e., w (u, v) ≥ 0 for each edge (u, v) ∈ E.
- Dijkstra's Algorithm maintains a set S of vertices whose final shortest - path weights from the source s have already been determined. That's for all vertices v ∈ S; we have d [v] = δ (s, v). The algorithm repeatedly selects the vertex u ∈ V - S with the minimum shortest - path estimate, insert u into S and relaxes all edges leaving u.

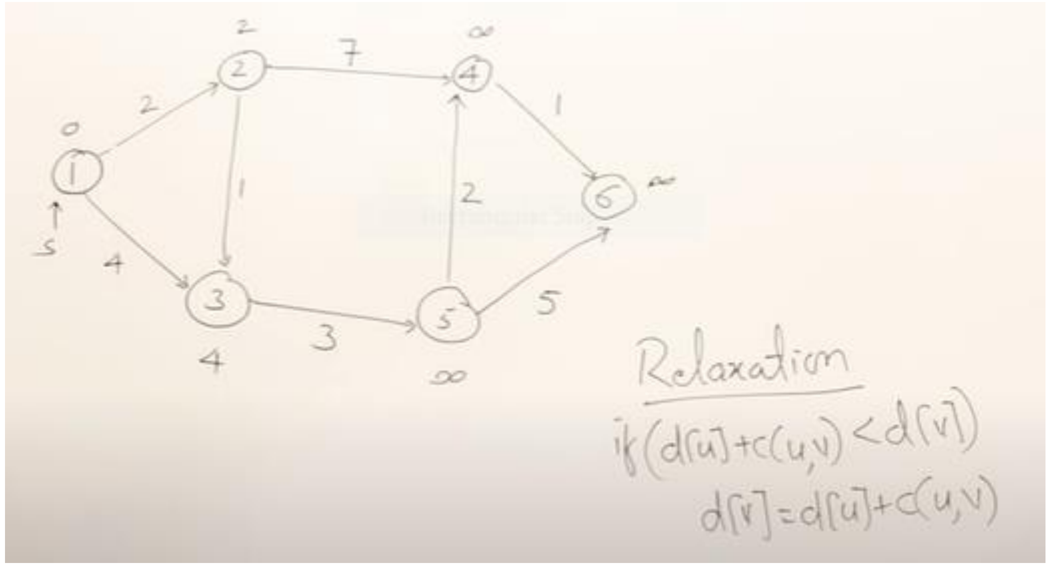
Dijkstra Algorithm - Single Source Shortest Path



Dijkstra Algorithm

Relaxation

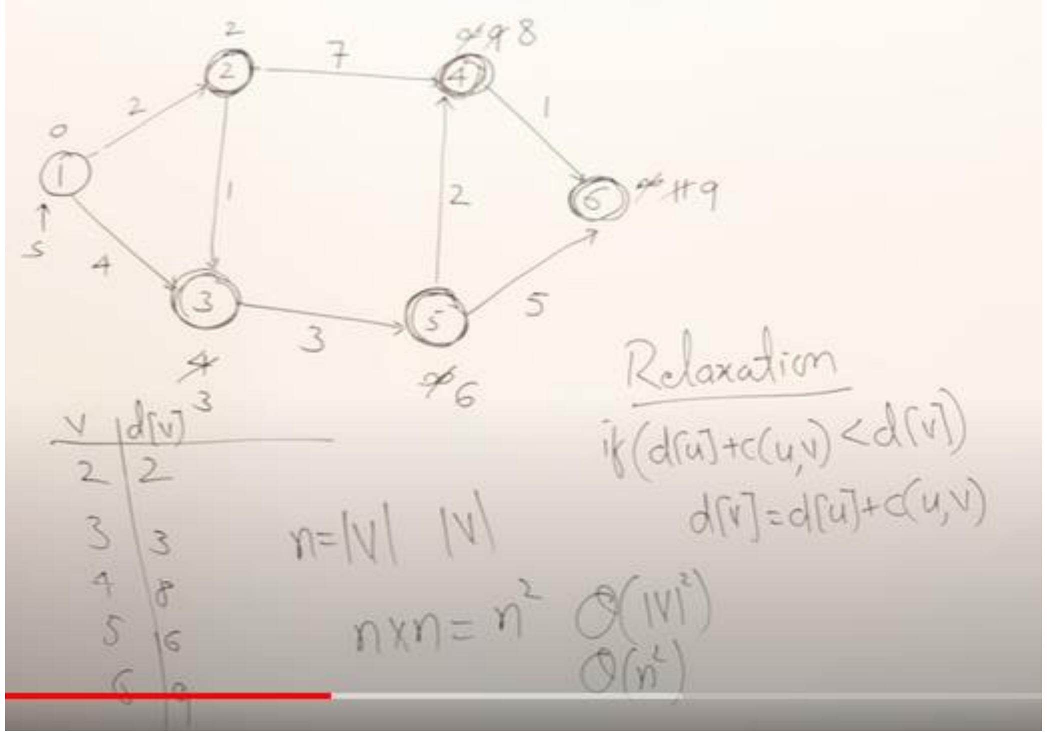
if (d[u]+c(u,v)<d[v])
  d[v]=d[u]+c(u,v)



Relaxation

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

Dijkstra Algorithm - Single Source Shortest Path



v	d[v]
2	2
3	3
4	8
5	6
6	9

$n = |V|$ $|V|$

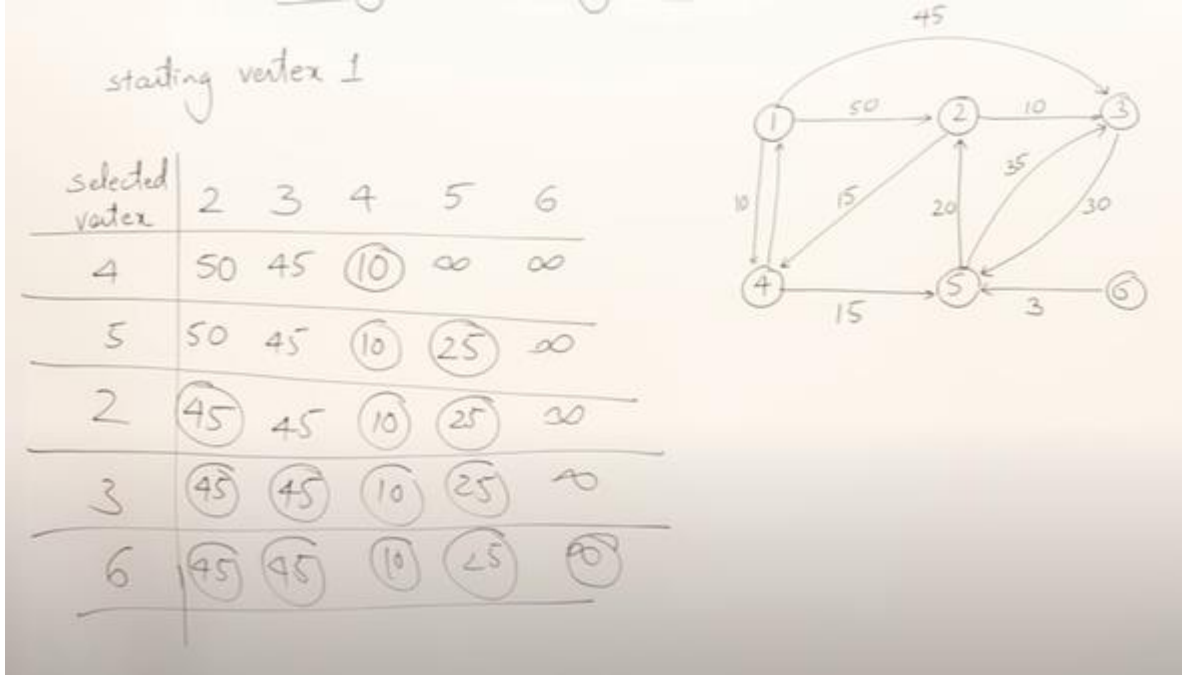
$n \times n = n^2$ $O(|V|^2)$

$O(n^2)$

Relaxation

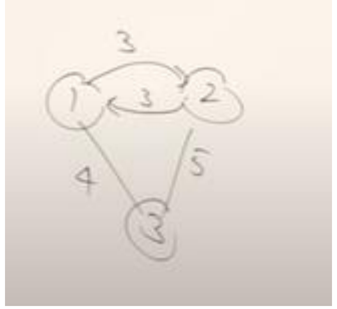
if $(d[u] + c(u,v) < d[v])$

$d[v] = d[u] + c(u,v)$



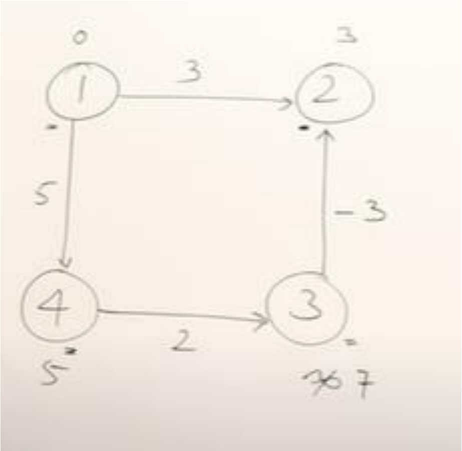
starting vertex 1

selected vertex	2	3	4	5	6
4	50	45	10	∞	∞
5	50	45	10	25	∞
2	45	45	10	25	∞
3	45	45	10	25	∞
6	45	45	10	25	∞

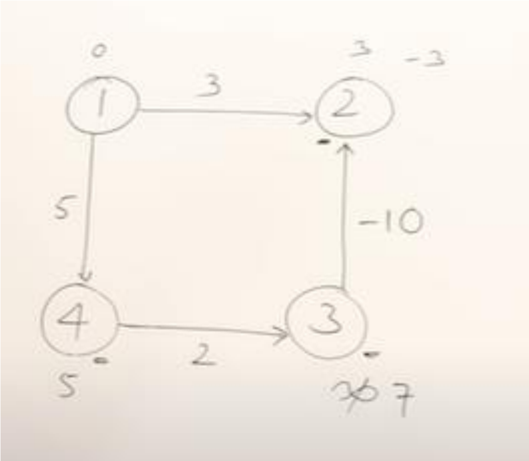


Dijkstra Algorithm - Single Source Shortest Path

Disadvantage:



0
3
1
3
2
5
-3
4
2
3
5
~~10~~ 7



0
3 -3
1
3
2
5
-10
4
2
3
5
~~10~~ 7