# (3) GREEDY METHOD

Used to solve optimisation problems
[ problems requiring either minimum (or) maximum result ]

P : A ⟶ B [ locations ]

S1 (walk), S2 (car), S3 (bus), S4 (train), S5 (flight)

suppose constraint : 12 h
train, flight → feasible solutions

suppose constraint : minimum cost

Now train → satisfies both constraints → optimal solution

---

**Strategies used for solving optimization problems :**

1. Greedy Method
2. Dynamic Programming
3. Branch and Bound

[ each are different ]

apsara

Date: __.__.__

## General Greedy Method

Algorithm Greedy (a, n)
{
  for i = 1 to n do
  { x = select (a);
    if Feasible (x) then
      solution = solution + x;
  }
}

n = 5  a

| | | | | |
|---|---|---|---|---|
| $a_1$ | $a_2$ | $a_3$ | $a_4$ | $a_5$ |
| 1 | 2 | 3 | 4 | 5 |

## 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 |

m → capacity of bag

profits → obtained on selling an item

⇓

maximize profit by filling bags with various objects

⇓

maximization

↳ optimization problem

apsara

Date: __.__.____

$$x \; ( \underset{x_1}{\overset{0 \text{ to } 1}{\square}}, \underset{x_2}{\overset{0 \text{ to } 1}{\square}}, \underset{x_3}{\overset{0 \text{ to } 1}{\square}} \ldots\ldots ) \qquad 0 \le \square \le 1$$

$\square$ represents fraction of that object taken,

↳ objects are divisible

Approach { → take maximum profit first
→ take minimum weight first

↳ combining both

take object with highest profit/weight ratio

Given example :

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| P/w | 5 | 1.66 | 3 | 1 | 6 | 4.5 | 3 |
| | $O_1$ | $O_2$ | $O_3$ | $O_4$ | $O_5$ | $O_6$ | $O_7$ |

☆ w = 15 kg, taking $O_5$, of 1 kg & ₹ 6.
Remaining w = 14 kg, profit = ₹ 6

☆ w = 14 kg, taking $O_1$ of 2 kg & ₹10
Remaining w = 12 kg, profit = ₹16

☆ w = 12 kg, taking $O_6$ of 4 kg & ₹18
Remaining w = 8 kg, profit = ₹34

apsara

Date: __.__.__

☆ $w = 8$ kg, taking $o_3$ of 5kg & ₹15
Remaining $w = 3$kg & profit = ₹49

☆ $w = 3$kg, taking $o_7$ of 1 kg & ₹3
Remaining $w = 2$ kg & profit = ₹52

☆ $w = 2$ kg, next we must choose $o_2$
$o_2$ : 3kg & ₹5.
Taking $(2/3)^{rds}$ of $o_2$, Remaining $w = 0$
profit = $52 + (2/3)(5)$ = $\boxed{₹ 55.33}$

maximised profit

$$x = \boxed{\begin{matrix} (1 & 2/3 & 1 & 0 & 1 & 1 & 1) \\ (x_1 & x_2 & x_3 & x_4 & x_5 & x_6 & x_7) \end{matrix}}$$

↳ solution.

constraint was $\sum x_i w_i \leq 15$,
we got $\sum x_i w_i = 15$ itself

objective : max $\sum x_i p_i$

objective obtained!

[ 0/1 knapsack ↮ Greedy method!! ]

# 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 |

- ☆ As many jobs have to be completed.
- ☆ Completing a job generates a profit.
- ☆ A job is either completed before its assigned deadline / not completed at all.

$\boxed{\text{Machine}}$

Assumption: Burst Time = 1 unit

Each job takes exactly one unit of time for completion.

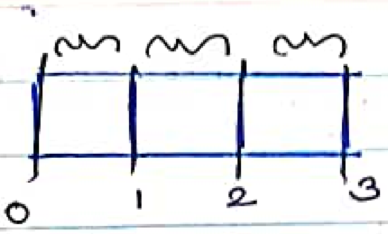
Question: Obtaining maximum profit by completing as many jobs as possible before their respective deadlines.

↓

maximization problem

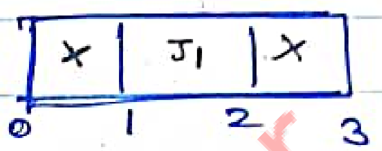
↳ optimization

↳ Greedy Method




Slots ⟹ choose job with maximum profit.

* selecting $J_1$
profit = 20



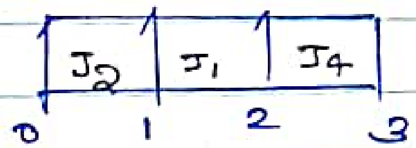

profit = 20

* selecting $J_2$
profit = 15


profit = 35

* selecting $J_3$ → deadline 1 x
cannot be scheduled

* selecting $J_4$
profit = 5




profit = 40

* selecting $J_5$ → all slots over x
cannot be scheduled.

$\{J_2, J_1, J_4\}$ (or)
$\{J_1, J_2, J_4\}$ are possible sequences
↓
Total maximized profit = 40

## Tabular representation

| Job considered | slot | soln | 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] | $J_1, J_2, J_3$ | 20+15+5 |
| $J_5$ × | [0,1] [1,2] [2,3] | $J_1, J_2, J_3$ | 20+15+5 |

Total profit = 20+15+5 = 40

## Another example

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 |

↓

maximum deadline = 4

| 0–1 | 1–2 | 2–3 | 3–4 |
|---|---|---|---|
| $J_4$ | $J_3$ | $J_1$ | $J_2$ |
| 20 | 25 | 35 | 30 |

profit = 110 → (maximised)

# HUFFMAN CODING

Message → BCCABBDDAECCBBAEDDCC

Data can be compressed and sent to reduce the cost of transmission.

Normal | Fixed size encoding | variable size encoding

$\underbrace{\text{variable size encoding}}_{\text{Huffman coding}}$

Length of above message = 20

Transferred in ASCII codes [8 bit]

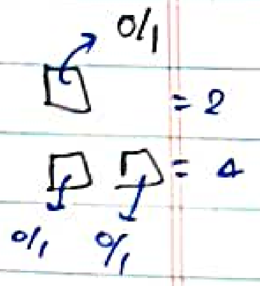
A = 65 = 01000001

B = 66 = 01000010

⋮

Totally 20 × 8 = 160 bit message

Fixed size encoding




| character | count/ Frequency | code |
|---|---|---|
| A | 3 | 000 |
| B | 5 | 001 |
| C | 6 | 010 |
| D | 4 | 011 |
| E | 2 | 100 |
| | 20 | |

new codes assigned

$20 \times 3 = 60$ bits only required

[ when compared to 160 bits ]

but

table also needed to store

$\underbrace{\text{characters}}_{5 \times 8}$ & $\underbrace{\text{corresponding codes}}_{5 \times 3}$

$40 + 15 = 55$ bits for table

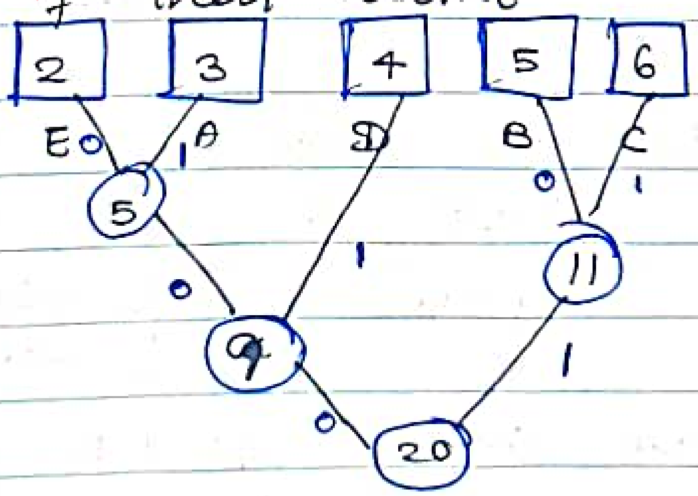
$60 + 55 = \underbrace{\boxed{115 \text{ bits}}}_{\text{smaller}}$ vs $\underbrace{160 \text{ bits}}_{\text{large}}$.

variable size encoding : Huffman code

↳ follows optimal merge pattern

⇓

Arrange alphabets in an increasing order of their count




[ optimal merge pattern tree ]

Left → mark 0

Right → mark 1

Go from parent to child & record numbers on branches.

A → 3 count, 001 code, 3 bits
B → 5 count, 10 code, 2 bits
C → 6 count, 11 code, 2 bits
D → 4 count, 01 code, 2 bits
E → 2 count, 000 code, 3 bits

variable size codes

B C C A B B D D A E C C ...
10 11 11 - - - - -

size of message = $\sum f_i b_i = \sum f_i d_i$

no. of bits = distance from root

$= (3\times3) + (5\times2) + (6\times2) + (4\times2) + (2\times3)$

= 45 bit message

For the table and tree

$\underbrace{(5\times8)}_{\text{characters}} + \underbrace{(12)}_{\text{total bits of code}} = 52 \text{ bits}$

Total size = 45 + 52 = 97 bits.

very small

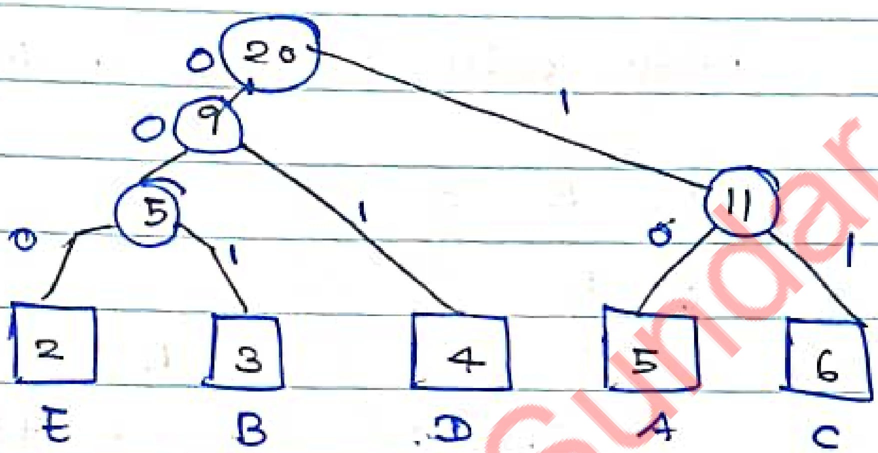
apsara

Date: __.__.____

Encode the given message :

B C C D A C C B D A

B C C D E A A E D A




001 11 11 01 001 11 11 10 01 001

10 11 11 01 000 001 001

000 01 001

encoded message

## OPTIMAL MERGE PATTERN

List A : 3 8 12 20

List B : 5 9 11 16

merge [ like in merge sort ]

List C : 3 5 8 9 11 12 16 20

[ sorted ]

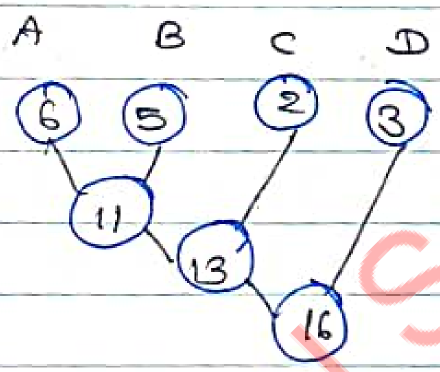
Time to merge : 4 elements + 4 = 8 units of time

# Merging of many lists

List $\rightarrow$ A B C D

Sizes $\rightarrow$ 6 5 2 3

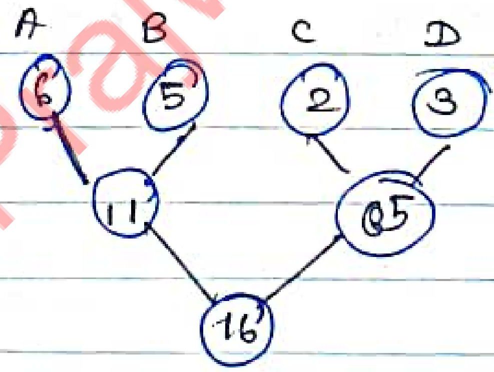
Merge only two at a time




11 + 13 + 16

= 40 unit of time

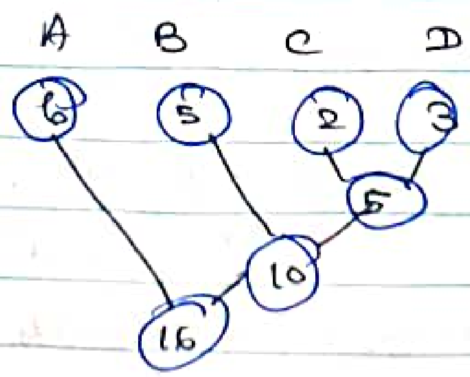
Another way :




11 + 5 + 16

= 32 units of time

Another way :




5 + 10 + 16

= 31 units of time

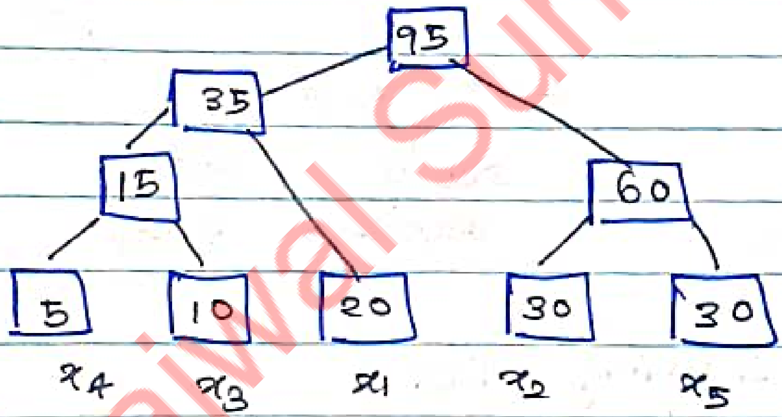
**Greedy Approach :** Always merge a pair of small sized lists to get the best result.

↳ Total merging time is reduced.

**combine file names with sizes :**

Lists → $x_1$ $x_2$ $x_3$ $x_4$ $x_5$

sizes → 20 30 10 5 30




Total cost of merging = sum of non-leaf nodes = 15 + 60 + 35 + 95 = 205 (minimum)

(or)

$\sum_1 l_i d_i$    $l_i$ → value of leaf node

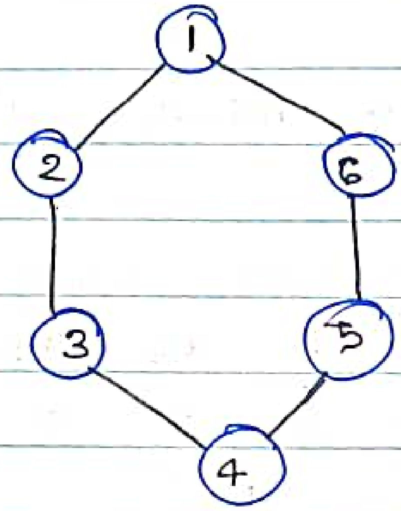
$d_i$ → distance of leaf from root

$= (3)(5) + (3)(10) + (2)(20) + (2)(30) + (2)(30)$

$= 15 + 30 + 40 + 60 + 60$

$= 205$ same result.

# MINIMUM COST SPANNING TREE



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

$E = \{(1,2), (2,3), (3,4), (4,5), (5,6), (6,1)\}$

$G = (V, E)$

set of vertices — set of edges



## Spanning Tree

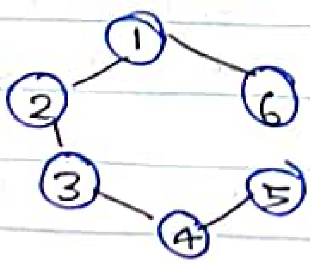
↳ A subgraph of graph G such that

- $V = |n| = 6$ vertices
- $E = |V| - 1 = 5$ edges
- no cycle in the subgraph

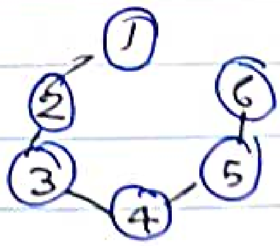
$S \subseteq G$, $S = (V', E')$

with $V' = V$

$E' = E - 1$



(or)



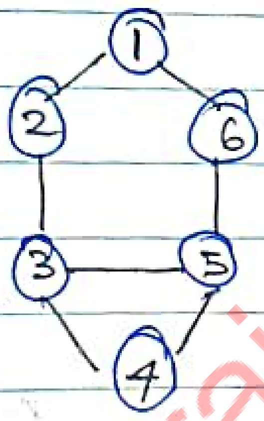
## No. of spanning trees

In given example

6 edges → choose ANY 5 edges → spanning tree is formed.

→ $^{6}C_5$ = 6 spanning trees

## Another Example




7 edges, 6 vertices

$^{7}C_5$ = 21 edges

2 cycles of size < 6

21 − 2 = 19 spanning trees

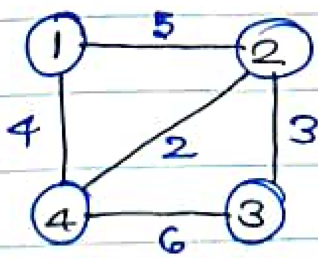
General Formula

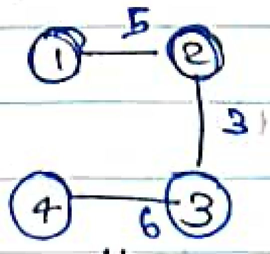
$^{|E|}C_{|V|-1}$ − no. of cycles



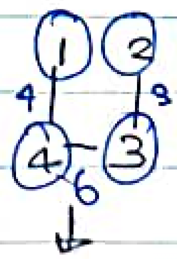
---

## case of weighted graphs

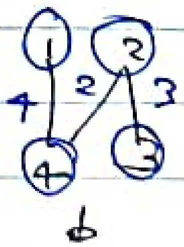



cost = 14




13

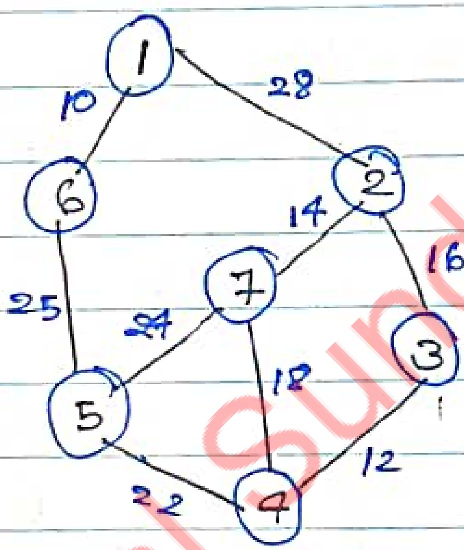



9

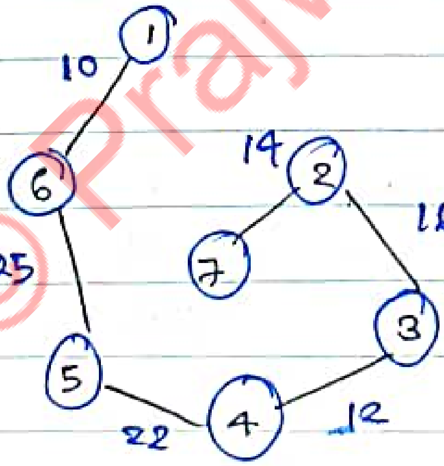
Greedy methods to find MST

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

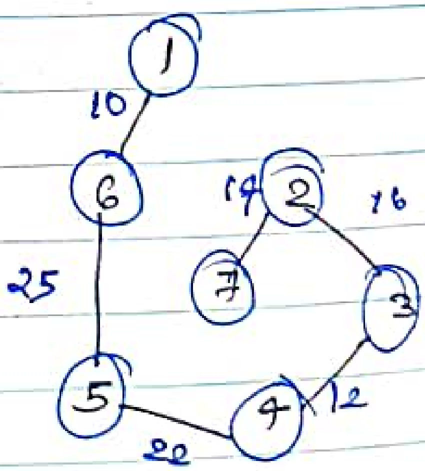
Prim
↓
select an edge always connected to the current tree



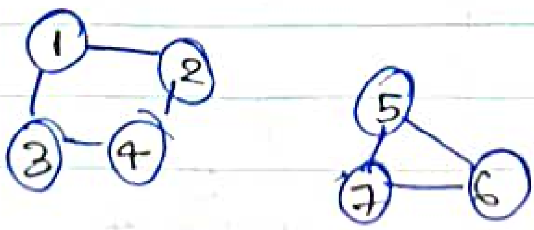
Kruskal
↓
select an edge [any]
↓
ensure no cycle is formed



cost = 99



cost = 99



⇒ Prim's algorithm cannot find MST [may find for a component]

|E| edges, |V| - 1 edges to be selected

↓

Time complexity $O(|V| \; |E|)$

$O(ne) \rightarrow$ kruskal's Algorithm

$O(n^2)$

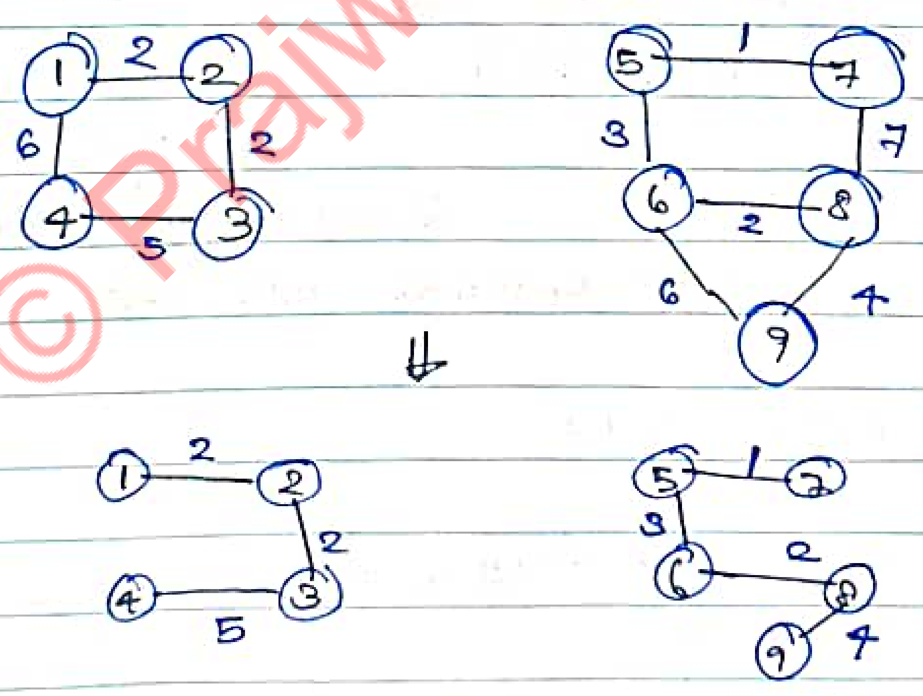
Min Heap

↓ Deleting always keeps minimum at the top → log n time

↳ $O(n \log n)$ → reduced time complexity

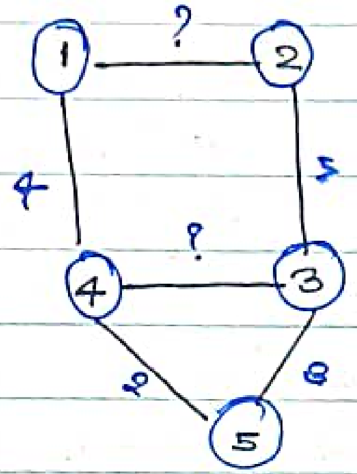
---

using kruskal's algorithm find MST



cannot find MST, but finds MST for individual connected components

Find min values of ? if all marked edges form a MST.



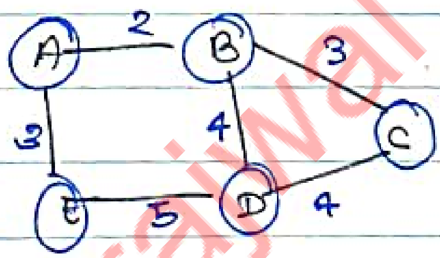

? chosen after 2 | ~~3~~.

? min → 4

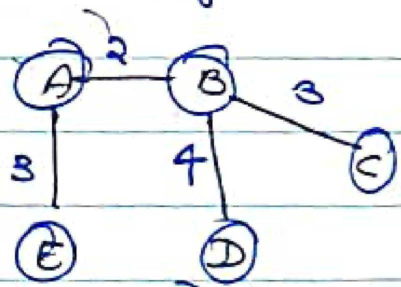
? chosen after 4 | 4 | 5

min → 6

Find minimum cost spanning tree :




⟹

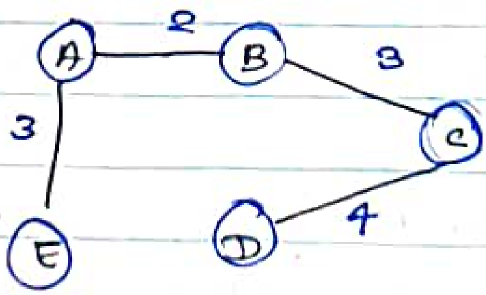



cost = 12

2 Minimal spanning trees possible

but optimal min cost remains same.

other MST :

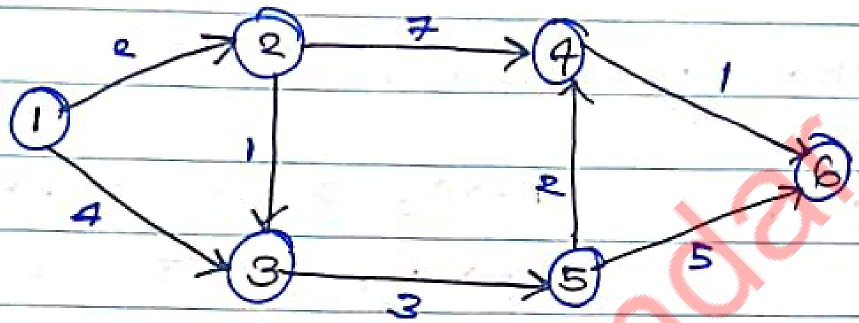



cost = 12 here also

# DIJKSTRA ALGORITHM

↓

single source shortest path problem



minimization problem

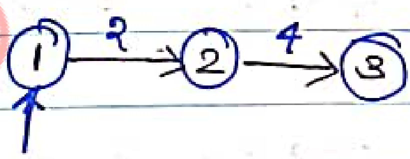
↪ optimization problem

↪ greedy approach

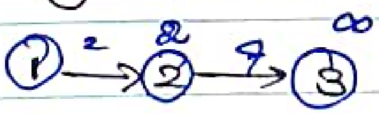
Dijkstra Algorithm → Undirected graphs ✓

→ Directed graphs ✓

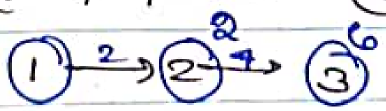
Smaller example



Initially @ vertex $v_1$



Initially @ $v_1$ , now @ $v_2$



Relaxation : updation of distance

apsara

Date: __.__.____

Relaxation

$$\text{if } (d[u] + c(u,v) < d[v])$$
$$d[v] = d[u] + c(u,v)$$

For first graph

@ $v_1$ : $1 \to 0,\ 2 \to 2,\ 3 \to 4$
$4 \to \infty,\ 5 \to \infty,\ 6 \to \infty$

select smallest
@ $v_2$ : $1 \to 0,\ 2 \to 2,\ 3 \to 3,$
$4 \to 9,\ 5 \to \infty,\ 6 \to \infty$

next smallest
@ $v_3$ : $1 \to 0,\ 2 \to 2,\ 3 \to 3,$
$4 \to 9,\ 5 \to 6,\ 6 \to \infty$

next smallest
@ $v_5$ $1 \to 0,\ 2 \to 2,\ 3 \to 3,$
$4 \to 8,\ 5 \to 6,\ 6 \to 11$

next smallest
@ $v_4$ : $1 \to 0,\ 2 \to 2,\ 3 \to 3$
$4 \to 8,\ 5 \to 6,\ 6 \to 9$

next smallest
@ $v_6$ : $\boxed{1 \to 0,\ 2 \to 2,\ 3 \to 3 \quad 4 \to 8,\ 5 \to 6,\ 6 \to 9}$

shortest distances from vertex $v_1$

Time complexity

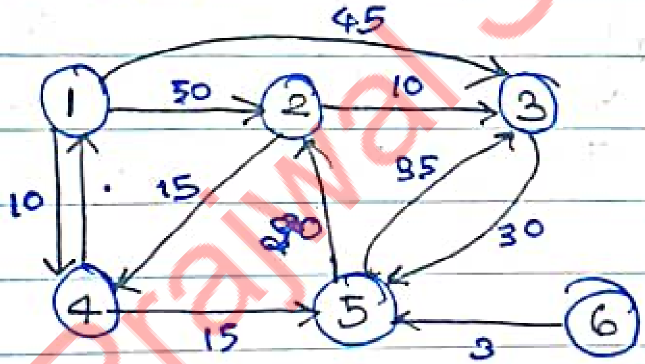
n vertices.

For complete graph, each vertex is connected to all other vertices → n

∴

$\boxed{\Theta(v^2) = \Theta(n^2)}$ is the worst case time complexity

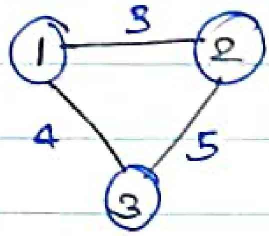
Run Dijkstra's Algorithm on the given graph :-




| Selected vertex | Vertices | | | | |
|---|---|---|---|---|---|
| | 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 | ∞ |

$\underbrace{\qquad\qquad\qquad}_{\text{shortest distances}}$

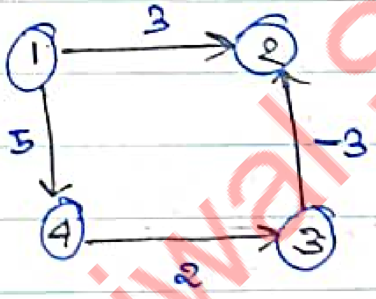
## Undirected graphs :-




| selected | 2 | 3 |
|---|---|---|
| 2 | 3 | 4 |
| 3 | 3 | 4 |

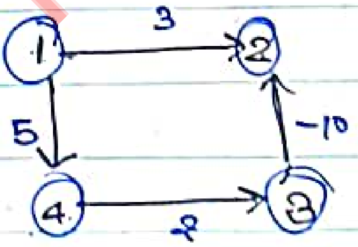
shortest distance

## Drawback of Dijkstra Algorithm




| selected | 2 | 3 | 4 |
|---|---|---|---|
| 2 | 3 | ∞ | 5 |
| 4 | 3 | ∞ | 5 |
| 3 | 3 | 7 | 5 |

shortest vertices' distances




| selected | 2 | 3 | 4 |
|---|---|---|---|
| 2 | 3 | ∞ | 5 |
| 4 | 3 | ∞ | 5 |
| 3 | 3 | 7 | 5 |

wrong

Negative edge may or may not work ⊗

as 2 : 1 → 4 → 3 → 2
−3 is correct
not 3