Find shortest path between all pair of vertices.

	Time	Space
Dijkstra →	$O(V^2)$	$O(V)$
Bellman Ford →	$O(V^3)$	$O(V+E)$

Shortest Path between a pair of vertices.

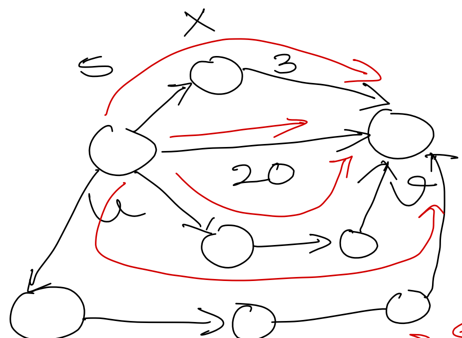
<u>Brute force</u> ⇒ $O(V^4)$

→ Find all pairs of vertices ⇒ $O(V^2)$

→ Find shortest path between each pair of vertices] ⇒ V^2 times $O(V^2)$

<u>Floyd-Warshall algorithm</u>

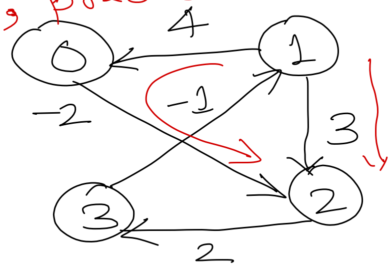
↳ Dynamic Programming



(u, v)

(u, x) (x, v)

current shortest path , predecessor



dist

	0	1	2	3
0	0	∞	-2, 0	∞
1	4, 1	0	3, 1	∞
2	∞	∞	0	2, 2
3	∞	-1, 3	∞	0

$x \rightarrow 0$ to $|V|-1 \longrightarrow V$ times

$u \rightarrow 0$ to $|V|-1 \longrightarrow V$ times

$v \rightarrow 0$ to $|V|-1 \longrightarrow V$ times

→ currDist uv = dist[u][v]

→ dist uv via x = dist[u][x] + dist[x][v]

→ if currDist uv > dist uv via x then

dist[u][v] = dist uv via x

pred[u][v] = pred[x][v]

Time complexity

$V \times V \times V$

$\Rightarrow O(V^3)$

Space complexity

$O(V^2)$

$x \rightarrow 0$ ~~0~~ ~~1~~ ~~2~~ 3

$u \rightarrow$

$v \rightarrow$ ~~0~~ ~~1~~ ~~2~~ ~~3~~ ~~0~~ ~~1~~ ~~2~~ ~~3~~ ~~0~~ ~~1~~ ~~2~~ ~~3~~ ~~0~~ ~~1~~ ~~2~~ 3

dist

	0	1	2	3
0	0	∞	-2,0	∞
1	4,1	0	~~3,1~~ 2,0	∞
2	∞	∞	0	2,2
3	∞	-1,3	∞	0

(1,2) ⇒ 3

(1,0) (0,2) ⇒ 2

4 -2

(2,1) ⇒ ∞

(2,0) (0,1) ⇒ ∞

∞ ∞

(1,3) ⇒ ∞

(1,0) (0,3) ⇒ ∞

4 ∞

(3,1) ⇒ -1

(3,0) (0,1) ⇒ ∞

∞ ∞

(2,3) ⇒ 2

(2,0) (0,3) ⇒ ∞

∞ ∞

(3,2) ⇒ ∞

(3,0) (0,2) ⇒ ∞

∞ -2

dist

	0	1	2	3
0	0	∞	-2, 0	∞
1	4, 1	0	2, 0	∞
2	∞	∞	0	2, 2
3	~~∞~~ 3, 1	-1, 3	~~∞~~ 1, 0	0

$x \rightarrow 1$

$u \rightarrow \cancel{0}\ \cancel{1}\ \cancel{2}\ 3$

$v \rightarrow \cancel{0}\ \cancel{1}\ \cancel{2}\ \cancel{3}\ \cancel{0}\ \cancel{1}\ \cancel{2}\ \cancel{3}\ 0\ 2$

$(0,2) \Rightarrow -2$

$(0,1)$ $\quad$ $(1,2)$

∞

$(0,3) \Rightarrow \infty$

$(0,1)$ $\quad$ $(1,3)$

$\Downarrow$

∞

$(2,0) \Rightarrow \infty$

$(2,1)$ $\quad$ $(1,0)$

$\Downarrow$

∞

$(2,3) \Rightarrow 2$

$(2,1)$ $\quad$ $(1,3)$

∞

$(3,0) \Rightarrow \infty$

$(3,1)$ $\quad$ $(1,0) \Rightarrow 3$

$\Downarrow$ $\quad$ $\Downarrow$

-2 $\quad$ 4

$(3,2) \Rightarrow \infty$

$(3,1)$ $\quad$ $(1,2) \Rightarrow 1$

$\Downarrow$

-1 $\quad$ 2

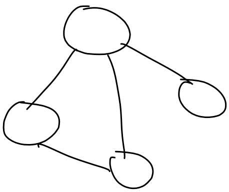
dist

	0	1	2	3
0	0	∞	-2, 0	∞
1	4, 1	0	2, 0	∞
2	∞	∞	0	2, 2
3	3, 1	-1, 3	1, 0	0

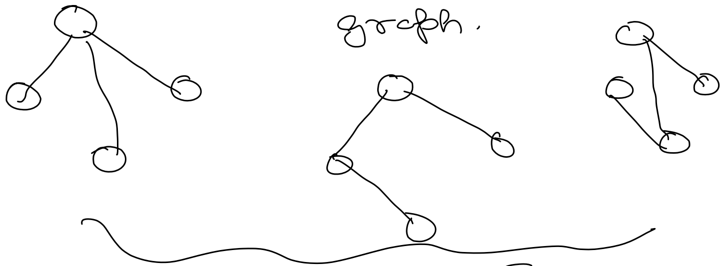
$x \rightarrow 2$

$x \rightarrow 3$

Spanning Tree ⇒ Is a tree that has all vertices of graph.



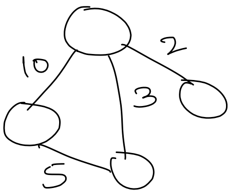
Graph



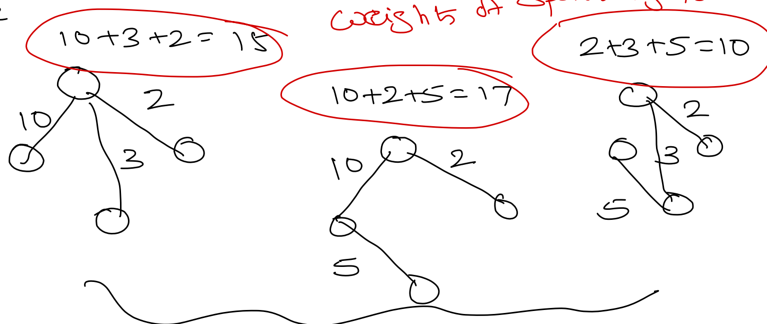
Spanning Tree

Min Spanning Tree

weights of spanning Tree

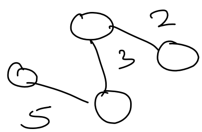


Graph



Spanning Tree

Min Spanning Tree: spanning tree with min weight



= 10

Find Min Spanning Tree

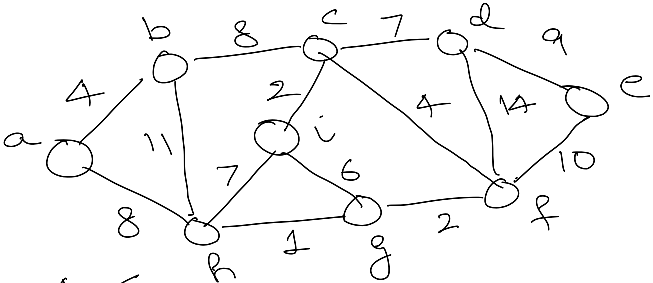
(1) Brute Force approach : Find all solutions & then pick the optimal.

O(max number of spanning trees)

For a graph with V vertices, we can have max $V^{(V-2)}$ spanning trees.

$$O\left(V^{(V-2)}\right)$$

(2) Prim's algorithm

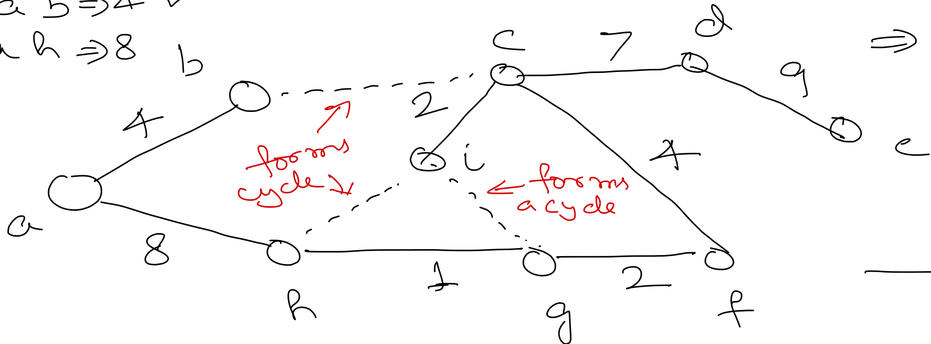


→ Start with empty tree

→ Pick a start vertex

ab ⇒ 4 ✓

ah ⇒ 8



⇒ Min Spanning Tree

~~4~~ + 8 + 1 + 2 + 4 + 7 + 9 + 2 = 37

---

bc ⇒ 8
bh ⇒ 11
hi ⇒ 7
gi ⇒ 6
fd ⇒ 14
fe ⇒ 10
fc ⇒ 4 ✓

ah ⇒ 8 ✓
bc ⇒ 8
bh ⇒ 11

bc ⇒ 8
bh ⇒ 11
hi ⇒ 7
hg ⇒ 1 ✓

bc ⇒ 8
bh ⇒ 11
hi ⇒ 7
gi ⇒ 6
gf ⇒ 2 ✓

bc ⇒ 8
bh ⇒ 11
hi ⇒ 7
gi ⇒ 6
fd ⇒ 14
fe ⇒ 10
cd ⇒ 7
ci ⇒ 2 ✓

bc ⇒ 8
bh ⇒ 11
hi ⇒ 7
gi ⇒ 6 × forms cycle.
fd ⇒ 14
fe ⇒ 10
cd ⇒ 7 ✓

bc ⇒ 8 ×
bh ⇒ 11
hi ⇒ 7 × forms cycle
fd ⇒ 14
fe ⇒ 10
de ⇒ 9 ✓

bh ⇒ 11
fd ⇒ 14
fe ⇒ 10

Prim's Minimum Spanning Tree Algorithm

- Initialise minimum spanning tree with any vertex from graph. → 1
- While we don't have |V| vertices in minimum spanning tree do → V times
    - Find all edges in graph that connect tree to the newly added vertex. → E
    - Add the edge with minimum weight to the tree, such that it does not for a cycle.

DFS ⇒ $O(V+E)$

If you visit a vertex already visited ⇒ cycle

log E → Priority Q / Min Heap

E → Linear Search

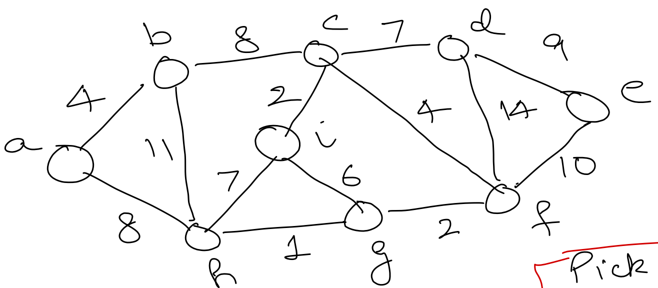
E log E → Sorting

$$V * (E + \log E + V + E)$$

$$VE + V\log E + V^2 + VE$$

$$\Rightarrow (V^2 + VE)$$

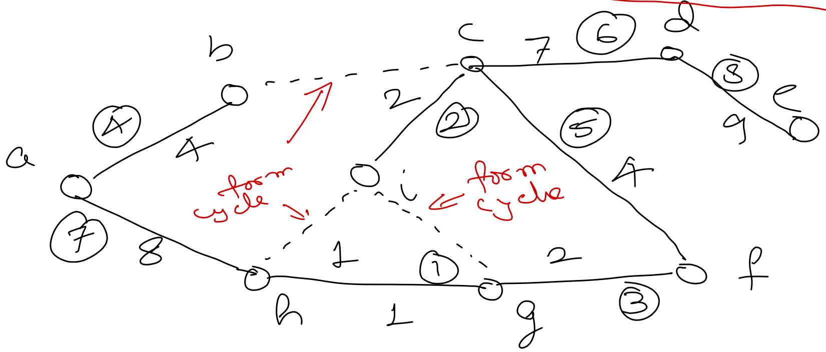
Kruskal's Min Spanning Tree



$|V| = 9$

gh ⇒ 1 (1)
ci ⇒ 2 (2)
fg ⇒ 2 (3)
ab ⇒ 4 (4)
cf ⇒ 4 (5)
gi ⇒ 6 ✗ ← cycle
cd ⇒ 7 (6)
ih ⇒ 7 ✗ → cycle
ah ⇒ 8 (7)
bc ⇒ 8 ✗
de ⇒ 9 (8)
ef ⇒ 10
bh ⇒ 11
df ⇒ 14

Pick |V| − 1 edges



Kruskal's Minimum Spanning Tree Algorithm

- Initialise minimum spanning tree to empty. → 1
- Sort all edges in increasing order of their weight. → $E \log E$
- While we don't have $|V| - 1$ edges in minimum spanning tree → V times
  - Get the edge with minimum weight. → 1
  - Add edge to minimum spanning tree if that do not result in cycle.

Size

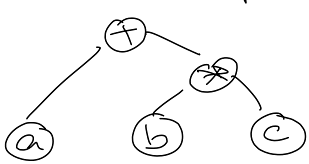
$O(V+E)$

$$E \log E + V(V+E)$$
$$= E \log E + V^2 + VE \Rightarrow O(V^2 + VE)$$

Expression Tree

$a + b * c$

- → Inorder ⇒ infix ⇒ $a + b * c$
- → Pre order ⇒ Prefix ⇒ $+ \; a \; * \; b \; c$
- → Post Order ⇒ Postfix ⇒ $a \; + \; b \; c \; *$



Algorithm Design

→ Divide and Conquer : If we can divide larger problem into smaller problems such that solution of smaller problems will give us solution of larger problem.

- ↳ Binary Search
- ↳ Quick Sort
- ↳ Merge Sort

Parts of D & C

1. Divide : Divide larger problem into smaller problems.
2. Conquer : Solve smaller problems, until smaller problems are base case.
   → Normally its done recursively.
3. Combine : Combine solutions of smaller sub problem to give solution of larger problem.

Adv : They can be parallalize.
Disadv : Recursion ⇒ Needs extra memory.

→ Greedy algorithms

- ↳ Shortest Path : Dijkstra
- ↳ Min Spanning Tree : Prim's / Kruskal's

⇓

we can find optimal solution by picking best available choice at each step.

Knapsack Problem

→ objects
→ weight
→ cost

Knapsack
↓
Capacity

objects →	A	B	C
weights →	10	3	5

capacity ⇒ 10

0/1 Knapsack

Cost/Prof → 10 5 6

w/c → 1 0.. 0....

Brute Force algorathim ⇒ Find all possible solutions & then pick the optimal one.

⇓

Time Complexity is very high.

A B C ⇒ $O(2^n)$

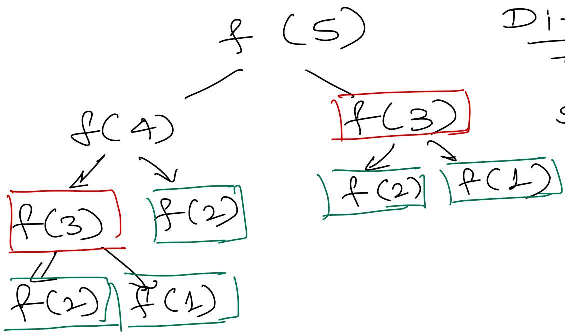
AB BC AC

Dynamic Programming → Overlapping sub problem → optimal substructure

1 1 2 3 5..

→ Find n^{th} fibonacci term.

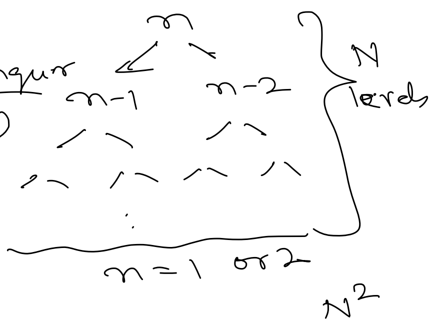
$$f(n) = \begin{cases} 1, & \text{if } n <= 2 \\ f(n-1) + f(n-2) \end{cases}$$



Divide & Conquer

Time ⇒ $O(n^2)$

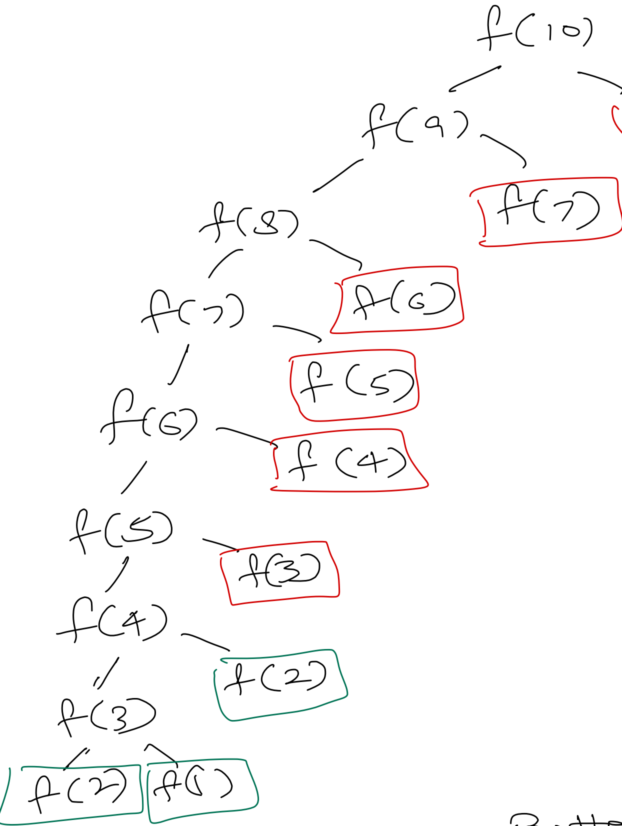
Space ⇒ $O(n)$ for recursion



→ Top down : Memoization ⇒ Remember solution of repeated sub problems.

→ Bottom up : Table

optimal substructure ⇒ optimal solution for the problem can be obtained from the optimal solution of sub problems.



Top Down

n levels

$\Rightarrow O(n)$

Time Complexity

Space Complexity

$O(n) \leftarrow$ Recursion

$O(n) \leftarrow$ Memoization

Bottom up

$O(1) \leftarrow$ Time Complexity

$O(n) \leftarrow$ Space complexity

Memory for Table.

$O(n) \leftarrow$ Pre Processing Time Complexity

Divide & Conquer $\Rightarrow$ can be improved via Dynamic Programming

Binary Search $\Rightarrow$ D & C

$\Rightarrow$ Can't use Dynamic Programming. As no overlapping subproblems.

Backtracking $\Rightarrow$ Involves recursion

→ We build solution step by step
→ Each step we discard ~~that~~ do not result in solution

→ Decision Problem $\Rightarrow$ Game Tree : Tic-Tac-Toe
→ Enumeration Problems $\Rightarrow$ Tree traversal
→ Optimization Problems $\Rightarrow$ Finding the best solution.

→ Sudoku Puzzle
→ Solving Maze
→ Knight tour