★ $QS(n)$

$QS(\frac{n}{3})$ $QS(n/7)$

$QS(\frac{n}{3^2})$ $QS(\frac{n}{3.7})$ $QS(\frac{n}{3.7})$ $QS(\frac{n}{7.7})$

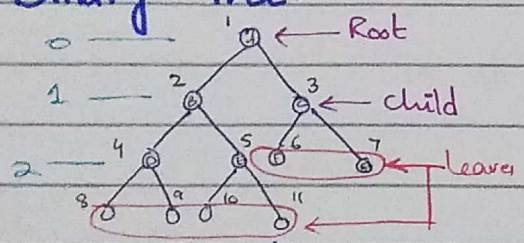
FINALS ★

Heap

30th Nov, 2020.

1. Binary Tree.
2. Complete B.T & Full Binary Tree.
3. Definition of Heap.
4. Create Heap | Insert Heap.
5. Delete heap.
6. Heap sort.
7. Heapify.

1. Binary Tree.



height $= 2^h - 1$ = no. nodes.

Properties : ⇒ level based Insertion

- max number of children are '2'
- Insertion at next available location.

• Nodes having no child are called leaves.

1	2	3	4	5	6	7	8
A	B	C	D	E	F	G	H

Heap.

2nd Dec. 2020.

1. Complete Binary Tree.
2. Array Representation of Binary Tree.
3. Full Binary Tree.
4. Heap | properties.
5. Insert | Remove.
6. Heapify.

★ Leftchild → even location

i C_L $= 2i$

★ Right → odd location

i C_R $2\times(i+1)$

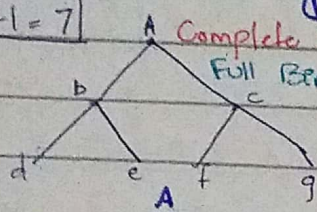
★ Parent of child $\lfloor \frac{i}{2} \rfloor$

Scanned with CamScanner

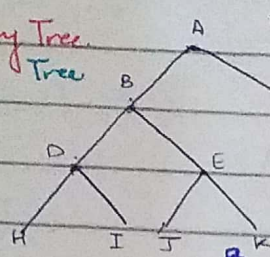
No. of nodes = $2^h - 1$ → levels of tree → height

Complete Binary Tree

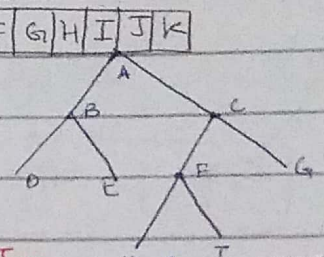
$2^3 - 1 = 7$



Complete Binary Tree
Full Binary Tree



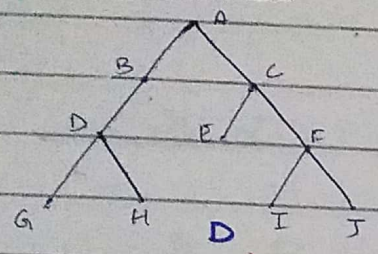
A	B	C	D	E	F	G	H	I	J	K



$2^4 - 1 = 15$

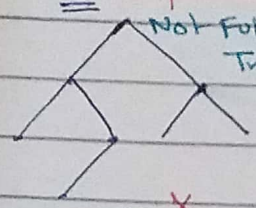
A

A	B	C	D	E	F	G



D ✗

B Complete Binary Tree
Not Full Binary Tree



E ✗

A	B	C	D	E	F	G	H	I	J

CR → B

A	B	C	D	E	F	G	H	I

Full Binary Tree.

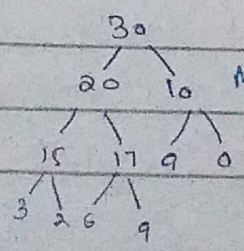
1. Complete Binary Tree.
2. No. of nodes must be exactly $2^h - 1$ where h = height of Tree.

Heap — Properties.

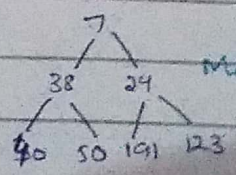
- commonly used structure.
- efficient insertion.

1. Must be a Complete Binary Tree. (after Insertion & Removal)
2. a. Max Heap (every node is greater than/equal to its childs)
   b. Min Heap (Node is smaller than/equal to its childs).

Example:



Max heap



Min heap

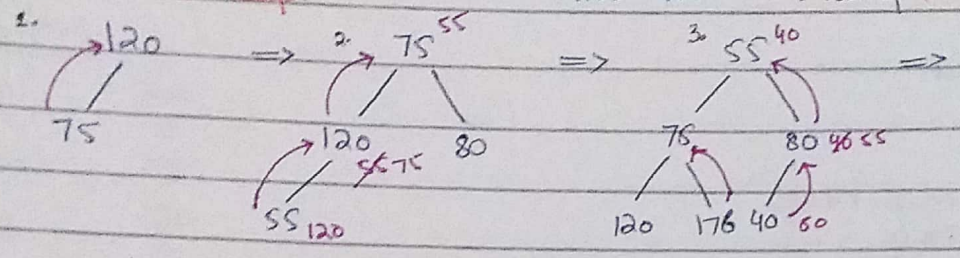
Scanned with CamScanner

for ($i = \frac{n}{2}$; $i >= 1$; $i--$) — $\frac{n}{2}$ → $O(n)$ Time complexity.

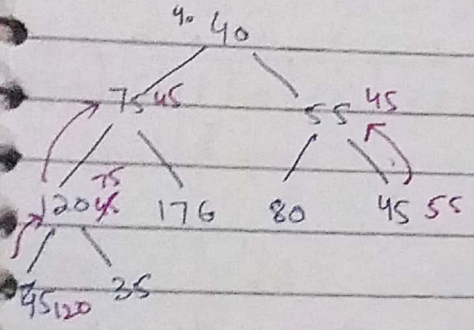
Build / Create Heap.

Min Heap.

Insertion. → Push up (bottom up)



1. | 120 | 75 |
2. | 75 | 120 | 80 | 55 |
   | 75 | 55 | 80 | 120 |
3. | 55 | 75 | 80 | 120 | 176 | 40 |
4. | 40 | 75 | 55 | 120 | 176 | 80 | 45 |
   | 40 | 75 | 45 | 120 | 176 | 80 | 55 |



Worst Case.

2^n

$\log_2 n$

1 Element insertion — $O(\log_2 n)$ Time Complexity.

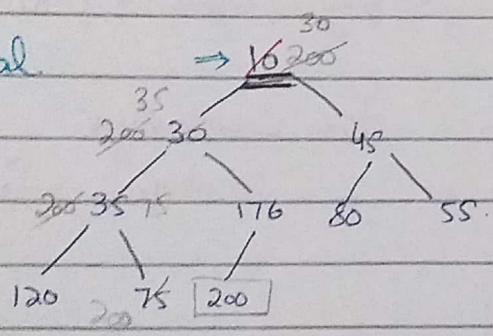
n Element insertion — $O(n \log n)$

Removal

from root ↓

push down

(Top down)



10	30	45	35	176	80	55	120	75	200

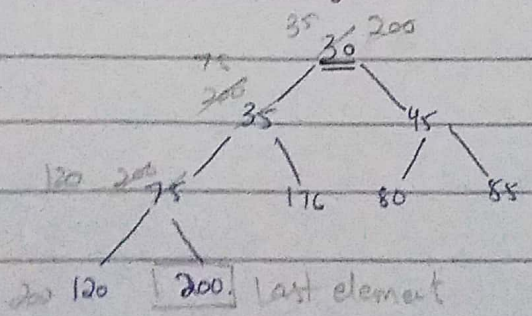
minheap.

30	35	45	75	176	80	55	120	200	10

Time complexity

$O(1)$

Restoration → $O(\log n)$. = $O(\log n)$.



30	35	45	75	176	80	55	120	200	10

deleted element

35	75	45	120	176	80	55	200	30	10

sorted element

Time complexity.

n

$O(\log n)$ — 1 element

Heap sort → $O(n \log n)$

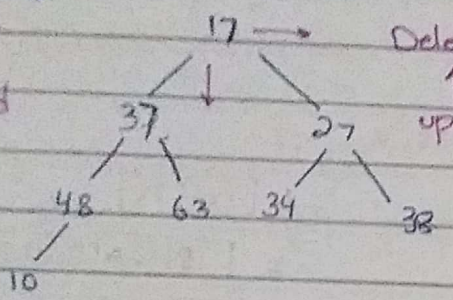
Min heap — desc order — asce

Max heap — asce order — desc

4th Dec. 2020

Heapify

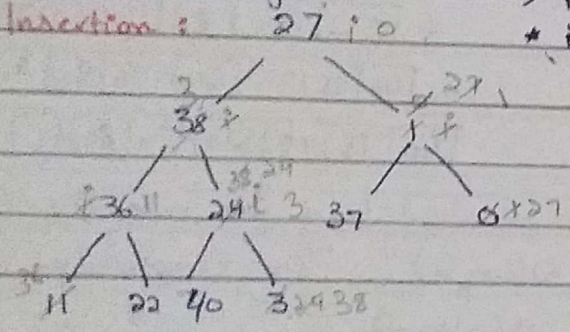
Insertion ↓ downward



Deletion ↑ upward.

• Complete Binary Tree ⟶ min heap.

Insertion :



* $i = \frac{n}{2}$

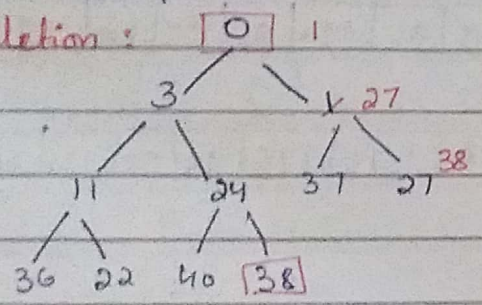
1	2	3	4	5	6	7	8	9	10	11
27	38	1	36	24	37	0	11	22	40	38

* no. of nodes $\leftarrow \frac{n}{2} = \frac{11}{2} = 5$

0	3	1	11	24	37	27	36	22	40	38

Top down process:

Deletion :



Heapify Insertion → $T(n) = O(\log n)$.

Time Complexity :

$O(n \log n)$ — for n elements.

$0 \rightarrow 9$

10^3	10^2	10^1	10^0
4000	700	80	9
	4	7	8	9
	3	2	1	0

Sorting Algorithms

- Bubble
- Selection
- Insertion
- Merge
- Shell
- Quick

Comparison Base

- Best case : $O(n \log n)$
- worst case : $O(n^2)$

- Radix sort
- Counting sort.

Scanned with CamScanner

Radix Sort.

Examples

120 100 18 357 0 75 91 115 786 8 9 420 7

Stack Unit stack.

for (1 → n)
O(n)

0	1	2	3	4	5	6	7	8	9
420	91				115		7	8	
0					75	786	357	18	9
100									
120									

420 0 100 120 91 115 75 786 7 357 8 18 9

hundred stack.

0	1	2	3	4	5	6	7	8	9
9									
8									
7									
100		120						18	
0	115	420			357		75	786	91

91 18 786 75 357 120 420 115 9 8 7 100 0

f (1 → n)
O(n).

counting sort

Counting Sort.

⇒ Ideally spread data.

Suppose → elements 11 → 29.

13 5 7 5 9 7 7 13 11 11 23 27 13 11

7 9 9 5 19 19 13

split = 5, 27 = 22 (smallest, largest)

Distinct elements = 13, 5, 7, 9, 11, 23, 27, 19.

largest element = 27.

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24 ... 27
Freq.						~~1~~ 2		~~1~~ 3		1		2		~~2~~ 3										1	

Freq[A[i]]++.

Ensure:

- Stability
- Sequence

7th Dec, 2020

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
3	4	7	4	3	1	0	4	3	7	7	4	3	0	2	1	1	2	3	2

n =

Range = minima → maxima.

k = 0 → 7 = 8 → Distinct no. of items.

	0	1	2	3	4	5	6	7
FC	2	3	3	5	4	0	0	3

FC[0] − 1

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
0	0	1	1	1	2	2	2	3	3	3	3	3	4	4	4	4	7	7	7

	0	1	2	3	4	5	6	7
FC	2	5	8	13	17	17	17	20

★ FC[0] − 1 = index.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
0	0	1	1	1	2	2	2	3	3	3	3	3	4	4	4	4	7	7	7

Scanned with CamScanner

```
CountingSort (A, n)
    Minima = getMin(A, n)  ---> O(n)
k = Maxima = getMax(A, n)  ---> O(n)   } O(n)
int FC[k+1]        int B[n]
  for( i=0 ; i< k ; i++)
      FC[A[i]]++ ;                    || n
  for (i=1 ; i<=k ; i++)              || k
      FC[i] += FC[i-1] . // update FC.
  for (i=n-1; i>=0 ; i--)
      FC[A[i]] = A[i].                | n
  for (i=0 ; i<n; i++)
      A[i] = B[i].                    | n
```

$n + k$.

$T(n) = O(5n + k)$

$T(n) = O(n + k)$

$S(n) = k, n$

$S(n) = O(n)$.

0	-5	2	-9	8	3	-9	2

↑ min

9	4	11	0	17	12	0	11

Example ~ Counting Sort.

9th Dec, 2020

0	1	2	3	4	5	6	7	8	9	10	11
1	2	8	4	9	2	4	9	9	8	1	1

	0	1	2	3	4	5	6	7	8	9
FC	0	3	2	0	2	0	0	0	2	3

	0	1	2	3	4	5	6	7	8	9
FC	0	3	5	5	7	7	7	7	9	12

0	1	2	3	4	5	6	7	8	9	10	11
1	1	1	2	2	4	4	8	8	9	9	9

Example: Radix Sort.

0	1	2	3	4	5	6	7	8	9	10	11
371	283	417	037	048	053	036	024	038	171	397	420
									i	j	k

	0	1	2	3	4	5	6	7	8	9
FC	1	2	0	2	1	0	1	3	2	0

Iteration = max digits.

$\boxed{A[i] \% 10}$

	0	1	2	3	4	5	6	7	8	9
updated FC.	1	3	3	5	6	6	7	10	12	12

0	1	2	3	4	5	6	7	8	9	10	11
420	371	171	283	053	024	036	417	037	397	048	038

$A[i] \div (Base)^{Iteration}$

```
for ( itr=1 ; LV ÷ itr>0 ; itr = itr*10 ) {
        CountingSort (A, n, itr)
    }.
```

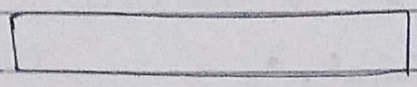
/ *

Hashing

11th Dec, 2020

Search $O(n)$ → without any algorithm

Sorted Array

* Binary Search  $O(\log n)$ → sorted by using any sorting algorithm.

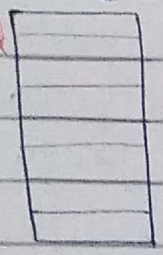
Search Technique
↓ Storage

Data is stored in the form which can be search with faster and minimum search effort, Insertion may be expensive.

Objective: Retrieval is efficient.

Hash Table

Mathematical Function
↓
store value based on Mathematical Function



- 1 location → 1 value. (1-1 mapping)

Data value → 1 location ↔ Ideal case.

No Collission.

- Many DV — 1 location.

Collission occurs.

* Collission Resolution : Next free location. (after collission occurs)

Example: 7, 3, 9, 11, 6, 5, 4, 70, 11

k
key

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	------	70
-1	-1	-1	3	4	5	6	7	-1	9	-1	11	-1	-1	-1	------	70

$h(k) = k$. (hash function) ⟹ Simple hashing techniques.

* Storage is wasted (Disadvantage)

Advantage:

* 1-1 Mapping (until & unless we have repeated values) → array is initialized with -1

|| → M-1 Mapping
└ collission occurs

M - prime number
↓
size of array

Scanned with CamScanner

Hash Function: $h(k) = k \% \text{ size of Table}$

$h(k) = k \% M$ — Modular Division ∵ M is size of array

$m >= \text{no. of keys}$

Example: 7, 3, 9, 11, 6, 5, 4, 70, 11, 13.

0	1	2	3	4	5	6	7	8	9
70	11	11	3	4	5	6	7	13	9

Collission Occurs → 11, 13

next available location → move Linearly

★ $m = 10$

k	h(k)
7	7%10 = 7
3	3%10 = 3
9	9%10 = 9
11	11%10 = 1
6	6%10 = 6
5	5%10 = 5
4	4%10 = 4
70	70%10 = 0
11	11%10 = 1
13	13%10 = 3

→ Hash Complexity.

Worst Case : $O(n)$

Best / Average case : $O(1)$

★ Effect of Array size:

$m = 11$ — Prime no. → unique / distinct value. → reduce collission.

0	1	2	3	4	5	6	7	8	9	10
11	11	13	3	4	5	6	7	70	9	-1

11, 70

String — Key Values.

Text — Integer

Farheen

Fayal

Adil

Aniza.

ASCII Code

$A + D + I + L =$

hash value same → max collission

In string:

TEAM

MEAT

MATE

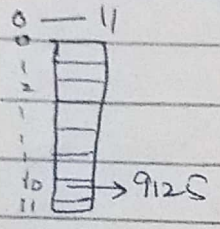
TAME

$\Rightarrow AC(T) \times 10^3 + AC(E) \times 10^2 + AC(A) \times 10^1 + AC(M) \times 10^0$

Scanned with CamScanner

Types of Hash Functions.

0. Simplex $h(k) = k$
1. Modular Division $h(k) = k \% m$ $m > \text{No. of } k$ $m \in P$
2. Mid Square Hashing $k \rightarrow k^2$



$k = 9125$

$k^2 = 83265625$

Mode

$65 \% m = 65 \% 11 = 10$

3. Subtraction → Bandwidth (BW) = maxima − minima

$h(k) = k - BW$ Assume: $\underbrace{1007 - 1100}_{m = 100 \text{ employees}}$ $BW = 93$

$1007 - 93 = 1014 \% 100$

$= 14^{th}$ location.

4. Folding Method.

→ shift - Folding → size of Address = 3

$h(k)$

- ★ Parts of key acc to size of Address
- ★ Accumulate parts
- ★ Ans is $h(k)$ after discarding right most value

key = 00978 4632

632
784
009
1425
↓
$h(k)$

→ Fold Boundaries.

- ★ Parts of key $k = 009784632$
- ★ Reverse value 236
- ★ Add up 487
- ★ 4th value discarded. 900
  1136 → $h(k)$

5. Rotation → $k = rev(k)$

Assume 97834 = 43879

$h(43879)$

Scanned with CamScanner

Best Approach

6. Pseudo Random

$Y = ak + b$

coefficient constant.

Pseudo Random no.

↓

Based on key

seed

Pseudo Random no.

Assume $k = 9, 8, 7, 3$

$b = 13 \in P$ (It is better that $B \in P$)

$a = 14738$

$Y = 14738(9) + 13$

$Y = Y \% m$

$h(k) = Y \% m$.

For every key value, a new random no. is generated.

→ Minimum no. of collission

collission is reduced not eliminated.

Techniques to Resolve Collission:

Major Methods

↳ Collission Resolution

1. Close Address | Chaining
2. Open addressing
   - ↳ Linear probbing
   - ↳ Quadratic probbing
   - ↳ Double hashing

Scanned with CamScanner

Collision Resolution Technique

Hashing ~ Example

14th Dec, 2020

351, 455, 777, 888, 36, 28, 113, 117, 119, 56

0	1	2	3	4	5	6	7	8	9
117	351	119	113	56	455	36	777	888	28

int A[10] = {-1}

351 % 10 = 1

455 % 10 = 5

777 % 10 = 7

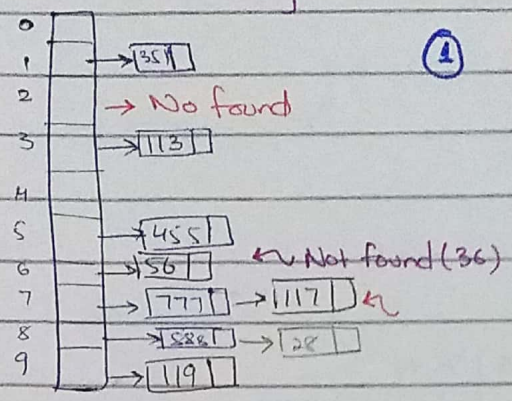
888 % 10 = 8

113 % 10 = 3

117 % 10 = 7

119 % 10 = 9

A Hash Array.



Array of Pointers.

① Direct hashing

↓

Separate Chaining

OR

Open Chaining

OR

Direct Chaining.

A[key % 10] = Pointer

* 36 % 10 = 6 Not found
* 117 % 10 = 7 Found.
* 132 % 10 = 2 Not found

Best Case = O(1)

Worst Case = O(n)

Modular Division

h(k) = key % m

m → size of hash array

② Open Addressing.

↳ Linear Probbing h(k) = k % m → used for insertion

In case of collission = (k + i) % m ∵ i = 1 → m

* At every collission, iteration start with 1.

0	36
1	351
2	28
3	
4	
5	455
6	665
7	777
8	888
9	487

k	h(k) = k % m	For Collission (k+i) % m Linear Prob
351	1	
455	5	
777	7	
888	8	
487	7	[(7+1) % 10 = 8] [(7+2) % 10 = 9]
665	5	[(5+1) % 10 = 6]
36	6	[(6+1) % 10 = 7]

(36+4) % 10 = 0 ← 36

28, 113, 119

↳ Quadratic Probbing

$H(k) = k \% m$

$h(k) = (k + i^2) \% m.$

1, 4, 9, 16, 25, 36

Most commonly used

↳ Double hashing.

$H_1(k) = k \% m \rightarrow 0$

In case of collission,

$H_2(k) = (k / m) \% m$

can be any function.

$= (3k + 7) \% m$

$= (k^2 / m^2) \% m$

Double hashing $H_1(k) = k \% m$, $H_2(k) = (k/m) \% m$

0	
1	351
2	28
3	36
4	487
5	455
6	665
7	777
8	886
9	

key	$H_1(k)$	if collission occurs, $H_2(k) = (k/m) \% m$
351	$351 \% 10 = 1$	
455	$455 \% 10 = 5$	
777	$777 \% 7 = 7$	
888	$= 8$	
487	$= 7$	$(487/10) \% 10 = 8$
		$(48/10) \% 10 = 4$
665	$= 5$	$(665/10) \% 10 = 6$
36	$= 6$	$(36/10) \% 10 = 3$
28	$= 8$	$(28/10) \% 10 = 2$

Scanned with CamScanner

16th Dec, 2020.

Graph.

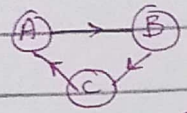
1. Abstraction Pictorial Presentation
   Model of some real world problem
2. Nodes — Edges

Representation:

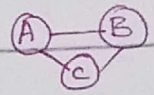
a. Adjacency Matrix — square — A^n
b. Adjacency List

Types:

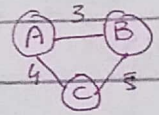
A — Directed Graph



— Undirected Graph



B — weighted Graph



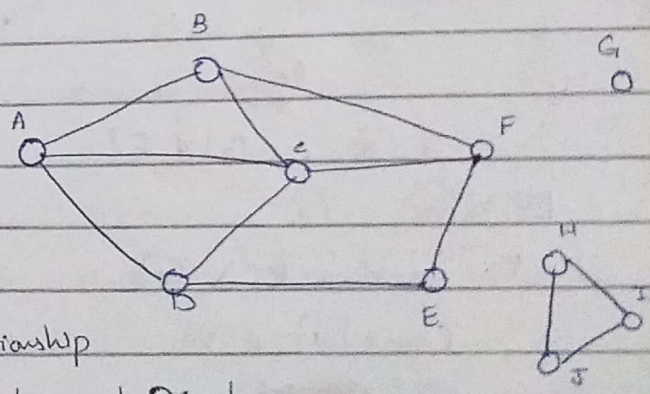
— Undirected Graph

UAG = Undirected Acyclic Graph

DAG = Directed Acyclic Graph

Graph Traversal.

1. Depth First Search (DFS)
2. Breadth First Search (BFS)
   Least Order Search



Tree	Graph
Root	No Root
Parent Child Relationship	No Parent Child Relationship
Only Forward direction	Both Forward & Backward Directions
No weighted Tree	Weighted Graph
All Nodes are connected	All Nodes are not necessarily connected.

1. Tree Edge: DFS → visited Edge.
2. Forward Edge: $E(u,v)$ — DFS

$$\left.\begin{array}{l} T(u) < T(v) \\ \& \quad u \rightarrow u \ \exists \text{ path} \end{array}\right\}$$

Scanned with CamScanner

3. Backward Edge: $E(u,v)$

$T(v) < T(u)$

$\forall \; \exists v \rightarrow u$

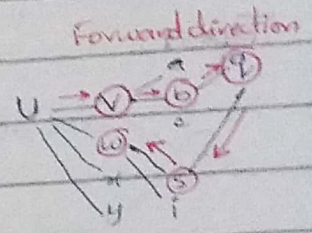
4. Cross Edge: $E(u,v)$

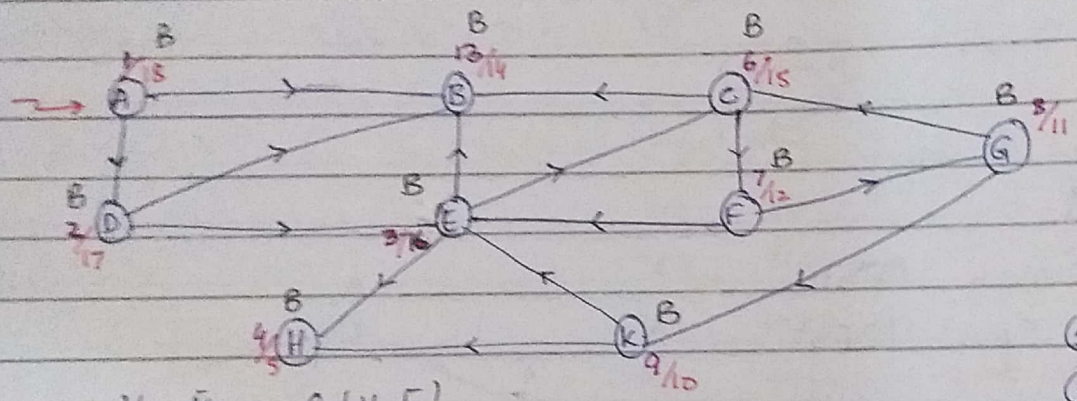
iff

DFS (Depth First Search)

Last In First Out

Data Structure = Stack





W – white

G – Gray

B – Black

1. Color
2. Previously visited Node
3. Time (Start)
4. Finish Time

$V, E, G(V, E)$

```
DFS(G)
  For each u ∈ V[G]
      Color[u] = W
      P[u] = Null
  Time = 0
  For each u ∈ V[G]
      if (color[u] = W)
          DFS_Visit(u)
```

```
DFS_Visit(u)
  Color[u] = G
  Time = Time + 1
  d[u] = Time
  For each v ∈ Adj[u]
      if (color[v] == W)
          P[v] = u
          DFS_Visit(v)
  Color[u] = Black
  F[u] = Time = Time + 1
```

Scanned with CamScanner

Vertices	A	B	C	D	E	F	G	H	K
Color	G	B	B	B	B	B	B	B	B
π/P				A	F	C	F	E	-
Start Time d[v]	13	1	3	14	7	4	5	8	17
Finish Time f[v]	16	2	12	15	10	11	6	9	18

Time = 0.

B → starting Node.

Time = 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18

u = B C F G E H A D

v = B

v = B F

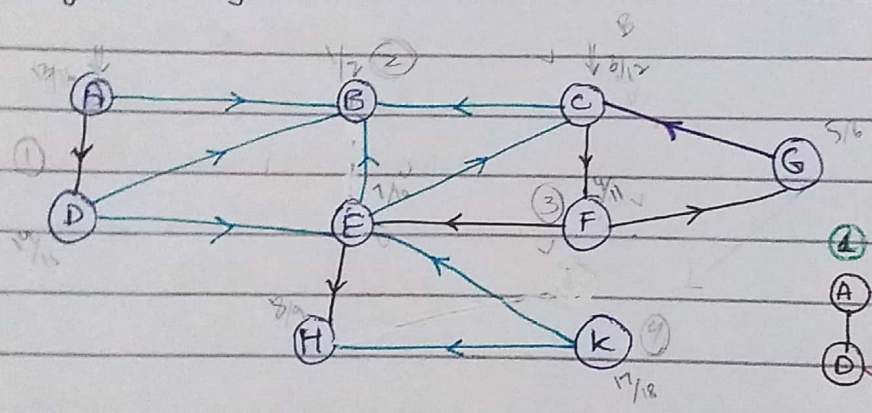
v = G E H

D

Edges:

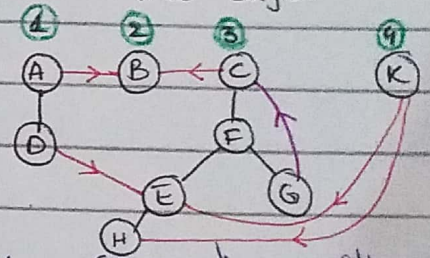
18th Dec, 2020

1. Tree Edges — Edges in DFS



- Backward Edge
- Tree edges
- Forward edge
- Cross Edge.

Tree Edges.



2. Forward Edges — For any two nodes u, v, connecting with an edge $e_1(u,v)$ $u < v$

$A(u) \rightarrow (v)$

Forest → 1 graph
↓
multiple trees.

* path u link v directly or indirectly.
  ⇒ other than tree edges

3. Backward Edges — $e(u,v)$

AB, BC, CA

$V \rightarrow u$ $v < u$

* cyclic path — if backward edge exist.

4. Cross Edges — other than tree edgeds. In b/w two different trees
   ⇒ which link two trees.

Scanned with CamScanner

BFS (Breadth First Search)

```
BFS  G(V, E)
For each u ∈ V[G] - s
        Color[u] = white
        d[u] = ∞
        P[u] = Nill
Color[s] = Grey
d[s] = 0
P[s] = Nill
Que = φ
EnQue (Q, s)
while (Que ≠ φ){
    u = DeQue (Q)
For each v ∈ Adj[u]
    if (color[v] == white)
        color[v] = Grey
        d[v] = d[u] + 1
        EnQue (Q, v)
color [u] = Blue

}
```

Initialization

Scanned with CamScanner

Minimum Spanning Tree (MST).

21st Dec, 2020.

$G(V, E)$

⇒ Graph may have cycles whereas Trees never have a cycle.

$V = \{v_1, v_2, v_3, \ldots v_n\}$

$V' = \{v_1, v_2, v_3, \ldots v_n\}$ $\qquad V = V'$

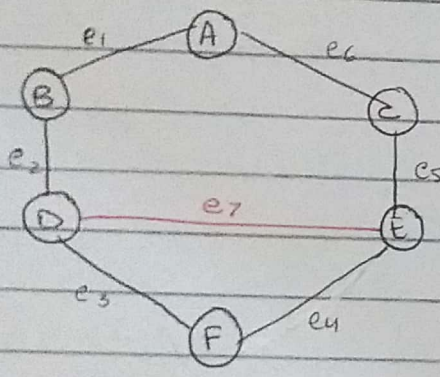
$E = \{e_1, e_2, e_3, \ldots e_m\}$ $\qquad E \supset E'$

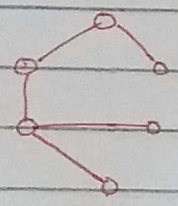
$E' = \{e_1', e_2', e_3' \ldots e_p\}$ $\quad p < m$

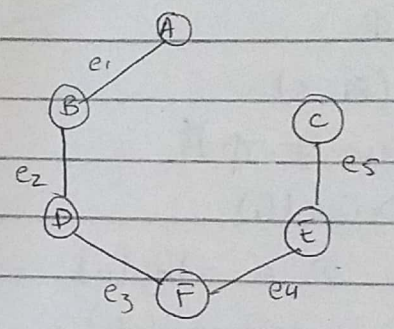
In spanning tree, edges are less than in graph.

★ ↑ G = multiple ST

G







Spanning Tree.

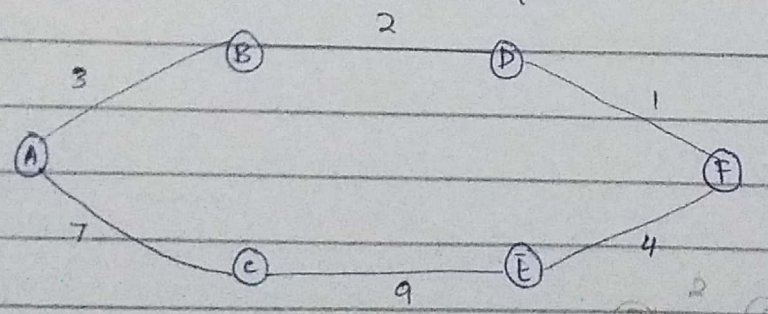
⇒ No. of cycles

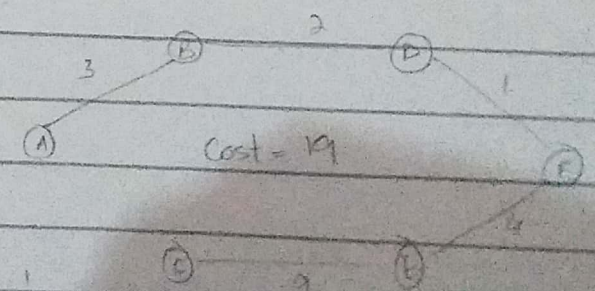
$E = E' \cup$ Edges making cycles.

$E = E -$ no. of cycle.

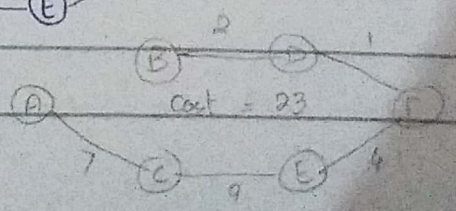
MST: Minimum cost spanning Tree.

- Undirected weighted Graph. ~basic restriction





Cost = 19

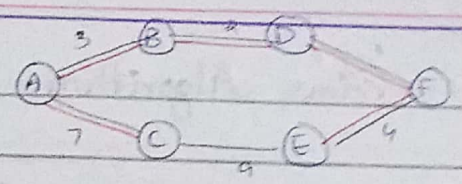


Cost = 23

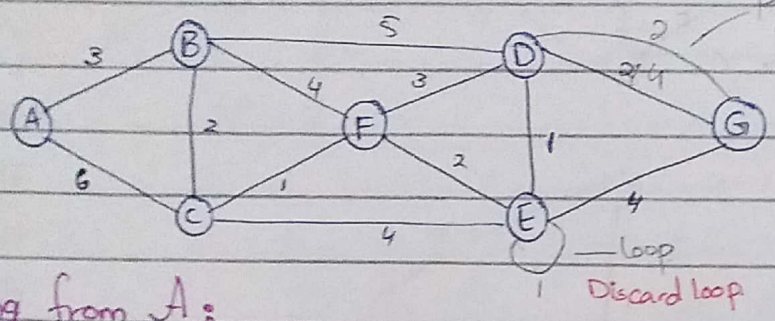
Scanned with CamScanner

Paths for MST.

Kruskal — local Approach
Prims — global Approach

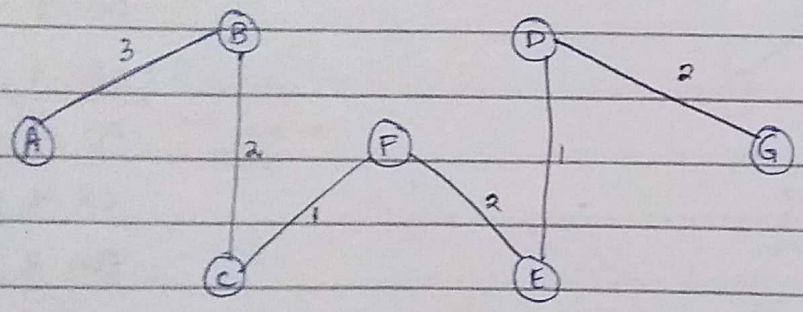


Kruskal's Algorithm.



For MST, eliminate:
1- loop
2- Parallel edges

Starting from A:



$V_1 = \{A,B\} \cup \{B,C\} = \{A,B,C\} \cup \{C,F\}$
$= \{A,B,C,F\} \cup \{F,E\}$
$= \{A,B,C,E,F\} \cup \{E,D\}$
$= \{A,B,C,D,E,F\} \cup \{DG\}$
$V_1 = \{A,B,C,D,E,F,G\}$
$V = V_1$

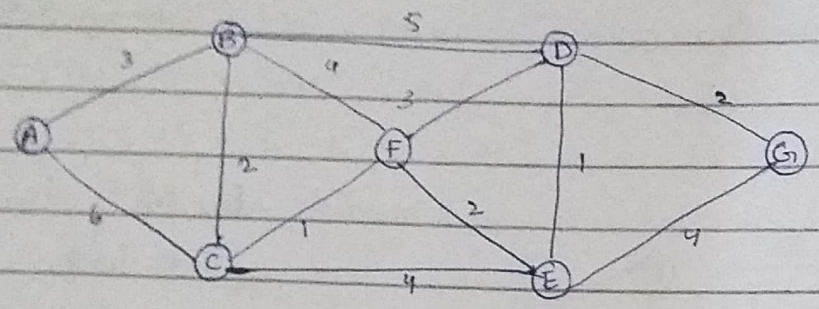
~~AB-3~~ ✓
AC-6
BD-5
~~BC-2~~ ✓
BF-4
CE-4
~~CF-4~~ ✓
~~FE-2~~ ✓
FD-3 → if DG-4 then FD is selected but it is not considered
~~ED-1~~ ✓
EG-4
~~DG-2~~ ✓

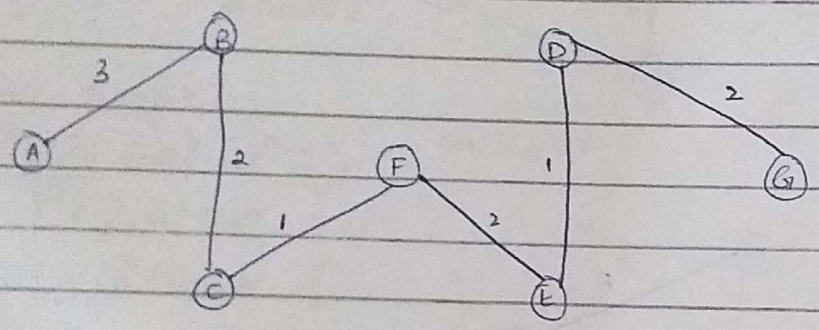
Disjoint Sets
$V_1 = \{A,B,D,E\}$
$V_2 = \{C,E,F\}$
✗ $e_1 = \{D,E\}$ → belong to same U set, dont consider this edge.
✗ $e_2 = \{C,E\}$
✓ $e_3 = \{B,F\}$
$V_3 = V_1 \cup V_2$
$V_3 = \{A,B,C,D,E,F\}$

Data Structure
↓
Min Heap.

Time Complexity.
$T(n) = O(V \log E)$

Prims Algorithm.





Min Heap :

~~CF - 1~~ ✓

~~DE - 1~~ ✓

~~BC - 2~~ ✓

~~FE - 2~~ ✓

~~DG - 2~~ ✓

~~AB - 3~~ ✓

all the vertices are visited.

FD - 3 ✗

CE - 4 ✗

EG - 4 ✗

BF - 4 ✗

BD - 5 ✗

AC - 6 ✗

$V_1 = \{C, F\} \cup \{B, C\} = \{B, C, F\}$

Disjoint set $V_2 = \{D, E\}$

$V_3 = V_1 \cup V_2 = \{B, C, F\} \cup \{D, E\} = \{B, C, D, E, F\} \cup \{D, G\}$

$\because$ FE

$F \in V_1$ & $E \in V_2$

$V_3 = \{B, C, D, E, F, G\} \cup \{A, B\}$

$= \{A, B, C, D, E, F, G\}$

$V = V_3$

$E' \subset E$

★ Edges dropped = no. of cycles.

Time Complexity

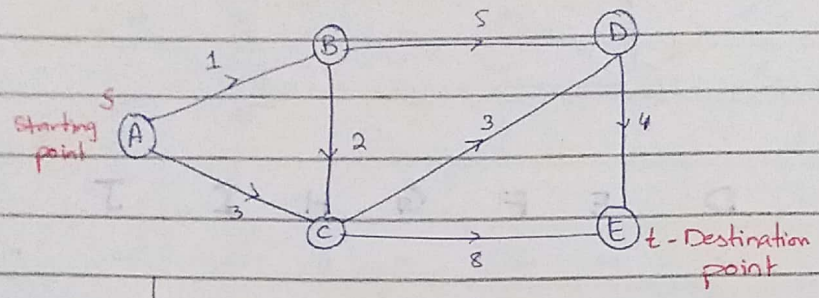
$T(n) = O(V \log E)$

Graph.

31st Dec, 2020

Single Source Shortest Path.

- Dijkastra's Algo
- Bellman Ford.



A → C → E | A → B → C → E | A → B → D → E | A → C → D → E

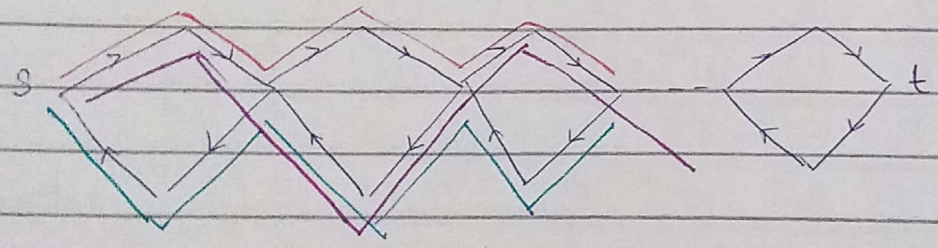
11 | 11 | 10 ✓ shortest path | 10

X — cost & no. of — Y

nodes are equal

A → B → C → D → E

10



n vertices

$\frac{n-1}{3}$

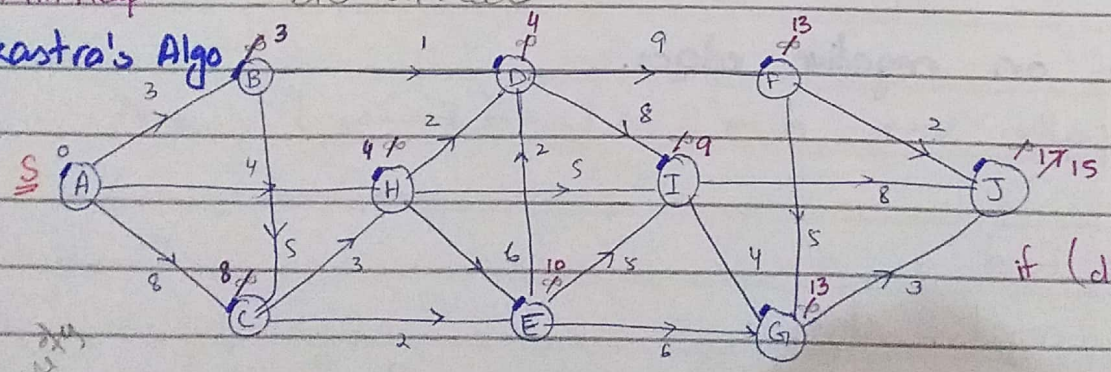
k-cubes

3k+1 vertices.

$2^{n/3}$ possible paths

Min-heap BFS with code

Dijkastra's Algo



if (d[v] < d[u] + w(u,v))

d[v] = d[u] + w(u,v).

A - 0
B - 3
C - 8
H - 4
D - 4
I - 9
E - 10
F - 13
G - 13
J - 17/15

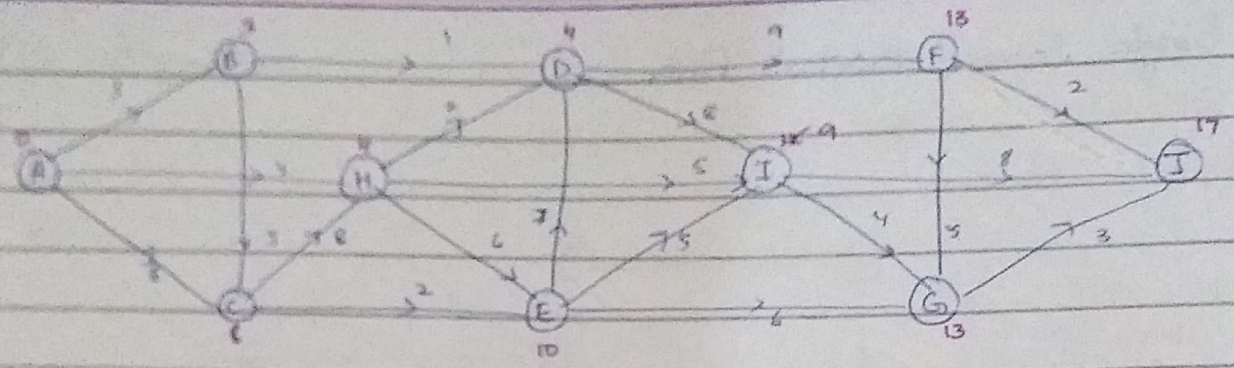
Time complexity

$T(n) = O(n \log n)$

$V \log V$ — cost of heap.

$T(n) = O(V \log V)$ — Total complexity.

depends on vertices.



From A

			B	C	D	E	F	G	H	I	J
SSSP	1	A	A^3	A^8	∞	∞	∞	∞	A^4	∞	∞
Dijkstra	2	B	A^3	A^8	B^4	∞	∞	∞	A^4	∞	∞
	3	D	A^3	A^8	B^4	∞	D^{19}	∞	A^4	D^{12}	∞
	4	H	A^3	A^8	B^4	H^{10}	D^{13}	∞	A^4	H^9	∞
	5	C	A^3	A^8	B^4	H^{10}	D^{13}	∞	A^4	H^9	∞
	6	I	A^3	A^8	B^4	H^{10}	D^{13}	I^{13}	A^4	H^9	I^{17}
	7	E	A^3	A^8	B^4	H^{10}	D^{13}	I^{13}	A^4	H^9	I^{17}
	8	F	A^3	A^8	B^4	H^{10}	D^{13}	I^{13}	A^4	H^9	F^{15}
	9	G	A^3	A^8	B^4	H^{10}	D^{13}	I^{13}	A^4	H^9	F^{15}
	10	J	A^3	A^8	B^4	H^{10}	D^{13}	I^{13}	A^4	H^9	F^{15}

Failure :

Does not work on negative edges.

Shortest path $\longleftrightarrow$ $A \rightarrow B \rightarrow D \rightarrow F \rightarrow J$ = 15

Solution: Bellman Ford

work on negative edges

Restriction = doesnot work on negative cycles

Scanned with CamScanner

Bellman Ford - SSSP Algo.

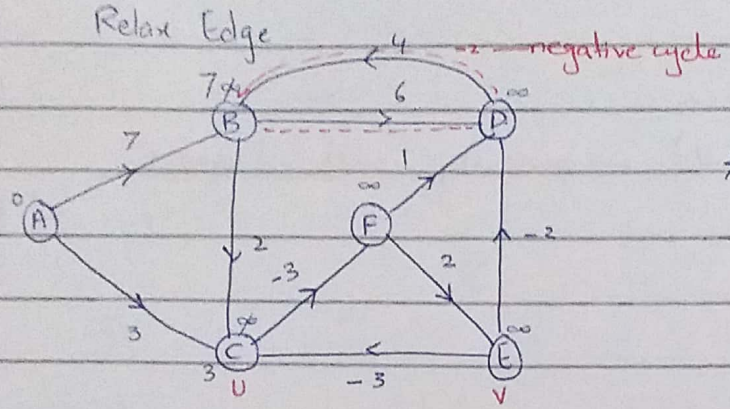
11th Jan, 2021.

n vertices V

n-1 { V Edges E

Relax Edge

Based upon edges



* n = 6 vertices
  (n-1)

$D(v) < D(u) + c(u,v)$

$\min(D[v], D[u] + c(u,v))$

* $D[S] = 0$
* $\forall\ D[u] = \infty$

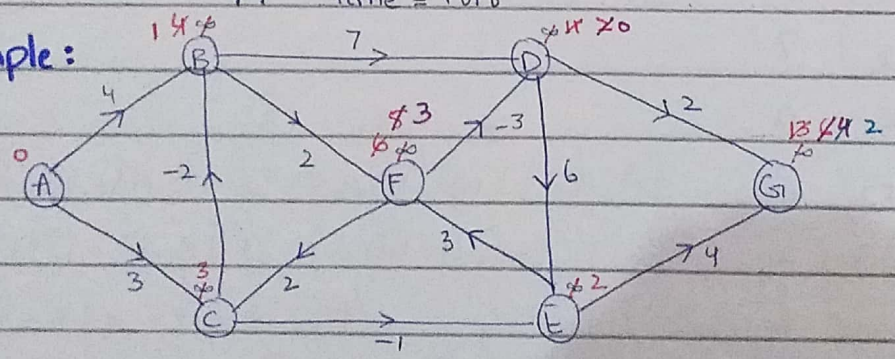
1 ⇒ (A,B), (A,C), (B,C), (B,D), (C,F), ...
(D,B), (E,C), (E,D), (F,D), (F,E)

check cost and update.

Failure: Does not work on negative cycles

n^{th} time = run

Example:



Iteration 1 → ✓
2 → ✓
3 → ✓
4 → ✓ No change
⇒ We may stop here

- Vertices = 7
- Iterations = n-1 = 6
- Edges: (A,B) (A,C) (B,D) (B,F) (C,B) (C,E) (D,G) (D,E) (E,G) (E,F) (F,D) (F,C)

- S = A
- D[S] = 0

$D[u] = \min(D(v), D[u] + w(u,v))$

for 1 to V-1

for each $e(u,v) \in E$

$D[v] = Min(D[u] + w(u,v), D[v])$

A – 0
B – 1
C – 3
D – 0
E – 2
F – 3
G – 2

Time Complexity.

$O(E.V-1) \Longrightarrow O(E.V)$ — partially connected graph.

In case of Mesh,

No. of Edges $= \frac{V(V-1)}{2}$

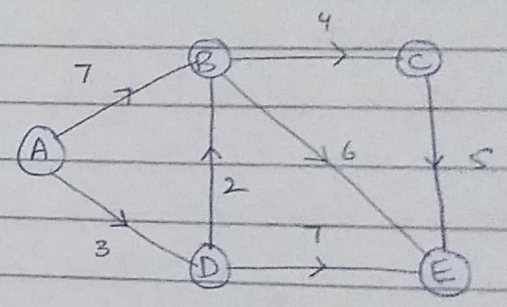
$T(n) = O(V.V^2)$

$= O(V^3)$ — completely connected graph.

Floyd warshall Single Source Shortest Path.

13th Jan 2021

	1 A	2 B	3 C	4 D	5 E
$D_0 =$ A	0	7	∞	3	∞
B	∞	0	4	∞	6
C	∞	∞	0	∞	5
D	∞	2	∞	0	7
E	∞	∞	∞	∞	0



	A	B	C	D	E
$D_1 =$ A	0	7	∞	3	∞
B	∞	0	4	∞	6
C	∞	∞	0	∞	5
D	∞	2	∞	∞	7
E	∞	∞	∞	∞	0.

★ B, C (A)

$D^{1-1}[B,C], D^{1-1}[B,A] + D^{1-1}[A,C]$

$min(D^0(B,C), D^0[B,A] + D^0(A,C))$

$4, \infty + \infty = (4, \infty) = 4$

★ B, D (A)

$min(D^0(B,D), D^0(B,A) + D^0(A,D))$

$\infty, \infty + 3 = (\infty, \infty)$

★ B, E (A)

$min(D^0(B,E), D^0(B,A) + D^0(A,E))$

$6, \infty + \infty = (6, \infty) = 6$

Scanned with CamScanner

	A	B	C	D	E
A	0	7	11	3	13
D_2 = B	∞	0	4	0	6
D_3 = C	∞	∞	0	∞	5
D_4 D	∞	2	6	0	7
D_5 E	∞	∞	∞	∞	0

* A, C (B)

D^{2-1} [A C]

min(D'(A,C), D'(A,B) + D'(B,C))

∞, 7 + 4 = (∞, 11) = 11

* A, D (B)

D'(A,D), D'(A,B) + D'(B,D)

3, 7 + ∞ = 3

* A, E (B)

D'(A,E), D'(A,B) + D'(B,E)

(∞, 13)

* D, A (B)

D'(D,A), D'(D,B) + D'(B,A)

∞, 2 + ∞ = ∞

* D, C

D'(D,C), D'(D,B) + D'(B,C)

∞ = 2 + 4 = (∞, 6) = 6

* D, E

D'(DE), D'(D,B) + D'(B,E)

7, 2 + 6 = (7, 8) = 7

* C, B (A)

min(D°(C,B), D°(C,A) + D°(A,B))

∞, ∞ + ∞ = ∞

* D, B (A)

min(D°(D,B), D°(D,A) + D°(A,B))

2, ∞ + ∞ = 2

* D, C (A)

min(D°(D,C), D°(D,A) + D°(A,C))

∞, ∞ + ∞ = ∞

* DE (A)

min(D°(DE), D°(D,A) + D°(A,E))

7, ∞ + ∞ = 7.

```
for (i=0; i<v; i++)
 for (j=0; j<v; j++)
  D^k[i,j] = (D^{k-1}[i,j], D^{k-1}[i,k] +
    D^{k-1}[k,j])
```

Time Complexity = $O(v^2)$

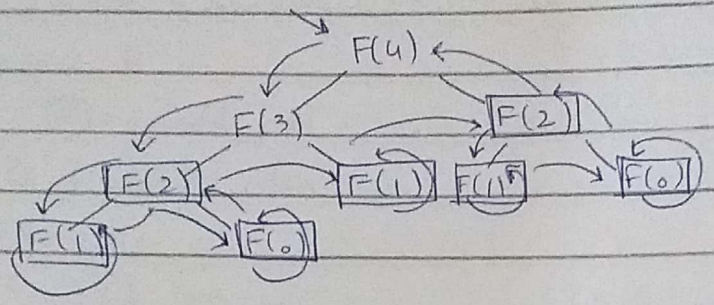
Scanned with CamScanner

18th Jan, 2021

Dynamic Programming.

DP

Fibonacci $F(n) = \begin{cases} 1 & n, 1, 0 \\ F(n-1) + F(n-1) & \forall\, n > 1 \end{cases}$



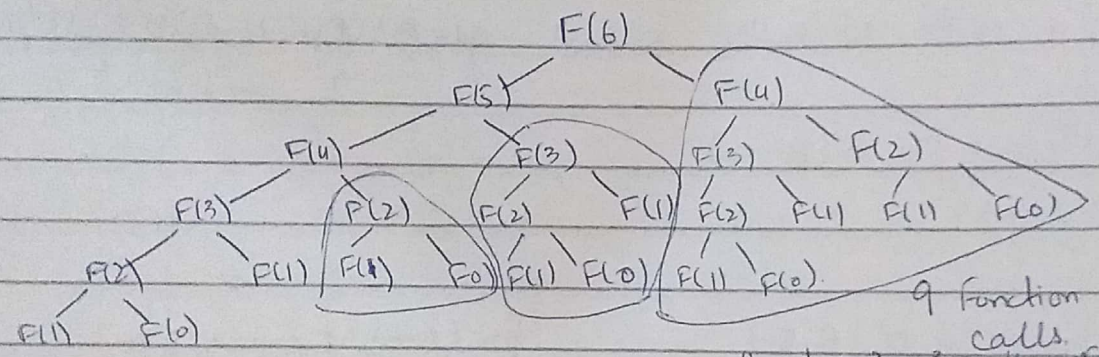
Reuse

Memory

Table

↓

store values.



9 function calls.

0	1	2	3	4	5	6	7
1	1	2	3	5	8	13	21

* DP = Recursion + Table

Recursion → Subproblem → Top down Approach.

Table → Memorize.

```
Fib(int n){
    F(0) = 1
    F(1) = 1
    for (i=2 ; i<=n ; i++)
        Fib[i] = Fib[i-1] + Fib[i-2]
}
```

→ Iterative Solution

$T(n) = O(n)$

$S(n) = O(n)$

```
Fib(n) {
    if (n>1)
        return Fib(n-1) + Fib(n-2)
    else
        return 1
}
```

→ Recursive Solution

$T(n) = O(n \log n)$

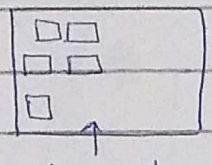
$S(n) = O(\log n)$

$\boxed{\text{DP} < \text{Recursive} \quad T(n) < T(n)}$

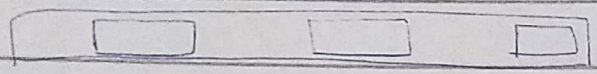
Scanned with CamScanner

DP Problems.

* Text Justification — DP problem
  M^t word → very common problem.
* KNapsack problem
  0/1
  no. of items ← max.
  cost of items.



  save items.
* Fractional
* Edit Distance. Problems



  S => ALGORITHM
  D => INSTITUTE
  min
  Replacement

  Longest common subsequence.
* Matrices chain Multiplication.
  1 ⟶ 10 → n
  $M_1, M_2, M_3, \ldots M(n)$
* Traveler Sales man Problem.
  → Shortest distance to distribute max. papers.
* Ford Fulkerson.

① Greedy
Max
Min

DP steps to solve Sub Problem:

1. Define Sub Problem / No. of Sub-Problem.
2. Guess Part Solution
3. Retake Sub problem Solution.
4. Build Algorithm.
   Recursive + Memorization.
   Building the DP Table - Bottom Up
5. Solve the Original Problem.

Scanned with CamScanner

DP Characteristics.

a. Brute Force Technique.

b. Guessing the Recursion + Memoization.

c. Shortest DAG. (Directed Acyclic Graph)

* Complexity = No. of subProb * Time Required for one subproblem.

Recursive —— $O(1)$

① Knapsack 0/1

Scanned with CamScanner