Graphs

A Graph is a collection of objects or entities called Nodes or Vertices connected through Edges.

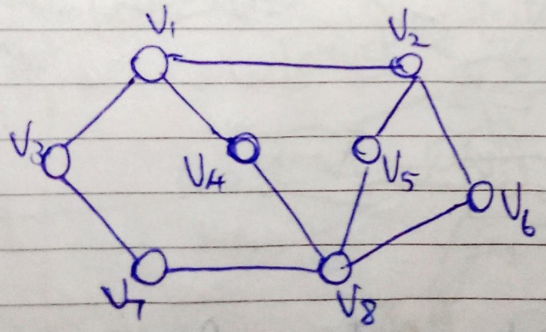
Mathematically, a Graph G is an ordered pair of a set V of vertices and a set E of edges.

$G = (V, E)$

V = set of Vertices
E = set of Edges

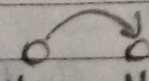
ordered pair : $(a,b) \neq (b,a)$ if $a \neq b$

unordered pair: $\{a,b\} = \{b,a\}$ (origin, destination)

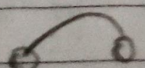


{ }

So set $V = \{V_1, V_2, V_3, V_4, V_5, V_6, V_7, V_8\}$



Directed

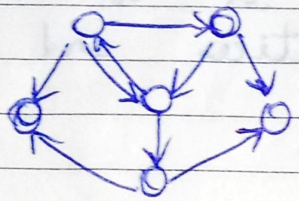


Undirected

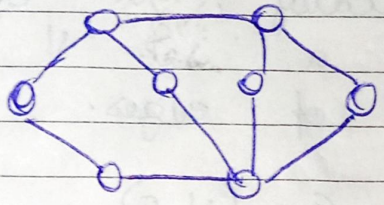
2021

$E = \{ \{V_1, V_2\}, \{V_1, V_3\}, \{V_3, V_7\}, \{V_7, V_8\}, \{V_1, V_4\}, \{V_4, V_8\}, \{V_8, V_5\}, \{V_8, V_6\}, \{V_2, V_5\}, \{V_2, V_6\} \}$

Directed vs Undirected Graph

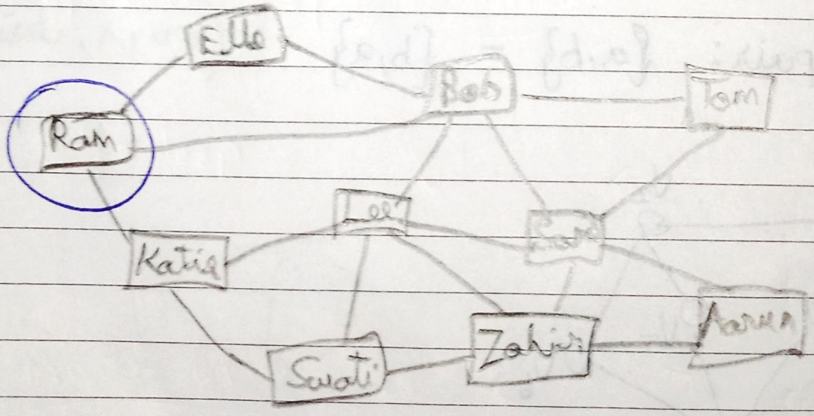


Directed



Undirected

(ex) Social Network (facebook) – Undirected



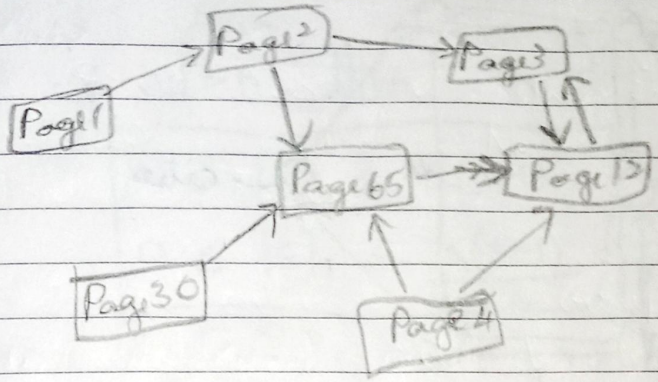
Swathi, Lee, Sam, Tom

Suggesting friends to Ram can be done by suggesting friends or Ram's friends.

(or)

Shortest Path from Node (Ram) equals 2

Ex Interlinked Web Pages - Directed Graph.

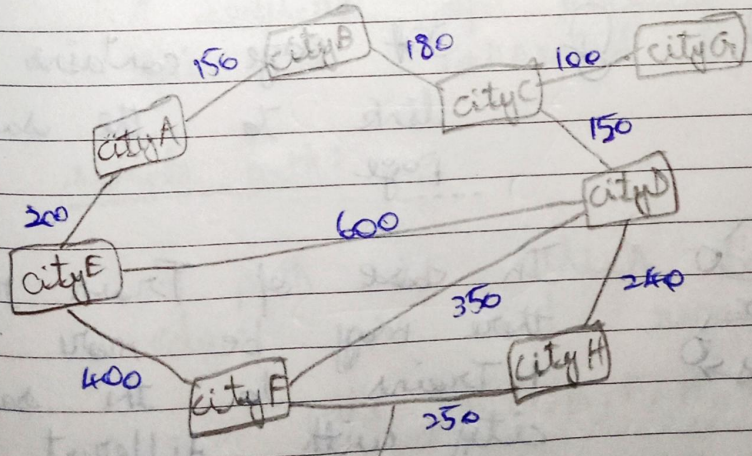


Page contains link to another page

Web Crawling or Act of Visiting all nodes in Graph

Weighted vs Unweighted Graph

Ex Intercity Road Network - Weighted



Undirected because of both ways travel.

shortest route from cityA to cityD is,

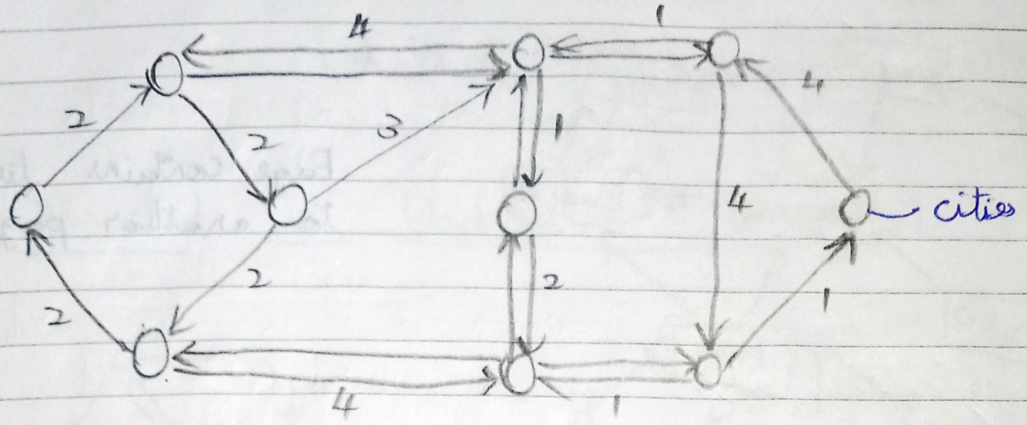
A-B-C-D

=> 150 + 120 + 150

=> 480

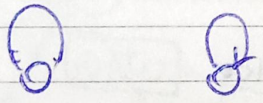
2021

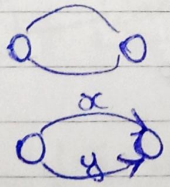
(ex) Intracity Road Network - Weighted Digraph.



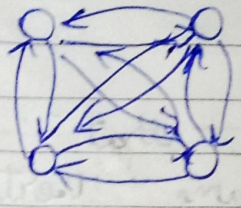
In intracity road networks between each node two way lanes as well as One way is present. So a digraph is optimal.

Properties of Graphs

self-loop :  If page contains link to the same page.

multi-edge :  In case of Train routes there may be more than 1 Trains to the same city with different costs, timings, etc.

number of edges:



$V = \{v_1, v_2, v_3, v_4\}$

$|V| = 4$

if $|V| = n$, then

$$0 \leq |E| \leq n(n-1) \quad \text{, if directed}$$

$$0 \leq |E| \leq \frac{n(n-1)}{2} \quad \text{, if undirected.}$$

dense graph & sparse graph:

too many edges	too few edges

Adjacency Matrix — Adjacency List to store the graph.

path:

A sequence of vertices where each adjacent pair is connected by an edge

simple path:

A path in which both vertices and edges are not repeated.

path = <A, B, F, H, E, B, A, D>

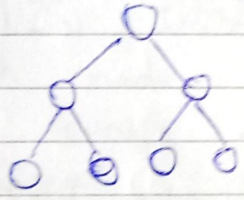
simple path = <A, B, F, H>

closed walk

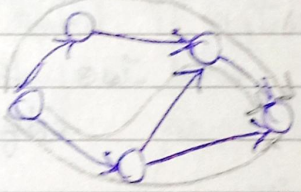
A walk that starts and ends at the same vertex.

Acyclic Graph

graph with no cycles (starting and ending at the same vertex)



Undirected Acyclic graph



Directed Acyclic Graph

Graph Representation

Edge List

~~Node~~

class Edge:

startVertex, endVertex, weight

Vertex List

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

Edge List

Adjacency Matrix

$O(V^2)$

Vertex List

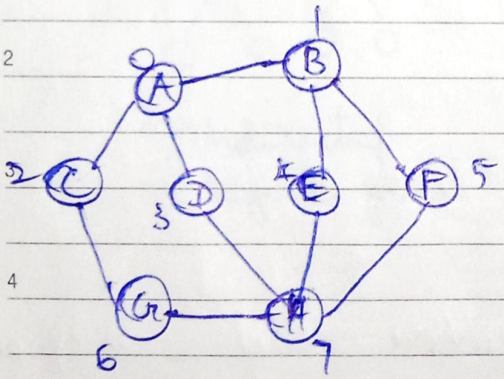
0	A
1	B
2	C
3	D
4	E
5	F
6	G
7	H

Adjacency Matrix

start \ end	0	1	2	3	4	5	6	7
0	0	1	1	1	0	0	0	0
1	1	0	0	0	1	1	0	0
2	1	0	0	0	0	0	1	0
3	1	0	0	0	0	0	0	1
4	0	1	0	0	0	0	0	1
5	0	1	0	0	0	0	0	1
6	0	0	1	0	0	0	0	1
7	0	0	0	1	1	1	1	0

→ values are stored even if nodes are not connected.

weights



Operation	Time-Cost	Space
finding adjacent nodes.	$O(V)$	$O(n)$
finding if two nodes are connected	$O(1)$ + ~~$O(V)$~~	

But the space complexity is very high because it is not efficient in storage, even if there is no edge it has to store value '0'.

Adjacency List

Most real-world graphs are sparse, meaning most of the connections are "0." Our number of connections is small compared to total number of possible connections.

For a Social network with a billion (10^9) users,

raw size = 10^9
(1 user)

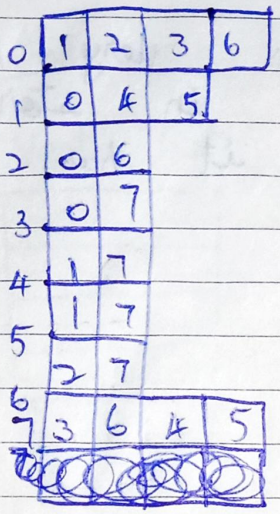
If the user has 1000 friends:

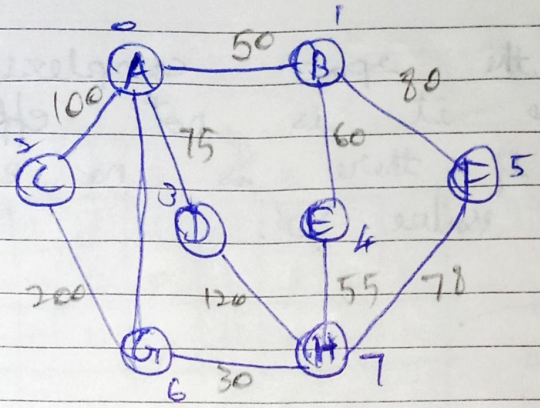
no. of 1s = 1000 ≈ 1KB 0,1 are boolean

no. of 0s = (10^9 - 1000) ≈ 1GB

Adjacency Matrix	Adjacency list
• find if 2 nodes are connected O(1)	O(V) – linear search O(logV) – binary search (BST)
• adjacent nodes O(1)	O(V)

2021

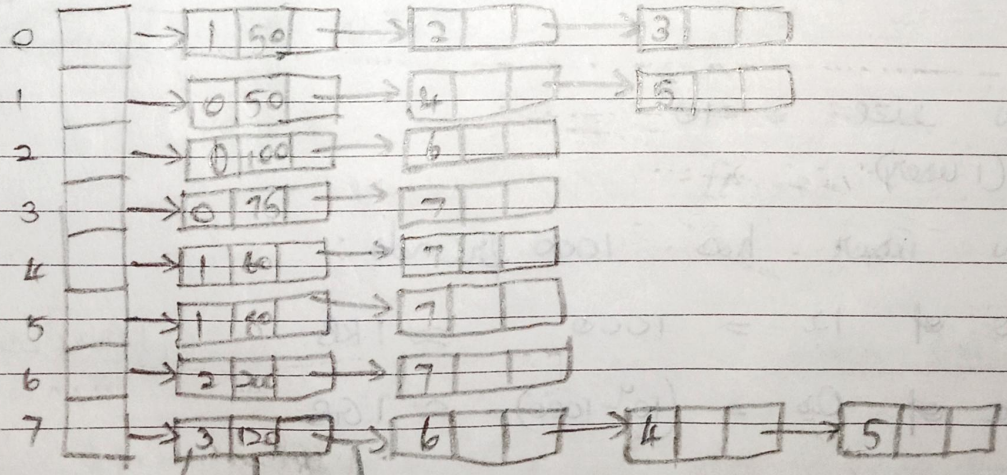


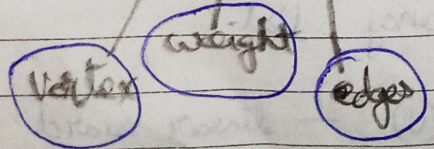


Instead we use an Array of linked list.

Adjacency List

Undirected Graph



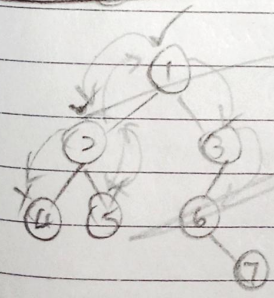


class
self.data = data
self.next = None
self.weight = ~~edges~~ weight
(or)
None

How good is it if you want to add or delete a new connection in complexity?
(consider using BST instead of LinkedList)

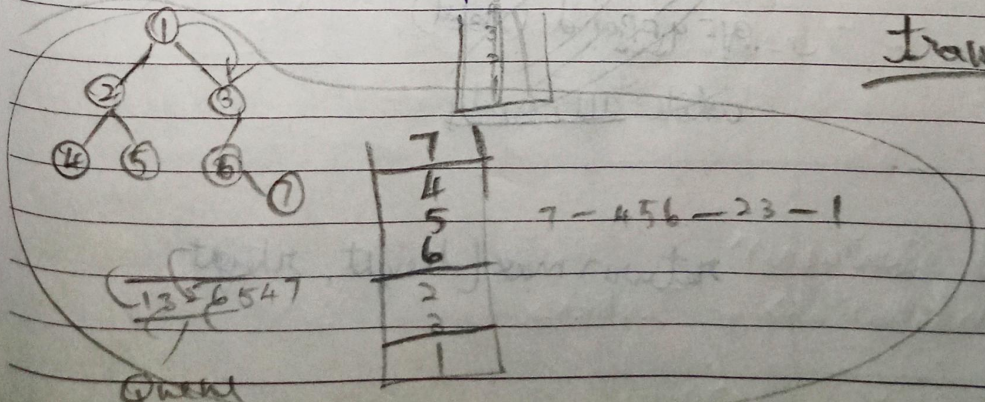
BFS
DFS
Shortest distance b/w nodes

finish Trees first - ~~BST~~, BinaryTree
(Depth first traversal)
then finish Graphs - BFS, DFS, shortest distance.
~~then~~ go to AVL trees, Tries, Red-Black Trees



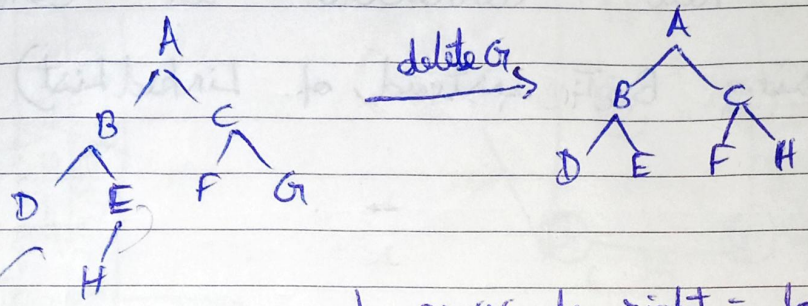
Reverse level Order traversal

SUNDAY 16



2021

deletion in a binary tree



prevnode = C
currnode = G
lastnode = H

prevnode.right = lastnode

prevnode = E
lastnode = H

prevnode.left = None

~~delete (root, key~~

delete (root, key)

delete (A, G)

{

delete (B, G)

delete (C, G)

~~if root.left == G:~~

if root.right == G:

root.right = None

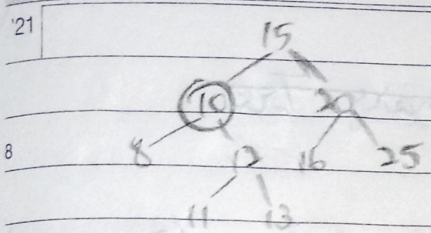
q = [ ]

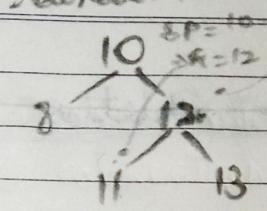
q.append (root)

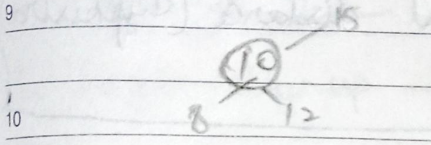
while q[ ] empty

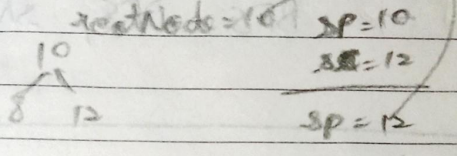
return max (lheight, rheight

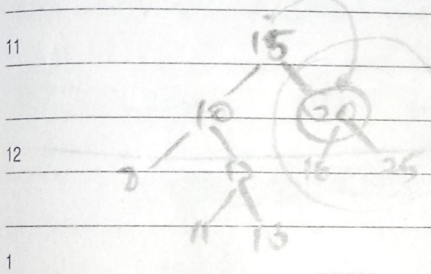
TUESDAY
MAY
18
WK 21 • 138-227

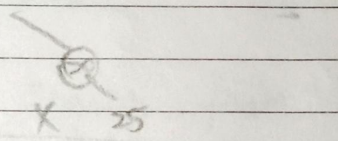


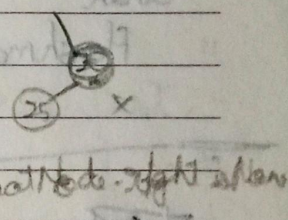


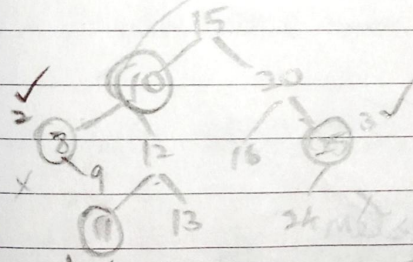


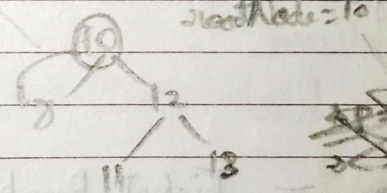


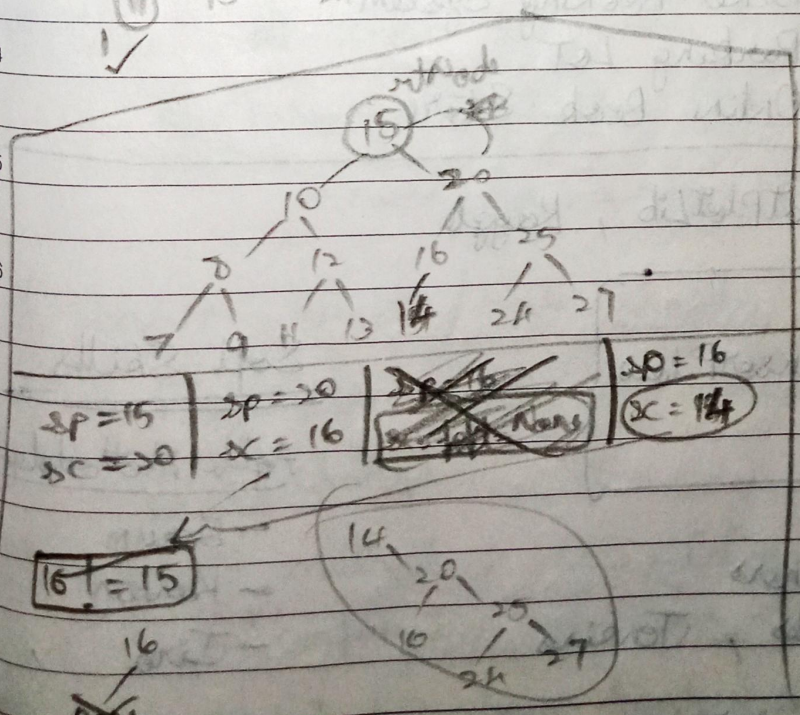


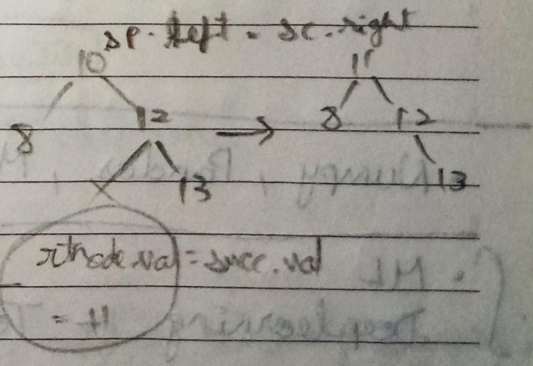






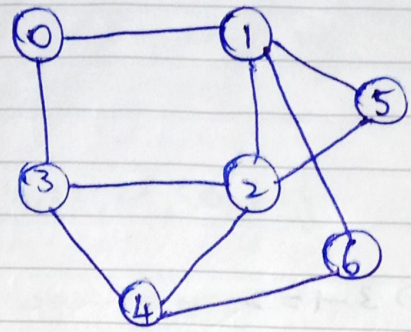






2021

Graph Traversals



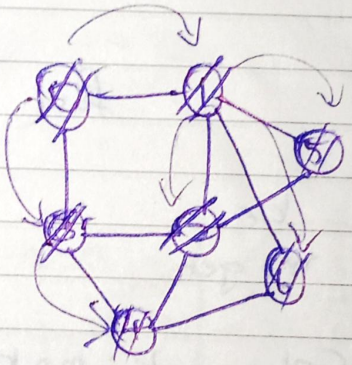
BFS

Q = ~~0~~ ∅ 0

Q = ∅ ~~1~~ 3 01

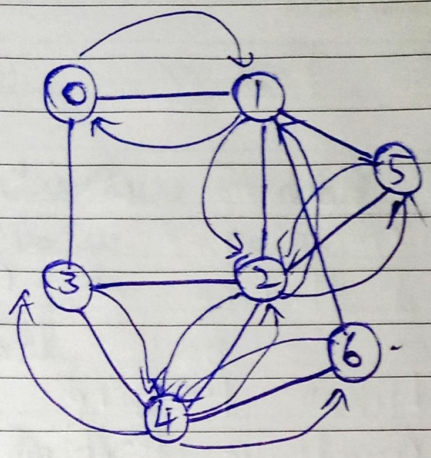
Q = ∅ ~~3~~ ~~2~~ 5 6 013

Q = ∅ ~~5~~ ~~6~~ ~~4~~ 4 0132564

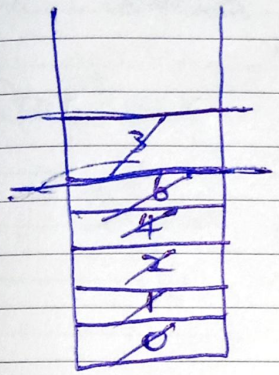


- A node is inserted initially
- Immediate connections of nodes which are unvisited are added to the Queue
- Upon deletion of a node the node is printed and its adjacents are added to Queue.

DFS



Stack



Print = 0 1 2 4 6 3 5

Hashtables

A Hashtable is a structure where data is stored in key value pairs.

Operations performed are,

Insert , Search , Delete
O(1) O(1) O(1)

Representation

The key-value pairs are stored in an Array structure where the indexing is done by using a Hash function on its key.

Hash (key) → index

a famous hash function is,

djb2

a hash function should be,

- fast to compute
- avoid collisions

Hashing

① Open Hashing or Separate Chaining

② Closed Hashing or Open Addressing

- Linear Probing
- Quadratic Probing
- Double Hashing

All the hashing techniques face collisions so there are methods to avoid this for each.

Separate Chaining

In the HashTable when collisions occur that spot is chained by the incoming element like a linked list.

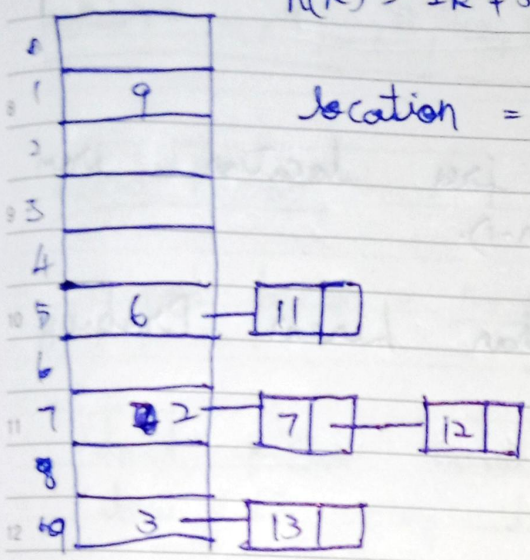
Ex A = 3, 2, 9, 6, 11, 13, 7, 12

$h(k) = 2k + 3$ $m = 10$

use Division Method & Closed Addressing to store these values.

$h(k) = 2k + 3$ $m = 10$

location $= (2k+3) \% 10$



key	location
3	9
2	7
9	1
⋮	⋮
12	7

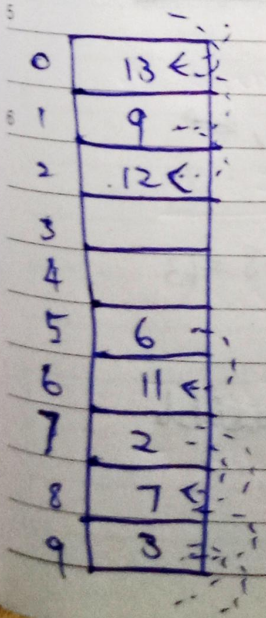
Linear Probing

Insert k_i at the first free location from $(h+i) \% m$ where $i = 0$ to $(m-1)$.

ex) $A = 3, 2, 9, 6, 11, 13, 7, 12$ | $h(k) = 2k+3$ | location $= (2k+3) \% 10$

① ② ③ ⑥

no of probes



order = 13, 9, 12, –, –, 6, 11, 2, 7, 3

SUNDAY 30

2021

Quadratic Probing

Insert k_i at the first free location from $(u+i^2)\%m$ where $i = 0$ to $(m-1)$.

It is generally better than Linear Probing.

Double Hashing

Insert k_i at the first free location from $(u+v*i)\%m$ where $i = 0$ to $(m-1)$.

(ex) A = 3, 2, 9, 6, 11, 13, 7, 12 | $h_1(k) = 2k+3$ | $h_2(k) = 3k+1$

location = $(2k+3)\%10$ — Division method

$v = (3k+1)\%10$ — Double hashing

0	
1	9
2	
3	11
4	12
5	6
6	
7	2
8	
9	3

$3 \to 9$, $2 \to 7$, $9 \to 1$, $6 \to 5$,

$11 \to$ ~~X~~

$d = (u+v*i)\%m = 5+4, 5+8 = 13$

$11 \to 3$

$13 \to$ ~~X~~

$d = 9+0, 9+0$ ~~X~~ not possible

$7 \to$ X $12 \to 4$

order = -, 9, -, 11, 12, 6, -, 2, -, 3

Load Factor

load factor = $\frac{\text{No of key-value pairs}}{\text{size of array}}$

load factor must be lower. If its too low it may lead in memory inefficiency.

19 WEDNESDAY
MAY
139-226 • WK 21

APR '21
1 2 3 4 5 6 7 8 9 10 11
12 13 14 15 16 17 18 19 20 21 22 23 24 25
26 27 28 29 30
M T W T F S S M T W T F S S

~~finish trees — deletion in a Binary Tree~~
first

ther finish — BFS, DFS, shortest distance (Dijkistra)
graphs

ther
Hashmaps

~~Tries~~

Dynamic Programming

SQL

System Design — Ticket Booking System
Parking Lot
Online Book Store

- Numpy, Pandas, MatPlotLib, Kaggle

- ML
  Deeplearning — TensorFlow
  MLOps
- AWS
- Docker and Containers
- CI & CD fundamentals, Jenkins

2021
- Fast API, Tensorflow serving.

Soft Skills

Agile Methodologies
- Scrum
- Kanban
- Jira