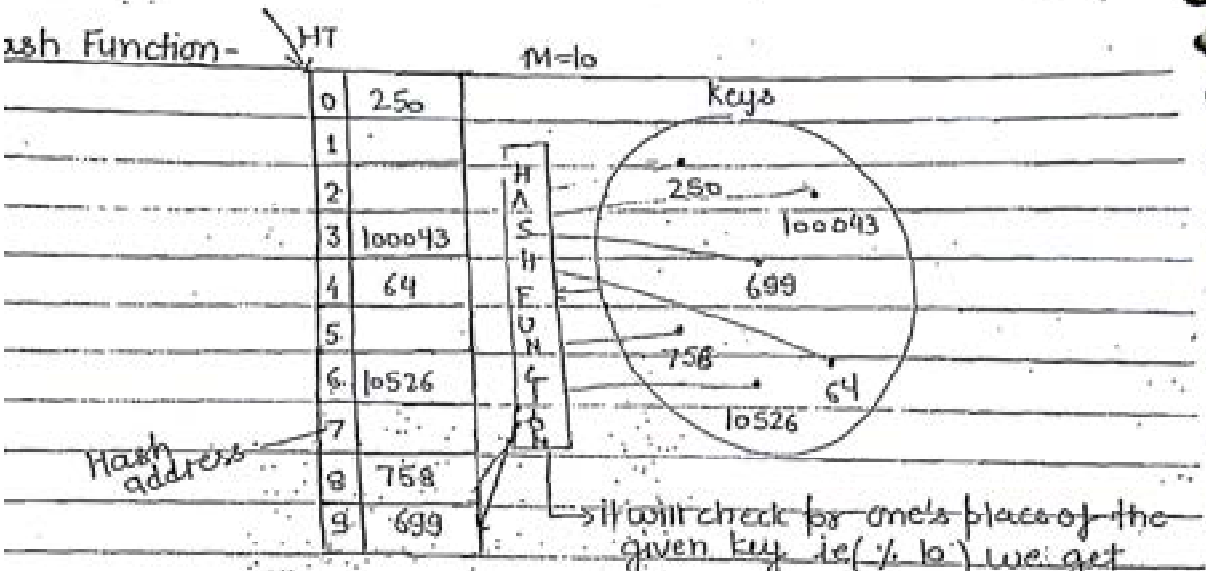
(If five keys were there)?



it will check for one's place of the given key ie (% 10) we get the (0-9) key just store it as 699 → 699 % 10 → 9

to retrieve 758 again % 10 = 8 (758) is stored at 8.

∴ Again time complexity = O(1)

Hash function take more time compared to Hash table due to function calculation.

Hash function is better as it also save the space.

∴ if n < 10 → slot = 0 to 9

n < 20 slot - 0 to 19

Collision - If two keys mapped to same hash address by hash function then it is called collision.

ie Ex- 699 759

both get collided at 9 posn

6 12345

% of it (is decided by us)

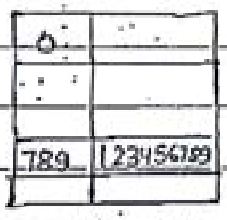
Hash function is better if no. of collision is as min. as pos.

∴ Types of Hash function-

1. Division Modulo Method-
2. Digit Extraction Method
3. Mid Square Method
4. Folding Method -
   (i) Fold Boundary Method.
   (ii) Fold Shifting Method.

1. Division Modulo Method-

ex1- Size of table m = 1000 (0 to 999)
key = 1 2 3 4 5 6 7 8 9
∴ Acc to Division Modulo Method
ie: Hash function (key) = key Modulo m
= 123456789 % 1000
= 789



ex2 - Size of table m = 8 (2^3) (take LSB 3 bit here)
key = 10100101011100101
∴ Hf(key) = 10100101011100101 mod 2^3
= 101

∴ if $m = 2^k$
then Hf(key)
↓
(k bits) of LSB

this Method donot bother about the whole data to how to store just take last k^{th} bits-

Ex 1010101111111111111 mod 2^4 = 1111

Ex 3 $m = 2^k$

key = 1010100101011111011110 mod 2^k = LSB k bits

if k = 8

∴ Hf(key) = 11011101

Hash function of

- if M is in power of 2, key will be LSB only that lead to more collision.
- So donot choose M as a power of 2.

Ex 1010100101011111011110 k=7

1010101111111111011110 LSB

Hf(key1) = 1011101

Hf(key2) = 1011110

1. So, do not pick M as exact power of 2 because if $M = 2^k$ then Hash function of key = LSB K bits always.
but if M is not power of 2 then less chances of collision.

2. Pick the M value which is a prime no but it cannot be close to power of 2

Ques- which M value can be choosen for better result

(a) 61 (c) 729

(b) 523 (d) 1024

729 is choosen as it is not power of 2 & not close of to power o

Steps - 1) Assume all are prime no

3) choose one which is not close to power of 2.

[encoding - decoding
store - retrieve

2. Digit Extraction Method- M = 1000

key = 789123456

hashfunction(key) = 7 8 9 1 2 3 4 5 6
1 2 3 4 5 6 7 8 9

extract some digits

= 715. extract digit - (1, 4, 8)

0	
1	
:	
715	789123456
:	

the digits to be extracted are define in question. (you cannot extract 4 digi as size of stack is from 0-999

; Here no consideration is done to storage time.

- When you are extracting 3 digit, may lead to one digit, or two digit address (if one of them become zero)

• Collision may also occur here as it

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

but it is not producing that much less collision i.e no of collision is more here.

∴ Digit Extraction Method produces more collision as compa to Division Modulo Method (as for Division modulo, you req M = 25) but here only some bits have to be changed.

• We extract 3 digit address as 3 digit addresses >> 2 digit ad

3. Mid-Square Method- M = 1000

key = 84925

Step 1-

1. Square the given key
2. check Middle of obtained key (take 3 bit from mid as it will have majority of 3-bit addresses.

Making center counter here will be bit difficult, so it lead to less counter & less collision because itself key a larger no produces a more largeing no.

Hash function = $(84925)^2$

$= 7212255625$

225 255 Cas no exact mid possible.

store at any one address

225	7212255625

To Retrieve- (1) Square the key

(2) take the mid, go to mid & extract the key.

Counter is not easy because of large each & every no (digit) participate in the calculation of square

Folding Method- M = 1000 (0-999)

key = 123456789

(i) Fold Boundary Method

- fold the boundary (take k bit from the key from boundary)

k ≥ no of digit which have more address

for 3 bits - more address

(123)(456)(789)

123

789

912

If it lead to

(912)

912 → 1 → 913

or

191 → 2 → 193

folding boundary again & again

key stored | 912 | 123456789

(123)(45) → 123 + 45 = 168

Counter example = (123)(89)(789) → 912 → again collision

counter is bit tougher than bit extraction as 6 digit participate

(ii) Fold Shifting Method

- Fold it according to fold boundary but fold every bit

(123)(456)(789)

123

456

789

(1368)

136

8

144 - 1234567

144

fold shift is better than fold boundary as it require boundary 9 bit ie all the bits.

- Counter is also possible

⇓ Ex 789 456 123

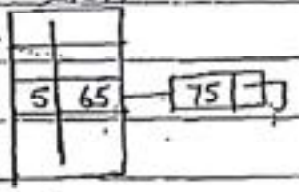
↓ since add^{n} satisfy commutative prop

ollision Resolution Technique-

1: chaining
2: open addressing

chaining

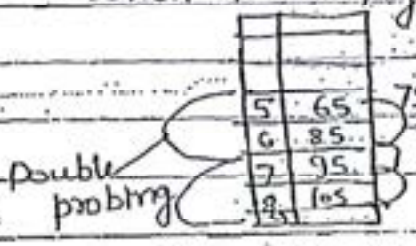
• outside (add linked list)



(If more than m keys are to be kept then chaining is used)

Open Addressing

• When only m keys are to kept
• inside (go to other slots which are empty
• Linear probing if 75 check for 5 fill 6→ fill and continous checking is done when no empty space



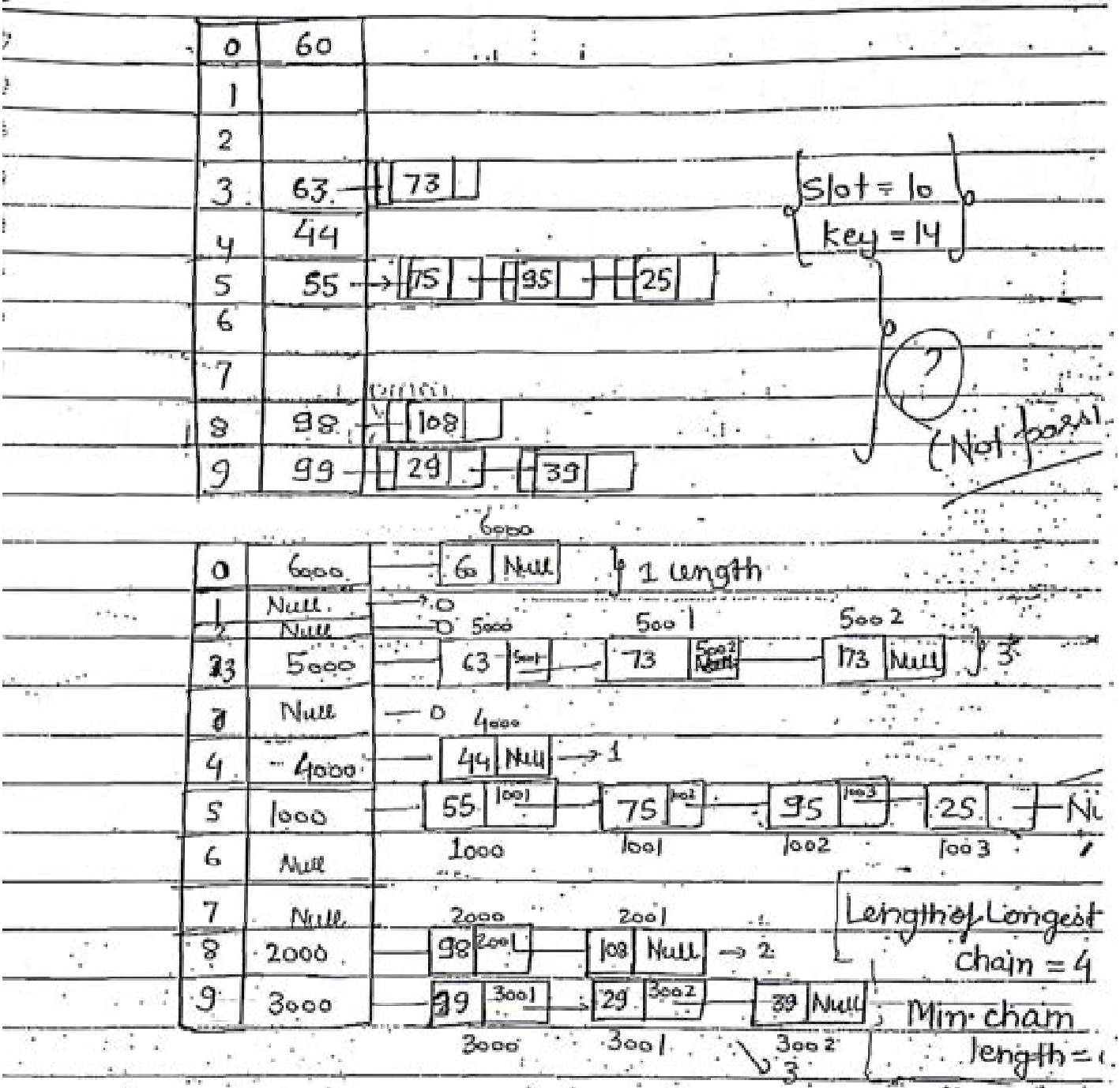
• Quoted: Quadratic probing where random jumps are made known as quadratic probing
• Double hashing when two values are directly jump

Chaining Process - m=10 (0-9)

key=(55, 98, 63, 75, 99, 73, 60, 95, 49, 29, 35, 73, 25, 108)

hf(key) = key mod m

Collision Resolution Technique = chaining



Here searching time increases.

∴ In Worst case, TC = O(n) as if all element occur in 1 slot.

but if you be using better hashing function then it can be reduced as O(1).

~~It is also~~

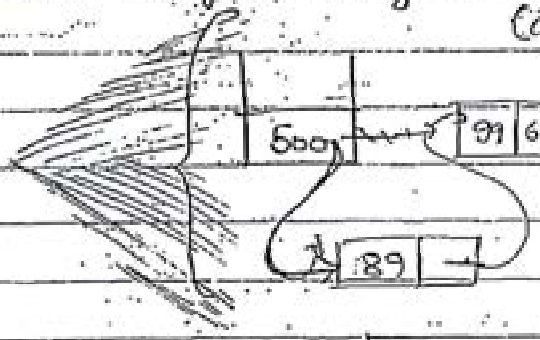
drawbacks

1. In chaining, space is wasted in the form of linked list even the space available inside.

2. The length of the longest chain possible is n. that lead to
Worst case searching time = O(n).
Best Case = O(1)
Avg. Case = O(n/2) = O(n)

3. the greatest advantage with the chaining is infinite no. of collision can be handled by i.e infinite no of keys can be stored because of linked list.

4. Insertion of one key will take O(1) time (BC, WC, AC)
(as insertion is done in linked list only.



n elements,
insertion will take O(1) time.

5. Deletion time will be
Best case = O(1)
Worst case = O(n) (as it may be at last
so you have to visit
each node)
If address of node which has to be deleted is given then
time = O(1) (BC)
O(n) - (WC)
[since you cannot go back. so you have traversed by start]

· Solve More (k)

is to be deleted & its address is provided with do
t then time to delete = O(1)
= O(1)

done in existence as in 10 slot you can store 100
ell.

ddressing- ~~sort~~

probing- M = 10 (0-9)

Hashing) Keys = (55, 99, 60, 75, 89, 42, 58, 69, 25)
hash function (key) = k mod m
C.R.T = Linear probing
bing (key, i) = (hf(key) + i) Mod m → attempt no
∀ i = {0, 1, 2, ... m-1}

0	60
1	89
2	42
3	69
4	
5	55
6	75
7	25
8	58
9	99

LP(55, 0) = (hf(55) + 0) mod 10 here sig
= 5 mod 10 1st atten
= 5 → (0) ex
LP(99, 0) = 9 — (0)
LP(60, 0) = 0 — (0) (in
LP(75, 0) = 5 (fail collision
LP(75, 1) = 6 (pass) ⇒ (2)
LP(89, 0) = 9 (fail)
LP(89, 1) = 0 (fail) ⇒ 2
LP(89, 2) = (pass)

5, 0) = 5 (fail)
5, 1) = 6 (fail) (2)
5, 2) = 7 (pass)

LP(42, 0) = 2 — (1)
LP(58, 0) = 8 — (1)
LP(69, 0) = 9 (fail)
LP(69, 1) = 0 (fail)
LP(69, 2) = 1 (fail)
LP(69, 3) = 2 (fail) (fail)
LP(69, 4) = 3 (p
(4)

Total collision = 9

You can store only M elements in M-slots.

To Retreive- same for 75 → goto 5 if there Yes
else (+1)

~~Quadratic~~

Double Hashing:- two hash functions - prn

Quadratic hashing (probing) - prn

Quadratic Probing:- M = 10 ~~as~~

Key (25, 39, 46, 55, 89, 23, 68, 70, 94, 78)

Hf(key) = key mod m $c_1 = c_2 = 1$

CRT = QP

$QP(key, i) = (Hf(key) + c_1 \cdot i + c_2 i^2) \bmod m$

quadratic partern

$\forall i = \{0, \ldots, M-1\}$

0	70
1	89
2	~~23~~ 78
3	23
4	94
5	25
6	46
7	55
8	68
9	39

QP(25,0) = 5 (0) collision

QP(39,0) = 9 (0) Total collision = 2

QP(46,0) = 6 (0)

QP(55,0) = 5 fail

QP(55,1) = 7 (pass) (9)

QP(89,0) = 9 fail (1)

QP(89,1) = 1 Pass

QP(23,0) = 3 (0)

QP(68,0) = 8 (0)

QP(70,0) = 0 (0)

QP(94,0) = 4 (0)

If we choose li
probe = 4 colli
i.e collision de

QP(56,0) = 6 (
Q(56,1) = 6+2 = 8
Q(56,2) = 6+5 = 1
~~Q(56,3) =~~

$\varphi P(78,0) = 8$ (fail)
$\varphi P(78,1) = 0$ (fail)
$\varphi P(78,2) = 14$ (fail)
$\varphi P(78,3) =$ fail 0
$\varphi P(78,4) = 8$ fail
$\varphi P(78,5) = 8$ fail
$\varphi(78,6) = 0$ (fail)
$\varphi(78,7) = 4$ (fail)
} 12 collis
$\varphi(78,8) =$ ~~2 fail~~ fail
~~at~~ $\varphi(78,9) =$ fail

∴ Total collision = 12

In Worst case, M-1 to 0 is covered ∴ O(m) you have to check.

∴ Worst case = O(m)

• Quadratic probing is faster than linear probing but it will generate inaccurate result (as gap may be there but you are not able to keep the element)

If Linear Probing is used, if anempty slot is present, elemen will surely get stored.

In Linear Probing, Worst case searching time = O(m)
Best case = O(1)
Average case = O(m)

Double Hashing:- M = 10

keys = (25, 39, 46, 55, 89, 23, 68, 70, 94, 78)

$hf_1(key) = key \bmod m$

$hf_2(key) = 1 + (key \bmod (m-2))$

CRT = Double hashing.

$DH(key, i) = (hf_1(key) + i \cdot hf_2(key))$

$\forall\ i = (0 \text{ to } M-1)$

0	70
1	89
2	78
3	55
4	94
5	25
6	46
7	23
8	68
9	39

$DH(25,0) = 5$

$DH(39,0) = 9 + 0(\ ) = 9$

$DH(46,0) = 6 + 0(\ ) = 6$

$DH(55,0) = 5 + 0 = 5$ fail

(1) $DH(55,1) = 5 + 1 \cdot hf_2\ 55 = 5 + (1 + 55 \bmod 8$

$= 5 + 8 = 13 \bmod 10 = 3$

$D(89,0) = 9$ (fail)

(1) $D(89,1) = 9 + (1 + 89 \bmod 8 = 11 \bmod 10$

$= 1$

$D(23,0) = 3$ (fail)

$D(23,1) = 3 + (1 + 23 \bmod 8$

(3) $= 11 \bmod 10 = 1$ (fail)

$D(23,2) = 3 + 2(8) = 19 = 9$

$D(23,3) = 7$

Total collision = 7

$D(68,0) = 8$ (pass) (0)

$D(70,0) = 0$ (pass) (0)

$D(94,0) = 4$ (pass) (0)

$D(78,0) = 8$ (fail)

(2) $D(78,1) = 8 + 1(1 + 6) = 15 \bmod 10 =$

$D(78,2) = 8 + 2(7) = 22 \bmod 10$

$= 2$ (pass)

Problem 1 Apply all 3 open addressing strategies.

$m = 10$

key = 25, 15, 90, 75, 69, 52, 44, 58, 85

0	90
1	59
2	85
3	
4	44
5	25
6	15
7	75
8	58
9	69

(Linear probing)

Total collision = 1+2+2 = 5

(In linear probing, order of occurence in sequence matters.)

If 25 came first it occupy the first pos has 5

- If you want to find any element find if gap occur then element is not present in table.

Quadratic probing

0	90
1	75
2	
3	85
4	44
5	25
6	
7	15
8	58
9	69

Collision = 1+2+1+6 = 12

3+1

1. We are not wasting space outside in the form of linked
2. Searching time = O(1) for Best case
   O(m) for worst case
3. Insertion time = O(1) for Best case
   O(m) for worst case
4. Deletion time = O(1) for Best case
   O(m) for Worst case
5. Whenever delete a particular element, place $ otherwise at a time of searching gap occurs & it may lead to mis-searching.

~~if yours table~~

(5)- If you delete one element, it will create trouble to other elements but we can manage with help of $ symbol if more dollar symbol than reHash again.

Quadratic Probing:-

1. No space is wasted outside in form of linked list.
2. Search time ⇒ O(1) = BC
   O(m) = WC
3) Insertion time ⇒ O(1) = BC
   O(m) = WC
3) deletion time ⇒ O(1) = BC
   O(m) = WC
4. deletion of one element will trouble to other element but can manage with $, if more deletion, than rehash again.

Double Hashing-

0	90
1	69
2	
3	15
4	
5	25
6	
7	59
8	58
9	75

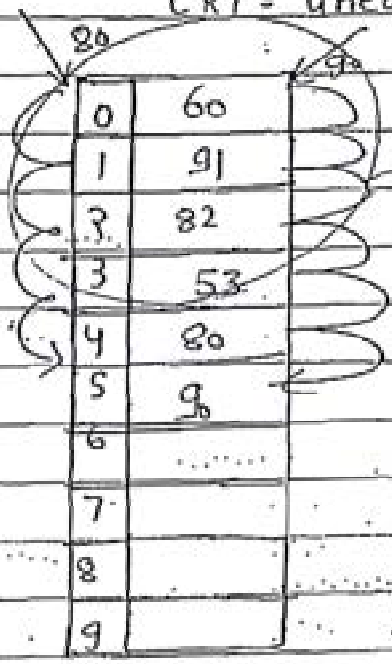
DH(25,0) = 5
DH(15,0) = 5 fail
DH(15,1) = 5+8 = 13 = 3
DH(90,0) = 90
DH(75,0) = 5 fail
DH(75,1) = 5+11

EX3 – m=10

keys = 53, 60, 80, 82, 91, 80, 90

hf(k) = k mod m

CRT = linear probing



primary clustering

LP(80, 0) = 0+0 = 0
0+1 = 1
0+2 = 2
0+3 = 3
0+4 = 4

LP(90, 0) = 0+0
0+1
0+2
0+3
0+4
0+5

Here both start with same addr follow same path unnecessarily

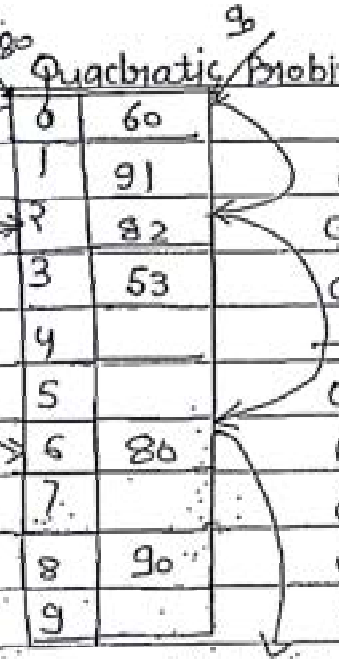
Avg = $\frac{1+2+3+\cdots+m}{m} = \frac{m(m+1)/2}{m} = \frac{m+1}{2}$

(1 + 2 + 3 + ... + m: 1 person, ..., m-th person; e person)

Primary Clustering: If two keys same initial hash address they both will follow same path unneccessarily in linear fashion because of this avg time increases, this problem is known as primary Clustering.

- Linear probing is suffering with primary clustering.
- Avg Case = $\frac{1+2+3+\cdots+m}{m} = \frac{m(m+1)}{2m} = \frac{m+1}{2}$ as it suffers with the problem primary Clustering as 2 will suffer with problem of 1.

y using Quadratic Probing



QP(80,0) = 0 (fail)

QP(80,1) = 0+1+1 (fail)

QP(80,2) = 0+2+2² = 6 (true)

QP(90,0) = 0 (fail)

QP(90,1) = 0+1+1 fail)

QP(90,2) = 0+2+2² = 6 (fail)

QP(90,3) = 0+3+3² = 12 = 2 (fail)

QP(90,4) = 0+20 = 0 (fail)

and so on QP(90,5) = 0 (fail)

QP(90,6) = 2 (fail)

QP(90,7) = 8 (true)

Here ~~primary~~ clustering occurs as what path is followed by 1 element will definitely followed by 2 element but here Avg. case deq decrease due to large jump Here clustering is known as secondary clustering.

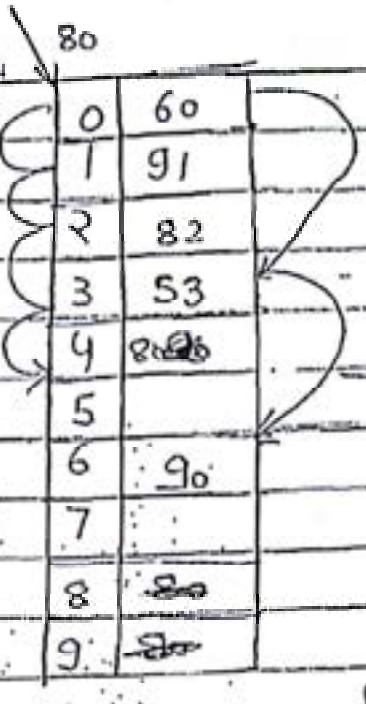
Secondary Clustering:- If two key contain same starting hash address, they both will follow same path unneccessarily in quadratic manner, because of this qeavg. avg. searching time increases (it is less than linear search probing time) ; this problem is known as secondary clustering

Quadratic probing is suffering with secondary clustering

Avg case = O(m) but actually it is less than $\left(\frac{m+1}{2}\right)$

(Asymptotically same but mathematically less)

Double Hashing:



DH(80,0) = 0
DH(80,1) = 1
DH(80,2) = 2
DH(80,3) = 3
DH(80,4) = 4

DH(90,0) = 0
DH(90,1) = 3
DH(90,2) = 0+6

Here path is diff with initially address (same).

Here diff path is carried out with same initial address. no path will be same.

DH(100,0) = 0.
DH(100,0) = 0+5 = 5 (Free)

$$Avg = \frac{3+5+7+\cdots+2}{m} = \frac{m}{m} = O(1)$$

(DH

(As a particular slot is covered by single element only. It might be possible that there must be some cases which lead to redundancy there in avg. case, we have to consider for all the given case) so it on an avg. it is said that it is repeating one time).

In Double Hashing:- Eventhough two key contain same starting hash address, they both will follow diff. path because second level hash fn will give diff. value.

because of this reason 99% of redundancy is eliminated.

• Avg case = 3 + 5 + 7 + ... + 2
(3: slot covered by I element; 2: slot covered by II element)

∴ on an avg = $\frac{m}{m}$ ⇒ $\frac{cm}{m} = O(1)$

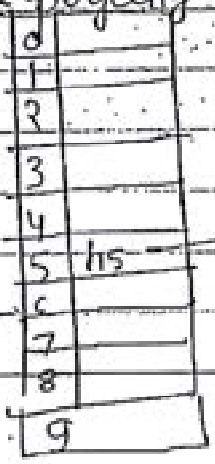
Avg. case = O(1)

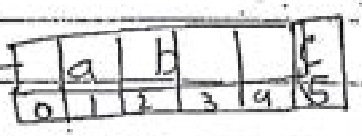
(as only few key will be repeating only
let 4 key repeated
total = $\frac{4m}{m}$
= O(4)

∴ In double hashing, secondary clustering problem is present littlebit (1%)

Note - Using Perfect Hashing, you will get Worst case Searching time = O(1)

in hashing if collision occur to a, b, c at slot 5 they will go to 5 which have another his which is best hash fn and a hash table to store those three element then collision is removed perfectly





∴ Total table = m+1
Total fn = M+1
∴ Worst case = O(1)

Space = $O(m^2)$ ✓ m

You cannot decide of size so use m size
Else use linked list
so space increases but time decrease

- m-slots n-keys
- m slots – will store n keys
- 1-slot will store $\frac{n}{m}$ keys

⇓

Load factor – no of keys getting stored in 1 slot

Load factor (α) = $\frac{n}{m}$ keys per slot

Note 1 - the expected no of probes (attempt) in an unsuccessful search of an open addressing technique is

$$\left\lceil \frac{1}{1-\alpha} \right\rceil$$

Note 2 - the expected no of probes in successful search of an open addressing technique is $= \left\lceil \frac{1}{\alpha} \log_e \frac{1}{1-\alpha} \right\rceil$

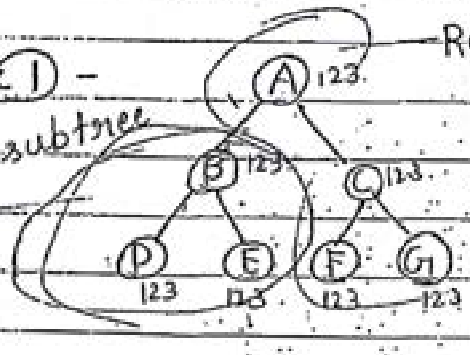
(CoreMan)

TREE AND GRAPH TRAVERSALS (2 Marks)

Tree Traversals -

1. Preorder (Root LST RST)
2. Postorder (LST RST Root)
3. Inorder (LST Root RST)

Ex. 1 -



tree is best solved by recursion as of having almost same quality.

traversal - visiting each node once in given tree.

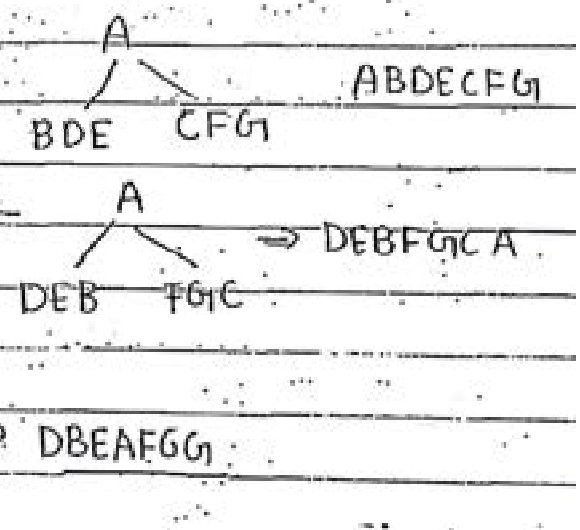
Note: Left & Right tree cannot changed i.e left will always come before Right subtree

if changed → 3!

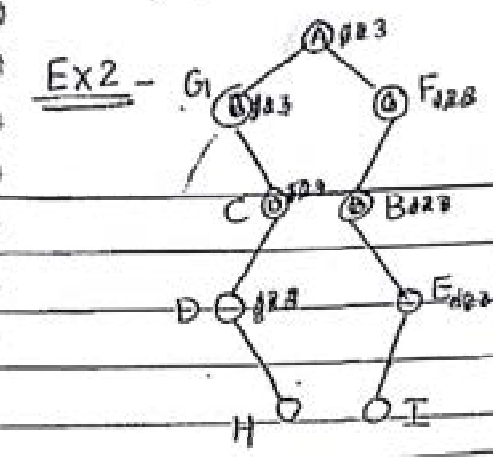
Preorder - ABDECFG

Postorder - DEBFGCA

Inorder - DBEAFCG



Ex2 -



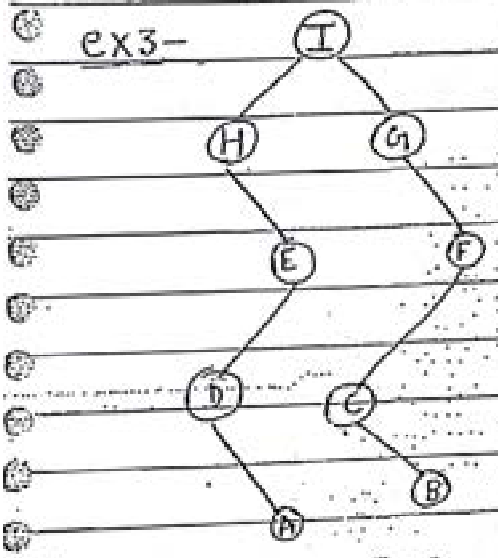
Preorder - AGCDHFBEI

Inorder - GDHC ABIEF

Postorder - HDCGIEBFA

GBGDHCABIEF

Ex3 -



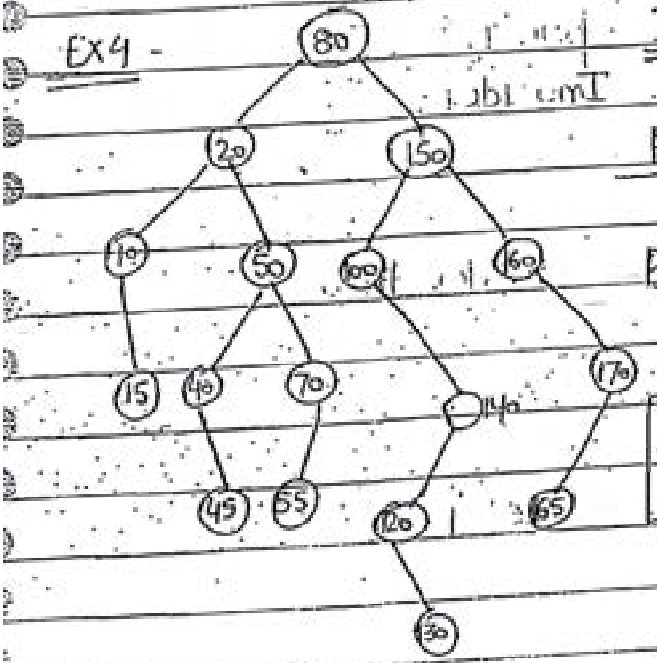
Preorder - IHEDAGFCB

Inorder - HDAEIGCBF

Postorder - ADEHBCFGI

To sort an array
- Create BST
- do inorder traversal

Ex4 -



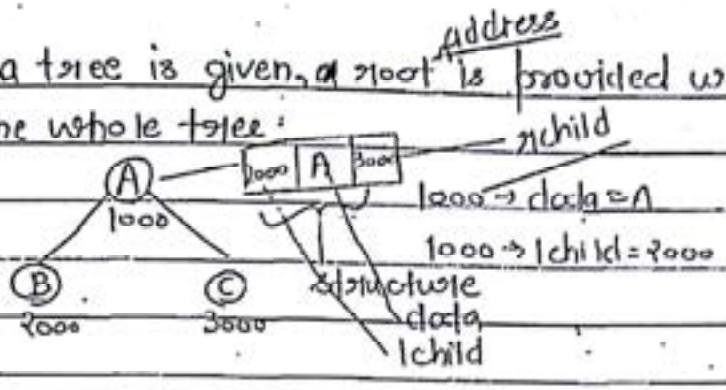
Inorder - 10 15 20 40 45 50 55 70 80 100 120 130 140 150 160 165 170

Preorder - 80 20 10 15 50 40 45 70 55 150 100 140 120 130 160 170 165

Postorder - 15 10 45 40 55 70 50 20 130 120 100 165 170 160 150 80

Note -
BST after inorder traversal will gen ate Ascending order

"Whenever a tree is given, a root (address) is provided which can be used to traverse the whole tree:




```
Preorder(root)
{
    printf("root->data);      O(1)
    preorder(root->lchild);   T(n/2)
    preorder(root->rchild);   T(n/2)
}
```

root = 1000
preorder(1000)
A
preorder(1000->lchild)
preorder(2000)
and so on.

```
postorder(root)
{
    postorder(root->lchild);
    postorder(root->rchild);
    printf("root->data);
}
```

```
Inorder(root)
{
    Inorder(root->lchild);
    printf(root->data);
    Inorder(root->rchild);
}
```

for preorder of a binary tree: $T(n) = 2T(n/2) + O(1)$ (by master theorem)

$= O(n)$ (Best case)

Worst partition - $T(n) = T(n-1) + T(0) + O(1)$

$\approx O(n)$ (Worst case)

for preorder, postorder, Inorder: Avg case $= O(n)$

WC, BC, AC same as each node has to visit once

Reccurence relation can also tell about stack spc
If $T(n/2) + \log n$
$T(n-1) + n$

Note –

* Preorder Inorder Postorder will take O(n) time on n-node binary tree (WC, AC, BC), because each node has to be visited once.

Stack Space.
~~Space complexity~~ = O(logn) for balanced tree (Best Case)
O(n) for unbalanced tree (Worst case)

Ques – if a BST is given, time to get sorted order – O(n)

Ques – if a BST is given time to sum element 20 to 50
→ O(n) – Worst case
O(1) – Best case (start from 20 when 20 came & add
50)

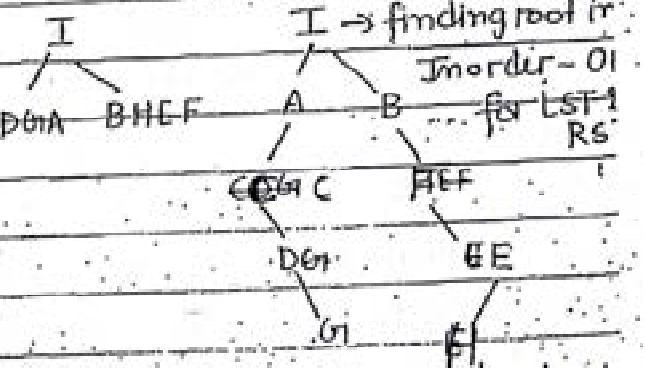
Ques – Consider the following binary tree data–

Preorder: IACDGBFEH

Inorder: CDGAIBHEF

Postorder– GDCAHEFBI

→ finding root



Inorder traversal cannot judge for proot because it cannot trees are not balanced always

Preorder can give root as well as postorde can give it

Inorder is used to give Left subtree & right subtr

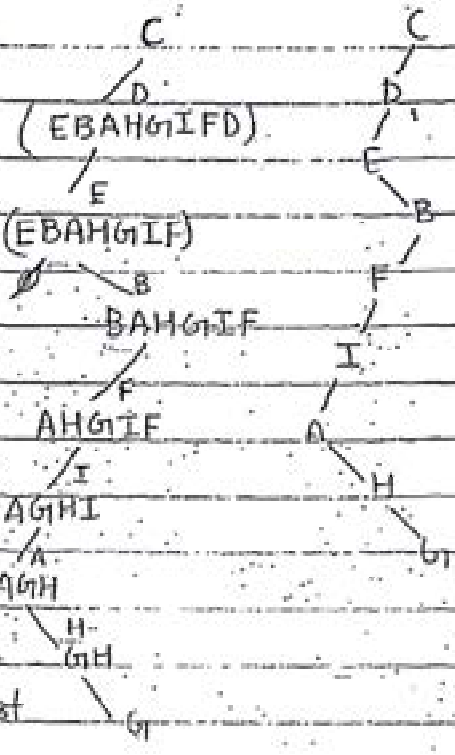
∴ Total complexity = n × O(n) [at 1 lvl (for LST & RST)]
= $O(n^2)$

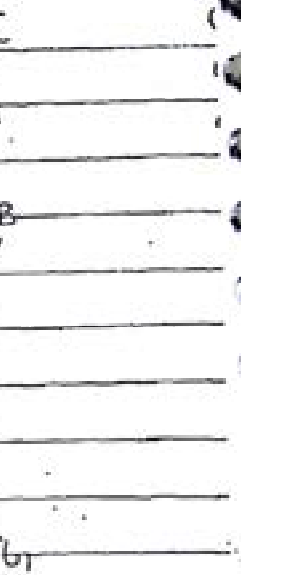
Ques- Consider the following binary Tree data-

Postorder - GHAIFBEDC

Inorder EBAHGIFDC

preorder - CDEBFIAHG





(X = sometime possible)

To find out directly any order find out root find its pos^n in inorder case the digit or no of element in inorder try to find them in postorder just reverse the order and vice versa.

shortcut Not applicable everywhere

Ex- Here postorder & Inorder given, C its position in inorder rightmost so C has empty right ST. then go to postorder get the element in LST of C in Inorder take them & reverse them to obtain post order.

Note - To construct Binary tree for the given preorder inorder or postorder inorder will take $O(n^2)$ time (n times linear search to find LST & RST in given inorder)

(n^2+n+1)

to find LST & RST — to find root n time

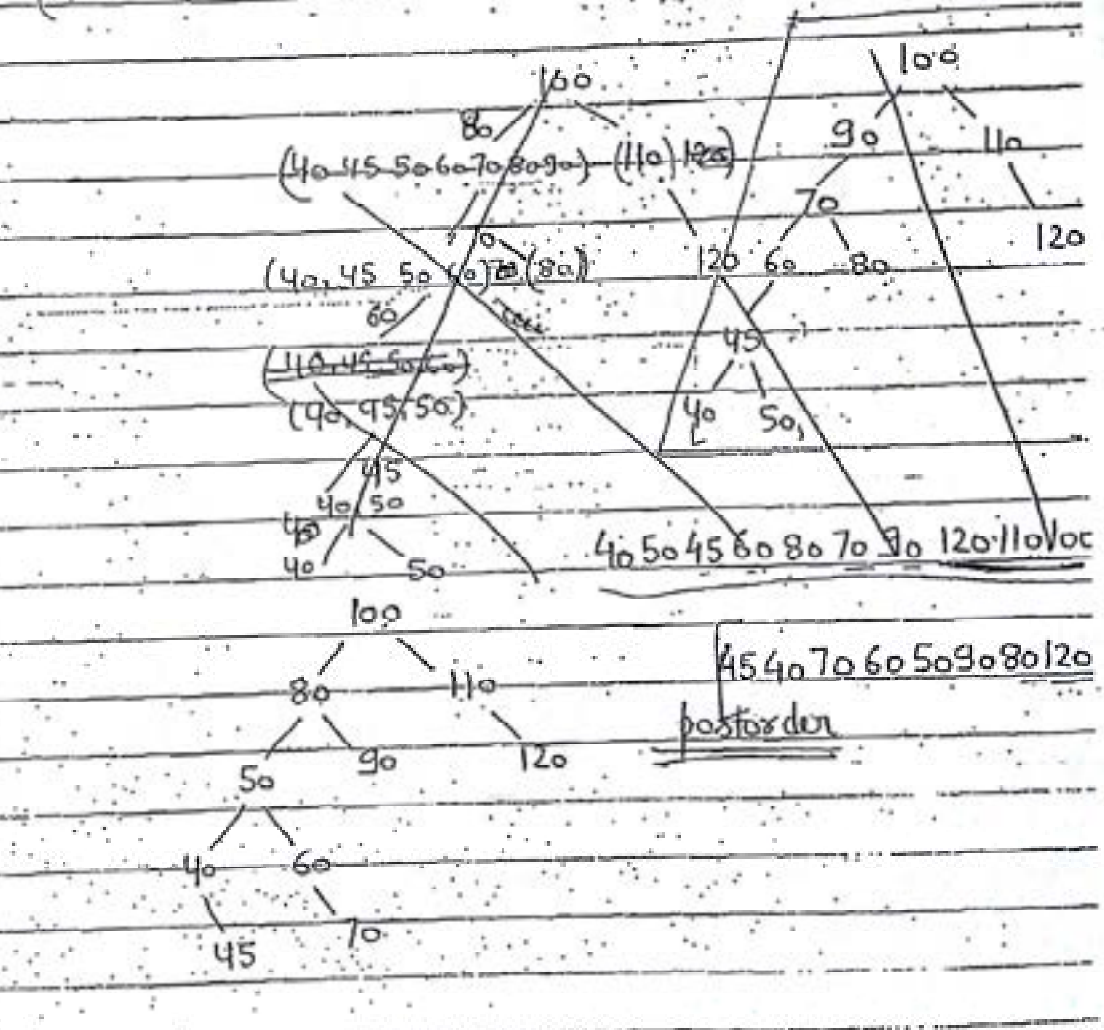
We cannot apply binary search as given sequence is not sorted but if BST it will take nlogn tr

Ques - Consider the following BST data-

Pre: 100 80 50 40 45 60 70 90 110 120

Post: ~~90 70 60 45 40 50 80 120 110 100~~

In- (40 45 50 60 70 80 90 100 110 120 (A.O)



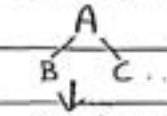
Note - for the given preorder inorder for postorder inorder to construct equivalent BT require (nlogn) time if inorder is already sorted

preorder: A B C
N L R

postorder: C B A
R L N

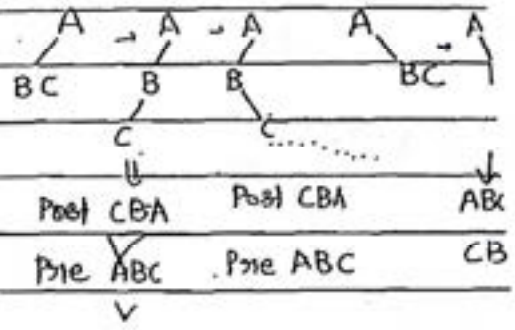
root = A

- Let both RST & LST present



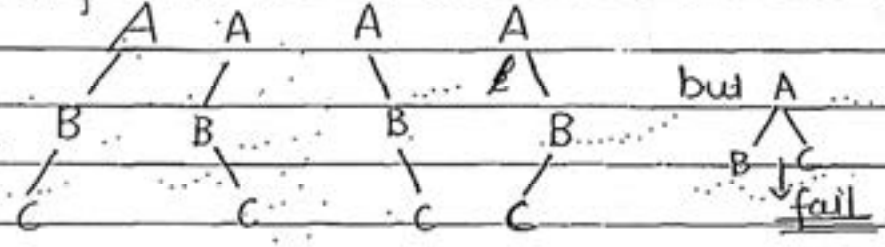
Not possible as postorder = BCA ✗

- if 1 absent(R)



- if 1 present

So possible trees are



but

Note

If Preorder Inorder given - unique binary tree ~~bst~~ / Binary tree / Unique BT

postorder Inorder given - Unique Binary tree

if preorder postorder given - more than one binary tree

↳ Binary tree possⁿ but not

To construct unique binary tree, Inorder is required (compulsor

When pre, post is given, manual checking is done To each no two possibilities either going to left or right so total possib for n node = $O(2^n)$

Upper bound as root will have only one possibility.

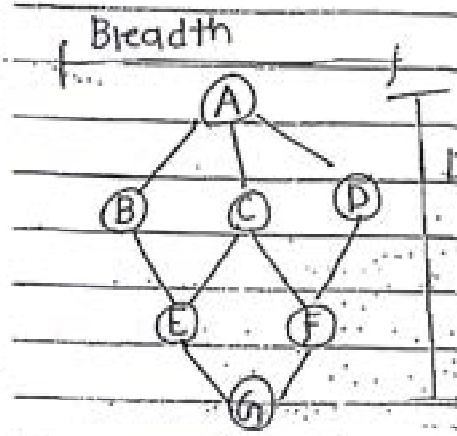
To find out cycle initially flag = 0 then if visited flag = 1 if again encounter flag 1 then it is a cycle.

Graph Traversal

1. BFT (Breadth First Traversal)
2. DFT (Depth First Traversal)

Total Graph Traversal time = O(V+E)

⇓ visiting each vertex & each edge



We donot consider edge in tree & edge is less than Evertices

Tree = (V+E)
V+V-1
= 2V-1 ≈ O(V)

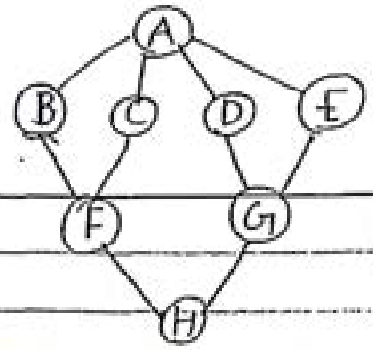
Breadth first traversal - A B C D E F G

Depth " " A B E G C F D

Breadth first Traversal:-

- Tree traversal are unique but graph traversal not.
- visited is required as graph contains cycle and there is no mean of visiting a vertex again & again.
- Queue is used in BFT as we need FIFO structure.

while loop is covering vertex & for loop is caring for edges.



Ex

Initially Visited = 0 0 0 0 0 0 0 0 (A B C D E F G H)

Starting with A

Queue empty

front = rear = 0

1 0 0 0 0 0 0 0 (A)

Q: A

x = A (delete)

ω = B C D E

Visited: 1 1 1 1 1 0 0 0 (A B C D E F G H)

Q: B C D E

BFT(v)

{

Visited[v] = 1

add(v, Q);

while (Q is not empty)

{

x = delete(Q)

print(Q)

for all ω adjacent to x

{

if (Visited[ω] = 0)

Visited[ω] = 1

add(ω, Q)

}

}

O/P → A B C D E F G H

(order of deletion of element)

B deleted ω = A F → add

Q: not added again

Visited: 1 1 1 1 1 1 0 0 (G H)

Q: C D E

C → deleted ω = A

Q: D E F

D → deleted ω = A

Q: E F G

E → deleted ω = A

Q: F G

F → deleted ω = B

Q: G

1

Q: H

empty

loop end

Note –

- To implement BFT, we use queue data structure.
- Total complexity = O(V+E) (time) (BC, WC, AC)

if adj matrix is used TC = $O(V^2)$

Space → i/P + extra

if graph is adjacency list

(V+E) (V+V)

queue visited array

= (V+E+2V)

= 3V+E

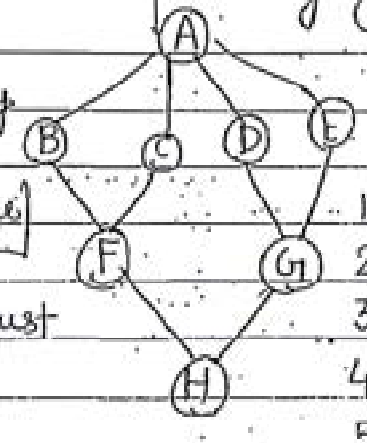
= O(V+E)

= $O(V^2)$ if adj. matrix is used

- BFT is also known as Level order traversal i.e level by level printing of nodes. (means adjacency list)

A → next level (adj of A)

Problem 2 – Consider the following graph –

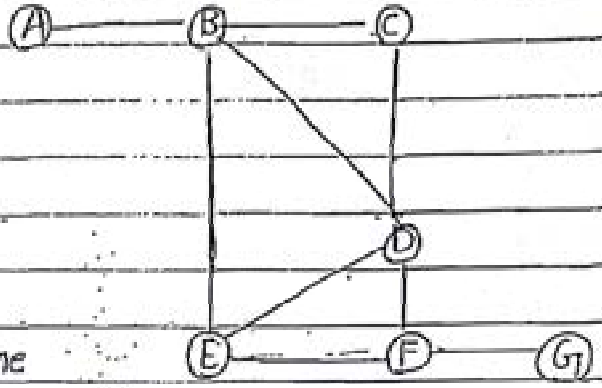


adj of any vertex may come in any order in which you arrange all adj of a vertex must come altogether.

Find correct BFT from the following:

1. ABCDEFGH (correct)
2. CAFBDEHG (correct)
3. ADCBEGFH (correct)
4. EAGCDFBH (false)
5. ABCDEGHF (Not correct)

Ques - Consider the following graph.



Whether these are correct BFTs or not

(a) EBDFCAG (correct)
(b) GFCEDBA (incorrect)
(c) CBDAEFG (correct)
(d) DBCFEAG (Correct)

Application of BFT :-

Ques - if a graph is given, how to check whether it is connect or not?

- ~~Draw~~ Apply BFT
- Check visited array
- if any 0 in visited array
- then graph disconnected.

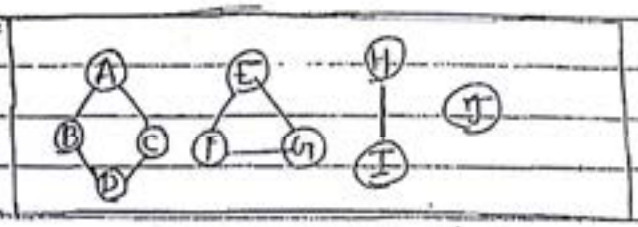
↓ to check for all

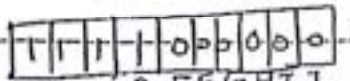
O(V+E)+V

if a linked list is given, to check whether it is connected

a linked list is also a graph but directed so apply the same procedure.

· We can verify given graph is connected or not





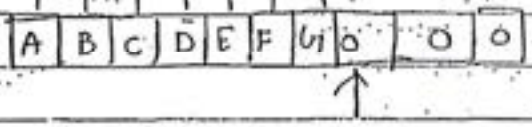
starting from A

→ no of disconnected component

∴ count=1 (initially zero)

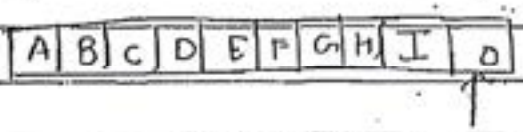
· When you get I zero, just stop & again apply the DFT again and so on, you can find ~~dis~~connected connected components can be find out

again



Count=2

again



Count=3

Count=4

∴ No of connected component = No of time BFT applied

∴ TC to find connected comp = O(V+E)

· Using BFT we can find out connected components in the given graph.

· if a null graph G(V, E) - no of connected component = (V)

3. to check for cycle in a given graph
- Apply BFT
→ If visited[i] = 1 & it encounter again then there is a cycle.

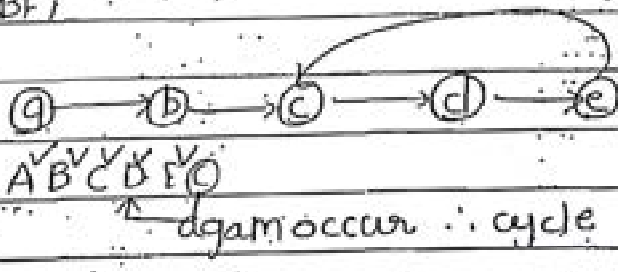
Time complexity = O(V+E) (as BFT work)

4. Whenever kruskal is going on edges are added to MST we have to check for occurence of cycle too as no cycle must occur, then by side BFT also working to check for visited[i] → it take constant time (for 1 edge)

$T(n) = E \log E + (V+E)$ ← for BFT cycle check

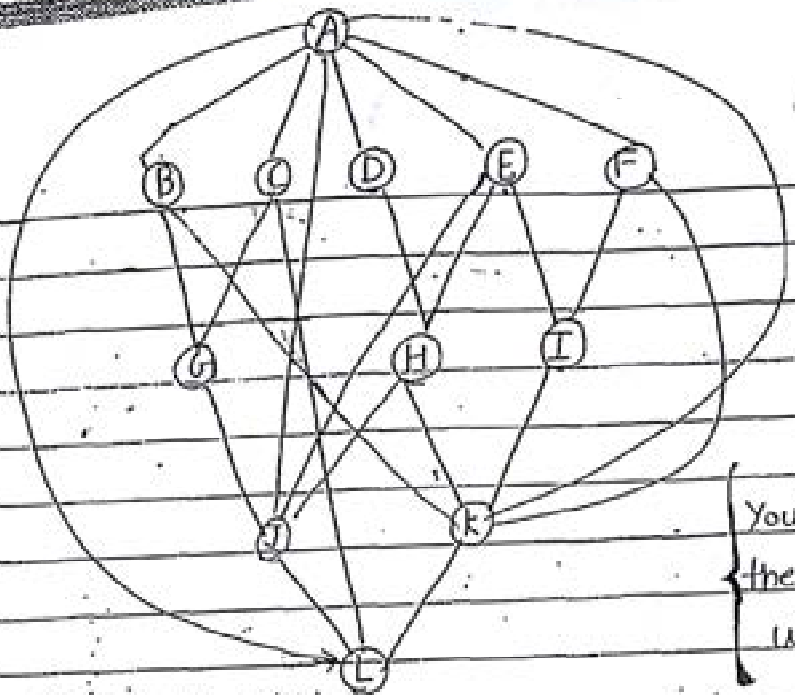
3. Using BFT, we can check given graph contain cycle or not.

If a linked list contain cycle, it can be checked easily by applying BFT



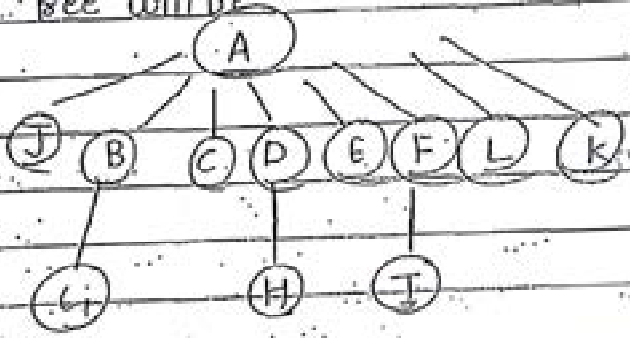
4. If an unweighted graph is given & single source shortest path is not to be finded, then BFT is the efficient one algo as it will find the sh

∴ Using BFT, we can find out shortest path from given source to every vertex in given unweighted graph or the graph having equal edge weight (+ive)



You can easily find out the shortest path using O(V+E) time.

the BFT tree will be



unweighted graph - default weight = 1

Ques- if a unweighted graph is given; the shortest path to be calculated from given source which data structure is used?

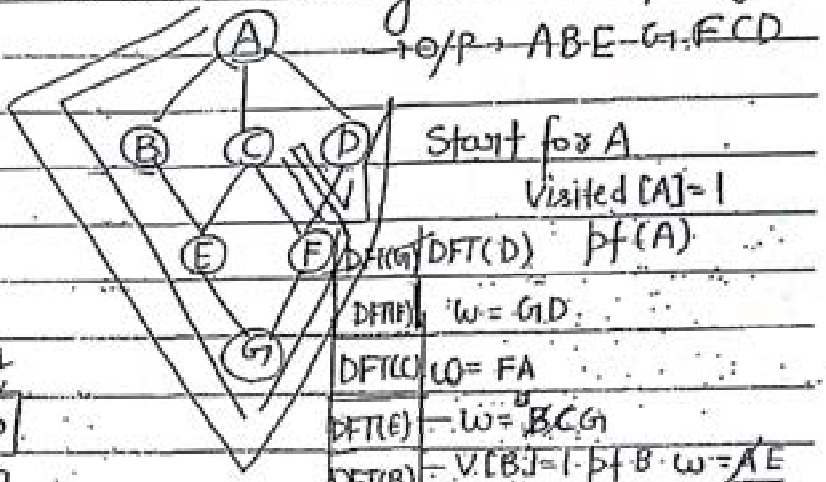
Queue

5. Using BFT we can verify a given graph is Bipartite graph?

Depth First Traversal

- It uses stack as recursion occur here.
- it will also maintain a visited array to check for cycle.

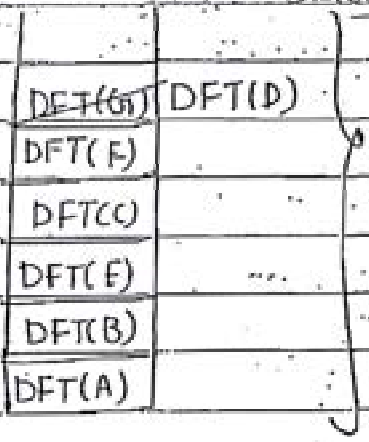
size V.



Visited

1	1	1	1	1	1	1
A	B	C	D	E	F	G

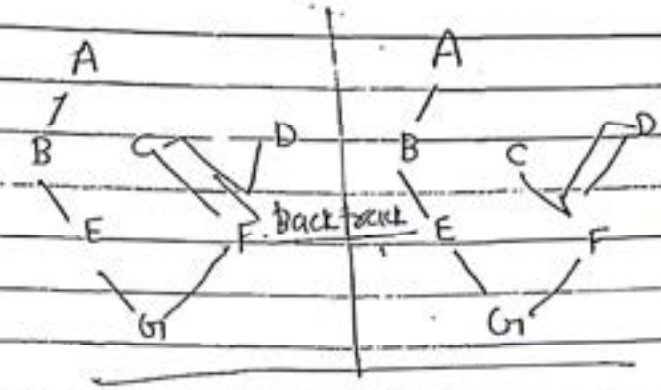
and soon stack size = O(n)



order of adjacent vertex can change the column traversal.

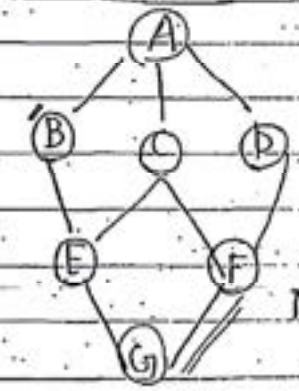
1. To implement DFS we are using stack.
2. Time Complexity = no of functional class calls * work done at each func call
   = V + E (as if work is summed up)
   = O(V+E)
3. Space Complexity = i/p + extra
   = V+2E (stack V, visit V) = O(V+E)
   V+2E ↓ for adjacency list

```
DFT(v)
{
1. Visited[v] = 1
2. pf(v);
3. for all w adj to v
   {  1. if (w is not visited)
      2.    DFT(w);
   }
}
```



In case of Adjacency Matrix = $O(V^2)$
(space as well as time complexity)

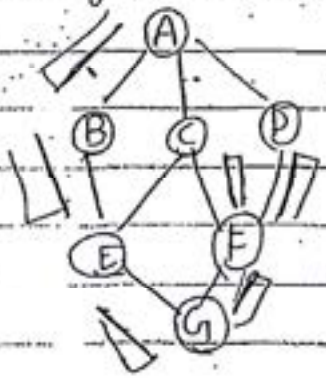
Problem - Consider the following graph -



Find Correct DFT's

(a) ABEGFCD

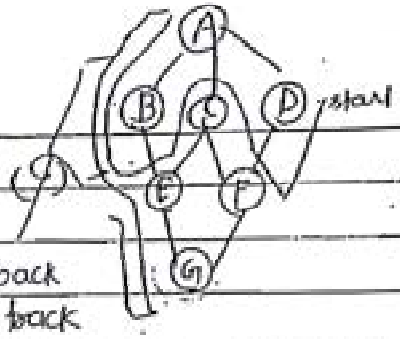
No of back track



(b) ABECFDG
↓
backtrack occurs here

Total Backtrack = 6

You can take any of the adjacent of a particular element.



DFCEBAG (correct)

(d) GFDABECG (No Back-back) (correct)
(Stack Size = n)
max. stack size = O(n)
↓ When no backtrack occurs

(e) GIECFABD (Incorrect) correct - GIECEDAB

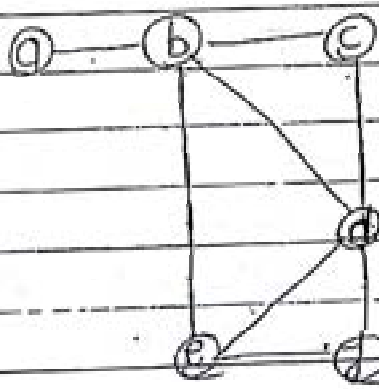
(f) ACFGEBD (correct)

(h) BACFDEG → (fail) Incorrect) correct - BACFDGE
because at D when possibilities ended we back track to F & F has G to be done but it is first computing G.

Ex. Back Tracking is done when all possibilities ended
Backtracking is nothing but poping

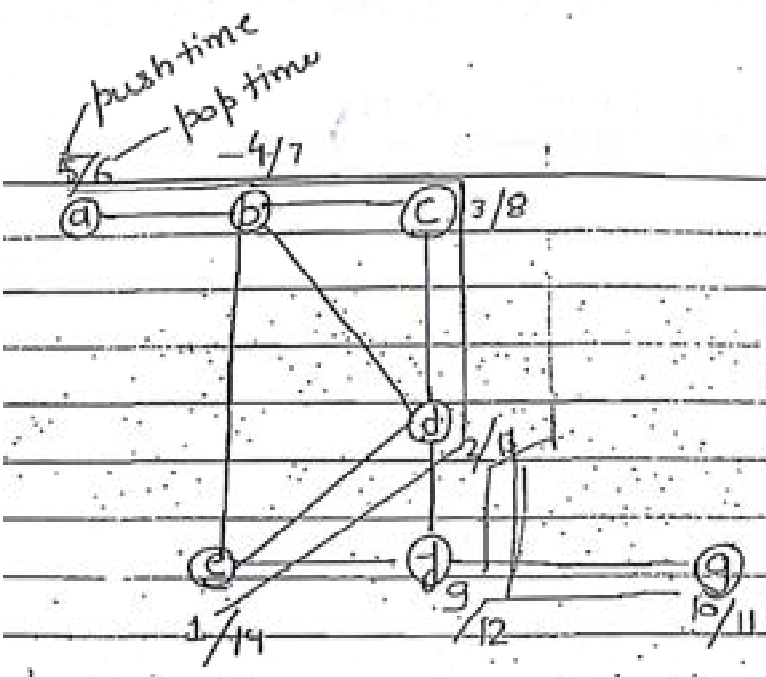
In DFT, if I print D after B, what is the relation b/w them
No relation because of backtracking any thing possible in DFT but BFT have adjacency.

Ques - Consider the following graph -



Check whether they are DFT or ~~BFT~~ or not.

1) edcbafg (DFT)
2) fdebagc (incorrect)
3) abdfgce (incorrect)
4) dcbefgga (correct)
5) befgdcca (DFT)



edcbafg

how many element where the diff b/w push & pop time is 1 (Immediate back track)

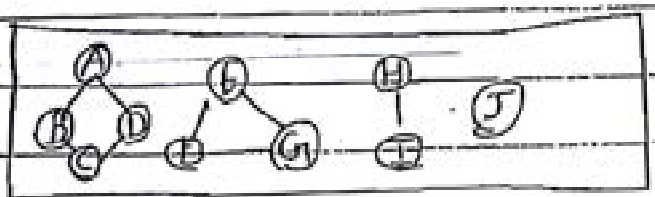
∴ Total element = 2 (a, g)

1/14 2/13 3/12 4/9 5/8 6/7 10/11
d c b e f g a

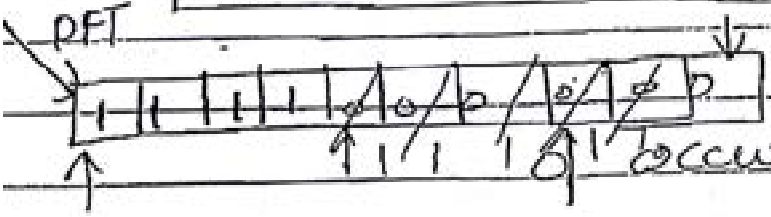
1/14 2/11 3/10 4/5 6/9 7/8 12/13
b e f g d c a

Applications of DFT-

- will check where graph is connected or not



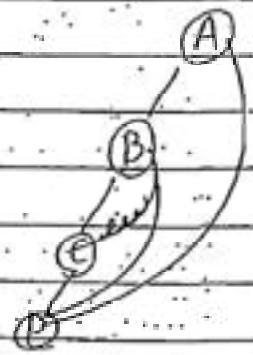
if start for A



O(V+E)

2. We can verify given graph contain cycle or not
O(V+E)

3. We can find out no of connected components
count = no of time DFT applied.

4. 

Here shortest path = 1 (1)
but if we take

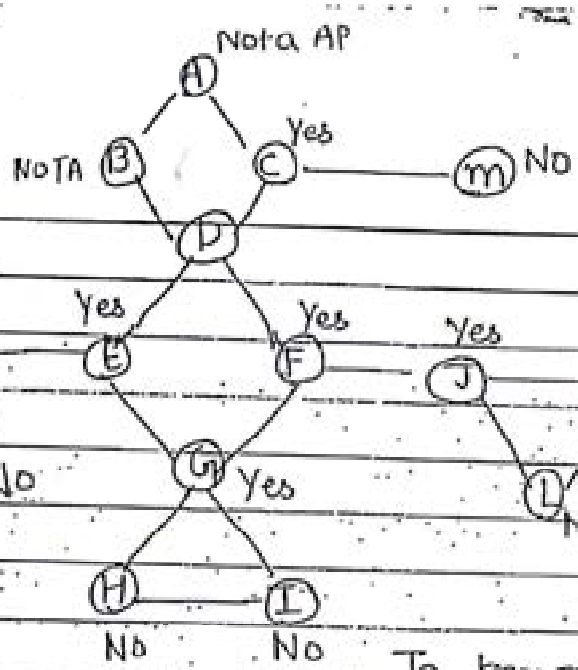
= 3 length path

So, DFT cannot (may or may not) work finding single source shortest path for given unweighted graph.

5. Using DFT, We can check given directed graph is strongly connected or not.?

6. Using DFT we can verify a vertex is articulation point or not.

Articulation point - if the deletion of an ~~edge lead~~ vertex ~~als~~ along with its edge lead to discon tion of graph known as articulation point.



To known whether a vertex is any articulation point-

1) Apply DFT on given graph
2) Delete that vertex & edge.
3) Again apply DFT
4) If visited contain 1 for all except that delete vertex then that is an articulation point

Total Complexity = $O(V+E) + O(V+E)$
$= O(V+E)$

Bipartite Graph using BFT

first time visited in one set

second time another set

(level by level storing of elements

trongly connected using DFT

r finding strongly connected component

you can use the concept of strongly connected graph

rticulation point

No of articulation point = No of strongly connected component + 1

Total complexity = E(V+E + V+E)

↳ for articulation point

$= EV + E^2$

$= O(E^2)$

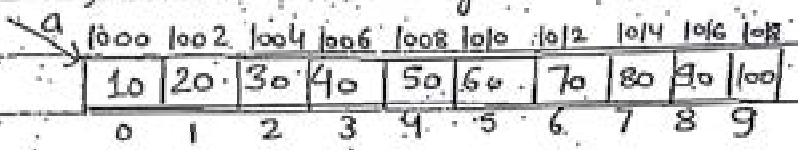
Programming-1

Array- 1-D Array

Ex-1

int a[10] = {10, 20, 30, 40, ..., 100} a will store base address.

↳ declaration (memory creation)



1 to 7 → How many elements = 7-1+1 = 7

-5 -4 -3 -2 -1 0 1 2 3 4 4-(-5)+1 = 10 elements

2-(-2)+1 = 5 elements

5 element occupy - 1010 & 101

print a = 1000 (base address)

Loc[a[5]] = base address + (size of data type)*(5-0)

= 1010

* 1010 - to access that particular M/M location)

Loc[a[9]] = base address + size of int * (9-0)

= 1000 + 2×9 = 1018

only 9-0 bcz you dont we to cross 9th element)

Ex2-

a[55 .... 550] Total element = 550-55+1 = 496

Base address = 990

Size of an element = 10 byte (C)

Loc[a[450]] = Base address + C*[450-55] (before 450 from 55)

= 990 + 10 × 395

= 3950
990
= 4940

*array must be continous no matter whether indices are (+)ve or (-)ve.

Ques a[-55 to 55] BA = 1000 C = 5

Loc[A[5]] = BA + [5-(-55)] x 5

= 1000 + 60 * 5 = 1000 + 300

= 1300

ex4 :- a[lb .. ub] BA = Base address

lb = lower bound, ub = upper bound, size = C

total element = ub - lb + 1

$$Loc[a[i]] = BA + C * [i - lb]$$

array index when started from 0, no need to do subtract at time to calculate location but when it start with 1, a subtract has to be done. So less time is required to calculate the value when starting from 0.

a[1 .. 10] Loc[a[5]] = BA + (5-1) * C (offset calculation)

a[0, 10] Loc[a[5]] = BA + (5-0) * C

* Array index is, by default, will start from 0 only because no need to calculate offset value

2-D Arrays:- int a[4][5]

row (0=3) (0=4) column

(5-1)+1 = 5 column

int a[1..4][1..5] = total row (4-1)+1 = 4 Row

	1	2	3	4	5
1	a_{11}	a_{12}	a_{13}	a_{14}	a_{15}
2	a_{21}	a_{22}	a_{23}	a_{24}	a_{25}
3	a_{31}	a_{32}	a_{33}	a_{34}	a_{35}
4	a_{41}	a_{42}	a_{43}	a_{44}	a_{45}
	a_{51}	a_{52}	a_{53}	a_{54}	a_{55}

Total element = 20 element

In m/m 40 bytes for continous m/m (no separate row & colu is present)

1000 02	04	06	08	10	12	14	16	18	20	22	24	26	28	30	32	34	36	38	1000 to 1039
a_{11}	a_{12}	a_{13}	a_{14}	a_{15}	a_{21}	a_{22}	a_{23}	a_{24}	a_{25}	a_{31}	a_{32}	a_{33}	a_{34}	a_{35}	a_{41}	a_{42}	a_{43}	a_{44}	a_{45}	
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	
		row 1				row 2					row 3						row 4			

Row Major order:- When elements are getting stored row by ro

Ex-1

Q. C follows Row Major Order

Loc[a[4][3]] = 1000 + [(4-1)×5 + (3-1)] × 2 (visit row, visit col)

= 1000 + 2(5*(4-1) + (3-1))

= 1000 + 2(17+2) = 1000 + 2(21)

= BA + C[Column*(x - lb1) + (y - lb2)]

= BA + C[(ub2 - lb2 + 1)(x - lb1) + (y - lb2)]

Loc a[2][5] = BA + C[5*[2-0] + (5-1)]

= 1000 + 2[5+4]

= 1018

Loc a[i][j] = BA + C[(ub2 - lb2 + 1)(i - lb1) + (j - lb2)]

lb1 ub1 lb2 ub2

(i - lb1): visit correct row.

(j - lb2): visiting column

Ex-2 → 726 rows → 71 column

a[25...750][80....150] c = 10 byte

Row Major order BA = 1000

1) Loc(a[550][140]) = BA + c[(150 - 80 + 1) * (~~140 - 25~~)(550 - 25) + (140 - 80)]

525
1 71

525
36750
37275
375

= 1000 + 10[71(525) + 60]

= 1000 + 10[37335

= 373350 + 1000

= 374350

Ex 3 a[-25 ... +25][-50 ... 50] ; BA = 0 c = 1 Byte RMO

51 101

Loc a[20][30] = BA + c[(50 + 50 + 1)(20 + 25) + (30 + 50)]

= 0 + 1[(101 * 45 + 80)

= 4545 + 80

= 4625

Ex 4 - a[lb1 ... ub1, lb2 ... ub2]

ub1 - lb1 + 1 ub2 - lb2 + 1

a[4][3]

BA, C

nc (no of column)

Row Major order a[i][j] = BA + c[(ub2 - lb2 + 1)(i - lb1) + (j - lb2)]

Column Major order a[i][j] = BA + c[(ub1 - lb1 + 1)(j - lb2) + (i - lb1)]

(no of rows)

26-Mar

Column Major Order a[1...4][1...5]

a	1	2	3	4	5
1	a_{11}	a_{12}	a_{13}	a_{14}	a_{15}
2	a_{21}	a_{22}	a_{23}	a_{24}	a_{25}
3	a_{31}	a_{32}	a_{33}	a_{34}	a_{35}
4	a_{41}	a_{42}	a_{43}	a_{44}	a_{45}

BA = 1000 C = 10

00	02	04	06	08	10	12	14	16	18	20	22	24	26	28	30	32	34	36	38
a_{11}	a_{21}	a_{31}	a_{41}	a_{12}	a_{22}	a_{32}	a_{42}	a_{13}	a_{23}	a_{33}	a_{43}	a_{14}	a_{24}	a_{34}	a_{44}	a_{15}	a_{25}	a_{35}	a_{45}
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
Column 1				Column 2				Column 3				Column 4				Column 5			

Loc[a[4][3]] = BA + C[(3-1)*(no. of Rows + (4-1)]

= 1000 + 10(2*4 + 3)

= 1022

Loc a[2][5] = 1000 + 10[(5-1)*no of row + (2-1)]

= 1000 + 2[4*4 + 1]

= 1034

both are same (Row Major, Column Major)

if no order is mentioned, go for Row Major (by default)

Ex2- a[-75...125][80...150] BA = 0 C = 10 byte

CMO

Loc[a[15][130]] = 0 + 10[(130-80)*(125+75+1) + (15+19)]

= 10(50*201 + 90)

= 10(10050 + 90) = 100640

= 101400

$nr = ub_1 - lb_1 + 1$

Ex3 - $a[lb_1 \ldots ub_1][lb_2 \ldots ub_2]$ BA, C, CMo

$\Downarrow$

$nc = ub_2 - lb_2 + 1$

$Loc[a[i][j]] = BA + c[(j - lb_2) * nr + (i - lb_1)]$

3-D Arrays- collection of 2-D arrays

$a[3 \ldots 9][15 \ldots 35][8 \ldots 50]$

$9-3+1 = 7$ $\quad$ $35-15+1 = 21$ $\quad$ $50-8+1 = 43$

$\Rightarrow$ 7 table of size 21×43

$= 7$ [2-D array of size 21×43]

7 $\quad$ 21 no of rows $\quad$ 43 no of column

BA, Base Address = 1000 $\quad$ C = 10 $\quad$ RMO

$Loc[a[7][30][40]] = BA + c[(7-3)(nc * nr) + (30-15) * nc + (40-8)]$

$= 1000 + 10(4 * 21 * 43 + 15 * 43 * 32]$

$= 43890$

CMO

$Loc[a[7][30][40]] = BA + 10[(7-3) * nc * nr * (40-8) * nr + (30-15)]$

$= BA + 10$

$= 1000 + 10(4 * 21 * 43 + 8 * 21 * 15)$

$= 1000 + 10(3612 + 168 + 15)$

$= 1000 + 10(3795$

$= 38950$

LBE

Ex a[lb1 ub2 l a[lb1 . ub1, lb2 ... ub2, lb3 .... ub3]

BA, c

nr = ub2 - lb2 + 1

nc = ub3 - lb3 + 1

no of 2D = ub1 - lb1 + 1

(R, M, O)

loc [a[i][j][k] = BA + c [(i - lb1) * nc * nr + (j - lb2) * nc * (k - l

CMO

loc [a[i][j][k] = BA + c [(i - lb1 * nc * nr + (k - lb3) * nr + (j - lb

Lower Triangular Matrix -

	1	2	3	4
1	a11	a12	a13	a14
2	a21	a22	a23	a24
3	a31	a32	a33	a34
4	a41	a42	a43	a44

=

a11	0	0	0
a21	a22	0	0
a31	a32	a33	0
a41	a42	a43	a44

↓ 16 element

6 = zero

(n/2) approx - zero

↓ to save spa

Why we should store zero

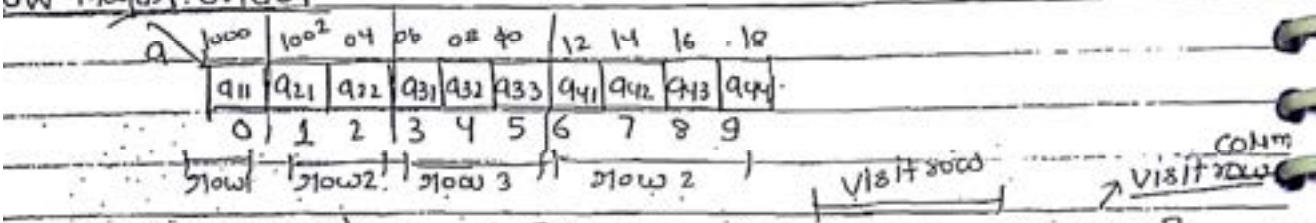
aij belong to upper part if {j > i}) return 0 at that time

Here store in row Major order -

row - 1 will have 1 element -

row - 2 will have 2 element

ow Major Order-



- Loc(a[4][3]) = BA + C [(4+3+2+1) - (4-1) (3+2+1) + (3-1)]

(sum of 3 natural no. — (4-1) nat no.)

= BA + C [sum of 3 natural no + (3-1)]

= 1000 + 2 [6 + 2] = 1016

[All triangular Matrix are square Matrix]

a[25...85, 25...85] (starting row no & colum no is also same)

BA = 1000

C = 1 Byte RMO LTM (Lower Triangular Matrix)

Loc [a[60][55]] = BA + C [sum of (60-25) natural no + (55-25)]

= ~~1000 + (60-25+1)~~

= BA + C [sum of (60-25) natural no + (55-25)]

= 1000 + C [(60-25)(60-25+1) / 2 + 30]

= 7600

formula

Loc [a[lb1...ub1, lb2...ub2] = BA + C RMO, LTM

{ Loc [a[i][j]] = BA + C [(i-lb1)(i-lb1+1) / 2 + (j-lb2)] }

a[4][3] = BA + c [sum of 3 natural no + (total rows - (3-1)]

= 1000 + 2 [6 + 2]

= 1016.

• a[25...85, 25, 85] BA = 1000 c = 10 Byte

CMO LTM

Rows = 61 column =

Loc a[60, 55] = BA + c [sum of (55-25) natural no + [no of rows - (55-1)

= 1000 + 10 [35×18 + 61 - [55-25-1)]

= 1000 + 10 [630 ... + 27

= 7570.

CMo a[i][j] = BA + c [sum of (j-lb1) no + (no of rows - (j-lb

natural

Extra

~~Fibonacci heap~~

BA

$Loc[a[4][3]) = 1000 + \left[\frac{4\times5}{2} - \frac{2\times3}{2} + (4-3)\right] * 2$

- $\frac{4\times5}{2}$ → crossing all column
- $\frac{2\times3}{2}$ → crossing unnecessary column → 3 row and 3 column
- $(4-3)$ → Visiting 4 thru

$= 1000 + 16 = 1016$

$Loc(a[4][1]) = BA + c\left[\text{sum of 4 nat. no} - \text{sum of } (4+1+1) \text{ no} + (4-1)\right]$

- sum of 4 nat. no ⇓ crossing all column
- sum of (4+1+1) no ⇓ crossing unnecessary crossed column
- (4−1) ⇓ Visiting row

it is used to cross all column

$= 1000 + 2[10 - 10 + 3]$

$= 1006$

Ex 2 - A[25.... 100, 25, 100] BA = 1000 c = 10B LTM CMO

$Loc(A[80][45]) = BA + c\left[\text{sum of } \overset{76}{\text{76 natural no}} - \text{sum of } (100-45+1) + (80-45)\right]$

$= 1000 + 10\left[\frac{76\times77}{2} + 31\times\overset{16}{32} + 10\right]$ ⇓ Visiting 80th row

$= 1000 + 10[38\times77 + 31\times16 + 10]$

$= 1000 + 10[2926 + \ldots$

$= 1000 + 10\left[\frac{76\times77}{2} + \frac{56\times57}{2} + 35\right]$

$= 151469$

5
38
77
266
2926

31
16
496

When you are at 15 col you are at 15 row.

101 101

Loc [A[5]

A[-50 .... 50, -50 ... +50] BA = 0 C = 1 Byte LRM, CM

Loc(A[25][15]) = 0 + 1 [sum of 101 nat no - sum of [50-15+ nat no

+ (25-15)]

= 101 x 102 / 2 + 10 - 36 x 37 / 2

101 / 51 / 101 / 505 / 51

= 5151 - 666

= 4485

↑ since you at 15 row (as 15 column will start. for 15 row as all element before 15 are: 2x10

Ex: A[lb1 ... ub1][lb2 ... ub2] BA, C, LTM, CMD

Loc[A[i][j]] = BA + C [sum of (ub2 - lb2 + 1) nat no - sum of (ub2 nat no

+ (i - j)]

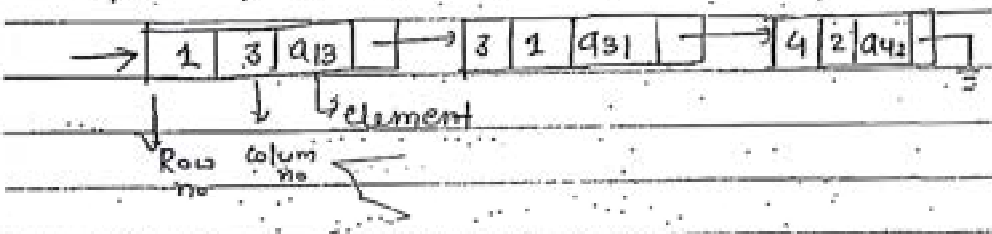
sum of all column

Sparse Matrix - A very less element in array are 1 and other are zero & there is no relation in them

$$\begin{bmatrix} 0 & 0 & a_{13} & 0 \\ 0 & 0 & 0 & 0 \\ a_{31} & 0 & 0 & 0 \\ 0 & a_{42} & 0 & 0 \end{bmatrix}$$

⇒ so need to store in array.

So we used linked list for its storage because if you use array lot of space is required.



Upper triangular Matrix :

Z Matrix - all matrix except Z are zero

to exec... | ...one...

to

Programming

Scope of a variable

(i) Static Scoping
(ii) Dynamic Scoping

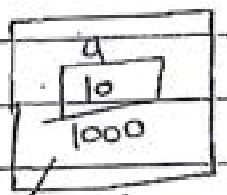
(data type followed by name of variab... declarat...

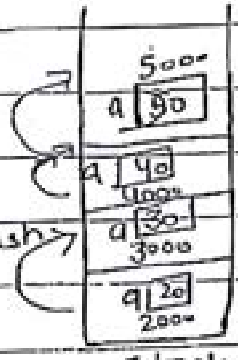
```
int a = 10;
main()
{
    int a = 20;
    B();
}
B()
{
    int a = 30;
    D();
}
C()
{
    int a = 40;
    D();
}
D()
{
    int a = 50;
    pf(a);
}
```

for all fn call, a single stack is u...

(both static & dynamic as no scope problem)

M/M ← stack (for local f^n variable)
← static (at time of compilation)
← heap (dynamic m/m)





O/P = 50

↓as stack will call top most value

Whenever a function is... memory to local vari... is created & its runn... completed, m/m is ... deactivated but ... global variable, m... created at time ... compilation. the ... m/m is created i... static area)

the space taken by ... an any f^n in stack ... is equal to local ... variable

static by default - C

if both local & global data are not there, no mean (variable)

free variable - if a fn is using a variable which is not declared in it.

as if D()
{
p f(a);
}

no a is declared but used

(A variable which is not initialized cannot be a free variable)

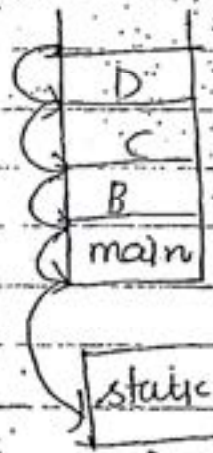
then
O/P = 10 (static)
40 (dynamic)

Whenever there is a free variable, compiler will identify it at time of compilation so, compiler will initialize it by global variable & this problem is known as scope problem & this problem if resolved at time of compilation, known as static scoping because at that time only global variable was initialized

i.e. compiler can't solve problem at time of compilation

but if compiler has restricted to solve scope problem at the time of compilation that it is not initialized by global data it will be resolved at run time if 'a' in not in D then go to its calling function & use from there and so on. if no calling function has a, then use global variable 'a'.

- Dynamic Scope will generate closer result as selecting the value from the closer fn



- Dynamic is difficult to implement

easier

static scoping - problem solve at compile time

dynamic scoping at run time

↓ difficult

because In dynamic scoping
Worst case - all fn
to be poped to che

static scoping - by default

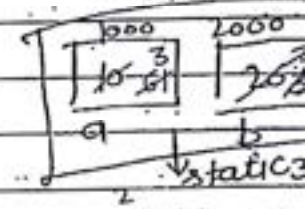
C, C++,

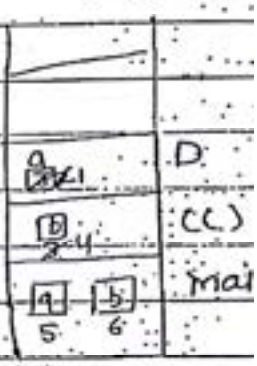
static variable get m/m first & last deallocated at l ie after ... ic

If a fn as D()
{ pf(a) and so on then static scoping - 10
dynamic - 40

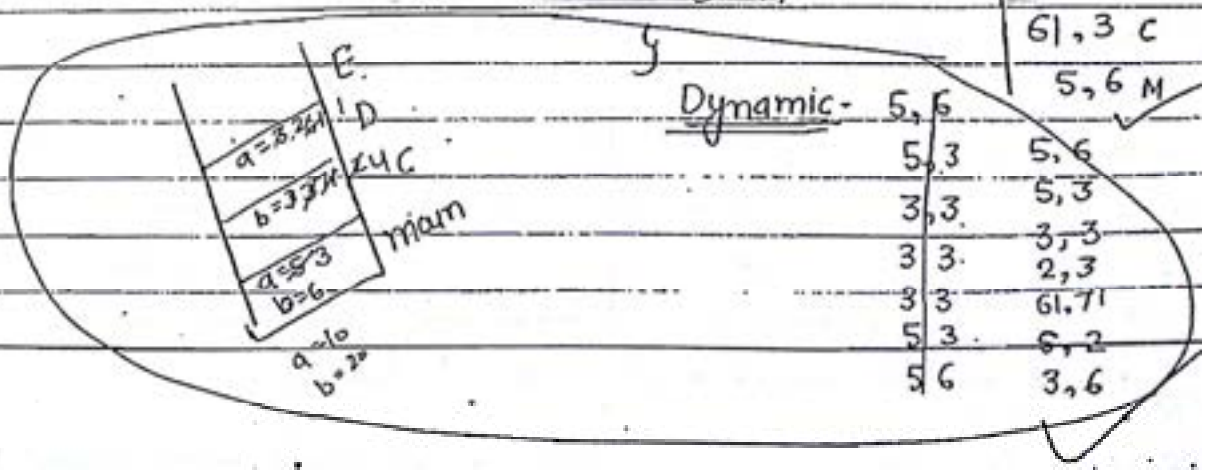
change done at compile time - static change
Change done at run time - dynamic change (in dynamic change are m to previous

Ex2
```
int a=10, b=20;
main()
{
  int a=5, b=6;
  pf(a,b);
  C();
  pf(a,b);
}
C()
{ int b=3;
  pf(a,b);
  D(b);
  pf(a,b);
  a=3, b=4;
}
D(int a)
{
  pf(a,b);
  a=2;
  b=3;
  E();
  pf(a,b);
  a=1;
  b=2;
}
E()
{
  pf(a,b);
  a=61;
  b=71;
}
```

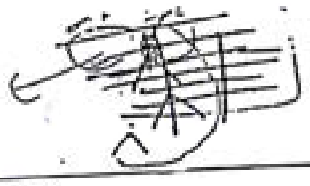




Static - 5,6 M
10,3 C
3,20 D
10,3 E
2,71 D
61,3 C
5,6 M



Dynamic - 5,6
5,3 5,6
3,3 5,3
3 3 3,3
3 3 2,3
3 3 61,71
5 3 6,2
5 6 3,6

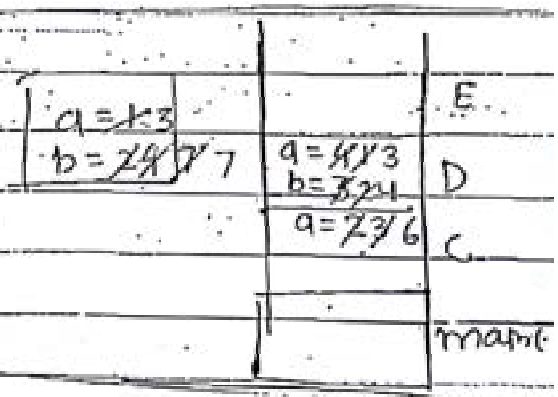


Q-3

```
int a=1, b=2;
main()
{
  fn(a,b);
  C(b);
  fn(a,b);
}
C(int a)
{
  a=3, b=4;
  fn(a,b);
  D(b,a)
  fn(a,b);
  a=6;
  b=7;
}
D(int a, int b)
{
  fn(a,b)
  a=1, b=2
  E();
  fn(a,b);
  a=3, b=4;
}
E()
{
  fn(a,b);
  a=3, b=7;
  fn(a,b);
```

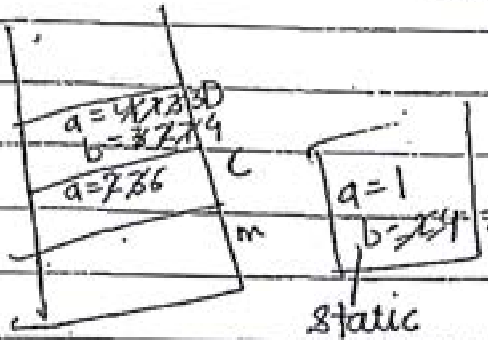
Static Scope -

1,2
3,4
4,3
1,4
3,7
1,2
3,7
3,7



Dynamic

1,2
3,4
4,3
1,2
3,7
3,7
3,4
1,7



Static Variable-

ex:
```
main()
{
    static int var = 5;
    printf("%d", var--);
    if(var--)
    {
        printf("%d", var);
        main();
    }
}
```

var = 5

Without static:
5 4
5 4
5 4
5 : → stack overflow
∞

[Infinite loop]

with sta
4
3
2
1
0

When local variable are declared with static; M/M will be created at compile time.

With static
main()
1. M/M already created
2. if(5)
   print(4)
   and so on
   no M/M created
   if(4)

8 4 var
Todo 3

• by default; global are static as m/m created at compile time.

o/p - 4 3 2 1 0
as M/M is created only once.

at end of program
var = -1

• M/M created i.e static is only one time as compilation is done one time only.

a = 3

b = a-- b = --a
pf(a,b) pf(a,b
(2,3) (2,2)
first assign and decrease afterwards

if(-10)
pf(hi) ⇒ Hi
else
pf(bye

if non zero it will prin
hi

if '\0' or Null o
if fa

local va
dynamic m/m → heap area

in char - '\0'
int - 0
float 0.0
pointer - Null