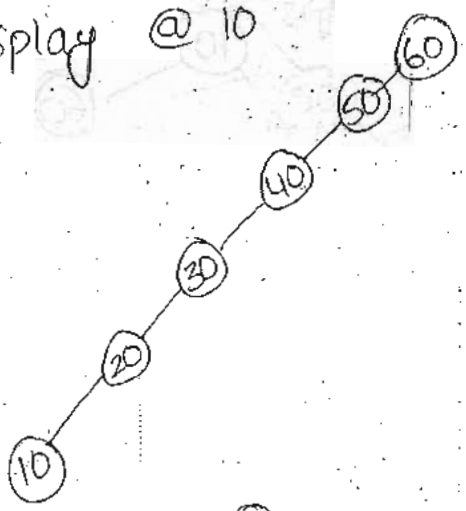
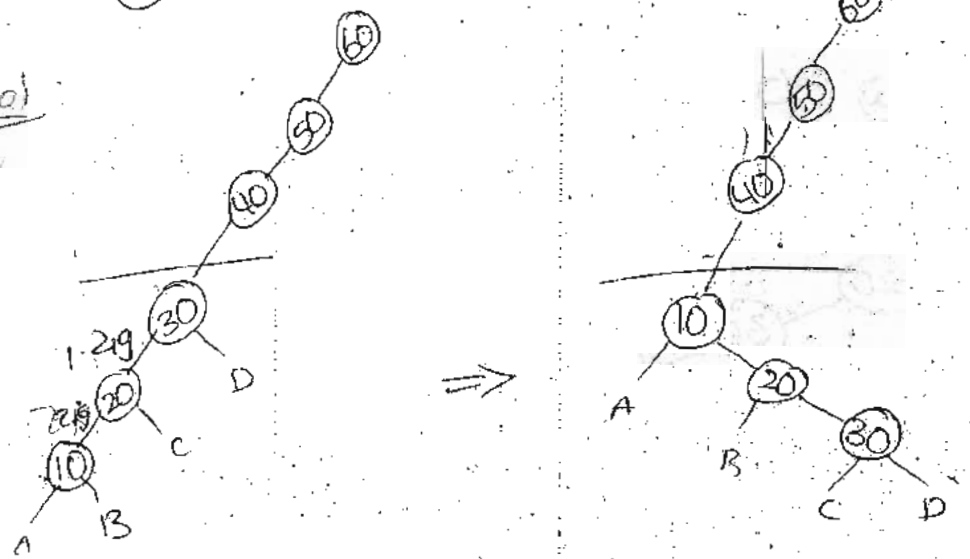
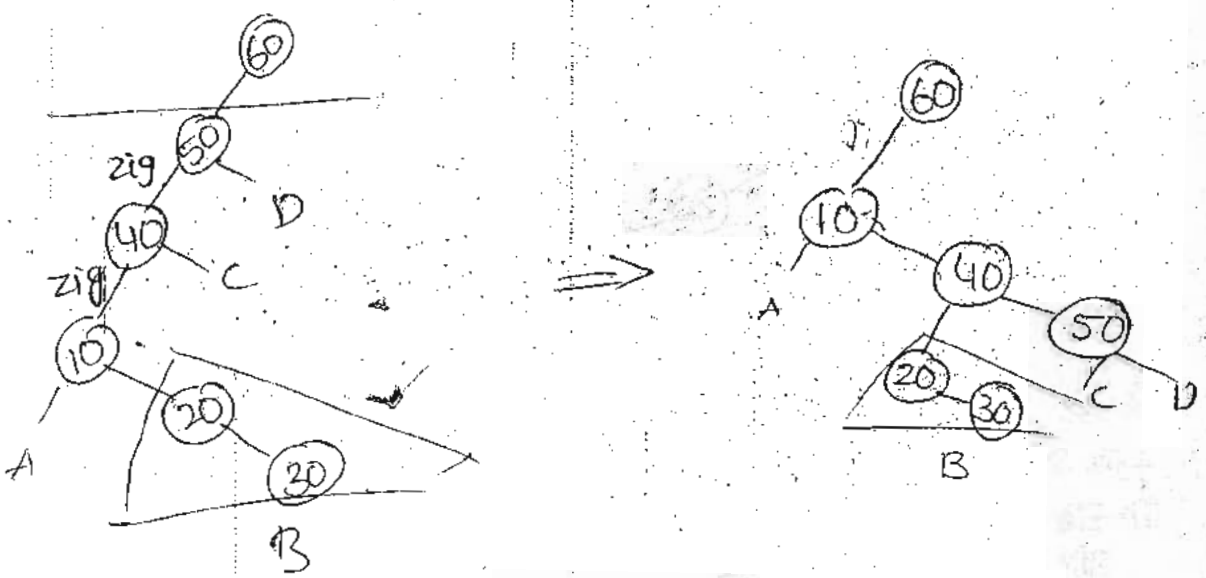
Note:
By one rotation, we can splay only two levels.

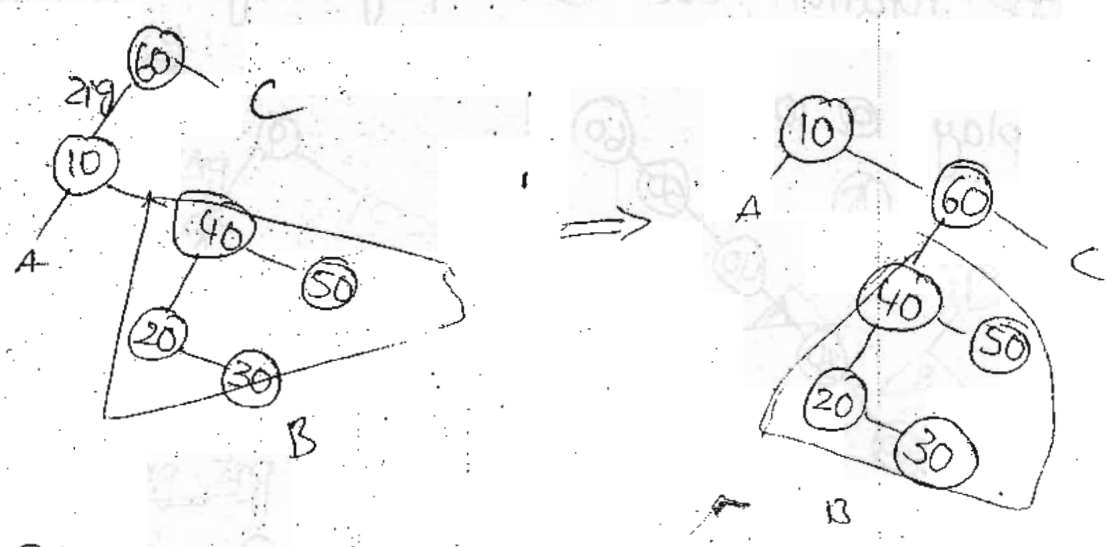
Eg: Splay @ 10



Sol

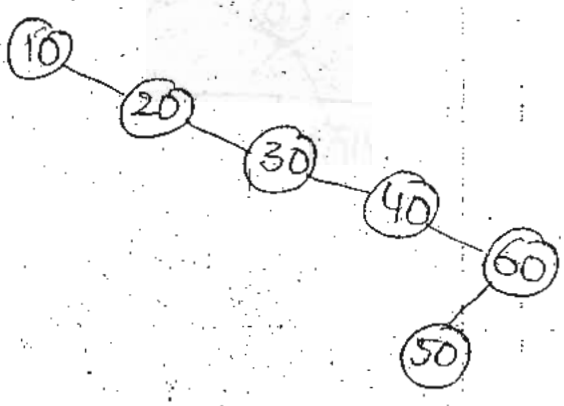




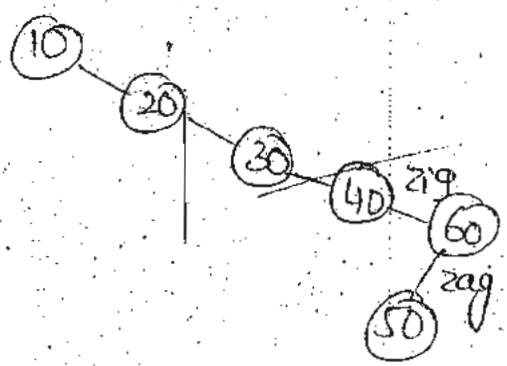


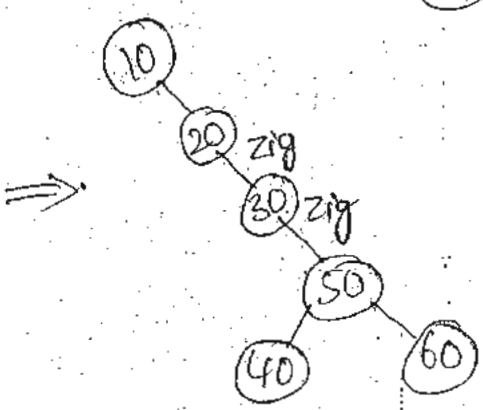
Eg:

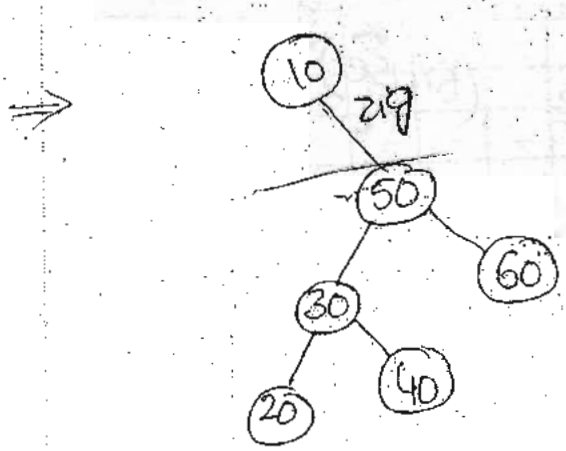
Splay @ 50

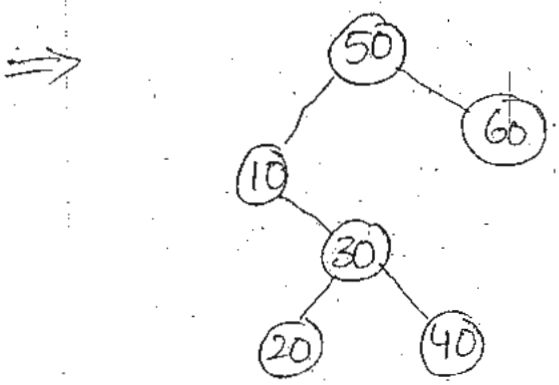


Sol:

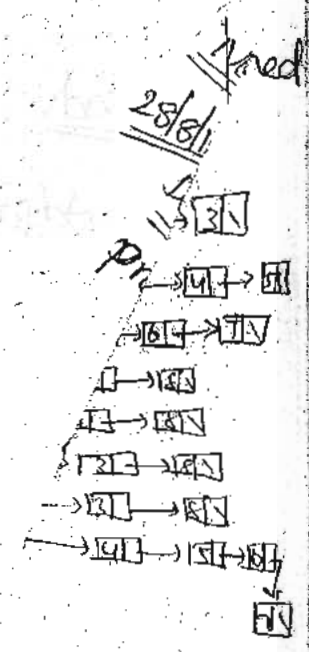








Note:
By default, splay trees are BST but not AVL Tree

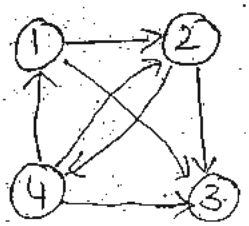


Graphs (Kruse)

$G = (V, E)$

V → vertices

E → edges



Representation in Memory:

I) Sets:

vertex	adjacency sets
1	{2,3}
2	{3,4}
3	∅
4	{1,2,3}

adv: pascal: supports sets as datatype

disadv: c/c++ does not support sets as datatype

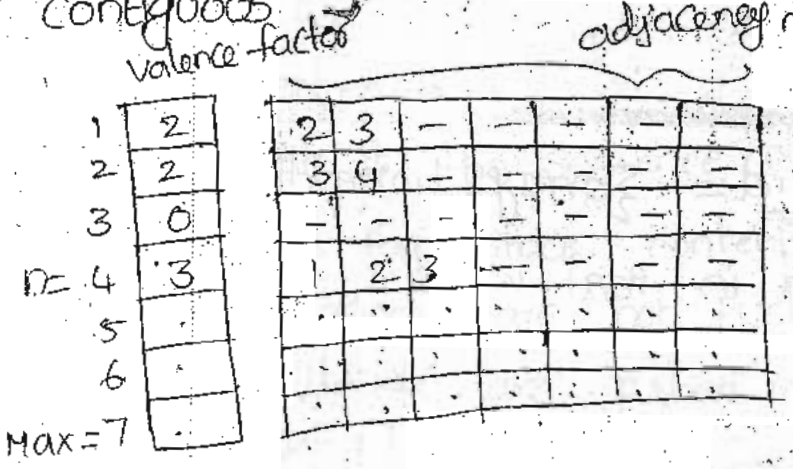
II) Adjacency Matrix (or) Table

	1	2	3	4
1	F	T	T	F
2	F	F	T	T
3	F	F	F	F
4	T	T	T	F

Adv: Array: simple datatype in all lang.

Disadv: static (can't add new vertices/edges)

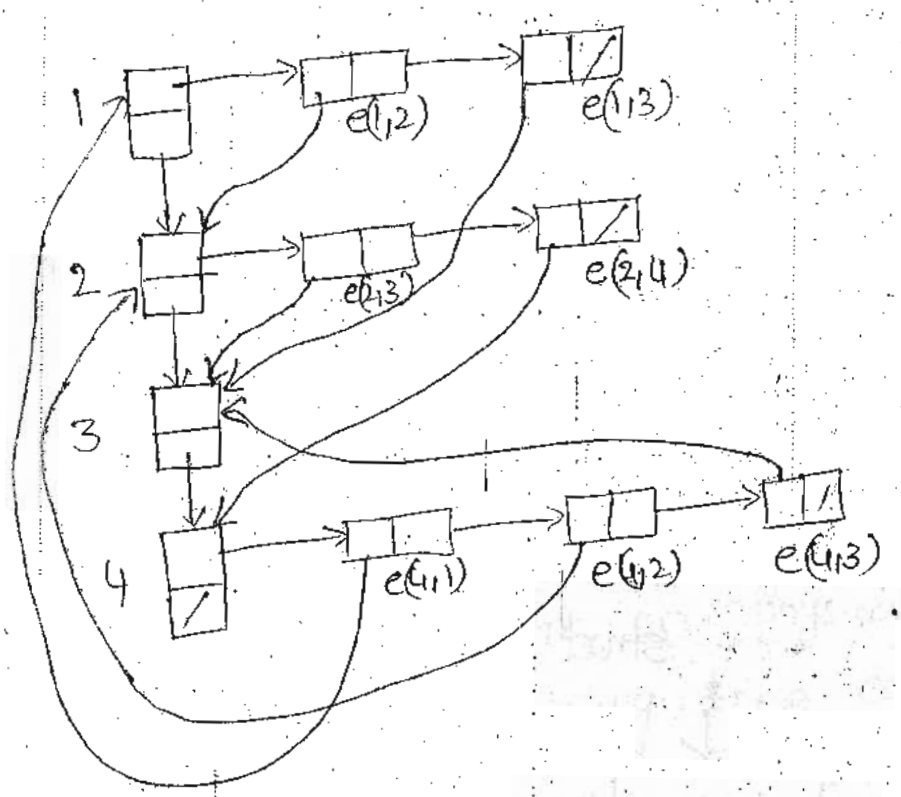
III) Contiguous



adv: limited dynamic

disadv: More space wastage

IV) Linked representation

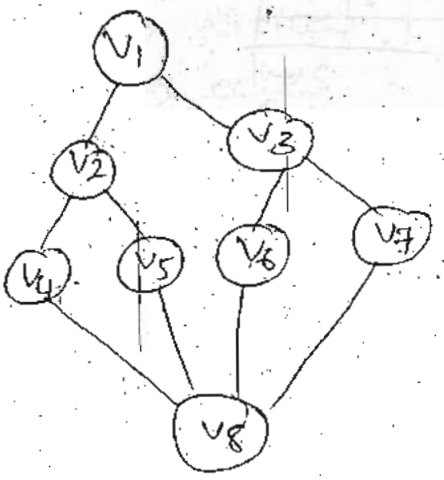


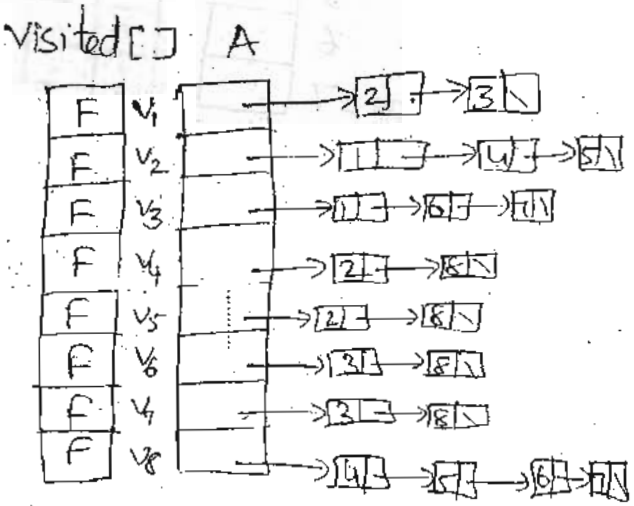
adv: dynamic

disadv: More space for links

IV) Hybrid = {array + linked lists}

- used in real world





→

```
enum bool {false, true}
Struct Graph
{
    int n; // No. of vertices
    bool visited[10]; // n ≤ 10
    struct Node *A[10];
};
```

Note: Graph is not self referential structure

28/8/11

Depth first Search (DFS):

```
procedure DFS(V)
    visited(V) = True
    for each vertex 'w', adjacent for 'v'
        if not visited(w)
            call DFS(w)
end
```

D) Traversal: DFS

① Starting vertex is V_1

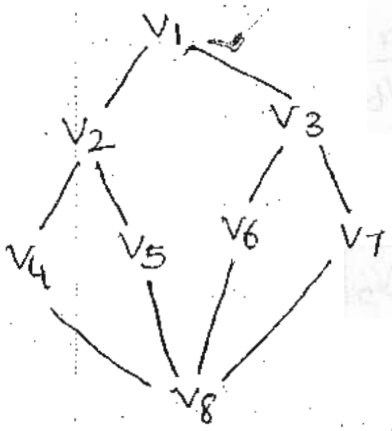
a) ~~calcul~~ what is the sequence of exploration

b) what is the stack contents

c) what vertices are not pushed in?

d) what vertices are pushed in for more than once

Sol:



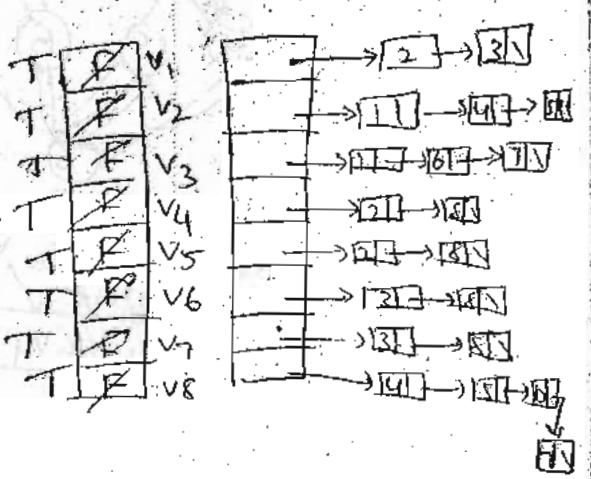
Its physical structure is hybrid structure

Given graph is a data structure. If we work on this datastructure, more than one pattern can be obtained.

But if we work on physical structure (memory) unique pattern is possible.

∴ so we need to work on physical structure so that unique DFS spanning tree can be obtained

V	W	stack contents
V_1	V_2	V_1
V_2	V_1	$V_1 \mid V_2$
	V_4	
V_4	V_2	$V_1 \mid V_2 \mid V_4$
	V_8	
V_8	V_4	$V_1 \mid V_2 \mid V_4 \mid V_8$
	V_5	

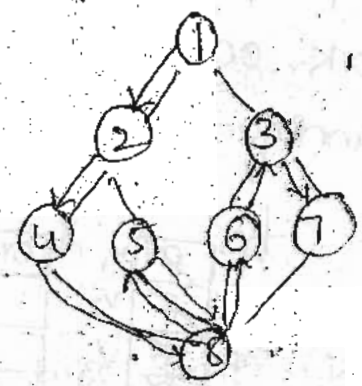


V	w	stack contents
v_5	v_2	
	v_8	v_1 v_2 v_4 v_8
	pop	
v_8	v_6	v_1 v_2 v_4 v_8
v_6	v_3	v_1 v_2 v_4 v_8 v_6
v_3	v_1	
	v_6	
	v_7	v_1 v_2 v_4 v_8 v_6 v_3
v_7		
	pop	
v_3:		

already all v_5 adjacents are explored.

Note:
when to push?
If a node has atleast one unexplored adjacent node

→



It is DS. But Remember we are working with physical structure

Sequence : v_1 v_2 v_4 v_8 v_5 v_8 v_6 v_3 v_7

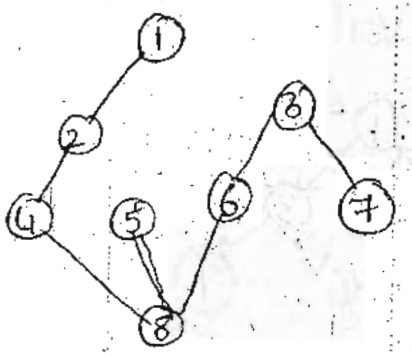
Stack : v_1 v_2 v_4 v_8 v_8 v_6 v_3

pop ← popped out

Not pushed in: v_5, v_7

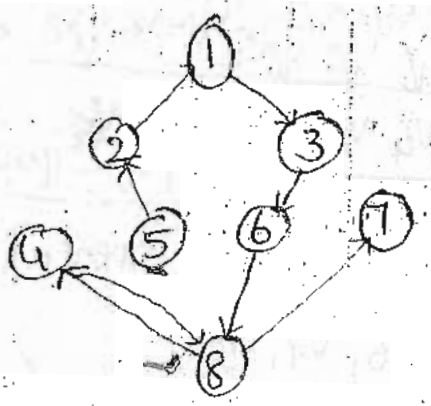
more than once: v_8

spanning Tree is



p) If the starting vertex is V_5 then repeat above a, b, c, d

Sol :



sequence: V_5 V_2 V_1 V_3 V_6 V_8 V_4 V_8 V_7

stack: V_5 V_2 V_1 V_3 V_6 V_8 V_8

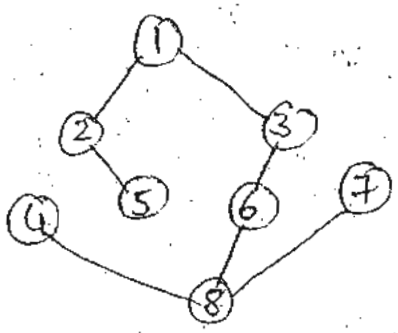
pop

popped out

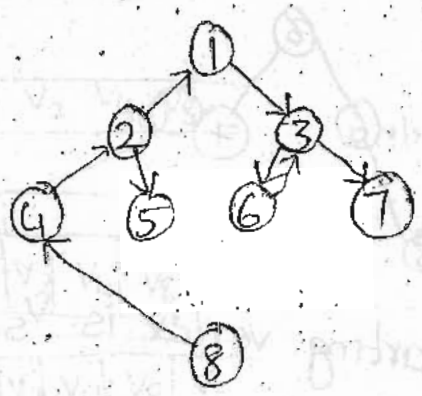
Not pushed in: V_4, V_7

pushed in more than once: V_8

∴ DFS spanning tree is



(b) If the starting vertex is V_8 then repeat a, b, c



Sequence: V_8 V_4 V_2 V_1 V_3 V_6 V_3 V_7 V_2 V_5

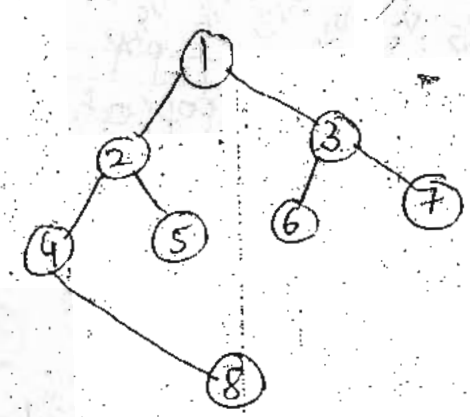
Stack: V_8 V_4 V_2 V_1 V_3 pop V_6 pop V_2

popped at

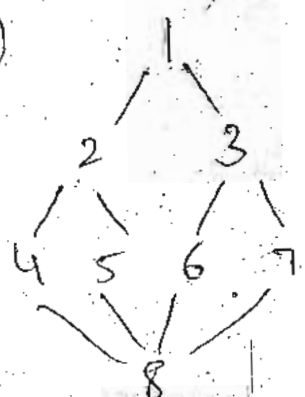
Not pushed in : V_6, V_7, V_5

more than once : V_3, V_2

∴ DFS spanning tree is



(c)



Traversal: DFS. If starting vertex is v_1, identify the valid DFS sequences

a) 1 3 6 8 7 5 2 4

b) 1 2 5 8 4 6 3 7

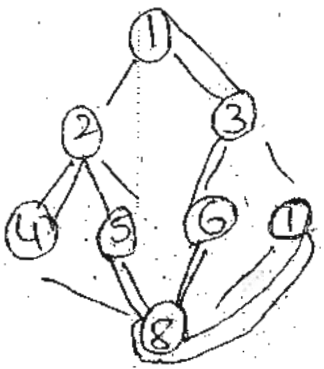
c) 1 2 4 8 7 3 6 5

d) 1 3 7 8 4 2 5 6

e) 1 2 4 5 8 6 7 3

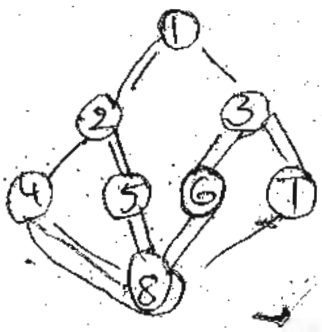
Sol: Here more than one sequence. so we need to work on Data structure

a)



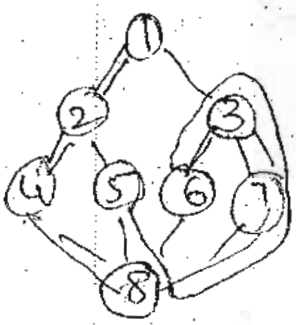
– valid

b)



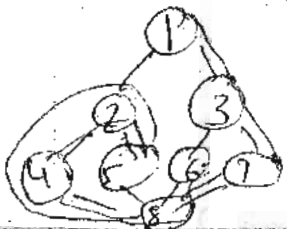
– valid

c)

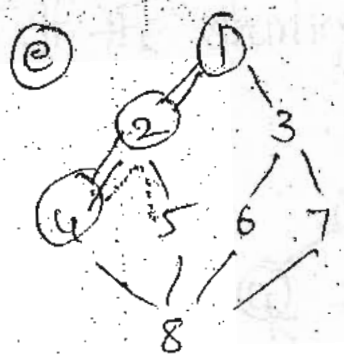


– valid

d)



– valid



5 → ∴ Invalid

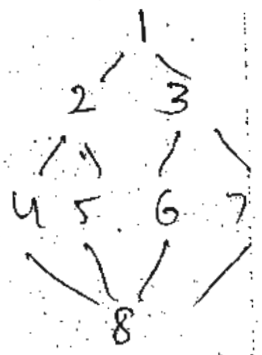
p) If the starting vertex is v_5 then identify invalid sequences

a) 5 2 4 8 6 1 3 — Invalid

b) 5 8 6 1 3 1 2 4 — Invalid

c) 5 2 4 8 7 3 6 1 — valid

d) 5 8 6 3 7 1 2 4 — valid



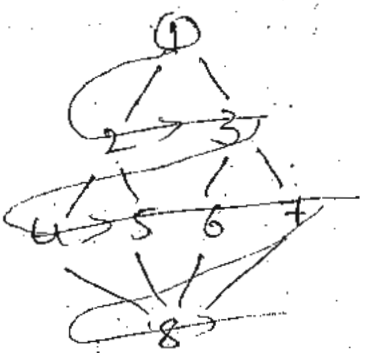
Breadth First Search (BFS):

```
procedure BFS (V)
    visited [V] = True;
    enqueue (V);
    while Queue is not empty
        V=dequeue( );
        For all vertices 'w', adjacent for 'V'
            if not visited(w)
```

```
    } visited(w) = True;
      enqueue (w)
  }
}
end
```

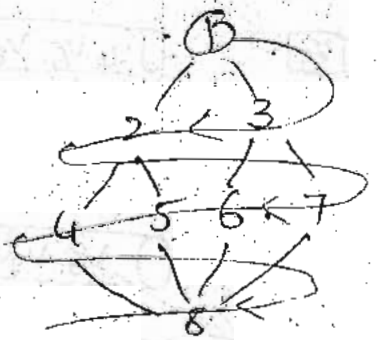
→ BFS

L-R BFS (left to right)



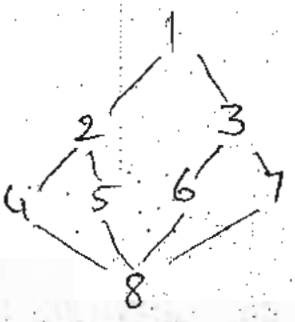
1, 2, 3, 4, 5, 6, 7, 8

R-L BFS (Right to left)



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

(P) Identify whether the above pseudo code is LR BFS (or) RL BFS

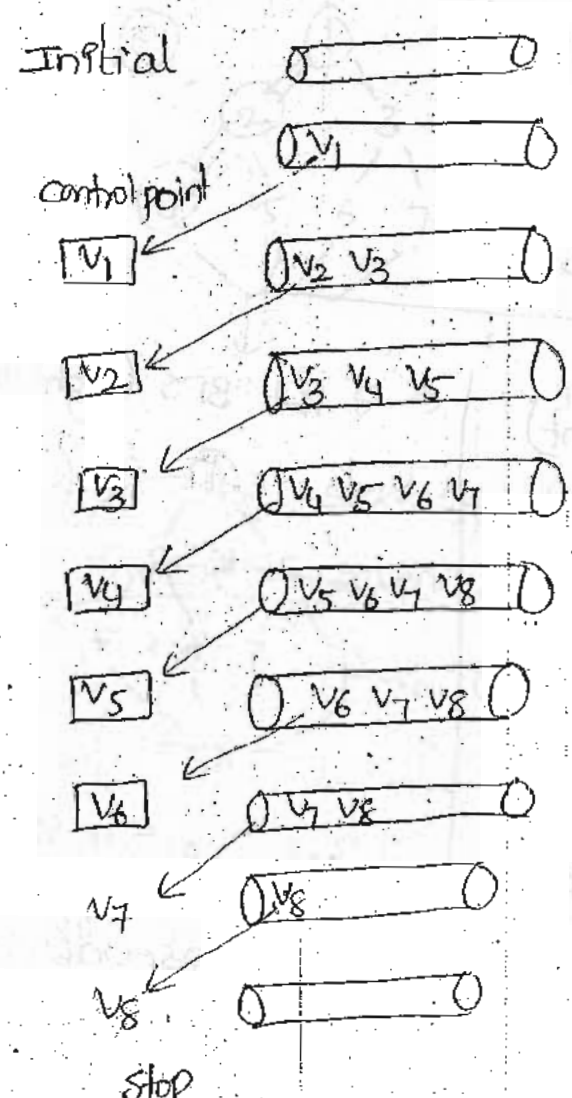


Starting vertex is V_1

Assume we are working on physical structures

Then we need to find the status of queue

Initially queue is empty



∴ No node is inserted more than once
every vertex is getting visited

p) what is the status of Queue after '4' is deled

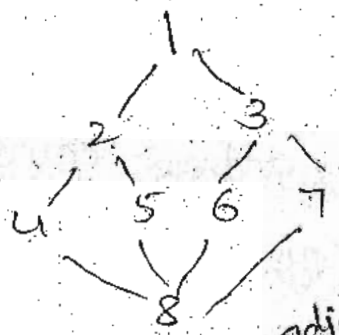
Sol: v_5 v_6 v_7 v_8

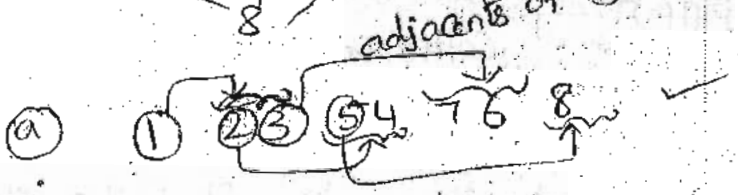
→ The above code does not distinguish L-R BFS (or) R-L BFS. It all depends on the way the graph is represented in physical memory
ie. physical structure

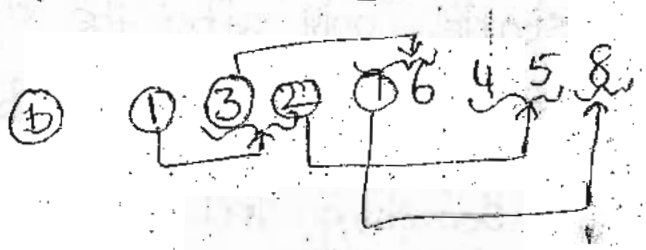
p) Traversal : BFS. If starting vertex is v_1
Identify valid/Invalid sequences.

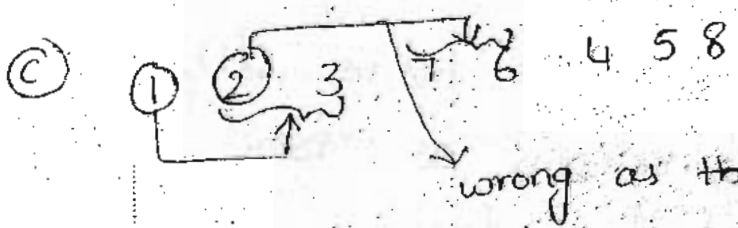
a) 1 2 3 5 4 7 6 8 - valid
b) 1 3 2 7 6 4 5 8 - valid
c) 1 2 3 7 6 4 5 8 - Invalid
d) 1 3 2 4 5 6 7 8 - Invalid

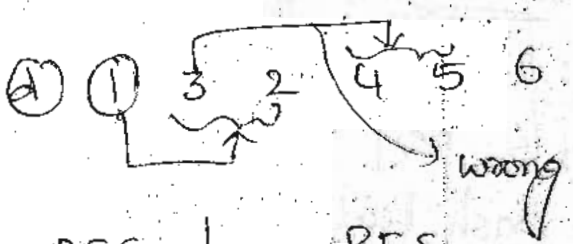
Sol: More than one sequence
Hence we need to work on Data structure graph











→

DFS	BFS
- Recursive	- Not Recursive (Iterative)
- stack (Implicitly)	- queue (explicitly)
- In algorithm Keyword - EACH	Keyword - ALL
- Similar to PREORDER	- similar to level order

- Certain vertices are not pushed in - more than once pushed - Termination: stack is empty	- All are pushed - only once - Queue is empty

Hashing

Definition:

Hash function – Key to address transformation

Perfect hash function (PHF) – 'n' keys Vs 'n' locations without collisions

minimal PHF – Keys > locations but collision starts only after the enter hash table is filled up

Goal:

- Hashing is a searching technique
- Search time is to be O(1) but not fulfilled. Still it remains as vision

Types of hash functions:

① Division method:

$H(x) = x \cdot 1 \cdot m$, x is key

where m = hash table size

Knuth, m is prime number. It is a suggestion

Eg: m=5

$H(x) = x \cdot 1 \cdot 5$

Key	69	77	100	83	16	18
loc	4	2	0	3	1	(3)

Collision

0	100
1	16
2	97
3	83
4	69

p) which of following is suitable hash function for storing the keys in the range 1 to 1000.

(a) $H(x) = x \bmod 1000$ → 0 − 999

(b) $H(x) = (x+1) \bmod 1000$ → 0 − 999

(c) $H(x) = x \bmod (1000+1)$

1001 → 0 − 1000

(d) $H(x) = (x \bmod 1000) + 1$

$(0-999) + 1$ → 1 − 1000

(2) Folding method:

fold shifting method

Key → loc

123 214 211 → ?

123
214
211
548

000
548 Key
999

Fold Boundary Method

Key → loc

123 214 211 → ?

123
211
334

000
334 Key
999

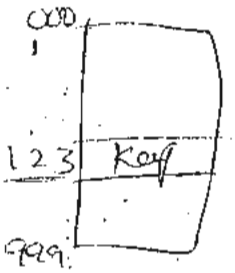
③ Mid square Method:

Key → loc

12345 → ?

$(12345)^2 = 152399025$

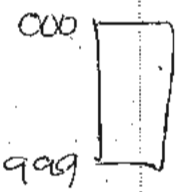
Suppose 123 Key



④ Truncation method:

Key → loc

123456	456
784967	967
314556	456 ✗
256456	456 ✗



– ④ is fast
But more collisions

30/8/11

Collision Resolution Techniques.

- open addressing (or) closed hashing
  - linear Probing → Primary clustering
  - Quadratic Probing → Secondary clustering
  - Random probing → Random clustering
  - Double hashing / Rehashing → clustering is reduced but not avoided
- separate chaining

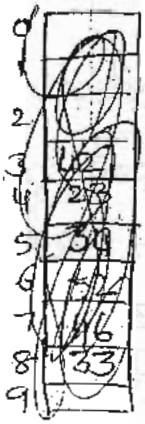
Disadvantages

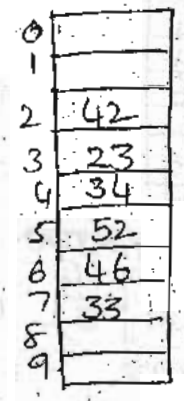
b) which of the following choices gives a possible order in which the key values could have been inserted for a hash function A(K) = K·1.10 with linear probing

(a) 46, 42, 34, 52, 23, 33 (b) 34, 42, 23, 52, 33, 46

(c) 46, 34, 42, 23, 52, 33 (d) 42, 46, 33, 23, 34, 52

Given hash table

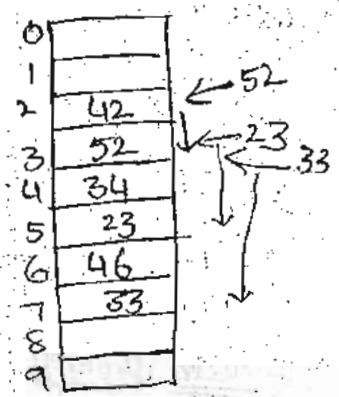




Sol:

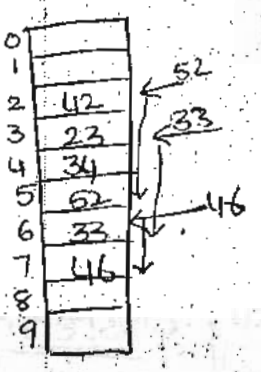
(a)

Key	46	42	34	52	23	33
loc	6	2	4	2	3	3



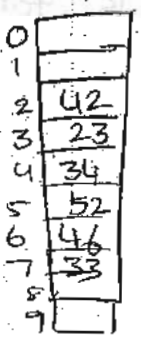
(b)

Key	34	42	23	52	33	46
loc	4	2	3	2	3	6



(c)

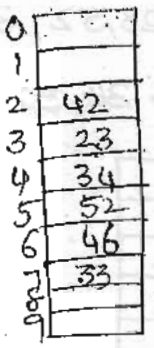
Key	46	34	42	23	52	33
loc	6	4	2	3	2	3



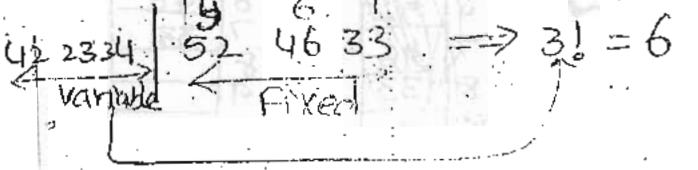
(d) How many different insertions sequences of the key values using the same function and probing will result in the hash table shown above

(a) 10 (b) 20 (c) 30 (d) 40

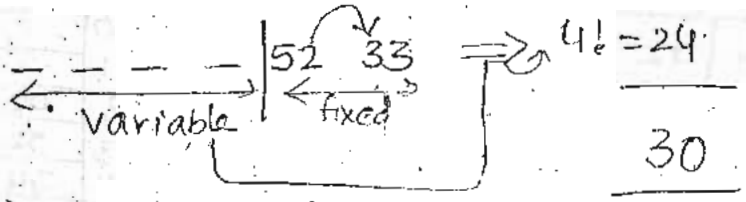
Sol:



Case(i): TO store 52



Case(ii): TO store 33



30

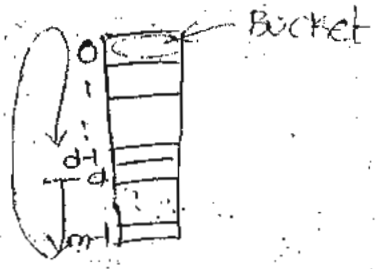
if: 42 23 34 | 52 33

(46)

Linear probing:

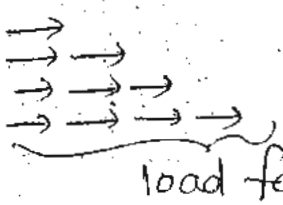
Sequence: $d, d+1, d+2, \ldots m-1, 0, 1, \ldots, d-1$

H(x): d (say)

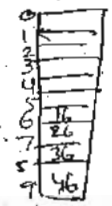


Primary clustering:

The trend is for long sequence of preoccupied positions still becomes longer, primarily at one place



Eg: 16: 6,
26: 6, 7,
36: 6, 7, 8,
46: 6, 7, 8, 9



Quadratic probing:

$h+p^2$

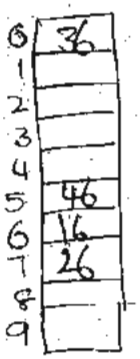
$p = 1, 2, 3, \ldots$

disadvantage: Secondary clustering

The trend in for long sequence of preoccupied positions still become longer, primarily at different places

Eg:

Key	16	26	36	46
loc $H(x) = x \% 10$	6	6	6	6



$H(16) = 6$

$H(26) = 6$

collision

say $h = 6$

$p = 1 \Rightarrow h+1^2 = 6+1^2 = 7 \% 10 = 7$

$H(36) = 6$

collision

say $h = 6$

$p = 1 \Rightarrow h+1^2 = 6+1^2 = 7 \% 10 = 7$

collision

$p = 2 \Rightarrow h+2^2 = 6+4 = 10 \% 10 = 0$

$H(46) = 6$

collision

say $h = 6$

$p = 1 \Rightarrow h+1^2 = 6+1^2 = 7$

$p = 2 \Rightarrow h+2^2 = 6+2^2 = 0$

$p = 3 \Rightarrow h+3^2 = 6+9 = 15 \% 10 = 5$

→ In quadratic probing all the buckets are not compared because of quadratic nature. so Linear probing is better

Random probing:

RNG: Random Number Generator

↓

No number should be repeated within random space

$Y_{new} = Y_{old} + c$

Y_{new} = seed value

c = constant

m = Hash table

Let $Y_{new} = 9$

$c = 4$

$m = 11$

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

$H(x) = x \% 11$

Key	17	28	39	50
loc	6	6	6	6

0	
1	
2	
3	39
4	
5	
6	17
7	50
8	
9	
10	28

17: 6
28: 6, 10
39: 6, 10, 3
50: 6, 10, 3, 7

Goal: change 'c', get new sequence. This is applied on double hashing

Double hashing / Rehashing

$H_1(x) = x \% m$ (given)

If collision

$Y = Y + c \bmod m$

you need 'c'

$H_2(x) = [x \bmod (m-2)] + 2$ (given)

Eg:

Key	17	28	39	50
10 c	6	6	6	6

$H_1(17) = 6$

$H_1(x) = x \% 11$

$H_2(x) = (x \% 9) + 2$

$H_1(28) = 6 = Y_{old}$

collision

we need c

$c = H_2(x)$

$H_2(28) = (28 \% 9) + 2$

$= 1 + 2 = 3$

$\therefore c = 3$

$Y_{new} = Y_{old} + c$

$= 6 + 3$

$= 9 \% 11 = 9$

$H_1(39) = 6 = Y_{old}$

collision

we need c

$c = H_2(39)$

$H_2(39) = (39 \% 9) + 2 = 5$

$Y_{new} = Y_{old} + c$

$= 6 + 5 = 11 \% 11 = 0$

0	39
1	
2	50
3	
4	
5	
6	17
7	
8	
9	28
10	

17 : 6

28 : 6, 9

39 : 6, 0

50 : 6, 2

clustering is reduced

$H_1(50) = 6 = Yold$

we need $C = H_2(x) = H_2(50)$

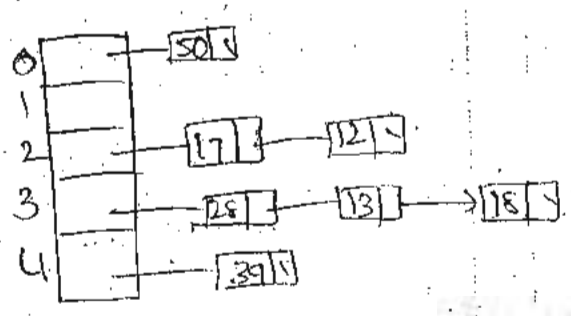
$H_2(50) = (50 \% 9) + 2 = 7$

$Ynew = 6 + 7 = 13 \% 11 = 2$

chaining:

Key	17	28	39	50	12	13	18
loc	2	3	4	0	2	3	3

$H(x) = x \% 5$



open addressing	chaining
disadvantage:	advantages
→ colliended records require more probes	→ less probes
→ Deletion is not possible.	→ possible
→ overflow problem	→ NO overflow problem