UT-II

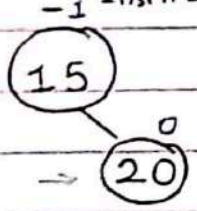
Hrs: 20m. DSA - Question Bank

1) Create AVL Tree (sequential sel) Show all steps and rotations
15, 20, 24, 10, 13, 7, 30, 35, 25

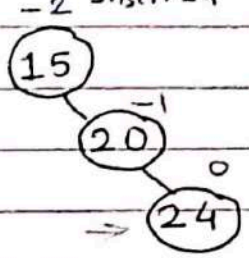
→ AVL Tree is called height-balanced tree was defined by mathematicians (Adelson, Velskii, Landis)
It is a self balancing Binary Search Tree (BST) where difference btw (height of left and Right subtree) is either 0, -1, 1.

Step1: Insert 15


Step2: Insert 20

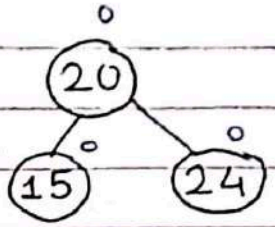



Step3: Insert 24

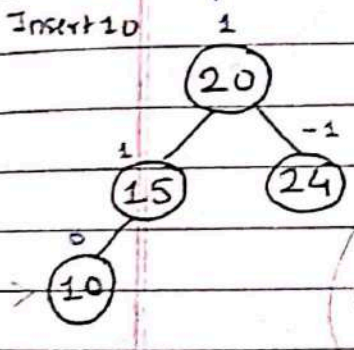



Step4:

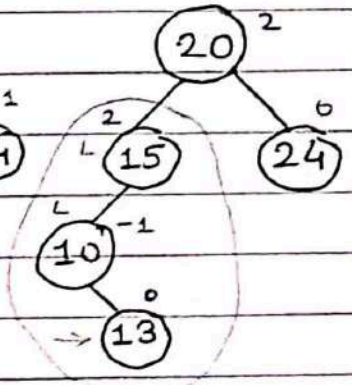
RR rotation anti ↻




Step5: Insert 10



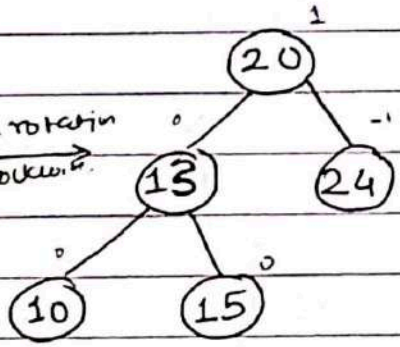

Step6: Insert 13



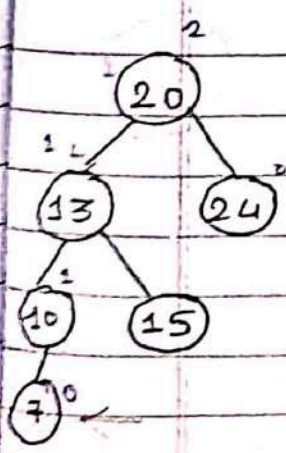

Imbalance

Step7:

LL rotation clockwise ↻

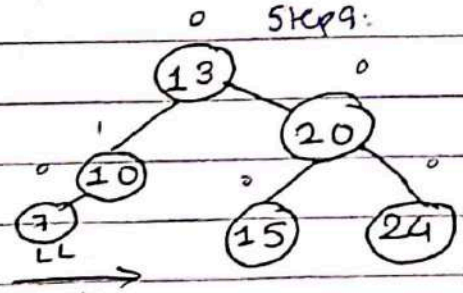



Step8: Insert 7:

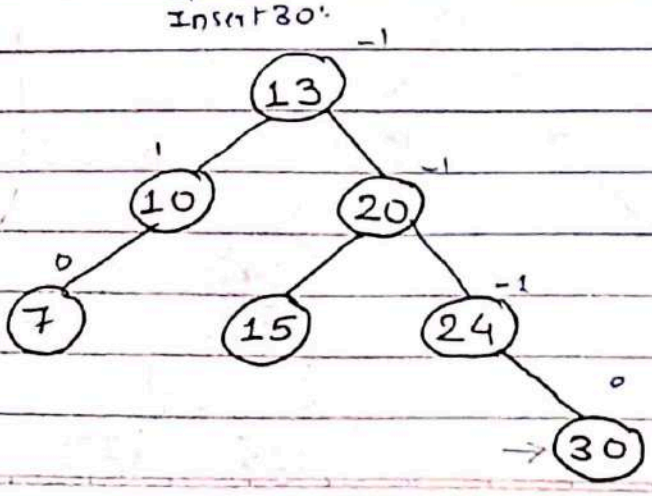



Step9:

LL rotation ↻

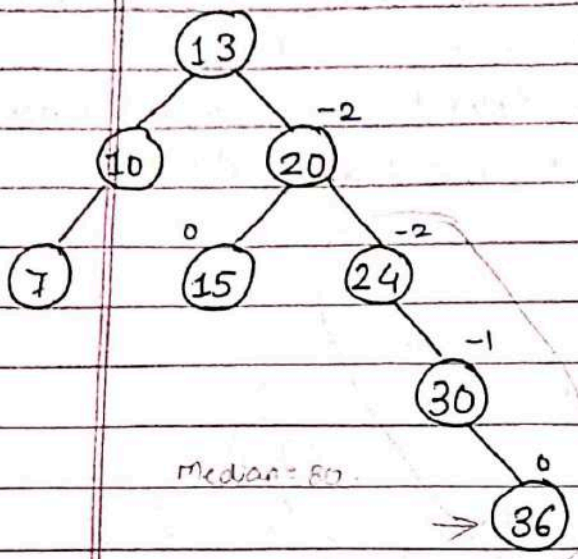



Step10: Insert 30:

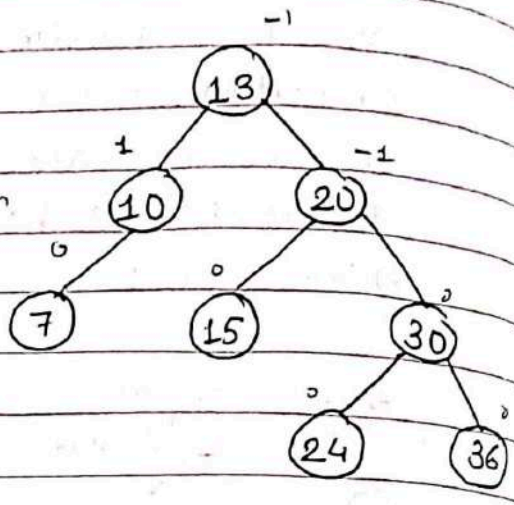


PAGE No
DATE / /

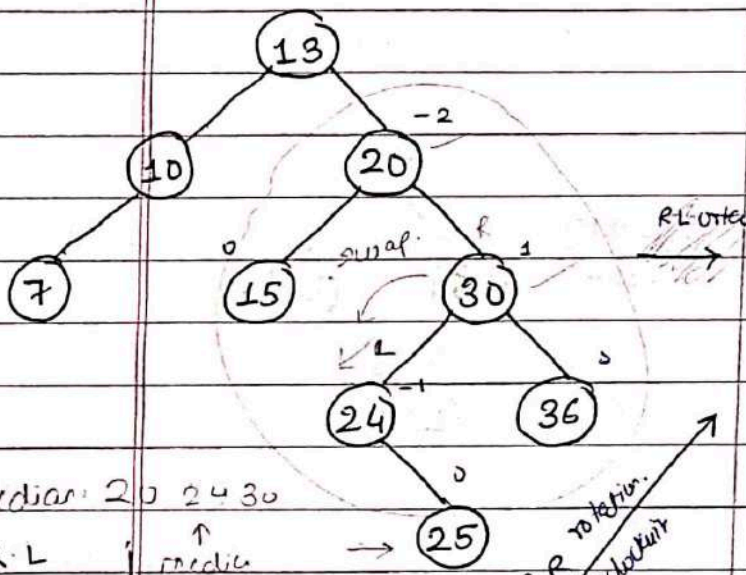
Step 11:
Insert 36.



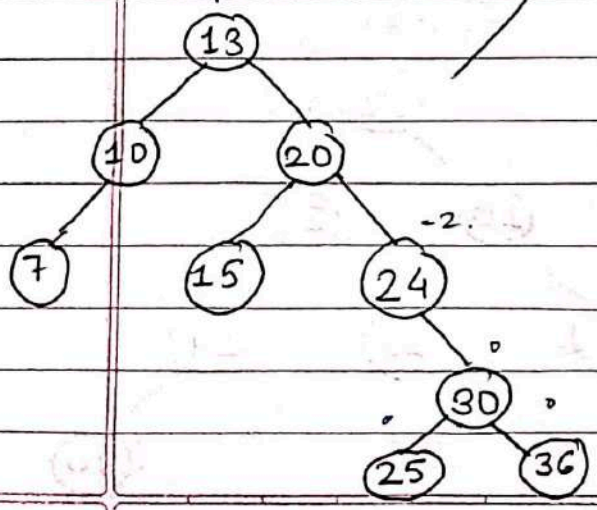

Step 12:



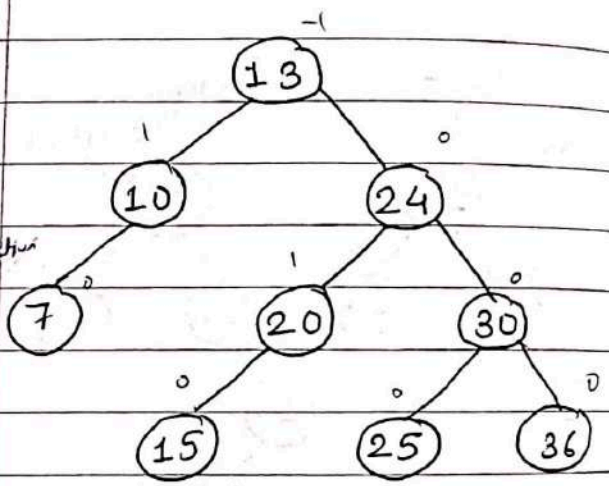

Step 13:
Insert 25.




Step 14:




Step 15:



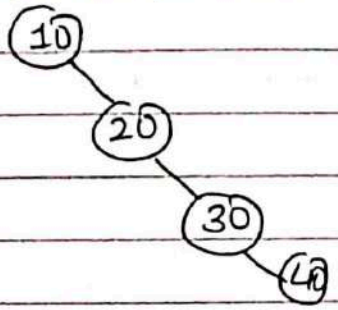

Final Ans.

2) AVL Tree differ from BST? Show result of inserting 15, 19, 22, 10, 3, 37, 25, 12, 13 one at a time into an initially empty AVL Tree.

→ Binary Search is a tree data structure that follows the condition of the binary tree. That is, Each node in a binary search tree should have the utmost two child nodes. It arranges its node elements in sorted manner. A Binary tree is referred as a binary search tree if for any node n in the tree:

1. The node elements in the left subtree of n are lesser in value than n.
2. The node elements in the right subtree of n are greater than or equal to n.

Consider following example: 10 20 30 40



Right Skewed BST.

It is observed that BST's worst case performance is closest to linear search algorithm / Linked List, that is O(n). In real time data, we cannot predict data pattern and their frequencies. So a need arises to balance BST.

Therefore we use AVL Tree.

- AVL Tree is a height balancing Tree or self balancing Tree binary search tree where difference between height of Left Subtree and height of Right Subtree is either 0, -1 or 1. This difference is called as Balance Factor (BF)
- In AVL Tree the height of tree is O(logn) n = no. of nodes. Searching is efficient in AVL Tree when there are large number of nodes in tree because tree's height is balanced
- To Balance the AVL Tree we perform some rotations:

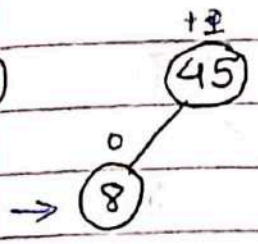
1) LL 2) RR 3) LR 4) RL.

Step 11

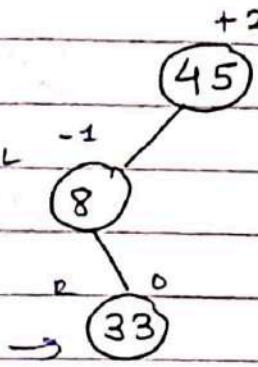
3) AVL Tree 45 8 33 85 61 10 48 76 57 99

Step 1: Insert 45

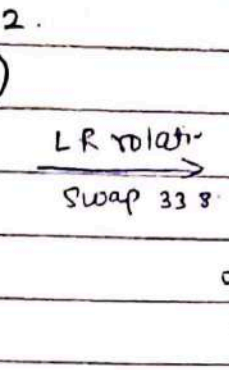

Step 2: Insert 8



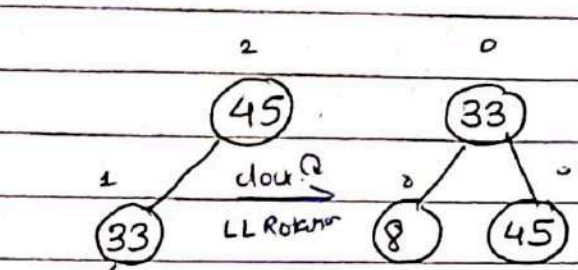

Step 3: Insert 33



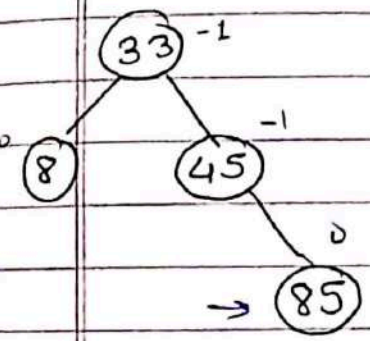

Step 4:



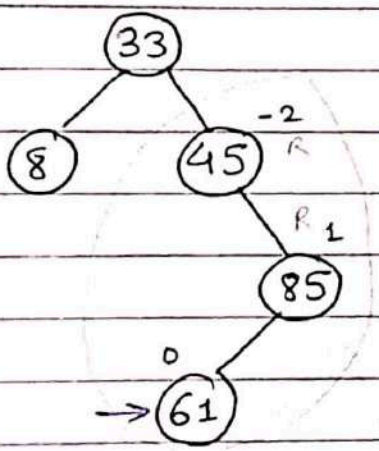

Step 5:




Step 6: Insert 85

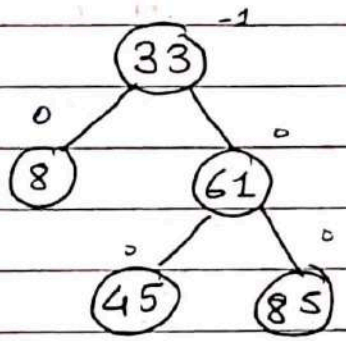



Step 7: Insert 61

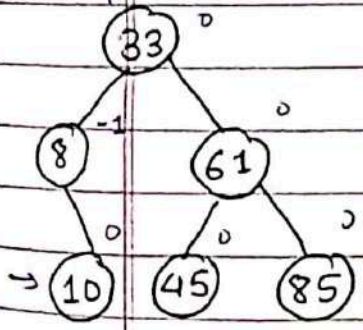



RR rotation

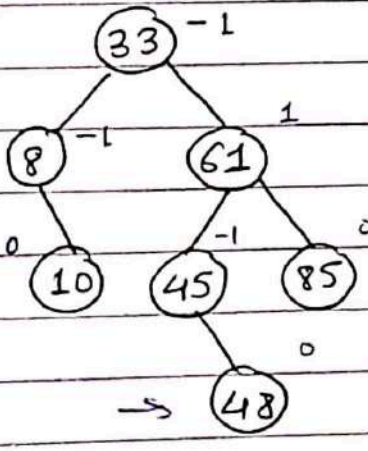
Step 8:



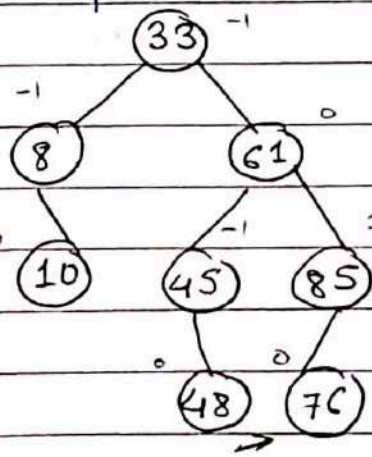

Step 9: Insert 10.




Step 10: Insert 48

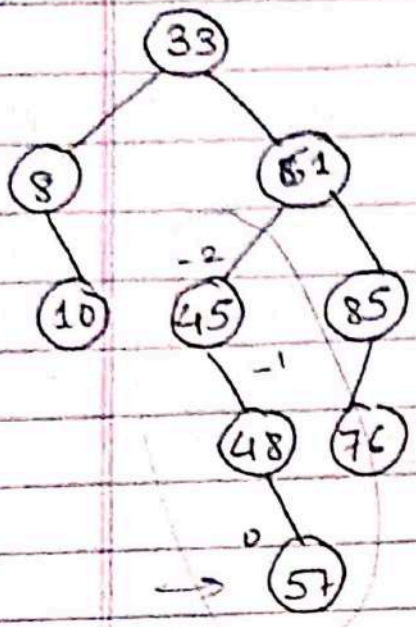



Step 11: Insert 76



Step 12:

Insert 57

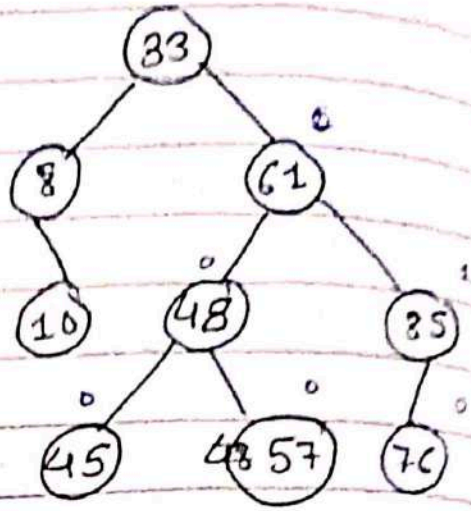



RR rotation

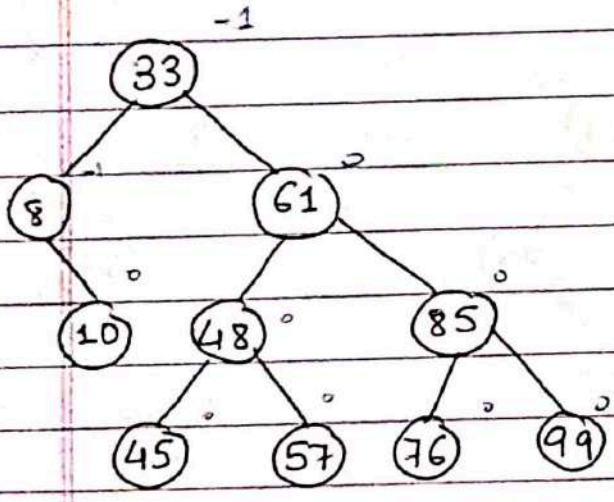
anticlockwise

Mean: 48.

Step 13:




Step 14: Insert 99



5) 92, 24, 6, 7, 11, 8, 22, 4, 5, 16, 19, 20, 78

ORDER = 3 = m

Max no. of child nodes: 3. = m

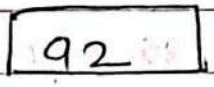
Max no. OR keys = m-1 = 2.

Min no. of child nodes $\frac{m}{2} = \frac{3}{2} = 1$

Min no. of keys: $\frac{m}{2} - 1 = 1$

Step1: Insert 92.

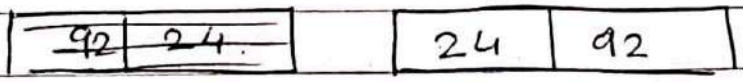
Since 92 is the first element into the tree it is inserted into a new node.




Step2: Insert 24.

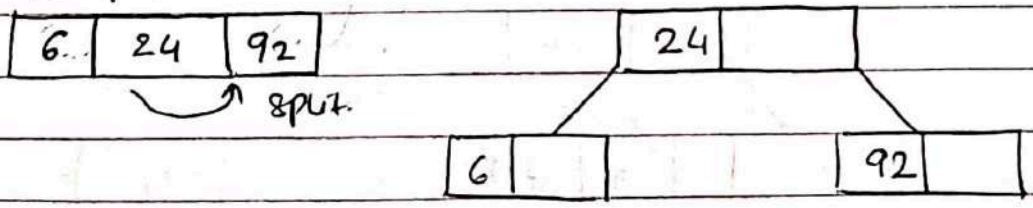
Element 24 is added to existing node.

24 is less than 92 so it is added at previous position of 92.



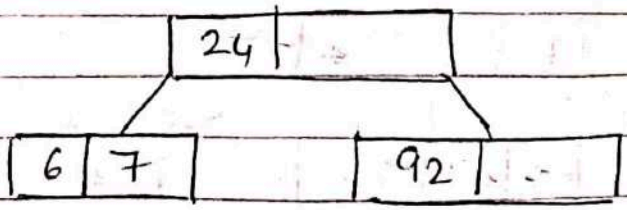

Step3: Insert 6. (Max key).

Element 6 is inserted as the maximum key allowed is 2 we split the node: By split by sending middle value to parent node.



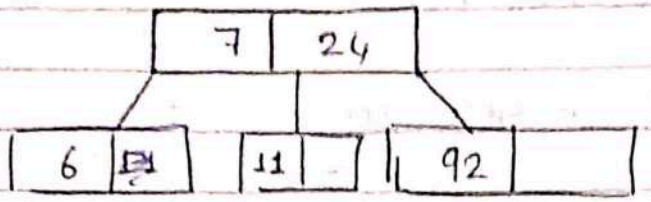

Step4: Insert 7.

7 is less than 24. So we go to left of 24. We insert 7 at the next position of 6.



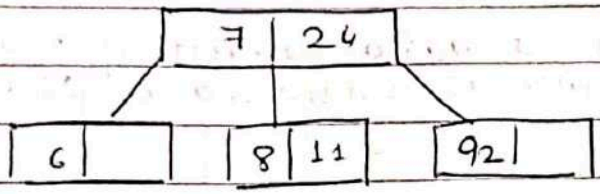
Step6: Insert 11.

Element 11 is less than 24. So we go to left of 24. The leaf node is already full. So we split the node by sending the middle value to its parent node.



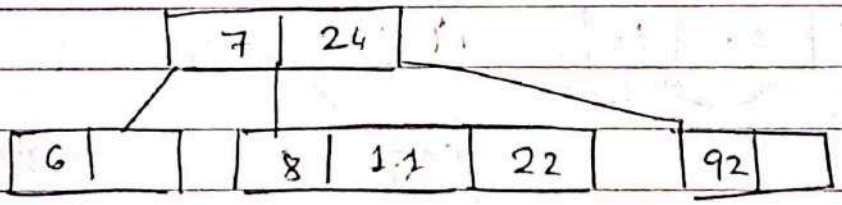

Step7: Insert 8.

8 lies between 7 and 24 so it goes to the middle node and is assign previous of 11.



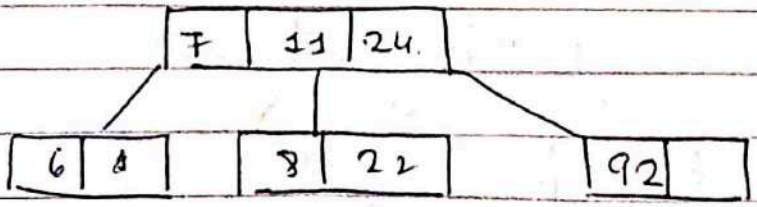

Step8: Insert 22.

22 lies between 7 and 22 so it goes to the middle node and is assigned at the next location of 11




Step9:

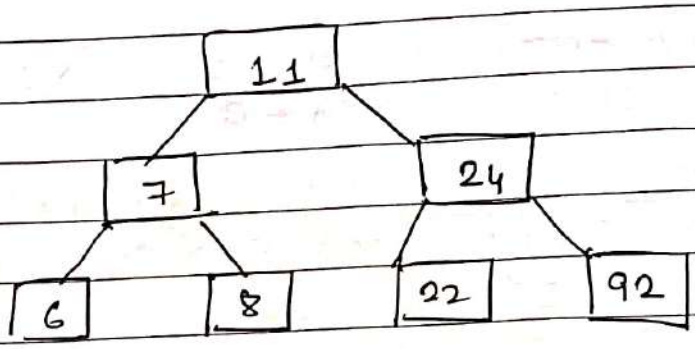
But the middle node is full (max key: 2) therefore we split and move the middle value to the root node



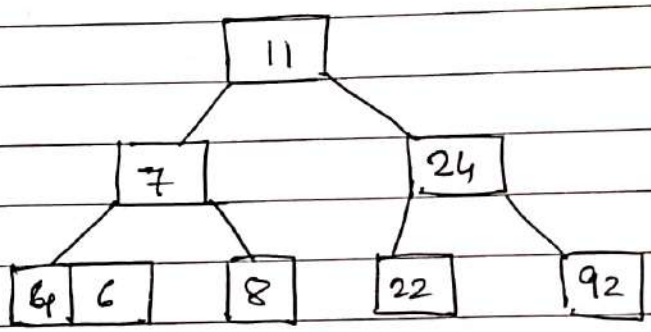
PAGE No.
DATE 9/10/18.

92 24 6 7 11 8 22 4 5 16

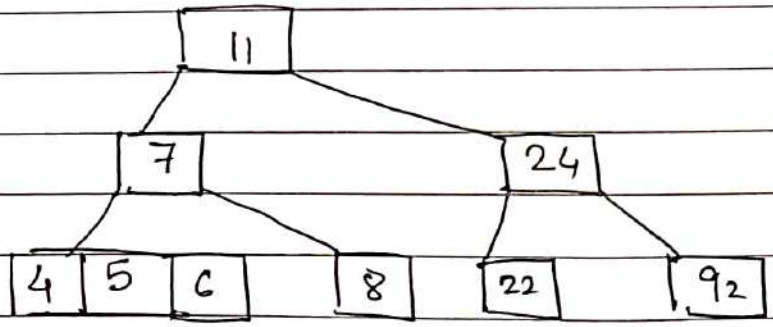
Step 10:
But Root node is full therefore we split by sending middle value 11 to root node.



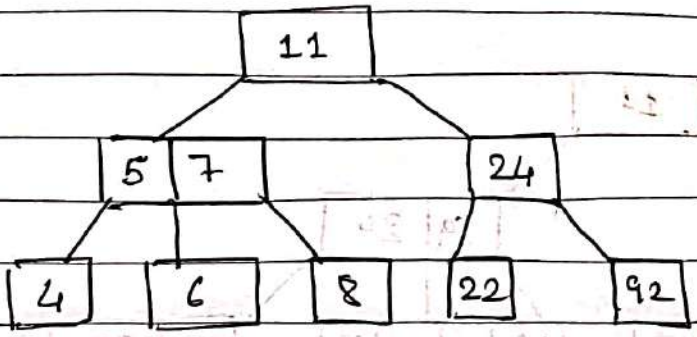

Step 11: Insert 4.



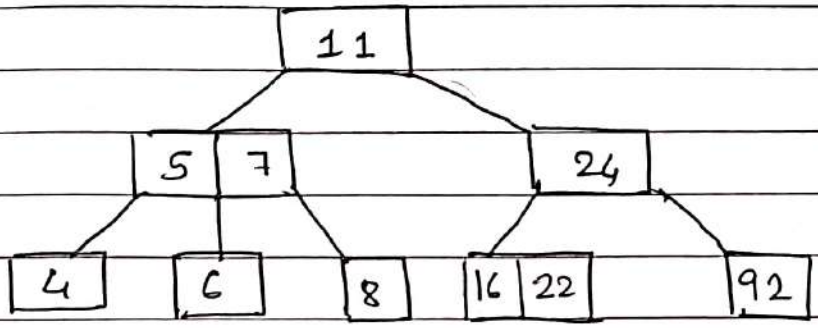

Step 12: Insert 5.



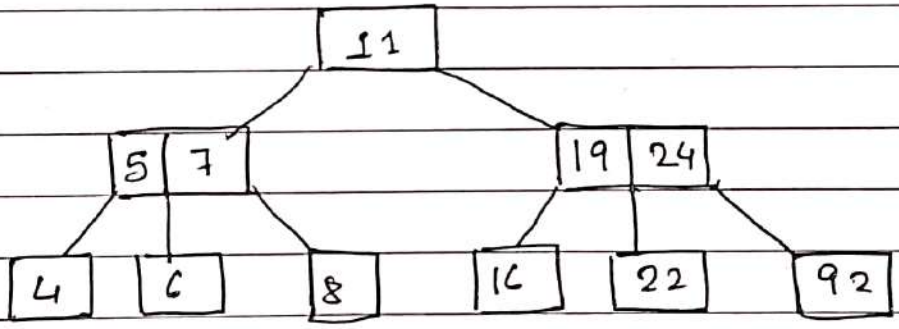

Step 13: Max Key : 2.
of child nodes of 7 : Therefore we shift middle value to parent node.



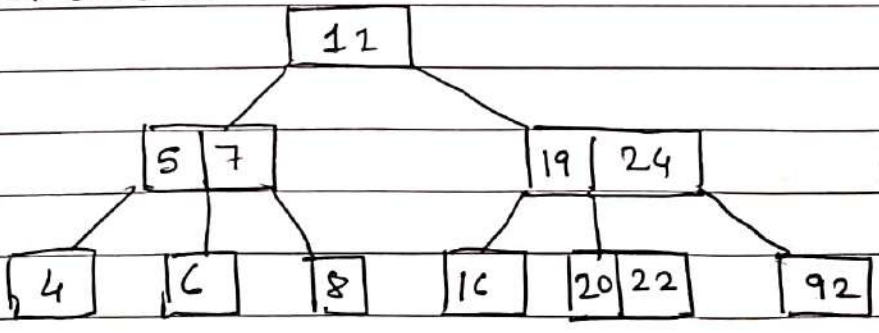

Step14: Insert 16




Step15: Insert 19 : Max key: 2



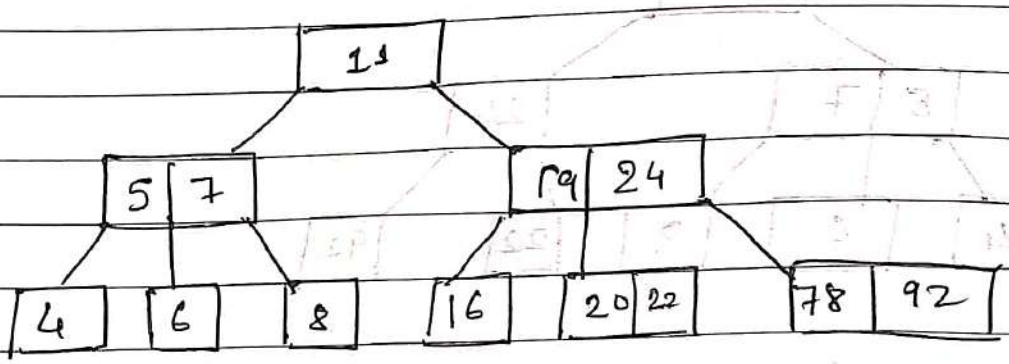

Step16: Insert 20:



PAGE No.

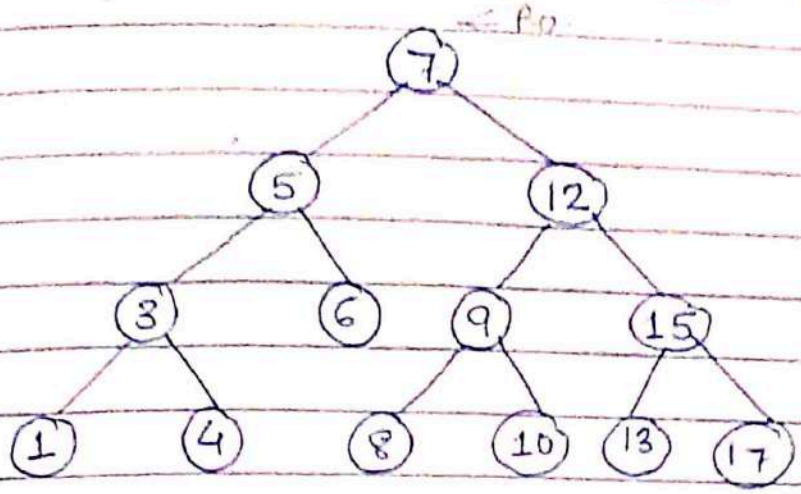
DATE / /

Insert 78:



4, 5, C

7. BST find post order traversal sequence.



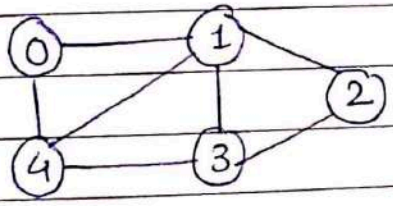

Solution: Post order:

Traverse Left subtree (L), Traverse Right (R), Traverse Rootnode (RO)

1 - 4 - 3 - 6 - 5 - 8 - 10 - 9 - 13 - 17 - 15 - 12 - 7

8. Breadth-first-Traversal of graph starting vertex 0.

Assignment

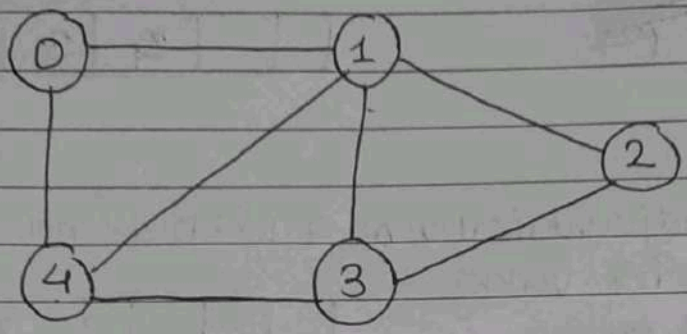



9. Depth-First-Search

Assignment

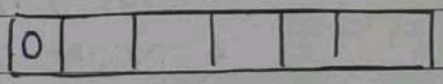
Sonal R. Sonarghare
21107033 (52)
Data Structures.

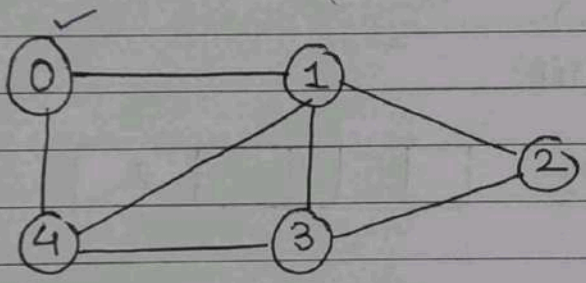
BFS: Queue.



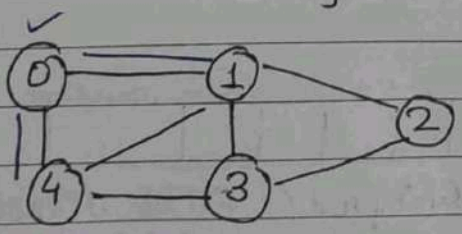

Step1: Select vertex 0 as starting point (visit 0)
- Insert 0 into the queue.

Queue:

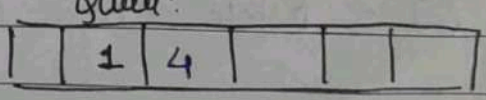


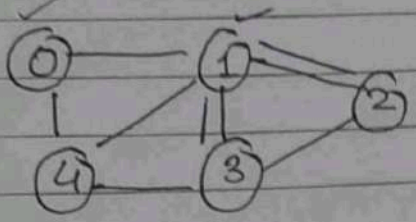

Step2: Visit all adjacent vertiles of 0 which are not visited (1, 4)
- Insert newly visited vertiles into queue and delete 0 from queue

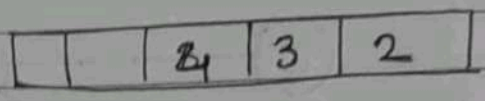



Queue:

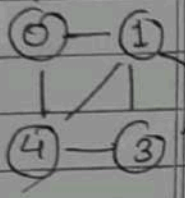


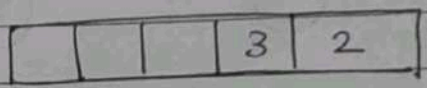
Step3: visit all adjacent vertices of 1 which are not visited which are (2, 3)
- Insert newly visited vertices into queue and delete 1



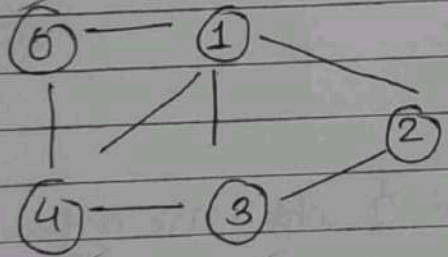



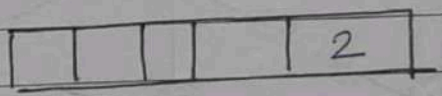

Step4: Visit all adjacent vertices of 4 which are not visited (there is not vertex)
- Delete 4 from queue.



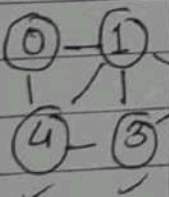



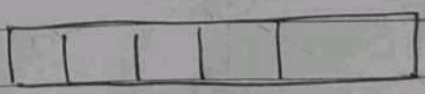

Step5: visit all adjacent vertices of 3 which are not visited (there is no vertex)
- Delete 3 from queue.



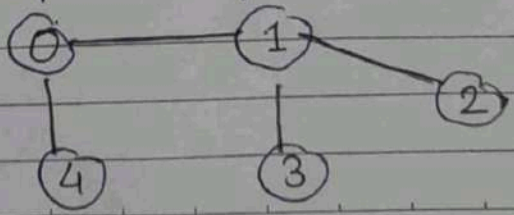




Step6: visit all adjacent vertices of 2 which are not visited (there is no vertex).
- Delete 2 from queue.



Queue became Empty. So stop the BFS process, final result of BFS is a spanning tree:



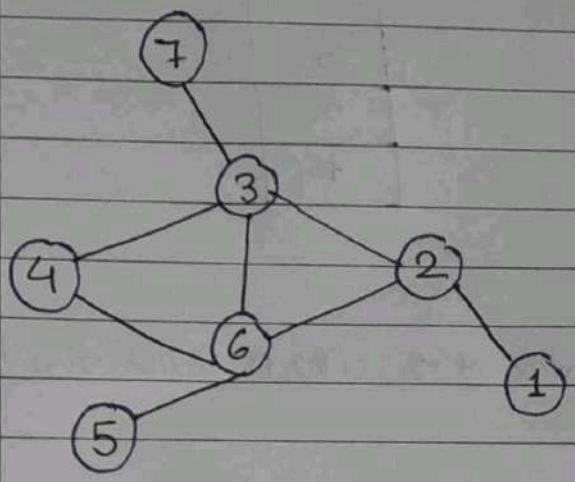
Sonal R. Sonarghare.

(5G).

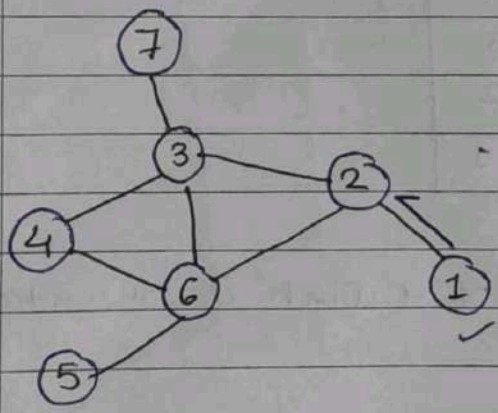
Data Structure

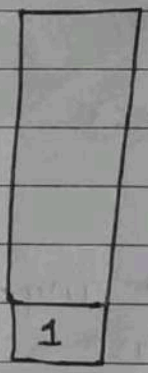
Depth First Search:

Starting with vertex 1.

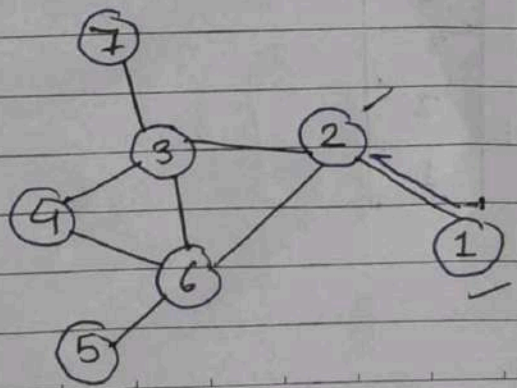


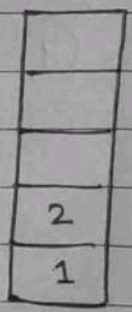

Step1 : Select vertex 1 as starting point. Push 1 on stack.



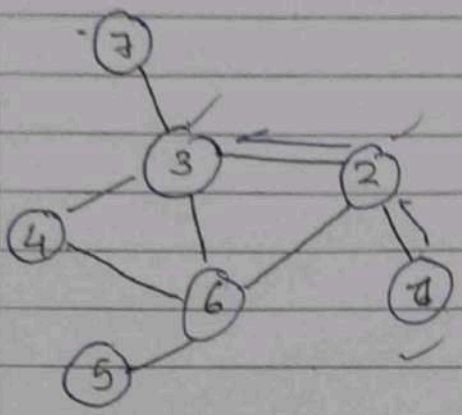



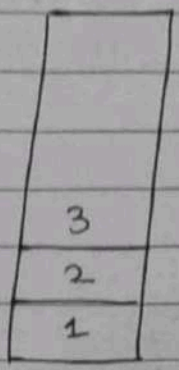

Step2: Visit any adjacent vertex of 1 which is not visited. Push 2 on stack



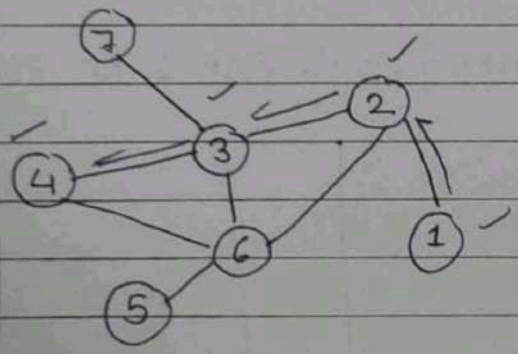



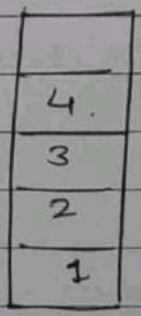
Step3: Visit any adjacent Vertex of 2 which is not visited. (3)
Push 3 on the Stack



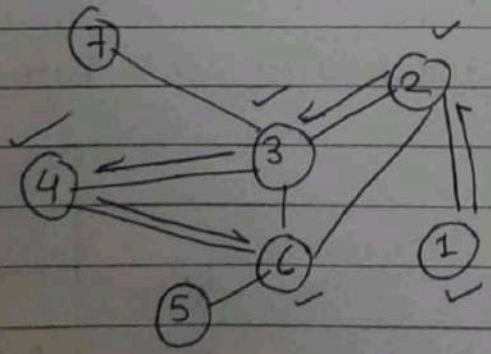



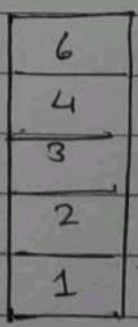

Step4: Visit any adjacent vertex of 3 which is not visited (4)
- Push 4 on the stack.



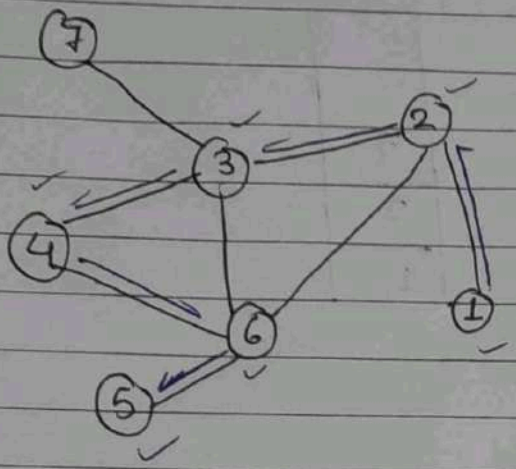




Step5: Visit any adjacent vertex of 4 which is not visited (6).
- Push 6 on the stack.



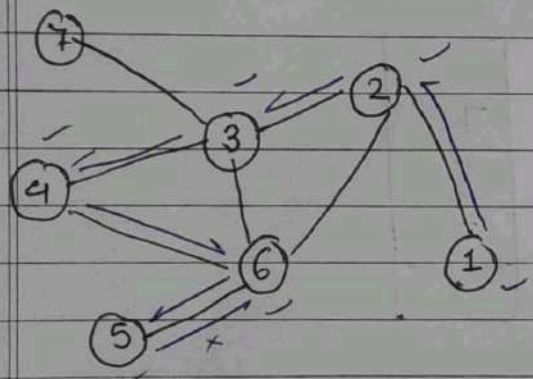



Step6: Visit any adjacent vertex of 6 which is not visited (5)
- Push 5 on the stack.



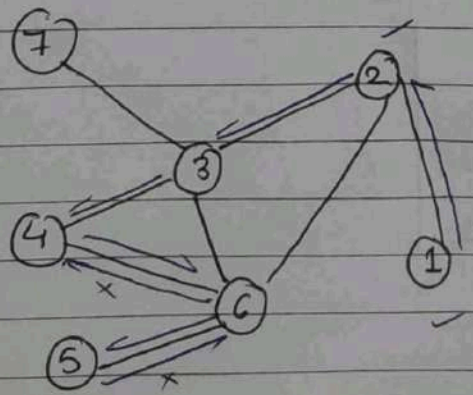

| 5 |
|---|
| 6 |
| 4 |
| 3 |
| 2 |
| 1 |

Step7: There is no new vertex to be visited from 5 So we use back track
Pop 5 from the stack.



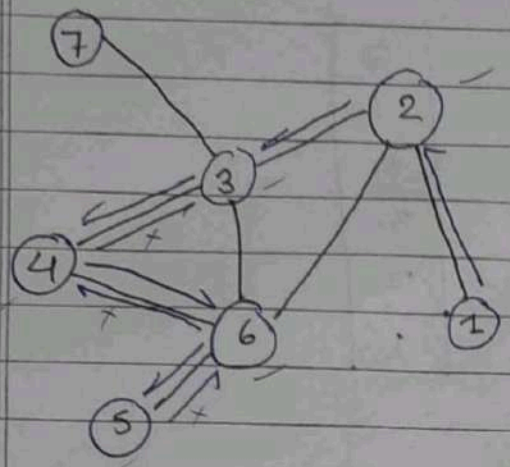

| |
|---|
| 6 |
| 4 |
| 3 |
| 2 |
| 1 |

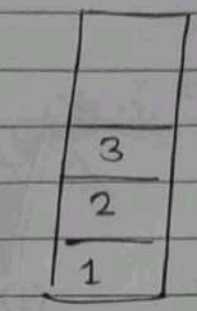
Step8: There is no new vertex to be visited from 6. So we use back track. Pop 6 from the stack.




| |
|---|
| 4 |
| 3 |
| 2 |
| 1 |

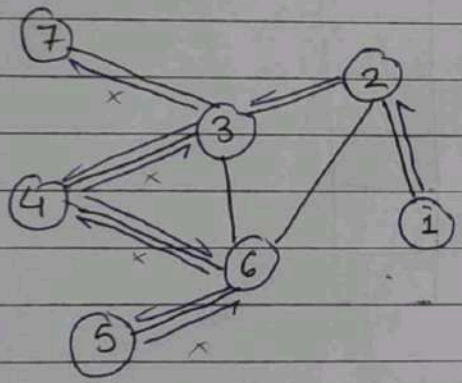
Step9: Now it no new vertex to be visited from 4. So we use back track. Pop 4 from the stack

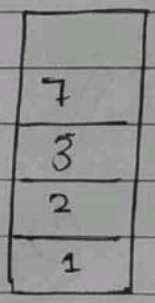


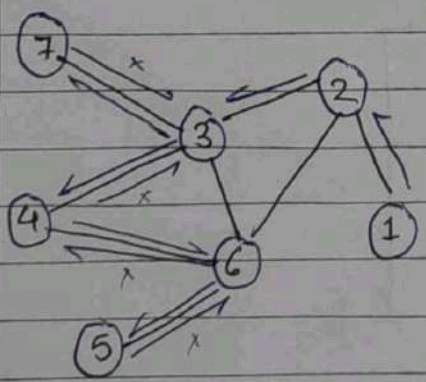

Step10: Now it no new vertex to be visited from 3.

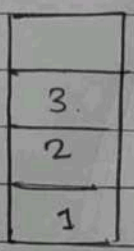
Step10: Visit any adjacent vertex of 3 which is 7. Push 7 on stack



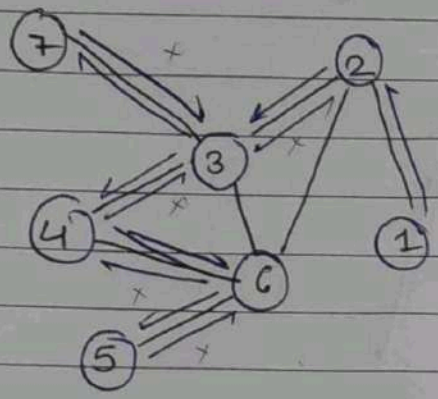



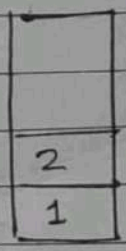

Step11: Now it no new vertex to be visited from 7. So we use back track. Pop 7 from stack.



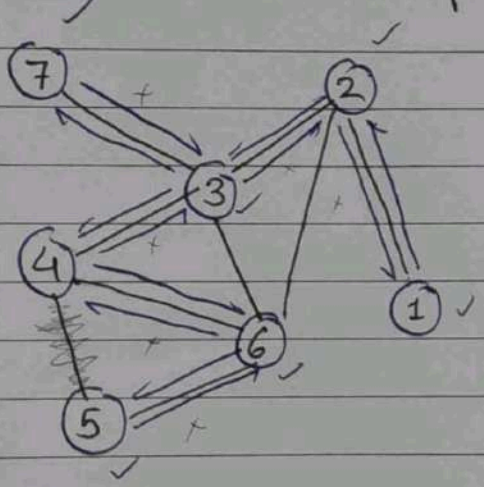



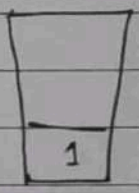
Step12: There is no new vertex to be visited from 3. So we use back track. Pop 3 from stack.



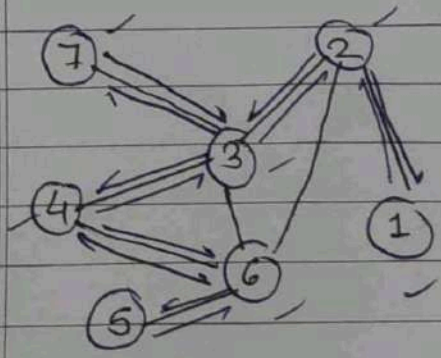




Step13: There is no new vertex to be visited from 2. So we use back track. Pop 2 from stack.

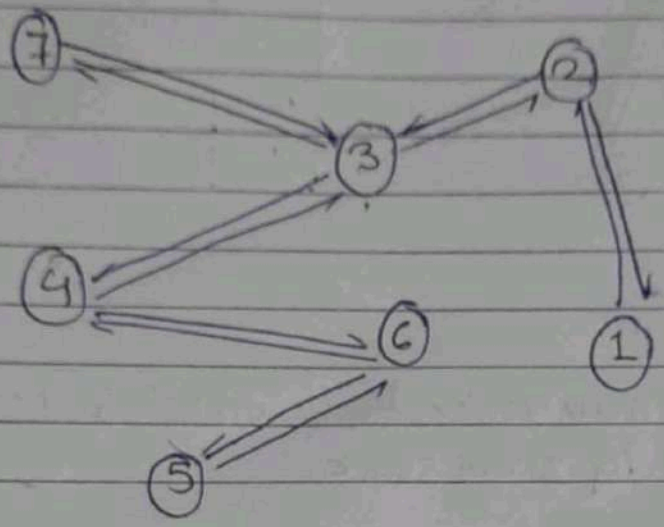


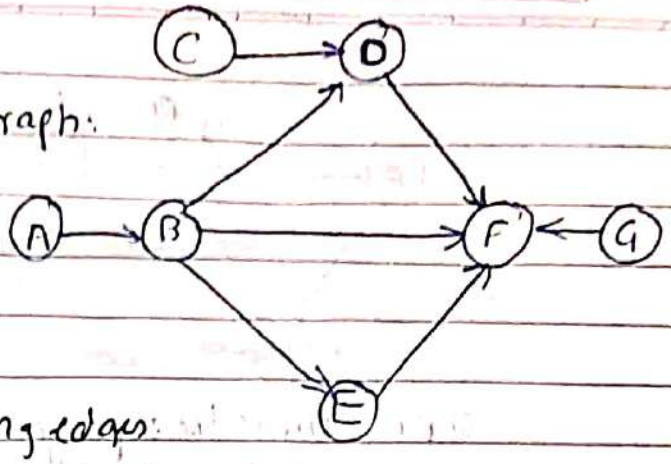

Step14: There is no new vertex to be visited from 1. So we use back track. Pop 1 from stack.



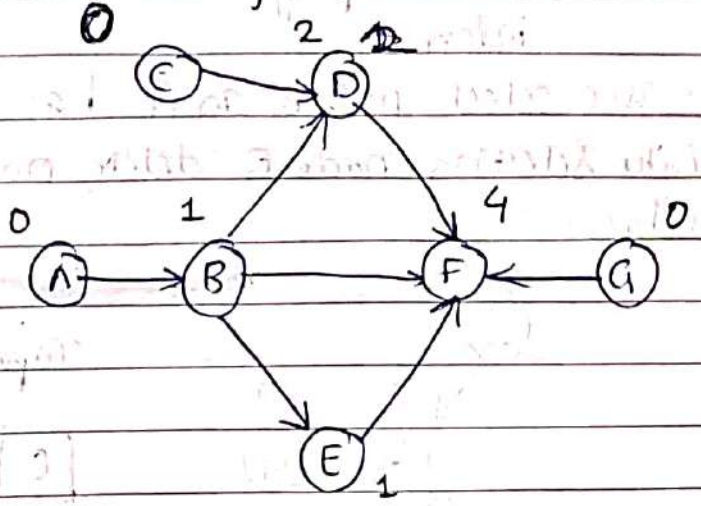

Stack became empty so stop DFS traversal
- final result of DFS traversal is following spanning tree.



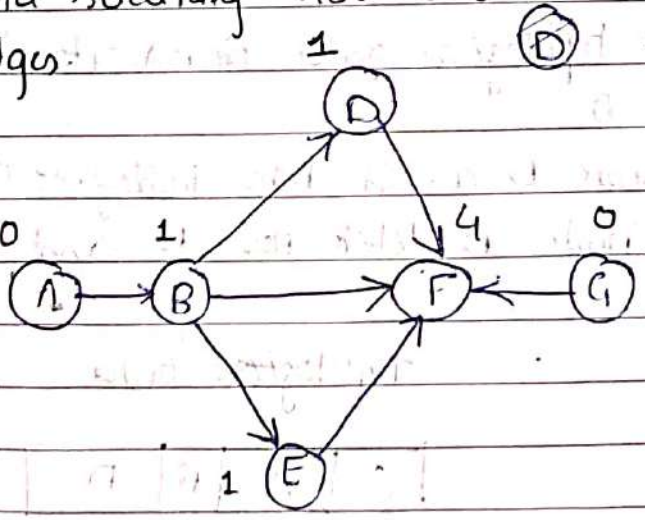
11 Directed Acyclic Graph:
Topological sort on the graph:




Step1:
Write indegrees of each
vertex :: indegree: incoming edges.

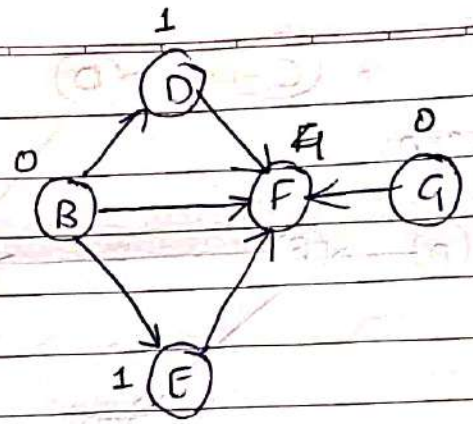



Step2:
Write the topological order from vertex having indegree 0.
So we select node C as it has indegree 0.
After selecting node C delete node C and outgoing edges.




Topological order.


Step3: Write the topological order from vertex having indegree 0.
We select A as it has indegree 0.
After selecting node A delete node A and outgoing edges.



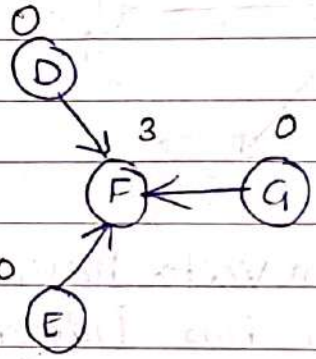

Topological order.

| C | A | | | | |
|---|---|---|---|---|---|

Step4: Write the topological order from vertex having indegree 0.

∴ We select node B as it has indegree 0.

After selecting node B delete node B and outgoing edges.



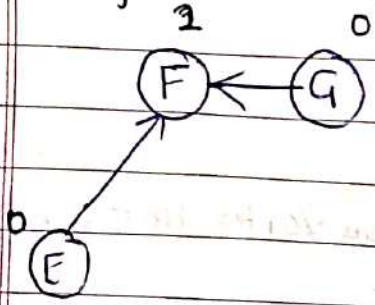

Topological order

| C | A | B | |
|---|---|---|---|

Step5: Write the topological order from vertex having indegree 0.

∴ We select node D as it has indegree 0.

After selecting node D delete node B and outgoing edges.




Topological order

| C | A | B | D |
|---|---|---|---|

Page No.
Date: / /

Step6: Write the topological order from vertex having indegree 0.

So we select E has it has indegree 0.

After selecting node E delete node E and its outgoing edges.


Topological order:

| C | A | B | D | E |
|---|---|---|---|---|

Step7: Write the topological order from vertex having indegree 0.

So we select G as it has indegree 0.

After selecting node G delete node G and its outgoing edges.


Topological order:

| C | A | B | D | E | G |
|---|---|---|---|---|---|

Step8:

Topological order:

| C | A | B | D | E | G | F |
|---|---|---|---|---|---|---|

12. Depth First Search (DFS) traversal:

- DFS algorithm is a traversing or searching tree data structure.
- DFS method visits the graph nodes along the different paths.
- It begins by analyzing the nodes from start to the end node and then proceeds along the next path from start node.
- This process is repeated until all the graph nodes are visited.
- DFS method also requires frequent back tracking to the already analyzed nodes.
- It uses the stack data structure for storing data.

Algo:

Step1: Define a stack of size total no. of vertices in the graph

Step2: Select any vertex as a starting point for traversal. Push it on stack.

Step3: Visit any one adjacent vertex of the vertex which is at the top of stack and not visited. Push it on the stack.

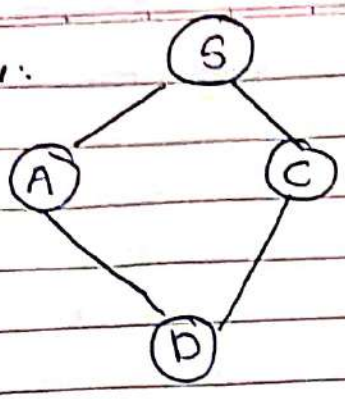
Step4: Repeat step3 until there are no new vertex to be visited from the vertex on the top of stack

Step5: When no new vertex to be visited use backtracking and pop one vertex from the stack.

Step6: Repat step 3, 4 and 5 until stack becomes Empty.

Step7: When stack becomes empty then produce final spanning tree by removing unused edges from the graph.

Example:



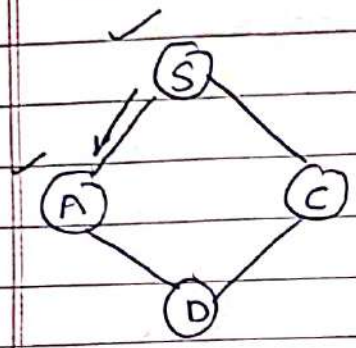

Stack

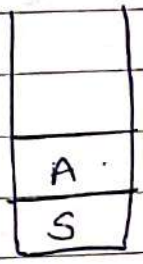

**Step 1:**

Select vertex S as a starting point. Push it on the stack.

Visit any adjacent vertex of S which is not visited.

We choose: A.

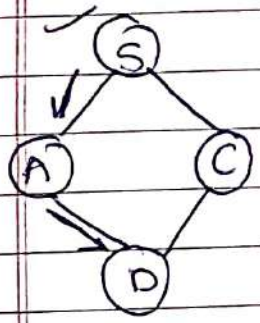



Push A on stack.

Visit any adjacent vertex of A which is not visited.

We Choose D.

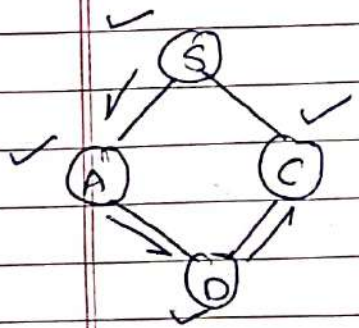


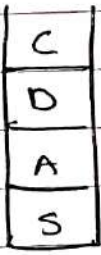


Push D on stack.

Visit any vertex (adjacent) of D which is not visited.

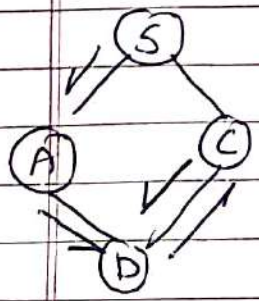
We choose C.

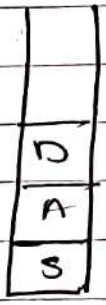



Push C on the stack.

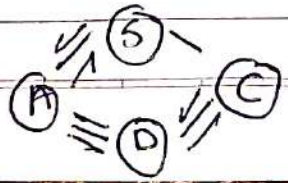
Visit any new vertex adjacent of C. There is not new adjacent vertex so we pop C from the stack.




Visit any new adjacent vertex of D there is no new vertex so we pop D from stack.

Likewise we pop till stack is empty.



14. Linear Probing:

45, 8, 33, 85, 61, 10, 48, 76, 89.

$h(k,i) = [h(k)+i] \bmod m$.


Step1: Draw an empty table of size 10.

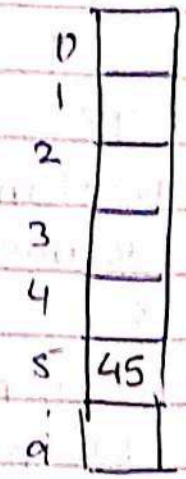
Step2:

Insert 45:

$h(45,0) = (h(45)+0) \bmod 10$

$= 45 \bmod 10$

$= 5.$




Step3: Insert 8.

$h(8,0) = (h(8)+0) \bmod m$

$= 8 \bmod 10 = 8.$

Step3: Insert 33

$h(33,0) = (h(33)+0) \bmod 10$

$= 33 \bmod 10 = 3.$

Step4: Insert 85

$h(85,0) = h(85)+0) \bmod 10$

$= 85 \bmod 10 = 5$ ✓

$h(85,1) = (h(85)+1) \bmod 10$

$= 86 \bmod 10$

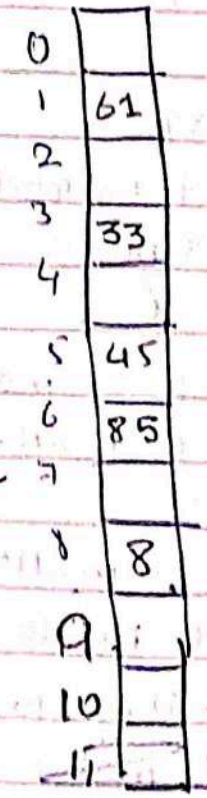
$= 6.$

Step5: Insert 61.

$h(61,0) = (h(61)+0) \bmod 10$

$= 61 \bmod 10$

$= 1.$



PAGE No.
DATE / /

Step6: Insert 10.

$h(10,0) = (h(10)+0) \bmod 10$
$= 10 \bmod 10 = 0.$

Step7: Insert 48.

$h(48,0) = (h(48)+0) \bmod 10$
$= 48 \bmod 10$
$= 8.$

$h(48,1) = (h(48)+1) \bmod 10$
$= 49 \bmod 10$
$= 9.$

Step8: Insert 76.

$h(76,0) = (h(76)+0) \bmod 10$
$= 76 \bmod 10$
$= 6.$

$h(76,1) = (h(76)+1) \bmod 10$
$= 77 \bmod 10$
$= 7.$

Step9: Insert 89.

$h(89,0) = (h(89)+0) \bmod 10$
$= 89 \bmod 10$
$= 9.$ ✓

$h(89,1) = (h(89)+1) \bmod 10$
$= 90 \bmod 10$
$= 0$ ✓

$h(89,2) = (89+2) \bmod 10$
$= 91 \bmod 10 = 1$ ✓

$h(89,3) = (89+3) \bmod 10$
$= 92 \bmod 10 = 2.$

| | |
|---|---|
| 0 | 10 |
| 1 | 61 |
| 2 | |
| 3 | 33 |
| 4 | |
| 5 | 45 |
| 6 | 85 |
| 7 | 76. |
| 8 | 8 |
| 9 | 48 |
| 10 | |
| ~~11~~ | |

| | |
|---|---|
| 0 | 10 |
| 1 | 61 |
| 2 | 89 |
| 3 | 33 |
| 4 | |
| 5 | 45 |
| 6 | 85 |
| 7 | 76 |
| 8 | 8 |
| 9 | 48 |
| 10 | |
| ~~11~~ | |

Division Modulo: $h(k) = k \% m$

45, 8, 33, 85, 61, 10, 48, 76, 89

Step1: Draw empty table of Size 11

Step2: Insert 45

$\therefore h(45) = 45 \% 10$
$= 5.$

Step3: Insert 8.

$h(8) = 8 \% 10$
$= 8.$

Step4: Insert 33

$h(33) = 33 \% 10$
$= 3.$

Step5: Insert 85

$h(85) = 85 \% 10$
$= 5.$

prob → (1)

| | |
|---|---|
| 0 | 10 |
| 1 | 61 |
| 2 | |
| 3 | 33 |
| 4 | |
| 5 | 45 |
| 6 | 85 |
| 7 | 76 |
| 8 | 8 |
| 9 | 48 |
| 10 | 76 |
| 11 | |

Since 5 is already occupied, 85 get stored in next available location i.e 6. (prob = 1)

Step6: Insert 61.

$h(61) = 61 \% 10 = 1.$

Step7: Insert 10

$h(10) = 10 \% 10 = 0$

Step8: $48 \% 10 = 8.$

Insert 48 Since 8 is already occupied, 48 stored in 9 (next).

Step9: Insert 76 $h(76) = 76 \% 10 = 6.$ $\therefore (7)$

Step10: Insert 89 $h(89) = 89 \% 10 = 9.$ $\therefore (10)$

PAGE No.
DATE / /

15. Given : Quadratic probing
Table size m=10.
Keys:- 27, 72, 63, 42, 36, 18, 29, 101
c1 = 1 c2 = 3.

$h(K,i) = (h(K) + c1 \cdot i + c2 i) \bmod m$.

Steps:-
Draw an empty table of size 10.

| | |
|---|---|
| 0 | |
| 1 | |
| 2 | |
| 3 | |
| 4 | |
| 5 | |
| 6 | |
| 7 | |
| 8 | |
| 9 | |

Step2:
Insert 27.
$\therefore h(27,0) = (h(27) + 1 \cdot 0 + 3 \cdot 0) \bmod 10$.
$= 27 \bmod 10$
$= 7$.

| | | |
|---|---|---|
| | 0 | |
| | 1 | |
| S2 → | 2 | 72 |
| S3 → | 3 | 63. |
| | 4 | |
| | 5 | |
| | 6 | |
| S1 → | 7 | 27. |
| | 8 | |
| | 9 | |

Step3 Insert 72
$h(72,0) = (72 + 1 \cdot 0 + 3 \cdot 0) \bmod 10$
$= 72 \bmod 10$
$= 2$.

Step4 Insert 63 = $(63 + 1 \cdot 0 + 3 \cdot 0) \bmod 10$
$= 63 \bmod 10$
$= 3$.

Step5 Insert 42 = $(42 + 1 \cdot 0 + 3 \cdot 0) \bmod 10$
$= 42 \bmod 10 = 2$.

PAGE No.
DATE / /

i=1. Insert 42.

$h(42,1) = (42 + 1\cdot1 + 3\cdot1) \bmod 10$
$= (42 + 1 + 3) \bmod 10$
$= 46 \bmod 10$
$= 6.$

| | |
|---|---|
| 0 | |
| 1 → | 36. |
| 2 | 72 |
| 3 | 63 |
| 4 | |
| 5 | |
| 6 | 42 |
| 7 | 27 |
| 8 | 18 |
| 9 | 29. |

Step6: Insert 36

$h(36,0) = (h(36) + 1\cdot0 + 3\cdot0) \bmod 10$
$= 36 \bmod 10$
$= 6$ ✓

Si $i=1$ $h(36,1) = (h(36) + 1\cdot1 + 3\cdot1) \bmod 10$
$= (36 + 1 + 4) \bmod 10$
$= (36 + 5) \bmod 10$
$= 41 \bmod 10$
$= 1.$

Step7: Insert 18.

$h(18,0) = (18 + 1\cdot0 + 3\cdot0) \bmod 10$
$= 18 \bmod 10$
$= 8.$

| | |
|---|---|
| 0 | |
| 1 | 36 |
| 2 | 72 |
| 3 | 63 |
| 4 | |
| 5 | 101. |
| 6 | 42 |
| 7 | 27 |
| 8 | 18 |
| 9 | 19. |

Step8: Insert 29.

$h(29,0) = (29 + 1\cdot0 + 3\cdot0) \bmod 10$
$= 29 \bmod 10$
$= 9.$ ✓

Step9: Insert 101

$h(101,0) = (h(101) + 1\cdot0 + 3\cdot0) \bmod 10$
$= 101 \bmod 10$
$= 1.$ ✓

$i=1$ $h(101,1) = (h(101) + 1\cdot1 + 3\cdot1) \bmod 10$
$= (101 + 1 + 3) \bmod 10$
$= 105 \bmod 10$
$= 5.$

16) Linear Probing:

63, 82, 94, 77, 53, 87, 23, 55, 10, 44

$h(k,i) = [h(k)+i] \bmod 10$.

Step1:

Draw Empty Hash Table.

Given size of Table = 10 = m

| | |
|---|---|
| 0 | |
| 1 | |
| 2 | |
| 3 | |
| 4 | |
| 5 | |
| 6 | |
| 7 | |
| 8 | |
| 9 | |

Step2:

Insert 63.

~~h(k)~~ k=63 i=0 m=10

$h^*(k,i) = [h(k)+i] \bmod m$

$h(63,0) = (h(63)+0) \bmod 10$

$= 63 \bmod 10$

$= 3.$ → 3

| | |
|---|---|
| 0 | |
| 1 | |
| 2 | |
| 3 | 63 |
| | |
| | |
| | |
| 9 | |

Step3:

Insert 82. k=82 i=0 m=10

$h(k,i) = [h(k)+i] \bmod m$

$h(82,0) = [h(82)+0] \bmod 10$

$= 82 \bmod 10 = 2.$ → 2

| | |
|---|---|
| 0 | |
| 1 | |
| 2 | 82 |
| 3 | |
| 9 | |

Step4:

Insert 94. k=94 i=0 m=10

$h(94,0) = [h(94)+0] \bmod 10$

$= 94 \bmod 10$

$= 4.$

| | |
|---|---|
| 0 | |
| 1 | |
| 2 | 82 |
| 3 | 63 |
| 4 | 94 |
| | |

Steps:

Insert 77

K=77 i=0 m=10

h(77,0) = [h(77)+0] mod 10

= 77 mod 10

= 7.

| | |
|---|---|
| 0 | |
| 1 | |
| 2 | 82 |
| 3 | 63 |
| 4 | 94 |
| 5 | |
| 6 | |
| 7 | 77 |
| 8 | |
| 9 | |

Steps:

Insert 53 i=0

h(53,0) = (h(53)+0] mod m

= 53 mod 10

= 3. ✓

i=1.

h(53,1) = (h(53)+1] mod 10

= (53+1) mod 10

= 54 mod 10

= 4. ✓

i=2.

h(53,2) = (h(53)+2) mod 10

= 55 mod 10

= 5

| | |
|---|---|
| 0 | |
| 1 | |
| 2 | 82 |
| 3 | 63 |
| 4 | 94 |
| 5 | 53 |
| 6 | |
| 7 | 77 |
| 8 | |
| 9 | |

Steps:

Insert 87

h(87,0) = [h(87)+0] mod 10

= 87 mod 10

= 7.

i=1.

h(87,1) = [h(87)+1] mod 10

= 88 mod 10

= 8.

| | |
|---|---|
| 0 | |
| 1 | |
| 2 | 82 |
| 3 | 63 |
| 4 | 94 |
| 5 | 53 |
| 6 | |
| 7 | 77 |
| 8 | 87 |
| 9 | |

Step 8:

Insert 23

$h(23,0) = [h(23)+i] \bmod m$

$= 23 \bmod 10$

$= 3$ ✓

$i = 1$

$h(23,1) = [h(23)+1] \bmod 10$

$= 24 \bmod 10$ ✓

$= 4$ ✓

$i = 2$

$h(23,2) = (h(23)+2) \bmod 10$

$= 25 \bmod 10$

$= 5$ ✓

$i = 3$

$h(23,3) = (h(23)+3) \bmod 10$

$= 26 \bmod 10$

$= 6.$

| | |
|---|---|
| 0 | |
| 1 | |
| 2 | 82 |
| 3 | 63 |
| 4 | 94 |
| 5 | 53 |
| 6 | 23 |
| 7 | 77 |
| 8 | 87 |
| 9 | |

Step 9:

Insert 55.

$h(55,0) = (h(55)+0] \bmod 10$

$= 55 \bmod 10$

$= 5$ ✓

$i = 1$ $h(55,1) = [h(55)+1] \bmod 10$

$= 56 \bmod 10$

$= 6$ ✓

$i = 2$ $h(55,2) = [h(55)+2] \bmod 10$

$= 57 \bmod 10$

$= 7$ ✓

$i = 3$ $h(55,3) = [h(55)+3] \bmod 10$

$= 58 \bmod 10.$ 8 ✓

$i = 4$ $h(55,4) = (h(55)+4) \bmod 10$

$= 59 \bmod 10$

$= 9$

| | |
|---|---|
| 0 | |
| 1 | |
| 2 | 82 |
| 3 | 63 |
| 4 | 94 |
| 5 | 53 |
| 6 | 23 |
| 7 | 77 |
| 8 | 87 |
| 9 | 55. |

PAGE No.
DATE / /

Step 10:

Insert 10.

$h(10, 0) = (h(10) + 0) \bmod 10$

$= 10 \bmod 10$

$= 0.$

| | |
|---|---|
| 0 | 10 |
| 1 | |
| 2 | 82 |
| 3 | 63 |
| . | |
| . | |
| 9 | 55 |

Step 11:

Insert 44.

$h(44, 0) = (h(44) + 0) \bmod 10$

$= 44 \bmod 10$

$= 4.$

$i = 1$ $h(44, 1) = (44 + 1) \bmod 10$

$= 45 \bmod 10$

$= 5.$ ✓

$i = 2$ $h(44, 2) = 44 + 2 \bmod 10$

$= 46 \bmod 10$

$= 6$ ✓

$i = 3 \rightarrow 7$

$i = 4 \rightarrow 8$

$i = 5 \rightarrow 9$

$i = 6$ $h(44, 6) = (44 + 6) \bmod 10$

$= 50 \bmod 10$

$= 0$

$i = 7$ $h(44, 7) = (44 + 7) \bmod 10$

$= 51 \bmod 10$

$= 1.$

| | |
|---|---|
| 0 | 10 |
| 1 | 44 |
| 2 | 82 |
| 3 | 63 |
| 4 | 94 |
| 5 | 53 |
| 6 | 23 |
| 7 | 77 |
| 8 | 87 |
| 9 | 55. |

PAGE NO
DATE / /

What is Hashing:

- Hashing is a technique of mapping a large chunk of data into small table using a hashing function.
- It is process of mapping keys and values into the hash table by using a hash function.
- It is done for faster access to elements.

PAGE No.
DATE / /

Quadratic Probing:

63, 82, 94, 77, 53, 87, 23, 55, 10, 44

$h(K,i) = [h(K) + i^2] \bmod m$

Steps: Draw empty table of size 10.

| | |
|---|---|
| 0 | |
| 1 | |
| 2 | |
| 3 | |
| 4 | |
| 5 | |
| 6 | |
| 7 | |
| 8 | |
| 9 | |

Step2: Insert 63.

$h(K,i) = (h(K) + i^2) \bmod m$

$= (63 + 0) \bmod 10$

$= 63 \bmod 10$

$= 3.$

| | |
|---|---|
| 0 | |
| 1 | |
| 2 | |
| 3 | 63 |
| 4 | |
| 5 | |
| 6 | |
| 7 | |
| 8 | |
| 9 | |

Step3: Insert 82.

$h(82,0) = (h(82) + 0^2) \bmod 10$

$= 82 \bmod 10$

$= 2.$

| | |
|---|---|
| 0 | |
| 1 | |
| 2 | 82 |
| 3 | 63 |
| 4 | |
| 5 | |
| 6 | |
| 7 | |
| 8 | |
| 9 | |

Step4: Insert 94

$h(94,0) = (h(94) + 0^2) \bmod 10$

$= 94 \bmod 10$

$= 4$

| | |
|---|---|
| 0 | |
| 1 | |
| 2 | 82 |
| 3 | 63 |
| 4 | 94 |
| 5 | |
| 6 | |
| 7 | |
| 8 | |
| 9 | |

PAGE No.
DATE / /

Steps: Insert 77.

$h(77, 0) = (77 + 0^2) \bmod 10$

$= 77 \bmod 10$

$= 7.$

| | |
|---|---|
| 0 | |
| 1 | |
| 2 | 82 |
| 3 | 63 |
| 4 | 94 |
| 5 | |
| 6 | |
| 7 | 77 |
| 8 | |
| 9 | |

Step6: Insert 53.

$h(53, 0) = (53 + 0^2) \bmod 10$

$= 53 \bmod 10$

$= 3$ ✓

i=1 $h(53, 0) = (53 + 1^2) \bmod 10$

$= 54 \bmod 10$

$= 4$ ✓

| | |
|---|---|
| 0 | |
| 1 | |
| 2 | 82 |
| 3 | 63 |
| 4 | 94 |
| 5 | |
| 6 | |
| 7 | 77 |
| 8 | |
| 9 | 53 |

i=2 $h(53, 2) = (53 + 2^2) \bmod 10$

$= 53 + 4 \bmod 10$

$= 57 \bmod 10$

$= 7$ ✓

i=3 $h(53, 3) = h(53) + 3^2) \bmod 10$

$= (53 + 9) \bmod 10$

$= 62 \bmod 10$

$= 2$ ✓

i=4 $h(53, 4) = (h(53) + 4^2) \bmod 10$

$= (53 + 16) \bmod 10$

$= 69 \bmod 10$

$= 9.$

PAGE No.
DATE / /

Step7: Insert 87.

$h(87,0) = (87+0^2) \bmod 10$
$= 87 \bmod 10$
$= 7$ ✓

i=1 $h(87,1) = (h(87)+1) \bmod 10$
$= 88 \bmod 10$
$= 8.$

| | |
|---|---|
| 0 | |
| 1 | |
| 2 | 82 |
| 3 | 63 |
| 4 | 94 |
| 5 | |
| 6 | |
| 7 | 77 |
| 8 | 87 |
| 9 | 53 |

Step8: Insert 23.

$h(23,0) = (23+0) \bmod 10$
$= 3$ ✓

$h(23,1) = (23+1) \bmod 10$
$= 24 \bmod 10$
$= 4$ ✓

$h(23,2) = (23+4) \bmod 10$
$= 27 \bmod 10$
$= 7$

$h(23,3) = (23+9) \bmod 10$
$= 32 \bmod 10$
$= 2$ ✓

$h(23,4) = (23+16) \bmod 10$
$= 39 \bmod 10$
$= 9.$ ✓

$h(23,5) = (23+25) \bmod 10$
$= 48 \bmod 10 = 8$ ✓

$h(23,6) = (23+36) \bmod 10$
$= 59 \bmod 10 = 9$ ✓

$h(23,7) = (23+49) \bmod 10$
$= 72 \bmod 10 = 2$ ✓

NO slot available.

PAGE NO:
DATE: / /

Step 9: Insert 55

h(55,0) = 55 mod 10
= 5 ✓

| | |
|---|---|
| 0 | |
| 1 | |
| 2 | 82 |
| 3 | 63 |
| 4 | 94 |
| 5 | 55 |
| 6 | |
| : | |

Step 10: Insert 10.

h(10,0) = (10+0) mod 10
= 0.

| | |
|---|---|
| 0 | 10 |
| 1 | |
| 2 | 82 |
| 3 | 63 |

Step 11: Insert 44

h(44,0) = 44 mod 10
= 4. ✓

h(44,1) = (44+1) mod 10
= 45 mod 10.
= 5 ✓

h(44,2) = (44+4) mod 10
= 48 mod 10
= 8 ✓

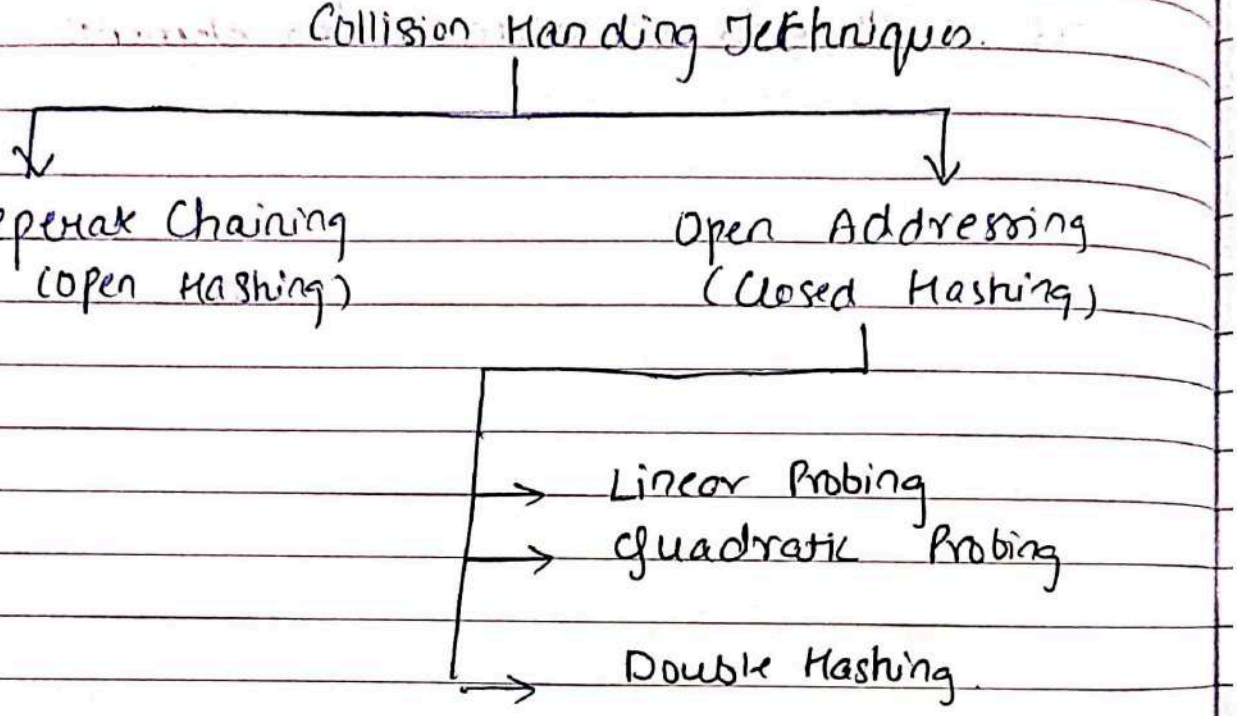
h(44,3) = 44+9) mod 10
= 53 mod 10 = 3.

No spot Available.

| | |
|---|---|
| 0 | 10 |
| 1 | |
| 2 | 82 |
| 3 | 63 |
| 4 | 94 |
| 5 | 55 |
| 6 | . |
| 7 | 77 |
| 8 | 87 |
| 9 | 53. |

17. Collision Handling Techniques:

In Hash Table, collision occurs when two keys are hashed to the same index in a hash table.
It means calculated hash value for the two key is the same.




Seperate Chaining:
To handle the collision
- This technique creates a linked list to the slot for which collision occurs.
- The new key is then inserted in the linked list.
- These linked list to the slot appear like chain that is why, this technique is called seperate chaining.

Time complexity:
Searching and Deletion: worst case O(n)

OPEN ADDRESSING

In open addressing:

- Unlike seperate Chaining, all the keys are stored inside the hash table.
- No keys is stored outside the hash table.

Technique used for Open Addressing:

1) Linear Probing:

In Linear Probing, we linearly probe for next slot.

- We keep probing until an empty slot is found.

To Resolve:

$h(k,i) = (h[k]+i) \bmod m$

i = count no of probe  k = key.

m = size of hash table

Advantage: easy to compute

Disadvantage: clustering, takes time to search an element or empty slot.

Time complexity: $O(m)$  m = table size.

2) Quadratic Probing:

In quadratic Probing we look for $i^2$th slot in the ith iteration if the given hash value collides with the hash Table.

$h(k,i) = [h[k] + i^2] \cdot \bmod m$.

Advantage: more efficient for closed hash table

Disadvantage: Secondary clustering.

two keys have the same probe sequence when they hash to the same location.

Time complexity: $O(N * L)$, N = length of array

Space complexity: $O(1)$.  L = size of hash table.

3) Double Hashing

idea of applying second hash function to key when collision occurs.

~~h(k,i) = h. hash 1 (key)~~

h(k,i) = ~~hash 1(~~

h(k,i) = [h1[k] + i * h2[k]] mod m.

we increase i when collision occurs.

Advantages: no clusters.

best for probing bcz it can find next free slot in Hash table more quickly than linear probing.

Disadvantages:

Time consuming to compute two hash function

Poor cache performance.

Time complexity: O(m) ... m = size of hash table
worst case.

18 Linear probing with example.

In Linear Probe, we linearly probe for next slot.
The simplest approach to resolve a collision is linear probing.
In this technique, if a value is already stored at a location generated by h(k), it means collision occured then we do a sequential search to find the next empty location.
Here the idea is to place a value in the next available position.
Because in this approach searches are performed sequentially, so it is knowns as linear probing.

Here array or hash table is considered circular because when the last slot reached an empty location not found then search proceeds to the first location of the array.

To Resolve the collision: $h(k,i) = [h(k) + i] \bmod m$
m = size of hash table
$h(k) = (k \bmod m)$
i = probe no varies from 0 to m-1.
Therefore for a key k, the 1st location generated by $[h(k) + 0] \bmod m$, $(i=0)$.

If the location is free, the value is stored at this location. If value successfully stored then probe count becomes 1.
If location is not free then 2nd probe generates address of location given by $[h(k) + 1] \bmod m$.
Time complexity: $O(m)$ m = table size.

PAGE No.
DATE / /

Example:

Insert the following sequence of keys in the hash table of 9, 7, 11, 13, 12, 84.

$h(k,i) = [h(k)+i] \bmod m$

$h(k) = 2k+5$

$m = 10$.

Solution:

Steps:

Empty Hash table of size 10.

| | |
|---|---|
| 0 | |
| 1 | |
| 2 | |
| 3 | |
| 4 | |
| 5 | |
| 6 | |
| 7 | |
| 8 | |
| 9 | |

Step 2:

Insert keys.

$h(k) = 2k+5$.

$h(9) = 2 \times 9 + 5$

$= 18 + 5$

$h(9) = 23$

$\therefore h(k,i) = [h(k)+i] \bmod m$

$h(9,0) = [h(7)+0] \bmod 10$

$= [23] \bmod 10$

$= 3$.

| | |
|---|---|
| 0 | |
| 1 | |
| 2 | |
| 3 | 9 |
| 4 | |
| 5 | |
| 6 | |
| 7 | |
| 8 | |
| 9 | |

Step3:

Insert key

$h(k) = 2k+5$

$h(7) = 2 \times 7 + 5$

$= 19$

$h[k,i] = [h(k)+i] \bmod m$

$h(7,0) = [h(7)+0] \bmod 10$

$= [19+0] \bmod 10$

$= 19 \bmod 10$

$= 9$

| | |
|---|---|
| 0 | |
| 1 | |
| 2 | |
| 3 | 9 |
| 4 | 4 |
| 5 | |
| 6 | |
| 7 | 11 |
| 8 | |
| 9 | 7 |

Step4:

Insert key

$h(k) = 2k+5$

$h(11) = 2 \times 11 + 5 = 22+5 = 27$

$h(k,i) = (h(k)+i] \bmod 10$

$= (h(11)+0] \bmod 10$

$= 27 \bmod 10 = 7$

Step5:

Insert key

$h(k) = 2k+5$.

$h(14) = 2 \times 14 + 5 = 33$.

$h(k,i) = (h(k)+i] \bmod 10$

$= (33+0] \bmod 10$

$= 3$.

Collision occurs location 3 is already full ∴.

$h(14,1) = (h(14)+1] \bmod 10$

$= 33+1 \bmod 10$

$= 34 \bmod 10$

$= 4$.

This is how linear probing collision resolution technique works.