II - I

DATA STRUCTURES

Binary Trees and Hashing

Module 5 QB Solutions

Ujjwal • Vishal • Rajeshwari • Hitesh

PART-A

Q1) The integers 1-1000 are stored in a binary search tree BST. Suppose The Search Algorithm Is Implemented On The Key 363,one of the following sequences is not a possible sequence of nodes that is examined. It is

i. 2, 252, 401, 398, 330, 344, 397, 363

ii. 924, 220, 911, 244, 898, 258, 362, 363

iii. 925, 202, 911, 240, 912, 345, 245, 363

iv. 2, 399, 387, 219, 266, 382, 381, 278, 363

A. **Sequence Analysis**

i. 2, 252, 401, 398, 330, 344, 397, 363

- Start with **2**: Move right (since 2 < 363).
- **252**: Move right (since 252 < 363).
- **401**: Move left (since 401 > 363).
- **398**: Move left (since 398 > 363).
- **330**: Move right (since 330 < 363).
- **344**: Move right (since 344 < 363).
- **397**: Move left (since 397 > 363).
- **363**: Valid sequence.

This sequence is possible.

ii. 924, 220, 911, 244, 898, 258, 362, 363

- Start with **924**: Move left (since 924 > 363).
- **220**: Move right (since 220 < 363).
- **911**: Move left (since 911 > 363).
- **244**: Move right (since 244 < 363).
- **898**: Move left (since 898 > 363).
- **258**: Move right (since 258 < 363).
- **362**: Move right (since 362 < 363).
- **363**: Valid sequence.

This sequence is possible.

iii. 925, 202, 911, 240, 912, 345, 245, 363

- Start with **925**: Move left (since 925 > 363).
- **202**: Move right (since 202 < 363).
- **911**: Move left (since 911 > 363).
- **240**: Move right (since 240 < 363).
- **912**: Move left (since 912 > 363).
- **345**: Move right (since 345 < 363).
- **245**: Move right (since 245 < 363).

This sequence is not valid. After moving right to 912, any subsequent move should not go to 345 or 245, as it violates the BST properties.

iv. 2, 399, 387, 219, 266, 382, 381, 278, 363

- Start with **2**: Move right (since 2 < 363).
- **399**: Move left (since 399 > 363).
- **387**: Move left (since 387 > 363).
- **219**: Move right (since 219 < 363).
- **266**: Move right (since 266 < 363).
- **382**: Move left (since 382 > 363).
- **381**: Move left (since 381 > 363).
- **278**: Move right (since 278 < 363).

This sequence is not valid, as it involves incorrect positioning of 219 after moving left to 387. The path does not align with BST traversal properties.

Conclusion

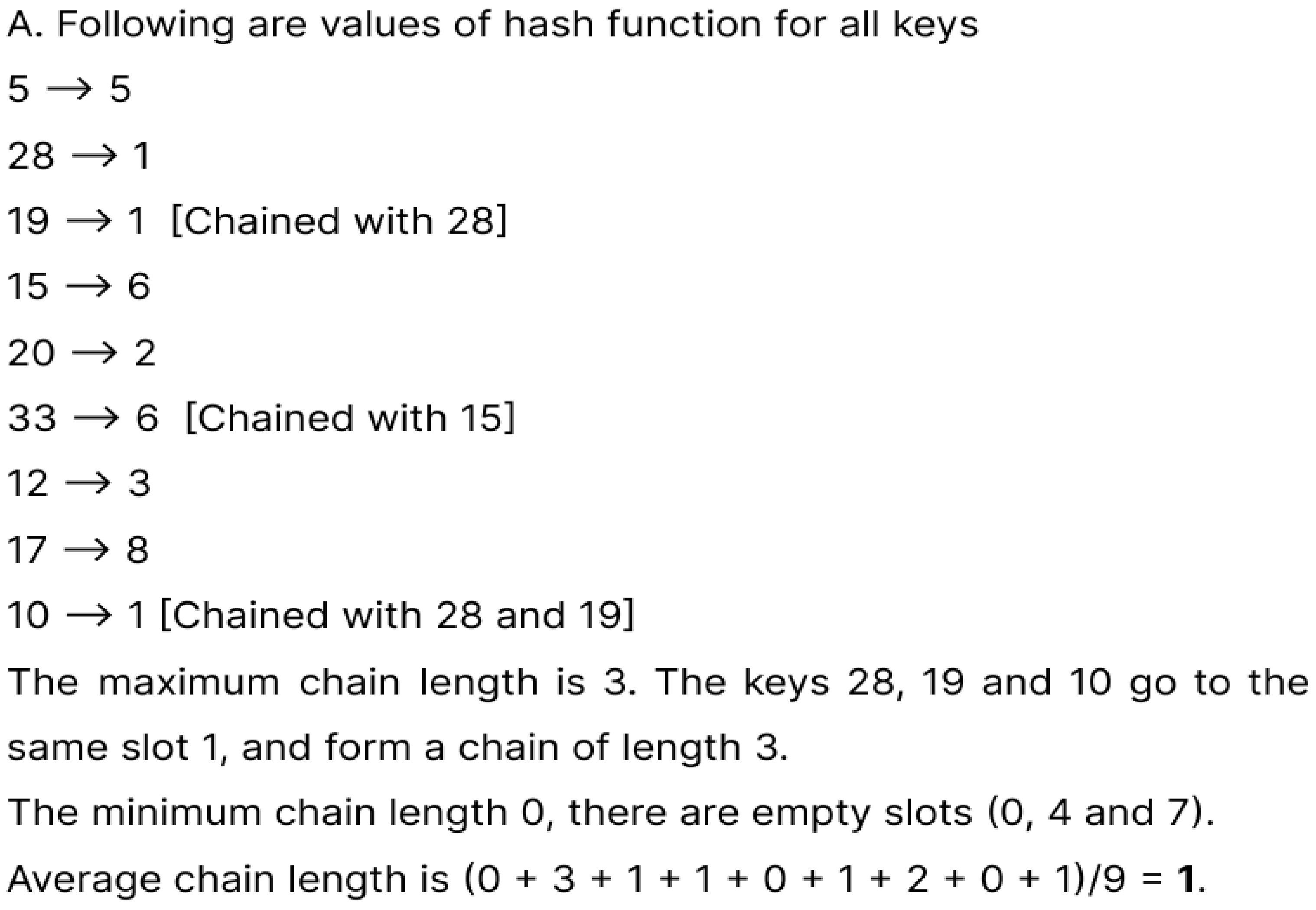
The sequences **iii. 925, 202, 911, 240, 912, 345, 245, 363** and **iv. 2, 399, 387, 219, 266, 382, 381, 278, 363** are not valid. However, among the options provided, sequence iii is the first invalid one.

Therefore, **iii. 925, 202, 911, 240, 912, 345, 245, 363** is the sequence that is not a possible sequence of nodes examined when searching for the key 363 in a BST.

Q2) If h is any hashing function and is used to hash n keys into a table of size m, where m>=n, the expected number of collisions involving a particular key x is :

A. If h is chosen from a universal collection of hash functions and is used to hash n keys into a table of size m, where n ≤ m, the expected number of collisions involving a particular key x is less than 1.

Q3) Consider a hash table with slots.The hash function is h(k)=k mod 9. The Collisions are resolved by chaining. The following 9 keys are inserted in the order: 5, 28, 19, 15, 20, 33,12, 17, 10. Find the maximum, minimum and average chain length in the hash table?



A. Following are values of hash function for all keys

5 → 5

28 → 1

19 → 1 [Chained with 28]

15 → 6

20 → 2

33 → 6 [Chained with 15]

12 → 3

17 → 8

10 → 1 [Chained with 28 and 19]

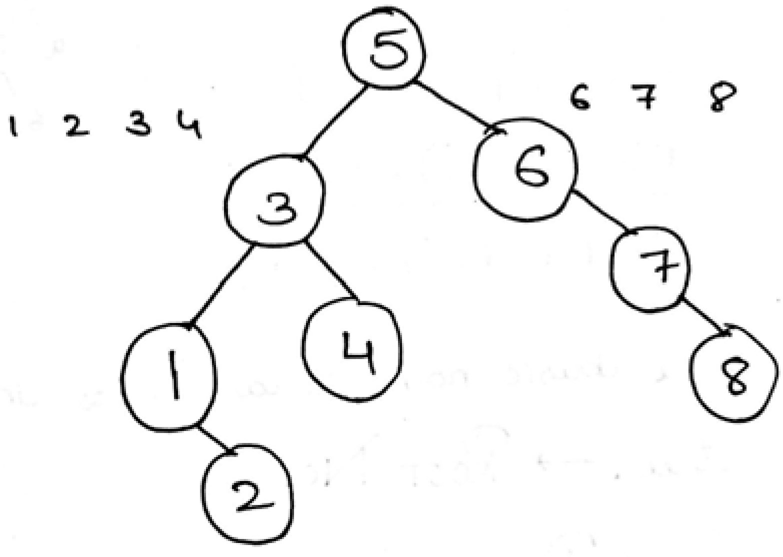
The maximum chain length is 3. The keys 28, 19 and 10 go to the same slot 1, and form a chain of length 3.

The minimum chain length 0, there are empty slots (0, 4 and 7).

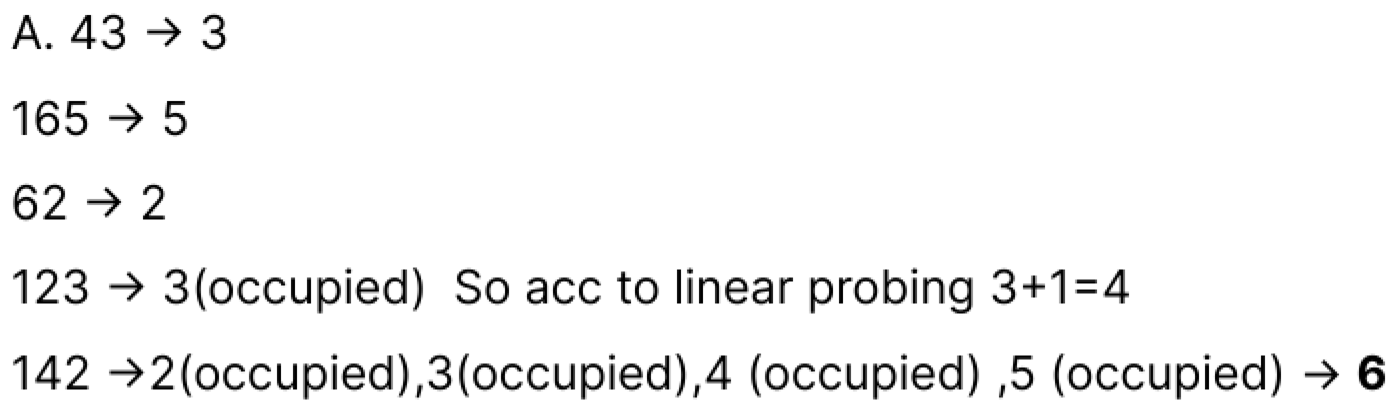
Average chain length is (0 + 3 + 1 + 1 + 0 + 1 + 2 + 0 + 1)/9 = **1**.

Q4) A binary search tree contains the numbers 1, 2, 3, 4, 5, 6, 7, 8. When the tree is traversed in pre-order and the values in each node printed out, the sequence of values obtained is 5, 3, 1, 2, 4, 6, 7, 8. Find the post order traversal sequence of the tree?

A. The tree can be constructed uniquely from the inorder and preorder traversal of the tree. The inorder traversal gives the sorted list i.e 1,2,3,4,5,6,7,8. So the unique BST is



Q5) A hash table contains 10 buckets and uses linear probing to resolve collisions. The key values are integers and the hash function used is key % 10. If the values 43, 165, 62, 123, 142 are inserted in the table, then find the location of the key value 142 in the table?



A. 43 → 3

165 → 5

62 → 2

123 → 3(occupied) So acc to linear probing 3+1=4

142 →2(occupied),3(occupied),4 (occupied) ,5 (occupied) → **6**

Q6) Find the smallest number of keys that will force a B-tree of order 3 to have a height 2?

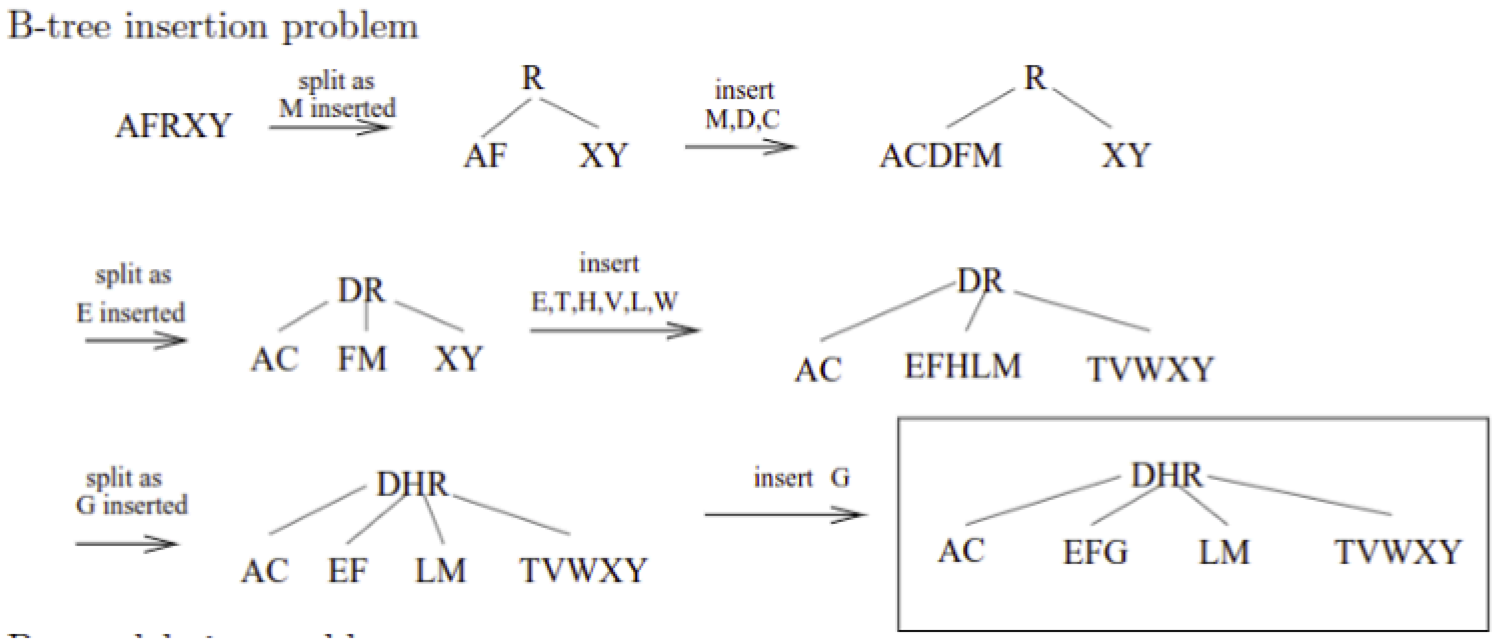
A. AB-tree of order m of height h will have the maximum number of keys when all nodes are completely filled. So, the B-tree will have n = (mh+1–1 keys in this situation. Therefore, the smallest number of keys is equal to 26.

Q7) Suppose that the computer you will be using has disk blocks holding 4096 bytes, the key is 4 bytes long, each child pointer (which is a disk block id)is 4 bytes,theparentis4byteslongand the data record reference (which is a disk block id along with a offset within the block) is 8 bytes. You have an application in which you want to store 1,000,000 items in your B-tree. What value would you select for t? Show how you derived it.) What is the maximum number of disk pages that will be brought to main memory during a search? Remember that the root is kept in main memory at all times.

A. We want to select so that a full node uses as much of a disk block as possible. In a full node, there are 2t−1 keys 4 bytes each), 2t−1 data record references 8 bytes each), 2t child pointers 4 bytes each), a parent pointer 4 bytes), the number of keys 4 bytes) and the leaf bit (which we'll go ahead and assume takes 4 bytes through 1 bit would do). Hence we want topicktas large as we can so that 122t−1 + 4(2t)+12 = 32t≤4096. Solving for yields that we needt= 128. In class, we argued that the number of disk pages that must be read (d-1using the notation from class) is atmos log t(n+1)/2. Since log128(n+1)/2=log 128 2.7 and the number of levels below the root must be an integer, at most 2 disk pages will need to be brought into main memory during a search.

Q8) Show the B-tree such that the results when inserting elements R,Y,F,X,A,M,C,D,E,T,H,V,L,W,G (in that order) branching factor of t=3. You need only draw the trees just before and after each split.

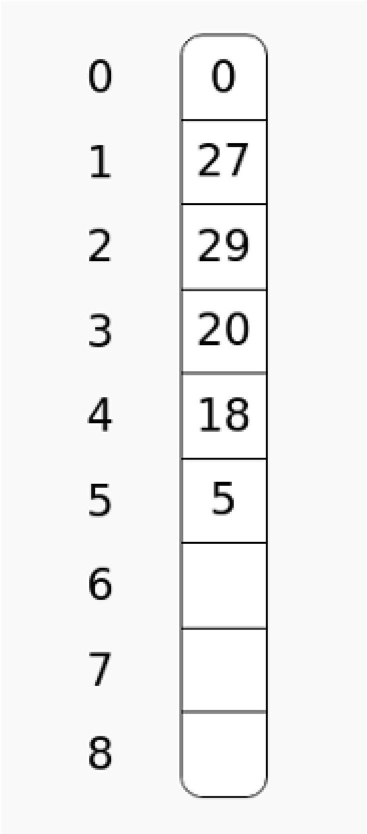
A.



B-tree insertion problem

Q9) Draw ahashtablewithopenaddressingandasizeof9.Usethe hash function ”k%9”. Insert the keys: 5, 29, 20, 0, 27 and 18 into your table (in that order).

A. Hash Table below



0	0
1	27
2	29
3	20
4	18
5	5
6	
7	
8	

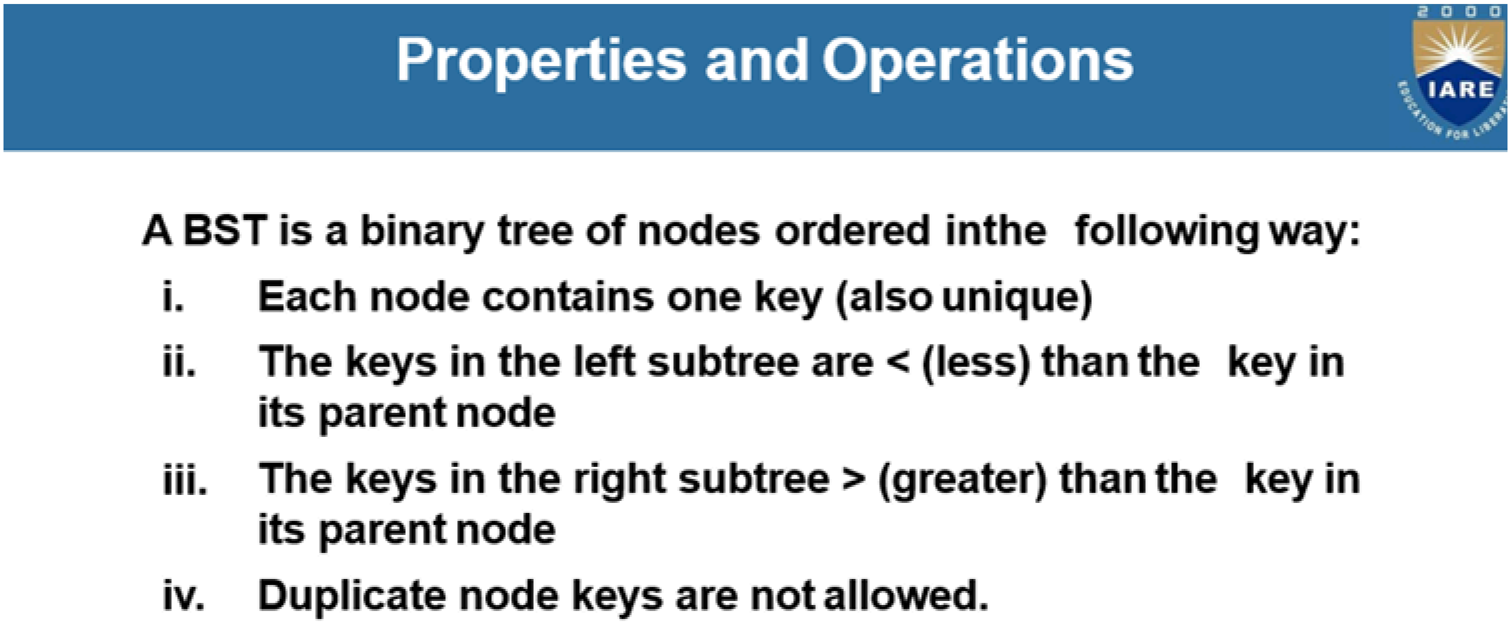
Q10) A cosmetician wants to represent a list of her clients' records (by their ID. For each client wewouldliketomarkwhetherheisa man or she is a woman. Suggest a data structure that supports the following operations in O(log n) time in the worst case, where n is the number of persons (men and women) in the data structure when the operation is executed: 1. Insert(k, c)- Insert a new client c with id = k to the data structure, at first mark the client as a woman. 2. Update(k)– Update client with ID=ktobea man. 3. FindDiff(k)– Find the difference between the number of women and the number of men (— #of women- #of men —) among all the clients with ID smaller than k.

A. AVL Tree.

PART-B

Q1)Define the properties of binary search trees? Write a program to construct a binary search tree with the given keys 8, 3, 10, 1, 6, 14, 4, 7, 13?

A.



Properties and Operations

2000
IARE
EDUCATION FOR LIBERATION

A BST is a binary tree of nodes ordered inthe following way:

i. Each node contains one key (also unique)

ii. The keys in the left subtree are < (less) than the key in its parent node

iii. The keys in the right subtree > (greater) than the key in its parent node

iv. Duplicate node keys are not allowed.

1. Initialize root as NULL.

```
2. Function Insert(Node, Key):
   a. If Node is NULL:
        i.  Create a new Node with Key.
        ii. Return the new Node.
   b. If Key < Node.Key:
        i.  Node.Left = Insert(Node.Left, Key)
   c. Else (Key > Node.Key):
        i.  Node.Right = Insert(Node.Right, Key)
   d. Return Node.

3. For each key in [8, 3, 10, 1, 6, 14, 4, 7, 13]:
   a. Root = Insert(Root, Key)
```

Q2) List out the operations of a binary search tree and write the procedure to search for a key 45 in a given binary search tree containing elements 25, 15, 50, 10, 22, 35, 70, 4, 12, 18, 24, 31, 44, 66, 90?

A. Basic operations on a BST Create: creates an empty tree. Insert: insert a node in the tree. Search: Searches for a node in the tree. Delete: deletes a node from the tree. Inorder: in-order traversal of the tree. Preorder: pre-order traversal of the tree. Postorder: post-order traversal of the tree.

Create

Initially an empty tree without any nodes is created. The variable/identifier which must point to the root node is initialized with a NULL value.

Search

You always start searching the tree at the root node and go down from there. You compare the data in each node with the one you are looking for. If the compared node doesn't match then you either proceed to the right child or the left child, which depends on the outcome of the following comparison: If the node that you are searching for is lower than the one you were comparing it with, you proceed to the left child, otherwise (if it's larger) you go to the right child. Why? Because the BST is structured (as per its definition), that the right child is always larger than the parent and the left child is always lesser.

Breadth-first search BFS Breadth-first search is an algorithm used to traverse a BST. It begins at the root node and travels in a lateral manner (side to side), searching for the desired node. This type of search can be described as O(n) given that each node is visited once and the size of the tree directly correlates to the length of the search.

Depth-first search DFS With a Depth-first search approach, we start with the root node and travel down a single branch. If the desired node is found along that branch, great, but if not, continue upwards and search unvisited nodes. This type of search also has a big O notation of O(n).

Insert

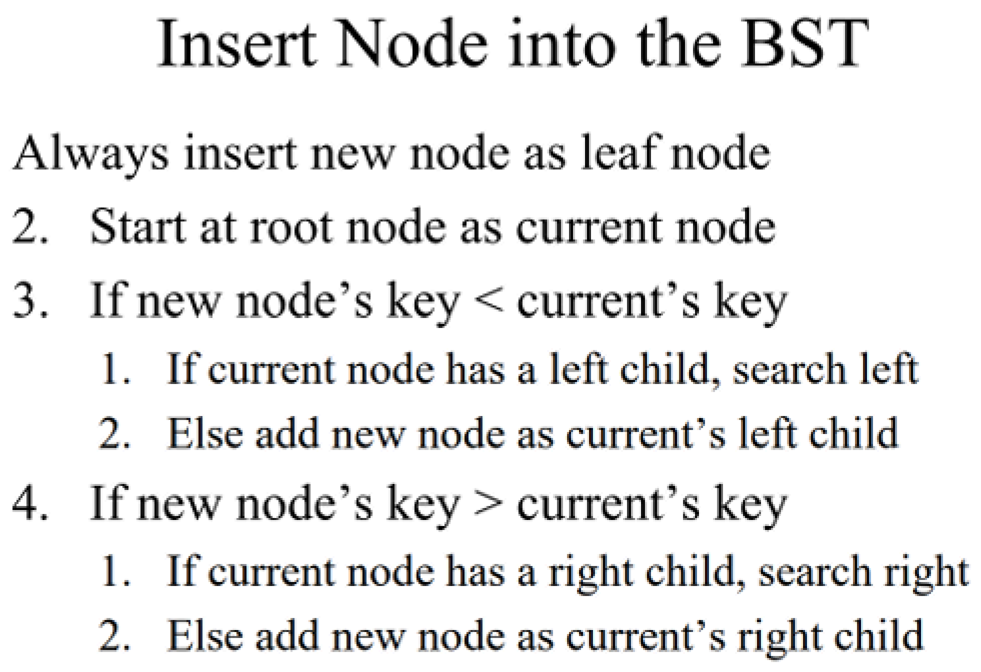
It is very similar to the search function. You again start at the root of the tree and go down recursively, searching for the right place to insert our new node, in the same way as explained in the search function. If a node with the same value is already in the tree, you can choose to either insert the duplicate or not. Some trees allow duplicates, some don't. It depends on the certain implementation.

Deletion There are 3 cases that can happen when you are trying to delete a node. If it has,

No subtree (no children): This one is the easiest one. You can simply just delete the node, without any additional actions required. One subtree (one child): You have to make sure that after the node is deleted, its child is then connected to the deleted node's parent. Two subtrees (two children): You have to find and replace the node you want to delete with its inorder successor (the leftmost node in the right subtree).

Q3) Write the procedure for inserting an element 60 in a given binary search tree containing elements 25, 15, 50, 10, 22, 35, 70, 4, 12, 18, 24, 31, 44, 66, 90?

A.



Insert Node into the BST

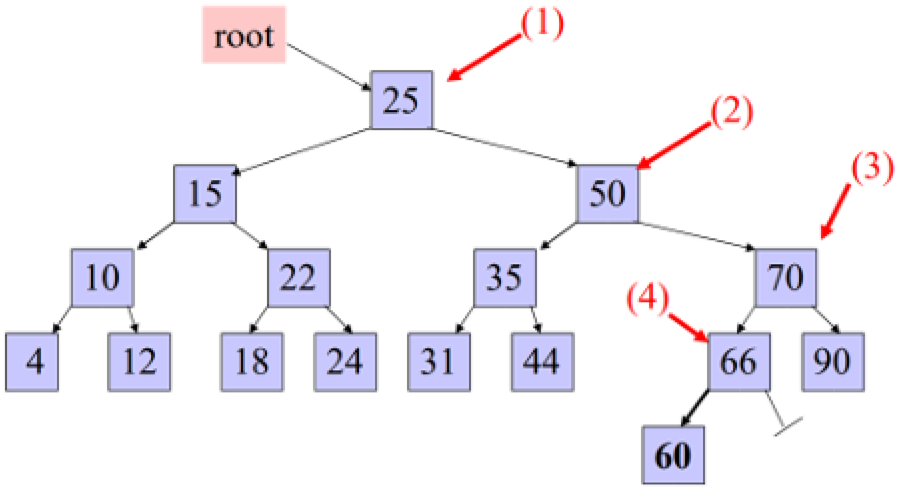
Always insert new node as leaf node

2. Start at root node as current node
3. If new node's key < current's key
    1. If current node has a left child, search left
    2. Else add new node as current's left child
4. If new node's key > current's key
    1. If current node has a right child, search right
    2. Else add new node as current's right child

.

Example: insert 60 in the tree:

1. start at the root, 60 is greater than 25, search in right subtree
2. 60 is greater than 50, search in 50's right subtree
3. 60 is less than 70, search in 70's left subtree
4. 60 is less than 66, add 60 as 66's left child



Q4) Explain the different possibilities that arise while deleting an element from a given binary search tree containing elements 50, 30, 70, 20, 40, 60, 80? i. Delete 20 ii. Delete 30 iii. Delete 50

A.

1) Node to be deleted is the leaf: Simply remove from the tree.

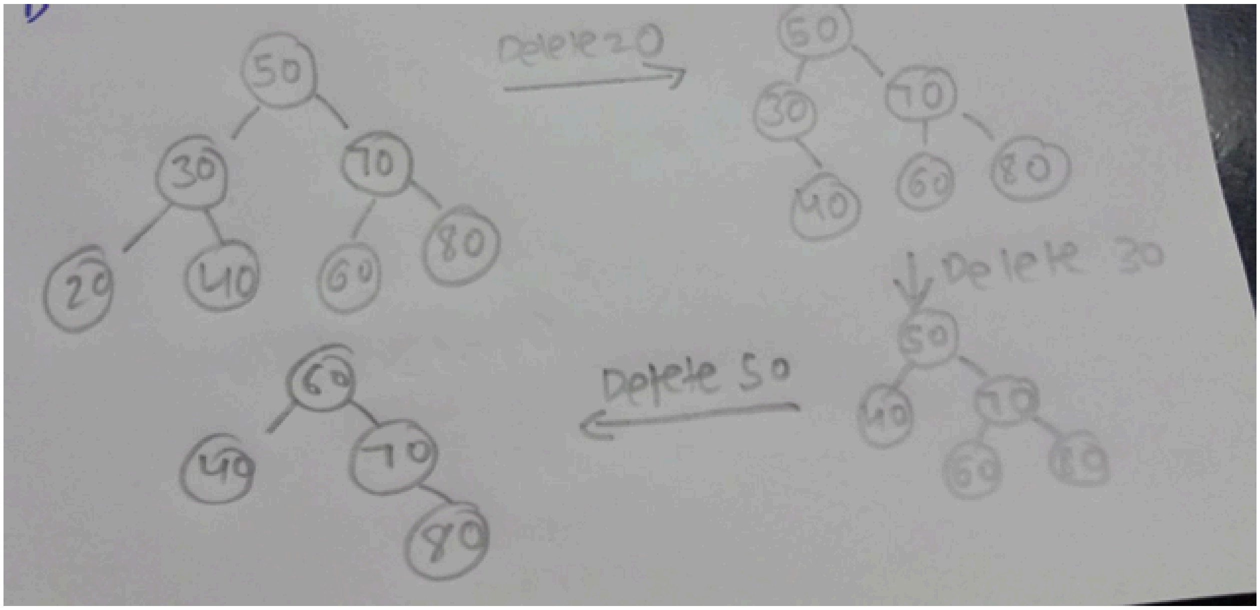
```
          50                            50
       /       \         delete(20)    /  \
      30       70       --------->   30    70
     / \      /    \                  \   /    \
   20  40 60      80                  40  60   80
```

2) Node to be deleted has only one child: Copy the child to the node and delete the child

```
         50                          50
       /    \       delete(30)      /  \
      30      70    --------->    40     70
       \     / \                        / \
       40  60   80                    60   80
```

3) Node to be deleted has two children: Find inorder successor of the node. Copy contents of the inorder successor to the node and delete the inorder successor. Note that inorder predecessor can also be used.

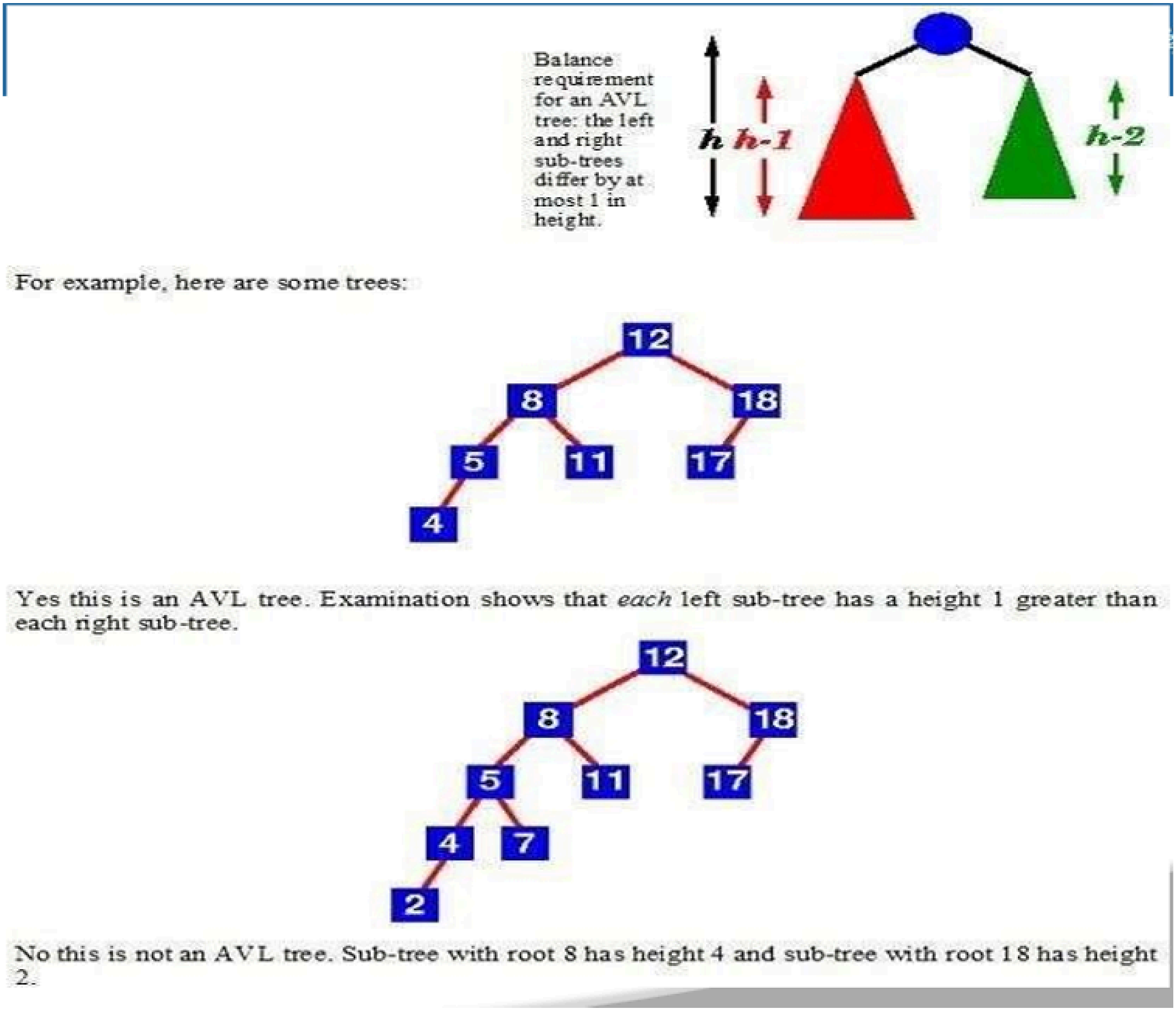
```
        50                          60
      /    \      delete(50)      /  \
     40      70     ──────→     40    70
            /  \                        \
           60   80                       80
```



Q5)Define an AVL tree and write the steps used to follow while inserting an element 3into angivenAVLtreecontainingelements 13, 10, 15, 5, 11, 16, 4, 8

A.

- Definition of an AVL tree: An AVL tree is a binary search tree which has the following properties:
  i. The sub-trees of every node differ in height by at most one.
  ii. Every sub-tree is anAVLtree.
- Balance requirement for an AVL tree: the left and right sub-trees differ by at most 1 in height.



Balance requirement for an AVL tree: the left and right sub-trees differ by at most 1 in height.

For example, here are some trees:

Yes this is an AVL tree. Examination shows that *each* left sub-tree has a height 1 greater than each right sub-tree.

No this is not an AVL tree. Sub-tree with root 8 has height 4 and sub-tree with root 18 has height 2.

To insert an element in the AVL tree, follow the following steps-

● Insert the element in the AVL tree in the same way the insertion is performed in BST.

- After insertion, check the balance factor of each node of the resulting tree. Now, following two cases are possible

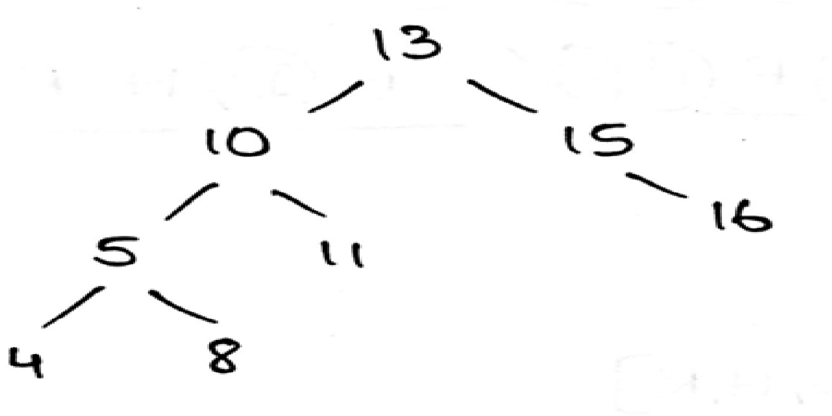
Case-01

- After the insertion, the balance factor of each node is either 0 or 1 or 1.
- In this case, the tree is considered to be balanced.
- Conclude the operation.
- Insert the next element if any.

Case-02

- After the insertion, the balance factor of at least one node is not 0 or 1 or 1.
- In this case, the tree is considered to be imbalanced.
- Perform the suitable rotation to balance the tree.
- After the tree is balanced, insert the next element if any.

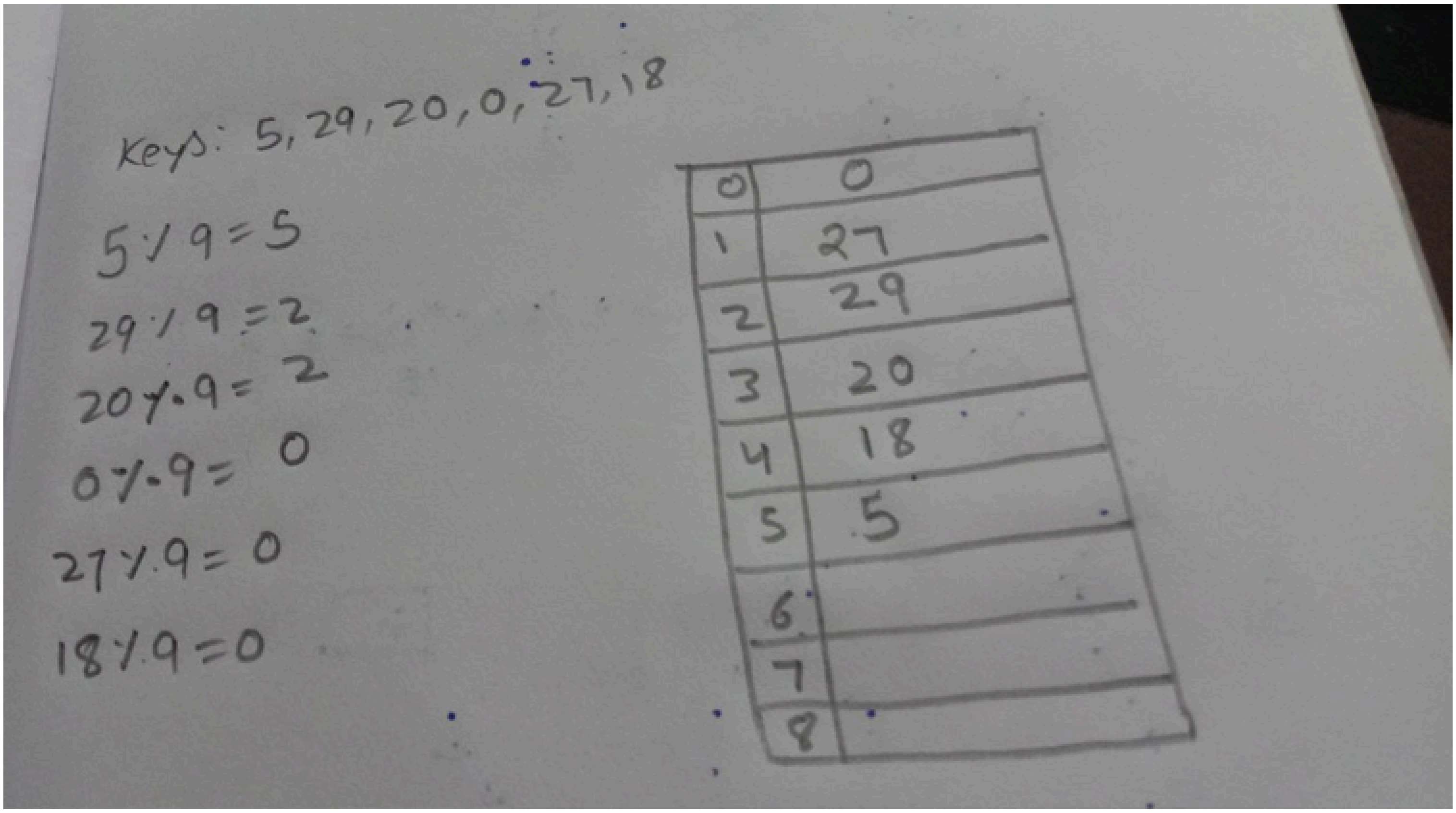
After Insertion



The difference between the heights of the left and right subtree is 1, which means there is no need of rotation for balancing the AVL Tree. Both the AVL Tree and the BST Tree in this example are identical.

Q6) Draw ahashtablewithopenaddressingandasizeof9.Usethe hash function (k mod 9. Insert the keys: 5, 29, 20, 0, 27 and 18 into the hash table (in that order).

A.



Keys: 5, 29, 20, 0, 27, 18

5%9=5

29%9=2

20%9=2

0%9=0

27%9=0

18%9=0

0	0
1	27
2	29
3	20
4	18
5	5
6	
7	
8	

Q7) Define a BTree and its properties? Construct a B-tree of minimum degree 3 from the following elements 1, 2, 3, 4,5,6,30, 40, 50, 60, 70, 80, 82, 84, 86.

A. B Tree is a specialized m-way tree that can be widely used for disk access. A BTree of order m can have at most (m-1) keys and m children. One of the main reasons for using B tree is its capability to store a large number of keys in a single node and large key values by keeping the height of the tree relatively small.

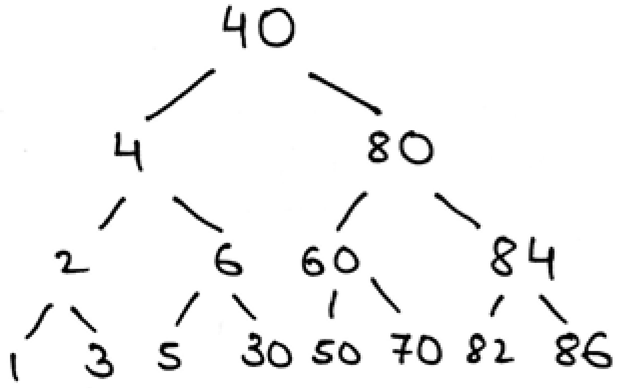
A B tree of order m contains all the properties of an M way tree. In addition, it contains the following properties.

1. Every node in a BTree contains at most m children.

2. Every node in a BTree except the root node and the leaf node contain at least m/2 children.

3. The root nodes must have at least 2 nodes.

4. All leaf nodes must be at the same level.



Q8) Write the procedure for insertion and deletion operation in a B tree with the following elements 10, 20, 30, 40, 50, 60, 70, 80, 90.

A. **Insertion in B-Tree**

1. **Insert in Leaf Node**:
   - ○ If the tree is empty, create a root node and insert the key.
   - ○ Otherwise, find the appropriate leaf node where the key should be inserted.
2. **Handle Overflow**:
   - ○ If the node has space, insert the key in sorted order.
   - ○ If the node is full, split it into two nodes:

        - Move the median key up to the parent node.
        - Split the keys into two separate nodes.
3. **Repeat**:
    - Repeat the process up the tree if necessary, splitting the parent nodes as needed.
    - If the root node is split, create a new root.

Deletion in B-Tree

1. **Delete from Leaf Node**:
    - If the key is in a leaf node, simply remove it.
2. **Delete from Internal Node**:
    - Replace the key with the predecessor or successor key from the leaf node.
    - Remove the predecessor or successor key from the leaf node.
3. **Handle Underflow**:
    - If the node has fewer keys than the minimum degree (t-1), rebalance the tree:
        - Borrow a key from a sibling.
        - Merge with a sibling if borrowing is not possible.

Example with Given Elements (Assuming Minimum Degree t = 3)

Step-by-Step Insertion:

1. **Insert 10**:
    - Tree: [10]
2. **Insert 20**:
    - Tree: [10, 20]
3. **Insert 30**:
    - Tree: [10, 20, 30]
4. **Insert 40**:
    - Tree: [10, 20, 30, 40]
5. **Insert 50**:

    - Tree: [10, 20, 30, 40, 50] (Node full, needs to split)
    - Split: 30 moves up
    - Tree:

[30]

/ \

[10,20] [40,50]

Insert 60:

- Tree:

[30]

/ \

[10,20] [40,50,60]

Insert 70:

- Tree:

[30]

/ \

[10,20] [40,50,60,70] (Node full, needs to split)

Split: 50 moves up

Tree:

[30, 50]

```
  /    |    \
[10,20] [40] [60,70]
```

Insert 80:

- Tree:

```
   [30, 50]
  /    |    \
[10,20] [40] [60,70,80]
```

Insert 90:

- Tree:

```
   [30, 50]
  /    |    \
[10,20] [40] [60,70,80,90] (Node full, needs to split)
```

Split: 70 moves up

Tree:

```
   [30, 50, 70]
  /    |    |    \
[10,20] [40] [60] [80,90]
```

Step-by-Step Deletion:

For example, to delete key 70:

1. **Delete 70**:
   - ○ 70 is in the root.
   - ○ Replace 70 with the largest key in the left subtree (60).
   - ○ Tree after deletion:

[30, 50, 60]

/ | | \

[10,20] [40] [ ] [80,90]

Q9) Explain the collision resolution techniques separate chaining and open addressing with suitable examples.

A.

LINK:https://www.gatevidyalay.com/collision-resolution-techniques-separate-chaining/

Q10) Explain the following:

i. Hashing

ii. Hash table

iii. Hash Function

A. Hashing: Hashing in the data structure is a technique of mapping a large chunk of data into small tables using a hashing function. It is also known as the message digest function. It is a technique that uniquely identifies a specific item from a collection of similar items. It uses hash tables to store the data in an array format. Each value in the array has assigned a unique index number. Hash tables use a technique to generate these unique index numbers for each value stored in an array format. This technique is called the hash technique.

The following are real-life examples of hashing in the data structure

- In schools, the teacher assigns a unique roll number to each student. Later, the teacher uses that roll number to retrieve information about that student.

- Alibrary has an infinite number of books. The librarian assigns a unique number to each book. This unique number helps in identifying the position of the books on the bookshelf.

Hash Function: The hash function in a data structure maps arbitrary size of data to fixed-sized data. It returns the following values: a small integer value (also known as hash value), hash codes, and hash sums.

hash = hash func(key)

index = hash % array_size

The has function must satisfy the following requirements:

- A good hash function is easy to compute.

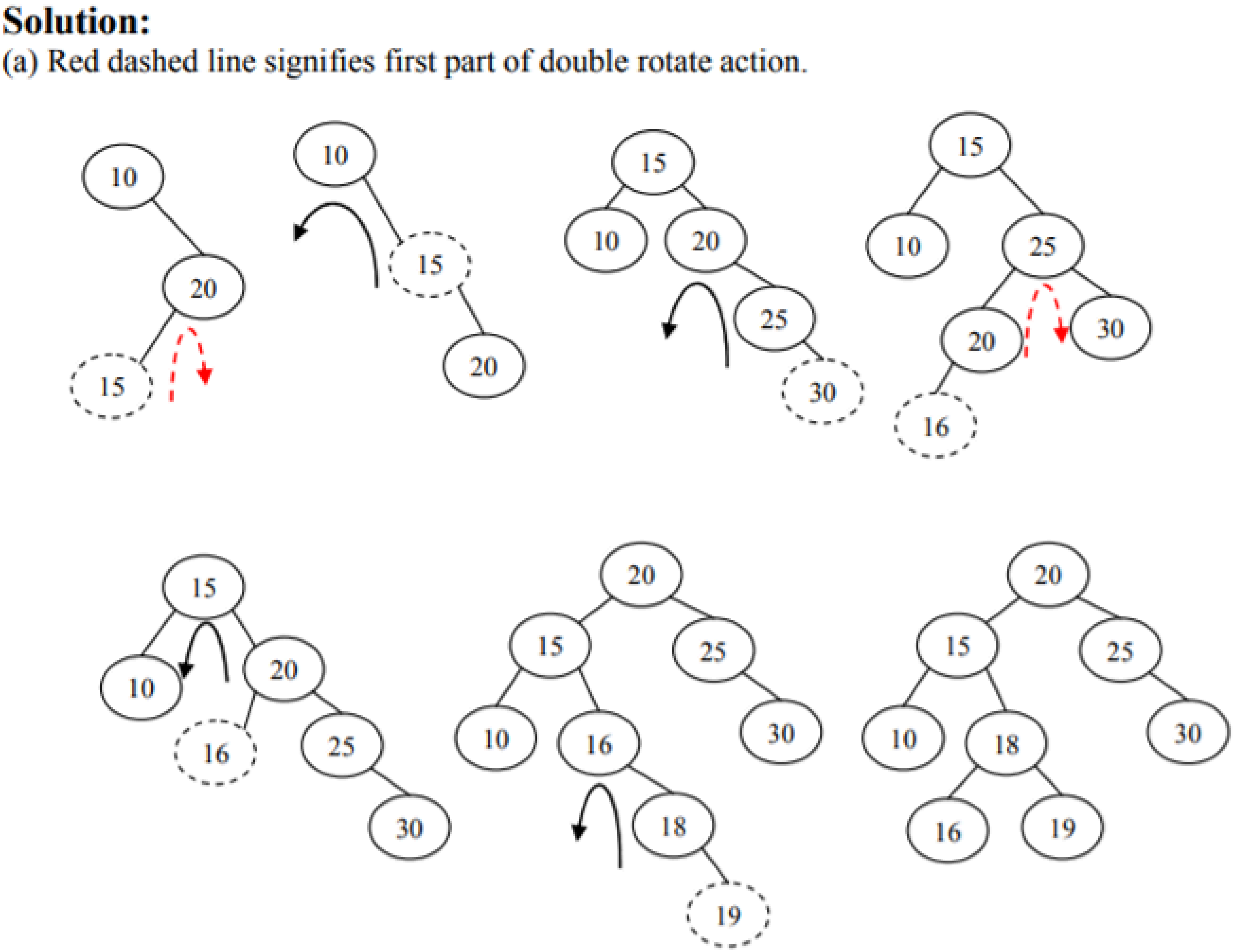
- A good hash function never gets stuck in clustering and distributes keys evenly across the hash table.

- A good hash function avoids collision when two elements or items get assigned to the same hash value.

Hash Table: Hashing in data structure uses hash tables to store the key-value pairs. The hash table then uses the hash function to generate an index. Hashing uses this unique index to perform insert, update, and search operations.

Q11) Insert the following sequence of elements into an AVL tree, starting with an empty tree: 10, 20, 15, 25, 30, 16, 18, 19 and delete 30 in the AVL tree that you got.

A.



Solution:

(a) Red dashed line signifies first part of double rotate action.

Q12) Explain the collision resolution technique double hashing and linear probing with suitable examples?

A. Double hashing: It is a technique used for avoiding collisions in hash tables. A collision occurs when two keys are hashed to the same index in a hash table. Collisions are a problem because every slot in a hash table is supposed to store a single element.

Let us consider the following keys, keys 10 ,90 ,30 ,40 ,50

Let us consider the following double hash function.

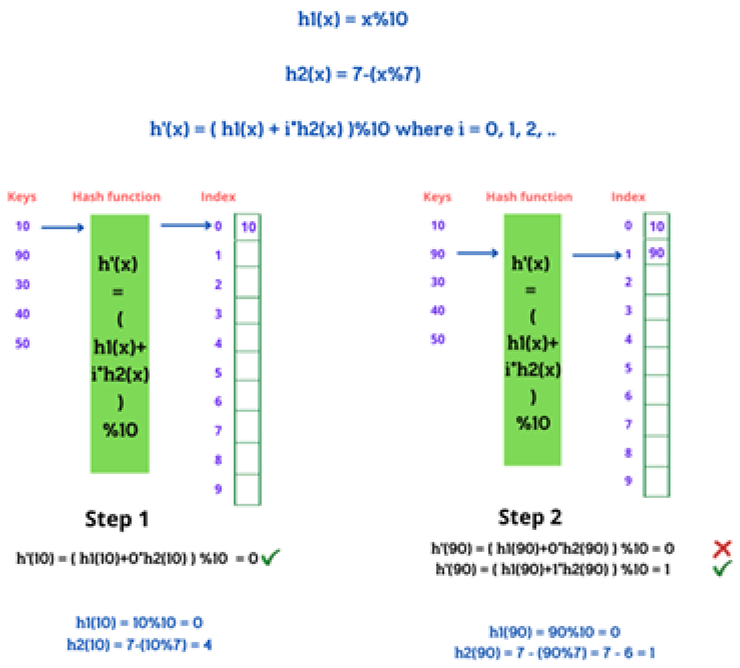
$h'(x) = (h1\ (x) + i * h2\ (x))\ \%\ hashsiz$�

let us take a hash table of size 10

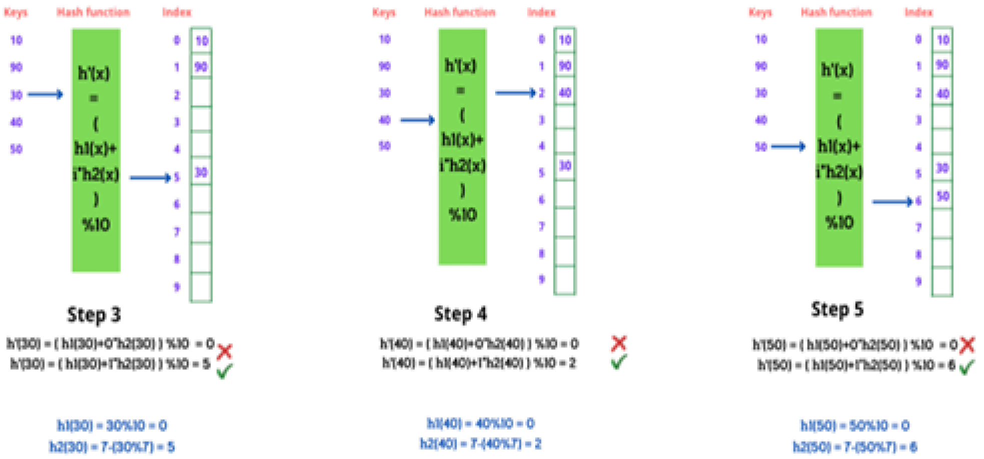
and let us consider h1 (x) = x%10, this hash function gives us the output in range of 0 to 9. You can also choose any other hash function here.

Let us take h2 (x) = 7– (x%7) where 7 is a prime number less than 10. And also, this hash function doesn’t give 0 as output and probes all other locations of the hash table.

Now let us try to insert the given keys into the hash table using the modified hash function h'(x).


h1(x) = x%10
h2(x) = 7-(x%7)
h'(x) = (h1(x) + i*h2(x))%10 where i = 0, 1, 2, ..
Keys
Hash function
Index
10
90
30
40
50
h'(x) = (h1(x)+ i*h2(x)) %10
0 1 2 3 4 5 6 7 8 9
10
Step 1
h'(10) = (h1(10)+0*h2(10)) %10 = 0 ✓
h1(10) = 10%10 = 0
h2(10) = 7-(10%7) = 4
Keys
Hash function
Index
10
90
30
40
50
h'(x) = (h1(x)+ i*h2(x)) %10
0 1 2 3 4 5 6 7 8 9
10
90
Step 2
h'(90) = (h1(90)+0*h2(90)) %10 = 0 ✗
h'(90) = (h1(90)+1*h2(90)) %10 = 1 ✓
h1(90) = 90%10 = 0
h2(90) = 7 - (90%7) = 7 - 6 = 1

As you can see here in step 2, we have encountered a collision. Then you should increment the value of i and find the new index to insert the collided key using a modified hash function.


Keys
Hash function
Index
10
90
30
40
50
h'(x) = (h1(x)+ i*h2(x)) %10
0
1
2
3
4
5
6
7
8
9
10
90
30
Step 3
h'(30) = (h1(30)+0*h2(30)) %10 = 0 ✗
h'(30) = (h1(30)+1*h2(30)) %10 = 5 ✓
h1(30) = 30%10 = 0
h2(30) = 7-(30%7) = 5
Keys
Hash function
Index
10
90
30
40
50
h'(x) = (h1(x)+ i*h2(x)) %10
0
1
2
3
4
5
6
7
8
9
10
90
40
30
Step 4
h'(40) = (h1(40)+0*h2(40)) %10 = 0 ✗
h'(40) = (h1(40)+1*h2(40)) %10 = 2 ✓
h1(40) = 40%10 = 0
h2(40) = 7-(40%7) = 2
Keys
Hash function
Index
10
90
30
40
50
h'(x) = (h1(x)+ i*h2(x)) %10
0
1
2
3
4
5
6
7
8
9
10
90
40
30
50
Step 5
h'(50) = (h1(50)+0*h2(50)) %10 = 0 ✗
h'(50) = (h1(50)+1*h2(50)) %10 = 6 ✓
h1(50) = 50%10 = 0
h2(50) = 7-(50%7) = 6

This is how you have to insert the keys into a hash table while using the technique of double hashing in data structure.

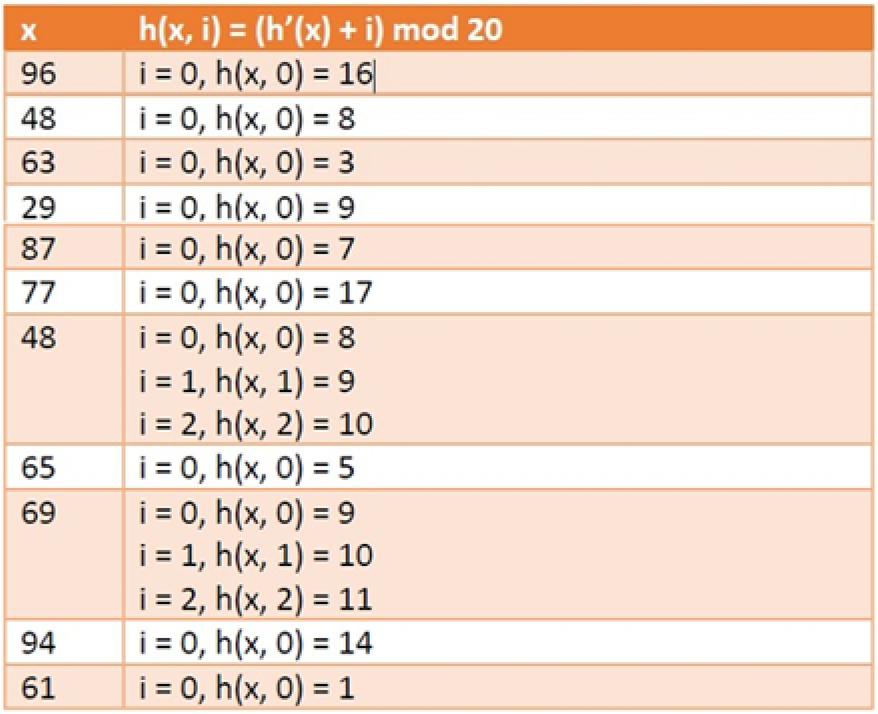
Linear probing: There is an ordinary hash function h´(x): U → 0, 1, . . ., m– 1. In an open addressing scheme, the actual hash function h(x) is taking the ordinary hash function h’(x) and attaching some other part with it to make one linear equation.

h´(x) = $x\ mod\ m$

$h\ (x, i)$ = $(h´(x)$ + $i)\ mod\ m$

The value of i| = 0, 1, . . ., m– 1. So, we start from i = 0, and increase this until we get one Free Space. So initially when i = 0, then the h (x, i) is the same as h´(x).

Example Suppose we have a list of size 20 (m = 20. We want to put some elements in linear probing fashion. The elements are {96, 48, 63, 29, 87, 77, 48, 65, 69, 94, 61}



x	h(x, i) = (h′(x) + i) mod 20
96	i = 0, h(x, 0) = 16
48	i = 0, h(x, 0) = 8
63	i = 0, h(x, 0) = 3
29	i = 0, h(x, 0) = 9
87	i = 0, h(x, 0) = 7
77	i = 0, h(x, 0) = 17

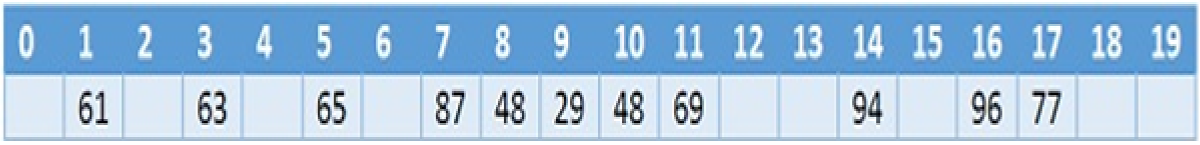
48	i = 0, h(x, 0) = 8
	i = 1, h(x, 1) = 9
	i = 2, h(x, 2) = 10

65	i = 0, h(x, 0) = 5

69	i = 0, h(x, 0) = 9
	i = 1, h(x, 1) = 10
	i = 2, h(x, 2) = 11

94	i = 0, h(x, 0) = 14
61	i = 0, h(x, 0) = 1

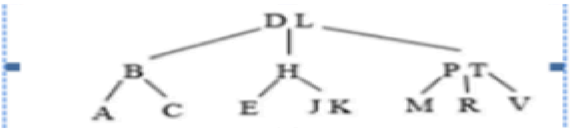
Hash Table

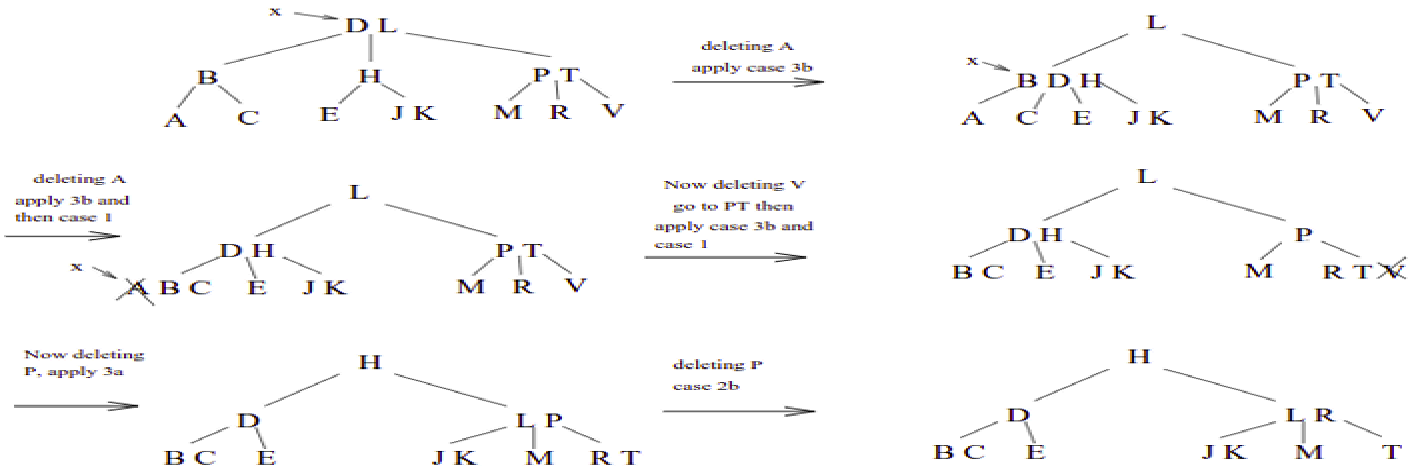


0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
	61		63		65		87	48	29	48	69			94		96	77		

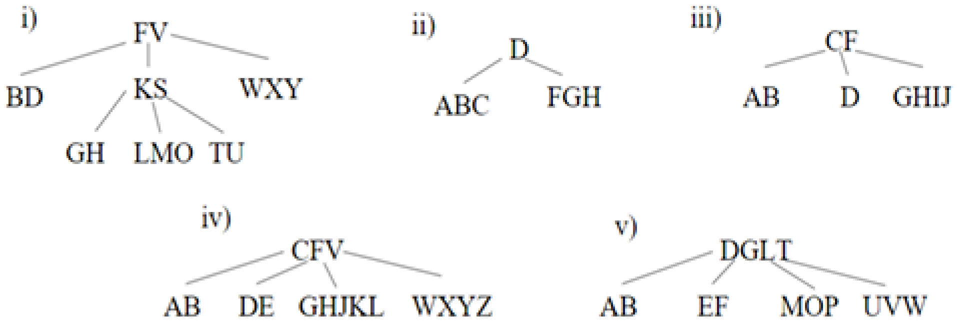
Q13) Show the B-tree the results when deleting A, then deleting V and then deleting P from the following B-tree with a minimum branching factor of t=2

A.





Q14) Which of the following are legal B-trees for when the minimum branching factor t = 3? For those that are nolegal,give one or two sentences very clearly explaining what property was violated?



A. i): Not legal since the height is not balanced. More specifically, both the nodes with “BD” and “KS” are at the same level but “BD” is a leaf and “KS” is not.

(ii): This is legal. Remember, that the root can have just a single key.

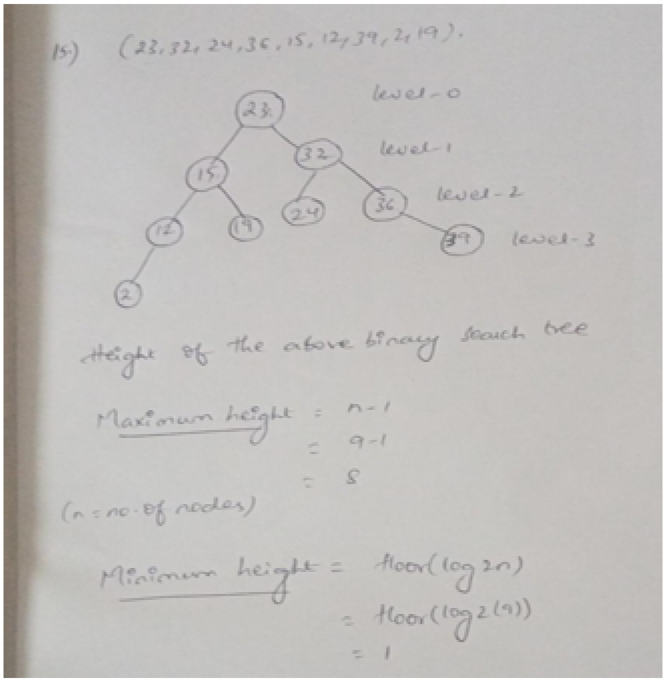
(iii): Not legal– the key “D” has less than the minimum allowable size of 2 keys.

(iv): This is legal.

(v): Not legal– there’s no leaf node corresponding to the keys between G and L.

Q15) Create a binary search tree for the following elements 23, 32, 24, 36, 15, 12, 39, 2, 19. Discuss about the height of the above binary search tree.

A.



23
32
15
24
36
12
19
39
2
level-0
level-1
level-2
level-3

15) (23, 32, 24, 36, 15, 12, 39, 2, 19).

Height of the above binary search tree

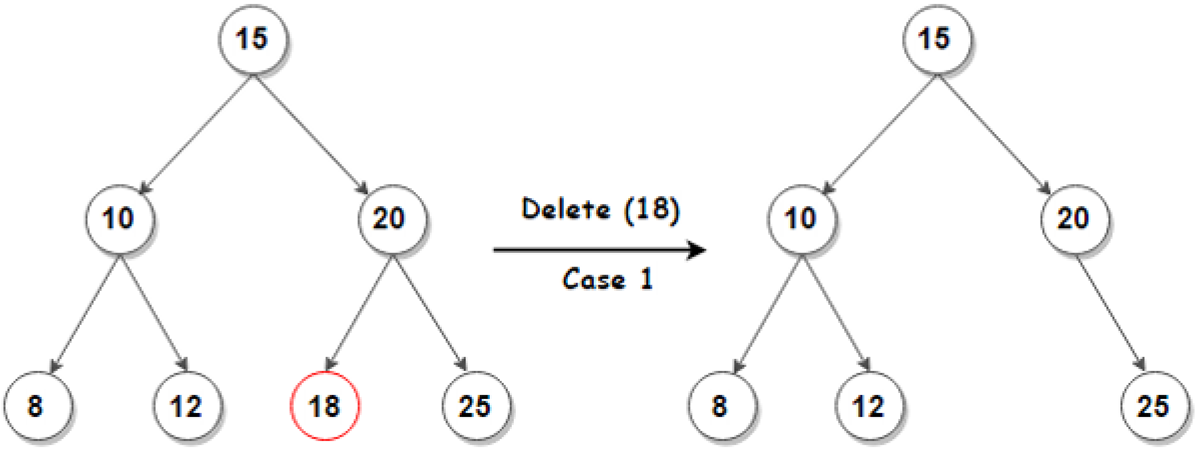
Maximum height : n-1
= 9-1
= 8

(n = no. of nodes)

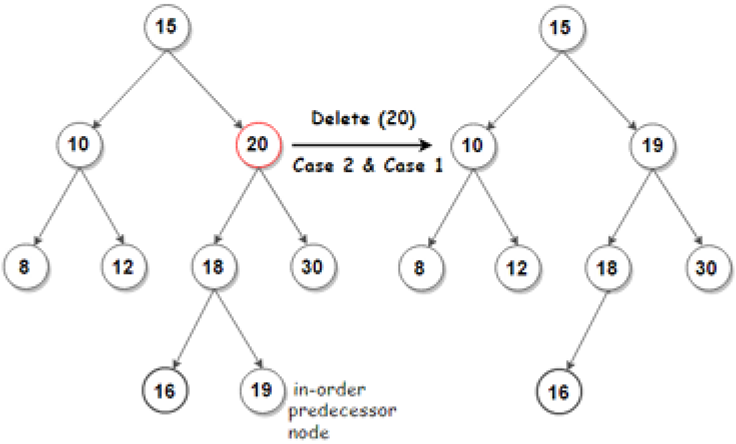
Minimum height = floor(log 2n)
= floor(log2(9))
= 1

Q16)Explain with examples different cases of deletion of elements in a binary search tree?

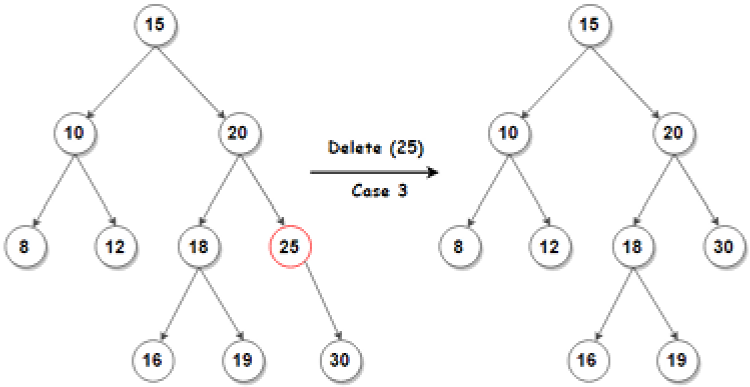
A. There are three possible cases to consider deleting a node from BST:
Case :1 Deleting a node with no children: remove the node from the tree.



Case 2 :Deleting a node with two children: call the node to be deleted N. Do not delete N. Instead, choose either its in order successor node or its in order predecessor node, R. Copy the value of R to N, then recursively call delete on R until reaching one of the first two cases. If we choose the in-order successor of a node, as the right subtree is not NULL (our present case is a node with 2 children), then it's in order successor is a node with the least value in its right subtree, which will have at a maximum of 1 subtree, so deleting it would fall in one of the first 2 cases.



Case 3 Deleting a node with one child: remove the node and replace it with its child.



Q17)Explain how M-way search trees differ from binary search trees with an example

A. An m-way search tree is a m-way tree in which: Each node has m children and m-1 key fields The keys in each node are in ascending order. The keys in the first i children are smaller than the I th key the keys in the last m-i children are larger than the ith key

An extension of a multiway search tree of order m is a B-tree of order m. This type of tree will be used when the data to be accessed/stored is located on secondary storage devices because they allow for large amounts of data to be stored in a node.

A B-tree of order m is a multiway search tree in which:

The root has at least two subtrees unless it is the only node in the tree. Each non root and each non leaf node has at most m nonempty children and at least m/2 nonempty children. The number of keys in each non root and each non leaf node is one less than the number of its nonempty children. All leaves are on the same level.

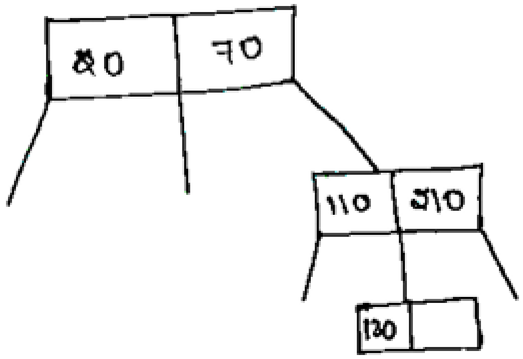
Q18) Construct a M-way search tree of order 3 for the following nodes 20,70,110,210,130

A.

(18) order = 3

keys = 3-1 = 2

Given nodes : 20,70,110,210,130



Q19) Considerahashtablewith 5slots.Thehashfunctionis h(k)=kmod 5.The Collisions Are Resolved By chaining.#the following keys reinserted in order: 5,28,19,15,20,33, 12,17,10.Findthe maximum,minimum and average chain length in hashtable?

A. **Hash Function**

The hash function is h(k)=kmod 5h(k) = k \mod 5, which means we divide the key by 5 and take the remainder as the hash value.

Insert Keys into the Hash Table

We will insert the keys one by one and calculate their hash values:

1. **Key 5**:

h(5)=5 mod 5=0 h(5) = 5 \mod 5 = 0

Insert into slot 0.

2. **Key 28**:

h(28)=28 mod 5=3h(28) = 28 \mod 5 = 3

Insert into slot 3.

3. **Key 19**:

h(19)=19 mod 5=4 h(19) = 19 \mod 5 = 4

Insert into slot 4.

4. **Key 15**:

h(15)=15 mod 5=0 h(15) = 15 \mod 5 = 0

Insert into slot 0 (chain with 5).

5. **Key 20**:

h(20)=20 mod 5=0 h(20) = 20 \mod 5 = 0

Insert into slot 0 (chain with 5, 15).

6. **Key 33**:

h(33)=33 mod 5=3h(33) = 33 \mod 5 = 3

Insert into slot 3 (chain with 28).

7. **Key 12**:

$h(12)=12 \bmod 5=2$h(12) = 12 \mod 5 = 2

Insert into slot 2.

8. **Key 17**:

$h(17)=17 \bmod 5=2$h(17) = 17 \mod 5 = 2

Insert into slot 2 (chain with 12).

9. **Key 10**:

$h(10)=10 \bmod 5=0$ h(10) = 10 \mod 5 = 0

Insert into slot 0 (chain with 5, 15, 20).

Hash Table Structure

The hash table with chaining will look like this:

- Slot 0: 5 -> 15 -> 20 -> 10
- Slot 1: (empty)
- Slot 2: 12 -> 17
- Slot 3: 28 -> 33
- Slot 4: 19

Chain Lengths

- Slot 0: 4 (5 -> 15 -> 20 -> 10)
- Slot 1: 0 (empty)
- Slot 2: 2 (12 -> 17)
- Slot 3: 2 (28 -> 33)
- Slot 4: 1 (19)

Maximum, Minimum, and Average Chain Length

- **Maximum Chain Length**: 4 (slot 0)
- **Minimum Chain Length**: 0 (slot 1)

- **Average Chain Length**:

Total number of elements=9\text{Total number of elements} = 9

Number of slots=5\text{Number of slots} = 5

Average Chain Length=Total number of elementsNumber of slots=95=1.8\text{Average Chain Length} = \frac{\text{Total number of elements}}{\text{Number of slots}} = \frac{9}{5} = 1.8

Summary

- **Maximum Chain Length**: 4
- **Minimum Chain Length**: 0
- **Average Chain Length**: 1.8

Q20) A hashtable contains15 buckets and uses linear probing to resolve collisions. Thekeyvaluesareintegers andhashfunctionusedis key%15. If The Values 43, 165,62,123,142are inserted in the table,then find the location of the key value 142inthetable?

A. To determine the location of the key value 142 in the hash table using linear probing, we'll follow these steps:

1. **Calculate the Hash Values**: The hash function used is keymod 15\text{key} \mod 15.
2. **Insert Each Key**: If a collision occurs (i.e., the slot is already occupied), we use linear probing to find the next available slot.

Step-by-Step Insertion

1. **Insert 43**:

hash(43)=43 mod 15=13\text{hash}(43) = 43 \mod 15 = 13

Insert 43 at index 13.

2. **Insert 165**:

hash(165)=165 mod 15=0\text{hash}(165) = 165 \mod 15 = 0

Insert 165 at index 0.

3. **Insert 62**:

hash(62)=62 mod 15=2\text{hash}(62) = 62 \mod 15 = 2

Insert 62 at index 2.

4. **Insert 123**:

hash(123)=123 mod 15=3\text{hash}(123) = 123 \mod 15 = 3

Insert 123 at index 3.

5. **Insert 142**:

hash(142)=142mod 15=7\text{hash}(142) = 142 \mod 15 = 7

Insert 142 at index 7.