***BSc.(Information Technology) (Semester III) 2019-20***

***Data Structures***

***(USIT 302 Core)***

***Paper Solution***

***By***

***Mr. Shajil Kumar P A***

***Question 1***

***Q1a. What is an algorithm? What are the characteristics of an algorithm?***

***Ans:*** Algorithm is a step-by-step procedure, which defines a set of instructions to be executed in a certain order to get the desired output. Algorithms are generally created independent of underlying languages, i.e. an algorithm can be implemented in more than one programming language.

***Characteristics of an Algorithm***

An algorithm should have the following characteristics −

1. Unambiguous − Algorithm should be clear and unambiguous. Each of its steps (or phases), and their inputs/outputs should be clear and must lead to only one meaning.
2. Input − An algorithm should have 0 or more well-defined inputs.
3. Output − An algorithm should have 1 or more well-defined outputs, and should match the desired output.
4. Finiteness − Algorithms must terminate after a finite number of steps.
5. Feasibility − Should be feasible with the available resources.
6. Independent − An algorithm should have step-by-step directions, which should be independent of any programming code.

***Q1b. What are the advantages and limitations of an array***

***Ans:* Advantages of an Array**

1. It is better and convenient way of storing the data of same datatype with same size.
2. It allows us to store known number of elements in it.
3. It allocates memory in contiguous memory locations for its elements. It does not allocate any extra space/ memory for its elements. Hence there is no memory overflow or shortage of memory in arrays.
4. Iterating the arrays using their index is faster compared to any other methods like linked list etc.
5. It allows to store the elements in any dimensional array - supports multidimensional array.

**Limitations of an Array**

1. It allows us to enter only fixed number of elements into it. We cannot alter the size of the array once array is declared. Hence if we need to insert more number of records than declared then it is not possible. We should know array size at the compile time itself.
2. Inserting and deleting the records from the array would be costly since we add / delete the elements from the array, we need to manage memory space too.
3. It does not verify the indexes while compiling the array. In case there is any indexes pointed which is more than the dimension specified, then we will get run time errors rather than identifying them at compile time.

***Q 1c. Explain with the help of an example how to merge two sorted arrays.***

***Ans:*** There are two different ways to merge the given sorted arrays. In the first approach the given arrays are combined end to end and then elements of the third combined array can be sorted by using some appropriate sorting algorithm. In the second approach the third array will be sorted while merging the given sorted arrays. In this approach the elements of the given arrays are compared and based on the comparison it is decided which element will go into the third array

Consider two arrays A1 and A2 as shown below

|  |  |
| --- | --- |
| 10 | 15 |

|  |  |  |
| --- | --- | --- |
| 5 | 12 | 14 |

Step 1:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 5 |  |  |  |  |

Step 2:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 5 | 10 |  |  |  |

Step 3:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 5 | 10 | 12 |  |  |

Step 4:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 5 | 10 | 12 | 14 |  |

Step 5:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 5 | 10 | 12 | 14 | 15 |

***Q1d. Write an algorithm for binary search in an array***

***Ans:*** Step 1: Set Start = lb , End =ub

Step 2: Repeat Steps 3 to 5 while Start < End

Step 3: Set Middle = Integer ((start + end)/2)

Step 4: If S[middle] =Data then

Print: “Element is found at position”: Middle

Exit

[End If]

Step 5: If S[Middle]< Data Then

Set Start =Middle + 1

Else

Set End = Middle -1

[End If]

[End Loop]

Step 6: Print: “Element does not exist in the array”

Step 7: Exit

***Q1e.What is bubble sort? Sort the following elements using bubble sort method 14, 33, 27, 35, 10***

***Ans:*** Bubble sort is a simple sorting algorithm. This sorting algorithm is comparison-based algorithm in which each pair of adjacent elements is compared and the elements are swapped if they are not in order. This algorithm is not suitable for large data sets as its average and worst case complexity are of Ο(n2) where n is the number of items.

Pass 1

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 14 | 33 | 27 | 35 | 10 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 14 | 27 | 33 | 35 | 10 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 14 | 27 | 33 | 10 | 35 |

Pass 2

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 14 | 27 | 10 | 33 | 35 |
| 14 | 10 | 27 | 33 | 35 |

Pass 3

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 10 | 14 | 27 | 33 | 35 |

***Q1f.Explain different methods of memory representation of sparse matrix***

***Ans:*** Method of Linearization – In this technique the non-zero elements of the matrix

are stored in one dimensional array. For example

|  |  |
| --- | --- |
| 1 | 7 |

Method Of Vector Representation – In this technique the non-zero elements are stored in the array along with its row id and column id

|  |  |  |  |
| --- | --- | --- | --- |
|  | Row-ID | Column-ID | Elements |
| V[1] | 2 | 2 | 1 |
| V[2] | 3 | 3 | 7 |

***Question 2***

***Q2a. Write and explain an algorithm to insert a node at the end of a single linked list***

***Ans:*** **Algorithm**

Step : If free = Null Then

Print: “Overflow: No free space available for insertion”

Exit

[End If]

Step 2: Allocate space to node New

(Set New = Free And Free= Free ---> Next)

Step 3: Set New ---> Info = Data, New ---> Next =Null

Step 4: If Begin = Null Then

Begin = New

Exit

[End If]

Step 5: Set pointer = Begin

Step 6: Repeat While Pointer ---> Next != Null

Set Pointer = Pointer ---> Next

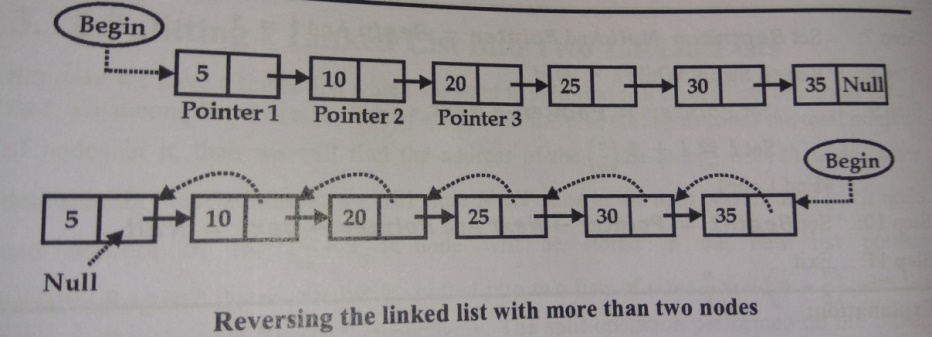
[End Loop]

Step 7: Set Pointer ---> Next = New

Step 8: Exit

***Q2b. Explain algorithmically reversing a single linked list.***

***Ans:*** To reverse a linked list we need to use three pointer variables One pointer variable is used to store the address of current node, second pointer variable is used to store the address of the next node and the third pointer variable is used to store the address of next to next of current node



**Algorithm:**

Step 1: If Begin = Null

Print : “No Node is present in the linked list ”

Exit

[End If]

Step 2: If Begin --> Next = Null Then

Print : “ Linked List is having only one node”

Exit

[End If]

Step 3: If Begin --> Next != Null Then

Set Pointer1 = Begin And Set Pointer2 = Begin --> Next

Set Pointer3 = Pointer2 --> Next

[End If]

Step 4: If Pointer3 = Null Then

Set Pointer2 --> Next = Pointer1 And Set Pointer1 --> Next = Null And Set Begin = Pointer2

Exit

[End If]

Step 5: Set Pointer1 --> Next = Null

Step 6: Repeat Step 7 to 10 while Pointer3 --> Next != Null

Step 7: Set Pointer2 --> Next = Pointer1 And Set Pointer1 = Pointer2

Step 8: Set Pointer2 = Pointer3

Step 9: Set Pointer3 = Pointer3 --> Next

[End Loop]

Step 10: Set Pointer2 --> Next = Pointer1

Step 11: Set Pointer3 --> Next = Pointer2

Step 12: Set Begin = Pointer3

Step 13: Exit

***Q2c. Write an algorithm to delete a node from the end of doubly linked list.***

***Ans:*** **Algorithm**

Step 1: If Begin = Null

Print: “Linked List is already empty”.

Exit

[End If]

Step 2: Set Pointer = Begin --> Next

Step 3: Repeat While Pointer --> Next != Null And Pointer --> Info != Item

Set Pointer = Pointer --> Next

[End Loop]

Step 4:If Pointer --> Next = Null Then

Set Previous = Pointer --> Pre And Previous --> Next = Null

Set End = Previous

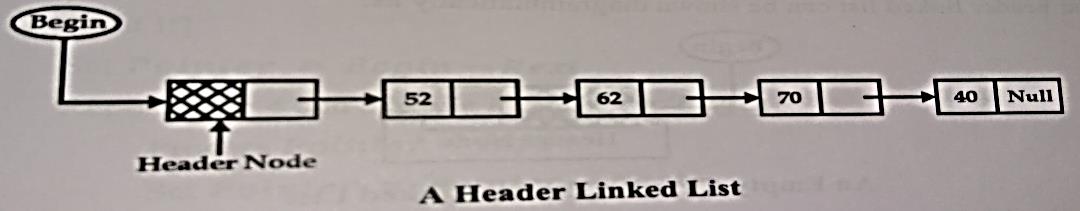
Step 5: Pointer --> Next = Free , Free = Pointer

Step 6: Exit

***Q2d. Explain different categories of header linked list.***

***Ans:*** A header linked list is a special kind of linked list which contains a special node at the beginning of the list, known as head node.

The head node contains information regarding the linked list. This information may be **total number of nodes** in the linked list, some description for the user like **creation date, modification date**, whether the data in the list is **sorted or unsorted**.



Different categories of Header Linked List are

**1. Grounded Header Linked List** – It is a list in which last node of the list contains the **Null** in its **Next** pointer field. If grounded header linked list is empty then the Null value will be stored in the Next pointer field of the head node

**2. Circular Header Linked List** – It is a list in which the last node of the list points back to the header node i.e. **Next** pointer field of the last node contains the address of the header node. If circular header linked list is empty then address of the head node is stored in the Next pointer field of the head node itself

**3. Two-way Header Linked List** – It is a list in which the **Pre** part of the header node contains the address of the first node and the **Next** part contains the address of the last node

**4. Two-way Circular Header Linked List** – It is a list in which the **Pre** part of the header node contains the address of the first node and the **Pre** part of the first node contains the address of the header node. The **Next** part of the header node contains the address of the last node and the **Next** part of the last node contains the address of the header node

***Q2e. Explain how memory is allocated and deallocated for linked list***

**Ans**:

* To insert an element in the linked list the first requirement is to get a free Node.

The task of obtaining an empty Node for insertion and returning free Node after deletion is accomplished by maintaining a separate list of free Nodes that begin with pointer Free that points to the first available free Node. Whenever a new Node is to be inserted in to the linked list then free memory is checked for the availability. If the Node is available then the node is added to the linked list and pointer Free will point to the next available node.

* To delete a Node from a linked list it is desirable to return the memory taken by deleted Node for its reusability.
* If there is no free node then it is indicated by Free = Null and this condition is known as Overflow.
* Whenever a node is deleted from the linked list it is inserted as the first node of the free storage. The pointer Free will point to the recently added node

***Q2f. Write an algorithm to copy one linked list into another linked list***

***Ans***: **Algorithm**

Step 1: If Begin1 = Null Then

Print : “Linked List is empty”

Exit

[End If]

Step 2: Begin2 = Null

Step 3: If Free = Null Then

Print: “Free space is not available”

Exit

Else

Set New = Free and Free = Free --> Next

[End If]

Step 4: Set New --> Info = Begin1 --> Info and New --> Next = Null

Step 5: Set Begin2 = New

Step 6: Set Pointer1 = Begin1 --> Next And Pointer2 = Begin2

Step 7: Repeat While Pointer1 ! = Null And Free != Null

Set New = Free

Set New --> Info = Pointer1 --> Info And New -->Next = Null

Set Pointer2 🡪 Next = New

Set Pointer1 = Pointer1 -- > Next And Pointer2 = New

[End Loop]

Step 8: If Pointer1 = Null Then

Print : “List Copied Successfully”

Exit

Else

Print : “No Enough space available to perform Copy Operations”:

[End If]

Step 9: Exit

***Question 3***

***Q3a. Write an algorithm to perform Push and Pop operation in stack data structure.***

***Ans:***

**Algorithm:**

Push Operation – Insert new element Data at the top of the stack represented by an array S of size MAX with stack index variable Top pointing to the topmost element of the stack

Step 1: If Top = MAX Then

Print “Stack is full Overflow condition”

Exit

[End If]

Step 2: Set Top = Top + 1

Step 3: Set S[Top] = Data

Step 4: Exit

**Algorithm:**

Pop Operation – Deleting an element from the stack represented by an array S and returns the element Data which is at the top of the stack

Step 1: If Top = Null Then

Print “Stack is empty Underflow condition”

Exit

[End If]

Step 2: Set Data = S[Top]

Step 3: Set Top = Top - 1

Step 4: Exit

***Q3b. Convert the given expression in postfix and prefix notation.***

**Iin = (x - y) \* ((z + y) / f)**

***Ans:***

Prefix Notations

Iin = (x - y) \* ((z + y) / f)

=[-xy] \* ([+zy]/f)

=[-xy] \* [/+zyf]

Ipre = \*-xy/+zyf

Postfix Notations

Iin = (x - y) \* ((z + y) / f)

= [xy-] \* ([zy+]/f)

= [xy-] \* [zy+f/]

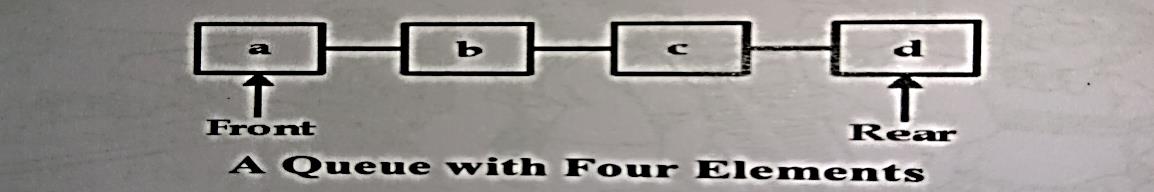
Ipost = xy-zy+f/\*

***Q3c. Define Queue. How queue is represented in memory using linked list?***

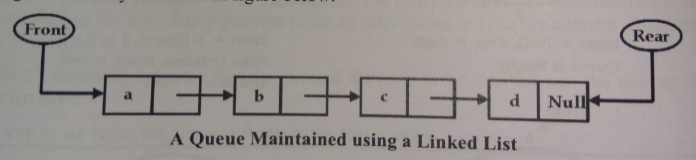
***Ans:***

A Queue is a linear structure which follows a particular order in which the operations are performed. The order is First In First Out (FIFO). A good example of a queue is any queue of consumers for a resource where the consumer that came first is served first.

Array representation of queue



* Linked list is an alternative and efficient way of representing queue .
* Two pointer variables **Front** and **Rear** contain the addresses of the element at the front and rear end of the queue.
* Initially when there is no is no element in the queue, both pointers **Front** and **Rear** will have **Null** indicating and empty queue.



***Q3d. Evaluate the given postfix expression P = 6 2 + 5 \* 8 4 / - using stack.***

***Ans:*** Consider the expression P = 6 2 + 5 \* 8 4 / - which will be evaluated as shown below

|  |  |
| --- | --- |
| **Character Scanned** | **Status Of Stack** |
| 6 | 6 |
| 2 | 2 |
| + | 8 |
| 5 | 5 |
| \* | 40 |
| 8 | 8 |
| 4 | 40 8 4 |
| / | 2 |
| - | 38 |

***Q3e. Define Recursion with suitable example.***

***Ans:***

The process in which a function calls itself directly or indirectly is called recursion and the corresponding function is called as recursive function. Using recursive algorithm, certain problems can be solved quite easily.

**Algorithm**: Calculate the value of n! recursively

int fact(int n)

{

if (n < = 1) // base case

return 1;

else

return n\*fact(n-1);

}

***Q3f Write a short note on Double Ended Priority Queue***

***Ans:*** A double-ended priority queue is a queue in which one can process the lowest priority element as well as the highest priority element. Double-ended priority queue enables the users to process any element depending upon the requirement.

To implement such a double-ended priority queue following listed methods can be used.

* Double ended priority queue using linked list
* Double ended priority using multiple queues
* Double ended priority queue using heap structure

In the linked list representation of double-ended priority queue each node of the linked list is divided into four parts

Pre Info Priority Next

* Pre Part – Holds the address of the previous node of the linked list
* Info Part – Holds the element of the queue
* Priority Part – Holds the priority number of the element

Next Part – Holds the address of the next node of the linked list

***Question 4***

***Q4a Write an algorithm for Merge Sort.***

**Ans: Algorithm**

**MergeSorting (Lower, Upper)**

Step 1: If Upper > Lower Then

Step 2: Set Mid = (Lower + Upper) / 2

Step 3: Call MergeSorting (Lower, Mid)

Step 4: Call MergeSorting (Mid + 1, Upper)

Step 5: Call MergeLists (Lower, Mid, Upper)

[End If]

Step 6: Return

**MergeLists (Lower, Mid, Upper)**

Step 1: Set Lb1 = Lower, Lb2 = Mid + 1, Ub1 = Mid, Ub2 = Upper, k = 1

Step 2: Repeat Step 3 While Lb1 < Ub1 And Lb2 < Ub2

Step 3: If A[Lb1] < A[Lb2] Then

Set B[k] = A[Lb1]

Set Lb1 = Lb1 + 1

Set k = k + 1

Else

Set B[k] = A[Lb2]

Set Lb2 = Lb2 + 1

Set k = k + 1

[End If]

[End Loop]

Step 4: If Lb1 > Ub1 Then

While Lb2 <=Ub2

Set B[k] = A[Lb2]

Set Lb2 = Lb2 + 1

Set k = k + 1

[End Loop]

Else

While Lb1 < Ub1

Set B[k] = A[Lb1]

Set Lb1 = Lb1 + 1

Set k = k + 1

[End Loop]

[End If]

Step 5: Repeat For k = Lower To Upper

Set A[k] = B[k]

[End Loop]

Step 6: Return

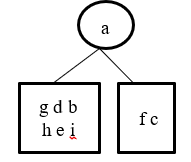
***Q4b* Reconstruct the binary tree whose in-order and pre-order traversals are:**

**In-order Traversal : g d b h e I a f c**

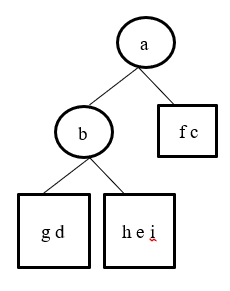
**Pre-order Traversal: a b d g e h I c f**

**Ans:**

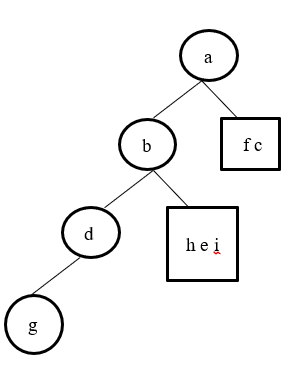
**1)**

****

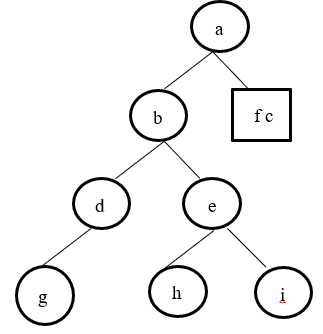
**2)**

****

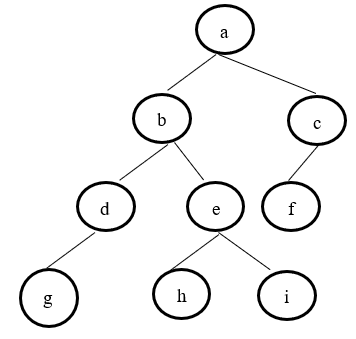
**3)**

****

**4)**

****

**5)**

****

***Q4c Explain the following terms i) Edge ii) Height of tree iii) Degree of a node iv) Path v) Internal nodes and External nodes***

**Ans:**

**Edge**

* The connecting link between any two nodes is called as an edge.
* In a tree with n number of nodes, there are exactly (n-1) number of edges.

**Height of tree**

* Total number of edges that lies on the longest path from any leaf node to a particular node is called as height of that node.
* Height of a tree is the height of root node.
* Height of all leaf nodes = 0

**Degree of a node**

* The degree of node can be defined as number of child nodes it has. A leaf node always has degree zero.

**Path**

* A sequence of distinct nodes in which successive nodes are connected by the edges. There must be exactly one path between the root and any other node in the tree.

**Internal nodes and External nodes**

* A node having no child is called a leaf node or terminal node. Sometimes terminal nodes are referred as external nodes and non-terminal nodes as internal nodes

***Q4d Write a short note on Threaded binary tree***

***Ans:***

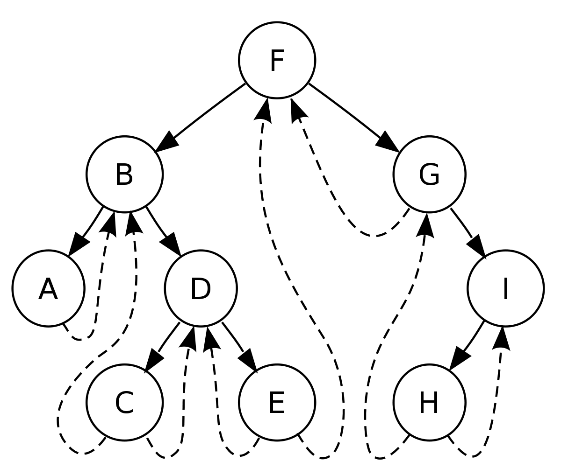
Threaded Binary Tree

Threaded binary trees is to make inorder traversal faster and do it without stack and without recursion. A binary tree is made threaded by making all right child pointers that would normally be NULL point to the inorder successor of the node (if it exists).

There are two types of threaded binary trees.

1. Single Threaded: Where a NULL right pointers is made to point to the inorder successor (if successor exists)
2. Double Threaded: Where both left and right NULL pointers are made to point to inorder predecessor and inorder successor respectively. The predecessor threads are useful for reverse inorder traversal and postorder traversal.
3. The threads are also useful for fast accessing ancestors of a node.

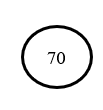
E.g –



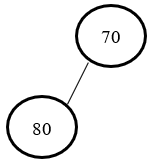
***Q4e Draw the min heap with the following elements 70 80 50 45 95 25 30 100 90 85 15 10***

***Ans:***

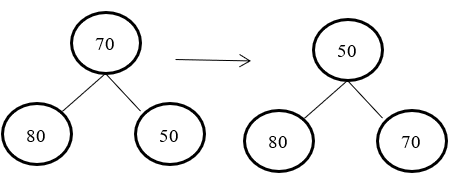
1. Insert 70



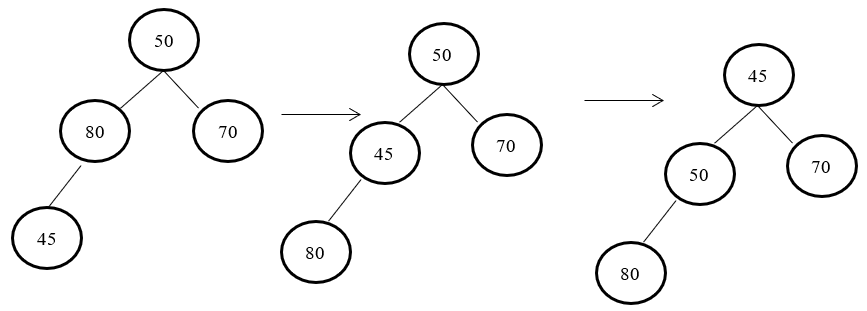
2. Insert 80



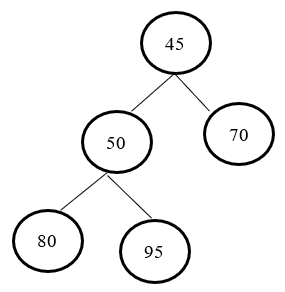
3. Insert 50



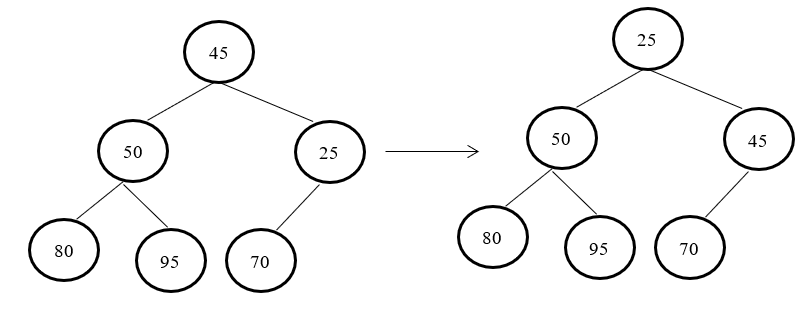
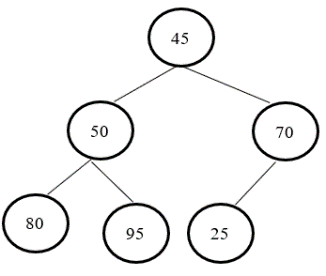
4. Insert 45



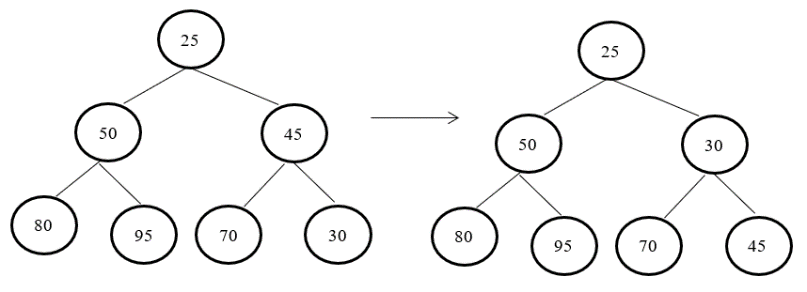
5. Insert 95



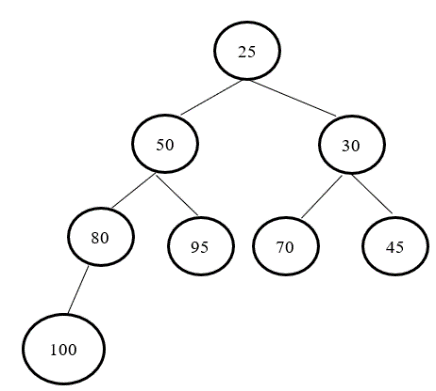
6. Insert 25



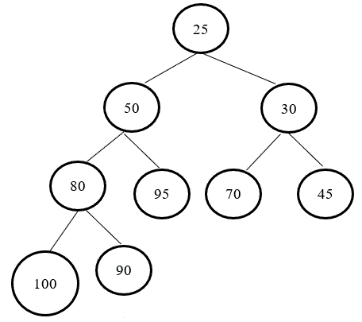
7. Insert 30



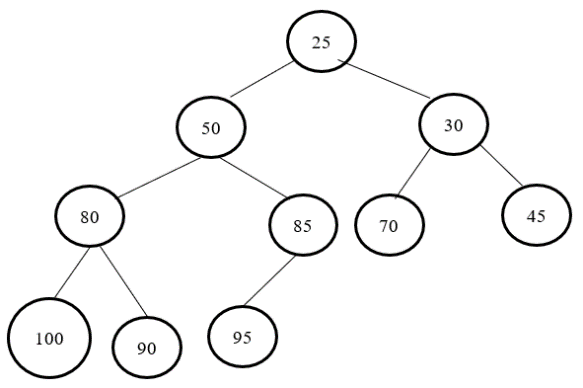
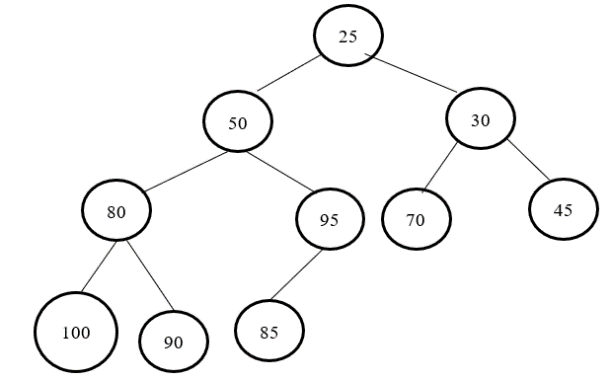
8. Insert 100



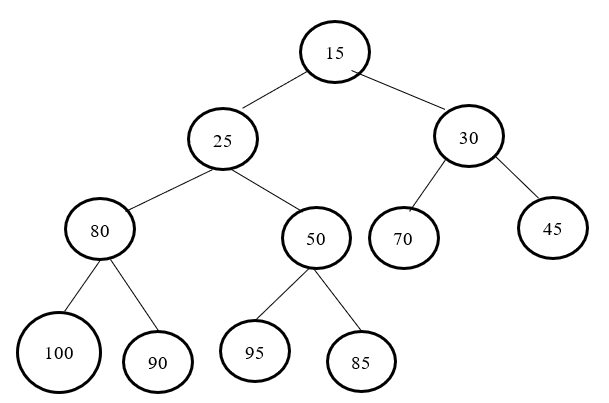
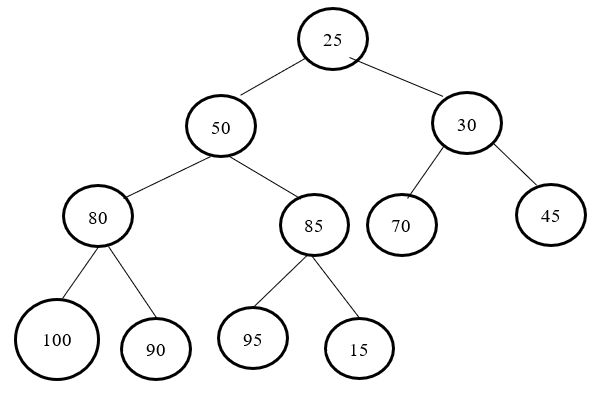
9. Insert 90



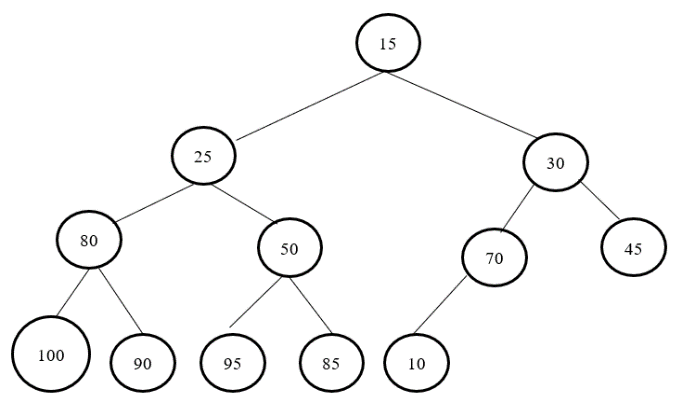
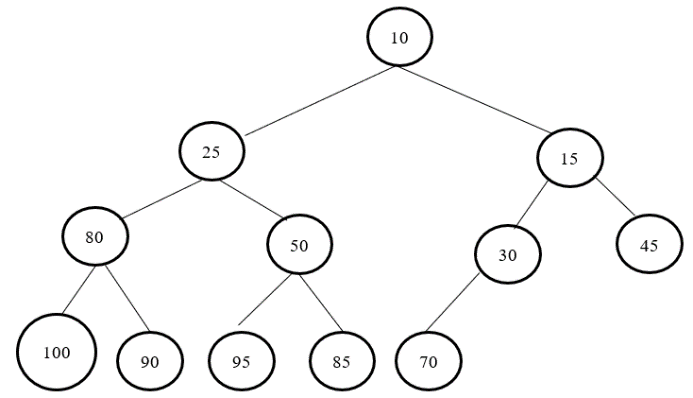
10. Insert 85



11. Insert 15



12. Insert 10

******

***Q4f Write a short note on Red Black Tree***

**Ans:**

Red-Black tree is a binary search tree in which every node is colored red or black.

* Red-Black tree is a balanced tree because no path is more than twice as long as any other path.
* Each node of the red-black tree contains five fields

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Color | Left | Key | Right | Parent |

* The color field of this node is taken as one bit field as only two possible values are used (Red and Black) for this field. Parent field of the node contains the address of the parent node of the current node. A red-black tree must satisfy the following properties:-

1. Every node of the tree is either red or black
2. The root node of the tree is always black
3. If a node in the tree is red then it’s both the child nodes must be black
4. All paths from a node to its descendent leaf nodes contain the same number of black nodes
5. All the leaf nodes must be black

***Question 5***

***Q5a Explain Adjacency Matrix Representation of Graph***

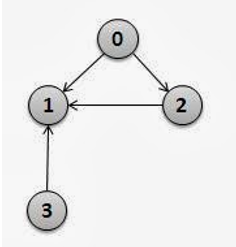
***Ans***:

The adjacency matrix, sometimes also called the connection matrix, of a simple labeled graph is a matrix with rows and columns labeled by graph vertices, with a 1 or 0 in position (v\_i,v\_j) according to whether v\_i and v\_j are adjacent or not. For a simple graph with no self-loops, the adjacency matrix must have 0s on the diagonal. For an undirected graph, the adjacency matrix is symmetric.

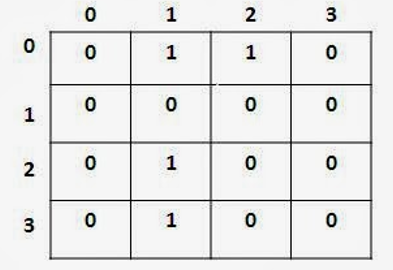
aij =

The adjacency matrix of a graph depends upon the ordering of its vertices that is if we change the order of vertices then it will result in different adjacency matrix.

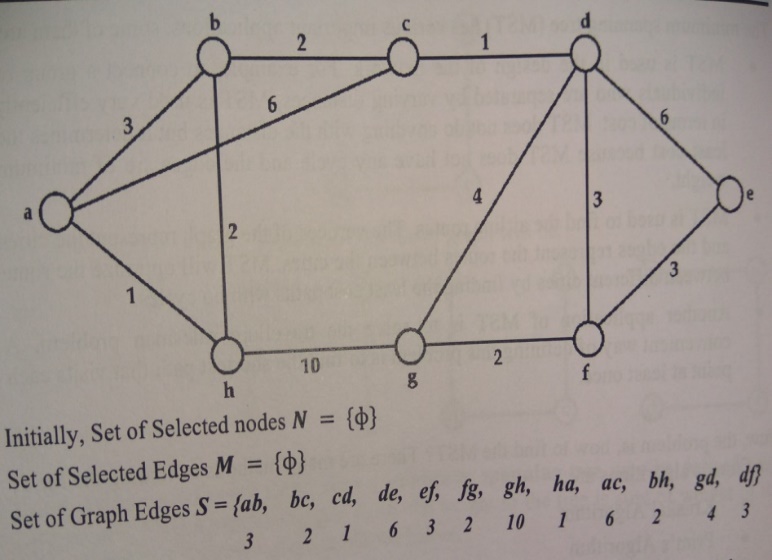
Consider the graph given below



The adjacency matrix corresponding to this ordering sequence will be



***Q5b Find the minimum spanning tree for the following graph using Prim’s algorithm and starting vertex’ a’***

****

***Ans:*** Set of Selected Nodes N = {0}

Set of Selected Edges M = {0}

Set of Graph Edges S = {ab, bc, cd, de, ef, fg, gh, ha, ac, bh, gd, df}

3 2 1 6 3 2 10 1 6 2 4 3

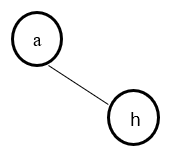
Put edge ah to list M and node h to list N

N = {a, h}

M = {ah}

S = {ab, bc, cd, de, ef, fg, gh, ac, bh, gd, df}

3 2 1 6 3 2 10 6 2 4 3



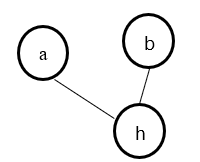
Now put edge bh to list M and node b to list N

N = {a, h, b}

M = {ah, hb}

S = {ab, bc, cd, de, ef, fg, gh, ac, gd, df}

3 2 1 6 3 2 10 6 4 3



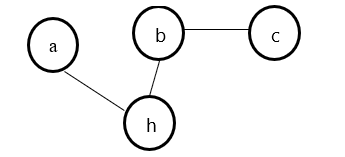
Now put edge bc to list M and node c to list N

N = {a, h, b, c}

M = {ah, hb, bc}

S = {ab, cd, de, ef, fg, gh, ac, gd, df}

3 1 6 3 2 10 6 4 3



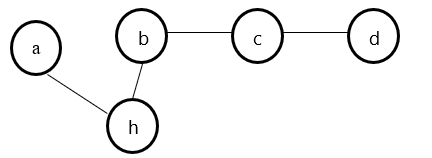
Now put edge cd to list M and node d to list N

N = {a, h, b, c, d}

M = {ah, hb, bc, cd}

S = {ab, de, ef, fg, gh, ac, gd, df}

3 6 3 2 10 6 4 3



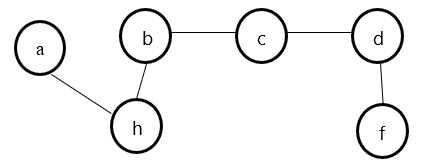
Now put edge df to list M and node f to list N

N = {a, h, b, c, d, f}

M = {ah, hb, bc, cd, df}

S = {ab, de, ef, fg, gh, ac, gd}

3 6 3 2 10 6 4



Now put edge fg to list M and node g to list N

N = {a, h, b, c, d, f, g}

M = {ah, hb, bc, cd, df, fg}

S = {ab, de, ef, gh, ac, gd}

3 6 3 10 6 4

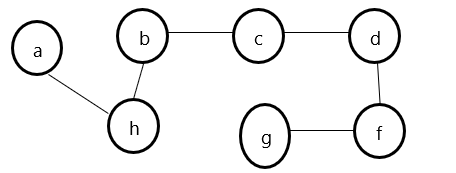
Now put edge ef to list M and node e to list N

N = {a, h, b, c, d, f, g, e}

M = {ah, hb, bc, cd, df, fg, fe}

S = {ab, de, gh, ac, gd}

3 6 10 6 4



Minimum Cost = 1 + 2 + 2 + 1 + 3 + 2 + 3 = 14

***Q5c Explain two different methods of Graph Traversal.***

***Ans:***

There are two graph traversal techniques and they are as follows...

1. DFS (Depth First Search)
2. BFS (Breadth First Search)

**Breadth-First Traversal (BFS)**

In BFS we begin with the start vertex v and mark it as discovered. A vertex is said to be processed by an algorithm has visited all vertices adjacent to it. All unvisited vertices adjacent to v will be visited next. Vertex v has now been processed. The newly visited vertices have not been processed yet, so these will be put at the end of the list of unprocessed vertices. Exploration continues until single unexplored vertex is left. Algorithm for traversal of a graph using BFS is given below

- Step 1: Initially all the vertices of the graph G are set with the status Undiscovered

- Step 2: Change the state of the starting vertex s of the graph to Discovered and put it into the queue Q

- Step 3: Repeat steps 4 to 5 while queue is not empty

- Step 4: Remove a vertex v which is at the front of the queue and change its state to Processed

- Step 5: Repeat for all Undiscovered adjacent vertices ui of vertex v

ui is set to the status Discovered and added to the queue

- [End Loop]

- [End Loop]

- Step 6: Exit

**Depth First Search (DFS)**

The concept behind the DFS is starting with a vertex s, follows a path from s to a dead end along a path P then backtrack on P and continue traversing along another path from start to end until all the vertices have been explained. The algorithm for the DFS is very similar to the BFS algorithm except that it uses stack to hold the unprocessed discovered vertices. Algorithm for traversal of a graph using DFS method where traversal starts at vertex s

- Step 1: Initially all vertices of the graph G are set with the status Undiscovered

- Step 2: Change the state of the starting vertex s of the graph to Discovered and put it onto the stack S

- Step 3: Repeat steps 4 and 5 while stack is not empty

- Step 4: Pop a vertex v which is at the Top of the stack and change its status to Processed

- Step 5: Repeat for all Undiscovered adjacent vertices ui of v

ui is set to the status Discovered and pushed to the stack S

- [End Loop]

- [End Loop]

- Step 6: Exit

***Q5d Write a short note on Mid-Square Hashing method.***

***Ans:***

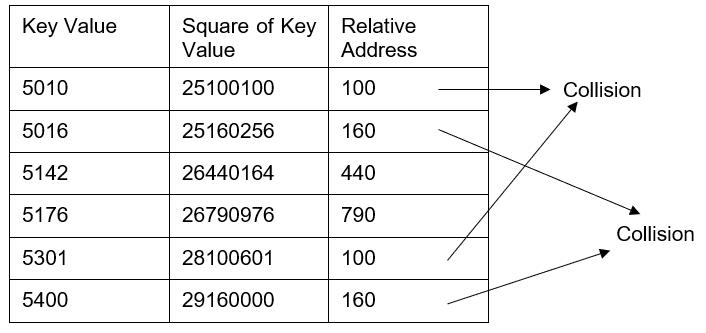
**Mid-Square Method**

It is one of the most commonly used for address calculation. In this method, the key value is squared (i.e multiplied by itself) and digits are taken from the middle of the square of the key value. Usually, the number of digits to be picked from the middle of the square depends on the address space on which records are to be mapped.

Example: Consider a 4-digit key 5010, the key will be squared to number 25100100.

Now, the address can be obtained by selecting the appropriate number of digits from the middle of the square. The number of digits to be picked from the middle of the square usually comes from the number of digits required for the relative address range.

Consider the mapping of 4- digits keys to 3-digits addresses by extracting digits from square which are at positions 4 to 6 from lower most digit of the number.



***Q5e Explain in brief Quadratic Probing collision resolution technique***

***Ans:***

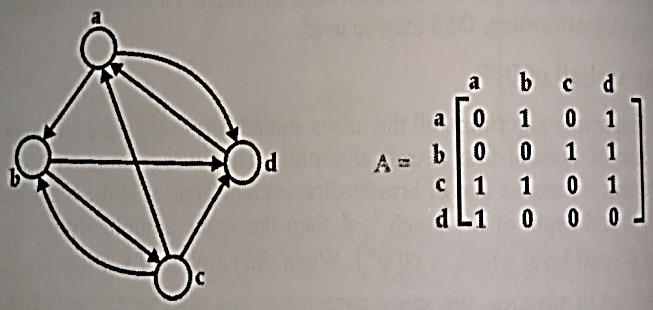
To use the quadratic probing technique the hash function will be, Hash(K , p) =[H(K)+ C1 p + C2p2 ] mod m

Here H(K) is the hash function used to find the relative address. Usually the remainder division method is used i.e p is the probe number which can be 0,1,2,…. m-1 where m is the size of the hash table and C1 and C2 are constants which are non-zero. For proper use of quadratic probing, the values of constants C1 and C2 must be chosen appropriately.

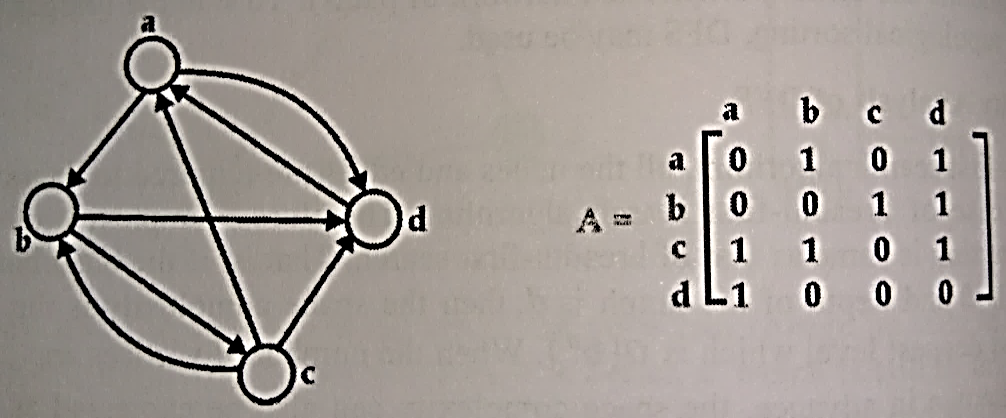
Consider a Hash Table of Size 10.

Suppose we want to insert some record with key values 33,101,93. Here the hash function used is taken as H(K)= K mod 10 The values of C1 and C2 are taken as 3 and 1 respectively.

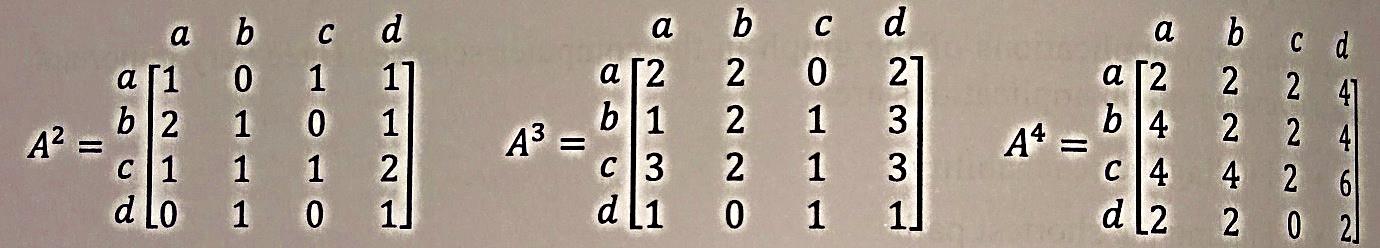
***Q5f For the given graph find the adjacency matrix A, using Matrix Multiplication method find A2, A3 and A4***

******

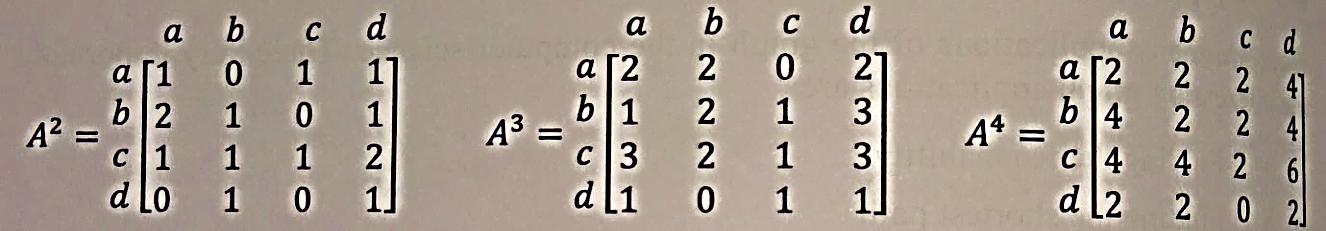
**Ans:** Adjacency matrix for the above graph is as below



By using the Matrix Multiplication method we get A2 as



By using the Matrix Multiplication method we get A3 as



By using the Matrix Multiplication method we get A4 as

