Q.2 a. Explain the following terms: Data Type, Abstract Data Type, and Primitive Data Structures.

Answer:

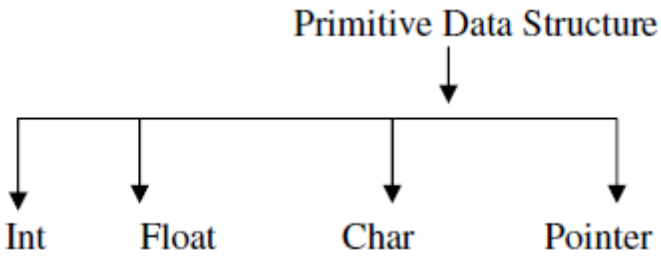
Data Type: We determine the amount of memory to reserve by determining the appropriate abstract data type group to use and then deciding which abstract data type within the group is right for the data.

There are four data type groups:

- **Integer** Stores whole numbers and signed numbers.
- **Floating-point** Stores real numbers (fractional values). Perfect for storing bank deposits where pennies (fractions of a dollar) can quickly add up to a few dollars.
- **Character** Stores a character. Ideal for storing names of things.
- **Boolean** Stores a true or false value. The correct choice for storing a yes or no or true or false response to a question.

Abstract Data Types: - A useful tool for specifying the logical properties of a data type is the abstract data type or ADT. The term "abstract data type" refers to the basic mathematical concept that defines the data type. In defining ADT as a mathematical concept, we are not concerned with space or time efficiency. An ADT consists of two parts:- a value definition and an operator definition. The value definition defines the collection of values for the ADT and consists of two parts: a definition clause and a condition clause. For example, the value consist definition for the ADT RATIONAL states that a RATIONAL value consists of two integers, the second of which does not equal 0. We use array notation to indicate the parts of an abstract type.

Primitive Data Structure: - These are the basic structure and are directly operated upon by the machine instructions. These in general have different representations on different computer. Integer floating point number, character constraints, string constants, pointer etc fall in this category.



b. Define an algorithm. Describe commonly used asymptotic notations and give their significance.

Answer:

Algorithm:

–Algorithm should have the following five characteristic features:
1. Input
2. Output
3. Definiteness
4. Effectiveness
5. Termination.

Therefore, an algorithm can be defined as a sequence of definite and effective instructions, which terminates with the production of correct output from the given input. Asymptotic notation describe the algorithm efficiency and performance in a meaningful way. These notations describe the behavior of time or space complexity for large instance characteristics. Some commonly used asymptotic notations are:

Big oh notation (O): The upper bound for the function 'f' is provided by the big oh notation (O). Considering 'g' to be a function from the non-negative integers to the positive real numbers, we define O(g) as the set of function f such that for some real constant c>0 and some non negative integers constant n0 , f(n)_cg(n) for all n_n0. Mathematically, O(g(n))={f(n): there exists positive constants such that 0_f f(n)_cg(n) for all n, n_n0} , we say "f is oh of g".

Big Omega notation (_): The lower bound for the function 'f' is provided by the big omega notation (_). Considering 'g' to be a function from the non-negative integers to the positive real numbers, we define _(g) as the set of function f such that for some real constant c>0 and some non negative integers constant n0 , f(n)_cg(n) for all n_n0. Mathematically, _(g(n))={f(n): there exists positive constants such that 0_cg(n) _f(n) for all n, n_n0}.

Big Theta notation (_): The upper and lower bound for the function 'f' is provided by the big oh notation (_). Considering 'g' to be a function from the non-negative integers to the positive real numbers, we define _(g) as the set of function f such that for some real constant c1 and c2 >0 and some non negative integers constant n0 , c1g(n)_f f(n)_c2g(n) for all n_n0. Mathematically, _(g(n))={f(n): there exists positive constants c1 and c2 such that c1g(n)_f f(n)_c2g(n) for all n, n_n0} , we say "f is oh of g"

c. Consider a 2-D array named GROUP [5] [7] is stored in row major order with base address 123. What is the address of GROUP [2] [3]?

Answer:
Given: A 2-D array named GROUP [5] [7]
Base address is 123
Let us assume that GROUP [5] [7] is of type integer.
The formula for row major form to calculate address is:

Loc(a[i] [j]) = base (a) + w [n (i –lbr) + (j-lbc)]
Where
n no. of column in array
w no of bytes per storage location
lbr lower bound for row index
lbc lower bound for column index.
Loc(GROUP [2] [3]) = 123 + 2[7(2-0) + (3-0)]
= 123 +2[14+3]
= 123 + 34
=157

Q.3 a. Define stack. Show the steps to convert the following infix expression to postfix form using stack.

$$A+(B*C-(D/E\wedge F)*E)*H$$

Answer: Page number 196 of Text Book

b. Consider the following sequentially implemented double ended queue of characters which is allocated 5 characters with Front = 1 and Rear = 2.

DQueue

	B	C		

F=1, R=2

Now, perform the following operations on the deque
(i) Add D from the front end.
(ii) Add E from the rear end.
(iii) Delete E from the rear end.
(iv) Delete D from the front end.

Answer: Page number 194 of Text Book**.**

Q.4 a. Distinguish between singly and doubly linked list using an example. Show a schematic diagram that represents the scenario of a linked list of two elements and a new element which has to be inserted between them.

Answer:

Each element in the singly linked list contains a reference to the next element in the list, while each element in the doubly linked list contains references to the next element as well as the previous element in the list. Doubly linked lists require more space for each element in the list and elementary operations such as insertion and deletion is more complex since they have to deal with two references. But doubly link lists allow easier manipulation since it allows traversing the list in forward and backward directions.

Each element in a singly linked list has two fields as shown in Figure 1. The data field holds the actual data stored and the next field holds the reference to the next

element in the chain. The first element of the linked list is stored as the head of the linked list.

Fig1: Elements of singly linked list

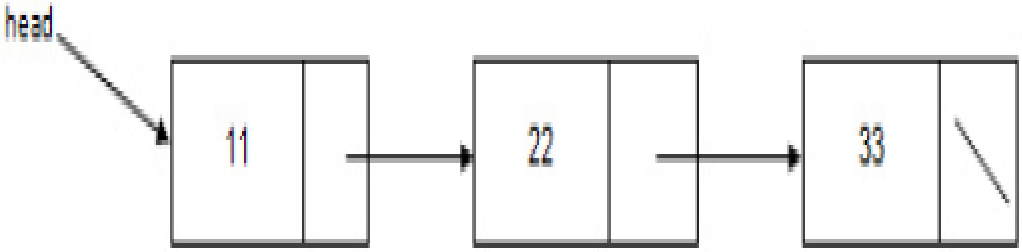


Fig2: A singly linked list

null value in its next field. Any element in the list can be accessed by starting at the head and following the next pointer until you meet the required element. Each element in a doubly linked list has three fields as shown in Figure 3. Similar to singly linked list, the data field holds the actual data stored and the next field holds the reference to the next element in the chain. Additionally, the previous field holds the reference to the previous element in the chain. The first element of the linked list is stored as the head of the linked list.

Fig3: Elements of doubly linked list

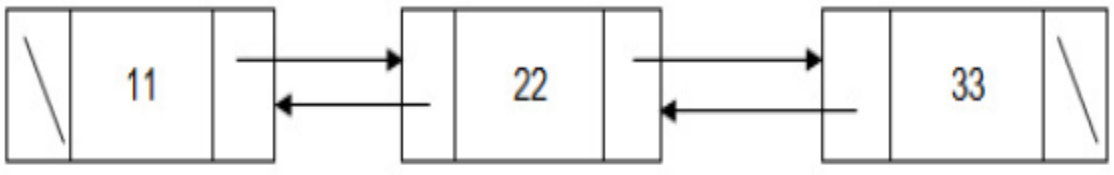


Fig4: A doubly linked list

Before insertion

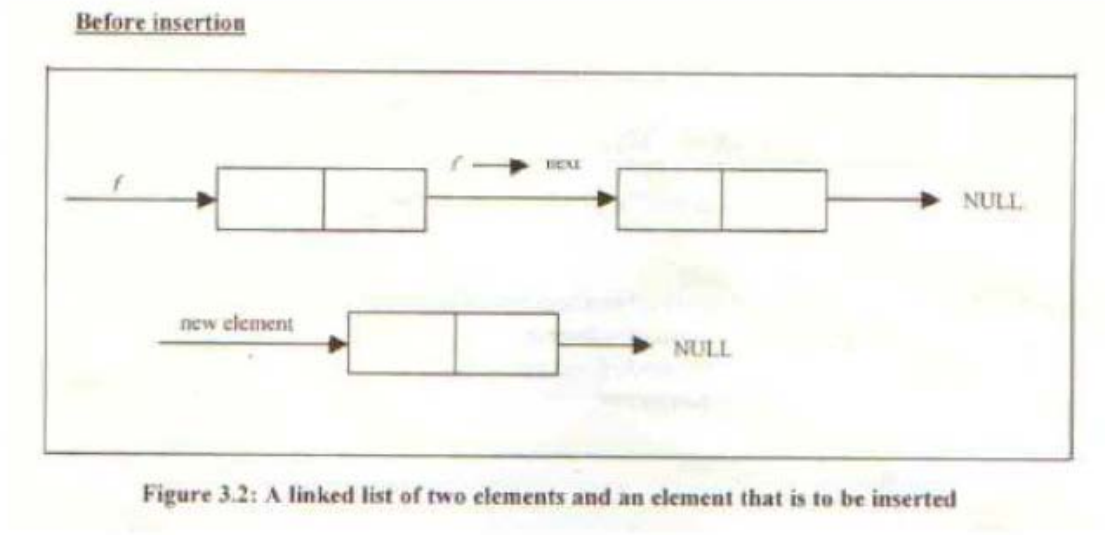
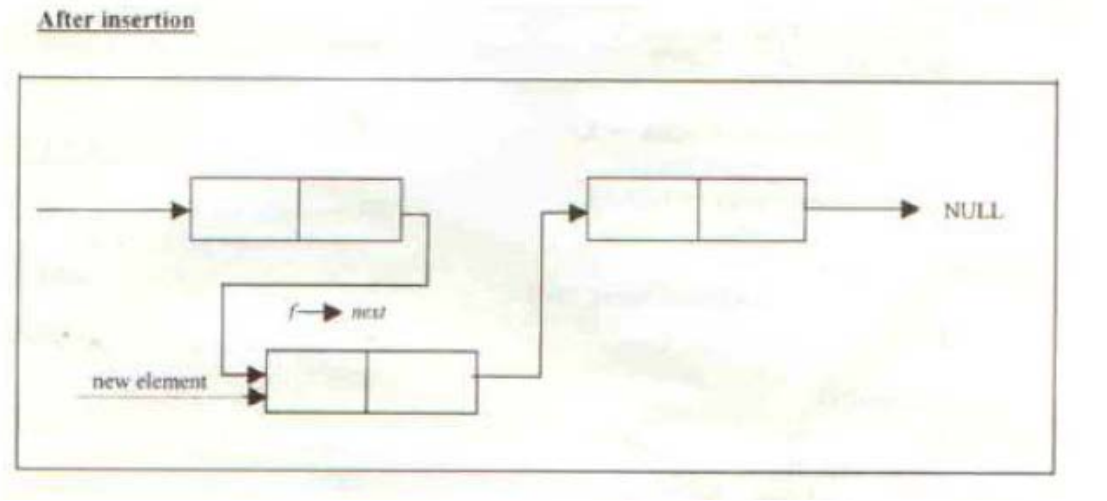


Figure 3.2: A linked list of two elements and an element that is to be inserted

After insertion



b. Write an algorithm to count the number of nodes in the circular linked list.

Answer:

Algorithm :- Counting No of Nodes in Circular List

Let list be a circular header list in memory. This algorithm traverse and counts number of nodes in a LIST.

0. set COUNT: = 0
1. Set PTR: = LINK [START]. {Initializes the pointer PTR}
2. Repeat steps 3, 4, 5 while PTR = START;

3. COUNT = COUNT + 1
4. Set PTR = LINK [PTR]. [PTR now points to next node]
[End of step 2 loop]
5. Exit

c. Write a program in C / C++ to implement queue using single linked list.

Answer: Page number of 236 for Text Book**.**

Q.5 a. Write the short notes on the following trees:
(i) Full binary tree
(ii) Binary search Tree
(iii) Threaded binary tree

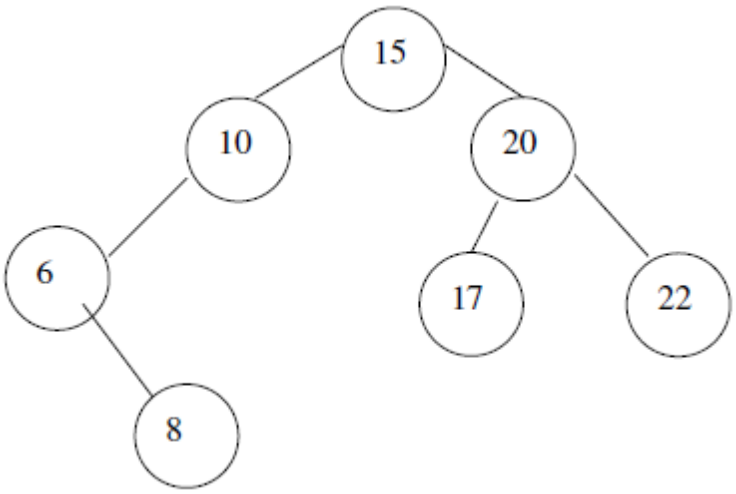
Answer:

(i) Full binary tree: A binary tree in which if all its levels except possibly the last, have the maximum number of
nodes and all the nodes at the last level appear as far left as possible, is known as full binary tree .

A full binary tree with 2n+1 nodes contain n non-leaf nodes

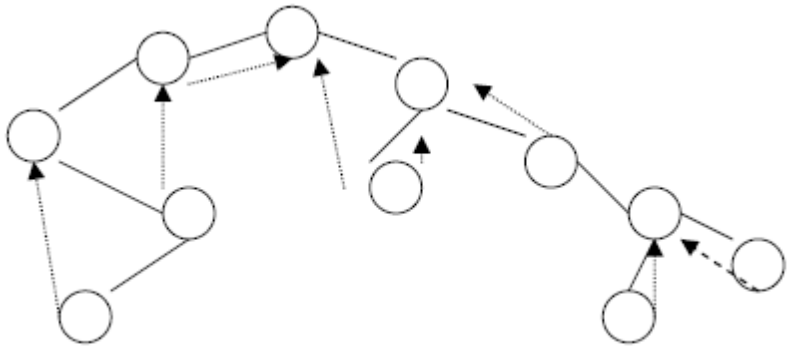
(ii) Binary search tree

A binary search tree is a binary tree that is either empty or in which each node contains a key that satisfies the following conditions: -
_ All keys (if any) in the left sub tree of the root precede the key in the root.
_ The key in the root precedes all keys (if any) in its right sub tree.
_ The left and right sub trees of the root are again search trees.



(iii) Threaded Binary Tree:-By changing the NULL lines in a binary tree to special links called threads, it is possible to perform traversal, insertion and deletion without using either a stack or recursion. In a **right in threaded binary tree** each NULL link is replaced by a special link to the successor of that node under inorder traversal called right threaded. Using right threads we shall find it easy to do an inorder traversal of the tree, since we need only follow either an ordinary link or a threaded to find the next node to visit.

If we replace each NULL left link by a special link to the predecessor of the node known as left threaded under inorder traversal the tree is known as **left in threaded binary tree**. If both the left and right threads are present in tree then it is known **as fully threaded binary tree** for example:

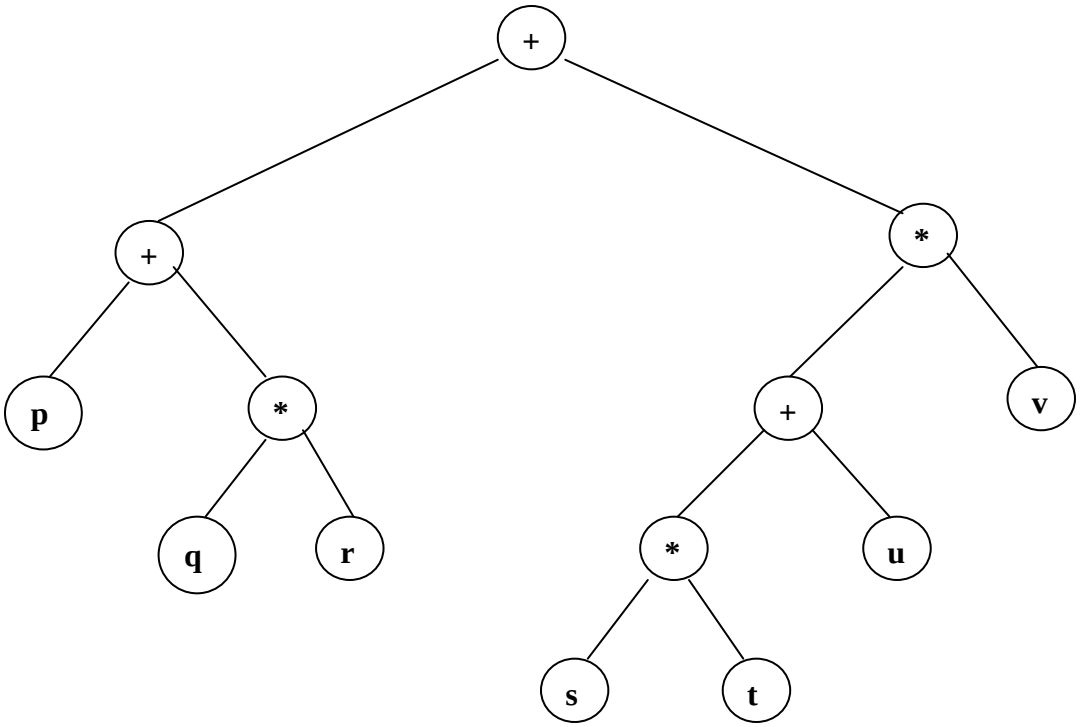


b. Explain Expression Tree. Draw the expression tree for the expression: (p + (q * r)) + (((s * t) + u) * v)

Answer:

The leaves of an expression tree are operands, such as constants or variable names, and the other nodes contain operators. This particular tree happens to be binary, because all of the operations are binary, and although this is the simplest case, it is possible for nodes to have more than two children. It is also possible for a node to have only one child, as is the case with the unary minus operator. We can evaluate an expression tree, T, by applying the operator at the root to the values obtained by recursively evaluating the left and right subtrees.

The expression tree for the expression (p + (q * r)) + (((s * t) + u) * v) is:

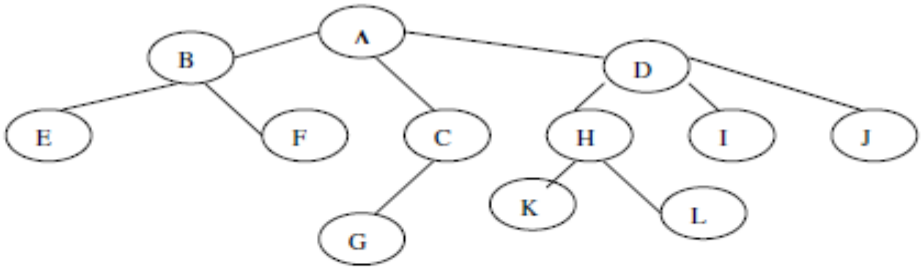


c. Discuss that how to convert a general tree to a Binary Tree.

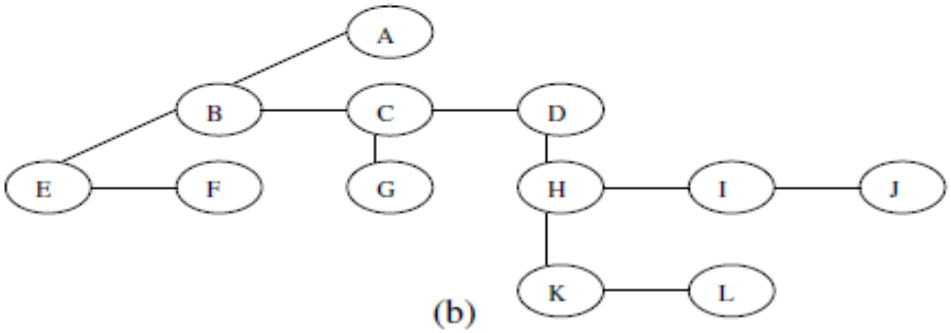
Answer:
Conversion of general trees to binary trees:

A general tree can be converted into an equivalent binary tree. This conversion process is called the natural correspondence between general and binary trees.

The algorithm is as follows:
(a) Insert edges connecting siblings from left to right at the same level.
(b) Delete all edges of a parent to its children except to its left most offspring.
(c) Rotate the resultant tree 450 to mark clearly left and right subtrees.
A general tree shown in figure (a) converted into a binary tree shown in (b)



(a)



(b)

Q.6 a. Define a graph. What are the standard ways in which a graph can be traversed?

Answer:

A graph G may be defined as a finite set V of vertices and a set E of edges (pair of connected vertices). The notation used is as follows:

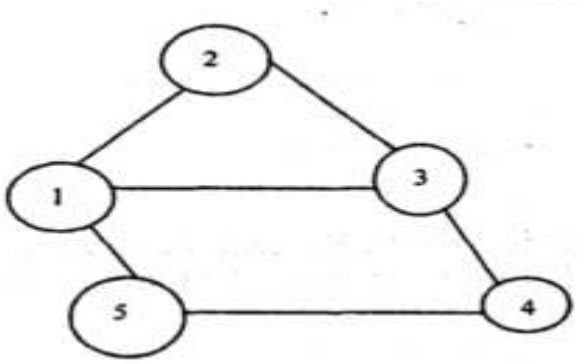
Graph G = (V, E)

Consider the graph of Figure give below:

The set of vertices for the graph is V — {1, 2, 3, 4, 5}.

The set of edges for the graph is 13 = {(1,2), (1,5), (1,3), (5,4), (4,3), (2,3) }.

The elements of 13 are always a pair of elements.



It may be noted that unlike nodes of a tree, graph has a very limited relationship between the nodes (vertices). There is no direct relationship between the vertices I and 4 although they are connected through 3. Directed graph and Undirected graph: If every edge (a,b) in a graph is marked by a direction from a to b, then we call it a Directed graph (digraph). On the other hand, if directions are not marked on the edges, then the graph is called an Undirected graph. In a Directed graph, the edges (1,5) and (5,1) represent two different edges whereas in an Undirected graph, (1,5) and (5,1) represent the same edge. Graphs are used in various types of modeling. For example, graphs can be used to represent connecting roads between cities.

The standard ways of traversing a graph are as follows:

i. The *depth-first traversal* of a graph is like the depth-first traversal of a tree. Since a graph has no root, when we do a depth-first traversal, we must specify the vertex at which to begin. Depth-first traversal of a graph visits a vertex and then recursively visits all the vertices adjacent to that node. The catch is that the graph may contain cycles, but the traversal must visit every vertex at most once. The solution to the problem is to keep track of the nodes that have been visited, so that the traversal does not suffer the fate of infinite recursion.

ii. The *breadth-first traversal* of a graph is like the breadth-first traversal of a tree. Breadth-first tree traversal first visits all the nodes at depth zero (i.e., the root), then all the nodes at depth one, and so on. Since a graph has no root, when we do a breadth-first traversal, we must specify the vertex at which to start the traversal. Furthermore, we can define the depth of a given vertex to be the length of the shortest path from the starting vertex to the given vertex. Thus, breadth-first traversal first visits th starting vertex, then all the vertices adjacent to the starting vertex, and the all the vertices adjacent to those, and so on.

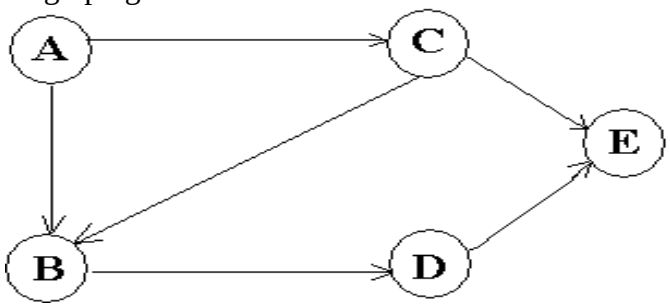
b. Write the Prim's algorithm for finding the minimum-spanning tree of a graph

Answer:

```
MST-PRIM(G, w, r)
 1  for each u ∈ G.V
 2       u.key = ∞
 3       u.π = NIL
 4  r.key = 0
 5  Q = G.V
 6  while Q ≠ ∅
 7       u = EXTRACT-MIN(Q)
 8       for each v ∈ G.Adj[u]
 9            if v ∈ Q and w(u, v) < v.key
10                 v.π = u
11                 v.key = w(u, v)
```

Prim's Algorithm:

c. Draw the adjacency matrix and the adjacency list representation for the directed graph given below



Answer:

Adjacency list representation*:* This representation of graph consists of an array Adj of |V| lists, one for each vertex in V. For each u_V, the adjacency list Adj[u] contains all the vertices v such that there is an edge (u,v)_ E that is Adj[u] consists of all the vertices adjacent to u in G. The vertices in each adjacency list are stored in an arbitrary order.

Adjacency Matrix representation: This representation consists of |V|*|V| matrix A= (aij) such that

$$a_{ij} = \begin{cases} 1 \text{ if}(i,j)\varepsilon\ E \\ 0 \text{ otherwise} \end{cases}$$

Q7 a. What do you mean by hashing? Describe any two commonly used hash functions

Answer:
A hash table is a data structure in which the location of a data item is determined directly as a function of data item itself rather than by a sequence of comparison. Under ideal condition, the time required to locate a data item in a hash table is 0(1) ie. it is constant and documents not depend on the number of data items stored. When the set of K of keys stored is much smaller then the universe U of all possible keys, a hash table require much less storage spare than a direct address table.

Hash Function
A hash function h is simply a mathematical formula that manipulates the key in some form to compute the index for this key in the hash table. Eg a hash function can divide the key by some number, usually the size of the hash table and return remainder as the index for the key. In general, we say that a hash function h maps the universe U of keys into the slots of a hash table T[0….n-1]. This process of mapping keys to appropriate slots in a hash table is known as **hashing.**

Some of methods of hashing are:-
a) Division method
b) Multiplication method
c) Midsquare method
d) Folding method

(a) Division Method:- In this method, key K to be mapped into one of the m states in the hash table is divided by m state in the hash table is divided by m and the remainder of this division taken as index into the hash table. That is the hash function is h(k) = k mod m where mod is the modules operations.

(b) Multiplication Method: The multiplication method operates in 2 steps. In the 1st step the key value K is multiplied by a constant A in the range O<A<1 and the fractional part of the value obtained above is multiplied by m and the floor of the result is taken as the hash values. That is the hash function is h(k) = [m (K A mod 1)]

b. Write an algorithm in C / C++ for Binary Search technique. Use the algorithm to search 20 in the following list of elements. Explain the process.
12, 16, 17, 19, 20, 22, 24, 29, 30, 32, 37

Answer: Page number 398 & 399 of Text Book**.**

Q8. a. Write an algorithm in C / C++ for quick sort. Sort the following sequence of numbers using quick sort method.
26, 58, 49, 37, 11, 91, 83, 32

Answer:

Quick sort is a sorting algorithm that uses the idea if divide and conquer. This algorithm chooses an element known as pivot element, finds its position in the array such that it divides the array into two sub arrays. The elements in the left sub array are less than and the elements in the right sub array are greater than the dividing element. Then the algorithm is repeated separately for these two sub array.

An algorithm for quick sort:

```
void quicksort ( int a[ ], int lower, int upper )
{
int i ;
if ( upper > lower ) {
i = split ( a, lower, upper ) ;
quicksort ( a, lower, i - 1 ) ;
quicksort ( a, i + 1, upper ) ; }
}
int split ( int a[ ], int lower, int upper ){
int i, p, q, t ;
p = lower + 1 ;
q = upper ;
i = a[lower] ;
while ( q >= p )
{
while ( a[p] < i )
p++ ;
while ( a[q] > i )
q-- ;
if ( q > p )
{
t = a[p] ;
a[p] = a[q] ;
a[q] = t ;
}
}
t = a[lower] ;
a[lower] = a[q] ;
a[q] = t ;
return q ; }
```

The given data is :- 26, 58, 49, 37, 11, 91, 83, 32
Pass 1:- (11), **26**, (58, 49, 37, 91, 83, 32)
Pass 2:- 11, 26, (58, 49, 37, 91, 83, 32)
Pass 3:- 11, 26, (49, 37, 32), 58, (91, 83)
Pass 4:- 11, 26, (37, 32), 49, 58, (91, 83)
Pass 5:- 11, 26, (32), 37, 49, 58, (91, 83)
Pass 6:- 11, 26, 32, 37, 49, 58, (83, 91)
Pass 7:- 11, 26, 32, 37, 49, 58, 83, 91

b. Explain the radix sort with its algorithm. What is the time complexity of radix sort?

Answer:
Radix-Sort: The radix-sort algorithm uses several passes of counting-sort to allow for a much greater range of maximum values.

Radix-sort sorts *w*-bit integers by using *w/d* passes of counting-sort to sort these integers *d* bits at a time. More precisely, radix sort first sorts the integers by their least significant *d* bits, then their next significant *d* bits, and so on until, in the last pass, the integers are sorted by their most significant *d* bits.

```
int[] radixSort(int[] a) {
      int[] b = null;
      for (int p = 0; p < w/d; p++) {
         int c[] = new int[1<<d];
         // the next three for loops implement counting-sort
         b = new int[a.length];
         for (int i = 0; i < a.length; i++)
            c[(a[i] >> d*p)&((1<<d)-1)]++;
         for (int i = 1; i < 1<<d; i++)
            c[i] += c[i-1];
         for (int i = a.length-1; i >= 0; i--)
            b[--c[(a[i] >> d*p)&((1<<d)-1)]] = a[i];
         a = b;
      }
      return b;
   }
```

Time complexity of radix sort: *For any integer d > 0, the radix sort (a,k) method can sort an array a containing* n w*-bit integers in O ((w/d) (n+*$2^{d)}$*) time.*
The radix sort (a,k) method can sort an array a containing n *integer values in the range { 0,…*n^c *– 1} in*
O (cn)time.

Q9. a. Write a program to copy the contents of one file into another file using command line arguments.

Answer:

```
#include<stdio.h>
#include<conio.h>
#include<stdlib.h>
void main(int arg,char *arr[])
{
FILE *fs,*ft;
char ch;
```

```
clrscr();
if(arg!=3)
{
printf("Argument Missing ! Press key to exit.");
getch();
exit(0);
}
fs = fopen(arr[1],"r");
if(fs==NULL)
{
 printf("Cannot open source file ! Press key to exit.");
getch();
exit(0);
 }
ft = fopen(arr[2],"w");
 if(ft==NULL)
{
printf("Cannot copy file ! Press key to exit.");
fclose(fs);
 getch();
exit(0);
}
while(1)
{
ch = getc(fs);
 if(ch==EOF)
{
break;
}
else
 putc(ch,ft);
 }
printf("File copied succesfully!");
 fclose(fs);
fclose(ft);
 }
```

b. Discuss the various methods for storing sequential files? Also write a program that creates random numbers in a given file.

Answer:

The methods available in storing sequential files are:

- Straight merging,
- Natural merging,
- Polyphase sort,
- Distribution of Initial runs.

Program :

```
#include <stdio.h>
#include <conio.h>
#include <stdlib.h>
void main( )
{
FILE *fp ;
char str [ 67 ] ;
int i, noofr, j ;
clrscr( ) ;
printf ( "Enter file name: " ) ;
scanf ( "%s", str ) ;
printf ( "Enter number of records: " ) ;
scanf ( "%d", &noofr ) ;
fp = fopen ( str, "wb" ) ;
if ( fp == NULL )
{
printf ( "Unable to create file." ) ;
getch( ) ;
exit ( 0 ) ;
}
randomize( ) ;
for ( i = 0 ; i < noofr ; i++ )
{
j = random ( 1000 ) ;
fwrite ( &j, sizeof ( int ), 1, fp ) ;
printf ( "%d\t", j ) ;
}
fclose ( fp ) ;
printf ( "\nFile is created. \nPress any key to continue." ) ;
getch( ) ;}
```

Text Book

Data Structures using C & C++, Rajesh K. Shukla, Wiley India, 2009.