STEP TOWARDS SUCCESS

AKASH'S

Guru Gobind Singh Indra Prastha University Series

SOLVED PAPERS

[PREVIOUS YEARS SOLVED QUESTION PAPERS]

[B.Tech]
THIRD SEMESTER
Data Structure
(CIC-209)

Rs.81.00/-

AKASH BOOKS
NEW DELHI

NEW TOPICS ADDED FROM ACADEMIC SESSION 2022-23
THIRD SEMESTER [B.TECH]
DATA STRUCTURE (CIC-209)

Q.1. What is the Time – Space Trade Off in Algorithm & also explain its type.

Ans. Time-Space Trade-Off in Algorithms

A trade off is a situation where one thing increases and another thing decreases. It is a way to solve a problem in:

- Either in less time and by using more space, or
- In very little space by spending a long amount of time.

The best Algorithm is that which helps to solve a problem that requires less space in memory and also takes less time to generate the output. But in general, it is not always possible to achieve both of these conditions at the same time. The most common condition is an algorithm using a lookup table. This means that the answers to some questions for every possible value can be written down. One way of solving this problem is to write down the entire lookup table, which will let you find answers very quickly but will use a lot of space. Another way is to calculate the answers without writing down anything, which uses very little space, but might take a long time. Therefore, the more time-efficient algorithms you have, that would be less space-efficient.

Types of Space-Time Trade-off

(a) Compressed or Uncompressed data: A space-time trade-off can be applied to the problem of data storage. If data stored is uncompressed, it takes more space but less time. But if the data is stored compressed, it takes less space but more time to run the decompression algorithm. There are many instances where it is possible to directly work with compressed data. In that case of compressed bitmap indices, where it is faster to work with compression than without compression.

(b) Re-Rendering or Stored images: In this case, storing only the source and rendering it as an image would take more space but less time i.e., storing an image in the cache is faster than re-rendering but requires more space in memory.

(c) Smaller code or Loop Unrolling: Smaller code occupies less space in memory but it requires high computation time that is required for jumping back to the beginning of the loop at the end of each iteration. Loop unrolling can optimize execution speed at the cost of increased binary size. It occupies more space in memory but requires less computation time.

(d) Lookup tables or Recalculation: In a lookup table, an implementation can include the entire table which reduces computing time but increases the amount of memory needed. It can recalculate i.e., compute table entries as needed, increasing computing time but reducing memory requirements.

Q.2. What is algorithm and why analysis of it is important?

Ans. In the analysis of the algorithm, it generally focused on CPU (time) usage, Memory usage, Disk usage, and Network usage. All are important, but the most concern is about the CPU time. Be careful to differentiate between:

- **Performance:** How much time/memory/disk/etc. is used when a program is run.

This depends on the machine, compiler, etc. as well as the code we write.

- **Complexity:** How do the resource requirements of a program or algorithm scale, i.e. what happens as the size of the problem being solved by the code gets larger.

Note: Complexity affects performance but not vice-versa.

Algorithm Analysis:

Algorithm analysis is an important part of computational complexity theory, which provides theoretical estimation for the required resources of an algorithm to solve a specific computational problem. Analysis of algorithms is the determination of the amount of time and space resources required to execute it.

Why Analysis of Algorithms is important?

- To predict the behavior of an algorithm without implementing it on a specific computer.
- It is much more convenient to have simple measures for the efficiency of an algorithm than to implement the algorithm and test the efficiency every time a certain parameter in the underlying computer system changes.
- It is impossible to predict the exact behavior of an algorithm. There are too many influencing factors.
- The analysis is thus only an approximation; it is not perfect.
- More importantly, by analyzing different algorithms, we can compare them to determine the best one for our purpose.

Q.3. Explain the different types of Algorithm analysis?

Ans. Types of Algorithm Analysis are as follows:

(a) Best case: Define the input for which algorithm takes less time or minimum time. In the best case calculate the lower bound of an algorithm. Example: In the linear search when search data is present at the first location of large data then the best case occurs.

(b) Worst Case: Define the input for which algorithm takes a long time or maximum time. In the worst calculate the upper bound of an algorithm. Example: In the linear search when search data is not present at all then the worst case occurs.

(c) Average case: In the average case take all random inputs and calculate the computation time for all inputs.

And then we divide it by the total number of inputs.

Average case = all random case time / total no of case

Q.4. What is the Asymptotic Analysis of Algorithms? Explain the Different types of Asymptotic Notations?

Ans. Asymptotic Analysis - Asymptotic analysis of an algorithm refers to defining the mathematical boundation/framing of its run-time performance. Using asymptotic analysis, we can very well conclude the best case, average case, and worst case scenario of an algorithm.

Asymptotic analysis is input bound i.e., if there's no input to the algorithm, it is concluded to work in a constant time. Other than the "input" all other factors are considered constant.

Asymptotic analysis refers to computing the running time of any operation in mathematical units of computation. For example, the running time of one operation is computed as f(n) and may be for another operation it is computed as $g(n^2)$. This means the first operation running time will increase linearly with the increase in n and the running time of the second operation will increase exponentially when n increases. Similarly, the running time of both operations will be nearly the same if n is significantly small.

Usually, the time required by an algorithm falls under three types –

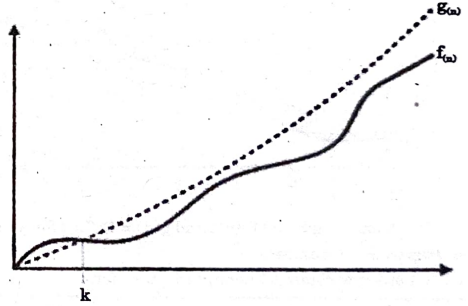
- **Best Case** – Minimum time required for program execution.
- **Average Case** – Average time required for program execution.
- **Worst Case** – Maximum time required for program execution.

Asymptotic Notations

Following are the commonly used asymptotic notations to calculate the running time complexity of an algorithm.

- O Notation
- Ω Notation
- θ Notation

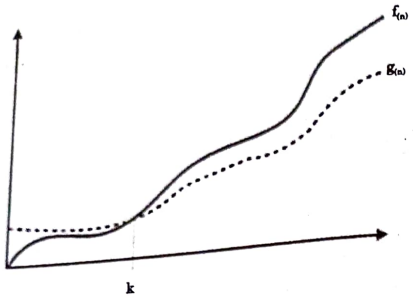
Big Oh Notation, O : The notation O(n) is the formal way to express the upper bound of an algorithm's running time. It measures the worst case time complexity or the longest amount of time an algorithm can possibly take to complete.



For example, for a function f(n)

$O(f(n)) = \{ g(n) : \text{there exists } c > 0 \text{ and } n_0 \text{ such that } f(n) \leq c.g(n) \text{ for all } n > n_0. \}$

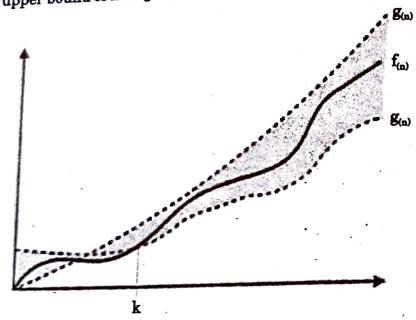
Omega Notation, Ω : The notation Ω(n) is the formal way to express the lower bound of an algorithm's running time. It measures the best case time complexity or the best amount of time an algorithm can possibly take to complete.



For example, for a function f(n)

$\Omega(f(n)) \geq \{ g(n)$: there exists $c > 0$ and n_0 such that $g(n) \leq c.f(n)$ for all $n > n_0$. $\}$

Theta Notation, θ **:** The notation $\theta(n)$ is the formal way to express both the lower bound and the upper bound of an algorithm's running time. It is represented as follows –



$\theta(f(n)) = \{ g(n)$ if and only if $g(n) = O(f(n))$ and $g(n) = \Omega(f(n))$ for all $n > n_0$. $\}$

Common Asymptotic Notations

Following is a list of some common asymptotic notations –

constant	–	O(1)
logarithmic	–	O(log n)
linear	–	O(n)
n log n	–	O(n log n)
quadratic	–	$O(n^2)$
cubic	–	$O(n^3)$
polynomial	–	$n^{O(1)}$
exponential	–	$2^{O(n)}$

Q.5. What is the Sparse Matrix? Explain the Array & Linked List representation of Sparse Matrix.

Ans. A matrix is a two-dimensional data object made of m rows and n columns, therefore having total m × n values. If most of the elements of the matrix have 0 value, then it is called a sparse matrix.

Why to use Sparse Matrix instead of simple matrix ?

- **Storage:** There are lesser non-zero elements than zeros and thus lesser memory can be used to store only those elements.
- **Computing time:** Computing time can be saved by logically designing a data structure traversing only non-zero elements..

Example:

0 0 3 0 4
0 0 5 7 0
0 0 0 0 0
0 2 6 0 0

Representing a sparse matrix by a 2D array leads to wastage of lots of memory as zeroes in the matrix are of no use in most of the cases. So, instead of storing zeroes with non-zero elements, we only store non-zero elements. This means storing non-zero elements with **triples- (Row, Column, value).**

Sparse Matrix Representations can be done in many ways following are two common representations:

1. Array representation 2. Linked list representation

Method 1: Using Arrays: 2D array is used to represent a sparse matrix in which there are three rows named as

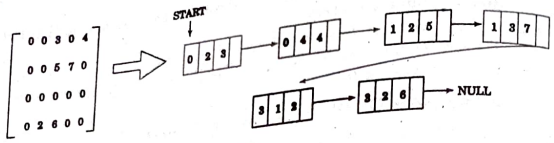
- **Row:** Index of row, where non-zero element is located
- **Column:** Index of column, where non-zero element is located
- **Value:** Value of the non zero element located at index – (row, column)

$$\begin{bmatrix} 0 & 0 & 3 & 0 & 4 \\ 0 & 0 & 5 & 7 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 2 & 6 & 0 & 0 \end{bmatrix}$$

Row	0	0	1	1	3	3
Column	2	4	2	3	1	2
Value	3	4	5	7	2	6

Method 2: Using Linked Lists : In linked list, each node has four fields. These four fields are defined as:

- **Row:** Index of row, where non-zero element is located
- **Column:** Index of column, where non-zero element is located
- **Value:** Value of the non zero element located at index – (row,column)
- **Next node:** Address of the next node



NODE STRUCTURE | ROW | COLUMN | VALUE | Address of next node

Q.6. What is a Disjoint set data structure & also explain its operations?

Ans. Two sets are called disjoint sets if they don't have any element in common, the intersection of sets is a null set.

A data structure that stores non overlapping or disjoint subset of elements is called disjoint set data structure. The disjoint set data structure supports following operations:

- Adding new sets to the disjoint set.
- Merging disjoint sets to a single disjoint set using Union operation.
- Finding representative of a disjoint set using Find operation.
- Check if two sets are disjoint or not.

Consider a situation with a number of persons and the following tasks to be performed on them:

- Add a new friendship relation, i.e. a person x becomes the friend of another person y i.e adding new element to a set.
- Find whether individual x is a friend of individual y (direct or indirect friend)

A union-find algorithm is an algorithm that performs two useful operations on such a data structure:

- **Find:** Determine which subset a particular element is in. This can be used for determining if two elements are in the same subset.
- **Union:** Join two subsets into a single subset. Here first we have to check if the two subsets belong to same set. If no, then we cannot perform union.

Q.7. What is the Polynomial?

Ans. A polynomial $p(x)$ is the expression in variable x which is in the form $(ax^n + bx^{n-1} + \ldots + jx + k)$, where a, b, c ...., k fall in the category of real numbers and 'n' is non negative integer, which is called the degree of polynomial.

An essential characteristic of the polynomial is that each term in the polynomial expression consists of two parts:

- one is the coefficient
- other is the exponent

Example:

$10x^2 + 26x$, here 10 and 26 are coefficients and 2, 1 is its exponential value.

Points to keep in Mind while working with Polynomials:

- The sign of each coefficient and exponent is stored within the coefficient and the exponent itself
- Additional terms having equal exponent is possible one
- The storage allocation for each term in the polynomial must be done in ascending and descending order of their exponent

Q.8. How to represent a Polynomial?

Ans. Polynomial can be represented in the various ways. These are:

- By the use of arrays
- By the use of Linked List

Representation of Polynomials Using Arrays

There may arise some situation where you need to evaluate many polynomial expressions and perform basic arithmetic operations like addition and subtraction with those numbers. For this, you will have to get a way to represent those polynomials. The simple way is to represent a polynomial with degree 'n' and store the coefficient of n+1 terms of the polynomial in the array. So every array element will consist of two values:

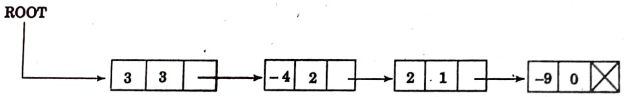
- Coefficient and
- Exponent

Polynomial representation using Linked List

The linked list can be used to represent a polynomial of any degree. Simply the information field is changed according to the number of variables used in the polynomial. If a single variable is used in the polynomial the information field of the node contains two parts: one for coefficient of variable and the other for degree of variable. Let us consider an example to represent a polynomial using linked list as follows:

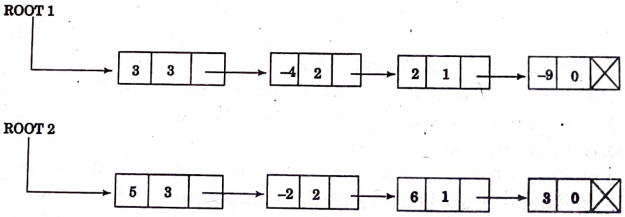
Polynomial: $3x^3 - 4x^2 + 2x - 9$

Linked List:

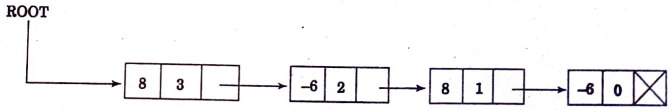


In the above linked list, the external pointer 'ROOT' point to the first node of the linked list. The first node of the linked list contains the information about the variable with the highest degree. The first node points to the next node with next lowest degree of the variable.

Representation of a polynomial using the linked list is beneficial when the operations on the polynomial like addition and subtractions are performed. The resulting polynomial can also be traversed very easily to display the polynomial.



The above two linked lists represent the polynomials, $3x^3 - 4x^2 + 2x - 9$ and $5x^3 - 2x^2 + 6x + 3$ respectively. If both the polynomials are added then the resulting linked will be:



The linked list pointer ROOT gives the representation for polynomial, $8x^3 - 6x^2 + 8x - 6$.

FIRST TERM EXAMINATION
THIRD SEMESTER (B.TECH) [ETCS-209]
DATA STRUCTURES—SEPT. 2014

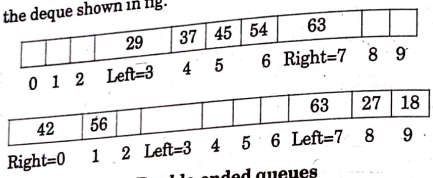
Time : 1.30 hrs. **M.M. : 30**

Note: *Attempt any three questions in total including Question No. 1 which is compulsory.*

Q.1.(a) Define a Double ended queue? What are its type?

Ans. A deque is a list in which the element can be inserted or deleted at either end. It is also known as a head-tail linked list, because elements can be added to or removed from either the front (head) or back (tail) end.

Consider the deque shown in fig.



Double-ended queues

Basically, there are two variants of double-ended queue.

(1) Input Restricted deque: In this dequeue, insertions can be done at one of the dequeue, while deletion can be done from both ends.

(2) Output Restricted deque: In this dequeue, deletions can be done only at one of the dequeue, while insertion can be done on both ends.

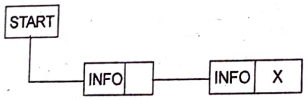
Q.1.(b) What are linear list? What are it types?

An. A linear list is a non-sequential collection of data items called nodes. These nodes in principle are structures containing fields, each node in a linked list has basically two field.

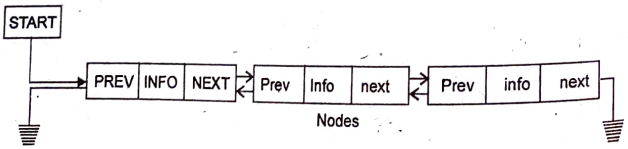
(1) Data field

(2) Link field.

The data field contains an actual value to be stored and processed. And the link field contain the address of the next data item in the linked list.



Single linked list



Doubly linked list

Types of linked list

(1) Singly linked list

(2) Doubly linked list

(3) Circular linked list

(4) Circular doubly linked list

Q.1.(c) Reverse a linked list without using the pointer?

Ans. To reverse a linear linked list, three pointer fields are used. These are tpt, ptr, cpt which hold the address of the previous node, current node and next node. To begin with the address of the first node, which is held in pointer variable FIRST, is assigned to PTR and typt is assigned value NULL.

Algorithm:

Step 1	PTR = FIRST
Step 2	TPT = NULL
Step 3	Repeat step 4 while PTR! = NULL
Step 4	(a) CPT = LINK (PTR)
	(b) LINK (PTR) = TPT
	(c) TPT = PTR
	(d) PTR = CPT
Step 5	STOP.

Q.1.(d)(i) Convert the following in fix expression into a postfix using a stack.

$$\{a + (b - c)\}^* \{(d - e)/(\delta + g - h\}$$

Ans.

	Symbol Scanned	Stack	Express (Post fix)
1.	{	{	a
2.	a	{	a
3.	+	{+	a
4.	(	{+(	ab
5.	b	{+(	ab
6.	–	{+(–	abc
7.	(	{+(–	abc –
8.	)	{+	abc –
9.	}		abc – +
10.	*	*	abc – +
11.	{	*{	abc – +
12.	(	*{(	abc – +
13.	d	*{(	abc – + d
14.	–	*{(–	abc – + d
15.	e	*{(-	abc – + de
16.	)	*{	abc – + de –
17.	/	*{/	abc – + de –
18.	(	*{/(	abc – + de –
19.	δ	*{/(	abc – + de – f
20.	+	*{/(+	abc – + de – f
21.	g	*{/(+	abc – + de – fg

22.	–	*{ :/(+-	abc – + de – δg
23.	h	*|/(+–	abc – + de – fgh
24.	)	*|/	abc – + de – fgh – +
25.	}	–	abc – + de – fgh – + / *

Q.1.(d)(ii) If you are using C language to implement the heterogenous linked list, what pointer type will you use?

Ans. We use the void pointer to implement the heterogeneous linked list. The heterogenous linked list contain different data types in its nodes and we need a link, pointer, to connect them. Since we can't use ordinary pointer for this, we use the void pointer. Void pointer is a generic pointer type, and capable of storing pointer to any type.

Q.1.(e) What are threaded binary Trees? How are threads different from pointers? And how we can determine the next node to which a thread points?

Ans. In a linked representation of any binary tree, half of the entries in the pointer field LEFT and RIGHT will contain null entries. This space may be more efficiehtly used by replacing the null entries by special pointers called threads. Which points to the nodes higher in the trees. Such trees are called Threaded Tree.

Difference between threads and pointer: Threads in a binary tree must be distinguished from normal pointer. In a graphical representation of a threaded binary tree, threads are shown by dotted lines. In a computer memory, an extra field called tag or flag is used to distinguish a thread from a normal pointer.

Tree can be threaded using one way threading or two way threading. In a one way threading, a thread will appear in the right field of the node and will point to the successor node in the in order traversal of tree; and in two way threading of tree a thread will also appear in the left field of a node and will point to the preceding node in the in order traversal of tree.

Q.2.(a) Write an algorithm or pseudo code to extract characters from a given string from a given position for n characters and store it another string which is initialized at run time.

Ans.

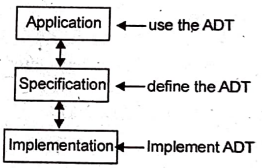
```
# include <stdio.h>
# define SIZE 100
    void print_str (char S []);
    void read_str (char S[]);
    main ( )
    {
    char str [SIZE];
    do {
      read_str (Str);
      print_str (Str);
   } while (Str [0]);
}
void read_str (char * S)
{
   int i;
   char c ;
   for (i = 0; (c = get char ( ) ) ! = '\n' ; i ++)
      S [i] = c;
      S[i] = NULL;
}
Void print_str (char*S)
{
    int i;
    for (i = 0; S [i]; i ++)
      putchar (S[i]); putchar ('\n');
      }
```


Q.2.(b) What is Abstract Data type and its implementation through data structures?

Ans. Abstract Data Type: Abstract data type is the specifications of the data type which specifies the logical and mathematical model of the delta type.

To implement a fraction data type:

```
Type day strict { int numerator, denomination; fraction; }
main ()
{
 fraction f;
 f. numerator = 1;
 f. denomination = 2;
 ....
}
```



When we use abstract data type, our program divide into two:

The Application: The part that use Abstract data type.

The Implementation: The part that implement the Abstract data type.

Q.2.(c) How will you check the validity of an expression containing nested parenthesis?

Ans. To check the validity of an expression containing nested parenthesis we use the stack. Stack works fine for this. The steps are:

(1) Maintain a stack of characters.

(2) Whenever you find opening braces '(', '{', or '[' push it on the stack.

(3) Whenever you find closing traces ')', '}', ']' check if top of stack is corresponding opening bracket, if yes then pop the stack, else break the loop and return false.

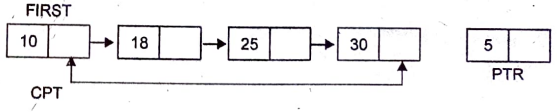
(4) Repeat step 2-3 until end of the string.

Q.3.(a) Write a pseudo code/algorithm for

(i) insertion at the end of a circular linked list.

(ii) Deletion of a node with a given data in a doubly linked list.

Ans.

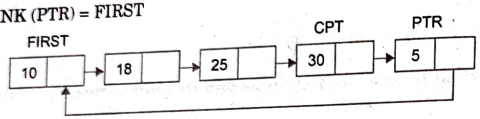


Repeat CPT = Link (CPT) till link (CPT) ≠ FIRST

i.e.

LINK (CPT) = PTR

LINK (PTR) = FIRST



At the end

1. If Avail = NULL then linked list is overflow and STOP.
2. PTR ← AVAIL
   AVAIl ← LINK (AVAIL)
   Read INFO (PTR)
3. CPT ← FIRST
4. Repeat step 5 while LINK (CPT) ! = FIRST
5. CPT ← LINK (CPT)
6. LINK (CPT) ← PTR
7. LINK (PTR) ← FIRST
8. STOP.

***(ii)* If a specific node is given.**

1. if FIRST = NULL then write UNDERFLOW and STOP.
2. Read DATA as information of node to be deleted.
3. PTR ← FIRST
4. Repeat step (5) while INFO (PTR) = DATA
5. PTR ← RPT (PTR)
6. CPT ← LPT (PTR)
   TPT ← RPT (PTR)
   RPT (CPT) ← TPT
   LPT (TPT) ← CPT
7. RPT (PTR) ← AVAIL
   AVAIL ← PTR
8. STOP.

Q.3.(*b*) How can we use linked list to store a sparse matrix?

Ans. The most advantage of linked list is manageability of memory and it is very important for large programs which are used to process a lot of data.

To represent a sparse matrix we use a linked list for each row and each column. So its needs two pointers : Next in this row and next in this column.

```
Struct node {
    node * next (01;
    node * next row;
    int row;
    int col;
    int value;
};
Class matrix
{
```



```
    node **colhead; // Head element of columns
    node ** rowhead; // the head element of rows
    int rows cols; // number of rows & columns of matrix;
    operations of the class;
}
```

Q.3.(*c*) Write a pseudo code/algorithm to remove duplicate element from a list?

Ans. Transverse the list from the head (or start) node. While Traversing, compare each node with its next node. If data of next node is same as current node then delete the next node. Before we delete a node, we need to store next pointer of the node.

Program:

```
# include < stdio.h>
# include < stdlib.h>
Struct node
    { int data;
    struct node * next;
};
Void remove Duplicates (Struct node * head)
{
    Struct node * current = head;
    Struct node * next – next;
    if (Current == NULL)
      return;
    While (Current > next! = NULL)
    {
      if (Currnet → 7 data = = current – > next – > data)
    {
    next – next = current – > next – > next;
      free (current _ next);
    }
    else
    {
      current = current – > next;
    }
  }
}
void Push (Struct node ** head – ref, int new-data)
{
    Struct node * new_node = (Struct node*) malloc (size of (Struct node));
      new_node – > next = (*head-ref);
      (*head-ref) = new-node;
    }
    Void printlist (Struct node * node)
    {
      While (node! = NULL)
      {
        printf ("/d", node – > data);
        node = node – > next;
      }
```

```
}
int main ()
{
    struct node * head = NULL;
     Push (& head, 20);
     Push (& head, 13);
     Push (& head, 13);
     Push (& head, 11);
     Push (& head, 11);
     Push (& head, 11);
  print f ("/In linked list before duplicate removal");
  print List (head);
  remove Duplicates (head);
  printf ("in linked list after duplicate Removal);
    printlist (head);
    getchar ();
}
```

Q.4.(a) Consider a float array APR (2 : 8, – 4 : 1), 6 : 10) whose base address is 200. Find the address of APR (5, – 1, 8]

Ans. APR (2 : 8, – 4 : 1), 6 : 10)

length of three dimensional of APR are

$4 = 8 - 2 + 1 = 7 \quad L_2 = 1 - (-4) + 1 = 6 \quad L_3 = 10 - 6 + 1 = 5$

Accordingly APR contains $L_1 : L_2 : L_3 = 7 \times 6 \times 5 = 210$ elements.

Suppose the programming language store APR in memory in row major order, and Base (APR) = 200 w = 4 words per memory cell.

The effective indices of the subscripts are:

$E_1 = 5 - 2 = 3 \quad E_2 = -1 - (-4) = 3 \quad E_3 = 8 - 6 = 2.$

For row major order, we have:

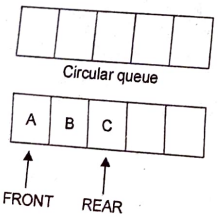
$$E_1L_2 = 3 \times 6 = 18$$
$$E_1L_2 + E_2 = 18 + 3 = 21$$
$$(E_1L_2 + E_2)L_3 = 21 \times 5 = 105$$
$$(E_1L_2 + E_3)L_3 + E_3 = 105 + 2 = 107$$

Add (APR [5, – 1, 8] = 200 + 4 (107) = 200 + 428 = 628.

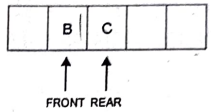
Q.4.(b) Consider a circular queue with fixed 5 position. Find the value of Front & REAR when following operations are performed

(i) A, B, C, inserted.
(ii) A is deleted.
(iii) D & E are inserted.
(iv) B & C deleted.
(v) F inserted.
(vi) D deleted.
(vii) G & H inserted.

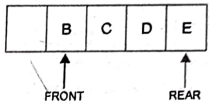
Ans. (i) A, B, C, inserted.



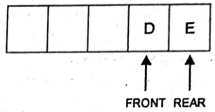
(ii) A is deleted



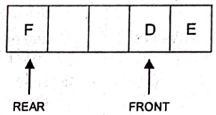
(iii) D & E are inserted.



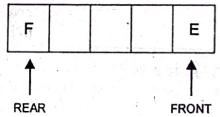
(iv) B & C deleted.



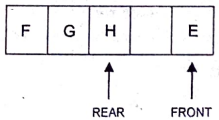
(v) F inserted



(vi) D is deleted.



(vii) G & H inserted.



Q.8. Write short notes on any two of the following:

(a) Bucket Hasing

Ans. Closed hashing stores all records directly in the hash table. Each record with the key value value K_R has a home position that is h (K_R). The slot computed by hash function.

When a record is inserted, the bucket to which it is mapped has available space to store the record. If the bucket does not have enough space, however it indicates an error condition called 'Bucket Overflow'. Bucket overflow occur done to following reasons.

Insufficient Buckets: The number of bucket which we denote by nb, must be choosen such that nb > nr/fr. Where nr denote total number of record.

fr denote the record that fit in the bucket.

Skewness: If the selection of a bucket, during insertion is more frequent than that of others, the bucket is said to be skewed as against being symmetrical.

• Bucket methods are good for implementing hash table stored on disk, because the bucket size can be set to the size of a disk block.

(b) Collision Resolution.

Ans. Suppose we want to add a new record R with the key K to our file F, but suppose the memory location address $H(k)$ is already occupied. This situation is called collision.

(1) Collision resolution by open addressing.

(2) Collision resolution by separate chaining.

Open Addressing: In this, all elements are stored in the hash table itself. That is each table entry contains either an element of the dynamic set or NIL.

Advantage: The advantage of open addressing is that it avoids pointers together. Instead of following pointers, we compute the sequence of slot to be examined. The extra memory freed by not storing pointers provides the hash table with a large number of slots for the same amount of memory.

Hashing with Changing: This method maintain a chain of elements which has the same hash address. We can take the hash table as an array of pointers. Size of hash table can be number of records. Here each pointer will point to one linked list and the elements which have the same hash address will be maintained in the linked list. We can maintained the linked list in sorted order and each elements of linked list will contain the whole record with key.

Advantage: The main advantage of this hashing is saving of memory space and perfect collision resolution.

• Here hash table contain only pointer which point to linked list-containing records.

Disadvantage: The main disadvantage is also wastage of memory space. In case where records are very small or contains only key.

(c) Complexity of program.

Ans. The complexity of an algorithm is a function $f(n)$ which measure the time and for space used by an algorithm in terms of input size n.

The choice of particular algorithm depends on following performance analysis and measurements.

(1) Space complexity: Analysis of space complexity of an algorithm or program is the amount of memory it needs to run to complete.

The space needed by the program consist of following components.

(1) Instruction space (2) Data space (3) Environment stack space.

(2) Time complexity: Time complexity of an algorithm is the amount of time it needs to run to completion. The exact time will depend on the implementation of the algorithm programming language, optimising the capability of the compiler used, the CPU speed, other hardware and so on.

• To measure the time complexity accurately, we have to count all sorts of operations performed in an algorithm.

• The time complexity also depend on the amount of data input to an algorithm.

FIRST TERM EXAMINATION [SEPT.-2015]
THIRD SEMESTER [B. TECH]
DATA STRUCTURE [ETCS-209]

Time: 1½ Hrs. **MM : 30**

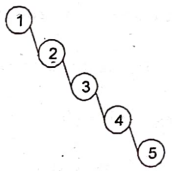
Note: *Attempt Q.No.1 which is compulsory any two more questions .*

Q.1. (a) How the resulting BST will look if the input is already sorted in non-decreasing order. **(1)**

Ans. If all nodes are already sorted in non decreasing order then all nodes are inserted into the right side of a binary search tree, for example →

nodes are 1, 2, 3, 4, 5

the BST will look like →



Q.1. (b) Construct Binary tree T from following sequence. **(2)**

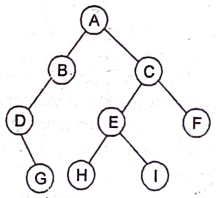
PREORDER: A B D G C E H I F

INORDER: D G B A H E I C F

Ans.

PREORDER: ABDGCEHIF

INORDER: DGBAHEICF



Q.1. (c) Number of nodes in complete tree is 1000000. Find its depth. **(1)**

Ans. The depth of complete binary tree is log(n) where no. of nodes is n. So the depth of complete binary tree which has the 1000000 nodes is.

$$\log (1000000) = 6$$

Q.1. (d) Write algorithm/program to perform POP operation on stack using singly linked list. **(2)**

Ans. Void pop (c)

```
{
    Struct node * ptr;
    If (top = = NULL)
{ printf ("Underflow")
 return;
}
ptr = top;
top = ptr → link;
free (ptr);
}
```

Q.1. (e) What is the difference between ground header linked list and circular header linked list? (1)

Ans. A grounded header list is a header list where the last node contains the null pointer.

A circular header list is a header list where the last node points back to the header node.

Q.1. (f) What is the condition that a circular queue is full if the queue is implemented using arrays? (1)

Ans. If ((front = = 0) && (rear == Max – 1) || (front == rear + 1))

that means queue is overflow or full.

Q.1. (g) Binary tree with N nodes has exactly ..... Null branches (1)

Ans. Binary tree with N nodes has exactly $n + 1$ null branches.

Q.1. (h) What is Time Space Trade off?

Ans. Time-Space tradeoff: Is a way of solving a problem calculation in less time by using more space (or memory), or by solving a problem in very little space by spending a long time. Most computers have a large amount of space, but not infinite space. Also, most people are willing to wait a little while for a big calculation, but not forever. So if your problem is taking a long time but not much memory, a space-time tradeoff would let you use more memory and solve the problem more quickly or it could be solved very quickly but requires more memory than you have, you can try to spend more time solving the problem in the limited memory.

Q.2. (a) Why height balancing tree is required? Create AVL tree of following. a, z, b, y, c, x, d, w, e, v, f. (8)

Ans. The first balanced binary search tree or height balancing tree was the AVL tree with has an additional balancing condition. The simplest idea is to require that the left and right subtrees have the same height, so height balancing tree is required.

AVL tree of nodes– *a, z, b, y, c, x, d, w, e, v, f.*

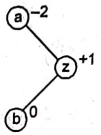
Creation of AVL Tree

Nodes: *a, z, b, y, c, x, d, w, e, v, f*

Insert *a*

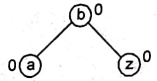
Insert *z*

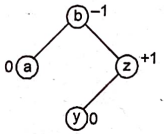
Insert *z*



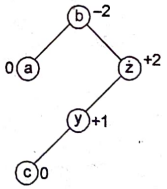
There is Balance factor of a is –2 so after *R-L* rotation the tree is.

Insert *y*

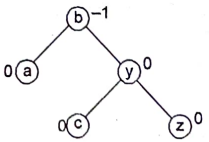




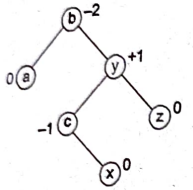
Insert *c*



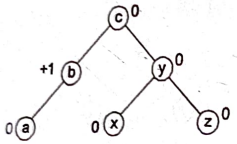
There is Balance factor of *z* is + 2 so after $L - L$ rotation the tree is –



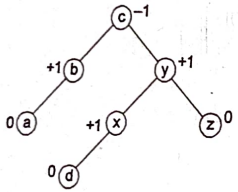
Insert *x*



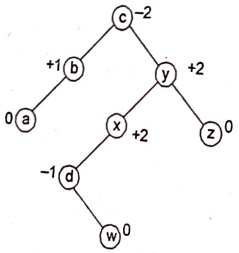
There is the Balance factor of *b* is – 2 So after *L–R* rotation the tree is–



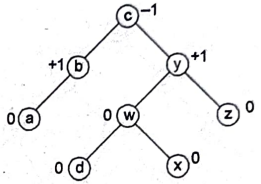
Insert *d*



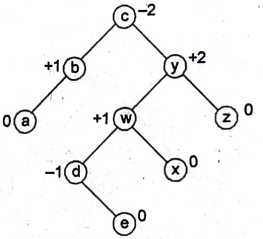
Insert *w*



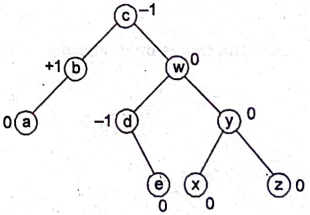
After Insertion of *w* the Balance factor is unbalanced so after applying the *L-R* rotation the tree is–



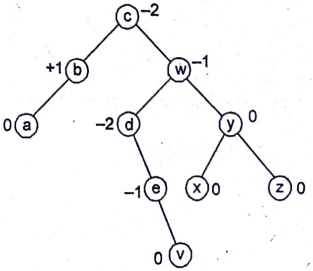
Insert *e*



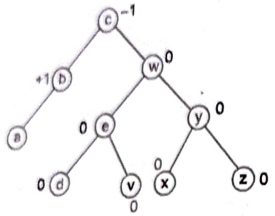
After Insertion of node *e* the tree is unbalanced so after applying the *L-L* rotation the tree is–



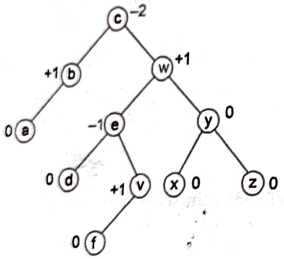
Insert *v*



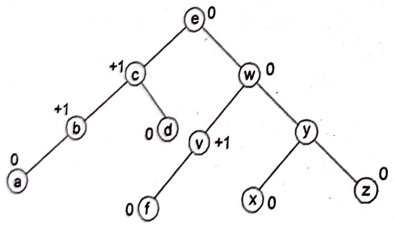
After Insert the node *v* the tree is unbalanced so after apply the R.R rotation the tree is–



Insert *f*



After Insert the node *f* the tree is unbalanced so after apply the *R–L* rotation the final tree is–



Q. 2. (*b*) Write a procedure/program to reverse a singly linked without using any more memory? **(2)**

Ans. Reverse a singly linked list.

Step 1 PTR = FIRST

Step 2 TPT = NULL

Step 3 Repeat step 4 while PTR! = NULL

Step 4 (a) CPT = LINk (PTR)

(*b*) LINK (PTR) = TPT

(*c*) TPT = PTR

(*d*) PTR = CPT

Step 5 Stop.

Q.3. (*a*) Consider the following in fix expression and convert into reverse polish notation using stack. **(5)**

Ans. (A + (B*C–(D/E^F)*G)*H)

Character Scanned	Stack	Reverse polish Notatic Postfix Notation
(	(	
A	(	A
+	(+	A
(	(+(	A
B	(+(	AB
*	(+(*	AB
(	(+(*	ABC
–	(+(–	ABC*
(	(+(–(	ABC*
D	(+(–(	ABC* D
/	(+(–(/	ABC * D
E	(+(–(/	ABC * DE
^	(+(–(/^	ABC * DE
F	(+(–(/^	ABC * DEF
)	(+(–	ABC * DEF ^
*	(+(–*	ABC * DEF ^ /
G	(+(–*	ABC * DEF ^ / G
)	(+	ABC * DEF ^ / G * –
*	(+*	ABC * DEF ^ / G * –
H	(+*	ABC * DEF ^ / G * –
)	—	ABC * DEF ^ / G * – H * +

Reverse polish notation

Q.3. (*b*) What is an algorithm/program for inserting and deleting a node at a given location in circular linked list. **(5)**

Ans. Inserting a node at given location.

Step 1. If AVAIL = NULL then write overflow and stop

Step 2. Read DATA as information of node after which insertion will be made.

Step 3. PTR = AVAIL
AVAIL = LINK (AVAIL)
Read Info (PTR)

Step 4. CPT = FIRST

Step 5. Repeat Step 6 while Info (CPT) ! = DATA

Step 6. CPT = LINK (CPT)
Step 7. LINK (PTR) = LINK (CPT)
LINK (CPT) = PTR
Step 8. Stop.

Deletion of node at given Location.

Step 1. If FIRST = NULL then write underflow and stop.
Step 2. Read DATA as information of node to be deleted.
Step 3. PTR = FIRST
Step 4. Repeat step 5 while info (PTR) ≠ DATA
Step 5. CPT = PTR
PTR = LINK (PTR)
Step 6. LINK (CPT) = LINK (PTR)
Step 7. LINK (PTR) = AVAIL
AVAIL = PTR
Step 8. Stop.

Q.4. (a) Suppose multidimensional arrays A and B are declared as A(-2:2, 2:22) and B (1:8, -5:5, -10:5) stored in column major order.

(i) Find the length of each dimensions of A and B. **(2.5)**

(ii) The number of elements in A and B. **(1)**

(iii) Consider the elements B(3, 3, 3) in B. Find the effective indices E1, E2, E3, and the address of the element assuming Base(b) = 400 and there are w = 4 words per memory location. **(2.5)**

Ans. (i) The length of a dimension is obtained by:

Length = upper bound – Lower bound + 1

Hence the length Li of the dimensions of A are:

L1 = 2 – (–2) + 1 = 5 and L2 = 22 – 2 + 1 = 21

L1 = 8 – 1 + 1 = 8, L2 = 5 – (–5) + 1 + 11, L35 – (–10) + 1 = 16

(ii) Accordingly A has 5*21 = 105 elements and B has 8*11*16 = 1408 elements.

(iii) The effective index Ei is obtained from Ei = ki–LB, where ki is the given index and LB is the lower bound.

Hence E1 = 3 – 1 = 2, E2 = 3 – (–5) = E3 = 3 – (–10) = 13

Considering *B* is stored in column-major order, the address of *B* [3,3,3] will be calculated as.

E3L2 = 13*11 = 143, E2L2 + E2 = 143 + 8 = 151

(E3L2 + E2) L1 = 151*8 = 1208, (E3L2 + E2) L1 + E1 = 1208 + 2 = 1210

Therfore address of B [3,3,3] = 400 + 4(1210) = 400 + 4840 = 5240.

Q.4 (b) What is Garbage Collections?

Ans. Gabage Collection: Garbage collection (GC) is a dynamic approach to automatic memory management and heap allocation that processes and identifies dead memory block and reallocates storage for reuse. The primary purpose of garbage collection is to reduce memory leaks.

Q.4. (c) Write algorithm/program for insertion operation on queue using Singly linked list.

Ans.

```
void insert ()
{
struct node * ptr;
ptr = (struct node *) malloc (size of (struct node));
int item;
prinf ("Input the element for Inserting");
scanf ("/d", and item);
ptr → info = item;
ptr → Link = NULL;
If (front = = NULL)
  front = Ptr;
else
rear → link = ptr;
rear = ptr;
}
```

SECOND TERM EXAMINATION [NOV.-2015]
THIRD SEMESTER [B. TECH]
DATA STRUCTURE [ETCS-209]

Time: 1½ Hrs. MM : 30

Note: *Attempt Q.No.1 which is compulsory any two more questions .*

Q.1. (a) Difference between B– tree and B+ tree? (2)

Ans. Original B-tree had record pointers in all of the index nodes; B+ tree only in leaf nodes.

Given a key K and the two node pointers L and R around it.

→ All key K and the two node pointer L and R around it

1. All key values pointed to by L are < K.
2. All key values pointed to by R and > = K.

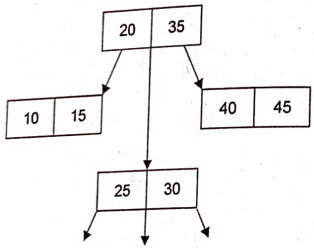
→ B+ tree data pages are linked together to form a sequential file.

Q.1. (b) Construct 3–way search tree for the following list of keys.

List: 20, 35, 40, 10, 15, 25, 30, 45

Ans. List: 20, 35, 40, 10, 15, 25, 30, 45

3 way Search tree.



Q.1. (c) Consider the following specification of a graph.

V(G) = {1, 2, 3, 4, 5} and E(G) = {(1, 2), (1, 3), (3, 3), (3, 4), (4, 1), (4, 5), (5, 2)} Draw its adjacency matrix. (1)

Ans. V(G) = {1, 2, 3, 4, 5}

E(G) = {(1, 2), (1, 3), (3, 3), (3, 4), (4, 1), (4, 5), (5, 2)}

Adjacency Matrix Representation.

	1	2	3	4	5
1	0	1	1	0	0
2	0	0	0	0	0
3	0	0	1	1	0
4	1	0	0	0	1
5	0	1	0	0	0

Q.1. (d) Compare the complexities in average case and worst case of the following sorting:

Merge sort, Heap sort, Quick sort, Selection sort

Ans.

Name	Best	Average	Worst
Quick Sort	N logN	N logN	N^2
Merge Sort	N logN	N logN	N logN
Heap Sort	N logN	N logN	N logN
Selection Sort	N^2	N^2	N^2

Q.1. (e) What is pseudorandom hashing method?

Ans. Pseudorandom Hashing

A common random-number generator is shown below.

$$y^* = ax + c$$

To use the pseudorandom-number generator as a hasing method, we set x to the key, multiply it by the coefficient a, and then add the constant c. The result is then divided by the list size, with the remainder being the hashed address.

Example:

$$Y = ((17 \times 122267) \text{ modulo } 307$$
$$Y = (2061539 + 7) \text{ modulo } 307$$
$$Y = 2061546$$
$$Y = 41$$

Q.2. (a) Construct B-TREE of order 5 by inserting the following elements. 3, 14, 7, 1, 8, 5, 11, 17, 13, 6, 23, 12, 20, 26, 4, 16, 18, 24, 25 and 19. And also delete values 6, 23, 3 from above constructed tree.

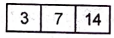
Ans. B-Tree of order 5:

Elements: 3, 14, 7, 1, 8, 5, 11, 17, 13, 6, 23, 12, 20, 26, 4, 16, 18, 24, 25, 19

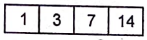
Insert 3

Insert 14

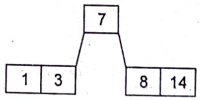
Insert 7



Insert 1

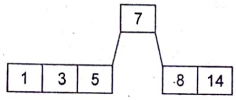


Insert 8

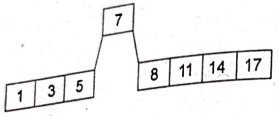


Here node was already full 50 after insert 8, it is splitted into 2 node and 7 is the median key.

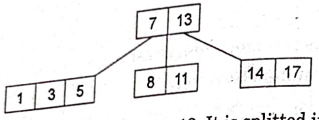
Insert 5



Insert 11 and 17

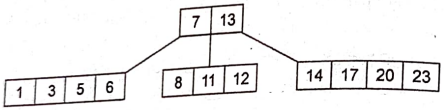


Insert 13

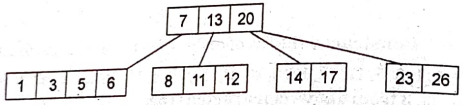


Here node was already full so after insert 13, It is splitted into 2 nodes and 13 is the median key.

Insert 6, 23, 12, 20

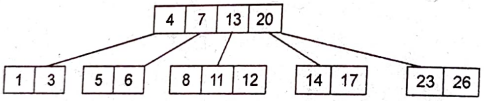


Insert 26



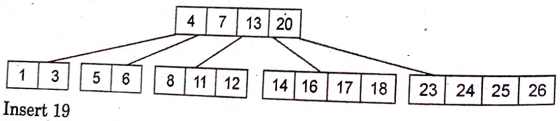
Here node was already full, so it is splitted into 2 nodes and 20 is the median key.

Insert 4

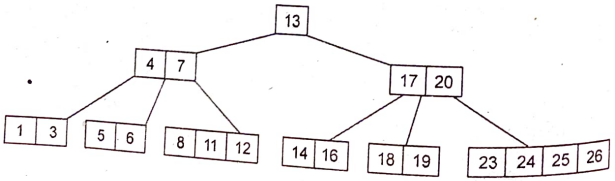


Here node was already full so after Insert 4, It is splitted into 2 nodes and 4 is the median key.

Insert 16, 18, 24, 25

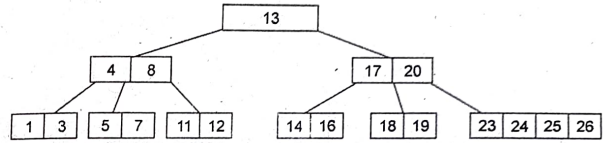


Insert 19

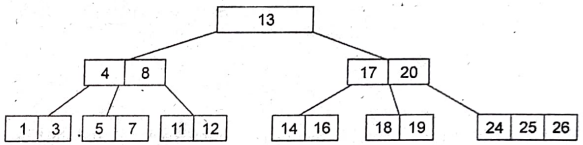


Here node was already full so after Insert 19 it is splitted, into 2 nodes. 17 is the median key so it will go into parent node. Here perent node is already full, so it is splitted in 2 nodes and 13 is the median key and it will become the new root.

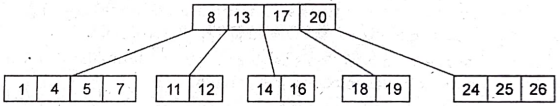
After delete the node 6



After delete the node 23



After delete the node 3



Q.2. (*b*) Write algorithm/program of depth first search for traversal of graph?

Ans. Step 1: Set status = 1 (ready state) for each node in G.

Step 2 : Push the starting node A on the stack and set its status = 2 (writing state).

Step 3 : Repeat steps 4 and 5 until stack is empty.

Step 4 : Pop the top Node N. Process it and set its status = 3 (processed state).

Step 5 : Push on to the stack all the neighbour of N that are in the ready state (whose status = 1) and set their status = 2 (waiting state).

Step 6 : Exit.

Q.3. (*a*) Suppose the table T has 11 memeory locations T {1}, T{2},.....T[11] and suppose the File F consist of 8 records *A, B, C, D, E, X, Y, Z* Record: *A, B, C, D, E, X, Y, Z* and its hash address U(K): 4, 8, 2, 11, 4, 11, 5, 1 resp. How file F will appear in memory when 8 records are entered into table T in above order using linear probing and quadratic probing collision resolution technique? Find average successful search probes (*S*) and average unsuccessful search probes (S) and average unsuccessful search probes (*U*) also for both? (6)

Ans.

Record	A,	B,	C,	D,	E,	X,	Y,	Z
H (*k*)	4,	8,	2,	11,	4,	11,	5,	1

Suppose the 8 records are entered into the table T in above order using the liner probing and Quadratic probing collision resolution technique then the file F will appear in memory as follows.

Table T:	X,	C,	Z,	A,	E,	Y,	–,	B,	–,	–,	D
Address:	1,	2,	3,	4,	5,	6,	7,	8,	9,	10,	11

Although Y is the only record with hash address $H(k) = 5$, the record is not assigned to T [5], since T [5] has already been filled by E because of a previous collision at T(4). Similarly Z does not appear in T (1).

The average no. S of probes for a successful search follows:

$$S = \frac{1+1+1+1+2+2+2+3}{8}$$

$$= \frac{13}{8} = 1.6$$

The average no. U of probes for an unsuccessful search follows.

$$U = \frac{7+6+5+4+3+2+1+2+1+1+8}{11}$$

$$= \frac{40}{11} = 3.6$$

The first sum adds the no. of probes to find each of he 8 records, and the second sum adds the no. of probes to find as empty location for each of the 11 locations.

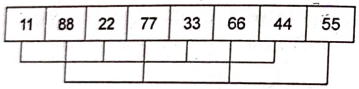
Q.3. (b) Difference between internal sorting and external sorting? Sort the following elements using shell sort: 11, 88, 22, 77, 33, 66, 44, 55. (1+3)

Ans. Internal sorting: Which deals with sorting the data stored in computer's main memory. Example: Bubble, selection etc. sorting.

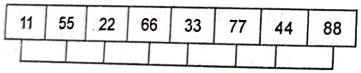
External sorting: Which deals with sorting the data stored in files. External sorting is applied when data can not be stored in the memory. Example: Files and its organization.

Shell sort: 11, 88, 22, 77, 33, 66, 44, 55

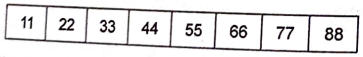
Input to pass 1 with distance = 2



Output of pass 1 is input to pass 2 and distance = 1



Output of pass 2 is



Q.4. (a) Consider a company with 68 employee. Each has been assigned a 4 digit employee number which has been assigned a 4 digit employee number which is used as the primary key in the company's employee file. Suppose L (set of memory addresses of the location) consist of 100 two digit addresses 00,01,02---99. Then apply the division method using prime no. closest to 99, mid square method and folding method to find out 2-digit hash address for each of the following employee no. 9614, 5882, 1825. (2+2+2)

Ans. (i) Division Method: Choose a prime no. M close to 99, such as M = 97 Then.

H (9614) = 11, H (5882) = 62 H (1825) = 79

That is dividing 9614 by 97 gives a remainder 11. dividing 5882 by 97 gives a remainder 62 and so on. In the case that the memory address begin with 01 rather than 00, we choose that the function H(k) = k (mod M) + 1 to obtain:

H (9614) = 11 + 1 = 12, H (5882) = 62 + 1 = 63 H (1825) = 79 + 1 = 80

(ii) Midsquare Method:

K	9614	5882	1825
k^2	92428996	34597924	3330625
H(k)	28	97	06

Observe 4th and 5th digit are choosen for Hash address.

(iii) Folding Method: Choping the keys K into 1, 2, 1 digits and add them. So the address will be:

H (9614) = 9 + 61 + 4 = 74

H (5882) = 5 + 88 + 2 = 95

H (1825) = 1 + 82 + 5 = 88

Q.4. (b) Write algorithm/program of anyone sorting technique which is used when input elements are small? Difference between comparison based sorting and non-comparison based sorting? (3+1)

Ans. When Input elements are small then many sorting techniques may be used such as, bubble, sort, insertion sort, selection sort etc.

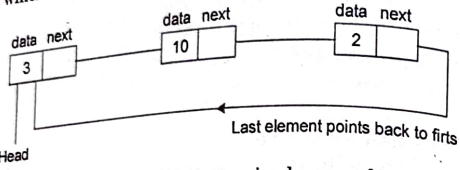
Algorithm of Selection Sort

1. n = length [A]
2. For $j = 1$ to $n - 1$
3. smallest = j
4. for $i = j + 1$ to n
5. If $A[i] < A$ smallest]
6. then smallest = i
7. exchange (A [j], A [smallest])

Comparison based and non comparison based sorting

Comparison based sorting is a type of soring that only reads the list of elements through a single abstract comparison operation that determines which of 2 elements should occur first in the final sorted list. While in non comparison based sorting algorithm there are no need of comparison between elements. Example of comparison based sorting are bubble, insertion, selection etc and radix sort is the non comparison based sorting algorithm.

each other in a circular way which forms a circular chain. The circular linked list has a dynamic size which means the memory can be allocated when it is required.



Circular lists are usually preferable to non-circular ones, for two reasons:

• They provide easy access to both ends of a list: notice that we do not need the first pointer in the header, because we can easily reach the first node with L->last->next

• The code for processing a circular list is often simpler than the code for processing a non-circular list-mainly because you don't have to constantly check if the next node is defined-it is always defined.

FIRST TERM EXAMINATION [SEPT. 2016]
THIRD SEMESTER [B.TECH]
DATA STRUCTURE [ETCS–209]

Time : 1.30 hrs. **M.M. : 30**

Note: *Attempt any three questions including Q.no. 1 which is compulsory.*

Q.1. (a) What are two main factors for measuring the performance of algorithm? Explain in brief. **(2)**

Ans. Space and Time complexities are the parameters by which we can analyze the performance of a program or algorithm.

Space Complexity: It is the amount of memory it needed to run to completion. It has a fixed pact that includes space for the cock, space for simple variable and fixed size component variables, constants etc. and a variable part that includes the space needed by component variable whose size is dependent on the particular problem instance being solved and the stack space.

Time Complexity: It is the amount of computer time needs to run to completion. It is the sum of the compile time & the run time. The compile time does not depend on the instance characteristics major concern is the run time of a program.

Q.1. (b) What do you mean by garbage collection? **(2)**

Ans. Garbage Collection: Garbage collection (GC) is a dynamic approach to automatic memory management and heap allocation that processes and identifies dead memory block and reallocates storage for reuse. The primary purpose of garbage collection is to reduce memory leaks.

Q.1. (c) Define time-space trade off? **(2)**

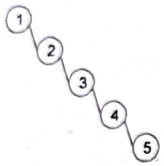
Ans. Time-Space tradeoff: Is a way of solving a problem calculation in less time by using more space (or memory), or by solving a problem in very little space by spending a long time. Most computers have a large amount of space, but not infinite space. Also, most people are willing to wait a little while for a big calculation, but not forever. So if your problem is taking a long time but not much memory, a space-time tradeoff would let you use more memory and solve the problem more quickly or it could be solved very quickly but requires more memory than you have, you can try to spend more time solving the problem in the limited memory.

Q.1. (d) How the resulting binary search tree will look like if the input is already sorted in non-decreasing order. **(2)**

Ans. If all nodes are already sorted in non decreasing order then all nodes are inserted into the right side of a binary search tree, for example →

nodes are 1, 2, 3, 4, 5

the BST will look like →



Q.1. (c) Number of nodes in a complete tree is 1000000. Find its depth? (2)

Ans. The depth of complete binary tree is log(n) where no. of nodes is n. So the depth of complete binary tree which has the 1000000 nodes is.

$$\log (1000000) = 6$$

Q.2. (a) Write a program/algorithm to evaluate a postfix expression using stack? Consider the following infix expression and convert into postfix expression using stack. (A + (B*C – (D\E ^ F)*G)*H) (2+3)

Ans. Algorithm for Evaluation of Postfix Expression

Create an empty stack and start scanning the postfix expression from left to right.

- If the element is an operand, push it into the stack.
- If the element is an operator O, pop twice and get A and B respectively. Calculate BOA and push it back to the stack.
- When the expression is ended, the value in the stack is the final answer.

Character Scanned	Stack	Reverse polish Notatic Postfix Notation
(	(	
A	(	A
+	(+	A
(	(+(	A
B	(+(	AB
*	(+(*	AB
(	(+(*	ABC
–	(+(–	ABC*
(	(+(–(	ABC*
D	(+(–(	ABC* D
/	(+(–(/	ABC * D
E	(+(–(/	ABC * DE
^	(+(–(/^	ABC * DE
F	(+(–(/^	ABC * DEF
)	(+(–	ABC * DEF ^
*	(+(–*	ABC * DEF ^ /
G	(+(–*	ABC * DEF ^ / G
)	(+	ABC * DEF ^ / G * –
*	(+*	ABC * DEF ^ / G * –
H	(+*	ABC * DEF ^ / G * –
)	—	ABC * DEF ^ / G * – H * +
		Reverse polish notation

Q.2. (b) Write a procedure/program to reverse a singly linked list without using any more memory? (5)

Ans. Refer Q.2. (a) End Term Examination 2016.

Q.3. (a) Suppose multidimensional arrays A and B are declared as A (–2.2, 2.22) and B(1 : 8, –5 : 5, –10 : 5) stored in column major order. (2+2+3)

***(i)* Find the length of each dimensional of A and B**

***(ii)* The number of elements in A and B.**

***(iii)* Consider the elements B[3, 3, 3] in B. Find the effective indices E1, E2, E3 and the address of the element assuming Base (b) =400 and there are w=4 words per memory location.**

Ans. ***(i)*** The length of a dimension is obtained by:

Length = upper bound – Lower bound + 1

Hence the length Li of the dimensions of A are:

L1 = 2 – (–2) + 1 = 5 and L2 = 22 – 2 + 1 = 21

L1 = 8 – 1 + 1 = 8, L2 = 5 – (–5) + 1 + 11, L35 – (–10) + 1 = 16

(ii) Accordingly A has 5*21 = 105 elements and B has 8*11*16 = 1408 elements.

(iii) The effective index Ei is obtained from Ei = ki–LB, where ki is the given index and LB is the lower bound.

Hence E1 = 3 – 1 = 2, E2 = 3 – (–5) = E3 = 3 – (–10) = 13

Considering *B* is stored in column-major order, the address of *B* [3,3,3] will be calculated as.

E3L2 = 13*11 = 143, E2L2 + E2 = 143 + 8 = 151

(E3L2 + E2) L1 = 151*8 = 1208, (E3L2 + E2) L1 + E1 = 1208 + 2 = 1210

Therfore address of B [3, 3, 3] = 400 + 4(1210) = 400 + 4840 = 5240.

Q.3. (b) What is an algorithm/program for deleting a node at a given location in circular linked list. **(3)**

Ans.

Deletion of node at given Location.

Step 1. If FIRST = NULL then write underflow and stop.

Step 2. Read DATA as information of node to be deleted.

Step 3. PTR = FIRST

Step 4. Repeat step 5 while info (PTR) ≠ DATA

Step 5. CPT = PTR

PTR = LINK (PTR)

Step 6. LINK (CPT) = LINK (PTR)

Step 7. LINK (PTR) = AVAIL

AVAIL = PTR

Step 8. Stop.

Q.4. (a) Why height balancing of tree is required? Create AVL tree of following:

a, z, b, y, c, x, d, w, e, v, f **(2+6)**

Ans. The first balanced binary search tree or height balancing tree was the AVL tree with has an additional balancing condition. The simplest idea is to require that the left and right subtrees have the same height, so height balancing tree is required.

AVL tree of nodes– *a, z, b, y, c, x, d, w, e, v, f.*

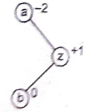
Creation of AVL Tree

Nodes: *a, z, b, y, c, x, d, w, e, v, f*

Insert *a*

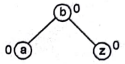
Insert *z*

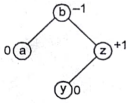
Insert *z*



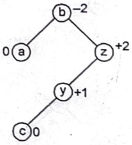
There is Balance factor of a is –2 so **after *R-L* rotation the tree is.**

Insert *y*

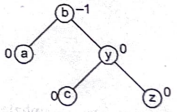




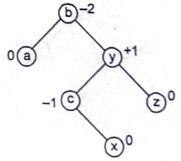
Insert *c*



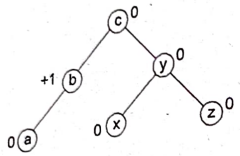
There is Balance factor of *z* is + 2 so after *L* – *L* rotation the tree is –



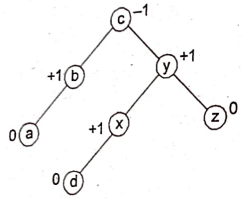
Insert *x*



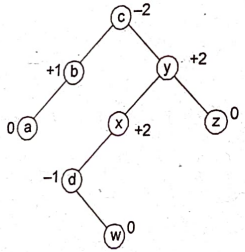
There is the Balance factor of b is – 2 So after L–R rotation the tree is–



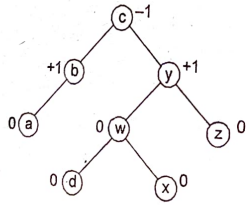
Insert d



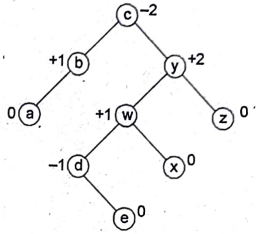
Insert w



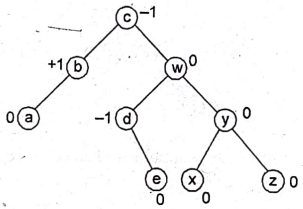
After Insertion of w the Balance factor is unbalanced so after applying the L-R rotation the tree is–



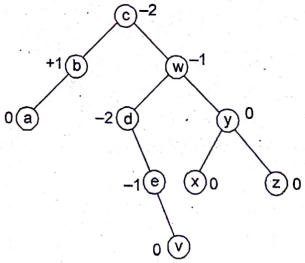
Insert e



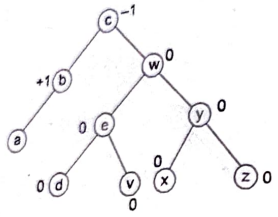
After Insertion of node e the tree is unbalanced so after applying the L-L rotation the tree is–



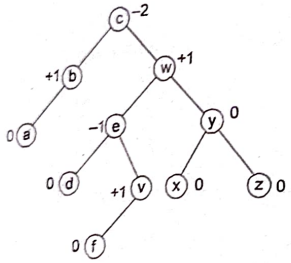
Insert v



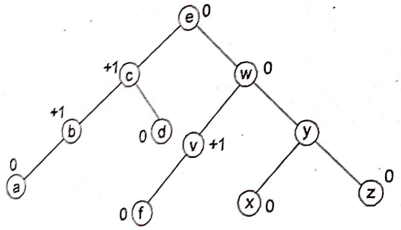
After Insert the node v the tree is unbalanced so after apply the R.R rotation the tree is–



Insert *f*



After Insert the node *f* the tree is unbalanced so after apply the *R–L* rotation the final tree is–



Q.4. (b) Following sequence gives the preorder and inorder of the Binary Tree T. (2)

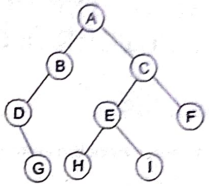
PREORDER : ABDGCEHIF

INORDER : DGBAHEICE

Construct a binary tree for the above sequence.

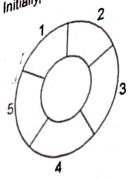
Ans. PREORDER:ABDGCEHIF

INORDER:DGBAHEICF

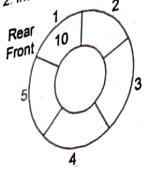


Example: Consider the following circular queue with N = 5.

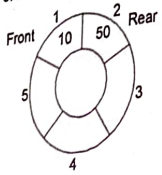
1. Initially, Rear = 0, Front = 0.



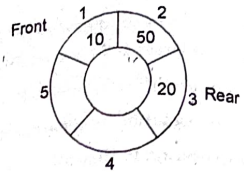
2. Insert 10, Rear = 1, Front = 1.



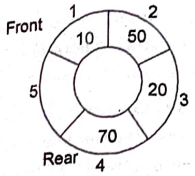
3. Insert 50, Rear = 2, Front = 1.



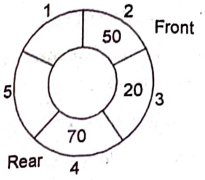
4. Insert 20, Rear = 3, Front = 0.



5. Insert 70, Rear = 4, Front = 1.



6. Delete front, Rear = 4, Front = 2.



FIRST TERM EXAMINATION [SEPT. 2017]
THIRD SEMESTER [B.TECH]
DATA STRUCTURES [ETCS–209]

Time : 1½ hrs. **M.M. : 30**

Note: *Q.1 is compulsory. Attmept any two questions from the remaining:*

Q.1. Attempt all parts:

Q.1. (*a*) Convert the given infix into post-fix expression. Demonstrate the steps of conversion. **(2)**

a + b* c– (d/e^f)*g)*h

Ans. Infix into Postfix expression of a + (b* c – (d/e^f)*g)*h:

Character Scanned	Stack	Postfix Notation
a		a
+	+	a
(	+(	a
b	+(	ab
*	+(*	ab
c	+(*	abc
–	+(–	abc*
(	+(–(	abc*
d	+(–(	abc*d
/	+(–(/	abc*d
e	+(–(/	abc*de
^	+(–(/^	abc*de
f	+(–(/^	abc*def
)	+(–	abc*def^/
*	+(–*	abc*def^/
g	+(–*	abc*def^/g
)	+	abc*def^/g*-
*	+*	abc*def^/g*-
h	+*	abc*def^g*-h
)		abc*def^/g*-h*+

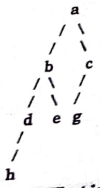
Q.1. (*b*) What is the maximum height of any AVL trees with 7 nodes? Assuming height is 0 for a tree with single node. **(2)**

Ans. The maximum height of any AVL-tree with 7 nodes is 3.

AVL trees are binary trees with the following restrictions.

1) the height difference of the children is at most 1.

2) both children are AVL trees



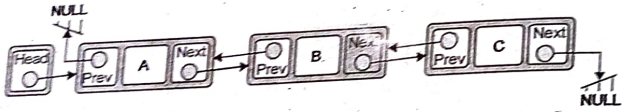
Q.1. (c) What is a doubly linked list? (2)

Ans. Doubly Linked List :

Doubly Linked List is a variation of Linked list in which navigation is possible in both ways, either forward and backward easily as compared to Single Linked List. Following are the important terms to understand the concept of doubly linked list.

- **Link** - Each link of a linked list can store a data called an element.
- **Next** - Each link of a linked list contains a link to the next link called Next.
- **Prev** - Each link of a linked list contains a link to the previous link called Prev.
- **LinkedList** - A Linked List contains the connection link to the first link called First and to the last link called Last.

Doubly Linked List Representation



Q.1. (d) What pointer type is used to implement a heterogneous list in C? Explain in brief? (2)

Ans. The heterogeneous linked list contains different data types in its nodes and we need a link, pointer to connect them. It is not possible to use ordinary pointers for this. So we go for void pointer. Void pointer is capable of storing pointer to any type as it is a generic pointer type.

Q.1. (e) Explain the following terms used in tree: (2)

(i) Terminal Node (ii) Siblings

(iii) Degree of Node (iv) Height of tree

Ans. (i) Terminal Node : In a tree data structure, the node which does not have a child is called as **LEAF Node**. In simple words, a leaf is a node with no child.

(ii) Siblings : In a tree data structure, nodes which belong to same Parent are called as **SIBLINGS**. In simple words, the nodes with same parent are called as Sibling nodes.

(iii) Degree of Node : In a tree data structure, the total number of children of a node is called as **DEGREE** of that Node. In simple words, the Degree of a node is total number of children it has.

(iv) Height of tree : In a tree data structure, the total number of egdes from leaf node to a particular node in the longest path is called as **HEIGHT** of that Node. In a tree, height of the root node is said to be **height of the tree**. In a tree, **height of all leaf nodes is '0'.**

Q.2. Attempt all parts:

Q.2. (a) Calculate the address of element at X[4, 3] in a 2D array X[1.5][1.4] stored in a row major order. Assume the base address to be 1000 and each element requires 4 words of storage. (5)

Ans. Row Major Order

LOC [A[i] [j]] = Base (A) + w[n (i – Lower bound for row index) + (j – lower bound for column index)]

Base (X) = 1000, w = 4, n = 4

So the address of X [4.3] = 1000 + 4 [4 (4 – 1) + (3 – 1)]

= 1000 + 4[4 × 3 + 2] = 1000 + 4 × 14 = 1056.

Q.2. (b) Write an algorithm to delete the last node of the linked list. (5)

Ans. Step 1: If linked list is empty (START == NULL)

Write "Sorry Buddy there is no node

Step 2: If linked list have only one node (START -> NEXT== NULL)

free(START)

START = NULL

Step 3: If the list have more than one node

PTR = START

while(PTR -> NEXT != NULL)

{

PREPTR = PTR

PTR = PTR -> NEXT

}

PREPTR -> NEXT = NULL

free(PTR)

Step 4 : EXIT

Q.3. Attempt all parts: (3 + 7=10)

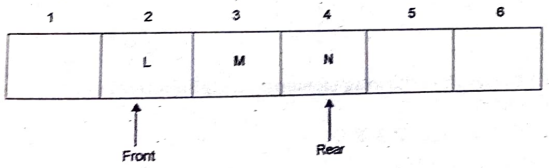
Q.3. (a) Consider the following circular queue with a storage size 6.

FRONT = 2, REAR = 4, QUEUE: _, L,M,N _,_

Describe the queue as following operations takes place:

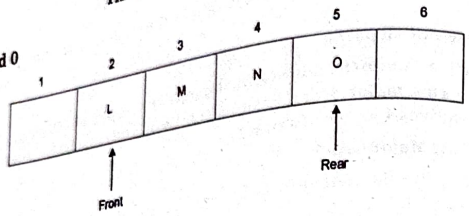
1. Add O 2. Add P 3. Delete two elements 4. Add Q,R,S 5. Delete one. (3)

Ans. Initially

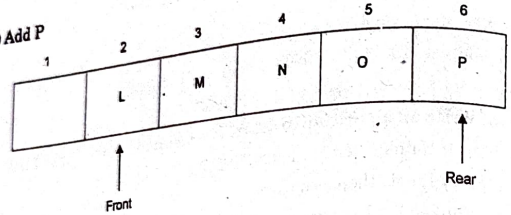


4–2017

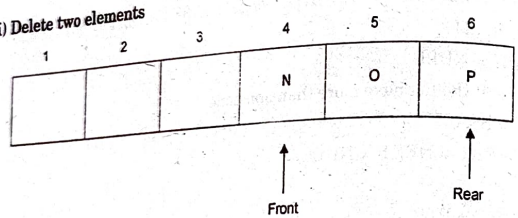
(i) Add 0



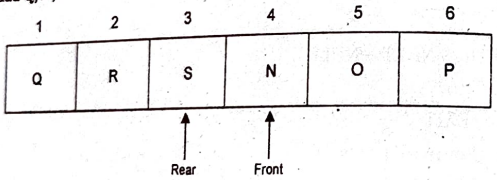
(ii) Add P



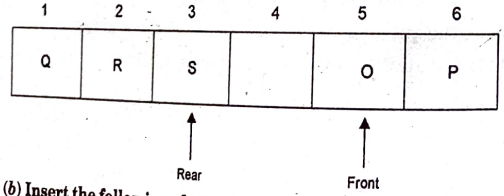
(iii) Delete two elements



(iv) Add Q, R, S



(v) Delete one letter



Q.3. (*b*) **Insert the following elements in alphabetical order into an empty binary search tree:** (7)

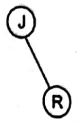
J, R, D, G, T, E, M, H, P, A, F, Q

1. Find the final tree T

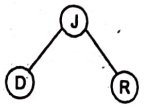
2. Find the post-order traversal of T.

Ans. Insert J

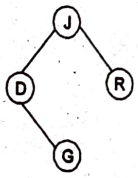
Insert R



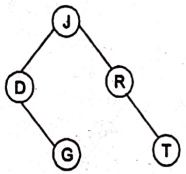
Insert D



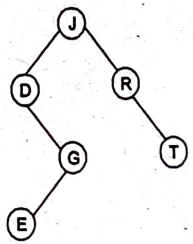
Insert G



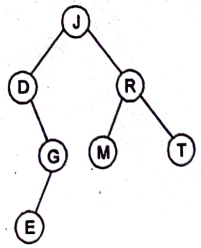
Insert T



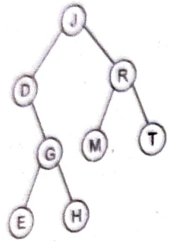
Insert E



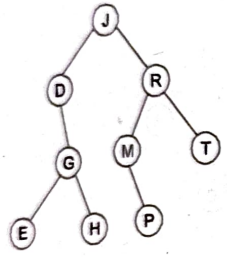
Insert M



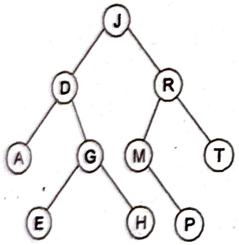
Insert H



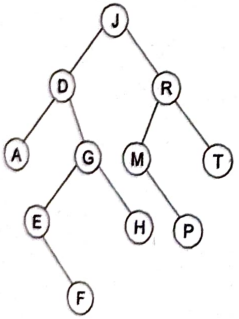
Insert P



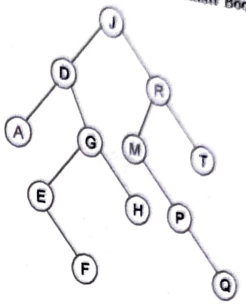
Insert A



Insert F



Insert Q



This is final tree T.

(ii) Post order traversal of tree T is :

AFEHGDQPMTRJ

Q.4. Attempt all parts:

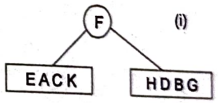
Q.4. (a) A binary tree has 9 nodes. The inorder and preorder traversals are of T yield the following sequences of nodes: **(5)**

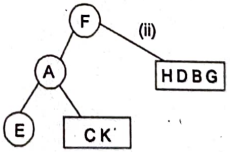
INORDER: E A C K F H D B G

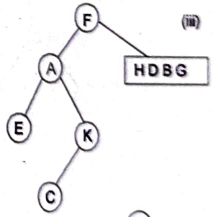
PREORDER: F A E K C D H G B. Draw the tree T.

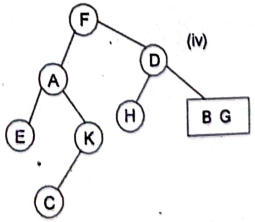
Ans. INORDER: E A C K F H D B G

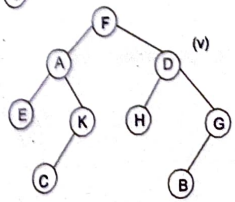
PREORDER: F A E K C D H G B











This is final tree T.

8–2017

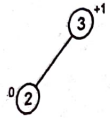
Q.4. (b) Construct the AVL tree if the following elements are inserted in order: (6)

3,2,1,4,5,6,16,15,14

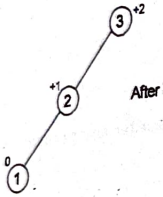
Ans. AVL tree of 3,2,1,4,5,6,16,15,14

Insert 3

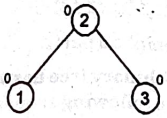
Insert 2



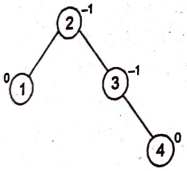
Insert 1



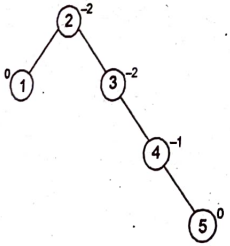
After Rebalancing



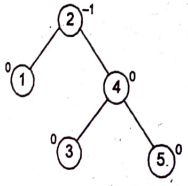
Insert 4



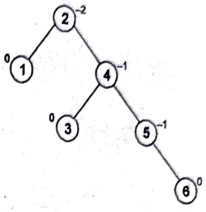
Insert 5



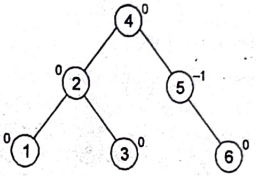
After Rebalancing



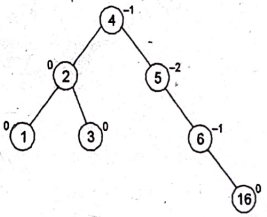
Insert 6



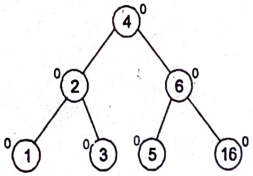
After Rebalancing



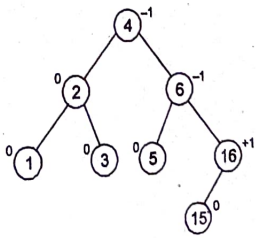
Insert 16



After Rebalancing

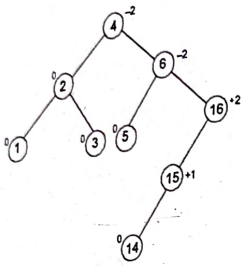


Insert 15

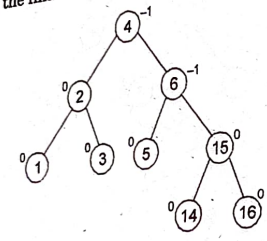


10–2017

Insert 14



After Rebalancing the final tree T is:



END TERM EXAMINATION [DEC. 2017]
THIRD SEMESTER [B.TECH]
DATA STRUCTURES [ETCS–209]

Time : 3 hrs. **M.M. : 75**

Note: *Attempt any five questions including Q.No. 1. Which is compulsory. Select one question from each unit.*

Q.1. (*a*) What is an Abstract Data Type (ADT)? **(2.5)**

Ans. Abstract Data Types

Abstract Data type (ADT) is a type (or class) for objects whose behavior is defined by a set of value and a set of operations.

The definition of ADT only mentions what operations are to be performed but not how these operations will be implemented. It does not specify how data will be organized in memory and what algorithms will be used for implementing the operations. It is called "abstract" because it gives an implementation independent view. The process of providing only the essentials and hiding the details is known as abstraction.

The user of data type need not know that data type is implemented, for example, we have been using int, float, char data types only with the knowledge with values that can take and operations that can be performed on them without any idea of how these types are implemented. So a user only needs to know what a data type can do but not how it will do it.

We can think of ADT as a black box which hides the inner structure and design of the data type. Now we'll define three ADTs namely List ADT, Stack ADT, Queue ADT.

Q.1. (*b*) What is a self-referential structure? **(2.5)**

Ans. A self referential structure is used to create data structures like linked lists, stacks, etc. Following is an example of this kind of structure:

```
struct struct_name
{
datatype datatypename;
struct_name * pointer_name;
};
```

A self-referential structure is one of the data structures which refer to the pointer to (points) to another structure of the same type. For example, a linked list is supposed to be a self-referential data structure. The next node of a node is being pointed, which is of the same struct type. For example,

```
typedef struct listnode {
void *data;
struct listnode *next;
} linked_list;
```

In the above example, the listnode is a self-referential structure – because the *next is of the type struct listnode.

(2) Linear Probing

Final Table

0	1	2	3	4	5	6
397	22	3920	2341	2839	430	4234

Value (430) is collide at 3

so the next probe is combuted as:

$H(430, 1) = [430 \bmod 7 + 1] \bmod 7$

$= [3 + 1] \bmod 7 = 4 \bmod 7 = 4$

But the T [4] is not free so again:

$H(430, 2) = [430 \bmod 7 + 2] \bmod 7 = 5 \bmod 7 = 5$

Now the value (397) is collide at 5.

so the next probe is:

$H(397, 1) = 6$

But six is not free so next probe is:

$H(397, 2) = 0$

Now the value (3920) is collide at 0.

So the next probe is:

$H(3920, 1) = 1$

But T[1] is not free so next probe is:

$H(3920, 2) = 2.$

(3) Quadratic probing

Final Table

0	1	2	3	4	5	6
430	22	3920	2341	2839	397	4234

Now the (430) is collide at 3. So the next probe is computed as:

$H(430, 1) = [430 \% 7 + 1^2] \bmod 7$

$= (3 + 1) \bmod 7 = 4$

But the T[4] is not free so next is:

$H(430, 2) = [430\%7 + 2^2] \bmod 7$

$= (3 + 4) \% 7 = 0$

Now (3920) is collide at 0. So the next probe is:

$H(3920, 1) = 1$ Here T [1] is not free.

$H(3920, 2) = 4$ again T [4] is not free

$H(3920, 3) = 2$ It is free.

FIRST TERM EXAMINATION [SEP. 2018]
THIRD SEMESTER [B.TECH]
DATA STRUCTURES [ETCS–209]

Time : 1.5 hrs. **M.M. : 30**

Note: *Q. No. 1 is compulsory. Attempt any two more Questions from the rest.*

Q.1. Attempt all parts:

(a) Calculate the postfix expression using stack also demonstrate the steps. (2)

5, 3, + 8, 2, –,*

Ans.

Symbol Scanned	Stack	Operation (B op A)
5	5	
3	5,3	
+	8	[5 + 3] (A = 3, B = 5)
8	8,8	
2	8, 8, 2	
–	8, 6	[8 – 2] (A = 2, B = 8)
*	48	[8 * 6] (A = 6, B = 8]

Q. 1. (b) What do you understand by the term space-time tradeoff? (2)

Ans. Time-Space tradeoff: Is a way of solving a problem calculation in less time by using more space (or memory), or by solving a problem in very little space by spending a long time. Most computers have a large amount of space, but not infinite space. Also, most people are willing to wait a little while for a big calculation, but not forever. So if your problem is taking a long time but not much memory, a space-time tradeoff would let you use more memory and solve the problem more quickly or it could be solved very quickly but requires more memory than you have, you can try to spend more time solving the problem in the limited memory.

Q. 1. (c) Differentiate between arrays and linked lists. (2)

Ans.

ARRAY	LINKED LIST
Array is a collection of elements of similar data type.	Linked List is an ordered collection of elements of same type, which are connected to each other using pointers.
Array supports **Random Access.**	Linked List supports **Sequential Access.**
In an array, elements are stored in **contiguous memory location** or consecutive manner in the memory.	In a linked list, new elements can be stored anywhere in the memory.
In array, **Insertion and Deletion** operation takes more time, as the memory locations are consecutive and fixed.	Insertion and Deletion operations are **fast** in linked list.
Memory is allocated as soon as the array is declared, at **compile time.** It's also known as **Static Memory Allocation.**	Memory is allocated at **runtime,** as and when a new node is added. It's also known as **Dynamic Memory Allocation.**

In array, each element is independent and can be accessed using it's index value.	In case of a linked list, each node/element points to the next, previous, or maybe both nodes.
Array can single dimensional, two dimensional or multidimensional	Linked list can be Linear(Singly), Doubly or Circular linked list.

Q. 1. (d) Briefly explain different types of queues. (2)

Ans. Types of queue:

Simple queue: In a simple FIFO queue, the order is retained and data leaves in the same order in which it comes in.

Priority queue: A queue in which the elements are given a predefined priority.

Circular queue: Similar to a simple queue, except that the back of the queue is followed by the front of the queue.

Double ended queue (Dequeue): Similar to the simple queue but can add or remove elements from either the front or the back of the queue.

Q. 1. (e) Define the following terms used in tree: (2)

(i) Terminal Node

(ii) Siblings

(iii) Balance factor

(iv) Height of tree

Ans. (i) **Terminal Node:** In a tree data structure, the node which does not have a child is called as **LEAF Node**. In simple words, a leaf is a node with no child.

(ii) **Siblings:** In a tree data structure, nodes which belong to same Parent are called as **SIBLINGS**. In simple words, the nodes with same parent are called as Sibling nodes.

(iii) **Balance Factor:** We define balance factor for each node as :

balanceFactor = height(left subtree) - height(right subtree)

The balance factor of any node of an AVL tree is in the integer range [-1,+1]. If after any modification in the tree, the balance factor becomes less than –1 or greater than +1, the subtree rooted at this node is unbalanced, and a rotation is needed.

(iv) **Height of tree:** In a tree data structure, the total number of egdes from leaf node to a particular node in the longest path is called as **HEIGHT** of that Node. In a tree, height of the root node is said to be **height of the tree**. In a tree, **height of all leaf nodes is '0'.**

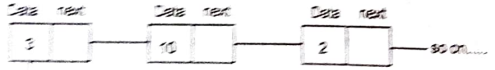
Q. 2. Attempt all parts:

(a) Explain different types of linked lists. (5)

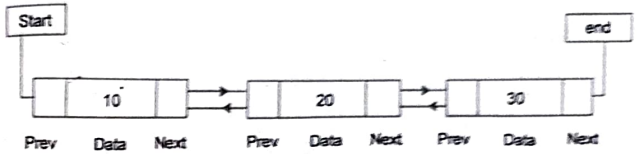
Ans. Types of Linked Lists: There are 3 different implementations of Linked List available, they are:

(i) **Singly Linked List:** Singly linked lists contain nodes which have a **data** part as well as an **address part** i.e. next, which points to the next node in the sequence of nodes.

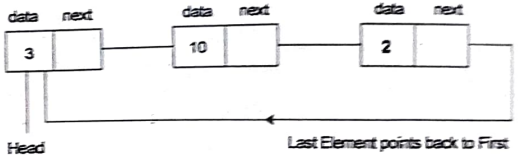
The operations we can perform on singly linked lists are **insertion, deletion** and traversal.



Doubly Linked List: In a doubly linked list, each node contains a **data** part and two addresses, one for the **previous** node and one for the **next** node.



Circular Linked List: In circular linked list the last node of the list holds the address of the first node hence forming a circular chain.



Q. 2. (b) Write an algorithm to delete the second-last node of a doubly linked list. (5)

Ans.

```
ptr=first;
while(ptr->rpt->rpt! = NULL)
{
pre=ptr;
ptr=ptr->rpt;
}
pre->rpt=ptr->rpt;
ptr->rpt->lpt=pre;
free(ptr);
```

Q. 3. Attempt all parts

(a) Write the algorithms for PUSH and POP operations in stack. (4)

Ans. Push Operation: The process of putting a new data element onto stack is known as a Push Operation. Push operation involves a series of steps

- **Step 1** – Checks if the stack is full.
- **Step 2** – If the stack is full, produces an error and exit.
- **Step 3** – If the stack is not full, increments **top** to point next empty space.
- **Step 4** – Adds data element to the stack location, where top is pointing.
- **Step 5** – Returns success.

Pop Operation: Accessing the content while removing it from the stack, is known as a Pop Operation. A Pop operation may involve the following steps–

- **Step 1** – Checks if the stack is empty.
- **Step 2** – If the stack is empty, produces an error and exit.
- **Step 3** – If the stack is not empty, accesses the data element at which **top** is pointing.
- **Step 4** – Decreases the value of top by 1.
- **Step 5** – Returns success.

Q. 3. (b) Insert the following elements in alphabetical order into an empty binary search tree: (5)

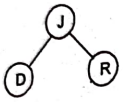
J, R, D, G, B, E, M, H, P, A, F, Z

1. Find the final tree 2. Find the in-order traversal of.

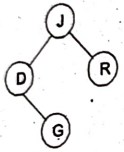
Ans. Insert J

Insert R

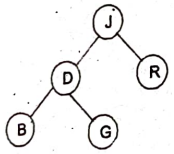
Insert D



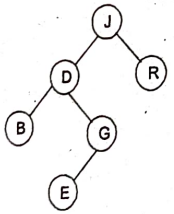
Insert G



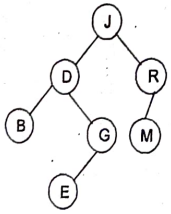
Insert B



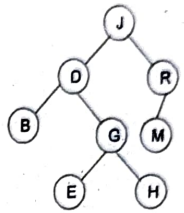
Insert E



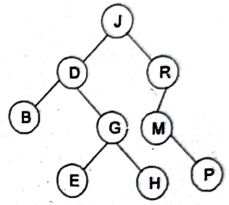
Insert M



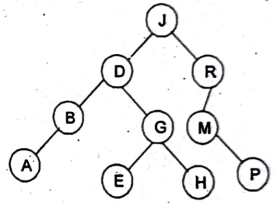
Insert H



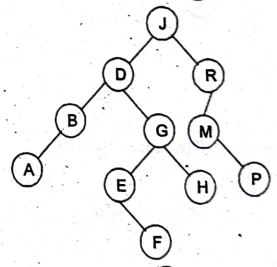
Insert P



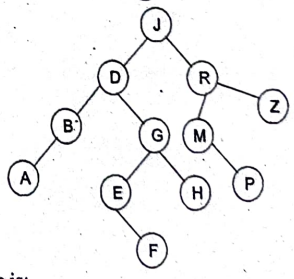
Insert A



Insert F



Insert Z



In order Transversal of above tree is:
A B E F G H D J M P R Z

Q. 4. Attempt all parts:

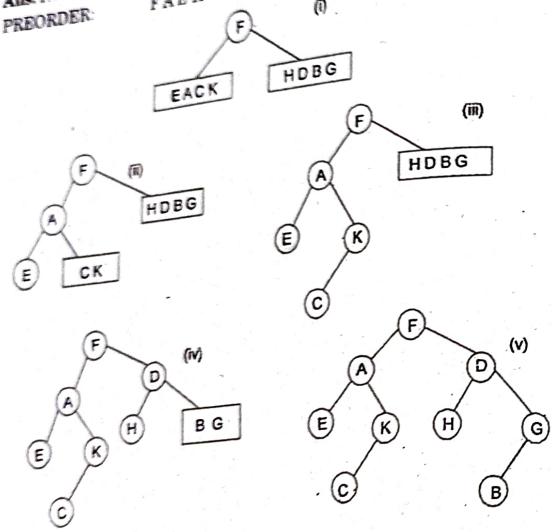
(a) A binary tree has 9 nodes. The inorder and preorder traversals of the tree yield the following sequences of nodes: (5)

INORDER: E A C K F H D B G

PREORDER: F A E K C D H G B. **Draw the tree.**

Ans. INORDER: E A C K F H D B G

PREORDER: F A E K C D H G B



This is final tree T

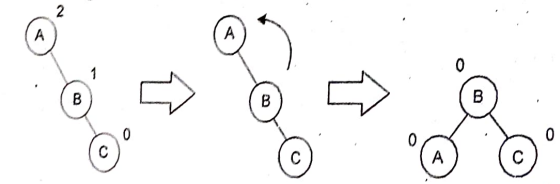
Q. 4. (b) Explain different types of rotations in AVL tree with the help of suitable examples. (5)

Ans. AVL Rotations

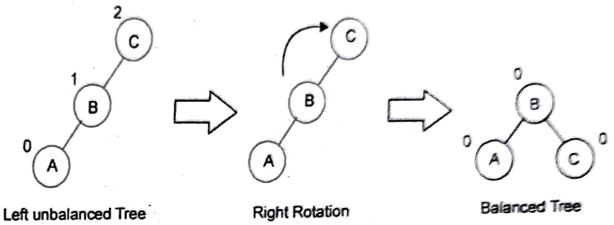
To balance itself, an AVL tree may perform the following four kinds of rotations–

- R-R rotation
- L-L rotation
- L-R rotation
- R-L rotation

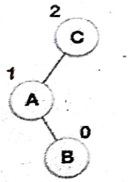
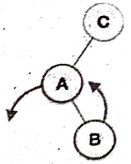
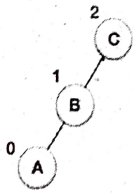
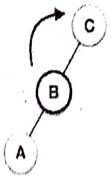
R-R Rotation: If a tree becomes unbalanced, when a node is inserted into the right subtree of the right subtree, then we perform a single left rotation

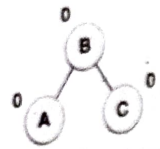


L-L Rotation: AVL tree may become unbalanced, if a node is inserted in the left subtree of the left subtree. The tree then needs a right rotation.



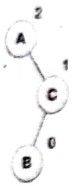
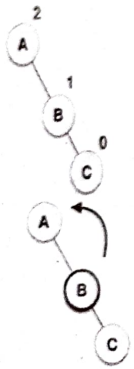
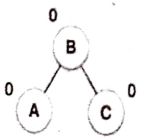
L-R Rotation: Double rotations are slightly complex version of already explained versions of rotations. To understand them better, we should take note of each action performed while rotation. Let's first check how to perform Left-Right rotation. A left-right rotation is a combination of left rotation followed by right rotation.

State	Action
	A node has been inserted into the right subtree of the left subtree. This makes **C** an unbalanced node. These scenarios cause AVL tree to perform left-right rotation.
	We first perform the left rotation on the left subtree of **C**. This makes **A**, the left subtree of **B**.
	Node **C** is still unbalanced, however now, it is because of the left-subtree of the left-subtree.
	We shall now right-rotate the tree, making **B** the new root node of this subtree. **C** now becomes the right subtree of its own left subtree.



The tree is now balanced.

R-L Rotation: The second type of double rotation is Right-Left Rotation. It is a combination of right rotation followed by left rotation.

State	Action
	A node has been inserted into the left subtree of the right subtree. This makes **A**, an unbalanced node with balance factor 2.
	First, we perform the right rotation along **C** node, making **C** the right subtree of its own left subtree **B**. Now, **B** becomes the right subtree of **A**.
	Node **A** is still unbalanced because of the right subtree of its right subtree and requires a left rotation.
	A left rotation is performed by making **B** the new root node of the subtree. **A** becomes the left subtree of its right subtree **B**.
	The tree is now balanced.

END TERM EXAMINATION [NOV-DEC 2018]
THIRD SEMESTER [B.TECH]
DATA STRUCTURES [ETCS–209]

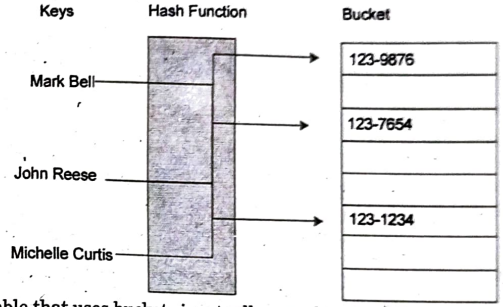
Time : 3 hrs. **M.M. : 75**

Note: Attempt any five questions including Q. No 1 which is compulsory. Select one question from each unit.

Q.1. Attempt all:

(a) What is bucket Hashing? **(2.5)**

Ans. Hash buckets are used to apportion data items for sorting or lookup purposes The aim of this work is to weaken the linked lists so that searching for a specific item can be accessed within a shorter timeframe.



A hash table that uses buckets is actually a combination of an array and a linked list. Each element in the array (the hash table) is a header for a linked list. All elements that hash into the same location will be stored in the list.

The hash function assigns each record to the first slot within one of the buckets. In case the slot is occupied, then the bucket slots will be searched sequentially until an open slot is found.

In case a bucket is completely full, the record will get stored in an overflow bucket of infinite capacity at the end of the table. All buckets share the same overflow bucket. However, a good implementation will use a hash function that distributes the records evenly among the buckets so that as few records as possible go into the overflow bucket.

Q. 1. (b) What is the condition to check overflow and underflow condition in circular queue? **(2.5)**

Ans. If value of REAR variable is greater then or equal to SIZE – 1 and value of FRONT variable is equal to 0 then we can not insert an element into Circular Queue. This condition is known as **"Overflow"**.

If Circular Queue is empty then we can not delete an element from Circular Queue. This condition is known as **"Underflow"**.

Q. 1. (c) What is Header Linked list? **(2.5)**

Ans. A Header linked list is one more variant of linked list. In Header linked list, we have a special node present at the beginning of the linked list. This special node is used to store number of nodes present in the linked list. In other linked list variant, if we want to know the size of the linked list we use traversal method. But in Header linked list, the size of the linked list is stored in its header itself.