Quantum Series

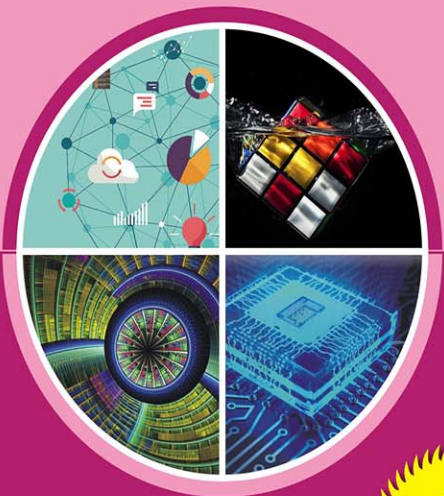
ENGINEERING

QUANTUM Series

Semester - 3

CS & IT

Data Structure



Session 2019-20 Odd Semester

- Topic-wise coverage of entire syllabus in Question-Answer form.
- Short Questions (2 Marks)

Includes solution of following AKTU Question Papers

2014-15 • 2015-16 • 2016-17 • 2017-18• 2018-19

QUANTUM SERIES

For

B.Tech Students of Second Year
of All Engineering Colleges Affiliated to
Dr. A.P.J. Abdul Kalam Technical University,
Uttar Pradesh, Lucknow
(Formerly Uttar Pradesh Technical University)

DATA STRUCTURES

By

Prashant Agarwal

Quantum Page

QUANTUM PAGE PVT. LTD.
Ghaziabad ■ New Delhi

PUBLISHED BY : **Apram Singh**
Quantum Page Pvt. Ltd.
Plot No. 59/2/7, Site - 4, Industrial Area,
Sahibabad, Ghaziabad-201 010

Phone : 0120 - 4160479

Email : pagequantum@gmail.com **Website: www.quantumpage.co.in**

Delhi Office : 1/6590, East Rohtas Nagar, Sahadara, Delhi-110032

Data Structures (CS/IT : Sem-3)

1st Edition : *2009-10*
2nd Edition : *2010-11*
3rd Edition : *2011-12*
4th Edition : *2012-13*
5th Edition : *2013-14*
6th Edition : *2014-15*
7th Edition : *2015-16*
8th Edition : *2016-17*
9th Edition : *2017-18*
10th Edition : *2018-19*
11th Edition : *2019-20* (*Thoroughly Revised Edition*)
12th Edition : *2020-21*

Price: Rs. 100/- only

Printed Version **:** e-Book.

CONTENTS

KCS 301 : DATA STRUCTURE

Quantum Series ENGINEERING

QUANTUM Series

Related titles in Quantum Series

For Semester - 3
(CS & IT)

- Engineering Science Course / Mathematics - IV
- Technical Communication / Universal Human Values
- Data Structure
- Computer Organization & Architecture
- Discrete Structures & Theory of Logic

A comprehensive book to get the big picture without spending hours over lengthy text books.

Quantum Series is the complete one-stop solution for engineering student looking for a simple yet effective guidance system for core engineering subject. Based on the needs of students and catering to the requirements of the syllabi, this series uniquely addresses the way in which concepts are tested through university examinations. The easy to comprehend question answer form adhered to by the books in this series is suitable and recommended for student. The students are able to effortlessly grasp the concepts and ideas discussed in their course books with the help of this series. The solved question papers of previous years act as a additional advantage for students to comprehend the paper pattern, and thus anticipate and prepare for examinations accordingly.

The coherent manner in which the books in this series present new ideas and concepts to students makes this series play an essential role in the preparation for university examinations. The detailed and comprehensive discussions, easy to understand examples, objective questions and ample exercises, all aid the students to understand everything in an all-inclusive manner.

- Topic-wise coverage in Question-Answer form.
- Clears course fundamentals.
- Includes solved University Questions.
- The perfect assistance for scoring good marks.
- Good for brush up before exams.
- Ideal for self-study.

Quantum Page™

Quantum Publications®
(A Unit of Quantum Page Pvt. Ltd.)
Plot No. 59/2/7, Site-4, Industrial Area, Sahibabad,
Ghaziabad, 201010, (U.P.) Phone: 0120-4160479
E-mail: pagequantum@gmail.com Web: www.quantumpage.co.in
Find us on: facebook.com/quantumseriesofficial

1
UNIT

Array and Linked List

CONTENTS

PART-1

Introduction : Basic Terminology, Elementary Data Organization Built in Data Types in C.

Questions-Answers

Long Answer Type and Medium Answer Type Questions

Que 1.1. **Define data structure. Describe about its need and types. Why do we need a data type ?**

AKTU 2014-15, Marks 05

Answer

Data structure :

1. A data structure is a way of organizing all data items that considers not only the elements stored but also their relationship to each other.
2. Data structure is the representation of the logical relationship existing between individual elements of data.
3. Data structure is define as a mathematical or logical model of particular organization of data items.

Data structure is needed because :

1. It helps to understand the relationship of one element with the other.
2. It helps in the organization of all data items within the memory.

The data structures are divided into following categories :

1. **Linear data structure :**
   a. A linear data structure is a data structure whose elements form a sequence, and every element in the structure has a unique predecessor and unique successor.
   b. Examples of linear data structure are arrays, linked lists, stacks and queues.
2. **Non-linear data structure :**
   a. A non-linear data structure it is a data structure whose elements do not form a sequence. There is no unique predecessor or unique successor.
   b. Examples of non-linear data structures are trees and graphs.

Need of data type : The data type is needed because it determines what type of information can be stored in the field and how the data can be formatted.

Que 1.2. **Discuss some basic terminology used and elementary data organization in data structures.**

Answer

Basic terminologies used in data structure :

1. **Data :** Data are simply values or sets of values. A data item refers to a single unit of values.
2. **Entity :** An entity is something that has certain attributes or properties which may be assigned values.
3. **Field :** A field is a single elementary unit of information representing an attribute of an entity.
4. **Record :** A record is the collection of field values of a given entity.
5. **File :** A file is the collection of records of the entities in a given entity set.

Data organization : Each record in a file may contain many field items, but the value in a certain field may uniquely determine the record in the file. Such a field K is called a primary key, and the values k_1, k_2,... in such a field are called keys or key values.

Que 1.3. **Define data types. What are built in data types in C ? Explain.**

Answer

1. C data types are defined as the data storage format that a variable can store a data to perform a specific operation.
2. Data types are used to define a variable before to use in a program.
3. There are two types of built in data types in C :

a. **Primitive data types :** Primitive data types are classified as :

i. **Integer type :** Integers are used to store whole numbers.

Size and range of integer type on 16-bit machine :

Type	Size (bytes)	Range	Format specifier
int or signed int	2	– 32,768 to 32,767	%d
unsigned int	2	0 to 65,535	%u
short int or signed short int	1	– 128 to 127	%hd
unsigned short int	1	0 to 255	%hu
long int or signed long int	4	– 2,147,483,648 to 2,147,483,647	%ld
unsigned long int	4	0 to 4,294,967,295	%lu

ii. **Floating point type :** Floating types are used to store real numbers.

Size and range of floating point type on 16-bit machine :

Type	Size (bytes)	Range	Format specifier
Float	4	3.4E – 38 to 3.4E+38	%f
double	8	1.7E – 308 to 1.7E+308	%lf
long double	10	3.4E – 4932 to 1.1E+4932	%lf

iii. **Character type :** Character types are used to store characters value.

Size and range of character type on 16-bit machine :

Type	Size (bytes)	Range	Format specifier
char or signed char	1	– 128 to 127	%c
unsigned char	1	0 to 255	%c

iv. **Void type :** Void type is usually used to specify the type of functions which returns nothing.

b. **Non-primitive data types :**
   i. These are more sophisticated data types. These are derived from the primitive data types.
   ii. The non-primitive data types emphasize on structuring of a group of homogeneous (same type) or heterogeneous (different type) items. For example, arrays, lists and files.

PART-2

Algorithm, Efficiency of an Algorithm, Time and Space Complexity.

Questions-Answers

Long Answer Type and Medium Answer Type Questions

Que 1.4. **Define algorithm. Explain the criteria an algorithm must satisfy. Also, give its characteristics.**

Answer

1. An algorithm is a step-by-step finite sequence of instructions, to solve a well-defined computational problem.
2. Every algorithm must satisfy the following criteria :
   i. **Input :** There are zero or more quantities which are externally supplied.

ii. **Output :** At least one quantity is produced.

iii. **Definiteness :** Each instruction must be clear and unambiguous.

iv. **Finiteness :** If we trace out the instructions of an algorithm, then for all cases the algorithm will terminate after a finite number of steps.

v. **Effectiveness :** Every instruction must be basic and essential.

Characteristics of an algorithm :

1. It should be free from ambiguity.
2. It should be concise.
3. It should be efficient.

Que 1.5. **How the efficiency of an algorithm can be checked ? Explain the different ways of analyzing algorithm.**

Answer

The efficiency of an algorithm can be checked by :

1. Correctness of an algorithm
2. Implementation of an algorithm
3. Simplicity of an algorithm
4. Execution time and memory requirements of an algorithm

Different ways of analyzing an algorithm :

a. **Worst case running time :**
   1. The behaviour of an algorithm with respect to the worst possible case of the input instance.
   2. The worst case running time of an algorithm is an upper bound on the running time for any input.

b. **Average case running time :**
   1. The expected behaviour when the input is randomly drawn from a given distribution.
   2. The average case running time of an algorithm is an estimate of the running time for an "average" input.

c. **Best case running time :**
   1. The behaviour of the algorithm when input is already in order. For example, in sorting, if elements are already sorted for a specific algorithm.
   2. The best case running time rarely occurs in practice comparatively with the first and second case.

Que 1.6. **Define complexity and its types.**

Answer

1. The complexity of an algorithm M is the function $f(n)$ which gives the running time and/or storage space requirement of the algorithm in terms of the size n of the input data.

2. The storage space required by an algorithm is simply a multiple of the data size n.
3. Following are various cases in complexity theory :
   a. **Worst case :** The maximum value of $f(n)$ for any possible input.
   b. **Average case :** The expected value of $f(n)$ for any possible input.
   c. **Best case :** The minimum possible value of $f(n)$ for any possible input.

Types of complexity :
1. **Space complexity :** The space complexity of an algorithm is the amount of memory it needs to run to completion.
2. **Time complexity :** The time complexity of an algorithm is the amount of time it needs to run to completion.

Que 1.7. **What do you understand by complexity of an algorithm ? Compute the worst case complexity for the following C code :**

```
main()
{
int s = 0, i, j, n;
for (j = 0;  j < (3 * n); j++)
{
for (i = 0; i < n; i++)
{
s = s + i;
}
printf("%d", i);
}}
```

AKTU 2014-15, Marks 05

Answer

Complexity of an algorithm : Refer Q. 1.6, Page 1–6A, Unit-1.
Worst case complexity : $\Omega(n) + \Omega(3n) = \Omega(n)$

Que 1.8. **How do you find the complexity of an algorithm ? What is the relation between the time and space complexities of an algorithm ? Justify your answer with an example.**

AKTU 2015-16, Marks 10

Answer

Complexity of an algorithm : Refer Q. 1.6, Page 1–6A, Unit-1.
Relation between the time and space complexities of an algorithm :
1. The time and space complexities are not related to each other.
2. They are used to describe how much space/time our algorithm takes based on the input.
3. For example, when the algorithm has space complexity of :

  a. O(1) *i.e.*, constant then the algorithm uses a fixed (small) amount of space which does not depend on the input. For every size of the input the algorithm will take the same (constant) amount of space.
  b. O(n), O(n^2), O(log (n)) - these indicate that we create additional objects based on the length of our input.
4. In contrast, the time complexity describes how much time our algorithm consumes based on the length of the input.
5. For example, when the algorithm has time complexity of :
  a. O(1) *i.e.*, constant then no matter how big is the input it always takes a constant time.
  b. O(n), O(n^2), O(log (n)) - again it is based on the length of the input.

For example :

```
function(list l) {
for (node in l) {
print(node) ;
}
}
```

```
function(list l) {
print("I got a list"); }
```

In this example, both take O(1) space as we do not create additional objects which shows that time and space complexity might be different.

PART-3

Asymptotic Notations : Big Oh, Big Theta, and Big Omega.

Questions-Answers

Long Answer Type and Medium Answer Type Questions

Que 1.9. **What is asymptotic notation ? Explain the big 'Oh' notation.**

Answer

1. Asymptotic notation is a shorthand way to describe running times for an algorithm.
2. It is a line that stays within bounds.
3. These are also referred to as 'best case' and 'worst case' scenarios respectively.

Big 'Oh' notation :

1. Big-Oh is formal method of expressing the upper bound of an algorithm's running time.
2. It is the measure of the longest amount of time it could possibly take for the algorithm to complete.
3. More formally, for non-negative functions, $f(n)$ and $g(n)$, if there exists an integer n_0 and a constant $c > 0$ such that for all integers $n > n_0$.

$$f(n) \leq cg(n)$$

4. Then, $f(n)$ is Big-Oh of $g(n)$. This is denoted as : $f(n) \in O(g(n))$ *i.e.*, the set of functions which, as n gets large, grow faster than a constant time $f(n)$.

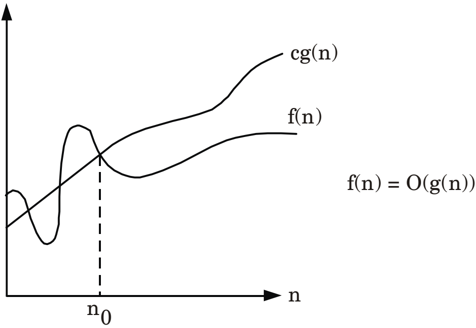


Fig. 1.9.1.

Que 1.10. **What is complexity of an algorithm ? Explain various notations used to express the complexity of an algorithm.**

OR

What are the various asymptotic notations ? Explain Big O notation.

AKTU 2017-18, Marks 07

Answer

Complexity of an algorithm : Refer Q. 1.6, Page 1–6A, Unit-1.

Notations used to express the complexity of an algorithm :

1. **θ-Notation (Same order) :**
   a. This notation bounds a function to within constant factors.
   b. We say $f(n) = \theta g(n)$ if there exist positive constants n_0, c_1 and c_2 such that to the right of n_0 the value of $f(n)$ always lies between $c_1 g(n)$ and $c_2 g(n)$ inclusive.

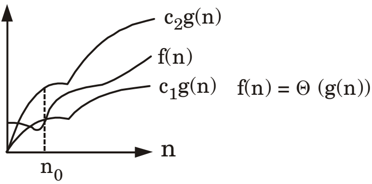


Fig. 1.10.1.

2. **Oh-Notation (Upper bound) :** Refer Q. 1.9, Page 1–8A, Unit-1.
3. **Ω-Notation (Lower bound) :**
   a. This notation gives a lower bound for a function to within a constant factor.

b. We write $f(n) = \Omega\, g(n))$ if there are positive constants n_0 and c such that to the right of n_0, the value of $f(n)$ always lies on or above $cg(n)$.

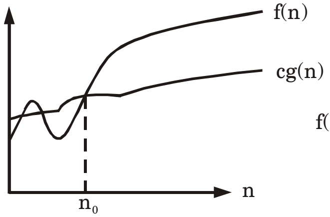


Fig. 1.10.2.

4. **Little - Oh notation (o) :**
   a. It is used to denote an upper bound that is asymptotically tight because upper bound provided by O-notation is not tight.
   b. We write $o(g(n)) = \{f(n)$: For any positive constant $c > 0$, if a constant $n_0 > 0$ such that $0 \leq f(n) < cg(n)\ \forall\ n \geq n_0\}$
5. **Little omega notation (ω) :**
   a. It is used to denote lower bound that is asymptotically tight.
   b. We write $\omega(g(n)) = \{f(n)$: For any positive constant $c > 0$, if a constant $n_0 > 0$ such that $0 \leq cg(n) < f(n)\}$

PART-4

Time-Space Trade-off, Abstract Data Types (ADT).

Questions-Answers

Long Answer Type and Medium Answer Type Questions

Que 1.11. Explain time-space trade-off in brief with suitable example.

OR

What do you understand by time and space trade-off ? Define the various asymptotic notations. Derive the O-notation for linear search.

AKTU 2018-19, Marks 07

Answer

Time-space trade-off :

1. The time-space trade-off refers to a choice between algorithmic solutions of data processing problems that allows to decrease the running time of an algorithmic solution by increasing the space to store data and vice-versa.

2. Time-space trade-off is basically a situation where either space efficiency (memory utilization) can be achieved at the cost of time or time efficiency (performance efficiency) can be achieved at the cost of memory.

   For Example : Suppose, in a file, if data stored is not compressed, it takes more space but access takes less time. Now if the data stored is compressed the access takes more time because it takes time to run decompression algorithm.

Various asymptotic notation : Refer Q. 1.10, Page 1–9A, Unit-1.

Derivation :

Best case : In the best case, the desired element is present in the first position of the array, *i.e.*, only one comparison is made.

So, $T(n) = O(1).$

Average case : Here we assume that ITEM does appear, and that is equally likely to occur at any position in the array. Accordingly the number of comparisons can be any of the number 1, 2, 3, ..., n and each number occurs with the probability $p = 1/n$. Then

$$\begin{aligned} T(n) &= 1 \,.\, (1/n) + 2 \,.\, (1/n) + 3 \,.\, (1/n) \,......\, + n \,.\, (1/n) \\ &= (1 + 2 + 3 + \,......\, + n) \,.\, (1/n) \\ &= n \,.\, (n + 1)/2 \,.\, (1/n) \\ &= (n + 1)/2 \\ &= O((n + 1)/2) \simeq O(n) \end{aligned}$$

Worst case : Worst case occurs when ITEM is the last element in the array or is not there at all. In this situation n comparison is made

So, $T(n) = O(n + 1) \simeq O(n)$

Que 1.12. What do you understand by time-space trade-off ? Explain best, worst and average case analysis in this respect with an example.

AKTU 2017-18, Marks 07

Answer

Time-space trade-off : Refer Q. 1.11, Page 1–10A, Unit-1.

Best, worst and average case analysis : Suppose we are implementing an algorithm that helps us to search for a record amongst a list of records. We can have the following three cases which relate to the relative success our algorithm can achieve with respect to time :

1. **Best case :**
   a. The record we are trying to search is the first record of the list.
   b. If $f(n)$ is the function which gives the running time and / or storage space requirement of the algorithm in terms of the size n of the input data, this particular case of the algorithm will produce a

complexity $C(n) = 1$ for our algorithm $f(n)$ as the algorithm will run only 1 time until it finds the desired record.

2. **Worst case :**
   a. The record we are trying to search is the last record of the list.
   b. If $f(n)$ is the function which gives the running time and / or storage space requirement of the algorithm in terms of the size n of the input data, this particular case of the algorithm will produce a complexity $C(n) = n$ for our algorithm $f(n)$, as the algorithm will run n times until it finds the desired record.
3. **Average case :**
   a. The record we are trying to search can be any record in the list.
   b. In this case, we do not know at which position it might be.
   c. Hence, we take an average of all the possible times our algorithm may run.
   d. Hence assuming for n data, we have a probability of finding any one of them is $1/n$.
   e. Multiplying each of these with the number of times our algorithm might run for finding each of them and then taking a sum of all those multiples, we can obtain the complexity $C(n)$ for our algorithm $f(n)$ in case of an average case as following :

$$C(n) = 1\cdot\frac{1}{2} + 2\cdot\frac{1}{2} + \ldots + n\cdot\frac{1}{2}$$

$$C(n) = (1 + 2 + \ldots + n)\cdot\frac{1}{2}$$

$$C(n) = \frac{n(n+1)}{2}\cdot\frac{1}{n} = \frac{n+1}{2}$$

Hence in this way, we can find the complexity of an algorithm for average case as

$$C(n) = O((n + 1)/2)$$

Que 1.13. **What do you mean by Abstract Data Type ?**

Answer

1. An Abstract Data Type (ADT) is defined as a mathematical model of the data objects that make up a data type as well as the functions that operate on these objects.
2. An Abstract Data Type (ADT) is the specification of the data type which specifies the logical and mathematical model of the data type.
3. It does not specify how data will be organized in memory and what algorithm will be used for implementing the operations.

4. An implementation chooses a data structure to represent the ADT.
5. The important step is the definition of ADT that involves mainly two parts :
   a. Description of the way in which components are related to each other.
   b. Statements of operations that can be performed on the data type.

PART-5

Array : Definition, Single and Multidimensional Array.

Questions-Answers

Long Answer Type and Medium Answer Type Questions

Que 1.14. **Define array. How arrays can be declared ?**

Answer

1. An array can be defined as the collection of the sequential memory locations, which can be referred to by a single name along with a number known as the index, to access a particular field or data.
2. The general form of declaration is :

   type variable-name [size];

   a. Type specifies the type of the elements that will be contained in the array, such as int, float or char and the size indicates the maximum of elements that can be stored inside the array.
   b. For example, when we want to store 10 integer values, then we can use the following declaration, int A[10].

Que 1.15. **Write short note on types of an array.**

Answer

There are two types of array :

1. One-dimensional array :

   a. An array that can be represented by only one-one dimension such as row or column and that holds finite number of same type of data items is called one-dimensional (linear) array.
   b. One dimensional array (or linear array) is a set of '*n*' finite numbers of homogeneous data elements such as :
      i. The elements of the array are referenced respectively by an index set consisting of '*n*' consecutive number.

ii. The elements of the array are stored respectively in successive memory locations.
'n' number of elements is called the length or size of an array. The elements of an array 'A' may be denoted in C language as :
$A[0], A[1], A[2], \ldots A[n-1]$

2. Multidimensional arrays :

a. An array can be of more than one dimension. There are no restrictions to the number of dimensions that we can have.
b. As the the dimensions increase the memory requirements increase drastically which can result in shortage of memory.
c. Hence a multidimensional array must be used with utmost care.
d. For example, the following declaration is used for 3-D array :

```
int   a [50] [50] [50];
```

PART-6

Representation of Arrays : Row Major Order, and Column Major Order, Derivation of Index Formulae for 1–D, 2–D, 3–D and n–D Array.

Questions-Answers

Long Answer Type and Medium Answer Type Questions

Que 1.16. What is row major order ? Explain with an example.

AKTU 2014-15, Marks 05

Answer

1. In row major order, the element of an array is stored in computer memory as row-by-row.
2. Under row major representation, the first row of the array occupies the first set of memory locations reserved for the array, the second row occupies the next set, and so forth.
3. In row major order, elements of a two-dimensional array are ordered as :
$A_{11}, A_{12}, A_{13}, A_{14}, A_{15}, A_{16}, A_{21}, A_{22}, A_{23}, A_{24}, A_{25}, A_{26}, A_{31}, \ldots, A_{46}, A_{51}, A_{52}, \ldots, A_{56}$

Example :

Let us consider the following two-dimensional array :

$$\begin{bmatrix} a & b & c & d \\ e & f & g & h \\ i & j & k & l \end{bmatrix}$$

a. Move the elements of the second row starting from the first element to the memory location adjacent to the last element of the first row.

b. When this step is applied to all the rows except for the first row, we have a single row of elements. This is the row major representation.

c. By application of above mentioned process, we get $\{a, b, c, d, e, f, g, h, i, j, k, l\}$

Que 1.17. Explain column major order with an example.

Answer

1. In column major order the elements of an array is stored as column-by-column, it is called column major order.
2. Under column major representation, the first column of the array occupies the first set of memory locations reserved for the array, the second column occupies the next set, and so forth.
3. In column major order, elements are ordered as :

   $A_{11}, A_{21}, A_{31}, A_{41}, A_{51}, A_{12}, A_{22}, A_{32}, A_{42}, A_{52}, A_{13}, \ldots\ldots, A_{55}, A_{16}, A_{26}, \ldots\ldots\ldots, A_{56}.$

   Example : Consider the following two-dimensional array :

$$\begin{bmatrix} a & b & c & d \\ e & f & g & h \\ i & j & k & l \end{bmatrix}$$

a. Transpose the elements of the array. Then, the representation will be same as that of the row major representation.

b. Then perform the process of row-major representation.

c. By application of above mentioned process, we get $\{a, e, i, b, f, j, c, g, k, d, h, l\}$.

Que 1.18. Write a short note on address calculation for 2D array.

OR

Determine addressing formula to find the location of $(i, j)^{th}$ element of a $m \times n$ matrix stored in column major order.

OR

Derive the index formulae for 1-D and 2-D array.

Answer

1. Let us consider a two-dimensional array A of size $m \times n$. Like linear array system keeps track of the address of first element only, *i.e.*, base address of the array (Base (A)).
2. Using the base address, the computer computes the address of the element in the i^{th} row and j^{th} column *i.e.*, LOC ($A[i][j]$).

Formulae :

a. Column major order :

LOC($A[i][j]$) = Base (A) + $w[m(j$ – lower bound for column index)

+ (i – lower bound for row index)]

LOC($A[i][j]$) = Base (A) + $w[mj + i]$ in C/C++

b. Row major order :

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

LOC($A[i][j]$) = Base (A) + $w[ni + j]$ in C/C++

where w denotes the number of words per memory location for the array A or the number of bytes per storage location for one element of the array.

Que 1.19. Explain the formulae for address calculation for 3-D array with example.

Answer

In three-dimensional array, address is calculated using following two methods :

Row major order :

Location ($A[i, j, k]$) = Base (A) + mn (k – 1) + n (i – 1) + (j – 1)

Column major order :

Location ($A[i, j, k]$) = Base (A) + mn (k – 1) + m (j – 1) + (i – 1)

For example : Given an array [1..8, 1..5, 1..7] of integers. If Base (A) = 900 then address of element A[5, 3, 6], by using rows and columns methods are :

The dimensions of A are : $M = 8, N = 5, R = 7, i = 5, j = 3, k = 6$

Row major order :

$$\begin{aligned} \text{Location } (A[i, j, k]) &= \text{Base } (A) + mn\,(k-1) + n\,(i-1) + (j-1) \\ \text{Location } (A[5, 3, 6]) &= 900 + 8 \times 5(6-1) + 5(5-1) + (3-1) \\ &= 900 + 40 \times 5 + 5 \times 4 + 2 \\ &= 900 + 200 + 20 + 2 = 1122 \end{aligned}$$

Column major order :

$$\begin{aligned} \text{Location } (A[i, j, k]) &= \text{Base } (A) + mn\,(k-1) + m\,(j-1) + (i-1) \\ \text{Location } (A[5, 3, 6]) &= 900 + 8 \times 5(6-1) + 8(3-1) + (5-1) \\ &= 900 + 40 \times 5 + 8 \times 2 + 4 \\ &= 900 + 200 + 16 + 4 = 1120 \end{aligned}$$

Que 1.20. Consider the linear arrays *AAA* [5 : 50], *BBB* [– 5 : 10] and *CCC* [1 : 8].

a. Find the number of elements in each array.

b. Suppose base (*AAA*) = 300 and w = 4 words per memory cell for *AAA*. Find the address of *AAA* [15], *AAA* [35] and *AAA* [55].

AKTU 2015-16, Marks 10

Answer

a. The number of elements is equal to the length; hence use the formula :

$$\text{Length} = \text{UB} - \text{LB} + 1$$
$$\text{Length}\ (AAA) = 50 - 5 + 1 = 46$$
$$\text{Length}\ (BBB) = 10 - (-5) + 1 = 16$$
$$\text{Length}\ (CCC) = 8 - 1 + 1 = 8$$

b. Use the formula

$$\text{LOC}\ (AAA\ [i]) = \text{Base}\ (AAA) + w\ (i - \text{LB})$$
$$\text{LOC}\ (AAA\ [15]) = 300 + 4\ (15 - 5) = 340$$
$$\text{LOC}\ (AAA\ [35]) = 300 + 4\ (35 - 5) = 420$$

AAA [55] is not an element of AAA, since 55 exceeds UB = 50.

Que 1.21. Suppose multidimensional arrays P and Q are declared as $P(-2: 2, 2: 22)$ and $Q(1: 8, -5: 5, -10 : 5)$ stored in column major order

i. Find the length of each dimension of P and Q.

ii. The number of elements in P and Q.

iii. Assuming base address (Q) = 400, W = 4, find the effective indices E_1, E_2, E_3 and address of the element Q[3, 3, 3].

AKTU 2018-19, Marks 07

Answer

i. The length of a dimension is obtained by

Length = Upper Bound – Lower Bound + 1

Hence, the lengths of the dimension of P are,

$L_1 = 2 - (-2) + 1 = 5;\quad L_2 = 22 - 2 + 1 = 21$

The lengths of the dimension of Q are,

$L_1 = 8 - 1 + 1 = 8;\quad L_2 = 5 - (-5) + 1 = 11;\quad L_3 = 5 - (-10) + 1 = 16$

ii. Number of elements in $P = 21 \times 5 = 105$ elements

Number of elements in $Q = 8 \times 11 \times 16 = 1408$ elements

iii. The effective index E_i is obtained from $E_i = k_i - \text{LB}$, where k_i is the given index and LB, is the Lower Bound. Hence,

$E_1 = 3 - 1 = 2;\quad E_2 = 3 - (-5) = 8;\quad E_3 = 3 - (-10) = 13$

The address depends on whether the programming language stores Q in row major order or column major order. Assuming Q is stored in column major order.

$E_3L_2 = 13 \times 11 = 143$

$E_3L_2 + E_2 = 143 + 8 = 151$

$(E_3L_2)L_1 = 151 * 8 = 1208$

$(E_2L_2+E_2)L_1 + E_1 = 1208 + 2 = 1210$

Therefore, LOC(Q[3,3,3]) = 400 + 4(1210) = 400 + 4840 = 5240

PART-7

Application of Arrays, Sparse Matrices and their Representation.

Questions-Answers

Long Answer Type and Medium Answer Type Questions

Que 1.22. Write a short note on application of arrays.

Answer

1. Arrays are used to implement mathematical vectors and matrices, as well as other kinds of rectangular tables. Many databases, small and large, consist of one-dimensional arrays whose elements are records.
2. Arrays are used to implement other data structures, such as lists, heaps, hash tables, queues and stacks.
3. Arrays are used to emulate in-program dynamic memory allocation, particularly memory pool allocation.
4. Arrays can be used to determine partial or complete control flow in programs, as a compact alternative to multiple "if" statements.
5. The array may contain subroutine pointers (or relative subroutine numbers that can be acted upon by SWITCH statements) that direct the path of the execution.

Que 1.23. What are sparse matrices ? Explain.

Answer

1. Sparse matrices are the matrices in which most of the elements of the matrix have zero value.
2. Two general types of n-square sparse matrices, which occur in various applications, as shown in Fig. 1.23.1.
3. It is sometimes customary to omit block of zeros in a matrix as in Fig. 1.23.1. The first matrix, where all entries above the main diagonal are zero or, equivalently, where non-zero entries can only occur on or below the main diagonal, is called a lower triangular matrix.
4. The second matrix, where non-zero entries can only occur on the diagonal or on elements immediately above or below the diagonal, is called tridiagonal matrix.

$$\begin{pmatrix} 4 & & & \\ 3 & -5 & & \\ 1 & 0 & 6 & \\ -7 & 8 & -1 & 3 \end{pmatrix}$$

(*a*) Triangular matrix

$$\begin{pmatrix} 5 & -3 & & & \\ 1 & 4 & 3 & & \\ & 9 & -3 & 6 & \\ & & 2 & 4 & -7 \end{pmatrix}$$

(*b*) Tridiagonal matrix

Fig. 1.23.1.

Que 1.24. Write a short note on representation of sparse matrices.

Answer

There are two ways of representing sparse matrices :

1. **Array representation :**
   i. In the array representation of a sparse matrix, only the non-zero elements are stored so that storage space can be reduced.
   ii. Each non-zero element in the sparse matrix is represented as (row, column, value).
   iii. For this a two-dimensional array containing three columns can be used. The first column is for storing the row numbers, the second column is for storing the column numbers and the third column represents the value corresponding to the non-zero element at (row, column) in the first two columns.
   iv. For example, consider the following sparse matrix :

$$\begin{bmatrix} 2 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 4 & 3 & 0 \end{bmatrix}$$

The above matrix can be represented as :

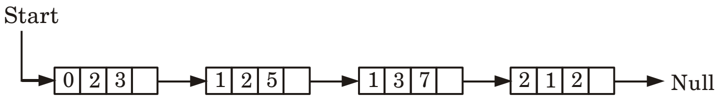
Row	Column	Value
0	0	2
1	1	1
2	1	4
2	2	3

2. **Linked representation :**
   i. In the linked list representation each node has four fields. These four fields are defined as :
      a. **Row :** Index of row, where non-zero element is located.
      b. **Column :** Index of column, where non-zero element is located.
      c. **Value :** Value of non-zero element located at index – (row, column).
      d. **Next node :** Address of next node.

Node structure :

Row	Column	Value	Address

Example : $\begin{bmatrix} 0 & 0 & 3 & 0 \\ 0 & 0 & 5 & 7 \\ 0 & 2 & 0 & 0 \end{bmatrix}$



Que 1.25. **Explain the upper triangular and lower triangular sparse matrices. Suggest a space efficient representation for sparse matrices.**

Answer

1. The matrix, where all entries above the main diagonal are zero or equivalently, where non-zero entries can only occur, on or below the main diagonal, is called lower triangular matrix.
2. A matrix in which all the entries below the main diagonal are zero is called upper triangular matrix.

Space efficient representation for sparse matrices : Refer Q. 1.24, Page 1–19A, Unit-1.

PART-8

Linked List : Array Implementation and Pointer Implementation of Singly Linked List.

Questions-Answers

Long Answer Type and Medium Answer Type Questions

Que 1.26. **Define the term linked list. Write a C program to implement singly linked list for the following function using array :**

i. Insert at beginning
ii. Insert at end
iii. Insert after element
iv. Delete at end
v. Delete at beginning
vi. Delete after element
vii. Display in reverse order

Answer

i. Linked list :

1. A linked list, or one-way list, is a linear collection of data elements, called nodes, where the linear order is given by means of pointers.

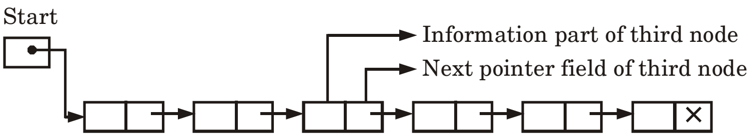


Fig. 1.26.1.

2. Each node is divided into two parts: the first part contains the information of the element, and the second part, called the link field or next pointer field, contains the address of the next node in the list.

Program :

```
#include<stdio.h>
#include<conio.h>
#include<alloc.h>
struct node {
    int info;
    struct node *link;
} ;
struct node *first;
void main( )
```

i. Insert at beginning :

```
void insert_beginning( ) {
struct node *ptr;
ptr = (struct node*)malloc(sizeof(struct node));
if (ptr == NULL) {
printf ("overflow\n" ) ;
return;
}
printf ("input new node information");
scanf ("%d", &ptr -> info) ;
ptr -> link = first;
first = ptr;
}
```

ii. Insert at end :

```
void insert_end( ) {
struct node *ptr; *cpt;
ptr = (struct node*)malloc(sizeof(struct node));
if (ptr == NULL) {
printf ("Link list is overflow\n");
return;
}
printf ("input new node information");
scanf ("%d", &ptr -> info);
cpt = first;
while (cpt -> link != NULL)
cpt = cpt -> link;
```

```
cpt -> link = ptr;
ptr -> link = NULL;
```

iii. Insert after element :

```
void insert_given_node( ) {
struct node *ptr, *cpt;
int data;
ptr = (struct node*)malloc(sizeof(struct node));
if (ptr == NULL) {
printf ("overflow\n");
return;
}
printf ("input new node information");
scanf ("%d", &ptr -> info);
printf ("input information of node after which insertion will be made") ;
scanf ("%d", &data) ;
cpt = first;
while (cpt -> info != data)
cpt = cpt -> link;
ptr -> link = cpt -> link;
cpt -> link = ptr;
}
```

iv. Delete at end :

```
void delete_end( ) {
struct node *ptr, *cpt;
if (first == NULL) {
printf ("underflow\n");
return;
}
ptr = first;
while (ptr -> link != NULL) {
cpt = ptr;
ptr = ptr -> link;
}
cpt -> link = NULL;
free (ptr);
}
```

v. Delete at beginning :

```
void delete_beginning( ) {
struct node *ptr;
if (first == NULL) {
printf ("underflow\n") ;
return;
}
ptr = first;
first = ptr -> link;
free (ptr) ;
}
```

vi. Delete after element :

```
void delete_given_info( ) {
struct node *ptr, *cpt;
int data;
if (first == NULL) {
printf ("underflow\n") ;
return;
}
ptr = first;
printf ("input information of node to be deleted") ;
scanf ("%d", & data);
while (ptr -> info != data) {
cpt = ptr;
ptr = ptr -> link;
}
cpt -> link = ptr -> link;
free (ptr);
}
```

vii. Display in reverse order :

```
reverse_list( ) {
        ptr = First;
        cpt = NULL;
        while (ptr != NULL) {
            cpt = ptr -> link;
            ptr -> link = tpt;
            cpt = ptr;
            ptr = cpt;
        } }
```

Que 1.27. **Write algorithm of following operation for linear linked list :**

i. Traversal
ii. Insertion at beginning
iii. Search an element
iv. Delete node at specified location
v. Deletion at end

Answer

i. **Traversing a linked list :** Let LIST be a linked list in memory. This algorithm traverses LIST, applying an operation PROCESS to each element of LIST. The variable PTR points to the node currently being processed.

1. Set PTR := START [Initializes pointer PTR]
2. Repeat Steps 3 and 4 while PTR != NULL
3. Apply PROCESS to PTR -> INFO
4. Set PTR := PTR -> LINK [PTR now points to the next node]
   [End of Step 2 loop]
5. Exit

ii. Insertion at beginning : Here START is a pointer variable which contains the address of first node. ITEM is the value to be inserted.

1. If (START == NULL) Then
2. START = New Node [Create a new node]
3. START->INFO = ITEM [Assign ITEM to INFO field]
4. START->LINK = NULL [Assign NULL to LINK field]
   Else
5. Set PTR = START [Initialize PTR with START]
6. START = New Node [Create a new node]
7. START->INFO = ITEM [Assign ITEM to INFO field]
8. START->LINK = PTR [Assign PTR to LINK field]
   [End of If]
9. Exit

iii. Search an element : Here START is a pointer variable which contains the address of first node. ITEM is the value to be searched.

1. Set PTR = START, LOC = 1 [Initialize PTR and LOC]
2. Repeat While (PTR != NULL)
3. If (ITEM == PTR -> INFO) Then [Check if ITEM matches with INFO field]
4. Print: ITEM is present at location LOC
5. Return
6. Else
7. PTR = PTR -> LINK [Move PTR to next node]
8. LOC = LOC + 1 [Increment LOC]
9. [End of If]
10. [End of While Loop]
11. Print: ITEM is not present in the list
12. Exit

iv. Delete node at specified position : Here START is a pointer variable which contains the address of first node. PTR is a pointer variable which contains address of node to be deleted. PREV is a pointer variable which points to previous node. ITEM is the value to be deleted.

1. If (START == NULL) Then [Check whether list is empty]
2. Print: Linked-List is empty.
3. Else If (START -> INFO == ITEM) Then
   [Check if ITEM is in 1st node]
4. PTR = START
5. START = START -> LINK [START now points to 2nd node]
6. Delete PTR
7. Else
8. PTR = START, PREV = START
9. Repeat While (PTR != NULL)
10. If (PTR -> INFO == ITEM) Then
    [If ITEM matches with PTR->INFO]
11. PREV -> LINK = PTR -> LINK [Assign LINK field of PTR to PREV]
12. Delete PTR
13. Else

14. PREV = PTR [Assign PTR to PREV]
15. PTR = PTR -> LINK [Move PTR to next node]
    [End of Step 10 If]
    [End of While Loop]
16. Print: ITEM deleted
    [End of Step 1 If]
17. Exit

v. **Deletion at end :** Here START is a pointer variable which contains the address of first node. PTR is a pointer variable which contains address of node to be deleted. PREV is a pointer variable which points to previous node. ITEM is the value to be deleted.

1. If (START == NULL) Then [Check whether list is empty]
2. Print: Linked-List is empty.
3. Else
4. PTR = START, PREV = START
5. Repeat While (PTR -> LINK != NULL)
6. PREV = PTR [Assign PTR to PREV]
7. PTR = PTR -> LINK [Move PTR to next node]
   [End of While Loop]
8. ITEM = PTR -> INFO [Assign INFO of last node to ITEM]
9. If (START -> LINK == NULL) Then
   [If only one node is left]
10. START = NULL [Assign NULL to START]
11. Else
9. PREV -> LINK = NULL
   [Assign NULL to link field of second last node]
   [End of Step 9 If]
10. Delete PTR
11. Print : ITEM deleted
    [End of Step 1 If]
12. Exit

Que 1.28. **Implement linear linked list using pointer for following functions :**

i. **Insert at beginning** **ii.** **Insert at end**
iii. **Insert after element** **iv.** **Delete at end**
v. **Delete at beginning** **vi.** **Delete after element**
vii. **Display in reverse order**

Answer

```
#include<stdio.h>
#include<conio.h>
#include<process.h>
typedef struct simplelink {
int data;
struct simplelink *next;
```

```
} node;
```

i. **Function to insert at beginning :**

```
node *insert_begin(node *p)
{
node *temp;
temp = (node *)malloc(sizeof(node));
printf("\nEnter the inserted data:");
scanf("%d",&temp->data);
temp->next = p;
p = temp;
return(p);
}
```

ii. **Function to insert at end :**

```
node *insert_end(node *p){
node *temp, *q;
q = p;
temp=(node*)malloc(sizeof(node));
printf("\nEnter the inserted data;");
scanf("%d",&temp->data);
while(p->next != NULL)
{
p = p->next;
}
p->next = temp;
temp->next = (node *)NULL;
return(q);
}
```

iii. **Function to insert after element:**

```
node *insert_after(node *p) {
node temp, *q;
int x;
q = p;
printf("\nEnter the data(after which you want to enter data):");
scanf("%d",&x);
while(p->data != x) {
p = p->next;
}
temp = (node *)malloc(sizeof(node));
printf("\nEnter the inserted data:");
scanf("%d",&temp->data);
temp->next = p->next;
p->next = temp;
return (q);
}
```

iv. **Function to delete last node :**

```
node *del end(node *p) {
```

```
node * q, *r;
r = p;
q = p;
if(p->next == NULL)
{
r = (node *)NULL;
}
else
{
while(p->next := NULL)
{
q = p;
p = p->next;
}
q->next = (node *)NULL;
}
free(p);
return(r);
}
```

v. Function to delete first node :

```
node *delete_begin(node *p) {
node *q;
q = p;
q = p->next;
free(q);
return(p);
}
```

vi. Function to delete node after element :

```
node "delete_after(node, *p)
{
node *temp, *q;
int x;
q = p;
printf("\nEnter the data(after which you want to delete):");
scanf("%d" ,&x);
while(p->data != x) {
p = p->next;
}
temp = p->next;
p->next = temp->next;
free(temp);
return (q);
}
```

vii. Function to reverse the list :

```
node *reverse(node *p) {
node *q, *r;
```

```
q = (node *)NULL;
while(p != NULL) {
r = q;
q = p;
p = p->next;
p->next = r;
}
return(q);
}
```

Que 1.29. What are the advantages and disadvantages of single linked list ?

Answer

Advantages :

1. Linked lists are dynamic data structures as it can grow or shrink during the execution of a program.
2. The size is not fixed.
3. Data can store non-continuous memory blocks.
4. Insertion and deletion of nodes are easier and efficient. Unlike array a linked list provides flexibility in inserting a node at any specified position and a node can be deleted from any position in the linked list.
5. Many more complex applications can be easily carried out with linked lists.

Disadvantages :

1. **More memory :** In the linked list, there is a special field called link field which holds address of the next node, so linked list requires extra space.
2. Accessing to arbitrary data item is complicated and time consuming task.

Que 1.30. Write an algorithm that reverses order of all the elements in a singly linked list.

Answer

1. To reverse a linear linked list, three pointer fields are used.
2. These are PREV, PTR, REV which hold the address of previous node, current node and will maintain the linked list.

Algorithm :

1. PTR = FIRST
2. TPT = NULL
3. Repeat step 4 while PTR != NULL
4. REV = PREV

5. PREV = PTR
6. PTR = PTR –> LINK
7. PREV –> LINK = REV
   [End of while loop]
8. START = PREV
9. Exit

Que 1.31. **Write difference between array and linked list.**

AKTU 2014-15, Marks 05

Answer

S. No.	Array	Linked list
1.	An array is a list of finite number of elements of same data type *i.e.*, integer, real or string etc.	A linked list is a linear collection of data elements called nodes which are connected by links.
2.	Elements can be accessed randomly.	Elements cannot be accessed randomly. It can be accessed only sequentially.
3.	Array is classified as : a. 1-D array b. 2-D array c. *n*-D array	A linked list can be linear, doubly or circular linked list.
4.	Each array element is independent and does not have a connection with previous element or with its location.	Location or address of element is stored in the link part of previous element or node.
5.	Array elements cannot be added, deleted once it is declared.	The nodes in the linked list can be added and deleted from the list.
6.	In array, elements can be modified easily by identifying the index value.	In linked list, modifying the node is a complex process.
7.	Pointer cannot be used in array.	Pointers are used in linked list.

PART-9

Time-Space Trade Off, Abstract Data Types (ADT).

Questions-Answers

Long Answer Type and Medium Answer Type Questions

Que 1.32. **Explain doubly linked list.**

Answer

1. The doubly or two-way linked list uses double set of pointers, one pointing to the next node and the other pointing to the preceding node.
2. In doubly linked list, all nodes are linked together by multiple links which help in accessing both the successor and predecessor node for any arbitrary node within the list.
3. Every node in the doubly linked list has three fields :

LPT	INFO	RPT

Fig. 1.32.1.

4. LPT will point to the node in the left side (or previous node) *i.e.*, LPT will hold the address of the previous node, RPT will point to the node in the right side (or next node) *i.e.*, RPT will hold the address of the next node.
5. INFO field store the information of the node.
6. A doubly linked list can be shown as follows :

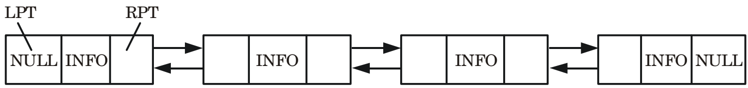


Fig. 1.32.2. Doubly linked list.

7. The structure defined for doubly linked list is:

```
struct node
    {
        int info;
        struct node *rpt;
        struct node *lpt;
    }   node;
```

Que 1.33. **What are doubly linked lists ? Write C program to create doubly linked list.** **AKTU 2015-16, Marks 10**

Answer

Doubly linked list : Refer Q. 1.32, Page 1–30A, Unit-1.

Program :

```
# include<stdio.h>
```

```
# include<conio.h>
# include<alloc.h>
struct node
    {
    int info ;
    struct node *lpt ;
    struct node *rpt ;
    } ;
struct node *first ;
void main ( )
{
create ( ) ;
getch ( ) ;
}
void create ( )
{
struct node *ptr, *cpt ;
char ch ;
ptr = (struct node *) malloc (size of (struct node)) ;
printf ("Input first node information") ;
scanf ("%d", & ptr → info) ;
ptr → lpt = NULL ;
first = ptr ;
do
{
cpt = (struct node *) malloc (size of (struct node)) ;
printf ("Input next node information");
scanf ("%d", & cpt → info) ;
ptr → rpt = cpt ;
cpt → lpt = ptr ;
ptr = cpt ;
printf ("Press <Y/N> for more node") ;
ch = getch ( );
}
while (ch == 'Y') ;
ptr → rpt = NULL ;
}
```

Que 1.34. **Implement doubly linked list using pointer for following functions :**

i. Insert at beginning
ii. Insert at end
iii. Searching an element
iv. Delete at beginning
v. Delete at end
vi. Delete entire list

Answer

```
#include<stdio.h>
#include<conio.h>
typedef struct n{
int data;
struct n *prev;
struct n *next;
}node;
node *head = NULL, *tail = NULL;
```

i. Function to insert at beginning :

```
void insert beg(node*h, int d) {
node *temp;
temp = (node *)malloc(sizeof(node));
temp->data = d;
temp->prev = NULL;
if(head == NULL)
{
temp->next = NULL;
head = tail = temp;
return;
}
temp->next = h;
h->prev = temp;
h = h->prev;
head = h;
}
```

ii. Function to insert at end :

```
void insert_end(node *t, int d) {
node *temp;
temp = (node*)malloc(sizeof(node));
temp->data = d;
temp->next = NULL;
if(head == NULL) {
temp->prev = NULL;
head = tail = temp;
return;
}
temp->prev = t;
t->next = temp;
t = t->next;
tail = t;
}
```

iii. Function to search an element :

```
node *find(node *h, int aft) {
while(h->next != head && h->data != aft)
```

```
h = h->next;
if(h->next == head && h->data != aft)
return (node*) NULL;
else
return h;
}
```

iv. Function to delete at beginning :

```
void delete_beg(node *h, node *t) {
if(head == (node*)NULL) {
printf("\nList is empty.");
getch( );
return;
}
if(head == tail) {
free(h);
head = tail = (node *)NULL;
return;
}
if(h->next == t) {
tail->prev = NULL;
head = tail;
}
else {
head = head->next;
head->prev = NULL;
}
free(h);
}
```

v. Function to delete at end :

```
void delete_end(node *h, node *t) {
if(head == (node *)NULL) {
printf("\nList is empty.");
getch( );
return;
}
if(head == tail) {
free(h);
head = tail = (node*)NULL;
return;
}
if(t->prev == h) {
head->next = NULL;
tail = head;
}
else {
tail = tail->prev;
tail->next = NULL;
```

```
}
free(t);
}
void display(node *h) {
while(h != NULL) {
printf(n"/%d", h->data);
h = h->next;
}
}
```

vi. Function to delete entire list :

```
void free_list(node *list) {
node *t;
while(list != NULL) {
t = list;
list = list->next;
free(t);
    }
    }
```

Que 1.35. **Write algorithm of following operation for doubly linked list :**

i. Traversal

ii. Insertion at beginning

iii. Delete node at specific location

iv. Deletion from end.

OR

Write an algorithm or C code to insert a node in doubly link list in beginning.

AKTU 2014-15, Marks 05

Answer

i. Traversing of two-way linked list :

a. Forward Traversing :

1. PTR ← FIRST.
2. Repeat step 3 to 4 while PTR != NULL.
3. Process INFO (PTR).
4. PTR ← RPT (PTR).
5. STOP.

b. Backward Traversing :

1. PTR ← FIRST.
2. Repeat step (3) while RPT (PTR) != NULL.
3. PTR ← RPT (PTR)
4. Repeat step (5) to (6) while PTR != NULL.
5. Process INFO (PTR).
6. PTR ←LPT (PTR).
7. STOP.

ii. Insertion at beginning :

1. IF PTR = NULL then Write OVERFLOW
   Go to Step 9
   [END OF IF]
2. SET NEW_NODE = PTR
3. SET PTR = PTR -> NEXT
4. SET NEW_NODE -> DATA = VAL
5. SET NEW_NODE -> PREV = NULL
6. SET NEW_NODE -> NEXT = START
7. SET HEAD -> PREV = NEW_NODE
8. SET HEAD = NEW_NODE
9. EXIT

iii. Delete node at specific location :

1. IF HEAD = NULL then Write UNDERFLOW
   Go to Step 9
   [END OF IF]
2. SET TEMP = HEAD
3. Repeat Step 4 while TEMP -> DATA != ITEM
4. SET TEMP = TEMP -> NEXT
   [END OF LOOP]
5. SET PTR = TEMP -> NEXT
6. SET TEMP -> NEXT = PTR -> NEXT
7. SET PTR -> NEXT -> PREV = TEMP
8. FREE PTR
9. EXIT

iv. Deletion from end :

1. IF HEAD = NULL
   Write UNDERFLOW
   Go to Step 7
   [END OF IF]
2. SET TEMP = HEAD
3. Repeat Step 4 WHILE TEMP -> NEXT != NULL
4. SET TEMP = TEMP -> NEXT
   [END OF LOOP]
5. SET TEMP -> PREV -> NEXT = NULL
6. FREE TEMP
7. EXIT

Que 1.36. **Write a program in C to delete a specific element in single linked list. Double linked list takes more space than single linked list for sorting one extra address. Under what condition, could a double linked list more beneficial than single linked list.**

AKTU 2018-19, Marks 07

Answer

Program to delete a specific element from a single linked list :

```
#include <stdio.h>
```

```
#include <stdlib.h>
// A linked list node
struct Node
{
int data;
struct Node *next;
};
/* Given a reference (pointer to pointer) to the head of a list
and an int, inserts a new node on the front of the list. */
void push(struct Node** head_ref, int new_data)
{
struct Node* new_node = (struct Node*) malloc(sizeof(struct Node));
new_node->data = new_data;
new_node->next = (*head_ref);
(*head_ref) = new_node;
}
/* Given a reference (pointer to pointer) to the head of a list
and a position, deletes the node at the given position */
void deleteNode(struct Node **head_ref, int position)
{
// If linked list is empty
if (*head_ref == NULL)
return;
// Store head node
struct Node* temp = *head_ref;
// If head needs to be removed
if (position == 0)
{
*head_ref = temp->next; // Change head
free(temp); // free old head
return;
}
// Find previous node of the node to be deleted
for (int i = 0; temp != NULL && i < position – 1; i++)
temp = temp->next;
// If position is more than number of nodes
if (temp == NULL || temp->next == NULL)
return;
// Node temp->next is the node to be deleted
// Store pointer to the next of node to be deleted
struct Node *next = temp->next->next;
// Unlink the node from linked list
free(temp->next);  // Free memory
temp->next = next;  // Unlink the deleted node from list
}
// This function prints contents of linked list starting from
// the given node
```

```
void printList(struct Node *node)
{
while (node != NULL)
{
printf("%d ", node->data);
node = node->next;
}
}
/* Program to test above functions*/
int main()
{
/* Start with the empty list */
struct Node* head = NULL;
push(&head, 7);
push(&head, 1);
push(&head, 3);
push(&head, 2);
push(&head, 8);
puts("Created Linked List: ");
printList(head);
deleteNode(&head, 4);
puts("\nLinked List after Deletion at position 4: ");
printList(head);
return 0;
}
```

Double linked list is more beneficial than single linked list because :

1. A double linked list can be traversed in both forward and backward direction.
2. The delete operation in double linked list is more efficient if pointer to the node to be deleted is given.
3. In double linked list, we can quickly insert a new node before a given node.
4. In double linked list, we can get the previous node using previous pointer but in singly liked list we traverse the list to get the previous node.

PART-10

Circular Linked List.

Questions-Answers

Long Answer Type and Medium Answer Type Questions

Que 1.37. **What is meant by circular linked list ? Write the functions to perform the following operations in a doubly linked list.**

a. Creation of list of nodes.
b. Insertion after a specified node.
c. Delete the node at a given position.
d. Sort the list according to descending order
e. Display from the beginning to end.

AKTU 2016-17, Marks 15

Answer

Circular linked list : A circular list is a linear linked list, except that the last element points to the first element, Fig. 1.37.1 shows a circular linked list with 4 nodes for non-empty circular linked list, there are no NULL pointers.

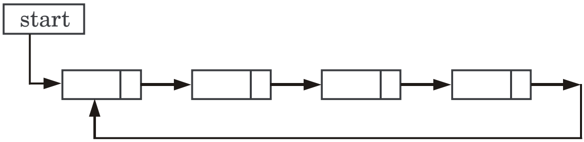


Fig. 1.37.1.

Functions :

a. To create a list : Refer Q. 1.33, Page 1–30A, Unit-1.

b. To insert after a specific node :

```
void insert_given_node ( )
{
struct node *ptr, *cpt, *tpt, *rpt, *lpt;
int m;
ptr = (struct node *) malloc (size of (struct node));
if (ptr == NULL)
{
printf ("OVERFLOW");
return;
}
printf ("input new node information");
scanf ("%d", & ptr –> info);
printf ("input node information after which insertion");
scanf ("%d", & m);
cpt = first;
while (cpt –> info != m)
cpt = cpt –> rpt;
tpt = cpt –> rpt;
cpt –> rpt = ptr;
```

```
ptr -> lpt = cpt;
ptr -> rpt = tpt;
tpt -> lpt = ptr;
printf ("Insertion is done\n");
}
```

c. To delete the node at a given position :

```
void deleteNode(int data) {
struct dllNode *nPtr, *tmp = head;
if (head == NULL) {
printf("Data unavailable\n");
return;
} else if (tmp->data == data) {
nPtr = tmp->next;
tmp->next = NULL;
free(tmp);
head = nPtr;
totNodes--;
} else {
while (tmp->next != NULL && tmp->data != data) {
nPtr = tmp;
tmp = tmp->next;
}
if (tmp->next == NULL && tmp->data != data) {
printf("Given data unavailable in list\n");
return;
} else if (tmp->next != NULL && tmp->data == data) {
nPtr->next = tmp->next;
tmp->next->previous = tmp->previous;
tmp->next = NULL;
tmp->previous = NULL;
free(tmp);
printf("Data deleted successfully\n");
totNodes --;
} else if (tmp->next == NULL && tmp->data == data) {
nPtr->next = NULL;
tmp->next = tmp->previous = NULL;
free(tmp);
printf("Data deleted successfully\n");
totNodes--;
}
}
}
```

d. To sort the list according to descending order :

```
void insertionSort() {
struct dllNode *nPtr1, *nPtr2;
int i, j, tmp;
nPtr1 = nPtr2 = head;
```

```
for (i = 0; i < totNodes; i++) {
tmp = nPtr1->data;
for (j = 0; j < i; j++)
nPtr2 = nPtr2->next;
for (j = i; j > 0 && nPtr2->previous->data < tmp; j--) {
nPtr2->data = nPtr2->previous->data;
nPtr2 = nPtr2->previous;
}
nPtr2->data = tmp;
nPtr2 = head;
nPtr1 = nPtr1->next;
}
}
```

e. To display from the beginning to end :

```
void display()
{
if(head == NULL)
printf("\nList is Empty!!!");
else
{
struct Node *temp = head;
printf("\nList elements are: \n");
printf("NULL <--- ");
while(temp -> next != NULL)
{
printf("%d <===> ",temp -> data);
}
printf("%d ---> NULL", temp -> data);
}
}
```

Que 1.38. Write a C program to implement circular linked list for following functions :

i. Searching of an element
ii. Insertion at specified position
iii. Deletion at the end
iv. Delete entire list

Answer

```
#include<stdio.h>
#include<conio.h>
typedef struct n{
int data;
struct n *next;
}node;
node *head = NULL;
```

```
void insert_cir_end node *h, int d) {
node *temp;
temp = (node*)malloc(sizeof(node));
temp->data = d;
if(head == NULL) {
head = temp;
temp->next = head;
return;
}
while(h->next != head)
h = h->next;
temp->next = h->next;
h->next = temp;
}
```

i. Function to search an element :

```
node *find(node *h, int aft) {
while(h->next != head && h->data != aft)
h = h->next;
if(h->next == head && h->data != aft)
return (node*)NULL;
else
return h;
}
```

ii. Function to insert node at specified position :

```
void insert_cirsp_pos(node *h, int pos, int d)
{
node *temp, *loc;
int p = 0;
while(h->next != head && p < pos – 1)
{
    loc = h;
    p++;
    h = h->next;
}
if(pos > pos + 1 && h->next == head) || pos < 0)
{
printf("\nPosition does not exists.");
getch( );
}
if((p + 2) == pos) {
loc = h;
}
temp = (node*)malloc(sizeof(node));
temp->data = d;
temp->next = loc->next;
if(pos == 1) {
    h = head;
```

```
        while(h->next != head)
        h = h->next;
        h->next = temp;
        head = temp;
        }
    else
        loc->next = temp;
    }
    void display (node *h) {
    while (h->next != head) {
    printf("%d", h->data);
    h = h->next;
    }
    printf("%d", h->data);
    }
```

iii. **Function to delete at the end :**

```
    void delete_cir_end(node *h) {
    node *temp;
    if(head == NULL) {
    printf("\nList is empty");
    getch( );
    }
    if(h->next == head) {
    printf("\nNode deleted. List is empty");
    getch( );
    head = NULL;
    free(h);
    return;
    }
    while(h->next != head) {
    temp = h;
    h = h->next;
    }
    temp->next = h->next;
    free(h);
    }
```

iv. **Function to delete entire list :**

```
    void free_list(node *list) {
    node *t;
    while(list != NULL) {
    t = list;
    list = list->next;
    }
    }
```

Que 1.39. **Write an algorithm to insert a node at the end in a circular linked list.** **AKTU 2017-18, Marks 07**

Answer

1. If PTR = NULL
2. Write OVERFLOW
3. Go to Step 1
   [END OF IF]
4. SET NEW_NODE = PTR
5. SET PTR = PTR -> NEXT
6. SET NEW_NODE -> DATA = VAL
7. SET NEW_NODE -> NEXT = HEAD
8. SET TEMP = HEAD
9. Repeat Step 10 while TEMP -> NEXT != HEAD
10. SET TEMP = TEMP -> NEXT
    [END OF LOOP]
11. SET TEMP -> NEXT = NEW_NODE
12. EXIT

PART- 11

Operation on a Linked List, Insertion, Deletion, Traversal Polynomial Representation and Addition, Subtraction and Multiplications of Single Variable and Two Variable Polynomial.

Questions-Answers

Long Answer Type and Medium Answer Type Questions

Que 1.40. **Write an algorithm to implement insertion, deletion and traversal on a singly linked list.**

Answer

Refer Q. 1.27, Page 1–23A, Unit-1.

Que 1.41. **Write a C program to implement insertion, deletion operation on a doubly linked list.**

Answer

Refer Q. 1.34, Page 1–31A, Unit-1.

Que 1.42. **Write a C function for traversal operation on a doubly linked list.**

Answer

Function for forward traversing :

```
void ftraverse ( )
{
    struct node *ptr;
    printf (“forward traversing :/n”);
    ptr = first ;
    while (ptr != NULL)
        {
        printf (“%d \n”, ptr -” info) ;
            ptr = ptr -> rpt;
        }
    }
```

Function for backward traversing :

```
void btraverse ( )
    {
        struct node * ptr ;
        printf (“Backward traversing :\n”)
        ptr = first ;
        while (ptr → rpt != NULL)
            ptr = ptr → rpt ;
        while (ptr != NULL)
        {
            printf (“%d \n”, ptr -> info) ;
            ptr = ptr -> lpt;
        }
    }
```

Que 1.43. **Write a program in C to implement insertion, deletion and traversal in circular linked list.**

Answer

```
#include<stdio.h>
#include<conio.h>
#include<alloc.h>
struct node
    {
        int info;
        struct node *link;
    } ;
    struct node *first;
    void main( )
```

```
    {
    void create( ), traverse( ), insert_beg( ), insert_end( ),
    delete_beg( ), delete_end( );
    clrscr( ) :
    create( ) ;
    traverse( ) ;
    insert_beg( ) ;
    traverse( ) ;
    insert_end( ) ;
    traverse( ) ;
    delete_beg( ) ;
    traverse( ) ;
    getch( ) ;
}
void create( )
{
    struct node *ptr, *cpt ;
    char ch ;
    ptr = (struct node *) malloc (size of (struct node)) ;
    printf("input first node") ;
    scanf("%d" & ptr -> info) ;
    first = ptr ;
    do {
    cpt = (struct node *) malloc (size of (struct node)) ;
    printf ("Input next node") ;
    scanf ("%d" & cpt -> info) ;
    ptr -> link = cpt ;
    ptr = cpt ;
    print f ("Press <Y/N> for more node") ;
    ch = getch ( ) ;
}
while (ch == "Y");
ptr -> link = first ;
}
void traverse ( )
{
    struct node *ptr ;
    printf ("Traversing of link list ; \n") ;
    ptr = first ;
    while = (ptr != first)
{
    printf ("%d\n", ptr -> info) :
    ptr = ptr -> link ;
}
}
void   insert_beg   (   )
{
```

```
struct node *ptr;
ptr = (struct node*) malloc (sizeof (struct node));
if (ptr == NULL)
{    printf ("overflow\n" ) ;
     return ;
}
printf ("Input New Node");
scanf ("%d", &ptr -> info) ;
cpt = first ;
while (cpt -> link != first)
     {
     cpt = cpt -> link ;
     }
ptr -> Link = first;
first = ptr ;
cpt -> link = first ;
}
void insert_end( )
{
struct node *ptr; *cpt;
ptr = (struct node*) malloc (sizeof (struct node));
if (ptr == NULL)
{
printf("overflow\n") ;
return ;
}
printf ("Input New Node information");
scanf ("%d", &ptr -> info);
cpt = first;
while (cpt -> link != first) ;
cpt = cpt -> link;
cpt -> link = ptr;
ptr -> link = first ;
}
void delete_beg ( )
{
struct node *ptr, *cpt ;
if (first == NULL)
{
printf ("underflow\n") ;
return;
}
cpt = first;
while (cpt -> link != First)
     cpt = cpt -> link ;
first = ptr -> link;
cpt -> link = first ;
```

```
free (ptr) ;
}
void delete_end( )
{
struct node *ptr, *cpt;
if (first == NULL)
{
    printf ("underflow\n");
    return;
}
cpt = first;
while (cpt -> link != first)
{
ptr = cpt;
cpt = cpt -> link;
}
ptr -> link = first;
free (cpt);
    }
```

Que 1.44. **Explain the method to represent the polynomial equation using linked list.**

Answer

1. In the linked representation of polynomials, each node should consist of three elements, namely coefficient, exponent and a link to the next term.
2. The coefficient field holds the value of the coefficient of a term, the exponent field contains the exponent value of that term and the link field contains the address of the next term in the polynomial.

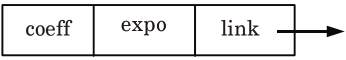


Fig. 1.44.1.

For example : Let us consider the polynomial of degree 4 *i.e.*, $3x^4 + 8x^2 + 6x + 8$ can be written as 3 * power $(x, 4)$ + 8 * power $(x, 2)$ + 6 * power $(x, 1)$ + 8 * power $(x, 0)$

It can be represented as linked list as

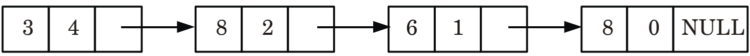


Fig. 1.44.2.

3. The link coming out of the last node is NULL pointer.
   In case of polynomial of 3 variables *i.e.*, x, y, z can also be represented as linked list as shown in Fig. 1.44.3.

power x	power y	power z	coeff	next

Fig. 1.44.3.

For example : Let us consider the following polynomial of 3 variable $3x^2 + 2xy^2 + 5y^3 + 7yz$.

We can replace each term of the polynomial with node of the linked list as

2	0	0	3	→	1	2	0	2	→	0	3	0	5	→	0	1	1	7	NULL

Fig. 1.44.4.

Que 1.45. Explain the method to represent the polynomial equation using linked list. Write and explain method to add two polynomial equations using linked list.

Answer

Representation of polynomial : Refer Q. 1.44, Page 1–47A, Unit-1.

Addition of two polynomials using linked lists :

Let p and q be the two polynomials represented by the linked list.

1. While p and q are not null, repeat step 2.
2. If powers of the two terms are equal then,

   if the terms do not cancel then insert the sum of the terms into the sum (resultant)

   Polynomial

   Update p

   Update q

   Else if the (power of the first polynomial) > (power of second polynomial)

   Then insert the term from first polynomial into sum polynomial

   Update p

   Else insert the term from second polynomial into sum polynomial

   Update q
3. Copy the remaining terms from the non-empty polynomial into the sum polynomial.

Example : Let us consider the addition of two polynomials of single variable $5x^4 + 6x^3 + 2x^2 + 10x + 4$ and $7x^3 + 3x^2 + x + 7$. We can visualize this as follows :

$$\begin{array}{r} 5x^4 + 6x^3 + 2x^2 + 10x + 4 \\ + 7x^3 + 3x^2 + x + 7 \\ \hline 5x^4 + 13x^3 + 5x^2 + 11x + 11 \\ \hline \end{array}$$

i.e., to add two polynomials, compare their corresponding terms starting from the first node and move towards the end node.

Que 1.46. **Write and explain method to multiply polynomial equation using linked list.**

Answer

1. The multiplication of polynomials is performed by multiplying coefficient and adding the respective power.
2. To produce the multiplication of two polynomials following steps are performed :
   a. Check whether two given polynomials are non-empty. If anyone polynomial is empty then polynomial multiplication is not possible. So exit.
   b. Second polynomial is scanned from left to right.
   c. For each term of the second polynomial, the first polynomial is scanned from left to right and its each term is multiplied by the term of the second polynomial, *i.e.*, find the coefficient by multiplying the coefficients and find the exponent by adding the exponents.
   d. If the product term already exists in the resulting polynomial then its coefficients are added, otherwise a new node is inserted to represent this product term.

   For example : Let us consider two polynomial $8x^4 + 6x^2 + 5x + 2$ and $3x^2 + x + 2$ and perform multiplication as

$$
\begin{array}{r}
8x^4 + 6x^2 + 5x + 2 \\
\times\ 3x^2 + x + 2 \\
\hline
24x^6 \quad\quad + 18x^4 + 15x^3 + 6x^2 \quad\quad\quad\quad \\
+ 8x^5 \quad\quad + 6x^3 + 5x^2 + 2x \\
+16x^4 \quad\quad + 12x^2 + 10x + 4 \\
\hline
24x^6 + 8x^5 + 34x^4 + 21x^3 + 23x^2 + 12x + 4 \\
\hline
\end{array}
$$

VERY IMPORTANT QUESTIONS

Following questions are very important. These questions may be asked in your SESSIONALS as well as UNIVERSITY EXAMINATION.

Q. 1. Define data structure. Describe about its need and types. Why do we need a data type ?

Ans. Refer Q. 1.1.

Q. 2. What do you understand by complexity of an algorithm ? Compute the worst case complexity for the following C code :

```
main()
{
int s = 0, i, j, n;
for (j = 0;  j < (3 * n); j++)
{
for (i = 0; i < n; i++)
{
s = s + i;
}
printf("%d", i);
}}
```

Ans. Refer Q. 1.7.

Q. 3. How do you find the complexity of an algorithm ? What is the relation between the time and space complexities of an algorithm ? Justify your answer with an example.

Ans. Refer Q. 1.8.

Q. 4. What are the various asymptotic notations ? Explain Big O notation.

Ans. Refer Q. 1.10.

Q. 5. What do you understand by time and space trade-off ? Define the various asymptotic notations. Derive the O-notation for linear search.

Ans. Refer Q. 1.11.

Q. 6. What do you understand by time-space trade-off ? Explain best, worst and average case analysis in this respect with an example.

Ans. Refer Q. 1.12.

Q. 7. Suppose multidimensional arrays P and Q are declared as $P(-2: 2, 2: 22)$ and $Q(1: 8, -5: 5, -10 : 5)$ stored in column major order

i. Find the length of each dimension of P and Q.

ii. The number of elements in P and Q.

iii. Assuming base address (Q) = 400, W = 4, find the effective indices E_1, E_2, E_3 and address of the element $Q[3, 3, 3]$.

Ans. Refer Q. 1.21.

Q. 8. Write difference between array and linked list.

Ans. Refer Q. 1.31.

Q. 9. What are doubly linked lists ? Write C program to create doubly linked list.

Ans. Refer Q. 1.33.

Q. 10. Write an algorithm or C code to insert a node in doubly link list in beginning.

Ans. Refer Q. 1.35.

Q. 11. Write a program in C to delete a specific element in single linked list. Double linked list takes more space than single linked list for sorting one extra address. Under what condition, could a double linked list more beneficial than single linked list.

Ans. Refer Q. 1.36.

Q. 12. What is meant by circular linked list ? Write the functions to perform the following operations in a doubly linked list.

a. Creation of list of nodes.

b. Insertion after a specified node.

c. Delete the node at a given position.

d. Sort the list according to descending order

e. Display from the beginning to end.

Ans. Refer Q. 1.37.

Q. 13. Write an algorithm to insert a node at the end in a circular linked list.

Ans. Refer Q. 1.39.

☺☺☺

2 UNIT

Stacks and Queues

CONTENTS

PART-1

Stacks : Abstract Data Type, Primitive Stack Operations : Push and Pop.

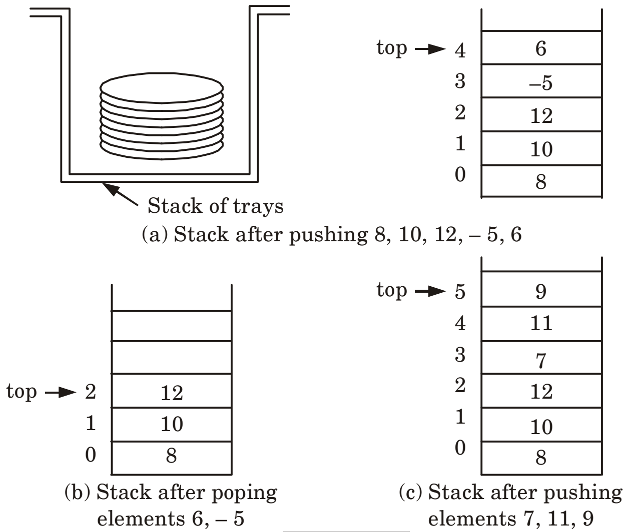
Questions-Answers

Long Answer Type and Medium Answer Type Questions

Que 2.1. What do you mean by stack ? Explain all its operation with suitable example.

Answer

1. A stack is one of the most commonly used data structure.
2. A stack, also called Last In First Out (LIFO) system, is a linear list in which insertion and deletion can take place only at one end, called top.
3. This structure operates in much the same way as stack of trays.
4. If we want to remove a tray from stack of trays it can only be removed from the top only.
5. The insertion and deletion operation in stack terminology are known as PUSH and POP operations.



(a) Stack after pushing 8, 10, 12, – 5, 6

(b) Stack after poping elements 6, – 5

(c) Stack after pushing elements 7, 11, 9

Fig. 2.1.1.

6. Following operation can be performed on stack :
   - **i. Create stack (*s*) :** To create an empty stack *s*.
   - **ii. PUSH (*s*, *i*) :** To push an element *i* into stack *s*.
   - **iii. POP (*s*) :** To access and remove the top element of the stack *s*.
   - **iv. Peek (*s*) :** To access the top element of stack *s* without removing it from the stack *s*.
   - **v. Overflow :** To check whether the stack is full.
   - **vi. Underflow :** To check whether the stack is empty.

Que 2.2. **Write short note on abstract data type.**

Answer

Refer Q. 1.13, Page 1–12A, Unit-1.

Que 2.3. **Discuss PUSH and POP operation in stack and write down their algorithm.**

Answer

PUSH operation : In push operation, we insert an element onto stack. Before inserting, first we increase the top pointer and then insert the element.

Algorithm :

PUSH (STACK, TOP, MAX, DATA)

1. If TOP = MAX – 1 then write "STACK OVERFLOW and STOP"
2. READ DATA
3. TOP ← TOP + 1
4. STACK [TOP] ← DATA
5. STOP

POP operation : In pop operation, we remove an element from stack. After every pop operation top of stack is decremented by 1.

Algorithm :

POP (STACK, TOP, ITEM)

1. If TOP < 0 then write "STACK UNDERFLOW and STOP"
2. STACK [TOP] ← NULL
3. TOP ← TOP – 1
4. STOP

PART-2

Arrays and Linked Implementation of Stack in C.

Questions-Answers

Long Answer Type and Medium Answer Type Questions

Que 2.4. **Write a C function for array implementation of stack. Write all primitive operations.** **AKTU 2015-16, Marks 10**

Answer

```
#include<stdio.h>
#include<conio.h>
#define  MAX  50
      int  stack  [MAX + 1],  top = 0 ;
      void main( )
      {
            clrscr( );
            void create( ), traverse( ), push( ), pop( );
            create( );
            printf("\n stack is:\n");
            traverse( );
            push( );
            printf("After  Push the element in the stack is :\n");
            traverse( );
            pop( );
            printf("After Pop the element in the stack is :\n");
            traverse( );
            getch( );
      }
      void create( )
      {
            char ch;
            do
            {
            top ++;
            printf("Input Element");
            scanf ("%d", stack[top]);
            printf("Press<Y>for more element\n");
            ch = getch( );
            }
      while (ch == 'Y' )
      }
      void traverse( )
      {
```

```
        int i ;
        for(i = top; i > 0; – –i)
        printf("%d\n", stack[i]);
}
void push( )
{
        int m;
        if(top == MAX)
{
        printf("Stack is overflow");
        return;
}
        printf("Input new element to insert");
        scanf("%d", &m) ;
        top++;
        stack[top] = m;
}
void pop( )
{
        if(top == 0)
{
        printf("Stack is underflow\n");
        return;
}
        stack[top] = '\0' ;
        top – – ;
}
```

Que 2.5. **Write a C function for linked list implementation of stack. Write all the primitive operations.**

AKTU 2015-16, Marks 10

Answer

```
#include<stdio.h>
#include<conio.h>
#include<alloc.h>
    struct node
    {
    int info;
    struct node *link;
    };
        struct node *top;
        void main()
        {
        void create(), traverse(), push(), pop();
        create();
```

```
printf("\n stack is :\n");
traverse();
pop();
printf("After push the element in the stack is : \n") ;
traverse();
pop( );
printf("After pop the element in the stack is : \n")
traverse();
getch();
}
void create()
{
struct node *ptr, *cpt;
char ch;
ptr = (struct node *) malloc (sizeof (struct node));
printf("Input first info");
scanf("%d", &ptr -> info);
ptr ->link = NULL;
do
{
cpt = (struct node *) malloc (sizeof (struct node));
printf("Input next information");
scanf("%d", &cpt -> info);
cpt -> link = ptr;
ptr = cpt;
printf("Press <Y/N> for more information");
ch = getch() ;
}
while (ch == 'Y')
top = ptr;
}
void traverse()
{
struct node *ptr ;
printf ("Traversing of stack : \n");
ptr = top ;
while (ptr != NULL)
{
printf ("%d\n", ptr -> info);
ptr = ptr ->link;
}
}
void push()
{
struct node *ptr;
ptr = (struct node *) malloc (sizeof (struct node));
if(ptr == NULL)
```

```
        {
        printf("Overflow\n");
        return;
        }
        printf("Input New node information");
        scanf("%d", &ptr -> info);
        ptr ->link = top;
        top = ptr;
        }
    void pop()
        {
        struct node *ptr;
        if(top == NULL)
        {
        printf ("Underflow \n");
        return;
        }
        ptr = top;
        top = ptr ->link;
        free (ptr);
        }
```

Que 2.6. **What is stack ? Implement stack with singly linked list.**

AKTU 2014-15, Marks 05

Answer

Stack : Refer Q. 2.1, Page, 2–2A Unit-2.

Implementation using singly linked list :

```
typedef struct stack
{
    int *data;
    struct stack *next;
    }stack;
void push(stack **top, int *data)
    {
    stack *newn;
    newn = (stack *)malloc(sizeof(stack));
    newn->data = data;
    newn->next = (stack *)NULL;
    if(*top == NULL)
    {
    *top = newn;
    return;
    }
    newn->next = (*top);
    *top = newn;
```

```
}
int *pop(stack **top)
{
int *rval = (int *)NULL;
stack *tmp;
if(*top != NULL)
{
      tmp = *top;
      *top = (*top)->next;
      rval = tmp->data;
      free(tmp);
}
      return(rval);
}
```

Que 2.7. **Write a function in C language to reverse a string using stack.**

AKTU 2014-15, Marks 05

OR

What is a stack ? Write a C program to reverse a string using stack.

AKTU 2017-18, Marks 07

Answer

Stack : Refer Q. 2.1, Page 2–2A, Unit-2.

```
#include<stdio.h>
#include<conio.h>
#include<string.h>
#define MAX 20
   int top = – 1;
   char stack [MAX];
   char pop();
   push(char);
   main()
      {
      clrscr();
      char str [20];
      int i;
      printf("Enter the string : ");
      gets(str);
      for(i = 0; i < strlen(str); i++)
      push (str [i]);
      for(i = 0; i < strlen(str); i++)
      str[i] = pop();
      printf("Reversed string is :");
```

```
        puts (str);
        getch();
        }
push (char item)
        {
        if(top == MAX – 1)
        printf("Stack overflow\n");
        else
        stack[++top] = item;
        }
char pop()
        {
        if(top == – 1)
        printf("Stack underflow \n");
        else
        return stack [top – –];
        }
```

PART-3

Application of Stack : Prefix and Postfix Expression Evaluation of Postfix Expression.

Questions-Answers

Long Answer Type and Medium Answer Type Questions

Que 2.8. **Write a short note on the application of stack.**

Answer

Applications of stack are as follows :

1. **Expression evaluation :** Stack is used to evaluate prefix, postfix and infix expressions.
2. **Expression conversion :** An expression can be represented in prefix, postfix or infix notation. Stack can be used to convert one form of expression to another.
3. **Syntax parsing :** Many compilers use a stack for parsing the syntax of expressions, program blocks etc. before translating into low level code.
4. **Parenthesis checking :** Stack is used to check the proper opening and closing of parenthesis.

5. **String reversal :** Stack is used to reverse a string. We push the characters of string one by one into stack and then pop character from stack.
6. **Function call :** Stack is used to keep information about the active functions or subroutines.

Que 2.9. **Write down the algorithm to convert infix notation into postfix.**

OR

Write down algorithm to evaluate the infix expression.

Answer

Polish (Q, P)

Let Q is an arithmetic expression written in infix notation. This algorithm finds the equivalent postfix expression P.

1. Push "("onto STACK, and add ")" to end of Q.
2. Scan Q from left to right and repeat steps 3 to 6 for each element of Q until the STACK is empty.
3. If an operand is encountered, add it to P.
4. If a left parenthesis is encountered push it onto STACK.
5. If an operator $\otimes$ is encountered, then :
   a. Repeatedly pop from STACK and add to P each operator (on the top of STACK) which has the same precedence as or higher precedence than $\otimes$.
   b. Add $\otimes$ to STACK.

      [End if]
6. If a right parenthesis is encountered, then :
   a. Repeatedly pop from STACK and add to P each operator (on the top of STACK) until a left parenthesis is encountered.
   b. Remove the left parenthesis [Do not add it to P]

      [End if]

      [End of step 2]
7. End.

Que 2.10. **Convert following infix expression into postfix expression A + (B * C + D)/E.** **AKTU 2014-15, Marks 05**

Answer

(A + (B *C + D)/E)

Character	Stack	Postfix
(	(	
A	(	A
+	(+	A
(	(+(	A
B	(+(	AB
*	(+(*	AB
C	(+(*	ABC
+	(+(+	ABC*
D	(+(+	ABC*D
)	(+	ABC*D+
/	(+/	ABC*D+
E	(+/	ABC*D+E
)	(	ABC*D+E/+

Resultant postfix expression : ABC * D + E/+

Que 2.11. Consider the following infix expression and convert into reverse polish notation using stack. ***A*** **+ (*B* * *C* – (*D*/*E* ^ *F*) * *H*)**

AKTU 2018-19, Marks 07

Answer

A + (*B***C* – (*D*/*E* ^ *F*)**H*)

Character	Stack	Postfix
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 ^/
H	(+ (– *	ABC*DEF ^/ H

Resultant reverse polish expression : *ABC* * *DEF* ^ / *H*

Que 2.12. Write down the algorithm to evaluate the postfix expression.

OR

Write down the algorithm to convert postfix to infix.

Answer

This algorithm finds the value of an arithmetic expression *P* written in postfix notation.

1. Add a right parenthesis ")" to *P*.
   [This acts as a sentinel]
2. Scan *P* from left to right and repeat step 3 and 4 for each element of *P* until the sentinel ")" is encountered.
3. If an operand is encountered, put it on STACK.
4. If an operator ⊗ is encountered then :
   a. Remove the top two elements of STACK, where *A* is the top element and *B* is the next-to-top element.
   b. Evaluate $B \otimes A$.
   c. Place the result of (b) back on STACK.
      [End of if structure]
      [End of step 2 loop]
5. Set value equal to top element on STACK.
6. End.

Que 2.13. Consider the following arithmetic expression written in infix notation :

$$\mathbf{E = (A + B) * C + D / (B + A * C) + D}$$
$$\mathbf{E = A/B \wedge C + D * E - A * C}$$

Convert the above expression into postfix and prefix notation.

Answer

a. **E = (A + B) * C + D / (B + A * C) + D**

Postfix :

$$
\begin{aligned}
E &= (A + B) * C + D / (B + T_1) + D & T_1 &= AC\,* \\
&= (A + B) * C + D / T_2 + D & T_2 &= BT_1 + \\
&= T_3 * C + D / T_2 + D & T_3 &= AB + \\
&= T_3 * C + T_4 + D & T_4 &= DT_2/ \\
&= T_5 + T_4 + D & T_5 &= T_3\,C\,* \\
&= T_6 + D & T_6 &= T_5T_4 + \\
&= T_7 & T_7 &= T_6D +
\end{aligned}
$$

On putting the values of T's

$$
\begin{aligned}
&= T_6D + \\
&= T_5T_4 + D + \\
&= T_3\,C * DT_2 / + D + \\
&= AB + C * DBT_1 + / + D + \\
&= AB + C * DBAC * + / + D +
\end{aligned}
$$

Prefix : $E = (A + B) * C + D / (B + A * C) + D$

$= (A + B) * C + D / (B + T_1) + D$ $\quad T_1 = * AC$

$= (A + B) * C + D / T_2 + D$ $\quad T_2 = + BT_1$

$= T_3 * C + D / T_2 + D$ $\quad T_3 = + AB$

$= T_3 * C + T_4 + D$ $\quad T_4 = / DT_2$

$= T_5 + T_4 + D$ $\quad T_5 = * T_3C$

$= T_6 + D$ $\quad T_6 = + T_5T_4$

$= T_7$ $\quad T_7 = + T_6D$

On putting the values of T's

$E = + T_6D$

$= + + T_5 T_4D$

$= + + * T_3C / DT_2D$

$= + + * + ABC / D + B T_1D$

$= + + * + ABC / D + B * ACD$

b. E = A /B ∧ C + D * E – A * C

Postfix : $E = A / T_1 + D * E - A * C$ $\quad T_1 = BC$ ^

$= T_2 + D * E - A * C$ $\quad T_2 = AT_1/$

$= T_2 + T_3 - A * C$ $\quad T_3 = DE *$

$= T_2 + T_3 - T_4$ $\quad T_4 = AC *$

$= T_5 - T_4$ $\quad T_5 = T_2T_3 +$

$= T_6$ $\quad T_6 = T_5 T_4 -$

On putting the values of T's

$= T_5 T_4 -$

$= T_2 T_3 + AC *$

$= AT_1/DE * + AC *$

$= ABC \wedge /DE * + AC *$

Prefix : $E = A /B \wedge C + D * E - A * C$

$= A / T_1 + D * E - A * C$ $\quad T_1 = \wedge BC$

$= T_2 + D * E - A * C$ $\quad T_2 = /AT_1$

$= T_2 + T_3 - A * C$ $\quad T_3 = * DE$

$= T_2 + T_3 - T_4$ $\quad T_4 = * AC$

$= T_5 - T_4$ $\quad T_5 = + T_2T_3$

$= T_6$ $\quad T_6 = - T_5 T_4$

On putting the values of T's

$= - T_5T_4$

$= - + T_2T_3 * AC$

$= - + / AT_1 * DE * AC$

$= - + / A \wedge BC * DE * AC$

Que 2.14. **Solve the following :**

a. **((A – (B + C) * D) / (E + F)) [Infix to postfix]**

b. **(A + B) + *C – (D – E) ∧ F [Infix to prefix]**

c. **7 5 2 + * 4 1 5 – / – [Evaluate the given postfix expression]**

AKTU 2016-17, Marks 10

Answer

a. **((A – (B + C)*D)/(E + F))**

$((A - (B + C)*D)/\underbrace{(EF +)}_{X})$

$((A - (B + C)*D)/X)$

$((A - \underbrace{(BC +)}_{Y} * D) / X)$

$((A - (Y * D)) / X$

$((A - \underbrace{(YD *)}_{Z}) / X)$

$((A - Z)) / X)$

$(\underbrace{(AZ -)}_{T} / X)$

(T / X)

$T X /$

Now put the values,

$AZ - EF + /$

$AYD * - EF + /$

$ABC + D * - EF + /$

This is the required postfix form.

b. **(A + B) + *C – (D – E) ∧ F**

$\underbrace{(+ AB)}_{X} + * C - (D - E) \wedge F$

$X + * C - (D - E) \wedge F$

$X + * C - \underbrace{(- DE)}_{Y} \wedge F$

$X + * C - (Y \wedge F)$

$X + * C - \underbrace{(Y \wedge F)}_{Z}$

$X + *(C - Z)$

$X + * \underbrace{(-\ CZ)}_{T}$

X + * T

* + XT

Now put the values,

* + + AB – CZ

* + + AB – C ∧ YF

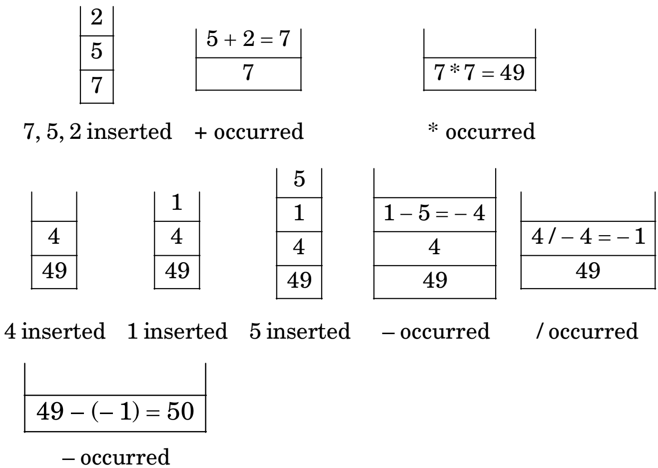
* + + AB – C ∧ – DEF

This is the required prefix form.

c. 752 + * 415 – / – :

i. First this expression is converted into infix expression as :

Symbol scanned	**Stack**
7	7
5	7, 5
2	7, 5, 2
+	7, 5 + 2
*	7*(5 + 2)
4	7*(5 + 2), 4
1	7*(5 + 2), 4, 1
5	7*(5 + 2), 4, 1, 5
–	7*(5 + 2), 4, 1 – 5
/	7*(5 + 2), 4/(1 – 5)
–	(7*(5 + 2)) – (4/(1 – 5))



Hence, the value is 50

PART-4

Iteration and Recursion, Principles of Recursion, Tail Recursion, Removal of Recursion Problem Solving Using Iteration and Recursion with Examples such as Binary Search, Fibonacci Number and Hanoi Towers, Trade off between Iteration and Recursion.

Questions-Answers

Long Answer Type and Medium Answer Type Questions

Que 2.15. What is iteration ? Explain.

Answer

1. Iteration is the repetition of a process in order to generate a (possibly unbounded) sequence of outcomes.
2. The sequence will approach some end point or end value.
3. Each repetition of the process is a single iteration, and the result of each iteration is then the starting point of the next iteration.
4. Iteration allows us to simplify our algorithm by stating that we will repeat certain steps until told.
5. This makes designing algorithms quicker and simpler because they do not have to include lots of unnecessary steps.
6. Iteration is used in computer programs to repeat a set of instructions.
7. Count controlled iteration will repeat a set of instructions upto a specific number of times, while condition controlled iteration will repeat the instructions until a specific condition is met.

Que 2.16. What is recursion ? Explain.

Answer

1. Recursion is a process of expressing a function that calls itself to perform specific operation.
2. Indirect recursion occurs when one function calls another function that then calls the first function.
3. Suppose P is a procedure containing either a call statement to itself or a call statement to a second procedure that may eventually result in a call statement back to the original procedure P.
4. Then P is called recursive procedure. So the program will not continue to run indefinitely.

5. A recursive procedure must have the following two properties :
   a. There must be certain criteria, called base criteria, for which the procedure does not call itself.
   b. Each time the procedure does call itself, it must be closer to the criteria.
6. A recursive procedure with these two properties is said to be well-defined.
7. Similarly, a function is said to be recursively defined if the function definition refers to itself.
8. Again, in order for the definition not to be circular, it must have the following two properties :
   a. There must be certain arguments, called base values, for which the function does not refer to itself.
   b. Each time the function does refer to itself, the argument of the function must be closer to a base value.

Que 2.17. **What is recursion ? Write a recursive program to find sum of digits of the given number. Also, calculate the time complexity.** **AKTU 2016-17, Marks 10**

Answer

Recursion : Refer Q. 2.16, Page 2–16A, Unit-2.

Program :

```
#include<stdio.h>
#include<conio.h>
int sum(int n)
{
    if(n < 10)
    return(n);
    else
    return(n % 10 + sum (n / 10));
}
main()
{
int s,n;
printf("\nEnter any number:");
scanf("%d",&n);
s = sum(n);
printf("\nSum of digits = %d", s);
getch();
return 0;
}
```

Time complexity :

i. Assume that n is a 10 digit number. The function is called 10 times as the problem is reduced by a factor of 10 each time the program recurse.

ii. So, we can conclude that time taken by program is linear in terms of the length of the digit of the input number n.

iii. So, time complexity is,

T(n) = O(length of digit of (n)) where n is the number whose sum of individual digit is to be found.

Que 2.18. Explain all types of recursion with example.

Answer

Types of recursion :

a. **Direct recursion :** A function is directly recursive if it contains an explicit call to itself.

For example :

```
int foo (int x)
{ if (x <= 0)
return x;
return foo (x – 1);
}
```

b. **Indirect recursion:** A function is indirectly recursive if it contains a call to another function.

For example :

```
int foo (int x)
{ if (x <= 0)
return x;
return bar (x) ;
}
int bar (int y)
{ return foo (y – 1) ;
}
```

c. **Tail recursion :**

1. Tail recursion (or tail-end recursion) is a special case of recursion in which the last operation of the function, the tail call is a recursive call. Such recursions can be easily transformed to iterations.

2. Replacing recursion with iteration, manually or automatically, can drastically decrease the amount of stack space used and improve efficiency.

3. Converting a call to a branch or jump in such a case is called a tail call optimization.

For example :

Consider this recursive definition of the factorial function in C :

factorial (n)

```
{
    if( n == 0)
    return 1;
    return n * factorial (n – 1);
    }
```

4. This definition is tail recursive since the recursive call to factorial is not the last thing in the function (its result has to be multiplied by *n*).

```
factorial (n, accumulator)
{
    if(n == 0)
    return accumulator;
return factorial (n – 1, n * accumulator);
}
factorial (n)
{
    return factorial (n – 1) ;
}
```

d. Linear and tree recursive :

1. A recursive function is said to be linearly recursive when no pending operation involves another recursive call to the function.
2. A recursive function is said to be tree recursive (or non-linearly recursive) when the pending operation does involve another recursive call to the function.
3. The Fibonacci function fib provides a classic example of tree recursion. The Fibonacci numbers can be defined by the rule :

```
int fib (int n)
{ /* n >= 0 */
if (n == 0)
return 0;
if (n == 1)
return 1;
return fib (n – 1) + fib (n – 2) ;
}
```

The pending operation for the recursive call is another call to fib. Therefore, fib is tree recursive.

Que 2.19. **Explain Tower of Hanoi.**

Answer

1. Suppose three pegs, labelled A, B and C is given, and suppose on peg A, there are finite number of *n* disks with decreasing size.
2. The object of the game is to move the disks from peg A to peg C using peg B as an auxiliary.

3. The rule of game is follows :
   a. Only one disk may be moved at a time. Specifically only the top disk on any peg may be moved to any other peg.
   b. At no time, can a larger disk be placed on a smaller disk.

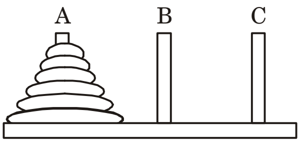


Fig. 2.19.1.

The solution to the Tower of Hanoi problem for $n = 3$.

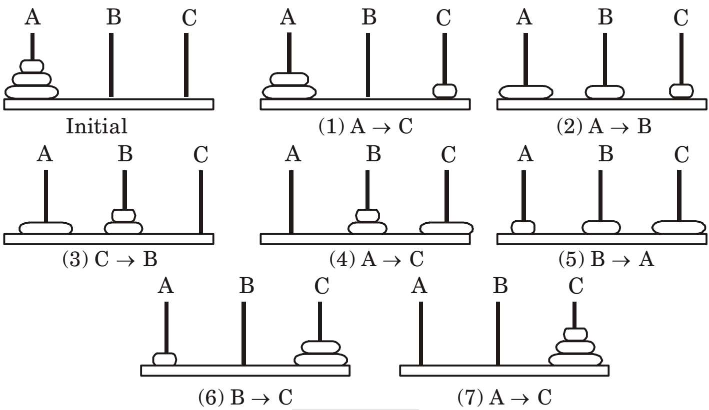


Fig. 2.19.2.

Total number of steps to solve Tower of Hanoi problem of n disk

$$= 2^n - 1 = 2^3 - 1 = 7$$

Que 2.20. What is Tower of Hanoi problem ? Write the recursive code in C language for the problem. **AKTU 2014-15, Marks 05**

Answer

Tower of Hanoi problem : Refer Q. 2.19, Page 2–19A, Unit-2.

Recursive code for Tower of Hanoi :

```
#include<stdio.h>
#include<conio.h>
void main()
```

```
        {
        clrscr( );
        int n;
        char A = 'A', B = 'B', C = 'C';
        void hanoi (int, char, char, char);
        printf("Enter number of disks :");
        scanf("%d", &n);
        printf("\n\n Tower of Hanoi problem with %d disks\n", n);
        printf("Sequence is : \n");
         hanoi (n, A, B, C);
                printf("\n");
                     getch();
        }
    void hanoi (int n, char A, char B, char C)
        {
        If(n ! = 0)
        {
        hanoi (n – 1, A, C, B);
        printf("Move disk %d from %c to %c\n , n, A, C,");
        hanoi (n – 1, B, A, C);
        }
        }
```

Que 2.21. **Write a recursive algorithm for solving the problem of Tower of Hanoi and also explain its complexity. Illustrate the solution for four disks and three pegs.**

OR

Explain Tower of Hanoi problem and write a recursive algorithm to solve it. **AKTU 2018-19, Marks 07**

OR

Write an algorithm for finding solution to the Tower of Hanoi problem. Explain the working of your algorithm (with 4 disks) with diagrams. **AKTU 2015-16, Marks 15**

Answer

Tower of Hanoi problem : Refer Q. 2.19, Page 2–19A, Unit-2.

Algorithm :

TOWER (N, BEG, AUX, END)

This procedure gives a recursive solution to the Tower of Hanoi problem for *N* disks.

1. If N = 1, then :
   a. Write: BEG $\to$ END
   b. Return
   [End of If structure]
2. [Move N – 1 disk from peg BEG to peg AUX]
   Call TOWER (N – 1, BEG, END, AUX)
3. Write: BEG $\to$ END
4. [Move N – 1 disk from peg AUX to peg END]
   Call TOWER (N – 1, AUX, BEG, END)
5. Return

Time complexity :

Let the time required for n disks is $T(n)$.

There are 2 recursive calls for $n - 1$ disks and one constant time operation to move a disk from 'from' peg to 'to' peg. Let it be k_l.

Therefore,

$T(n) = 2\ T(n - 1) + k_1$

$T(0) = k_2$, a constant.

$T(1) = 2k_2 + k_1$

$T(2) = 4k_2 + 2k_1 + k_1$

$T(2) = 8k_2 + 4k_1 + 2k_1 + k_1$

Coefficient of $k_1 = 2^n$

Coefficient of $k_2 = 2^n - 1$

Time complexity is $O(2^n)$ or $O(a^n)$ where a is a constant greater than 1.

So, it has exponential time complexity.

Space complexity :

Space for parameter for each call is independent of n *i.e.*, constant. Let it be k.

When we do the 2nd recursive call 1st recursive call is over. So, we can reuse the space of 1st call for 2nd call. Hence,

$T(n) = T(n - 1) + k$

$T(0) = k$

$T(1) = 2k$

$T(2) = 3k$

$T(3) = 4k$

So, the space complexity is $O(n)$.

Numerical : Fig. 2.21.1 contains a schematic illustration of the recursive solution for TOWER (4, A, B, C) (4 disks, 3 pegs)

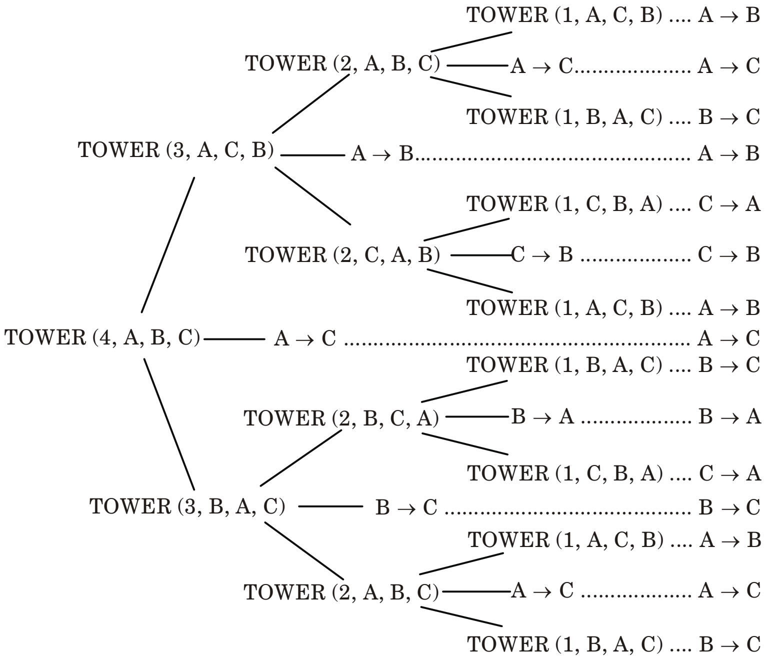


Fig. 2.21.1. Recursive solution to Tower of Hanoi problem for n = 4.

Observe that the recursive solution for n = 4 disks consist of the following 15 moves :

A → B A → C B → C A → B C → A C → B A → B A → C B → C
B → A C → A B → C A → B A → C B → C

Que 2.22. **Discuss the principle of recursion.**

Answer

1. Recursion is implemented through the use of function.
2. A function that contains a function call to itself or a function call to a second function which eventually calls the first function, is known as a recursive function.
3. Two important conditions must be satisfied by any recursive function :
   a. Each time a function calls itself it must be closer, in some sense to a solution.
   b. There must be a discussion criterion for stopping the process or computation.

Que 2.23. **How recursion can be removed ?**

Answer

There are two ways to remove recursion :

1. **By iteration :** All tail recursion function can be removed by iterative method.
2. **By using stack :** All non-tail recursion method can be removed by using stack.

Que 2.24. **Define the recursion. Write a recursive and non-recursive program to calculate the factorial of the given number.**

AKTU 2017-18, Marks 07

Answer

Recursion : Refer Q. 2.16, Page 2–16A, Unit-2.

Program :

```
#include <stdio.h>
#include <conio.h>
void main()
{
int n, a, b;
clrscr();
printf("Enter any number\n");
scanf("%d", &n);
a = recfactorial(n);
printf("The factorial of a given number using recursion is %d
\n", a);
b = nonrecfactorial(n);
printf("The factorial of a given number using nonrecursion is
%d ", b);
getch();
}
int recfactorial(int x)
{
int f;
if(x == 0)
{
return(1);
}
else
{
f = x * recfactorial(x – 1);
```

```
return(f);
}
}
int nonrecfactorial(int x)
{
int i, f = 1;
for(i = 1; i <= x; i++)
{
f = f * i;
}
return(f);
}
```

PART-5

Queues : Operation on Queue : Create, Add, Delete, Full and Empty.

Questions-Answers

Long Answer Type and Medium Answer Type Questions

Que 2.25. **Discuss queue.**

Answer

1. Queue is a linear list which has two ends, one for insertion of elements and other for deletion of elements.
2. The first end is called 'Rear' and the later is called 'Front'.
3. Elements are inserted from Rear end and deleted from Front end.
4. Queues are called First In First Out (FIFO) list, since the first element in a queue will be the first element out of the queue.
5. The two basic operations that are possible in a queue are :
   a. Insert (or add) an element to the queue (push) or Enqueue.
   b. Delete (or remove) an element from a queue (pop) or Dequeue.

Example :

Suppose we have an empty queue, with 5 memory cells such as :

0	1	2	3	4

Front = – 1

Rear = – 1 *i.e.*, Empty queue.

Que 2.26. **Write the procedures for insertion, deletion and traversal of a queue.** **AKTU 2014-15, Marks 05**

OR

Discuss various algorithms for various operation of queue.

Answer

1. **Insertion :**
   Insert in Q (Queue, Max, Front, Rear, Element)
   Let Queue is an array, Max is the maximum index of array, Front and Rear to hold the index of first and last element of Queue respectively and Element is value to be inserted.
   Step 1 : If Front = 1 and Rear = Max or if Front = Rear + 1
   Display "Overflow" and Return
   Step 2 : If Front = NULL [Queue is empty]
   Set Front = 1 and Rear = 1
   else if Rear = N, then
   Set Rear = 1
   else
   Set Rear = Rear + 1
   [End of if Structure]
   Step 3 : Set Queue [Rear] = Element [This is new element]
   Step 4 : End
2. **Deletion :**
   Delete from Q (Queue, Max, Front, Rear, Item)
   Step 1 : If Front = NULL [Queue is empty]
   display "Underflow" and Return
   Step 2 : Set Item = Queue [Front]
   Step 3 : If Front = Rear [Only one element]
   Set Front = Rear and Rear = NULL
   Else if
   Front = N, then
   Set Front = 1
   Else
   Set Front = Front + 1
   [End if structure]
   Step 4 : End
3. **Traversal of a queue :** Here queue has Front End FE and Rear End RE. This algorithm traverse queue applying an operation PROCESS to each element of queue :
   Step 1 : [Initialize counter] Set K = FE
   Step 2 : Repeat step 3 and 4 while K ≤ RE
   Step 3 : [Visit element] Apply PROCESS to queue [K]
   Step 4 : [Increase counter] Set K = K + 1
   [End of step 2 loop]
   Step 5 : Exit

PART-6

Circular Queue, Array and Linked Implementation of Queue in C.

Questions-Answers

Long Answer Type and Medium Answer Type Questions

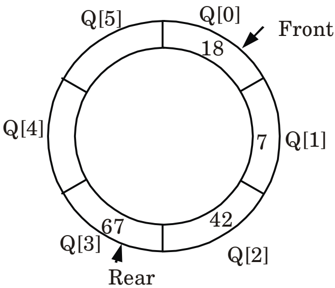
Que 2.27. **What is circular queue ? Write a C code to insert an element in circular queue. Write all the condition for overflow.**

AKTU 2014-15, Marks 05

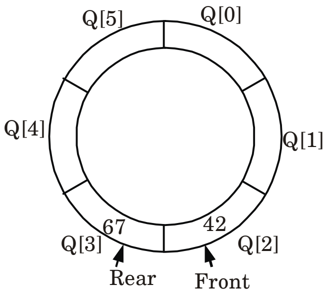
Answer

1. A circular queue is one in which the insertion of a new element is done at the very first location of the queue if the last location at the queue is full.
2. In circular queue, the elements Q[0], Q[1], Q[2] ... Q[n – 1] is represented in a circular fashion.

   For example : Suppose *Q* is a queue array of six elements.
3. PUSH and POP operation can be performed on circular queue. Fig. 2.27.1 will illustrate the same.



(a) A circular queue after inserting 18, 7 , 42, 67.



(b) A circular queue after popping 18, 7.

Fig. 2.27.1.

C code to insert an element in circular queue :

```
void insert ()
{
    int item;
    if((front == 0 && rear == Max – 1) | |((front == rear + 1))
    {
    printf("Queue is overflow\n");
    return;
```

```
}
if(front == –1)   / *If queue is empty*/
{
front = 0;
rear = 0;
}
else
if(rear == Max – 1)   /*rear is at last position of queue*/
rear = 0;
else
rear = rear + 1;
printf("Input the element for insertion :");
scanf("%d", &item);
cqueue [rear] = item;
}
```

Conditions for overflow : There are two conditions :

1. (front = 0) and (rear = Max – 1)
2. front = rear + 1

If any of these two conditions is satisfied, it means that overflow occurs.

Que 2.28. Write an algorithm to insert and delete an item from the circular linked list.

Answer

Insertion in circular linked list :

i. At the beginning :

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 (PTR) ← FIRST
   FIRST ← PTR
   LINK (CPT) ← FIRST
7. STOP

ii. 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

Deletion in circular linked list :

i. From the beginning

1. If FIRST = NULL then linked list is UNDERFLOW and STOP
2. CPT ← FIRST
3. Repeat step 4 while LINK (CPT) != FIRST
4. CPT ← LINK (CPT)
5. PTR ← FIRST
   FIRST ← LINK (PTR)
   LINK (CPT) ← FIRST
6. LINK (PRT) ← AVAIL
   AVAIL ← PTR
7. STOP

ii. From the end

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

Que 2.29. Write a C program to implement the array representation of circular queue. **AKTU 2016-17, Marks 10**

Answer

```
#include<stdio.h>
#include<conio.h>
#include<process.h>
#define MAX 10
    typedef struct {
        int front, rear ;
        int elements [MAX];
        } queue;
    void createqueue (queue *aq) {
        aq -> front = aq -> rear = – 1
    }
    int isempty (queue *aq)
    {
        if(aq -> front = = – 1)
                return 1;
        else
                return 0;
```

```
}
int isfull (queue *aq) {
        if(((aq -> front = = 0) && (aq -> rear = = MAX – 1))
                                        ||(aq - > front == aq - > rear + 1))
                    return 1;
        else
                    return 0;
}
void insert (queue *aq, int value) {
        if(aq -> front = = – 1)
                    aq -> front = aq -> rear = 0;
        else
                    aq -> rear = (aq -> rear + 1) % MAX;
                    aq -> element [aq -> rear] = value;
}
int delete (queue *aq) {
        int temp;
        temp = aq -> element [aq -> front];
        if(aq -> front = = aq ->rear)
        aq -> front = aq -> rear = – 1;
        else
        aq -> front = (aq -> front + 1) % MAX ;
        return temp;
        }
void main( )
{
        int ch, elmt;
        queue q;
        create queue (&q);
        while (1) {
        printf("1. Insertion \n");
        printf("2. Deletion \n");
        printf("3. Exit \n");
        printf("Enter your choice");
         scanf("%d",&ch) ; .
                  switch (ch)
                              {
                                case 1:
                                if(isfull (&q))
                                {
                                printf ("queue is full");
                                getch();
                                }
                                else
                                {
                                printf("Enter value");
```

```
scanf("%d", &elmt) ;
insert (&q, elmt) ;
}
break;
case 2: if (isempty (&q))
{
printf("queue empty");
getch();
}
else
{
printf("Value deleted is % d", delete (&q));
getch( );
}
break;
case 3:
exit(1);
}
} }
```

Que 2.30. **Write a C program to implement queue using linked list.**

Answer

```
#include<stdio.h>
#include<conio.h>
#include<alloc.h>
#include<stdlib.h>
    struct node
        {
            int info;
            struct node *link;
        };
        struct node *front, *rear;
        void main( )
            {
              clrscr( );
              void insert( ), delete( ), display( );
              int ch;
              while (1)
                {
            printf("1. Insert\n");
            printf("2. Delete\n");
            printf("3.Display\n");
            printf("4. Exit\n");
            printf("Enter your choice :");
            scanf("%d", &ch);
```

```
        switch(ch)
        {
          case 1  :
            insert( );
            break;
          case 2  :
            delete( );
            break;
          case 3 :
            display(  );
            break;
          case 4 :
            exit(0);
            default;
               printf("Please enter correct choice \n");
            }
        }
        getch(  );
      }
      void insert(  )
        {
          struct node *ptr;
          ptr =  (struct node*)malloc(sizeof  (struct node));
          int item;
          printf("Input the element for inserting :\n");
          scanf("%d",&item);
          prt-> info = item;
          ptr->link = NULL;
          if (front == NULL)                                    /* queue is empty*/
          front = ptr;
          else
          rear->link = ptr;
          rear = ptr;
      }
  void delete( )
      {
        struct node *ptr;
        if   (front == NULL)
      {
        printf("Queue is underflow \n") ;
        return;
      }
      if  (front == rear) {
```

```
            free(front);
            rear = NULL;
        }
        else
        {
            ptr = front;
            front = ptr->link;
            free (ptr);
        }
    }
    void display( )
    {
    struct node *ptr;
    ptr = front;
    if  (front == NULL)
        printf("Queue is empty\n");
    else
    {
        printf("\n Elements in the Queue are :\n");
        while(ptr != NULL)
            {
             printf("%d\n", ptr->info);
             ptr = ptr->link;
            }
            printf("\n");
        }
    }
```

Que 2.31. **Explain how a circular queue can be implemented using arrays. Write all functions for circular queue operations.**

AKTU 2018-19, Marks 07

Answer

Implementation of circular queue using array :
Refer Q. 2.29, Page 2–29A, Unit-2.

Function to create circular queue :

```
    void Queue :: enQueue(int value)
    {
```

```
if ((front == 0 && rear == size – 1) || (rear == (front – 1)%(size – 1)))
{
printf("\nQueue is Full");
return;
}
else if (front == –1) /* Insert First Element */
{
front = rear = 0;
arr[rear] = value;
}
else if (rear == size–1 && front != 0)
{
rear = 0;
arr[rear] = value;
}
else
{
rear++;
arr[rear] = value;
}
}
```

Function to delete element from circular queue :

```
int Queue :: deQueue()
{
if (front == –1)
{
printf("\nQueue is Empty");
return INT_MIN;
}
int data = arr[front];
arr[front] = – 1;
if (front == rear)
{
front = – 1;
rear = – 1;
}
else if (front == size – 1)
front = 0;
else
front++;
return data;
}
```

PART-7

Dequeue and Priority Queue.

Questions-Answers

Long Answer Type and Medium Answer Type Questions

Que 2.32. **Explain dequeue with its types.**

Answer

1. In a dequeue, both insertion and deletion operations are performed at either end of the queues. That is, we can insert an element from the rear end or the front end. Also deletion is possible from either end.

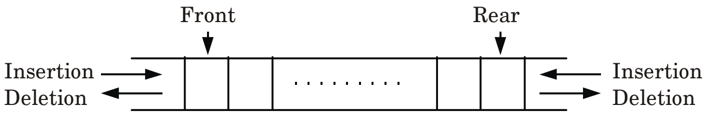
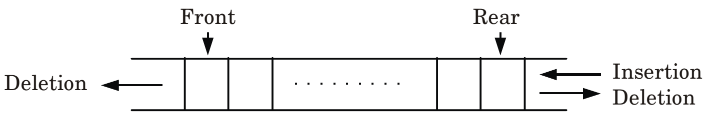


Fig. 2.32.1. Structure of a dequeue.

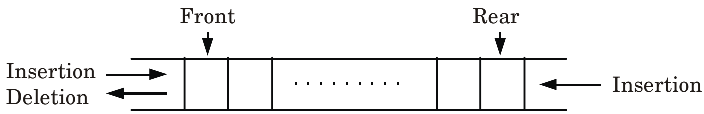
2. This dequeue can be used both as a stack and as a queue.
3. There are various ways by which this dequeue can be represented. The most common ways of representing this type of dequeue are :
   a. Using a doubly linked list
   b. Using a circular array

Types of dequeue :

1. **Input-restricted dequeue :** In input-restricted dequeue, element can be added at only one end but we can delete the element from both ends.
2. **Output-restricted dequeue :** An output-restricted dequeue is a dequeue where deletions take place at only one end but allows insertion at both ends.



(a) Input-restricted dequeue



(b) Output-restricted dequeue

Fig. 2.32.2.

Que 2.33. **What do you mean by priority queue ? Describe its applications.**

Answer

1. A priority queue is a data structure in which each element has been assigned a value called the priority of the element and an element can be inserted or deleted not only at the ends but at any position on the queue.
2. A priority queue is a collection of elements such that each element has been assigned an explicit or implicit priority and such that the order in which elements are deleted and processed comes from the following rules :
   a. An element of higher priority is processed before any element of lower priority.
   b. Two elements with the same priority are processed to the order in which they were inserted to the queue.

Types of priority queues are :

1. **Ascending priority queue :** In ascending priority queue, elements can be inserted in an order. But, while deleting elements from the queue, always a small element to be deleted first.
2. **Descending priority queue :** In descending priority queue, elements are inserted in any order but while deleting elements from the queue always a largest element to be deleted first.

Applications of priority queue :

1. The typical example of priority queue is scheduling the jobs in operating system. Typically operating system allocates priority to jobs. The jobs are placed in the queue and position 1 of the job in priority queue determines their priority.
2. In network communication, to manage limited bandwidth for transmission, the priority queue is used.
3. In simulation modelling, to manage the discrete events the priority queue is used.

VERY IMPORTANT QUESTIONS

Following questions are very important. These questions may be asked in your SESSIONALS as well as UNIVERSITY EXAMINATION.

Q. 1. Write a C function for array implementation of stack. Write all primitive operations.

Ans. Refer Q. 2.4.

Q. 2. Write a C function for linked list implementation of stack. Write all the primitive operations.

Ans. Refer Q. 2.5.

Q. 3. What is stack ? Implement stack with singly linked list.

Ans. Refer Q. 2.6.

Q. 4. Write a function in C language to reverse a string using stack.

Ans. Refer Q. 2.7.

Q. 5. Convert following infix expression into postfix expression A + (B * C + D)/E.

Ans. Refer Q. 2.10.

Q. 6. Consider the following infix expression and convert into reverse polish notation using stack. $A + (B * C - (D/E \wedge F) * H)$

Ans. Refer Q. 2.11.

Q. 7. Solve the following :

a. ((A – (B + C) * D) / (E + F)) [Infix to postfix]

b. (A + B) + *C – (D – E) $\wedge$ F [Infix to prefix]

c. 7 5 2 + * 4 1 5 – / – [Evaluate the given postfix expression]

Ans. Refer Q. 2.14.

Q. 8. What is recursion ? Write a recursive program to find sum of digits of the given number. Also, calculate the time complexity.

Ans. Refer Q. 2.17.

Q. 9. What is Tower of Hanoi problem ? Write the recursive code in C language for the problem.

Ans. Refer Q. 2.20.

Q. 10. Explain Tower of Hanoi problem and write a recursive algorithm to solve it.

Ans. Refer Q. 2.21.

Q. 11. Define the recursion. Write a recursive and non-recursive program to calculate the factorial of the given number.

Ans. Refer Q. 2.24.

Q. 12. Write the procedures for insertion, deletion and traversal of a queue.

Ans. Refer Q. 2.26.

Q. 13. What is circular queue ? Write a C code to insert an element in circular queue. Write all the condition for overflow.

Ans. Refer Q. 2.27.

Q. 14. Write a C program to implement the array representation of circular queue.

Ans. Refer Q. 2.29.

Q. 15. Explain how a circular queue can be implemented using arrays. Write all functions for circular queue operations.

Ans. Refer Q. 2.31.

☺☺☺

3
UNIT

Searching and Sorting

CONTENTS

PART-1

Searching : Concept of Searching, Sequential Search, Index Sequential Search, Binary Search.

Questions-Answers

Long Answer Type and Medium Answer Type Questions

Que 3.1. What do you mean by searching ? Explain.

Answer

1. Searching is the process of finding the location of given element in the linear array.
2. The search is said to be successful if the given element is found, *i.e.*, the element does exists in the array; otherwise unsuccessful.
3. There are two searching techniques :
   a. Linear search (sequential) b. Binary search
4. The algorithm which we choose depends on organization of the array elements.
5. If the elements are in random order, then we have to use linear search technique, and if the array elements are sorted, then it is preferable to use binary search.

Que 3.2. Write a short note on sequential search and index sequential search.

Answer

Sequential search :

1. In sequential (or linear) search, each element of an array is read one-by-one sequentially and it is compared with the desired element. A search will be unsuccessful if all the elements are read and the desired element is found.
2. Linear search is the least efficient search technique among other search techniques.
3. It is used when the records are stored without considering the order or when the storage medium lacks the direct access facility.
4. It is the simplest way for finding an element in a list.
5. It searches the elements sequentially in a list, no matter whether list is sorted or unsorted.
   a. In case of sorted list in ascending order, the search is started from 1st element and continued until desired element is found or the element whose value is greater than the value being searched.

b. In case of sorted list in descending order, the search is started from 1st element and continued until the desired element is found or the element whose value is smaller than the value being searched.

c. If the list is unsorted searching started from 1st location and continued until the element is found or the end of the list is reached.

Index sequential search :

1. In index sequential search, an index file is created, that contains some specific group or division of required record, once an index is obtained, then the partial searching of element is done which is located in a specified group.
2. In indexed sequential search, a sorted index is set aside in addition to the array.
3. Each element in the index points to a block of elements in the array or another expanded index.
4. First the index is searched that guides the search in the array.
5. Indexed sequential search does the indexing multiple times like creating the index of an index.
6. When the user makes a request for specific records it will find that index group first where that specific record is recorded.

Que 3.3. Write down algorithm for linear/sequential search technique. Give its analysis.

Answer

LINEAR(DATA, *N*, ITEM, LOC)

Here DATA is a linear array with N elements, and ITEM is a given item of information. This algorithm finds the location LOC of ITEM in DATA, or sets LOC := 0 if the search is unsuccessful.

1. [Insert ITEM at the end of DATA] Set DATA[N + 1] := ITEM
2. [Initialize counter] Set LOC := 1
3. [Search for ITEM]
   Repeat while DATA[LOC] $\neq$ ITEM
      Set LOC := LOC + 1
   [End of loop]
4. [Successful?] If LOC = N + 1, then : Set LOC := 0
5. Exit

Analysis of linear search :

Best case : Element occur at first position. Time complexity is O(1).

Worst case : Element occur at last position. Time complexity is O(n).

Que 3.4. Write down the algorithm of binary search technique. Write down the complexity of algorithm.

Answer

Binary search (*A*, *n*, item, loc)

Let A is an array of 'n' number of items, item is value to be searched.

1. Set : beg = 0, Set : end = $n - 1$, Set : mid = (beg + end) / 2
2. While ((beg $\leq$ end) and (a [mid] != item))
3. If (item < a[mid])
   then Set : end = mid – 1
   else
   Set : beg = mid + 1
   endif
4. Set : mid = (beg + end) / 2
   endwhile
5. If (beg > end) then
   Set : loc = – 1 // element not found
   else
   Set : loc = mid
   endif
6. Exit

Analysis of binary search :

The complexity of binary search is $O(\log_2 n)$.

Que 3.5. **What is difference between sequential (linear) search and binary search technique ?**

Answer

S. No.	Sequential (linear) search	Binary search
1.	No elementary condition *i.e.*, array can be sorted or unsorted.	Elementary condition *i.e.*, array should be sorted.
2.	It takes long time to search an element.	It takes less time to search an element.
3.	Complexity is $O(n)$.	Complexity is $O(\log_2 n)$.
4.	It searches data linearly.	It is based on divide and conquer method.

PART-2

Concept of Hashing and Collision Resolution Techniques used in Hashing.

Questions-Answers

Long Answer Type and Medium Answer Type Questions

Que 3.6. **What do you mean by hashing ?**

Answer

1. Hashing is a technique that is used to uniquely identify a specific object from a group of similar objects.
2. Hashing is the transformation of a string of characters into a usually shorter fixed-length value or key that represents the original string.
3. In hashing, large keys are converted into small keys by using hash functions.
4. The values are then stored in a data structure called hash table.
5. The task of hashing is to distribute entries (key/value pairs) uniformly across an array.
6. Each element is assigned a key (converted key). By using that key we can access the element in O(1) time.
7. Using the key, the algorithm (hash function) computes an index that suggests where an entry can be found or inserted.
8. Hashing is used to index and retrieve items in a database because it is faster to find the item using the shorter hashed key than to find it using the original value.
9. The element is stored in the hash table where it can be quickly retrieved using hashed key which is defined by

   Hash Key = Key Value % Number of Slots in the Table

Que 3.7. **Discuss types of hash functions.**

Answer

Types of hash functions :

a. Division method :

1. Choose a number m larger than the number n of key in K. (The number m is usually chosen to be a prime number or a number without small divisors, since this frequently minimizes the number of collisions.)
2. The hash function H is defined by :
   $$H(k) = k \pmod{m} \quad \text{or} \quad H(k) = k \pmod{m} + 1$$
3. Here $k \pmod{m}$ denotes the remainder when k is divided by m.
4. The second formula is used when we want the hash addresses to range from 1 to m rather than from 0 to $m - 1$.

b. Midsquare method :

1. The key k is squared.
2. The hash function H is defined by : $H(k) = l$
   where l is obtained by deleting digits from both end of k^2.
3. We emphasize that the same positions of k^2 must be used for all of the keys.

c. Folding method :

1. The key k is partitioned into a number of parts, $k_1, \ldots, k_r$, where each part, except possibly the last, has the same number of digits as the required address.

2. Then the parts are added together, ignoring the last carry *i.e.*,
$$H(k) = k_1 + k_2 + ... + k_r$$
where the leading-digit carries, if any, are ignored.
4. Now truncate the address upto the digit based on the size of hash table.

Que 3.8. **What is collision ? Discuss collision resolution techniques.**

OR

Write a short note on hashing techniques.

AKTU 2017-18, Marks 3.5

Answer

Collision :

1. Collision is a situation which occur when we want to add a new record R with key k to our file F, but the memory location address $H(k)$ is already occupied.
2. A collision occurs when more than one keys map to same hash value in the hash table.

Collision resolution technique :

Hashing with open addressing :

1. In open addressing, all elements are stored in the hash table itself.
2. While searching for an element, we systematically examine table slots until the desired element is found or it is clear that the element is not in the table.
3. Thus, in open addressing, the load factor λ can never exceed 1.
4. The process of examining the locations in the hash table is called probing.
5. Following are techniques of collision resolution by open addressing :
   - **a. Linear probing :**
     - i. Given an ordinary hash function h' : U [0, 1, ..., $m - 1$], the method of linear probing uses the hash function.
       $$h\,(k, i) = (h'\,(k) + i) \bmod m$$
       where 'm' is the size of the hash table and $h'\,(k) = k \bmod m$ (basic hash function).
   - **b. Quadratic probing :**
     - i. Suppose a record R with key k has the address $H(k) = h$ then instead of searching the locations with address h, $h + 1$, $h + 2$, ..., we linearly search the locations with addresses $h, h + 1, h + 4, h + 9, ..., h + i^2$.
     - ii. Quadratic probing uses a hash function of the form
       $$h\,(k, i) = (h'\,(k) + c_1 i + c_2 i^2) \bmod m$$
       where (as in linear probing) h' is an auxiliary hash function, c_1 and $c_2 \neq 0$ are auxiliary constants, and $i = 0, 1, ..., m - 1$.
   - **c. Double hashing :**
     - i. Double hashing is one of the best methods available for open addressing because the permutations produced have many of the characteristics of randomly chosen permutations.

ii. Double hashing uses a hash function of the form :

$$h(k, i) = (h_1(k) + ih_2(k)) \bmod m,$$

where h_1 and h_2 are auxiliary hash functions and m is the size of the hash table.

Hashing with separate chaining :

1. This method maintains the chain of elements which have same hash address.
2. We can take the hash table as an array of pointers.
3. Size of hash table can be number of records.
4. Here each pointer will point to one linked list and the elements which have same hash address will be maintained in the linked list.
5. We can maintain the linked list in sorted order and each elements of linked list will contain the whole record with key.
6. For inserting one element, first we have to get the hash value through hash function which will map in the hash table, then that element will be inserted in the linked list.
7. Searching a key is also same, first we will get the hash key value in hash table through hash function, then we will search the element in corresponding linked list.
8. Deletion of a key contains first search operation then same as delete operation of linked list.

Que 3.9. What do you mean by hashing and collision ? Discuss the advantages and disadvantages of hashing over other searching techniques.

AKTU 2014-15, Marks 10

Answer

Hashing : Refer Q. 3.6, Page 3–4A, Unit-3.

Collision : Refer Q. 3.8, Page 3–6A, Unit-3.

Advantages of hashing over other search techniques :

1. The main advantage of hash tables over other table data structures is speed. This advantage is more apparent when the number of entries is large (thousands or more).
2. Hash tables are particularly efficient when the maximum number of entries can be predicted in advance, so that the bucket array can be allocated once with the optimum size and never resized.
3. If the set of key-value pairs is fixed and known ahead of time (so insertions and deletions are not allowed), one may reduce the average lookup cost by a careful choice of the hash function, bucket table size, and internal data structures.

Disadvantages of hashing over other search techniques :

1. Hash tables can be more difficult to implement than self-balancing binary search trees. Choosing an effective hash function for a specific application is more an art than a science. In open-addressed hash tables it is fairly easy to create a poor hash function.

2. The cost of a good hash function can be significantly higher than the inner loop of the lookup algorithm for a sequential list or search tree.
3. Hash tables are not effective when the number of entries is very small. For certain string processing applications, such as spell-checking, hash tables may be less efficient than trees, finite automata, or arrays.
4. If each key is represented by a small enough number of bits, then, instead of a hash table, one may use the key directly as the index into an array of values.

Que 3.10. Write short notes on garbage collection.

AKTU 2017-18, Marks 3.5

AKTU 2014-15, Marks 05

Answer

1. When some memory space becomes reusable due to the deletion of a node from a list or due to deletion of entire list from a program then we want the space to be available for future use.
2. One method to do this is to immediately reinsert the space into the free-storage list. This is implemented in the linked list.
3. This method may be too time consuming for the operating system of a computer.
4. In another method, the operating system of a computer may periodically collect all the deleted space onto the free storage list. This type of technique is called garbage collection.
5. Garbage collection usually takes place in two steps. First the computer runs through all lists, tagging those cells which are currently in use and then the computer runs through the memory, collecting all untagged space onto the free storage list.
6. The garbage collection may take place when there is only some minimum amount of space or no space at all left in the free storage list or when the CPU is idle and has time to do the collection.

Que 3.11. Write the conditions when collision occurs in hashing. Describe any collision detection algorithm in brief.

Answer

Condition when collision occurs : Refer Q. 3.8, Page 3–6A, Unit-3.

Collision detection algorithm :

a. One of the collision detection algorithms is grid based algorithm.
b. In this algorithm, grids are space-filling.
c. Each cell or voxel (volume pixel) has a list of objects which intersects it.
d. The uniform grid is used to determine which objects are near to an object by examining object-lists of the cells the object overlaps.
e. Intersections for a given object are found by going through the object lists for all voxels containing the object, performing intersection tests against objects on those lists.

f. A grid based collision detection algorithm then works as follows :

1. for $i = 1$ to n
2. v_{min} = voxel (min (bbox (object (i))))
3. v_{max} = voxel (max (bbox (object (i))))
4. for $x = v_{min_x}$ to $x = v_{max_x}$
5. for $y = v_{min_y}$ to $y = v_{max_y}$
6. for $z = v_{min_z}$ to $z = v_{max_z}$
7. for $j = 1$ to n_objects (voxel (x, y, z))
8. if (not tested (object (i), object (j)))
9. intersect (object (i), object (j))

PART-3

Sorting : Insertion Sort, Selection Sort, Bubble Sort.

Questions-Answers

Long Answer Type and Medium Answer Type Questions

Que 3.12. Write a short note on insertion sort.

AKTU 2014-15, Marks 05

Answer

1. In insertion sort, we pick up a particular value and then insert it at the appropriate place in the sorted sublist, *i.e.*, during k^{th} iteration the element $a[k]$ is inserted in its proper place in the sorted sub-array $a[1], a[2], a[3]$ .... $a[k-1]$.
2. This task is accomplished by comparing $a[k]$ with $a[k-1]$, $a[k-2]$, $a[k-3]$ and so on until the first element $a[j]$ such that $a[j] \le a[k]$ is found.
3. Then each of the elements $a[k-1], a[k-2], a[j+1]$ are moved one position up and then element $a[k]$ is inserted in $[j+1]^{st}$ position in the array.

Insertion-Sort (A)

1. for $j \leftarrow 2$ to length$[A]$
2. do key $\leftarrow A[j]$ /*Insert $A[j]$ into the sorted sequence $A[1....j-1]$.*/
3. $i \leftarrow j - 1$
4. while $i > 0$ and $A[i] >$ key
5. do $A[i + 1] \leftarrow A[i]$
6. $i \leftarrow i - 1$
7. $A[i + 1] \leftarrow$ key

Analysis of insertion sort :

Complexity of best case is $O(n)$

Complexity of average case is $O(n^2)$

Complexity of worst case is $O(n^2)$

Que 3.13. **Write a short note on selection sort.**

Answer

1. In selection sort we repeatedly find the next largest (or smallest) element in the array and move it to its final position in the sorted array.
2. We begin by selecting the largest element and moving it to the highest index position.
3. We can do this by swapping the element at the highest index and the largest element.
4. We then reduce the effective size of the array by one element and repeat the process on the smaller sub-array.
5. The process stops when the effective size of the array becomes 1 (an array of 1 element is already sorted).

Selection-Sort (*A*) :

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

Analysis of selection sort :

Complexity of best case is $O(n^2)$.
Complexity of average case is $O(n^2)$.
Complexity of worst case is $O(n^2)$.

Que 3.14. **Discuss bubble sort.**

Answer

1. Bubble sort is the simplest sorting algorithm that works by repeatedly swapping the adjacent element if they are in wrong order.
2. Bubble sort procedure is based on following idea :
   a. Suppose if the array contains n elements, then $(n - 1)$ iterations are required to sort this array.
   b. The set of items in the array are scanned again and again and if any two adjacent items are found to be out of order, they are reversed.
   c. At the end of the first iteration, the lowest value is placed in the first position.
   d. At the end of the second iteration, the next lowest value is placed in the second position and so on.
3. It is very efficient in large sorting jobs. For n data items, this method requires $n(n - 1)/2$ comparisons.

Bubble-sort (A) :

1. for $i \leftarrow 1$ to length [A]
2. for $j \leftarrow$ length[A] down to $i + 1$

3. if $A[j] < A[j-1]$
4. exchange($A[j]$, $A[j-1]$)

Analysis of bubble sort :

Complexity of best case is $O(n)$.
Complexity of worst case is $O(n^2)$.
Complexity of average case is $O(n^2)$.

PART-4

Quick Sort.

Questions-Answers

Long Answer Type and Medium Answer Type Questions

Que 3.15. Explain and give quick sort algorithm. Determine its complexity.

Answer

1. Quick sort is a sorting algorithm that also uses the idea of divide and conquer.
2. This algorithm finds the elements, called pivot, that partitions the array into two halves in such a way that the elements in the left sub-array are less than and the elements in the right sub-array are greater than the partitioning element.
3. Then these two sub-arrays are sorted separately. This procedure is recursive in nature with the base criteria.

Algorithm :

QUICK (*A*, *N*, BEG, END, LOC) :

1. [Initialize] Set LEFT := BEG, RIGHT := END and LOC := BEG
2. [Scan from RIGHT to LEFT]
   a. Repeat while A [LOC] $\leq$ A [RIGHT] and LOC $\neq$ RIGHT
      RIGHT := RIGHT – 1
      [End of Loop]
   b. If LOC = RIGHT, then : Return
   c. If A [LOC] > A [RIGHT], then :
      i. [Interchange A [LOC] and A [RIGHT]]
         TEMP := A [LOC], A [LOC] := A [RIGHT],
         A [RIGHT] = TEMP
      ii. Set LOC := RIGHT
      iii. Go to step 3
         [End of if structure]
3. [Scan from LEFT to RIGHT]
   a. Repeat while A [LEFT] $\leq$ A [LOC] and LEFT $\neq$ LOC :

LEFT := LEFT + 1
[End of Loop]

b. If LOC = LEFT, then : Return,

c. If A [LEFT] > A [LOC], then

i. [Interchange A [LEFT] and A [LOC]]
TEMP := A [LOC], A [LOC] := A [LEFT],
A [LEFT] := TEMP

ii. Set LOC := LEFT

iii. Go to step 2

[End of if structure]

Quick sort : This algorithm sorts an array *A* with *N* elements.

1. [Initialize] Top := NULL
2. [PUSH boundary values of A onto stack when A has 2 or more elements]
   If $N > 1$, then : TOP := TOP + 1, LOWER [1] := 1, UPPER [1] := N
3. Repeat steps 4 to 7 while TOP ≠ NULL
4. [POP sublist from stacks]
   Set BEG := LOWER [TOP], END := UPPER [TOP],
   TOP = TOP – 1
5. Call Quick (A, N, BEG, END, LOC)
6. [PUSH left sublist onto stack when it has 2 or more elements]
   If BEG < LOC – 1 then :
   TOP := TOP + 1, LOWER [TOP] := BEG,
   UPPER [TOP] = LOC – 1
   [End of if structure]
7. [PUSH right sublist onto stack when it has 2 or more elements]
   If LOC + 1 < END, then :
   TOP := TOP + 1, LOWER [TOP] := LOC + 1
   UPPER [TOP] := END
   [END of if structure]
   [END of step 3 loop]
8. Exit

Analysis of quick sort :

Complexity of worst case is $O(n^2)$.

Complexity of best case is $O(n \log n)$.

Complexity of average case is $O(n \log n)$.

Que 3.16. **Write a recursive quick sort algorithm.**

Answer

QUICK-SORT (*A*, *p*, *r*) :

1. If $p < r$ then
2. $q \leftarrow$ PARTITION (A, p, r)
3. QUICK-SORT $(A, p, q - 1)$
4. QUICK-SORT $(A, q + 1, r)$

PARTITION (*A*, *p*, *r*) :

1. $x \leftarrow A[r]$
2. $i \leftarrow p - 1$

```
for j → p to r – 1
    do if A[j] ≤ x
        then i → i + 1
            exchange A[i] ↔ A[j]
exchange A[i + 1] ↔ A[r]
return i + 1
```

Que 3.17. What is quick sort ? Sort the given values using quick sort; present all steps/iterations :

38, 81, 22, 48, 13, 69, 93, 14, 45, 58, 79, 72 **AKTU 2016-17, Marks 10**

Answer

Quick sort : Refer Q. 3.15, Page 3–11A, Unit-3.

Numerical : A = 38, 81, 22, 48, 13, 69, 93, 14, 45, 58, 79, 72. Choose the pivot element to be the element in position (left + right)/2.

During the partitioning process,

1. Elements strictly to the left of position *lo* are less than or equivalent to the pivot element (69).
2. Elements strictly to the right of position *hi* are greater than the pivot element. When *lo* and *hi* cross, we are done. The final value of *hi* is the position in which the partitioning element ends up.

   Swap pivot element with leftmost element *lo* = left + 1; *hi* = right;

left	left+1										right
38	81	22	48	13	69	93	14	45	58	79	72

Move *hi* left and *lo* right as far as we can; then swap $A[lo]$ and $A[hi]$, and move *hi* and *lo* one more position.

lo *hi←hi←hi*

69	81*	22	48	13	38	93	14	45	58*	79*	72*

Repeat above

lo → *lo* → *lo* → *lo* → *lo* *hi*

69	58	22*	48*	13*	38*	93*	14	45*	81	79	72

Repeat above until *hi* and *lo* cross; then *hi* is the final position of the pivot element, so swap $A[hi]$ and $A[left]$.

hi
lo→*lo*

69	58	22	48	13	38	45	14**	93*	81	79	72

Partitioning complete; return value of *hi*.

							hi				
14	58	22	48	13	38	45	69	93	81	79	72

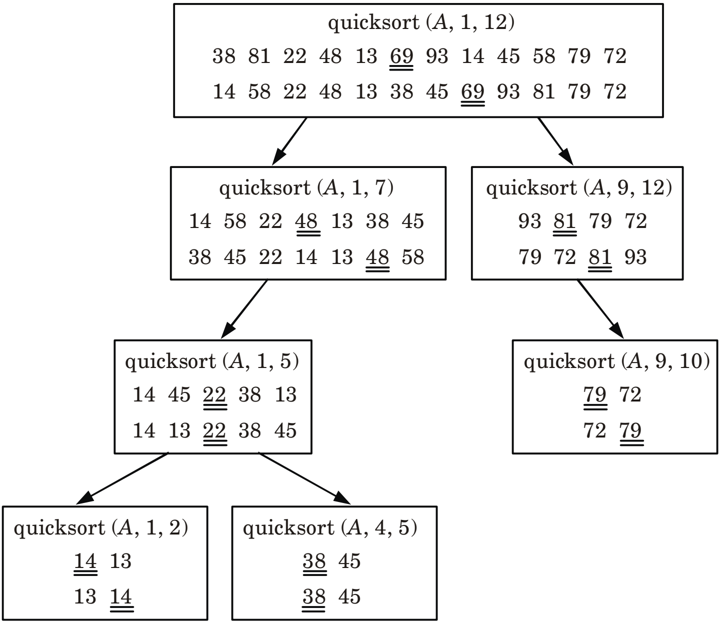


Fig. 3.17.1.

Que 3.18. **Write algorithm for quick sort. Trace your algorithm on the following data to sort the list: 2, 13, 4, 21, 7, 56, 51, 85, 59, 1, 9, 10. How the choice of pivot elements affects the efficiency of algorithm.**

AKTU 2018-19, Marks 07

Answer

Quick sort algorithm : Refer Q. 3.16, Page 3–12A, Unit-3.

Numerical :

1	2	3	4	5	6	7	8	9	10	11	12
2	13	4	21	7	56	51	85	59	1	9	10

Here $p = 1, r = 12$

$$x = A[12] = 10$$
$$i = p - 1 = 0$$
$$j = 1 + 0 = 1$$

Now, $\quad j = 1$ and $i = 0$

$A[1] = 2 \leq 10$ (True)

then $i = 0 + 1 = 1$ and $A[1] \leftrightarrow A[1]$

Now, $j = 2$ and $i = 1$

$A[2] = 13$ and $13 \nleq 10$ (False)

So, $j = 3 \quad i = 1$

$A[3] = 4$ and $4 \leq 10$ (True)

then, $i = 1 + 1 = 2$ and $A[2] \leftrightarrow A[3]$

i.e.,

1	2	3	4	5	6	7	8	9	10	11	12
2	4	13	21	7	56	51	85	59	1	9	10

Now, $j = 4$ and $i = 2$

$A[4] = 21$ and $21 \nleq 10$ (False)

$j = 5$ and $i = 2$

$A[5] = 7 \leq 10$ (True)

then, $i = 2 + 1 = 3$ and $A[3] \leftrightarrow A[5]$

i.e.,

1	2	3	4	5	6	7	8	9	10	11	12
2	4	7	21	13	56	51	85	59	1	9	10

Now, $j = 6$ and $i = 3$

$A[6] = 56$ and $56 \nleq 10$

So, $j = 7$ and $i = 3$

$A[7] = 51$ and $51 \nleq 10$

$j = 8$ and $i = 3$

$A[8] = 85$ and $85 \nleq 10$

$j = 9$ and $i = 3$

$A[9] = 59$ and $59 \nleq 10$

$j = 10$ and $i = 3$

$A[10] = 1 \leq 10$ (True)

then, $i = 3 + 1 = 4$ and $A[4] \leftrightarrow A[10]$

i.e.,

1	2	3	4	5	6	7	8	9	10	11	12
2	4	7	1	13	56	51	85	59	21	9	10

$j = 11$ and $i = 4$

$A[11] = 9 \leq 10$ (True)

$i = 4 + 1 = 5$ and $A[5] \leftrightarrow A[11]$

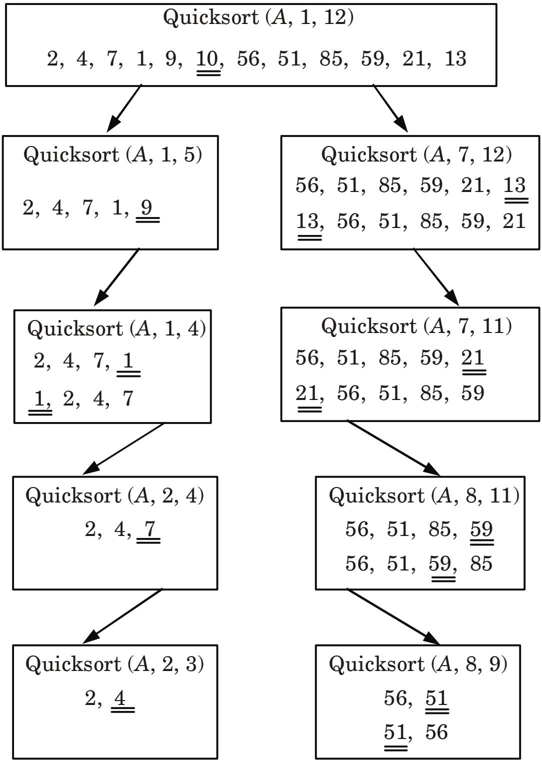
i.e.,

1	2	3	4	5	6	7	8	9	10	11	12
2	4	7	1	9	56	51	85	59	21	13	10

$A[6] \leftrightarrow A[12]$

Partitioning complete, return value of *q* :

1	2	3	4	5	6	7	8	9	10	11	12
2	4	7	1	9	10	56	51	85	59	21	13



Choice of pivot element affects the efficiency of algorithm : If we choose the last or first element of an array as pivot element then it results in worst case scenario with $O(n^2)$ time complexity. If we choose the median as pivot element then it divides the array into two halves every time and results in best or average case scenario with time complexity $O(n \log n)$. Thus, the efficiency of quick sort algorithm depends on the choice of pivot element.

Que 3.19. **Use quick sort algorithm to sort 15, 22, 30, 10, 15, 64, 1, 3, 9, 2. Is it a stable sorting algorithm? Justify.**

AKTU 2017-18, Marks 07

Answer

Let A [ ] =

1	2	3	4	5	6	7	8	9	10
15	22	30	10	15	64	1	3	9	2

Here $p = 1$, $r = 10$

$$x = A[10] \text{ i.e., } x = 2$$
$$i = p - 1 \text{ i.e., } i = 0$$
$$j = 1 \text{ to } 9$$

Now, $j = 1$ and $i = 0$

$$A[j] = A[1] = 15 \text{ and } 15 \le 2$$

So, $j = 2$ and $i = 0$

$$A[2] = 22 \le 2 \text{ (False)}$$

Now, $j = 3$ and $i = 0$

$$A[3] = 30 \le 2 \text{ (False)}$$
$$j = 4 \text{ and } i = 0$$
$$A[4] = 10 \le 2 \text{ (False)}$$
$$j = 5$$
$$A[5] = 15 \le 2 \text{ (False)}$$
$$j = 6$$
$$A[6] = 64 \le 2 \text{ (False)}$$
$$j = 7$$
$$A[7] = 1 \le 2 \text{ (True)}$$
$$i = 0 + 1 = 1$$
$$A[1] \leftrightarrow A[7]$$

i.e.,

1	2	3	4	5	6	7	8	9	10
1	22	30	10	15	64	15	3	9	2

$$j = 8 \text{ and } i = 1$$
$$A[8] = 3 \le 2 \text{ (False)}$$
$$j = 9 \text{ and } i = 1$$
$$A[9] = 9 \le 2 \text{ (False)}$$

then, $A[1 + 1] \leftrightarrow A[r]$

$$A[2] \leftrightarrow A[10]$$
$$q \leftarrow 2$$

i.e.,

1	2	3	4	5	6	7	8	9	10
1	2	30	10	15	64	15	3	9	22

QUICK SORT (A, 1, 1)

1	2
1	2

QUICK SORT (A, 3, 10)

3	4	5	6	7	8	9	10
30	10	15	64	15	3	9	22

Here $p = 3, r = 10$

$$x = A[10] = 22$$
$$i = 3 - 1 = 2$$
$$j = 3 \text{ to } 9; \quad j = 3 \text{ and } i = 2$$
$$A[3] = 30 \le 22 \text{ (False)}$$
$$j = 4 \text{ and } i = 2$$
$$A[4] = 10 \le 22 \text{ (True)}$$
$$i = 2 + 1 = 3 \text{ and } A[3] \leftrightarrow A[4]$$

i.e.,

3	4	5	6	7	8	9	10
10	30	15	64	15	3	9	22

$$j = 5 \text{ and } i = 3$$
$$A[5] = 15 \leq 22 \text{ (True)}$$
$$i = 3 + 1 = 4 \text{ and } A[4] \leftrightarrow A[5]$$

3	4	5	6	7	8	9	10
10	15	30	64	15	3	9	22

Similarly

$$j = 7, i = 4$$
$$A[7] = 15 \leq 22 \text{ (True)}$$
$$i = 4 + 1 = 5 \text{ and } A[5] \leftrightarrow A[7]$$

i.e.,

3	4	5	6	7	8	9	10
10	15	15	64	15	3	9	22

Similarly, we get another pivot element

10	15	15	3	9	22	64	30

Thus, this is a stable algorithm.

PART-5

Merge Sort.

Questions-Answers

Long Answer Type and Medium Answer Type Questions

Que 3.20. Describe two way merge sort method. Explain the complexities of merge sort method.

Answer

Merge sort :

a. Merge sort is a sorting algorithm that uses the idea of divide and conquer.

b. This algorithm divides the array into two halves, sorts them separately and then merges them.

c. This procedure is recursive, with the base criteria that the number of elements in the array is not more than 1.

MERGE_SORT (*a*, *p*, *r*) :

1. if $p < r$
2. then $q \leftarrow \lfloor (p + r)/2 \rfloor$
3. MERGE-SORT (A, p, q)
4. MERGE-SORT $(A, q + 1, r)$

5. MERGE (A, p, q, r)

MERGE (A, p, q, r) :

1. $n_1 = q - p + 1$
2. $n_2 = r - q$
3. Create arrays L [1 ....n_1 + 1] and
   R [1......n_2 + 1]
4. for i = 1 to n_1
   do
   $L[i] = A\ [p + i - 1]$
   endfor
5. for j = 1 to n_2
   do
   $R[j] = A[q + j]$
   endfor
6. $L[n_1 + 1] = \infty, R[n_2 + 1] = \infty$
7. $i = 1, j = 1$
8. for $k = p$ to r
   do
   if $L[i] \le R[j]$
   then $A[k] \leftarrow L[i]$
   $i = i + 1$
   else $A[k] = R[j]$
   $j = j + 1$
   endif
   endfor
9. exit

Complexity of merge sort algorithm :

1. Let $f(n)$ denote the number of comparisons needed to sort an n-element array A using the merge sort algorithm.
2. The algorithm requires at most log n passes.
3. Moreover, each pass merges a total of n elements, and by the discussion on the complexity of merging, each pass will require at most n comparisons.
4. Accordingly, for both the worst case and average case,
$$f(n) \le n \log n$$
5. This algorithm has the same order as heap sort and the same average order as quick sort.
6. The main drawback of merge sort is that it requires an auxiliary array with n elements.
7. Each of the other sorting algorithms requires only a finite number of extra locations, which is independent of n.
8. The results are summarized in the following table :

Algorithm	Worst case	Average case	Extra memory
Merge sort	$n \log n = O(n \log n)$	$n \log n = O(n \log n)$	$O(n)$

Que 3.21. **Write an algorithm for merge sorting. Using the algorithm sort in ascending order : 10, 25, 16, 5, 35, 48, 8**

AKTU 2014-15, Marks 10

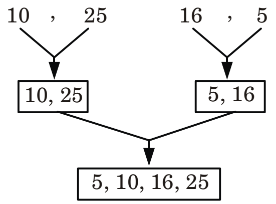
Answer

Merge sorting : Refer Q. 3.20, Page 3–18A, Unit-3.

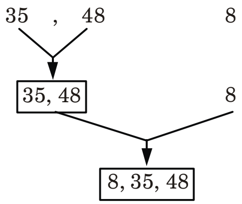
Numerical :

10, 25, 16, 5, 35, 48, 8

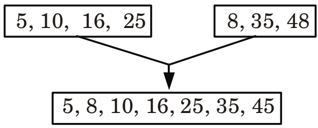
1. Divide first half | 10, 25, 16, 5 | | 35, 48, 8 |
2. **Consider the first half :** 10, 25, 16, 5 again divide into two sub- arrays



3. **Consider the second half :** 35, 48, 5 again divide into two sub-arrays



4. Merge these two sorted sub-arrays,



This is the sorted array.

Que 3.22. **How do you calculate the complexity of sorting algorithms ? Also, write a recursive function in 'C' to implement the merge sort on given set of integers.** **AKTU 2015-16, Marks 10**

Answer

Complexity : Refer Q. 3.20, Page 3–18A, Unit-3.

Function :

```
void merge (int low, int mid, int high)
{
int temp [MAX] ;
int i = low;
int j = mid + 1;
int k = low;
while ((i <= mid) && (j <= high))
{
if (array [i] <= array [j ] )
	temp [k++] = array [i++];
else
	temp [k++] = array [j++] ;
}
	while (i <= mid)
temp [k++] = array [i++];
while (j <= high)
temp [k++] = array [j++] ;
for (i = low; i <= high; i++)
array [i] = temp [i];
}
void merge_sort (int low, int high)
{
int mid;
if (low != high)
{
mid = (low + high) / 2;
merge_sort (low, mid);
merge_sort (mid + 1, high);
merge (low, mid, high);
}
}
```

PART-6

Heap Sort and Radix Sort.

Questions-Answers

Long Answer Type and Medium Answer Type Questions

Que 3.23. **Write a short note on heap sort.**

AKTU 2014-15, Marks 05

OR

Explain heap sort. **AKTU 2017-18, Marks 3.5**

Answer

1. Heap sort finds the largest element and puts it at the end of array, then the second largest item is found and this process is repeated for all other elements.
2. The general approach of heap sort is as follows :
   a. From the given array, build the initial max heap.
   b. Interchange the root (maximum) element with the last element.
   c. Use repetitive downward operation from root node to rebuild the heap of size one less than the starting.
   d. Repeat step (a) and (b) until there are no more elements.

Analysis of heap sort :

Complexity of heap sort for all cases is $O(n \log_2 n)$.

MAX-HEAPIFY (A, i) :

1. $i \leftarrow$ left $[i]$
2. $r \leftarrow$ right $[i]$
3. if $l \leq$ heap-size $[A]$ and $A[l] > A[i]$
4. then largest $\leftarrow l$
5. else largest $\leftarrow i$
6. if $r \leq$ heap-size $[A]$ and $A[r] > A$ [largest]
7. then largest $\leftarrow r$
8. if largest $\neq i$
9. then exchange $A[i] \leftrightarrow A$[largest]
10. MAX-HEAPIFY [A, largest]

HEAP-SORT(A) :

1. BUILD-MAX-HEAP (A)
2. for $i \leftarrow$ length $[A]$ down to 2
3.   do exchange $A[1] \leftrightarrow A[i]$
4.   heap-size $[A] \leftarrow$ heap-size $[A] - 1$
5.   MAX-HEAPIFY (A, 1)

Que 3.24. **Write a short note on radix sort.**

OR

Explain radix sort. **AKTU 2017-18, Marks 3.5**

Answer

1. Radix sort is a small method that many people uses when alphabetizing a large list of names (here Radix is 26, 26 letters of alphabet).
2. Specifically, the list of name is first sorted according to the first letter of each name, *i.e.*, the names are arranged in 26 classes.
3. Intuitively, one might want to sort numbers on their most significant digit.

4. But radix sort do counter-intuitively by sorting on the least significant digits first.
5. On the first pass entire numbers sort on the least significant digit and combine in an array.
6. Then on the second pass, the entire numbers are sorted again on the second least-significant digits and combine in an array and so on.
7. Following example shows how radix sort operates on seven 3-digit number.

Table 3.24.1.

Input	1st pass	2nd pass	3rd pass
329	720	720	329
457	355	329	355
657	436	436	436
839	457	839	457
436	657	355	657
720	329	457	720
355	839	657	839

8. In the above example, the first column is the input.
9. The remaining shows the list after successive sorts on increasingly significant digits position.
10. The code for radix sort assumes that each element in the n-element array A has d digits, where digit 1 is the lowest-order digit and d is the highest-order digit.

RADIX_SORT (A, d)

for $i \leftarrow 1$ to d do
use a stable sort to sort array A on digit i
// counting sort will do the job

Analysis :

1. The running time depends on the table used as an intermediate sorting algorithm.
2. When each digit is in the range 1 to k, and k is not too large, COUNTING_SORT is the obvious choice.
3. In case of counting sort, each pass over n d-digit numbers takes $\Theta(n + k)$ time.
4. There are d passes, so the total time for radix sort is $\Theta(n + k)$ time. There are d passes, so the total time for radix sort is $\Theta(dn + kd)$. When d is constant and $k = \Theta(n)$, the radix sort runs in linear time.

VERY IMPORTANT QUESTIONS

Following questions are very important. These questions may be asked in your SESSIONALS as well as UNIVERSITY EXAMINATION.

Q. 1. What is collision ? Discuss collision resolution techniques.

Ans. Refer Q. 3.8.

Q. 2. What do you mean by hashing and collision ? Discuss the advantages and disadvantages of hashing over other searching techniques.

Ans. Refer Q. 3.9.

Q. 3. Write short notes on garbage collection.

Ans. Refer Q. 3.10.

Q. 4. Write a short note on insertion sort.

Ans. Refer Q. 3.12.

Q. 5. What is quick sort ? Sort the given values using quick sort; present all steps/iterations :
38, 81, 22, 48, 13, 69, 93, 14, 45, 58, 79, 72

Ans. Refer Q. 3.17.

Q. 6. Write algorithm for quick sort. Trace your algorithm on the following data to sort the list: 2, 13, 4, 21, 7, 56, 51, 85, 59, 1, 9, 10. How the choice of pivot elements affects the efficiency of algorithm.

Ans. Refer Q. 3.18.

Q. 7. Use quick sort algorithm to sort 15, 22, 30, 10, 15, 64, 1, 3, 9, 2. Is it a stable sorting algorithm? Justify.

Ans. Refer Q. 3.19.

Q. 8. Write an algorithm for merge sorting. Using the algorithm sort in ascending order : 10, 25, 16, 5, 35, 48, 8

Ans. Refer Q. 3.21.

Q. 9. How do you calculate the complexity of sorting algorithms ? Also, write a recursive function in 'C' to implement the merge sort on given set of integers.

Ans. Refer Q. 3.22.

Q. 10. Write a short note on heap sort.

Ans. Refer Q. 3.23.

Q. 11. Write a short note on radix sort.

Ans. Refer Q. 3.24.

☺☺☺

4 UNIT

Graphs

CONTENTS

PART-1

Graphs : Terminology used with Graph.

Questions-Answers

Long Answer Type and Medium Answer Type Questions

Que 4.1. **What is a graph ? Describe various types of graph. Briefly explain few applications of graph.**

Answer

1. A graph is a non-linear data structure consisting of nodes and edges.
2. A graph is a finite sets of vertices (or nodes) and set of edges which connect a pair of nodes.

Types of graph :

1. **Undirected graph :**
   a. If the pair of vertices is unordered then graph G is called an undirected graph.
   b. That means if there is an edge between v_1 and v_2 then it can be represented as (v_1, v_2) or (v_2, v_1) also. It is shown in Fig. 4.1.1.

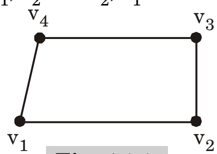


Fig. 4.1.1.

2. **Directed graph :**
   a. If the pair of vertices is ordered then graph G is called directed graph.
   b. That is, a directed graph or digraph is a graph which has ordered pair of vertices (v_1, v_2) where v_1 is the tail and v_2 is the head of the edge.
   c. If the graph is directed then the line segments of arcs have arrow heads indicating the direction. It is shown in Fig. 4.1.2.

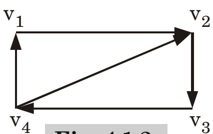


Fig. 4.1.2.

3. **Weighted graph :** A graph is said to be a weighted graph if all the edges in it are labelled with some numbers. It is shown in the Fig. 4.1.3.

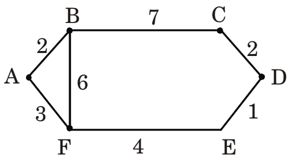


Fig. 4.1.3.

4. **Simple graph :** A graph or directed graph which does not have any self-loop or parallel edges is called a simple graph.
5. **Multi-graph :** A graph which has either a self-loop or parallel edges or both is called a multi-graph.
6. **Complete graph :**
   a. A graph is complete graph if each vertex is adjacent to every other vertex in graph or there is an edge between any pair of nodes in the graph.
   b. An undirected complete graph will contain $n(n-1)/2$ edges.
7. **Regular graph :**
   a. A graph is regular if every node is adjacent to the same number of nodes.
   b. Here every node is adjacent to 3 nodes.

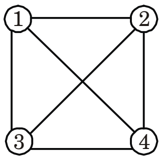


Fig. 4.1.4.

8. **Planar graph :** A graph is planar if it can be drawn in a plane without any two intersecting edges.
9. **Connected graph :**
   a. In a graph G, two vertices v_1 and v_2 are said to be connected if there is path in G from v_1 to v_2 or v_2 to v_1.
   b. Connected graph can be of two types :
      i. Strongly connected graph
      ii. Weakly connected graph

(a) Connected graph (b) Not connected graph

Fig. 4.1.5.

10. **Acyclic graph :** If a graph (digraph) does not have any cycle then it is called as acyclic graph.
11. **Cyclic graph :** A graph that has cycles is called a cyclic graph.
12. **Biconnected graph :** A graph with no articulation points is called a biconnected graph.

Applications of graph :

1. Graph is a non-linear data structure and is used to present various operations and algorithms.
2. Graphs are used for topological sorting.
3. Graphs are used to find shortest paths.
4. They are required to minimize some aspect of the graph, such as distance among all the vertices in the graph.

Que 4.2. What is graph ? Discuss various terminologies used in graph.

Answer

Graph : Refer Q. 4.1, Page 4–2A, Unit-4.

Various terminologies used in graphs are :

1. **Self loop :** If there is an edge whose starting and end vertices are same that is (v_2, v_2) is an edge then it is called a self loop or simply a loop.
2. **Parallel edges :** A pair of edges e and e' of G are said to be parallel iff they are incident on precisely the same vertices.
3. **Adjacent vertices :** A vertex u is adjacent to (or the neighbour of) other vertex v if there is an edge from u to v.
4. **Incidence :** In an undirected graph the edge (u, v) is incident on vertices u and v. In a digraph the edge (u, v) is incident from node u and is incident to node v.
5. **Degree of vertex :** The degree of a vertex is the number of edges incident to that vertex. In an undirected graph, the number of edges connected to a node is called the degree of that node.

PART-2

Data Structure for Graph Representations : Adjacency Matrices, Adjacency List, Adjacency.

Questions-Answers

Long Answer Type and Medium Answer Type Questions

Que 4.3. Discuss the various types of representation of graph.

Answer

Two types of graph representation are as follows :

1. **Matrix representation :** Matrices are commonly used to represent graphs for computer processing. Advantages of representing the graph in matrix lies on the fact that many results of matrix algebra can be readily applied to study the structural properties of graph from an algebraic point of view.

   a. **Adjacency matrix :**

      i. **Representation of undirected graph :** The adjacency matrix of a graph G with n vertices and no parallel edges is a $n \times n$ matrix A = $[a_{ij}]$ whose elements are given by

         a_{ij} = 1, if there is an edge between i^{th} and j^{th} vertices
         = 0, if there is no edge between them

      ii. **Representation of directed graph :** The adjacency matrix of a digraph D, with n vertices is the matrix

         $A = [a_{ij}]_{n \times n}$ in which
         $a_{ij} = 1$ if arc (v_i, v_j) is in D
         $= 0$ otherwise

         For example : Representation of following undirected and directed graph is :

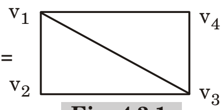
A = 

Fig. 4.3.1.

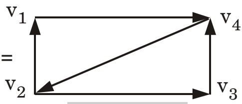
B = 

Fig. 4.3.2.

$$A = \begin{array}{c} \\ v_1 \\ v_2 \\ v_3 \\ v_4 \end{array} \begin{array}{c} \begin{array}{cccc} v_1 & v_2 & v_3 & v_4 \end{array} \\ \begin{bmatrix} 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 \end{bmatrix} \end{array} \qquad B = \begin{array}{c} \\ v_1 \\ v_2 \\ v_3 \\ v_4 \end{array} \begin{array}{c} \begin{array}{cccc} v_1 & v_2 & v_3 & v_4 \end{array} \\ \begin{bmatrix} 0 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 \end{bmatrix} \end{array}$$

   b. **Incidence matrix**

      i. **Representation of undirected graph :** Consider a undirected graph $G = (V, E)$ which has n vertices and m edges all labelled. The incidence matrix $I(G) = [b_{ij}]$, is then $n \times m$ matrix, where

         $b_{ij} = 1$ when edge e_j is incident with v_i
         $= 0$ otherwise

      ii. **Representation of directed graph :** The incidence matrix $I(D) = [b_{ij}]$ of digraph D with n vertices and m edges is the $n \times m$ matrix in which.

         $b_{ij} = 1$ if arc j is directed away from vertex v_i
         $= -1$ if arc j is directed towards vertex v_i
         $= 0$ otherwise.

         Find the incidence matrix to represent the graph shown in Fig. 4.3.3.

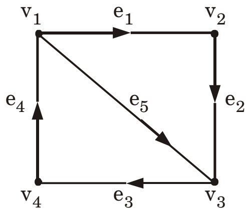


Fig. 4.3.3.

The incidence matrix of the digraph of Fig. 4.3.3 is

$$I(D) = \begin{bmatrix} 1 & 0 & 0 & -1 & 1 \\ -1 & 1 & 0 & 0 & 0 \\ 0 & -1 & 1 & 0 & -1 \\ 0 & 0 & -1 & 1 & 0 \end{bmatrix}$$

2. Linked representation :

i. In linked representation, the two nodes structures are used :

a. For non-weighted graph :

INFO	Adj-list

b. For weighted graph :

Weight	INFO	Adj-list

where Adj-list is the adjacency list *i.e.*, the list of vertices which are adjacent for the corresponding node.

ii. The header nodes in each list maintain a list of all adjacent vertices of that node for which the header node is meant.

iii. Suppose a directed graph

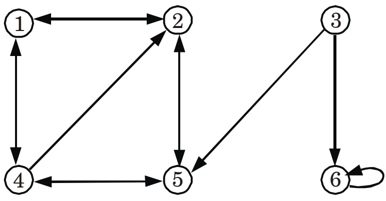


Fig. 4.3.4.

iv. The adjacency list representation of this graph.

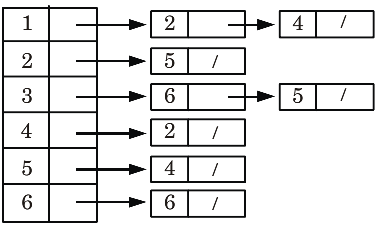


Fig. 4.3.5.

Que 4.4. **Explain adjacency multilists.**

Answer

1. Adjacency multilist representation maintains the lists as multilists, that is, lists in which nodes are shared among several lists.
2. For each edge there will be exactly one node, but this node will be in two lists *i.e.*, the adjacency lists for each of the two nodes, it is incident to. The node structure now becomes :

Mark	vertex 1	vertex 2	path 1	path 2

where mark is a one bit mark field that may be used whether or not the edge has been examined. The declarations in C are :

```
#define n 20
typedef struct edge {BOOLEAN mark;
    int vertex1, vertex2;
    NEXTEDGE path1, path2;
    }*NEXTEDGE;
    NEXTEDGE headnode [n];
```

PART-3

Graph Traversal : Depth First Search and Breadth First Search, Connected Component.

Questions-Answers

Long Answer Type and Medium Answer Type Questions

Que 4.5. **Write a short note on graph traversal.**

Answer

Traversing a graph :

i. Graph is represented by its nodes and edges, so traversal of each node is the traversing in graph.
ii. There are two standard ways of traversing a graph.
iii. One way is called a breadth first search, and the other is called a depth first search.
iv. During the execution of our algorithms, each node N of G will be in one of three states, called the status of N, as follows :

Status = 1 $\Rightarrow$ (Ready state). The initial state of the node N.
Status = 2 $\Rightarrow$ (Waiting state). The node N is on the queue or stack, waiting to be processed.
Status = 3 $\Rightarrow$ (Processed state). The node N has been processed.

1. **Breadth First Search (BFS) :** The general idea behind a breadth first search beginning at a starting node A is as follows :
   a. First we examine the starting node A.
   b. Then, we examine all the neighbours of A, and so on.
   c. We need to keep track of the neighbours of a node, and that no node is processed more than once.
   d. This is accomplished by using a queue to hold nodes that are waiting to be processed, and by using a field STATUS which tells us the current status of any node.

Algorithm : This algorithm executes a breadth first search on a graph G beginning at a starting node A.

i. Initialize all nodes to ready state (STATUS=1).
ii. Put the starting node A in queue and change its status to the waiting state (STATUS = 2).
iii. Repeat steps (iv) and (v) until queue is empty.
iv. Remove the front node N of queue. Process N and change the status of N to the processed state (STATUS = 3).
v. Add to the rear of queue all the neighbours of N that are in the ready state (STATUS=1) and change their status to the waiting state (STATUS = 2).
   [End of loop]
vi. End.

2. **Depth First Search (DFS) :** The general idea behind a depth first search beginning at a starting node A is as follows :
   a. First, we examine the starting node A.
   b. Then, we examine each node N along a path P which begins at A; that is, we process neighbour of A, then a neighbour of neighbour of A, and so on.
   c. This algorithm uses a stack instead of queue.

Algorithm :

i. Initialize all nodes to ready state (STATUS = 1).
ii. Push the starting node A onto stack and change its status to the waiting state (STATUS = 2).
iii. Repeat steps (iv) and (v) until queue is empty.
iv. Pop the top node N of stack, process N and change its status to the processed state (STATUS = 3).
v. Push onto stack all the neighbours of N that are still in the ready state (STATUS = 1) and change their status to the waiting state (STATUS = 2).
   [End of loop]
vi. End.

Que 4.6. **Write and explain DFS graph traversal algorithm.**

OR

Write DFS algorithm to traverse a graph. Apply same algorithm for the graph given in Fig. 4.6.1 by considering node 1 as starting node.

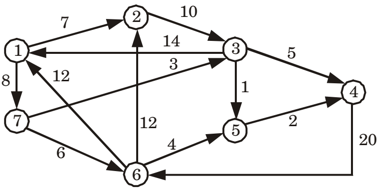


Fig. 4.6.1.

AKTU 2014-15, Marks 10

Answer

DFS : Refer Q. 4.5, Page 4–7A, Unit-4.

Numerical : Adjacency list of the given graph :

$1 \rightarrow 2, 7$

$2 \rightarrow 3$

$3 \rightarrow 5, 4, 1$

$4 \rightarrow 6$

$5 \rightarrow 4$

$6 \rightarrow 2, 5, 1$

$7 \rightarrow 3, 6$

1. Initially set STATUS = 1 for all vertex
2. Push 1 onto stack and set their STATUS = 2

1

3. Pop 1 from stack, change its STATUS = 1 and
   Push 2, 7 onto stack and change their STATUS = 2; DFS = 1

7
2

4. Pop 7 from stack, Push 3, 6; DFS = 1, 7

6
3
2

5. Pop 6 from stack, Push 5; DFS = 1, 7, 6

5
3
2

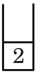
6. Pop 5 from stack, Push 4; DFS = 1, 7, 6, 5

4
3
2

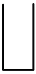
7. Pop 4 from stack; DFS = 1, 7, 6, 5, 4

3
2

8. Pop 3 from stack; DFS = 1, 7, 6, 5, 4, 3



9. Pop 2 from stack; DFS = 1, 7, 6, 5, 4, 3



Now, the stack is empty, so the depth first traversal of a given graph is 1, 7, 6, 5, 4, 3.

Que 4.7. **Implement BFS algorithm to find the shortest path from node *A* to *J*.**

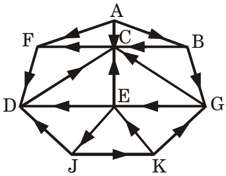


Fig. 4.7.1.

OR

Explain in detail about the graph traversal techniques with suitable example.

AKTU 2018-19, Marks 07

Answer

Following are the two traversal techniques :

1. **Depth First Search (DFS) :** Refer Q. 4.5, Page 4–7A, Unit-4.
   Example : Refer Q. 4.6, Page 4–8A, Unit-4.
2. **Breadth First Search (BFS) :** Refer Q. 4.5, Page 4–7A, Unit-4.

To find the shortest path from node *A* to node *J*

Adjacency list of the graph is :

A	:	F, C, B
B	:	G, C
C	:	F
D	:	C
E	:	D, C, J
F	:	D
G	:	C, E
J	:	D, K
K	:	E, G

a. Initially set STATUS=1 for all vertex.

b. Now add 'A' to Queue and set STATUS = 2
   Queue: A

c. Remove A from Queue and set STATUS = 3
   and add F, C, B in Queue and change their STATUS = 2
   BFS = A Queue: F, C, B

d. Remove F, add D in Queue
   BFS = A, F Queue = C, B, D,
e. Remove C, add F, but F is already visited. So no vertex will be added in this step
   BFS = A, F, Queue = B, D
f. Remove B, add G, BFS = A, F, C, B, Queue = D, G
g. Remove D, BFS = A, F, C, B, D, Queue = G
h. Remove G, add E, BFS = A, F, C, B, D, G, Queue = E
i. Remove E, add J, BFS = A, F, C, B, D, G, E, Queue = J
j. Remove J, BFS = A, F, C, B, D, G, E, J
   J is our final destination. We now back track from J to find the path from J to A : $J \leftarrow E \leftarrow G \leftarrow B \leftarrow A$

Que 4.8. **Illustrate the importance of various traversing techniques in graph along with its applications.**

AKTU 2016-17, Marks 10

Answer

Various types of traversing techniques are :
1. Breadth First Search (BFS) 2. Depth First Search (DFS)

Importance of BFS :
1. It is one of the single source shortest path algorithms, so it is used to compute the shortest path.
2. It is also used to solve puzzles such as the Rubik's Cube.
3. BFS is not only the quickest way of solving the Rubik's Cube, but also the most optimal way of solving it.

Application of BFS : Breadth first search can be used to solve many problems in graph theory, for example :
1. Copying garbage collection.
2. Finding the shortest path between two nodes u and v, with path length measured by number of edges (an advantage over depth first search).
3. Ford-Fulkerson method for computing the maximum flow in a flow network.
4. Serialization/Deserialization of a binary tree vs serialization in sorted order, allows the tree to be re-constructed in an efficient manner.
5. Construction of the failure function of the Aho-Corasick pattern matcher.
6. Testing bipartiteness of a graph.

Importance of DFS : DFS is very important algorithm as based upon DFS, there are $O(V + E)$-time algorithms for the following problems :
1. Testing whether graph is connected.
2. Computing a spanning forest of G.
3. Computing the connected components of G.
4. Computing a path between two vertices of G or reporting that no such path exists.
5. Computing a cycle in G or reporting that no such cycle exists.

Application of DFS : Algorithms that use depth first search as a building block include :

1. Finding connected components.
2. Topological sorting.
3. Finding 2-(edge or vertex)-connected components.
4. Finding 3-(edge or vertex)-connected components.
5. Finding the bridges of a graph.
6. Generating words in order to plot the limit set of a group.
7. Finding strongly connected components.

Que 4.9. **Define connected component and strongly connected component. Write an algorithm to find strongly connected components.**

Answer

Connected component : Connected component of an undirected graph is a sub-graph in which any two vertices are connected to each other by paths, and which is connected to no additional vertices in the super graph.

Strongly connected component : A directed graph is strongly connected if there is a path between all pairs of vertices. A strong component is a maximal subset of strongly connected vertices of subgraph.

Kosaraju's algorithm is used to find strongly connected components in a graph.

Kosaraju's algorithm :

1. For each vertex u of the graph, mark u as unvisited. Let L be empty.
2. For each vertex u of the graph do Visit(u), where Visit(u) is the recursive subroutine. If u is unvisited then :
   a. Mark u as visited.
   b. For each out-neighbour v of u, do Visit(v).
   c. Prepend u to L. Otherwise do nothing.
3. For each element u of L in order, do Assign(u, u) where Assign (u, root) is the recursive subroutine. If u has not been assigned to a component then :
   a. Assign u as belonging to the component whose root is root.
   b. For each in-neighbour v of u, do Assign (v, root).

   Otherwise do nothing.

PART-4

Spanning Tree, Minimum Cost Spanning Trees : Prim's and Kruskal's Algorithm.

Questions-Answers

Long Answer Type and Medium Answer Type Questions

Que 4.10. **What do you mean by spanning tree and minimum spanning tree ?**

Answer

Spanning tree :

1. A spanning tree of an undirected graph is a sub-graph that is a tree which contains all the vertices of graph.
2. A spanning tree of a connected graph G contains all the vertices and has the edges which connect all the vertices. So, the number of edges will be 1 less than the number of nodes.
3. If graph is not connected, *i.e.*, a graph with n vertices has edges less than $n - 1$ then no spanning tree is possible.
4. A connected graph may have more than one spanning trees.

Minimum spanning tree :

1. In a weighted graph, a minimum spanning tree is a spanning tree that has minimum weight than all other spanning trees of the same graph.
2. There are number of techniques for creating a minimum spanning tree for a weighted graph but the most famous methods are Prim's and Kruskal's algorithm.

Que 4.11. **Write down Prim's algorithm to find out minimal spanning tree.**

Answer

First it chooses a vertex and then chooses an edge with smallest weight incident on that vertex. The algorithm involves following steps :

Step 1 : Choose any vertex V_1 of G.

Step 2 : Choose an edge $e_1 = V_1V_2$ of G such that $V_2 \neq V_1$ and e_1 has smallest weight among the edge e of G incident with V_1.

Step 3 : If edges $e_1, e_2, \ldots\ldots, e_i$ have been chosen involving end points $V_1, V_2, \ldots\ldots\ldots\ldots, V_{i+1}$, choose an edge $e_{i+1} = V_jV_k$ with $V_j = \{V_1 \ldots\ldots V_{i+1}\}$ and $V_k \notin \{V_1 \ldots\ldots\ldots\ldots V_{i+1}\}$ such that e_{i+1} has smallest weight among the edges of G with precisely one end in $\{V_1 \ldots\ldots\ldots\ldots\ldots V_{i+1}\}$.

Step 4 : Stop after $n - 1$ edges have been chosen. Otherwise goto step 3.

Que 4.12. **Define spanning tree. Find the minimal spanning tree for the following graph using Prim's algorithm.**

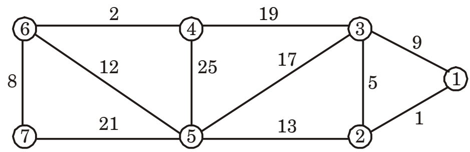


Fig. 4.12.1.

AKTU 2014-15, Marks 10

Answer

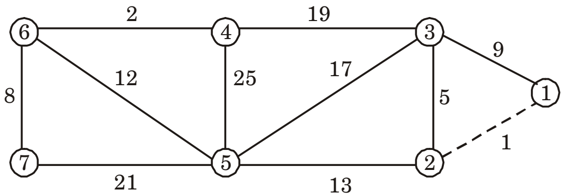
Spanning tree : Refer Q. 4.10, Page 4–13A, Unit-4.

Numerical :

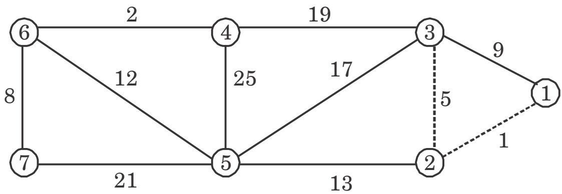
$$\begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \\ 7 \end{array} \begin{array}{c} \begin{array}{ccccccc} 1 & 2 & 3 & 4 & 5 & 6 & 7 \end{array} \\ \begin{bmatrix} - & 1 & 9 & - & - & - & - \\ 1 & - & 5 & - & 13 & - & - \\ 9 & 5 & - & 19 & 17 & - & - \\ - & - & 19 & - & 25 & 2 & - \\ - & 13 & 17 & 25 & - & 12 & 21 \\ - & - & - & 2 & 12 & - & 8 \\ - & - & - & - & 21 & 8 & - \end{bmatrix} \end{array}$$

According to Prim's algorithm, we choose vertex 1.

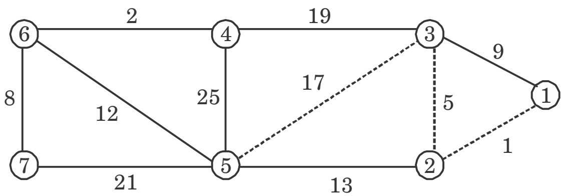
We choose edge (1, 2), since it has minimum value.



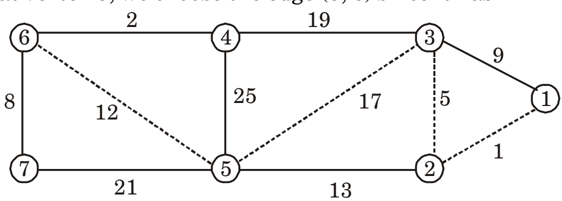
Now at vertex 2, we choose the edge (2, 3), since it has minimum value.



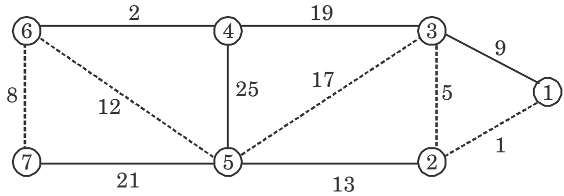
Now at vertex 3, we cannot choose edge (3, 1) because it will create a cycle so we choose (3, 5).

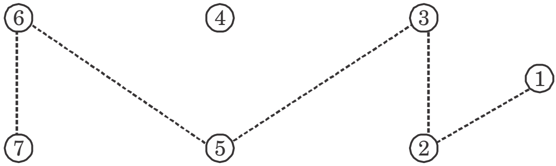


Now at vertex 5, we choose the edge (5, 6) since it has minimum value.



Now at vertex 6, we choose the edge (6, 7) since it has minimum value.





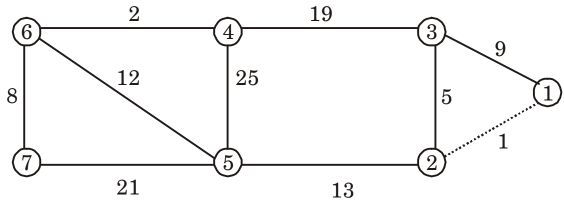
Since in spanning tree, the tree should cover all the vertices and should not make cycle.

But in the above tree, 4 is remaining so the above asked question is wrong. If we assume to remove the edge from {3, 5} then the spanning tree is :

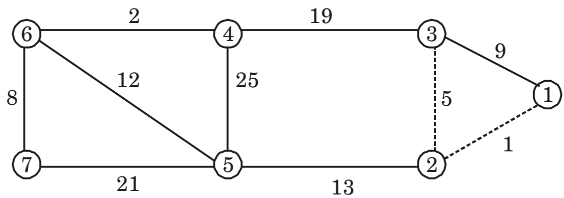
$$
\begin{array}{c@{\,}c}
 & \begin{array}{ccccccc} 1 & 2 & 3 & 4 & 5 & 6 & 7 \end{array} \\
\begin{array}{c} 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \\ 7 \end{array} &
\left[\begin{array}{ccccccc}
- & 1 & 9 & - & - & - & - \\
1 & - & 5 & - & 13 & - & - \\
9 & 5 & - & 19 & - & - & - \\
- & - & 19 & - & 25 & 2 & - \\
- & 13 & 17 & 25 & - & 12 & 21 \\
- & - & - & 2 & 12 & - & 8 \\
- & - & - & - & 21 & 8 & -
\end{array}\right]
\end{array}
$$

According to Prim's algorithm, let's choose vertex 1.

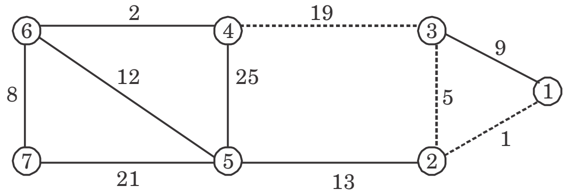
We choose edge {1, 2}, since it has minimum value.



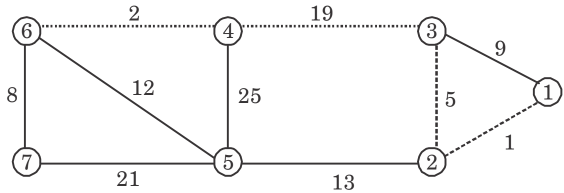
Now at vertex 2, we choose the edge (2, 3), since it has minimum value.



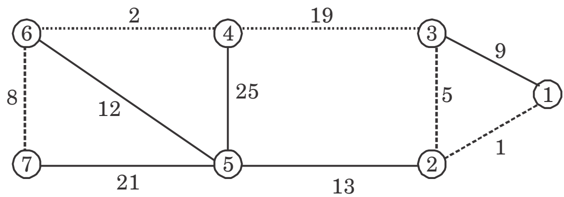
Now at vertex 3, we choose the edge (3, 4), since it has minimum value.



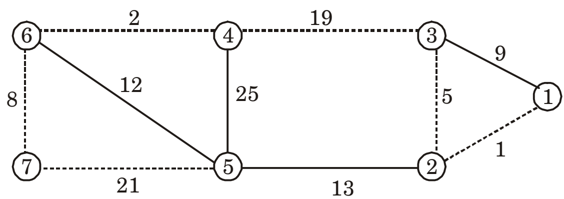
Now at vertex 4, we choose the edge (4, 6), since it has minimum value.



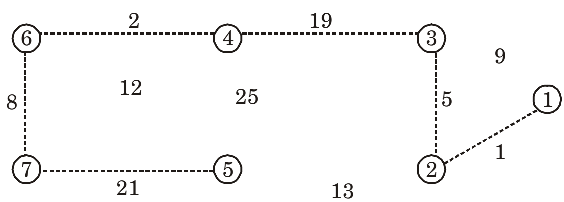
Now at vertex 6, we choose the edge (6, 7), since it has minimum value.



Now at vertex 7, we cannot choose the edge (7, 6), because we have already traversed this edge these we choose (7, 5).



∴ The spanning tree is



Que 4.13. **Define spanning tree. Also construct minimum spanning tree using Prim's algorithm for the given graph.**

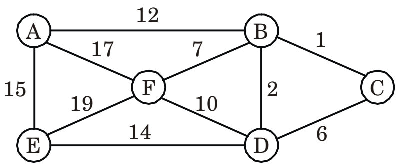


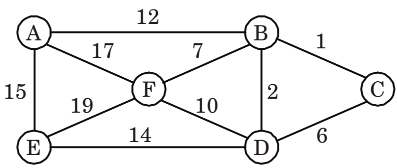
Fig. 4.13.1.

AKTU 2017-18, Marks 07

Answer

Spanning tree : Refer Q. 4.10, Page 4–13A, Unit-4.

Numerical :

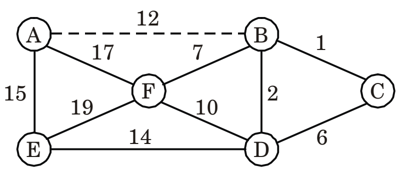


Let us take A as source node.

Now we look on weight

$w(A, B) = 12$, $w(A, F) = 17$, $w(A, E) = 15$

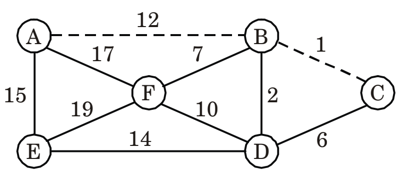
$\because$ $w(A, B)$ is smallest. Choose $e = (AB)$



Now we look on weight

$w(B, F) = 7$, $w(B, D) = 2$, $w(B, C) = 1$

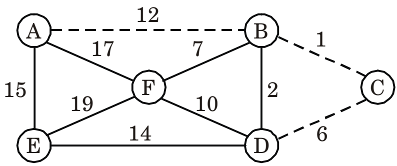
$\because$ $w(B, C)$ is smallest $\therefore$ choose $e = (BC)$



Now we look on weight

$$w(C, D) = 6$$

$\because$ $w(C, D)$ is smallest $\therefore$ choose $e = (CD)$



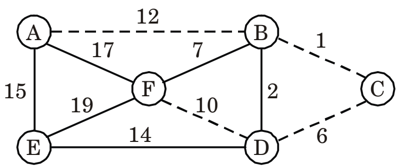
Now we look on weight

$w(D, B) = 2$, $w(D, F) = 10$, $w(D, E) = 14$

$\because$ $w(D, B)$ is smallest but forms a cycle

$\therefore$ Discard it.

Now $w(D, F) = 10$ is smallest $\therefore$ Choose $e = (DF)$



Now we look on weight

$w(F, B) = 7$, $w(F, A) = 17$, $w(F, E) = 19$

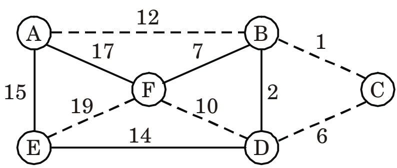
$\because$ $w(F, B)$ is smallest but forms cycle

$\therefore$ Discard it

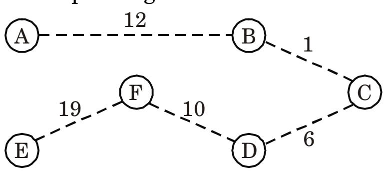
$\because$ $w(F, A)$ is smallest but forms cycle

$\therefore$ Discard it

$\therefore$ choose $e = (FE)$



The final minimum spanning tree is :



Que 4.14. **Write Kruskal's algorithm to find minimum spanning tree.**

Answer

i. In this algorithm, we choose an edge of G which has smallest weight among the edges of G which are not loops.

ii. This algorithm gives an acyclic subgraph T of G and the theorem given below proves that T is minimal spanning tree of G. Following steps are required :

Step 1 : Choose e_1, an edge of G, such that weight of e_1, $w(e_1)$ is as small as possible and e_1 is not a loop.

Step 2 : If edges $e_1, e_2, \ldots\ldots\ldots, e_i$ have been selected then choose an edge e_{i+1} not already chosen such that

i. the induced subgraph, $G[\{e_1 \ldots\ldots e_{i+1}\}]$ is acyclic and

ii. $w(e_{i+1})$ is as small as possible

Step 3 : If G has n vertices, stop after $n - 1$ edges have been chosen. Otherwise repeat step 2.

If G be a weighted connected graph in which the weight of the edges are all non-negative numbers, let T be a sub-graph of G obtained by Kruskal's algorithm then, T is minimal spanning tree.

Que 4.15. Consider the following undirected graph.

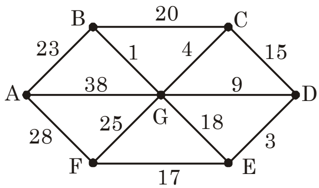


Fig. 4.15.1.

a. Find the adjacency list representation of the graph.

b. Find a minimum cost spanning tree by Kruskal's algorithm.

AKTU 2015-16, Marks 10

Answer

a.

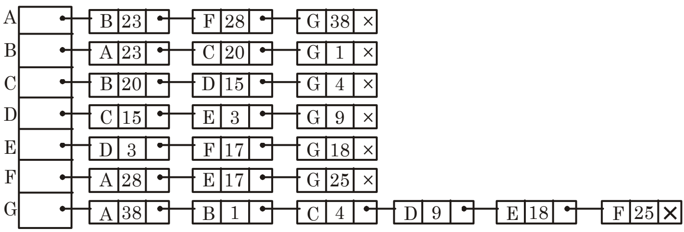
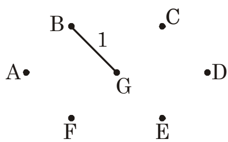


Fig. 4.15.2.

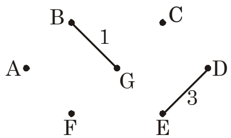
b.

Kruskal's algorithm :

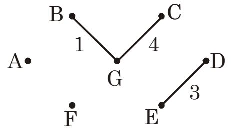
i. We will choose $e = BG$ as it has minimum weight.



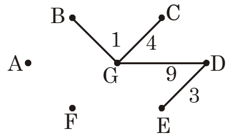
ii. Now choose $e = ED$.



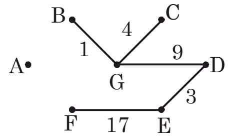
iii. Choose $e = CG$, since it has minimum weights.



iv. Choose $e = GD$.



v. Choose $e = EF$ and discard BC, CD and GE because they form cycle.



vi. Now choose $e = AB$ and discard AG, FG and AF because they form cycle. Final minimum spanning tree is given as :

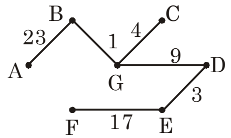


Fig. 4.15.3.

Que 4.16. **What is spanning tree ? Describe Kruskal's and Prim's algorithm to find the minimum cost spanning tree and explain the complexity. Determine the minimum cost spanning tree for the graph given below :**

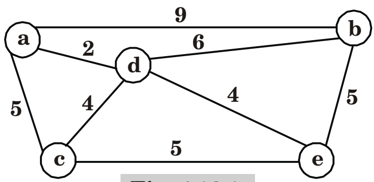


Fig. 4.16.1.

Answer

Spanning tree : Refer Q. 4.10, Page 4–13A, Unit-4.

Kruskal's algorithm : Refer Q. 4.14, Page 4–18A, Unit-4.

Prim's algorithm : Refer Q. 4.11, Page 4–13A, Unit-4.

Complexity :

A. Time complexity of Prim's algorithm :

1. The time complexity of Prim's algorithm depends on the data structures used for the graph and for ordering the edges by weight.
2. A simple implementation of Prim's, using an adjacency matrix or an adjacency list graph representation and linearly searching an array of weights to find the minimum weight edge to add, requires O($|V|2$) running time.

B. Time complexity of Kruskal's algorithm :

1. Kruskal's algorithm can be shown to run in $O(E \log E)$ time, or equivalently, $O(E \log V)$ time, where E is the number of edges in the graph and V is the number of vertices, all with simple data structures.
2. These running times are equivalent because :
   a. E is at most V^2 and $\log V^2 = 2 \log V$ is $O(\log V)$.
   b. Each isolated vertex is a separate component of the minimum spanning forest. If we ignore isolated vertices we obtain $V \leq 2E$, so $\log V$ is $O(\log E)$.

Numerical :

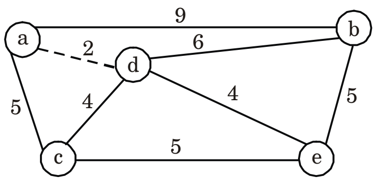
Let us take 'a' as a source node.

Now look on weight

$$w(a, d) = 2, w(a, b) = 9$$
$$w(a, c) = 5$$

$\because$ $w(a, d)$ is smallest.

$\therefore$ Choose $e = (a, d)$

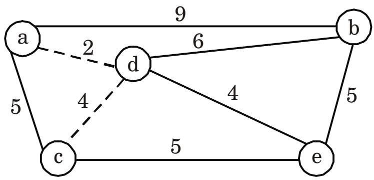


Now look on weight

$$w(d, b) = 6, w(d, c) = 4, w(d, e) = 4$$

$\because$ $w(d, c)$ is smallest.

$\therefore$ Choose $e = (d, c)$

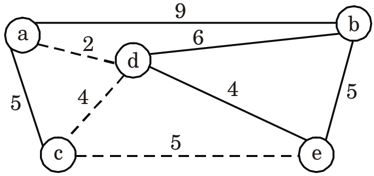


Now look on weight : $w(c, e) = 5$, $w(c, a) = 5$

$\because$ $w(c, a)$ is smallest but forms a cycle. So discard it.

Now $w(c, e)$ is smallest.

$\therefore$ Choose $e = (c, e)$

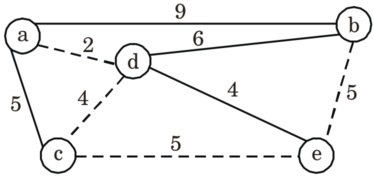


Now look on weight : $w(e, b) = 5$, $w(e, d) = 4$

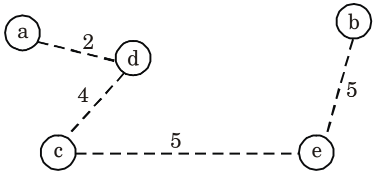
$\because$ $w(e, d)$ is smallest but forms a cycle. So discard it.

Now $w(e, b)$ is smallest.

$\therefore$ Choose $e = (e, b)$



Final minimum spanning tree is :



The minimal spanning tree is *adceb*.

Cost of minimal spanning tree is = 2 + 4 + 5 + 5 = 16.

Que 4.17. **Find the minimum spanning tree for following graph using Prim's and Kruskal's algorithms.**

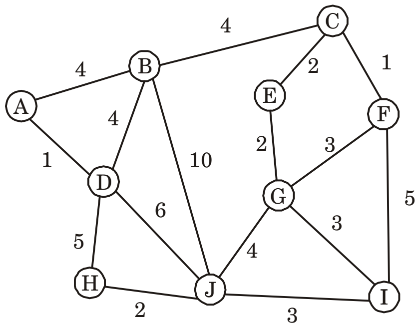


Fig. 4.17.1.

Answer

Prim's algorithm :

1. According to algorithm we choose vertex A from the set $\{A, B, C, D, E, F, G, H, I, J\}$.

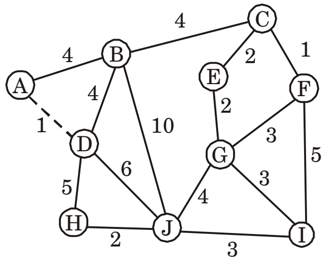


Fig. 4.17.2.

2. Now edge with smallest weight incident on A is $e = (AD)$

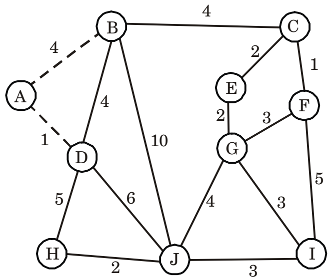


Fig. 4.17.3.

Now we look on weight

$$w(A, B) = 4, w(D, B) = 4$$
$$w(D, H) = 5, w(D, J) = 6$$

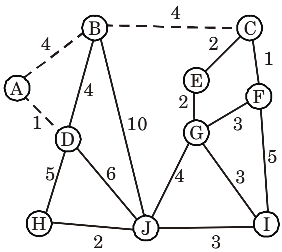


Fig. 4.17.4.

We choose $e = AB$ since it is minimum.

$w(D, B)$ can also be chosen because it has same value.

Again, $w(B, C) = 4, w(B, J) = 10, w(D, H) = 5, w(D, J) = 6$

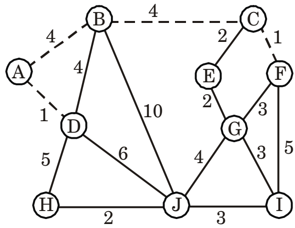


Fig. 4.17.5.

We choose $e = BC$ since it has minimum value.

Now, $w(B, J) = 10, w(C, E) = 2, w(C, F) = 1$

We choose $e = CF$ because $w(C, F)$ has minimum value.

Now, $w(C, E) = 2, w(F, G) = 3, w(F, I) = 5$

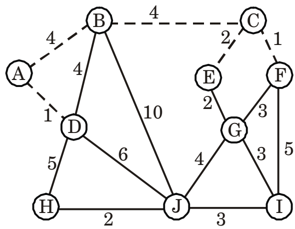


Fig. 4.17.6.

We choose $e = CE$, since $w(C, E)$ has minimum value.

$w(E, G) = 2, w(F, G) = 3, w(F, I) = 5$

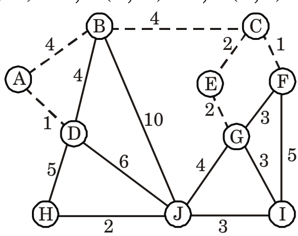


Fig. 4.17.7.

We choose $e = EG$, since $w(E, G)$ has minimum value.

$w(G, J) = 4, w(G, I) = 3, w(F, I) = 5$

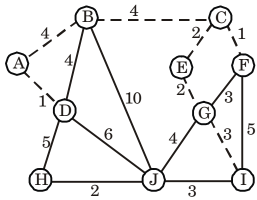


Fig. 4.17.8.

We choose $e = GI$, since $w(G, I)$has minimum value.

$$w(I, J) = 3, w(G, J) = 4$$

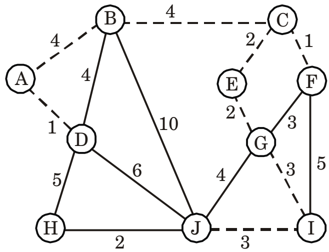


Fig. 4.17.9.

We choose $e = IJ$, since $w(I, J)$ has minimum value, $w(J, H) = 2$
Hence, $e = JH$ will be chosen. The final minimal spanning tree is :

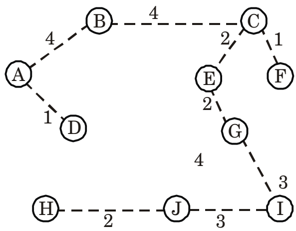


Fig. 4.17.10.

Kruskal's algorithm :

i. We will choose $e = AD$ and CF as it has minimum weight.

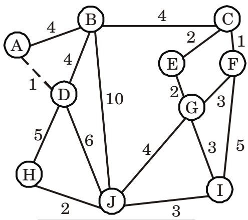


Fig. 4.17.11.

ii. Now choose $e = CF$.

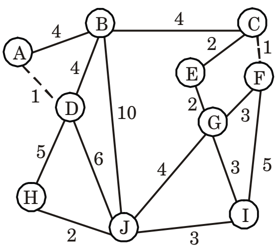


Fig. 4.17.12.

iii. Choose *CE*, *EG* and *HJ* since they have same and minimum weights.

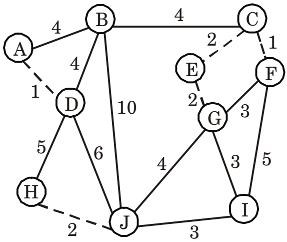


Fig. 4.17.13.

iv. Choose *IJ* and *GI* as it has minimum weight and discard *GF* because it forms cycle.

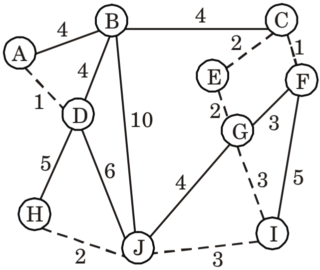


Fig. 4.17.14.

v. Choose *AB* and *BC* and discard *BD*, *GJ*, *DH*, *DJ*, *BJ*, *FI* because they form cycle.
We get the final minimal spanning tree as

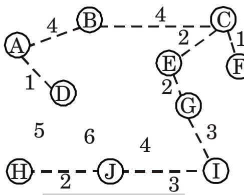


Fig. 4.17.15.

Que 4.18. **Discuss Prim's and Kruskal's algorithm. Construct minimum spanning tree for the below given graph using Prim's algorithm (Source node = *a*).** **AKTU 2016-17, Marks 15**

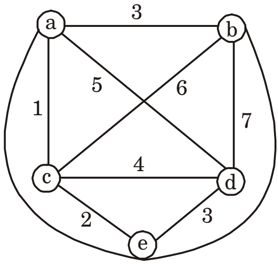


Fig. 4.18.1.

Answer

Prim's algorithm : Refer Q. 4.11, Page 4–13A, Unit-4.
Kruskal's algorithm : Refer Q. 4.14, Page 4–18A, Unit-4.
Numerical :

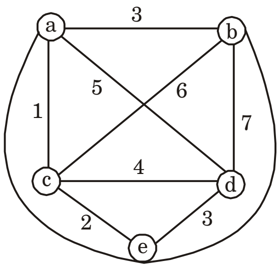


Fig. 4.18.2.

Start with source node = a

Now, edge with smallest weight incident on a is $e = (a, c)$.

So, we choose $e = (a, c)$.

Now we look on weights :

$$w(c, d) = 4, w(c, e) = 2, w(c, b) = 5$$

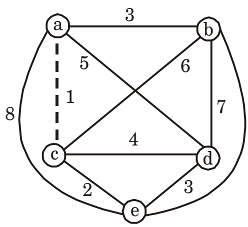


Fig. 4.18.3.

Since minimum is $w(c, e) = 2$. We choose $e = (c, e)$

Again, $w(e, d) = 3$

$w(e, a) = 8$

$w(e, b) = 7$

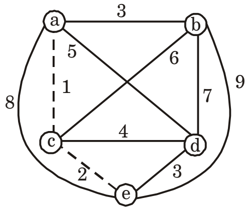


Fig. 4.18.4.

Since minimum is $w(e, d) = 3$, we choose $e = (e, d)$

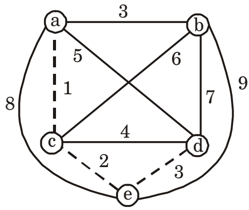


Fig. 4.18.5.

Now, $w(d, b) = 7$, and $w(a, b) = 3$
Since minimum is $w(a, b) = 3$, we choose $e = (a, b)$

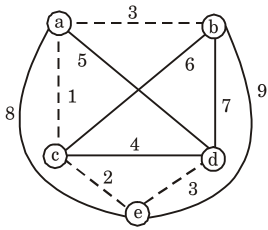


Fig. 4.18.6.

Therefore, the minimum spanning tree is :

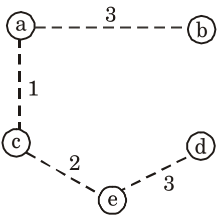


Fig. 4.18.7.

PART-5

Transitive Closure and Shortest Path Algorithm : Warshall Algorithm and Dijkstra Algorithm.

Questions-Answers

Long Answer Type and Medium Answer Type Questions

Que 4.19. **Explain transitive closure.**

Answer

1. The transitive closure of a graph G is defined to be the graph G' such that G' has the same nodes as G and there is an edge (v_i, v_j) in G' whenever there is a path from v_i to v_j in G.
2. Accordingly the path matrix P of the graph G is precisely the adjacency matrix of its transitive closure G'.
3. The transitive closure of a graph G is defined as G^* or $G' = (V, E^*)$.
   where, $E^* = \{(i, j)$ there is a path from vertex i to vertex j in $G\}$
4. We construct the transitive closure $G^* = (V, E^*)$ by putting edge (i, j) into E^* if and only if $t_{ij}(n) = 1$.
5. The recursive definition of $t_{ij}^{(k)}$ is

$$t_{ij}^{(0)} = \begin{cases} 0 \text{ if } i \neq j \text{ and } (i, j) \notin E \\ 1 \text{ if } i = j \text{ or } (i, j) \in E \end{cases}$$

and for $k \geq 1$

$$t_{ij}^{(k)} = t_{ij}^{(k-1)} \vee (t_{ik}^{(k-1)} \wedge t_{kj}^{(k-1)})$$

Que 4.20. **Write down Warshall's algorithm for finding all pair shortest path.**

Answer

1. Floyd Warshall algorithm is a graph analysis algorithm for finding shortest paths in a weighted, directed graph.
2. A single execution of the algorithm will find the shortest path between all pairs of vertices.
3. It does so in $\Theta(V^3)$ time, where V is the number of vertices in the graph.
4. Negative-weight edges may be present, but we shall assume that there are no negative-weight cycles.

5. The algorithm considers the "intermediate" vertices of a shortest path, where an intermediate vertex of a simple path $p = (v_1, v_2, ..., v_m)$ is any vertex of p other than v_1 or v_m, that is, any vertex in the set $\{v_2, v_3,..., v_{m-1}\}$.
6. Let the vertices of G be $V = \{1, 2,..., n\}$, and consider a subset $\{1, 2, ..., k\}$ of vertices for some k.
7. For any pair of vertices $i, j \in V$, consider all paths from i to j whose intermediate vertices are all drawn from $\{1, 2,..., k\}$, and let p be a minimum-weight path from among them.
8. Let $d_{ij}^{(k)}$ be the weight of a shortest path from vertex i to vertex j with all intermediate vertices in the set $\{1, 2, ..., k\}$.
   A recursive definition is given by

$$d_{ij}^{(k)} = \begin{cases} w_{ij} & \text{if } k = 0 \\ \min(d_{ij}^{(k-1)}, d_{ik}^{(k-1)} + d_{kj}^{(k-1)}) & \text{if } k \geq 1 \end{cases}$$

Floyd Warshall (w) :

1. $n \leftarrow$ rows $[w]$
2. $D^{(0)} \leftarrow w$
3. for $k \leftarrow 1$ to n
4. do for $i \leftarrow 1$ to n
5. do for $j \leftarrow 1$ to n
6. do $d_{ij}^{(k)} \leftarrow \min(d_{ij}^{(k-1)}, d_{ik}^{(k-1)} + d_{kj}^{(k-1)})$
7. return $D^{(n)}$

Que 4.21. Write the Floyd Warshall algorithm to compute the all pair shortest path. Apply the algorithm on following graph :

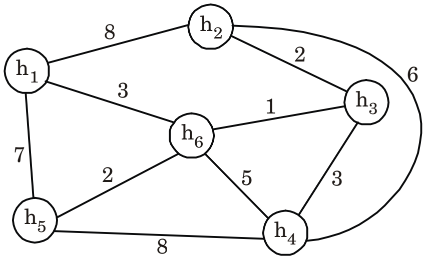


Fig. 4.21.2.

AKTU 2018-19, Marks 07

Answer

Floyd's Warshall algorithm : Refer Q. 4.20, Page 4–29A, Unit-4.
Numerical : We cannot solve this using Floyd Warshall algorithm because the given graph is undirected.

Que 4.22. **Write and explain Dijkstra's algorithm for finding shortest path.**

OR

Write and explain an algorithm for finding shortest path between any two nodes of a given graph.

Answer

a. Dijkstra's algorithm, is a greedy algorithm that solves the single-source shortest path problem for a directed graph $G = (V, E)$ with non-negative edge weights, *i.e.*, we assume that $w(u, v) \geq 0$ each edge $(u, v) \in E$.
b. Dijkstra's algorithm maintains a set S of vertices whose final shortest-path weights from the source s have already been determined.
c. That is, for all vertices $v \in S$, we have $d[v] = \delta(s, v)$.
d. The algorithm repeatedly selects the vertex $u \in V - S$ with the minimum shortest-path estimate, inserts u into S, and relaxes all edges leaving u.
e. We maintain a priority queue Q that contains all the vertices in $v - s$, keyed by their d values.
f. Graph G is represented by adjacency list.
g. Dijkstra's always chooses the "lightest or "closest" vertex in $V - S$ to insert into set S, that it uses as a greedy strategy.

DIJKSTRA (G, w, s)
1. INITIALIZE-SINGLE-SOURCE (G, s)
2. $S \leftarrow \phi$
3. $Q \leftarrow V[G]$
4. while $Q \neq \phi$
5. do $u \leftarrow$ EXTRACT-MIN (Q)
6. $S \leftarrow S \cup \{u\}$
7. for each vertex $v \in$ Adj $[u]$
8. do RELAX (u, v, w)

Que 4.23. **Find out the shortest path from node 1 to node 4 in a given graph (Fig. 4.23.1) using Dijkstra shortest path algorithm.**

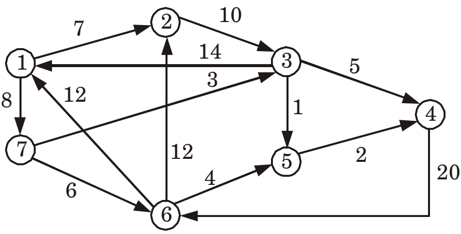


Fig. 4.23.1.

AKTU 2014-15, Marks 10

Answer

1	2	3	4	5	6	7
$\boxed{0}$	∞	∞	∞	∞	∞	∞
$\boxed{0}$	$\boxed{7}$	$\boxed{17}$	∞	∞	∞	8
$\boxed{0}$	$\boxed{7}$	$\boxed{17}$	∞	∞	∞	$\boxed{8}$
$\boxed{0}$	$\boxed{7}$	$\boxed{11}$	∞	∞	14	$\boxed{8}$
$\boxed{0}$	$\boxed{7}$	$\boxed{11}$	16	$\boxed{12}$	14	$\boxed{8}$
$\boxed{0}$	$\boxed{7}$	$\boxed{11}$	14	$\boxed{12}$	14	$\boxed{8}$
$\boxed{0}$	$\boxed{7}$	$\boxed{11}$	$\boxed{14}$	$\boxed{12}$	$\boxed{14}$	$\boxed{8}$

Shortest path from node 1 to node 4 = 0 + 7 + 11 + 14 = 32

Que 4.24. **Describe Dijkstra's algorithm for finding shortest path. Describe its working for the graph given below.**

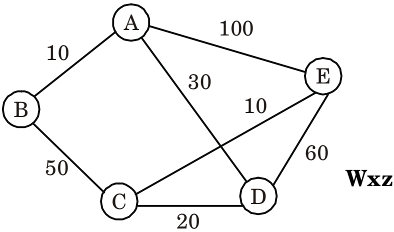


Fig. 4.24.1.

AKTU 2017-18, Marks 07

OR

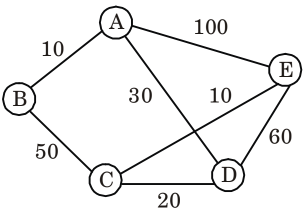
Explain Dijkstra's algorithm with suitable example.

AKTU 2015-16, Marks 10

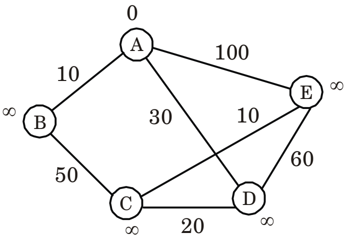
Answer

Algorithm : Refer Q. 4.22, Page 4–31A, Unit-4.

Numerical :

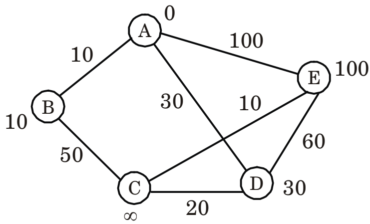


Extract min (*A*) :



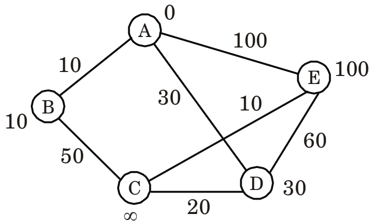
A	B	C	D	E
$\boxed{0}$	∞	∞	∞	∞

All edges leaving *A* :



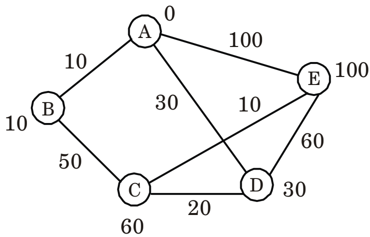
A	B	C	D	E
$\boxed{0}$	∞	∞	∞	∞
	10	∞	30	100

Extract min (*B*) :



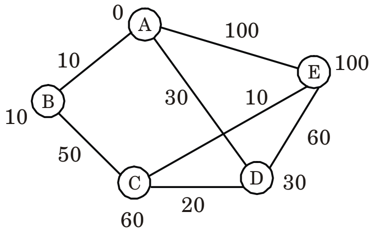
A	B	C	D	E
$\boxed{0}$	∞	∞	∞	∞
	$\boxed{10}$	∞	30	100

All edges leaving *B* :



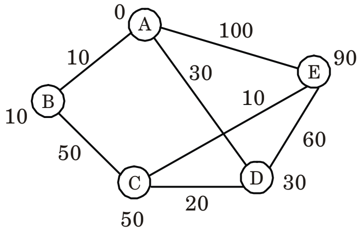
A	B	C	D	E
$\boxed{0}$	∞	∞	∞	∞
	$\boxed{10}$	∞	30	100
		60	30	100

Extract min(D) :



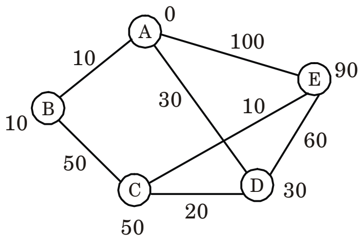
A	B	C	D	E
$\boxed{0}$	∞	∞	∞	∞
	$\boxed{10}$	∞	30	100
		60	$\boxed{30}$	100

All edges leaving (D) :



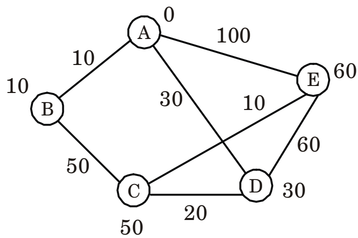
A	B	C	D	E
$\boxed{0}$	∞	∞	∞	∞
	$\boxed{10}$	∞	30	100
		60	$\boxed{30}$	100
		50		90

Extract min(C) :



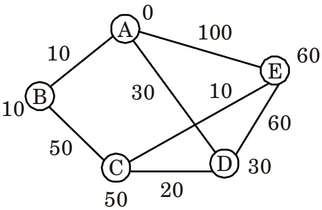
A	B	C	D	E
$\boxed{0}$	∞	∞	∞	∞
	$\boxed{10}$	∞	30	100
		60	$\boxed{30}$	100
		$\boxed{50}$		90

All edges leaving C :



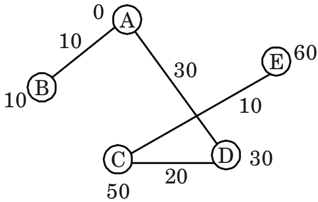
A	B	C	D	E
$\boxed{0}$	∞	∞	∞	∞
	$\boxed{10}$	∞	30	100
		60	$\boxed{30}$	100
		$\boxed{50}$		90
				60

Extract min(*E*) :



A	B	C	D	E
$\boxed{0}$	∞	∞	∞	∞
	$\boxed{10}$	∞	30	100
		60	$\boxed{30}$	100
		$\boxed{50}$		90
				$\boxed{60}$

Shortest path :



Que 4.25. **By considering vertex '1' as source vertex, find the shortest paths to all other vertices in the following graph using Dijkstra's algorithms. Show all the steps.**

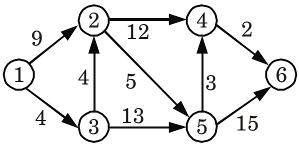
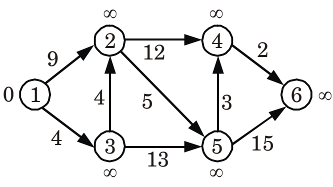


Fig. 4.25.1.

AKTU 2018-19, Marks 07

Answer

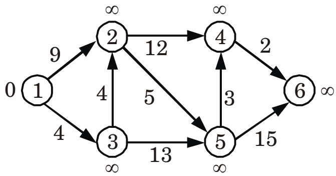
Initialize :



S = { }

Q :	1	2	3	4	5	6
	0	∞	∞	∞	∞	∞

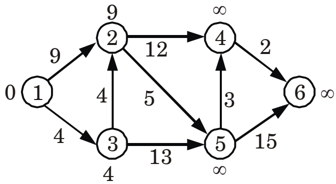
EXTRACT – MIN (1) :



S = { 1 }

Q :	1	2	3	4	5	6
	$\boxed{0}$	∞	∞	∞	∞	∞

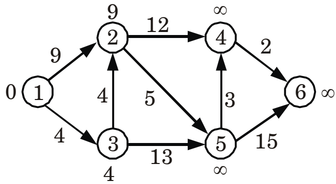
Relax all edges leaving 1 :



S = { 1 }

Q :	1	2	3	4	5	6
	$\boxed{0}$	∞	∞	∞	∞	∞
		9	4	–	–	–

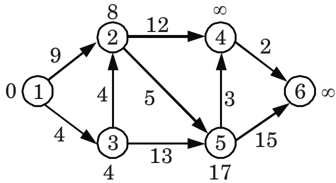
EXTRACT – MIN (3) :



S = { 1, 3 }

Q :	1	2	3	4	5	6
	$\boxed{0}$	∞	∞	∞	∞	∞
		9	$\boxed{4}$	–	–	–

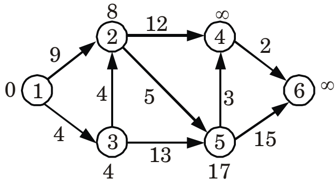
Relax all edges leaving 3 :



S = { 1, 3 }

Q :	1	2	3	4	5	6
	$\boxed{0}$	∞	∞	∞	∞	∞
		9	$\boxed{4}$	–	–	–
		8		–	17	–

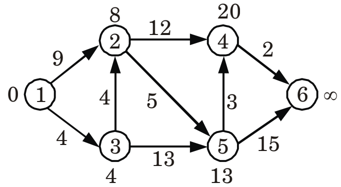
EXTRACT – MIN (2) :



S = { 1, 3, 2 }

Q :	1	2	3	4	5	6
	$\boxed{0}$	∞	∞	∞	∞	∞
		9	$\boxed{4}$	–	–	–
		$\boxed{8}$		–	17	–

Relax all edges leaving 2 :

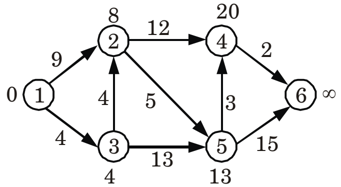


S = { 1, 3, 2 }

Q :

1	2	3	4	5	6
$\boxed{0}$	∞	∞	∞	∞	∞
	9	$\boxed{4}$	–	–	–
	$\boxed{8}$		–	17	–
			20	13	–

EXTRACT – MIN (5) :

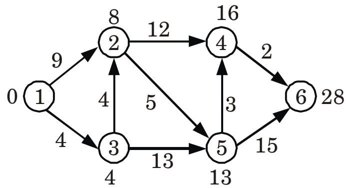


S = { 1, 3, 2, 5 }

Q :

1	2	3	4	5	6
$\boxed{0}$	∞	∞	∞	∞	∞
	9	$\boxed{4}$	–	–	–
	$\boxed{8}$		–	17	–
			20	$\boxed{13}$	–

Relax all edges leaving 5 :

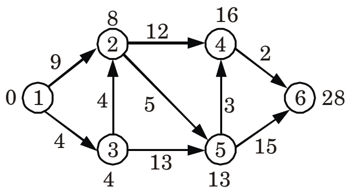


S = { 1, 3, 2, 5 }

Q :

1	2	3	4	5	6
$\boxed{0}$	∞	∞	∞	∞	∞
	9	$\boxed{4}$	–	–	–
	$\boxed{8}$		–	17	–
			20	$\boxed{13}$	–
			16		28

EXTRACT – MIN (4) :

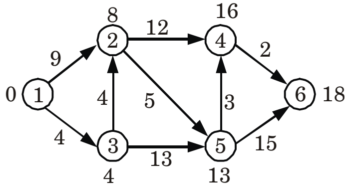


S = { 1, 3, 2, 5, 4 }

Q :

1	2	3	4	5	6
$\boxed{0}$	∞	∞	∞	∞	∞
	9	$\boxed{4}$	–	–	–
	$\boxed{8}$		–	17	–
			20	$\boxed{13}$	–
			$\boxed{16}$		28

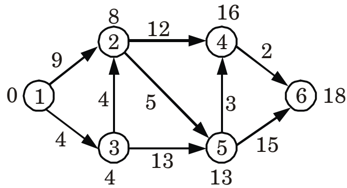
Relax all edges leaving 4 :



S = { 1, 3, 2, 5, 4 }

Q :	1	2	3	4	5	6
	$\boxed{0}$	∞	∞	∞	∞	∞
		9	$\boxed{4}$	–	–	–
		$\boxed{8}$		–	17	–
				20	$\boxed{13}$	–
				$\boxed{16}$		28
						18

EXTRACT – MIN (6) :



S = { 1, 3, 2, 5, 4, 6 }

Q :	1	2	3	4	5	6
	$\boxed{0}$	∞	∞	∞	∞	∞
		9	$\boxed{4}$	–	–	–
		$\boxed{8}$		–	17	–
				20	$\boxed{13}$	–
				$\boxed{16}$		28
						$\boxed{18}$

VERY IMPORTANT QUESTIONS

Following questions are very important. These questions may be asked in your SESSIONALS as well as UNIVERSITY EXAMINATION.

Q. 1. Write DFS algorithm to traverse a graph. Apply same algorithm for the graph given in Fig. 1 by considering node 1 as starting node.

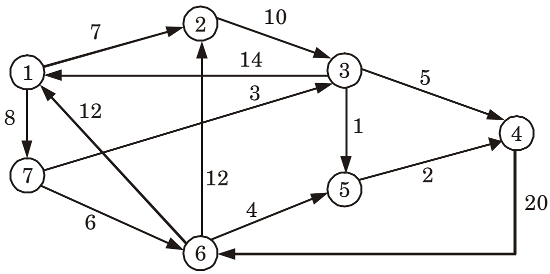


Fig. 1.

Ans. Refer Q. 4.6.

Q. 2. Illustrate the importance of various traversing techniques in graph along with its applications.

Ans. Refer Q. 4.8.

Q. 3. Define spanning tree. Also construct minimum spanning tree using Prim's algorithm for the given graph.

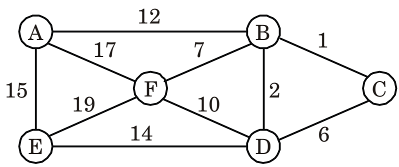


Fig. 2.

Ans. Refer Q. 4.13.

Q. 4. Consider the following undirected graph.

a. Find the adjacency list representation of the graph.

b. Find a minimum cost spanning tree by Kruskal's algorithm.

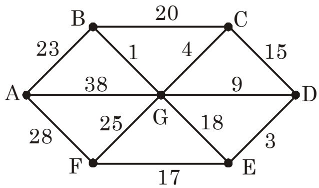


Fig. 3.

Ans. Refer Q. 4.15.

Q. 5. Discuss Prim's and Kruskal's algorithm. Construct minimum spanning tree for the below given graph using Prim's algorithm (Source node = *a*).

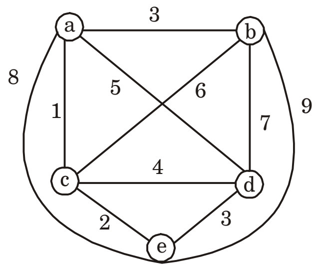


Fig. 4.

Ans. Refer Q. 4.18.

Q. 6. Write the Floyd Warshall algorithm to compute the all pair shortest path. Apply the algorithm on following graph :

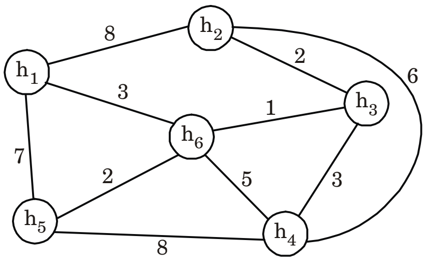


Fig. 5.

Ans. Refer Q. 4.21.

Q. 7. Describe Dijkstra's algorithm for finding shortest path. Describe its working for the graph given below.

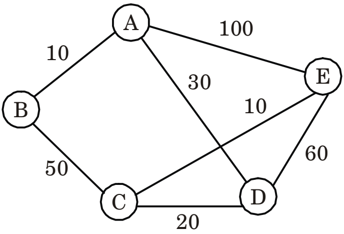


Fig. 6.

Ans. Refer Q. 4.24.

Q. 8. Find out the shortest path from node 1 to node 4 in a given graph (Fig. 7) using Dijkstra shortest path algorithm.

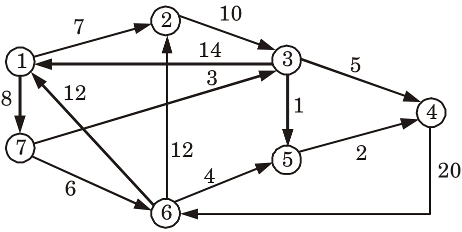


Fig. 7.

Ans. Refer Q. 4.23.

Q. 9. By considering vertex '1' as source vertex, find the shortest paths to all other vertices in the following graph using Dijkstra's algorithms. Show all the steps.

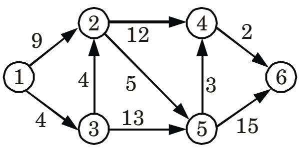


Fig. 8.

Ans. Refer Q. 4.25.

☺☺☺

5

UNIT

Trees

CONTENTS

PART-1

Basic Terminology used with Tree, Binary Trees, Binary Tree Representation : Array Representation and Pointer (Linked List) Representation.

Questions-Answers

Long Answer Type and Medium Answer Type Questions

Que 5.1. **Explain the following terms :**

i. Tree
ii. Vertex of Tree
iii. Depth
iv. Degree of an element
v. Degree of Tree
vi. Leaf

Answer

i. **Tree :** A tree T is a finite non-empty set of elements. One of these elements is called the root, and the remaining elements, if any is partitioned into trees is called subtree of T. A tree is a non-linear data structure.

ii. **Vertex of tree :** Each node of a tree is known as vertex of tree.

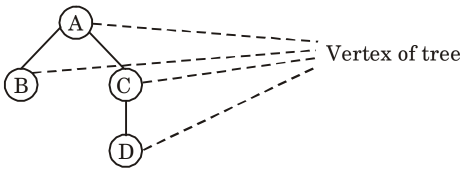


Fig. 5.1.1.

iii. **Depth :** The depth of binary tree is the maximum level of any leaf in the tree. This equals the length of the longest path from the root to any leaf.

Depth of Fig. 5.1.2 tree is 2.

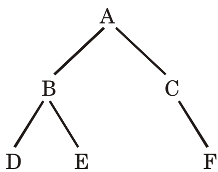


Fig. 5.1.2.

iv. **Degree of an element :** The number of children of node is known as degree of the element.

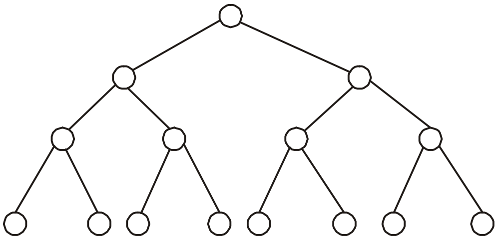
v. **Degree of tree :** In a tree, node having maximum number of degree is known as degree of tree.

vi. **Leaf :** A terminal node in tree is known as leaf node or a node which has no child is known as leaf node.

Que 5.2. **Show that the maximum number of nodes in a binary tree of height h is $2^{h+1} - 1$.**

Answer

If we consider the maximum nodes in a tree then all leaves will have the same depth and all internal nodes have left child and right child both.



Node

depth 0 $2^0 = 1$

depth 1 $2^1 = 2$

depth 2 $2^2 = 4$

depth 3 $2^3 = 8$

Fig. 5.2.1.

1. The root has 2 children at depth 1, each of which has 2 children at depth 2 *i.e.*, 4.
2. Thus, the number of leaves at depth h is 2^h, so we can calculate the maximum number of nodes in a binary tree as :

$$= 1 + 2 + 4 + 8 + 16 + \dots 2^h$$
$$= 2^0 + 2^1 + 2^2 + 2^3 + \dots 2^h$$
$$= \sum_{i=0}^{h} 2^i = \frac{2^{h+1} - 1}{2 - 1} = 2^{h+1} - 1$$

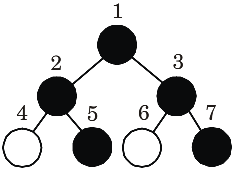
Thus, a binary tree having height h, has $2^{h+1} - 1$ maximum number of nodes.

Que 5.3. **Explain binary tree representation using array.**

Answer

1. In an array representation, nodes of the tree are stored level-by-level, starting from 0^{th} level.
2. Missing elements are represented by white boxes.
3. This representation scheme is wasteful of space when many elements are missing.
4. In fact, a binary tree that has n-elements may require an array of size up to 2^n (including position 0) for its representation.

5. This maximum size is needed when each element (except the root) of the n-element binary tree is the right child of its parent.
6. Fig. 5.3.1, shows such a binary tree with four elements. Binary trees of this type are called right-skewed binary trees.



A	B	C		D		E

Fig. 5.3.1.

Que 5.4. **Explain binary tree representation using linked list.**

Answer

1. Consider a binary tree T which uses three parallel arrays, INFO, LEFT and RIGHT, and a pointer variable ROOT.
2. First of all, each node N of T will correspond to a location K such that :
   a. INFO[K] contains the data at the node N.
   b. LEFT[K] contains the location of the left child of node N.
   c. RIGHT[K] contains the location of the right child of node N.
3. ROOT will contain the location of the root R of T.
4. If any subtree is empty, then the corresponding pointer will contain the null value.
5. If the tree T itself is empty, then ROOT will contain the null value.
6. INFO may actually be a linear array of records or a collection of parallel arrays.

Que 5.5. **Write a C program to implement binary tree insertion, deletion with example.** **AKTU 2016-17, Marks 10**

Answer

```
#include<stdlib.h>
#include<stdio.h>
struct bin_tree {
int data;
struct bin_tree *right, *left;
};
typedef struct bin_tree node;
void insert(node *tree, int val)
{
node *temp = NULL;
if(!(*tree))
{
```

```
temp = (node *)malloc(sizeof(node));
temp->left = temp->right = NULL;
temp->data = val;
*tree = temp;
return;
}
if(val < (*tree)->data)
{
insert(&(*tree)->left, val);
}
else if(val > (*tree)->data)
{
insert(&(*tree)->right, val);
}
}
void print_inorder(node *tree)
{
if (tree)
{
print_inorder(tree->left);
printf("%d\n",tree->data);
print_inorder(tree->right);
}
}
void deltree(node *tree)
{
if (tree)
{
deltree(tree->left);
deltree(tree->right);
free(tree);
}
}
void main()
{
node *root;
node *tmp;
//int i;
root = NULL;
/* Inserting nodes into tree */
insert(&root, 9);
insert(&root, 4);
insert(&root, 15);
insert(&root, 6);
insert(&root, 12);
insert(&root, 17);
insert(&root, 2);
```

```
/* Printing nodes of tree */
printf("After insertion inorder display\n");
print_inorder(root);
/* Deleting all nodes of tree */
deltree(root);
printf("Tree is empty");
}
```

Output of program :

After insertion inorder display
2
4
6
9
12
15
17
Tree is empty.

Que 5.6. **Write the C program for various traversing techniques of binary tree with neat example.** **AKTU 2016-17, Marks 10**

Answer

```
#include<stdio.h>
#include<stdlib.h>
struct node
{
int value;
node* left;
node* right;
};
struct node* root;
struct node* insert(struct node* r, int data);
void inorder(struct node* r);
void preorder(struct node* r);
void postorder(struct node* r);
int main()
{
root = NULL;
int n, v;
printf("How many data do you want to insert ?\n");
scanf("%d", &n);
for(int i=0; i<n; i++){
printf("Data %d: ", i+1);
scanf("%d", &v);
root = insert(root, v);
```

```
}
printf("Inorder Traversal :");
inorder(root);
printf("\n");
printf("Preorder Traversal :");
preorder(root);
printf("\n");
printf("Postorder Traversal :");
postorder(root);
printf("\n");
return 0;
}
struct node* insert(struct node* r, int data)
{
if(r==NULL)
{
r = (struct node*) malloc(sizeof(struct node));
r->value = data;
r->left = NULL;
r->right = NULL;
}
else if(data < r->value){
r->left = insert(r->left, data);
}
else {
r->right = insert(r->right, data);
}
return r;
}
void inorder(struct node* r)
{
if(r!=NULL){
inorder(r->left);
printf("%d ", r->value);
inorder(r->right);
}
}
void preorder(struct node* r)
{
if(r!=NULL){
printf("%d", r->value);
preorder(r->left);
preorder(r->right);
}
}
void postorder(struct node* r)
{
```

```
if(r!=NULL){
postorder(r->left);
postorder(r->right);
printf("%d", r->value);
}
}
```

Output :

How many data do you want to insert ?
5
Preorder Traversal :
3 2 1 4 5
Inorder Traversal :
1 2 3 4 5
Postorder Traversal :
1 2 5 4 3

PART-2

Binary Search Tree, Strictly Binary Tree, Complete Binary Tree, A Extended Binary Tree.

Questions-Answers

Long Answer Type and Medium Answer Type Questions

Que 5.7. Explain binary search tree and its operations. Make a binary search tree for the following sequence of numbers, show all steps : 45, 32, 90, 34, 68, 72, 15, 24, 30, 66, 11, 50, 10.

AKTU 2015-16, Marks 10

Answer

Binary search tree :

1. A binary search tree is a binary tree.
2. Binary search tree can be represented by a linked data structure in which each node is an object.
3. In addition to a key field, each node contains fields left, right and *P*, which point to the nodes corresponding to its left child, its right child and its parent respectively.
4. A non-empty binary search tree satisfies the following properties :
   a. Every element has a key (or value) and no two elements have the same value.
   b. The keys, if any, in the left subtree of root are smaller than the key in the node.

c. The keys, if any in the right subtree of the root are larger than the keys in the node.

d. The left and right subtrees of the root are also binary search tree.

Various operations of BST are :

a. Searching in a BST :

Searching for a data in a binary search tree is much faster than in arrays or linked lists. The TREE-SEARCH (x, k) algorithm searches the tree root at x for a node whose key value equals to k. It returns a pointer to the node if it exist otherwise NIL.

TREE-SEARCH (x, k)

1. If x = NIL or k = key [x]
2. then return x
3. If k < key [x]
4. then return TREE-SEARCH (left [x], k)
5. else return TREE-SEARCH (right [x], k)

b. Traversal operation on BST :

All the traversal operations are applicable in binary search trees. The inorder traversal on a binary search tree gives the sorted order of data in ascending (increasing) order.

c. Insertion of data into a binary search tree :

To insert a new value w into a binary search tree T, we use the procedure TREE-INSERT. The procedure passed a node z for which key[z] = w, left [z] = NIL and Right [z] = NIL.

1. $y \leftarrow$ NIL
2. $x \leftarrow$ root [T]
3. while $x \neq$ NIL
4. do $y \leftarrow x$
5. if key [z] < key [x]
6. then $x \leftarrow$ left [x]
7. else $x \leftarrow$ right [x]
8. $P[z] \leftarrow y$
9. if y = NIL
10. then root [T] $\leftarrow z$
11. else if key [z] < key [y]
12. then left [y] $\leftarrow z$
13. else right [y] $\leftarrow z$

d. Delete a node : Deletion of a node from a BST depends on the number of its children. Suppose to delete a node with key = z from BST T, there are 3 cases that can occur.

Case 1 : N has no children. Then N is deleted from T by simply replacing the location of N in the parent node $P(N)$ by the null pointer.

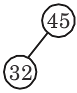
Case 2 : N has exactly one child. Then N is deleted from T by simply replacing the location of N in $P(N)$ by the location of the only child of N.

Case 3 : N has two children. Let $S(N)$ denote the inorder successor of N. (The reader can verify that $S(N)$ does not have a left child). Then N is deleted from T by first deleting $S(N)$ from T (by using Case 1 or Case 2) and then replacing node N in T by the node $S(N)$.

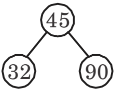
Numerical :

1. Insert 45 :

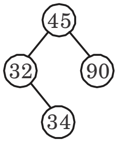
2. Insert 32 :



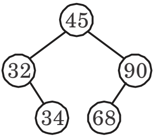
3. Insert 90 :



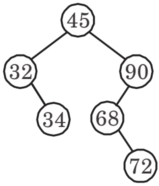
4. Insert 34 :



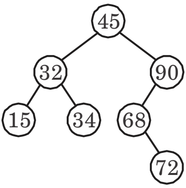
5. Insert 68 :



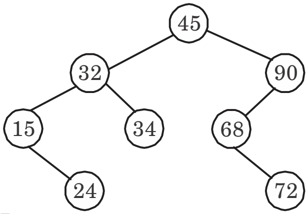
6. Insert 72 :



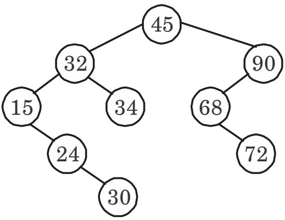
7. Insert 15 :



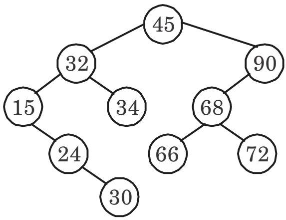
8. Insert 24 :



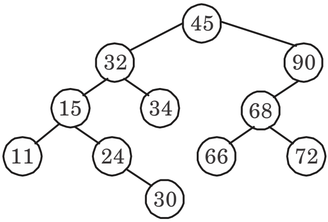
9. Insert 30 :



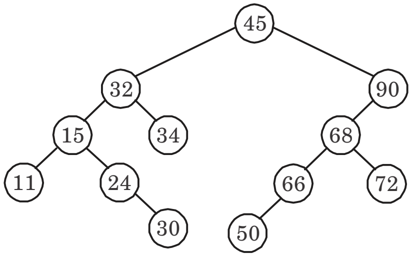
10. Insert 66 :



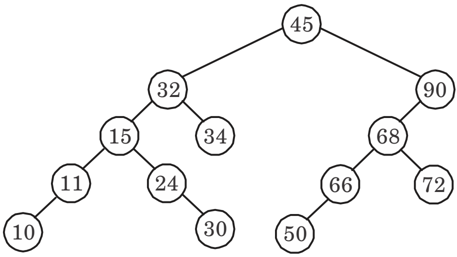
11. Insert 11 :



12. Insert 50 :



13. Insert 10 :



Que 5.8. **Define binary search tree. Create BST for the following data, show all steps :**

20, 10, 25, 5, 15, 22, 30, 3, 14, 13 **AKTU 2014-15, Marks 10**

Answer

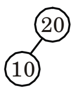
Binary search tree : Refer Q. 5.7, Page 5–8A, Unit-5.

Numerical :

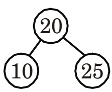
20, 10, 25, 5, 15, 22, 30, 3, 14, 13

1. Insert 20 :

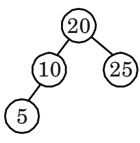
2. Insert 10 :



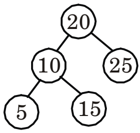
3. Insert 25 :



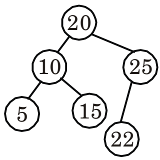
4. Insert 5 :



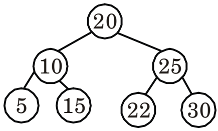
5. Insert 15 :



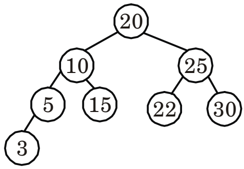
6. Insert 22 :



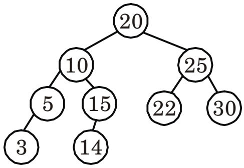
7. Insert 30 :



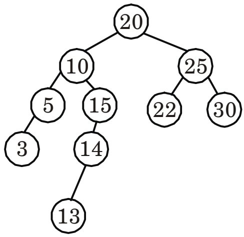
8. Insert 3 :



9. Insert 14 :



10. Insert 13 :



Que 5.9. Write a short note on strictly binary tree, complete binary tree and extended binary tree.

Answer

Strictly binary tree :

a. If every non-leaf node in a binary tree has non-empty left and right subtree, the tree is termed as strictly binary tree.
b. A strictly binary tree with n leaves always contains $2n - 1$ nodes.
c. If every non-leaf node in a binary tree has exactly two children, the tree is known as strictly binary tree.

Complete binary tree : A tree is called a complete binary tree if tree satisfies following conditions :

a. Each node has exactly two children except leaf node.
b. All leaf nodes are at same level.
c. If a binary tree contains m nodes at level l, it contains atmost $2m$ nodes at level $l + 1$.

Extended binary tree :

a. A binary tree T is said to be 2-tree or extended binary tree if each node has either 0 or 2 children.
b. Nodes with 2 children are called internal nodes and nodes with 0 children are called external nodes.

PART-3

Tree Traversal Algorithm : Inorder, Preorder and Postorder, Constructing Binary Tree From Given Tree Traversal.

Questions-Answers

Long Answer Type and Medium Answer Type Questions

Que 5.10. Define tree, binary tree, complete binary tree and full binary tree. Write algorithm or function to obtain traversals of a binary tree in preorder, postorder and inorder.

AKTU 2017-18, Marks 07

Answer

Tree : Refer Q. 5.1, Page 5–2A, Unit-5.

Binary tree :

1. A binary tree T is defined as a finite set of elements called nodes, such that :
   a. T is empty (called the null tree).
   b. T contains a distinguished node R, called the root of T, and the remaining nodes of T form an ordered pair of disjoint binary trees T_1 and T_2.
2. If T does contain a root R, then the two trees T_1 and T_2 are called, respectively, the left and right subtrees of R.
3. If T_1 is non-empty, then its root is called the left successor of R similarly, if T_2 is non-empty, then its root is called the right successor of R.

Complete binary tree : Refer Q. 5.10, Page 5–14A, Unit-5.

Full binary tree :

1. A full binary tree is formed when each missing child in the binary tree is replaced with a node having no children.
2. These leaf nodes are drawn as squares in the Fig. 5.10.1.

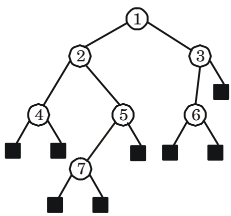


Fig. 5.10.1. Full binary tree.

3. Each node is either a leaf or has degree exactly 2.

Algorithm for preorder traversal :

Preorder (INFO, LEFT, RIGHT, ROOT)

1. [Initially push NULL onto STACK, and initialize PTR]
   Set TOP = 1, STACK [1] = NULL and PTR = ROOT
2. Repeat steps 3 to 5 while PTR $\neq$ NULL
3. Apply process to INFO [PTR]
4. [Right child?]
   If RIGHT [PTR] $\neq$ NULL
   Then
   [Push on STACK]
   Set TOP = TOP + 1 and
   STACK [TOP] = RIGHT [PTR]
   Endif
5. [Left child?]
   If LEFT [PTR] $\neq$ NULL then
   set PTR = LEFT[PTR]
   Else
   [Pop from STACK]
   set PTR = STACK[TOP] and TOP = TOP – 1
   Endif
   End of step 2
6. Exit

Algorithm for inorder traversal :

Inorder (INFO, LEFT, RIGHT, ROOT)

1. [Push NULL onto STACK and initialize PTR]
   Set TOP = 1, STACK[1] = NULL and PTR = ROOT
2. Repeat while PTR $\neq$ NULL
   [Push leftmost path onto STACK]
   a. Set TOP = TOP + 1 and
   STACK [TOP] = PTR
   b. Set PTR = LEFT [PTR]
   End loop
3. Set PTR = STACK[TOP] and TOP = TOP – 1
4. Repeat steps 5 to 7 while PTR $\neq$ NULL
5. Apply process to INFO[PTR]
6. [Right Child?] If RIGHT [PTR] $\neq$ NULL
   Then
   a. Set PTR = RIGHT [PTR]
   b. goto step 2
   Endif
7. Set PTR = STACK[TOP] and TOP = TOP – 1
   End of Step 4 Loop
8. Exit

Algorithm for postorder traversal :

Postorder (INFO, LEFT, RIGHT, ROOT)

1. [Push NULL onto STACK and initialize PTR]
   Set TOP = 1, STACK[1] = NULL and PTR = ROOT
2. [Push leftmost path onto STACK]
   Repeat steps 3 to 5 while PTR $\neq$ NULL
3. Set TOP = TOP + 1 and STACK [TOP] = PTR
   [Pushes PTR on STACK]
4. If RIGHT [PTR] $\neq$ NULL
   Then
   Set TOP = TOP + 1 and STACK [TOP] = RIGHT [PTR]
   Endif
5. Set PTR = LEFT [PTR]
   End of step 2 loop
6. Set PTR = STACK [TOP] and TOP = TOP – 1
   [Pops node from STACK]
7. Repeat while PTR > 0
   a. Apply process to INFO [PTR]
   b. Set PTR = STACK [TOP] and TOP = TOP – 1
   End loop
8. If PTR < 0 Then
   a. Set PTR = – PTR
   b. goto step 2
   Endif
9. Exit

Que 5.11. **Construct a binary tree for the following :**

Inorder : ***Q, B, K, C, F, A, G, P, E, D, H, R***

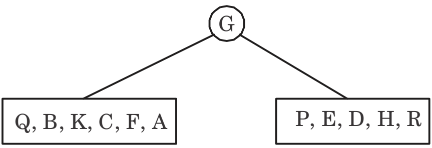
Preorder : ***G, B, Q, A, C, K, F, P, D, E, R, H***

Find the postorder of the tree. **AKTU 2018-19, Marks 07**

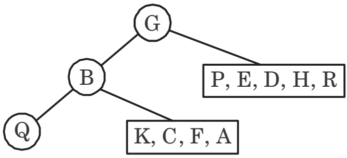
Answer

Step 1 : In preorder traversal root is the first node. So, *G* is the root node of the binary tree. So,

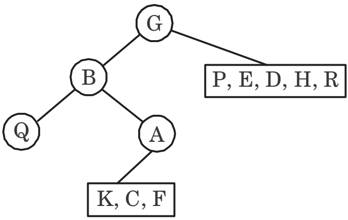
Step 2 : We can find the node of left sub-tree and right sub-tree with inorder sequence. So,



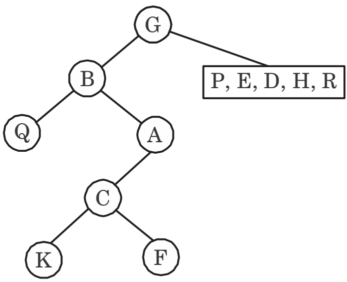
Step 3 : Now, the left child of the root node will be the first node in the preorder sequence after root node *G*. So,



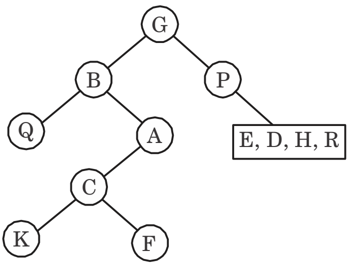
Step 4 : In inorder sequence, *Q* is on the left side of *B* and *A* is on the right side *B*. So,



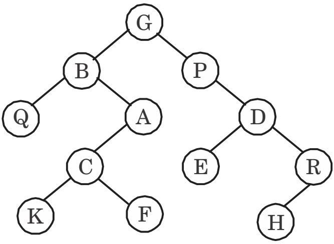
Step 5 : In inorder sequence, *C* is on the left side of *A* . Now according to inorder sequence, *K* is on the left side of *C* and *F* is on the right side of *C*.



Step 6 : Similarly, we can go further for right side of *G*.



So, the final tree is



Postorder of tree : *Q*, *K*, *F*, *A*, *B*, *E*, *H*, *R*, *D*, *P*, *G*

Que 5.12. **Draw a binary tree with following traversal :**

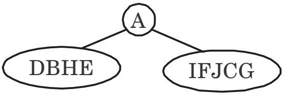
Inorder : ***D B H E A I F J C G***

Preorder : ***A B D E H C F I J G***

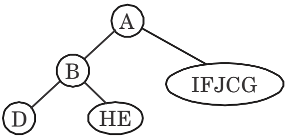
AKTU 2015-16, Marks 10

Answer

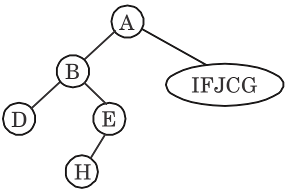
From preorder traversal, we get root node to be *A*.



Now considering left subtree.
Observing both the traversal we can get *B* as root node and *D* as left child and *HE* as a right subtree.



Now observing the preorder traversal we get *E* as a root node and *H* as a left child.



Repeating the above process with the right subtree of root node *A*, we finally obtain the required tree in given Fig. 5.12.1.

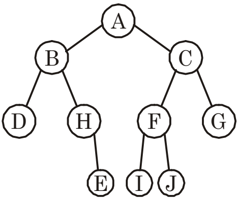


Fig. 5.12.1.

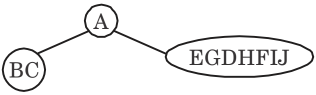
Que 5.13. **Draw a binary tree with following traversals :**

Inorder : ***B C A E G D H F I J***

Preorder : ***A B C D E F G H I J*** **AKTU 2017-18, Marks 07**

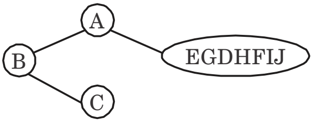
Answer

From preorder traversal, we get root node to be *A*.



Now considering left subtree.

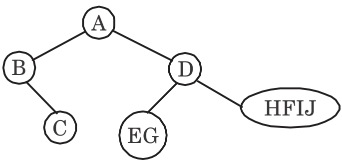
Observing both the traversal we can get *B* as root node and *C* as right child.



Now, consider the right subtree.

Preorder traversal is *DEGFHIJ*, which shows *D* is root node.

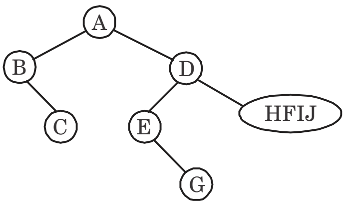
Inorder traversal is *EGDHFIJ*, which shows *EG* is left subtree and *HFIJ* is right subtree.



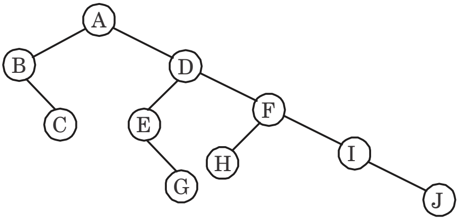
Now, consider the left subtree of *D*.

Preorder traversal is *EG* and inorder traversal is *EG*.

∴ *E* is root node and *G* is right subtree.



Similarly, following the same procedure, we finally get



PART-4

Operation of Insertion, Deletion, Searching and Modification of Data in Binary Search.

Questions-Answers

Long Answer Type and Medium Answer Type Questions

Que 5.14. Write a procedure to insert a new element in a binary search tree.

Answer

INSBST(INFO, LEFT, RIGHT, ROOT, AVAIL, ITEM, LOC)

A binary search tree *T* is in memory and an ITEM of information is given. This algorithm finds the location LOC of ITEM in *T* or adds ITEM as a new node in *T* at location LOC.

1. Call FIND(INFO, LEFT, RIGHT, ROOT, ITEM, LOC, PAR).
2. If LOC ≠ NULL, then Exit.
3. [Copy ITEM into new node in AVAIL list.]
   a. If AVAIL = NULL, then write OVERFLOW, and Exit.
   b. Set NEW := AVAIL, AVAIL := LEFT[AVAIL] and INFO[NEW] := ITEM.
   c. Set LOC := NEW, LEFT[NEW] := NULL and RIGHT[NEW] := NULL.
4. [Add ITEM to tree.]
   If PAR = NULL, then :
      Set ROOT := NEW.
   Else if ITEM < INFO[PAR], then:
      Set LEFT[PAR] := NEW.
   Else :
      Set RIGHT[PAR] := NEW
   [End of If structure]

5. Exit.

Que 5.15. **Write the algorithm for deletion of an element in binary search tree.** **AKTU 2018-19, Marks 07**

Answer

DEL(INFO, LEFT, RIGHT, ROOT, AVAIL, ITEM)
A binary search tree T is in memory, and an ITEM of information is given. This algorithm deletes ITEM from the tree.

1. [Find the locations of ITEM and its parent]
   Call FIND(INFO, LEFT, RIGHT, ROOT, ITEM, LOC, PAR)
2. [ITEM in tree ?]
   If LOC = NULL, then write ITEM not in tree, and Exit.
3. [Delete node containing ITEM]
   If RIGHT[LOC] ≠ NULL and LEFT[LOC] ≠ NULL, then :
   Call CASEB(INFO, LEFT, RIGHT, ROOT, LOC, PAR)
   Else :
   Call CASEA(INFO, LEFT, RIGHT, ROOT, LOC, PAR)
   [End of If structure]
4. [Return deleted node to the AVAIL list]
   Set LEFT[LOC] := AVAIL and AVAIL := LOC
5. Exit.

Que 5.16. **Write a procedure to delete an element from binary search tree where node does not have two children.**

Answer

CASEA(INFO, LEFT, RIGHT, ROOT, LOC, PAR)
This procedure deletes the node N at location LOC, where N does not have two children. The pointer PAR gives the location of the parent of N, or else PAR = NULL indicates that N is the root node. The pointer CHILD gives the location of the only child of N, or else CHILD = NULL indicates N has no children.

1. [Initializes CHILD]
   If LEFT[LOC] = NULL and RIGHT[LOC] = NULL, then:
   Set CHILD := NULL.
   Else if LEFT(LOC) ≠ NULL, then :
   Set CHILD := LEFT[LOC].
   Else
   Set CHILD := RIGHT[LOC].
   [End of If structure.]
2. If PAR ≠ NULL, then :
   If LOC = LEFT[PAR], then :
      Set LEFT[PAR] := CHILD.
   Else :
      Set RIGHT[PAR] := CHILD.

[End of If structure.]
Else :
Set ROOT := CHILD.
[End of If structure.]
3. Return.

Que 5.17. **Write procedure to delete an element from binary search tree where node has two children.**

Answer

CASEB(INFO, LEFT, RIGHT, ROOT, LOC, PAR)

This procedure will delete the node N at location LOC, where N has two children. The pointer PAR gives the location of the parent of N, or else PAR = NULL indicates that N is the root node. The pointer SUC gives the location of the inorder successor of N, and PARSUC gives the location of the parent of the inorder successor.

1. [Find SUC and PARSUC]
   a. Set PTR := RIGHT[LOC] and SAVE := LOC.
   b. Repeat while LEFT[PTR] ≠ NULL:
      Set SAVE := PTR and PTR := LEFT[PTR].
      [End of loop.]
   c. Set SUC := PTR and PARSUC := SAVE.
2. [Delete inorder successor]
   Call CASEA(INFO, LEFT, RIGHT, ROOT, SUC, PARSUC).
3. [Replace node N by its inorder successor.]
a. If PAR ≠ NULL, then:
   If LOC = LEFT[PAR], then:
      Set LEFT[PAR] := SUC.
   Else :
      Set RIGHT[PAR] := SUC
   [End of If structure.]
   Else :
      Set ROOT := SUC.
   [End of If structure.]
b. Set LEFT[SUC] := LEFT[LOC] and
   RIGHT[SUC] := RIGHT[LOC].
4. Return.

Que 5.18. **Write a procedure to search an element in the binary search tree.**

Answer

FIND(INFO, LEFT, RIGHT, ROOT, ITEM, LOC, PAR)

A binary search tree T is the memory and an ITEM of information is given. This procedure finds the location LOC of ITEM in T and also the location PAR of the parent of ITEM. There are three special cases :

i. LOC = NULL and PAR = NULL will indicate that the tree is empty.
ii. LOC ≠ NULL and PAR = NULL will indicate that ITEM is the root of T.
iii. LOC = NULL and PAR ≠ NULL will indicate that ITEM is not in T and can be added to T as a child of the node N with location PAR.

1. [Tree empty ?]
   If ROOT = NULL, then: Set LOC := NULL and PAR := NULL, and Return.
2. [ITEM at root ?]
   If ITEM = INFO[ROOT], then: Set LOC := ROOT and PAR := NULL, and Return.
3. [Initialize pointers PTR and SAVE.]
   If ITEM < INFO[ROOT], then:
      Set PTR := LEFT[ROOT] and SAVE := ROOT.
   Else :
      Set PTR := RIGHT[ROOT] and SAVE := ROOT.
   [End of If structure.]
4. Repeart steps 5 and 6 while PTR ≠ NULL:
5. [ITEM found ?]
   If ITEM = INFO[PTR], then: Set LOC := PTR and PAR := SAVE, and Return.
6. If ITEM < INFO[PTR], then:
      Set SAVE := PTR and PTR := LEFT[PTR].
   Else :
      Set SAVE := PTR and PTR := RIGHT[PTR].
   [End of If structure]
   [End of step 4 loop.]
7. [Search unsuccessful.] Set LOC := NULL and PAR := SAVE.
8. Exit.

PART-5

Threaded Binary Trees, Traversing Threaded Binary Trees, Huffman Coding using Binary Tree.

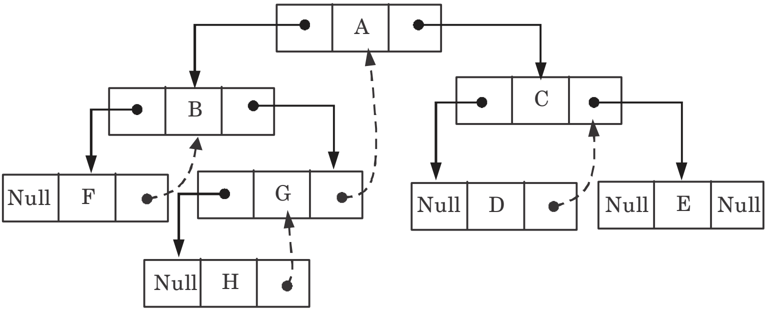
Questions-Answers

Long Answer Type and Medium Answer Type Questions

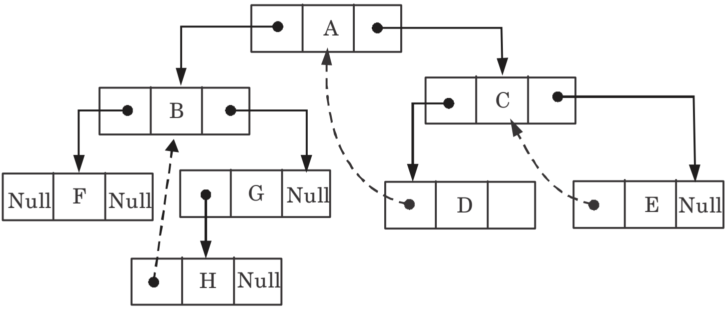
Que 5.19. **What is a threaded binary tree ? Explain the advantages of using a threaded binary tree.** **AKTU 2017-18, Marks 07**

Answer

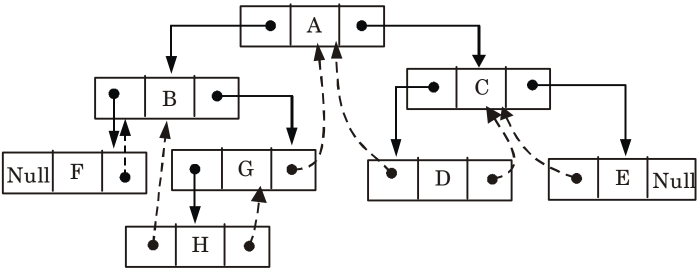
Threaded binary tree is a binary tree in which all left child pointers that are NULL points to its inorder predecessor and all right child pointers that are NULL points to its inorder successor.



(a) Right threaded binary tree.



(b) Left threaded binary tree



(c) Fully threaded binary tree

Fig. 5.19.1.

Advantages of using threaded binary tree :

1. In threaded binary tree the traversal operations are very fast.

2. In threaded binary tree, we do not require stack to determine the predecessor and successor node.
3. In a threaded binary tree, one can move in any direction *i.e.*, upward or downward because nodes are circularly linked.
4. Insertion into and deletions from a threaded tree are all although time consuming operations but these are very easy to implement.

Que 5.20. Write algorithm/function for inorder traversal of threaded binary tree.

Answer

Algorithm for inorder traversal in threaded binary tree :

1. Initialize current as root
2. While current is not NULL
   If current does not have left child
   a. Print current's data
   b. Go to the right, *i.e.*, current = current->right
   Else
   a. Make current as right child of the rightmost node in current's left subtree
   b. Go to this left child, *i.e.*, current = current->left

Que 5.21. What is Huffman tree ? Create a Huffman tree with following numbers :

24, 55, 13, 67, 88, 36, 17, 61, 24, 76 **AKTU 2014-15, Marks 10**

Answer

Huffman tree is a binary tree in which each node in the tree represents a symbol and each leaf represent a symbol of original alphabet.

Huffman algorithm :

1. Suppose, there are n weights W_1, W_2, ..., W_n.
2. Take two minimum weights among the n given weights. Suppose W_1 and W_2 are first two minimum weights then subtree will be :

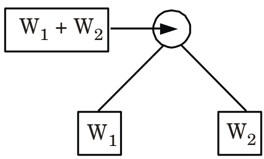


Fig. 5.21.1.

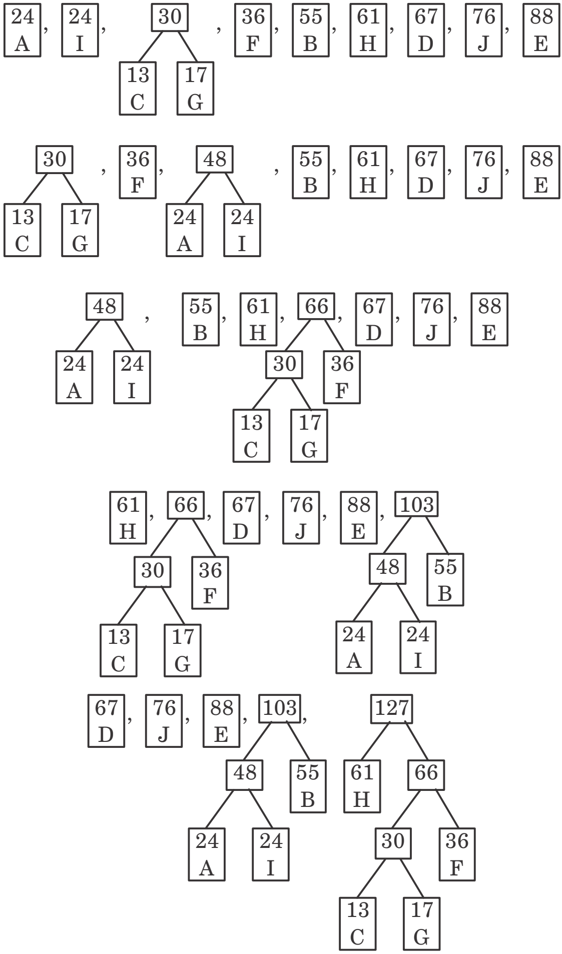
3. Now the remaining weights will be $W_1 + W_2$, W_3, W_4, ..., W_n.
4. Create all subtree at the last weight.

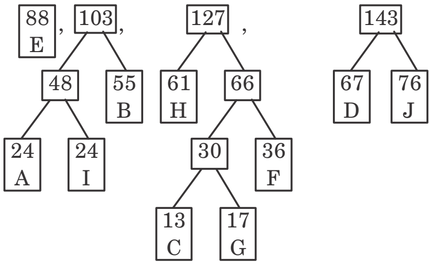
Numerical :

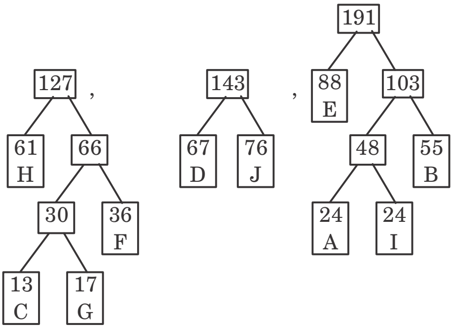
24	55	13	67	88	36	17	61	24	76
A	B	C	D	E	F	G	H	I	J

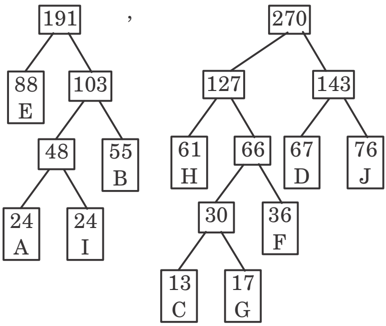
Arrange all the numbers in ascending order :

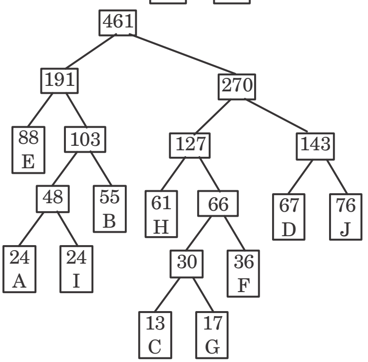
13	17	24	24	36	55	61	67	76	88
C	G	A	I	F	B	H	D	J	E




88 E
103
48
55 B
24 A
24 I
127
61 H
66
30
36 F
13 C
17 G
143
67 D
76 J


127
61 H
66
30
36 F
13 C
17 G
143
67 D
76 J
191
88 E
103
48
55 B
24 A
24 I


191
88 E
103
48
55 B
24 A
24 I
270
127
143
61 H
66
67 D
76 J
30
36 F
13 C
17 G


461
191
270
88 E
103
127
143
48
55 B
61 H
66
67 D
76 J
24 A
24 I
30
36 F
13 C
17 G

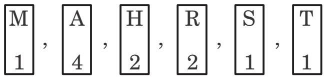
Que 5.22. **Explain Huffman algorithm. Construct Huffman tree for *MAHARASHTRA* with its optimal code.**

AKTU 2018-19, Marks 07

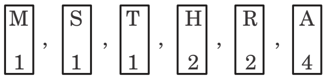
Answer

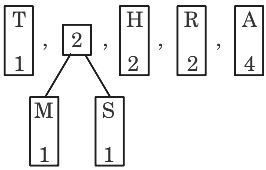
Huffman algorithm : Refer Q. 5.21, Page 5–24A, Unit-5.

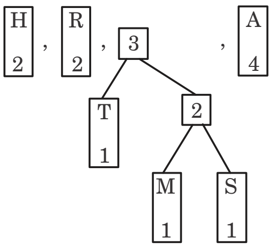
Numerical :

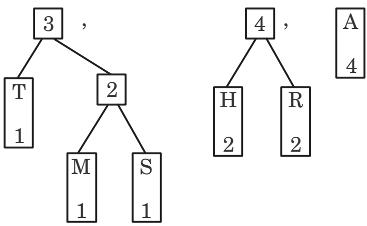


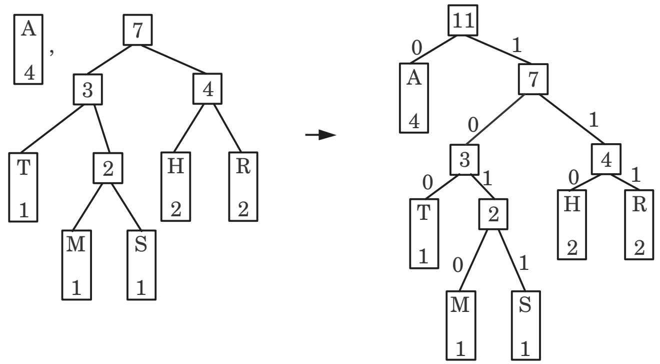
Arrange all the number in ascending order.











Character	Code
M	1010
A	0
H	110
R	111
S	1011
T	100

Optimal code for *MAHARASHTRA* is :

1010011001110101111101001110

PART-6

Concept and Basic Operation for AVL Tree, B tree and Binary Heaps.

Questions-Answers

Long Answer Type and Medium Answer Type Questions

Que 5.23. **Define AVL trees. Explain its rotation operations with example. Construct an AVL tree with the values 10 to 1 numbers into an initially empty tree.** **AKTU 2016-17, Marks 15**

Answer

i. An AVL (or height balanced) tree is a balanced binary search tree.

ii. In an AVL tree, balance factor of every node is either –1, 0 or +1.

iii. Balance factor of a node is the difference between the heights of left and right subtrees of that node.

Balance factor = height of left subtree – height of right subtree

iv. In order to balance a tree, there are four cases of rotations :

1. **Left Left rotation (LL rotation) :** In LL rotation every node moves one position to left from the current position.

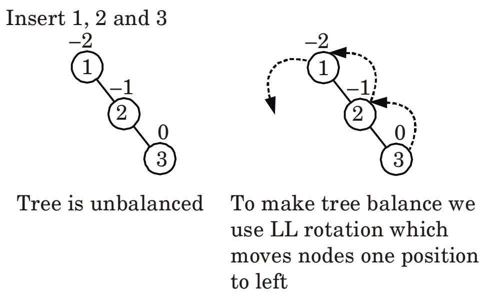


Fig. 5.23.1.

2. **Right Right rotation (RR rotation) :** In RR rotation every node moves one position to right from the current position.

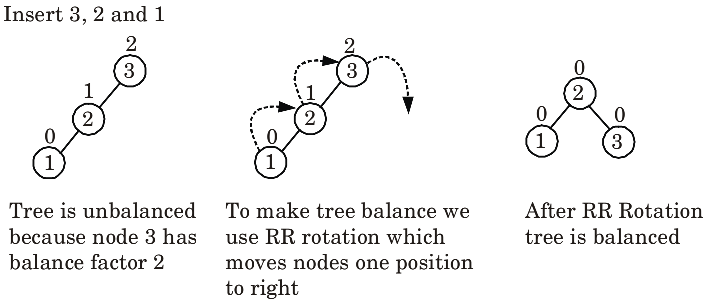


Fig. 5.23.2.

3. **Left Right rotation (LR rotation) :** The LR Rotation is combination of single left rotation followed by single right rotation. In LR rotation, first every node moves one position to left then one position to right from the current position.

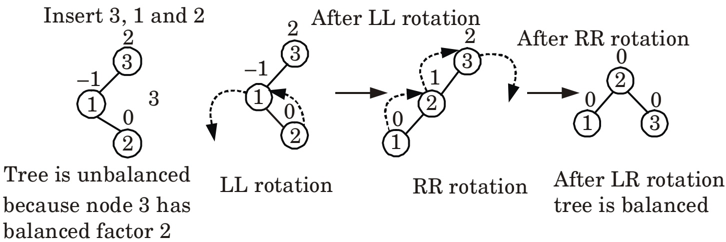


Fig. 5.23.3.

4. **Right Left rotation (RL rotation) :** The RL rotation is the combination of single right rotation followed by single left rotation. In RL rotation, first every node moves one position to right then one position to left from the current position.

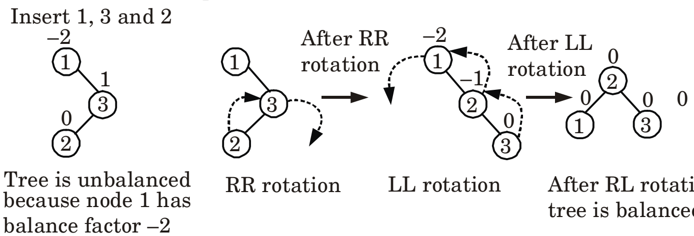
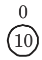


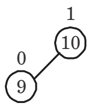
Fig. 5.23.4.

Numerical :

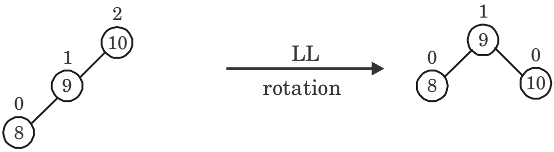
Insert 10 :



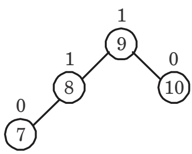
Insert 9 :



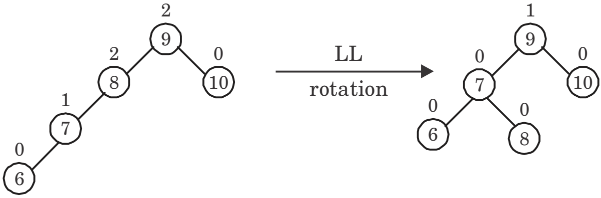
Insert 8 :



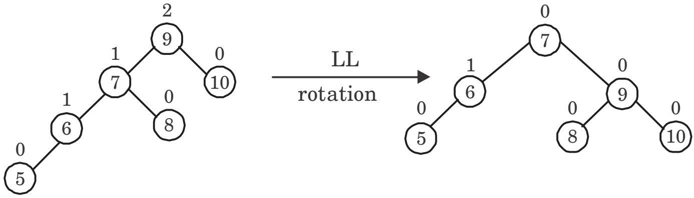
Insert 7 :



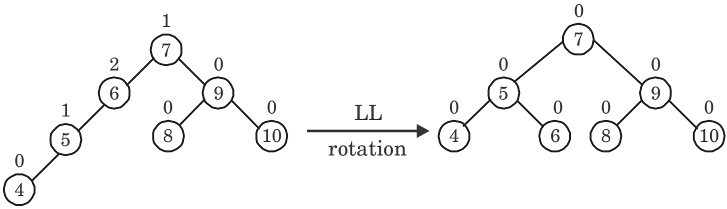
Insert 6 :



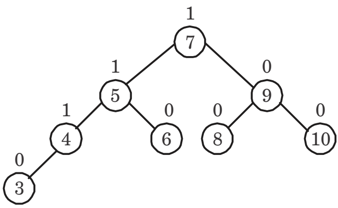
Insert 5 :



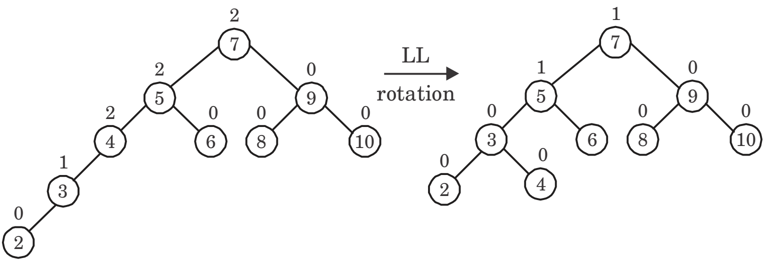
Insert 4 :



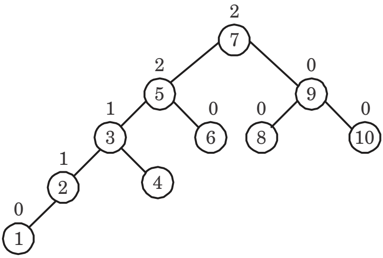
Insert 3 :

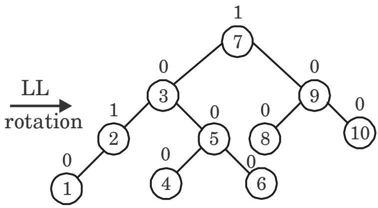


Insert 2 :



Insert 1 :





Que 5.24. **Consider the following AVL tree and insert 2, 12, 7 and 10 as new node. Show proper rotation to maintain the tree as AVL.**

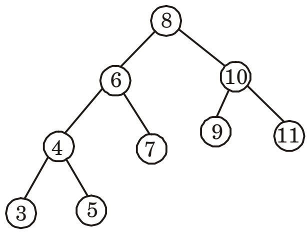
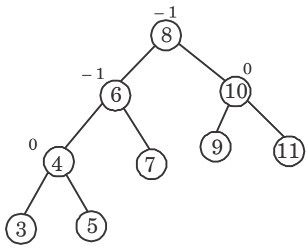


Fig. 5.24.1.

AKTU 2017-18, Marks 07

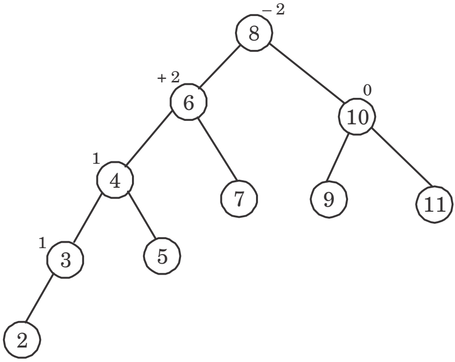
Answer

Given tree :

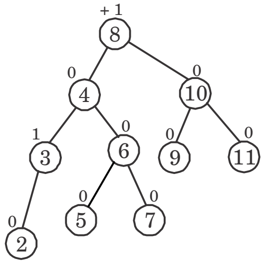


Balanced tree

Insert 2 :

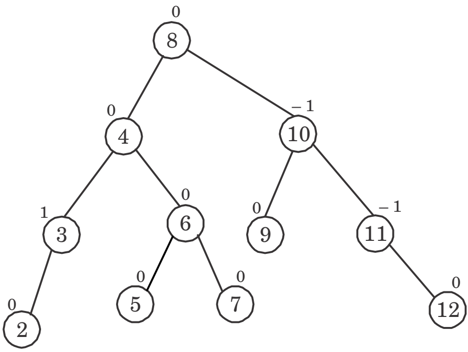


Tree is unbalanced, now LL rotation is required to balance it.



Now the tree is balanced.

Insert 12 :



Tree is balanced, so there is no need to balance the tree.

Insert 7 : 7 is already in the tree hence it cannot be inserted in the AVL tree.

Insert 10 : 10 is also in the tree hence it cannot be inserted in the AVL tree.

Que 5.25. What is height balanced tree ? Why height balancing of tree is required ? Create an AVL tree for the following elements : *a*, *z*, *b*, *y*, *c*, *x*, *d*, *w*, *e*, *v*, *f*.

AKTU 2018-19, Marks 07

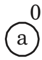
Answer

Height balanced tree : Refer Q. 5.23, Page 5–28A, Unit-5.

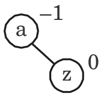
Height balancing of tree is required : Height balancing of tree is required to implement an AVL tree. Each node must contain a balance factor, which indicates its states of balance relative to its sub-tree.

Numerical :

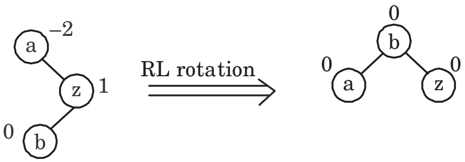
Insert *a* :



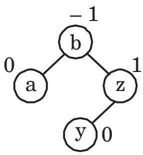
Insert *z* :



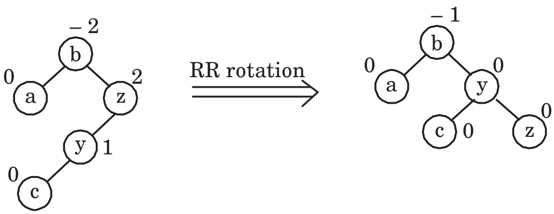
Insert *b* :



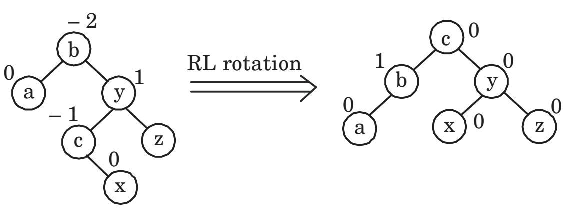
Insert *y* :



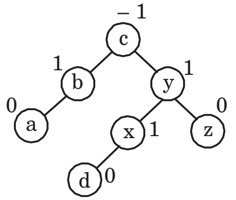
Insert *c* :



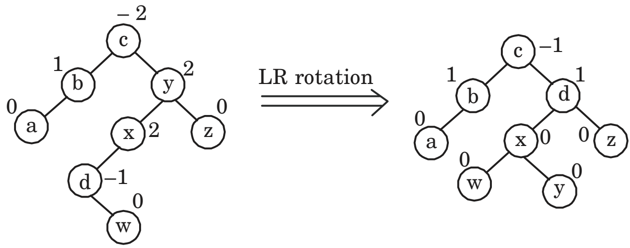
Insert *x* :



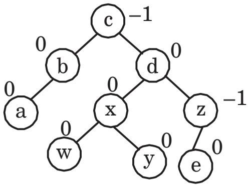
Insert *d* :



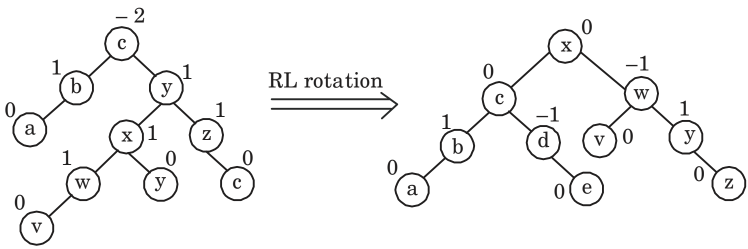
Insert *w* :



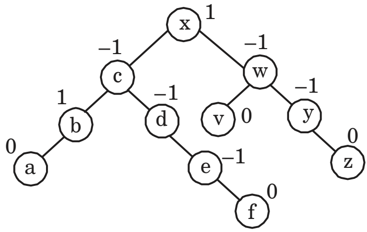
Insert *e* :



Insert *v* :



Insert *f* :



No, rebalancing required. So, this is final AVL search tree.

Que 5.26. **Construct a height balanced binary search tree by performing following operations :**
Step 1 : Insert
19, 16, 21, 11, 17, 25, 6, 13
Step 2 : Insert
3
Step 3 : Delete
16

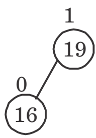
AKTU 2014-15, Marks 10

Answer

Step 1 : Insert 19, 16, 21, 11, 17, 25, 6, 13 :

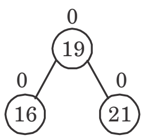
Insert 19 :

Insert 16 :

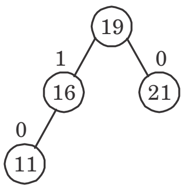


1

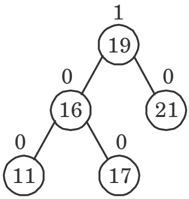
Insert 21 :



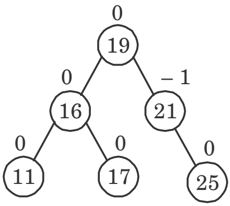
Insert 11 :



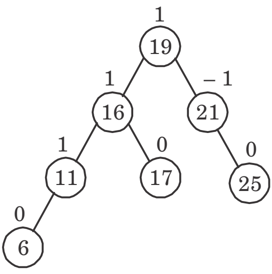
Insert 17 :



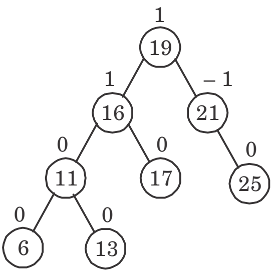
Insert 25 :



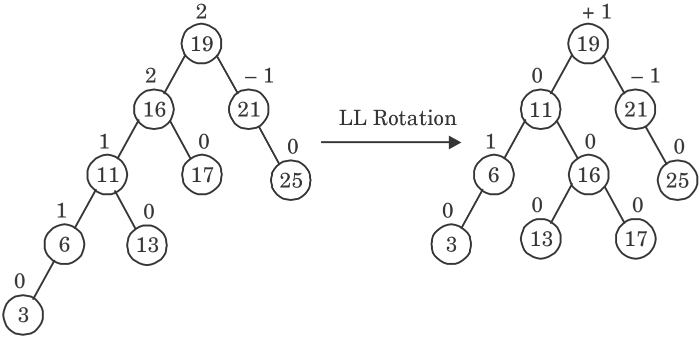
Insert 6 :



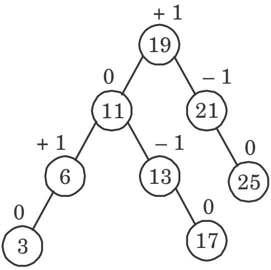
Insert 13 :



Step 2 : Insert 3 :



Step 3 : Delete 16 :



Que 5.27. **Describe all rotations in AVL tree. Construct AVL tree from the following nodes : *B, C, G, E, F, D, A*.**

AKTU 2015-16, Marks 10

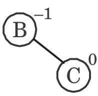
Answer

AVL rotations : Refer Q. 5.23, Page 5–28A, Unit-5.

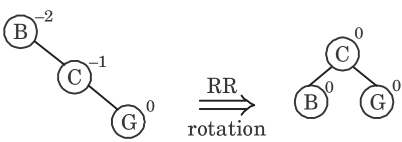
Construction of AVL tree : B, C, G, E, F, D, A

Insert B : 

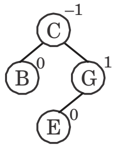
Insert C :



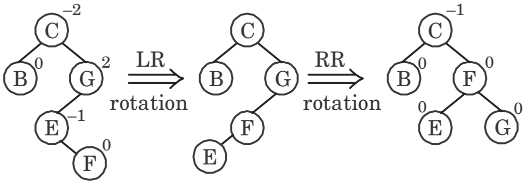
Insert G :



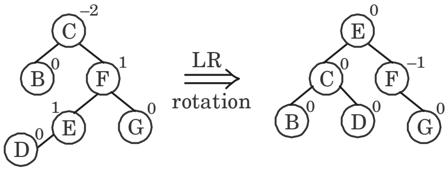
Insert E :



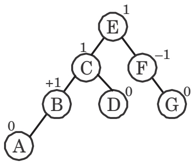
Insert F :



Insert D :



Insert A :



Que 5.28. **Define a B-tree. What are the applications of B-tree ? Draw a B-tree of order 4 by insertion of the following keys in order : *Z, U, A, I, W, L, P, X, C, J, D, M, T, B, Q, E, H, S, K, N, R, G, Y, F, O, V.***

AKTU 2015-16, Marks 15

OR

Write a short notes on B-tree. **AKTU 2014-15, Marks 05**

Answer

B-tree :

1. A B-tree is a self-balancing tree data structure that keeps data sorted and allows searches, sequential access, insertions, and deletions in logarithmic time.
2. A B-tree of order m is a tree which satisfies the following properties :
a. Every node has at most m children.
b. Every non-leaf node (except root) has at least $\lceil m/2 \rceil$ children.
c. The root has at least two children if it is not a leaf node.

d. A non-leaf node with k children contains $k - 1$ keys.
e. All leaves appear in the same level.

Application of B-tree : The main application of a B-tree is the organization of a huge collection of records into a file structure. The organization should be in such a way that any record in it can be searched very efficiently *i.e.*, insertion, deletion and modification operations can be carried out perfectly and efficiently.

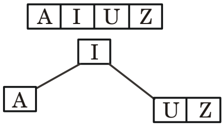
Construction of B-tree :

Insert Z :

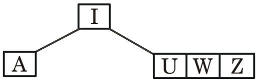
Insert U :

Insert A :

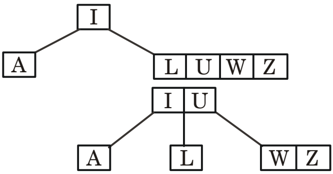
Insert I :



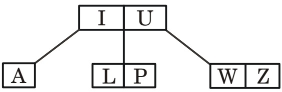
Insert W :



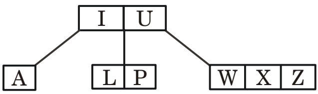
Insert L :



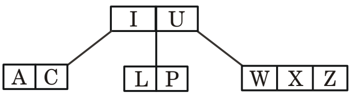
Insert P :



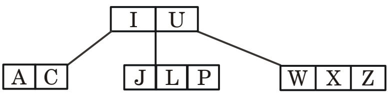
Insert X :



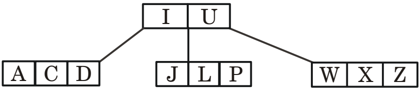
Insert C :



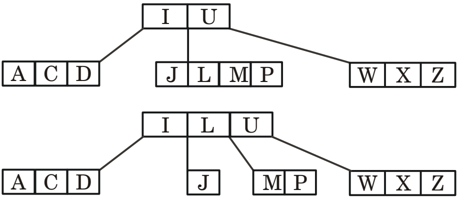
Insert J :



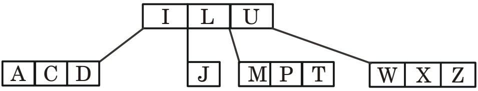
Insert D :



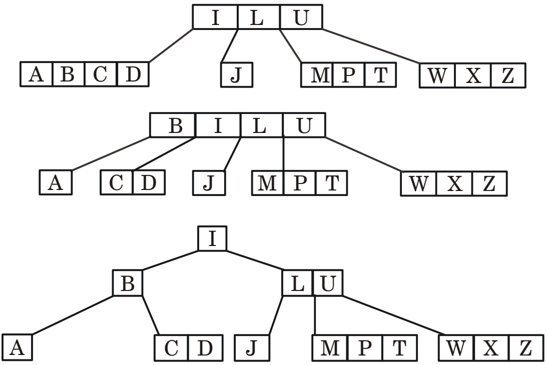
Insert M :



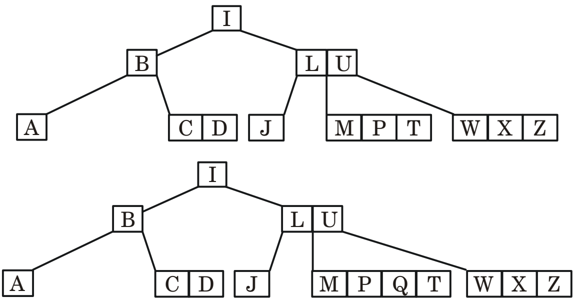
Insert T :



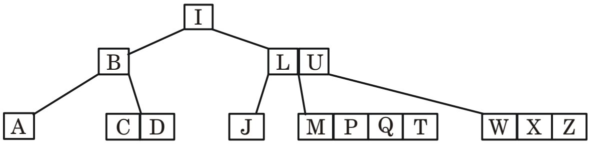
Insert B :



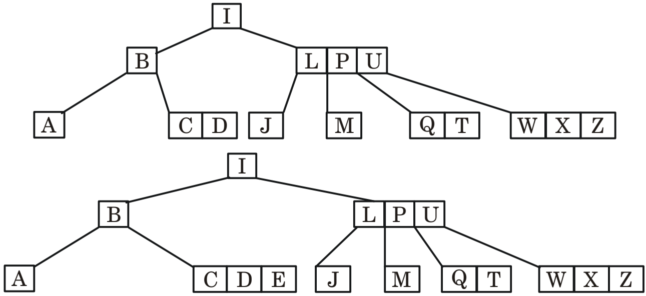
Insert Q :



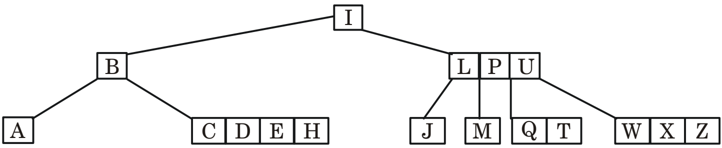
Insert E :



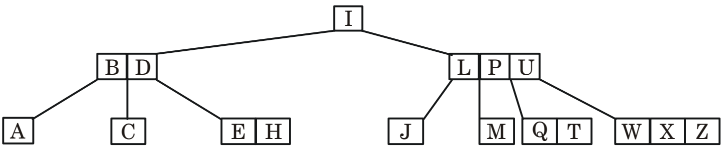
Insert H :



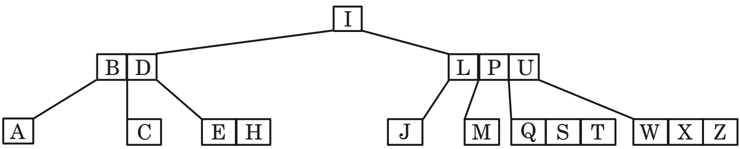
Insert S :

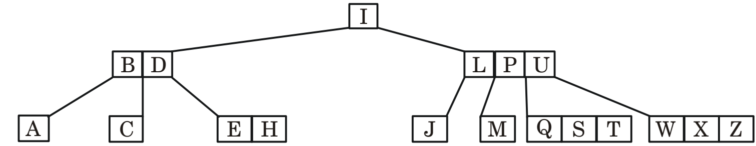


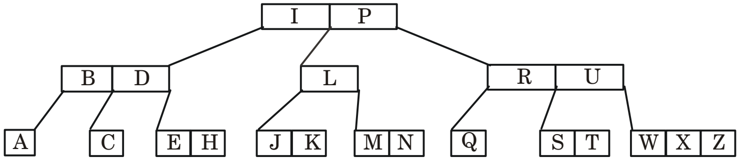
Insert K :



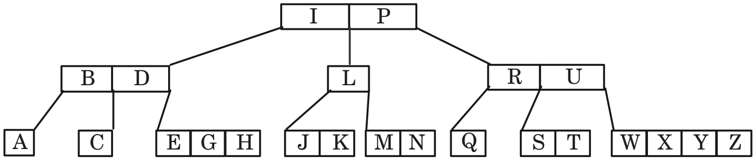
Insert N, R :

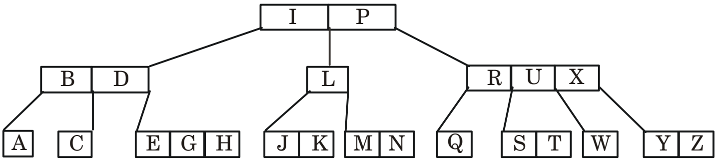




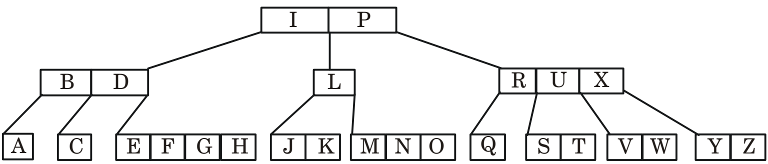


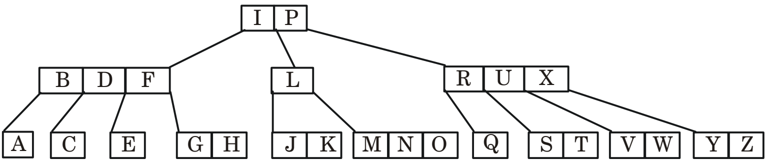
Insert G, Y :





Insert F, O & V :





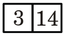
Que 5.29. **Construct a B-tree of order 5 created by inserting the following elements 3, 14, 7, 1, 8, 5, 11, 17, 13, 6, 23, 12, 20, 26, 4, 16, 18, 24, 25, 19. Also delete elements 6, 23 and 3 from the constructed tree.**

AKTU 2018-19, Marks 07

Answer

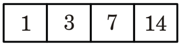
Insert 3 :

Insert 14 :

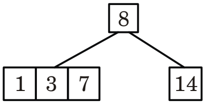


Insert 7 :

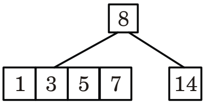
Insert 1 :



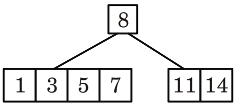
Insert 8 :



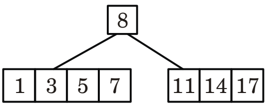
Insert 5 :



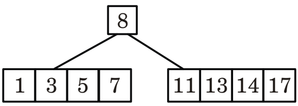
Insert 11 :



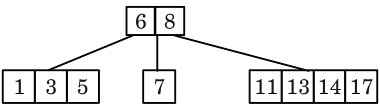
Insert 17 :



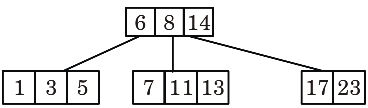
Insert 13 :



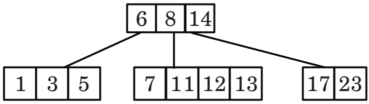
Insert 6 :



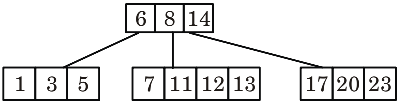
Insert 23 :



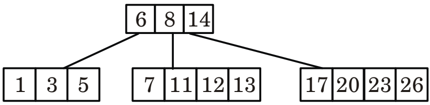
Insert 12 :



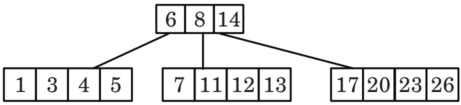
Insert 20 :



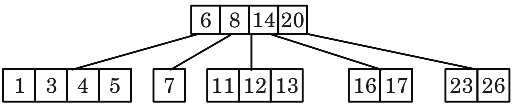
Insert 26 :



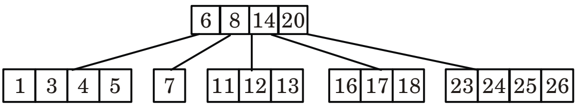
Insert 4 :



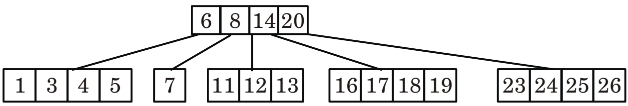
Insert 16 :



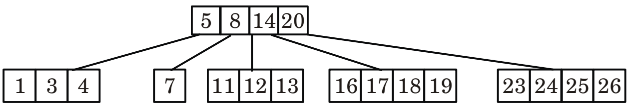
Insert 18, 24, 25 :



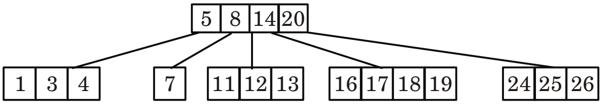
Insert 19 :



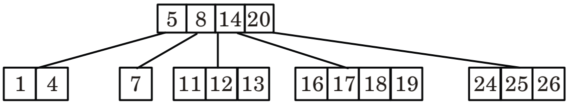
Delete 6 :



Delete 23 :



Delete 3 :



Que 5.30. **Construct a B-tree on following sequence of inputs.**

10, 20, 30, 40, 50, 60, 70, 80, 90

Assume that the order of the B-tree is 3.

AKTU 2017-18, Marks 07

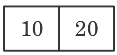
Answer

10, 20, 30, 40, 50, 60, 70, 80, 90

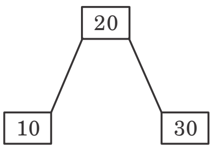
Order of the *B*-tree is 3.

1. **Insert 10 :**

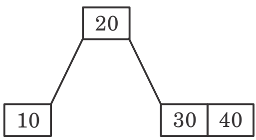
2. **Insert 20 :**



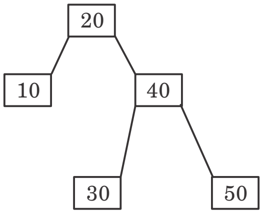
3. **Insert 30 :**



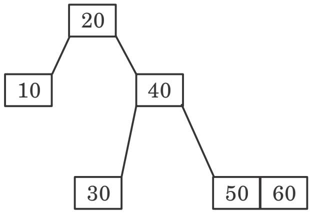
4. **Insert 40 :**



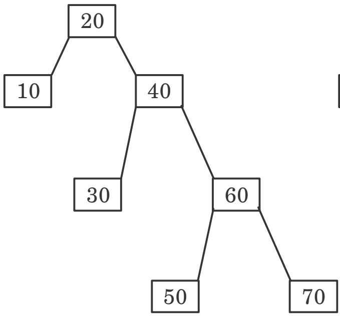
5. Insert 50 :



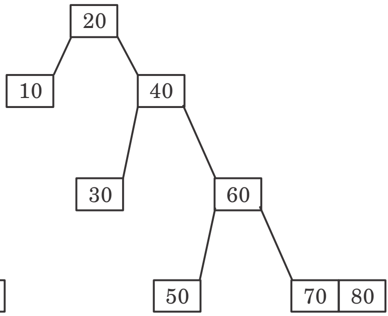
6. Insert 60 :



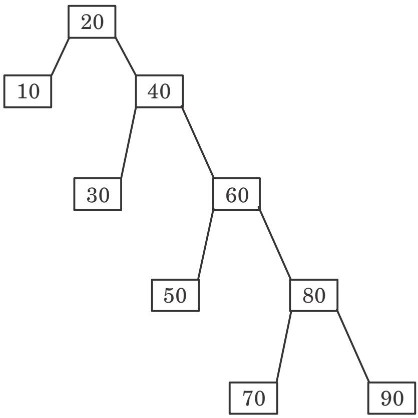
7. Insert 70 :



8. Insert 80 :



9. Insert 90 :



This is final B-tree of order 3.

Que 5.31. **Compare and contrast the difference between B+ tree index files and B-tree index files with an example.**

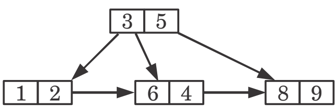
AKTU 2016-17, Marks 10

Answer

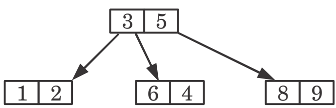
S. No.	Basis	B^+ tree	B-tree
1.	Definition	B^+ tree is an n-array tree with a variable but often large number of children per node. A B^+ tree consists of a root, internal nodes and leaves. The root may be either a leaf or a node with two or more children.	A B-tree is an organizational structure for information storage and retrieval in the form of a tree in which all terminal nodes are at the same distance from the base, and all non-terminal nodes have between n and $2n$ sub-trees or pointers (where n is an integer).
2.	Space complexity	O(n)	O(n)
3.	Storage	In a B^+ tree, data is stored only in leaf nodes.	In a B-tree, search keys and data are stored in internal or leaf nodes.
4.	Data	The leaf nodes of the tree store the actual record rather than pointers to records.	The leaf nodes of the tree store pointers to records rather than actual records.
5.	Space	These trees do not waste space.	These trees waste space.
6.	Function of leaf nodes	In B^+ tree, leaf node data are ordered in a sequential linked list.	In B-tree, the leaf node cannot store using linked list.
7.	Searching	In B^+ tree, searching of any data is very easy because all data is found in leaf nodes.	In B-tree, searching becomes difficult as data cannot be found in the leaf node.
8.	Search accessibility	In B^+ tree, the searching becomes easy.	In B-tree, the search is not that easy as compared to a B^+ tree.
9.	Redundant key	They store redundant search key.	They do not store redundant search key.

Example :

B^+ tree :



B-tree :



Que 5.32. **Write a short note on binary heaps.**

Answer

1. The binary heap data structure is an array that can be viewed as a complete binary tree.
2. Each node of the binary tree corresponds to an element of the array.
3. The array is completely filled on all levels except possibly lowest.
4. We represent heaps in level order, going from left to right.
5. If an array A contains key values of nodes in a heap, length [*A*] is the total number of elements.

   Heap-size [*A*] = Length [*A*] = Number of elements.
6. The root of the tree A[1] and given index i of a node the indices of its parent, left child and right child can be computed :

```
PARENT (i)
return floor (i/2)
LEFT(i)
return 2i
RIGHT (i)
return 2i + 1
```

VERY IMPORTANT QUESTIONS

Following questions are very important. These questions may be asked in your SESSIONALS as well as UNIVERSITY EXAMINATION.

Q. 1. Write a C program to implement binary tree insertion, deletion with example.

Ans. Refer Q. 5.5.

Q. 2. Write the C program for various traversing techniques of binary tree with neat example.

Ans. Refer Q. 5.6.

Q. 3. Explain binary search tree and its operations. Make a binary search tree for the following sequence of numbers, show all steps : 45, 32, 90, 34, 68, 72, 15, 24, 30, 66, 11, 50, 10.

Ans. Refer Q. 5.7.

Q. 4. Define binary search tree. Create BST for the following data, show all steps :
20, 10, 25, 5, 15, 22, 30, 3, 14, 13

Ans. Refer Q. 5.8.

Q. 5. Define tree, binary tree, complete binary tree and full binary tree. Write algorithm or function to obtain traversals of a binary tree in preorder, postorder and inorder.

Ans. Refer Q. 5.10.

Q. 6. Construct a binary tree for the following :
Inorder : *Q, B, K, C, F, A, G, P, E, D, H, R*
Preorder : *G, B, Q, A, C, K, F, P, D, E, R, H*
Find the postorder of the tree.

Ans. Refer Q. 5.11.

Q. 7. Draw a binary tree with following traversal :
Inorder : *D B H E A I F J C G*
Preorder : *A B D E H C F I J G*

Ans. Refer Q. 5.12.

Q. 8. Draw a binary tree with following traversals :
Inorder : *B C A E G D H F I J*
Preorder : *A B C D E F G H I J*

Ans. Refer Q. 5.13.

Q. 9. What is a threaded binary tree ? Explain the advantages of using a threaded binary tree.

Ans. Refer Q. 5.19.

Q. 10. What is Huffman tree ? Create a Huffman tree with following numbers :
24, 55, 13, 67, 88, 36, 17, 61, 24, 76

Ans. Refer Q. 5.21.

Q. 11. Explain Huffman algorithm. Construct Huffman tree for *MAHARASHTRA* with its optimal code.

Ans. Refer Q. 5.22.

Q. 12. Define AVL trees. Explain its rotation operations with example. Construct an AVL tree with the values 10 to 1 numbers into an initially empty tree.

Ans. Refer Q. 5.23.

Q. 13. Consider the following AVL tree and insert 2, 12, 7 and 10 as new node. Show proper rotation to maintain the tree as AVL.

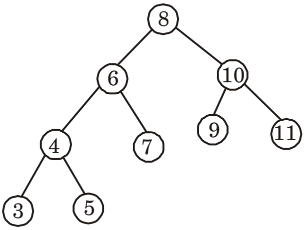


Fig. 5.24.1.

Ans. Refer Q. 5.24.

Q. 14. Construct a height balanced binary search tree by performing following operations :
Step 1 : Insert
19, 16, 21, 11, 17, 25, 6, 13
Step 2 : Insert
3
Step 3 : Delete
16

Ans. Refer Q. 5.26.

Q. 15. What is height balanced tree ? Why height balancing of tree is required ? Create an AVL tree for the following elements : *a, z, b, y, c, x, d, w, e, v, f*.

Ans. Refer Q. 5.25.

Q. 16. Describe all rotations in AVL tree. Construct AVL tree from the following nodes : *B, C, G, E, F, D, A*.

Ans. Refer Q. 5.27.

Q. 17. Define a B-tree. What are the applications of B-tree ? Draw a B-tree of order 4 by insertion of the following keys in order : *Z, U, A, I, W, L, P, X, C, J, D, M, T, B, Q, E, H, S, K, N, R, G, Y, F, O, V*.

Ans. Refer Q. 5.28.

Q. 18. Construct a B-tree of order 5 created by inserting the following elements 3, 14, 7, 1, 8, 5, 11, 17, 13, 6, 23, 12, 20, 26, 4, 16, 18, 24, 25, 19. Also delete elements 6, 23 and 3 from the constructed tree.

Ans. Refer Q. 5.29.

Q. 19. Construct a B-tree on following sequence of inputs.
10, 20, 30, 40, 50, 60, 70, 80, 90
Assume that the order of the B-tree is 3.

Ans. Refer Q. 5.30.

☺☺☺

1 UNIT

Array and Linked List (2 Marks Questions)

1.1. Define the term data structure. List some linear and non-linear data structures stating the application area where they will be used. **AKTU 2017-18, Marks 02**

Ans. It is a particular way of storing and organizing data in a computer so that it can be used efficiently.
It can be classified into two types :

i. Linear data structures :
1. Array
2. Stacks
3. Queue
4. Linked list

ii. Non-linear data structures :
1. Tree
2. Graph

1.2. Name few terminologies used in data structure.

Ans. Few terminologies used in data structure are :
1. Data
2. Entity
3. Field
4. Record
5. File

1.3. What are the data types used in C ?

Ans. Data type used in C are :
1. Primitive data types
2. Non-primitive data types

1.4. Name some primitive data types.

Ans. Primitive data types are :
1. Integer data type
2. Floating point data type
3. Character data type
4. Void data type

1.5. Define an algorithm.

Ans. An algorithm is a step-by-step finite sequence of instruction, to solve a well-defined computational problem.

1.6. Give the criteria that an algorithm must satisfy.

Ans. Every algorithm must satisfy the following criteria :

1. Input
2. Output
3. Definiteness
4. Effectiveness
5. Finiteness

1.7. What are the characteristics of an algorithm ?

Ans. Characteristics of an algorithm are :

1. It should be free from ambiguity.
2. It should be concise.
3. It should be efficient.

1.8. What are the different ways of analyzing an algorithm ?

Ans. Different ways of analyzing an algorithm :

1. Worst case running time
2. Average case running time
3. Best case running time

1.9. Define complexity.

Ans. The complexity of an algorithm M is the function $f(n)$ which gives the running time and/or storage space requirement of the algorithm in terms of the size n of the input data.

1.10. Define time complexity and space complexity of an algorithm. **AKTU 2016-17, Marks 02**

Ans. Time complexity : Time complexity is the amount of time it needs to run to completion.

Space complexity : Space complexity is the amount of memory it needs to run to completion.

1.11. What are the various asymptotic notations ? Explain the Big-Oh notation. **AKTU 2015-16, Marks 02**

Ans. Various asymptotic notations are :

1. Theta notation (θ - notation)
2. Big-Oh (O - notation)
3. Omega notation (Ω - notation)

Big-Oh notation : It is used when there is only an asymptotic upper bound. For a given function $g(n)$, $O(g(n))$ is denoted by a set of functions.

1.12. Define time-space tradeoff.

Ans. The time-space tradeoff refers to a choice between algorithmic solutions of data processing problems that allows to decrease the running time of an algorithmic solution by increasing the space to store data and vice versa.

1.13. Write down the properties of Abstract Data Type (ADT).

Ans. **Properties of Abstract Data Type (ADT) :**

i. It is used to simplify the description of abstract algorithm to classify and evaluate data structure.

ii. It is an important conceptual tool in OOPs and design by contract methodologies for software development.

1.14. Differentiate linear and non-linear data structures.

AKTU 2016-17, Marks 02

Ans.

S. No.	Linear data structure	Non-linear data structure
1.	It is a data structure whose elements form a sequence.	It is a data structure whose elements do not form a sequence.
2.	Every element in the structure has a unique predecessor and unique successor.	There is no unique predecessor or unique successor.
3.	Examples of linear data structure are arrays, linked lists, stacks and queues.	Examples of non-linear data structures are trees and graphs.

1.15. What do you mean by an array ?

Ans. An array is a list of finite number of elements of same data type *i.e.*, integer, real or strings.

1.16. What are the merits and demerits of array data structures ?

AKTU 2016-17, Marks 02

Ans. **Merits of array :**

1. Array is a collection of elements of similar data type.
2. Hence, multiple applications that require multiple data of same data type are represented by a single name.

Demerits of array :

1. Linear arrays are static structures, *i.e.*, memory used by them cannot be reduced or extended.
2. Previous knowledge of number of elements in the array is necessary.

1.17. Define pointer.

Ans. Pointers are variable which can hold the address of another variable. Some of the examples of pointer declarations are :

```
int * ptr1;
float * ptr2;
unsigned int * ptr3;
```

1.18. Differentiate between array and pointer.

Ans.

S. No.	Array	Pointer
1.	Array can be initialized at definition.	Pointer cannot be initialized at definition.
2.	Static in nature.	Dynamic in nature.
3.	It cannot be resized.	It can be resized.

1.19. Differentiate between overflow and underflow condition in a linked list. **AKTU 2018-19, Marks 02**

Ans.

S. No.	Overflow	Underflow
1.	Overflow condition occurs in linked list when data are inserted into a list but there is no available space.	Underflow condition occurs when we delete data from empty linked list.
2.	In linked list overflow occurs when AVAIL = NULL and there is an insertion operation.	In linked list underflow occurs when START = NULL and there is a deletion operation.

1.20. Write a function to reverse the list.

Ans.

```
node * reverse (node * p){
node *q, *r;
q = (node*) NULL;
while (p != NULL)
{
r = q;
q = p;
p = p → next;
p → next = r;
}
return(q);
}
```

1.21. Given a 2D array A [– 100 : 100, – 5 : 50]. Find the address of element A [99, 49] considering the base address 10 and each element requires 4 bytes for storage. Follow row major order.

AKTU 2015-16, Marks 02

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

$$\text{LOC }(A[99][49]) = 10 + 4\ [50\ (99 - (-100) + 49 - (-5)]$$
$$= 10 + 4\ [50\ (199) + 54] = 40026$$

1.22. Explain the application of sparse matrices.

AKTU 2015-16, Marks 02

Ans. There are two applications of sparse matrix which are :

1. **Triangular matrix :** In this, all entries above the main diagonal are zero or, equivalently, where non-zero entries can only occur on or below the main diagonal.
2. **Tridiagonal matrix :** In this, all non-zero entries can occur only on the diagonal or on elements immediately above or below the diagonal.

☺☺☺

2 UNIT

Stacks and Queues (2 Marks Questions)

2.1. What are the applications of stack ?

Ans. Applications of stack are :

i. Infix to postfix conversion.
ii. Implementing function calls.
iii. Page-visited history in a web browser.
iv. Undo sequence in a text editor.

2.2. Mention the limitations of stack using array.

Ans. Limitations of stacks using array :

i. The maximum size of the stack once defined cannot be changed.
ii. Trying to push a new element into a full stack causes an overflow condition.

2.3. What are the notations used in evaluation of arithmetic expressions using prefix and postfix forms ?

AKTU 2015-16, Marks 02

Ans. Notations used in evaluation of arithmetic expressions are :

i. Infix notation : In this notation, the operator symbol is placed between its two operands.
For example : To add A to B we can write as, A + B or B + A

ii. Polish (Prefix) notation : Here the operator symbol is placed before its two operands.
For example : To add A to B we can write as, + AB or + BA

iii. Reverse polish (Postfix) notation : In this notation, the operator symbol is placed after its two operands.
For example : To add A and B we can write as : AB+ or BA+

2.4. If the Tower of Hanoi is operated on *n* = 10 disks, calculate the total number of moves. **AKTU 2015-16, Marks 02**

OR

Calculate total number of moves for Tower of Hanoi for *n* = 10 disks. **AKTU 2017-18, Marks 02**

Ans. For n number of disks, total number of moves $= 2^n - 1$

For 10 disks, *i.e.*, $n = 10$, total number of moves $= 2^{10} - 1$

$= 1024 - 1$

$= 1023$

Therefore, if the Tower of Hanoi is operated on $n = 10$ disks, then total number of moves are 1023.

2.5. Give the infix, postfix and prefix notation of (A + B) + C.

Ans. **Infix notation :** (A + B) + C
Postfix notation : (AB) ++ C = AB + C +
Prefix notation : + AB + C = ++ ABC

2.6. How do you push elements in a linked stack ?

AKTU 2016-17, Marks 02

Ans. To insert an element onto stack is known as PUSH operation. Before inserting first we increase the top pointer and then insert the element.

2.7. Differentiate between iteration and recursion.

Ans.

S. No.	Iteration	Recursion
i.	It is a process of executing statement until some specified condition is satisfied.	It is a technique of defining anything in terms of itself.
ii.	Iterative counterpart of a problem is more efficient in term of memory utilization and execution speed.	It is a worse option to go for simple problems.

2.8. Discuss the steps for converting an infix expression to postfix expression.

Ans. **Steps for converting an infix expression to postfix expression :**

i. Parenthesize the expression starting from left to right.
ii. During parenthesizing the expression, the operands associated with operator having higher precedence are first parenthesized.
iii. Once the expression is converted to postfix then remove the parenthesis.

2.9. Write some applications of queue.

Ans. **Application of queues are :**

i. Operating systems schedule jobs in the order of arrival.
ii. Simulation of real world queues such as lines at a ticket counter.
iii. Multiprogramming.
iv. Waiting times for customers at call center.

2.10. Translate infix expression into its equivalent postfix expression : A * (B + D)/E – F * (G + H/K).

AKTU 2015-16, Marks 02

Ans. Infix expression : A * (B + D)/E – F * (G + H/K)
A * (BD +)/E – F * (G + HK/)

A (BD +)*/E – F* (GHK/+)
(ABD + * E/) – (FGHK/+*)
ABD + * E/FGHK / + * –
Equivalent postfix expression is :
ABD + * E/FGHK / + * –

2.11. Convert the following arithmetic infix expression into its equivalent postfix expression.

Expression : A – B/C + D*E + F **AKTU 2017-18, Marks 02**

Ans. (A – B/C + D*E + F)

Character	Stack	Postfix
(	(	
A	(	A
–	(–	A
B	(–	AB
/	(– /	AB
C	(– /	ABC
+	(– +	ABC /
D	(– +	ABC / D
*	(– + *	ABC / D
E	(– + *	ABC /DE
+	(– ++	ABC / DE*
F	(– ++	ABC / DE*F
)	(	ABC / DE*F ++ –

2.12. Write the difference between stack and queue.

Ans.

S. No.	Stack	Queue
i.	A stack is logically a LIFO type of list.	A queue is logically a FIFO type of list.
ii.	No element other than the top of stack element is visible.	No element other than front and rear element are visible.

2.13. What are the advantages of queue over stack ?

Ans. **Advantages of queue over stack are :**

i. An element that is inserted first in the queue will be the first element to be removed.
ii. Insertion and deletion, both are possible only on one end in stack. While in queue elements are inserted at one end and elements are deleted at other end.

2.14. Write down the limitations of circular queue.

Ans. **Limitations of circular queue are :**

i. We cannot distinguish between full and empty queue.

ii. Front and rear indices are in exactly the same relative positions for an empty and for a full queue.

2.15. What is the significance of priority queue ?

AKTU 2016-17, Marks 02

Ans. Priority queue is a data structure in which elements can be stored as per their priorities. And therefore one can remove the elements from such queue according to their priorities. Such type of queue is useful to operating system in job scheduling algorithms.

2.16. Name the types of recursion.

Ans. **Types of recursion are :**

i. Direct recursion

ii. Indirect recursion

iii. Tail recursion

iv. Linear and tree recursion

2.17. Write the syntax to check whether a given circular queue is full or empty.

AKTU 2018-19, Marks 02

OR

Explain circular queue. What is the condition if circular queue is full ?

AKTU 2017-18, Marks 02

Ans. **Circular queue :** A circular queue is one in which the insertion of a new element is done at the very first location of the queue if the last location at the queue is full.

Syntax to check circular queue is full :

If ((front == MAX – 1) || (front == 0 && rear == MAX – 1))

Syntax to check circular queue is empty :

If (front == 0 && rear == – 1)

2.18. What is recursion ? Give disadvantages of recursion.

AKTU 2018-19, Marks 02

Ans. **Recursion :** Recursion is the process of expressing a function that calls itself to perform specific operation.

Disadvantages of recursion :

1. Recursive solution is always logical and it is very difficult to trace, debug and understand.
2. Recursion takes a lot of stack space, usually not considerable when the program is small and running on a PC.
3. Recursion uses more processor time.

☺☺☺

3 UNIT

Searching and Sorting (2 Marks Questions)

3.1. What do you mean by searching ?

Ans. Searching is a process of finding the location of given elements in the linear arrays. The search is said to be successful if the given element is found.

3.2. Name two searching techniques.

Ans. Two searching techniques are :
1. Linear (sequential) search
2. Binary search

3.3. Define sequential search.

Ans. In sequential search, each element of an array is read one-by-one sequentially and it is compared with the desired element.

3.4. Define index sequential search.

Ans. In index sequential search, an index file is created, that contains some specific group or division of required record when the index is obtained, then the partial indexing takes place as it is loaded in a specific group.

3.5. What do you mean by hashing ?

Ans. Hashing is a searching technique that is used to uniquely identify a specific object from a group of similar objects.

3.6. Classify the hashing functions based on the various methods by which the key value is found.

AKTU 2015-16, Marks 02

Ans. Hashing functions on various methods by which the key value is founded are :

i. Division method
ii. Multiplication method
iii. Mid square method
iv. Folding method

3.7. What is collision ?

Ans. Collision is a situation which occur when we want to add a new record R with key K to our file F, but the memory location address $H(k)$ is already occupied.

3.8. Discuss various collision resolution strategies for hash table.

Ans. Collision resolution strategies for hash table are :

i. Chaining method : It hold the address of a table element by using $h(K)$ = key% table slots.

ii. Open addressing method : In this, all the elements of the dynamic sets are stored in hash table itself.

3.9. What is sorting ? How is sorting essential for database applications ? **AKTU 2016-17, Marks 02**

Ans. Sorting : It is an operation which is used to put the elements of list in a certain order. *i.e.*, either in decreasing or increasing order.

Sorting essential for database applications : Sorting is easier and faster to locate items in a sorted list than unsorted. Sorting algorithms can be used in a program to sort an array for later searching or writing out to an ordered file or report. Sorted arrays/ lists make it easier to find things more quickly.

3.10. What do you understand by stable and in-place sorting ? **AKTU 2018-19, Marks 02**

Ans. Stable sorting : Stable sorting is an algorithm where two objects with equal keys appear in the same order in sorted output as they appear in the input unsorted array.

In-place sorting : An in-place sorting is an algorithm that does not need an extra space and produces an output in the same memory that contains the data by transforming the input 'in-place'. However, a small constant extra space used for variables is allowed.

3.11. Differentiate between internal and external sorting.

Ans.

S. No.	Internal sorting	External sorting
i.	The internal sorting resides in main memory.	External sorting resides in secondary memory.
ii.	It is independent of time to read/write a record.	It is dependent on time.

3.12. Give the worst case and best case time complexity of binary search. **AKTU 2016-17, Marks 02**

Ans. Worst case : In each comparison, the size of the search area is reduced by half. So, the efficiency of the binary search method at the worst case is $\log_2 n + 1$, *i.e.*, $O(\log_2 n + 1)$ where n is the total number of items that will be used for the binary search.

Best case : The best case of binary search occurs when the element we are searching for is the middle element of the list/array because in that case we will get the desired result in a single go. In this case, the time complexity of the algorithm will be O(1).

☺☺☺

4 UNIT

Graphs (2 Marks Questions)

4.1. What are the applications of graphs ?

Ans. Applications of graph are :

i. Representing relationship between components in electronic circuits.

ii. Transportation network in highway network, flight network.

iii. Computer network in local area network.

4.2. Write down the applications of DFS.

Ans. Applications of DFS :

i. Topological sorting.

ii. Finding connected components.

iii. Finding strongly connected components.

iv. Solving puzzles such as mazes.

4.3. Write down the applications of BFS.

Ans. Applications of BFS :

i. Finding all nodes within one connected component.

ii. Finding the shortest path between two nodes.

4.4. Define connected and strongly connected graph.

AKTU 2015-16, Marks 02

Ans. Connected graph : A graph G is said to be connected if there is at least one path between every pair of vertices in G.

Strongly connected graph : A graph G is said to be strongly connected if there is at least one directed path from every vertex to every other vertex.

4.5. What are the advantages of DFS over BFS ?

Ans. Advantages of DFS over BFS :

i. DFS has much lower memory requirement than BFS.

ii. DFS is better than BFS if the solution is at maximum depth.

4.6. How the graph can be represented in memory ? Explain with suitable example.

AKTU 2018-19, Marks 02

OR

List the different types of representation of graphs.

AKTU 2017-18, Marks 02

Ans. **Graph can be represented in memory using :**

1. Matrix representation
2. Linked representation

For example : Consider the following directed graph :

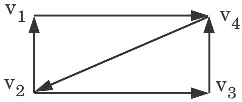


Fig. 1.

Matrix representation :

$$\begin{array}{c} \\ v_1 \\ v_2 \\ v_3 \\ v_4 \end{array} \begin{array}{c} \begin{array}{cccc} v_1 & v_2 & v_3 & v_4 \end{array} \\ \begin{bmatrix} 0 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 \end{bmatrix} \end{array}$$

Linked representation :

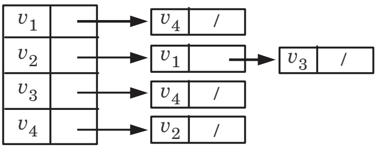


Fig. 2.

4.7. Discuss the disadvantages of Dijkstra's algorithm.

Ans. **Disadvantages of Dijkstra's algorithm are :**

i. It does a blind search thereby consuming a lot of time and wasting necessarily resource.

ii. It cannot handle negative edges. This leads to acyclic graphs.

4.8. How many ways are there to implement Kruskal's algorithm ?

Ans. **Ways to implement Kruskal's algorithm :**

i. By using disjoint sets : Using UNION and FIND operation.

ii. By using priority queue : Maintains weights in priority queue.

4.9. How Prim's algorithm is similar to Dijkstra's algorithm ?

Ans.

i. Similar to Dijkstra algorithm, in Prim's algorithm we also keep distance values and paths in distance table.

ii. The implementation of Prim's algorithm is identical to that of Dijkstra's algorithm so the running time is $O(|V|^2)$ without heaps and $O(E \log V)$ using binary heaps.

4.10. Prove that the number of odd degree vertices in a connected graph should be even. **AKTU 2016-17, Marks 02**

Ans. Let V_1 and V_2 be the set of vertices of even and odd degrees respectively. Thus, $V_1 \cap V_2 = \phi$ and $V_1 \cup V_2 = V$.

By Handshaking theorem,

$$2|E| = \sum_{v \in V} \deg(v) = \sum_{v \in V_1} \deg(v) + \sum_{v \in V_2} \deg(v)$$

As both $2|E|$ and $\sum_{v \in V_1} \deg(v)$ are even. So, $\sum_{v \in V_2} \deg(v)$ must be even.

Since, $\deg(v)$ is odd for all $v \in V_2$. So, the number of odd degree vertices in a connected graph must be even.

4.11. Number of nodes in a complete tree is 100000. Find its depth. **AKTU 2018-19, Marks 02**

Ans. Number of nodes in a complete tree = 100000

We know that,

$$n = 2^{h+1} - 1$$

$$(n + 1) = 2^{h+1}$$

$$\log_2(n + 1) = h + 1$$

$$\log_2(n + 1) - 1 = h$$

Putting

$$n = 100000$$

$$h = \log_2(100000 + 1) - 1$$

$$h = 15 \text{ (approx)}$$

☺☺☺

5 UNIT

Trees (2 Marks Questions)

5.1. Define tree.

Ans. A tree *T* is a finite non-empty set of elements. One of these elements is called the root, and the remaining elements, if any is partitioned into trees is called subtree of *T*. A tree is a non-linear data structure.

5.2. Define the depth of a node.

Ans. The depth of a node is the length of the path from the root to the node. A (rooted) tree with only one node (the root) has a depth of zero.

5.3. Describe the properties of binary tree.

Ans. **Properties of binary tree are :**

i. The number of nodes n in a full binary tree is $2^{n+1} - 1$.

ii. The number of leaf nodes in a full binary tree is 2^h where h is the height.

iii. The number of NULL links in a complete binary tree of n nodes is $n + 1$.

5.4. Define complete binary tree. Give example.

AKTU 2016-17, Marks 02

Ans. A tree is called complete binary tree if tree satisfies following conditions :

1. Each node has exactly two children except leaf node.
2. All leaf nodes are at same level.
3. If a binary tree contains m nodes at level l, it contains at most $2m$ nodes at level $l + 1$.

Example :

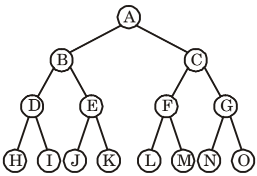


Fig. 1.

5.5. For tree construction which is the suitable and efficient data structure and why ?

AKTU 2015-16, Marks 02

Ans. Linked list is the most suitable and efficient data structure because it is easily accessible due to the concept of pointer used in it.

5.6. Discuss the concept of "successor" and "predecessor" in binary search tree. **AKTU 2017-18, Marks 02**

Ans. In binary search tree, if a node *X* has two children, then its predecessor is the maximum value in its left subtree and its successor is the minimum value in its right subtree.

5.7. Explain height balanced tree. List general cases to maintain the height. **AKTU 2017-18, Marks 02**

Ans.

i. An AVL (or height balanced) tree is a balanced binary search tree.
ii. In an AVL tree, balance factor of every node is either –1, 0 or +1.
iii. Balance factor of a node is the difference between the heights of left and right subtrees of that node.
Balance factor = height of left subtree – height of right subtree.

General cases to maintain the height are :

a. Left Left rotation (LL rotation)
b. Right Right rotation (RR rotation)
c. Left Right rotation (LR rotation) d. Right Left rotation (RL rotation)

5.8. Draw a binary tree for the expression : $A * B – (C + D) * (P/Q)$ **AKTU 2018-19, Marks 02**

Ans.

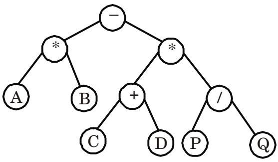


Fig. 2.

5.9. What is the maximum height of any AVL tree with 7 nodes ? **AKTU 2015-16, Marks 02**

Ans. Maximum height of any AVL tree with 7 nodes is 3.

5.10. When does a graph become tree ? **AKTU 2016-17, Marks 02**

Ans. A graph becomes a tree when there is exactly one path between every pair of its vertices.

5.11. How can we traverse a binary tree ?

Ans. We can traverse a binary tree using :

1. Inorder traversing 2. Preorder traversing 3. Postorder traversing

☺☺☺

B.Tech.

(SEM. III) ODD SEMESTER THEORY EXAMINATION, 2014-15

DATA STRUCTURES USING C

Time : 3 Hours **Max. Marks : 100**

1. Attempt any **four** parts of the following : **(5 × 4 = 20)**

a. Define data structure. Describe about its need and types. Why do we need a data type ?

b. Write difference between array and linked list.

c. What do you understand by complexity of an algorithm ? Compute the worst case complexity for the following C code :

```
main()
{
int s = 0, i, j, n;
        for (j = 0;  j < (3 * n); j++)
{
for (i = 0; i < n; i++)
{
s = s + i;
}
printf("%d", i);
}}
```

d. Write the difference between malloc( ) and calloc( ) functions. Why do we use dynamic memory allocation ?

e. Write an algorithm or C code to insert a node in doubly link list in beginning.

f. What is row-major order ? Explain with an example.

2. Attempt any **four** parts of the following : **(5 × 4 = 20)**

a. What is Tower of Hanoi problem ? Write the recursive code in C language for the problem.

b. What is circular queue ? Write a C code to insert an element in circular queue. Write all the condition for overflow.

c. What is stack ? Implement stack with singly linked list.

d. Write the procedures for insertion, deletion and traversal of a queue.

e. Write a function in C language to reverse a string using stack.

f. Convert following infix expression into postfix expression *A* + (*B* * *C* + *D*)/*E*.

3. Attempt any **two** parts of the following : **(10 × 2 = 20)**

a. Construct a height balanced binary search tree by performing following operations :

Step 1 : Insert
19, 16, 21, 11, 17, 25, 6, 13

Step 2 : Insert
3

Step 3 : Delete
16

b. What is Huffman tree ? Create a Huffman tree with following numbers :
24, 55, 13, 67, 88, 36, 17, 61, 24, 76

c. Define binary search tree. Create BST for the following data, show all steps :
20, 10, 25, 5, 15, 22, 30, 3, 14, 13

4. Attempt any **two** parts of the following : **(10 × 2 = 20)**

a. Define spanning tree. Find the minimal spanning tree for the following graph using Prim's algorithm.

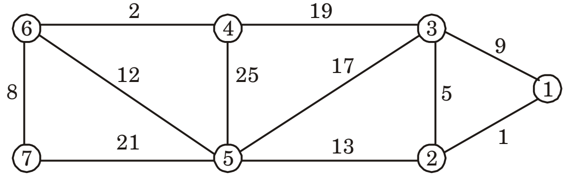


Fig. 1.

b. Find out the shortest path from node 1 to node 4 in a given graph (Fig. 4) using Dijkstra shortest path algorithm.

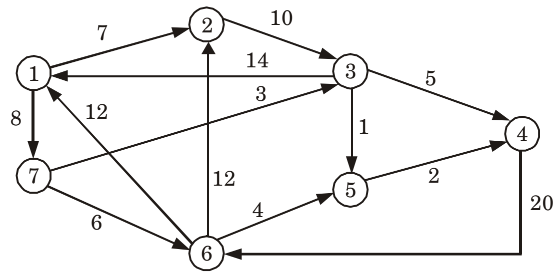


Fig. 2.

c. Write DFS algorithm to traverse a graph. Apply same algorithm for the graph given in Fig. 5 by considering node 1 as starting node.

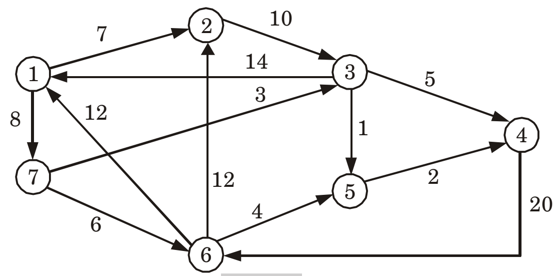


Fig. 3.

5. Attempt any **two** parts of the following : **(10 × 2 = 20)**

a. What do you mean by hashing and collision ? Discuss the advantages and disadvantages of hashing over other searching techniques.

b. Write an algorithm for merge sorting using the algorithm sort in ascending order.
10, 25, 16, 5, 35, 48, 8

c. Write short notes on any three of the following :

i. B-tree
ii. Insertion sort
iii. Heap sort
iv. Garbage collection

☺☺☺

SOLUTION OF PAPER (2014-15)

1. Attempt any **four** parts of the following : **(5 × 4 = 20)**

a. Define data structure. Describe about its need and types. Why do we need a data type ?

Ans. Data structure :

1. A data structure is a way of organizing all data items that considers not only the elements stored but also their relationship to each other.
2. Data structure is the representation of the logical relationship existing between individual elements of data.
3. Data structure is define as a mathematical or logical model of particular organization of data items.

Data structure is needed because :

1. It helps to understand the relationship of one element with the other.
2. It helps in the organization of all data items within the memory.

The data structures are divided into following categories :

1. Linear data structure :

a. A linear data structure is a data structure whose elements form a sequence, and every element in the structure has a unique predecessor and unique successor.

b. Examples of linear data structure are arrays, linked lists, stacks and queues.

2. Non-linear data structure :

a. A non-linear data structure it is a data structure whose elements do not form a sequence. There is no unique predecessor or unique successor.

b. Examples of non-linear data structures are trees and graphs.

Need of data type : The data type is needed because it determines what type of information can be stored in the field and how the data can be formatted.

b. Write difference between array and linked list.

Ans.

S. No.	Array	Linked list
1.	An array is a list of finite number of elements of same data type *i.e.*, integer, real or string etc.	A linked list is a linear collection of data elements called nodes which are connected by links.
2.	Elements can be accessed randomly.	Elements cannot be accessed randomly. It can be accessed only sequentially.

3.	Array is classified as : a. 1-D array b. 2-D array c. *n*-D array	A linked list can be linear, doubly or circular linked list.
4.	Each array element is independent and does not have a connection with previous element or with its location.	Location or address of element is stored in the link part of previous element or node.
5.	Array elements cannot be added, deleted once it is declared.	The nodes in the linked list can be added and deleted from the list.
6.	In array, elements can be modified easily by identifying the index value.	In linked list, modifying the node is a complex process.
7.	Pointer cannot be used in array.	Pointers are used in linked list.

c. What do you understand by complexity of an algorithm ? Compute the worst case complexity for the following C code :

```
main()
{
int s = 0, i, j, n;
        for (j = 0;  j < (3 * n); j++)
{
for (i = 0; i < n; i++)
{
s = s + i;
}
printf("%d", i);
}}
```

Ans.

1. The complexity of an algorithm M is the function $f(n)$ which gives the running time and/or storage space requirement of the algorithm in terms of the size n of the input data.
2. The storage space required by an algorithm is simply a multiple of the data size n.

   Worst case complexity : $\Omega(n) + \Omega(3n) = \Omega(n)$

   The time complexity can be calculated by computing the frequency count. This calculation is as follows :

Code	Frequency count
for (j = 0; j < (3*n); j ++)	(3*n) + 1
for (i = 0; i < n; i ++)	((3*n)*n) + 1
$s = s + i$;	((3*n)*n)
printf("%d", i)	$3n$
Total	$6n^2 + 6n + 2$

By considering only the order of magnitude, we can express the worst case time complexity O(n^2).

d. Write the difference between malloc( ) and calloc( ) functions. Why do we use dynamic memory allocation ?

Ans.

S. No.	malloc( )	calloc( )
1.	It takes single argument.	It takes two argument.
2.	Does not initialize the allocated memory.	Initialize the allocated memory to zero.
3.	**Syntax of malloc( ) :** Void *malloc (size_t n);	**Syntax of calloc( ) :** Void * calloc (size = t n, size_t size);

Uses of dynamic memory allocation :

i. In dynamic memory allocation, data structure can grow and shrink during the execution time.

ii. They have efficient memory utilization because memory is not preallocated.

iii. Insertion and deletion can be done very easily at the desired position.

e. Write an algorithm or C code to insert a node in doubly link list in beginning.

Ans. **Insertion at beginning :**

1. IF PTR = NULL then Write OVERFLOW
   Go to Step 9
   [END OF IF]
2. SET NEW_NODE = PTR
3. SET PTR = PTR -> NEXT
4. SET NEW_NODE -> DATA = VAL
5. SET NEW_NODE -> PREV = NULL
6. SET NEW_NODE -> NEXT = START
7. SET HEAD -> PREV = NEW_NODE
8. SET HEAD = NEW_NODE
9. EXIT

f. What is row-major order ? Explain with an example.

Ans.

1. In row major order, the element of an array is stored in computer memory as row-by-row.
2. Under row major representation, the first row of the array occupies the first set of memory locations reserved for the array, the second row occupies the next set, and so forth.
3. In row major order, elements of a two-dimensional array are ordered as :
$A_{11}, A_{12}, A_{13}, A_{14}, A_{15}, A_{16}, A_{21}, A_{22}, A_{23}, A_{24}, A_{25}, A_{26}, A_{31}, \ldots\ldots, A_{46}, A_{51}, A_{52}, \ldots\ldots, A_{56}$

Example :

Let us consider the following two-dimensional array :

$$\begin{bmatrix} a & b & c & d \\ e & f & g & h \\ i & j & k & l \end{bmatrix}$$

a. Move the elements of the second row starting from the first element to the memory location adjacent to the last element of the first row.
b. When this step is applied to all the rows except for the first row, we have a single row of elements. This is the row major representation.
c. By application of above mentioned process, we get $\{a, b, c, d, e, f, g, h, i, j, k, l\}$

2. Attempt any **four** parts of the following : **(5 × 4 = 20)**

a. What is Tower of Hanoi problem ? Write the recursive code in C language for the problem.

Ans. Tower of Hanoi problem :

1. Suppose three pegs, labelled A, B and C is given, and suppose on peg A, there are finite number of n disks with decreasing size.
2. The object of the game is to move the disks from peg A to peg C using peg B as an auxiliary.
3. The rule of game is follows :
a. Only one disk may be moved at a time. Specifically only the top disk on any peg may be moved to any other peg.
b. At no time, can a larger disk be placed on a smaller disk.

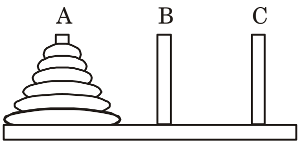


Fig. 1.

The solution to the Tower of Hanoi problem for $n = 3$.

Total number of steps to solve Tower of Hanoi problem of n disk

$$= 2^n - 1 = 2^3 - 1 = 7$$

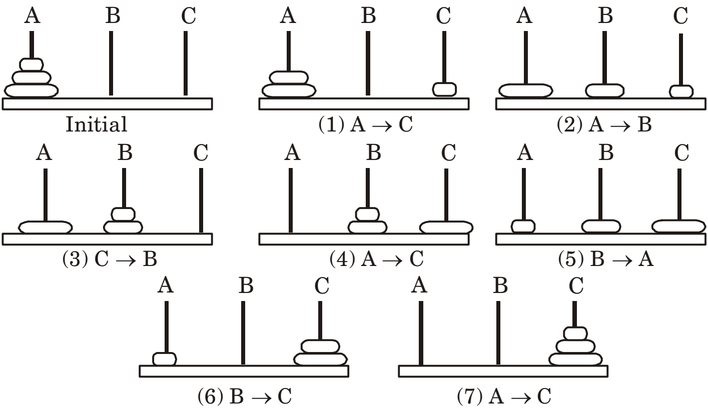


Fig. 2.

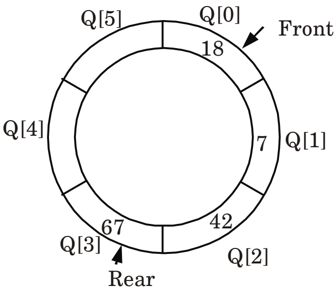
Recursive code for Tower of Hanoi :

```
#include<stdio.h>
#include<conio.h>
void main()
{
clrscr( );
int n;
char A = 'A', B = 'B', C = 'C';
void hanoi (int, char, char, char);
printf("Enter number of disks :");
scanf("%d", &n);
printf("\n\n Tower of Hanoi problem with %d disks\n", n);
printf("Sequence is : \n");
hanoi (n, A, B, C);
        printf("\n");
              getch();
}
void hanoi (int n, char A, char B, char C)
{
If(n ! = 0)
{
hanoi (n – 1, A, C, B);
printf("Move disk %d from %c to %c\n , n, A, C,");
hanoi (n – 1, B, A, C);
}
}
```

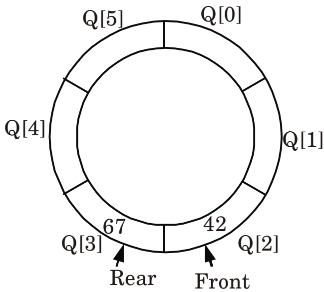
b. What is circular queue ? Write a C code to insert an element in circular queue. Write all the condition for overflow.

Ans.

1. A circular queue is one in which the insertion of a new element is done at the very first location of the queue if the last location at the queue is full.
2. In circular queue, the elements Q[0], Q[1], Q[2] ... Q[n – 1] is represented in a circular fashion.
   For example : Suppose *Q* is a queue array of six elements.
3. PUSH and POP operation can be performed on circular queue. Fig. 3 will illustrate the same.



(a) A circular queue after inserting 18, 7 , 42, 67.



(b) A circular queue after popping 18, 7.

Fig. 3.

C code to insert an element in circular queue :

```
void insert ()
{
    int item;
    if((front == 0 && rear == Max – 1) | |((front == rear + 1))
    {
    printf("Queue is overflow\n");
    return;
    }
    if(front == –1)    / *If queue is empty*/
    {
    front = 0;
    rear = 0;
    }
    else
    if(rear == Max – 1)    /*rear is at last position of queue*/
    rear = 0;
    else
    rear = rear + 1;
    printf("Input the element for insertion :");
    scanf("%d", &item);
    cqueue [rear] = item;
    }
```

Conditions for overflow : There are two conditions :

1. (front = 0) and (rear = Max – 1)
2. front = rear + 1

If any of these two conditions is satisfied, it means that overflow occurs.

c. What is stack ? Implement stack with singly linked list.

Ans. Stack :

1. A stack is one of the most commonly used data structure.
2. A stack, also called Last In First Out (LIFO) system, is a linear list in which insertion and deletion can take place only at one end, called top.
3. This structure operates in much the same way as stack of trays.
4. If we want to remove a tray from stack of trays it can only be removed from the top only.
5. The insertion and deletion operation in stack terminology are known as PUSH and POP operations.

Implementation using singly linked list :

```
typedef struct stack
{
    int *data;
    struct stack *next;
    }stack;
void push(stack **top, int *data)
    {
    stack *newn;
    newn = (stack *)malloc(sizeof(stack));
    newn->data = data;
    newn->next = (stack *)NULL;
    if(*top == NULL)
    {
    *top = newn;
    return;
    }
    newn->next = (*top);
    *top = newn;
    }
    int *pop(stack **top)
    {
    int *rval = (int *)NULL;
    stack *tmp;
    if(*top != NULL)
    {
        tmp = *top;
        *top = (*top)->next;
        rval = tmp->data;
        free(tmp);
```

```
    }
        return(rval);
    }
```

d. Write the procedures for insertion, deletion and traversal of a queue.

Ans.

1. **Insertion :**
   Insert in Q (Queue, Max, Front, Rear, Element)
   Let Queue is an array, Max is the maximum index of array, Front and Rear to hold the index of first and last element of Queue respectively and Element is value to be inserted.
   Step 1 : If Front = 1 and Rear = Max or if Front = Rear + 1
   Display "Overflow" and Return
   Step 2 : If Front = NULL [Queue is empty]
   Set Front = 1 and Rear = 1
   else if Rear = N, then
   Set Rear = 1
   else
   Set Rear = Rear + 1
   [End of if Structure]
   Step 3 : Set Queue [Rear] = Element [This is new element]
   Step 4 : End
2. **Deletion :**
   Delete from Q (Queue, Max, Front, Rear, Item)
   Step 1 : If Front = NULL [Queue is empty]
   display "Underflow" and Return
   Step 2 : Set Item = Queue [Front]
   Step 3 : If Front = Rear [Only one element]
   Set Front = Rear and Rear = NULL
   Else if
   Front = N, then
   Set Front = 1
   Else
   Set Front = Front + 1
   [End if structure]
   Step 4 : End
3. **Traversal of a queue :** Here queue has Front End FE and Rear End RE. This algorithm traverse queue applying an operation PROCESS to each element of queue :
   Step 1 : [Initialize counter] Set K = FE
   Step 2 : Repeat step 3 and 4 while K ≤ RE
   Step 3 : [Visit element] Apply PROCESS to queue [K]
   Step 4 : [Increase counter] Set K = K + 1
   [End of step 2 loop]
   Step 5 : Exit

e. Write a function in C language to reverse a string using stack.

Ans.

```
#include<stdio.h>
#include<conio.h>
#include<string.h>
#define MAX 20
    int top = – 1;
    char stack [MAX];
    char pop();
    push(char);
    main()
        {
        clrscr();
        char str [20];
        int i;
        printf("Enter the string : ");
        gets(str);
        for(i = 0; i < strlen(str); i++)
        push (str [i]);
        for(i = 0; i < strlen(str); i++)
        str[i] = pop();
        printf("Reversed string is :");
        puts (str);
        getch();
        }
    push (char item)
        {
        if(top == MAX – 1)
        printf("Stack overflow\n");
        else
        stack[++top] = item;
        }
    char pop()
        {
        if(top == – 1)
        printf("Stack underflow \n");
        else
        return stack [top – –];
        }
```

f. Convert following infix expression into postfix expression $A + (B * C + D)/E$.

Ans. (A + (B * C + D)/E)

Character	Stack	Postfix
(	(	
A	(	A
+	(+	A
(	(+(	A
B	(+(	AB
*	(+(*	AB
C	(+(*	ABC
+	(+(+	ABC*
D	(+(+	ABC*D
)	(+	ABC*D+
/	(+/	ABC*D+
E	(+/	ABC*D+E
)	(	ABC*D+E/+

Resultant postfix expression : ABC * D + E/+

3. Attempt any **two** parts of the following : **(10 × 2 = 20)**

a. Construct a height balanced binary search tree by performing following operations :

Step 1 : Insert
19, 16, 21, 11, 17, 25, 6, 13

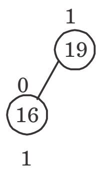
Step 2 : Insert
3

Step 3 : Delete
16

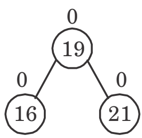
Ans. Step 1 : Insert 19, 16, 21, 11, 17, 25, 6, 13 :

Insert 19 :

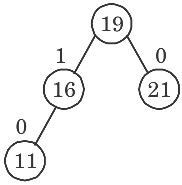
Insert 16 :



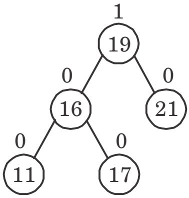
Insert 21 :



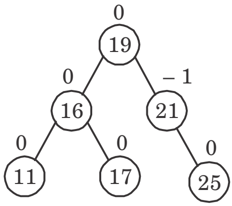
Insert 11 :



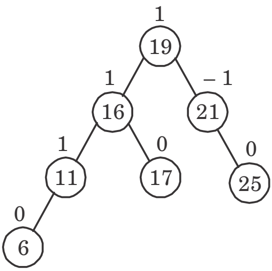
Insert 17 :



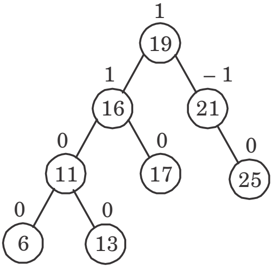
Insert 25 :



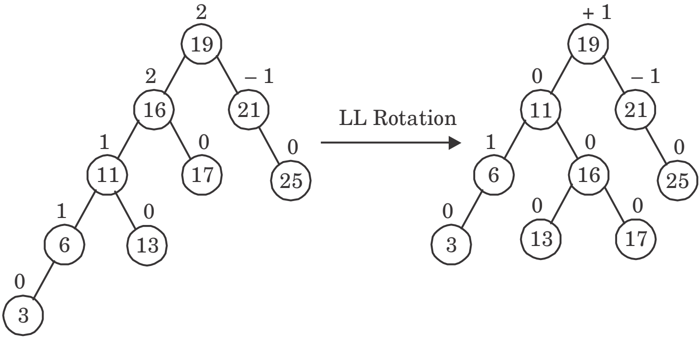
Insert 6 :



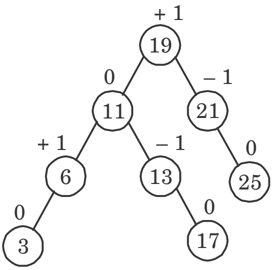
Insert 13 :



Step 2 : Insert 3 :



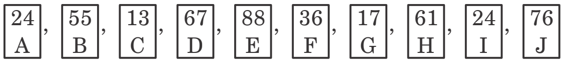
Step 3 : Delete 16 :



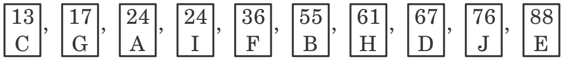
b. What is Huffman tree ? Create a Huffman tree with following numbers :
24, 55, 13, 67, 88, 36, 17, 61, 24, 76

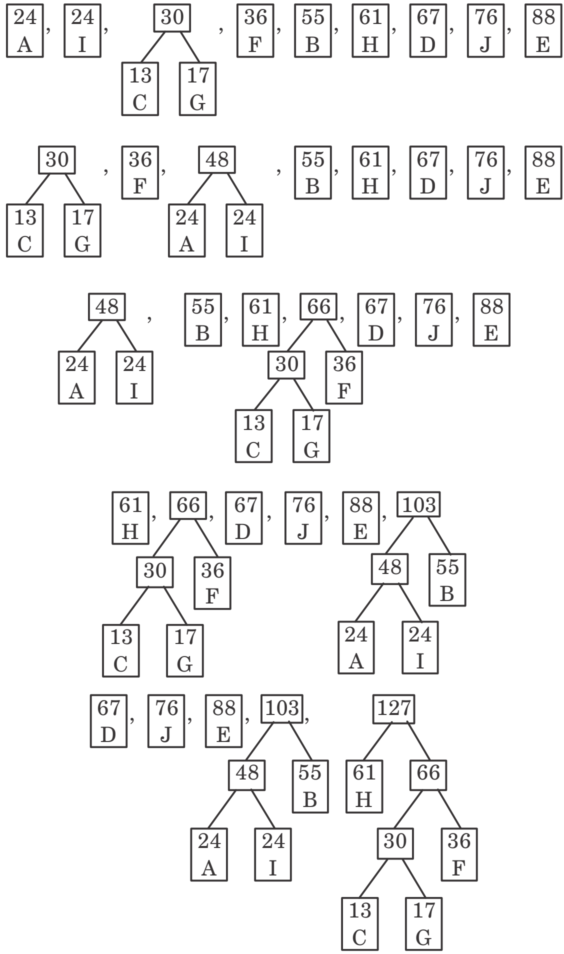
Ans. Huffman tree is a binary tree in which each node in the tree represents a symbol and each leaf represent a symbol of original alphabet.

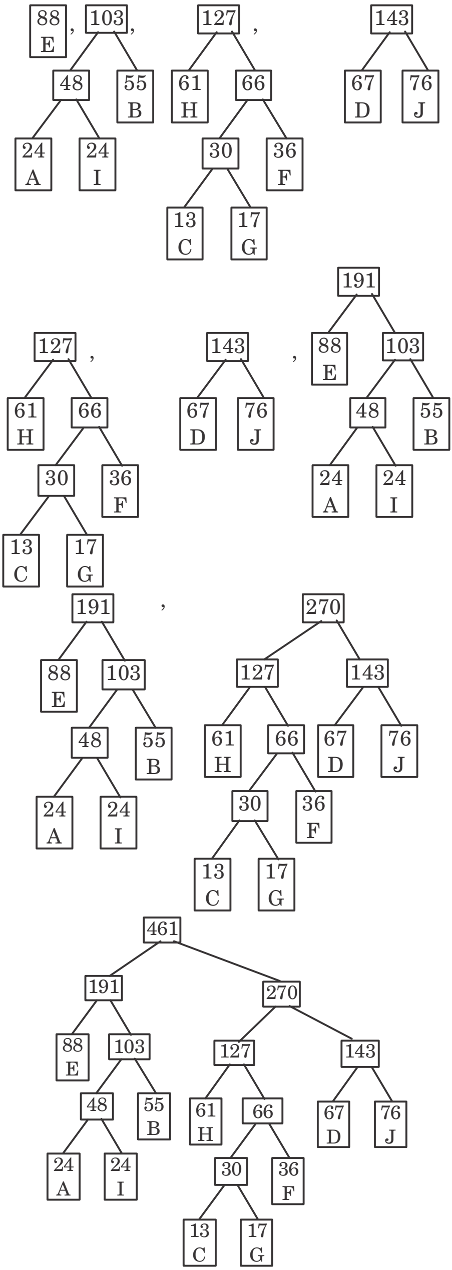
Numerical :



Arrange all the numbers in ascending order :







88 E
103
48
55 B
24 A
24 I
127
61 H
66
30
36 F
13 C
17 G
143
67 D
76 J
191
270
461

c. Define binary search tree. Create BST for the following data, show all steps :
20, 10, 25, 5, 15, 22, 30, 3, 14, 13

Ans. Binary search tree :

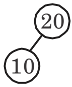
1. A binary search tree is a binary tree.
2. Binary search tree can be represented by a linked data structure in which each node is an object.
3. In addition to a key field, each node contains fields left, right and P, which point to the nodes corresponding to its left child, its right child and its parent respectively.

Numerical :

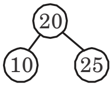
20, 10, 25, 5, 15, 22, 30, 3, 14, 13

1. Insert 20 :

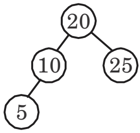
2. Insert 10 :



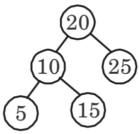
3. Insert 25 :



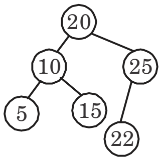
4. Insert 5 :



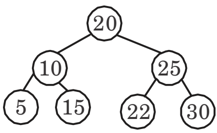
5. Insert 15 :



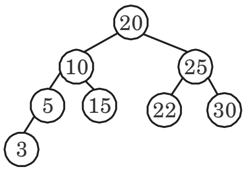
6. Insert 22 :



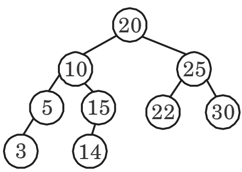
7. Insert 30 :



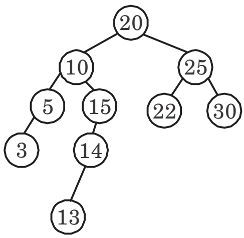
8. Insert 3 :



9. Insert 14 :



10. Insert 13 :



4. Attempt any **two** parts of the following : **(10 × 2 = 20)**

a. Define spanning tree. Find the minimal spanning tree for the following graph using Prim's algorithm.

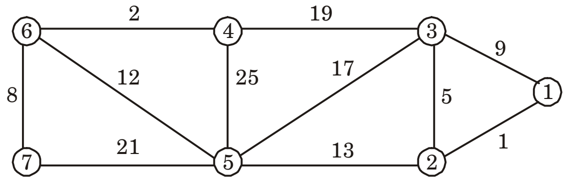


Fig. 4.

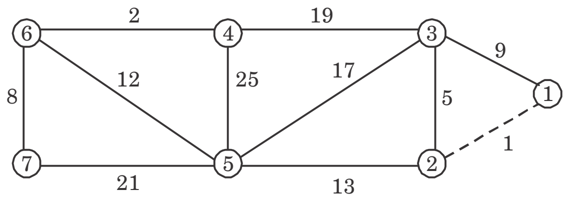
Ans. Spanning tree :

1. A spanning tree of a graph is a sub-graph which is a tree and contains all the vertices of graph.

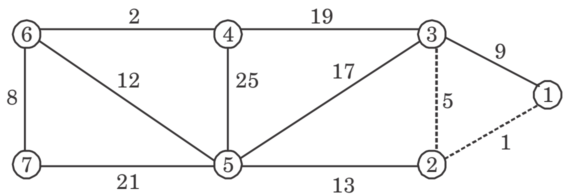
Numerical :

$$\begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \\ 7 \end{array}
\begin{array}{c} \begin{array}{ccccccc} 1 & 2 & 3 & 4 & 5 & 6 & 7 \end{array} \\
\begin{bmatrix} - & 1 & 9 & - & - & - & - \\ 1 & - & 5 & - & 13 & - & - \\ 9 & 5 & - & 19 & 17 & - & - \\ - & - & 19 & - & 25 & 2 & - \\ - & 13 & 17 & 25 & - & 12 & 21 \\ - & - & - & 2 & 12 & - & 8 \\ - & - & - & - & 21 & 8 & - \end{bmatrix} \end{array}$$

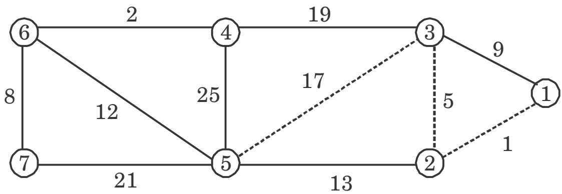
According to Prim's algorithm, we choose vertex 1.
We choose edge (1, 2), since it has minimum value.



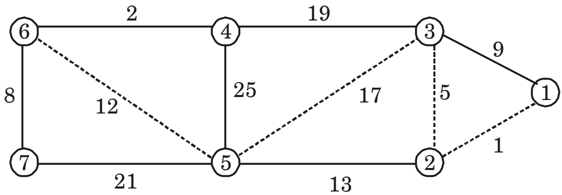
Now at vertex 2, we choose the edge (2, 3), since it has minimum value.



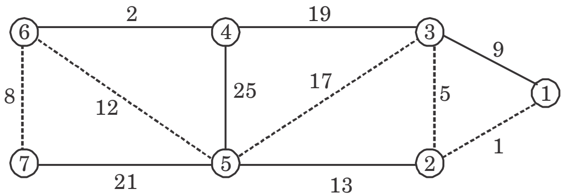
Now at vertex 3, we cannot choose edge (3, 1) because it will create a cycle so we choose (3, 5).

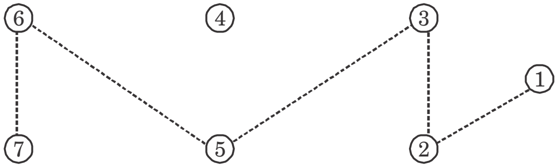


Now at vertex 5, we choose the edge (5, 6) since it has minimum value.



Now at vertex 6, we choose the edge (6, 7) since it has minimum value.



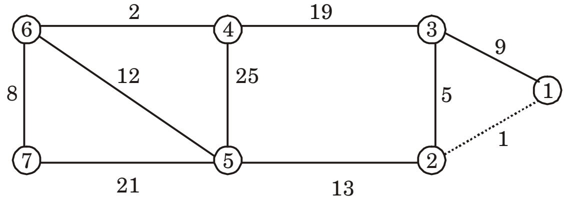


Since in spanning tree, the tree should cover all the vertices and should not make cycle.

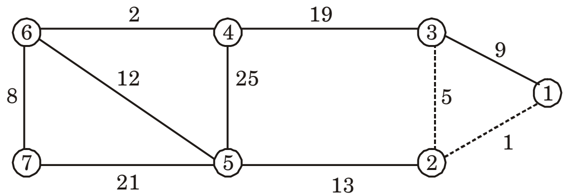
But in the above tree, 4 is remaining so the above asked question is wrong. If we assume to remove the edge from {3, 5} then the spanning tree is :

$$
\begin{array}{c c}
 & \begin{array}{ccccccc} 1 & 2 & 3 & 4 & 5 & 6 & 7 \end{array} \\
\begin{array}{c} 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \\ 7 \end{array} &
\left[\begin{array}{ccccccc}
- & 1 & 9 & - & - & - & - \\
1 & - & 5 & - & 13 & - & - \\
9 & 5 & - & 19 & - & - & - \\
- & - & 19 & - & 25 & 2 & - \\
- & 13 & 17 & 25 & - & 12 & 21 \\
- & - & - & 2 & 12 & - & 8 \\
- & - & - & - & 21 & 8 & -
\end{array}\right]
\end{array}
$$

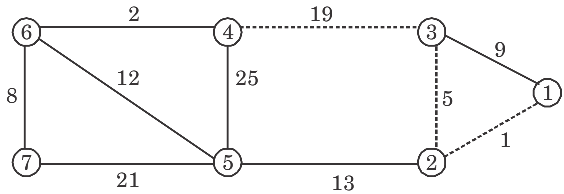
According to Prim's algorithm, let's choose vertex 1.
We choose edge {1, 2}, since it has minimum value.



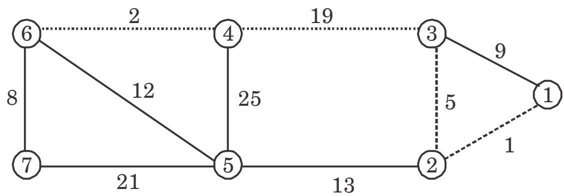
Now at vertex 2, we choose the edge (2, 3), since it has minimum value.



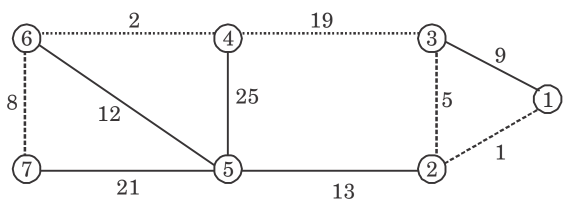
Now at vertex 3, we choose the edge (3, 4), since it has minimum value.



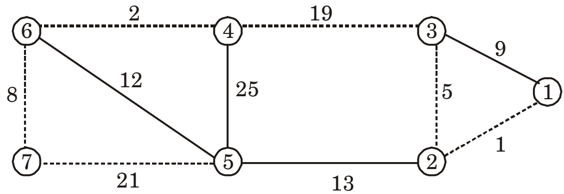
Now at vertex 4, we choose the edge (4, 6), since it has minimum value.



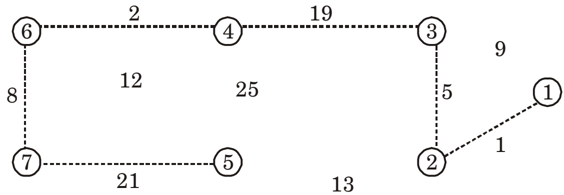
Now at vertex 6, we choose the edge (6, 7), since it has minimum value.



Now at vertex 7, we cannot choose the edge (7, 6), because we have already traversed this edge these we choose (7, 5).



∴ The spanning tree is



b. Find out the shortest path from node 1 to node 4 in a given graph (Fig. 5) using Dijkstra shortest path algorithm.

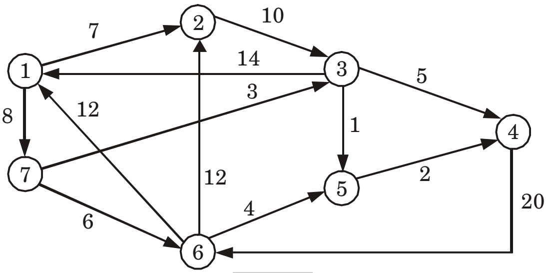


Fig. 5.

Ans.

1	2	3	4	5	6	7
$\boxed{0}$	∞	∞	∞	∞	∞	∞
$\boxed{0}$	$\boxed{7}$	$\boxed{17}$	∞	∞	∞	8
$\boxed{0}$	$\boxed{7}$	$\boxed{17}$	∞	∞	∞	$\boxed{8}$
$\boxed{0}$	$\boxed{7}$	$\boxed{11}$	∞	∞	14	$\boxed{8}$
$\boxed{0}$	$\boxed{7}$	$\boxed{11}$	16	$\boxed{12}$	14	$\boxed{8}$
$\boxed{0}$	$\boxed{7}$	$\boxed{11}$	14	$\boxed{12}$	14	$\boxed{8}$
$\boxed{0}$	$\boxed{7}$	$\boxed{11}$	$\boxed{14}$	$\boxed{12}$	$\boxed{14}$	$\boxed{8}$

Shortest path from node 1 to node 4 = 0 + 7 + 11 + 14 = 32

c. Write DFS algorithm to traverse a graph. Apply same algorithm for the graph given in Fig. 6 by considering node 1 as starting node.

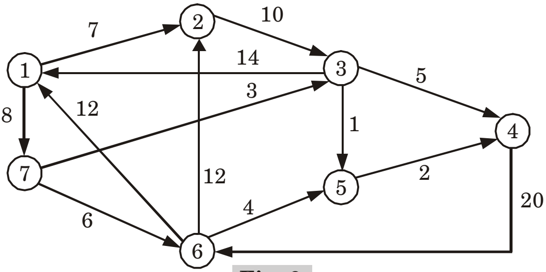


Fig. 6.

Ans. **Depth First Search (DFS) :** The general idea behind a depth first search beginning at a starting node A is as follows :

a. First, we examine the starting node A.

b. Then, we examine each node N along a path P which begins at A; that is, we process neighbour of A, then a neighbour of neighbour of A, and so on.

c. This algorithm uses a stack instead of queue.

Algorithm :

i. Initialize all nodes to ready state (STATUS = 1).

ii. Push the starting node A onto stack and change its status to the waiting state (STATUS = 2).

iii. Repeat steps (iv) and (v) until queue is empty.

iv. Pop the top node N of stack, process N and change its status to the processed state (STATUS = 3).

v. Push onto stack all the neighbours of N that are still in the ready state (STATUS = 1) and change their status to the waiting state (STATUS = 2).

[End of loop]

vi. End.

Numerical : Adjacency list of the given graph :

$1 \rightarrow 2, 7$

$2 \rightarrow 3$

$3 \rightarrow 5, 4, 1$

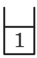
$4 \rightarrow 6$

$5 \rightarrow 4$

$6 \rightarrow 2, 5, 1$

$7 \rightarrow 3, 6$

1. Initially set STATUS = 1 for all vertex
2. Push 1 onto stack and set their STATUS = 2



3. Pop 1 from stack, change its STATUS = 1 and

Push 2, 7 onto stack and change their STATUS = 2; DFS = 1

7
2

4. Pop 7 from stack, Push 3, 6; DFS = 1, 7

6
3
2

5. Pop 6 from stack, Push 5; DFS = 1, 7, 6

5
3
2

6. Pop 5 from stack, Push 4; DFS = 1, 7, 6, 5

4
3
2

7. Pop 4 from stack; DFS = 1, 7, 6, 5, 4

3
2

8. Pop 3 from stack; DFS = 1, 7, 6, 5, 4, 3

2

9. Pop 2 from stack; DFS = 1, 7, 6, 5, 4, 3

Now, the stack is empty, so the depth first traversal of a given graph is 1, 7, 6, 5, 4, 3.

5. Attempt any **two** parts of the following : **(10 × 2 = 20)**

a. What do you mean by hashing and collision ? Discuss the advantages and disadvantages of hashing over other searching techniques.

Ans. **Hashing :**

1. Hashing is a technique that is used to uniquely identify a specific object from a group of similar objects.
2. Hashing is the transformation of a string of characters into a usually shorter fixed-length value or key that represents the original string.

Collision :

1. Collision is a situation which occur when we want to add a new record R with key k to our file F, but the memory location address $H(k)$ is already occupied.
2. A collision occurs when more than one keys map to same hash value in the hash table.

Advantages of hashing over other search techniques :

1. The main advantage of hash tables over other table data structures is speed. This advantage is more apparent when the number of entries is large (thousands or more).
2. Hash tables are particularly efficient when the maximum number of entries can be predicted in advance, so that the bucket array can be allocated once with the optimum size and never resized.
3. If the set of key-value pairs is fixed and known ahead of time (so insertions and deletions are not allowed), one may reduce the average lookup cost by a careful choice of the hash function, bucket table size, and internal data structures.

Disadvantages of hashing over other search techniques :

1. Hash tables can be more difficult to implement than self-balancing binary search trees. Choosing an effective hash function for a specific application is more an art than a science. In open-addressed hash tables it is fairly easy to create a poor hash function.
2. The cost of a good hash function can be significantly higher than the inner loop of the lookup algorithm for a sequential list or search tree.
3. Hash tables are not effective when the number of entries is very small. For certain string processing applications, such as spell-checking, hash tables may be less efficient than trees, finite automata, or arrays.
4. If each key is represented by a small enough number of bits, then, instead of a hash table, one may use the key directly as the index into an array of values.

b. Write an algorithm for merge sorting using the algorithm sort in ascending order.
10, 25, 16, 5, 35, 48, 8

Ans. **Merge sort :**

a. Merge sort is a sorting algorithm that uses the idea of divide and conquer.
b. This algorithm divides the array into two halves, sorts them separately and then merges them.
c. This procedure is recursive, with the base criteria that the number of elements in the array is not more than 1.

MERGE_SORT (a, p, r) :

1. if $p < r$
2. then $q \leftarrow \lfloor (p + r)/2 \rfloor$
3. MERGE-SORT (A, p, q)
4. MERGE-SORT ($A, q + 1, r$)
5. MERGE (A, p, q, r)

MERGE (A, p, q, r) :

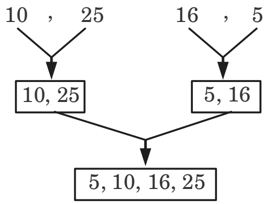
1. $n_1 = q - p + 1$
2. $n_2 = r - q$

3. Create arrays L [1 ....n_1 + 1] and
   R [1......n_2 + 1]
4. for $i = 1$ to n_1
   do
   $L[i] = A\ [p + i - 1]$
   endfor
5. for $j = 1$ to n_2
   do
   $R[j] = A[q + j]$
   endfor
6. $L[n_1 + 1] = \infty$, $R[n_2 + 1] = \infty$
7. $i = 1, j = 1$
8. for $k = p$ to r
   do
   if $L[i] \leq R[j]$
   then $A[k] \leftarrow L[i]$
   $i = i + 1$
   else $A[k] = R[j]$
   $j = j + 1$
   endif
   endfor
9. exit

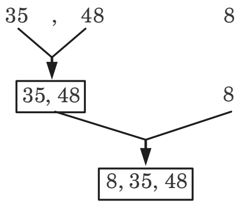
   Numerical :

   10, 25, 16, 5, 35, 48, 8

1. Divide first half $\boxed{10,\ 25,\ 16,\ 5}$ $\boxed{35,\ 48,\ 8}$
2. **Consider the first half :** 10, 25, 16, 5 again divide into two sub-arrays



3. **Consider the second half :** 35, 48, 5 again divide into two sub-arrays



4. Merge these two sorted sub-arrays,

5, 10, 16, 25		8, 35, 48

↓

5, 8, 10, 16, 25, 35, 45

This is the sorted array.

c. Write short notes on any three of the following :

i. B-tree

Ans. B-tree :

1. A B-tree is a self-balancing tree data structure that keeps data sorted and allows searches, sequential access, insertions, and deletions in logarithmic time.
2. A B-tree of order m is a tree which satisfies the following properties :

a. Every node has at most m children.

b. Every non-leaf node (except root) has at least $\lceil m/2 \rceil$ children.

c. The root has at least two children if it is not a leaf node.

d. A non-leaf node with k children contains $k - 1$ keys.

e. All leaves appear in the same level.

ii. Insertion sort

Ans.

1. In insertion sort, we pick up a particular value and then insert it at the appropriate place in the sorted sublist, *i.e.*, during k^{th} iteration the element $a[k]$ is inserted in its proper place in the sorted sub-array $a[1], a[2], a[3] .... a[k-1]$.
2. This task is accomplished by comparing $a[k]$ with $a[k-1]$, $a[k-2]$, $a[k-3]$ and so on until the first element $a[j]$ such that $a[j] \le a[k]$ is found.
3. Then each of the elements $a[k-1], a[k-2], a[j+1]$ are moved one position up and then element $a[k]$ is inserted in $[j+1]^{\text{st}}$ position in the array.

Insertion-Sort (*A*)

```
for j ← 2 to length[A]
do key ← A[j]    /*Insert A[j] into the sorted sequence A[1....j – 1].*/
i ← j – 1
while i > 0 and A[i] > key
  do A[i + 1] ← A[i]
i ← i – 1
A[i + 1] ← key
```

Analysis of insertion sort :

Complexity of best case is $O(n)$

Complexity of average case is $O(n^2)$

Complexity of worst case is $O(n^2)$

iii. Heap sort

Ans.

1. Heap sort finds the largest element and puts it at the end of array, then the second largest item is found and this process is repeated for all other elements.
2. The general approach of heap sort is as follows :

a. From the given array, build the initial max heap.
b. Interchange the root (maximum) element with the last element.
c. Use repetitive downward operation from root node to rebuild the heap of size one less than the starting.
d. Repeat step (a) and (b) until there are no more elements.

MAX-HEAPIFY (A, i) :

1. $i \leftarrow$ left $[i]$
2. $r \leftarrow$ right $[i]$
3. if $l \leq$ heap-size $[A]$ and $A[l] > A[i]$
4. then largest $\leftarrow l$
5. else largest $\leftarrow i$
6. if $r \leq$ heap-size $[A]$ and $A[r] > A$ [largest]
7. then largest $\leftarrow r$
8. if largest $\neq i$
9. then exchange $A[i] \leftrightarrow A$[largest]
10. MAX-HEAPIFY [A, largest]

HEAP-SORT(A) :

1. BUILD-MAX-HEAP (A)
2. for $i \leftarrow$ length $[A]$ down to 2
3. do exchange $A[1] \leftrightarrow A[i]$
4. heap-size $[A] \leftarrow$ heap-size $[A] - 1$
5. MAX-HEAPIFY (A, 1)

iv. Garbage collection

Ans.

1. When some memory space becomes reusable due to the deletion of a node from a list or due to deletion of entire list from a program then we want the space to be available for future use.
2. One method to do this is to immediately reinsert the space into the free-storage list. This is implemented in the linked list.
3. This method may be too time consuming for the operating system of a computer.
4. In another method, the operating system of a computer may periodically collect all the deleted space onto the free storage list. This type of technique is called garbage collection.
5. Garbage collection usually takes place in two steps. First the computer runs through all lists, tagging those cells which are currently in use and then the computer runs through the memory, collecting all untagged space onto the free storage list.
6. The garbage collection may take place when there is only some minimum amount of space or no space at all left in the free storage list or when the CPU is idle and has time to do the collection.

☺☺☺

B.Tech.

(SEM. III) ODD SEMESTER THEORY EXAMINATION, 2015-16

DATA STRUCTURES USING C

Time : 3 Hours **Max. Marks : 100**

Section-A

1. Attempt all parts. All parts carry equal marks. Write answer of each part in short. (2 × 10 = 20)

a. Given a 2D array *A* [– 100 : 100, – 5 : 50]. Find the address of element *A* [99, 49] considering the base address 10 and each element requires 4 bytes for storage. Follow row-major order.

b. What are the various asymptotic notations ? Explain the Big-oh notation.

c. What are the notations used in evaluation of arithmetic expressions using prefix and postfix forms ?

d. Classify the hashing functions based on the various methods by which the key value is found.

e. What is the maximum height of any AVL tree with 7 nodes ?

f. If the Tower of Hanoi is operated on n = 10 disks, calculate the total number of moves.

g. Define connected and strongly connected graph.

h. Translate infix expression into its equivalent postfix expression : $A * (B + D)/E - F * (G + H/K)$.

i. For tree construction which is the suitable and efficient data structure and why ?

j. Explain the application of sparse matrices.

Section-B

Note : Attempt any **five** questions from this section. **(10 × 5 = 50)**

2. Consider the linear arrays *AAA* [5 : 50], *BBB* [– 5 : 10] and *CCC* [1 : 8].

a. Find the number of elements in each array.

b. Suppose base (*AAA*) = 300 and *w* = 4 words per memory cell for *AAA*. Find the address of *AAA* [15], *AAA* [35] and *AAA* [55].

3. Describe all rotations in AVL tree. Construct AVL tree from the following nodes : *B, C, G, E, F, D, A*.

4. Explain binary search tree and its operations. Make a binary search tree for the following sequence of numbers, show all steps : 45, 32, 90, 34, 68, 72, 15, 24, 30, 66, 11, 50, 10.

5. Explain Dijkstra's algorithm with suitable example.

6. Write a C function for linked list implementation of stack. Write all the primitive operations.

7. Draw a binary tree with following traversal :
Inorder : *D B H E A I F J C G*
Preorder : *A B D E H C F I J G*

8. Consider the following undirected graph.

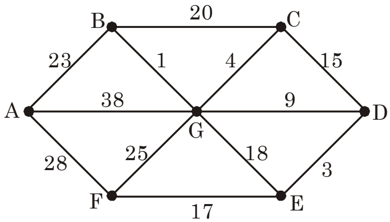


Fig. 1.

a. Find the adjacency list representation of the graph.

b. Find a minimum cost spanning tree by Kruskal's algorithm.

9. How do you calculate the complexity of sorting algorithms ? Also, write a recursive function in 'C' to implement the merge sort on given set of integers.

Ans.

Section-C

Note : Attempt any **two** questions from this section. **(15 × 2 = 30)**

10. What are doubly linked lists ? Write C program to create doubly linked list.

OR

How do you find the complexity of an algorithm ? What is the relation between the time and space complexities of an algorithm ? Justify your answer with an example.

11. Write an algorithm for finding solution to the Tower of Hanoi problem. Explain the working of your algorithm (with 4 disks) with diagrams.

12. Define a B-tree. What are the applications of B-tree ? Draw a B-tree of order 4 by insertion of the following keys in order : *Z, U, A, I, W, L, P, X, C, J, D, M, T, B, Q, E, H, S, K, N, R, G, Y, F, O, V*.

☺☺☺

SOLUTION OF PAPER (2015-16)

Section-A

1. Attempt all parts. All parts carry equal marks. Write answer of each part in short. (2 × 10 = 20)

a. Given a 2D array A [– 100 : 100, – 5 : 50]. Find the address of element A [99, 49] considering the base address 10 and each element requires 4 bytes for storage. Follow row-major order.

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

$$\text{LOC}(A[99][49]) = 10 + 4\,[50\,(99 - (-100)) + 49 - (-5)]$$
$$= 10 + 4\,[50\,(199) + 54] = 40026$$

b. What are the various asymptotic notations ? Explain the Big-oh notation.

Ans. Various asymptotic notations are :

1. Theta notation (θ - notation)
2. Big-Oh (O - notation)
3. Omega notation (Ω - notation)

Big-Oh notation : It is used when there is only an asymptotic upper bound. For a given function $g(n)$, O($g(n)$) is denoted by a set of functions.

c. What are the notations used in evaluation of arithmetic expressions using prefix and postfix forms ?

Ans. Notations used in evaluation of arithmetic expressions are :

i. **Infix notation :** In this notation, the operator symbol is placed between its two operands.
For example : To add A to B we can write as, A + B or B + A

ii. **Polish (Prefix) notation :** Here the operator symbol is placed before its two operands.
For example : To add A to B we can write as, + AB or + BA

iii. **Reverse polish (Postfix) notation :** In this notation, the operator symbol is placed after its two operands.
For example : To add A and B we can write as : AB+ or BA+

d. Classify the hashing functions based on the various methods by which the key value is found.

Ans. Hashing functions on various methods by which the key value is founded are :

i. Division method
ii. Multiplication method
iii. Mid square method
iv. Folding method

e. What is the maximum height of any AVL tree with 7 nodes ?

Ans. Maximum height of any AVL tree with 7 nodes is 3.

f. If the Tower of Hanoi is operated on *n* = 10 disks, calculate the total number of moves.

Ans. For n number of disks, total number of moves $= 2^n - 1$

For 10 disks, *i.e.*, $n = 10$, total number of moves $= 2^{10} - 1$

$= 1024 - 1$

$= 1023$

Therefore, if the Tower of Hanoi is operated on $n = 10$ disks, then total number of moves are 1023.

g. Define connected and strongly connected graph.

Ans. **Connected graph :** A graph G is said to be connected if there is at least one path between every pair of vertices in G.

Strongly connected graph : A graph G is said to be strongly connected if there is at least one directed path from every vertex to every other vertex.

h. Translate infix expression into its equivalent postfix expression : *A* * (*B* + *D*)/*E* – *F (*G* + *H*/*K*).**

Ans. Infix expression : A * (B + D)/E – F*(G + H/K)

A * (BD +)/E – F * (G + HK/)

A (BD +)*/E – F* (GHK/+)

(ABD + * E/) – (FGHK/+*)

ABD + * E/FGHK / + * –

Equivalent postfix expression is :

ABD + * E/FGHK / + * –

i. For tree construction which is the suitable and efficient data structure and why ?

Ans. Linked list is the most suitable and efficient data structure because it is easily accessible due to the concept of pointer used in it.

j. Explain the application of sparse matrices.

Ans. There are two applications of sparse matrix which are :

1. The sparse matrices are useful for computing large scale operations that dense matrices can not handle.
2. It is used in solving partial differential equations.

Section-B

Note : Attempt any **five** questions from this section. **(10 × 5 = 50)**

2. Consider the linear arrays *AAA* [5 : 50], *BBB* [– 5 : 10] and *CCC* [1 : 8].

a. Find the number of elements in each array.

b. Suppose base (*AAA*) = 300 and *w* = 4 words per memory cell for *AAA*. Find the address of *AAA* [15], *AAA* [35] and *AAA* [55].

Ans.

a. The number of elements is equal to the length; hence use the formula :

$$\text{Length} = \text{UB} - \text{LB} + 1$$

$$\text{Length } (AAA) = 50 - 5 + 1 = 46$$

$$\text{Length } (BBB) = 10 - (-5) + 1 = 16$$

$$\text{Length } (CCC) = 8 - 1 + 1 = 8$$

b. Use the formula

$\text{LOC } (AAA\ [i]) = \text{Base } (AAA) + w\ (i - \text{LB})$

$\text{LOC } (AAA\ [15]) = 300 + 4\ (15 - 5) = 340$

$\text{LOC } (AAA\ [35]) = 300 + 4\ (35 - 5) = 420$

AAA [55] is not an element of *AAA*, since 55 exceeds UB = 50.

3. Describe all rotations in AVL tree. Construct AVL tree from the following nodes : *B, C, G, E, F, D, A.*

Ans. AVL rotations :

i. An AVL (or height balanced) tree is a balanced binary search tree.

ii. In an AVL tree, balance factor of every node is either –1, 0 or +1.

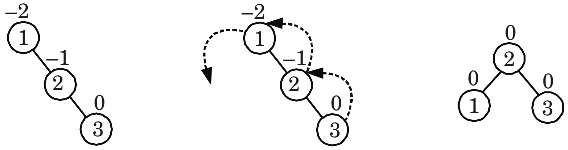
iii. Balance factor of a node is the difference between the heights of left and right subtrees of that node.

Balance factor = height of left subtree – height of right subtree

iv. In order to balance a tree, there are four cases of rotations :

1. Left Left rotation (LL rotation) : In LL rotation every node moves one position to left from the current position.

Insert 1, 2 and 3



Tree is unbalanced

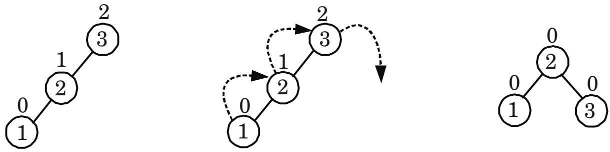
To make tree balance we use LL rotation which moves nodes one position to left

After LL rotation tree is balanced

Fig. 1.

2. Right Right rotation (RR rotation) : In RR rotation every node moves one position to right from the current position.

Insert 3, 2 and 1



Tree is unbalanced because node 3 has balance factor 2

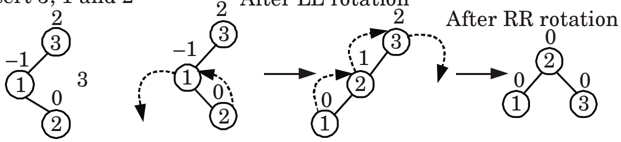
To make tree balance we use RR rotation which moves nodes one position to right

After RR Rotation tree is balanced

Fig. 2.

3. **Left Right rotation (LR rotation) :** The LR Rotation is combination of single left rotation followed by single right rotation. In LR rotation, first every node moves one position to left then one position to right from the current position.

Insert 3, 1 and 2



Tree is unbalanced because node 3 has balanced factor 2

LL rotation

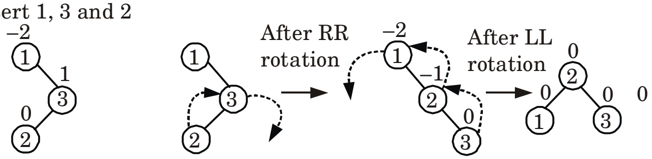
RR rotation

After LR rotation tree is balanced

Fig. 3.

4. **Right Left rotation (RL rotation) :** The RL rotation is the combination of single right rotation followed by single left rotation. In RL rotation, first every node moves one position to right then one position to left from the current position.

Insert 1, 3 and 2



Tree is unbalanced because node 1 has balance factor –2

RR rotation

LL rotation

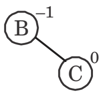
After RL rotation tree is balanced

Fig. 4.

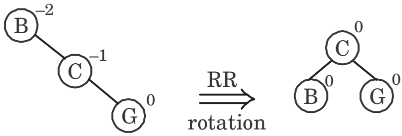
Construction of AVL tree : B, C, G, E, F, D, A

Insert B :

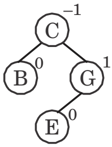
Insert C :



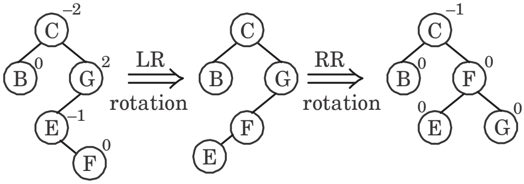
Insert G :



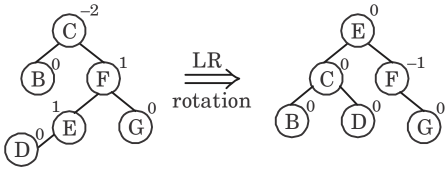
Insert E :



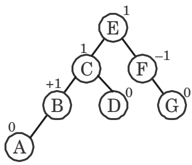
Insert F :



Insert D :



Insert A :



4. **Explain binary search tree and its operations. Make a binary search tree for the following sequence of numbers, show all steps : 45, 32, 90, 34, 68, 72, 15, 24, 30, 66, 11, 50, 10.**

Ans. **Binary search tree :**

1. A binary search tree is a binary tree.
2. Binary search tree can be represented by a linked data structure in which each node is an object.
3. In addition to a key field, each node contains fields left, right and P, which point to the nodes corresponding to its left child, its right child and its parent respectively.
4. A non-empty binary search tree satisfies the following properties :

a. Every element has a key (or value) and no two elements have the same value.
b. The keys, if any, in the left subtree of root are smaller than the key in the node.
c. The keys, if any in the right subtree of the root are larger than the keys in the node.
d. The left and right subtrees of the root are also binary search tree.

Various operations of BST are :

a. Searching in a BST :

Searching for a data in a binary search tree is much faster than in arrays or linked lists. The TREE-SEARCH (x, k) algorithm searches the tree root at x for a node whose key value equals to k. It returns a pointer to the node if it exist otherwise NIL.

TREE-SEARCH (x, k)

```
If x = NIL or k = key [x]
then return x
If k < key [x]
then return TREE-SEARCH (left [x], k)
else return TREE-SEARCH (right [x], k)
```

b. Traversal operation on BST :

All the traversal operations are applicable in binary search trees. The inorder traversal on a binary search tree gives the sorted order of data in ascending (increasing) order.

c. Insertion of data into a binary search tree :

To insert a new value w into a binary search tree T, we use the procedure TREE-INSERT. The procedure passed a node z for which key[z] = w, left [z] = NIL and Right [z] = NIL.

```
y ← NIL
x ← root [T]
while x ≠ NIL
do y ← x
if key [z] < key [x]
then x ← left [x]
else x ← right [x]
P[z] ← y
if y = NIL
then root [T] ← z
else if key [z] < key [y]
then left [y] ← z
else right [y] ← z
```

d. Delete a node : Deletion of a node from a BST depends on the number of its children. Suppose to delete a node with key = z from BST T, there are 3 cases that can occur.

Case 1 : N has no children. Then N is deleted from T by simply replacing the location of N in the parent node $P(N)$ by the null pointer.

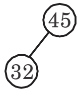
Case 2 : N has exactly one child. Then N is deleted from T by simply replacing the location of N in $P(N)$ by the location of the only child of N.

Case 3 : N has two children. Let $S(N)$ denote the inorder successor of N. (The reader can verify that $S(N)$ does not have a left child). Then N is deleted from T by first deleting $S(N)$ from T (by using Case 1 or Case 2) and then replacing node N in T by the node $S(N)$.

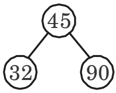
Numerical :

1. **Insert 45 :**

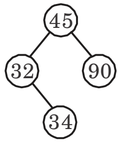
2. **Insert 32 :**



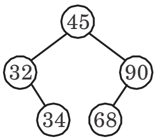
3. **Insert 90 :**



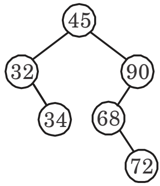
4. **Insert 34 :**



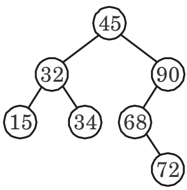
5. **Insert 68 :**



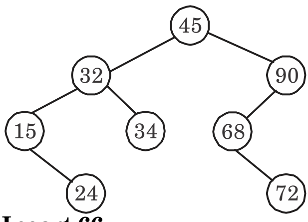
6. **Insert 72 :**



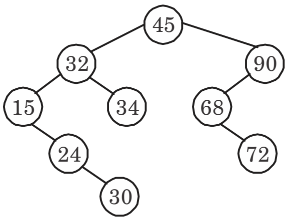
7. **Insert 15 :**



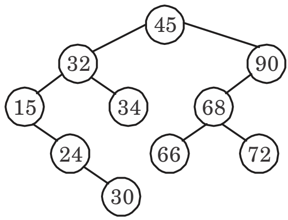
8. **Insert 24 :**



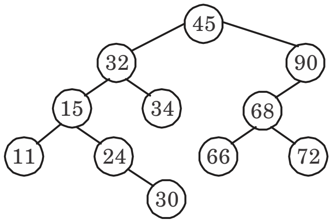
9. **Insert 30 :**



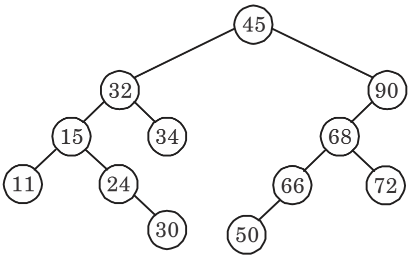
10. **Insert 66 :**



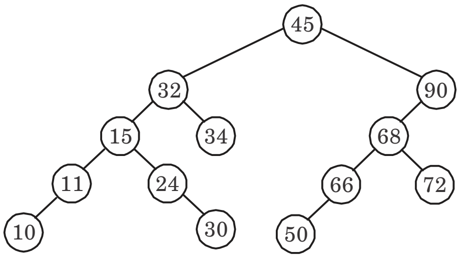
11. Insert 11 :



12. Insert 50 :



13. Insert 10 :



5. Explain Dijkstra's algorithm with suitable example.

Ans. Algorithm :

a. Dijkstra's algorithm, is a greedy algorithm that solves the single-source shortest path problem for a directed graph $G = (V, E)$ with non-negative edge weights, *i.e.*, we assume that $w(u, v) \geq 0$ each edge $(u, v) \in E$.

b. Dijkstra's algorithm maintains a set S of vertices whose final shortest-path weights from the source s have already been determined.

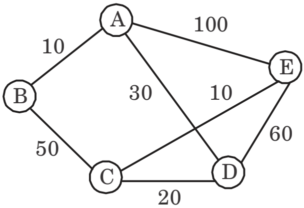
c. That is, for all vertices $v \in S$, we have $d[v] = \delta(s, v)$.

d. The algorithm repeatedly selects the vertex $u \in V - S$ with the minimum shortest-path estimate, inserts u into S, and relaxes all edges leaving u.

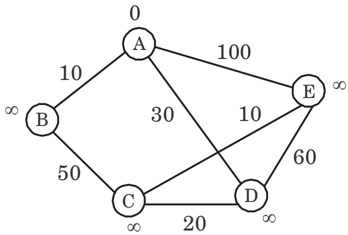
e. We maintain a priority queue Q that contains all the vertices in $v - s$, keyed by their d values.
f. Graph G is represented by adjacency list.
g. Dijkstra's always chooses the "lightest or "closest" vertex in $V - S$ to insert into set S, that it uses as a greedy strategy.

DIJKSTRA (G, w, s)
1. INITIALIZE-SINGLE-SOURCE (G, s)
2. $S \leftarrow \phi$
3. $Q \leftarrow V[G]$
4. while $Q \neq \phi$
5. do $u \leftarrow$ EXTRACT-MIN (Q)
6. $S \leftarrow S \cup \{u\}$
7. for each vertex $v \in$ Adj $[u]$
8. do RELAX (u, v, w)

Example : Working of Dijkstra's algorithm for following graph :

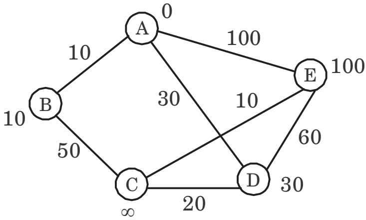


Extract min (A) :



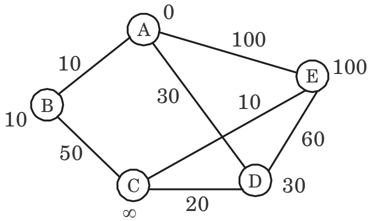
A	B	C	D	E
$\boxed{0}$	∞	∞	∞	∞

All edges leaving A :



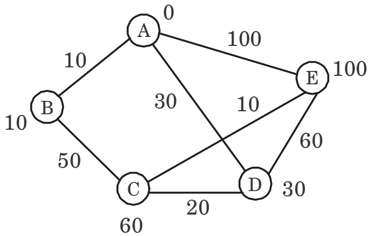
A	B	C	D	E
$\boxed{0}$	∞	∞	∞	∞
	10	∞	30	100

Extract min (*B*) :



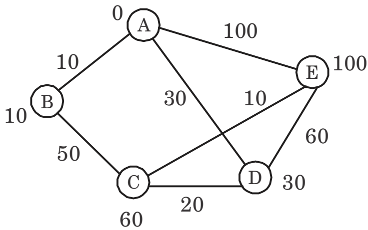
A	B	C	D	E
$\boxed{0}$	∞	∞	∞	∞
	$\boxed{10}$	∞	30	100

All edges leaving *B* :



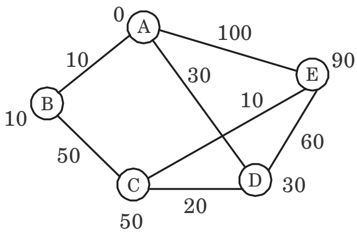
A	B	C	D	E
$\boxed{0}$	∞	∞	∞	∞
	$\boxed{10}$	∞	30	100
		60	30	100

Extract min(*D*) :



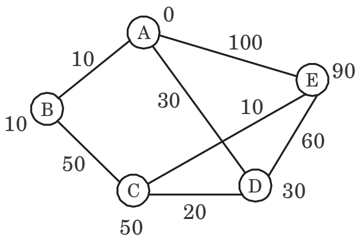
A	B	C	D	E
$\boxed{0}$	∞	∞	∞	∞
	$\boxed{10}$	∞	30	100
		60	$\boxed{30}$	100

All edges leaving (*D*) :



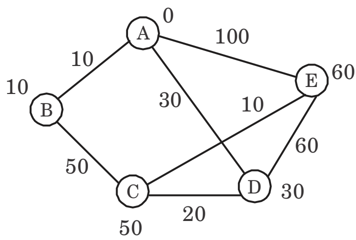
A	B	C	D	E
$\boxed{0}$	∞	∞	∞	∞
	$\boxed{10}$	∞	30	100
		60	$\boxed{30}$	100
		50		90

Extract min(C) :



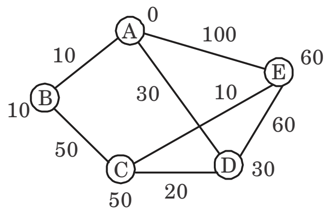
A	B	C	D	E
$\boxed{0}$	∞	∞	∞	∞
	$\boxed{10}$	∞	30	100
		60	$\boxed{30}$	100
		$\boxed{50}$		90

All edges leaving C :



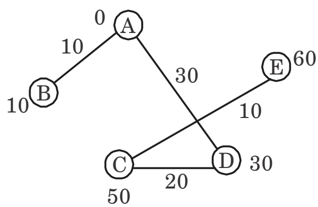
A	B	C	D	E
$\boxed{0}$	∞	∞	∞	∞
	$\boxed{10}$	∞	30	100
		60	$\boxed{30}$	100
		$\boxed{50}$		90
				60

Extract min(E) :



A	B	C	D	E
$\boxed{0}$	∞	∞	∞	∞
	$\boxed{10}$	∞	30	100
		60	$\boxed{30}$	100
		$\boxed{50}$		90
				$\boxed{60}$

Shortest path :



6. Write a C function for linked list implementation of stack. Write all the primitive operations.

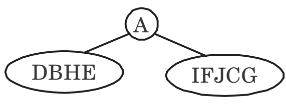
Ans.

```
#include<stdio.h>
#include<conio.h>
#include<alloc.h>
    struct node
    {
    int info;
    struct node *link;
    };
        struct node *top;
        void main()
        {
        void create(), traverse(), push(), pop();
        create();
        printf("\n stack is :\n");
        traverse();
        pop();
        printf("After push the element in the stack is : \n") ;
        traverse();
        pop( );
        printf("After pop the element in the stack is : \n")
        traverse();
        getch();
        }
        void create()
        {
        struct node *ptr, *cpt;
        char ch;
        ptr = (struct node *) malloc (sizeof (struct node));
        printf("Input first info");
        scanf("%d", &ptr -> info);
        ptr ->link = NULL;
    do
        {
        cpt = (struct node *) malloc (sizeof (struct node));
        printf("Input next information");
        scanf("%d", &cpt -> info);
        cpt -> link = ptr;
        ptr = cpt;
        printf("Press <Y/N> for more information");
        ch = getch() ;
        }
        while (ch == 'Y')
        top = ptr;
        }
```

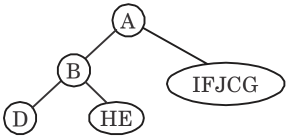
```
void traverse()
    {
    struct node *ptr ;
    printf ("Traversing of stack : \n");
    ptr = top ;
    while (ptr != NULL)
    {
    printf ("%d\n", ptr -> info);
    ptr = ptr ->link;
    }
    }
void push()
    {
    struct node *ptr;
    ptr = (struct node *) malloc (sizeof (struct node));
    if(ptr == NULL)
    {
    printf("Overflow\n");
    return;
    }
    printf("Input New node information");
    scanf("%d", &ptr -> info);
    ptr ->link = top;
    top = ptr;
    }
void pop()
    {
    struct node *ptr;
    if(top == NULL)
    {
    printf ("Underflow \n");
    return;
    }
    ptr = top;
    top = ptr ->link;
    free (ptr);
    }
```

7. Draw a binary tree with following traversal :
Inorder : *D B H E A I F J C G*
Preorder : *A B D E H C F I J G*

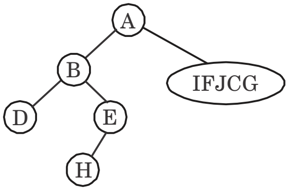
Ans. From preorder traversal, we get root node to be *A*.



Now considering left subtree.
Observing both the traversal we can get *B* as root node and *D* as left child and *HE* as a right subtree.



Now observing the preorder traversal we get *E* as a root node and *H* as a left child.



Repeating the above process with the right subtree of root node *A*, we finally obtain the required tree in given Fig. 5.

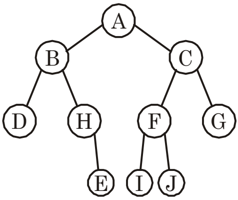


Fig. 5.

8. Consider the following undirected graph.

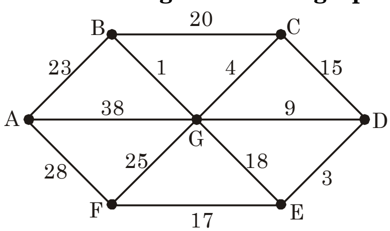


Fig. 6.

a. Find the adjacency list representation of the graph.
b. Find a minimum cost spanning tree by Kruskal's algorithm.

Ans.

a.

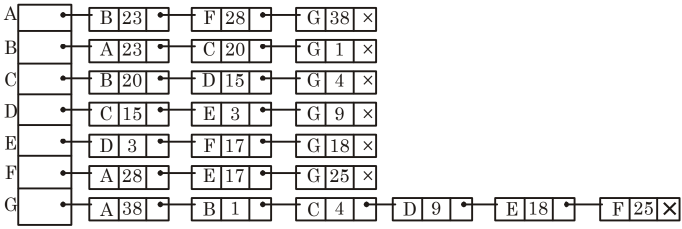
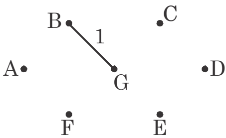


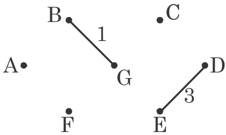
Fig. 7.

b. Kruskal's algorithm :

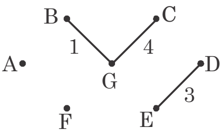
i. We will choose $e = BG$ as it has minimum weight.



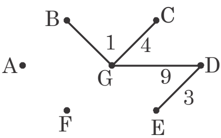
ii. Now choose $e = ED$.



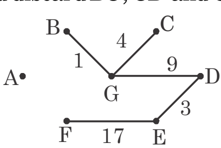
iii. Choose $e = CG$, since it has minimum weights.



iv. Choose $e = GD$.



v. Choose $e = EF$ and discard BC, CD and GE because they form cycle.



vi. Now choose $e = AB$ and discard AG, FG and AF because they form cycle. Final minimum spanning tree is given as :

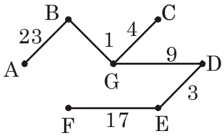


Fig. 8.

9. How do you calculate the complexity of sorting algorithms ? Also, write a recursive function in 'C' to implement the merge sort on given set of integers.

Ans. Complexity of sorting algorithms :

1. The complexity of a sorting algorithm measures the running time as a function of the number n of items to be sorted.
2. We know that each sorting algorithm S is made up of the following operations, where $A_1, A_2, ..., A_n$ contain the items to be sorted and B is an auxiliary location :
   a. Comparisons, which test whether $A_i < A_j$ or test whether $A_i < B$
   b. Interchanges, which switch the contents of A_i and A_j or of A_i and B
   c. Assignments, which set $B := A_i$ and then set $A_j := B$ or $A_j := A_i$
3. The complexity function measures only the number of comparisons.
4. There are two main cases whose complexity we calculate *i.e.*, the worst case and the average case.

Function :

```
void merge (int low, int mid, int high)
{
int temp [MAX] ;
int i = low;
int j = mid + 1;
int k = low;
while ((i <= mid) && (j <= high))
{
if (array [i] <= array [j ] )
      temp [k++] = array [i++];
else
      temp [k++] = array [j++] ;
}
while (i <= mid)   {
temp [k++] = array [i++];
while (j <= high)
temp [k++] = array [j++] ;
for (i = low; i <= high; i++)
array [i] = temp [i];
}
void merge_sort (int low, int high)
```

```
{
int mid;
if (low != high)
{
mid = (low + high) / 2;
merge_sort (low, mid);
merge_sort (mid + 1, high);
merge (low, mid, high);
}
}
```

Section-C

Note : Attempt any **two** questions from this section. **(15 × 2 = 30)**

10. What are doubly linked lists ? Write C program to create doubly linked list.

Ans. Doubly linked list :

1. The doubly or two-way linked list uses double set of pointers, one pointing to the next node and the other pointing to the preceding node.
2. In doubly linked list, all nodes are linked together by multiple links which help in accessing both the successor and predecessor node for any arbitrary node within the list.
3. Every node in the doubly linked list has three fields :

LPT	INFO	RPT

Fig. 9.

4. LPT will point to the node in the left side (or previous node) *i.e.*, LPT will hold the address of the previous node, RPT will point to the node in the right side (or next node) *i.e.*, RPT will hold the address of the next node.
5. INFO field store the information of the node.
6. A doubly linked list can be shown as follows :

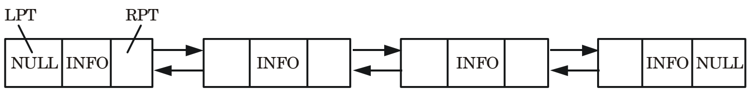


Fig. 10. Doubly linked list.

7. The structure defined for doubly linked list is :

```
struct node
    {
        int info;
        struct node *rpt;
        struct node *lpt;
    } node;
```

Program :

```
# include<stdio.h>
```

```
# include<conio.h>
# include<alloc.h>
struct node
    {
    int info ;
    struct node *lpt ;
    struct node *rpt ;
    } ;
struct node *first ;
void main ( )
{
create ( ) ;
getch ( ) ;
}
void create ( )
{
struct node *ptr, *cpt ;
char ch ;
ptr = (struct node *) malloc (size of (struct node)) ;
printf (“Input first node information”) ;
scanf (“%d”, & ptr → info) ;
ptr → lpt = NULL ;
first = ptr ;
do
{
cpt = (struct node *) malloc (size of (struct node)) ;
printf (“Input next node information”);
scanf (“%d”, & cpt → info) ;
ptr → rpt = cpt ;
cpt → lpt = ptr ;
ptr = cpt ;
printf (“Press <Y/N> for more node”) ;
ch = getch ( );
}
while (ch == ‘Y’) ;
ptr → rpt = NULL ;
}
```

OR

How do you find the complexity of an algorithm ? What is the relation between the time and space complexities of an algorithm ? Justify your answer with an example.

Ans. Complexity of an algorithm :

1. The complexity of an algorithm M is the function $f(n)$ which gives the running time and/or storage space requirement of the algorithm in terms of the size n of the input data.
2. The storage space required by an algorithm is simply a multiple of the data size n.
3. Following are various cases in complexity theory :

a. **Worst case :** The maximum value of $f(n)$ for any possible input.

b. **Average case :** The expected value of $f(n)$ for any possible input.

c. **Best case :** The minimum possible value of $f(n)$ for any possible input.

Types of complexity :

1. **Space complexity :** The space complexity of an algorithm is the amount of memory it needs to run to completion.
2. **Time complexity :** The time complexity of an algorithm is the amount of time it needs to run to completion.

Relation between the time and space complexities of an algorithm :

1. The time and space complexities are not related to each other.
2. They are used to describe how much space/time our algorithm takes based on the input.
3. For example, when the algorithm has space complexity of :
   a. O(1) *i.e.*, constant then the algorithm uses a fixed (small) amount of space which does not depend on the input. For every size of the input the algorithm will take the same (constant) amount of space.
   b. $O(n)$, $O(n^2)$, $O(\log(n))$ - these indicate that we create additional objects based on the length of our input.
4. In contrast, the time complexity describes how much time our algorithm consumes based on the length of the input.
5. For example, when the algorithm has time complexity of :
   a. O(1) *i.e.*, constant then no matter how big is the input it always takes a constant time.
   b. $O(n)$, $O(n^2)$, $O(\log(n))$ - again it is based on the length of the input.

For example :

```
function(list l) {              function(list l) {
for (node in l) {               print("I got a list"); }
print(node) ;
}
}
```

In this example, both take O(1) space as we do not create additional objects which shows that time and space complexity might be different.

11. Write an algorithm for finding solution to the Tower of Hanoi problem. Explain the working of your algorithm (with 4 disks) with diagrams.

Ans. Tower of Hanoi problem :

1. Suppose three pegs, labelled A, B and C is given, and suppose on peg A, there are finite number of n disks with decreasing size.
2. The object of the game is to move the disks from peg A to peg C using peg B as an auxiliary.
3. The rule of game is follows :

a. Only one disk may be moved at a time. Specifically only the top disk on any peg may be moved to any other peg.
b. At no time, can a larger disk be placed on a smaller disk.

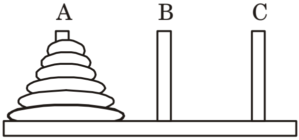


Fig. 11.

The solution to the Tower of Hanoi problem for $n = 3$.

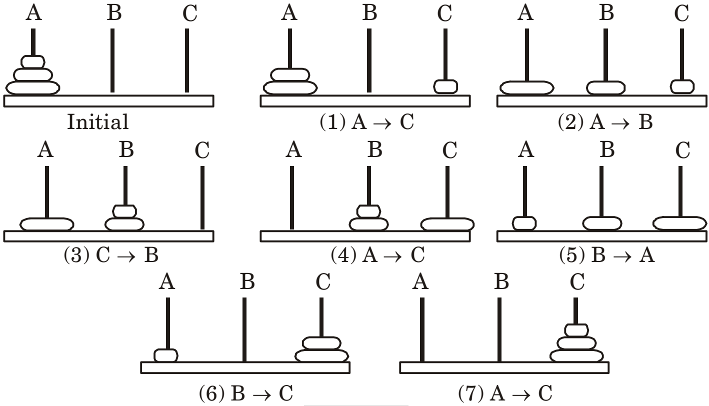


Fig. 12.

Total number of steps to solve Tower of Hanoi problem of n disk

$$= 2^n - 1 = 2^3 - 1 = 7$$

Algorithm :

TOWER (N, BEG, AUX, END)

This procedure gives a recursive solution to the Tower of Hanoi problem for N disks.

1. If N = 1, then :
   a. Write: BEG → END
   b. Return
   [End of If structure]
2. [Move N – 1 disk from peg BEG to peg AUX]
   Call TOWER (N – 1, BEG, END, AUX)
3. Write: BEG → END
4. [Move N – 1 disk from peg AUX to peg END]
   Call TOWER (N – 1, AUX, BEG, END)
5. Return

Time complexity :

Let the time required for n disks is $T(n)$.

There are 2 recursive calls for $n - 1$ disks and one constant time operation to move a disk from 'from' peg to 'to' peg. Let it be k_l.

Therefore,

$T(n) = 2\,T(n-1) + k_1$
$T(0) = k_2$, a constant.
$T(1) = 2k_2 + k_1$
$T(2) = 4k_2 + 2k_1 + k_1$
$T(2) = 8k_2 + 4k_1 + 2k_1 + k_1$
Coefficient of $k_1 = 2^n$
Coefficient of $k_2 = 2^n - 1$
Time complexity is $O(2^n)$ or $O(a^n)$ where a is a constant greater than 1.
So, it has exponential time complexity.

Space complexity :

Space for parameter for each call is independent of n *i.e.*, constant. Let it be k.
When we do the 2nd recursive call 1st recursive call is over. So, we can reuse the space of 1st call for 2nd call. Hence,
$T(n) = T(n-1) + k$
$T(0) = k$
$T(1) = 2k$
$T(2) = 3k$
$T(3) = 4k$
So, the space complexity is $O(n)$.

Numerical : Fig. 13 contains a schematic illustration of the recursive solution for TOWER (4, A, B, C) (4 disks, 3 pegs)

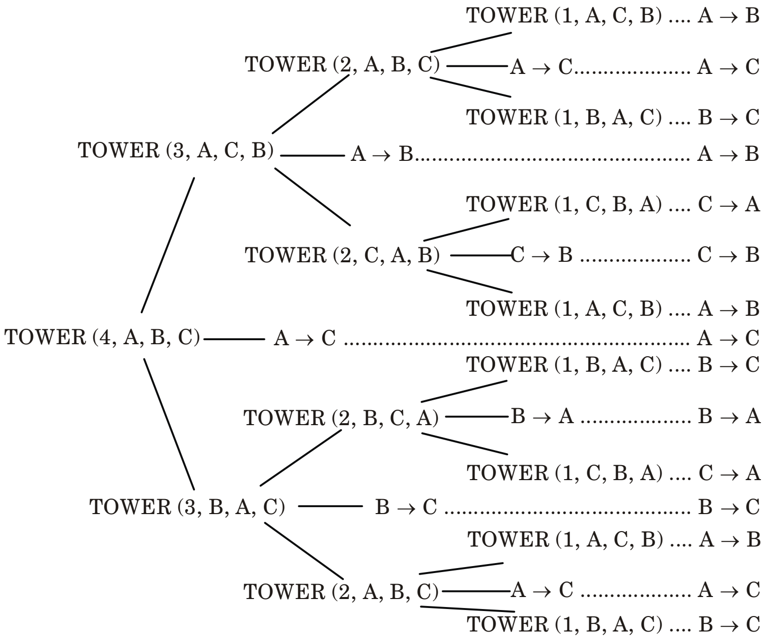


Fig. 13. Recursive solution to Tower of Hanoi problem for n = 4.

Observe that the recursive solution for $n = 4$ disks consist of the following 15 moves :

$A \to B$ $A \to C$ $B \to C$ $A \to B$ $C \to A$ $C \to B$ $A \to B$ $A \to C$ $B \to C$ $B \to A$ $C \to A$ $B \to C$ $A \to B$ $A \to C$ $B \to C$

12. Define a B-tree. What are the applications of B-tree ? Draw a B-tree of order 4 by insertion of the following keys in order : *Z, U, A, I, W, L, P, X, C, J, D, M, T, B, Q, E, H, S, K, N, R, G, Y, F, O, V*.

Ans. B-tree :

1. A B-tree is a self-balancing tree data structure that keeps data sorted and allows searches, sequential access, insertions, and deletions in logarithmic time.
2. A B-tree of order m is a tree which satisfies the following properties :
   a. Every node has at most m children.
   b. Every non-leaf node (except root) has at least $\lceil m/2 \rceil$ children.
   c. The root has at least two children if it is not a leaf node.
   d. A non-leaf node with k children contains $k - 1$ keys.
   e. All leaves appear in the same level.

Application of B-tree : The main application of a B-tree is the organization of a huge collection of records into a file structure. The organization should be in such a way that any record in it can be searched very efficiently *i.e.*, insertion, deletion and modification operations can be carried out perfectly and efficiently.

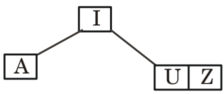
Construction of B-tree :

Insert Z : | Z |

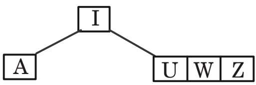
Insert U : | U | Z |

Insert A : | A | U | Z |

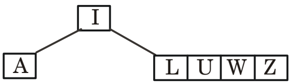
Insert I : | A | I | U | Z |

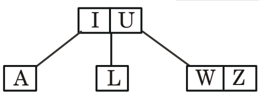


Insert W :

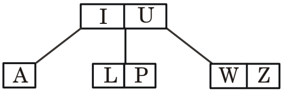


Insert L :

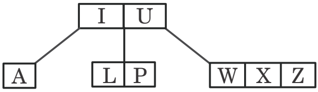




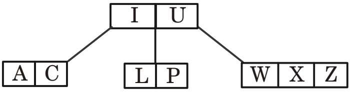
Insert P :



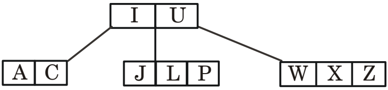
Insert X :



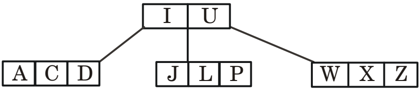
Insert C :



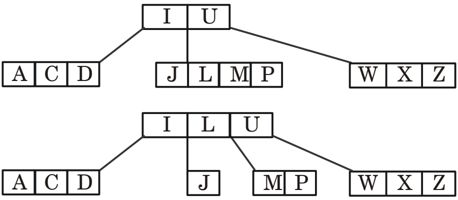
Insert J :



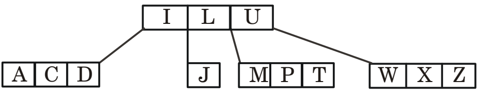
Insert D :



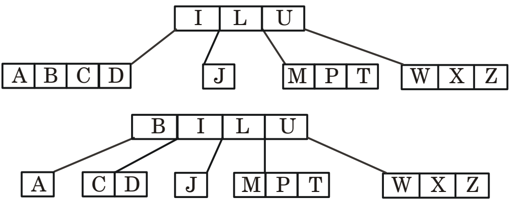
Insert M :

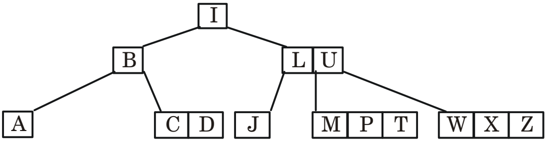


Insert T :

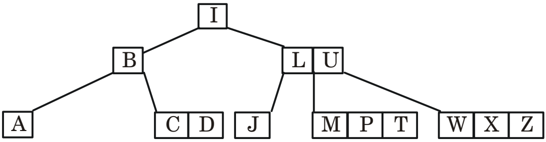


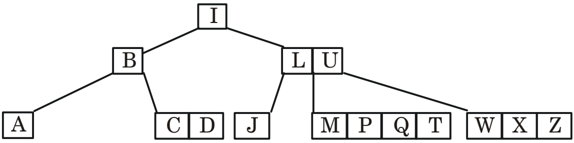
Insert B :



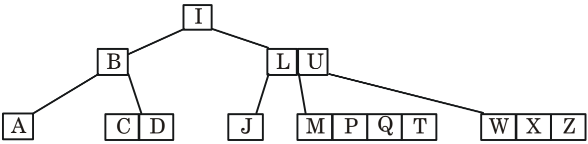


Insert Q :

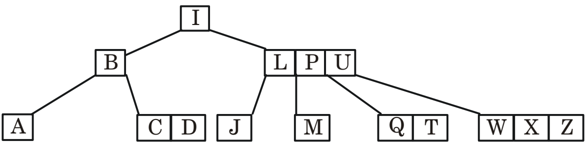


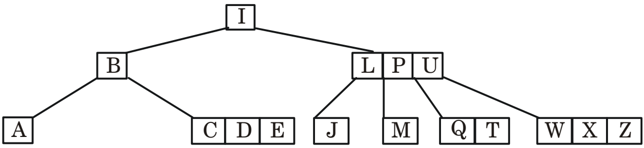


Insert E :

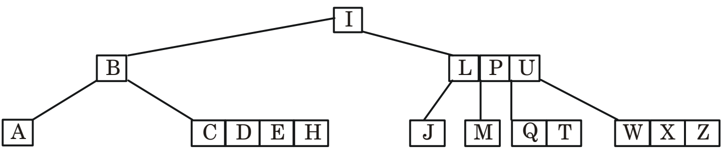


Insert H :

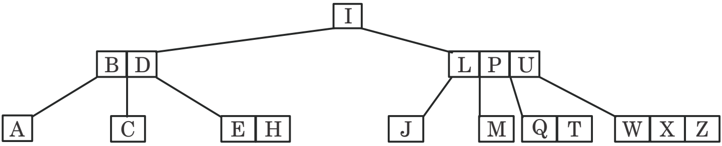




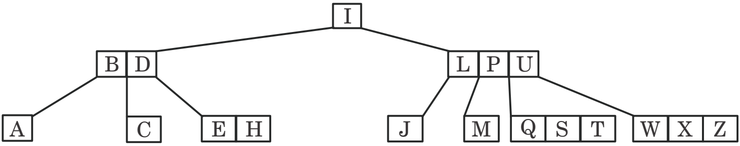
Insert S :

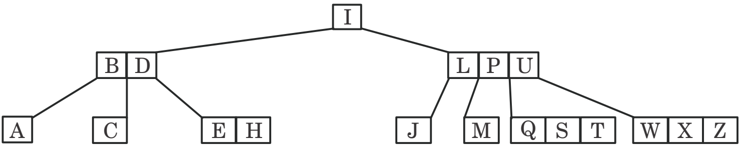


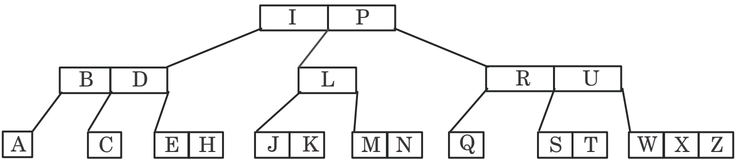
Insert K :



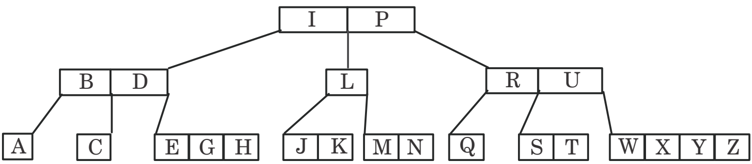
Insert N, R :

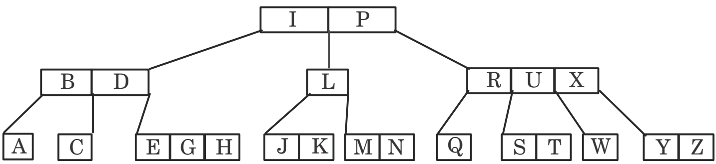




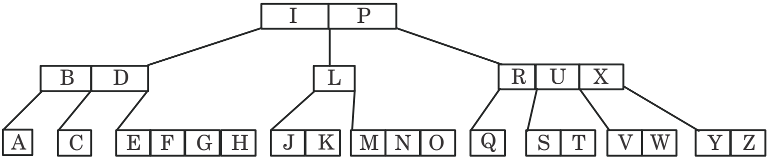


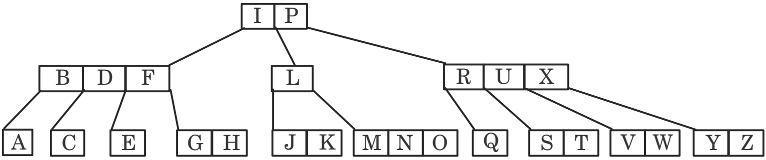
Insert G, Y :





Insert F, O & V :


I P
B D
L
R U X
A
C
E F G H
J K
M N O
Q
S T
V W
Y Z


I P
B D F
L
R U X
A
C
E
G H
J K
M N O
Q
S T
V W
Y Z

B.Tech.

(SEM. III) ODD SEMESTER THEORY EXAMINATION, 2016-17

DATA STRUCTURES USING 'C'

Time : 3 Hours **Max. Marks : 100**

Section-A

1. Attempt **all** parts. **All** parts carry equal marks. Write answer of each part in short. **(2 × 10 = 20)**

a. Define time complexity and space complexity of an algorithm.

b. What are the merits and demerits of array data structures ?

c. How do you push elements in a linked stack ?

d. Differentiate linear and non-linear data structures.

e. What is the significance of priority queue ?

f. Define complete binary tree. Give example.

g. When does a graph become tree ?

h. Prove that the number of odd degree vertices in a connected graph should be even.

i. What is sorting ? How is sorting essential for database applications ?

j. Give the worst case and best case time complexity of binary search.

Section-B

Note : Attempt any **five** questions from this section. **(10 × 5 = 50)**

2. What is recursion ? Write a recursive program to find sum of digits of the given number. Also, calculate the time complexity.

3. Solve the following :

a. $((A - (B + C) * D) / (E + F))$ **[Infix to postfix]**

b. $(A + B) + *C - (D - E) \wedge F$ **[Infix to prefix]**

c. 7 5 2 + * 4 1 5 – / – [Evaluate the given postfix expression]

4. Write a C program to implement the array representation of circular queue.

5. Write a C program to implement binary tree insertion, deletion with example.

6. Write the C program for various traversing techniques of binary tree with neat example.

7. What is quick sort ? Sort the given values using quick sort; present all steps/iterations :
38, 81, 22, 48, 13, 69, 93, 14, 45, 58, 79, 72

8. Illustrate the importance of various traversing techniques in graph along with its applications.

9. Compare and contrast the difference between B+ tree index files and B-tree index files with an example.

Note : Attempt any **two** questions from this section. **(15 × 2 = 30)**

10. What is meant by circular linked list ? Write the functions to perform the following operations in a doubly linked list.

a. Creation of list of nodes.

b. Insertion after a specified node.

c. Delete the node at a given position.

d. Sort the list according to descending order.

e. Display from the beginning to end.

11. Define AVL trees. Explain its rotation operations with example. Construct an AVL tree with the values 10 to 1 numbers into an initially empty tree.

12. Discuss Prim's and Kruskal's algorithm. Construct minimum spanning tree for the below given graph using Prim's algorithm (Source node = a).

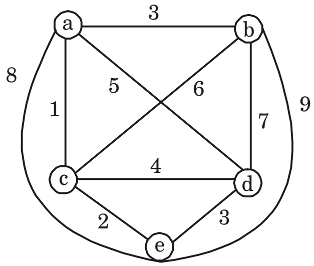


Fig. 1.

☺☺☺

SOLUTION OF PAPER (2016-17)

Section-A

1. Attempt **all** parts. **All** parts carry equal marks. Write answer of each part in short. **(2 × 10 = 20)**

a. Define time complexity and space complexity of an algorithm.

Ans. **Time complexity :** Time complexity is the amount of time it needs to run to completion.

Space complexity : Space complexity is the amount of memory it needs to run to completion.

b. What are the merits and demerits of array data structures ?

Ans. **Merits of array :**

1. Array is a collection of elements of similar data type.
2. Hence, multiple applications that require multiple data of same data type are represented by a single name.

Demerits of array :

1. Linear arrays are static structures, *i.e.*, memory used by them cannot be reduced or extended.
2. Previous knowledge of number of elements in the array is necessary.

c. How do you push elements in a linked stack ?

Ans. To insert an element onto stack is known as PUSH operation. Before inserting first we increase the top pointer and then insert the element.

d. Differentiate linear and non-linear data structures.

Ans.

S. No.	Linear data structure	Non-linear data structure
1.	It is a data structure whose elements form a sequence.	It is a data structure whose elements do not form a sequence.
2.	Every element in the structure has a unique predecessor and unique successor.	There is no unique predecessor or unique successor.
3.	Examples of linear data structure are arrays, linked lists, stacks and queues.	Examples of non-linear data structures are trees and graphs.

e. What is the significance of priority queue ?

Ans. Priority queue is a data structure in which elements can be stored as per their priorities. And therefore one can remove the elements from such queue according to their priorities. Such type of queue is useful to operating system in job scheduling algorithms.

f. Define complete binary tree. Give example.

Ans. A tree is called complete binary tree if tree satisfies following conditions :

1. Each node has exactly two children except leaf node.
2. All leaf nodes are at same level.
3. If a binary tree contains m nodes at level l, it contains at most $2m$ nodes at level $l + 1$.

Example :

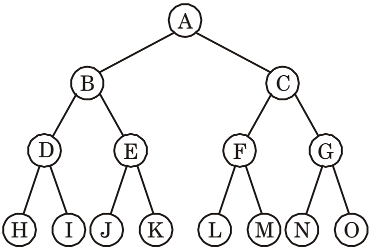


Fig. 1.

g. When does a graph become tree ?

Ans. A graph becomes a tree when there is exactly one path between every pair of its vertices.

h. Prove that the number of odd degree vertices in a connected graph should be even.

Ans. Let V_1 and V_2 be the set of vertices of even and odd degrees respectively. Thus, $V_1 \cap V_2 = \phi$ and $V_1 \cup V_2 = V$.

By Handshaking theorem,

$$2|E| = \sum_{v \in V} \deg(v) = \sum_{v \in V_1} \deg(v) + \sum_{v \in V_2} \deg(v)$$

As both $2|E|$ and $\sum_{v \in V_1} \deg(v)$ are even. So, $\sum_{v \in V_2} \deg(v)$ must be even.

Since, $\deg(v)$ is odd for all $v \in V_2$. So, the number of odd degree vertices in a connected graph must be even.

i. What is sorting ? How is sorting essential for database applications ?

Ans. **Sorting :** It is an operation which is used to put the elements of list in a certain order. *i.e.*, either in decreasing or increasing order.

Sorting essential for database applications : It is easier and faster to locate items in a sorted list than unsorted. Sorting algorithms can be used in a program to sort an array for later searching or writing out to an ordered file or report. Sorted arrays/ lists make it easier to find things more quickly.

j. Give the worst case and best case time complexity of binary search.

Ans. **Worst case :** In each comparison, the size of the search area is reduced by half. So, the efficiency of the binary search method at the worst case is $\log_2 n + 1$, *i.e.*, $O(\log_2 n + 1)$ where n is the total number of items that will be used for the binary search.

Best case : The best case of binary search occurs when the element we are searching for is the middle element of the list/array because in that case we will get the desired result in a single go. In this case, the time complexity of the algorithm will be O(1).

Section-B

Note : Attempt any **five** questions from this section. **(10 × 5 = 50)**

2. What is recursion ? Write a recursive program to find sum of digits of the given number. Also, calculate the time complexity.

Ans. **Recursion :**

1. Recursion is a process of expressing a function that calls itself to perform specific operation.
2. Indirect recursion occurs when one function calls another function that then calls the first function.

Program :

```
#include<stdio.h>
#include<conio.h>
int sum(int n)
{
    if(n < 10)
    return(n);
    else
    return(n % 10 + sum (n / 10));
}
main()
{
int s,n;
printf("\nEnter any number:");
scanf("%d",&n);
s = sum(n);
printf("\nSum of digits = %d", s);
```

```
getch();
return 0;
}
```

Time complexity :

i. Assume that n is a 10 digit number. The function is called 10 times as the problem is reduced by a factor of 10 each time the program recurse.

ii. So, we can conclude that time taken by program is linear in terms of the length of the digit of the input number n.

iii. So, time complexity is,
T(n) = O(length of digit of (n)) where n is the number whose sum of individual digit is to be found.

3. Solve the following :

a. ((*A* – (*B* + *C*) * *D*) / (*E* + *F*)) [Infix to postfix]

b. (*A* + *B*) + **C* – (*D* – *E*) ^ *F* [Infix to prefix]

c. 7 5 2 + * 4 1 5 – / – [Evaluate the given postfix expression]

Ans.

a. ((A – (B + C)*D)/(E + F))

$((A - (B + C)*D)/\underbrace{(EF +)}_{X})$

((A – (B + C)*D)/X)

$((A - \underbrace{(BC +)}_{Y} * D) / X)$

((A – (Y * D)) / X

$((A - \underbrace{(YD *)}_{Z}) / X)$

((A – Z)) / X)

$(\underbrace{(AZ -)}_{T} / X)$

(T / X)

T X /

Now put the values,

AZ – EF + /

AYD * – EF + /

ABC + D * – EF + /

This is the required postfix form.

b. (A + B) + *C – (D – E) ∧ F

$\underbrace{(+ AB)}_{X} + * C - (D - E) \wedge F$

X + * C – (D – E) ∧ F

$X + * C - \underbrace{(- DE)}_{Y} \wedge F$

X + * C – (Y ∧ F)

$X + * C - \underbrace{(Y \wedge F)}_{Z}$

$X + *(C - Z)$

$X + * \underbrace{(- CZ)}_{T}$

$X + * T$

$* + XT$

Now put the values,

$* + + AB - CZ$

$* + + AB - C \wedge YF$

$* + + AB - C \wedge - DEF$

This is the required prefix form.

c. 752 + * 415 – / – :

First this expression is converted into infix expression as :

Symbol scanned	Stack
7	7
5	7, 5
2	7, 5, 2
+	7, 5 + 2
*	7*(5 + 2)
4	7*(5 + 2), 4
1	7*(5 + 2), 4, 1
5	7*(5 + 2), 4, 1, 5
–	7*(5 + 2), 4, 1 – 5
/	7*(5 + 2), 4/(1 – 5)
–	(7*(5 + 2)) – (4/(1 – 5))

2
5
7

7, 5, 2 inserted

5 + 2 = 7
7

+ occurred

7 * 7 = 49

* occurred

4
49

4 inserted

1
4
49

1 inserted

5
1
4
49

5 inserted

1 – 5 = – 4
4
49

– occurred

4 / – 4 = – 1
49

/ occurred

49 – (– 1) = 50

– occurred

Hence, the value is 50

4. Write a C program to implement the array representation of circular queue.

Ans.

```
#include<stdio.h>
#include<conio.h>
#include<process.h>
#define MAX 10
    typedef struct {
        int front, rear ;
        int elements [MAX];
        } queue;
    void createqueue (queue *aq) {
        aq -> front = aq -> rear = – 1
    }
    int isempty (queue *aq)
    {
        if(aq -> front = = – 1)
            return 1;
        else
            return 0;
    }
    int isfull (queue *aq) {
        if(((aq -> front = = 0) && (aq -> rear = = MAX – 1))
                    | |(aq - > front == aq - > rear + 1))
                return 1;
        else
                return 0;
    }
    void insert (queue *aq, int value) {
        if(aq -> front = = – 1)
            aq -> front = aq -> rear = 0;
        else
            aq -> rear = (aq -> rear + 1) % MAX;
            aq -> element [aq -> rear] = value;
    }
    int delete (queue *aq) {
        int temp;
        temp = aq -> element [aq -> front];
        if(aq -> front = = aq ->rear)
        aq -> front = aq -> rear = – 1;
        else
        aq -> front = (aq -> front + 1) % MAX ;
        return temp;
        }
    void main( )
    {
        int ch, elmt;
```

```
queue q;
create queue (&q);
while (1) {
printf("1. Insertion \n");
printf("2. Deletion \n");
printf("3. Exit \n");
printf("Enter your choice");
scanf("%d",&ch) ; .
        switch (ch)
                {
                    case 1:
                    if(isfull (&q))
                    {
                    printf ("queue is full");
                    getch();
                    }
                    else
                    {
                    printf("Enter value");
                    scanf("%d", &elmt) ;
                    insert (&q, elmt) ;
                    }
                    break;
                    case 2: if (isempty (&q))
                    {
                    printf("queue empty");
                    getch();
                    }
                    else
                    {
                    printf("Value deleted is % d", delete (&q));
                    getch( );
                    }
                    break;
                    case 3:
                    exit(1);
                }
}   }
```

5. Write a C program to implement binary tree insertion, deletion with example.

Ans.

```
#include<stdlib.h>
#include<stdio.h>
struct bin_tree {
int data;
struct bin_tree *right, *left;
```

```
};
typedef struct bin_tree node;
void insert(node *tree, int val)
{
node *temp = NULL;
if(!(*tree))
{
temp = (node *)malloc(sizeof(node));
temp->left = temp->right = NULL;
temp->data = val;
*tree = temp;
return;
}
if(val < (*tree)->data)
{
insert(&(*tree)->left, val);
}
else if(val > (*tree)->data)
{
insert(&(*tree)->right, val);
}
}
void print_inorder(node *tree)
{
if (tree)
{
print_inorder(tree->left);
printf("%d\n",tree->data);
print_inorder(tree->right);
}
}
void deltree(node *tree)
{
if (tree)
{
deltree(tree->left);
deltree(tree->right);
free(tree);
}
}
void main()
{
node *root;
node *tmp;
//int i;
root = NULL;
/* Inserting nodes into tree */
```

```
insert(&root, 9);
insert(&root, 4);
insert(&root, 15);
insert(&root, 6);
insert(&root, 12);
insert(&root, 17);
insert(&root, 2);
/* Printing nodes of tree */
printf("After insertion inorder display\n");
print_inorder(root);
/* Deleting all nodes of tree */
deltree(root);
printf("Tree is empty");
}
```

Output of program :

```
After insertion inorder display
2
4
6
9
12
15
17
Tree is empty.
```

6. Write the C program for various traversing techniques of binary tree with neat example.

Ans.

```
#include<stdio.h>
#include<stdlib.h>
struct node
{
int value;
node* left;
node* right;
};
struct node* root;
struct node* insert(struct node* r, int data);
void inorder(struct node* r);
void preorder(struct node* r);
void postorder(struct node* r);
int main()
{
root = NULL;
int n, v;
printf("How many data do you want to insert ?\n");
scanf("%d", &n);
for(int i=0; i<n; i++){
```

```
printf("Data %d: ", i+1);
scanf("%d", &v);
root = insert(root, v);
}
printf("Inorder Traversal :");
inorder(root);
printf("\n");
printf("Preorder Traversal :");
preorder(root);
printf("\n");
printf("Postorder Traversal :");
postorder(root);
printf("\n");
return 0;
}
struct node* insert(struct node* r, int data)
{
if(r==NULL)
{
r = (struct node*) malloc(sizeof(struct node));
r->value = data;
r->left = NULL;
r->right = NULL;
}
else if(data < r->value){
r->left = insert(r->left, data);
}
else {
r->right = insert(r->right, data);
}
return r;
}
void inorder(struct node* r)
{
if(r!=NULL){
inorder(r->left);
printf("%d ", r->value);
inorder(r->right);
}
}
void preorder(struct node* r)
{
if(r!=NULL){
printf("%d", r->value);
preorder(r->left);
preorder(r->right);
}
```

```
}
void postorder(struct node* r)
{
if(r!=NULL){
postorder(r->left);
postorder(r->right);
printf("%d", r->value);
}
}
```

Output :

How many data do you want to insert ?
5
Preorder Traversal :
3 2 1 4 5
Inorder Traversal :
1 2 3 4 5
Postorder Traversal :
1 2 5 4 3

7. What is quick sort ? Sort the given values using quick sort; present all steps/iterations :
38, 81, 22, 48, 13, 69, 93, 14, 45, 58, 79, 72

Ans. Quick sort :

1. Quick sort is a sorting algorithm that also uses the idea of divide and conquer.
2. This algorithm finds the elements, called pivot, that partitions the array into two halves in such a way that the elements in the left sub-array are less than and the elements in the right sub-array are greater than the partitioning element.
3. Then these two sub-arrays are sorted separately. This procedure is recursive in nature with the base criteria.

Numerical : A = 38, 81, 22, 48, 13, 69, 93, 14, 45, 58, 79, 72. Choose the pivot element to be the element in position (left + right)/2.

During the partitioning process,

1. Elements strictly to the left of position *lo* are less than or equivalent to the pivot element (69).
2. Elements strictly to the right of position *hi* are greater than the pivot element. When *lo* and *hi* cross, we are done. The final value of *hi* is the position in which the partitioning element ends up.

Swap pivot element with leftmost element *lo* = left + 1; *hi* = right;

left	left+1										right
38	81	22	48	13	69	93	14	45	58	79	72

Move *hi* left and *lo* right as far as we can; then swap $A[lo]$ and $A[hi]$, and move *hi* and *lo* one more position.

lo *hi*←*hi*←*hi*

69	81*	22	48	13	38	93	14	45	58*	79*	72*

Repeat above

lo → *lo* → *lo* → *lo* → *lo* *hi*

69	58	22*	48*	13*	38*	93*	14	45*	81	79	72

Repeat above until *hi* and *lo* cross; then *hi* is the final position of the pivot element, so swap *A*[*hi*] and *A*[*left*].

hi

lo→*lo*

69	58	22	48	13	38	45	14**	93*	81	79	72

Partitioning complete; return value of *hi*.

hi

14	58	22	48	13	38	45	69	93	81	79	72

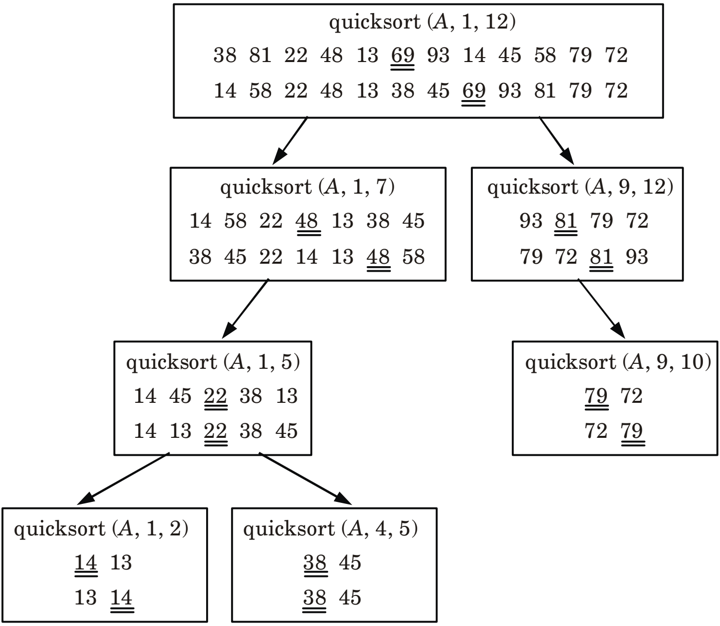


Fig. 2.

8. Illustrate the importance of various traversing techniques in graph along with its applications.

Ans. Various types of traversing techniques are :
1. Breadth First Search (BFS)
2. Depth First Search (DFS)

Importance of BFS :
1. It is one of the single source shortest path algorithms, so it is used to compute the shortest path.
2. It is also used to solve puzzles such as the Rubik's Cube.
3. BFS is not only the quickest way of solving the Rubik's Cube, but also the most optimal way of solving it.

Application of BFS : Breadth first search can be used to solve many problems in graph theory, for example :
1. Copying garbage collection.
2. Finding the shortest path between two nodes u and v, with path length measured by number of edges (an advantage over depth first search).
3. Ford-Fulkerson method for computing the maximum flow in a flow network.
4. Serialization/Deserialization of a binary tree vs serialization in sorted order, allows the tree to be re-constructed in an efficient manner.
5. Construction of the failure function of the Aho-Corasick pattern matcher.
6. Testing bipartiteness of a graph.

Importance of DFS : DFS is very important algorithm as based upon DFS, there are $O(V + E)$-time algorithms for the following problems :
1. Testing whether graph is connected.
2. Computing a spanning forest of G.
3. Computing the connected components of G.
4. Computing a path between two vertices of G or reporting that no such path exists.
5. Computing a cycle in G or reporting that no such cycle exists.

Application of DFS : Algorithms that use depth first search as a building block include :
1. Finding connected components.
2. Topological sorting.
3. Finding 2-(edge or vertex)-connected components.
4. Finding 3-(edge or vertex)-connected components.
5. Finding the bridges of a graph.
6. Generating words in order to plot the limit set of a group.
7. Finding strongly connected components.

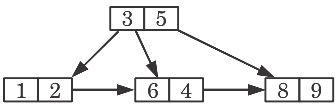
9. Compare and contrast the difference between B+ tree index files and B-tree index files with an example.

Ans.

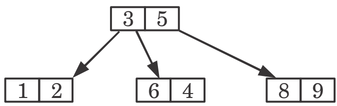
S. No.	Basis	B$^+$ tree	B-tree
1.	Definition	B$^+$ tree is an n-array tree with a variable but often large number of children per node. A B$^+$ tree consists of a root, internal nodes and leaves. The root may be either a leaf or a node with two or more children.	A B-tree is an organizational structure for information storage and retrieval in the form of a tree in which all terminal nodes are at the same distance from the base, and all non-terminal nodes have between n and $2n$ sub-trees or pointers (where n is an integer).
2.	Space complexity	O(n)	O(n)
3.	Storage	In a B$^+$ tree, data is stored only in leaf nodes.	In a B-tree, search keys and data are stored in internal or leaf nodes.
4.	Data	The leaf nodes of the tree store the actual record rather than pointers to records.	The leaf nodes of the tree store pointers to records rather than actual records.
5.	Space	These trees do not waste space.	These trees waste space.
6.	Function of leaf nodes	In B$^+$ tree, leaf node data are ordered in a sequential linked list.	In B-tree, the leaf node cannot store using linked list.
7.	Searching	In B$^+$ tree, searching of any data is very easy because all data is found in leaf nodes.	In B-tree, searching becomes difficult as data cannot be found in the leaf node.
8.	Search accessibility	In B$^+$ tree, the searching becomes easy.	In B-tree, the search is not that easy as compared to a B$^+$ tree.
9.	Redundant key	They store redundant search key.	They do not store redundant search key.

Example :

B$^+$ tree :



B-tree :



Note : Attempt any **two** questions from this section. **(15 × 2 = 30)**

10. What is meant by circular linked list ? Write the functions to perform the following operations in a doubly linked list.

a. Creation of list of nodes.

b. Insertion after a specified node.

c. Delete the node at a given position.

d. Sort the list according to descending order.

e. Display from the beginning to end.

Ans. **Circular linked list :** A circular list is a linear linked list, except that the last element points to the first element, Fig. 3 shows a circular linked list with 4 nodes for non-empty circular linked list, there are no NULL pointers.

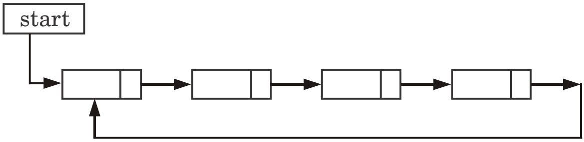


Fig. 3.

Functions :

a. To create a list :

```
void create ( )
{
struct node *ptr, *cpt ;
char ch ;
ptr = (struct node *) malloc (size of (struct node)) ;
printf (“Input first node information”) ;
scanf (“%d”, & ptr → info) ;
ptr → lpt = NULL ;
first = ptr ;
do
{
cpt = (struct node *) malloc (size of (struct node)) ;
printf (“Input next node information”);
scanf (“%d”, & cpt → info) ;
ptr → rpt = cpt ;
cpt → lpt = ptr ;
ptr = cpt ;
printf (“Press <Y/N> for more node”) ;
ch = getch ( );
}
while (ch == ‘Y’) ;
ptr → rpt = NULL ;
}
```

b. To insert after a specific node :

```
void insert_given_node ( )
```

```
{
struct node *ptr, *cpt, *tpt, *rpt, *lpt;
int m;
ptr = (struct node *) malloc (size of (struct node));
if (ptr == NULL)
{
printf (“OVERFLOW”);
return;
}
printf (“input new node information”);
scanf (“%d”, & ptr –> info);
printf (“input node information after which insertion”);
scanf (“%d”, & m);
cpt = first;
while (cpt –> info != m)
cpt = cpt –> rpt;
tpt = cpt –> rpt;
cpt –> rpt = ptr;
ptr –> lpt = cpt;
ptr –> rpt = tpt;
tpt –> lpt = ptr;
printf (“Insertion is done\n”);
}
```

c. To delete the node at a given position :

```
void deleteNode(int data) {
struct dllNode *nPtr, *tmp = head;
if (head == NULL) {
printf(“Data unavailable\n”);
return;
} else if (tmp->data == data) {
nPtr = tmp->next;
tmp->next = NULL;
free(tmp);
head = nPtr;
totNodes– –;
} else {
while (tmp->next != NULL && tmp->data != data) {
nPtr = tmp;
tmp = tmp->next;
}
if (tmp->next == NULL && tmp->data != data) {
printf(“Given data unavailable in list\n”);
return;
} else if (tmp->next != NULL && tmp->data == data) {
nPtr->next = tmp->next;
tmp->next->previous = tmp->previous;
```

```
tmp->next = NULL;
tmp->previous = NULL;
free(tmp);
printf("Data deleted successfully\n");
totNodes – –;
} else if (tmp->next == NULL && tmp->data == data) {
nPtr->next = NULL;
tmp->next = tmp->previous = NULL;
free(tmp);
printf("Data deleted successfully\n");
totNodes– –;
}
}
}
```

d. To sort the list according to descending order :

```
void insertionSort() {
struct dllNode *nPtr1, *nPtr2;
int i, j, tmp;
nPtr1 = nPtr2 = head;
for (i = 0; i < totNodes; i++) {
tmp = nPtr1->data;
for (j = 0; j < i; j++)
nPtr2 = nPtr2->next;
for (j = i; j > 0 && nPtr2->previous->data < tmp; j--) {
nPtr2->data = nPtr2->previous->data;
nPtr2 = nPtr2->previous;
}
nPtr2->data = tmp;
nPtr2 = head;
nPtr1 = nPtr1->next;
}
}
```

e. To display from the beginning to end :

```
void display()
{
if(head == NULL)
printf("\nList is Empty!!!");
else
{
struct Node *temp = head;
printf("\nList elements are: \n");
printf("NULL <--- ");
while(temp -> next != NULL)
{
printf("%d <===> ",temp -> data);
}
```

```
printf("%d ---> NULL", temp -> data);
}
}
```

11. Define AVL trees. Explain its rotation operations with example. Construct an AVL tree with the values 10 to 1 numbers into an initially empty tree.

Ans.

i. An AVL (or height balanced) tree is a balanced binary search tree.

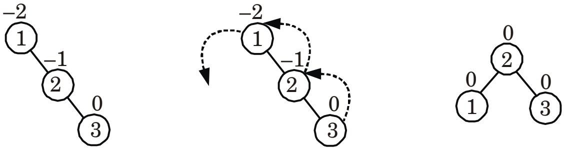
ii. In an AVL tree, balance factor of every node is either –1, 0 or +1.

iii. Balance factor of a node is the difference between the heights of left and right subtrees of that node.
Balance factor = height of left subtree – height of right subtree

iv. In order to balance a tree, there are four cases of rotations :

1. **Left Left rotation (LL rotation) :** In LL rotation every node moves one position to left from the current position.

Insert 1, 2 and 3



Tree is unbalanced

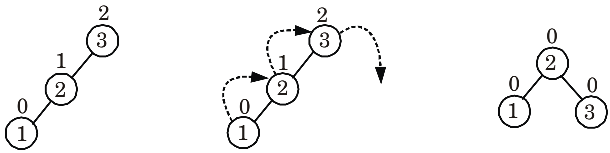
To make tree balance we use LL rotation which moves nodes one position to left

After LL rotation tree is balanced

Fig. 4.

2. **Right Right rotation (RR rotation) :** In RR rotation every node moves one position to right from the current position.

Insert 3, 2 and 1



Tree is unbalanced because node 3 has balance factor 2

To make tree balance we use RR rotation which moves nodes one position to right

After RR Rotation tree is balanced

Fig. 5.

3. **Left Right rotation (LR rotation) :** The LR Rotation is combination of single left rotation followed by single right rotation. In LR rotation, first every node moves one position to left then one position to right from the current position.

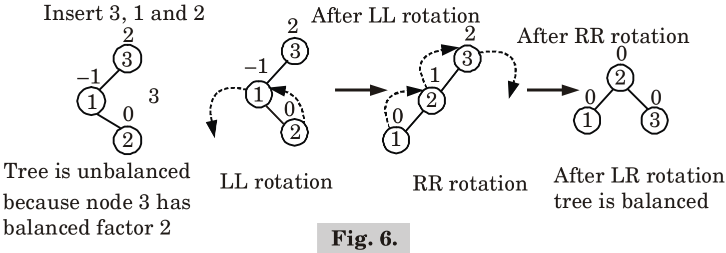


Fig. 6.

4. **Right Left rotation (RL rotation) :** The RL rotation is the combination of single right rotation followed by single left rotation. In RL rotation, first every node moves one position to right then one position to left from the current position.

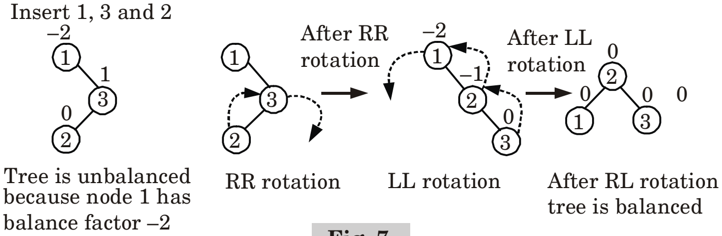
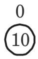


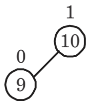
Fig. 7.

Numerical :

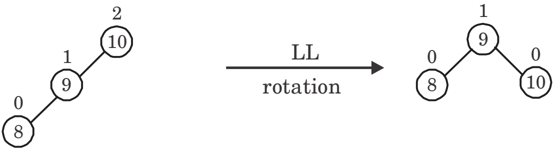
Insert 10 :



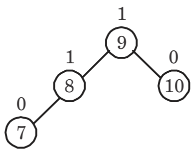
Insert 9 :



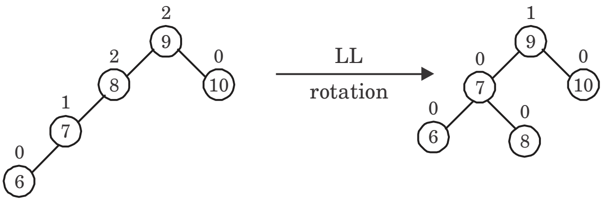
Insert 8 :



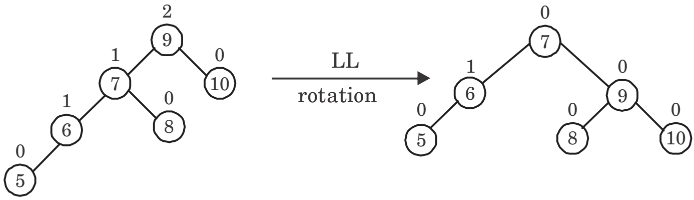
Insert 7 :



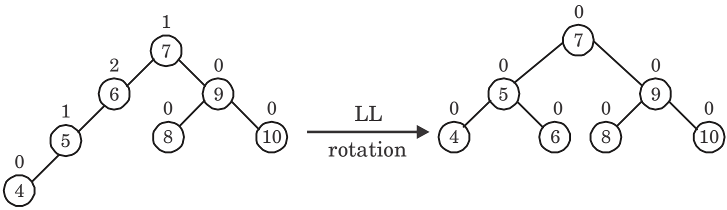
Insert 6 :



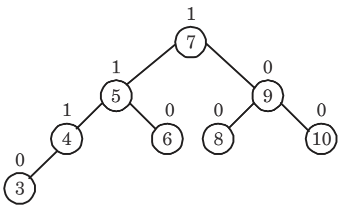
Insert 5 :



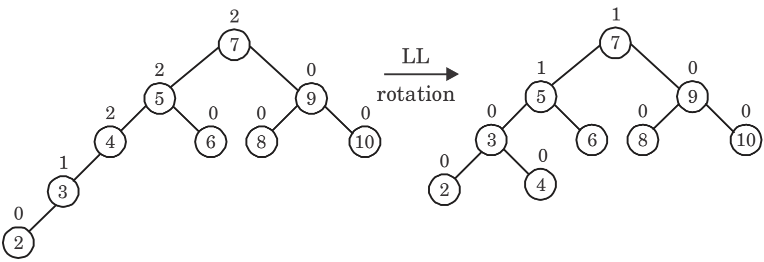
Insert 4 :



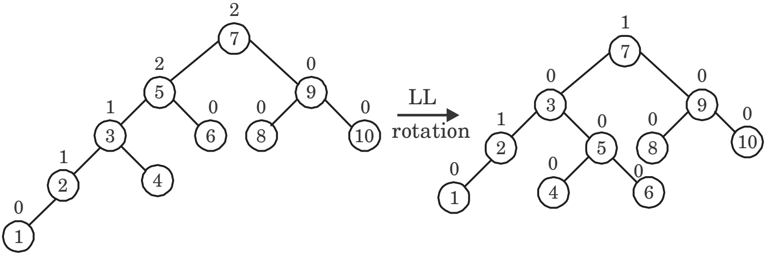
Insert 3 :



Insert 2 :



Insert 1 :



12. Discuss Prim's and Kruskal's algorithm. Construct minimum spanning tree for the below given graph using Prim's algorithm (Source node = a).

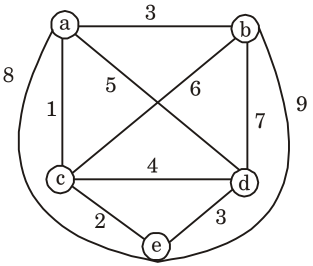


Fig. 8.

Ans. Prim's algorithm :

First it chooses a vertex and then chooses an edge with smallest weight incident on that vertex. The algorithm involves following steps :

Step 1 : Choose any vertex V_1 of G.

Step 2 : Choose an edge $e_1 = V_1V_2$ of G such that $V_2 \neq V_1$ and e_1 has smallest weight among the edge e of G incident with V_1.

Step 3 : If edges $e_1, e_2, \ldots\ldots, e_i$ have been chosen involving end points $V_1, V_2, \ldots\ldots\ldots\ldots, V_{i+1}$, choose an edge $e_{i+1} = V_jV_k$ with $V_j = \{V_1 \ldots\ldots V_{i+1}\}$ and $V_k \notin \{V_1 \ldots\ldots\ldots\ldots V_{i+1}\}$ such that e_{i+1} has smallest weight among the edges of G with precisely one end in $\{V_1 \ldots\ldots\ldots\ldots\ldots V_{i+1}\}$.

Step 4 : Stop after $n - 1$ edges have been chosen. Otherwise goto step 3.

Kruskal's algorithm :

i. In this algorithm, we choose an edge of G which has smallest weight among the edges of G which are not loops.

ii. This algorithm gives an acyclic subgraph T of G and the theorem given below proves that T is minimal spanning tree of G. Following steps are required :

Step 1 : Choose e_1, an edge of G, such that weight of e_1, $w(e_1)$ is as small as possible and e_1 is not a loop.

Step 2 : If edges $e_1, e_2, \ldots\ldots\ldots, e_i$ have been selected then choose an edge e_{i+1} not already chosen such that

i. the induced subgraph, $G[\{e_1 \ldots\ldots e_{i+1}\}]$ is acyclic and

ii. $w(e_{i+1})$ is as small as possible

Step 3 : If G has n vertices, stop after $n - 1$ edges have been chosen. Otherwise repeat step 2.

If G be a weighted connected graph in which the weight of the edges are all non-negative numbers, let T be a sub-graph of G obtained by Kruskal's algorithm then, T is minimal spanning tree.

Numerical :

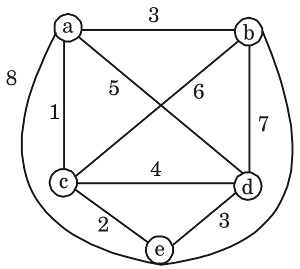


Fig. 9.

Start with source node = a

Now, edge with smallest weight incident on a is $e = (a, c)$.

So, we choose $e = (a, c)$.

Now we look on weights :

$$w(c, d) = 4, w(c, e) = 2, w(c, b) = 5$$

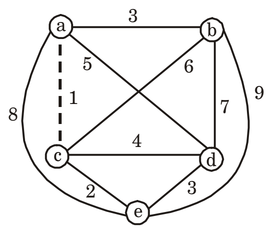


Fig. 10.

Since minimum is $w(c, e) = 2$. We choose $e = (c, e)$

Again, $w(e, d) = 3$

$w(e, a) = 8$

$w(e, b) = 7$

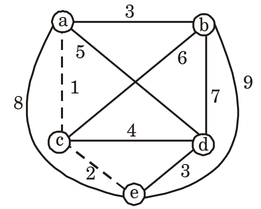


Fig. 11.

Since minimum is $w(e, d) = 3$, we choose $e = (e, d)$

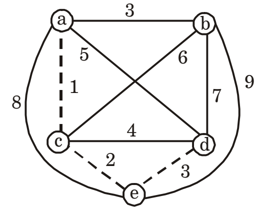


Fig. 12.

Now, $w(d, b) = 7$, we choose $w(d, b)$

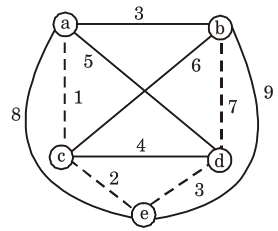


Fig. 13.

Therefore, the minimum spanning tree is :

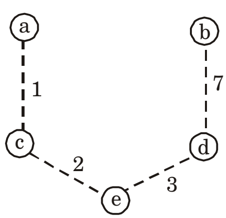


Fig. 14.

☺☺☺

B.Tech.

(SEM. III) ODD SEMESTER THEORY EXAMINATION, 2017-18

DATA STRUCTURES

Time : 3 Hours **Max. Marks : 70**

Note : Attempt **all** sections. Assume missing data, if any.

Section – A

1. Attempt **all** questions in brief. **(2 × 7 = 14)**

a. Define the term data structure. List some linear and non-linear data structures stating the application area where they will be used.

b. Discuss the concept of "successor" and "predecessor" in binary search tree.

c. Convert the following arithmetic infix expression into its equivalent postfix expression.
Expression : $A - B/C + D*E + F$

d. Explain circular queue. What is the condition if circular queue is full ?

e. Calculate total number of moves for Tower of Hanoi for $n = 10$ **disks.**

f. List the different types of representation of graphs.

g. Explain height balanced tree. List general cases to maintain the height.

Section-B

2. Attempt any **three** of the following : **(7 × 3 = 21)**

a. What do you understand by time space tradeoff ? Explain best, worst and average case analysis in this respect with an example.

b. Use quick sort algorithm to sort 15, 22, 30, 10, 15, 64, 1, 3, 9, 2. Is it a stable sorting algorithm? Justify.

c. Define spanning tree. Also construct minimum spanning tree using Prim's algorithm for the given graph.

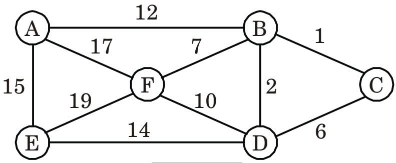


Fig. 1.

d. Define tree, binary tree, complete binary tree and full binary tree. Write algorithm or function to obtain traversals of a binary tree in preorder, postorder and inorder.

e. Construct a B-tree on following sequence of inputs.
10, 20, 30, 40, 50, 60, 70, 80, 90
Assume that the order of the B-tree is 3.

Section-C

3. Attempt any **one** part of the following : **(7 × 1 = 7)**

a. What are the various asymptotic notations ? Explain Big O notation.

b. Write an algorithm to insert a node at the end in a circular linked list.

4. Attempt any **one** part of the following : **(7 × 1 = 7)**

a. What is a stack ? Write a C program to reverse a string using stack.

b. Define the recursion. Write a recursive and non-recursive program to calculate the factorial of the given number.

5. Attempt any **one** part of the following : **(7 × 1 = 7)**

a. Draw a binary tree with following traversals :
Inorder : ***B C A E G D H F I J***
Preorder : ***A B C D E F G H I J***

b. Consider the following AVL tree and insert 2, 12, 7 and 10 as new node. Show proper rotation to maintain the tree as AVL.

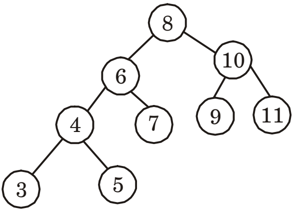


Fig. 2.

6. Attempt any **one** part of the following : **(7 × 1 = 7)**

a. What is a threaded binary tree ? Explain the advantages of using a threaded binary tree.

b. Describe Dijkstra's algorithm for finding shortest path. Describe its working for the graph given below.

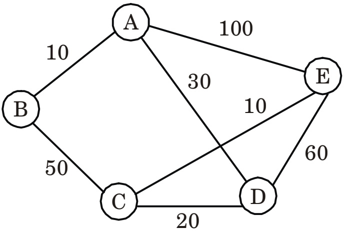


Fig. 3.

7. Attempt any **one** part of the following : **(7 × 1 = 7)**

a. Write short notes on :

i. Hashing technique

ii. Garbage collection

b. Explain the following : (7 × 1 = 7)

i. Heap sort

ii. Radix sort

☺☺☺

SOLUTION OF PAPER (2017-18)

Note : Attempt **all** sections. Assume missing data, if any.

Section – A

1. Attempt **all** questions in brief. **(2 × 7 = 14)**

a. Define the term data structure. List some linear and non-linear data structures stating the application area where they will be used.

Ans. It is a particular way of storing and organizing data in a computer so that it can be used efficiently.

It can be classified into two types :

i. Linear data structures :

1. Array 2. Stacks
3. Queue 4. Linked list

ii. Non-linear data structures :

1. Tree
2. Graph

b. Discuss the concept of "successor" and "predecessor" in binary search tree.

Ans. In binary search tree, if a node *X* has two children, then its predecessor is the maximum value in its left subtree and its successor is the minimum value in its right subtree.

c. Convert the following arithmetic infix expression into its equivalent postfix expression.

Expression : $A - B/C + D*E + F$

Ans. (A – B/C + D*E + F)

Character	Stack	Postfix
(	(	
A	(	A
–	(–	A
B	(–	AB
/	(– /	AB
C	(– /	ABC
+	(– +	ABC /
D	(– +	ABC / D
*	(– + *	ABC / D
E	(– + *	ABC /DE
+	(– ++	ABC / DE*
F	(– ++	ABC / DE*F
)	(	ABC / DE*F ++ –

d. Explain circular queue. What is the condition if circular queue is full ?

Ans. Circular queue : A circular queue is one in which the insertion of a new element is done at the very first location of the queue if the last location at the queue is full.

Syntax to check circular queue is full :

If ((front == MAX – 1) || (front == 0 && rear == MAX – 1))

e. Calculate total number of moves for Tower of Hanoi for n = 10 disks.

Ans. For n number of disks, total number of moves $= 2^n - 1$

For 10 disks, *i.e.*, $n = 10$, total number of moves $= 2^{10} - 1$

$= 1024 - 1$

$= 1023$

Therefore, if the Tower of Hanoi is operated on $n = 10$ disks, then total number of moves are 1023.

f. List the different types of representation of graphs.

Ans. Different types of representation of graphs :

1. Matrix representation
2. Linked representation

g. Explain height balanced tree. List general cases to maintain the height.

Ans.

i. An AVL (or height balanced) tree is a balanced binary search tree.

ii. In an AVL tree, balance factor of every node is either –1, 0 or +1.

iii. Balance factor of a node is the difference between the heights of left and right subtrees of that node.

Balance factor = height of left subtree – height of right subtree.

General cases to maintain the height are :

a. Left Left rotation (LL rotation)

b. Right Right rotation (RR rotation)

c. Left Right rotation (LR rotation)

d. Right Left rotation (RL rotation)

Section-B

2. Attempt any **three** of the following : **(7 × 3 = 21)**

a. What do you understand by time space tradeoff ? Explain best, worst and average case analysis in this respect with an example.

Ans. Time-space trade-off :

1. The time-space trade-off refers to a choice between algorithmic solutions of data processing problems that allows to decrease the

running time of an algorithmic solution by increasing the space to store data and vice-versa.

2. Time-space trade-off is basically a situation where either space efficiency (memory utilization) can be achieved at the cost of time or time efficiency (performance efficiency) can be achieved at the cost of memory.

Best, worst and average case analysis : Suppose we are implementing an algorithm that helps us to search for a record amongst a list of records. We can have the following three cases which relate to the relative success our algorithm can achieve with respect to time :

1. **Best case :**
   a. The record we are trying to search is the first record of the list.
   b. If $f(n)$ is the function which gives the running time and / or storage space requirement of the algorithm in terms of the size n of the input data, this particular case of the algorithm will produce a complexity $C(n) = 1$ for our algorithm $f(n)$ as the algorithm will run only 1 time until it finds the desired record.
2. **Worst case :**
   a. The record we are trying to search is the last record of the list.
   b. If $f(n)$ is the function which gives the running time and / or storage space requirement of the algorithm in terms of the size n of the input data, this particular case of the algorithm will produce a complexity $C(n) = n$ for our algorithm $f(n)$, as the algorithm will run n times until it finds the desired record.
3. **Average case :**
   a. The record we are trying to search can be any record in the list.
   b. In this case, we do not know at which position it might be.
   c. Hence, we take an average of all the possible times our algorithm may run.
   d. Hence assuming for n data, we have a probability of finding any one of them is $1/n$.
   e. Multiplying each of these with the number of times our algorithm might run for finding each of them and then taking a sum of all those multiples, we can obtain the complexity $C(n)$ for our algorithm $f(n)$ in case of an average case as following :

$$C(n) = 1\cdot\frac{1}{2} + 2\cdot\frac{1}{2} + \ldots + n\cdot\frac{1}{2}$$

$$C(n) = (1 + 2 + \ldots + n)\cdot\frac{1}{2}$$

$$C(n) = \frac{n(n+1)}{2}\cdot\frac{1}{n} = \frac{n+1}{2}$$

Hence in this way, we can find the complexity of an algorithm for average case as

$$C(n) = O((n + 1)/2)$$

b. Use quick sort algorithm to sort 15, 22, 30, 10, 15, 64, 1, 3, 9, 2. Is it a stable sorting algorithm? Justify.

Ans.

Let A [ ] =

1	2	3	4	5	6	7	8	9	10
15	22	30	10	15	64	1	3	9	2

Here $p = 1, r = 10$

$x = \text{A}[10]$ *i.e.*, $x = 2$

$i = p - 1$ *i.e.*, $i = 0$

$j = 1$ to 9

Now, $j = 1$ and $i = 0$

$A[j] = A[1] = 15$ and $15 \le 2$

So, $j = 2$ and $i = 0$

$A[2] = 22 \le 2$ (False)

Now, $j = 3$ and $i = 0$

$A[3] = 30 \le 2$ (False)

$j = 4$ and $i = 0$

$A[4] = 10 \le 2$ (False)

$j = 5$

$A[5] = 15 \le 2$ (False)

$j = 6$

$A[6] = 64 \le 2$ (False)

$j = 7$

$A[7] = 1 \le 2$ (True)

$i = 0 + 1 = 1$

$A[1] \leftrightarrow A[7]$

i.e.,

1	2	3	4	5	6	7	8	9	10
1	22	30	10	15	64	15	3	9	2

$j = 8$ and $i = 1$

$A[8] = 3 \le 2$ (False)

$j = 9$ and $i = 1$

$A[9] = 9 \le 2$ (False)

then, $A[1 + 1] \leftrightarrow A[r]$

$A[2] \leftrightarrow A[10]$

$q \leftarrow 2$

i.e.,

1	2	3	4	5	6	7	8	9	10
1	2	30	10	15	64	15	3	9	22

QUICK SORT (A, 1, 1)

1	2
1	2

QUICK SORT (A, 3, 10)

3	4	5	6	7	8	9	10
30	10	15	64	15	3	9	22

Here $p = 3, r = 10$

$x = A[10] = 22$

$$i = 3 - 1 = 2$$
$$j = 3 \text{ to } 9; \quad j = 3 \text{ and } i = 2$$
$$A[3] = 30 \leq 22 \text{ (False)}$$
$$j = 4 \text{ and } i = 2$$
$$A[4] = 10 \leq 22 \text{ (True)}$$
$$i = 2 + 1 = 3 \text{ and } A[3] \leftrightarrow A[4]$$

i.e.,

3	4	5	6	7	8	9	10
10	30	15	64	15	3	9	22

$$j = 5 \text{ and } i = 3$$
$$A[5] = 15 \leq 22 \text{ (True)}$$
$$i = 3 + 1 = 4 \text{ and } A[4] \leftrightarrow A[5]$$

3	4	5	6	7	8	9	10
10	15	30	64	15	3	9	22

Similarly

$$j = 7, i = 4$$
$$A[7] = 15 \leq 22 \text{ (True)}$$
$$i = 4 + 1 = 5 \text{ and } A[5] \leftrightarrow A[7]$$

i.e.,

3	4	5	6	7	8	9	10
10	15	15	64	15	3	9	22

Similarly, we get another pivot element

10	15	15	3	9	22	64	30

Thus, this is a stable algorithm.

c. Define spanning tree. Also construct minimum spanning tree using Prim's algorithm for the given graph.

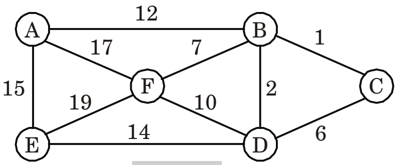
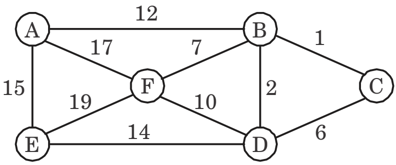


Fig. 1.

Ans. Spanning tree :

1. A spanning tree of graph is a sub-graph which is a tree and contains all the vertices of graph.

Numerical :

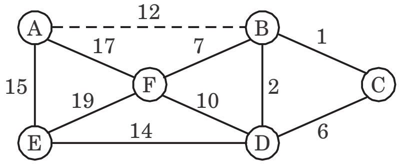


Let us take A as source node.

Now we look on weight

$w(A, B) = 12$, $w(A, F) = 17$, $w(A, E) = 15$

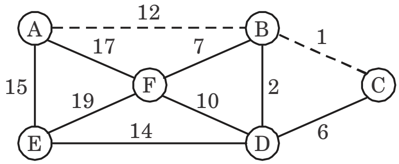
$\because$ $w(A, B)$ is smallest. Choose $e = (AB)$



Now we look on weight

$w(B, F) = 7$, $w(B, D) = 2$, $w(B, C) = 1$

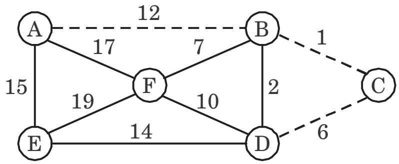
$\because$ $w(B, C)$ is smallest $\therefore$ choose $e = (BC)$



Now we look on weight

$w(C, D) = 6$

$\because$ $w(C, D)$ is smallest $\therefore$ choose $e = (CD)$



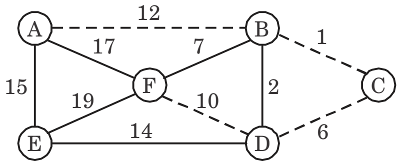
Now we look on weight

$w(D, B) = 2$, $w(D, F) = 10$, $w(D, E) = 14$

$\because$ $w(D, B)$ is smallest but forms a cycle

$\therefore$ Discard it.

Now $w(D, F) = 10$ is smallest $\therefore$ Choose $e = (DF)$



Now we look on weight

$w(F, B) = 7$, $w(F, A) = 17$, $w(F, E) = 19$

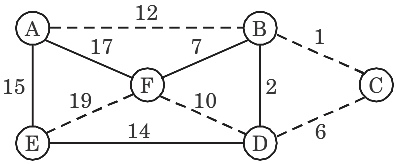
$\because$ $w(F, B)$ is smallest but forms cycle

$\therefore$ Discard it

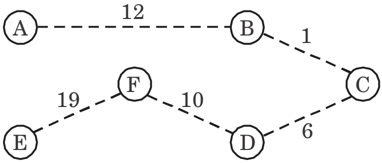
$\because$ $w(F, A)$ is smallest but forms cycle

$\therefore$ Discard it

$\therefore$ choose $e = (FE)$



The final minimum spanning tree is :



d. Define tree, binary tree, complete binary tree and full binary tree. Write algorithm or function to obtain traversals of a binary tree in preorder, postorder and inorder.

Ans. **Tree :** A tree T is a finite non-empty set of elements. One of these elements is called the root, and the remaining elements, if any is partitioned into trees is called subtree of T. A tree is a non-linear data structure.

Binary tree :

1. A binary tree T is defined as a finite set of elements called nodes, such that :
   a. T is empty (called the null tree).
   b. T contains a distinguished node R, called the root of T, and the remaining nodes of T form an ordered pair of disjoint binary trees T_1 and T_2.

Complete binary tree : A tree is called a complete binary tree if tree satisfies following conditions :

a. Each node has exactly two children except leaf node.
b. All leaf nodes are at same level.
c. If a binary tree contains m nodes at level l, it contains atmost $2m$ nodes at level $l + 1$.

Full binary tree :

A full binary tree is formed when each missing child in the binary tree is replaced with a node having no children.

Algorithm for preorder traversal :

Preorder (INFO, LEFT, RIGHT, ROOT)

1. [Initially push NULL onto STACK, and initialize PTR]
   Set TOP = 1, STACK [1] = NULL and PTR = ROOT
2. Repeat steps 3 to 5 while PTR $\neq$ NULL
3. Apply process to INFO [PTR]
4. [Right child?]
   If RIGHT [PTR] $\neq$ NULL
   Then

[Push on STACK]
Set TOP = TOP + 1 and
STACK [TOP] = RIGHT [PTR]
Endif
5. [Left child?]
If LEFT [PTR] ≠ NULL then
set PTR = LEFT[PTR]
Else
[Pop from STACK]
set PTR = STACK[TOP] and TOP = TOP – 1
Endif
End of step 2
6. Exit

Algorithm for inorder traversal :

Inorder (INFO, LEFT, RIGHT, ROOT)
1. [Push NULL onto STACK and initialize PTR]
Set TOP = 1, STACK[1] = NULL and PTR = ROOT
2. Repeat while PTR ≠ NULL
[Push leftmost path onto STACK]
a. Set TOP = TOP + 1 and
STACK [TOP] = PTR
b. Set PTR = LEFT [PTR]
End loop
3. Set PTR = STACK[TOP] and TOP = TOP – 1
4. Repeat steps 5 to 7 while PTR ≠ NULL
5. Apply process to INFO[PTR]
6. [Right Child?] If RIGHT [PTR] ≠ NULL
Then
a. Set PTR = RIGHT [PTR]
b. goto step 2
Endif
7. Set PTR = STACK[TOP] and TOP = TOP – 1
End of Step 4 Loop
8. Exit

Algorithm for postorder traversal :

Postorder (INFO, LEFT, RIGHT, ROOT)
1. [Push NULL onto STACK and initialize PTR]
Set TOP = 1, STACK[1] = NULL and PTR = ROOT
2. [Push leftmost path onto STACK]
Repeat steps 3 to 5 while PTR ≠ NULL
3. Set TOP = TOP + 1 and STACK [TOP] = PTR
[Pushes PTR on STACK]
4. If RIGHT [PTR] ≠ NULL
Then

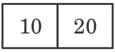
Set TOP = TOP + 1 and STACK [TOP] = RIGHT [PTR]
Endif
5. Set PTR = LEFT [PTR]
End of step 2 loop
6. Set PTR = STACK [TOP] and TOP = TOP – 1
[Pops node from STACK]
7. Repeat while PTR > 0
a. Apply process to INFO [PTR]
b. Set PTR = STACK [TOP] and TOP = TOP – 1
End loop
8. If PTR < 0 Then
a. Set PTR = – PTR
b. goto step 2
Endif
9. Exit

e. Construct a B-tree on following sequence of inputs.
10, 20, 30, 40, 50, 60, 70, 80, 90
Assume that the order of the B-tree is 3.

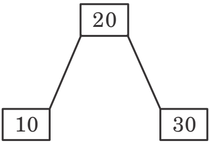
Ans. 10, 20, 30, 40, 50, 60, 70, 80, 90
Order of the *B*-tree is 3.

1. Insert 10 :

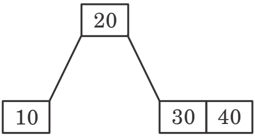
2. Insert 20 :



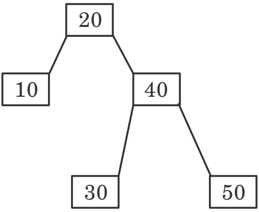
3. Insert 30 :



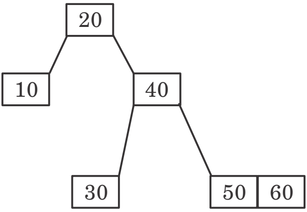
4. Insert 40 :



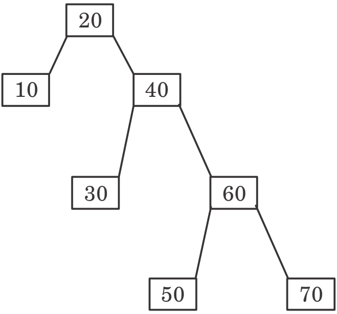
5. Insert 50 :



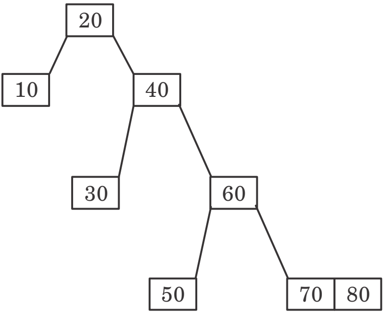
6. Insert 60 :



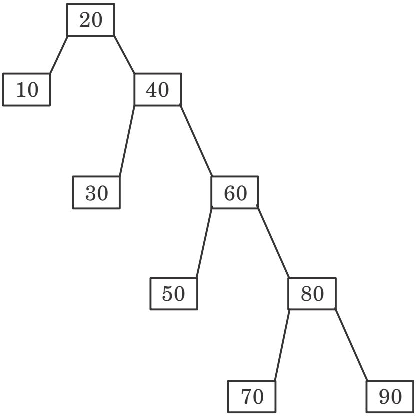
7. Insert 70 :



8. Insert 80 :



9. Insert 90 :



This is final B-tree of order 3.

Section-C

3. Attempt any **one** part of the following : **(7 × 1 = 7)**

a. What are the various asymptotic notations ? Explain Big O notation.

Ans. **Various asymptotic notations :**

1. θ-Notation (Same order)
2. Oh-Notation (Upper bound)
3. Ω-Notation (Lower bound)
4. Little - Oh notation (o)
5. Little omega notation (ω)

Big Oh-Notation :

1. Big-Oh is formal method of expressing the upper bound of an algorithm's running time.
2. It is the measure of the longest amount of time it could possibly take for the algorithm to complete.
3. More formally, for non-negative functions, $f(n)$ and $g(n)$, if there exists an integer n_0 and a constant $c > 0$ such that for all integers $n > n_0$.

$$f(n) \leq cg(n)$$

4. Then, $f(n)$ is Big-Oh of $g(n)$. This is denoted as : $f(n) \in O(g(n))$ *i.e.*, the set of functions which, as n gets large, grow faster than a constant time $f(n)$.

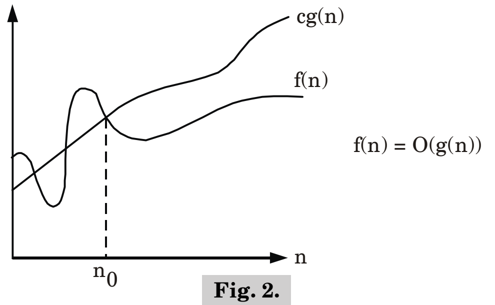


Fig. 2.

b. Write an algorithm to insert a node at the end in a circular linked list.

Ans.

1. If PTR = NULL
2. Write OVERFLOW
3. Go to Step 1
   [END OF IF]
4. SET NEW_NODE = PTR
5. SET PTR = PTR -> NEXT
6. SET NEW_NODE -> DATA = VAL
7. SET NEW_NODE -> NEXT = HEAD
8. SET TEMP = HEAD
9. Repeat Step 10 while TEMP -> NEXT != HEAD
10. SET TEMP = TEMP -> NEXT
    [END OF LOOP]
11. SET TEMP -> NEXT = NEW_NODE
12. EXIT

4. Attempt any **one** part of the following : **(7 × 1 = 7)**

a. What is a stack ? Write a C program to reverse a string using stack.

Ans.

1. A stack is one of the most commonly used data structure.
2. A stack, also called Last In First Out (LIFO) system, is a linear list in which insertion and deletion can take place only at one end, called top.
3. This structure operates in much the same way as stack of trays.
4. If we want to remove a tray from stack of trays it can only be removed from the top only.
5. The insertion and deletion operation in stack terminology are known as PUSH and POP operations.

Program to reverse a string using stack :

```
#include<stdio.h>
#include<conio.h>
#include<string.h>
#define MAX 20
    int top = – 1;
    char stack [MAX];
    char pop();
    push(char);
    main()
        {
        clrscr();
        char str [20];
        int i;
        printf("Enter the string : ");
        gets(str);
        for(i = 0; i < strlen(str); i++)
        push (str [i]);
        for(i = 0; i < strlen(str); i++)
        str[i] = pop();
        printf("Reversed string is :");
        puts (str);
        getch();
        }
    push (char item)
        {
        if(top == MAX – 1)
        printf("Stack overflow\n");
        else
        stack[++top] = item;
        }
```

```
char pop()
    {
    if(top == – 1)
    printf("Stack underflow \n");
    else
    return stack [top – –];
    }
```

b. Define the recursion. Write a recursive and non-recursive program to calculate the factorial of the given number.

Ans. **Recursion :**

1. Recursion is a process of expressing a function that calls itself to perform specific operation.
2. Indirect recursion occurs when one function calls another function that then calls the first function.

Program :

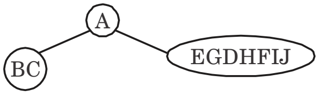
```
#include<stdio.h>
#include<conio.h>
void main()
{
int n, a, b;
clrscr();
printf("Enter any number\n");
scanf("%d", &n);
a = recfactorial(n);
printf("The factorial of a given number using recursion is %d
\n", a);
b = nonrecfactorial(n);
printf("The factorial of a given number using nonrecursion is
%d ", b);
getch();
}
int recfactorial(int x)
{
int f;
if(x == 0)
{
return(1);
}
else
{
f = x * recfactorial(x – 1);
return(f);
}
}
int nonrecfactorial(int x)
```

```
{
int i, f = 1;
for(i = 1; i <= x; i++)
{
f = f * i;
}
return(f);
}
```

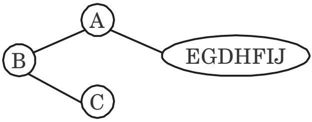
5. Attempt any **one** part of the following : **(7 × 1 = 7)**

a. Draw a binary tree with following traversals :
Inorder : *B C A E G D H F I J*
Preorder : *A B C D E F G H I J*

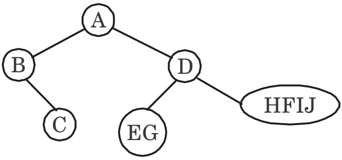
Ans. From preorder traversal, we get root node to be *A*.



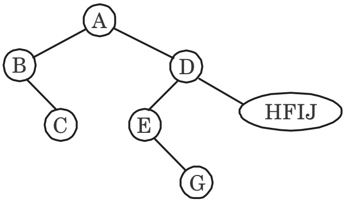
Now considering left subtree.
Observing both the traversal we can get *B* as root node and *C* as right child.



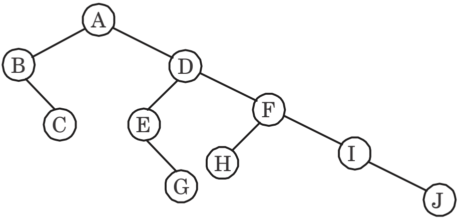
Now, consider the right subtree.
Preorder traversal is *DEGFHIJ*, which shows *D* is root node.
Inorder traversal is *EGDHFIJ*, which shows *EG* is left subtree and *HFIJ* is right subtree.



Now, consider the left subtree of *D*.
Preorder traversal is *EG* and inorder traversal is *EG*.
∴ *E* is root node and *G* is right subtree.



Similarly, following the same procedure, we finally get



b. Consider the following AVL tree and insert 2, 12, 7 and 10 as new node. Show proper rotation to maintain the tree as AVL.

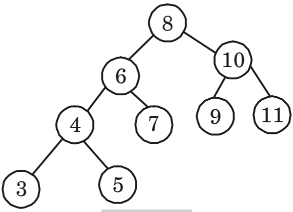
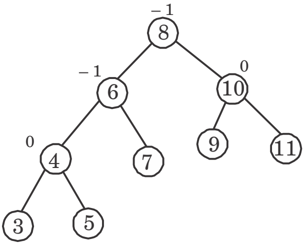


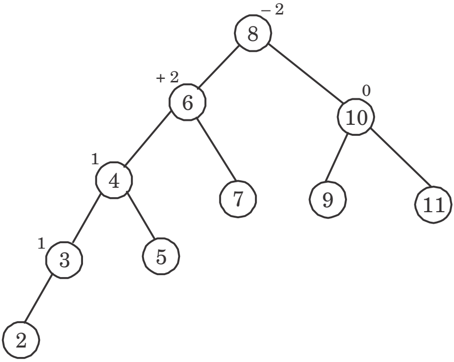
Fig. 3.

Ans. Given tree :

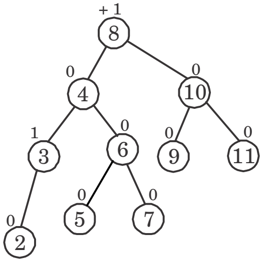


Balanced tree

Insert 2 :

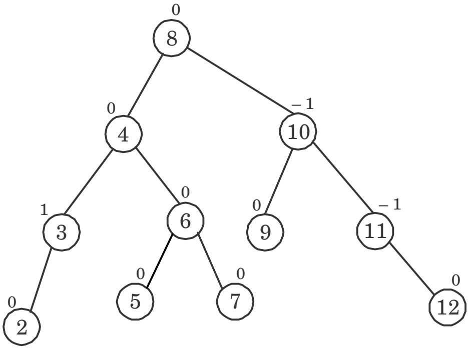


Tree is unbalanced, now LL rotation is required to balance it.



Now the tree is balanced.

Insert 12 :



Tree is balanced, so there is no need to balance the tree.

Insert 7 : 7 is already in the tree hence it cannot be inserted in the AVL tree.

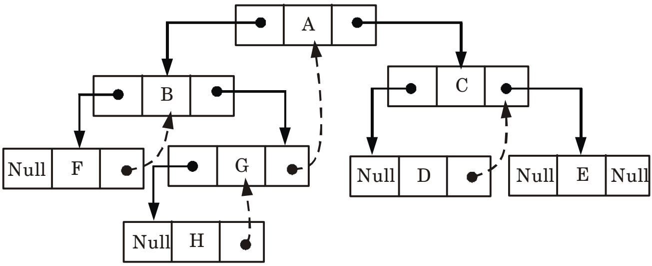
Insert 10 : 10 is also in the tree hence it cannot be inserted in the AVL tree.

6. Attempt any **one** part of the following : **(7 × 1 = 7)**

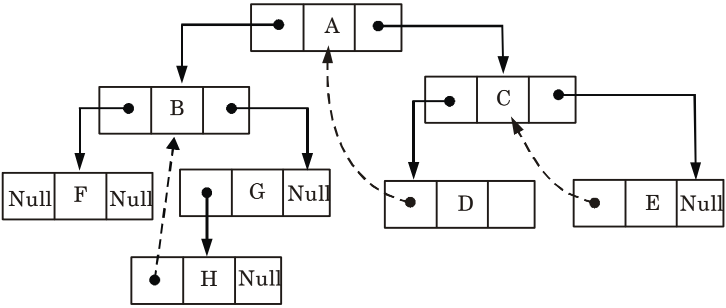
a. What is a threaded binary tree ? Explain the advantages of using a threaded binary tree.

Ans.

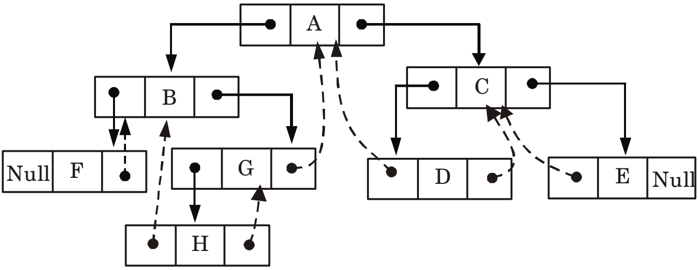
1. To make traversal of nodes more efficient we can utilize space occupied by the NULL pointers in the leaf nodes and internal nodes having only one child node.
2. These pointers can be modifies to point to their corresponding in-order successor, in-order predecessor or both.
3. These modified pointers are known as threads and binary trees having such type of pointers are known as threaded binary tree.



(a) Right threaded binary tree.



(b) Left threaded binary tree



(c) Fully threaded binary tree

Fig. 4.

Advantages of using threaded binary tree :

1. In threaded binary tree the traversal operations are very fast.
2. In threaded binary tree, we do not require stack to determine the predecessor and successor node.
3. In a threaded binary tree, one can move in any direction *i.e.*, upward or downward because nodes are circularly linked.
4. Insertion into and deletions from a threaded tree are all although time consuming operations but these are very easy to implement.

b. Describe Dijkstra's algorithm for finding shortest path. Describe its working for the graph given below.

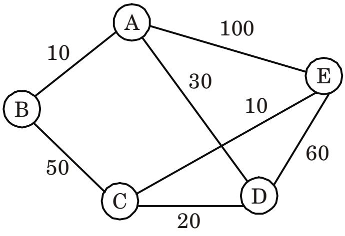


Fig. 5.

Ans. **Algorithm :**

a. Dijkstra's algorithm, is a greedy algorithm that solves the single-source shortest path problem for a directed graph $G = (V, E)$ with non-negative edge weights, *i.e.*, we assume that $w(u, v) \geq 0$ each edge $(u, v) \in E$.

b. Dijkstra's algorithm maintains a set S of vertices whose final shortest-path weights from the source s have already been determined.

c. That is, for all vertices $v \in S$, we have $d[v] = \delta(s, v)$.

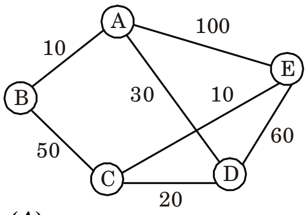
d. The algorithm repeatedly selects the vertex $u \in V - S$ with the minimum shortest-path estimate, inserts u into S, and relaxes all edges leaving u.

e. We maintain a priority queue Q that contains all the vertices in $v - s$, keyed by their d values.

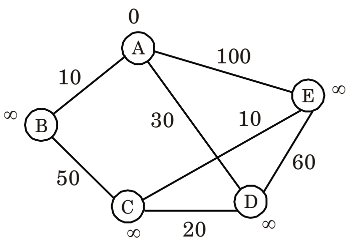
f. Graph G is represented by adjacency list.

g. Dijkstra's always chooses the "lightest or "closest" vertex in $V - S$ to insert into set S, that it uses as a greedy strategy.

Numerical :

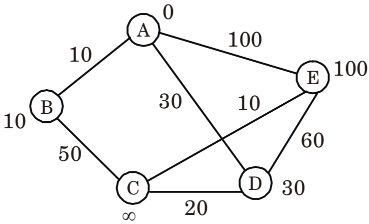


Extract min (A) :



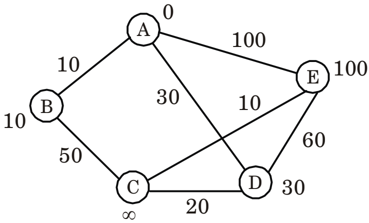
A	B	C	D	E
$\boxed{0}$	∞	∞	∞	∞

All edges leaving *A* :



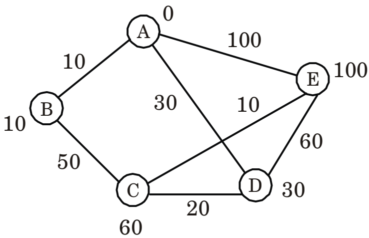
A	B	C	D	E
$\boxed{0}$	∞	∞	∞	∞
	10	∞	30	100

Extract min (*B*) :



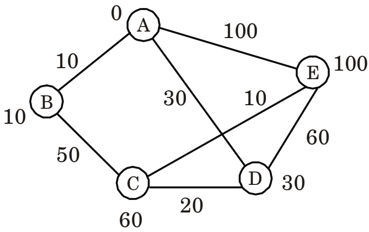
A	B	C	D	E
$\boxed{0}$	∞	∞	∞	∞
	$\boxed{10}$	∞	30	100

All edges leaving *B* :



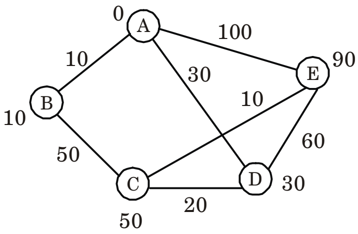
A	B	C	D	E
$\boxed{0}$	∞	∞	∞	∞
	$\boxed{10}$	∞	30	100
		60	30	100

Extract min(*D*) :



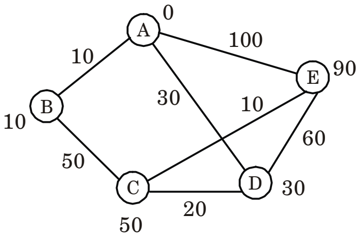
A	B	C	D	E
$\boxed{0}$	∞	∞	∞	∞
	$\boxed{10}$	∞	30	100
		60	$\boxed{30}$	100

All edges leaving (*D*) :



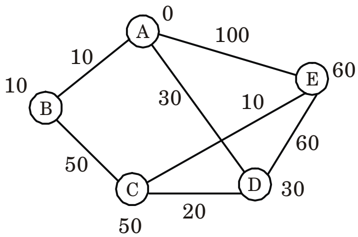
A	B	C	D	E
$\boxed{0}$	∞	∞	∞	∞
	$\boxed{10}$	∞	30	100
		60	$\boxed{30}$	100
		50		90

Extract min(*C*) :



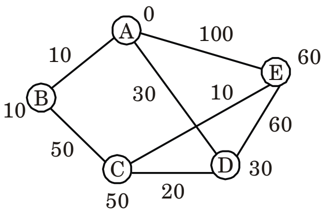
A	B	C	D	E
$\boxed{0}$	∞	∞	∞	∞
	$\boxed{10}$	∞	30	100
		60	$\boxed{30}$	100
		$\boxed{50}$		90

All edges leaving *C* :



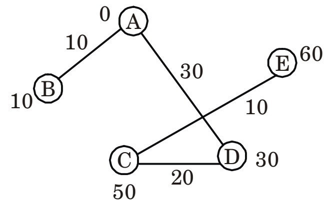
A	B	C	D	E
$\boxed{0}$	∞	∞	∞	∞
	$\boxed{10}$	∞	30	100
		60	$\boxed{30}$	100
		$\boxed{50}$		90
				60

Extract min(*E*) :



A	B	C	D	E
$\boxed{0}$	∞	∞	∞	∞
	$\boxed{10}$	∞	30	100
		60	$\boxed{30}$	100
		$\boxed{50}$		90
				$\boxed{60}$

Shortest path :



7. Attempt any **one** part of the following : **(7 × 1 = 7)**

a. Write short notes on :

i. Hashing technique

ii. Garbage collection

Ans.

i. Hashing technique :

1. Hashing technique is one of the complex searching techniques. In this technique, we consider a class of search techniques whose search time is dependent on the number of entries available in the table.
2. Here, we fix the position of the key (element) into the table or the file, which is determined by the hash function.
3. The function in which we use this key is known as the hashing function.
4. For example, if we want to search a number from the ten numbers of a file, then we must find the number throughout this range from the first number to the tenth number. When we use the key for fixing its position in a table, then the number can be searched very easily.

ii. Garbage collection :

1. When some memory space becomes reusable due to the deletion of a node from a list or due to deletion of entire list from a program then we want the space to be available for future use.
2. One method to do this is to immediately reinsert the space into the free-storage list. This is implemented in the linked list.
3. This method may be too time consuming for the operating system of a computer.
4. In another method, the operating system of a computer may periodically collect all the deleted space onto the free storage list. This type of technique is called garbage collection.
5. Garbage collection usually takes place in two steps. First the computer runs through all lists, tagging those cells which are currently in use and then the computer runs through the memory, collecting all untagged space onto the free storage list.
6. The garbage collection may take place when there is only some minimum amount of space or no space at all left in the free storage list or when the CPU is idle and has time to do the collection.

b. Explain the following : **(7 × 1 = 7)**

i. Heap sort

ii. Radix sort

Ans.

i. Heap sort :

1. Heap sort finds the largest element and puts it at the end of array, then the second largest item is found and this process is repeated for all other elements.
2. The general approach of heap sort is as follows :
   a. From the given array, build the initial max heap.
   b. Interchange the root (maximum) element with the last element.
   c. Use repetitive downward operation from root node to rebuild the heap of size one less than the starting.
   d. Repeat step (a) and (b) until there are no more elements.

ii. Radix sort :

1. Radix sort is a small method that many people uses when alphabetizing a large list of names (here Radix is 26, 26 letters of alphabet).
2. Specifically, the list of name is first sorted according to the first letter of each name, *i.e.*, the names are arranged in 26 classes.
3. Intuitively, one might want to sort numbers on their most significant digit.
4. But radix sort do counter-intuitively by sorting on the least significant digits first.
5. On the first pass entire numbers sort on the least significant digit and combine in an array.
6. Then on the second pass, the entire numbers are sorted again on the second least-significant digits and combine in an array and so on.

☺☺☺

B.Tech.

(SEM. III) ODD SEMESTER THEORY EXAMINATION, 2018-19

DATA STRUCTURES

Time : 3 Hours **Max. Marks : 70**

Note : Attempt **all** sections. Assume missing data, if any.

Section – A

1. Attempt **all** questions in brief. **(2 × 7 = 14)**

a. How the graph can be represented in memory ? Explain with suitable example.

b. Write the syntax to check whether a given circular queue is full or empty ?

c. Draw a binary tree for the expression : $A * B - (C + D) * (P/Q)$

d. Differentiate between overflow and underflow condition in a linked list.

e. What do you understand by stable and in-place sorting ?

f. Number of nodes in a complete tree is 100000. Find its depth.

g. What is recursion ? Give disadvantages of recursion.

Section-B

2. Attempt any **three** of the following : **(7 × 3 = 21)**

a. What do you understand by time and space tradeoff ? Define the various asymptotic notations. Derive the O-notation for linear search.

b. Consider the following infix expression and convert into reverse polish notation using stack. $A + (B * C - (D/E \wedge F) * H)$

c. Explain Huffman algorithm. Construct Huffman tree for *MAHARASHTRA* with its optimal code.

d. What is height balanced tree ? Why height balancing of tree is required ? Create an AVL tree for the following elements : *a, z, b, y, c, x, d, w, e, v, f.*

Ans. Refer Q. 5.25, Page 5–33A, Unit-5.

e. Write the Floyd Warshall algorithm to compute the all pair shortest path. Apply the algorithm on following graph :

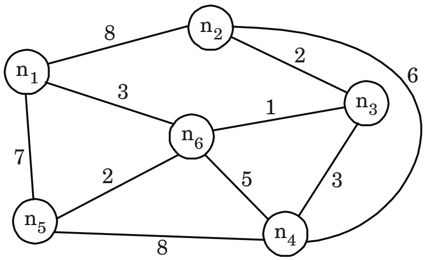


Fig. 1.

Section-C

3. Attempt any **one** part of the following : **(7 × 1 = 7)**

a. Write a program in C to delete a specific element in single linked list. Double linked list takes more space than single linked list for sorting one extra address. Under what condition, could a double linked list more beneficial than single linked list.

b. Suppose multidimensional arrays *P* and *Q* are declared as *P*(– 2: 2, 2: 22) and *Q*(1: 8, – 5: 5, – 10 : 5) stored in column major order

i. Find the length of each dimension of *P* and *Q*

ii. The number of elements in *P* and *Q*

iii. Assuming base address (*Q*) = 400, *W* = 4, find the effective indices E_1, E_2, E_3 and address of the element *Q*[3, 3, 3].

4. Attempt any **one** part of the following : **(7 × 1 = 7)**

a. Explain Tower of Hanoi problem and write a recursive algorithm to solve it.

b. Explain how a circular queue can be implemented using arrays. Write all functions for circular queue operations.

5. Attempt any **one** part of the following : **(7 × 1 = 7)**

a. Write the algorithm for deletion of an element in binary search tree.

b. **Construct a binary tree for the following :**
Inorder : *Q, B, K, C, F, A, G, P, E, D, H, R*
Preorder : *G, B, Q, A, C, K, F, P, D, E, R, H*
Find the postorder of the tree.

6. Attempt any **one** part of the following : **(7 × 1 = 7)**

a. **By considering vertex '1' as source vertex, find the shortest paths to all other vertices in the following graph using Dijkstra's algorithms. Show all the steps.**

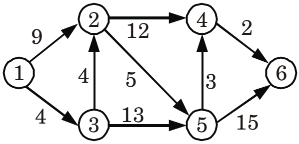


Fig. 2.

b. **Explain in detail about the graph traversal techniques with suitable examples.**

7. Attempt any **one** part of the following : **(7 × 1 = 7)**

a. **Write algorithm for quick sort. Trace your algorithm on the following data to sort the list: 2, 13, 4, 21, 7, 56, 51, 85, 59, 1, 9, 10. How the choice of pivot elements effects the efficiency of algorithm ?**

b. **Construct a B-tree of order 5 created by inserting the following elements 3, 14, 7, 1, 8, 5, 11, 17, 13, 6, 23, 12, 20, 26, 4, 16, 18, 24, 25, 19. Also delete elements 6, 23 and 3 from the constructed tree.**

☺☺☺

SOLUTION OF PAPER (2018-19)

Note : Attempt **all** sections. Assume missing data, if any.

Section – A

1. Attempt **all** questions in brief. **(2 × 7 = 14)**

a. How the graph can be represented in memory ? Explain with suitable example.

Ans. Graph can be represented in memory :

1. Matrix representation
2. Linked representation

For example : Consider the following directed graph :

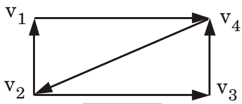


Fig. 1.

Matrix representation :

$$\begin{array}{c c} & \begin{array}{cccc} v_1 & v_2 & v_3 & v_4 \end{array} \\ \begin{array}{c} v_1 \\ v_2 \\ v_3 \\ v_4 \end{array} & \begin{bmatrix} 0 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 \end{bmatrix} \end{array}$$

Linked representation :

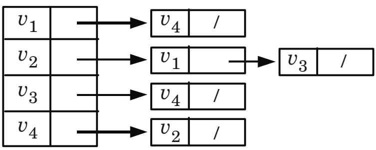


Fig. 2.

b. Write the syntax to check whether a given circular queue is full or empty ?

Ans. Syntax to check circular queue is full :

If ((front == MAX – 1) || (front == 0 && rear == MAX – 1))

Syntax to check circular queue is empty :

If (front == 0 && rear == – 1)

c. Draw a binary tree for the expression : $A * B - (C + D) * (P/Q)$

Ans.

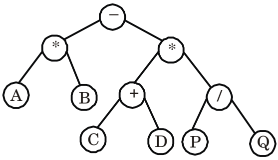


Fig. 3.

d. Differentiate between overflow and underflow condition in a linked list.

Ans.

S. No.	Overflow	Underflow
1.	Overflow condition occurs in linked list when data are inserted into a list but there is no available space.	Underflow condition occurs when we delete data from empty linked list.
2.	In linked list overflow occurs when AVAIL = NULL and there is an insertion operation.	In linked list underflow occurs when START = NULL and there is a deletion operation.

e. What do you understand by stable and in-place sorting ?

Ans. **Stable sorting :** Stable sorting is an algorithm where two objects with equal keys appear in the same order in sorted output as they appear in the input unsorted array.

In-place sorting : An in-place sorting is an algorithm that does not need an extra space and produces an output in the same memory that contains the data by transforming the input 'in-place'. However, a small constant extra space used for variables is allowed.

f. Number of nodes in a complete tree is 100000. Find its depth.

Ans. Number of nodes in a complete tree = 100000

We know that,
$$n = 2^{h+1} - 1$$
$$(n + 1) = 2^{h+1}$$
$$\log_2(n + 1) = h + 1$$
$$\log_2(n + 1) - 1 = h$$

Putting
$$n = 100000$$
$$h = \log_2(100000 + 1) - 1$$
$$h = 15 \text{ (approx)}$$

g. What is recursion ? Give disadvantages of recursion.

Ans. **Recursion :** Recursion is the process of expressing a function that calls itself to perform specific operation.

Disadvantages of recursion :

1. Recursive solution is always logical and it is very difficult to trace, debug and understand.
2. Recursion takes a lot of stack space, usually not considerable when the program is small and running on a PC.
3. Recursion uses more processor time.

Section-B

2. Attempt any **three** of the following : **(7 × 3 = 21)**

a. What do you understand by time and space tradeoff ? Define the various asymptotic notations. Derive the O-notation for linear search.

Ans. **Time-space trade-off :**

1. The time-space trade-off refers to a choice between algorithmic solutions of data processing problems that allows to decrease the running time of an algorithmic solution by increasing the space to store data and vice-versa.
2. Time-space trade-off is basically a situation where either space efficiency (memory utilization) can be achieved at the cost of time or time efficiency (performance efficiency) can be achieved at the cost of memory.

For Example : Suppose, in a file, if data stored is not compressed, it takes more space but access takes less time. Now if the data stored is compressed the access takes more time because it takes time to run decompression algorithm.

Various asymptotic notation :

1. **θ-Notation (Same order) :** This notation bounds a function to within constant factors.
2. **Oh-Notation (Upper bound) :** It is the measure of the longest amount of time it could possibly take for the algorithm to complete.
3. **Ω-Notation (Lower bound) :** This notation gives a lower bound for a function to within a constant factor.
4. **Little - Oh notation (o) :** It is used to denote an upper bound that is asymptotically tight because upper bound provided by O-notation is not tight.
5. **Little omega notation (ω) :** It is used to denote lower bound that is asymptotically tight.

Derivation :

Best case : In the best case, the desired element is present in the first position of the array, *i.e.*, only one comparison is made.

So, $T(n) = O(1)$.

Average case : Here we assume that ITEM does appear, and that is equally likely to occur at any position in the array. Accordingly the number of comparisons can be any of the number 1, 2, 3, ..., n and each number occurs with the probability $p = 1/n$. Then

$$T(n) = 1 \,.\, (1/n) + 2 \,.\, (1/n) + 3 \,.\, (1/n) \,......\, + n \,.\, (1/n)$$

$$= (1 + 2 + 3 + \ldots\ldots + n) \cdot (1/n)$$
$$= n \cdot (n + 1)/2 \cdot (1/n) = (n + 1)/2$$
$$= O((n + 1)/2) \simeq O(n)$$

Worst case : Worst case occurs when ITEM is the last element in the array or is not there at all. In this situation n comparison is made.

So, $T(n) = O(n + 1) \simeq O(n)$

b. Consider the following infix expression and convert into reverse polish notation using stack. *A* + (*B* * *C* – (*D*/*E* ^ *F*) * *H*)

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

Character	Stack	Postfix
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 ^/
H	(+ (– *	ABC*DEF ^/ H

Resultant reverse polish expression : *ABC * DEF ^ / H*

c. Explain Huffman algorithm. Construct Huffman tree for *MAHARASHTRA* with its optimal code.

Ans. **Huffman algorithm :**

1. Suppose, there are n weights $W_1, W_2, \ldots\ldots, W_n$.
2. Take two minimum weights among the n given weights. Suppose W_1 and W_2 are first two minimum weights then subtree will be :

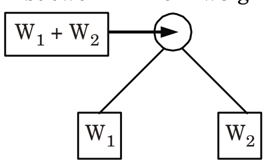
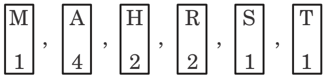


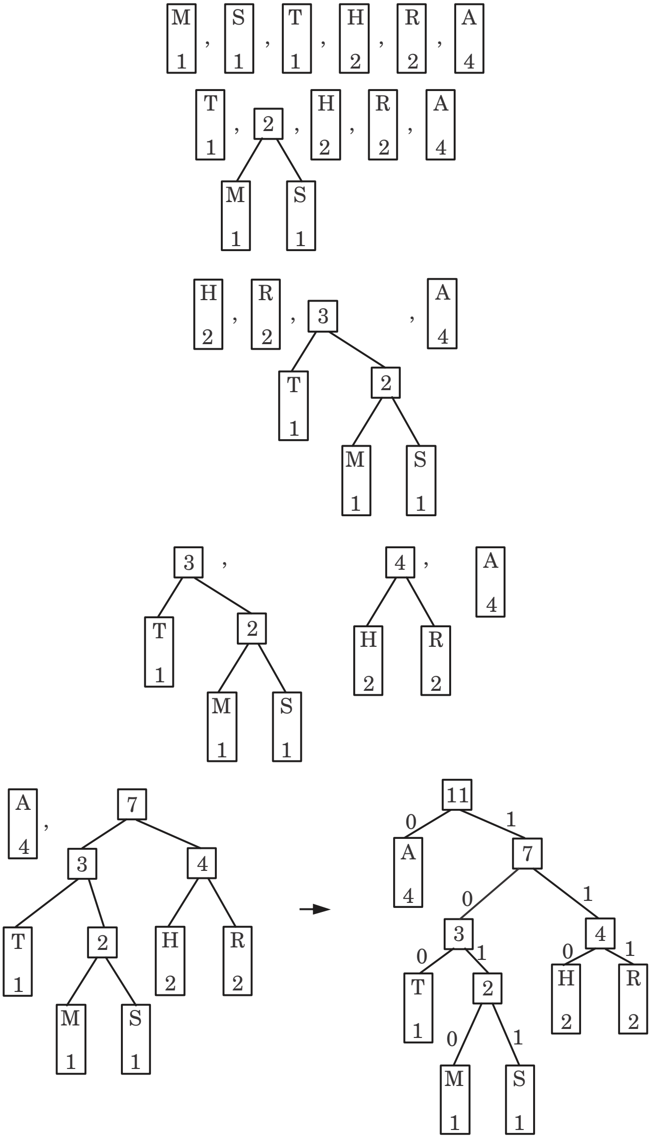
Fig. 4.

3. Now the remaining weights will be $W_1 + W_2, W_3, W_4, \ldots\ldots, W_n$.
4. Create all subtree at the last weight.

Numerical :



Arrange all the number in ascending order.



Character	Code
M	1010
A	0
H	110
R	111
S	1011
T	100

Optimal code for *MAHARASHTRA* is :
101001100111010111101001110

d. What is height balanced tree ? Why height balancing of tree is required ? Create an AVL tree for the following elements : *a, z, b, y, c, x, d, w, e, v, f*.

Ans. Height balanced tree :

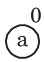
i. An AVL (or height balanced) tree is a balanced binary search tree.
ii. In an AVL tree, balance factor of every node is either –1, 0 or +1.
iii. Balance factor of a node is the difference between the heights of left and right subtrees of that node.

Balance factor = height of left subtree – height of right subtree

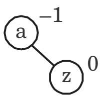
Height balancing of tree is required : Height balancing of tree is required to implement an AVL tree. Each node must contain a balance factor, which indicates its states of balance relative to its sub-tree.

Numerical :

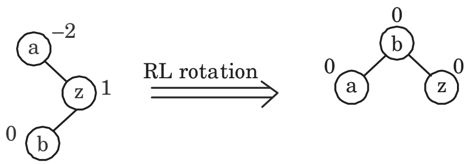
Insert *a* :



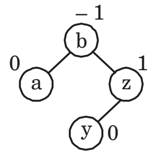
Insert *z* :



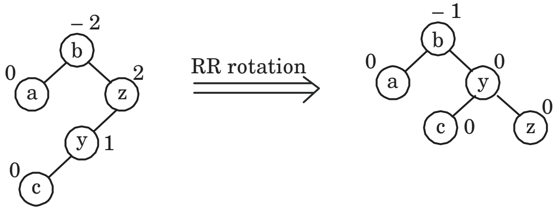
Insert *b* :



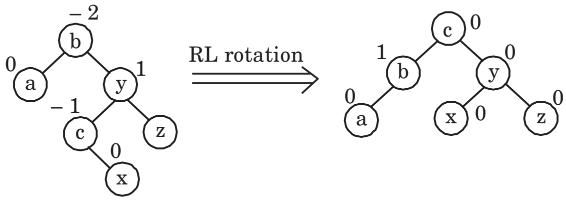
Insert *y* :



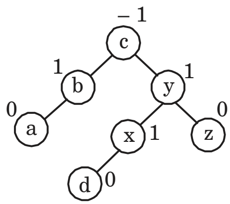
Insert *c* :



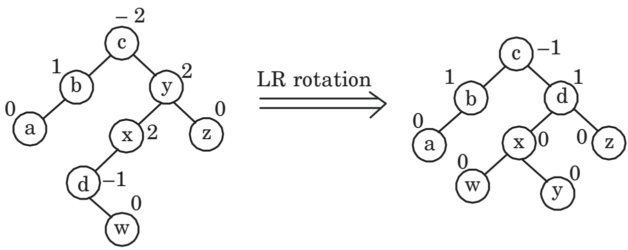
Insert *x* :



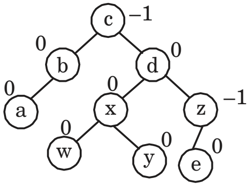
Insert *d* :



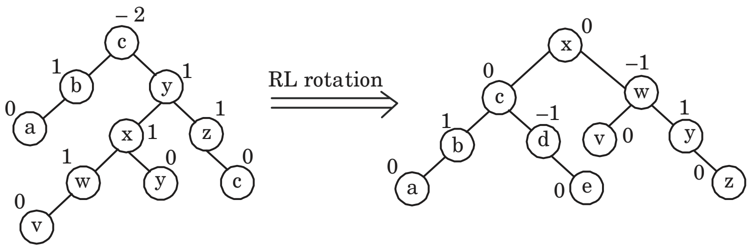
Insert *w* :



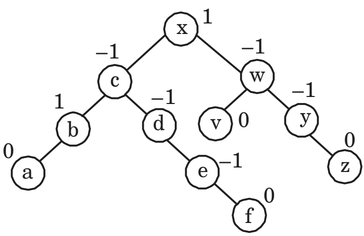
Insert *e* :



Insert *v* :



Insert *f* :



No, rebalancing required. So, this is final AVL search tree.

e. Write the Floyd Warshall algorithm to compute the all pair shortest path. Apply the algorithm on following graph :

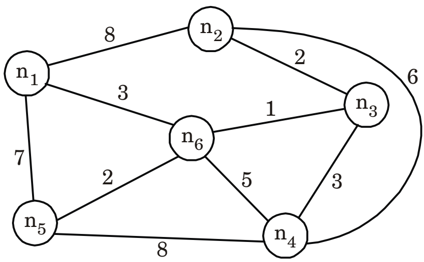


Fig. 5.

Ans. **Floyd's Warshall algorithm :**

Floyd Warshall (w) :

1. $n \leftarrow$ rows $[w]$
2. $D^{(0)} \leftarrow w$
3. for $k \leftarrow 1$ to n
4. do for $i \leftarrow 1$ to n
5. do for $j \leftarrow 1$ to n
6. do $d_{ij}^{(k)} \leftarrow \min(d_{ij}^{(k-1)}, d_{ik}^{(k-1)} + d_{kj}^{(k-1)})$
7. return $D^{(n)}$

Numerical : We cannot solve this using Floyd Warshall algorithm because the given graph is undirected.

Section-C

3. Attempt any **one** part of the following : **(7 × 1 = 7)**

a. Write a program in C to delete a specific element in single linked list. Double linked list takes more space than single linked list for sorting one extra address. Under what condition, could a double linked list more beneficial than single linked list.

Ans.

```
#include <stdlib.h>
// A linked list node
struct Node
{
int data;
struct Node *next;
};
/* Given a reference (pointer to pointer) to the head of a list
and an int, inserts a new node on the front of the list. */
void push(struct Node** head_ref, int new_data)
{
struct Node* new_node = (struct Node*) malloc(sizeof(struct
Node));
new_node->data = new_data;
new_node->next = (*head_ref);
(*head_ref) = new_node;
}
/* Given a reference (pointer to pointer) to the head of a list
and a position, deletes the node at the given position */
void deleteNode(struct Node **head_ref, int position)
{
// If linked list is empty
if (*head_ref == NULL)
return;
// Store head node
```

```
struct Node* temp = *head_ref;
// If head needs to be removed
if (position == 0)
{
*head_ref = temp->next; // Change head
free(temp); // free old head
return;
}
// Find previous node of the node to be deleted
for (int i = 0; temp != NULL && i < position – 1; i++)
temp = temp->next;
// If position is more than number of nodes
if (temp == NULL || temp->next == NULL)
return;
// Node temp->next is the node to be deleted
// Store pointer to the next of node to be deleted
struct Node *next = temp->next->next;
// Unlink the node from linked list
free(temp->next); // Free memory
temp->next = next; // Unlink the deleted node from list
}
// This function prints contents of linked list starting from
// the given node
void printList(struct Node *node)
{
while (node != NULL)
{
printf("%d ", node->data);
node = node->next;
}
}
/* Program to test above functions*/
int main()
{
/* Start with the empty list */
struct Node* head = NULL;
push(&head, 7);
push(&head, 1);
push(&head, 3);
push(&head, 2);
push(&head, 8);
puts("Created Linked List: ");
printList(head);
deleteNode(&head, 4);
puts("\nLinked List after Deletion at position 4: ");
printList(head);
```

return 0;
}

Double linked list is more beneficial than single linked list because :

1. A double linked list can be traversed in both forward and backward direction.
2. The delete operation in double linked list is more efficient if pointer to the node to be deleted is given.
3. In double linked list, we can quickly insert a new node before a given node.
4. In double linked list, we can get the previous node using previous pointer but in singly liked list we traverse the list to get the previous node.

b. Suppose multidimensional arrays *P* and *Q* are declared as *P*(– 2: 2, 2: 22) and *Q*(1: 8, – 5: 5, – 10 : 5) stored in column major order

i. Find the length of each dimension of *P* and *Q*

ii. The number of elements in *P* and *Q*

iii. Assuming base address (*Q*) = 400, *W* = 4, find the effective indices E_1, E_2, E_3 and address of the element *Q*[3, 3, 3].

Ans.

i. The length of a dimension is obtained by
Length = Upper Bound – Lower Bound + 1
Hence, the lengths of the dimension of *P* are,
$L_1 = 2 - (-2) + 1 = 5$; $L_2 = 22 - 2 + 1 = 21$
The lengths of the dimension of *Q* are,
$L_1 = 8 - 1 + 1 = 8$; $L_2 = 5 - (-5) + 1 = 11$; $L_3 = 5 - (-10) + 1 = 16$

ii. Number of elements in $P = 21 \times 5 = 105$ elements
Number of elements in $Q = 8 \times 11 \times 16 = 1408$ elements

iii. The effective index E_i is obtained from $E_i = k_i - \text{LB}$, where k_i is the given index and LB, is the Lower Bound. Hence,
$E_1 = 3 - 1 = 2$; $E_2 = 3 - (-5) = 8$; $E_3 = 3 - (-10) = 13$
The address depends on whether the programming language stores *Q* in row major order or column major order. Assuming *Q* is stored in column major order.
$E_3L_2 = 13 \times 11 = 143$
$E_3L_2 + E_2 = 143 + 8 = 151$
$(E_3L_2)L_1 = 151 * 8 = 1208$
$(E_2L_2+E_2)L_1 + E_1 = 1208 + 2 = 1210$
Therefore, LOC(Q[3,3,3]) = 400 + 4(1210) = 400 + 4840 = 5240

4. Attempt any **one** part of the following : **(7 × 1 = 7)**

a. Explain Tower of Hanoi problem and write a recursive algorithm to solve it.

Ans. **Tower of Hanoi problem :**

1. Suppose three pegs, labelled A, B and C is given, and suppose on peg A, there are finite number of n disks with decreasing size.
2. The object of the game is to move the disks from peg A to peg C using peg B as an auxiliary.
3. The rule of game is follows :
a. Only one disk may be moved at a time. Specifically only the top disk on any peg may be moved to any other peg.
b. At no time, can a larger disk be placed on a smaller disk.

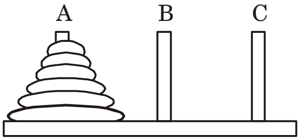


Fig. 6.

The solution to the Tower of Hanoi problem for $n = 3$.

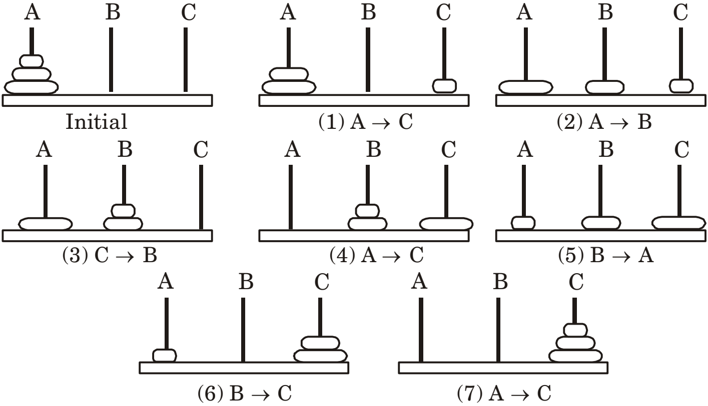


Fig. 7.

Total number of steps to solve Tower of Hanoi problem of n disk

$$= 2^n - 1 = 2^3 - 1 = 7$$

Algorithm :

TOWER (N, BEG, AUX, END)

This procedure gives a recursive solution to the Tower of Hanoi problem for N disks.

1. If N = 1, then :
   a. Write: BEG → END
   b. Return
   [End of If structure]
2. [Move N – 1 disk from peg BEG to peg AUX]

Call TOWER (N – 1, BEG, END, AUX)

3. Write: BEG → END
4. [Move N – 1 disk from peg AUX to peg END]
   Call TOWER (N – 1, AUX, BEG, END)
5. Return

b. Explain how a circular queue can be implemented using arrays. Write all functions for circular queue operations.

Ans. Implementation of circular queue using array :

```
#include<stdio.h>
#include<conio.h>
#include<process.h>
#define MAX 10
    typedef struct {
        int front, rear ;
        int elements [MAX];
        } queue;
    void createqueue (queue *aq) {
    aq -> front = aq -> rear = – 1
    }
    int isempty (queue *aq)
    {
        if(aq -> front = = – 1)
        return 1;
else
    return 0;
}
int isfull (queue *aq) {
    if(((aq -> front = = 0) && (aq -> rear = = MAX – 1))
                    ||(aq - > front == aq - > rear + 1))
        return 1;
    else
        return 0;
}
void insert (queue *aq, int value) {
    if(aq -> front = = – 1)
        aq -> front = aq -> rear = 0;
    else
        aq -> rear = (aq -> rear + 1) % MAX;
        aq -> element [aq -> rear] = value;
}
int delete (queue *aq) {
    int temp;
    temp = aq -> element [aq -> front];
    if(aq -> front = = aq ->rear)
    aq -> front = aq -> rear = – 1;
    else
```

```
        aq -> front = (aq -> front + 1) % MAX ;
        return temp;
        }
    void main( )
{
        int ch, elmt;
        queue q;
        create queue (&q);
        while (1) {
        printf("1. Insertion \n");
        printf("2. Deletion \n");
        printf("3. Exit \n");
        printf("Enter your choice");
        scanf("%d",&ch) ; .
        switch (ch)
                    {
                     case 1:
                     if(isfull (&q))
                     {
                     printf ("queue is full");
                     getch();
                     }
                     else
                     {
                     printf("Enter value");
                     scanf("%d", &elmt) ;
                     insert (&q, elmt) ;
                     }
                     break;
                     case 2: if (isempty (&q))
                     {
                     printf("queue empty");
                     getch();
                     }
                     else
                     {
                     printf("Value deleted is % d", delete (&q));
                     getch( );
                     }
                     break;
                     case 3:
                     exit(1);
                    }
    }   }
```

Function for circular queue operations :

Function to create circular queue :

```
void Queue :: enQueue(int value)
```

```
{
if((front == 0 && rear == size – 1) || (rear == (front – 1)%(size – 1)))
{
printf("\nQueue is Full");
return;
}
else if (front == –1) /* Insert First Element */
{
front = rear = 0;
arr[rear] = value;
}
else if (rear == size–1 && front != 0)
{
rear = 0;
arr[rear] = value;
}
else
{
rear++;
arr[rear] = value;
}
}
```

Function to delete element from circular queue :

```
int Queue :: deQueue()
{
if (front == –1)
{
printf("\nQueue is Empty");
return INT_MIN;
}
int data = arr[front];
arr[front] = – 1;
if (front == rear)
{
front = – 1;
rear = – 1;
}
else if (front == size – 1)
front = 0;
else
front++;
return data;
}
```

5. Attempt any **one** part of the following : **(7 × 1 = 7)**

a. Write the algorithm for deletion of an element in binary search tree.

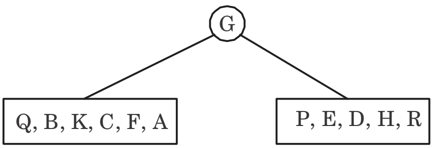
Ans. DEL(INFO, LEFT, RIGHT, ROOT, AVAIL, ITEM)
A binary search tree *T* is in memory, and an ITEM of information is given. This algorithm deletes ITEM from the tree.

1. [Find the locations of ITEM and its parent]
   Call FIND(INFO, LEFT, RIGHT, ROOT, ITEM, LOC, PAR)
2. [ITEM in tree ?]
   If LOC = NULL, then write ITEM not in tree, and Exit.
3. [Delete node containing ITEM]
   If RIGHT[LOC] ≠ NULL and LEFT[LOC] ≠ NULL, then :
   Call CASEB(INFO, LEFT, RIGHT, ROOT, LOC, PAR)
   Else :
   Call CASEA(INFO, LEFT, RIGHT, ROOT, LOC, PAR)
   [End of If structure]
4. [Return deleted node to the AVAIL list]
   Set LEFT[LOC] := AVAIL and AVAIL := LOC
5. Exit.

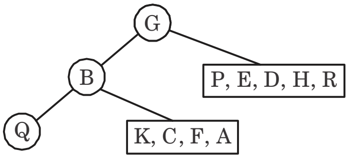
b. Construct a binary tree for the following :
Inorder : *Q, B, K, C, F, A, G, P, E, D, H, R*
Preorder : *G, B, Q, A, C, K, F, P, D, E, R, H*
Find the postorder of the tree.

Ans. **Step 1 :** In preorder traversal root is the first node. So, *G* is the root node of the binary tree. So,

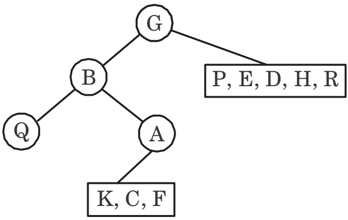
Step 2 : We can find the node of left sub-tree and right sub-tree with inorder sequence. So,



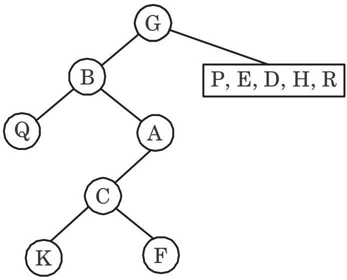
Step 3 : Now, the left child of the root node will be the first node in the preorder sequence after root node *G*. So,



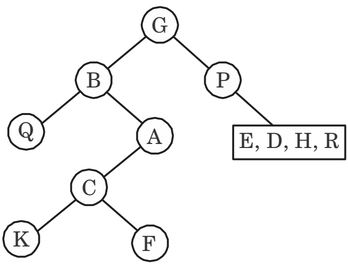
Step 4 : In inorder sequence, *Q* is on the left side of *B* and *A* is on the right side *B*. So,



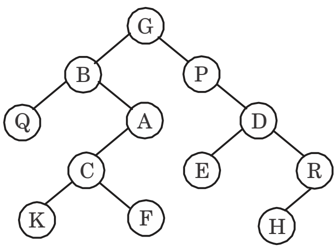
Step 5 : In inorder sequence, C is on the left side of A . Now according to inorder sequence, K is on the left side of C and F is on the right side of C.



Step 6 : Similarly, we can go further for right side of G.



So, the final tree is



Postorder of tree : $Q, K, F, A, B, E, H, R, D, P, G$

6. Attempt any **one** part of the following : **(7 × 1 = 7)**

a. By considering vertex '1' as source vertex, find the shortest paths to all other vertices in the following graph using Dijkstra's algorithms. Show all the steps.

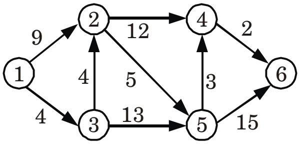
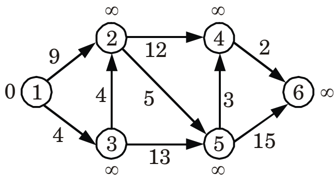


Fig. 8.

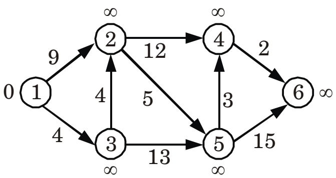
Ans. **Initialize :**



S = { }

Q : 1	2	3	4	5	6
0	∞	∞	∞	∞	∞

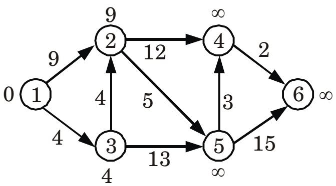
EXTRACT – MIN (1) :



S = { 1 }

Q : 1	2	3	4	5	6
$\boxed{0}$	∞	∞	∞	∞	∞

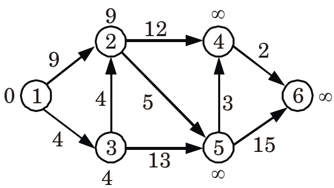
Relax all edges leaving 1 :



S = { 1 }

Q : 1	2	3	4	5	6
$\boxed{0}$	∞	∞	∞	∞	∞
	9	4	–	–	–

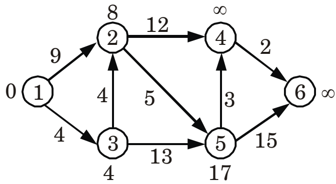
EXTRACT – MIN (3) :



S = { 1, 3 }

Q : 1	2	3	4	5	6
$\boxed{0}$	∞	∞	∞	∞	∞
	9	$\boxed{4}$	–	–	–

Relax all edges leaving 3 :

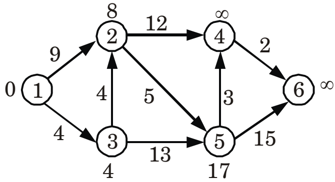


S = { 1, 3 }

Q :

1	2	3	4	5	6
$\boxed{0}$	∞	∞	∞	∞	∞
	9	$\boxed{4}$	–	–	–
	8		–	17	–

EXTRACT – MIN (2) :

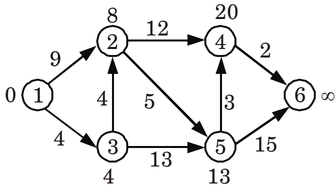


S = { 1, 3, 2 }

Q :

1	2	3	4	5	6
$\boxed{0}$	∞	∞	∞	∞	∞
	9	$\boxed{4}$	–	–	–
	$\boxed{8}$		–	17	–

Relax all edges leaving 2 :

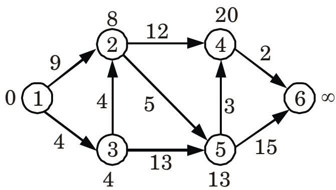


S = { 1, 3, 2 }

Q :

1	2	3	4	5	6
$\boxed{0}$	∞	∞	∞	∞	∞
	9	$\boxed{4}$	–	–	–
	$\boxed{8}$		–	17	–
			20	13	–

EXTRACT – MIN (5) :

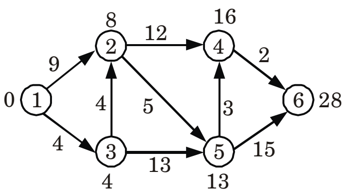


S = { 1, 3, 2, 5 }

Q :

1	2	3	4	5	6
$\boxed{0}$	∞	∞	∞	∞	∞
	9	$\boxed{4}$	–	–	–
	$\boxed{8}$		–	17	–
			20	$\boxed{13}$	–

Relax all edges leaving 5 :

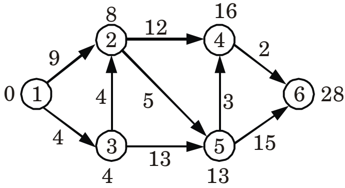


S = { 1, 3, 2, 5 }

Q :

1	2	3	4	5	6
$\boxed{0}$	∞	∞	∞	∞	∞
	9	$\boxed{4}$	–	–	–
	$\boxed{8}$		–	17	–
			20	$\boxed{13}$	–
			16		28

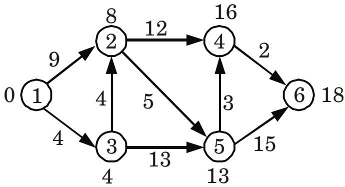
EXTRACT – MIN (4) :



S = { 1, 3, 2, 5, 4 }

Q : 1	2	3	4	5	6
$\boxed{0}$	∞	∞	∞	∞	∞
	9	$\boxed{4}$	–	–	–
	$\boxed{8}$		–	17	–
			20	$\boxed{13}$	–
			$\boxed{16}$		28

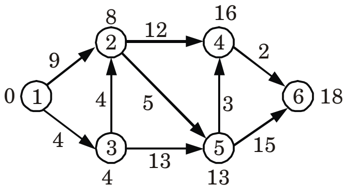
Relax all edges leaving 4 :



S = { 1, 3, 2, 5, 4 }

Q : 1	2	3	4	5	6
$\boxed{0}$	∞	∞	∞	∞	∞
	9	$\boxed{4}$	–	–	–
	$\boxed{8}$		–	17	–
			20	$\boxed{13}$	–
			$\boxed{16}$		28
					18

EXTRACT – MIN (6) :



S = { 1, 3, 2, 5, 4, 6 }

Q : 1	2	3	4	5	6
$\boxed{0}$	∞	∞	∞	∞	∞
	9	$\boxed{4}$	–	–	–
	$\boxed{8}$		–	17	–
			20	$\boxed{13}$	–
			$\boxed{16}$		28
					$\boxed{18}$

b. **Explain in detail about the graph traversal techniques with suitable examples.**

Ans. Following are the two traversal techniques :

1. **Depth First Search (DFS) :**

The general idea behind a depth first search beginning at a starting node *A* is as follows :

a. First, we examine the starting node *A*.

b. Then, we examine each node *N* along a path *P* which begins at *A*; that is, we process neighbour of *A*, then a neighbour of neighbour of *A*, and so on.

c. This algorithm uses a stack instead of queue.

Algorithm :

i. Initialize all nodes to ready state (STATUS = 1).

ii. Push the starting node *A* onto stack and change its status to the waiting state (STATUS = 2).

iii. Repeat steps (iv) and (v) until queue is empty.
iv. Pop the top node *N* of stack, process *N* and change its status to the processed state (STATUS = 3).
v. Push onto stack all the neighbours of *N* that are still in the ready state (STATUS = 1) and change their status to the waiting state (STATUS = 2).
[End of loop]
vi. End.
For example : Consider the following graph

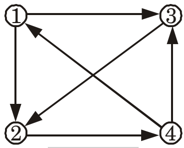
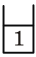


Fig. 9.

Adjacency list of the given graph :

$1 \rightarrow 2, 3$
$2 \rightarrow 4$
$3 \rightarrow 2$
$4 \rightarrow 3, 1$

1. Initially set STATUS = 1 for all vertex
2. Push 1 onto stack and set their STATUS = 2

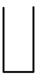


3. Pop 1 from stack, change its STATUS = 1 and
Push 2, 3 onto stack and change their STATUS = 2; DFS = 1

4. Pop 3 from stack, Push 2, but it is already in the stack; DFS = 1, 3

5. Pop 2 from stack, Push 4; DFS = 1, 3, 2

6. Pop 4 from stack; DFS = 1, 3, 2, 4



Now, the stack is empty, so the depth first traversal of a given graph is 1, 3, 2, 4.

2. Breadth First Search (BFS) :
The general idea behind a breadth first search beginning at a starting node *A* is as follows :

a. First we examine the starting node A.
b. Then, we examine all the neighbours of A, and so on.
c. We need to keep track of the neighbours of a node, and that no node is processed more than once.
d. This is accomplished by using a queue to hold nodes that are waiting to be processed, and by using a field STATUS which tells us the current status of any node.

Algorithm : This algorithm executes a breadth first search on a graph G beginning at a starting node A.

i. Initialize all nodes to ready state (STATUS=1).
ii. Put the starting node A in queue and change its status to the waiting state (STATUS = 2).
iii. Repeat steps (iv) and (v) until queue is empty.
iv. Remove the front node N of queue. Process N and change the status of N to the processed state (STATUS = 3).
v. Add to the rear of queue all the neighbours of N that are in the ready state (STATUS=1) and change their status to the waiting state
(STATUS = 2).
[End of loop]
vi. End.

For example : Consider the same graph in Fig. 11.
To find the shortest path from node 1 to node 4.
Adjacency list of the graph is :

1 : 2, 3
2 : 4
3 : 2
4 : 1, 3

a. Initially set STATUS=1 for all vertex.
b. Now add '1' to Queue and set STATUS = 2
Queue : 1
c. Remove 1 from Queue and set STATUS = 3
and add 2, 3 in Queue and change their STATUS = 2
BFS = 1 Queue : 2, 3
d. Remove 2, add 4 in Queue
BFS = 1, 2 Queue = 3, 4
e. Remove 3, add 2, but 2 is already visited. So no vertex will be added in this step
BFS = 1, 2, 3, Queue = 4
f. Remove 4, BFS = 1, 2, 3, 4
Now, the Queue is empty, so breadth first search of a given graph is 1, 2, 3, 4.

7. Attempt any **one** part of the following : **(7 × 1 = 7)**

a. Write algorithm for quick sort. Trace your algorithm on the following data to sort the list: 2, 13, 4, 21, 7, 56, 51, 85, 59,

1, 9, 10. How the choice of pivot elements effects the efficiency of algorithm ?

Ans. **Quick sort algorithm :**

QUICK-SORT (A, p, r) :

1. If $p < r$ then
2. $q \leftarrow$ PARTITION (A, p, r)
3. QUICK-SORT $(A, p, q - 1)$
4. QUICK-SORT $(A, q + 1, r)$

PARTITION (A, p, r) :

1. $x \leftarrow A[r]$
2. $i \leftarrow p - 1$
3. for $j \rightarrow p$ to $r - 1$
4. do if $A[j] \leq x$
5. then $i \rightarrow i + 1$
6. exchange $A[i] \leftrightarrow A[j]$
7. exchange $A[i + 1] \leftrightarrow A[r]$
8. return $i + 1$

Numerical :

1	2	3	4	5	6	7	8	9	10	11	12
2	13	4	21	7	56	51	85	59	1	9	10

Here $p = 1$, $r = 12$

$x = A[12] = 10$

$i = p - 1 = 0$

$j = 1 + 0 = 1$

Now, $j = 1$ and $i = 0$

$A[1] = 2 \leq 10$ (True)

then $i = 0 + 1 = 1$ and $A[1] \leftrightarrow A[1]$

Now, $j = 2$ and $i = 1$

$A[2] = 13$ and $13 \nleq 10$ (False)

So, $j = 3 \quad i = 1$

$A[3] = 4$ and $4 \leq 10$ (True)

then, $i = 1 + 1 = 2$ and $A[2] \leftrightarrow A[3]$

i.e.,

1	2	3	4	5	6	7	8	9	10	11	12
2	4	13	21	7	56	51	85	59	1	9	10

Now, $j = 4$ and $i = 2$

$A[4] = 21$ and $21 \nleq 10$ (False)

$j = 5$ and $i = 2$

$A[5] = 7 \leq 10$ (True)

then, $i = 2 + 1 = 3$ and $A[3] \leftrightarrow A[5]$

i.e.,

1	2	3	4	5	6	7	8	9	10	11	12
2	4	7	21	13	56	51	85	59	1	9	10

Now, $j = 6$ and $i = 3$

$A[6] = 56$ and $56 \nleq 10$

So, $j = 7$ and $i = 3$

$A[7] = 51$ and $51 \nleq 10$

$$j = 8 \text{ and } i = 3$$
$$A[8] = 85 \text{ and } 85 \nleq 10$$
$$j = 9 \text{ and } i = 3$$
$$A[9] = 59 \text{ and } 59 \nleq 10$$
$$j = 10 \text{ and } i = 3$$
$$A[10] = 1 \leq 10 \text{ (True)}$$

then, $i = 3 + 1 = 4$ and $A[4] \leftrightarrow A[10]$

i.e.,

1	2	3	4	5	6	7	8	9	10	11	12
2	4	7	1	13	56	51	85	59	21	9	10

$$j = 11 \text{ and } i = 4$$
$$A[11] = 9 \leq 10 \text{ (True)}$$
$$i = 4 + 1 = 5 \text{ and } A[5] \leftrightarrow A[11]$$

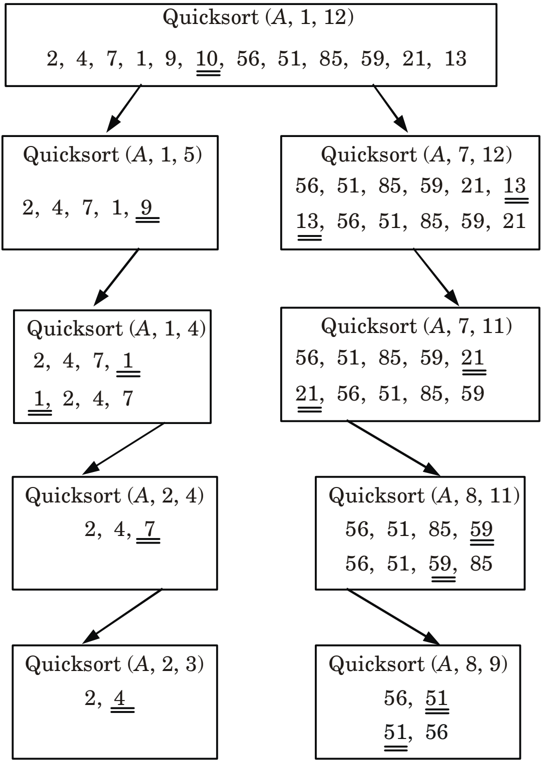
i.e.,

1	2	3	4	5	6	7	8	9	10	11	12
2	4	7	1	9	56	51	85	59	21	13	10

$A[6] \leftrightarrow A[12]$

Partitioning complete, return value of *q* :

1	2	3	4	5	6	7	8	9	10	11	12
2	4	7	1	9	10	56	51	85	59	21	13



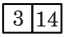
Choice of pivot element affects the efficiency of algorithm : If we choose the last or first element of an array as pivot element then it results in worst case scenario with $O(n^2)$ time complexity. If we choose the median as pivot element then it divides the array into two halves every time and results in best or average case scenario with time complexity $O(n \log n)$. Thus, the efficiency of quick sort algorithm depends on the choice of pivot element.

b. Construct a B-tree of order 5 created by inserting the following elements 3, 14, 7, 1, 8, 5, 11, 17, 13, 6, 23, 12, 20, 26, 4, 16, 18, 24, 25, 19. Also delete elements 6, 23 and 3 from the constructed tree.

Ans.

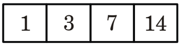
Insert 3 :

Insert 14 :

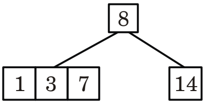


Insert 7 :

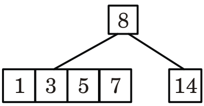
Insert 1 :



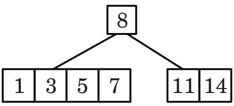
Insert 8 :



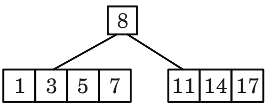
Insert 5 :



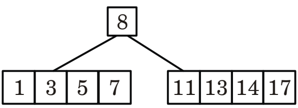
Insert 11 :



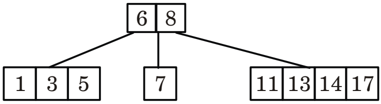
Insert 17 :



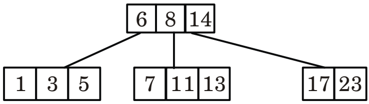
Insert 13 :



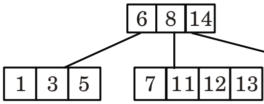
Insert 6 :



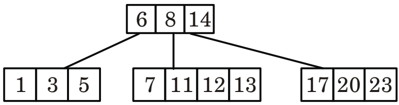
Insert 23 :



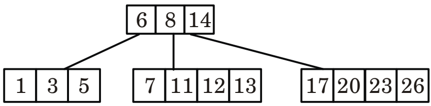
Insert 12 :



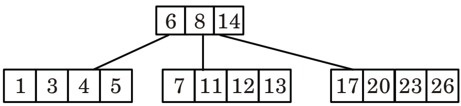
Insert 20 :



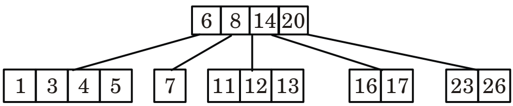
Insert 26 :



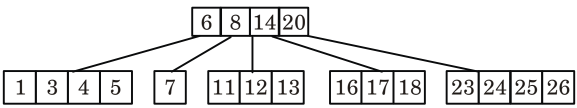
Insert 4 :



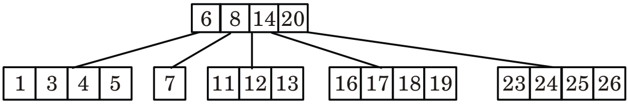
Insert 16 :



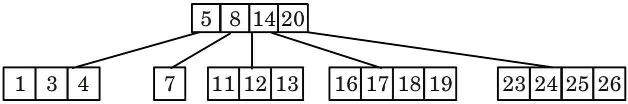
Insert 18, 24, 25 :



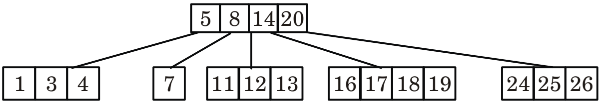
Insert 19 :



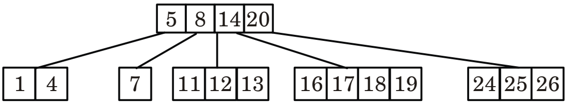
Delete 6 :



Delete 23 :



Delete 3 :



B.Tech.
(SEM. III) ODD SEMESTER THEORY EXAMINATION, 2019-20
DATA STRUCTURES

Time : 3 Hours **Max. Marks : 100**

Note : **1.** Attempt **all** Section.

Section-A

1. Answer **all** questions in brief. **(2 × 10 = 20)**

a. How can you represent a sparse matrix in memory ?

b. List the various operations on linked list.

c. Give some applications of stack.

d. Explain tail recursion.

e. Define priority queue. Given one application of priority queue.

f. How does bubble sort work ? Explain.

g. What is minimum cost spanning tree ? Give its applications.

h. Compare adjacency matrix and adjacency list representations of graph.

i. Define extended binary tree, full binary tree, strictly binary tree and complete binary tree.

j. Explain threaded binary tree.

Section-B

2. Answer any **three** of the following : **(3 × 10 = 30)**

a. What are the merits and demerits of array ? Given two arrays of integers in ascending order, develop an algorithm to merge these arrays to form a third array sorted in ascending order.

b. Write algorithm for push and pop operations in stack. Transform the following expression into its equivalent postfix expression using stack :
$A + (B * C - (D/E \uparrow F) * G) * H$

c. How binary search is different from linear search ? Apply binary search to find item 40 in the sorted array: 11, 22, 30, 33, 40, 44, 55, 60, 66, 77, 80, 88, 99. Also discuss the complexity of binary search.

d. Find the minimum spanning tree in the following graph using Kruskal's algorithm :

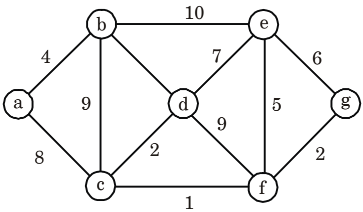


Fig. 1.

e. What is the difference between a binary search tree (BST) and heap ? For a given sequence of numbers, construct a heap and a BST.
34, 23, 67, 45, 12, 54, 87, 43, 98, 75, 84, 93, 31

Section-C

3. Answer any **one** part of the following : **(1 × 10 = 10)**

a. What is doubly linked list ? What are its applications ? Explain how an element can be deleted from doubly linked list using C program.

b. Define the following terms in brief :

i. Time complexity **ii. Asymptotic notation**

iii. Space complexity **iv. Big O notation**

4. Answer any **one** part of the following : **(1 × 10 = 10)**

a. i. Differentiate between iteration and recursion.

ii. Write the recursive solution for Tower of Hanoi problem.

b. Discuss array and linked representation of queue data structure. What is dequeue ?

5. Answer any **one** part of the following : **(10 × 1 = 10)**

a. Why is quick sort named as quick ? Show the steps of quick sort on the following set of elements : 25, 57, 48, 37, 12, 92, 86, 33 Assume the first element of the list to be the pivot element.

b. What is hashing ? Give the characteristics of hash function. Explain collision resolution technique in hashing.

6. Answer any **one** part of the following : **(1 × 10 = 10)**

a. Explain Warshall's algorithm with the help of an example.

b. Describe the Dijkstra algorithm to find the shortest path. Find the shortest path in the following graph with vertex 'S' as source vertex.

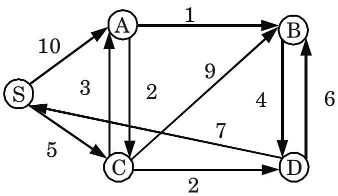


Fig. 2.

7. Answer any **one** part of the following : **(7 × 1 = 7)**

a. Can you find a unique tree when any two traversals are given ? Using the following traversals construct the corresponding binary tree :
INORDER : ***HKDBILEAFCMJG***
PREORDER : ***ABDHKEILCFGJM***
Also find the post order traversal of obtained tree.

b. What is a B-Tree ? Generate a B-Tree of order 4 with the alphabets (letters) arrive in the sequence as follows :
a g f b k d h i n j e s i r x c l n t u p.

☺☺☺

SOLUTION OF PAPER (2019-20)

Note : 1. Attempt **all** Section.

Section-A

1. Answer **all** questions in brief. **(2 × 10 = 20)**

a. How can you represent a sparse matrix in memory ?

Ans. There are two ways of representing sparse matrix in memory :

1. Array representation
2. Linked representation

b. List the various operations on linked list.

Ans. Various operations on linked list are :

1. Insertion at beginning
2. Insertion at end
3. Deletion at beginning
4. Deletion at end
5. Deletion of an element at specified location
6. Insertion of an element at specified location

c. Give some applications of stack.

Ans. Applications of stack are :

i. Infix to postfix conversion.
ii. Implementing function calls.
iii. Page-visited history in a web browser.
iv. Undo sequence in a text editor.

d. Explain tail recursion.

Ans.

Tail recursion (or tail-end recursion) is a special case of recursion in which the last operation of the function, the tail call is a recursive call. Such recursions can be easily transformed to iterations.

e. Define priority queue. Given one application of priority queue.

Ans. A priority queue is a data structure in which each element has been assigned a value called the priority of the element and an element can be inserted or deleted not only at the ends but at any position on the queue.

Applications of priority queue : In network communication, to manage limited bandwidth for transmission, the priority queue is used.

f. How does bubble sort work ? Explain.

Ans. Bubble sort procedure is based on following idea :

a. Suppose if the array contains n elements, then $(n - 1)$ iterations are required to sort this array.

b. The set of items in the array are scanned again and again and if any two adjacent items are found to be out of order, they are reversed.

c. At the end of the first iteration, the lowest value is placed in the first position.

d. At the end of the second iteration, the next lowest value is placed in the second position and so on.

g. What is minimum cost spanning tree ? Give its applications.

Ans. Minimum cost spanning tree : In a weighted graph, a minimum spanning tree is a spanning tree that has minimum weight than all other spanning trees of the same graph.

Application of minimum cost spanning tree :

1. Used for network designs.
2. Used to find approximate solutions for complex mathematical problems.
3. Cluster analysis.

h. Compare adjacency matrix and adjacency list representations of graph.

Ans.

S. No.	Adjacency matrix	Adjacency list
1.	An adjacency matrix is a square matrix used to represent a finite graph.	Adjacency list is a collection of unordered lists used to represent a finite graph.
2.	The elements of the matrix indicate whether pairs of vertices are adjacent or not in the graph.	Each list describes the set of adjacent vertices in the graph.
3.	Space complexity in the worst case is $O(\lvert V\rvert^2)$.	Space complexity in the worst case is $O(\lvert V\rvert+\lvert E\rvert)$.

i. Define extended binary tree, full binary tree, strictly binary tree and complete binary tree.

Ans. Extended binary tree : A binary tree T is said to be 2-tree or extended binary tree if each node has either 0 or 2 children.

Full binary tree : A full binary tree is formed when each missing child in the binary tree is replaced with a node having no children.

Strictly binary tree : If every non-leaf node in a binary tree has non-empty left and right subtree, the tree is termed as strictly binary tree.

Complete binary tree : A tree is called a complete binary tree if tree satisfies following conditions :

a. Each node has exactly two children except leaf node.
b. All leaf nodes are at same level.
c. If a binary tree contains m nodes at level l, it contains atmost $2m$ nodes at level $l + 1$.

j. Explain threaded binary tree.

Ans.

1. To make traversal of nodes more efficient we can utilize space occupied by the NULL pointers in the leaf nodes and internal nodes having only one child node.
2. These pointers can be modifies to point to their corresponding in-order successor, in-order predecessor or both.
3. These modified pointers are known as threads and binary trees having such type of pointers are known as threaded binary tree.

Section-B

2. Answer any **three** of the following : **(3 × 10 = 30)**

a. What are the merits and demerits of array ? Given two arrays of integers in ascending order, develop an algorithm to merge these arrays to form a third array sorted in ascending order.

Ans. **Merits of array :**

1. Array is a collection of elements of similar data type.
2. Hence, multiple applications that require multiple data of same data type are represented by a single name.

Demerits of array :

1. Linear arrays are static structures, *i.e.*, memory used by them cannot be reduced or extended.
2. Previous knowledge of number of elements in the array is necessary.

Algorithm : Algorithm receives as input indexes i, m, and j and an array a, where $a[i]$, ... , $a[m]$ and $a[m + 1]$, ... , $a[j]$ are two sorted in asecnding order. These two sorted arrays are merged into a single ascending array.

```
merge (a, i, m, j) {
p = i                // index in a[i] ... a[m]
q = m + 1            // index in a [m + 1] ... a [j]
r = i                // index of array C
while (p ≤ m L q ≤ j) {
if (a [p] ≤ a [q]) {
            c [r] = a [p] ;
              p = p + 1 ;
```

```
else {
        c[r] = a[q] ;
           q = q + 1 ;
}
           r = r + 1 ;
}
while (p ≤ m) {
        c [r] = a [p]
           p = p + 1 ;
           r = r + 1 ;
}
while (q ≤ j)
        c[r] = a [q]
           q = q + 1 ;
           r = r + 1 ;
}
for (r = i to j)
        a [r] = c [r]
```

b. Write algorithm for push and pop operations in stack. Transform the following expression into its equivalent postfix expression using stack :

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

Ans. Algorithm for push and pop operations :

Algorithm for PUSH operation :

PUSH (STACK, TOP, MAX, DATA)

1. If TOP = MAX – 1 then write "STACK OVERFLOW and STOP"
2. READ DATA
3. TOP ← TOP + 1
4. STACK [TOP] ← DATA
5. STOP

Algorithm for POP operation :

POP (STACK, TOP, ITEM)

1. If TOP < 0 then write "STACK UNDERFLOW and STOP"
2. STACK [TOP] ← NULL
3. TOP ← TOP – 1
4. STOP

Numerical : $A + (B*C - (D/E \uparrow F)*G) * H$

Character	Stack	Postfix
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

Resultant postfix expression : ABC*DEF↑/G* – H* +

c. How binary search is different from linear search ? Apply binary search to find item 40 in the sorted array: 11, 22, 30, 33, 40, 44, 55, 60, 66, 77, 80, 88, 99. Also discuss the complexity of binary search.

Ans. Difference :

S. No.	Binary search	Sequential (linear) search
1.	Elementary condition *i.e.*, array should be sorted.	No elementary condition *i.e.*, array can be sorted or unsorted.
2.	It takes less time to search an element.	It takes long time to search an element.
3.	Complexity is O(log2 n).	Complexity is O(n).
4.	It is based on divide and conquer method.	It searches data linearly.

Numerical :

Given sorted array :

	0	1	2	3	4	5	6	7	8	9	10	11	12
A	11	22	30	33	40	44	55	60	66	70	80	88	99

To search element 40

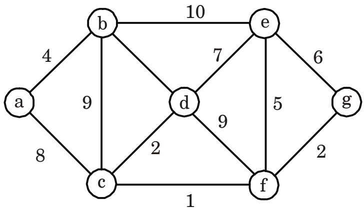
beg = 0, end = 12
mid = (0 + 12)/2 = 6
a[mid] = a[6] = 55 $\neq$ 40 (False)
40 < a[6]
end = 6 – 1 = 5
Now, beg = 0 end = 5

mid = (0 + 5)/2= $\lfloor 2.5 \rfloor$ = 2
a[mid] = a[2] = 30 $\neq$ 40 (False)
40 > a[2]
beg = 2 + 1 = 3
Now, beg = 3, end = 5
mid = (3 + 5)/2 = 4
a[mid] = a[4] = = 40 (True)
loc = 4
So, element 40 is present at location 4.

Complexity of binary search :

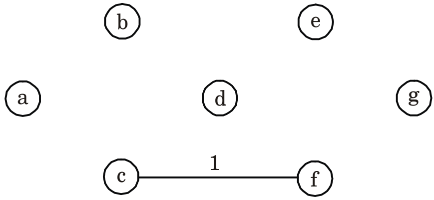
The complexity of binary search is $O(\log_2 n)$.

d. Find the minimum spanning tree in the following graph using Kruskal's algorithm :

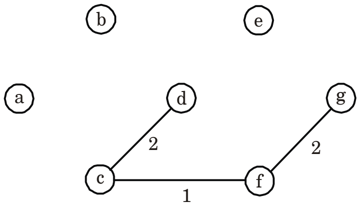


Ans.

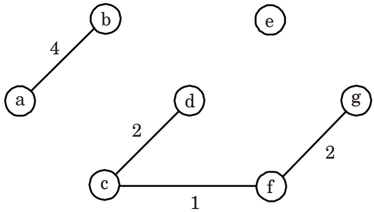
1. We will choose edge = *cf* as it has minimum weight.



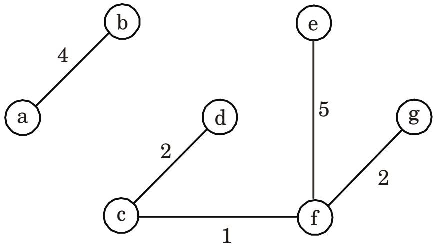
2. Now choose edge = *cd* and *fg* as it has minimum weight.



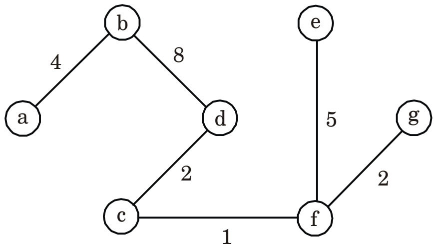
3. Now choose edge = *ab*



4. Now choose edge = *eg*



5. Now choose edge = *bd* and discard *be*, *eg*, *de*, *df*, *bc* and *ac* because they form cycle and we get the final minimal spanning tree as



e. What is the difference between a binary search tree (BST) and heap ? For a given sequence of numbers, construct a heap and a BST.
34, 23, 67, 45, 12, 54, 87, 43, 98, 75, 84, 93, 31

Ans. **Difference :**

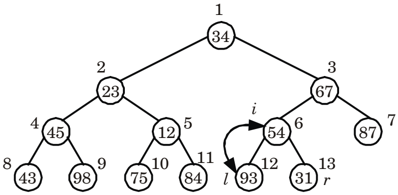
S. No.	Binary search tree (BST)	Heap
1.	In binary search tree, for every node except the leaf node, the left child has a less key value and right child has a greater key value.	In heap, for every node other than the root, the key value of the parent node is greater or smaller or equal to the key value of the child node.
2.	It guarantees the order (from left to right).	It guarantees that the element at higher level is smaller or greater than element at lower level.
3.	Time complexity to find min/ max element is O(log n).	Time complexity to find min/max is O(1).

Numerical :

Construction of heap :

	1	2	3	4	5	6	7	8	9	10	11	12	13
A =	34	23	67	45	12	54	87	43	98	75	84	93	31

Originally,



For $i = 6$

MAX-HEAPIFY (A, 6)

$l = 12 \quad r = 13$

12 < 13 and A[12] = 93 A[6] = 34

A[12] > A[6]

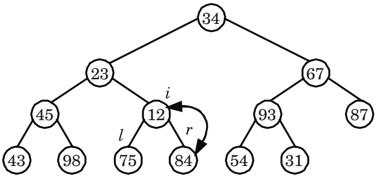
largest ← 12

13 = 13   A[13] = 31 A[12] = 93

A[13] $\not>$ A[12]

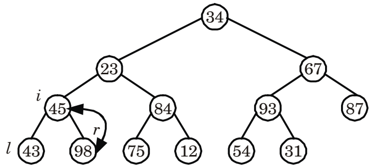
Exchange A[i] ↔ A[l]

A[6] ↔ A[12]

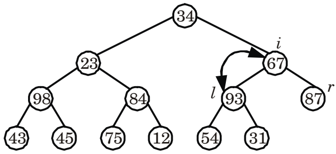


For i = 5

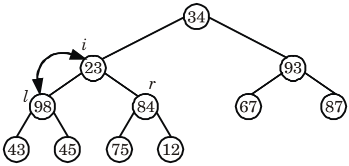
MAX-HEAPIFY (A, 5)
$l = 10 \quad r = 11$
$10 < 13$ and $A[10] > A[5]$
largest $\leftarrow 10$
$11 < 13$ and $A[11] > A[10]$
largest $\leftarrow 11$
Exchange $A[5] \leftrightarrow A[11]$



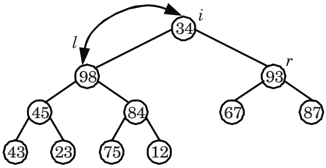
After MAX-HEAPIFY (A, 4)



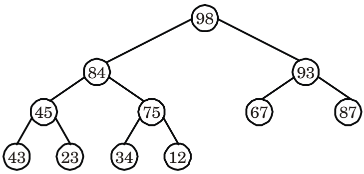
After MAX-HEAPIFY (A, 3)



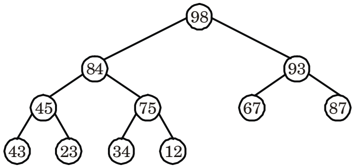
After MAX-HEAPIFY (A, 2)



After MAX-HEAPIFY (A, 1)



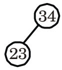
So, final tree after BUILD-MAX-HEAP is



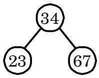
Construction of BST :

Insert 34 :

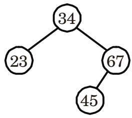
Insert 23 :



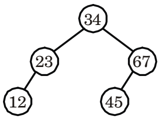
Insert 67 :



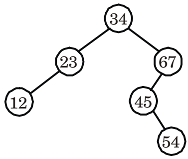
Insert 45 :



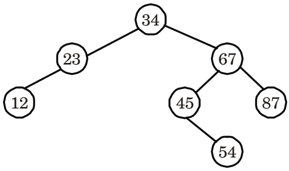
Insert 12 :



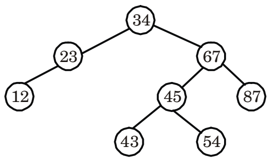
Insert 54 :



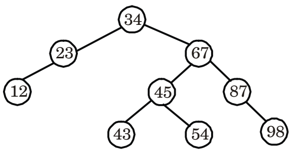
Insert 87 :



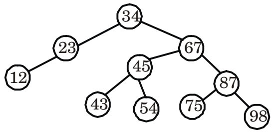
Insert 43 :



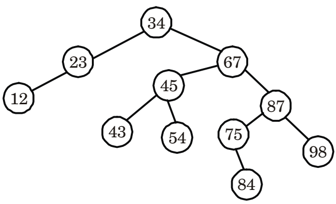
Insert 98 :



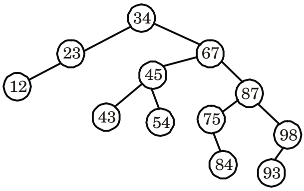
Insert 75 :



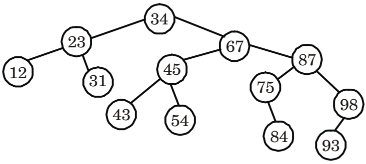
Insert 84 :



Insert 93 :



Insert 31 :



Section-C

3. Answer any **one** part of the following : **(1 × 10 = 10)**

a. What is doubly linked list ? What are its applications ? Explain how an element can be deleted from doubly linked list using C program.

Ans. Doubly linked list :

1. The doubly or two-way linked list uses double set of pointers, one pointing to the next node and the other pointing to the preceding node.
2. In doubly linked list, all nodes are linked together by multiple links which help in accessing both the successor and predecessor node for any arbitrary node within the list.

Applications of doubly linked list are :

1. Doubly linked list can be used in navigation systems where both front and back navigation is required.
2. It is used by browsers to implement backward and forward navigation of visited web pages.
3. It is used by various applications to implement undo and redo functionality.
4. It can be used to represent deck of cards in games.
5. It is used to represent various states of a game.

Deletion from doubly linked list using C program :

```
#include<stdio.h>
#include<conio.h>
typedef struct n{
int data;
struct n *prev;
struct n *next;
}node;
```

```
node *head = NULL, *tail = NULL;
```

Function to delete element :

```
void delete_beg(node *h, node *t) {
if(head == (node*)NULL) {
printf("\nList is empty.");
getch( );
return;
}
if(head == tail) {
free(h);
head = tail = (node *)NULL;
return;
}
if(h->next == t) {
tail->prev = NULL;
head = tail;
}
else {
head = head->next;
head->prev = NULL;
}
free(h);
}
```

b. Define the following terms in brief :

i. Time complexity **ii. Asymptotic notation**

iii. Space complexity **iv. Big O notation**

Ans.

i. Time complexity

1. The complexity of an algorithm M is the function $f(n)$ which gives the running time and/or storage space requirement of the algorithm in terms of the size n of the input data.
2. The storage space required by an algorithm is simply a multiple of the data size n.
3. Following are various cases in complexity theory :

a. Worst case : The maximum value of $f(n)$ for any possible input.

b. Average case : The expected value of $f(n)$ for any possible input.

c. Best case : The minimum possible value of $f(n)$ for any possible input.

ii. Asymptotic notation :

1. Asymptotic notation is a shorthand way to describe running times for an algorithm.
2. It is a line that stays within bounds.
3. These are also referred to as 'best case' and 'worst case' scenarios respectively.

iii. Space complexity :

1. The space complexity of an algorithm is the amount of memory it needs to run to completion.
2. It is expressed using only Big Oh notation.
3. Algorithm/program should have the less space complexity.
4. Lesser space used by algorithm/program, the faster it executes.

iv. Big 'Oh' notation :

1. Big-Oh is formal method of expressing the upper bound of an algorithm's running time.
2. It is the measure of the longest amount of time it could possibly take for the algorithm to complete.
3. More formally, for non-negative functions, $f(n)$ and $g(n)$, if there exists an integer n_0 and a constant $c > 0$ such that for all integers $n > n_0$.

$$f(n) \leq cg(n)$$

4. Then, $f(n)$ is Big-Oh of $g(n)$. This is denoted as : $f(n) \in O(g(n))$ *i.e.*, the set of functions which, as n gets large, grow faster than a constant time $f(n)$.

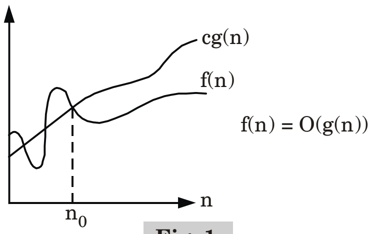


Fig. 1.

4. Answer any **one** part of the following : **(1 × 10 = 10)**

a. i. Differentiate between iteration and recursion.

Ans.

S. No.	Iteration	Recursion
1.	Allows the set of instructions to be repeatedly executed.	The statement in a body of function calls the function itself.
2.	Iteration includes initialization, condition, execution of statement within loop and update the control variable.	In recursive function, only termination condition (base case) is specified.

3.	The iteration statement is repeatedly executed until a certain condition is reached.	A conditional statement is included in the body of the function to force the function to return without recursion call being executed.
4.	Infinite loop uses CPU cycles repeatedly.	Infinite recursion can crash the system.
5.	Iteration is applied to iteration statements or loops.	Recursion is always applied to functions.
6.	Stack is not used.	Stack is used.
7.	Fast in execution	Slow in execution.

ii. Write the recursive solution for Tower of Hanoi problem.

Ans. Fig. 2 contains a schematic illustration of the recursive solution for TOWER (4, A, B, C) (4 disks, 3 pegs)

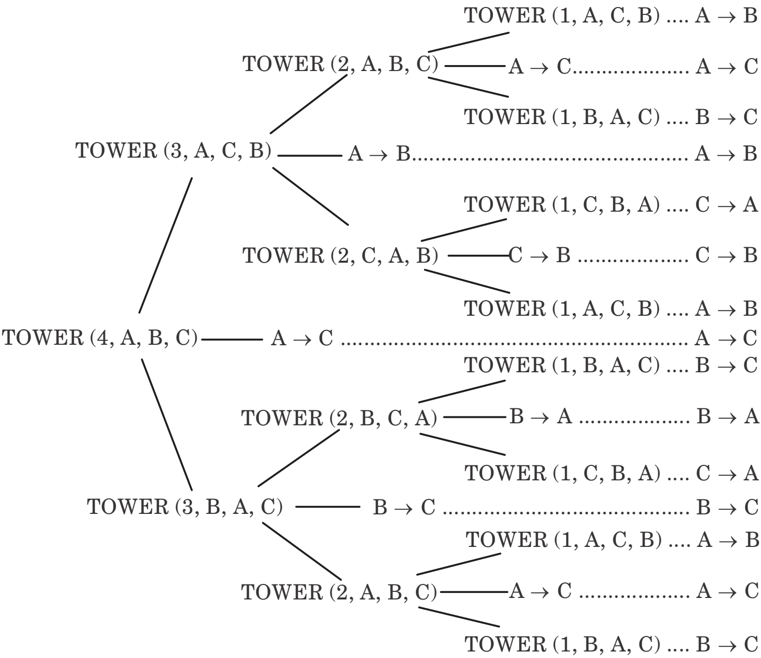


Fig. 2. Recursive solution to Tower of Hanoi problem for $n = 4$.

Observe that the recursive solution for $n = 4$ disks consist of the following 15 moves :

A → B A → C B → C A → B C → A C → B A → B A → C B → C
B → A C → A B → C A → B A → C B → C

b. Discuss array and linked representation of queue data structure. What is dequeue ?

Ans. Array representation of queue :

Algorithm to insert any element in a queue :

Step 1 : If REAR = MAX – 1
Write Overflow
Go to step 4. [End of if]

Step 2 : If FRONT = – 1 and REAR = – 1
Set FRONT = REAR = 0
else
Set REAR = REAR + 1 [End of if]

Step 3 : Set QUEUE[REAR] = NUM

Step 4 : Exit

Linked representation of queue :

Algorithm to insert an element in queue :

Step 1 : Allocate the space for the new node PTR.

Step 2 : Set PTR → DATA = VAL

Step 3 : If FRONT = NULL
Set FRONT = REAR = PTR
Set FRONT → NEXT = REAR → NEXT = NULL
else
Set REAR → NEXT = PTR
Set REAR = PTR
Set REAR → NEXT = NULL [End of if]

Step 4 : Exit

Algorithm for deletion of an element from queue :

Step 1 : If FRONT = NULL
Write Underflow
Go to Step 5 [End of if]

Step 2 : Set PTR = FRONT

Step 3 : Set FRONT = FRONT → NEXT

Step 4 : Free PTR

Step 5 : End

Dequeue :

1. In a dequeue, both insertion and deletion operations are performed at either end of the queues. That is, we can insert an element from the rear end or the front end. Also deletion is possible from either end.

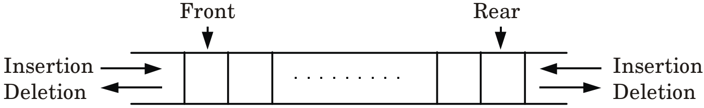


Fig. 2. Structure of a dequeue.

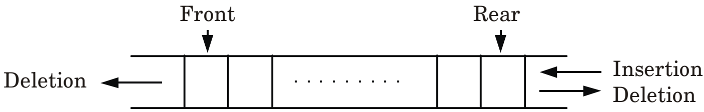
2. This dequeue can be used both as a stack and as a queue.
3. There are various ways by which this dequeue can be represented. The most common ways of representing this type of dequeue are :

a. Using a doubly linked list

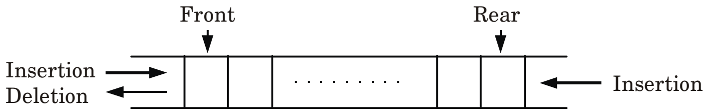
b. Using a circular array

Types of dequeue :

1. **Input-restricted dequeue :** In input-restricted dequeue, element can be added at only one end but we can delete the element from both ends.
2. **Output-restricted dequeue :** An output-restricted dequeue is a dequeue where deletions take place at only one end but allows insertion at both ends.



(a) Input-restricted dequeue



(b) Output-restricted dequeue

Fig. 3.

5. Answer any **one** part of the following : **(10 × 1 = 10)**

a. Why is quick sort named as quick ? Show the steps of quick sort on the following set of elements : 25, 57, 48, 37, 12, 92, 86, 33 Assume the first element of the list to be the pivot element.

Ans. **Quick sort named as Quick because :**

1. It works very fast in most practical cases with time complexity of O(n log n) in average case.
2. It does not need much extra memory *i.e.*, can be implemented in-place without time overheads.

Numerical :

1	2	3	4	5	6	7	8
25	57	48	37	12	92	86	33

Here $p = 1, r = 8$

$$x = A[1] = 25, i = p - 1 = 0, j = 1 \text{ to } 7$$

Now, $j = 1$ and $i = 0$

$A[1] = 25$

then $i = 0 + 1 = 1$ and $A[1] \leftrightarrow A[1]$

Now, $j = 2$ and $i = 1$

$A[2] = 57 \nleq 25$ (False)

$j = 3 \quad i = 1$

$A[3] = 48 \nleq 25$ (False)

$j = 4 \quad i = 1$

$A[4] = 37 \nleq 25$ (False)

$j = 5 \quad i = 1$

$A[5] = 12 < 25$ (True)

$$i = 1 + 1 = 2$$

Exchange $A[2] \leftrightarrow A[5]$

i.e.,

1	2	3	4	5	6	7	8
25	12	48	37	57	92	86	33

Now, $j = 6$ and $i = 2$

$$A[6] = 92 \nleq 25$$

$$j = 7 \text{ and } i = 2$$

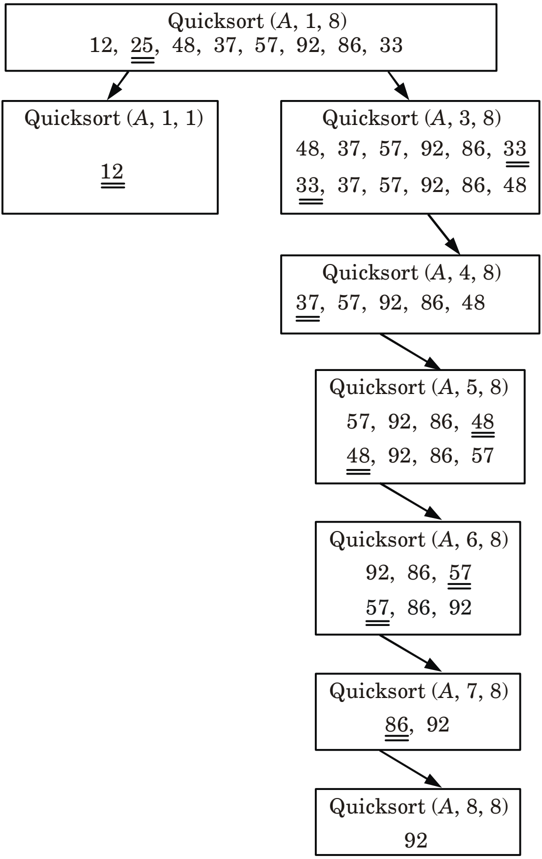
$$A[7] = 86 < 25$$

Exchange, $A[2] \leftrightarrow A[1]$

i.e.,

1	2	3	4	5	6	7	8
12	25	48	37	57	92	86	33

$$q = 2$$



Sorted array using quick sort

	1	2	3	4	5	6	7	8
$A =$	12	25	33	37	48	57	86	92

b. What is hashing ? Give the characteristics of hash function. Explain collision resolution technique in hashing.

Ans. Hashing :

1. Hashing is a technique that is used to uniquely identify a specific object from a group of similar objects.
2. Hashing is the transformation of a string of characters into a usually shorter fixed-length value or key that represents the original string.
3. In hashing, large keys are converted into small keys by using hash functions.
4. The values are then stored in a data structure called hash table.
5. The task of hashing is to distribute entries (key/value pairs) uniformly across an array.
6. Each element is assigned a key (converted key). By using that key we can access the element in O(1) time.
7. Using the key, the algorithm (hash function) computes an index that suggests where an entry can be found or inserted.
8. Hashing is used to index and retrieve items in a database because it is faster to find the item using the shorter hashed key than to find it using the original value.
9. The element is stored in the hash table where it can be quickly retrieved using hashed key which is defined by

   Hash Key = Key Value % Number of Slots in the Table

Characteristics of hash function :

1. The hash value is fully determined by the data being hashed.
2. The hash function uses all the input data.
3. The hash function "uniformly" distributes the data across the entire set of possible hash values.
4. The hash function generates very different hash values for similar strings.

Collision resolution technique :

Collision :

1. Collision is a situation which occur when we want to add a new record R with key k to our file F, but the memory location address $H(k)$ is already occupied.
2. A collision occurs when more than one keys map to same hash value in the hash table.

Collision resolution technique :

Hashing with open addressing :

1. In open addressing, all elements are stored in the hash table itself.
2. While searching for an element, we systematically examine table slots until the desired element is found or it is clear that the element is not in the table.
3. Thus, in open addressing, the load factor λ can never exceed 1.

4. The process of examining the locations in the hash table is called probing.
5. Following are techniques of collision resolution by open addressing :
a. Linear probing
b. Quadratic probing
c. Double hashing

Hashing with separate chaining :

1. This method maintains the chain of elements which have same hash address.
2. We can take the hash table as an array of pointers.
3. Size of hash table can be number of records.
4. Here each pointer will point to one linked list and the elements which have same hash address will be maintained in the linked list.
5. We can maintain the linked list in sorted order and each elements of linked list will contain the whole record with key.
6. For inserting one element, first we have to get the hash value through hash function which will map in the hash table, then that element will be inserted in the linked list.
7. Searching a key is also same, first we will get the hash key value in hash table through hash function, then we will search the element in corresponding linked list.
8. Deletion of a key contains first search operation then same as delete operation of linked list.

6. Answer any **one** part of the following : **(1 × 10 = 10)**

a. Explain Warshall's algorithm with the help of an example.

Ans. **Floyd's Warshall algorithm :**

1. Floyd Warshall algorithm is a graph analysis algorithm for finding shortest paths in a weighted, directed graph.
2. A single execution of the algorithm will find the shortest path between all pairs of vertices.
3. It does so in $\Theta(V^3)$ time, where V is the number of vertices in the graph.
4. Negative-weight edges may be present, but we shall assume that there are no negative-weight cycles.
5. The algorithm considers the "intermediate" vertices of a shortest path, where an intermediate vertex of a simple path $p = (v_1, v_2, ..., v_m)$ is any vertex of p other than v_1 or v_m, that is, any vertex in the set $\{v_2, v_3, ..., v_{m-1}\}$.
6. Let the vertices of G be $V = \{1, 2, ..., n\}$, and consider a subset $\{1, 2, ..., k\}$ of vertices for some k.
7. For any pair of vertices $i, j \in V$, consider all paths from i to j whose intermediate vertices are all drawn from $\{1, 2, ..., k\}$, and let p be a minimum-weight path from among them.
8. Let $d_{ij}^{(k)}$ be the weight of a shortest path from vertex i to vertex j with all intermediate vertices in the set $\{1, 2, ..., k\}$.
A recursive definition is given by

$$d_{ij}^{(k)} = \begin{cases} w_{ij} & \text{if } k = 0 \\ \min(d_{ij}^{(k-1)}, d_{ik}^{(k-1)} + d_{kj}^{(k-1)}) & \text{if } k \geq 1 \end{cases}$$

Floyd Warshall (w) :

$n \leftarrow$ rows $[w]$
$D^{(0)} \leftarrow w$
for $k \leftarrow 1$ to n
    do for $i \leftarrow 1$ to n
        do for $j \leftarrow 1$ to n
do $d_{ij}^{(k)} \leftarrow \min(d_{ij}^{(k-1)}, d_{ik}^{(k-1)} + d_{kj}^{(k-1)})$
return $D^{(n)}$

For example : Consider the graph :

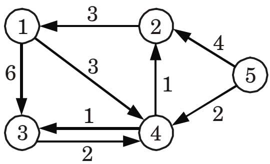


Fig. 4.

$$d_{ij}^{(k)} = \min[d_{ij}^{(k-1)}, d_{ik}^{(k-1)} + d_{kj}^{(k-1)}]$$

$$\pi_{ij}^{(k)} = \begin{cases} \pi_{ij}^{(k-1)} \text{ if } d_{ij}^{(k-1)} \leq d_{ik}^{(k-1)} + d_{kj}^{(k-1)} \\ \pi_{kj}^{(k-1)} \text{ if } d_{ij}^{(k-1)} > d_{ik}^{(k-1)} + d_{kj}^{(k-1)} \end{cases}$$

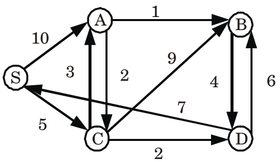
$$D^{(0)} = \begin{array}{c c} & \begin{array}{ccccc} 1 & 2 & 3 & 4 & 5 \end{array} \\ \begin{array}{c} 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array} & \begin{bmatrix} 0 & \infty & 6 & 3 & \infty \\ 3 & 0 & \infty & \infty & \infty \\ \infty & \infty & 0 & 2 & \infty \\ \infty & 1 & 1 & 0 & \infty \\ \infty & 4 & \infty & 2 & 0 \end{bmatrix} \end{array}$$

$$D^{(1)} = \begin{array}{c c} & \begin{array}{ccccc} 1 & 2 & 3 & 4 & 5 \end{array} \\ \begin{array}{c} 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array} & \begin{bmatrix} 0 & 4 & 6 & 3 & \infty \\ 3 & 0 & 9 & 6 & \infty \\ 6 & 4 & 0 & 2 & \infty \\ 4 & 1 & 1 & 0 & \infty \\ 7 & 4 & 3 & 2 & 0 \end{bmatrix} \end{array}$$

$$
D^{(2)} = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array}
\begin{array}{c}
\begin{array}{ccccc} 1 & 2 & 3 & 4 & 5 \end{array} \\
\begin{bmatrix}
0 & 4 & 6 & 3 & \infty \\
3 & 0 & 7 & 6 & \infty \\
6 & 3 & 0 & 2 & \infty \\
4 & 1 & 1 & 0 & \infty \\
6 & 3 & 3 & 2 & 0
\end{bmatrix}
\end{array}
$$

Now, if we find $D^{(3)}$, $D^{(4)}$ and $D^{(5)}$ there will be no change in the entries.

b. Describe the Dijkstra algorithm to find the shortest path. Find the shortest path in the following graph with vertex 'S' as source vertex.



Ans. Dijkstra algorithm :

a. Dijkstra's algorithm, is a greedy algorithm that solves the single-source shortest path problem for a directed graph $G = (V, E)$ with non-negative edge weights, *i.e.*, we assume that $w(u, v) \geq 0$ each edge $(u, v) \in E$.

b. Dijkstra's algorithm maintains a set S of vertices whose final shortest-path weights from the source s have already been determined.

c. That is, for all vertices $v \in S$, we have $d[v] = \delta(s, v)$.

d. The algorithm repeatedly selects the vertex $u \in V - S$ with the minimum shortest-path estimate, inserts u into S, and relaxes all edges leaving u.

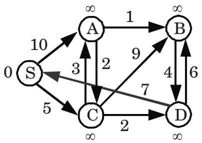
e. We maintain a priority queue Q that contains all the vertices in $v - s$, keyed by their d values.

f. Graph G is represented by adjacency list.

g. Dijkstra's always chooses the "lightest or "closest" vertex in $V - S$ to insert into set S, that it uses as a greedy strategy.

Numerical :

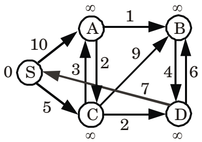
Initialize :



S = { }

Q :	S	A	B	C	D
	0	∞	∞	∞	∞

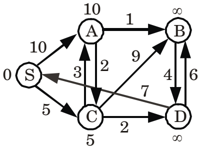
EXTRACT – MIN (S) :



S = { S }

Q :	S	A	B	C	D
	$\boxed{0}$	∞	∞	∞	∞

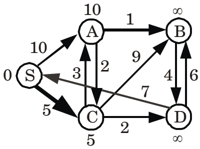
Relax all edges leaving (S) :



S = { S }

Q :	S	A	C	B	D
	$\boxed{0}$	∞	∞	∞	∞
		10	5	–	–

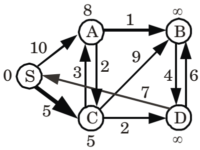
EXTRACT – MIN (C) :



S = { S, C }

Q :	S	A	C	B	D
	$\boxed{0}$	∞	∞	∞	∞
		10	$\boxed{5}$	–	–

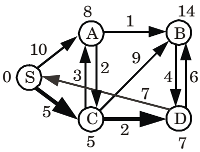
Relax all edges leaving C :



S = { S, C }

Q :	S	A	C	B	D
	$\boxed{0}$	∞	∞	∞	∞
		10	$\boxed{5}$	–	–
		8		14	7

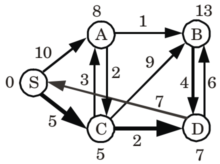
EXTRACT – MIN (D) :



S = { S, C, D }

Q :	S	A	C	B	D
	$\boxed{0}$	∞	∞	∞	∞
		10	$\boxed{5}$	–	–
		8		14	$\boxed{7}$

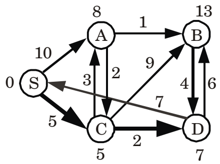
Relax all edges leaving D :



S = { S, C, D }

Q : S	A	C	B	D
$\boxed{0}$	∞	∞	∞	∞
	10	$\boxed{5}$	–	–
	8		14	$\boxed{7}$
			13	

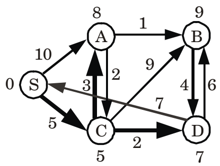
EXTRACT – MIN (A) :



S = { S, C, D, A }

Q : S	A	C	B	D
$\boxed{0}$	∞	∞	∞	∞
	10	$\boxed{5}$	–	–
	$\boxed{8}$		14	$\boxed{7}$
			13	

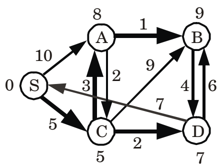
Relax all edges leaving A :



S = { S, C, D, A }

Q : S	A	C	B	D
$\boxed{0}$	∞	∞	∞	∞
	10	$\boxed{5}$	–	–
	$\boxed{8}$		14	$\boxed{7}$
			9	

EXTRACT – MIN (B) :



S = { S, C, D, A, B }

Q : S	A	C	B	D
$\boxed{0}$	∞	∞	∞	∞
	10	$\boxed{5}$	–	–
	$\boxed{8}$		14	$\boxed{7}$
			$\boxed{9}$	

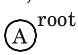
7. Answer any **one** part of the following : **(7 × 1 = 7)**

a. Can you find a unique tree when any two traversals are given ? Using the following traversals construct the corresponding binary tree :
INORDER : ***HKDBILEAFCMJG***
PREORDER : ***ABDHKEILCFGJM***
Also find the post order traversal of obtained tree.

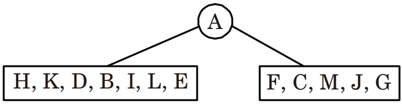
Ans. No, we cannot find unique tree when any two traversals are given. If preorder and postorder are given then we cannot find unique tree. We can find unique tree if one of the given traversal is inorder.

Numerical :

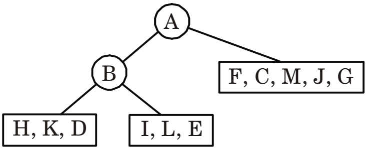
Step 1 : In preorder traversal root is the first node. So, *A* is the root node of the binary tree. So,



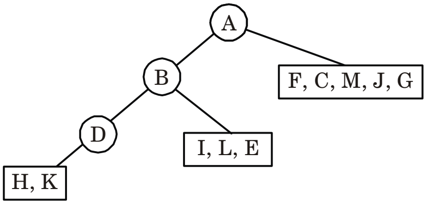
Step 2 : We can find the node of left sub-tree and right sub-tree with inorder sequence. So,



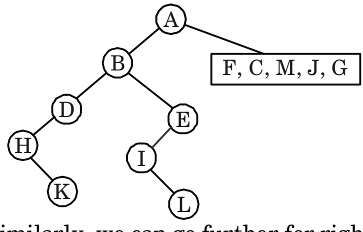
Step 3 : Now, the left child of the root node will be the first node in the preorder sequence after root node *A i.e*. *B* So,



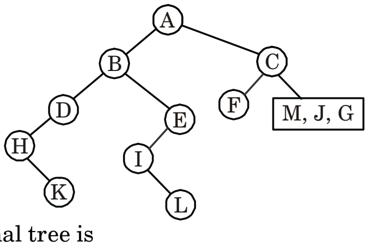
Step 4 : Now the root node is *D*. In inorder sequence, *H*, *K* is on the left side of *D*. So



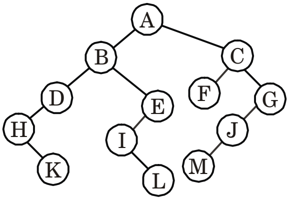
Step 5 : Now the root is *H*. In inorder sequence, *K* is on the right side of *H*.



Step 6 : Similarly, we can go further for right side of *A*.



So, the final tree is



Postorder of tree : *K, H, D, L, I, E, B, F, M, J, G, C, A*

b. What is a B-Tree ? Generate a B-Tree of order 4 with the alphabets (letters) arrive in the sequence as follows : ***a g f b k d h i n j e s i r x c l n t u p.***

Ans. B-tree :

1. A B-tree is a self-balancing tree data structure that keeps data sorted and allows searches, sequential access, insertions, and deletions in logarithmic time.
2. A B-tree of order m is a tree which satisfies the following properties :

a. Every node has at most m children.

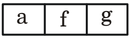
b. Every non-leaf node (except root) has at least $\lceil m/2 \rceil$ children.

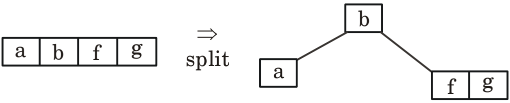
c. The root has at least two children if it is not a leaf node.

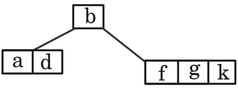
d. A non-leaf node with k children contains $k - 1$ keys.

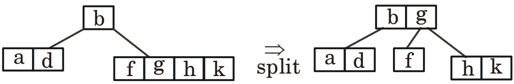
e. All leaves appear in the same level.

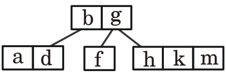
Construction of B-tree :

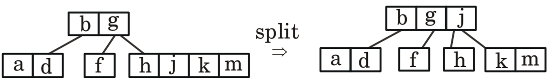
Insert a, g, f : 

Insert b : 

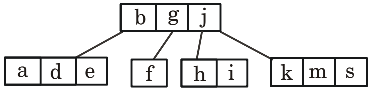
Insert k, d : 

Insert h : 

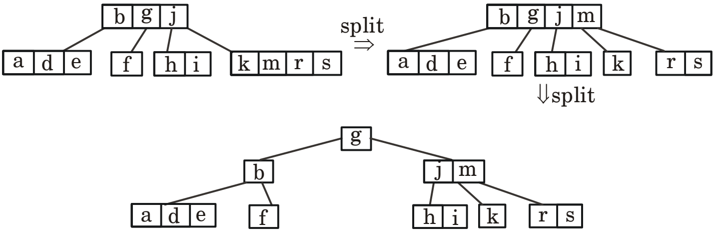
Insert m : 

Insert j : 

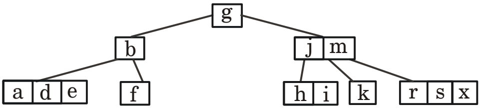
Insert e, s, i :



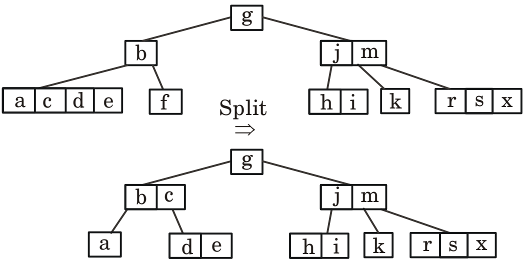
Insert r :



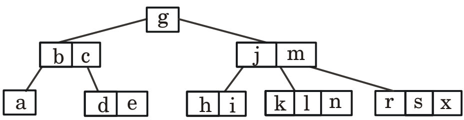
Insert x :



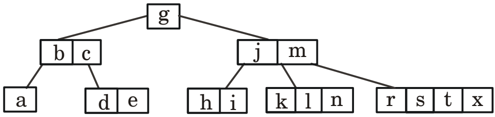
Insert c :



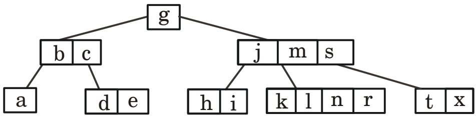
Insert l, n :



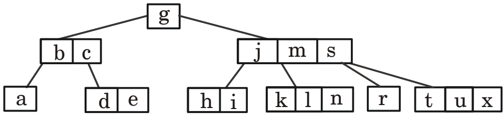
Insert t :



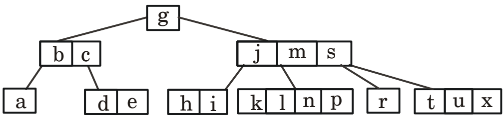
⇓ Split



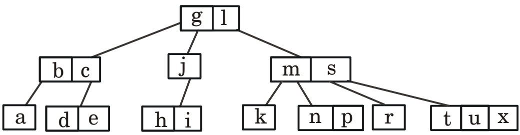
Insert u :



Insert p :



⇓ Split



☺☺☺