Data Structures Notes

Contributor: Abhishek Sharma

[Founder at TutorialsDuniya.com]

Computer Science Notes

Data structure:-

*A data structure is a specialised format which is used for organising and storing the data in a proper manner.

*usually data structures are classified into 2 types they are 1) linear data structures
2) non-linear data structures

Linear data structures:- These data structures will organise and store the data in a linear manner (or) sequencial order

Ex:- Arrays, stacks, queues, linked lists

Non-linear data structures:- These data structures will organise and store the data in a non-linear manner (or) random order

ex:- Trees, Graphs.

*Data structures are mainly used in computer memory to perform various tasks by the computer the computer will perform different operations by using different data structures inside its memory

Data:- whatever we are entering into computer is known as data.

Process:- the data in the execution is known as process

Information:- The processed data is known as Information

Program:- A program is a set of instructions.

Algorithm:- An algorithm is a step by step process of a program (or) problem

Software:- A software is a set of programs.

Ex:- MS-office, OS, unix, Linux etc...

Hard ware:- Hard ware is an inter connection of various hard ware devices such as mouse, keyboard, hard disk etc..

Language:- A language is a medium between user and computer the language has been divided into two types in the computer they are 1) low level language 2) high level language

Low level language:- It is the combination of two languages such as machine level language and assembly level language

Machine level language:- It is computer understandable language and it can also be called as binary language why because it will be in the form of 0's and 1's.

Ex:- 1010 → 10
1000 → 8

Assembly level language:- It can be understable by human and it can also be called as symbolic language why because it would be in the form symbolic notations.

Ex:- load, XCH, MOV, Add, sub etc...

High level language:- It would be in the form of english statements by combining numericals, alphabets and special characters.

* In the high level language has be divided into 2 types

1) procedure oriented programing language
2) object oriented programing language

procedure oriented programing language:- In the procedure oriented programing language a problem (or) program will be divided into group of functions

Ex:- C, COBAL, PASCAL, FORTRAN, BASIC etc..

object oriented programing language:- In this problem (or) program is divided into group of objects

Ex:- C++, JAVA, .Net, small talks, etc..

Translator:- A translator is a software program which converts source language instructions into destination language instructions

Compiler:- compiler is a translator which converts high level language instructions into machine level language instructions and it also compile (or) check the errors in the program.

Assembler:- It is also one of the translator which converts assembly language instructions into machine language instructions.

UNIT-I

Arrays

Basic concepts of OOP's:- The basic concepts of Oop's have been classified into six major types

1) class
2) object
3) Inheritance
4) polymorphism
5) Encapsulation
6) Abstraction

class:- A class is a collection of data members (variables (or) attributes (or) fields (or) properties) and member functions (functions (or) operations)

* In order to create a class use a keyword known as class
* usually the class is used to generate user defined data type. through which objects will be created

Syntax:-

```
class classname
{
    private:
        Data members;
        member functions;
    protected:
        Data members;
        member functions;
    public:
        Data members;
        member functions;
};
```

private, protected, public } access specifiers

Private:- If we declare the data members and member functions under private access specifier then they can be accessed with in the class only.

protected:- If we declare the data members and member functions under protected access specifiers then they can be accessed with in the class and inherited class

public:- If we declare the data members and member functions under public access specifiers then they can be accessed with in the class and inherited class and outside the class.

Ex:-
```
class student
{
private:
    int sno;
    char sname;
    void scount();
protected:
    ----
    ----
public:
    ----
    ----
};
```

Object:- Each class variable can be called as an object

* An object is mainly used to invoke / call the member functions for their execution.

Syntax:- classname objname;

Simple Program for Class & Object:

```
#include<iostream.h>
#include<conio.h>
class person
{
 char name[30];
 int age;
public:
  void getdata();
  void display();
};
void person::getdata()
{
cout<<"Enter the name:";
cin>>name;
cout<<"Enter the age:";
cin>>age;
}
void person::display()
{
 cout<<"\n Name is:"<<name;
 cout<<"\n Age is:"<<age;
}
int main()
{
 person p;
 p.getdata();
 p.display();
 return 0;
}
```

Inheritance:- The process of creating a new class from already existed class is known as an Inheritance.

where the existed class can be called as Base class (or) parent class (or) Super class and the new class can be called as derived class (or) child class (or) sub class.

Syntax:-

```
class derived class-name :visibility-mode Base class-name
{
  ....  //members of derived class
}
```

Inheritance is mainly used for achieving code reusability

Note:- If we not declare data members and member functions under another any access specifiers in the class then those data members and member functions will be considered by default under private access specifier

* The visibility mode is optional that may be either private (or) public

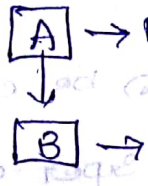
Types of Inheritance:- An Inheritance has been classified into 5 types

1) Single level Inheritance
2) Multi-level Inheritance
3) Multiple inheritance
4) Hierarchical Inheritance
5) Hybrid Inheritance

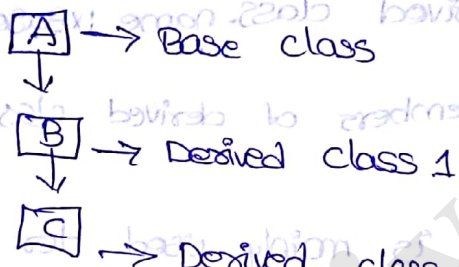
Single level Inheritance:-

If a class is derived from only single base

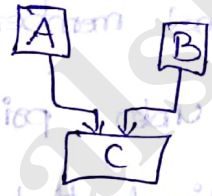
class then such inheritance is known as Single level Inheritance.



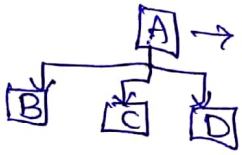
Multi-level Inheritance:- If a class is derived from already derived class then such Inheritance is known as multi-level Inheritance



Multiple Inheritance:- If a class is derived from more than one base class then such type of Inheritance is known as multiple Inheritance.



Hierarchical Inheritance:- If more no. of derived class are derived from single base class such as Inheritance is known as hierarchical Inheritance.



Hybrid Inheritance:- combination of more than one Inheritances is called hybrid inheritance

Example program for single level inheritance;

```
#include<iostream.h>
#include<conio.h>
class A
{ public:
  int a,b; data members
public:
    void getdata(); member function
    };
class B: public A
  {
    int c,d;
  public:
    void add();
    void sub();
    void display();
  };
void A::getdata()
  {
  cout<<"Enter a value:"<<endl;
  cin>>a;
  cout<<"enter b value:"<<endl;
  cin>>b;
  }
void B::add()
  {
  c=a+b;
  }
void B::sub()
  {
  d=a-b;
  }
void B::display()
  {
  cout<<"Addition is:"<<c<<endl;
  cout<<"substraction is:"<<d;
  }
int main()
{
B obj;
clrscr();
obj.getdata();
obj.add();
obj.sub();
obj.display();
getch();
return 0;
}
```

\n occupies space in the memory

endl does not occupies "

Polymorphism:-

* The polymorphism is a greek term and which is used to represent single form into more than one form
* The polymorphism has been divided into two types they are 1) compile time polymorphism
  2) Run time polymorphism

Compile time polymorphism:-

If the polymorphism is done during the compilation time then such polymorphism is said to be compile time polymorphism.

Eg:- function overloading, operator overloading.

Run time Polymorphism:-

If the polymorphism is done during the run time then such polymorphism is said to be run time polymorphism

Eg:- Virtual functions.

Encapsulation:- The process of wrapping (or) grouping the data members and members functions into a single unit (class) is known as encapsulation

* The encapsulation concept can we acheived by creating class
* Using the encapsulation we can provide security for the data members and member functions. by making them as private in the class.

Abstraction:- The process of hiding unnecessary information inside and revealing (or) displaying required information outside is known as abstraction.

Abstract data types:-

* An abstract data type is a collection of data elements and associated operations
* An abstract data type is also a user defined data type like as class
* Usually Arrays, linked lists, stacks, queues, sets, Graphs are said to be abstract data types.

→ for stack ADT (Abstract data type) basic operations may be push & pop

→ for the queue ADT basic operations may be enqueue and Dequeue

→ for the set ADT the basic operations may be union, intersection, complement and size etc..

→ for the graph ADT the basic operations may be no.of vertices, no.of edges etc.

- the main purpose of abstract data type is write the operations of ADT once in the program and perform their from any part of the program by calling their appropriate functions

Polynomial representation using arrays:-

A polynomial is an expression which contains more than one term.

means combination of operators & operands

Each term may be made with one (or) more exponents and co-efficients

Ex:

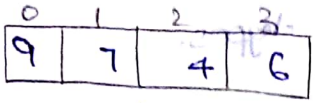
$$P(x) = 6x^3 + 4x^2 + 7x + 9$$

In order to represent single variable polynomial we

use single dimension array

similarly to represent multivariable polynomial we use multi-dimensional array.

The indexes of the array can be considered as exponents of the polynomial and the coefficients will be stored at particular indexes of the array



Array

// Arrays as an abstract data type:-

An array is the basic abstract data type because which contains collection of homogeneous elements and associated operations such as Traversing, Insertion, Deletion, Searching, Sorting, Merging

The array elements can be stored in consecutive locations in the memory and they can be accessed by using integer index set and the array index starts with 'zero'.

The no.of elements stored in the array will be called as the length of the array and which can be determined by the following formula

$$\text{Length} = UB - LB + 1$$

where UB = upper boundary index

LB = lower boundary index

The elements of the array can be denoted using different notations

subscript notation $(A_0, A_1, A_2 \ldots)$

parenthesis (A(0), A(1), A(2), ...)

Bracket notation (A[0], A[1], A[2], ...)

let A[K] be the array and 'K' is an index (or) subscript

Representation of the arrays in the memory

Consider A[K] be the array and the (A[K]) is the location address

Usually the elements of the array will be stored in consecutive memory location in the computer

A[ ] = {10, 20, 30, 40, 50};

A[0]	A[1]	A[2]	A[3]	A[4]
10	20	30	40	50
Base - 1000	1001	1002	1003	1004

* The computer does not need to keep the addresses of all the elements in the array but needs to keep only the address of first element in the array. which is known as base address of the array and which can be denoted by Base(A)

* the computer can easily calculate the location address of any element using the base address of the array by the following formula.

$$Loc(A[K]) = Base(A) + W(K - LB)$$

where W = memory space allocated for each element

K = index

LB = lower boundary index

Construct the elements A[6] be the array

A[6] = {10, 20, 30, 40, 50}; and the memory space is allocated for each element is 1 byte

A[0]	A[1]	A[2]	A[3]	A[4]	A[5]
10	20	30	40	50	
100	101	102	103	104	105

find out location address of (A[4])

TutorialsDuniya.com

Download FREE Computer Science Notes, Programs, Projects, Books PDF for any university student of BCA, MCA, B.Sc, B.Tech CSE, M.Sc, M.Tech at https://www.tutorialsduniya.com

- Algorithms Notes
- Artificial Intelligence
- Android Programming
- C & C++ Programming
- Combinatorial Optimization
- Computer Graphics
- Computer Networks
- Computer System Architecture
- DBMS & SQL Notes
- Data Analysis & Visualization
- Data Mining
- Data Science
- Data Structures
- Deep Learning
- Digital Image Processing
- Discrete Mathematics
- Information Security
- Internet Technologies
- Java Programming
- JavaScript & jQuery
- Machine Learning
- Microprocessor
- Operating System
- Operational Research
- PHP Notes
- Python Programming
- R Programming
- Software Engineering
- System Programming
- Theory of Computation
- Unix Network Programming
- Web Design & Development

Please Share these Notes with your Friends as well

Loc (A[K]) = Base (A) + w(K - LB)

Loc (A[4]) = 100 + 1 (4 - 0)

= 100 + 4

Loc (A[4]) = 104.

// Polynomial addition:-

Algorithm:-

Assume these are two polynomials P & Q

The addition of two polynomials is stored in third polynomial Sum

Step (1) Start

Step (2) while P & Q are not null, repeat Step 3

Step (3) If the powers of two terms of the polynomi are equal then

Insert the addition of two terms into Sum polynomial

Else if the x power of first polynomial > the pow term x power of secon polynomial then, insert the term of first polynomial in Sum polynomial

Else

Insert the term of second polynomial into Sum polynom

Step (4)

Copy the remaining terms from the two polynomials into Sum polynomial

Step (5)

Print Sum polynomial

Step (6)

Stop.

Eg: $P(x) = 2x^4 + 5x + x$

$Q(x) = 5x^4 + 4x^3 + 2x^2 + 3x - 14$

$Sum(x) = 7x^5 + 5x^4 + 4x^3 + 7x^2 + 4x - 14$

Program-

```
# include <iostream.h>
# include <conio.h>
void main( )
{
 int  a[6], b[6], c[6], i, flag=0;
 clrscr( );
 for(i=0; i<6; i++)
 {
  a[i]=0 ;
  b[i] =0;
 }
 a[1] =1; a[2]= 5; a[5] = 7;
 b[0] = -14; b[1]= 3; b[2]= 2;
 b[3] =4; b[4] =5;
    for (i=5 ; i>0; i--)
    {
     if (a[i] ! =0 && i>0)
     {
      cout <<a[i] <<"x" <<i;
      flag = 1;
     }
    if (i>0 && a[i-1] !=0 && flag = = 1)
      cout << "+";
      if (a[i] ! =0 & & i==0)
      cout <<a[i];
     }
     flag=0;
```

```
cout<<"second polynomial is:";
for (i=5; i>=0; i--)
{
 if (b[i]!=0 && i>0)
   {
    cout << b[i] <<"x"<<i;
        flag=1;
       }
    if (i>0 && b[i-1]>0 && flag ==1)
    cout<<"+";
    if (b[i]!=0 && i==0)
    cout << b[i];
   }
  cout<<"polynomial addition is:";
      for (i=6; i>0; i++)
          {
           c[i] = a[i]+b[i];
          }
cout<<"the resultant polynomial is:";
 for (i=5; i>=0; i--)
 {
  if (c[i]!=0 && i>0)
  {
  cout << c[i] <<"x"<<i;
    flag=1;
  }
  if (i>0 && c[i-1]>0 && flag==1)
    cout<<" + ";
  if(c[i]!=0 && i==0)
   cout<<c[i];
 }getch();
```

a[5]	a[4]	a[3]	a[2]	a[1]	a[0]
7	0	0	5	1	0

⇒ 7x5 + 5x2 + 1x1

b[6]

b[5]	b[4]	b[3]	b[2]	b[1]	b[0]
0	5	4	2	3	-14

⇒ 5x4 + 4x3 + 2x2 + 3x1 - 14

c[6]

c[5]	c[4]	c[3]	c[2]	c[1]	c[0]
7	5	4	7	4	-14

⇒ 7x5 + 5x4 + 4x3 + 7x2 + 4x1 - 14

Matrix Multiplication:-

```
#include<iostream.h>
#include<conio.h>
  int main()
  {
int a[10][10], b[10][10], c[10][10];
int m,n,p,q, i,j,k;
clrscr();
cout<<"Enter the no.of rows & columns of first matrix:"<<
 endl;
cin>>m>>n;
cout<<"Enter the no.of rows & columns of second matrix:"<<endl;
cin>>p>>q;
if (n==p)
{
cout<<"Enter the elements of first matrix is:"<<endl;
for (i=0; i<m; i++)
 {
 for (j=0; j<n; j++)
  {
   cin>>a[i][j];
 }}
```

```
cout<<" Enter the element of second matrix is:"<<endl;
for(i=0; i<p; i++)
{
 for(j=0; j<q; j++)
 {
  cin>>b[i][j];
 }
}
cout<<"the matrix A is:"<<endl;
 for(i=0; i<m; i++)
  {
  for(j=0; j<n; j++)
   {
   cout<<" "<<" "<<" "<<a[i][j]<<endl;
   }
  }
 cout<<"the matrix B is:"<<endl;
  for(i=0; i<p; i++)
   {
    for(j=0; j<q; j++)
    {
     cout<<" "<<" "<<" "<<b[i][j]<<endl;
    }
   }
  cout<<"The resultant matrix is:"<<endl;
  for(i=0; i<m; i++)
   {
   for(j=0; j<q; j++)
    {
     c[i][j]=0;
     for(k=0; k<p; k++)
     {
     c[i][j] = c[i][j] + (a[i][k] * b[k][j]);
     }
```

o/p:-

```
enter the no.of rows and columns of first matrix:
3 3
enter the no.of rows and columns of second matrix:
3 3
enter the elements of first matrix is:
1 4 5
2 3 6
7 8 9
enter the elements of second matrix is:
4 5 6
1 0 3
2 6 9
the matrix A is:
1
4
5
2
3
6
7
8
9
the matrix B is:
4
5
6
1
0
3
2
6
9
The resultant matrix is:
18
35
63
23
46
75
54
89
147
```

```
    }
  cout<<" "<<" "<<" "<<c[i][j]<<endl;
  }
else
cout<<"matrix multiplication is not possible"<<endl;
  getch();
  return 0;
}
```

Sparse Matrix:-

* Usually the two dimensional array can be used to represent Matrices.
* If a matrix contains more number of zero values than non zero values. then such type of matrix can be called as "sparse matrix."
* In contrast, If a matrix contains more number of non zero values than zero values then such type of matrix can be called as "dense matrix."

Eg:- consider a matrix size of 5x6 & contains 6 non-zero elements

$$\begin{array}{c} \\ 0 \\ 1 \\ 2 \\ 3 \\ 4 \end{array} \begin{array}{c} \begin{array}{cccccc} 0 & 1 & 2 & 3 & 4 & 5 \end{array} \\ \begin{pmatrix} 0 & 0 & 0 & 0 & 9 & 0 \\ 0 & 8 & 0 & 0 & 0 & 0 \\ 4 & 0 & 0 & 2 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 5 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix} \end{array}$$

* when the sparse matrix is represented in 2 dimensional array then while implementing in computers memory, A lot of memory space will be wasted and C.P.U access time is also increased while accessing non-zero elements

by scaning all the elements (0 and non-zero elements) in the sparse matrix

* In order to avoid memory space wastage and to decrease cpu access time we represent the sparse matrix into two ways they are 1) Triplet representation 2) linked representation

Triplet Representation:-

* In the Triplet representation we consider only non-zero values along with their rows and columns
* In this representation, the zeroth row stores the total no.of rows, total no.of columns and total no. non-zero values existed in the sparse matrix
* From the above example the sparse matrix can be represented in triplet representation as

	0	1	2	3	4	5
0	0	0	0	0	9	0
1	0	8	0	0	0	0
2	4	0	0	2	0	0
3	0	0	0	0	0	5
4	0	0	0	0	0	0

Row	column	value
5	6	6
0	4	9
1	1	8
2	0	4
2	3	2
3	5	5
4	2	2

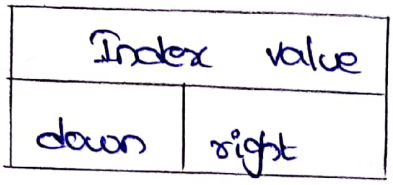
Linked representation:-

* In the linked representation the sparse matrix can be represented using linked lists. In this, the linked lists use two different nodes they are
  1. Header node
  2. element node

Header Node:- The header node can be used to specify (or) specify index values which can be either row wise (or) column wise

* The header node contains 3 fields such as Index value, down, right fields

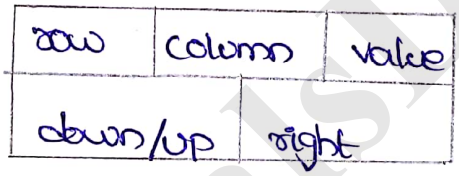
Header node

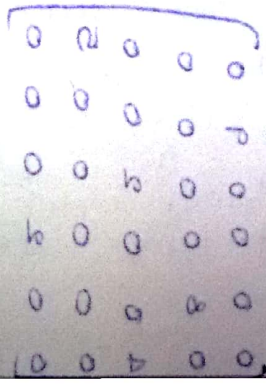


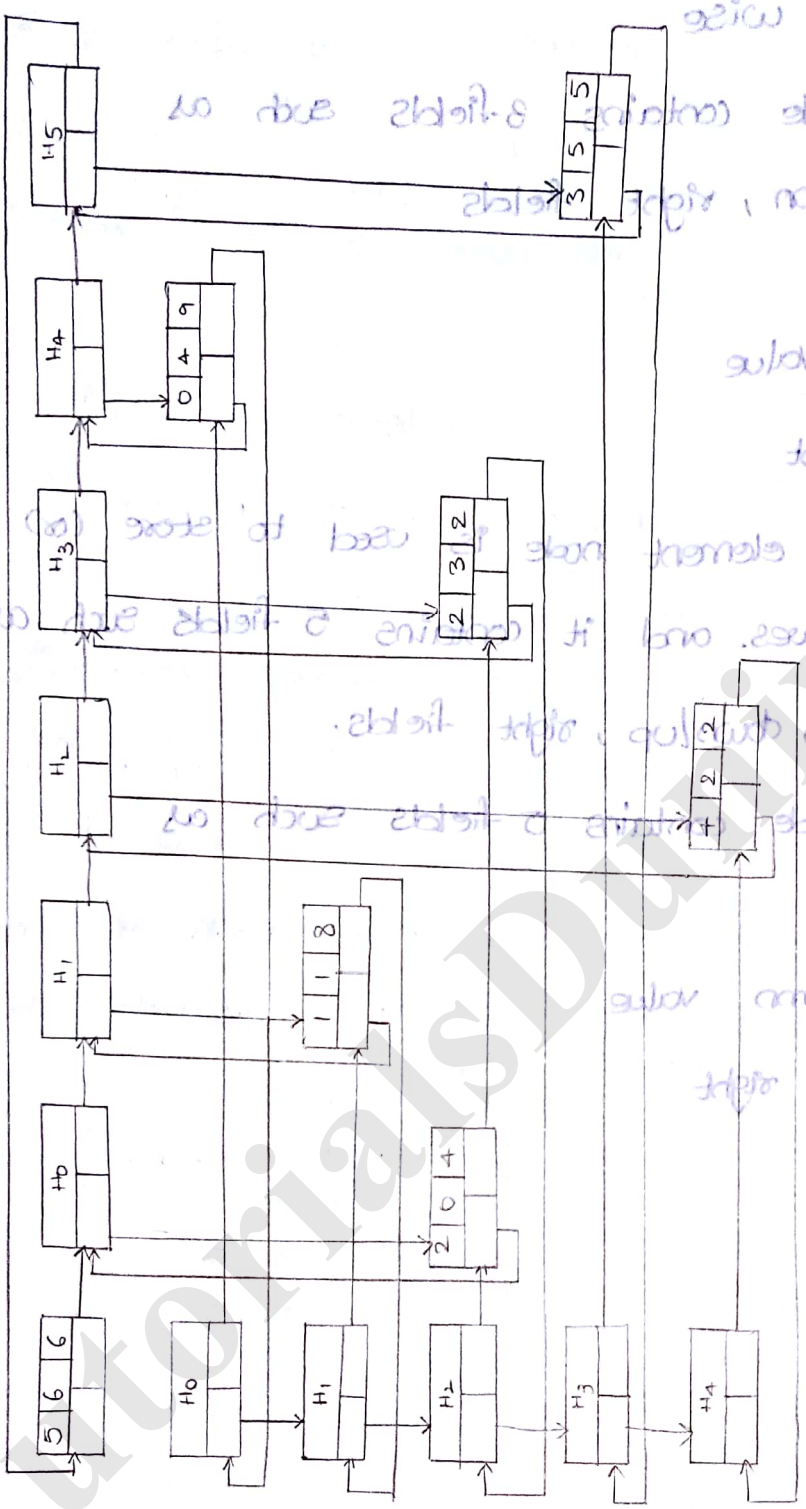
Element node:- An element node is used to store (or) specify non-zero values. and it contains 5 fields such as row, column, value, down/up, right fields.

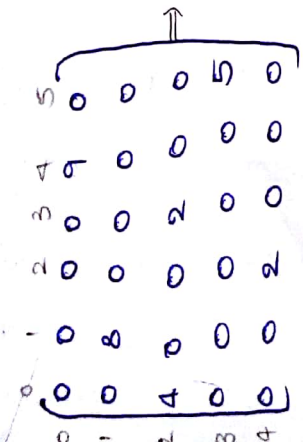
* The element node contains 5 fields such as

Element node:-









Transposing from Triplet Sparse Matrix

```
#include<iostream.h>
#include<conio.h>
Void main()
{
 int S[7][3], t[7][3];
 int r,c,nzvs,i;
 clrscr();
 cout<<" Enter no.of rows, columns and non zero values of
      Sparse triplet from matrix:"<<endl;
 cin>>r>>c>>nzvs;
  S[0][0]=r;
  S[0][1]=c;
  S[0][2]=nzvs;
 cout<<" Reading the elements of sparse triplet from matrix:"
                                                  <<endl;
for(i=0;i<=nzvs;i++)
{   cout<<"Enter the next triplet (row, column, value):"<<endl;
  cin>>S[i][0]>>S[i][1]>>S[i][2];
cout<<"The Sparse triplet from matrix is:"<<endl;
for(i=0;i<=nzvs;i++)
{
cout<<"rows"<<"\t"<<"columns"<<"\t"<<"values"<<endl;
cout<<S[i][0]<<"\t"<<S[i][1]<<"\t"<<S[i][2]<<endl;
}
cout<<"transposing Sparse triplet from matrix is:"<<endl;
for(i=0;i<=nzvs;i++)
 {
  t[i][0]=S[i][1];
  t[i][1]=S[i][0];
  t[i][2]=S[i][2];
 }
cout<<"the transposed sparse tir triplet from matrix is:"<<
                                                  endl;
```

```
for(i=0; i<=nzvs;i++)
{
cout<<"row"<<"\t"<<"column"<<"\t"<<"value"<<endl;
cout<<t[i][0]<<"\t"<<t[i][1]<<"\t"<<t[i][2]<<endl;
}
getch();
}
```

I/p:-

Enter no of rows, columns and non zero values of sparse triplet fo

matrix: 5 6 6

reading the elements into sparse triplet form matrix:

enter the next triplet (row, column value):
0 4 9

enter the next triplet (row, column value):
1 1 8

enter the next triplet (row, column value):
2 0 4

enter the next triplet (row, column value):
2 3 2

enter the next triplet (row, column value):
3 5 5

enter the next triplet (row, column value):
4 2 2

enter the next triplet (row, column value):
6 6 3

rows	columns	values
0	4	9
1	1	8
2	0	4
2	3	2
3	5	5
4	2	2
6	6	3

transposing sparse triplet from matrix is:
the transposed sparse triplet from matrix is:

	0	1	2	3	4	5	6
0	0	0	4	0	0	0	0
1	0	8	0	0	0	0	0
2	0	0	0	2	0	0	0
3	0	0	2	0	0	0	0
4	9	0	0	0	0	0	0
5	0	0	0	5	0	0	0
6	0	0	0	0	0	0	3

Representation of Arrays:

* An array is a collection of homogeneous elements

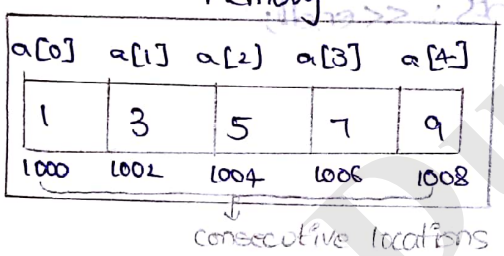
Usually arrays are classified into two major types

1. Single dimensional array
2. Multi dimensional array

Representation of Single dimensional array in the memory:

* The single dimensional array can be represented in the memory as in the form of consecutive locations which mean that the elements of single dimensional array will be stored in the memory in consecutive locations

Eg: int a[] = {1,3,5,7,9};



Representation of Multi dimensional array in the memory:

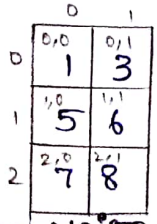
* The multi dimensional array can be represented in the memory in the form of matrix by using 2 types of forms they are i) Row-major order

ii) column-major order

Row-Major order:-

In this multi-dimensional array elements will be stored (or) represented row by row

Eg: int a[3][2];



Column Major order:-

In this the elements of multi-dimensional array can be represented column by column in row wise

Eg:- int a[3][2]

o/p →

	0	1	2
0	1	5	7
1	3	6	8

Input :-

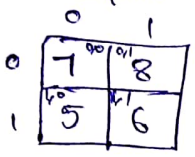
1 3
5 6
7 8

Program to illustrate row-major order & column major-order

```
#include <iostream.h>
#include <conio.h>
void main()
{
int a[10][10], m,n,i,j;
clrscr();
cout<<"Enter the no.of rows & columns :"<<endl;
cin >> m>>n;
cout<<"Enter the elements :"<<endl;
for (i=0; i<m; i++)
{
for (j=0; j<n ; j++)
{
cin>>a[i][j];
}
}
cout<<" the row-major order is:"<<endl;
for (i=0; i<m; i++)
{
for(j=0 ;j<n ; j++)
{
cout <<a[i][j]<<"\t";
}
cout <<endl;
}
cout<<" the column -major order is:"<<endl;
for(i=0;i<m; i++)
{
```

```
for(j=0;j<m;j++)
{
cout<<a[j][i]<<"\t";
}
cout<<endl;
}
getch();
}
```

O/p:- Input



O/p:- Enter the no.of rows and columns:
2 2
enter the elements:
7 8
5 6

Row-major order is:

7 8
5 6

column-major order is:

7 5
8 6

TutorialsDuniya.com

Download FREE Computer Science Notes, Programs, Projects, Books PDF for any university student of BCA, MCA, B.Sc, B.Tech CSE, M.Sc, M.Tech at https://www.tutorialsduniya.com

- Algorithms Notes
- Artificial Intelligence
- Android Programming
- C & C++ Programming
- Combinatorial Optimization
- Computer Graphics
- Computer Networks
- Computer System Architecture
- DBMS & SQL Notes
- Data Analysis & Visualization
- Data Mining
- Data Science
- Data Structures
- Deep Learning
- Digital Image Processing
- Discrete Mathematics
- Information Security
- Internet Technologies
- Java Programming
- JavaScript & jQuery
- Machine Learning
- Microprocessor
- Operating System
- Operational Research
- PHP Notes
- Python Programming
- R Programming
- Software Engineering
- System Programming
- Theory of Computation
- Unix Network Programming
- Web Design & Development

Please Share these Notes with your Friends as well

Stacks & Queues

Templates:-

* A template is a method for creating a single function (or) class for a family of similar type functions (or) classes for making (or) creating generic program
* Basically templates are classified into two types they are
  i) function template
  ii) class template

function template:-

If a function is created for a group of similar type functions using the templates then such function can be called as function template

Syntax:-

```
template <class T>
T function_name(T argument)
{
  :
  :
  return(T);
}
```

* where template, class are the keywords in C++ and T is a generic data type (or) parameterised data type
* the function template must contain atleast one generic data type argument.

Ex:-

```
#include<iostream.h>
#include<conio.h>
  template <class T>
  T getmax (T a,T b)
    {
  T result;
  result=(a>b) ? a:b;
```

```
return result;
}
int main()
{
 int ix, iy;
 float fx, fy;
 clrscr();
 cout<<"enter any two integer values:"<<endl;
 cin>>ix>>iy;
 cout<<"Enter any two floating point values:"<<endl;
 cin>>fx>>fy;
 int k=getmax(ix,iy);
 float n=getmax(fx,fy);
 cout<<"the max value in integers is:"<<k<<endl;
 cout<<"the max value in float values is:"<<n<<endl;
 getch();
 return 0;
}
```

```
enter any two integer values:
8
enter any two floating point values:
9.5
5.8
The max value in integers is: 8
The max value in float is: 9.5
```

Class Template:-

* If a class is created for a group of similar classes using template keyword then such class can be called as class template
* The class template can work on different data types without rewritting specific class for each data type

Syntax:-

```
template <class T>
class class_name
{
private:
 :
public:
 :
```

```
#include<iostream.h>
#include<conio.h>
template<class T>
class Sample
{
private:
T a,b;
public:
void getdata();
void Sum();
};
template<class T>
void Sample<T>::getdata()
{
cin>>a>>b;
}
template<class T>
Void Sample<T>::Sum()
{
T c;
c=a+b;
Cout<<"Sum is="<<c<<endl;
}
void main()
{
Sample<int> obj1;
Sample<float> obj2;
clrscr();
cout<<"Enter any two integer values:"<<endl;
obj1.getdata();
obj1.Sum();
cout<<"Enter any two floating point values:"<<endl;
Obj2.getdata();
obj2.Sum();
getch();
}
```

O/P:

```
Enter any two integer values:
5  8
Sum is=13
Enter any two floating point values:
3.5  8.5
Sum is=12
```

Note:- The templates which may be either function template (or) class template do you work on different data types to perform different operations without re-writting specific function (or) class for each data type

Advantage:- The templates can reduce the length of the program code by writting once and used by many different data types

Stack ADT:- → It must be elements and deals with operations

* A Stack is an abstract data type because it contains (or) stores set of elements and has some associated operations
* A stack is a linear data structure which stores the elements in a sequencial order.
* Usually the stack follows a principle known as LIFO (last-in-first-out) which mean that the element which is inserted last into the stack will be deleted first

There are two operations will be performed on the stack they are 1. push
2. pop

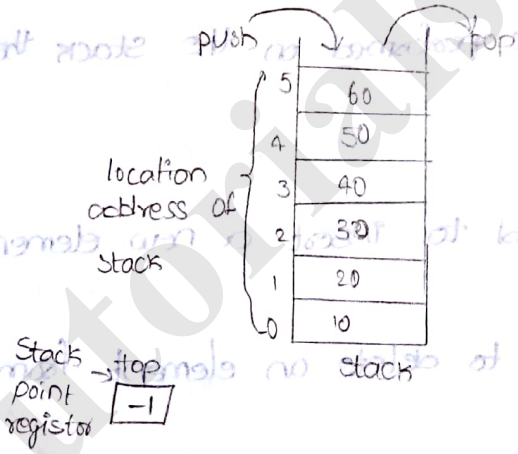
Push:- The push operation is used to insert a new element into the stack.

Pop:- The pop operation is used to delete an element from the stack.

* In general the stack uses a special small register which is know as stack pointer register which will be refered as top and whose initial values -1 → to store the address of the stack
* The stack pointer register value can be incremented (or) decremented to perform either push (or) pop operation on the stack

* while performing push operation on the stack, the stack pointer register value must be incremented and then the new element will be inserted into the stack.
* while performing pop operation on the stack, first the stack element must be deleted from the stacks and then the stack pointer register value should be decremented by 1.
* On the stack, the push and pop operations can be performed from only a single end which is known as end of the stack.
* When we try to insert an element into the stack, even though the stacks is full of elements. then a situation is occured known as stack overflowed
* When we try to delete an element from the stack, even though the stack is empty then a situation is occured know as stack under flow

Ex:- 10, 20, 30, 40, 50, 60



* The stack can be implemented using either array (or) linked lists

Ex:-
```
#include<iostream.h>
#include<conio.h>
int stack[100],n,choice,top,x,i;
void push();
void pop();
```

(global declaration; → to indicate the length of array; → we use switch operation)

```
void display();
void peek();
void main()
{
 top=-1;
 clrscr();
 cout<<"Enter the size of the stack"<<endl;
 cin>>n;
 cout<<"the stack operations are:"<<endl;
 cout<<"1.push"<<"\t"<<"2.pop"<<"\t"<<"3.display"<<"\t"<<"
                                    4.peek"<<"\t"<<"5.Exit"<<endl;
 do
  {
  cout<<"Enter your choice"<<endl;
  cin>>choice;
  switch(choice)
  {
   case 1:
    {
     push();
     break;
    }
   case 2:
     {
      pop();
      break;
     }
   case 3:
     {
      display();
      break;
     }
   case 4:
    {  peek();
       break;
    }
   case 5:
   {
   cout<<"exit"<<endl;
   break;
  }
```

```
    default:
     {
    cout<<" Enter a valid choice (1/2/3/4/5):" <<endl;
     }
    }
}while (choice!=5);
  getch( );
}

 void push( )
 {
 if (top >=n-1)
{
 cout<<" stack is overfloa"<<endl;
 }
else
  {
 cout<<"the element to be inserted into the stack is:";
  cin>>x;
  top++;
  stack [top]=x;
 }
}
 void pop( )
 {
 if (top<=-1)
  {
 cout<<" stack is underflow"<<endl;
  }
 else
  {
  cout<<"the deleted element is:"<<stack [top];
    top--;
  }
 }
 void display( )
  {
  if (top>=0)
```

```
cout<<"the elements in the stack are:"<<endl;
for(i=top; i>=0; i--)
 {
  cout<<Stack[i]<<endl;
 }
 cout<<"enter the next choice:"<<endl;
 }
else
 {
 cout<<"the stack is empty:"<<endl;
 }
}

void peek()
 {
if(top==-1)
 {
 cout<<"the stack is empty"<<endl;
 }
else
 {
 cout<<"the top element in the stack is:"<<stack[top]<<endl;
 }
}
```

Queue ADT:-

* A queue is also an abstract data type why because it contains set of elements (or) storage locations and some associated operations

* A queue is a linear data structure which stores the elements in a sequencial order

* Usually the queue follows a principle known as FIFO (first in first out) which mean that what ever the element which is inserted first into the queue will be deleted first.

* On the queue we perform two basic operations they are
  1) Enqueue
  2) Dequeue

ENQUEUE:-

* The enqueue operation is used to insert a new element into the queue

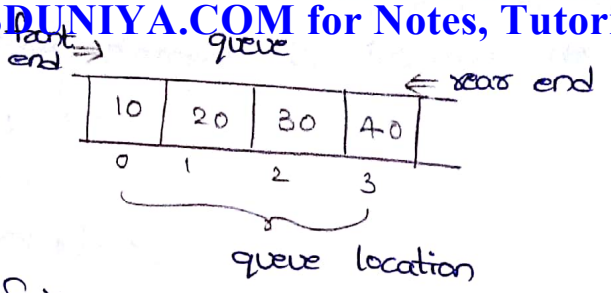
Dequeue:-

* The dequeue operation is used to delete an element from the queue

* The queue uses two small registors known as rear and front which are used to hold the address of queue location while performing enqueue and dequeue operations. The rear and front registors initial values are assigned to -1.

* In order to perform enqueue operation, first we must increment the rear registor value by 1 and then the new element will be inserted into the queue

* In order to perform dequeue operation, first we must decrement the front registor value by 1 and then the element will be deleted into the queue.



Initially

rear [-1] front [-1]

program for the queue implementation using arrays.

```
#include<iostream.h>
#include<conio.h>
int queue[100], n, rear, front, choice, x, i;
void enqueue( );
void dequeue( );
void display( );
void main( )
{
rear = -1;
front = -1;
clrscr( );
cout<<"enter the size of the queue:";
cin>>n;
cout<<" the queue operations are:"<<endl;
cout<<"1. enqueue"<<"\t"<<"2. dequeue"<<"\t"<<"3. display"<<
                "\t"<<"4. exit"<<endl;
do
{
cout<<"enter your choice:";
cin>>choice;
switch (choice)
{
case 1:
{
```

```
        enqueue();
        break;
      }
      case 2:
      {
       dequeue();
       break;
      }
      case 3:
      {
       display();
        break;
      }
      case 4:
      {
       cout<<" exit"<<endl;
       break;
      }
      default:
      {
       cout <<" enter a valid choice (1/2/3/4): "<<endl;
       break;
       }
      }
     }
     while(choice != 4);
     getch();
     }
     void enqueue()
     {
      if (rear >= n-1)
      {
      cout<<" queue is overflow"<<endl;
       }
     else
      {
```

```
cout<< the element is inserted into the queue is:";
cin>>x;
rear++;
queue [rear]=x;
}
}
void dequeue()
{
if (front==rear)
{
cout<<" the queue is underflow"<<endl;
}
else
{
front++;
cout<<" the deleted element from the queue is:"<<queue [front]<<endl;
}
}
void display()
{
if (front==rear)
{
cout<<" the queue is empty"<<endl;
}
else
{
cout<<" the list of elements stored in the queue are:";
for (i=front+1; i<=rear; i++)
{
cout<<queue[i]<<" ";
}
}
}
```

Expression:-

An expression is the combination of operands and operators which ultimately represents a value.

Types of expression:

Based on the position of the operator the expressions are classified into 3 types they are 1. Infix expression
2. pre-fix expression
3. post-fix expression

Infix expression :-

In an expression if the operator is to be based b/w the operands then such expression can be called as infix expression.

General form:

operand1 operator operand 2

Ex:- a+b

prefix expression:

In an expression if the operator is to be based b/w the first of the operands then such expression can be called as prefix expression.

GF

operator operand 1 operand 2

Eg:- +ab

postfix expression:-

In an expression if the operator is to be placed in the ending of the position of the operand then such expression can be called as postfix.

GF operand1 operand2 operator

Eg: ab+

we can convert one expression form into another expression like

Infix to prefix

Infix to postfix

prefix to postfix and vice-versa

** conversion from infix expression into postfix expression:-

Algorithm steps:-

high
/, *, %, ^
low
+ -

Step 1: Start

Step 2: Add left paranthesis "(" to the stack and right paranthess ")" to the end of the given infix expression.

Step 3: Scan (or) read all the characters one by one from left to right in the given infix expression.

Step 4: If the character is an operand then place it into postfix expression.

Step 5: If the character is left paranthesis "(" then push it into the stack.

Step 6: If the character is right paranthesis ")" then pop the operator characters from the stack and place into postfix expression until respective left paranthesis "(" is encountered in the stack.

Step 7: If the character is an operator then push it into Stack.

If the current operator priority is >= to the already existed operator in the stack the push it into stack.

Otherwise pop the existed operator from the stack and push the current operator into stack. and placed popped operation into postfix expression

Step 8: Stop

TutorialsDuniya.com

Download FREE Computer Science Notes, Programs, Projects, Books PDF for any university student of BCA, MCA, B.Sc, B.Tech CSE, M.Sc, M.Tech at https://www.tutorialsduniya.com

- Algorithms Notes
- Artificial Intelligence
- Android Programming
- C & C++ Programming
- Combinatorial Optimization
- Computer Graphics
- Computer Networks
- Computer System Architecture
- DBMS & SQL Notes
- Data Analysis & Visualization
- Data Mining
- Data Science
- Data Structures
- Deep Learning
- Digital Image Processing
- Discrete Mathematics
- Information Security
- Internet Technologies
- Java Programming
- JavaScript & jQuery
- Machine Learning
- Microprocessor
- Operating System
- Operational Research
- PHP Notes
- Python Programming
- R Programming
- Software Engineering
- System Programming
- Theory of Computation
- Unix Network Programming
- Web Design & Development

Please Share these Notes with your Friends as well

Eg: Convert the infix expression (A+B) *(C+D) into postfix expression.

Infix expression (input)	Stack	postfix expression (output)
(	(	
(	((	
A	((	A
+	((+	A
B	((+	AB
)	(	AB+
*	(*	AB+
(	(*(	AB+
& c	(*(	AB+C
+	(*(+	AB+C
D	(*(+	AB+CD
)	(*	AB+CD+
)		AB+CD+*

A+(B/C+(D%E*F)/G)*H

A-(B/C+(D%E*F)/G)*H

Infix	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
)		ABC/DEF*%G/+H*-

1, *, %, /

(,), +, -

x+(y*z/m)+s/(l/n%p)

infix	stack	postfix
	(	
x	(	x
+	(+	x
(	(+(	x
y	(+(	xy
*	(+(*	xy
z	(+(*	xyz
/	(+(/	xyz*
m	(+(/	xyzm
)	(+	xyzm/*
+	(++	xyzm/*
s	(++	xyzm/*s
/	(++/	xyzm/*s
(	(++/(	xyzm/*s
l	(++/(	xyzm/*sl
/	(++/(/	xyzm/*sl
n	(++/(/	xyzm/*sln
%	(++/(/%	xyzm/*sln
p	(++/(/%	xyzm/*slnp
)	(++/	xyzm/*slnp%/
)		xyzm/*slnp%/++

Evaluation of postfix expression:-

Algorithm steps:-

Step 1:- Start

Step 2:- Scan (or) read all the characters one by one from left to right in the given postfix expression.

Step 3:- If the characters is an operand then push it into the stack

Step 4:- If the character is an operator then pop two operands from the stack and perform the operation with that operator from left to right and also push the result back into stack.

Step 5:- Repeat Steps ③ & ④ until all the characters are scanned from the given postfix expression

Step 6:- pop the final result from the Stack as output.

Step 7:- Stop

Eg: Evalue the postfix expression 23*45*+

postfix expression	stack
	⊏
2	2 ⊏
3	2 3
*	6
4	6 4
5	6 4 5
*	6 20
+	26
	⊏
	∴ o/p :- 26

Evaluate the postfix expression 12+34+^

postfix	stack
	⊏
1	1
2	1 2
+	3
3	3 3
4	3 3 4
+	3 7
^	2187
	⊏
	∴ o/p:- 2187

Subtyping:-

The process of creating sub class (or) sub types from their subclass (or) supertypes.

By creating sub class (or) subtypes we can achieve code reusability because the subtype can accesses (or) use all the properties from its super type.

Eg:-

```
#include<iostream.h>
#include<conio.h>
class A
{
public:
int a,b
void getdata();
};
Class B:public A
{
```

```
int c,d;
public:
void add();
void sub();
void display();
};
void A::getdata()
{
cout<<"enter the a value:"<<endl;
cin>>a;
cout<<"enter b value:"<<endl;
cin>>b;
}
void B::add()
{
c=a+b;
}
void B::sub()
{
d=a-b;
}
void B::display()
{
cout<<"addition is:"<<c<<endl;
cout<<"substraction is:"<<d;
}
int main()
{
B obj;
disco();
obj.getdata();
obj.add();
obj.sub();
obj.display();
return 0;
}
```

UNIT-3

Linked Lists

* A linked list is a collection of nodes (or) collection of inter-connection nodes

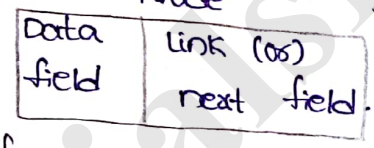
* The linked list is a linear data structure in which the nodes are connected (created) sequencially by the establishing connection from one node to another node.

* Basically the linked lists are classified into 3 types they are i) single linked list
  ii) Doubly linked list
  iii) circular linked list.

Single linked list:—

In the single linked list each node contains two fields they are 1) Data fields 2) link (or) next field

Node structure:— in the single linked list:—

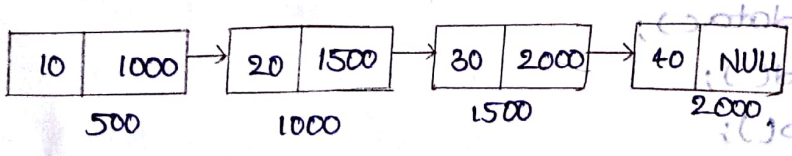


Data field:- The data field can be used to store the data (or) an element.

Link (or) next field:- The link field can be used to store the address of next node in the single linked list

* In the single linked linked list the link field of last node is NULL.

Eg:- 10, 20, 30, 40



Operations on single linked lists:

On the single linked lists we perform four basic operations they are 1) create
2) Insert
3) Delete
4) Display

Create:- The create operation is used to create a node in order to form a single linked list.

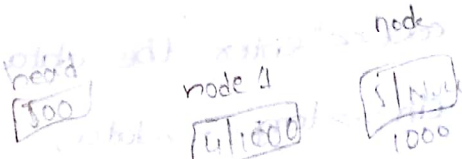
Insert:- This operation is used to insert a node either front (or) middle (or) last to the existing node(s) in the single linked list.

Delete:- In this operation can be used to delete a specific node in the single linked list

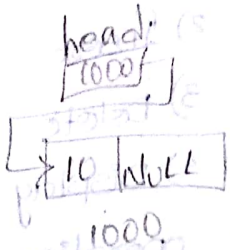
Display:- this operation can be used to display the list of elements in the single linked list.

Ex:- Program to implement single linked list.

```
#include<iostream.h>
#include<conio.h>
#include<process.h>
  class slink
   {
   struct node  -> structure name
   {
    int data;
    node *next;
    } *head;
 public:
    slink()  -> default constructor
    {
    head=NULL;
    }
```

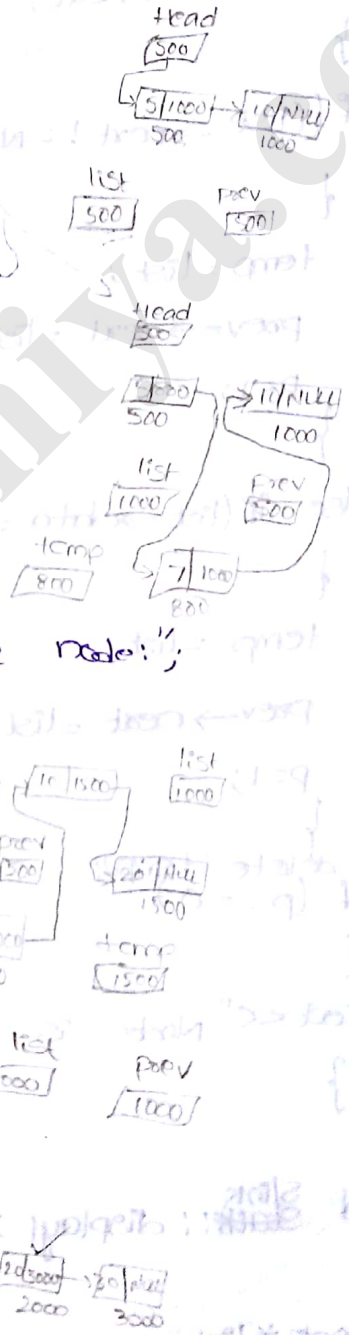


```
void create ();
void insert ();
void delnode ();
void display ();
};
```
member functions

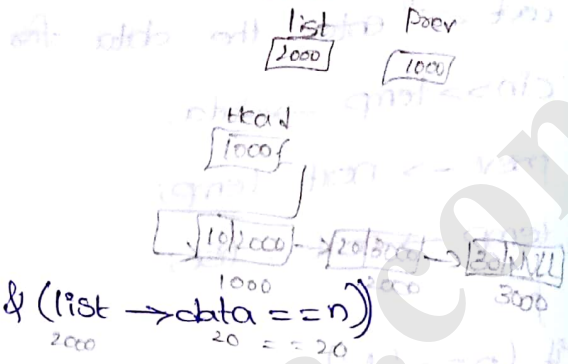


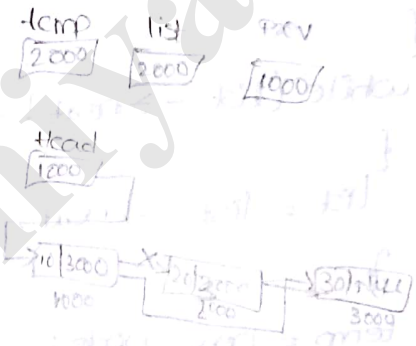

```
void slink::create()
{
 head = new node;
 cout<<"Enter the data for the node:";
 cin>>head->data;
 head->next =NULL;
}
void slink::insert()
 {
 int n;
 char p;
 node *list, *prev, *temp;
 list =head;
 cout<<"Insert the node at front/middle/last? f/m/l:";
 cin>>p;
 if (p=='f')
 {
 temp = new node;
 cout<<"Enter the data for the node:";
 cin>>temp->data;
 temp->next =head;
 head =temp;
 }
 if (p=='m')
 {
 cout<<"Enter the data for the node before to insert ne
                                    node at middle";
 cin>>n;
```

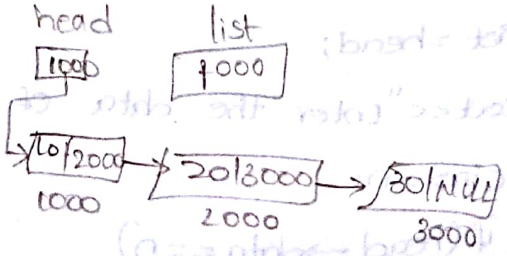
```
while (list -> data != n)
 {
  prev = list;
  list = list -> next;
 }
temp = new node;
cout << "Enter the data for the node";
cin >> temp -> data;
prev -> next = temp;
temp -> next = list;
}
if (p == 'l')
{
 while (list -> next != NULL)
 {
  list = list -> next;
 }
 temp = new node;
 cout << "Enter the data for the node:";
 cin >> temp -> data;
  list -> next = temp;
  temp -> next = NULL;
 }
}
Void slink :: delnode( )
{
int n, p = 0;
node *list, *prev, *temp;
list = head;
cout << "Enter the data of the node to be deleted:";
cin >> n;
if (head -> data == n)
{
```



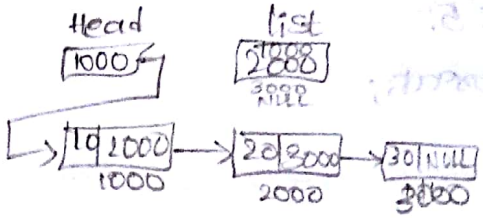
```
head = head -> next;
p=1;
}
else
{
while ((list -> next != NULL) && (list -> data != n))
{
prev = list;
list = list -> next;
}
if ((list -> next != NULL) && (list -> data == n))
{
temp = list;
prev -> next = list -> next;
p=1;
}
else if (list -> data == n)
{
temp = list;
prev -> next = list -> next;
p=1;
}
delete (temp);
if (p==0)
{
cout<<" Node is not present :";
}
}
void slink::display()
{
node *list;
list = head;
if (list == NULL)
{
cout<<" list is empty:";
```







```
}
else
 {
 while (list != NULL)
 {
 cout<<list->data<<"<=>";
 list = list->next;
 }
 cout<<"NULL";
 } //else over
} //display over
void mainc )
{
 Slink s;
 int option;
 clrscr );
 cout<<"1.create()"<<endl;
 cout<<"2.insert()"<<endl;
 cout<<"3.delnode()"<<endl;
 cout<<"4.display()"<<endl;
 cout<<"5.exit"<<endl;
 do
  {
  cout<<"\n Enter your choice:";
  cin>>option;
  switch(option)
   {
case 1:
  s.create();
  break;
case 2:
  s.insert();
  break;
case 3:
  s.delnode();
  break;
```



O/P

10<=>20<=>30<=>NULL

```
        case 4:
            s.display();
            break;
        case 5:
            break;
        }
    }
    while (option!=5);
    getch();
}
```

1!=5
2!=5
3!=5
4!=5
5!=5 → false

Double linked list:-

In the doubly linked list each node contains 3 fields they are 1. left link field
2. data field
3. Right link field

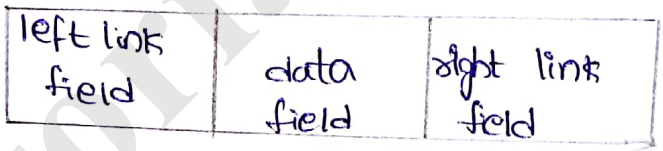
1. left link field:- The left link field is used to store the address of previous node

2. Data field:- The data field is used to store the data (or) an element

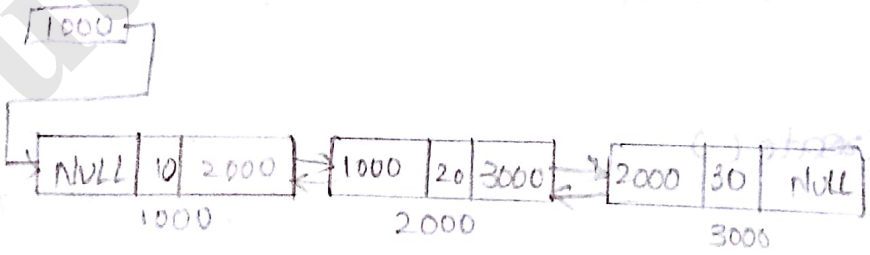
3. Right link field:- The right link field is used to store the address of next node.

* In the doubly linked list the left link field of the first node is NULL node and also the right link field of the last node is NULL.

Node structure of the doubly linked list:-



Eg:- head



TutorialsDuniya.com

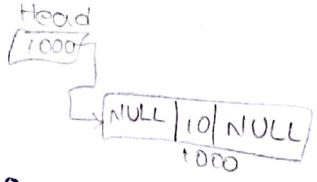
Download FREE Computer Science Notes, Programs, Projects, Books PDF for any university student of BCA, MCA, B.Sc, B.Tech CSE, M.Sc, M.Tech at https://www.tutorialsduniya.com

- Algorithms Notes
- Artificial Intelligence
- Android Programming
- C & C++ Programming
- Combinatorial Optimization
- Computer Graphics
- Computer Networks
- Computer System Architecture
- DBMS & SQL Notes
- Data Analysis & Visualization
- Data Mining
- Data Science
- Data Structures
- Deep Learning
- Digital Image Processing
- Discrete Mathematics
- Information Security
- Internet Technologies
- Java Programming
- JavaScript & jQuery
- Machine Learning
- Microprocessor
- Operating System
- Operational Research
- PHP Notes
- Python Programming
- R Programming
- Software Engineering
- System Programming
- Theory of Computation
- Unix Network Programming
- Web Design & Development

Please Share these Notes with your Friends as well

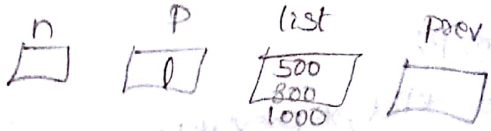
Program to implement doubly linked list:

```
#include<iostream.h>
#include<conio.h>
#include<process.h>
class Dlink
{
struct node
{
node *llink;
int data;
node *rlink;
} *head;
public:
Dlink()
{
head=NULL;
}
void create();
void insert();
void delnode();
void display();
};
void Dlink::create()
{
head=new node;
cout<<"Enter the data for the node:";
cin>>head->data;
head->llink=NULL;
head->rlink=NULL;
}
```

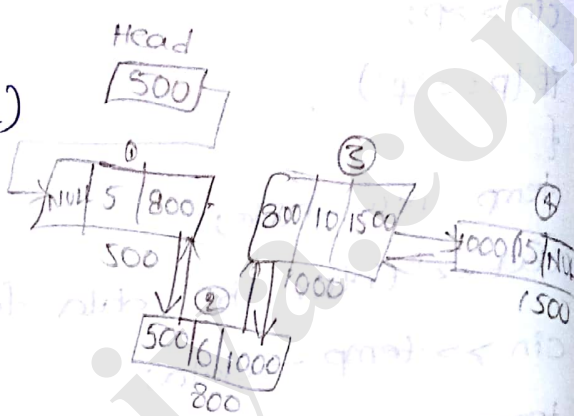


```
void Dlink::insert()
{
 int n;
 char p;
 node *list,*prev,*temp;
 list = head;
 cout<<"Insert the node at front/middle/last? f/m/l:";
 cin>>p;
 if (p=='f')
 {
 temp = new node;
 cout<<"Enter the data for the node:";
 cin>>temp->data;
 temp->rlink = head;
 head->llink = temp;
 temp->llink = NULL;
 head = temp;
 }
 if (p=='m')
 {
 cout<<"Enter the data of a node, before to insert new
                                    node. as middle:";
 cin>>n;
 while (list->data!=n)
 {
 prev = list;
 list = list->rlink;
 }
 temp = new node;
 cout<<"Enter the data for the node:";
 cin>>temp->data;
```

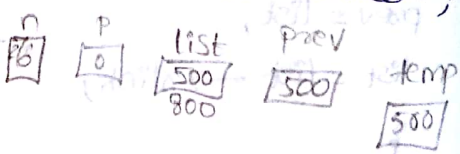
```
prev -> rlink = temp;
temp -> llink = prev;
temp -> rlink = list;
list -> llink = temp;
}
if (p == 'l')
{
while (list -> rlink != NULL)
  {
  list = list -> rlink;
  }
  temp = new node;
  cout<<" Enter the data for the node: ";
  cin >> temp -> data;
  list -> rlink = temp;
  temp -> llink = list;
  temp -> rlink = NULL;
  }
void Dlink::delnode()
{
  int n, p=0;
node *list, *prev, *temp;
  list = head;
  cout<<" Enter the data of a node to be deleted: ";
  cin >> n;
  if (head -> data == n)
  {
  temp = head;
  head = head -> rlink;
  head -> llink = NULL;
  p=1;
delete (temp);
```

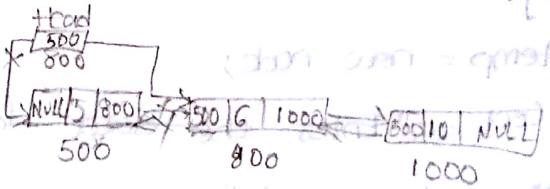


500 != NULL
800 != NULL
1000 != NULL
NULL != NULL



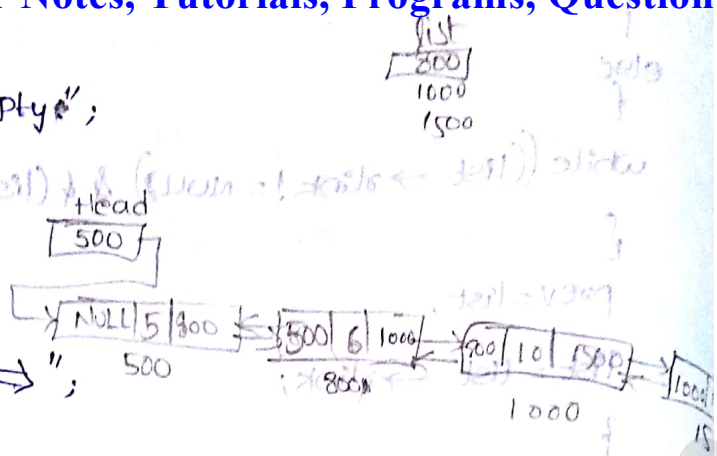
n = 6
5 == 5
5 == 6





```
    }
    else
    {
    while((list->rlink!=NULL) && (list->data!=n))
    {
    prev=list;
    list=list->rlink;
    }
    if((list->rlink!=NULL) && (list->data==n))
    {
    temp=list;
    prev->rlink=list->rlink;
    list->rlink->llink=prev;
    p=1;
    delete(temp);
    }
    else if(list->data==n)
    {
    temp=list;
    prev->rlink=list->rlink;
    p=1;
    delete(temp);
    }
    if(p==0)
    {
    cout<<"Node is not present:";
    }
    }
    void Dlink::display()
    {
    node *list;
    list=head;
    if(list==NULL)
```

```
{
cout << "list is empty";
}
else
{
cout << "NULL" << " <=> ";
while (list != NULL)
{
cout << "list -> data << " <=> ";
list = list -> rlink;
}
cout << "NULL";
}
}
void main( )
{
Dlink d;
int option;
clrscr( );
cout << " 1. create( )" << endl;
cout << " 2. insert( )" << endl;
cout << " 3. delnode( )" << endl;
cout << " 4. display( )" << endl;
cout << " 5. exit" << endl;
do
{
cout << "\n enter your choice:";
cin >> option;
switch (option)
{
case 1:
d. create( );
break;
case 2:
d. insert( );
```



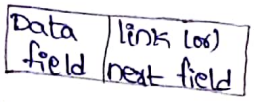
```
        break;
case 3:
        d.delnode();
        break;
case 4:
        d.display();
        break;
case 5:
        break;
    }
  }
while (option != 5);
 getch();
}
```

Circular linked list:-

The circular linked list has been classified into two types they are 1. Circular single linked list
2. Circular doubly linked list.

1. **Circular single linked list:-** In this each node contains two fields they are. 1. Data field
2. link (or) next field.

Node structure:-

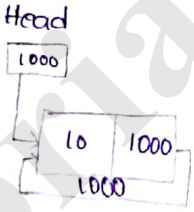


Data field:- The Data field can be used to store the data (or) an element

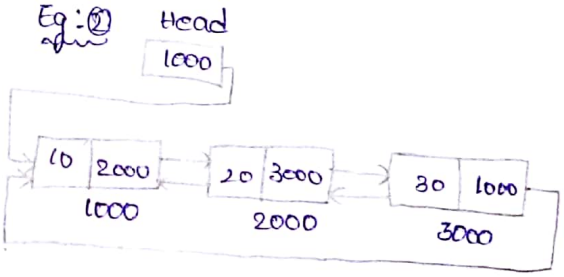
link (or) next field:- the link field can be used to store the address of next node.

* In the circular single linked list the last node link field holds the address of first node in order to make the linked list as circular

Eg:-①



Eg:-②



2. **Circular doubly linked list:-** In this each node contains 3 fields they are left link field, Data field, right link field.

Node structure:-

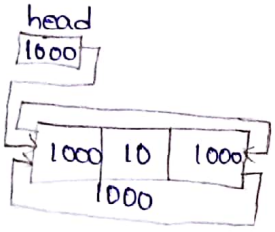
Left link field:- left link field holds the address of previous node.

Data field:- Data field can be used to store the data of an element.

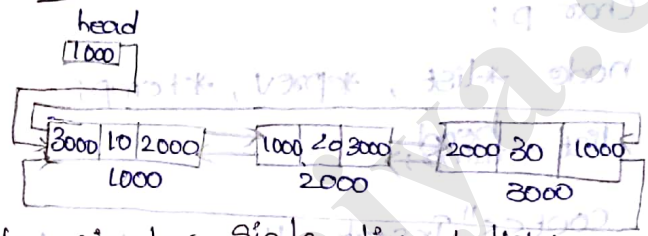
Right link field:- Right link field holds the address of next node

* In the circular doubly linked list the last node right link field stores the address of first node and the first node left link feld stores the address of last node, In order to make the linked list has circular.

Eg:- ①



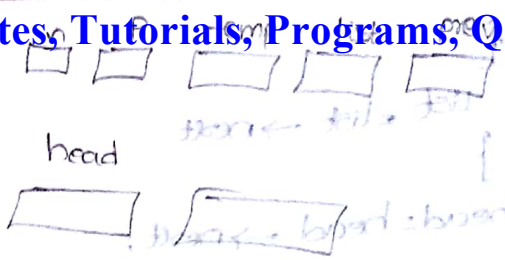
Eg:②



Program to illustrate circular single linked list:-

```
#include <iostream.h>
# include <conio.h>
# include <process.h>
  class clink
   {
   struct node
   {
    int data;
   node *next;
   }*head;
  public:
    clink( )
      {
       head = NULL;
      }
    void create( );
   void insert( );
   void delnode( );
   void display( );
};
```

```
void CSlink::create()
{
head = new node;
cout<<"Enter the data for the node:";
cin>>head->data;
head->next = head;
}
void CSlink::insert()
{
 int n;
 char p;
 node *list, *prev, *temp;
 list = head;
 cout<<"Insert the node at front/middle/last ? f/m/l:";
 cin>>p;
if (p=='f')
{
 temp = new node;
cout<<"Enter the data for the node:";
cin>>temp->data;
temp->next = head;
head->next = temp;
head = temp;
}
if (p=='m')
{
cout<<"Enter the data for the node, before to insert
                new node as middle:";
cin>>n;
while(list->data!=n)
{
```



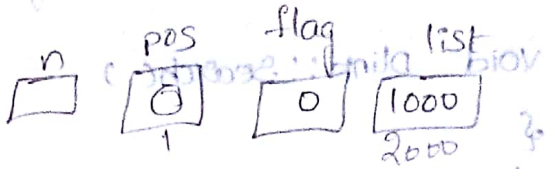

```
prev = list;
list = list -> next;
}
temp = new node;
cout<<"Enter the data for the node:";
cin >> temp -> data;
prev -> next = temp;
temp -> next = list;
}
if (p == '1')
{
while (list -> next != head)
{
list = list -> next;
}
temp = new node;
cout <<"Enter the data for the node:";
cin >> temp -> data;
list -> next = temp;
temp -> next = head;
}
}
Void cslinks::delnode()
{
int n, p = 0;
node *list, *prev, *temp;
list = head;
cout<<"Enter the data of a node to be deleted:";
cin>> n;
if (head -> data == n)
{
temp = head;
while (list -> next != head)
```

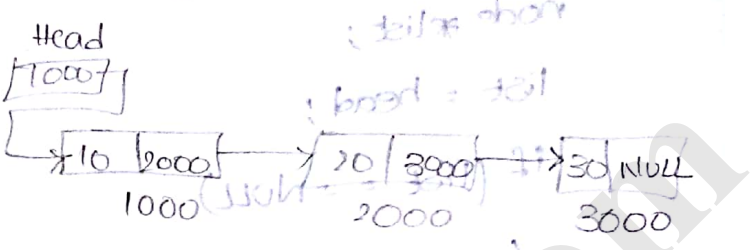
```
{
 list =list ->next
}
head=head ->next;
list ->next =head;
p=1;
delete (temp);
}
else
{
while ((list ->next != head) && (list ->data != n))
 {
 prev=list;
 list = list->next;
 }
if ((list ->next != head) && (list->data== =n))
 {
 temp =list;
 prev->next=list ->next;
 p=1;
 delete (temp);
 }
else if (list ->data==n)
{
 temp =list;
 prev ->next =list ->next;
 p=1;
 delete (temp);
}
} //else
if ( p==0)
{
```

```
    cout<<"node is not present.";
  }
}

Void CSlink::display( )
 {
  node *list;
   list =head;
  if (list = =NULL)
    {
    cout<<" list is empty:";
    }
  else
    {
     while(list ->next! =head)
      {
      cout<<list ->data <<"<=>";
      list = list ->next;
      }
     cout<<list->data;

Void main( )
 {
  CSlink cs;
  int option;
  clrscr( );
  cout<<"1. create ( )"<<endl;
  cout<<" 2. insert ( )"<<endl;
  cout<<" 3. delnode ( )"<<endl;
  cout<<" 4. display ( )"<<endl;
  cout<<" 5. exit "<<endl;
do
  {
  cout<<" \n Enter your choice:";
  cin>>option;
  switch (option)
```

```
{
case 1:
    cs.create();
    break;
case 2:
    cs.insert();
    break;
case 3:
    cs.delnode();
    break;
case 4:
    cs.display();
    break;
case 5:
    break;
    }
}
while (option!=5);
getch();
}
```

```
void Slink::search()
{
int n, pos=0, flag=0;
node *list;
list=head;
if(list==NULL)
  {
  cout<<"list is empty";
  }
else
  {
  cout<<"Enter the data of a node to be searched";
  cin>>n;
  while(list!=NULL)
  {
  pos++;
  if(list->data==n)
   {
   cout<<list->data<<"is found at:"<<pos<<endl;
   flag=1;
  }
  list=list->next;
 }//while
}//else
if(flag==0)
 {
 cout<<"Node is not present with the given element:";
 }
}//search()
```





o/p: 10 is found at: 1
30 is found at: 3
20 is found at: 2

TutorialsDuniya.com

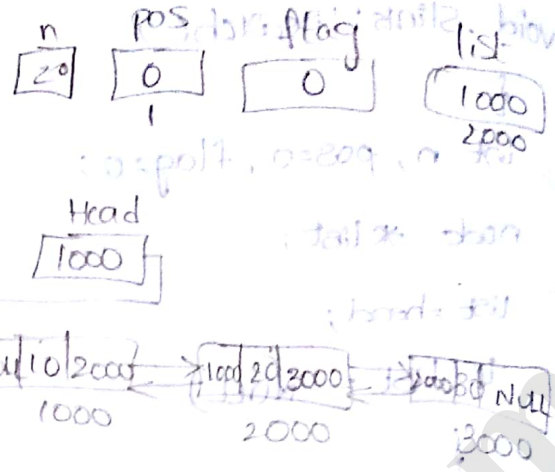
Download FREE Computer Science Notes, Programs, Projects, Books PDF for any university student of BCA, MCA, B.Sc, B.Tech CSE, M.Sc, M.Tech at https://www.tutorialsduniya.com

- Algorithms Notes
- Artificial Intelligence
- Android Programming
- C & C++ Programming
- Combinatorial Optimization
- Computer Graphics
- Computer Networks
- Computer System Architecture
- DBMS & SQL Notes
- Data Analysis & Visualization
- Data Mining
- Data Science
- Data Structures
- Deep Learning
- Digital Image Processing
- Discrete Mathematics
- Information Security
- Internet Technologies
- Java Programming
- JavaScript & jQuery
- Machine Learning
- Microprocessor
- Operating System
- Operational Research
- PHP Notes
- Python Programming
- R Programming
- Software Engineering
- System Programming
- Theory of Computation
- Unix Network Programming
- Web Design & Development

Please Share these Notes with your Friends as well

search() operation for doubly linked list:-

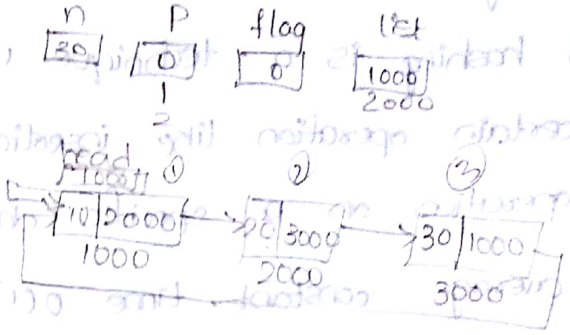
```
void Dlink::search()
{
 int n, pos=0, flag=0
 node *list;
 list = head;
 if (list == NULL)
 {
 cout<<"list is empty:";
 }
else
 {
 cout<<"enter the data of a node to be searched:
 cin>>n;
 while (list != NULL)
 {
 pos++;
 if (list->data == n)
 {
 cout<<list->data<<"is found at:"<<pos<<endl;
 flag=1;
 }
 list = list->rlink;
 }
 }
 if (flag==0)
 {
 cout<<"Node is not present with the given element:";
 }
}
```



dp: 20 is found at: 2

search() operation for circular linked list:-

```
Void CSlink::Search()
{
int n, p=0, flag=0;
node *list;
 list =head;
if (list == NULL)
{
cout<<" list is empty:";
}
else
 {
cout<<" Enter the data of a node to be searched:";
cin>>n;
while (list->next!=head)
 {
 pos++;
 if (list->data==n)
 {
cout<<list->data<<" is found at node position:"<<
 pos<<endl;
 flag=1;
 }
 list=list->next;
 } //while
if (list->data==n)
{
 pos++;
cout<<list->data<<" is found at node position:"<<pos
 <<endl;
 flag=1;
 }
} //else
if (flag==0)
{
cout<<" Node is not present with the given element:";
}
}
```



Hashing:-

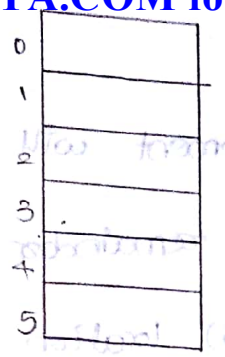
* A hashing is a technique which is used to perform certain operation like insertion, deletion and search operation on a specific data structure with an average constant time O(1) [order of 1].
* Using the hashing technique the search operation can be performed very efficiently when compare with some of the technique like linear, binary search and fibanacci search etc.
* The hashing technique can be used usually to implement a special data structure is known as dictionary. A dictionary is set of elements (or) contain.
* The hashing technique contains are uses three key components in order to perform insertion, deletion are search on a data structure. They are.
  1. Hash table
  2. Hash function
  3. collision

Hash table:-

* A hash table is a data structure which contains a fixed length array with set of index values to store (or) search (or) retrie an element into (or) from the hash table.
* The index of the hash table starts from 0 to N-1 where 'N' is the size of the hash table.

hash table (H)

0 to N-1



* In the hash table each key element is mapped (or) associated with the corresponding index value of the hash table in order to perform the operation insertion deletion and search on the hash table.

Hash function:-

* the hash function is a function which can be used to map the key elements with the corresponding index values of the hash-table to perform the operation (insertion, deletion and search)

* the value which is generated by the hash function can be called hash code (or) hash address (or) index of the hash table.

- the hash function is broadly classified into 4 types based

1. Division method.
2. Mid-square method
3. folding method
4. Multiplication method

- the hash function can be denoted by 'H'.

Division Method:-

* In this method, each key element will be modulo divided by hash table size and the remainder value can be treated as hash address (or) location address of the hash table. where we can insert (or) delete (or) search the particular key element into (or) from hash table
* The general form of this Method is

$$H(k) = k \bmod m$$

where

K = key

m = hash table size.

Eg:- m = 10

keys = 20, 28, 32, 44

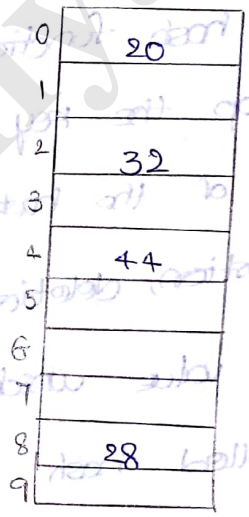
H(20) = 20%10 = 0

H(28) = 28%10 = 8

H(32) = 32%10 = 2

H(44) = 44/10 = 4

Hash table



Mid-Square Method:-

* In this method each key element is squared and middile bits are extracted from the squared number depending up on the size of the hash table and also the middle bits will be modulo divided by hash table size and the remainder value can be considered as hash address (or) location address of the hash table.

m = 10

keys = 6, 72, 15, 38

$H(K) = K^2 = 6^2$

$(72)^2 = 5184 \% 10 = 18 \% 10 = 8$

$(15)^2 = 225 \% 10 = 2 \% 10 = 2$

$(38)^2 = 1444 \% 10 = 44 \% 10$
$= 4$

0	
1	
2	15
3	
4	38
5	
6	36
7	
8	72
9	

Folding Method:- This method is divided into two types they are 1. fold shifting method
2. fold boundary method

Fold shifting method:-

*In this Method the key element is divided into equal partitions based on the hash table size and all the partitions will be added and the sum will be considered as the location address of the hash table if any carry is generated while performing addition operation between all the partitions, the carry will be ignored.

Eg:- m=10

k = 123456

= 1+2+3+4+5+6

= 21 → location address of hash table

carry is ignored

Fold boundary Method:-

*The procedure of this method is exactly similar to the fold shifting method, exp except that the first and last partitions will be reversed and also all the partitions will be added and then the sum will be considered as hash address if any carry is generated

that will be ignored.

Eg:- m = 100

K = 123456

= 12 34 56

= 21 + 34 + 65

= 120 → sum

carry is ignored.

$K \times A$ $m \neq 0$

$A = \frac{\sqrt{5}+1}{2}$

$= 0.618033$

20×0.618033

$= 12.36066$

0.36066 fraction

Multiplication Method:-

In this Method the key value is multiplied by a constant value and from the result the fraction part will be extracted and that will be multiplied by Size of the hash table and from the final result the integer part will be considered as the hash address

$m = 10$, $K = 20$

$K \times A$

$A = \frac{\sqrt{5}-1}{2}$

$A = 0.618033$

$KA = 20 \times 0.618033$

$= 12.36066$

$m[KA] = 10 \times 0.36066$ → fractional part

$= 3.6066$

Integer part

$= 3$

The general form of this method is $H(K) = m[KA]$

This method will work efficiently by taking the constant value $A = \frac{\sqrt{5}-1}{2} = 0.618033$

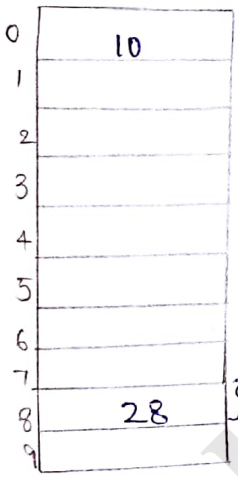
Collision:-

The collision is a problem (or) situation which will be occured when more than one element need to be stored into the same location address of the hash table.

The collision problem can be avoided by using two types of techniques there are 1. open hashing
2. closed hashing

open hashing:- this technique can also be called as chaining (or) seperate chaining this technique can be used to resolve the collision problem by using linked lists

Eg:- $m = 10$
$k = 10, 20, 28, 38$



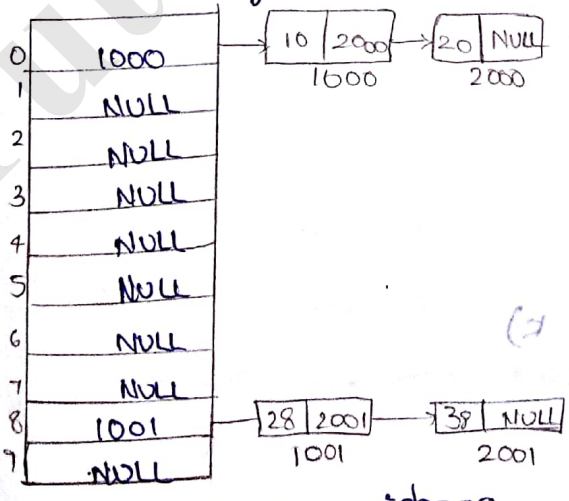
} collision is occured

$H(10) = 10\%10 = 0$
$H(20) = 20\%10 = 0$ } collision
$H(28) = 28\%10 = 8$
$H(38) = 38\%10 = 8$ } collision

} collision occured

(a) collision

$m = 10$, keys = 10, 20, 28, 38



$H(10) = 10\%10 = 0$
$H(20) = 20\%10 = 0$
$H(28) = 28\%10 = 8$
$H(38) = 38\%10 = 8$

(b) collision avoidance

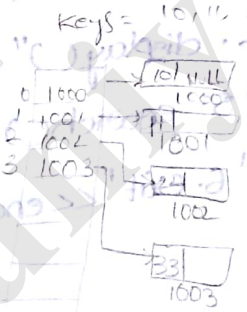
Program to implement chaining

```
#include <iostream.h>
#include <conio.h>
#define max
class chain
{
struct node
{
 int data;
 node *next;
}*hashtable[max];
public:
 void initializehashtable();
 int hashfun(int);
 void insert();
 void delnode();
 void display();
 void search();
};
void chain::initializehashtable()
{
 int i;
 for(i=0;i<max;i++)
 {
  hashtable[i]=NULL;
 }
}
int chain::hashfun(int k)
{
return(k%max);
}
void chain::insert()
{
```

```
int pos;
node *temp, *list;
temp = new node;
cout << " Enter the data for the node: ";
cin >> temp -> data;
temp -> next = NULL;
pos = hashfun (temp -> data);
if (hashtable [pos] == NULL)
{
hashtable [pos] = temp;
}
else
 {
 list = hashtable [pos];
 while (list -> next != NULL)
 {
  list = list -> next;
  }
 if (list -> next == NULL)
   {
    list -> next = temp;
   }
 }
}

void chain::delnode( )
   {
 int pos, n, p = 0;
 node *temp, *list;
 cout << " Enter the element to be deleted: ";
 cin >> n;
 pos = hashfun (n);
 temp = hashtable [pos];
 if (temp -> data == n)
```

```
{
list = temp;
temp = temp -> next;
hashtable [pos] = temp;
p = 1;
delete(list);
}
else
{
while((temp -> next != NULL) && (temp -> data != n))
{
list = temp;
temp = temp -> next;
}
if((temp -> next != NULL) && (temp -> data == n))
{
list -> next = temp -> next;
p=1;
delete (temp);
}
else if(temp -> data == n)
{
list -> next = temp -> next;
p=1;
delete (temp);
}
}
if (p == 0)
{
cout<<" n <<" is not found at position: "<<pos<<endl
}
}
```

```
void chain::display()
{
int i;
node *list;
for(i=0; i<max; i++)
 {
 list = hashtable[i];
 if (list == NULL)
   {
   cout<<" list is empty:";
   }
   while(list != NULL)
    {
    cout<<list->data <<" ";
    list=list->next;
    }
   }
  }

void chain::search()
{
 int pos, k;
 node *list;
 cout<<"enter the element to be searched:";
 cin>>k;
 pos=hashfun(k);
 list=hashtable[pos];
 while((list!=NULL)&&(list->data!=k))
   {
   list = list->next;
   }
 if((list!=NULL)&&(list->data==k))
 {
 cout<<list->data<<" is found at position:"<<pos
                                          <<endl;
 }
 else
 {
```



```
        cout<<"is not found at position"<<pos<<endl;
    }
}

void main()
{
    chain c;
    int option;
    clrscr();
    cout<<"1. Initializehashtable()"<<endl;
    cout<<"2. insert()"<<endl;
    cout<<"3. delnode()"<<endl;
    cout<<"4. display()"<<endl;
    cout<<"5. search()"<<endl;
    cout<<"6. Exit"<<endl;
    do
    {
        cout<<"\nEnter your choice:";
        cin>>option;
        switch(option)
        {
        case 1:
            c.initializehashtable();
            break;
        case 2:
            c.insert();
            break;
        case 3:
            c.delnode();
            break;
        case 4:
            c.display();
            break;
```

```
case 5:
        c.Search();
        break;
case 6:
        break;
      }
    }
  while(option!=6);
  getch();
}
```

TutorialsDuniya.com

Download FREE Computer Science Notes, Programs, Projects, Books PDF for any university student of BCA, MCA, B.Sc, B.Tech CSE, M.Sc, M.Tech at https://www.tutorialsduniya.com

- Algorithms Notes
- Artificial Intelligence
- Android Programming
- C & C++ Programming
- Combinatorial Optimization
- Computer Graphics
- Computer Networks
- Computer System Architecture
- DBMS & SQL Notes
- Data Analysis & Visualization
- Data Mining
- Data Science
- Data Structures
- Deep Learning
- Digital Image Processing
- Discrete Mathematics
- Information Security
- Internet Technologies
- Java Programming
- JavaScript & jQuery
- Machine Learning
- Microprocessor
- Operating System
- Operational Research
- PHP Notes
- Python Programming
- R Programming
- Software Engineering
- System Programming
- Theory of Computation
- Unix Network Programming
- Web Design & Development

Please Share these Notes with your Friends as well

UNIT - 4

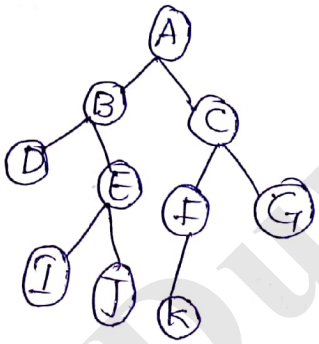
TREES

* A tree is a non-linear data structure in which the elements are organised in the form of hierarchical structure.

* A tree is a collection of elements (or) nodes and some associated operations.

* Hence the tree can also be called as an abstract data type.

Eg:-



Terminology of tree:-

1. Root:- In the tree, the topest node can be called as Root (or) Root node.

* Every tree must contain only one root node.

2. Edge:- In the tree, a connecting link between any two nodes is known as an edge, If the tree contains 'n' no.of nodes then the tree contains n-1 no.of edges,

3. Parent:- In the tree, If a node contains child (or) children then such node can be called as a parent (or) parent node.

4. child:- In the tree, If a node is having parent node then such node can be called as a child

5. Leaf :- In the tree, If a node does not have child (or) children then such node can be called as leaf (or) leaf node.

* leaf nodes can also be called as external nodes (or) terminal nodes.

6. Internal nodes :- In the tree, except the leaf nodes the remaining nodes can be called as Internal nodes

* the Internal nodes can also be called as non-terminal nodes.

7. Degree :- In the tree, the total no.of childs of a particular node is said to be degree of that particular node.

* Among all the nodes the node which is having highest degree can be treated as the degree of the entire tree.

8. Height :- In the tree, the total no.of edges from any leaf node to the specific node is said to be the height of that particular node

* the height of the entire tree is the height of the root node.

* The height of the leaf nodes is 'zero'

9. Depth :- In the tree, the total no.of edges from the root node to any particular node is said to be depth of that node.

* the heighest depth of a node can be treated as the entire depth of the tree.

* the depth of the root node is zero.

10. Level:- In the tree, usually the root node is represented at level 'zero' and its children are represented at level 'one' and their children will be represented at level 'two'. and so...

11. Path:- The sequence of consecutive edges from one node to another node is known as a path

* The length of the path is equal to the total no. of nodes existed along that particular path.

12. Sub tree:- In the tree, A sub tree can be formed by using child (or) children of a particular node

13. Anchestors & Descendants:- In the tree, all the nodes from root node to specific node are said to be anchestors for that node and all the nodes from the specific node to any leaf node are said to be descendants for that node.

14. Siblings:- In the tree, the nodes which are having the same parent node are said to be sibling

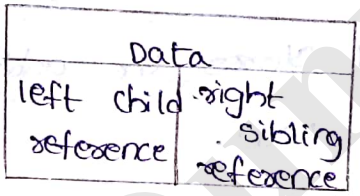
General Tree representation:-

Usually the general tress are represented using two ways they are 1. left child and 2) right sibling representation

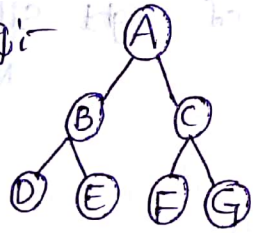
left child and right sibling representation:

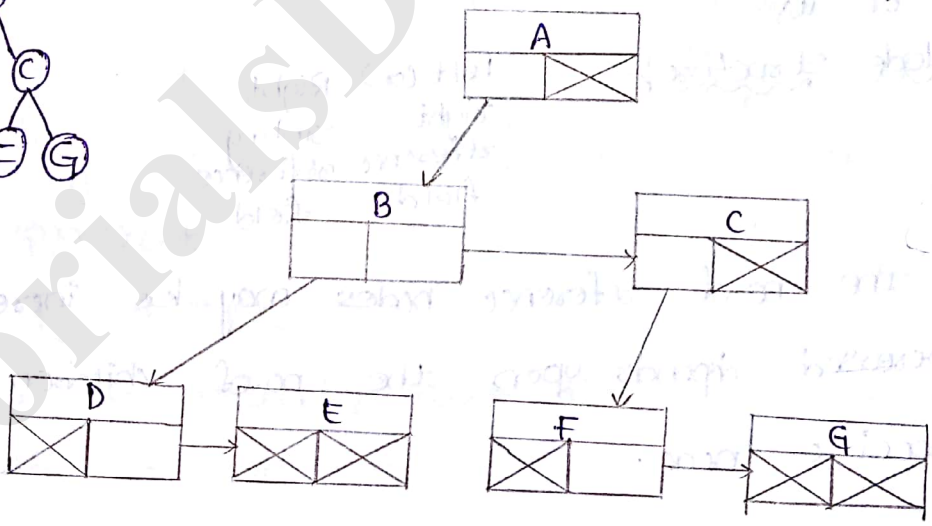
* In this representation, we use a special node which contains three fields such as data, left child reference and right sibling reference fields.
* the data field stores the actual element (or) value.
* The left child reference field stores the address of left child of the specific node
* The right sibling reference field stores the address of right sibling of the particular node.

Node structure:-



Eg:-





'X' indicates NULL.

List representation:-

In this representation we use two special nodes such as data node and reference node.

Data Node:- The data node contains 2 fields.

* The first field stores the actual element (or) value.
the second field stores the address of either left (or) right child.

Node structure:-

Data field	left (or) right child field

Reference Node:-
The reference node contains two fields

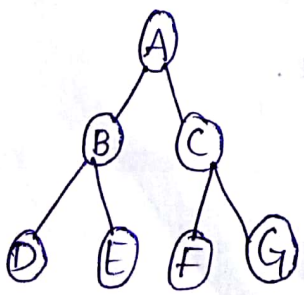
* The first field stores the address of either left (or) right child of a specific node.
* The second field stores the address of right sibling of its Node

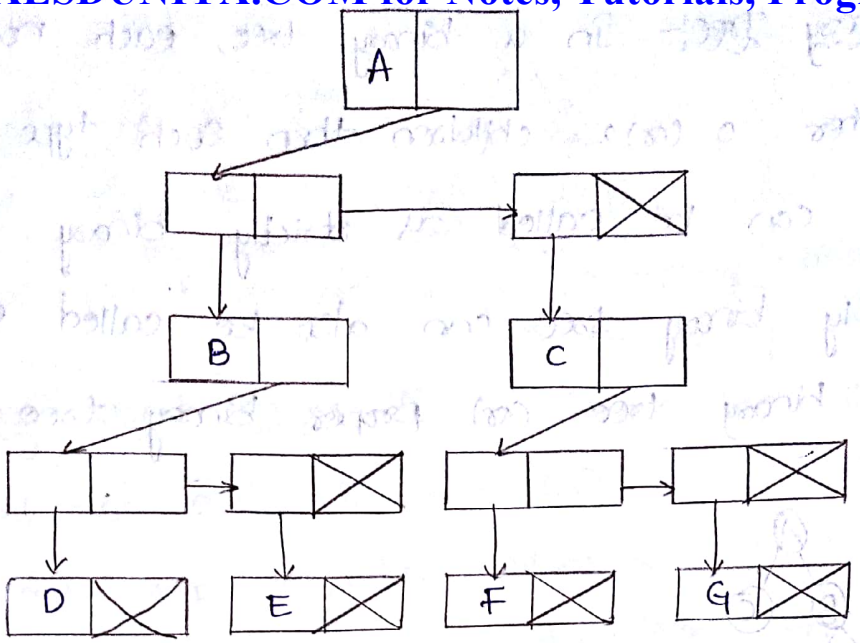
Node structure:-

left (or) right reference field	Right sibling reference field

* The no.of reference nodes may be increased (or) decreased depends upon the no.of children of a specific node.

Eg:-

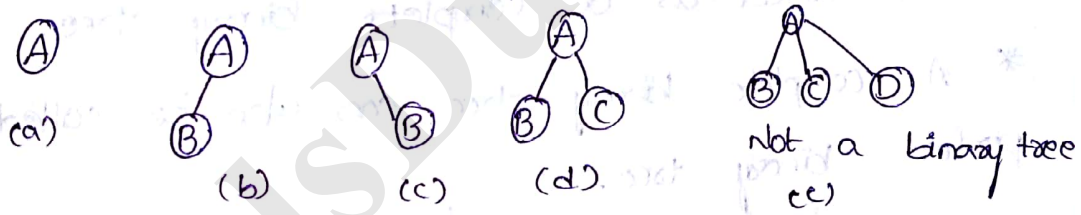




Binary Trees:-

* In a tree, if every node contains either '0' (or) '1' (or) '2' children but not more than '2' children then such type of binary tree can be called as a binary tree

Eg:-



* the binary trees are widely used to represent algebic expressions

Types of binary Trees:-

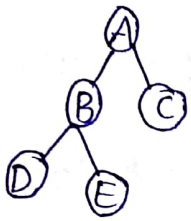
Based on the frequently usage the binary trees are classified into 5 types. they are.

1. strictly binary tree
2. complete binary tree
3. Extended binary tree
4. Binary Search tree
5. threaded binary tree.

Strictly binary Tree:- In a binary tree, each node contains either 0 (or) 2 children then such type of binary tree can be called as strictly binary tree.

* The strictly binary tree can also be called as either full binary tree (or) proper binary tree (or) 2-tree

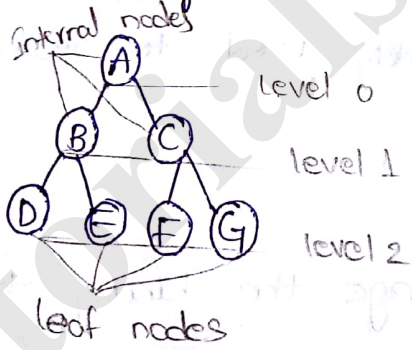
Eg:-



Complete Binary Tree:- In a binary tree, if every node contains exactly 2 children and all the leaf nodes are at same level then such type of binary tree can be called as a complete binary tree.

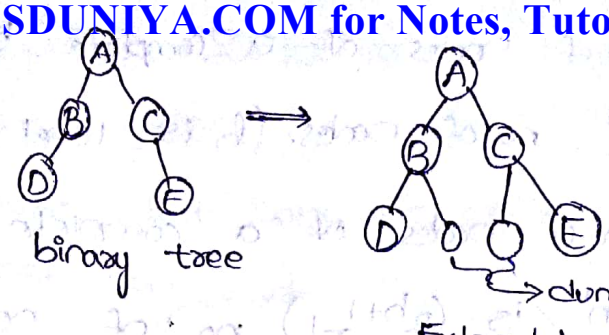
* A complete binary tree can also be called as a perfect binary tree.

Eg:-



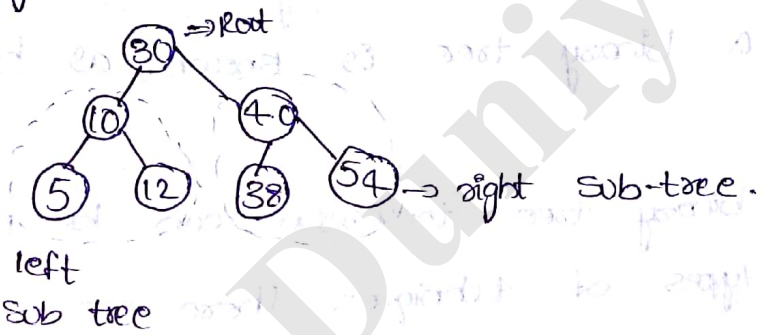
Extended binary tree:- If a binary tree is converted into strictly binary tree by adding some dummy nodes to the existing nodes then the converted strictly binary tree can be called as an extended binary tree.

Eg:-



* **Binary Search Tree:-** In a binary tree, the values in of all the nodes in the left sub-tree are less than the root node and the values of all the nodes in the right sub-tree are greater than (or) equal to the root node then such type of binary tree can be called as a binary search tree.

Eg:-



Threaded Binary Tree:- When a binary tree is implemented ~~when~~ in the form of linked list then we may get NULL values in some of ~~the~~ fields of the nodes if NULL values are replaced with some addresses of the nodes in that particular binary tree then such type of binary tree can be called as a threaded binary tree.

Binary Tree Properties:-

1. If the binary tree contains 'N' no.of nodes then it contains N-1 no.of edges.
2. In the binary tree, every node contains only one parent node except the root node.

3. The maximum no.of nodes of a complete binary tree at level 'l' is 2^l no.of nodes. (l is level number)

4. The maximum no.of nodes of a complete binary tree with height 'h' is $(2^{h+1}-1)$ no of nodes

5. The total no.of leaf nodes of a complete binary tree is $(n+1)/2$ no.of nodes (n = total no.of nodes)

6. The total no.of external nodes = total no.of internal nodes + 1.

Binary Tree Traversal:-

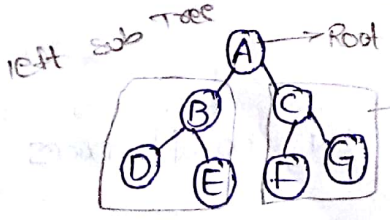
* The process of visiting and displaying every node in a binary tree is known as binary tree traversal

* the binary tree traversal can be performed by using 3 types of techniques there are

1) In-ordered Traversal
2) Pre-ordered Traversal
3) post-ordered Traversal.

In-ordered Traversal:- In this technique first the left sub-Tree will be traversed (or) visited and then root node and last the right sub Tree will be traversed. In the binary tree.

Eg:-



In-ordered :- ~~traversal~~ D-B-E-A-F-C-G

Pre-ordered traversal:-

procedure steps:-

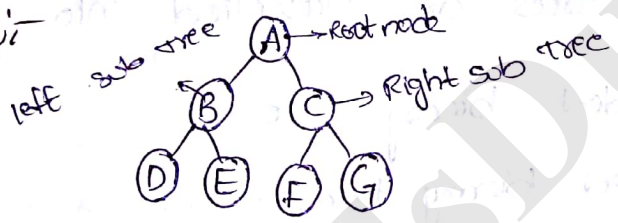
1. Traverse left sub-Tree
2. Traverse root node
3. traverse right sub-Tree.

pre-ordered Traversal:- In this technique first the root node is visited and then left sub-Tree and last the right sub-tree will be visited in the binary Tree.

procedure Steps:-

1. Transverse root node
2. Transverse left sub-Tree
3. Transverse right sub-Tree.

Eg:-



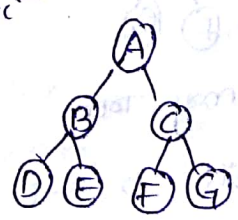
pre-order:- A - B - D - E - C - F - G

post-order Traversal:- In this technique first the left sub-Tree is will be traversed and then the right sub-Tree and last the root node will be traversed in the binary Tree.

procedure steps:-

1. Transverse left sub-Tree
2. Transverse right sub-Tree
3. Transverse root-node.

Eg:-



pre-order:-

D - E - B - F - G - C - A

Threaded Binary Tree:- If a binary tree is implemented in the form of double linked list then we may get NULL values in some fields of the nodes. If the NULL values are replaced with the addresses of some appropriate node in that binary tree then such binary tree can be called as a threaded binary tree.

The NULL value in the left link field of a node is replaced with predecessor node address of that particular node in the inorder form similarly the NULL value in the right link field of a node is replaced with sucessor node address of that particular node in the inorder form.

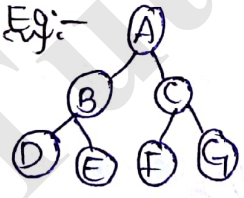
The threaded binary trees are classified into two types i) one-way threaded binary tree

ii) two-way threaded binary tree.

One-way threaded binary Tree:-

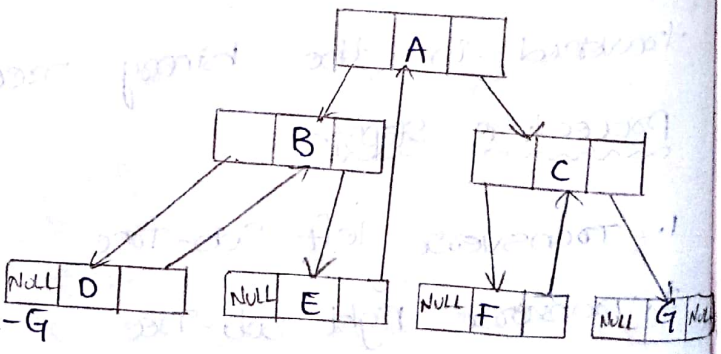
In this only the NULL value stored in the right link field of a node is replaced with appropriate node address in the binary tree

Eg:-



Binary Tree

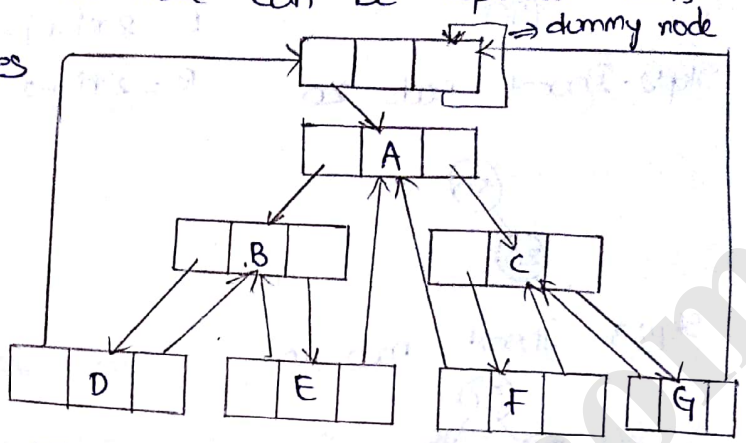
In-order:- D-B-E-A-F-C-G



One-way threaded binary Tr

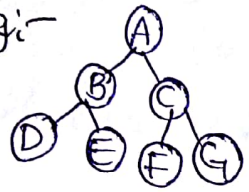
* Two-way threaded Binary Tree:-

In this the NULL values stored in the left link and right link fields of a node can be replaced with appropriate nodes addresses



Two-way threaded binary tree

Eg:-



Binary Tree

Inorder:-

D-B-E-A-F-C-G

Heap (or) Binary Heap:-

A heap is a tree based data structure in which the elements are arranged in the form of hierarchical structure

The heap follows two basic properties they are.

i) shape (or) structure property

ii) heap property

Shape property:- this property states that the heap must be either complete (or) almost complete binary tree. In the complete binary tree the elements are organised from top to bottom and left to right in each level.

Heap property:- this property states that all the parent nodes in the heap must be either greater than (or) less than to their children.

* Usually the heaps are classified into two types they are i) max heap ii) Min heap.

Max heap:- In a heap if all the parent nodes are greater than their children then such heap can be called as Max heap.

Eg:- construct a Max heap by the following elements.

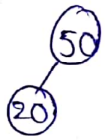
50, 20, 10, 40, 100.

Step:1 - Insert node 50

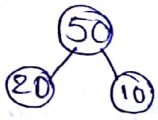
L = 2*i + 1 = 1

R = 2*i + 2

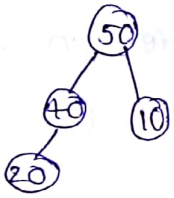
Step:2 - Insert node 20



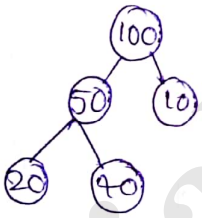
Step: 3 Insert node 10



Step:4 - Insert node 40

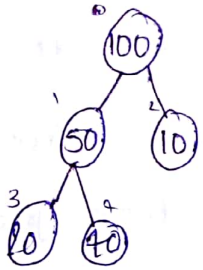


Step: 5 Insert node 100

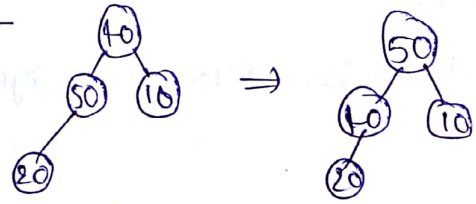


Deletion:-

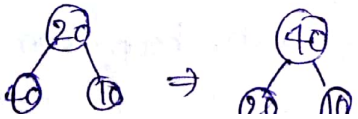
Eg:- Delete the element from the above Max heap.



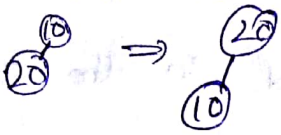
Step (1):-



(2) delete node 50



(3) delete node 40



4) delete node 20.

5) delete node 10

Min heap:- In a heap all the parent nodes are less than their children then such heap can be called as Min heap.

Priority Queue:- A priority queue is a linear data structure

Priority Queue:- A priority queue is a linear data structure in which the elements are stored in the sequencial order.

* A priority queue is a special queue in which the elements are stored in the order as they arrive along with their priorities and the elements will be deleted depends upon either max priority (or) Min priority.
* Usually the priority queues can be used in cpu shedduling
* the priority queue is classified into two types they are

1) Max priority queue
2) Min priority queue.

Max priority queue:- In this the elements are stored in the order as they arrive along with priorities and the elements are deleted depends upon highest priority.

Eg:- Insert the following elements into the max priority que elements are 40, 20, 30, 10 and priorities 5, 6, 4, 2 respectively

pq[4]

	0	1	2	3
Data	40	20	30	10
priority	5	6	4	2

MAX priority queue

Max priority queue deleted

40	30	10
5	4	2

Min priority queue:- In this the elements are stored in the order as they arrive along with priorities and the elements are deleted depends upon lowest priority.

Eg:- Insert the following elements into the Min priority queue.

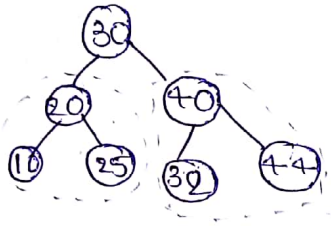
Min priority elements are. 40, 20, 30, 10 and 5, 6, 4, 2 respectively.

PQ [4]

Data	40	20	30	10
priority	5	6	4	2

Binary search Trees:- In a binary tree if the values of all the nodes in the left sub-tree are lessthan the root node and the values of all the nodes in the right sub tree are greaterthan (or) equal to the root node

Eg:-



In the Binary search tree we perform 5 basic operation they are
1. create()
2. Insert()
3. Delete()
4. search()
5. Display()

Time complexity:- the binary search takes O(log n) time complexity while performing either search (or) insert (or) delete operation

create:- this operation creates an initial node in the binary search tree.

Eg:- 30

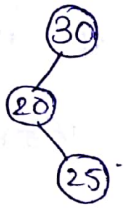
Insert:- this operation inserts a new node to the binary search tree

Eg:- Insert the following elements into the binary search tree.

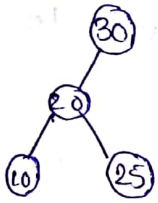
20, 25, 10, 40, 39, 45

1. Insert node 20

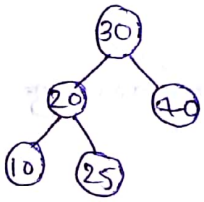
2. Insert node 25



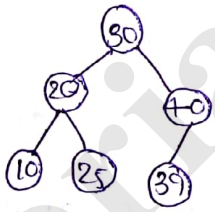
3. Insert node 10



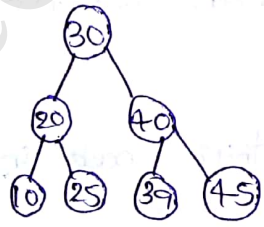
4. Insert node 40.



5. Insert node 39



6. Insert node 45



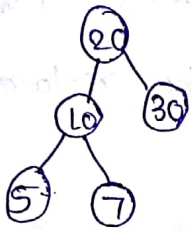
Delete :-

In the binary search tree the deletion can be performed in 3 possible cases they are.

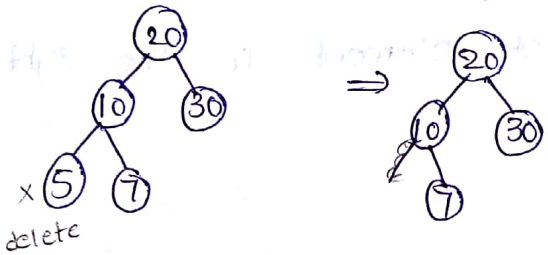
* case (1) Deleting a leaf node
case (2) Deleting a node having only one child
case (3) Deleting a node " Two children

case(1): In this case first we perform the search operation to find out the specific leaf node. and then the leaf node can be deleted by removing its connection from its parent node.

Eg:-



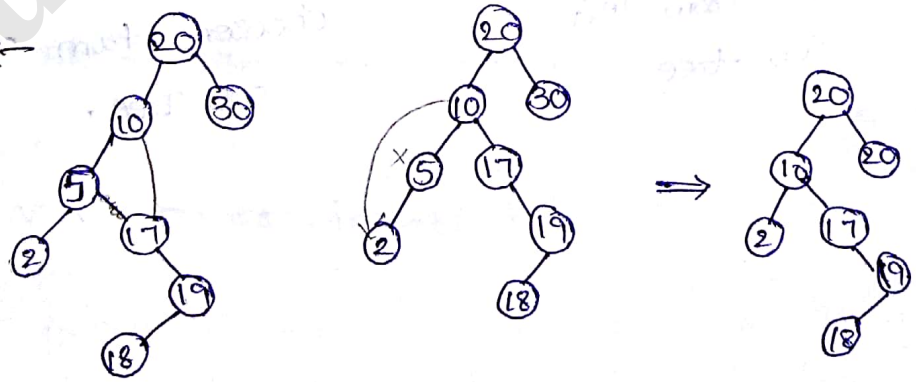
Delete node 5



case(2): Deleting a node having only one child.

* In this case first we perform search operation to locate specific element for the deletion and then the connection will be established between child node and parent node of the specific node.
* In other words a node can be deleted having only one child by establishing the connection between its child node and parent node

Eg:-



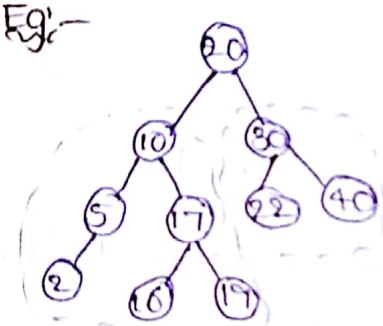
Delete node 5

Case(3): Deleting a node having 2 children:-

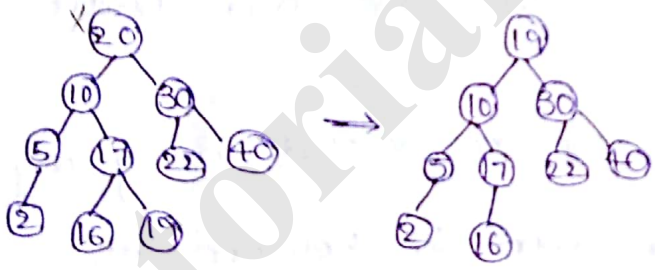
* In this case first we perform the search operation to locate the specific element for the deletion and then the highest element from the left sub-tree (or) lowest element from the right sub-tree can be replaced to the specific node for its deletion.

* In other words the node which is having 2 children can be deleted by replacing it with highest element in the left sub-tree (or) lowest element in the right sub-tree of that specific node.

Eg:-

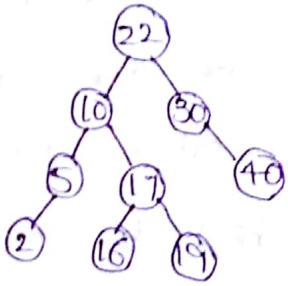


Delete node 20



highest elements choosen from left sub-tree

(or)



less element is choosen from right sub-Tree.

TutorialsDuniya.com

Download FREE Computer Science Notes, Programs, Projects, Books PDF for any university student of BCA, MCA, B.Sc, B.Tech CSE, M.Sc, M.Tech at https://www.tutorialsduniya.com

- Algorithms Notes
- Artificial Intelligence
- Android Programming
- C & C++ Programming
- Combinatorial Optimization
- Computer Graphics
- Computer Networks
- Computer System Architecture
- DBMS & SQL Notes
- Data Analysis & Visualization
- Data Mining
- Data Science
- Data Structures
- Deep Learning
- Digital Image Processing
- Discrete Mathematics
- Information Security
- Internet Technologies
- Java Programming
- JavaScript & jQuery
- Machine Learning
- Microprocessor
- Operating System
- Operational Research
- PHP Notes
- Python Programming
- R Programming
- Software Engineering
- System Programming
- Theory of Computation
- Unix Network Programming
- Web Design & Development

Please Share these Notes with your Friends as well

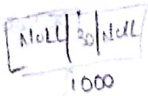
Implement binary search tree

```
#include <iostream.h>
#include <conio.h>
struct node
{
 int data;
 node *left;
 node *right;
}*root;
```

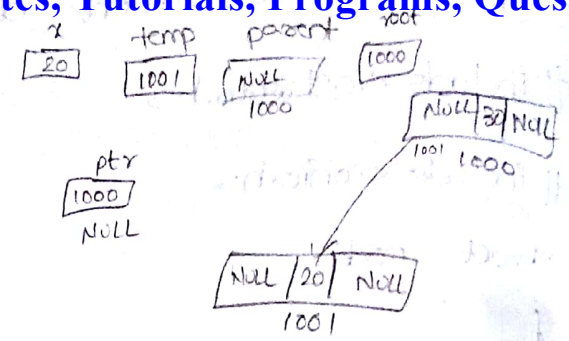
global structure.

```
class bstree.
 {
 public:
 void create( );
 void insert( );
 void in_order(node *);
 void pre_order(node *);
 void post_order(node *);
 void search(int);
 };
 void bstree::create( )
 {
 root = new node;
 cout<<"\n enter the data for the root node:";
 cin >>root ->data ;
 root -> left =root ->right = NULL ;
 }
 void bstree::insert( )
 {
  int x;
  node *temp, *parent ;
  cout<<"\n enter the element to be inserted:";
```

root 1000



```
cin>>x;
temp =new node;
temp->data = x;
temp -> left = NULL;
temp -> right = NULL;
parent = NULL;
if (root == NULL)
 {
 root = temp;
 }
else
{
node   *ptr;
ptr = root;
while(ptr != NULL)
{
 parent =ptr;
 if (x > ptr -> data)
 parent
 ptr = ptr ->right;
 else
 ptr = ptr -> left;
 }
 if (x < parent -> data)
 parent -> left = temp;
 else
 parent -> right = temp;
 }
}
void  bstree::in_order(node  *ptr)
{
if (ptr != NULL)
{
```



```
in_order (ptr->left);
cout<<"  "<<ptr->data<<"  ";
in_order (ptr->right);
 }
}
void  bstree::pre_order(node *ptr)
{
if(ptr!=NULL)
 {
 cout<<"  "<<ptr->data<<"  ";
 pre_order (ptr->left);
 pre_order (ptr->right);
 }
}
void  bstree::post_order(node *ptr)
 {
 if(ptr!=NULL)
 {
 post_order (ptr->left);
 post_order (ptr->right);
 cout<<"  "<<ptr->data<<"  ";
 }
 }
 void  bstree::search(int key)
 {
 int  flag=0;
 node *temp;
 temp=root;
 if temp==NULL)
 {
 cout<<"\n BST is empty";
```

```
    }
else
{
while(temp != NULL)
{
if (temp -> data == key)
{
cout<<key<<"  "<<"is found"<<endl;
flag=1;
break;
}
if (temp -> data > key)
temp= temp -> left;
if (temp -> data < key)
temp = temp -> right;
}
}
if (flag==0)
cout<<key<<"  "<<" is not found"<<endl;
}
void main()
{
bstree bs;
int option, k;
clrscr();
cout<<"1. create()"<<endl;
cout<<" 2. Insert()"<<endl;
cout<<" 3. in_order()"<<endl;
cout<<" 4. pre_order()"<<endl;
cout<<" 5. post_order()"<<endl;
```

```
cout<<"(6. search)"<<endl;
cout <<"7. exit "<<endl;
do
 {
 cout<<"\n enter your option: ";
 cin>> option;
 switch (option)
  {
  case 1:
  bs. create( ) ;
  break;
  case 2:
  bs. insert( );
   break;
  case 3:
  bs. in_order(root);
  break;
   case 4:
  bs. pre_order(root);
  break;
  case 5:
  bs. post_order(root);
   break;
  case 6:
  cout<<"enter the element to be searched:";
   cin>>k;
   bs. search(k);
   break;
  default :
  cout<<"\n enter a valid option: ";
   }
  }
 while (option!= 7);
 getch( );
}
```

Graphs

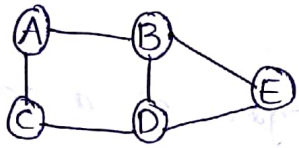
* A graph is a non-linear data structure in which the elements are arranged in the form of random order. A graph is a collection of vertices and edges

*Usually A graph can be represented as $G = (V, E)$

where V = collection of vertices (or) nodes

E = collection of edges

Eg:-



V = A, B, C, D, E

E = {(A,B), (A,C), (B,D), (B,E), (C,D), (D,E)}

Terminology of graphs:-

vertex:-

In a graph every individual element (or) node can be called as a vertex

Edge:- In a graph, a connection link between a pair of vertices is known as an edge.

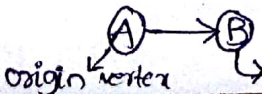
End points (or) End vertices:-

In a graph if there is an edge between a pair of vertices then those vertices are said to be end points (or) end vertices of that edge.

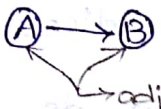
Origin and Destination vertices:-

In a graph if there is a directed edge between a pair of vertices then the starting vertex of the edge can be called as origin vertex and the last vertex of an edge can be called as destination vertex.

Eg:-



Adjacent vertices:- In a graph if there is an edge between the vertices A and B then A and B vertices are said to be adjacent vertices.



Degree of a vertex:- In a graph, the total no of edges are connected to a vertex is said to be degree of that vertex.

* In a directed graph the degree of a vertex can be measured in two ways i) In-degree of a vertex
ii) out-degree of a vertex.

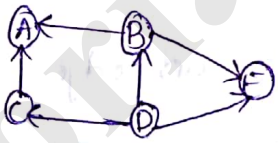
In-degree of a vertex:-

In a graph the total no of edges are coming to a vertex is known as in-degree of that vertex.

Out-degree of a vertex:-

In a graph the total no of edges are going outside from a vertex is known as out-degree of that vertex

Eg:-

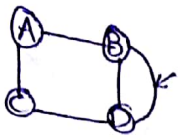


Vertex	In-degree	out-degree.
A	2	0
B	1	2
C	1	1
D	0	3

Loop (or) self Loop:- In a graph, if an edge contains same end-points then such edge can be called as loop (or) self loop.

Parallel edges:- In a graph if a pair of vertices are having more than one edge then those edges are said to be parallel edges.

Eg:-



Parallel edges

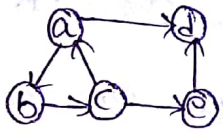
Path:- A sequence of consecutive edges from one vertex to another vertex in a graph is said to be a path.

A-D ⇒ A-B-D (or) A-C-D.

Closed path:-

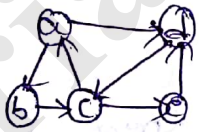
In a graph if a path contains same end points then such path can be called as a closed path.

Eg



a-b-c-a is closed path

Cycle:- In a graph every closed path is known as a cycle. Eg:-



a-b-c-a is a cycle

d-c-e-d is a cycle

a-d-c-a is a cycle

a-b-c-e-d-c-a is a cycle

Types of graphs:-

There have several graphs

These are several graphs and among them the graphs are classified into based on the frequently usage. They are.

1. NULL graph
2. Trivial graph
3. Regular "
4. Sub "
5. undirected "
6. directed "
7. Mixed "
8. weighted "
9. Simple "
10. Multi "
11. Complete "
12. Connected "

NULL Graph:- In a graph if there are no more edges between the vertices then such graph can be called as a NULL graph.

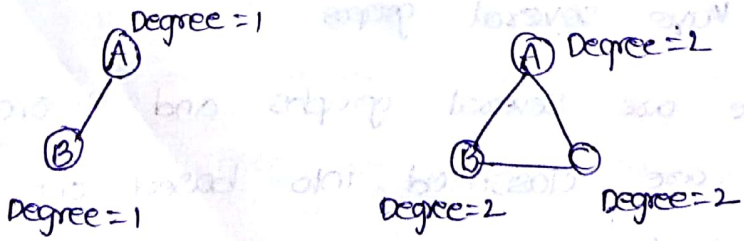
Eg:- (A) (B) (C)

Trivial Graph:- In a graph if there is only one vertex then such graph can be called as a trivial graph.

Eg:- (A)

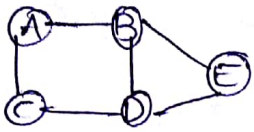
Regular Graph:- In a graph every vertex has same degree then such graph can be called as a regular graph.

Eg:-

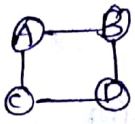


Sub-graph:- In a graph a part of graph can be called as a sub-graph.

Eg:-



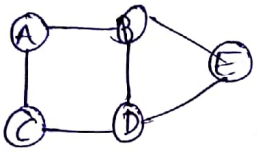
graph



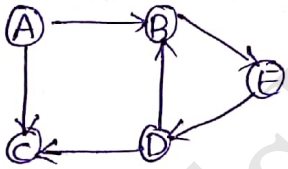
sub graph.

Undirected-graph:- In a graph, all the edges do not have directions then such graph can be called as an undirected graph.

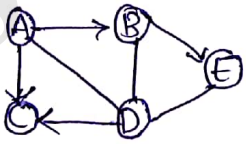
Eg:-



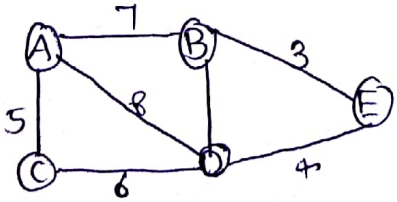
Directed graph:- In a graph, if all the edges have directions then such graph can be called as directed graph



Mixed graph:- In a graph, if some edges have directions and some edges do not have directions then such graph can be called as a mixed graph.

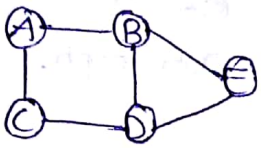


Weighted graph:- In a graph all the edges are assigned with some weights (or) values then such graph can be called as weighted (or) label graphs

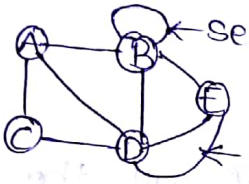


Simple graph:- In a graph, if there is no self loops (or) parallel edges then such graph can be called as Simple graph.

Eg:-

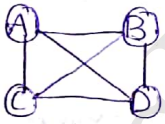


Multi-graph:- In a graph, if there is either self loop (or) parallel edges (or) both then such graph can be called as a multi-graph.

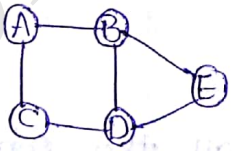


Complete graph:- In a graph, if every vertex is adjacent to other vertices then such graph can be called as a complete graph.

Eg:-



Connected graph:- In a graph, if every pair of vertices is having a path then such graph can be called as a connected graph.



Graph Representation:-

Usually the graphs are represented in two ways

1. adjacency matrix representation
2. Adjacency list representation.

Adjacency Matrix representation:- An adjacency Matrix is a square matrix which stores the value either '0' (or) '1'.

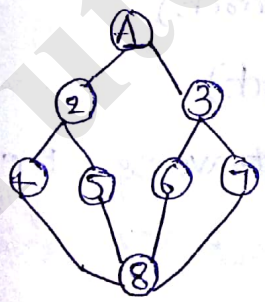
* In the adjacency matrix, if there is an edge from one vertex to another vertex in the graph, then 'one' will be stored otherwise '0' will be stored at a specific position.
* If a graph contains 'n' no.of vertices then the adjacency matrix contains nxn no.of rows & columns

$$Adj_{i,j} = \begin{cases} 1 & \text{if there is an edge from } v_i \text{ to } v_j \\ 0 & \text{otherwise} \end{cases}$$

where i represents row

j represents column

Eg:- Graph



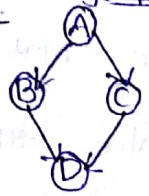
⇒

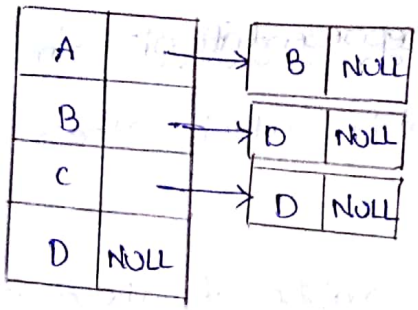
	1	2	3	4	5	6	7	8
1	0	1	1	0	0	0	0	0
2	1	0	0	1	1	0	0	0
3	1	0	0	0	0	1	1	0
4	0	1	0	0	0	0	0	1
5	0	1	0	0	0	0	0	1
6	0	0	1	0	0	0	0	1
7	0	0	1	0	0	0	0	1
8	0	0	0	1	1	1	1	0

Adjacency List Representation:-

In this representation, the adjacency vertices of a vertex x can be represented in the form of single link list.

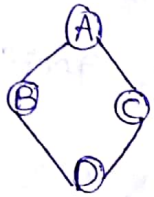
Eg:- graph

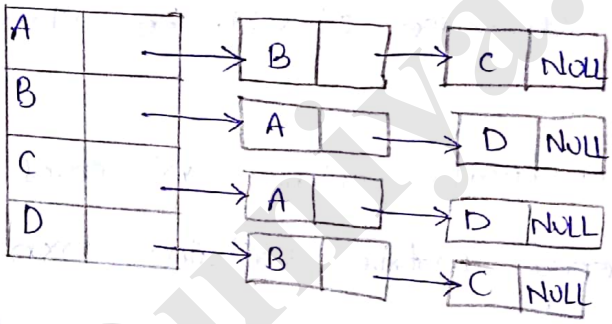




adjacency list.

Eg:-





Graph Traversal:-

The process of visiting and displaying every vertex in a graph is known as graph traversal.

The graph traversal can be performed using two techniques they 1) BFS (Breadth first search)
2) DFS (Depth first search)

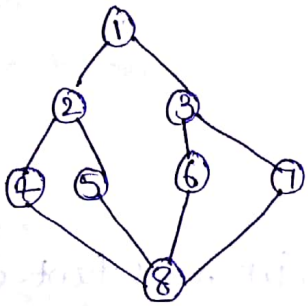
BFS :- this technique uses queue to traverse the given graph.

* In this we first visit any vertex from the graph and then its adjacent unvisited vertices will be visited
* Among the no. of adjacent visited vertices we select one visited vertex and visit its adjacent unvisited

vertices and second so on.

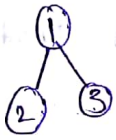
* this process will be continued until all the vertices are visited in the given graph.

Eg:-

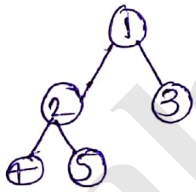


Step (1):- Visit the vertex (1) as the starting vertex.

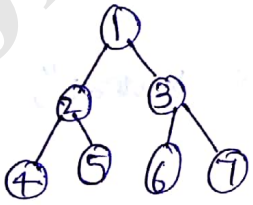
step (2):- vertex (1) has two unvisited vertices (2) & (3) so that visit both



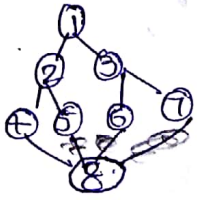
step(3):- vertex (2) has two unvisited vertices (4) & (5) so that visit both



step(4):- vertex (3) has two unvisited vertices (6) & (7) so that visit both



Step(5):- vertex (4) has only 1 unvisited vertex that is vertex (8) so that visit it.



Step(6):- vertex (5) does not have any unvisited vertices

Step(7):- vertex (6) does not have any unvisited vertices

Step(8):- vertex (7) does not have any unvisited vertice

Step(9):- vertex (8) does not have any unvisited vertice

Program to Implement BFS:-

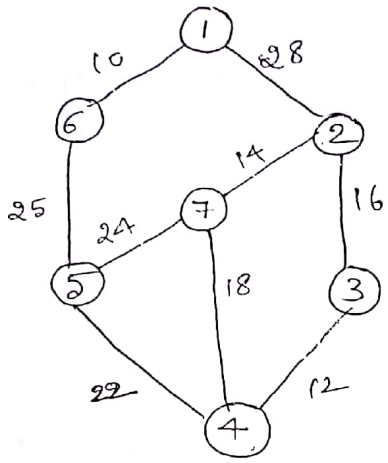
```
#Include <iostream.h>
#include <conio.h>
int adj[10][10], queue[10], visited[10], i,j,k,n,v, front=0, rear=
void main()
{
clrscr();
cout<<" Enter the no of vertices :";
cin>>n;
cout<<"\n Enter the data for the adjacency matrix\n";
for(i=1;i<=n;i++)
{
for (j=1; j<=n; j++)
{
cin >>adj[i][j];
}
}
cout<<"\n Enter the initial vertex:";
cin >>v;
cout<<"\n Enter the visited vertex order is :";
cout <<v<<" ";
visited[v]=1;
k=1;
while(k<n)
{
for (j=1 ; j<=n; j++)
```

```
{
if (adj[v][j]!=0 && visited[j]!=1)
  {
  rear++;
  queue[rear]=j;
  }
  front++;
  v=queue[front];
  cout<<v<<"  ";
  visited[v]=1;
  k++;
  }
getch();
}
```

Minimum Spanning Trees :-

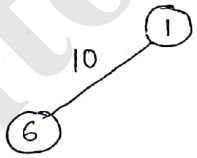
Given a connected weighted graph G it is often desired to create a Spanning Tree T for G such that the sum of the weights of the tree edges in T is as small as possible. Such a tree is called a Minimum Spanning Tree

Krushkal's Algorithm:-

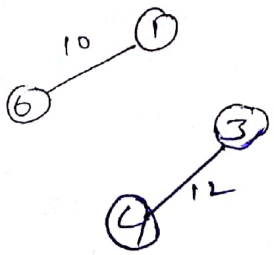


Step 1:- Arrange the weights of all the edges in ascending order 10, 12, 14, 16, 18, 22, 24, 25, 28.

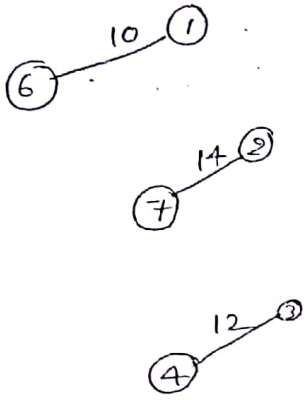
Step 2:- Select the minimum edge from the sorted list ie; edge (1,6)



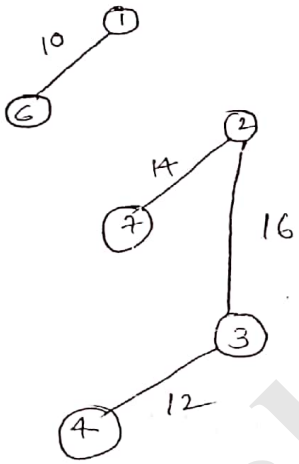
Step 3:- Select the next minimum edge from the sorted list ie; edge (3,4)



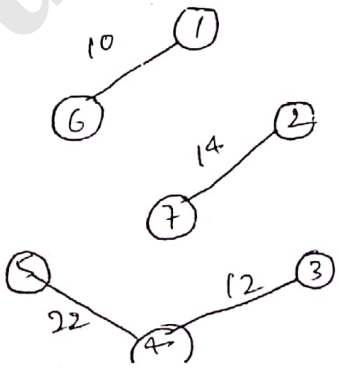
Step4:- Select the next minimum edge from the sorted list ie; edge(2,7)

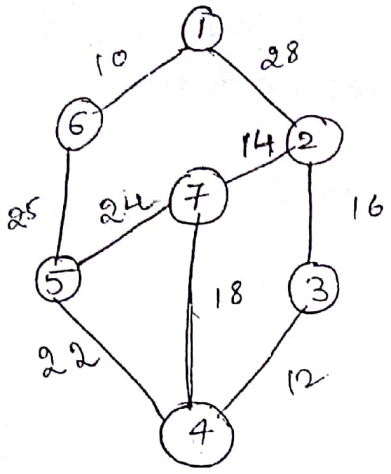


Step5:- Select the next minimum edge from the sorted list ie edge(2,3)

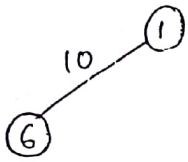


Step6:- Select the next minimum edge from the sorted list ie edge(7,4) By placing the edge(7,4) a loop is formed so reject the edge(7,4) and Select the next minimum edge ie edge(5,4)

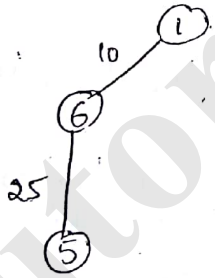




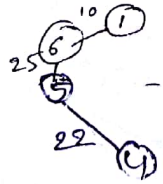
Step 1:- Select the minimum edge from the graph ie; edge (1, 6)



Step 2:- The adjacent edge for vertex ① is edge (1, 2) with cost 28 and the adjacent edge for vertex ⑥ is edge (6, 5) with cost 25 select the minimum edge from the above two edges ie ; edge (6, 5)



Step 3:- The adjacent edge for vertex ① is edge (1, 2) with cost 28 and the adjacent edge for vertex ⑤ is edge (5, 4) with cost 22 select the minimum edge from the above two edges ie ; edge (5, 4)



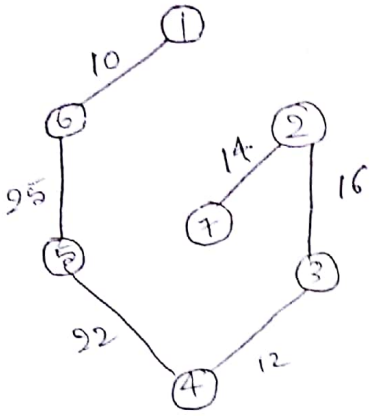
TutorialsDuniya.com

Download FREE Computer Science Notes, Programs, Projects, Books PDF for any university student of BCA, MCA, B.Sc, B.Tech CSE, M.Sc, M.Tech at https://www.tutorialsduniya.com

- Algorithms Notes
- Artificial Intelligence
- Android Programming
- C & C++ Programming
- Combinatorial Optimization
- Computer Graphics
- Computer Networks
- Computer System Architecture
- DBMS & SQL Notes
- Data Analysis & Visualization
- Data Mining
- Data Science
- Data Structures
- Deep Learning
- Digital Image Processing
- Discrete Mathematics
- Information Security
- Internet Technologies
- Java Programming
- JavaScript & jQuery
- Machine Learning
- Microprocessor
- Operating System
- Operational Research
- PHP Notes
- Python Programming
- R Programming
- Software Engineering
- System Programming
- Theory of Computation
- Unix Network Programming
- Web Design & Development

Please Share these Notes with your Friends as well

Step 4:- Select the next minimum edge from the sorted list is edge (5,7) it is rejected and the next minimum edge (5,6) is selected.



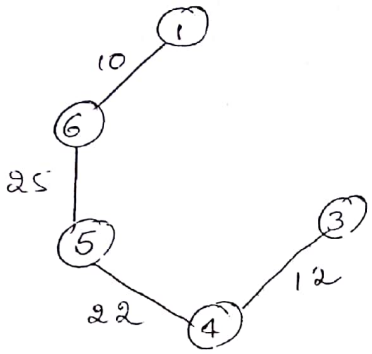
The Minimum cost of the spanning tree is

$10+12+14+16+22+25 = 99$

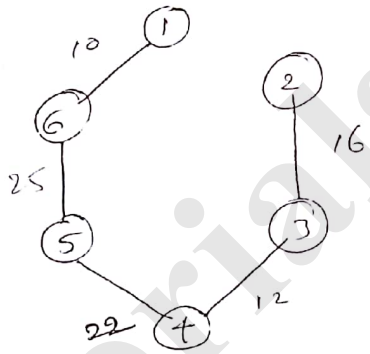
Algorithm :-

```
MST-KRUSKAL(G,W)
1) A ← ∅
2) for each vertex V ∈ V[G]
       do MAKE-SET(V)
3) Sort the edges of E into increasing order by weight W
4) for each edge (u,v) ∈ E taken in increasing order
   do if FIND-SET(u) ≠ FIND-SET(V)
      then A ← A∪{u,v}
           UNION(u,v)
5) return A
```

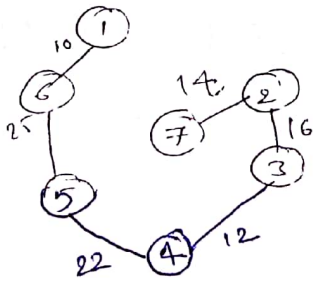
The adjacent edge for vertex ① is edge (1,2) with cost 28 and the adjacent edge for vertex ④ is edge (4,3) with cost 12 and also edge (4,7) with cost 18 so select the minimum edge from the above edges i.e edge (4,3)



Step5:- The adjacent edge for vertex ① is edge (1,2) with cost 28 and the adjacent edge for vertex ③ is edge (3,2) with cost 16, select the minimum edge from the above edges i.e; edge (3,2)



Step6:- The adjacent edge for vertex ① is edge (1,2) with cost 28 and the adjacent edges for vertex ② is edge (2,7) with cost 14, select the minimum edge from the above edges i.e edge (2,7)



Minimum costs =
10+25+22+12+16+14
= 99

Algorithm :-

```
Prims_Algorithm(int graph[u][v])
{
 int parent[v]; //Array to store
 int key[v];          constructed MST
 bool mstset[v]; //to represent set of vertices included
                                        in MST
 for(int i=0; i<v; i++)
  key[i]=INT_MAX, mstset[i]=false;
  key[0]=0;
  Parent[0]=-1;
  for(int count=0; count<v-1; count++)
   {
    int u=minkey(key, mstset);
    mstset[u]=true;
    for(int v=0; v<V; v++)
    if(graph[u][v] && mstset[v]==false && graph[u][v
                                    <key[v])

    Parent[v]=u, key[v]=graph[u][v];
   }
    Print MST(parent, V, graph)
  }
```

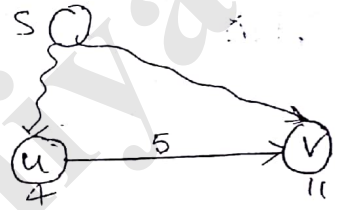
Dijstra's Single Source shortest-path Algorithm

Dijstra's Algorithm is named after a Dutch computer scientist and Dijstra is a greedy algorithm that solves the Single-source shortest path problem for a directed graph G=(V,E) with non-negative edge weights

ra's Algorithm maintains a set S of vertices
e final shortest path weights from the source
have already been determined.

The Algorithm repeatedly selects the vertex with
he minimum shortest path estimate and relaxes
ll the edges from that vertex. The single source
shortest path algorithm is based on a technique
known as relaxation. The process of relaxing
n edge (u,v) consist of testing whether we can
mprove the shortest path from u to v found
ofar going through u.

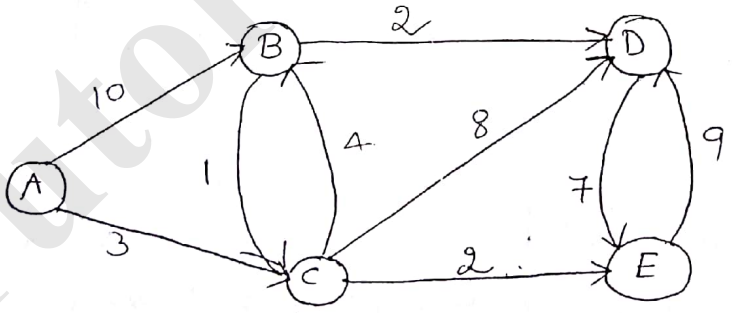
Algorithm for relaxing an edge:-



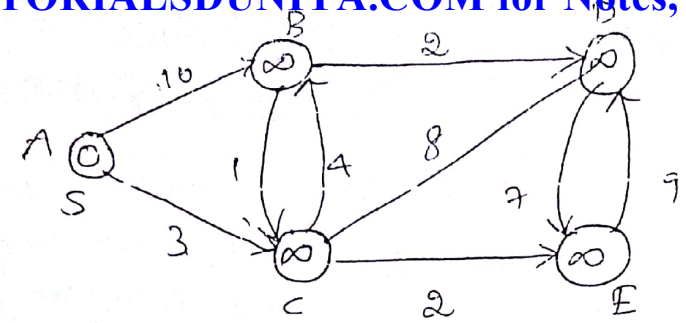

```
Relax(u,v,w)
{
 if  d[v] > d[u] + w(u,v)
 then  d[v] ← d(u) + w(u,v)
   π[v] ← u;
}
```

p

Q:-



Step1:- Initialize the minimum value to 0 and remaining edges to ∞

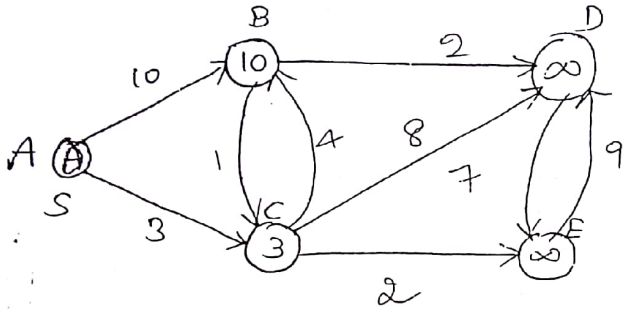


Q :	A	B	C	D
	0	∞	∞	∞

$S : \{A\}$

Step2:- EXTRACT_MIN(Q) [A]

Relax all the edges leaving A

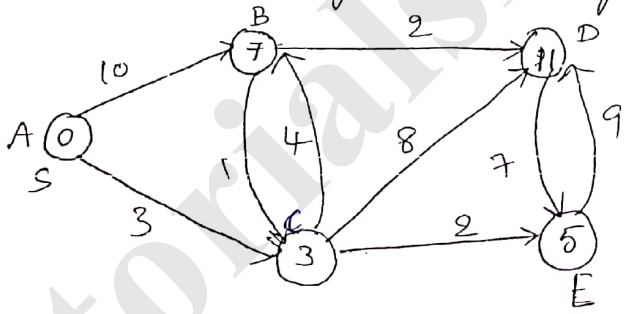


Q :	A	B	C	D	E
	0	∞	∞	∞	∞
		10	3	∞	∞

$S : \{A\}$

Step3:- EXTRACT_MIN(Q) [C] $S : \{A, C\}$

Relax all the edges leaving C

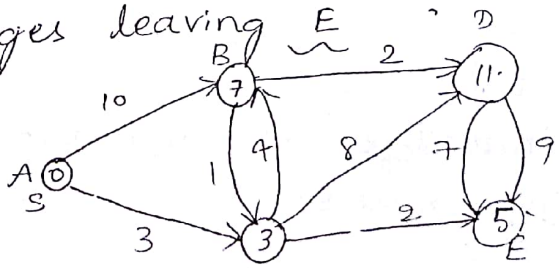


Q :	A	B	C	D	E
	0	∞	∞	∞	∞
		10	3	∞	∞
		7		11	5

Step4:- EXTRACT_MIN(Q) [E]

Relax all the edges leaving E

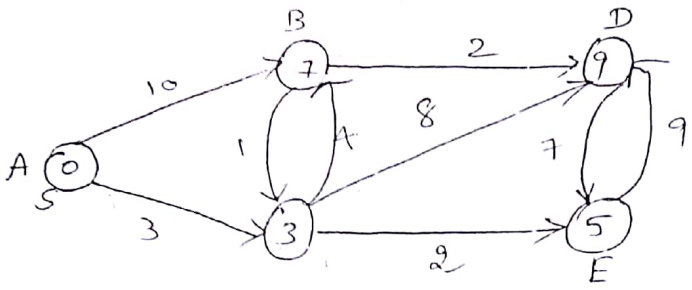
$S : \{A, C, E\}$



5:- EXTRACT-MIN(Q) [B]

Q:	A	B	C	D	E
	0	∞	∞	∞	∞
		10	3	∞	∞
		7		11	5
	0	7	3	9	5

S: { A, C, E, B }



S: { A, C, E, B, D }

The shortest path from a single source A to the remaining vertices : A — B = 7

A — C = 3

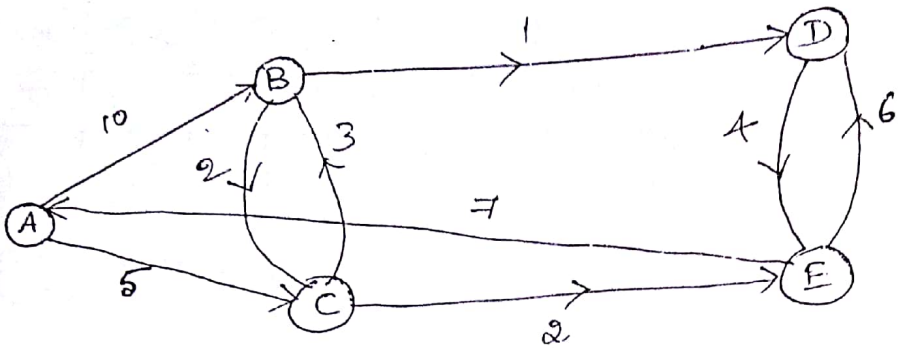
A — D = 9

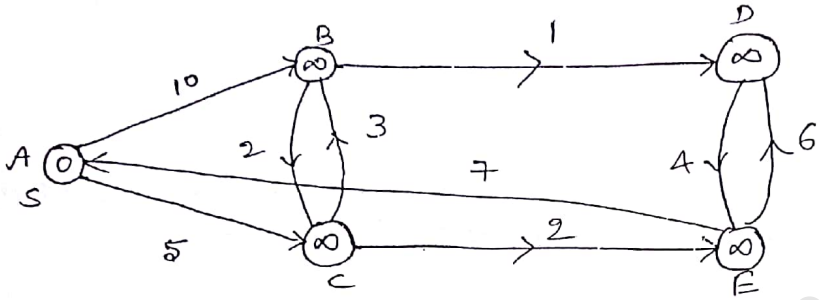
A — E = 5

Algorithm

```
Dijstra (G, w, S)
{
  initialize-Single-Source (G, S)
    S ← ϕ;
    Q ← V[G];
    while Q = ϕ
      do
        u ← Extract-MIN(Q)
        S ← S ∪ {u}
      for each vertex V ∈ adj[u]
        do relax (u, v, w)
}
```

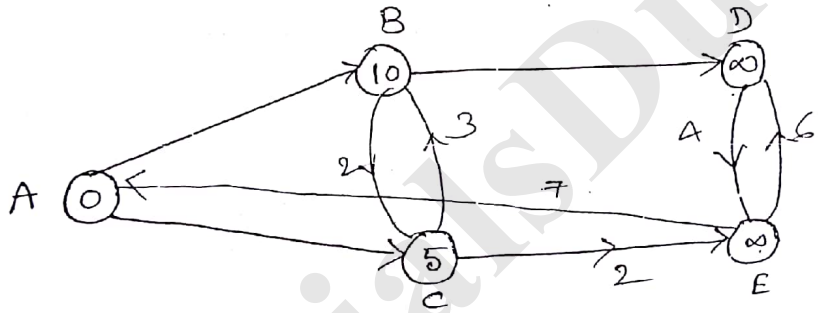
Q:-





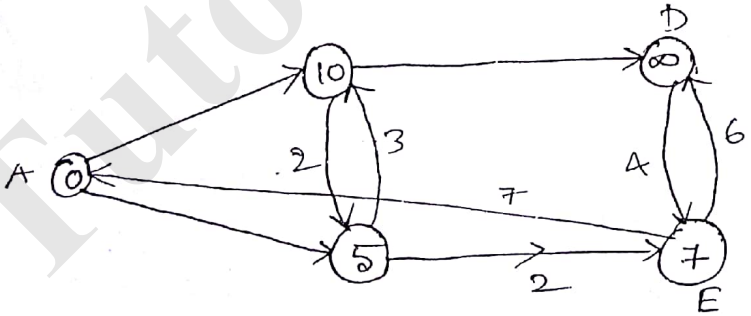
S: {A}

Q	A	B	C	D	E
	0	∞	∞	∞	∞



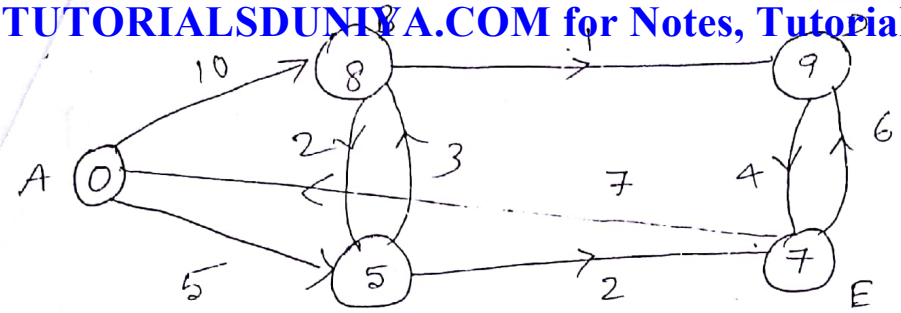
S: {A, C}

Q	A	B	C	D	E
	0	∞	∞	∞	∞
		10	5	∞	∞



S: {A, C, E}

Q	A	B	C	D	E
	0	∞	∞	∞	∞
		10	5	∞	∞
		10	5	∞	

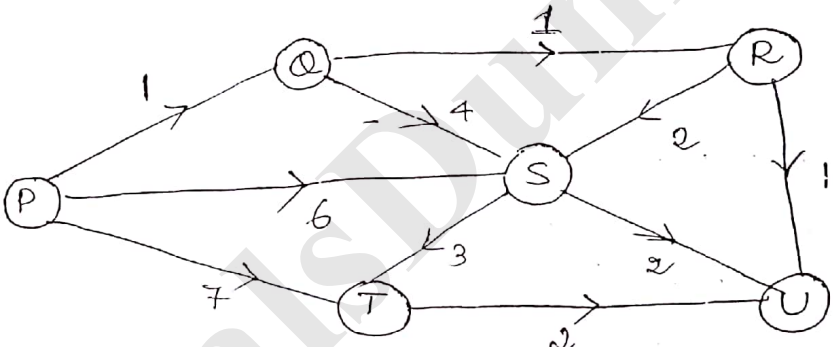


Q	A	B	C	D	E
	0	∞	∞	∞	∞
		10	5	∞	∞
		10	5	∞	7
		10	5	8	7

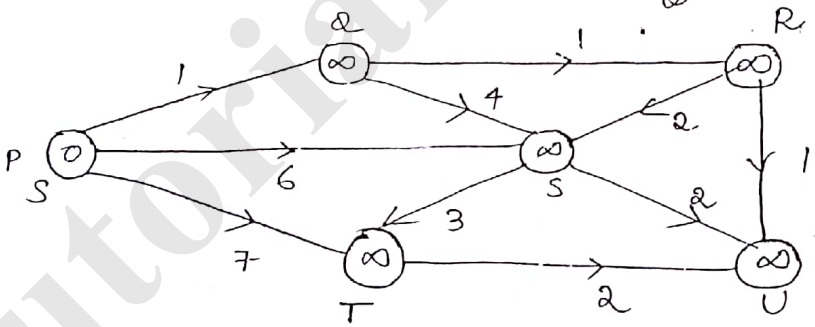
S: {A, C, E, B, D}

The shortest path from a single source A to the remaining vertices: A — B = 10
A — C = 5
A — D = 8
A — E = 7

eg:-

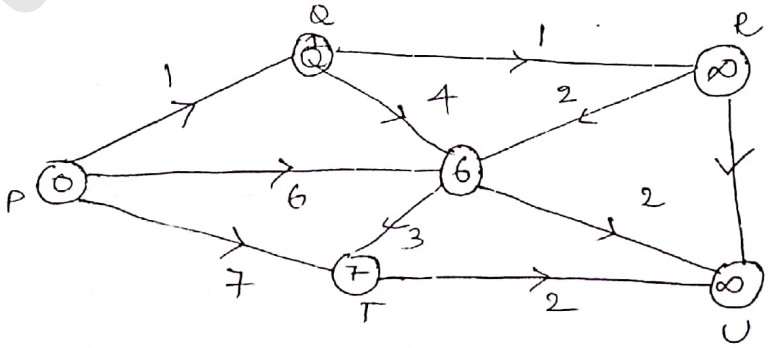


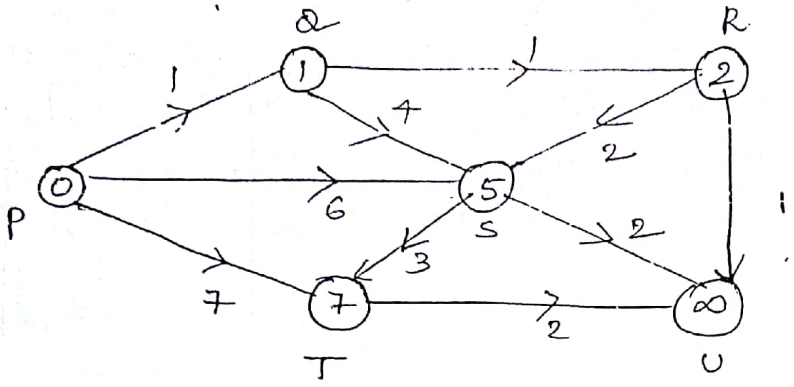
i)



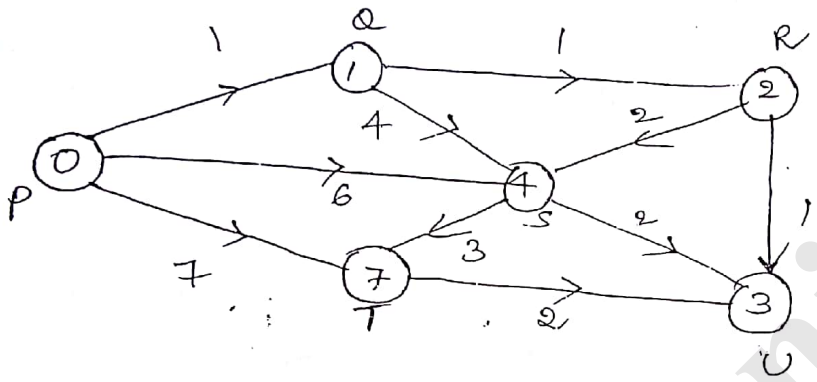
S: {P}

ii)





S: {P, Q, R}



S: {P, Q, R, U}

S: {P, Q, R, U, S}

S: {P, Q, R, U, S, T}

The shortest path from a single source A to the remaining vertices: P — Q = 1

P − R = 2

P − S = 4

P − T = 7

P − U = 3

TUTORIALSDUNIYA.COM

Get FREE Compiled Books, Notes, Programs, Question Papers with Solution etc of the following subjects at https://www.tutorialsduniya.com

- C and C++
- Java
- Data Structures
- Computer Networks
- Android Programming
- PHP Programming
- JavaScript
- Java Server Pages
- Python
- Microprocessor
- Artificial Intelligence
- Machine Learning
- Computer System Architecture
- Discrete Structures
- Operating Systems
- Algorithms
- DataBase Management Systems
- Software Engineering
- Theory of Computation
- Operational Research
- System Programming
- Data Mining
- Computer Graphics
- Data Science

- DU Programs: https://www.tutorialsduniya.com/programs
- TutorialsDuniya App: http://bit.ly/TutorialsDuniyaApp
- C++ Tutorial: https://www.tutorialsduniya.com/cplusplus
- Java Tutorial: https://www.tutorialsduniya.com/java
- JavaScript Tutorial: https://www.tutorialsduniya.com/javascript
- Python Tutorial: https://www.tutorialsduniya.com/python
- Kotlin Tutorial: https://www.tutorialsduniya.com/kotlin
- JSP Tutorial: https://www.tutorialsduniya.com/jsp

All pairs shortest path Algorithm (or) Floyd warshall Algorithm.

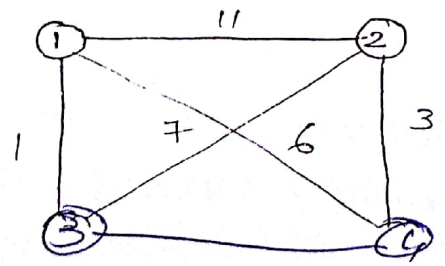
The Algorithm considers the intermediate vertices of a shortest path where an intermediate vertex of a simple path $p = \langle v_1, v_2, \ldots, v_m \rangle$ is any vertex of p other than v_1 or v_m. The Floydwarshall algorithm is based on the following observation:

let the vertices of G be $V = \{1, 2, 3 \ldots n\}$ and consider subset $\{1, 2, 3 \ldots k\}$ of vertices for any pair of vertices $i, j \in V$. consider all paths from i to j whose intermediate vertices are all drawn from $\{1, 2 \ldots k\}$ and let p be the minimum weight path from among them. The Floydwarshall algorithm exploits a relationship between path p and shortest paths from i to j with all intermediate vertices in the set $\{1, 2 \ldots k-1\}$

If k is intermediate vertex of path p and let $d_{ij}^{(k)}$ be the weight of a shortest path from vertex i to j with all intermediate vertices in the set $\{1, 2 \ldots k\}$

$$d_{ij}^{(k)} = \begin{cases} w_{ij} & \text{if } k = 0 \\ \min\left(d_{ij}^{(k-1)}, d_{ik}^{(k-1)} + d_{kj}^{(k-1)}\right) & \text{if } k \geq 1 \end{cases}$$

Example:-



$$D^0 = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \end{array} \begin{array}{c} \begin{array}{cccc} 1 & 2 & 3 & 4 \end{array} \\ \begin{bmatrix} 0 & 11 & 1 & 6 \\ 11 & 0 & 7 & 3 \\ 1 & 7 & 0 & 2 \\ 6 & 3 & 2 & 0 \end{bmatrix} \end{array}$$

Algorithm:-

```
FLOYD-WARSHALL(W)
{
 n ← rows[W]
 D(0) ← W
for k ← 1 to n
do for i ← 1 to n
do for j ← 1 to n
do dij(k) ← min(dij(k-1), dik(k-1) + dkj(k-1))
return D(n)
}
```

Step 1:- calculate D^1 {1}

To calculate D^1 fix the first row and first colocumn of the D^0 matrix. Find the remaining elements add the corresponding row and colocumn numbers and replace the position with minimum

$$D^1 = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \end{array} \begin{array}{c} \begin{array}{cccc} 1 & 2 & 3 & 4 \end{array} \\ \begin{bmatrix} 0 & 11 & 1 & 6 \\ 11 & 0 & 7 & 3 \\ 1 & 7 & 0 & 2 \\ 6 & 3 & 2 & 0 \end{bmatrix} \end{array}$$

$D^1(2,3) = \min((2,3), (2,1)+(1,3))$

$= \min(7, 11+1)$

$= \min(7, 12)$

$= 7$

$D^1(2,4) = \min((2,4), (2,1)+(1,4))$

$= \min(3, 11+6) = \min(3, 17) = 3$

$D'(3,2) = \min((3,2), (3,1)+(1,2))$

$= \min(7, 1+11)$

$= \min(7, 12)$

$= 7$

$D'(3,4) = \min((3,4), (3,1)+(1,4))$

$= \min(2, 1+6)$

$= \min(2, 1+6)$

$= 2$

$D'(4,2) = \min((4,2), (4,1)+(1,2))$

$= \min(3, 6+11)$

$= \min(3, 17)$

$= 3$

$D'(4,3) = \min((4,3), (4,1)+(1,3))$

$= \min(2, 6+1)$

$= \min(2, 7)$

$= 2$

$$D' = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \end{array} \begin{array}{c} \begin{array}{cccc} 1 & 2 & 3 & 4 \end{array} \\ \begin{bmatrix} 0 & 11 & 1 & 6 \\ 11 & 0 & 7 & 3 \\ 1 & 7 & 0 & 2 \\ 6 & 3 & 2 & 0 \end{bmatrix} \end{array}$$

Step 2:- Calculate $D^2\{1,2\}$

To calculate D^2 fix the second row and second coloumn of D' add the corresponding elements from choosen row and coloumn and update with the minimum value

$$D^2 = \begin{array}{c|cccc} & 1 & 2 & 3 & 4 \\ \hline 1 & 0 & 4 & 11 & 6 \\ 2 & 11 & 0 & 7 & 3 \\ 3 & 1 & 7 & 0 & 2 \\ 4 & 6 & 3 & 9 & 0 \end{array}$$

$D^2(1,3) = \min((1,3), (1,2)+(2,3))$

$= \min(11, 4+7)$

$= \min(11, 11)$

$= 11$

$D^2(1,4) = \min((1,4), (1,2)+(2,4))$

$= \min(6, 4+3)$

$= \min(6, 7)$

$= 6$

$D^2(3,1) = \min((3,1), (3,2)+(2,1))$

$= \min(1, 7+4)$

$= \min(1, 11)$

$= 1$

$D^2(3,4) = \min((3,4), (3,2)+(2,4))$

$= \min(2, 7+3)$

$= \min(2, 10)$

$= 2$

$D^2(4,1) = \min((4,1), (4,2)+(2,1))$

$= \min(6, 3+4)$

$= 6$

$D^2(4,3) = \min((4,3), (4,2)+(2,3))$

$= \min(9, 3+7)$

$= \min(9, 10)$

$= 9$

3:- To calculate $D^3\{1,2,3\}$

Fix the third column and third row from D^2 and add corresponding elements of selected row and column choose the minimum and update the matrix

$$D^3 = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \end{array} \begin{array}{c} \begin{array}{cccc} 1 & 2 & 3 & 4 \end{array} \\ \begin{bmatrix} 0 & 8 & 1 & 3 \\ 8 & 0 & 7 & 3 \\ 1 & 7 & 0 & 2 \\ 3 & 3 & 2 & 0 \end{bmatrix} \end{array}$$

$D^3(1,2) = \min((1,2), (1,3)+(3,2))$
$= \min(11, 1+7)$
$= \min(11, 8)$
$= 8$

$D^3(2,1) = \min((2,1), (2,3)+(3,1))$
$= \min(11, 7+1)$
$= \min(11, 8)$
$= 8$

$D^3(1,4) = \min((1,4), (1,3)+(3,4))$
$= \min(6, 1+2)$
$= \min(6, 3)$
$= 3$

$D^3(4,1) = \min((4,1), (4,3)+(3,1))$
$= \min(6, 2+1)$
$= \min(6,3)$
$= 3$

$D^3(2,4) = \min((2,4), (2,3)+(3,4))$
$= \min(3, 7+2)$
$= \min(3, 9)$
$= 3$

$D^3(4,2) = \min((4,2), (4,3)+(3,2))$

To calculate D^4 fix the fourth row and [illegible] of D^3 and add the corresponding elements and select the minimum value and update the matrix.

$$D^4 = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \end{array} \begin{array}{c} \begin{array}{cccc} 1 & 2 & 3 & 4 \end{array} \\ \begin{bmatrix} 0 & 6 & 1 & 3 \\ 6 & 0 & 5 & 3 \\ 1 & 5 & 0 & 2 \\ 3 & 3 & 2 & 0 \end{bmatrix} \end{array}$$

$D^4(1,2) = \min((1,2), (1,4)+(4,2))$
$= \min(8, 3+3)$
$= \min(8,6)$
$= 6$

$D^4(1,3) = \min((1,3), (1,4)+(4,3))$
$= \min(1, 3+2)$
$= \min(1,5)$
$= 1$

$D^4(2,1) = \min((2,1), (2,4)+(4,1))$
$= \min(8, 3+3)$
$= \min(8,6)$
$= 6$

$D^4(2,3) = \min((2,3), (2,4)+(4,3))$
$= \min(7, 3+2)$
$= \min(7,5)$
$= 5$

$D^4(3,1) = \min((3,1), (3,4)+(4,1))$
$= \min(1, 2+3)$
$= \min(1,5)$
$= 1$

$$\leftarrow (3,2) = \min((3,2), (3,4)+(4,2))$$
$$= \min(7, 2+3)$$
$$= \min(7,5)$$
$$= 5$$

WARSHALL's Algorithm :-

It is used to find the transtive closure of a graph. The transtive closure of a graph G is defined to be "a graph G' such that G' has the same nodes as G and there is an edge v_i, v_j in G' whereever there is a path from v_i to v_j in G"

The path matrix P of the graph G is the adjacency matrix of its transtive closure G'

The Recursive definition of transtive closure is,

$$t_{ij}^{0} = \begin{cases} 0 & \text{if } i \neq j \text{ and } (i,j) \notin E \\ 1 & \text{if } i=j \text{ (or) } (i,j) \in E \end{cases}$$

The reachability is called transtive closure of a graph

$$t_{ij}^{(k)} = t_{ij}^{(k-1)} \vee (t_{ik}^{(k-1)} \wedge t_{kj}^{(k-1)})$$

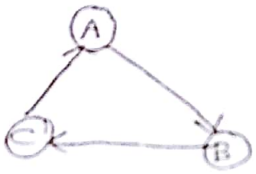
Transtive closure:- Algorithm

```
Transtive-closure(G)
{
 n ← v[G]
for i ← 1 to n
  do for  j ← 1 to n
  do if i=j (or) (i,j) ∈ E[G]
  ↵        , (0)
```

$t_{ij}^{(0)} \leftarrow 0$

for $k \leftarrow 1$ to n
do for $i \leftarrow 1$ to n
do for $j \leftarrow 1$ to n
do
$t_{ij}^{(k)} \leftarrow t_{ij}^{(k-1)} \vee (t_{ik}^{(k-1)} \wedge t_{kj}^{(k-1)})$

Q:-



The adjacency matrix for the above graph

$$T^0 = \begin{array}{c} \\ A \\ B \\ C \end{array} \begin{array}{c} \begin{array}{ccc} A & B & C \end{array} \\ \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{bmatrix} \end{array}$$

Step 1:- calculate T^1 by using T^0

At $k=1$, $i=1$ $j=1$

$T_{11}^1 \leftarrow T_{11}^0 \vee [T_{11}^0 \wedge T_{11}^0]$
$= 0 \vee [0 \wedge 0]$
$= 0$

$$T^1 = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 1 & 0 \end{bmatrix}$$

At $k=1$, $i=1$, $j=2$

$T_{12}^1 = T_{12}^0 \vee (T_{11}^0 \wedge T_{12}^0)$
$= 1 \vee [0 \wedge 1] = 1$

At $k=1$ $i=1$ $j=3$

$T_{13}^1 = T_{13}^0 \vee (T_{11}^0 \wedge T_{13}^0) = 0 \vee [0 \wedge 0] = 0$

$k=1,\ i=2,\ j=1$

$$T^1_{21} = T^0_{21} \vee [T^0_{21} \wedge T^0_{11}] = 0 \vee [0 \wedge 0] = 0$$

At $k=1,\ i=2,\ j=2$

$$T^1_{22} = T^0_{22} \vee (T^0_{21} \wedge T^0_{12})$$
$$= 0 \vee [0 \wedge 1] = 0$$

At $k=1,\ i=2,\ j=3$

$$T^1_{23} = T^0_{23} \vee (T^0_{21} \wedge T^0_{13})$$
$$= 1 \vee (0 \wedge 0) = 1 \vee 0 = 1$$

At $k=1,\ i=3,\ j=1$

$$T^1_{31} = T^0_{31} \vee (T^0_{31} \wedge T^0_{11})$$
$$= 1 \vee (1 \wedge 0)$$
$$= 1$$

At $k=1,\ i=3,\ j=2$

$$T^1_{32} = T^0_{32} \vee (T^0_{31} \wedge T^0_{12})$$
$$= 0 \vee [1 \wedge 1] = 1$$

At $k=1,\ i=3,\ j=3$

$$T^1_{33} = T^0_{33} \vee (T^0_{31} \wedge T^0_{13})$$
$$= 0 \vee (1 \wedge 0) = 0$$

Step2:- calculate T^2 by using T^1

At $k=2,\ i=1,\ j=1$

$$T^2_{11} = T^1_{11} \vee (T^1_{12} \wedge T^1_{21})$$
$$= 0 \vee (1 \wedge 0)$$
$$= 0 \vee 0 = 0$$

$$T^2 = \begin{bmatrix} 0 & 1 & 1 \\ 0 & 0 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

TutorialsDuniya.com

Download FREE Computer Science Notes, Programs, Projects, Books PDF for any university student of BCA, MCA, B.Sc, B.Tech CSE, M.Sc, M.Tech at https://www.tutorialsduniya.com

- Algorithms Notes
- Artificial Intelligence
- Android Programming
- C & C++ Programming
- Combinatorial Optimization
- Computer Graphics
- Computer Networks
- Computer System Architecture
- DBMS & SQL Notes
- Data Analysis & Visualization
- Data Mining
- Data Science
- Data Structures
- Deep Learning
- Digital Image Processing
- Discrete Mathematics
- Information Security
- Internet Technologies
- Java Programming
- JavaScript & jQuery
- Machine Learning
- Microprocessor
- Operating System
- Operational Research
- PHP Notes
- Python Programming
- R Programming
- Software Engineering
- System Programming
- Theory of Computation
- Unix Network Programming
- Web Design & Development

Please Share these Notes with your Friends as well

At $k=2, i=1, j=2$

$$T_{12}^2 = T_{12}^1 \vee (T_{12}^1 \wedge T_{22}^1)$$
$$= 1 \vee (1 \wedge 0)$$
$$= 1 \vee 0$$
$$= 1$$

At $k=2, i=1, j=3$

$$T_{13}^2 = T_{13}^1 \vee (T_{12}^1 \wedge T_{23}^1)$$
$$= 0 \vee (1 \wedge 1)$$
$$= 0 \vee (1)$$
$$= 1$$

At $k=2, i=2, j=1$

$$T_{21}^2 = T_{21}^1 \vee (T_{22}^1 \wedge T_{21}^1)$$
$$= 0 \vee (0 \wedge 0)$$
$$= 0 \vee 0 = 0$$

At $k=2, i=2, j=2$

$$T_{22}^2 = T_{22}^1 \vee (T_{22}^1 \wedge T_{22}^1)$$
$$= 0 \vee (0 \wedge 0)$$
$$= 0 \vee (0)$$
$$= 0$$

At $k=2, i=2, j=3$

$$T_{23}^2 = T_{23}^1 \vee (T_{22}^1 \wedge T_{23}^1)$$
$$= 1 \vee (0 \wedge 1)$$
$$= 1 \vee 0 = 1$$

, $i=3,\ j=1$

$= T_{31}^{1} \vee (T_{32}^{1} \wedge T_{21}^{1})$

$= 1 \vee (1 \wedge 0)$

$= 1 \vee 0 = 1$

$k=2,\ i=3,\ j=2$

$= T_{32}^{1} \vee (T_{32}^{1} \wedge T_{22}^{1})$

$= 1 \vee (1 \wedge 0)$

$= 1 \vee (0)$

$= 1$

$k=2,\ i=3,\ j=3$

$= T_{33}^{1} \vee (T_{32}^{1} \wedge T_{23}^{1})$

$= 0 \vee (1 \wedge 1)$

$= 0 \vee 1$

$= 1$

Ep 3:- calculate T^3 by using T^2

$k=3,\ i=1,\ j=1$

$= T_{11}^{2} \vee (T_{13}^{2} \wedge T_{31}^{2})$

$= 1 \vee (1 \wedge 1)$

$= 1 \vee 1$

$= 1$

$k=3,\ i=1,\ j=2$

$T_{12}^{3} = T_{12}^{2} \vee (T_{13}^{2} \wedge T_{32}^{2})$

$= 1 \vee (1 \wedge 1)$

$= 1$

$k=3,\ i=1,\ j=3$

$T^3 = T^2 \vee (T_{13}^{2} \wedge T_{33}^{2}) = 1 \vee (1 \wedge 1) = 1$

$$T^3 = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

At $k=3, i=2, j=1$

$$T_{21}^3 = T_{21}^2 \vee (T_{23}^2 \wedge T_{31}^2)$$
$$= 0 \vee (1 \wedge 1)$$
$$= 1$$

At $k=3, i=2, j=2$

$$T_{22}^3 = T_{22}^2 \vee (T_{23}^2 \wedge T_{32}^2)$$
$$= 0 \vee (1 \wedge 1)$$
$$= 1$$

At $k=3, i=2, j=3$

$$T_{23}^3 = T_{23}^2 \vee (T_{23}^2 \wedge T_{33}^2)$$
$$= 1 \vee (1 \wedge 1)$$
$$= 1$$

At $k=3, i=3, j=1$

$$T_{31}^3 = T_{31}^2 \vee (T_{33}^2 \wedge T_{31}^2)$$
$$= 1 \vee (1 \wedge 1)$$
$$= 1$$

At $k=3, i=3, j=2$

$$T_{32}^3 = T_{32}^2 \vee (T_{33}^2 \wedge T_{32}^2)$$
$$= 1 \vee (1 \wedge 1)$$
$$= 1$$

At $k=3, i=3, j=3$

$$T_{33}^3 = T_{33}^2 \vee (T_{33}^2 \wedge T_{33}^2)$$
$$= 1 \vee (1 \wedge 1)$$
$$= 1$$

$$T^3 = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

UNIT-6

Sorting

* A Sorting is a technique which is mainly used to arrange the given elements either in ascending (or) descending order.

* We have different type of sorting techniques such as bubble sort, heap sort, insertion sort, quick sort, merge sort, radic sort, selection sort etc....

Insertion Sort:- This sorting technique selects one element for each position from the given list of elements, and the selected element will be inserted into its proper position in the list.

Time complexity:-

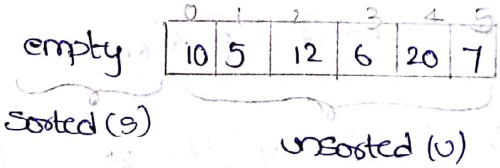
Best case is order of n - $O(n)$

Worst case is order of n^2 . $O(n^2)$

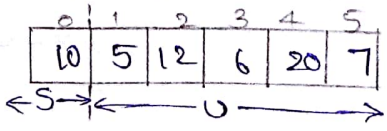
Procedure:-

Eg:- 10, 5, 12, 6, 20, 7

consider the above elements are in the unsorted portion of the list and the sorted portion is empty.

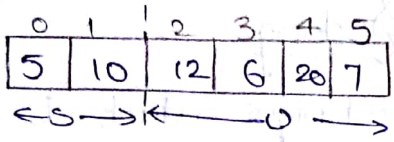


Pass(1):- move the first element 10 to the sorted portion of the list.

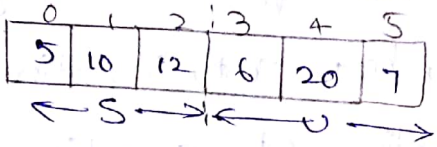


Pass(2):- move the second element 5 from unsorted portion to sorted portion by comparin

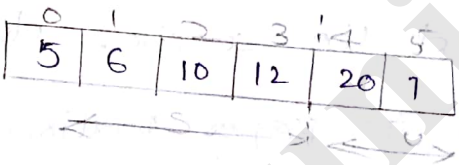
the existed elements in the sorted position.



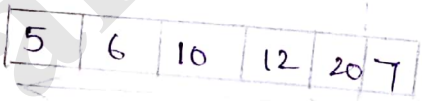
Pass(3):- move the third element 12 from unsorted portion to sorted portion by comparing the existed elements in the sorted portion



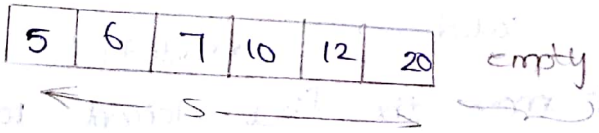
Pass(4):- move the fourth element 6 from unsorted portion to sorted portion by comparing the existed elements in the sorted portion.



Pass (5):- move the fifth element 20 from unsorted portion to sorted portion by comparing the existed elements in the sorted portion



Pass(6):- move the sixth element 7 from unsorted portion to sorted portion by comparing the existed elements in the sorted portion



```
#include <iostream.h>
#include <conio.h>
void main( )
{
 int a[50], n, i, j, temp;
 clrscr( );
 cout << "enter the no.of elements:";
 cin >>n;
 cout<<"\n enter the elements:";
 for (i=0; i<n; i++)
 {
  cin >> a[i];
 }
 for (i=1; i<n ; i++)
 {
  temp = a[i];
  j = i-1;
  while((j>=0) && (a[i]>temp))
  {
   a[j+1] =a[j];
   j--
  }
  a[j+1] = temp;
 }
 cout <<"\n the elements after insertion sort are:";
 for (i=0; i<n ; i++)
 {
  cout<<a[i]<<" ";
 }
 getch( );
}
```

Heap sort:-

* A heap sort is one of the best sorting technique to organize the elements either in ascending (or) descending order.

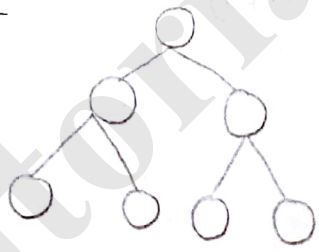
* Usually a heap is a tree based data structure which organises the elements in the form of tree manner.

* The heap follows two basic properties they are
  1. shape property
  2. heap property

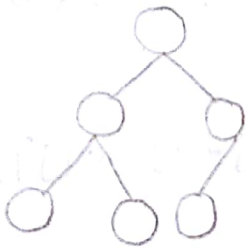
1. **Shape property:-** This property states that the heap should be either complete binary tree (or) almost complete binary tree.

Complete binary Tree:- In a binary tree, if all the nodes are fully filled in each level except the last level then such binary tree can be called as complete binary tree.

Ex:-



Complete binary Tree



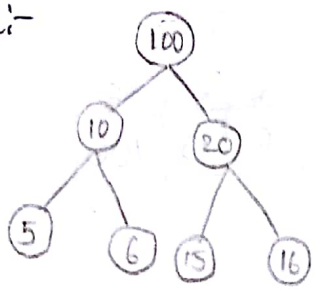
almost complete binary Tree

2. **heap property:-** In the complete binary tree are almost complete binary tree, if all the parent nodes should be greater than ~~equal to~~ their child nodes (or) all the parent nodes less than ~~equal to~~ their child nodes

* If all the parent nodes are greater than (or) equal to their child nodes then such heap can be called as Max-heap

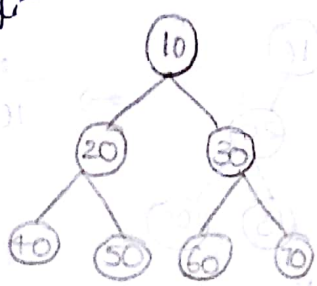
* If all the parent nodes are less than (or) equal to their child nodes then such heap can be called as Min-heap.

Eg:-



Max-heap

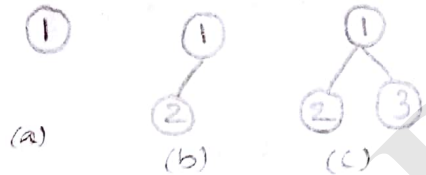
Eg:-



Min-heap

Binary Tree:- In a tree, each node contains 0 (or) 1 (or) 2 child nodes then such tree can be called binary tree.

Eg:-

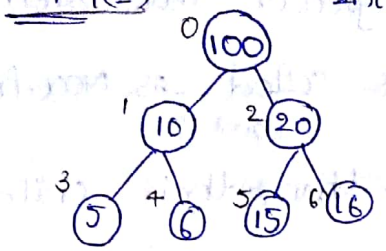


* The heap sort, first creates the heap either Max-heap (or) Min-heap with the given elements then it swaps the root element with the specific position element in the heap (or) array. and after the heap will be reconstructed either max-heap (or) min-heap depending upon the sorting order this process will be continued until all the elements are sorted

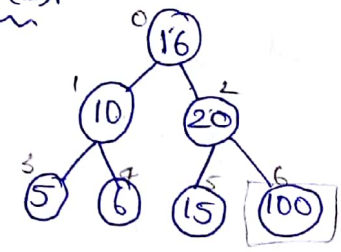
Eg:- 100, 10, 20, 5, 6, 15, 16

0	1	2	3	4	5	6
100	10	20	5	6	15	16

Step (1) Initial elements

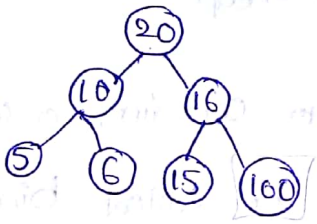


Step (1):-

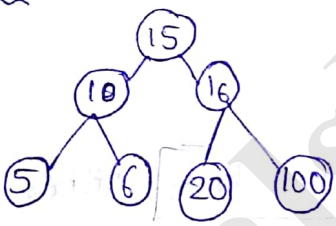


⇒

0	1	2	3	4	5	6
16	10	20	5	6	15	100

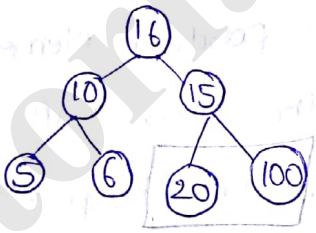


Step (2):-

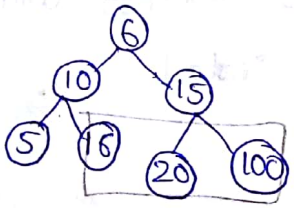


⇒

0	1	2	3	4	5	6
15	10	16	5	6	20	100

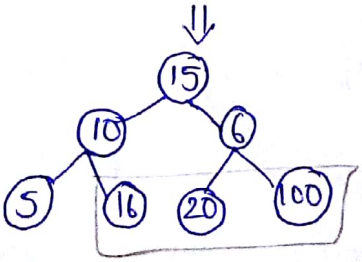


Step (3):-

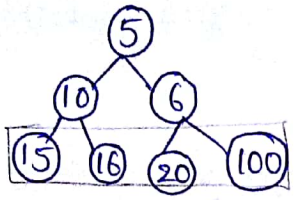


⇒

0	1	2	3	4	5	6
6	10	15	5	16	20	100

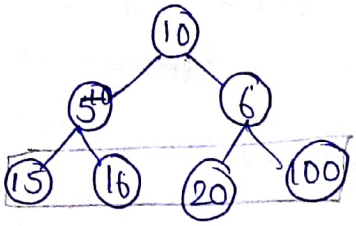


Step (4):-

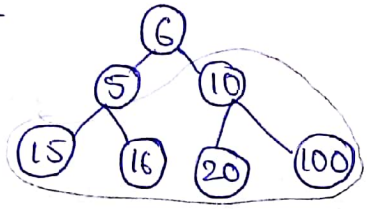


⇒

0	1	2	3	4	5	6
10	5	6	15	16	20	100



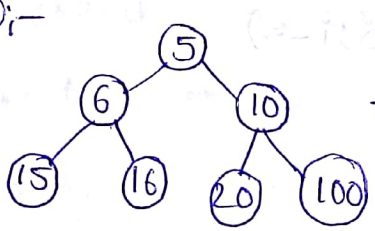
Step (5):-



⇒

0	1	2	3	4	5	6
6	5	10	15	16	20	100

Step (6):-



⇒

0	1	2	3	4	5	6
5	6	10	15	16	20	100

Program to implement heap sort:-

```
#include <iostream.h>
#include <conio.h>
 void  heapsort();
 void  heapadjust(int, int);
 int  a[50], n, i, temp;
  void  main()
    {
     clrscr();
     cout<<" Enter the no.of elements: ";
     cin >>n;
     cout<<"\n Enter the elements:";
      for(i=0; i<n;i++)
       {
        cin>>a[i];
```

```
}
heapsort();
cout<<"in the elements after heapsort are:";
for(i=0;i<n;i++)
{
cout<<a[i]<<" ";
}
getch();
}
void heapsort()
{
for(i=(n/2)-1;i>=0;i--)
{
 heapadjust(n,i);
 }
 for(i=n-1;i>=0;i--)
 {
 temp=a[i];
 a[i]=a[0];
 a[0]=temp;
 heapadjust(i,0);
 }
 }
void heapadjust(int n,int i)
 {
int large=i,left=2*i+1,right=2*i+2;
if((left<n)&&(a[left]>a[large]))
large=right;
```

```
        if(large != i)
        {
        temp = a[large];
        a[large] = a[i];
        a[i] = temp;
        heapadjust(n, large);
        }
    }
```

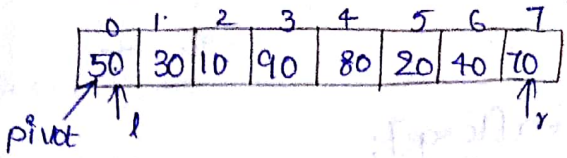
* quick Sort:-

* The quick sort is one of efficient sorting technique which uses divide and conquer approach in order to sort the given list of elements either in ascending (or) Descending order.
* In this technique the first element from the list of elements is selected as pivot element
* When the pivot element is placed at a respective position in the list then the elements which are lessthan pivot elements will be taken into left sub-Array and the elements which are greater than the pivot element will be taken into Right sub-array this procedure will be continued untill all the elements are placed at their respective positions.

Time complexity:-

1. while((pivot >= a[l]) && (l <= r)) ⇒ T
   l++;
2. while((pivot < a[r]) && (l <= r)) ⇒ T
   r--;
3. If (l < r) ⇒ T ⇒ Swap a[l] & a[r]
   F ⇒ Swap pivot & a[r]

Ex: 50, 30, 10, 90, 80, 20, 40, 70



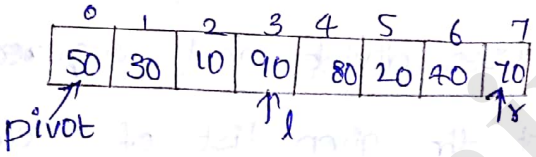
pivot >= a[l] && l<=r

50 >= 50 && 0<=7 } l++ l=1
T T

50 >= 30 && 1<=7 } l++ l=2
T T

50 >= 10 && 2<=7 } l++ l=3
T T

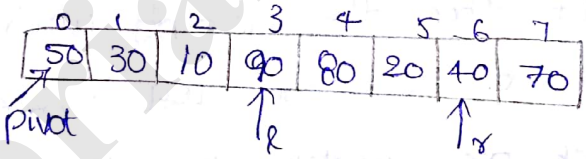
50 >= 90 && 3<=7 } stop incrementation l
F T



pivot < a[r] && l<=r

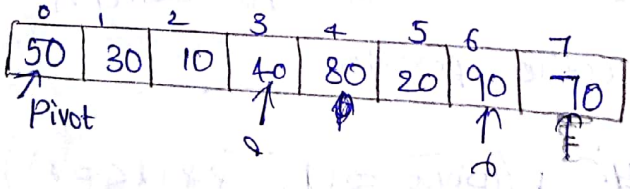
50 < 70 && 3<=7 } r-- r=6
T T

50 < 40 && 3<=6 } stop decrementing r
F T



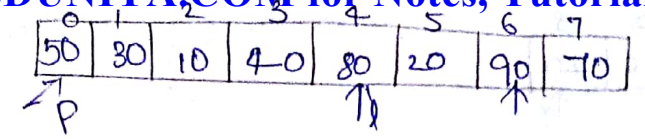
(l<r)

3<6 ⇒ T, swap a[l] & a[r]



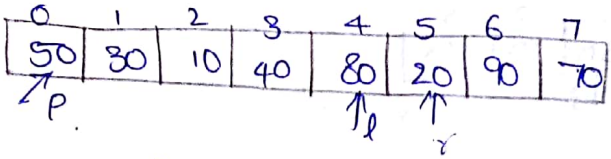
50 >= 40 && 3<=6 } l++ l=4
T T

50 >= 80 && 4<=6 } stop incrementing l
F T



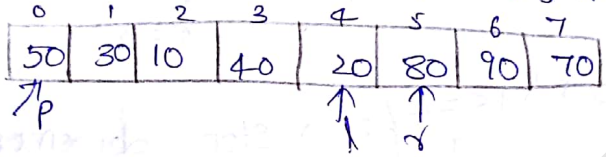
50<90 && 4<=6 } r--
T T r=5

50<20 && 4<=5 } F Stop decrementing r
F T



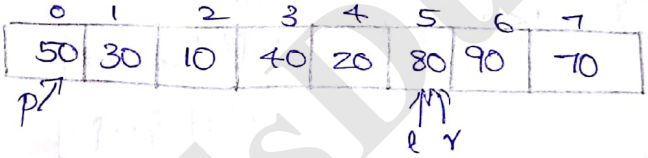
l<r

4<5 ⇒T, Swap a[l] & a[r]



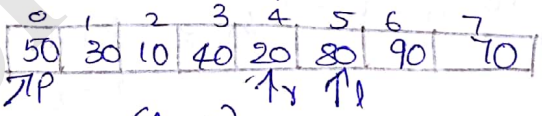
50>=20 && 4<=5 } l++
T T l=5

50>=80 && 5<=5 } Stop incrementing l
F T



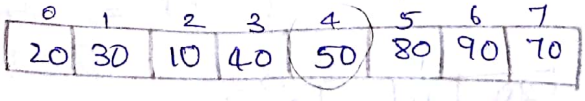
50<80 && 5<=5 } r--
T T r=4

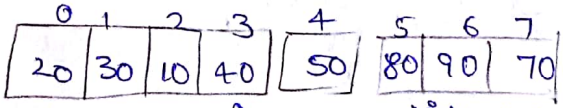
50<20 && 5<=4 } Stop decrementing r
F F



(l<r)

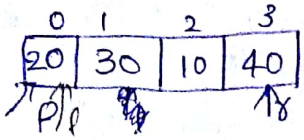
5<4 ⇒F Swap pivot &a[r]





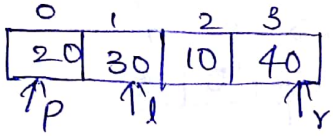
left sub-array

right sub-array



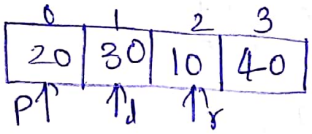
20 >= 20 && 0 <= 3 } l++
T T l = 1

20 >= 30 && 1 <= 3 } F ⇒ Stop incrementing l
F T

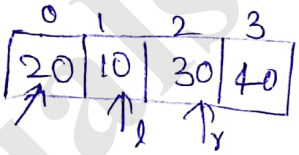


20 <= 40 && 1 <= 3 } T ⇒ r--
T T r = 2

20 < 10 && 1 <= 2 } F ⇒ Stop decrementing r
F T

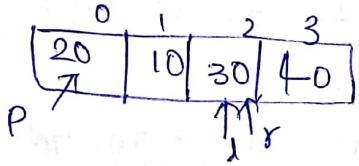


l < r 1 < 2 ⇒ T ⇒ Swap a[l] & a[r]
a[1] & a[2]



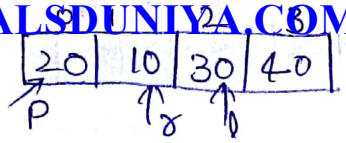
20 >= 10 && 1 <= 2 } T ⇒ l++
T T l = 2

20 >= 30 && 2 <= 2 } F ⇒ Stop incrementing l
F T



20 < 30 && 2 <= 2 } T ⇒ r--
T T r = 1

20 < 10 && 2 <= 1 } F ⇒ Stop decrementing r
F F

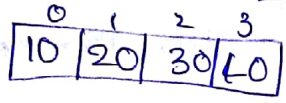


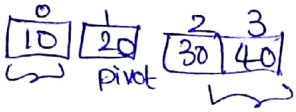
$l < r$

$2 < 1 \Rightarrow F \Rightarrow$ Swap

pivot & a[r]

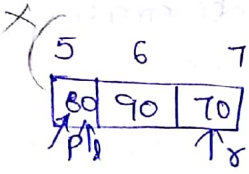
20 & a[10]





left sub-array

right sub-array



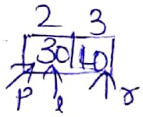
$(80 >= 80)$ && $5 <= 7$ } $T \Rightarrow l++$

T T

$l = 6$

$80 >= 90$ && $6 <= 7$ } $F \Rightarrow$ Stop incrementing l

F T



$30 >= 30$ && $2 <= 3$ } $T \Rightarrow l++$

T T

$l = 3$

$30 >= 40$ && $3 <= 3$ } $F \Rightarrow$ Stop incrementing l

F T

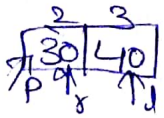
$30 < 40$ && $3 <= 3$ } $T \Rightarrow r--$

T T

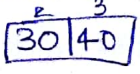
$r = 2$

$30 < 30$ && $3 <= 2$ } $F \Rightarrow$ Stop decrementing r

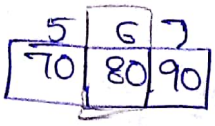
F F

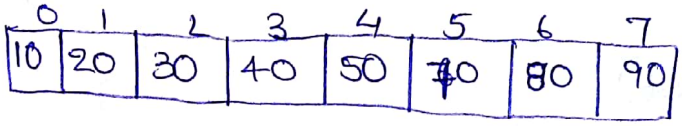


$l < r \Rightarrow 3 < 2 \Rightarrow$ F Swap $\Rightarrow$ pivot & a[r]



80 & 70





Program to implement quick sort:-

```
#include <iostream.h>
#include <conio.h>
void quicksort (int [ ], int, int);
int partition (int[ ], int, int);
void main( )
{
int a[50], n, i;
clrscr( );
cout<<"Enter the no.of elements: ";
cin>>n;
cout<<"\n Enter the elements:";
for (i=0; i<n; i++)
{
cin>>a[i];
}
quicksort(a, 0, n-1);
cout<<"\n The elements after quick sort are:";
for (i=0; i<n; i++)
{
cout<<a[i]<<" ";
}
```

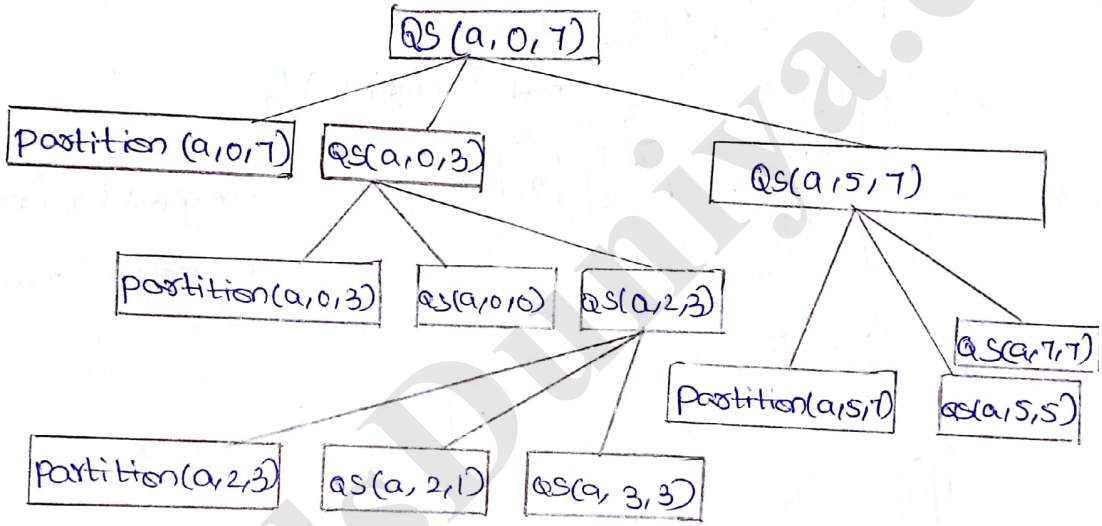
```
    getch();
}
void quicksort (int a[ ], int first, int last)
{
 int p;
 if (first < last)
 {
 p = partition (a, first, last);
 quicksort (a, first, p-1);
 quicksort (a, p+1, last);
 }
}
int partition (int a[ ], int l, int r)
{
 int pivot, i, j, temp;
 pivot = a[l];
 i = l;
 j = r;
 while(i<j)
 {
 while ((pivot >= a[i]) && (i <= j))
 i++;
 while ((pivot < a[j]) && (i <= j))
 j--;
 if (i<j)
 {
 temp = a[j];
 a[j] = a[i];
```

declaration

declaration

```
        a[i]=temp;
        }
    else
        {
        a[l] = a[j];
        a[j] = pivot;
        }
    }
    return j;
}
```

Tree structure for Quick Sort:-



Merge Sort:- The Merge sort is one of the efficient sorting technique which also follows Divide & conquer approach to sort the given list of elements into either ascending (or) Descending order

* In this technique the list of elements is divided into two parts and each sub-list is again divided into two sub-parts untill each sub-list contains single element. This process is known as Divide process.
* After dividing each sub-list containing single element then all the sub-lists will be conquered (or)

merged. this process is known as conquer process

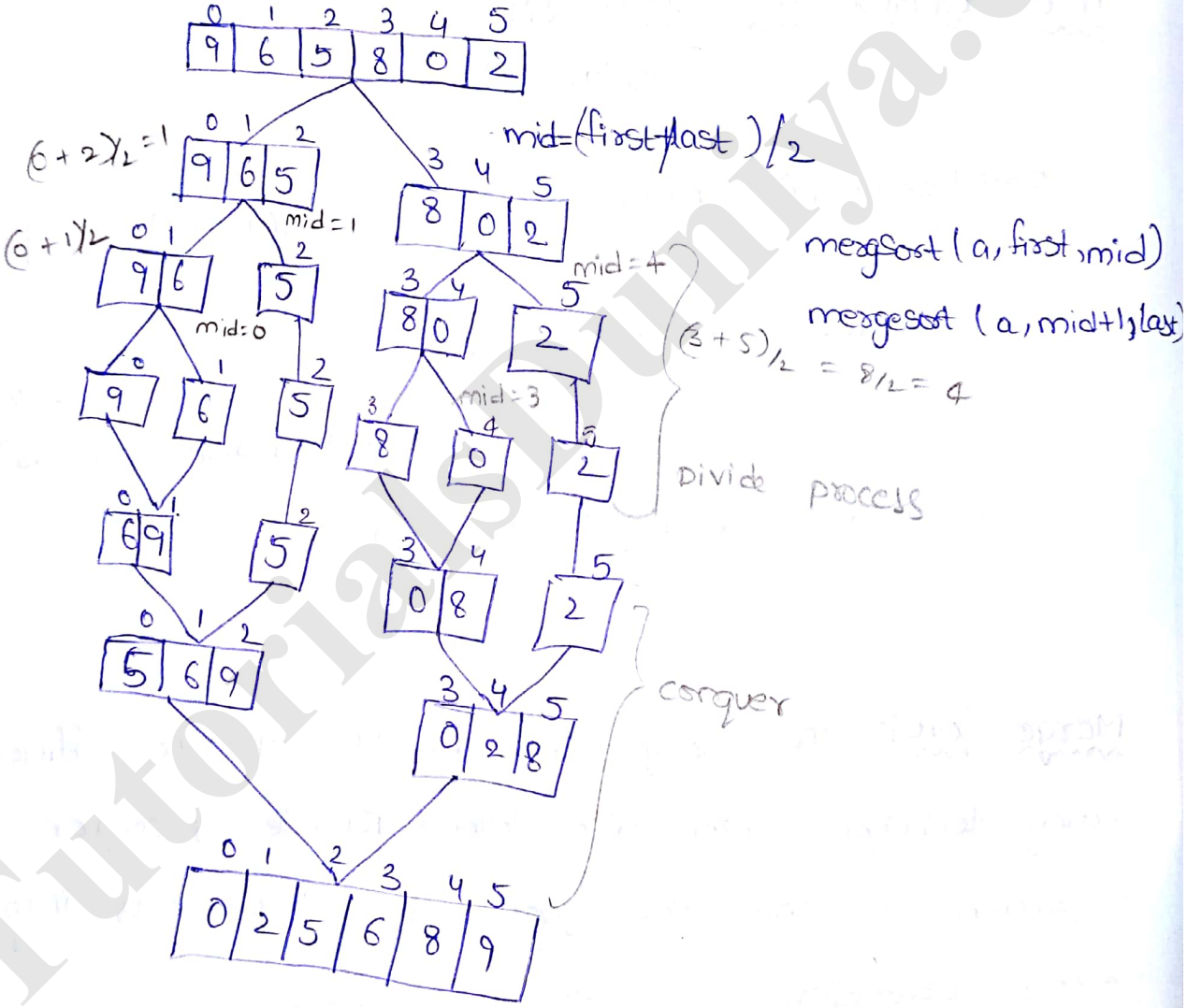
this process will be continued untill all the elements in the list are sorted.

Time complexity:-

For best case is $O(n \log(n))$

Worst case is $O(n \log(n))$

Eg:- 9, 6, 5, 8, 0, 2



Program to implement merge Sort :-

```
#include <iostream.h>
#include <conio.h>
void mergesort (int[ ], int, int);
void merge (int[ ], int, int, int);
int a[50], b[50];
void main( )
{
int n, i;
clrscr( );
cout<<"Enter the no.of elements:";
cin>>n;
cout<<"\n Enter the elements:";
for(i=0; i<n; i++)
{
cin>> a[i];
}
mergesort(a,0, n-1)
cout<<"\n The elements after merge Sort are:";
for(i=0; i<n; i++)
{
cout<< a[i]<<" ";
}
getch( );
}
void mergesort (int a[ ], int first, int last)
{
int mid;
```

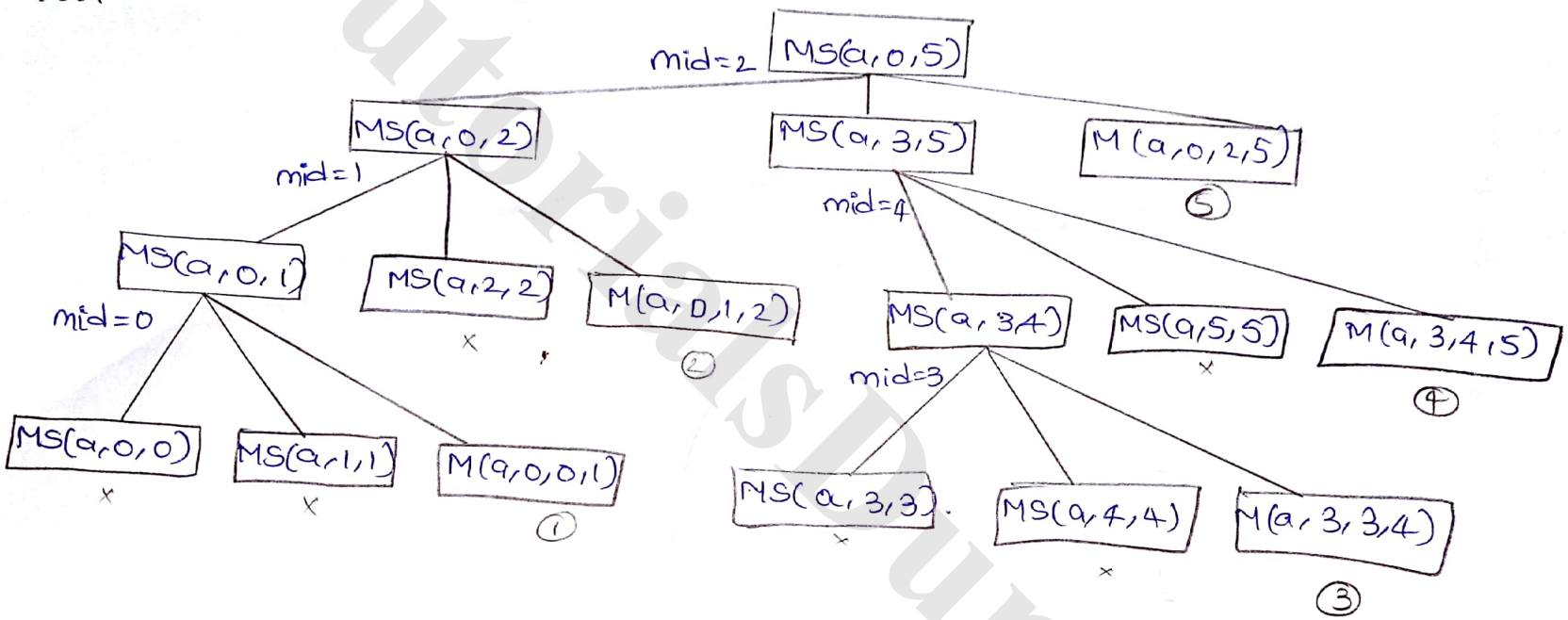
declaration

definition

```
if (first < last)
  {
mid = (first + last)/2 ;
mergesort (a, first, mid);
mergesort (a, mid+1, last);
merge(a, first, mid, last);
  }
}

Void  merge(int a[ ], int fst, int mid, int lst)
{
int f, i, j;
f1 = fst;
i = fst;
j = mid+1;
while ((f1 <= mid) && (j <= lst))
  {
  if (a[f1] < a[j])
  {
  b[i] = a[f1];
  f1++;
  }
else
  {
  b[i] = a[j];
  j++;
  }
  i++;
}
while (f1 <= mid)
{
```

```
b[i] = a[f1];
f1++;
i++;
}
while(j<=lst)
{
b[i] = a[j];
j++;
i++;
}
for(i=fst; i<=lst; i++)
{
a[i] = b[i];
}
}
```

Tree structure for Merge Sort :-



Iterative Merge Sort:

program to implement merge sort using iterative loop:-

```
#include<iostream.h>
#include <conio.h>
 int  min(int, int);
 Void  mergesort (int [ ], int, int);
 void  merge (int [ ], int, int);
 int  a[50], b[50];
 Void  main( )
 {
  int  n, i;
  clrscr( );
  cout<<"Enter the no.of elements:";
  cin>>n;
  cout <<"\n Enter the elements:";
  for(i=0; i<n; i++)
  {
   cin >> a[i];
  }
  mergesort (a, 0, n-1);
  cout<<"\n the elements after merge sort are:";
  for( i=0; i<n; i++)
  {
   cout << a[i]<<"   ";
   }
   getch( );
 }
  int   min(int  x, int  y)
  {
   return(x<y) ? x:y;
```

```
}
void mergesort(int a[], int low, int high)
{
 int m, k, first, mid, last;
 for (m=1; m<=high; m= 2*m)
 {
 for (k=low; k<high; k =k+(2*m))
 {
 first =k;
 mid = k+m-1;
 last =min (k+ (2*m) -1, high)
 merge (a, first, mid, last);
   }
 }
}
void merge (int a[], int fst, int mid, int lst)
{
 int f1, i, j;
 f1 =fst;
 i = fst;
 j = mid+1;
 while((f1 <=mid) && (j <=lst))
 {
  if (a[f1] <a[j])
  {
   b[i] = a[f1];
   f1++;
  }
 else
  {
  b[i] = a[j];
  j++;
```

```
    }
    i++;
  }
  while (f1<=mid)
  {
  b[i]= a[f1];
    f1++;
    i++;
  }
  while(j<=lst)
  {
    b[i]=a[j];
    j++;
    i++;
  }
   for (i=fst; i<=lst; i++)
     {
      a[i] = b[i];
     }
   }
```

TutorialsDuniya.com

Download FREE Computer Science Notes, Programs, Projects, Books PDF for any university student of BCA, MCA, B.Sc, B.Tech CSE, M.Sc, M.Tech at https://www.tutorialsduniya.com

- Algorithms Notes
- Artificial Intelligence
- Android Programming
- C & C++ Programming
- Combinatorial Optimization
- Computer Graphics
- Computer Networks
- Computer System Architecture
- DBMS & SQL Notes
- Data Analysis & Visualization
- Data Mining
- Data Science
- Data Structures
- Deep Learning
- Digital Image Processing
- Discrete Mathematics
- Information Security
- Internet Technologies
- Java Programming
- JavaScript & jQuery
- Machine Learning
- Microprocessor
- Operating System
- Operational Research
- PHP Notes
- Python Programming
- R Programming
- Software Engineering
- System Programming
- Theory of Computation
- Unix Network Programming
- Web Design & Development

Please Share these Notes with your Friends as well