**Design And Analysis Of Algorithms**

**Practicals**

**LIST OF PRACTICALS**

1. i. Implement **Insertion Sort** (The program should report the number of comparisons)

ii. Implement **Merge Sort** (The program should report the number of comparisons)

2. Implement **Heap Sort** (The program should report the number of comparisons)

3. Implement **Randomized Quick sort** (The program should report the number of comparisons)

4. Implement **Radix Sort**

5. Create a **Red-Black Tree** and perform following operations on it:

i. Insert a node

ii. Delete a node

iii. Search for a number & also report the color of the node containing this number.

6. Write a program to determine the **LCS of two given sequences**

7. Implement **Breadth-First Search** in a graph

8. Implement **Depth-First Search** in a graph

9. Write a program to determine the **minimum spanning tree** of a graph .

For the algorithms at S.No 1 to 3 test run the algorithm on 100 different inputs of sizes varying from 30 to 1000. Count the number of comparisons and draw the graph. Compare it with a graph of nlogn.

For the algorithms at S.No 4 test run the algorithm on 100 different inputs of sizes varying from 30 to 1000. Count the number of operations and draw the graph.

For plotting the above mentioned graphs, on x axis plot different values of n(input size) and on y axis plot corresponding number of comparisons.

1. **Implement Insertion Sort (The program should report the number of comparisons)**
2. **ii. Implement Merge Sort(The program should report the number of comparisons)**

**I)**

**// Insertion Sort**

#include<iostream.h>

#include<conio.h>

#include<stdlib.h>

#include<fstream.h>

float inSort(int a[1000],int n)

{

float s = 0;

for(int i=1;i < n;i++)

{

int value = a[i];

int hole = i;

while(hole > 0 && a[hole-1] > value)

{

a[hole] = a[hole-1];

hole = hole-1;

s++;

}

a[hole] = value;

}

return s;

}

void main()

{

clrscr();

ofstream fout;

fout.open("insert.txt");

char ch;

int a[1000],c,i;

do

{

clrscr();

cout<<"\n---------------MENU----------------\n\n";

cout<<"Press 1 for automatic\n";

cout<<"Press 2 for manual ";

cin>>c;

switch(c)

{

case 1:

for(i=30;i<1000;i+=9)

{

cout<<"\n\nNo. of element in the array: "<<i;

fout<<"\n\nNo. of elemrnts in the array: "<<i;

for(int j=0;j<i;j++)

a[j]=rand()%1000;

float s = inSort(a,i);

cout<<"\nThe number of comparisons are: "<<s;

fout<<"\nThe number of comaprisons are: "<<s;

}

break;

case 2: int \*b,n;

cout<<"Enter the size of array :";

cin>>n;

b = new int[n];

cout<<"Enter the elements ";

for(int i=0;i<n;i++)

cin>>b[i];

inSort(b,n);

cout<<"\nSorted array is :";

for(i=0;i<n;i++)

cout<<b[i]<<" ";

break;

default : cout<<"Wrong Choice\n\n";

}

cout<<"\n\nDo u want to continue : ";

cin>>ch;

}while (ch=='y'||ch=='Y');

fout.close();

getch();

}

**II)**

**// Merge sort**

#include<iostream.h>

#include<conio.h>

#include<stdlib.h>

#include<fstream.h>

#define inf 999

int comp = 0;

void merge(int A[1000], int p, int q, int r)

{

int n1= q-p+1;

int n2 = r-q;

int L[1000] , R[1000];

for(int i = 1 ; i <= n1 ; i++)

L[i] = A[p+i-1] ;

for ( int j = 1 ; j <= n2 ; j++)

R[j] = A[q+j];

L[n1+1] = inf;

R[n2+1] = inf;

i = j = 1;

for (int k = p ; k <= r ; k++) // loop for combining two sorted arrays

{

comp++;

if ( L[i] <= R[j] )

{

A[k] = L[i];

i = i+1;

}

else

{

A[k] = R[j];

j = j+1 ;

}

}

} // end of merge

void mergesort(int A[1000], int p, int r)

{

int q;

if (p < r)

{

q = (p+r) / 2 ;

mergesort (A,p,q);

mergesort (A,q+1,r);

merge (A,p,q,r);

}

} // end of mergesort function

void main()

{

clrscr();

ofstream fout;

fout.open("merged.txt");

char Ch;

int A[1000] , n ,ch;

do

{

clrscr();

cout<<"----------MENU---------\n\n";

cout<<"Press 1 for automatic\n";

cout<<"Press 2 for manual\n";

cout<<"Enter your choice ";

cin>>ch;

switch(ch)

{

case 1 :

for(int i=30;i<1000;i+=9)

{

cout<<"\n\nNo. of element in the array:"<<i;

fout<<"\n\nNo of element in the array: "<<i;

for(int j=0;j<i;j++)

A[j]=rand()%1000;

mergesort(A,1,i);

cout<<"\nThe no. of comparision : "<<comp;

fout<<"\nThe no. of comparision : "<<comp;

comp = 0;

}

break;

case 2 : cout<<"\nEnter the no. of elements in the array : ";

cin>>n;

cout<<"\nEnter the elements in the array :";

for (i = 1 ; i < n+1 ; i++)

cin>>A[i];

mergesort (A, 1 , n);

cout<<"\nThe array after sorting :";

for ( i = 1 ; i < n+1 ; i++)

cout<<" " <<A[i];

cout<<"\nThe number of comparisions are : "<<comp;

comp = 0;

break;

default : cout<<"You Pressed a wrong keyword..\n";

}

cout<<"\n\n\nDo you want to continue ? ";

cin>>Ch;

}while(Ch=='y'||Ch=='Y');

fout.close();

getch();

}

**2) Implement Heap Sort (The program should report the number of comparisons)**

#include<iostream.h>

#include<conio.h>

#include<stdlib.h>

#include<fstream.h>

int comp = 0;

int left(int i) // for getting left child

{

return 2\*i;

}

int right (int i) // for getting right child

{

return 2\*i + 1;

}

void max\_heapify ( int A[1000] , int i , int n )

{

int l , r , max;

l = left(i);

r = right(i);

comp++;

if ( l <= n && A[l] > A[i] ) // comparing left child by its parent

max = l;

else

max = i;

comp++;

if ( r <= n && A[r] > A[max] )

max = r;

if ( max != i )

{

int temp = A[i];

A[i] = A[max];

A[max] = temp;

max\_heapify ( A, max, n);

}

} // end of max\_heapify

void build\_max\_heap (int A[1000] , int n)

{

for (int i = n/2 ; i >= 1 ; i--)

max\_heapify (A, i, n);

}

void heapsort (int A[1000] , int n)

{

build\_max\_heap ( A, n );

for(int i = n ; i >= 2 ; i--)

{

int temp = A[i];

A[i] = A[1] ;

A[1] = temp;

n = n-1;

max\_heapify (A,1,n);

}

} // end of heapsort function

void main()

{

clrscr();

ofstream fout;

fout.open("heap.txt");

char Ch;

int A[1000] , n,ch;

do

{

clrscr();

cout<<"----------MENU---------\n\n";

cout<<"Press 1 for automatic\n";

cout<<"Press 2 for manual\n";

cout<<"Enter your choice ";

cin>>ch;

switch(ch)

{

case 1 :

for(int i=30;i<1000;i+=9)

{

cout<<"\n\nNo. of elements in the array: "<<i;

fout<<"\n\nNo. of elements in the array: "<<i;

for(int j=0;j<i;j++)

A[j]=rand()%1000;

heapsort(A,i);

cout<<"\nThe no. of comparision : "<<comp;

fout<<"\nThe no of comparision : "<<comp;

comp = 0;

}

break;

case 2: cout<<"\nEnter the no. of elements in the array :";

cin>>n;

cout<<"\nEnter the elements of the array :";

for(i = 1 ; i < n+1 ; i++)

cin>>A[i];

heapsort(A, n);

cout<<"\nThe array after sorting :";

for( i = 1 ; i < n+1 ; i++)

cout<<" "<<A[i];

cout<<"\nThe no of comparisions are "<<comp;

comp=0;

break;

default : cout<<"You Pressed wrong Keyword..!!!\n";

}

cout<<"\n\n\nWant to continue ? ";

cin>>Ch;

}while(Ch=='y'||Ch=='Y');

fout.close();

getch();

}

**3) Implement Randomized Quick sort (The program should report the number of comparisons)**

#include<iostream.h>

#include<stdlib.h>

#include<conio.h>

#include<stdio.h>

#include<fstream.h>

int comp = 0;

int partition(int A[1000], int p , int r)

{

int temp , x = A[r];

int i = p-1 ;

for(int j = p ; j < r ; j++ )

{

comp++;

if ( A[j] <= x )

{

i = i+1 ;

temp = A[j];

A[j] = A[i];

A[i] = temp;

}

}

temp = A[i+1] ;

A[i+1] = A[r] ;

A[r] = temp ;

return i+1;

} // end of partition function

int rand\_partition( int A[1000] , int p , int r)

{

int i = rand()%(r-p)+p;

int t = A[r];

A[r] = A[i];

A[i] = t;

return partition(A, p , r);

}

void quicksort(int A[1000],int p,int r)

{

if (p < r)

{

int q = rand\_partition(A,p,r);

quicksort(A,p,q-1);

quicksort(A,q+1,r);

}

}

// end of quicksort function

void main()

{

clrscr();

ofstream fout;

fout.open("quick.txt");

char Ch;

int A[1000],n,ch;

do

{

clrscr();

cout<<"----------MENU---------\n\n";

cout<<"Press 1 for automatic\n";

cout<<"Press 2 for manual\n";

cout<<"Enter your choice ";

cin>>ch;

switch(ch)

{

case 1 :

for(int i=30;i<1000;i+=9)

{

cout<<"\n\nNo. of element in the array: "<<i;

fout<<"\n\nNo. of element in the array: "<<i;

for(int j = 0;j < i;j++)

A[j] = rand()%1000;

quicksort(A,1,i);

cout<<"\nThe no. of comparision : "<<comp;

fout<<"\nThe no. of comparision : "<<comp;

comp = 0;

}

break;

case 2:

cout<<"\nEnter the no. of elements in the array : ";

cin>>n;

cout<<"\nEnter the elements of the array : ";

for(i = 1 ; i < n+1 ; i++ )

cin>>A[i];

quicksort(A,1,n);

cout<<"\nThe array after sorting : ";

for (i = 1 ; i < n+1 ; i++)

cout <<" "<<A[i];

cout<<"\nNumber of comparisions are : "<<comp;

comp = 0;

break;

default: cout<<"Pressed a wrong keyword...!!!";

}

cout<<"\n\n\nWant to continue (y/n) ? ";

cin>>Ch;

}while(Ch=='y'||Ch=='Y');

fout.close();

getch();

}

**4) Implement Radix Sort**

#include<iostream.h>

#include<conio.h>

#include<stdlib.h>

#include<fstream.h>

int comp = 0; //for no of comparisons

int nod(int k) // for no of digits

{

int nd = 0;

for(int i = 0;i < 10;i++)

{

k = k/10;

nd++;

if(k!=0)

continue;

else

break;

}

return nd;

}

int max(int a[10000],int n)

{

int max = a[1];

for(int i = 2;i <= n; i++)

{

if(max < a[i])

max=a[i];

}

return max;

} // end of max function

void count\_sort(int a[10000],int x[10000],int c[10000],int k,int n)

{

int arr[10000],aux[10000],b[10000];

for(int i = 0;i <= k;i++)

aux[i]=0;

for(int j = 1;j<=n;j++)

aux[x[j]]=aux[x[j]]+1;

for(i = 1;i<=k;i++)

aux[i]=aux[i]+aux[i-1];

for(j =n;j>=1;j--)

{

b[aux[x[j]]] = a[j];

arr[aux[x[j]]] = c[j];

aux[x[j]] = aux[x[j]]-1;

}

for(i = 1;i <=n;i++)

{

a[i]=b[i];

c[i]=arr[i];

}

} // end of count\_sort

void radix(int a[1000],int n)

{

int x[10000],c[10000];

int k = max(a,n);

int A = nod(k);

for(int i = 1;i <= n;i++)

c[i]=a[i];

for(i = 1;i <= A;i++) // loop for extracting digit

{

for(int j = 1;j <= n;j++)

{

if(c[j] != 0)

{

x[j] = c[j] % 10; // for unit digit extracting

c[j] /= 10;

}

else

x[j] = 0;

}

count\_sort(a,x,c,k,n);

}

}// end of radix sort

void main()

{

clrscr();

ofstream fout;

fout.open("radix.txt");

char CH;

int a[10000],ch,n;

do{

clrscr();

cout<<"\*\*\*\*\*\*MENU\*\*\*\*\*\*\*\n\n";

cout<<"1.For Automatic \n";

cout<<"2.For Manual \n";

cout<<"Enter your choice : ";

cin>>ch;

switch(ch)

{

case 1:

for(int i=30;i<1000;i+=9)

{

cout<<"\n\nNo. of elements in the array: "<<i;

fout<<"\n\nNo. of elements in the array: "<<i;

for(int j=0;j<i;j++)

a[j]=rand()%1000;

radix(a,i);

cout<<"\nThe no. of comparision : "<<comp;

fout<<"\nThe no of comparision : "<<comp;

comp = 0;

}

break;

case 2:

cout<<"\nEnter the no of elements in the array : ";

cin>>n;

cout<<"\nEnter elements : ";

for(i =1;i <= n;i++)

cin>>a[i];

radix(a,n);

cout<<"\nSorted array is : ";

for(i = 1;i <= n;i++)

cout<<a[i]<<" ";

break;

default : cout<<"You entered wrong Keyword ...!!!\n";

}

cout<<"\n\nDo u want to continue (y/n) or (Y/N) ?";

cin>>CH;

}while(CH=='Y'||CH=='y');

fout.close();

getch();

} // end of main

**5) Create a Red-Black Tree and perform following operations on it:**

**i. Insert a node**

**ii. Delete a node**

**iii. Search for a number & also report the color of the node containing this number.**

#include<stdio.h>

#include<iostream.h>

#include<conio.h>

template<class T>

class RBTNode{

T info;

RBTNode<T> \*left, \*right, \*p;

char color;

public:

RBTNode(){ //for nil

color='b';

info=-1 ;

left=right=p=0;

}

RBTNode(T info, RBTNode<T> \*p, RBTNode<T>\* left,\

RBTNode<T> \*right){

this->info=info;

this->right=right;

this->left=left;

this->color='r';

this->p=p;

}

~RBTNode(){

delete left, right;/\*take care always delete a leaf node, else this

destructor ensures the entire subtree is deleted\*/

}

/\*could implement getters and setters here or since RBT is only class

that would use the member variables, I'm using friend class concept\*/

friend class RBT<T>;

}

template<class T>

class RBT

{

RBTNode<T> \*root, \*nil;

void rbtInsertFixup(RBTNode<T>\*);

void rbtDeleteFixup(RBTNode<T>\*);

void left\_rotate(RBTNode<T>\*);

void right\_rotate(RBTNode<T>\*);

void DLS(RBTNode<T>\*,int);

int getmaxheight(RBTNode<T>\*);

public:

RBT(){

root=nil=new RBTNode<T>();

}

~RBT(){

delete root;//deletes entire tree via ~RBTNode()

delete nil;

}

void insert(T);

void findAndDeleteKey(T);

void deleteRBTNode(RBTNode<T> \*);

void IDDFS();

};

template <class T>

void RBT<T>::left\_rotate(RBTNode<T>\* x){

RBTNode<T>\* y=x->right;

x->right=y->left;

y->left->p=x;

y->left=x;

y->p=x->p;

if(root==x)

root=y;

else if(x==x->p->left)

y->p->left=y;

else

y->p->right=y;

x->p=y;

}

template <class T>

void RBT<T>::right\_rotate(RBTNode<T>\* x){

RBTNode<T>\* y=x->left;

x->left=y->right;

y->right->p=x;

y->right=x;

y->p=x->p;

if(root==x)

root=y;

else if(x==x->p->left)

y->p->left=y;

else

y->p->right=y;

x->p=y;

}

template <class T>

void RBT<T>::insert(T info){

RBTNode<T> \* prt=nil;//parent node

RBTNode<T> \*x=root;

while(x!=nil){

prt=x;

if(info<x->info)

x=x->left;

else

x=x->right;

}

RBTNode<T>\* z=new RBTNode<T>(info,prt, nil, nil);

if(prt==nil)

root=z;

else if(info>prt->info)

prt->right=z;

else

prt->left=z;

rbtInsertFixup(z);

cout<<endl;

}

template <class T>

void RBT<T>::rbtInsertFixup(RBTNode<T> \*z){

while(z->p->color=='r'){

if(z->p==z->p->p->left){

RBTNode<T> \*u= z->p->p->right;

if(u->color=='r'){//case 1

z->p->color='b';

u->color='b';

z->p->p->color='r';

z=z->p->p;

}//now u is black

else{

if(z==z->p->right){//case 2

z=z->p;

left\_rotate(z);

}

//case 3

right\_rotate(z->p->p);

z->p->color='b';

z->p->right->color='r';

}

}

else{

RBTNode<T> \*u= z->p->p->left;

if(u->color=='r'){//case 1

z->p->color='b';

u->color='b';

z->p->p->color='r';

z=z->p->p;

}

else {

//case 2

if(z==z->p->left){

z=z->p;

right\_rotate(z);

}

//case 3

left\_rotate(z->p->p);

z->p->color='b';

z->p->left->color='r';

}

}

}//end of while

root->color='b';

}

template <class T>

void RBT<T>::findAndDeleteKey(T key){

RBTNode<T> \*z=root;

while(z!=nil&& key!=z->info){

if(key>z->info)

z=z->right;

else

z=z->left;

}

if(z==nil)

cout<<"\nCan't Delete: Key not found!";

else

deleteRBTNode(z);

}

template <class T>

void RBT<T>::deleteRBTNode(RBTNode<T> \*z){

//delete by copy

RBTNode<T> \*y,\*x;

if(z->left==nil||z->right==nil)

y=z;

else{ //z has both children

//y must point to successor whose value is to be copied in z

y=z->right; //goto right subtree

while(y->left)

y=y->left; //goto leftmost node of this right subtree

//now y is a node with one child or none: z itself or successor

z->info=y->info;//copy in z (delete key of z) then delete duplicate y

}

if(y->left!=nil)

x=y->left;

else

x=y->right;

//x may be nil now

//splice out y by making x supplant y

x->p=y->p;

if(y==y->p->right)

y->p->right=x;

if(y==y->p->left)

y->p->left=x;

if(root==y)

root=x;

if(y->color=='b')

rbtDeleteFixup(x);

y->right=y->left=NULL;//to prevent deletion of underlying children via destructor

delete y;

}

template <class T>

void RBT<T>::rbtDeleteFixup(RBTNode<T>\*x){

while(x!=root &&x->color=='b'){

//first 4 set of cases

if(x==x->p->left){

RBTNode<T> \* w=x->p->right;

//case 1

if(w->color=='r'){

w->color='b';

w->p->color='r';

left\_rotate(w->p);

w=x->p->right;

}//now the sibling is black

//case 2 both children of sibling black

if(w->right->color=='b'&& w->left->color=='b'){

w->color='r';

x=x->p;

}

else{

//case 3

if(w->right->color=='b'){

w->left->color='b';

w->color='r';

right\_rotate(w);

w=w->p;

}

//case 4

w->color=x->p->color;

x->p->color='b';

w->right->color='b';

left\_rotate(x->p);

x=root;

}

}//1st 4 set complete

else{

RBTNode<T> \* w=x->p->left;

//case 1

if(w->color=='r'){

w->color='b';

w->p->color='r';

right\_rotate(w->p);

w=x->p->left;

}//now the sibling is black

//case 2

if(w->left->color=='b'&&w->right->color=='b'){

w->color='r';

x=x->p;

}

//case 3

else

{

if(w->left->color=='b'){

w->right->color='b';

w->color='r';

left\_rotate(w);

w=w->p;

}

//case 4

w->color=x->p->color;

x->p->color='b';

w->left->color='b';

right\_rotate(x->p);

x=root;

}

}//2nd set complete

}//end of while

x->color='b';

}

template<class T>

void RBT<T>::IDDFS(){

int maxheight=getmaxheight(root);

cout<<"maxheight="<<maxheight<<endl;

for(int i = 0; i<= maxheight+1; i++){

DLS(root,i);

cout<<endl;

}

}

template<class T>

void RBT<T>::DLS(RBTNode<T> \*node,int depth){

textcolor(node->color=='r'?4:15);

if (node!= nil)

if (depth == 0)

cprintf("%d",node->info);

else{

DLS(node->left, depth - 1);

DLS(node->right, depth - 1);

}

else

cprintf("%c",'N');

}

template<class T>

int RBT<T>::getmaxheight(RBTNode<T> \* r){

if(r==nil)

return -1;

else {

int left\_height=getmaxheight(r->left);

int right\_height=getmaxheight(r->right);

return left\_height>right\_height?left\_height+1:right\_height+1;

}

}

void main(){

clrscr();

RBT<int> t;

t.insert(9);

t.insert(8);

t.insert(7);

t.insert(5);

t.IDDFS();

t.insert(6);

t.insert(4);

t.IDDFS();

getch();

t.findAndDeleteKey(4);

t.IDDFS();

getch();

t.findAndDeleteKey(5);

t.IDDFS();

t.findAndDeleteKey(6);

t.IDDFS();

getch();

t.findAndDeleteKey(7);

t.IDDFS();

getch();

t.findAndDeleteKey(8);

t.IDDFS();

getch();

t.findAndDeleteKey(9);

cout<<"\ntree after deletion";

t.IDDFS();

getch();

}

**6) Write a program to determine the LCS of two given sequences**

#include<stdio.h>

#include<iostream.h>

#include<conio.h>

#include<string.h>

#define maxn 100

int \*\*c;

char \*\*p;

int Slength,Tlength;

int plcs(char \*\*b,char x[],int i,int j)

{

if(i==0||j==0)

return 0;

if(b[i][j]=='\^')

{plcs(b,x,i-1,j-1) ;

cout<<x[i];

}

else

{if(b[i][j]=='|')

plcs(b,x,i-1,j);

else {plcs(b,x,i,j-1);}

}

}

void LongestcSubsequence(char S[],char T[])

{

Slength = strlen(S);

Tlength = strlen(T);

int i,j;

for(i=Slength;i>=1;i--)

{

S[i] = S[i-1];

}

for(i=Tlength;i>=1;i--)

{

T[i] = T[i-1];

}

c=new int \*[Slength+1];

p=new char \*[Slength+1];

for( i=0;i<Slength+1;i++)

{ c[i]=new int[Tlength+1];

p[i]=new char[Tlength+1];

}

for(i=0;i<=Tlength;i++)

{

c[i][0]=0;

}

for(i=0;i<=Slength;i++)

{

c[0][j]=0;

}

for(i=1;i<=Slength;i++)

{

for(j=1;j<=Tlength;j++)

{

if(S[i] == T[j] )

{

c[i][j] = c[i-1][j-1] + 1;

p[i][j]='\^';

}

else {if(c[i-1][j]>=c[i][j-1])

{

c[i][j] =c[i-1][j];

p[i][j]='|';

}

else

{

c[i][j]=c[i][j-1];

p[i][j]=' ';

}

}

}

plcs(p,S,Slength,Tlength);

}

}

int main()

{ clrscr();

char S[maxn],T[maxn];

cin>>S>>T;

LongestcSubsequence( S, T);

getch(); }

**7) Implement Breadth-First Search in a graph**

#include<iostream.h>

#include<conio.h>

int time ;

class vertex

{

public:

int parent;

int info , dt ;

char color;

};

class node

{

public:

int info;

node\* next;

node(int a , node\* n = 0)

{

info = a ;

next = n ;

}

};

class linked\_list

{

public:

node\* head ;

node\* tail ;

linked\_list()

{

head = tail = 0;

}

void add\_to\_tail ( int a )

{

node\* z = new node (a);

if ( head == 0 )

head = tail = z ;

else

{

tail->next = z ;

tail = z ;

}

}

};

// end of class linked\_list

class queue

{

public:

node\* head ;

node\* tail ;

queue()

{ head = tail = 0 ; }

int isempty();

void enqueue(int);

int dequeue();

};

// end of class queue

int queue :: isempty()

{

if ( head == 0 )

return 1 ;

else

return 0 ;

}

void queue :: enqueue (int a)

{

node\* z = new node (a);

if ( head == 0 )

head = tail = z ;

else

{

tail->next = z ;

tail = z ;

}

}

// end of enqueue function

int queue :: dequeue ()

{

node\* z ;

if ( !isempty() )

{

z = head ;

head = head->next ;

if ( head == 0 )

tail = 0 ;

}

return z->info ;

}

// end of dequeue function

class BFS

{

public:

linked\_list llist[10];

vertex var[20];

queue q ;

void enter();

void bfs();

void print();

};

void BFS :: enter()

{

int size;

cout<<"\nEnter the no. of vertices in the graph : " ;

cin>>size;

cout<<"\nEnter the value of the vertices : ";

for ( int k = 1 ; k <= size ; k++ )

cin>>var[k].info;

for ( int i = 1 ; i <= size ; i++ )

{

cout<<"\n\nFor vertex containing value " << var[i].info <<" :\n";

cout<<"\nEnter the no. of adjacent vertices : ";

int adj;

cin>>adj;

for ( int j = 1 ; j <= adj ; j++ )

{

cout<<"\nEnter the value of the adjacent verteices : ";

int a;

cin>>a;

llist[i].add\_to\_tail(a);

}

}

} // end of function enter()

void BFS :: bfs ()

{

int i = 1;

while (var[i].info )

{

var[i].color = 'w' ;

var[i].dt = -1 ;

var[i].parent = 0 ;

i++;

}

cout <<"\nEnter the value of that vertex that u want to be the source : ";

int k ;

cin>>k;

for ( i = 1 ; var[i].info ; i++ )

{

if ( var[i].info == k )

{

var[i].color = 'g' ;

var[i].dt = 0 ;

break;

}

}

q.enqueue ( var[i].info ) ; // the source is enqueued

node\* temp = q.head;

while ( temp != 0 )

{

int u = q.dequeue() ;

node\* z = llist[u].head ;

for ( ; z != 0 ; z = z->next )

{

if ( var[z->info].color == 'w' )

{

var[z->info].color = 'g';

var[z->info].dt = var[u].dt + 1 ;

var[z->info].parent = u;

q.enqueue ( var[z->info].info );

}

}

var[u].color = 'b';

temp = q.head;

}

// end of while loop

}

// end of bfs function

void BFS :: print()

{

int u = 1 ;

while ( var[u].info )

{

cout<<"\n\nVertex info : " << var[u].info ;

cout<<"\t Colour : " << var[u].color;

cout<<"\t Parent : " << var[u].parent;

cout<<"\t Discovery time : " << var[u].dt;

u++;

}

}

void main ()

{

clrscr();

BFS b;

b.enter();

b.bfs();

b.print();

getch();

}

**8) Implement Depth-First Search in a graph**

#include<iostream.h>

#include<conio.h>

int time ;

class vertex

{

public:

int parent;

int info , ft , dt ;

char color;

};

class node

{

public:

int info;

node\* next;

node(int a , node\* n = 0)

{

info = a ;

next = n ;

}

};

class linked\_list

{

public:

node\* head ;

node\* tail ;

linked\_list()

{

head = tail = 0;

}

void add\_to\_tail ( int a )

{

node\* z = new node (a);

if ( head == 0 )

head = tail = z ;

else

{

tail->next = z ;

tail = z ;

}

}

};

// end of class linked\_list

class DFS

{

public:

linked\_list llist[10];

vertex var[20];

void enter();

void dfs();

void dfs\_visit ( int );

void print();

};

void DFS :: enter()

{

int size;

cout<<"\nEnter the no. of vertices in the graph : " ;

cin>>size;

cout<<"\nEnter the value of the vertices : ";

for ( int k = 1 ; k <= size ; k++ )

cin>>var[k].info;

for ( int i = 1 ; i <= size ; i++ )

{

cout<<"\n\nFor vertex containing value " << var[i].info <<" :\n";

cout<<"\nEnter the no. of adjacent vertices : ";

int adj;

cin>>adj;

for ( int j = 1 ; j <= adj ; j++ )

{

cout<<"\nEnter the value of the adjacent verteices : ";

int a;

cin>>a;

llist[i].add\_to\_tail(a);

}

}

} // end of function enter

void DFS :: dfs\_visit ( int u )

{

var[u].color = 'g';

time = time + 1;

var[u].dt = time ;

node\* t = llist[u].head;

for ( ; t != 0 ; t = t->next )

{

if ( var[ t->info ].color == 'w' )

{

var [ t->info ].parent = u ;

dfs\_visit ( t->info );

}

}

var[u].color = 'b';

time = time + 1 ;

var[u].ft = time ;

}

// end of function dfs\_visit

void DFS :: dfs ()

{

int u = 1 ;

while ( var[u].info )

{

var[u].color = 'w';

var[u].parent = 0 ;

u++;

}

time = 0 ;

u = 1 ;

while ( var[u].info )

{

if ( var[u].color == 'w' )

dfs\_visit ( u ) ;

u++;

}

}

// end of dfs function

void DFS :: print ()

{

int u = 1 ;

while ( var[u].info )

{

cout<<"\n\nVertex info : " << var[u].info ;

cout<<"\t Colour : " << var[u].color;

cout<<"\t Parent : " << var[u].parent;

cout<<"\t Discovery time : " << var[u].dt;

cout<<"\t Finishing time : " << var[u].ft;

u++;

}

}

void main()

{

clrscr();

DFS d;

d.enter();

d.dfs();

d.print();

getch();

}

**9) Write a program to determine the minimum spanning tree of a graph**

#include<iostream.h>

#include<stdio.h>

#include<conio.h>

int weight[20][20],visited[20],d[20],p[20];

int v,e;

void creategraph()

{

int i,j,a,b,w;

cout<<"\nEnter number of vertices";

cin>>v;

cout<<"\nEnter number of edges";

cin>>e;

for(i=1;i<=v;i++)

for(j=1;j<=v;j++)

weight[i][j]=0;

for(i=1;i<=v;i++)

{

p[i]=visited[i]=0;

d[i]=32767;

}

for(i=1;i<=e;i++)

{

cout<<"\nEnter edge a,b and weight w:";

cin>>a>>b>>w;

weight[a][b]=weight[b][a]=w;

}

}

void prim()

{

int current,totalvisited,mincost,i;

current=1;d[current]=0;

totalvisited=1;

visited[current]=1;

while(totalvisited!=v)

{

for(i=1;i<=v;i++)

{

if(weight[current][i]!=0)

if(visited[i]==0)

if(d[i]>weight[current][i])

{

d[i]=weight[current][i];

p[i]=current;

}

}

mincost=32767;

for(i=1;i<=v;i++)

{

if(visited[i]==0)

if(d[i]<mincost)

{

mincost=d[i];

current=i;

}

}

visited[current]=1;

totalvisited++;

}

for(i=2;i<=v;i++)

cout<<"\nVertex "<<i<<" is connected to"<<p[i];

}

main()

{

clrscr();

creategraph();

prim();

getch();

}

**Some Additional Practicals**

1. **Bubble Sort**

#include<iostream.h>

#include<conio.h>

int bbsort( int a[100] , int n )

{

int count = 0;

for(int i = 1 ; i < n ; i++)

{

for(int j = 0 ; j < n-1 ; j++)

{

count++;

if(a[j] > a[j+1])

{

int temp = a[j];

a[j] = a[j+1];

a[j+1] = temp ;

}

}

}

return count;

}

void main()

{

clrscr();

int a[100] , n;

cout<<"\nEnter the no. of elements in the array :";

cin>>n;

cout<<"\nEnter the elements of the array :";

for(int i = 0 ; i < n ; i++)

{

cin>>a[i];

}

cout<<"\nThe no. of comparisons :"<<bbsort(a,n);

cout<<"\nThe array after sorting :";

for(i = 0; i < n ; i++)

{

cout<<a[i]<<" ";

}

getch();

}

1. **Selection Sort**

#include<iostream.h>

#include<conio.h>

int selsort(int a[100] , int n)

{

int min , temp , count = 0;

for(int i = 0 ; i < n-1 ; i++)

{

min = i;

for(int j = i+1 ; j <= n-1 ; j++)

{

count++;

if(a[j] < a[min])

{

min = j;

}

}

temp = a[i] ;

a[i] = a[min] ;

a[min] = temp ;

}

return count;

}

void main()

{

clrscr();

int a[100] , n;

cout<<"\nEnter the no. of elements in the array :";

cin>>n;

cout<<"\nEnter the elements of the array :";

for(int i = 0 ; i < n ; i++)

{

cin>>a[i];

}

cout<<"\nThe no. of comparisions : "<<selsort(a,n);

cout<<"\nThe array after sorting :";

for(i = 0 ; i < n ; i++)

{

cout<<" "<<a[i];

}

getch(); }