**1.Write a program for creating max./min. heap using INSERT.**

#include<iostream.h>

#include<conio.h>

class HEAP

{

int n, A[200];

public:

HEAP(int);

void INSERT(int);

void SET\_ELE();

void GET\_ELE();

}

HEAP::HEAP(int par)

{

n=par;

}

void HEAP::INSERT(int no)

{

int j =no;

int i=no/2;

int item = A[no]; //new ele to be added in exhi. heap

while(i>0 && A[i] < item) //not a root && parent < new ele

{

A[j] = A[i]; //shift the parent down

j = i; // move the pair of ptrs up

i= i/2;

}

A[j]=item;

}

void HEAP::GET\_ELE()

{

cout<<endl;

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

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

}

void HEAP::SET\_ELE()

{

cout<<endl<<"Enter elements is:\n ";

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

cin>>A[i];

}

void main()

{

clrscr();

int n;

cout<<"Enter Total Numbers of elements:\n";

cin>>n;

HEAP obj(n);

obj.SET\_ELE();

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

obj.INSERT(i);

cout<<endl<<"Heap elements are : ";

obj.GET\_ELE();

getch();

}

**2. Write a program for creating max./min. heap using ADJUST/HEAPIFY.**

#include<iostream.h>

#include<conio.h>

class HEAP

{

int n, A[100];

public:

HEAP(int);

void HEAPIFY(int);

void ADJUST(int);

void SET\_ELE();

void GET\_ELE();

};

HEAP::HEAP(int para)

{

n=para;

}

void HEAP::ADJUST(int i)

{

int j=2\*i;

int item=A[i];

while(j<=n)

{

if(j<n && A[j]<A[j+1])

{

j=j+1;

}

if(item >= A[j])

{

break;

}

else

{

A[j/2]=A[j];//shift the child up

j=j\*2;//move the ptr down

}

}

A[j/2]=item;

}

void HEAP::HEAPIFY(int n)

{

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

{

ADJUST(i);

}

}

void HEAP::GET\_ELE()

{

//cout<<endl<<"\t node \t parent";

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

{

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

}

}

void HEAP::SET\_ELE()

{

cout<<endl<<"Enter elements:";

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

{

cin>>A[i];

}

}

void main()

{

clrscr();

int n;

cout<<"Enter Total Number of element: ";

cin>>n;

HEAP obj(n);

obj.SET\_ELE();

obj.HEAPIFY(n);

cout<<endl<<"Heap elements are:";

obj.GET\_ELE();

getch();

}

**3. Write a program to implement union and find operation.**

#include<iostreame.h>

#include<conio.h>

class SET

{

int n,PAR[20];

public:

SET(int);

void UNION(int,int);

int FIND(int);

void SHOW();

void MENU();

};

SET::SET(int para)

{

n=para;

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

{

PAR[i]=-1;

}

}

void SET::UNION(int i,int j)

{

//x is total weight of sub-trees i & j

int x=PAR[i]+PAR[j];

if(PAR[i]>PAR[j])

{

PAR[i]=j;

PAR[j]=x;

}

else

{

PAR[j]=i;

PAR[i]=x;

}

}

int SET::FIND(int i)

{

//find root first

int k,j=i;

while(PAR[j]> -1)

{

j=PAR[j];

}

//then collapse the tree

k=i;

while(k!=j)

{

int temp=PAR[k];

PAR[k]=j;

k=temp;

}

return j;

}

void SET::SHOW()

{

cout<<endl<<"\t Node \t Parent";

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

{

cout<<endl<<"\t"<<i<<"\t"<<PAR[i];

}

}

void SET::MENU()

{

int opt,i,j;

SHOW();

do

{

cout<<endl<<"Choose option:";

cout<<endl<<"--------------";

cout<<endl<<"1.Union";

cout<<endl<<"2.Find";

cout<<endl<<"3.Exit";

cout<<endl<<"--------------";

cout<<endl<<"Enter your option:";

cin>>opt;

switch(opt)

{

case 1:

cout<<endl<<"Enter 2 root element:";

cin>>i>>j;

UNION(i,j);

SHOW();

break;

case 2:

cout<<endl<<"Enter element to find:";

cin>>i;

cout<<endl<<"Root of "<<i<<"is"<<FIND(i);

SHOW();

break;

case 3:

return;

default:

cout<<endl<<"Invalid option:";

}

}while(1);

}

void main()

{

clrscr();

int n;

cout<<endl<<"Enter initial number of individual elements:\n";

cin>>n;

SET obj(n);

obj.MENU();

getch();

}

**4. Write a program to find minimum and maximum form a given array.**

#include<iostreame.h>

#include<conio.h>

#include<stdlib.h>

#include<timer.h>

class LIST

{

int n,\*A;

public:

LIST(int);

void SET\_ELE();

void GET\_ELE();

void MAXMIN(int,int,int&,int&);

};

LIST::LIST(int para)

{

n=para;

A=new int[n+1];

}

void LIST::GET\_ELE()

{

cout<<endl;

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

{

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

}

}

void LIST::SET\_ELE()

{

cout<<endl<<"Enter elements:";

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

{

A[i]=random(100);

//cin>>A[i];

}

}

void LIST::MAXMIN(int i,int j,int &fmax,int &fmin) //i is total no.of elements in heap

{

int gmax,gmin,hmax,hmin;

if(i==j)

fmax=fmin=A[i];

else

{

if(i==j-1)

{

if(A[i]>A[j])

{

fmax=A[i];

fmin=A[j];

}

else

{

fmax=A[j];

fmin=A[i];

}

}

else

{

int mid=(i+j)/2;

MAXMIN(i,mid,gmax,gmin);

MAXMIN(mid+1,j,hmax,hmin);

if(gmax>hmax)

fmax=gmax;

else

fmax=hmax;

if(gmin<hmin)

fmin=gmin;

else

fmin=hmin;

}

}

}

void main()

{

int n,max,min;

clrscr();

Timer Tobj;

cout<<"Enter Total Number of element: ";

cin>>n;

LIST obj(n);

obj.SET\_ELE();

// cout<<endl<<"Elements of the List are:";

obj.GET\_ELE();

Tobj.start();

obj.MAXMIN(1,n,max,min);

Tobj.stop();

//obj.GET\_ELE();

cout<<endl<<"\n\n Min="<<min<<”\n”<<"And max ="<<max;

cout<<endl<<"Time taken for execution="<<Tobj.time();

getch();

}

**5. Write a program for searching element form given array using binary search for 1000,2000,3000 find exact time of execution.**

#include<iostream.h>

#include<conio.h>

#include<stdlib.h>

#include<timer.h>

class LIST

{

int \*A,n;

public:

LIST(int);

void SET\_ELE();

void GET\_ELE();

void BINARY\_SEARCH(int,int &);

void BINARY\_SEARCH1(int,int &);

void SORT();

};

LIST::LIST(int par)

{

n=par;

A=new int[n+1];

}

void LIST::SET\_ELE()

{

//cout<<endl<<"Enter list elements:\n";

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

// cin>>A[i];

A[i]=random(100);

}

void LIST::GET\_ELE()

{

cout<<endl<<"List elements are:\n";

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

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

}

void LIST::BINARY\_SEARCH(int x,int &j)

{

int low,high,mid;

low=1;

high=n;

while(low<=high)

{

mid=(low+high)/2;

if(x<A[mid])

high=mid-1;

else

{

if(x>A[mid])

low=mid+1;

else

{

j=mid;

return;

}

}

}

j=0;

}

void LIST::SORT()

{

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

{

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

{

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

{

int temp=A[j];

A[j]=A[j+1];

A[j+1]=temp;

}

}

}

}

void LIST::BINARY\_SEARCH1(int x, int &j)

{

int low,high,mid;

low=1;

high=n+1;

while(low<high-1)

{

mid=(low+high)/2;

if(x<A[mid])

high=mid;

else

low=mid;

}

if(x==A[low])

j=low;

else

j=0;

}

void main()

{

int n,pos,x;

clrscr();

Timer T;

cout<<"\n Enter Total number of element:";

cin>>n;

LIST obj(n);

obj.SET\_ELE();

obj.SORT();

cout<<endl<<"Elements of the List are:\n";

obj.GET\_ELE();

cout<<endl<<"Enter element to search in list:\n";

cin>>x;

T.start();

obj.BINARY\_SEARCH(x,pos);

//obj.BINARY\_SEARCH1(x,pos);

T.stop();

if(pos)

cout<<endl<<"Element found at:"<<pos;

else

cout<<endl<<"Element not found:";

cout<<endl<<"Time taken for execution="<<T.time();

getch();

}

**6. Write a program for sorting given array in ascending/descending order with n=1000,2000, 3000 find exact time of execution using**

* **Heap sort:**

#include<iostream.h>

#include<conio.h>

#include<stdlib.h>

#include<timer.h>

class HEAP

{

int n,\*A;

public:

HEAP(int);

void HEAP\_SORT();

void HEAPIFY();

void ADJUST(int,int);

void SET\_ELE();

void GET\_ELE();

};

HEAP::HEAP(int para)

{

n=para;

A=new int[n+1];

}

void HEAP::HEAPIFY()

{

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

{

ADJUST(i,n);

}

}

void HEAP::HEAP\_SORT()

{

HEAPIFY();

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

{

//exchange

int temp=A[i];

A[i]=A[1];

A[1]=temp;

//heapify the disturbed heap again

ADJUST(1,i-1);

}

}

void HEAP::ADJUST(int i,int n) //i is total no.of elements in heap

{

int j=2\*i;

int item=A[i];

while(j<=n)

{

if(j<n && A[j]<A[j+1])

{

j=j+1;

}

if(item >= A[j])

{

break;

}

else

{

A[j/2]=A[j];//shift the child up

j=j\*2;//move the ptr down

}

}

A[j/2]=item;

}

void HEAP::GET\_ELE()

{

//cout<<endl<<"\t node \t parent";

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

{

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

}

}

void HEAP::SET\_ELE()

{

cout<<endl<<"Enter elements:";

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

{

A[i]=random(1000);

// cin>>A[i];

}

}

void main()

{

int n;

Timer Tobj;

clrscr();

cout<<"Enter Total Number of element: ";

cin>>n;

HEAP obj(n);

obj.SET\_ELE();

cout<<endl<<"\n Elements Before sorting are:\n";

obj.GET\_ELE();

Tobj.start();

obj.HEAP\_SORT();

Tobj.stop();

cout<<endl<<"\n Elements After sorting are:\n";

obj.GET\_ELE();

cout<<endl<<"Time taken for execution="<<Tobj.time();

getch();

}

* **Merge sort:**

#include<iostream.h>

#include<conio.h>

#include<stdlib.h>

#include<timer.h>

class LIST

{

int n,\*A;

public:

LIST(int);

void SET\_ELE();

void GET\_ELE();

void MERGE(int,int,int);

void MERGESORT(int,int);

};

LIST::LIST(int par)

{

n=par;

A=new int[n+1];

}

void LIST::SET\_ELE()

{

//cout<<endl<<"Enter the list elements is: \n";

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

{

//cin>>A[i];

A[i]=random(100);

}

}

void LIST::GET\_ELE()

{

// cout<<endl<<"The list elements is: \n";

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

{

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

}

}

void LIST::MERGE(int low,int mid,int high)

{

int \*B=new int[n+1];

int i=low;

int h=low;

int j=mid+1;

while(h<=mid && j<=high)

{

if(A[h]<A[j])

{

B[i]=A[h];

h=h+1;

}

else

{

B[i]=A[j];

j=j+1;

}

i=i+1;

}

if(h>mid)

{

for(int k=j;k<=high;k++)

{

B[i]=A[k];

i=i+1;

}

}

else

{

for(int k=h;k<=mid;k++)

{

B[i]=A[k];

i=i+1;

}

}

for(int k=low;k<=high;k++)

{

A[k]=B[k];

}

delete B;

}

void LIST::MERGESORT(int low,int high)

{

if(low<high)

{

int mid=(low+high)/2;

MERGESORT(low,mid);

MERGESORT(mid+1,high);

MERGE(low,mid,high);

}

}

void main()

{

int n;

clrscr();

cout<<"Enter total no.of elements \n";

cin>>n;

LIST obj(n);

Timer Tobj;

obj.SET\_ELE();

cout<<endl<<"\nElement Before sorting Element is: \n";

obj.GET\_ELE();

Tobj.start();

obj.MERGESORT(1,n);

Tobj.stop();

cout<<endl<<"\nElement After sorting Element is: \n";

obj.GET\_ELE();

cout<<endl<<"Time taken for execution="<<Tobj.time();

getch();

}

* **Quick sort:**

#include<iostream.h>

#include<conio.h>

#include<stdlib.h>

#include<timer.h>

class LIST

{

int \*A,n;

public:

LIST(int);

void SET\_LIST();

void QUICK\_SORT(int,int);

void PARTITION(int,int&);

void SHOW\_LIST();

};

LIST::LIST(int para)

{

n=para;

A= new int[n+2];

}

void LIST::SET\_LIST()

{

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

A[i]=random(5000);

A[i]=9999;

}

void LIST::QUICK\_SORT(int p,int q)

{

if(p<q)

{

int j=q+1;

PARTITION(p,j);

QUICK\_SORT(p,j-1);

QUICK\_SORT(j+1,q);

}

}

void LIST::PARTITION(int m, int & p)

{

int v =A[m];

int i=m;

do

{

do

{

i=i+1;

}while(A[i]<v);

do

{

p=p-1;

}while(A[p]>v);

if(i<p)

{

int temp=A[i];A[i]=A[p];A[p]=temp;

}

else

break;

}while(1);

A[m]=A[p];

A[p]=v;

}

void LIST::SHOW\_LIST()

{

//cout<<endl;

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

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

}

void main()

{

clrscr();

int n;

Timer T;

cout<<endl<<"Enter no. of elemets ; ";

cin>>n;

LIST obj(n);

obj.SET\_LIST();

cout<<endl<<"\nElements before sorting : \n";

obj.SHOW\_LIST();

T.start();

obj.QUICK\_SORT(1,n);

T.stop();

cout<<endl<<"\nElements after sorting : \n";

obj.SHOW\_LIST();

cout<<endl<<"Time taken by the prog to sort : "<<T.time();

getch();

}

**7. Write a program for matrix multiplication using Strassen's matrix multiplication.**

#include<iostream.h>

#include<conio.h>

class SET

{

int A[3][3],B[3][3],C[3][3];

public:

void READ\_MATRIX();

void STRESSEN();

void SHOW\_MATRIX();

};

void SET::READ\_MATRIX()

{

cout<<endl<<"Enter elements of first matrix A:\n";

for(int i=1;i<=2;i++)

for(int j=1;j<=2;j++)

cin>>A[i][j];

cout<<endl<<"Enter elements of second matrix B:\n";

for(i=1;i<=2;i++)

for(j=1;j<=2;j++)

cin>>B[i][j];

}

void SET::STRESSEN()

{

int p,q,r,s,t,u,v;

p=(A[1][1] + A[2][2]) \* (B[1][1] + B[2][2]);

q=(A[2][1] + A[2][2]) \* B[1][1];

r=A[1][1] \* (B[1][2] - B[2][2]);

s=A[2][2] \* (B[2][1] - B[1][1]);

t=(A[1][1] + A[1][2]) \* B[2][2];

u=(A[2][1] - A[1][1]) \* (B[1][1] + B[1][2]);

v=(B[1][2] - A[2][2]) \* (B[2][1] + B[2][2]);

C[1][1] = p + s - t + v;

C[1][2] = r + t;

C[2][1] = q + s;

C[2][2] = p + r - q + u;

}

void SET::SHOW\_MATRIX()

{

for(int i=1;i<=2;i++)

{

cout<<endl;

for(int j=1;j<=2;j++)

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

cout<<" ";

for(j=1;j<=2;j++)

cout<<B[i][j]<<" ";

cout<<" ";

for(j=1;j<=2;j++)

cout<<C[i][j]<<" ";

}

}

void main()

{

clrscr();

SET obj;

obj.READ\_MATRIX();

obj.STRESSEN();

obj.SHOW\_MATRIX();

getch();

}

**8.1 Write a program to find solution of Knapsack instant. Greedy method(Fractional)**

#include<iostream.h>

#include<conio.h>

#include<stdlib.h>

#include<timer.h>

class GREEDY

{

int n;

float P[20],W[20],X[20],M,max\_profit;

public:

GREEDY(int);

void READ\_LIST();

void SHOW\_LIST();

void GREEDY\_KNAPSACK();

void SORT\_OBJECTS();

};

GREEDY::GREEDY(int para)

{

n=para;

max\_profit=0;

}

void GREEDY::READ\_LIST()

{

cout<<endl<<"Enter weights of objects : \n";

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

cin>>W[i];

cout<<endl<<"Enter profit/ values of objects : \n";

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

cin>>P[i];

cout<<endl<<"Enter max capacity of knapsack : ";

cin>>M;

}

void GREEDY::SORT\_OBJECTS()

{

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

{

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

{

if(P[j]/W[j] < P[j+1]/W[j+1] )

{

float temp=P[j];

P[j]=P[j+1];

P[j+1]=temp;

temp=W[j];

W[j]=W[j+1];

W[j+1]=temp;

}

}

}

}

void GREEDY::GREEDY\_KNAPSACK()

{

// int X;

float cu;

SORT\_OBJECTS();

cout<<endl<<"Objects after sorting:\n ";

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

cout<P[i]<<" ";

cout<<endl;

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

cout<<W[i]<<" ";

cout<<endl;

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

cout<<P[i]/W[i]<" ";

cu=M;

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

X[i]=0;

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

{

if(W[i]<=cu)

{

X[i]=1.0;

cu=cu-W[i];

}

else

break;

}

if(i<=n)

{

X[i]=cu/W[i];

}

}

void GREEDY::SHOW\_LIST()

{

int i,k;

cout<<endl<<"\n Solution vector is : \n";

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

{

cout<<X[i]<<" ";

max\_profit=max\_profit+ P[i]\*X[i];

}

cout<<endl<<"\nTotal profit earned is =\n"<<max\_profit;

}

void main()

{

clrscr();

int n;

cout<<endl<<"Enter no of objects ; ";

cin>>n;

GREEDY obj(n);

obj.READ\_LIST();

//obj.SORT\_OBJECTS();

obj.GREEDY\_KNAPSACK();

obj.SHOW\_LIST();

getch();

}

**8.2 Write a program to find solution of Knapsack instant Knapsack (1/0):**

#include<iostream.h>

#include<conio.h>

#include<stdlib.h>

#include<timer.h>

class KNAPSACK

{

int n,X[10]; //B(n x M )

float P[10],W[10],M,B[10][20],max\_profit;

public:

KNAPSACK(int);

void GET\_DATA();

void KNAPSACK\_DP();

void SHOW\_SOL();

float MAX(float,float);

};

KNAPSACK::KNAPSACK(int par)

{

n=par;

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

X[i]=0;

max\_profit=0;

}

void KNAPSACK::GET\_DATA()

{

cout<<endl<<"Enter weights of objects : ";

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

cin>>W[i];

cout<<endl<<"Enter profit / value of objects : ";

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

cin>>P[i];

cout<<endl<<"Enter Max capacity of Knapsack : ";

cin>>M;

}

void KNAPSACK::KNAPSACK\_DP()

{

//first row zero

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

B[0][i]=0;

//first column zero

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

B[i][0]=0;

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

for(int cu=1;cu<=M;cu++)

if(W[i]<=cu)

// MaxOf(profdueto new or old profit)

B[i][cu]=MAX(P[i]+B[i-1][cu-W[i]] , B[i-1][cu]);

else

B[i][cu]=B[i-1][cu];

}

float KNAPSACK::MAX(float a, float b)

{

if(a>b)

return a;

else

return b;

}

void KNAPSACK::SHOW\_SOL()

{

int i,k;

cout<<endl;

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

{

cout<<endl;

for(int cu=1;cu<=M;cu++)

cout<<B[i][cu]<<" ";

}

i=n;

k=M;

while(i>0 && k>0)

{

if(B[i][k]!=B[i-1][k])

{

X[i]=1;

k=k-W[i];

}

i=i-1;

}

cout<<endl<<"Solution Vector is \n";

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

{

cout<<X[i]<<" ";

max\_profit=max\_profit + P[i] \* X[i];

}

cout<<endl<<"\nMax profit gained = \n"<<max\_profit;

}

void main()

{

int n;

clrscr();

cout<<endl<<"Enter no of objects : ";

cin>>n;

KNAPSACK obj(n);

obj.GET\_DATA();

obj.KNAPSACK\_DP();

obj.SHOW\_SOL();

getch();

}

**9. Write a program to find shortest path using single source shortest path.**

#include<iostream.h>

#include<conio.h>

#include<stdlib.h>

#include<timer.h>

int COST[10][10]={

{0,0,0,0,0,0,0},

{0, 999,50,10,999,45,999},

{0, 999,999,15,999,10,999},

{0, 20,999,999,15,999,999},

{0, 999,20,999,999,35,999},

{0, 999,999,999,30,999,999},

{0, 999,999,999,3,999,999}

};

class GRAPH

{

int n,v,DIST[10];

//int COST[10][10];

public:

GRAPH(int);

void READ\_GRAPH();

void SSSP();

void SHOW\_GRAPH();

int MIN(int,int);

};

GRAPH::GRAPH(int para)

{

n=para;

}

void GRAPH::READ\_GRAPH()

{

/\*cout<<endl<<"Enter cost adj matrix : \n";

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

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

cin>>COST[i][j];

\*/

cout<<endl<<"Enter source vertex : ";

cin>>v;

}

void GRAPH::SSSP()

{

int u,w,S[10];

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

{

S[i]=0;

DIST[i]=COST[v][i];

}

DIST[v]=0;

S[v]=1;

for(int num=2;num<=n-1;num++)

{

// find u such that .....

int min=999;

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

{

if(S[w]==0 && DIST[w]<min)

{

min=DIST[w];

u=w;

}

}

S[u]=1;

//update DIST[]

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

{

if(S[w]==0)

DIST[w]=MIN( DIST[w], (DIST[u]+COST[u][w]) );

}

}

}

int GRAPH::MIN(int x,int y)

{

if(x<y) return x; else return y;

}

void GRAPH::SHOW\_GRAPH()

{

cout<<endl<<"\nCOST Adj matrix is :\n ";

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

{

cout<<endl;

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

cout<<COST[i][j]<<" ";

}

cout<<endl<<"Source\tDesti\t Distance";

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

cout<<endl<<v<<"\t"<<i<<"\t"<<DIST[i];

}

void main()

{

clrscr();

int n;

cout<<endl<<"Enter no of vertices : ";

cin>>n;

GRAPH obj(n);

obj.READ\_GRAPH();

obj.SSSP();

obj.SHOW\_GRAPH();

getch();

}

**10. Write a program to find Minimum-Cost Spanning Trees (Prim's & Kruskal's Algorithm).**

**PRIM’S:**

#include<iostream.h>

#include<conio.h>

int COST[10][10]={

{0,0,0,0,0,0,0},

{0, 999,55 ,999, 50,999,999},

{0, 55,999,15 , 40, 20,999},

{0, 999, 15,999,999, 10, 30},

{0, 50, 40,999,999, 45,999},

{0, 999, 20, 10, 45,999, 25},

{0, 999,999, 30,999, 25,999}

};

class GRAPH

{

int n;

//int COST[10][10];

int T[10][2],min\_cost;

public:

GRAPH(int);

void READ\_GRAPH();

void SHOW\_GRAPH();

void PRIMS();

void SHOW\_TREE();

};

GRAPH::GRAPH(int para)

{

n=para;

}

void GRAPH::READ\_GRAPH()

{

cout<<endl<<"Enter cost adj matrix : \n";

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

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

cin>>COST[i][j];

}

void GRAPH::SHOW\_GRAPH()

{

cout<<endl<<"Cost adj matrix is : \n";

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

{

cout<<endl;

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

cout<<COST[i][j]<<" ";

}

}

void GRAPH::PRIMS()

{

int NEAR[10],i,j,k,l,min;

//find min cost edge

min=999;

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

{

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

{

if(COST[i][j]<min)

{

min=COST[i][j];

k=i;l=j;

}

}

}

min\_cost=COST[k][l];

// add first edge in tree

T[1][1]=k; T[1][2]=l;

// initialize NEAR[]

cout<<endl;

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

{

if( COST[i][l]<COST[i][k] )

NEAR[i]=l;

else

NEAR[i]=k;

}

NEAR[k]=0; NEAR[l]=0;

// find remaining n-2 edges for sp.tree

for(i=2;i<=n-1;i++)

{

// find j such that COST[j][NEAR[j]] is min

min=999;

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

{

if(NEAR[w]!=0 && COST[w][NEAR[w]]<min)

{

min=COST[w][NEAR[w]];

j=w;

}

}

// cout<<endl<<"j="<<j;

min\_cost=min\_cost+COST[j][NEAR[j]];

//add next edge to tree

T[i][1]=j;

T[i][2]=NEAR[j];

NEAR[j]=0;

// update NEAR[]

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

{

if( NEAR[w] !=0 && COST[w][j] < COST[w][NEAR[w]] )

NEAR[w]=j;

}

}

if(min\_cost < 999)

SHOW\_TREE();

else

cout<<endl<<"No spanning Tree !";

}

void GRAPH::SHOW\_TREE()

{

cout<<endl<<"Min Cost spannig tree is : \n";

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

cout<<endl<<T[i][1]<<" "<<T[i][2];

cout<<endl<<"Min Cost is : "<<min\_cost;

}

void main()

{

clrscr();

int n;

cout<<endl<<"Enter no of vertices : ";

cin>>n;

GRAPH obj(n);

obj.SHOW\_GRAPH();

obj.PRIMS();

getch();

}

**KRUSKAL:**

#include<iostream.h>

#include<conio.h>

int COST[10][10]={

{0,0,0,0,0,0,0},

{0, 999,55 ,999, 50,999,999},

{0, 55,999,15 , 40, 20,999},

{0, 999, 15,999,999, 10, 30},

{0, 50, 40,999,999, 45,999},

{0, 999, 20, 10, 45,999, 25},

{0, 999,999, 30,999, 25,999}

};

class GRAPH

{

int n,e;

//int COST[10][10];

int T[10][2],min\_cost,PAR[10];

int EDGE[20][4];

public:

GRAPH(int);

void READ\_GRAPH();

void SHOW\_GRAPH();

void KRUSKAL();

void UNION(int,int);

int FIND(int);

void SHOW\_TREE();

void CREATE\_EDGE\_LIST();

void SORT\_EDGE\_LIST();

};

GRAPH::GRAPH(int para)

{

n=para;

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

PAR[i] = -1;

}

void GRAPH::READ\_GRAPH()

{

cout<<endl<<"Enter cost adj matrix : \n";

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

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

cin>>COST[i][j];

}

void GRAPH::SHOW\_GRAPH()

{

cout<<endl<<"Cost adj matrix is : \n";

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

{

cout<<endl;

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

cout<<COST[i][j]<<" ";

}

}

void GRAPH::UNION(int i,int j)

{

PAR[i]=j;

}

int GRAPH::FIND(int i)

{

int j=i;

while(PAR[j] > -1)

j=PAR[j];

return j;

}

void GRAPH::CREATE\_EDGE\_LIST()

{

//find min cost edge

e=0;

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

{

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

{

if(COST[i][j] < 999)

{

e=e+1;

EDGE[e][1]=i;

EDGE[e][2]=j;

EDGE[e][3]=COST[i][j];

}

}

}

SORT\_EDGE\_LIST();

}

void GRAPH::SORT\_EDGE\_LIST()

{

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

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

if(EDGE[j][3] > EDGE[j+1][3] )

{

int temp= EDGE[j][1];

EDGE[j][1]=EDGE[j+1][1];

EDGE[j+1][1]=temp;

temp= EDGE[j][2];

EDGE[j][2]=EDGE[j+1][2];

EDGE[j+1][2]=temp;

temp= EDGE[j][3];

EDGE[j][3]=EDGE[j+1][3];

EDGE[j+1][3]=temp;

}

cout<<endl<<"\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n";

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

cout<<endl<<EDGE[i][1]<<" "<<EDGE[i][2]<<" "<<EDGE[i][3];

}

void GRAPH::KRUSKAL()

{

int u,v,i=0,j,k;

min\_cost=0;

int ptr=0;

while(i<n-1 && ptr<e)

{

ptr=ptr+1;

u=EDGE[ptr][1];

v=EDGE[ptr][2];

j=FIND(u);

k=FIND(v);

if(j!=k)

{

// add first edge in tree

T[i][1]=u;

T[i][2]=v;

UNION(j,k);

min\_cost=min\_cost + COST[u][v];

}

}

if(min\_cost < 999)

SHOW\_TREE();

else

cout<<endl<<"No spanning Tree !";

}

void GRAPH::SHOW\_TREE()

{

cout<<endl<<"Min Cost spannig tree is : \n";

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

cout<<endl<<T[i][1]<<" "<<T[i][2];

cout<<endl<<"Min Cost is : "<<min\_cost;

}

void main()

{

clrscr();

int n;

cout<<endl<<"Enter no of vertices : ";

cin>>n;

GRAPH obj(n);

//obj.READ\_GRAPH();

obj.SHOW\_GRAPH();

obj.CREATE\_EDGE\_LIST();

obj.KRUSKAL();

getch();

}

**11. Write a program to find shortest path using all pair path.**

#include<iostream.h>

#include<conio.h>

int COST[10][10]={

{0, 0, 0, 0, 0, 0},

{0, 999,999,999, 7, 6},

{0, 3,999, 3,999, 8},

{0, 4, 9,999,999, 2},

{0, 999,999, 1,999,999},

{0, 999,999,999, 5,999}

};

class GRAPH

{

int n,A[10][10];

//int COST[10][10];

public:

GRAPH(int);

void READ\_GRAPH();

void SHOW\_GRAPH();

void ALL\_PAIRS();

void SHOW\_PATH\_DIST();

};

GRAPH::GRAPH(int para)

{

n=para;

}

void GRAPH::READ\_GRAPH()

{

cout<<endl<<"Enter cost adj matrix : \n";

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

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

cin>>COST[i][j];

}

void GRAPH::SHOW\_GRAPH()

{

cout<<endl<<"Cost adj matrix is : \n";

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

{

cout<<endl;

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

cout<<COST[i][j]<<" ";

}

}

void GRAPH::SHOW\_PATH\_DIST()

{

cout<<endl<<"\nAll pair shortest path distance matrix is : \n";

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

{

cout<<endl;

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

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

}

}

void GRAPH::ALL\_PAIRS()

{

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

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

A[i][j]=COST[i][j];

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

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

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

if( A[i][k]+A[k][j] < A[i][j] )

A[i][j]=A[i][k]+A[k][j];

}

void main()

{

clrscr();

int n;

cout<<endl<<"Enter no of vertices : ";

cin>>n;

GRAPH obj(n);

//obj.READ\_GRAPH();

obj.SHOW\_GRAPH();

obj.ALL\_PAIRS();

obj.SHOW\_PATH\_DIST();

getch();

}

**12. Write a program to find longest common subsequence.**

#include<iostream.h>

#include<conio.h>

#include<string.h>

class STRING

{

char X[30],Y[30],Z[30],B[30][30];

int m,n,C[30][30];

public:

STRING();

void GET\_STRING();

void SET\_STRING();

void LCS\_LENGTH();

void LCS\_PRINT(int,int);

};

STRING::STRING()

{

}

void STRING::GET\_STRING()

{

cout<<endl<<"Entered strings are : \n";

cout<<X<<endl;

cout<<Y;

}

void STRING::SET\_STRING()

{

cout<<endl<<"Enter a string : ";

cin>>X;

cout<<endl<<"Enter another string : ";

cin>>Y;

}

void STRING::LCS\_LENGTH()

{

m=strlen(X);

n=strlen(Y);

for(int i=0;i<=m;i++) //first col zero

C[i][0]=0;

for(int j=0;j<=n;j++) //first row zero

C[0][j]=0;

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

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

{

if(X[i-1]==Y[j-1])

{

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

B[i][j] = '\\';

}

else

{

if(C[i-1][j]>=C[i][j-1])

{

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

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

}

else

{

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

B[i][j] = '-';

}

}

}

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

{

cout<<endl;

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

cout<<C[i][j]<<" ";

}

cout<<endl;

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

{

cout<<endl;

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

cout<<B[i][j]<<" ";

}

cout<<endl;

LCS\_PRINT(m,n);

}

void STRING::LCS\_PRINT(int r,int c)

{

if(r>0 && c>0)

{

if(B[r][c]=='\\')

{

LCS\_PRINT(r-1,c-1);

cout<<X[r-1]<<" ";

}

else

if(B[r][c]=='|')

LCS\_PRINT(r-1,c);

else

LCS\_PRINT(r,c-1);

}

}

void main()

{

clrscr();

STRING obj;

obj.SET\_STRING();

//obj.GET\_STRING();

obj.LCS\_LENGTH();

getch();

}

**13. Write a program to implement breadth first and depth first search.**

#include<iostream.h>

#include<conio.h>

int A[10][10]={

{0, 0,0,0,0,0,0,0,0},

{0, 0,1,0,1,1,0,0,0},

{0, 1,0,0,0,1,1,1,0},

{0, 0,0,0,0,1,0,0,1},

{0, 1,0,0,0,0,0,0,1},

{0, 1,1,1,0,0,1,0,0},

{0, 0,1,0,0,1,0,0,0},

{0, 0,1,0,0,0,0,0,0},

{0, 0,0,1,1,0,0,0,0}

};

class GRAPH

{

int n;

int T[10][2],min\_cost;

public:

GRAPH(int);

void READ\_GRAPH();

void SHOW\_GRAPH();

void BFS(int);

void DFS(int);

};

GRAPH::GRAPH(int para)

{

n=para;

}

void GRAPH::READ\_GRAPH()

{

cout<<endl<<"Enter cost adj matrix : \n";

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

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

cin>>A[i][j];

}

void GRAPH::SHOW\_GRAPH()

{

cout<<endl<<"Cost adj matrix is : \n";

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

{

cout<<endl;

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

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

}

}

void GRAPH::BFS(int v)

{

int VISITED[10],u;

int Q[10],front,rear;

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

VISITED[i]=0;

VISITED[v]=1;

u=v;

front=rear=0;

do

{

cout<<u<<" ";

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

{

if(A[u][w]==1 && VISITED[w]==0)

{

if(front==0)

front=1;

rear = rear+1;

Q[rear] = w;

VISITED[w]=1;

}

}

if(front==0)

return;

else // del from Q and store in u

{

u=Q[front];

if(front==rear)

front=rear=0;

else

front =front+1;

}

}while(1);

}

void GRAPH::DFS(int v)

{

int VISITED[10],u;

int STK[10],top;

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

VISITED[i]=0;

VISITED[v]=1;

u=v;

top=0;

do

{

cout<<u<<" ";

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

{

if(A[u][w]==1 && VISITED[w]==0)

{

top = top+1;

STK[top] = w;

VISITED[w]=1;

}

}

if(top==0)

return;

else // del from Q and store in u

{

u=STK[top];

top = top-1;

}

}while(1);

}

void main()

{

clrscr();

int n,v;

cout<<endl<<"Enter no of vertices : ";

cin>>n;

GRAPH obj(n);

//obj.READ\_GRAPH();

obj.SHOW\_GRAPH();

cout<<endl<<"Enter source vertex : ";

cin>>v;

cout<<endl<<”Breadth First Search Graph sequence=”;

//cout<<endl;

obj.BFS(v);

cout<<endl;

cout<<endl<<”Depth First Search Graph sequence=”;

obj.DFS(v);

getch();

}

**14. Write a program to implement breadth first and depth first traversal.**

#include<iostream.h>

#include<conio.h>

int A[10][10]={

{0, 0,0,0,0,0,0,0,0},

{0, 0,1,0,1,1,0,0,0},

{0, 1,0,0,0,1,1,1,0},

{0, 0,0,0,0,1,0,0,1},

{0, 1,0,0,0,0,0,0,1},

{0, 1,1,1,0,0,1,0,0},

{0, 0,1,0,0,1,0,0,0},

{0, 0,1,0,0,0,0,0,0},

{0, 0,0,1,1,0,0,0,0}

};

class GRAPH

{

int n;

int T[10][2],min\_cost,VISITED[10];

public:

GRAPH(int);

void READ\_GRAPH();

void SHOW\_GRAPH();

void BFS(int);

void DFS(int);

void BFT();

void DFT();

};

GRAPH::GRAPH(int para)

{

n=para;

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

VISITED[i]=0;

}

void GRAPH::READ\_GRAPH()

{

cout<<endl<<"Enter cost adj matrix : \n";

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

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

cin>>A[i][j];

}

void GRAPH::SHOW\_GRAPH()

{

cout<<endl<<"Cost adj matrix is : \n";

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

{

cout<<endl;

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

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

}

}

void GRAPH::BFS(int v)

{

int VISITED[10],u;

int Q[10],front,rear;

VISITED[v]=1;

u=v;

front=rear=0;

do

{

cout<<u<<" ";

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

{

if(A[u][w]==1 && VISITED[w]==0)

{

if(front==0)

front=1;

rear = rear+1;

Q[rear] = w;

VISITED[w]=1;

}

}

if(front==0)

return;

else // del from Q and store in u

{

u=Q[front];

if(front==rear)

front=rear=0;

else

front =front+1;

}

}while(1);

}

void GRAPH::DFS(int v)

{

int VISITED[10],u;

int STK[10],top;

VISITED[v]=1;

u=v;

top=0;

do

{

cout<<u<<" ";

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

{

if(A[u][w]==1 && VISITED[w]==0)

{

top = top+1;

STK[top] = w;

VISITED[w]=1;

}

}

if(top==0)

return;

else // del from Q and store in u

{

u=STK[top];

top = top-1;

}

}while(1);

}

void GRAPH::BFT()

{

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

{

if(VISITED[i]==0)

BFS(i);

}

}

void GRAPH::DFT()

{

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

{

if(VISITED[i]==0)

DFS(i);

}

}

void main()

{

clrscr();

int n,v;

cout<<endl<<"Enter no of vertices : ";

cin>>n;

GRAPH obj1(n);

GRAPH obj2(n);

//obj.READ\_GRAPH();

obj.SHOW\_GRAPH();

//cout<<endl<<"Enter source vertex : ";

//cin>>v;

cout<<endl<<”Breadth First Traversal Graph sequence=”;

//cout<<endl;

obj.BFT(v);

cout<<endl;

cout<<endl<<”Depth First Traversal Graph sequence=”;

obj.DFT(v);

getch();

}

**15. Write a program to find all solutions for 8-queen problem using backtracking**

#include "iostream.h"

#include "conio.h"

#include "stdlib.h"

class QUEEN

{

int n,X[10];

int count;

public:

QUEEN(int);

void N\_QUEEN();

int PLACE(int);

void SHOW();

};

QUEEN::QUEEN(int para)

{

n=para;

count=0;

}

void QUEEN::N\_QUEEN()

{

int k=1;

X[k]=0;

while(k>0)

{

X[k]=X[k]+1;

while(X[k]<=n && ! PLACE(k))

{

X[k]=X[k]+1;

}

if(X[k]<=n)

{

if(k==n)

{

count++;

SHOW();

}

else

{

k=k+1; X[k]=0;

}

}

else // backtrack

{

k=k-1;

}

}

}

int QUEEN::PLACE(int k)

{

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

if(X[i]==X[k] || abs(X[i]-X[k])==abs(i-k))

return 0;

return 1;

}

void QUEEN:: SHOW()

{

cout<<endl<<"\nN Queen Solution no "<<count<<" \n";

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

cout<<X[i]<<" ";

/\*

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

{

cout<<endl<<"--------------------"<<endl;

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

if(X[i]==j)

cout<<"| Q"<<i;

else

cout<<"| ";

}

cout<<endl<<"--------------------"<<endl;

\*/

}

//////////////////////////////////////////////

void main()

{

clrscr();

int n;

cout<<endl<<"Enter no of queens : ";

cin>>n;

QUEEN obj(n);

obj.N\_QUEEN();

getch();

}