**FLOYDS**

#include<stdio.h>

#include<stdlib.h>

#define INF 9999

#define MAX 1000

int min(int a,int b){

return (a<b)? a:b;

}

void floyd(int graph[MAX][MAX],int n){

int i,j,k;

for(k=0;k<n;k++){

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

for(j=0;j<n;j++){

graph[i][j]=min(graph[i][j],graph[i][k]+graph[k][j]);

}

}

}

}

int main(){

int n,i,j;

printf("Enter the no of vertices:");

scanf("%d",&n);

int graph[MAX][MAX];

printf("Enter Adjacency Matrix:\n");

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

for(j=0;j<n;j++){

scanf("%d",&graph[i][j]);

if(i!=j && graph[i][j]==-1)

graph[i][j]=INF;

}

}

floyd(graph,n);

printf("Shortest distance between every pair of vertices:\n");

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

for(j=0;j<n;j++){

if(graph[i][j]==INF)

printf("INF\t");

else

printf("%d\t",graph[i][j]);

}

printf("\n");

}

return 0;

}

**WARSHAL**

**void warshal(int graph[MAX][MAX],int n){**

**int i,j,k;**

**for(k=0;k<n;k++){**

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

**for(j=0;j<n;j++){**

**graph[i][j]=(graph[i][j]) || (graph[i][k] && graph[k][j]);**

**}**

**}**

**}**

**}**

**PRIMS**

#include<stdio.h>

#include<stdlib.h>

#define INF 9999

#define MAX 1000

void prims(int cost[MAX][MAX],int n, int s){

int min,u,v,visited[MAX],mincost=0,e=1;

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

visited[i]=0;

}

visited[s]=1;

while(e<n){

min=INF;

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

if(visited[i]==1){

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

if(!visited[j] && cost[i][j]<min){

min=cost[i][j];

u=i;

v=j;

}

}

}

}

if(visited[v]==0){

printf("\n Edge %d:(%d %d) cost:%d",e++,u,v,min);

mincost+=min;

visited[v]=1;

}

cost[u][v]=cost[v][u]=INF;

}

printf("\nTotal cost:%d",mincost);

}

int main(){

int n,s,cost[MAX][MAX];

printf("Enter no of nodes:");

scanf("%d",&n);

printf("Enter Adjacency Matrix:\n");

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

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

scanf("%d",&cost[i][j]);

if(cost[i][j]==-1)

cost[i][j]=INF;

}

}

printf("Enter Source:");

scanf("%d",&s);

prims(cost,n,s);

return 0;

}

**Graph Connectivity**

#include<stdio.h>

#define MAX 1000

int a[MAX][MAX],q[MAX],visited[MAX],n,i,j,f=0,r=-1;

void bfs(int v){

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

if(a[v][i] && !visited[i]){

q[++r]=i;

}

}

if(f<=r){

visited[q[f]]=1;

bfs(q[f++]);

}

}

int main(){

int v=1,count=0;

printf("enter the no of vertices:");

scanf("%d",&n);

printf("Enter Adjacency matrix:\n");

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

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

scanf("%d",&a[i][j]);

}

}

bfs(v);

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

if(visited[i])

count++;

}

if(count==n)

printf("\nGraph is connected.");

else

printf("\nGraph is not connected");

return 0;

}

**QUICKSORT**

#include<stdio.h>

#include<stdlib.h>

#include<time.h>

#define MAX 1000

int count;

int partition(int a[MAX],int l,int r){

int pivot=a[l];

int i=l+1,j=r,temp;

while(1){

while(pivot>=a[i] && i<=r){

i++;

count++;

}

while(pivot<a[j] && j>l){

j--;

count++;

}

if(i<j){

temp=a[i];

a[i]=a[j];

a[j]=temp;

}

else{

temp=a[l];

a[l]=a[j];

a[j]=temp;

return j;

}

}

}

void quicksort(int a[MAX],int l,int r){

if(l<r){

int s=partition(a,l,r);

quicksort(a,l,s-1);

quicksort(a,s+1,r);

}

}

int main(){

clock\_t start,end;

int a[MAX],x[MAX],y[MAX],z[MAX];

int n,i,j,c1,c2,c3;

printf("enter no of array elements:");

scanf("%d",&n);

printf("enter array elements:\n");

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

scanf("%d",&a[i]);

}

start=clock();

quicksort(a,0,n-1);

end=clock();

printf("\n Sorted array:");

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

printf("%d\t",a[i]);

}

printf("\nTIME TAKEN TO SORT IS:%fs",(double)(end-start)/CLOCKS\_PER\_SEC);

printf("\ncount=%d",count);

printf("\nSIZE\tASC\tDESC\tRAND\n");

for(i=16;i<MAX;i\*=2){

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

x[j]=j;

y[j]=i-j-1;

z[j]=rand() %i;

}

count=0;

quicksort(x,0,i-1);

c1=count;

count=0;

quicksort(y,0,i-1);

c2=count;

count=0;

quicksort(z,0,i-1);

c3=count;

printf("%d\t%d\t%d\t%d\n",i,c1,c2,c3);

}

return 0;

}

**MERGESORT**

#include<stdio.h>

#include<stdlib.h>

#include<time.h>

#define MAX 1000

int count;

int merge(int a[MAX],int low,int mid,int high){

int i,j,k;

int b[MAX];

i=low;

j=mid+1;

k=low;

while(i<=mid && j<=high){

if(a[i]<=a[j]){

b[k++]=a[i++];

count++;

}

else{

b[k++]=a[j++];

count++;

}

}

while(i<=mid){

b[k++]=a[i++];

count++;

}

while(j<=high){

b[k++]=a[j++];

count++;

}

for(i=0;i<=high;i++){

a[i]=b[i];

}

}

void mergesort(int a[MAX],int low,int high){

if(low<high){

int mid=(low+high)/2;

mergesort(a,low,mid);

mergesort(a,mid+1,high);

merge(a,low,mid,high);

}

}

int main(){

clock\_t start,end;

int a[MAX],b[MAX],c[MAX];

int n,i,j,c1,c2,c3;

printf("Enter no of elements:");

scanf("%d",&n);

printf("Enter Array elements:\n");

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

scanf("%d",&a[i]);

}

start=clock();

mergesort(a,0,n-1);

end=clock();

printf("\nSorted Array:");

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

printf("%d\t",a[i]);

}

printf("\nTIME TAKEN TO SORT IS:%fs",(double)(end-start)/CLOCKS\_PER\_SEC);

printf("\ncount=%d",count);

printf("\nSIZE\tASC\tDESC\tRAND\n");

for(i=16;i<MAX;i\*=2){

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

a[j]=j;

b[j]=i-j-1;

c[j]=rand() %i;

}

count=0;

mergesort(a,0,i-1);

c1=count;

count=0;

mergesort(b,0,i-1);

c2=count;

count=0;

mergesort(c,0,i-1);

c3=count;

printf("%d\t%d\t%d\t%d\n",i,c1,c2,c3);

}

return 0;

}

**HEAPSORT**

#include<stdio.h>

#include<stdlib.h>

#include<time.h>

#define MAX 1000

void heapify(int a[MAX],int n){

int i,j,k,key,heap;

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

k=i;

key=a[k];

heap=0;

while(!heap && 2\*k<=n){

j=2\*k;

if(j<n){

if(a[j]<a[j+1]){

j=j+1;

}

}

if(key>=a[j]){

heap=1;

}

else{

a[k]=a[j];

k=j;

}

}

a[k]=key;

}

}

void heapsort(int a[MAX],int n){

int i,temp;

heapify(a,n);

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

temp=a[1];

a[1]=a[i];

a[i]=temp;

heapify(a,i-1);

}

}

int main(){

clock\_t start,end;

int a[MAX],n,i;

printf("Enter no of elelments:");

scanf("%d",&n);

printf("\nEnter array elements:\n");

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

scanf("%d",&a[i]);

}

start=clock();

heapsort(a,n);

end=clock();

printf("\nTIME TAKEN TO SORT IS:%fs",(double)(end-start)/CLOCKS\_PER\_SEC);

printf("\n Sorted array:");

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

printf("%d\t",a[i]);

}

return 0;

}

**HORSEPOOL**

#include<stdio.h>

#include<stdlib.h>

#include<string.h>

#define MAX 256

int table[MAX];

int count=0;

void shifttable(char pat[MAX]){

int i,j,m;

m=strlen(pat);

for(i=0;i<MAX;i++){

table[i]=m;

}

for(j=0;j<m;j++){

table[pat[j]]=m-1-j;

}

}

int horsepool(char src[MAX],char pat[MAX]){

int i,j,k,m,n;

n=strlen(src);

m=strlen(pat);

i=m-1;

while(i<n){

k=0;

while((k<m) && (pat[m-1-k]==src[i-k])){

k++;

}

if(k==m)

return i-m+1;

else

count+=table[src[i]];

i=i+table[src[i]];

}

return -1;

}

int main(){

char src[MAX],pat[MAX];

int pos;

printf("\nEnter source string:");

scanf("%s",src);

printf("\nEnter pattern to be searched:");

scanf("%s",pat);

shifttable(pat);

pos=horsepool(src,pat);

if(pos!=-1){

printf("\nFound at %d position",pos+1);

}

else

printf("\npattern not found");

printf("\nshifts=%d",count);

return 0;

}

**TOPOLOGICAL ORDERING**

#include<stdio.h>

#include<stdlib.h>

#define MAX 1000

int k=1,res[MAX],isCyclic=0;

void topo(int a[MAX][MAX],int vis[MAX],int n,int source){

vis[source]=2;

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

if(vis[i]==0 && a[source][i]==1){

topo(a,vis,n,i);

}

else if(vis[i]==2 && a[source][i]==1){

isCyclic=1;

return;

}

}

vis[source]=1;

res[k++]=source;

}

int main(){

int n,i,j,a[MAX][MAX],vis[MAX];

printf("\nEnter no of nodes:");

scanf("%d",&n);

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

vis[i]=0;

}

printf("\nEnter Adjacency matrix:\n");

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

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

scanf("%d",&a[i][j]);

}

}

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

if(vis[i]==0){

topo(a,vis,n,i);

}

}

if(isCyclic==1){

printf("\nIt is cyclic");

return 0;

}

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

printf("%d\t",res[i]);

}

return 0;

}

**NQUEENS**

#include<stdio.h>

#include<stdlib.h>

#define MAX 100

int x[MAX];

int solutions=1;

void printboard(int n){

int i,j;

printf("\nsolution %d\n",solutions++);

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

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

if(x[i]==j){

printf("Q\t");

}

else

printf("-\t");

}

printf("\n");

}

}

int place(int k,int i){

for(int j=1;j<k;j++){

if((x[j]==i) || abs(x[j]-i)==abs(j-k))

return 0;

}

return 1;

}

void nqueen(int k,int queens){

for(int i=1;i<=queens;i++){

if(place(k,i)){

x[k]=i;

if(k==queens){

printboard(queens);

}

else

nqueen(k+1,queens);

}

}

}

int main(){

int queens;

printf("Enter no of Queens:");

scanf("%d",&queens);

nqueen(1,queens);

if(solutions==1)

printf("no solutions!");

return 0;

}

**SUM OF SUBSET**

#include<stdio.h>

#include<stdlib.h>

#define MAX 100

int subset[MAX],x[MAX],match;

int solutions=1;

void sum\_subset(int s,int k,int total){

x[k]=1;

if(s+subset[k]==match){

printf("SUBSET %d\n",solutions++);

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

if(x[i]==1)

printf("%d\t",subset[i]);

}

printf("\n");

}

else if(s+subset[k]+subset[k+1]<=match)

sum\_subset(s+subset[k],k+1,total-subset[k]);

if((s+total-subset[k]>=match)&&(s+subset[k+1]<=match)){

x[k]=0;

sum\_subset(s,k+1,total-subset[k]);

}

}

int main(){

int n,i,sum=0;

printf("Enter no of elements:");

scanf("%d",&n);

printf("Enter %d elements in ascending order:\n",n);

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

scanf("%d",&subset[i]);

sum+=subset[i];

}

printf("Enter subset match value:");

scanf("%d",&match);

if(sum<match || subset[1]>match){

printf("no Solution!");

exit(0);

}

sum\_subset(0,1,sum);

return 0;

}

**KNAPSACK**

#include<stdio.h>

#include<stdlib.h>

#define SIZE 100

int w[SIZE],p[SIZE],v[SIZE][SIZE],x[SIZE];

int max(int a,int b){

return(a>b)? a:b;

}

int knapsack(int n,int m){

int i,j;

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

for(j=0;j<=m;j++){

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

v[i][j]=0;

}

else if(j-w[i]<0){

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

}

else{

v[i][j]=max(v[i-1][j],p[i]+v[i-1][j-w[i]]);

}

}

}

return v[n][m];

}

void display(int n,int m){

int i,j;

i=n;j=m;

while(i>0 || j>0){

if(v[i][j]!=v[i-1][j]){

x[i]=1;

j=j-w[i];

}

i--;

}

}

void printtable(int n,int m){

int i,j;

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

for(j=0;j<=m;j++){

printf("%4d",v[i][j]);

}

printf("\n");

}

}

int main(){

int i,n,m,profit=0;

printf("Enter no of nodes:");

scanf("%d",&n);

printf("Enter weights and profits:\n");

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

printf("Weight[%d]=",i);

scanf("%d",&w[i]);

printf("Profit[%d]=",i);

scanf("%d",&p[i]);

}

printf("\nEnter Knapsack capacity:");

scanf("%d",&m);

profit=knapsack(n,m);

printtable(n,m);

printf("\nThe max profit is:%d",profit);

display(n,m);

printf("\nThe included objects are:");

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

if(x[i]==1){

printf("\nThe included object is w[%d]=%d",w[i],p[i]);

}

}

return 0;

}