1. Print all the nodes reachable from a given starting node in a digraph using **BFS**. Give the trace of this algorithm.

#include<stdio.h>

void BFS(int[20][20],int,int[20],int);

void main()

{

int n,a[20][20],i,j,visited[20],source;

printf("enter number of vertices:");

scanf("%d",&n);

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++)

visited[i]=0;

printf("\nEnter the source node:");

scanf("%d",&source);

visited[source]=1;

BFS(a,source,visited,n);

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

{

if(visited[i]!=0)

printf("\nnode %d is reachable",i);

else

printf("\nnode %d is not reachable",i);

}

}

void BFS(int a[20][20], int source, int visited[20], int n)

{

int queue[20],f,r,u,v;

f=0;r=-1;

queue[++r]=source;

while(f<=r)

{

u=queue[f++];

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

{

if(a[u][v]==1&&visited[v]==0)

{

queue[++r]=v;

visited[v]=1;

}

}

}

}

**Output:**

Enter no. of vertices; 4

Enter adjacency matrix

0 1 1 1

0 0 0 0

0 0 0 1

0 0 0 0

Enter the source node: 3

Node 1 not reachable

Node 2 not reachable

Node 3 reachable

Node 4 reachable

1. Implement   **0/1    Knapsack**   problem   using   dynamic programming. Give the trace of this algorithm.

#include<stdio.h>

int max(int a,int b)

{

if(a>b)

return a;

else

return b;

}

void knapsack(int m,int n,int w[],int p[])

{

int v[100][200],x[10],i,j,sum=0;

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

v[0][i]=0;

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

v[0][i]=0;

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

{

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

{

if(j>=w[i])

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

else

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

}

}

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

x[i]=0;

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--;

}

printf("\nThe optimal set of items is:\n");

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

{

if(x[i]==1)

{

printf("%d\t",i);

sum=sum+p[i];

}

}

printf("\n\ntotal profit=%d",sum);

}

void main()

{

int w[10],p[10],m,n,i;

printf("enter no of items:");

scanf("%d",&n);

printf("enter the weights of the items:\n");

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

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

printf("enter the profits:\n");

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

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

printf("enter the capacity of knapsack:");

scanf("%d",&m);

knapsack(m,n,w,p);

}

**Output:**

Enter no. of items:

5

Enter the weights

2 5 2 6 9

Enter profits:

50 40 20 60 55

Enter Capacity: 6

Optimal set of items:

1 3

Total profit = 70

1. Find Minimum Cost Spanning Tree of a given undirected graph using **Prim’s/Kruskal’s** algorithm. Give the trace of this algorithm.

//Prims algorithm for minimum cost spanning tree

#include<stdio.h>

int ne=1,min\_cost=0;

void main()

{

int n,i,j,min,cost[20][20],a,u,b,v,source,visited[20];

printf("enter the number of nodes:");

scanf("%d",&n);

printf("enter the cost matrix:\n");

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

{

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

{

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

}

}

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

visited[i]=0;

printf("enter the root node:");

scanf("%d",&source);

visited[source]=1;

printf("minimum cost spanning tree is\n");

while(ne<n)

{

min=999;

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

{

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

{

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

if(visited[i]==0)

continue;

else

{

min=cost[i][j];

a=u=i;

b=v=j;

}

}

}

if(visited[u]==0||visited[v]==0)

{

printf("\nedge%d\t(%d->%d)=%d\n",ne++,a,b,min);

min\_cost=min\_cost+min;

visited[b]=1;

}

cost[a][b]=cost[b][a]=999;

}

printf("\nminimum cost=%d\n",min\_cost);

}

Enter no of vertices: 4

Enter the cost matrix:

0 1 5 2

1 0 0 0

5 0 0 3

2 0 3 0

Edges of spanning tree are:

Edge 1->2 = 1

Edge 1->4 = 2

Edge 3->4 = 3

Min cost = 6

**//Kruskal algorithm**

#include<stdio.h>

int ne=1,min\_cost=0;

void main()

{

int n,i,j,min,a,u,b,v,cost[20][20],parent[20];

printf("enter no of vertices:");

scanf("%d",&n);

printf("enter the cost matrix:\n");

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

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

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

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

parent[i]=0;

printf("\nthe edges of spanning tree are\n");

while(ne<n)

{

min=999;

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

{

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

{

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

{

min=cost[i][j];

a=u=i;

b=v=j;

}

}

}

while(parent[u])

u=parent[u];

while(parent[v])

v=parent[v];

if(u!=v)

{

printf("Edge%d\t(%d->%d)=%d\n",ne++,a,b,min);

min\_cost=min\_cost+min;

parent[v]=u;

}

cost[a][b]=cost[a][b]=999;

}

printf("\nmin\_cost=%d\n",min\_cost);

}

**Output:**

Enter no of vertices: 4

Enter the cost matrix:

0 1 5 2

1 0 0 0

5 0 0 3

2 0 3 0

Edges of spanning tree are:

Edge 1->2 = 1

Edge 1->4 = 2

Edge 3->4 = 3

Min cost = 6

1. From a given vertex in a weighted connected graph, find shortest paths to other vertices using **Dijkstra's** algorithm. Give the trace of this algorithm.

#include<stdio.h>

void dijkstra(int,int [20][20],int [20],int [20],int);

void main()

{

int i,j,n,visited[20],source,cost[20][20],d[20];

printf("enter no of vertices:");

scanf("%d",&n);

printf("enter the cost adjacency matrix\n");

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

{

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

{

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

}

}

printf("\nenter the source node:");

scanf("%d",&source);

dijkstra(source,cost,visited,d,n);

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

if(i!=source)

printf("\nshortest path from %d to %d is %d",source,i,d[i]);

}

void dijkstra(int source,int cost[20][20],int visited[20],int d[20],int n)

{

int i,j,min,u,w;

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

{

visited[i]=0;

d[i]=cost[source][i];

}

visited[source]=1;

d[source]=0;

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

{

min=999;

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

{

if(!visited[i])

{

if(d[i]<min)

{

min=d[i];

u=i;

}

}

}

visited[u]=1;

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

{

if(cost[u][w]!=999&&visited[w]==0)

{

if(d[w]>cost[u][w]+d[u])

d[w]=cost[u][w]+d[u];

}

}

}

}

Output:

Enter no of vertices: 5

Enter adjacency matrix:

0 1 0 1 0

1 0 0 1 0

0 1 0 1 1

1 1 1 0 1

0 0 1 1 0

Enter the source node: 1

Shortest path from 1 to 2 is 3

Shortest path from 1 to 4 is 5

Shortest path from 1 to 3 is 7

Shortest path from 1 to 5 is 9

1. Sort a given set of elements using the **Heap Sort** method and determine the time required to sort the elements. Plot a graph of number of elements versus time taken. Specify the time efficiency class of this algorithm.

#include<stdio.h>

#include<math.h>

#include<time.h>

int h[50];

void heapify(int h[],int n)

{

int i,j,n1,k,heap,v;

n1=n;

for(i=n1/2;i>=1;i--)

{

k=i;

v=h[k];

heap=0;

while(!(heap)&&(2\*k)<=n1)

{

j=2\*k;

if(j<n1)

if(h[j]<h[j+1])

j=j+1;

if(v>=h[j])

heap=1;

else

{

h[k]=h[j];

k=j;

}

h[k]=v;

}

}

}

void main()

{

int i,n,size,t;

clock\_t start,end;

double totaltime;

printf("enter value of n\n");

scanf("%d",&n);

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

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

{

h[i]=rand();

}

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

printf("elements\n");

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

{

printf("%d\n",h[i]);

}

start=clock();

heapify(h,n);

printf("elemts heapified:\n");

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

printf("%d\n",h[i]);

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

{

t=h[1];

h[1]=h[i];

h[i]=t;

size=i-1;

heapify(h,size);

}

end=clock();

printf("elemtents sorted\n");

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

printf("%d\n",h[i]);

totaltime=(double)(end-start)/CLOCKS\_PER\_SEC;

printf("time taken is %e",totaltime);

}

**Output:**

Enter value of n: 4

Enter array elements:

2 1 3 4

Elements heapified:

4 2 3 1

Sorted elements;

1 2 3 4

1. Implement **N-Queen's** problem using Back Tracking. Give the trace of this algorithm.

#include<stdio.h>

void nqueens(int);

int place(int[],int);

void printsolution(int,int[]);

void main()

{

int n;

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

scanf("%d",&n);

nqueens(n);

}

void nqueens(int n)

{

int x[10],count=0,k=1;

x[k]=0;

while(k!=0)

{

x[k]=x[k]+1;

while(x[k]<=n&&(!place(x,k)))

x[k]=x[k]+1;

if(x[k]<=n)

{

if(k==n)

{

count++;

printf("\nsolution %d\n",count);

printsolution(n,x);

}

else

{

k++;

x[k]=0;

}

}

else

{

k--;

}

}

return;

}

int place(int x[], int k)

{

int i;

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

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

return 0;

return 1;

}

void printsolution(int n,int x[])

{

int i,j;

char c[10][10];

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

{

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

c[i][j]='X';

}

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

c[i][x[i]]='Q';

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

{

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

{

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

}

printf("\n");

}

}

**Output:**

Enter no of queens: 4

Solution1:

X Q X X

X X X Q

Q X X X

X X Q X

Solution2:

X X Q X

Q X X X

X X X Q

X Q X X