**VISVESVARAYA TECHNOLOGICAL UNIVERSITY**

**“JnanaSangama”, Belgaum -590014, Karnataka.**

****

**LAB REPORT**

**on**

**Analysis and Design of Algorithms**

***Submitted by***

**GAMANA YELURI R (1BM21CS065)**

***in partial fulfillment for the award of the degree of***

**BACHELOR OF ENGINEERING**

***in***

**COMPUTER SCIENCE AND ENGINEERING**



**B.M.S. COLLEGE OF ENGINEERING**

**(Autonomous Institution under VTU)**

**BENGALURU-560019**

**June-2023 to September-2023**

**B. M. S. College of Engineering,**

**Bull Temple Road, Bangalore 560019**

(Affiliated To Visvesvaraya Technological University, Belgaum)

**Department of Computer Science and Engineering**



**CERTIFICATE**

This is to certify that the Lab work entitled “**Analysis and Design of Algorithms**” carried out by **GAMANA YELURI R (1BM21CS065),** who is a bonafide student of **B.M.S. College of Engineering.** It is in partial fulfillment for the award of **Bachelor of Engineering in Computer Science and Engineering** of the Visvesvaraya Technological University, Belgaum during the academic semester June-2023 to September-2023. The Lab report has been approved as it satisfies the academic requirements in respect of a **Analysis and Design of Algorithms (22CS4PCADA)** work prescribed for the said degree.

Name of the Lab-In charge:               Dr. Jyothi S Nayak

Designation Professor and Head

Department of CSE Department of CSE

BMSCE, Bengaluru BMSCE, Bengaluru

**Index Sheet**

| **Lab Program No.** | **Program Details** | **Page No.** |
| --- | --- | --- |
| 1 | Write program to do the following:  a. Print all the nodes reachable from a given starting node in a digraph using the BFS method.  b. Check whether a given graph is connected or not using the DFS method. | 4 |
| 2 | Write a program to obtain the Topological ordering of vertices in a given digraph. | 9 |
| 3 | Implement Johnson Trotter algorithm to generate permutations. | 12 |
| 4 | Sort a given set of N integer elements using Merge Sort technique and compute its time taken. | 16 |
| 5 | Sort a given set of N integer elements using Quick Sort technique and compute its time taken. | 19 |
| 6 | Implement 0/1 Knapsack problem using dynamic programming. | 21 |
| 7 | Implement All Pair Shortest paths problem using Floyd’s algorithm. | 24 |
| 8 | Find Minimum Cost Spanning Tree of a given undirected graph using Prim’s and Kruskal’s algorithm. | 26 |
| 9 | From a given vertex in a weighted connected graph, find shortest paths to other vertices using Dijkstra’s algorithm. | 32 |
| 10 | Implement “N-Queens Problem” using Backtracking. | 35 |
| 11 | Sort a given set of N integer elements using Heap Sort technique. | 38 |

**Course Outcome**

| CO1 | Analyze time complexity of Recursive and Non-recursive algorithms using asymptotic notations. |
| --- | --- |
| CO2 | Apply various design techniques for the given problem. |
| CO3 | Apply the knowledge of complexity classes P, NP, and NP-Complete and prove certain  problems are NP-Complete |
| CO4 | Design efficient algorithms and conduct practical experiments to solve problems. |

**WEEK 1**

**Write program to do the following:**

**a. Print all the nodes reachable from a given starting node in a digraph using**

**BFS method.**

**b. Check whether a given graph is connected or not using the DFS method.**

**a)BFS**

Code:

#include<stdio.h>

#include<conio.h>

int a[15][15],n;

void bfs(int);

void main() {

int i,j,src;

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

scanf("%d",&n);

printf("\nEnter the adjacency matrix:\n");

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

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

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

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

scanf("%d",&src);

bfs(src);

}

void bfs(int src) {

int q[15],f=0,r=-1,vis[15],i,j;

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

vis[j]=0;

vis[src]=1;

r=r+1;

q[r]=src;

while(f<=r) {

i=q[f];

f=f+1;

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

{

if(a[i][j]==1&&vis[j]!=1) {

vis[j]=1;

r=r+1;

q[r]=j;

}

}

}

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

if(vis[j]!=1)

printf("\nNode %d is not reachable",j);

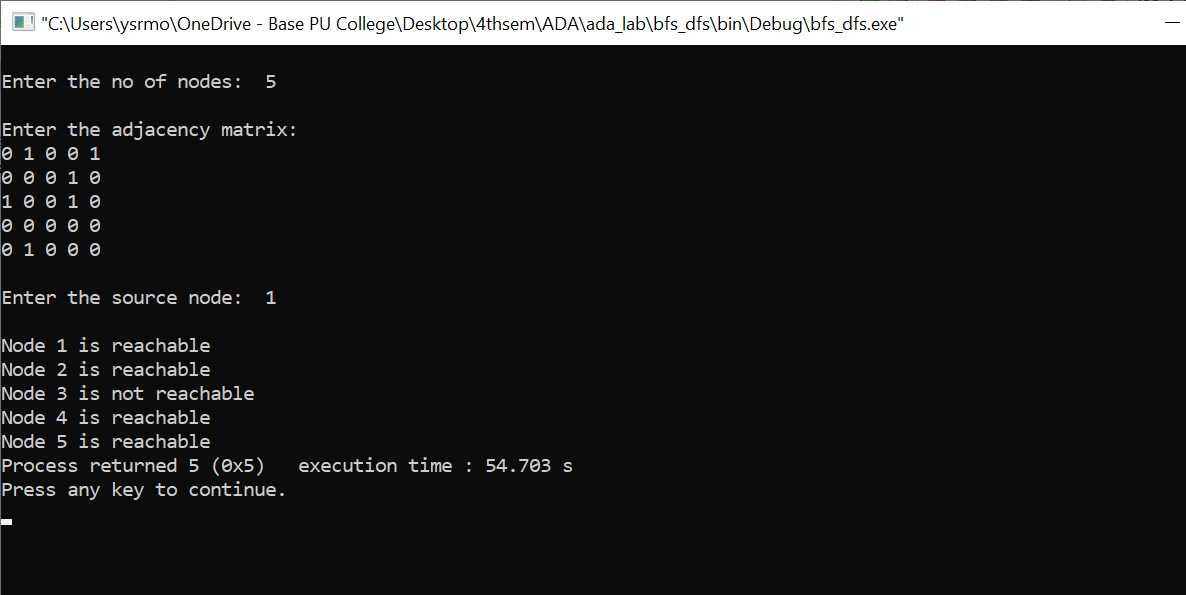
else

printf("\nNode %d is reachable",j);

}

}

Output:



**b)DFS**

Code:

#include<stdio.h>

#include<conio.h>

int a[10][10],n,vis[10];

int dfs(int src){

int j;

vis[src]=1;

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

if(a[src][j]==1&&vis[j]!=1)

dfs(j);

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

if(vis[j]!=1)

return 0;

}

return 1;

}

void main()

{

int i,j,src,ans;

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

vis[j]=0;

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

scanf("%d",&n);

printf("\nEnter the adjacency matrix:\n");

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

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

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

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

scanf("%d",&src);

ans=dfs(src);

if(ans==1)

printf("\nGraph is connected\n");

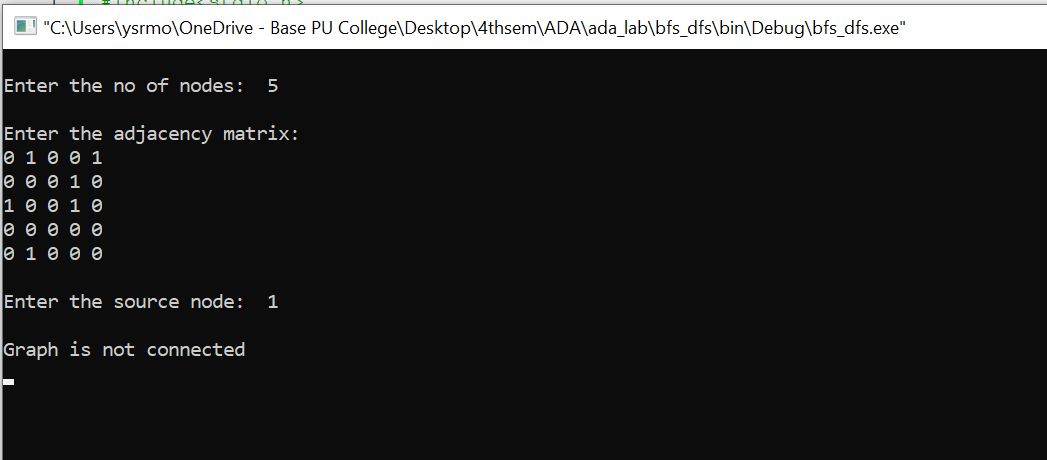
else

printf("\nGraph is not connected\n");

getch();

}

Output:



**WEEK 2**

**Write program to obtain the Topological ordering of vertices in a given**

**digraph.**

Code:

#include<stdio.h>

#include<conio.h>

void dfs(int n, int a[10][10]) {

int i,j,k,u,v,top,s[10],t[10],indeg[10],sum;

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

sum=0;

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

sum+=a[j][i];

indeg[i]=sum;

}

top=-1;

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

if(indeg[i]==0)

s[++top]=i;

}

k=0;

while(top!=-1) {

u=s[top--];

t[k++]=u;

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

if(a[u][v]==1) {

indeg[v]=indeg[v]-1;

if(indeg[v]==0)

s[++top]=v;

}

}

}

printf("Topological order :");

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

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

}

void main() {

int i,j,a[10][10],n;

printf("Enter number of nodes\n");

scanf("%d", &n);

printf("Enter the adjacency matrix\n");

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

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

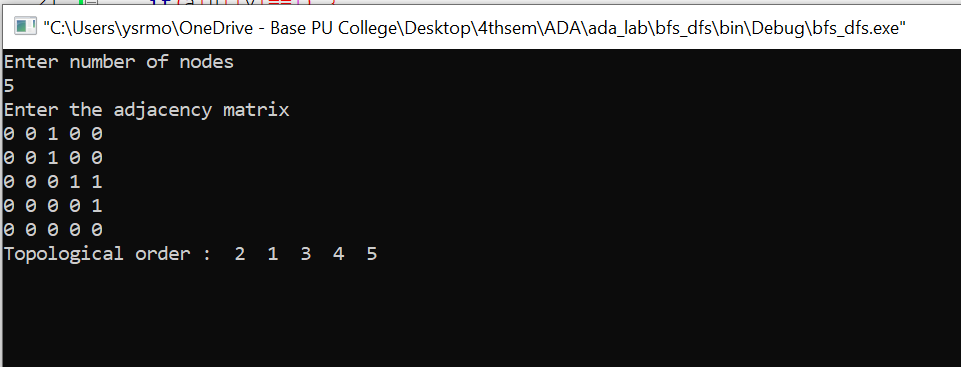
scanf("%d", &a[i][j]);

dfs(n,a);

getch();

}

Output:



**WEEK 3**

**Implement Johnson Trotter algorithm to generate permutations.**

CODE:

#include <stdio.h>

#include <stdlib.h>

int flag = 0;

int swap(int \*a,int \*b) {

int t = \*a;

\*a = \*b;

\*b = t;

}

int search(int arr[],int num,int mobile)

{

int g;

for(g=0;g<num;g++) {

if(arr[g] == mobile)

return g+1;

else

flag++;

}

return -1;

}

int find\_Moblie(int arr[],int d[],int num)

{

int mobile = 0;

int mobile\_p = 0;

int i;

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

{

if((d[arr[i]-1] == 0) && i != 0)

{

if(arr[i]>arr[i-1] && arr[i]>mobile\_p)

{

mobile = arr[i];

mobile\_p = mobile;

}

else

flag++;

}

else if((d[arr[i]-1] == 1) & i != num-1)

{

if(arr[i]>arr[i+1] && arr[i]>mobile\_p)

{

mobile = arr[i];

mobile\_p = mobile;

}

else

flag++;

}

else

flag++;

}

if((mobile\_p == 0) && (mobile == 0))

return 0;

else

return mobile;

}

void permutations(int arr[],int d[],int num)

{

int i;

int mobile = find\_Moblie(arr,d,num);

int pos = search(arr,num,mobile);

if(d[arr[pos-1]-1]==0)

swap(&arr[pos-1],&arr[pos-2]);

else

swap(&arr[pos-1],&arr[pos]);

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

{

if(arr[i] > mobile)

{

if(d[arr[i]-1]==0)

d[arr[i]-1] = 1;

else

d[arr[i]-1] = 0;

}

}

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

{

printf(" %d ",arr[i]);

} }

int factorial(int k)

{

int f = 1;

int i = 0;

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

f = f\*i;

return f;

}

int main()

{

int num = 0;

int i;

int j;

int z = 0;

printf("Enter the number\n");

scanf("%d",&num);

int arr[num],d[num];

z = factorial(num);

printf("total permutations = %d",z);

printf("\npossible permutations: \n");

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

{

d[i] = 0;

arr[i] = i+1;

printf(" %d ",arr[i]);

}

printf("\n");

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

permutations(arr,d,num);

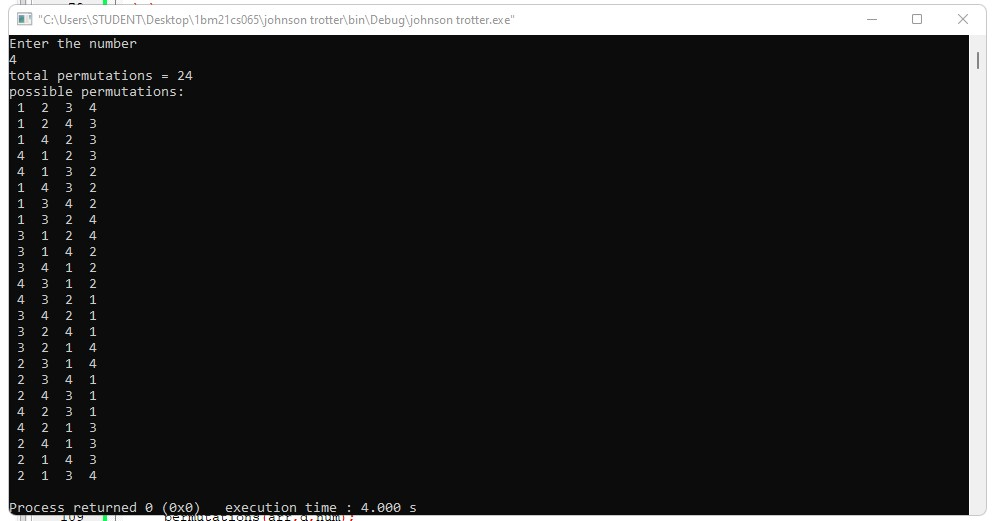
printf("\n");

}

return 0;

}

OUTPUT:



**WEEK 4**

**Sort a given set of N integer elements using Merge Sort technique.**

CODE:

#include <stdio.h>

#include <stdlib.h>

void merge(int low,int mid,int high,int array[20],int mer[20])

{

int i = low;

int j = mid+1;

int k = 0;

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

{

if(array[i]<array[j])

{

mer[k] = array[i];

i++;

k++;

}

else

{

mer[k] = array[j];

j++;

k++;

}

}

while (i <= mid)

{

mer[k] = array[i];

i++;

k++;

}

while (j <= high)

{

mer[k] = array[j];

j++;

k++;

}

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

{

array[low+i] = mer[i];

}

}

void merge\_sort(int low,int high,int array[20],int merged[20])

{

if(low<high)

{

int mid = (low+high)/2;

merge\_sort(low,mid,array,merged);

merge\_sort(mid+1,high,array,merged);

merge(low,mid,high,array,merged);

}

}

int main()

{

int n,array[30];

printf("Enter no. of elements:");

scanf("%d",&n);

printf("Enter elements:");

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

{

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

}

int merged[30];

merge\_sort(0,n-1,array,merged);

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

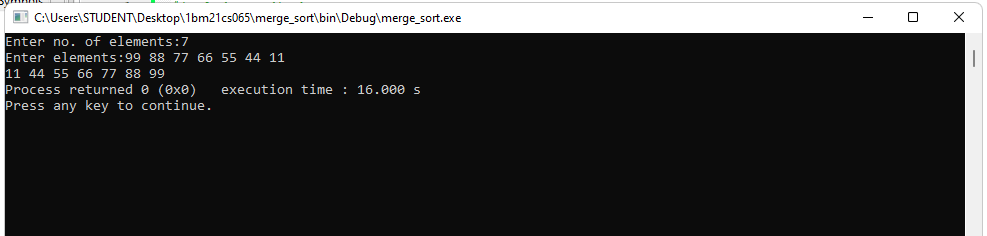
{

printf("%d ",array[i]);

}

}

OUTPUT:



**WEEK 5**

**Sort a given set of N integer elements using Quick Sort technique.**

CODE:

#include<stdio.h>

void quicksort(int number[25],int first,int last)

{

int i, j, pivot, temp;

if(first<last)

{

pivot=first;

i=first;

j=last;

while(i<j)

{

while(number[i]<=number[pivot]&&i<last)

i++;

while(number[j]>number[pivot])

j--;

if(i<j)

{

temp=number[i];

number[i]=number[j];

number[j]=temp;

}

}

temp=number[pivot];

number[pivot]=number[j];

number[j]=temp;

quicksort(number,first,j-1);

quicksort(number,j+1,last);

}

}

int main()

{

int i, count, number[25];

printf("enter no of elements : ");

scanf("%d",&count);

printf("Enter %d elements: ", count);

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

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

quicksort(number,0,count-1);

printf("Sorted elements: ");

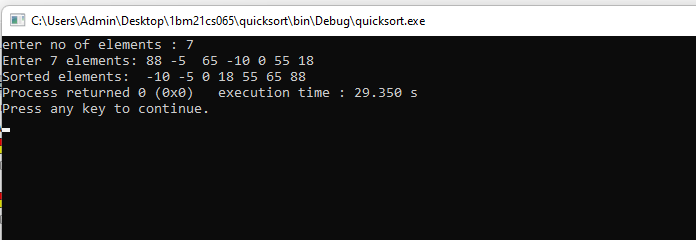
for(i=0;i<count;i++)

printf(" %d",number[i]);

return 0;

}

OUTPUT:



**WEEK 6**

**Implement 0/1 Knapsack problem using dynamic programming.**

CODE:

#include <stdio.h>

#include <conio.h>

void knapsack();

int max(int, int);

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

void main()

{

printf("\nEnter the no. of items:\n");

scanf("%d", &n);

printf("\nEnter the weight of the each item:\n");

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

{

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

}

printf("\nEnter the profit of each item:\n");

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

{

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

}

printf("\nEnter the knapsack's capacity:\n");

scanf("%d", &m);

knapsack();

getch();

}

void knapsack()

{

int x[10];

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

}

}

}

printf("\nThe output is:\n");

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

{

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

{

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

}

printf("\n\n");

}

printf("\nThe optimal solution is %d", v[n][m]);

printf("\nThe solution vector is:\n");

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

{

if (v[i][m] != v[i - 1][m])

{

x[i] = 1;

m = m - w[i];

}

else

{

x[i] = 0;

}

}

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

{

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

}

}

int max(int x, int y)

{

if (x > y)

{

return x;

}

else

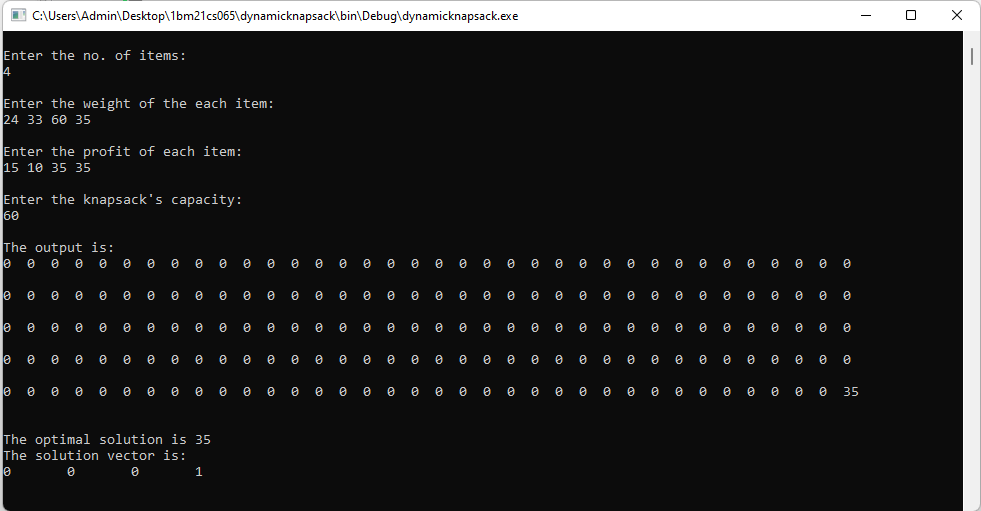
{

return y;

}

}

OUTPUT:



**WEEK 7**

**Implement All Pair Shortest paths problem using Floyd’s algorithm.**

CODE:

#include<stdio.h>

void main()

{

int i,j,k,n,p[10][10],o[10][10];

printf("Enter number of nodes \n");

scanf("%d",&n);

printf("Enter %dX%d adjacency matrix of \n",n,n);

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

{

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

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

}

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

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

o[i][j]=p[i][j];

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

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

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

if(p[i][j] > p[k][j]+p[i][k])

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

printf("\nOriginal Adjacency Matrix \n");

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

{

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

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

printf("\n");

}

printf("\nUpdated Adjacency Matrix \n");

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

{

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

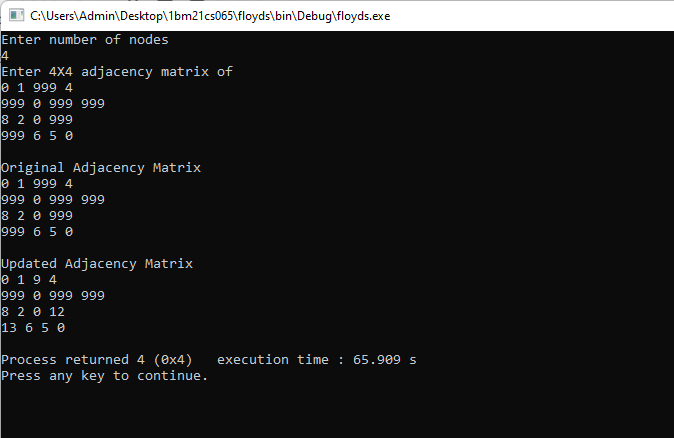
printf("%d ",p[i][j]);

printf("\n");

}

}

OUTPUT:



**WEEK 8**

**Find the minimum cost spanning tree of a given undirected graph using prims and Kruskal's algorithm.**

**PRIMS:**

CODE:

#include<stdio.h>

float cost[10][10];

int vt[10],et[10][10],vis[10],j,n;

float sum=0;

int x=1;

int e=0;

void prims();

void main()

{

int i;

printf("enter the number of vertices\n");

scanf("%d",&n);

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

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

{

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

{

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

}

vis[i]=0;

}

prims();

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

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

{

printf("%d,%d\t",et[i][0],et[i][1]);

}

printf("weight=%f\n",sum);

}

void prims()

{

int s,m,k,u,v;

float min;

vt[x]=1;

vis[x]=1;

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

{

j=x;

min=999;

while(j>0)

{

k=vt[j];

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

{

if(vis[m]==0)

{

if(cost[k][m]<min)

{

min=cost[k][m];

u=k;

v=m;

}

}

}

j--;

}

vt[++x]=v;

et[s][0]=u;

et[s][1]=v;

e++;

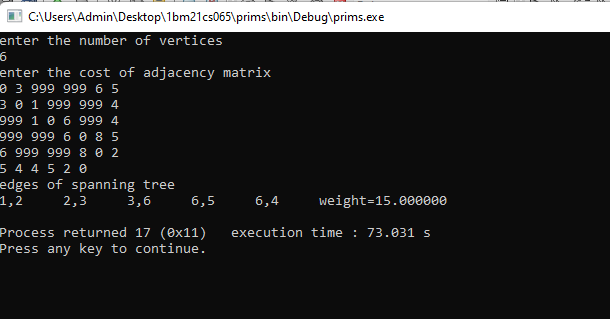
vis[v]=1;

sum=sum+min;

}

}

OUTPUT:



**KRUSHKAL’S:**

CODE:

#include <stdio.h>

#include <conio.h>

#include <stdlib.h>

int i,j,k,a,b,u,v,n,ne=1;

int min,mincost=0,cost[9][9],parent[9];

int find(int);

int uni(int,int);

void main()

{

printf("\nEnter the no. of vertices:");

scanf("%d",&n);

printf("\nEnter the cost of adjacency matrix:\n");

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

{

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

{

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

if(cost[i][j]==0)

cost[i][j]=999;

}

}

printf("The edges of Minimum Cost Spanning Tree are\n");

while(ne < n)

{

for(i=1,min=999;i<=n;i++)

{

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

{

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

{

min=cost[i][j];

a=u=i;

b=v=j;

}

}

}

u=find(u);

v=find(v);

if(uni(u,v))

{

printf("%d edge (%d,%d) =%d\n",ne++,a,b,min);

mincost +=min;

}

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

}

printf("\nMinimum cost = %d\n",mincost);

getch();

}

int find(int i)

{

while(parent[i])

i=parent[i];

return i;

}

int uni(int i,int j)

{

if(i!=j)

{

parent[j]=i;

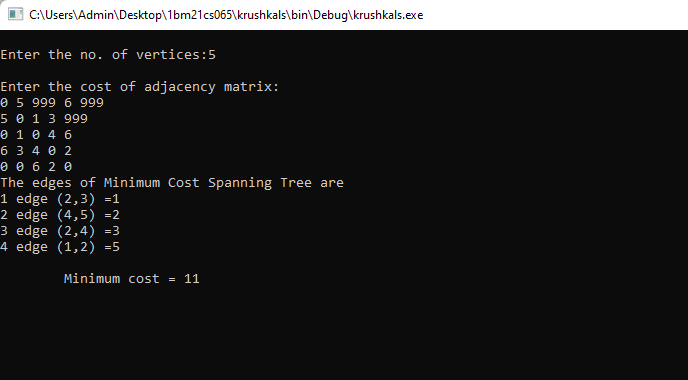
return 1;

}

return 0;

}

OUTPUT:



**WEEK 9**

**From a given vertex in a weighted connected graph, find shortest paths to other vertices using dijkstra's algorithm.**

CODE:

#include<stdio.h>

#include<conio.h>

#define INFINITY 999

#define MAX 10

void dijkstra(int G[MAX][MAX],int n,int startnode);

int main()

{

int G[MAX][MAX],i,j,n,u;

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

scanf("%d",&n);

printf("\nEnter the adjacency matrix:\n");

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

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

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

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

scanf("%d",&u);

dijkstra(G,n,u);

return 0;

}

void dijkstra(int G[MAX][MAX],int n,int startnode)

{

int cost[MAX][MAX],distance[MAX],pred[MAX];

int visited[MAX],count,mindistance,nextnode,i,j;

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

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

if(G[i][j]==0)

cost[i][j]=INFINITY;

else

cost[i][j]=G[i][j];

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

{

distance[i]=cost[startnode][i];

pred[i]=startnode;

visited[i]=0;

}

distance[startnode]=0;

visited[startnode]=1;

count=1;

while(count<n-1)

{

mindistance=INFINITY;

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

if(distance[i]<mindistance&&!visited[i])

{

mindistance=distance[i];

nextnode=i;

}

visited[nextnode]=1;

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

if(!visited[i])

if(mindistance+cost[nextnode][i]<distance[i])

{

distance[i]=mindistance+cost[nextnode][i];

pred[i]=nextnode;

}

count++;

}

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

if(i!=startnode)

{

printf("\nDistance of node%d=%d",i,distance[i]);

printf("\nPath=%d",i);

j=i;

do

{

j=pred[j];

printf("<-%d",j);

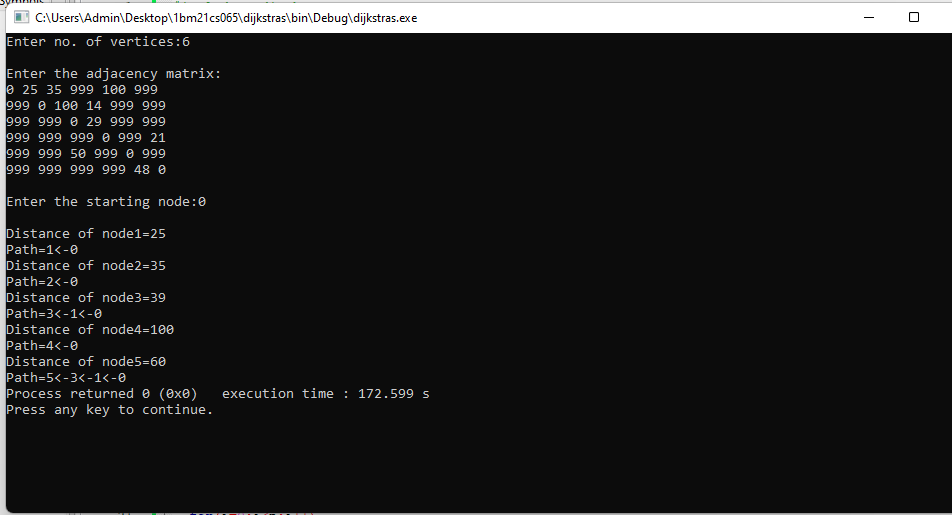
}

while(j!=startnode);

}

}

OUTPUT:



**WEEK 10**

**Implement “N-Queens Problem” using Backtracking.**

CODE:

#include<stdio.h>

#include<math.h>

int board[20],count;

int main()

{

int n,i,j;

void queen(int row,int n);

printf("\n\nEnter no of Queens:");

scanf("%d",&n);

queen(1,n);

return 0;

}

void print(int n)

{

int i,j;

printf("\n\nOutput %d:\n\n",++count);

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

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

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

{

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

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

{

if(board[i]==j)

printf("\tQ");

else

printf("\t-");

}

}

}

int place(int row,int column)

{

int i;

for(i=1;i<=row-1;++i)

{

if(board[i]==column)

return 0;

else

if(abs(board[i]-column)==abs(i-row))

return 0;

}

return 1;

}

void queen(int row,int n)

{

int column;

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

{

if(place(row,column))

{

board[row]=column;

if(row==n)

print(n);

else

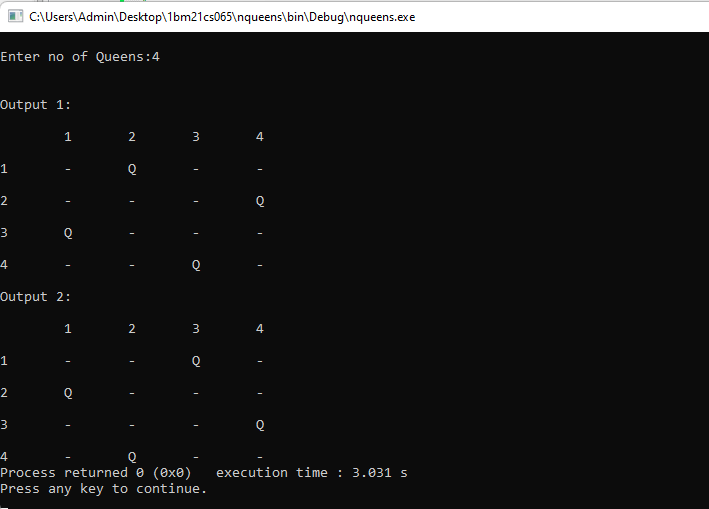
queen(row+1,n);

}

}

}

OUTPUT:



**WEEK 11**

**Sort a given set of N integer elements using Heap Sort technique.**

CODE:

#include <stdio.h>

void heapify(int arr[], int n, int i) {

int largest = i, left = 2 \* i + 1, right = 2 \* i + 2;

if (left < n && arr[left] > arr[largest])

largest = left;

if (right < n && arr[right] > arr[largest])

largest = right;

if (largest != i) {

int temp = arr[i];

arr[i] = arr[largest];

arr[largest] = temp;

heapify(arr, n, largest);

}

}

void heapsort(int arr[], int n) {

for (int i = n / 2 - 1; i >= 0; i--)

heapify(arr, n, i);

for (int i = n - 1; i >= 0; i--) {

int temp = arr[0];

arr[0] = arr[i];

arr[i] = temp;

heapify(arr, i, 0);

}

}

int main() {

int arr[10], n, i;

printf("Enter number of elements \n");

scanf("%d", &n);

printf("Enter %d elements \n", n);

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

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

heapsort(arr, n);

printf("\nSorted array: ");

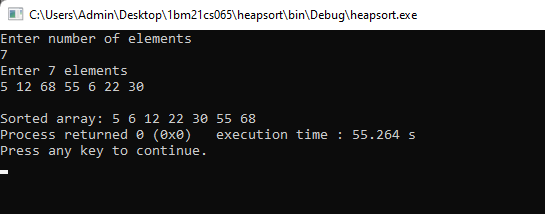
for (i = 0; i < n; i++)

printf("%d ", arr[i]);

return 0;

}

OUTPUT:

****