**Amity School of Engineering and Technology**



**ANALYSIS AND DESIGN OF ALGORITHM LAB**

Name: **Jayant Bhalla**

Branch: **B.Tech- 6CSE10-X**

Enrollment No.: **A12405215021**

Batch: **2015-19**

Faculty**: Dr. Ranjeet Kr. Rout**

Signature**: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**INDEX**

|  |  |  |  |
| --- | --- | --- | --- |
| S.No. | Experiment | Date | Sign |
| **1.** | Program to implement Selection Sort |  |  |
| **2.** | Program to implement Insertion Sort |  |  |
| **3.** | Program to implement Quick Sort |  |  |
| **4.** | Program to implement Merge Sort |  |  |
| **5.** | Program to implement Binary Search |  |  |
| **6.** | Program to implement Heap Sort |  |  |
| **7.** | Program to implement Matrix Chain Multiplication |  |  |
| **8.** | Program to implement Longest Common Subsequence |  |  |
| **9.** | Program to implement 0-1 Knapsack |  |  |
| **10.** | Program to implement Fractional Knapsack |  |  |
| **11.** | Program to implement DFS, BFS, Minimum Spanning tree using Kruskal Algorithm Prims Algorithm |  |  |
| **12.** | Program to implement DFS, BFS, Minimum Spanning tree using Prims Algorithm |  |  |
| **13.** | Program to implement Dijkstra Algorithm |  |  |
| **14.** | Program to implement Floyd-Warshall Algorithm |  |  |
| **15.** | Program to implement Bellman Ford Algorithm |  |  |
| **16.** | Program to solve Travelling Salesman Problem and N-Queen Problem |  |  |

**EXPERIMENT – 1**

**OBJECTIVE:** Sort a given set of elements using selection sort method and determine the time required to sort the elements.

**SOFTWARE USED:** Turbo C++.

**Input:**

#include <stdio.h>

int \*selectionSort(int array[],int n)

{int j,temp,i;

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

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

{if(array[i] > array[j])

{temp = array[i];

array[i] = array[j];

array[j] = temp; } } }

return array;}

int main()

{ int array[1000],n,i;

printf("Enter the number of element you want to Sort : ");

scanf("%d",&n);

printf("Enter Elements in the list : ");

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

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

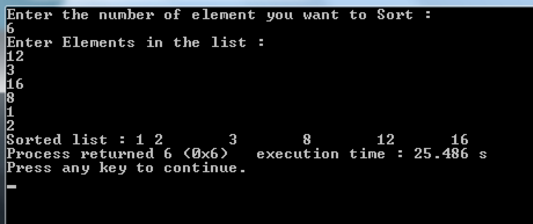
int \*sortArray = selectionSort(array,n);

printf("Sorted list : ");

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

{printf("%d\t",sortArray[i]); }}

**Output:**



**EXPERIMENT – 2**

**OBJECTIVE:** Sort a given set of elements using insertion sort method and determine the time required to sort the elements.

**SOFTWARE USED:** Turbo C++.

**Input:**

#include <stdio.h>

int main()

{

int n, array[1000], c, d, t;

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

scanf("%d", &n);

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

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

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

}

for (c = 1 ; c <= n - 1; c++) {

d = c;

while ( d > 0 && array[d-1] > array[d]) {

t = array[d];

array[d] = array[d-1];

array[d-1] = t;

d--;} }

printf("Sorted list in ascending order:\n");

for (c = 0; c <= n - 1; c++) {

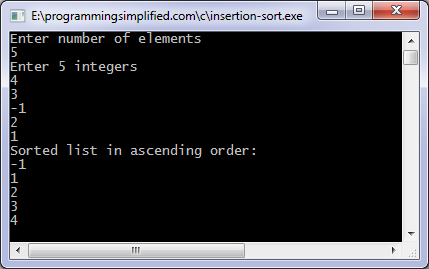
printf("%d\n", array[c]);

}

return 0;

}

**OUTPUT:**



**EXPERIMENT – 3**

**OBJECTIVE:** Sort a given set of elements using quick sort method and determine the time required to sort the elements.

**SOFTWARE USED:** Turbo C++.

**Input:**

#include<iostream>

using namespace std;

int a[10], l, u, i, j, n, count = 0;

double timecount;

clock\_t startclock, finishclock;

void quick(int\*, int, int);

int main()

{double timecount;

clock\_t startclock, finishclock;

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

cin>>n;

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

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

{cin>>a[i];}

u = n-1;

startclock = clock();

quick(a, l, u);

finishclock = clock();

cout<<"Sorted array is :";

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

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

} cout<<"\n";

timecount = double (finishclock-startclock) / 100;

cout<<"Timecount : "<<timecount<<"\n";

return 0;}

void quick(int a[], int l, int u)

{int p, temp;

if(l < u)

{p = a[l];

i = l;

j = u;

while(i < j)

{while(a[i] <= p && i < j)

{i++;}

while(a[j] > p && i <= j)

{j--;}

if(i<= j)

{temp = a[i];

a[i] = a[j];

a[j] = temp;

}}

temp =a[j];

a[j] = a[l];

a[l] = temp;

cout<<"Iteration "<<count+1<<" :";

count++;

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

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

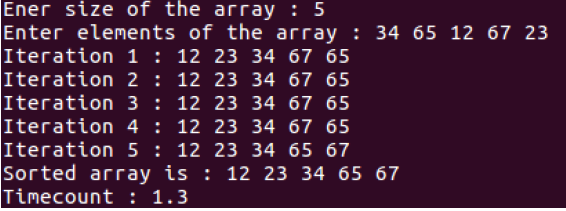
} cout<<"\n";

quick(a, l, j-1);

quick(a, j+1, u);

}}

**OUTPUT:**



**EXPERIMENT – 4**

**OBJECTIVE:** Sort a given set of elements using merge sort method and determine the time required to sort the elements.

**SOFTWARE USED:** Turbo C++.

**Input:**

#include<stdio.h>

void mergesort(int a[],int i,int j);

void merge(int a[],int i1,int j1,int i2,int j2);

int main()

{int a[30],n,i;

 printf("Enter no of elements:");

 scanf("%d",&n);

 printf("Enter array elements:");

 for(i=0;i<n;i++)

 scanf("%d",&a[i]);

 mergesort(a,0,n-1);

 printf("\nSorted array is :");

 for(i=0;i<n;i++)

 printf("%d ",a[i]);

 return 0;}

void mergesort(int a[],int i,int j)

{ int mid;

 if(i<j)

 { mid=(i+j)/2;

  mergesort(a,i,mid);

  mergesort(a,mid+1,j);

merge(a,i,mid,mid+1,j);     }}

void merge(int a[],int i1,int j1,int i2,int j2)

{int temp[50];

    int i,j,k;

    i=i1;

    j=i2;

    k=0;

 while(i<=j1 && j<=j2)    //while elements in both lists

    { if(a[i]<a[j])

            temp[k++]=a[i++];

        else

            temp[k++]=a[j++];   }

    while(i<=j1)

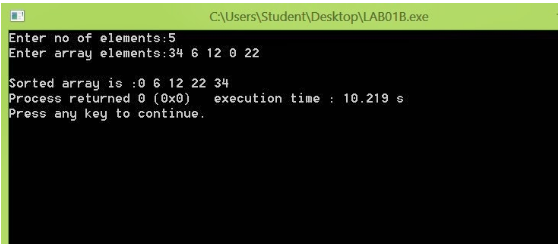
        temp[k++]=a[i++];

    while(j<=j2)

        temp[k++]=a[j++];

    for(i=i1,j=0;i<=j2;i++,j++)

        a[i]=temp[j];}

**OUTPUT:**

**EXPERIMENT – 5**

**OBJECTIVE:** Sort a given set of elements using binary search method and determine the time required to search the elements.

**SOFTWARE USED:** Turbo C++.

**Input:**

#include<iostream>

using namespace std;

int main()

{int i, j, a[50], n, temp, beg, mid, end, item, flag = 1;

double timecount;

clock\_t startclock, finishclock;

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

cin>>n;

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

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

{cin>>a[i];}

cout<<"Sorted array is :";

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

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

{if(a[i] > a[j])

{temp = a[i];

a[i] = a[j];

a[j] = temp;}}}

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

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

cout<<"\nEnter the element to be searched : ";

cin>>item;

beg = 0;

end = n - 1;

startclock = clock();

while(beg <= end)

{mid = (beg + end)/2;

if(item == a[mid])

{cout<<item<<" found at position "<<mid+1<<"\n";

flag = 0;

break;}

else if(item >a[mid])

{beg = mid + 1;}

else

{end = mid - 1;}}

finishclock = clock();

if(flag == 1)

{cout<<item<<" is not present in the array\n";}

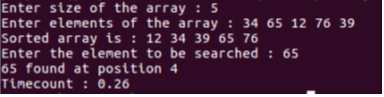
timecount = double (

finishclock-startclock) /100;

cout<<"Timecount : "<<timecount<<"\n";

return 0;}

**OUTPUT:**



**EXPERIMENT – 6**

**OBJECTIVE:** Sort a given set of elements using heap sort method and determine the time required to sort the elements.

**SOFTWARE USED:** Turbo C++.

**Input:**

#include<stdio.h>

#include<conio.h>

int main()

{int TREE[10],N,i,j,K,p,c,temp;

printf("\n\n Enter no of elements..");

scanf("%d",&N);

printf("\n\n Enter the nos..");

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

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

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

{K=i;

do

{ if(TREE[K]>TREE[K/2])

{temp=TREE[K];

TREE[K]=TREE[K/2];

TREE[K/2]=temp;}

p=K/2;

K=p;}

while(K!=0);}

printf("\n\n\n On inserting values are arranged as \n");

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

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

for(j=N;j>0;j--)

{temp=TREE[1];

TREE[1]=TREE[j];

TREE[j]=temp;

p=0;

do

{c=2\*p+2;

if((TREE[c][/c]<TREE[c language="+1"][/c]) && c<j-1)

c++;

if(TREE[p]<TREE[c][/c] && c<j)

{temp=TREE[p];

TREE[p]=TREE[c][/c];

TREE[c][/c]=temp;}

p=c;}

while(c<(j+1)); }

printf("\n\n\n The sorted nos are..");

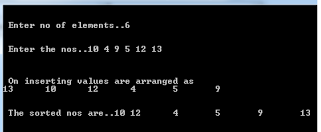
for(i=1;i<=N;i++)

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

getch();

}

**OUTPUT:**



**EXPERIMENT – 7**

**OBJECTIVE:** WAP for Matrix Chain Multiplication.

**SOFTWARE USED:** Turbo C++.

**Input:**

#include<stdio.h>

#include<stdlib.h>

int minimum\_cost(int matrix[20], int t)

{

int x, small;

if(t == 1)

{

return matrix[0];

}

else

{

small = matrix[0];

for(x = 1; x < t; x++)

{

if(matrix[x] < small)

{

small = matrix[x];

}

}

return small;

}

}

int main()

{

int t, i, l, j, k, limit;

int matrix[30], multiplier[10][15], columns[15], rows[15], temp[15];

printf("\nEnter Total Number of Matrices:\t");

scanf("%d", &limit);

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

{

printf("\nEnter Number of Columns of Matrix %d:\t", i + 1);

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

printf("Enter Number of Rows of Matrix %d:\t", i + 1);

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

}

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

{

temp[i] = columns[i];

}

temp[i] = rows[i - 1];

printf("\n");

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

{

for(j = k, i = 1; j <= limit; j++, i++)

{

multiplier[i][j] = 0;

}

}

for(l = 2; l <= limit; l++)

{

for(j = l, i = 1; j <= limit; j++, i++)

{

t = 0;

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

{

matrix[t++] = (multiplier[i][k] + multiplier[k + 1][j]) + (temp[i - 1] \* temp[k] \* temp[j]);

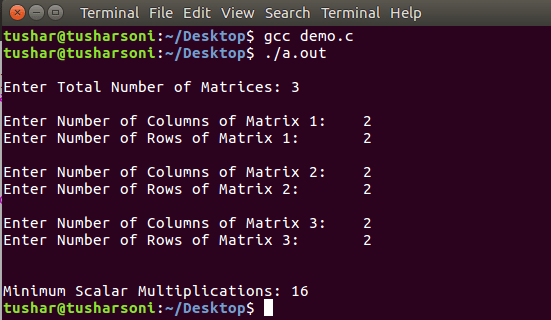
}

multiplier[i][j] = minimum\_cost(matrix, t);}}

printf("\nMinimum Scalar Multiplications:\t%d\n", multiplier[1][limit]);

return 0;}

**OUTPUT:**



**EXPERIMENT – 8**

**OBJECTIVE:** WAP for Longest Common Subsequence.

**SOFTWARE USED:** Turbo C++.

**Input:**

#include<stdio.h>

#include<conio.h>

#include<string.h>

int i,j,m,n,a,c[20][20];

char x[15],y[15],b[20][20];

void print\_lcs(int i,int j)

{

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

return;

if(b[i][j]=='c')

{

print\_lcs(i-1,j-1);

printf(" %c",x[i-1]);

}

else if(b[i][j]=='u')

print\_lcs(i-1,j);

else

print\_lcs(i,j-1);

}

void lcs\_length(void)

{

m=strlen(x);

n=strlen(y);

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

c[i][0]=0;

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

c[0][i]=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]='c';

}

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

{

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

b[i][j]='u';

}

else

{

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

b[i][j]='l';

}}

print\_lcs(m,n);

}

void main()

{printf("Enter 1st sequence : ");

gets(x);

printf("Enter 2nd sequence : ");

gets(y);

printf("\nlongest common subsequence is : ");

lcs\_length();

getch();

}

**OUTPUT:**



**EXPERIMENT – 9**

**OBJECTIVE:** WAP for 0-1 Knapsack Problem.

**SOFTWARE USED:** Turbo C++.

**Input:**

#include<iostream>

using namespace std;

int main()

{int w[10], v[10], W, n, i, c[10][30], u, k;

cout<<"Enter capacity of Knapsack : ";

cin>>W;

cout<<"Enter number of items : ";

cin>>n;

cout<<"Enter the weight of items : ";

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

{cin>>w[i];

} cout<<"Enter the value of items :";

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

{cin>>v[i];

} cout<<"Weight array is :";

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

{cout<<w[i]<<" ";}

cout<<"\nValue array is : ";

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

{cout<<v[i]<<" ";}

for(u = 0; u <= W; u++)

{c[0][u] = 0;}

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

{c[i][0] = 0;}

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

{for(u = 1; u <= W; u++)

{if(w[i] <= u)

{if(v[i] + c[i-1][u-w[i]] > c[i-1][u])

{c[i][u] = v[i] + c[i-1][u-w[i]];

} else

{c[i][u] = c[i-1][u];}}

else

{c[i][u] = c[i-1][u];

}}}

cout<<"\nWeight-value (cost) matrix : \n";

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

{for(u = 0; u <= W; u++)

{cout<<c[i][u]<<" ";}

cout<<"\n";}

cout<<"Maximum value of items included in Knapsack ="<<c[n][W]<<"\n";i = n;k = W;

cout<<"Items in Knapsack are : ";

while(i > 0 && k > 0)

{if(c[i][k] != c[i-1][k])

{

cout<<i<<" ";

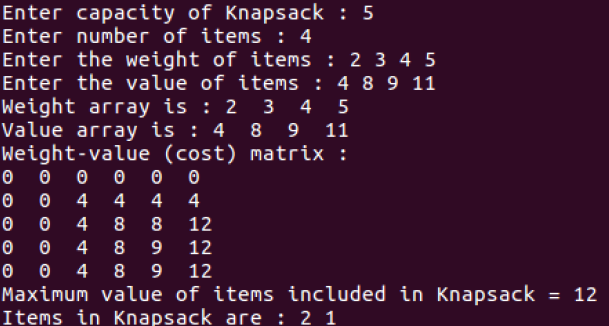
k = k - w[i];}

i= i - 1;

} cout<<"\n";

return 0;}

**OUTPUT:**



**EXPERIMENT – 10**

**OBJECTIVE:** WAP for Fractional Knapsack Problem.

**SOFTWARE USED:** Turbo C++.

**Input:**

# include<stdio.h>

void knapsack(int n, float weight[], float profit[], float capacity) {

float x[20], tp = 0;

int i, j, u;

u = capacity;

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

x[i] = 0.0;

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

if (weight[i] > u)

break;

else {

x[i] = 1.0;

tp = tp + profit[i];

u = u - weight[i];

} }

if (i < n)

x[i] = u / weight[i];

tp = tp + (x[i] \* profit[i]);

printf("\nThe result vector is:- ");

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

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

printf("\nMaximum profit is:- %f", tp);}

int main() {

float weight[20], profit[20], capacity;

int num, i, j;

float ratio[20], temp;

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

scanf("%d", &num);

printf("\nEnter the wts and profits of each object:- ");

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

scanf("%f %f", &weight[i], &profit[i]);

}

printf("\nEnter the capacityacity of knapsack:- ");

scanf("%f", &capacity);

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

ratio[i] = profit[i] / weight[i];

}

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

for (j = i + 1; j < num; j++) {

if (ratio[i] < ratio[j]) {

temp = ratio[j];

ratio[j] = ratio[i];

ratio[i] = temp;

temp = weight[j];

weight[j] = weight[i];

weight[i] = temp;

temp = profit[j];

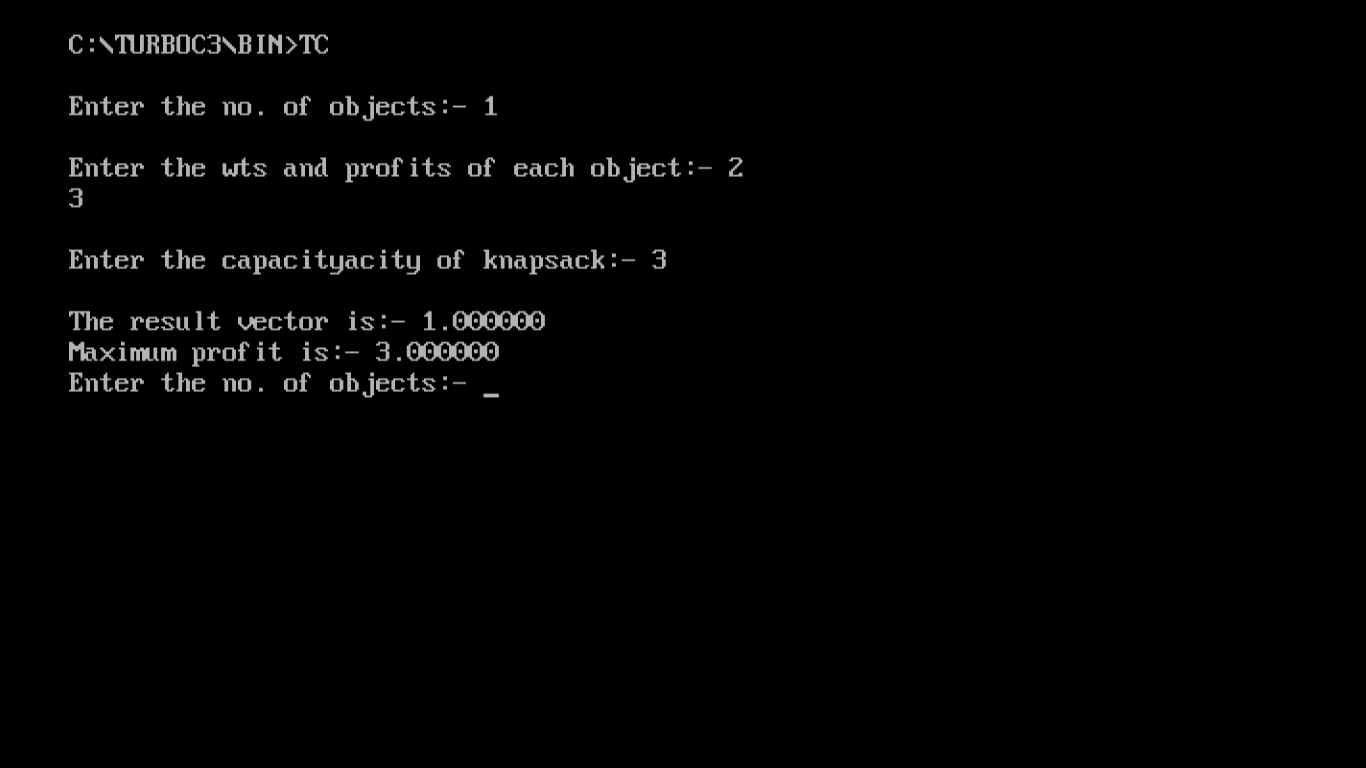
profit[j] = profit[i];

profit[i] = temp; }}}

knapsack(num, weight, profit, capacity);

return(0);

**OUTPUT:**



**EXPERIMENT – 13**

**OBJECTIVE:** WAP for Dijkstra Algorithm.

**SOFTWARE USED:** Turbo C++.

**Input:**

#include<stdio.h>

#include<conio.h>

#include<process.h>

#include<string.h>

#include<math.h>

int dijkstra(int cost[][N],int source,int target);

void main() {

int cost[N][N],i,j,w,ch,co;

int source,target,x,y;

clrscr();

printf("Shortest path algorithm DIJKSTRA'S ALGORITHM \n\n");

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

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

{cost[i][j]=IN;}}

for(x=1;x<N;x++)

{for(y=x+1;y<N;y++)

{printf("Enter the weight of the path between node %d and %d:",x,y); scanf("%d",&w);

cost[x][y]=cost[y][x]=w;}

printf("\n");}

printf("\n Enter the source:"); scanf("%d",&source); printf("\n Enter the target"); scanf("%d",&target); co=dijsktra(cost,source,target); printf("\n shortest path:%d",co); getch();}

int dijsktra(int cost[][N],int source,int target)

{int dist[N],prev[N],selected[N]={0},i,m,min,start,d,j; char path[N];

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

{dist[i]=IN; prev[i]=-1;}

start=source;

selected[start]=1;

dist[start]=0;

while(selected[target]==0)

{min=IN;m=0;

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

{d=dist[start]+cost[start][i]; if(d<dist[i] && selected[i]==0)

{dist[i]=d;

prev[i]=start;}

if(min>dist[i] && selected[i]==0)

{min=dist[i];

m=i;}}

start=m;

selected[start]=1;}

start=target;

j=0; while(start!=-1)

{path[j++]=start+65;

start=prev[start];}

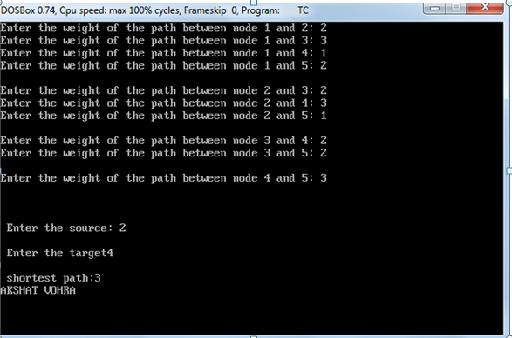
path[j]='\0';

strrev(path);

printf("%s",path);

return dist[target];}

**OUTPUT:**



**EXPERIMENT – 14**

**OBJECTIVE:** WAP for Floyd-Warshal Algorithm.

**SOFTWARE USED:** Turbo C++.

**Input:**

#include <iostream.h>

#include <conio.h>

void floyds(int b[][7])

{

int i, j, k;

for (k = 0; k < 7; k++)

{

for (i = 0; i < 7; i++)

{

for (j = 0; j < 7; j++)

{

if ((b[i][k] \* b[k][j] != 0) && (i != j))

{

if ((b[i][k] + b[k][j] < b[i][j]) || (b[i][j] == 0))

{

b[i][j] = b[i][k] + b[k][j];}}}}}

for (i = 0; i < 7; i++)

{

cout<<"\nMinimum Cost With Respect to Node:"<<i<<endl;

for (j = 0; j < 7; j++)

{

cout<<b[i][j]<<"\t";

}}}

int main()

{

int b[7][7];

cout<<"ENTER VALUES OF ADJACENCY MATRIX\n\n";

for (int i = 0; i < 7; i++)

{

cout<<"enter values for "<<(i+1)<<" row"<<endl;

for (int j = 0; j < 7; j++)

{

cin>>b[i][j];

}

}

floyds(b);

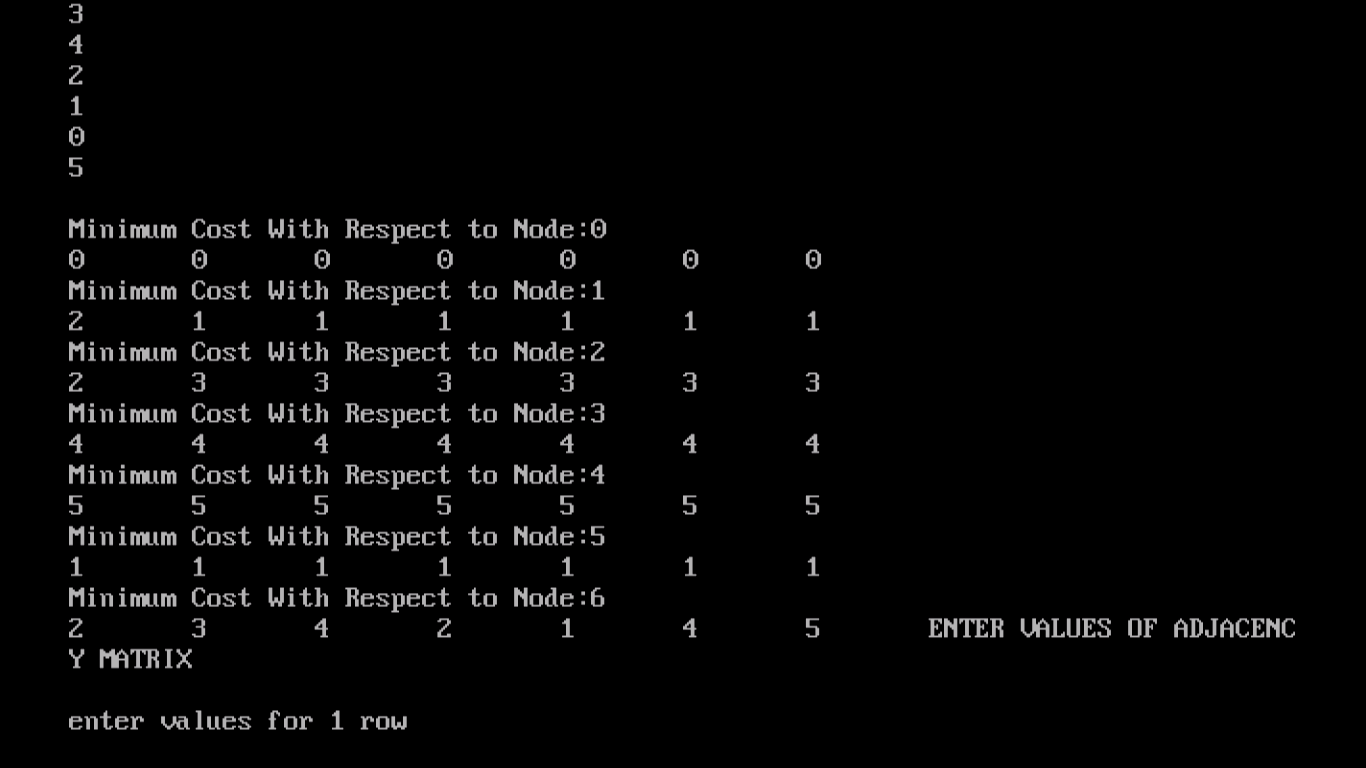
getch();

}

**OUTPUT:**







**EXPERIMENT – 15**

**OBJECTIVE:** WAP for Bellman Ford Algorithm.

**SOFTWARE USED:** Turbo C++.

**Input:**

#include <stdio.h>

#include <stdlib.h>

int Bellman\_Ford(int G[20][20] , int V, int E, int edge[20][2])

{

int i,u,v,k,distance[20],parent[20],S,flag=1;

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

distance[i] = 1000 , parent[i] = -1 ;

printf("Enter source: ");

scanf("%d",&S);

distance[S-1]=0 ;

for(i=0;i<V-1;i++)

{

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

{

u = edge[k][0] , v = edge[k][1] ;

if(distance[u]+G[u][v] < distance[v])

distance[v] = distance[u] + G[u][v] , parent[v]=u ;

}

}

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

{

u = edge[k][0] , v = edge[k][1] ;

if(distance[u]+G[u][v] < distance[v])

flag = 0 ;

}

if(flag)

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

printf("Vertex %d -> cost = %d parent = %d\n",i+1,distance[i],parent[i]+1);

return flag;}

int main()

{

int V,edge[20][2],G[20][20],i,j,k=0;

printf("BELLMAN FORD\n");

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

scanf("%d",&V);

printf("Enter graph in matrix form:\n");

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

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

{

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

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

edge[k][0]=i,edge[k++][1]=j;}

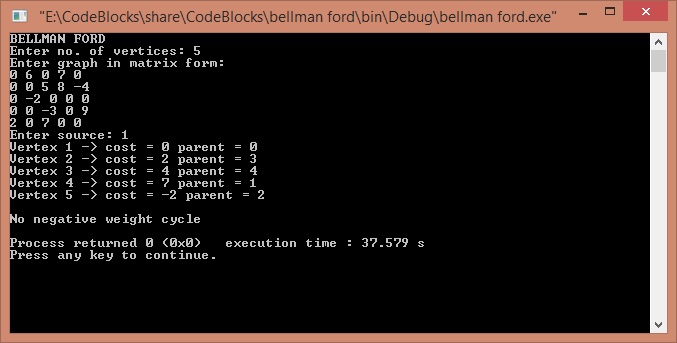
if(Bellman\_Ford(G,V,k,edge))

printf("\nNo negative weight cycle\n");

else printf("\nNegative weight cycle exists\n");

return 0;}

**OUTPUT:**



**EXPERIMENT – 16**

**OBJECTIVE:** WAP for Travelling Salesman Problem.

**SOFTWARE USED:** Turbo C++.

**Input:**

#include<stdio.h>

int matrix[25][25], visited\_cities[10], limit, cost = 0;

int tsp(int c)

{

int count, nearest\_city = 999;

int minimum = 999, temp;

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

{

if((matrix[c][count] != 0) && (visited\_cities[count] == 0))

{

if(matrix[c][count] < minimum)

{

minimum = matrix[count][0] + matrix[c][count];

}

temp = matrix[c][count];

nearest\_city = count;}}

if(minimum != 999)

{

cost = cost + temp;

}

return nearest\_city;}

void minimum\_cost(int city)

{

int nearest\_city;

visited\_cities[city] = 1;

printf("%d ", city + 1);

nearest\_city = tsp(city);

if(nearest\_city == 999)

{

nearest\_city = 0;

printf("%d", nearest\_city + 1);

cost = cost + matrix[city][nearest\_city];

return;

}

minimum\_cost(nearest\_city);}

int main()

{ int i, j;

printf("Enter Total Number of Cities:\t");

scanf("%d", &limit);

printf("\nEnter Cost Matrix\n");

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

{ printf("\nEnter %d Elements in Row[%d]\n", limit, i + 1);

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

{

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

}

visited\_cities[i] = 0;

}

printf("\nEntered Cost Matrix\n");

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

{

printf("\n");

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

{

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

}

}

printf("\n\nPath:\t");

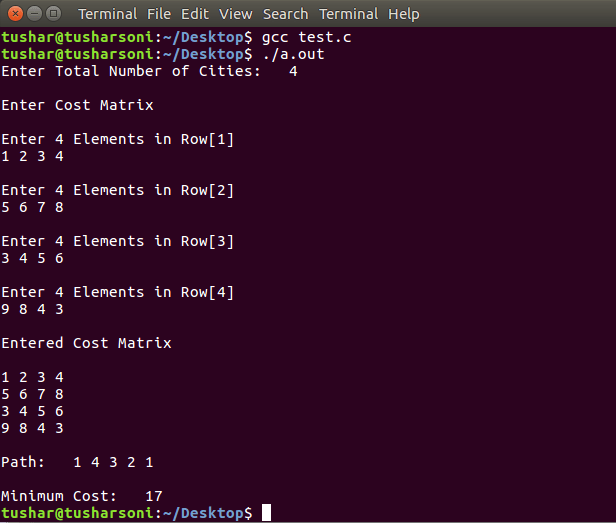
minimum\_cost(0);

printf("\n\nMinimum Cost: \t");

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

return 0;}

**OUTPUT:**



**EXPERIMENT – 11**

**OBJECTIVE:** WAP for Graph Theory- DFS, BFS and Minimum Spanning Tree using Kruskal’s Algorithm.

**SOFTWARE USED:** Turbo C++.

**Input:**

1. #include<iostream.h>

#include<conio.h>

#include<stdlib.h>

int cost[10][10],i,j,k,n,stack[10],top,v,visit[10],visited[10];

void main()

{ int m;

cout <<"enter no of vertices";

cin >> n;

cout <<"enter no of edges";

cin >> m;

cout <<"\n EDGES \n";

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

{ cin >>i>>j;

cost[i][j]=1;

}

cout <<"enter initial vertex";

cin >>v;

cout <<"ORDER OF VISITED VERTICES";

cout << v <<" ";

visited[v]=1;

k=1;

while(k<n)

{

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

if(cost[v][j]!=0 && visited[j]!=1 && visit[j]!=1)

{

visit[j]=1;

stack [top]=j;

top++;

}

v= stack [--top];

cout<<v << " ";

k++;

visit[v]=0;

visited[v]=1;}

getch();}

**2.** #include<iostream.h>

using namespace std;

int c = 0, t = 0;

struct node\_info

{

int no;

int st\_time;

}\*q = NULL, \*r = NULL, \*x = NULL;

struct node

{node\_info \*pt;

node \*next;

}\*front = NULL, \*rear = NULL, \*p = NULL, \*np = NULL;

void push(node\_info \*ptr)

{np = new node;

np->pt = ptr;

np->next = NULL;

if(front == NULL)

{front = rear = np;

rear->next = NULL;

13

} else

{rear->next = np;

rear = np;

rear->next = NULL;}}

node\_info \*remove()

{if(front == NULL)

{cout<<"Empty queue\n";}

else

{p = front;

x = p->pt;

front = front->next;

delete(p);

return(x);

}}

void bfs(int \*v,int am[][10],int i, int n)

{if(c == 0)

{q = new node\_info;

q->no = i;

q->st\_time = t++;

cout<<"At number "<<q->st\_time+1<<", node "<<q->no+1<<" is traversed\n";

v[i] = 1;

push(q);

} c++;

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

{if(am[i][j] == 0 || (am[i][j] == 1 && v[j] == 1))

continue;

else if(am[i][j] == 1 && v[j] == 0)

{r = new node\_info;

r->no = j;

r->st\_time = t++;

cout<<"At number "<<r->st\_time+1<<", node "<<r->no+1<<" is

traversed\n";

v[j] = 1;

push(r);}}

remove();

if(c <= n-1 && front != NULL)

{bfs(v, am, remove()->no, n);

}}

int main()

{int i, j, n, v[10], am[10][10];

cout<<"Enter number of nodes in graph : ";

cin>>n;

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

{v[i] = 0;}

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

{cout<<"Enter the adjacency matrix for node "<<i+1<<" : ";

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

{cin>>am[i][j];}}

bfs(v, am, 0, n);

return 0;}

**3.** #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()

{clrscr();

printf("\n\tImplementation of Kruskal's algorithm\n");

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

scanf("%d",&n);

printf("\nEnter the cost 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("\n\tMinimum 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)

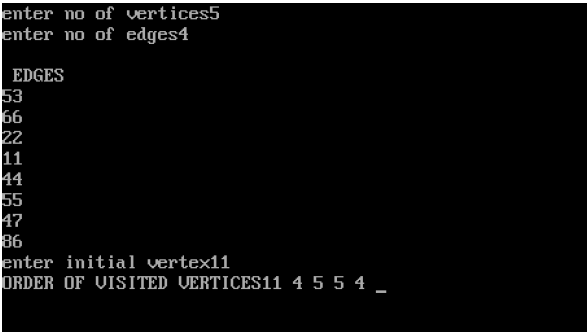
{parent[j]=i;

return 1;}

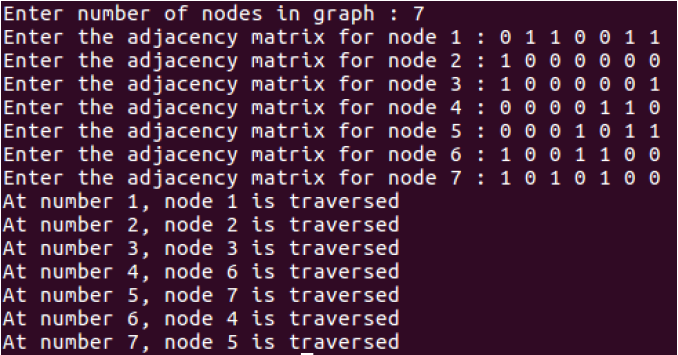
return 0;}

**OUTPUT:**

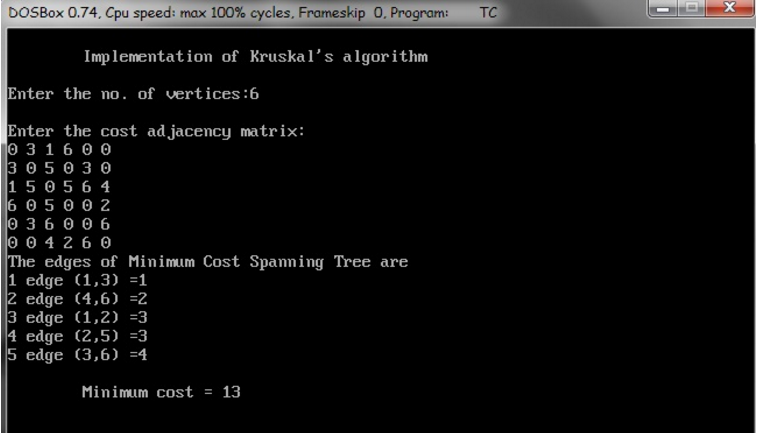
1.



**2.**



**3.**



**EXPERIMENT – 12**

**OBJECTIVE:** WAP for Minimum Spanning Tree, BFS and DFS using Prism’s Algorithm.

**SOFTWARE USED:** Turbo C++.

**Input:**

**1**. #include <iostream.h>

#include <conio.h>

#define ROW 7

#define COL 7

#define infi 5000 //infi for infinityclass prims

{

int graph[ROW][COL],nodes;

public:

prims();

void createGraph();

void primsAlgo();

};

prims :: prims(){

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

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

graph[i][j]=0;

}

void prims :: createGraph(){

int i,j;

cout<<"Enter Total Nodes : ";

cin>>nodes;

cout<<"\n\nEnter Adjacency Matrix : \n";

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

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

cin>>graph[i][j];

//Assign infinity to all graph[i][j] where weight is 0.for(i=0;i<nodes;i++){

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

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

graph[i][j]=infi;

}

}

}

void prims :: primsAlgo(){

int selected[ROW],i,j,ne; //ne for no. of edgesintfalse=0,true=1,min,x,y;

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

selected[i]=false;

selected[0]=true;

ne=0;

while(ne < nodes-1){

min=infi;

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

{

if(selected[i]==true){

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

if(selected[j]==false){

if(min > graph[i][j])

{

min=graph[i][j];

x=i;

y=j;

}

}

}

}

}

selected[y]=true;

cout<<"\n"<<x+1<<" --> "<<y+1;

ne=ne+1;

}

}

void main(){

prims MST;

clrscr();

cout<<"\nPrims Algorithm to find Minimum Spanning Tree\n";

MST.createGraph();

MST.primsAlgo();

getch();

}

2. #include<iostream.h>

using namespace std;

int c = 0, t = 0;

struct node\_info

{

int no;

int st\_time;

}\*q = NULL, \*r = NULL, \*x = NULL;

struct node

{node\_info \*pt;

node \*next;

}\*front = NULL, \*rear = NULL, \*p = NULL, \*np = NULL;

void push(node\_info \*ptr)

{np = new node;

np->pt = ptr;

np->next = NULL;

if(front == NULL)

{front = rear = np;

rear->next = NULL;

13

} else

{rear->next = np;

rear = np;

rear->next = NULL;}}

node\_info \*remove()

{if(front == NULL)

{cout<<"Empty queue\n";}

else

{p = front;

x = p->pt;

front = front->next;

delete(p);

return(x);

}}

void bfs(int \*v,int am[][10],int i, int n)

{if(c == 0)

{q = new node\_info;

q->no = i;

q->st\_time = t++;

cout<<"At number "<<q->st\_time+1<<", node "<<q->no+1<<" is traversed\n";

v[i] = 1;

push(q);

} c++;

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

{if(am[i][j] == 0 || (am[i][j] == 1 && v[j] == 1))

continue;

else if(am[i][j] == 1 && v[j] == 0)

{r = new node\_info;

r->no = j;

r->st\_time = t++;

cout<<"At number "<<r->st\_time+1<<", node "<<r->no+1<<" is

traversed\n";

v[j] = 1;

push(r);}}

remove();

if(c <= n-1 && front != NULL)

{bfs(v, am, remove()->no, n);

}}

int main()

{int i, j, n, v[10], am[10][10];

cout<<"Enter number of nodes in graph : ";

cin>>n;

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

{v[i] = 0;}

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

{cout<<"Enter the adjacency matrix for node "<<i+1<<" : ";

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

{cin>>am[i][j];}}

bfs(v, am, 0, n);

return 0;}

1. 3. #include<iostream.h>

#include<conio.h>

#include<stdlib.h>

int cost[10][10],i,j,k,n,stack[10],top,v,visit[10],visited[10];

void main()

{ int m;

cout <<"enter no of vertices";

cin >> n;

cout <<"enter no of edges";

cin >> m;

cout <<"\n EDGES \n";

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

{ cin >>i>>j;

cost[i][j]=1;

}

cout <<"enter initial vertex";

cin >>v;

cout <<"ORDER OF VISITED VERTICES";

cout << v <<" ";

visited[v]=1;

k=1;

while(k<n)

{

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

if(cost[v][j]!=0 && visited[j]!=1 && visit[j]!=1)

{

visit[j]=1;

stack [top]=j;

top++;

}

v= stack [--top];

cout<<v << " ";

k++;

visit[v]=0;

visited[v]=1;}

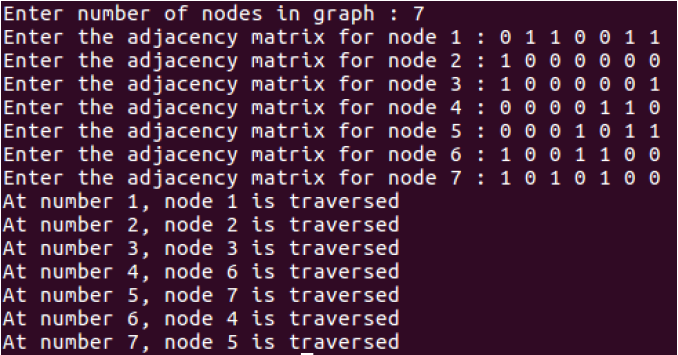
getch();}

**OUTPUT:**

**1.**



**2.**



**3.**

