**Program’s**

1. Design and implement C/C++ program to find minimum cost spanning tree of a given connected undirected graph using Kruskal’s algorithm.

#define INF 999

#define MAX 100

int p[MAX], c[MAX][MAX], t[MAX][2];

int find(int v)

{

while (p[v])

v = p[v];

return v;

}

void union1(int i, int j)

{

p[j] = i;

}

void kruskal(int n)

{

int i, j, k, u, v, min, res1, res2, sum = 0;

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

{

min = INF;

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

{

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

{

if (i == j) continue;

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

{

u = find(i);

v = find(j);

if (u != v)

{

res1 = i;

res2 = j;

min = c[i][j];

}

}

}

}

union1(res1, find(res2));

t[k][1] = res1;

t[k][2] = res2;

sum = sum + min;

}

printf("\nCost of spanning tree is=%d", sum);

printf("\nEdgesof spanning tree are:\n");

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

printf("%d -> %d\n", t[i][1], t[i][2]);

}

int main()

{

int i, j, n;

printf("\nEnter the n value:");

scanf("%d", & n);

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

p[i] = 0;

printf("\nEnter the graph data:\n");

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

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

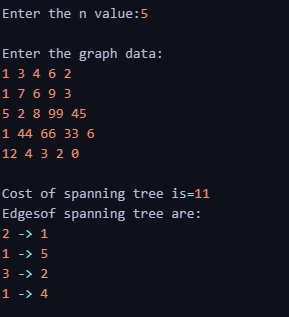
scanf("%d", & c[i][j]);

kruskal(n);

return 0;

}

**Output:-**



1. Design and implement C/C++ program to find minimum cost spanning tree of a given connected undirected graph using Prim’s algorithm.

#include<stdio.h>

#define INF 999

int prim(int c[10][10],int n,int s)

{

int v[10],i,j,sum=0,ver[10],d[10],min,u;

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

{

ver[i]=s;

d[i]=c[s][i];

v[i]=0;

}

v[s]=1;

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

{

min=INF;

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

if(v[j]==0 && d[j]<min)

{

min=d[j];

u=j;

}

v[u]=1;

sum=sum+d[u];

printf("\n%d -> %d sum=%d",ver[u],u,sum);

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

if(v[j]==0 && c[u][j]<d[j])

{

d[j]=c[u][j];

ver[j]=u;

}

}

return sum;

}

void main()

{

int c[10][10],i,j,res,s,n;

printf("\nEnter n value:");

scanf("%d",&n);

printf("\nEnter the graph data:\n");

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

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

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

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

scanf("%d",&s);

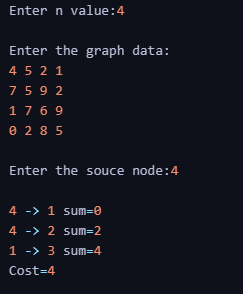
res=prim(c,n,s);

printf("\nCost=%d",res);

getch();

}

**Output:-**



1. A) Design and implement C/C++ program C Program to solve All-Pairs Shortest Paths problem using Floyd's algorithm.

#include<stdio.h>

#include<conio.h>

#define INF 999

int min(int a,int b)

{

return(a<b)?a:b;

}

void floyd(int p[][10],int n)

{

int i,j,k;

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

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

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

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

}

void main()

{

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

printf("\nEnter the n value:");

scanf("%d",&n);

printf("\nEnter the graph data:\n");

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

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

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

floyd(a,n);

printf("\nShortest path matrix\n");

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

{

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

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

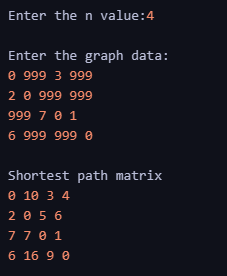
printf("\n");

}

getch();

}

**Output:-**



1. B) Design and implement C Program to find the transitive closure using

Warshal's algorithm.

#include<stdio.h>

void warsh(int p[][10],int n)

{

int i,j,k;

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

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

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

p[i][j]=p[i][j] || p[i][k] && p[k][j];

}

int main()

{

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

printf("\nEnter the n value:");

scanf("%d",&n);

printf("\nEnter the graph data:\n");

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

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

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

warsh(a,n);

printf("\nResultant path matrix\n");

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

{

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

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

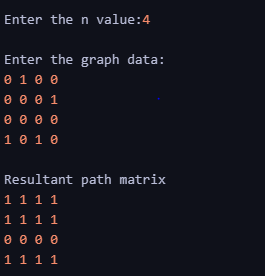
printf("\n");

}

return 0;

}

**Output:-**



1. Design and implement C Program to find shortest paths from a given vertex in a weighted connected graph to other vertices using Dijkstra's algorithm.

#include<stdio.h>

#define INF 999

void dijkstra(int c[10][10],int n,int s,int d[10])

{

int v[10],min,u,i,j;

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

{

d[i]=c[s][i];

v[i]=0;

}

v[s]=1;

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

{

min=INF;

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

if(v[j]==0 && d[j]<min)

{

min=d[j];

u=j;

}

v[u]=1;

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

if(v[j]==0 && (d[u]+c[u][j])<d[j])

d[j]=d[u]+c[u][j];

}

}

int main()

{

int c[10][10],d[10],i,j,s,sum,n;

printf("\nEnter n value:");

scanf("%d",&n);

printf("\nEnter the graph data:\n");

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

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

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

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

scanf("%d",&s);

dijkstra(c,n,s,d);

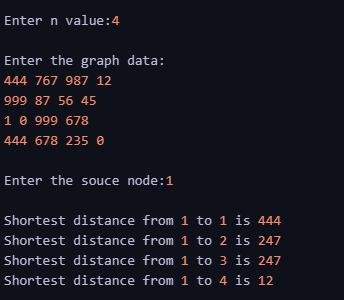
for(i=1; i<=n; i++)

printf("\nShortest distance from %d to %d is %d",s,i,d[i]);

return 0;

}

**Output:-**



1. Design and implement C Program to obtain the Topological ordering of vertices in a given digraph.

#include<stdio.h>

#include<conio.h>

int temp[10],k=0;

void sort(int a[][10],int id[],int n)

{

int i,j;

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

{

if(id[i]==0)

{

id[i]=-1;

temp[++k]=i;

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

{

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

id[j]--;

}

i=0;

}

}

}

void main()

{

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

printf("\nEnter the n value:");

scanf("%d",&n);

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

id[i]=0;

printf("\nEnter the graph data:\n");

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

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

{

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

if(a[i][j]==1)

id[j]++;

}

sort(a,id,n);

if(k!=n)

printf("\nTopological ordering not possible");

else

{

printf("\nTopological ordering is:");

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

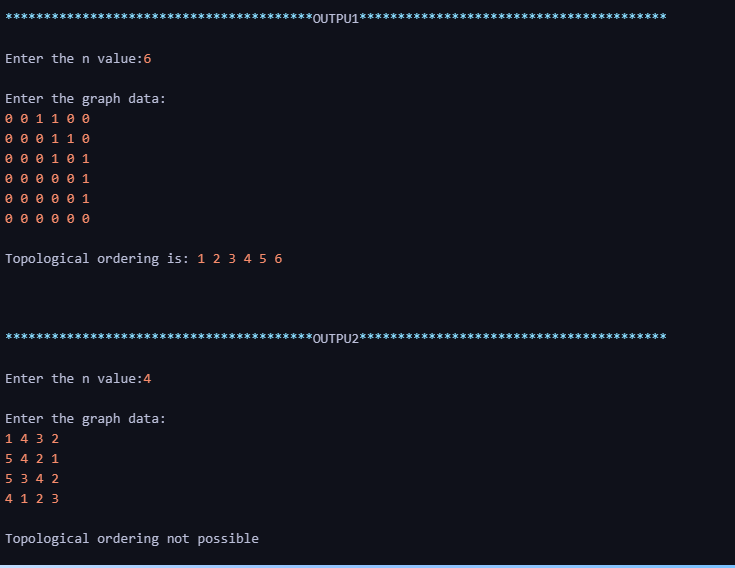
printf("%d ",temp[i]);

}

getch();

}

**Output:-**



1. Design and implement C/C++ Program to solve 0/1 Knapsack problem using Dynamic Programming method.

#include<stdio.h>

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

int max(int a,int b)

{

return a>b?a:b;

}

int knap(int i,int m)

{

if(i==n) return w[i]>m?0:p[i];

if(w[i]>m) return knap(i+1,m);

return max(knap(i+1,m),knap(i+1,m-w[i])+p[i]);

}

int main()

{

int m,i,max\_profit;

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

scanf("%d",&n);

printf("\nEnter the knapsack capacity:");

scanf("%d",&m);

printf("\nEnter profit followed by weight:\n");

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

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

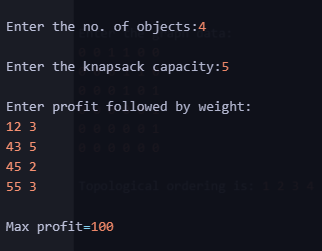
max\_profit=knap(1,m);

printf("\nMax profit=%d",max\_profit);

return 0;

}

**Output:-**



1. Design and implement C/C++ Program to solve discrete Knapsack and continuous Knapsack problems using greedy approximation method.

#include <stdio.h>

#define MAX 50

int p[MAX], w[MAX], x[MAX];

double maxprofit;

int n, m, i;

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

{

double ratio[MAX];

// Calculate the ratio of profit to weight for each item

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

{

ratio[i] = (double)p[i] / w[i];

}

// Sort items based on the ratio in non-increasing order

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

{

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

{

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

{

double temp = ratio[i];

ratio[i] = ratio[j];

ratio[j] = temp;

int temp2 = w[i];

w[i] = w[j];

w[j] = temp2;

temp2 = p[i];

p[i] = p[j];

p[j] = temp2;

}

}

}

int currentWeight = 0;

maxprofit = 0.0;

// Fill the knapsack with items

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

{

if (currentWeight + w[i] <= m)

{

x[i] = 1; // Item i is selected

currentWeight += w[i];

maxprofit += p[i];

}

else

{

// Fractional part of item i is selected

x[i] = (m - currentWeight) / (double)w[i];

maxprofit += x[i] \* p[i];

break;

}

}

printf("Optimal solution for greedy method: %.1f\n", maxprofit);

printf("Solution vector for greedy method: ");

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

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

}

int main()

{

printf("Enter the number of objects: ");

scanf("%d", &n);

printf("Enter the objects' weights: ");

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

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

printf("Enter the objects' profits: ");

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

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

printf("Enter the maximum capacity: ");

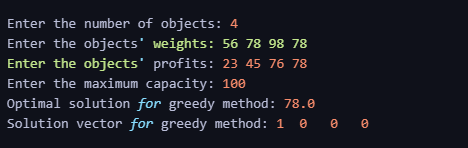
scanf("%d", &m);

greedyKnapsack(n, w, p, m);

return 0;

}

**Output:-**



1. Design and implement C/C++ Program to find a subset of a given set S = {sl , s2,.....,sn} of n positive integers whose sum is equal to a given positive integer d.

#include<stdio.h>

#define MAX 10

int s[MAX],x[MAX],d;

void sumofsub(int p,int k,int r)

{

int i;

x[k]=1;

if((p+s[k])==d)

{

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

if(x[i]==1)

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

printf("\n");

}

else if(p+s[k]+s[k+1]<=d)

sumofsub(p+s[k],k+1,r

-s[k]);

if((p+r

-s[k]>=d) && (p+s[k+1]<=d))

{

x[k]=0;

sumofsub(p,k+1,r

-s[k]);

}

}

int main()

{

int i,n,sum=0;

printf("\nEnter the n value:");

scanf("%d",&n);

printf("\nEnter the set in increasing order:");

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

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

printf("\nEnter the max subset value:");

scanf("%d",&d);

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

sum=sum+s[i];

if(sum<d || s[1]>d)

printf("\nNo subset possible");

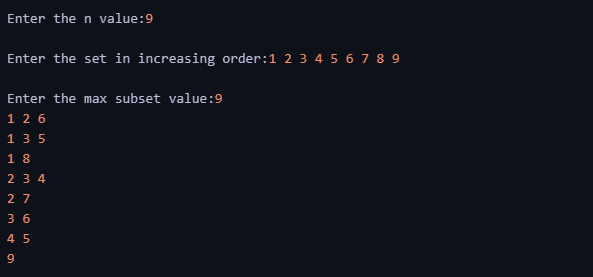
else

sumofsub(0,1,sum);

return 0;

}

**Output:-**



1. Design and implement C Program to sort a given set of n integer elements using Selection Sort method and compute its time complexity. Run the program for varied values of n> 5000 and record the time taken to sort. Plot a graph of the time taken versus n. The elements can be read from a file or can be generated using the random number generator.

#include<stdio.h>

#include<time.h>

void selsort(int a[], int n)

{

int i, j, small, pos, temp;

for(j = 0; j < n - 1; j++)

{

small = a[j];

pos = j;

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

{

if(a[i] < small)

{

small = a[i];

pos = i;

}

}

temp = a[j];

a[j] = small;

a[pos] = temp;

}

}

int main()

{

int a[10], i, n;

struct timespec start, end;

double dura;

printf("\nEnter the n value:");

scanf("%d", &n);

printf("\nEnter the array:");

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

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

clock\_gettime(CLOCK\_MONOTONIC, &start);

selsort(a, n);

clock\_gettime(CLOCK\_MONOTONIC, &end);

dura = (end.tv\_sec - start.tv\_sec) + (end.tv\_nsec - start.tv\_nsec) / 1e9;

printf("\nTime taken is: %lf seconds", dura);

printf("\nSorted array is:");

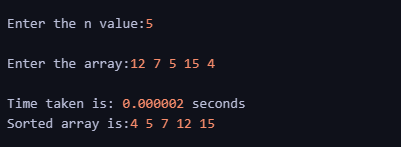
for(i = 0; i < n; i++)

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

return 0;

}

**Output:-**



1. Design and implement C Program to sort a given set of n integer elements using Quick Sort method and compute its time complexity. Run the program for varied values of n> 5000 and record the time taken to sort. Plot a graph of the time taken versus n. The elements can be read from a file or can be generated using the random number generator.