

**Name: - Sahitya Roy   
Enrolment no.: - 12020002026003  
Sec- B2  
Sub- DAA Lab**

**Instructor : Prof. Dipta Mukherjee**

**Q-1Write a Program to search an element in an array with Binary Search**.  
Code:

#include<stdio.h>

int main()

{

int c, first, last, middle, n, search, array[100];

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]);

printf("Enter the value to find:\n");

scanf("%d", &search);

first = 0;

last = n - 1;

middle = (first+last)/2;

while (first <= last) {

if (array[middle] < search)

first = middle + 1;

else if (array[middle] == search) {

printf("%d is present at index %d.\n", search, middle+1);

break;

}

else

last = middle - 1;

middle = (first + last)/2;

}

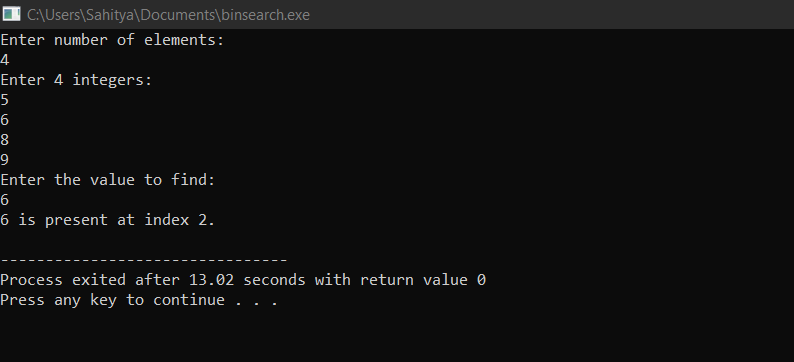
if (first > last)

printf("Not found! %d is not present in the list.\n", search);

return 0;

}

**Output:**



**Q-2 Merge Sort In C program**

**Code:**

#include<stdio.h>  
#include<time.h>

#include <stdlib.h>

void mergesort(int x[],int lb,int ub);

void merge(int x[],int lb1,int ub1,int ub2);

int count=0;

int main()

{

time\_t start,end;

int x[50];

int n,i;

start=time(NULL);

printf("Enter the no of elements:\n");

scanf("%d",&n);

printf("The unsorted list is:");

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

{

x[i]=rand();

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

}

mergesort(x,0,n-1);

printf("\n\nThe sorted list is\n");

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

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

end=time(NULL);

printf("The time diff is:%0.10f\n",difftime(end,start));

printf("The no of int. is %d",count);

}

void mergesort(int x[],int lb,int ub)

{

int mid;

if(lb < ub)

{

count++;

mid=(lb+ub)/2;

mergesort(x,lb,mid);

mergesort(x,mid+1,ub);

merge(x,lb,mid,ub);

}

}

void merge(int x[],int lb1,int ub1,int ub2)

{

int temp[50],i,j,k;

i=lb1;

j=ub1+1;

k=0;

while(i <= ub1&&j <= ub2)

{

count++;

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

{

count++;

temp[k++]=x[i++];

}

else

{

count++;

temp[k++]=x[j++];

}

}

while(i <= ub1)

{

count++;

temp[k++]=x[i++];

}

while(j <= ub2)

{

count++;

temp[k++]=x[j++];

}

for(i=lb1,j=0;i <= ub2;i++,j++)

{

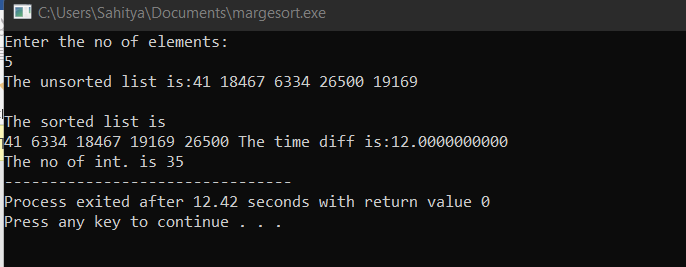
count++;

x[i]=temp[j];

}

}

**Output:**



**3. Quick Sort**

**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 some elements (Max. - 25): ");

scanf("%d",&count);

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

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

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

quicksort(number,0,count-1);

printf("The Sorted Order is: ");

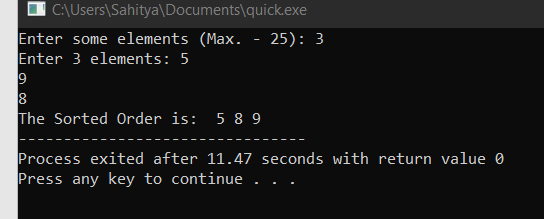
for(i=0;i<count;i++)

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

return 0;

}

**Output:**



**4. Maximum and Minimum in Divide and Conquer method**

Code:

#include <limits.h>

#include <stdio.h>

void findMinimumMaximum(int arr[], int N)

{

int i;

int minE = INT\_MAX, maxE = INT\_MIN;

// Traverse the given array

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

if (arr[i] < minE) {

minE = arr[i];

}

if (arr[i] > maxE) {

maxE = arr[i];

}

}

// Print the minimum and maximum element

printf("The minimum element is %d", minE);

printf("\n");

printf("The maximum element is %d", maxE);

return;

}

int main()

{

// Given array

int arr[] = { 1, 2, 4, -1 };

// length of the array

int N = sizeof(arr) / sizeof(arr[0]);

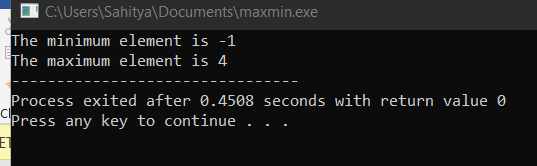
// Function call

findMinimumMaximum(arr, N);

return 0;

}

**Code:**



**5. Knapsack Problem**

Code:

#include<stdio.h>

int main ()

{

int n, m, w[100], p[100], ratio[100] , i, j, u, temp;

float xr, x[100], total\_profit=0, total\_weight=0;

printf ("Enter the number of items(n): ");

scanf ("%d", &n);

printf ("Enter the capacity of the Knapsack(m): ");

scanf ("%d", &m);

//Initializing remaining capacity of Knapsack (u)

u = m;

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

{

x[i]=0;

}

printf ("Enter the Weights of items: ");

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

{

printf ("\n\tWeight of item %d = ", i + 1);

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

}

printf ("\nEnter the Profit Values of items: ");

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

{

printf ("\n\tProfit of item %d = ", i + 1);

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

}

//Calculating Pi/Wi ratio of each item and storing in array ratio[]

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

{

ratio[i] = p[i] / w[i];

}

//Sorting all the arrays based on the ratio in descending order

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

{

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

{

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

{

temp = ratio[i];

ratio[i] = ratio[j];

ratio[j] = temp;

temp = w[i];

w[i] = w[j];

w[j] = temp;

temp = p[i];

p[i] = p[j];

p[j] = temp;

}

}

}

printf("\n The Table After Sorting based on the Ratio: \n");

//Printing Item numbers

printf("\nItem:\t\t");

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

{

printf("%d\t",i+1);

}

//Printing Profit Array

printf("\nProfit:\t\t");

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

{

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

}

//Printing Weight Array

printf("\nWeights:\t");

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

{

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

}

//Printing RATIO Array

printf ("\nRATIO:\t\t");

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

{

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

}

//Calculating Solution Array x

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

{

if(w[i]<=u)

{

x[i]=1; //Setting solution index as 1

u=u-w[i]; //updating remaining knapsack capacity

}

else if(w[i]>u)

{

break;

}

}

if(i<=n)

{

xr = (float)u/w[i];

x[i] = xr;

}

printf("\n X = [");

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

{

printf("%.3f , ",x[i]);

}

printf("]");

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

{

total\_profit += x[i]\*p[i];

total\_weight += x[i]\*w[i];

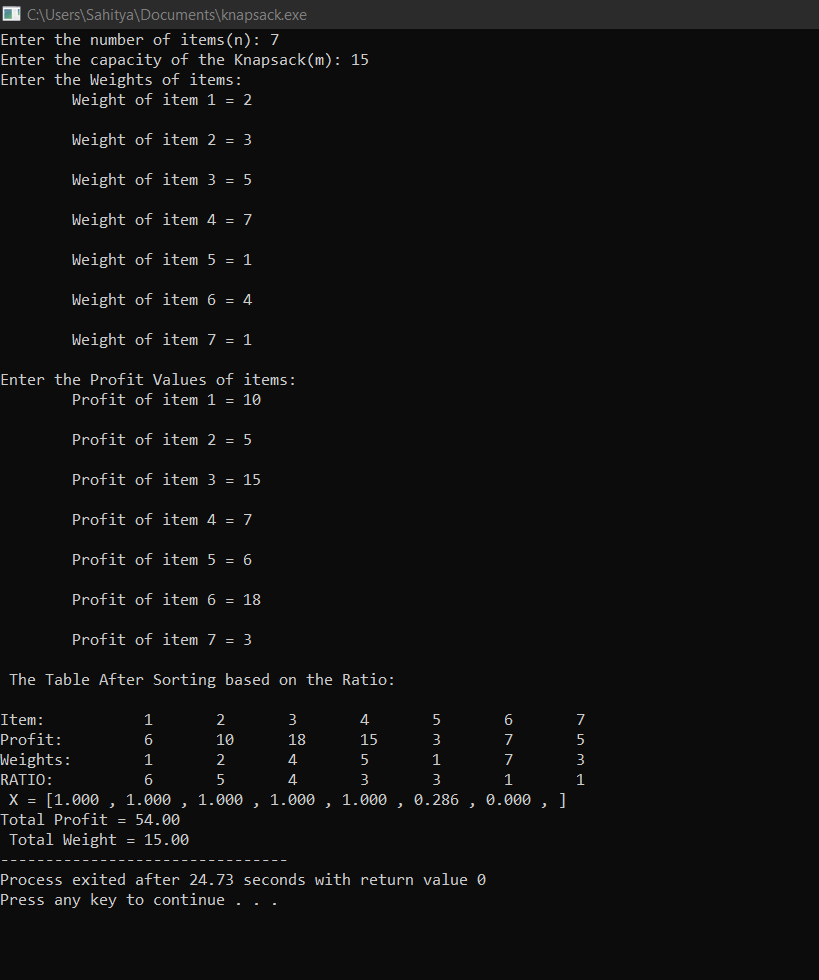
}

//Displaying Total Profit and Total Weight

printf("\nTotal Profit = %.2f \n Total Weight = %.2f ",total\_profit,total\_weight);

}

**Output:**



**5. Prims Algorithm**

Code:

// Prim's Algorithm in C

#include<stdio.h>

#include<stdbool.h>

#include <string.h>

#define INF 9999999

// number of vertices in graph

#define V 5

// create a 2d array of size 5x5

//for adjacency matrix to represent graph

int G[V][V] = {

{0, 9, 75, 0, 0},

{9, 0, 95, 19, 42},

{75, 95, 0, 51, 66},

{0, 19, 51, 0, 31},

{0, 42, 66, 31, 0}};

int main() {

int no\_edge; // number of edge

int selected[V];

// set selected false initially

memset(selected, false, sizeof(selected));

// set number of edge to 0

no\_edge = 0;

selected[0] = true;

int x;

int y;

// print for edge and weight

printf("Edge : Weight\n");

while (no\_edge < V - 1) {

int min = INF;

x = 0;

y = 0;

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

if (selected[i]) {

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

if (!selected[j] && G[i][j]) { // not in selected and there is an edge

if (min > G[i][j]) {

min = G[i][j];

x = i;

y = j;

}

}

}

}

}

printf("%d - %d : %d\n", x, y, G[x][y]);

selected[y] = true;

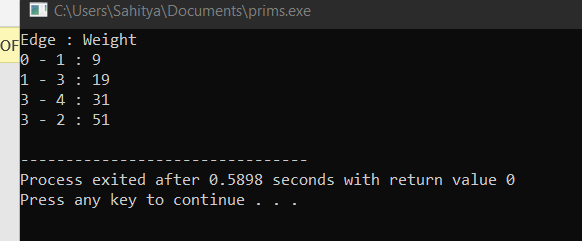
no\_edge++;

}

return 0;

}

Output:



**6. Kruskal Algorithm**

Code:

/ Kruskal's algorithm in C

#include <stdio.h>

#define MAX 30

typedef struct edge {

int u, v, w;

} edge;

typedef struct edge\_list {

edge data[MAX];

int n;

} edge\_list;

edge\_list elist;

int Graph[MAX][MAX], n;

edge\_list spanlist;

void kruskalAlgo();

int find(int belongs[], int vertexno);

void applyUnion(int belongs[], int c1, int c2);

void sort();

void print();

// Applying Krushkal Algo

void kruskalAlgo() {

int belongs[MAX], i, j, cno1, cno2;

elist.n = 0;

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

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

if (Graph[i][j] != 0) {

elist.data[elist.n].u = i;

elist.data[elist.n].v = j;

elist.data[elist.n].w = Graph[i][j];

elist.n++;

}

}

sort();

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

belongs[i] = i;

spanlist.n = 0;

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

cno1 = find(belongs, elist.data[i].u);

cno2 = find(belongs, elist.data[i].v);

if (cno1 != cno2) {

spanlist.data[spanlist.n] = elist.data[i];

spanlist.n = spanlist.n + 1;

applyUnion(belongs, cno1, cno2);

}

}

}

int find(int belongs[], int vertexno) {

return (belongs[vertexno]);

}

void applyUnion(int belongs[], int c1, int c2) {

int i;

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

if (belongs[i] == c2)

belongs[i] = c1;

}

// Sorting algo

void sort() {

int i, j;

edge temp;

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

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

if (elist.data[j].w > elist.data[j + 1].w) {

temp = elist.data[j];

elist.data[j] = elist.data[j + 1];

elist.data[j + 1] = temp;

}

}

// Printing the result

void print() {

int i, cost = 0;

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

printf("\n%d - %d : %d", spanlist.data[i].u, spanlist.data[i].v, spanlist.data[i].w);

cost = cost + spanlist.data[i].w;

}

printf("\nSpanning tree cost: %d", cost);

}

int main() {

int i, j, total\_cost;

n = 6;

Graph[0][0] = 0;

Graph[0][1] = 4;

Graph[0][2] = 4;

Graph[0][3] = 0;

Graph[0][4] = 0;

Graph[0][5] = 0;

Graph[0][6] = 0;

Graph[1][0] = 4;

Graph[1][1] = 0;

Graph[1][2] = 2;

Graph[1][3] = 0;

Graph[1][4] = 0;

Graph[1][5] = 0;

Graph[1][6] = 0;

Graph[2][0] = 4;

Graph[2][1] = 2;

Graph[2][2] = 0;

Graph[2][3] = 3;

Graph[2][4] = 4;

Graph[2][5] = 0;

Graph[2][6] = 0;

Graph[3][0] = 0;

Graph[3][1] = 0;

Graph[3][2] = 3;

Graph[3][3] = 0;

Graph[3][4] = 3;

Graph[3][5] = 0;

Graph[3][6] = 0;

Graph[4][0] = 0;

Graph[4][1] = 0;

Graph[4][2] = 4;

Graph[4][3] = 3;

Graph[4][4] = 0;

Graph[4][5] = 0;

Graph[4][6] = 0;

Graph[5][0] = 0;

Graph[5][1] = 0;

Graph[5][2] = 2;

Graph[5][3] = 0;

Graph[5][4] = 3;

Graph[5][5] = 0;

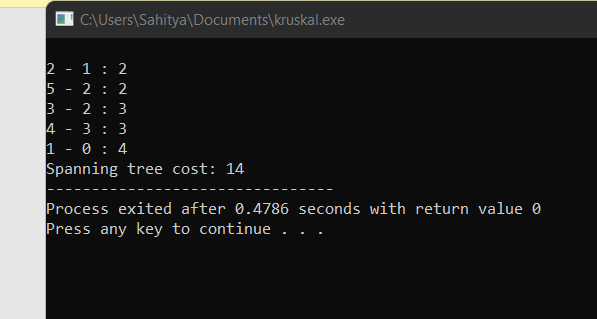
Graph[5][6] = 0;

kruskalAlgo();

print();

}

Output:



**7. Dijkstra Algorithm**   
// Dijkstra's Algorithm in C

#include <stdio.h>

#define INFINITY 9999

#define MAX 10

void Dijkstra(int Graph[MAX][MAX], int n, int start);

void Dijkstra(int Graph[MAX][MAX], int n, int start) {

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

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

// Creating cost matrix

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

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

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

cost[i][j] = INFINITY;

else

cost[i][j] = Graph[i][j];

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

distance[i] = cost[start][i];

pred[i] = start;

visited[i] = 0;

}

distance[start] = 0;

visited[start] = 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++;

}

// Printing the distance

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

if (i != start) {

printf("\nDistance from source to %d: %d", i, distance[i]);

}

}

int main() {

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

n = 7;

Graph[0][0] = 0;

Graph[0][1] = 0;

Graph[0][2] = 1;

Graph[0][3] = 2;

Graph[0][4] = 0;

Graph[0][5] = 0;

Graph[0][6] = 0;

Graph[1][0] = 0;

Graph[1][1] = 0;

Graph[1][2] = 2;

Graph[1][3] = 0;

Graph[1][4] = 0;

Graph[1][5] = 3;

Graph[1][6] = 0;

Graph[2][0] = 1;

Graph[2][1] = 2;

Graph[2][2] = 0;

Graph[2][3] = 1;

Graph[2][4] = 3;

Graph[2][5] = 0;

Graph[2][6] = 0;

Graph[3][0] = 2;

Graph[3][1] = 0;

Graph[3][2] = 1;

Graph[3][3] = 0;

Graph[3][4] = 0;

Graph[3][5] = 0;

Graph[3][6] = 1;

Graph[4][0] = 0;

Graph[4][1] = 0;

Graph[4][2] = 3;

Graph[4][3] = 0;

Graph[4][4] = 0;

Graph[4][5] = 2;

Graph[4][6] = 0;

Graph[5][0] = 0;

Graph[5][1] = 3;

Graph[5][2] = 0;

Graph[5][3] = 0;

Graph[5][4] = 2;

Graph[5][5] = 0;

Graph[5][6] = 1;

Graph[6][0] = 0;

Graph[6][1] = 0;

Graph[6][2] = 0;

Graph[6][3] = 1;

Graph[6][4] = 0;

Graph[6][5] = 1;

Graph[6][6] = 0;

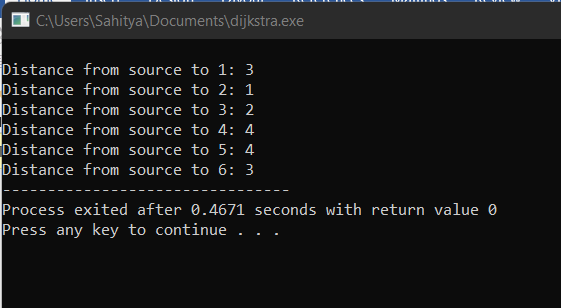
u = 0;

Dijkstra(Graph, n, u);

return 0;

}

Output:



**8. Job Sequencing**

Code:

#include <stdbool.h>

#include <stdio.h>

#include <stdlib.h>

typedef struct Job {

char id;

int dead;

int profit;

} Job;

int compare(const void\* a, const void\* b)

{

Job\* temp1 = (Job\*)a;

Job\* temp2 = (Job\*)b;

return (temp2->profit - temp1->profit);

}

// Find minimum between two numbers.

int min(int num1, int num2)

{

return (num1 > num2) ? num2 : num1;

}

// Returns minimum number of platforms required

void printJobScheduling(Job arr[], int n)

{

qsort(arr, n, sizeof(Job), compare);

// sort(arr, arr+n, comparison);

int result[n];

bool slot[n];

// Initialize all slots to be free

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

slot[i] = false;

// Iterate through all given jobs

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

for (int j = min(n, arr[i].dead) - 1; j >= 0; j--) {

// Free slot found

if (slot[j] == false) {

result[j] = i; // Add this job to result

slot[j] = true; // Make this slot occupied

break;

}

}

}

// Print the result

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

if (slot[i])

printf("%c ", arr[result[i]].id);

}

// Driver code

int main()

{

Job arr[] = { { 'a', 2, 20 },

{ 'b', 2, 15 },

{ 'c', 1, 10 },

{ 'd', 3, 5 },

{ 'e', 3, 1 } };

int n = sizeof(arr) / sizeof(arr[0]);

printf(

"Following is maximum profit sequence of jobs \n");

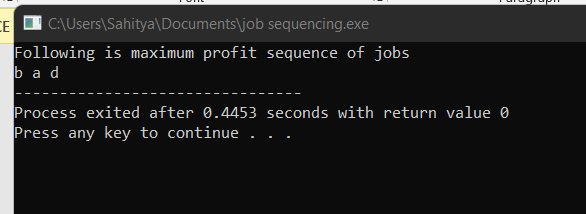
// Function call

printJobScheduling(arr, n);

return 0;

}

Output:



**9. Floyed Warshell Algorithm**

Code:

#include <stdio.h>

#define nV 4

#define INF 999

void printMatrix(int matrix[][nV]);

void floydWarshall(int graph[][nV]) {

int matrix[nV][nV], i, j, k;

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

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

matrix[i][j] = graph[i][j];

// Adding vertices individually

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

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

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

if (matrix[i][k] + matrix[k][j] < matrix[i][j])

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

}

}

}

printMatrix(matrix);

}

void printMatrix(int matrix[][nV]) {

for (int i = 0; i < nV; i++) {

for (int j = 0; j < nV; j++) {

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

printf("%4s", "INF");

else

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

}

printf("\n");

}

}

int main() {

int graph[nV][nV] = {{0, 3, INF, 5},

{2, 0, INF, 4},

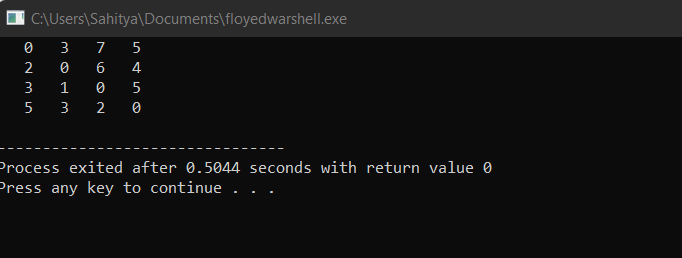
{INF, 1, 0, INF},

{INF, INF, 2, 0}};

floydWarshall(graph);

}

Output:



**10. Bellmanford Algorithm**

Code:

#include <stdio.h>

#include <stdlib.h>

#define INFINITY 99999

//struct for the edges of the graph

struct Edge {

int u; //start vertex of the edge

int v; //end vertex of the edge

int w; //weight of the edge (u,v)

};

//Graph - it consists of edges

struct Graph {

int V; //total number of vertices in the graph

int E; //total number of edges in the graph

struct Edge \*edge; //array of edges

};

void bellmanford(struct Graph \*g, int source);

void display(int arr[], int size);

int main(void) {

//create graph

struct Graph \*g = (struct Graph \*)malloc(sizeof(struct Graph));

g->V = 4; //total vertices

g->E = 5; //total edges

//array of edges for graph

g->edge = (struct Edge \*)malloc(g->E \* sizeof(struct Edge));

g->edge[0].u = 0;

g->edge[0].v = 1;

g->edge[0].w = 5;

//edge 0 --> 2

g->edge[1].u = 0;

g->edge[1].v = 2;

g->edge[1].w = 4;

//edge 1 --> 3

g->edge[2].u = 1;

g->edge[2].v = 3;

g->edge[2].w = 3;

//edge 2 --> 1

g->edge[3].u = 2;

g->edge[3].v = 1;

g->edge[3].w = 6;

//edge 3 --> 2

g->edge[4].u = 3;

g->edge[4].v = 2;

g->edge[4].w = 2;

bellmanford(g, 0); //0 is the source vertex

return 0;

}

void bellmanford(struct Graph \*g, int source) {

//variables

int i, j, u, v, w;

//total vertex in the graph g

int tV = g->V;

//total edge in the graph g

int tE = g->E;

int d[tV];

int p[tV];

//step 1: fill the distance array and predecessor array

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

d[i] = INFINITY;

p[i] = 0;

}

//mark the source vertex

d[source] = 0;

//step 2: relax edges |V| - 1 times

for (i = 1; i <= tV - 1; i++) {

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

//get the edge data

u = g->edge[j].u;

v = g->edge[j].v;

w = g->edge[j].w;

if (d[u] != INFINITY && d[v] > d[u] + w) {

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

p[v] = u;

}

}

}

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

u = g->edge[i].u;

v = g->edge[i].v;

w = g->edge[i].w;

if (d[u] != INFINITY && d[v] > d[u] + w) {

printf("Negative weight cycle detected!\n");

return;

}

}

printf("Distance array: ");

display(d, tV);

printf("Predecessor array: ");

display(p, tV);

}

void display(int arr[], int size) {

int i;

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

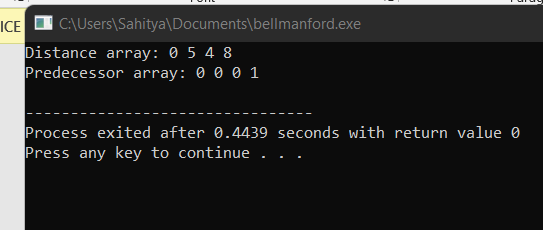
printf("%d ", arr[i]);

}

printf("\n");

}

**Output:**



**11.   
C program for 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 Queens Problem Using Backtracking -");

printf("\n\nEnter number of Queens:");

scanf("%d",&n);

queen(1,n);

return 0;

}

//function for printing the solution

void print(int n)

{

int i,j;

printf("\n\nSolution %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) //for nxn board

{

if(board[i]==j)

printf("\tQ"); //queen at i,j position

else

printf("\t-"); //empty slot

}

}

}

/\*funtion to check conflicts

If no conflict for desired postion returns 1 otherwise returns 0\*/

int place(int row,int column)

{

int i;

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

{

//checking column and digonal conflicts

if(board[i]==column)

return 0;

else

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

return 0;

}

return 1; //no conflicts

}

//function to check for proper positioning of queen

void queen(int row,int n)

{

int column;

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

{

if(place(row,column))

{

board[row]=column; //no conflicts so place queen

if(row==n) //dead end

print(n); //printing the board configuration

else //try queen with next position

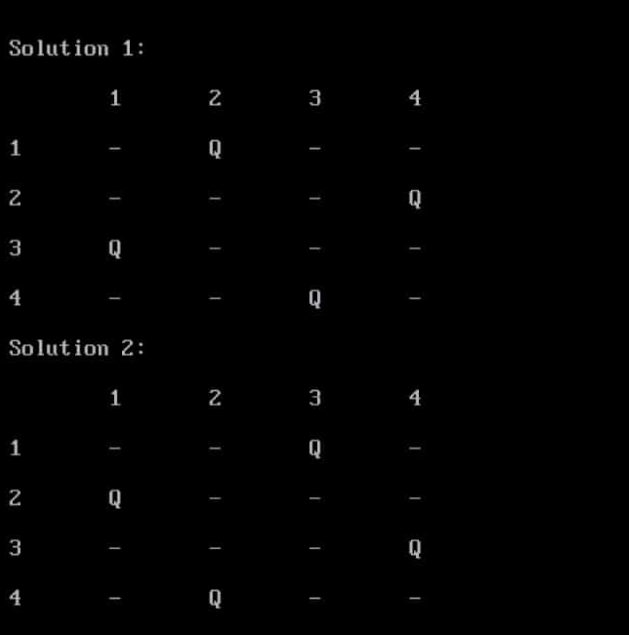
queen(row+1,n);

}

}

}

**Output:**



**12. KMP Algo**

Code:

#include<stdio.h>

#include<string.h>

#include<stdlib.h>

void computeLPSArray(char \*pat, int M, int \*lps);

void KMPSearch(char \*pat, char \*txt)

{

int M = strlen(pat);

int N = strlen(txt);

// create lps[] that will hold the longest prefix suffix

// values for pattern

int \*lps = (int \*)malloc(sizeof(int)\*M);

int j = 0; // index for pat[]

// Preprocess the pattern (calculate lps[] array)

computeLPSArray(pat, M, lps);

int i = 0; // index for txt[]

while (i < N)

{

if (pat[j] == txt[i])

{

j++;

i++;

}

if (j == M)

{

printf("Found pattern at index %d \n", i-j);

j = lps[j-1];

}

// mismatch after j matches

else if (i < N && pat[j] != txt[i])

{

// Do not match lps[0..lps[j-1]] characters,

// they will match anyway

if (j != 0)

j = lps[j-1];

else

i = i+1;

}

}

free(lps); // to avoid memory leak

}

void computeLPSArray(char \*pat, int M, int \*lps)

{

int len = 0; // length of the previous longest prefix suffix

int i;

lps[0] = 0; // lps[0] is always 0

i = 1;

// the loop calculates lps[i] for i = 1 to M-1

while (i < M)

{

if (pat[i] == pat[len])

{

len++;

lps[i] = len;

i++;

}

else // (pat[i] != pat[len])

{

if (len != 0)

{

// This is tricky. Consider the example

// AAACAAAA and i = 7.

len = lps[len-1];

// Also, note that we do not increment i here

}

else // if (len == 0)

{

lps[i] = 0;

i++;

}

}

}

}

// Driver program to test above function

int main()

{

char \*txt = "ABABDABACDABABCABAB";

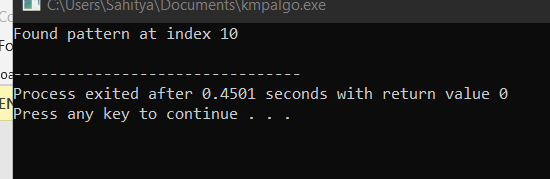
char \*pat = "ABABCABAB";

KMPSearch(pat, txt);

return 0;

}

Code:



Thank You