**Program :**

                dist[v] = dist[u] + weight;

                parent[v] = u;

            }

        }

    }

    for (int i = 0; i < E; i++) {

        int u = graph->edge[i].src;

        int v = graph->edge[i].dest;

        int weight = graph->edge[i].weight;

        if (dist[u] != 9999

            && dist[u] + weight < dist[v]) {

            cout << "Graph contains negative weight cycle";

            return;

        }

    }

    cout << "Shortest path estimate" << endl;

    printArr(dist, V); cout << endl;

    cout << "Predecessor" << endl;

    printArr(parent,V); cout << endl;

    cout << "Single source shortest path exists" << endl << endl;

    for (int i = 0; i < V; i++){

        if (i != src && dist[i] < 9999) {

            cout <<"Path from " << src << " to " << i << " is ";

            printPath(parent, i, src);

            cout << endl;

        }

    }

    return;

}

int main(){

    cout << "Enter number of vertices and edges of the graphs\n";

    int V,E; cin >> V >> E;

    struct Graph\* graph = createGraph(V, E);

    cout << "Enter source vertex, destination vertex and the weight between them\n";

    for(int i=0;i<E;i++) {

        cin >> graph->edge[i].src;

        cin >> graph->edge[i].dest;

        cin >> graph->edge[i].weight;

    }

    printf("Edge table\n");

    for(int i=0;i<E;i++) {

        cout << graph->edge[i].src << " ---> " << graph->edge[i].dest << " : " << graph->edge[i].weight << endl;

    }

    cout << "Enter source vertex in graph" << endl;

    int src; cin >> src;

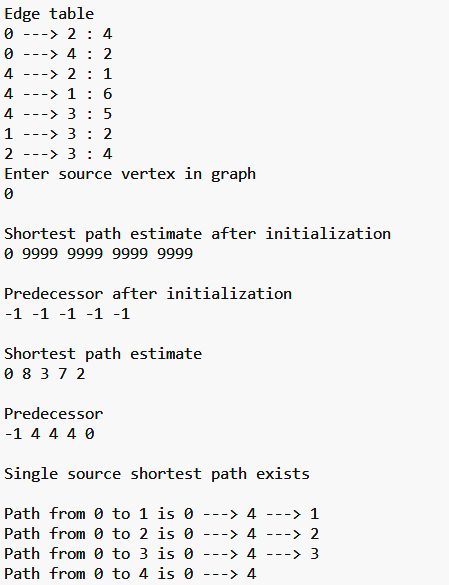
    cout << endl;

    BellmanFord(graph, src);

    return 0;

}

**Output :**

****

**Program :**

#include <bits/stdc++.h>

using namespace std;

bool canPlace(vector<vector<char>> mat, int r, int c) {

    int N = mat.size(); //checking for rows

    for (int i=0; i<r; i++) {

        if (mat[i][c] == 'Q') {

            return false;

        }

    } //checking for cols

    for (int i=r, j=c; i>=0 && j>=0; i--, j--) {

        if (mat[i][j] == 'Q') {

            return false;

        }

    }

    //checking for diagonals

    for (int i=r, j=c; i>=0 && j<N; i--, j++) {

        if (mat[i][j] == 'Q') {

            return false;

        }

    } // if no queen affects this one, then return true

    return true;

}

void print(vector<vector<char>> mat)

{

    int N = mat.size();

    for(int i=0; i<N; i++) {

        cout << " \_\_";

    }

    cout << endl;

    for (int i=0; i<N; i++) {

        for (int j=0; j<N; j++) {

            cout << "|" << mat[i][j] << '\_';

        }

        cout << "|" << endl;

    }

    cout << "\n";

}

void nQueen(vector<vector<char>> mat, int r) {

    int N = mat.size();

    if (r == N) {

        print(mat);

        return;

    }

    for (int i=0; i<N; i++) {

        if (canPlace(mat, r, i)) {

            mat[r][i] = 'Q';

            nQueen(mat, r + 1);

            mat[r][i] = '\_';

        }

    }

}

int main() {

    cout << "Enter number of queens:\n";

    int n;

    cin >> n;

    cout << endl;

    vector<vector<char>> mat(n, vector<char>(n));

    for(int i=0; i<n; i++) {

        for(int j=0; j<n; j++) {

            mat[i][j] = '\_';

        }

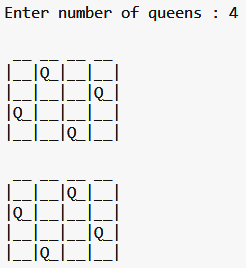
    }

    nQueen(mat, 0);

    return 0;

}

**Output :**

****

**Program :**

#include <stdio.h>

#define TRUE 1

#define FALSE 0

int inc[50], w[50], sum, n;

int promising(int i, int wt, int total){

    return (((wt + total) >= sum) && ((wt == sum) || (wt + w[i + 1] <= sum)));

}

void sumOfSubset(int i, int wt, int total){

    int j;

    if (promising(i, wt, total)){

        if (wt == sum){

            printf("{");

            for (j = 0; j <= i; j++){

                if (inc[j]){

                    printf("%d", w[j]);

                    if(j!=i) printf(", ");

                }

            }

            printf("}\n");

        }

        else{

            inc[i + 1] = TRUE;

            sumOfSubset(i + 1, wt + w[i + 1], total - w[i + 1]);

            inc[i + 1] = FALSE;

            sumOfSubset(i + 1, wt, total - w[i + 1]);

        }

    }

}

int main(){

    int i, j, n, temp, total = 0;

    printf("Enter number of elements in the set : ");

    scanf("%d", &n);

    printf("Enter %d number to add to the set : ", n);

    for (i = 0; i < n; i++){

        scanf("%d", &w[i]);

        total += w[i];

    }

    printf("Input the sum value to create sub set : ");

    scanf("%d", &sum);

    //sort in ascending order

    for (i = 0; i <= n; i++)

        for (j = 0; j < n - 1; j++)if (w[j] > w[j + 1]){

            temp = w[j];

            w[j] = w[j + 1];

            w[j + 1] = temp;

        }

    printf("\nThe given %d numbers in ascending order: \n", n);

    for (i = 0; i < n; i++)

        printf("%d\t", w[i]);

    if ((total < sum))

        printf("\nSubset cannot be made.");

    else{

        for (i = 0; i < n; i++)

            inc[i] = 0;

        printf("\nThe solutions is/are :\n");

        sumOfSubset(-1, 0, total);

    }

    return 0;

}

**Output :**

