**LCS**

#include <bits/stdc++.h>

using namespace std;

int b[1000][1000];

void print\_LCS(int b[100][1000], char st1[], int i, int j)

{

    if (i == 0 || j == 0)

        return;

    if (b[i][j] == 3)

    {

        print\_LCS(b, st1, i - 1, j - 1);

        cout << st1[i - 1];

    }

    else if (b[i][j] == 1)

        print\_LCS(b, st1, i - 1, j);

    else

        print\_LCS(b, st1, i, j - 1);

}

void LCS(char st1[], char st2[])

{

    int m = strlen(st1);

    int n = strlen(st2);

    int c[m + 1][n + 1];

    for (int i = 0; i < m + 1; i++)

        c[i][0] = 0;

    for (int i = 1; i < n + 1; i++)

        c[0][i] = 0;

    for (int i = 1; i < m + 1; i++)

    {

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

        {

            if (st1[i - 1] == st2[j - 1])

            {

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

                b[i][j] = 3;

            }

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

            {

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

                b[i][j] = 1;

            }

            else

            {

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

                b[i][j] = 2;

            }

        }

    }

    cout << "\nTotal length of characters : " << c[m][n] << endl;

    cout << "LCS string : ";

    print\_LCS(b, st1, m, n);

    cout << endl;

}

int main()

{

    // string str1,str2;

    // memset(dp,-1,sizeof(dp));

    char str1[100], str2[1000];

    cout << "Enter the first string : ";

    cin >> str1;

    cout << "Enter the second string : ";

    cin >> str2;

    // int m=str1.length();

    // int n=str2.length();

    // char st1[m+1];

    // char st2[n+1];

    LCS(str1, str2);

    return 0;

}

**BELLMAN FORD**

#include <bits/stdc++.h>

#define inf INT32\_MAX;

int G[10][10], d[10], p[10], edge[10][3];

using namespace std;

void readGraph(int V)

{

    cout << "Enter the graph matrix : " << endl;

    for (int i = 0; i < V; i++)

    {

        for (int j = 0; j < V; j++)

            cin >> G[i][j];

    }

}

void displayGraph(int V)

{

    for (int i = 0; i < V; i++)

    {

        for (int j = 0; j < V; j++)

            cout << G[i][j] << "\t";

        cout << endl;

    }

}

void createTable(int V, int E)

{

    int k = 0;

    for (int i = 0; i < V; i++)

    {

        for (int j = 0; j < V; j++)

        {

            if (G[i][j] != 0)

            {

                edge[k][0] = i;

                edge[k][1] = j;

                edge[k][2] = G[i][j];

                k++;

            }

            else

                continue;

        }

        cout << "Edge Table : " << endl;

        for (int i = 0; i < E; i++)

            cout << edge[i][0] << " --> " << edge[i][1] << " : " << edge[i][2] << endl;

    }

}

void initialize(int V, int s)

{

    for (int i = 0; i < V; i++)

    {

        d[i] = inf;

        p[i] = -1;

    }

    d[s] = 0;

}

void Relax(int u, int v, int w)

{

    if (d[v] > d[u] + w)

    {

        d[v] = d[u] + w;

        p[v] = u;

    }

}

int negCycle(int E)

{

    for (int i = 0; i < E; i++)

    {

        if (edge[i][1] > edge[i][0] + edge[i][2])

            return 0;

        return 1;

    }

}

void printPath(int s, int v)

{

    if (v == s)

        cout << s;

    else if (p[v] == -1)

        cout << "No path exists";

    else

    {

        printPath(s, p[v]);

        cout << " --> " << v;

    }

}

int main()

{

    int V, E, s;

    cout << "Enter the number of vertices : ";

    cin >> V;

    cout << "Enter the number of Edges : ";

    cin >> E;

    readGraph(V);

    cout << "Graph is : " << endl;

    displayGraph(V);

    createTable(V, E);

    cout << "Enter the source vertex : ";

    cin >> s;

    initialize(V, s);

    cout << "\n shortest path estimate after initialization : " << endl;

    for (int i = 0; i < V; i++)

        cout << d[i] << "\t";

    cout << endl;

    cout << "\n predecessor after initialization : " << endl;

    for (int i = 0; i < V; i++)

        cout << p[i] << "\t";

    for (int i = 0; i < V - 1; i++)

    {

        for (int j = 0; j < E; j++)

            Relax(edge[j][0], edge[j][1], edge[j][2]);

    }

    int flag = negCycle(E);

    if (flag == 0)

    {

        cout << "Negative weight cycle exists" << endl;

        return;

    }

    else

        cout << "SSSP exists" << endl;

    cout << "Printing path : " << endl;

    for (int i = 0; i < V; i++)

    {

        if (i == s)

            continue;

        cout << "Path from " << s << " to " << i << " : ";

        printPath(s, i);

        cout << endl;

    }

    return 0;

}

**PRIM**

#include "stdio.h"

#include "stdlib.h"

#define INF 9999

int V, E = 0;

int G[10][10];

int p[10], visited[10], d[10];

int startvertex;

void readGraph()

{

    printf("\nEnter adjacenc matrix \n");

    for (int i = 0; i < V; i++)

    {

        for (int j = 0; j < V; j++)

        {

            scanf("%d", &G[i][j]);

            if (G[i][j] != 0)

                E++;

        }

    }

}

void displaygraph()

{

    printf("\nGraph is... \n");

    for (int i = 0; i < V; i++)

    {

        for (int j = 0; j < V; j++)

        {

            printf("%d\t", G[i][j]);

        }

        printf("\n");

    }

}

void initialize()

{

    for (int i = 0; i < V; i++)

    {

        visited[i] = 0;

        d[i] = INF;

        p[i] = -1;

    }

}

void print1Darray(char \*str, int a[])

{

    printf("\n%s [] : ", str);

    for (int i = 0; i < V; i++)

    {

        printf("\t%d", a[i]);

    }

}

void print\_vis\_d\_p()

{

    print1Darray("visited", visited);

    print1Darray("d array", d);

    print1Darray("p array", p);

}

void prims()

{

    int curr, totalvis, mincost, i;

    printf("\n\nEnter starting vertex : ");

    scanf("%d", &curr);

    startvertex = curr;

    // curr = 0;

    d[curr] = 0;

    visited[curr] = 1;

    totalvis = 1;

    printf("\n order in which edges are selected in MST: \n");

    while (totalvis != V)

    {

        for (int i = 0; i < V; i++)

        {

            if (G[curr][i] != 0)

            {

                if (visited[i] == 0)

                {

                    if (d[i] > G[curr][i])

                    {

                        d[i] = G[curr][i];

                        p[i] = curr;

                    }

                }

            }

        } // end of for

        mincost = 8999;

        for (int i = 0; i < V; i++)

        {

            if (visited[i] == 0)

            {

                if (d[i] < mincost)

                {

                    mincost = d[i];

                    curr = i;

                }

            }

        }

        printf("\nEdge (%d,%d)", p[curr], curr);

        visited[curr] = 1;

        totalvis++;

    }

    printf("\n\nFinal Values are ....");

    print\_vis\_d\_p();

    mincost = 0;

    printf("\n\n Edges in minimum spanning tree are :\n ");

    for (int i = 0; i < V; i++)

    {

        mincost += d[i];

        if (p[i] != -1)

            printf("\n Edge (%d,%d) : %d ", p[i], i, G[p[i]][i]);

    }

    printf("\n\nTotal Cost : %d \n", mincost);

}

void printpath(int s, int v)

{

    if (v == s)

        printf(" %d ", s);

    else if (v == startvertex)

    {

        printf(" %d ", startvertex);

    }

    else

    {

        printpath(s, p[v]);

        printf("-->");

        printf(" %d ", v);

    }

}

void main()

{

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

    scanf("%d", &V);

    readGraph();

    displaygraph();

    printf("No.of vertex : %d\nNo.of Edges : %d\n", V, E);

    initialize();

    printf("\nAfter initialization....");

    print\_vis\_d\_p();

    prims();

    printf("\nPrinting Path\n");

    for (int i = 0; i < V; i++)

    {

        if (i == startvertex)

            continue;

        printf("path from %d to %d is ", startvertex, i);

        printpath(startvertex, i);

        printf("\n");

    }

}

**KRUSKAL**

#include <bits/stdc++.h>

using namespace std;

class DSU

{

    int \*parent, \*rank;

public:

    DSU(int n)

    {

        parent = new int[n];

        rank = new int[n];

        for (int i = 0; i < n; i++)

        {

            parent[i] = -1;

            rank[i] = 1;

        }

    }

    int find(int i)

    {

        if (parent[i] == -1)

            return i;

        return parent[i] = find(parent[i]);

    }

    void unite(int x, int y)

    {

        int s1 = find(x);

        int s2 = find(y);

        if (s1 != s2)

        {

            if (rank[s1] < rank[s2])

            {

                parent[s1] = s2;

                rank[s2] += rank[s1];

            }

            else

            {

                parent[s2] = s1;

                rank[s1] += rank[s2];

            }

        }

    }

};

class Graph

{

    vector<vector<int>> EdgeList;

    int V;

public:

    Graph(int V)

    {

        this->V = V;

    }

    void addEdge(int x, int y, int w)

    {

        EdgeList.push\_back({w, x, y});

    }

    void Kruskal()

    {

        sort(EdgeList.begin(), EdgeList.end());

        DSU dsu(V);

        int ans = 0;

        for (auto edge : EdgeList)

        {

            int w = edge[0];

            int x = edge[1];

            int y = edge[2];

            if (dsu.find(x) != dsu.find(y))

            {

                dsu.unite(x, y);

                ans += w;

                cout << x << " --- " << y << "  ==  " << w << endl;

            }

        }

        cout << "Cost of minimum spanning tree : " << ans << endl;

    }

};

int main()

{

    Graph g(4);

    g.addEdge(0, 1, 10);

    g.addEdge(1, 3, 15);

    g.addEdge(2, 3, 4);

    g.addEdge(2, 0, 6);

    g.addEdge(0, 3, 5);

    g.Kruskal();

    return 0;

}

**NQUEEN**

#include <bits/stdc++.h>

using namespace std;

#define N 4

bool isSafe(int board[N][N], int row, int col)

{

    for (int i = 0; i < col; i++)

        if (board[row][i])

            return false;

    for (int i = row, j = col; i >= 0 && j >= 0; i--, j--)

        if (board[i][j])

            return false;

    for (int i = row, j = col; i < N && j >= 0; i++, j--)

        if (board[i][j])

            return false;

    return true;

}

bool solveNQ(int board[N][N], int col)

{

    if (col >= N)

        return true;

    for (int i = 0; i < N; i++)

    {

        if (isSafe(board, i, col))

        {

            board[i][col] = 1;

            if (solveNQ(board, col + 1))

                return true;

            board[i][col] = 0;

        }

    }

    return false;

}

bool NQ()

{

    int board[N][N];

    for (int i = 0; i < N; i++)

    {

        for (int j = 0; j < N; j++)

            board[i][j] = 0;

    }

    if (solveNQ(board, 0) == false)

    {

        cout << "\n\nSolution does not exist" << endl;

        return false;

    }

    cout << "Thse solution is : " << endl;

    for (int i = 0; i < N; i++)

    {

        for (int j = 0; j < N; j++)

            cout << board[i][j] << "  ";

        cout << endl;

    }

    return true;

}

int main()

{

    NQ();

    return 0;

}

**SUM OF SUBSET**

#include <bits/stdc++.h>

using namespace std;

int x[100] = {0}, m, w[100], n, cnt = 0, nosolns = 0;

void displaySubset(int x[], int k, int num)

{

    printf("Subset %d : {", num);

    for (int i = 0; i <= k; i++)

        if (x[i] == 1)

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

    printf("}\n------------------------------------------");

}

void display(int x[], int k)

{

    nosolns += 1;

    printf("\n\nSolution %d : {", nosolns);

    for (int i = 0; i <= k; i++)

        cout << " " << x[i] << " ";

    if (k < n)

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

            cout << " 0 ";

    printf("}\n");

    displaySubset(x, k, nosolns);

}

void SumOfSubset(int s, int r, int k)

{

    cnt += 1;

    x[k] = 1;

    if (s + w[k] == m)

        display(x, k);

    else if (s + w[k] + w[k + 1] <= m)

        SumOfSubset(s + w[k], r - w[k], k + 1);

    if ((s + r - w[k] >= m) && (s + w[k + 1] <= m))

    {

        x[k] = 0;

        SumOfSubset(s, r - w[k], k + 1);

    }

}

int main()

{

    cout << "Enter the total sum required : ";

    cin >> m;

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

    cin >> n;

    int r = 0;

    for (int i = 0; i < n; i++)

    {

        cin >> w[i];

        r += w[i];

    }

    SumOfSubset(0, r, 0);

    cout << "\nNumber of Nodes : " << cnt << endl;

    return 0;

}

**FLOYD WARSHALL**

#include <stdio.h>

#include <string.h>

#define INF 999

int D[10][10];

int P[10][10];

int V, E;

int G[10][10];

// Reading Graph

void readGraph()

{

    int i = 0, j = 0;

    printf("\nEnter graph in matrix form\n");

    for (i = 0; i < V; i++)

    {

        for (j = 0; j < V; j++)

        {

            scanf("%d", &G[i][j]);

        }

    }

}

// Displaying graph

void displayGraph(int G[][10])

{

    for (int i = 0; i < V; i++)

    {

        for (int j = 0; j < V; j++)

            printf("%d\t", G[i][j]);

        printf("\n");

    }

    printf("\n");

}

// Initializing

void initialize()

{

    for (int i = 0; i < V; i++)

    {

        for (int j = 0; j < V; j++)

        {

            if (i == j)

            {

                D[i][j] = 0;

                P[i][j] = -1;

            }

            else if (G[i][j] == 0)

            {

                D[i][j] = INF;

                P[i][j] = -1;

            }

            else

            {

                D[i][j] = G[i][j];

                P[i][j] = i;

            }

        }

    }

}

// floyd warshall

void floydwarshall()

{

    int i, j, k;

    for (k = 0; k < V; k++)

    {

        for (i = 0; i < V; i++)

        {

            for (j = 0; j < V; j++)

            {

                if (D[i][j] <= D[i][k] + D[k][j])

                    continue;

                else

                {

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

                    P[i][j] = P[k][j];

                }

            }

        }

    }

}

// Printing Path

void printpath(int s, int v)

{

    if (v == s)

        printf(" %d  ", s);

    else if (P[s][v] == -1)

        printf("No path exist");

    else

    {

        printpath(s, P[s][v]);

        printf("-->");

        printf(" %d ", v);

    }

}

int main()

{

    printf("\n Enter number of vertices : ");

    scanf("%d", &V);

    readGraph();

    displayGraph(G);

    initialize();

    printf("\nAfter initialization...D and P matrix\n");

    printf("\n D Matrix...\n");

    displayGraph(D);

    printf("\n P Matrix...\n");

    displayGraph(P);

    floydwarshall();

    printf("\nAfter Floyd Warshall...D and P matrix\n");

    printf("\n D Matrix...\n");

    displayGraph(D);

    printf("\n P Matrix...\n");

    displayGraph(P);

    printf("\nPrinting Path\n");

    for (int i = 0; i < V; i++)

    {

        for (int j = 0; j < V; j++)

        {

            if (i == j)

                continue;

            printf("path from %d to %d is ", i, j);

            printpath(i, j);

            printf("\n");

        }

    }

    return 0;

}

**RABIN KARP**

#include <bits/stdc++.h>

#include <cstring>

#include <math.h>

using namespace std;

#define d 256

void search(char pat[], char txt[], int q)

{

    int M = strlen(pat);

    int N = strlen(txt);

    int i, j, p = 0, t = 0, h = 1;

    for (i = 0; i < M - 1; i++)

        h = (h \* d) % q;

    for (i = 0; i < M; i++)

    {

        p = (d \* p + pat[i]) % q;

        t = (d \* t + txt[i]) % q;

    }

    for (i = 0; i <= N - M; i++)

    {

        if (p == t)

        {

            bool flag = true;

            for (j = 0; j < M; j++)

            {

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

                {

                    flag = false;

                    break;

                }

            }

            if (j == M)

                cout << "Pattern found at index " << i << endl;

        }

        if (i < N - M)

        {

            t = (d \* (t - txt[i] \* h) + txt[i + M]) % q;

            if (t < 0)

                t = (t + q);

        }

    }

}

int main()

{

    string text;

    string pattern;

    cout << "Enter the text : ";

    getline(cin, text);

    cout << "Enter the pattern : ";

    getline(cin, pattern);

    char txt[text.length() + 1];

    char pat[pattern.length() + 1];

    strcpy(txt, text.c\_str());

    strcpy(pat, pattern.c\_str());

    int q = 101;

    search(pat, txt, q);

    return 0;

}

**KMP**

#include <bits/stdc++.h>

#include <cstring>

using namespace std;

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

{

    int len = 0;

    lps[0] = 0;

    int i = 1;

    while (i < M)

    {

        if (pat[i] == pat[len])

        {

            len++;

            lps[i] = len;

            i++;

        }

        else

        {

            if (len != 0)

            {

                len = lps[len - 1];

            }

            else

            {

                lps[i] = 0;

                i++;

            }

        }

    }

}

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

{

    int M = strlen(pat);

    int N = strlen(txt);

    int lps[M];

    computeLPSArray(pat, M, lps);

    int i = 0;

    int j = 0;

    while (i < N)

    {

        if (pat[j] == txt[i])

        {

            j++;

            i++;

        }

        if (j == M)

        {

            cout << "Found pattern at index " << i - j << endl;

            j = lps[j - 1];

        }

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

        {

            if (j != 0)

                j = lps[j - 1];

            else

                i = i + 1;

        }

    }

}

int main()

{

    string text;

    string pattern;

    cout << "Enter the text : ";

    getline(cin, text);

    cout << "Enter the pattern : ";

    getline(cin, pattern);

    char txt[text.length() + 1];

    char pat[pattern.length() + 1];

    strcpy(txt, text.c\_str());

    strcpy(pat, pattern.c\_str());

    KMPSearch(pat, txt);

    return 0;

}

**01 KNAPSACK**

#include <bits/stdc++.h>

using namespace std;

int k[10][10];

int wt[10], p[10];

void knapsack(int m, int n)

{

    int k[n + 1][m + 1];

    for (int i = 0; i <= n; i++)

    {

        for (int w = 0; w <= m; w++)

        {

            if (i == 0 || w == 0)

                k[i][w] = 0;

            else if (wt[i] <= w)

            {

                k[i][w] = max(p[i] + k[i - 1][w - wt[i]], k[i - 1][w]);

            }

            else

                k[i][w] = k[i - 1][w];

        }

    }

    cout << "\nThe maximum profit is  " << k[n][m] << endl;

    int i = n, j = m;

    cout << "\nelements position that  are added : ";

    while (i > 0 and j > 0)

    {

        if (k[i][j] != k[i - 1][j])

        {

            cout << (i + 1) << " ";

            j -= wt[i];

            i--;

        }

        else

        {

            i--;

        }

    }

}

int main()

{

    int m, n;

    cout << "Enter total weight of object" << endl;

    cin >> m;

    cout << "Enter total number of objects " << endl;

    cin >> n;

    cout << "Enter the weight and value of object  below " << endl;

    for (int i = 0; i < n; i++)

    {

        cout << "Enter the weight of object and value for object" << (i + 1) << " ";

        cin >> wt[i] >> p[i];

    }

    knapsack(m, n);

    return 0;

}

**FRACTIONAL KNAPSACK**

#include <bits/stdc++.h>

#include <algorithm>

using namespace std;

bool cmp(double a, double b)

{

    return (a > b);

}

double knapsack(int cap, double wt[], double val[   ], int n)

{

    double x1[n], x2[n], x[n], r[n], finalval = 0.0;

    int i, currentwt = 0, j, remain = 0;

    for (i = 0; i < n; i++)

    {

        r[i] = val[i] / wt[i];

    }

    sort(r, r + n, cmp);

    for (i = 0; i < n; i++)

    {

        for (j = 0; j < n; j++)

        {

            if (r[i] == (double)val[j] / wt[j])

            {

                x1[i] = val[j];

                x2[i] = wt[j];

                val[j] = 0;

                wt[j] = 0;

            }

        }

    }

    for (i = 0; i < n; i++)

    {

        if (currentwt + x2[i] <= cap)

        {

            currentwt += x2[i];

            finalval += x1[i];

        }

        else

        {

            remain = cap - currentwt;

            finalval += (double)r[i] \* (double)remain;

            break;

        }

    }

    return finalval;

}

int main()

{

    int n, cap, i;

    cout << "Enter the capacity" << endl;

    cin >> cap;

    cout << "Enter the number of elements" << endl;

    cin >> n;

    double wt[n], val[n];

    for (i = 0; i < n; i++)

    {

        cout << "Enter the weight for " << i + 1 << " : " << endl;

        cin >> wt[i];

        cout << "Enter the value for " << i + 1 << " : " << endl;

        cin >> val[i];

    }

    cout << knapsack(cap, wt, val, n);

    return 0;

}