**Dijkstra:**

#include<stdio.h>

int par[20], v[20], dist[20], n, s, i, j,k;

int graph[6][6]={{0, 4, 0, 0, 0, 0},

                {4, 0, 8, 0, 0, 0},

                {0, 8, 0, 7, 0, 4},

                {0, 0, 7, 0, 9, 14},

                {0, 0, 0, 9, 0, 10},

                {0, 0, 4, 14, 10, 0}};

void initialise\_single\_source(){

    for(i = 0;i<n;i++){

        dist[i] = 10000;

        par[i]=-1;

    }

    dist[s] = 0;

}

int find\_min(){

    int min=10000;

    int temp=-1;

    // int i;

    for(k=0;k<n;k++){

        if(dist[k]<min && v[k]==0){

            temp=k;

            min=dist[k];

        }

    }

    v[temp]=1;

    return temp;

}

void relax(int u, int v){

    if(dist[v]>(dist[u]+graph[u][v])){

        dist[v]=(dist[u]+graph[u][v]);

        par[v]=u;

    }

}

void display(){

    printf("dist\tpar\n");

    for(i = 0;i<n;i++){

        printf("%d\t %d\n",dist[i],par[i]);

    }

}

void Dijkstra(){

    initialise\_single\_source();

    for(i = 0;i<n;i++){

        int u = find\_min();

        for(j=0;j<n;j++){

            if(graph[u][j]!=0 && v[j]==0){

                relax(u,j);

            }

        }

    }

    display();

}

int main(){

    n=6;

    for(i=0;i<n;i++){

        for(j=0;j<n;j++){

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

        }

        printf("\n");

    }

    printf("Enter source: ");

    scanf("%d",&s);

    Dijkstra();

    return 0;

}

**Bellman:**

#include<stdio.h>

#include<stdlib.h>

int dist[20], edges[20][3];

void bellman(int nodes, int ecount, int start){

    int source, destination, weight;

    for(int i=0;i<nodes;i++){

        if(i==start){

            dist[i]=0;

        }

        else{

            dist[i]=1000;

        }

    }

    for(int i=0;i<nodes;i++){

        for(int j=0;j<ecount;j++){

            source = edges[j][0];

            destination = edges[j][1];

            weight = edges[j][2];

            if(dist[source]!=1000 && dist[destination]>dist[source]+weight){

                dist[destination]=dist[source]+weight;

            }

        }

    }

    for(int j=0;j<ecount;j++){

            source = edges[j][0];

            destination = edges[j][1];

            weight = edges[j][2];

            if(dist[source]!=1000 && dist[destination]>dist[source]+weight){

                printf("negative loop");

                return;

            }

        }

        for(int i=0;i<nodes;i++){

            printf("shortest path from %d to %d is:\t %d\n",start, i,dist[i]);

        }

}

void main(){

    int start, nodes, ecount;

    printf("Enter start node, number of nodes and number of edges: ");

    scanf("%d%d%d",&start, &nodes, &ecount);

    printf("Enter edges as source - destination - weight \n");

    for(int i=0;i<ecount;i++){

        scanf("%d%d%d",&edges[i][0],&edges[i][1],&edges[i][2]);

    }

    bellman(nodes, ecount, start);

}

**Graph color:**

#include<stdio.h>

int graph[10][10], i,j,count = 0, n,m,x[10],f=0;

void nextvalue(int k){

    while(1){

        x[k] = (x[k]+1)%(m+1);

        if(x[k]==0){

            return;

        }

        for(j=1;j<=n;j++){

            if(graph[k][j]==1 && x[k]==x[j]){

                break;

            }

        }

        if(j==(n+1)){

            return;

        }

    }

}

void graphcolouring(int k){

    while(1){

        nextvalue(k);

        if(x[k]==0){

            return;

        }

        if(k==n){

            f = 1;

            for(i=1;i<=n;i++)

            {

            printf("%d ", x[i]);

            }

            count++;

            printf("\n");

            }

        else{

            graphcolouring(k+1);

        }

    }

    }

void main(){

    printf("number of nodes: ");

    scanf("%d",&n);

    printf("Enter Adjacency Matrix: ");

    for(i = 1; i<=n;i++){

        x[i]=0;

        for(j=1;j<=n;j++){

            scanf("%d",&graph[i][j]);

        }

    }

    printf("Possible Solutions: ");

    for(m=1;m<=n;m++){

        if (f==1){

            break;

        }

        graphcolouring(1);

    }

    printf("\nThe chromatic number is %d", m-1);

    printf("\nThe total number of solutions is %d", count);

}

**Prims**

#include<stdio.h>

int dist[20], par[20], v[20], n, s;

int g[6][6] = {{0, 4, 0, 0, 0, 0},

            {4, 0, 8, 0, 0, 0},

            {0, 8, 0, 7, 0, 4},

            {0, 0, 7, 0, 9, 14},

            {0, 0, 0, 9, 0, 10},

            {0, 0, 4, 14, 10, 0}};

void initialise\_single\_source(){

    for(int i=0;i<n;i++){

        dist[i] = 10000;

        par[i]=-1;

        // v[i]=0;

    }

    dist[s]=0;

}

int find\_min(){

    int min = 1000;

    int temp = -1;

    for(int i=0;i<n;i++){

        if(dist[i]<min && v[i]==0){

            temp = i;

            min = dist[i];

        }

    }

    v[temp]=1;

    return temp;

}

void relax(int u,int v){

    if(dist[v]>g[u][v]){

        dist[v]=g[u][v];

        par[v]=u;

    }

}

void display(){

    int sum = 0;

    printf("dist\tpar\n");

    for(int i=0;i<n;i++){

        printf("%d\t%d\n",dist[i],par[i]);

        sum = sum + dist[i];

    }

    printf("total cost of MST: %d",sum);

}

void prim(){

    initialise\_single\_source();

    for(int i=0;i<n;i++){

        int u = find\_min();

        for(int j=0;j<n;j++){

            if(g[u][j]!=0 && v[j]==0){

                relax(u,j);

            }

        }

    }

    display();

}

void main(){

    n=6;

    printf("Graph:\n");

    for(int i=0;i<n;i++){

        for(int j=0;j<n;j++){

        printf("%d ",g[i][j]);

        }

        printf("\n");

    }

    printf("Enter starting point: ");

    scanf("%d",&s);

    prim();

}

**Kruskal**

#include <stdio.h>

#include <stdlib.h>

// Comparator function to use in sorting

int comparator(const void\* p1, const void\* p2)

{

const int(\*x)[3] = p1;

const int(\*y)[3] = p2;

return (\*x)[2] - (\*y)[2];

}

// Initialization of parent[] and rank[] arrays

void makeSet(int parent[], int rank[], int n)

{

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

parent[i] = i;

rank[i] = 0;

}

}

// Function to find the parent of a node

int findParent(int parent[], int component)

{

if (parent[component] == component)

return component;

return parent[component]

= findParent(parent, parent[component]);

}

// Function to unite two sets

void unionSet(int u, int v, int parent[], int rank[], int n)

{

// Finding the parents

u = findParent(parent, u);

v = findParent(parent, v);

if (rank[u] < rank[v]) {

parent[u] = v;

}

else if (rank[u] > rank[v]) {

parent[v] = u;

}

else {

parent[v] = u;

// Since the rank increases if the ranks of two sets are same

rank[u]++;

}

}

// Function to find the MST

void kruskalAlgo(int n, int edge[n][3])

{

// First we sort the edge array in ascending order

// so that we can access minimum distances/cost

qsort(edge, n, sizeof(edge[0]), comparator);

int parent[n];

int rank[n];

// Function to initialize parent[] and rank[]

makeSet(parent, rank, n);

// To store the minimun cost

int minCost = 0;

printf("Following are the edges in the constructed MST\n");

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

int v1 = findParent(parent, edge[i][0]);

int v2 = findParent(parent, edge[i][1]);

int wt = edge[i][2];

// If the parents are different that

// means they are in different sets so

// union them

if (v1 != v2) {

unionSet(v1, v2, parent, rank, n);

minCost += wt;

printf("%d -- %d == %d\n", edge[i][0],

edge[i][1], wt);

}

}

printf("Minimum Cost Spanning Tree: %d\n", minCost);

}

int main()

{

int edge[5][3] = { { 0, 1, 10 },

{ 0, 2, 6 },

{ 0, 3, 5 },

{ 1, 3, 15 },

{ 2, 3, 4 } };

kruskalAlgo(5, edge);

return 0;

}

**Greedy knapsack**

#include<stdio.h>

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

    float x[20],tp;

    int i,u;

    u = c;

    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);

}

void main(){

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

    int temp,n,i,j,c;

    printf("number of objects: ");

    scanf("%d",&n);

    printf("enter weight and profit of each: ");

    for(i=0;i<n;i++){

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

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

    }

    printf("enter capacity: ");

    scanf("%d",&c);

    for(i=0;i<n;i++){

        for(j=i+1;j<n;j++){

            if(ratio[i]<ratio[j]){

                temp = ratio[i];

                ratio[i] = ratio[j];

                ratio[j] = temp;

                temp = weight[i];

                ratio[i] = weight[j];

                weight[j] = temp;

                temp = profit[i];

                profit[i] = profit[j];

                profit[j] = temp;

            }

        }

    }

    printf("in ascending order\nWeights\tProfits\n");

    for(i=0;i<n;i++){

        printf("%f\t%f\n",weight[i],profit[i]);

    }

    knapsack(n,weight,profit, c);

}

**Sum of subset:**

#include<stdio.h>

#include<stdbool.h>

void subset\_sum(int arr[], int n, int sum, int subset[], int k, int idx, bool\* found) {

    if (sum == 0) {

        // print the subset that adds up to the given sum

        printf("{");

        for (int i = 0; i < k; i++) {

            printf("%d", subset[i]);

            if (i < k-1) printf(", ");

        }

        printf("}\n");

        \*found = true;

        return;

    }

    if (sum < 0 || idx >= n) {

        return;

    }

    // include the current element in the subset

    subset[k] = arr[idx];

    subset\_sum(arr, n, sum-arr[idx], subset, k+1, idx+1, found);

    // exclude the current element from the subset

    subset\_sum(arr, n, sum, subset, k, idx+1, found);

}

int main() {

    int arr[] = {2,3, 4, 5, 6, 7,8};

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

    int sum = 10;

    int subset[n];

    bool found = false;

    printf("All subsets that add up to %d:\n", sum);

    subset\_sum(arr, n, sum, subset, 0, 0, &found);

    if (!found) {

        printf("No subset found that adds up to %d\n", sum);

    }

    return 0;

}

**Rabin karp**

#include <stdio.h>

#include <string.h>

int main()

{

    char txt[80], pat[80];

    int q;

    printf("Enter the container string");

    scanf("%s", &txt);

    printf("Enter the pattern to be searched ");

    scanf("%s", &pat);

    int d = 256;

    printf("Enter a prime number ");

    scanf("%d", &q);

    int M = strlen(pat);

    int N = strlen(txt);

    int i, j;

    int p = 0;

    int t = 0;

    int 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)

        {

            for (j = 0; j < M; j++)

            {

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

                    break;

            }

            if (j == M)

                printf("Pattern found at index %d ", i);

        }

        if (i < N - M)

        {

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

            if (t < 0)

                t = (t + q);

        }

    }

    return 0;

}

**KMP:**

#include<stdio.h>

#include<string.h>

void computeLPS(char par[], int m, int lps[]) {

    int len = 0;

    lps[0] = 0;

    int i=1;

    while(i<m){

        if(par[i]==par[len]){

            len++;

            lps[i] = len;

            i++;

        }

        else{

            if(len!=0){

                len = lps[len-1];

            }

            else{

                lps[i]=0;

                i++;

            }

        }

    }

}

void KMPSearch(char txt[], char par[]){

    int m = strlen(par);

    int n = strlen(txt);

    int lps[20];

    computeLPS(par,m,lps);

        int i =0,j=0;

    while(i<n){

        if(txt[i]==par[j]){

            i++;

            j++;

        }

        if(j==m){

            printf("pattern found at: %d\n", i-j);

            j = lps[j-1];

        }

        else if(i<n && txt[i]!=par[j]){

            if(j!=0){

                j = lps[j-1];

            }

            else{

                i++;

            }

        }

    }

    if(j!=m){

        printf("no more patterns");

    }

}

void main(){

    char txt[] = {"ABABDABACDABABCABAB"};

    char par[] = {"ABAB"};

    KMPSearch(txt, par);

}

**Nqueen:**

#include<stdio.h>

#include<math.h>

#include<stdlib.h>

int a[30],count=0;

int place(int r)

{

    int j;

    for(j=1;j<r;j++)

    {

        if(a[j]==a[r]|| abs(a[j]-a[r])== abs(j-r))

        return 0;

    }

    return 1;

}

void print\_sol(int n)

{

    int i,j;

    count++;

    printf("solution %d\n",count);

    for(i=1;i<=n;i++)

    {

        for(j=1;j<=n;j++)

        {

            if(a[i]==j)

            {

                printf("Q\t");

            }

            else

            printf("\*\t");

        }

        printf("\n");

    }

}

void queen(int n)

{

    int k=1;

    a[k]=0;

    while(k!=0)

    {

        do{

            a[k]++;

        }while(a[k]<=n && !place(k));

        if(a[k]<=n)

        {

            if(k==n)

            {

                print\_sol(n);

            }

            else

            {

                k++;

                a[k]=0;

            }

        }

        else

        k--;

    }

}

void main()

{

    int n;

    printf("Enter no of queens");

    scanf("%d",&n);

    queen(n);

    printf("Total Solutions: %d",count);

}

**LCS**

#include<stdio.h>

#include<string.h>

int m, n,i,j,LCS\_table[20][20];

char s1[20], s2[20], b[20][20];

void lcsalgo(){

    m=strlen(s1);

    n=strlen(s2);

    for (i = 0; i <= m; i++)

        LCS\_table[i][0] = 0;

    for (i = 0; i <= n; i++)

        LCS\_table[0][i] = 0;

    for(i=1;i<=m;i++){

        for(j=1;j<=n;j++){

            if(s1[i-1]==s2[j-1]){

                LCS\_table[i][j] = LCS\_table[i-1][j-1]+1;

            }

            else if(LCS\_table[i-1][j]>=LCS\_table[i][j-1]){

                LCS\_table[i][j] = LCS\_table[i-1][j];

            }

            else{

                LCS\_table[i][j] = LCS\_table[i][j-1];

            }

        }

    }

    int index = LCS\_table[m][n];

    char lcsalgo[index+1];

    lcsalgo[index]='\0';

    int i = m, j = n;

    while(i>0 && j>0){

        if(s1[i-1]==s2[j-1]){

            lcsalgo[index-1]=s1[i-1];

            i--;

            j--;

            index--;

        }

        else if(LCS\_table[i-1][j]>LCS\_table[i][j-1]){

            i--;

        }

        else{

            j--;

        }

    }

    for(i=0;i<m;i++){

        for(j=0;j<n;j++){

            printf("%d ",LCS\_table[i][j]);

        }

        printf("\n");

    }

    printf("s1: %s\ns2: %s\n",s1,s2);

    printf("LCS: %s",lcsalgo);

}

void main(){

    printf("enter the string1: ");

    scanf("%s",s1);

    printf("enter the string2: ");

    scanf("%s",s2);

    lcsalgo();

}