**INSERTION SORT**

#include<stdio.h>

#include<conio.h>

#define SIZE 100

void main()

{

int i,j,n,key;

printf("Enter the length of elements to store in array\n");

scanf("%d",&n);

int a[SIZE];

printf("Enter the elements\n");

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

{

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

}

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

{

key=a[i];

j=i-1;

while(j>=0 && a[j]>=key)

{

a[j+1]=a[j];

j=j-1;

}

a[j+1]=key;

}

printf("The sorted array is:\n");

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

{

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

}

getch(); }

**MERGE SORT**

#include <stdio.h>

#include <conio.h>

#define MAX 100

#define INF 999999

void merge(int a[], int start, int q, int end) {

int left[MAX], right[MAX];

int i, j, k;

int n1 = q - start + 1;

int n2 = end - q;

for(i = 0; i < n1; i++) {

left[i] = a[start + i];

}

for(j = 0; j < n2; j++) {

right[j] = a[q + 1 + j];

}

left[n1] = INF;

right[n2] = INF;

i = 0;

j = 0;

for(k = start; k <= end; k++) {

if(left[i] <= right[j]) {

a[k] = left[i++];

} else {

a[k] = right[j++];

}

}

}

void mergesort(int a[], int start, int end) {

if(start < end) {

int q = (start + end) / 2;

mergesort(a, start, q);

mergesort(a, q + 1, end);

merge(a, start, q, end);

}

}

void main() {

int a[MAX], n, i;

clrscr();

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

scanf("%d", &n);

printf("Enter the elements:\n");

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

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

}

mergesort(a, 0, n - 1);

printf("Sorted array:\n");

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

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

}

    getch();

}

**KTH SMALLEST ELEMENT**

#include <stdio.h>

#include <conio.h>

#define MAX 100

#define INF 999999

void merge(int a[], int start, int q, int end) {

int left[MAX], right[MAX];

int i, j, k;

int n1 = q - start + 1;

int n2 = end - q;

for(i = 0; i < n1; i++) {

left[i] = a[start + i];

}

for(j = 0; j < n2; j++) {

right[j] = a[q + 1 + j];

}

left[n1] = INF;

right[n2] = INF;

i = 0;

j = 0;

for(k = start; k <= end; k++) {

if(left[i] <= right[j]) {

a[k] = left[i++];

} else {

a[k] = right[j++];

}

}

}

void mergesort(int a[], int start, int end) {

if(start < end) {

int q = (start + end) / 2;

mergesort(a, start, q);

mergesort(a, q + 1, end);

merge(a, start, q, end);

}

}

int ksmall(int a[],int k)

{

return a[k-1];

}

void main() {

int a[MAX], n, i,k;

clrscr();

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

scanf("%d", &n);

printf("Enter the elements:\n");

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

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

}

mergesort(a, 0, n - 1);

printf("Sorted array:\n");

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

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

}

printf("Enter the value of k to find the kth smallest element");

scanf("%d",&k);

if(k>=1 && k<=n)

{

printf("\nThe %dth smallest elements is %d",k,ksmall(a,k));

}

else

printf("invalid value of k");

    getch();

}

**1/0 KNAPSACK USING DYNAMIC PROGRAMMING**

#include<stdio.h>

#include<conio.h>

void knapsack(int val[],int w[],int n,int W)

{

int i,j;

int k[20][20];

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

{

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

{

if(i=0 || j=0)

k[i][j]=0;

else if(j<w[i])

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

else

{

if(k[i-1][j]>k[i-1][j-w[i]]+val[i])

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

else

k[i][j]=[k[i-1][j-w[i]]]+val[i];

}

}

}

printf("Maximum profit is %d",k[n][W]);

}

void main()

{

int i,n,W;

int val[20],w[20];

clrscr();

printf("Enter the number of objects");

scanf("%d",&n);

printf("Enter the Maximum knapsack capacity");

scanf("%d",&W);

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

{

printf("Enter object %d - weight",i+1);

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

printf("Enter object %d - value",i+1);

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

}

knapsack(val,w,n,W);

}

**FRACTIONAL KNAPSACK USING GREEDY METHOD**

#include<stdio.h>

#include<conio.h>

void knapsack(float w[],float val[],int n,float W)

{

float x[100];

int i;

float tp=0;

float u=W;

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

{

x[i]=0;

}

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

{

if(w[i]>u)

break;

else

{

x[i]=1.0;

tp=tp+val[i];

u=u-w[i];

}

}

if(i<n)

{

x[i]=u/w[i];

tp=tp+(x[i]\*val[i]);

}

printf("\nresult vector of object used ");

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

{

printf("%.2f ",x[i]);

}

printf("\nmaximum profit %.2f ",tp);

}

void main()

{

float w[100],val[100],ratio[100],W;

int i,j,n;

float temp;

clrscr();

printf("\nenter the number of objects ");

scanf("%d",&n);

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

{

printf("Item %d-Weight ",i+1);

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

printf("Item %d-Profit ",i+1);

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

}

printf("\nenter the maximum knapsack weight ");

scanf("%f",&W);

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

{

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

}

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

{

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

{

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

{

temp=ratio[j];

ratio[j]=ratio[i];

ratio[i]=temp;

temp=w[j];

w[j]=w[i];

w[i]=temp;

temp=val[j];

val[j]=val[i];

val[i]=temp;

}

}

}

knapsack(w,val,n,W);

getch();

}

**GRAPH COLORING**

#include <stdio.h>

#include <conio.h>

#define MAX\_VERTICES 10

#define bool int

#define true 1

#define false 0

int graph[MAX\_VERTICES][MAX\_VERTICES]; // Adjacency matrix

int assignedColors[MAX\_VERTICES]; // Stores assigned colors

int colorOptions[MAX\_VERTICES]; // Stores available color options

int numVertices, numColors;

void graphColoring(int vertex);

bool checkVertex(int vertex, int color);

void main() {

int i, j;

clrscr();

printf("Enter the number of vertices in the graph: ");

scanf("%d", &numVertices);

printf("Enter the adjacency matrix:\n");

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

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

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

}

}

printf("Enter the number of colors available: ");

scanf("%d", &numColors);

printf("Enter the color options (as numbers):\n");

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

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

}

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

assignedColors[i] = 0; // Initialize all to 0

}

printf("Coloring the graph...\n");

graphColoring(0);

getch();

}

void graphColoring(int vertex) {

int colorIndex;

if (vertex == numVertices) {

int i;

printf("Solution found:\n");

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

printf("Vertex %d --> Color %d\n", i, assignedColors[i]);

}

printf("\n");

return;

}

for (colorIndex = 0; colorIndex < numColors; colorIndex++) {

if (checkVertex(vertex, colorOptions[colorIndex])) {

assignedColors[vertex] = colorOptions[colorIndex];

graphColoring(vertex + 1);

assignedColors[vertex] = 0; // Backtrack

}

}

}

bool checkVertex(int vertex, int color) {

int i;

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

if (graph[vertex][i] && assignedColors[i] == color) {

return false;

}

}

  return true;

}

**N QUEEN**

#include <stdio.h>

#include <conio.h>

#include <math.h>

#include <stdlib.h>

int x[100]; // Array to store queen positions

void printBoard(int n) {

int i, j;

printf("\nSolution:\n");

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

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

if (x[i] == j)

printf(" Q "); // Place queen

else

printf(" . "); // Empty space

}

printf("\n");

}

}

int place(int k, int i) {

int j;

for (j = 1; j < k; j++) {

if (x[j] == i || abs(x[j] - i) == abs(j - k)) // Check column and diagonals

return 0;

}

return 1;

}

void NQueen(int k, int n) {

int i;

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

if (place(k, i)) {

x[k] = i;

if (k == n)

printBoard(n);

else

NQueen(k + 1, n);

}

}

}

void main() {

int n;

clrscr(); // Clear the screen

printf("Enter the value of N: ");

scanf("%d", &n);

if (n < 1) {

printf("N should be greater than or equal to 1.\n");

} else {

NQueen(1, n); // Start placing queens from row 1

}

getch(); // Wait for key press before closing

}

**SINGLE SOURCE SHORTEST PATH USING GREEDY METHOD (DIJIKSTRA ALGO)**

#include <stdio.h>

#include <conio.h>

#define MAX 10

#define INF 9999

int minDistance(int dist[], int visited[], int n) {

int i, min = INF, min\_index = -1;

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

if (!visited[i] && dist[i] <= min) {

min = dist[i];

min\_index = i;

}

}

return min\_index;

}

void dijkstra(int graph[MAX][MAX], int src, int n) {

int dist[MAX], visited[MAX];

int i, v, count;

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

dist[i] = INF;

visited[i] = 0;

}

dist[src] = 0;

for (count = 0; count < n - 1; count++) {

int u = minDistance(dist, visited, n);

visited[u] = 1;

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

if (!visited[v] && graph[u][v] && dist[u] + graph[u][v] < dist[v]) {

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

}

}

}

printf("\nVertex\tDistance from Source %d\n", src);

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

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

}

}

void main() {

int graph[MAX][MAX], n, i, j, src;

clrscr();

printf("Enter number of vertices (max %d): ", MAX);

scanf("%d", &n);

printf("Enter adjacency matrix:\n");

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

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

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

}

}

printf("Enter source vertex (0 to %d): ", n - 1);

scanf("%d", &src);

dijkstra(graph, src, n);

    getch();

}

**ALL PAIR SHORTEST PATH (FLOYD WARSHALL ALGO)**

#include <stdio.h>

#include <conio.h>

#define MAX 10

#define INF 100

void floydWarshall(int graph[MAX][MAX], int n) {

int i, j, k;

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

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

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

if (graph[i][j] > graph[i][k] + graph[k][j]) {

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

}

}

}

}

}

void printGraph(int graph[MAX][MAX], int n) {

int i, j;

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

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

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

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

else

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

}

printf("\n");

}

}

void main() {

int graph[MAX][MAX];

int n, i, j;

clrscr();

printf("Enter number of vertices (max %d): ", MAX);

scanf("%d", &n);

printf("Enter the adjacency matrix (use 100 for INF):\n");

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

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

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

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

}

}

printf("\nOriginal Graph:\n");

printGraph(graph, n);

floydWarshall(graph, n);

printf("\nShortest Path Matrix:\n");

printGraph(graph, n);

    getch();

}