Dijkstra's Algorithm

```c
#include <stdio.h>
int minDistance(int dist[10], int included[],int n)
{
    int v,min =999, min_index;
    for (v = 0; v < n; v++)
        if (included[v] == 0 && dist[v] <= min)
            min = dist[v], min_index = v;

    return min_index;
}
void printSolution(int dist[10], int n)
{ int i;
    printf("Vertex Distance from Source\n");
    for (i = 0; i < n; i++)
        printf("\t%d \t\t\t\t %d\n", i, dist[i]);
}
void dijkstra(int graph[10][10], int src, int n)
{
    int dist[n],i,count,v;
    int included[n];
    for (i = 0; i < n; i++)
        dist[i] = 999, included[i] = 0;
    dist[src] = 0;
    for (count = 0; count < n - 1; count++) {
        int u = minDistance(dist, included,n);
        included[u] =1;
        for (v = 0; v < n; v++)
            if (included[v]==0 && graph[u][v] && dist[u] != 999
                && dist[u] + graph[u][v] < dist[v])
                dist[v] = dist[u] + graph[u][v];
    }
    printSolution(dist, n);
}
void printmatrix(int a[10][10],int n){
 int i,j;
 for (i = 0; i < n; i++)
 {
  for (j = 0; j < n; j++)
  {
   printf("%d\t", a[i][j]);
  }
  printf("\n");
 }
}
int main()
{
 int a[10][10], i, j, n,src;
 printf("Enter the number of vertices: ");
 scanf("%d", &n);
 printf("Enter the adjacency matrix:\n");
 for (i = 0; i < n; i++)
 {
   for (j = 0; j < n; j++)
   {
    scanf("%d", &a[i][j]);
   }
  }
  printf("Entered adjacency matrix is:\n");
  printmatrix(a,n);
  printf("Enter the source vertex:(Any vertex 0 to n-1)");
  scanf("%d", &src);
  dijkstra(a,src,n);
  return 0;
 }
```

Topological Sort using DFS

```
#include <stdio.h>
#include <stdlib.h>
#define MAX_VERTEX 100
int adj[MAX_VERTEX][MAX_VERTEX];
int stack[MAX_VERTEX];
int visited[MAX_VERTEX];
int top=-1;
void dfs(int v)
{ int i;
 visited[v] = 1;
 for (i = 0; i < MAX_VERTEX; i++)
 {
  if (adj[v][i] && !visited[i])
  {
   dfs(i);
  }
 }
 stack[++top] = v;
}
void topologicalSort(int V)
{ int i;
 for (i = 0; i < V; i++)
 {
  if (!visited[i])
  {
   dfs(i);
  }
 }
 printf("Topological Sort Order: \n");
 while (top != -1)
 {
  printf("%d ", stack[top--]);
 }
}
void main()
{
 int n,i,j;
 printf("Enter the number of vertices: ");
 scanf("%d", &n);
 printf("Enter the adjacency matrix: \n");
 for (i = 0; i < n; i++)
  for(j=0;j<n;j++)
   scanf("%d", &adj[i][j]);
 topologicalSort(n);
}
```

0/1 Knapsack Problem using DP

```
#include <stdio.h>
void displayinfo(int m, int n, int w[], int p[]);
void knapsack(int m, int n, int w[], int p[], int v[][10]);
void optimal(int m, int n, int w[], int v[][10]);
int max(int i, int j);
int main()
{
 int v[10][10], w[10], p[10], i, j;
 printf("************* KNAPSACK PROBLEM ***********\n");
 printf("Enter the total number of items: ");
 int n; scanf("%d", &n);
 printf("Enter the weight of each item: \n");
 for (i = 1; i <= n; i++)
  scanf("%d", &w[i]);
 printf("Enter the profit of each item: \n");
 for (i = 1; i <= n; i++)
  scanf("%d", &p[i]);
 printf("Enter the knapsack capacity: ");
 int m;
 scanf("%d", &m);
 displayinfo(m, n, w, p);
 knapsack(m, n, w, p, v);
 printf("The contents of the knapsack table are:\n");
 for (i = 0; i <= n; i++)
 {
  for (j = 0; j <= m; j++)
  {
   printf("%d ", v[i][j]);
  }
  printf("\n");
 }
 optimal(m, n, w, v);
}
void displayinfo(int m, int n, int w[], int p[])
{
 printf("Entered information about knapsack problem are:\n");
 printf("ITEM\tWEIGHT\tPROFIT\n");
 for (int i = 1; i <= n; i++)
  printf("%d\t%d\t%d\n", i, w[i], p[i]);
 printf("Capacity = %d\n", m);
}
void knapsack(int m, int n, int w[], int p[], int v[][10])
{
 int i,j;
 for (i = 0; i <= n; i++)
 {
  for (j = 0; j <= m; j++)
  {
   if (i == 0 || j == 0)
    v[i][j] = 0;
   else if (j < w[i])
    v[i][j] = v[i-1][j];
   else
    v[i][j] = max(v[i-1][j], v[i-1][j-w[i]] + p[i]);
  }
 }
}
int max(int i, int j)
{
 if (i > j)
  return i;
 else
  return j;
}
void optimal(int m, int n, int w[], int v[][10])
{
 int i = n, j = m, item = 0, x[10] = {0};
 printf("Optimal solution is: %d\n", v[n][m]);
 printf("Selected items are: ");
 while (i != 0 && j != 0)
 {
  if (v[i][j] != v[i-1][j])
  {
   x[i] = 1;
   j = j - w[i];
  }
  i = i - 1;
 }
 for (i = 1; i <= n; i++)
 {
  if (x[i] == 1)
  {
   printf("%d ", i); item = 1;
  }
 }
 if (item == 0)
 printf("NIL\t Sorry! No item can be placed in Knapsack\n");
 printf("\n********* ********************** *************\n");
}
```

Fractional Knapsack Problem using Greedy

```
#include<stdio.h>
int main()
{
 float weight[50],profit[50],ratio[50],Totalvalue,temp,capacity,amount;
 int i,j,num;
 printf("Enter number of items :");
 scanf("%d",&num);
 for (i = 0; i < num; i++)
 {
  printf("\n\nEnter Weight and Profit for item[%d] :\n",i);
  scanf("%f %f", &weight[i], &profit[i]);
 }
 printf("\n\nEnter capacity of knapsack :\n"); scanf("%f",&capacity);
 for(i=0;i<num;i++)
 {
  ratio[i]=profit[i]/weight[i];
 }
 for (i = 0; i < num; i++)
 {
  for (j = i + 1; j < num; j++)
  {
   if (ratio[i] < ratio[j])
   {
    temp = ratio[j]; ratio[j] = ratio[i]; ratio[i] = temp;
    temp = weight[j]; weight[j] = weight[i]; weight[i] = temp;
    temp = profit[j]; profit[j] = profit[i]; profit[i] = temp;
   }
  }
 }
 printf("\nKnapsack Problem using Greedy Method :\n"); for (i = 0; i < num; i++)
 {
  if (weight[i] > capacity)
  break;
  else
  {
   Totalvalue = Totalvalue + profit[i];
   capacity = capacity - weight[i];
  }
 }

 if (i < num)
 {
  Totalvalue = Totalvalue + (ratio[i]*capacity);
 }
 printf("\nThe maximum value is :%f\n",Totalvalue);
 return 0;
}
```

N-Queens Problem using Backtracking

```c
#include <stdio.h>
#include <stdlib.h>

int count = 0, x[10];

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))
            return 0;
    }
    return 1;
}

void nqueen(int k, int n) {
    int i, j, p;
    for (i = 1; i <= n; i++) {
        if (place(k, i)) {
            x[k] = i;
            if (k == n) {
                count++;
                printf("Solution = %d\n", count);
                for (j = 1; j <= n; j++) {
                    for (p = 1; p <= n; p++) {
                        if (x[j] == p)
                            printf("$\t");
                        else
                            printf("0\t");
                    }
                    printf("\n");
                }
                printf("\n");
            } else {
                nqueen(k + 1, n);
            }
        }
    }
}

int main() {
    int n;
    printf("Enter the number of queens: ");
    scanf("%d", &n);
    nqueen(1, n);
    if (count == 0)
        printf("\nNo solution found\n");
    else
        printf("\nNumber of solutions found: %d\n", count);
    return 0;
}
```

WARSHALL'S ALGORITHM:

```
#include <stdio.h>
void printmatrix(int a[10][10],int n){
 int i,j;
 for (i = 0; i < n; i++)
 {
  for (j = 0; j < n; j++)
  {
   printf("%d\t", a[i][j]);
  }
  printf("\n");
 }
}
void warshall(int a[10][10], int n)
{
 int k, i, j;
 for (k = 0; k < n; k++)
 {
  for (i = 0; i < n; i++)
  {
   for (j = 0; j < n; j++)
   {
    a[i][j]=a[i][j]||(a[i][k]&&a[k][j]);
   }
  }
 }
}
int main()
{
 int a[10][10], i, j, n;
 printf("Enter the number of vertices: ");
 scanf("%d", &n);
 printf("Enter the adjacency matrix:\n");
 for (i = 0; i < n; i++)
 {
  for (j = 0; j < n; j++)
  {
   scanf("%d", &a[i][j]);
  }
 }
 printf("Entered adjacency matrix is:\n");
 printmatrix(a,n);
 warshall(a, n);
 printf("Transitive closure:\n");
 printmatrix(a,n);
 return 0;
}
```

FLOYD'S ALGORITHM:

```c
#include <stdio.h>
void printmatrix(int a[10][10],int n){
 int i,j;
 for (i = 0; i < n; i++)
 {
  for (j = 0; j < n; j++)
  {
   printf("%d\t", a[i][j]);
  }
  printf("\n");
 }
}
void floyd(int a[10][10], int n)
{
 int k, i, j;
 for (k = 0; k < n; k++)
 {
  for (i = 0; i < n; i++)
  {
   for (j = 0; j < n; j++)
   {
    if(a[i][j]> a[i][k] + a[k][j])
     a[i][j] = a[i][k] + a[k][j];
   }
  }
 }
}
int main()
{
 int a[10][10], i, j, n;
 printf("Enter the number of vertices: ");
 scanf("%d", &n);
 printf("Enter the adjacency matrix:\n");
 for (i = 0; i < n; i++)
 {
  for (j = 0; j < n; j++)
  {
   scanf("%d", &a[i][j]);
  }
 }
 printf("Entered adjacency matrix is:\n");
 printmatrix(a,n);
 floyd(a, n);
 printf("All pair shortest path matrix:\n");
 printmatrix(a,n);
 return 0;
}
```