## 7. Topological Sort using DFS

#include <iostream>  
#include <vector>  
#include <stack>  
using namespace std;  
  
void dfs(int node, vector<vector<int>>& graph, vector<bool>& visited, stack<int>& Stack) {  
 visited[node] = true;  
 for (int i = 0; i < graph[node].size(); ++i) {  
 int neighbor = graph[node][i];  
 if (!visited[neighbor])  
 dfs(neighbor, graph, visited, Stack);  
 }  
 Stack.push(node);  
}  
  
void topologicalSortDFS(vector<vector<int>>& graph, int nodes) {  
 vector<bool> visited(nodes, false);  
 stack<int> Stack;  
 for (int i = 0; i < nodes; ++i)  
 if (!visited[i])  
 dfs(i, graph, visited, Stack);  
   
 cout << "Topological Sort Order (DFS-based): ";  
 while (!Stack.empty()) {  
 cout << Stack.top() << " ";  
 Stack.pop();  
 }  
 cout << endl;  
}  
  
int main() {  
 int nodes;  
 cout << "Enter number of nodes: ";  
 cin >> nodes;  
  
 vector<vector<int>> graph(nodes);  
 cout << "Enter adjacency matrix:\n";  
 for (int i = 0; i < nodes; ++i) {  
 for (int j = 0; j < nodes; ++j) {  
 int val;  
 cin >> val;  
 if (val) graph[i].push\_back(j);  
 }  
 }  
 topologicalSortDFS(graph, nodes);  
 return 0;  
}

## 8. Topological Sort using Kahn’s Algorithm

#include <iostream>  
#include <vector>  
#include <queue>  
using namespace std;  
  
void topologicalSortKahn(vector<vector<int>>& graph, int nodes) {  
 vector<int> in\_degree(nodes, 0);  
 for (int u = 0; u < nodes; ++u)  
 for (int v : graph[u])  
 in\_degree[v]++;  
  
 queue<int> q;  
 for (int i = 0; i < nodes; ++i)  
 if (in\_degree[i] == 0)  
 q.push(i);  
  
 cout << "Topological Sort Order (Kahn’s Algorithm): ";  
 while (!q.empty()) {  
 int u = q.front();  
 q.pop();  
 cout << u << " ";  
 for (int v : graph[u]) {  
 if (--in\_degree[v] == 0)  
 q.push(v);  
 }  
 }  
 cout << endl;  
}  
  
int main() {  
 int nodes;  
 cout << "Enter number of nodes: ";  
 cin >> nodes;  
  
 vector<vector<int>> graph(nodes);  
 cout << "Enter adjacency matrix:\n";  
 for (int i = 0; i < nodes; ++i) {  
 for (int j = 0; j < nodes; ++j) {  
 int val;  
 cin >> val;  
 if (val) graph[i].push\_back(j);  
 }  
 }  
 topologicalSortKahn(graph, nodes);  
 return 0;  
}

## 9. Prim’s Algorithm (Sprinkler system)

#include <iostream>  
#include <climits>  
using namespace std;  
  
#define V 6  
  
int minKey(int key[], bool mstSet[]) {  
 int min = INT\_MAX, min\_index;  
 for (int v = 0; v < V; v++)  
 if (!mstSet[v] && key[v] < min)  
 min = key[v], min\_index = v;  
 return min\_index;  
}  
  
void printMST(int parent[], int graph[V][V]) {  
 int totalWeight = 0;  
 cout << "Edges in MST:\n";  
 for (int i = 0; i < V; i++) {  
 if (parent[i] != -1) {  
 cout << char(parent[i] + 'A') << " - " << char(i + 'A')  
 << " Weight: " << graph[i][parent[i]] << "\n";  
 totalWeight += graph[i][parent[i]];  
 }  
 }  
 cout << "Total Pipe Length: " << totalWeight << " meters\n";  
}  
  
void primMST(int graph[V][V], int start) {  
 int parent[V], key[V];  
 bool mstSet[V];  
 for (int i = 0; i < V; i++)  
 key[i] = INT\_MAX, mstSet[i] = false, parent[i] = -1;  
  
 key[start] = 0;  
  
 for (int count = 0; count < V - 1; count++) {  
 int u = minKey(key, mstSet);  
 mstSet[u] = true;  
 for (int v = 0; v < V; v++)  
 if (graph[u][v] && !mstSet[v] && graph[u][v] < key[v])  
 parent[v] = u, key[v] = graph[u][v];  
 }  
 printMST(parent, graph);  
}  
  
int main() {  
 int graph[V][V] = {  
 {0, 100, 100, 100, 100, 8},  
 {100, 0, 2, 100, 100, 100},  
 {100, 2, 0, 3, 3, 2},  
 {100, 100, 3, 0, 100, 100},  
 {100, 100, 3, 100, 0, 100},  
 {8, 100, 2, 100, 100, 0}  
 };  
 primMST(graph, 5);  
 return 0;  
}

## 10. Prim’s Algorithm (Car Route – Minimum Cost)

#include <iostream>  
#include <climits>  
using namespace std;  
  
#define V 6  
  
int minKey(int key[], bool mstSet[]) {  
 int min = INT\_MAX, min\_index;  
 for (int v = 0; v < V; v++)  
 if (!mstSet[v] && key[v] < min)  
 min = key[v], min\_index = v;  
 return min\_index;  
}  
  
void printMST(int parent[], int graph[V][V]) {  
 int totalCost = 0;  
 cout << "Edges in MST:\n";  
 for (int i = 1; i < V; i++) {  
 cout << char(parent[i] + 'A') << " - " << char(i + 'A')  
 << " Weight: " << graph[i][parent[i]] << "\n";  
 totalCost += graph[i][parent[i]];  
 }  
 cout << "Total Minimum Cost: " << totalCost << "\n";  
}  
  
void primMST(int graph[V][V]) {  
 int parent[V], key[V];  
 bool mstSet[V];  
 for (int i = 0; i < V; i++)  
 key[i] = INT\_MAX, mstSet[i] = false, parent[i] = -1;  
 key[0] = 0;  
  
 for (int count = 0; count < V - 1; count++) {  
 int u = minKey(key, mstSet);  
 mstSet[u] = true;  
 for (int v = 0; v < V; v++)  
 if (graph[u][v] && !mstSet[v] && graph[u][v] < key[v])  
 parent[v] = u, key[v] = graph[u][v];  
 }  
 printMST(parent, graph);  
}  
  
int main() {  
 int graph[V][V] = {  
 {0, 4, 2, 0, 0, 0},  
 {4, 0, 1, 0, 5, 0},  
 {2, 1, 0, 3, 8, 0},  
 {0, 0, 3, 0, 0, 7},  
 {0, 5, 8, 0, 0, 9},  
 {0, 0, 0, 7, 9, 0}  
 };  
 primMST(graph);  
 return 0;  
}

## 11. Dijkstra’s Algorithm (Shortest Path from San Francisco to NY)

#include <iostream>  
#include <climits>  
using namespace std;  
  
#define V 7  
  
int minDistance(int dist[], bool visited[]) {  
 int min = INT\_MAX, min\_index = -1;  
 for (int i = 0; i < V; i++)  
 if (!visited[i] && dist[i] < min)  
 min = dist[i], min\_index = i;  
 return min\_index;  
}  
  
void dijkstra(int graph[V][V], int src) {  
 int dist[V], parent[V];  
 bool visited[V];  
 for (int i = 0; i < V; i++) {  
 dist[i] = INT\_MAX;  
 visited[i] = false;  
 parent[i] = -1;  
 }  
 dist[src] = 0;  
  
 for (int count = 0; count < V - 1; count++) {  
 int u = minDistance(dist, visited);  
 visited[u] = true;  
 for (int v = 0; v < V; v++) {  
 if (graph[u][v] && !visited[v] && dist[u] + graph[u][v] < dist[v]) {  
 dist[v] = dist[u] + graph[u][v];  
 parent[v] = u;  
 }  
 }  
 }  
  
 const char\* cityName[V] = {  
 "San Francisco", "Los Angeles", "Denver",  
 "Dallas", "Chicago", "New York", "Boston"  
 };  
  
 cout << "Shortest response times from " << cityName[src] << ":\n";  
 for (int i = 0; i < V; i++)  
 cout << " To " << cityName[i] << " : " << dist[i] << " sec\n";  
  
 cout << "\nShortest path to New York:\n";  
 int target = 5;  
 if (dist[target] == INT\_MAX) {  
 cout << " No path\n";  
 } else {  
 int path[V], idx = 0;  
 for (int v = target; v != -1; v = parent[v])  
 path[idx++] = v;  
 for (int i = idx - 1; i >= 0; i--) {  
 cout << cityName[path[i]];  
 if (i > 0) cout << " -> ";  
 }  
 cout << "\nTotal: " << dist[target] << " sec\n";  
 }  
}  
  
int main() {  
 int graph[V][V] = {  
 {0,3,4,5,0,0,0},  
 {3,0,7,5,0,0,0},  
 {4,7,0,4,6,0,0},  
 {5,5,4,0,5,6,0},  
 {0,0,6,5,0,4,3},  
 {0,0,0,6,4,0,2},  
 {0,0,0,0,3,2,0}  
 };  
 dijkstra(graph, 0);  
 return 0;  
}

## 12. 0/1 Knapsack Problem – Maximize Profit

#include <iostream>  
#include <cstring>  
using namespace std;  
  
#define N 4  
#define CAPACITY 5  
  
int max(int a, int b) {  
 return (a > b) ? a : b;  
}  
  
void knapsack(int weight[], int profit[], int n, int capacity) {  
 int dp[N+1][capacity+1];  
 memset(dp, 0, sizeof(dp));  
  
 for (int i = 1; i <= n; i++) {  
 for (int w = 1; w <= capacity; w++) {  
 if (weight[i-1] <= w)  
 dp[i][w] = max(dp[i-1][w], profit[i-1] + dp[i-1][w - weight[i-1]]);  
 else  
 dp[i][w] = dp[i-1][w];  
 }  
 }  
  
 cout << "Maximum Profit: " << dp[n][capacity] << "\nSelected items: ";  
 int w = capacity;  
 for (int i = n; i > 0 && w > 0; i--) {  
 if (dp[i][w] != dp[i-1][w]) {  
 cout << "Item" << i << " ";  
 w -= weight[i-1];  
 }  
 }  
 cout << "\n";  
}  
  
int main() {  
 int weight[N] = {2, 3, 4, 1};  
 int profit[N] = {3, 4, 5, 3};  
 knapsack(weight, profit, N, CAPACITY);  
 return 0;  
}

## 13. 0/1 Knapsack Problem – Thief Story

#include <iostream>  
using namespace std;  
  
int max(int a, int b) {  
 return (a > b) ? a : b;  
}  
  
int knapsack(int W, int wt[], int val[], int n) {  
 int dp[n+1][W+1];  
 for (int i = 0; i <= n; i++) {  
 for (int w = 0; w <= W; w++) {  
 if (i == 0 || w == 0)  
 dp[i][w] = 0;  
 else if (wt[i-1] <= w)  
 dp[i][w] = max(val[i-1] + dp[i-1][w - wt[i-1]], dp[i-1][w]);  
 else  
 dp[i][w] = dp[i-1][w];  
 }  
 }  
 return dp[n][W];  
}  
  
int main() {  
 int val[] = {10, 20, 50, 60};  
 int wt[] = {2, 3, 4, 5};  
 int W = 8;  
 int n = sizeof(val) / sizeof(val[0]);  
 cout << "Maximum value that can be stolen: " << knapsack(W, wt, val, n) << endl;  
 return 0;  
}