#include <stdio.h>

#include <stdlib.h>

#include <limits.h>

#include <stdbool.h>

#define NUM\_CITIES 100

#define INF INT\_MAX

// 定义边的结构体

typedef struct Edge {

int dest;

int weight;

struct Edge\* next;

} Edge;

// 定义图的邻接表

typedef struct Graph {

Edge\* adj[NUM\_CITIES]; // 邻接表数组

} Graph;

// 定义城市的名称

const char\* cities[NUM\_CITIES] = {"乌鲁木齐", "兰州", "西宁", "呼和浩特", "北京","天津","沈阳","长春","哈尔滨","大连","西安","郑州","徐州","拉萨","成都","武汉","上海","昆明","贵阳","株洲","南昌","福州","柳州","南宁","广州","深圳"};

// 创建边

Edge\* createEdge(int dest, int weight) {

Edge\* edge = (Edge\*)malloc(sizeof(Edge));

edge->dest = dest;

edge->weight = weight;

edge->next = NULL;

return edge;

}

// 初始化图

Graph\* createGraph() {

Graph\* graph = (Graph\*)malloc(sizeof(Graph));

for (int i = 0; i < NUM\_CITIES; i++) {

graph->adj[i] = NULL;

}

return graph;

}

// 添加边

void addEdge(Graph\* graph, int src, int dest, int weight) {

Edge\* edge = createEdge(dest, weight);

edge->next = graph->adj[src];

graph->adj[src] = edge;

}

// 释放图的内存

void freeGraph(Graph\* graph) {

for (int i = 0; i < NUM\_CITIES; i++) {

Edge\* temp = graph->adj[i];

while (temp) {

Edge\* toDelete = temp;

temp = temp->next;

free(toDelete);

}

}

free(graph);

}

// 找到尚未处理的顶点中距离最小的顶点的索引

int minDistance(int dist[], bool sptSet[], int V) {

int min = INT\_MAX, min\_index = -1;

for (int v = 0; v < V; v++) {

if (!sptSet[v] && dist[v] <= min) {

min = dist[v];

min\_index = v;

}

}

return min\_index;

}

// 打印从源点到目标点的最短路径

void printPath(int parent[], int j) {

// 基例: 如果j是源节点

if (parent[j] == -1) {

return;

}

printPath(parent, parent[j]);

printf(" -> %s", cities[j]);

}

// 打印从源点到目标点的最短路径及其距离

void printSolution(int dist[], int parent[], int src, int dest) {

printf("从 %s 到 %s 的最短路径距离是 %d\n", cities[src], cities[dest], dist[dest]);

printf("路径: %s", cities[src]);

printPath(parent, dest);

printf("\n");

}

// Dijkstra算法实现

void dijkstra(Graph\* graph, int src, int dest) {

int dist[NUM\_CITIES]; // 从源点到每个顶点的最短距离

bool sptSet[NUM\_CITIES]; // 是否已经找到最短路径

int parent[NUM\_CITIES]; // 记录路径

for (int i = 0; i < NUM\_CITIES; i++) {

dist[i] = INT\_MAX;

sptSet[i] = false;

parent[i] = -1;

}

dist[src] = 0;

for (int count = 0; count < NUM\_CITIES - 1; count++) {

int u = minDistance(dist, sptSet, NUM\_CITIES);

if (u == -1) break; // 如果没有找到最小距离的顶点，跳出循环

sptSet[u] = true;

Edge\* edge = graph->adj[u];

while (edge) {

if (!sptSet[edge->dest] && dist[u] != INT\_MAX && dist[u] + edge->weight < dist[edge->dest])

{

dist[edge->dest] = dist[u] + edge->weight;

parent[edge->dest] = u;

}

edge = edge->next;

}

}

printSolution(dist, parent, src, dest);

}

int main() {

// 创建图

Graph\* graph = createGraph();

// 添加边

addEdge(graph, 0, 1, 1892);

addEdge(graph, 2, 1, 216);

addEdge(graph, 1, 3, 1145);

addEdge(graph, 3, 4, 668);

addEdge(graph, 4, 5, 137);

addEdge(graph, 5, 6, 704);

addEdge(graph, 6, 7, 305);

addEdge(graph, 7, 8, 242);

addEdge(graph, 6, 9, 397);

addEdge(graph, 1, 10, 676);

addEdge(graph, 10, 11, 511);

addEdge(graph, 4, 11, 695);

addEdge(graph, 11, 12, 349);

addEdge(graph, 5, 12, 674);

addEdge(graph, 10, 14, 842);

addEdge(graph, 13, 14, 1990);

addEdge(graph, 11, 15, 534);

addEdge(graph, 12, 16, 651);

addEdge(graph, 14, 18, 967);

addEdge(graph, 14, 17, 1100);

addEdge(graph, 15, 19, 409);

addEdge(graph, 16, 20, 825);

addEdge(graph, 17, 18, 639);

addEdge(graph, 18, 24, 902);

addEdge(graph, 19, 20, 367);

addEdge(graph, 20, 21, 622);

addEdge(graph, 18, 22, 607);

addEdge(graph, 19, 22, 672);

addEdge(graph, 19, 24, 675);

addEdge(graph, 22, 23, 255);

addEdge(graph, 24, 25, 140);

// 用户输入源点和目标点

int src, dest;

printf("输入出发地的编号(0-乌鲁木齐, 1-兰州,2-西宁,3-呼和浩特,4-北京,5-天津,6-沈阳,7-长春,8-哈尔滨,9-大连,10-西安,11-郑州,12-徐州,13-拉萨,14-成都,15-武汉,16-上海,17-昆明,18-贵阳,19-株洲,20-南昌,21-福州,22-柳州,23-南宁,24-广州,25-深圳): ");

scanf("%d", &src);

printf("输入目的地的编号(0-乌鲁木齐, 1-兰州,2-西宁,3-呼和浩特,4-北京,5-天津,6-沈阳,7-长春,8-哈尔滨,9-大连,10-西安,11-郑州,12-徐州,13-拉萨,14-成都,15-武汉,16-上海,17-昆明,18-贵阳,19-株洲,20-南昌,21-福州,22-柳州,23-南宁,24-广州,25-深圳): ");

scanf("%d", &dest);

// 调用Dijkstra算法

dijkstra(graph, src, dest);

// 释放图的内存

freeGraph(graph);

return 0;

}