**任务 2——路线规划导航系统（最短路径算法）：**

**1. 阶段目标**

掌握最短路径的基本概念，熟练掌握 Dijkstra 算法和 Floyed 算法等最短路径

算法，重点掌握 Dijkstra 算法的设计思想，能够应用 Dijkstra 算法解决现实生活

中的实际问题。

**2. 最终成果形式**

要求同学们能使用 Dijkstra 算法编写代码，实现对中国主要城市之间的交通

路线进行规划和导航，能正确找到目的城市与出发城市之间的最短路径。最终实

现结果如下图所示：

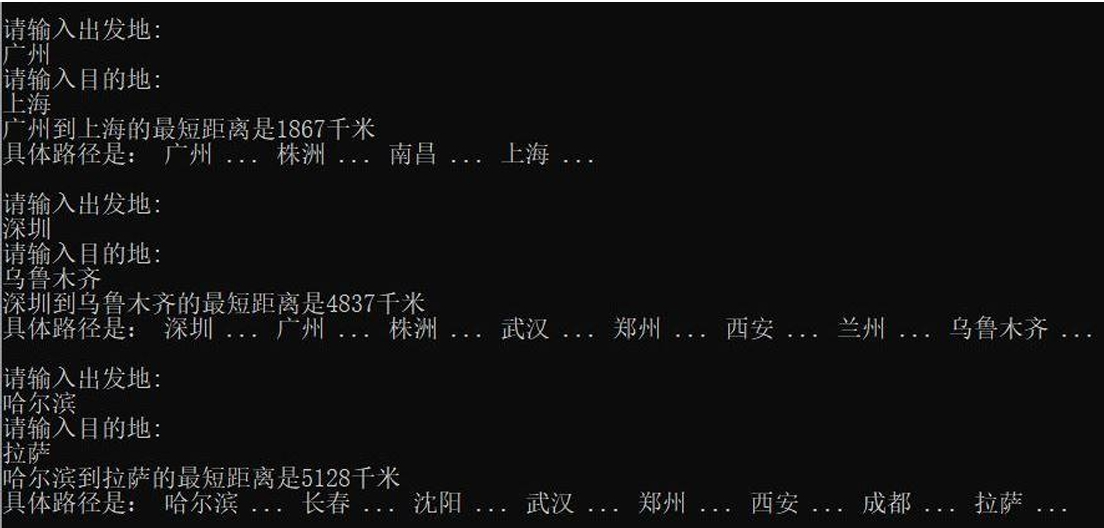


图 2 算法界面展示图

**3. 步骤与要求**

1、定义邻接矩阵。

2、设置数组 s 保存已经求得最短路径的终点序号。

3、设置数组 dist 保存源点到各个结点的当前最短距离。

4、设置一个与 dist 数组对应的 path 数组,path 数组的元素类型为指向边结点

单链表的指针，path[j]指向的链表保存着从源点 V0 到 Vj 的最短路径。

5、初始化邻接矩阵，初始化数组 s，dist 和 path。

6、按照 Dijkstra 的算法思想编写代码，求得源点到各个结点的最短路径。

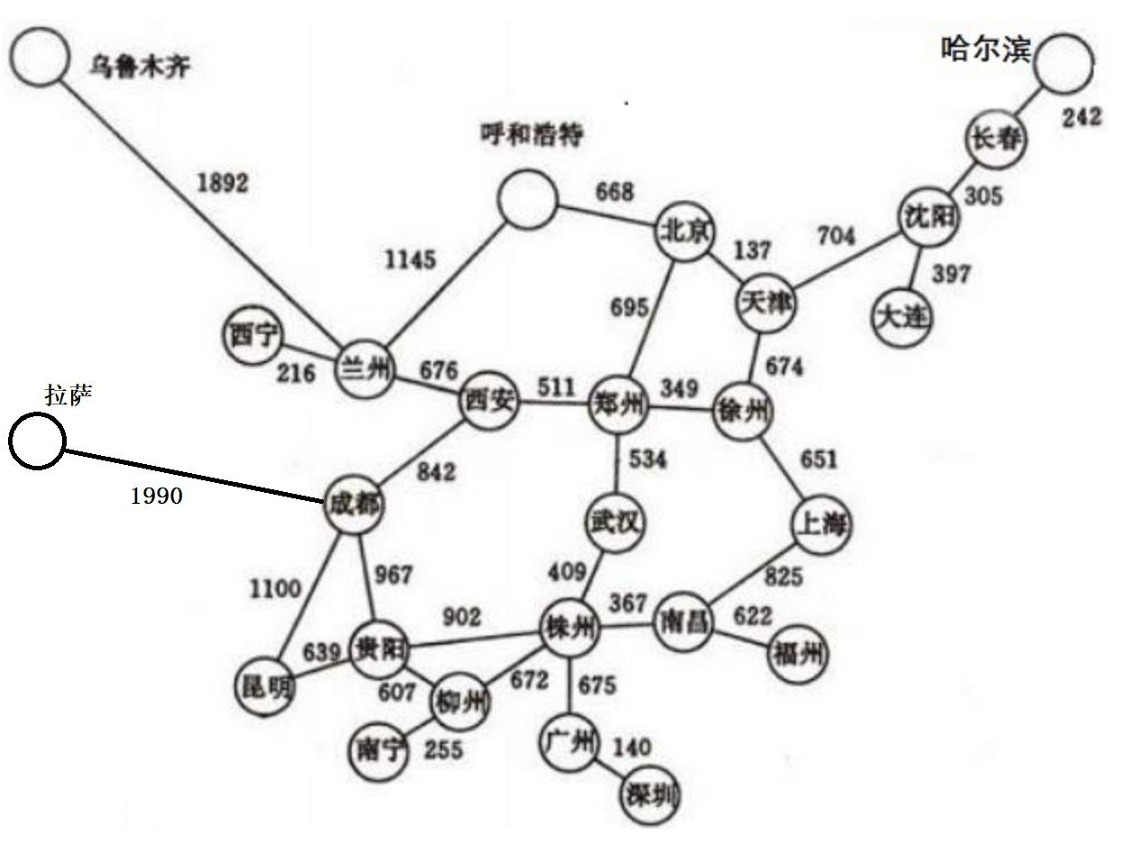
**网 络 互 联 技 术 课 程 设 计 （ 实 训 ） 指 导 书项 目 （ 实 训 ） 指 导 书**

7、编写 Path()方法，实现对源点到各个顶点最短路径的记录。

8、根据客户输入的出发地和目的地求得最短路径。

**4. 关键步骤与技术要点**

1、已知中国的主要城市交通网络如图所示：



2、根据一定的规则对各个城市进行数据化，并用邻接矩阵表示。

3、根据邻接矩阵图对数组 s、dist、path 进行初始化。

4、利用 Dijkstra 的算法思想编写最短路径计算的代码并对路径进行记录。

5、根据客户输入的出发地和目的地求得最短路径。

**附录：**

**Dijkstra 算法概述**

Dijkstra 算 法 是 由 荷 兰 计 算 机 科 学 家 Edsger W. Dijkstra 在 1956 年提出的。它是一种用于在加权图中找到单个源点到所有其他

顶点的最短路径的算法。Dijkstra 算法特别适用于处理没有负权重边的图。算

法原理。

Dijkstra 算法的核心思想是贪心策略，即在每一步选择当

前已知的最短路径。算法的步骤如下：

1. 初始化：将源点到自己的距离设为 0，其他所有顶点的距离设为无穷大（表

示尚未找到路径）。

2. 创建一个集合：用于存储已经找到最短路径的顶点。

3. 创建一个优先队列：用于存储所有顶点及其到源点的距离，优先队列按照距

离排序。

4. 循环：从优先队列中取出距离最小的顶点，如果这个顶点是目标顶点，则算

法结束。

5. 更新距离：对于取出顶点的所有邻居，如果通过这个顶点到邻居的距离小于

已知的距离，则更新邻居的距离，并将邻居加入到优先队列中。

6. 重复：重复步骤 4 和 5，直到所有顶点都被访问或者到达目标顶点。

以下是使用 C 语言实现 Dijkstra 算法的一个简单示例。这个示

例使用邻接矩阵来表示图，并假设图中没有负权重边。（此段代码用

于帮助理解 Dijkstra 算法）

#include <stdio.h>

#include <limits.h>

#define V 9 // 顶点的数量

// 找到距离集合中具有最小距离值的顶点

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

int min = INT\_MAX, min\_index;

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

if (sptSet[v] == 0 && dist[v] <= min)

min = dist[v], min\_index = v;

return min\_index;

}

// 打印构建的最短路径

void printSolution(int dist[]) {

printf("Vertex

Distance from Source\n");

for (int i = 0; i < V; i++)

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

}

// 实现 Dijkstra 算法

void dijkstra(int graph[V][V], int src) {

int dist[V]; // 输出数组，dist[i]将保存从源到 i 的最短距离

int sptSet[V]; // sptSet[i]为真，如果顶点 i 包含在最短路径树中或最短距离确定

// 初始化所有距离为无穷大，sptSet[]为假

for (int i = 0; i < V; i++)

dist[i] = INT\_MAX, sptSet[i] = 0;

// 源点到自身的距离总是 0

dist[src] = 0;

// 找到最短路径，对所有顶点

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

int u = minDistance(dist, sptSet);

sptSet[u] = 1;

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

if (!sptSet[v] && graph[u][v] && dist[u] != INT\_MAX && dist[u] + graph[u][v] <

dist[v])

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

}

printSolution(dist);- 11 -

}

int main() {

int graph[V][V] = {{0, 4, 0, 0, 0, 0, 0, 8, 0},

{4, 0, 8, 0, 0, 0, 0, 11, 0},

{0, 8, 0, 5, 0, 0, 0, 0, 2},

{0, 0, 5, 0, 4, 0, 0, 0, 0},

{0, 0, 0, 4, 0, 9, 0, 0, 0},

{0, 0, 0, 0, 9, 0, 6, 2, 0},

{0, 0, 0, 0, 0, 6, 0, 0, 1},

{8, 11, 0, 0, 0, 2, 0, 0, 7},

{0, 0, 2, 0, 0, 0, 1, 7, 0}};

dijkstra(graph, 0); // 0 代表源顶点为顶点 1

return 0;

}

以下是使用 C 语言实现的 Dijkstra 算法，处理上述图的一个简单示例：

(邻接表方法）

#include <stdio.h>

#include <limits.h>

#include <stdbool.h>

#define V 7

// 函数声明

int minDistance(int dist[], bool sptSet[]);

void printSolution(int dist[], int graph[V][V]);

// 使用 Dijkstra 算法找到从源点到所有顶点的最短路径

void dijkstra(int graph[V][V], int src) {

int dist[V]; // 从源点到每个顶点的最短距离

bool sptSet[V]; // 是否已经找到最短路径

for (int i = 0; i < V; i++)

dist[i] = INT\_MAX, sptSet[i] = false;

dist[src] = 0;

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

int u = minDistance(dist, sptSet);

sptSet[u] = true;

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

if (!sptSet[v] && graph[u][v] && dist[u] != INT\_MAX && dist[u] + graph[u][v] <

dist[v])

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

}

printSolution(dist, graph);

}

// 找到尚未处理的顶点中距离最小的顶点

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

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;

}

// 打印从源点到所有顶点的最短路径

void printSolution(int dist[], int graph[V][V]) {- 13 -

printf("Vertex \t Distance from Source\n");

for (int i = 0; i < V; i++)

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

}

int main() {

int graph[V][V] = {{0, 5, 1, 0, 0, 0, 0},

{5, 0, 0, 3, 4, 0, 0},

{1, 0, 0, 2, 0, 3, 0},

{0, 3, 2, 0, 4, 0, 2},

{0, 4, 0, 4, 0, 0, 1},

{0, 0, 3, 0, 0, 0, 0},

{0, 0, 0, 2, 1, 0, 0}};

dijkstra(graph, 0); // 0 代表源顶点

return 0;

}

针对多顶点的稀疏网，使用邻接表而不是邻接矩阵可以更有效地节省内存空

间。以下是使用 C 语言和邻接表实现 Dijkstra 算法的示例代码：

#include <stdio.h>

#include <stdlib.h>

#include <limits.h>

#define V 1000000 // 假设有 100 万个顶点

// 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[V]; // 邻接表数组

} Graph;

// 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 < V; 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 < V; i++) {

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

while (temp) {

Edge\* toDelete = temp;

temp = temp->next;

free(toDelete);

}

}

free(graph);

}

// 其余的 minDistance 和 printSolution 函数与之前相同，但需要修改以适用于邻接表。

// Dijkstra 算法实现

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

int dist[V]; // 从源点到每个顶点的最短距离

bool sptSet[V]; // 是否已经找到最短路径

for (int i = 0; i < V; i++) {- 15 -

dist[i] = INT\_MAX;

sptSet[i] = false;

}

dist[src] = 0;

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

int u = minDistance(dist, sptSet, V);

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;

}

edge = edge->next;

}

}

printSolution(dist, V);

}

// 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 == INT\_MAX) ? -1 : min\_index; // 如果所有顶点都在集合中，则返回-1

}

// 打印从源点到所有顶点的最短路径

void printSolution(int dist[], int V) {

printf("Vertex \t Distance from Source\n");

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

if (dist[i] == INT\_MAX) {

printf("%d \t -\n", i); // 表示没有路径可达

} else {

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

}

}

}

int main() {

// 创建图

Graph\* graph = createGraph();

// 添加边，这里不再赘述添加边的代码

// 调用 Dijkstra 算法

int src = 0; // 假设源点为顶点 0

dijkstra(graph, src);

// 释放图的内存

freeGraph(graph);

return 0;

}