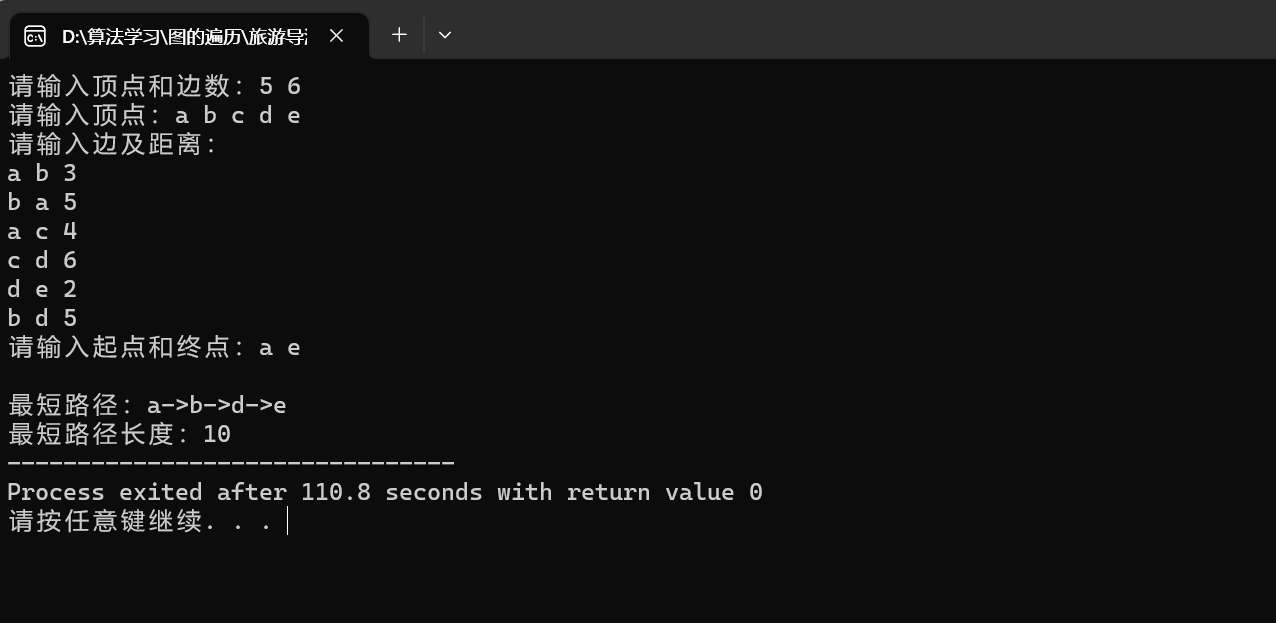
**西南石油大学实验报告**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **项目编号** | **001063000504** | **项目名称** | **图的应用** | **成绩** |  |
| **专业年级** | **计算机科学与技术** | | | **指导教师** | **胡卫东** |
| **姓名** | **王磊** | **学号** | **202231060435** | **实验日期** | **2023.06.03** |

1. **实验目的**
2. 掌握图的邻接矩阵、邻接表、十字链表等不同存储形式的表示方法。
3. 掌握图的两种不同遍历方法的基本思想并能编程实现。
4. 掌握求最小生成树的的两种算法，即Prim算法和Kruscal算法的思想，并能用C/C++语言编程实现。
5. 掌握求最短路径的两种不同情形的算法，即单源最短路径和任何一对顶点之间的最短路径，并能用C/C++语言编程实现。
6. 掌握求关键路径的算法，并能编程实现。
7. 能够灵活运用图的相关算法解决相应的实际问题。
8. **实验工具**

PC微机，Windows，DOS，Turbo C或Visual C++

1. **实验步骤**
2. 先定义结构体储存顶点和边的信息
3. 定义最短路径数组及其最短距离数组
4. 用create函数创建图和构造邻接矩阵
5. 用locate函数定位顶点所对应数组下标
6. 用**ShortPath\_Floyd函数实现Floyd算法**
7. main函数定义，调用函数，输出最短路径及长度
8. **实验结果**



1. **实验总结**

Floyd算法是一种用于求解任意两个顶点间最短路径的经典算法。在旅游景区中，可以将景点之间的步行时间表示为带权邻接矩阵，然后利用Floyd算法求解任意两个景点之间的最短步行时间。

通过实现Floyd算法，我们可以实现以下功能：

输入顶点和边的信息，构建旅游景区的有向网，并使用邻接矩阵表示景点间的步行时间。

利用Floyd算法，计算出任意两个景点之间的最短步行时间。

根据用户输入的起点和终点，输出最短路径和最短步行时间。

通过实验，我们可以得出以下结论：

Floyd算法适用于解决有向网中任意两个顶点间的最短路径问题。

通过合理设计数据结构，如顶点数组和邻接矩阵，能够方便地表示和处理景点的信息和关系。

使用Floyd算法可以高效地计算任意两个景点之间的最短步行时间，避免了遍历所有可能路径的复杂性。

输入的数据必须符合预期格式，包括顶点和边的数量，顶点的名称以及边的起点、终点和步行时间。

**旅游导游系统问题**

**（1）问题描述**

假设一个旅游景区由n个不同景点组成（有向网），并用带权邻接矩阵表示，权值表示两个景点间的步行时间，试编写程序求任意两个景点间的最短步行时间。

**（2）基本要求**

实际上是求有向图中任意两顶点间的最短路径问题。利用Floyed算法编写函数实现求图的任意两点间的最短路径。

1. **算法描述**

**创建一个结构体MGraph，包含顶点数组Vexs和邻接矩阵Arcs，以及顶点数vexnum和边数arcnum。**

**定义全局常量MVNum表示最大顶点数和GRAPH\_INFINITY表示无穷大。**

**创建函数create用于输入顶点和边信息，并构建邻接矩阵。**

**创建函数locate用于在顶点数组中定位指定顶点的索引。**

**创建函数ShortPath\_Floyd实现Floyd算法，求解任意两个景点之间的最短步行时间。在函数中：**

**初始化最短路径矩阵D和路径矩阵P，将它们初始化为邻接矩阵中的对应元素。**

**使用三层循环遍历所有顶点对(v, w)和中间顶点k：**

**如果经过中间顶点k的路径比直接路径更短，更新最短路径矩阵D和路径矩阵P。**

**最终，D中存储了任意两个景点之间的最短步行时间，P中存储了最短路径上的顶点索引。**

**在main函数中：**

**调用create函数创建图，并输入顶点和边信息。**

**调用ShortPath\_Floyd函数，计算任意两个景点之间的最短步行时间。**

**使用locate函数获取起点和终点的索引。**

**如果起点和终点存在且有最短路径，则打印最短路径和最短步行时间；否则，打印无法到达的消息。**

**（4）算法实现——示例程序**

**#include <stdio.h>**

**#include <stdlib.h>**

**#define MVNum 100 //最大顶点数**

**#define GRAPH\_INFINITY 65535**

**typedef int Patharc[MVNum][MVNum];**

**typedef int ShortPathTable[MVNum][MVNum];**

**typedef struct Graph {**

**char Vexs[MVNum]; //顶点数组**

**int Arcs[MVNum][MVNum]; //邻接矩阵**

**int vexnum, arcnum; //顶点数和边数**

**} MGraph;**

**void create(MGraph \*G);**

**void ShortPath\_Floyd(MGraph \*G, Patharc \*P, ShortPathTable \*D); //P:最短路径 D：带权长度**

**int locate(MGraph G, char c);**

**int main() {**

**int v, w;**

**MGraph G;**

**Patharc P;**

**ShortPathTable D;**

**create(&G);**

**ShortPath\_Floyd(&G, &P, &D);**

**char c1, c2;**

**printf("请输入起点和终点：");**

**scanf(" %c %c", &c1, &c2);**

**printf("\n最短路径：");**

**v = locate(G, c1);**

**w = locate(G, c2);**

**if (v != -1 && w != -1 && D[v][w] != GRAPH\_INFINITY) {**

**printf("%c", G.Vexs[v]);**

**while (v != w) {**

**v = P[v][w];**

**printf("->%c", G.Vexs[v]);**

**}**

**printf("\n");**

**printf("最短路径长度：%d", D[locate(G, c1)][locate(G, c2)]);**

**} else {**

**printf("无法从起点到达终点。\n");**

**}**

**return 0;**

**}**

**int locate(MGraph G, char c) {**

**for (int i = 0; i < G.vexnum; i++) {**

**if (c == G.Vexs[i]) {**

**return i;**

**}**

**}**

**return -1;**

**}**

**void create(MGraph \*G) {**

**printf("请输入顶点和边数：");**

**scanf("%d %d", &G->vexnum, &G->arcnum);**

**getchar();**

**printf("请输入顶点：");**

**for (int i = 0; i < G->vexnum; i++) {**

**scanf(" %c", &G->Vexs[i]);**

**}**

**for (int i = 0; i < G->vexnum; i++) { /\* 初始化图 \*/**

**for (int j = 0; j < G->vexnum; j++) {**

**if (i == j)**

**G->Arcs[i][j] = 0;**

**else**

**G->Arcs[i][j] = GRAPH\_INFINITY;**

**}**

**}**

**printf("请输入边及距离：\n");**

**for (int j = 0; j < G->arcnum; j++) {**

**char c1, c2;**

**int x;**

**scanf(" %c %c %d", &c1, &c2, &x);**

**int v = locate(\*G, c1);**

**int w = locate(\*G, c2);**

**G->Arcs[v][w] = x;**

**}**

**}**

**void ShortPath\_Floyd(MGraph \*G, Patharc \*P, ShortPathTable \*D) {**

**int v, w, k;**

**for (v = 0; v < G->vexnum; v++) {**

**for (w = 0; w < G->vexnum; w++) {**

**(\*D)[v][w] = G->Arcs[v][w];**

**if (G->Arcs[v][w] < GRAPH\_INFINITY) {**

**(\*P)[v][w] = w; //路径**

**} else {**

**(\*P)[v][w] = -1; //无路径**

**}**

**}**

**}**

**for (k = 0; k < G->vexnum; ++k) {**

**for (v = 0; v < G->vexnum; ++v) {**

**for (w = 0; w < G->vexnum; ++w) {**

**if ((\*D)[v][k] < GRAPH\_INFINITY && (\*D)[k][w] < GRAPH\_INFINITY && (\*D)[v][w] > (\*D)[v][k] + (\*D)[k][w]) {**

**(\*D)[v][w] = (\*D)[v][k] + (\*D)[k][w];**

**(\*P)[v][w] = (\*P)[v][k];**

**}**

**}**

**}**

**}**

**}**

