《数据结构与算法分析》

实验报告

实验名称	基于 Dijkstra 算法的最短路径求解
姓名	金家豪
学号	23060842
院系	自动化（人工智能）学院
专业	人工智能
班级	23061011
实验时间	2024 年 5 月 23 日

一、 实验目的

1.掌握图的邻接矩阵表示法,掌握采用邻接矩阵表示法创建图的算法。

2.掌握求解最短路径的 Dijsktra 算法。

二、 实验设备与环境

华硕天选 2 、vscode

三、 实验内容与结果

```
#include <iostream>
#include <string>
#include <cstring>
#define MAX 32767
#define vexnum_MAX 100
using namespace std;

typedef struct{
    int vex_num, arc_num;
    int arcs[vexnum_MAX][vexnum_MAX];
    char vexs[vexnum_MAX];
} AMGraph;

int find_index(char* list, char goal){
    for (int i = 0; i < vexnum_MAX;i++){
        if(goal==list[i]){
            return i;
        }
    }
}

void create_AMGraph(AMGraph &G){
    cin >> G.vex_num >> G.arc_num;

    for (int i = 0; i < G.vex_num;i++){
        std::cin >> G.vexs[i];
    }
```

```
    for (int i = 0; i < G.vex_num; i++)
    {
        for (int j = 0; j < G.vex_num; j++)
        {
            G.arcs[i][j] = MAX;
        }
    }
    for (int k = 0; k < G.arc_num;k++){
        char v1, v2;
        int weight;
        int i = 0, j = 0;
        cin >> v1 >> v2 >> weight;
        i = find_index(G.vexs, v1);
        j = find_index(G.vexs, v2);
        G.arcs[i][j] = weight;
    }
}

void show_short_path(AMGraph &G, char *path, char goal, char v0)
{
    char list[vexnum_MAX] = {goal};
    char temp;
    int count = 0;
    if (goal == v0)
        std::cout << 0;
    for (int i = 1; i < vexnum_MAX; i++)
    {
        if (goal != v0)
        {
            list[i] = path[find_index(G.vexs, goal)];
            goal = list[i];
            count++;
        }
        else
        {
            break;
        }
    }

    while (count >= 0)
    {
        std::cout << list[count]<<' ';
        count--;
    }
```

```
72.     std::cout <<std::endl;
73. }
74. 
75. void ShortPath_DIJ(AMGraph &G, char v0,char vk){
76.     int n = 0;
77.     n = G.vex_num;
78. 
79.     //记录起点到 vi 是否已经确定最短路径
80.     bool S[n];
81.     //记录起点到终点 vi 最短路径上的直接前驱节点
82.     char path[n];
83.     //记录起点到终点 vi 最短路径的权重
84.     int D[n];
85.     //初始化
86.     for (int i = 0; i < n;i++){
87.         S[i] = false;
88.         D[i] = G.arcs[find_index(G.vexs, v0)][i];
89.         if(D[i]<MAX){
90.             path[i] = v0;
91.         }
92.         else{
93.             path[i] ='e';
94.         }
95.     }
96.     S[find_index(G.vexs, v0)] = true;
97.     D[find_index(G.vexs, v0)] = 0;
98. 
99.     for (int i = 1; i < n;i++){
100.         int min = MAX;
101.         char v;
102.         for (int j = 0; j < n; j++)
103.         {
104.             if(!S[j] && D[j]<min){
105.                 v = G.vexs[j];
106.                 min=D[j];
107.             }
108.         }
109.         S[find_index(G.vexs, v)] = true;
110. 
111.         for (int j = 0; j < n;j++){
112.             if(!S[j] && (D[find_index(G.vexs,v)]+G.arcs[find_index(G.v
    exs,v),v][j]<D[j])){
113.                 D[j] = D[find_index(G.vexs, v)] + G.arcs[find_index(G.
    vexs, v), v][j];
```

```
                path[j] = v;
            }
        }
    }

    std::cout << D[find_index(G.vexs, vk)] <<  std::endl;
    show_short_path(G, path, vk, v0);
}


int  main(){
    AMGraph G;
    create_AMGraph(G);
    char v0, vk;
    std::cin >> v0 >> vk;
    ShortPath_DIJ(G, v0, vk);
}
```

四、实验总结及体会

基于 Dijkstra 算法的最短路径求解，主要内容是对邻接矩阵的应用和 Dijktra 算法的实现。

这个项目第一步在于创建邻接矩阵保存各个节点之间是否有连接以及有连接的话权重的大小。定义 AMGraph 这个结构体，其中包含边的数量，节点的数量，邻接矩阵和节点存放数组。邻接矩阵中若两个节点没有边相连，权重设为无穷大。

第二步在于实现 Dijkstra 算法。初始化 S[]、D[]、path[]三个数组，分别储存已经确定最短路径的节点、起点到 vk 的最短路径权重、起点到 vk 最短路径下，vk 的直接前驱节点。

初始化 S，v0 为 true。初始化 D[v0]为本身 v0。之后开始循环，依次找到不在 S 数组中、与 S 中节点有边相连的节点；之后比较通过这个节点到别的节点是否比 D[j]本身的 weight 小，小则进行替换，同时更新 path 数组。不断重复这个过程，知道选出最小路径。

输出时，只要知道终点和起点，从终点开始，不断读取 path 中的前驱节点，直到读到起点为止，完成最短路径的输出。

做这个小项目，对于我最大的收获在于，实现比较具有现实意义的且经典的最短路径求解算法。与此同时，我也在这个过程中，巩固了 c++语言的语法。总体来说效果还是比较好的。

这极大激发了我对于学习更便捷的算法和更多数据结构的兴趣，促进了我的学习热情，好好学习，进一步提高自己。