

哈尔滨工业大学计算学部

实验报告

课程名称：数据结构与算法

课程类型：专业基础（必修）

实验项目：图型结构及其应用

实验题目：最短路径算法

实验日期：2022 年 11 月 3 日

班级：2103601

学号：2021112845

姓名：张智雄

设计成绩	报告成绩	指导老师
		李秀坤

一、实验目的

最短路径问题研究的主要有：单源最短路径问题和所有顶点对之间的最短路径问题。在计算机领域和实际工程中具有广泛的应用，如集成电路设计、GPS/游戏地图导航、智能交通、路由选择、铺设管线等。本实验要求设计和实现 Dijkstra 算法和 Floyd-Warshall 算法，求解最短路径问题。

二、实验要求及实验环境

实验要求：

1. 实现单源最短路径的 Dijkstra 算法，输出源点及其到其他顶点的最短路径长度和最短路径。
2. 实现全局最短路径的 Floyd-Warshall 算法。计算任意两个顶点间的最短距离矩阵和最短路径矩阵，并输出任意两个顶点间的最短路径长度和最短路径。
3. 用 Dijkstra 或 Floyd-Warshall 算法解决单目标最短路径问题：找出图中每个顶点 v 到某个指定顶点 c 最短路径。
4. 利用 Dijkstra 或 Floyd-Warshall 算法解决单顶点对间最短路径问题：对于某对顶点 u 和 v ，找出 u 到 v 和 v 到 u 的一条最短路径。
5. 以文件形式输入图的顶点和边，并以适当的方式展示相应的结果。要求顶点不少于 10 个，边不少于 13 个。

实验环境：

Windows11 操作系统+VS Code 编译器

三、设计思想

数据结构：

本实验对图的相关信息采用邻接矩阵的数据结构进行存储，对最短路径以及最短路径长度均采用顺序结构（逻辑与物理结构相统一，即数组的格式进行储存），具体格式以及功能如下：

序号	名称	功能
1	matrix[N][N]	有向图的邻接矩阵
2	Distance[N]	Dijkstra算法最短路径长度
3	Path[N]	Dijkstra算法最短路径
4	A[N][N]	Floyd算法最短路径长度
5	P[N][N]	Floyd算法最短路径

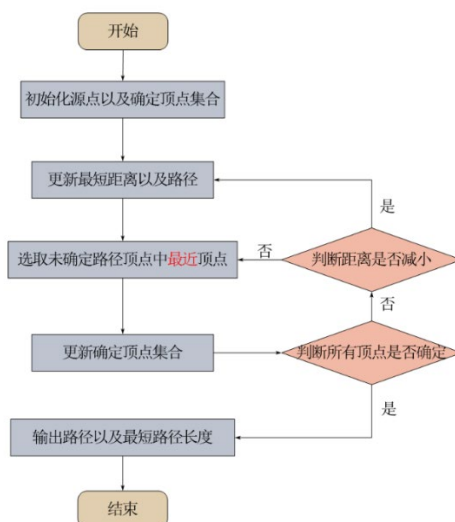
🎨 函数以及其主要功能：

序号	函数名	函数功能	函数参数	函数返回值
1	Dijkstra()	Dijkstra最短路径算法	源点i, 最短距离Distance[N], 最短路径Path[N], 邻接矩阵matrix[][N]	无
2	Floyd()	Floyd最短路径算法	最短距离A[][N], 最短路径P[][N], 邻接矩阵matrix[][N]	无
3	Dijkstra_show_path()	展示Dijkstra算法最短路径	起点u, 终点v, 最短距离Distance[], 最短路径Path[]	无
4	Floyd_show_path()	展示Floyd算法最短路径	起点i, 终点c, 最短距离A[][N], 最短路径P[][N]	无

🎨 核心算法的主要步骤：

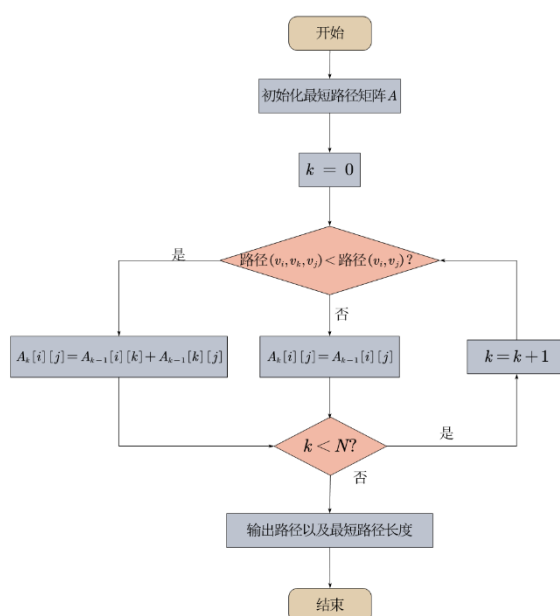
1. Dijkstra 最短路径算法

Dijkstra 最短路径算法主要流程图如下：



2. Floyd 最短路径算法

Floyd 最短路径算法主要流程图如下：



3. 最短路径的展示

在 Dijkstra 最短路径算法中，基于栈的数据结构，从终点开始依次访问路径数组上的邻接顶点并加入栈中，直至访问至源点或起点后依次输出。

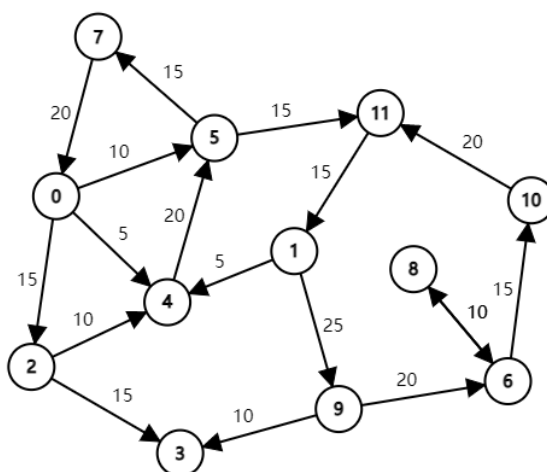
而在 Floyd 最短路径算法中，由于此算法基于动态规划的思想，每次需要考虑 $i \sim k$ 和 $k \sim j$ 两段的情况，因而采用递归的思想，依次访问前后两段，直至找到不可再分的点后输出最短路径

4. 单目标最短路径算法

将单源的 Dijkstra 算法中的参数矩阵转置即可，输出路径时不需要栈结构辅助（本身相当于已经逆置一次）

四、测试结果

测试样例如右图：



运行结果如下：

```
C:\Windows\system32\cmd.exe
图存入成功！！
=====
计算单源最短路径，指定源点为： 5
从第5个顶点到顶点0的最短路径为： 5->7->0, 最短距离为35
从第5个顶点到顶点1的最短路径为： 5->11->1, 最短距离为30
从第5个顶点到顶点2的最短路径为： 5->7->0->2, 最短距离为50
从第5个顶点到顶点3的最短路径为： 5->11->1->9->3, 最短距离为65
从第5个顶点到顶点4的最短路径为： 5->11->1->4, 最短距离为35
从第5个顶点到顶点6的最短路径为： 5->11->1->9->6, 最短距离为75
从第5个顶点到顶点7的最短路径为： 5->7, 最短距离为15
从第5个顶点到顶点8的最短路径为： 5->11->1->9->6->8, 最短距离为85
从第5个顶点到顶点9的最短路径为： 5->11->1->9, 最短距离为55
从第5个顶点到顶点10的最短路径为： 5->11->1->9->6->10, 最短距离为90
从第5个顶点到顶点11的最短路径为： 5->11, 最短距离为15
=====
最短距离矩阵如下：
0      40      15      30      5      10      85      25      95      65      100      25
60      0      75      35      5      25      45      40      55      25      60      40
65      60      0      15      10      30      105      45      115      85      120      45
inf     inf     inf     inf     inf     inf     inf     inf     inf     inf     inf     inf
55      50      70      85      0      20      95      35      105      75      110      35
35      30      50      65      35      0      75      15      85      55      90      15
110     50      125     85      55      75      0      90      10      75      15      35
20      60      35      50      25      30      105      0      115      85      120      45
120     60      135     95      65      85      10      100      0      85      25      45
130     70      145     10      75      95      20      110      30      0      35      55
95      35      110      70      40      60      80      75      90      60      0      20
75      15      90      50      20      40      60      55      70      40      75      0
选择任意两顶点p,q:2 7
从第2个顶点到顶点7的最短路径为： 2->4->5->7, 最短距离为45
=====
C:\Windows\system32\cmd.exe
=====
计算单目标最短路径，指定顶点c为： 7
从第0个顶点到顶点7的最短路径为： 0->5->7, 最短距离为25
从第1个顶点到顶点7的最短路径为： 1->4->5->7, 最短距离为40
从第2个顶点到顶点7的最短路径为： 2->4->5->7, 最短距离为45
从第3个顶点到顶点7不连通！
从第4个顶点到顶点7的最短路径为： 4->5->7, 最短距离为35
从第5个顶点到顶点7的最短路径为： 5->7, 最短距离为15
从第6个顶点到顶点7的最短路径为： 6->10->11->1->4->5->7, 最短距离为90
从第8个顶点到顶点7的最短路径为： 8->6->10->11->1->4->5->7, 最短距离为100
从第9个顶点到顶点7的最短路径为： 9->6->10->11->1->4->5->7, 最短距离为110
从第10个顶点到顶点7的最短路径为： 10->11->1->4->5->7, 最短距离为75
从第11个顶点到顶点7的最短路径为： 11->1->4->5->7, 最短距离为55
=====
计算单顶点间最短路径，顶点对u, v依次为： 3 8
从第3个顶点到顶点8不连通！
从第8个顶点到顶点3的最短路径为： 8->6->10->11->1->9->3, 最短距离为95
请按任意键继续. . .
```

五、经验体会与不足

经验体会：

通过本实验，加深了对图型结构的理解，同时对 Dijkstra 算法和 Floyd-Warshall 算法有了一定了解，对其中的贪心思想以及动态规划思想有了一定体会，同时在单源与单目标的问题的转换中，体会到了灵活的设计思维，而非简单粗暴的遍历求解。

存在不足：

1. 可以考虑计算有向图的可达矩阵，理解可达矩阵的含义。
2. 利用堆结构（优先级队列）改进和优化 Dijkstra 算法。

六、附录：源代码（带注释）

```
1. #include<iostream>
2. #include<math.h>
3. #include<algorithm>
4. #include<stack>
5. #include<fstream>
6. #define N 12
7. #define inf 1000
8. using namespace std;
9.
10. //Dijkstra 算法
11. void Dijkstra(int i, int Distance[N], int Path[N], int matrix[][N]
);
12. //Floyd 算法
13. void Floyd(int A[][N], int P[][N], int matrix[][N]);
14. //单目标最短路径输出
15. void Dijkstra_show_path_1(int u, int v, int Distance[], int Path[
]);
16. //单源最短路径输出
17. void Dijkstra_show_path(int u, int v, int Distance[], int Path[])
;
18. //Floyd 最短路径输出
19. void Floyd_show_path(int i, int c, int A[][N], int P[][N]);
20.
21. int main()
22. {
23.     int Distance[2][N], Path[2][N];
24.     int C[N][N], A[N][N], P[N][N];
25.     int m, n;
26.     //初始化
27.     for(int i = 0; i < N; i++)
28.         for(int j = 0; j < N; j++)
29.             C[i][j] = inf;
30.
31.     //文件导入边信息
32.     ifstream infile;
33.     infile.open("data.txt");
34.     while(!infile.eof())
35.         infile >> m >> n >> C[m][n];
36.     cout << "图存入成功!!" << endl;
37.
38.     Floyd(A, P, C);
39.     //Dijkstra 输出源点及其到其他顶点的最短路径长度和最短路径
```

```

40.     int a;
41.     cout<<"===== "<<endl;
42.     cout<<"计算单源最短路径, 指定源点为: ";
43.     cin>>a;
44.     Dijkstra(a, Distance[0], Path[0], C);
45.     for(int i = 0; i < N; i++)
46.         if(a != i) Dijkstra_show_path(a,i,Distance[0],Path[0]);
47.
48.     //Floyd 计算任意两个顶点间的最短距离矩阵和最短路径矩阵, 并输出任意
两个顶点间的最短路径长度和最短路径
49.     cout<<"===== "<<endl;
50.     cout<<"最短距离矩阵如下: "<<endl;
51.     for(int i = 0; i < N; i++)
52.     {
53.         for(int j = 0; j < N; j++)
54.         {
55.             if(i != j && A[i][j] < inf) cout<<A[i][j]<<"\t";
56.             else if(A[i][j] >= inf) cout<<"inf"<<"\t";
57.             else cout<<0<<"\t";
58.         }
59.         cout<<endl;
60.     }
61.     int p,q;
62.     cout<<"选择任意两顶点 p,q:";
63.     cin>>p>>q;
64.     Floyd_show_path(p,q,A,P);
65.
66.     //Dijkstra 计算单目标最短路径
67.     cout<<"===== "<<endl;
68.     int c;
69.     cout<<"计算单目标最短路径, 指定顶点 c 为: ";
70.     cin>>c;
71.     //矩阵转置
72.     int C1[N][N];
73.     for(int i = 0; i < N; i++)
74.         for(int j = 0; j < N; j++)
75.             C1[j][i] = C[i][j];
76.     Dijkstra(c, Distance[0], Path[0], C1);
77.     for(int i = 0; i < N; i++)
78.         if(c != i) Dijkstra_show_path_1(c,i,Distance[0],Path[0])
;
79.
80.     //Dijkstra 计算单顶点对间最短路径问题
81.     cout<<"===== "<<endl;

```

```

82.     int u,v;
83.     cout<<"计算单顶点到间最短路径，顶点对 u, v 依次为: ";
84.     cin>>u>>v;
85.     Dijkstra(u, Distance[0], Path[0], C);
86.     Dijkstra_show_path(u,v,Distance[0],Path[0]);
87.     Dijkstra(v, Distance[1], Path[1], C);
88.     Dijkstra_show_path(v,u,Distance[1],Path[1]);
89. }
90.
91. void Dijkstra(int i,int Distance[N], int Path[N], int matrix[][N]
)
92. {
93.     //数据结构初始化
94.     //Distance[]-源点到顶点的最短路径长度，Path[]为最短路径上最后经过
    的顶点，S[]存放源点和已生成的终点
95.     bool flag[N];
96.     for(int j = 0; j < N; j++){
97.         Distance[j] = matrix[i][j];
98.         Path[j] = i;
99.         flag[j] = false;
100.    }
101.
102.    flag[i] = true;
103.
104.    for(int k = 0; k < N - 1; k++){
105.        int min = inf;
106.        int w = i;
107.        for(int j = 0; j < N; j++){
108.            if(flag[j] == true) continue;
109.            if(Distance[j] < min) {w = j; min = Distance[j];}
110.        }
111.
112.        flag[w] = true;
113.
114.        for(int j = 0; j < N; j++){
115.            if(flag[j] == true) continue;
116.            else{
117.                if(Distance[j] >= Distance[w] + matrix[w][j])
118.                    {Distance[j] = Distance[w] + matrix[w][j]; P
ath[j] = w;}
119.            }
120.        }
121.    }
122. }

```



```

123.
124. void Floyd(int A[][N], int P[][N], int matrix[][N])
125. {
126.     //初始化
127.     for(int i = 0; i < N; i++)
128.         for(int j = 0; j < N; j++) {
129.             A[i][j] = matrix[i][j];
130.             P[i][j] = -1;
131.         }
132.     //迭代
133.     for(int k = 0; k < N; k++)
134.         for(int i = 0; i < N; i++)
135.             for(int j = 0; j < N; j++) {
136.                 if(A[i][j] > (A[i][k] + A[k][j])){
137.                     A[i][j] = A[i][k] + A[k][j];
138.                     P[i][j] = k;
139.                 }
140.             }
141. }
142. void find(int i, int c, int P[][N])
143. {
144.     if(P[i][c] == -1) cout<<"->"<<c;
145.     else{
146.         find(i, P[i][c], P);
147.         find(P[i][c], c, P);
148.     }
149. }
150. void Floyd_show_path(int i, int c, int A[][N], int P[][N])
151. {
152.     if(A[i][c] < inf){
153.         cout<<"从第"<<i<<"个顶点到顶点"<<c<<"的最短路径为: ";
154.         cout<<i;
155.         find(i,c,P);
156.         cout<<" ,最短距离为"<<A[i][c]<<endl;
157.     }
158.     else cout<<"从第"<<i<<"个顶点到顶点"<<c<<"不连通! "<<endl;
159. }
160.
161. void Dijkstra_show_path(int u, int v, int Distance[], int Path[]
)
162. {
163.     if(Distance[v] < inf){
164.         cout<<"从第"<<u<<"个顶点到顶点"<<v<<"的最短路径为: ";
165.         cout<<u<<"->";

```

```

166.         int y = v;
167.         stack<int> p;
168.         while(Path[y] != u){
169.             y = Path[y];
170.             p.push(y);
171.         }
172.         while(!p.empty())
173.             {cout<<p.top()<<"->";p.pop();}
174.         cout<<v<<" ,最短距离为"<<Distance[v]<<endl;
175.     }
176.     else cout<<"从第"<<u<<"个顶点到顶点"<<v<<"不连通! "<<endl;
177. }
178.
179. void Dijkstra_show_path_1(int u, int v, int Distance[], int Path
[[])
180. {
181.     if(Distance[v] < inf){
182.         cout<<"从第"<<v<<"个顶点到顶点"<<u<<"的最短路径为: ";
183.         cout<<v<<"->";
184.         int y = v;
185.         stack<int> p;
186.         while(Path[y] != u){
187.             y = Path[y];
188.             cout<<y<<"->";
189.         }
190.         cout<<u<<" ,最短距离为"<<Distance[v]<<endl;
191.     }
192.     else cout<<"从第"<<v<<"个顶点到顶点"<<u<<"不连通! "<<endl;
193. }

```