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** | |

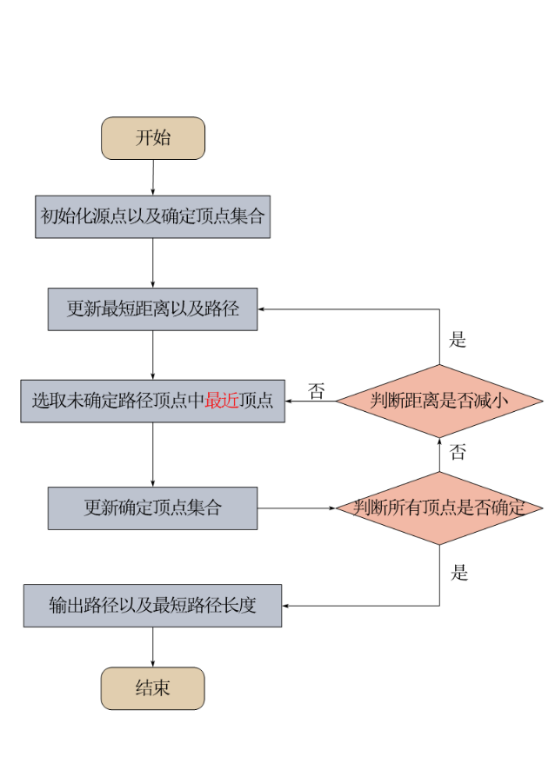
* **函数以及其主要功能：**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **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：**

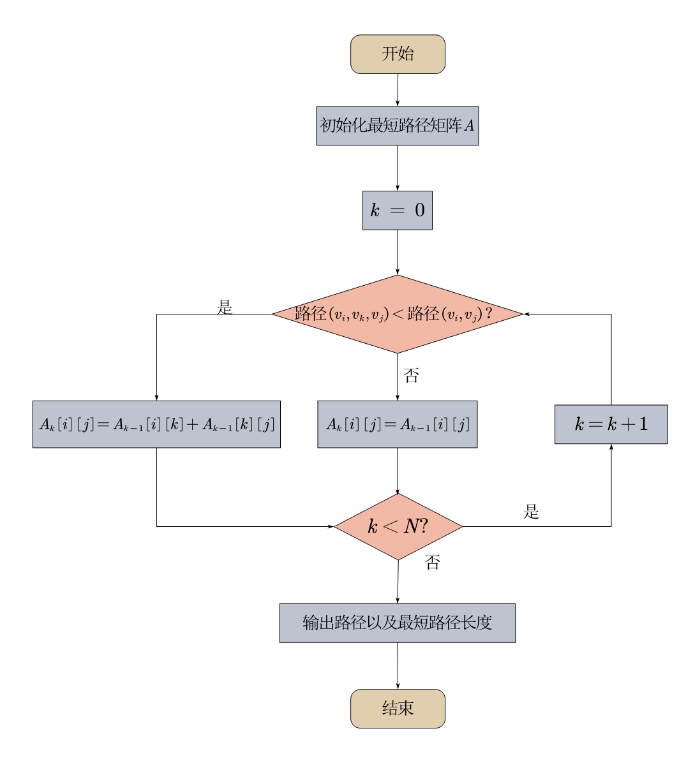
1. **Dijkstra最短路径算法**

**Dijkstra最短路径算法主要流程图如下：**



1. **Floyd最短路径算法**

**Floyd最短路径算法主要流程图如下：**

****

1. **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~k和k~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

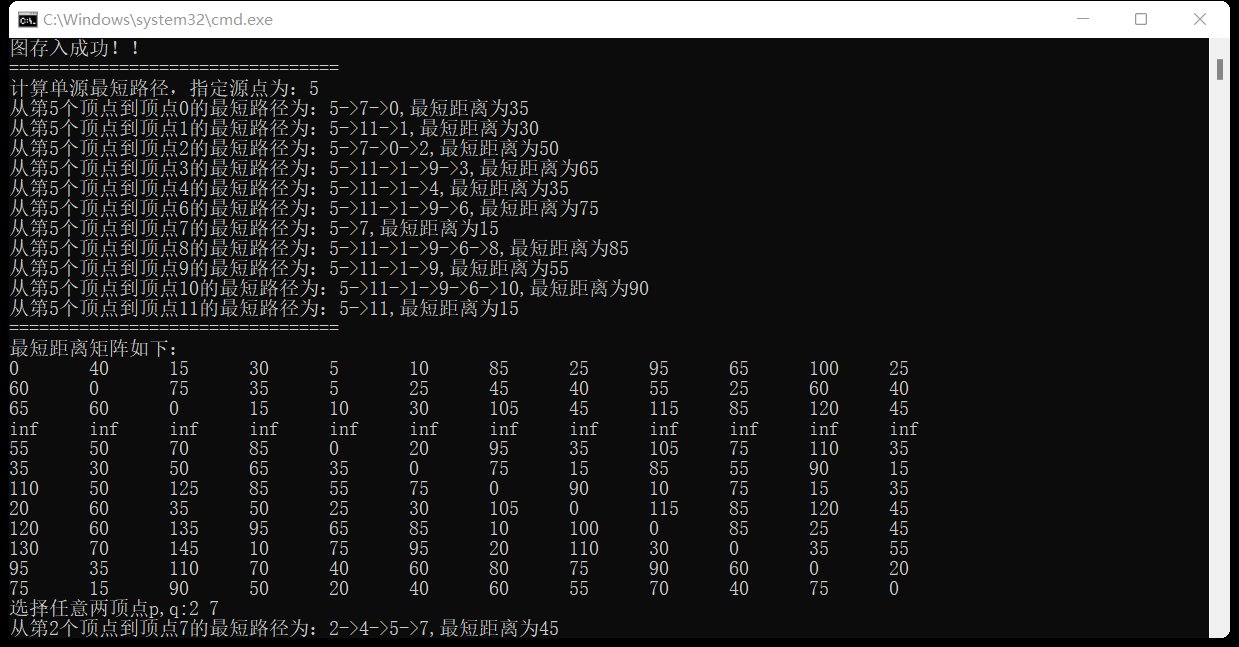
1. **单目标最短路径算法**

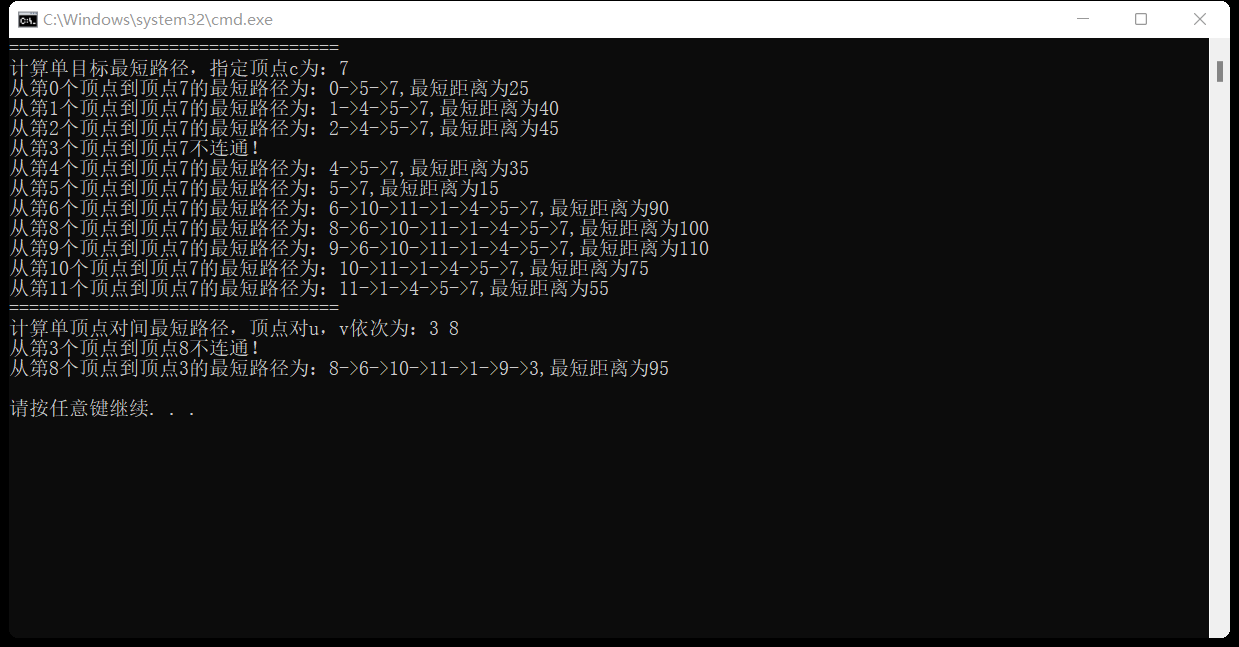
将单源的Dijkstra算法中的参数矩阵转置即可，输出路径时不需要栈结构辅助（本身相当于已经逆置一次）

**图示

描述已自动生成四、测试结果**

**测试样例如右图：**

**运行结果如下：**

****

**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. *//Dijkstra算法*
10. void Dijkstra(int i,int Distance[N], int Path[N], int matrix[][N]);
11. *//Floyd算法*
12. void Floyd(int A[][N], int P[][N], int matrix[][N]);
13. *//单目标最短路径输出*
14. void Dijkstra\_show\_path\_1(int u, int v, int Distance[], int Path[]);
15. *//单源最短路径输出*
16. void Dijkstra\_show\_path(int u, int v, int Distance[], int Path[]);
17. *//Floyd最短路径输出*
18. void Floyd\_show\_path(int i, int c, int A[][N], int P[][N]);
19. int main()
20. {
21. int Distance[2][N], Path[2][N];
22. int C[N][N],A[N][N],P[N][N];
23. int m,n;
24. *//初始化*
25. for(int i = 0; i < N; i++)
26. for(int j = 0; j < N; j++)
27. C[i][j] = inf;
28. *//文件导入边信息*
29. ifstream infile;
30. infile.open("data.txt");
31. while(!infile.eof())
32. infile>>m>>n>>C[m][n];
33. cout<<"图存入成功！！"<<endl;
34. Floyd(A, P, C);
35. *//Dijkstra输出源点及其到其他顶点的最短路径长度和最短路径*
36. int a;
37. cout<<"================================="<<endl;
38. cout<<"计算单源最短路径，指定源点为：";
39. cin>>a;
40. Dijkstra(a, Distance[0], Path[0], C);
41. for(int i = 0; i < N; i++)
42. if(a != i)  Dijkstra\_show\_path(a,i,Distance[0],Path[0]);
44. *//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*
45. cout<<"================================="<<endl;
46. cout<<"最短距离矩阵如下："<<endl;
47. for(int i = 0; i < N; i++)
48. {
49. for(int j = 0; j < N; j++)
50. {
51. if(i != j && A[i][j] < inf)  cout<<A[i][j]<<"\t";
52. else if(A[i][j] >= inf)  cout<<"inf"<<"\t";
53. else cout<<0<<"\t";
54. }
55. cout<<endl;
56. }
57. int p,q;
58. cout<<"选择任意两顶点p,q:";
59. cin>>p>>q;
60. Floyd\_show\_path(p,q,A,P);
61. *//Dijkstra计算单目标最短路径*
62. cout<<"================================="<<endl;
63. int c;
64. cout<<"计算单目标最短路径，指定顶点c为：";
65. cin>>c;
66. *//矩阵转置*
67. int C1[N][N];
68. for(int i = 0; i < N; i++)
69. for(int j = 0; j < N; j++)
70. C1[j][i] = C[i][j];
71. Dijkstra(c, Distance[0], Path[0], C1);
72. for(int i = 0; i < N; i++)
73. if(c != i)  Dijkstra\_show\_path\_1(c,i,Distance[0],Path[0]);
74. *//Dijkstra计算单顶点对间最短路径问题*
75. cout<<"================================="<<endl;
76. int u,v;
77. cout<<"计算单顶点对间最短路径，顶点对u，v依次为：";
78. cin>>u>>v;
79. Dijkstra(u, Distance[0], Path[0], C);
80. Dijkstra\_show\_path(u,v,Distance[0],Path[0]);
81. Dijkstra(v, Distance[1], Path[1], C);
82. Dijkstra\_show\_path(v,u,Distance[1],Path[1]);
83. }
84. void Dijkstra(int i,int Distance[N], int Path[N], int matrix[][N])
85. {
86. *//数据结构初始化*
87. *//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*
88. bool flag[N];
89. for(int j = 0; j < N; j++){
90. Distance[j] = matrix[i][j];
91. Path[j] = i;
92. flag[j] = false;
93. }
95. flag[i] = true;
96. for(int k = 0; k < N - 1; k++){
97. int min = inf;
98. int w = i;
99. for(int j = 0; j < N; j++){
100. if(flag[j] == true) continue;
101. if(Distance[j] < min)  {w = j; min = Distance[j];}
102. }
103. flag[w] = true;
104. for(int j = 0; j < N; j++){
105. if(flag[j] == true) continue;
106. else{
107. if(Distance[j] >= Distance[w] + matrix[w][j])
108. {Distance[j] = Distance[w] + matrix[w][j]; Path[j] = w;}
109. }
110. }
111. }
112. }
113. void Floyd(int A[][N], int P[][N], int matrix[][N])
114. {
115. *//初始化*
116. for(int i = 0; i < N; i++)
117. for(int j = 0; j < N; j++) {
118. A[i][j] = matrix[i][j];
119. P[i][j] = -1;
120. }
121. *//迭代*
122. for(int k = 0; k < N; k++)
123. for(int i = 0; i < N; i++)
124. for(int j = 0; j < N; j++) {
125. if(A[i][j] > (A[i][k] + A[k][j])){
126. A[i][j] = A[i][k] + A[k][j];
127. P[i][j] = k;
128. }
129. }
130. }
131. void find(int i, int c,int P[][N])
132. {
133. if(P[i][c] == -1) cout<<"->"<<c;
134. else{
135. find(i, P[i][c], P);
136. find(P[i][c], c, P);
137. }
138. }
139. void Floyd\_show\_path(int i, int c, int A[][N], int P[][N])
140. {
141. if(A[i][c] < inf){
142. cout<<"从第"<<i<<"个顶点到顶点"<<c<<"的最短路径为：";
143. cout<<i;
144. find(i,c,P);
145. cout<<",最短距离为"<<A[i][c]<<endl;
146. }
147. else cout<<"从第"<<i<<"个顶点到顶点"<<c<<"不连通！"<<endl;
148. }
149. void Dijkstra\_show\_path(int u, int v, int Distance[], int Path[])
150. {
151. if(Distance[v] < inf){
152. cout<<"从第"<<u<<"个顶点到顶点"<<v<<"的最短路径为：";
153. cout<<u<<"->";
154. int y = v;
155. stack<int> p;
156. while(Path[y] != u){
157. y = Path[y];
158. p.push(y);
159. }
160. while(!p.empty())
161. {cout<<p.top()<<"->";p.pop();}
162. cout<<v<<",最短距离为"<<Distance[v]<<endl;
163. }
164. else cout<<"从第"<<u<<"个顶点到顶点"<<v<<"不连通！"<<endl;
165. }
166. void Dijkstra\_show\_path\_1(int u, int v, int Distance[], int Path[])
167. {
168. if(Distance[v] < inf){
169. cout<<"从第"<<v<<"个顶点到顶点"<<u<<"的最短路径为：";
170. cout<<v<<"->";
171. int y = v;
172. stack<int> p;
173. while(Path[y] != u){
174. y = Path[y];
175. cout<<y<<"->";
176. }
177. cout<<u<<",最短距离为"<<Distance[v]<<endl;
178. }
179. else cout<<"从第"<<v<<"个顶点到顶点"<<u<<"不连通！"<<endl;
180. }