哈尔滨工业大学计算机科学与技术学院

实验报告

课程名称：数据结构

课程类型：必修

试验项目名称：图结构及其应用

实验题目：图的遍历算法

班级：1603005

学号：1160300524

姓名：冯运

1. 实验目的

通过深度优先搜索的递归和非递归和广度优先搜索算法对图进行遍历搜索，同时学会以合理的方式保存图的信息

1. 实验要求及实验环境
2. 由用户输入图的点和边的信息，建立起图的数据结构
3. 分别使用深度优先搜索的递归方式和非递归方式遍历图的各个点，输出信息
4. 以广度优先搜索的方式遍历图，输出遍历的结果

实验环境：寝室，机房，Clion集成开发环境

1. 设计思想

数据类型定义：

1. **class** map{
2. **public**:
3. **int** pointNum,edgeNum;  //点的个数 边的个数
4. **int** point[maxNum],edge[maxNum][maxNum];  //在图的节点处存的信息  在图的边上存的信息(边的权重)
5. }
6. //实例一个图的对象
7. map newMap;
8. //在dfs和bfs时需要判断节点是否已经访问过，这个数组用来存储节点是否已经访问过
9. **bool** visited[maxNum];
10. //存储搜索得到的搜索路径
11. **int** dfsCode[maxNum];

函数的定义

1. //将邻接矩阵初始化为全0
2. **void** initMap(**int** p,**int** e);

5. //为图的类添加一条无向的边
6. **void** addEdge(**int** a,**int** b,**int** weight);
8. //通过文件读入图的信息并建立图
9. **void** readFile();
11. //深度优先搜索递归方式的写法
12. **void** DFSRecursive(**int** currentPoint);
14. //深度优先索搜用到的主函数
15. **void** DFSRecursiveMain();
17. //深度优先搜索的非递归的写法
18. **void** DFSNotRecursive();

21. //广度优先搜索的函数
22. **void** BFS();

主要函数的设计思想

void DFSRecursiveMain()；

这个函数首先将visited[]初始化，dfsCode[](记录搜索路径的)也都初始化为1，然后在一重循环里开始递归调用DFSRecursive()函数，开始深度优先搜索，这样做是为了保证即使图是不连通的，也可以重新从另一个连通分量开始深度优先搜索。

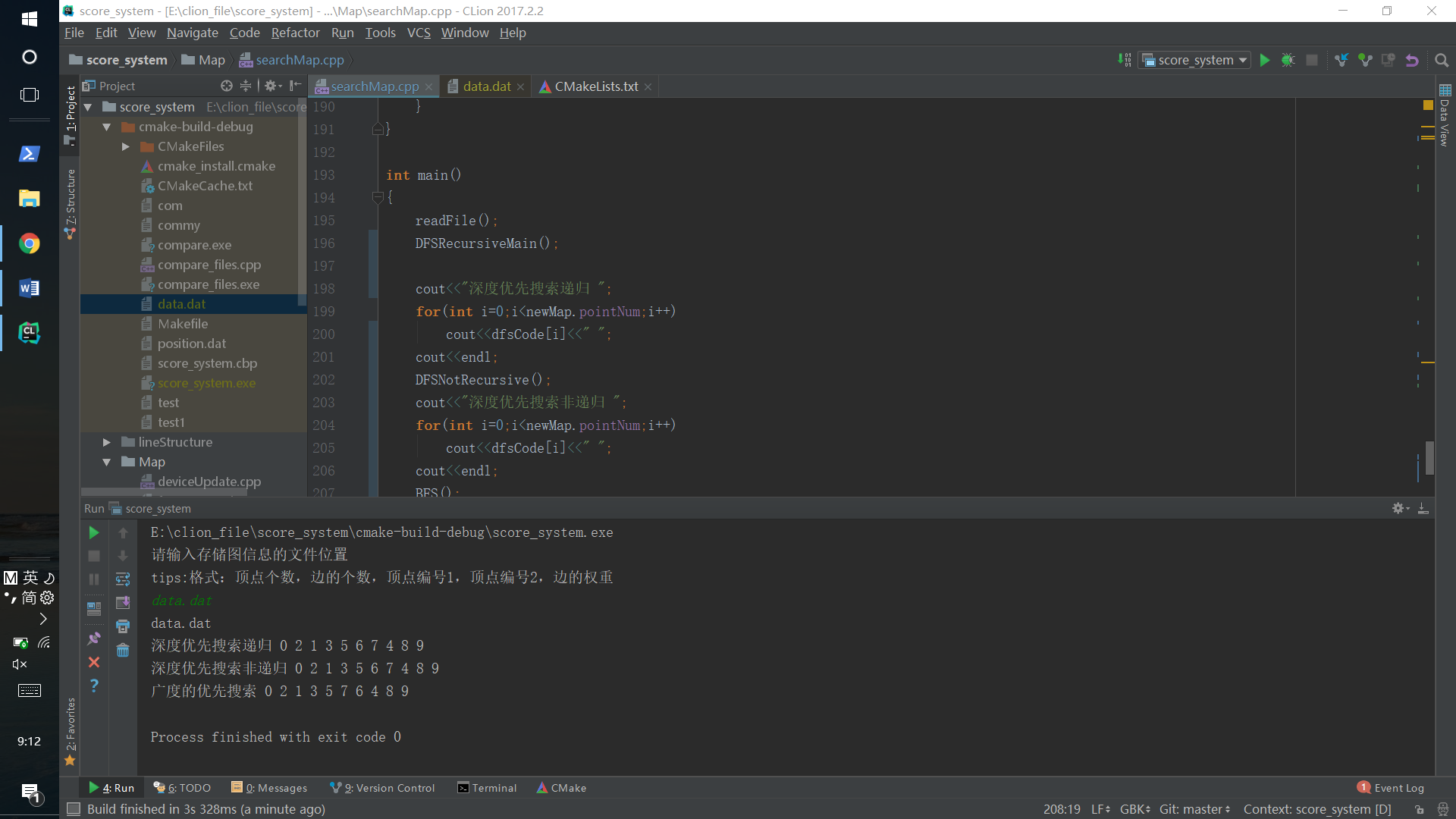
void DFSRecursive(int currentPoint)

首先将传入的参数的设置为已访问，然后循环遍历邻接矩阵中与该节点有边相连的点，如果该点没有访问过，就将该点的标号作为参数，进行下一次递归调用，如果没有这样的节点，就结束函数返回。这样就保证在搜索某条路径的终点时可以再回溯到上一个访问的节点，这样的搜索路径就是以深度为优先搜索的搜索方法。

void DFSNotRecursive()

这个非递归的函数与递归的思路基本一致，只是递归使用的是函数栈，如果不使用递归，需要自己写一个栈的数据结构，自己控制回溯的过程。

1. **for**(**int** i=0;i<newMap.pointNum;i++)
2. {
3. index=i;
4. **while**(!visited[index])    //这个循环保证所有与该节点相连的没有访问过的点的都会搜索到
5. {
6. visited[index]=**true**;
7. dfsCode[count]=index;
8. trace.push(index);
9. count++;
10. **do**
11. {
12. flag=0;
13. **for**(**int** j=0;j<newMap.pointNum;j++)
14. {
15. **if**(!visited[j]&&newMap.edge[index][j])
16. {
17. index=j;
18. flag=1;
19. **break**;
20. }
21. }
22. **if**(flag==0)  //与该节点没有相连且没有访问过的点 ，需要回溯到之前访问的节点
23. {
24. trace.pop();
25. **if**(trace.empty())
26. {
27. flag=1;
28. **break**;
29. }
30. **else**
31. {
32. index=trace.top();
33. }
34. }
35. }**while**(flag==0);    //该循环一直持续到回溯到有满足条件的新点可以访问为止
36. }
37. }
38. 运行结果



文件录入的数据如下：

10 8  
0 2 1  
1 2 1  
2 3 1  
3 5 1  
5 6 1  
3 7 1  
7 4 1  
8 9 1

1. 系统的不足与体会

在刚开始只是想要完成三种搜索方式，没有考虑如何展示搜索的结果，后来才想到需要将搜索的结果展现出来。目前的搜索结果展示不够直观，不够完善，不能让别人直观的看出是否正确，这一点的还需要对程序进行完善

1. 源代码
2. //
3. // Created by Fitz on 2017/12/2.
4. //
5. #include <iostream>
6. #include <fstream>
7. #include <stack>
8. #include<queue>
9. #include <cstdlib>
10. **using** **namespace** std;
11. #define maxNum 30
13. **class** map{
14. **public**:
15. **int** pointNum,edgeNum;  //点的个数 边的个数
16. **int** point[maxNum],edge[maxNum][maxNum];  //在图的节点处存的信息  在图的边上存的信息(边的权重)
17. **void** initMap(**int** p,**int** e)    //将邻接矩阵初始化为全0
18. {
19. pointNum=p;
20. edgeNum=e;
21. **for**(**int** i=0;i<p;i++)
22. {
23. **for**(**int** j=0;j<p;j++)
24. {
25. edge[i][j]=0;
26. }
27. }
28. }
30. map() = **default**;
32. //为图的类添加一条无向的边
33. **void** addEdge(**int** a,**int** b,**int** weight)
34. {
35. edge[a][b]=weight;
36. edge[b][a]=weight;
37. }
39. };
41. //实例一个图的对象
42. map newMap;
43. //在dfs和bfs时需要判断节点是否已经访问过，这个数组用来存储节点是否已经访问过
44. **bool** visited[maxNum];
45. //存储搜索得到的搜索路径
46. **int** dfsCode[maxNum];
47. **int** count=0;
48. //通过文件读入图的信息并建立图
49. **void** readFile()
50. {
51. ifstream fileReader;
52. string filePath;
53. **int** p,e,a,b,w;
54. cout<<"请输入存储图信息的文件位置"<<endl<<"tips:格式：顶点个数，边的个数，顶点编号1，顶点编号2，边的权重"<<endl;
55. cin>>filePath;
56. fileReader.open(filePath.c\_str());
57. **if**(!fileReader.is\_open())
58. {
59. cout<<"找不到该文件，请输入有效的文件路径"<<endl;
60. exit(0);
61. }
62. fileReader>>p>>e;
63. newMap.initMap(p,e);
64. **for**(**int** i=0;i<e;i++)
65. {
66. fileReader>>a>>b>>w;
67. newMap.addEdge(a,b,w);
68. }
69. fileReader.close();
70. }
72. //深度优先搜索递归方式的写法
73. **void** DFSRecursive(**int** currentPoint)
74. {
75. visited[currentPoint]=**true**;
76. **for**(**int** i=0;i<newMap.pointNum;i++)
77. {
78. **if**(!visited[i]&&newMap.edge[currentPoint][i])
79. {
80. dfsCode[count]=i;
81. count++;
82. DFSRecursive(i);
83. }
84. }
85. }
87. //深度优先索搜用到的主函数
88. **void** DFSRecursiveMain()
89. {
90. count=0;
91. **for**(**int** i=0;i<newMap.pointNum;i++)
92. {
93. visited[i]=**false**;
94. dfsCode[i]=-1;
95. }
96. **for**(**int** i=0;i<newMap.pointNum;i++)
97. {
98. **if**(!visited[i])
99. {
100. dfsCode[count]=i;
101. count++;
102. DFSRecursive(i);
103. }
104. }
105. }
107. //深度优先搜索的非递归的写法
108. **void** DFSNotRecursive()
109. {
110. count=0;
111. **int** index,flag;
112. stack<**int**> trace;
113. **for**(**int** i=0;i<newMap.pointNum;i++)
114. {
115. visited[i]=**false**;
116. dfsCode[i]=-1;
117. }
118. **for**(**int** i=0;i<newMap.pointNum;i++)
119. {
120. index=i;
121. **while**(!visited[index])  //这个循环保证所有与该节点相连的没有访问过的点的都会搜索到
122. {
123. visited[index]=**true**;
124. dfsCode[count]=index;
125. trace.push(index);
126. count++;
127. **do**
128. {
129. flag=0;
130. **for**(**int** j=0;j<newMap.pointNum;j++)
131. {
132. **if**(!visited[j]&&newMap.edge[index][j])
133. {
134. index=j;
135. flag=1;
136. **break**;
137. }
138. }
139. **if**(flag==0)    //与该节点没有相连且没有访问过的点 ，需要回溯到之前访问的节点
140. {
141. trace.pop();
142. **if**(trace.empty())
143. {
144. flag=1;
145. **break**;
146. }
147. **else**
148. {
149. index=trace.top();
150. }
151. }
152. }**while**(flag==0);    //该循环一直持续到回溯到有满足条件的新点可以访问为止
153. }
154. }
155. }
157. //广度优先搜索的函数
158. **void** BFS()
159. {
160. queue<**int**> point;
161. count=0;
162. **for**(**int** i=0;i<newMap.pointNum;i++)
163. {
164. visited[i]=**false**;
165. dfsCode[i]=-1;
166. }
167. **for**(**int** i=0;i<newMap.pointNum;i++)
168. {
169. **if**(!visited[i])
170. {
171. visited[i]=**true**;
172. dfsCode[count]=i;
173. count++;
174. point.push(i);
175. **while**(!point.empty())
176. {
177. **for**(**int** j=0;j<newMap.pointNum;j++)
178. {
179. **if**(!visited[j]&&newMap.edge[point.front()][j])
180. {
181. point.push(j);
182. visited[j]=**true**;
183. dfsCode[count]=j;
184. //cout<<j<<endl;
185. count++;
186. }
187. }
188. point.pop();
189. }
190. }
191. }
192. }
194. **int** main()
195. {
196. readFile();
197. DFSRecursiveMain();
199. cout<<"深度优先搜索递归 ";
200. **for**(**int** i=0;i<newMap.pointNum;i++)
201. cout<<dfsCode[i]<<" ";
202. cout<<endl;
203. DFSNotRecursive();
204. cout<<"深度优先搜索非递归 ";
205. **for**(**int** i=0;i<newMap.pointNum;i++)
206. cout<<dfsCode[i]<<" ";
207. cout<<endl;
208. BFS();
209. cout<<"广度的优先搜索 ";
210. **for**(**int** i=0;i<newMap.pointNum;i++)
211. cout<<dfsCode[i]<<" ";
212. cout<<endl;
213. **return** 0;
214. }

哈尔滨工业大学计算机科学与技术学院

实验报告

课程名称：数据结构

课程类型：必修

试验项目名称：图结构及其应用

实验题目：设备更新问题

班级：1603005

学号：1160300524

姓名：冯运

一、 实验目的

通过设备更新问题这个实际的应用型问题，学会如何应用学过的图的模型将实际问题抽象成数学模型，并编程解决抽象出的图，从而达到加深对图的应用能力

二、实验要求及实验环境

实验要求：

要求根据给出的各年份的购进新设备和维修旧设备的价格，定制出一份各年份应该购进设备还是维修设备的规划，使花费的总金额最少。这个实验要求将实际的问题抽象成图的模型，使用图的算法将该问题解决。

实验环境：

实验环境：寝室，机房，Clion集成开发环境

三、设计思想

数据类型定义

1. **int** pointNum;  //图上点的个数
2. **double** edge[Max][Max];   //图上边的长度
3. **bool** s[Max];   //记录某个节点是否访问过
4. **double** dis[Max],p[Max];   //dis[]记录从初始节点到其他各个节点的最短距离  p[]记录当前节点的前一个访问的节点
5. MapMatrix newMap;   //用图的类实例化一个图的对象
6. **int** year,n;   //year 是总的年份
7. **int** newDevicePrice[Max];   //在某一年更新设备的花费
8. **int** repareDevicePrice[Max]; //在某一年维修设备的花费
9. stack<**int**> trace;   //将路径的信息倒序时需要用到的栈的数据结构

函数定义

1. **void** initMap(**int** p);   //将邻接矩阵初始化为全0
2. **void** addEdge(**int** a ,**int** b , **double** weight);   //为图的类添加一条无向的边
3. **void** dijkstra();   //dijkstra算法的具体实现

建立数学模型的思路：

先建立一个由问题抽象出来的图，定义从i点到j点的一条边表示在i年购进设备，一直用到第j-1年，在第j年购进新设备，边的权重为这段时间的花费。

在建图的过程中还有一点注意事项：除了遵循此规则外，还要在储存接通最后一个

节点的边时（到第n年时），比较维修的花费大还是购置的费用大。存储的较小的那一个。

按照这些规则建立图之后，将图的点和边的信息存入邻接矩阵中，使用dijkstra算法求出从第1点到第n点的最小路径即可求解。

主要函数的原理

1. //这个三重循环主要用来为图录入边的信息
2. **for**(**int** i=1;i<=year;i++)
3. {
4. //以第i年为边的一个起点，以第j年为中点
5. **for**(**int** j=i+1;j<=year;j++)
6. {
7. temp=0;
8. //这个循环 需要加上多年的维修费用
9. **for**(**int** k=0;k<j-i;k++)
10. {
11. temp+=repareDevicePrice[k];
12. //如果是计算到最后一年的花费，需要判断最后一年重新买新的设备合算还是维修合算，这样就可以减少一个节点
13. **if**(k==year-i-1)
14. temp+=repareDevicePrice[k+1]<newDevicePrice[year-1]+repareDevicePrice[0]?repareDevicePrice[k+1]:newDevicePrice[year-1]+repareDevicePrice[0];
15. }
16. newMap.addEdge(i,j,newDevicePrice[i-1]+temp);   //在图的对象中添加该边的信息
17. }
18. }

四、测试结果如图1

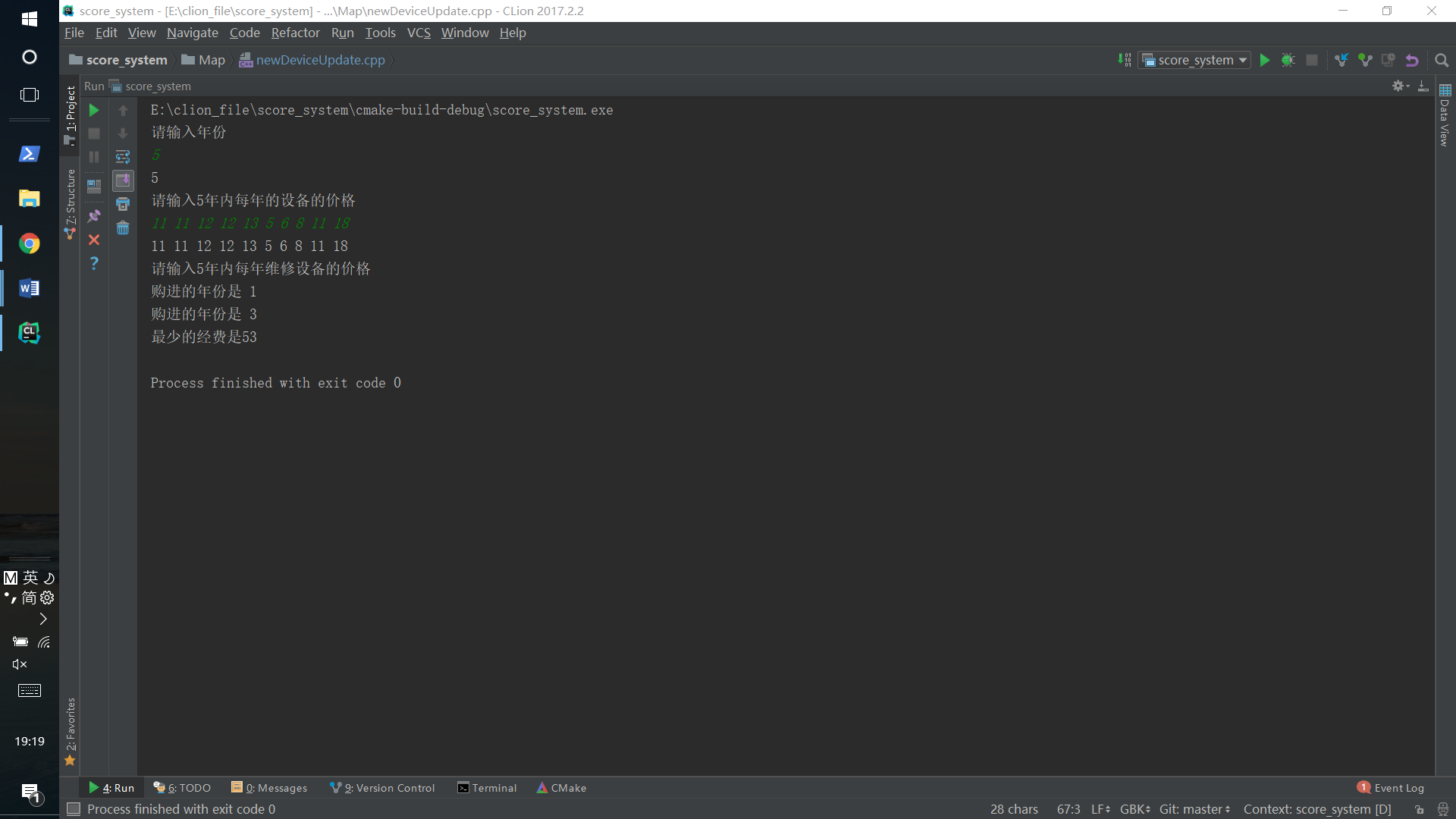


图1

五、系统的不足与体会

不足：

这个实验的主要难度在于将实际的问题抽象成一个可以用算法解决的数学模型，我在求解实际的问题方面明显表现出经验和能力不足，缺乏训练，以后需要多练习抽象实际问题的能力，提高应用能力。

体会：

这个系统在刚开始的时候没有输出购进的年份，只输出了最少的经费，但从实际的应用需求出发，更需要输出购进的年份，如何购进的策略，这才是用计算机解决这个问题的意义所在，而不仅仅得到最少的经费就结束，以后编写程序要多从用户的需求方面考虑。

六 、源代码

1. **class** MapMatrix {
2. #define Max 501
3. #define infinite 0x3fffffff
4. **public**:
5. **int** pointNum;  //图上点的个数
6. **double** edge[Max][Max];   //图上边的长度
7. **bool** s[Max];   //记录某个节点是否访问过
8. **double** dis[Max],p[Max];   //dis[]记录从初始节点到其他各个节点的最短距离  p[]记录当前节点的前一个访问的节点
9. **void** initMap(**int** p);   //将邻接矩阵初始化为全0
10. **void** addEdge(**int** a ,**int** b , **double** weight);   //为图的类添加一条无向的边
11. **void** dijkstra();   //dijkstra算法的具体实现
13. **private**:
14. **int** dijkstra\_MinCost();   //选取起始点到其他为访问过的点的距离最小的点
15. };



20. **void** MapMatrix::initMap(**int** p)
21. {
22. pointNum=p;
23. **for**(**int** i=1;i<=pointNum;i++)
24. {
25. **for**(**int** j=1;j<=pointNum;j++)
26. {
27. edge[i][j]=infinite;
28. }
29. }
30. }
32. **void** MapMatrix::addEdge(**int** a,**int** b,**double** weight)
33. {
34. edge[a][b]=weight;
35. }
37. **void** MapMatrix::dijkstra()
38. {
39. **int** w;
40. **double** sum;
41. **for**(**int** i=2;i<=pointNum;i++)
42. {
43. dis[i]=edge[1][i];
44. s[i]=**false**;
45. p[i]=1;
46. }
47. dis[1]=0;
48. s[1]=**true**;
49. **for**(**int** i=1;i<=pointNum-1;i++)
50. {
51. w=dijkstra\_MinCost();
52. s[w]=**true**;
53. **for**(**int** v=2;v<=pointNum;v++)
54. {
55. **if**(!s[v])
56. {
57. sum=dis[w]+edge[w][v];
58. **if**(sum<dis[v])
59. {
60. dis[v]=sum;
61. p[v]=w;
62. }
63. }
64. }
65. }
66. }
67. **int** MapMatrix::dijkstra\_MinCost()
68. {
69. **int** temp=infinite;
70. **int** w=2;
71. **for**(**int** i=2;i<=pointNum;i++)
72. {
73. **if**(!s[i]&&dis[i]<temp)
74. {
75. temp=dis[i];
76. w=i;
77. }
78. }
79. **return** w;
80. }



85. #include<iostream>
86. #include<stack>
87. #include"Map.h"
88. #include <cstdlib>
89. **using** **namespace** std;
91. MapMatrix newMap;   //用图的类实例化一个图的对象
92. **int** year,n;   //year 是总的年份
93. **int** newDevicePrice[Max];   //在某一年更新设备的花费
94. **int** repareDevicePrice[Max]; //在某一年维修设备的花费
95. stack<**int**> trace;   //将路径的信息倒序时需要用到的栈的数据结构

98. **int** main()
99. {
100. **int** w,sum,temp;
101. cout<<"请输入年份"<<endl;
102. cin>>year;
103. cout<<"请输入"<<year<<"年内每年的设备的价格"<<endl;
104. **for**(**int** i=0;i<year;i++)
105. {
106. cin>>newDevicePrice[i];
107. }
108. cout<<"请输入"<<year<<"年内每年维修设备的价格"<<endl;
109. **for**(**int** i=0;i<year;i++)
110. {
111. cin>>repareDevicePrice[i];
112. }
113. n=year;
114. newMap.initMap(year);
115. //这个三重循环主要用来为图录入边的信息
116. **for**(**int** i=1;i<=year;i++)
117. {
118. //以第i年为边的一个起点，以第j年为中点
119. **for**(**int** j=i+1;j<=year;j++)
120. {
121. temp=0;
122. //这个循环 需要加上多年的维修费用
123. **for**(**int** k=0;k<j-i;k++)
124. {
125. temp+=repareDevicePrice[k];
126. //如果是计算到最后一年的花费，需要判断最后一年重新买新的设备合算还是维修合算，这样就可以减少一个节点
127. **if**(k==year-i-1)
128. temp+=repareDevicePrice[k+1]<newDevicePrice[year-1]+repareDevicePrice[0]?repareDevicePrice[k+1]:newDevicePrice[year-1]+repareDevicePrice[0];
129. }
130. newMap.addEdge(i,j,newDevicePrice[i-1]+temp);   //在图的对象中添加该边的信息
131. }
132. }
133. newMap.dijkstra();
134. **int** i=n;
135. **while**(newMap.p[i]!=0)
136. {
137. i=newMap.p[i];
138. trace.push(i);
139. }
140. **while**(!trace.empty())
141. {
142. cout<<"购进的年份是 "<<trace.top()<<endl;
143. trace.pop();
144. }
145. cout<<"最少的经费是"<<newMap.dis[n]<<endl;
146. **return** 0;
147. }