最短路径实验报告

PB17050948 陈文博

PB17061172 冯宇衡

实验内容

1.结构体定义

railway结构体：

```
  typedef struct railway{
     int**G;//图的邻接矩阵
     int**D;//图的最短路径长矩阵
     std::string*city;//城市名数组
     std::string**path;//最短路径数组
 }railway;
```

结构体定义在Railway.h文件中

2. 函数定义

程序Railway.cpp中的函数包括：

void Create_Graph(int **G); void Path_Init(railway R); void Floyd(railway&R); bool Delete_City(railway&R,std::string city); void Reset(railway&R);

(1) void Ceate_Graph(int **G)

1. 函数功能：从文本中读取信息创建图的邻接矩阵。
2. 函数入口：二维数组G。
3. 算法流程：为G开辟内存空间，遍历文件，将相应的值赋到G中。
4. 算法分析：时间复杂度O（n^2），空间复杂度O（n^2）

(2) void Path_Init(railway R)

1. 函数功能：路径数组初始化
2. 函数入口：railway结构体R
3. 算法流程：遍历R.path，若R.G[i][j]!=-1（即城市存在），则R.path[i][j]=R.city[i]+" → "+R.city[j]，否则填入"None"
4. 算法分析：时间复杂度O（n^2），空间复杂度O（n^2）

(3) void Floyd(railway&R);

1. 函数功能：Floyd算法求解最短路径
2. 函数入口：railway结构体R

3. 算法流程：将变量k，i，j分别由0到MAXSIZE嵌套遍历，若R.D[i][j]>R.D[i][k]+R.D[k][j]，且三个值任意一个不为-1，则更新R.D[i][j]的值：R.D[i][j]=R.D[i][k]+R.D[k][j]，同时更新path的值：R.path[i][j]=R.path[i][k]+get_tail(R.path[k][j])
4. 算法分析：时间复杂度O（n^3），空间复杂度O（n^2）

（4）bool Delete_City(railway&R,std::string city)

1. 函数功能：从图中删除城市
2. 函数入口：railway结构体R，城市名city
3. 算法流程：将R.G中城市对应编号的行列元素全部置-1
4. 算法分析：时间复杂度O（n），空间复杂度O（n^2）

（5）void Reset(railway&R)

1. 函数功能：重置最短路径以及最短路径长数组
2. 函数入口：railway结构体R
3. 算法流程：遍历R.D并赋值G.[i][j]，初始化D.path
4. 算法分析：时间复杂度O（n^2），空间复杂度O（n^2）

3. 实验结果

程序运行初始状态：

```
1. Get shortest path
2. Delete city
3. Quit
Input index to select operation:
```

打印沈阳至西安:

```
Path:Shenyang → Tianjin → Beijing → Zhengzhou → Xi'an
Distance:2047
```

呼和浩特至成都：

```
Path:Huhehaote → Lanzhou → Xi'an → Chengdu
Distance:2663
```

上海至乌鲁木齐:

```
Path:Shanghai → Xuzhou → Zhengzhou → Xi'an → Lanzhou → Wulumuqi
Distance:4079
```

删除郑州再打印：

沈阳至西安:

```
Path:Shenyang → Tianjin → Beijing → Huhehaote → Lanzhou → Xi'an
Distance:3330
```

呼和浩特至成都：

```
Path:Huhehaote → Lanzhou → Xi'an → Chengdu
Distance:2663
```

上海至乌鲁木齐：

```
Path:Shanghai → Xuzhou → Tianjin → Beijing → Huhehaote → Lanzhou → Wulumuqi
Distance:5167
```

4.讨论与总结及组队心得

在用Dijkstra算法进行编程的过程中明显发现代码极其复杂，而且由于算法只能求取从某一个点出发的路径，因此每次求取特定两点的最短路径时都需要重新进行路径的计算，虽然Dijkstra算法单次运行的复杂度是n^2，但是平均下来复杂反而比Floyd算法高（系数高，阶数均为n^3）。

基于上述问题，两人上网搜索并学习了Floyd算法，不禁惊叹于其巧妙。算法通过构建一个最短路径中的相邻点矩阵，非常巧妙地在一个二维矩阵中存储了任意两点之间的最短路径信息，并且代码极其简洁。缺点就是不那么能完全把握其思想（但是算法本身非常直观）。

结队编程过后的心得是：

1.敲代码时被队员盯着容易分神，导致编程时的思考效率低于单人编程。虽然到实验课结束都没有太圆满地解决这个问题，但是我们确信这是可以通过训练和磨合来克服的；

2.有时敲代码的队员会灵光一闪然后自顾自写起来，另一个队员常常跟不上其思路，于是敲代码的队员不得不停下来细细解释，导致编程效率降低；

3.但是上述问题在某种程度上可以被结对编程的优势取代。虽然解释很费时，但是有一个头脑清醒的审视者坐在一旁，一些由于考虑不周或者一时冲动所犯的错误可以被及时发现并纠正，大大降低代码思路、结构方面的出错率；

4.在运行发现错误时，两个人的排错能力明显高于独自一人，相对更不容易陷在某个错误中解脱不得。但是，当两个人都没有发现某个细微错误时，由于排错的人需要确认队员的想法，反而把“通过设断点或者穷举排错”的效率拉低；

5.由于这次结队编程过程比较有趣，两人精神都一直很高昂，反观独自写代码时，很容易在长时间的排错工作中渐渐颓废懈怠下来。

总的来说，结队编程优劣之处兼而有之，如果从编程经历角度来看还是一次有趣的体验，也加深了相互之间的了解。对方一些优秀的编程思路和风格也有不小的启发意义。但是，如果在未来的程序设计工作中让我们选择的话，应该还是会毫不犹豫地独自编程吧，哈哈。