**数据结构课程设计报告**

# 课程设计题目：用单链表实现集合的操作

## 一、问题描述

用有序单链表实现集合的判等、交、并和差等基本运算。基本要求：⑴ 对集合中的元素用有序单链表进行存储；⑵ 实现交、并、差等基本运算时，不能另外申请存储空间；⑶ 充分利用单链表的有序性，要求算法有较好的时间性能。

## 二、数据结构的设计

通过分析题目得到信息应当使用单链表这种数据结构，其中节点采用结构体定义。

//结构定义

typedef int DataType; /\*定义线性表内存储数据的数据类型，假设为int型\*/

typedef struct Node /\*定义单链表的结点类型\*/

{

DataType data;

struct Node \*next;

} Node;

## 三、算法设计及性能分析

### 主函数的调用流程：

1. 首先对所给数据进行初始化排序，调用QuickSort函数
2. 定义变量用于存储题目数据
3. 采用尾插法，调用TailCreatList函数完成单向链表的创建
4. 展示当前链表，调用PrintList函数
5. 根据用户需求，对应完成集合的判等、交、并和差等基本运算。
6. 调用DestroyList函数释放单链表

### 函数：共有12个函数

1. void PrintList(Node \*first)

传入单链表头节点，完成链表的打印功能

1. int Length(Node \*first)

传入单链表头节点，返回当前链表的长度

1. Node \*TailCreatList(DataType a[], int n)

传入集合数组以及其长度，通过尾插法完成单链表的创建，返回单链表的头节点

1. void DestroyList(Node \*first)

传入一个单链表的头节点，释放其内存

1. int Partition(int r[], int first, int end)

void QuickSort(int r[], int first, int end)

传入集合数组以及其起点与终点，采用快速排序算法完成对集合的初始化排序

1. int IsEqual(Node \*A, Node \*B)

输入：两个单链表的头节点

输出：1表示两集合相等，0表示两集合不等

算法性能分析：O（n+m）

1. Node \*pa = A->next;

Node \*pb = B->next;

2.循环直到pa或pb为NULL

2.1如果两指针指向的节点数据域相等则指针后移,否则退出循环

3.判断指针pa和pb是否都为NULL

* 1. 若都为NULL则输出1，否则输出0

7.1 void Interest(Node \*A, Node \*B)

输入：两个单链表的头节点

输出：无

算法性能分析：O（n+m）

1. Node \*p = A->next;

Node \*q = B->next;

Node \*pre = A;

2.循环直到p或q为NULL

2.1如果p->data < q->data

则pre->next = p->next;

delete p;

p = pre->next;

2.2如果p->data > q->data

则q = q->next;

2.3否则则为p->data = q->data

pre = p;

p = p->next;

q = q->next;

7.2 Node \*Union\_newset(Node \*A, Node \*B)（思考题对应的函数）

输入：两个单链表的头节点

输出：存储结果的一个单链表

算法性能分析：O（n+m）

1. Node \*p = A->next;

Node \*q = B->next;

Node \*newList = (Node \*)malloc(sizeof(Node));

newList->next = NULL;

Node \*newp = newList;

2.循环直到p或q为NULL

2.1如果p->data < q->data

则p = p->next;

2.2如果p->data > q->data

则q = q->next;

2.3否则

则为newp->next = p;

newp = newp->next;

p = p->next;

q = q->next;

3.输出生成的单链表

* 1. void Union(Node \*A, Node \*B)

输入：两个单链表的头节点

输出：无

算法性能分析：O（n+m）

1. Node \*p = A->next;

Node \*q = B->next;

Node \*pre = A;

2.循环直到p或q为NULL

2.1如果p->data < q->data

则p = p->next;

pre = pre->next;

2.2如果p->data > q->data

则pre->next = q;

q = q->next;

pre = pre->next;

pre->next = p;

2.3否则则为p->data = q->data

p = p->next;

q = q->next;

pre = pre->next;

3．判断如果q不为NULL

p->next = q;

* 1. Node \*Union\_newset(Node \*A, Node \*B)（思考题对应的函数）

输入：两个单链表的头节点

输出：存储结果的一个单链表

算法性能分析：O（n+m）

1. Node \*p = A->next;

Node \*q = B->next;

Node \*newList = (Node \*)malloc(sizeof(Node));

newList->next = NULL;

Node \*newp = newList;

2.循环直到p或q为NULL

2.1如果p->data < q->data

newp->next = p;

newp = newp->next;

p = p->next;

2.2如果p->data > q->data

newp->next = q;

newp = newp->next;

q = q->next;

2.3否则

则为newp->next = p;

newp = newp->next;

p = p->next;

q = q->next;

3.判断如果q不为NULL

newp->next = q;

4.判断如果p不为NULL

newp->next = p;

5.输出生成的单链表



9.1 void Difference (Node \*A, Node \*B)

输入：两个单链表的头节点

输出：无

算法性能分析：O（n+m）

1. Node \*p = A->next;

Node \*q = B->next;

Node \*pre = A;

2.循环直到p或q为NULL

2.1如果p->data < q->data

则p = p->next;

pre = pre->next;

2.2如果p->data > q->data

则q = q->next;

2.3否则则为p->data = q->data

pre->next = p->next;

delete p;

p = pre->next;

q = q->next;

9.2 Node \* Difference\_newset (Node \*A, Node \*B)（思考题对应的函数）

输入：两个单链表的头节点

输出：存储结果的一个单链表

1. Node \*p = A->next;

Node \*q = B->next;

Node \*newList = (Node \*)malloc(sizeof(Node));

newList->next = NULL;

Node \*newp = newList;

2.循环直到p或q为NULL

2.1如果p->data < q->data

newp->next = p;

newp = newp->next;

p = p->next;

2.2如果p->data > q->data

q = q->next;

2.3否则

p = p->next;

q = q->next;

3输出生成的单链表

## 四、测试数据

输入数据：

setA[5] = {3, 1, 7, 9, 5}

setB[5] = {4, 2, 8, 5, 9}

获得用户的输入：

1.表示判断A和B是否相等

2.表示求集合A和B的交集

3.表示求集合A和B的并集

4.表示求集合A和B的差集

其余数字则为非法输入

输出数据：

1. 集合A是不等于集合B的！

感谢您的测试~

2. 集合A与集合B的交集为：59

以下为思考题的输出

集合A与集合B的交集为：59

感谢您的测试~

3. 集合A与集合B的并集为：12345789

以下为思考题的输出

集合A与集合B的并集为：12345789

感谢您的测试~

4. 集合A与集合B的差集为：137

以下为思考题的输出

集合A与集合B的差集为：137

感谢您的测试~

0. 请输入合法操作！

感谢您的测试~

## 五、心得体会

问题：

由于对单链表知识点产生的遗忘，导致产生空指针异常情况

在switch语句中定义变量会产生报错，因此变量只能定义在外面

函数传参时有关指针的部分虽然有所长进，但仍需提高

心得体会：

对于数据结构的底层逻辑仍需加强记忆

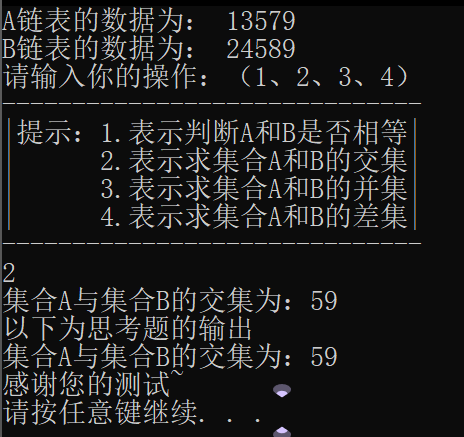
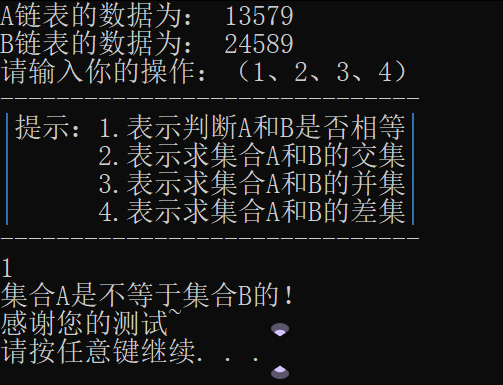
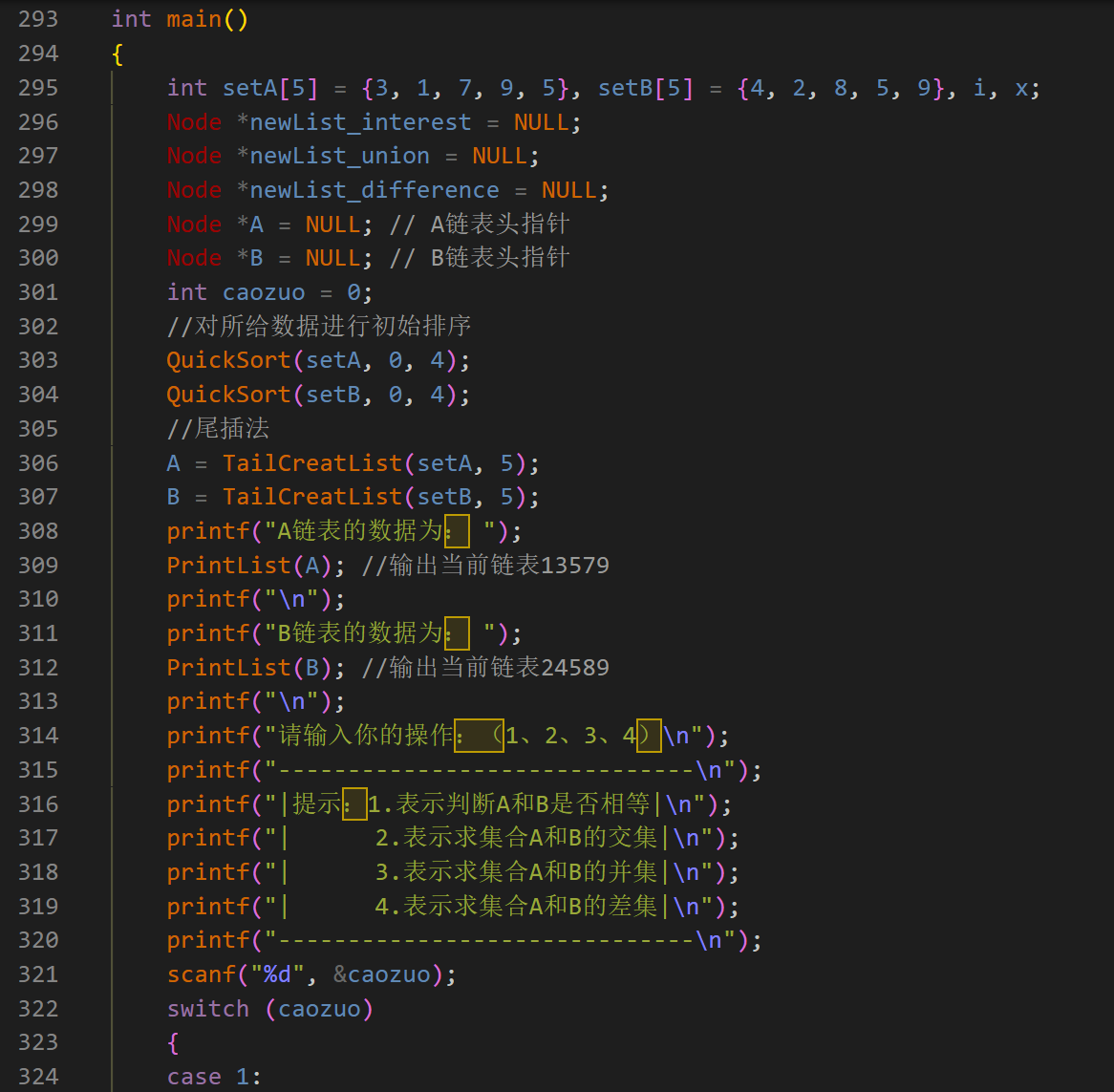
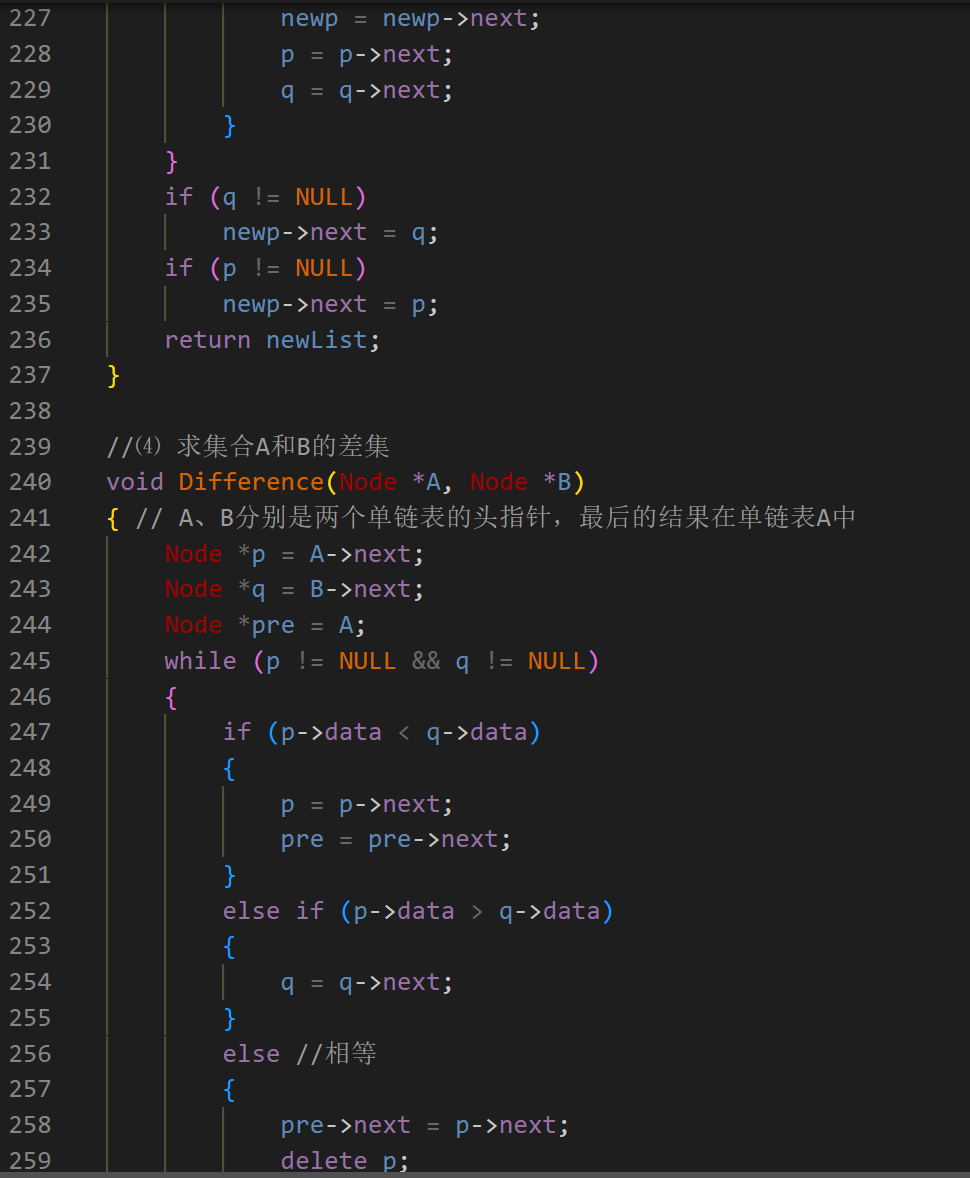
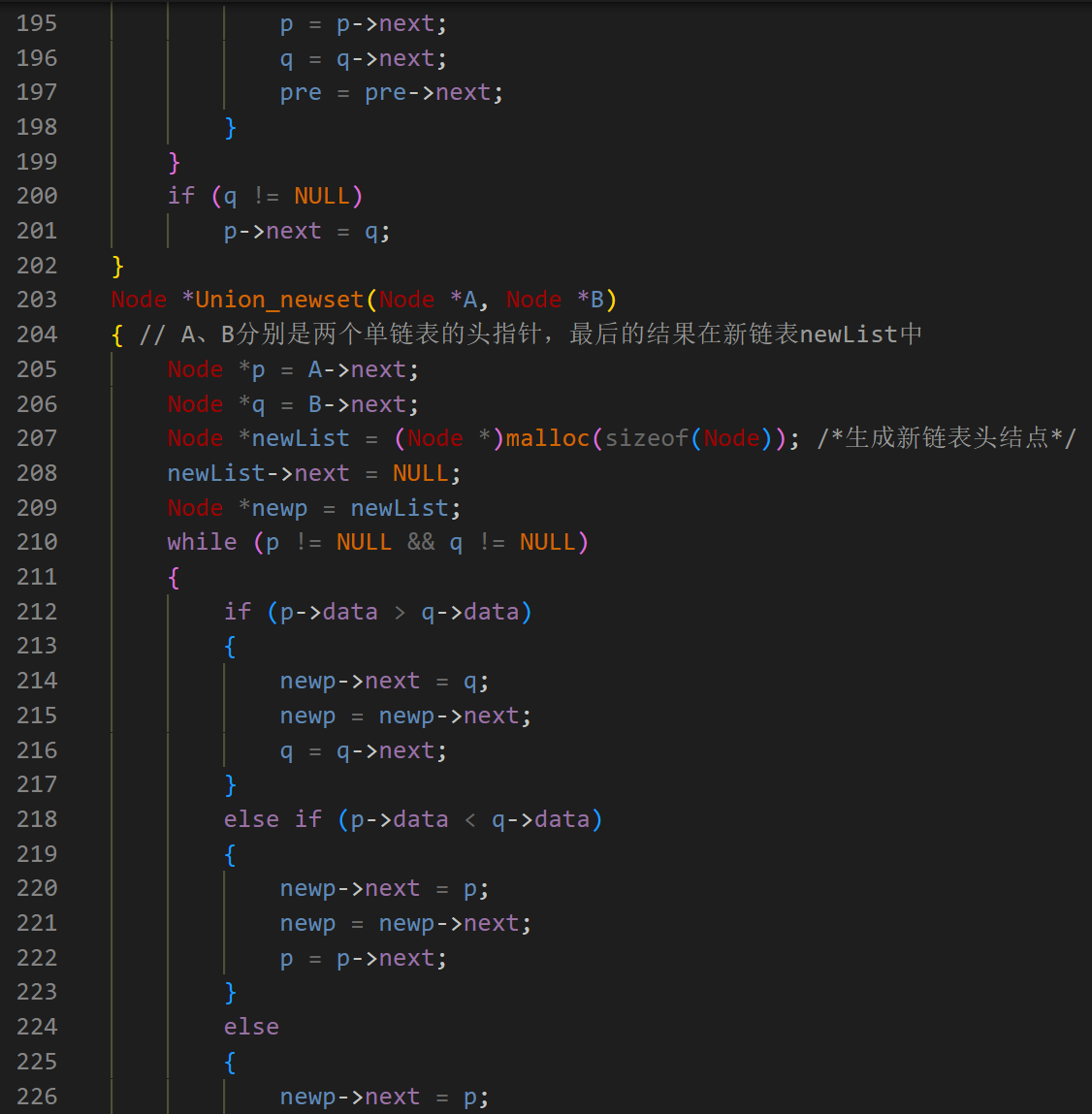
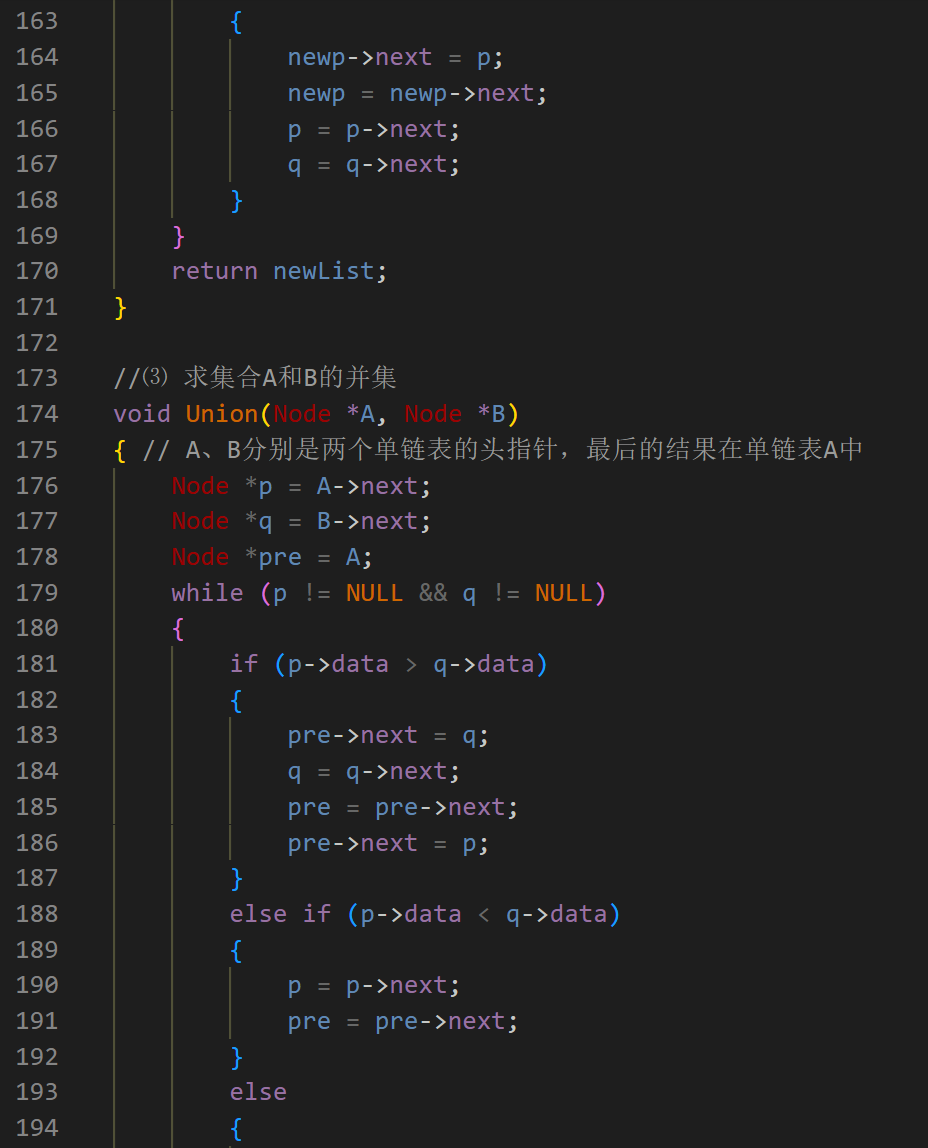
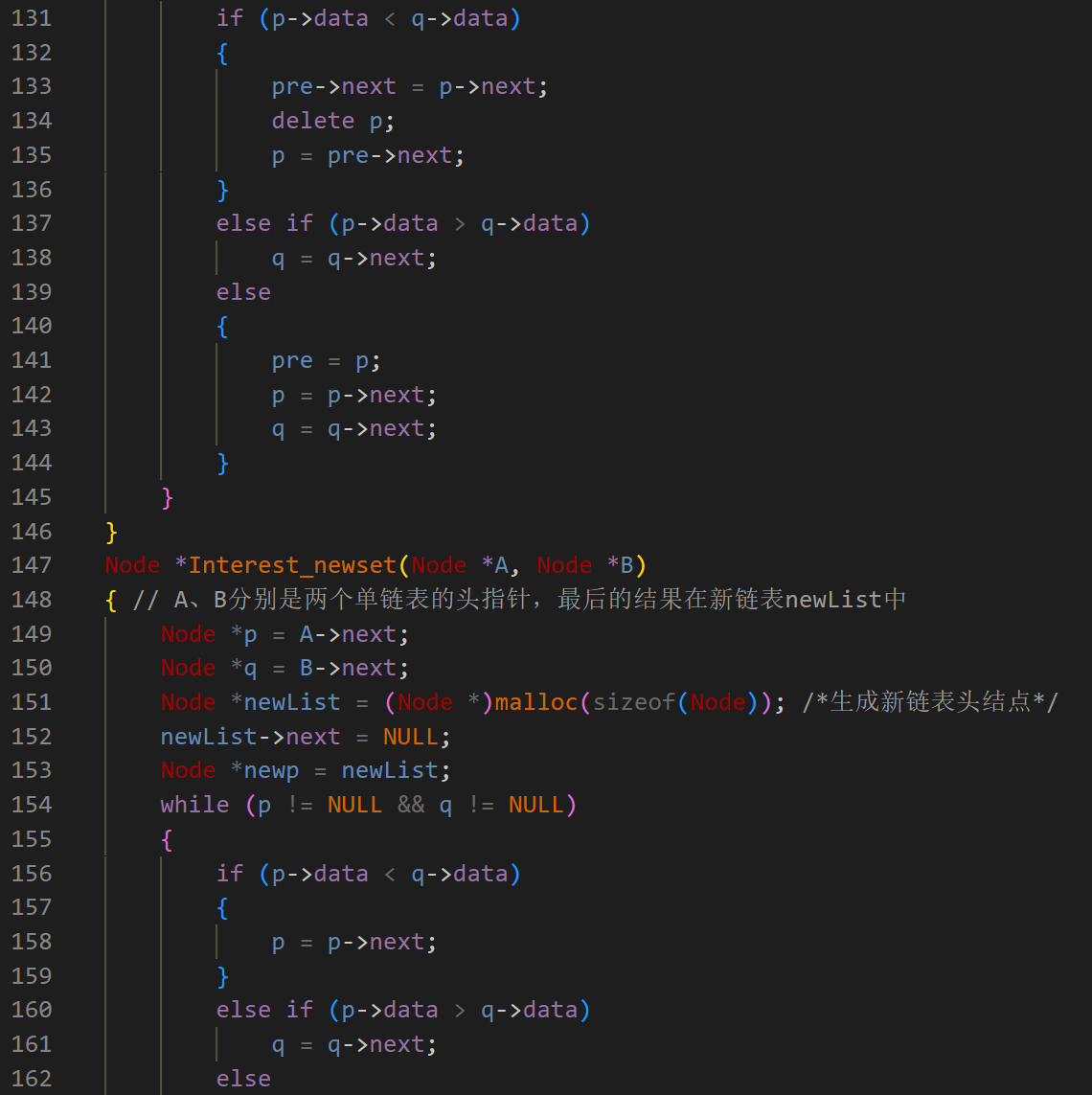
对于链表数据结构的存储方式还要继续学习

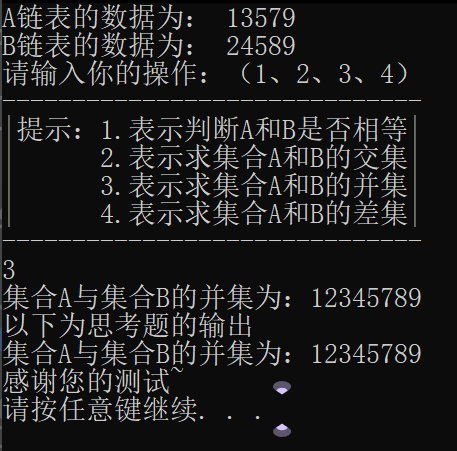
对于C语言的语法知识还需要积累

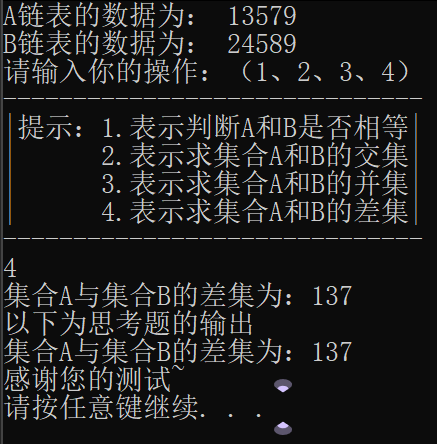
通过对本次题目的实践，也加强了我的代码熟练程度

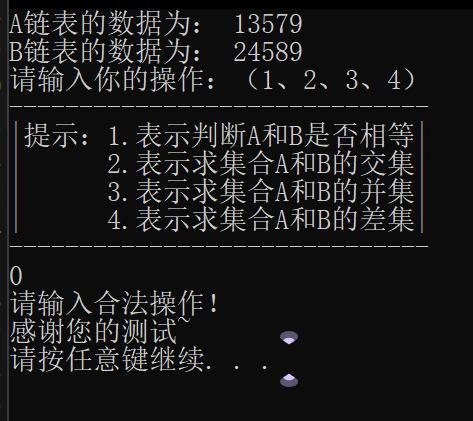
以及对于算法的设计能力

## 六、程序源码及运行结果

****

****

****

****

课程设计题目：迷宫问题

## 一、问题描述

迷宫求解是实验心理学中的一个经典问题，心理学家把一只老鼠从一个无顶盖的大盒子的入口处赶进迷宫，迷宫中设置很多隔壁，对前进方向形成了多处障碍，假设前进的方向有四个，分别是上、下、左、右，心理学家在迷宫的唯一出口处放置了一块奶酪，吸引老鼠在迷宫中寻找通路以到达出口。例如，图1所示为一个迷宫示意图。

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 入口 |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  | 出口 |

图1 迷宫示意图

基本要求：⑴ 设计数据结构存储迷宫；

⑵ 设计存储结构保存从入口到出口的通路；

⑶ 设计算法完成迷宫问题的求解；

⑷ 分析算法的时间复杂度。

## 二、数据结构的设计

通过分析题目得到信息应当使用二维数组这种数据结构

//创建迷宫地图

int map[8][10] = {0};

并将找到的“通路”标注在此地图中，因此也采用二维数组存储从入口到出口的“通路”

## 三、算法设计及性能分析

### 主函数的调用流程：

1. 首先创建一个二维数组用作地图
2. 初始化当前地图，调用initMap函数
3. 查看当前地图，调用watchMap函数
4. 寻找迷宫的通路，调用findWay函数

并展示搜索的过程（思考题1）

1. 查看寻找完成通路后的地图，调用watchMap函数

### 函数：共有4个函数

1. void initMap(int map[][10])

传入保存迷宫地图的二维数组，完成地图的初始化

1. void watchMap(int map[][10])

传入保存迷宫地图的二维数组，输出当前地图情况

1. int move(int i, int j, int m)

传入当前位置坐标以及移动的方向，完成位置的移动

1. bool findWay(int map[][10], int i, int j, int fi, int fj)

输入：保存迷宫地图的二维数组，起始位置坐标和终点坐标

输出：true表示找到了通路 false表示未找到通路

算法性能分析：O（n\*m）

伪代码：

/\* 规定：

0 无障碍且未走过

1 为墙

2 为通路

3 为无通路

（i,j）是起点

（fi,fj）是终点 \*/

1. 判断终点位置是否为2

如果找到通路，就返回true, 否则返回false

1. 否则如果map[i][j] == 0 则表示当前坐标没有走过

假定该点是可以走通

2.1如果按照策略 上->右->下->左 “走”

（思考题2）则在后面对应添加左上->左下->右下->右上 即可

能走通返回true 否则false

* 1. 均无法走通则说明该点是走不通，是死路

返回false

1. 否则如果map[i][j] != 0 , 可能是 1、 2、 3

返回false

## 四、测试数据

输入数据：

按照所给图片创建地图如下：

1 1 1 1 1 1 1 1 1 1

1 0 0 1 1 0 1 1 1 1

1 1 0 0 0 1 1 1 1 1

1 0 1 0 0 0 0 0 1 1

1 0 1 1 1 0 1 1 1 1

1 1 0 0 1 0 0 0 0 1

1 0 1 1 0 0 1 1 0 1

1 1 1 1 1 1 1 1 1 1

输出数据：

以下为试探通路的过程：

当前位于坐标(1,1)处

当前位于坐标(1,2)处

当前位于坐标(2,2)处

当前位于坐标(2,3)处

当前位于坐标(2,4)处

当前位于坐标(3,4)处

当前位于坐标(3,5)处

当前位于坐标(3,6)处

当前位于坐标(3,7)处

(2,8)是死路，四面八方都走不通！唯有原路返回了！！

当前位于坐标(4,5)处

当前位于坐标(5,5)处

当前位于坐标(5,6)处

当前位于坐标(5,7)处

当前位于坐标(5,8)处

当前位于坐标(6,8)处

通路找到啦！

以下为搜寻后的迷宫地图：（其中‘2’即为所寻找到的通路）

1 1 1 1 1 1 1 1 1 1

1 2 2 1 1 0 1 1 1 1

1 1 2 2 2 1 1 1 1 1

1 0 1 0 2 2 3 3 1 1

1 0 1 1 1 2 1 1 1 1

1 1 0 0 1 2 2 2 2 1

1 0 1 1 0 0 1 1 2 1

1 1 1 1 1 1 1 1 1 1

## 五、心得体会

问题：

在本题中用到递归算法，由于对底层逻辑还没有搞懂，导致陷入死循环

心得体会：

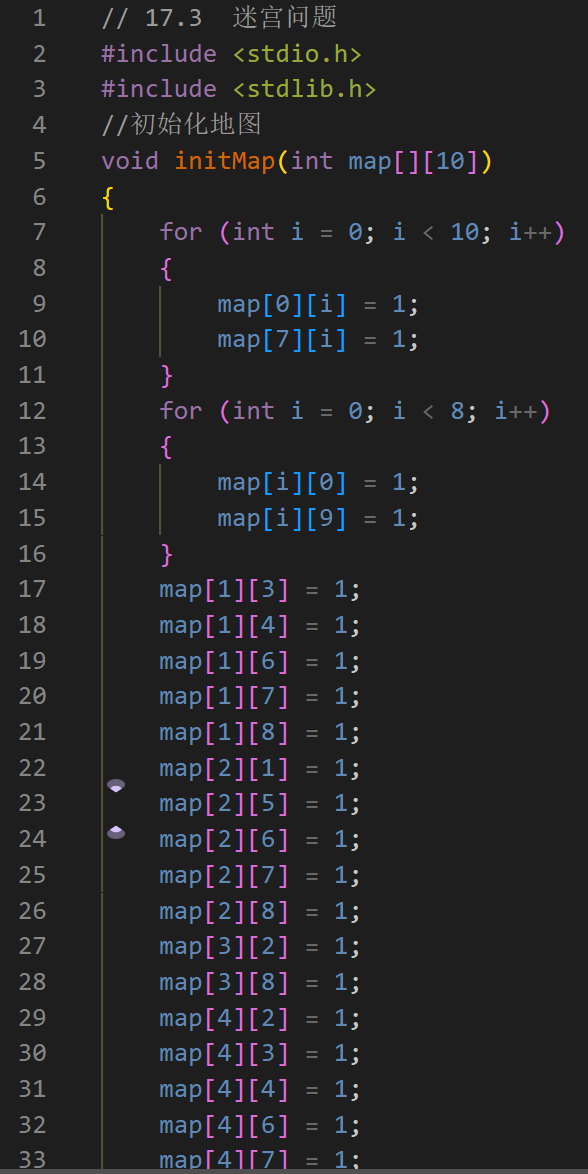
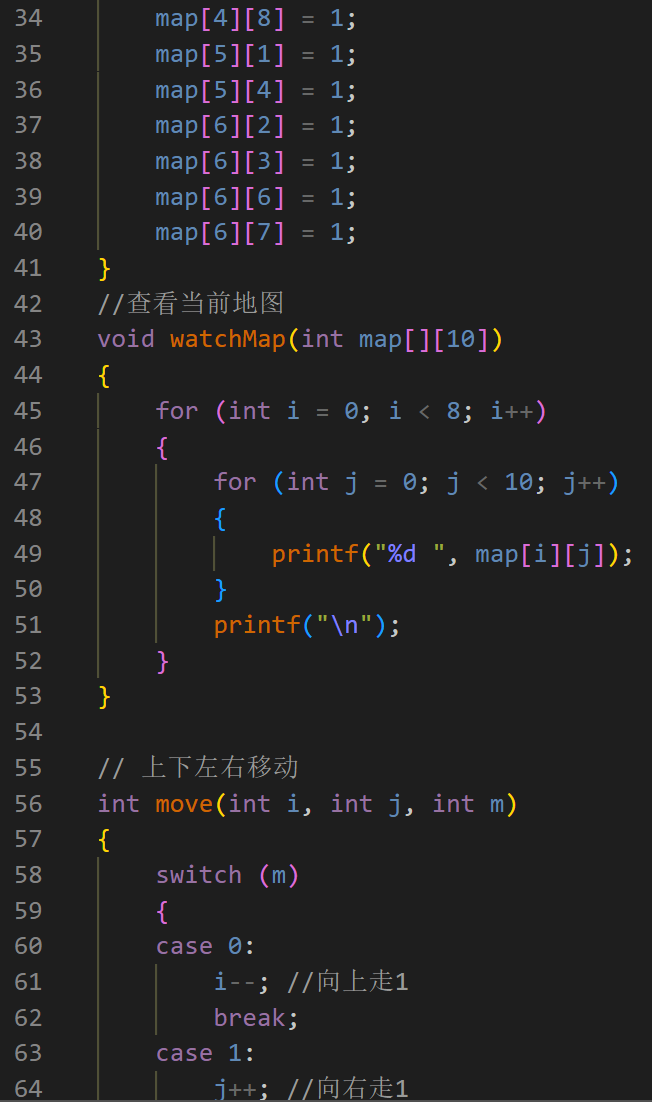
加强了对于递归算法的学习

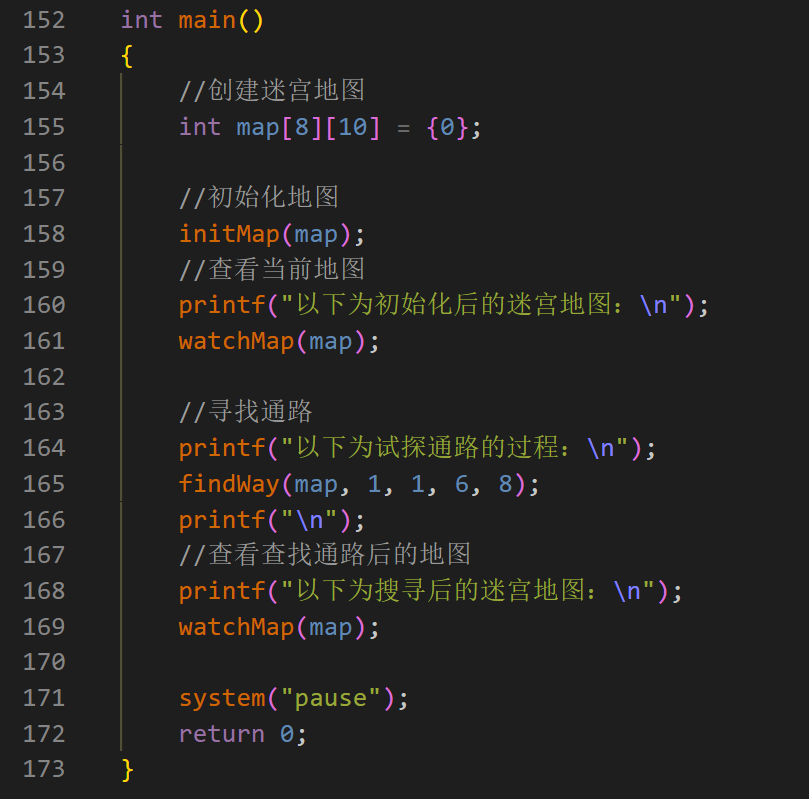
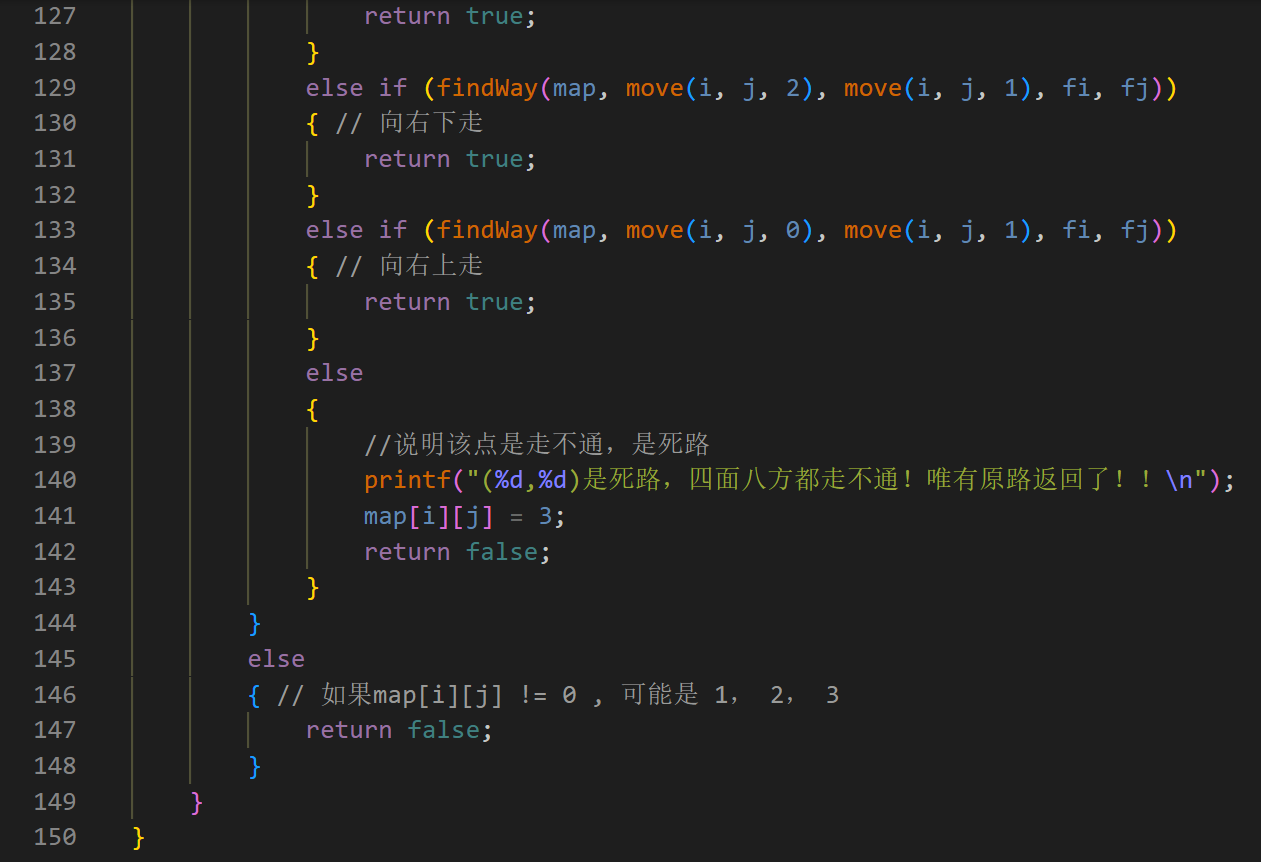
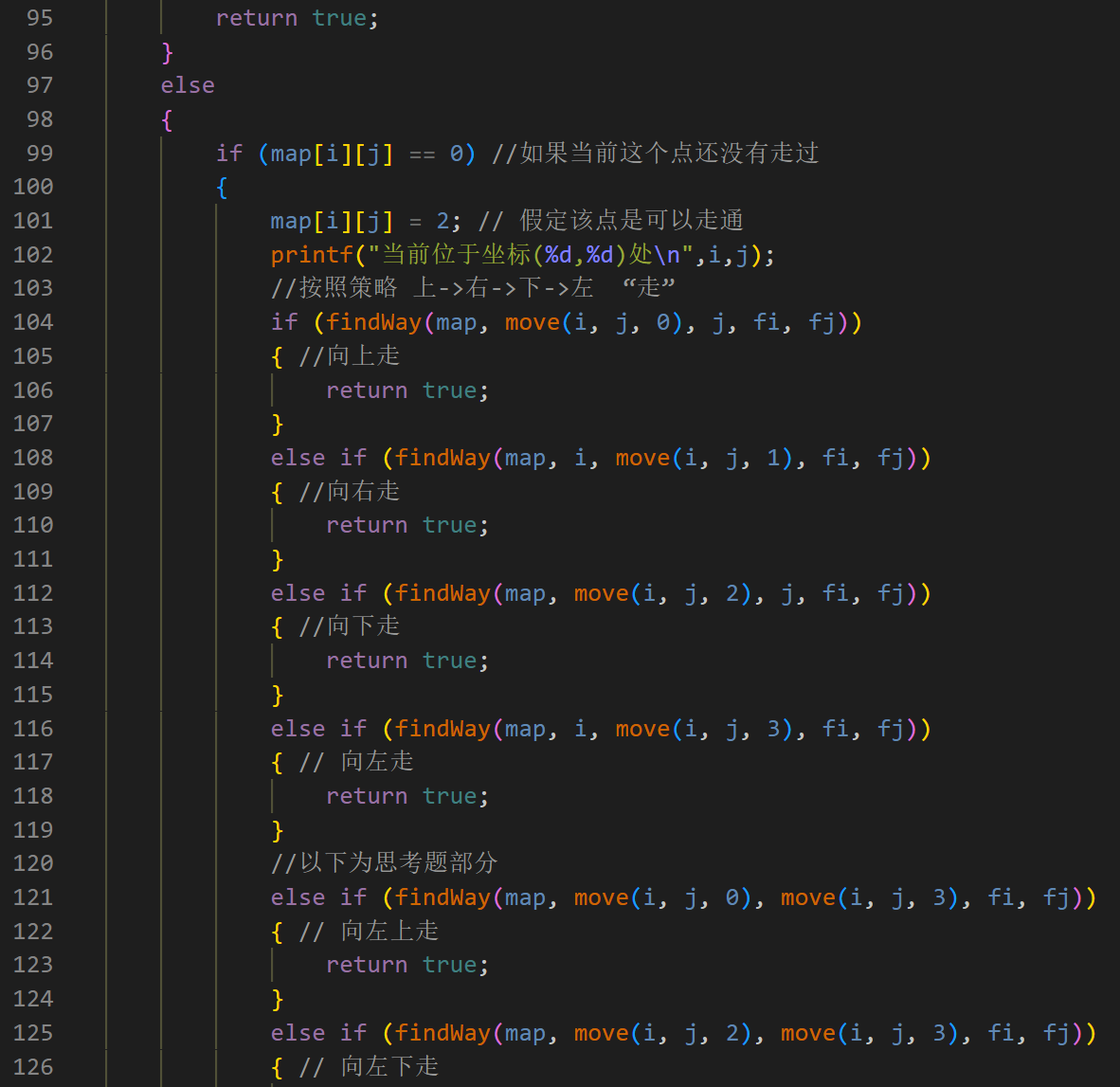
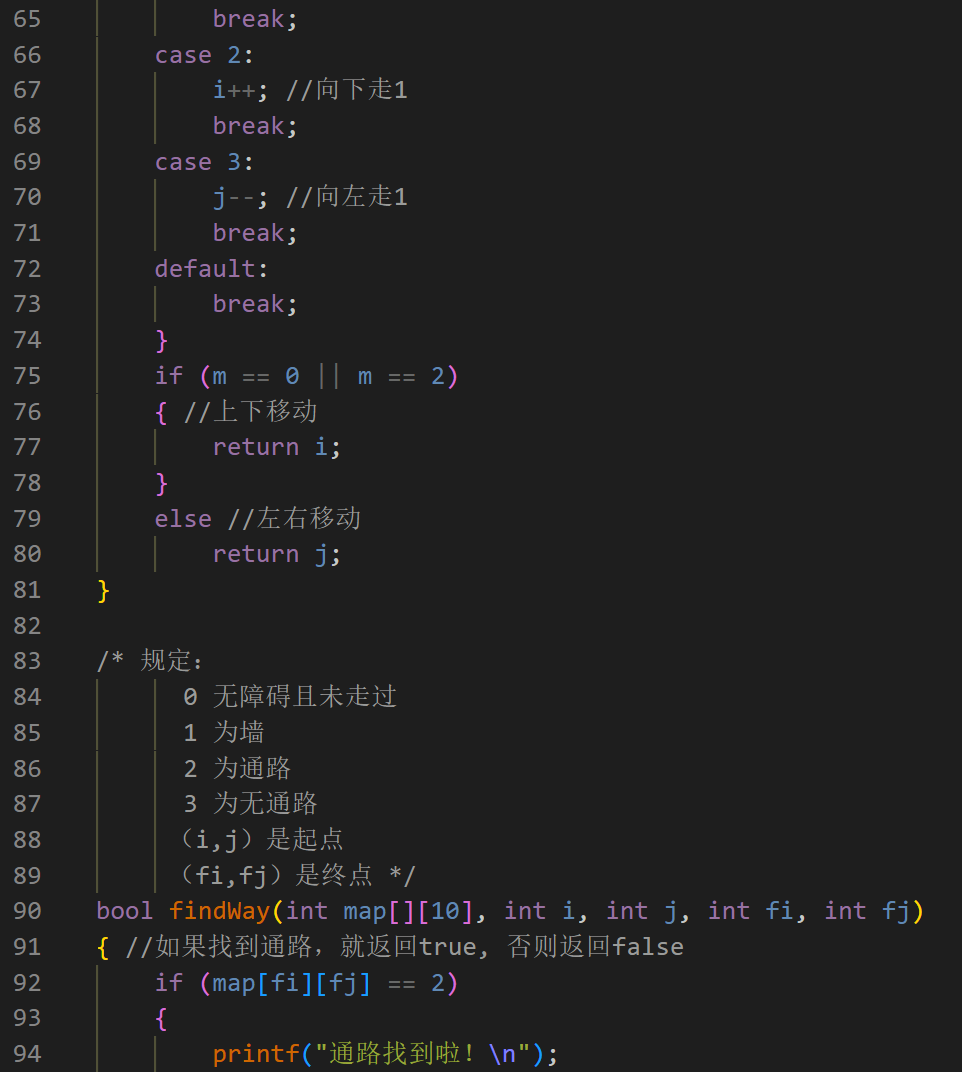
了解并熟悉了回溯法的思想，虽然仍不太熟练，但收获也是很大

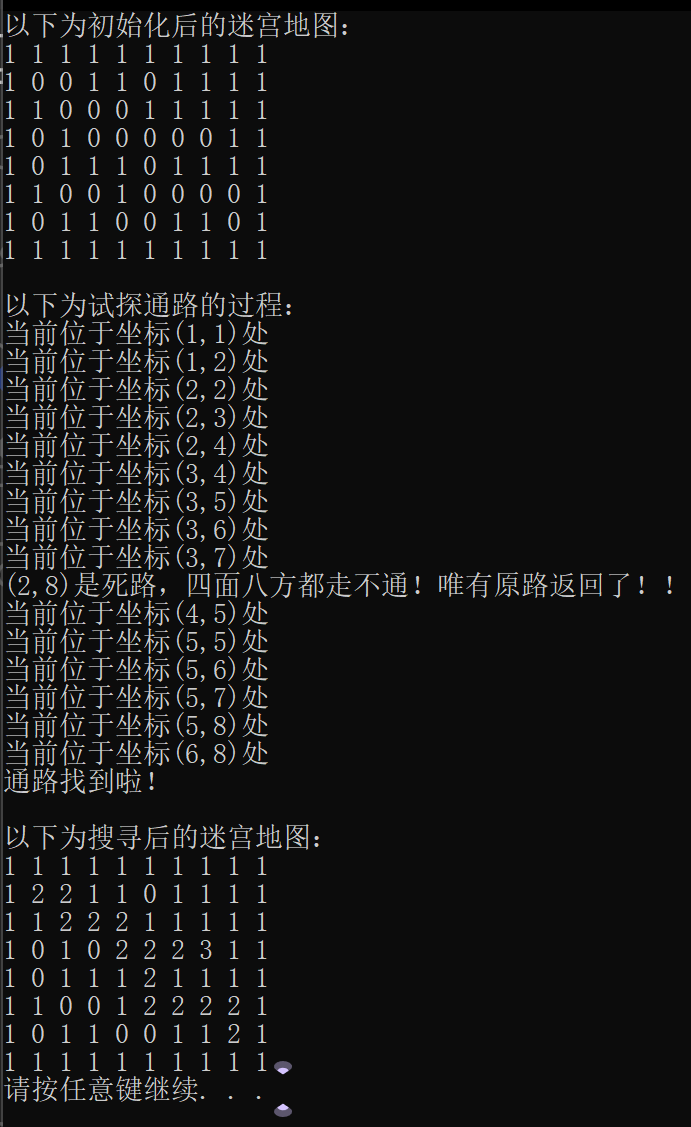
通过本题，对于函数的调用顺序理解也加强了

Debug查找错误的能力得到了加强

## 六、程序源码及运行结果

**** ****



****