****

**Tongji University**

**数据结构课程设计实验报告**

**003-勇闯迷宫游戏**

专 业： 软 件 工 程

指导教师： 张 颖

学 号： 2 1 5 3 0 6 1

姓 名： 谢 嘉 麒

1. **项目概述**
   1. **项目背景**

迷宫只有两个门，一个门叫入口，另一个门叫出口。一个骑士骑马从入口进入迷宫，迷宫设置很多障碍，骑士需要在迷宫中寻找通路以到达出口。

求解迷宫问题在生活中有很多应用场景，可以通过解决这类问题的思路类比解决地图寻路系统，路线搜寻系统等；依托这一项目的实现，可以为更进一步的优化奠定理论和实践基础。

* 1. **项目功能介绍**

迷宫问题的求解过程可以采用回溯法即在一定的约束条件下试探地搜索前进，若前进中受阻，则及时回头纠正错误另择通路继续搜索的方法。从入口出发，按某一方向向前探索，若能走通，即某处可达，则到达新点，否则探索下一个方向；若所有的方向均没有通路，则沿原路返回前一点，换下一个方向再继续试探，直到所有可能的道路都探索到，或找到一条通路，或无路可走又返回入口点。

在求解过程中，为了保证在达到某一个点后不能向前继续行走时，能正确返回前一个以便从下一个方向向前试探，则需要在试探过程中保存所能够达到的每个点的下标以及该点前进的方向，当找到出口时试探过程就结束了。

* 1. **项目需求分析**

对于解决迷宫寻路问题，在考虑算法的同时也需要考虑到算法的时间复杂度和空间复杂度，在一定程度上保证迷宫问题的求解速度（这一点在用户的实时需求中显得尤为重要）

而同时也可选择设定随机迷宫的生成算法，使得项目完整普适，实现多种情况下的测试。

1. 功能完备性：

本次实验需要考虑到项目的完备性，在实现迷宫寻路的同时，也实现了随机迷宫的生成算法。

1. 执行效率最优：

要考虑到数据较大情况下迷宫的求解速率，优化用户体验，实行执行效率最优。这一点主要体现在短时间内实现复杂迷宫的寻路，也要实现大迷宫下的求解两点路径问题。

1. 考虑程序健壮性

程序需要保证客户在输入错误数据之后，自动识别错误并分类提示，实现程序的健壮性。

1. 实现客户端迷宫的可视化

为了优化用户体验，需要将迷宫实现可视化，使得程序更加形象立体。

1. **项目设计思路**
   1. **项目设计流程示意图**

用户输入行列

Prim算法随机生成地图

判断行列正确性

打印地图，图形化显示

递归寻路

打印寻路地图

* 1. **项目主要函数调用示意图**

判断正误

置入迷宫行列

While循环

打印迷宫

Prim算法随机生成迷宫

Main函数

输入迷宫起点终点

判断正误

打印路径

打印迷宫

递归算法寻路

无有效路径

* 1. **项目主要功能实现**
     1. **随机迷宫的生成**

本次随机迷宫的生成选择使用prim算法。

通过对于问题的分析可知，可以将整张地图视作图，并且利用prim算法在此图上生成一个最小生成数。根据prim算法的过程，可以将整张图划分成已生成和未联通部分，每次迭代选择未联通的两部分进行连接，最后得到路径。剩下位置置为墙或者置空，整体生成随机迷宫。

* + 1. **迷宫寻路算法（DFS算法）**

本次迷宫寻路选择采用DFS算法。

通过对于项目功能要求的分析，选择采用DFS算法对迷宫进行寻路，可以不断递归寻找可行路径，而当遇到阻碍时则进行回溯，找到最后一个可行位置并再次出发。

1. **数据结构的撰写与使用**
   1. **双向链表类的使用**

本次项目使用链表作为基础。不同于之前程序使用的单向链表，考虑到程序本身存在递归等操作，故使用双向链表在牺牲空间复杂度的条件下降低时间复杂度，提高链表增删查改的速度，减少不必要的遍历和操作，优化用户体验。

本次链表类的设计共包含以下几个部分：

1. 设置linkNode结构存放结点内容，包括：

Type data;

linkNode<Type>\* next, \* prev;

1. 设置迭代器（游标）类方便实现链表类的自增，自减和运算符运算：
2. 默认构造函数

iterator()

1. //构造函数

iterator(\_iterator datas);

1. //读数据操作

Type operator\*()

1. //比较运算符

inline bool operator==(const iterator& v)const

inline bool operator<(const iterator& v)const

inline bool operator>(const iterator& v)const

inline bool operator<=(const iterator& v)const

inline bool operator>=(const iterator& v)const

inline bool operator!=(const iterator& v)const

1. //自增运算符

//前置自增

inline linkList<Type>::iterator& operator++()

//后置自增

linkList<Type>::iterator operator++(int)

1. //自减运算符

//前置自减

inline linkList<Type>::iterator& operator--()

//后置自减

linkList<Type>::iterator operator--(int)

1. //后面加上一个数

linkList<Type>::iterator operator+(int index)

1. //前面加上一个数

friend typename linkList<Type>::iterator operator+()

1. //后面减去一个数

linkList<Type>::iterator operator-(int index)

1. //自加

linkList<Type>::iterator& operator+=(int index)

1. //自减

linkList<Type>::iterator& operator-=(int index)

1. //计算两个迭代器之间的距离

int operator-(linkList<Type>::iterator i)

1. 设定链表类成员函数
2. //求解链表长度

inline int size()const;

1. //判断链表是否为空

inline bool Isempty()const;

1. //在尾部插入新数据

void push\_rear(Type data);

1. //在头部插入新数据

void push\_front(Type data);

1. //查找指定数据，返回迭代器

linkList<Type>::iterator find(const Type& data)const;

1. //删除指定迭代器对应的结点

Type remove(linkList<Type>::iterator index);

1. //删除尾结点

void erase\_rear();

1. //删除第一个结点

void erase\_front();

1. //清空链表

void clear();

1. //插入结点

bool insert(linkList<Type>::iterator index, Type i);

1. //返回指定位置的一系列函数

//起始位置迭代器

inline linkList<Type>::iterator begin();

inline const linkList<Type>::iterator begin()const;

//最后位置迭代器

inline linkList<Type>::iterator end();

inline const linkList<Type>::iterator end()const;

//重载等号运算符

linkList<Type>& operator=(const linkList<Type>& i);

* 1. **向量类的定义与使用**

本次项目实现向量作为**一**维数组的类版本，它与数组相似，其中的元素项是连续存储的，因此具有和数组相类似的特点：一方面体现在其中元素以严格的线性顺序进行存储，因此在重载取下标运算符[]的基础上，可以实现和数组类似的直接存取；另一方面则可以通过指针、泛型指针等方式实现对向量内容的直接存取，大大减少时间复杂度。

但它和数组不同的是：向量类的动态存储设定使得向量中存储元素的多少可以在运行中根据需要动态的增长或缩小；同时其中存储的数据类型可以依托类模板实现多样化，可以存放各种类型的数据。

本次向量类中设定数据大小，向量大小和类模板型元素等几个变量，为了方便实现对于向量类元素的增加，删除，查找和改正，增加迭代器类（与链表类中迭代器内容基本相似）。这一设定在充当泛型指针的同时也能够方便减小时间复杂度的堆积，实现运算符重载的自增，自减等操作。

1. 向量类中的数据内容：
2. //数据大小

int \_size;

1. //当前容量

int \_capacity;

1. //数据

Type\* \_elements;

1. 迭代器中实现内容和操作与链表类基本一致，故不再赘述。
2. 向量类成员函数的实现与封装：
3. //向量大小

inline int size()const;

1. //判断向量是否为空

inline bool Isempty()const;

1. //尾部插入数据

void push\_back(const Type data);

1. //删除数据

Type pop\_back();

1. //在指定迭代器前插入数据

bool insert(const Vector<Type>::iterator place, Type item);

1. //删除指定迭代器的数据

bool erase(const Vector<Type>::iterator place);

1. //清空数据

void clear();

1. //统一赋值

void fillData(const Type data);

1. //返回起始位置迭代器

inline Vector<Type>::iterator begin();

inline const Vector<Type>::iterator begin()const;

1. //返回最后位置迭代器

inline Vector<Type>::iterator end();

inline const Vector<Type>::iterator end()const;

1. //获取最后一个元素

inline Vector<Type>::iterator last();

inline const Vector<Type>::iterator last()const;

1. //获取第一个数据

Type& front()const

//运算符重载

1. //取下标运算符的重载

inline Type& operator[](int i);

1. //等号运算符

Vector<Type>& operator=(const Vector<Type>& v);

* 1. **成对类的使用**

考虑到本次项目的地图多以坐标形式表示，也需要客户手动输入起点和终点的坐标来实现令用户满意的寻路，故需要多次进行成对化的操作。在此基础上建立pair类，方便函数的调用和函数形参的传入，也能够使各对坐标更加紧凑具体，不易出错。

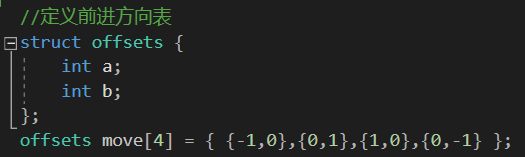
1. 成对类内容：
2. T1 first;
3. T2 second;
4. 成对类成员函数的实现与封装：

主要体现在重载=/==/!=三种运算符上。

1. Pair& operator=(const Pair& rhs)
2. inline bool operator==(const Pair<T1, T2>& rhs)
3. inline bool operator!=(const Pair<T1, T2>& rhs)
4. **算法与功能函数实现**
   1. **寻路算法函数seekpath()的实现**

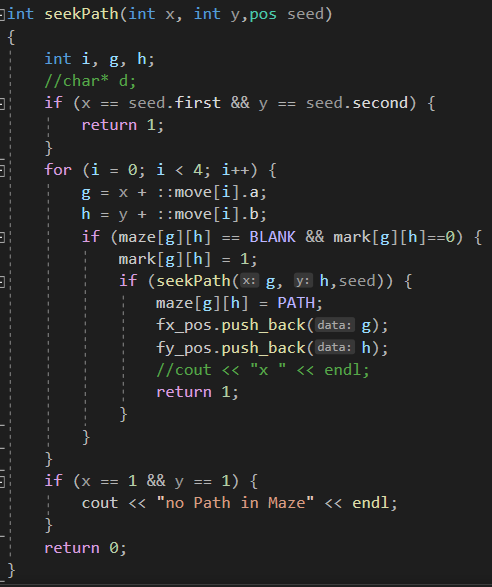
本次项目中寻路函数采用DFS算法的实现思路，并进一步修改算法使得算法更加贴合本次项目的寻路目的。

其中设定前进方向表，简化并集合搜索的“上”“下”“左”“右”方向，方便算法处理。



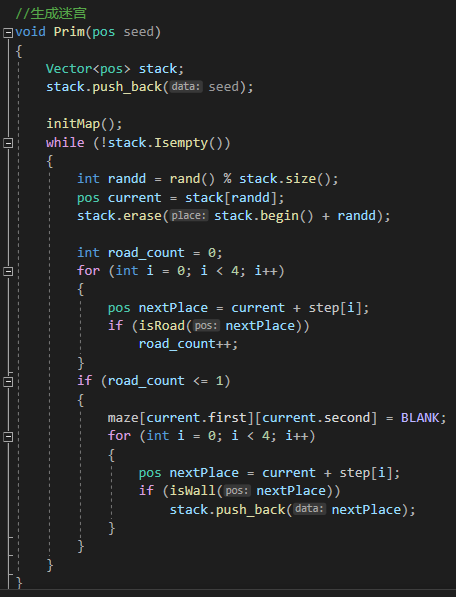
函数使用递归思路，将目标终点定义为成对类pos，并作为形参seed传入。

1. 如果本次搜寻到的位置（x,y）和（seed.first,seed.second）对应相等，则说明搜索成功，返回；
2. 否则利用for语句循环搜索方向表中的上下左右四个方向，令寻点坐标依照方向表加减变化，如果寻得maze为空且mark为0（即此点未被访问过）的点，则将此点mark置1，继续递归。
3. 递归完成后，若成功，则退栈并逐个更新maze点为PATH（路径可以通过），并且将此点横纵坐标传入向量空间中等待后续输出。
4. 若寻路失败，则打印“no Path in Maze”并返回失败值。



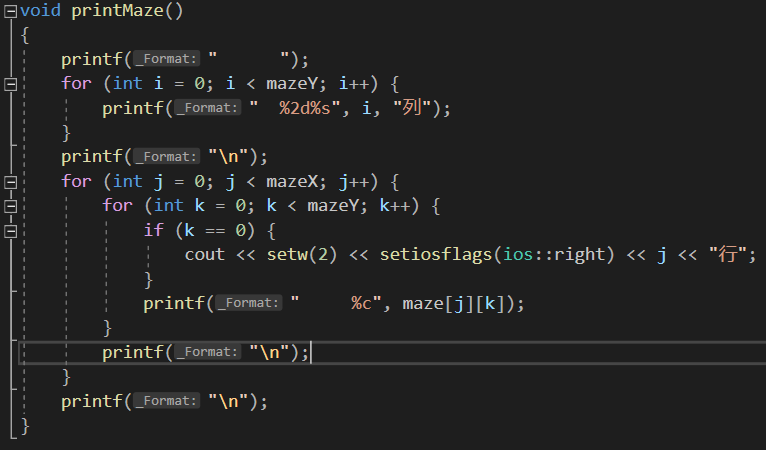
* 1. **随机地图生成算法prim（）的实现**

Prim算法的实现过程即最小生成树的构造。将整张图看作最小生成树的点，在其上构造最小生成树，并将不需要的点赋予另值。



* 1. **其余辅助函数的实现**
     1. **地图打印函数printMaze()函数的实现**

地图打印函数利用C++语言输入输出格式化控制符<iomanip>中的setw作为依托，先打印第一列的指示数字。此后逐行打印地图的行和地图的内容。



* + 1. **用户输入内容的政务判断函数实现**

报错“行列有误”

行列小于等于零

case

switch

输入行列数

报错“行列过大”

行列大于等于100

报错“输入错误”

cin.fail()

输入起点终点

报错“起点有误”

Start是否可达

报错“终点有误”

End是否可达

* 1. **项目流程图图解**

结束

输入迷宫行列lie列

合法？

否

是

Prim算法生成迷宫

打印迷宫

输入起终点坐标列

合法？

否

是

寻路

存在路径？

是

否

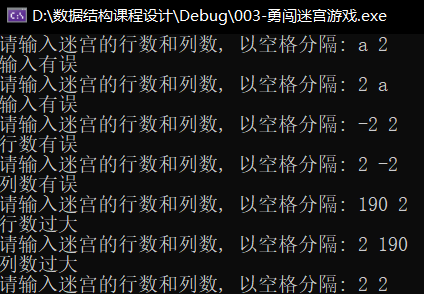
打印路径

返回错误

开始

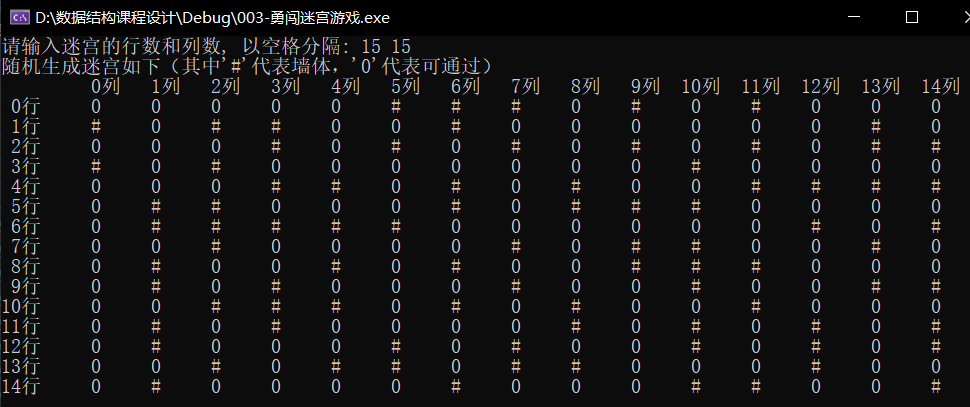
1. **项目测试**
   1. **输入输出行列测试**

测试用例：a 2 / 2 a / -2 2 / 2 -2 / 190 2 / 2 190 / 2 2

****

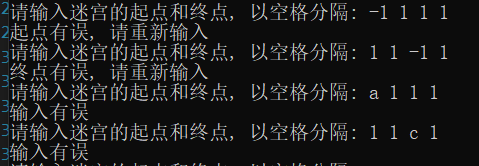
* 1. **随机生成迷宫算法测试**

测试用例：15 15



* 1. **输入输出起点终点测试**

测试用例：-1 1 1 1 / 1 1 -1 1 / a 1 1 1 / 1 1 c 1

****

* 1. **迷宫寻路算法和路径打印测试**

测试用迷宫： 15 \* 15

