着色(多叉路口红绿灯管理)问题 作业

这是这学期第一次做编程的作业，其实说起来也挺作死的，把本来可以在写程序时可以确定的题目的参数在运行时再输入。虽然提高了可扩展性，但却大大提高了整体代码的复杂度。

1、最终程序的结构是这样的：

首先提示用户输入路的条数，从而动态分配两个用来计算各个走法关系的两个布尔型数组。再让用户输入，初始化这两个数组来表征每条路是否允许进路口和出路口。

计算出路径的总数，初始化一个路径结构体数组

possi=no\_in\*no\_out-two\_ways; //int possi 为了方便使用，是在全局声明的

this->routes=new route[possi];

然后遍历每个路口对，建立每种可行的走法和这个数组中元素的对应关系。

然后用一系列路径，允许出入的状态和其之间的关系初始化一个二维的布尔型数组（c/c++不支持直接动态分配行、列均在运行时确定的二维数组的内存空间，为此写了一个模板类，再继承）。这之后就和示例程序最开始的状态一样了。

然后把每个加入分组的走法按顺序看它是否和已有的分组兼容（遍历所有走法，和所有这个组的走法比较兼容性），若兼容就加入这个组，否则比较其他的组，最后再创建新组。（和PPT上讨论的算法思想上一致）。

最后把结果输出，看用户还要不要再算别的情形，不的话就退出。

2、一些遇到的问题和解决方法

第一件事情就是怎么表征各个分组和走法，和怎么计算走法间的关系。最开始有好几天不得要领，总是因为路口在各个路径关系中的重要意义，在想路口的表示方式，怎么动态分配。甚至在想把走法写成一个路口的布尔型二维数组，用true和false表征可不可以，然后走法的关系就变成了更麻烦的四维数组。几天后才想明白真正要讨论的是两个路口组成的路径，路口只是一个表示方式。

在这个过程中找了很多资料，学了C++怎么写模板类和模板函数（模板和实例啥的），然后找了很多资料，终于成功地用new和delete写成了可以动态分配的任意大小，空间连续分布的二维数组（方便随机存取？疑惑，但毕竟连续似乎比较好）。虽然代码看起来很简单，就是生成一个row\*col大小的数组再生成一个row大小的指针数组，用一个指针指向它。最后把它封装在了一个类里面，调好了析构函数，避免内存泄漏。

后来想明白了，就把之前写的这些代码扔到了一个垃圾箱文件夹里，基本只重用了之前讲的那个模板类，用string来在c++ 完成自动的realloc操作倒也是那时想到的，不过一直没有付诸实践。

然后决定用一个结构体表示路径然后在确定路况（允不允许）以后生成它，本来还想创造一个指针和目前位置的结构体成员，但最后发现是不必要的。

然后就找到了之前说的用户输入->初始化bool \*in \*out -> 输入路口情况到in和out -> 初始化 route –> 找出各个route的过程。

然后为了各个cpp用同一个头文件，共享声明的变量，而不至于到处写 ”extern” ，定义了一个宏来解决。

后来把这些东西封装在了一个Road类里面，把和用户交互的输入输出封装在了构造函数里。

后来学了一点派生和继承的东西，然后就写了一个relations类，继承Array\_2D<bool>，然后补上了各个可能会用到的函数。因为二维数组需要之前的输入和计算才能初始化，只好把之前的封装在单独的一个类里，然后用一个指针完成数据的交互(Road\* This)，在构造函数中完成初始化。

在这个找关系过程中把兼容的路的走法分了几类，有路线完全“平行”的，出/入相同路口的，和有一个路口出入相反，但满足靠右行就不会撞的。这个“靠右行”打破了逆时针输入和顺时针计算的对称性，所以就定下了顺时针的要求（不然 “left\_dist” 和 “right\_dist” 会反过来）。然后把它们写了几个函数完成这个工作。然后两个循环算出关系矩阵。

然后就是对分组的计算。因为一开始路径没有写分组的标识，就又写了一个int型的数组group，每位对应route中对应位的分组。然后就用一个个分组的方式（前面已经提到）把各个走法分组得到结果。

最后补充了一些和用户交互的内容，尽量把东西封装起来，以后好加到我给cmd写的shell里。（用extern “C” + LoadLibrary时避免命名重复、冲突啥的）

把程序可以多次运行做出来以后出现了同样的东西输入两次，分组不同的问题（但关系表是对的），经过排查发现是group这个数组最开始没有初始化。操作系统第一次把一段内存分配给程序时可能出于安全性的考虑是初始化为0的，但再次分配不会这么做。所以补上了初始化的

new int[Crossing::possi] ();

中的括号，然后就解决了

3、一些方法的选择和可以改进的地方

首先，在最后求解的时候每看一个走法和一个组的关系时，都要遍历所有走法，找出这个组的成员。这是效率极低的。可以考虑增加一些指针数组表示各个组，然后用指针直接访问这个组的元素。

但是这可能需要一个不定长的指针数组来完成。一般的数组并不能完成这个任务，提前分配最多可能的内存则会造成很大的内存浪费。

一个C++中可能的方法是用一个c++的字符串string来完成，然后用substr()成员函数来访问其中储存的指针。（c++的string应该是把realloc函数优化后封装在里面了，因为c++的new和delete分配的空间不支持调整大小，所以只好这么做了）。

但是这样做的指针数组很难维护，甚至对应的字符数会和操作系统的位数有关，要分别再去写，极难维护，所以目前还没有去做这项工作。

还有就是其实在内存空间中实际存在的二维数组实际上不一定是必须的。我的程序中为了生成这个数组写了一个函数来计算这些元素的值，而实际计算中完全可以在需要这个数组的内容时调用这个函数来进行计算。这样虽然需要多计算很多次，但是可以在n（走法数）在极限的情况下保证内存不会不足（内存只需要o(n)，而不是o(n^2)）。

但是因为时间原因，并没有再去做效率的测试。

4、一些可能存在的问题

其实我们说这种分配方式是组数最少的，并没有给出证明。如果有几个元素和其他的元素几乎全都不兼容，但它们自己之间兼容很好，我们可能会因为它们而创建很多有几个别的元素，和一个这种元素的组，增加分配的组数。

比如下面的关系表：

A B C D E F G H I J

A 0 1 1 1 1 1 0 0 0 0

B 1 0 1 1 1 0 1 0 0 0

C 1 1 0 1 1 0 0 1 0 0

D 1 1 1 0 1 0 0 0 1 0

E 1 1 1 1 0 0 0 0 0 1

F 1 0 0 0 0 0 1 1 1 1

G 0 1 0 0 0 1 0 1 1 1

H 0 0 1 0 0 1 1 0 1 1

I 0 0 0 1 0 1 1 1 0 1

J 0 0 0 0 1 1 1 1 1 0

这种排列方式用这种算法跑出来可以得到结果：

第一组： A B C D E

第二组： F G H I J

但如果重新排序一下，变成：

A F B G C H D I E J

A 0 1 1 0 1 0 1 0 1 0

F 1 0 0 1 0 1 0 1 0 1

B 1 0 0 1 1 0 1 0 1 0

G 0 1 1 0 0 1 0 1 0 1

C 1 0 1 0 0 1 1 0 1 0

H 0 1 0 1 1 0 0 1 0 1

D 1 0 1 0 1 0 0 1 1 0

I 0 1 0 1 0 1 1 0 0 1

E 1 0 1 0 1 0 1 0 0 1

J 0 1 0 1 0 1 0 1 1 0

把这个按之前的算法计算，则会得到结果

第一组： A F

第二组： B G

第三组： C H

第四组： D I

第五组： E J

第六组： F K

明显不一定解出的不一定是最优解。也就是说尽量把新元素放入已有组并不能保证组数一定是最小的。“按顺序来”这件事已经破坏了关于各个元素顺序的对称性，导致结果和计算顺序有关。

在这个过马路的问题，所求解的情形下，这件事不一定不存在。

所以可能需要找更好的算法来处理这个问题

（如果这真的和地图着色问题相同，原则上可以按组数<=4作一个大致的定性判断）

5、总结

这次作业真的做了很久，花了很多精力。其实主要遇到的难题都是自己作的，其实不去允许算其它情形应该也能很好地完成作业。加上了前面的内容，从代码量也能看出来和后面比多了几倍，而且用到了很多以前从来没用过，只是听说过的方法和概念更是增加了查资料的量。

不过我觉得还是挺值的，毕竟现在会了不少东西，毕竟我一直是想干什么就去做，经常不计成本的（虽然也有不想做，拖到最后的情况。。。）。希望以后干这些事能顺利些。