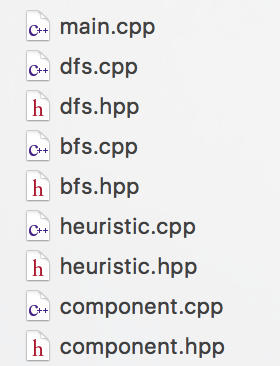
实验报告

1. 实验环境

MacOS, C++

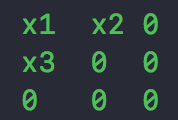
1. 代码结构说明

代码结构如下图所示, dfs.cpp, bfs.cpp, heuristic.cpp三个文件分别实现了三种策略, component.cpp中实现了实验需要的组件, main.cpp中测试了三种方法的结果。

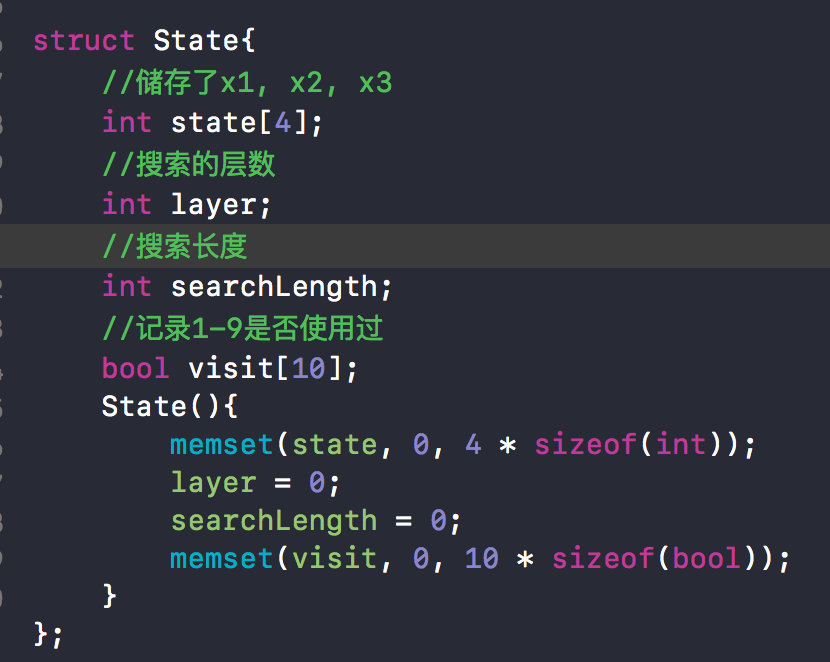


1. 问题求解

通过对题目的分析，我发现其实只需要确定九宫格中的三个值，就可以确定整个九宫格，所以我们只需要搜索这三个值即可。在实验中，这三个值我选定为下图所示的x1, x2, x3。最终的搜索路径也可以用这三个值来表示。



然后我定义了一个结构体来表示每个搜索状态：



对于具体代码来说，三个策略的框架完全一样，不同的是dfs使用栈，bfs使用队列，heuristic使用最小优先队列。代码意思的说明可以参考我的注释。

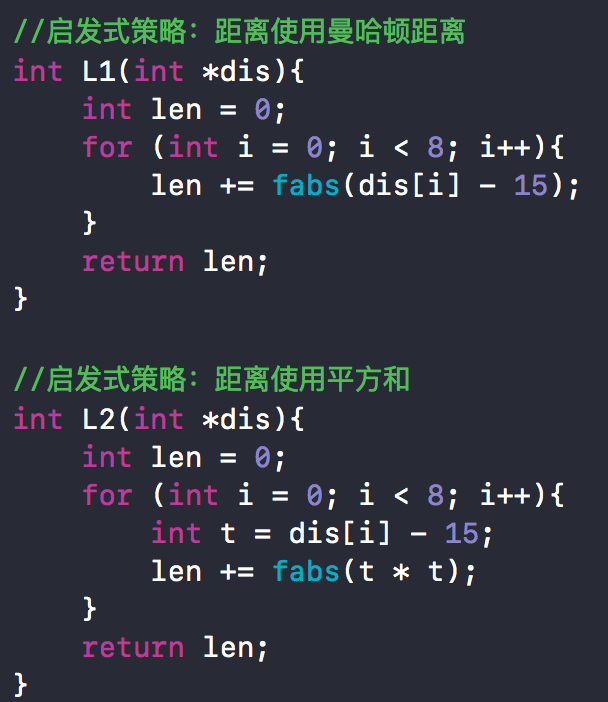
启发式搜索设计：

我定义了两个距离：

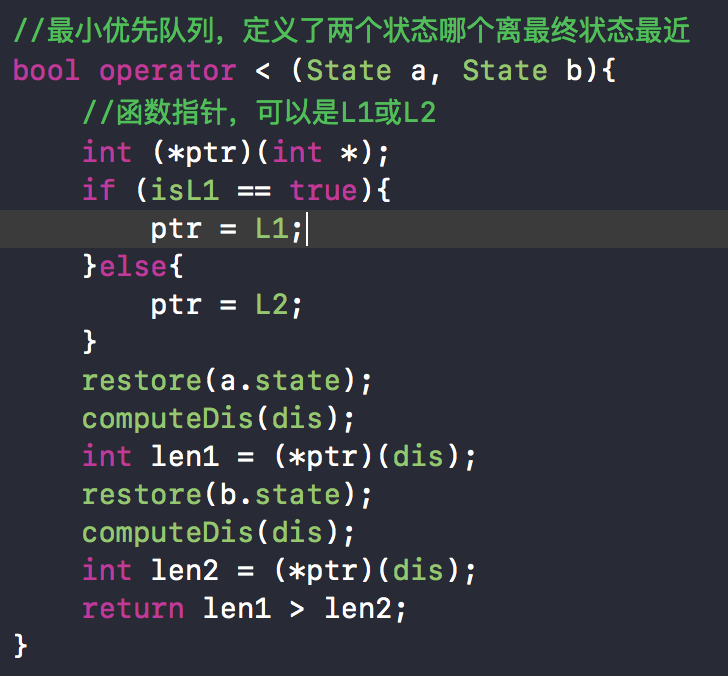
dis是一个八元组

(第一行和，第二行和，第三行和，第一列和，第二列和，第三列和，左上到右下，右上到左下)

L1距离是与U = (15, 15, 15, 15, 15, 15, 15, 15)的曼哈顿距离，L2距离是与U的距离的平方和。



然后为了使用最小优先队列，我重载了< 操作符



当x1, x2, x3均不为0时，restore函数会将map还原，当三个值中有任意一个为零时，restore函数不会将map还原，即其他值依然为0。然后计算得出当前dis, 通过L1或L2计算当前距离。最小优先队列以这个距离为指标排列。

1. 实验结果

具体搜索路径和所有结果记录在result.txt中。

* dfs:

最快搜索距离：114

* bfs

最快搜索距离：179

* heuristic

L1: 最快搜索距离：15

L2: 最快搜索距离：15

1. 效率分析

由结果可以看出heuristic的两种距离定义得到的最快路径长度都是15， dfs 114次之，bfs179最慢。