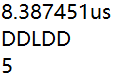
**Lab01 24数码 实验报告**

**项子扬 PB16001768**

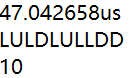
1. **实验结果**

一、Ah1

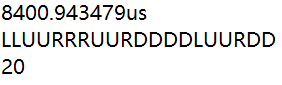
1.



2.



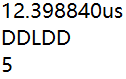
3.



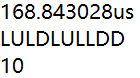
4以及之后无法得出结果。

二、Ah2

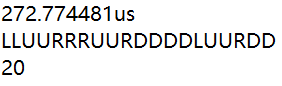
1.



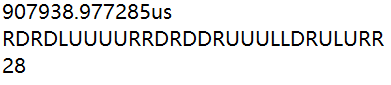
2.



3.



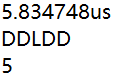
4.



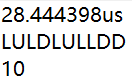
5以及之后无法得出结果。

三、IDAh1

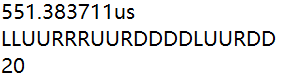
1.



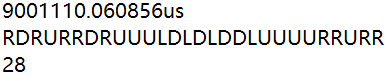
2.



3.



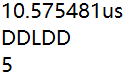
4.



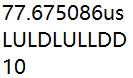
5以及之后无法得出结果

四、IDAh2

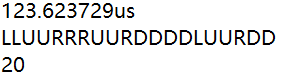
1.



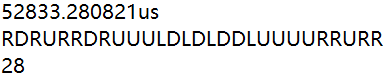
2.



3.



4.



5以及之后无法得出结果

**2.算法时空复杂度**

一、Ah1

时间复杂度大于O(n^3) 空间呈指数型增长。

二、Ah2

时间复杂度在临近计算性能极限时近似为O(n^2)，临近极限时无法估计。

空间呈指数型增长。

三、IDAh1

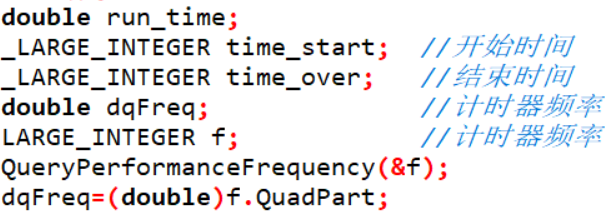
比Ah1好，比Ah2差。

四、IDAh2

性能在四者中最好。

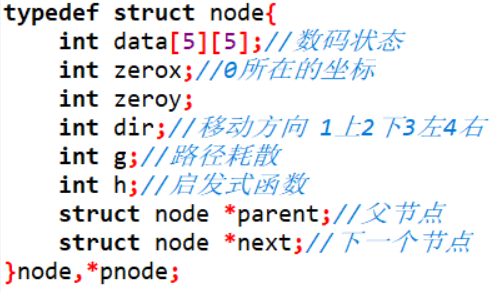
**3.关键代码说明**

一、Ah1、Ah2

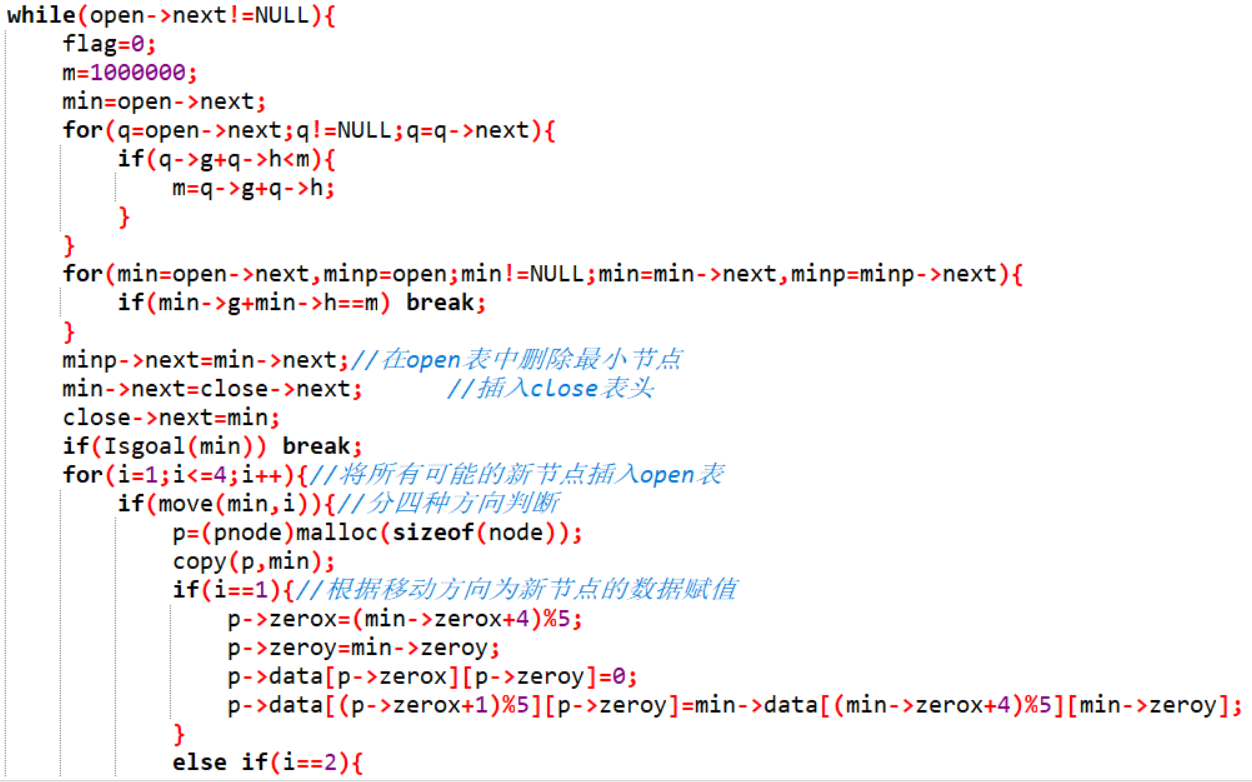


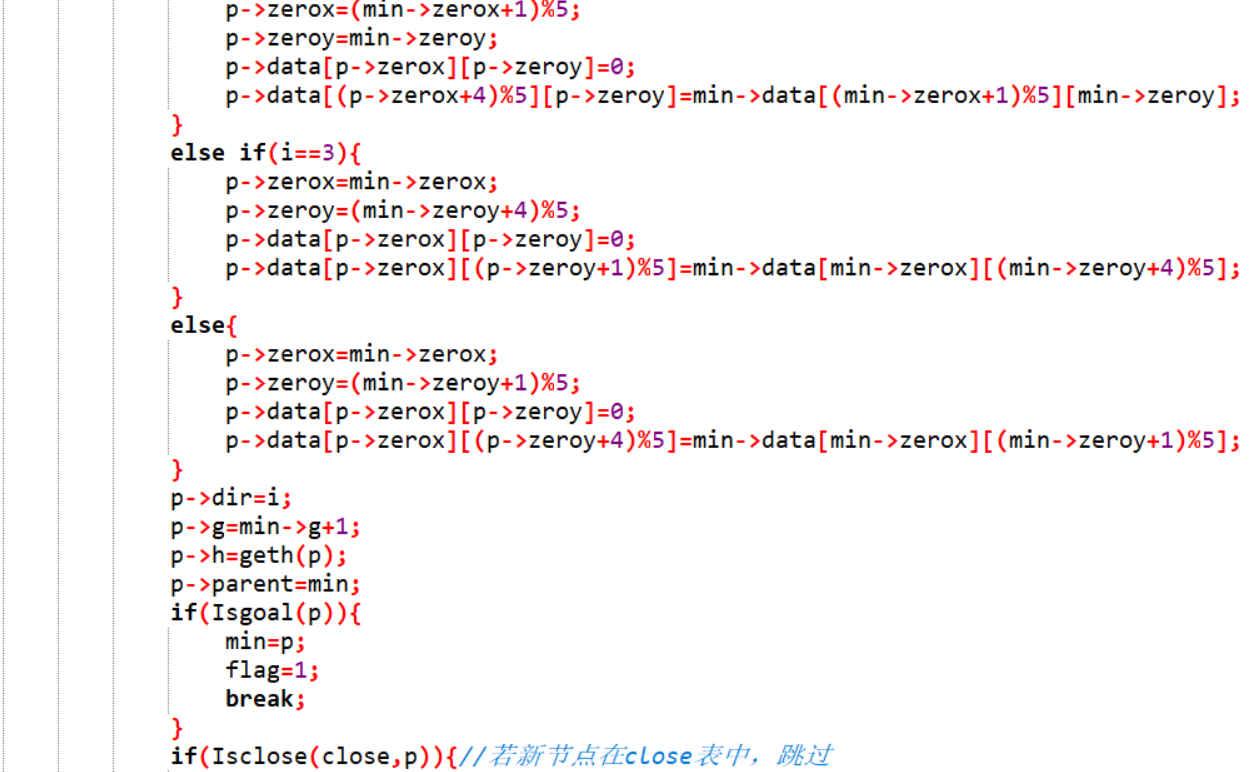


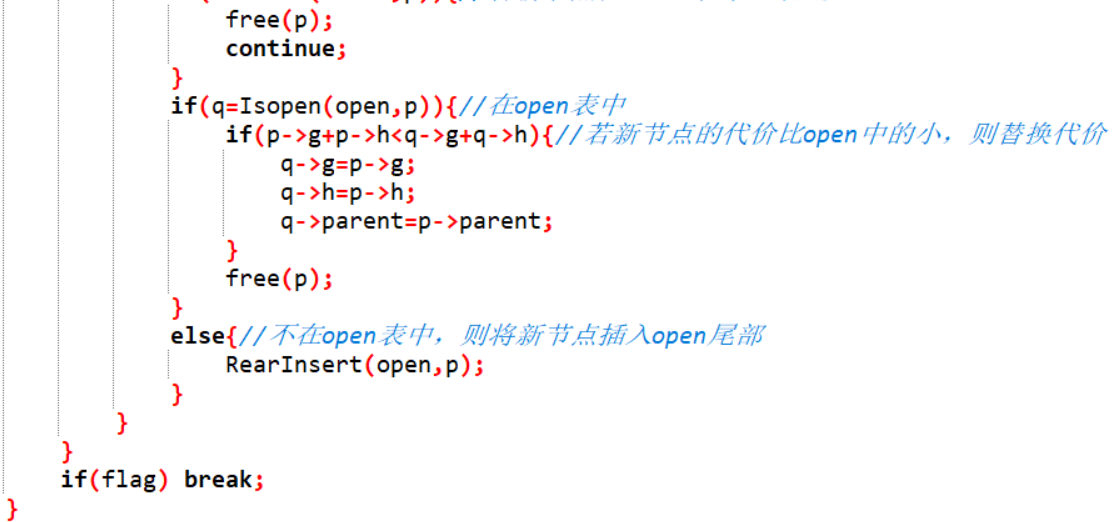
此部分用于程序计时。



此为节点的结构体，data数组用于存储数码状态，zerox和zeroy记录0的位置，g和h是节点的路径耗散和启发式函数，加起来就是估值f，dir记录上一个节点的移动方向，parent指向上一个将其展开的节点，dir与parent一起可以回溯出搜索的具体路径；next指针用于形成open表或close表的链表结构。



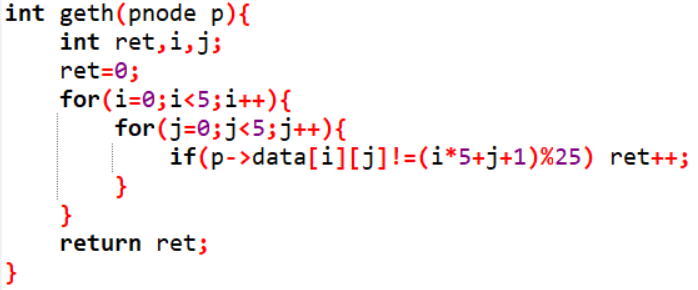


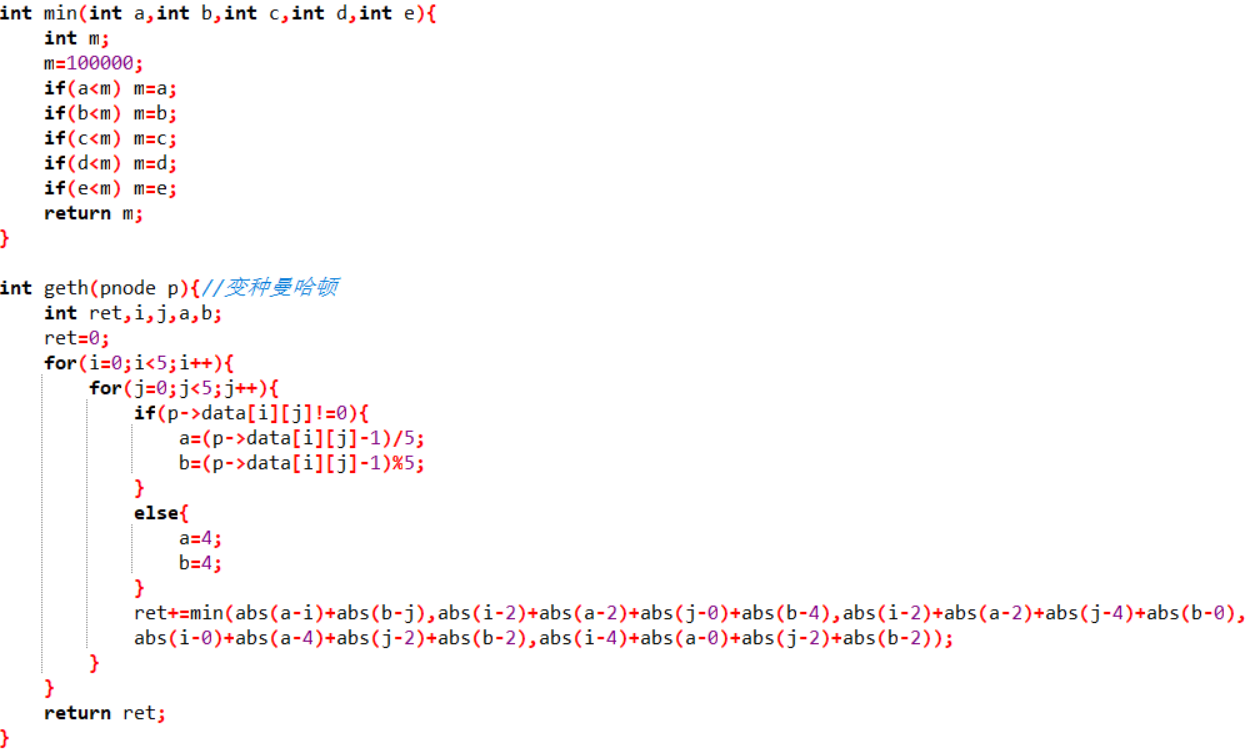


此部分为循环迭代部分，其中采用了顺序遍历的方法找到open表中的估值最小节点，在对节点做展开时先用一个move函数判断移动方向是否可行，若可行，则进行如下判断：

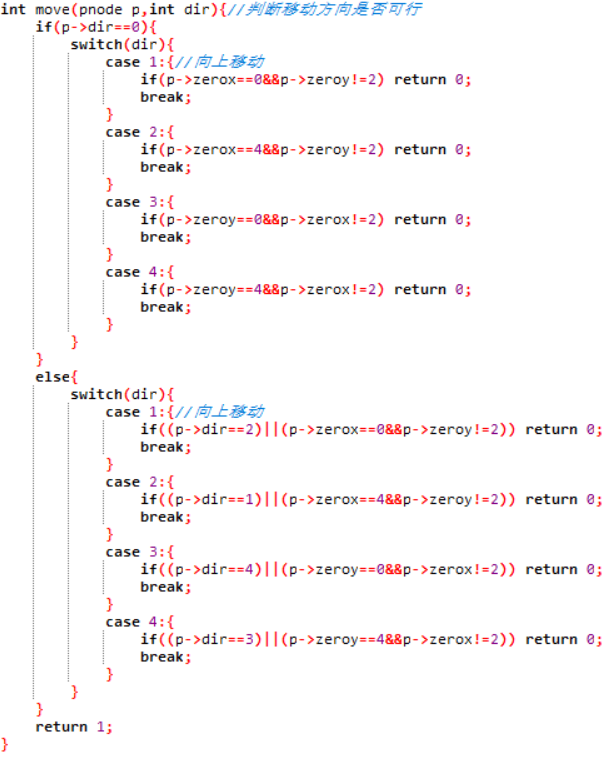
若新节点就是目标状态，直接返回；若新节点在close表中，则舍弃；若新节点在open表中，则比较两者的估值，若新节点的估值更小，则将open表中原来的节点相关信息改为新节点的值（如父节点），反之则丢弃；若新节点不在open表中，则将新节点插入open表的尾部。

其中用到的重要函数：





此部分为启发式函数，在Ah1中为错位的棋子数，Ah2中采用了变种曼哈顿距离的计算，即在曼哈顿距离的计算方法上额外考虑四条路线，在5者中取最小值即为两点之间的最短曼哈顿距离。



此函数用于判断当前节点是否能朝所选方向移动（1上2下3左4右），若节点不是初始节点，则不能朝反方向移动。

二、IDAh1、IDAh2

与A\*算法不同的地方在于不在需要查重，每次迭代设置一个阈值limit，大于limit的节点不予展开，并根据这些不展开的节点的估值得出下一次迭代的新阈值。每次迭代开始时，都从最初的节点开始重新展开，直到得到目标状态。

函数主体部分：





其余部分函数与A\*大致相同，不再赘述。

**4.使用说明以及如何提升速度**

使用说明：在源代码的同目录下添加input.txt文件，由于本次实验中的目标状态都相同，故已在代码中初始化，无需再添加target.txt文件。Input文件中为5\*5矩阵内容，直接编译运行源代码，会在同一目录下生成对应的solution.txt，打开即为运行结果。

如何提升运行速度：

以某种规律对数码状态进行分类后排序，在表中查找时采用二分查找，可以提高程序运行效率。