|  |
| --- |
| C:\Users\Public\Pictures\1000861110.jpg  **教师：徐建良**  **评分：** |
| 人工智能Projecet1项目研究报告  N-Puzzle & N-Sliding |
| |  |  | | --- | --- | | **小组:** | **Alphamiao** | | **成员：** | **崔泽宇 刘魏鑫宁 任峥 王刚 吴敬宇**  **张寅丁** | | **日期:** | **2019.4.23** | |
| 中国海洋大学  信息科学与工程学院计算机系 |

# 目录:

目录

[**一、 问题概述：** **3**](#_Toc29480_WPSOffice_Level1)

[1. N-Puzzle问题： 3](#_Toc22226_WPSOffice_Level2)

[2. N-Sliding问题： 3](#_Toc26682_WPSOffice_Level2)

[**二、 启发式算法A\*的简述** **3**](#_Toc22226_WPSOffice_Level1)

[1. 算法描述 （伪代码）f（n） = g（n）+ h（n） 3](#_Toc11165_WPSOffice_Level2)

[2. A\*的优点 4](#_Toc30612_WPSOffice_Level2)

[3. A\*的缺点 5](#_Toc6781_WPSOffice_Level2)

[**三、 启发式搜索IDA\*算法的简述** **5**](#_Toc26682_WPSOffice_Level1)

[1. 算法描述（伪代码描述） 5](#_Toc456_WPSOffice_Level2)

[2. 优点 6](#_Toc28295_WPSOffice_Level2)

[3. 缺点 6](#_Toc18518_WPSOffice_Level2)

[**四、 N-Puzzle的三个阶段的算法及改进** **6**](#_Toc11165_WPSOffice_Level1)

[1. 第一阶段 6](#_Toc756_WPSOffice_Level2)

[2. 第二阶段 7](#_Toc17000_WPSOffice_Level2)

[3. 第三阶段 7](#_Toc8145_WPSOffice_Level2)

[**五、 N-Puzzle三个阶段的程序效果对比** **8**](#_Toc30612_WPSOffice_Level1)

[1. 图表： 8](#_Toc14139_WPSOffice_Level2)

[2. 总结 8](#_Toc6464_WPSOffice_Level2)

[**六、 N-Sliding三个阶段的算法改进** **9**](#_Toc6781_WPSOffice_Level1)

[1. 第一阶段 9](#_Toc5332_WPSOffice_Level2)

[2. 第二阶段 9](#_Toc11739_WPSOffice_Level2)

[3. 第三阶段 10](#_Toc11632_WPSOffice_Level2)

[**七、 N-Sliding三个阶段的程序效果对比** **11**](#_Toc456_WPSOffice_Level1)

[1. 图表 11](#_Toc17640_WPSOffice_Level2)

[2. 结论 11](#_Toc6471_WPSOffice_Level2)

[**八、 结合N-Puzzle和N-Sliding分析各个算法的优缺点** **12**](#_Toc28295_WPSOffice_Level1)

[A. A\*的优缺点 12](#_Toc29576_WPSOffice_Level2)

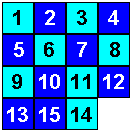
[B. IDA\*的优缺点 12](#_Toc10141_WPSOffice_Level2)

[**九、 针对N-Puzzle问题比较“错位将牌数”和“曼哈顿距离”两个方法的性能** **12**](#_Toc18518_WPSOffice_Level1)

[**十、 N-Puzzle和N-Sliding可视化的实现** **13**](#_Toc756_WPSOffice_Level1)

1. 问题概述：
2. N-Puzzle问题：

N-puzzle历史：是由1878年 ，美国最伟大的拼图专家萨姆·洛伊德发明的，这是十五年前由纽约压花公司制造和销售的“15拼图”的变体。这个拼图由15个编号的方块组成，可以在方形盒子里滑动，大小足以包含16个。这些碎片应该随意放在盒子里，你应该按升序对碎片进行分类，而不是将碎片从盒子中取出（因此唯一允许的是滑动碎片）。

此后人们在此问题的基础上依然深入讨论，从15puzzle的解决方案，到之后的24puzzle问题，一直推到现在人们熟知的N（n²-1）Puzzle问题来。

1. Puzzle问题：（也称推盘游戏问题）是经典的人工智能计算问题，人们往往用N-Puzzle问题的求解为例子来说明各种搜索策略。所谓的N-Puzzle问题指的是在K\*K的方格棋盘上（这里），将数字1，2，3，...，n以任意的次序放入棋盘的各个方格中，留出任意一个空格。当n = 8和n = 15时，分别是经典的重排九宫问题和十五迷问题。

对于N-Puzzle的计算方法已经出现包括经典和现代的多种算法途径，包

括深度优先搜索法（DFS），宽度优先搜索法(BFS),A\*算法、遗传算法等等。 随着对于N-Puzzle问题的深入研究以及计算技术的发展，出现了一些新兴 的计算方法（如量子计算、生物计算、光计算等），相信未来N-Puzzle问 题一定会有更好的解决办法。

本组根据老师给出的A\*算法进行优化改进，使用了IDA\*算法和disjoint pattern database方法对N-Puzzle进行了研究，并取得了较好的结果。

1. N-Sliding问题：
2. Sliding问题(也称滑动积木块游戏)和N-Puzzle一样，也是一个经典的NP问题，游戏规则由于我们非常熟悉就不再多说了。游戏的最终目标是使所有的白将牌都处在黑将牌的左边（称为目标格局），而随着将牌的移动，我们同样也会得到一些中间格局和后继格局。我们要如何移动白将牌才能让我们求出最优解，这是我们一直要研究的问题，我们基于一开始老师给出的A\*算法，不断进行优化，终于可以较快的解出九阶的N-Sliding问题。
3. 启发式算法A\*的简述
4. 算法描述 （伪代码）f（n） = g（n）+ h（n）

A算法是一种典型的启发式索索算法。其基本思想是：定义一个函数f，对当前的搜索状态进行评估，找出一个最有希望的结点进行扩展。评价函数的形式为f(n) = g(n) + h(n)，n为被评价的结点。

g\*(n)表示从初结点s到结点n的最短路径的耗散值，h\*(n)表示从结点n到目标结点g的最短路径的耗散值，f\*(n)=g\*(n)+h\*(n)表示从初始点s经过结点n到目标结点g的最短路径的耗散值。而F(n),H(n),G(n)分别表示f\*(n)，h\*(n)，g\*(n)3个函数值的估计值。

A算法每次根据f(n)值的大小对OPEN表中的元素进行排序，f值小的放在前面，f值大的结点放在OPEN表的后面，每次扩展结点时总是选择当前f值最小的结点优先扩展。

下面是A\*的算法过程

1 把起点加入open list

2 重复如下过程

A 遍历open list，查找F值最小的节点，把它作为当前要处理的节点

B 把这个节点移动到close list

C 对当前方格的8个相邻方格的每一个方格：

* 如果它是不可抵达的或者它在close list中，忽略它，否则，做如下操作：
* 如果它不在open list中，把他加入到open list，并且把当前方格设置为它的父亲，记录该方格的F，G，H值。
* 如果它已经在open list中，检查这条路径（即经由当前方格到达它那里）是否更好，用G值作为参考。更小的G值表示这是更好的路径。如果是这样，把它的父亲设置为当前方格，并重新计算它的G和F值。如果你的open list是按F值排序的话，改变后你可能需要重新排序。

D 停止，当你：

* 把重点加入到了open list中，此时路径已经找到了，或者
* 查找终点失败，并且open list是空的，此时没有路径

3 保存路径，从终点开始，每个方格沿着父节点移动至起点，这就是你 的路径。

1. A\*的优点

A\*成功的秘决在于，它把Dijkstra算法（靠近初始点的结点）和BFS算法（靠近目标点的结点）的信息块结合起来。在讨论A\*的标准术语中，g(n)表示从初始结点到任意结点n的代价，h(n)表示从结点n到目标点的启发式评估代价（heuristic estimated cost）。在上图中，yellow(h)表示远离目标的结点而teal(g)表示远离初始点的结点。当从初始点向目标点移动时，A\*权衡这两者。每次进行主循环时，它检查f(n)最小的结点n，其中f(n) = g(n) + h(n)。

1. A\*的缺点

[A\*算法](https://www.baidu.com/s?wd=A*%E7%AE%97%E6%B3%95&tn=SE_PcZhidaonwhc_ngpagmjz&rsv_dl=gh_pc_zhidao)通过比较当前路径栅格的8个邻居的启发式函数值F来逐步确定下一个路径栅格，那么如果存在多个最小值时[A\*算法](https://www.baidu.com/s?wd=A*%E7%AE%97%E6%B3%95&tn=SE_PcZhidaonwhc_ngpagmjz&rsv_dl=gh_pc_zhidao)不能保证搜索的路径最优。并且A\*算法一般要使用大量的空间用于存储已搜索过的中间状态，防止重复搜索。

1. 启发式搜索IDA\*算法的简述
2. 算法描述（伪代码描述）

IDA\*算法其实就是在A\*算法上使用了迭代加深算法，在搜索过程中采用了估值函数，以减少不必要的的搜索，降低了时间复杂度。

下面是IDA\*的算法伪代码：

is\_success()//检查结果的合法性

h()//估价函数，用于剪枝

dfs()//遍历

{

is\_success();

h();//利用h()判断是否应该剪枝

for()//递归枚举

{

保存目前状态。

递归枚举深入。

恢复状态。

}

}

solve()//迭代加深基本框架

{

if(is\_success)//有时可以不用此判断。

int max\_ans;//可能的最深

for(maxd = 0;;maxd++)

{ if(dfs(0,maxd)) return maxd; }

return max\_ans;

}

1. 优点

IDA\*算法在A\*算法上使用了回溯的方法，不需要保存中间状态，大大节省了空间。

1. 缺点

由于IDA\*在每次depth变大的时候都要从头再次搜索，这种操作虽然保证了最优解，但是重复的搜索使得程序有小概率比A\*慢。

1. N-Puzzle的三个阶段的算法及改进
2. 第一阶段
3. 首先对于Zobrist的优化。一开始在老师给出的Zobrist的值需要一共异或九次才能算出Zobrist值，我们每次移动一个方格就要重新计算一遍Zobrist值，由于异或九次，所以综合计算下来还是很大的。

通过观察我们发现，移动过后，其实Zobrist值的改变其实主要与移动的那一个方块的改变有关，其余方块对于Zobrist值的贡献不变，所以我们对算法进行了改进，每次只要计算移动的方块对于Zobrist值的增减，然后用旧的Zobrist值减去增减值就可以获取新的正确的Zobrist值，在使用了这个方法后，A\*算法的速度有着明显的加快。

1. 老师在Explored和Fringe的文件中，使用了ArrayList来存取评估的函数值，这样的话，我们每次存取的时候需要判断那个是最大的并且取出来然后进行更新，这样不免有点浪费时间。

经过修改，我们采用了优先级队列priority\_queue来存取每次的评估值，这样评估值就会按从大到小的顺序排列，我们需要的当前最优解就会出现在队列开头，那么我们直接pop一个排头即可，节省了大量的时间，使得程序更加的效率。

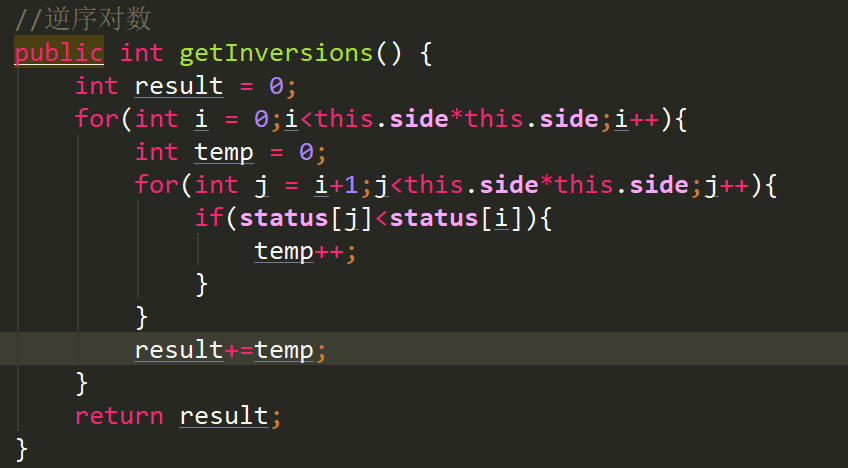
1. 老师在原先的代码中，使用了ArrayList来存取每一个Node节点的Zobrist值，这样在遍历查找是否重复的时候就会变得比较慢，我们直接使用HashMap存取它的Zobrist值，这样在遍历是否出现这个Zobrist值的时候，查询时间就会很快，从而加快程序效率。
2. 第二阶段
3. 由于我们在第二阶段都要使用老师的框架，所以改动相对较少，在IDA\*的基础上，我们优化了下一步的选择的方向，相当于变相的降低了循环的概率和陷入局部最优解的概率。

我们使用一个数来记住从A到B状态的空白块移动方向，并且在由B到

C（B的下一个状态）的时候，禁止向上一个移动方向的反方向移动，这样就 减少了重复回到上一步然后判断是否重复出现所耗费的时间，这样也会加快 程序的速度。

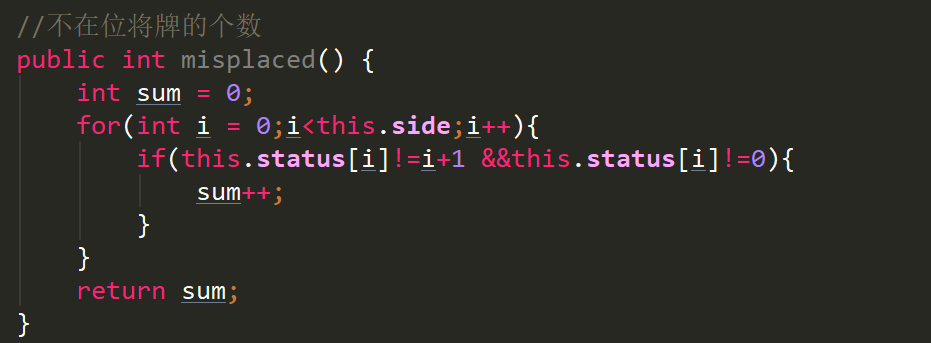
1. 第三阶段
2. 让我们重申一下我们现在已有的评估函数。
   * + 1. 逆序对数

我们用来计算status[i]后面的全部j>i,但是status[j]<status[i]的格子数，这种是最慢的。启发性最差



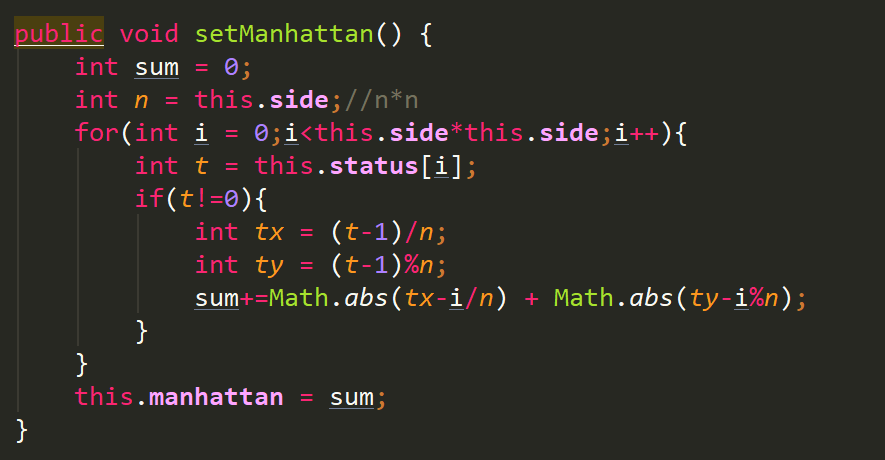
* + - 1. 错位将牌数（海明距离）：

我们需要计算出不在正确的位置上的将牌数量。但是我们需要忽略空格的存在，即使我们记它的value值为0.然而，错位将牌数得到的启发性并不强，也非常的慢，但是比较逆序对数要好一些，并且与曼哈顿距离相比我们将要搜索比其他评估函数多得多的结点，才能到达目的地。



* + - 1. 曼哈顿距离：

曼哈顿距离是计算所有格子与目标格局的行列距离的绝对值之和，因此，我们需要计算所有格子的曼哈顿距离之和，除了空白格子之外。曼哈顿距离比海明距离更快了，但是对于高阶的4阶puzzle仍然无能为力。

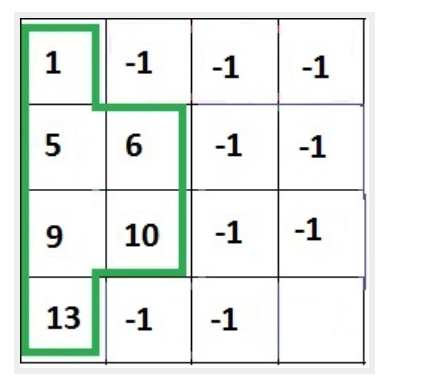


* + - 1. 在第三阶段，我们使用了disjoint pattern database方法。具体来说，就是将整个4\*4的格局分成4个格局，我们将要单独计算四次。比如：

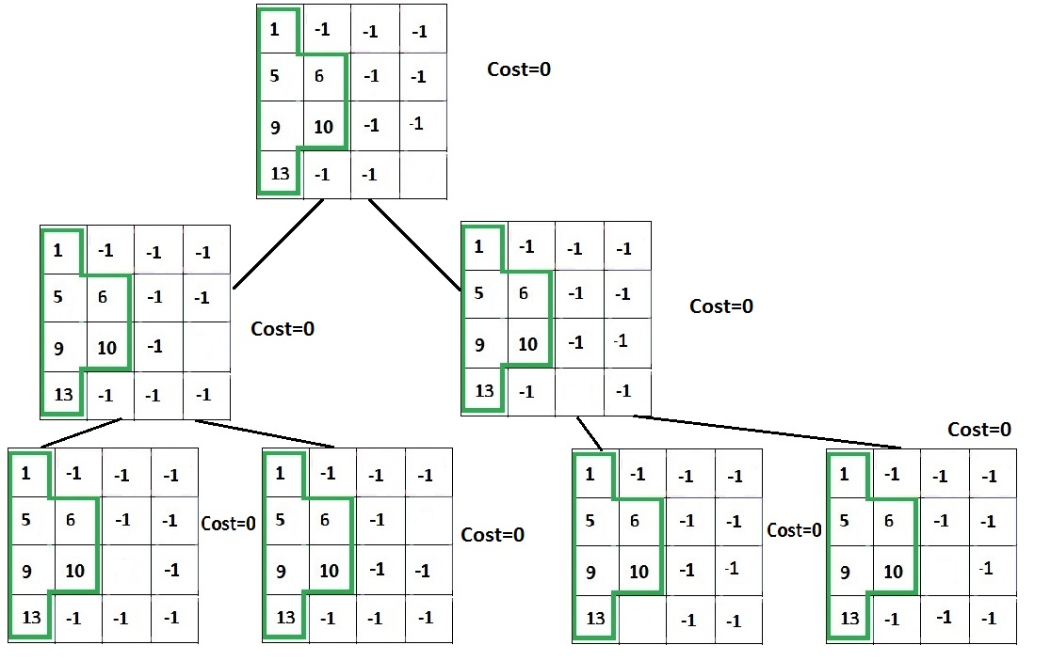


1. 每次计算时假设除了对应的格局之外的格子中的值都是-1，而空格的值是0.当空格和-1交换的时候，cost不变，如果空格和小格局中的值交换的时候，cost+1,其余时刻，cost不变。我们接着我们将4个2\*2的小格局分别存在4个二进制文件中（ouput1.txt——output4.txt）中，方便读取。那么之后对于A\*或者是IDA\*解法求Npuzzle时所得到的中间格局的评估函数则可以利用这里求得。我们把此中间格局划分为四个部分，分别对应四个格局。比如说也做了一些优化，我们把读取文件等相关操作聚集起来单独放到一个Readtxt类里面。并且一开始的时候我们是把小格局对应的zobrist值，以及对应的cost这样一行存进去output文件的。后来经过和同学们的讨论后发现可以直接把存键值对的hashMap存进去。之后对于某一个中间格局，我们只需要把它相对应的四个小格局找出来，然后取出hashMap，相加之后我们就可以得到启发性很强的评估函数值。
2. 另外，我们使用一个名为Hashtable.txt的文件来存储一开始生成的Zobrist随机数组。Hashtable中是用一个long [16][16]的数组来存储Zobrist的值。这样我们使得整个测试的过程基本都在数据库中有所体现，并且可以利用其高效的查询机制来使得程序更快的找到最有的路径。事实证明，在使用了disjoint pattern database方法后，我们可以解出排序为15，14，13...这种n=15情况下最为困难的N-Puzzle问题，而且时间相对来说非常的快，只需要3-4s左右。
3. 经过讨论，我们发现三阶的puzzle用disjoint database求解会比IDA+曼哈顿距离耗时更长，因为曼哈顿距离本身可以求解出结果，但是disjoint database本身在产生数据库和读取数据库的时候就会耗费时间，并且它是要产生所有可能的结局，所以我们决定如果是三阶的puzzle就用IDA+曼哈顿距离，如果是四阶的话就用可以产生启发性更强评估值的disjoint database。
4. Joint database的算法思路

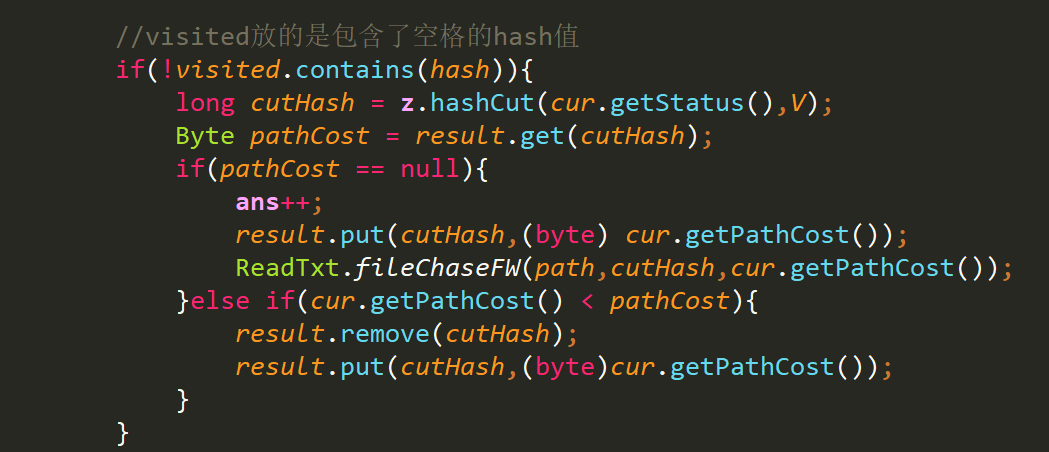
我们通读的论文是把puzzle分成6-6-3 part，但是在生成数据库的时候严重超时，所以我们后来才分成4-4-4-4 part。现在先用6-6-3 part来举个例子。



对于一个已经排好序的目标格局，我们现在只专注于一个小格局中的格子，其他的格子除了空格以外全部设为-1.现在我们将对这个格局做BFS，得出所有它可以得到的格局，（思路其实就是从目标开始推导所有的可能的路径，只有这些路径上的结点才能到达目标结点）。



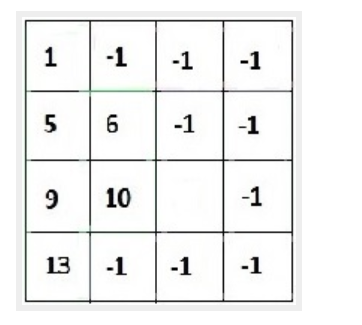
NOTE:我们在创建database时我们并没有存储空格的位置，我们只考虑小格局中的格子的位置，因此对于小格局中的格子分布位置相同，空格不同或者相同，但是Cost不相等的状态，我们只取最小cost的那个。又因为我们最先遇到的结点一定是cost最小的，所以我们可以省略判断的这个步骤，事实证明的确是不需要的。



所以做个优化可以直接删掉BFS中的这一句。

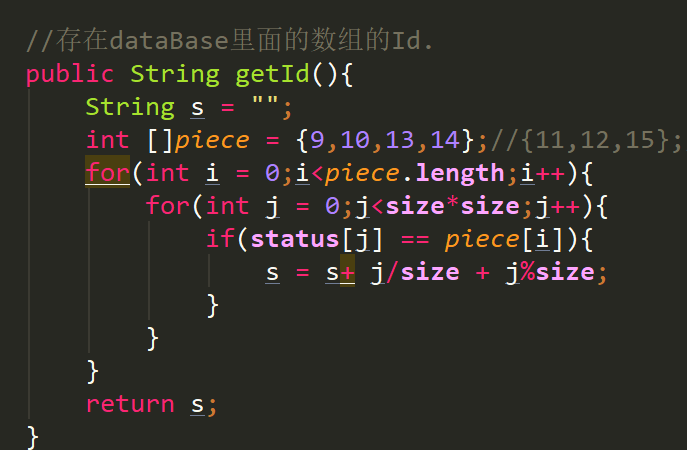
但是我们要注意的是中间结点在压入队列时我们必须要考虑空格的位置，因此那些小格局相同，但是空格位置不同的结点，我们必须要储存它的空格位置+Zobrist值，Cost键值对。

对于下图的格局：

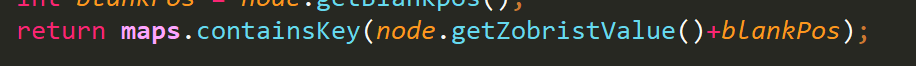


我们看到的建议是储存起每个格子的行列值：112122313241。如果9和10互换了就变成：112122323141。

但是每次都要计算一次，我们还是采用了Zobrist值，只是每次都是按1 5 6 9 10 13的格子顺序来异或的。



对于空格的部分，我们一开始想用Pair<Zobrist,空格的位置>,但是爆内存了。于是后来采取的方法是Zobrist+blankPos来记录这个状态，因为考虑到Zobrist发生碰撞的几率很小。



1. N-Puzzle三个阶段的程序效果对比
2. 图表：

注：

1. 以下时间均是经过10次测试的平均值，由于十次测试所占篇幅较大，所以就省略掉，直接在结果中显示平均值。
2. 除样例1和样例2为三阶外，其余均为4阶
3. 表格中的时间单位均为秒（s）
4. 若超过10分钟没有解出答案，则表中填写无穷大

|  |  |  |  |
| --- | --- | --- | --- |
|  | 第一阶段 | 第二阶段 | 第三阶段 |
| 样例1 | 0.899 | 0.034 | 0.024 |
| 样例2 | 0.776 | 0.024 | 0.017 |
| 样例3 |  | 173.74 | 0.289 |
| 样例4 |  | 0.071 | 0.690 |
| 样例5 |  | 2.095 | 0.339 |
| 样例6 |  | 8.674 | 0.025 |
| 样例7 |  | 54.029 | 0.652 |
| 样例8 |  | 2.661 | 0.021 |
| 样例9 |  | 158.326 | 0.302 |
| 样例10 |  |  | 9.472 |

1. 总结

从上述表格中我们不难发现，每一个阶段都有一个显著的速度提升。在第一

阶段仅仅是使用A\*算法的情况下，只能解出三阶的问题，对于第四阶问题就有点有心无力了，并且对于少数的第三阶问题也有一定概率没法顺利解出。

而对于优化IDA\*算法，我们可以看出它基本可以解决三阶和四阶的问题，但是速度相对来说非常的不稳定，有时候可以很快地解出答案，有时候却需要甚至于五分钟的时间来解出答案，这对于我们来说仍然是很难接受的。

在第三阶段，我们发现，在所有优化做完后，我们可以轻松的解决三阶和四阶的问题，对于最复杂的四阶情况，我们也可以在十秒内求出结果，符合我们的预期目标。

综合来说我们不难发现，A\*算法对于较为简单的问题有着很快的求解速度以及最优解的保证，但是我们从数据中不难发现，它只能解决相对来说较为简单的N-Puzzle问题，对于复杂的四阶问题，即使做了很多优化也无济于事。但是对于IDA\*来说，我们可以解决相对来说比较复杂的N-Puzzle问题，且可以保证最优解，但是在这种情况下，我们的时间有可能会有较大的不稳定性，有可能很快，有可能会需要非常长的时间，对于最复杂的四阶问题依旧存在着无法解出的问题。在最后的使用disjoint pattern database的方法中，我们无论是三阶还是大部分四阶都可以在1s内求出结果，即使面对最复杂的四阶问题都可以进行非常迅速的求解。

1. N-Sliding三个阶段的算法改进
2. 第一阶段

在第一阶段，我们除了用A\*算法外，我们使用了和N-Puzzle一样的优化方法：

1. 利用上一个状态的Zobrist值和本次的移动方向来确定移动后

Zobrist值，不同重新用九次异或计算。

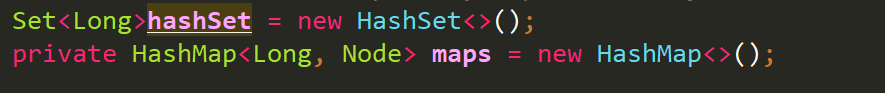
1. 利用优先级队列priority\_queue来存取F值，使得F值可以从大到

小排列，每次只需要从队首pop出第一个值就是我们需要的F值最大的路径， 优化了选择路径的过程。

1. 使用HashMap代替ArrayList来存储节点Node，更加快速的查询是

否重复遍历这个节点。

1. 第二阶段
2. 我们在IDA\*启发式算法的基础上我们进行了进一步改进，使用了set而不是HashSet来存取每一个Node的State，因为这样就不需要在浪费空间去存取Node，我们不需要Node，只需要判断当时的状态即可。这样节省了程序运行和查找的时间，使得整个程序的运行时间有了一定幅度的缩短。



1. 我们尝试用在open表上再继续构建一个best表，这样的话就可以大幅度缩减时间，并且通过一定程度上的剪枝操作使得可以选择的路径为相对来说最优的一个，从而大幅度缩减时间。
2. 第三阶段
3. 在第三阶段，我们发现best表虽然程序执行速度非常快，但是在九阶及以上的情况下，会出现不是最优解，而且会有一定几率出现跳错的情况，在这种情况下，我们果断放弃了best表，在IDA\*算法上继续加以改进。
4. 我们使用了双向搜索的方法，即从两端开始搜索，如果两边都搜到同一个结果，那么直接就可以停止搜索，返回一个答案，这种双向搜索的方法可以对程序的速度有一个略微的提升。

事实上双向搜索对于该问题并不能提供很好的优化，由于该问题状态空

间可视为一张无向图。故由目标状态与初始状态之间的最短路是相同的；由初始状态寻找目标状态 等价于 由目标状态寻找初始状态。所以我们考虑采用双向搜索的策略。算法如下

1.建立两个open表与close表，将初末状态分别加入各自的open表。

2：由初始状态开始，取出一节点。若该节点已被标记逆向访问，则不再继续搜索该节点，并记录正逆向到达该节点的耗散值之和；

3：进行一层A\*搜索。标记这些节点已被正向访问

4：由目标状态开始，取出一节点。若该节点已被标记逆向访问，则不再继续搜索该节点，并记录正逆向到达该节点的耗散值之和；

5：进行一层A\*搜索。标记这些节点已被逆向访问

6：转1，直到两个open表均为空。

1. 在第三阶段，我们进一步优化了Zobrist值。通过讨论，我们认为Zobrist的值虽然不需要每次进行计算，但是生成和使用使用仍略显累赘，我们希望能够找到一种更为简单明了的方法来表示一个Block的状态值。通过小组讨论分析，我们发现了一下的方法：

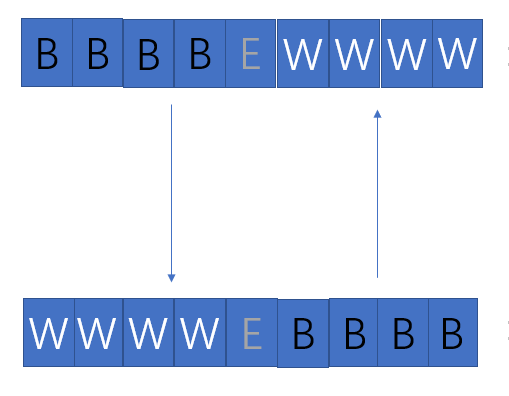
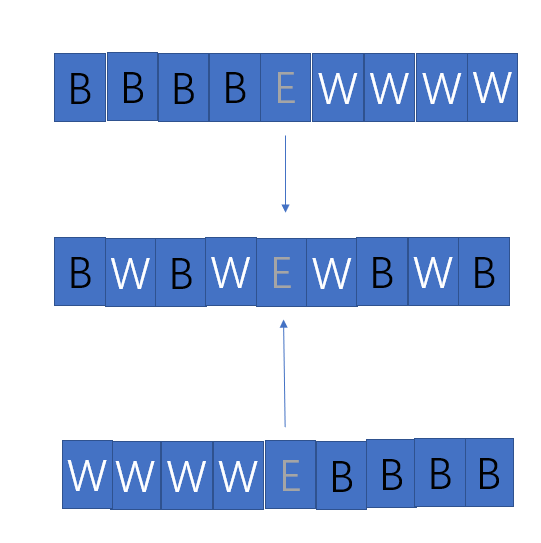
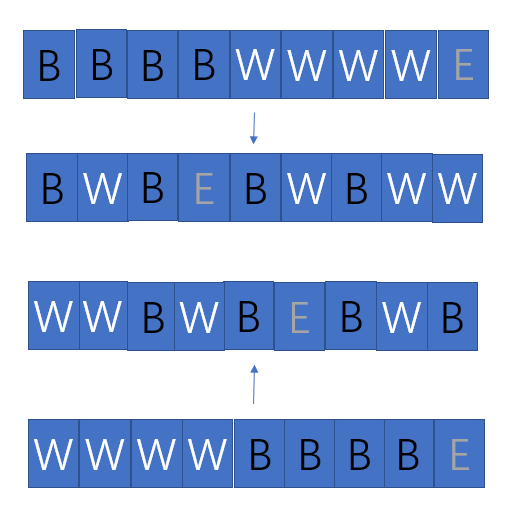
一个Block的每一个格子只有三个状态，分别是白色方格、黑色方格

和空格。并且我们发现，每一个状态下空格都只有一个，此时，3减去1等于2，作为一个计算机专业学生应有的直觉，我们立刻想到了可以使用2进制的方法来表示一个状态。我们用黑色方格 = 1 ，白色方格 =0，空格暂且忽略的方法，先去除空格，形成一个由0、1组成的二进制数字，然后我们将整个的二进制数字向前移动六位。为什么要移动六位呢？因为老师一开始要求的是30阶，那么30个黑色方块加上30个白色方块再加上一个空格正好为61，小于64，也就是2的6次方，所以我们前移所空出来的后六位就可以记录空格的位置，比如空格如果在第0格，那么后六位就是0 0 0 0 0 0，如果空格在第2格，那么后六位就是0 0 0 0 1 0 ，以此类推，我们就可以用一个二级制数字来表示一个Block的状态了。与此同时，如果我们需要比较两个状态是否相同，只需要对两个数进行异或即可，非常的简洁明了快速，同时也可以极大的提高程序运行的效率。

1. （创新的想法）对称性的思路:

让我们考虑更为简单的问题：由初状态寻找末状态，其中，初末两

状态对称。如下图：

定义一：若对于状态A的序列{S}，有S[i]=S[2N-i]，i=0,1…N-1，则称状态A对称；若对于状态A的序列{S}与状态B的序列{Q}，若S[i]=Q[2N-i]，i=0,1…N-1，则称状态A与B对称，A和B构成对称状态对；

定理一：若初末状态对称，则所有解路径中必定存在对称状态（对）。

证明：由于初末状态对称，由末状态向初状态搜索 和 由初状态向末状态搜索本质上是完全一样的。那么对于初状态向末状态的一条解路径，都可以应用在由末状态向初状态转移上。考虑其中任意一条解路径，同时应用在末状态与初状态上。由于初末状态对称，那么每行进一步，所产生的两个后继状态仍然相互对称；所以当初末状态在这条路径上相遇时，此时的状态即为对称状态或对称状态对。故所有解路径中必定存在对称状态（对）。

定义二：若对于状态A，若A对称，且到达A的最小耗散值小与到达其他对称状态的最小耗散值，则称A为最小对称状态，记此时耗散值为f(A)。对于对称状态对A&B，同理有最小对称状态对，记此时耗散值为f(A，B)。

定理二：若初末状态对称，且正向搜索到达状态最小对称状态（对）A（，B），则耗散值最小路径经过该节点，且耗散值为2\*f(A)。

证明：由于初末状态对称，且A对称，所以A到初状态的最小耗散值等于A到末状态的最小耗散值。总的耗散值为2\*f（A）。又据定理一，所有解路径必定存在一对称状态A’,则所有解的耗散值均可表示为2\*f（A’）；则当A’为最小对称状态（对）A时，2\*f(A’)最小，为2\*f(A)。此时解路径经过状态A，定理得证。

由上述定理二我们发现，我们不需要遍历整个状态空间，当我们第一次发现对称状态时即可退出搜索，后半部分的路径可由前半部分构造出来，这样即相当于在双向搜索第一次见面时完成搜索。复杂度下降到了o（XL/2）。

但是因为实验要求是不对称的，并且还有blockV2是要求hn = 0时就视为到达目标格局的情况，所以我们并没有使用这一思想，但是我们却从这个思路中想到了双向BFS，虽然没有解决12阶的block，但是却使9阶的block变得更加快，并且可以解10阶的block了。

1. N-Sliding三个阶段的程序效果对比
2. 图表

注：

1. 由于我们的N-Sliding始终是在A\*和IDA\*上进行优化，所以我们将整个实验对比分成第一和第二两个阶段进行比较，由于我们没能跑出十二阶，所以我们拿九阶为例进行数据比较
2. 数据实验的单位为秒（s）

|  |  |  |
| --- | --- | --- |
|  | 第一阶段 | 第二阶段 |
| 样例1 | 8.762 | 4.377 |
| 样例2 | 35.679 | 16.907 |
| 样例3 | 8.662 | 4.310 |
| 样例4 | 37.689 | 15.843 |

1. 结论

通过分析可知，在A\*环境下的九阶即使我们做了优化，所需要的时间也要在8s和35s左右，虽然可以解出但是所需时间较长。在使用了IDA\*并作相应优化之后，程序的运行时间明显缩短了一半，仅仅需要4s和16s左右，但是对于十二阶的情况仍然没有办法解出，算是一种遗憾吧。

1. 结合N-Puzzle和N-Sliding分析各个算法的优缺点

结合两个实验，我们其实公共使用的方法只有A\*和IDA\*中，而这两个启发性算法的优缺点在前面的几个阶段也表现的比较淋漓尽致。

1. A\*的优缺点

A\*的好处是它结合了Dijkstra算法和BFS算法，当h为0的时候，那么它就是Dijkstra算法。我们可以用A\*算法来解决一些低阶情况下的求最短路径的最优解问题，是一种最好优先算法。

但是A\*要使用大量的空间去存取中间值，这就使得A\*在解决高阶问题的情况下变得稍显无力。

1. IDA\*的优缺点

IDA\*算法在A\*算法上使用了回溯的方法，不需要保存中间状态，我们明显可以感受到它大大节省了空间，同时也提升了解决更高阶问题的可行性。

但是由于IDA\*在每次depth变大的时候都要从头再次搜索，这种操作虽然保证了最优解，但是重复的搜索使得程序有小概率比A\*慢且稍显累赘。

1. 针对N-Puzzle问题比较“错位将牌数”和“曼哈顿距离”两个方法的性能

我们在第一阶段使用了 “逆序对数”，“错位将牌数”在第二阶段使用了“曼哈顿距离”，其比较结果如下图：

|  |  |  |
| --- | --- | --- |
|  | “错位将牌数” | “曼哈顿距离” 逆序对数 |
| 样例1 | 0.999 | 0.044 8.344 |
| 样例2 | 0.798 | 0.034 8.348 |
| 样例3 |  | 172.74 |
| 样例4 |  | 0.061 |
| 样例5 |  | 1.995 |
| 样例6 |  | 8.374 |
| 样例7 |  | 53.229 |
| 样例8 |  | 2.771 |
| 样例9 |  | 159.337 |
| 样例10 |  | 253.260 |

根据上表我们可以得出结论，在使用“错位将牌数”的时候，我们尽管可以在很快时间内求出较为简答的N-Puzzle问题，但是对于复杂的情况来说，寻找时间仍然是很长的，甚至是求不出解的，这在一定程度上限制了“错位将牌数”的使用范围。

而在使用“曼哈顿距离”的时候，通过我们的优化，我们可以跑出四阶几乎所有的情况，并且保证是最优解，虽然要比“错位将牌数”方法要好很多，但是对于时间的把控还是非常的不稳定，会出现有的非常快，有的却很慢的情况，这就导致在解决部分四阶问题时有可能会出现超时的情况。但是总的来说，“曼哈顿距离”的方法的性能要比“错位将牌数”高出一个档次，其能解出的最优解的情况和范围也就更大更多。

1. N-Puzzle和N-Sliding可视化的实现
2. 实验心得体会

在本次的项目一中，我们通过分别实现了N-Puzzle和N-Sliding（滑动积木

块）两个问题并对其进行优化，从而对于人工智能这门课有了一点最初的感悟。

首先，也是感触最深的一点就是，这一门课程比之前所有的课程都要来的独立与突然，这是一门真的只适合大三下学期学生的课程。为什么说是独立和突然的呢？首先，大三下学期的学生通过五个学期的学习，对于整个计算机科学与技术这个专业有了一个大致的了解，其次，我们通过C、C++、JAVA等各种编程语言的练习，对于计算机编程语言的各类语法都有了大致的了解和认识，对于面向对象等专业词汇也有了一定程度上的理解。但是与此同时，我们所做的课堂实验大多是在老师的提示和帮助下完成的，基本上按照老师给出的ppt进行操作就可以完成了，所以我们对于之前的实验可以说只是理解了老师布置的部分，并没有去深度挖掘什么，但是这门课从一开始的两节课的理论讲解后就再没有了课堂的讲解，与此同时，我们也没有任何多余的老师给出的提示性的参考资料，需要我们独立自主的去网上查询相关的论文文献，这极大的考验了我们查找有效资料以及自主学习的能力，这种突然性我是之前所没有想到的，同时这也激发了我们小组对于战胜这个问题的动力。还有，如果我们没有前五个学期对与代码理解能力的积累，那么我们就算是再努力也不太可能在老师规定的时间内搞懂代码并且进行那么多的优化，这门课程中用到的许多数据结构，比如优先队列等概念，我们虽然在数据结构中学到过，但是很少将其应用到实践去检验效果如何，这门课程的优化过程使得我们可以用这些比较经典的数据结构去优化一些相对繁琐的代码，从而使得程序的运行时间更快，效率更高。

第二点，这门课程非常考验团队整体的配合性。这门课程和之前的很多课程不一样，之前有许多课程即使分组之后，仅仅依靠一个或者两个人的努力是完全可以完成的，但是这门课程所布置的代码无论是从数量还是从难易程序上来看，仅仅依靠一两个人的力量是绝对不可能完成的。N-Puzzle和滑动积木块再加上可视化，单单是大模块就有至少三个方面，一到两个人是完全没有办法兼顾的，与此同时，在N-Puzzle和滑动积木块有四个阶段每个阶段都有新的需求，我们需要在更新更好的算法之外，对之前老旧的数据结构进行优化和改进，不断提高程序的运行效率，以此来缩短程序的运行时间，以便可以获得更好的分数。

最后，也是比较关键的一点，第一阶段虽然看起来跟我们想像的人工智能有着较大的区别，可能N-Puzzle和滑动积木块在我们看来并不算是人工智能，但是其实，这两个问题都是提出时间长，非常经典的两个人工智能问题，即使到了今天，也依旧有很多人在这个领域进行研究，希望有更好的方法对这个问题进行求解，所以说项目一这两个程序也可以说是带我们初步领略了人工智能的魅力，同时也在提醒我们，人工智能并没有那么简单，想要在这个领域做出成就，有所建树就必须有持之以恒的耐心，孜孜不倦的求学精神以及较好的合作能力，希望我们在之后的项目二中能够延续项目一中团队合作的精神，努力争取获得更好的成绩！

项目贡献度：

刘魏鑫宁（30%）：编写Npuzzle程序以及block程序，程序优化，实验报告编写，PPT文档编写。

王刚（30%）：编写Npuzzle程序以及block程序，程序优化

崔泽宇（15%）：代码可视化，PPT文档和实验报告可视化部分编写。

吴敬宇（15%）：项目研究报告编写

任峥（5%）：参与讨论，编写puzzle的历史，以及前两个阶段程序的使用文档。

张寅丁（5%）：撰写PPT