**深 圳 大 学 实 验 报 告**

**课程名称： 算法设计与分析**

**实验名称： 回溯法求解消消乐问题**

**学院： 计算机与软件学院 专业：软件工程**

**报告人： 蔡晓鑫 学号： 2017151019 班级： 1**

**同组人： 无**

**指导教师： 杨煊**

**实验时间： 2019/4/7——2019/4/30**

**实验报告提交时间： 2019/4/27**

**教务处制**

**一、实验目的：**

（1）掌握回溯法设计思想。

（2）掌握消消乐问题的回溯法解法。

《开心消消乐》是一款[乐元素](https://baike.baidu.com/item/%E4%B9%90%E5%85%83%E7%B4%A0)研发的三消类休闲游戏。游戏中消除的对象为小动物的头像，包括小浣熊、小狐狸、小青蛙和小鸡等动物头像。玩家通过移动动物头像位置凑够同行/同列3个或3个以上即可消除。



图1

现在定义消消乐规则如下：

* 游戏中共允许K种对象，分布在大小为M×N的格子布局中。

例：图1中，有K=4（小浣熊、小青蛙、小河马、小鸡）种对象，布局大小为M×N=8×4。

* 交换两个对象位置，凑够3个或3个以上即可消除，被消除的空格由正上方对象掉落填充。

例：图1中，可交换（3，1）和（3，2）消去[（5，2），（4，2），（3，2）]和[（3，2），（3，3），（3，4）]。消去过程见图2。

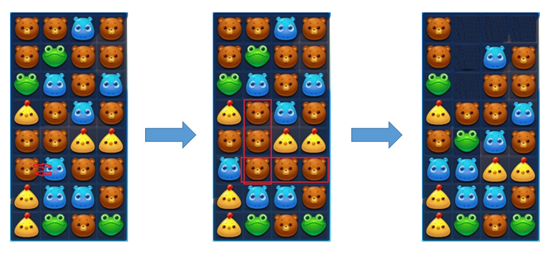


图 2

* 可能出现三种消除方式： 同行(列)三个、同行(列)四个、同行(列)五个。分别        得分1分、4分、10分。注：图2中得分2分。
* 当没有可通过交换消除的对象时，游戏终止。

例：图3将终止游戏。

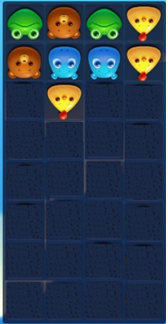


图3

**二、内容：**

1.  给定K, M, N编写代码计算通过一步操作（交换）可得的最大得分。

例如，K=4, M=8, N=4

测试数据（对应图1）

3 3 4 3

3 2 3 3

2 4 3 4

1 3 4 3

3 3 1 1

3 4 3 3

1 4 4 3

1 2 3 2

2． 在1的基础上利用回溯算法，找出X交换步骤之后的最大得分。

3． 对于数值较大的K、M、N、X，在允许近似最优解的情况下，对2中实现的算法进行优化剪枝。并与内容2中最终结果和执行速度进行比较。

4.  如果能实现可视化输出计算结果（包括回溯过程），如图2，可加分。

**三、实验要求**

1. 对不同K，M, N, X的问题依次求解，演示你的求解结果，请提供你机器上能求解的问题最大规模。

2. 在blackboard提交电子版实验报告。源代码和PPT作为实验报告附件上传。

3. 在实验完成之后，将进行一次PPT介绍。

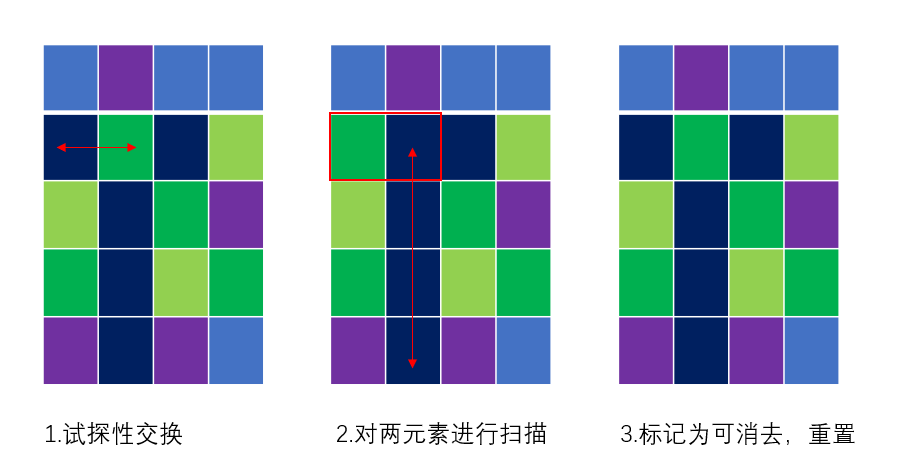
4. 在实验报告中要求详细说明”实验内容1”和“实验内容2”的实现思想。

5. 讨论”实验内容1”和“实验内容2”问题复杂度和K, M, N, X的关系。

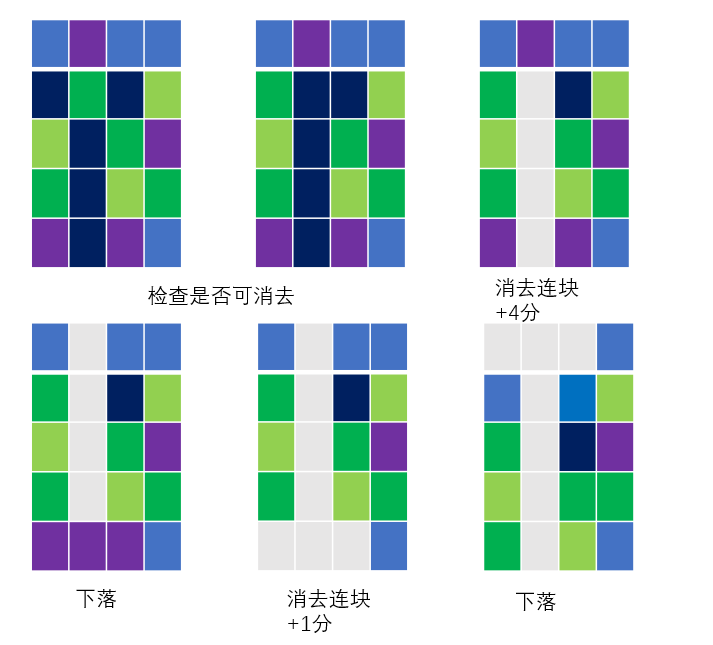
**四、实验成绩**

实验成绩的给分标准是实验报告50%，PPT汇报50%。

1. **消消乐实现**
2. **基础方法实现**
3. 检查某元素与相邻元素交换后能否消去
4. 思路：试探性交换该元素和右相邻（下相邻），对两个相邻元素进行平行，垂直扫描，如若出现三连块状态，标记为可消去状态。以左右交换为例，如下图



1. 进行消去并下落计分，直至再无消去状态
2. 思路：通过1）检查消去状态，重复“消去连块-计分-下落-消去连块-计分…”的步骤，直至棋盘不再具有连块。接着1）图例，如下



1. **通过一步交换获得最大得分**
2. 思路：遍历矩阵，判定元素先右交换后能否发生消去，如若发生消去，记录交换得分，与最高分比较并更新最高分。否则遍历下一元素。再度遍历矩阵，判定元素向下交换的情况，操作同判定向右交换。
3. 伪代码

**int solve\_of\_oneStep()**

**for i = 1 to M**

**for j = 1 to N**

**swap(chess[i][j],chess[i][j+1])**

**if can delete**

**score = getScore()**

**Max\_score=max(score,Max\_score)**

**for i = 1 to M**

**for j = 1 to N**

**swap(chess[i][j],chess[i+1][j])**

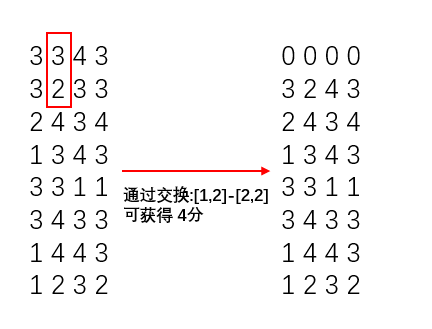
**if can delete**

**score = getScore()**

**Max\_score=max(score,Max\_score)**

**return Max\_score**

1. 样例测试



1. **通过回溯法，找出交换X步骤的最高得分**
2. 思路：在一步交换的基础上，在最大交换次数X的限制下，进行深度优先搜索，每搜索一个状态前，更新当前最高得分。
3. 伪代码

**void backTracing(int score,int depth)**

**Max\_score=max(score,Max\_score)**

**if depth > X**

**return**

**for method= VERTICAL and HORIZON**

**for i = 1 to M**

**for j= 1 to N**

**if method = VERTICAL**

**swap(chess[i][j],chess[i+1][j])**

**else if method = HORIZON**

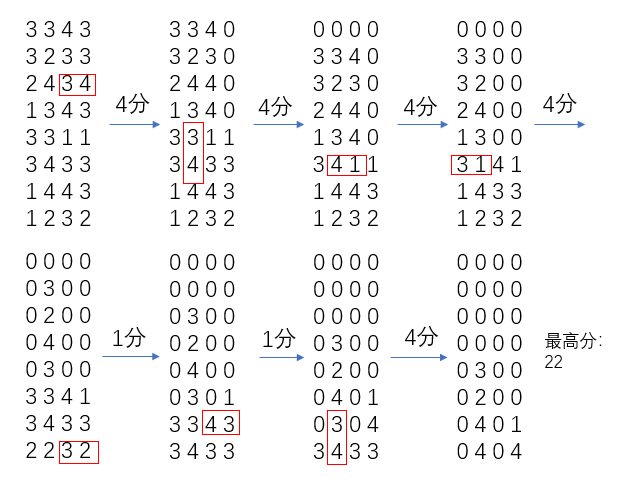
**swap(chess[i][j],chess[i][j+1])**

**if can delete**

**get\_score=getScore()**

**backTracing(score+get\_score,depth+1)**

1. 样例测试



1. 算法复杂度分析
2. 单步复杂度分析：

最大遍历M\*N个节点，进行消去需要最大遍历M\*N个元素，时间复杂度为

1. 每个节点最多产生M\*N个子节点，构造深度为X的解空间树，可得该树具有 个节点，由a)可得每个节点的开销为，可得总的时间复杂度为
2. **剪枝优化回溯法**
3. **剪枝思路**：回溯法的效率极大取决于解空间树的体量，解空间树层节点数随着深度增加以平方倍增长，解空间树的广度越大，层节点的增长规模也就越大。通过剪枝，降低解空间树的深度，压缩宽度，使解空间树变得窄扁。
4. **上下界剪枝**
5. 深度搜索的大部分深入路径都是在做无用功，同时却增加解空间树的深度，增大求解问题规模。通过记录已得最高分，如果按最优情况预估后续深度所得最高分仍无法得到更优解，则剔除该枝。
6. 上下界剪枝伪代码

**average = Max\_score/X**

**depth\_noFind = X – depth + 1**

**if (Max\_score-score)/depth\_noFind > average \* parms**

**return**

1. parms是需调整的参数，假设整个问题最优解为S，那么Max\_score <= S，所以在实际算出剩下层数得到最优解的平均值average后，剪值条件应该略大于average，防止真正的最大值被减去，提高准确率，由此可得parms>1
2. 调整parms参数

在不同问题规模（棋盘规格：5-10，递归深度1-5，K=3）下测试调整parms：1.1 – 2，增量为0.1统计准确率及效率比，期望是取得准确率高，效率比低的parms值，以 平均效率比/平均准确率作为评测值，低值表示性价比高。如下图

与期望符合，当parms增大，解空间树的规模越大，评测值也相应增大。可见1.2、1.3为可考虑值，在parms=1.2时，准确率为0.92，而parms=1.3时，准确率为0.9876。优先设定parms值为1.3

1. **横向剪枝**
2. 为了有效限制解空间树分支的扩展，通过设置宽度阈值，记录每个层次的搜索次数和到达该层的分数，如果当前分支的分数达不到该层的期望，则做剪枝处理。
3. 横向剪枝伪代码

**crossLevel\_num[depth]++**

**levelSum[depth]+=score;**

**if crossLevel\_num[depth] >= maxWidth**

**if levelSum[depth]/crossLevel\_num[depth] > score**

**return**

1. maxWidth是需要调整的参数，maxWidth太小，突破该界定的分支越多，效率越低，maxWidth太大，砍掉的枝条越多，准确率降低。
2. 调整maxWidth参数

在M=10，N=6，K=3，X=4的规模下，同上下界剪枝的评测标准测试maxWidth从5-30的评价值变化

随着maxWidth的增长，评价值呈现出快速下降并趋于平缓的趋势，符合预期。当maxWidth>=9后，曲线趋于平缓，说明此时剪枝达到极限，为保证准确率和剪枝效率，选择maxWidth=12

1. **综合剪枝：**
2. 根据参数调整后，上下界剪枝和横向剪枝都体现出近似解条件下的高效率，通过结合两种剪枝方式，压缩解空间树的宽度和深度，充分提高效率。
3. 通过前两轮调参，设置parms=1.3，maxWidth=12
4. **测试样例**

以K=4，M=8，N=4，X=7 测试报告给定的数据，得到最高分数及效率比

|  |  |  |  |
| --- | --- | --- | --- |
| 测试样例 | 最高得分 | 运行时间/ms | 效率比 |
| 普通回溯法 | 22 | 1632.43 | 1 |
| 上下界剪枝 | 22 | 80.917 | 0.0495686 |
| 横向剪枝 | 22 | 102.933 | 0.0630552 |
| 综合剪枝 | 22 | 26.969 | 0.0165208 |

1. **分析算法运行与M，N，K，X的关系**
2. **在M=8，N=8，X=4的规模下分析K（元素种类）对算法运行时间的影响**

**分析1：**当K值单一改变时，K值越大，同等规模下运行时间下降明显，在K达到某个值（如上图为5）后，运行时间变化不再明显并向0趋近。在棋盘规模固定时，K值的增大会使整个棋盘可交换进行消除的可能性降低，从而减少搜索分支数，体现为解决问题时间消耗变小，当K值增大到一定程度，棋盘基本无法进行消去时，解空间树可能只有几个节点，所以运行时间将逼近0。

1. **在M=8，N=8，K=4的规模下分析X（最大交换次数）对算法运行时间的影响**

**分析1：**当X值单一改变时，随X的增大，算法运行时间增加，普通回溯法的运行时间增长最为明显。在没有剪枝的情况下，解空间树n层的节点数远大于前n-1层节点数总和，于是随着X的增加，求解问题的时间消耗加剧明显。三种剪枝算法与普通回溯法比较，在性能上得到非常明显的提升，通过对三种剪枝算法比较，进一步分析。

**分析2：**随着X的增大，三种剪枝策略的效率分化明显，呈现出算法效率：综合剪枝>上下界剪枝>横向剪枝 的趋势。当搜索深度成为影响算法效率的重要因素时，上下界剪枝直接限制树枝向下蔓延扩张结点的策略明显优于横向剪枝通过设定阈值限制树的宽度的策略，后者无法阻止树的层数整加，从而无法有效抑制树随着深度增加结点数量爆炸性增长的现象。而综合剪枝充分结合两种剪枝策略的特点，进一步压缩解空间树，从而运行效率最高。

1. **在N=6，K=4，X=5的规模下分析M对算法运行时间的影响**

**分析1：**当M值单一变化时，随着M值的增长，算法运行时间增加，说明M值的增大会扩大解空间树的规模。其中普通回溯法的增长趋势最为明显，通过不同策略的剪枝，算法得到明显的优化。通过比较三种剪枝策略的算法运行时间进一步分析。

**分析2：**M在一定范围内，三种剪枝算法运行时间差别不大且缓慢增长。当M达到某个取值（M=11），横向剪枝算法的运行时间上升明显，如同2.分析1，当数据规模达到一定程度，横向剪枝抑制结点数爆炸现象不明显。从总体上看，算法效率：综合剪枝>上下界剪枝>横向剪枝。

1. **在M=6，K=4，X=5的规模下分析N对算法运行时间的影响**

**分析1：**当N值单一变化时，随着N值的增长，算法运行时间增加，说明N值的增大会扩大解空间树的规模。其中普通回溯法的增长趋势最为明显，通过不同策略的剪枝，算法得到明显的优化。通过比较三种剪枝策略的算法运行时间进一步分析。

**分析2：**N一定范围内，三种剪枝算法运行时间差别不大且缓慢增长。当N达到某个取值（N=11），横向剪枝算法的运行时间上升明显，如同2.的分析1当数据规模达到一定程度，横向剪枝抑制结点数爆炸现象不明显。从总体上看，算法效率：综合剪枝>上下界剪枝>横向剪枝。

1. **算法最大规模求解**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | M | N | K | X | 耗时/ms |
| 普通回溯法 | 10 | 10 | 4 | 5 | 1479820 |
| 上下界剪枝 | 10 | 10 | 4 | 9 | 13157 |
| 横向剪枝 | 10 | 10 | 4 | 6 | 94849.5 |
| 综合剪枝 | 10 | 10 | 4 | 10 | 322638 |

1. **总结**
2. **算法选择策略**

在(三)中的分析中，算法效率的排名不约而同都为：综合剪枝>上下界剪枝>横向剪枝>普通回溯法，但同时注意到在数据规模较小下，由于结点繁殖规模较小，横向剪枝的效果略优于上下界剪枝，在数据处理的过程中，发现在数据规模很小，例如M=6，N=6，K=4，X=5的情况下，横向剪枝错剪的几率较大，所以横向剪枝更适用于中等数据规模的求解。

汇总前面数据测试的数据并补充，得到不同剪枝策略的准确率，如下图表

|  |  |
| --- | --- |
| 上下界剪枝 | 0.967 |
| 横向剪枝 | 0.921 |
| 综合剪枝 | 0.9172 |

上下界剪枝的准确率非常高，而且效率也相对较高。而效率最优的综合剪枝由于集成了前两种策略的特点，所以错剪的几率也提高，通过对时间消耗及准确率的权衡，选择不同的策略应用于不同场景。

1. **实验中遇到的问题**
2. 生成随机矩阵：一开始使用shuffle\_random，打乱元素数组的顺序，再深度遍历得到一个符合不可消去条件的矩阵，但是实际使用发现，shuffle\_random生成的随机矩阵具有统一性，应该涉及撒播随机种子的策略问题，于是最后更换为随机数引擎生成种子，shuffle函数打乱数组顺序的方法生成随机矩阵。随机数引擎必须声明成静态，短时间内多次调用会导致程序崩溃。
3. 编写消去并获取本次消去得分的机制

会出现几种情况，消去并下落后，还会再发生消去，或者即将消去的连块呈十字形。

对于第一种情况，如果只检查并下落计分一次，那消去后棋盘会存在需消去的连块，导致测试数据无意义。所以在此需要对棋盘不断扫描，直到返回的得分为0。

对于第二种情况，如果每次遇到连块都直接消去，不考虑十字形的连块，那么并不能获得本轮消去的准确得分，所以用标记矩阵标记应该消去的位置，在扫描完整个棋盘之前不更改棋盘位置的值。

1. **参数parms和maxWidth的调整**

在允许近似解的前提下，需要充分权衡效率和准确率，这种权衡体现在参数的调整上，参数实际上代表该结点上的路径能得到最优解的期望。对于不同的问题求解规模参数需要进行微调，限于本人计算机的性能，只能在完成实验报告所需的数据规模下进行调参。

1. **解空间树**

在完成本次实验报告的过程中，印象最深刻的便是当最大交换次数增加1时，问题的求解代价直线上升，原因是树状结构当层的结点是远超之前所有结点总和的（对n叉树而言），所以在求解实际问题的过程中，需要注意对解空间树进行纵向剪枝，从而降低求解问题的代价。