**深 圳 大 学 实 验 报 告**

**课程名称： 算法设计与分析**

**实验项目名称： 实验3 回溯法求解消消乐问题**

**学院： 计算机与软件学院**

**专业： 计算机科学与技术**

**指导教师： 刘刚**

**报告人：黄志鹏 学号：2017303008 班级：计科1班**

**实验时间： 2020年4月21日 - 2020年5月12日**

**实验报告提交时间： 2020年5月12日**

**教务部制**

**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。

1. **实验内容**
2. 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

1. 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。
2. 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。
3. 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。
4. **实验步骤与结果**
5. **问题描述**
6. 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。
7. 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。
8. 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。
9. **算法原理描述**
10. 消消乐设计
11. 创建二维数组代表棋盘，创建后检测是否合法（初始态不能有消除）
12. 交换两点，并判断是否合法交换：

-交换的两元素不能相同

-交换的位置不能超过边界

-不能与空元素交换

-交换后必须至少消去一组元素

交换两点后，若判断交换不合法，则恢复两点的位置。

1. 交换后，消去符合要求的元素，并记分

-利用一个标记棋盘从左下到右上判断是否可消去，横向可消去的元素设为1，纵向可消去的元素设为2，纵向及横向都可消去则设为3

-遍历标记棋盘，根据设置的值计算得分

1. 更新棋盘，将空元素上方的元素填充空元素位置，重复步骤（c），直到不能更新为止。
2. 棋盘设置完毕
3. 无剪枝回溯法

（a）以棋盘状态作为结点，每一个棋盘状态代表一个结点

（b）以最初始的状态为初始结点，以没有可以消除元素的棋盘为叶子结点

（c）先从左下角开始交换，判断是否可以产生新的结点，如果可以产生新的结点，则继续递归从左下角开始交换，继续判断是否可以产生新的结点，直到没有可以消除的元素时，则返回上一个结点，交换下一个位置的元素，继续判断，直到所有棋盘状态都判断过，则终止。

3. 剪枝设计

（a）该剪枝需要以当前层次得分平均值avg\_score作为参数，平均值初始值设为1，在遍历过程中足部调整平均值，若当前得分now\_score > avg\_score，则继续递归，并改变平均值（avg\_score+now\_score）/n+1。

（b）为了提高准确率，在（1）基础上再加一个倍数参数p，取值1.8-2.2间（根据实测判断），即若当前得分now\_score \* p > avg\_score，则继续递归。

1. **算法实现的核心伪代码**
2. 标记消消乐可消去元素，更新并返回得分

check\_max\_score(int temp\_board[M][N], int x1, int y1, int x2, int y2)

    if (!if\_swap\_point(temp\_board, x1, y1, x2, y2)) return 0 // 判断元素交换是否合法

swap(temp\_board[x2][y2], temp\_board[x1][y1]);

while(true)

        flag\_point(temp\_board, flag\_board) // 生成标记棋盘

        if (check\_flag\_board\_zero(flag\_board)) break // 棋盘为空则退出while循环

        score += delete\_points\_and\_count\_score(temp\_board, flag\_board); // 每次更新棋盘得分累加

        update\_chessboard(temp\_board); // 更新棋盘

    if (score == 0) // 若得分为零，则为不合法交换，恢复元素位置

        swap(temp\_board[x2][y2], temp\_board[x1][y1]);

return score

1. 无剪枝回溯算法

swap\_x\_step\_backtrack(int chess\_board[M][N], int depth, int max\_score)

    update\_max\_score(score)

    if (depth > X)  return // 步数耗尽返回

    for i=M-1 to 0 ; --i // 从左下到右上判断

        for j=0 to N-1; ++j

            swap\_up\_point\_score = check\_max\_score(temp\_board, i, j, i - 1, j) // 交换上方元素得分

            if (swap\_up\_point\_score != 0) // 得分为零则交换不合法

                score += swap\_up\_point\_score // 累计得分

                swap\_x\_step\_backtrack(temp\_board, depth, score) // 递归调用

            swap\_right\_point\_score = check\_max\_score(temp\_board, i, j, i, j + 1)// 交换有右方元素得分

            if (swap\_right\_point\_score != 0)// 得分为零则交换不合法

                score += swap\_right\_point\_score // 累计得分

                swap\_x\_step\_backtrack(temp\_board, depth, score) // 递归调用

1. 有剪枝回溯法

swap\_x\_step\_backtrack\_horizontal\_pruning(int chess\_board[M][N], int depth, int max\_score)

    update\_max\_score(score);

    if (depth > X) return;

for i = M - 1 to 0

        for j = 0 to N - 1

            swap\_up\_point\_score = check\_max\_score(temp\_board, i, j, i - 1, j); // 交换上方元素得分

            if (swap\_up\_point\_score != 0) // 交换合法

                temp\_score = float(swap\_up\_point\_score) \* parameter; // 当前得分\*倍数因子

                count(avg\_score) // 计算当前平局值

                if(temp\_score>avg\_score){

                    horizontal\_purning\_depth\_num[depth] += 1; // 更新平均值

                    horizontal\_purning\_sum[depth] += swap\_up\_point\_score; // 更新平均值

                    swap\_x\_step\_backtrack(temp\_board, depth, score); // 递归调用

            int swap\_right\_point\_score = check\_max\_score(temp\_board, i, j, i, j + 1);// 交换有右方元素得分

            if (swap\_right\_point\_score != 0)

                temp\_score = float(swap\_right\_point\_score) \* parameter; // 当前得分\*倍数因子

               count(avg\_score) // 计算当前平局值

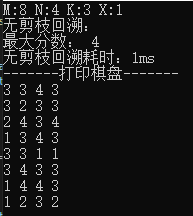
                if (temp\_score > avg\_score)

                    horizontal\_purning\_depth\_num[depth] += 1; // 更新平均值

                    horizontal\_purning\_sum[depth] += swap\_right\_point\_score; // 更新平均值

                    swap\_x\_step\_backtrack(temp\_board, depth, score); // 递归调用

1. **算法测试结果及效率分析**
2. 交换一步
3. 测试例子给出棋盘的最大得分，可知一步最大得分为4，结果如下所示：



1. 测试最大规模，一步交换得分，结果如下所示：



1. 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。

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 最大步数 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
| 得分 | 4 | 9 | 15 | 17 | 20 | 21 | 22 | 22 | 22 |

样例交换最大步数和最大得分关系如下表所示：

1. 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。
2. 固定N=9，K=3，X=4时，时间与M的关系

剪枝回溯法的准确率如下图所示：

分析：

1. 当棋盘的行数增大时，无剪枝法所需时增加的非常厉害
2. 通过剪枝，可以有效的减少运行时间
3. 当M=8时，剪枝法所需时间稍有下降，可能时由于数据比较特殊所导致的
4. 随着M增大，剪枝法的准确率基本不变，误差与M的增大无明显关系
5. 固定M=9，K=3，X=4时，时间与N的关系

剪枝回溯法的准确率如下图所示：

分析：

1. N增大对时间的影响与M增大对时间的影响大致相同，在次不再累述
2. 同时可看出准确率与N无明显关系
3. 固定M=8，N=8，X=4时，时间与K的关系

剪枝回溯法的准确率如下图所示：

分析：

1. 随着元素种类K的增大，无剪枝和有剪枝回溯耗时都越来越小，主要原因是当元素种类增多时，棋盘更难出现相同的状态，所调用的递归就越来越少
2. 随着元素种类K的增大，无剪枝和有剪枝回溯耗时接近相同，主要原因也是当元素种类增多时，棋盘更难出现相同的状态，从而剪枝的效果愈来愈小
3. 由于棋盘更难出现可消去的状态，所以剪枝准确率有提高。
4. 固定M=8，N=8，K=4时，时间与X的关系

剪枝回溯法的准确率如下图所示：

分析：

1. 随着步数X的的增加，无剪枝回溯耗时增加十分显著，当X到9且8\*8棋盘式，再我的机器上跑了20分钟也未能跑出结果
2. 通过剪枝，可有效的减少运行时间
3. 当步数增加时，剪枝算法越来越难比较最优解
4. **总结**

有上述分析可知，随着M，N，K的增加，该剪枝法对减少运行时间有明显的效果，同时准确率也不会有太大的误差。对于交换步数X的增加，该剪枝法也具有显著的效果，但随着步数的增多，准确率会有所下降，这是因为步数越多，越难以判断棋盘的最优解。同时使用剪枝法时可调整倍速因子parameter，根据需要可获得更好准确或者更短的耗时。

**四．实验心得**

通过本次实验，加深了我对回溯算法的认识，本次实验主要有两个难点，消消乐的设计和剪枝算法的设计，消消乐设计需要考虑多种消去情况和更新棋盘，剪枝算法设计要需要权衡运行效率和准确率。

同时，在进行算法设计中，若控制数据读写、修改的操作尽量减少，对于算法性能提升也具有一定效果，合理分析和利用算法能极大的提高效率。

|  |
| --- |
| 指导教师批阅意见：  成绩评定：  指导教师签字：  年 月 日 |
| 备注： |

注：1、报告内的项目或内容设置，可根据实际情况加以调整和补充。

2、教师批改学生实验报告时间应在学生提交实验报告时间后10日内。