Lesson 2——搜索

NOIP 非基础新课

Garen Wang

2019 年 3 月 2 日

目录

搜索可以讲得很难，但是我讲不来。所以大概就是下面这些：

目录

搜索可以讲得很难，但是我讲不来。所以大概就是下面这些：

回溯思想 (backtracking)

回溯思想 (backtracking)

- 利用递归搞的搜索大多都要用到这种回溯思想。

回溯思想 (backtracking)

- 利用递归搞的搜索大多都要用到这种回溯思想。
- 回溯法采用试错的思想，它尝试分步的去解决一个问题。

回溯思想 (backtracking)

- 利用递归搞的搜索大多都要用到这种回溯思想。
- 回溯法采用试错的思想，它尝试分步的去解决一个问题。
- 在分步解决问题的过程中，当它通过尝试发现现有的分步答案不能得到有效的正确的解答的时候，它将取消上一步甚至是上几步的计算，再通过其它的可能的分步解答再次尝试寻找问题的答案。

回溯思想 (backtracking)

- 利用递归搞的搜索大多都要用到这种回溯思想。
- 回溯法采用试错的思想，它尝试分步的去解决一个问题。
- 在分步解决问题的过程中，当它通过尝试发现现有的分步答案不能得到有效的正确的解答的时候，它将取消上一步甚至是上几步的计算，再通过其它的可能的分步解答再次尝试寻找问题的答案。
- 具体实现的样子就是：

回溯思想 (backtracking)

- 利用递归搞的搜索大多都要用到这种回溯思想。
- 回溯法采用试错的思想，它尝试分步的去解决一个问题。
- 在分步解决问题的过程中，当它通过尝试发现现有的分步答案不能得到有效的正确的解答的时候，它将取消上一步甚至是上几步的计算，再通过其它的可能的分步解答再次尝试寻找问题的答案。
- 具体实现的样子就是：
- 进入新一步，修改相应状态；退出一步，把之前改过的改回来，走其他的路。

回溯框架

```
void dfs(int t, 其他需要的参数) {
    if(t > n) {
        if(满足条件) 输出;
        return;
    }
    for(int i = start; i <= end; i++) {
        if(满足能选的条件) {
            做标记;
            dfs(t + 1, 其他需要的参数);
            把标记改回来;
        }
    }
}
```

例题：八皇后问题

例题：八皇后问题

- 给你 8*8 的方格，放 8 个皇后，让皇后们不能互相攻击，即不能处于同一行、同一列或同一对角线。

例题：八皇后问题

- 给你 8*8 的方格，放 8 个皇后，让皇后们不能互相攻击，即不能处于同一行、同一列或同一对角线。
- 利用回溯法，在第　行放皇后视为第　步，行中的每一个点都去判断是否合法，若合法即放置皇后进入下一步。

例题：八皇后问题

- 给你 8*8 的方格，放 8 个皇后，让皇后们不能互相攻击，即不能处于同一行、同一列或同一对角线。
- 利用回溯法，在第　行放皇后视为第　步，行中的每一个点都去判断是否合法，若合法即放置皇后进入下一步。
- 同样，进入下一步涉及到标记的修改和改回来两个操作。

例题：八皇后问题

- 给你 8*8 的方格，放 8 个皇后，让皇后们不能互相攻击，即不能处于同一行、同一列或同一对角线。
- 利用回溯法，在第　行放皇后视为第　步，行中的每一个点都去判断是否合法，若合法即放置皇后进入下一步。
- 同样，进入下一步涉及到标记的修改和改回来两个操作。
- 八皇后问题用回溯法解决还没什么大问题，因为回溯法不加剪枝最坏是有指数级的复杂度。

例题：八皇后问题

- 给你 8*8 的方格，放 8 个皇后，让皇后们不能互相攻击，即不能处于同一行、同一列或同一对角线。
- 利用回溯法，在第　行放皇后视为第　步，行中的每一个点都去判断是否合法，若合法即放置皇后进入下一步。
- 同样，进入下一步涉及到标记的修改和改回来两个操作。
- 八皇后问题用回溯法解决还没什么大问题，因为回溯法不加剪枝最坏是有指数级的复杂度。
- 八皇后的扩展：n 皇后问题。有一种二进制的方法可以快速解决。这里就不讲了。

dfs（深度优先搜索）

dfs（深度优先搜索）

- 在树的遍历中有三种深度优先的遍历，同理也有深度优先的搜索。

dfs（深度优先搜索）

- 在树的遍历中有三种深度优先的遍历，同理也有深度优先的搜索。
- 在寻找连通块的过程中，经常使用到 dfs。这时有一种称呼：floodfill（洪水填充）。

dfs（深度优先搜索）

- 在树的遍历中有三种深度优先的遍历，同理也有深度优先的搜索。
- 在寻找连通块的过程中，经常使用到 dfs。这时有一种称呼：floodfill（洪水填充）。
- 回溯法就是使用 dfs 的框架。

dfs（深度优先搜索）

- 在树的遍历中有三种深度优先的遍历，同理也有深度优先的搜索。
- 在寻找连通块的过程中，经常使用到 dfs。这时有一种称呼：floodfill（洪水填充）。
- 回溯法就是使用 dfs 的框架。
- dfs 相信大家都学过，就不细讲了。

例题 1：luoguP1312

2 、任一时刻，如果在一横行或者竖列上有连续三个或者三个以上相同颜色的方块，则它们将立即被消除（参见图 1 到图3）。

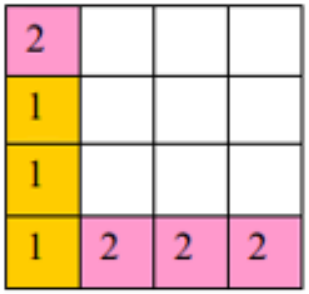


图 4

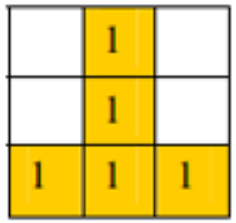


图 5

注意：

a) 如果同时有多组方块满足消除条件，几组方块会同时被消除（例如下面图4 ，三个颜色为1 的方块和三个颜色为 2 的方块会同时被消除，最后剩下一个颜色为2的方块）。

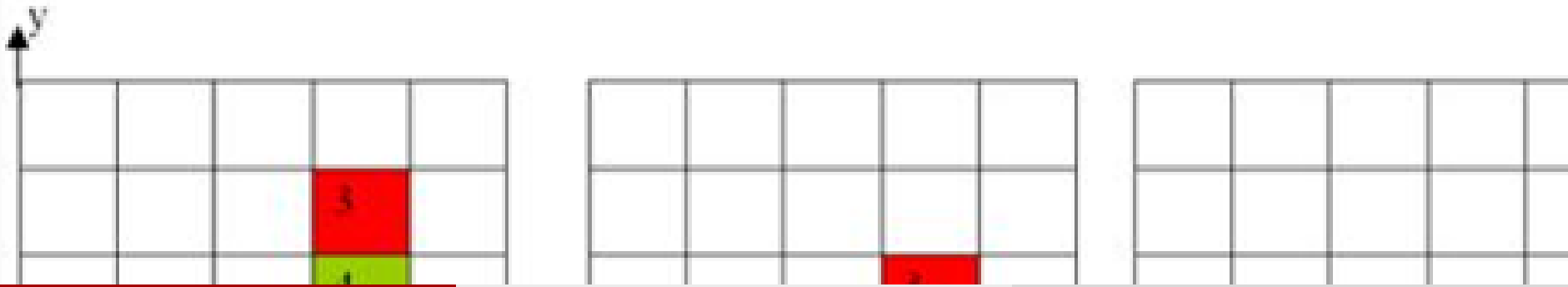
b) 当出现行和列都满足消除条件且行列共享某个方块时，行和列上满足消除条件的所有方块会被同时消除（例如下面图5 所示的情形，5 个方块会同时被消除）。

3 、方块消除之后，消除位置之上的方块将掉落，掉落后可能会引起新的方块消除。注意：掉落的过程中将不会有方块的消除。

例题 1：luoguP1312

Mayan puzzle 是最近流行起来的一个游戏。游戏界面是一个7 行×
悬空堆放，即方块必须放在最下面一行，或者放在其他方块之上。游
消除方块的规则如下：

1、每步移动可以且仅可以沿横向（即向左或向右）拖动某一方块一
置（以下称目标位置）也有方块，那么这两个方块将交换位置（参见
标位置上没有方块，那么被拖动的方块将从原来的竖列中抽出，并从
和图2）；



例题 1——分析

例题 1——分析

- 这道题就是一道大模拟题，主要考察 dfs 搜索框架、剪枝和码力。

例题 1——分析

- 这道题就是一道大模拟题，主要考察 dfs 搜索框架、剪枝和码力。
- 因为搜索步数只有 5，所以直接用 dfs 就行了。

例题 1——分析

- 这道题就是一道大模拟题，主要考察 dfs 搜索框架、剪枝和码力。
- 因为搜索步数只有 5，所以直接用 dfs 就行了。
- 搜索的每一步就是枚举一个方块，与左右交换，同时增加一步。

例题 1——分析

- 这道题就是一道大模拟题，主要考察 dfs 搜索框架、剪枝和码力。
- 因为搜索步数只有 5，所以直接用 dfs 就行了。
- 搜索的每一步就是枚举一个方块，与左右交换，同时增加一步。
- 其实这道题最重要的就是如何处理方块掉落和方块消除。

例题 1——分析

- 这道题就是一道大模拟题，主要考察 dfs 搜索框架、剪枝和码力。
- 因为搜索步数只有 5，所以直接用 dfs 就行了。
- 搜索的每一步就是枚举一个方块，与左右交换，同时增加一步。
- 其实这道题最重要的就是如何处理方块掉落和方块消除。
- 方块掉落：枚举每一列，从下往上遍历，记录当前遍历到的空格数，用老方块迭代掉新方块。

例题 1——分析

- 这道题就是一道大模拟题，主要考察 dfs 搜索框架、剪枝和码力。
- 因为搜索步数只有 5，所以直接用 dfs 就行了。
- 搜索的每一步就是枚举一个方块，与左右交换，同时增加一步。
- 其实这道题最重要的就是如何处理方块掉落和方块消除。
- 方块掉落：枚举每一列，从下往上遍历，记录当前遍历到的空格数，用老方块迭代掉新方块。
- 方块消除：弄一个 vis 数组，枚举所有方块，用一个 if 确定至少的 3 个，然后左右或者上下扩展。

例题 1——分析

例题 1——分析

- 但是你以为这样就完了？

例题 1——分析

- 但是你以为这样就完了？
- 其实还需要剪枝。

例题 1——分析

- 但是你以为这样就完了？
- 其实还需要剪枝。
- 这里有两个剪枝：

例题 1——分析

- 但是你以为这样就完了？
- 其实还需要剪枝。
- 这里有两个剪枝：
  1. 交换两个相同的方块时，剪枝！

例题 1——分析

- 但是你以为这样就完了？
- 其实还需要剪枝。
- 这里有两个剪枝：
  1. 交换两个相同的方块时，剪枝！
  2. 若一个方块向左交换另一个方块时，剪枝！（因为从另一个方块向右交换过来会更优）

例题 1——分析

- 但是你以为这样就完了？
- 其实还需要剪枝。
- 这里有两个剪枝：
  1. 交换两个相同的方块时，剪枝！
  2. 若一个方块向左交换另一个方块时，剪枝！（因为从另一个方块向右交换过来会更优）

然后就完事了。

例题 2：luoguP2668

现在，牛牛只想知道，对于自己的若干组手牌，分别最少需要多少次出牌可以将它们打光。请你帮他解决这个问题。

需要注意的是，本题中游戏者每次可以出手的牌型与一般的斗地主相似而略有不同。具体规则如下：

牌型	牌型说明	牌型举例照片
火箭	即双王（双鬼牌）。	
炸弹	四张同点牌。如四个 A。	
单张牌	单张牌，比如 3。	
对子牌	两张码数相同的牌。	
三张牌	三张码数相同的牌。	
三带一	三张码数相同的牌 + 一张单牌。例如：三张 3+单 4	
三带二	三张码数相同的牌 + 一对牌。例如：三张 3+对 4	
单顺子	五张或更多码数连续的单牌（不包括 2 点和双王）例如：单 7+单 8+单 9+单 10+单 J。另外，**在顺牌（单顺子、双顺子、三顺子）中，牌的花色不要求相同。**	
双顺子	三对或更多码数连续的对牌（不包括 2 点和双王）。例如：对 3+对 4+对 5。	
三顺子	二个或更多码数连续的三张牌（不能包括 2 点和双王）。例如：三张 3+三张 4+三张 5。	

例题 2——分析

例题 2——分析

- 同样是个大模拟。

例题 2——分析

- 同样是个大模拟。
- 我们用桶把所有的牌都存下来。之后的所有操作都建立在桶上。

例题 2——分析

- 同样是个大模拟。
- 我们用桶把所有的牌都存下来。之后的所有操作都建立在桶上。
- 因为题意里面与花色一点关系都没有，所以直接抛弃掉这个信息。

例题 2——分析

- 同样是个大模拟。
- 我们用桶把所有的牌都存下来。之后的所有操作都建立在桶上。
- 因为题意里面与花色一点关系都没有，所以直接抛弃掉这个信息。
- 考虑你玩斗地主的做法：先出带牌和顺子，再出对子和单牌。

例题 2——分析

- 同样是个大模拟。
- 我们用桶把所有的牌都存下来。之后的所有操作都建立在桶上。
- 因为题意里面与花色一点关系都没有，所以直接抛弃掉这个信息。
- 考虑你玩斗地主的做法：先出带牌和顺子，再出对子和单牌。
- 打到最后的单牌是没必要回溯的，是可以直接看出答案的。

例题 2——分析

- 同样是个大模拟。
- 我们用桶把所有的牌都存下来。之后的所有操作都建立在桶上。
- 因为题意里面与花色一点关系都没有，所以直接抛弃掉这个信息。
- 考虑你玩斗地主的做法：先出带牌和顺子，再出对子和单牌。
- 打到最后的单牌是没必要回溯的，是可以直接看出答案的。
- 所以程序的思路就出来了：先暴力枚高级牌，再贪心打低级牌。

例题 2——分析

例题 2——分析

- 所以那个 dfs 函数的思路就可以出来了：

例题 2——分析

- 所以那个 dfs 函数的思路就可以出来了：
  1. 考虑四带两对、四带两张单牌、炸弹、三带一对、三带一张单牌、三张牌。

例题 2——分析

- 所以那个 dfs 函数的思路就可以出来了：
  1. 考虑四带两对、四带两张单牌、炸弹、三带一对、三带一张单牌、三张牌。
  2. 考虑三顺子、双顺子、单顺子。

例题 2——分析

- 所以那个 dfs 函数的思路就可以出来了：
  1. 考虑四带两对、四带两张单牌、炸弹、三带一对、三带一张单牌、三张牌。
  2. 考虑三顺子、双顺子、单顺子。
  3. 最后遍历所有的牌，剩一张就单张牌，剩两张就出对。

例题 2——分析

- 所以那个 dfs 函数的思路就可以出来了：
  1. 考虑四带两对、四带两张单牌、炸弹、三带一对、三带一张单牌、三张牌。
  2. 考虑三顺子、双顺子、单顺子。
  3. 最后遍历所有的牌，剩一张就单张牌，剩两张就出对。
- 注意：火箭在这里没什么卵用，只需要看成一种对子就可以。

例题 2——分析

- 所以那个 dfs 函数的思路就可以出来了：
  1. 考虑四带两对、四带两张单牌、炸弹、三带一对、三带一张单牌、三张牌。
  2. 考虑三顺子、双顺子、单顺子。
  3. 最后遍历所有的牌，剩一张就单张牌，剩两张就出对。
- 注意：火箭在这里没什么卵用，只需要看成一种对子就可以。
- 因为求的是最少次数，也可以适当性来一点最优性剪枝。

例题 2——分析

- 所以那个 dfs 函数的思路就可以出来了：
  1. 考虑四带两对、四带两张单牌、炸弹、三带一对、三带一张单牌、三张牌。
  2. 考虑三顺子、双顺子、单顺子。
  3. 最后遍历所有的牌，剩一张就单张牌，剩两张就出对。
- 注意：火箭在这里没什么卵用，只需要看成一种对子就可以。
- 因为求的是最少次数，也可以适当性来一点最优性剪枝。
- 因为 $n \leq 23$，实际上搜一组数据也并不会太慢。

作业

- luogu 试炼场 pj 组的“深度优先搜索”专题做到绿。
- luoguP1784 数独（后面会用到）

bfs（广度优先搜索）

bfs（广度优先搜索）

- 树是有层次遍历的，就叫 bfs。在搜索上也有类似的 bfs。

bfs（广度优先搜索）

- 树是有层次遍历的，就叫 bfs。在搜索上也有类似的 bfs。
- bfs 需要使用队列实现，每次取出队头，进行扩展，扩展出的新节点在队尾进入。

bfs（广度优先搜索）

- 树是有层次遍历的，就叫 bfs。在搜索上也有类似的 bfs。
- bfs 需要使用队列实现，每次取出队头，进行扩展，扩展出的新节点在队尾进入。
- 给一个例子理解一下：

bfs 框架

```
void bfs(起点) {
    std::queue<类型> q;
    q.push(起点); vis[起点] = true;
    while(!q.empty()) {
        类型 x = q.front(); q.pop();
        for(扩展新节点) {
            if(!vis[新节点]) {
                q.push(新节点);
                vis[新节点] = true;
            }
        }
    }
}
```

例题 1：luoguP3956

题目描述

有一个$m \times m$的棋盘，棋盘上每一个格子可能是红色、黄色或没有任何颜色的。你现在要从棋盘的最左上角走到棋盘的最右下角。

任何一个时刻，你所站在的位置必须是有颜色的（不能是无色的），你只能向上、下、左、右四个方向前进。当你从一个格子走向另一个格子时，如果两个格子的颜色相同，那你不需要花费金币；如果不同，则你需要花费 1个金币。

另外，你可以花费 2 个金币施展魔法让下一个无色格子暂时变为你指定的颜色。但这个魔法不能连续使用，而且这个魔法的持续时间很短，也就是说，如果你使用了这个魔法，走到了这个暂时有颜色的格子上，你就不能继续使用魔法；只有当你离开这个位置，走到一个本来就有颜色的格子上的时候，你才能继续使用这个魔法，而当你离开了这个位置（施展魔法使得变为有颜色的格子）时，这个格子恢复为无色。

例题 1——分析

例题 1——分析

- 一看就知道要用 bfs 嘛。用 dfs 可能会爆栈。

例题 1——分析

- 一看就知道要用 bfs 嘛。用 dfs 可能会爆栈。
- 利用 bfs 的一个优良性质：当第一次遍历到某节点时，即是最优。

例题 1——分析

- 一看就知道要用 bfs 嘛。用 dfs 可能会爆栈。
- 利用 bfs 的一个优良性质：当第一次遍历到某节点时，即是最优。
- 所以我们从 (1,1) 初始状态开始 bfs，直到到达 (m, m) 为止，第一次搜到 (m, m) 即是最少的金币数。

例题 1——分析

- 一看就知道要用 bfs 嘛。用 dfs 可能会爆栈。
- 利用 bfs 的一个优良性质：当第一次遍历到某节点时，即是最优。
- 所以我们从 (1,1) 初始状态开始 bfs，直到到达 (m, m) 为止，第一次搜到 (m, m) 即是最少的金币数。
- 状态相比于普通 bfs 只需要再来一个 bool：判断是否用了膜法。

例题 1——分析

- 一看就知道要用 bfs 嘛。用 dfs 可能会爆栈。
- 利用 bfs 的一个优良性质：当第一次遍历到某节点时，即是最优。
- 所以我们从 (1,1) 初始状态开始 bfs，直到到达 (m, m) 为止，第一次搜到 (m, m) 即是最少的金币数。
- 状态相比于普通 bfs 只需要再来一个 bool：判断是否用了膜法。
- 状态向四个方向转移，若颜色相同无需代价，若颜色不同需 1 块。

例题 1——分析

- 一看就知道要用 bfs 嘛。用 dfs 可能会爆栈。
- 利用 bfs 的一个优良性质：当第一次遍历到某节点时，即是最优。
- 所以我们从 (1,1) 初始状态开始 bfs，直到到达 (m, m) 为止，第一次搜到 (m, m) 即是最少的金币数。
- 状态相比于普通 bfs 只需要再来一个 bool：判断是否用了膜法。
- 状态向四个方向转移，若颜色相同无需代价，若颜色不同需 1 块。
- 当现在踩着的不是用膜法变的时，可以考虑花 2 块变色并踩上去。

例题 1——分析

- 一看就知道要用 bfs 嘛。用 dfs 可能会爆栈。
- 利用 bfs 的一个优良性质：当第一次遍历到某节点时，即是最优。
- 所以我们从 (1,1) 初始状态开始 bfs，直到到达 (m, m) 为止，第一次搜到 (m, m) 即是最少的金币数。
- 状态相比于普通 bfs 只需要再来一个 bool：判断是否用了膜法。
- 状态向四个方向转移，若颜色相同无需代价，若颜色不同需 1 块。
- 当现在踩着的不是用膜法变的时，可以考虑花 2 块变色并踩上去。
- 这就是最朴素的 bfs 思路。

例题 1——分析

例题 1——分析

- 你以为这样就能过了？Too young!

例题 1——分析

- 你以为这样就能过了？Too young!
- 由于状态巨大，并且可能跑回去，实际上朴素 bfs 是有很大浪费的。

例题 1——分析

- 你以为这样就能过了？Too young!
- 由于状态巨大，并且可能跑回去，实际上朴素 bfs 是有很大浪费的。
- 我们可以记录一个 dist 数组，记录这种状态下最小金币花费。

例题 1——分析

- 你以为这样就能过了？Too young!
- 由于状态巨大，并且可能跑回去，实际上朴素 bfs 是有很大浪费的。
- 我们可以记录一个 dist 数组，记录这种状态下最小金币花费。
- 当转移后金币花费比最小金币花费还要贵的时候，不转移。只转移那些可能更优的状态。

例题 1——分析

- 你以为这样就能过了？Too young!
- 由于状态巨大，并且可能跑回去，实际上朴素 bfs 是有很大浪费的。
- 我们可以记录一个 dist 数组，记录这种状态下最小金币花费。
- 当转移后金币花费比最小金币花费还要贵的时候，不转移。只转移那些可能更优的状态。
- 给大家看看代码。

作业

- 把 luogu 试炼场 pj 组“广度优先搜索”做绿。

剪枝

剪枝

- 普通的搜索人人都会写，但是写出来的不一定跑得过。

剪枝

- 普通的搜索人人都会写，但是写出来的不一定跑得过。
- 在搜索的过程中可能会出现冗余状态，我们可以想办法删去。

剪枝

- 普通的搜索人人都会写，但是写出来的不一定跑得过。
- 在搜索的过程中可能会出现冗余状态，我们可以想办法删去。
- 这种把搜索中的冗余状态消除掉的做法，叫做剪枝 (prune)。

剪枝

- 普通的搜索人人都会写，但是写出来的不一定跑得过。
- 在搜索的过程中可能会出现冗余状态，我们可以想办法删去。
- 这种把搜索中的冗余状态消除掉的做法，叫做剪枝 (prune)。
- 剪枝是一项高级的运用，下面通过很多例题来了解一下。

例题 1：luoguP1433

题目描述

房间里放着n块奶酪。一只小老鼠要把它们都吃掉，问至少要跑多少距离？老鼠一开始在(0,0)点处。

例题 1——分析

例题 1——分析

- 震惊！堂堂 TSP 问题沦落为普及-！

例题 1——分析

- 震惊！堂堂 TSP 问题沦落为普及-！
- 如果用搜索的思路，就是一个回溯的问题了。

例题 1——分析

- 震惊！堂堂 TSP 问题沦落为普及-！
- 如果用搜索的思路，就是一个回溯的问题了。
- 自信地交上去，T 了 4 个点。

例题 1——分析

- 震惊！堂堂 TSP 问题沦落为普及-！
- 如果用搜索的思路，就是一个回溯的问题了。
- 自信地交上去，T 了 4 个点。
- 在这里介绍第一种剪枝：最优性剪枝。

例题 1——分析

- 震惊！堂堂 TSP 问题沦落为普及-！
- 如果用搜索的思路，就是一个回溯的问题了。
- 自信地交上去，T 了 4 个点。
- 在这里介绍第一种剪枝：最优性剪枝。
- 题目如果让你求最优答案，则大部分可以使用最优性剪枝：

例题 1——分析

- 震惊！堂堂 TSP 问题沦落为普及-！
- 如果用搜索的思路，就是一个回溯的问题了。
- 自信地交上去，T 了 4 个点。
- 在这里介绍第一种剪枝：最优性剪枝。
- 题目如果让你求最优答案，则大部分可以使用最优性剪枝：
- 如果当前的中途答案比当前最优解还要劣，那么剪枝。

例题 1——分析

- 震惊！堂堂 TSP 问题沦落为普及-！
- 如果用搜索的思路，就是一个回溯的问题了。
- 自信地交上去，T 了 4 个点。
- 在这里介绍第一种剪枝：最优性剪枝。
- 题目如果让你求最优答案，则大部分可以使用最优性剪枝：
- 如果当前的中途答案比当前最优解还要劣，那么剪枝。
- 具体在这道题，直接一行代码搞定。

例题 1——分析

- 震惊！堂堂 TSP 问题沦落为普及-！
- 如果用搜索的思路，就是一个回溯的问题了。
- 自信地交上去，T 了 4 个点。
- 在这里介绍第一种剪枝：最优性剪枝。
- 题目如果让你求最优答案，则大部分可以使用最优性剪枝：
- 如果当前的中途答案比当前最优解还要劣，那么剪枝。
- 具体在这道题，直接一行代码搞定。
- 听说是可以用模拟退火的，但是我写没过。

例题 2：luoguP1731

题目背景

7月17日是Mr.W的生日，ACM-THU为此要制作一个体积为Nπ的M层

生日蛋糕，每层都是一个圆柱体。

设从下往上数第i(1<=i<=M)层蛋糕是半径为Ri, 高度为Hi的圆柱。当i<M时，要求$R_i > R_{i+1}$且$H_i > H_{i+1}$。

由于要在蛋糕上抹奶油，为尽可能节约经费，我们希望蛋糕外表面（最下一层的下底面除外）的面积Q最小。

令Q= Sπ

请编程对给出的N和M，找出蛋糕的制作方案（适当的Ri和Hi的值），使S最小。

（除Q外，以上所有数据皆为正整数）

题目描述

例题 2——分析

例题 2——分析

- 爆搜的思路确实很好想，但是不敢想：状态太多了！

例题 2——分析

- 爆搜的思路确实很好想，但是不敢想：状态太多了！
- 结合前面的最优性剪枝，这里在介绍另外一种剪枝：可行性剪枝。

例题 2——分析

- 爆搜的思路确实很好想，但是不敢想：状态太多了！
- 结合前面的最优性剪枝，这里在介绍另外一种剪枝：可行性剪枝。
- 当我们根据当前状态已经能够推断绝对不可能比当前最优解更优，则剪枝。这就是可行性剪枝。

例题 2——分析

- 爆搜的思路确实很好想，但是不敢想：状态太多了！
- 结合前面的最优性剪枝，这里在介绍另外一种剪枝：可行性剪枝。
- 当我们根据当前状态已经能够推断绝对不可能比当前最优解更优，则剪枝。这就是可行性剪枝。
- 参考这两个大方向，我们就有 4 个剪枝：

例题 2——分析

例题 2——分析

1. 当前表面积比当前最小表面积大，剪枝。

例题 2——分析

1. 当前表面积比当前最小表面积大，剪枝。
2. 当前表面积加上最少的表面积还比当前最小表面积大，剪枝。最少的表面积可以拿半径都是 1 的情况（虽然不可能实现）。

例题 2——分析

1. 当前表面积比当前最小表面积大，剪枝。
2. 当前表面积加上最少的表面积还比当前最小表面积大，剪枝。最少的表面积可以拿半径都是 1 的情况（虽然不可能实现）。
3. 当前体积加上最大的体积还比 n 小，剪枝。当前最大的体积就是以当前的半径叠满为止（同样不可能实现，但不影响）。

例题 2——分析

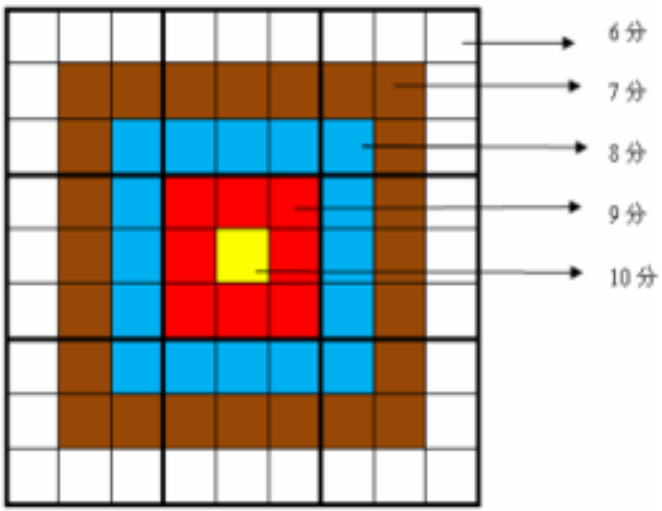
1. 当前表面积比当前最小表面积大，剪枝。
2. 当前表面积加上最少的表面积还比当前最小表面积大，剪枝。最少的表面积可以拿半径都是 1 的情况（虽然不可能实现）。
3. 当前体积加上最大的体积还比 n 小，剪枝。当前最大的体积就是以当前的半径叠满为止（同样不可能实现，但不影响）。
4. 把那些体积大于 n、叠 $m+1$ 层等非法情况掐掉。

例题 2——分析

1. 当前表面积比当前最小表面积大，剪枝。
2. 当前表面积加上最少的表面积还比当前最小表面积大，剪枝。最少的表面积可以拿半径都是 1 的情况（虽然不可能实现）。
3. 当前体积加上最大的体积还比 n 小，剪枝。当前最大的体积就是以当前的半径叠满为止（同样不可能实现，但不影响）。
4. 把那些体积大于 n、叠 $m+1$ 层等非法情况掐掉。
5. 最后就能过了。。。

例题 3：luoguP1074

靶形数独的方格同普通数独一样，在 9 格宽×9 格高的大九宫格中有9 个 3 格宽×3 格高的小九宫格（用粗黑色线隔开的）。在这个大九宫格中，有一些数字是已知的，根据这些数字，利用逻辑推理，在其他的空格上填入 1 到 9 的数字。每个数字在每个小九宫格内不能重复出现，每个数字在每行、每列也不能重复出现。但靶形数独有一点和普通数独不同，即每一个方格都有一个分值，而且如同一个靶子一样，离中心越近则分值越高。（如图）



上图具体的分值分布是：最里面一格（黄色区域）为 10 分，黄色区域外面的一圈（红色区域）每个格子为9分，再外面一圈（蓝色区域）每个格子为8 分，蓝色区域外面一圈（棕色区域）每个格子为7分，最外面一圈（白色区域）每个格子为6分，如上图所示。比赛的要求是：每个人必须完成一个给定的数独（每个给定数独可能有不同的填法），而且要争取更高的总分数。而这个总分数即每个方格上的分值和完成这个数独时填在相应格上的数字的乘积的总和

例题 3——分析

例题 3——分析

- 这道题其实更重要的并不是剪枝，而是你会不会解数独。。。

例题 3——分析

- 这道题其实更重要的并不是剪枝，而是你会不会解数独。。。
- 如何解数独？在前面 dfs 的作业就已经要求大家去学了。

例题 3——分析

- 这道题其实更重要的并不是剪枝，而是你会不会解数独。。。
- 如何解数独？在前面 dfs 的作业就已经要求大家去学了。
- 如果能解数独，那么这道题就要求你快速解出权值最大的解。

例题 3——分析

- 这道题其实更重要的并不是剪枝，而是你会不会解数独。。。
- 如何解数独？在前面 dfs 的作业就已经要求大家去学了。
- 如果能解数独，那么这道题就要求你快速解出权值最大的解。
- 这道题是有两个剪枝的：

例题 3——分析

- 这道题其实更重要的并不是剪枝，而是你会不会解数独。。。
- 如何解数独？在前面 dfs 的作业就已经要求大家去学了。
- 如果能解数独，那么这道题就要求你快速解出权值最大的解。
- 这道题是有两个剪枝的：
  1. 在空格较少的一边开始搜，状态会比从空格多的一边搜少得多。

例题 3——分析

- 这道题其实更重要的并不是剪枝，而是你会不会解数独。。。
- 如何解数独？在前面 dfs 的作业就已经要求大家去学了。
- 如果能解数独，那么这道题就要求你快速解出权值最大的解。
- 这道题是有两个剪枝的：
  1. 在空格较少的一边开始搜，状态会比从空格多的一边搜少得多。
  2. 最优性剪枝：设当前还没有填入的数字总和为 sum，当前的得分为 cur，最优得分为 best，如果 sum*10+cur<=best，那么已经没有继续搜索的必要了，剪枝.（来自 NOI 导刊）

例题 3——分析

- 这道题其实更重要的并不是剪枝，而是你会不会解数独。。。
- 如何解数独？在前面 dfs 的作业就已经要求大家去学了。
- 如果能解数独，那么这道题就要求你快速解出权值最大的解。
- 这道题是有两个剪枝的：
  1. 在空格较少的一边开始搜，状态会比从空格多的一边搜少得多。
  2. 最优性剪枝：设当前还没有填入的数字总和为 sum，当前的得分为 cur，最优得分为 best，如果 sum*10+cur<=best，那么已经没有继续搜索的必要了，剪枝.（来自 NOI 导刊）

作业

- luoguP1092 虫食算
- luogu 试炼场 TG 组的搜索 Ex 做绿