暴力出奇迹

n+*e*

Tsinghua University

2017 年 7 月 19 日

- ZLD 昨天已经给你萌介绍了什么是搜索，那么今天就来看看如何科学地：
  1. 写暴力
  2. ~~AK~~
  3. ~~进队~~
  4. ~~保送~~
  5. ~~走上人生巅峰~~

枚举 n 的全排列

```
1  void dfs(int x){
2      if(x==n+1){
3          for(int i=1;i<=n;i++)printf("%d␣",a[i]);
4          return;
5      }else
6          for(a[x]=1;a[x]<=n;a[x]++)
7              if(used[a[x]]==0){
8                  used[a[x]]=1;
9                  dfs(x+1);
10                 used[a[x]]=0;
11             }
12 }
13 int main(){scanf("%d",&n);dfs(1);}
```

- 现在只学 BFS 不行了，要学 SPFA（读作死趴法 233）
- 因为很多搜索问题的模型其实也是图论中的最短路模型。

- 现在只学 BFS 不行了，要学 SPFA（读作死趴法 233）
- 因为很多搜索问题的模型其实也是图论中的最短路模型。
- 演示一遍代码：给定 n 个点求点 s 到点 t 的最短路

```
1  struct Edge{int to,v,nxt;}e[MAXM];
2  int n,dist[MAXN],et,last[MAXN],q[1<<20],l,r,in[MAXN];
3  void add(int x,int y,int v)
4  {e[++et]=(Edge){y,v,last[x]},last[x]=et;}
5  int spfa(int s,int t){
6      memset(dist,63,sizeof dist);dist[s]=0,in[s]=1;
7      for(q[l=r=1]=s;l<=r;in[q[l++]]=0)
8          for(int i=last[q[l]];i;i=e[i].nxt)
9              if(dist[e[i].to]>dist[q[l]]+e[i].v){
10                 dist[e[i].to]=dist[q[l]]+e[i].v;
11                 if(!in[e[i].to])in[q[++r]=e[i].to]=1;
12             }
13     return dist[t];
14 }
```

- 一般来说，搜索题目中的模型，所有的边权均为 1，比如下面这个问题。因此这份代码对于特定的问题而言，有不少地方是可以不用写的。
- 接下来我们来画一张迷宫再来演示一发。针对迷宫而言，每个节点的出边可以直接计算出来，并且所有边权 =1，因此没有必要用边表存储所有边的信息。

存储

- 个人建议是用类似边表的结构，方便插入和查找，还能存储其他信息

```
1 struct E{//edge
2     int key, next, num, ...;
3 }e[Maxn];
4 int last[Maxn], et = 0;
```

插入

- 将元素 x 经过函数处理后得到 key, 然后把 x 和 key 塞到边表里面

```
1  void insert (int x, int key) {
2      int i, flag = 0;
3      for (i = last[key]; i && !flag; i = e[i].next)
4          if (e[i].key == x) e[i].num++, flag = 1;
5      if (!flag) {
6          e[++et] = (E) {x, last[key], 1};
7          last[key] = et;
8      }
9  }
```

查找

```
1 int find (int x, int key) {
2     for (int i = last[key]; i; i = e[i].next)
3     if (e[i].key == x) return 1;
4     return 0;
5 }
```

- 现在是怎么构建一个函数，把 x 弄成 key 的问题了
- 数：直接找个大数 mod 了。这里如果数是随机的话，没有必要找一个质数
  如果怕出事，平时随便找个质数顶上去就好了。戳我

- 现在是怎么构建一个函数，把 x 弄成 key 的问题了
- 数：直接找个大数 mod 了。这里如果数是随机的话，没有必要找一个质数
  如果怕出事，平时随便找个质数顶上去就好了。戳我
- 坐标 & DP 状态 & 搜索状态：$i \times n + j$、$(i \times n + j) \times m + k$。
  表达式越简单越好

- 现在是怎么构建一个函数，把 x 弄成 key 的问题了
- 数：直接找个大数 mod 了。这里如果数是随机的话，没有必要找一个质数
  如果怕出事，平时随便找个质数顶上去就好了。戳我
- 坐标 & DP 状态 & 搜索状态：$i \times n + j$、$(i \times n + j) \times m + k$。表达式越简单越好
- 字符串 & 数组：转成 H 进制 (H 可以取 $26, 131, 13131 \cdots$) 或者类 H 进制 (就是每个位置再加一个权), 转成一个数字
  取中间的一段: hash 要支持区间减法。演示？类似“差分”与“前缀和”

- 现在是怎么构建一个函数，把 x 弄成 key 的问题了
- 数：直接找个大数 mod 了。这里如果数是随机的话，没有必要找一个质数
  如果怕出事，平时随便找个质数顶上去就好了。戳我
- 坐标 & DP 状态 & 搜索状态：$i \times n + j$、$(i \times n + j) \times m + k$。
  表达式越简单越好
- 字符串 & 数组：转成 H 进制 (H 可以取 $26, 131, 13131 \cdots$) 或者类 H 进制 (就是每个位置再加一个权), 转成一个数字
  取中间的一段: hash 要支持区间减法。演示？类似“差分”与“前缀和”
- 图 & 树: 找重心, 最小表示法, $\cdots$ 不展开
  你们以后会明白的, 现在暂时用不到

解决冲突的方法

1. mod 的数变大 (int 改 long long, 效率变慢, 空间消耗变大)
2. 双 hash, 三 hash (同上, 至今不会算碰撞概率, 玄学)
3. **增加信息**: 最无脑最好用

- 在搜索的时候，搜索出来的树太大了
- 从起始状态和目标状态同时开始搜索一定层数
- 把搜索出来的所有状态扔到 hash 表里面，看看有没有重复的
- 能够提升一倍答案的效率，比如原来复杂度是 $O(2^n)$，现在可以变成 $O(2^{n/2})$，相当于复杂度开根号。
- 画张图

- 听起来很厉害的样子
- 一切都源于——《靶型数独》？

- 听起来很厉害的样子
- 一切都源于——《靶型数独》？
- DLX 的理念：动态减少搜索中的状态数
- 用链表来减少对无效内存的访问

- 听起来很厉害的样子
- 一切都源于——《靶型数独》？
- DLX 的理念: 动态减少搜索中的状态数
- 用链表来减少对无效内存的访问
- 具体代码一搜一大堆
- 至今没写过 (捂脸跑)

- 01 矩阵化, 选取一个行的集合, 使得集合中每一列都恰好包含一个 1
- 行: 所有位置 × 所有可能情况
- 列: 各种约束条件
- 怎么填 1: 第 i 行的情况满足第 j 个条件约束, 则 $A[i][j] = 1$, 否则 $A[i][j] = 0$

N 皇后

- 行: $N \times N$
- 列: $N + N + 2N - 1 + 2N - 1 = 6N - 2$。$N$ 行, N 列, $2N - 1$ 左斜线, $2N - 1$ 右斜线
- 矩阵的每行都有 4 个 1, 分别分布在行域, 列域, 左斜线域, 右斜线域

N 皇后

- 行: $N \times N$
- 列: $N+N+2N-1+2N-1=6N-2$。N 行, N 列, $2N-1$ 左斜线, $2N-1$ 右斜线
- 矩阵的每行都有 4 个 1, 分别分布在行域, 列域, 左斜线域, 右斜线域
- 在编程求解这个问题时, 需要做一点变通, 因为左斜线域, 右斜线域的列不可能被全部覆盖, 因此只需行域和列域被完全覆盖就算找到问题的一个解了。
- Tips: 可以调整列的顺序来加快搜索速度 (行列行列 $\cdots$)

Sudoku

- 行: $N \times N \times N$。因为一共 $N \times N$ 个小格, 每个小格有 N 中可能性 $(1 - N)$, 每一种可能对应这一行。
- 列: $(N + N + N) \times N + N \times N$。其中前面 3 个 N 分别代表着 N 行 N 列和 N 小块, 乘以 N 表示 N 中可能, 每种可能只可以选一个。$N \times N$ 表示 $N \times N$ 个小格, 限制每一个小格只可以放一个地方。
- 如果一个位置已经确定, 则只插入一行, 否则插入 $N \times N$ 行, 代表 $N \times N$ 种可能。

Puzzle

- 行: 拼图个数 × 每个拼图能够放置的位置数
- 列: 拼图个数 + 格子个数. 这两个都要精确覆盖, 也就是有且仅有一个 1
- *NOI*2005 智慧珠游戏

顾名思义

Iternative Deepening A*

↓

迭代加深 A* 算法

- 给出一个限制 bound, 规定当搜索层数 $>$ bound 时直接剪枝
- 在最外层 for(bound), 如果无解就继续

- 给出一个限制 bound, 规定当搜索层数 $>$ bound 时直接剪枝
- 在最外层 for(bound), 如果无解就继续
- 缺点: 重复计算

- $f(n) = g(n) + h(n)$
- $g(n)$: 从起始状态到当前状态 n 的代价
- $h(n)$: 从当前状态 n 到目标状态的**估计**代价
- 估价函数 $h(n)$ 若选取不当, 则可能找不到解, 或找到的解也不是最优解.
- 又 Wa 又 T 一时爽:(已加密)

- 定义 $h^*(n)$ 为从当前状态 n 到目标状态的**实际**代价
- 必须满足 $h(n) \leq h^*(n)$, 否则嘿嘿嘿
- 估计总是过于乐观的

$h(n)$ 的相容

- 如果 h 函数对任意状态 s_1 和 s_2 还满足

$$h(s_1) \leq h(s_2) + c(s_1, s_2)$$

- $c(s_1, s_2)$ 是 s_1 转移到 s_2 的步数，则称 h 是相容的.
- h 相容能确保随着一步步往前走，f 递增，这样 A* 能更高效找到最优解.

$h(n)$ 怎么写?

- 不要太过分就好. 一般来说没问题的

$h(n)$ 怎么写?

- 不要太过分就好. 一般来说没问题的
- 脑洞要大

A*?

- 比 IDA* 还麻烦……
- 听说还要 hash 判重和堆维护？每次要选取 $f(n)$ 最小的更新
- 时间和空间都不行

- 在一般的问题中是这样使用 IDA* 算法的
- 当前局面的估价函数值 $h(n)$ + 当前的搜索深度 $g(n)$ > 预定义的最大搜索深度 bound 时，就停止继续往下搜索

- 在一般的问题中是这样使用 IDA* 算法的
- 当前局面的估价函数值 $h(n)$ + 当前的搜索深度 $g(n)$ > 预定义的最大搜索深度 bound 时，就停止继续往下搜索
- 写成代码就是

```
if dep + h() > bound then return;
```

- 没了

Advantages

- 省时间省空间
- 试一下挺好写的

```
1  int dfs (int dep) {
2  //dfs 返回大于 bound 的局面中最小的 f(n), 这样的话 bound
       就不用一个个 for 了, 不然浪费时间
3      int hv = state.h();
4      if (hv == 0) return flag = 1, dep; //找到解
5      if (dep + hv > bound) return dep + hv;
6      int next_bound = 1000;
7      for (int i = 0; i < 4; i++) {
8          calc(new_state); int tmp = dfs(dep+1);
9          if (flag) return tmp; //找到解
10         if (tmp < next_bound) next_bound = tmp;
11     }
12     return next_bound;
13 }
14 for (bound = state.h(); !flag; bound = dfs(0));
```

运用

- 各种游戏求最少步数，普通搜索会爆炸
- 有的时候不一定非要用 IDA*，可能双向广搜会更快。
- **要学会选择合适的算法**

- S 是起始状态，T 是目标状态，n 是当前状态
- 不加 * 表示估计，加 * 表示是实际值
- g*(n)=S 到 n 所花费的实际最小代价（步数）
- h*(n) 表示 n 到 T 所花费的实际最小代价（步数）
- 搜索算法按照 f(n) 从小到大的顺序拓展新节点

BFS g(n)=g*(n),h(n)=0,f(n)=g(n)

DFS g(n)=0,h(n) 足够大，f(n)=h(n)

A* g(n)=g*(n),f(n)=g(n)+h(n)

- 已知一个 n 元高次方程：

$$k_1 x_1^{p_1} + k_2 x_2^{p_2} + \ldots + k_n x_n^{p_n} = 0$$

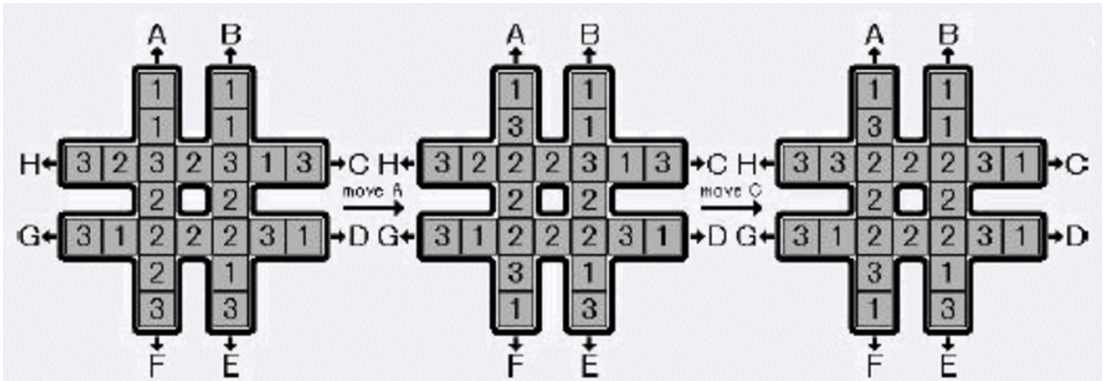
其中：$x_1, x_2, \cdots, x_n$ 是未知数，$k_1, k_2, \cdots, k_n$ 是系数，$p_1, p_2, \cdots, p_n$ 是指数。且方程中的所有数均为整数

- 假设未知数 $1 \le x_i \le M, i = 1 \cdots n$，求这个方程的整数解的个数。
- $1 \le n \le 6, 1 \le M \le 150, p \le 4$，方程的整数解的个数小于 2^{31}。
- 保证 $\left|k_1 M^{p_1}\right| + \left|k_2 M^{p_2}\right| + \ldots + \left|k_n M^{p_n}\right| < 2^{31}$
- 对于部分数据，$n \le 3$

- 看到 n 这么小，对，就是 dfs
- 常规的做法是，枚举每个 $x_i =?$，时间需要 $150^6 = 11390625000000$，用超算跑还是会 T 的 2333
- 怎么办呢？

- 看到 n 这么小，对，就是 dfs
- 常规的做法是，枚举每个 x_i =?，时间需要 $150^6 = 11390625000000$，用超算跑还是会 T 的 2333
- 怎么办呢？
- 30% 的数据，复杂度为 $150^3 = 3375000$，我们可不可以把 6 降成 3？
- 当然可以的啦，把 6 个数前三个分一组，后三个分一组，把前三个数和后三个数所有可能的情况都搞出来，如果前三个数的某个和与后三个数的某个和的和为 0，就会贡献答案。
- 于是开个哈希表，记录前三个数加起来和为 x 的方案数，这一步来一个 dfs，然后下一步再用一个 dfs 找后三个数加起来的和为-x，这就叫做双向搜索。

- 一个井字形棋盘，上面有 24 个格子 (如下图)。这些格子上面有 1,2,3 三种数字，且每种数字有 8 格。一开始，这些格子上的数字是随机分布的。你的任务是移动这些格子使得中间 8 个格子的数字相同。
- 有 8 种移动方式，分别标记为 A 到 H，可以理解为拉动 4 条链，如图的变换为 “AC”。问至少需要多少次拉动，才能从初始状态到达目标状态?(保证数据有解)
- 对于 100% 的数据，最大移动次数 ≤ 12，数据组数 ≤ 30。



- 已经连续两年的夏令营出过了
- 既可以双搜又可以 IDA*
- 下面请各位巨巨开始你萌的表演 ~

						1		
		1	7	4		9	5	
				8		7	4	2
					5		8	
4		7				5		3
	9		4					
2	6	4		3				
	8	9		5	7	4		
		5						

- 我都说了我不会写 DLX……

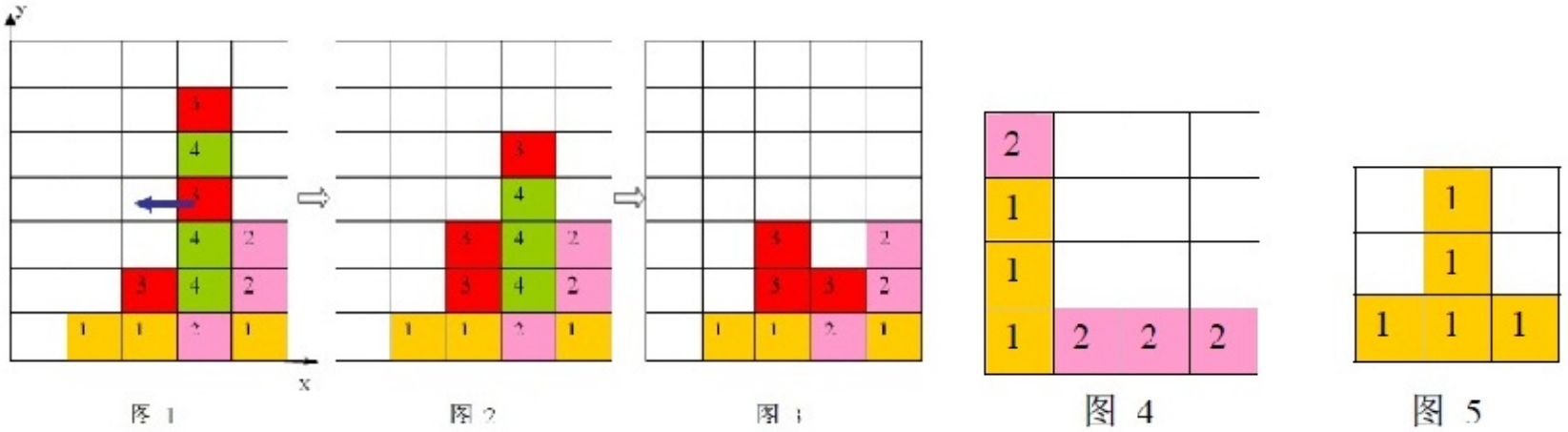
- 我都说了我不会写 DLX……
- 常见的搜索优化: 调整搜索序, 二进制压位, 多加剪枝, 估价 (还要多少步才能达到最终局面, 如果以最优解法达到最终局面的步数还是比当前最优解大就 cut), 迭代加深

- 我都说了我不会写 DLX……
- 常见的搜索优化: 调整搜索序, 二进制压位, 多加剪枝, 估价 (还要多少步才能达到最终局面, 如果以最优解法达到最终局面的步数还是比当前最优解大就 cut), 迭代加深
- 调整搜索序: DLX 是动态调整搜索序, 于是我就搞了一个静态调整搜索序
- 二进制压位: and, or, xor 来调整状态, 省去访问数组的时间, 比链表还快
- 剩下两个: IDA*, 不过我没用到

Mayan puzzle 是最近流行起来的一个游戏。游戏界面是一个 7 行 5 列的棋盘，上面堆放着一些方块，方块不能悬空堆放，即方块必须放在最下面一行，或者放在其他方块之上。游戏通关是指在规定的步数内消除所有的方块，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）拖动某一方块一格：当拖动这一方块时，如果拖动后到达的位置（以下称目标位置）也有方块，那么这两个方块将交换位置（参见图 6 到图 7）；如果目标位置上没有方块，那么被拖动的方块将从原来的竖列中抽出，并从目标位置上掉落（直到不悬空，参见图 1 和图 2）；

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）。
   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。



上面图 1 到图 3 给出了在棋盘上移动一块方块之后棋盘的变化。棋盘的左下角方块的坐标为 $(0, 0)$，将位于 $(3, 3)$ 的方块向左移动之后，游戏界面从图 1 变成图 2 所示的状态，此时在一竖列上有连续三块颜色为 4 的方块，满足消除条件，消除连续 3 块颜色为 4 的方块后，上方的颜色为 3 的方块掉落，形成图 3 所示的局面。

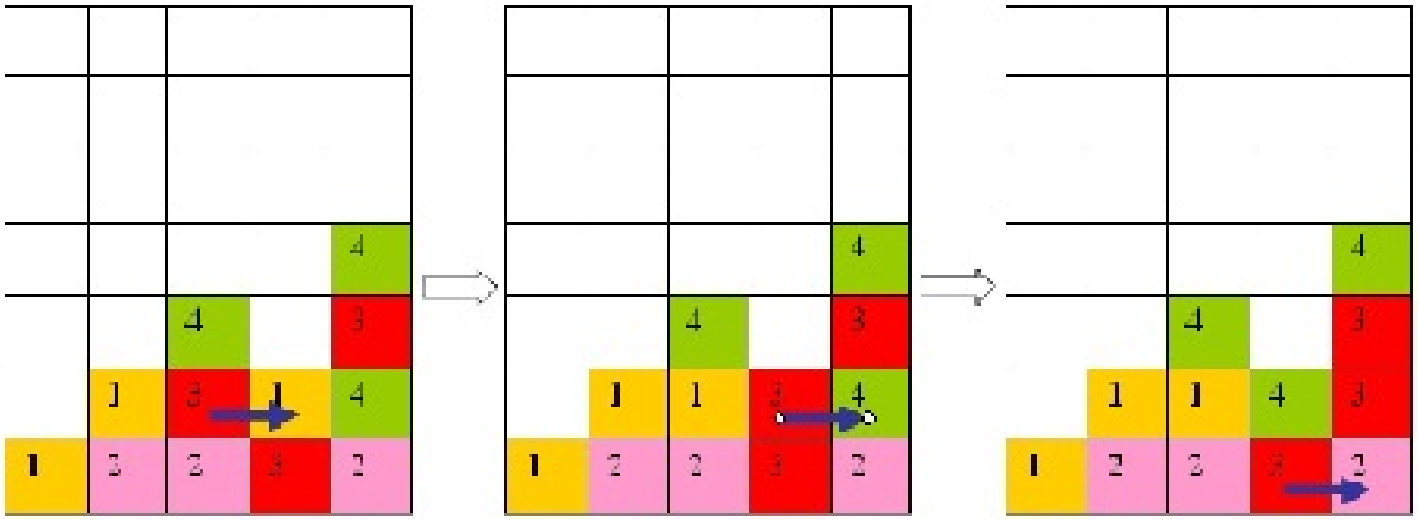


图 6　图 7　图 8

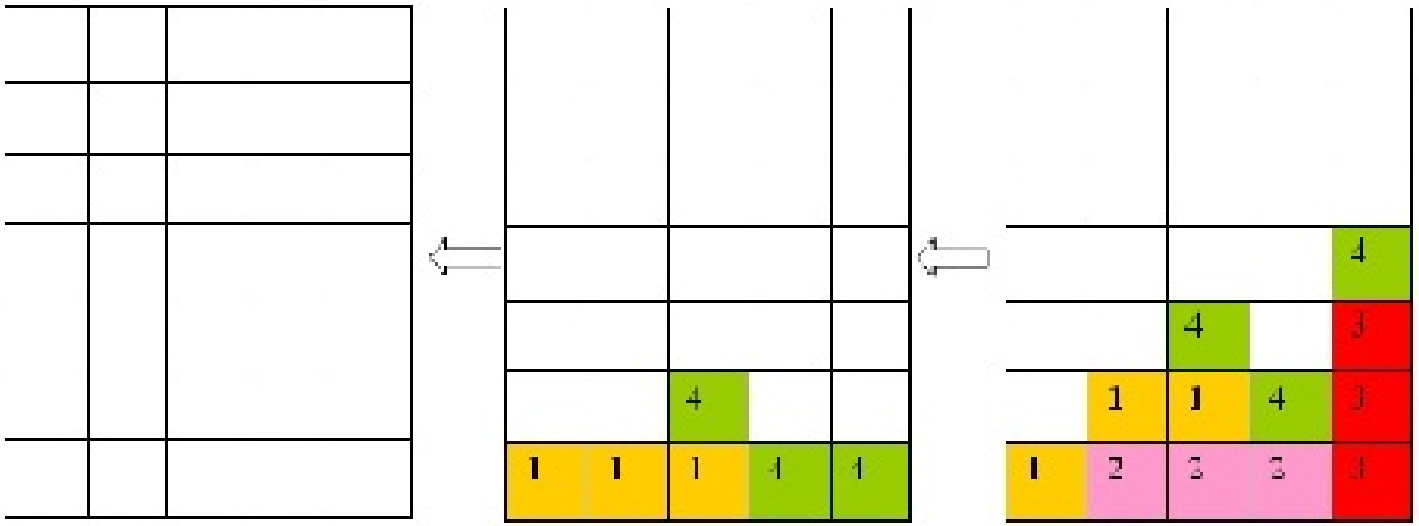


图 11　图 10　图 9

模拟搜索。因为 3s 时限。因为题目要求输出字典序最小解，于是就可以按题目中所给的第一、二、三关键字进行搜索，一旦找到一个解直接输出。

对于两个相邻方块 A (x，y)，B(x+1，y) 而言，

1. 若 A=B，移动无意义，return；
2. A 与 B 如果之前已经交换过了，则不要再交换
3. A 向左交换与 B 向右交换是等价的，于是只要枚举向右交换即可
4. 当 A=0 时，可以考虑将 B 左移

对于整个界面而言：

5. 如果当前界面中某个颜色只有 1 块或 2 块，不用继续做
6. 如果当前的局面已经在之前搜过了（hash 判重），return

剪枝就是这些