# 2. 递归问题、数学归纳法

张桄玮(gwzhang@cug.edu.cn)

郑州一中(Legacy)

2024-08-02

# 大问题化为小问题

# Uva177 折纸痕



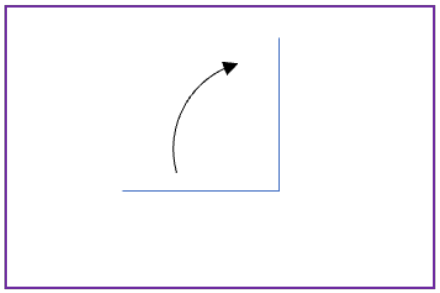
问题：你喜欢这折纸吗？给你一张很大的纸，对折以后再对折，再对折……每次对折都是从右往左折，因此在折了很多次以后，原先的大纸会变成一个窄窄的纸条。现在把这个纸条沿着折纸的痕迹打开，每次都只打开“一半”，即把每个痕迹做成一个直角，那么从纸的一端沿着纸面平行的方向看过去，会看到一个美妙的曲线。

观察: 折纸的过程是重复的 $\rightarrow$ 带来的图像也是重复的.

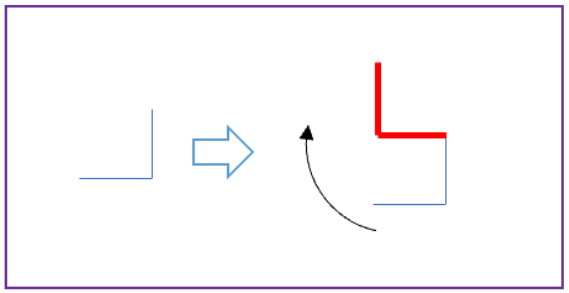
- 绕着最后一次的内容旋转 90 度就行了!

# Uva177 折纸痕

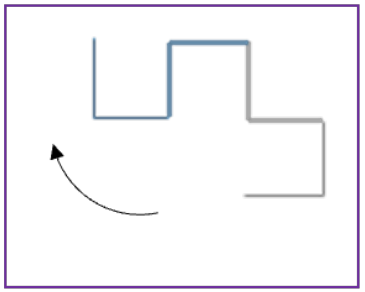




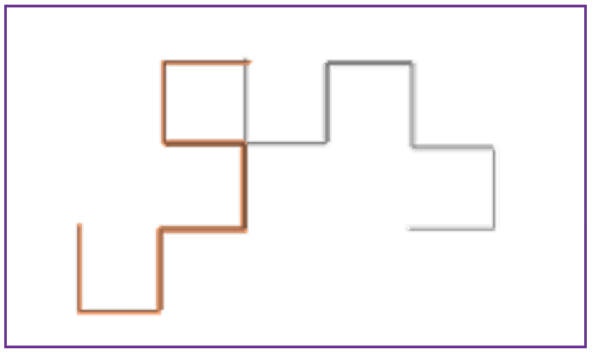
折1次



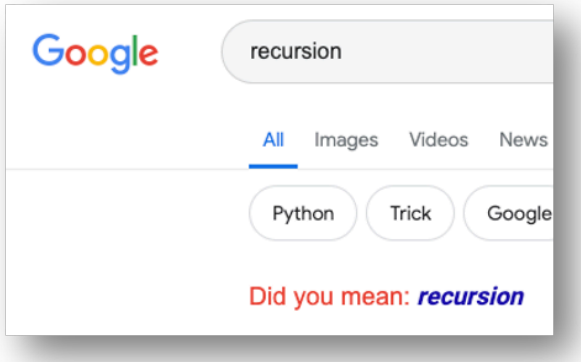
折2次



折3次



折4次




- 问题: 如何刻画屏幕上面的旋转?

考虑: 只要知道点, 知道点怎么连成线就好了!

# Uva177 折纸痕

- 怎么表示点?
  - 使用$(x, y)$坐标即可

```
struct Point {
    int x, y;
    Point(int x = 0, int y = 0) : x(x), y(y) {}
};
```

- 怎么表示线段? 线段=起点+$(x, y)$
  - 往$x$走多少, 往$y$走多少

```
struct Vector {
    int x, y;
    Vector(int x = 0, int y = 0) : x(x), y(y) {}
};
```

- 简化方式: 不是一模一样吗? `typedef Point Vector.`

# Uva177 折纸痕

为什么抽象出向量: 可以不管起点, 方便统一考虑.

- 加法, 减法: 一个人先走了$(2,3)$, 再走了$(4,-1)$, 现在他在哪?
- 数乘: 相当于拉伸走的距离; 乘负数相当于反向

如: 点绕着另一点顺时针旋转 90 度

- 把点和绕着的那个点连起来得到向量$(x,y)$, 然后得到$(y,-x)$
- 然后挪到起点就好了

偷懒: 扫描所有的线段, 把$x$坐标放大 1 倍(留出空隙), $y$坐标

# P5657 Gray 码

- 递归的构造过程

| | | |
|---|---|---|
| 1位的Gray码 | 0 | 1 |
| 2位的Gray码 | 00 01 | 11 10 |

当前的 ← 上一个阶段

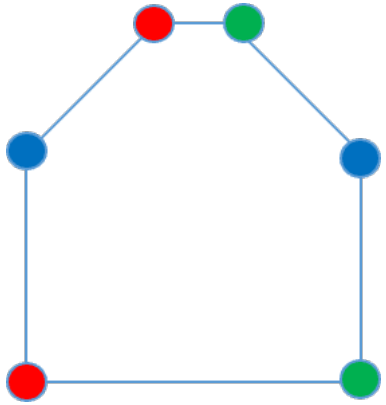
- 前半段: 开头位置是 0
- 后半段: 开头位置是 1, 但是判断顺序的方式要反过来

一直判断下去, 直到只有 1 位的情况.

# URAL1181 切涂色的多边形

问题：题目大意：

- 给定一个$N$边形，任意顶点只可能有 3 种颜色：R,G,B，至少出现 1 次.
- 任意两个相邻顶点不同色.
- 分割，使得分为$N-2$个三角形，使得三角形三个顶点不同色



特殊情况：

# URAL1181 切涂色的多边形



- 某种颜色的顶点有且仅有 1 个
  - 用任意相邻两顶点 + 独特颜色的顶点构成三角形

否则, 必有连续的三个顶点两两颜色不同

- 反证法(奇偶分类讨论; 不然限制条件就太强了)
- 用这连续的 3 个顶点构成三角形

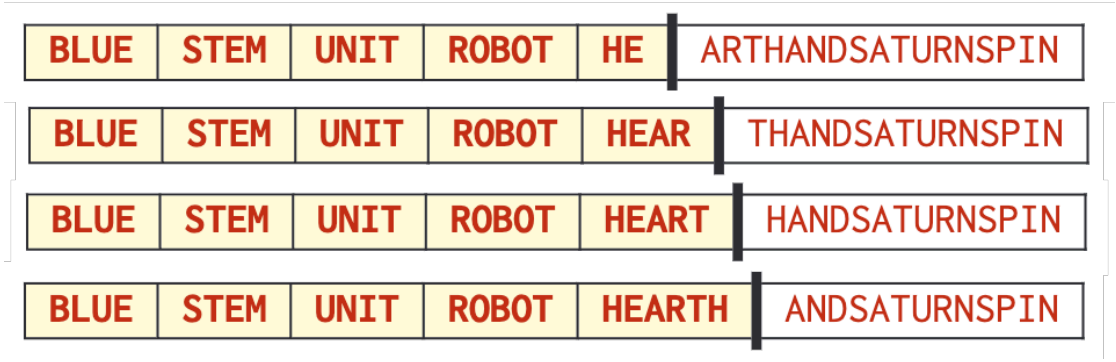
# 经典例子: 文本划分

问题: 有一列文本$a[]$, 有函数 isword, 问这一系列文本是否可以被分为若干个 word 的集合?

- 非常麻烦的想法: 枚举每一个词语的拼接, 然后检查和原文是不是一样的.

考虑子问题:

- $A[i..j]$是不是可以划分?
- 例如有词语 HE, HEAR, HEART, HEARTH



# 经典例子: 文本划分



- 最基础的情况: 长度为 0, 返回 OK.

修改: 每一次并没有修改整个字符串, 因此用下标代指当前位置.

# 经典例子: 最长递增的子序列



问题: 给定输入 $A[1..n]$, 计算元素递增的最长可能序列。

决策过程

- 对于每一个 $j$, 从 1 到 $n$ 顺序遍历
- 判断是否将 $A[j]$ 包含在序列中


以后可能会见到 “转移方程”:

# 经典例子: 最长递增的子序列



$$\text{LISbigger}(i,j) = \begin{cases} 0 & \text{if } j > n \\ \text{LISbigger}(i, j+1) & \text{if } A[i] \geq A[j] \\ \max\left\{ \begin{array}{c} \text{LISbigger}(i, j+1) \\ 1 + \text{LISbigger}(j, j+1) \end{array} \right\} & \text{otherwise} \end{cases}$$

无非就是代码的更紧凑的表达罢了.

```
int LISBigger(vector<int>& A, int i, int j, int n) {
    if (j >= n) {return 0;}
    else if (A[i] >= A[j]) {
        return LISBigger(A, i, j + 1, n);
    } else {
        int skip = LISBigger(A, i, j + 1, n);
        int take = 1 + LISBigger(A, j, j + 1, n);
        return max(skip, take);
    }
}
```

# 经典例子: 最长递增的子序列

## 关于最长递增子序列的另一种看法

考虑如果有下一个输出的话, 下一个输出元素的位置在哪?

考虑子问题: 给一个下标$i$, 找到在$A[i..n]$最长的递增子序列, 以$A[i]$开始.

```
int LISfirst(const vector<int>& A, int i) {
    int best = 0; // 初始化 best 为 0

    for (int j = i + 1; j < A.size(); ++j) {
        if (A[j] > A[i]) {
            best = max(best, LISfirst(A, j));
        }
    }

    return 1 + best;
}
```

# 递归的结构: 以二叉树为例

# 基本定义

**定义 01** (二叉树):

- 二叉树要么为空;
- 要么由根节点, 左子树(是一个二叉树), 右子树(也是一个二叉树)组成.

如何表示?

- 方法 1: 编号
  - “有组织, 有计划地"编号(按照满的情况)
    - 节点编号$o$的左孩子是$2o$, 右孩子是$2o+1$.
  - 像链表一样编号
    - `left[i]:=`节点$i$的左边的那个节点的编号是什么
    - `right[i]:=`节点$i$的右边的那个节点的编号是什么
- 方法 2: 动态分配内存(略)

# UVA679 小球下落

问题: 有一棵二叉树, 最大深度为 $D$, 且所有叶子的深度都相同。所有结点从上到下从左到右编号为 $1, 2, 3, ..., 2^D - 1$ 。在结点 1 处放一个小球, 它会往下落。每个内结点上都有一个开关, 初始全部关闭, 当每次有小球落到一个开关上时, 状态都会改变。当小球到达一个内结点时, 如果该结点上的开关关闭, 则往左走, 否则往右走, 直到走到叶子结点。

一些小球从结点 1 处依次开始下落, 最后一个小球将会落到哪里呢? 输入叶子深度 $D$ 和小球个数 $I$, 输出第 $I$ 个小球最后所在的叶子编号。假设 $I$ 不超过整棵树的叶子个数。$D \leq 20$ 。输入最多包含 1000 组数据。

朴素想法: 一个一个模拟!

# UVA679 小球下落

计算得到: 模拟会超时

- 下落总层数: $2^{19} \times 19 = 9961472 \sim 10^7$, 有 $10000$ 组数据
- $10^{11}$的数量级难以承受!

找规律: 都从根节点放

- 前几个形成左, 右, 左, 右的情形.
- 考虑递归! 化为了一个更小的子问题(在子树, 树顶的开关开/关)
  - 对于编号$I$的小球,
    - 奇数 $\rightarrow$ 往左走的$\left\lceil \frac{I}{2} \right\rceil$的球
    - 偶数 $\rightarrow$ 往右走的$\frac{I}{2}$的球

模拟最后一个球就好了.

# UVA122 树的层次遍历

问题: 给定一个二叉树, 请你按层次遍历. 最深 255 层.

无法再次按照 “规定” 编号! ($2^{255} \approx 10^{77}$).

考虑来一个分配一个, 使用内存池技术.

例如, 单个节点:

```
struct Node {
    bool have_value;
    int v; // 值
    int left, right; // 当前节点的左/右边节点的编号是什么?
    Node() : have_value(false), left(-1), right(-1) {}
};
Node nodes[maxNodes];  // 节点构成的池子
int nodeCount;  // 用来记录当前节点的数量
```

# UVA122 树的层次遍历

如何层序遍历?

- 如何遍历?
  - 把起始点丢进包里
  - 只要还能从包里面拿出来东西:
    - 标记拿出来的东西为访问过了
    - 把能由这个东西到达的所有东西(不包括自己)找出来放到包里

包的实现: 栈=DFS; 队列=BFS.

# 递归遍历

PreOrder($T$) = $T$ 的根结点 + PreOrder($T$ 的左子树 ) + PreOrder($T$ 的右子树 );

InOrder(T) = InOrder(T 的左子树 ) + $T$ 的根结点 + InOrder($T$ 的右子树)

PostOrder( $T$ )=PostOrder( $T$ 的左子树 ) + PostOrder($T$ 的右子树 +$T$ 的根结点

问题: 上述的遍历顺序, 给出哪个(哪些), 才可以完整确定一个二叉树?

答案: (Preorder/Postorder)+Inorder

- 原因: 一个子树在区间里面是连续的;

# 递归遍历

  - 先根据后序遍历找到树根, 再中序遍历中找到树根 → 左右节点的列表

- 没有 “终结符”, 搞不清楚左右孩子从哪里开始

**UVA548 树**

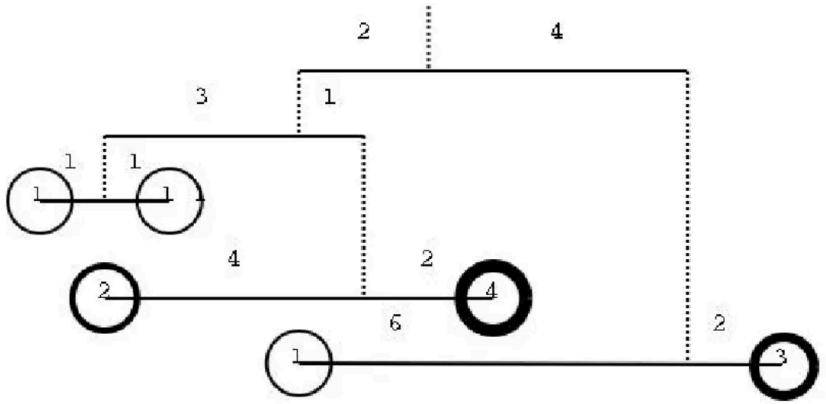
> 问题: 输入一个二叉树的中序和后序遍历，请你输出一个叶子节点，该叶子节点到根的数值总和最小，且这个叶子是编号最小的那个。

- 通过递归建立二叉树
- 找到总和最小的那个

# UVA839 天平

问题: 输入一个树状天平, 根据力矩相等原则判断是否平衡。所谓力矩相等,就是 $W_lD_l = W_rD_r$, 其中 $W_l$ 和 $W_r$ 分别为左右两边砝码的重量, $D$ 为距离。

采用递归 (先序) 方式输入: 每个天平的格式为 $W_l, D_l, W_r, D_r$, 当 $W_l$ 或 $W_r$ 为 0 时,表示该 “砝码” 实际是一个子天平, 接下来会描述这个子天平。当 $W_l = W_r = 0$ 时, 会先描述左子天平, 然后是右子天平。



# UVA839 天平

- 基本情况: 两边挂的是砝码;
- 递归情况: 已经平衡的天平可以看做一个重物(整体法)

那就存个二叉树然后从下往上算一遍不就好了嘛!

遇见的问题: 需要返回 2 个值 – 是否平衡, 以及这一坨视为一个整体多重

怎么办?

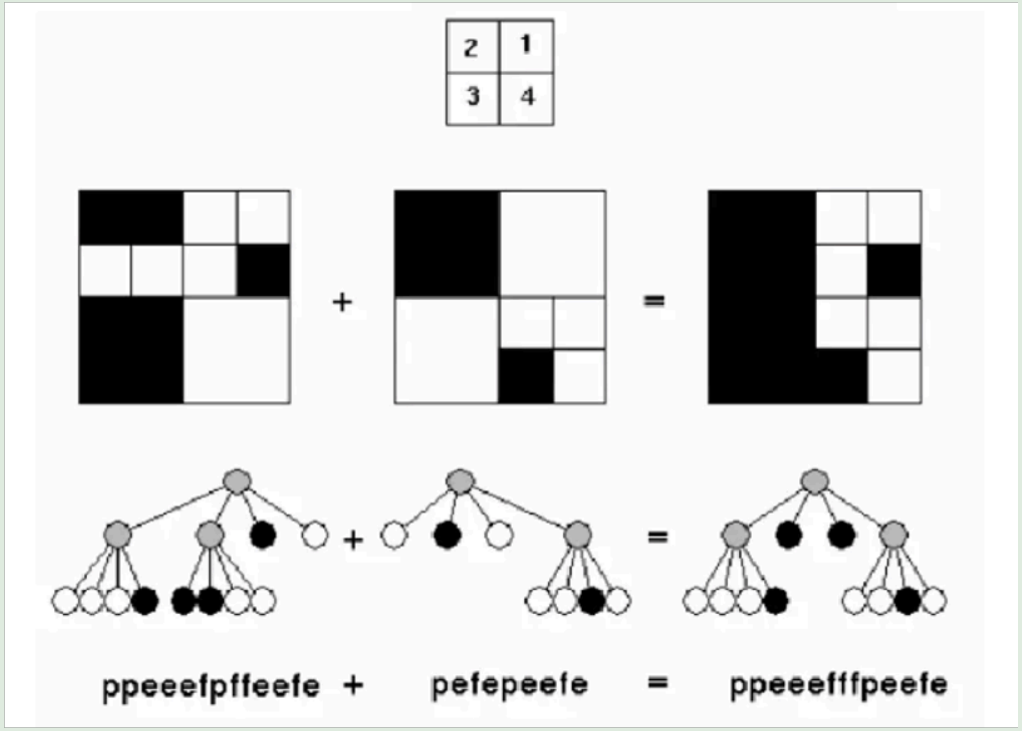
- 返回一个 pair
- 使用引用

有没有发现根本不用费劲建树就可以通过(?)

- 输入就是先序排列的呀!

# Uva297 四分树

问题:




问题: 两个四分树的和.

# Uva297 四分树

这玩意有个递归定义: 一个高度为$h$, 宽度为$w$的四分树$(h, w)$可以表示以为:

- 白色的节点/灰色的节点
- 灰色的节点(四分树, 四分树, 四分树, 四分树).

大小都是 h/2,w/2

问题: 这东西为啥就不用知道前序/后序就可以决定了?

- 因为有足够多的终结符指示现在递归的顺序是啥.

# 花这么大劲研究二叉树干什么

任何树都同构与二叉树

- 左孩子有兄弟表示法

多叉树: 可以用不定多少叉的链表表示.

# 调试递归程序

原则

- 只要初始状态好
- 递归状态转移好(一层足以说明)
- 求的没写错

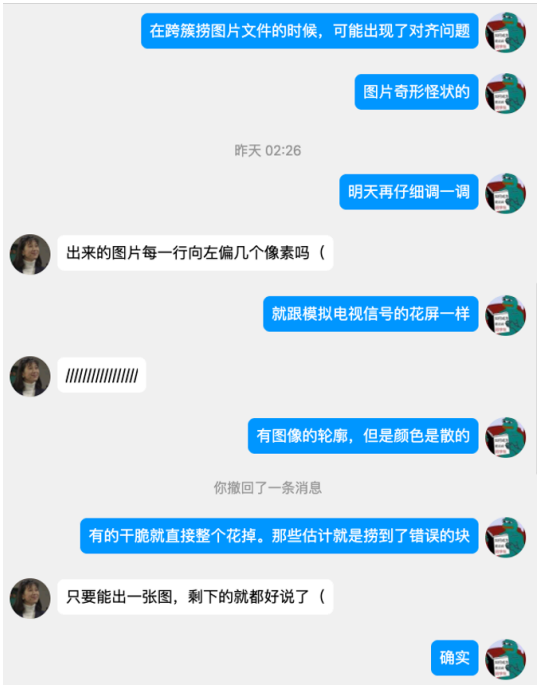
程序就大差不差

推论

- 在入口处打印当前递归调用的参数
- gdb 使用 bt(backtrace)显示栈帧

“只要能出一张图, 剩下的就好说了

——老代码(行为)艺术家,小木曾せつな Official(Bilibili)”



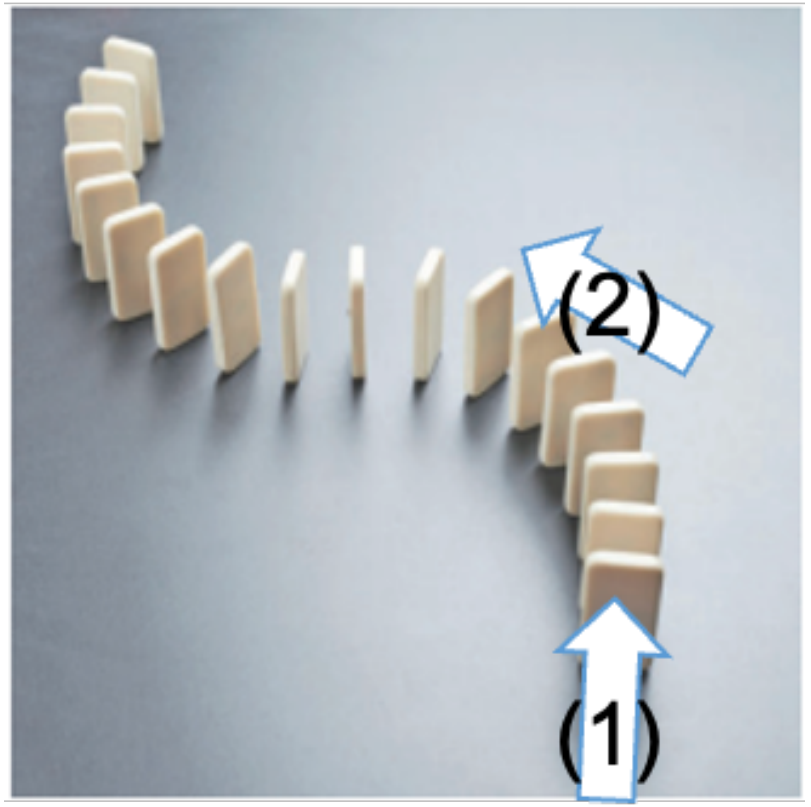
# 数学归纳法, 结构归纳法以及其他

# 来自同学的评论


背景: 科研实验中, 在调一个难搞的 bug, 最后用了上述两种办法搞定了...

# 数学归纳法

1. 归纳奠基(basis): 证明出最初的一个命题正确
2. 递推(induction): 证明出每一个命题是正确的⇒下一个命题正确




问题: 你怎么数学归纳法是正确的?

- 良序原理: 自然数集合的每一个非空子集都有最小元
  - 看上去平凡? “最小的正实数是多少?”

重写数学归纳法:

# 数学归纳法

- 证明$n = n_0$的时候命题成立
- 假设任意$n = k$的时候成立, 证明$n = k + 1$的时候也成立

证明: 反证法: 完成了(1), (2), 但是还有反例, 反例对应的$n$值收集到集合里面

- 正整数$\Rightarrow$一定存在最小元$m$,
- 由于$n$是任意的, 置$n := m - 1$, 由于第$m - 1$个命题$\Rightarrow$第$m$个命题
- 第$m$个命题不是反例, 矛盾!

■

**实际上是递归地写证明的 “程序”**

```
#define BASE_CASE 0

void write_proof(int n){
```

# 数学归纳法

```
    if(n == BASE_CASE){
        printf("Here is how to prove base case...\n");
        return ;
    }
    write_proof(n-1);
    printf("Assuming n-1 is right, here is why n is
right..."\n);
}
```

归纳法无法处理有关无穷的命题:

- 任意$n > 1, \frac{1}{n} > 0$ – 没问题
- $\lim_{n\to\infty} \frac{1}{n} > 0$ – 错!

# 正确和错误的归纳法

## (a) 考试 “悖论”

**问题**：某教授对学生们说，下周我讲对你们进行一场出其不意的考试，它将安排在下周一到周五的某一天，但你们不可能预先推知究竟在哪一天。显然，这样的考试是可以实施的。但有学生通过逻辑论证说，该考试不可能安排在周五。因为如果它被安排在周五，则周一到周四都未考试，就可以推算出在周五，该考试就不再出其不意。同样该考试不可能安排在周四，因为如果它被安排在周四，那么周一至周三都没有考试，学生就可以预知考试安排在周四或周五，根据前面的论证，考试不可能在周五，因此只能够在周四，因此考试不再是出其不意。类似的可以论证周一到周三也不可能安排考试。学生们由此得出结论：这样的考试不可能存在。但该教授确实在该周内随便某一天宣布：现在开始考试。这确实出乎学生意料之外。由此得出一个悖论，这样的考试既可能实施，又不可能实施。

- 这个论断对吗?
-

# 正确和错误的归纳法

## (a) 考试 “悖论”

**问题**：某教授对学生们说，下周我讲对你们进行一场出其不意的考试，它将安排在下周一到周五的某一天，但你们不可能预先推知究竟在哪一天。显然，这样的考试是可以实施的。但有学生通过逻辑论证说，该考试不可能安排在周五。因为如果它被安排在周五，则周一到周四都未考试，就可以推算出在周五，该考试就不再出其不意。同样该考试不可能安排在周四，因为如果它被安排在周四，那么周一至周三都没有考试，学生就可以预知考试安排在周四或周五，根据前面的论证，考试不可能在周五，因此只能够在周四，因此考试不再是出其不意。类似的可以论证周一到周三也不可能安排考试。学生们由此得出结论：这样的考试不可能存在。但该教授确实在该周内随便某一天宣布：现在开始考试。这确实出乎学生意料之外。由此得出一个悖论，这样的考试既可能实施，又不可能实施。

- 这个论断对吗?
- Not even wrong! → 什么叫做 "出其不意" 的考试?

# 正确和错误的归纳法

## (b) 所有的马都是白马

问题：下面用数学归纳法证明“任意$n$匹马只有1种颜色。”

- $n = 1$时，命题成立;
- 设$n = k$时命题成立，即对任意的$k$匹马，只有 1 种颜色。现在从$k$匹马中任意取出 1 匹马（记为$x$），再放进去 1 匹马（记为$y$），根据假设，此时的$k$匹马只有一种颜色。最后，我们将$x$放回里面，这$k + 1$匹马仍然只有1种颜色。于是，$n = k + 1$时命题成立。

这个论断是否正确?

- 考虑$k = 1 \Rightarrow k = 2$的时候, 完全可以取出一个颜色的马, 放进另一个颜色的马

## (c) 蓝眼睛人的论断

问题：在一个岛上住着一个部落。这个部落有 1000 人，眼睛颜色各不相同。然而，他们的宗教禁止他们知道自己的眼睛颜色，甚至禁止讨论这个话题；因此，每个居民可以（且确实会）看到所有其他居民的眼睛颜色，但无法通过各种手段发现自己的眼睛颜色（例如没有任何反光的表面）。如果一个部落成员发现了自己的眼睛颜色，那么他们的宗教要求他们在次日中午在村庄码头离开岛屿，供所有人见证。所有部落成员都是高度逻辑性和虔诚的，并且他们都知道彼此也是高度逻辑性和虔诚的（而且他们都知道他们彼此都是高度逻辑性和虔诚的，如此往复）。

# 正确和错误的归纳法

在这 1000 名岛民中，有 100 人是蓝眼睛，900 人是棕眼睛，尽管岛民们起初并不知道这些统计数据（每个人当然只能看到 1000 个部落成员中的 999 个）。

有一天，一个蓝眼睛的外国人访问了这个岛，并赢得了部落的完全信任。一天晚上，他在致谢全体部落成员时，提到了眼睛颜色，犯了一个错误。他说：“在这个世界的这个地区看到另一个蓝眼睛的人是多么不寻常。”

这个失礼行为对部落有什么影响呢？

想法 1: 外国人的言论并没有产生任何影响，因为他的评论并没有告诉部落任何他们不知道的事情（部落里的每个人都已经看到部落里有多个蓝眼睛的人）。

# 正确和错误的归纳法

想法 2: 100 天后, 所有蓝色眼睛的人都会集体离岛.

证明:

- 基础步骤: $n = 1$. 唯一的蓝眼人: 你直接念我身份证吧.
- 归纳假设: 有 $n$ 个蓝眼人时, 前 $n - 1$ 天无人离岛, 第 $n$ 天集体离岛。
- 归纳步骤: 考虑恰有 $n + 1$ 个蓝眼人的情况。
  - 每个蓝眼人都如此推理: 我看到了 $n$ 个蓝眼人, 他们应该在第 $n$ 天集体离岛。
  - 但是, 每个蓝眼人都在等其它 $n$ 个蓝眼人离岛, 因此, 第 $n$ 天无人离岛。
  - 每个蓝眼人继续推理: 一定不止 $n$ 个蓝眼人, 但是我看到的其余人都不是蓝眼。
  - 所以, “小丑竟是我自己”。

# 正确和错误的归纳法

■

很奇妙: $n = 1, n = 2$的时候的情形是

- 我不知道...
- 我知道...
- 我知道你知道...
- 我知道你知道我知道...

# 对于某一结构的归纳

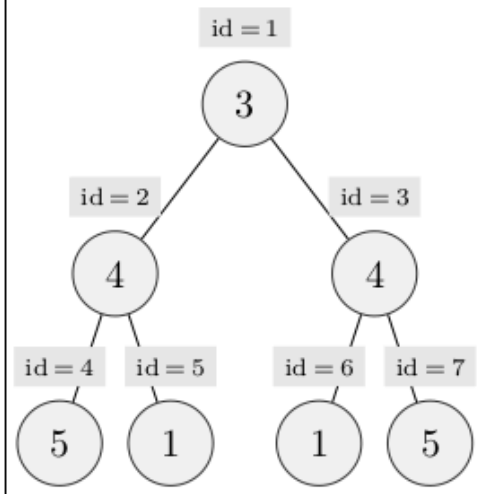
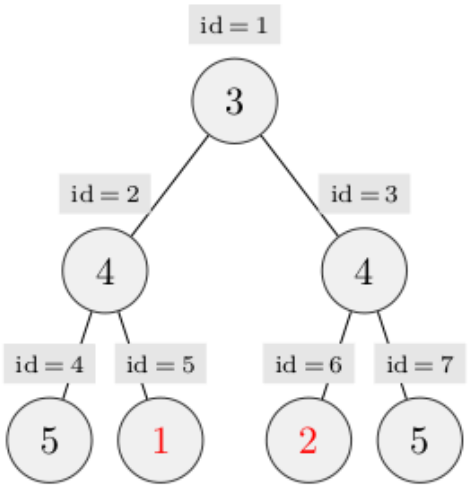
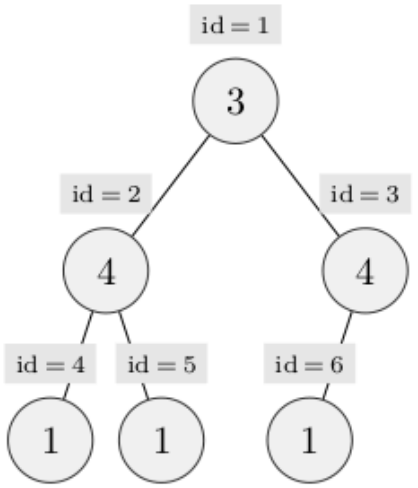
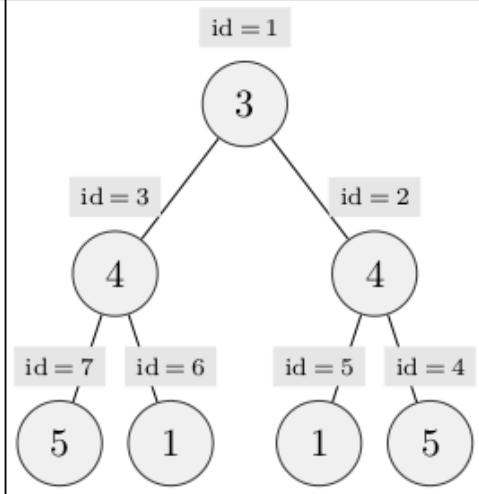
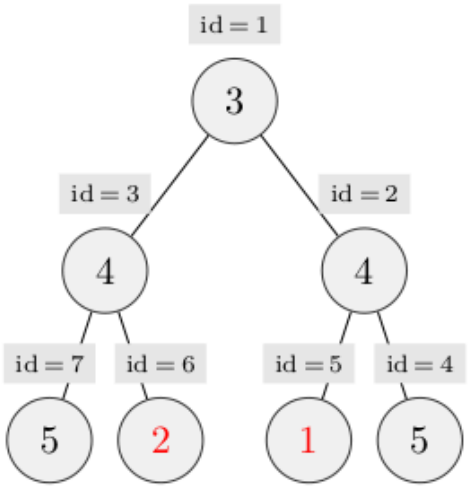
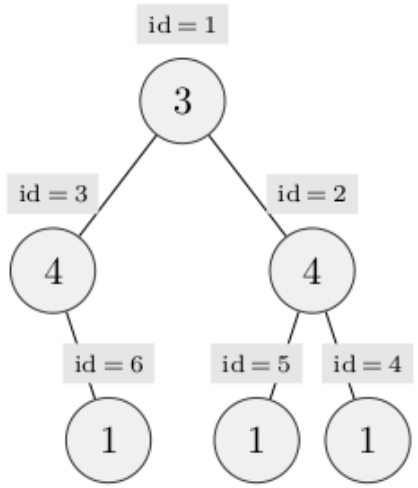
自动写证明的程序: 多传进去点参数

```
void write_proof(State n){
  if(State is base case){…; return;}
  Assume we can reduce n to n’;
  write_proof(State n’);
  write proof from n’ to n;
}
```

例子: 刚刚的二叉树

# P5018 对称二叉树



| | 对称二叉树 | 非对称二叉树<br>（权值不对称） | 非对称二叉树<br>（结构不对称） |
|---|---|---|---|
| 原树 |  |  |  |
| 所有结点的左右子树交换后 |  |  |  |

洛谷

# P5018 对称二叉树



- 如何判定两棵树是不是对称的?
  - 基本情况: 两个节点为空/一个为空一个不为空/不为空但是节点值(相同/不同)
  - 归纳情况: 根据二叉树的定义
- 如何数树上的节点?
  - 基本情况: 空节点的大小为 0.
  - 归纳情况: 大小=左边子树大小+右边子树大小+1.

# 更多的应用: 暴力枚举

# 问题 1: 枚举$1\sim n$的排列

- 想法: 一个固定的数字$m$ $\rightarrow$写$m$重循环
- 不固定的情形: 考虑每一步: 序列是什么? 每一个数用过了吗?

```
void GetPermutation(string& str, int index, vector<bool>&
visit, vector<char>& sequence) {
  if (index == str.size()) {打印, 返回;}
  for (int i = 0; i < str.size(); ++i) {
      if (visit[i]) { // 被访问过就跳过
          continue;
      } else {
          visit[i] = true;
          sequence[index] = str[i];
          // 接着查找下一位
          GetPermutation(str, index+1, visit, sequence);
          visit[i] = false; // 回溯
      }
    }
}
```

# 问题 1: 枚举$1\sim n$的排列

使用 C 版本:

```c
//...上面省略
// 尝试在 A[cur] 中填各个整数 i
    for (int i = 1; i <= n; i++) {
        int ok = 1;
        // 检查 i 是否已经在 A[0] 到 A[cur-1] 中出现过
        for (int j = 0; j < cur; j++) {
            if (A[j] == i) {
                ok = 0; break;
            }
        }
        // 如果 i 没有出现过，选择 i 并递归填下一位
        if (ok) {
            A[cur] = i;
            print_permutation(n, A, cur + 1);
        }
    }
```

# 问题 2: 多重集的全排列

输入数组$P$, $A$是结果数组. 有可能出现多个?

想法 1:

- 统计$A[0]$~$A[\text{cur}-1]$中$P[i]$出现的次数$c_1$, 统计$P$中$P[i]$出现的次数$c_2$. 只要$c_1 < c_2$, 就可以递归调用.

```
for (int i = 0; i < n; i++) {
        int c1 = 0, c2 = 0;
        for (int j = 0; j < cur; j++) {
            if (A[j] == P[i]) c1++;
        for (int j = 0; j < n; j++)
            if (P[i] == P[j]) c2++;
        if (c1 < c2) {
            A[cur] = P[i];
            print_permutation(n, P, A, cur + 1);
        }
    }
}
```

# 问题 2: 多重集的全排列

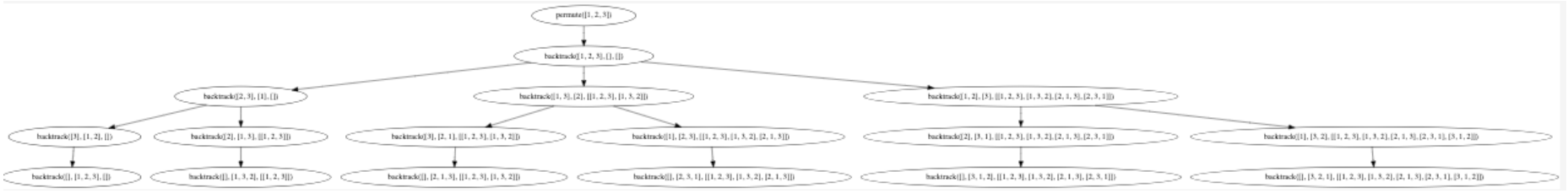
问题: 输出了 27 个 111

- 原因: 重复了

```
for (int i = 0; i < n; i++) {
        if (i == 0 || P[i] != P[i - 1]) { // 防止重复递归
          for (int j = 0; j < cur; j++) {
            ... // 中间的不变
            print_permutation(n, P, A, cur + 1);
          }
        }
```

# 生成全排列的函数调用关系






任何递归-回溯的框架都可以化为这样的图.