习题课-第9周

目录

- 第7、8周-Quiz
- 第四、五、六次上机题目
- 第三次书面作业
- 补充

Quiz

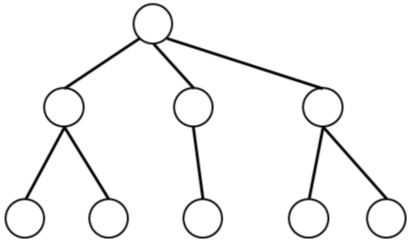
- 若某树有 n_1 个度数为 1 的结点，有 n_2 个度数为 2 的结点， ……，有 n_m 个度数为m的结点。树的度数为m。试问它有多少个叶结点，给出计算的过程。
  - 用两种方式来计算边的总数 E
  - 边即父子关系，分别从“父”和“子”的角度来考虑
  - (1) 除根节点外，每个结点都有一个父节点

$$E = n_0 + n_1 + n_2 + \ldots\ldots + n_m - 1$$

  - (2) 每个结点具有的子结点数量

$$E = n_1 + 2*n_2 + \ldots\ldots m*n_m$$

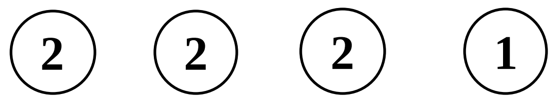
  - 解出： $n_0 = n_2 + 2*n_3 + \ldots\ldots + (m-1)*n_m + 1$



Quiz

- 利用Huffman算法，用’0’, ‘1’两个字符来编码字符串“abcdabc”。写出该字符串最短编码的总长度。
  - 统计各个待编码字符的频率（权重）：

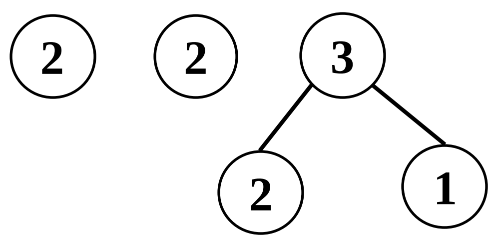
    {‘a’: 2, ‘b’: 2, ‘c’: 2, ‘d’: 1}
  - 构建Huffman树：不断取权重最小的结点合并，组成新节点



二叉Huffman树

- 利用Huffman算法，用’0’, ‘1’两个字符来编码字符串“abcdabc”。写出该字符串最短编码的总长度。
  - 统计各个待编码字符的频率（权重）：

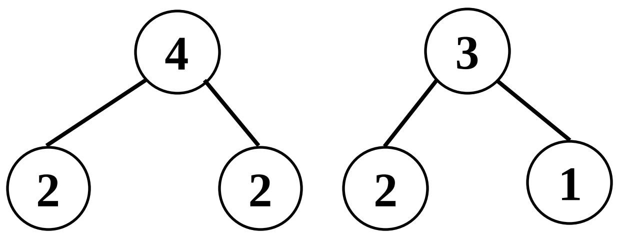
    {‘a’: 2, ‘b’: 2, ‘c’: 2, ‘d’: 1}
  - 构建Huffman树：不断取权重最小的结点合并，组成新节点



二叉Huffman树

- 利用Huffman算法，用’0’, ‘1’两个字符来编码字符串“abcdabc”。写出该字符串最短编码的总长度。
  - 统计各个待编码字符的频率（权重）：

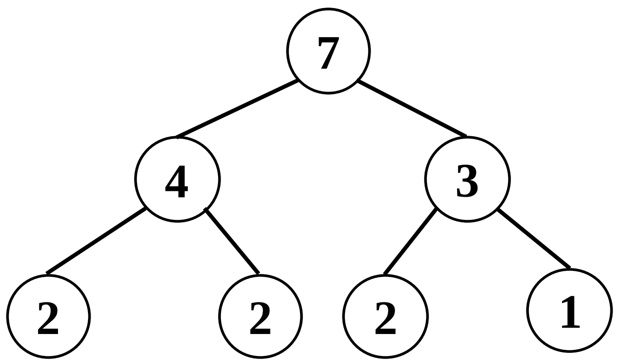
    {‘a’: 2, ‘b’: 2, ‘c’: 2, ‘d’: 1}
  - 构建Huffman树：不断取权重最小的结点合并，组成新节点



二叉Huffman树

- 利用Huffman算法，用’0’, ‘1’两个字符来编码字符串“abcdabc”。写出该字符串最短编码的总长度。
  - 统计各个待编码字符的频率（权重）：

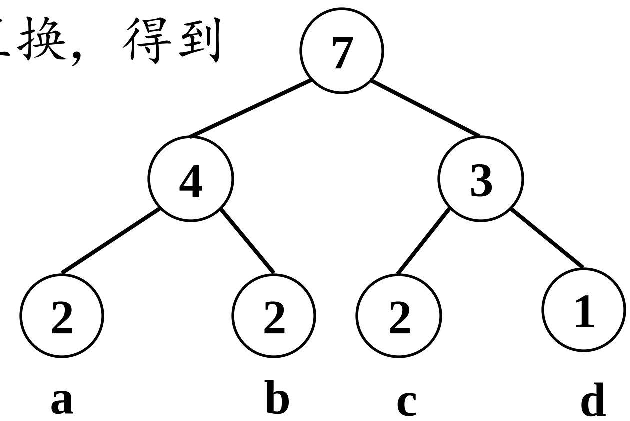
    {‘a’: 2, ‘b’: 2, ‘c’: 2, ‘d’: 1}
  - 构建Huffman树：不断取权重最小的结点合并，组成新节点



二叉Huffman树

- 利用Huffman算法，用’0’, ‘1’两个字符来编码字符串“abcdabc”。写出该字符串最短编码的总长度。
  - 统计各个待编码字符的频率（权重）：

    {‘a’: 2, ‘b’: 2, ‘c’: 2, ‘d’: 1}
  - 构建Huffman树：不断取权重最小的结点合并，组成新节点
  - 编码：{‘a’: 00, ‘b’: 01, ‘c’: 10, ‘d’: 11}
  - 编码不唯一。例如可以将编码的第一位0，1互换，得到
    - {‘a’: 10, ‘b’: 11, ‘c’: 00, ‘d’: 01}
  - 原字符串编码结果的长度：14

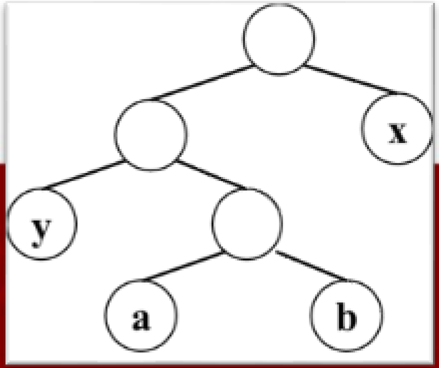


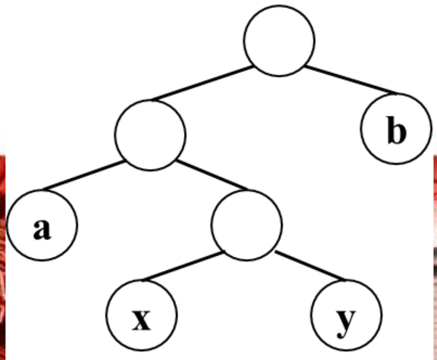
Huffman算法的正确性

- Huffman算法是一种贪心算法，求解最优二叉树编码问题
  - 满足最优子结构性质、贪心选择性质
- 最优二叉编码树的问题，可以表述如下
  - 给定n个权值非负的结点
  - 每次选择两个结点，合并权值并生成一个父节点
  - 直到仅剩下一个结点，此时二叉树构建完成
  - 要求找到一种最优的选择序列，使得构建的二叉树的带权外部路径长度之和（WPL）最小
- 最优子结构性质：原问题的最优解中包含子问题的最优解
  - 显然成立。最优解的选择序列中，首先合并了两个结点之后，剩余的结点构成子问题。后续的选择序列也一定是该子问题的最优解

Huffman算法的正确性

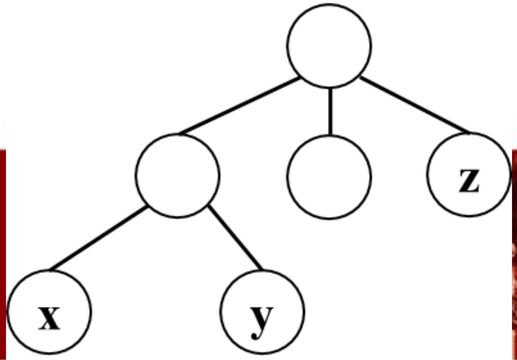
- 贪心选择性质：假设x, y是待编码字符中频率最小的两个字符（权重最小），那么一定存在一个最优编码二叉树，使得x, y的码字长度相同，且仅有最后一位不同
  - 也就是，首先合并x和y，能够构成一个全局最优解
- 假设T是任意一个最优编码二叉树
  - 假设a和b是树中深度最大的兄弟叶节点（它们一定存在）
  - x, y可能出现于T中的任意位置上
  - 可以证明，将x, y替换到a, b的位置上，不会增加WPL，这样就构造出了符合要求的编码树
    - 参考：https://zhuanlan.zhihu.com/p/574351593





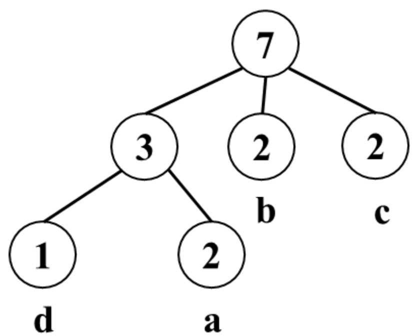
K叉Huffman树

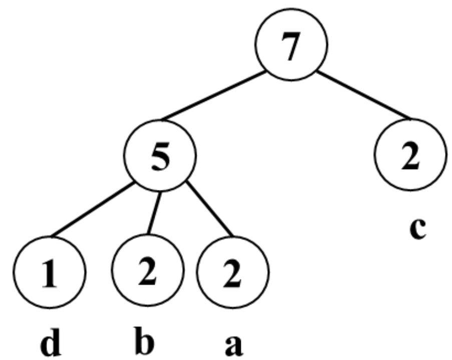
- 如何将这一方法应用于K叉Huffman树的构建？（假定K=3）
  - 例如，如果可以用0, 1, 2三种字符编码，就应该构建最优三叉编码树
- 最直观：每次取K个权重最小的结点合并
- 问题的最优子结构没有改变
- 如果想要以同样的方式论证贪心选择性质
  - 假设权值最小的3个结点是x, y, z
  - 想要把它们移动到深度最深的3个兄弟节点上去，从而构造最优解
  - 发现不总是可行，最深的兄弟结点数量可能<3，因为合并时并非必须总是要合并3个结点。
    - 但是合并时至少要合并2个结点，所以这种做法在K=2时可行



K叉Huffman树

- Quiz题目本身就提供了这种做法的一个反例
  - 最优的三叉编码树（左）：WPL=10
  - 每次选取K个最小权重结点（右）：WPL=12

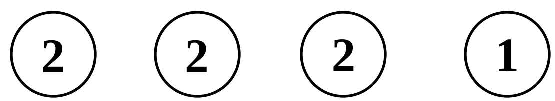




  - 关键：K叉哈夫曼编码树可能不是满K叉树，但“空隙”必须出现在最深一层
  - 严格来说：假如最长的码字长为L，则任何长度小于L的编码序列，或者编码了字符，或者其前缀编码了字符，不允许空闲。

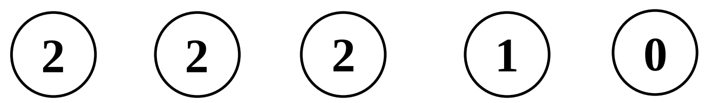
K叉Huffman树

- K叉Huffman树的构建算法只需要做一处调整即可
  - 在合并结点前，先补充若干个权值为0的结点
  - 保证每次都选取K个结点合并，最后恰好剩下一个结点
  - 这样，就保证最优编码树一定是满K叉树，可以沿用Huffman算法
  - 补充的“哑结点”的数量m应该满足：(N+m) % (K-1) = 1
- 原问题：N=4, K=3，补充m=1个哑结点



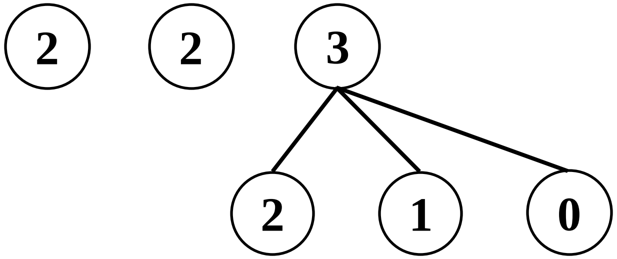
K叉Huffman树

- K叉Huffman树的构建算法只需要做一处调整即可
  - 在合并结点前，先补充若干个权值为0的结点
  - 保证每次都选取K个结点合并，最后恰好剩下一个结点
  - 这样，就保证最优编码树一定是满K叉树，可以沿用Huffman算法
  - 补充的“哑结点”的数量m应该满足：(N+m) % (K-1) = 1
- 原问题：N=4, K=3，补充m=1个哑结点



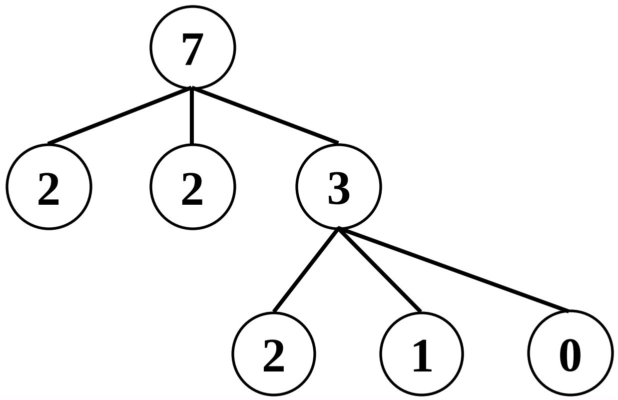
K叉Huffman树

- K叉Huffman树的构建算法只需要做一处调整即可
  - 在合并结点前，先补充若干个权值为0的结点
  - 保证每次都选取K个结点合并，最后恰好剩下一个结点
  - 这样，就保证最优编码树一定是满K叉树，可以沿用Huffman算法
  - 补充的“哑结点”的数量m应该满足：(N+m) % (K-1) = 1
- 原问题：N=4, K=3，补充m=1个哑结点



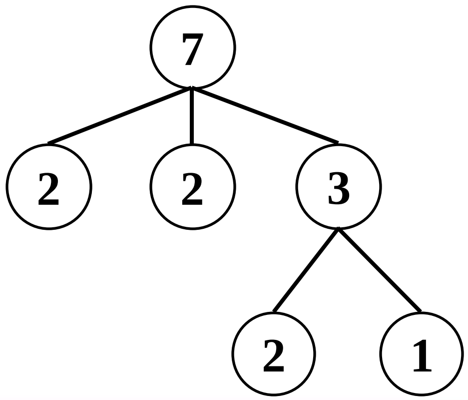
K叉Huffman树

- K叉Huffman树的构建算法只需要做一处调整即可
  - 在合并结点前，先补充若干个权值为0的结点
  - 保证每次都选取K个结点合并，最后恰好剩下一个结点
  - 这样，就保证最优编码树一定是满K叉树，可以沿用Huffman算法
  - 补充的“哑结点”的数量m应该满足：(N+m) % (K-1) = 1
- 原问题：N=4, K=3，补充m=1个哑结点



K叉Huffman树

- K叉Huffman树的构建算法只需要做一处调整即可
  - 在合并结点前，先补充若干个权值为0的结点
  - 保证每次都选取K个结点合并，最后恰好剩下一个结点
  - 这样，就保证最优编码树一定是满K叉树，可以沿用Huffman算法
  - 补充的“哑结点”的数量m应该满足：(N+m) % (K-1) = 1
- 最终结果：



Huffman算法的时间复杂度

- Huffman算法的主要代价为，不断地从剩余结点中取出权值最小的结点
- 这一操作共进行了O(N)次
- 适合使用堆结构来实现取最小的操作。
  - 每次弹出堆顶元素的代价为O(log N)
  - 建堆的代价为O(log N)
- 总体的时间复杂度为 O(N*logN)
  - 如果不使用堆优化，每次遍历取最小权值的结点，复杂度为$O(N^2)$
  - 如果每次排序所有结点，复杂度将是$O(N^2*\log N)$

Quiz

- 考虑关键字集合K={19，1，23，34，20，84，27，53，11，56，29}，设计一个散列表：
  - 固定表长为13，地址空间为0~12
  - 采用哈希函数：h(key)=key%13，
- 1.使用线性探查法。写出依次插入上述关键码后的散列表结果

散列地址	0	1	2	3	4	5	6	7	8	9	10	11	12
关键码		1	27	53	56	29	19	20	34	84	23	11	

- 2.使用双散列函数法，h_2(key)=key%11+1。写出依次插入上述关键码后的散列表结果

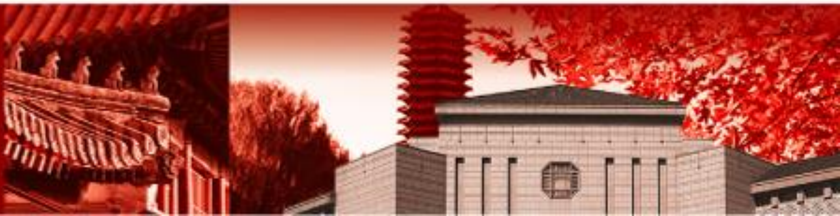
散列地址	0	1	2	3	4	5	6	7	8	9	10	11	12
关键码	27	1		29	56		19	20	34	84	23	53	11

上机题目：升空的焰火，从侧面看

- 题目：输出二叉树的层序遍历中，每一层的最后一个元素组成的序列。
- 只需要对层序遍历程序略微修改即可
  - 原程序：首先入队根结点、每次取出队首元素、访问该元素、并依次入队其全部子结点
  - 修改：首先入队根节点、每次取出队列中的全部元素、依次访问、并依次入队全部子节点
  - 保证队列中的元素始终是同一层的
  - 额外使用一个队列作为缓冲
  - 每次出队时，将最后一个元素添加至结果

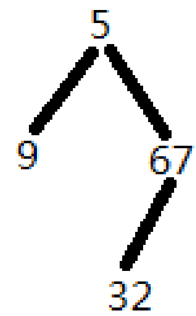
```
   1            <---
 /   \
2     3         <---
 \     \
  5     4       <---
```

北京大學



上机题目：由中序序列和后序序列重建二叉树

- 中序序列：9, 5, 32, 67
- 后序序列：9, 32, 67, 5



- 1. 首先确认序列中的根结点：后序序列的尾元素
- 2. 确认左子树和右子树的范围。
  - 在中序序列中，根的左右两侧即为左子树、右子树的中序序列
  - 后序序列中，左子树与右子树的后序序列都在根之前，无法直接区分
  - 但是利用中序序列，可以知道左子树、右子树分别包含的结点数量
- 3. 至此，分别知道了左右子树的中序、后序序列，递归求解即可。

上机题目：Pre-Post-erous!

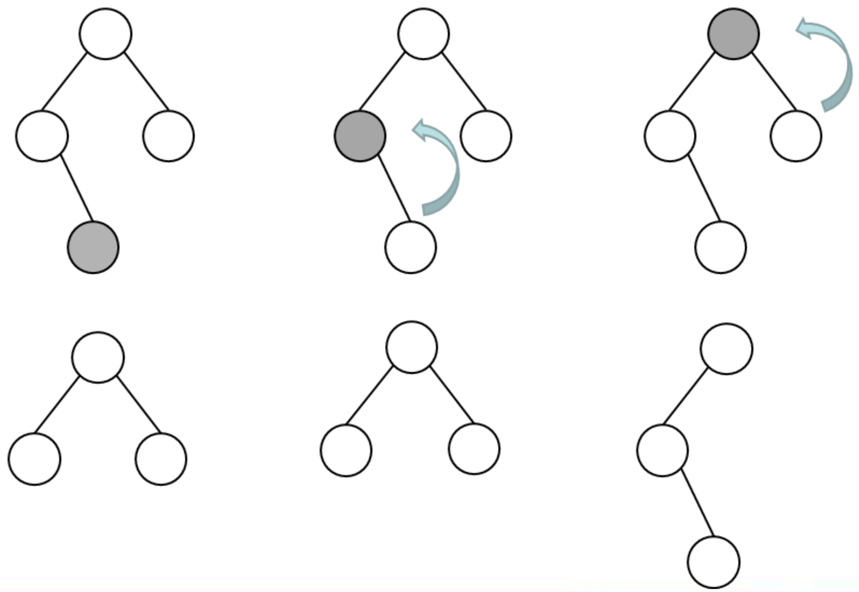
- 题目：利用二叉树的前序序列与后序序列不足以重建二叉树（K叉树）。给定前序、后序序列，求可能的K叉树数量。
- 例：K=13, 前序：abejkcfghid、后序：jkebfghicda
- 理解的关键：为什么不足以重建二叉树？
  - 仍然可以确定根节点：a
  - 事实上，还可以确定根的每一棵子树
    - 根据前序序列，第一棵子树的根为b
    - 在后序序列中找到b，就知道第一棵子树包含的结点数量为4： bejk, jkeb
    - 依次类推，可以递归确定所有结点的所有子树
  - 问题在于：K叉树不同于有序树。
  - 若根的度数 $m < K$，即使确定了所有的子树，也无法确定K叉树，但可以确定唯一的有序树。

上机题目：Pre-Post-erous!

- 对于任何一个结点，其度数 $m < K$，则其m棵子树具有 $C(m, K)$ 种排列方式，形成不同的K叉树
- 各结点对树形数量的影响相互独立
- 因此，只需按照上述算法，求出所有结点的度数，计算所有的 $C(m, K)$ 并求乘积即为答案

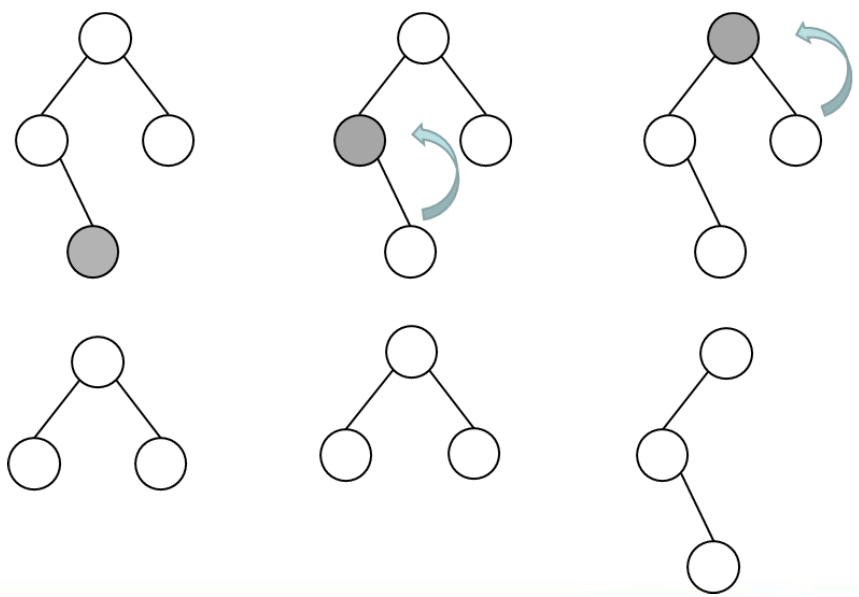
上机题目：二叉搜索树的结点删除

- 题目：实现二叉搜索树的删除
- 回顾：BST删除结点的三种情形
  - 待删除结点度数为0：直接删除



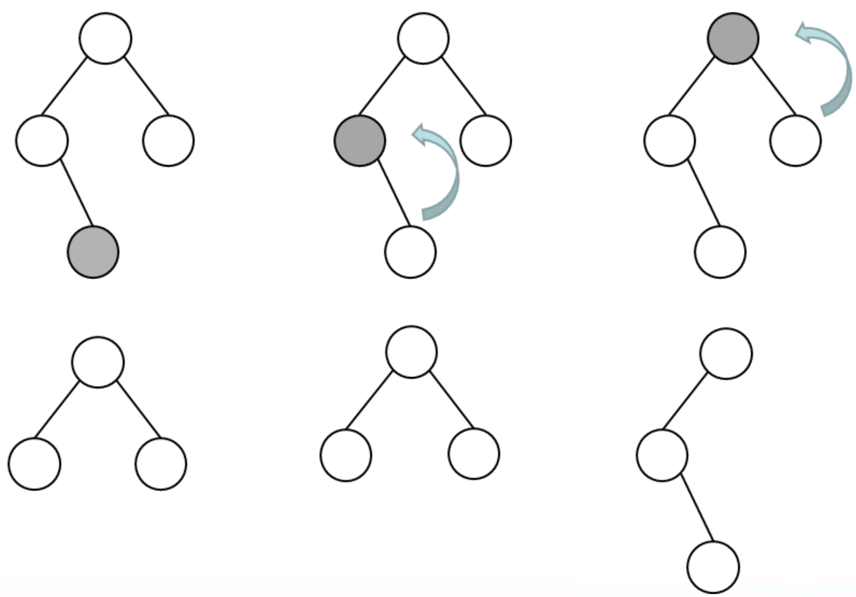
上机题目：二叉搜索树的结点删除

- 题目：实现二叉搜索树的删除
- 回顾：BST删除结点的三种情形
  - 待删除结点度数为1：左子或右子顶替待删除结点



上机题目：二叉搜索树的结点删除

- 题目：实现二叉搜索树的删除
- 回顾：BST删除结点的三种情形
  - 待删除结点度数为2：先删除后继元素（或前驱元素，度数至多为1），然后用后继元素的值顶替待删除元素



上机题目：二叉搜索树的结点删除

- 实现中的注意事项：
  - 删除一个结点：调整所有指向该结点的指针
  - 包括子结点的parent指针，父结点的child指针
  - 特别考虑根节点：没有父结点，并且作为整棵树的“入口”
- 调错(Debug)经验：
  - Runtime Error：空指针引用、递归栈溢出、输入异常、除0错误
  - 发现在一组数据上程序行为不及预期怎么办：这是好消息，使用print调试（输出运行时的关键信息），或使用断点与逐步调试
  - 发现在所有样例上都正确，但是WA怎么办：样例只有2-3个例子，而在有的问题上（比如这道题）我们自己可以轻易构造出很多测试数据。花一些时间在测试上，尝试不同的二叉树形状、以不同的顺序删除、尝试“极端”情况。如果发现输出结果错误，就可以高效改进。

上机题目：约瑟夫问题

1:约瑟夫问题

总时间限制: 1000ms　　内存限制: 65536kB

描述

约瑟夫问题：有 n 只猴子，按顺时针方向围成一圈选大王（编号从 1 到 n），从第 1 号开始报数，一直数到 m，数到m的猴子退出圈外，剩下的猴子再接着从1开始报数。就这样，直到圈内只剩下一只猴子时，这个猴子就是猴王，编程求输入 n，m后，输出最后猴王的编号。

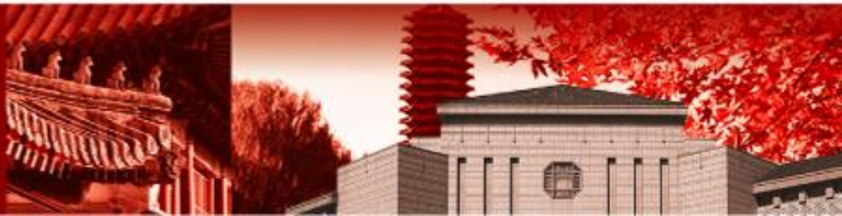
输入

每行是用空格分开的两个整数，第一个是 n, 第二个是 m (0 < m,n <=300)。最后一行是:

0 0

输出

对于每行输入数据（最后一行除外)，输出数据也是一行，即最后猴王的编号



上机题目：约瑟夫问题

- 用列表模拟队列
- 用index记录报数
- 每次从队列头部pop，增加index，判断是否到m
- 如果没到则重新入队列
- 如果到了则index清零
- 一直到队列中只有一个元素

上机题目：约瑟夫问题

```python
while True:
    m , n = map(int, input().split())
    if m == 0 and n == 0:
        break
    monkey_list = []
    for i in range(m):
        monkey_list.append(i)
    index = 0
    while len(monkey_list) > 1:
        index = index + 1
        monkey = monkey_list.pop(0)
        if index == n:
            index = 0
            continue
        else:
            monkey_list.append(monkey)
    print(monkey_list[0] + 1)
```

上机题目：放苹果

2:放苹果

总时间限制: 1000ms　内存限制: 65536kB

描述

把M个同样的苹果放在N个同样的盘子里，允许有的盘子空着不放，问共有多少种不同的分法？（用K表示）5，1，1和1，5，1 是同一种分法。

输入

第一行是测试数据的数目t（0 <= t <= 20）。以下每行均包含二个整数M和N，以空格分开。1<=M，N<=10。

输出

对输入的每组数据M和N，用一行输出相应的K。

样例输入

```
1
7 3
```

样例输出

```
8
```

上机题目：放苹果

- 重点考察思路，代码比较简单
- F(M,N)表示把M个苹果放到N个盘子里的方法数
- 分情况讨论：存在空盘子或者不存在空盘子
- 因此可以把问题拆成两个部分
- F(M,N)=F(M,N-1)+F(M-N,N)
- 前者表示存在空盘子，可以把原问题变成把M个苹果放到N-1个盘子里
- 后者表示不存在空盘子，相当于每个盘子里面都至少有一个苹果，可以把原问题变成把M-N个苹果放到N个盘子里
- 注意结束时的边界条件，如果N大于M，F(M,N)=F(M,M)

上机题目：放苹果

```python
def count_ways(m, n):
    if m == 0 or n == 1:
        return 1
    if n > m:
        return count_ways(m, m)
    return count_ways(m, n - 1) + count_ways(m - n, n)

t = int(input())
for _ in range(t):
    m, n = map(int, input().split())
    result = count_ways(m, n)
    print(result)
```

上机题目：切割回文

3:切割回文

总时间限制: 1000ms　内存限制: 65536kB

描述

阿福最近对回文串产生了非常浓厚的兴趣。

如果一个字符串从左往右看和从右往左看完全相同的话，那么就认为这个串是一个回文串。例如，“abcaacba”是一个回文串，“abcaaba”则不是一个回文串。

阿福现在强迫症发作，看到什么字符串都想要把它变成回文的。阿福可以通过切割字符串，使得切割完之后得到的子串都是回文的。

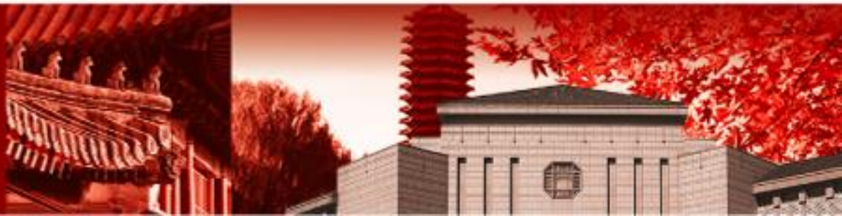
现在阿福想知道他最少切割多少次就可以达到目的。例如，对于字符串“abaacca”，最少切割一次，就可以得到“aba”和“acca”这两个回文子串。

输入

输入的第一行是一个整数 T (T <= 20) ，表示一共有 T 组数据。
接下来的 T 行，每一行都包含了一个长度不超过的 1000 的字符串，且字符串只包含了小写字母。

输出

对于每组数据，输出一行。该行包含一个整数，表示阿福最少切割的次数，使得切割完得到的子串都是回文的。



上机题目：切割回文

- 动态规划问题，将原问题划分成规模更小的子问题
- 我们对于该字符串s的每一位，例如第i位，我们用dp[i]来保存s[0:i+1]最少切割的次数
- 当我们计算第j位时的dp[j]，我们已经有了前面的所有dp数组，因此我们只需要对于之前的每一位k，判断s[k+1:j+1]是不是回文数，然后使dp[j]=min(dp[j],dp[k]+1)
- 这个dp数组的计算便是解决动态规划问题的关键

上机题目：切割回文

```
def is_palindrome(s):
    return s == s[::-1]


def min_cuts(s):
    n = len(s)
    dp = [float('inf')] * n
    for i in range(n):
        if is_palindrome(s[:i + 1]):
            dp[i] = 0
        else:
            for j in range(i):
                if is_palindrome(s[j + 1:i + 1]):
                    dp[i] = min(dp[i], dp[j] + 1)
    return dp[-1]


t = int(input())
for _ in range(t):
    s = input()
    result = min_cuts(s)
    print(result)
```

上机题目：宗教信仰

4:宗教信仰

总时间限制: 5000ms　　内存限制: 65536kB

描述

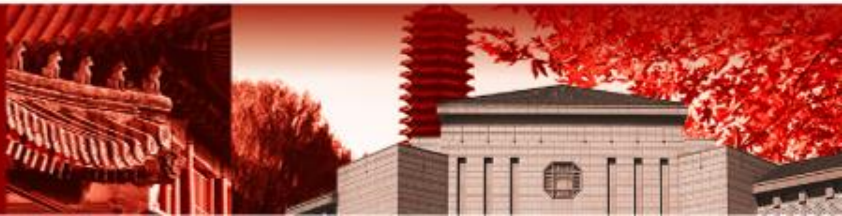
世界上有许多宗教，你感兴趣的是你学校里的同学信仰多少种宗教。
你的学校有n名学生（0 < n <= 50000），你不太可能询问每个人的宗教信仰，因为他们不太愿意透露。但是当你同时找到2名学生，他们却愿意告诉你他们是否信仰同一宗教，你可以通过很多这样的询问估算学校里的宗教数目的上限。你可以认为每名学生只会信仰最多一种宗教。

输入

输入包括多组数据。
每组数据的第一行包括n和m，0 <= m <= n(n-1)/2，其后m行每行包括两个数字i和j，表示学生i和学生j信仰同一宗教，学生被标号为1至n。输入以一行 n = m = 0 作为结束。

输出

对于每组数据，先输出它的编号（从1开始），接着输出学生信仰的不同宗教的数目上限。



上机题目：宗教信仰

- 并查集的问题
- 并查集的关键在于find和union
- 初始化父节点数组，每个人的父节点都指向自己
- 当两个学生信仰同一种宗教时，用union将他们进行联系
- 最后看那些父节点仍是自己的便是一种不同的宗教

上机题目：宗教信仰

```python
def find(parent, x):
    if parent[x] != x:
        parent[x] = find(parent, parent[x])
    return parent[x]


def union(parent, a, b):
    a_root = find(parent, a)
    b_root = find(parent, b)
    if a_root < b_root:
        parent[b_root] = a_root
    else:
        parent[a_root] = b_root


case_num = 1
while True:
    n, m = map(int, input().split())
    if n == 0 and m == 0:
        break
    parent = list(range(n + 1))
    for _ in range(m):
        i, j = map(int, input().split())
        union(parent, i, j)
    religion_count = 0
    for i in range(1, n + 1):
        if parent[i] == i:
            religion_count += 1
    print(f"Case {case_num}: {religion_count}")
    case_num += 1
```

上机题目：食物链

5:食物链

总时间限制: 1000ms　 内存限制: 65536kB

描述

动物王国中有三类动物A,B,C，这三类动物的食物链构成了有趣的环形。A吃B， B吃C，C吃A。
现有N个动物，以1－N编号。每个动物都是A,B,C中的一种，但是我们并不知道它到底是哪一种。
有人用两种说法对这N个动物所构成的食物链关系进行描述：
第一种说法是"1 X Y"，表示X和Y是同类。
第二种说法是"2 X Y"，表示X吃Y。
此人对N个动物，用上述两种说法，一句接一句地说出K句话，这K句话有的是真的，有的是假的。当一句话满足下列三条之一时，这句话就是假话，否则就是真话。
1） 当前的话与前面的某些真的话冲突，就是假话；
2） 当前的话中X或Y比N大，就是假话；
3） 当前的话表示X吃X，就是假话。
你的任务是根据给定的N（1 <= N <= 50,000）和K句话（0 <= K <= 100,000），输出假话的总数。

输入

第一行是两个整数N和K，以一个空格分隔。
以下K行每行是三个正整数 D，X，Y，两数之间用一个空格隔开，其中D表示说法的种类。
若D=1，则表示X和Y是同类。
若D=2，则表示X吃Y。

输出

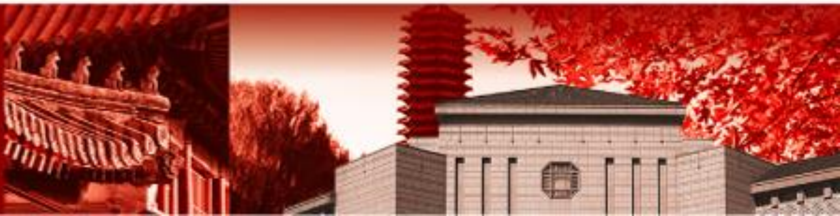
只有一个整数，表示假话的数目。

上机题目：食物链

- 这个题很难
- 并查集对于“同类”的关系很好处理，因为“同类”关系总是满足传递性，即A的同类的同类也是A的同类。
- 但是对于“捕食”关系，并查集并不好维护，因为“捕食”并不具备传递性，因为A的猎物的猎物是A的天敌。
- 并查集维护有向关系，一个常见的技巧就是“扩展域”，在这个题中，我们可以维护一个并查集，由三个域组成：同类域、猎物域、天敌域。

上机题目：食物链

- 若A和B是同类，则一定有：
- A的同类与B的同类是同类；
- A的猎物与B的猎物是同类；
- A的天敌与B的天敌是同类。
- 若A捕猎B，则一定有：
- A的同类是B的天敌；
- A的猎物是B的同类；
- A的天敌是B的猎物。
- CSDN补充链接：https://blog.csdn.net/WillHou/article/details/141758917



上机题目：食物链

```python
for _ in range(k):
    d, x, y = map(int, input().split())
    x -= 1
    y -= 1
    if x < 0 or x >= n or y < 0 or y >= n:
        false_count += 1
        continue
    if d == 1:
        if find(x) == find(y + n) or find(x) == find(y + 2 * n) or find(x + n) == find(y) or find(x + 2 * n) == find(y):
            false_count += 1
        else:
            union(x, y)
            union(x + n, y + n)
            union(x + 2 * n, y + 2 * n)
    else:
        if x == y or find(x) == find(y) or find(x) == find(y + 2 * n):
            false_count += 1
        else:
            union(x, y + n)
            union(x + n, y + 2 * n)
            union(x + 2 * n, y)
```

第三次作业

- 一棵完全二叉树上有1001个节点，其中叶子结点的个数是501。
- 一个具有1025个节点的二叉树的高为10-1024。
  - 因为这道题没有说高怎么计算，所以11-1025也是对的。
- 已知一棵树的中序遍历序列为DBGEACF，后序遍历序列为DGEBFCA，则这棵树的前序遍历结果为ABDEGCF。
- 某段电文中只有a,b,c,d四种字符，各种字符出现的次数为：a出现1000次，b出现2000次，c出现6000次，d出现1000次，采用Huffman编码该电文的长度为16000个比特。

第三次作业

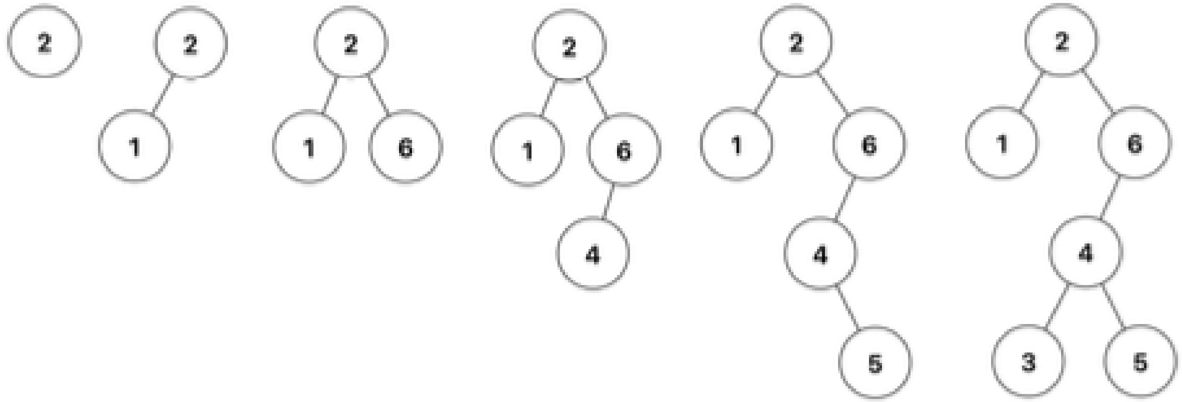
- 定义二叉树中一个节点的度数为其子节点的个数。现有一棵节点总数为101的二叉树，其中度数为1的节点数有30个，则度数为0的节点有36个。
- 定义完全二叉树的根节点所在的层为第1层，如果一个完全二叉树的第6层有23个叶节点，则它的总节点数可能为54，80，81 。
- 已知二叉树的前序遍历结果为ADC，这棵二叉树的树型有5种可能。
- 对于具有57个节点的完全二叉树，如果按层次自顶向下，同一层自左向右，顺序从0开始对全部节点进行编号，则编号为18的节点的父节点的编号是8，编号为19的右子女节点的编号是40。

第三次作业

- 对于一棵包含k个节点的满二叉树，其叶子结点的个数为 $\lfloor k/2 \rfloor+1$ or（k-1）/2+1。

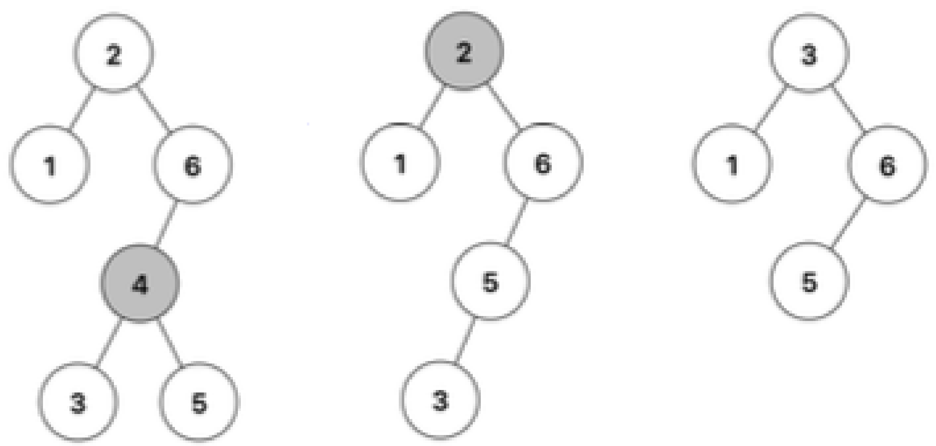
第三次作业

- 简要回答下列Binary Search Tree（BST）及BST更新过程的相关问题。
  - 请简述什么是BST。
  - BST或者是一棵空树；或者是具有下列性质的二叉树：对于任何一个结点，设其值为K，则该结点的左子树(若不空)的任意一个结点的值都小于 K；该结点的右子树(若不空)的任意一个结点的值都大于 K；而且它的左右子树也分别为BST
  - 请图示2，1，6，4，5，3按顺序插入一棵BST树的中间过程和最终形态。



第三次作业

- 简要回答下列Binary Search Tree（BST）及BST更新过程的相关问题。
  - 请图示以上BST树，依次删除节点4和2的过程和树的形态。



第三次作业

- 哈夫曼树是进行编码的一种有效方式。设给定五个字符，其相应的权值分别为 {4，8，6，9，18}，试画出相应的哈夫曼树，并计算它的带权外部路径长度WPL。

下面是构建哈夫曼树的过程：

1. 将给定的五个字符按照权值从小到大排序：
   1. {4，6，8，9，18}
2. 不断合并权值最小的两个节点，直到只剩下一个节点：
   1. 合并 4 和 6，得到节点 a，权值为 10
   2. 合并 8 和 9，得到节点 b，权值为 17
   3. 合并节点 a 和 b，得到节点c，权值为 27
   4. 合并节点c和 18，得到根节点d，权值为45

下面是相应的哈夫曼树：

```
              (45)
             /    \
          (27)     18
          /    \
   (10)         (17)
   /  \         /    \
  4    6       8      9
```

计算带权外部路径长度（WPL）：

WPL = 4 * 3 + 6 * 3 + 8 * 3 + 9 * 3 + 18 * 1
  = 12 + 18 + 24 + 27 + 18
  = 99

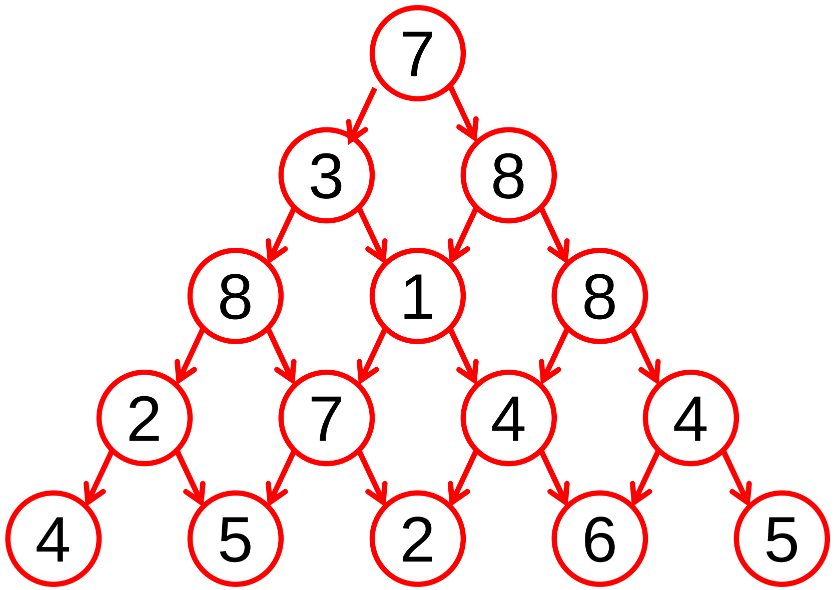
所以，哈夫曼树的带权外部路径长度为 99。

补充部分

动态规划&树

数字三角形

- 给定一个由n行非负数字组成的数字三角形，如下图所示。试设计一个算法，计算从三角形的顶至底的一条路径，使该路径经过的数字总和最大。



数字三角形——递归

- 递归？
  - $num[i][j]$：表示第i行的第j个数字
  - $max\_sum[i][j]$: 表示从位置(i, j)到底边的各条路径中，最佳路径的数字之和
  - $max\_sum[i][j] = \max\{max\_sum[i+1][j], max\_sum[i+1][j+1]\} + num[i][j]$

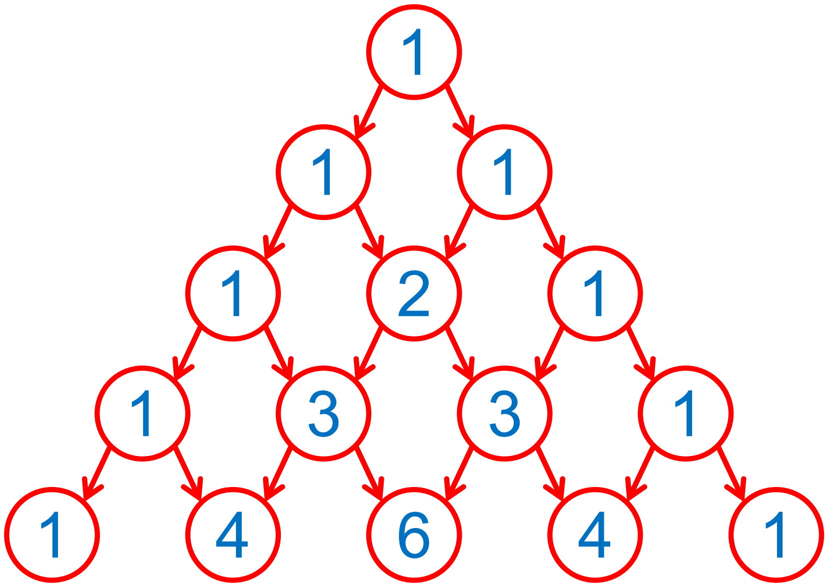
数字三角形——递归

- 递归？
  - num[i][j]：表示第i行的第j个数字
  - max_sum[i][j]: 表示从位置(i, j)到底边的各条路径中，最佳路径的数字之和
  - max_sum[i][j] = max(max_sum[i+1][j], max_sum[i+1][j+1]) + num[i][j]

```python
def max_sum_recursive(i, j, num, n):
    if i == n - 1:
        return num[i][j]
    x = max_sum_recursive(i + 1, j, num, n)
    y = max_sum_recursive(i + 1, j + 1, num, n)
    return max(x, y) + num[i][j]
```

数字三角形——递归

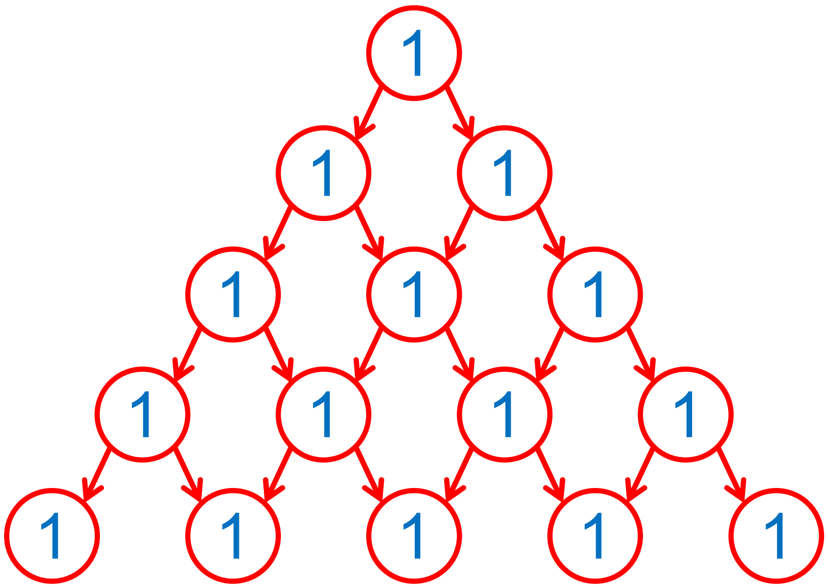
- 递归？
  - 时间复杂度：$O(2^n)$
  - 大量重复计算



每个子问题的计算次数

数字三角形——“记忆化”

- 递归 → “记忆化”
  - 避免重复计算，保证每个max_sum值只计算一次
  - 时间复杂度：$O(n^2)$



每个子问题的计算次数

数字三角形——“记忆化”

- 递归 → “记忆化”
  - 避免重复计算，保证每个max_sum值只计算一次
  - 时间复杂度：$O(n^2)$

```python
def max_sum_memoized(i, j, num, n, max_sum):
    if max_sum[i][j] != -1:
        return max_sum[i][j]
    if i == n - 1:
        max_sum[i][j] = num[i][j]
    else:
        x = max_sum_memoized(i + 1, j, num, n, max_sum)
        y = max_sum_memoized(i + 1, j + 1, num, n, max_sum)
        max_sum[i][j] = max(x, y) + num[i][j]
    return max_sum[i][j]
```

数字三角形——递推形式

- “记忆化”递归 → 递推形式
  - 避免递归带来的函数调用开销
  - “自下而上”依次计算：max_sum[i][j]只与max_sum[i+1][j], max_sum[i+1][j+1] 相关

数字三角形——递推形式

- “记忆化”递归 → 递推形式
  - 避免递归带来的函数调用开销
  - “自下而上”依次计算：max_sum[i][j]只与max_sum[i+1][j], max_sum[i+1][j+1] 相关

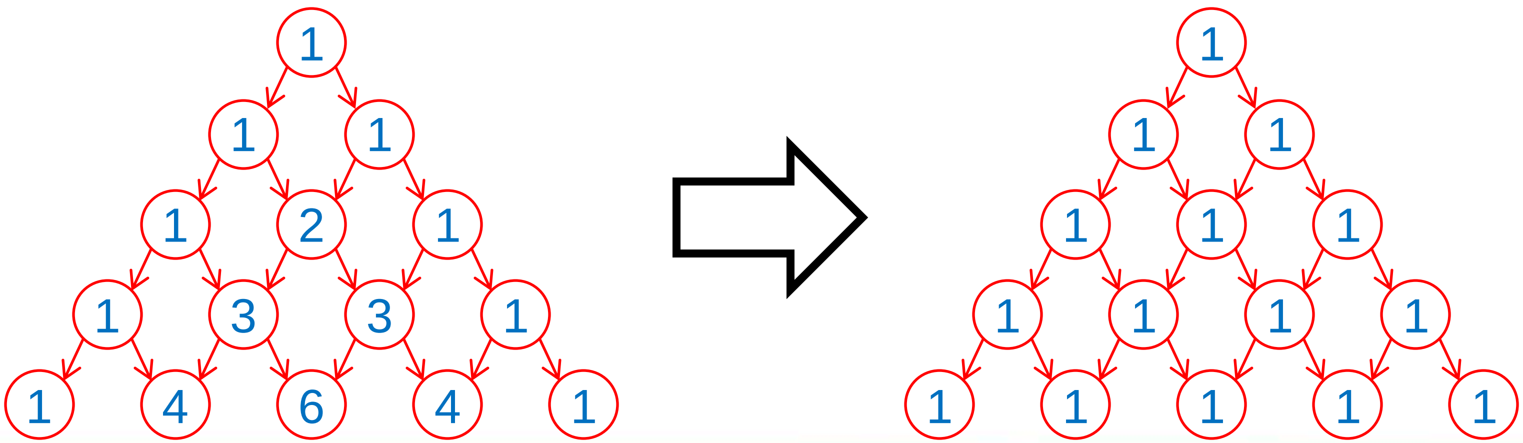
```python
def max_sum_iterative(n, num):
    max_sum = [[0] * (n + 1) for _ in range(n + 1)]
    for i in range(n - 1, -1, -1):
        for j in range(i + 1):
            max_sum[i][j] = max(max_sum[i + 1][j], max_sum[i + 1][j + 1]) + num[i][j]
    return max_sum[0][0]
```

动态规划——基本概念

- 动态规划是一类将原问题分解为相对简单的子问题的方式来求解复杂问题的方法，适用于：
  - 重叠子问题
  - 最优子结构

动态规划——基本概念

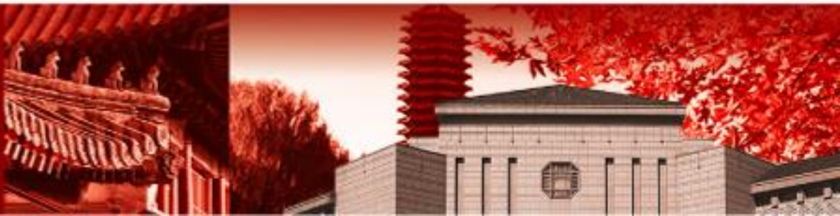
- 重叠子问题
  - 把原问题分解为若干个子问题，子问题和原问题形式相同或类似，只不过规模变小
  - 每个子问题只需求解一次，避免重复计算



动态规划——基本概念

- 最优子结构
  - 即局部最优解（子问题）能决定全局最优解（原问题）
  - 如何决定？ 由状态转移方程给出

$$max\_sum[i][j] = max(\underbrace{max\_sum[i+1][j]}_{\text{子问题1}}, \underbrace{max\_sum[i+1][j+1]}_{\text{子问题2}}) + num[i][j]$$



动态规划——基本概念

- 最优子结构
  - 要求子结构无后效性：即子问题的解一旦确定，就不再改变，不受在这之后、包含它的更大的问题的求解决策影响

动态规划——基本概念

- 动态规划、递归、递推（迭代）的区别与联系？

动态规划——基本概念

- 动态规划是<u>算法思想</u>
- 递归、递推（迭代）是<u>实现方式</u>

动态规划

- **递归实现，即“记忆化”递归**

确定求解顺序

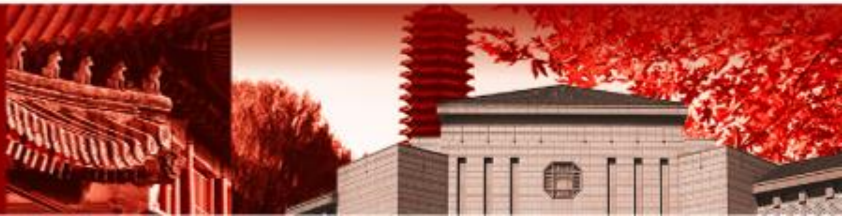
- **递推（迭代）实现**

0-1背包问题

我们有n种物品，物品i的重量w[i]，价格为v[i]。重量和价格都是非负的。背包所承受的重量最大为W。如果限定每种物品（0）要么完全放进背包（1）要么不放进背包；即物品是不可分割的。问：如何选择这n中物品，保证放进背包的物品总价格最大。

index	0	1	2	3
w[i]	4	2	2	3
v[i]	3	1	2	10

n = 4, W = 6



0-1背包问题

- **动态规划**
  - 确定子问题（最优子结构）： **f[i][j]**表示前**i**个物品，装入容量为**j**的背包，所能选取的最大价值。
  - 原始问题：**f[n][W]**

0-1背包问题

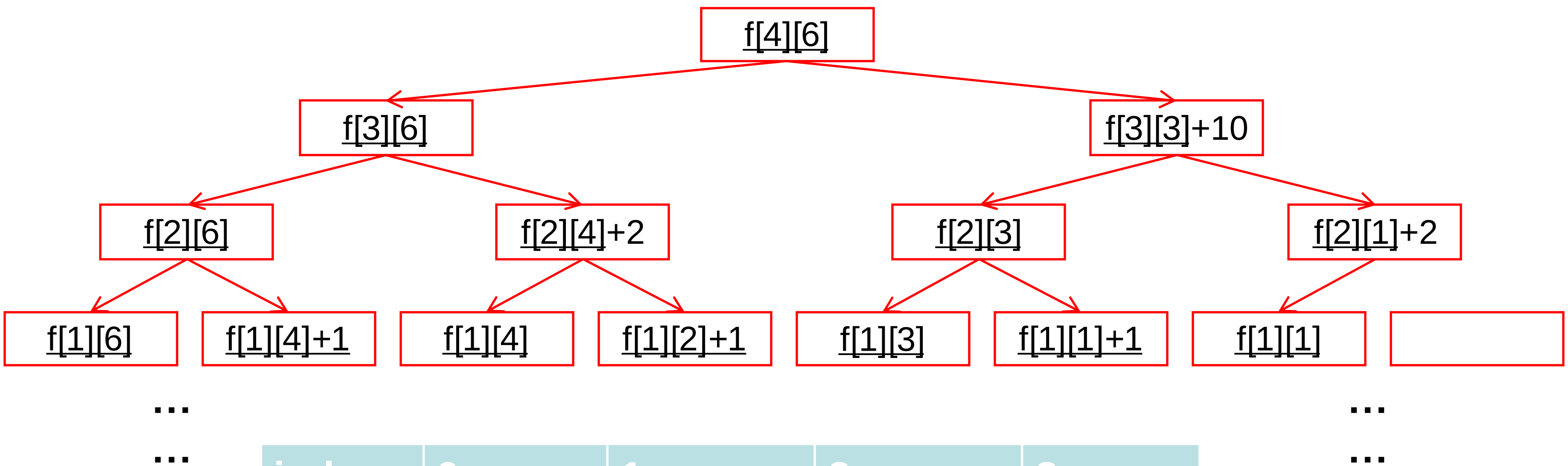
- **动态规划**
  - 状态转移方程：$f[i][j] = \max(\underbrace{f[i-1][j]}_{\text{不取物品}i}, \underbrace{f[i-1][j-w[i-1]]+v[i-1]}_{\text{取物品}i})$
  - 无后效性：$f[i][j]$的求解与$f[i+x][j+y]$ $(x,y > 0)$无关

0-1背包问题

- **动态规划**
  - 状态转移方程：$f[i][j] = \max(\underbrace{f[i-1][j]}_{\text{不取物品}i}, \underbrace{f[i-1][j-w[i-1]]+v[i-1]}_{\text{取物品}i})$
  - 无后效性：$f[i][j]$的求解与$f[i+x][j+y]$ $(x,y > 0)$无关

```python
def dp_knapsack_memoized(w, v, c, f, n):
    if f[n][c] != -1:
        return f[n][c]
    if n == 0:
        f[n][c] = 0
        return f[n][c]
    f[n][c] = dp_knapsack_memoized(w, v, c, f, n - 1)
    if c - w[n - 1] >= 0:
        f[n][c] = max(f[n][c], v[n - 1] + dp_knapsack_memoized(w, v, c - w[n - 1], f, n - 1))
    return f[n][c]
```

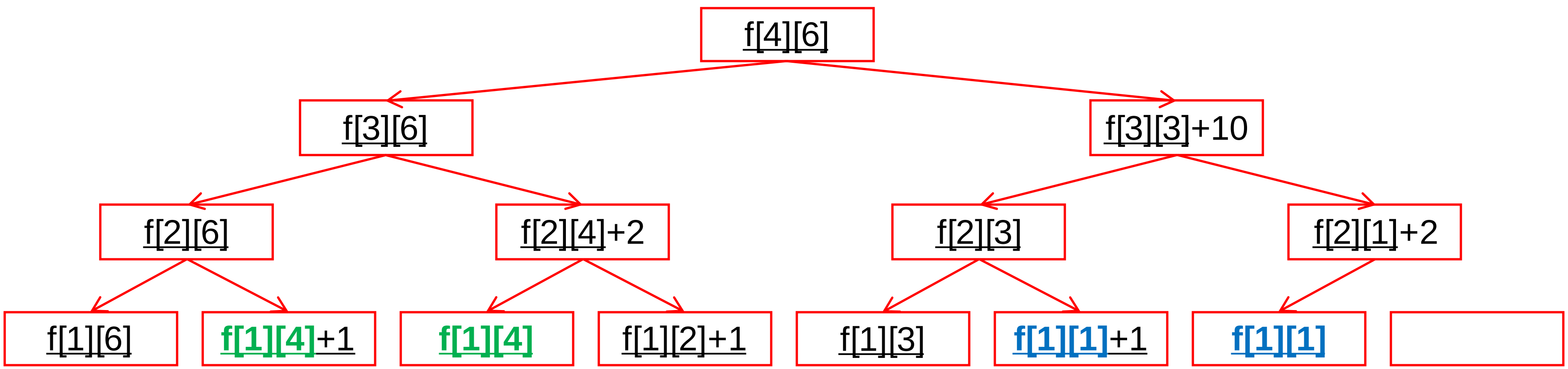
0-1背包问题



index	0	1	2	3
w[i]	4	2	2	3
v[i]	3	1	2	10

n = 4, W = 6

0-1背包问题



… 重叠子问题！ …

… …

index	0	1	2	3
w[i]	4	2	2	3
v[i]	3	1	2	10

n = 4, W = 6

0-1背包问题——递推实现

- **递归实现，即 “记忆化”递归**

确定求解顺序

f[i][j] 求解只与 f[i-1][j]和f[i-1][j-w[i-1]]有关

- **递推（迭代）实现**

f[i][j] = max(f[i-1][j], f[i-1][j-w[i-1]]+v[i-1])

0-1背包问题——递推实现

```python
def dp_knapsack_iterative(w, v, W, f, n):
    for j in range(W + 1):
        f[0][j] = 0
    for i in range(1, n + 1):
        for j in range(W + 1):
            if j - w[i - 1] >= 0:
                f[i][j] = max(f[i - 1][j], f[i - 1][j - w[i - 1]] + v[i - 1])
            else:
                f[i][j] = f[i - 1][j]
    return f[n][W]
```

0-1背包问题——动态规划

- **“记忆化”递归实现**
  - 只计算必要的子状态，“惰性求值”
- **递推实现**
  - 每个子状态**f[i][j]**都恰好计算一次

0-1背包问题 – 动态规划

n \ W	0	1	2	3	4	5	6
0		0	0	0	0		0
1		0	0	0	3		3
2		0		1	3		4
3				2			5
4							12

“记忆化”递归

n \ W	0	1	2	3	4	5	6
0	0	0	0	0	0	0	0
1	0	0	0	0	3	3	3
2	0	0	1	1	3	3	4
3	0	0	2	2	3	3	5
4	0	0	2	10	10	12	12

递推

index	0	1	2	3
w[i]	4	2	2	3
v[i]	3	1	2	10

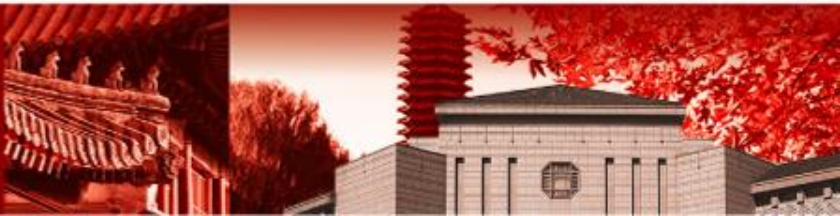
n = 4, W = 6

字符串编辑距离

字符串的编辑距离，又称为Levenshtein距离，由俄罗斯的数学家Vladimir Levenshtein在1965年提出。是指利用字符操作，把字符串A转换成字符串B所需要的最少操作数。其中，字符操作包括：

- 删除一个字符　　e.g. abcd → abd
- 插入一个字符　　e.g. abd → abcd
- 修改一个字符　　e.g. abcd → abed

问：给定两个字符串A和B，长度分别为n和m，求字符串A至少经过多少步字符操作变成字符串B。



字符串编辑距离

- 步骤：
  - 确定子问题
  - 子问题是否满足最优子结构性质？且无后效性？
  - 根据最优子结构性质写出状态转移方程
  - 子问题是否具有重叠性质？
  - 求解顺序？
  - 进一步优化时间、空间复杂度？

字符串编辑距离

- 子问题
  - f[i][j]表示：字符串A[0, …, i-1]与字符串B[0, …, j-1]之间的编辑距离
- 原问题
  - f[n][m]
- 状态转移方程
  - $$\mathbf{f[i][j]} = \begin{cases} \mathbf{f[i-1][j-1]}, & \mathbf{if\ A[i-1] == B[j-1]} \\ \mathbf{MIN\{\underbrace{f[i][j-1]+1}_{\text{插入B[j-1]}}, \underbrace{f[i-1][j]+1}_{\text{删除A[i-1]}}, \underbrace{f[i-1][j-1]+1}_{\text{A[i-1]替换为B[j-1]}}\}}, & \mathbf{if\ A[i-1]\ !=B[j-1]} \end{cases}$$

字符串编辑距离——“记忆化”递归

```python
def edit_distance_memoized(A, B, f, n, m):
    if f[n][m] != -1:
        return f[n][m]
    if n == 0:
        f[n][m] = m
        return f[n][m]
    elif m == 0:
        f[n][m] = n
        return f[n][m]
    elif A[n - 1] == B[m - 1]:
        f[n][m] = edit_distance_memoized(A, B, f, n - 1, m - 1)
    else:
        f[n][m] = min(edit_distance_memoized(A, B, f, n, m - 1) + 1,
                      min(edit_distance_memoized(A, B, f, n - 1, m) + 1,
                          edit_distance_memoized(A, B, f, n - 1, m - 1) + 1))
    return f[n][m]
```

字符串编辑距离——递推实现

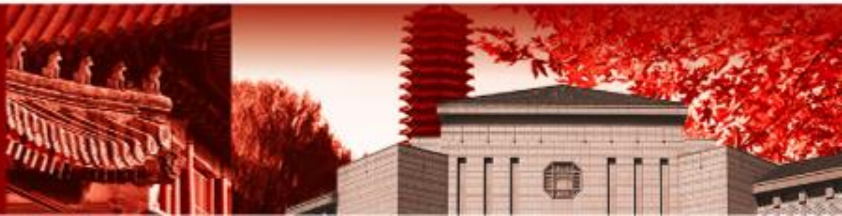
- 确定求解顺序：
  - f[i][j]只与f[i-1][j-1]，f[i][j-1]，f[i-1][j]有关
  - “从上到下，从左到右”

字符串距离——递推实现

```python
def edit_distance_iterative(A, B, n, m):
    f = [[0] * (m + 1) for _ in range(n + 1)]
    for i in range(n + 1):
        f[i][0] = i
    for j in range(m + 1):
        f[0][j] = j
    for i in range(1, n + 1):
        for j in range(1, m + 1):
            if A[i - 1] == B[j - 1]:
                f[i][j] = f[i - 1][j - 1]
            else:
                f[i][j] = min(f[i][j - 1] + 1, min(f[i - 1][j] + 1, f[i - 1][j - 1] + 1))
    return f[n][m]
```

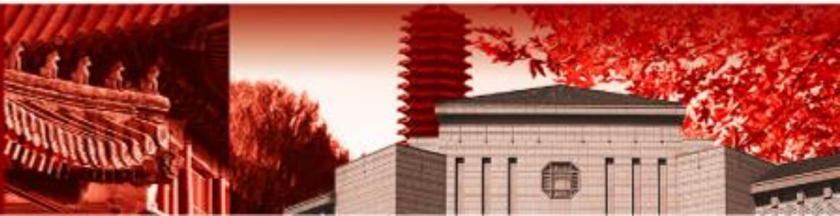
k叉树

- 一棵高度为h的满k叉树有如下性质：根结点所在层次为0；第h层上的结点都是叶子结点；其余各层上每个结点都有k棵非空子树，如果按层次自顶向下，同一层自左向右，顺序从1开始对全部结点进行编号，试问：
- (1) 各层的结点个数是多少?
- (2) 编号为i的结点的第m个孩子结点（若存在）的编号是多少？
- (3) 编号为i的结点有右兄弟的条件是什么？其右兄弟结点的编号是多少？
- 请简要写出推算过程。



k叉树

- 一棵高度为h的满k叉树有如下性质：根结点所在层次为0；第h层上的结点都是叶子结点；其余各层上每个结点都有k棵非空子树，如果按层次自顶向下，同一层自左向右，顺序从1开始对全部结点进行编号，试问：
- (1) 各层的结点个数是多少?
- k^l(l为层数，按题意，根结点为0层)



k叉树

- 一棵高度为h的满k叉树有如下性质：根结点所在层次为0；第h层上的结点都是叶子结点；其余各层上每个结点都有k棵非空子树，如果按层次自顶向下，同一层自左向右，顺序从1开始对全部结点进行编号，试问：
- (2) 编号为i的结点的第m个孩子结点（若存在）的编号是多少？
- 结点i的最右边的子女为$i*k+1$，故结点 i的第 m个孩子的编号是$(i-1)*k+1+m$。

k叉树

- 一棵高度为h的满k叉树有如下性质：根结点所在层次为0；第h层上的结点都是叶子结点；其余各层上每个结点都有k棵非空子树，如果按层次自顶向下，同一层自左向右，顺序从1开始对全部结点进行编号，试问：
- (3) 编号为i的结点有右兄弟的条件是什么？其右兄弟结点的编号是多少？
- 根据以上分析，结点i有右兄弟的条件是，它不是双亲的第k子女，即 (i-1)%k!=0，其右兄弟编号是i+1

二叉搜索树

- 在一棵表示有序集S的无重复元素二叉搜索树中，任意一条从根到叶子结点的路径将S分为3个部分：在该路径左边结点中的元素组成的集合S1；在该路径上的结点中的元素组成的集合S2；在该路径右边结点中的元素组成的集合S3。S = S1∪S2∪S3。若对于任意的a∈S1，b∈S2，c∈S3，判断以下表达式是否总是成立，若成立，简要叙述理由，若不成立，给出反例：
- 1）a<b 2）b<c 3）a<c

二叉搜索树

- 1）2）均不总是成立

```
        11
       /  \
      5    12
     / \
    2   7
       / \
      6   9
         / \
        8   10
```

6>5,11>10

二叉搜索树

- 3）考虑a跟c在S2中的第一个祖先 b1 跟 b2

1. b1 < b2 则 a < b1 < b2 < c

- 2. b1 > b2
- 这不可能， b1 要么处于 b2 的右子树中，要么 b2 处于b1的左子树中。
- 因为 b1 处于 b2 的右子树时，S2经过b2往右走，这样c的第一个祖先就不可能是b2，至少是b2的右儿子，矛盾
- 因为b2 处于b1的左子树时， S2经过b1往左走，这样a的第一个S2祖先就不是b1，矛盾