高级算法设计复习

# ASYMPTOIC ANALYSIS

T(n) = O(f(n)) if there exist positive constant c and n0 such that T (n) ≤ cf (n) when n ≥ n0. cf(n) is an upper bound for T(n).

T (n) = Ω (g(n)) if there exist positive constant c and n0 such that T (n) ≥ cg (n) when n ≥ n0. Cg(n) is an lower bound for T(n).

T (n) = Θ( f (n) ) if and only if T (n) = O ( f (n) ) and T (n) = Ω (f (n) )

可以用极限工具来确定T(n)和f(n)的关系。

分治法：归并排序

### 主方法（The master method）

对T(n)=aT(n/b)+f(n)，a≥1，b>1，存在，，有：

1. 若)，则
2. 若)，则
3. 若)且存在使得，则.

不适用主方法的例子：

# SORTING ALGORITHM

表格

描述已自动生成

Shell排序

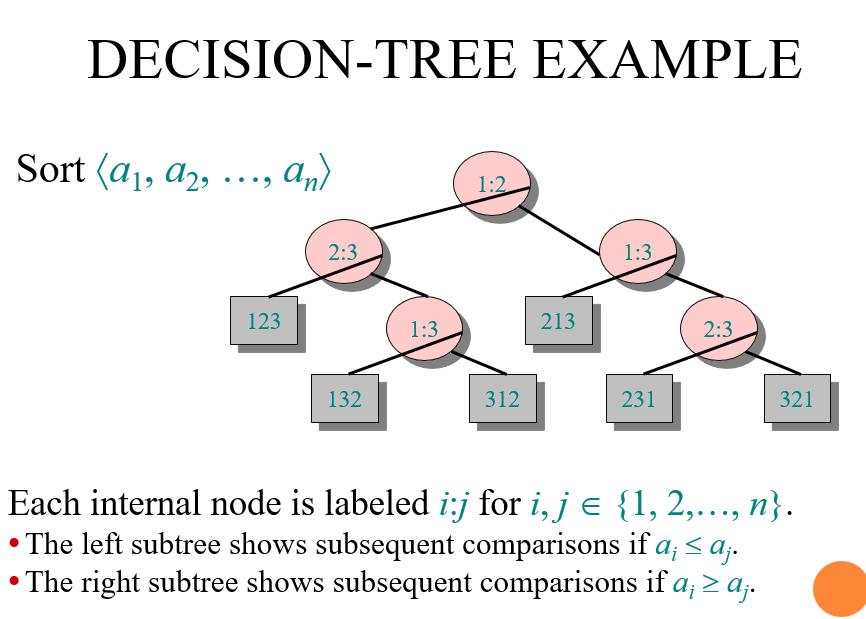
每一轮按照事先决定的间隔（降序）进行插入排序，间隔会依次缩小，最后一次一定要是1。

稳定性：不稳定。示例：3，2，2，1，3。以间隔为[2, 1]进行希尔排序，结果2调换。

复杂度：约O(n^1.25)

基于比较的排序算法下界O(nlgn)：使用决策树证明

构造决策树如下：



这样的决策树有n!个叶子节点，而一棵完全二叉树有2^n个节点，因此n!≤2^n。根据斯特林公式（Stirling’s approximation）：，因此。

计数排序

统计小于等于该元素值的个数i，然后把该元素放在第i位。

长度为n的待排序数组A，A的数值范围为1~k。长度为n的输出数组B，长度为k的辅助数组C。先用C记录每个A中元素的个数，然后令C[i+1]=C[i+1]+C[i]，这时C[i]为所有小于等于元素i的个数。倒序遍历A，让B[C[A[i]]]=A[i], C[A[i]]=C[A[i]]-1。

稳定性：稳定

复杂度：Θ(n+k)

基数排序

从小端的数字开始进行计数排序，然后对大端进行排序。

稳定性：稳定

复杂度：考虑待排序的数字有n个，每个数字是b位r进制的，则算法复杂度为Θ(b(n+r))

桶排序

假设待排序数字位于[a,b]区间，将它们均匀分配到m个桶上，上一个桶的最大数小于下一个桶的最小数，桶用链表实现。

先把所有数字分配到对应的桶中，然后对每个桶做插入排序，最后合并每个桶的结果。

稳定性：稳定

复杂度：Θ(n)

# HASH TABLE

简单均匀哈希：每个key有相等的概率独立地被hash到任一slot。

令n是keys的数量，m是slots的数量，则α=n/m是装载因子（load factor）。一次失败的查询期望时间是，1是应用哈希函数和访问slot的时间，α是搜索列表的时间。如果α=O(1)，则期望时间为Θ(1)。

Division method：h(k) = k mod m，取m为大质数

Multiplication method

开放地址（open addressing）

构造哈希函数h(k,i)，0≤i≤m-1。尝试插入k时，第一次先尝试h(k,0)，失败后尝试h(k,1)…

线性探测（linear probing）：

h(k,i)=(h’(k)+i) mod m

二次探测（quadratic probing）：

h(k,i)=(h’(k)+c1i+c2i^2) mod m

Double Hashing：

h(k,i)=(h1(k)+ih2(k)) mod m

在开放地址的均匀哈希表中，给定装载因子α=n/m<1，那么在一次失败的搜索中，期望的探测次数为1/(1-α)。

# BINARY SEARCH TREE

对一棵二叉树，它的每个节点都满足：左孩子节点的key值小于等于该节点的key值小于等于右孩子节点的key值。

操作：前驱，后继，插入，删除

* 前驱：每个节点的前驱是它中序遍历的前驱。要么是其左子树的最大值，要么是其作为右孩子的最近祖先。
* 后继：每个节点的后继是它中序遍历的后继。要么是其右子树的最小值，要么是其作为左孩子的最近祖先。
* 删除：如果是叶子节点直接删除；如果有一个孩子节点，则与孩子节点交换，然后删除；如果有两个孩子节点，则与其后继交换，然后删除。

使用BST进行排序与快排比较哪个好？快排，因为它是本地排序，且不用构建数据结构。

使用全局变量/栈来保存前驱。

随机构建的BST：

随机构建的BST的树高为O(lgn)，这意味着求前驱、后继、插入、删除等操作的复杂度都是O(lgn)。

证明随机构建的BST树高为O(lgn)：证明其高h的期望满足2^h≤O(n^3)。

# RED BLACK TREE

在BST基础上增加了颜色属性，能确保构建的树的树高为O(lgn)。

定义：

* 每个节点要么是红的要么是黑的
* 每个叶子节点（这里指的是最后的空指针）是黑的
* 如果节点是红的，那么它的左右孩子是黑的
* 从每个节点出发，到所有后续的叶子节点（空指针）的每条路径，都有相同数量的黑色节点
* 树根是黑色的

证明：一棵n个节点的红黑树，它的高度h≤2lg(n+1)

定义：以为树根，到其叶子节点路径上的黑色节点个数。

引理：以为根节点的子树至少含有个内部节点。对树高使用归纳法证明：

1. ，此时必为黑色，（算上空节点），则，为真
2. 假设时都成立，当时：，如果是红色的。否则，。节点的左右子树的树高为，假设成立，则左右子树至少含有个节点，因此至少有个节点，证明完毕。

对一棵个节点的红黑树，它的树高满足：

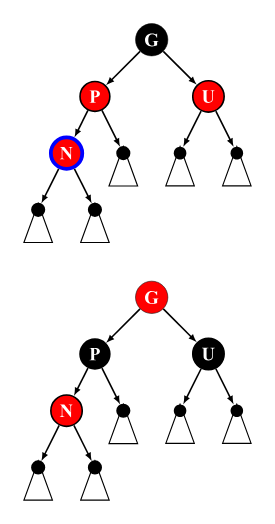
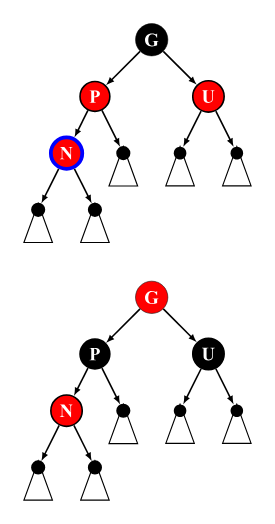
（红黑相间时最小）

（由引理）

（取对数）

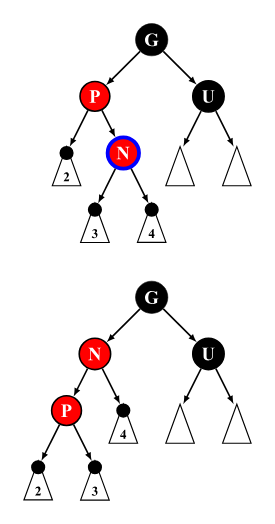
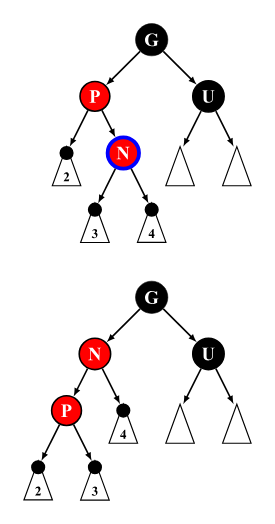
操作：插入和删除

* 插入：默认插入红色节点N，违反定义后进行调整。
  + CASE1：插入位置的父节点P是黑色节点，无需调整。
  + CASE2：插入位置的父节点P是红色节点，且其叔叔节点U也是红色节点。



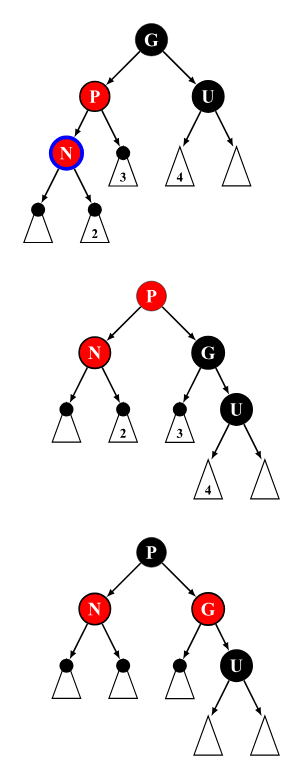
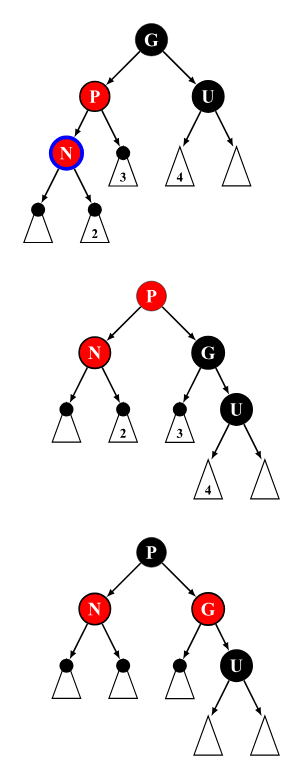
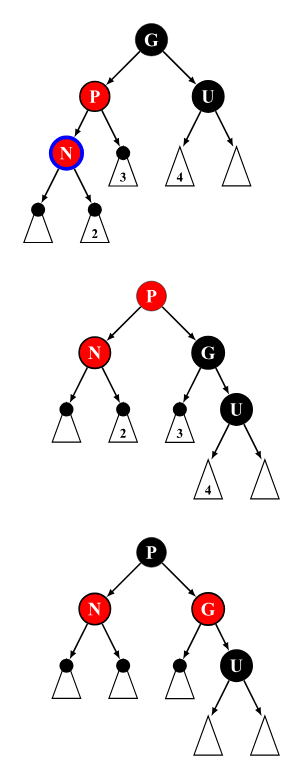
将G变为红色，将P和U变成黑色，并对节点G递归维护，直到所有节点满足定义

* + CASE3：插入位置的父节点P是红色节点，叔叔节点U是黑色节点，且P和N方向不同。



对P进行左旋转，进入CASE4进行调整。

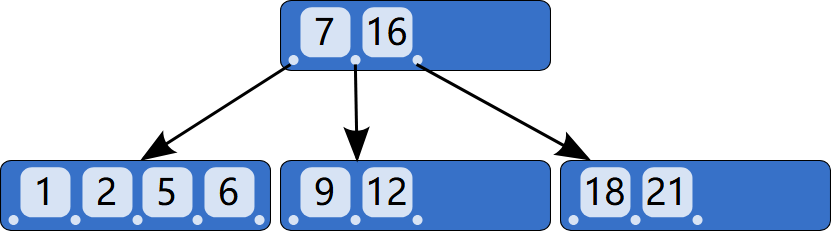
* + CASE4：插入位置的父节点P是红色节点，叔叔节点U是黑色节点，且P和N方向相同。



如果N是左子节点：将G进行右旋转，再将P和G反色。

如果N是右子节点：将G进行左旋转，再将P和G反色。

# B TREE



箱线图

描述已自动生成

n[x]：节点所含keys的个数

ci[x]：第i个指向孩子节点的指针

leaf[x]：x是否是叶子节点

keyi[x]：第i个key

B树的最小度数（***minimum degree***）t≥2：B树的节点至少要有t-1个keys，即至少有t个孩子。至多有2t-1个keys，即至少有2t个孩子。

这与中文中的阶数定义略有区别，说一棵B树是m阶的，是说它最多有m个孩子，至少有m/2（向上取整）个孩子。

B树的树高h满足，尽管在最坏的情况下它与其他树结构复杂度相同，但它倾向于更少的访问节点和更低的树高。

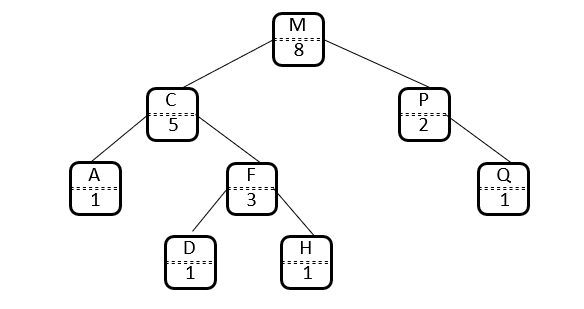
查找操作复杂度：

操作：插入和删除

* 插入
  + 先找到插入位置并插入
  + 插入后如果节点超过2t-1个keys，则需要进行分裂操作
  + 分裂：节点的中位数key（第t个）上升到父节点，左边的key保留，右边的key放到新节点上
  + 递归地对父节点进行维护，如果它们需要分裂
* 删除
  + 先找到删除位置并删除。删除该元素后，首先判断该元素是否有左右孩子节点。如果有，则上移孩子节点中的某相近元素（「左孩子最右边的节点」或「右孩子最左边的节点」）到父节点中。如果没有，不做其他操作。
  + 对减少key值的节点进行维护：
    - 满足定义，不做其他操作。
    - 不满足定义，做下面操作：
      * 尝试向有多余key值的左右兄弟节点借一个key值，操作为先下移一个父亲节点的key值给当前节点，然后兄弟节点上移一个key值补充给父亲节点，以满足定义。
      * 如果左右兄弟节点都没有多余的key值，那么需要合并。合并操作将该节点与相邻的某一兄弟节点合并，并下移对应的父亲节点的key值当作中位数key值。
  + 第二步需要递归的进行，直到所有节点满足定义。

# ORDER STATISTIC TREE

在红黑树的基础上，增加一个size属性，用于快速计算一个元素的顺序统计量。



size属性：以该节点为树根的子树所含的节点数量。

rank属性：该节点在n个节点中排在第rank个。

显然，size[x]=size[left[x]]+size[right[x]]+1。

计算rank[x]：复杂度为树高O(lgn)

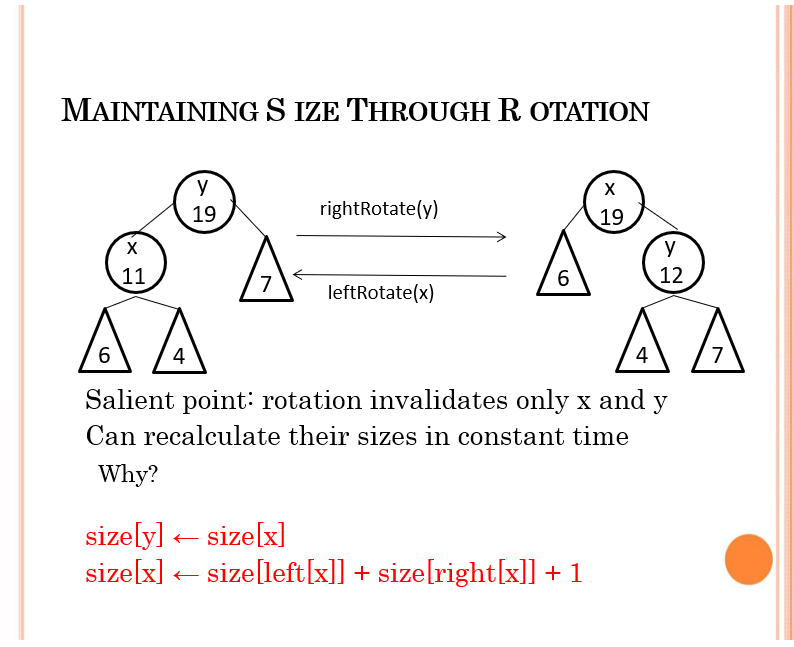
先令rank[x]=size[left[x]]+1，再往上找它的父节点，如果x是父节点p的右子节点，则令rank[x]=rank[x]+size[p.left]+1，循环，直到找到根节点为止。

维护OS TREE：

插入节点时，令搜索路径上的所有节点的size加1，使用size[x]=size[left[x]]+size[right[x]]+1来初始化插入节点的size。

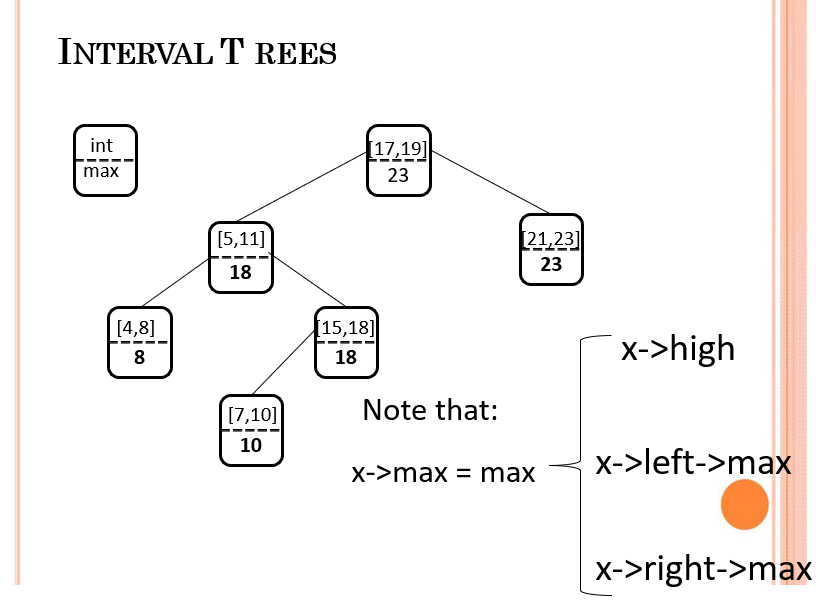
删除节点时，令搜索路径上的所有节点的size减1。删除后如果需要其前驱或者后继节点将其替换，则进一步将到前驱或者后继节点的路径上的所有节点size减1，然后更新替换节点的size。

维护阶段遇到旋转操作，使用下面的公式更新size[x]。

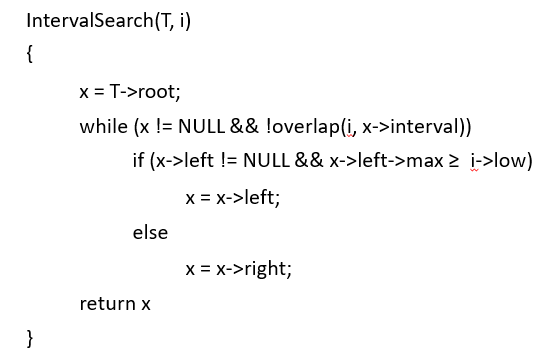


# INTERVAL TREE

在红黑树的基础上，每个节点增加一个区间属性和一个max属性，以区间的low当作key值。



给定区间i，搜索树中第一个与区间i重复的区间



算法的正确性：

考虑以x为根节点的区间树。若给定的区间i不与x重叠，则考虑x的左孩子和右孩子。

先证明满足if条件时，目标区间在x的左子树，否则目标区间根本不存在。

此时，x.left.max≥i.low。如果目标区间在x的左子树，结束。否则，x.left.max来自于左子树的某一个节点y.int.high，则y.int.high≥i.low。要想目标区间不存在于左子树，必须有y.int.low≥i.high，此时由于区间的low作为key值，右子树的所有区间的low都大于i.high，右子树也不存在目标区间。

再证明不满足if条件时，目标区间在右子树，否则目标区间根本不存在。

此时，x.left==NULL或x.left.max<i.low。如果目标区间在x的右子树，结束。否则，容易知道左子树也不存在目标区间。

维护INTERVAL TREE

插入：对root到插入位置这条路径上的所有节点，比较它们的max与interval.high的大小。

删除：先向下找前驱或后继节点替换，递归维护到前驱或后继节点的路径上所有节点的max。然后再向上维护到root的max。

维护旋转时的max是简单的，子树根节点的max不变，被带动旋转的那个节点需要重新计算，其他节点都不变。

# TREAP

BST+HEAP

在BST的基础上，为每个节点增加属性priority。新增定义：父节点的priority比其孩子节点小（更优先）。

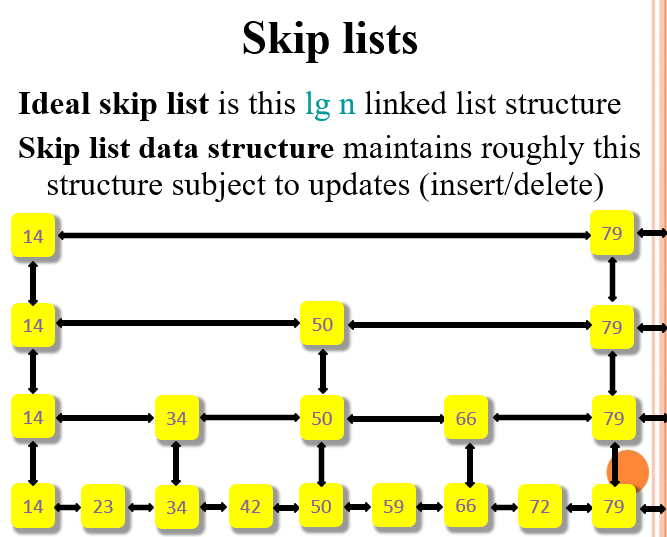
插入时，将节点按照BST的规则插入。如果插入后，不满足定义，则通过不断旋转将新插入的节点往上移，直到符合定义位为止。

TREAP中，插入给定的一系列节点keyi和priorityi，得到的TREAP是唯一的，而与插入节点的顺序无关。

复杂度：Θ(lgn)

# SKIP LIST

lgn条排序好的链表。



查找：

手机屏幕的截图

描述已自动生成

插入：

先找到要插入的位置，然后以0.5的概率决定它是否要提升到上一级，遍历到最高层。

删除：删除所有层级中的该节点。

With high probability, 在n个元素的跳表中每次查询花费为O(lgn)

With high probability, n个元素的跳表有O(lgn)级

# AMORITIZED ANALYSIS

Aggregate Method

直接对每个操作的花费进行求和

Accounting Method

为每个操作假设一个均摊代价（amortized cost），确保，因此是总花费的一个上界。

使用银行法来说明。每次将存入银行，遇到时花费存款，确保银行的余额不为负。

Potential Method

每个操作使得数据结构发生变化，为每个结构状态定义一个势能函数。

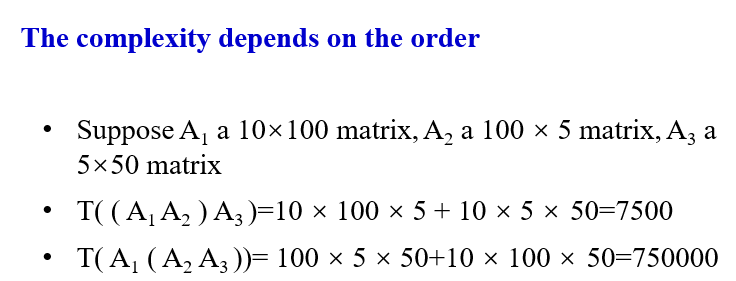
文本

描述已自动生成

# DYNAMIC PROGRAMMING

矩阵链乘法

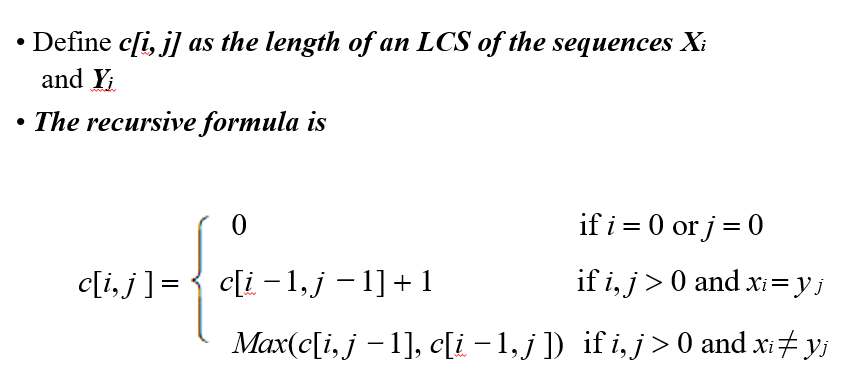
矩阵连乘中间打括号问题。



图片包含 文本

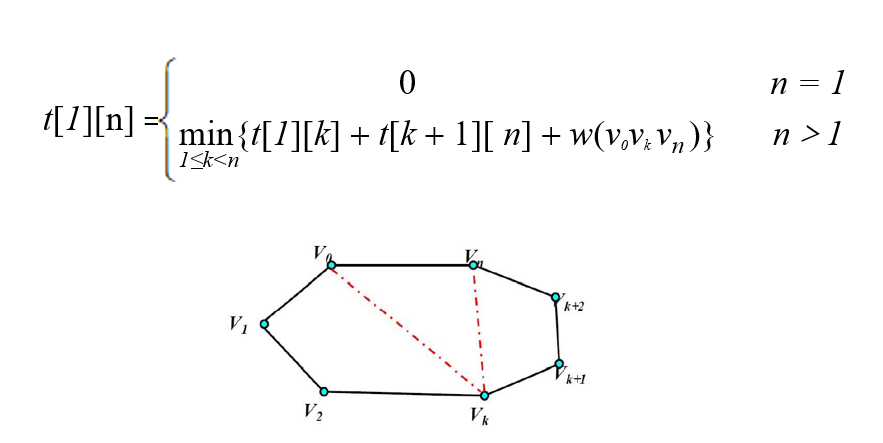
描述已自动生成

最长公共子序列



凸多边形的三角分解

与矩阵链乘法十分相似，在凸边形中选择一个点vk，连接(v0,vk)和(vk,vn)，构成如下结构：



其中是凸多边形最优三角剖分的权重。

最优二叉搜索树

本质与矩阵链乘法一样，选择一个节点当作根节点。其他节点高度加一，增加的权重为，但它的权值是推理出来的。

其中。

0-1背包

放与不放两种状态。

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

[**最长递增子序列的个数**](https://leetcode.cn/problems/number-of-longest-increasing-subsequence/)

[718. 最长重复子数组 - 力扣（LeetCode）](https://leetcode.cn/problems/maximum-length-of-repeated-subarray/description/)

[最长回文子串 - 力扣（LeetCode）](https://leetcode.cn/problems/longest-palindromic-substring/description/)

[96. 不同的二叉搜索树 - 力扣（LeetCode）](https://leetcode.cn/problems/unique-binary-search-trees/)

[最长递增子序列 - 力扣（LeetCode）](https://leetcode.cn/problems/longest-increasing-subsequence/)

[最长回文子序列 - 力扣（LeetCode）](https://leetcode.cn/problems/longest-palindromic-subsequence/)

[139. 单词拆分 - 力扣（LeetCode）](https://leetcode.cn/problems/word-break/)

# GREEDY ALGORITHM

贪心选择性（Greedy-choice property）：每一步的贪心选择推导出全局最优

最优子结构（Optimal substructure）：原问题的最优解包含子问题的最优解

以活动选择问题为例：

贪心选择性：归纳法证明，证明第一步的贪心步骤是最优解的一部分，假设前k步都是最优解的一部分，证明第k+1步是最优的

最优子结构：选择某个活动ai后，被它分成的两个子问题最优

# LINEAR PROGRAMMING

单纯形法：

化为标准形式，目标函数是max，取反使b向量不为负，加入松弛变量变为等式，大N法加入人工变量凑初始解，挑紧的界更新。

目标函数的权重全为负时算法结束。如果人工变量不为0，原问题无可行解。

# FFT

表格

描述已自动生成

卡通人物

低可信度描述已自动生成

注意原向量是乘在DFT矩阵的右边。

求逆时，F的逆是其共轭转置乘以一个常数。这是容易验证的。

# NP-COMPLETENESS

P问题：能在多项式时间内求解的问题

NP问题：给定一个解，能在多项式时间内验证该解是否正确的问题

A问题规约为B问题：A化为B。

例：不知道如何求解一元一次方程，但知道如何求解一元二次方程。说前者可以规约为后者。

NP难：存在这样一个问题，所有的NP问题都可以约化成它。

NP完全：存在这样一个NP问题，所有的NP问题都可以约化成它。

证明问题D是NP完全问题：将D规约到一个著名的NP完全问题。

形状

低可信度描述已自动生成