数据分析与算法设计

复习课

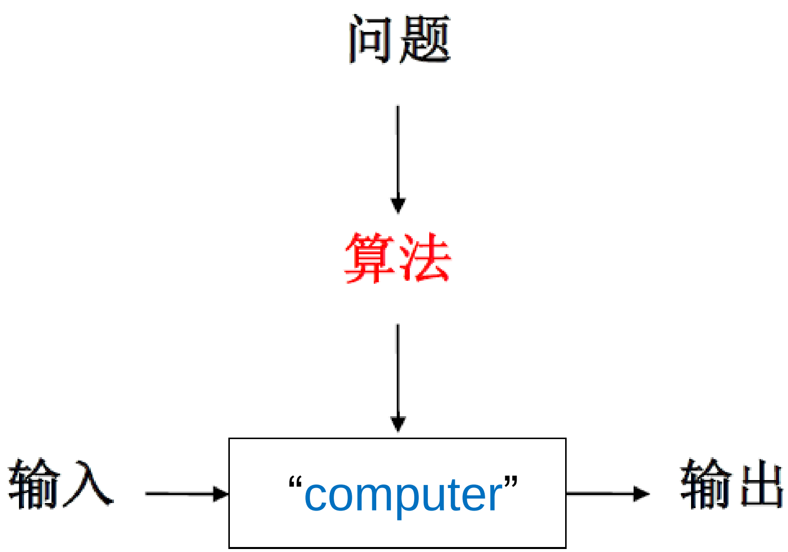
李旻
百人计划研究员
浙江大学 信息与电子工程学院
Email: min.li@zju.edu.cn

目录

- 算法基本概念
  - 效率分析理论
  - 算法能力的极限
- 算法的通用设计技术
- 典型应用
  - 排序问题
  - 查找问题
  - 字符串匹配问题
  - 图问题
  - 组合问题

算法的基本概念

- 算法是**一系列解决问题的明确指令**，也就是说，对于符合**一定规范的输入**，能够**在有限时间内获得所要求的输出**



- 所有算法必须要有1个以上输入？
- 是否所有问题都可以用算法求解？

➢ “computer”: 理解和执行指令的人或物

算法的效率分析理论

- 对于给定的问题及某个给定的算法，可用算法的**时间效率**作为算法运行**速度快慢的度量**
  - 非渐近效率
    - 输入规模
    - 基本操作、及执行次数

输入规模

$$\boxed{T(n) \approx c_{op}C(n)}$$

算法运行时间

每次基本操作的执行时间

基本操作的执行次数

举例

- 对于以下每种算法，请指出（i）其输入规模的合理度量标准；（ii）它的基本操作；（iii）对于规模相同的输入来说，其基本操作的次数是否会有不同？

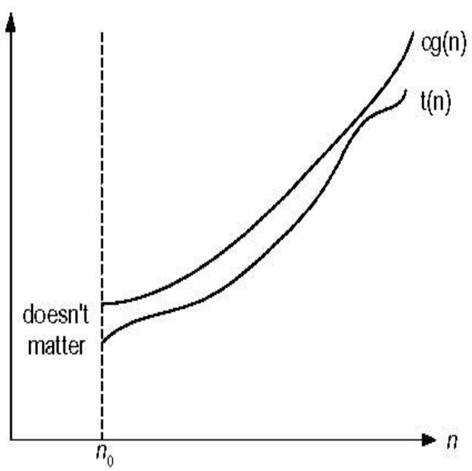
① 找出包含n个数字的列表的最大元素

② 计算n!

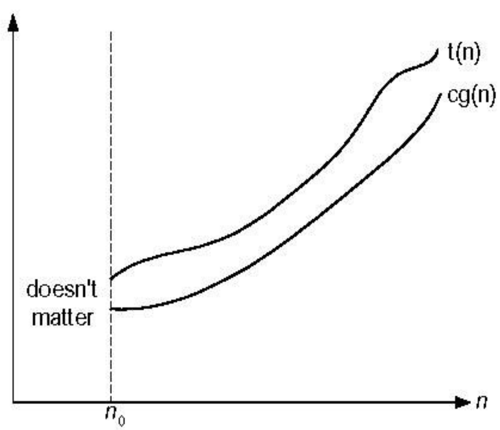
③ 欧几里得法求公约数

算法的效率分析理论

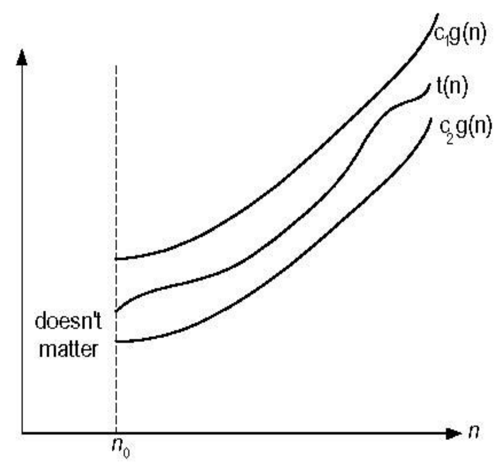
- 对于给定的问题及某个给定的算法，可用算法的时间效率作为算法运行速度快慢的度量
  - 渐近效率
    - 基本操作执行次数的增长次数（增长速率）



t(n) ∈ O(g(n))



t(n) ∈ Ω(g(n))



t(n) ∈ Θ(g(n))

算法的效率分析理论

- 渐近效率
  - 非递归算法：分析基本操作执行次数
  - 递归算法：利用基本操作执行次数的递推关系式分析

$$\lim_{n\to\infty}\frac{t(n)}{g(n)}=\begin{cases}0, & \Rightarrow t(n)\in O(g(n))\\ c>0, & \Rightarrow t(n)\in \Theta(g(n))\\ \infty, & \Rightarrow t(n)\in \Omega(g(n))\end{cases}$$

  - 分治算法：

$T(n) = aT(n/b) + f(n)$ 其中， $f(n) \in \Theta(n^d)$， $d \geq 0$

主定理: 当 $a < b^d$， $T(n) \in \Theta(n^d)$

当 $a = b^d$， $T(n) \in \Theta(n^d \log n)$

当 $a > b^d$， $T(n) \in \Theta(n^{\log_b a})$

注意：对符号O和Ω，类似的结论也成立

基本的效率类型

$$O(1) < O(\log n) < O(n) < O(n\log n) < O(n^2) < O(n^3) < O(2^n) < O(n!) < O(n^n)$$

类　型	名　称	注　　释
1	常量	为数很少的效率最高的算法，很难举出几个合适的例子，因为典型情况下，当输入的规模变得无穷大时，算法的运行时间也会趋向于无穷大
$\log n$	对数	一般来说，算法的每一次循环都会消去问题规模的一个常数因子(参见 4.4 节)。注意，一个对数算法不可能关注它的输入的每一个部分(哪怕是输入的一个固定部分)：任何能做到这一点的算法最起码拥有线性运行时间
n	线性	扫描规模为 n 的列表(例如，顺序查找)的算法属于这个类型
$n\log n$	线性对数	许多分治算法(参见第 5 章)，包括合并排序和快速排序的平均效率，都属于这个类型
n^2	平方	一般来说，这是包含两重嵌套循环的算法的典型效率(参见下一节)。基本排序算法和 n 阶方阵的某些特定操作都是标准的例子
n^3	立方	一般来说，这是包含三重嵌套循环的算法的典型效率(参见下一节)。线性代数中的一些著名的算法属于这一类型
2^n	指数	求 n 个元素集合的所有子集的算法是这种类型的典型例子。“指数”这个术语常常被用在一个更广的层面上，不仅包括这种类型，还包括那些增长速度更快的类型
$n!$	阶乘	求 n 个元素集合的完全排列的算法是这种类型的典型例子

举例

- 以下算法解决什么问题？基本操作是什么？基本操作执行次数为多少？算法的基本效率类型是什么？

```
ALGORITHM Enigma(A[0..n − 1, 0..n − 1])
    //Input: A matrix A[0..n − 1, 0..n − 1] of real numbers
    for i ← 0 to n − 2 do
        for j ← i + 1 to n − 1 do
            if A[i, j] ≠ A[j, i]
                return false
    return true
```

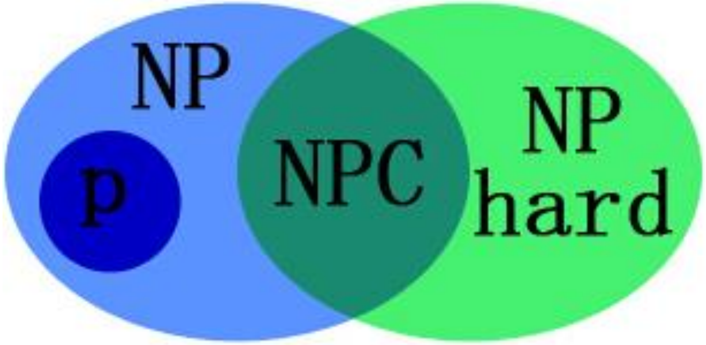
```
ALGORITHM Riddle(A[0..n − 1])
    //Input: An array A[0..n − 1] of real numbers
    if n = 1 return A[0]
    else temp ← Riddle(A[0..n − 2])
        if temp ≤ A[n − 1] return temp
        else return A[n − 1]
```

算法能力的极限

- 效率下界：对于给定的问题，解决该问题的任意算法最佳的最差效率
  - 若某算法和下界的效率类型相同，则该界是下确界，即当前算法的改进空间仅为一个常量因子
  - 平凡下界：对问题的输入中必须要处理的项进行计数，同时对必须要输出的项计数
  - 信息论下界：根据算法必须处理的信息量来建立效率的下界（决策树）
  - 敌手下界：基于一种恶意而又诚实的敌手逻辑—恶意使它不断地把算法推向最消耗时间的路径，而诚实又迫使它必须和已做出的选择保持一致

算法能力的极限

- P类问题：是一类能够用（确定性的）算法在多项式的时间内求解的判定问题
- NP类问题：是一类可以用不确定多项式算法求解的判定问题
- NP完全问题：一个判定问题D是NP完全问题，条件是
  - 它属于NP类型
  - NP中的任何问题都能够在多项式时间内化简为D



目录

- 算法基本概念
  - 效率分析理论
  - 算法能力的极限
- 算法的通用设计技术
- 典型应用
  - 排序问题
  - 查找问题
  - 字符串匹配问题
  - 图问题
  - 组合问题

算法设计技术

蛮力法：
基于问题描述及定义直接求解

改进 →

回溯法：
排除不需要考虑的部分解，减小搜索空间

改进 →

分支界限法：
回溯法应用于最优化问题的改进

深度优先　广度优先
状态空间树

减治法：
利用小规模问题与大规模问题解的关系，迭代或递归得到大规模问题的解

分治法：
分别求解若干不交叠的子问题，并将子问题的解合并为原问题的解

变治法：
将问题进行某种变换后，变成更为容易求解的问题实例

部分解->完整解

子问题

变问题

变输入

变解

贪婪技术：
迭代构造问题的解，每一步扩展目前的部分解，直到获得问题的完整解

动态规划：
求解的问题由交叠的子问题构成，对每个子问题只求解一次并把结果记录在表中，并从表中得出原问题的解

时空权衡：
通过输入增强或预构造技术，实现空间的增加换取时间效率的提升

迭代改进：
从满足问题约束的某些可行解出发，通过重复应用一些简单的步骤来不断地改进它，从而得到最终解

目录

- 算法基本概念
  - 效率分析理论
  - 算法能力的极限
- 算法的通用设计技术
- 典型应用
  - 排序问题
  - 查找问题
  - 字符串匹配问题
  - 图问题
  - 组合问题

排序问题

- 选择排序、冒泡排序（蛮力法）

$$A_0 \leqslant A_1 \leqslant \cdots \leqslant A_{i-1} \mid A_i, \cdots, A_{\min}, \cdots, A_{n-1}$$

已经位于最终的位置上　　最后的$n-i$个元素

$$A_0, \cdots, A_j \overset{?}{\leftrightarrow} A_{j+1}, \cdots, A_{n-i-1} \mid A_{n-i} \leqslant \cdots \leqslant A_{n-1}$$

已经位于最终的位置上

- 插入排序（减治法）

$$A[0] \leqslant \cdots \leqslant A[j] < A[j+1] \leqslant \cdots \leqslant A[i-1] \mid A[i] \cdots A[n-1]$$

小于等于$A[i]$　　大于$A[i]$

  - 插入排序的改进：希尔排序

```
算法  InsertionSort(A[0..n-1])
      //用插入排序对给定数组排序
      //输入：n 个可排序元素构成的一个数组 A[0..n-1]
      //输出：非降序排列的数组 A[0..n-1]
      for i ← 1 to n-1 do
          v ← A[i]
          j ← i-1
          while j ≥ 0 and A[j] > v do
              A[j+1] ← A[j]
              j ← j-1
          A[j+1] ← v
```

排序问题

- 合并排序、快速排序（分治法）

算法 Mergesort($A[0..n-1]$)
//递归调用 mergesort 来对数组 $A[0..n-1]$ 排序
//输入：一个可排序数组 $A[0..n-1]$
//输出：非降序排列的数组 $A[0..n-1]$
if $n > 1$
  copy $A[0..\lfloor n/2 \rfloor -1]$ **to** $B[0..\lfloor n/2 \rfloor -1]$
  copy $A[\lfloor n/2 \rfloor ..n-1]$ **to** $C[0..\lceil n/2 \rceil -1]$
  Mergesort($B[0..\lfloor n/2 \rfloor -1]$)
  Mergesort($C[0..\lceil n/2 \rceil -1]$)
  Merge(B,C,A) //参见下文 → 子问题划分很直接，主要工作在合并

算法 Quicksort($A[l..r]$)
//用 Quicksort 对子数组排序
//输入：数组 $A[0..n-1]$中的子数组 $A[l..r]$，由左右下标 l 和 r 定义
//输出：非降序排列的子数组 $A[l..r]$
if $l < r$
  $s \leftarrow$ Partition ($A[l..r]$) //s 是分裂位置 → 子问题解的合并很自然，主要工作在划分
  Quicksort($A[l..s-1]$)
  Quicksort($A[s+1..r]$)

- Lomuto划分
- Hoare划分

排序问题

- Lomuto划分

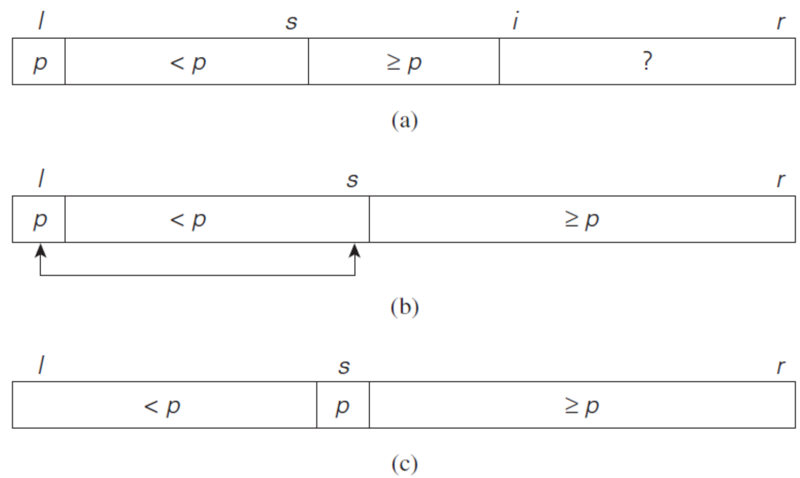


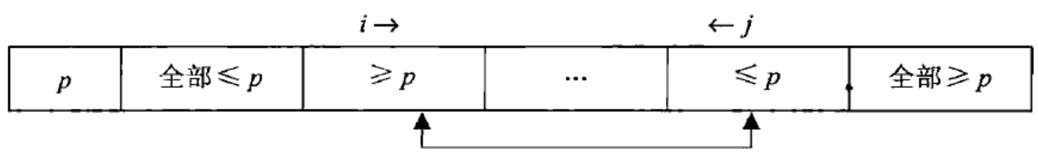
FIGURE 4.13 Illustration of the Lomuto partitioning.

算法 LomutoPartition($A[l..r]$)
//采用 Lomuto 算法，用第一个元素作为中轴对子数组进行划分
//输入：数组 $A[0..n-1]$ 的一个子数组 $A[l..r]$，它由左右两边的索引 l 和 r $(l \leqslant r)$定义
//输出：$A[l..r]$的划分和中轴的新位置
$p \leftarrow A[l]$
$s \leftarrow l$
for $i \leftarrow l + 1$ **to** r **do**
  if $A[i] < p$
    $s \leftarrow s + 1$; swap($A[s]$, $A[i]$)
swap($A[l]$, $A[s]$)
return s

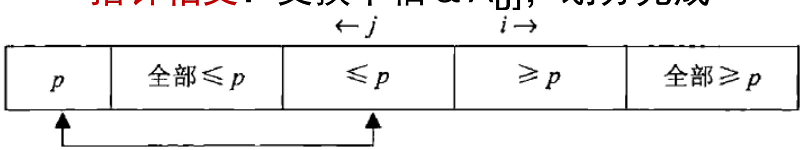
排序问题

- Hoare划分
  - 以数组的第一个元素为中轴，分别从数组的两端进行扫描
    - 指针i从左开始扫描，忽略小于中轴的元素，直到碰到大于或等于中轴的元素A[i]
    - 指针j从右开始扫描，忽略大于中轴的元素，直到碰到小于或等于中轴的元素A[j]

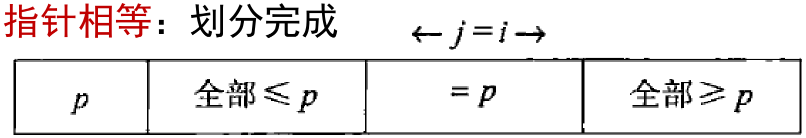
指针不相交：交换A[i] & A[j]，i++, j--，继续扫描



指针相交：交换中轴 & A[j]，划分完成

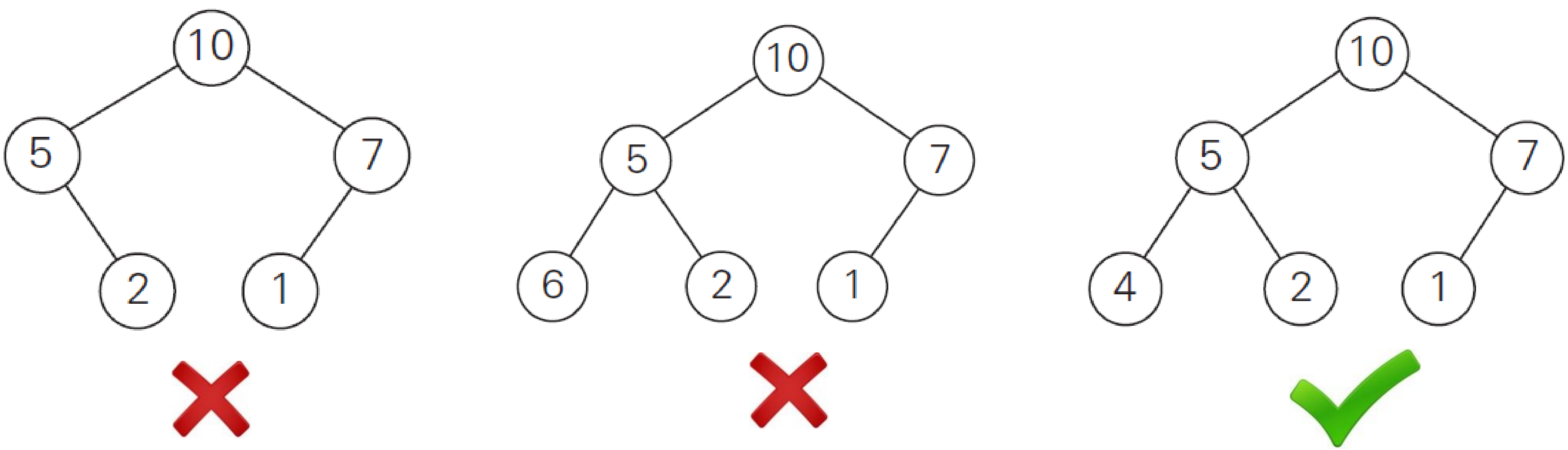


指针相等：划分完成



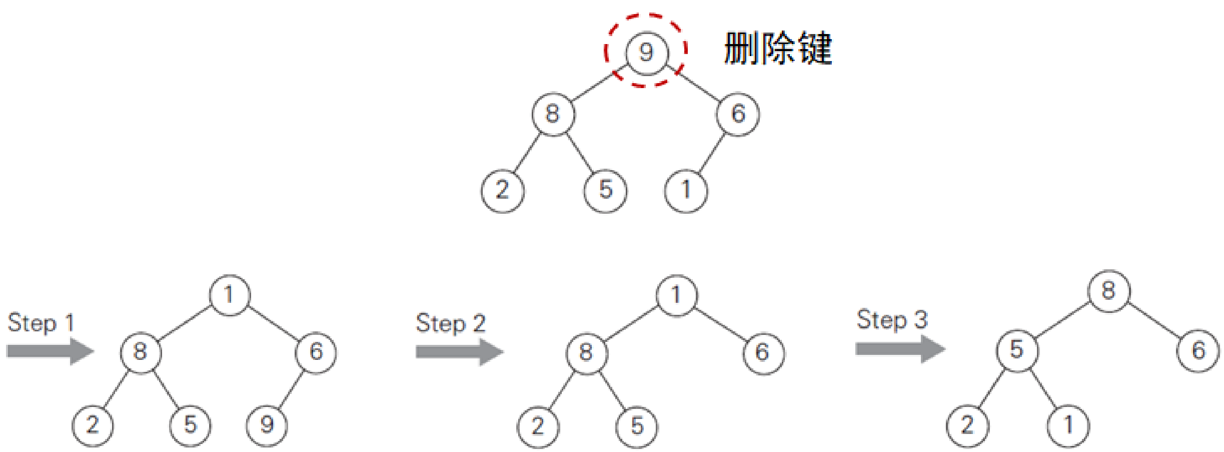
排序问题

- 堆是一棵满足以下条件的二叉树：
  ① 基本完备（essentially complete）：树的每一层都是满的，除了最后一层最右边的元素有可能缺位
  ② 父母优势（parental dominance）：每个节点键值都要大于或等于它子女的键



排序问题

- 堆排序　（变治法）
  - ①　第一步：为给定的数组构造一个堆（“改变表现”）
    - ■　自底向上，或自顶向下构造
  - ②　第二步：删除最大键，即对剩下的堆应用n-1次根删除操作
    - ■　根的键和堆的最后一个键做交换
    - ■　堆的规模减1
    - ■　采用自底向上堆构造算法对该树进行“堆化”



排序问题

- 计数排序（时空权衡）、拓展：桶排序；基排序

算法 DistributionCountingSort ($A[0..n-1], l, u$)
//用分布计数法，对来自于有限范围整数的一个数组进行排序
//输入：数组 $A[0..n-1]$，数组中的整数位于 l 和 u 之间($l \leqslant u$)
//输出：A 中元素构成的非降序数组 $S[0..n-1]$

for $j \leftarrow 0$ **to** $u-l$ **do** $D[j] \leftarrow 0$ // 初始化频率数组
for $i \leftarrow 0$ **to** $n-1$ **do** $D[A[i]-l] \leftarrow D[A[i]-l]+1$ // 计算频率值
for $j \leftarrow 1$ **to** $u-l$ **do** $D[j] \leftarrow D[j-1]+D[j]$ // 重用于分布
for $i \leftarrow n-1$ **downto** 0 **do**
    $j \leftarrow A[i]-l$
    $S[D[j]-1] \leftarrow A[i]$
    $D[j] \leftarrow D[j]-1$
return S

基于分布反向填充目标数组

排序问题

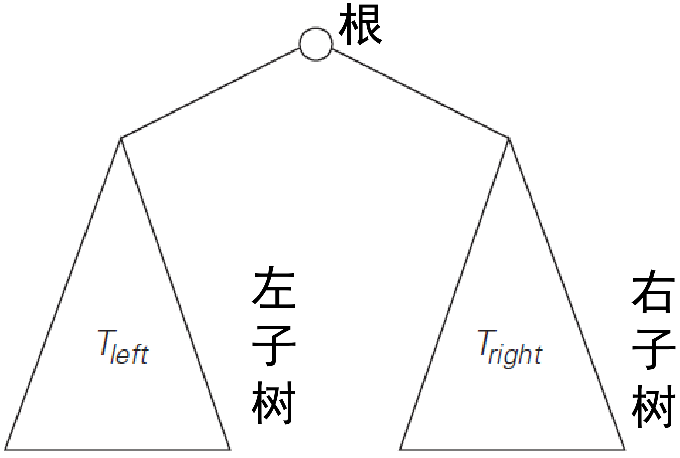
比较型排序

非比较型排序

排序方法	时间复杂度（平均）	时间复杂度（最坏）	时间复杂度（最好）	空间复杂度	稳定性
插入排序	$O(n^2)$	$O(n^2)$	$O(n)$	$O(1)$	稳定
希尔排序	$O(n^{1.3})$	$O(n^2)$	$O(n)$	$O(1)$	不稳定
选择排序	$O(n^2)$	$O(n^2)$	$O(n^2)$	$O(1)$	不稳定
堆排序	$O(nlog_2n)$	$O(nlog_2n)$	$O(nlog_2n)$	$O(1)$	不稳定
冒泡排序	$O(n^2)$	$O(n^2)$	$O(n)$	$O(1)$	稳定
快速排序	$O(nlog_2n)$	$O(n^2)$	$O(nlog_2n)$	$O(nlog_2n)$	不稳定
归并排序	$O(nlog_2n)$	$O(nlog_2n)$	$O(nlog_2n)$	$O(n)$	稳定
计数排序	$O(n+k)$	$O(n+k)$	$O(n+k)$	$O(n+k)$	稳定
桶排序	$O(n+k)$	$O(n^2)$	$O(n)$	$O(n+k)$	稳定
基数排序	$O(n*k)$	$O(n*k)$	$O(n*k)$	$O(n+k)$	稳定

查找问题

- 基于数组的键值查找
  - 顺序查找（蛮力）、折半查找（减治）、预排序（变治）
- 基于树的查找
  - 二叉树、节点遍历方式：
    - 前序(preorder)：根->左子树->右子树
    - 中序(inorder)：左子树->根->右子树
    - 后序(postorder)：左子树->右子树->根



  - 最优二叉查找树的构造

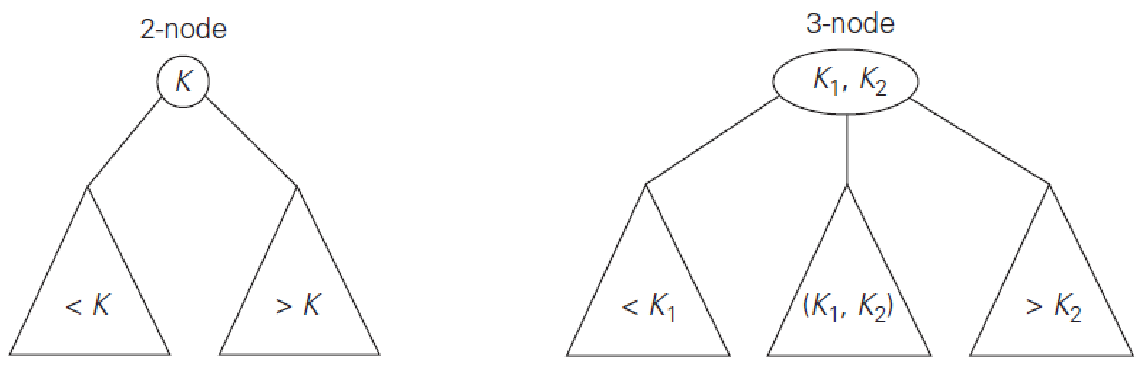
问题描述：设$a_1, a_2, \ldots, a_n$是从小到大排列的互不相等的键，$p_1, p_2, \ldots, p_n$是它们的查找频率，则构造一棵最优二叉树，使在树中成功查找键的平均查找次数最少

（动态规划）

  - 哈夫曼树的构造

查找问题

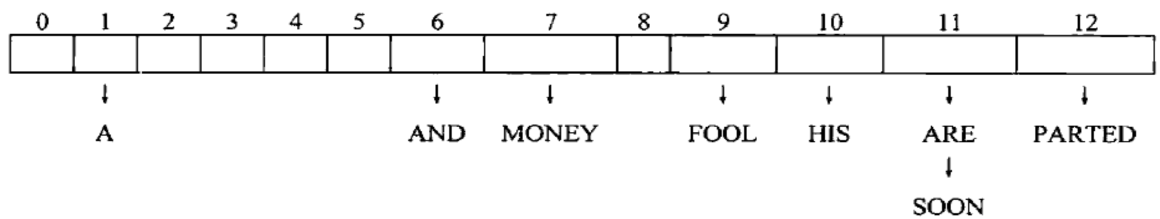
- 基于树的查找
  - 平衡查找树
    - AVL树：通过左单转、右单转、左右双转或右左双转来保持树的平衡性（每个节点的平衡因子为0，+1或-1）
    - 2-3树：包含两种类型节点的树
      - 2节点：一个键K和两个子女
      - 3节点：两个有序的键K_1和K_2和3个子女



查找问题

- 基于散列表的查找（时空权衡）：采用预先定义的函数对键进行计算，将键分布在散列表中
  - 开散列：分配到同一散列表单元格中的键用链表表示

键	A	FOOL	AND	HIS	MONEY	ARE	SOON	PARTED
散列地址	1	9	6	10	7	11	11	12



  - 闭散列：所有的键都存储在散列表中，采用“线性探测”解决碰撞

0	1	2	3	4	5	6	7	8	9	10	11	12
	A											
	A								FOOL			
	A					AND			FOOL			
	A					AND			FOOL	HIS		
	A					AND	MONEY		FOOL	HIS		
	A					AND	MONEY		FOOL	HIS	ARE	
	A					AND	MONEY		FOOL	HIS	ARE	SOON
PARTED	A					AND	MONEY		FOOL	HIS	ARE	SOON

字符串匹配问题

- 蛮力法：模式跟文本按从左到右的顺序做比较，若不匹配，则将模式右移一个字符，再做比较
- Horspool算法：给定模式P，及匹配查找时可能碰到任意的字符c，预先计算出模式的右移距离t(c)，并把它们存储在索引表中

$$t(c)=\begin{cases}\text{模式的长度}m(\text{如果}c\text{不包含在模式的前}m-1\text{个字符中})\\ \text{模式前}m-1\text{个字符中最右边的}c\text{到模式最后一个字符的距离(在其他情况下)}\end{cases}$$

- Boyer-Moore算法：Horspool算法的增强版
  - 坏符号
  - 好后缀

```
B E S S _ K N E W _ A B O U T _ B A O B A B S
B A O B A B
d1 = t1(K) - 0 = 6    B A O B A B
                      d1 = t1(_) - 2 = 4    B A O B A B
                      d2 = 5                d1 = t1(_) - 1 = 5
                      d = max{4, 5} = 5     d2 = 2
                                            d = max{5, 2} = 5
                                                          B A O B A B
```

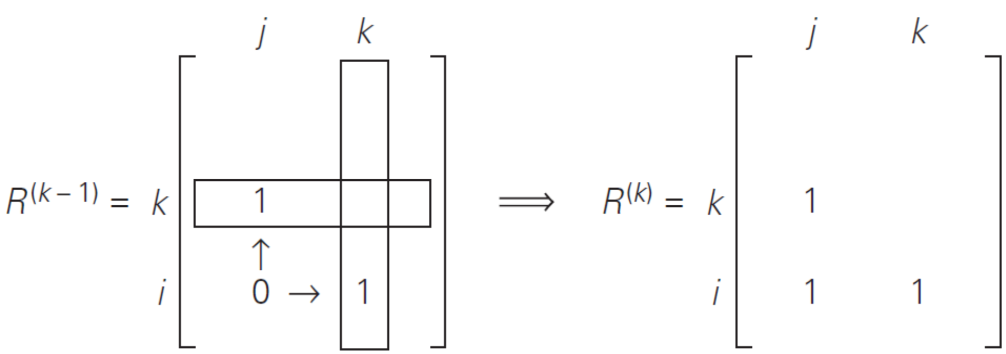
图问题

- 图的遍历算法（蛮力法）
  - 深度优先查找：适合用栈来实现
  - 广度优先查找：适合用队列来实现

- Warshall算法(动态规划)：计算有向图的传递闭包
  - 通过一系列n阶的布尔矩阵构造传递闭包

$$\boldsymbol{R}^{(0)}, \dots, \boldsymbol{R}^{(k-1)}, \boldsymbol{R}^{(k)}, \dots, \boldsymbol{R}^{(n)}$$

    - 矩阵$\boldsymbol{R}^{(0)}$为邻接矩阵
    - 矩阵$\boldsymbol{R}^{(k-1)}$构造$\boldsymbol{R}^{(k)}$

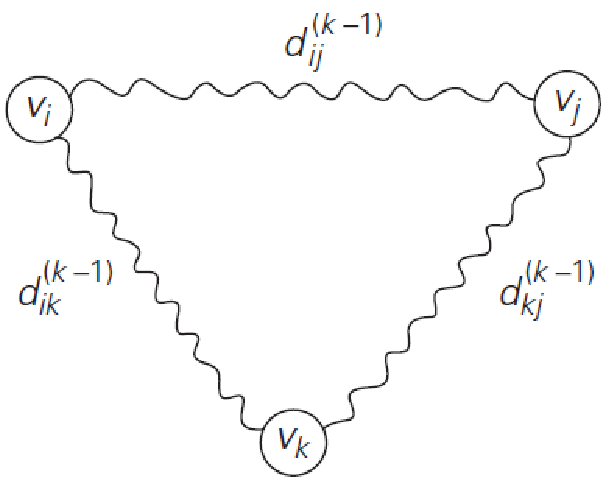


图问题

- Floyd算法(动态规划)：计算有向图的距离矩阵
  - 通过一系列n阶矩阵来构造距离矩阵

$$\boldsymbol{D}^{(0)},\ldots,\boldsymbol{D}^{(k-1)},\boldsymbol{D}^{(k)},\ldots,\boldsymbol{D}^{(n)}$$

    - 矩阵$\boldsymbol{D}^{(0)}$为权重矩阵
    - 从矩阵$\boldsymbol{D}^{(k-1)}$来构造$\boldsymbol{D}^{(k)}$



$$\boxed{d_{ij}^{(k)}=\min\{d_{ij}^{(k-1)},\ d_{ik}^{(k-1)}+d_{kj}^{(k-1)}\}\quad \text{for } k\ge 1,\quad d_{ij}^{(0)}=w_{ij}.}$$

图问题

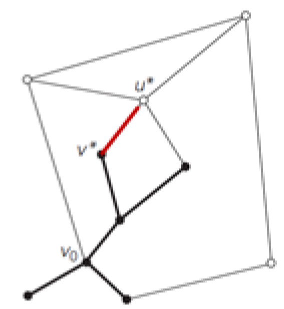
- Dijkstra算法（贪婪技术）：计算加权图中单源点最短路径问题

基本思想：由近而远，依次寻找离源点最近的顶点

- i=1次迭代：根据权重矩阵，找到距离源点最短路径长度及相应的顶点；由源点、该顶点及源点到该顶点的路径构成一棵子树T_1
- i$\in [2:n]$次迭代：从与T_{i-1}的顶点相邻的顶点集合（边缘顶点）中找到与源点距离最近的顶点，并将该顶点（如图中的u^*）加入子树中，得T_i
  - 确定加入顶点u^*后，还需以下两个操作
    - 把u^*从边缘顶点集合移到树顶点
    - 更新每个边缘顶点u的标记，如果

$$d_{u^*} + w(u^*, u) < d_u$$

则u的标记更新为$u(u^*, d_{u^*} + w(u^*, u))$



- Prim/Kruskal算法（贪婪技术）：最小生成树的构造

图问题

- Floyd算法 vs. Dijkstra算法
  - Floyd算法
    - 任意两顶点间的最短路径
    - 适用于带负权值的边，但不能带负权值回路
  - Dijkstra算法
    - 从源点到其它顶点的最短路径
    - 不适用于带负权值边的有向图

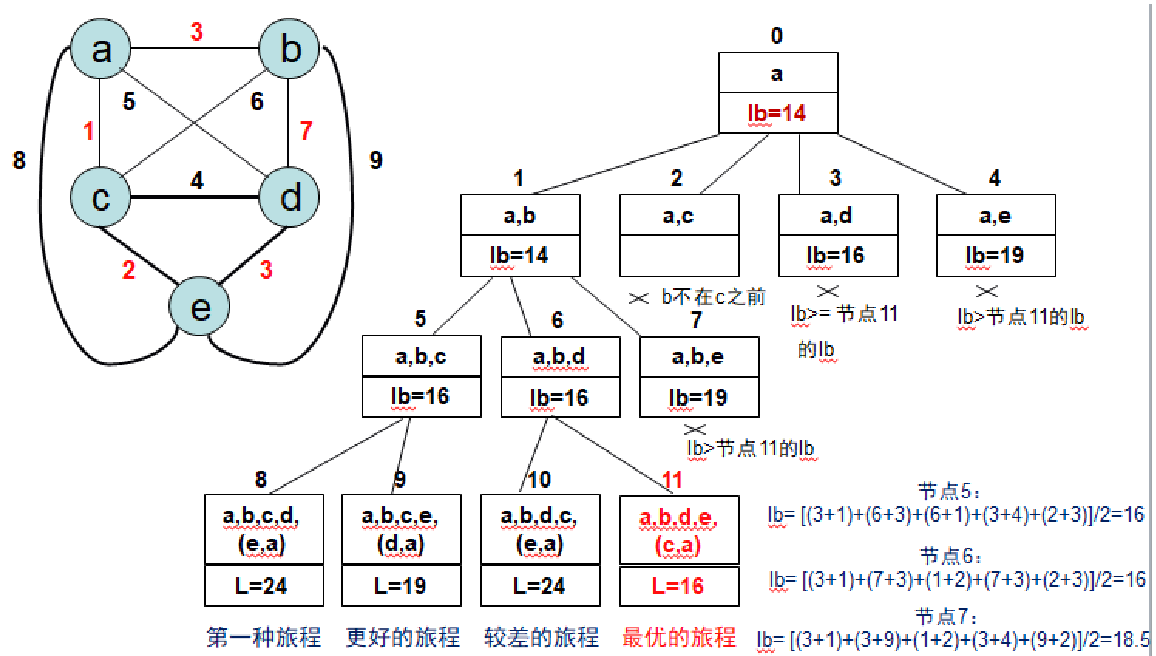
图问题

- 最短增益路径算法(迭代改进)：解决最大流量问题
  - 顶点标记方式：用两个记号来标记一个新的顶点
    - 第一个标记指出从源点到被标记顶点还可增加的流量
    - 第二个标记记录其前序顶点，并加上“+”或“–”号来表示该顶点到本顶点是前向边还是后向边
  - 用队列来存取标记的顶点
    - 建立一个空队列，用于存放每次标记的节点。第一次将源点放入此队列中，以后取节点操作都是对队列进行
    - 前向边扫描：每次都先取队列的第一个节点，然后查看此节点指向哪些节点，若被指向的节点未被标记，且相应的正向边未使用容量为正，则将被指向的节点进行标记后放入队列。循环做此步直至当前节点所指向的点全部标记且入队列
    - 后向边扫描：当前节点的前向边遍历结束后，进行反向边遍历，将有正流量的反向边节点进行标记后放入队列
    - 当前节点前向、后向遍历结束后，检查汇点是否被标记，若没被标记则继续取队列的下一个节点；若被标记则表示一条增广路径已被找到，更新各边的流量，然后将所有节点的标记都初始化、将队列清空，将源点入队，开始下一条增益路径的寻找

网络中的最大流量值等于它的最小割的容量

组合问题

- 旅行商问题
  - 穷举查找（蛮力法）
  - 分支界限法
  - 图神经网络



- 背包问题
  - 蛮力法、动态规划、线性规划、分支界限法

思考题(1)

- 问题描述：给定两个字符串s和t，它们只包含小写字母。字符串t由字符串s随机重排，然后在随机位置添加一个字母得到。请设计一个高效的算法找出在t中被添加的字母
- 例如：
  - 输入s = "abcd"， t = "acebd"，输出"e"
  - 输入s = "a"， t = "aa"，输出"a “

- 思路：
  ① 分别统计s和t中出现的字母及频次，t中要么出现新字母，要么某个字母的频次比s中相应字母多1
  ② S=[s t]，然后进行“字母的异或”运算，输出结果即为被添加的字母

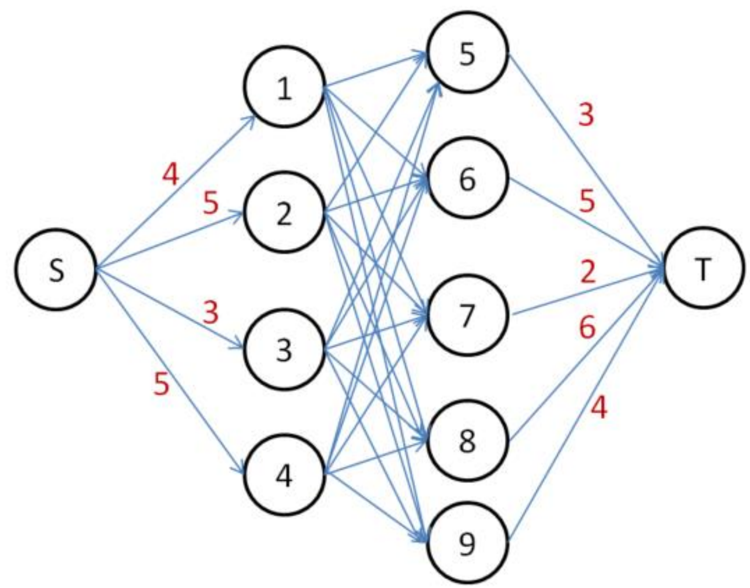
思考题（2）

- 问题描述:假设有来自n家不同机构的代表参加一个国际会议。每家机构的代表人数分别为$r_i(i = 1,2, \dots , n)$。会议餐厅共有m张餐桌，每张餐桌可容纳$c_i\ (i = 1,2, \dots , m)$个代表就餐。为了使代表们充分交流，希望从同一机构来的代表不在同一个餐桌就餐
- 请设计一个高效的算法(用伪代码或图的形式描述)，用于求解满足要求的代表就餐方案

思考题（2）

- 将上述问题转化为一个网络最大流量问题
  - 假设每个机构为X集合中的顶点，每张餐桌为Y集合中的顶点
  - 引入虚拟的源点S及汇点T
  - 源点S到X中每个顶点的边的权重为相应机构派出的人数
  - X中的每个顶点与Y中的每个顶点都有权重为1的边
  - Y中每个顶点到汇点T的边的权重为相应餐桌可容纳的人数

举例：4家机构，代表人数分别为4, 5, 3, 5；5张餐桌，可容纳人数分别为3, 5, 2, 6, 4



- 若网络的最大流量=代表人数总数，则该问题有解；否则，无解

不忘初心

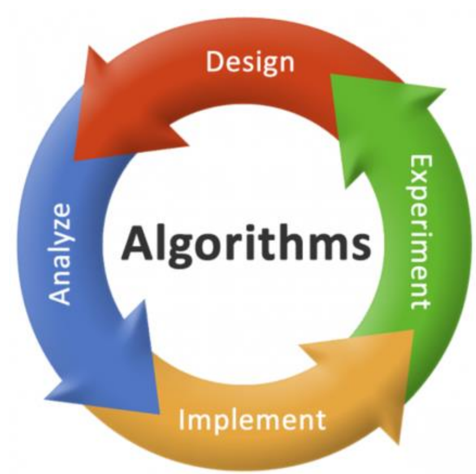
- 重视算法思想，触类旁通
- 重视编程实践，熟能生巧

You Are Supposed to Be:

Cool minded

Self motivated

Programming addicted



大数据+大模型+强算法

感谢大家对本课程的支持！
祝期末考试顺利！