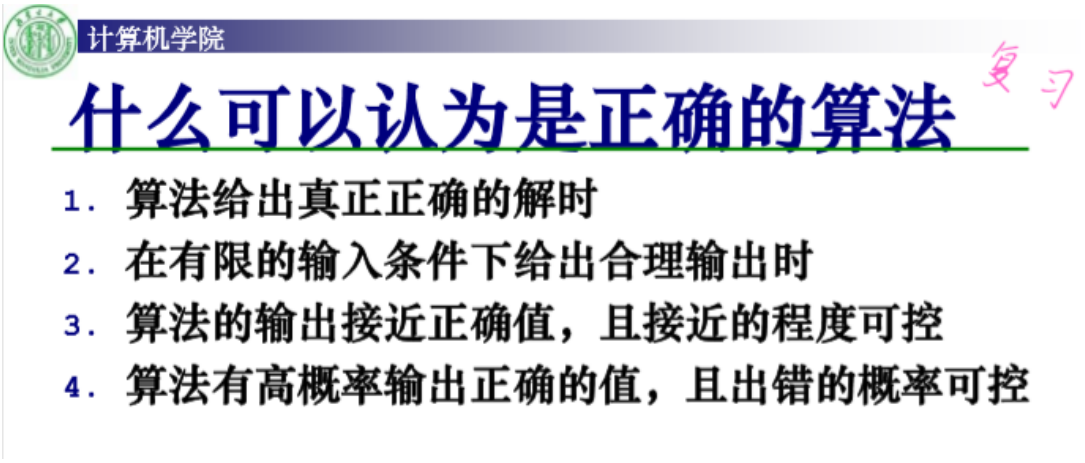
一、


计算机学院
复习
什么可以认为是正确的算法
1. 算法给出真正正确的解时
2. 在有限的输入条件下给出合理输出时
3. 算法的输出接近正确值，且接近的程度可控
4. 算法有高概率输出正确的值，且出错的概率可控

二、


计算机学院
复习
讨论：从特殊到一般
■深度学习
□把大模型
□用大数据
□做大规模训练
Specialization
特殊/特化/演绎
Generalization
一般/泛化/归纳

单源最短路

- 拓扑排序最短路 $\theta(n+m)$
  - 有向无环图
- 迪杰斯特拉算法 $O(n\lg n+m)$、$O(m\lg n)$、$O(n^2)$
  - 有向正权图
- 贝尔曼福特算法 $\theta(nm)$
  - 有向无无权图

Specialization
特殊/特化/演绎

Generalization
一般/泛化/归纳

三、

算法工具箱

复习

- 正确性
  - 证明：
    - 正确性：循环/递归不变式
    - 不正确：反例
- 复杂度
  - 渐进复杂度：$\Omega\ \theta\ O$
  - P问题-NP问题
- 设计
  - 递归
  - 分而治之
  - 动态规划、贪心
  - 编码-解码镜像操作
  - 近似算法

正确性

计算机学院

循环不变式(Loop invariants)

- 循环不变式是一个命题
- 命题是一个真假判断
- 在循环的每次迭代开始时，循环不变量永为真

- 需要证明以下三条：

1. 初始条件，Initialization，
   第一次迭代之前，循环不变式为真
2. 维持条件，Maintenance，
   如果在迭代开始时是真的，那么在下一次迭代开始前也是真的
3. 终止条件，Termination，
   循环必须终止，终止时，循环不变式对问题求解是有用的

计算机学院

循环不变式(Loop invariants)

- Function better_linear_search(A, n, x)
- 输入输出同linear_search

1. for i=0:n-1
   A. if A[i]==x then return i
2. return NOT_FOUND

- 循环不变式
  - 如果x在数组A中，那么它一定出现在A[i…n-1]这个子数组中。
- 证明：

1. 初始条件：当i=0时，显然成立。
2. 维持条件：若第i次迭代成立，且未从1A返回，A[i]!=x,则如果x在数组A中，那么它一定出现在A[i+1…n-1]这个子数组中，成立。
3. 终止条件：循环一定终止，若未从1A返回，必然从2返回，此时循环不变式成立，子数组为空，由循环不变式的逆否命题可知，x不在数组A中，返回NOT_FOUND是正确的。QED

二分查找：正确性

- Function binary_search(A, n, x)
- 输入输出同上

1. p = 0, r = n-1
2. while p<r
   A. q = floor((p+r)/2)
   B. if A[q]==x then return q
   C. if A[q]>x then r=q-1
   D. else p = q+1
3. return NOT_FOUND

- 循环不变式
  - 如果x在数组A中，那么它一定出现在A[p…r]这个子数组中。

- 证明：

1. 初始条件：当p=0，r=n-1时A[p…r]是原数组，显然成立。
2. 维持条件：2C和2D的结果是正确的。
3. 终止条件：p增大，r减小，总会有p>r，循环一定终止。循环结束时，子数组为空，由循环不变式的逆否命题可知，x不在数组A中，返回NOT_FOUND是正确的。QED

选择排序：正确性

- 循环不变式
  - 外循环：
    - 子数组A[0…i-1]中有数组A中的最小的i个数，并且它是排好序的
  - 内循环：
    - A[smallest]是A[i…j-1]的最小值

插入排序：正确性

- 循环不变式
  - 外循环：
    - 子数组A[0…i-1]中有数组A中的前i个数，并且它是排好序的

归并排序：分析

- 排列n个元素需要$T(n)$
- $T(n) = 2 \cdot T\left(\frac{n}{2}\right) + \theta(n) = \theta(n\lg n)$

- 空间复杂度 $\theta(n)$

快速排序：运行时间分析

- 排列n个元素需要$T(n)$
- 分
  - q = partition(A, p, r)　　$\theta(n)$
- 治
  - 排 A[p…q-1], A[q+1…r]　　?
- 合
  - 什么也不做　　0

◆ 最坏情形 (q = r)
len(A[p…q-1]) = n-1
len(A[q+1…r]) = 0　　$T(n) = T(n-1) + \theta(n) = \theta(n^2)$

◆ 最好情形 (q = (p+r)/2)
len(A[p…q-1]) ≈ n/2
len(A[q+1…r]) ≈ n/2　　$T(n) = T\left(\frac{n}{2}\right) + \theta(n) = \theta(n\lg n)$

记数排序：运行时间分析

- Function counting_sort(A, n, m)
- Input: A要排序的数组，数组里只有0到m-1的整数
  n 数组A的长度
  m 数组A中不同元素的个数
- Output: B,按升序排列的数组A

1.	equal = count_key_equal(A, n, m)	$\theta(m+n)$
2.	less = count_key_less(equal, m)	$\theta(m)$
3.	B = rearrange(A, less, n, m)	$\theta(m+n)$
4.	Return B	
		$\theta(n)$

基数排序：运行时间分析

- 排序键有d位 $\theta(d)$
- 每一位的范围0...m-1 $\theta(m+n)$
- $\theta\big(d\cdot(m+n)\big)$
- $\theta(n)$

拓扑排序：正确性

- 循环不变式
  - 步骤5的循环开始前
    已输出的节点不依赖于任意未输出节点

拓扑排序运行时间 $\theta(m+n)$

- Function topological_sort(G)
- Input: G 包含n个节点的有向无环图
- Output: 线性顺序表，其中若G中包含边(u,v)，在顺序表中u出现在v之前

1.	设输出为空		$\theta(1)$
2.	设in_degree[0…n]为全0		$\theta(n)$
3.	for each 节点u		$\theta(m+n)$
	A. for each 与u相邻的节点v		
	I. in_degree[v] += 1		
4.	将所有in_degree[u]==0的节点u添加到列表next中		$O(n)$
5.	while next不空	$\theta(n)$	$\theta(m+n)$
	A. 从next删除一个节点u	$\theta(1)$	
	B. 输出u	$\theta(1)$	
	C. for each 与u相邻的节点v	$\theta(m)$	
	I. in_degree[v] -= 1		
	II. if in_degree[v]==0 then 将节点v添加到列表next中		
6.	返回		

单源最短路 时间 $\theta(m+n)$

- Function dag_shortest_path(G, s)
- Input: G 图，s 源节点
- Result: 最短路保存在shortest和pred中

1.	l = topological_sort(G)	$\theta(m+n)$
2.	prepare()	$\theta(n)$
3.	for each 节点u in l	$\theta(m+n)$
	A. for each 与u相邻的节点v	
	I. relax(u,v)	

单源最短路径的正确性

证明：

假设存在从 s 点到 v 点的最短路径 p，则 p 的权重比其他任何 s 点到 v 点的路径权重都小。特别地，路径 p 没有比从源 s 到顶点 u 采用短路径的特殊路径更重，然后取得边缘（u，v）

迪杰斯特拉算法：正确性

- 循环不变式
  - 在第3步的每次迭代开始时，
    如果v不在集合Q中，那么
    shortest[v]是从s到v的最短路的权重
- 证明
  - 初始条件：$\forall v \in Q$ 循环不变式成立
  - 维持条件：设不在Q中的节点v的最短路已确定，集合Q中的节点最短路只能因集合Q中的其他节点而改变。此时选择循shortest最小的，它已再无变小的可能，它的最短路确定，后续调整保证下一次迭代满足此条件。
  - 结束条件：集合Q一直减小，最终为空，循环结束。

迪杰斯特拉算法：运行时间

- Function dijkstra(G, s)
- Input: G 图，s 源节点
- Result: 最短路保存在shortest和pred中

1. prepare() $\theta(n)$
2. q = 包含所有节点的集合
3. while Q 不空 $\theta(n)$
   A. u = 在Q中shortest[u]最小的节点
   B. for each 与u相邻的节点v $\theta(m)$
      I. relax(u,v) $\theta(1)$

拓扑排序

- 入度（in degree）
  - 节点的入度是进入该节点的边数
- 出度（out degree）
  - 节点的出度是离开该节点的边数
- 定理：有向无环图中至少有一个入度为0的节点，至少有一个出度为0的节点
  - 证明：随机选择一个节点，沿边跳转，最多经过n-1步将结束到一个出度为0的节点，否则原图有环。
    将原图的边方向取反，同理，随机选择一个节点，沿边跳转，最多经过n-1步，将结束到一个出度为0的节点，即原图中入度为0的节点，否则原图有环。 ■

贝尔曼福特算法：运行时间

- Function bellman_ford (G, s)
- Input: G 图，s 源节点
- Result: 最短路保存在shortest和pred中

1. Prepare()
2. for i=1 to n-1 $\theta(n)$ $\theta(nm)$
   A. for each 边(u,v) $\theta(m)$
      I. relax(u,v)

复杂度

渐进表示：θ记号

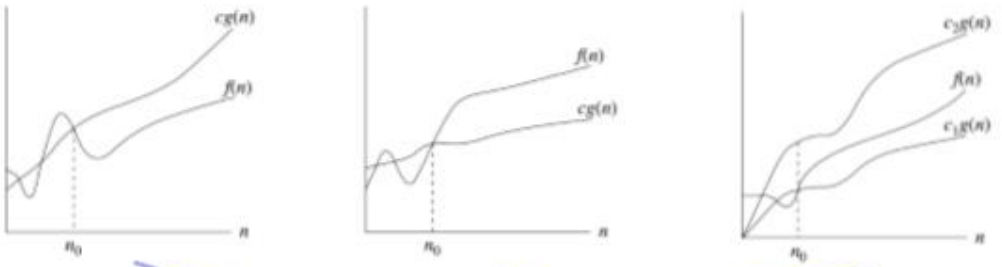
- 对于两个函数$f(n)$和$g(n)$，若对足够大的$n > n_0$，总有
$$c_1 \cdot g(n) < f(n) < c_2 \cdot g(n)$$
其中$c_1 > 0$，$c_2 > 0$是常数，则称$f(n) \in \theta(g(n))$

- 例：$\frac{n^2}{4} + 100 \cdot n + 50 \in \theta(n^2)$
- $\frac{n^2}{4} + 100 \cdot n + 50 = \theta(n^2)$

- 对于两个函数$f(n)$和$g(n)$，若对足够大的$n > n_0$，总有
$$c \cdot g(n) < f(n)$$
其中$c > 0$是常数，则称$f(n) \in \Omega(g(n))$
- Ω 读作 Omega

- 上界$O(\cdot)$，下界$\Omega(\cdot)$，既是上界又是下界$\theta(\cdot)$

- 定理：$f(n) \in \theta(g(n)) \Leftrightarrow f(n) \in (O(g(n)) \cap \Omega(g(n)))$



$\theta(g(n))$　　$O(g(n))$　　$\Omega(g(n))$

二分查找：运行时间分析

- 输入大小：n，数组A的大小
- 时间复杂度：$O(\lg n)$

- 最坏情形：$\theta(\lg n)$
- 最好情形：$\theta(1)$

选择排序：运行时间 $\theta(n^2)$

- 证明：
  - 运行时间是$O(n^2)$
    - 每次迭代中内层循环最多做n-1次，所以内层循环是$O(n)$
    - 外层循环做了n-1次，所以外层循环是$O(n)$
    - 合起来是$O(n^2)$
  - 运行时间是$\Omega(n^2)$
    - 每次迭代中外层循环最少做$\frac{n}{2}$次
    - 在前n/2次外层循环中，内层循环最少要做$\frac{n}{2}$次
    - $\frac{n}{2}\cdot\frac{n}{2}=\frac{n^2}{4}=\Omega(n^2)$
  - 所以运行时间是$\theta(n^2)$

插入排序：运行时间

- 最好情形
  - 每次都不移动⇒A[i-1]<=A[i]⇒A从小到大排列⇒ $\theta(n)$
- 最坏情形
  - 每次都移动最大次数⇒A[i-1]>A[i]⇒A从大到小排列
  - 移动次数 $1+2+3+\cdots+(n-2)+(n-1)=\frac{n^2-n}{2}=\theta(n^2)$
- 平均情形
  - 设A的顺序是随机的，比A[i]小的元素有50%的几率出现在A[i]的前面
  - 移动次数减半 $\frac{1+2+3+\cdots+(n-2)+(n-1)}{2}=\frac{n^2-n}{4}=\theta(n^2)$
- 若A中仅有k个元素的顺序不对
  - 每次移动的最大距离不超过k
  - 总的移动次数 $(n-1)\cdot k=\theta(kn)=\theta(n)$　k是常数

P-NP

P 问题：多项式时间内可以解决的问题

NP 问题：在多项式时间内可以被证明的问题

NP 完全问题：“不易解决”

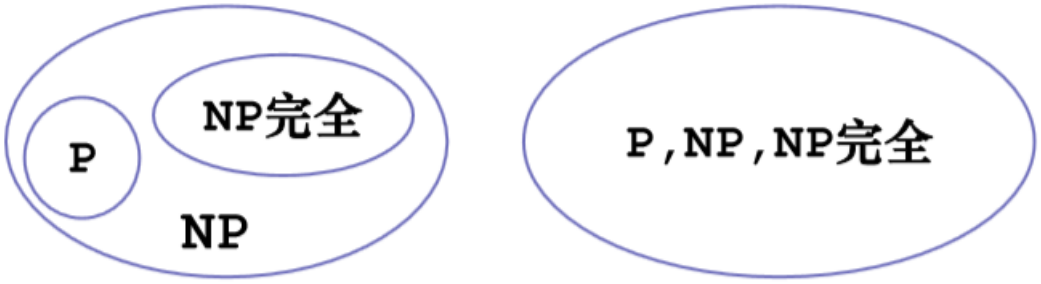
- 时间复杂度<$\theta(n^c)$的算法
- 多项式时间算法（P）

- NP问题
- nondeterministic polynomial time
- 不确定是否有多项式时间算法的问题

- 最短路问题是P问题
- 贝尔曼-福特算法$\theta(nm)$，$\theta(n^3)$
- 最长路问题是NP问题

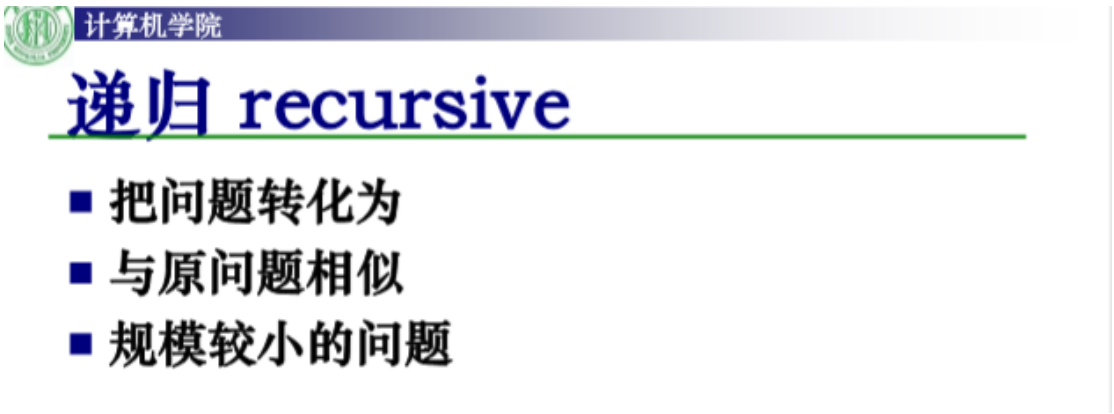
开放性问题

- NP=P?
- NP完全问题



设计

递归



计算机学院

循环⇔递归(recursive)

- Function recursive_linear_search(A, n, i, x)
- Input: A 要查找的数组
  n A的长度
  i 递归计数器
  x 要查找的值
- Output: 如果x在A中，返回使A[i]==x的索引i,否则返回NOT_FOUND

```
1. If i>n-1 then return NOT_FOUND
2. if A[i]==x then return i
3. return recursive_linear_search(A, n, i+1, x)
```

- 初始调用 recursive_linear_search(A, n, 0, x)

分而治之

思想：将原问题分解为几个规模较小但与原问题的类似的子问题，递归地求解这些子问题，然后再合并这些子问题的解来建立原问题的解。

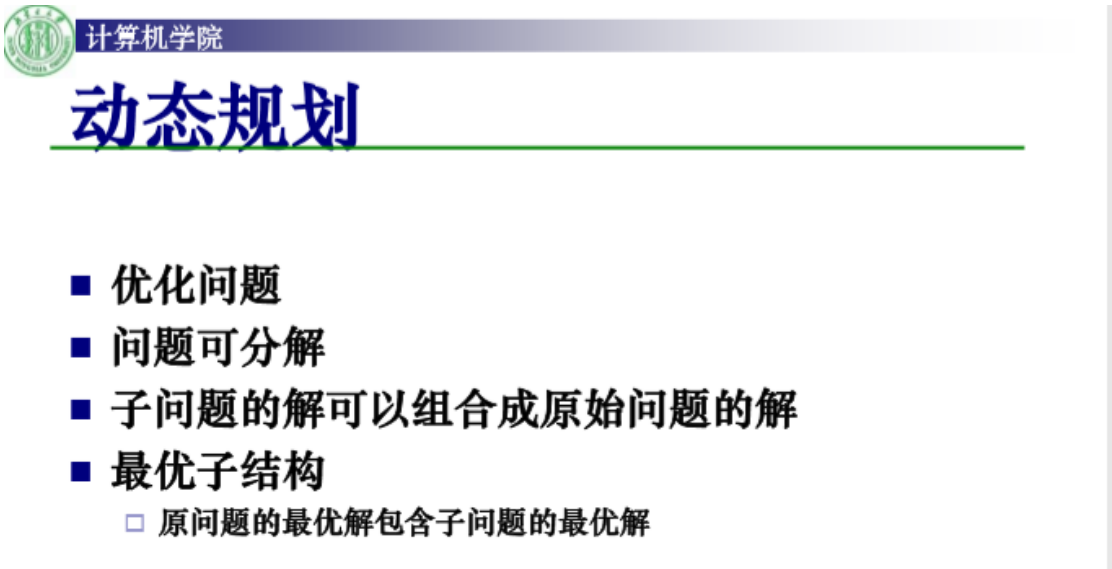
计算机学院

分而治之—分治

- 分
  - 分解成子问题（与原问题相似但规模较小）
- 治
  - 解决每个问题（使用递归方法）
- 合
  - 把子问题的解合并

动态规划 自底向上

步骤：
1.刻画一个最优解的结构特征
2.递归地定义最优解的值
3.计算最优解的值，通常采用自底向上的方法
4.利用计算出的信息构造一个最优解
备忘机制，避免重复求解相同的子问题



最大公共子序列

设 $X = \langle x_1, x_2, \dots, x_m \rangle$ 和 $Y = \langle y_1, y_2, \dots, y_n \rangle$为两个序列，并设 $Z = \langle z_1, z_2, \dots, z_k \rangle$为 X 和 Y 的任意一个最长公共子序列。则我们可以得出以下的结论：

1) 若 $X_m = Y_n$，则 $Z_k = X_m = Y_n$，且 z_{k-1} 是 X_{m-1} 和 Y_{n-1} 的最长公共子序列。

2) 若 $X_m \neq Y_n$，$Z_k \neq X_m$ 则 Z 是 X_{m-1} 和 Y_n 的最长公共子序列。

3) 若 $X_m \neq Y_n$，$Z_k \neq Y_n$ 则 Z 是 X_m 和 Y_{n-1} 的最长公共子序列。

其中 $X_{m-1} = \langle x_1, x_2, \cdots, x_{m-1} \rangle$，$Y_{n-1} = \langle y_1, y_2, \cdots, y_{n-1} \rangle$，$Z_{k-1} = \langle z_1, z_2, \cdots, z_{k-1} \rangle$。

两个序列的最长公共子序列包含了这两个序列的前缀的最长公共子序列。因此，最长公共子序列问题具有最优子结构性质。

贪心

通过做出一系列来求出问题的最优解。通过局部最优解来构造全局最优解。**自顶向下**

步骤：

1.确定问题的最优子结构

2.设计一个递归算法

3.证明如果我们做出一个贪心选择，则只剩一个子问题

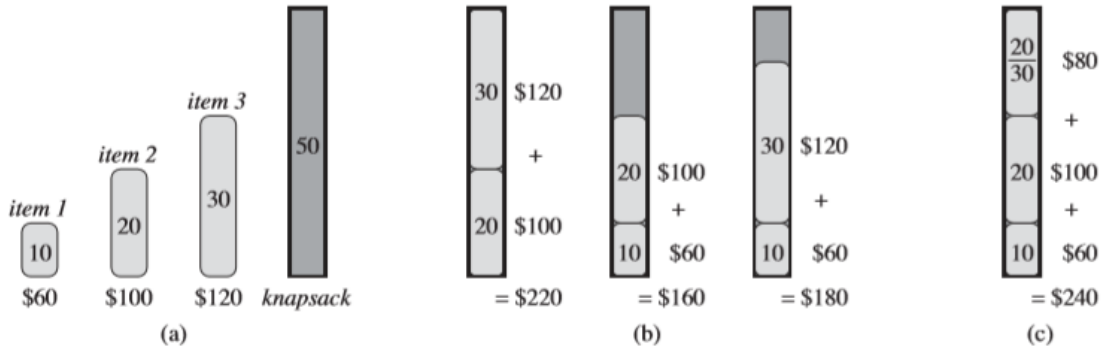
4.证明贪心选择总是安全的

5.设计一个递归算法的贪心策略

6.将递归算法转换为迭代算法。

贪心算法对 0-1 背包问题无效，产生空闲空间，降低了有效价值。

An example showing that the greedy strategy does not work for the 0-1 knapsack problem：

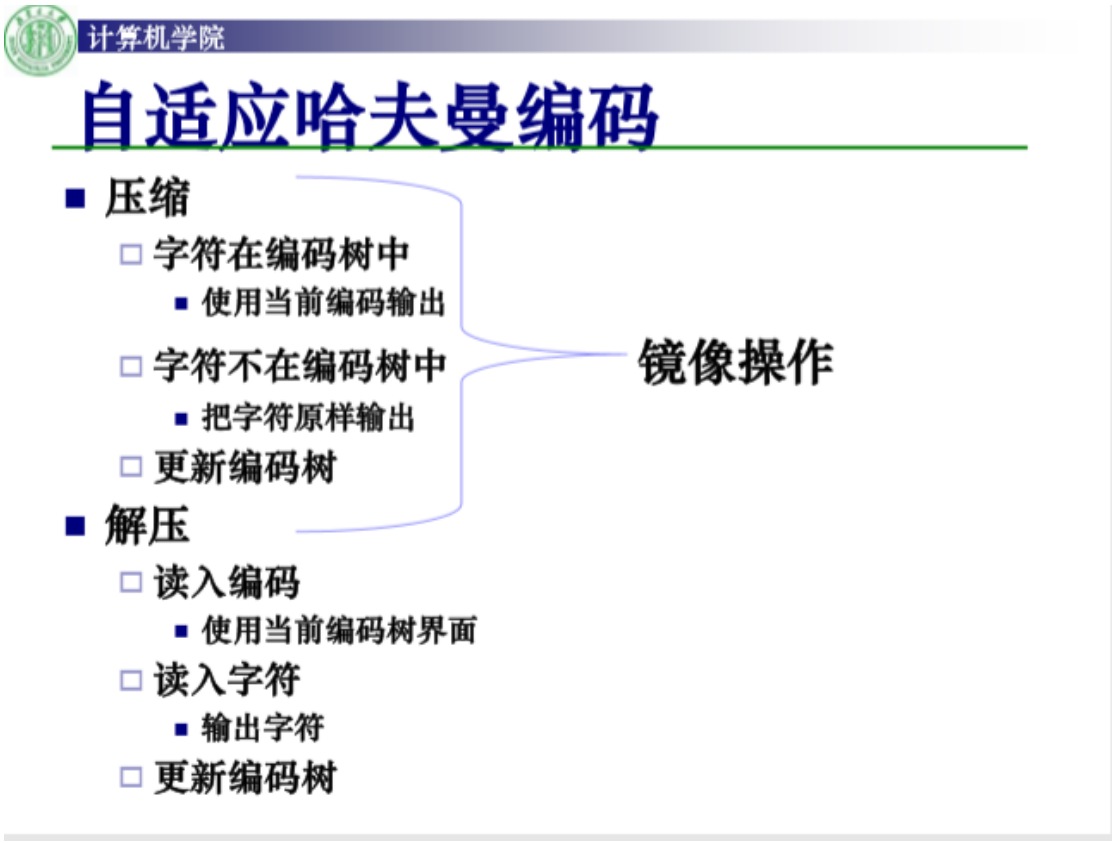


(a) The thief must select a subset of the three items shown whose weight must not exceed 50 pounds.

(b) The optimal subset includes items 2 and 3. Any solution with item 1 is suboptimal, even though item 1 has the greatest value per pound.

(c) For the fractional knapsack problem, taking the items in order of greatest value per pound yields an optimal solution.

镜像操作



哈夫曼编码

第一遍 统计字符频率，构建编码树
第二遍 根据编码树进行编码

自适应哈夫曼编码

构造动态哈夫曼编码树的过程，需要遵循两条重要规则：
1.权重值大的节点，节点编号也较大
2.父节点的节点编号总是大于子节点的节点编号
这两条也称为兄弟属性
FGK 算法：在大部分情况会浪费许多空间
Vitter 算法
为了维持 Vitter 的不变性，即权重 w 的所有叶子都在权重 w 的所有内部节点之前（在隐式编号中）
注意：
同一级，叶节点在中间节点之前

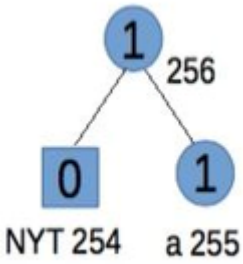
叶节点：先移动，后更新
中间节点：先更新，后移动

Example

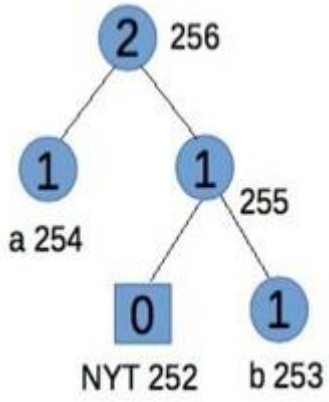
设 a、b 固定编码
a=01100001
b=01100010
编码“abb”给出 01100001 001100010 11。

第 1 步：

从一棵空树开始。
对于“a”输出其二进制代码。
Output：01100001
第 2 步：



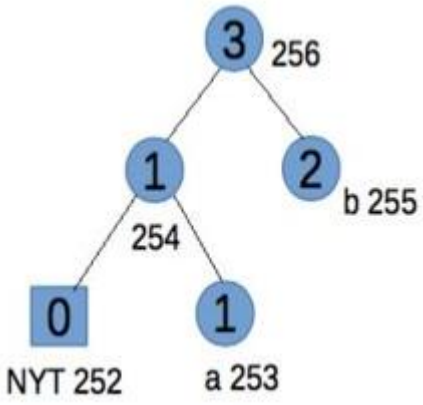
NYT 产生两个子节点：254 和 255，两者都具有权重 0。增加根和 255 的权重。与节点 255 相关联的“a”的权重是 1。
对于“b”输出 0（对于 NYT 节点）然后是其二进制代码。
Output：0110 0001　0 0110 0010
第 3 步：



NYT 产生两个子节点：NYT 为 252，叶节点为 253，权重为 0.增加权重为 253,254 和 root。

为了维持 Vitter 的不变性，即权重 w 的所有叶子都在权重 w 的所有内部节点之前（在隐式编号中），从节点 254 开始的分支应当与节点 255 交换（就符号和权重而言，而不是数字排序）。
“b”的代码是 11。

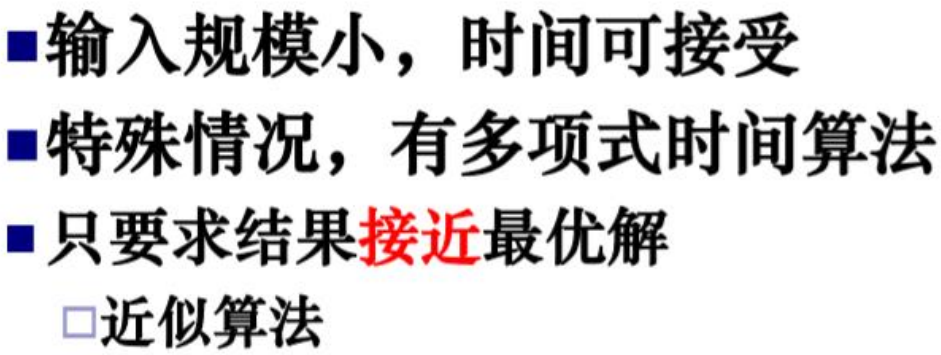
对于第二个“b”编码 11。
为了便于解释，这一步并不完全遵循 Vitter 的算法[1]，但效果是等效的。
Output：0110 0001　0 0110 0010　11
第 4 步：



转到叶节点 253.注意，我们有两个权重为 1 的块。节点 253 和 254 是一个块（由叶子组成），节点 255 是另一个块（由内部节点组成）。对于节点 253，其块中的最大数字是 254，因此交换节点 253 和 254 的权重和符号。现在，节点 254 和从节点 255 开始的分支满足 SlideAndIncrement 条件[1]，因此必须是 swop。最后增加节点 255 和 256 的权重。

“b”的未来代码为 1，而“a”现在为 01，这反映了它们的频率。

近似算法—对抗 NP 完全问题



近似程度的度量

- 代价函数 cost(·)
- 最优解的代价是 C^*
- 近似解的代价是 C
- 近似比
  - $\max\left\{\frac{C}{C^*}, \frac{C^*}{C}\right\} = \rho(n)$
  - 1 -近似算法——最优算法
  - $(1+\varepsilon)$-近似算法 $O(n^{\frac{1}{\varepsilon}})$

近似算法——正确性

- 算法输出是可以覆盖原图的顶点集

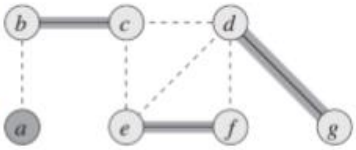
- 代价函数输出的节点个数
- 最优解的代价|C*|
- 近似解的代价|C|
- ∃C,|C|<= 2|C*|

近似算法 $O(n+m)$

- Function approx_vertex_cover(E)
- Input: E 图中边的集合
- Output: 覆盖原图的顶点集合C

1. C = ∅
2. while E 不空
   A. 从E中任选一条边(u,v)
   B. C = C ∪ {u, v}
   C. 从E中删除所有和u,v相关的边
3. return C

近似算法——正确性



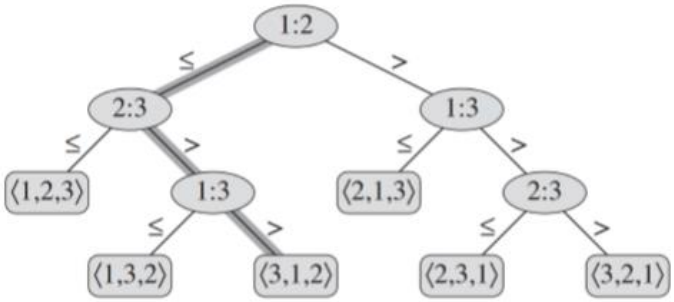
- 定理 ∀C, |C|<= 2|C*|
- 证明：考虑由2A步挑选的n条边构成的子图，这个子图有2n个节点，覆盖这个子图的最优解需要n个节点。
- 显然覆盖原图的节点数大于等于覆盖它的图的节点数n，即|C*|>=n
- |C| = 2n <= 2|C*|
- $\max\left\{\frac{|C|}{|C^*|},\frac{|C^*|}{|C|}\right\}=2$

四、

计算机学院

复习

比较排序



- 输入列表长度 n
- 叶子节点的个数 l： $l \geq n!$
- 若二叉树的高度为 h，其叶子节点的个数为 2^h （满二叉树）
- $2^h \geq l \geq n! \Rightarrow 2^h \geq n!$
- $h \geq \lg(n!) = \theta(n\lg n)$

- $\theta(n\lg n)$是比较次数的下界，是最差情形下比较排序的时间复杂度下界
- $\theta(n)$ 是比较排序的时间复杂度全局下界

比较排序可以被抽象为一棵决策树，决策树是一棵完全二叉树，它可以表示在给定输入规模情况下，某一特定排序算法对所有元素的比较操作。
一个比较排序算法的最坏情况比较次数就等于其决策树的高度。等于Ω(nlgn)