算法分析

算法

算法 (algorithm) 就是任何良定义的计算过程，该过程取某个值或值的集合作为 输入并产生某个值或值的集合作为输出。这样算法就是把输入转换成输出的计算步骤的一个 序列。

算法是一个计算步骤的序列，用以将输入转换为输出。

~~想到子体检时阿姨问我什么叫算法~~，必考。

检测有毒药的水瓶

有1000瓶水，其中有一瓶有毒，小白鼠只要尝一点带毒的水24小时后就会死亡。 至少要多少只小白鼠才能在24小时时鉴别出那瓶水有毒?

1. 直观的想法：一只老鼠喝一瓶水，需要1000只老鼠；
2. 稍有改进的想法：只需要999只老鼠，如果没有老鼠死亡，则是没有喝的那一瓶。

问题转换：如何从0～999中找到特定的数字?

使用二分法，则至少需要$\lfloor \log_2 n \rfloor$只老鼠，但需要10天的时间。 在二分法查找过程中，10只老鼠需要按序进行喝水。

考虑采用**编码**的方式来求解，先考虑一个小规模问题。 假设有编号为0~7的八个水瓶，其中只有一瓶中装有毒药，我们按照二进制编码方式， 八个水瓶的标签分别为，hui

水瓶	编号
0	000
1	001
2	010
3	011
4	100
5	101
6	110
7	111

每只老鼠按照二进制编号的列，如果为1，则喝该水瓶中的水，否则不喝。 例如，第一只老鼠所对应的列为 `0000 1111` ，所以它只喝后四瓶水，也就是编号为4、5、6和7的水瓶。

如果编号为2的水瓶有毒，那么只有第二只老鼠会死亡。 如果第一只和第二只老鼠都死亡，那么说明编号为6的水瓶有毒。 这样，通过编号与水瓶的对应，可以判断哪些老鼠死亡时，是哪一杯水有毒。

从使用编码解决问题的过程中，学到了两点：

1. 复杂问题可以先从简单方法入手；
2. 大规模问题可以先从小规模试解。

最大公约数

欧几里得算法

```
fn gcd(a: i32, b: i32) {
   match b {
      0 => a,
      _ => gcd(b, a%b),
   }
}
```

连续整数检测算法

```
fn gcd(a: i32, b: i32) {
   let t = min(a, b);
   while a%t != 0 && b%t != 0 {
      t -= 1;
   }
   t
}
```

初始输入a和b不能为0，而欧几里得算法则可以。

质因数法

找到a和b所有的质因数；如果p是公因数，那么在结果中p将重复$\min pa, pb$次。 其中pa和pb分别为p在a和b的因式中出现的次数。

例如，

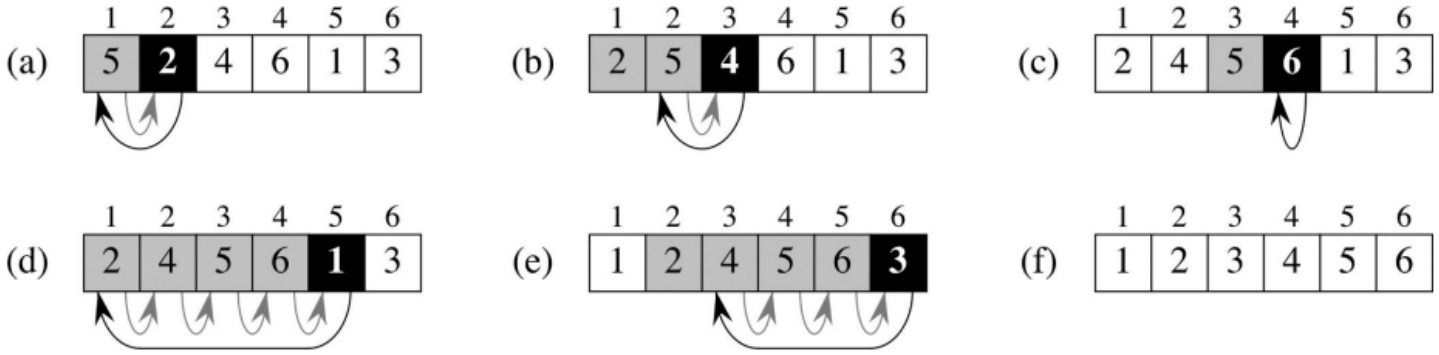
$$60 = 2 \times 2 \times 3 \times 5$$
$$24 = 2 \times 2 \times 2 \times 3$$

因此有，

$$gcd(60, 24) = 2 \times 2 \times 3 = 12$$

插入排序

插入排序在数组 A=<5, 2, 4, 6, 1, 3>上的执行过程。

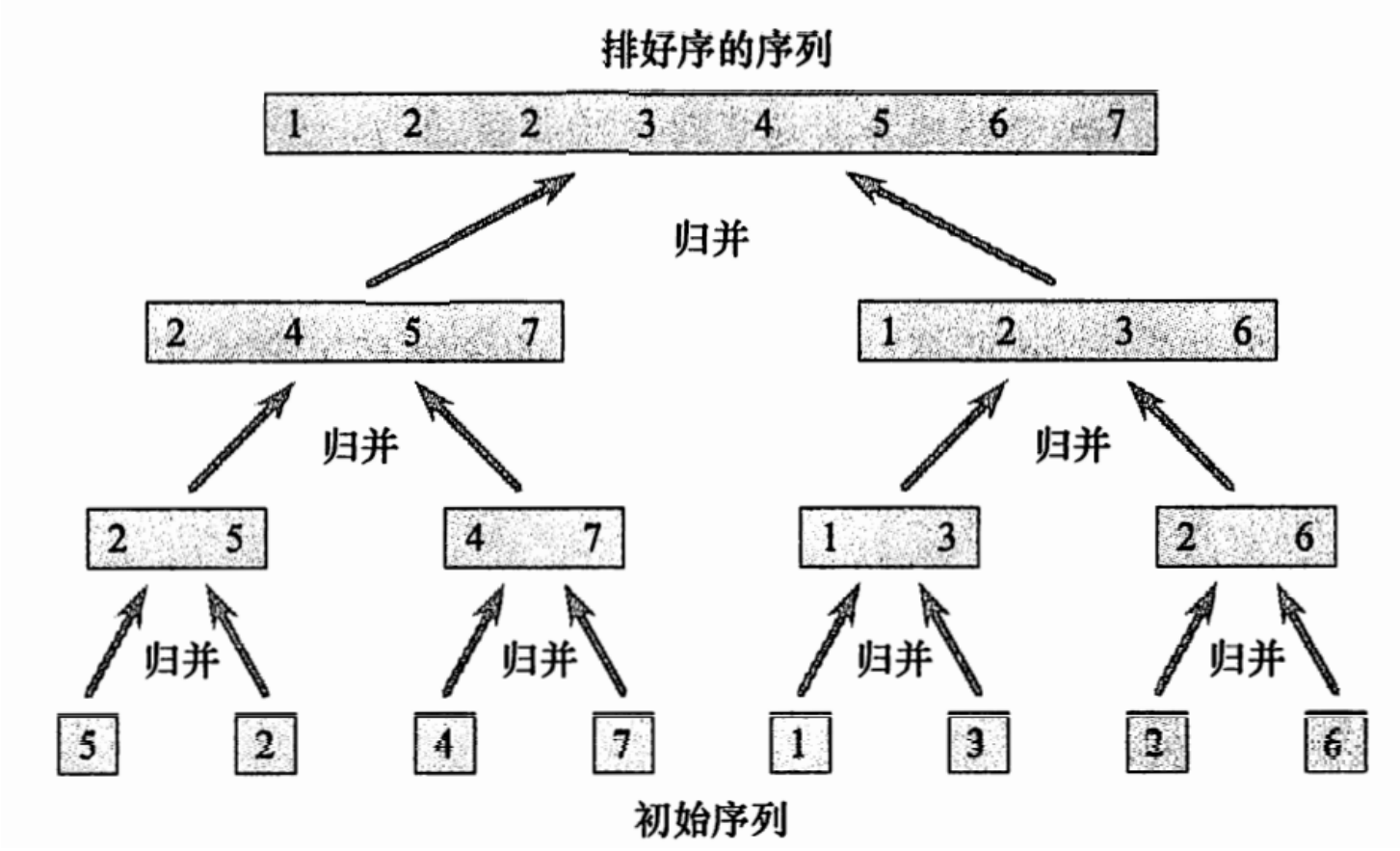


插入排序算法，逐行分析。

INSERTION-SORT(A)	代价	次数
1 **for** $j = 2$ **to** $A.length$	c_1	n
2 $key = A[j]$	c_2	$n-1$
3 // Insert $A[j]$ into the sorted sequence $A[1..j-1]$.	0	$n-1$
4 $i = j - 1$	c_4	$n-1$
5 **while** $i > 0$ and $A[i] > key$	c_5	$\sum_{j=2}^{n} t_j$
6 $A[i+1] = A[i]$	c_6	$\sum_{j=2}^{n} (t_j - 1)$
7 $i = i - 1$	c_7	$\sum_{j=2}^{n} (t_j - 1)$
8 $A[i+1] = key$	c_8	$n-1$

归并排序

归并排序在数组 A=<5, 2, 4, 7, 1, 3, 2, 6〉上的操作。



> 描述一个运行时间为$\Theta(n \log n)$的算法，给定n个整数的集合S和另一个整数x，该算法能确定S中是否存在两个其和刚好为x的元素。

LeetCode经典题目，两数之和。

1. 排序后使用双指针查找；
2. 借助hashmap。

斯特林（Stirling）近似公式

$$n! = \sqrt{2\pi n}\left(\frac{n}{e}\right)^n\left(1+\Theta\left(\frac{1}{n}\right)\right)$$

推论：

$$n! = o(n^n)$$

$$n! = \omega(2^n)$$

$$\lg(n!) = \Theta(n\lg n)$$

多项式边界（polynomially bounded）

Proving that a function $f(n)$ is polynomially bounded is equivalent to proving that $\lg(f(n)) = O(\lg n)$ for the following reasons.

- 如果f有多项式边界，那么存在$f(n) \leq cn^k$，进而有$\lg(f(n)) \leq kc\lg n$，即$\lg(f(n)) = O(\lg n)$；
- 同理，如果$\lg(f(n)) = O(\lg n)$，那么f就有多项式边界。

概率分析与随机算法

> 假设有10张牌，每张牌上分别标有从10的数字，且每张牌上的数字均不同，将牌 充分混合。然后从牌堆中一次一张地移除三张牌。那么我们选出的三张牌按照（递增）顺 序排列的概率是多少?

抽三张牌可能的排列总数为$10 \times 9 \times 8 = 720$， 而抽三张牌的组合数为$\binom{10}{3} = 120$，其中一个组合对应多种排列，有且仅有一种排列 是递增的，因此三张牌的组合数就是抽三张牌为递增的事件数量。

综上所诉，概率为$P = 120/720 = 1/6$。

参考链接

> 你参加一个游戏。该游戏将奖品藏在了三个幕布之后。如果你选对了幕布，则可以赢得 奖品。在你选择了一个幕布后，但是幕布还未揭开之前，主持人会揭开另两个幕布中一 个空幕布（主持人知道哪个幕布后是空的），之后会询问你要不要改变你的选择。请问如 果你改变了选择，那么你赢得奖品的几率将如何改变？（这一问题是著名的 Monty hall 题，是以一个主持人经常让参赛者陷入这种困境的节目命名的。）

三门问题，个人的理解： 一开始在A、B和C的概率分别是$1/3$，在选择A后，奖品在B和C（不在A中）中的概率为$2/3$。

这时候，主持人确定奖品不在B中，所以此时奖品不在A中的概率就等同于在C中的概率$2/3$。 可以粗略的说，原本在B的概率流向了C中。

> 一个监狱看守从三个罪犯中随机挑选一个释放，并处死另两人。这个看守知道每个人会 被释放还是处死，但是被禁止透漏给囚犯其自身的处置信息。称罪犯为X、Y和Z。罪 X以他已经知道了中至少有一人会死为理由，私下问警卫两人中哪个会被处 死。警卫不能透露给关于他自身的信息，但他告诉了X, Y将被处死。 感到很开 心，因为他认为他或者将被释放，这意味着他被释放的概率现在是1/2了。请问他的 想法正确吗，或者他被释放的概率仍为1/3? 请解释。

一共有三种情况，XY被处死，YZ被处死，XZ被处死。 如果是XY这种情况，看守只能告诉X是Y将被处死（被禁止透漏给囚犯其自身的处置信息）。

设事件B是得知Y将被处死，事件A_1是XY被处死，事件A_2是X被释放，Y被处死（即YZ被处死）。

因此有条件概率，

$$P(B|A_1) = 1, \quad P(B|A_2) = 1/2$$

因此，在得知Y将被处死的条件下，X存活（事件A_2）发生的概率可以由贝叶斯公式计算，

$$\begin{aligned} P(A_2|B) &= \frac{P(B|A_2)P(A_2)}{P(B|A_1)P(A_1)+P(B|A_2)P(A_2)} \\ &= \frac{1/2 \times 1/3}{1 \times 1/3 + 1/2 \times 1/3} \\ &= 1/3 \end{aligned}$$

> 在一场狂欢节游戏中，将3个骰子放在一个罩子中。一位游戏者可以在1到6中的任意数字 上赌1美元。主持人摇罩子，并按如下方案确定游戏者所得回报。如果游戏者赌的数字没有 出现在任何一个骰子上，则他输掉1美元。如果他赌的数字恰好出现在k个骰子上， k=l, 2, 3, 则他可以保留他的1美元，并赢得k美元。请计算玩这个游戏一次的期望收入。

美元	概率
-1	$125/215$
1	$25/72$
2	$5/72$
3	$1/216$

以$k=2$为例，需要有选两个骰子与选择的点数相同，剩下一个还要与它不同。总的样本空间数就是 三个骰子组成的所有可能性。

$$P = \frac{\binom{3}{2} \times 5}{6^3} = 5/72$$

最后数学期望为$E(X) = -0.08$。

拉斯维加斯（Las Vegas）算法

一旦用拉斯维加斯算法找到一个解，那么这个解肯定是正确的，但有时用拉斯维加斯算法可能找不到解。

例如：八皇后问题，每次随机分配八个皇后的位置，检查这个解是否成立，如果不成立则重新调用算法。

舍伍德（Sherwood）算法

总能求得问题的一个解，且求得的解总是正确的。

很多算法对于不同的输入实例，其运行时间差别很大。此时，可以采用舍伍德型概 率算法来消除算法的时间复杂性与输入实例间的这种联系。

例如：在快排前对数组进行一次随机打乱，防止初始序列使枢纽点的选择变成最差的情况。 把所有情况变成平均的情况。

> 请描述 RANDOM(a, b) 过程的一种实现，它只调用 RANDOM(O, 1) 。作为a和b的函 数，你的过程的期望运行时间是多少?

```
// RANDOM
fn random(a: u32, b: u32) -> u32 {
    //returns a random integer number between a and b, includes a and b
    let (p, len, diff) = (1, 1, b - a);
    //Sift [a, b] to [0, b-a]
    //Generate a random number in [0, b-a] by bit
    while p < diff {
       p = p << 1;
       len += 1;
    }
    //Get smallest number p is greater than difference in the power of 2 format
    let rst;

    loop {
        rst = 0;

        for i in 1..=len {
            rst = (rst << 1) + RANDOM(0, 1);
        }

        if rst <= b {
            break;
        }
    }

    rst + a
}
```

这个算法可以看出有拉斯维加斯算法思想的，因为区间大小不一定为2的幂，因此我们是假设范围是 覆盖区间的最小2的幂，然后随机生成一个数，如果这个数不在结果区间内，那就重新跑一次算法（loop）。

> 假设你希望以 1/2的概率输出 。你可以自由使用一个输出的过程 BIASED-RANDOM。它以某概率p输出 1, 概率1-p输出0, 其中 $O < p < 1$, 但是p的值未知。 请给出一个利用 BIASED-RANDOM 作

> 为子程序的算法，返回一个无偏的结果，能以概率 1/2返回0, 以概率1/2返回 。作为p的函数，你的算法的期望运行时间是多少?

还是拉斯维加斯算法的思想，构造概率相等的情况，BIASED-RANDOM生成1和0的概率分别为p和1-p， 那么生成10这个事件的概率就是p(1-p)，同理生成01的概率为(1-p)p，这两个不同的事件概率相等。

```
//UNBIASED-RANDOM
fn unbiased_ramdom() -> u32 {
    loop {
        let x = biased_random();
        let y = biased_random();
        if x != y {
            break x;
        }
    }
}
```

> 在HIRE-ASSISTANT 中，假设应聘者以随机顺序出现，你正好雇用一次的概率是多少？ 正好雇用n次的概率是多少?

应聘者的**质量**（优秀程度）可以看作<1, 2, 3, ..., n>的一个排列， 正好雇用一次是因为最优秀的应聘者在1号位置； 正好雇用n次是应聘者质量递增。

因此概率分别为，

$$P_1 = \frac{(n-1)!}{n!} = 1/n \quad P_2 = \frac{1}{n!}$$

> 在HIRE-ASSISTANT 中，假设应聘者以随机顺序出现，你正好雇用两次的概率是多少?

可以第一个人隐藏后面的人，直到最优秀的应聘者出现。

E_i是第1个面聘者是第i优秀，显然$P(E_i) = 1/n$。 F是事件第2个到第j-1应聘者不如第1位应聘者。$F|e_i$前i人的顺序只要保证第一个人是i，后 n-i名候选者需要第一个人是最优秀的。

$$P(F|E_i) = \frac{(i-1)!(n-i-1)!}{(i-1)!(n-i)!} = 1/(n-i)$$

只雇用两次事件$A = F \cap (E_1 \cup E_2 \cup \cdots \cup E_n)$，

进而概率，

$$
\begin{aligned}
P(A) &= \sum_{i=1}^{n-1} P(F \cap E_i) \\
&= \sum_{i=1}^{n-1} P(F|E_i)P(E_i) \\
&= \frac{1}{n}\sum_{i=1}^{n-1} \frac{1}{n-i}
\end{aligned}
$$

错误想法

~~第二个人最优秀，其余人随意排列，概率仍旧为$1/n$。~~

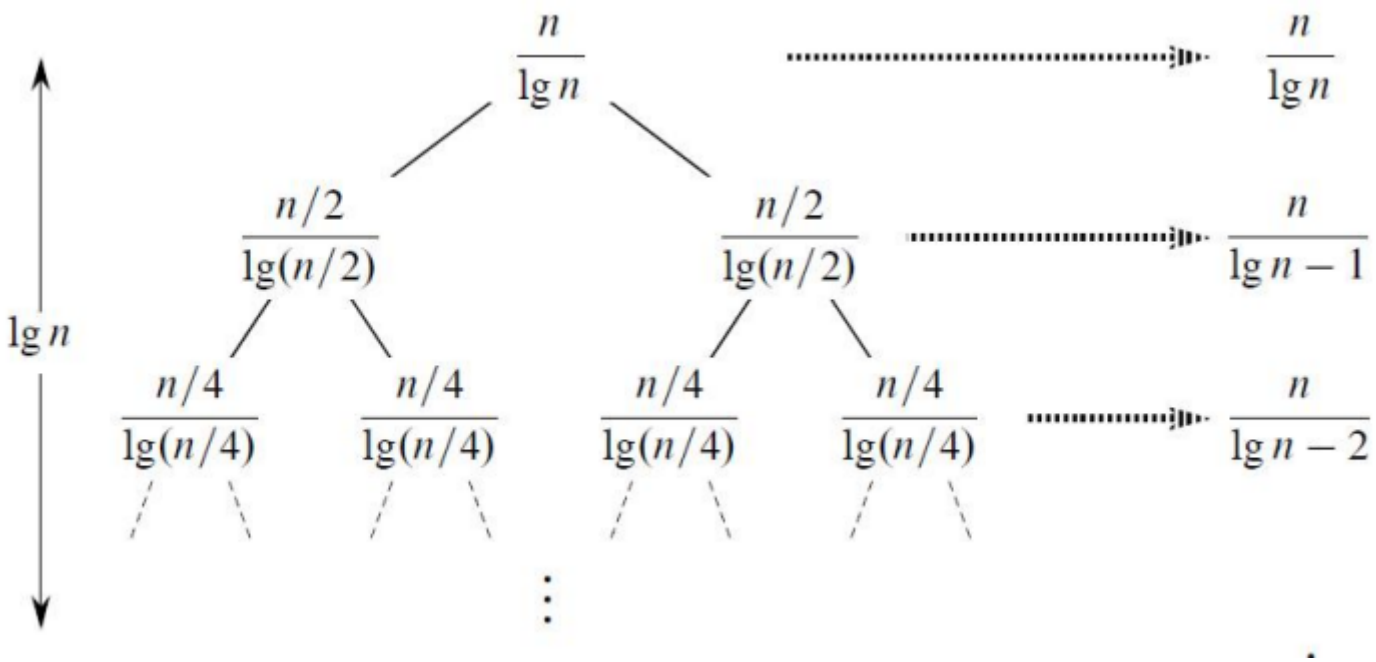
> Armstrong 教授建议用下面的过程来产生一个均匀随机排列：
>
> ```
> PER TE-BY-CYCLIC(A)
> 1 n = A. length
> 2 let B[l. . n] be a new array
> 3 offset= RANOOM(l, n)
> 4 for i = 1 to n
> 5     dest = i + offset
> 6     if dest > n
> 7         dest = dest - n
> 8         B[dest] = A[i]
> 9 return B
> ```
>
> 请说明每个元素 A[i]出现在B中任何特定位置的概率是 1/n 。然后通过说明排列结果不 是均匀随机排列，表明 Armstrong 教授错了。

每个元素A[i]将会放到B数组中的dest位置，dest是1到n等可能的；但每个元素任意位置是等可能的， 并不代表总体排列是均匀的，因为dest可能会发生冲突。

分治法

> 用递归树方法解$T(n) = 2T(n/2) + n/\lg n$。

将树展开到第k层，$T(n/2^k)$，并且假设$2^k = n(k = log_2 n)$方便计算，



第k层的每个节点的代价为$(n/2^k)/\log_2(n/2^k)$，而第k层一共有2^k个节点，因此第k层的 总代价为$n/\log_2(n/2^k) = n/(log_2 n - k)$。

计算全部k层代价之和，

$$\begin{aligned}\sum_{k=0}^{\log_2 n-1}\frac{n}{\log_2 n-k} =& \frac{n}{\log_2 n}+\frac{n}{\log_2 n-1}+\cdots+\frac{n}{2}+\frac{n}{1}\\ =& n\sum_{k=1}\log_2 n\frac{1}{k}\\ =& n(\log_2\log_2 n+O(1))\\ =& \Theta(n\log_2\log_2 n)\end{aligned}$$

主定理

$$T(n) = aT(n/b) + f(n), \quad a \geq 1, b > 1, f(n) > 0$$

$$T(n)=\begin{cases}\Theta(n^{\log_b^a}) & \text{if} f(n)\in O(n^{\log_b a-\epsilon}),\epsilon>0\\ \Theta(n^{\log_b^a}\lg^{k+1}n) & \text{if } f(n)\in\Theta(n^{\log_b a}\lg^k n),k\geq 0\\ \Theta(f(n)) & \text{if} f(n)\in\Omega(n^{\log_b a+\epsilon}),\epsilon>0\end{cases}$$

正则条件

$$af(n/b)\leq cf(n)$$

二维最近点对问题

最大子数组问题

Strassen矩阵乘法

凸包问题

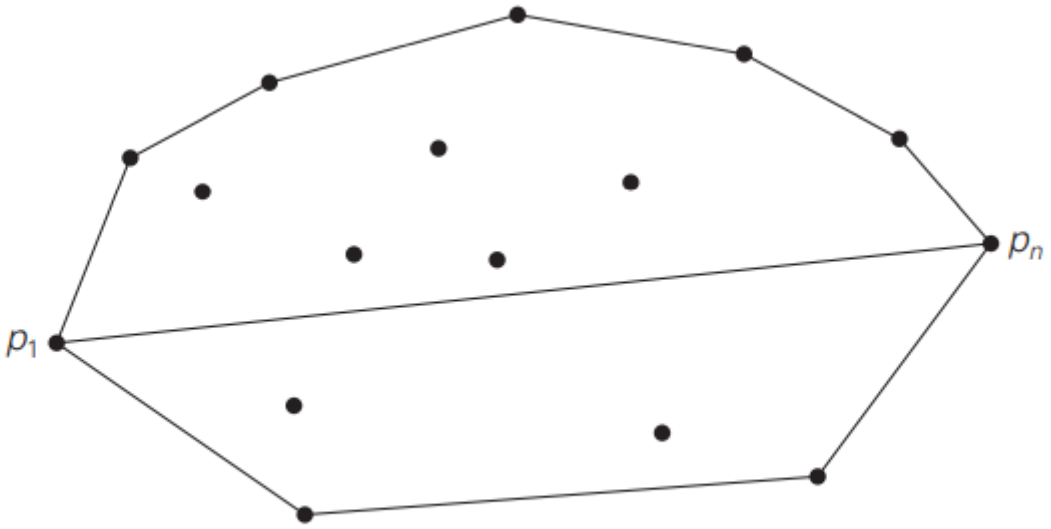


FIGURE 5.8 Upper and lower hulls of a set of points.

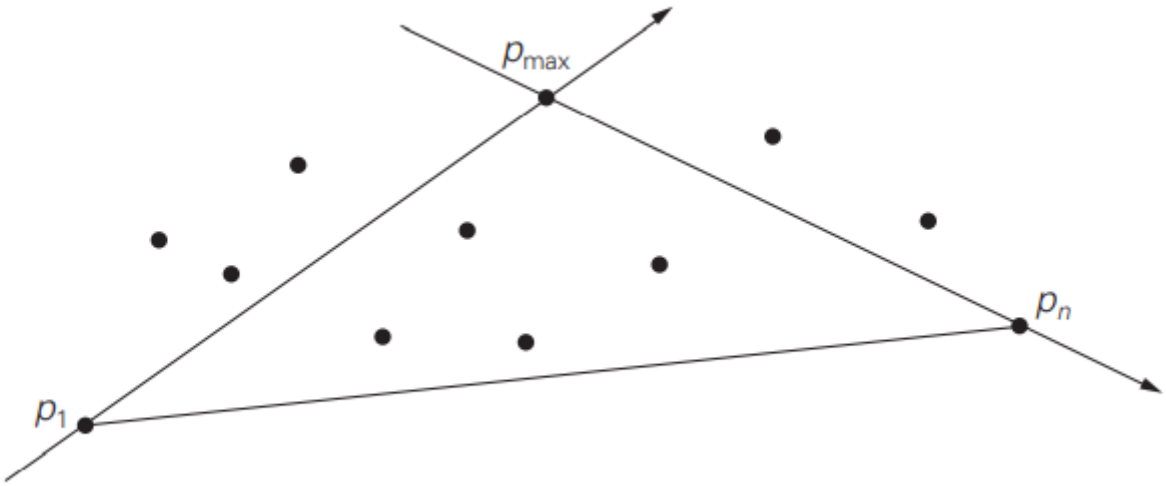


FIGURE 5.9 The idea of quickhull.

快速凸包算法平均时间复杂度$O(n \lg n)$，最坏情况下时间复杂度为$O(n^2)$。

在排序之后，找到最左侧和最右侧的点P_1和P_n，将凸包问题划分成两个子凸包。

对于上面的凸包，可以确定$P_{\max}$一定是上包的顶点。 $\Delta P_1 P_n P_{\max}$内的点一定不是上包顶点，不用去考虑。

向量$P_1 \vec{P}_{\max}$和向量$\vec{P_{\max}P_n}$左侧的点不存在 （注意向量有方向，图中最上方部分）。

判断点是否在向量的左侧的方法，$P_1(x_1, y_1), P_2(x_2, y_2), P_3(x_3, y_3)$，计算

$$\begin{vmatrix} x_1 & y_1 & 1 \\ x_2 & y_2 & 1 \\ x_3 & y_3 & 1 \end{vmatrix}$$

如果结果为正，则P_3位于向量$\vec{P_1P_2}$左侧。

基于树的搜索

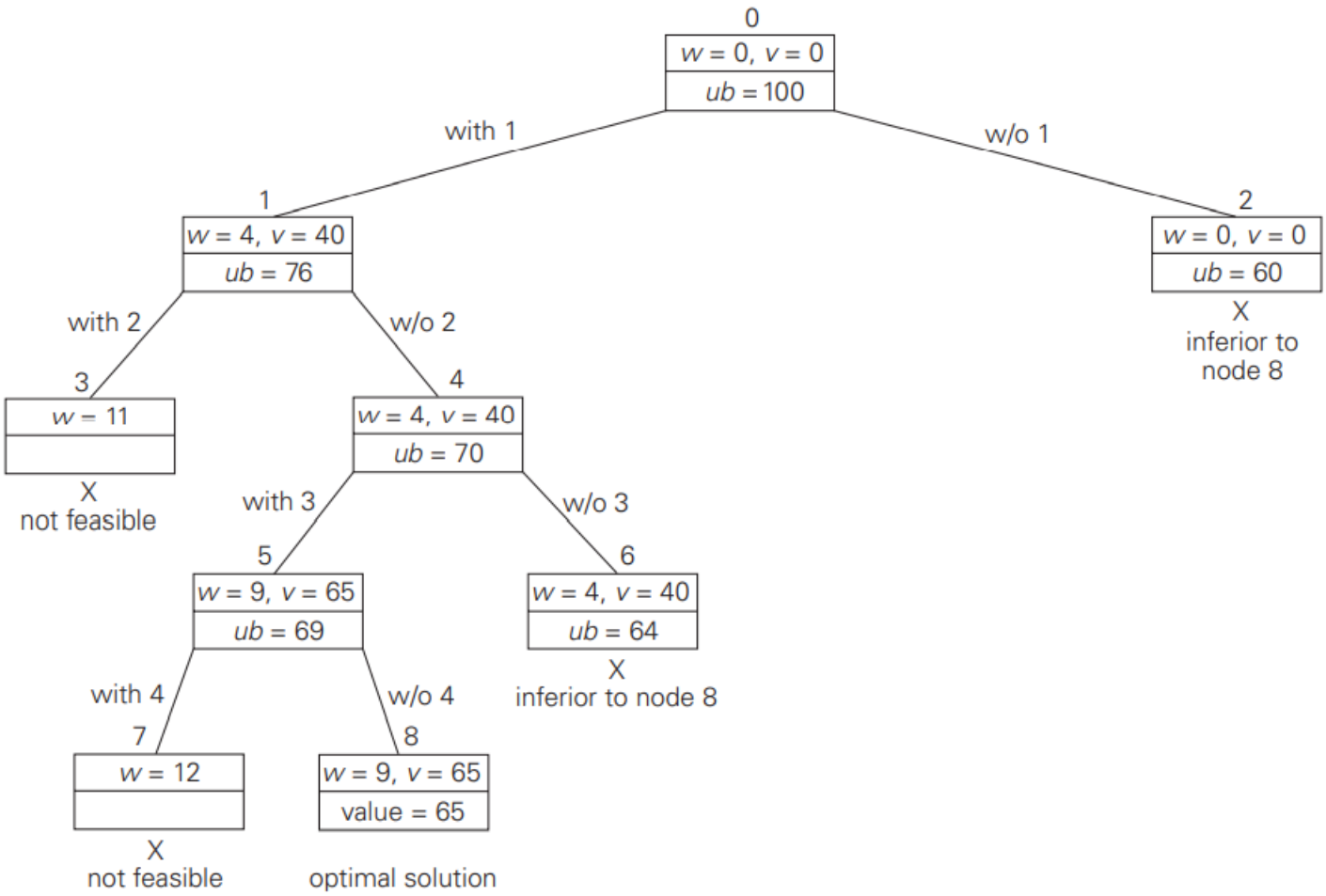
深度优先搜索、广度优先搜索

回溯法、分支界限法

回溯法的求解目标是找出T中满足约束条件的所有解， 而分支限界法（Branch-and-Bound）的求解目标则是找出满足约束条件的一个解， 或是在满足约束条件的解中找出使某一目标函数值达到极大或极小的解， 即在某种意义下的最优解。

item	weight	value	$\frac{\text{value}}{\text{weight}}$
1	4	\$40	10
2	7	\$42	6
3	5	\$25	5
4	3	\$12	4

The knapsack's capacity W is 10.



局部剪枝搜索（Local Beam Search）

局部剪枝搜索从k个随机生成的状态开始， 每步生成全部k个状态的所有后继状态，

- 如果其中之一是目标状态，算法停止；
- 态中选择最佳的k个状态继续搜索。

在局部剪枝搜索过程中，有用的信息在k个并行的搜索线程之间传递， 算法会很快放弃没有成果的搜索而把资源放在取得最大进展的搜索。

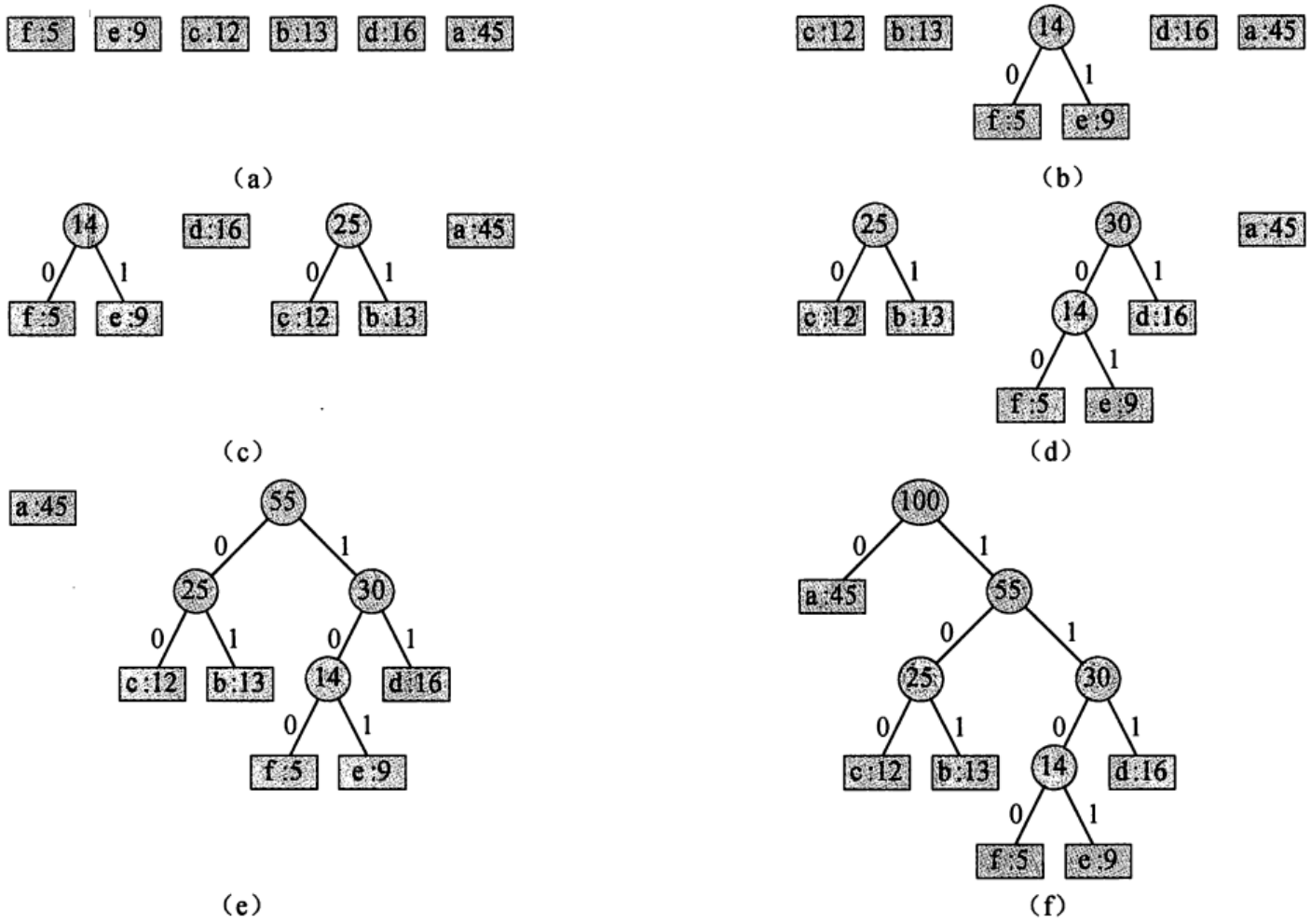
随机剪枝搜索

随机剪枝搜索不是选择最好的k个后代， 而是按照一定概率随机地选择k个后继状态。

选择给定后继状态的概率是状态值的递增函数，用随机来增加状态的多样性。

贪心算法

Huffman编码




```
HUFFMAN(C)
1 n=ICI
2 Q=C
3 for i=1 to n—1
4 allocate a new node z
5 z.left = x = EXTRACT-MIN(Q)
6 z.right = y = EXTRACT-MIN(Q)
7 z.freq = x. freq + y. freq
8 INSERT(Q, z)
9 return EXTRACT-MIN(Q)
```

Kruskal算法

对所有的边排序，从权重较小的边开始遍历，判断能否加入到结果集合中。

对所有边进行排序：$O(|E|\lg|E|)$，

从最小的边开始遍历：$O(E)$，

判断该边能否加入到结果中，也就是判断边的两个顶点是否在同一集合中， 使用并查集数据结构，查找的速度快，$O(1)$，查找后需要合并操作，$O(\lg|V|)$。 因为$E \leq V^2$，所以$\lg|E| = O(\lg V)$。 总体时间复杂度为$O(|E|\lg|V|)$。

Prim算法

从初始根节点开始，每次增加一条最小的安全边。

使用最小堆：排序的不在树中的结点到树的最小边的权重

8-11行更新弹出结点后，堆中其余结点权重更新。

```
MST-PRIM(G,w,r)
1 for each uEG. V
2     u.key = INFTY
3     u:father = NIL
4 r:key=0
5 Q=G.V
6 while Q is not empty
7     u= EXTRACT-MIN(Q)
8     for each v in G.Adj[u]
9     if vE Q and w(u,v) < v.key
10     v.father = u
11     v. key = w(u,v)
```

活动选择问题

区间贪心：优先选择最早结束的

0-1背包问题

没有贪心法解决方案，可以参考分支界限法求解0-1背包问题。 但贪心算法可以求解部分背包问题（Fractional Knapsack Problem）。

所谓部分背包问题，就是可以将0-1背包问题中的每件物品只拿取一部分，而不是拿整个物品。

动态规划

动态规划通过组合子问题的解来求解原问题。 与分治法相比，分治法是把问题划分为互不相交的子问题；而动态规划应用于子问题重叠的情况。

计算Fibnacci数

$$F(n) = F(n-1) + F(n-2) \qquad F(0) = 0, F(1) = 1$$

把计算$F(n)$这个问题分解为两个更小的交叠子问题$F(n-1)$和$F(n-2)$。

0-1背包问题

之前使用分支界限法求解0-1背包问题，也可以用贪心算法求解部分背包问题。

使用动态规划求解0-1背包问题，设$V[i,j]$是能放进重量为j的前i个物品最有价值子集的总价值。

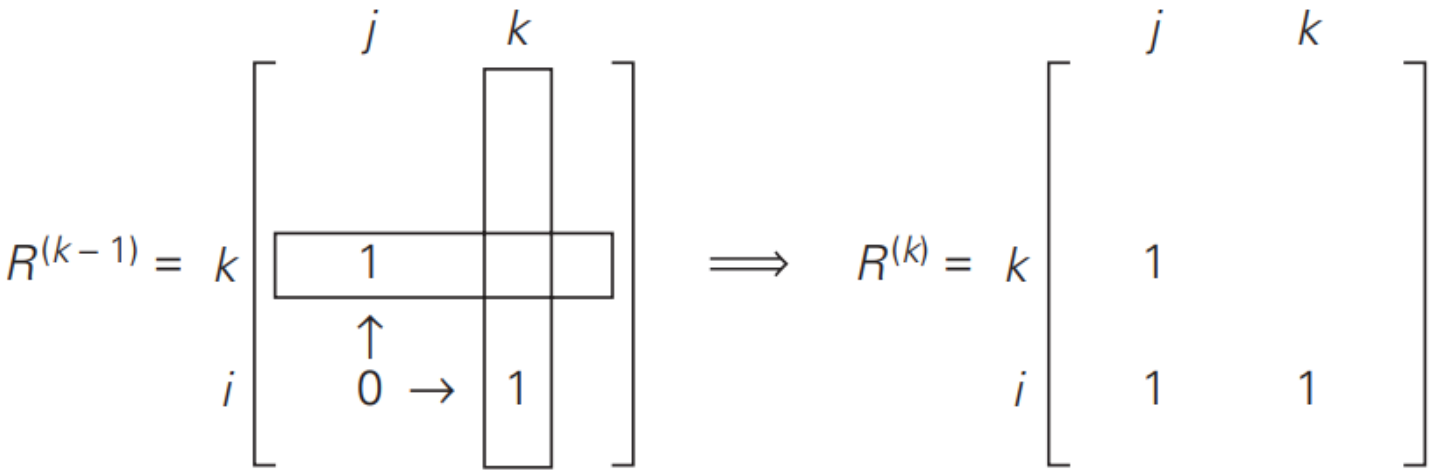
对于第i个物品，如果背包空间不足，那一定不会考虑放它； 如果背包空间能容纳他，$V[i,j]$的含义是，前$i-1$个物品的最优子集加上第i个物品， 放入承受重量为$j-w_i$的背包。或者，不放入第i个物品， 前$i-1$个物品的最优子集放入承重为j的背包。

```
Algorithm OptimalKnapsack(w[1..n], v[1..n], W)
//Finds the items composing an optimal solution to the knapsack problem
//Input: Arrays w[1..n] and v[1..n] of weights and values of n items,
//       knapsack capacity W, and table V [0..n, 0..W] generated by
//       the dynamic programming algorithm
//Output: List L[1..k] of the items composing an optimal solution
k ← 0 //size of the list of items in an optimal solution
j ← W //unused capacity
for i ← n downto 1 do
    if V [i, j] > V [i – 1, j]
        k ← k + 1; L[k] ← i //include item i
        j ← j – w[i]
return L
```

传递闭包问题

Warshall算法计算有向图的传递闭包，$R^{(k)}$是矩阵中任意两点的路径中间点的数量不超过k个。

$R^{(0)}$就是有向图的邻接矩阵，$R^{(k)}$的计算只与$R^{(k-1)}$有关。



多源最短路径

Floyd算法，核心逻辑就是i和j之间的距离能否通过借道k来缩短。

```
Algorithm Floyd(W[1..n, 1..n])
//Implements Floyd’s algorithm for the all-pairs shortest-paths problem
//Input: The weight matrix W of a graph with no negative-length cycle
//Output: The distance matrix of the shortest paths’ lengths
D ← W //is not necessary if W can be overwritten
for k ← 1 to n do
    for i ← 1 to n do
        for j ← 1 to n do
            D[i, j] ← min{D[i, j], D[i, k] + D[k, j]}
return D
```

最长公共子序列（LCS）

状态转移方程

$$c[i,j]=\begin{cases}0 & i=0 \text{ or } j=0\\ c[i-1,j-1]+1, & i,j>0 \text{ and } x_i=y_i\\ \max(c[i,j-1],c[i-1,j]) & i,j>0 \text{ and } x_i\neq y_i\end{cases}$$

P、NP和NP完全问题

P类问题

P类问题是一类能够用**确定的**算法在多项式的时间内求解的**判定问题**。

不可判定问题（停机问题）给定一段计算机程序和它的一个输入，判断该程序对于该输入是否会中止，还是会无限运行下去。

可以判定但难求解的问题

- 哈密顿回路
- 旅行商问题
- 背包问题
- 划分问题
- 装箱问题
- 图的着色问题
- 整数线性规划

不确定算法

不确定算法：把一个判定问题的实例I作为它的输入，

- 非确定（猜测）阶段：生成任意串S，把它当作给定实例I的一个候选解。
- 确定（验证）阶段：确定算法把I和S都作为它的输入，如果S是I的一个解，就输出“是”。

NP类问题

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。

停机问题是不属于NP的判定问题。（NP-Hard）

NP完全问题

一个判定问题D是NP完全问题，条件是：

1. 它属于NP类型；
2. NP中的任何问题都能够在多项式时间内化简为D。

哈密顿回路问题可以多项式化简为旅行商问题的判定版本。