Growth of functions

渐进紧确界Θ

$$\Theta(g(n)) = \{f(n) : \exists c_1, c_2, n_0 \in \mathbb{R}^+, s.t. \\ \forall n \geq n_0, 0 \leq c_1 g(n) \leq f(n) \leq c_2 g(n)\}$$

渐进上界O

$$O(g(n)) = \{f(n) : \exists c, n_0 \in \mathbb{R}^+, s.t. \\ \forall n \geq n_0, 0 \leq f(n) \leq cg(n)\}$$

渐进下界Ω

$$\Omega(g(n)) = \{f(n) : \exists c, n_0 \in \mathbb{R}^+, s.t. \\ \forall n \geq n_0, 0 \leq cg(n) \leq f(n)\}$$

非渐进紧确上下界o、ω

$$o(g(n)) = \{f(n) : \forall c > 0, \exists n_0 > 0, \\ s.t. \forall n \geq n_0, 0 \leq f(n) < cg(n)\}$$

Denoted as $f(n) = o(g(n))$. Intuitively, $\lim_{n\to\infty} \frac{f(n)}{g(n)} = 0$.

$$\omega(g(n)) = \{f(n) : \forall c > 0, \exists n_0 > 0, \\ s.t. \forall n \geq n_0, 0 \leq cg(n) < f(n)\}$$

The relation $f(n) = \omega(g(n))$ implies that $\lim_{n\to\infty} \frac{f(n)}{g(n)} = \infty$.

$$f(n) = O(g(n)) \iff g(n) = \Omega(f(n)) \\ f(n) = o(g(n)) \iff g(n) = \omega(f(n))$$

易错点: $n!$:

$2^n <_o n! <_o n^n$:

易错点: lgn:

lgn 处理, 令 $n = 2^k$:

$$(lgn)! = k! >_o 2^{3k} = n^3$$

$\lceil \frac{a}{b} \rceil \leq \frac{a+(b-1)}{b}$

$\lfloor \frac{a}{b} \rfloor \leq \frac{a-(b-1)}{b}$

$a^{log_b c} = c^{log_b a}$(两边取logb即可证)

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

多重对数函数:

Functional iteration

$$f^{(i)}(n) = \begin{cases} n & i = 0 \\ f(f^{(i-1)}(n)) & i > 0 \end{cases}$$

The iterated logarithm function:

$\lg^* n = \min\{i \geq 0 : \lg^{(i)} n \leq 1\}$ $\lg^* 2 = 1, \lg^* 4 = 2, \lg^* 16 = 3, \lg^* 65536 = 4, \lg^*(2^{65536}) = 5.$

几个事实：$lg^*(lg\, n) \approx lg^*\, n - 1$，故$lg^*(lg\, n) < lg(lg^* n)$

另一个事实：宇宙原子数目n远小于2^{65536}，即$\lg^* n < 5$

Recurrences

猜想+数学归纳法

例子：T(n)=9T(n/3)+n，猜想T(k)<ck^2+dk

The master method applies to recurrences of the form

$$T(n) = aT(n/b) + f(n),$$

where $a \geq 1, b > 1$, and f is asymptotically positive.

Three common cases

Compare $f(n)$ with $n^{\log_b a}$:

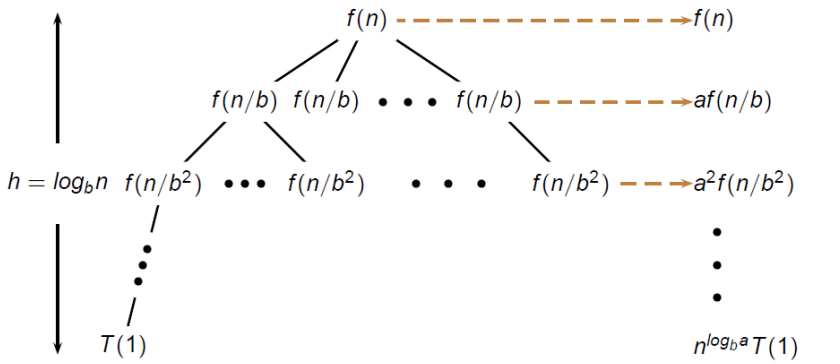
1. If $f(n) = O(n^{\log_b a - \epsilon})$ for some constant $\epsilon > 0$, then $T(n) = \Theta(n^{\log_b a})$.
2. If $f(n) = \Theta(n^{\log_b a})$, then $T(n) = \Theta(n^{\log_b a} \lg n)$.
3. If $f(n) = \Omega(n^{\log_b a + \epsilon})$ for some constant $\epsilon > 0$, and if $af(n/b) \leq cf(n)$ for some constant $c < 1$ and all sufficiently large n, then $T(n) = \Theta(f(n))$.

Xsu1023 14:21 回复

case 1、2之间有gap
如f(n)=O(n^logb_a*logn)则既不属于1也不属于2
同理，case2、3也有gap
最后，主定理使用时需要满足f>0的正则化

- $T(n) = 3T(n/4) + n \lg n$
  $a = 3, b = 4, f(n) = n \lg n, f(n) = \Omega(n^{\log_4 3 + \epsilon})$, where $\epsilon \approx 0.2$. For sufficiently large n,
  $af(n/b) = 3(n/4) \lg(n/4) \leq (3/4)n \lg n$ for $c = 3/4$.
  By **case 3**, $T(n) = \Theta(n \lg n)$.

主定理的证明：



$$T(n) = \Theta(n^{\log_b a}) + \sum_{j=0}^{\log_b n - 1} a^j f(n/b^j)$$

技巧：

Changing variables

$$T(n) = 2T(\lfloor \sqrt{n} \rfloor) + \lg n$$

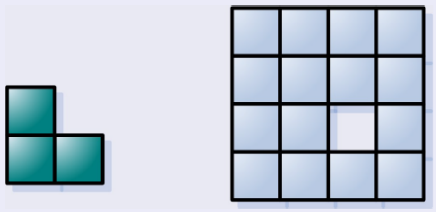
- Let $m = \lg n$, then $T(2^m) = 2T(2^{m/2}) + m$.
- Let $S(m) = T(2^m)$, then $S(m) = 2S(m/2) + m$.
- $T(n) = T(2^m) = S(m) = \Theta(m \lg m) = \Theta(\lg n \lg \lg n)$.

练习4.6-2：如果$f(n) = \Theta(n^{log_b a} lg^k n)$则$T(n) = \Theta(n^{log_b a} lg^{k+1} n)$

Divide and Conquer

分治三步骤：Divide Conquer Combine

1. Triomino拼图，用一个L型瓦片（含三个方块）覆盖一个缺少了一个方块的$2^n \times 2^n$的棋盘。设计此问题的分治算法并分析复杂度。



基本思想：大的L型可以通过小的L型获得（归纳法），那么对于一个2^n*2^n，将它分为四个$2^{n-1}*2^{n-1}$，三个没空格的用大L型填充，一个有空格的继续递归

大整数乘法：

Big integers multiplication

A general example

$$
\begin{aligned}
c = a * b &= (a_1 10^{n/2} + a_0) * (b_1 10^{n/2} + b_0) \\
&= (a_1 * b_1)10^n + (a_1 * b_0 + a_0 * b_1)10^{n/2} \\
&\quad + (a_0 * b_0) \\
&= c_2 10^n + c_1 10^{n/2} + c_0 \\
c_1 &= (a_1 + a_0) * (b_1 + b_0) - (c_2 + c_0)
\end{aligned}
$$

Divide: $\Theta(n)$ Conquer: 3T(n/2) Combine: $\Theta(n)$ Total: $\Theta(n^{log_2 3})$

矩阵乘法:

Idea of Divide and Conquer

Divide a $n \times n$ matrix multiplication into 2×2 $(n/2) \times (n/2)$ submatrix multiplication.

$$
\begin{bmatrix} r & s \\ t & u \end{bmatrix} = \begin{bmatrix} a & b \\ c & d \end{bmatrix} \cdot \begin{bmatrix} e & f \\ g & h \end{bmatrix}
$$

$$
C = A \cdot B
$$

$$
\begin{aligned}
r &= ae + bg, & s &= af + bh, \\
t &= ce + dg, & u &= cf + dh.
\end{aligned}
$$

Strassen's idea

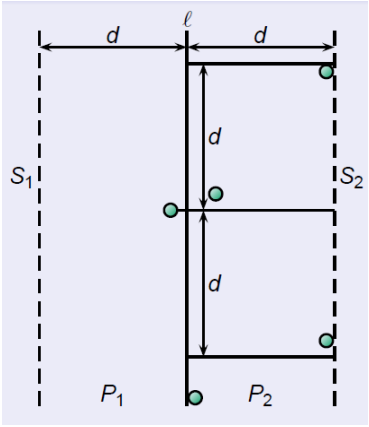
$$
\begin{aligned}
P_1 &= a \cdot (f - h) \\
P_2 &= (a + b) \cdot h \\
P_3 &= (c + d) \cdot e \\
P_4 &= d \cdot (g - e) \\
P_5 &= (a + d) \cdot (e + h) \\
P_6 &= (b - d) \cdot (g + h) \\
P_7 &= (a - c) \cdot (e + f)
\end{aligned}
$$

$$
\begin{aligned}
r &= P_5 + P_4 - P_2 + P_6 \\
s &= P_1 + P_2 \\
t &= P_3 + P_4 \\
u &= P_5 + P_1 - P_3 - P_7
\end{aligned}
$$

7 mults, 18 adds/subs

$T(n) = 7T(n/2) + \Theta(n^2)$ Total: $T(n) = \Theta(n^{log_2 7}) \approx \Theta(n^{2.81})$

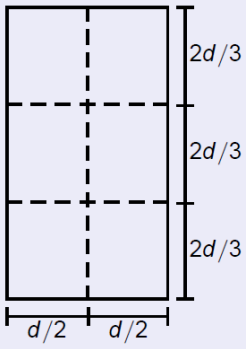
最近点对:



How many points in the region?

6 is the maximum!

$$(x(u)-x(v))^2+(y(u)-y(v))^2 \le (d/2)^2+(2d/3)^2 = 25d^2/36$$



$T(n)=2T(n/2)+\Theta(n)$ Total: $\Theta(nlogn)$

Randomized Algorithms

雇佣问题：随机排列一串数（满意度），若i满意度大于1~(i-1)中所有数满意度，则解雇当前最优雇佣i，求雇佣次数的期望：

$EX_i=(k-1)!/k!=1/k$，则$total=\sum 1/k=O(lg\ n)$

注意：

$$\ln(n+1)=\int_1^{n+1}\frac{dx}{x}\le\sum_{k=1}^{n}\frac{1}{k}\le\int_1^{n}\frac{dx}{x}+1=\ln n+1$$

随机序列1：为序列每一个数i随机选择一个rank=random(1,n^3)，按照rank排序，可以证明所有rank唯一的概率是1-1/n

随机序列2：i从1到n，随机选择swap(A[i], A[random(i,n)])，可以证明给定一个排列S，交换后A变成S的概率是$1/n!$，即每个排列都是等可能的

在线雇佣问题：前k个不雇佣，设最大值maxi，后(n-k)个雇佣第一次大于maxi的人；（计算时注意只需考虑最大值位置i前面的i-1个数之中的最大值放在哪里）k=n/e时雇佣maxi的可能性最大（1/e）

Heapsort

下标：从1开始，与oi时相同，<<1、<<1|1、>>1均相同

MAX-HEAPIFY: 将当前i向下递归，通过与儿子交换保持最大/小堆的性质,$T(n)\le T(2n/3)+\Theta(1)\Rightarrow T(n)=O(lg\ n)$

BUILD-MAX-HEAP: 从下向上对每一个节点进行MAX-HEAPIFY，通过对每一层累加得复杂度O(n)

HEAPSORT：选择最后一个数和顶点交换，然后对顶点MAX-HEAPIFY

HEAP-EXTRACT-MAX：同HEAPSORT

HEAP-INCREASE-KEY：（最大堆）上滤

MAX-HEAP-INSERT：堆底扩充一个负无穷然后用HEAP-INCREASE-KEY

Quicksort

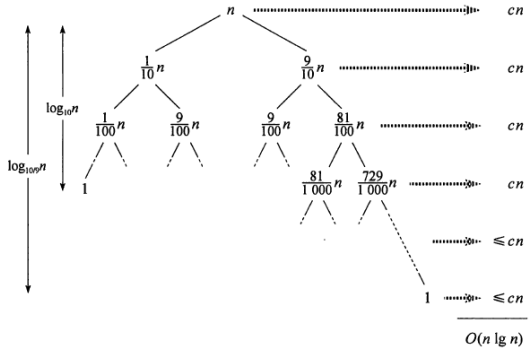
分治时采用双指针，ij均从左端，i表示小于key的指针，j表示已经遍历过的指针

若T(n)=T(9n/10)+T(n/10)+cn，则如下画出递归树，每一层均为cn，$cnlog_{10}n\le T(n)\le cnlog_{10/9}n$

例如，假设划分算法总是产生 9：1 的划分，乍一看，这种划分是很不平衡的。此时，我们得到的快速排序时间复杂度的递归式为：

$$T(n) = T(9n/10) + T(n/10) + cn$$

这里，我们显式地写出了 $\Theta(n)$项中所隐含的常数 c。图 7-4 显示了这一递归调用所对应的递归树。注意，树中每一层的代价都是 cn，直到在深度 $\log_{10} n = \Theta(\lg n)$处达到递归的边界条件时为



随机选择pivot的复杂度：

$$\begin{aligned}\Pr\{z_i \text{ 与 } z_j \text{ 进行比较}\} &= \Pr\{z_i \text{ 或 } z_j \text{ 是集合 } Z_{ij} \text{ 中选出的第一个主元}\} \\ &= \Pr\{z_i \text{ 是集合 } Z_{ij} \text{ 中选出的第一个主元}\} \\ &\quad + \Pr\{z_j \text{ 是集合 } Z_{ij} \text{ 中选出的第一个主元}\} \\ &= \frac{1}{j-i+1} + \frac{1}{j-i+1} = \frac{2}{j-i+1} \qquad (7.3)\end{aligned}$$

上式中第二行成立的原因在于其中涉及的两个事件是互斥的。将公式(7.2)和公式(7.3)综合起来，有：

$$\mathrm{E}[X] = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} \frac{2}{j-i+1}$$

在求这个累加和时，可以将变量做个变换($k=j-i$)，并利用公式(A.7)中给出的有关调和级数的界，得到：

$$\mathrm{E}[X] = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} \frac{2}{j-i+1} = \sum_{i=1}^{n-1} \sum_{k=1}^{n-i} \frac{2}{k+1} < \sum_{i=1}^{n-1} \sum_{k=1}^{n} \frac{2}{k} = \sum_{i=1}^{n-1} O(\lg n) = O(n \lg n) \qquad (7.4)$$

于是，我们可以得出结论：使用 RANDOMIZED-PARTITION，在输入元素互异的情况下，快速排序算法的期望运行时间为 $O(n \lg n)$。

Sorting in Linear Time

决策树：每个分支代表一种可能比较结果之后还需要进行的比较；叶节点包含所有n!种排列，因此决策树树高h满足
$h \geq log_2(n!) = \Omega(nlogn)$

计数排序Counting Sort：

i from 1 to n: C [A [i]]++;

i from 1 to k: C [i] += C [i - 1]

i from n to 1: B [C [A [i] --] = A [i] // 注意，逆序为了保证排序稳定性

基数排序Radix Sort：

给定一个d位数，每个数位k个可能取值（k进制），则RADIX-SORT使用**稳定排序**法耗时$\Theta(n+k)$，总耗时$\Theta(d(n+k))$

给定一个b比特数（2^b），对于任意$r \leq b$，可以选取2^r进制，从而使得总耗时为$\Theta(\frac{b}{r}(n+2^r))$：

- 当$b < logn$时，稀疏，选取$r = b$最优，有总耗时$\Theta(n)$
- 当$b > logn$时，稠密，选取$r = logn$最优，有总耗时$\Theta(bn/logn)$

桶排序Bucket Sort：

现在来分析调用插入排序的时间代价。假设 n_i 是表示桶 $B[i]$中元素个数的随机变量，因为插入排序的时间代价是平方阶的(见 2.2 节)，所以桶排序的时间代价为：

$$T(n) = \Theta(n) + \sum_{i=0}^{n-1} O(n_i^2)$$

我们现在来分析桶排序在平均情况下的运行时间。通过对输入数据取期望，我们可以计算出期望的运行时间。对上式两边取期望，并利用期望的线性性质，我们有：

$$\begin{aligned} \mathrm{E}[T(n)] &= \mathrm{E}\left[\Theta(n) + \sum_{i=0}^{n-1} O(n_i^2)\right] \\ &= \Theta(n) + \sum_{i=0}^{n-1} \mathrm{E}[O(n_i^2)] \quad \text{（利用期望的线性性质）} \\ &= \Theta(n) + \sum_{i=0}^{n-1} O(\mathrm{E}[n_i^2]) \quad \text{（利用公式(C.22)）} \end{aligned} \tag{8.1}$$

我们断言：

$$\mathrm{E}[n_i^2] = 2 - 1/n \tag{8.2}$$

对所有 $i=0, 1, \cdots, n-1$ 成立。这一点不足为奇：因为输入数组 A 的每一个元素是等概率地落入任意一个桶中，所以每一个桶 i 具有相同的期望值 $\mathrm{E}[n_i^2]$。为了证明公式(8.2)，我们定义指示器随机变量：对所有 $i=0, 1, \cdots, n-1$ 和 $j=1, 2, \cdots, n$，

$$X_{ij} = \mathrm{I}\{A[j] \text{ 落入桶 } i\}$$

因此，

$$n_i = \sum_{j=1}^{n} X_{ij}$$

算出$E(n_i^2)$即可最终$E(T(n)) = \Theta(n)$.

Medians and Order Statistics

同时找到最大最小值：取两个数内部比较，较大的跟目前最大值比，较小的跟目前最小值比，比较次数为$\lfloor 3n/2 \rfloor$

随机化下期望为线性的算法:

随机选择pivot，将array劈成两半，寻找对应的一半

SELECT，指示器随机变量 X_k 恰好在给定的 k 值上取值 1，对其他值都为 0。当 $X_k=1$ 时，我们可能要递归处理的两个子数组的大小分别为 $k-1$ 和 $n-k$。因此可以得到递归式：

$$\begin{aligned} T(n) &\leqslant \sum_{k=1}^{n} X_k \cdot (T(\max(k-1, n-k)) + O(n)) \\ &= \sum_{k=1}^{n} X_k \cdot T(\max(k-1, n-k)) + O(n) \end{aligned}$$

两边取期望值，得到

$$\begin{aligned} \mathrm{E}[T(n)] &\leqslant \mathrm{E}\left[\sum_{k=1}^{n} X_k \cdot T(\max(k-1, n-k)) + O(n)\right] \\ &= \sum_{k=1}^{n} \mathrm{E}[X_k \cdot T(\max(k-1, n-k))] + O(n) \quad \text{（期望的线性性质）} \\ &= \sum_{k=1}^{n} \mathrm{E}[X_k] \cdot \mathrm{E}[T(\max(k-1, n-k))] + O(n) \quad \text{（利用公式(C.24)）} \end{aligned}$$

$$= \sum_{k=1}^{n} \frac{1}{n} \cdot \mathrm{E}[T(\max(k-1, n-k))] + O(n) \quad \text{（利用公式(9.1)）}$$

公式(C.24)的应用依赖于 X_k 和 $T(\max(k-1, n-k))$是独立的随机变量。练习 9.2-2 要求证明这个命题。

下面来考虑一下表达式 $\max(k-1, n-k)$。我们有

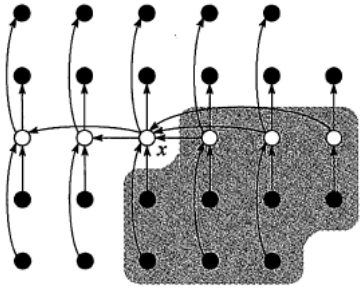
$$\max(k-1, n-k) = \begin{cases} k-1 & \text{若 } k > \lceil n/2 \rceil \\ n-k & \text{若 } k \leqslant \lceil n/2 \rceil \end{cases}$$

如果 n 是偶数，则从 $T(\lceil n/2 \rceil)$到 $T(n-1)$的每一项在总和中恰好出现两次。如果 n 是奇数，除了 $T(\lfloor n/2 \rfloor)$出现一次以外，其他这些项也都会出现两次。因此，我们有

$$\mathrm{E}[T(n)] \leqslant \frac{2}{n} \sum_{k=\lfloor n/2 \rfloor}^{n-1} \mathrm{E}[T(k)] + O(n)$$

然后数学归纳法假设T(n)<cn，即可得证.

最坏为线性的算法



将n个数每5个分一组，共$\lceil n/5 \rceil$组；每组内插入排序找到中位数，然后对所有$\lceil n/5 \rceil$个中位数递归调用算法寻找中位数，时间开销T($\lceil n/5 \rceil$)；之后根据中位数的中位数x将$\lceil n/5 \rceil$组分为两堆，已知比x小的最少有$3(\lceil \lceil n/5 \rceil /2 \rceil - 2) \geq \frac{3n}{10} - 6$（减去的两个一个是x所在，一个是可能残缺的一组），最多$7n/10 + 6$个；比x大的也如是，因此有$T(n) \leq T(\lceil n/5 \rceil) + T(7n/10 + 6) + O(n)$，归纳假设$\forall n \geq 140, T(n) \leq cn$，展开即可得证.

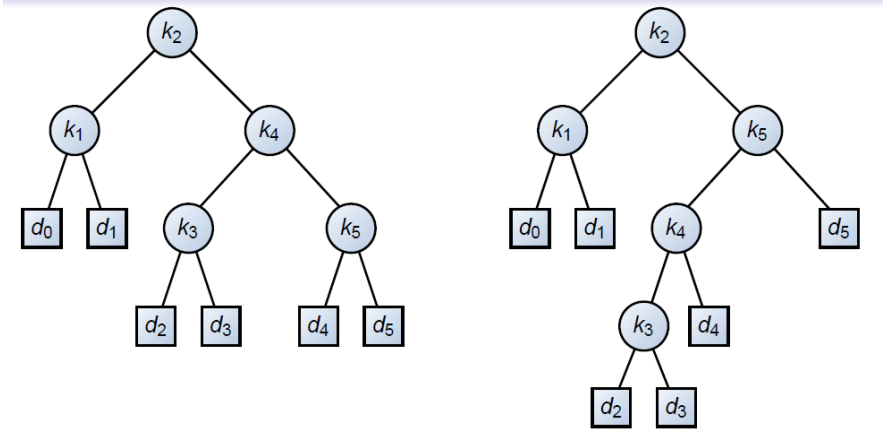
Dynamic Programming

最优子结构：一个问题的最优解包含子问题的最优解

u、v之间最长简单路径（中间取一个变量w作为中间结点）不具有最优子结构，NPC问题，原因是两个子问题不独立，会彼此影响

典型问题：钢条切割（完全背包）、矩阵分割（魔法石）、LCS（两个array，讨论A[i]与B[j]是否相等）

最优二叉搜索树



e[i, j] = p_r + (e[i, r − 1] + w(i, r − 1)) + (e[r + 1, j] + w(r + 1, j))

$$w(i,j) = \sum_{l=i}^{j} p_l + \sum_{l=i-1}^{j} q_l$$

注意这种分段问题，先从长度开始枚举，移动前端点并枚举中间终断点

Greedy Algorithms

最优子结构：与DP不同的是，它**要求贪心地选出一个必然存在于最优结构中的元素**a_k，并且将S_{ij}分为S_{ik}、a_k、S_{kj}，在两个S中求最优子结构

证明方法：取一个最优解，证明贪心选出的元素一定在最优解里

典型问题：**活动调度（取出结束时间最早的元素）、Huffman编码（取出频率最低的两个元素）、单位时间任务（每个任务1个单位时间，未在规定时间前完成受到惩罚，求惩罚的最小值）**

拟阵是序偶M=(S,I)，满足：

- S有限，I是S的子集集合，I元素称为S的独立子集
- I遗传：$\forall B \in I \& A \subseteq B,\ A \in I$
- M交换：$\forall A, B \in I, if\ |A| < |B|, then\ \exists x \in B - A\ s.t.\ A \cup \{x\} \in I$

图拟阵：$M_G = (S_G, I_G)$，其中S_G是G的子图(V, A)的边集A，I是所有不成环的边集集合

- 证明交换性：对于森林有连通分量=|V|-|E|，|A|<|B|显然有S_A分量数大于S_B，分析B的每个分量中的点在A中的分量归属，显然存在一个B中分量T使得其中有两点属于A的不同分量；由于T连通，因此存在T中相邻两点uv在A中属于不同分量，且A+(u,v)不构成环

矩阵向量拟阵：某一矩阵的列向量构成S

加权拟阵：每个S中元素都有正权值；最优拟阵是指I中权值最大的元素

性质：

- 拟阵最大独立子集有相同大小

- 贪心选择的元素在最优加权子集里：寻找一个最优解S，向目前的贪心选择元素里面加入S的元素直至和S等长证明

Krustal：O(nlogn+nf(n)) 其中f(n)是并查集复杂度

Amortized Analysis

聚合法、核算法、势能法

$\hat{c_i} = c_i + \Phi(i) - \Phi(i-1)$

表扩张/收缩：

Table contraction

When to contract?

- When $\alpha(T) = num[T]/size[T] < 1/2$?
- $\alpha(T) < 1/4$ is a good choice.

Potential function

$$\Phi(T) = \begin{cases} 2 \cdot num[T] - size[T] & \text{if } \alpha(T) \geq 1/2, \\ size[T]/2 - num[T] & \text{if } \alpha(T) < 1/2. \end{cases}$$

Fibonacci Heaps（只有pop、delete是logn的）

势函数：$\Phi(x) = t(H) + 2m(H)$，其中t(H)表示根节点数目，m(H)表示已标记节点数目

INSERT：建立一个根节点，并且确定是否需要移动min指针，**与二叉堆O(logn)相比，摊还代价为O(1)**

UNION：合并堆，直接根节点合并更新min指针，**与二叉堆O(n)相比，摊还代价为O(1)**

EXTRACT-MIN：抽取最小节点，将每个min的孩子都变为根节点，然后进行CONSOLIDATE，O(logn)

- CONSOLIDATE：开辟数组A[0..D(n)]，对于每个根节点填充A，若A未填充则填充，已填充则选取key较小的作为父亲节点两两合并，且**清除MARK标记**，根节点度数+1，继续查看是否冲突，直至无冲突；所有节点遍历完后统一寻找min，摊还代价O(logn)

DECREASE-KEY：关键词减值，改变堆序则CUT并对父亲节点级联CUT，**与二叉堆O(logn)相比，摊还代价O(1)**

- CUT：直接移到根节点，MARK=false
- CASCADING-CUT：如果没标记则标记，已标记则剪去移到根节点并**取消标记**，递归继续级联CUT

DELETE：设为最小值然后pop最小值（O(logn)）

INCREASE-KEY：DELETE后INSERT（O(logn)）

性质

- 设x是堆节点，$x.degree = k$，$y_1 \cdots y_k$是按照链入先后排序的x的儿子节点，则有$y_i.degree \geq i - 2$
- 设x是堆节点，$x.degree = k$，则$size(x) \geq F_{k+2} \geq \phi^k$
- n个节点的堆中任意节点最大度数为D(n)=O(logn)

String Matching（此章尤其需要注意下标）

- prefix of a string, denoted $w \sqsubset x$, if $x = wy$ for some string $y \in \sum^*$.
- suffix of a string, denoted $w \sqsupset x$, if $x = yw$ for some string $y \in \sum^*$.

把模板串（Pattern）前k个字符组成的前缀记为P_k

把文本串（Text）k个字符组成的后缀记为T_k

Algorithm	Preprocessing	Matching time
Naive	0	$O((n-m+1)m)$
Rabin-Karp	$\Theta(m)$	$O((n-m+1)m)$
Finite automaton	$O(m\|\sum\|)$	$\Theta(n)$
Knuth-Morris-Pratt	$\Theta(m)$	$\Theta(n)$
Boyer-Moore	$\Theta(m+\|\sum\|)$	$\Omega(n/m), O(mn), O(n)$

Rubin-Karp

朴素想法$t_{s+1}=10(t_s-10^{m-1}T[s+1])+T[s+m+1]$

$$t_{s+1}=(d(t_s-T[s+1]h)+T[s+m+1])\bmod q$$

其中$h\equiv d^{m-1}(mod\ q)$

```
RABIN-KARP-MATCHER(T,P,d,q)
 1  n = T.length
 2  m = P.length
 3  h = d^(m-1) mod q
 4  p = 0
 5  t0 = 0
 6  for i = 1 to m                    // preprocessing
 7      p = (dp+P[i]) mod q
 8      t0 = (dt0+T[i]) mod q
 9  for s = 0 to n-m                  // matching
10      if p == ts
11          if P[1..m] == T[s+1..s+m]
12              print"Pattern occurs with shift"s
13      if s<n-m
14          ts+1=(d(ts-T[s+1]h )+T[s+m+1]) mod q
```

假设是随机生成的串，伪匹配上的概率是1/q，设有效匹配个数为v，那么Rubin-Karp算法运行时间为$O(n+m(v+n/q))$

实际上，当匹配串为a^m，模板串为a^n时，能够达到运行时间上限$O(m(n-m+1))$.

Automata

终态函数$\phi(wa)=\delta(\phi(w),a),\phi(\epsilon)=q_0$

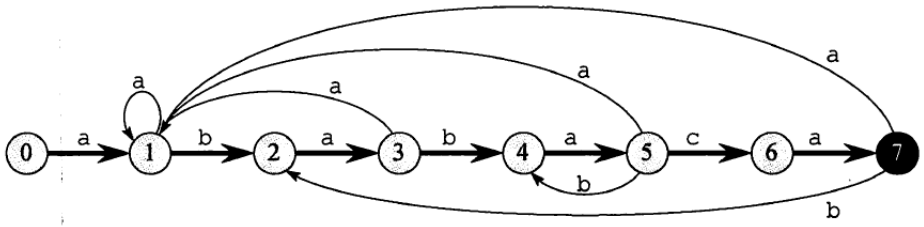
后缀函数$\sigma(x):=max\{k:P_k\sqsupset x\}$

转移函数$\delta(q,a):=\sigma(P_qa)$

性质：$\phi(T_{i+1})=\delta(T_ia)=\delta(\phi(T_i),a)=\sigma(P_qa)=\sigma(T_ia)=\sigma(T_{i+1})$ (其中$\phi(T_i)=q$)

```
1: π ← COMPUTE-PREFIX-FUNCTION(P)
2: for a in Σ do
3:     δ(0,a) = 0
4: end for
5: δ(0,P[1]) = 1
6: for a in Σ do
7:     for q in range 1,m do
8:         if q = m or P[q+1] ≠ a then
9:             δ(q,a) = δ(π[q],a)
10:        else
11:            σ(q,a) = q+1
12:        end if
13:    end for
14: end for
```

预处理复杂度$O(m|\Sigma|)$，运行复杂度O(n)



KMP

前缀函数：$\pi[i] = max\{k : k < i \ \& \ P_k \sqsupset P_i\}$

i	1	2	3	4	5	6	7	8	9	10
$P[i]$	a	b	a	b	a	b	a	b	c	a
$\pi[i]$	0	0	1	2	3	4	5	6	0	1

```
KMP-MATCHER(T,P)
 1  n=T.length
 2  m=P.length
 3  π=COMPUTE-PREFIX-FUNCTION(P)
 4  q=0                                  // number of characters matched
 5  for i=1 to n                         // scan the text from left to right
 6      while q>0 and P[q+1]≠T[i]
 7          q=π[q]                       // next character does not match
 8      if P[q+1] == T[i]
 9          q=q+1                        // next character matches
10      if q == m                        // is all of P matched?
11          print "Pattern occurs with shift" i−m
12      q=π[q]                           // look for the next match
```

```
COMPUTE-PREFIX-FUNCTION(P)
 1  m=P.length
 2  let π[1..m] be a new array
 3  π[1]=0
 4  k=0
 5  for q=2 to m
 6      while k>0 and P[k+1]≠P[q]
 7          k=π[k]
 8      if P[k+1]==P[q]
 9          k=k+1
10      π[q]=k
11  return π
```

均摊法算出计算π数组O(m)，匹配O(n)

Boyer-moore

Bad Character Shift

Pattern: GCAGAGAG

$bmBc[A] = 1; bmBc[C] = 6;$
$bmBc[G] = 2; bmBc[T] = 8$

Pattern: ANPANMAN

$bmBc[A] = 1; bmBc[M] = 2;$
$bmBc[N] = 3; bmBc[P] = 5$

shift: bmBc[s+ i] +m - i

Good Suffix Shift

$Osuff[i] = max\{k : P[i-k+1..i] = P[m-k+1..m]\}$

- 即P_i与P的最大重合后缀位数

i	1	2	3	4	5	6	7	8
P[i]	G	C	A	G	A	G	A	G
Osuff[i]	1	0	0	2	0	4	0	8

```
void suffix(char *pattern,int m,int suff[]) {
   int f, g, i;
   suff[m-1]=m;
   g=m-1;
   for(i=m-2;i>=0;--i){
      if(i>g&&suff[i+m-1-f]<i-g)
         suff[i]=suff[i+m-1-f];
      else {
         if(i< g)
            g=i;
         f=i;
         while(g>=0&&pattern[g]==pattern[g+m-1-f])
            --g;
         suff[i]=f-g;
      }
   }
}
```

```
COMPUTE-BMBC(P, bmBc)
1  m = P.length
2  for each character a ∈ ∑
3        bmBc[a] = m
4  for i = 1 to m − 1
5        bmBc[P[i]] = m − i
```

$bmGs[i] = min\{s > 0 : Cs(i, s)\ and\ Co(i, s)\ hold\}$

- Cs(i, s) : for each k such that i < k ≤ m, P[k − s] = P[k] or s ≥ k
  - 翻译一下就是这一位往后的所有串都匹配上了，寻找下一个匹配时要寻找这一位往后组成的串s在P中何处再出现；注意，开头i<=0部分可以当作任何字符，因此和P前缀重合也可
- Co(i, s) : if s < i then P[i − s] != P[i]
  - 翻译一下就是满足Cs后还要保证找到的再出现的串之前一个字符不能还是这个失配的字符

```
1   m = P.length
2   COMPUTE-OSUFF(P, Osuff)
3   for i = 1 to m
4         bmGs[i] = m
5   j = 1
6   for i = m − 1 downto 1
7         if Osuff[i] == i
8               while j ≤ m − i
9                     if bmGs[j] == m
10                          bmGs[j] = m − i
11                    j = j + 1
12  for i = 1 to m − 1
13        bmGs[m − Osuff[i]] = m − i
```

注意第6行逆序，第8行范围和j的变化

Ex：GCAGAGAG / ANPANMAN / AT_THAT / AGAGTAGAG

77727471 / 66666381 / 555531 / 555557291

```
BM-MATCHER(T, P)
 1  n = T.length
 2  m = P.length
 3  COMPUTE-BMBC(P, bmBc)
 4  COMPUTE-BMGS(P, bmGs)
 5  s = 0
 6  while s <= n − m
 7        i = m
 8        while P[i] == T[s + i]
 9              if i == 1
10                    print "Pattern occurs with shift" s
11              else i = i − 1
12        s = s + MAX(bmGs[i], bmBc[T[s + i]] − m + i)
```

NP-Completeness

如果不采用一元编码（如k编码为k个0），那么编码对于算法复杂度影响就不是很大

编码多项式相关：多项式时间内编码可以互相转换

Problem vs Language

We can view a decision problem Q as a **language L** over $\sum = \{0, 1\}$.

$$L = \{x \in \sum^* : Q(x) = 1\}.$$

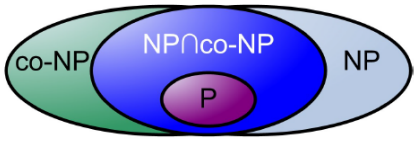
Example

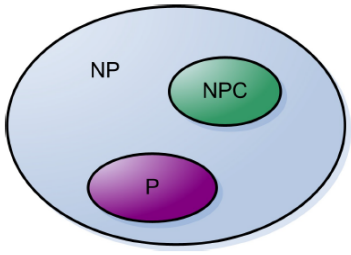
$$\text{PATH} = \{\langle G, u, v, k\rangle : G = (V, E) \text{ is an undirected graph,}$$
$$u, v \in V,$$
$$k \geq 0 \text{ is an interger, and}$$
$$\text{there exists a path from } u \text{ to } v \text{ in G}$$
$$\text{consisting of at most } k \text{ edges}\}.$$

可判定问题decided：对于每一个输入串都能输出0或1

$P = \{L \subseteq \{0,1\}^*$：存在一个可以多项式内判定$L$的算法$\} = \{L \subseteq \{0,1\}^*$：$L$能被一个多项式时间算法接受$\}$

$NP = \{L : \forall L$中元素$x \in \{0,1\}^*, \exists$证书y，多项式时间验证算法A，$|y| = O(|x|^c)$，$A(x, y) = 1\}$





co-P=P，其中co-P定义为$\overline{L} \in P$的L集合

A归约B = 利用B解决A = $A \leq_P B$ = $\exists$多项式计算函数$f, s.t. \forall x \in L_A \Leftrightarrow f(x) \in L_B$

NPC=NP+NP-hard（注意，NP难度问题不一定不是NP）

定理：如果L'是NPC，L满足$L' \leq_P L$，则L'是NP-hard的

常见NPC问题

CURCUIT-SAT：数字逻辑电路

SAT：谓词逻辑

3-CNF：3合取范式$(A_1 \vee A_2 \vee A_3) \wedge (B_1 \vee B_2 \vee B_3)$

CLIQUE团问题：由3-CNF归约而来

VERTEX-COVER顶点覆盖：每个边至少被一个顶点覆盖，问最少多少个顶点；**由CLIQUE归约而来，方法是取补图**

HAM-CYCLE哈密顿回路

TSP旅行商问题：由HAM-CYCLE归约而来

SUBSET-SUM子集合问题：S中有一些正整数，问子集S'之和为t的子集是否存在

IS独立集问题：每条边至多一个端点在集合，问集合的最大元素个数；**由顶点覆盖（原图）或团问题（补图）归约而来**

Approximation Algorithms

近似比例：$max(\frac{C}{C^*}, \frac{C^*}{C}) \leq \rho(n)$，称为$\rho(n)$近似算法

近似模式：$\rho(n) = const$，则为$(1+\epsilon)$近似算法；时间复杂度为$O(n^{2/\epsilon})$也是*多项式时间近似模式*

完全多项式时间近似模式：时间复杂度为n和1/ϵ的多项式，如$O((1/\epsilon)^2 n^3)$

顶点覆盖问题 O(m+n)

算法：任意从E中选出一条边e，将其两个顶点加入C，并删去两个顶点连接的所有边，继续进行查找，直至E删空

结论：运行时间O(V+E)，2近似算法

证明：设最优为C*，简易证法：对于每一条选出的边e，C中含两个点，而C*中至少含一个点；设e_i对应的C中点子集为c_i，C*为从c^*_i对边选出的点累加，一边是C，另一边是C*（由于会删去e两个端点连接的所有边，因此c_i、c^*_i累加并不存在重复的点；反证法可以说明C*无遗漏）；因此由$c_i >= c^*_i$得 |C|>=|C*|

TSP O(n^2)

满足三角不等式的TSP：搞一个最小生成树，然后按照前序遍历的节点顺序走一遍

结论：2近似算法

证明：设最优解为cost*，最小生成树代价cost'，该算法TSP代价为cost，则cost<=2cost'（三角不等式放缩，相当于来去走两遍最小生成树）<=2*cost*

备注：一般TSP不存在多项式时间ρ近似算法（否则P=NP），用哈密顿回路是NPC反证，假设存在近似算法，则对于某哈密顿问题实例G，在完全图中取权重1（如果在G中）/$\rho|V|+1$（如果不在G中）；计算TSP近似算法，若结果小于等于$\rho|V|$则有哈密顿回路，否则无；以此多项式时间解决了NPC问题，那么P=NP

集合覆盖问题 O(最大集合元素个数)

算法：每次取出一个点最多的set，加入最终解集中，随后在其他所有set中删去该set含的点，反复进行

结论：多项式时间的(ln|X| + 1)近似算法

证明：大题思路为，设S_i为算法第i次取得set，并设每次取一个S_i代价为1，设每个通过S_i第一次取到的节点$x \in S_i$平均到的代价为$c_x = 1/|S_i - S_1 \cup \cdots \cup S_{i-1}|$，因此有$|\mathcal{C}| = \sum_{x \in X} c_x \leq \sum_{S \in \mathcal{C}^*} \sum_{x \in S} c_x$

下面证$\sum_{x \in S} c_x \leq H(|S|)$，$\forall S \in \mathscr{F}$，设$u_i = |S - S_1 \cup \cdots \cup S_i|$，有

$\sum_{x \in S} c_x = \sum_{i=1}^{|\mathcal{C}|} \frac{u_{i-1}-u_i}{|S_i - S_1 \cup \cdots \cup S_{i-1}|} \leq \sum_{i=1}^{|\mathcal{C}|} \frac{u_{i-1}-u_i}{|S - S_1 \cup \cdots \cup S_{i-1}|} \leq \sum_{i=1}^{|\mathcal{C}|} \frac{u_{i-1}-u_i}{u_i} = \leq \sum_{i=1}^{|\mathcal{C}|} \sum_{j=u_i+1}^{u_{i-1}} \frac{1}{u_i} \leq \sum_{i=1}^{|\mathcal{C}|} \sum_{j=u_i+1}^{u_{i-1}} \frac{1}{j} \leq \sum_{i=1}^{|\mathcal{C}|} (H(u_{i-1}) - H(u_i)) \leq$

则$|\mathcal{C}| \leq \sum_{S \in \mathcal{C}^*} \sum_{x \in S} c_x \leq \sum_{S \in \mathcal{C}^*} H(|S|) \leq |\mathcal{C}^*| H(max|S|)$

随机化MAX-3-CNF

如果通过随机算法解出的解的期望与实际解存在近似比例，则称是随机化的ρ近似算法

MAX-3-CNF是让为真的子句尽可能多，$x_1, \cdots, x_n$，m个子句，假设每个子句没有一个变量和它的否定，那么随机给所有x_i赋0/1，一个子句为真的期望为7/8，$E[\sum Y_i] = \sum E[Y_i] = m \times 7/8 = 7m/8$，近似比m/(7m/8)=8/7

最小权顶点覆盖问题

选择每个顶点都对应权值，要选择最小权顶点覆盖

用于寻找最小权值顶点覆盖的 0-1 整数规划：

$$\text{minimize} \quad \sum_{v \in V} w(v) x(v) \tag{35.14}$$

条件是

$$x(u) + x(v) \geqslant 1, (u, v) \in E \tag{35.15}$$

$$x(v) \in \{0, 1\}, v \in V \tag{35.16}$$

将其转化为松弛版一般线性规划

在特殊情况下，所有权重 $w(v)$等于 1，这个公式是 NP 难的顶点覆盖问题的最优化版本。假设去掉了 $x(v)\in\{0，1\}$这一限制，并代之以 $0\leqslant x(v)\leqslant 1$，就可以得到如下线性规划，称为**线性规划松弛**：

$$\text{minimize} \quad \sum_{v\in V} w(v)x(v) \tag{35.17}$$

约束是

$$x(u)+x(v)\geqslant 1 \quad ，其中 (u,v)\in E \tag{35.18}$$

$$x(v)\leqslant 1 \quad ，其中\ v\in V \tag{35.19}$$

$$x(v)\geqslant 0 \quad ，其中\ v\in V \tag{35.20}$$

线性规划后，对于每个边上权重大于等于1/2的点，将其加入集合中

证明：设线性规划得到连续的权值是$cost$，经过1/2离散化得到的是cost'，而最优解cost*

显然cost<=cost*<=cost'，而我们对离散化过程分析，

$cost' = \sum_{x(v)\geq 1/2} w(v) \leq \sum_{x(v)\geq 1/2} 2x(v)w(v) \leq \sum_{v\in V} 2x(v)w(v) = 2cost \leq 2cost^*$

因此得到的是2近似算法

子集和问题 O(nlnt/epsilon)

算法：我们要得到一个$(1+\epsilon)$近似算法，为此取$\delta = \epsilon/2n$，并且每次加入一个元素bfs时进行trim剪枝，条件是通过z减去所有大于等于z且小于等于$(1+\delta)$z的y

算法正确性证明：归纳法证明$y^* \in P_i$是第i个元素bfs时最优解，z^*是当前剩余的最优解，有$y^* = (1+\delta)^i z^*$，i=n时有
$z^*/y^* = (1+\frac{\epsilon}{2n})^n \leq e^{\epsilon/2} \leq 1+(\epsilon/2)+(\epsilon/2)^2 \leq 1+\epsilon$

算法复杂度证明：$log_{(1+\epsilon/2n)}t + 2 = \frac{\ln t}{ln(1+\epsilon/2n)+2} \leq \frac{(1+\epsilon/2n)\ln t}{\epsilon/2n} + 2 \leq \frac{3n\ln t}{\epsilon} + 2$（使用了$\frac{x}{1+x} \leq \ln x$），这是每次列表元素数，乘以n即为复杂度，因此是完全多项式时间近似模式

Multithreaded Algorithms

工作时间T_1 持续时间T_∞ 加速比T_1/T_P

定律：$T_1/P \leq T_P$，$T_\infty \leq T_P$

线性加速，如果$T_1/T_P = \Theta(P)$；完美线性加速，如果$T_1/T_P = P$

并行度T_1/T_∞

松弛度$T_1/(T_\infty P)$，松弛度小于1则得不到完美线性加速

贪心调度器的运行时间$T_P \leq T_1/P + T_\infty$（分为完全步和非完全步讨论）；因此有$T_P \leq 2T_P^*$，$\forall T_P^*$

当$P \ll T_1/T_\infty$时，有$T_P \leq T_1/P + T_\infty \approx T_1/P$，而又有$T_1/P \leq T_P$，因此$T_1/T_P \approx P$，近似于完美线性加速