The Design and Analysis of Algorithms

作者：CJSHP（1367456929@qq.com）

Chapter 1 绪论

1.1 什么是算法?

算法

算法是一个满足有穷性、确定性、能行性、输入、输出的计算。

中文名称：周髀算经

英文名称：来自于9世纪波斯数学家花拉子米(al-Khwarizmi)，原为"Algorism"，即"Al-Khwarizmi"的音转，即花拉子米的运算法则。18世纪演变为"Algorithm"

最早的算法：gcd

问题

设Input和Output是两个集合，一个问题是一个关系R $\subset$ Input $\times$ Output，Input称为R的输入集合，Input的每个元素称为R的一个输入，Output称为问题R的输出或结果集合，Output的每个元素称为R的一个结果。**问题定义了输入和输出的关系**

Eg:排序问题

输入集合Input = $\{< a_1, \ldots, a_n >\}$

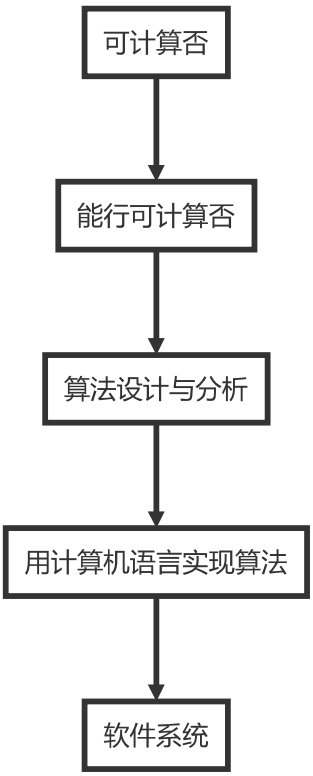
输出集合Output = $\{< b_1, \ldots, b_n > \mid b_1 \leq \ldots \leq b_n\}$

问题SORT = $\{(< a_1, \ldots, a_n >, < b_1, \ldots, b_n >) | < a_1, \ldots, a_n > \in Input, < b1, \ldots, b_n > \in Output, \{a_1, \ldots, a_n\} = \{b_1, \ldots, b_n\}\}$
问题P的一个实例是P的一个二元组。

注意，一个算法面向一个问题，而不是仅求解一个问题的一个或几个实例。

1.2 计算机科学中算法的位置

解决一个计算问题的过程



1.3 算法分析引论

算法正确性：

一个算法是正确的，如果它对每一个输入都最终停止，而且产生正确的输出。

不正确算法：①不停止 ②对所有输入都停止，但对某输入产生不正确结果

近似算法： ①对所有输入都停止 ②产生近似正确的解或产生不多的不正确解

算法正确性证明：①证明算法对所有输入都停止 ②证明对所有输入都产生正确结果

调试程序≠程序正确性证明：程序调试只能证明程序有错，不能证明程序无错误！

插入排序正确性：

①循环不变量：在每次循环的开始，子数组A[1, … , j-1]包含原来数组A[1, … , j-1]但是已经有序

②证明：

-初始化：j = 2, A[1, … , j-1] = A[1, … , 1] = A[1]，已经有序。

-维护：每一层循环维护循环不变量。

-终止：j=n+1，所以A[1, … , j-1] = A[1, … , n] 有序。

算法的复杂性分析

最坏复杂性、最小复杂性、平均复杂性

1.4 算法设计引论

算法设计模式：

暴力搜索，分治法，图搜索与枚举（分支界限、回溯），随机化算法

算法实现方法：

递归与迭代， 顺序、并行与分布式， 确定性与非确定性， 近似求解与精确求解， 量子算法

最优化算法设计方法：

线性规划、动态规划、贪心法、启发式方法

Chapter 2 算法分析的数学基础

2.1 计算复杂性函数的阶

用$\Theta(n^2)$描述插入排序的最坏运行时间

渐进效率：①输入规模非常大 ②忽略低阶项和常系数 ③只考虑最高阶（增长的阶）

同阶函数集合：$\Theta(g(n)) = \{f(n)|\exists c_1, c_2 > 0, n_0, \forall n > n_0, c_1 g(n) \leq f(n) \leq c_2 g(n)\}$称为与$g(n)$同阶的函数集合。

如果$f(n) \in \Theta(g(n))$，$g(n)$与$f(n)$同阶。$f(n) \in \Theta(g(n))$，记作$f(n) = \Theta(g(n))$。

Eg:

1）证明$\frac{1}{2}n^2 - 3n = \Theta(n^2)$.

2）证明$6n^3 \neq \Theta(n^2)$.

低阶函数集合：$O(g(n)) = \{f(n)|\exists c > 0, n_0, \forall n > n_0, 0 \leq f(n) \leq cg(n)\}$称为$g(n)$的低阶函数集合。

$f(n) = \Theta(g(n)) \Rightarrow f(n) = O(g(n))$.（$\Theta$标记强于$O$标记）

$\Theta(g(n)) \subset O(g(n))$.

$an^2 + bn + c = \Theta(n^2) = O(n^2)$. $an + b = O(n^2)$. $n = O(n^2)$

$f(n) \in O(g(n))$，记作$f(n) = O(g(n))$。

如果$f(n) = O(n^k)$，则称$f(n)$是多项式界限的。

高阶函数集合：$\Omega(g(n)) = \{f(n)|\exists c > 0, n_0, \forall n \geq n_0, 0 \leq cg(n) < f(n)\}$称为$g(n)$的低阶函数集合。

$f(n) \in \Omega(g(n))$，记作$f(n) = \Omega(g(n))$。

O标记表示**渐进上界**，Θ标记表示**渐进紧界**，Ω标记表示**渐进下界**。

关于Ω标记：

①用来描述运行时间的最好情况

②对所有输入都正确

③可以用来描述问题的复杂性（最低不会低于）

严格低阶函数：$o(g(n)) = \{f(n)|\forall c > 0, \exists n_0 > 0, \forall n \geq n_0, 0 \leq f(n) < cg(n)\}$.

$f(n) \in o(g(n))$，记作$f(n) = o(g(n))$。

Eg:

1. $2n = o(n^2)$.
2. $2n^2 \neq o(n^2)$.

O可能是紧的可能不是紧的，o肯定不是紧的

严格高阶函数：$\omega(g(n)) = \{f(n) | \forall c > 0, \exists n_0 > 0, \forall n \geq n_0, 0 \leq cg(n) < f(n)\}$.

$f(n) \in \omega(g(n))$，记作$f(n) = \omega(g(n))$。

表示不紧的下界

有结论$f(n) = \omega(g(n)) \Leftrightarrow g(n) = o(f(n))$.

渐进符号的性质

①传递性：$f(n) = \Theta(g(n)), g(n) = \Theta(h(n)) \Rightarrow f(n) = \Theta(h(n))$.

②自反性：$f(n) = A(f(n)), A \in \{O, \Theta, \Omega\}$.

③对称性：$f(n) = \Theta(g(n)) \Leftrightarrow g(n) = \Theta(f(n))$.

④反对称性：$f(n) = O(g(n)) \Leftrightarrow g(n) = \Omega(f(n))$. $f(n) = o(g(n)) \Leftrightarrow g(n) = \omega(f(n))$.

并非所有函数都是可比的，如n和n^{1+sinn}.

2.2 和式的估计与界限

线性和、级数

级数:

$$\sum_{i=1}^{n} i = \frac{n(n+1)}{2} = \Theta(n^2).$$

$$\sum_{k=0}^{k} x^k = \frac{x^{n+1} - 1}{x - 1}.$$

$$\sum_{k=0}^{\infty} x^k = \frac{1}{1-x} \quad |x| < 1.$$

$$H_n = \sum_{k=1}^{n} \frac{1}{k} = lnn + O(1)\ .$$

$$\sum_{k=1}^{n} (a_k - a_{k-1}) = a_n - a_0.$$

$$\sum_{k=1}^{n-1} \frac{1}{k(k+1)} = \sum_{k=1}^{n-1} (\frac{1}{k} - \frac{1}{k-1}) = 1 - \frac{1}{n}.$$

$$lg(\prod_{k=1}^{n} a_k) = \sum_{k=1}^{n} lg\ a_k.$$

Eg.

1. 证明$\sum_{k=0}^{n} 3^k = O(3^n)$.
2. 证明$\sum_{k=1}^{n} k = \Theta(n^2)$.
3. $\sum_{k=1}^{n} k \leq \sum_{k-1}^{n} n = n^2$.
4. $\sum_{k=1}^{n} a_i \leq n \times max\{a_k\}$.
5. 设$\forall\ k \geq 0,\ a_{k+1}/\ a_k \leq r < 1$，求$\sum_{k=0}^{n} a_k$的上界。
6. 求$\sum_{k=1}^{\infty}(k/3^k)$的界。
7. 使用分裂和的方法求$\sum_{k=1}^{n} k$的下界。
8. 求$\sum_{k=0}^{\infty} \frac{k^2}{2^k}$的上界。
9. 求$H_n = \sum_{k=1}^{n} \frac{1}{k}$的上界。
10. 如果$f(k)$单调递增，则$\int_{m-1}^{n} f(x)dx \leq \sum_{k=m}^{n} f(k) \leq \int_{m}^{n+1} f(x)dx$.
11. 如果$f(k)$单调递减，则$\int_{m}^{n+1} f(x)dx \leq \sum_{k=m}^{n} f(k) \leq \int_{m-1}^{n} f(x)dx$.
12. $\sum_{k=1}^{n} \frac{1}{k} \geq \int_{1}^{n+1} \frac{dx}{x} = ln(n+1)$，$\sum_{k=2}^{n} \frac{1}{k} \leq \int_{1}^{n} \frac{dx}{x} = lnn$.

2.3 递归方程

求解递归方程的三个主要方法：①猜想+数学归纳法 ②把方程转化为一个和式，然后用估计和的方法求解 ③Master定理：求解形如$T(n) = aT(n/b) + f(n)$的递归方程。

求解$T(n) = 2T(\frac{n}{2} + 17) + n$：联想已知的$T(n)$。

求解$T(n) = 2T(\frac{n}{2}) + n$：猜测上下界。

求解$T(n) = T(\lfloor n/2 \rfloor) + T(\lceil n/2 \rceil) + 1$：从中减去低阶项。

求解$T(n) = 2T(\lfloor n/2 \rfloor) + n$：数学归纳法可能出错。

求解$T(n) = 2T(\sqrt{n}) + lgn$：变量替换。

迭代方法：循环展开递归方程，把递归方程展开成和式，然后用求和解决。

求解$T(n) = 3T(\lfloor n/4 \rfloor) + n$.

Master定理：适用$T(n) = aT(n/b) + f(n)$型方程，$a \geq 1, b \geq 0$是常数，$f(n)$是正函数。

①$f(n) = O(n^{log_b a - \epsilon})$，$\epsilon > 0$是常数，则$T(n) = \Theta(n^{log_b a})$。

②$f(n) = \Theta(n^{log_b a})$，则$T(n) = \Theta(n^{log_b a} logn)$。

③$f(n) = \Omega(n^{log_b a + \epsilon})$，$\epsilon > 0$是常数，且对于所有充分大的$n$有$af(n/b) \leq cf(n)$，$c \geq 1$是常数，则$T(n) = \Theta(f(n))$。

求解$T(n)=9T(n/3)+n$。

求解$T(n)=T(2n/3)+1$。

求解$T(n)=3T(n/4)+nlogn$。

求解$T(n)=2T(n/2)+nlogn$。

Chapter 3 分治法

3.1 分治法

设计：三个阶段：划分、求解、合并

分析：建立递归方程、求解。

递归方程建立方法：

① 设输入大小为n，$T(n)$为时间复杂性。

② 当$n<c$，$T(n)=\Theta(1)$。

③ 划分阶段的时间复杂性

1. 划分阶段为n个子问题。
2. 每个子问题大小为n/b。
3. 划分时间可直接得到$D(n)$。

④ 递归求解阶段的时间复杂性

```
1. 递归调用。
```

2. 求解时间 = $aT(n/b)$。

⑤ 合并阶段的时间复杂性

时间可直接得到 = $C(n)$。

总之，

$$\begin{aligned} T(n) &= \Theta(1) \qquad if \quad n<c \\ &= aT(n/b)+D(n)+C(n) \quad if \quad n\geq c \end{aligned}$$

3.2 分治法的简单实例

$O(n^2)$简单分治

$X=A\times 2^{n/2}+B$，$Y=C\times 2^{n/2}+D$。

$X\times Y=AC\times 2^n+(AD+BC)\times 2^{n/2}+BD$。

1. 划分产生A、B、C、D。
2. 计算$n/2$位乘法AC、AD、BC、BD
3. 计算$AD+BC$
4. AC左移n位，$(AD+BC)$左移$n/2$位。
5. 计算XY。

$T(n)=4T(n/2)+\Theta(n)\Rightarrow T(n)=\Theta(n^2)$。

$O^{1.59}$**分治**：

$X=A\times 2^{n/2}+B$，$Y=C\times 2^{n/2}+D$。

$X\times Y=AC\times 2^n+(AD+BC)\times 2^{n/2}+BD$。

$AD+BC=(A+B)(C+D)-AC-BD$。

1. 划分产生A、B、C、D。
2. 计算$A+B$、$C+D$。
3. 计算$n/2$位乘法AC、BD、$(A+B)(C+D)$。
4. 计算$(A+B)(C+D)-AC-BD$。
5. AC左移n位，$(A+B)(C+D)-AC-BD$左移$n/2$位。
6. 计算XY。

$T(n)=3T(n/2)+O(n)\Rightarrow T(n)=O(n^{log3})=O(n^{1.59})$。

最大值最小值：

类似线段树。

如果中间只有一个数，返回。如果有两个数，返回。

区间划分

3.3 元素选取问题的简单线性时间算法

中位数问题：

$$
\begin{aligned}
&T(n)\leq\Theta(1)\qquad if\quad n\leq c\\
&T(n)\leq T(\lfloor n/5\rfloor)+T(7n/10+6)+\Theta(n)\qquad if\quad n>c
\end{aligned}
$$

$T(n)=O(n)$。

3.4 快速傅里叶变换

FFT

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

Chapter 4 动态规划

4.1 动态规划的原理

分治的问题

1. 子问题相互独立
2. 如果子问题不相互独立，公共子问题会被重复计算，效率很低。

优化问题

1. 给定一组约束条件和一个代价函数，在解空间中搜索具有最小或最大价值的优化解。

2. 很多优化问题可分为多个子问题，子问题相互关联，子问题的解被重复使用。

动态规划的特点

```
1. 把原始问题划分为一系列子问题
```

2. 求解每个子问题仅一次，并将其结果保存在一个表中，以后用到的时候直接存取，不重复计算，节省计算时间。
3. 自底向上计算。

适用范围

一类优化问题，可分为多个相关子问题，子问题的解被重复使用。

使用动态规划的条件

优化子结构：

1. 一个问题的优化解包括了子问题的优化解时，这个问题具有优化子结构。
2. 缩小子问题集合，只需那些优化问题中包含的子问题，降低实现复杂性。
3. 优化子结构使得我们能自下而上的完成求解过程。

重叠子结构：

在问题的求解过程中，很多子问题的解将被重复使用。

设计步骤

1. 分析优化解的结构
2. 递归的定义最优解的代价
3. 自底而上的计算优化解的代价保存之，并获取构造最优解的信息
4. 根据构造最优解的信息构造最优解

4.2 矩阵乘法问题

具有优化子结构：**问题的优化解包括子问题优化解。**

具有子问题重叠性

```
Matrix-Chain-Order(p)
n = length(p) - 1;
FOR i = 1 TO n DO
    m[i,i] = 0;
FOR l = 2 TO n DO
    FOR i = 1 TO n-l+1 DO
        j = i+l-1;
        m[i,j] = inf;
        FOR k = i TO j-1 DO
            q = m[i,k] + m[k+1,j] + p[i-1]*p[k]*p[j];
            if (q < m[i][j]) m[i][j] = q;
return m.
```

获取构造最优解的信息：`s[i][j]`用来记录`a[i]...a[j]`的最优划分解在`a[k]`与`a[k+1]`之间。

`s[i,s[i][j]]`、`s[s[i][j]+1,j]`来递归记录，调用后序遍历的递归函数即可输出。

时间复杂度$O(n^3)$，空间复杂度$O(n^2)$。

4.3 最长公共子序列问题

`dp[i][j] = dp[i-1][j-1] + 1, a[i] == b[j]`

`= max(dp[i-1][j],dp[i][j-1]), a[i] != b[j]`

4.4 0-1背包问题

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

4.5 最优二分搜索树（BST）

Optimal BST

`W[i][i-1] = q[i-1]`

`W[i][j] = W[i][j-1] + p[j] + q[j]`

`E[i][j] = min(E[i][j],E[i][r-1] + E[r+1][j] + W[i][j])`

Chapter 5 贪心算法

5.1 贪心法的基本原理

基本思想:

求解最优化问题的算法包含一系列步骤，每一步都有一组选择，作出在当前看来最好的选择，希望通过作出局部优化选择达到全局优化选择。

贪心算法产生优化解的条件

贪心选择性、优化子结构

贪心选择性

如果一个问题的全局优化解能通过局部优化选择得到，则称该问题具有贪心选择性。

dp和贪心的区别

dp：优化子结构、子问题重叠性、子问题空间小

贪心：优化子结构、贪心选择性

贪心算法正确性证明方法

1. 证明算法所求解的问题具有**优化子结构**
2. 证明算法所求解的问题具有**贪心选择性**
3. 证明算法确实按照贪心选择性进行局部优化选择

5.2 任务安排问题

按结束时间升序排序。

5.3 哈夫曼编码问题

贪心，合并果子。

合并石子是区间dp。

5.4 最小生成树问题

Prim：$O(n^2)$或$O(mlogm)$。

Kruskal：排序+并查集，$O(mlogm)$。

Chapter 6 搜索策略

6.1 暴力美学：搜索漫谈

布尔表达式可满足问题：DFS **八数码**：BFS **Hamilton环问题**：DFS

6.2 深度优先与广度优先

八数码：BFS **子集合和问题**：DFS **Hamilton环问题**：DFS

6.3 搜索的优化

爬山法：启发式测度来排序节点扩展的顺序。（加栈，启发式测度由大到小入栈，局部优化）

Best-First：结合DFS和BFS，根据估价函数选择最小评价值节点扩展（用堆BFS，全局优化）

分支界限：组合优化问题，发现优化解的一个界限，缩小解空间，提高求解效率。

多阶段图搜索问题：转换成树，然后爬山。

6.4 剪枝方法论与人员安排问题

拓扑序列树：表示可能的各种拓扑序列。

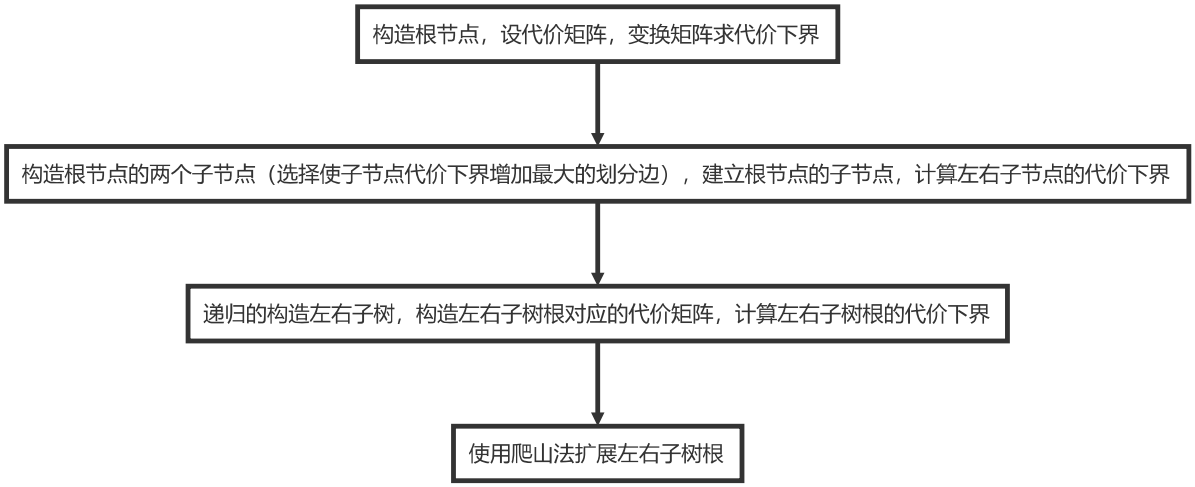
求解问题的分支界限搜索算法：

1. 计算解的代价的下界。
2. 分支界限搜索（爬山法）。

6.5 旅行商问题

问题定义：从任意节点开始，经过每个节点一次，最后返回起点的最短路径。

求解方法：转换为树搜索问题，所有解集合变成树根，权值由代价矩阵使用行列消除得到，用爬山法递归划分解空间，得到二叉树。并使用分支界限搜索算法。



Eg：见PPT。

6.6 A*算法

分支界限搜索是为了剪掉不能达到优化解的分支，关键是**界限。**

A*算法的核心是告诉我们，某些情况下，我们得出的解一定是优化解，于是算法可以退出。它试图尽早找出优化解，经常使用Best-First策略求解优化问题。

关键：代价函数

对于任意节点n：

$g(n)$表示从树根到n的代价

$h^*(n)$表示从n到目标节点的优化路径的代价。

$f^*(n) = h^*(n) + g(n)$表示节点n的代价。

但是我们不知道$h^*(n)$，也就不知道$f^*(n)$。

所以我们用$h(n)$估计$h^*(n)$，且$h(n) \leq h^*(n)$恒为真，$h(n)$也就恒小于$h^*(n)$，定义它为n的代价。

最短路问题的A*求解

破禁策略

停止规则

Chapter 7 平摊分析

7.1 平摊分析原理

基本思想：

在平摊分析中，执行一系列数据结构操作所需要的时间是通过对执行的所有操作求平均值而得出的。

对一个数据结构要执行一系列操作，有的代价很高，有的代价一般，有的代价很低。将总的代价平摊到每个操作上就是**平摊代价**，平摊的过程不涉及概率，也不同于平均情况分析。

方法：聚集方法、会计方法、势能方法

7.2 聚集方法

$cost = \frac{\sum_{i=1}^{n} t_i}{n}$，操作序列中的每个操作都被赋予相同的代价，不管操作的类型。

Eg:

栈操作:

每个操作的代价是$O(1)$，n个就是$\Theta(n)$。

增加`multipop(n,k)`方法后，分析方式变化，`POP`不会超过`PUSH`，所以$T(n) \leq 2n$，均摊之后还是$O(1)$。

没有使用任何概率！

二进制计数器:

输入x，输出$(x+1) \ \% \ 2^k$。

每次+1的代价和被改变值的位数成线性关系，粗略的讲，+1最多改变k位，也就是说n次+1的代价最多是$O(nk)$。精确一些，低位的第i位每隔2^i次发生一次变化，所以改变$\sum_{i=0,i+=2}^{\lceil logn \rceil} < 2n$。

7.3 会计方法

一个操作中有不同类型的操作，不同类型的操作代价各不相同，于是我们为不同的操作分配不同的平摊代价。

平摊代价可能比实际代价大，也可能比实际代价小。

操作被执行的时候，支付了平摊代价。

如果平摊代价比实际代价大，多出的一部分作为存款附加在数据结构的具体数据对象上。

如果平摊代价比实际代价小，平摊代价及数据对象上的存款用来支付实际代价。

只要我们能保证：在任何操作序列上，存款的总额非负。则，所有操作平摊代价的总和就是实际代价总和的上界。

于是：我们在各种操作上定义平摊代价使得任意操作序列上存款总量是非负的，将操作序列上平摊代价求和即可得到这个操作序列的复杂度上界。

只要我们操作序列合理，存款的总额保证是非负的。于是所有操作的平摊代价总和就是操作序列代价总和的上界。

栈操作：长度为n的操作序列中，`PUSH`至多n次，所以$T(n) \leq 2n$。

二进计数器：显然操作序列的代价和翻转次数成正比。

定义：0-1翻转代价为2, 1-0翻转代价为0.

任何操作序列，存款余额是计数器中1的个数，非负。所以，所有翻转操作的平摊代价的和就是操作序列代价的上界。

对每次+1操作，找到左起的第一个0，翻转支付平摊代价2，将0之前的所有1翻转，支付平摊代价0。对这个+1操作，支付了平摊代价2。

对于长度为n的+1操作序列，支付的平摊代价和是$2n$。

所以，这样的操作序列复杂度上界是$2n$。

7.4 势能方法

会计方法中，如果操作的平摊代价比实际要大，我们就把余额和具体的数据对象相关联。如果我们把这些余额和整个数据结构相关联，所有的余额之和就构成了数据结构的**势能**。

如果平摊代价大于实际代价，势能增加，反之，要用数据结构的势能支付实际代价，势能减少。

势能的定义：对于一个数据结构执行n个操作，对操作i：

实际代价c_i将数据结构从D_{i-1}变成了D_i。

势函数ϕ把每个数据结构D_i映射成一个实数$\phi(D_i)$

平摊代价c_i'定义为：$c_i' = c_i + \phi(D_i) - \phi(D_{i-1})$。

n个操作的平摊代价为：

$$\begin{aligned}\sum_{i=1}^{n} c_i' &= \sum_{i=1}^{n}(c_i + \phi(D_i) - \phi(D_{i-1})) \\ &= \sum_{i=1}^{n} c_i + \phi(D_n) - \phi(D_0)\end{aligned}$$

于是势函数ϕ满足$\phi(D_n) \geq \phi(D_0)$，则总的平摊代价就是总的实际代价的一个上界。

在实践中，我们定义$\phi(D_0) = 0$，然后证明$\phi(D_i) \geq 0$。

平摊代价依赖于所选的势函数ϕ，不同的势函数可能会产生不同的平摊代价，但它们都是实际代价的上界。

栈操作：

$\phi(D)$ = 栈D中对象的个数

初始栈D_0为，$\phi(D_0) = 0$

因为栈中的对象数始终非负，第i个操作之后的栈D_i满足$\phi(D_i) \geq 0 = \phi(D_0)$。

于是： 以ϕ表示的n个操作的平摊代价的总和就表示了实际代价的一个上界

作用于包含s个对象的栈上的栈操作的平摊代价：

如果第i个操作是`PUSH`操作，实际代价$c_i = 1$，势差：$\phi(D_i) - \phi(D_{i-1}) = (s+1) - s = 1$，所以平摊代价$c_i' = c_i + \phi(D_i) - \phi(D_{i-1}) = 1 + 1 = 2$。

如果第i个操作是`MULTIPOP(s,k)`操作，弹出了$k' = min(k, s)$个对象，实际代价$c_i = k'$，势差：$\phi(D_i) - \phi(D_{i-1}) = -k'$，所以平摊代价$c_i' = c_i + \phi(D_i) - \phi(D_{i-1}) = 0$。

如果第i个操作是 POP 操作，实际代价$c_i = 1$，势差：$\phi(D_i) - \phi(D_{i-1}) = -1$，所以平摊代价$c'_i = c_i + \phi(D_i) - \phi(D_{i-1}) = 0$。

平摊分析：每个栈操作的平摊代价都是$O(1)$，n个操作序列的总平摊代价就是$O(n)$，因为$\phi(D_i) \geq \phi(D_0)$，n个操作的总平摊代价就是总的实际代价的一个上界，即n个操作的最坏情况代价为$O(n)$。

二进计数器：

$\phi(D_)$ = 计数器D中1的个数

计数器初始状态D_0中1的个数为0，$\phi(D_0) = 0$

因为栈中的对象数始终为负，第i个操作之后的栈D_i满足$\phi(D_i) \geq 0 = \phi(D_0)$

于是，n个操作的平摊分析的总和就表示了实际代价的一个上界。

第i次+1操作的平摊代价

设第i次+1操作对t_i个位进行了置零

至多将一个位置置1

该操作的实际代价：$c_i = t_i + 1$

在第i次操作后计数器中1的个数为$b_i \leq b_{i-1} - t + 1$

势差：$\phi(D_i) - \phi(D_{i-1}) \leq (b_{i-1} - t_i + 1) - b_{i-1} = 1 - t_i$

平摊代价：$c'_i = c_i + \phi(D_i) - \phi(D_{i-1}) \leq (t_i + 1) + (1 - t_i) = 2$

计数器状态初始为0的平摊分析：

每个栈操作的平摊代价都是$O(1)$，n个就是$O(n)$，因为$\phi(D_i) \geq \phi(D_0)$，n个操作的总平摊代价就是总的实际代价的一个上界，即n个操作的最坏情况代价为$O(n)$。

最开始有b_0个1，n次之后就有b_n个1，所以总代价为$\sum_{i=1}^{n} c_i = \sum_{i=1}^{n} 2 - b_n + b_0 = 2n - b_n + b_0$。

如果我们执行了至少$n = \Omega(k)$次+1操作，则无论计数器中包含什么样的初值，总实际代价都是$O(n)$。

7.5 动态表操作的平摊分析

利用平摊分析证明插入和删除操作的平摊代价为$O(1)$，即使当它们引起了表的扩张和收缩时具有较大的实际代价。

非空表T的装载因子$\alpha(T) = T$存储的对象数（表大小）

- 空表的大小为0，装载因子为1
- 如果动态表的装载因子以一个常数为下界，则表中未使用的空间就始终不会超过整个空间的一个常数部分。

设T表示一个表：

- table[T]是一个指向表示表的存储块的指针
- num[T]包含了表中的项数
- size[T]是T的大小
- 开始时，num[T] = size[T] = 0

向表中插入一个数组元素的时候，分配一个包含比原表更多的槽的新表，再将原表中的各项复制到新表中去

一种常见的启发式技术是分配一个比原表大一倍的新表，如果只对表执行插入操作，则表的装载因子总是至少为1/2，这样浪费掉的空间就始终不会超过表总空间的一半。

- **粗略分析**：第i次操作的代价c_i
  - 如果$i = 1, c_i = 1$。
  - 如果表有空间，$c_i = 1$。
  - 如果表是满的，$c_i = i$。

  如果一共有n次操作，最坏情况下总的代价上界为$O(n^2)$。但是这个界并不精确，因为执行n次的过程中不常常包括扩张表的过程，仅当$i - 1 = 2^k$时第i次操作才会引起一次表的扩张。
- **聚集分析**：第i次操作的代价c_i
  - 如果$i = 2^m$，$c_i = i$。
  - 否则$c_i = 1$。

  所以n次操作的总代价是$\sum_{i=1}^{n} c_i \leq n + \sum_{j=0}^{\lfloor logn \rfloor} 2^j < n + 2n = 3n$。

  每一操作的平摊代价是$3n/n = 3$。
- **会计法分析**：每次执行的平摊代价是3
  - 支付基本操作的实际代价
  - 作为自身的存款
  - 存入表中第一个没有存款的数据上

  当发生表的扩张时，数据的复制代价由数据上的存款来支付。（任意时刻存款总和非负）

  初始为空的表上n次TABLE_INSERT的平摊代价总和是$3n$。
- **势能法分析**：刚补充完，$\phi(T) = 0$；表满时，$\phi(T) = size(T)$。
  - $\phi(T) = 2 * num[T] - size[T]$。
  - 势能函数满足要求并且：由于$num[T] \geq size[T]/2$，并且$\phi(T) \geq 0$，那么n次TABLE_INSERT操作的总的平摊代价就是总的实际代价的一个上界。
  - 不管是否发生扩张，$c_i = 3$

  初始为空的表上n次TABLE_INSERT的平摊代价总和为$3n$。

表的扩张、收缩：

理想情况下，我们期望表具有一定的丰满度，并且表的操作序列是线性的。

根据表的扩张策略，很自然的想到下满的收缩策略

但是表的装载因子<1/2时，收缩表为原表的一半

$n = 2^k$，下面的一个长度为n的操作序列：前$n/2$个操作是插入，后跟IDIDDI.....

每次扩张和收缩的代价为$O(n)$，共有$O(n)$次扩张或收缩

总代价为$O(n^2)$，而每一次的平摊代价为 $O(n)$。

但是可以改善

当向满的表中插入一项时，还是将表扩大一倍，但是删除表中的一项时而引起表不如$1/4$满时，我们就把表缩小为原来的一半。这样，扩张和收缩过程都使得表的装载因子变为$1/2$，但是表的装载因子下界是$1/4$。

势函数法：操作序列过程中的表T，势能总是非负的，这样才能保证一列操作的总平摊代价为其实际代价的一个上界。表的扩张和收缩要消耗大量的势，这样我们的势就满足：

1. num(T) = size(T) / 2时，势最小
2. num(T)减小时，势增加直到收缩
3. num(T)增大时，势增加直到扩充

势函数的定义：

$$\begin{aligned}\Phi(T) &= 2num[T] - size[T], \quad \alpha(T) \geq 1/2 \\ &= size[T]/2 - num[T], \quad \alpha(T) < 1/2\end{aligned}$$

空表的势为0，且势总是非负的。这样，以ϕ表示的一系列操作的总平摊代价即为其实际代价的一个上界。

势函数的某些性质：

- 当装载因子为1/2时，势为0
- 当装载因子为1时，有num[T] = size[T]，这就意味着，$\phi(T)$ = num[T]。这样当因为插入一项而引起一次扩张时，就可以用势来支付其代价。
- 当装载因子为1/4时，size[T] = 4*num[T]，这就意味着，$\phi(T) = num[T]$。因而当删除一项而引起一次收缩时就可以用势来支付其代价。
- 第i次操作的平摊代价：$c_i' = c_i + \phi(T_i) - \phi(T_{i-1})$
- 第i次操作是TABLE_INSERT：未扩张，$c_i' \leq 3$
- 第i次操作是TABLE_INSERT：扩张，$c_i' \leq 3$
- 第i次操作是TABLE_DELETE：未收缩，$c_i' \leq 3$
- 第i次操作是TABLE_DELETE：收缩，$c_i' \leq 3$
- 所以作用于同一动态表上的n个操作的实际时间为$O(n)$

Chapter 8 图论算法

8.1 网络流

网络、流、余图、增广路径、增广。

FF算法(Ford-Fulkerson)：复杂度$O(mc)$。

加速：最短/大增广路径算法

最大流最小割

8.2 最大二分匹配

超级源、超级汇

8.3 最短路

Bellman-Ford、Dijkstra、Floyd（中序遍历输出路径）

Edmonds算法：无向图中的匹配

Chapter 9 字符串算法

9.1 精确字符串匹配

暴力搜索$O(nm)$

指纹：Hash，$O(n)$**，最坏**$O(nm)$

Rabin-Karp，$O(n)$

BMH最坏$O(nm)$**、KMP最坏** $O(n+m)$

9.2 字符串查找数据结构

BST、Trie树、紧缩Trie树（单词匹配）**、Patrica Trie、后缀树、B树**

9.3 近似字符串匹配

略
