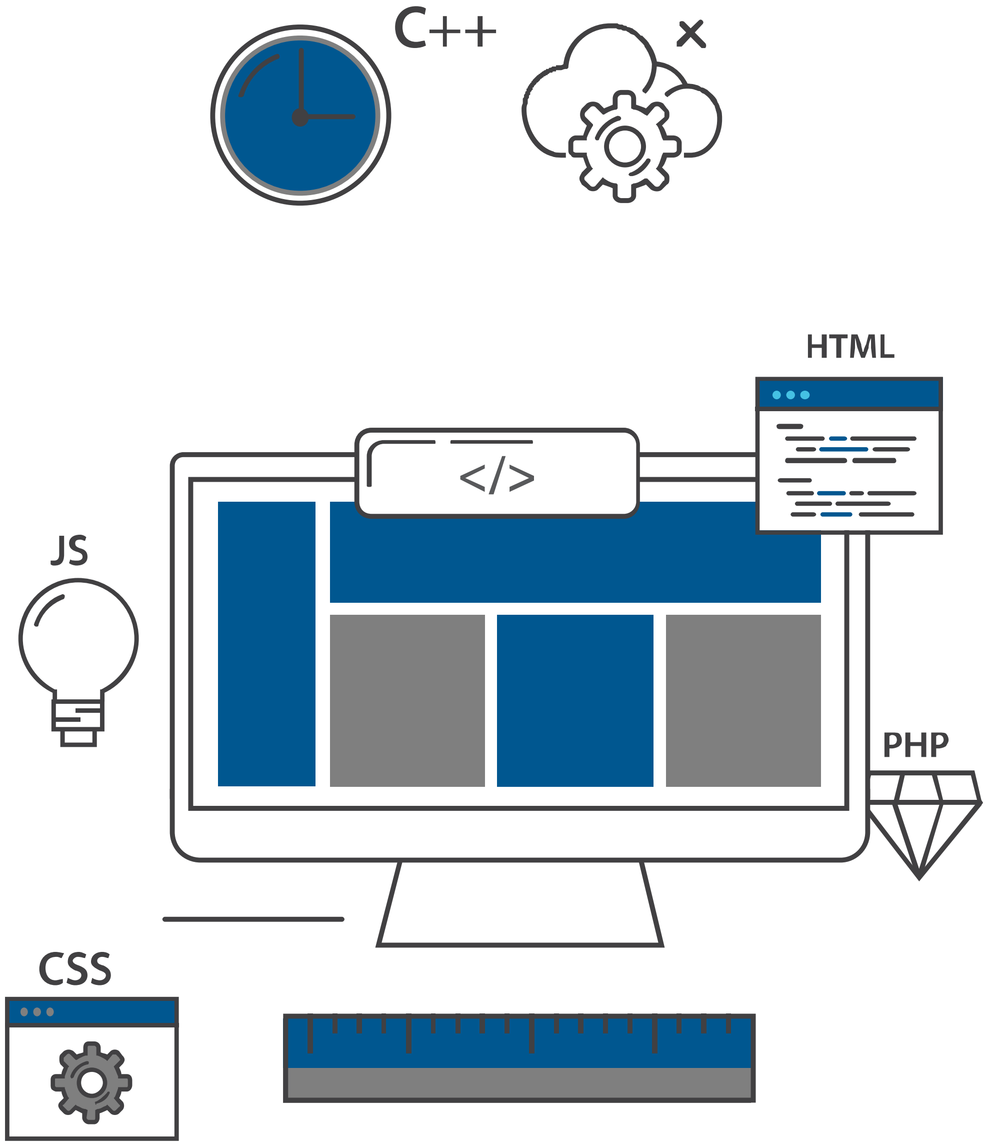
第二章 算法分析基础

李 翔
https://implus.github.io/



算法分析

- 什么叫一个算法好，运行得有效率?

- 概念设计：效率

定义效率

- 提出效率定义(1) 当实现一个算法时，如果它在真实的输入实例上运行的快，那么这个算法是有效的。

- 寻找效率的定义：与平台无关，实例无关，并且随着输入规模的增长是可以预测的

定义效率

- 提出效率定义(2) 在分析的层次上，如果一个算法与蛮力搜索(Brute Force)比较，**最坏情况**下达到质量上更好的性能，就说这个算法是有效的。

定义效率

Table 2.1 The running times (rounded up) of different algorithms on inputs of increasing size, for a processor performing a million high-level instructions per second. In cases where the running time exceeds 10^{25} years, we simply record the algorithm as taking a very long time.

	n	$n \log_2 n$	n^2	n^3	1.5^n	2^n	$n!$
$n = 10$	< 1 sec	< 1 sec	< 1 sec	< 1 sec	< 1 sec	< 1 sec	4 sec
$n = 30$	< 1 sec	< 1 sec	< 1 sec	< 1 sec	< 1 sec	18 min	10^{25} years
$n = 50$	< 1 sec	< 1 sec	< 1 sec	< 1 sec	11 min	36 years	very long
$n = 100$	< 1 sec	< 1 sec	< 1 sec	1 sec	12,892 years	10^{17} years	very long
$n = 1{,}000$	< 1 sec	< 1 sec	1 sec	18 min	very long	very long	very long
$n = 10{,}000$	< 1 sec	< 1 sec	2 min	12 days	very long	very long	very long
$n = 100{,}000$	< 1 sec	2 sec	3 hours	32 years	very long	very long	very long
$n = 1{,}000{,}000$	1 sec	20 sec	12 days	31,710 years	very long	very long	very long

定义效率

一个算法被称为是多项式时间的如果满足如下的性质：当算法输入的规模增长时，算法的运行时间是多项式有界的。也就是：存在常数c>0, d>0, 使得对于每一个问题输入的规模N, 算法的运行都能够在 cN^d 步骤内完成。

定义效率

- 提出效率定义(3) 如果一个算法有多项式运行时间，称为这个算法是有效的。

- 来自于数学形式和经验证据
- 但并不绝对反映真实的运行时间
- 符合实际情况的**合理近似**

定义效率

注记:

- 这种定义有利于一般性的研究
- 这种定义能够清楚的表达：对某个问题不存在有效的算法
- ✓ $6.02 \times 10^{23} \times N^{20}$ 也是多项式时间,但是在实际上是不可行的。
- ✓ 实际中，大多数开发出的多项式时间的算法中幂次指数，系数，常数项都比较小。
- ✓ 一些最坏情形是指数阶复杂度的算法也可能广泛使用，这种情形极少出现。
- ✓ 能够改进指数阶穷举算法的要点在于发现问题内在的一些特殊结构。

增长的渐进阶

- 为什么需要这样一个概念？

✓ 得到一个准确的界是很困难的；

✓ 我们的目标是识别类似行为算法的大类，按照**粗粒度**进行分析；

✓ 讨论算法执行的步数可能依赖于所使用的语言，因此复杂度的多项式系数会不一样。

复杂度符号

- 因为上述原因，希望以不受常数因子，低项影响的方式表达运行时间的增长率。

- $O, \Omega, \because$

复杂度符号

- **Upper bounds.** $T(n)$ is $O(f(n))$ if there exist constants $c > 0$ and $n_0 \geq 0$ such that for all $n \geq n_0$ we have $T(n) \leq c \cdot f(n)$.
- **Lower bounds.** $T(n)$ is $\Omega(f(n))$ if there exist constants $c > 0$ and $n_0 \geq 0$ such that for all $n \geq n_0$ we have $T(n) \geq c \cdot f(n)$.
- **Tight bounds.** $T(n)$ is $\Theta(f(n))$ if $T(n)$ is both $O(f(n))$ and $\Omega(f(n))$.

复杂度符号

- Ex: $T(n) = 32n^2 + 17n + 32.$

复杂度符号

- Ex: $T(n) = 32n^2 + 17n + 32$.
  - $T(n)$ is $O(n^2)$, $O(n^3)$, $\Omega(n^2)$, $\Omega(n)$, and $\Theta(n^2)$.
  - $T(n)$ is not $O(n)$, $\Omega(n^3)$, $\Theta(n)$, or $\Theta(n^3)$.

复杂度性质

- 设f和g是两个函数，$\lim_{n\to\infty}\frac{f(n)}{g(n)}$存在，并且等于某个常数c>0, 那么$f(n)=\Theta(g(n))$.

- 证明：根据极限的定义,存在n_0,对所有的n>= n_0,f(n)<=2cg(n)从而推出f(n)=O(g(n)). 类似的f(n)= Ω (g(n)). 根据定义可知命题成立。

极限的定义

zh.wikipedia.org/wiki/极限_(数学)

形式上，我们可以定义：

$$\lim_{x \to \infty} f(x) = L$$

为

$$(\forall \epsilon > 0)(\exists \delta > 0)(\forall x \in D_f)[(\delta < x) \Rightarrow (|f(x) - L| < \epsilon)]$$

复杂度性质

- 传递性
  - If f = $O(g)$ and g = $O(h)$ then f = $O(h)$.
  - If f = $\Omega(g)$ and g = $\Omega(h)$ then f = $\Omega(h)$.
  - If f = $\Theta(g)$ and g = $\Theta(h)$ then f = $\Theta(h)$.

复杂度性质

- 传递性
  - If $f = O(g)$ and $g = O(h)$ then $f = O(h)$.
  - If $f = \Omega(g)$ and $g = \Omega(h)$ then $f = \Omega(h)$.
  - If $f = \Theta(g)$ and $g = \Theta(h)$ then $f = \Theta(h)$.

本质是：不等号的传递性!

复杂度性质

- 可加性
  - If f = $O(h)$ and g = $O(h)$ then f + g = $O(h)$.
  - If f = $\Omega(h)$ and g = $\Omega(h)$ then f + g = $\Omega(h)$.
  - If f = $\Theta(h)$ and g = $\Theta(h)$ then f + g = $\Theta(h)$.

复杂度性质

- 可加性
  - If f = $O(h)$ and g = $O(h)$ then f + g = $O(h)$.
  - If f = $\Omega(h)$ and g = $\Omega(h)$ then f + g = $\Omega(h)$.
  - If f = $\Theta(h)$ and g = $\Theta(h)$ then f + g = $\Theta(h)$.

本质是：不等号的可加性!

复杂度性质

- 命题 令f是一个d阶多项式， $a_0 + a_1n + \ldots + a_dn^d$, $a_d>0$, 那么$f= \Theta(n^d)$.
- 命题 对于任何正整数， $O(\log_a n) = O(\log_b n)$.

复杂度性质

- 命题 对于 $x > 0$, $\log n = O(n^x)$.

对数函数增长比多项式增长慢!

- 命题 对每个$r>1$和每个$d>0$, $n^d = O(r^n)$.

多项式增长比指数函数慢!

一般运行时间的描述

- 对日常中我们经常碰到的算法问题，按照阶数的大小进行分类
- O(logn)
- O(n),O(nlogn)
- O(n^2),O(n^3),O(n^k)
- O(a^n),O(n!)

亚线性时间(O(logn)) ******

- 例子：给定一个排好序的含有n个数的数组A，我们想确定一个给定的数p是否属于这个数组。
- 算法要点:每次比较p与剩余集合中间元素的值
- k次试探以后，剩余的区间大小为：$\left(\frac{1}{2}\right)^k n$

所以选择$k=\log_2(n)$，剩余集合的大小减少到一个常数，最后可在常数时间内直接搜索。

二分查找

- 给定一个不降序整数数组A（长度为N>0），例如 −1，0，1，1，4，6，6，6，10，请用log(N)的复杂度完成如下C++语言算法，数组下标从0开始。给定一个查询整数q，返回该查询q在数组A中**最后一次出现**的下标位置，若未出现，则返回−1；（例如q=6，返回7）

```
int A[N+1]; # 假设A已经赋值完毕
int l = _____, r = ______, m;
while(r - l > ____) {
    m = (l + r) / 2;
    if (A[_____] _____ q) r = m; else l = m;
}
if (A[_____] _____ q) cout<<l; else cout<<-1;
```

课堂作业二

- 给定一个不降序整数数组A（长度为N>0），例如 −1，0，1，1，4，6，6，6，10，请用log(N)的复杂度完成如下C++语言算法，数组下标从0开始。给定一个查询整数q，返回该查询q在数组A中**第一次出现**的下标位置，若未出现，则返回−1；（例如q=6，返回5）

```
int A[N+1]; # 假设A已经赋值完毕
A[N] = inf;
int l = _____, r = ______, m;
while(r - l > ____) {
     m = (l + r) / 2;
     if (A[_____] _____ q) r = m; else l = m;
}
if (A[_____] _____ q) cout<<r; else cout<<-1;
```

线性时间

- 线性时间 O(n)

问题：计算n个数$a_1, \ldots, a_n$中的最大数。

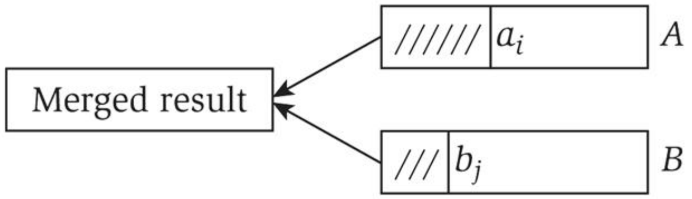
```
max ← a1
for i = 2 to n {
   if (ai > max)
      max ← ai
}
```

线性时间

归并两个排好序的数表

- 问题：给定两个数组，每个包含有n个数，并且每个按照上升的顺序排列，$A = a_1, a_2, \ldots, a_n$；$B = b_1, b_2, \ldots, b_n$，把它们归并成按照上升顺序排列的数组。

线性时间




```
i = 1, j = 1
while (both lists are nonempty) {
   if (ai ≤ bj) append ai to output list and increment i
   else(ai > bj)append bj to output list and increment j
}
append remainder of nonempty list to output list
```

O(nlogn)阶时间

- 快速排序(Quicksort)
- FFT(快速傅立叶变换)
- 归并排序， 堆排序

O(nlogn)阶时间

- 实际中碰到的问题：
- **最大时间间隔问题**： 给定一组n个时间标签$x_1, \ldots, x_n$，一个文件的副本在这些时间到达一个服务器，我们想找出在第一个和最后一个时间标签之间的***最大的***没有文件副本到达的***时间区间***。
- 本质上是一个排序问题： 寻找最大的时间间隔

平方时间

- 问题：假设平面上给定n个点，每个点由(x,y)坐标指定。需要找最邻近(距离最小)的点对。
- O(n^2)解决方法：一个个点对尝试

平方时间

```
min ← (x1 - x2)² + (y1 - y2)²
for i = 1 to n {
   for j = i+1 to n {
      d ← (xi - xj)² + (yi - yj)²
      if (d < min)
         min ← d
   }
}
```

$O(n^2)$的解决办法看起来是最好的，
但还有更好的方法！

立方时间

- 问题：给定集合S_1, …, S_n，它们都是{1,2,…,n}的子集，我们想知道这些子集中是否有某对子集是**不相交**的，也就是没有共同的元素。
- 解决思路：对每对集合S_i,S_j,确定是否S_i,S_j有一个共同的元素。

立方时间

```
foreach set Si {
   foreach other set Sj {
      foreach element p of Si {
         determine whether p also belongs to Sj
      }
      if (no element of Si belongs to Sj)
         report that Si and Sj are disjoint
   }
}
```

$O(n^3)$ solution

$O(n^k)$时间

- 问题： 对某个固定常数k, 我们想知道给定的n个结点的输入图是否有一个大小为k的独立集。

- 如果用蛮力搜索的方法：枚举所有k个结点的子集， 并且对每个子集S检查是否存在一条边与S中的任意两个元素相交

$O(n^k)$时间

```
foreach subset S of k nodes {
   check whether S in an independent set
   if (S is an independent set)
      report S is an independent set
   }
}
```

枚举所有**k**个结点的子集: $\binom{n}{k} = \frac{n\,(n-1)\,(n-2)\cdots(n-k+1)}{k\,(k-1)\,(k-2)\cdots(2)\,(1)} \le \frac{n^k}{k!}$

判断**k**个结点的子集是否独立： $O(k^2)$

所以蛮力搜索的代价为： $O(k^2\, n^k / k!) = O(n^k)$

指数时间(Exponential time)

- 问题：假设给定一个图，需要找一个最大规模的独立集。
- 解决思路：蛮力搜索的方法, 检查每一个子集是否是独立集，是否最大

指数时间(Exponential time)

```
S* ← ϕ
foreach subset S of nodes {
   check whether S in an independent set
   if (S is largest independent set seen so far)
      update S* ← S
   }
}
```

估算计算时间的代价

$$O(n^2 2^n)$$

n!时间

- 对稳定匹配问题的穷举搜索: n!
- 二分匹配(二部图每边存在n个结点)问题
  如果穷举搜索， 代价是n!
- 巡回售货员(TSP:Travelling Salesman Problem)问题：给定n个城市的集合， 每对城市之间都有距离， 什么是访问所有城市的最短旅行？固定第一个(结束)城市
  穷举代价是(n-1)!

n!时间

- Stirling公式 (1730)

$$\lim_{n\to\infty}\frac{n!}{\sqrt{2\pi n}\,n^n e^{-n}}=1.$$

记成当 $n\to\infty, n!\approx\sqrt{2\pi n}\left(\frac{n}{e}\right)^n.$

稳定匹配上机实验

U117121 A 稳定匹配

登录后才可提交

题目背景

复制 Markdown 展开 进入 IDE 模式

算法导论第一次作业

题目描述

要求按照书中 Page 5 的 G-S 算法写一个匹配程序（由男方发起邀请）。

输入格式

第一行数据表示 Man(Woman)的数目 n； 接下来的数据中， 第一个数据块表示男人的优先列表，另一个数据块表示女人的优先列表。

输出格式

一行n个数字，用空格隔开，依次表示男人 1 到 n 的匹配对象。

输入输出样例

输入 #1 复制

```
5
2 1 4 5 3
4 2 1 3 5
2 5 3 4 1
1 4 3 2 5
2 4 1 5 3

5 1 2 4 3
3 2 4 1 5
2 3 4 5 1
1 5 4 3 2
4 2 5 3 1
```

输出 #1 复制

```
1 3 2 5 4
```

说明/提示

不要输出额外的提示信息