西南大学

算法-引论

张里博 lbzhang@swu.edu.cn

目录

PART 01

一些基本概念

算法与程序

- 算法是完成一个任务的流程，或者解决一个问题的方案。程序是实现预期目的而在**计算机上编写的**一系列语句和指令；
- 程序不一定满足有输出、有穷性（操作系统）等条件。
- 算法就是程序的灵魂，算法的优劣决定了程序的好坏
- 一个特定功能，实现的算法有很多种，所以一个算法的质量优劣将直接影响程序的效率和效果。

算法的基本特征

- 输入(Input)
- 一个算法有0个或多个输入，以刻画运算对象的初始情况；
- 输出(Output)
- 一个算法有一个或多个输出，以反映对输入数据加工后的结果；
- 确定性(Definiteness)
- 算法中的每一条指令必须有确切的含义，不能产生多义性；
- 可执行性(Effectiveness)
- 算法中的每一条指令可以通过基本运算执行有限次来实现；
- 有穷性（Finiteness）
- 算法必须能在有限步后终止；

算法的设计

- 算法设计一般遵循的原则：自顶向下、**逐步求精**；
- **依赖于抽象数据类型等工具，算法的顶层设计与底层实现分离**
- **进行顶层设计时不用考虑数据类型、运算表示和具体实现。算法的复杂性降低，条理性和模块化也增强了；**
- 顶层设计与底层实现局部化，容易查找和纠正错误。

算法正确性分析

- **算法分析分为算法正确性分析和复杂性分析**

- 程序调试（验证）**只能证明算法错误**，不能证明算法正确。
- 测试和验证不能穷尽所有的可能。

- 算法可以通过数学方法来进行推理证明
- 常见的方法：**直接证明、反证法和数学归纳法**。

- **算法的正确性证明仍是一项很有挑战性的工作。**

递归与迭代

递归采用自顶向下的策略，通过不断循环调用自身，降低单个问题的规模；
迭代采用自底向上的策略，通过不断循环逼近目标；

已知：1!=1; n!=n* (n-1)!
分别采用递归和迭代计算n!。（n≥1）

```
factorial_rec(n)
1.   if  n==1  then
2.       return 1;
3.   else
4.       return n*factorial_rec(n-1);
5.   end
```

```
factorial_ite(n)
1.   product←1;
2.   for i ← 2 to n do
3.       product← product*i;
4.   end
5.   return product;
```

递归三板斧：
1.起名字；2.截止条件；3.递归调用

迭代：
自底向上，从小到大逼近

PART 02

算法的描述

算法的描述

- 算法的描述方法有很多，常见的有：**自然语言、流程图和伪码**；
- 高级程序设计语言十分接近算法语言，但并不是算法的描述方式；
- **算法描述主要关注算法的主干部分，即算法的顶层设计，是求解问题的主要步骤；**
- **此部分描述的首要目的是方便阅读、理解和交流**

算法的描述

- 算法的描述方法有多种，例如自然语言、流程图和伪代码等，但都要满足算法的基本特征
- 1. 自然语言
- 优点：容易理解；缺点：冗长，容易出现二义性，不直观，细节呈现不多

- 2. 流程图
- 优点：直观易懂；缺点：严密性不足

- **3. 伪代码**
- 优点：表达能力强，细节呈现多；缺点：没有图形直观

算法的伪码表示

- 伪代码(Pseudocode)是一种算法描述语言，由带符号的指令组成，是**算法步骤的描述**。
- 忽略了程序的实现细节，而更加关注于求解问题的思路与步骤。
- 伪代码必须结构清晰（例如**缩进**，分支和循环体的**end**），并且类似自然语言，可读性好。

伪码规则

- *赋值语句*：←
- 分支语句（*以end结尾*）：if …then … [else…]; switch
- 循环语句（*以end结尾*）：while，for，repeat until
- 转向语句：goto
- 输出语句：return
- 调用：直接写过程的名字
- 注释：//…或者/*…*/
- *相等*：==
- …


```
if  x<100 then
    j← 0;
elif  x>200 then
    j← 2;
else
    j← 1;
end
```

算法的伪代码描述

算法1.1 Euclid(*m*,*n*)

输入：非负正整数 *m*, *n*

输出：*m* 与 *n* 的最大公约数

辗转相除法

1. **while *m*>0 do**
2. ***r*← *n* mod *m*;** /*取模运算，即得到余数*/
3. ***n*← *m*;**
4. ***m*← *r*;**
5. ***end***
6. **return *n***

n=16,m=6;

16%6 =4；

6% 4 =2；

4 %2= 0

n<m，？

问题建模

输入：任务集 $S=\{1, 2, \ldots, n\}$，

第 j 项任务加工时间：$t_j \in Z^+, j=1, 2, \ldots, n$

输出：调度方案I，即S的排列 $i_1, i_2, \ldots, i_n$

目标函数：I的总等待时间 $t(I)=\sum_{k=1}^{n}(n-k+1)t_{i_k}$

$$\arg\min_{I}\{t(I)\} \text{ or } \min_{I}\{t(I)=\sum_{k=1}^{n}(n-k+1)t_{i_k}\}$$

采用贪心策略，即加工时间短的先安排

算法的伪码表示

$$\min\{\sum_{k=1}^{n}(n-k+1)t_{i_k}\}$$

- 输入:任务集 S={1, 2, … , n}，任务执行时间 $t_j \in Z^+, j=1, 2, \ldots, n$
- 输出： 最佳加工顺序I_*和最短总等待时间t_*

采用贪心策略，即加工时间短的先安排

- 1. $T \leftarrow [t_1, t_2 \ldots t_n]$; /*$t_i$是第$i$个任务的执行时间*/
- *2.* [*I, T'*]← *sort (T);*
- /* *T'*是将向量*T*中元素升序排列后的向量， *I是T'*中元素在向量*T*中的序号（下标）*/
- *3.* $t \leftarrow 0$;
- *4.* for $j \leftarrow 1$ to n do
- 5. $t \leftarrow t + (n-j+1)*T'(j)$; /*计算每个任务的等待时间*/
- 6. end
- 7. retrurn {*I, t*}

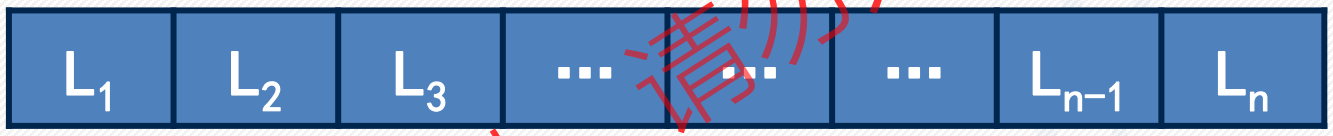
查找问题中的算法设计

算法1.2.1 Search(*L*,*x*)

数组*L*是有序的

输入：数组*L*[1..*n*]，其元素按照从小到大排列，数 *x*.

输出：若 *x* 在*L*中，输出第一个 *x* 的位置下标 *j*；否则输出0.



寻找第一个大于等于（不小于）x的元素，然后进行判断是否等于

1 ≤ j ≤ n, *x>L[j]*?
Right? Go on;
Wrong? Break;
x<L[j]?
Right? *j*← 0
Return *j*

x=8

顺序查找算法的伪代码描述

算法1.2.1 Search(L,x)

输入：数组$L[1..n]$，其元素按照从小到大排列;数 x.

输出：若 x 在L中，输出第一个 x 的位置下标 j；否则输出0.

1. $j \leftarrow 1$;
2. while $j \le n$ and $x > L[j]$ do
3. $\quad j \leftarrow j+1$;
4. end
5. if $x < L[j]$ or $j > n$ then
6. $\quad j \leftarrow 0$;
7. end
8. return j

$j==1$ *and* $x \le L[1]$; <?

$j==1$ and $x>L[1]$; <?

$1<j<n$ and $L[j-1]<x \le L[j]$; <?

$j == n$ and $x \le L[n]$; <?

$j == n$ and $x>L[n]$; $j \leftarrow 0$

PART 03

算法的时间复杂度

算法的时间复杂度分析

- **算法分析分为算法正确性分析和复杂性分析**
- 算法正确性分析确保算法对符合要求的输入在有限时间内能产生正确的结果；
- 程序调试（验证）**只能证明算法错误**，不能证明算法正确。
- 测试和验证不能穷尽所有的可能。因此，通过测试的算法不一定没有错误。

一、基本运算次数

- **基本运算：**算法运行过程中起主要作用且**花费时间最多**的运算。
  - 包括：加法，乘法，比较，元素的交换（移动）等；
  - 具体场景下的约定，例如排序和**检索问题**中，一般约定**比较运算**为基本运算

- **基本运算运行时间的差异可以忽略。因此，基本运算次数**常被用来**衡量算法的效率**。

时间复杂度函数（简称为时间复杂度）：将基本运算次数表示为问题规模的函数。

输入：无序数组$L[1\cdots n]$；

输出：数组$L[1..n]$中最大的元素值；

顺序检索最大值算法

将第一个作为最大值，从前向后依次比较，遇见更大的就替换

假设数组L有n个元素，顺序检索最大值算法的时间复杂度函数

共需要进行$n-1$次比较，时间复杂度函数为$T(n)=n-1$；

二、时间复杂度函数

输入：无序数组$L[1\cdots n]$，元素 x;

输出：若元素 x 存在于数组 L 中，则输出 x 的位置下标；

若元素 x 不存在于数组 L 中，则输出 0；

20	7	5	9	14	12	2

相同问题规模下，算法的时间复杂度与输入有关

最好情况下的时间复杂度：x是L 的第一个元素，时间复杂度为 1；

最坏情况下的时间复杂度：x 是L 的最后一个元素或不在 L 中，

时间复杂度为 n;

平均情况下的时间复杂度：x每种情况的概率乘以对应的时间复杂度

二、时间复杂度函数

假如你委托一个公司开发算法，你希望对方的承诺最好情况下，最坏情况下，还是平均情况下的运行时间？

最好情况下？

平均情况下？能否预知每种合法输入出现的概率？

“最坏情况下的时间复杂度，代表了对用户的承诺”

麻省理工学院（MIT） 《Introduction To Algorithms》

因此，最常用的是最坏情况下的时间复杂度

三、渐进时间复杂度

- 时间复杂度函数的渐近表示，也被称为算法的**渐进时间复杂度**，是被广泛认同的衡量算法效率的方式。

- 目的：问题**规模大的时候，算法的计算效率。即**输入**规模很大的时候，时间复杂度函数的增长速率。**

- **问题规模大的时候，**影响时间复杂度函数**增长速率**的**关键因素是函数的最高阶**

三、渐进时间复杂度

渐近上界 $f(n)= O(g(n))$ ： $a_f \le a_g$;

渐近下界 $f(n)= \Omega(g(n))$ ： $a_f \ge a_g$;

渐近紧界 $f(n)= \Theta(g(n))$ ： $a_f = a_g$;

非紧上界 $f(n)= o(g(n))$ ： $a_f < a_g$;

非紧下界 $f(n)= \omega(g(n))$ ： $a_f > a_g$.

问题规模很大时，最高阶是影响函数增长速率的关键因素，而常数、系数、（除最高阶外的）低阶函数影响不大。

定理1.1 设 f 和g是定义域为自然数集合的函数.

(1)如果 $\lim\limits_{n\to\infty}\dfrac{f(n)}{g(n)}$ 存在且等于某个常数$c>0$，那么

$$f(n)=\Theta(g(n)).$$

(2)如果 $\lim\limits_{n\to\infty}\dfrac{f(n)}{g(n)}=0$，那么 $f(n)=o(g(n))$.

(3)如果 $\lim\limits_{n\to\infty}\dfrac{f(n)}{g(n)}=+\infty$，那么 $f(n)=\omega(g(n))$.

三、渐进时间复杂度

- 求解同一个问题时
- 算法A（最坏情况下）的时间复杂度函数是 $f(n)=n^2+100n$;
- 算法B 最坏情况下）的时间复杂度函数是$g(n)=2n^2$;
- 算法C（最坏情况下）的时间复杂度函数是$h(n)=0.5n^3$;

- $f(n) = \Theta(g(n))$，$f(n) = o(h(n))$
- 因此，从渐近时间复杂度的角度（问题规模足够大），算法A与算法B时间效率没有明显差别，但都优于算法C。

算法最坏情况下的时间复杂度

算法1.2 Search_max(*L*, *x*)

输入：无序数组*L*[1…*n*]

输出：数组*L*[1..*n*]中最大的元素值；

```
1.  x ← L[1];
2.  for  j ← 2  to  n do
3.      if L[j] > x then
4.          x ← L[j];
5.      end
6.  end
7.  return x
```

问题规模很大时，最高阶是影响函数增长速率的关键因素，而常数、系数、（除最高阶外的）低阶函数影响不大。

只将比较作为基本运算

n−1次

比较、赋值都作为基本运算

1次
n−1次
n−1次
n−1次

1次

渐近时间复杂度

$T(n)=n-1=O(n)$

$T(n)=3n-1=O(n)$

几个常见的渐进时间复杂度

- 1. $n^d = o(r^n), r > 1,\ d > 0;$
- 2. $log_a n = o(n^d);$
- 3. $log_a n = \Theta(log_b n);$
- 4. $log(n!) = \Theta(n\ log\ n);$
- 5. $n = \Theta(r^{log\ n});$

函数的阶

- 阶的符号：$O, \Omega, \Theta, o, \omega$
- 阶的高低

  至少指数级：$2^n, 3^n, n!, \ldots$

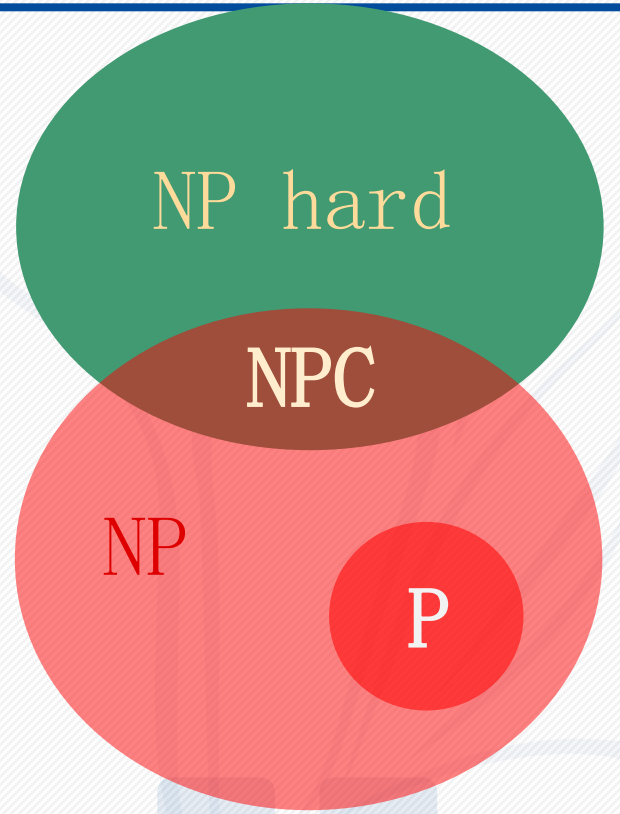
  多项式级：$n, n^2, n\log n, n^{1/2}, \ldots$

  $\log n$的多项式级：$\log n, \log^2 n, \ldots$

- 注意

  阶反映的是大的n（$n>n_0$）的情况——可以忽略前面的有限项

- P问题是指存在多项式时间求解算法；NP问题不确定是否存在多项式时间求解算法，但确定存在多项式时间验证解正确性的算法
- P问题是NP问题的子集，因为存在多项式时间求解算法的问题，一定能够在多项式时间内被验证。
- NP hard问题不一定是NP问题，有可能是不可判定问题。这时候说明原问题也是不可判定的。
- NPC是否存在多项式时间尚未得到证明，是P≠NP成立的关键
- 一般认为P≠NP ，NPC问题算法的时间复杂度是指数级甚至更高阶（如果P≠NP），该问题是NP问题和NPH问题的交集。




西南大學
谢谢大家！
内部资料，请勿外传。
西南大學
西南大學