CCS592 NOTEBOOK

liyu0x

2021.03-2021.07

目录

1 Algorithm Analyse

1.1 Concepts(3)

Keyword	Interpretation
scalability	Scalability refers to the ability of a system to gracefully accommodate growing sizes of input or amounts of workload.
algorithm	An algorithm is a step-by-step procedure for performing some task in a finite amount of time.
data structure	a data structure is a systematic way of organizing and accessing data.

1.2 Pseudocode

Less detailed than a program.More structured than English prose.

Event	Notation
Control flow 流程控制 (5)	

	if...then...[*else...*]
	while...do...
	repeat...until...
	for...do...
	Indentation replace brace 缩进替换括号，参考 python
Method declaration 方法声明 (1)	
	Algorithm method (*arg*, [, *arg...*])
	Input...
	Output...
Method Call 方法调用 (1)	
	method (*arg* [, *arg...*])
Return Value 方法调用 (1)	
	return expression
Expression 表达式 (3)	
	← *Assignment* 赋值
	= *Equality testing* 判断等于
	n^2 Superscripts and other mathematical formatting allowed 可以使用数学表达式

1.3 RAM

the random access machine model 随机访问模型

A RAM consists of

- A CPU
- An potentially unbounded bank of memory cells, each of which can hold an arbitrary number or character 无限的存储单元，每一个存储单元能存储一个任意基本类型的数据
- Memory cells are numbered and accessing any cell in memory takes unit time. 每个存储单有编号，并且访问任意的存储单元只用单位时间

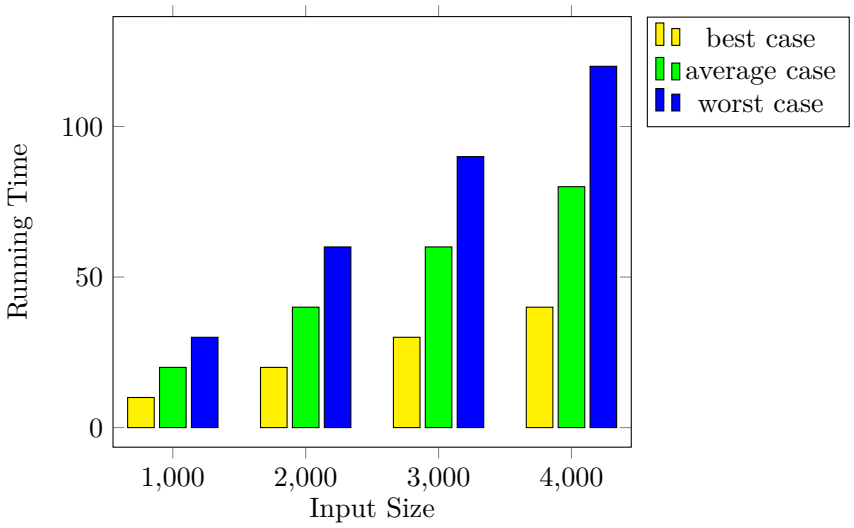
How use the RAM 将算法运行在 RAM 模型中，这样算法的每一个 Primitive Operation 都可以以单位时间执行而不必考虑环境差异，在 RAM 中统计有限个的 Primitive Operation，将 Primitive Operation 的运行次数作为 t，则 t 与算法运行时间成正相关。

1.4 Running Time

Thing about RT

1) Most algorithm transform input objects into output objects.
2) The running time of an algorithm typically grows with the input size. 时间随入参规模增长
3) Average case time is often difficult to determine. 一般案例的运行时间通常难以判断
4) We focus primarily on the worst case running time

- easier to analyse 坏情况更容易分析
- crucial to application such as games, finance and robotics 能造成更严重的影响



Runtime and Input size

Growth Rate 算法运行时间函数 $T(n)$ 随着入参规模的增长率 (其实就是时间复杂度函数的斜率，比如时间复杂度为 n^2，当遇到规模为 $4n$ 的问题时，时间增长 16 倍)，当运行环境发生变化时：

- Affecs $T(n)$ by a constant factor $T(n)$ 受到影响
- Dose not alter the growth of $T(n)$ 但它的增长率不会改变

Affect Factors 算法的运行时间容易受到 Hardware/software 环境的影响。运行时间会小于一个带有常数的函数，比如 $n < cn$,c 就是由软硬件决定的常量因子。

1.5 Primitive Operations

基础操作，也可以理解为原子操作. Primitive Operations 是分析算法的基本单位。

Properties

- Basic Computations performed by an algorithm 算法表现的基本计算
- Identifiable in pseudocode 用伪代码来表示
- Largely independent from the programming language 不依赖特定的编程语言语法
- Exact definition not important 不用准确定义算法，比如边界判断之类
- Assumed to take a constant amount of time in the RAM model 每步操作在 RAM 模型中是单位耗时

1.6 Analyse Method

1.6.1 Experiments

Experimental Studies Step(3)

1) write a program implementing the algorithm 实现一个算法
2) run the program with inputs of varying size and composition, noting the time needed 使用不同规模的入场并记录时间
3) plot the result 绘制结果

Limitation(3)

- It is necessary to implement the algorithm, which may be difficult 算法实现可能太复杂
- Results may not be indicative of the running time on other inputs not included in the experiment 无法覆盖所有的入参规模
- In order to compare two algorithms, the same hardware and software environments must be used 必须在相同环境下进行比较测试

1.6.2 Theoretical

可以理解为形式化验证，即用伪代码来展示

Theoretical method step(2)

- uses Pseudocode description of the algorithm instead of an implementation 使用伪代码
- characterizes running time as a function of the input size, n 运行时间可以用一个入参为 n 的函数表示

Advantage(2)

- Takes into account all possible inputs 覆盖所有可能的输入
- Independent of the hardware/software environment 不依赖运行环境差异

1.7 Analysis Case

1.7.1 $arrayMax(A, n)$

Algorithm 1 arrayMax

Input: An array A storing n >= 1 integers.

Output: The maximum element in A.

```
1: function ARRAYMAX(A, n)
2:     currentMax ← A[0]
3:     % 2 个操作: indexing(索引) 和 assigning(赋值)
4:
5:     for i ← 1 to n − 1 do
```

```
6:      % for 循环包括多个操作：
7:      % 首先 for 初始化时 (i <- 1), 是一个 assigning(赋值) 操作且只会执行一次
8:      % n 个操作: i < n 判断操作要比循环操作多一次
9:      % 2(n-1) 个操作: indexing(索引) 和 summing(加法) 循环的时候需要 i = i + 1
10:
11:     if currentMax < A[i] then
12:         % 2(n-1) 个操作: indexing(索引) 和 comparing(比较)
13:
14:         currentMax ← A[i]
15:         % 2(n-1) 个操作: indexing(索引) 和 assigning(赋值)
16:     end if
17:   end for
18:   return currentMax
19:   % 1 个操作: returning(返回)
20: end function
```

Conclusion 分析增长率:

- Worst case operations: $7n-2 \quad \rightarrow \quad 2+1+n+2(n-1)+2(n-1)+2(n-1)+1$ 最坏可能即每个语句都将被执行
- Base case operations: $5n \quad \rightarrow \quad 2+1+n+2(n-1)+2(n-1)+1$ 最好的可能即尽可能执行少的语句也就是 $currentMax < A[i]$ 总是为 $false$

And We assume following situation:

- a = Time taken by the fastest primitive operation 每个基本操作都是最快的情况下所耗时间
- b = Time taken by the slowest primitive operation 每个基本操作都是最慢的情况下所耗时间
- $T(n)$ = worst-case time of this function 该函数最差运行时间

Then, We can get a formula:
$a(5n) \leq T(n) \leq b(7n-2)$
该算法时间受两个线性函数限制，所以它的增长率是线性 (Linear)

Properties	Value	Explain
Time	$7n-2$	上面分析
Space	n+3	分别是入参变量 n, 临时变量 $currentMax$ 和循环计数器 i，及其数组 A 中的 n 个元素

PS: 我觉得吧，这里如果 $T(n)$ 是最差值，不应该就是 $b(7n-2)$ 吗？查阅了 MR1 的教科书，$T(n)$ 是引入的一个变量代表算法的运行时间，我直接理解 $T(n)$ 就是该算法的运行时间，然后有如下时间选项：

1. $a(5n)$ 在效率最高的基本操作环境下执行 $5n$ 个语句，耗时最快

2. $a(7n-2)$ 在效率最高的基本操作环境下执行 $(7n-2)$ 个语句，耗时与 3 无法比较，但比 4 快，比 1 慢
3. $b(5n)$ 在效率低的基本操作环境下执行 $5n$ 个语句，耗时与 2 无法比较，但比 4 快，比 1 慢
4. $b(7n-2)$ 在效率最低的基本操作环境下执行 $(7n-2)$ 个语句，耗时最慢

此时要在插入一个 T 在这些时间段里面，很明显我只能插入在 1 和 4 之间，因为 2 和 3 无法估计。

1.7.2 $recursiveMax(A,n)$ 递归函数

Algorithm 2 recursive

Input: An array A storing n >= 1 integers.

Output: The maximum element in A.

```
1: function RECURSIVEMAX(A, n)
2:     if n = 1 then
3:         $ 一个操作: 比较
4:
5:         return A[0]
6:         % 两个操作: 索引和返回
7:     end if
8:     return max{recursiveMax(A, n − 1), A[n − 1])}
9:     %
10: end function
```

Properties	Value	Explain
Time	3 when $n=1$	上面
	T(n-1)+7 otherwise	也可以写成 7(n-1)+3
Space	n+(n-1)	这里的答案不一定正确，因为 PPT 上没有，n 是数组 a 的大小，由于数组在很多语言中实现都是传址而不是传值，所以即使递归也不会造成重复开辟空间，但是 n 是传值，所以每次都有需要有参数来接受，这里没有算上创建方法栈所需要的空间

1.8 Big-O

1.8.1 7 Important O

FORMAT	TYPE	EXAMPLE
$O(1)$	Constant	Hash Table
$O(logn)$	Logarithmic	Binary Search
$O(n)$	Linear	Linear Search 或线性访问
$O(nlogn)$	Superlinear	Heap Sort, Merge Sort
$O(n^c)$	Polynomial	Bubble sort, Bucket Sort

$O(c^n)$	Exponential	Tower of Hanoi
$O(n!)$	Factorial	行列式展开

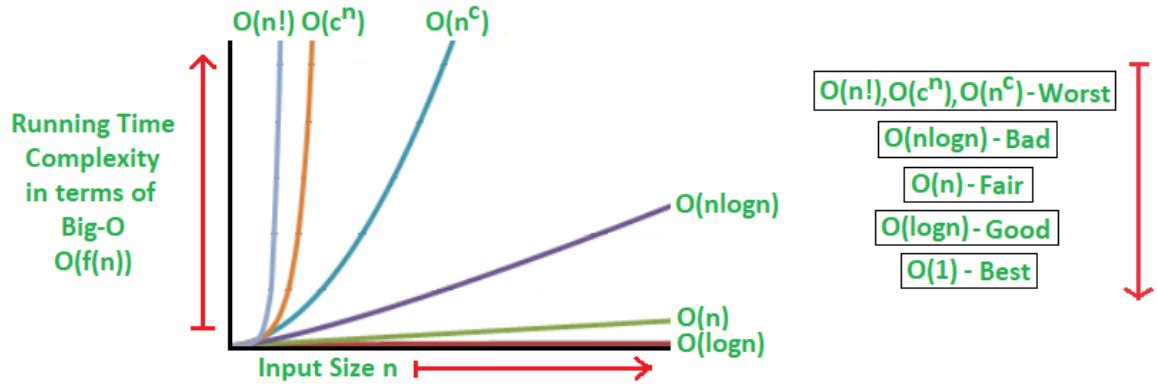


图 1: O graphic picture

Picture From: https://www.geeksforgeeks.org/analysis-algorithms-big-o-analysis/

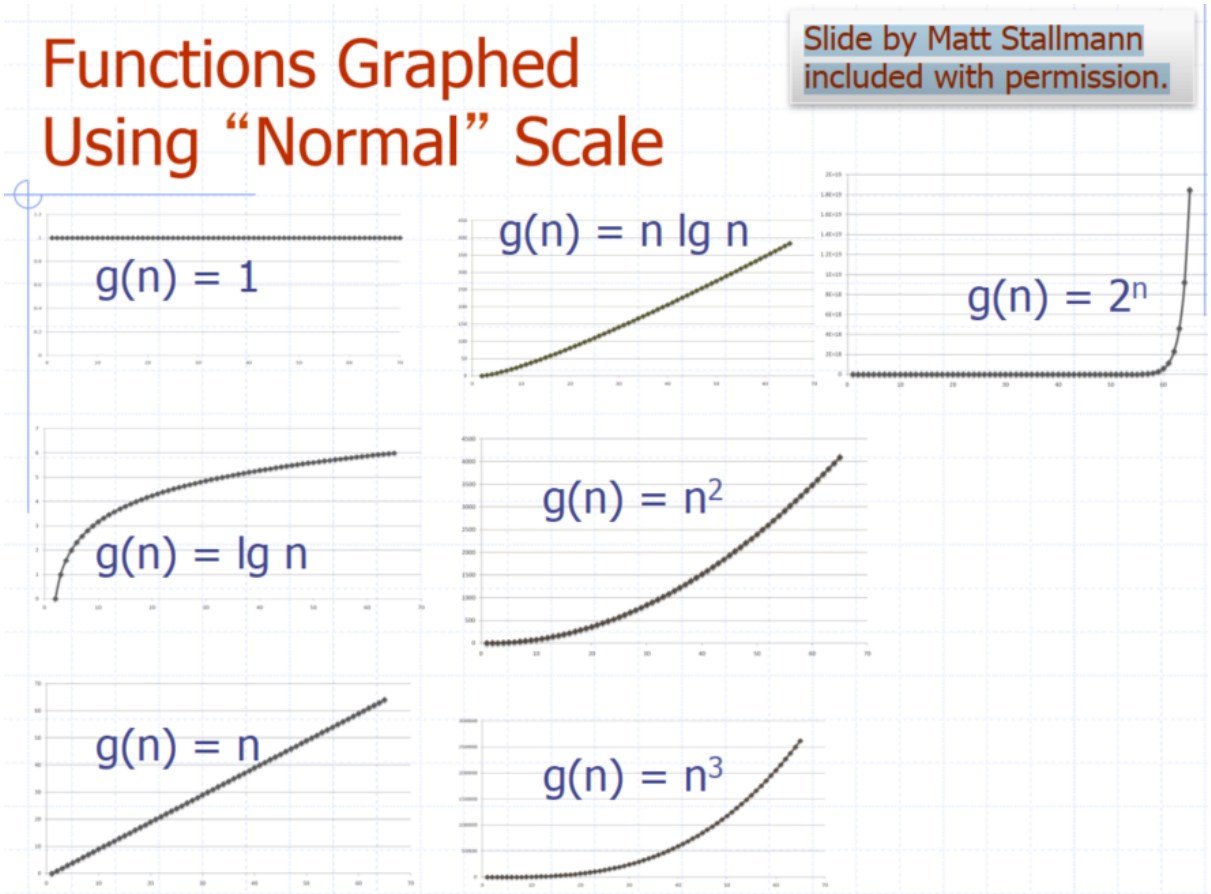


图 2: Mathematical graphic picture