第 2 讲: 算法的效率

姓名： 姚梦雨　　**学号：** 191840305

评分： ______　　**评阅：** ______

2020 年 3 月 3 日

请独立完成作业，不得抄袭。
若得到他人帮助, 请致谢。
若参考了其它资料，请给出引用。
鼓励讨论，但需独立书写解题过程。

1　作业（必做部分）

题目 1 (DH Problem 6.18: $\log_m n$)
Design an algorithm $LG1$,with input integers $m, n > 1$,that calculates $\log_m n$ by repeatedly calculating the powers $m^0, m^1, ..., m^k$, until a number k is found satisfying $m^k \le n < m^{k+1}$. Analyze the time and space complexity of $LG1$.

Algorithm 1 $\log_m n$

解答:

function $LG1(m, n)$　　▷ 程序已在 Clion 上运行正确，可用循环不变式证明
　$X \leftarrow 1$;　　▷ $T_1 = O(1)$
　$k \leftarrow 0$;　　▷ $T_2 = O(1)$
　while $X \le n$ **do**　　▷ $T_3 = O(\log_m n)$
　　$X \leftarrow mX$;　　▷ $T_4 = O(\log_m n)$
　　$k \leftarrow k + 1$;　　▷ $T_5 = O(\log_m n)$
　end while
　return $k - 1$　　▷ $T_6 = O(1)$
end function

Analysis:
(1)Through the time of each process listed above,the time complexity of $LG1$ is the plus of them, it is $O(\log_m n)$
(2)As the algorithm only occupy constant number of space,the space complexity of it is $O(1)$.

题目 2 (DH Problem 6.19: $\log_m n$)
It is well known that each positive integer k can be written uniquely as a sum of integer powers of 2,i.e.,in the form $k = 2^{l_1} + 2^{l_2} + \cdots + 2^{l_j}$,where $l_1 > l_2 > \cdots > l_j \ge 0$.For example, $12 = 2^3 + 2^2$, and $31 = 2^4 + 2^3 + 2^2 + 2^1 + 2^0$. Hence, $m^k = m^{2^{l_1}} \times m^{2^{l_2}} \times$

$\cdots \times m^{2^{l_j}}$, and if we need to calculate $k = \log_m n$, it is enough to find the appropriate exponents $l_1, l_2, ..., l_j$.
Design an iterative(i.e.,nonrecursive)algorithm $LG2$ to calculate $\log_m n$ by first finding an integer l_1 satisfying $m^{2^{l_1}} \le n < m^{2^{l_1+1}}$, then finding an integer $l_2 < l_1$, satisfying $m^{2^{l_1}} \times m^{2^{l_2}} \le n < m^{2^{l_1}} \times m^{2^{l_2+1}}$, and so on. Use a fixed amount of memory space (no lists). Analyze the time and space complexity of algorithm $LG2$.

Algorithm 2 $\log_m n$

解答:

function $LG2(m, n)$ ▷ 程序已在 Clion 上运行正确，可用循环不变式证明
  $k \leftarrow 0$
  $X \leftarrow 1$
  while $(m \times X) \le n$ **do**
    $k_0 \leftarrow 1$
    $X_1 \leftarrow m \times m$
    $X_2 \leftarrow m \times X$
    while $(X_1 \times X) \le n$ **do** ▷ $X_1 = m^{2k_0}$
      $X_2 \leftarrow X$
      $X_2 \leftarrow X_2 \times X_1$
      $k_0 \leftarrow 2k_0$
      $X_1 = X_1 \times X_1$
    end while
    $X \leftarrow X_2$
    $k \leftarrow k + k_0$
  end while
  return k
end function

Analysis:
(1)Suppose that $log_m n = k = 2^{l_1} + 2^{l_2} + \cdots + 2^{l_j}$,so it is obvious that the external loop executes j times,while the inner loop executes $(l_1 + l_2 + ... + l_3)$ times.The order of growth of j and l_k is apparently $\log_2(\log_m n)$.Ignoring the constant coefficient and smaller growth,the inner loop dominates the time complexity of the algorithm ,and its order of growth is $(\log_2(\log_m n))^2$,that is $O((\log_2(\log_m n))^2)$.Thus,if we move the base of the logarithms,the time complexity is $O((\log(\log n))^2)$.
(2)Since the algorithm just occupies constant number of space,the space complexity is $O(1)$.

题目 3 (DH Problem 6.20 (a): $\log_m n$)
The time complexity of the previous algorithm can be improved by calculating each of the values $m^{2^{l_1}}, m^{2^{l_2}}, ..., m^{2^{l_j}}$ only once.
(a) Design such an algorithm $LG3$, and analyze its time and space complexity.
(b) Discuss the time/space tradeoff concerning the last two algorithms. Suggest joint time/space complexity measures under which $LG3$ has better/equivalent/worse complexity than $LG2$. What happens when you replace $LG2$ in this analysis with $LG1$?

解答:
(a)

Algorithm 3 $\log_m n$

function $LG3(m, n)$ ▷ 程序已在 Clion 上运行正确，可用循环不变式证明
  $X \leftarrow 1$
  $i \leftarrow 1$
  $X_1 \leftarrow m$
  $k_1 \leftarrow 1$
  while $(X_i \times X) \le n$ **do** ▷ $X_j = m^{2^{j-1}}, k_j = 2^{j-1}$
    $k_{i+1} \leftarrow 2k_i$
    $X_{i+1} \leftarrow X_i \times X_i$
    $i \leftarrow i + 1$
  end while
  $l \leftarrow i - 1$
  $k \leftarrow k_l$
  $X \leftarrow X_l$
  while $(m \times X) \le n$ **do**
    $l \leftarrow l - 1$
    while $(X \times X_l) > n$ **do**
      $l \leftarrow l - 1$
    end while
    $X \leftarrow X \times X_l$
    $k \leftarrow k + k_l$
  end while
  return k
end function

Analysis:
(1)The first loop :$O(\log_2(\log_m n))$;The outer of the second:$O(\log_2(\log_m n))$;The inner of the second $O(\log_2(\log_m n))$.Therefore, the total time complexity of the algorithm is $O(\log_2(\log_m n))$,if base of the logarithm removed, it is $O(\log(\log n))$.
(2)The algorithm occupies two lists k_i and X_i and constant number of variables.The oders of growth of space of the two lists are all $O((\log_2(\log_m n)))$.So the space complexity of the algorithm is $O((\log_2(\log_m n)))$,if base of the logarithm removed, it is $O(\log(\log n))$.
(b)(1)In terms of time complexity, $LG3$ is better than that of $LG2$,while in terms of space complexity ,$LG3$ is worse than $LG2$.Actually,the product of their own time and space complexity is equal,so we may say under this jiont measure,$LG3$ is equivalent to $LG2$,as it just sacrifices larger space for less time.
(2)Compared with $LG1$.In terms of time complexity, $LG3$ is better than that of $LG1$,while in terms of space complexity ,$LG3$ is worse than $LG1$.However,the product of their own time and space complexity is not equal,that is $LG3$ is better than $LG1$ under this measure as its product is smaller.

题目 4 (TC Exercise 3.1-6)
Prove that the running time of an algorithm is $\Theta(g(n))$ if and only if its worst-case running time is $O(g(n))$ and its best-case running time is $\Omega(g(n))$.

解答:
Denote that the time of the worst case is $T_w(n)$,and that of the best case is $T_b(n)$.So the running time of the algorithm $T(n)$ satisfying for each n,$T_b(n) \le T(n) \le T_w(n)$.(and the equality will hold)
(1)Proof for:$T_w(n) = O(g(n)), T_b(n) = \Omega(g(n)) \Longrightarrow T(n) = \Theta(g(n))$

(i)Since its worst running time is $O(g(n))$,there exist $n_1 > 0$ and $c_1 > 0$ such that when $n \geq n_1, T_w(n) \leq c_1 g(n)$.
(ii)Since its best running time is $\Omega(g(n))$,there exist $n_2 > 0$ and $c_2 > 0$ such that when $n \geq n_2, T_b(n) \geq c_2 g(n)$.
Therefore,there exist $n_0 = max(n_1, n_2) > 0$ and $c_1, c_2 > 0$ such that when $n \geq n_0, c_2 g(n) \leq T_b(n) \leq T(n) \leq T_w(n) \leq c_1 g(n)$.So the running time of the algorithm is $\Theta(g(n))$
(2)Proof for:$T(n) = \Theta(g(n)) \Longrightarrow T_w(n) = O(g(n)), T_b(n) = \Omega(g(n))$
Since $T(n) = \Theta(g(n))$,there exist $n_0 > 0$ and $c_1, c_2 > 0$ such that when $n \geq n_0, c_2 g(n) \leq T(n) \leq c_1 g(n)$.As $\{T_b(n), T_w(n)\} \subseteq \{T(n)\}$,there exist $n_0 > 0$ and $c_2 > 0$ such that when $n \geq n_0, T_b(n) = \text{some T(n)} \geq c_2 g(n)$.,there exist $n_0 > 0$ and $c_1 > 0$ such that when $n \geq n_0, T_w(n) = \text{some T(n)} \leq c_1 g(n)$.so the best running time is $\Omega(g(n))$,the worst running time is $O(g(n))$.
In conclusion,the running time of an algorithm is $\Theta(g(n))$ if and only if its worst-case running time is $O(g(n))$ and its best-case running time is $\Omega(g(n))$.

题目 5 (TC Exercise 3.1-7)
Prove that $o(g(n)) \cap \omega(g(n))$ is the empty set.

解答:
Assume that it is not an empty set,so there exist such an $f(n) \in o(g(n)) \cap \omega(g(n))$
Since $f(n) \in o(g(n)) \cap \omega(g(n))$,$f(n) \in o(g(n))$ and $f(n) \in \omega(g(n))$,so $\forall c > 0, \exists n_1 > 0$, such that when $n \geq n_1, 0 \leq f(n) < cg(n)$ and $\forall c > 0, \exists n_2 > 0$, such that when $n \geq n_2, 0 \leq cg(n) < f(n)$.
So let $c = c_1$,$\exists n_1 > 0$ such that when $n \geq n_1, 0 \leq f(n) < c_1 g(n)$,$\exists n_2 > 0$, such that when $n \geq n_2, 0 \leq c_1 g(n) < f(n)$.Thus,when $n \geq max(n_1, n_2)$,$c_1 g(n) < f(n) < c_1 g(n)$,which is impossible.
Therefore,the assumption is false,$o(g(n)) \cap \omega(g(n))$ is the empty set.

题目 6 (TC Problem 3-3 (a))
a. Rank the following functions by order of growth; that is, find an arrangement $g_1, g_2, ..., g_{30}$ of the functions satisfying $g_1 = \Omega(g_2), g_2 = \Omega(g_3), ..., g_{29} = \Omega(g_{30})$. Partition your list into equivalence classes such that functions $f(n)$ and $g(n)$ are in the same class if and only if $f(n) = \Theta(g(n))$.
$lg(lg^* n)$, $2^{lg^* n}$, $(\sqrt{2})^{lg\ \mathrm{n}}$, n^2 , $n!$, $(lg\ n)!$
$(\frac{3}{2})^n$, n^3 , $lg^2 n$, $lg(n!)$, 2^{2^n} , $n^{1/lgn}$
$ln\ ln\ n$, $lg^* n$, $n \cdot 2^n$, $n^{lg\ \mathrm{lg\ n}}$, $ln\ n$, 1
$2^{lg\ \mathrm{n}}$, $(lgn)^{lgn}$, e^n , 4^{lgn} , $(n+1)!$, $\sqrt{lgn}$
$lg^*(lgn)$, $2^{\sqrt{2lgn}}$, n , 2^n , $nlgn$, $2^{2^{n+1}}$

解答:
根据常见的增长阶可知以下顺序：$n^n, n!, a^n, n^a, lgn, lg^* n, c$（其中，$a, c$ 是常数）
此后，对各个式子进行化简并初步分类，并逐级进行比较，使用同时取对数等方法，可得到下表，其中处于同一行的为等价类, 由定义知，相差常数的函数为等价类, 而此三十个函数大多数的比较都可以通过上述常见增长阶顺序比较出结果，进行适当的化简后可以较简单的发现等价类。
具体排序方法简述：现根据常见增长阶从高到低可排到 g_8, 从低到高可排到 g_{21}, 由此剩下 10 个函数可初步判定夹在中间，通过化简找出等价类后仅需进行 7 个函数的增长阶排列，其中 4 个的先后顺序已经清楚，此后来判断较难判断的函数 (lgn^{lgn}) 和

$2^{\sqrt{2lgn}}$，$(lgn)!$ 的位置，从对数与多项式的增长量级常识以及取对数等做法即可得出答案：

等价类标号	函数
1	$g_1 = 2^{2^{n+1}}$
2	$g_2 = 2^{2^n}$
3	$g_3 = (n+1)!$
4	$g_4 = n!$
5	$g_5 = e^n$
6	$g_6 = n2^n$
7	$g_7 = 2^n$
8	$g_8 = (\frac{3}{2})^n$
9	$g_9 = n^{lg \text{ lg n}}$,$g_{10} = (lgn)^{lgn}$
10	$g_{11} = (lgn)!$
11	$g_{12} = n^3$
12	$g_{13} = n^2$,$g_{14} = 4^{lgn}$
13	$g_{15} = nlgn$,$g_{16} = lg(n!)$
14	$g_{17} = n$,$g_{18} = 2^{lgn}$
15	$g_{19} = (\sqrt{2})^{lgn}$
16	$g_{20} = 2^{\sqrt{2lgn}}$
17	$g_{21} = lg^2n$
18	$g_{22} = lnn$
19	$g23 = \sqrt{lgn}$
20	$g_{24} = lnlnn$
21	$g_{25} = 2^{lg^*n}$
22	$g_{26} = lg^*(lgn)$,$g_{27} = lg^*n$
23	$g_{28} = lg(lg^*n)$
24	$g_{29} = n^{1/lgn}$,$g_{30} = 1$

---

2 作业 (选做部分)

题目 1 (DH Problem 6.13)

Prove a lower bound of $O(N \times \log_2 N)$ on the time complexity of any comparison-based sorting algorithm.

解答：

构造决策树模型：设 $A[1, 2...N]$ 待排序

每个节点（除叶节点）以 (i, j) 进行标记，表示在该节点进行 A_i 与 A_j 的比较，节点的左子树表示当 $A_i \le A_j$ 时的后续操作路径，右子树表示 $A_i > A_j$ 时的后续操作路径，叶节点表示该 N 个数已经排好序，过程结束。显然，从根节点到任意一个可达叶节点的最长简单路径的长度即为该过程中进行比较的次数。因此比较排序算法中比较次数（运行时间）的下界就是决策树高度的下界。

易知，对于 N 个数来说，$N!$ 个所有可能的排列都是叶节点，即可达叶节点至少有 $N!$ 个，设构成的决策树的高度为 h，可达叶节点为 t, 而高度为 h 的决策树叶节点的数目不超过 2^h 个，由此得到不等式 $n! \le t \le 2^h, 2^h \ge N!$, 则有 $h \ge lg(N!)$, 而通过计算可知知 $lg(N!) = O(N \times \log_2 N)$, 所以 h 的下界为 $O(N \times \log_2 N)$

即 a lower bound of $O(N \times \log_2 N)$ on the time complexity of any comparison-based sorting algorithm.

---

3 Open Topics

本次 OT 介绍两种证明问题下界的常用技术。

Open Topics 1 (Decision Trees)
介绍 Decison Trees (决策树) 的概念以及在证明问题下界时的应用 (包括但不限于本次选做题 DH 6.13)。

参考资料:

- Decision tree model @ wiki
- lecture-note by jeffe

Open Topics 2 (Adversary Argument)
介绍 Adversary Argument (对手论证) 的概念以及在证明问题下界时的应用。

- lecture-note by jeffe

4 反馈