AI2615 算法设计与分析	2020-2021 春季学期

Lecture 1 – Introduction

2021 年 2 月 26 日

Lecturer: 张驰豪 *Scribe:* 张宇昊

1 Running Time of Algorithms

Definition 1 (Running Time). The running time of an algorithm for a specific input is the number of *atomic operations* (steps) executed.

Example 2. How to calculate the Fibonacci Number?

$$\forall n \in N, \texttt{Fib}(n) = \begin{cases} \texttt{Fib}(n-1)+\texttt{Fib}(n-2) & n \geq 2 \\ 1 & n = 1 \\ 0 & n = 0 \end{cases}$$

Algorithm 1 Recurrence

```
Function Fib_1(n)
  if n = 0 return 0
  if n = 1 return 1
  if n ≥ 2 return Fib_1(n−1) + Fib_1(n−2)
EndFunction
```

Running time of Algorithm 1 Let $T_1(n)$ be the running time of Algorithm 1 for input number n.

$$T_1(n) = \begin{cases} 1 & n \leq 1 \\ T_1(n-1)+T_1(n-2)+1 & n \geq 2 \end{cases}$$

We have $T_1(n) \geq \text{Fib}(n) = 2^{\Omega(n)}$.

Running time of Algorithm 2 Let $T_2(n)$ be the running time of Algorithm 2 for input number n. The algorithm exactly runs $n-2$ iterations, can we say that $T_2(n) = O(n)$? The answer is no because "addition" is not an atomic operation when n grows large. In fact, if we write $F[n]$ in binary, it is a $O(n)$-length string. Therefore, we need to add two length-n numbers in the worst case, and this requires $O(n)$ operations. operations. Thus, $T_2(n) = O(n^2)$.

Algorithm 2 Non-recursive Algorithm

```
Function Fib_2(n)
    F[0] ← 0, F[1] ← 1
    for i = 2 to n
        F[i] ← F[i−1] + F[i−2]
    return F[n]
EndFunction
```

One can observe that for every $n \geq 2$, it holds that $\begin{pmatrix} \mathrm{Fib}(n) \\ \mathrm{Fib}(n-1) \end{pmatrix} = \begin{pmatrix} 0 & 1 \\ 1 & 1 \end{pmatrix} \begin{pmatrix} \mathrm{Fib}(n-1) \\ \mathrm{Fib}(n-2) \end{pmatrix}$, and therefore $\begin{pmatrix} \mathrm{Fib}(n) \\ \mathrm{Fib}(n-1) \end{pmatrix} = \begin{pmatrix} 0 & 1 \\ 1 & 1 \end{pmatrix}^n \begin{pmatrix} 1 \\ 0 \end{pmatrix}$. Our third algorithm is to compute the matrix multiplication in one iteration.

Algorithm 3 Matrix Power

Compute $\begin{pmatrix} \mathrm{Fib}(n) \\ \mathrm{Fib}(n-1) \end{pmatrix} = \begin{pmatrix} 0 & 1 \\ 1 & 1 \end{pmatrix}^n \begin{pmatrix} 0 \\ 1 \end{pmatrix}$ via iteration.

Running time of Algorithm 3 Let $T_3(n)$ be the running time of Algorithm 3 for input number n. Since the iteration needs to multiply n matrices and each of which involves additions of two $O(n)$-length integers. Therefore $T_3(n) = O(n^2)$.

In fact, the trick of *exponentiation by squaring* can boost the computation of A^n for a matrix A. First assume $n = 2^k$. Then $A^n = A^{2^k}$ can be computed using k matrix multiplications following the recursion:

$$A^{2^k} = \left(A^{2^{k-1}}\right)^2.$$

For those n who are not necessarily the power of 2, we can write it in binary $n = \sum_{i=0}^{\lfloor \log_2 n \rfloor} a_i \cdot 2^i$ and decompose $A^n = A^{\sum_{i=0}^{\lfloor \log_2 n \rfloor} a_i \cdot 2^i} = \prod_{i=0}^{\lfloor \log_2 n \rfloor} \left(A^{2^i}\right)^{a_i}$. Therefore, one requires $O(\log n)$ matrix multiplications to compute A^n.

Algorithm 4 Matrix Power

Use "Exponentiation by Squaring" to calculate $\begin{pmatrix} 0 & 1 \\ 1 & 1 \end{pmatrix}^n$, and get $\mathrm{Fib}(n)$ by $\begin{pmatrix} \mathrm{Fib}(n) \\ \mathrm{Fib}(n-1) \end{pmatrix} = \begin{pmatrix} 0 & 1 \\ 1 & 1 \end{pmatrix}^n \begin{pmatrix} 0 \\ 1 \end{pmatrix}$.

Running time of Algorithm 4 Let $T_4(n)$ be the running time of Algorithm 4 for input number n. *Exponentiation by Squaring* requires $O(\log n)$ multiplications of two 2×2 matrices, and each of the multiplications needs to compute the product of two $O(n)$-length numbers. Let $M(n)$ be the running time of one multiplying task for two n-length number, we can simply have $T_4(n) = O(\log n \cdot M(n))$.

However, since the product of two n-length binary number is of at most length $2n$. We have

$$T_4(n) = O(M(1) + M(2) + M(4) + M(8) + \dots . M(n)) \le O(M(n)), \text{when } M(n) \ge n,$$

where the last inequality can be proved by induction. Therefore $T_4(n) = O(M(n))$.
Finally, we remark that $M(n) = O(n^2)$ with the naive algorithm, and we will learn the *Fast Fourier Transform* in this course that two lenght-n integers can be multiplied using $O(n \log n)$ operations. This gives $T_4(n) = O(n \log n)$.

Algorithm 5 Direct calculation

Directly calculate $\mathrm{Fib}(n) = \frac{1}{\sqrt{5}}(\frac{1+\sqrt{5}}{2})^n - \frac{1}{\sqrt{5}}(\frac{1-\sqrt{5}}{2})^n$.

Running time of Algorithm 5 There is no direct way to calculate $\sqrt{5}$ accurately since it is irrational.

Polynomial-time Algorithm Algorithm 2, Algorithm 3 and Algorithm 4 are polynomial-time algorithms if we encode the input n using *unary number*, namely a string of n 1s.

Why we care polynomial-time algorithm?

1. Efficient: go much slower than exponential.
2. Closed: Closed over composition $(A(B(C(x))))$.
3. Robustness: Robust to machine model. (Randomized Machine: Complexity-theoretic Church-Turing Thesis.)