Tsinghua University

# Homework #2

Student name:

Course: 算法分析与设计 – Professor: 王振波

Due date: *2020 年 10 月 6 日*

**2.6** Consider the following basic problem. You're given an array $A$ consisting of $n$ integers $A[1], A[2], \ldots, A[n]$. You'd like to output a two-dimensional $n$ -by- $n$ array $B$ in which $B[i,j]$ (for $i < j$) contains the sum of array entries $A[i]$ through $A[j]-$ that is, the sum $A[i] + A[i+1] + \cdots + A[j]$. (The value of array entry $B[i,j]$ is left unspecified whenever $i \geq j$, so it doesn't matter what is output for these values.)

(a) For some function $f$ that you should choose, give a bound of the form $O(f(n))$ on the running time of this algorithm on an input of size $n$ (i.e., a bound on the number of operations performed by the algorithm).

(b) For this same function $f$, show that the running time of the algorithm on an input of size $n$ is also $\Omega(f(n))$. (This shows an asymptotically tight bound of $\Theta(f(n))$ on the running time.)

(c) Although the algorithm you analyzed in parts (a) and (b) is the most natural way to solve the problem-after all, it just iterates through the relevant entries of the array $B$, filling in a value for each-it contains some highly unnecessary sources of inefficiency. Give a different algorithm to solve this problem, with an asymptotically better running time. In other words, you should design an algorithm with running time $O(g(n))$, where $\lim_{n\to\infty} g(n)/f(n) = 0$

(a) 对于固定的 $i, j$ 而言，需要 $j-i \quad (j > i)$ 次加法运算，故需要的运算次数为：

$$\begin{aligned}\sum_{i=1}^{n}\sum_{j=i+1}^{n}(j-i) &= \sum_{i=1}^{n}\frac{(n-i)(n-i+1)}{2}\\ &=\frac{1}{2}\sum_{i=1}^{n-1}(i^2+i)\\ &=\frac{1}{6}(n^3-n)\end{aligned}$$

故可以取 $f(n) = n^3$，该算法是 $O(n^3)$ 的。

■

(b) 由 (a) 知运算次数 $T(n) = \frac{1}{6}(n^3 - n)$，很显然若 $n > 3$，我们有 $\frac{1}{12}n^3 \leq T(n) \leq \frac{1}{6}n^3$。故 $T(n) = \Theta(n^3)$

■

(c) 事实上，我们只需要计算矩阵 $B$ 的严格上三角部分。对 $B$ 进行简单观察可知，$B$ 的第 $i$ 行和第 $i+1$ 行的对应元素相比，只多了一个加数 $A[i]$。于是我们可以得到如下算法：

```
Input: 序列 $A[1:n]$，矩阵 $B = 0$
Output: 矩阵 $B$ 满足 $B[i,j] = A[i] + A[i+1] + \cdots + A[j] \quad ,j > i$
1: $B[1,1] = A[1]$
2: for $i = 2 : n$ do
3:     $B[1,i] = B[1,i-1] + A[i]$
4: end for
5: for $i = 2 : n$ do
6:     for $j = i+1 : n$ do
7:         $B[i,j] = B[i-1,j] - A[i-1]$
8:     end for
9: end for
```

上述算法的 $1-4$ 步算出矩阵 $B$ 的第一行，共需要 $n-1$ 次加法运算；$5-9$ 步通过第 $i-1$ 行减去 $A[i-1]$ 得到第 $i$ 行，共需要 $\sum_{i=2}^{n}\sum_{j=i+1}^{n} 1 = \frac{(n-1)(n-2)}{2}$ 次减法运算。故一共只需要 $T'(n) = \frac{(n-1)n}{2}$ 次运算。

此时可取 $g(n) = n^2$，满足 $\lim_{n\to\infty} g(n)/f(n) = 0$。

■

**2.8** You're doing some stress-testing on various models of glass jars to determine the height from which they can be dropped and still not break. The setup for this experiment, on a particular type of jar, is as follows. You have a ladder with $n$ rungs, and you want to find the highest rung from which you can drop a copy of the jar and not have it break. We call this the highest safe rung.

(a) Suppose you are given a budget of $k = 2$ jars. Describe a strategy for finding the highest safe rung that requires you to drop a jar at most $f(n)$ times, for some function $f(n)$ that grows slower than linearly. (In other words, it should be the case that $\lim_{n\to\infty} f(n)/n = 0$.)

(b) Now suppose you have a budget of $k > 2$ jars, for some given $k$ Describe a strategy for finding the highest safe rung using at most $k$ jars. If $f_k(n)$ denotes the number of times you need to drop a jar according to your strategy, then the functions $f_1, f_2, f_3, \ldots$ should have the property that each grows asymptotically slower than the previous one: $\lim_{n\to\infty} f_k(n)/f_{k-1}(n) = 0$ for each $k$.

(a) 我们采取如下策略:

1. 在 $[\sqrt{n}]$ 处释放罐子；
2. 如果坏了，则从 1 到 $[\sqrt{n}]-1$ 逐级测试，直至损坏；
3. 如果没有坏，则将 $[\sqrt{n}]+1$ 号瓶看作 1 号瓶 (此时瓶子为 1 到 $n-[\sqrt{n}]$)，并重复步骤 1.

**我们归纳证明** $f(n) \le 3\sqrt{n}$。显然 $f(1) = 1 \le 3\sqrt{1}, f(2) = 2 \le 3\sqrt{2}$。

假设对于 $k \le n-1$，有 $f(k) \le 3\sqrt{k}$，我们来证明 $f(n) \le 3\sqrt{n}$：

第一次释放，若碎了，则至多还需要 $[\sqrt{n}]-1$ 次；若没碎，则还至多需要 $f(n-[\sqrt{n}])$ 次。从而可知

$$f(n) \le \max\{[\sqrt{n}], f(n-[\sqrt{n}])+1\} \tag{1}$$

很显然 $\max\{[\sqrt{n}], f(n-[\sqrt{n}])+1\} \le 3\sqrt{n}$ 对 $n \in \mathbb{Z}^+$ 均成立，从而 $f(n) \le 3\sqrt{n}$ 总成立，并且 $\lim_{n\to\infty} f(n)/n = 0$。

∎

(b) 类似的，对有 k 个罐子的情形，我们采取如下策略:

1. 在 $[n^{\frac{k-1}{k}}]$ 处释放罐子；

2. 如果坏了，则从 1 到 $[n^{\frac{k-1}{k}}]-1$ 逐级测试，直至损坏；
3. 如果没有坏，则将 $[n^{\frac{k-1}{k}}]+1$ 号瓶看作 1 号瓶 (此时瓶子为 1 到 $n-[n^{\frac{k-1}{k}}]$)，并重复步骤 1.

我们对 $k$ 归纳证明 $f_k(n) \leq (k+1)n^{\frac{1}{k}}$:

$k=1,2$ 的情形我们已经证明，假设对于 $m \leq k-1$，有 $f_m(n) \leq (m+1)n^{\frac{1}{m}}$，而类似于式 (1)，我们有

$$f_k(n) \leq \max\{f_{k-1}([n^{\frac{k-1}{k}}]-1)+1, f_k(n-[n^{\frac{k-1}{k}}])+1\} \tag{2}$$

从而我们只需要证明：

$$k([n^{\frac{k-1}{k}}]-1)^{\frac{1}{k-1}}+1 \leq (k+1)n^{\frac{1}{k}} \tag{3}$$

$$(k+1)(n-[n^{\frac{k-1}{k}}])^{\frac{1}{k}}+1 \leq (k+1)n^{\frac{1}{k}} \tag{4}$$

对于 (3)，我们有

$$k([n^{\frac{k-1}{k}}]-1)^{\frac{1}{k-1}}+1 \leq k(n^{\frac{k-1}{k}})^{\frac{1}{k-1}}+1 = kn^{\frac{1}{k}}+1 \leq (k+1)n^{\frac{1}{k}} \tag{5}$$

对于 (4)，由伯努利不等式可知:

$$\begin{aligned}(k+1)(n-[n^{\frac{k-1}{k}}])^{\frac{1}{k}}+1 &\leq (k+1)(n-n^{\frac{k-1}{k}}+1)^{\frac{1}{k}}+1\\ &= (k+1)n^{\frac{1}{k}}(1-n^{-\frac{1}{k}}-\frac{1}{n})^{\frac{1}{k}}+1\\ (\text{伯努利不等式}) &\leq (k+1)n^{\frac{1}{k}}(1-\frac{1}{kn^{\frac{1}{k}}}-\frac{1}{kn})+1\\ &\leq (k+1)n^{\frac{1}{k}}\end{aligned} \tag{6}$$

从而 $f_k(n) \leq (k+1)n^{\frac{1}{k}}$ 成立。故 $\lim_{n\to\infty} f_k(n)/f_{k-1}(n) = 0$ 成立。 ■

**3.5** A binary tree is a rooted tree in which each node has at most two children. Show by induction that in any binary tree the number of nodes with two children is exactly one less than the number of leaves.

我们对节点数 $n$ 进行归纳，当节点数 $n$ 为 $1,2$ 时结论显然成立，我们假设结论对 $k \leq n-1$ 都成立，下面考虑有 $n$ 个节点的二叉树。

我们选取一个叶节点并将其删除，考虑其父节点，此时有两种情况：

- 父节点原本有两个子节点，现在还剩下一个子节点，此时总的叶节点数减少一个，总的有两个子节点的节点数也减少了一个。而对于 $n-1$ 的情形，由归纳知有两个子节点的节点数比叶节点数少 1，从而这种情况下，结论对于 $n$ 成立；
- 父节点原本只有一个子节点，现在没有子节点了，所以它变成了新的叶节点，此时总的叶节点数和总的有两个子节点的节点数都没有发生变化，由归纳知有两个子节点的节点数比叶节点数少 1，从而这种情况下，结论对于 $n$ 也成立。

■

**3.6** We have a connected graph $G = (V, E)$, and a specific vertex $u \in V$. Suppose we compute a depth-first search tree rooted at $u$, and obtain a tree $T$ that includes all nodes of G. Suppose we then compute a breadth-first search tree rooted at $u$, and obtain the same tree $T$. Prove that $G = T$. (In other $r$ words, if $T$ is both a depth-first search tree and a breadth-first search tree rooted at $u$, then $G$ cannot contain any edges that do not belong to T.)

反证法。我们不妨假设边 $(x, y) \in G$，但 $(x, y) \notin T$。不妨假设在 BFS 算法中，$x$ 先被检索到，那么可知 $y$ 在 $x$ 的同层或者下一层。

现在我们考虑 DFS 算法，我们说明 $x$ 和 $y$ 至少相差 2 层。不妨假设 $x$ 先被检索到，那么由于 $(x, y) \in G$，那么 $y$ 一定在 $x$ 的分支下。但是 $(x, y) \notin T$，所以 $y$ 不可能在 $x$ 的下一层 (否则它会和 $x$ 相连)，所以 $x$ 和 $y$ 至少相差 2 层，这与 BFS 算法中的结论矛盾。■