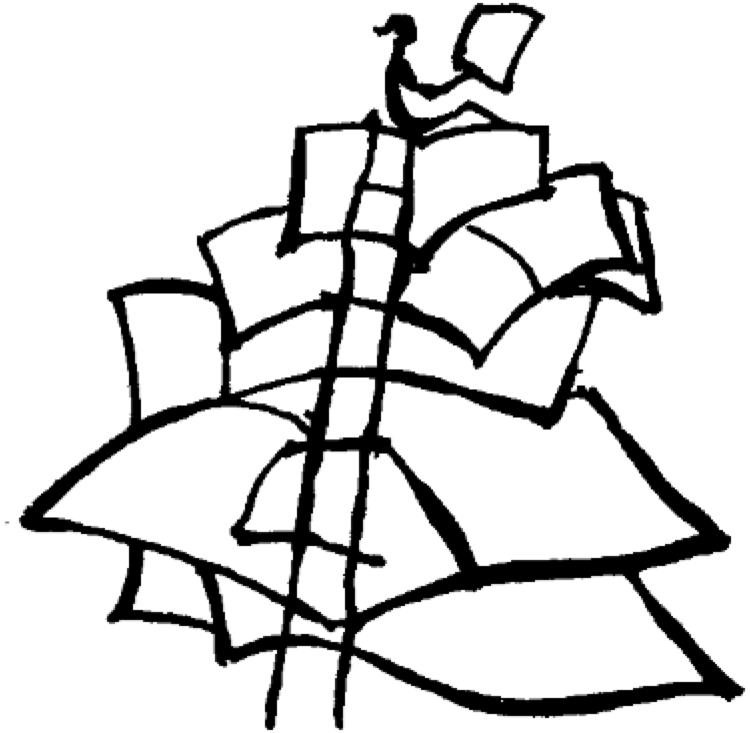
计算机问题求解—论题4.9
随机算法的概念

陶先平

2015年5月05日

问题1: 卷首语是什么含义？它和随机算法有什么关联？

Randomized Algorithms



"For him
who seeks the truth,
an error is nothing unknown."

JOHANN WOLFGANG VON GOETHE

确定性图灵机

一台**图灵机**是一个七元组$(Q, \Sigma, \Gamma, \delta, q_0, q_{accept}, q_{reject})$，其中$Q, \Sigma, \Gamma$都是有限集合，且满足

1. Q是状态集合；
2. Σ是输入字母表，其中不包含特殊的空白符$\square$；
3. $b \in \Gamma$为空白符；
4. Γ是带字母表，其中$\square \in \Gamma$且$\Sigma \subset \Gamma$；
5. $\delta : Q \times \Gamma \rightarrow Q \times \Gamma \times \{L, R\}$是转移函数，其中$L, R$表示读写头是向左移还是向右移；
6. $q_0 \in Q$是起始状态；
7. $q_{accept} \in Q$是接受状态。$q_{reject} \in Q$是拒绝状态，且$q_{reject} \neq q_{accept}$。

可以看出，转换函数决定了这么一个图灵机在每个格局(读写头位置，当前带字母，当前状态）下，其转换新状态是唯一的，计算是唯一的

非确定图灵机

- 唯一的区别：

$$\delta : Q \times \Gamma \to 2^{Q \times \Gamma \times \{L,R\}}$$

例如，设非确定型图灵机M的当前状态为q，当前读写头所读的符号为x，若

$$\delta(q,x) = \{\underline{(q_1,x_1,d_1),(q_2,x_2,d_2),\ldots,(q_k,x_k,d_k)}\}$$

则M将*任意地*选择一个(q_i,x_i,d_i)，按其进行操作，然后进入下一步计算。

This means that a nondeterministic TM (algorithm) may have a lot of <u>computations</u> on an input x, while any deterministic TM (algorithm) has exactly one computation for every input.

问题2：什么是随机算法

A randomized algorithm can be viewed as a nondeterministic algorithm that has a probability distribution for every nondeterministic choice.

问题3：这两句话是一致的吗？

Another possibility is to consider a randomized algorithm as a deterministic algorithm with an additional input that consists of a sequence of random bits.

问题4：你能根据这个描述画出随机算法的图灵机模型的示意图吗？

If one wants to formalize the notion of randomized algorithms, then one can take deterministic Turing machines A with an infinite additional tape. This additional tape is read-only; it contains an infinite random sequence of 0s and 1s, and A may move on it only from the left to the right. Another possibility

问题5：你能根据下个描述写出随机算法的图灵机模型吗？

Another possibility is to take a nondeterministic Turing machine with nondeterministic guesses over at most two possibilities and to assign the probability $\frac{1}{2}$ to every such possibility.

随机算法的概率空间

- $(S_{A,x}, Prob)$:
- $S_{A,x} = \{C | C \text{ is a computation of } A \text{ on } x\}$;
  $Prob \text{ is a distribution over } S_{A,x}$,
- $Prob_{A,x}(C)$: $\text{it is } {}^{1}\!/_{2} \text{ to the power of the number of random bits asked in } C$.
- *Prob(A(x) = y), is the sum of all* $Prob_{A,x}(C)$*, where C outputs y.*

如何评价随机算法：两个随机变量

算法的输出是随机变量

The probability that A **outputs** y **for an input** x, $\boldsymbol{Prob}(A(x) = y)$, is the sum of all $Prob_{A,x}(C)$, where C outputs y. Obviously, the aim of the randomized

算法的时间代价也是随机变量

Let $\boldsymbol{Time}(C)^{10}$ be the time complexity of the run C of A on x. Then the **expected time complexity of** A **on** x is

$$\boldsymbol{Exp\text{-}Time}_A(x) = E[Time] = \sum_C Prob_{A,x}(C) \cdot Time(C),$$

问题5：

随机算法的期望时间复杂度和确定算法的平均复杂度有什么不同？

For every randomized algorithm A we consider a new complexity measure – the number of random bits used. Let $\boldsymbol{Random_A(x)}$ be the maximum number of random bits used over all random runs (computations) of A on x. Then, for every $n \in \mathbb{N}$,

$$\boldsymbol{Random_A(n)} = \max\{Random_A(x) \mid x \text{ is an input of size } n\}.$$

问题6:

这是什么意思?为什么需要这个定义?

问题7：在随机算法分析中，为什么我们需要“output？”？

randomized algorithms may allow infinite runs provided that they occur with a reasonably small probability on any given input.

这两个地方用词的单、复数使用，给你什么启发?

Obviously, a deterministic algorithm is never allowed to take an infinite computation on an input x,

LAS VEGAS算法第一种定义:永远正确

The first approach says that a randomized algorithm A is a **Las Vegas algorithm computing a problem F** if for any input instance x of F,

$$Prob(A(x) = F(x)) = 1,$$

where $F(x)$ is the solution of F for the input instance x. For this definition the time complexity of A is always considered to be the expected time complexity $Exp\text{-}Time_A(n)$.

永远正确

不以worst case为代表来讨论时间开销

Las Vegas算法第二种定义：永不犯错

In the second approach to defining Las Vegas algorithms we allow the answer "?".[15] We say that a randomized algorithm A is a **Las Vegas algorithm computing a problem F** if for every input instance x of F,

多数情况下正确，且：可以保持沉默但永不犯错

$$Prob(A(x) = F(x)) \geq \frac{1}{2},$$

$$Prob(A(x) = \text{"?"}) = 1 - Prob(A(x) = F(x)) \leq \frac{1}{2}.$$

In this second approach one may consider $Time_A(n)$ as the time complexity of A because the nature of this second approach is to stop after $Time_A(|x|)$

the **(worst case) time complexity of A** is

$$\boldsymbol{Time_A(n)} = \max\{Time_A(x) \mid x \text{ is an input of size } n\}$$

Best known Las Vegas algorithm：

Algorithm 5.2.2.2. RQS (RANDOMIZED QUICKSORT)

Input: $a_1, \ldots, a_n$, $a_i \in S$ for $i = 1, \ldots, n$, $n \in \mathbb{N}$.

Step 1: Choose an $i \in \{1, \ldots, n\}$ uniformly at random.
{Every $i \in \{1, \ldots, n\}$ has equal probability to be chosen.}

Step 2: Let A be the multiset $\{a_1, \ldots, a_n\}$.
if $n = 1$ **output**(S)
else the multisets $S_<, S_=, S_>$ are created.

$S_< := \{b \in A \mid b < a_i\}$;
$S_= := \{b \in A \mid b = a_i\}$;
$S_> := \{b \in A \mid b > a_i\}$.

Step 3: Recursively sort $S_<$ and $S_>$.

Output: RQS$(S_<)$, $S_=$, RQS$(S_>)$.

随便问个问题：一般情况下，第一种LAS VEGAS算法的定义适合于计算一个函数，而第二种定义方法适合于判定问题。为什么？

Example 5.2.2.3. Here we consider the problem of finding the kth smallest element of a given set of elements. The idea of the randomized algorithm for this problem is similar to RQS.

Algorithm 5.2.2.4. RANDOM-SELECT(S, k)

Input: $S = \{a_1, a_2, \ldots, a_n\}$, $n \in \mathbb{N}$, and a positive integer $k \leq n$.
Step 1: **if** $n = 1$ **then return** a_1
else $\boxed{\text{choose an } i \in \{1, 2, \ldots, n\} \text{ randomly.}}$
Step 2: $S_< = \{b \in S \mid b < a_i\}$;

We can easily observe that, for every input (S, k), the worst case running time is $\Theta(n^2)$ (i.e., there exists a sequence of random bits leading to $\Theta(n^2)$)

Output: the kth smallest element of S
(i.e., an a_l such that $|\{b \in S \mid b < a_l\}| = k - 1$).

Random select：比较数期望值的上限

- $E[T_{s,k}]$: 在给定集合S和序号k时，随机算法A进行比较的次数的期望值
- 记号$T_{n,k}$: $|S|=n$; 在忽略S的具体意义之后，用于表示$T_{s,k}$
- T_n: $\max\{T_{n,k} \mid 1<=k<=n\}$

期望值的上限：$E[T_n]$的上限

E[T_n]的上限：

算法的第一、二、三步的比较次数

$$T_{|S|,k} \le \boxed{n-1} + \max\{\underline{T_{|S_<|,k}}, \boxed{T_{|S_>|,k-|S_<|-1}}\}$$

如果第k个数落在$S_<$中，递归中比较次数

如果第k个数落在$S_>$中，递归中比较次数

$E[T_n]$的上限：

$$
\begin{aligned}
T_{|S|,k} &\le n - 1 + \max\{T_{|S_<|,k}, T_{|S_>|,k-|S_<|-1}\} \\
&= n - 1 + \max\{T_{j-1,k}, T_{n-j,k-j}\} \\
&\le n - 1 + \max\{T_{j-1}, T_{n-j}\}.
\end{aligned}
$$

算法第一步的a_i选择为第j个小的数的概率是1/n

$$
\begin{aligned}
E[T_n] &\le n - 1 + \frac{1}{n}\sum_{j=1}^{n-1} \max\{E[T_{j-1}], E[T_{n-j}]\} \\
&\le n - 1 + \frac{1}{n}\sum_{j=1}^{n-1} E\left[T_{max\{j-1,n-j\}}\right] \\
&\le n - 1 + \frac{2}{n}\sum_{l=\lceil n/2 \rceil}^{n-1} E[T_l].
\end{aligned}
$$

上限：

Now, we prove $E[T_n] \le 5 \cdot n$ for every n by induction. Obviously, this is true for $n = 1$. Consider that it is true for all $m < n$. Then

$$
\begin{aligned}
E[T_n] &\underset{ind.}{\le} n - 1 + \frac{2}{n} \sum_{l=\lceil n/2 \rceil}^{n-1} 5 \cdot l \\
&\le n - 1 + \frac{10}{n} \left(\sum_{l=1}^{n-1} l - \sum_{l=1}^{\lceil n/2 \rceil - 1} l \right) \\
&= n - 1 + \frac{10}{n} \left(\frac{n \cdot (n-1)}{2} - \frac{(\lceil n/2 \rceil - 1) \cdot \lceil n/2 \rceil}{2} \right) \\
&\le n - 1 + 5(n-1) - 5 \cdot (\lceil n/2 \rceil - 1) \cdot \frac{1}{2} \\
&\le 5 \cdot n.
\end{aligned}
$$

一个关于通信的例子

两台计算机采用通信方式协同计算一个函数：

$$Choice_n : \{0,1\}^n \times \{1,2,\ldots,n\} \to \{0,1\}$$

第一台计算机生成的报文，发给第二台计算机

第二台计算机的输入

Las Vegas One-Way Protocol ($D_{\mathrm{I}}, D_{\mathrm{II}}$)

Input: (x,j), $x = x_1 \ldots x_n \in \{0,1\}^n$, $j \in \{1,\ldots,n\}$.

Step 1: D_{I} chooses a random bit $r \in \{0,1\}$.

Step 2: D_{I} sends the message $c_1c_2\ldots c_{n/2+1} = 0x_1\ldots x_{n/2} \in \{0,1\}^{n/2+1}$ if $r = 0$, and
D_{I} sends the message $c_1c_2\ldots c_{n/2+1} = 1x_{n/2+1}\ldots x_n \in \{0,1\}^{n/2+1}$ if $r = 1$.

Step 3: If $r = 0$ and $j \in \{1,2,\ldots,n/2\}$ then D_{II} outputs $c_{j+1} = x_j$.
If $r = 1$ and $j \in \{n/2+1,\ldots,n\}$ then D_{II} outputs $c_{j-n/2+1} = x_j = Choice(x,j)$.
Else, D_{II} outputs "?".

问题7：

为什么说这个随机算法代价"可证明"好于"任何"确定算法？

几个小问题

- Las Vegas算法会犯错吗？
  - 永远正确
  - 决不犯错
- 你能分别举一个上述算法的代表作吗？
- 既然如此，我们为什么还要引入随机概念？
  - 我们看中Las VEGAS算法的哪一点？

一个小问题：

会犯错的算法，我们为什么还要研究它并尝试接受它？

单边错误Monte Carlo算法

Let L be a language, and let A be a randomized algorithm. We say that A is a **one-sided-error Monte Carlo algorithm** recognizing L if[18]

(i) for every $x \in L$, $Prob(A(x) = 1) \geq 1/2$, and
(ii) for every $x \notin L$, $Prob(A(x) = 0) = 1$.

问题8：何谓one side error?

问题9：为什么说单边Monte Carlo算法非常实用?

两个数据库一致性判断方法：

$$Non\text{-}Eq_n : \{0,1\}^n \times \{0,1\}^n \to \{0,1\}$$

$$Non\text{-}Eq_n(x,y) = 1 \text{ iff } x \neq y.$$

Random Inequality $(R_{\mathrm{I}}, R_{\mathrm{II}})$

Input: $x, y \in \{0,1\}^n$

Step 1: R_{I} chooses uniformly a prime p from the interval $[2, n^2]$ at random. {Note that there are approximately $n^2/\ln n^2$ primes in this interval and so $2\lceil \log_2 n \rceil$ random bits are enough to realize this random choice.}

Step 2: R_{I} computes $s = Number(x) \bmod p$ and sends p and s to R_{II}. {The length of the message is $4\lceil \log_2 n \rceil$ ($2\lceil \log_2 n \rceil$ bits for each of p and s). This is possible because $s \leq p \leq n^2$.}

Step 3: R_{II} computes $q = Number(y) \bmod p$.
If $q \neq s$, then R_{II} outputs 1 ("accept").
If $q = s$, then R_{II} outputs 0 ("reject").

单边错误证明：

首先，有一边不会犯错：

If $x = y$, then $Number(x) \bmod p = Number(y) \bmod p$ for every prime p. So,

$$Prob\left((R_{\mathrm{I}}, R_{\mathrm{II}}) \text{ rejects } (x, y)\right) = 1.$$

其次，另一边犯错的概率小于1/2：

This means that at most $n-1$ primes l from the at least $n^2/\ln n^2$ primes from $\{2, 3, \ldots, n^2\}$ have the property

$$Number(x) \bmod l = Number(y) \bmod l. \tag{5.1}$$

双边错Monte Carla算法

Let F be a computing problem. We say that a randomized algorithm A is a **two-sided-error Monte Carlo algorithm computing F** if there exists a real number ε, $0 < \varepsilon \leq 1/2$, such that for every input x of F

$$Prob(A(x) = F(x)) \geq \frac{1}{2} + \varepsilon.$$

问题10：

既然对错全然不知（只是一个概率），这个算法还有用吗？

重复出现的事情往往反映了真理的存在

Algorithm A_t

Input: x
Step 1: Run the algorithm A on x t times independently and save the t outputs $y_1, y_2, \ldots, y_t$.
Step 2: $y :=$ an output from the multiset $\{y_1, \ldots, y_t\}$ with the property $y = y_i$ for at least $\lceil t/2 \rceil$ different is from $\{1, \ldots, t\}$, if any.
$\{y = ?$ if there is no output with at least $\lceil t/2 \rceil$ occurrences in the sequence $y_1, \ldots, y_t.\}$
Output: y

直观上：不断重复执行，直到输出有意义的y_i

实际上，当我们需要控制双边错算法的正确率时，我们可以预估执行次数：

一次就成功的概率： $p = p(x) \geq 1/2 + \varepsilon$

在t次运行中，恰好成功i次的概率：

$$pr_i(x) = \binom{t}{i} p^i (1-p)^{t-i} = \binom{t}{i} (p(1-p))^i (1-p)^{2(\frac{t}{2}-i)}$$

在t次运行中，恰好成功i次($i \leq \lfloor t/2 \rfloor$)的概率：

$$\leq \binom{t}{i} \cdot \left(\frac{1}{4} - \varepsilon^2\right)^i \left(\frac{1}{4} - \varepsilon^2\right)^{\frac{t}{2}-i} = \binom{t}{i} \cdot \left(\frac{1}{4} - \varepsilon^2\right)^{\frac{t}{2}}.$$

实际上，当我们需要控制双边错算法的正确率时，我们可以预估执行次数：

当算法停止时，在t次运行中，成功的次数一定大于了t/2:

$$
\begin{aligned}
Prob(A_t(x) = F(x)) &\geq 1 - \sum_{i=0}^{\lfloor t/2 \rfloor} pr_i(x) \\
&> 1 - \sum_{i=0}^{\lfloor t/2 \rfloor} \binom{t}{i} \cdot \left(\frac{1}{4} - \varepsilon^2\right)^{t/2} > 1 - 2^{t-1}\left(\frac{1}{4} - \varepsilon^2\right)^{t/2} \\
&= 1 - \frac{1}{2}(1 - 4\varepsilon^2)^{t/2}. \qquad (5.2)
\end{aligned}
$$

意味着：当我们得到一个双边错算法后，如果我们想控制算法求解的准确度必须达到1-δ时：

$$Prob(A_k(x) = F(x)) \geq 1 - \delta \quad \Longrightarrow \quad k \geq \frac{2\ln 2\delta}{\ln(1 - 4\varepsilon^2)}.$$

如何理解下文：

too. Thus, $\boxed{Time_{A_k}(n)} \in O(\boxed{Time_A(n)})$. This means that if A is asymptotically better than the best known deterministic algorithm for F, then A_k is too.

无界错Monte Carlo算法：

双边错算法

$$Prob(A(x) = F(x)) \geq \frac{1}{2} + \varepsilon$$

在双边错算法定义中，如果ε不存在，我们该如何看待这种情况？

随着x的规模的增大，成功的概率和0.5之间的差距趋近于0！

我们针对一个特定的输入，这种算法的重复执行次数将不再实际可控:

$$Time_{A_{k(n)}}(n) = O(2^{2Random_A(n)} \cdot Time_A(n)).$$

作业：

- PAGE 352：5.2.2.7
- PAGE 364：5.2.2.8