算法设计与分析
Algorithms Design & Analysis

第六讲：随机化算法

随机化算法（Randomized Algorithms）

- Average-case analysis of deterministic algorithms(确定性算法的平均情况分析)
- Expected running time of randomized algorithms(随机算法的期望运行时间)

Introduction: 雇佣问题（The Hiring Problem)

- **Problem(问题描述)**
  - Use a job agency to hire a new office assistant(通过中介招聘新职员)
  - Every day we interview a new candidate provided by the agency(一天面试一个中介介绍的候选人)
  - If the candidate is better qualified than our current office assistant, we fire the current assistant and hire the candidate(优胜劣汰)

The Hiring Problem—An Introduction

```
Hire-Assistant(A)
    current ← an infinitely useless dummy
              assistant(初值)
    for i = 1..n
        do if A[i] is better qualified than current
              then current ← A[i]
    return current
```


代价模型（A Cost Model）

- **Every candidate we interview costs us c_i dollars(面试代价)**
  - c_1 dollars fee to be paid to the job agency(中介费用)
  - c_2 dollars travel expenses of the candidate(候选人路费等等) …
- **If we hire the candidate, we have to pay c_h extra dollars(雇佣代价)**
  - c_3 dollars additional charge to be paid to the job agency(因雇佣付给中介的额外费用)
  - c_4 dollars internal administration(雇佣导致的内部管理费用)

代价模型（A Cost Model）

- **Question:** How much should we expect to pay when selecting an office assistant out of n candidates? (n 选一的期望支出为多少?)

- **General answer(问题解答):**
  - We always pay $c_i \cdot n$ dollars for the interviews(面试n个人的费用)
  - We pay $c_h \cdot m$ dollars for hiring people, where m is the number of candidates we hire (and fire) in the process(m个人获得雇佣的费用,比如给中介)
  - $c_i \cdot n + c_h \cdot m$
- **Best case(最好情况):**
  - The first candidate is the best(第一个最优)
  - Then our cost is
  - $c_i \cdot n + c_h$

7

- **Worst case(最坏情况):**
  - The candidates show up in increasing order of qualification(越来越好)
  - Then we hire every candidate(每次面试的结果都是被招聘)
  - The cost is
  - $c_i \cdot n + c_h \cdot n$
- **Abstract problem:** Compute the expected number of candidates we would hire.(问题:期望的m是什么?)

8

概率分析(Probability Analysis)

- To compute expectations, we need a probability distribution over all possible inputs.
- (欲求期望的m,则需要了解输入的分布)
- **For the hiring problem(雇佣问题概率分析):**
  - Assume that no two candidates are equally qualified(互异)
  - Then we can rank the candidates 1, 2, ⋯, n(排序)
  - We assume that every possible order (permutation) of the candidates occurs with equal probability(每种排列的概率相等)
- This is called a ***uniformly random permutation***.(称为一致性随机排列)

9

期望雇佣人数(The Expected Number of Hired Candidates)

- The number m of candidates we hire is a random variable that depends on the given input permutation.(与输入的排列有关)
- Our goal is to compute $E(m)$, the expected value of m.(目标: $E(m)$)

10

指示变量(Indicator Variables)

- For an event E, we can define an indicator variable $X_E = I(E)$: (定义)

$$I(E) = \begin{cases} 1 & \text{if event } E \text{ occurs} \\ 0 & \text{if event } E \text{ does not occur} \end{cases}$$

- Indicator variables help to convert between probabilities and expectations.(作用:概率和期望值之间的转换,见如下定理)
- **Lemma:** *For an event E and its indicator variable $X_E = I(E)$, $Pr(E) = E(X_E)$.*

11

雇佣问题平均情况分析(Average-Case Analysis of Hiring Algorithm)

- **Events:** $E_1, E_2, \cdots, E_n$ (事件)
- E_i = we hire the i-th candidate(第i个受聘用)
- **Indicator variables:** $X_1, X_2, \cdots, X_n$ (指示变量)
- $X_i = I(E_i)$
- Then

12

$$m = \sum_{i=1}^{n} X_i$$

$$E(X_i) = \frac{1}{i}$$

$$E(m) = E\left(\sum_{i=1}^{n} X_i\right)$$

$$= O(\lg n)$$

- **Lemma:**

Assuming that the order in which the candidates are presented is a uniformly random permutation, the expected cost for hiring a new assistant out of n candidates is $\Theta(c_i \cdot n + c_h \cdot \lg n)$*.(期望雇佣代价)*

13

平均情况分析的缺限(The Flaw of Average-Case Analysis)

- **Adversarial Behaviour**
- The hiring algorithm is deterministic(算法确定)
- Every deterministic algorithm has an input that elicits its worst-case behaviour(有最坏的输入)
- An adversary (the job agency) may always provide us with this worst-case input(可能为不良中介利用)

- **The big flaw of average-case analysis: (严重情况)**
- *The input distribution we assume may or may not be correct.(假设分布错误)*

14

解决办法-随机化(Randomization-Beating the Adversary)

- **Modification of our hiring algorithm(改进雇佣算法):**
  - Get the complete list of candidates from the agency(从中介获得所有候选人名单)
  - ***Permute them in a uniformly random order(产生输入的一致性随机排列)***
  - Interview them in this order using our old strategy(再面试)

- **Expected cost(期望代价):** $\Theta(c_i \cdot n + c_h \cdot \lg n)$

15

解决办法-随机化(Randomization-Beating the Adversary)

- **Big difference(不同之处):**
  - No more questionable assumptions(没有争议性假设)
  - No input is guaranteed to elicit a worst-case behaviour(不存在明确的导致最坏情况的输入)
  - The adversary is powerless(削弱了对手的操控能力)
  - 如何实现?

16

一致随机排列生成(Generating Uniformly Random Permutations)

- **Tool:** Random number generator(随机数生成器)
  - random(a, b) generates random integer between a and b in constant time. (产生(a, b)之间的随机整数)

- **Question:** How fast can we generate a random permutation of n elements so that every permutation is equally likely?(如何得到n个元素的一致性随机排列,方法的效率如何?两种方法)

17

排序法排列(Permuting by Sorting)

- **Permute-By-Sorting(*A*)**
- 1 Allocate an array B of size n *(数组)*
- 2 **for** i = 1..n
- 3 **do** $B[i] \leftarrow$ random(1, n^3)
- 4 Sort arrays A, using B as sort keys(排序,以B为参考)

a	b	c	d	e	f	g	h	i	j
340	23	223	55	99	722	182	500	15	88

i	b	d	j	e	g	c	a	h	f
15	23	55	88	99	182	223	340	500	722

Lemma: ***Procedure Permute-By-Sorting takes O(n lg n) time.****(开销)*

Lemma: ***Procedure Permute-By-Sorting produces a uniformly random permutation of array A.*(一致性随机排列)**

18

换位法排列(Permuting in Place)

```
Permute-In-Place(A)
for i = 1..n - 1
    do j ← random(i, n)
        swap A[i] ↔ A[j] (交换)
```

Lemma: *Procedure Permute-In-Place takes linear time* $O(n)$. *开销* $O(n)$.

Lemma: *Procedure Permute-In-Place produces a uniformly random permutation.* .(一致性随机排列)

19

快速排序(Quick Sort)

- Based on the three-step process of divide-and-conquer(分治).
- To sort $A[p..r]$:
  - **Divide:** Partition $A[p..r]$, into two (possibly empty) subarrays $A[p..q-1]$ and $A[q+1..r]$, such that each element in the first subarray $A[p..q-1]$ is $\leq A[q]$ and $A[q]$ is $\leq$ each element in the second subarray $A[q+1..r]$.(根据选定某个中心,将A分成两个部分)
  - **Conquer:** Sort the two subarrays by recursive calls to QUICKSORT.(每一部分用QUICKSORT嵌套处理)
  - **Combine:** No work is needed to combine the subarrays, because they are sorted in place.(组合)

20

```
QUICKSORT(A, p, r)
if p < r
    then q ← PARTITION(A, p, r)
         QUICKSORT(A, p, q − 1)
         QUICKSORT(A, q + 1, r)
```

21

Paratition(A,p,r)

- Perform the divide step by a procedure PARTITION, which returns the index q that marks the position separating the subarrays. (分割，返回分离点的位置q)

22

```
PARTITION(A, p, r)
x ← A[r]
i ← p − 1
for j ← p to r − 1
    do if A[j] ≤ x
        then i ← i + 1
             exchange A[i] ↔ A[j]
exchange A[i + 1] ↔ A[r]
return i + 1
```

23

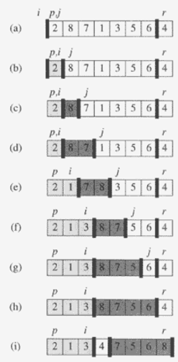


Figure 7.1 The operation of PARTITION on a sample array. Lightly shaded array elements are all in the first partition with values no greater than x. Heavily shaded elements are in the second partition with values greater than x. The unshaded elements have not yet been put in one of the first two

24

Partition(A,p,r)解释

- PARTITION always selects the last element *A*[*r*] in the subarray *A*[*p* .. *r*] as the ***pivot***—the element around which to partition.(选择最后一个元素作为分割中心)
- As the procedure executes, the array is partitioned into four regions, some of which may be empty(分割过程出现四个区域)
  1. All entries in *A*[*p* .. *i*] ≤ pivot.
  2. All entries in *A*[*i*+1 .. *j*-1] > pivot.
  3. *A*[*j* . . *r*-1]
  4. *A*[*r*] = pivot.

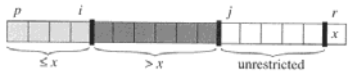
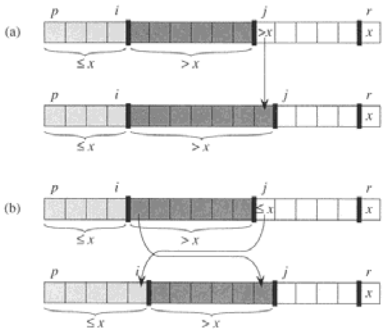


Figure 7.2 The four regions maintained by the procedure PARTITION on a subarray $A[p..r]$. The values in $A[p..i]$ are all less than or equal to x, the values in $A[i+1..j-1]$ are all greater than x, and $A[r] = x$. The values in $A[j..r-1]$ can take on any values.



两种情况

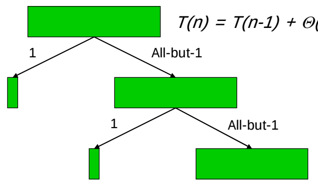
Figure 7.3 The two cases for one iteration of procedure PARTITION. **(a)** If $A[j] > x$, the only action is to increment j, which maintains the loop invariant. **(b)** If $A[j] \le x$, index i is incremented, $A[i]$ and $A[j]$ are swapped, and then j is incremented. Again, the loop invariant is maintained.

快速排序分析：最坏情况
Quicksort Analysis: Worst Case

When does worst case happen? 最坏情况何时发生

Pivot is always smallest or largest remaining element.
(分割中心选择了最大或最小)

$T(n) = T(n\text{-}1) + \Theta(n) = \Theta(n^2)$

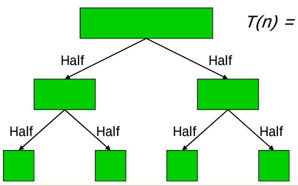


最好情况(Best Case)

When does best case happen?（最好情况何时发生）

Pivot is always median element.
(分割中心每次都选择在中值)

$T(n) = 2{\times}T(n/2) + \Theta(n) = \Theta(n \lg n)$



平均情况(Average Case)

- Average case is more like the best case than the worst case.(平均情况接近最好情况)
- **如何实现?**

随机化快速排序(Randomized Quicksort)

```
Quicksort(A, p, r)
    if p < r
    then swap A[r] ↔ A[random(p, r)] （随机交换）
         q ← Partition(A, p, r)
         Quicksort(A, p, q – 1)
         Quicksort(A, q + 1, r)
```

14	27	3	75	41	4	13	65	23	54

Partition

14	3	4	13	23	27	75	65	54	41

Sort recursively

3	4	13	14	23	27	41	54	65	75

31

快速排序运行时间(The Running Time of Quicksort)

```
Quicksort(A, p, r)
    if p < r
    then swap A[r] ↔ A[random(p, r)]
         q ← Partition(A, p, r)
         Quicksort(A, p, q – 1)
         Quicksort(A, q + 1, r)
```

```
Partition(A, p, r)
    x ← A[r]
    i ← p – 1
    for j = p..r – 1
    do if A[j] ≤ x
        then i ← i + 1
            swap A[i] ↔ A[j]
    swap A[i + 1] ↔ A[r]
    return i + 1
```

- **Lemma:** *Let X be the total number of comparisons performed by Quicksort. Then the running time is O(n + X). (X: Quicksort作比较的总次数)*
- It suffices to show that $E(X) = O(n \lg n)$.

32

比较次数计算(Counting Comparisons)

- Let $Z_{ij} = \{z_i, z_{i+1}, \cdots, z_j\}$
- **Observation:** *Any two elements z_i and z_j are compared at most once.(任意两个元素至多比较一次，指示变量为)*

$$X_{ij} = \begin{cases} 1 & \text{if } z_i \text{ and } z_j \text{ are compared} \\ 0 & \text{if } z_i \text{ and } z_j \text{ are not compared} \end{cases}$$

$$\text{Then } X = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} X_{ij}$$

33

The Expected Number of Comparisons

$$E(X) = E\left(\sum_{i=1}^{n-1} \sum_{j=i+1}^{n} X_{ij}\right) = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} E(X_{ij}) = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} Pr(z_i \text{ and } z_j \text{ are compared})$$

34

两个元素的比较概率(The Probability of Comparing Two Elements)

- If this pivot is neither z_i nor z_j, then z_i and z_j will never be compared.
- If this pivot is z_i or z_j, then z_i and z_j are compared when partitioning the chunk containing Z_{ij} around the pivot.

$$Pr(z_i \text{ and } z_j \text{ are compared}) = Pr(z_i \text{ or } z_j \text{ is the first pivot chosen from } Z_{ij}) = \frac{2}{j-i+1}$$

35

期望比较次数(The Expected Number of Comparisons)

$$E(X) = E\left(\sum_{i=1}^{n-1} \sum_{j=i+1}^{n} X_{ij}\right) = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} E(X_{ij}) = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} Pr(z_i \text{ and } z_j \text{ are compared}) = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} \frac{2}{j-i+1} = \sum_{i=1}^{n-1} O(\lg n) = O(n \lg n)$$

Lemma: *The expected running time of random Quicksort is $O(n \lg n)$.*

36

随机算法分类

- Las Vegas算法：随机算法总是或者给出正确的解，或者无解。
- Monte Carlo算法：居多能够给出正确的解，偶尔产生错误的解。可以采取措施使产生错误解的概率降低到任意的程度。

测试串的相等性

- 问题描述

信道

A：长串x　　x=y？　　B：长串y

测试方法

- A将x发送给B；或B将y发送给A，然后判断x=y？
  缺点：资源浪费
- A从x中取出一个短得多的串作为x的“指纹”发送给B，B用同样的方法获得y的“指纹”，如果两者的“指纹”相同，则认为x=y，否则x=y 不成立。
  特点：节约了资源，但理论上不能100%正确。

“指纹”生成

- 对于一个串w，设I（w）是比特串w表示的一个整数，一种产生“指纹”的方法是选择一个素数p，通过“指纹”函数产生。

$$I_p(x) = I(x) \pmod p$$

$I_p(x) \neq I_p(y)$，则$x \neq y$

$I_p(x) = I_p(y)$，则$x = y$ （？）

测试串相等性算法

1、A从小于M的素数集中随机选择p；

2、A将p和$I_p(x)$发送给B；

3、B检查是否$I_p(x) = I_p(y)$，确定x和y是否相等。

失败概率分析

- 失败概率是：

$$\frac{\left|\{p \mid 素数p < 2^n 且p整除 I(x)-I(y)\}\right|}{\pi(M)} \le \frac{\pi(n)}{\pi(M)}$$

n 是x的二进制的表示形式的位数

$\pi(n)$是小于 n 的不同素数的个数

$\pi(n) \to n/\ln(n)$

问题：
如何降低失败的概率?