CS483 Design and Analysis of Algorithms

Lectures 4-5 Randomized Algorithms

Instructor: Fei Li

lifei@cs.gmu.edu with subject: CS483

Office hours: STII, Room 443, Friday 4:00pm - 6:00pm or by appointments

Course web-site:

http://www.cs.gmu.edu/∼lifei/teaching/cs483_fall08/

Figures unclaimed are from books "Algorithms" and "Introduction to Algorithms"

Announcements

1. Assignment 3 is updated (with 2 more questions).
2. Lecture note 3 is updated.
3. Assignment 3 is due on September 24, next Wednesday.
4. Solutions of the first 3 assignments will be given in class (September 24). Thus, late submission will not be accepted. Please inform your classmates.
5. Regarding to Assignment 4 to Assignment 8: Each assignment deserves 6 points.

Chapter 5 of CLRS — Probabilistic Analysis and Randomized Algorithms

1. Example
2. Indicator Random Variables
3. Probabilistic Analysis
4. Randomized Algorithms

Example — Hiring Problem

Need an office assistant and minimize the cost

a. Agency sends one candidate every day — pay C_i
b. Interview the person, either hire him/her (and fire the old one), or keep old one — pay $C_h \gg C_i$
c. Always want the best person — hire if interviewee is better than current person

Solution

?

Example — Hiring Problem

```
function hire(n)

    best = 0;

    for (i = 1 to n)
        interview candidate i;
        if i is better than best
            best = i;
            hire i;
```

m people hired. Total cost

$$n \cdot C_i + m \cdot c_h$$

Worst-case cost: ?

Best-case cost: ?

Probabilistic Analysis

```
function hire(n)

    best = 0;

    for (i = 1 to n)
        interview candidate i;
        if i is better than best
            best = i;
            hire i;
```

m people hired. Total cost

$$n \cdot C_i + m \cdot c_h$$

Average-case cost

1. Assume applicants come in random order
2. Each permutation of applicants is equally likely

Probabilistic Analysis versus Randomized Algorithms

1. Randomness internal to the algorithm or not
2. Input distribution is known or not

(CLSR 5.1-3)

- A `Random-Number-Generator` outputs 1 with some probability p and 0 with probability $1 - p$, where $0 < p < 1$. You do not know p.
- Give an algorithm that uses `Random-Number-Generator` as a subroutine, and returns an unbiased answer, returning 0 with probability $1/2$ and 1 with probability $1/2$
- What is the running time of your algorithm?

Probability Review — Indicator Random Variable

Probability Review — Indicator Random Variable

1. Indicator variable associated with event A in sample space $\{H,\ T\}$. It converts between *probabilities* and *expectations*.

$$I\{A\} = \begin{cases} 1, & \text{if } A \text{ occurs,} \\ 0, & \text{otherwise.} \end{cases}$$

Probability Review — Indicator Random Variable

1. Indicator variable associated with event A in sample space $\{H,\ T\}$. It converts between *probabilities* and *expectations*.

$$I\{A\} = \begin{cases} 1, & \text{if } A \text{ occurs,} \\ 0, & \text{otherwise.} \end{cases}$$

2. X is a random variable in a sample space $\{H,\ T\}$. X_H is the expected number of H in flipping coins

$$X_H = I\{X = H\} = \begin{cases} 1, & \text{if } H \text{ occurs,} \\ 0, & \text{otherwise.} \end{cases}$$

$$\begin{aligned} E[X_H] &= E[I\{X = H\}] \\ &= 1 \cdot \Pr\{H\} + 0 \cdot \Pr\{T\} \\ &= 1 \cdot (1/2) + 0 \cdot (1/2) \\ &= 1/2 \end{aligned}$$

Probability Review

Lemma

Given a sample space S and an event A in the sample space S, let $X_A = I\{A\}$ (X_A is an indicator random variable). Then $E[X_A] = \Pr\{A\}$.

Proof.

By the definition of an indicator random variable and the definition of expected value, we have

$$\begin{aligned} E[X_A] &= E[I\{A\}] \\ &= 1 \cdot \Pr\{A\} + 0 \cdot \Pr\{\bar{A}\} \\ &= \Pr\{A\} \end{aligned}$$

□

Linearity of expectation

$$E[X + Y] = E[X] + E[Y]$$

Let $X_i = I\{\text{the } i\text{th flip results in event } H\}$
Let X be the random variable denoting the total number of heads in n coin flips

$$X \quad = \quad ? \quad \sum_{i=1}^{n} X_i$$

Practice — Hiring Problem

Let X_i be the indicator variable associated with the event that ith candidate is hired

$$X_i = I\{\text{candidate } i \text{ is hired}\} = \begin{cases} 1, & \text{if candidate } i \text{ is hired,} \\ 0, & \text{otherwise.} \end{cases}$$

Let X be the random variable whose value equals the number of times we hire a new assistant

$$E[X] = \sum_{i=1}^{n} x \cdot \Pr\{X = x\}$$

$$\begin{aligned} X &= X_1 + X_2 + \ldots + X_n \\ E[X_i] &= \Pr\{\text{candidate } i \text{ is hired}\} = ? \end{aligned}$$

Practice — Hiring Problem

$$
\begin{aligned}
E[X] &= E[\sum_{i=1}^{n} X_i] \\
&= \sum_{i=1}^{n} E[X_i] \\
&= \sum_{i=1}^{n} (1/i) \\
&= \ln n + O(1)
\end{aligned}
$$

Theorem

Assuming the candidates are presented in a random order (i.e., we know the distribution), algorithm `HIRE` *has a total (expected) hiring cost of* $O(c_h \cdot \ln n)$

Randomized Algorithms

Instead of assuming a distribution, we impose a distribution, which is internal to the algorithm

```
function rand-hire(n)

    random permutate the list of candidates;

    best = 0;

    for (i = 1 to n)
        interview candidate i;
        if i is better than best
            best = i;
            hire i;
```

Theorem

Assuming the candidates are presented in a random order (i.e., we know the distribution), algorithm `RAND-HIRE` *has a total expected hiring cost of* $O(c_h \cdot \ln n)$

Producing a Uniform Random Permutation

Definition

A **uniform random permutation** is one in which each of the $n!$ possible permutations are equally likely

Definition

Given a set of n elements, a k-**permutation** is a sequence containing k of the n elements. There are $\frac{n!}{(n-k)!}$ such k-permutations

Producing a Uniform Random Permutation

Definition

A **uniform random permutation** is one in which each of the $n!$ possible permutations are equally likely

Definition

Given a set of n elements, a k-**permutation** is a sequence containing k of the n elements. There are $\frac{n!}{(n-k)!}$ such k-permutations

```
function permutate-by-sorting(A)

    n = length[A];

    for i = 1 to n
        P[i] = RANDOM(1, n^3);  Choose a number between 1 and n^3

    sort A, using P as sort keys;

    return A;
```

Producing a Uniform Random Permutation

```
function permutate-by-sorting(A)

    n = length[A];

    for i = 1 to n
        P[i] = RANDOM(1, n^3);  Choose a number between 1 and n^3

    sort A, using P as sort keys;

    return A;
```

Lemma

Procedure `permutate-by-sorting` *produces a uniform random permutation of the input, assuming that all priorities are distinct.*

Proof.

?

What is the running time?

Background: Mathematical Induction

Theorem

Principle of Mathematical Induction. *Let P be a property of positive integer such that:*

1. Base case*: $P(1)$ is true; and*
2. Induction step*: If $P(n)$ is true, then $P(n+1)$ is true.*

Then $P(n)$ is true for all positive integers.

Exercise

Prove that the sum of the n first odd positive integers is n^2, i.e.,
$1 + 3 + 5 + \ldots + (2 \cdot n - 1) = n^2$

Proof.

?

□

Exercise

Find the largest number $R(n)$ of regions in which the plane can be divided by n straight lines.

Proof.

?

□

Background: Mathematical Inductions

1. **Generalization of the base case.**

Exercise

Prove that $2 \cdot n + 1 \leq 2^n$, *for* $n \geq 3$.

Background: Mathematical Inductions

1. **Generalization of the base case**.

Exercise

Prove that $2 \cdot n + 1 \leq 2^n$, *for* $n \geq 3$.

2. **Strong form of mathematical induction**.

Theorem

Let P *be a property of positive integers such that*

1. Base case*:* $P(1)$ *is true, and*
2. Induction step*: If* $P(k)$ *is true for all* $1 \leq k \leq n$, *then* $P(n+1)$ *is true.*

Then $P(n)$ *is true for all positive integers.*

Exercise

Prove that every integer $n \geq 2$ *is prime or a product of primes.*

Proof.

?

Background: Loop Invariant Proof

We use loop invariant to help us understand why an algorithm is correct. We must show three things about a loop invariant:

1. **Initialization**. It is true prior to the first iteration of the loop.
2. **Maintenance**. If it is true before an iteration of the loop, it remains true before the next iteration.
3. **Termination**. When the loop terminates, the invariant gives us a useful property that helps show that the algorithm is correct. This property is perhaps the most important one, since we are using the loop invariant to show correctness. It also differs from the usual use of mathematical induction, in which the inductive step is used infinitely; here, we stop the "induction" when the loop terminates.

Producing a Uniform Random Permutation

Definition

A **uniform random permutation** is one in which each of the $n!$ possible permutations are equally likely

Definition

Given a set of n elements, a k-**permutation** is a sequence containing k of the n elements. There are $\frac{n!}{(n-k)!}$ such k-permutations

```
function rand-in-space(A)

    n = length[A];

    for i = 1 to n
        swap A[i] with A[RANDOM(i, n)];
```

Proof Using Loop Invariants

Remark

For each possible $(i-1)$-permutation, the sub-array $A[1,2,\ldots,i-1]$ contains this $(i-1)$-permutation with probability $\frac{(n-i+1)!}{n!}$

Proof.

? **Maintenance.** We assume that just before the $(i-1)$-st iteration, each possible $(i-1)$-permutation appears in the sub-array $A[1,\ldots,i-1]$ with probability $\frac{(n-i+1)!}{n!}$. We will show that after the ith iteration, each possible i-permutation appears in the sub-array $A[1,\ldots,i]$ with probability $\frac{(n-i)!}{n!}$. Incrementing i for the next iteration will then maintain the loop invariant
(Continue)

□

Proof Using Loop Invariants

Remark

For each possible $(i-1)$-permutation, the sub-array $A[1, 2, \ldots, i-1]$ contains this $(i-1)$-permutation with probability $\frac{(n-i+1)!}{n!}$

Proof.

Let E_1 denote the event in which the first $i-1$ iterations have created the particular $(i-1)$-permutation $< x_1, \ldots, x_{i-1} >$ in $A[1, \ldots, i-1]$. By the loop invariant, $\Pr(E_1) = \frac{(n-i+1)!}{n!}$. Let E_2 be the event that i-th iteration puts x_i in position $A[i]$

$$\begin{aligned} Pr\{E_2 \cap E_1\} &= \Pr\{E_2|E_1\} \cdot \Pr\{E_1\} \\ &= \frac{1}{n-i+1} \cdot \frac{(n-i+1)!}{n!} \\ &= \frac{(n-i)!}{n!} \end{aligned}$$

□

Online Hiring Problem

Online Hiring Problem

- Hire exactly once + Maximize the probability of hiring the best candidate ?

Online Hiring Problem

- Hire exactly once + Maximize the probability of hiring the best candidate ?
- `function online-hire(k, n)`

```
    best-score = 0;

    for i = 1 to k
        if score(i) > best-score
            best-score = score(i);

    for i = k + 1 to n
        if score(i) > best-score
            return i;

    return n;
```

Analysis of Online Hiring Problem

Let S be the event we succeed in choosing the best-qualified candidate $s_{\max}$
Let S_i be the event we succeed when $s_{\max}$ is the i-th candidate we interviewed

$$\Pr\{S\} = \sum_{i=1}^{n} \Pr\{S_i\} = \sum_{i=k+1}^{n} \Pr\{S_i\}$$

Let B_i be the event $s_{\max}$ is in position i
Let O_i be the event `online-hire` does no choose any in a position $k+1$ to $i-1$

$$\Pr\{S_i\} = \Pr\{B_i \cap O_i\} = \Pr\{B_i\} \cdot \Pr\{O_i\} = \frac{1}{n} \cdot \frac{k}{i-1}$$

?

$$\Pr\{S\} = \frac{k}{n} \cdot \sum_{i=k}^{n-1} \frac{1}{i} = \frac{k}{n} \cdot (\ln n - \ln k - 1) \Rightarrow k = \frac{n}{e}$$