Chapter 14
Randomized Algorithms

11/19/2015

Deterministic Algorithms

Goal: Prove for all input instances the algorithm solves the problem correctly and the number of steps is bounded by a polynomial in the size of the input.

Randomized Algorithms

• In addition to input, algorithm takes a source of random numbers and makes random choices during execution;
• Behavior can vary even on a fixed input;

Motivation for Randomized Algorithms

• Simplicity;
• Performance;
• Reflects reality better (Online Algorithms);
• For many hard problems it helps obtain better complexity bounds when compared to deterministic approaches;

Types of Randomized algorithms

• Las Vegas
• Monte Carlo

Monte Carlo

• The time is limited by an upper bound.
• It may produce incorrect answer!
• We are able to bound its error by probability.
• By running it many times on independent random variables, we can make the failure probability arbitrarily small at the expense of running time.
Monte Carlo Example

- Suppose we want to decide a n-place function always returns zero, i.e.,
  \[ F(x_1, x_2, \ldots, x_n) = 0 \] for all xi?
- We may randomly choose values for xi and see if \( F(x_1, x_2, \ldots, x_n) = 0 \).
- It’s impossible to exhaust all values of xi.
- However, if we have checked enough times and \( F(x_1, x_2, \ldots, x_n) \) is always 0, then we have high probability the answer is true.

Monte Carlo Algorithms

Goal: Prove that the algorithm
- with high probability solves the problem correctly;
- for every input the number of steps is bounded by a polynomial in the input size.

Note: The expectation is over the random choices made by the algorithm.

Las Vegas

- Always gives the true answer.
- Running time is random.
- Running time is bounded.
- Randomized Quicksort is a Las Vegas algorithm.

Las Vegas Algorithms

Goal: Prove that for all input instances the algorithm solves the problem correctly and the expected number of steps is bounded by a polynomial in the input size.

Note: The expectation is over the random choices made by the algorithm.

Probabilistic Analysis of Algorithms

Input is assumed to be from a probability distribution.

Goal: Show that for all inputs the algorithm works correctly and for most inputs the number of steps is bounded by a polynomial in the size of the input.
QuickSort

Select: pick an arbitrary element x in S to be the pivot.

Partition: rearrange elements so that elements with value less than x go to List L to the left of x and elements with value greater than x go to the List R to the right of x.

Recursion: recursively sort the lists L and R.

Worst Case Partitioning of QuickSort

Best Case Partitioning of QuickSort

Average Case of QuickSort

Randomized QuickSort

Randomized-Partition(A, p, r)
1. i \leftarrow \text{Random}(p, r)
2. exchange A[i] \leftrightarrow A[r]
3. return Partition(A, p, r)

Randomized-QuickSort(A, p, r)
1. if p < r
2. then q \leftarrow \text{Randomized-Partition}(A, p, r)
3. \text{Randomized-QuickSort}(A, p, q-1)
4. \text{Randomized-QuickSort}(A, q+1, r)

Randomized QuickSort

- The pivot element is equally likely to be any of input elements.
- For any given input, the behavior of Randomized QuickSort is determined not only by the input but also by the random choices of the pivot.
- We add randomization to QuickSort to obtain for any input the expected performance of the algorithm to be good.
**Expectation**

If a random variable $X$ has probability $p_i$ to be $a_i$, the expected value of $X$ is $E[X] = p_1a_1 + p_2a_2 + \ldots + p_na_n$.

E.g., the expected value of a die is $E[X] = (1+2+3+4+5+6)/6 = 3.5$.

If $X_1, X_2, \ldots, X_n$ are random variables, then

$$E[X_1 + X_2 + \ldots + X_n] = E[X_1] + E[X_2] + \ldots + E[X_n]$$

**Notation**

- Rename the elements of $A$ as $z_1, z_2, \ldots, z_n$, with $z_i$ being the $i$th smallest element (Rank "$i$").
- Define the set $Z_{ij} = \{z_i, z_{i+1}, \ldots, z_j\}$ be the set of elements between $z_i$ and $z_j$, inclusive.

**Expected Number of Total Comparisons in PARTITION**

Let $X_i = I\{z_i \text{ is compared to } z_j\}$ indicator random variable

Let $X$ be the total number of comparisons performed by the algorithm. Then

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

The expected number of comparisons performed by the algorithm is

$$E[X] = \sum_{i=1}^{n} \sum_{j=i+1}^{n} E[X_{ij}]$$

by linearity of expectation

$$= \sum_{i=1}^{n} \sum_{j=i+1}^{n} \Pr\{z_i \text{ is compared to } z_j\}$$

**Comparisons in PARTITION**

Observation 1: Each pair of elements is compared at most once during the entire execution of the algorithm
- Elements are compared only to the pivot point!
- Pivot point is excluded from future calls to PARTITION

Observation 2: Only the pivot is compared with elements in both partitions

$Z_{ij} = \{z_i, z_{i+1}, \ldots, z_j\}$

Elements between different partitions are never compared

**Expected Number of Comparisons in PARTITION**

$\Pr\{Z_i \text{ is compared with } Z_j\}$

$= \Pr\{Z_i \text{ or } Z_j \text{ is chosen as pivot before other elements in } Z_{ij} = 2 / (j-i+1)$

$$E[X] = \sum_{i=1}^{n} \sum_{j=i+1}^{n} \Pr\{z_i \text{ is compared to } z_j\}$$

$$E[X] = \sum_{i=1}^{n} \sum_{j=i+1}^{n} \Pr\{z_i \text{ is compared to } z_j\}$$

$$= O(n \log n)$$
Pattern matching

- **Pattern string**: \( Y \) length: \( m \)
- **Text string**: \( X \) length: \( n \), \( n \geq m \)

To find the first occurrence of \( Y \) as a consecutive substring of \( X \).

Assume that \( X \) and \( Y \) are binary strings.

- e.g. \( Y = 01001 \), \( X = 10100111 \)

- Straightforward method: \( O(mn) \)
- The randomized algorithm: \( O(m+n) \) with a mistake of small probability \( 1/n \).

Fingerprints of binary strings

- Similar to the idea of hashing.
- Let \( p \) be a randomly chosen prime number less than \( 2^{mn} \).
- Notation: \( (x_i)_p = x_i \mod p \)

- **Fingerprints** of \( X(i) \) and \( Y \):
  \[
  B(Y) = B(Y) \mod p = ((y_1 \cdot 2^n \cdot y_2 \cdot 2^{n-1} + \cdots + y_m \cdot 2^0) \mod p)
  \]
  \[
  B_p(Y) = B(Y) \mod p = ((y_1 \cdot 2^n \cdot y_2 \cdot 2^{n-1} + \cdots + y_m \cdot 2^0) \mod p) \mod p
  \]

- If \( X(i) = Y \), then \( B_p(X(i)) = B_p(Y) \), but not vice versa.

Examples for using fingerprints

- Example: \( Y = 10110 \), \( X = 110110 \)
  - Suppose \( m = 5 \), \( n = 6 \), \( t = n - m + 1 = 2 \)
  - Suppose \( p = 3 \).
    - \( B(Y) = 22 \mod 3 = 1 \)
    - \( B_p(Y) = 19 \mod 3 = 1 \)
    - \( X(1) \neq Y \) WRONG!
  - If \( B_p(X(i)) \neq B_p(Y) \), then \( Y \neq X(i) \).
  - If \( B_p(X(i)) = B_p(Y) \), we may do a bit by bit checking, or try different \( p \).

A randomized algorithm for pattern matching

- **Input**: A text \( X = x_1 \ldots x_n \), a pattern \( Y = y_1 \ldots y_m \)
- **Output**:
  - (1) No, there is no consecutive substring in \( X \) which matches with \( Y \).
  - (2) Yes, \( X(i) = x_{i-1} \ldots x_{i+m-1} \) matches with \( Y \) which is the first occurrence in \( X \).

If the answer is “No”, there is no mistake.
If the answer is “Yes”, there is some probability that a mistake is made.
Monte Carlo Random Pattern Matching

**Step 1:** Pick a random prime $p < 2mn^2$ and let $i = 1$.

**Step 2:** If $B(X(i))_p \neq (B(Y))_p$, then go to step 3.

*return X(i) as the answer (probably right).*

**Step 3:** If $i = n-m+1$, return "No, there is no consecutive substring in X which matches with Y."

$$i = i + 1.$$ 

Go to Step 2.

Source of error:

$B(X(i))_p = (B(Y))_p$ but $X(i) \neq Y$ (or $B(X(i)) \neq B(Y)$).

What is the probability for this error?

How often does a mistake occur?

- If a mistake occurs in X and Y(i), then $B(X) - B(Y(i)) = 0$, and $B(X(i))_p = (B(Y))_p$, or $p$ divides $|B(X) - B(Y(i))|$.

- Let $Q = \prod_{i \text{ where } p \text{ divides } |B(X) - B(Y(i))|} |B(X) - B(Y(i))|$.

- $Q < 2^{nm}$

Reason: $B(Y) < 2^m$, and at most $(n-m+1)$ $B(X(i))'$s

Theorem for number theory

- **Theorem:** If $u \geq 29$ and $a < 2^u$, then $a$ has fewer than $\pi(u)$ different prime number divisors, where $\pi(u)$ is the number of prime numbers smaller than $u$, and approximately $\pi(u) = u/\ln(u)$.

- Assume $nm \geq 29$.

  $Q < 2^{nm}$

  $\Rightarrow Q$ has fewer than $\pi(nm)$ different prime number divisors.

- If $p$ is a prime number selected from $\{1, 2, \ldots, M = 2mn\}$, the probability that $p$ divides $Q$ is less than:

  $\frac{x(\ln m)}{\pi(M)} \cdot \frac{(\ln m) / \ln nm}{1}$

Las Vegas Random Pattern Matching

**Step 1:** min = 1; j = 1;

**Step 2:** Pick a random prime $p < 2mn^2$ and let $i = \text{min}$;

**Step 3:** If $B(X(i))_p \neq (B(Y))_p$, then go to step 3;

**Step 4:** if $j = K$ goto Step 6 else $j = j+1$;

**Step 5:** If $i = n-m+1$, return "No, there is no consecutive substring in X which matches with Y."

  $\min = i = i + 1$.

  Go to Step 3.

**Step 6:** If $X(i) = Y$ return $X(i)$ as the answer; else go to step 4

An example for the algorithm

- $Y = 10110$, $X = 100111$, $P_1 = 3$, $P_2 = 5$

  $B_3(Y) = (22)_3 = 1$  
  $B_5(Y) = (22)_5 = 2$  
  $B_3(X(1)) = (19)_3 = 1$  
  $B_5(X(1)) = (19)_5 = 4$  

  $\Rightarrow X(1) \neq Y$

Choose one more prime number, $P_3 = 7$

$B_7(Y) = (22)_7 = 1$

$B_7(X(2)) = (7)_7 = 0$

$\Rightarrow X(2) \neq Y$

Monte Carlo versus Las Vegas

- A Monte Carlo algorithm runs produces an answer that is correct with non-zero probability, whereas a Las Vegas algorithm always produces the correct answer.

- The running time of both types of randomized algorithms is a random variable whose expectation is bounded say by a polynomial in terms of input size.

- These expectations are only over the random choices made by the algorithm independent of the input. Thus independent repetitions of Monte Carlo algorithms drive down the failure probability exponentially.