Chapter 5 Lecture

Randomized Algorithms

Sections 5.1 – 5.3

source: 91.404 textbook Cormen et al.

Analyzing the Hiring Problem

- Worst-Case Analysis:
  - Hire every candidate interviewed
  - How can this occur?
    - If candidates come in increasing quality order
    - Hire n times: total hiring cost = \(O(n\cdot c_0)\)
  - Probabilistic Analysis:
    - Appropriate if information about random distribution of inputs is known
    - Use a random variable to represent cost (or run-time)
    - Find expected (average) cost (or run-time) over all inputs
    - We’ll use this technique to analyze hiring cost...
      - First need to introduce indicator random variables to simplify analysis

Indicator Random Variables

- Indicator random variable \(I\{A\}\) associated with event \(A\):
  \[
  I\{A\} = \begin{cases} 1 & \text{if } A \text{ occurs} \\ 0 & \text{if } A \text{ does not occur} \end{cases}
  \]
- Example: Find expected number of heads when flipping a fair coin
  - Sample space \(S = \{H, T\}\)
  - Random variable \(Y\) takes on values \(H, T\)
    - \(\text{Prob}(H) = \text{Prob}(T) = 1/2\)
  - \(X_n\) counts number of heads in a single flip
  - Expected number of heads in a single coin flip = expected value of \(X_n\)

\[
E[X_n] = E[I\{Y = H\}] = 1 \cdot \text{Pr}(Y = H) + 0 \cdot \text{Pr}(Y = T) = 1 \cdot \left(\frac{1}{2}\right) + 0 \cdot \left(\frac{1}{2}\right) = \frac{1}{2}
\]
**Analyzing the Hiring Problem using Indicator Random Variables (continued)**

- Assume candidates arrive in random order
- \( X = \) random variable modeling number of times we hire
- Without indicator random variables:
  \[
  E[X] = \sum_{x \in \{0, 1, \ldots, n\}} x \Pr(X = x)
  \]
- With indicator random variables:
  \[
  X = X_1 + X_2 + \cdots + X_n
  \]
  \[
  X_i = I\{\text{candidate } i \text{ is hired}\} = \begin{cases} 1 & \text{if candidate } i \text{ is hired} \\ 0 & \text{if candidate } i \text{ is not hired} \end{cases}
  \]

Need to find \( E[X] \) and \( E[X_i] = \Pr\{\text{candidate } i \text{ is hired}\} \) by Lemma 5.1

**Analyzing the Hiring Problem using Indicator Random Variables**

- Candidate 0 is a least-qualified dummy candidate
- Randomness assumption: any of first \( i \) candidates is equally likely to be best-qualified to be
- Candidate is better than probability of \( 1/i \) of being better than each of candidates 1...\( i \)
- Probability candidate \( i \) will be hired is therefore \( 1/i \)

\[
E[X_i] = \Pr\{\text{candidate } i \text{ is hired}\} = 1/i
\]

**Analyzing the Hiring Problem using Indicator Random Variables (continued)**

- \( HIRE-ASSISTANT(n) \)
  - best ← 0
  - for \( i \leftarrow 1 \) to \( n \)
    - do interview candidate \( i \)
    - if candidate \( i \) is better than candidate best
      - then best ← \( i \)
        - hire candidate \( i \)
  - Need to calculate probability that these lines are executed

\[
E[X] = \sum_{i=1}^{n} \Pr\{\text{candidate } i \text{ is hired}\} = \ln n + O(1)
\]
And now for something different...

- **Randomized Algorithm**
  - Put randomness into algorithm itself
  - Make choices randomly (e.g., using coin flips)
  - Use pseudorandom-number generator
  - Algorithm itself behaves randomly
  - Different from using probabilistic assumptions about inputs to analyze average-case behavior of a deterministic (non-random) algorithm
  - Randomized algorithms are often easy to design & implement & can be very useful in practice

Nothing is ever really free...

**Randomized Hiring Algorithm**

**RANDOMIZED-HIRE-ASSISTANT(n)**

randomly permute the list of candidates

- `best ← 0`  \(\triangleright\) candidate 0 is a least-qualified dummy candidate
- for \(i ← 1\) to \(n\)
  - do interview candidate \(i\)
  - if candidate \(i\) is better than candidate `best`
    - then `best ← i`
    - hire candidate `i`

**Goal**: randomly permute the list of candidates

Represent an input using candidate ranks: \(\{\text{rank}(1), \text{rank}(2), \ldots, \text{rank}(n)\}\)

Randomly permuting candidate list creates a random list of candidate ranks.

This step dominates the running time. It takes \(\Omega(n)\) time if pairs of values are computed while sorting.

Now we need to show that PERMUTE-BY-SORTING produces a uniform random permutation of the input, assuming all priorities are distinct.

**Randomized Permutation Algorithm**

**Approach**: assign each element \(A[i]\) a random priority \(P[i]\)

**PERMUTE-BY-SORTING(A)**

- \(n ← \text{length}(A)\)
- for \(i ← 1\) to \(n\)
  - do \(P[i] ← \text{RANDOM}(1, n)^{\text{distinct}}\)
  - sort \(A\), using \(P\) as sort keys
- return \(A\)

Choose a random number between 1 and \(n\) (inclusive) that all priorities are unique.

This also demonstrates the running time. It takes \(\Omega(n)\) time if pairs of values are computed while sorting.
Randomized Permutation
Algorithm Analysis

**Claim:** PERMUTE-BY-SORTING produces a uniform random permutation of the input, assuming all priorities are distinct.

**Proof Sketch:** Consider perm. in which each $A[i]$ has $i$th smallest priority. To show this permutation occurs with probability $1/n!$...

Let $X_i$ be event that element $A[i]$ receives $i$th smallest priority.

Probability that, for all $i$, event $X_i$ occurs is:

$$
\Pr\{X_1 \cap X_2 \cap X_3 \cap \ldots \cap X_{n-1} \cap X_n\} = \Pr\{X_1\} \cdot \Pr\{X_2 \mid X_1\} \cdot \Pr\{X_3 \mid X_2 \cap X_1\} \ldots \Pr\{X_n \mid X_{n-1} \cap \ldots \cap X_1\}
$$

$$
\Pr\{X_1 \cap X_2 \cap X_3 \cap \ldots \cap X_{n-1} \cap X_n\} = \frac{1}{n!} \cdot \frac{1}{n-1} \cdot \frac{1}{2} \cdot \frac{1}{1} = \frac{1}{n!}
$$

To complete the proof, now apply the argument above to any fixed permutation of $\{1, 2, \ldots, n\}$: $\sigma = (\sigma(1), \sigma(2), \ldots, \sigma(n))$

Randomized Permutation
Algorithm Improvement

**Algorithm Improvement:**

**Randomized Permutation**

**Algorithm Analysis**

**Randomized Permutation**

**Algorithm Improvement**

**Invariant:**

Just prior to the $i$th iteration of the for loop, for each possible $(i - 1)$-permutation, the subarray $A[1..i-1]$ contains this $(i-1)$-permutation with probability $(n - i + 1)/n!$

**Initialization:** $i = 1$.

For each possible $0$-permutation, $A[1..0]$ contains this $0$-permutation with probability $(n - i + 1)/n! = n!/n! = 1$. Follows from the fact that $A[0..1]$ is empty and the $0$-permutation has no elements.

**Termination:** $i = n + 1$.

The subarray $A[1..n]$ contains a given $n$-permutation with probability $(n - n)/n! = 0/n! = 1/n!$.

**Maintenance (the hard part):**

**Induction Assumption:** Just prior to the $i$th iteration of the for loop, each possible $(i-1)$-permutation appears in the subarray $A[1..i-1]$ with probability $(n - i + 1)/n!$.

**Proof:** After the $i^{th}$ iteration of the for loop, each possible $i$-permutation appears in $A[1..i]$ with probability $(n - i + 1)/n!$.

**Proof:** Start from the induction assumption. After the $i^{th}$ iteration, $A[1..i]$ will contain a permutation $(x_1, \ldots, x_i)$ followed by the value $x_{i+1}$ placed by the algorithm in $A[i]$. Let $E_i$ denote the event "the first $i$ iterations have created a particular $(i-1)$-permutation $(x_1, \ldots, x_{i-1})$ in $A[1..i-1]$." By the induction assumption (loop invariant), $\Pr(E_i) = (n - i + 1)/n!$. Let $E_{i+1}$ denote the event "the $i^{th}$ iteration puts $x_{i+1}$ in position $A[i]$." But $(x_1, \ldots, x_i)$ occurs in $A[1..i]$ exactly when both $E_i$ and $E_{i+1}$ occur: we need to compute $\Pr(E_i \cap E_{i+1})$. 

**Randomized Permutation**

**Algorithm Improvement**
Randomized Permutation Algorithm Improvement

Maintenance (the hard part 2):

Proof: $Pr(E_1 \cap E_2) = ???.$
$Pr(E_1 \cap E_2) = Pr(E_2 \mid E_1) \cdot Pr(E_1)$, by definition of conditional probability.
$Pr(E_2 \mid E_1) = 1/(n-i+1)$ since $i$ is chosen randomly from the $(n-i+1)$ positions in $A[i..n]$.
We have:

$Pr(E_1 \cap E_2) = Pr(E_2 \mid E_1) \cdot Pr(E_1) = 1/(n-i+1) \cdot (n-i+1)!/n! = (n-i)!/n!$