Lecture 5.

Probabilistic Analysis and Randomized Algorithms

Sultan ALPAR
associate professor, IITU
s.alpar@iitu.edu.kz

Outline

Explain the *differences* between *probabilistic analysis* and *randomized algorithms*

Present the technique of *indicator random variables*

Give another example of the analysis of a randomized algorithm (permuting an array in place)

The hiring problem

Scenario:

- You are using an employment agency to hire a new office assistant.
- The agency sends you one candidate each day.
- You interview the candidate and must immediately decide whether or not to hire that person. But if you hire, you must also fire your current office assistant—even if it's someone you have recently hired.
- Cost to interview is c_i per candidate (interview fee paid to agency).
- Cost to hire is c_h per candidate (includes cost to fire current office assistant + hiring fee paid to agency).
- Assume that $c_h > c_i$.
- You are committed to having hired, at all times, the best candidate seen so far. Meaning that whenever you interview a candidate who is better than your current office assistant, you must fire the current office assistant and hire the candidate. Since you must have someone hired at all times, you will always hire the first candidate that you interview.

Goal: Determine what the price of this strategy will be.

Pseudocode to model this scenario: Assumes that the candidates are numbered 1 to n and that after interviewing each candidate, we can determine if it's better than the current office assistant. Uses a dummy candidate 0 that is worse than all others, so that the first candidate is always hired.

```
HIRE-ASSISTANT(n)
best ← 0        ▷ 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
```

Cost: If n candidates, and we hire m of them, the cost is $O(nc_i + mc_h)$.

- Have to pay nc_i to interview, no matter how many we hire.
- So we focus on analyzing the hiring cost mc_h.
- mc_h varies with each run—it depends on the order in which we interview the candidates.
- This is a model of a common paradigm: we need to find the maximum or minimum in a sequence by examining each element and maintaining a current "winner." The variable m denotes how many times we change our notion of which element is currently winning.

The hiring problem $O(nc_i+mc_h)$

- Interview & hire
- Point: after interviewing candidate i, determine if candidate i is the best candidate you have seen so far

Worst-case analysis

- We hire every candidate that we interview
- Hiring cost: $O(nc_h)$
- We neither have any idea nor have any control over the order of candidates

What we expect to happen in a *typical or average* case?

Probabilistic Analysis(1)

The use of probability in the analysis of problems

Use knowledge (*make assumption*) about the distribution of the inputs

Expected cost/running time

Not applicable to all problems

Random order (imply a total order)-> uniform random permutation: n!

Probabilistic Analysis(2)

In general, we have no control over the order in which candidates appear.

We could assume that they come in a random order:

- Assign a rank to each candidate: $rank(i)$ is a unique integer in the range 1 to n.
- The ordered list $\langle rank(1), rank(2), \ldots, rank(n) \rangle$ is a permutation of the candidate numbers $\langle 1, 2, \ldots, n \rangle$.
- The list of ranks is equally likely to be any one of the $n!$ permutations.
- Equivalently, the ranks form a ***uniform random permutation***: each of the possible $n!$ permutations appears with equal probability.

Essential idea of probabilistic analysis: We must use knowledge of, or make assumptions about, the distribution of inputs.

- The expectation is over this distribution.
- This technique requires that we can make a reasonable characterization of the input distribution.

Randomized Algorithms(1)

Algorithms that make random decisions

That is

- Can generate a random number x from some range $\{1, 2 \ldots R\}$.
- Make decisions based on the value of x
- Instead of *guessing* whether the order is in a random order, we *impose* a random order
- Its behavior is determined not only by its input but also by values produced by a random-number generator

Randomized Algorithms(2)

For the hiring problem: Change the scenario:

- The employment agency sends us a list of all n candidates in advance.
- On each day, we randomly choose a candidate from the list to interview (but considering only those we have not yet interviewed).
- Instead of relying on the candidates being presented to us in a random order, we take control of the process and enforce a random order.

What makes an algorithm randomized: An algorithm is ***randomized*** if its behavior is determined in part by values produced by a ***random-number generator***.

- RANDOM(a, b) returns an integer r, where $a \le r \le b$ and each of the $b - a + 1$ possible values of r is equally likely.
- In practice, RANDOM is implemented by a ***pseudorandom-number generator***, which is a deterministic method returning numbers that "look" random and pass statistical tests.

Indicator Random Variables(1)

Given a sample space and an event A, we define the ***indicator random variable***

$$\mathrm{I}\{A\} = \begin{cases} 1 & \text{if } A \text{ occurs}, \\ 0 & \text{if } A \text{ does not occur}. \end{cases}$$

Lemma

For an event A, let $X_A = \mathrm{I}\{A\}$. Then $\mathrm{E}[X_A] = \Pr\{A\}$.

Proof Letting $\overline{A}$ be the complement of A, we have

$$\begin{aligned} \mathrm{E}[X_A] &= \mathrm{E}[\mathrm{I}\{A\}] \\ &= 1 \cdot \Pr\{A\} + 0 \cdot \Pr\{\overline{A}\} \quad \text{(definition of expected value)} \\ &= \Pr\{A\} . \end{aligned}$$

■ (lemma)

Indicator Random Variables(2)

Provide a convenient method for converting between probabilities and *expectations*

Sample space and event A, associated with a indicator random variable I{A}

Lemma:

- The expected value of an indicator random variable *associated with an event A* is equal to the *probability* that A occurs

Is very useful and convenient for analyzing situations in which we perform *repeated trials*

Analysis of the hiring problem using indicator random variable

- Assume the candidates arrive in a random order
- X: number of times we hire a new office assistant

$X = X_1 + X_2 + \ldots + X_n$ -€ $X_i = \{$*candidate i is hired*$\}$

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

$E[X] = \ln n + O(1)$

Conclusion: Even though we interview n candidates, we only actually hire ln*n* of them

Useful properties:

- $X = X_1 + X_2 + \cdots + X_n$.
- Lemma $\Rightarrow$ $\mathrm{E}[X_i] = \Pr\{\text{candidate } i \text{ is hired}\}$.

We need to compute $\Pr\{\text{candidate } i \text{ is hired}\}$.

- Candidate i is hired if and only if candidate i is better than each of candidates $1, 2, \ldots, i-1$.
- Assumption that the candidates arrive in random order $\Rightarrow$ candidates $1, 2, \ldots, i$ arrive in random order $\Rightarrow$ any one of these first i candidates is equally likely to be the best one so far.
- Thus, $\Pr\{\text{candidate } i \text{ is the best so far}\} = 1/i$.
- Which implies $\mathrm{E}[X_i] = 1/i$.

Hiring Problem

- Lemma 5.2

  Assuming that the candidates are presented in a random order, algorithm HIRE has a total hiring cost of $O(C_h \ln n)$

- Worst-case Running time

  The worst-case hiring cost is $O(nC_h)$. The expected interview cost is a significant improvement over the worst-case cost

Random Algorithms

- Explore the distinction between probabilistic analysis and randomized algorithms
- Instead of assuming a distribution, we *impose* a distribution
- For probabilistic analysis, the algorithm is *deterministic*.
- The number of times we hire differs for different inputs (expensive, inexpensive and moderately expensive inputs)
- For randomized algorithm, the *randomization* is in the algorithm not in the *input distribution*
- For a given input, each time we run the randomized algorithm, we get *different updates* so as to get different cost
- No particular input elicits its *worst-case behavior*

Randomly Permuting arrays

- Randomize the input by permuting the given input array
- Assign each element of the array a random priority *p[i]* and then sort the array according to these priorities
- Randomly permuting produces a uniform random permutation
- Permute the given array in place

Permute-By-Sorting

Permute-By-Sorting (A)

1 n ← length [A]

2 for i ← 1 to n

3. do P [i] = $RANDOM\ (1, n^3)$

4. sort A, using P as sort keys

5. return A

Procedure Permute-By-Sorting produces a *uniform random permutation* of the input in $\Theta(n \lg n)$, assume all priorities are *unique*

$$\Pr(X_1 \cap X_2 \cap X_3 \cap \ldots \cap X_{n-1} \cap X_n)$$

$$= \Pr(X_1) \cdot \Pr(X_2 \mid X_1) \cdot \Pr(X_3 \mid X_1 \cap X_2)$$

$$\cdots \Pr(X_i \mid X_1 \cap X_2 \cap X_3 \ldots \cap X_{i-1}) \cdots$$

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

$$= \frac{1}{n} \cdot \frac{1}{n-1} \cdot \frac{1}{n-2} \cdots 1 = \frac{1}{n!}$$

There are total n! permutations and each permutation appears with 1/n! probability.

Randomize in Place

1. n €length [A]
2. for i € 1 to n
3. do *swap* A [i] €€ A [*RANDOM (i,n)*]

Idea:

- In iteration i, choose $A[i]$ randomly from $A[i \ldots n]$.
- Will never alter $A[i]$ after iteration i.

Time: $O(1)$ per iteration $\Rightarrow$ $O(n)$ total.

$$C_n^k k! = \frac{n!}{k!(n-k)!} k! = \frac{n!}{(n-k)!}$$

$$= n(n-1)\ldots(n-k+1)$$

Correctness: Given a set of n elements, a ***k-permutation*** is a sequence containing k of the n elements. There are $n!/(n-k)!$ possible k-permutations.

Lemma

RANDOMIZE-IN-PLACE computes a uniform random permutation.

Proof of Lemma 5.5

Proof Use a loop invariant:

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

Initialization: Just before first iteration, $i = 1$. Loop invariant says that for each possible 0-permutation, subarray $A[1..0]$ contains this 0-permutation with probability $n!/n! = 1$. $A[1..0]$ is an empty subarray, and a 0-permutation has no elements. So, $A[1..0]$ contains any 0-permutation with probability 1.

Maintenance: Assume that just prior to the ith iteration, each possible $(i-1)$-permutation appears in $A[1..i-1]$ with probability $(n-i+1)!/n!$. Will show that after the ith iteration, each possible i-permutation appears in $A[1..i]$ with probability $(n-i)!/n!$. Incrementing i for the next iteration then maintains the invariant.

Consider a particular i-permutation $\pi = \langle x_1, x_2, \ldots, x_i \rangle$. It consists of an $(i-1)$-permutation $\pi' = \langle x_1, x_2, \ldots, x_{i-1} \rangle$, followed by x_i.

Let E_1 be the event that the algorithm actually puts π' into $A[1..i-1]$. By the loop invariant, $\Pr\{E_1\} = (n-i+1)!/n!$.

Let E_2 be the event that the ith iteration puts x_i into $A[i]$.

We get the i-permutation π in $A[1..i]$ if and only if both E_1 and E_2 occur $\Rightarrow$ the probability that the algorithm produces π in $A[1..i]$ is $\Pr\{E_2 \cap E_1\}$.

Equation (C.14) $\Rightarrow \Pr\{E_2 \cap E_1\} = \Pr\{E_2 \mid E_1\} \Pr\{E_1\}$.

The algorithm chooses x_i randomly from the $n-i+1$ possibilities in $A[i..n]$ $\Rightarrow \Pr\{E_2 \mid E_1\} = 1/(n-i+1)$. Thus,

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

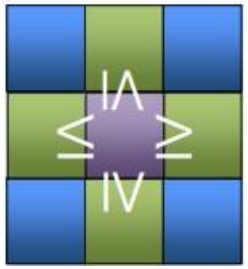
Termination: At termination, $i = n+1$, so we conclude that $A[1..n]$ is a given n-permutation with probability $(n-n)!/n! = 1/n!$. ■ (lemma)

Homework

- 5.1-2,5.2-4
- 5.3-3,
- Problem 5-1

2D Peak Finding

- Given $n \times n$ matrix of numbers
- Want an entry not smaller than its (up to) 4 neighbors:



9	3	5	2	4	9	8
7	2	5	1	4	0	3
9	8	9	3	2	4	8
7	6	3	1	3	2	3
9	0	6	0	4	6	4
8	9	8	0	5	3	0
2	1	2	1	1	1	1