Design & Analysis of Algorithms
COMP 482 / ELEC 420

John Greiner

Math Background: Review & Beyond

1. Asymptotic notation
2. Math used in asymptotics
3. Recurrences
4. Probabilistic analysis

To do:
[CLRS] 5
#2

Not All Algorithms are Deterministic

- Review a little probability
- Extended example:
  - Review probability
  - Analysis technique

Conditional Probability

$$\Pr\{\text{Event}_1 \mid \text{Event}_2\} = \frac{\Pr\{\text{Event}_1 \cap \text{Event}_2\}}{\Pr\{\text{Event}_2\}}$$

Random Variables

What is the probability of getting exactly 1 “tail” when flipping two normal coins?

$\Pr\{X=x\} = \Sigma_{\{s \in S \mid X(s)=x\}} \Pr\{s\}$

X = #Tails, x=1
S = TwoNormalCoinsFlipped = {HH, HT, TH, TT}

Pr{#Tails=1}
= $\Sigma_{\{\text{coins} \in \text{TwoNormalCoinsFlipped} \mid \#\text{Tails(coins)}=1\}}$ Pr{coins}
= Pr{HT} + Pr{TH}
= ¼ + ¼
= ½

Random Variables

Random variables X,Y “independent”
$\leftrightarrow$
$\Pr\{X=x \wedge Y=y\} = \Pr\{X=x\} \times \Pr\{Y=y\}$

Expected values of random variables:
- $E[X] = \Sigma_x (x \times \Pr\{X=x\})$
- $E[X+Y] = E[X]+E[Y]$ linearity of expectation
- $E[a\times X] = a\times E[X]$
- $E[X\times Y] = E[X]\times E[Y]$ if X,Y independent

Hiring Problem

Goal: Hire the most qualified person for a job.

Algorithm we'll use:

1. Decide how many people (n) to interview.
2. Hire the first candidate.
3. For each interviewee, in turn
   a. Interview.
   b. Hire, if better than current employee.

Costs

(n people interviewed $\times$ c_i each) + (m people hired $\times$ c_h each)
$= O(c_i\, n + c_h\, m)$

By definition, $n \geq m$. Assume $c_i \ll c_h$.

? Easy so far, but what is m? ?

7

Hiring Problem: Worst-Case

m=n

In other words,

– Hire every interviewee.
– Interviewees (unluckily) arranged in ascending order of quality.

Cost $= O(c_i\, n + c_h\, n) = O(c_h\, n)$

8

Hiring Problem: Average-Case

What is m likely to be?

Wrong answer: m can be 1…n, so use the mean m=(n+1)/2.

Instead, average cost over all possible interviewee orders.

Number each interviewee 1…n.
Consider each permutation of the set {1,…,n}.

To compute this average, use probabilistic techniques to avoid listing every possible permutation.

9

Averaging via Expected Values

Define: TotalHired $= \Sigma_{i=1\ldots n}$ Hired$?_i$ — *random variable*

Hired$?_i$ = Pr{interviewee #i hired} — *indicator random variable*
= 1 if #i hired, 0 if #i not hired

E[TotalHired] $= E[\Sigma_{i=1..n}$ Hired$?_i]$
$= \Sigma_{i=1..n}$ E[Hired$?_i$]

E[Hired$?_i$] $= 1 \times$ Pr{#i hired} $+ 0 \times$ Pr{#i not hired}
= Pr{#i hired}
= 1/i — Each has equal chance to be the best of first i.

$= \Sigma_{i=1..n}\, 1/i$
$= \ln n + O(1)$

Expected algorithm cost
$= O(c_i\, n + c_h \log n)$

10

On-Line vs. Off-Line Hiring Problem

Let's say the candidates come from an employment agency.

Algorithm's cost is greatly affected by the agency.

– Agency could send best applicants first.
  • Typical of well-run agencies.
  • Waste of money to interview later applicants.
– Agency could send worst applicants first.
  • Maximizes their fees.

We can randomize order, to minimize chance of worst case.

– Protects against poor input distributions. (Accidental or malicious.)
– Can still produce worst case, but only with low probability.

11

Randomizing Order of an Array

Let n = # elements in A
For i = 1 to n, swap A[i] with A[random(i…n)]

Simple algorithm, with an interesting proof that it computes a uniform random permutation.

The key – use a loop invariant:

Before i^{th} iteration, for any (i-1)-permutation of elements from 1…n, A[1…i-1] contains the permutation with probability ((n-i+1)!)/(n!).

12

Randomizing Order of an Array

Let n = # elements in A
For i=1 to n, swap A[i] with A[random(i…n)]

Proving loop invariant:
Before i^{th} iteration, for any (i-1)-permutation of elements from 1…n, A[1…i-1] contains the permutation with probability ((n-i+1)!)/(n!).

Base case, i=1:

A[1..0] contains the 0-permutation (no data) with probability (n!)/(n!)=1.
Holds trivially.

Randomizing Order of an Array

Let n = # elements in A
For i=1 to n, swap A[i] with A[random(i…n)]

Proving loop invariant:
Before i^{th} iteration, for any (i-1)-permutation of elements from 1…n, A[1…i-1] contains the permutation with probability ((n-i+1)!)/(n!).

Inductive case, show holds for i+1:

For an arbitrary i-permutation, the permutation is obtained ↔

- $Event_1$: its first i-1 elements, an (i-1)-permutation, were obtained in the first i-1 iterations, **and**
- $Event_2$: the i^{th} iteration chooses that i-permutation's i^{th} element.

$\Pr\{Event_1 \cap Event_2\} = \Pr\{Event_1\} \times \Pr\{Event_2 \mid Event_1\}$
$= (n-i+1)!/n! \times 1/(n-i+1)$

Inductive hypothesis   Each remaining element equally likely

Randomizing Order of an Array

Let n = # elements in A
For i=1 to n, swap A[i] with A[random(i…n)]

Proving loop invariant:
Before i^{th} iteration, for any (i-1)-permutation of elements from 1…n, A[1…i-1] contains the permutation with probability ((n-i+1)!)/(n!).

Inductive case, show holds for i+1:

For an arbitrary i-permutation, the permutation is obtained ↔

- $Event_1$: its first i-1 elements, an (i-1)-permutation, were obtained in the first i-1 iterations, **and**
- $Event_2$: the i^{th} iteration chooses that i-permutation's i^{th} element.

$\Pr\{Event_1 \cap Event_2\} = \Pr\{Event_1\} \times \Pr\{Event_2 \mid Event_1\}$
$= (n-i+1)!/n! \times 1/(n-i+1)$
$= (n-i)!/n!$ Algebra, cancelling term
$= (n-(i+1)+1)!/n!$ Algebra

Randomizing Order of an Array

Let n = # elements in A
For i=1 to n, swap A[i] with A[random(i…n)]

Proving loop invariant:
Before i^{th} iteration, for any (i-1)-permutation of elements from 1…n, A[1…i-1] contains the permutation with probability ((n-i+1)!)/(n!).

At termination (i=n+1)

Invariant → any n-permutation of elements 1…n occurs in A[1..n] with probability 1/(n!).

I.e., uniform random distribution.

On-Line Hiring Problem

Don't want to interview all candidates.
Instead, interview some until find one "good enough". More realistic.

Algorithm:

1. For each interviewee,
   a. Interview
   b. Hire and stop loop, if good enough.

Cost = $O(c_i\, n + c_h \times 1)$
$= O(c_i\, n)$

Clearly, might not hire the best candidate.

Might hire before we see the best one.
Might not hire the best one, hoping for someone better.

How to define to maximize likelihood of getting the best candidate?

On-Line Hiring Problem

One possible algorithm:

Interview k candidates, and hire the first one after that with a better score than previously found.

```
BestScore := -∞
For i=1 to k,
    BestScore = max(BestScore, Score(i))
For i=k+1 to n,
    If Score(i) > BestScore,
    Then Hire #i & Quit
Hire #n
```

Strategy to find best k:

1. Fix k, & compute the probability of getting best candidate.
2. Find k to maximize probability.

On-Line Hiring Problem

Define: S = succeed in finding best candidate
S_i = succeed in finding best candidate & best is #i

Since S_i's disjoint,
$\Pr\{S\} = \Sigma_{i=1\ldots n} \Pr\{S_i\}$

Since we never hire from the first k,
$\Pr\{S_1\} = \ldots = \Pr\{S_k\} = 0$
$\Pr\{S\} = \Sigma_{i=k+1\ldots n} \Pr\{S_i\}$

Need to determine $\Pr\{S_i\}$ for $i \in k+1\ldots n$.

S_i = candidate #i is best & candidates $k+1\ldots i-1$ not chosen
= candidate #i is best & best among $1\ldots i-1$ is among $1\ldots k$ (not $k+1\ldots i-1$)

Independent

$\Pr\{S_i\} =$? $= 1/n \times k/(i-1)$
$= k/(n(i-1))$

19

On-Line Hiring Problem

Define: S = succeed in finding best candidate
S_i = succeed in finding best candidate & best is #i

$\Pr\{S_i\} = 0$ for $i \in 1\ldots k$
$\Pr\{S_i\} = k/(n(i-1))$ for $i \in k+1\ldots n$

$\Pr\{S\} = \Sigma_{i=k+1\ldots n} \Pr\{S_i\}$
$= \Sigma_{i=k+1\ldots n} k/(n(i-1))$
$= k/n \; \Sigma_{i=k+1\ldots n} 1/(i-1)$
$= k/n \; \Sigma_{j=k\ldots n-1} 1/j$ Renaming: $j=i+1$

? What now? ?

$\geq k/n \int_k^n 1/x \, dx$ Alternative: use harmonic sum.
$= k/n (\ln n - \ln k)$

20

On-Line Hiring Problem

$\Pr\{S\} \geq k/n (\ln n - \ln k)$ Find k to maximize $\Pr\{S\}$.

Differentiate w.r.t. k, and set to zero:
$1/n (\ln n - \ln k - 1) = 0$
$\ln k = \ln n - 1$
$\ln k = \ln n/e$
$k = n/e$

Plug in:
$\Pr\{S\} \geq (n/e)/n (\ln n - \ln (n/e))$
$= 1/e (\ln n - \ln n + \ln e)$
$= 1/e \times 1$
$= 1/e$
≈ 0.368

Algorithm has better than 1/3 chance of getting best candidate.

21