095946- ADVANCED ALGORITHMS AND PARALLEL PROGRAMMING

Fabrizio Ferrandi

a.a. 2021-2022

INTRODUCTION TO RANDOMIZED BASED ALGORITHMS

material from Eric Torng Michigan State University

PROBABILISTIC (AVERAGE-CASE) ANALYSIS

they follow a sequence of steps

Algorithm is deterministic; for a fixed input, it will run the same every time

Analysis Technique

generally we're considering the average case, not the worse case.

- Assume a probability distribution for your inputs
- Analyze item of interest over the distribution

Caveats

- Specific inputs may have much worse performance
- If the distribution is wrong, analysis may give misleading picture

RANDOMIZED ALGORITHM

"Randomize" the algorithm; for a fixed input, it will run differently depending on the result of random "coin tosses"

Randomization examples/techniques (different types for applying randomizations)

- Randomize the order that candidates arrive
- Randomly select a pivot element
- Randomly select from a collection of deterministic algorithms

Key points

- Works well ***with high probability*** on ***every*** input (the algorithm)
- May fail on every input with low probability

KEY ANALYSIS TOOLS

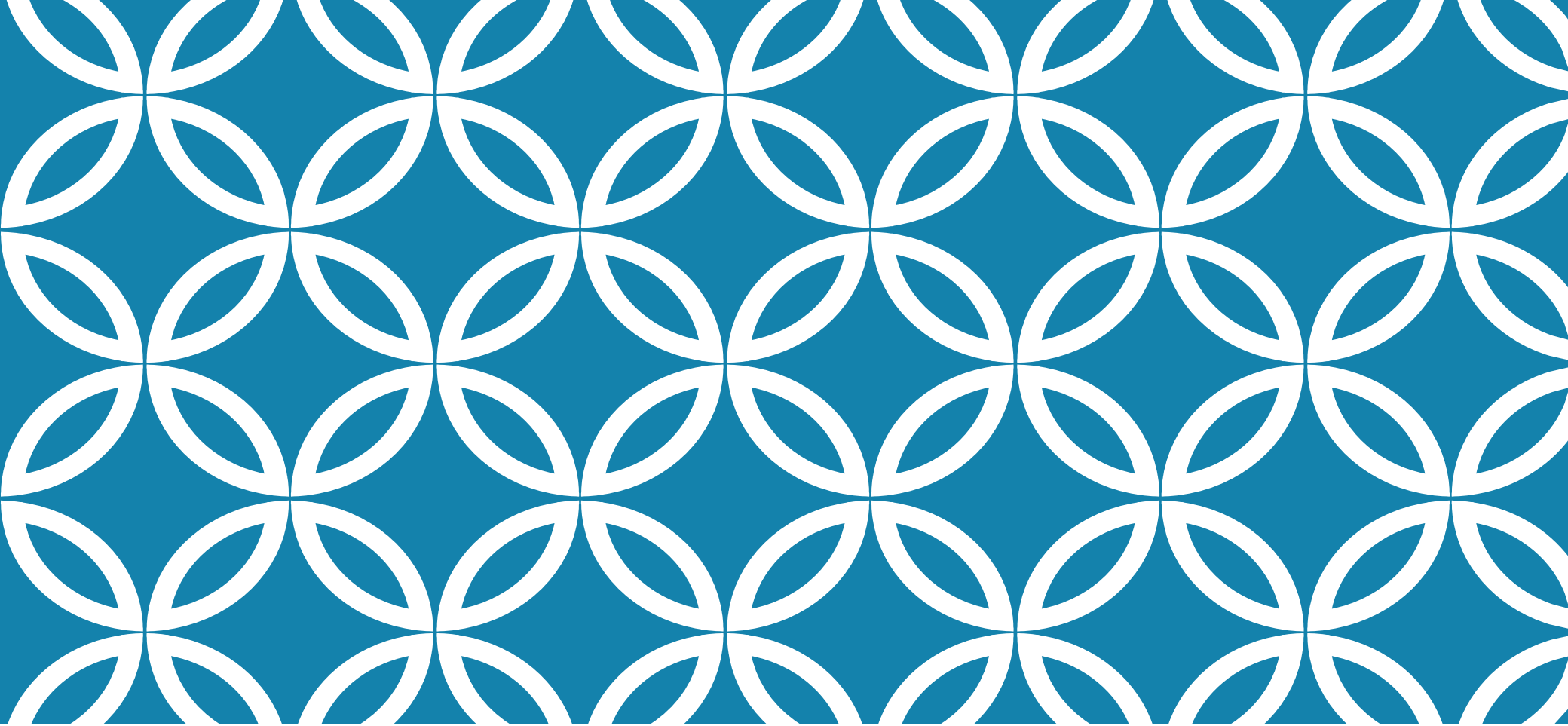
Indicator variables

- Suppose we want to study random variable X that represents a composite of many random events
- Define a collection of "indicator" variables X_i that focus on individual events; typically $X = \Sigma X_i$

Linearity of expectations

- Let X, Y, and Z be random variables s.t. $X = Y + Z$
- Then $E[X] = E[Y+Z] = E[Y] + E[Z]$ (this is expectation)

Recurrence Relations



HIRING PROBLEM AND GENERATING RANDOM PERMUTATIONS

DERIVED FROM MATERIAL PREPARED BY

ANDREAS KLAPPENECKER AND PROF. WELCH

main goal -> try to minimize the cost when you're hiring the a new employee

You need to hire a new employee.

The headhunter sends you a different applicant every day for n days.

If the applicant is better than the current employee, then fire the current employee and hire the applicant.

there is a probab. that the new person is better than the previous one

Firing and hiring is expensive.

How expensive is the whole process?

in the worse case you have to spend N (with N the number of persons) and for each new arrival you have to compute a substitution

in the best case you dont have to substitute none => so it costs 1 => what is the average cost?

Worst case is when the headhunter sends you the n applicants in increasing order of goodness.

Then you hire (and fire) each one in turn: n hires.

Best case is when the headhunter sends you the best applicant on the first day.

Total cost is just 1 (fire and hire once).

What about the average cost?

An input to the hiring problem is an ordering of the n applicants.

There are n! different inputs.

Assume there is some distribution on the inputs

- for instance, each ordering is equally likely
- but other distributions are also possible

Average cost is expected value…

We want to know the expected cost of our hiring algorithm, in terms of how many times we hire an applicant

Elementary event s is a sequence of the n applicants

Sample space is all n! sequences of applicants

Assume uniform distribution, so each sequence is equally likely, i.e., has probability $1/n!$

Random variable X(s) is the number of applicants that are hired, given the input sequence s

What is E[X]?

Break the problem down using indicator random variables and properties of expectation

Change viewpoint: instead of one random variable that counts how many applicants are hired, consider n random variables, each one keeping track of whether or not a particular applicant is hired.

Indicator random variable X_i for applicant i:
1 if applicant i is hired, 0 otherwise

Important fact: $X = X_1 + X_2 + \ldots + X_n$

- number hired is sum of all the indicator r.v.'s

Important fact:

- $E[X_i]$ = Pr["applicant i is hired"]
- Why? Plug in definition of expected value.

Probability of hiring i is probability that i is better than the previous i-1 applicants…

abbiamo 4 applicants e vogliamo vedere cosa succede al terzo.

Suppose n = 4 and i = 3.

In what fraction of all the inputs is the 3rd applicant better than the 2 previous ones?

1234	2134	3124	4123
1243	2143	3142	4132
1324	2314	3214	4213
1342	2341	3241	4231
1423	2413	3412	4312
1432	2431	3421	4321

the red are the best -> 1 is better than 3 and 2 for example

8/24 = 1/3

prob of third applicant to be hired.

General case

In general, since all permutations are equally likely, if we only consider the first i applicants, the largest of them is equally likely to occur in each of the i positions.

Thus $\Pr[X_i = 1] = 1/i$.

Recall that X is random variable equal to the number of hires

Recall that X = the sum of the X_i's (each X_i is the random variable that tells whether or not the i-th applicant is hired)

$E[X] = E[\sum X_i]$

$= \sum E[X_i]$, by property of E

$= \sum Pr[X_i = 1]$, by property of X_i

$= \sum 1/i$, by argument on previous slide

$\leq \ln n + 1$, by formula for harmonic number

so the average case is in between worse and the best one -> it is logn

So average number of hires is ln n, which is much better than worst case number (n).

But this relies on the headhunter sending you the applicants in random order. so we can keep logn

What if you cannot rely on that?

- maybe headhunter always likes to impress you, by sending you better and better applicants

solution: If you can get access to the list of applicants in advance, you can create your own randomization, by randomly permuting the list and then interviewing the applicants.

Move from (passive) probabilistic analysis to (active) randomized algorithm by putting the randomization under your control!

it could be a good tool for solving problems

Instead of relying on a (perhaps incorrect) assumption that inputs exhibit some distribution, make your own input distribution by, say, permuting the input randomly or taking some other random action

On the same input, a randomized algorithm has multiple possible executions

No one input elicits worst-case behavior

Typically we analyze the average case behavior for the worst possible input

Suppose we have access to the entire list of candidates in advance

Randomly permute the candidate list

Then interview the candidates in this random sequence

Expected number of hirings/firings is O(log n)

- *no matter what the original input is*

LAS VEGAS AND MONTE CARLO

these are randomized-based algorithms

LAS VEGAS VS MONTE CARLO

randomized sorting algorithm and the min-cut algorithm exemplify two different types of randomized algorithms

Las Vegas algorithm

- The sorting algorithm always gives the correct solution
  - the only variation from one run to another is its running time

2 different types of randomization

Monte Carlo algorithm

- the min-cut algorithm may sometimes produce a solution that is incorrect

we are able to bound the probability of such an incorrect solution

it produces an approximate solution (that could be accepted) and not the best one(which could require more time obv)

MONTE CARLO ALGORITHMS

For decision problems (problems for which the answer to an instance is YES or NO), there are two kinds of Monte Carlo algorithms:

- those with one-sided error, (only one decision is wrong)
- and those with two-sided error. both decisions can be wrong

A Monte Carlo algorithm is said to have two-sided error if there is a non-zero probability that it errs when it outputs either YES or NO.

It is said to have one-sided error if the probability that it errs is zero for at least one of the possible outputs (YES/NO) that it produces.

LAS VEGAS VS MONTE CARLO

Which is better, Monte Carlo or Las Vegas?

- The answer depends on the application - in some applications an incorrect solution may be catastrophic

A Las Vegas algorithm is by definition a Monte Carlo algorithm with error probability 0

in las vegas there are no errors -> the final algorithm is correct, but it is an unefficient algorithm

A Las Vegas algorithm is an efficient Las Vegas algorithm if on any input its expected running time is bounded by a polynomial function of the input size

A Monte Carlo algorithm is an efficient Monte Carlo algorithm if on any input its worst-case running time is bounded by a polynomial function of the input size

EXAMPLE

Let us consider the problem of finding character ‘a’ in an array of *n* elements.

Input: An array of n elements, in which half are ‘a’s and the other half are ‘b’s.

Output: Find an ‘a’ in the array.

We give two versions of the algorithm, one Las Vegas algorithm and one Monte Carlo algorithm.

LAS VEGAS ALGORITHM

```
findingA_LV(array A, n)
begin
    repeat
        Randomly select one element out of n
elements.
    until 'a' is found
end
```

This algorithm succeeds with probability 1. The running time is random (and arbitrarily large) but its expectation is upper-bounded by O(1)

MONTE CARLO ALGORITHM

```
findingA_MC(array A, n, k)
begin
    i=1
    repeat
        Randomly select one element out of n elements.
        i = i + 1
    until i=k or 'a' is found
end
```

If an 'a' is found, the algorithm succeeds, else the algorithm fails. After *k* times execution, the probability of finding an 'a' is:

$$\Pr(\text{find 'a'}) = 1 - \frac{1}{2^k}$$

This algorithm does not guarantee success, but the run time is fixed. The selection is executed exactly *k* times, therefore the runtime is O(k)