University of Freiburg
Dept. of Computer Science
Prof. Dr. F. Kuhn
P. Bamberger

Algorithm Theory
Sample Solution Exercise Sheet 6

Exercise 1: Balls and Bins *(3+6+3 Points)*

Assume we have n balls and n bins. We want to throw each ball into a bin without any central coordination (like "first ball in first bin"). The aim is to distribute the balls uniformly among the bins, i.e., to keep the maximum number of balls in one bin small. We use the following randomized algorithm:

For each ball, choose a bin uniformly at random.

We want to show that the maximum bin load is $O(\log n)$, with high probability.

a) For a bin i, what is the expected number of balls in i? Proof your answer.

b) Use Chernoff's bound to show that for each bin i, the number of balls in i is $O(\log n)$, w.h.p.

c) Use a union bound to show that the bin with the maximum number of balls contains $O(\log n)$ balls, w.h.p.

Sample Solution

a) For each ball $j \in \{1, \ldots, n\}$ we introduce a random variable X_j which equals 1 if ball j is thrown in bin i and 0 otherwise. Then $Y_i = \sum_{j=1}^{n} X_i$ describes the number of balls in bin i. We obtain

$$E[Y_i] = E\left[\sum_{j=1}^{n} X_i\right] = \sum_{j=1}^{n} E[X_i] = \sum_{j=1}^{n} \Pr(X_i = 1) = n \cdot (1/n) = 1 \ .$$

b) As the X_j are pairwise independent and $E[Y_i] = 1$, by Chernoff's bound we obtain

$$\Pr(Y_i \geq 1 + \delta) \leq \left(\frac{e^\delta}{(1+\delta)^{1+\delta}}\right) \leq \left(\frac{e}{1+\delta}\right)^{1+\delta}$$

For a given $c > 1$ choose $\delta = c \cdot e \ln(n) - 1$. We obtain

$$\Pr(Y_i \geq c \cdot e \ln n) \leq \left(\frac{e}{c \cdot e \ln n}\right)^{c \cdot e \ln n} = \frac{1}{(c \ln n)^{c \cdot e \ln n}} = \frac{1}{n^{c \cdot e \ln(c \ln n)}} \overset{(*)}{\leq} \frac{1}{n^c}$$

$(*)$ for $n \geq 3$

c) The probability for a single bin to contain more than $\ln n$ balls is at most $1/n^c$ and hence the probability that at least one bin contains more than $\ln n$ balls is at most $1/n^{c-1}$.

Exercise 2: Finding Prime Numbers *(10 Points)*

In the lecture, we have seen a randomized primality test which for a number N, tests whether N is a prime. If N is a prime, the test always returns "yes", if N is not prime, the test returns "no" with probability at least $3/4$. The running time of the test is $O(\log^2 N \cdot \log\log N \cdot \log\log\log N)$. Your task now is to find an efficient randomized algorithm that for a given (sufficiently large) input number n, finds a prime number p between n and $2n$, w.h.p. You can assume that the number of primes between n and $2n$ is at least $\frac{n}{3\ln n}$. What is the running time of your algorithm?

Sample Solution

Let $c > 1$. We define an algorithm that finds a prime number between n and $2n$ with probability $1 - \frac{1}{n^c}$ in time $\tilde{O}(\log^5 n)$ where c only influences the hidden constant in the runtime.

We define `test_prime`(N) as the algorithm that repeats the primality test (Miller-Rabin) on N for $\frac{c+3}{2}\log n$ times and outputs "no" if Miller-Rabin outputs "no" in at least one iteration and "yes" otherwise.

Algorithm 1 `test_prime`(N)

for $\frac{c+3}{2}\log n$ times **do**
  if `Miller-Rabin`(N)="no" **then**
    return "no"
return "yes"

If N is prime, `test_prime`(N) returns "yes" and if N is not prime, `test(`N`)` returns "no" with probability at least $1 - 4^{-\frac{c+3}{2}\log n} = 1 - \frac{1}{n^{c+3}}$.

For finding a prime number between n and $2n$ we define the following algorithm.

Algorithm 2 `find_prime`$(n, 2n)$

for $3(c+1)\ln^2 n$ times **do**
  Choose an $N \in \{n+1, \ldots, 2n\}$ uniformly at random
  if `test(`N`)`="yes" **then**
    return N
return "no prime found"

`find_prime` can fail in two ways. It can return "no prime found" or it can return a number that is not a prime.

If `find_prime`$(n, 2n)$ returns "no prime found" we know that all $3(c+1)\ln^2 n$ sampled values are not prime. Hence we obtain

$$\Pr(\texttt{find\_prime}(n,2n) \text{ returns "no prime found"}) \leq \Pr(\text{all } 3(c+1)\ln^2 n \text{ sampled values are not prime})$$
$$\leq \left(\frac{n - \frac{n}{3\ln n}}{n}\right)^{3(c+1)\ln^2 n} = \left(1 - \frac{1}{3\ln n}\right)^{3(c+1)\ln^2 n} \overset{(*)}{\leq} e^{-\frac{3(c+1)\ln^2 n}{3\ln n}} = e^{-(c+1)\ln n} = \frac{1}{n^{c+1}}$$

At $(*)$ we used that $(1-x) \leq e^{-x}$ for all $x \in \mathbb{R}$.

`find_prime` samples at most $3(c+1)\ln^2 n$ non-prime numbers and for each the probability to wrongly return it as a prime is at most $\frac{1}{n^{c+3}}$. Hence, the probability that `find_prime` returns a number that is not a prime is at most

$$3(c+1)\ln^2 n \cdot \frac{1}{n^{c+3}} \overset{(*)}{\leq} \frac{1}{n^{c+1}}$$

$(*)$: For n sufficiently large we have $n \geq 3c\ln^2$.

It follows that the probability that `find_prime` fails in one of the two ways is at most $\frac{2}{n^{c+1}} \leq \frac{1}{n^c}$ (for $n \geq 2$).

The runtime for `test_prime`(N) is $\tilde{O}(\log^3 N)$ and hence the time for testing one $N \in \{n+1, \ldots, 2n\}$ is $\tilde{O}(\log^3 n)$. It follows that `find_prime`$(n, 2n)$ runs in $\tilde{O}(\log^5 n)$

Exercise 3: Minimum Cut *(8 Points)*

You are given a randomized algorithm called ALG that takes as input an undirected graph G and outputs in linear time a number k with the following property:

With probability at least $1/n$, the number k is the size of a minimum cut of G.

Someone now has the idea to increase the probability of getting the size of a minimum cut by running ALG n times and take the minimum of the outputs (which results in an $O(n^2)$ algorithm). Is this a reasonable approach and what is the main difference of the above algorithm to the contraction algorithm discussed in the lecture?

Sample Solution

Repeating ALG n times yields a failure probability of at most $(1-1/n)^n$. As $\lim_{n\to\infty}(1-1/n)^n = 1/e$, we can bound the failure probability only by a constant > 0 and not by a function that converges to 0. In contrast, the randomized contraction algorithm gives a high probability bound.