Algorithm Theory, Tutorial 6

Johannes Kalmbach

University of Freiburg

johannes.kalmbach@gmail.com

January 2020

General Hints

- Contact tutor (`johannes.kalmbach@gmail.com`) for questions concerning corrections etc.
- Contact forum (`daphne.informatik.uni-freiburg.de`) for everything else
- Suggestion: Submit in groups of two (better for understandable algorithms)
- Submit readable solutions (LaTeX as pdf, CLEAN handwriting (+ good scan if necessary))
- Spend enough time on exercise sheets and writeup (you and I have to understand your submission).

Exercise 1

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.

Expected Load for a Single Bin

- Linearity of Expecation:
$$\mathbb{E}[X+Y] = \mathbb{E}[X] + \mathbb{E}[Y]$$
holds **ALWAYS** (no need for independence of $X, Y \ldots$)
- Let X be the number of balls in an arbitrary but fixed bin b.
- Let $X_k = 1$ iff the k-th ball lands in bin b, else 0.
- Then $X = \sum_{i=1}^{n} X_i$ and $\mathbb{E}[X] = \sum_{i=1}^{n} \mathbb{E}[X_i]$
- $E[X_i] =$
- $E[X] =$

Load in fixed bin is $O(\log n)$ w.h.p

-
- Let $E[X], E[X_k]$ as in 1.1
- X_k are independent 0-1 variables (Bernoulli variables), we can apply Chernoff to X.
- We want the probability that $X \notin O(\log n)$ to be $< \frac{1}{n^c}$ for $c > 0$.
- Approach: Estimate $P[X > c \log n]$ for sufficiently large n.
- Chernoff bound: if X is sum of independent Bernoulli varibles then:

$$\forall \delta > 0 : \mathbb{P}(X > (1+\delta)\mathbb{E}[X]) < \left(\frac{e^{\delta}}{(1+\delta)^{1+\delta}}\right)^{\mathbb{E}[X]}$$

$$\forall \delta > 0 : \mathbb{P}(X > (1+\delta)\mathbb{E}[X]) < \left(\frac{e^{\delta}}{(1+\delta)^{1+\delta}}\right)^{\mathbb{E}[X]}$$

$P(X > c \log n)$

Union bound: bin with maximum number of balls contains $O(\log n)$ balls, w.h.p.

- Trick: Biggest bin contains $O(\log n)$ balls $\Leftrightarrow$ all bins contain $O(\log n)$ balls.
- Let E_{max} denote "All bins are small enough" and E_i "bin i is small enough"
- $\mathbb{P}(E_{max}) =$

Finding Primes

- Blackbox from lecture: 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 $3/4$.
- Running time is $O(\log^2 N \cdot \log\log N \cdot \log\log\log N)$.
- Task: 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.
- Hint: The number of primes between n and $2n$ is at least $\frac{n}{3\ln n}$.
- What is the running time of your algorithm?

First Step: Make blackbox work

- For a non-prime, the blackbox fails with probability $< 1/4$.
- How can we turn this into a high probability result?
-
-
-
-
- In Runtime $O($
  we can test a single number for primality with error probability for an

Finding Prime in $[n, \ldots 2n]$, Algorithm 1

- For each number k in $[n, \ldots 2n]$ run MR $O(\log n)$ times. If result is always "yes", return the number, else advance to the next number.
- $\mathbb{P}$(Algorithm doesn't return any number) $=$
- Other failure case: We return a non-prime number. **IMPORTANT**: The test may return a false positive for **any** of the numbers tested before the first actual prime.
- Let $k+1$ be the position of the first prime in $[n, 2n]$ (exists but might come late)
- $E_{wrong} :=$ "Any of the k numbers before the first prime is falsely returned"
- E_i: "The i-th number that is not a prime is falsely returned.
- $\mathbb{P}(E_{wrong}) = \ldots$ space for calculation on next slide.

- Let $k+1$ be the position of the first prime in $[n, 2n]$ (exists but might come late)
- E_{wrong} := "Any of the k numbers before the first prime is falsely returned"
- E_i: "The i-th number that is not a prime is falsely returned.
- $\mathbb{P}(E_{wrong}) =$

Runtime of this algorithm

- Runtime is
- This is linear in n which is somewhat efficient, but can we do better?
- So far we have not (because we couldn't) made use of the following:
- Hint: The number of primes between n and $2n$ is at least $\frac{n}{3 \ln n}$.
- Idea: Sample (asymptotically) less than n numbers, and run our check on them.
- We still do not return a non-prime whp (same argument as before), but we might terminate without having found a return value at all.

Algorithm 2, Sketch

- Sample k numbers from $[n, 2n]$ uniformly at random.
- For each sampled number run MR $O(\log n)$ times. If result is always "yes", return the number, else advance to the next number.
- Return "failure" if we have sampled k numbers without success.
- How many numbers do we have to sample to hit at least one prime w.h.p.

How many samples until we hit a prime

- There are at least $\frac{n}{3 \ln n}$ primes between n and $2n$.
- The probability, that a sampled number is not a prime is $\leq$
- The probability, that k sampled numbers are all no primes is $\leq$
- if this probability is $\leq \frac{1}{n^c}$ we find a prime w.h.p, calculate appropriate k

$$\left(\frac{n-\frac{n}{3\ln n}}{n}\right)^k \leq$$

- If we sample $3c \ln^2 n$ numbers, the possibility of not hitting a prime is $< \frac{1}{n^c}$
- $3c \ln^2 n < n$ for large n, and so the probability of returning a non-prime is also $< \frac{1}{n^c}$
- The probability of the union of those "bad" events is $< \frac{2}{n^c} < \frac{2}{n^{c-1}}$ (for large enough n).
- So if we choose $c > 0$ (which doesn't affect the asymptotic runtime). We also find a prime w.h.p.

- Runtime is now:

Exercise 3

- 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.
- This 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?

- The probability that this Algorithm doesn't output the size of a mincut is
- $<$
- How can we make the algorithm suceed w.h.p.?
-
- The basic contraction algorithm needs $O(n^4 \log n)$ which is worse, why is it still useful?
-