ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Institute of Theoretical Computer Science
Mohsen Ghaffari, Angelika Steger, David Steurer, Emo Welzl, Peter Widmayer

Algorithms, Probability, and Computing **Special Assignment 1** **HS17**

- The solution is due on **Tuesday, October 24, 2017** by **2:15 pm**. Please bring a printout of your solution with you to the lecture. If you cannot attend (and please only then), you may alternatively send your solution as a PDF, likewise **until 2:15pm**, to `njerri@inf.ethz.ch`. We will send out a confirmation that we have received your file. Make sure you receive this confirmation within the day of the due date, otherwise complain timely.
- Please solve the exercises carefully and then write a nice and complete exposition of your solution using a computer, where we strongly recommend to use LaTeX. A tutorial can be found at `http://www.cadmo.ethz.ch/education/thesis/latex`.
- For geometric drawings that can easily be integrated into LaTeX documents, we recommend the drawing editor IPE, retrievable at `http://ipe7.sourceforge.net/` in source code and as an executable for Windows.
- Keep in mind the following premises:
  - When writing in English, write short and simple sentences.
  - When writing a proof, write precise statements.

  The conclusion is, of course, that your solution should consist of sentences that are short, simple, and precise!
- This is a theory course, which means: if an exercise does not explicitly say "you do not need to prove your answer" or "justify intuitively", then a formal proof is **always** required. You can of course refer in your solutions to the lecture notes and to the exercises, if a result you need has already been proved there.
- We would like to stress that the ETH Disciplinary Code applies to this special assignment as it constitutes part of your final grade. The only exception we make to the Code is that we encourage you to verbally discuss the tasks with your colleagues. It is strictly prohibited to share any (hand)written or electronic (partial) solutions with any of your colleagues. We are obligated to inform the Rector of any violations of the Code.
- There will be two special assignments this semester. Both of them will be graded and the average grade will contribute 20% to your final grade.
- As with all exercises, the material of the special assignments is relevant for the (midterm and final) exams.

Exercise 1 — 20 points

From ancestors to parents

We have investigated in the lecture the ancestor relationship of nodes in a random search tree over n keys. Recall that for $i, j \in [n]$ we defined the indicator variable A_i^j as the indicator for the event that node j is an ancestor of node i. Similarly, let us define $P_i^j = P_i^j(n)$ as an indicator variable for the event that node j is a parent of node i in a random search tree with keys $[n]$. We consider that no node is a parent of itself implying that $P_i^i(n) = 0$ for all $i \in [n]$.

(a) By Lemma 1.5 from the lecture notes the expectation $\mathbf{E}[A_i^j]$ only depends on the difference $|i - j|$. In particular, the expectation is symmetric so that $\mathbf{E}[A_i^j] = \mathbf{E}[A_j^i]$ and the expectation is independent of n. By considering small examples, show that these two properties fail for the expectation $\mathbf{E}[P_j^i(n)]$.

(b) For every $n \geq 3$ and every $i \in [n] \setminus \{1, 2\}$ compute the sum $\sum_{k=i}^{n} \frac{1}{k(k-1)(k-2)}$.
HINT: Use the partial fraction decomposition (*Partialbruchzerlegung*).

(c) For every $n \in \mathbf{N}$ and for every $i \in [n]$ compute the probability $Q_{i,n} := \Pr[P_i^1(n) = 1]$.

(d) For $n \geq 2$ what is the expected child of node 1 given that node 1 has a child?

Exercise 2 — 20 points
Sample and factor

The lack of efficient algorithms for factoring numbers is the basis of many cryptographic applications. We consider a variant of the general factoring problem which turns out to be polynomial time solvable: Given a natural number $n \in \mathbf{N}$ the task is to output a number $r \in_{u.a.r.} [n]$ along with the factorization of r into its prime factors. We assume that the input number n is encoded as usual in binary so that the *input size* is $\text{size}(n) := \lceil \log(n+1) \rceil$. Therefore polynomial runtime in this context is $O(\text{poly}(\log n))$. We will use as a black box that the primality of a number $n \in \mathbf{N}$ can be *decided* deterministically in time $O(\text{poly}(\log n))$ with the AKS primality test.[1]

We define below an algorithm SampleAndFactor that solves the sampling problem described above.

Input: A natural number $n \in \mathbf{N}$.
Output: A number $r \in_{u.a.r.} [n]$ and the factorization of r.
SampleAndFactor(n):

1. Let $s_1 \in_{u.a.r.} [n]$ and sample further numbers $s_i \in_{u.a.r.} [s_{i-1}]$ until reaching an index k for which $s_k = 1$. This defines a sequence of numbers $(s_i)_{i=1}^{k}$.
2. Use the AKS primality test on the s_i's (don't test repeated numbers twice) and let r be the product of all s_i's that are prime (an empty product is 1).
3. If $r \leq n$, then with probability $\frac{r}{n}$ return r in its factored form.
4. Otherwise repeat from step 1.

When SampleAndFactor returns, it clearly returns a number and its factorization. Your tasks are to show that it runs in expected polynomial time and that the output number r is uniformly distributed in $[n]$. We have divided these two tasks into subtasks and we start by showing uniformity of the output.

(a) For the purpose of the analysis only let us define an alternative process for sampling a uniformly random number from $[n]$. For $i = 1, \ldots, n$ let c_i be a random variable, a coin, which has the distribution

$$c_i := \begin{cases} H & \text{with probability } \frac{1}{i}, \text{ and} \\ T & \text{with probability } 1 - \frac{1}{i} \end{cases}$$

where H denotes "heads" and T denotes "tails". Now toss the coins in the order $c_n, c_{n-1}, \ldots$ according to their distributions until reaching the first index j so that $c_j = H$. Prove that for any $i \in \{1, \ldots, n\}$ it holds that $\Pr[j = i] = \frac{1}{n}$.

(b) For every prime $p \leq n$ fix a nonnegative integer $\alpha_p \in \mathbf{N}_0$ and define $t := \prod_{p \leq n} p^{\alpha_p}$ where the product is over all primes at most n. Consider one iteration of the steps 1-3 of SampleAndFactor(n). Show that at the end of step 2 it holds that $\Pr[r = t] = \frac{M_n}{t}$ where $M_n := \prod_{p \leq n}(1 - \frac{1}{p})$. You may find it useful to interpret the random process in step 1 via the coins defined in part (a).

[1] Agrawal, Manindra, Neeraj Kayal, and Nitin Saxena. "PRIMES is in P." Annals of mathematics (2004): 781-793.

(c) Show that if SampleAndFactor(n) returns a number r, then the number is uniformly distributed in $[n]$. What is the probability that we return a number in one fixed iteration through the steps 1-3 in SampleAndFactor(n) as a function of M_n?

We now turn our attention to showing that the runtime of the algorithm is polynomial in expectation. For doing that we will show that the expected number of primality tests over one run of SampleAndFactor(n) is polynomial in the input size.

(d) For $i \in [n], \ell \in \mathbf{N}$ let $X_{i,\ell}$ be an indicator variable for the event "number i is tested for primality in the ℓ'th iteration through the steps 1-3 of SampleAndFactor(n)". Express the expected number of primality tests during the course of the algorithm in terms of the indicator variables $X_{i,\ell}$.

(e) What is the expected number of primality tests over one run of SampleAndFactor(n)? You can use without a proof that $M_n = \Theta(\frac{1}{\log n})$ which is known as Merten's theorem.

Exercise 3 — 20 points

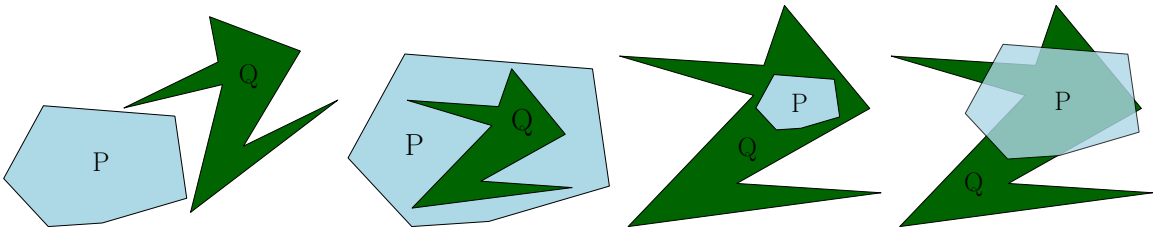
Intersection patterns of polygons

We have seen convex polygons in the lecture. More generally, a *polygon* Q is a bounded set in the plane whose boundary, denoted by ∂Q, is a finite union of straight line segments that together form a closed, non-selfintersecting loop. These line segments are the *edges* of the polygon and the endpoints of the line segments are the *vertices* of the polygon.

Let P be a convex polygon and let Q be a polygon. The polygons P and Q intersect in one of the four ways listed in Figure 1. That is, either (a) $P \cap Q = \emptyset$, (b) $Q \subseteq P$, (c) $P \subseteq Q$, or (d) $P \cap Q \neq \emptyset$ and there are points $p \in P, q \in Q$ so that $p \notin Q$ and $q \notin P$.

Design an algorithm that takes as input a convex polygon P and a polygon Q and distinguishes between the cases (a)-(d). More specifically, the input to the algorithm consists of two lists $(p_1, \ldots, p_n)$ and $(q_1, \ldots, q_m)$ of points in the plane that describe the counterclockwise order of vertices of the two polygons P and Q respectively. The output is one of the four cases (a)-(d) that describes how the input polygons intersect. You can assume that the vertices of the polygons are in general position so that no three vertices are on a common line and no two vertices share a common x-coordinate or y-coordinate. Your algorithm should run in expected time $O(n + m \log m)$.

HINT: If you don't find a way to start, you can try to distinguish case (b) from the other cases.



(a) $P \cap Q = \emptyset$. (b) $Q \subseteq P$. (c) $P \subseteq Q$. (d) Proper intersection.

Figure 1: The four different ways P and Q can intersect.