ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Institute of Theoretical Computer Science
Bernd Gärtner, Mohsen Ghaffari, Rasmus Kyng, Angelika Steger, David Steurer

Algorithms, Probability, and Computing **Special Assignment 1** **HS21**

- Write an exposition of your solution using a computer, where we strongly recommend to use LaTeX. **We do not grade hand-written solutions**.
- You need to submit your solution via Moodle until **Tuesday, November 2, 2021** by **10 pm**. Late solutions will not be graded.
- For geometric drawings that can easily be integrated into LaTeX documents, we recommend the drawing editor IPE, retrievable at `http://ipe.otfried.org` in source code and as an executable for Windows.
- Write short, simple, and precise sentences.
- 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. However, you need to write down the names of all your collaborators at the beginning of the writeup. 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 1A

10 **points**

(*A slightly different MSF algorithm*)

Consider the following variation of the Minimum Spanning Forest Algorithm in the script.

> RANDOMIZED MINIMUM SPANNING FOREST ALGORITHM(G):
> Perform two iterations of Borůvka's algorithm
> In the new graph:
> Select edges with probability p and compute recursively
> an MSF for the graph consisting of the selected edges.
> Call this forest T.
> Use FINDHEAVY to find all unselected edges that are not T-heavy.
> Add all edges that are *not* T-heavy to T and delete all other edges.
> recurse (until the graph contains only one vertex)

The two differences compared to the algorithm in the script are highlighted in red. First, the algorithm performs only two instead of three Borůvka iterations in the beginning. Second, each edge is sampled with probability p instead of probability $1/2$. Find a value for p such that the expected running time of the algorithm is $O(n + m)$. You can refer to the analysis of the original algorithm in the script. You don't need to analyze the correctness of the algorithm.

Exercise 1B

20 **points**

(*Black-Box Minimum Spanning Tree*)

In this exercise we ask you to design a randomized algorithm that computes a Minimum Spanning Tree of some weighted input graph G. We assume that G is connected and that the edge weights are pairwise distinct. Note that you have seen a randomized algorithm that solves this problem in expected $O(m)$ time during the first lecture of the course. This time, your algorithm cannot directly access the input graph. Instead, besides having access to the number of vertices and number of edges of a given graph it is only allowed to invoke each of the following black-box routines a *constant* number of times.

1. Algorithm $\mathcal{A}_{\text{MSF}}$ takes as input some weighted graph H with n nodes and at most $100n^{4/3}$ edges. It returns a Minimum Spanning Forest of the graph H.

2. Algorithm $\mathcal{A}_{\text{sample}}$ takes as input some weighted graph H and some $p \in [0, 1]$. It returns a weighted graph obtained from H by including each edge of H independently with probability p.

3. Algorithm $\mathcal{A}_{\text{T-heavy}}$ takes as input some weighted graph H and some weighted Forest T, with both H and T having the same set of vertices. It returns a weighted graph obtained from H by removing all T-heavy edges.

Your algorithm needs to output a Minimum Spanning Tree with probability at least $1-O(1/n)$. You only need to analyze the success probability, but not the runtime of the algorithm. You can obtain partial points by proving that your algorithm succeeds with a positive constant probability.
Hint: The exercise is very similar to the first exercise of last year's special assignment. Feel free to take a look at the past exercise `https://www.ti.inf.ethz.ch/ew/courses/APC20/spa/spa1_v2.pdf` and the master solution `https://www.ti.inf.ethz.ch/ew/courses/APC20/spa/spa1-solution.pdf`.

Exercise 2

25 **points**

(*Approximate Minimum Cuts*)

Let G denote a connected, undirected and simple n-node graph with $n \geq 2$. In contrast to the lecture notes, we define a cut as a partition (S, T) of the vertices of G with $S, T \neq \emptyset$. The size of a cut is defined as the total number of edges with one endpoint in S and one endpoint in T. Let $k = \mu(G)$ denote the minimum cut size of G. An α-approximate minimum cut of G is a cut of size at most αk.

(a) Consider an arbitrary $\alpha \geq 1$. Devise a (randomized) algorithm that outputs a single cut such that each α-approximate minimum cut in G is outputted with probability at least $\frac{1}{(2n)^{\lceil 2\alpha \rceil}}$.

(b) Show that G has at most $O((2n)^{\lceil 2\alpha \rceil})$ different α-approximate minimum cuts.

(c) Consider an arbitrary integer $2\alpha \geq 2$. For every n and even k (k can potentially be larger than n), find a *multigraph* H with n vertices and $\mu(H) = k$ that has $\Omega(n^{\lceil 2\alpha \rceil})$ α-approximate minimum cuts . The constant term hidden in the Ω-notation is allowed to depend on α.

Recall that for any natural numbers a and b with $a \geq b$ we have $\binom{a}{b} \geq \left(\frac{a}{b}\right)^b$.

Exercise 3

25 **points**

(*Random Binary Search Tree*)

Consider a random binary search tree with the n distinct keys $1, 2, \ldots, n$ and let $r \in \{2, \ldots, n\}$ be arbitrary. We are interested in the distance between the smallest key and the r-th smallest key, i.e., the number of edges along the path between the two corresponding nodes. We denote that distance by the random variable $X_{n,r}$. The goal of this exercise is to show that $\mathbb{E}[X_{n,r}] = O(\log r)$.

In one of the exercises, we have seen that one can obtain a random binary search tree by first choosing a permutation $\pi\colon \{1, 2, \ldots, n\} \mapsto \{1, 2, \ldots, n\}$ uniformly at random and then inserting the keys in the order given by the permutation, i.e., first inserting $\pi(1)$, then $\pi(2)$ and so on. For such a permutation π as above, let j denote the smallest index with $\pi(j) \in \{1, 2, \ldots, r\}$. For $i \in \{r+1, r+2, \ldots, n\}$, we denote with A_i the event that i is the smallest value in the

set $\{\pi(1), \pi(2), \ldots, \pi(j-1)\}$. To illustrate the definition, consider the permutation 54123. For $r = 2$, we have $j = 3$ and therefore $\{\pi(1), \pi(2), \ldots, \pi(j-1)\} = \{\pi(1), \pi(2)\} = \{5, 4\}$. Hence, the event A_4 occurred unlike the events A_3 and A_5.

(a) Argue that $\Pr[A_i] = \frac{r}{i(i-1)}$.

(b) Argue that $\mathbb{E}[X_{n,r}|A_i] = O(\log i)$.

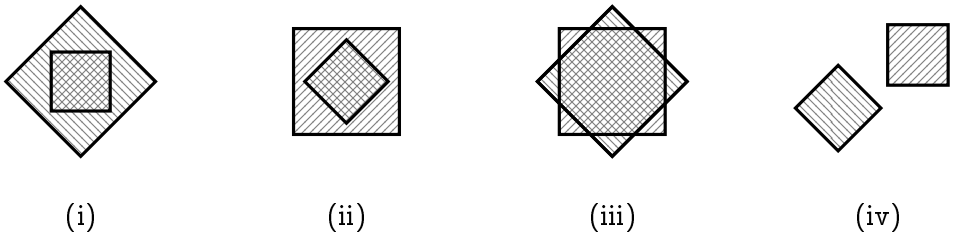
(c) Show that $\mathbb{E}[X_{n,r}] = O(\log r)$ by using the results from the previous two subtasks.

Exercise 4

20 **points**

(*Intersecting Convex Polygons*)

Let P and Q be two distinct convex polygons in the plane (for the purpose of this exercise, by *polygon* we mean the set of all boundary and interior points). They are both given as sorted arrays that contain the respective vertices in clockwise order. Design a deterministic algorithm that runs in time $O(n)$ (where n is the overall number of vertices) and which decides which one of the following four cases holds: (i) P contains Q, (ii) Q contains P, (iii) P and Q intersect but neither polygon contains the other, (iv) P and Q are disjoint (see also Figure 1).
REMARK: You can make any general position assumptions in order to avoid tedious special cases, but you should always make such assumptions explicit.



(i) (ii) (iii) (iv)

Figure 1: The four cases to be distinguished with P =  and Q =  .