ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Institute of Theoretical Computer Science
Bernd Gärtner, Rasmus Kyng, Angelika Steger, David Steurer, Emo Welzl

Algorithms, Probability, and Computing **Special Assignment 1** **HS23**

- 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 **November 1st** by **2 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 1

40 **points**

(*Weighted Minimum Cut*)
The aim of this exercise is to prove that the MinCut algorithm that we have seen in the lecture can find the minimum cut in a weighted graph.

Given a (multi)-graph $G = (V, E)$ with n vertices and m (multi-)edges and with weights $w : E \mapsto \mathbb{R}$ on the multi-edges, define the weight of a cut $C \subseteq E$ as $w(C) = \sum_{e \in C} w(e)$. A *minimum weighted* cut is a cut with minimum weight. For simplicity we denote the weight of the minimum weighted cut in G as $\mu(G)$.

Consider the algorithm WeightedMinCut(G) reported below. The line "collapse parallel edges of G", substitutes all the set of parallel edges $e_1, e_2, \ldots, e_i$ between two vertices u and v with only one edge between (u, v) with weight $w(e_1) + w(e_2) + \cdots + w(e_i)$. Note that this operation turns the weighted (multi)graph into a weighted graph.

(a) Consider the contraction of a random multi-edge $e \in E(G)$ with probability $\frac{w(e)}{\sum_{i \in E(G)} w(i)}$. Prove that probability of $\mu(G) = \mu(G/e)$ for a randomly chosen edge e $E(G)$ is at least $1 - 2/n$. Deduce that the algorithm WeightedMinCut(G) is correct, i.e. with high probability the cut returned is the minimum cut. Note that $\mu(G/e)$ may contain multi-edges even if G does not.

(b) Suppose to have access to a black-box method SampleEdge(G) that returns a random multi-edge $e \in E(G)$ with probability $\frac{w(e)}{\sum_{i \in E(G)} w(i)}$. Argue that WeightedRandomContract(G, t) can be implemented using $O(n)$ calls to SampleEdge(G).

(c) Suppose that $W = \text{poly}(n)$ and that you can sample a number from $\{1, 2, \ldots, \text{poly}(n)\}$ in $O(1)$ time. Assume initially that $m = O(n^2)$ (there is at most a quadratic number of multi-edges). We want to design the method SampleEdge(G) that given the weight $w(e) \in \{1, 2, \ldots, W\}$ for each edge e allow you to sample from the weighted distribution.

- Design a data structure that can be built in $O(m)$ time and $O(m)$ space and allows you to sample from the set of all the edges $E = \{1, 2, \ldots, m\}$ with probability $\frac{w_e}{\sum_{i \in E} w(i)}$ where $w_i \in \{1, 2, \ldots, W\}$ in $O(\log m)$ time.
- Now suppose that some of the edges have been removed and others have been collapsed. Let $S \subseteq E$, such that $\sum_{i \in S} w(i) \geq \frac{\sum_{i \in E} w(i)}{2}$, be the set of edges that have not been remove and $T \subset 2^S$ be a partition of S. Devise a way to use sample a set $t \in T$ with probability $\frac{\sum_{j \in t} w(j)}{\sum_{i \in S} w(i)}$ using a constant number of calls to the data structure from the previous point in expectation. (You can assume given a $i \in S$ you can get the set $t \in T$ that contains i in constant time.)
- Explain why, building $O(\log n)$ instances of the data structure you developed in expectation, you can get an implementation SampleEdge(G) needed in point (b) with an *amortized* query time of $O(\log n)$. (The amortized query time is the sum of the time needed by the queries made during the execution of WeightedMinCut divided by the number of queries.)

(d) Let $\epsilon > 0$. Now suppose that you have access WeightedMinCut but you can only use it for graphs with weights in the range $\{1, 2, \ldots, \lceil 10n/\epsilon \rceil\}$.
Let G be a graph with weights in the range $\{1, 2, \ldots, W\}$. Design an algorithm that uses at most $\log(nW)$ call to WeightedMinCut and finds a cut C of graph G such that $w(C) \leq (1+\epsilon)\mu(G)$.

WeightedRandomContract(G, t):
While $V(G) > t$ do
  Sample $e \in E(G)$ with probability $\frac{w(e)}{\sum_{i \in E(G)} w(i)}$
  $G \leftarrow G/e$
End while
collapse parallel edges of G
Return G

WeightedMinCut(G, k):
If $n \leq 16$ then
  Compute $\mu(G)$ deterministically
  Return $\mu(G)$
else
  $t \leftarrow \lceil n/\sqrt{2} \rceil + 1$
  $H_1 \leftarrow$ WeightedRandomContract(G, t)
  $H_2 \leftarrow$ WeightedRandomContract(G, t)
  Return
min(WeightedMinCut(H_1, k), WeightedMinCut(H_2, k))

Answer to Exercise 1

(a) Let $w(E(G)) = \sum_{e \in E(G)} w(e)$. The degree of each vertex is at least $\mu(G)$ so $w(E(G)) \geq \frac{n\mu(G)}{2}$. So, the probability that we contract an edge not in the min-cut is at least

$$1 - \frac{\mu(G)}{w(E(G))} \geq 1 - \frac{2}{n}.$$

Once we know this, all the remaining part of the analysis is identical.

(b) At each iteration of the "while" loop in WheightedRandomContract we call SampleEdge in order to sample a random edge that we then contract. After the contraction of one edge, the number of vertices in the graph decreases by 1. As a consequence, we have to sample at most n edges and therefore, we only need a linear number of calls to SampleEdge.

(c)
- We create an array V where each entry $V[i]$ contains the quantity $\sum_{j \leq i} w(j)$. In order to sample a random edge, we generate a random number x in $\{1, 2, \ldots, \sum_{i=1}^{n} w(i)\}$, we perform a binary search to locate the index ℓ of the array such that $V[\ell - 1] \leq x \leq V[\ell]$ (Suppose that $V[-1] = 0$), and finally, we return ℓ.
- We keep sampling edges from the distribution with all the edges using the data structure just described. When we sample an edge $e \in S$, we return the set $t \in T$ that contains e. Since $\sum_{i \in S} w(i) \geq \frac{\sum_{i \in E} w(i)}{2}$, in expectation, we have to sample 2 edges from the distribution with all the edges to get a sample from the distribution with the removed/collapsed edges.
- We create the the data structure we developed in the first point and we use the technique we developed in the second point to sample the edges from the distribution we need after we removed or collapsed some of the edges. When the condition $\sum_{i \in S} w(i) \geq \frac{\sum_{i \in E} w(i)}{2}$ fails, we start from scratch and build a new array that contains only the edges that have not been removed. Since the we do this operation only when the total weight of the edges has been halved, we do this operation at

most $\log_2(n)$ times.

Now we need to compute the total cost for all the calls to SAMPLEEDGE made by WEIGHTEDMINCUT divided by the number of time SAMPLEEDGE get called. We distinguish two types of costs, there is the cost due to the binary search that is performed at each call and then the cost needed to build (and rebuild) the array. The cost due to the binary search never more than $\log(n)$ for each call. For the other term, in expectation we are more likely to sample high weight edges earlier so the weight is more likely to be halved before the number of edges is. Thus we need to build at most one array (in expectation) of size m, at most 2 arrays of size $m/2$, 4 arrays of size $m/4$ and so on until we have $2^{O(\log m)}$ arrays of constant size. The total cost is then $O(m\log(m)) = O(m\log(n))$ that has to be divided by $O(m)$ number of calls.

(d) Note that we will need weights in the range $\{1, 2, \ldots, \lceil 10m/\epsilon \rceil\}$ for the following solution.

Suppose that we know the value of $\mu(G)$ up to a constant factor. For example suppose that we know that $Q/10 \leq \mu(G) \leq Q$. At this point, we contract all the edges e in G with $w(e) > Q$ without changing the min-cut. Then, we approximate the weights in G with the weight function

$$w'(e) = \lceil \frac{w(e)}{Q} \frac{10m}{\epsilon} \rceil,$$

(Note that w' takes values in $\{1, \ldots, 10m/\epsilon\}$) and compute the min-cut of G with the weights w' and obtain the value μ'. Now we have that

$$\mu(G) \leq \frac{Q\epsilon}{10m}\mu' \leq \mu(G) + \frac{Q\epsilon}{10} \leq (1+\epsilon)\mu(G).$$

Finally, we know that $1 \leq \mu(G) \leq nW$ so we can try the values $U = 1, 10, 100, \ldots, 10^{\lceil \log_{10}(nW) \rceil}$.

.

Exercise 2

20 **points**

(*Nodes at distance 3 in Random search trees*)

Let $n \in \mathbb{N}$. Denote with $W(n, d)$ the expected number of nodes of depth d in a random search tree for n keys.

(a) Compute $W(n, 0)$ and $W(n, 1)$ for $n \geq 1$.

(b) Let $d \geq 2$. Write $W(n, d)$ as a function of $W(n-1, d)$ and $W(n-1, d-1)$.

(c) Prove that

$$W(n, 2) = \begin{cases} 0 & n \in \{1, 2\}, \\ 4 - 4\frac{H_{n-1}}{n} - \frac{4}{n} & n > 2. \end{cases}$$

Answer to Exercise 2

(a) The root is the only node of depth 0, so $W(n, 0) = 1$.
When $n = 1$, there are no nodes of depth 1. When $n > 1$, we have one node of depth 1, if the root has rank 1 or n and two nodes in all the other cases. Since the root is uniformly distributed, it has rank 1 or n with probability $2/n$. In conclusion,

$$W(n, 1) = \begin{cases} 0 & n = 1, \\ \frac{2n-2}{n} & n > 1. \end{cases}$$

(b) Let with $w(n, d)$ be the number of nodes of depth d in a tree. First, we condition on the rank of the root of the tree,
$W(n, d) = \mathbf{E}[w(n, d)] = \sum_{i=0}^{n-1} \mathbf{E}[w(n, d) | \text{the root has rank } i+1]\, \mathbf{Pr}[\text{the root has rank } i+1] = \frac{1}{n} \sum_{i=0}^{n-1} \mathbf{E}[w(n, d) | \text{the root has rank } i+1]$.
Once we know the rank of the root, the number of nodes of depth d is equal to the sum of the nodes of depth $d-1$ in the left and right sub-trees. Hence,

$$W(n, d) = \frac{1}{n} \sum_{i=0}^{n-1} \left(W(i, d-1) + W(n-i-1, d-1) \right).$$

Then for $n > 1$, we consider the quantity $nW(n, d) - (n-1)W(n-1, d)$ and after some algebraic manipulations we obtain

$$W(n, d) = \frac{n-1}{n} W(n-1, d) + \frac{2}{n} W(n-1, d-1).$$

(c) Firstly, for $n \leq 2$, we have $W(n, 2) = 0$, since we can't have a node of depth 2 with less than 3 nodes. For $n \geq 3$, we compute the base case $W(3, 2) = \frac{2}{3}$ and we note that it satisfies the expression given. Finally, for $n > 3$ we substitute the expression into the recurrence relation of point (b),

$$W(n, 2) = \frac{n-1}{n} W(n-1, 2) + \frac{2}{n} W(n-1, 1).$$

$$= \frac{n-1}{n} 4 \frac{(n-1) - H_{n-2} - 1}{n-1} + \frac{2}{n} 2 \frac{(n-1)-1}{n-1}$$
$$= 4 \frac{n - H_{n-1} - 1}{n}.$$

Since the recurrence relation is satisfied, the expression must be correct.

Exercise 3

20 **points**

(*Ancestor with biggest rank*)

Recall that a node u is an ancestor of node v in a rooted tree if u lies on the unique path from v to the root on the tree. Note that v is an ancestor of itself. Furthermore, given two nodes u, v in a tree there is a unique path that connects these two nodes, we call the number of edges in this path the distance between u and v. Given a random search tree T, compute the expectation of the sum of the distances from each point to its ancestor with the biggest rank.

Answer to Exercise 3

The root node is the ancestor with the biggest rank for all nodes in the subtree rooted in its left child and none of the nodes in the subtree rooted in its right child. This means, for the left subtree the sum of the distances correspond to the sum of the depths of each node and for the right subtree we can compute it recursively. The root node is also its own ancestor with the biggest rank, which corresponds to a distance of 0. Denote with f_n the quantity we want to compute. $f_1 = 0$, $f_2 = 1/2$. For $n \geq 2$,

$$f_n = \frac{1}{n} \sum_{i=0}^{n-1} (h_i + i + f_{n-i-1})$$

Where h_n is the overall depth of a node in a random search tree with n entries. From the script we know that $h_n = \sum_{i=1} n D_n^{(i)} = 2(n+1)H_n - 4n$. Compute $nf_n - (n-1)f_{n-1}$ and get

$$f_n - f_{n-1} = 2H_{n-1} - 3\frac{n-1}{n}.$$

Finally, sum up all the terms to get the final result,

$$f_n = f_1 + \sum_{i=2}^{n} (f_i - f_{i-1}) = 3H_n + 2\sum_{i=1}^{n-1} H_i - 3n = (2n+3)H_n - 5n.$$

Exercise 4

20 **points**

(*Nearest point or segment in the square*)
We call a *tiling* of the unit square $[0,1] \times [0,1]$ a partition of the square into convex polygons such that the interiors of no two polygons intersect and the union of all the polygons covers the entire surface of the square.
You are given a tiling of the unit square together with a collection of points inside the square. Suppose that the tiling is given by collection of all the edges of the polygons and let n be the sum of the number of edges in the tiling and the number of points given. Design a data structure that given a query point $p \in [0,1] \times [0,1]$, returns the closest point or segment.
In order to get full score, your data structure should take $O(n)$ space and expected $O(\log n)$ query time, while the preprocessing time has to be polynomial in n. You can assume that the points and the lines supporting all edges are in general position.

Answer to Exercise 4

Since n is the number of points and segments, there will be at most n points and n segments. We build two independent structures for finding the closest point and the closest line and then we return the closest one.
For the points, we can simply use the trapezoidal decomposition to store the Voronoi diagram; with every segment of the Voronoi diagram we associate the point of the Voronoi cell below that segment. Since the Voronoi diagram consists of $O(n)$ segments, the data structure takes $O(n)$ space, the query time to find the closest point is $O(\log n)$ in expectation, and can be built in polynomial time.
For the segments, we build a trapezoidal decomposition to store the Voronoi diagram of the edges. Fix a segment e and look at one polygon P of the two polygons of the tiling that contain e. The part of the Voronoi cell of e contained in P is delimited by parts of the bisectors of the angles defined by the line that contains e and a line that contains another edge of P. This implies that the cells of the diagram are polygons and that each cell has at most n edges. In order to get the $O(n)$ bound on the size, this bound is not sufficient and we have to prove that the Voronoi diagram contains at most $O(n)$ edges. We do this following the same reasoning we used in the lectures for the Voronoi diagram of a set of points: Consider the dual of the graph given by the Voronoi Diagram, this has $O(n)$ vertices (one for each segment) and, since it is a planar graph, at most $O(n)$ edges. The number of edges in the Voronoi diagram is exactly the same as the number of edges in the dual graph so the number of vertices, edges, and cells in the diagram is $O(n)$ and we can store it using the trapezoidal decomposition. This data structure also takes $O(n)$ space, the query time to find the closest point is $O(\log n)$ in expectation. It is clear that the Voronoi diagram and the trapezoidal decomposition can be built in polynomial time.