CS 466/666: Algorithm Design and Analysis **University of Waterloo: Fall 2023**

Homework 1

Due: Monday, September 25, 2023

Problem 1. In this question, we examine a way of speeding up Karger's minimum cut algorithm, due to Karger and Stein (JACM 1996). Basically, the issue with Karger's algorithm is that it "waits too long" before repeating the algorithm. In other words, we had the basic contraction algorithm that succeeds with probability roughly $1/n^2$, and then we repeat it $O(n^2)$ to boost the probability to a constant.

But, the probability that the basic contraction algorithm fails in its first step is very low—only $2/n$—, which is good for us, while the probability that it fails in its last step is very high—already $1/3$—which is undesirable. Karger-Stein algorithm is exploiting this phenomenon cleverly by *interjecting* repetition steps in the middle of basic contraction algorithm. Basically, we are unlikely to destroy the min-cut in the early steps of the contraction algorithm, so why repeat all these steps altogether?

(a) Show that each contraction operation can be implemented in $O(n)$ time. Conclude that the original Karger's algorithm that succeeds with probability at least $2/3$ can be implemented in $O(n^4)$ time.

(5 points)

(b) Karger's basic contraction algorithm runs $n-2$ contractions consecutively, and ends with a graph on 2 vertices, which preserves a *fixed* minimum cut $(S, V \setminus S)$ with probability at least $2/n \cdot (n-1)$.

Instead, consider running only $n - n/\sqrt{2}$ random contraction steps. Prove that the probability that a fixed minimum cut survives this contraction process is at least $1/2$. **(5 points)**

(c) Consider the following Karger-Stein algorithm: starting from a multi-graph G on n vertices, randomly contract edges until $n/\sqrt{2}$ vertices remain; call the new graph G'. Recursively, run *two* copies of the algorithm *independently* on G' and return the smallest of the two cuts found as the answer.

Define the recurrence $T(n)$ as the worst-case runtime of the above algorithm on n-vertex graphs. Prove:

$$T(n) \leqslant O(n^2) + 2 \cdot T(n/\sqrt{2}).$$

Use this recurrence to bound the runtime of the algorithm with $O(n^2 \cdot \log n)$ time. **(7.5 points)**

(d) Let $P(n)$ denote the worst-case probability that the above algorithm returns a minimum cut of a given n-vertex graph. Prove:

$$P(n) \geqslant \frac{1 - (1 - P(n/\sqrt{2}))^2}{2}.$$

Use this recurrence to bound the probability of success of the algorithm with $\Omega(1/\log n)$.

(7.5 points)

Hint: The easiest way (that I know of!) to solve such a recurrence is by induction: take your induction hypothesis to be that $P(n) \geqslant \frac{c}{\log n}$ for a sufficiently small constant c (and sufficiently large n).

The conclusion is that we found an algorithm with runtime $O(n^2 \cdot \log n)$ that solves the minimum cut problem with probability $\Omega(1/\log n)$. As in the class, we can repeat this algorithm $O(\log n)$ time and return the best answer to succeed with probability at least $2/3$ in $O(n^2 \cdot \log^2 n)$ time. This is a pretty fast algorithm now! (on dense graphs, this is just tiny bit slower than reading the input graph itself).

Problem 2. A family of functions $\mathcal{H} = \{h \mid h : [n] \mapsto [m]\}$ is called a **pairwise independent hash family** iff for all $x \neq y \in [n]$ and $a, b \in [m]$, for h chosen uniformly at random from $\mathcal{H}$, we have,

$$\Pr\left(h(x) = a \wedge h(y) = b\right) = \frac{1}{m^2}.$$

In this problem, we investigate one method to obtain such a hash family (although this is *not* the most efficient method).

(*a*) Let $X_1, X_2, \ldots, X_k$ be independent and uniform binary random variables, i.e.,

$$\Pr(X_i = 0) = \Pr(X_i = 1) = 1/2.$$

Given $S \subseteq \{1, \ldots, k\}$, $S \neq \emptyset$, define a random variable $Y_S = \oplus_{i \in S} X_i$, i.e., the XOR of X_i's for $i \in S$. Prove that the set

$$\{Y_S \mid S \subseteq \{1, \ldots, k\}, S \neq \emptyset\}$$

is a collection of **pairwise independent random variables**, meaning that for all non empty sets $S \neq T \subseteq \{1, \ldots, k\}$ and pairs $a, b \in \{0, 1\}$:

$$\Pr[Y_S = a \wedge Y_T = b] = \frac{1}{4}.$$

(12.5 points)

(*b*) Use the construction in part (a) to design a family of pairwise independent hash functions

$$\mathcal{H} = \{h \mid h : [n] \mapsto [m]\}$$

constructed using $O(\log n \log m)$ random bits and $\text{poly}(\log n, \log m)$ time to compute. **(12.5 points)**

Problem 3. Consider a complete tree of height h, wherein the root, as well as any internal node has exactly 3 child-nodes; thus, the tree has $n = 3^h$ nodes. Suppose each leaf of the tree is assigned a Boolean value. We define the value of each internal node as the *majority* of the value of its child-nodes. The goal in this problem is to determine the value of the root.

An algorithm for this problem is provided with the structure of the tree (not the valuation of the leaves) and at each step it can *query* a leaf and read its value.

(*a*) Show that for any deterministic algorithm, there is an instance (a set of Boolean values for the leaves) that forces the algorithm to query all the $n = 3^h$ leaves.

(10 points)

(*b*) Consider the recursive randomized algorithm that evaluates two subtrees of the root chosen at random. If the values returned disagree, it proceeds to evaluate the third subtree. Show that the expected number of the leaves queried by the algorithm on any instance is at most $n^{0.9}$. **(15 points)**

Problem 4. Given a set of numbers S and a number $x \in S$, the **rank** of x is defined to be the number of elements in S that have value at most x:

$$rank(x, S) = |\{y \in S : y \leqslant x\}|$$

Given a parameter $\varepsilon \in (0, 1/2]$, we say that an element $x \in S$ is an **ε-approximate element of rank** r if

$$(1 - \varepsilon) \cdot r \leqslant rank(x, S) \leqslant (1 + \varepsilon) \cdot r$$

Recall the streaming model of computation discussed in the class. Suppose we are given a stream of numbers $S = (s_1, s_2, \ldots, s_n)$, where $s_i \in [m]$ for $i \in [n]$, and assume that all s_i's are distinct. Our goal is to design an $O(\varepsilon^{-2} \log m \log n)$ space streaming algorithm for retrieving an ε-approximate element for any given rank value.

(a) Recall that the median of a set S of n (distinct) elements is the element of rank $r = \lfloor n/2 \rfloor$ in S.

Consider this algorithm for computing an ε-approximate median: sample $O(\varepsilon^{-2} \log n)$ numbers from the stream uniformly at random (with repetition) and then return the median of the sampled numbers. Prove that this algorithm returns an ε-approximate median with probability at least $1 - 1/\text{poly}(n)$.

(12.5 points)

(b) We now extend the previous algorithm to compute an ε-approximate element of rank r for any $r \in [n]$.

Consider this algorithm: Let $t = \lceil 24\varepsilon^{-2} \log m \rceil$. If $r \leqslant t$, then simply maintain a list T of r smallest elements seen in the stream, and output the largest element in T at the end of the stream. Otherwise, choose each element in the stream with probability t/r, and maintain the t smallest sampled values in a list T. At the end of the stream, output the largest number in T. Prove that this algorithm outputs an ε-approximate element of rank r with probability at least $1 - 1/\text{poly}(n)$. **(12.5 points)**

Problem 5 (Extra credit). Let us examine a different contraction-based algorithm for minimum cuts. Consider an undirected graph $G = (V, E)$ and suppose for every vertex $v \in V$, we sample *two* edges uniformly at random, with repetition, from the edges incident on v to obtain a graph H. Let $C_1, \ldots, C_k$ be the *connected components* of H. Contract $C_1, \ldots, C_k$ in G to obtain a new graph $\tilde{G}$.

(a) Suppose $\lambda(G)$—the minimum cut size of G—is *strictly* smaller than the minimum degree of G. Prove that with some *constant* probability, $\lambda(\tilde{G}) = \lambda(G)$. [1]

(+10 points)

(b) Prove that $\tilde{G}$ has $O(n/\lambda(G))$ vertices with constant probability. [2] **(0 points)**

The nice thing about this approach is that, we can, in $O(m)$ time, reduce the number of vertices in the graph dramatically while preserving the minimum cut with some constant probability. Thus, we can instead focus on solving the minimum cut problem on this possibly much smaller graph.

For instance, suppose $\lambda(G) = \Omega(\sqrt{n})$ which means the graph has $\Omega(n^{1.5})$ edges. Running this algorithm gives us a graph on $O(\sqrt{n})$ vertices also and running Karger-Stein algorithm on that graph only requires $O((\sqrt{n})^2 \cdot \log^2(n)) = O(n \log^2 n)$ time. Thus, the overall runtime of the algorithm will only be $O(n^{1.5})$ (for the first step), asymptotically as fast as reading the input graph.

[1] A general comment: we are typically interested in minimum cut problem when its value is strictly smaller than the minimum degree; otherwise, the min-cut is a singleton vertex which can be found trivially by checking the degrees of all vertices.

[2] I am assigning a grade of zero to this part, to *discourage*(!) you from attempting this question *in hope of helping your grade.* While this is not an impossibly hard statement to prove, I do not know any simple enough proof for it either. So, only attempt this part *if* you are really interested in challenging yourself and after you finished the rest of your homework!