Exercise Set VIII, Algorithms II 2024

These exercises are for your own benefit. Feel free to collaborate and share your answers with other students. Solve as many problems as you can and ask for help if you get stuck for too long. Problems marked * are more difficult but also more fun :).

These problems are taken from various sources at EPFL and on the Internet, too numerous to cite individually.

1 In the previous exercise set you showed that if n balls are placed in n bins using a pairwise independent hash function $h : \{1, 2, \ldots, n\} \to \{1, 2, \ldots, n\}$, then the maximum bin load is $O(\sqrt{n})$ with high constant probability. Give an example of a pairwise independent hash family (i.e., a distribution over hash functions) such that the expected maximum bin load is $\Omega(\sqrt{n})$ with high probability.

Solution: First, choose a random k between 1 and n to be the 'crowded bin'. Next, choose a random permutation π of $\{1, 2, \ldots, n-1\}$. For $1 \leq i \leq n-1$ put ball i into bin

$$\begin{cases} k & \text{with probability } 1/\sqrt{n} \\ k + \pi(i) & \text{with probability } 1 - 1/\sqrt{n}, \end{cases}$$

where the sum is taken $\mod n$. Put ball n into a random bin.

We now prove that the allocation is pairwise independent. Consider balls i, j, where $1 \leq i < j \leq n-1$. The bins these two balls fall into are independent. They both go into the crowded bin with probability $1/n$, and the crowded bin is equally likely to be any of the n bins. And if they don't both go into the crowded bin, by symmetry they are equally likely to be in any two different bins. Thus, the bins these two balls are placed into are independent.

Since ball n is placed into a random bin, it is independent of the other balls.

The expected max load here is essentially the expected number of balls in the crowded bin, which is $\Omega(\sqrt{n})$.

2 In this problem we shall use the following properties of MinHashing (prove them if you feel good today): Let $X_1, \ldots, X_n$ be independent random variables uniformly distributed in $[0, 1]$ and let $Y = \min\{X_1, \ldots, X_n\}$. Then $\mathbb{E}[Y] = \frac{1}{n+1}$ and $\mathrm{Var}(Y) \leq \frac{1}{(n+1)^2}$.

We now use these properties to analyze a different algorithm than the one explained in class for estimating the number of distinct elements in a sequence. Indeed, consider the following algorithm for estimating F_0, the number of distinct elements in a sequence $x_1, \ldots, x_m \in \{0, 1, \ldots, n-1\}$. Let $h : \{0, 1, \ldots, n-1\} \to [0, 1]$ s.t. $h(i)$ is chosen uniformly and independently at random in $[0, 1]$ for each i. We start with $Y = 1$. After reading each element x_i in the sequence we let $Y = \min\{Y, h(x_i)\}$.

2a Show that by the end of the stream $\frac{1}{\mathbb{E}[Y]} - 1$ is equal to F_0.

2b *(*)* Use the above idea to design a streaming algorithm to estimate the number of distinct elements in the sequence with multiplicative error $1 \pm \epsilon$. For the analysis you can assume that you have access to k independent hash functions as described above. Show that $k \leq O(1/\epsilon^2)$ many such hash functions are enough to estimate the number of distinct elements within factor $1 + \epsilon$ with probability at least $9/10$.

Hint: run the k copies of the above algorithm in parallel. Let Y_i be the Y variable of the i:th copy. Then what is the expected value and variance of the random variable $(Y_1 + Y_2 + \ldots Y_k)/k$? Then apply Chebychev's Inequality.

Solution: As defined in the problem statement, let F_0 be the number of distinct elements in the given sequence. Then by the definition of h, our algorithm uses F_0 uniform and independent random variables in $[0, 1]$, let us call them $X_1, ..., X_F$. Therefore, by definition $Y = \min\{X_1, \ldots, X_{F_0}\}$. Using the property of hashing in the problem statement we get that $\mathbb{E}[Y] = \frac{1}{F_0+1}$. So, $\frac{1}{\mathbb{E}[Y]} - 1 = F_0$ and $\mathrm{Var}(Y) \leq \frac{1}{(n+1)^2}$.

For the second part of this problem, we need to use the Chebychev's Inequality. Let us first recall this inequality:

Let X be a random variable. Then for any real number $t > 0$, we have that $\Pr[|X - \mathbb{E}[X]| \geq t\sqrt{\mathrm{Var}[X]}] \leq 1/t^2$.

Now let $X = \frac{Y_1+\cdots+Y_k}{k}$. We know that $\mathbb{E}[X] = 1/(F_0 + 1)$ and $\sqrt{\mathrm{Var}[X]} = \sqrt{\frac{\sum \mathrm{Var}[Y_i]}{k}} \leq \frac{1}{\sqrt{k}(n+1)}$ since Y_i are independent.

Using Chebychev's inequality, we get that for any $\epsilon_1 > 0$:

$$\Pr[|X - \frac{1}{F_0 + 1}| \geq \epsilon_1(1/(F_0 + 1))] \leq \frac{1}{(\epsilon_1\sqrt{k})^2}$$

. Now by choosing $k = O((1/\epsilon_1)^2)$ we get that:

$$\Pr[|X - 1/(F_0 + 1)| \geq \epsilon_1(1/(F_0 + 1))] \leq 0.1,$$

and so

$$\Pr[|X - 1/(F_0 + 1)| \leq \epsilon_1(1/(F_0 + 1))] \geq 0.9.$$

The above inequality means that with a high probability we estimate $1/(F_0+1)$ within a $(1\pm\epsilon_1)$-factor.

3 Consider the problem where we wish to find a large cardinality matching in a graph in the semi-streaming model. That is, the edges are streamed one-by-one. The graph has n vertices and we assume that your algorithm has access to storage space $O(n \cdot \mathrm{poly}\log n)$ (that is why it is called semi-streaming and not streaming as we have quite a lot of memory compared to the logarithmic memory seen in the lecture). Notice that although you have quite a lot of storage space, you do not have enough memory to store all (potentially $\Omega(n^2)$ many) edges. Devise an algorithm in this setting that returns a matching that has cardinality at least $1/2$ that of a maximum cardinality matching. (We note that improving the factor $1/2$ is considered a major open problem.)

Hint: think greedy.

Solution: We use the most straightforward greedy algorithm. Namely, we maintain a matching M. Initially $M = \emptyset$. Whenever an edge e arrives, we test whether $M \cup \{e\}$ is a matching; if so, we add e to M.

Clearly, the algorithm stores only $O(n)$ edges and thus it fits in memory.

Now we analyze the approximation factor. Let us call a matching M *maxim**al*** (as opposed to maxim*um*) if it cannot be extended, i.e., there is no edge $e \in E \setminus M$ such that $M \cup \{e\}$ is a matching. Notice that our matching M is maximal: if it were possible to add an edge e, then it would have been possible to add it to the partial matching that the algorithm had had at the time e was being processed. We conclude with the following fact.

Lemma 1. *The size of a maximal matching M is at least half the size of a maximum matching N.*

Proof. Intuitively: how wrong can it be to pick a bad edge for M? The worst that can happen is that this blocks the choice of two good edges (from N), one per endpoint.

Formally: define a function $f : M \to 2^N$ by $f(e) := \{f \in N : e \text{ is adjacent to } f\}$. Then:

- $|f(e)| \leq 2$ for any $e \in M$,
- $\bigcup_{e \in M} f(e) = N$: for if there was an edge $f \in N$ not adjacent to any edge of M, then it could be added to M,
- therefore $|N| = \left|\bigcup_{e \in M} f(e)\right| \leq |M| \cdot 2$.

□

4 *(Final exam question from 2017)* **Set packing in the semi-streaming model.**

Consider the problem of finding a maximum cardinality set packing in the semi-streaming model. An instance of this problem consists of a known universe U of n elements and sets $S \subseteq U$ are streamed one-by-one. The goal is to select a family $\mathcal{T}$ of pairwise disjoint sets (i.e., $S \cap S' = \emptyset$ for any two distinct sets $S, S' \in \mathcal{T}$) of maximum cardinality while only using $O(n \cdot \operatorname{poly} \log n)$ storage space.

Devise an algorithm in this setting that returns a set packing of cardinality at least $1/k$ times that of a maximum cardinality set packing, assuming that each streamed set S has cardinality at most k, i.e., $|S| \leq k$.

(In this problem you are asked to (i) design the algorithm, (ii) show that it uses $O(n \cdot \operatorname{polylog} n)$ space, and (iii) prove that it returns a solution of cardinality at least $1/k$ times the cardinality of a maximum cardinality set packing. Recall that you are allowed to refer to material covered in the course.)

Solution:

We run the simple greedy algorithm:

1. Initially, let $\mathcal{T} = \emptyset$.
2. For each streamed S: if S is disjoint from all sets in $\mathcal{T}$, add S to $\mathcal{T}$.
3. At the end, we simply return $\mathcal{T}$ as our solution.

We now analyze the greedy algorithm in terms of space and approximation guarantee.

Space: Since at all times $\mathcal{T}$ is a family of disjoint sets, we have $\sum_{S\in\mathcal{T}} |S| \leq n$. If we store each selected set S as a list of its elements this will take space $|S| \log n$ for each set $S \in \mathcal{T}$ (the $\log n$ is the space required to save the identifier of each element). Thus, as $\sum_{S\in\mathcal{T}} |S| \leq n$, we require $O(n \log n)$ space in total.

Approximation ratio: Let $\mathcal{O}$ be an optimal set packing. The greedy algorithm returns a maximal set packing so any $O \in \mathcal{O}$ intersects at least one set in $\mathcal{T}$ (maybe itself if it was selected by greedy). Moreover, any set S can intersect at most k sets in $\mathcal{O}$ since $|S| \leq k$. Therefore,

$$|\mathcal{O}| = \sum_{O\in\mathcal{O}} 1 \leq \sum_{O\in\mathcal{O}} \sum_{S\in\mathcal{T}:S\cap O\neq\emptyset} 1 = \sum_{S\in\mathcal{T}} \sum_{O\in\mathcal{O}:S\cap O\neq\emptyset} 1 \leq \sum_{S\in\mathcal{T}} k = k|\mathcal{T}|\,.$$