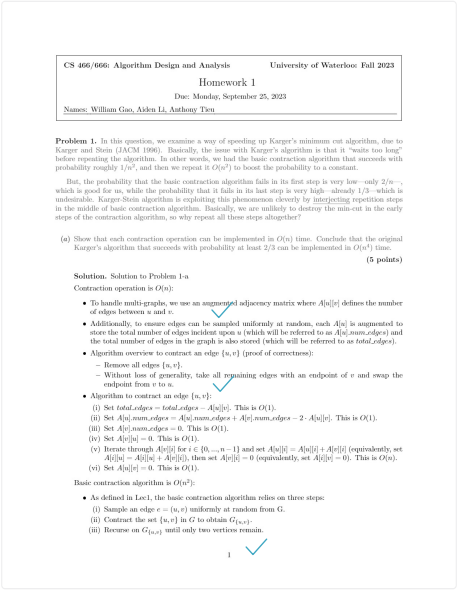
My grades for Homework 1

Q1 24 / 25

Refer to the assignment PDF.



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

Homework 1

Due: Monday, September 25, 2023

Names: William Gao, Aiden Li, Anthony Tieu

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)

Solution. Solution to Problem 1-a

Contraction operation is $O(n)$:

- To handle multi-graphs, we use an augmented adjacency matrix where $A[u][v]$ defines the number of edges between u and v.
- Additionally, to ensure edges can be sampled uniformly at random, each $A[u]$ is augmented to store the total number of edges incident upon u (which will be referred to as $A[u].num\_edges$) and the total number of edges in the graph is also stored (which will be referred to as $total\_edges$).
- Algorithm overview to contract an edge $\{u, v\}$ (proof of correctness):
  - Remove all edges $\{u, v\}$.
  - Without loss of generality, take all remaining edges with an endpoint of v and swap the endpoint from v to u.
- Algorithm to contract an edge $\{u, v\}$:
  (i) Set $total\_edges = total\_edges - A[u][v]$. This is $O(1)$.
  (ii) Set $A[u].num\_edges = A[u].num\_edges + A[v].num\_edges - 2 \cdot A[u][v]$. This is $O(1)$.
  (iii) Set $A[v].num\_edges = 0$. This is $O(1)$.
  (iv) Set $A[v][u] = 0$. This is $O(1)$.
  (v) Iterate through $A[v][i]$ for $i \in \{0, ..., n-1\}$ and set $A[u][i] = A[u][i] + A[v][i]$ (equivalently, set $A[i][u] = A[i][u] + A[v][i]$), then set $A[v][i] = 0$ (equivalently, set $A[i][v] = 0$). This is $O(n)$.
  (vi) Set $A[u][v] = 0$. This is $O(1)$.

Basic contraction algorithm is $O(n^2)$:

- As defined in Lec1, the basic contraction algorithm relies on three steps:
  (i) Sample an edge $e = (u, v)$ uniformly at random from G.
  (ii) Contract the set $\{u, v\}$ in G to obtain $G_{\{u,v\}}$.
  (iii) Recurse on $G_{\{u,v\}}$ until only two vertices remain.

1

- Algorithm overview for Step (i) (proof of correctness):
  - In our augmented adjacency matrix, each entry $A[u][v]$ keeps track of how many edges exist between vertices u and v. By the Handshaking lemma, each edge is then counted twice (once per endpoint).
  - As such, we can assign two unique indices to each edge (one per endpoint) from $[0, 2 \cdot total\_edges)$.
  - Randomly sampling a number from $[0, 2 \cdot total\_edges)$ using a uniform distribution will allow us to select an edge uniformly at random.
- Algorithm for Step (i). This is $O(n)$ time.
  (a) Randomly sample a number, x, from $[0, 2 \cdot total\_edges)$ using a uniform distribution. This is $O(1)$.
  (b) Let $u \in \{0, ..., n-1\}$. Determine the smallest value of u such that $\sum_{i=0}^{u} A[i].num\_edges > x$. This can be done in $O(n)$ by iteration.
  (c) Let $v \in \{0, ..., n-1\}$. Determine the smallest value of v such that $\sum_{i=0}^{u-1} A[i].num\_edges + \sum_{j=0}^{v} A[u][j] > x$. This can be done in $O(n)$ by iteration.
  (d) Use the computed values of u, v as the edge selection (u, v).
- Step (ii) As specified above, the contraction step is $O(n)$.
- Combined, Step (i) and (ii) take $O(n)$ time.
- Step (iii) Recurses on (i) and (ii) until only two vertices remain, that is, we recurse $n-2$ times on an $O(n)$ operation. Thus, the basic contraction algorithm is $O(n^2)$.

Original Karger's algorithm is $O(n^4)$:

- As defined in Lec1, Karger's algorithm relies on two steps:
  (i) Independently run basic contraction for n^2 times to obtain cuts.
  (ii) Return the smallest of these independent cuts.
- Step (i) is $O(n^4)$ since it repeats an $O(n^2)$ operation n^2 times.
- Step (ii) is $O(1)$.

We conclude that the original Karger's algorithm can be implemented in $O(n^4)$ time.

correct, and in the future there is no need to provide almost code-level detail

(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)**

Solution. Solution to Problem 1-b

Proof. Let G_i represent the resulting graph after the i-th contraction, for $i \in \{1...n - \frac{n}{\sqrt{2}} - 1\}$:

$$\Pr(\text{new-contraction}(G) \text{ outputs a minimum cut of } G) = \Pr(\lambda(G_{n-\frac{n}{\sqrt{2}}-1}) = \lambda(G))$$

By Lec1 Claim 2:

$$\lambda(G_{n-\frac{n}{\sqrt{2}}-1}) \geqslant \lambda(G_{n-\frac{n}{\sqrt{2}}-2}) \geqslant ... \geqslant \lambda(G_1) \geqslant \lambda(G)$$

Hence:

$$\begin{aligned}
\Pr(\lambda(G_{n-\frac{n}{\sqrt{2}}-1}) = \lambda(G)) &= \Pr(\lambda(G_{n-\frac{n}{\sqrt{2}}-1}) = \lambda(G_{n-\frac{n}{\sqrt{2}}-2}) \wedge \lambda(G_{n-\frac{n}{\sqrt{2}}-2}) = \lambda(G_{n-\frac{n}{\sqrt{2}}-3}) \wedge ... \wedge \lambda(G_1) = \lambda(G)) \\
&= \prod_{i=1}^{n-\frac{n}{\sqrt{2}}-1} \Pr(\lambda(G_i) = \lambda(G_{i-1}) | \lambda(G_{i-2}) = ... = \lambda(G_1) = \lambda(G)) \\
&\geqslant \prod_{i=1}^{n-\frac{n}{\sqrt{2}}-1} \left(1 - \frac{2}{n-i+1}\right) \quad \text{By Lec1 Lemma 4} \\
&= \prod_{i=1}^{n-\frac{n}{\sqrt{2}}-1} \left(\frac{n-i-1}{n-i+1}\right) \\
&= \left(\frac{n-2}{n}\right)\left(\frac{n-3}{n-1}\right) ... \left(\frac{n-(n-n/\sqrt{2}-2)-1}{n-(n-n/\sqrt{2}-2)+1}\right)\left(\frac{n-(n-n/\sqrt{2}-1)-1}{n-(n-n/\sqrt{2}-1)+1}\right) \\
&= \left(\frac{n-2}{n}\right)\left(\frac{n-3}{n-1}\right) ... \left(\frac{\frac{n}{\sqrt{2}}+1}{\frac{n}{\sqrt{2}}+3}\right)\left(\frac{\frac{n}{\sqrt{2}}}{\frac{n}{\sqrt{2}}+2}\right) \\
&= \frac{(\frac{n}{\sqrt{2}}+1)(\frac{n}{\sqrt{2}})}{n(n-1)} \\
&= \frac{n(\frac{n}{2}+\frac{\sqrt{2}}{2})}{n(n-1)} \\
&= \frac{n+\sqrt{2}}{2 \cdot (n-1)} \\
&= \frac{1}{2} \cdot \frac{n+\sqrt{2}}{n-1} \\
&\geqslant \frac{1}{2} \quad \text{since } \frac{n+\sqrt{2}}{n-1} >= 1 \text{ for any } n >= 2
\end{aligned}$$

(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)**

Solution. Solution to Problem 1-c

Proof.

- During each recursive step, we contract $n - n/\sqrt{2}$ edges.
- From Part (a), we know that a single contraction takes $O(n)$ time.
- Performing n contractions is $O(n^2)$, thus, performing $n - n/\sqrt{2}$ must necessarily be $\leqslant O(n^2)$.
- Additionally, during each recursive step, we recurse twice on G' which contains $n/\sqrt{2}$ vertices, hence $2 \cdot T(n/\sqrt{2})$.

Bounding runtime to $O(n^2 \cdot \log n)$:

- By the Master Theorem (as described in CS 341), for $a > 0$, $b > 1$, $c \geqslant 0$, if $T(n) = aT(n/b) + n^c$ and $c = \log_b a$, then $T(n) = O(n^c \log n)$.
- Given that $T(n) \leqslant 2 \cdot T(n/\sqrt{2}) + O(n^2)$, we obtain our values $a = 2, b = \sqrt{2}, c = 2$. Then:

$$\begin{aligned} c &= \log_b a \\ 2 &= \log_{\sqrt{2}} 2 \\ 2 &= 2 \end{aligned}$$

We conclude that $T(n) \leqslant O(n^2 \log n)$.

(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).

Solution. Solution to Problem 1-d

This solution uses log to mean $\log_2$.

For the algorithm to succeed, the first part must not destroy the min cut and one of the two copies of the second part needs to succeed. From part b), the probability that the first part does not destroy the min cut is at least $\frac{1}{2}$.

Furthermore, the probability that one of the copies of the second part fails is
probability that both fail is $(1 - P(n/\sqrt{2}))^2$ and thus the probability that a
$1 - (1 - P(n/\sqrt{2}))^2$. Therefore, the probability that the algorithm succeeds is:

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

We now prove that $P(n) \geq \frac{1}{2\log n}$ by induction on n. When $n = 2$, then the alg
and we can verify that $1 \geq \frac{1}{2\log 2}$.

Now assume $P(k) \geq \frac{1}{2\log k}$ for $k = 2, \ldots, n-1$. We wish to prove that this hold

$$\begin{aligned} P(n) &\geq \frac{1 - (1 - P(n/\sqrt{2}))^2}{2} \\ &= \frac{1 - (1 - 2P(n/\sqrt{2}) + P(n/\sqrt{2})^2)}{2} \\ &= P(n/\sqrt{2}) - \frac{P(n/\sqrt{2})^2}{2} \end{aligned}$$

By the inductive hypothesis, we know that $P(n/\sqrt{2}) \geq \frac{1}{2\log(n/\sqrt{2})}$. Moreover
above by 1. Notice that for $0 \leq x \leq 1$, the function $x - \frac{x^2}{2}$ is increasing. Thus,

$$P(n/\sqrt{2}) - \frac{P(n/\sqrt{2})^2}{2} \geq \frac{1}{2\log(n/\sqrt{2})} - \frac{1}{8\log(n/\sqrt{2})^2}$$

Now, it remains to be proved that $\frac{1}{2\log(n/\sqrt{2})} - \frac{1}{8\log(n/\sqrt{2})^2} \geq \frac{1}{2\log(n)}$. Examine

should be "at most" (since $P()$ is the worst-case success probability)

$$\frac{1}{\log(n)} \geq \frac{1}{2\log(n/\sqrt{2})} \qquad \text{(For } n \geq 2\text{)}$$
$$\Rightarrow \frac{1}{2\log(n)} \geq \frac{1}{4\log(n/\sqrt{2})}$$
$$\Rightarrow 1 \geq 1 - \frac{1}{2\log(n)} + \frac{1}{4\log(n/\sqrt{2})}$$
$$\Rightarrow 1 \geq \frac{\log(n)}{\log(n)} - \frac{1/2}{\log(n)} + \frac{1}{4\log(n/\sqrt{2})}$$
$$\Rightarrow 1 \geq \frac{\log(n/\sqrt{2})}{\log(n)} + \frac{1}{4\log(n/\sqrt{2})}$$
$$\Rightarrow 1 - \frac{1}{4\log(n/\sqrt{2})} \geq \frac{\log(n/\sqrt{2})}{\log(n)}$$
$$\Rightarrow \frac{1}{\log(n/\sqrt{2})}\left(1 - \frac{1}{4\log(n/\sqrt{2})}\right) \geq \frac{1}{\log(n)}$$
$$\Rightarrow \frac{1}{\log(n/\sqrt{2})} - \frac{1}{4\log(n/\sqrt{2})^2} \geq \frac{1}{\log(n)}$$
$$\Rightarrow \frac{1}{2\log(n/\sqrt{2})} - \frac{1}{8\log(n/\sqrt{2})^2} \geq \frac{1}{2\log(n)}$$

As desired. ■

The conclusion is that we found an algorithm with runtime O
problem with probability $\Omega(1/\log n)$. As in the class, we can repea
the best answer to succeed with probability at least $2/3$ in $O(n^2 \cdot \log$
now! (on dense graphs, this is just tiny bit slower than reading th

how do you reach this conclusion?

24/25

Q2 25 / 25

Refer to the assignment PDF.

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)

Solution. Solution to Problem 2-a

We will prove $Pr[Y_S = a \wedge Y_T = b] = \frac{1}{4}$ by first proving $Pr[Y_S = 0] = Pr[Y_S = 1] = \frac{1}{2}$ for $S \subseteq \{1, ..., k\}$ and then proving that Y_S and Y_T are independent for $S \neq T \subseteq \{1, ..., k\}$.

First, we prove $Pr[Y_S = 0] = Pr[Y_S = 1] = Pr\left[\bigoplus_{i=1}^{n} X_i = 0\right] = Pr\left[\bigoplus_{i=1}^{n} X_i = 1\right] = \frac{1}{2}$ for $S = \{s_1, s_2, ..., s_n\} \subseteq \{1, ..., k\}$ by induction on n.

When $n = 1$, we have $Pr\left[\bigoplus_{i=1}^{n} X_i = 0\right] = Pr\left[\bigoplus_{i=1}^{n} X_i = 1\right] = Pr[X_1 = 0] = Pr[X_1 = 1] = \frac{1}{2}$ by definition.

Now, assume $Pr\left[\bigoplus_{i=1}^{m} X_i = 0\right] = Pr\left[\bigoplus_{i=1}^{m} X_i = 1\right] = \frac{1}{2}$ for $m = n - 1$. We wish to prove that it holds for $m = n$:

$$\begin{aligned} Pr\left[\bigoplus_{i=1}^{n} X_i = 0\right] &= Pr\left[X_n = 0 \mid \bigoplus_{i=1}^{n-1} X_i = 0\right] \cdot Pr\left[\bigoplus_{i=1}^{n-1} X_i = 0\right] + Pr\left[X_n = 1 \mid \bigoplus_{i=1}^{n-1} X_i = 1\right] \cdot Pr\left[\bigoplus_{i=1}^{n-1} X_i = 1\right] \\ &= \frac{1}{2} \cdot Pr\left[\bigoplus_{i=1}^{n-1} X_i = 0\right] + \frac{1}{2} \cdot Pr\left[\bigoplus_{i=1}^{n-1} X_i = 1\right] \qquad \text{(Independence of } X_i\text{)} \\ &= \frac{1}{2} \cdot \frac{1}{2} + \frac{1}{2} \cdot \frac{1}{2} \\ &= \frac{1}{2} \end{aligned}$$

Similarly,

$$\begin{aligned} Pr\left[\bigoplus_{i=1}^{n} X_i = 1\right] &= Pr\left[X_n = 1 \mid \bigoplus_{i=1}^{n-1} X_i = 0\right] \cdot Pr\left[\bigoplus_{i=1}^{n-1} X_i = 0\right] + Pr\left[X_n = 0 \mid \bigoplus_{i=1}^{n-1} X_i = 1\right] \cdot Pr\left[\bigoplus_{i=1}^{n-1} X_i = 1\right] \\ &= \frac{1}{2} \cdot Pr\left[\bigoplus_{i=1}^{n-1} X_i = 0\right] + \frac{1}{2} \cdot Pr\left[\bigoplus_{i=1}^{n-1} X_i = 1\right] \qquad \text{(Independence of } X_i\text{)} \\ &= \frac{1}{2} \cdot \frac{1}{2} + \frac{1}{2} \cdot \frac{1}{2} \\ &= \frac{1}{2} \end{aligned}$$

As desired. Now, it remains to be proved that Y_S and Y_T are independent for $S \neq T \subseteq \{1, ..., k\}$. We will prove this by showing $Pr[Y_S = a | Y_T = b] = Pr[Y_S = a]$. Without loss of generality, assume $S \setminus T \neq \emptyset$ and take $X_i \in S \setminus T$.

Since $X_i \notin T$, it does not depend on Y_T. That is, $Pr[X_i = c | Y_T = b] = Pr[X_i = c]$. Now, examine:

$$\begin{aligned} Pr[Y_S = a | Y_T = b] &= Pr[X_1 \oplus X_2 \oplus ... \oplus X_i, ... \oplus X_n = a | Y_T = b] \\ &= Pr[X_1 \oplus (X_1 \oplus ... \oplus X_{i-1} \oplus X_{i+1}, ... \oplus X_n) = a | Y_T = b] \end{aligned}$$

If we fix $X_1 \oplus ... \oplus X_{i-1} \oplus X_{i+1}, ... \oplus X_n$, then $Pr[X_i \oplus (X_1 \oplus ... \oplus X_{i-1} \oplus X_{i+1}, ... \oplus X_n) = a | Y_T = b]$ relies independently on X_i. So $Pr[Y_S = a | Y_T = b] = Pr[X_i] = \frac{1}{2}$ and Y_S, Y_T are independent.

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

(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)**

Solution. Solution to Problem 2-b

For each $h \in H$,

1. Generate $\log n$ integers $P_1, \ldots, P_{\log n}$ uniformly at random from $[m]$ so that the values of $P_1, \ldots, P_{\log n}$ will be independent of each other.
2. For each input key, we first convert it into its binary representation. Then, we create a corresponding set $S \subseteq \{1, \ldots, k\}$ by including the indices of these bits whose value is 1.
3. Then, we calculate the hash value $h(s) = \oplus_{i \in S} P_i$. (i.e. for $P_i \oplus P_j$, we XOR the $\log n$ pairs of bits from P_i and P_j respectively, and then convert the resulting $\log m$ bits into an integer in $[m]$.)

From (1), we know that $Pr[P_i = a] = \frac{1}{m}$ for $a \in [m]$.
We first prove $Pr[\oplus_{i \in S} P_i = b] = \frac{1}{m}$ for $b \in [m]$ and $|S| \geq 1$ like what we did in part (a):

$$\begin{aligned} Pr[\oplus_{i \in S} P_i = b] &= Pr[P_{S_1} \oplus P_{S_2} \oplus \cdots \oplus P_{S_k} = b] \\ &= Pr[P_{S_1} \oplus (P_{S_2} \oplus \cdots \oplus P_{S_k}) = b] \end{aligned}$$

Base Case: when $|S| = 1$, $Pr[\oplus_{i \in S} P_i = b] = Pr[P_{S_1} = b] = \frac{1}{m}$ for any P_{S_1} according to step(1)
IH: $Pr[\oplus_{i \in S} P_i = b] = \frac{1}{m}$ for any S where $|S| = k - 1 \geq 1$
IS: for $|S| = k$

$$\begin{aligned} Pr[\oplus_{i \in S} P_i = b] &= Pr[P_{S_1} \oplus (P_{S_2} \oplus \cdots \oplus P_{S_k}) = b] \\ &= Pr[P_{S_1} = b \oplus (P_{S_2} \oplus \cdots \oplus P_{S_k})] \end{aligned}$$

Given IH, we know there are m possible values for $P_{S_2} \oplus \ldots P_{S_k}$. Therefore, for a fixed b,

$$\begin{aligned} Pr[\oplus_{i \in S} P_i = b] &= Pr[P_{S_1} = b \oplus (P_{S_2} \oplus \cdots \oplus P_{S_k})] \\ &= \frac{1}{m} \end{aligned}$$

Conclusion: Hence, $Pr[\oplus_{i \in S} P_i = b] = \frac{1}{m}$ for b in $[m]$ and any such S wi

Now, it remains to be proved that $h(x) = a$ and $h(y) = b$ are independent
$a, b \in [m]$, for h chosen uniformly at random from $\mathcal{H}$. We can prove it by sho

$$Pr[h(x) = a | h(y) = b] = Pr[h(x) = a]$$

Suppose S is the set we get from step(2) for x while T is the set for t. WLO

$$Pr[h(x) = a | h(y) = b] = Pr[(P_z) \oplus (\oplus_{i \in S \setminus \{z\}} P_i) = a | \oplus_{j \in T}$$

Since $z \notin T$, $P_z = a$ does not depend on $\oplus_{j \in T} P_j = b$, which means

$$Pr[P_z = a | \oplus_{j \in T} P_j] = Pr[P_z = a]$$

it's easier to use the independence of the indices and part (a)

For any fixed value of $\oplus_{i \in S \setminus \{z\}} P_i$, we've proved that

$$Pr[\oplus_{i \in S \setminus \{z\}} P_i = a] = \frac{1}{m}$$

Therefore, we have

$$\begin{aligned} Pr[h(x) = a | h(y) = b] &= Pr[(P_z) \oplus (\oplus_{i \in S \setminus \{z\}} P_i) = a | \oplus_{j \in T} \\ &= Pr[P_z = a \oplus (\oplus_{i \in S \setminus \{z\}} P_i) | \oplus_{j \in T} P_j \\ &= Pr[P_z = a \oplus (\oplus_{i \in S \setminus \{z\}} P_i)] \\ &= \frac{1}{m} \end{aligned}$$

Hence, $Pr[h(x) = a | h(y) = b] = Pr[h(x) = a]$ since we first proved that

$$Pr[h(x) = a] = Pr[\oplus_{i \in S} P_i = b] = \frac{1}{m}$$

So,

$$Pr[h(x) = a \wedge h(y) = b] = Pr[h(x) = a] \cdot Pr[h(y) = b] = \frac{1}{m} \cdot \frac{1}{m} = \frac{1}{m^2}$$

Step 1 uses $O(\log n \log m)$ bits, and it takes $poly(\log n, \log m)$ time to generate $\log n$ integers uniformly at random from $[m]$
Step 2 takes $O(\log n)$ time to go through the $\log n$ bits and add those with value 1 to the set. We can store the set using $O(\log n)$ bits.
Step 3 takes $poly(\log m, \log n)$ time to XOR these $\log n$ P_i's. And to store them, it's going to cost $O(\log m)$ bits.
Hence, the hash function is constructed using $O(\log n \log m)$ random bits and $poly(\log n, \log m)$ time to compute.

very good!

Q3

23 / 25

Refer to the assignment PDF.

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)

Solution. Solution to Problem 3-a

Definitions:

- Let D be an arbitrary deterministic algorithm.
- Let $f(x)$ be a function that returns the Boolean value of a leaf node x; that is, $f(x)$ either returns 1 or 0.
- We say that D can make an inference from $f(x)$ if, after querying the value of leaf node x, the value of some other node(s) can be determined.
- Let $p(x)$ be the parent node of x (where x can be a leaf or an internal node).

Given D (and arbitrary n), our solution seeks to provide a method of constructing an input tree, t, (with n leaf nodes) such that D must query all $n = 3^h$ leaves before outputting the value of the root.

Claim 1:

If, after a call to $f(x)$, D still cannot infer the value of $p(x)$, then D cannot infer the value of any other node.

Proof:

- The values of sibling nodes are independent and cannot be inferred from $f(x)$.
- If the value of the grandparent node is not already known, then by definition, it is dependent on the value of the parent node. As such, if we cannot infer the value of $p(x)$, then we cannot infer the value of the grandparent node; by the same argument, we also cannot infer the value of any other ancestor node.

Claim 2:

Suppose D queries the value of leaf node x. If x is the first of its sibling nodes to be queried, then D cannot infer the value of $p(x)$, regardless of the value of $f(x)$.

Claim 3:

Suppose D queries the value of leaf node x. If x is the second of its sibling nodes to be queried (and the first sibling returned a Boolean value of k), then D cannot infer the value of $p(x)$ if $f(x) = -k$. If this is the case, it follows that the value of $p(x)$ is equal to the value of the final sibling node.

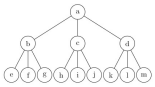
Using the Claims above, we define the values of the leaf nodes in t:

- If leaf node x is the first of its siblings to be queried, then we set $f(x) = 0$.
- If leaf node x is the second of its siblings to be queried, then we set $f(x) = 1$.

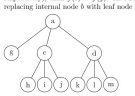
By Claim 2 and 3, D would not be able to infer the value of the parent node without querying
sibling node. Furthermore, the value of the parent node is equal to the value of the final sib
It remains to define the value of the final sibling node. To do this, observe the following exa

11

why are there no proofs for claims 2 and 3?



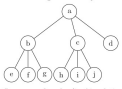
Suppose, without loss of generality, that out of the three sibling leaf nodes of $\{e, f, g\}$, nodes e and f were queried first and second (as such, we assign them the values of $f(e) = 0$ and $f(f) = 1$, respectively); then $f(b) = f(g)$. As such, we simplify the tree by removing all child-nodes of b and replacing internal node b with leaf node g, as follows:



By extending this argument, we can simplify the subtrees of nodes c and d as well (suppose, without loss of generality, that h is the final sibling node to be queried from $\{h, i, j\}$ and m is the final sibling node to be queried from $\{k, l, m\}$):

Once again, we have three sibling leaf nodes $\{g, h, m\}$ and we can use the same process for defining their values. In general, this process can be applied recursively (once we reach the root, we can define the value of the final child-node to be either 0 or 1).

Similarly, we can use the same way for complete trees with the last level not fully filled. However, it's worth showing that the way we defined above can handle the following case as well:



Suppose e, f, and g (in this relative order) are queried before knowing the values of c and d. We set $f(e) = 0$ and $f(f) = 1$ following the rules. Then we simplify the tree by replacing the subtree rooted at b by the leaf node g.

12

Then, when g is queried, note that even though g is the last sibling of $\{e, f, g\}$ to be queried, it's the first sibling of $\{g, c = \{h, i, j\}, d\}$ to be queried. Therefore, following the way we defined previously, we must set $f(g) = 0$ here.

Using this approach to construct an input tree for a given D, we have shown that there exists an instance for each deterministic algorithm such that the algorithm must query all $n = 3^h$ leaves.

the argument is nice and correct but you need to justify your claims

(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)**

Solution. Solution to Problem 3-b

Since each internal node has exactly 3 child-nodes, observe that there are two possible configurations for the Boolean values of the child-nodes:

- All child-nodes share the same Boolean value. In this case, the probability of randomly selecting two child-nodes such that their values disagree is 0.
- Two child-nodes share the same Boolean value (and one child-node has a different Boolean value). In this case, the probability of randomly selecting two child-nodes such that their values disagree is $\binom{2}{1}\binom{1}{1}\Big/\binom{3}{2} = \frac{2}{3}$.

Let $f(n)$ return the expected number of leaves queried by the algorithm on a tree with n leaves, then:

$$f(n) \leqslant \frac{1}{3}(2 \cdot f(n/3)) + \frac{2}{3}(3 \cdot f(n/3))$$

We will prove $f(n) \leqslant n^{0.9}$. The proof is by induction on n (note that n must be a multiple of 3 as defined in the problem statement).

Base Case:

$n = 3$ (given that the root node must have exactly three child-nodes). $f(3) = \frac{1}{3}(2) + \frac{2}{3}(3) = \frac{8}{3} \leqslant 3^{0.9}$ as required.

Inductive Step:

Now, assume $f(k) \leqslant k^{0.9}$ for $k = n/3$. We wish to prove that it holds for n.

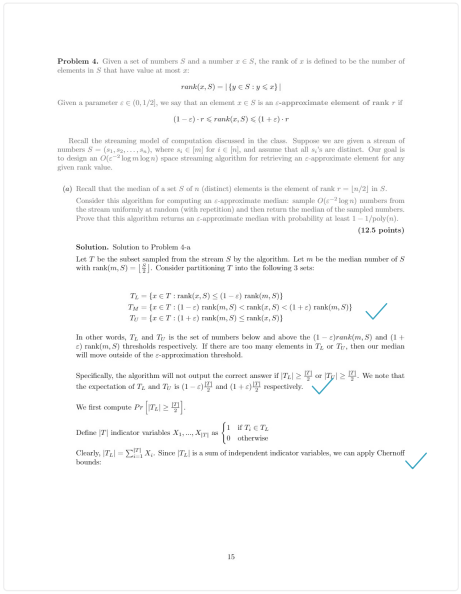
$$\begin{aligned}
f(n) &\leqslant \frac{1}{3}(2 \cdot f(n/3)) + \frac{2}{3}(3 \cdot f(n/3)) \\
&\leqslant \frac{1}{3}\left(2 \cdot \left(\frac{n}{3}\right)^{0.9}\right) + \frac{2}{3}\left(3 \cdot \left(\frac{n}{3}\right)^{0.9}\right) && \text{Inductive hypothesis} \\
&= \frac{2}{3}\left(\frac{n}{3}\right)^{0.9} + 2\left(\frac{n}{3}\right)^{0.9} \\
&= \frac{2n^{0.9}}{3^{1.9}} + \frac{2n^{0.9}}{3^{0.9}} \\
&= \frac{2n^{0.9} + 2(3)n^{0.9}}{3^{1.9}} \\
&= \frac{n^{0.9}(2+6)}{3^{1.9}} \\
&= \frac{n^{0.9}(8)}{3^{1.9}} \\
&\leqslant n^{0.9}
\end{aligned}$$

As desired. ■

very good! 23/25

Q4 25 / 25

Refer to the assignment PDF.



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:

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

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

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

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 $1 - 1/\mathrm{poly}(n)$.
(12.5 points)

Solution. Solution to Problem 4-a

Let T be the subset sampled from the stream S by the algorithm. Let m be the median number of S with $\mathrm{rank}(m, S) = \lfloor \frac{n}{2} \rfloor$. Consider partitioning T into the following 3 sets:

$$T_L = \{x \in T : \mathrm{rank}(x, S) \leq (1-\varepsilon)\ \mathrm{rank}(m, S)\}$$
$$T_M = \{x \in T : (1-\varepsilon)\ \mathrm{rank}(m, S) < \mathrm{rank}(x, S) < (1+\varepsilon)\ \mathrm{rank}(m, S)\}$$
$$T_U = \{x \in T : (1+\varepsilon)\ \mathrm{rank}(m, S) \leq \mathrm{rank}(x, S)\}$$

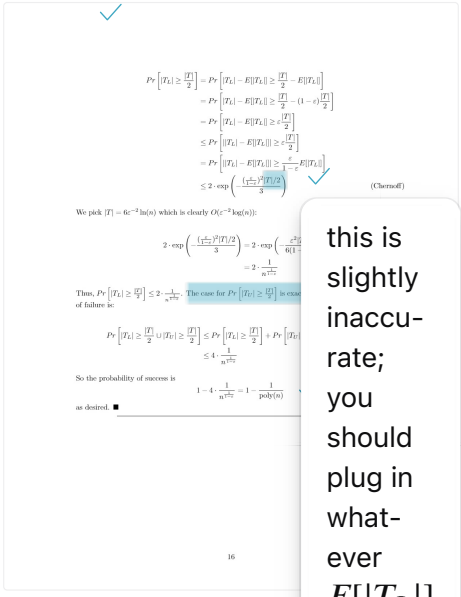
In other words, T_L and T_U is the set of numbers below and above the $(1-\varepsilon)\mathit{rank}(m, S)$ and $(1+\varepsilon)\ \mathrm{rank}(m, S)$ thresholds respectively. If there are too many elements in T_L or T_U, then our median will move outside of the ε-approximation threshold.

Specifically, the algorithm will not output the correct answer if $|T_L| \geq \frac{|T|}{2}$ or $|T_U| \geq \frac{|T|}{2}$. We note that the expectation of T_L and T_U is $(1-\varepsilon)\frac{|T|}{2}$ and $(1+\varepsilon)\frac{|T|}{2}$ respectively.

We first compute $Pr\left[|T_L| \geq \frac{|T|}{2}\right]$.

Define $|T|$ indicator variables $X_1, ..., X_{|T|}$ as $\begin{cases} 1 & \text{if } T_i \in T_L \\ 0 & \text{otherwise} \end{cases}$

Clearly, $|T_L| = \sum_{i=1}^{|T|} X_i$. Since $|T_L|$ is a sum of independent indicator variables, we can apply Chernoff bounds:



$$\begin{aligned}
Pr\left[|T_L| \geq \frac{|T|}{2}\right] &= Pr\left[|T_L| - E[|T_L|] \geq \frac{|T|}{2} - E[|T_L|]\right] \\
&= Pr\left[|T_L| - E[|T_L|] \geq \frac{|T|}{2} - (1-\varepsilon)\frac{|T|}{2}\right] \\
&= Pr\left[|T_L| - E[|T_L|] \geq \varepsilon\frac{|T|}{2}\right] \\
&\leq Pr\left[||T_L| - E[|T_L|]| \geq \varepsilon\frac{|T|}{2}\right] \\
&= Pr\left[|T_L| - E[|T_L|]| \geq \frac{\varepsilon}{1-\varepsilon}E[|T_L|]\right] \\
&\leq 2 \cdot \exp\left(-\frac{(\frac{\varepsilon}{1-\varepsilon})^2 |T|/2}{3}\right) \qquad \text{(Chernoff)}
\end{aligned}$$

We pick $|T| = 6\varepsilon^{-2}\ln(n)$ which is clearly $O(\varepsilon^{-2}\log(n))$:

$$2 \cdot \exp\left(-\frac{(\frac{\varepsilon}{1-\varepsilon})^2 |T|/2}{3}\right) = 2 \cdot \exp\left(-\frac{\varepsilon^2 |T|}{6(1-\varepsilon)^2}\right)$$
$$= 2 \cdot \frac{1}{n^{\frac{1}{(1-\varepsilon)^2}}}$$

Thus, $Pr\left[|T_L| \geq \frac{|T|}{2}\right] \leq 2 \cdot \frac{1}{n^{\frac{1}{(1-\varepsilon)^2}}}$. The case for $Pr\left[|T_U| \geq \frac{|T|}{2}\right]$ is exac
of failure is:

$$Pr\left[|T_L| \geq \frac{|T|}{2} \cup |T_U| \geq \frac{|T|}{2}\right] \leq Pr\left[|T_L| \geq \frac{|T|}{2}\right] + Pr\left[|T_U|$$
$$\leq 4 \cdot \frac{1}{n^{\frac{1}{(1-\varepsilon)^2}}}$$

So the probability of success is

$$1 - 4 \cdot \frac{1}{n^{\frac{1}{(1-\varepsilon)^2}}} = 1 - \frac{1}{\mathrm{poly}(n)}$$

as desired. ■

this is slightly inaccurate; you should plug in whatever $E[|T_L|]$ is

(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 $t = \lceil 24\epsilon^{-2} \log m \rceil$. 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)**

Solution. Solution to Problem 4-b

Case 1: $r \leqslant t$

Since we always select the r-th smallest element in the stream, by definition of ϵ-approximate rank, this always succeeds. As such, the probability of success in this case is at least $1-1/poly(n)$.

Case 2: $r \geqslant t$

Definitions:

- Let S be the set of all elements in the stream.
- Let A be the set of all elements selected from the stream.
- Let $A_L = \{x \in A : rank(x, S) < (1-\epsilon)r\}$.
- Let $A_M = \{x \in A : (1-\epsilon)r \leqslant rank(x, S) \leqslant (1+\epsilon)r\}$.

Observe that A, A_L, A_M are all sums of independent indicator random variables:

- Define $|S|$ indicator variables $X_1, ..., X_{|S|}$ as $\begin{cases} 1 & \text{if } S_i \in A \\ 0 & \text{otherwise} \end{cases}$, clearly, $|A| = \sum_{i=1}^{|S|} X_i$
- Define $|S|$ indicator variables $X_1, ..., X_{|S|}$ as $\begin{cases} 1 & \text{if } S_i \in A_L \\ 0 & \text{otherwise} \end{cases}$, clearly, $|A_L| = \sum_{i=1}^{|S|} X_i$
- Define $|S|$ indicator variables $X_1, ..., X_{|S|}$ as $\begin{cases} 1 & \text{if } S_i \in A_M \\ 0 & \text{otherwise} \end{cases}$, clearly, $|A_M| = \sum_{i=1}^{|S|} X_i$

We define all failure cases as follows:

- $|A| = 0$, always fails
- $0 < |A| < t$, fails when $|A_M| = 0$ or $|A_L| + |A_M|$
- $|A| \geqslant t$, fails when $|A_L| \geqslant t$ or $|A_L| + |A_M| < t$

For simplicity, we upper bound the failure cases as f

(i) $|A_L| \geqslant t$, always fails

(ii) $|A_L| + |A_M| < t$, always fails

By taking the sum of all failure cases, we obtain an this can then be used to lower bound the probability element of a desired rank r.

Important values:

- $E[|A_L|] = (1-\epsilon)r \cdot \frac{t}{r} = (1-\epsilon)t$. As such, note
- $E[|A_L| + |A_M|] = (1+\epsilon)r \cdot \frac{t}{r} = (1+\epsilon)t$ As suc

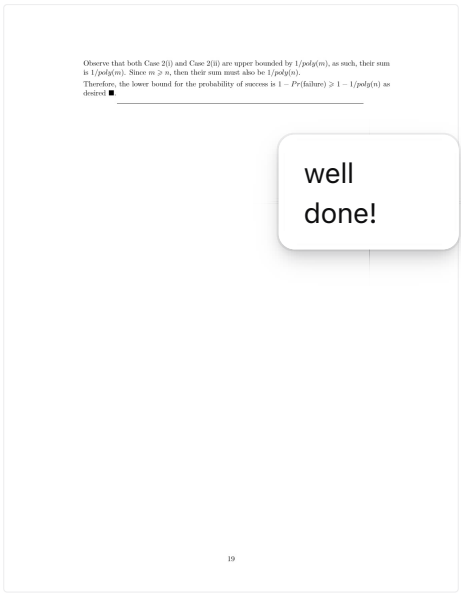
you should give them different names

Case 2(i):

$$
\begin{aligned}
&Pr(|A_L| \geqslant t) \\
&\leqslant Pr(||A_L| - E[|A_L|]| \geqslant t - E[|A_L|]) \\
&= Pr(||A_L| - E[|A_L|]| \geqslant t - (1-\epsilon)t) \\
&= Pr(||A_L| - E[|A_L|]| \geqslant \epsilon t) \\
&= Pr(||A_L| - E[|A_L|]| \geqslant \left(\tfrac{\epsilon}{1-\epsilon}\right)(1-\epsilon)t) \\
&= Pr(||A_L| - E[|A_L|]| \geqslant \left(\tfrac{\epsilon}{1-\epsilon}\right) E[|A_L|]) \\
&\leqslant 2 \cdot exp\left(-\frac{\left(\frac{\epsilon}{1-\epsilon}\right)^2 (1-\epsilon)t}{3}\right) && \text{Chernoff} \\
&= 2 \cdot exp\left(-\frac{\epsilon^2 t}{3(1-\epsilon)}\right) \\
&\leqslant 2 \cdot exp\left(-\frac{\epsilon^2 24\epsilon^{-2} \log m}{3(1-\epsilon)}\right) && \text{Given that } 24\epsilon^{-2}\log m \leqslant t \\
&= 2 \cdot exp\left(-\frac{8 \log m}{1-\epsilon}\right) \\
&= 2 \cdot \left(\frac{1}{m}\right)^{\frac{8}{1-\epsilon}}
\end{aligned}
$$

Case 2(ii):

$$
\begin{aligned}
&Pr(|A_L| + |A_M| < t) \\
&= Pr(|A_L| + |A_M| - E[|A_L| + |A_M|| < t - E[|A_L| + |A_M|]) \\
&= Pr(|A_L| + |A_M| - E[|A_L| + |A_M|| < t - (1+\epsilon)t) \\
&= Pr(|A_L| + |A_M| - E[|A_L| + |A_M|| < -\epsilon t) \\
&\leqslant Pr(||A_L| + |A_M| - E[|A_L| + |A_M|| \geqslant \epsilon t) \\
&= Pr(||A_L| + |A_M| - E[|A_L| + |A_M|| \geqslant \left(\tfrac{\epsilon}{1+\epsilon}\right)(1+\epsilon)t) \\
&= Pr(||A_L| + |A_M| - E[|A_L| + |A_M|| \geqslant \left(\tfrac{\epsilon}{1+\epsilon}\right) E[|A_L| + |A_M|]) \\
&\leqslant 2 \cdot exp\left(-\frac{\left(\frac{\epsilon}{1+\epsilon}\right)^2 (1+\epsilon)t}{3}\right) && \text{Chernoff} \\
&= 2 \cdot exp\left(-\frac{\epsilon^2 t}{3(1+\epsilon)}\right) \\
&\leqslant 2 \cdot exp\left(-\frac{\epsilon^2 24\epsilon^{-2} \log m}{3(1+\epsilon)}\right) && \text{Given that } 24\epsilon^{-2}\log m \leqslant t \\
&= 2 \cdot exp\left(-\frac{8 \log m}{1+\epsilon}\right) \\
&= 2 \cdot \left(\frac{1}{m}\right)^{\frac{8}{1+\epsilon}}
\end{aligned}
$$



Observe that both Case 2(i) and Case 2(ii) are upper bounded by $1/poly(m)$, as such, their sum is $1/poly(m)$. Since $m \geqslant n$, then their sum must also be $1/poly(n)$.
Therefore, the lower bound for the probability of success is $1 - Pr(\text{failure}) \geqslant 1 - 1/poly(n)$ as desired ■.

19

well done!

Q5 0 / 10

Refer to the assignment PDF.

This question wasn't answered