CIS 677

Homework 1

Daniel Sparber

Collaborators: Julian Schnitzler & Seyoung Kim

Problem 1

Input

Matrix $M \in \mathbb{R}^{n \times n}$

Output

Return `true` if M is a diagonal matrix, `false` otherwise.

Algorithm

Algorithm A

1. Pick $v \in \{0,1\}^n$ uniformly at random.
2. Calculate $x = Mv$
3. Calculate $y = Mu$, where $u = [1, \cdots, 1]^T$
4. Return `true` if $\forall i \in \{1, \dots, n\} : y_i \cdot v_i = x_i$, otherwise `false`

Algorithm B

Repeat Algorithm A $\log n$ times. If A always outputs `true`, return `true`, otherwise return `false`.

Analysis

Runtime

Algorithm A performs exactly 2 matrix-vector query. Algorithm B calls A $\log n$ times. Thus, algorithm B queries $\mathcal{O}(\log n)$ vector-matrix products.

Correctness

Let $M \in \mathbb{R}^{n \times n}$ be arbitrary.

Case 1: M is diagonal

Claim

Algorithm B accepts

Proof

Lemma 1: Given M is diagonal, algorithm A always returns `true`.

Let $v \in \{0,1\}^n$ be arbitrary, M is diagonal.

d_i, $i \in \{1, \dots, n\}$ denotes the i-th diagonal element of M.

$x = Mv = [d_1 v_1, \dots, d_n v_n]^T$

$y = Mu = [d_1, \dots, d_n]^T$

$\forall i \in \{1, \dots, n\}$: $y_i \cdot v_i = d_i \cdot v_i = x_i$. Thus, the algorithm A returns `True`.

Using lemma 1:

A always returns `true`. Thus, B returns `true`.

Case 2: M is not diagonal

Claim

Algorithm B rejects with probability at least $1 - \frac{1}{n}$, i.e. $\Pr[B \textit{ error}] \leq \frac{1}{n}$

Proof

Lemma 2: Given M is not diagonal, algorithm A rejects with probability at least $\frac{1}{2}$, i.e. $\Pr[A \textit{ error}] \leq \frac{1}{2}$

Let $v \in \{0,1\}^n$ be arbitrary, M is not diagonal.

Since M is not diagonal, $m_{i,j} \neq 0$ for some, $i, j \in \{1, \dots, n\}$ where $i \neq j$

$\Pr[A \textit{ error}] \leq \Pr][\textit{Error in line } i] = \Pr[y_i \cdot v_i = x_i] = \Pr[y_i \cdot v_i - x_i = 0]$

$x_i = m_{i,j} \cdot v_j + \sum_{k \in \{1,\cdots,n\} k \neq j} m_{i,k} \cdot v_k$

$y_i \cdot v_i = m_{i,j} \cdot v_i + \sum_{k \in \{1,\cdots,n\} k \neq j} m_{i,k} \cdot v_i$

$y_i \cdot v_i - x_i = m_{i,j} \cdot v_i - m_{i,j} \cdot v_j + S$, where $S = \sum_{k \in \{1,\cdots,n\} k \neq j} m_{i,k} \cdot (v_i - v_k)$

- Case $S = 0$:

  $\Pr[A \textit{ error}] \leq \Pr[m_{i,j} \cdot v_i - m_{i,j} \cdot v_j = 0] = \Pr[v_i = v_j] = \frac{1}{2}$

- Case $S \neq 0$:

  $\Pr[A \ error] \leq \Pr[m_{i,j} \cdot v_i - m_{i,j} \cdot v_j + S = 0] \leq \Pr[v_i \neq v_j] = \frac{1}{2}$

Using lemma 2:

$\Pr[B \ error] = (\Pr[A \ error])^{\log n} \leq \frac{1}{2}^{\log n} = \frac{1}{n}$

Problem 2

Claim

C = BPP

Proof

By proofing C $\subseteq$ BPP and BPP $\subseteq$ C.

BPP $\subseteq$ C

Let $L \in$ BPP be arbitrary.

By definition of BPP: There exists a poly-time algorithm A which on any input x has the following behavior:

- If $x \in L$, then $\Pr[A \textit{ accepts } x] > \frac{3}{4}$
- If $x \notin L$, then $\Pr[A \textit{ rejects } x] > \frac{3}{4}$

A is also a valid algorithm for C, since A fulfills the definition of C:

Let $\text{poly}(n) = 4$:

- If $x \in L$, then $\Pr[A \textit{ accepts } x] \geq \frac{3}{4} = \frac{1}{2} + \frac{1}{4} = \frac{1}{2} + \frac{1}{\text{poly}(n)}$
- If $x \notin L$, then $\Pr[A \textit{ rejects } x] \geq \frac{3}{4} \geq \frac{1}{2}$

C $\subseteq$ BPP

Let $L \in$ C be arbitrary.

By definition of C: There exists a poly-time algorithm A which on any input x has the following behavior:

- If $x \in L$, then $\Pr[A \textit{ accepts } x] \geq \frac{1}{2} + \frac{1}{\text{poly}(n)}$
- If $x \notin L$, then $\Pr[A \textit{ rejects } x] \geq \frac{1}{2}$

Let B be an algorithm defined as follows:

- Repeat algorithm A k times. Accept, if at least $l = \frac{k}{2} + \frac{k}{2\,\text{poly}(n)}$ of executions of A accepted, otherwise reject.

Where $k = 32(\text{poly}(n))^2 \log(2)$

Claim

B is a valid algorithm for BPP, i.e. B has the following properties:

- If $x \in L$, then $\Pr[B \text{ accepts } x] > \frac{3}{4}$
- If $x \notin L$, then $\Pr[B \text{ rejects } x] > \frac{3}{4}$

Proof

- Assume $x \in L$:

  Let X_i be a 0/1 random variable, that indicates if the i-th execution of A accepted.

  Let $X = \sum_{i=1}^{k} X_i$, $E[X] = k \cdot \left(\frac{1}{2} + \frac{1}{\text{poly}(n)}\right) = \frac{k}{2} + \frac{k}{\text{poly}(n)}$

  $\Pr[B \text{ accepts } x] = \Pr[A \text{ accepts } x \text{ at least } l \text{ times}] = \Pr[X \geq l] = 1 - \Pr[X < l] = 1 - \Pr[X \leq l]$

  X consists of independent 0/1 random variables, thus Chernoff bounds with additive error can be applied:

  $$\Pr[X \leq l] = \Pr\left[X \leq \frac{k}{2} + \frac{k}{2\,\text{poly}(n)}\right] = \Pr\left[X \leq \left(\frac{k}{2} + \frac{k}{\text{poly}(n)}\right) - k \cdot \frac{1}{2\,\text{poly}(n)}\right] < e^{-\frac{\frac{1}{(2\,\text{poly}(n))^2}k}{2}}$$

  $$= e^{-\frac{\frac{1}{4(\text{poly}(n))^2}32(\text{poly}(n))^2\log(2)}{2}} = e^{-\log(16)} = \frac{1}{16} \leq \frac{1}{4}$$

  $\Rightarrow \Pr[B \text{ accepts } x] > \frac{3}{4}$

- Assume $x \notin L$:

  Let Y_i be a 0/1 random variable, that indicates if the i-th execution of A accepts.

  Let $Y = \sum_{i=1}^{k} Y_i$, $E[Y] = k \cdot \frac{1}{2} = \frac{k}{2}$

  $\Pr[B \text{ rejects } x] = \Pr[A \text{ accepts } x \text{ less than } l \text{ times}] = \Pr[Y < l] = 1 - \Pr[Y \geq l]$

  Applying Chernoff:

  $$\Pr[Y \geq l] = \Pr\left[Y \geq \frac{k}{2} + \frac{k}{2\,\text{poly}(n)}\right] = \Pr\left[Y \geq \frac{k}{2} + k \cdot \frac{1}{2\,\text{poly}(n)}\right] < e^{-\frac{\frac{1}{(2\,\text{poly}(n))^2}k}{4}}$$

  $$= e^{-\frac{\frac{1}{4(\text{poly}(n))^2}32(\text{poly}(n))^2\log(2)}{4}} = e^{-\log(4)} = \frac{1}{4}$$

  $\Rightarrow \Pr[B \text{ rejects } x] > \frac{3}{4}$

Since A is a polynomial-time algorithm and A is executed $32(\text{poly}(n))^2\log(2)$ times, B is a polynomial-time algorithm.

Thus, B is a valid algorithm for BPP.

Problem 3

Part a - Deterministic algorithm

To show

For any deterministic algorithm and $h \geq 1$, there is an instance that forces the algorithm to query all $n = 3^h$ leaves in order to compute the value of the root.

Proof

Proof by induction over h.

Induction hypothesis (IH): For any deterministic algorithm A and $h \geq 1$: If the algorithm only looks at $n-1$ leaves, there exist two trees T_1 and T_2 with different roots for which the A computes the same root value.

Let A be an arbitrary deterministic algorithm that computes the root of a given tree.

Case $h = 1$

Let T_1 be a tree with the leaves v_1, v_2, v_3, and T_2 a tree with the leaves u_1, u_2, u_3.

Assume A looks only at $n - 1 = 2$ leaves. Let v_1, v_2 respectively u_1, u_2 be the values A looks at.

For $v_1 = \texttt{True}, v_2 = \texttt{False}, v_3 = \texttt{False}$ and $u_1 = \texttt{True}, u_2 = \texttt{False}, u_3 = \texttt{True}$, T_1 has root `false` and T_2 has root `true`.

Since A only looks at the first two leaves, it must have the same state for T_1 and T_2, thus it will compute the same root value for both trees.

Case $h > 2$

Let T_1 be a tree with the nodes v_1, v_2, v_3 as children of the root, and T_2 a tree with the nodes u_1, u_2, u_3 as children of the root.

Assume A looks only at $n-1$ leaves. Without loss of generality: A does not look at one leaf contained in v_3 respectively u_3.

By induction hypothesis, there exist two trees $\bar{T}_1$ and $\bar{T}_2$ with different roots for which A computes the same value.

For any $v_1 = u_1 = \bar{T}_1$, $v_2 = u_2 = \bar{T}_2$, $v_3 = \bar{T}_1$ and $u_3 = \bar{T}_2$, T_1 and T_2 have a different root, but A computes the same root value

q.e.d.

Thus, any deterministic algorithm must look at all n leaves to compute the value at the root.

Part b - Randomized algorithm

Claim

The expected number of the leaves queried by the algorithm on any instance is at most $n^{0.9}$.

Proof

Let T be arbitrary, let $h \geq 1$ be arbitrary. Assume h is the height of tree T.

Let X_h be the number of leaves queried by the algorithm for the tree T with height $h \geq 1$.

Let p_a be the probability, that two values chosen uniformly at random have the same value.

Two sub trees need to be evaluated. If the values disagree, the third sub tree needs to be evaluated.

The cost of evaluating a leaf is 1. The cost of evaluating an inner node v is equal to evaluating the sub tree with root v. Thus evaluating a tree with height $h - 1$.

Therefore, he expected value can be defined recursively as following:

$\mathbb{E}[X_1] = p_a \cdot 2 \cdot 1 + (1 - p_a) \cdot 3 \cdot 1 = (3 - p_a)$

$\mathbb{E}[X_h] = p_a \cdot 2 \cdot \mathbb{E}[X_{h-1}] + (1 - p_a) \cdot 3 \cdot \mathbb{E}[X_{h-1}] = (3 - p_a) \cdot \mathbb{E}[X_{h-1}]$

Any inner node of T has either three children with the same value or two children with one value and one child with the other value.

- Case 1: Node has different child values

  To get two agreeing values, the first and the second child need to have the same value, i.e.:

  $p_a = \frac{2}{3} \cdot \frac{1}{2} = \frac{1}{3}$

- Case 2: Node has same child values

  No matter which two children are chosen, the value always agrees, thus:

  $p_a = 1$

Thus $p_a \geq \frac{1}{3}$, and therefore:

$\mathbb{E}[X_1] \leq (3 - \frac{1}{3}) = \frac{8}{3}$

$\mathbb{E}[X_h] \leq (3 - \frac{1}{3}) \cdot \mathbb{E}[X_{h-1}] = \frac{8}{3} \cdot \mathbb{E}[X_{h-1}]$

Lemma: $\forall h \geq 1 : \mathbb{E}[X_h] \leq \left(\frac{8}{3}\right)^h$

Proof by induction over h.

Induction hypothesis IH: $\mathbb{E}[X_h] \leq \left(\frac{8}{3}\right)^h$

Let $h \geq 1$ be arbitrary.

- Case $h = 1$:

  $\mathbb{E}[X_1] \leq \frac{8}{3} = \left(\frac{8}{3}\right)^1$

- Case $h \geq 1$:

  $\mathbb{E}[X_h] \leq \frac{8}{3} \cdot \mathbb{E}[X_{h-1}] = \frac{8}{3} \cdot \left(\frac{8}{3}\right)^{h-1} = \left(\frac{8}{3}\right)^h$

Thus, IH holds.

By applying *Lemma*:

$\mathbb{E}[X_h] \leq \left(\frac{8}{3}\right)^h \leq (3^{0.9})^h = (3^h)^{0.9} = n^{0.9}$

Problem 4

Part a - Deterministic algorithm

Let $U = \{1, 2, \dots, n\}$

For any integer x, x_k is the k-th bit of x in binary representations

Input

$V \in \mathbb{R}^n_{\geq 0}$, with at least one non-zero element.

Output

Returns unique index i such that $V[i] \neq 0$ whenever V contains exactly one non-zero element. Otherwise outputs "More than one non-zero entry".

Algorithm

1. $a = 0$
2. For $k \in \{0, \dots, \lceil \log n \rceil\}$:
   - $S_k := \{s \mid s \in U \text{ if } s_k = 0\}$, $\bar{S}_k = U \setminus S_k$
   - if $Q(S_k) \neq 0$ and $Q(\bar{S}_k) \neq 0$: return "More than one non-zero entry"
   - $a_k = 1$ if $Q(S_k) \neq 0$ else 0
3. return a

Run time

Run time in terms of summation queries:

Every iteration performs $\mathcal{O}(1)$ queries. There is a total number of $\lceil \log n \rceil$ iterations.

Thus, the algorithm performs $\mathcal{O}(\log n)$ summation queries.

Correctness

- Case 1: More than one non-zero entry

  There exists $i, j \in U$ with $i \neq j$, $V[i] > 0$ and $V[j] > 0$.

Since $i \neq j$, there exists at least one index k with $i_k \neq j_k$.

Thus, only i or j is in S_k. Therefore, $Q(S_k) > 0$ and $Q(U \setminus S_k) > 0$

The algorithm always outputs "More than one non-zero entry"

- Case 2: Exactly on non-zero entry

  There exists a unique b with $V[b] > 0$

  For every $k \in \{0, \dots, \lceil \log n \rceil\}$:

  - Case $b_k = 0$: $b \notin S_k$, since $V[b]$ is the only non-zero element: $Q(S_k) = 0 \Rightarrow a_k = 0$
  - Case $b_k = 1$: $b \in S_k \Rightarrow Q(S_k) = V[b] > 0 \Rightarrow a_k = 1$,
  - Therefore $b_k = a_k$

  $\Rightarrow a = b$

  Since b is unique, the algorithm returns the only correct index.

Part b - Randomized algorithm

Input

$V \in \mathbb{R}^n_{\geq 0}$, with exactly $d \in [1..n]$ non-zero elements, accuracy parameter δ.

Output

Returns an index i with $V[i] \neq 0$ with probability of at least $1 - \delta$.

Algorithm

- Repeat k times:
  1. For $i \in [1..n]$: Draw $x_i \in \{0, 1\}$ with $\Pr[x_i = 1] = \frac{1}{d}$
  2. Let $S = \{i \mid i \in [1..n] \text{ if } x_i = 1\}$
  3. If $Q(S) \neq 0$:
     - Run deterministic algorithm with the following modification:

       Instead of using S_k, $S'_k = \{s \mid s \in S_k \text{ if } x_s = 1\}$ is used.

       Instead of using $\bar{S_k}$, $\bar{S_k}' = \{s \mid s \in \bar{S_k} \text{ if } x_s = 1\}$ is used.
     - If index is returned by deterministic algorithm: return index
- return 0

Where $k = 9 \cdot \log(1/\delta)$ and $Q_W(S) = \sum_{j \in S} W[j]$ is the summation query over vector W

Run time

If $d = 1$, the algorithm runs in $\mathcal{O}(\log n)$.

Otherwise, one iteration of the algorithm performs one summation query plus at most $\mathcal{O}(\log n)$ summation queries by calling the deterministic algorithm.

Thus, one iteration performs $\mathcal{O}(\log n)$ summation queries.

By repeating $k \in \mathcal{O}(\log(1/\delta))$ times, the total number of summation queries is in $\mathcal{O}(\log(n) \cdot \log(1/\delta))$.

Correctness

- Case $d = 1$:

For every $i \in [1, \dots, n]$ $x_i = 1$ with probability 1. Thus, the algorithm simply performs the deterministic algorithm. One iteration of the randomized algorithm is performed.

The correctness follows from correctness of deterministic algorithm. The probability of returning the correct index is 1 and therefore at least $1 - \delta$.

- Case $d \neq 1$

Every index i, where $x_i = 0$, is never in any set S'_k and $\bar{S_k}'$. Thus, both summation queries never contains $V[i]$.

Likewise, every index i, where $x_i = 1$ is always in S'_k whenever i is in S_k and in $\bar{S_k}'$ whenever i is in $\bar{S_k}$

Let $W = [V[0] \cdot x_0, \dots, V[n] \cdot x_n]$

For this definition of W, performing the summation queries on W with S_k and $\bar{S_k}$ is equivalent to performing the summation query on V with S'_k and $\bar{S_k}'$. In the following steps, the equivalent problem, using W is analyzed.

Analyzing one iteration:

Claim: W contains exactly one non-zero element with probability $p \geq \frac{1}{4}$

Proof:

Every index with a zero element in V is also an index with a zero element in W by construction of W.

Therefore, only non-zero elements of V might remain in W. By construction, each of the d non-zero element remains in W with probability $\frac{1}{d}$. Let X be the amount of non-zero elements remaining in W.

$X \sim \text{Binomial}\left(d, \frac{1}{d}\right)$

$\Pr[X = 1] = \binom{d}{1} \cdot \frac{1}{d} \cdot \left(1 - \frac{1}{d}\right)^{d-1} = \left(1 - \frac{1}{d}\right)^{d-1} = \frac{\left(1-\frac{1}{d}\right)^d}{1-\frac{1}{d}} \geq \frac{\left(1-\frac{1}{d}\right)^d}{1-0} = \left(1 - \frac{1}{d}\right)^d$

Since $d \in [1..n]$ and $d \neq 1$: $d \geq 2$

For $d = 2$: $\left(1 - \frac{1}{d}\right)^d = \frac{1}{4}$

The term $\left(1 - \frac{1}{d}\right)^d$ is monotonically increasing as d increases (see calculus).

Thus, $\left(1 - \frac{1}{d}\right)^d \geq \frac{1}{4}$

$\Rightarrow \Pr[X = 1] \geq \frac{1}{4}$

q.e.d

The if condition ensures, that W contains at least one non-zero element.

Therefore, the input conditions for the deterministic algorithm are met.

Whenever W contains exactly one non-zero element, the correct index of that element is returned by the deterministic algorithm. Otherwise, a string is returned.

By construction of W, this index is also a valid non-zero index in the original vector. Hence, the entire algorithm succeeds and returns a valid index.

The probability of W containing exactly one non-zero element is $p \geq \frac{1}{4}$. Thus, one iteration succeeds with $p \geq \frac{1}{4}$.

After k iterations

The algorithm returns a correct element whenever one iteration returns.

The algorithm can only fail, if no valid index was found within k iterations, i.e. if every iteration fails.

$$\Rightarrow \Pr[error] \leq (1-p)^k \leq \left(1-\tfrac{1}{4}\right)^k = \left(\tfrac{3}{4}\right)^k = e^{k\cdot\log(3/4)} =$$

$$= e^{9\cdot\log(1/\delta)\cdot\log(3/4)} \leq e^{\frac{-1}{\log(3/4)}\cdot\log(1/\delta)\cdot\log(3/4)} = e^{-\log(1/\delta)} = e^{\log\delta} = \delta$$

$$\Rightarrow \Pr[success] = 1 - \Pr[error] \geq 1-\delta$$

Problem 5

Input

Array $A[1..n]$ of n distinct integers, parameter ϵ.

Output

If A is sorted, returns YES.

If A is ϵ-far from being sorted, returns NO with probability at least 3/4.

Otherwise return either YES or NO.

Algorithm

1. Repeat k times:
    - Pick $x \in A$ uniformly at random
    - Search for x with binary search.
    - If search failed: return NO
2. Return YES

Where $k = \frac{2}{\epsilon}$

Run time

Binary search on an array with length n has run time $\mathcal{O}(\log n)$. Thus, picking x and checking whether the search failed can be done in $\mathcal{O}(1)$. Thus one repetition runs in $\mathcal{O}(\log n)$.

Repeting $k = \frac{2}{\epsilon}$ times leads to a total run time of $\mathcal{O}\left(\frac{\log n}{\epsilon}\right)$.

Correctness

- Case 1: A is sorted

  Finding $x \in A$ with binary search is always successful when A is sorted. Thus the algorithm outputs YES.

- Case 2: A is ϵ-far from being sorted

  To show: Algorithm returns NO with probability at least 3/4.

Analyzing one run of the loop body:

Lets $s_1, s_2, \ldots, s_m$ be all the elements in A that can be found by binary search.

Claim: $\forall i, j \in \{1, \ldots, m\} : i < j \Rightarrow s_i < s_j$

Proof: Assume $i < j$.

By the way binary search works: There must exist an index l where the binary search diverged. I.e. $\exists l : i < l$ and $l \leq j$. Since the search diverged at l: $s_i < s_l$ and $s_l \leq s_j$. By transitivity: $s_i < s_j$.

Thus, $s_1, s_2, \ldots, s_m$ is a sorted sequence. The longest sorted sequence has length $(1 - \epsilon) \cdot n$. Therefore: $m \leq (1 - \epsilon) \cdot n$.

Drawing $x \in A$ uniformly at random: x can only be found if $x = s_i$ for some $i \in \{1, \ldots, m\}$.

Therefore, the probability p of drawing a x which can be found is at most $\frac{m}{n} = \frac{(1-\epsilon)\cdot n}{n} = 1 - \epsilon$

$\Rightarrow p \leq (1 - \epsilon)$

Repeating the body of the loop k times:

$\Pr[YES] = p^k = (1 - \epsilon)^k \leq e^{-\epsilon \cdot k} = e^{-\epsilon \cdot \frac{2}{\epsilon}} = e^{-2} = e^{\log\left(\frac{1}{4}\right)} = \frac{1}{4}$

$\Pr[NO] = 1 - \Pr[YES] \geq 1 - \frac{1}{4} = \frac{3}{4}$

Therefore, the algorithm yields NO with probability at least 3/4.

- Case 3: A is neither sorted nor ϵ far from being sorted

  The algorithm terminates on every input. YES and NO are the only possible outputs. Thus, the algorithm always outputs either YES or NO.