Last time
• Poisson process
• Use of probability in algorithms (sorting uniform input in linear time)

Today
• Use of probability in data structures (hashing and Bloom filters)
• Sublinear-time algorithms

Reminder
• HW 12 is due today
Static dictionary problem

Motivating example
Password checker to prevent people from using common passwords.

- $S$ is the set of common passwords

- **Universe**: set $U$
- $S \subseteq U$ and $m = |S|$
- $m \ll |U|$

**Goal**: A data structure for storing $S$ that supports the search query

“Does $w \in S$?” for all words $w \in U$. 
Deterministic solutions

- Store $S$ as a sorted array (or as a binary search tree)

**Search time:** $O(\log m)$, **Space:** $O(m)$

- Store an array that for each $w \in U$ has 1 if $w \in S$ and 0 otherwise.

**Search time:** $O(1)$, **Space:** $O(|U|)$

A randomized solution

- Hashing
Hashing with Chaining

- **Hash table:** \( n \) bins, words that fall in the same bin are chained into a linked list.
- **Hash function:** \( h : U \rightarrow \{1, \ldots, n\} \)

To construct the table
- hash all elements of \( S \)

To search for word \( w \)
- check if \( w \) is in bin \( h(w) \)

Elements of \( S \)
A random hash function

- **Simplifying assumption:** hash function $h$ is selected at random:
  \[
  \Pr[h(w) = j] = \frac{1}{n} \quad \text{for all } w \in U \text{ and } j \in \{1, \ldots, n\}
  \]

- Once $h$ is chosen, every evaluation of $h$ yields the same answer.

**Search time:**
- If $w \notin S$, expected number of words in bin $h(w)$ is
- If $w \in S$, expected number of words in bin $h(w)$ is

If we set $n = m$, then
- the expected search time is $\mathcal{O}(1)$
- max time to search is max load: w.p. close to 1, it is $\Theta\left(\frac{\ln m}{\ln \ln m}\right)$

Faster than a search tree, with space still $\Theta(m)$. 
Approximate solution

for static dictionary problem

- **False positives:** If $w \in S$, our data structure must answer correctly. If $w \notin S$, we may err with small probability.
- E.g., we prevent all unsuitable passwords and some suitable ones, too.

- **Bloom filters**
  - give trade off between space and false positive probability
  - have parameters $k, n$
Bloom filter with \( n \) bits and \( k \) hash functions

- Bloom filter: array of \( n \) bits \( A[1], \ldots, A[n] \)
  - Initially: all bits are 0
    
    | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
    -------------------------------------
    | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | ... | n |

  - \( k \) independent random hash functions \( h_1, \ldots, h_k \) with range \( \{1, \ldots, n\} \)

- To represent set \( S \)
  - For each \( x \in S \) and \( i \in \{1, \ldots, k\} \), set bits \( A[h_i(x)] \) to 1.

- To decide if \( w \in S \):
  - If for all \( i \in \{1, \ldots, k\} \), bits \( A[h_i(w)] = 1 \), accept, o.w. reject.
Analysis of False Positive rate

• For any $n$, we can set $k \approx \frac{n}{m} \ln 2$ .
• Consider $w \in U - S$.
• Let $B_i = A[h_i(w)]$ for all $i \in \{1, ..., k\}$
• After $m$ elements hashed into Bloom filter, $\Pr[B_i = 0] =$
A Sublinear-Time Algorithm

randomized algorithm

approximate answer

Quality of approximation

Resources
- number of queries
- running time


? B

? L

? L

? A
Randomized algorithms; property testers

Randomized Algorithm

YES

Accept with probability $\geq \frac{2}{3}$

NO

Reject with probability $\geq \frac{2}{3}$

Property Tester

YES

Accept with probability $\geq \frac{2}{3}$

Far from YES

$\varepsilon$

Don’t care

Reject with probability $\geq \frac{2}{3}$

$\varepsilon$ - far = differs in many places ($\geq \varepsilon$ fraction of places)
Example: Testing if a List is Sorted

**Input:** a list of $n$ numbers $x_1, x_2, \ldots, x_n$

- A list of numbers is **sorted** if $x_1 \leq x_2 \leq \cdots \leq x_n$.

- **Question:** Is the list sorted?

  Requires reading entire list: $\Omega(n)$ time

- **Approximate version:** Is the list sorted or $\varepsilon$-far from sorted?
  (An $\varepsilon$ fraction of $x_i$’s have to be changed to make it sorted.)

  $O\left(\frac{\log n}{\varepsilon}\right)$ time
Sortedness Testing: Attempts

1. **Test**: Pick a uniformly random \( i \in \{1, \ldots, n - 1\} \) and reject if \( x_i > x_{i+1} \).
   
   Fails on: \[ 1 \, 1 \, 1 \, 1 \, 0 \, 0 \, 0 \, 0 \] ← 1/2-far from sorted

2. **Test**: Pick uniformly random \( i < j \) in \( \{1, \ldots, n\} \) and reject if \( x_i > x_j \).
   
   Fails on: \[ 1 \, 0 \, 2 \, 1 \, 3 \, 2 \, 4 \, 3 \, 5 \, 4 \] ← 1/2-far from sorted
Is a list Sorted or $\varepsilon$-far from Sorted?

Idea: Associate positions in the list with vertices of the directed line.

Construct a graph (2-spanner)

- by adding a few “shortcut” edges $(i, j)$ for $i < j$
- where each pair of vertices is connected by a path of length at most 2

$\leq n \log n$ edges
Is a list Sorted or $\varepsilon$-far from Sorted?

Test

Pick a random edge $(i, j)$ from the 2-spanner and reject if $x_i > x_j$.

Analysis:

- Call an edge $(i, j)$ violated if $x_i > x_j$, and satisfied otherwise.
- If $i$ is an endpoint of a violated edge, call $x_i$ bad. Otherwise, call it good.

Claim 1. All good numbers are sorted.

Proof: Consider any two good numbers, $x_i$ and $x_j$.

They are connected by a path of (at most) two satisfied edges $(i, k), (k, j)$

$\Rightarrow x_i \leq x_k$ and $x_k \leq x_j$

$\Rightarrow x_i \leq x_j$
Is a list Sorted or $\varepsilon$-far from Sorted?

**Test**

Pick a random edge $(i, j)$ from the 2-spanner and reject if $x_i > x_j$.  

**Analysis:**

- Call an edge $(i, j)$ **violated** if $x_i > x_j$, and **satisfied** otherwise.
- If $i$ is an endpoint of a **violated** edge, call $x_i$ **bad**. Otherwise, call it **good**.

**Claim 1.** All **good** numbers are sorted.

**Claim 2.** An $\varepsilon$-far list violates $\geq \frac{\varepsilon}{2 \log n}$ fraction of edges in 2-spanner.

**Proof:** If a list is $\varepsilon$-far from sorted, it has $\geq \varepsilon n$ **bad** numbers. (Claim 1)

- Each **violated** edge contributes 2 **bad** numbers.
- 2-spanner has $\geq \frac{\varepsilon n}{2}$ **violated** edges out of $n \log n$. 

4/27/2023

Tiago Januario, Sofya Raskhodnikova; Probability in Computing
Is a list Sorted or $\varepsilon$-far from Sorted?

**Test**

Pick a random edge $(i, j)$ from the 2-spanner and **reject** if $x_i > x_j$.

**Analysis:**

- Call an edge $(i, j)$ **violated** if $x_i > x_j$, and **satisfied** otherwise.

**Claim 2.** An $\varepsilon$-far list violates $\geq \varepsilon/(2 \log n)$ fraction of edges in 2-spanner.

**Algorithm**

Sample $\frac{4\log n}{\varepsilon}$ edges $(i, j)$ from the 2-spanner and **reject** if $x_i > x_j$.

**Guarantee:** All sorted lists are accepted.

All lists that are $\varepsilon$-far from sorted are rejected with probability $\geq 2/3$. 
We can determine if a list of \( n \) numbers is sorted or \( \varepsilon \)-far from sorted in \( O\left(\frac{\log n}{\varepsilon}\right) \) time.
Sublinear Algorithms: Summary

- Many problems admit sublinear-time algorithms
- Algorithms are often simple
- Analysis requires creation of interesting combinatorial, geometric and algebraic tools