CS 577- Intro to Algorithms

Randomness (Part 2)

Dieter van Melkebeek

December 10, 2020

Outline

Outline

Idea

Outline

Idea

Make a random choice when there are many good choices but it is hard to deterministically find one.

Outline

Idea

Make a random choice when there are many good choices but it is hard to deterministically find one.

Benefits

Outline

Idea

Make a random choice when there are many good choices but it is hard to deterministically find one.

Benefits

- Simpler algorithms

Outline

Idea

Make a random choice when there are many good choices but it is hard to deterministically find one.

Benefits

- Simpler algorithms
- More time and/or space efficient algorithms

Outline

Idea

Make a random choice when there are many good choices but it is hard to deterministically find one.

Benefits

- Simpler algorithms
- More time and/or space efficient algorithms

Applications

Outline

Idea

Make a random choice when there are many good choices but it is hard to deterministically find one.

Benefits

- Simpler algorithms
- More time and/or space efficient algorithms

Applications

- Selection

Outline

Idea

Make a random choice when there are many good choices but it is hard to deterministically find one.

Benefits

- Simpler algorithms
- More time and/or space efficient algorithms

Applications

- Selection
- Sorting

Outline

Idea

Make a random choice when there are many good choices but it is hard to deterministically find one.

Benefits

- Simpler algorithms
- More time and/or space efficient algorithms

Applications

- Selection
- Sorting
- Dynamic dictionary problem

Outline

Idea

Make a random choice when there are many good choices but it is hard to deterministically find one.

Benefits

- Simpler algorithms
- More time and/or space efficient algorithms

Applications

- Selection
- Sorting
- Dynamic dictionary problem: hashing

Outline

Idea

Make a random choice when there are many good choices but it is hard to deterministically find one.

Benefits

- Simpler algorithms
- More time and/or space efficient algorithms

Applications

- Selection
- Sorting
- Dynamic dictionary problem: hashing
- Finding a closest pair of points in the plane

Outline

Idea

Make a random choice when there are many good choices but it is hard to deterministically find one.

Benefits

- Simpler algorithms
- More time and/or space efficient algorithms

Applications

- Selection
- Sorting
- Dynamic dictionary problem: hashing
- Finding a closest pair of points in the plane
- Polynomial identity testing

Dynamic Dictionary Problem

Dynamic Dictionary Problem

Goal

Dynamic Dictionary Problem

Goal

- Maintain a small subset S of a huge universe U.

Dynamic Dictionary Problem

Goal

- Maintain a small subset S of a huge universe U.
- Supported operations:

Dynamic Dictionary Problem

Goal

- Maintain a small subset S of a huge universe U.
- Supported operations:
  - Insertion: adding an element to S

Dynamic Dictionary Problem

Goal

- Maintain a small subset S of a huge universe U.
- Supported operations:
  - Insertion: adding an element to S
  - Deletion: removing an element from S

Dynamic Dictionary Problem

Goal

- Maintain a small subset S of a huge universe U.
- Supported operations:
  - Insertion: adding an element to S
  - Deletion: removing an element from S
  - Lookup: checking whether an element belongs to S (and retrieving information about that element)

Dynamic Dictionary Problem

Goal

- Maintain a small subset S of a huge universe U.
- Supported operations:
  - Insertion: adding an element to S
  - Deletion: removing an element from S
  - Lookup: checking whether an element belongs to S (and retrieving information about that element)

Realizations

Dynamic Dictionary Problem

Goal

- Maintain a small subset S of a huge universe U.
- Supported operations:
  - Insertion: adding an element to S
  - Deletion: removing an element from S
  - Lookup: checking whether an element belongs to S (and retrieving information about that element)

Realizations

- Characteristic vector

Dynamic Dictionary Problem

Goal

- Maintain a small subset S of a huge universe U.
- Supported operations:
  - Insertion: adding an element to S
  - Deletion: removing an element from S
  - Lookup: checking whether an element belongs to S (and retrieving information about that element)

Realizations

- Characteristic vector: time $O(1)$, space $O(|U|)$

Dynamic Dictionary Problem

Goal

- Maintain a small subset S of a huge universe U.
- Supported operations:
  - Insertion: adding an element to S
  - Deletion: removing an element from S
  - Lookup: checking whether an element belongs to S (and retrieving information about that element)

Realizations

- Characteristic vector: time $O(1)$, space $O(|U|)$
- Balanced search tree

Dynamic Dictionary Problem

Goal

- Maintain a small subset S of a huge universe U.
- Supported operations:
  - Insertion: adding an element to S
  - Deletion: removing an element from S
  - Lookup: checking whether an element belongs to S (and retrieving information about that element)

Realizations

- Characteristic vector: time $O(1)$, space $O(|U|)$
- Balanced search tree: time $O(\log |S|)$, space $O(|S|)$

Dynamic Dictionary Problem

Goal

- Maintain a small subset S of a huge universe U.
- Supported operations:
  - Insertion: adding an element to S
  - Deletion: removing an element from S
  - Lookup: checking whether an element belongs to S (and retrieving information about that element)

Realizations

- Characteristic vector: time $O(1)$, space $O(|U|)$
- Balanced search tree: time $O(\log |S|)$, space $O(|S|)$
- Hash table

Dynamic Dictionary Problem

Goal

- Maintain a small subset S of a huge universe U.
- Supported operations:
  - Insertion: adding an element to S
  - Deletion: removing an element from S
  - Lookup: checking whether an element belongs to S (and retrieving information about that element)

Realizations

- Characteristic vector: time $O(1)$, space $O(|U|)$
- Balanced search tree: time $O(\log |S|)$, space $O(|S|)$
- Hash table: expected time $O(1)$, space $O(|S|)$

Hashing

Hashing

Idea

Hashing

Idea

- Pick a random function $h : U \to [m]$ for $m = \Theta(|S|)$.

Hashing

Idea

- Pick a random function $h : U \to [m]$ for $m = \Theta(|S|)$.
- Store each $x \in S$ in cell $h(x)$ of a table of size m.

Hashing

Idea

- Pick a random function $h : U \to [m]$ for $m = \Theta(|S|)$.
- Store each $x \in S$ in cell $h(x)$ of a table of size m.

Issues

Hashing

Idea

- Pick a random function $h : U \to [m]$ for $m = \Theta(|S|)$.
- Store each $x \in S$ in cell $h(x)$ of a table of size m.

Issues

- Succinct description of h

Hashing

Idea

- Pick a random function $h : U \to [m]$ for $m = \Theta(|S|)$.
- Store each $x \in S$ in cell $h(x)$ of a table of size m.

Issues

- Succinct description of h
  - Small universal families of hash functions (pairwise independence)

Hashing

Idea

- Pick a random function $h : U \to [m]$ for $m = \Theta(|S|)$.
- Store each $x \in S$ in cell $h(x)$ of a table of size m.

Issues

- Succinct description of h
  - Small universal families of hash functions (pairwise independence)
  - Example: $h(x) = ax + b \bmod m$ for m prime

Hashing

Idea

- Pick a random function $h : U \to [m]$ for $m = \Theta(|S|)$.
- Store each $x \in S$ in cell $h(x)$ of a table of size m.

Issues

- Succinct description of h
  - Small universal families of hash functions (pairwise independence)
  - Example: $h(x) = ax + b \bmod m$ for m prime
- Collision resolution

Hashing

Idea

- Pick a random function $h : U \to [m]$ for $m = \Theta(|S|)$.
- Store each $x \in S$ in cell $h(x)$ of a table of size m.

Issues

- Succinct description of h
  - Small universal families of hash functions (pairwise independence)
  - Example: $h(x) = ax + b \bmod m$ for m prime
- Collision resolution
  - Chaining

Hashing

Idea

- Pick a random function $h : U \to [m]$ for $m = \Theta(|S|)$.
- Store each $x \in S$ in cell $h(x)$ of a table of size m.

Issues

- Succinct description of h
  - Small universal families of hash functions (pairwise independence)
  - Example: $h(x) = ax + b \bmod m$ for m prime
- Collision resolution
  - Chaining
  - Open addressing (linear, quadratic, double)

Hash Collisions

Hash Collisions

Definition

Two distinct elements $x, x' \in S$ with $h(x) = h(x')$.

Hash Collisions

Definition

Two distinct elements $x, x' \in S$ with $h(x) = h(x')$.

Fact

For any two distinct $x, x' \in U$,

$$\Pr[h(x) = h(x')] = \frac{1}{m}$$

when $h : U \to [m]$ is picked uniformly at random.

Hash Collisions

Definition

Two distinct elements $x, x' \in S$ with $h(x) = h(x')$.

Fact

For any two distinct $x, x' \in U$,

$$\Pr[h(x) = h(x')] = \frac{1}{m}$$

when $h : U \to [m]$ is picked uniformly at random.

Corollary

For any fixed $x \in S$, the expected number of collisions with x equals $\frac{|S|-1}{m}$.

Closest Pair of Points in the Plane

Closest Pair of Points in the Plane

Problem

Input: $p_i \doteq (x_i, y_i) \in \mathbb{R}^2$ for $i \in [n]$

Closest Pair of Points in the Plane

Problem

Input: $p_i \doteq (x_i, y_i) \in \mathbb{R}^2$ for $i \in [n]$

Output: $\delta \doteq \min\{d(p_i, p_j) : i, j \in [n] \text{ with } i \neq j\}$ where
$d(p_i, p_j) \doteq \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2}$

Closest Pair of Points in the Plane

Problem

Input: $p_i \doteq (x_i, y_i) \in \mathbb{R}^2$ for $i \in [n]$

Output: $\delta \doteq \min\{d(p_i, p_j) : i, j \in [n] \text{ with } i \neq j\}$ where
$d(p_i, p_j) \doteq \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2}$

Algorithms

Closest Pair of Points in the Plane

Problem

Input: $p_i \doteq (x_i, y_i) \in \mathbb{R}^2$ for $i \in [n]$

Output: $\delta \doteq \min\{d(p_i, p_j) : i, j \in [n] \text{ with } i \neq j\}$ where
$d(p_i, p_j) \doteq \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2}$

Algorithms

- Trivial: $O(n^2)$

Closest Pair of Points in the Plane

Problem

Input: $p_i \doteq (x_i, y_i) \in \mathbb{R}^2$ for $i \in [n]$

Output: $\delta \doteq \min\{d(p_i, p_j) : i, j \in [n] \text{ with } i \neq j\}$ where
$d(p_i, p_j) \doteq \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2}$

Algorithms

- Trivial: $O(n^2)$
- Divide & Conquer: $O(n \log n)$

Closest Pair of Points in the Plane

Problem

Input: $p_i \doteq (x_i, y_i) \in \mathbb{R}^2$ for $i \in [n]$

Output: $\delta \doteq \min\{d(p_i, p_j) : i, j \in [n] \text{ with } i \neq j\}$ where
$d(p_i, p_j) \doteq \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2}$

Algorithms

- Trivial: $O(n^2)$
- Divide & Conquer: $O(n \log n)$
- Randomized: $O(n)$ in expectation

Closest Pair of Points in the Plane – approach

Closest Pair of Points in the Plane – approach

- Process points $p_1, p_2, \ldots, p_n$ one by one.

Closest Pair of Points in the Plane – approach

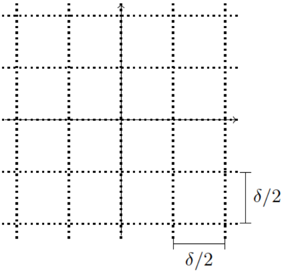
- Process points $p_1, p_2, \ldots, p_n$ one by one.
- Maintain shortest pairwise distance δ among processed points.

Closest Pair of Points in the Plane – approach

- Process points $p_1, p_2, \ldots, p_n$ one by one.
- Maintain shortest pairwise distance δ among processed points.
- To process new point p_i, find all processed points p that are within δ from p_i, and update δ to $\min(\delta, \min_p(d(p_i, p)))$.

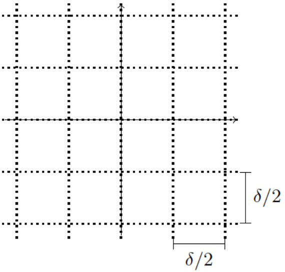
Closest Pair of Points in the Plane – approach

- Process points $p_1, p_2, \ldots, p_n$ one by one.
- Maintain shortest pairwise distance δ among processed points.
- To process new point p_i, find all processed points p that are within δ from p_i, and update δ to $\min(\delta, \min_p(d(p_i, p)))$.

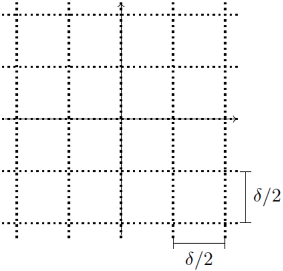


Closest Pair of Points in the Plane – dictionary

Closest Pair of Points in the Plane – dictionary

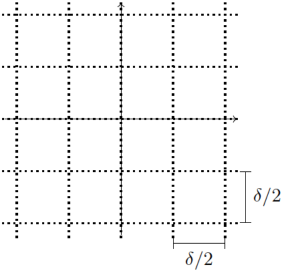


Closest Pair of Points in the Plane – dictionary



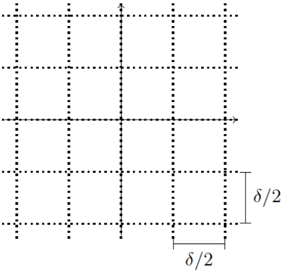
- Dictionary of cells in grid with side $\delta/2$ that contain a processed point.

Closest Pair of Points in the Plane – dictionary



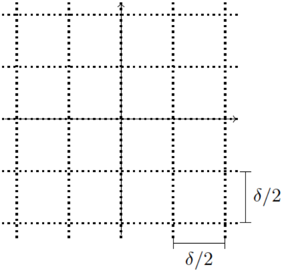
- Dictionary of cells in grid with side $\delta/2$ that contain a processed point.
- Uses hash function $h : U \to [m]$ where U is the set of all cells.

Closest Pair of Points in the Plane – dictionary



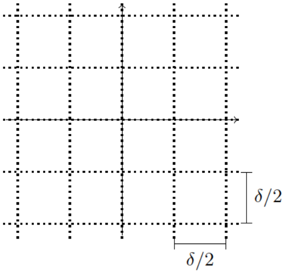
- Dictionary of cells in grid with side $\delta/2$ that contain a processed point.
- Uses hash function $h : U \to [m]$ where U is the set of all cells.
- To process a new point p_i:

Closest Pair of Points in the Plane – dictionary



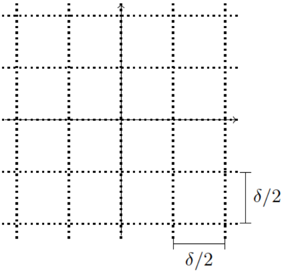
- Dictionary of cells in grid with side $\delta/2$ that contain a processed point.
- Uses hash function $h : U \to [m]$ where U is the set of all cells.
- To process a new point p_i:
  - Insert cell of p_i in the dictionary.

Closest Pair of Points in the Plane – dictionary



- Dictionary of cells in grid with side $\delta/2$ that contain a processed point.
- Uses hash function $h : U \to [m]$ where U is the set of all cells.
- To process a new point p_i:
  - Insert cell of p_i in the dictionary.
  - Check each of the neighboring cells for a processed point that is closer to p_i than δ.

Closest Pair of Points in the Plane – dictionary



- Dictionary of cells in grid with side $\delta/2$ that contain a processed point.
- Uses hash function $h : U \to [m]$ where U is the set of all cells.
- To process a new point p_i:
  - Insert cell of p_i in the dictionary.
  - Check each of the neighboring cells for a processed point that is closer to p_i than δ.
  - If so, update δ and rehash.

Closest Pair of Points in the Plane - running time

Closest Pair of Points in the Plane - running time

Comparisons, insertions, and lookups

Closest Pair of Points in the Plane - running time

Comparisons, insertions, and lookups

- Expected number of points that are checked in each step is $O(1)$ provided $m = \Theta(n)$.

Closest Pair of Points in the Plane - running time

Comparisons, insertions, and lookups

- Expected number of points that are checked in each step is $O(1)$ provided $m = \Theta(n)$.
- $O(n)$ in total.

Closest Pair of Points in the Plane - running time

Comparisons, insertions, and lookups

- Expected number of points that are checked in each step is $O(1)$ provided $m = \Theta(n)$.
- $O(n)$ in total.

Rehashing

Closest Pair of Points in the Plane - running time

Comparisons, insertions, and lookups

- Expected number of points that are checked in each step is $O(1)$ provided $m = \Theta(n)$.
- $O(n)$ in total.

Rehashing

- Takes time $O(i)$ if needed in step i.

Closest Pair of Points in the Plane - running time

Comparisons, insertions, and lookups

- Expected number of points that are checked in each step is $O(1)$ provided $m = \Theta(n)$.
- $O(n)$ in total.

Rehashing

- Takes time $O(i)$ if needed in step i.
- Rehash in i-th step only needed if i-th point is involved in closest pair among the first i points.

Closest Pair of Points in the Plane - running time

Comparisons, insertions, and lookups

- Expected number of points that are checked in each step is $O(1)$ provided $m = \Theta(n)$.
- $O(n)$ in total.

Rehashing

- Takes time $O(i)$ if needed in step i.
- Rehash in i-th step only needed if i-th point is involved in closest pair among the first i points.
- If we consider points in random order, i-th point is involve in closest pair among first i points with probability at most $2/i$.

Closest Pair of Points in the Plane - running time

Comparisons, insertions, and lookups

- Expected number of points that are checked in each step is $O(1)$ provided $m = \Theta(n)$.
- $O(n)$ in total.

Rehashing

- Takes time $O(i)$ if needed in step i.
- Rehash in i-th step only needed if i-th point is involved in closest pair among the first i points.
- If we consider points in random order, i-th point is involve in closest pair among first i points with probability at most $2/i$.
- Total expected time for rehashing then becomes $O(\sum_{i=1}^{n} \frac{2}{i} \cdot i) = O(n)$.

Polynomial Identity Testing

Polynomial Identity Testing

Problem

Polynomial Identity Testing

Problem

Input: Multivariate polynomials p_1 and p_2

Polynomial Identity Testing

Problem

Input: Multivariate polynomials p_1 and p_2

Output: Is $p_1 \equiv p_2$?

Polynomial Identity Testing

Problem

Input: Multivariate polynomials p_1 and p_2

Output: Is $p_1 \equiv p_2$?

Examples

- $p_1 = (x_1 + x_2) \cdot (x_1 - x_2)$
- $p_2 = x_1 \cdot x_1 - x_2 \cdot x_2$

Polynomial Identity Testing

Problem

Input: Multivariate polynomials p_1 and p_2

Output: Is $p_1 \equiv p_2$?

Examples

- $p_1 = (x_1 + x_2) \cdot (x_1 - x_2)$
- $p_2 = x_1 \cdot x_1 - x_2 \cdot x_2$
- $p_3 = (x_1 + x_2) \cdot (x_3 + x_4) \cdot \dots \cdot (x_{2n-1} + x_{2n})$

Polynomial Identity Testing – algorithms

Polynomial Identity Testing – algorithms

Deterministic

- All known algorithms for general case run in exponential time.

Polynomial Identity Testing – algorithms

Deterministic

- All known algorithms for general case run in exponential time.

Randomized

Polynomial Identity Testing – algorithms

Deterministic

- All known algorithms for general case run in exponential time.

Randomized

- Try random assignment:

Polynomial Identity Testing – algorithms

Deterministic

- All known algorithms for general case run in exponential time.

Randomized

- Try random assignment:
  1. Pick $x_1^*, x_2^*, \ldots, x_n^* \in [m]$ uniformly at random.
  2. Output whether $p_1(x_1^*, \ldots, x_n^*) = p_2(x_1^*, \ldots, x_n^*)$

Polynomial Identity Testing – algorithms

Deterministic

- All known algorithms for general case run in exponential time.

Randomized

- Try random assignment:
  1. Pick $x_1^*, x_2^*, \ldots, x_n^* \in [m]$ uniformly at random.
  2. Output whether $p_1(x_1^*, \ldots, x_n^*) = p_2(x_1^*, \ldots, x_n^*)$
- Probability of false positive is at most d/m, where $d = \max(\deg(p_1), \deg(p_2))$.

Polynomial Identity Testing – algorithms

Deterministic

- All known algorithms for general case run in exponential time.

Randomized

- Try random assignment:
  1. Pick $x_1^*, x_2^*, \ldots, x_n^* \in [m]$ uniformly at random.
  2. Output whether $p_1(x_1^*, \ldots, x_n^*) = p_2(x_1^*, \ldots, x_n^*)$
- Probability of false positive is at most d/m, where $d = \max(\deg(p_1), \deg(p_2))$.
- Yields polynomial-time algorithm.