9 Random Tricks

25 June 2025

Prof. Dr. Sebastian Wild

CS627 (Summer 2025)
Philipps-Universität Marburg

version `2025-07-02 10:00`

Outline

9 Random Tricks

Uses of Randomness

- Since it is likely that $\mathsf{BPP} = \mathsf{P}$, we focus on the more fine-grained benefits of randomization:
  - simpler algorithms (with same performance)
  - improving performance (but not jumping from exponential to polytime)
  - improved robustness
- Here: Collection of examples illustrating different techniques
  - fingerprinting / hashing
  - exploiting abundance of witnesses
  - random sampling

9.1 Hashing – Balls Into Bins

Fingerprinting / Hashing

- Often have elements from huge universe $U = [0..u)$ of possible values, but only deal with few actual items $x_1, \ldots, x_n$ at one time. $\in \mathcal{U}$
  Think: $n \ll u$
- Fingerprinting can help to be more efficient in this case
  - fingerprints from $[0..m)$
  - $m \ll u$
  - *Hash Function* $h : U \to [0..m)$

h will have collisions

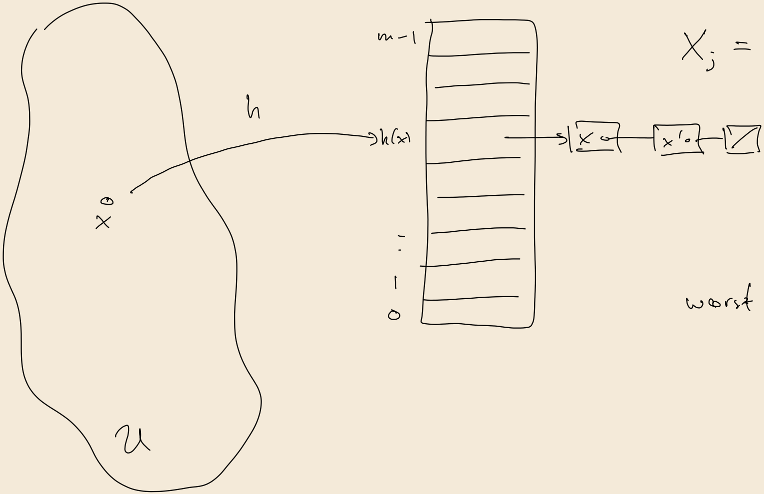
$(x, y \in \mathcal{U} : h(x) = h(y))$

Fingerprinting / Hashing

- Often have elements from huge universe $U = [0..u)$ of possible values, but only deal with few actual items $x_1, \ldots, x_n$ at one time. Think: $n \ll u$
- Fingerprinting can help to be more efficient in this case
  - fingerprints from $[0..m)$
  - $m \ll u$
  - *Hash Function* $h : U \to [0..m)$
- Classic Example: hash tables and Bloom filters

Hash Tables

indirect chaining



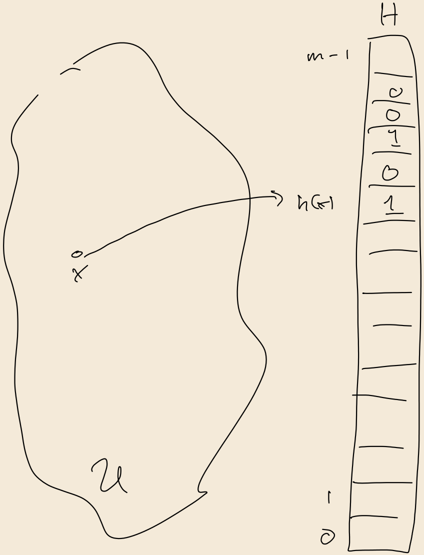
Performance:

How big are buckets?

$X_j = \#$ keys x with $h(x) = j$ in our HT

worst case $X_1 = n$

Bloom Filters



$insert(x)$: $H[h(x)] := 1$

$query(x)$: $H[h(x)]$

└ output 1 (Yes) can be a <u>false positive</u>!

output 0 (No) correct

(reduce false positive rate using independent $h_1, h_2, h_3, h_4 \ldots$)

application: segmented database "cheap first check"

Uniform – Universal – Perfect

Randomness is essential for hashing to make any sense! Three very different uses

1. *uniform hashing assumption*: (optimistic, often roughly right in practice!)
   How good is hashing if input is "as nicely random" as possible?

Uniform hashing assumption:

All m^n possible hash seq. $h(x_1), h(x_2), h(x_3), \ldots, h(x_n)$ are equally likely.

Uniform – Universal – Perfect

Randomness is essential for hashing to make any sense! Three very different uses

1. *uniform hashing assumption*: (optimistic, often roughly right in practice!)
   How good is hashing if input is "as nicely random" as possible?

2. Since fixed h is prone to "algorithmic complexity attacks" (worst case inputs)
   $\rightsquigarrow$ *universal hashing*: pick h at random from class H of suitable functions

   universal class of hash functions

Uniform – Universal – Perfect

Randomness is essential for hashing to make any sense! Three very different uses

1. *uniform hashing assumption*: (optimistic, often roughly right in practice!)
   How good is hashing if input is "as nicely random" as possible?

2. Since fixed h is prone to "algorithmic complexity attacks" (worst case inputs)
   $\rightsquigarrow$ *universal hashing*: pick h at random from class H of suitable functions

   universal class of hash functions

3. For given keys, can construct collision-free hash function
   $\rightsquigarrow$ *perfect hashing*

Uniform Hashing – Balls into Bins

Uniform Hashing Assumption:
When n elements $x_1, \ldots, x_n$ are inserted, for their *hash sequence* $h(x_1), \ldots, h(x_n)$, all m^n possible values are **equally likely**.

$\rightsquigarrow$ behavior of data structure completely **independent** of $x_1, \ldots, x_n$!

Uniform Hashing – Balls into Bins

Uniform Hashing Assumption:
When n elements $x_1, \ldots, x_n$ are inserted, for their *hash sequence* $h(x_1), \ldots, h(x_n)$, all m^n possible values are **equally likely**.

$\rightsquigarrow$ behavior of data structure completely **independent** of $x_1, \ldots, x_n$!

$\rightsquigarrow$ might as well forget data!

Balls into bins model (a.k.a. balanced allocations)

- throw n balls into m bins

⚠ Literature usually swaps n and m!

- each ball picks bin *i.i.d. uniformly* at random

B_i = bin of ith ball $\overset{\mathcal{D}}{=} \mathcal{U}([0..m))$

- classic abstract model to study randomized algorithms
  - For hashing, effectively the best imaginable case tends to be a bit optimistic!
  - but: data in applications often not far from this

A Paradox?

- X_j: Number of balls in bin j:

$\rightsquigarrow$ $X_1 \stackrel{\mathcal{D}}{=} \cdots \stackrel{\mathcal{D}}{=} X_m \stackrel{\mathcal{D}}{=} \mathrm{Bin}(n, \frac{1}{m})$

$\rightsquigarrow$ All X_j concentrated around expectation $\frac{n}{m}$ (Chernoff!)

A Paradox?

- X_i: Number of balls in bin i:

$\rightsquigarrow$ $X_1 \stackrel{\mathcal{D}}{=} \cdots \stackrel{\mathcal{D}}{=} X_m \stackrel{\mathcal{D}}{=} \mathrm{Bin}(n, \frac{1}{m})$

$\rightsquigarrow$ All X_i concentrated around expectation $\frac{n}{m}$ (Chernoff!)

Consider $\boxed{m = n}$ $\rightsquigarrow$ $\mathbb{E}[X_i] = 1$

A Paradox?

- X_i: Number of balls in bin i:

$\rightsquigarrow$ $X_1 \overset{\mathcal{D}}{=} \cdots \overset{\mathcal{D}}{=} X_m \overset{\mathcal{D}}{=} \mathrm{Bin}(n, \frac{1}{m})$

$\rightsquigarrow$ All X_i concentrated around expectation $\frac{n}{m}$ (Chernoff!)

Consider $\boxed{m = n}$ $\rightsquigarrow$ $\mathbb{E}[X_i] = 1$

- But also: expected number of *empty* bins:

$$\mathbb{E}[\#i \text{ with } X_i = 0] \;=\; \sum_{i=1}^{m} \mathbb{P}[X_i = 0]$$

A Paradox?

- X_i: Number of balls in bin i:

$\rightsquigarrow$ $X_1 \stackrel{\mathcal{D}}{=} \cdots \stackrel{\mathcal{D}}{=} X_m \stackrel{\mathcal{D}}{=} \mathrm{Bin}(n, \frac{1}{m})$

$\rightsquigarrow$ All X_i concentrated around expectation $\frac{n}{m}$ (Chernoff!)

Consider $\boxed{m = n}$ $\rightsquigarrow$ $\mathbb{E}[X_i] = 1$

- But also: expected number of *empty* bins:

$$\begin{aligned}\mathbb{E}[\#i \text{ with } X_i = 0] &= \sum_{i=1}^{m} \mathbb{P}[X_i = 0] \\ &= m \cdot \left(1 - \frac{1}{m}\right)^n \qquad (m = n,\ \underline{(1 + 1/n)^n \approx e})\end{aligned}$$

A Paradox?

- X_i: Number of balls in bin i:

$\rightsquigarrow$ $X_1 \stackrel{\mathcal{D}}{=} \cdots \stackrel{\mathcal{D}}{=} X_m \stackrel{\mathcal{D}}{=} \mathrm{Bin}(n, \frac{1}{m})$

$\rightsquigarrow$ All X_i concentrated around expectation $\frac{n}{m}$ (Chernoff!)

Consider $\boxed{m = n}$ $\rightsquigarrow$ $\mathbb{E}[X_i] = 1$

- But also: expected number of *empty* bins:

$$\begin{aligned}
\mathbb{E}[\#i \text{ with } X_i = 0] &= \sum_{i=1}^{m} \mathbb{P}[X_i = 0] \\
&= m \cdot \left(1 - \frac{1}{m}\right)^n \qquad (m = n,\ (1 + 1/n)^n \approx e) \\
&= n \cdot e(1 \pm O(n^{-1}))
\end{aligned}$$

A Paradox?

- X_i: Number of balls in bin i:

$\rightsquigarrow$ $X_1 \overset{\mathcal{D}}{=} \cdots \overset{\mathcal{D}}{=} X_m \overset{\mathcal{D}}{=} \mathrm{Bin}(n, \frac{1}{m})$

$\rightsquigarrow$ All X_i concentrated around expectation $\frac{n}{m}$ (Chernoff!)

Consider $\boxed{m = n}$ $\quad\rightsquigarrow\quad$ $\mathbb{E}[X_i] = 1$

- But also: expected number of *empty* bins:

$$\begin{aligned} \mathbb{E}[\#i \text{ with } X_i = 0] &= \sum_{i=1}^{m} \mathbb{P}[X_i = 0] \\ &= m \cdot \left(1 - \frac{1}{m}\right)^n \qquad (m = n,\ (1 + 1/n)^n \approx e) \\ &= n \cdot \frac{1}{e}(1 \pm O(n^{-1})) \end{aligned}$$

$\rightsquigarrow$ In expectation, $\frac{1}{e}$ fraction (37%) of bins empty!

How does that fit together with $\mathbb{E}[X_i] = 1$? Which expectation should we expect?

A Paradox?

- X_i: Number of balls in bin i:

$\rightsquigarrow$ $X_1 \overset{\mathcal{D}}{=} \cdots \overset{\mathcal{D}}{=} X_m \overset{\mathcal{D}}{=} \mathrm{Bin}(n, \frac{1}{m})$

$\rightsquigarrow$ All X_i concentrated around expectation $\frac{n}{m}$ (Chernoff!)

actually, just shows $X_i = n/m \pm n^{0.501}$

Consider $\boxed{m = n}$ $\rightsquigarrow$ $\mathbb{E}[X_i] = 1$

- But also: expected number of *empty* bins:

$$\begin{aligned}
\mathbb{E}[\#i \text{ with } X_i = 0] &= \sum_{i=1}^{m} \mathbb{P}[X_i = 0] \\
&= m \cdot \left(1 - \frac{1}{m}\right)^n \qquad (m = n,\ (1 + 1/n)^n \approx e) \\
&= n \cdot e(1 \pm O(n^{-1}))
\end{aligned}$$

$\rightsquigarrow$ In expectation, $\frac{1}{e}$ fraction (37%) of bins empty!

How does that fit together with $\mathbb{E}[X_i] = 1$? Which expectation should we expect?

Birthday Paradox

- Let's consider a different question to approach this . . .
- ***Birthday 'Paradox':***
  How many people does it take to likely have two people with the same birthday?

Birthday Paradox

- Let's consider a different question to approach this ...
- ***Birthday 'Paradox':***
  How many people does it take to likely have two people with the same birthday?
- In balls-into-bins language: What n makes it likely that $\exists j \in [m] : X_j \geq 2$?

Birthday Paradox

- Let's consider a different question to approach this . . .
- ***Birthday 'Paradox':***
  How many people does it take to likely have two people with the same birthday?
- In balls-into-bins language: What n makes it likely that $\exists j \in [m] : X_j \geq 2$?

  Compute counter-probability: $\mathbb{P}[\max X_j \leq 1]$

$$\underbrace{1}_{\text{ball 1}} \cdot \underbrace{\left(1 - \frac{1}{m}\right)}_{\text{ball 2}} \cdot \left(1 - \frac{2}{m}\right) \cdots \left(1 - \frac{n-1}{m}\right)$$

Birthday Paradox

- Let's consider a different question to approach this . . .
- ***Birthday 'Paradox':***
  How many people does it take to likely have two people with the same birthday?
- In balls-into-bins language: What n makes it likely that $\exists j \in [m] : X_j \geq 2$?

  Compute counter-probability: $\mathbb{P}[\max X_j \leq 1]$

  Taylor series $e^x = \underbrace{1 + x} \pm O(x^2)$ as $x \to 0$

$$1 \cdot \left(1 - \frac{1}{m}\right) \cdot \left(1 - \frac{2}{m}\right) \cdots \left(1 - \frac{n-1}{m}\right) \quad = \quad e^{-\frac{1}{m}} \cdot e^{-\frac{2}{m}} \cdots e^{-\frac{n-1}{m}} \cdot \left(1 \pm O\left(\left(\frac{n}{m}\right)^2\right)\right)$$

Birthday Paradox

- Let's consider a different question to approach this ...
- ***Birthday 'Paradox':***
  How many people does it take to likely have two people with the same birthday?
- In balls-into-bins language: What n makes it likely that $\exists j \in [m] : X_j \geq 2$?

  Compute counter-probability: $\mathbb{P}[\max X_j \leq 1]$

  Taylor series $e^x = 1 + x \pm O(x^2)$ as $x \to 0$

$$1 \cdot \left(1 - \frac{1}{m}\right) \cdot \left(1 - \frac{2}{m}\right) \cdots \left(1 - \frac{n-1}{m}\right) = e^{-\frac{1}{m}} \cdot e^{-\frac{2}{m}} \cdots e^{-\frac{n-1}{m}} \cdot \left(1 \pm O\left(\left(\frac{n}{m}\right)^2\right)\right)$$

$$= e^{-\frac{n^2}{2m} \pm O(\frac{n}{m})} \qquad (\tfrac{n}{m} \to 0)$$

Birthday Paradox

- Let's consider a different question to approach this ...
- ***Birthday 'Paradox':***
  How many people does it take to likely have two people with the same birthday?
- In balls-into-bins language: What n makes it likely that $\exists j \in [m] : X_j \geq 2$?

  Compute counter-probability: $\mathbb{P}[\max X_j \leq 1]$ Taylor series $e^x = 1 + x \pm O(x^2)$ as $x \to 0$

$$1 \cdot \left(1 - \frac{1}{m}\right) \cdot \left(1 - \frac{2}{m}\right) \cdots \left(1 - \frac{n-1}{m}\right) = e^{-\frac{1}{m}} \cdot e^{-\frac{2}{m}} \cdots e^{-\frac{n-1}{m}} \cdot \left(1 \pm O\left(\left(\frac{n}{m}\right)^2\right)\right)$$

$$= e^{-\frac{n^2}{2m} \pm O(\frac{n}{m})} \qquad (\tfrac{n}{m} \to 0)$$

$\rightsquigarrow$ Only for $\boxed{n = \Theta(\sqrt{m})}$ nontrivial probability

- $\mathbb{P}[\max X_j \leq 1] = \frac{1}{2}$ for $n \approx \sqrt{2m \ln(2)}$, so for $m = 365$ days, need $n \approx 22.49$ people

Birthday Paradox

- Let's consider a different question to approach this ...
- ***Birthday 'Paradox':***
  How many people does it take to likely have two people with the same birthday?
- In balls-into-bins language: What n makes it likely that $\exists j \in [m] : X_j \geq 2$?

  Compute counter-probability: $\mathbb{P}[\max X_j \leq 1]$ Taylor series $e^x = 1 + x \pm O(x^2)$ as $x \to 0$

$$1 \cdot \left(1 - \frac{1}{m}\right) \cdot \left(1 - \frac{2}{m}\right) \cdots \left(1 - \frac{n-1}{m}\right) = e^{-\frac{1}{m}} \cdot e^{-\frac{2}{m}} \cdots e^{-\frac{n-1}{m}} \cdot \left(1 \pm O\left(\left(\frac{n}{m}\right)^2\right)\right)$$

$$= e^{-\frac{n^2}{2m} \pm O(\frac{n}{m})} \qquad (\tfrac{n}{m} \to 0)$$

⇝ Only for $\boxed{n = \Theta(\sqrt{m})}$ nontrivial probability

- $\mathbb{P}[\max X_j \leq 1] = \frac{1}{2}$ for $n \approx \sqrt{2m\ln(2)}$, so for $m = 365$ days, need $n \approx 22.49$ people

⇝ *Can't expect to see **all** bins close to **expected** occupancy.*

Fullest Bin

$$\hat{X} = \max_{j \in [m]} X_j$$

$\backslash\backslash$

Theorem 9.1

If we throw n balls into n bins, then w.h.p., the *fullest bin* has $O\left(\frac{\log n}{\log \log n}\right)$ balls.

Proof:

$$\mathbb{P}[\max X_j \geq M] \underset{\text{union bound}}{\leq} m \cdot \mathbb{P}[X_1 \geq M]$$

$$\mathbb{P}[X_1 \geq M] = \mathbb{P}\Big[\bigcup_{\substack{I \subseteq [n] \\ |I| = M}} \text{balls } i \in I \text{ land in bin } 1\Big]$$

$$\leq \binom{n}{M} \mathbb{P}(|I| \text{ balls land in bin } 1]$$

$$\leq \binom{n}{M} \left(\frac{1}{m}\right)^M \qquad \boxed{n = m}$$

$$= \binom{n}{M} \left(\frac{1}{n}\right)^M = \frac{n!}{M! \, (n-M)! \, n^M} \quad \leq \quad \frac{n^M}{n^M} \leq 1$$

Fullest Bin [2]

Proof (cont.):

$$\leq \frac{1}{M!} \qquad \text{Stirling} \quad M! \geq \left(\frac{M}{e}\right)^M \sqrt{2\pi M}$$

$$\leq \left(\frac{e}{M}\right)^M \cdot \frac{1}{\sqrt{2\pi M}} \quad \leq 1$$

$$\leq \left(\frac{e}{M}\right)^M \qquad \text{need this } \leq \frac{1}{n}$$

$$\left(\frac{e}{M}\right)^M \approx \frac{1}{n} \qquad M = c\,\frac{\ln n}{\ln\ln n}$$

$$\exp\left(\ln\left(\left(\frac{e}{M}\right)^M\right)\right) = \exp\left(M\left(\cancel{\ln e}^{\,1} - \ln M\right)\right) = \exp\left(-\ln n - \ln\ln n - \ln\ln n\right) \approx \frac{1}{n}$$

to show w.h.p. $\mathbb{P}[\hat{X} \geq M] = O(n^{-d})$

$$n \cdot \left(\frac{e}{M}\right)^M = n\left(\frac{e}{c} \cdot \frac{\ln\ln n}{\ln n}\right)^{c\frac{\ln n}{\ln\ln n}} \qquad \frac{e}{c} < 1 \qquad c > e$$

$$= \exp\left(\ln n + c \cdot \frac{\ln n}{\ln\ln n} \ln\left(\frac{\ln\ln n}{\ln n}\right)\right)$$

■

$$= \exp\left(\ln n + \ln n \, \frac{c \ln\ln\ln n}{\ln\ln n} - c \ln n \cdot \frac{\cancel{\ln\ln n}}{\cancel{\ln\ln n}} \right)$$

$$= \exp\left((1-c)\ln n \overset{-\ln n + \ln n}{} + \ln n \, \frac{c \ln\ln\ln n}{\ln\ln n} \right)$$

$$= n^{2-c} \cdot \underbrace{\exp\left(\ln n \left(\underbrace{\frac{c \ln\ln\ln n}{\ln\ln n}}_{o(1) \,\leq\, 1 \text{ for large } n} - 1 \right)\right)}_{\leq 1 \text{ for large } n}$$

$$\leq n^{2-c} = O(n^{-d}) \qquad \text{for } c > d+2$$

Fullest Bin – Consequences

- Closer analysis shows for $n = \alpha m$, constant α ("load factor"),

  $$\max X_j = \frac{\ln n}{\ln(\ln(n)/\alpha)} \cdot \bigl(1 + o(1)\bigr) \quad \text{w.h.p.}$$

Fullest Bin – Consequences

- Closer analysis shows for $n = \alpha m$, constant α ("load factor"),

$$\max X_j = \frac{\ln n}{\ln(\ln(n)/\alpha)} \cdot \big(1 + o(1)\big) \quad \text{w.h.p.}$$

What can we learn from this?

1. Under *uniform hashing assumption*, even **worst case** of chaining hashing cost beats BST.
2. . . . but not by much.
3. Expected costs aren't fully informative for hashing;
   (big difference between expected average case and expected worst case)

Fullest Bin – Consequences

- Closer analysis shows for $n = \alpha m$, constant α ("load factor"),

$$\max X_j = \frac{\ln n}{\ln(\ln(n)/\alpha)} \cdot \big(1 + o(1)\big) \quad \text{w.h.p.}$$

What can we learn from this?

1. Under *uniform hashing assumption*, even **worst case** of chaining hashing cost beats BST.
2. . . . but not by much.
3. Expected costs aren't fully informative for hashing;
   (big difference between expected average case and expected worst case)

Biggest caveat: uniform hashing assumption!

$\rightsquigarrow$ . . . we'll come back to that

Fullest Bin – Consequences

- Closer analysis shows for $n = \alpha m$, constant α (“load factor”),

$$\max X_j = \frac{\ln n}{\ln(\ln(n)/\alpha)} \cdot \big(1 + o(1)\big) \quad \text{w.h.p.}$$

What can we learn from this?

1. Under *uniform hashing assumption*, even **worst case** of chaining hashing cost beats BST.
2. . . . but not by much.
3. Expected costs aren’t fully informative for hashing;
   (big difference between expected average case and expected worst case)

Biggest caveat: uniform hashing assumption!

$\rightsquigarrow$ . . . we’ll come back to that

- Cool trick: *Power of 2 choices*
  Assume ***two*** candidate bins per ball (hash functions), take less loaded bin

$\rightsquigarrow$ $\max X_j = \ln\ln n / \ln 2 \pm O(1)$ (!) analysis more technical; details in *Mitzenmacher & Upfal*

Coupon Collector

- Balls into bins nicely models other situations worth memorizing
- ***Coupon Collector Problem:***
  How many (wrapped) packs do I need to buy to get all collectibles?

Coupon Collector

- Balls into bins nicely models other situations worth memorizing
- ***Coupon Collector Problem:***
  How many (wrapped) packs do I need to buy to get all collectibles?
- Balls-into-bins: What n makes it likely that $\forall j : X_j \geq 1$?
  - Define S_i as the number of balls to get from i empty bins to $i - 1$ empty bins.

  $\rightsquigarrow$ $S = S_m + S_{m-1} + \cdots + S_1$ is the total number of balls for coupon collector

Coupon Collector

- Balls into bins nicely models other situations worth memorizing
- ***Coupon Collector Problem:***
  How many (wrapped) packs do I need to buy to get all collectibles?
- Balls-into-bins: What n makes it likely that $\forall j : X_j \geq 1$?
  - Define S_i as the number of balls to get from i empty bins to $i-1$ empty bins.
  - $\rightsquigarrow$ $S = S_m + S_{m-1} + \cdots + S_1$ is the total number of balls for coupon collector
  - $S_i \overset{\mathcal{D}}{=} \mathrm{Geo}(p_i)$ where $p_i = \dfrac{i}{m}$

Coupon Collector

- Balls into bins nicely models other situations worth memorizing
- ***Coupon Collector Problem:***
  How many (wrapped) packs do I need to buy to get all collectibles?
- Balls-into-bins: What n makes it likely that $\forall j : X_j \geq 1$?
  - Define S_i as the number of balls to get from i empty bins to $i-1$ empty bins.

  $\rightsquigarrow$ $S = S_m + S_{m-1} + \cdots + S_1$ is the total number of balls for coupon collector
  - $S_i \overset{\mathcal{D}}{=} \mathrm{Geo}(p_i)$ where $p_i = \dfrac{i}{m}$ $\rightsquigarrow$ $\mathbb{E}[S_i] = \dfrac{1}{p_i} = \dfrac{m}{i}$

Coupon Collector

- Balls into bins nicely models other situations worth memorizing
- ***Coupon Collector Problem:***
  How many (wrapped) packs do I need to buy to get all collectibles?
- Balls-into-bins: What n makes it likely that $\forall j : X_j \geq 1$?
  - Define S_i as the number of balls to get from i empty bins to $i-1$ empty bins.

  $\rightsquigarrow$ $S = S_m + S_{m-1} + \cdots + S_1$ is the total number of balls for coupon collector
  - $S_i \stackrel{\mathcal{D}}{=} \mathrm{Geo}(p_i)$ where $p_i = \dfrac{i}{m}$ $\rightsquigarrow$ $\mathbb{E}[S_i] = \dfrac{1}{p_i} = \dfrac{m}{i}$
  - $\mathbb{E}[S] = \sum_{i=1}^{m} \mathbb{E}[S_i] = m \sum_{i=1}^{m} \dfrac{1}{i} = mH_m = m \ln m \pm O(m)$

Coupon Collector

- Balls into bins nicely models other situations worth memorizing
- ***Coupon Collector Problem:***
  How many (wrapped) packs do I need to buy to get all collectibles?
- Balls-into-bins: What n makes it likely that $\forall j : X_j \ge 1$?
  - Define S_i as the number of balls to get from i empty bins to $i-1$ empty bins.

  $\rightsquigarrow$ $S = S_m + S_{m-1} + \cdots + S_1$ is the total number of balls for coupon collector
  - $S_i \overset{\mathcal{D}}{=} \mathrm{Geo}(p_i)$ where $p_i = \dfrac{i}{m}$ $\rightsquigarrow$ $\mathbb{E}[S_i] = \dfrac{1}{p_i} = \dfrac{m}{i}$
  - $\mathbb{E}[S] = \displaystyle\sum_{i=1}^{m} \mathbb{E}[S_i] = m \sum_{i=1}^{m} \frac{1}{i} = mH_m = m \ln m \pm O(m)$
- Can similarly show $\mathrm{Var}[S] = \Theta(m^2)$
  (since S_i are independent, stdev is linear + using $\mathrm{Var}[S_i] = \dfrac{1-p_i}{p_i^2}$)

  $\rightsquigarrow$ $\sigma[S] = \Theta(m) = o(\mathbb{E}[S])$, so S converges in probability to $\mathbb{E}[S]$ (Chebyshev)

9.2 Universal Hashing

Randomized Hashing

- Balls-into-bins model is worryingly optimistic.
  - Assumes that chosen bins $B_1, \ldots, B_n \in [m]$ are *mutually independent*.
  - $\rightsquigarrow$ Assumes both that input is not adversarial **and** that hash functions work well.

Randomized Hashing

- Balls-into-bins model is worryingly optimistic.
  - Assumes that chosen bins $B_1, \ldots, B_n \in [m]$ are *mutually independent*.
  - ⇝ Assumes both that input is not adversarial **and** that hash functions work well.

⇝ To replace the assumption about the input by explicit randomization, would need a *fully random hash function* $h : [n] \to [m]$

- if we were to uniformly choose from m^n possibilities
  we'd need to store $\lg(m^n) = n \lg m$ bits just for h
- (even if we did so, how to efficiently *evaluate* h then is unclear)
- ϟ too expensive

Randomized Hashing

- Balls-into-bins model is worryingly optimistic.
  - Assumes that chosen bins $B_1, \ldots, B_n \in [m]$ are *mutually independent*.
  - ⇝ Assumes both that input is not adversarial **and** that hash functions work well.

⇝ To replace the assumption about the input by explicit randomization, would need a *fully random hash function* $h : [n] \to [m]$

  - if we were to uniformly choose from m^n possibilities
    we'd need to store $\lg(m^n) = n \lg m$ bits just for h
  - (even if we did so, how to efficiently *evaluate* h then is unclear)
  - ϟ too expensive

⇝ Pick h at random, but from a smaller class $\mathcal{H}$ of "convenient" functions

Universal Hashing

What's a convenient class?

Definition 9.2 (Universal Family)

Let $\mathcal{H}$ be a set of hash functions from U to $[m]$ and $|U| \ge m$.
Assume $h \in \mathcal{H}$ is chosen uniformly at random.

(a) Then $\mathcal{H}$ is called a *universal* if

$$\forall x_1, x_2 \in U \quad : \quad x_1 \neq x_2 \implies \mathbb{P}_h\big[h(x_1) = h(x_2)\big] \le \frac{1}{m}.$$

(b) $\mathcal{H}$ is called *strongly universal* or *pairwise independent* if

$$\forall x_1, x_2 \in U, y_1, y_2 \in R \quad : \quad x_1 \neq x_2 \implies \mathbb{P}_h\big[h(x_1) = y_1 \wedge h(x_2) = y_2\big] \le \frac{1}{m^2}.$$

Universal Hashing

What's a convenient class?

Definition 9.2 (Universal Family)

Let $\mathcal{H}$ be a set of hash functions from U to $[m]$ and $|U| \geq m$.
Assume $h \in \mathcal{H}$ is chosen uniformly at random.

(a) Then $\mathcal{H}$ is called a *universal* if

$$\forall x_1, x_2 \in U \;:\; x_1 \neq x_2 \implies \mathbb{P}\big[h(x_1) = h(x_2)\big] \leq \frac{1}{m}.$$

(b) $\mathcal{H}$ is called *strongly universal* or *pairwise independent* if

$$\forall x_1, x_2 \in U, y_1, y_2 \in R \;:\; x_1 \neq x_2 \implies \mathbb{P}\big[h(x_1) = y_1 \wedge h(x_2) = y_2\big] \leq \frac{1}{m^2}.$$

◀

- strong universal implies universal
- In the following, always assume (uniformly) **random** $h \in \mathcal{H}$.
- by contrast, $x_1, \ldots, x_n$ may be chosen adversarially (but all distinct) from $[u]$

Examples of universal families

$$
\begin{aligned}
h_{ab}(x) &= \Big(a \cdot x + b \bmod p\Big) \bmod m & & p \text{ prime}, p \geq m \\
h_a(x) &= \Big(a \cdot x \bmod 2^k\Big) \operatorname{div} 2^{k-\ell} & & u = 2^k, m = 2^\ell
\end{aligned}
$$

- $\mathcal{H}_1 = \big\{h_{ab} : a \in [1..p), b \in [0..p)\big\}$ is universal
- $\mathcal{H}_0 = \big\{h_{ab} : a \in [0..p), b \in [0..p)\big\}$ is strongly universal
- $\mathcal{H}_2 = \big\{h_a : a \in [1..2^k), a \text{ odd}\big\}$ is universal

How good is universal hashing?

balls into bins

$\hat{X} \approx \frac{\ln n}{\ln \ln n}$

Theorem 9.3

Assign $x_1, \ldots, x_n \in [u]$ to bins $h(x_i) \in [m]$ using hash function h, uniformly chosen from a universal family of hash functions $\mathcal{H}$.

Let X_j be the load of bin $j \in [m]$.

Then $\mathbb{P}\left[\underbrace{\max X_j}_{\hat{X}} \geq \sqrt{2} \cdot \frac{n}{\sqrt{m}}\right] \leq \frac{1}{2}$.

$n = m \quad \sqrt{2n}$

$\boxed{X_j \overset{\mathcal{D}}{\neq} \mathrm{Bin}(n,p)}$

Proof:

C_{ij} = 'x_i and x_j collide' $= [h(x_i) = h(x_j)]$

$\underset{\text{univ.}}{\Rightarrow} \quad \mathbb{P}[C_{ij}] \leq \frac{1}{m}$

$C = \sum_{1 \leq i < j \leq n} C_{ij} \qquad \mathbb{E}[C] = \sum \mathbb{E}[C_{ij}] \leq \binom{n}{2} \cdot \frac{1}{m} < \frac{n^2}{2m}$

$\hat{X}$ itself implies $\binom{\hat{X}}{2}$ collisions

$\Rightarrow \quad C \geq \binom{\hat{X}}{2} = \frac{\hat{X}(\hat{X}-1)}{2} \geq \frac{(\hat{X}-1)^2}{2}$

How good is universal hashing [2]

Proof:

$$\mathbb{P}\left[\hat{X} \geq n\sqrt{\frac{2}{m}}\right] \leq \mathbb{P}\left[C \geq \frac{n^2}{m}\right] = \mathbb{P}\left[C \geq 2\cdot\mathbb{E}[C]\right] \underset{\text{Markov}}{\leq} \frac{1}{2}$$

$$\wedge$$

$$+1$$

then $\hat{X}^2 \geq n\sqrt{\frac{2}{m}} + 1$ implies

$$\frac{(\hat{X}-1)^2}{2} \geq \frac{n^2}{m}$$ which implies

$$C \geq \frac{n^2}{m}$$

$\square$

So, how good is universal hashing?

- For $n = m$, fullest bin $\leq \sqrt{2n}$
- Much worse than $\Theta(\log n / \log \log n)$!

So, how good is universal hashing?

- For $n = m$, fullest bin $\le \sqrt{2n}$
- Much worse than $\Theta(\log n / \log \log n)$!
- Note that we only proved an upper bound, however
  - bound is tight in the worst case
    (if all we know is pairwise independence of hash values)
    $\rightsquigarrow$ exercises
  - for practical choices like $\mathcal{H}_0, \mathcal{H}_1, \mathcal{H}_2$ better bounds are proven
    (close to $O(n^{1/3})$ instead of $O(n^{1/2})$)
    but still far worse than uniform hashing

9.3 Perfect Hashing

Perfect Hashing: Random Sampling

A hash function $h : [u] \to [m]$ is called

- *perfect* for a set $\mathcal{X} = \{x_1, \ldots, x_n\} \subset [u]$ if $i \neq j$ implies $h(x_i) \neq h(x_j)$
- *minimal* for set $\mathcal{X} = \{x_1, \ldots, x_n\} \subset [u]$ if $m = n$

Perfect Hashing

- only possible for $n \leq m$
- stringent requirement $\rightsquigarrow$ here focus on static $\mathcal{X}$
  - carefully chosen variants with partial rebuilding allow insertion and deletion in $O(1)$ amortized expected time

Perfect Hashing: Random Sampling

A hash function $h : [u] \to [m]$ is called

- *perfect* for a set $\mathcal{X} = \{x_1, \ldots, x_n\} \subset [u]$ if $i \neq j$ implies $h(x_i) \neq h(x_j)$
- *minimal* for set $\mathcal{X} = \{x_1, \ldots, x_n\} \subset [u]$ if $m = n$

Perfect Hashing

- only possible for $n \leq m$
- stringent requirement $\rightsquigarrow$ here focus on static $\mathcal{X}$
  - carefully chosen variants with partial rebuilding allow insertion and deletion in $O(1)$ amortized expected time
- further requirements
  1. Hash function must be fast to evaluate (ideally $O(1)$ time)
  2. Hash function must be small to store (ideally $O(n)$ space)
  3. should be fast to compute given $\mathcal{X}$ (ideally $O(n)$ time)
  4. Have small m (ideally $m = \Theta(n)$)

Perfect Hashing: Simple, but space inefficient

Start simple: pick $h \in \mathcal{H}$ from universal class

what can we guarantee? n fixed

$\Rightarrow$ increase m until likely that h perfect

birthday paradox! $n \approx \sqrt{m}$

So let's try $m \geq n^2$

$\hat{X} \geq 2$ implies $C \geq 1$

which implies $C \geq \frac{n^2}{m}$

$$\mathbb{P}[\hat{X} \geq 2] \leq \mathbb{P}\left[C \geq \frac{n^2}{m}\right] = \mathbb{P}[C \geq 2\,\mathbb{E}(C)] \leq \frac{1}{2}$$

as for universal hashing above

$$\Rightarrow \mathbb{P}\{\text{no collisions}\} \geq \frac{1}{2}$$

good prob. to get a perfect hash function! $\lightning$ HT has n^2 space

Perfect Hashing: Two-tier solution

h_1 $\mathcal{X}$ — bucket 1, bucket 2, $\vdots$

$O(n)$ tier 1 buckets

for each tier 1 bucket i

choose $h_2^{(i)}$ as above

using $m_i = X_i^2$ $\quad X_i = \#$ elements in bucket i

↳ after expected linear time

overall hash function is perfect

$x \mapsto h_2^{(h_1(x))}(x)$

To show: overall space (for all secondary hash tables) small

Perfect Hashing:

(1) Choose h_1 uniformly from $\mathcal{H}$ (universal)
with $m = n$ bins until C (#collisions) $\leq n$

(2) For each bucket $i = 1, \dots, n$
If $X_i \geq 2$ draw random hash function $h_2^{(i)}$ from $\mathcal{H}'$ (universal)
with X_i^2 bins
repeat until $h_2^{(i)}$ perfect

X_i in bin $\Rightarrow \binom{X_i}{2}$ collisions
$\frac{X_i^2}{2} - \frac{X_i}{2}$

(3) Output $h_2^{(h_1(x))}$

Claim: (a) #repetitions small
(b) space usage small

(a) (1) $\mathbb{E}[C] = \binom{n}{2} \cdot \frac{1}{n} = \frac{n-1}{2} \xrightarrow{\text{Markov}} \mathbb{P}[C \leq n] \geq \frac{1}{2}$
(2) see above: $\mathbb{E}$ #rep. $= 2$
$\Rightarrow$ $\mathbb{E}$ time $= O(n)$

(b) space $= \Theta(n) + \sum_{i=1}^{n} X_i^2 = \Theta(n) + \underbrace{\sum \binom{X_i}{2}}_{= C \leq n} + \underbrace{\sum \frac{X_i}{2}}_{\leq n}$
top level HT and all $h_2^{(i)}$'s
$= \Theta(n)$

9.4 Primality Testing

Abundance of Witnesses

- Suppose $L \in \mathsf{NP}$ and all of the following are true:
  - alleged certificate must be easy to check
    trivially in polytime; often very fast
  - for $x \in L$, there are **many** certificates that show $x \in L$
    not generally true, but sometimes!

$\rightsquigarrow$ Conceivable that a randomized algorithm succeeds:
  - Guess a random certificate string
  - Check if it decides the problem

Primality Testing

Testing if a given number n is *prime* is one of the oldest algorithmic questions.

Trivial approach: test for all (primes) $p \le \sqrt{n}$ whether $p \mid n$

```
procedure sieveOfEratosthenes(n):
    isPrime[2..n] := true
    for i := 2, 3, ..., ⌊√n⌋
        if isPrime[i]
            for j = i, i+1, i+2, ..., ⌊n/i⌋
                isPrime[i · j] := false
    return {p ∈ [2..n] : isPrime[p]}

procedure isPrimeTrivial(n):
    P := sieveOfEratosthenes(⌊√n⌋)
    return ∀p ∈ P : p ∤ n
```

Primality Testing

Testing if a given number n is *prime* is one of the oldest algorithmic questions.

Trivial approach: test for all (primes) $p \le \sqrt{n}$ whether $p \mid n$

```
procedure sieveOfEratosthenes(n):
    isPrime[2..n] := true
    for i := 2, 3, ..., ⌊√n⌋
        if isPrime[i]
            for j = i, i+1, i+2, ..., ⌊n/i⌋
                isPrime[i · j] := false
    return {p ∈ [2..n] : isPrime[p]}

procedure isPrimeTrivial(n):
    P := sieveOfEratosthenes(⌊√n⌋)
    return ∀p ∈ P : p ∤ n
```

Running time:

- dominated by sieving primes up to $m = \lfloor \sqrt{n} \rfloor$
- $T(m) \le m + \sum_{\substack{p \le m \\ p \text{ prime}}} \frac{m}{p} \le m + m \sum_{p=1}^{m} \frac{1}{p}$

$\rightsquigarrow T(m) = O(m \log m)$

Primality Testing

Testing if a given number n is *prime* is one of the oldest algorithmic questions.

Trivial approach: test for all (primes) $p \le \sqrt{n}$ whether $p \mid n$

```
procedure sieveOfEratosthenes(n):
    isPrime[2..n] := true
    for i := 2, 3, ..., ⌊√n⌋
        if isPrime[i]
            for j = i, i+1, i+2, ..., ⌊n/i⌋
                isPrime[i · j] := false
    return {p ∈ [2..n] : isPrime[p]}

procedure isPrimeTrivial(n):
    P := sieveOfEratosthenes(⌊√n⌋)
    return ∀p ∈ P : p ∤ n
```

Running time:

- dominated by sieving primes up to $m = \lfloor \sqrt{n} \rfloor$
- $T(m) \;\le\; m + \sum\limits_{\substack{p \le m \\ p \text{ prime}}} \frac{m}{p} \;\le\; m + m \sum\limits_{p=1}^{m} \frac{1}{p}$

$\rightsquigarrow$ $T(m) = O(m \log m)$

- closer analysis: actually $T(m) = O(m \log \log m)$

Space: $\sqrt{n}$ bits

Complexity of Primality Testing and Factorization

- PRIMES:
  - **Given:** Integer n in binary encoding
  - **Goal:** Check if n is a prime number
- INTEGERFACTORIZATION:
  - **Given:** Integer n in binary encoding
  - **Goal:** Find nontrivial factors $n = m_1 \cdot m_2, 2 \leq m_1, m_2 < n$ or determine "n prime"

Complexity of Primality Testing and Factorization

- PRIMES:
  - **Given:** Integer n in binary encoding
  - **Goal:** Check if n is a prime number
- INTEGERFACTORIZATION:
  - **Given:** Integer n in binary encoding
  - **Goal:** Find nontrivial factors $n = m_1 \cdot m_2, 2 \le m_1, m_2 < n$ or determine "n prime"
- If n is composite, a factorization is a certificate for *non-primality* $\rightsquigarrow$ PRIMES $\in$ co-NP
  - n encoded in binary $\rightsquigarrow$ Sieve of Eratosthenes is pseudopolynomial

$\in$ NP ?

Complexity of Primality Testing and Factorization

- PRIMES:
  - **Given:** Integer n in binary encoding
  - **Goal:** Check if n is a prime number
- INTEGERFACTORIZATION:
  - **Given:** Integer n in binary encoding
  - **Goal:** Find nontrivial factors $n = m_1 \cdot m_2$, $2 \le m_1, m_2 < n$ or determine "n prime"
- If n is composite, a factorization is a certificate for *non-primality* $\rightsquigarrow$ PRIMES $\in$ co-NP
  - n encoded in binary $\rightsquigarrow$ Sieve of Eratosthenes is pseudopolynomial
- we will show PRIMES $\in$ co-RP $\subset$ BPP

Complexity of Primality Testing and Factorization

- PRIMES:
  - **Given:** Integer n in binary encoding
  - **Goal:** Check if n is a prime number
- INTEGERFACTORIZATION:
  - **Given:** Integer n in binary encoding
  - **Goal:** Find nontrivial factors $n = m_1 \cdot m_2, 2 \le m_1, m_2 < n$ or determine "n prime"
- If n is composite, a factorization is a certificate for *non-primality* $\rightsquigarrow$ PRIMES $\in$ co-NP
  - n encoded in binary $\rightsquigarrow$ Sieve of Eratosthenes is pseudopolynomial
- we will show PRIMES $\in$ co-RP $\subset$ BPP
- Major theoretical breakthrough: PRIMES $\in$ P *Agrawal, Kayal, and Saxena (2004)*
- This is not known for INTEGERFACTORIZATION
  - Indeed much of classic cryptography (RSA) builds on factoring being intractable
  - *Shor's algorithm* can factor integers on a (theoretical) quantum computer in polytime!
    (not clear whether or when this is a practical concern)

Does PRIMES have abundance of witnesses?

NOTPRIME

try the obvious:

factors? composite numbers w/ 2 large prime factors ?

hardly promising approach

since it solves FACTORIZATION

Primality Testing: Fermat's Little Theorem

Theorem 9.4 (Fermat's Little Theorem)
For p a prime and $a \in [1..p-1]$ holds

$$a^{p-1} \equiv 1 \pmod{p} \qquad (*)$$

Pick a random $a \in [1..p-1]$, compute $a^{p-1} \bmod p$ if $\neq 1$

$\Rightarrow p$ not prime.

only $\Rightarrow$, not $\Leftarrow$

indeed Carmichael numbers are not prime, but fulfill $(*)$

Primality Testing: Second Attempt

Theorem 9.5 (Euler's Criterion)
Let $p > 2$ an odd number.

field $\mathbb{Z}$ modulo p

$$p \text{ prime} \iff \forall a \in \mathbb{Z}_p \setminus \{0\} : a^{\frac{p-1}{2}} \bmod p \in \{1, \overset{p-1}{\overset{\shortparallel}{-1}}\}$$

Primality Testing: Second Attempt

Theorem 9.5 (Euler's Criterion)
Let $p > 2$ an odd number.

$$p \text{ prime} \iff \forall a \in \mathbb{Z}_p \setminus \{0\} : a^{\frac{p-1}{2}} \bmod p \in \{1, -1\}$$

Theorem 9.6 (Number of Witnesses)
For every odd $n \in \mathbb{N}$, $(n-1)/2$ odd, we have:

1. If n is prime then $a^{(n-1)/2} \bmod n \in \{1, n-1\}$, for all $a \in \{1, \ldots, n-1\}$.
2. If n is not prime then $a^{(n-1)/2} \bmod n \notin \{1, n-1\}$ for *at least half* of the elements in $\{1, \ldots, n-1\}$.

Simple Solovay-Strassen Primality Test

Input: an odd number n with $(n-1)/2$ odd.

1. Choose a random $a \in \{1, 2, \ldots, n-1\}$.
2. Compute $A := a^{(n-1)/2} \bmod n$.
3. If $A \in \{1, n-1\}$ then output "n probably prime" (reject);
4. otherwise output "n not prime" (accept).

Simple Solovay-Strassen Primality Test

Input: an odd number n with $(n-1)/2$ odd.

1. Choose a random $a \in \{1, 2, \ldots, n-1\}$.
2. Compute $A := a^{(n-1)/2} \bmod n$.
3. If $A \in \{1, n-1\}$ then output "n probably prime" (reject);
4. otherwise output "n not prime" (accept).

Theorem 9.7 (Correctness)

The simple Solovay-Strassen algorithm is a polynomial **OSE-MC** algorithm to detect composite numbers n with $n \bmod 4 = 3$. ◀

Simple Solovay-Strassen Primality Test

Input: an odd number n with $(n-1)/2$ odd.

1. Choose a random $a \in \{1, 2, \ldots, n-1\}$.
2. Compute $A := a^{(n-1)/2} \bmod n$.
3. If $A \in \{1, n-1\}$ then output "n probably prime" (reject);
4. otherwise output "n not prime" (accept).

Theorem 9.7 (Correctness)

The simple Solovay-Strassen algorithm is a polynomial **OSE-MC** algorithm to detect composite numbers n with $n \bmod 4 = 3$. ◀

Corollary 9.8

For positive integers n with $n \bmod 4 = 3$ the simple Solovay-Strassen algorithm provides a polynomial **TSE-MC** algorithm to detect prime numbers. ◀

Sampling Primes

RandomPrime(ℓ, k) Input: $\ell, k \in \mathbb{N}$, $\ell \geq 3$.

1. Set $X :=$ "not found yet"; $I := 0$;
2. `while` $X =$ "not found yet" and $I < 2\ell^2$ `do`
   - generate random bit string $a_1, a_2, \ldots, a_{\ell-2}$ and
   - compute $n := 2^{\ell-1} + \sum_{i=1}^{\ell-2} a_i \cdot 2^i + 1$

     // This way n becomes a random, odd number of length ℓ
   - Realize k independent runs of Solovay-Strassen-algorithm on n;
   - `if` at least one output says "$n \notin PRIMES$" `then` $I := I + 1$

     `else` $X :=$"PN found"; `output` n;
3. `if` $I = 2 \cdot \ell^2$ `then output` "no PN found".

$O(\ell^5)$

Thm: RandomPrime(ℓ, ℓ) is TSE-MC, poly-time for generating primes of length ℓ.

to show (a) output prime w/ prob $\left(\frac{1}{2}\right)^k$

(b) does not fail w/ prob $\frac{1}{2\ell}$

key ingredient: prob to hit a prime number

when choosing odd n uniformly $\in [2^{\ell-1}, 2^{\ell})$

$\boxed{\underline{\text{Prime Number Theorem:}} \quad \pi(x) = \#\text{primes} \leq x = \frac{x}{\ln(x)}(1+o(1))}$

$\rightsquigarrow$ # repetitions to sample prime $\approx \ell$

9.5 Schöning’s Satisfiability

Random Sampling

If a solution is tricky to construct in a target fashion,
but many solutions are known to exist, random sampling can help.

Generate random object according to simple procedure until solution found.

We've seen ideas of random sampling in perfect hashing.

Now: Use more aggressive sampling to find rare objects.

Warmup: 2SAT

Famously, 3SAT is NP-complete.

2SAT: Given CNF formula φ with $\leq$ **2** literals per clause; is φ satisfiable?

Warmup: 2SAT

Famously, 3SAT is NP-complete.

2SAT: Given CNF formula φ with $\leq$ **2** literals per clause; is φ satisfiable?

By contrast, 2SAT $\in P$

Warmup: 2SAT

Famously, 3SAT is NP-complete.

2SAT: Given CNF formula φ with $\leq$ **2** literals per clause; is φ satisfiable?

By contrast, 2SAT $\in P$ $\qquad a \to b \; \equiv \; \neg a \vee b$

Idea: Any clause $(\ell_1 \vee \ell_2)$ is equivalent to the *implications* $\neg\ell_1 \to \ell_2$ and $\neg\ell_2 \to \ell_1$

$\rightsquigarrow$ Represent formula as *implication graph*:

- vertices = literals in φ
- edges = all implications equivalent to some clause

Warmup: 2SAT

Famously, 3SAT is NP-complete.

2SAT: Given CNF formula φ with $\leq$ **2** literals per clause; is φ satisfiable?

By contrast, 2SAT $\in P$

Idea: Any clause $(\ell_1 \vee \ell_2)$ is equivalent to the *implications* $\neg\ell_1 \to \ell_2$ and $\neg\ell_2 \to \ell_1$

$\rightsquigarrow$ Represent formula as *implication graph*:

- vertices = literals in φ
- edges = all implications equivalent to some clause

$\rightsquigarrow$ Can show: φ satisfiable $\iff$ no SCC contains both x_i and $\neg x_i$

strongly connected component

- SCCs computable in linear time
- indeed, if no strong component contains contradiction, topological sort of components allows to read off satisfying assignment

Warmup: 2SAT

Famously, 3SAT is NP-complete.

2SAT: Given CNF formula φ with $\leq$ **2** literals per clause; is φ satisfiable?

By contrast, 2SAT $\in P$

Idea: Any clause $(\ell_1 \vee \ell_2)$ is equivalent to the *implications* $\neg\ell_1 \to \ell_2$ and $\neg\ell_2 \to \ell_1$

$\rightsquigarrow$ Represent formula as *implication graph*:

- vertices = literals in φ
- edges = all implications equivalent to some clause

$\rightsquigarrow$ Can show: φ satisfiable $\iff$ no SCC contains both x_i and $\neg x_i$

strongly connected component

- SCCs computable in linear time
- indeed, if no strong component contains contradiction, topological sort of components allows to read off satisfying assignment

$\rightsquigarrow$ Basically, a solved problem . . . we will use it for demonstration purposes only

Warmup: A randomized 2SAT algorithm

```
procedure localSearch2SAT(φ, confidence):
    k := number of variables in ϕ
    Choose assignment α ∈ {0,1}^k uniformly at random.
    for j = 1, ..., confidence · 2k^2
        if α fulfills φ return α // satisfiable!
        Arbitrarily choose clause C = ℓ_1 ∨ ℓ_2 not satisfied under α.
        Choose ℓ from {ℓ_1, ℓ_2} uniformly at random.
        α = assignment obtained by negating ℓ.
    return PROBABLY_NOT_SATISFIABLE
```

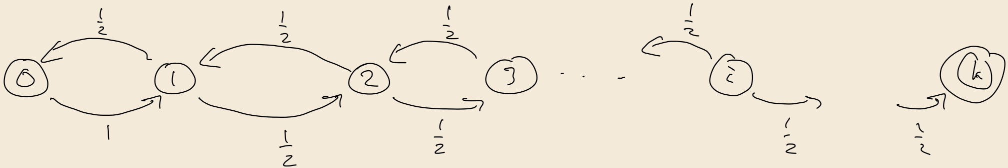
Warmup: A randomized 2SAT algorithm

procedure localSearch2SAT(φ, *confidence*):
  k := number of variables in ϕ
  Choose assignment $\alpha \in \{0,1\}^k$ uniformly at random.
  for $j = 1, \ldots, \mathit{confidence} \cdot 2k^2$
    if α fulfills φ **return** α // *satisfiable!*
    Arbitrarily choose clause $C = \ell_1 \vee \ell_2$ not satisfied under α.
    Choose ℓ from $\{\ell_1, \ell_2\}$ uniformly at random.
    α = assignment obtained by negating ℓ.
  return `PROBABLY_NOT_SATISFIABLE`

Theorem 9.10 (localSearch2SAT is OSE-MC for 2SAT)

Let φ be a 2SAT formula.

1. If φ is unsatisfiable, localSearch2SAT always returns `PROBABLY_NOT_SATISFIABLE`.
2. If φ is satisfiable, localSearch2SAT returns satisfying assignment with probability at least $1 - 2^{-\mathit{confidence}}$.



X_t

y_i = starting in (i) expected # steps until (k)