Summer School on Strings & Privacy

SDU
University of
Southern Denmark

Program

Saturday 13.06.

→**10-12h: Part I: Introduction to Differential Privacy**

→12-13h Lunch (Provided)

→**13h-16h Part II: Differential Privacy for Strings**

Sunday 14.06.

→**10-13h: Part III: Combinatorial Sanitization of Strings (Solon Pissis)**

→13h Lunch (Provided)

→Afternoon: Canoeing in Mølleå – bus from here. Rikke will help you find your way back to Copenhagen afterwards

SDU
University of Southern Denmark

Part 1: Intro to Differential Privacy

SDU
University of
Southern Denmark

Part 1

→ Why do we need strong privacy guarantees?

→ Definition of differential privacy

→ Basic techniques for designing differentially private algorithms

Why Differential Privacy?

→ We collect and analyze data all the time

→ How do we make it anonymous?

→ Case: You are a streaming service for TV shows. You want to improve your recommendation algorithms. For this, you write out a competition, and share some of your user data to test the code on. The algorithm which performs best wins. **How do you do anonymize the data?**

Example 1: How not to - Netflix Prize[1]

On October 2, 2006, Netflix, the world's largest online DVD rental service, announced the $1-million Netflix Prize for improving their movie recommendation service [11]. To aid contestants, Netflix publicly released a dataset containing 100, 480, 507 movie ratings, created by 480, 189 Netflix subscribers between December 1999 and December 2005. At the end of 2005, Netflix had approximately 4 million subscribers, so almost $\frac{1}{8}$ of them had their records published. The ratings data appear to not have been perturbed to any significant extent (see appendix C).

While movie ratings are not as sensitive as, say, medical records, release of massive amounts of data about individual Netflix subscribers raises interesting privacy issues. Among the Frequently Asked Questions on the Netflix Prize webpage [20], there is the following question: "Is there any customer information in the dataset that should be kept private?" Netflix answers this question as follows:

> "No, all customer identifying information has been removed; all that remains are ratings and dates. This follows our privacy policy, which you can review here. Even if, for example, you knew all your own ratings and their dates you probably couldn't identify them reliably in the data because only a small sample was included (less than one-tenth of our complete dataset) and that data was subject to perturbation. Of course, since you know all your own ratings that really isn't a privacy problem is it?"

[1]Arvind Narayanan, Vitaly Shmatikov: Robust De-anonymization of Large Sparse Datasets. SP 2008: 111-125

Example 1: How not to - Netflix Prize[1]

Results of de-anonymization. Very little auxiliary information is needed for de-anonymize an average subscriber record from the Netflix Prize dataset. With 8 movie ratings (of which 2 may be completely wrong) and dates that may have a 14-day error, 99% of records be uniquely identified in the dataset.

Even without any dates, a substantial privacy breach occurs, especially when the auxiliary information consists of movies that are not blockbusters (Figure 4).[1] Two movies are no longer sufficient, but 84% of subscribers can be uniquely identified if the adversary knows 6 out of 8 moves outside the top 500 (as shown in appendix D, this is not a significant limitation).

→Is it possible to find the needed information for de-identifying?

→imdb.com

→Is this a problem?

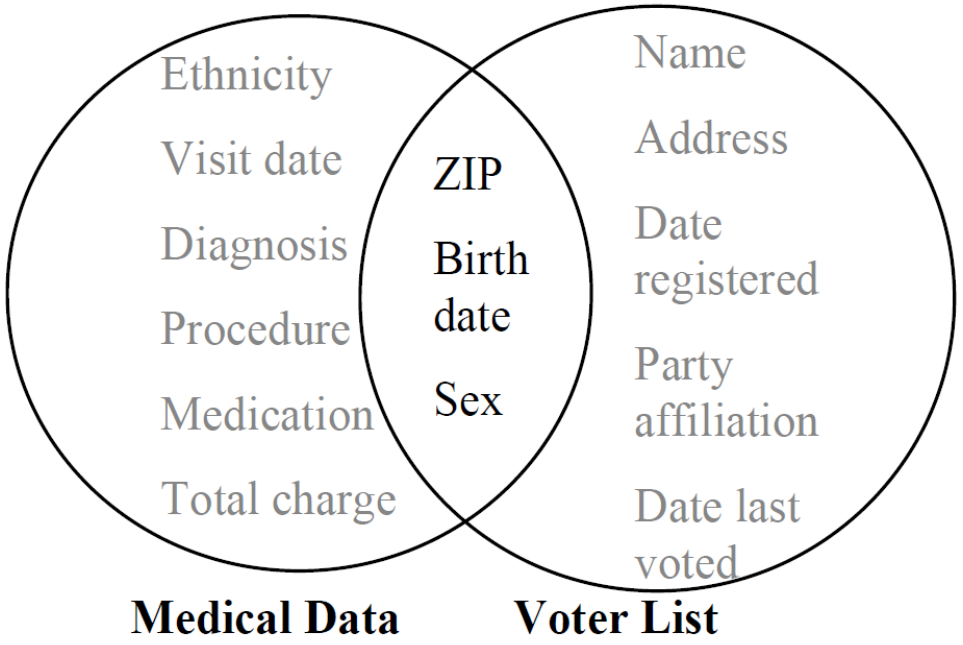
[1]Arvind Narayanan, Vitaly Shmatikov: Robust De-anonymization of Large Sparse Datasets. SP 2008: 111-125

Example 2: Massachusetts Group Insurance Commission

→MGIC: made hospital visit records freely available to researchers

→"anonymized": ZIP code, birthday, sex, and condition

→Latanya Sweeney: uniquely identifies 87% of US population



Take away

→What did both of the cases we just saw have in common?

What we would like of a privacy definition

→ Protect secret information of any individual

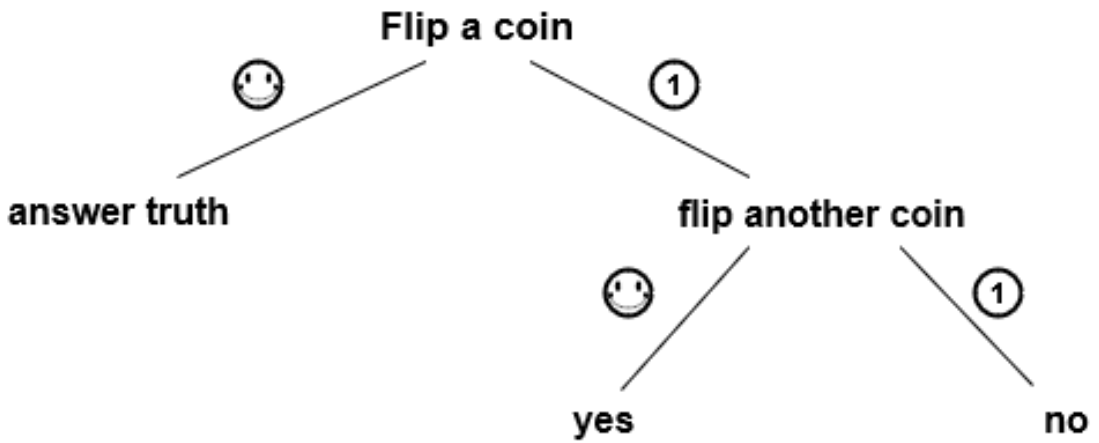
→ Be robust with respect to background knowledge

→ Understand what happens if we combine several ”privatized” statistics (composition)

→ Understand trade-off between privacy and accuracy

Example: Randomized Response [Warner 65]

→ „Have you ever cheated at a university exam?“



Example: Randomized Response [Warner 65]

- ✓ Protect secret information of any individual
- ✓ Be robust with respect to background knowledge

→ Understand what happens if we combine several ”privatized” statistics (composition)

→Understand trade-off between privacy and accuracy

Example: Randomized Response [Warner 65]

- ✓ Protect secret information of any individual
- ✓ Be robust with respect to background knowledge

→ Understand what happens if we combine several ”privatized” statistics (composition) → later

→Understand trade-off between privacy and accuracy → Exercise.

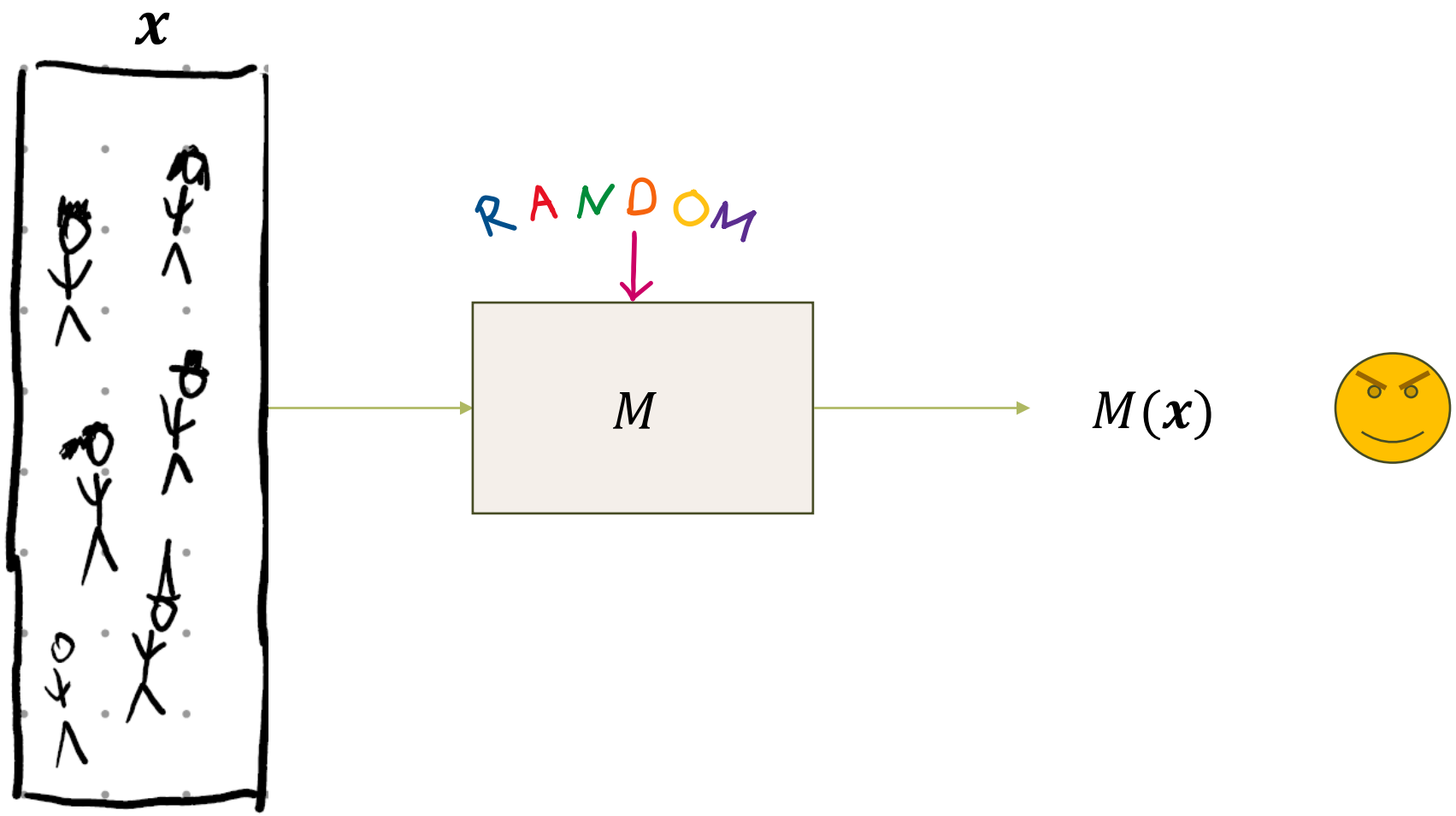
Differential Privacy [Dwork, McSherry, Nissim, Smith ’03]

→General framework for how randomization protects the privacy of individuals in a data set

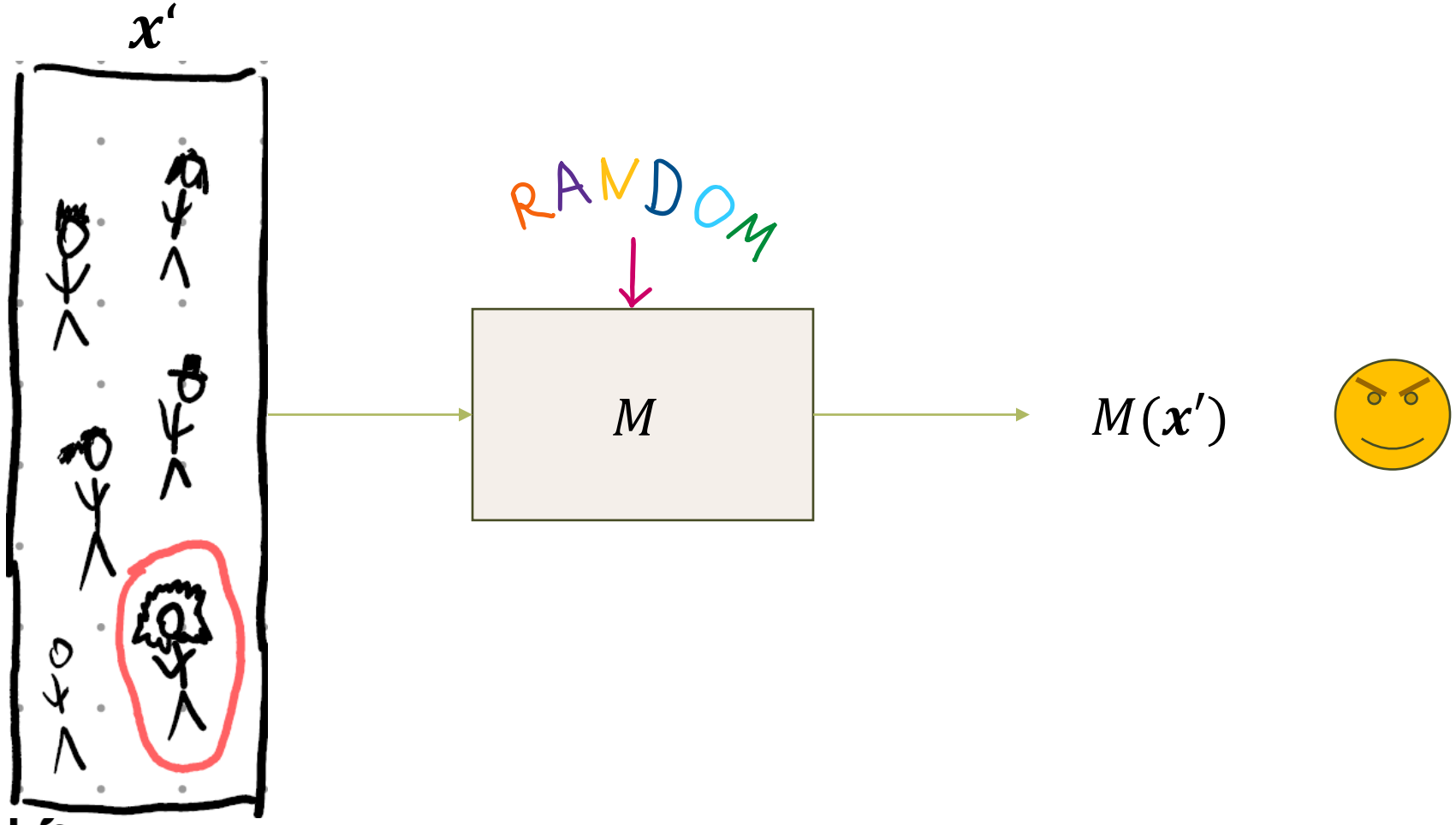
→Inspired by the attacks and by Randomized Response

→Today: Seen as the golden standard for privacy and used by US census bureau, Google, Apple, LinkedIn…

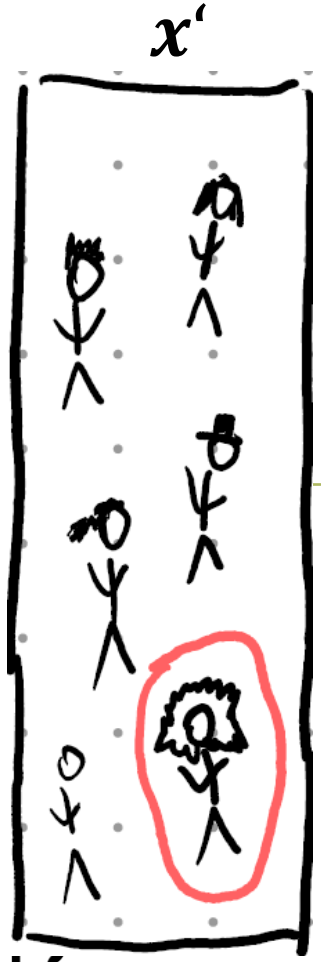
Anonymization through Randomization

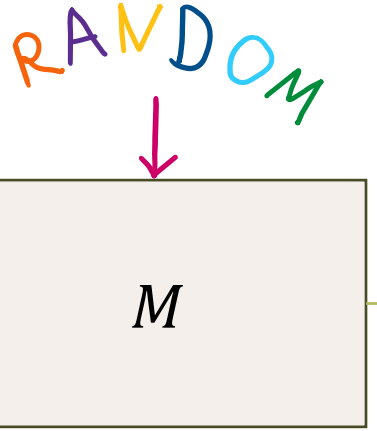


Anonymization through Randomization

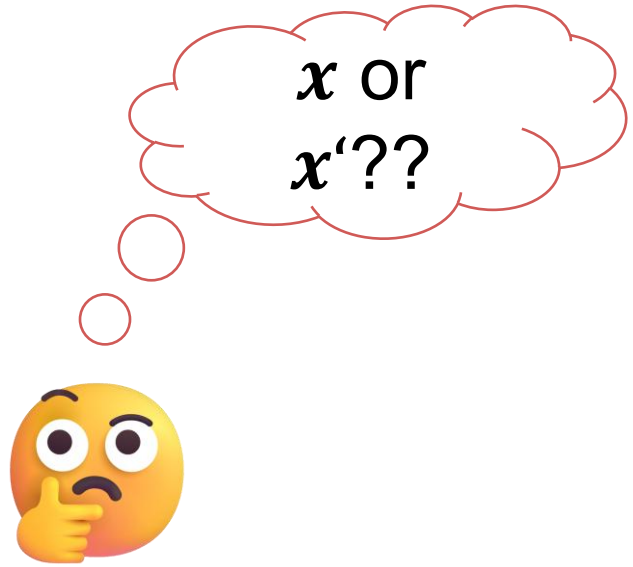


Anonymization through Randomization





$M(x')$



Mathematical Definition (neighboring)

→ Let U be some data universe

→ $\boldsymbol{x} = \{x_1, \ldots, x_n\}$ with $x_i \in U$ for all $i \in \{1, \ldots, n\}$ is a data set

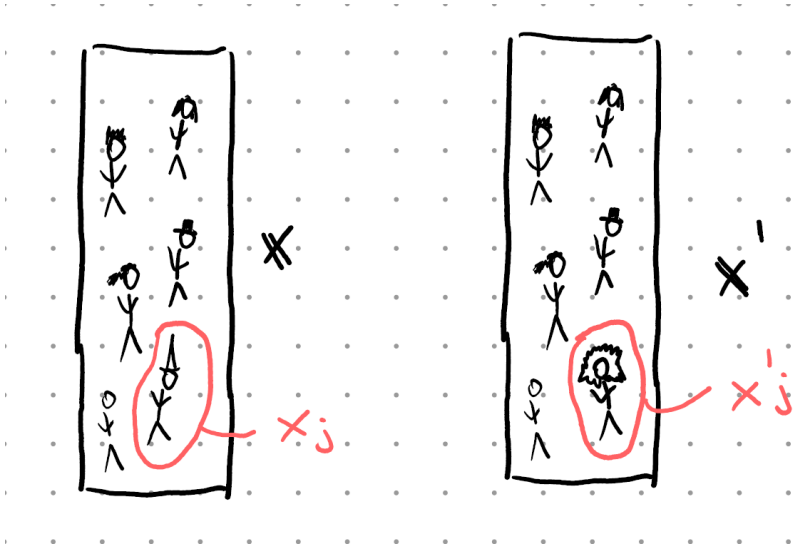
→ $\boldsymbol{x}$ and $\boldsymbol{x}' = \{x'_1, \ldots, x'_n\}$ are neighbouring, if they differ in only one entry, that is $\exists j$ such that $x_j \neq x'_j$ and $x_i = x'_i$ for all $i \neq j$

Mathematical Definition (neighboring)

→ Let U be some data universe

→ $\boldsymbol{x} = \{x_1, \ldots, x_n\}$ with $x_i \in U$ for all $i \in \{1, \ldots, n\}$ is a data set

→ $\boldsymbol{x}$ and $\boldsymbol{x'} = \{x'_1, \ldots, x'_n\}$ are neighbouring, if they differ in only one entry, that is $\exists j$ such that $x_j \neq x'_j$ and $x_i = x'_i$ for all $i \neq j$



(Pure) Differential Privacy

→Let M be a randomized algorithm (also called mechanism) which takes inputs from U^n.

M is ε – differentially private, if for all neighboring datasets $\boldsymbol{x}$ and $\boldsymbol{x}$' and any $S \subseteq \text{range}(M)$:

$$\Pr(M(\boldsymbol{x}) \in S) \leq e^{\varepsilon} \cdot \Pr(M(\boldsymbol{x}') \in S)$$

Approximate Differential Privacy

→Let M be a randomized algorithm (also called mechanism) which takes inputs from U^n.

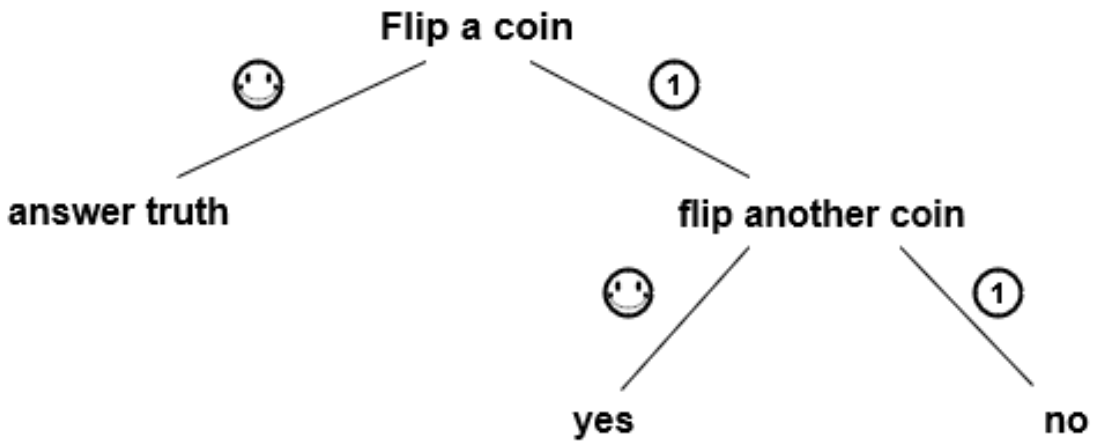
M is ε – differentially private, if for all neighboring datasets $\boldsymbol{x}$ and $\boldsymbol{x}$' and any $S \subseteq \mathrm{range}(M)$:

$$\Pr(M(\boldsymbol{x}) \in S) \leq e^{\varepsilon} \cdot \Pr(M(\boldsymbol{x}') \in S) + \delta$$

→ Other definitions: Renyi-DP, CDP, Gaussian DP…

Privacy Proof (blackboard)

→Randomized Response fulfills ln(3)-differential privacy



(Pure) Differential Privacy

→Let M be a randomized algorithm (also called mechanism) which takes inputs from U^n.

M is ε – differentially private, if for all neighboring datasets $\boldsymbol{x}$ and $\boldsymbol{x}$' and any $S \subseteq \text{range}(M)$:

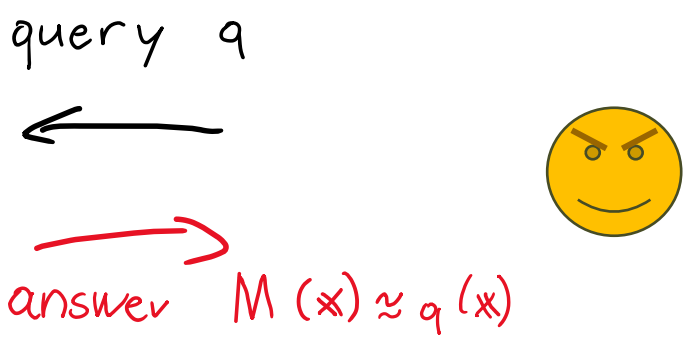
$$\Pr(M(\boldsymbol{x}) \in S) \leq e^{\varepsilon} \cdot \Pr(M(\boldsymbol{x}') \in S)$$

→Break & Exercises 1-3

Differential Privacy – Example

x

Name\likes	chocolate	coffee	pizza	illegal drugs
Anthony	0	1	0	0
Benedict	1	1	1	1
Colin	1	1	1	0
Daphne	0	0	0	0
Eloise	0	1	1	1
Franchesca	1	0	1	0
Gregory	1	0	0	0
Hyacinth	1	0	1	1
Ida	0	0	1	0
John	1	0	1	0
Mary	0	0	0	1
Noah	1	0	0	1
Peter	1	1	0	0



Laplace mechanism

→We want to answer a query q.

→We compute the true answer.

→We add random noise.

→How much noise do we need to add?

Mathematical Definition (neighboring)

→ Let U be some data universe

→ $\boldsymbol{x} = \{x_1, \ldots, x_n\}$ with $x_i \in U$ for all $i \in \{1, \ldots, n\}$ is a data set

→ $\boldsymbol{x}$ and $\boldsymbol{x}' = \{x'_1, \ldots, x'_n\}$ are neighbouring, if they differ in only one entry, that is $\exists j$ such that $x_j \neq x'_j$ and $x_i = x'_i$ for all $i \neq j$

→ We write $\boldsymbol{x} \sim \boldsymbol{x}'$

1- dimensional: Sensitivity

→ $q: U^n \rightarrow \mathbb{R}$

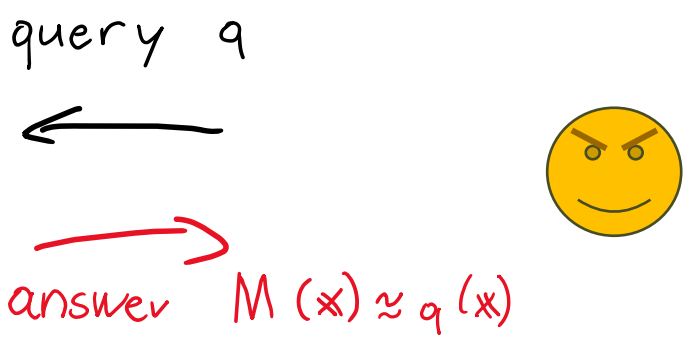
→ $\Delta q = \max_{\boldsymbol{x} \sim \boldsymbol{x}'} |q(\boldsymbol{x}) - q(\boldsymbol{x}')|$

→Δq measures how much the output of q can differ between neighboring data sets

Differential Privacy – Example

x

Name\likes	chocolate	coffee	pizza	illegal drugs
Anthony	0	1	0	0
Benedict	1	1	1	1
Colin	1	1	1	0
Daphne	0	0	0	0
Eloise	0	1	1	1
Franchesca	1	0	1	0
Gregory	1	0	0	0
Hyacinth	1	0	1	1
Ida	0	0	1	0
John	1	0	1	0
Mary	0	0	0	1
Noah	1	0	0	1
Peter	1	1	0	0

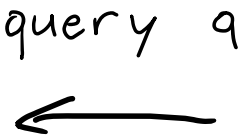


Differential Privacy – Example

x

Name\likes	chocolate	coffee	pizza	illegal drugs
Anthony	0	1	0	0
Benedict	1	1	1	1
Colin	1	1	1	0
Daphne	0	0	0	0
Eloise	0	1	1	1
Franchesca	1	0	1	0
Gregory	1	0	0	0
Hyacinth	1	0	1	1
Ida	0	0	1	0
John	1	0	1	0
Mary	0	0	0	1
Noah	1	0	0	1
Peter	1	1	0	0

query q



answer $M(x) \approx q(x)$

q... How many people like illegal drugs?

$q(x) = 5$

SDU University of Southern Denmark

Differential Privacy – Example

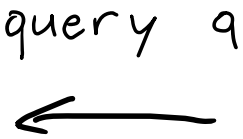
x

Name\likes	chocolate	coffee	pizza	illegal drugs
Anthony	0	1	0	0
Benedict	1	1	1	1
Colin	1	1	1	0
Daphne	0	0	0	0
Eloise	0	1	1	1
Franchesca	1	0	1	0
Gregory	1	0	0	0
Hyacinth	1	0	1	1
Ida	0	0	1	0
John	1	0	1	0
Mary	0	0	0	1
Noah	1	0	0	1
Peter	1	1	0	0

q... How many people like illegal drugs?

$q(x) = 5$

query q



answer $M(x) \approx q(x)$

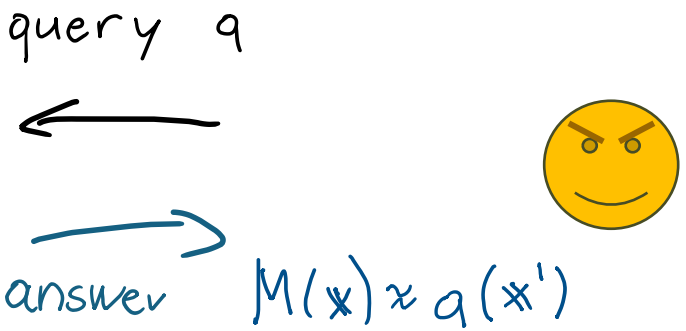
Differential Privacy – Example

x'

Name\likes	chocolate	coffee	pizza	illegal drugs
Anthony	0	1	0	0
Benedict	1	1	1	1
Colin	1	1	1	0
Daphne	0	0	0	0
Elizabeth	1	0	1	0
Franchesca	1	0	1	0
Gregory	1	0	0	0
Hyacinth	1	0	1	1
Ida	0	0	1	0
John	1	0	1	0
Mary	0	0	0	1
Noah	1	0	0	1
Peter	1	1	0	0

q... How many people like illegal drugs?

$q(x') = 4$



SDU University of Southern Denmark

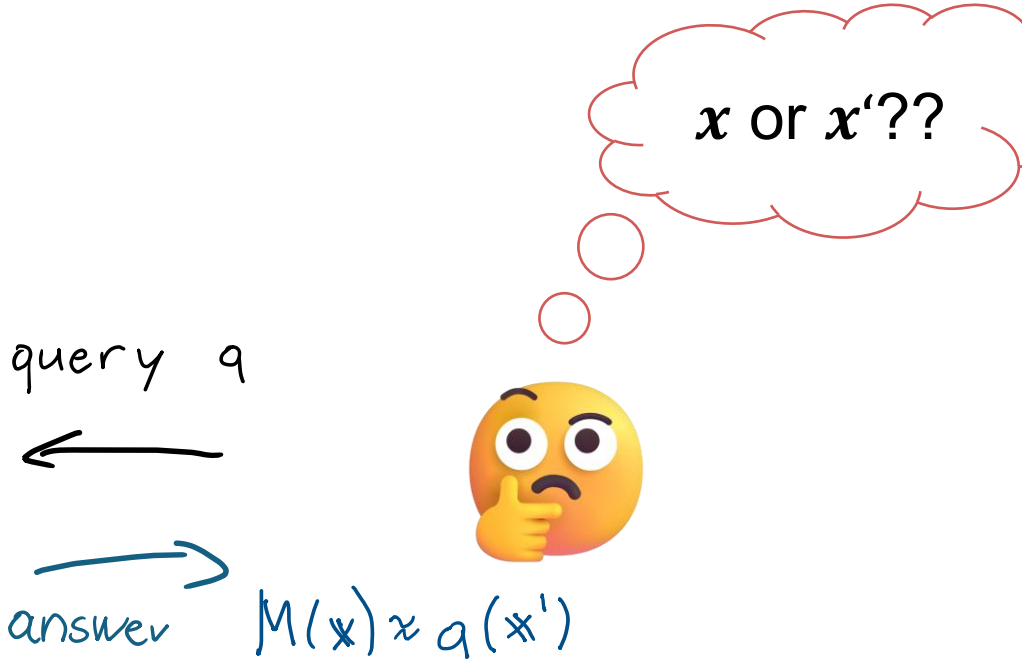
Differential Privacy – Example

x'

Name\likes	chocolate	coffee	pizza	illegal drugs
Anthony	0	1	0	0
Benedict	1	1	1	1
Colin	1	1	1	0
Daphne	0	0	0	0
Elizabeth	1	0	1	0
Franchesca	1	0	1	0
Gregory	1	0	0	0
Hyacinth	1	0	1	1
Ida	0	0	1	0
John	1	0	1	0
Mary	0	0	0	1
Noah	1	0	0	1
Peter	1	1	0	0

q... How many people like illegal drugs?

$q(x') = 4$



1- dimensional: Sensitivity

→ $q: U^n \rightarrow \mathbb{R}$

→ $\Delta q = \max_{\boldsymbol{x} \sim \boldsymbol{x}'} |q(\boldsymbol{x}) - q(\boldsymbol{x}')|$

→ Δq measures how much the output of q can differ between neighboring data sets

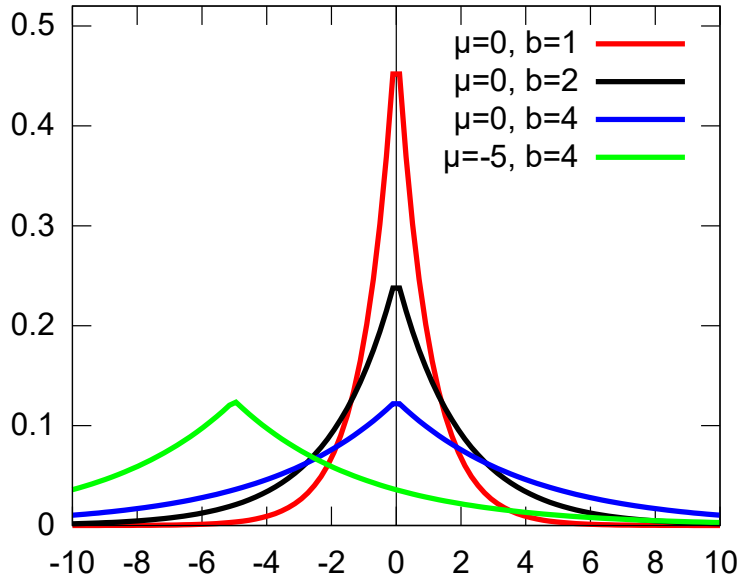
→ In the Laplace mechanism, the standard deviation of the noise is $\Theta\left(\frac{\Delta q}{\varepsilon}\right)$

Laplace Distribution

→$\mathrm{Lap}(\mu, b)$ is the Laplace distribution with mean μ and scale b

→Its density function is given by

$$p(y) = \frac{1}{2b} e^{\frac{|y-\mu|}{b}}$$



1-dimensional Laplace mechanism [1]

→Let $q: U^n \rightarrow \mathbb{R}$ with sensitivity Δq

→The algorithm which takes q, $\boldsymbol{x}$ and ε as inputs and outputs

$$M(\boldsymbol{x}) = q(\boldsymbol{x}) + Y$$

with Y drawn from $\mathrm{Lap}\left(0, \frac{\Delta q}{\varepsilon}\right)$ is called the Laplace mechanism. It preserves ε-differential privacy.

[1] Dwork, McSherry, Nissim, and Smith, ”Calibrating noise to sensitivity in private data analysis” , TCC 2006

Privacy Proof (blackboard)

(α, β)-accuracy

→We say a randomized algorithm $M: U^n \rightarrow \mathbb{R}$ is (α, β)-accurate for a function q, if for all inputs $\boldsymbol{x}$,

$$\Pr(|M(\boldsymbol{x}) - q(\boldsymbol{x})| > \alpha) \leq \beta$$

1-dimensional Laplace mechanism (accuracy)

→ Fact: If $Y \sim \mathrm{Lap}(0, b)$, then $\Pr(|Y| \geq t \cdot b) = e^{-t}$

→The algorithm which takes q, $\boldsymbol{x}$ and ε as inputs and outputs

$$M(\boldsymbol{x}) = q(\boldsymbol{x}) + Y$$

with Y drawn from $\mathrm{Lap}\left(0, \frac{\Delta q}{\varepsilon}\right)$ is called the Laplace mechanism. It preserves ε-differential privacy.

→ The 1-dimensinoal Laplace mechanism is (α, β)-accurate for $\alpha = \frac{\Delta q}{\varepsilon} \ln\left(\frac{1}{\beta}\right)$

Differential Privacy – Many Queries

x

Name\likes	chocolate	coffee	pizza	illegal drugs
Anthony	0	1	0	0
Benedict	1	1	1	1
Colin	1	1	1	0
Daphne	0	0	0	0
Eloise	0	1	1	1
Franchesca	1	0	1	0
Gregory	1	0	0	0
Hyacinth	1	0	1	1
Ida	0	0	1	0
John	1	0	1	0
Mary	0	0	0	1
Noah	1	0	0	1
Peter	1	1	0	0

q_1... How many people like illegal drugs?

q_2... How many people like pizza?

k-dimensional Laplace mechanism [1]

→$q: U^n \to \mathbb{R}^k$

→$\Delta_1 q = \max_{\boldsymbol{x}\sim\boldsymbol{x}'} \|q(\boldsymbol{x}) - q(\boldsymbol{x}')\|_1 = \max_{\boldsymbol{x}\sim\boldsymbol{x}'} \sum_{i=1}^{k} |q_i(\boldsymbol{x}) - q_i(\boldsymbol{x}')|$

→$\Delta_1 q$ is called the L_1- sensitivity

→The algorithm which takes q, $\boldsymbol{x}$ and ε as inputs and outputs

$$M(\boldsymbol{x}) = \begin{pmatrix} q_1(\boldsymbol{x}) + Y_1 \\ \dots \\ q_k(\boldsymbol{x}) + Y_k \end{pmatrix}$$

with Y_i drawn from $\mathrm{Lap}\left(0, \frac{\Delta_1 q}{\varepsilon}\right)$ is called the Laplace mechanism. It preserves ε-differential privacy.

[1] Dwork, McSherry, Nissim, and Smith, ”Calibrating noise to sensitivity in private data analysis” , TCC 2006

Differential Privacy – Many Queries

x

Name\likes	chocolate	coffee	pizza	illegal drugs
Anthony	0	1	0	0
Benedict	1	1	1	1
Colin	1	1	1	0
Daphne	0	0	0	0
Eloise	0	1	1	1
Franchesca	1	0	1	0
Gregory	1	0	0	0
Hyacinth	1	0	1	1
Ida	0	0	1	0
John	1	0	1	0
Mary	0	0	0	1
Noah	1	0	0	1
Peter	1	1	0	0

q_1... How many people like illegal drugs?

q_2... How many people like pizza?

Accuracy of *k*-dimensional Laplace mechanism

→We say a randomized algorithm $M: U^n \to \mathbb{R}^k$ is (α, β)-accurate for a function q, if for all inputs $\boldsymbol{x}$,

$$\Pr\left(\max_{i=1..k} |M(\boldsymbol{x})_i - q_i(\boldsymbol{x})| > \alpha\right) \leq \beta$$

→ The k-dimensional Laplace mechanism is (α, β)-accurate for $\alpha = \frac{\Delta_1 q}{\varepsilon} \ln\left(\frac{k}{\beta}\right)$

Union Bound

→For any set of events $A_1, \ldots, A_k$ we have

$$\Pr\left(\bigcup_{i=1}^{k} A_i\right) \leq \sum_{i=1}^{k} \Pr(A_i)$$

Simple Composition

→Let $M_1 : U^n \to \mathrm{range}(M_1)$ be ε_1- differentially private and $M_2 : U^n \to \mathrm{range}(M_2)$ be ε_2- differentially private. Further, let the randomness of M_1 and M_2 be independent.

→Then M defined by $M(x) = (M_1(x), M_2(x))$ is $(\varepsilon_1 + \varepsilon_2)$- differentially private.

→Something stronger is true: M_2 can depend on the output of M_1 (but the randomness still needs to be independent).

Postprocessing

→Let $M: U^n \rightarrow \text{range}(M)$ be ε- differentially private and $B: \text{range}(M) \rightarrow \text{range}(B)$ be any algorithm.

→Then $B \circ M$ is ε- differentially private.

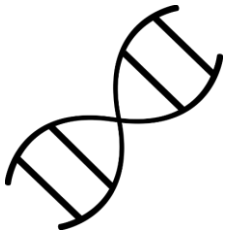
→Exercises 4 - 7. Lunch at 12h.

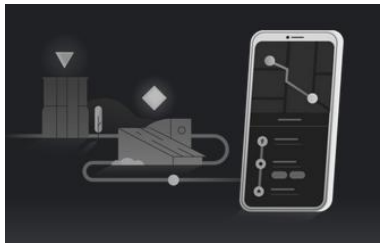
Part 2: Differential Privacy for Strings

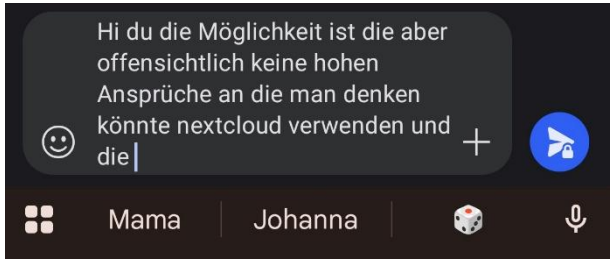
Strings

→ A *string* S is a sequence of symbols from a predefined alphabet ∑.

→ $S = S[1]S[2] \dots S[n]$



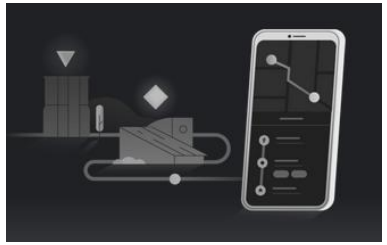




What is neighboring?

→ Option 1: Two strings S and S' are neighboring, if they differ in one position.

→ $S = \text{abbbbcabaaab}$
→ $S' = \text{abbbbaabaaab}$



→ Differentially private approximate pattern matching, ITCS 2024

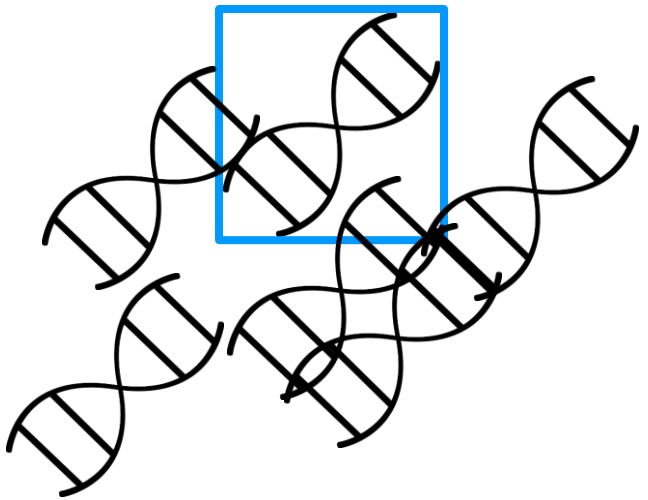
→ Today: Rank / Select + some exercises on string problems

What is neighboring?

→ Option 2: Two *collections of strings* D and D' are neighboring, if they differ in one string.

→ $D = (S_1, S_2, \dots, S_j, \dots, S_n)$

→ $D' = (S_1, S_2, \dots, S'_j, \dots, S_n)$



→ Differentially Private Substring and Document Counting. PODS 2025 → later

Part 2.1: Position privacy

What is neighboring?

→ Option 1: Two strings S and S' are neighboring, if they differ in one position.

→ $S = \text{abbbbcabaaab}$
→ $S' = \text{abbbbaabaaab}$

Rank / Select

→$\mathrm{rank}_{\mathrm{a}}(j) = |\{i \in \{1, \dots, j\}: \mathrm{S}[i] = \mathrm{a}\}|$

→$\mathrm{select}_{\mathrm{a}}(x) = \min\{j \in \{1, \dots, n\}: \mathrm{rank}_{\mathrm{a}}(j) = x\}$

→ S = abbbbcabaaab
→ $\mathrm{rank}_{\mathrm{a}}(8) = 2$
→ $\mathrm{select}_{\mathrm{a}}(2) = 7$

Differentially Private Rank

→ $\mathrm{rank}_a(j) = |\{i \in \{1, \dots, j\} : S[i] = a\}|$

→ S = abbbbcabaaab
→ $\mathrm{rank}_a(8) = 2$

→ What is the sensitivity of $\mathrm{rank}_a(j)$?

→ What is the L_1- sensitivity of $(\mathrm{rank}_a(j))_{j \in \{1,\dots,n\}}$?

→ What is the resulting (asymptotic) error of the Laplace mechanism?

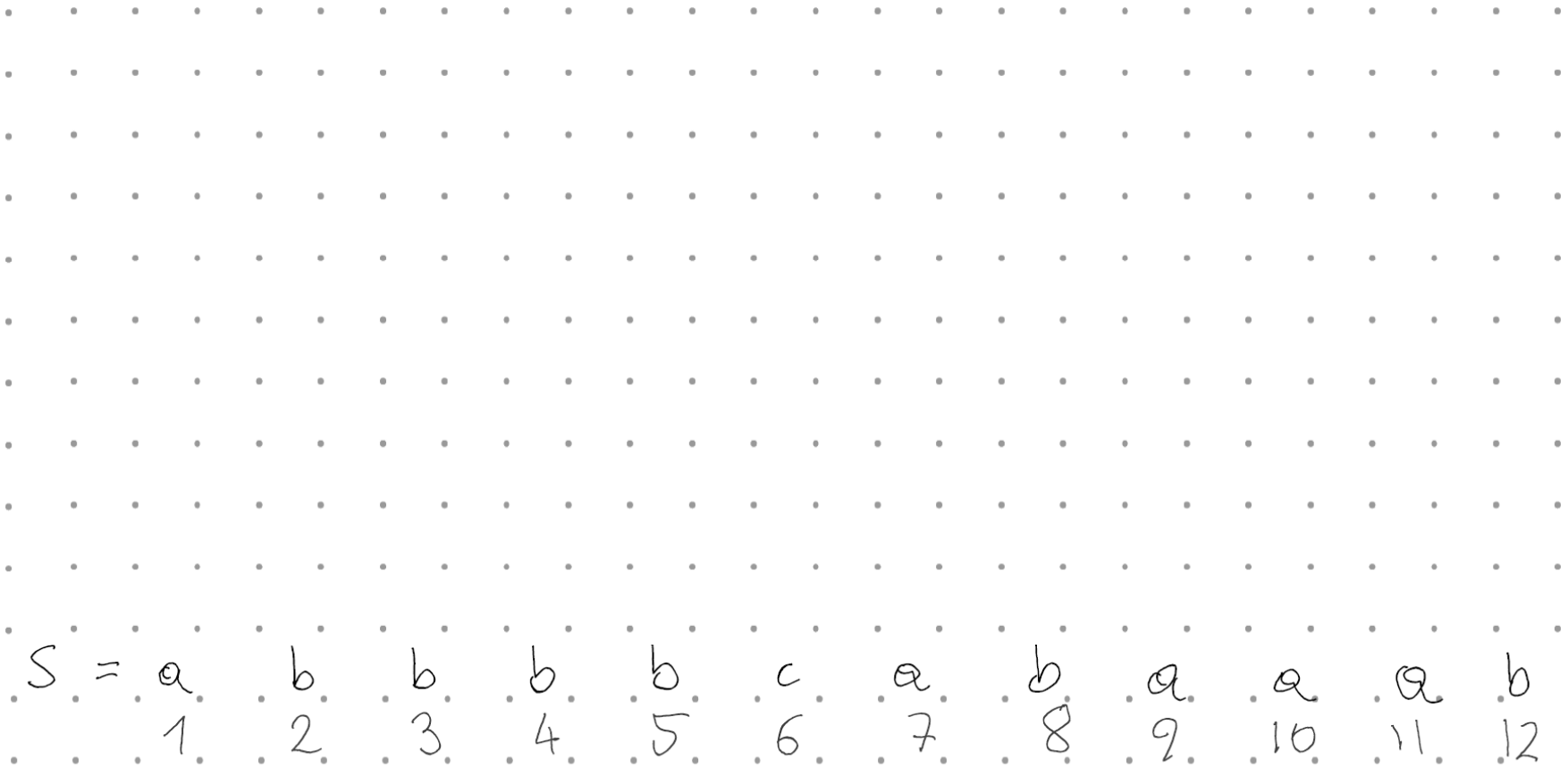
Binary Tree Mechanism

→ developed for differential private continual counting

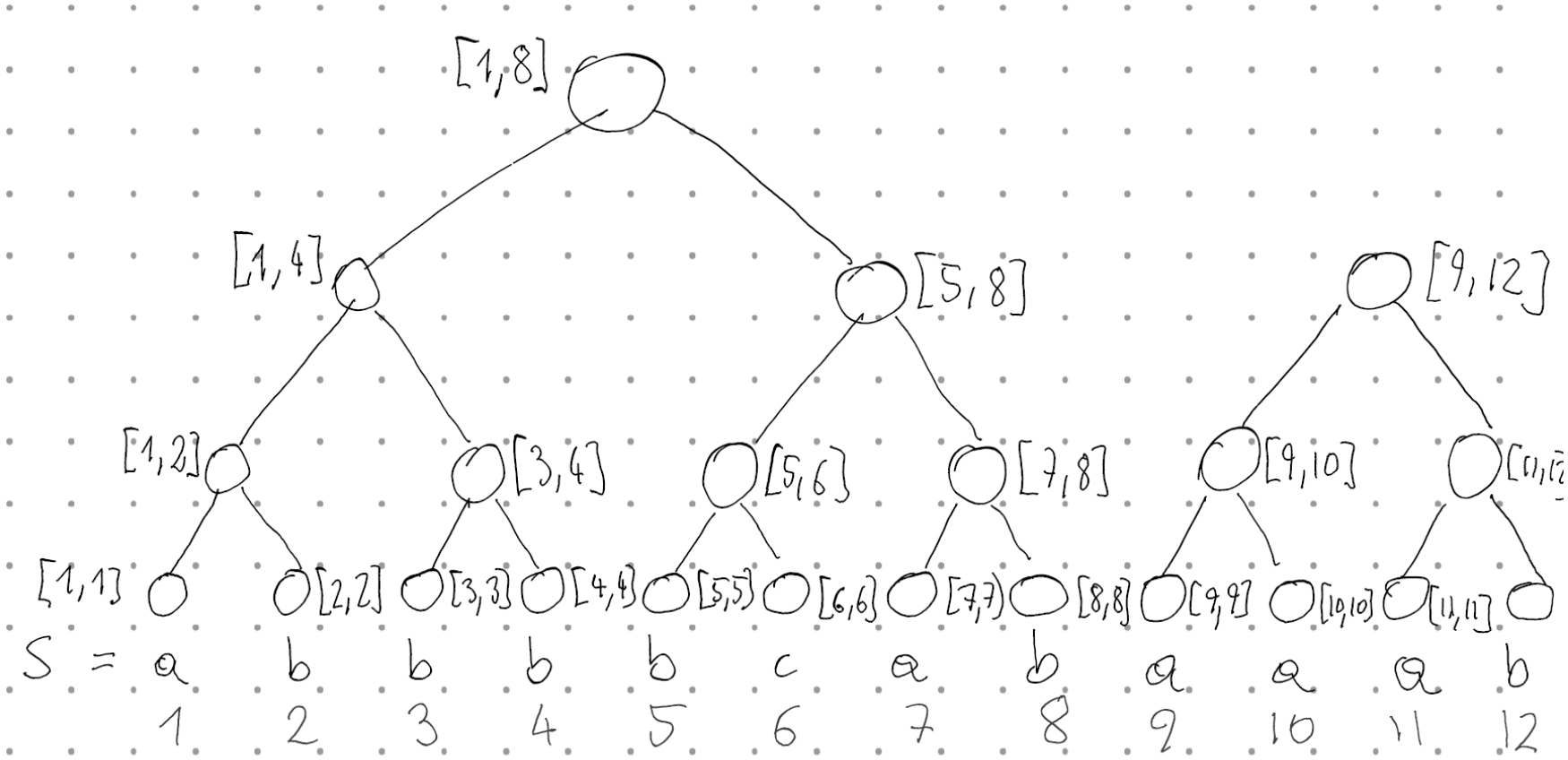
→ can also be used for dp interval counting, computing quantiles, range queries

→ and for rank / select!

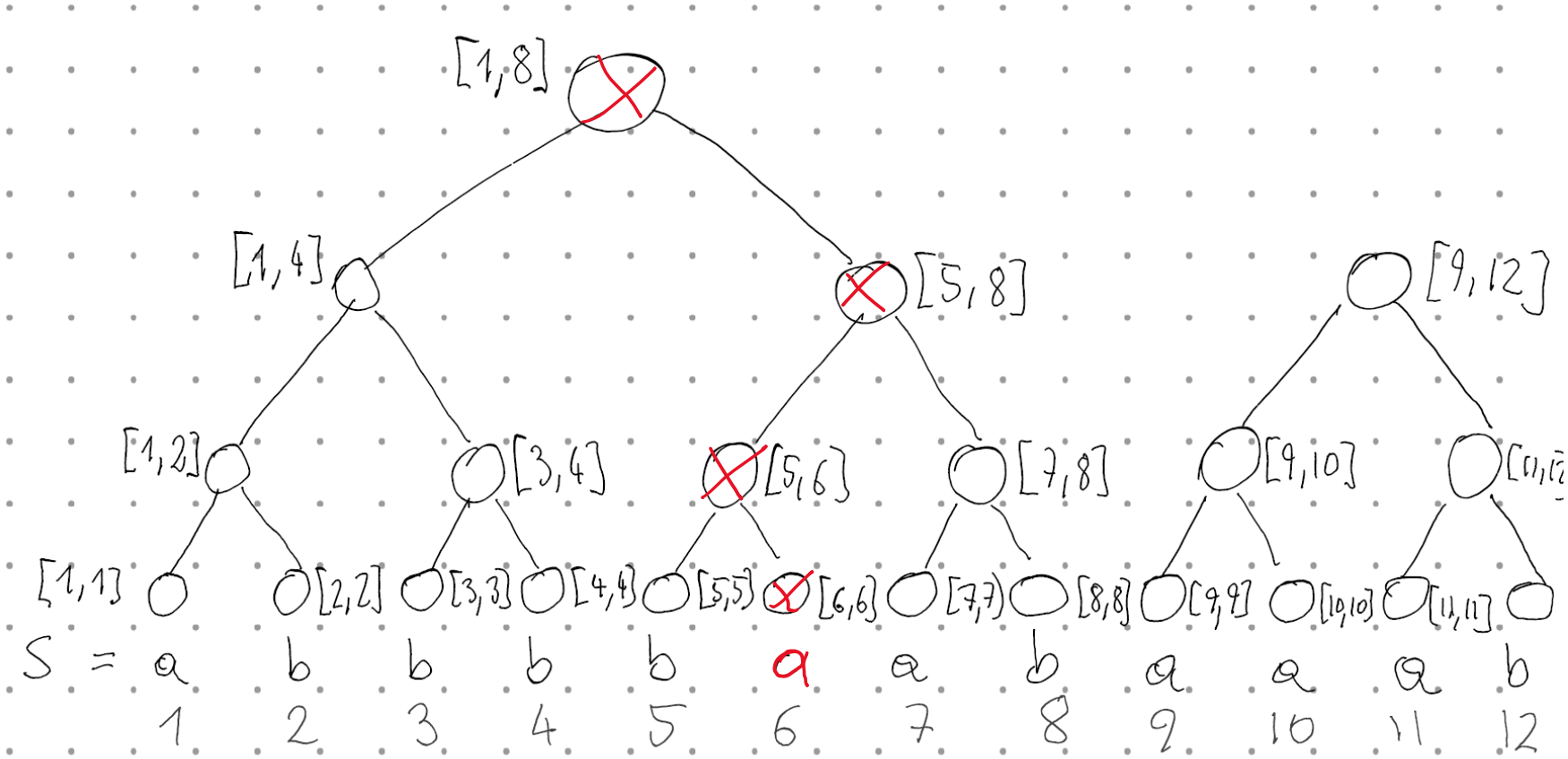
Binary Tree Mechanism for Rank



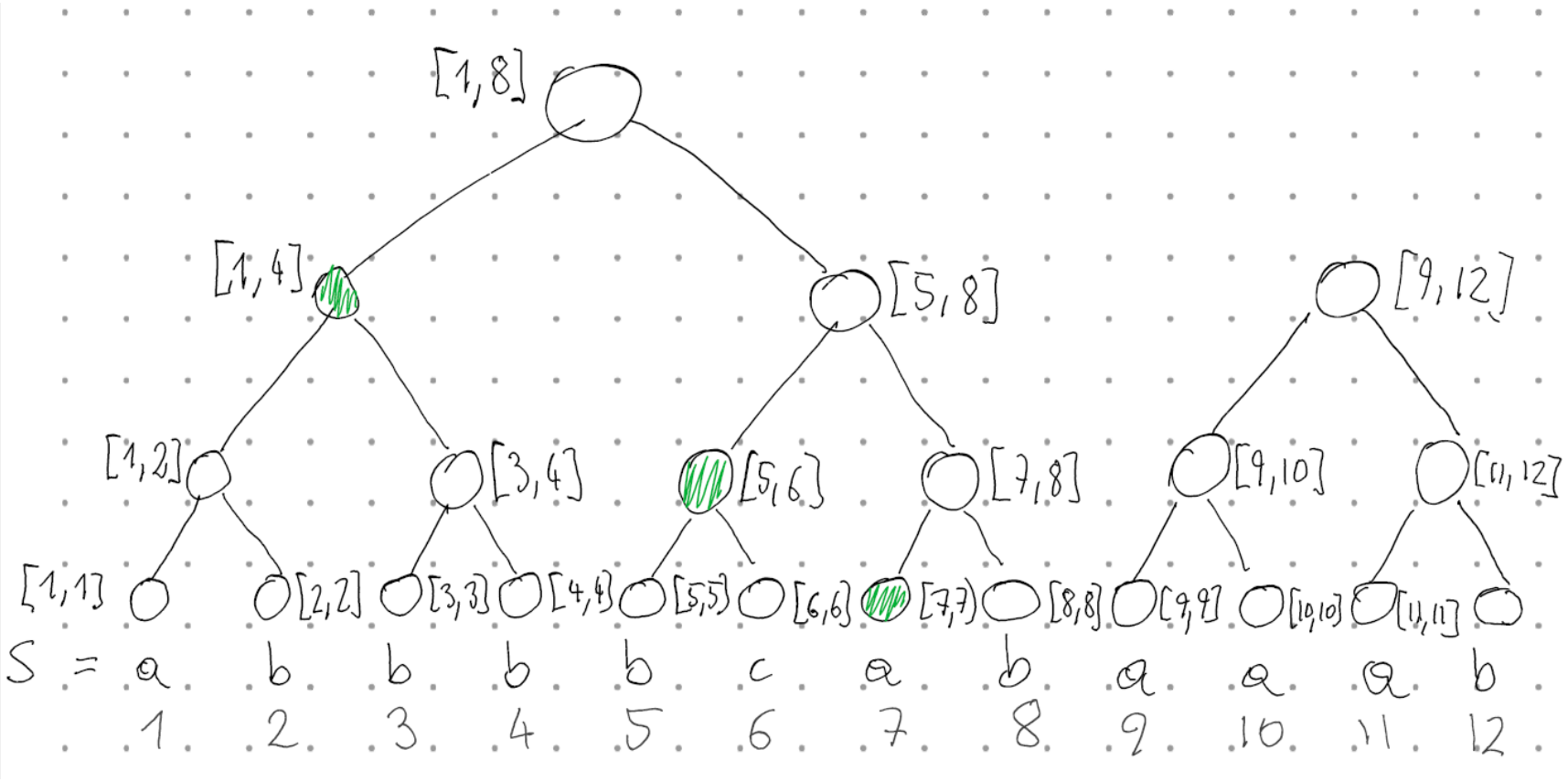
Binary Tree Mechanism for Rank



Binary Tree Mechanism for Rank



Binary Tree Mechanism for Rank



Accuracy Analysis (blackboard)

Accuracy of *k*-dimensional Laplace mechanism

→We say a randomized algorithm $M: U^n \to \mathbb{R}^k$ is (α, β)-accurate for a function q, if for all inputs $\boldsymbol{x}$,

$$\Pr\left(\max_{i=1..k} |M(\boldsymbol{x})_i - q_i(\boldsymbol{x})| > \alpha\right) \leq \beta$$

→ The k-dimensional Laplace mechanism is (α, β)-accurate for $\alpha = \frac{\Delta_1 q}{\varepsilon} \ln\left(\frac{k}{\beta}\right)$

Differentially Private Rank

→ $\mathrm{rank}_a(j) = |\{i \in \{1, \dots, j\} : S[i] = a\}|$

→ S = abbbbcabaaab

→ $\mathrm{rank}_a(8) = 2$

→ What if we want to compute a rank data structure for every letter?

Open Problem 1

→ Binary interval counting with ε- differential privacy: $O(\log^2 n)$ vs $\Omega(\log n)$

→Exercises 8 – 10 & break

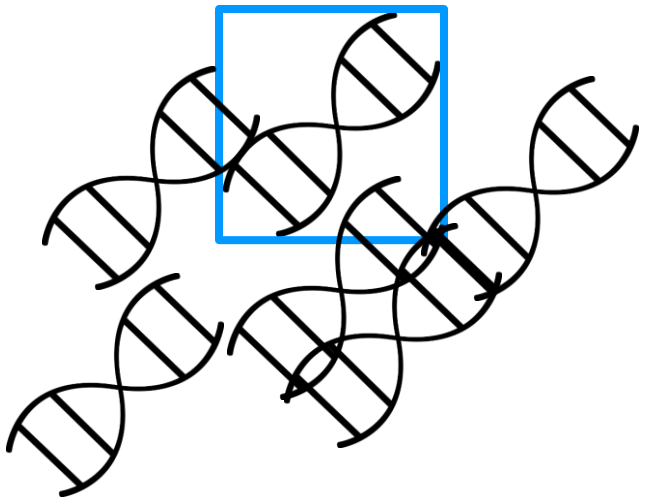
Part 2.2: Document privacy

What is neighboring?

→ Option 2: Two *collections of strings* D and D' are neighboring, if they differ in one string.

→ $D = (S_1, S_2, \dots, S_j, \dots, S_n)$

→ $D' = (S_1, S_2, \dots, S'_j, \dots, S_n)$

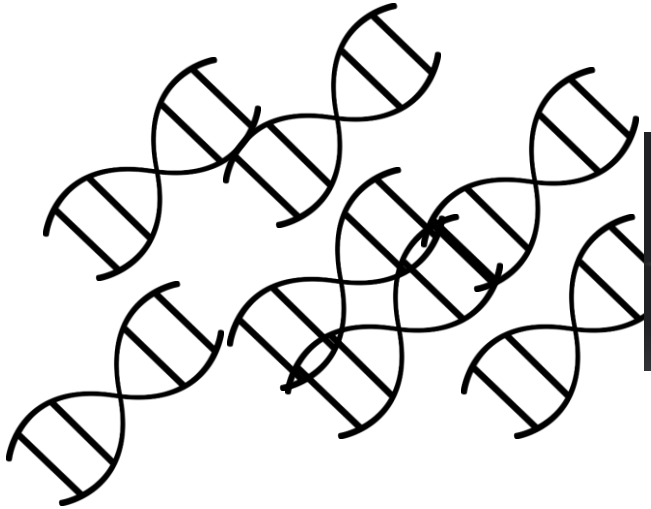


→ Differentially Private Substring and Document Counting. PODS 2025 → now

Substring and Document Counting

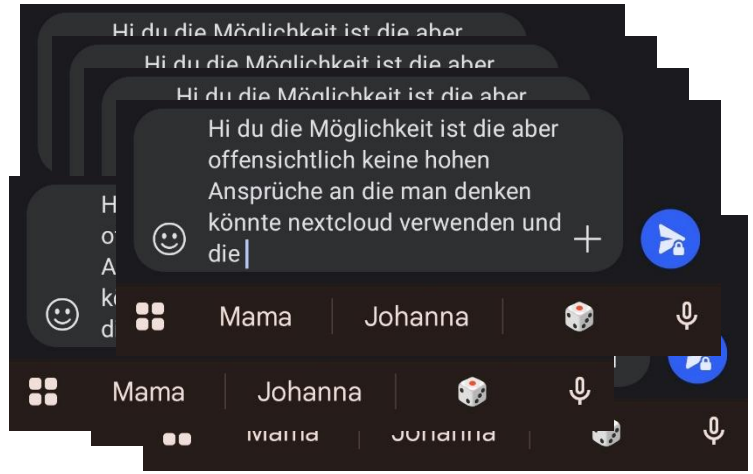
→ **Substring Count**: How often does a given pattern appear in total?

Genomes



Transit data

Texting

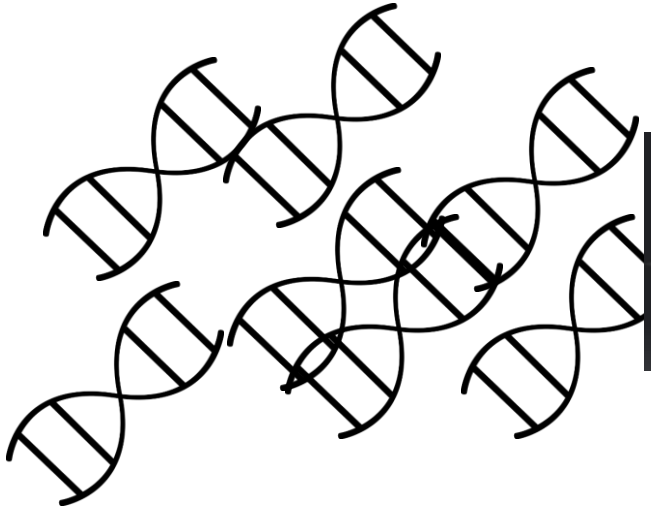


SDU
University of Southern Denmark

Substring and Document Counting

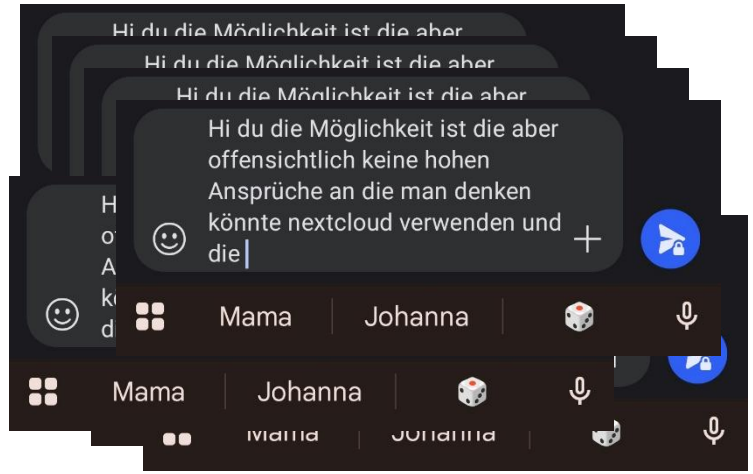
→ **Substring Count**: How often does a given pattern appear in total?

Genomes



Transit data

Texting



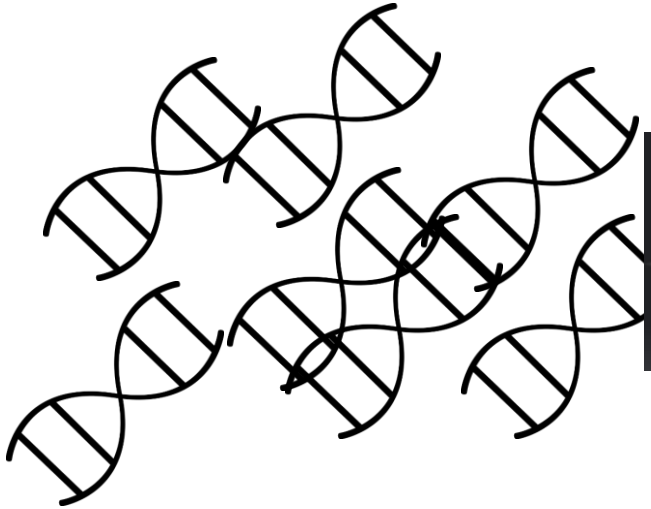
”die” appears 4 times in this text - contributes 4 towards substring count

SDU
University of Southern Denmark

Substring and Document Counting

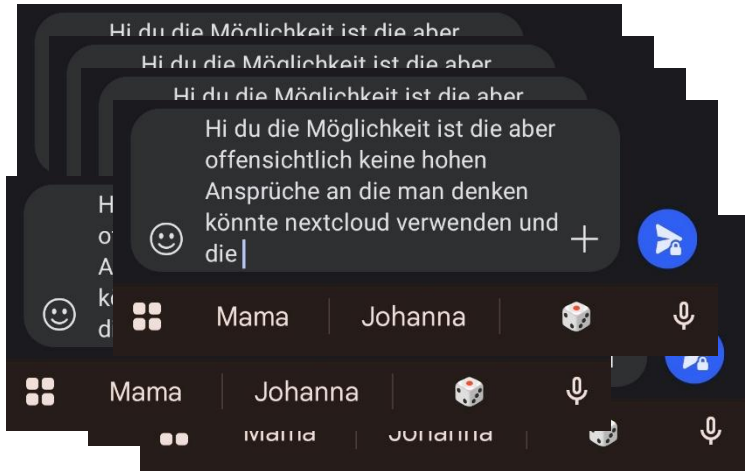
→ **Substring Count**: How often does a given pattern appear in total?
→ **Document Count**: How many documents contain a given pattern ?

Genomes



Transit data

Texting

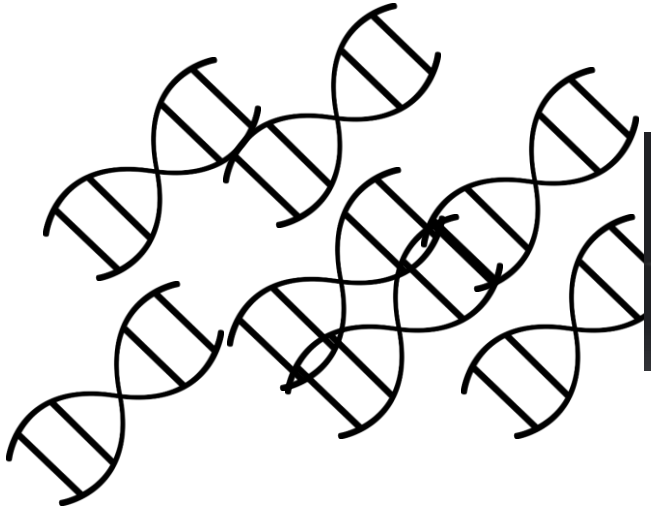


SDU
University of Southern Denmark

Substring and Document Counting

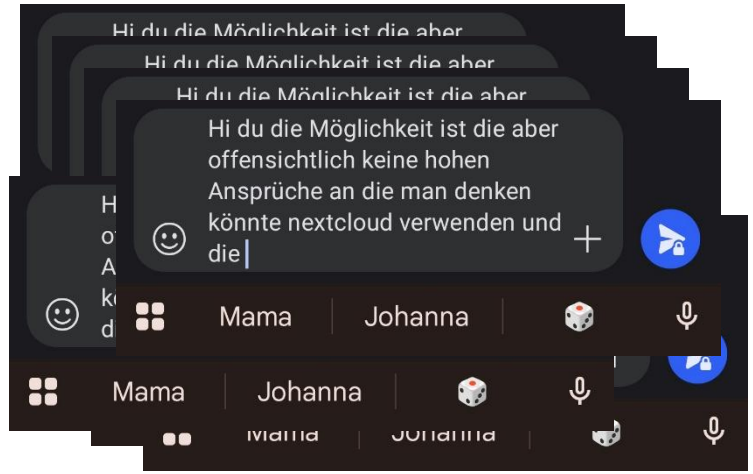
→ **Substring Count**: How often does a given pattern appear in total?
→ **Document Count**: How many documents contain a given pattern ?

Genomes



Transit data

Texting



”die” appears 4 times in this text – contributes 1 towards document count

SDU
University of Southern Denmark

Substring Count (blackboard)

Substring Count Results

→ Lower bound $\Omega(l)$ holds even for one pattern

→ We get a data structure of size $\mathrm{O}(nl^2)$, query time $O(|P|)$, and max error $\mathrm{O}(l\ \mathrm{polylog}(nl|\Sigma|))$ whp

Document Count (blackboard)

Document Count Results

→ Lower bound $\Omega(l)$ holds for all patterns

→ We get a data structure of size $\mathrm{O}(nl^2)$, query time $O(|P|)$, and max error $\mathrm{O}(l\ \mathrm{polylog}(nl|\Sigma|))$ whp

Main ideas for our data structure

→ When changing S to S‘, the affected patterns are not independent

→ Insight 1: A pattern $P = P_1P_2$ has a lower count than both P_1 and P_2
→ used for filtering non-frequent patterns

→ Insight 2: All substrings of S are prefixes of suffixes of S. Hence, they lie on at most l many paths in a trie
→ used for computing counts on paths in a smart way

Two Main Steps

→ Step 1: Reduce universe size by filtering

→ Step 2: Build a dp data structure on remaining strings

Step 1: Candidate set

→ We build a candidate set such that any string not in the candidate set has a count at most $\mathrm{O}(l\ \mathrm{polylog}(nl|\Sigma|))$ whp

→ For this, we use Insight 1: A pattern $P = P_1P_2$ has a lower count than both P_1 and P_2

Step 1: Candidate set

Example:

$D = \{\text{aaaa}, \text{abe}, \text{absab}, \text{babe}, \text{bee}, \text{bees}\}$

$\boldsymbol{P}_1 = \{\text{a}, \text{b}, \text{e}, \text{s}\}$

→ We start by estimating frequent letters

→ There are $|\Sigma|$ letters

→ Substring count of all letters has sensitivity l

→ Laplace mechanism has error $O(\, l \log |\Sigma|)$

→ We keep everything with a count at least $\tau = \Theta(\, l \log |\Sigma|)$ in a set $\boldsymbol{P}_1$

→ We make sure $|\boldsymbol{P}_1| \leq nl$

Step 1: Candidate set

Example:

$D = \{\text{aaaa}, \text{abe}, \text{absab}, \text{babe}, \text{bee}, \text{bees}\}$

$\boldsymbol{P}_1 = \{\text{a}, \text{b}, \text{e}, \text{s}\}$

$\boldsymbol{P}_1\boldsymbol{P}_1 = \{\text{aa}, \text{ab}, \text{ae}, \text{as}, \text{ba}, \text{bb}, \text{be}, \text{bs}, \text{ea}, \text{eb}, \text{ee}, \text{es}, \text{sa}, \text{sb}, \text{se}, \text{ss}\}$

$\boldsymbol{P}_2 = \{\text{aa}, \text{ab}, \text{ba}, \text{be}, \text{bs}, \text{ee}, \text{sa}\}$

→ We iteratively double the lengths of substrings we consider

→ Given $\boldsymbol{P_1}$, we compute the count of all strings in $\boldsymbol{P_1P_1}$

→ Substring count of all patterns of fixed length has sensitivity l

→ Laplace mechanism has error $O(\, l \log |\boldsymbol{P_1P_1}|) = O(\, l \log(nl))$

→ We keep everything with a count at least $\tau = \Theta(\, l \log(nl))$ in a set $\boldsymbol{P}_2$

→ We make sure $|\boldsymbol{P}_2| \leq nl$

Step 1: Candidate set

Example:

$D = \{\text{aaaa}, \text{abe}, \text{absab}, \text{babe}, \text{bee}, \text{bees}\}$

$\boldsymbol{P}_1 = \{\text{a}, \text{b}, \text{e}, \text{s}\}$

$\boldsymbol{P}_2 = \{\text{aa}, \text{ab}, \text{ba}, \text{be}, \text{bs}, \text{ee}, \text{sa}\}$

$\boldsymbol{P}_4 = \{\text{aaaa}, \text{absa}, \text{babe}, \text{bsab}, \text{aaab}\}$

→ We iteratively double the lengths of substrings we consider

→Given $\boldsymbol{P}_{2^i}$, we compute the count of all strings in $\boldsymbol{P}_{2^i}\boldsymbol{P}_{2^i}$

→ Substring count of all patterns of fixed length has sensitivity l

→ Laplace mechanism has error $O(\, l \log |\boldsymbol{P}_{2^i}\boldsymbol{P}_{2^i}|) = O(\, l \log(nl))$

→ We keep everything with a count at least $\tau = \Theta(\, l \log(nl))$ in a set $\boldsymbol{P}_{2^{i+1}}$

→ We make sure $\left|\boldsymbol{P}_{2^{i+1}}\right| \leq nl$

Step 1: Candidate set

Example:

$D = \{\text{aaaa, abe, absab, babe, bee, bees}\}$

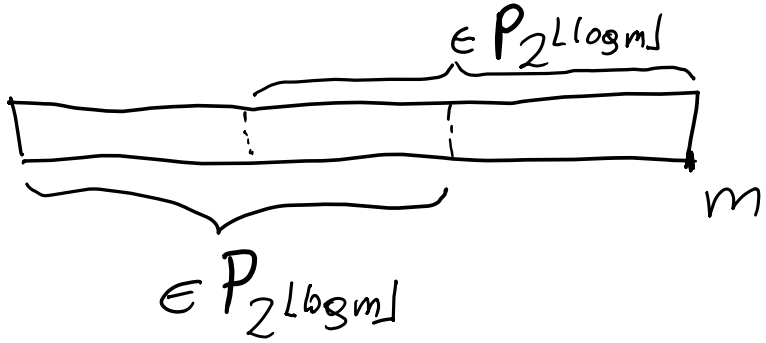
$\boldsymbol{P}_1 = \{\text{a, b, e, s}\}$

$\boldsymbol{P}_2 = \{\text{aa, ab, ba, be, bs, ee, sa}\}$

$\boldsymbol{P}_4 = \{\text{aaaa, absa, babe, bsab, aaab}\}$

→ What about strings which have a length which is not a power of two?

→ They have an overlapping prefix and suffix of a length of power of two



$\boldsymbol{C}_3 = \{\text{aaa, aab, aba, abe, abs, baa, bab, be, bsa, eee, saa, sab}\}$

$\boldsymbol{C}_5 = \{\text{aaaaa, aaaab, absab}\}$

→ C_m are all strings obtained by an overlapping prefix and suffix from $\boldsymbol{P}_{2^{\lfloor \log m \rfloor}}$

Step 1: Candidate set

→ The candidate set C is the union of all $\boldsymbol{P}_{2^i}$ and C_m

→ $|C| \leq \sum_{m=1}^{l} (nl)^2 \leq n^2 l^3$

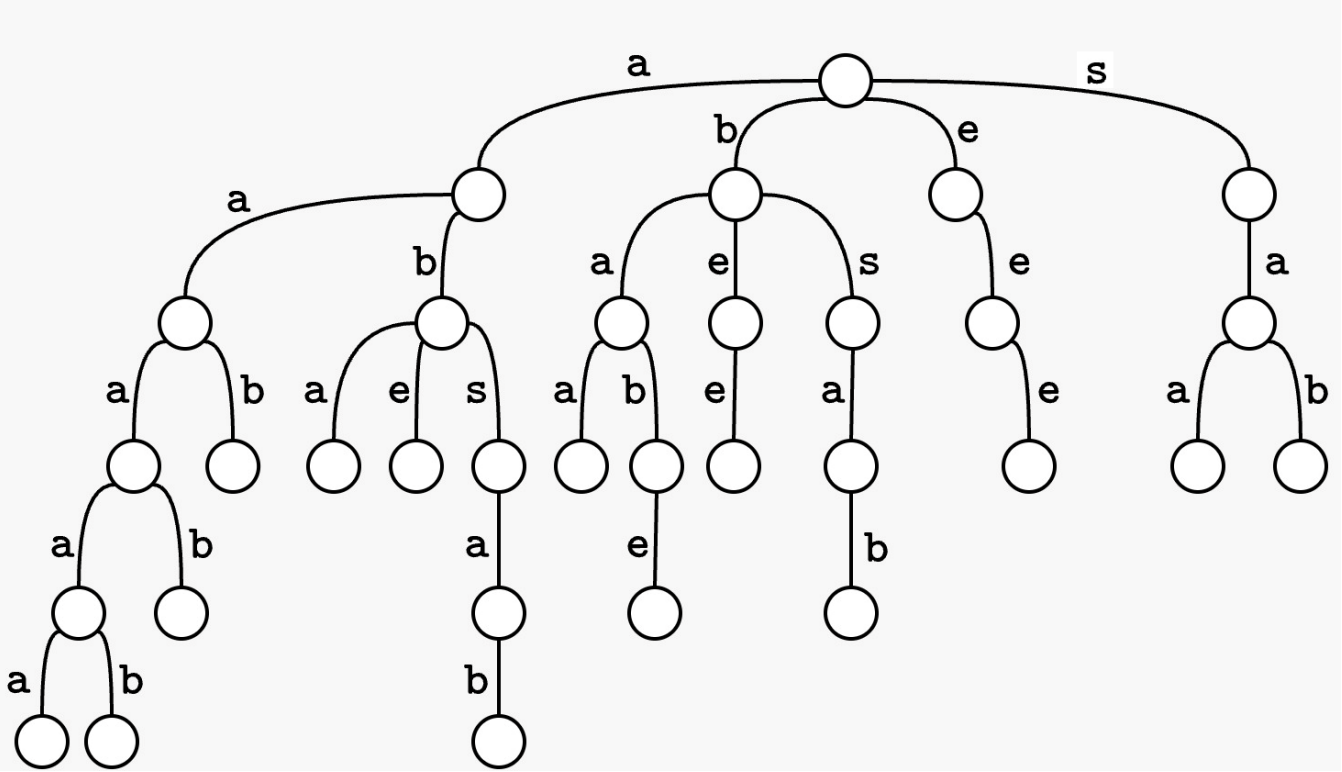
→Any string not in the candidate set has a count at most $\mathrm{O}(l \ \mathrm{polylog}(nl|\Sigma|))$ whp

Two Main Steps

✓ Step 1: Reduce universe size by filtering

→ Step 2: Build a dp data structure on remaining strings

Step 2: Trie data structure on C



Example:

$D = \{\text{aaaa, abe, absab, babe, bee, bees}\}$

$\boldsymbol{P}_1 = \{\text{a, b, e, s}\}$

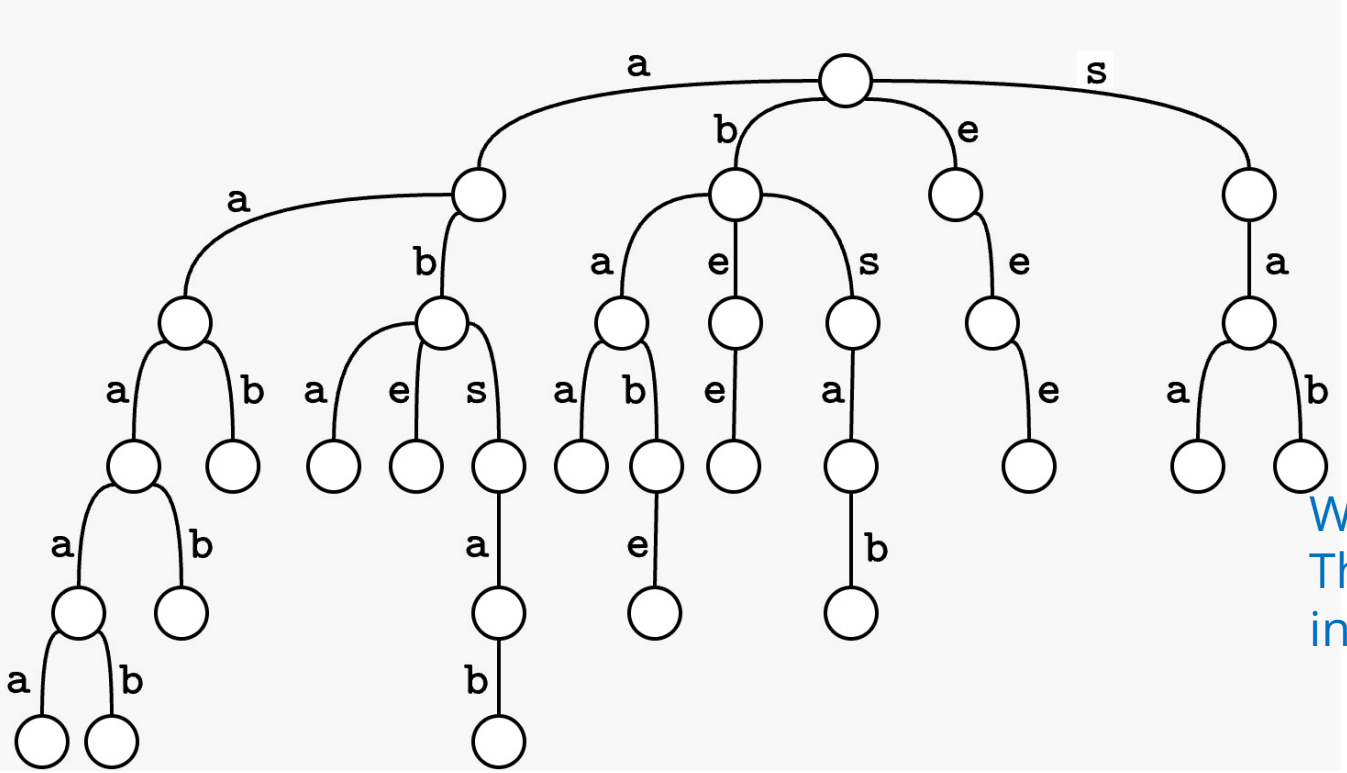
$\boldsymbol{P}_2 = \{\text{aa, ab, ba, be, bs, ee, sa}\}$

$\boldsymbol{P}_4 = \{\text{aaaa, absa, babe, bsab, aaab}\}$

$\boldsymbol{C}_3 = \{\text{aaa, aab, aba, abe, abs, baa, bab, be, bsa, eee, saa, sab}\}$

$\boldsymbol{C}_5 = \{\text{aaaaa, aaaab, absab}\}$

Step 2: Trie data structure on C



Example:

$D = \{\text{aaaa, abe, absab, babe, bee, bees}\}$

$\boldsymbol{P}_1 = \{\text{a, b, e, s}\}$

$\boldsymbol{P}_2 = \{\text{aa, ab, ba, be, bs, ee, sa}\}$

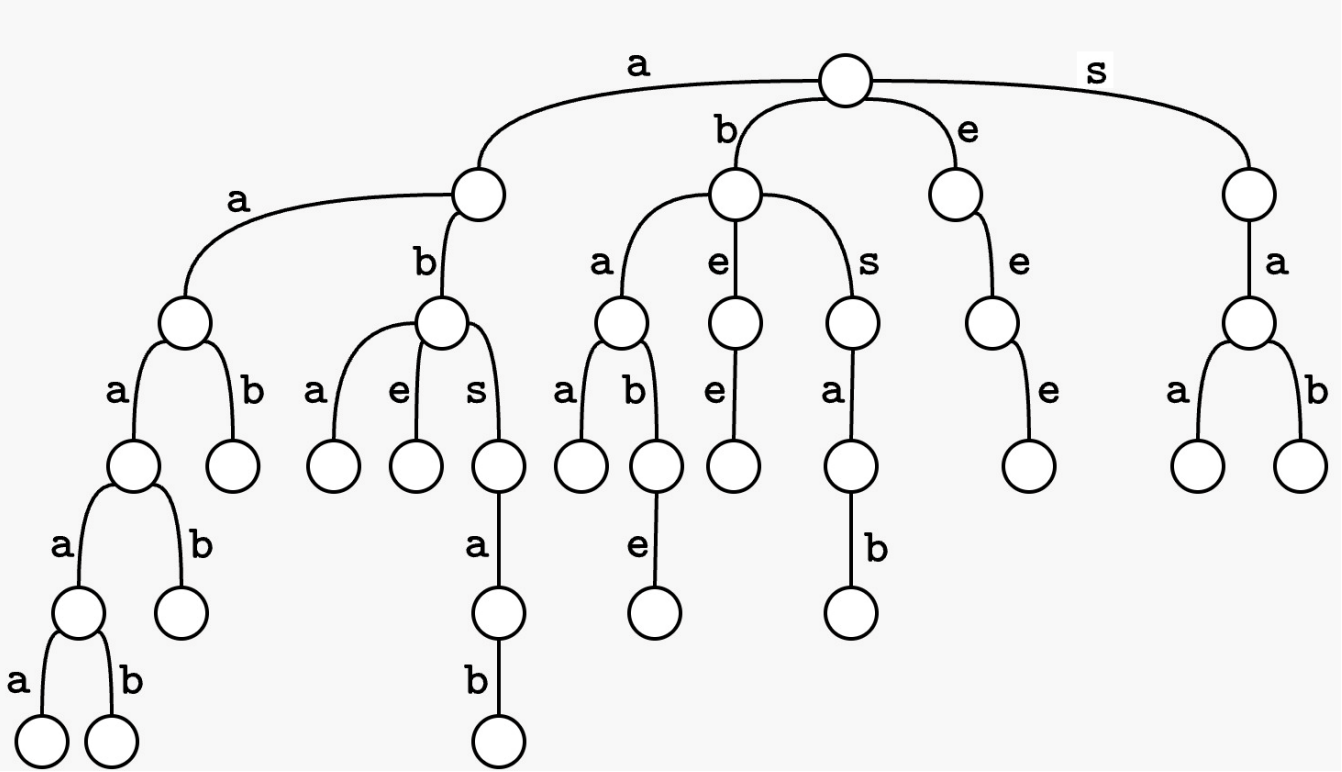
$\boldsymbol{P}_4 = \{\text{aaaa, absa, babe, bsab, aaab}\}$

$\boldsymbol{C}_3 = \{\text{aaa, aab, aba, abe, abs, baa, bab, be, bsa, eee, saa, sab}\}$

$\boldsymbol{C}_5 = \{\text{aaaaa, aaaab, absab}\}$

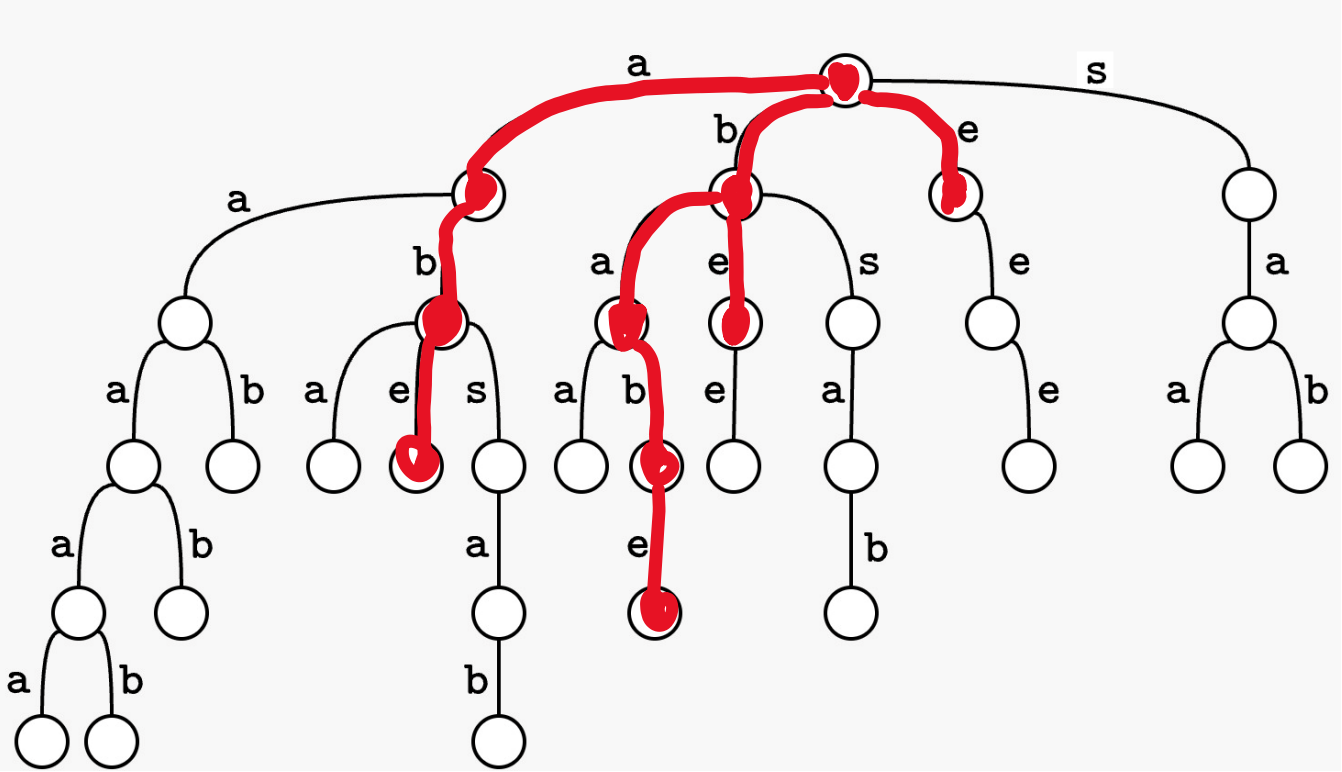
We want to compute noisy counts for each node:
The noisy counts should be close to the true counts in *D*

Step 2: Trie data structure on C



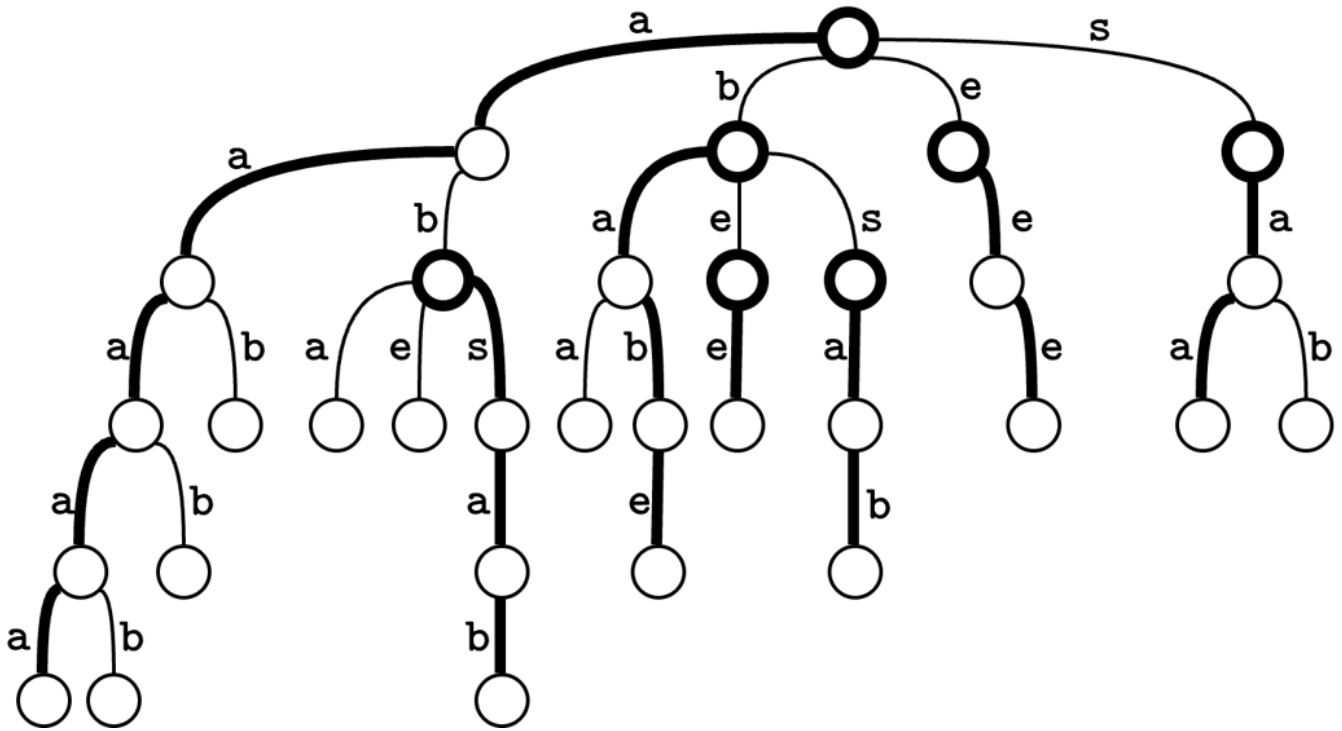
Insight 2: All substrings of S are prefixes of suffixes of S. Hence, they lie on at most l many paths in a trie.

Step 2: Trie data structure on C



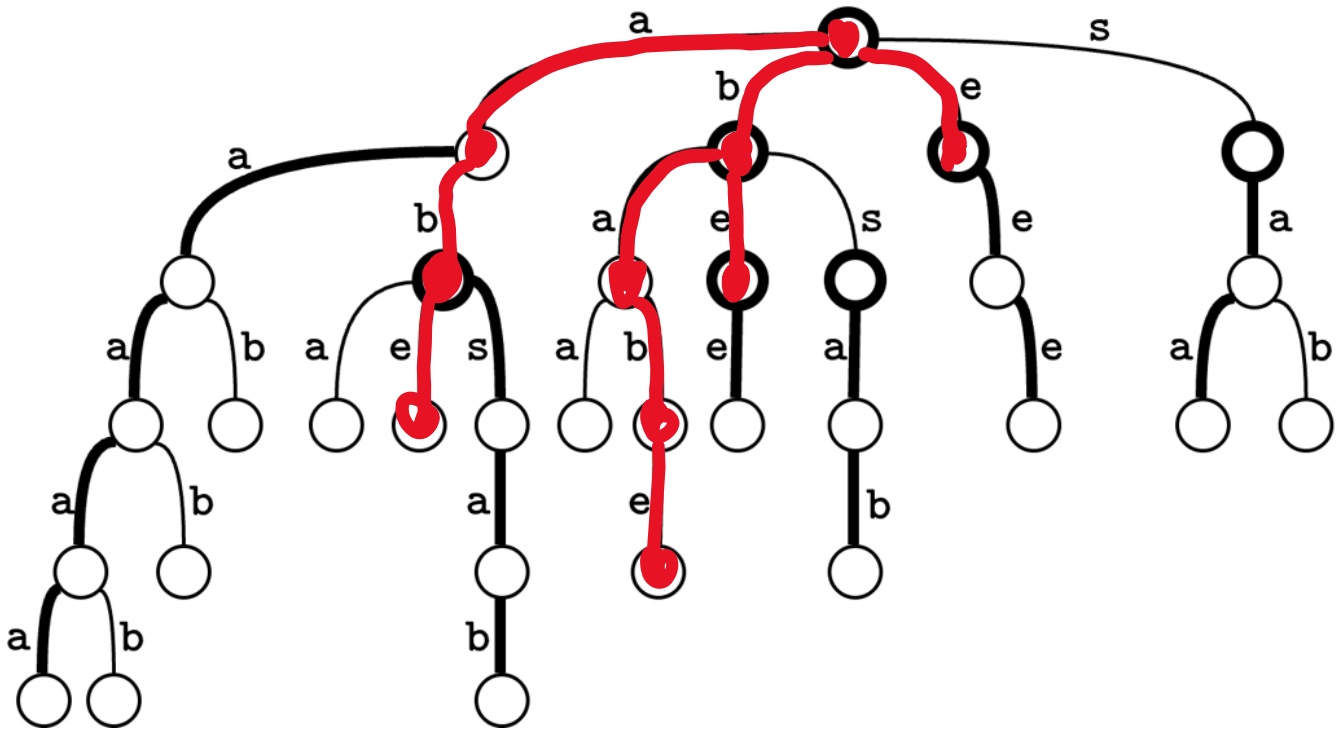
Insight 2: All substrings of S are prefixes of suffixes of S. Hence, they lie on at most l many paths in a trie.

Step 2: Trie data structure on C



Heavy path decomposition: any root-to-leaf path crosses at most $O(\log|T|) = O(\log(nl))$ heavy paths.

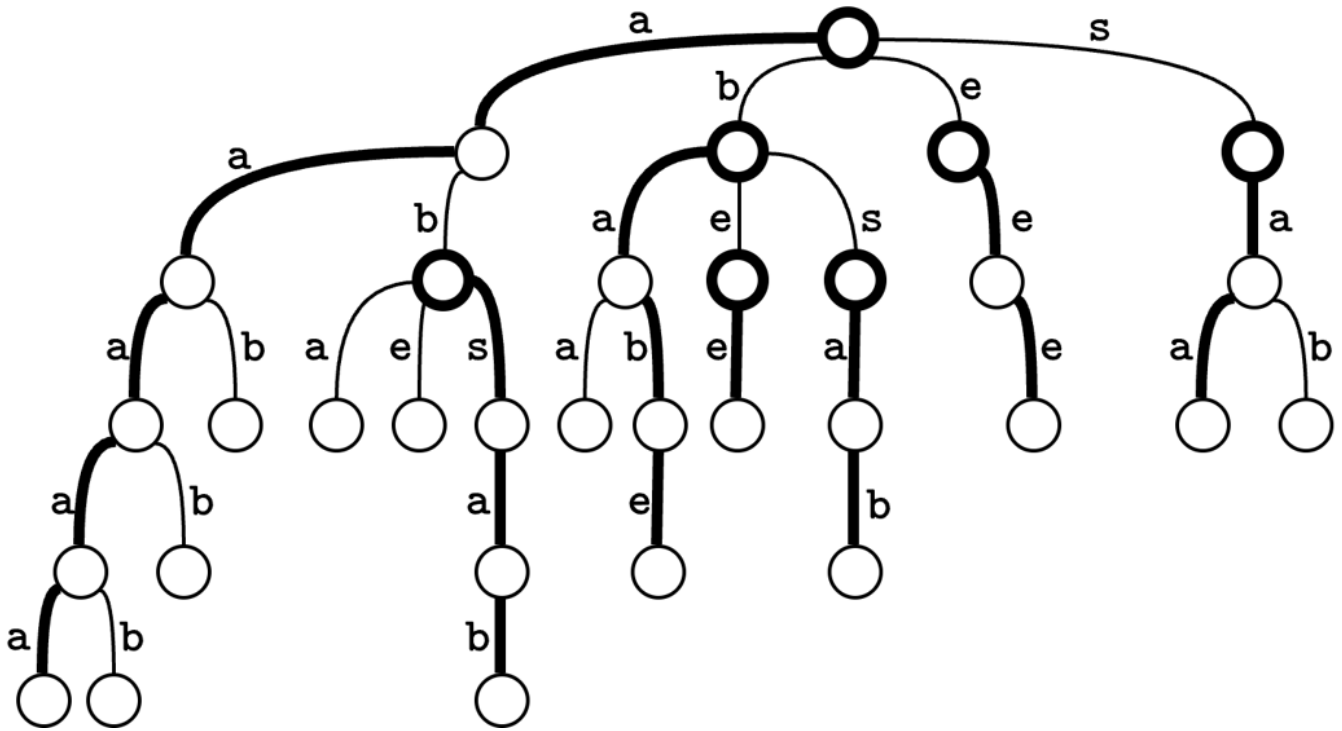
Step 2: Trie data structure on C



Heavy path decomposition: any root-to-leaf path crosses at most $O(\log|T|) = O(\log(nl))$ heavy paths.

l paths cross at most $O(l\log(nl))$ heavy paths.

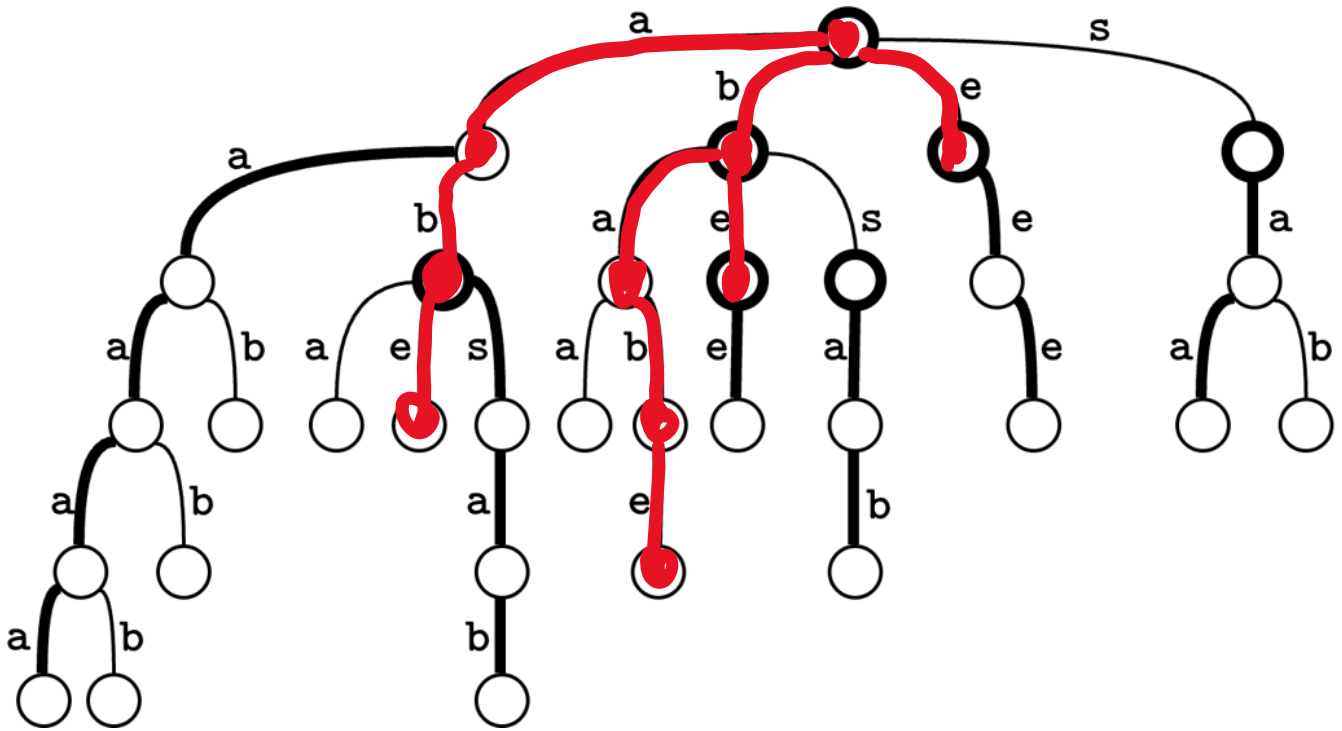
Step 2: Trie data structure on C



Strategy for computing noisy counts:

- We compute a noisy count for each heavy path root.
- We compute prefix sums of the difference sequence of counts for each heavy path.

Step 2: Trie data structure on C

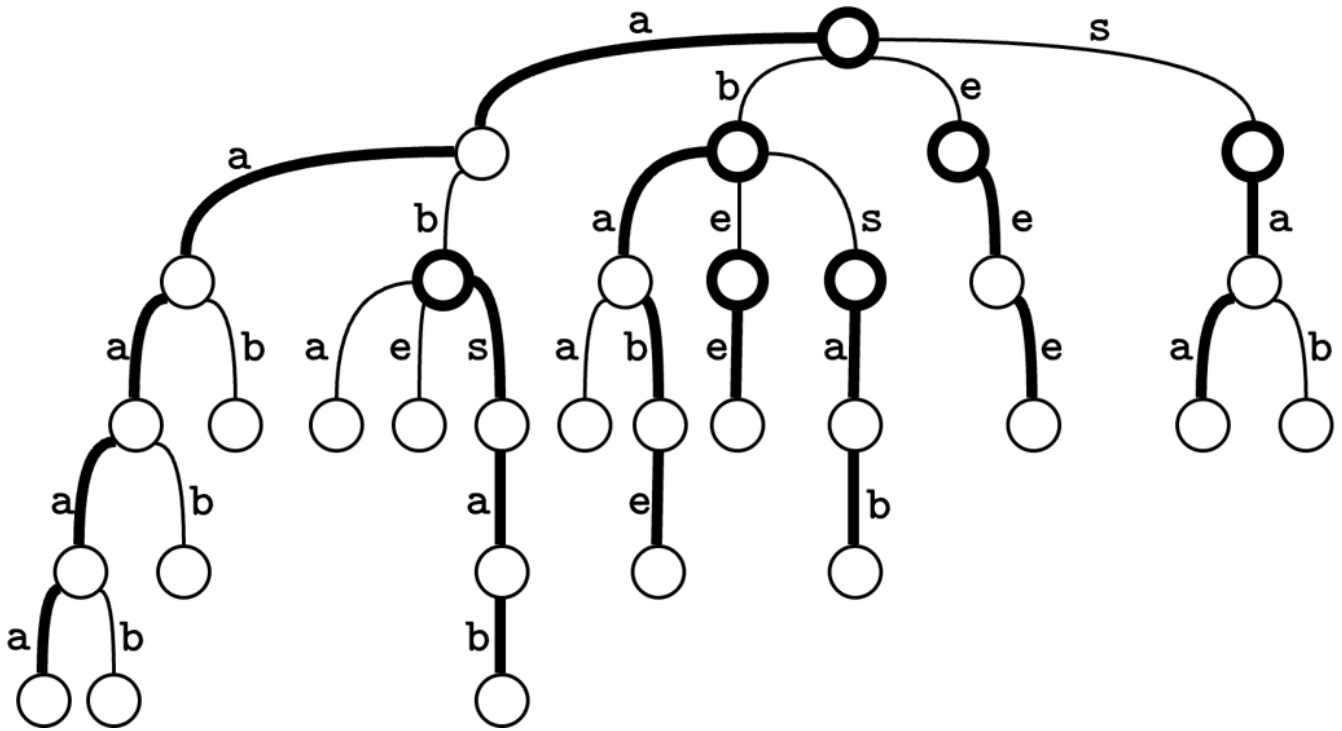


Lemma: The sensitivity of the counts of all heavy path roots is $O(l \log nl)$

→ A string *S* influences only the count of a root by the number of suffixes which have a path through it

→ Each suffix goes through at most $\log nl$ roots

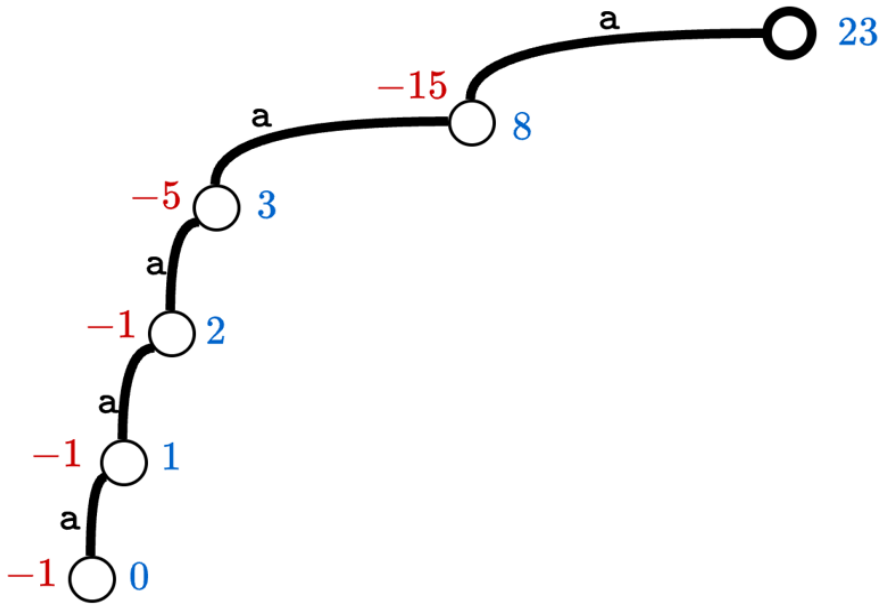
Step 2: Trie data structure on C



Strategy for computing noisy counts:

- ✓ We compute a noisy count for each heavy path root.
- We compute prefix sums of the difference sequence of counts for each heavy path.

Step 2: Trie data structure on C



(a)

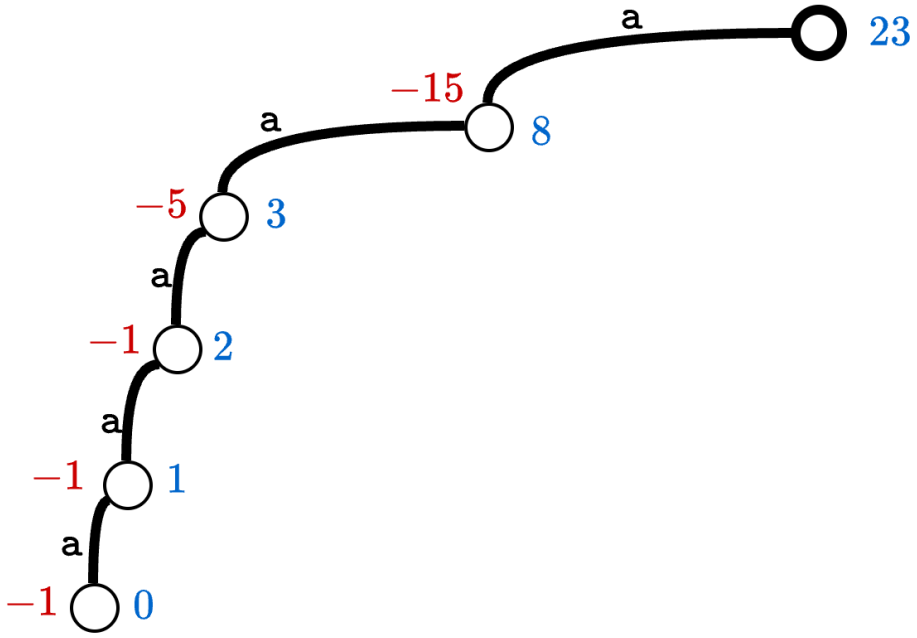
Difference sequence on a heavy path

$$D = \{\text{aaaa}, \text{abe}, \text{absab}, \text{babe}, \text{bee}, \text{bees}\}$$

→ If we have the count of the root and the prefix sums of the heavy path, we can compute a noisy count for each node.

→ How can we compute the prefix sums of the difference sequence?

Step 2: Trie data structure on C



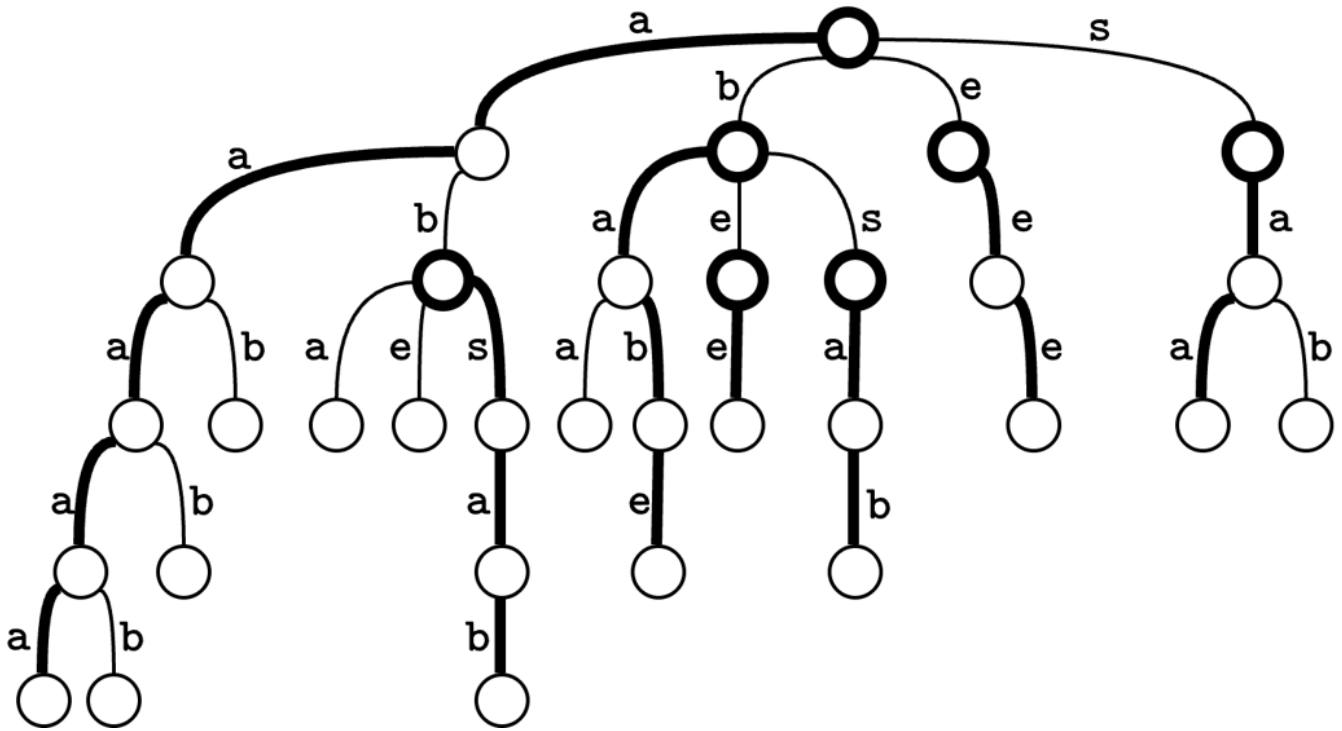
(a)

−15	−20	−22	−23
−5			
−1	−2		
−1			
−1	−1	−1	

(b)

- Binary Tree mechanism!
- The sensitivity of all difference sequences is bounded the sensitivity of the heavy path roots.
- Binary Tree mechanism gives prefix sums with only additional log factors.
- Error $O(l\,\mathrm{polylog}(nl))$

Step 2: Trie data structure on C



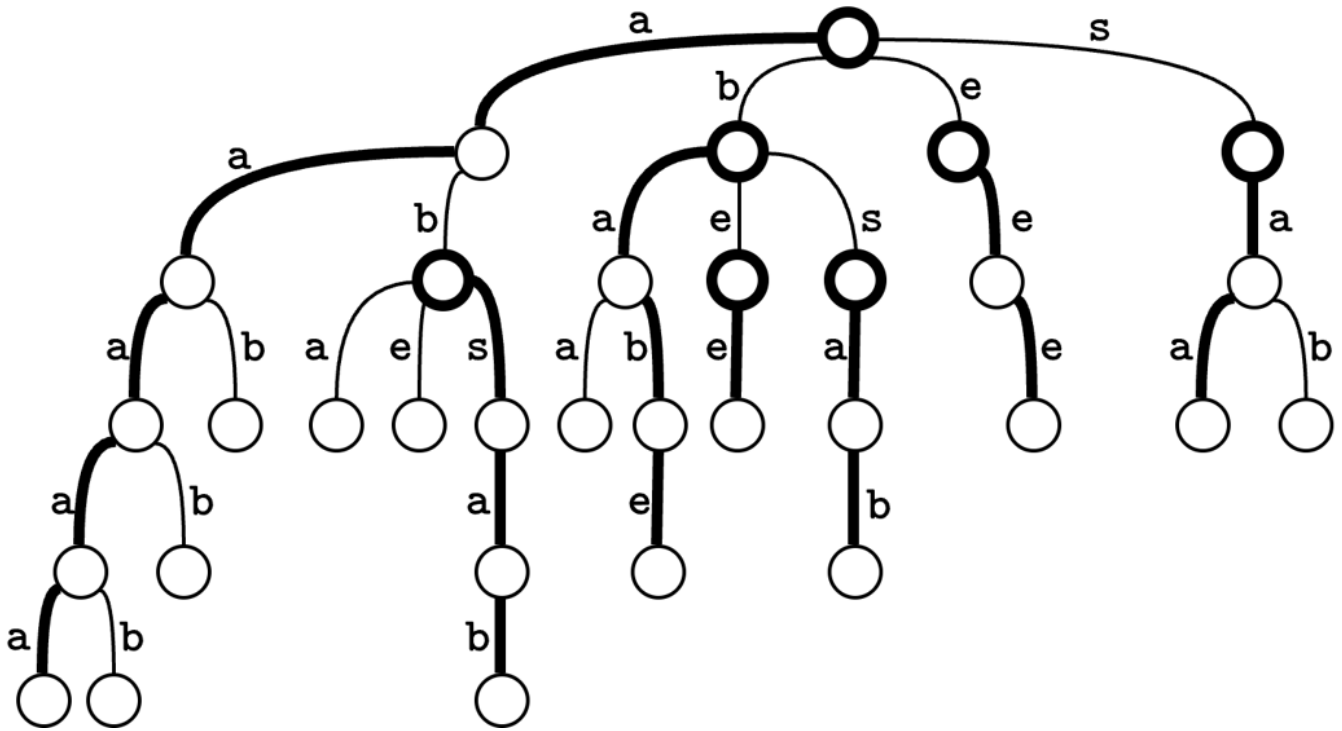
Strategy for computing noisy counts:

- ✓ We compute a noisy count for each heavy path root.
- ✓ We compute prefix sums of the difference sequence of counts for each heavy path.
- ✓ Counts for all nodes can be computed from those.

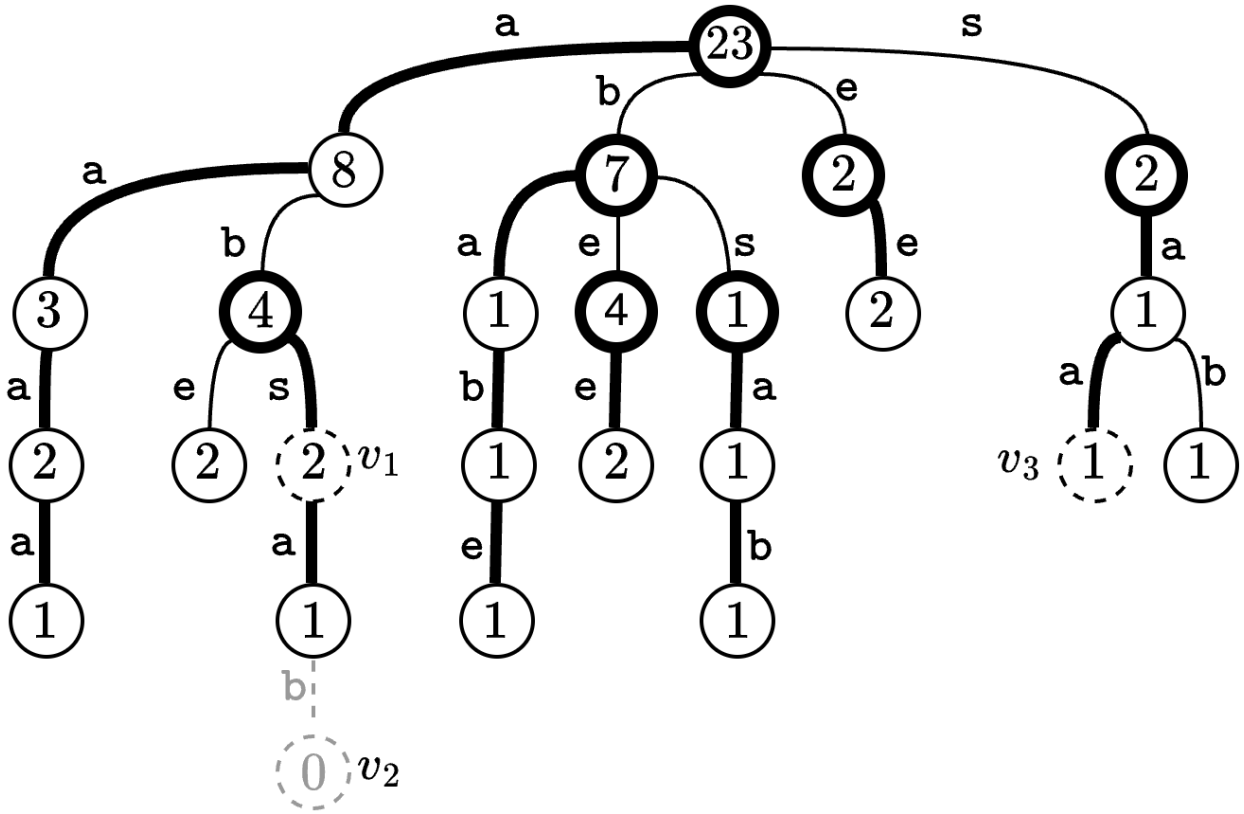
Two Main Steps

- ✓ Step 1: Reduce universe size by filtering
- ✓ Step 2: Build a dp data structure on remaining strings

Secret Final Step: Prune trie



Secret Final Step: Prune trie



Remove nodes with low count.

$D = \{\text{aaaa}, \text{abe}, \text{absab}, \text{babe}, \text{bee}, \text{bees}\}$

Upper bound Results

→ We get a data structure of size $\mathrm{O}(nl^2)$, query time $O(|P|)$, and max error $\mathrm{O}(l\ \mathrm{polylog}(nl|\Sigma|))$ whp for document count and substring count.

Open Problem 2

→ Two *collections of strings* D and D' are neighboring, if they differ in one position in one string.

→ $D = (S_1, S_2, \dots, S_j, \dots, S_n)$

→ $D' = (S_1, S_2, \dots, S'_j, \dots, S_n)$

→ S_j and S'_j differ in one position.

→Substring Count is still hard. What about Document Count?

→Rest of the day: work on exercises, open problems, chat.