

# Making Text Search More Powerful by Supporting Fuzzy String Matching

Chen Li

UCIrvine  

---

University of California, Irvine



# A few real stories

---

- “Chen Li” vs “Chen Li”
- “Chen Li” vs “Lei Chen”
- .....



# Demo

---

- <http://phwww.cwis.uci.edu/cgi-bin/phonebook>
- <http://psearch.ics.uci.edu>
- <http://www.ics.uci.edu/>



# Approximate string selections



Schwarrzenger

Keanu Reeves
Samuel Jackson
Schwarzenegger
Samuel Jackson
...

## Query errors:

- Limited knowledge about data
- Typos
- Limited input device (cell phone) input

## Data errors

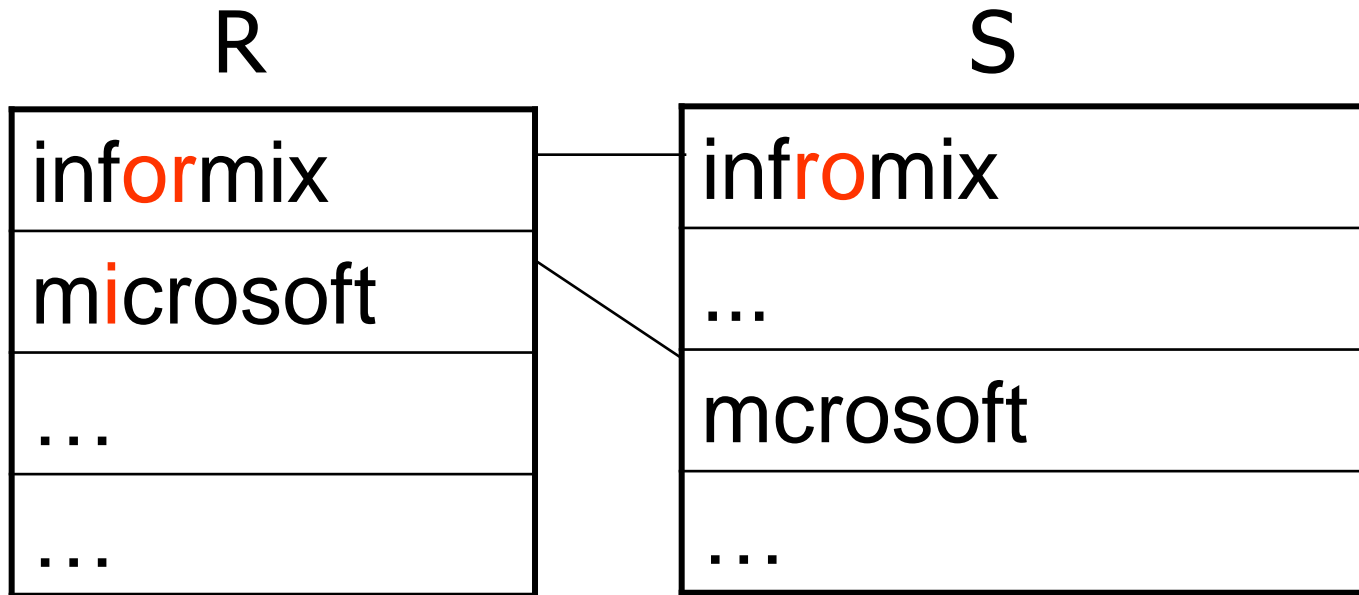
- Typos
- Web data
- OCR

## Applications

- Spellchecking
- Query relaxation
- ...



# Approximate string joins



- Edit distance
- Jaccard
- Cosine
- ...

# Challenges

---



- Similarity functions
- Efficient indexes and algorithms



# Similarity Functions

- **Similar to:**
  - a domain-specific function
  - returns a similarity value between two strings
- **Examples:**
  - Edit distance:  $\text{ed}(\text{Schwarrzenger}, \text{Schwarzenegger})=2$
  - Cosine similarity
  - Jaccard coefficient distance
  - Soundex
  - ...



# Edit Distance

- A widely used metric to define string similarity
- $Ed(s1,s2)$  = minimum # of operations (insertion, deletion, substitution) to change  $s1$  to  $s2$
- Example:

$s1$ : Tom Hanks

$s2$ : Ton Hank

$ed(s1,s2) = 2$



# Research Overview

---

- Learning similarity functions
- Efficient list-merging algorithms
  - Variable-length grams “VGRAM”
- Selectivity estimation
- Similarity matching using DBMS
- ...



# Goal

---

- Reducing index size (memory)
- Reducing running time



# "q-grams" of strings

---

u n i v e r s a l

The diagram shows the string "universal" with vertical dashed lines between each character. Below the string, horizontal red lines are drawn between pairs of adjacent characters, representing the extraction of 2-grams. The red lines are positioned between 'u' and 'n', 'n' and 'i', 'i' and 'v', 'v' and 'e', 'e' and 'r', 'r' and 's', 's' and 'a', and 'a' and 'l'.

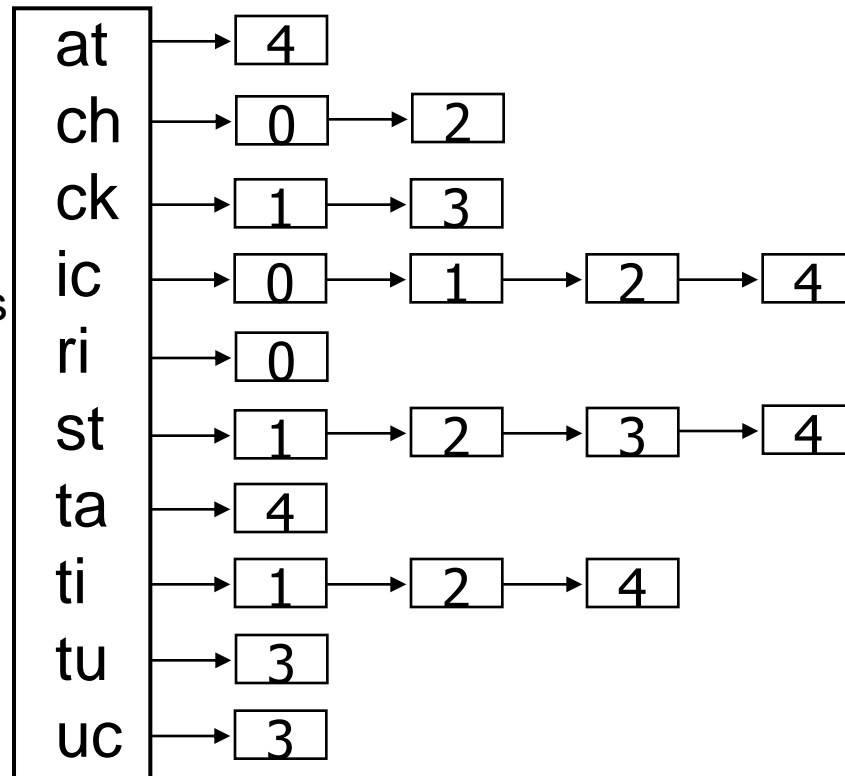
**2-grams**



# q-gram inverted lists

id	strings
0	rich
1	stick
2	stich
3	stuck
4	static

2-grams





# Searching using inverted lists

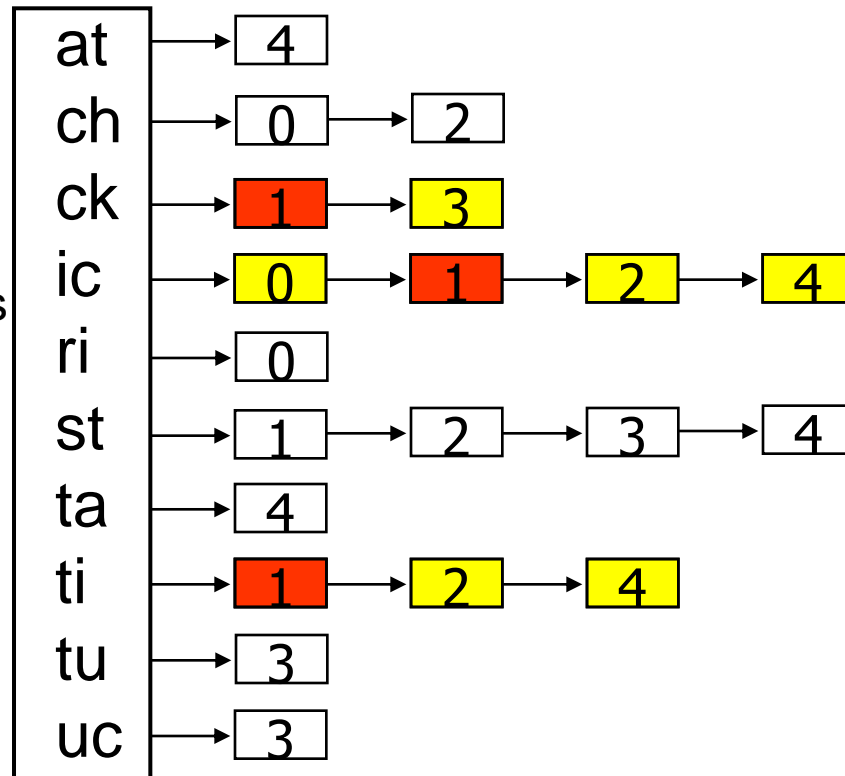
- Query: "shtick",  $ED(\text{shtick}, ?) \leq 1$

sh ht ti ic ck

# of common grams  $\geq 3$

id	strings
0	rich
1	stick
2	stich
3	stuck
4	static

2-grams





# 2-grams -> 3-grams?

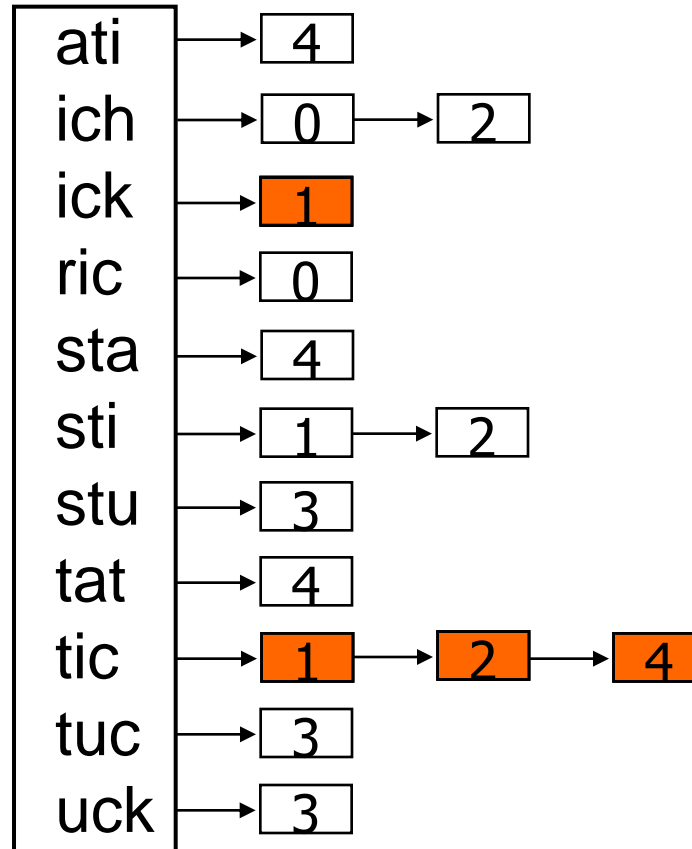
- Query: "shtick",  $ED(shtick, ?) \leq 1$

sht hti tic ick

# of common grams  $\geq 1$

id	strings
0	rich
1	stick
2	stich
3	stuck
4	static

3-grams





# Outline

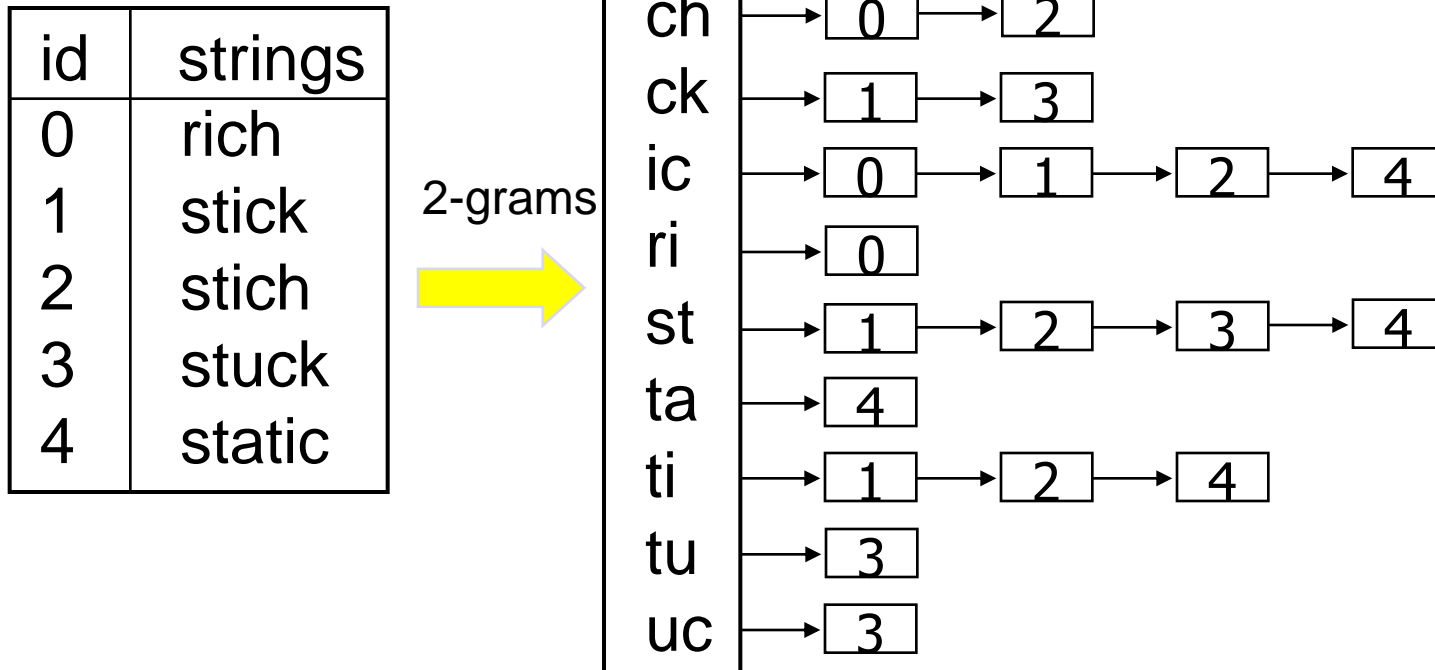
---

- **Motivation**
- **VGRAM**
  - Main idea
  - Decomposing strings to grams
  - Choosing good grams
  - Effect of edit operations on grams
  - Adopting vgram in existing algorithms
- **Experiments**



# Observation 1: dilemma of choosing “q”

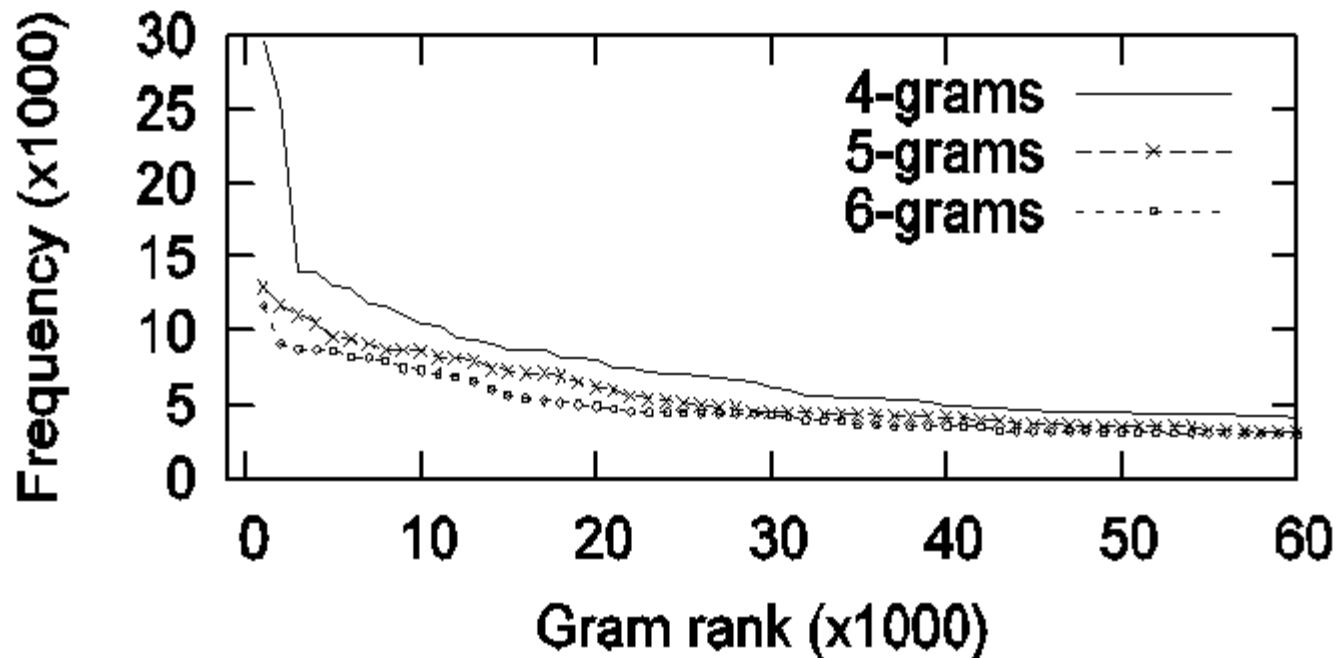
- Increasing “q” causing:
  - Longer grams → Shorter lists
  - Smaller # of common grams of similar strings





## Observation 2: skew distributions of gram frequencies

- DBLP: 276,699 article titles
- Popular 5-grams: **ation** (>114K times), **tions**, **ystem**, **catio**





# VGRAM: Main idea

---

- Grams with **variable lengths** (between  $q_{\min}$  and  $q_{\max}$ )
  - **zebra**
    - ze(123)
  - **corrasion**
    - co(5213), cor(859), corr(171)
- Advantages
  - Reduce index size 😊
  - Reducing running time 😊
  - Adoptable by many algorithms 😊



# Challenges

---

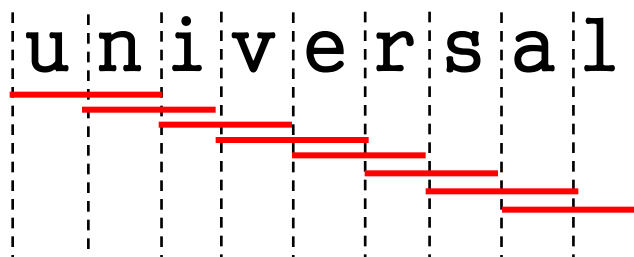
- Generating **variable-length** grams?
- Constructing a **high-quality** gram dictionary?
- **Relationship** between string similarity and their gram-set similarity?
- **Adopting** VGRAM in existing algorithms?



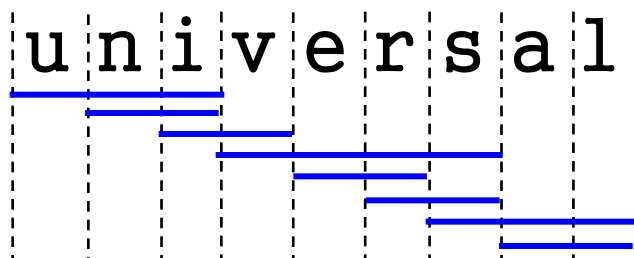


# Challenge 1: String $\rightarrow$ Variable-length grams?

- Fixed-length 2-grams



- Variable-length grams



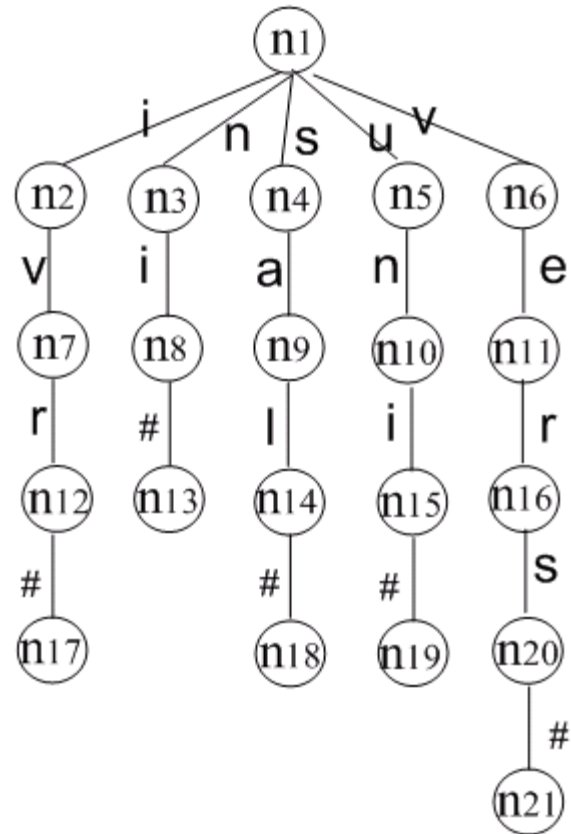
[2,4]-gram dictionary

ni
ivr
sal
uni
vers



# Representing gram dictionary as a trie

ni  
ivr  
sal  
uni  
vers



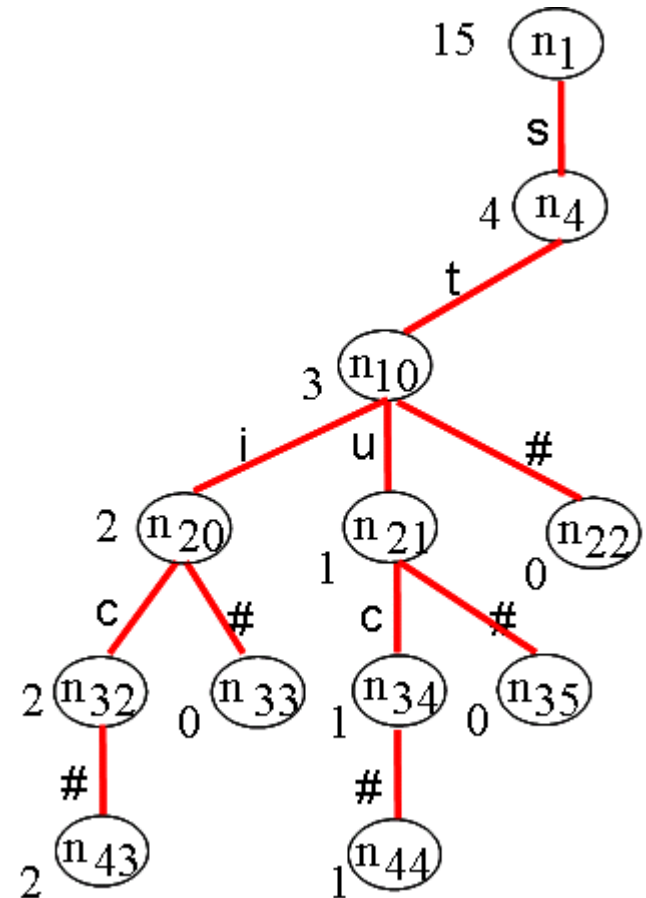
## Challenge 2: Constructing gram dictionary

Step 1: Collecting frequencies of grams with length in [qmin, qmax]

id	string
0	stick
1	stich
2	such
3	stuck

(a) strings

st → 0, 1, 3  
sti → 0, 1  
stu → 3  
stic → 0, 1  
stuc → 3



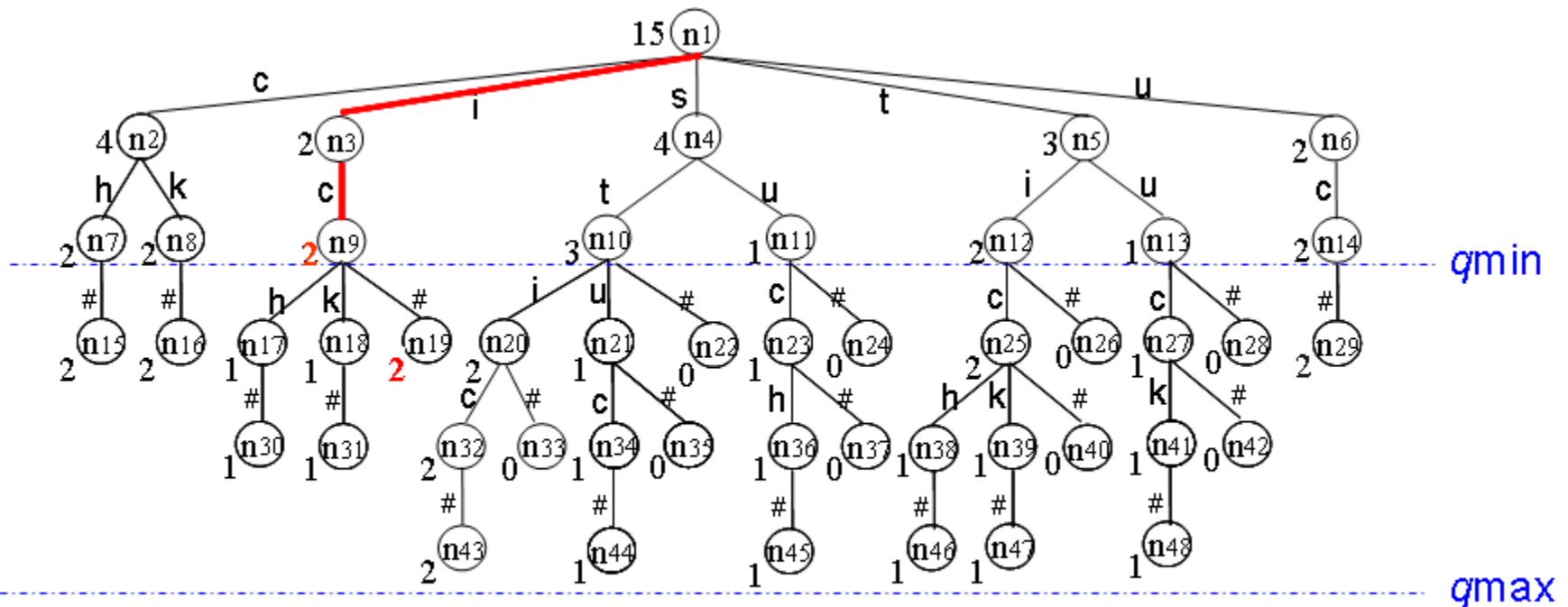
Gram trie with frequencies



# Step 2: selecting grams

- Pruning trie using a frequency threshold  $T$  (e.g., 2)

id	string
0	stick
1	stich
2	such
3	stuck



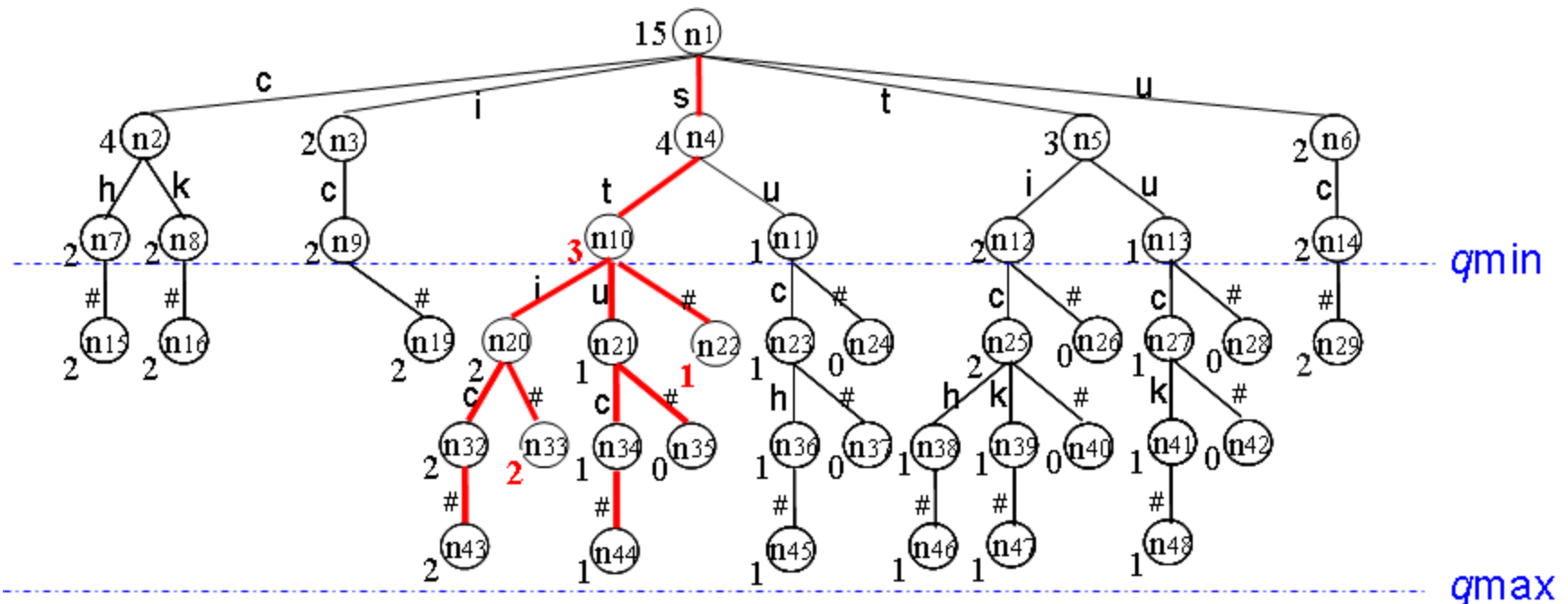
A gram-frequency trie: [2,4]-gram

# Step 2: selecting grams (cont)

id	string
0	stick
1	stich
2	such
3	stuck

(a) strings

Threshold  $T = 2$



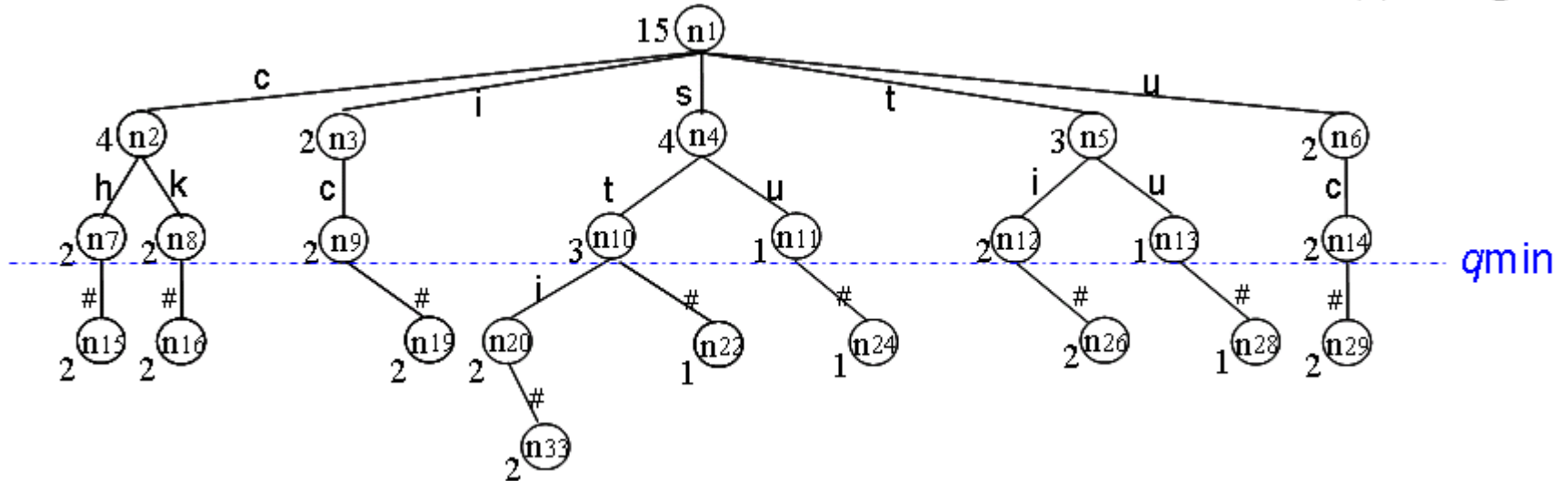
A gram-frequency trie: [2,4]-gram



# Final gram dictionary

id	string
0	stick
1	stich
2	such
3	stuck

(a) strings



[2,4]-grams



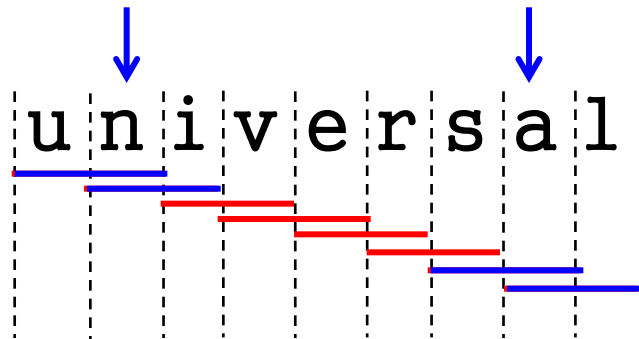
# Outline

---

- Motivation
- VGRAM
  - Main idea
  - Decomposing strings to grams
  - Choosing good grams
  - → **Effect of edit operations on grams**
  - Adopting vgram in existing algorithms
- Experiments



## Challenge 3: Edit operation's effect on grams

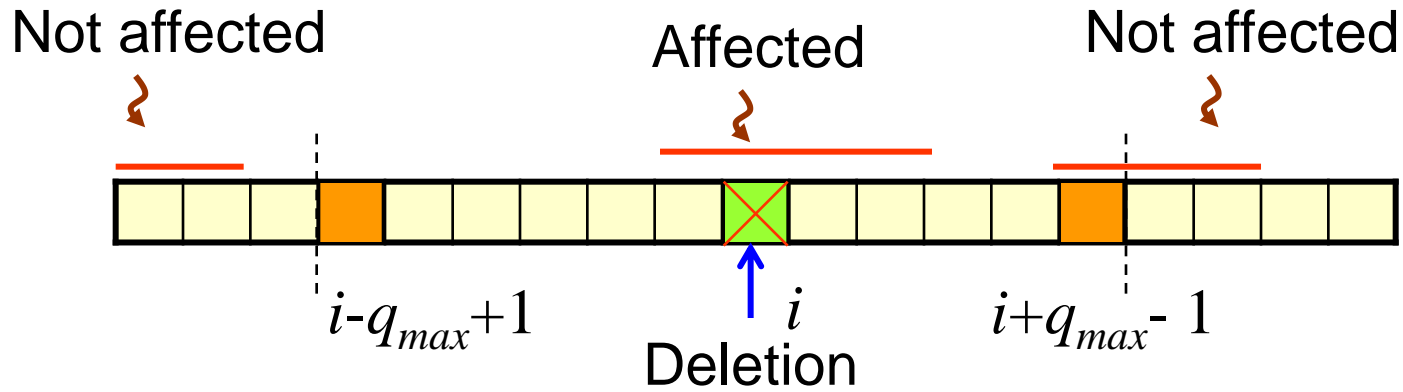


Fixed length:  $q$

$k$  operations could affect  $k * q$  grams



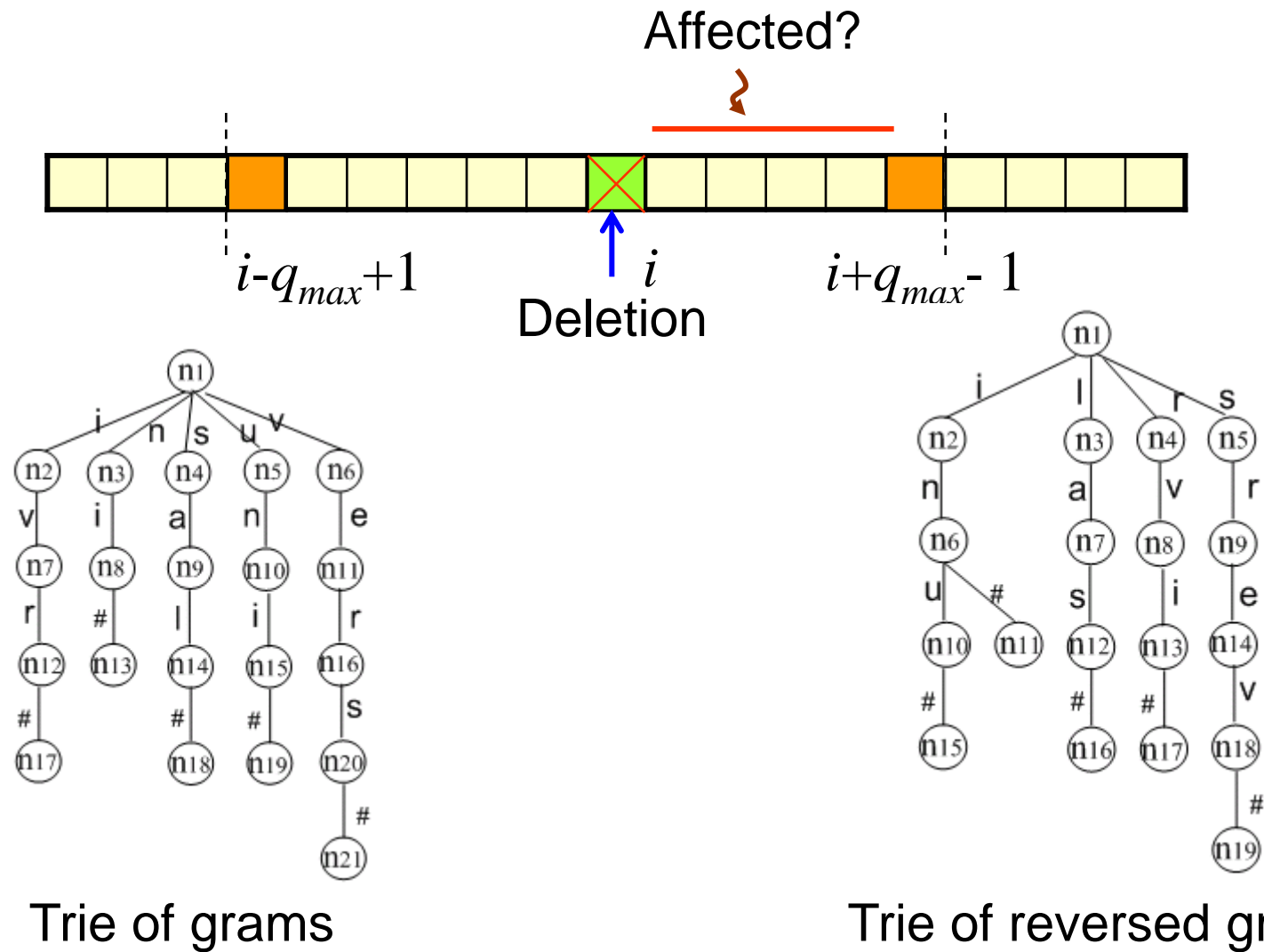
# Deletion affects variable-length grams







# Grams affected by a deletion (cont)





# # of grams affected by each operation

Deletion/substitution



Insertion



0	1	1	1	1	2	1	2	2	2	1	1	1	2	1	1	1	1	0
-	u	-	n	-	i	-	v	-	e	-	r	-	s	-	a	-	l	-



# Max # of grams affected by k operations

---

Vector of  $s = \langle 2, 4, 6, 8, 9 \rangle$



With 2 edit operations, at most 4 grams can be affected

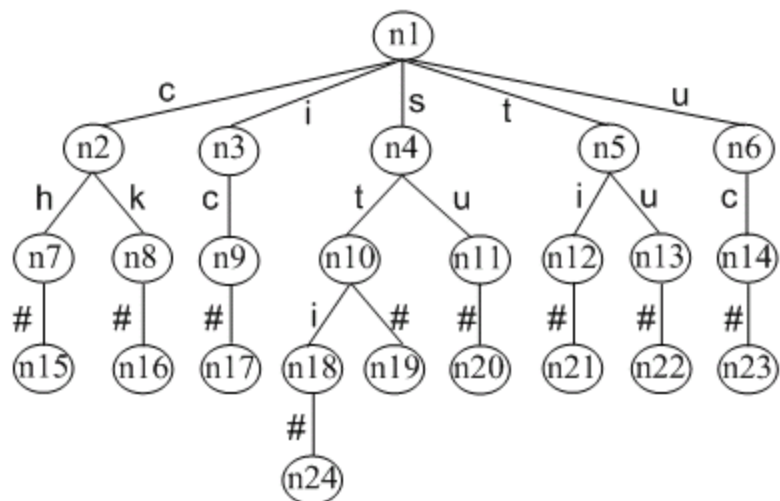
- Called NAG vector (# of affected grams)
- Precomputed and stored



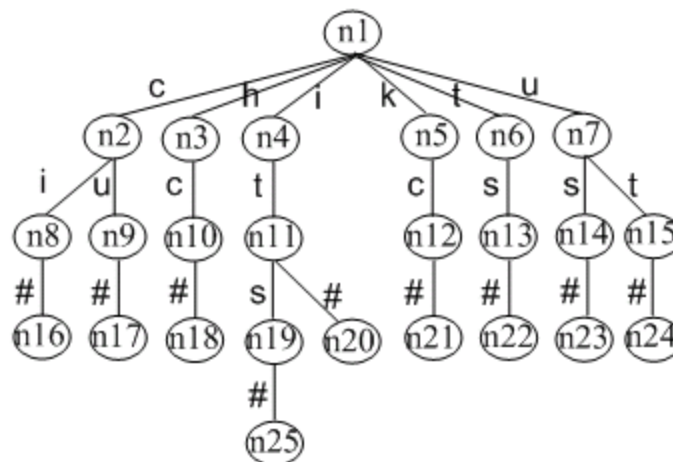
# Summary of VGRAM index

id	string
0	stick
1	stich
2	such
3	stuck

(a) strings



(b) Gram dictionary as a trie



(c) Reversed-gram trie

id	NAG vector
0	2, 3
1	2, 3
2	2, 3
3	3, 4

(d) NAG vectors



# Challenge 4: adopting VGRAM

---

Easily adoptable by many algorithms

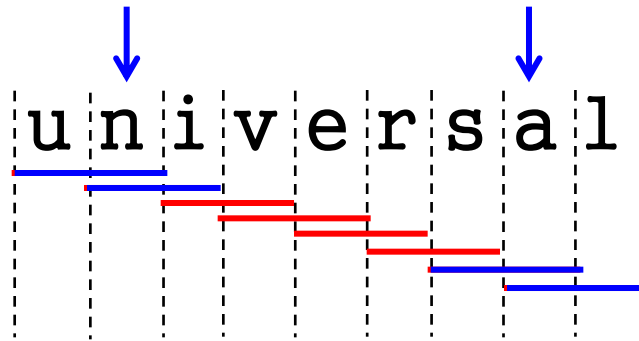
Basic interfaces:

- String  $s \rightarrow$  grams
- String  $s_1, s_2$  such that  $\text{ed}(s_1, s_2) \leq k \rightarrow$  min # of their common grams



# Lower bound on # of common grams

## Fixed length (q)



If  $ed(s_1, s_2) \leq k$ , then their # of common grams  $\geq$ :

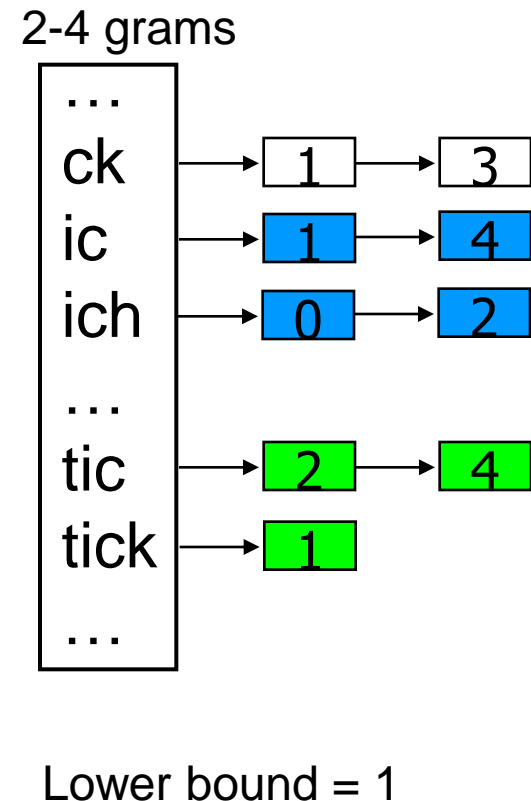
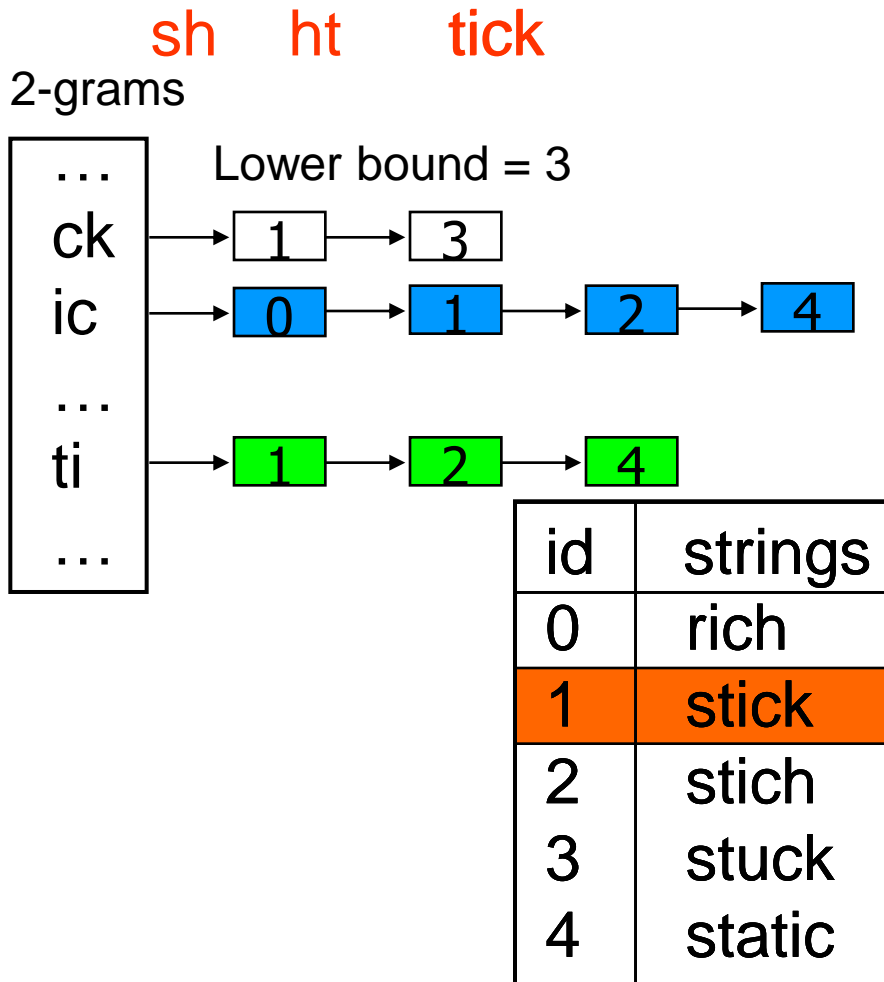
$$(|s_1| - q + 1) - k * q$$

Variable lengths: # of grams of  $s_1$  – NAG( $s_1, k$ )



# Example: algorithm using inverted lists

- Query: “shtick”,  $ED(\text{shtick}, ?) \leq 1$





# Outline

---

- Motivation
- VGRAM
  - Main idea
  - Decomposing strings to grams
  - Choosing good grams
  - Effect of edit operations on grams
  - Adopting vgram in existing algorithms
- Experiments



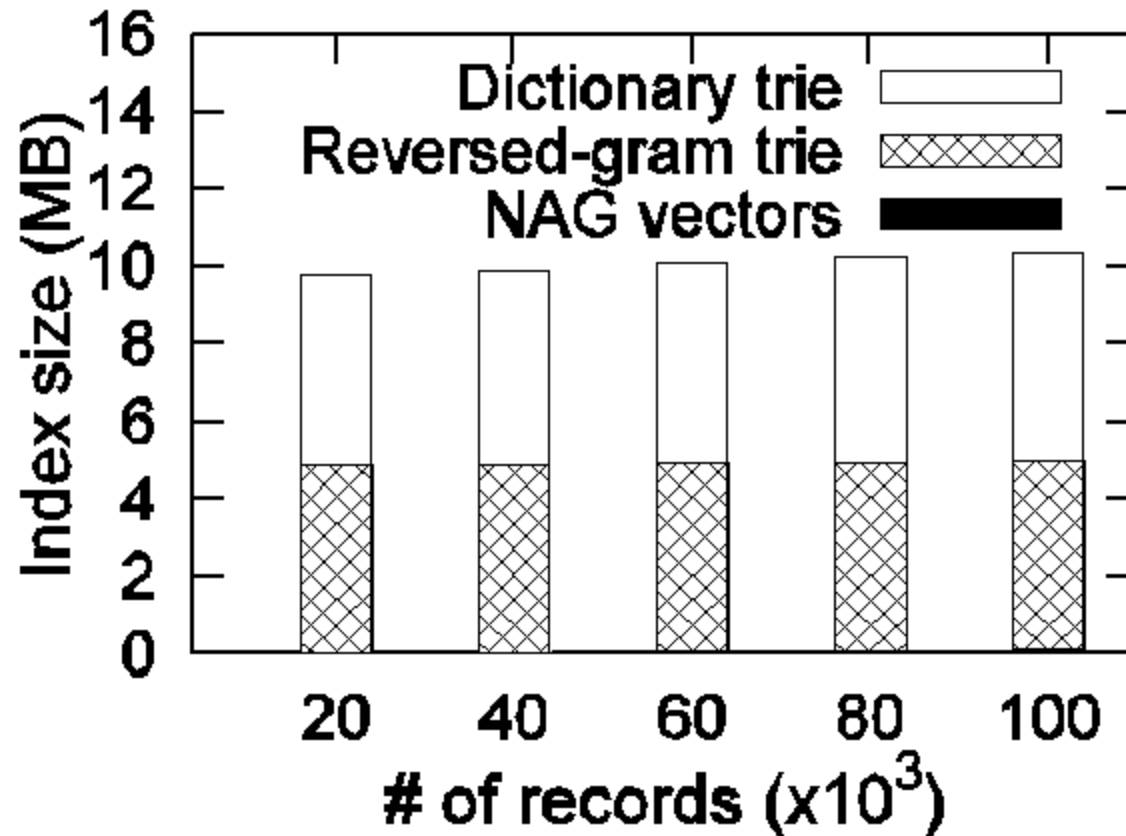
# Data sets

---

- *Data set 1*: Texas Real Estate Commission.
  - 151K person names, average length = 33.
- *Data set 2*: English dictionary from the Aspell spellchecker for Cygwin.
  - 149,165 words, average length = 8.
- *Data set 3*: DBLP Bibliography.
  - 277K titles, average length = 62.



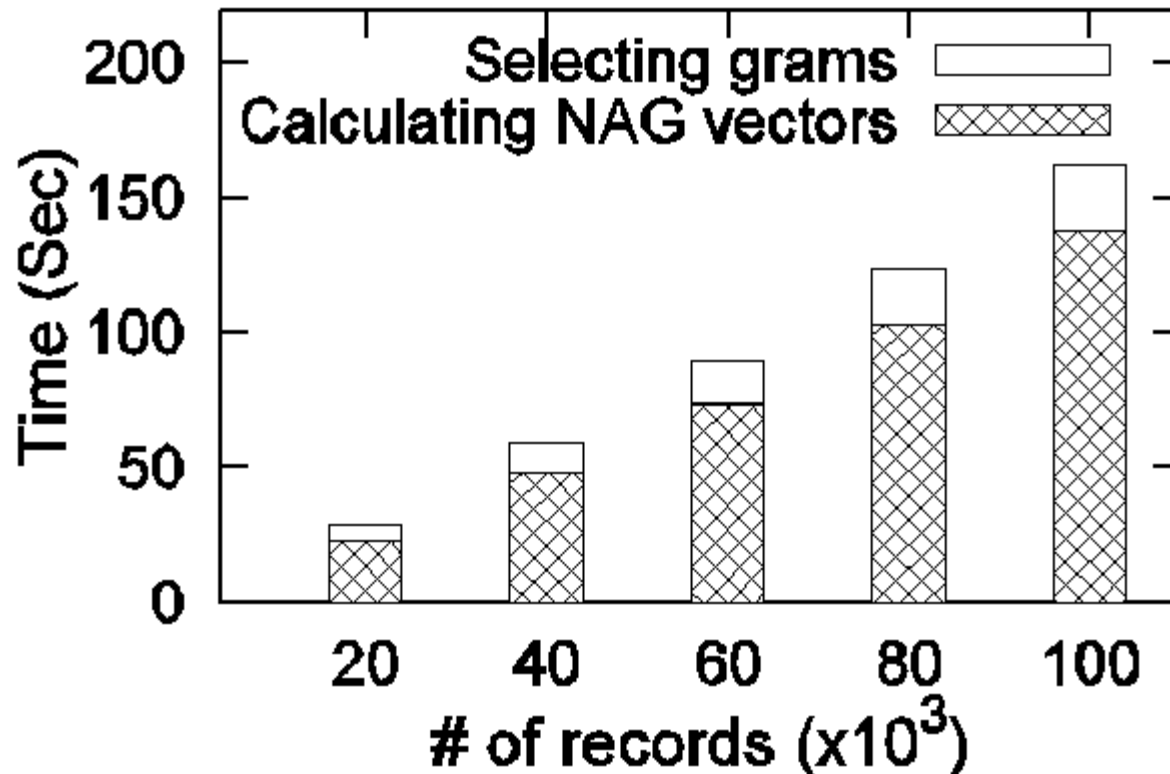
# VGRAM overhead (index size)



Dataset 3: DBLP titles



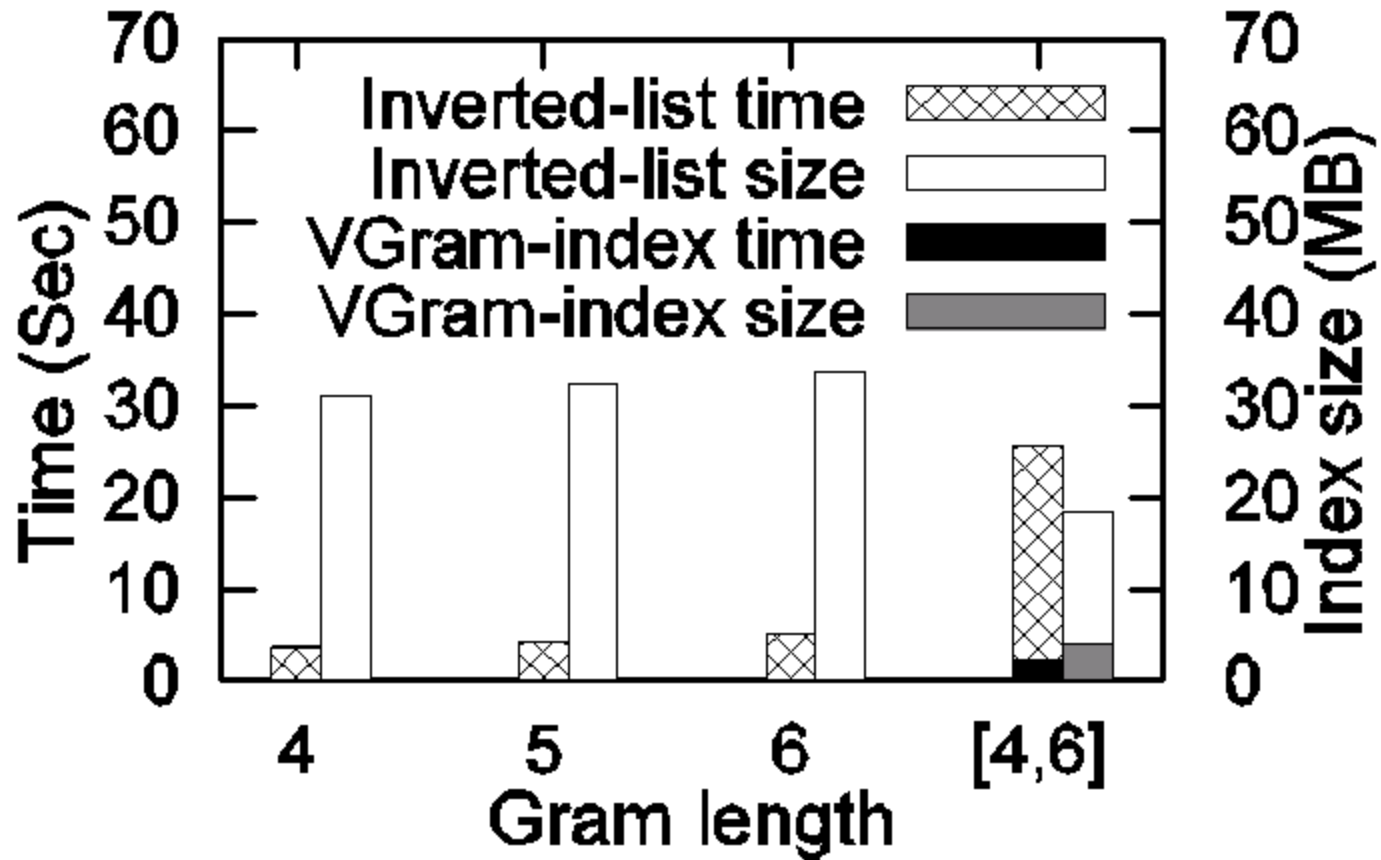
# VGRAM overhead (construction time)



Dataset 3: DBLP titles



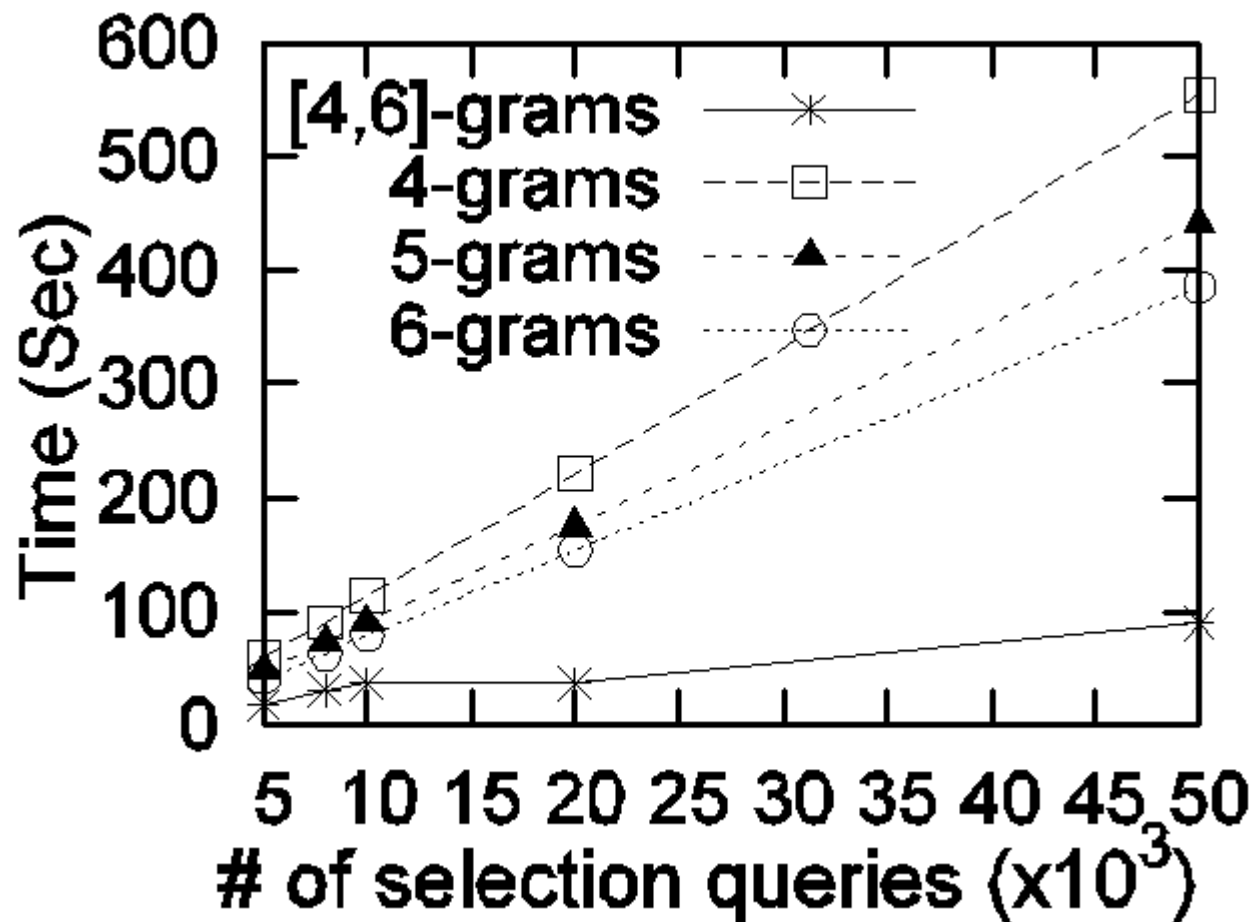
# Benefits over fixed-length grams (index)



Dataset 1: Person names



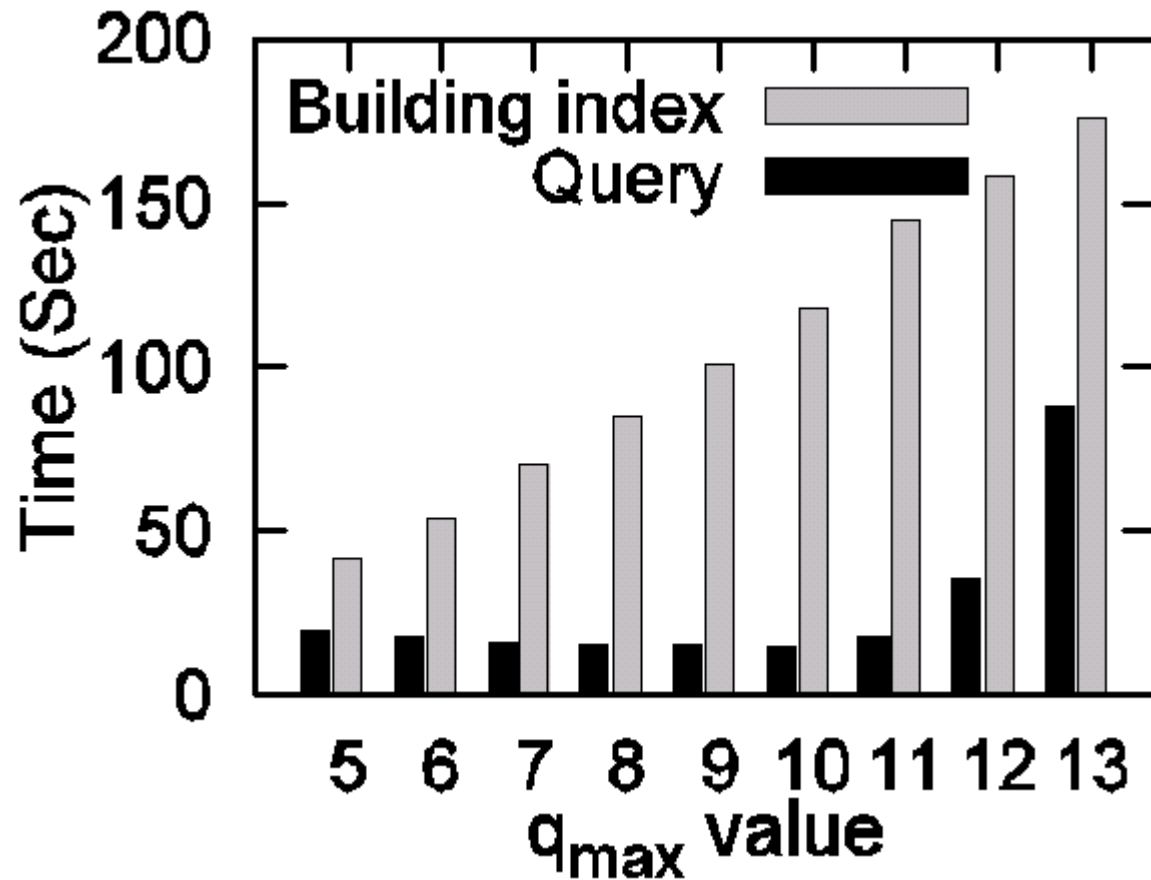
## Benefits over fixed-length grams (running time)



Dataset 1: Person names



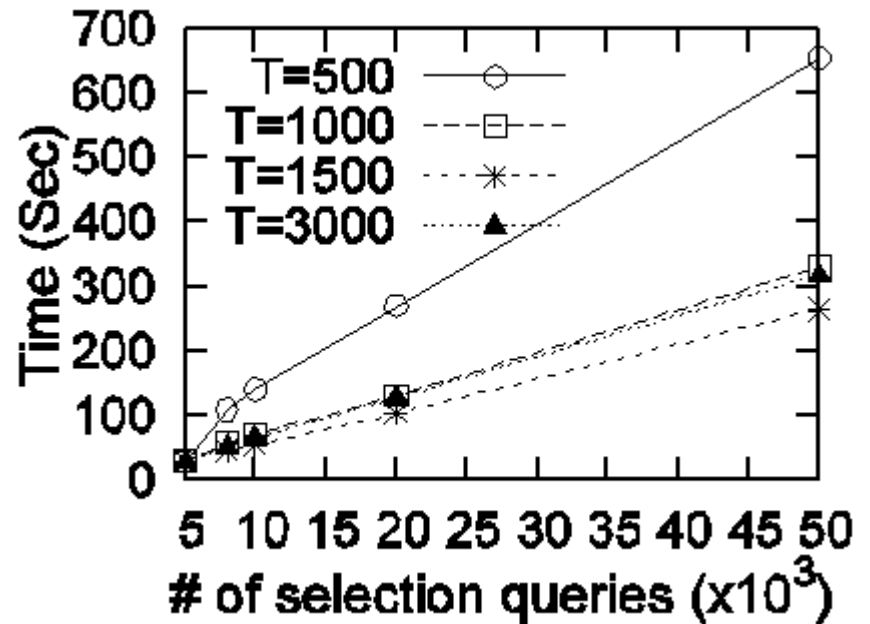
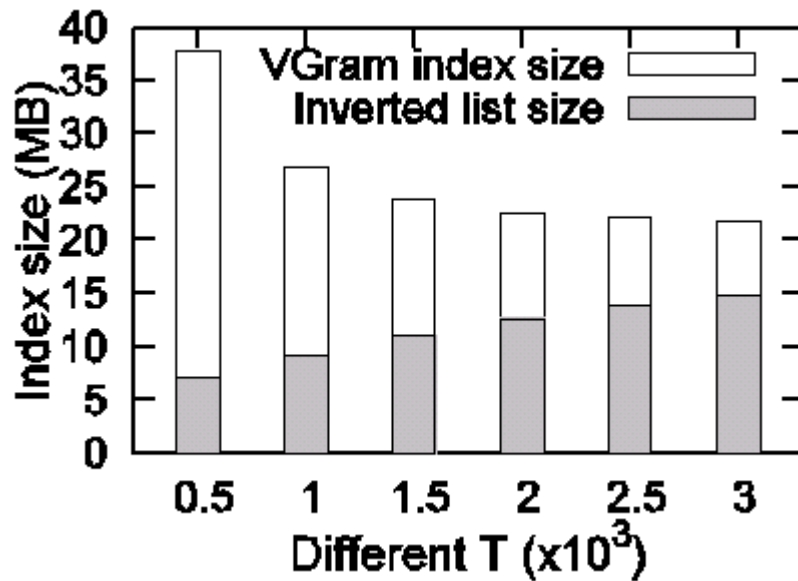
# Effect of $q_{max}$



Dataset 1: Person names



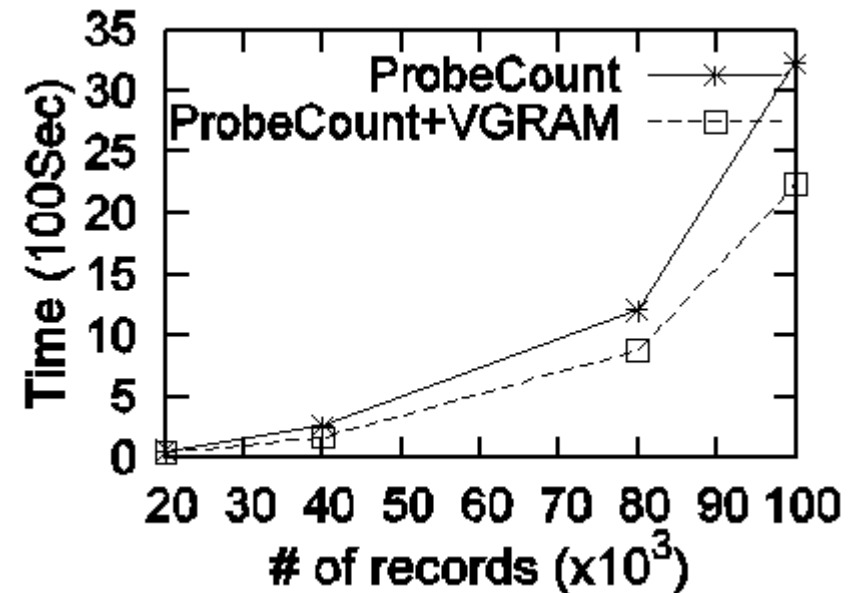
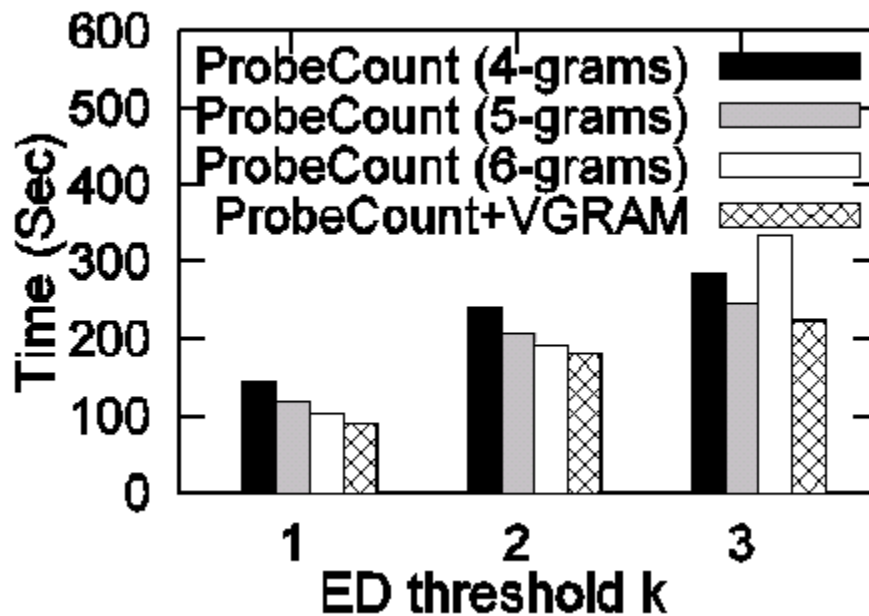
# Effect of frequency threshold $T$



Dataset 1: Person name



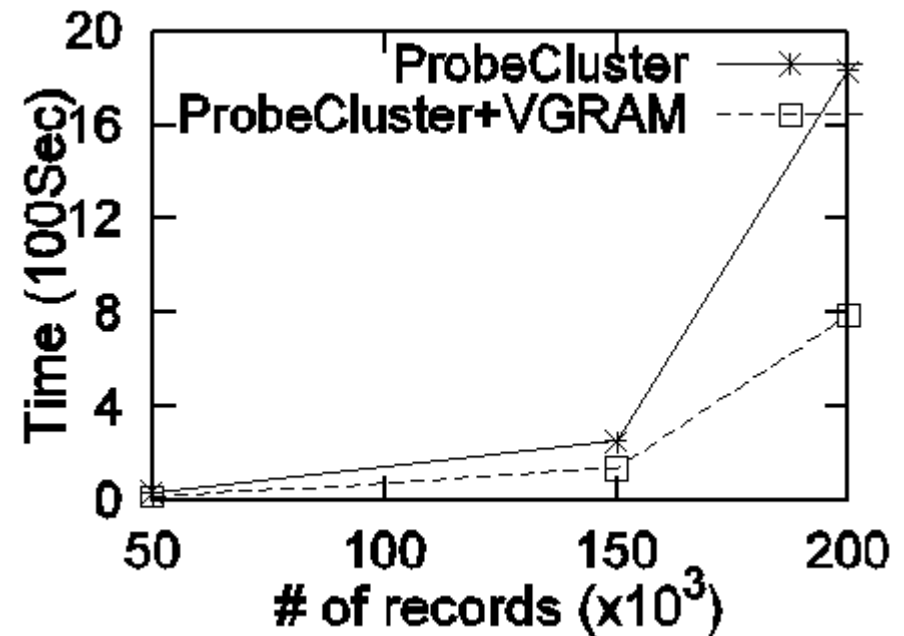
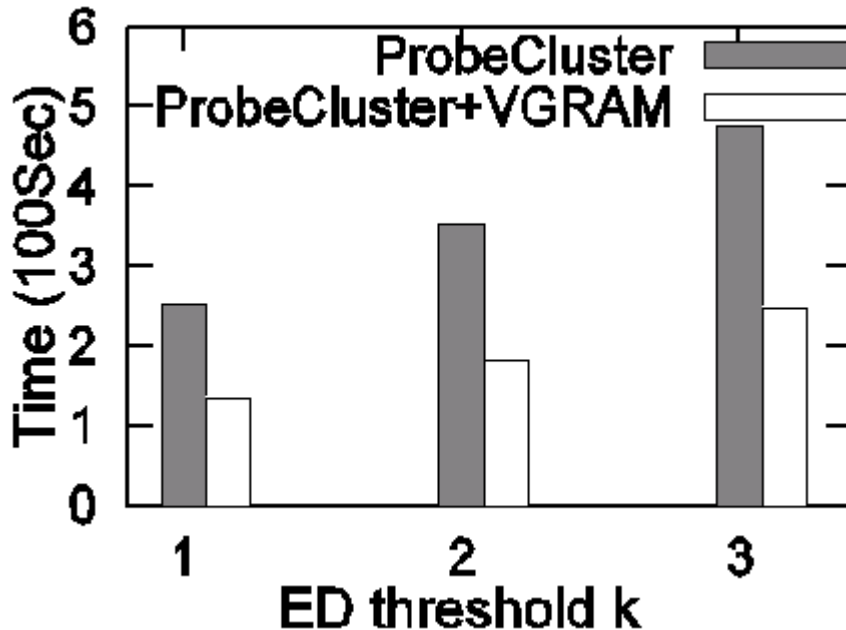
# Improving algorithm ProbeCount



Dataset 1: Person name



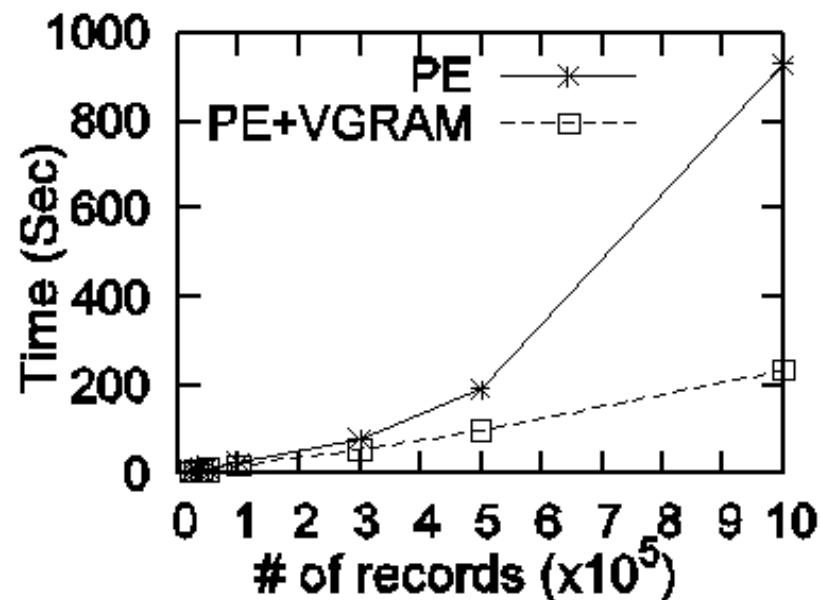
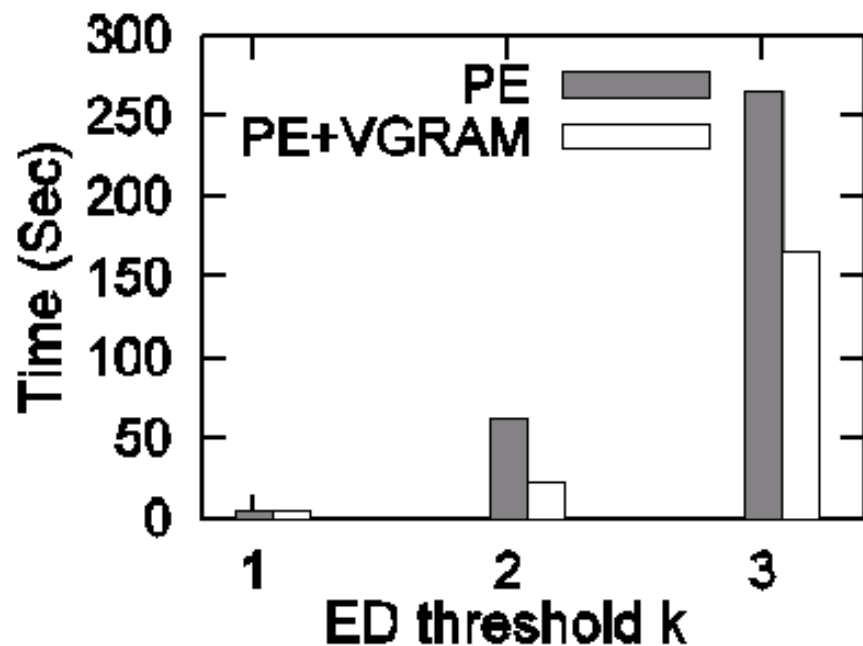
# Improving algorithm ProbeCluster



Dataset 1: Person name



# Improving algorithm PartEnum



Dataset 1: Person name



# Discussions

---

- Dynamic maintenance
- Edit distance variants
  - Approximate substring queries
  - Block moves
- Using VGRAM in DBMS



# Conclusions

---

- Computational challenges of record linkage
  - Domain-independent issues
  - Generic issues
- Efficient techniques