

---

# Approximate Tree Matching with pq-Grams

**Nikolaus Augsten<sup>a</sup>, Michael Böhlen, Johann Gamper**

DIS - Center for Database and Information Systems

Free University of Bozen-Bolzano, Italy

[www.inf.unibz.it](http://www.inf.unibz.it)

1 – <b>Motivation</b> . . . . .	2
2 – <b>Related Work</b> . . . . .	6
3 – <b><i>pq</i>-Grams</b> . . . . .	7
4 – <b>Properties</b> . . . . .	11
5 – <b>Experiments</b> . . . . .	14
6 – <b>Conclusion and Future Work</b> . . . . .	21

---

<sup>a</sup>Supported by the Municipality of Bozen-Bolzano.

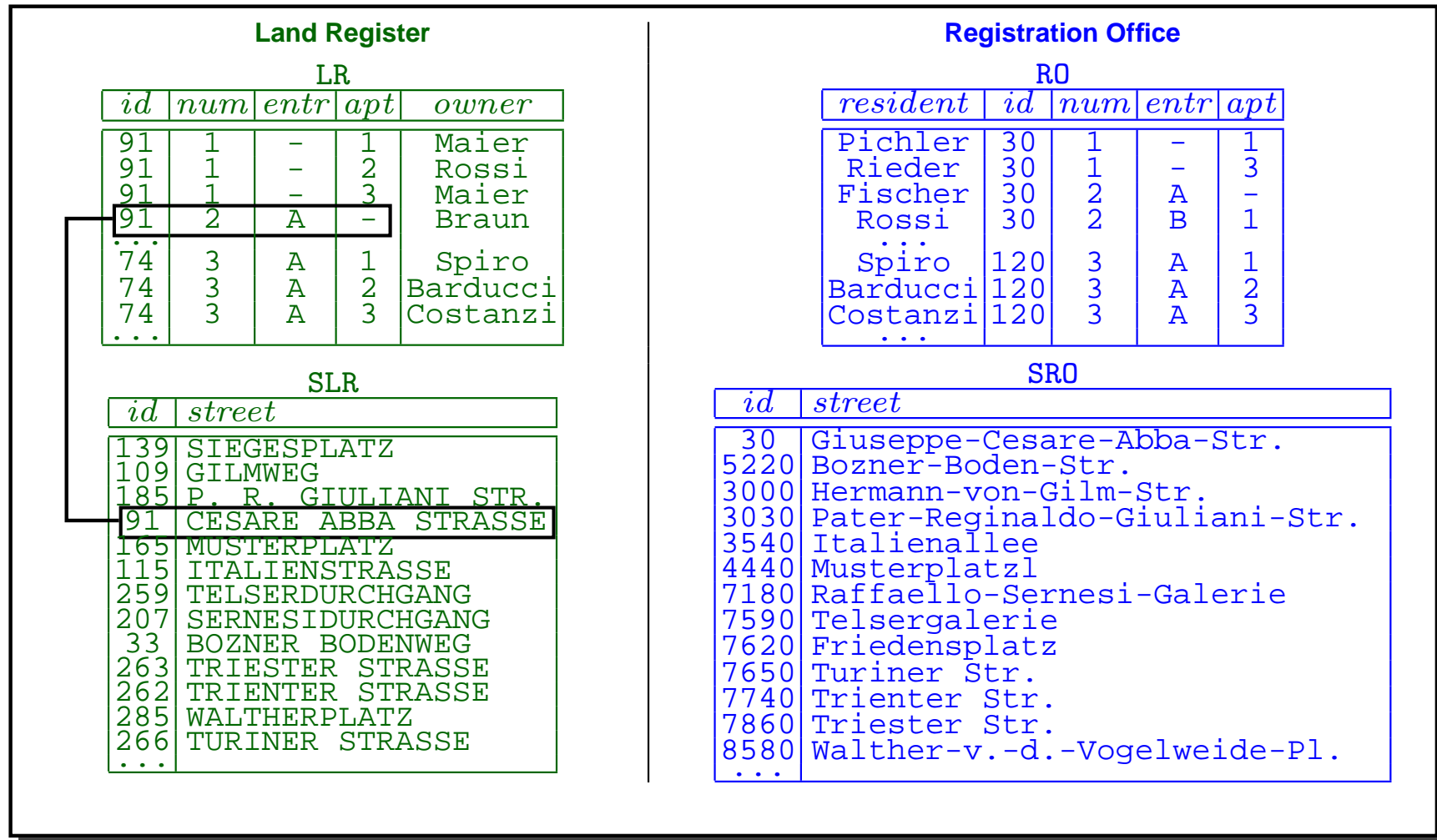
# Motivation — Example Data Sources

---

☞ We want to **link data items** in **different databases** that correspond to the **same real world object**.

# Motivation — Example Data Sources

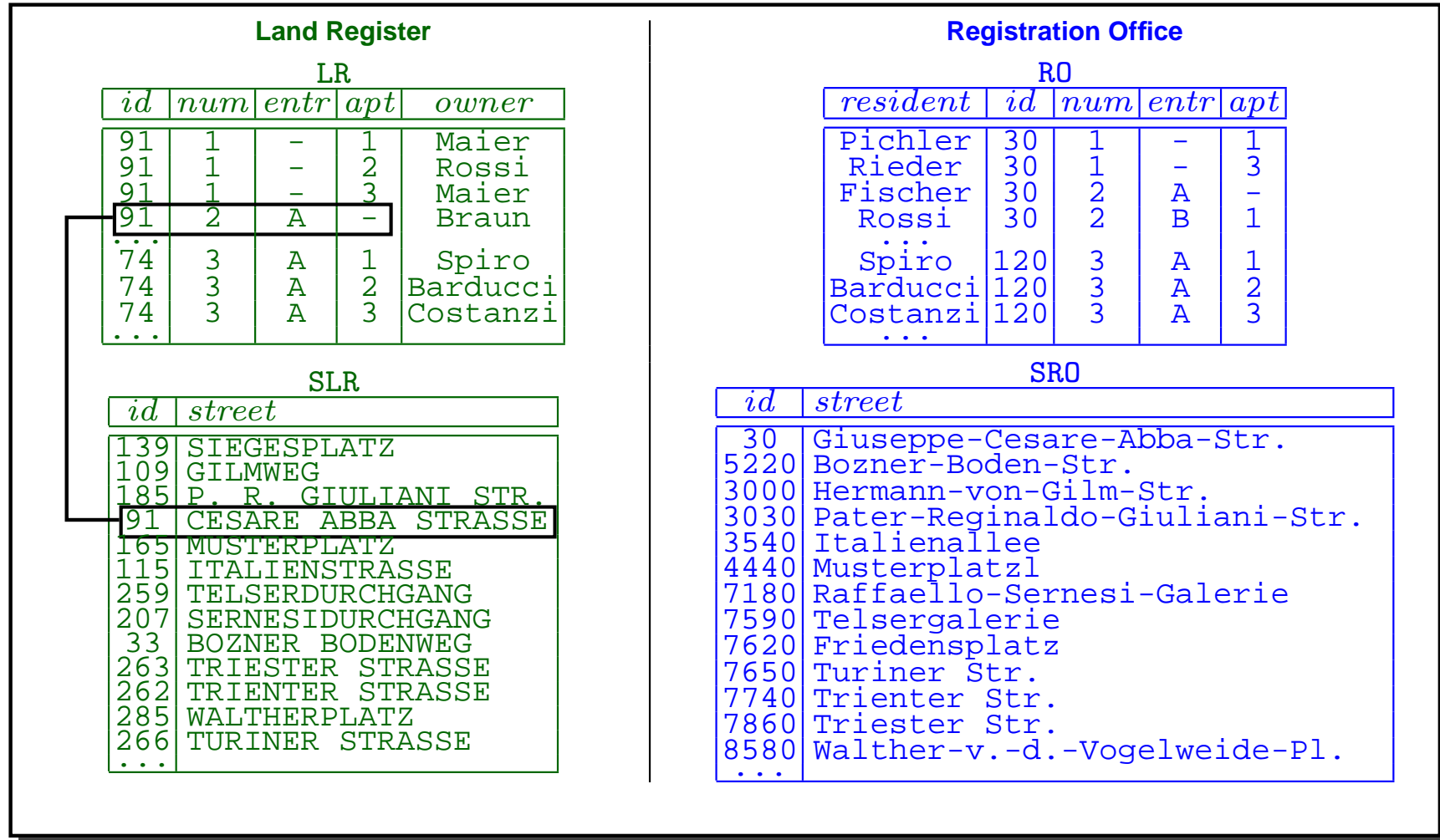
☞ We want to **link data items** in **different databases** that correspond to the **same real world object**.



# Motivation — Example Data Sources

➡ We want to **link data items in different databases** that correspond to the **same real world object**.

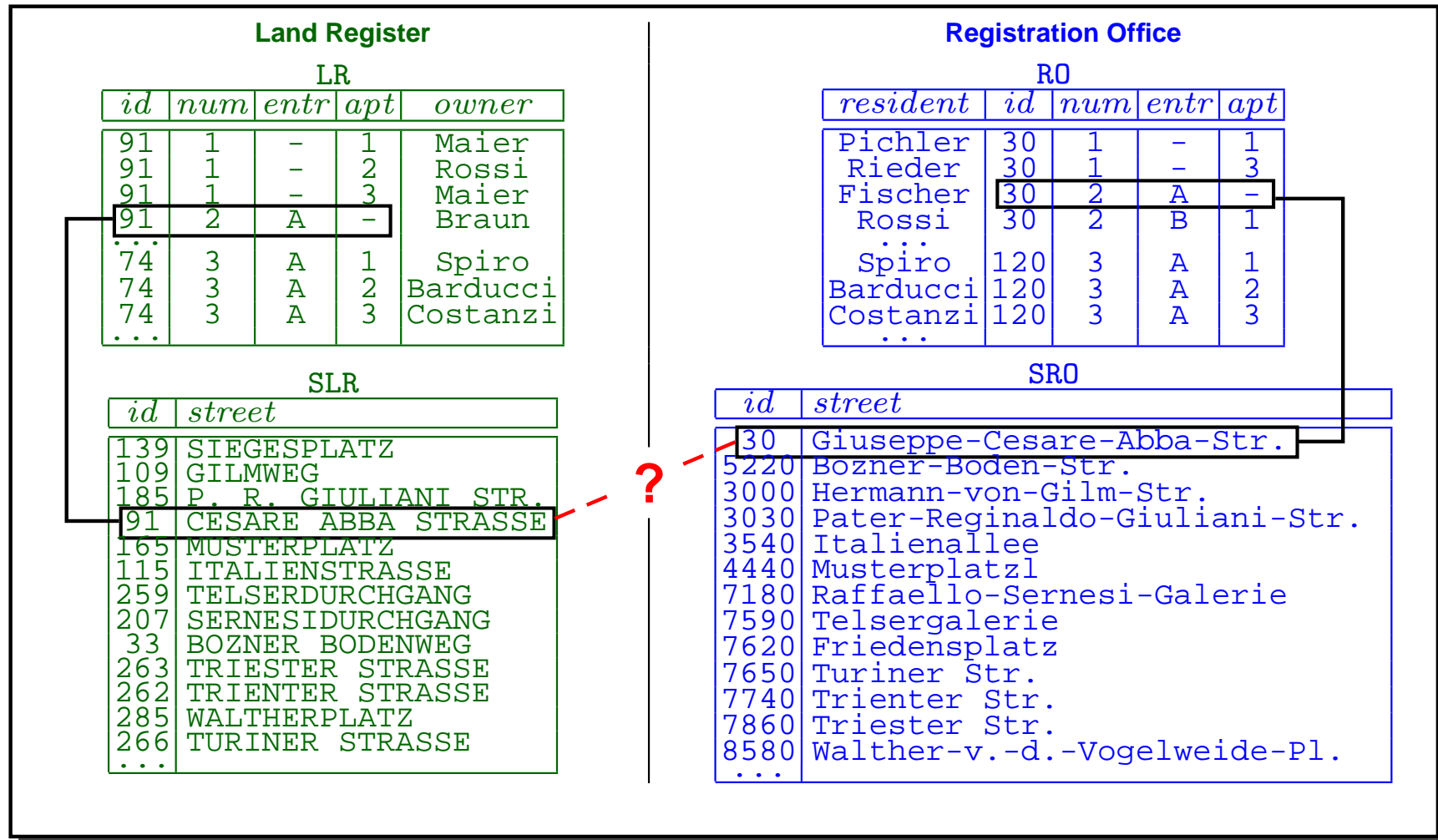
➡ **Example query:** Who lives in Braun's apartment?



# Motivation — Example Data Sources

➡ We want to **link data items** in **different databases** that correspond to the **same real world object**.

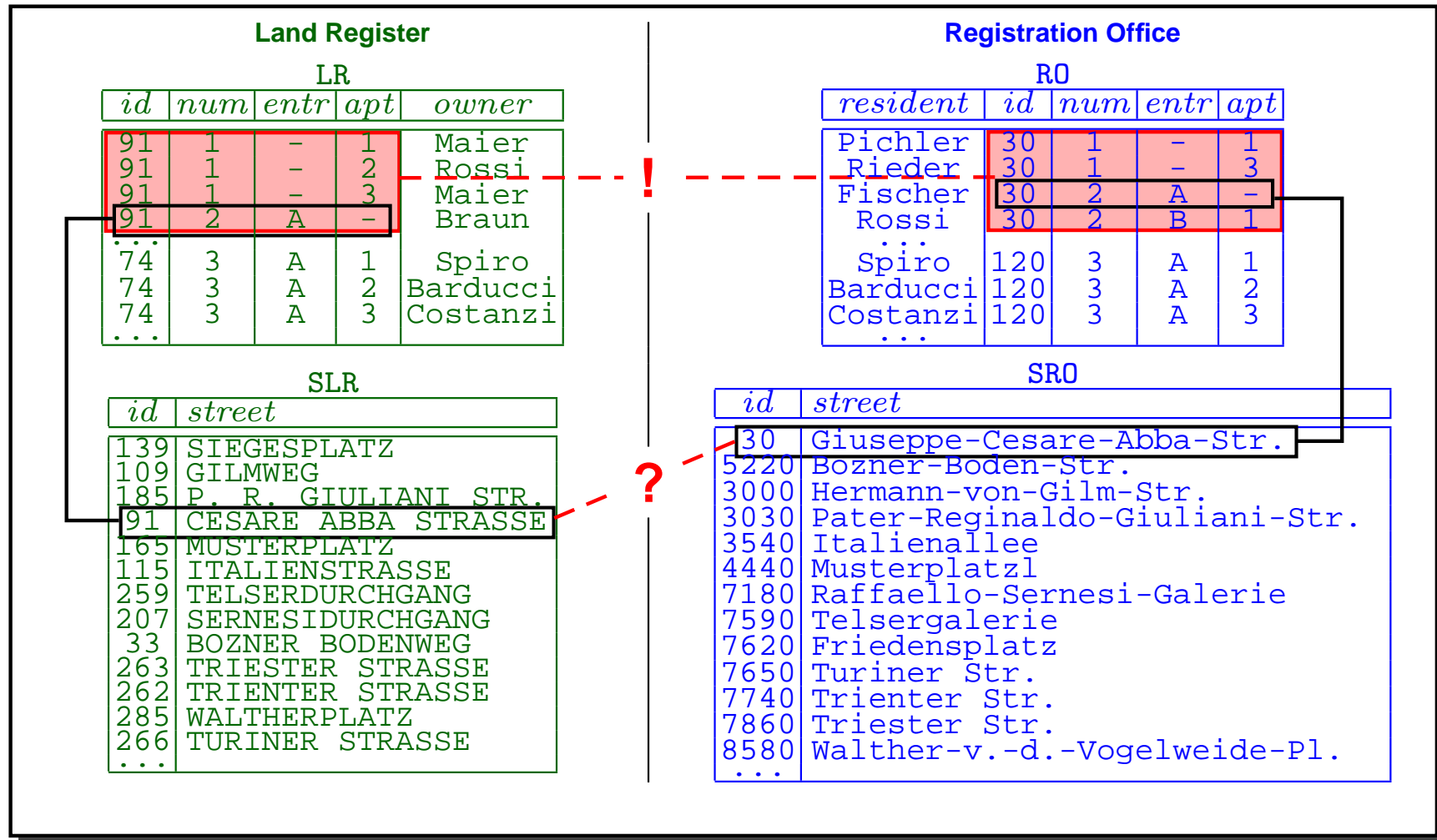
➡ **Example query:** Who lives in Braun's apartment?



# Motivation — Example Data Sources

➔ We want to **link data items in different databases** that correspond to the **same real world object**.

➔ **Example query:** Who lives in Braun's apartment?

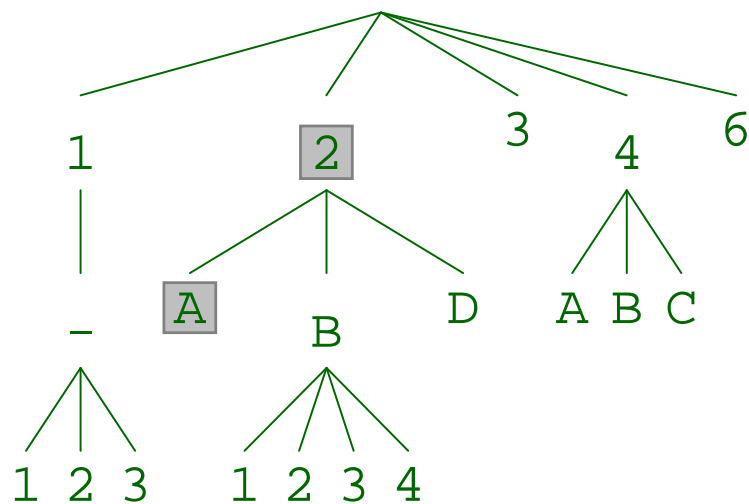


# Motivation — Address Trees

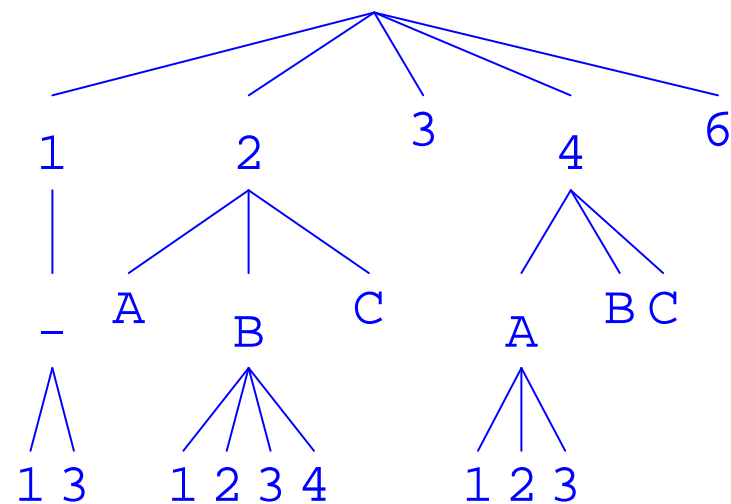
👉 residential addresses are hierarchical → **address tree**

**Address trees:**

CESARE ABBA STRASSE



Giuseppe-Cesare-Abba-Str.



# Motivation — Address Trees

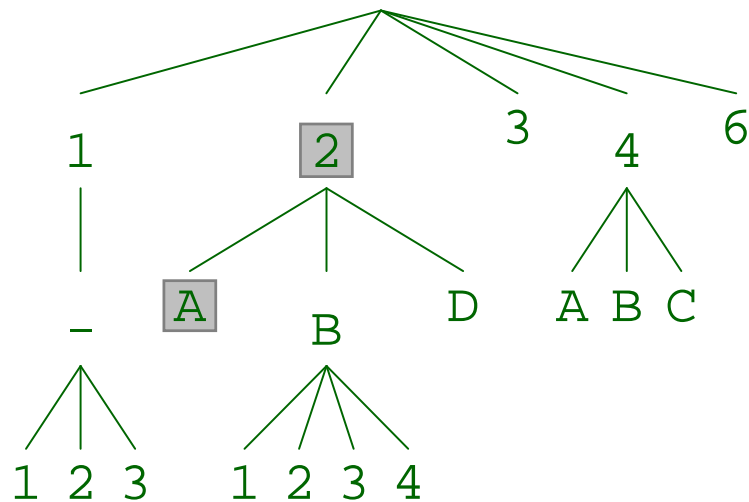
👉 residential addresses are hierarchical → **address tree**

👉 **Idea:** corresponding streets ⇒ similar address tree

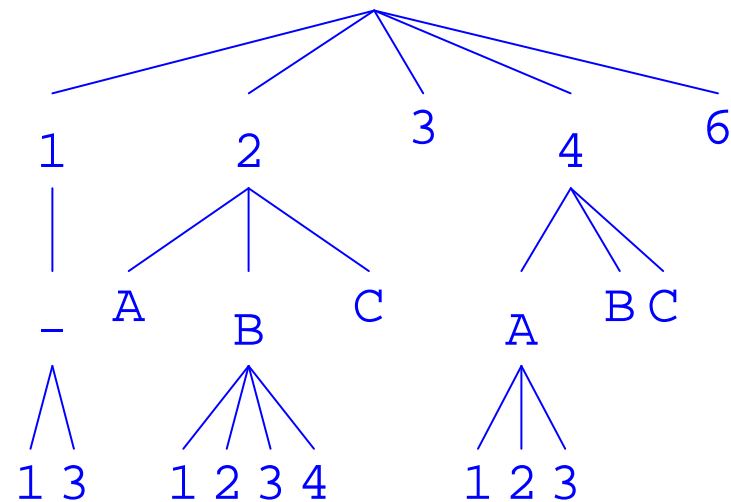
**How similar are two address trees?**

**Address trees:**

CESARE ABBA STRASSE



Giuseppe-Cesare-Abba-Str.





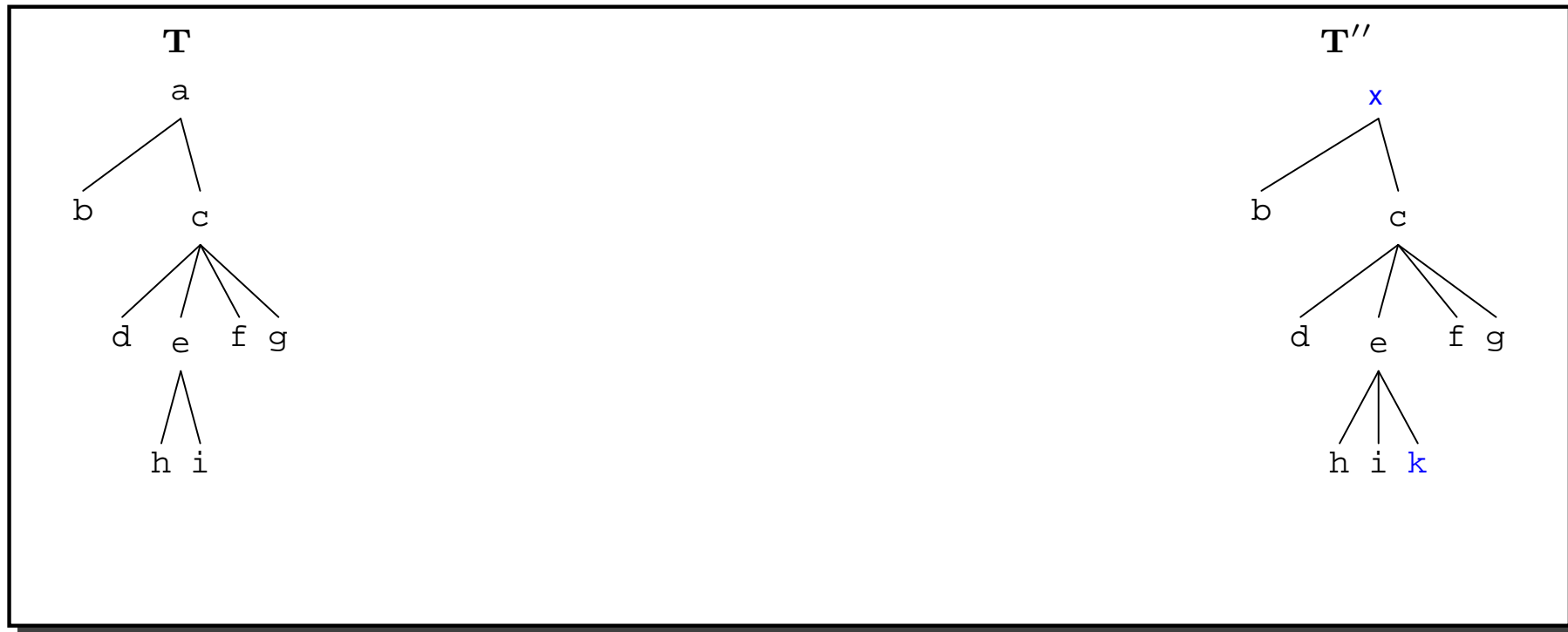
# Motivation — Standard Solution: The Edit Distance

---

➡ **Edit distance:** Minimum cost sequence of edit operations (node insertion, node deletion, and label change) that transform one tree into an other.

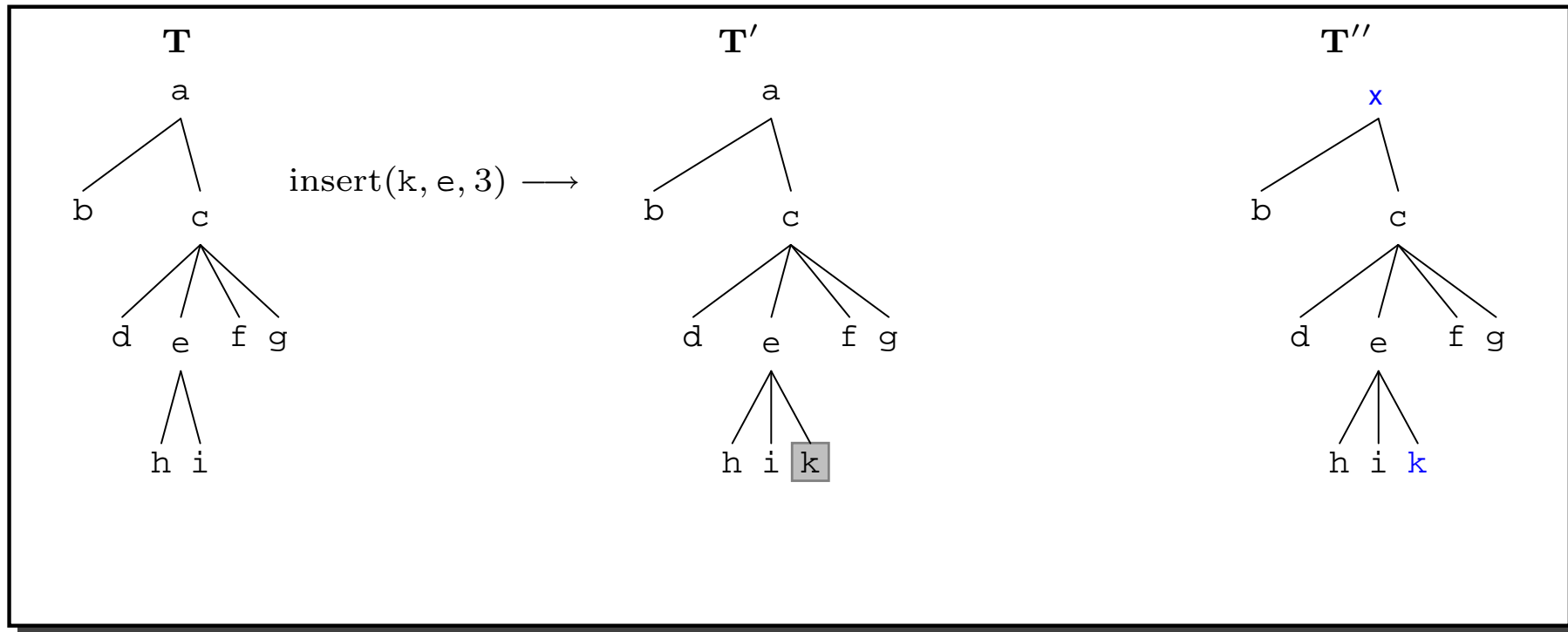
# Motivation — Standard Solution: The Edit Distance

➔ **Edit distance:** Minimum cost sequence of edit operations (node insertion, node deletion, and label change) that transform one tree into another.



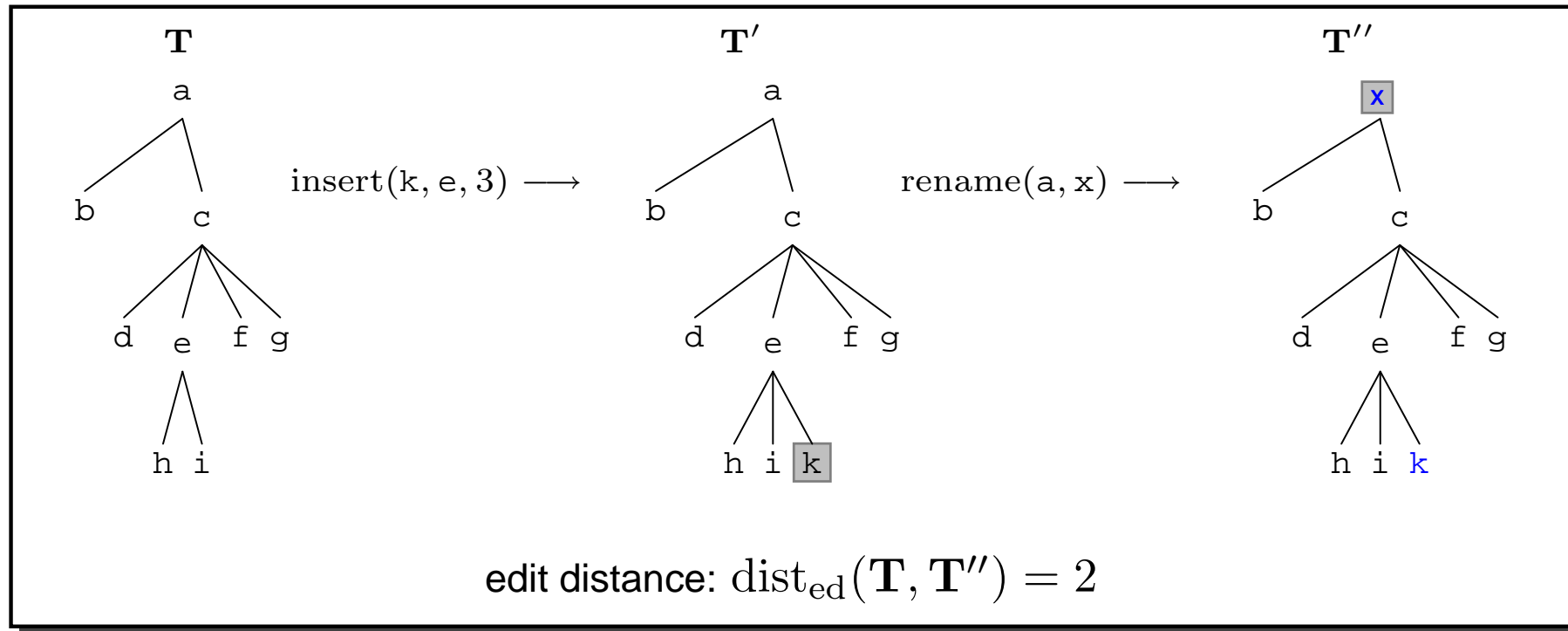
# Motivation — Standard Solution: The Edit Distance

➔ **Edit distance:** Minimum cost sequence of edit operations (node insertion, node deletion, and label change) that transform one tree into another.



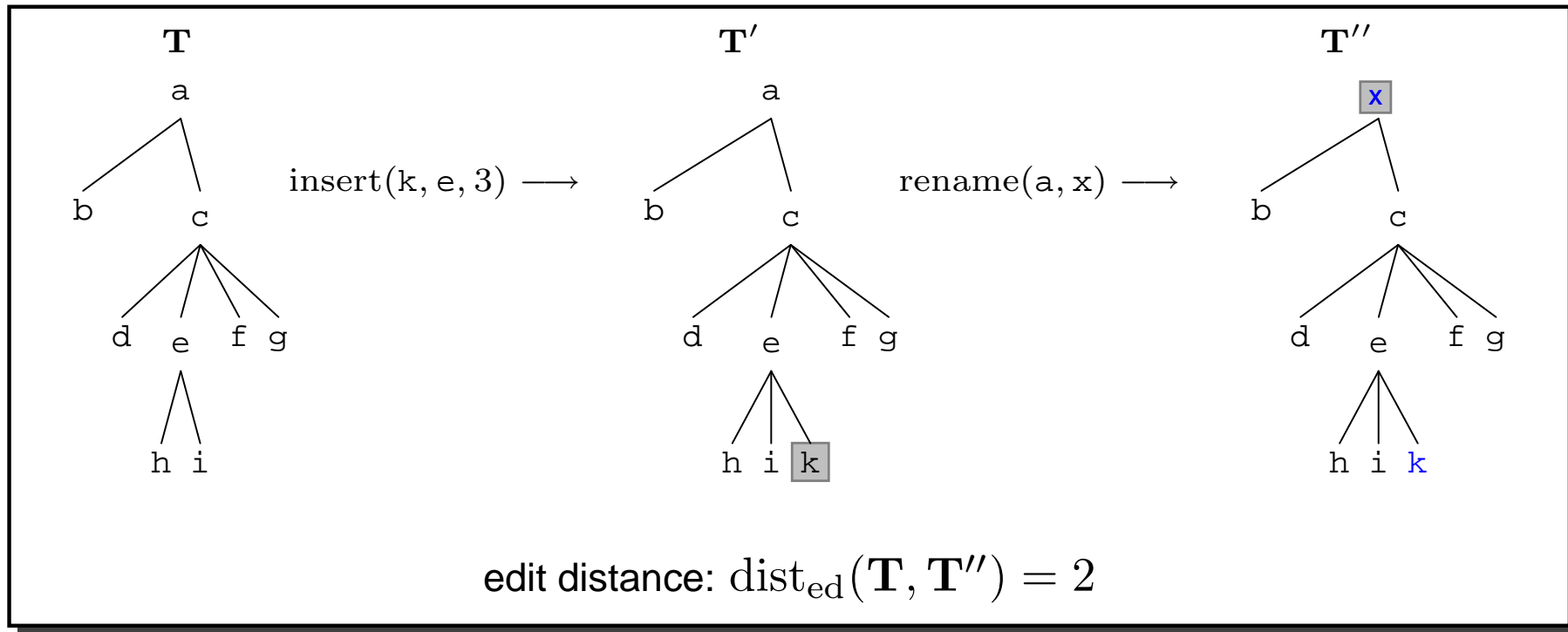
# Motivation — Standard Solution: The Edit Distance

➔ **Edit distance:** Minimum cost sequence of edit operations (node insertion, node deletion, and label change) that transform one tree into another.



# Motivation — Standard Solution: The Edit Distance

➡ **Edit distance:** Minimum cost sequence of edit operations (node insertion, node deletion, and label change) that transform one tree into another.



➡ **Problem:** Best algorithms  $O(n^2 \log^2(n)) \Rightarrow$  not scalable.

- ☞ **Our goal:** Find an **efficient** and **effective approximation** of the tree edit distance that
  - ▣► is **scalable** for large trees,
  - ▣► emphasizes **structure**.

# Related Work — Tree Distances

---

☞  $n \rightarrow$  number of tree nodes

☞ **Tree edit distance:**

⇒ for balanced trees [[Zhang and Shasha, 1989](#)]:  $O(n^2 \log^2(n))$

⇒ for arbitrary trees [[Klein, 1998](#)]:  $O(n^3 \log(n))$

☞  $n \rightarrow$  number of tree nodes

☞ **Tree edit distance:**

⇒ for balanced trees [Zhang and Shasha, 1989]:  $O(n^2 \log^2(n))$

⇒ for arbitrary trees [Klein, 1998]:  $O(n^3 \log(n))$

☞ **Tree edit distance approximations:**

⇒ Restricted versions of the tree edit distance:

▮ Alignment [Jiang et al., 1995]:  $O(n^2)$

▮ Isolated subtree [Tanaka and Tanaka, 1988]:  $O(n^2)$

▮ Top-down [Selkow, 1977, Yang, 1991]:  $O(n^2)$

▮ Bottom-up [Valiente, 2001]:  $O(n) \rightarrow$  only very specific domains



☞  $n$  → number of tree nodes

☞ **Tree edit distance:**

⇒ for balanced trees [Zhang and Shasha, 1989]:  $O(n^2 \log^2(n))$

⇒ for arbitrary trees [Klein, 1998]:  $O(n^3 \log(n))$

☞ **Tree edit distance approximations:**

⇒ Restricted versions of the tree edit distance:

▮ Alignment [Jiang et al., 1995]:  $O(n^2)$

▮ Isolated subtree [Tanaka and Tanaka, 1988]:  $O(n^2)$

▮ Top-down [Selkow, 1977, Yang, 1991]:  $O(n^2)$

▮ Bottom-up [Valiente, 2001]:  $O(n)$  → only very specific domains

⇒ XML versioning [Chawathe et al., 1996, Chawathe and Garcia-Molina, 1997, Lee et al., 2004]:  $O(n^2)$  for very different trees

# Related Work — Tree Distances

---

☞  $n$  → number of tree nodes

## ☞ **Tree edit distance:**

⇒ for balanced trees [Zhang and Shasha, 1989]:  $O(n^2 \log^2(n))$

⇒ for arbitrary trees [Klein, 1998]:  $O(n^3 \log(n))$

## ☞ **Tree edit distance approximations:**

⇒ Restricted versions of the tree edit distance:

▮ Alignment [Jiang et al., 1995]:  $O(n^2)$

▮ Isolated subtree [Tanaka and Tanaka, 1988]:  $O(n^2)$

▮ Top-down [Selkow, 1977, Yang, 1991]:  $O(n^2)$

▮ Bottom-up [Valiente, 2001]:  $O(n)$  → only very specific domains

⇒ XML versioning [Chawathe et al., 1996, Chawathe and Garcia-Molina, 1997, Lee et al., 2004]:  $O(n^2)$  for very different trees

⇒ Tree-edit distance embedding [Garofalakis and Kumar, 2003, Garofalakis and Kumar, 2005]:

▮  $O(n \log n)$

▮ guaranteed distance distortion for tree edit distance with subtree move

☞  $n \rightarrow$  number of tree nodes

## ☞ **Tree edit distance:**

⇒ for balanced trees [Zhang and Shasha, 1989]:  $O(n^2 \log^2(n))$

⇒ for arbitrary trees [Klein, 1998]:  $O(n^3 \log(n))$

## ☞ **Tree edit distance approximations:**

⇒ Restricted versions of the tree edit distance:

▮ Alignment [Jiang et al., 1995]:  $O(n^2)$

▮ Isolated subtree [Tanaka and Tanaka, 1988]:  $O(n^2)$

▮ Top-down [Selkow, 1977, Yang, 1991]:  $O(n^2)$

▮ Bottom-up [Valiente, 2001]:  $O(n) \rightarrow$  only very specific domains

⇒ XML versioning [Chawathe et al., 1996, Chawathe and Garcia-Molina, 1997, Lee et al., 2004]:  $O(n^2)$  for very different trees

⇒ Tree-edit distance embedding [Garofalakis and Kumar, 2003, Garofalakis and Kumar, 2005]:

▮  $O(n \log n)$

▮ guaranteed distance distortion for tree edit distance with subtree move

## ☞ **Related work for *strings*:**

⇒ Navarro [Navarro, 2001]: good overview of the edit distance for strings and its variants

⇒ Ukkonen [Ukkonen, 1992]:  $q$ -grams as lower bound for string edit distance

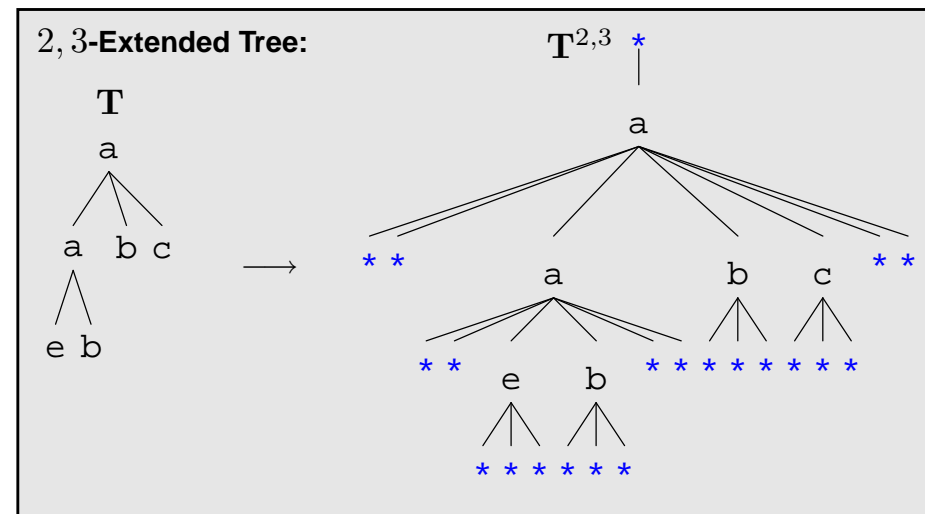
## Extended Tree $T^{pq}$ :

Patch boundaries by adding null nodes (\*):

$\Rightarrow p - 1$  ancestors to the root

$\Rightarrow q - 1$  nodes before the first and after the last child of each non-leaf node

$\Rightarrow q$  children to each leaf



# $pq$ -Grams — Subtrees of the $pq$ -Extended Tree

## Extended Tree $T^{pq}$ :

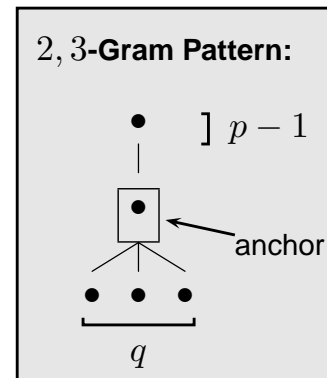
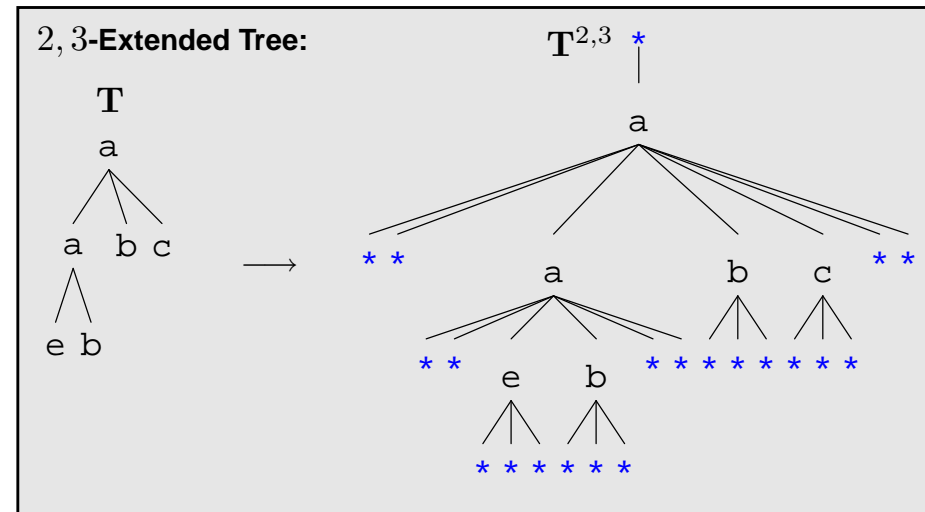
Patch boundaries by adding null nodes (\*):

- $\Rightarrow p - 1$  ancestors to the root
- $\Rightarrow q - 1$  nodes before the first and after the last child of each non-leaf node
- $\Rightarrow q$  children to each leaf

## $pq$ -Gram $G$ : Subtree of $T^{pq}$ .

- $\Rightarrow$  Anchor node
- $\Rightarrow$  with  $p - 1$  ancestors
- $\Rightarrow$  and  $q$  children.

Contiguous siblings in  $G$  are contiguous siblings in  $T^{pq}$ .



# $pq$ -Grams — Subtrees of the $pq$ -Extended Tree

## Extended Tree $T^{pq}$ :

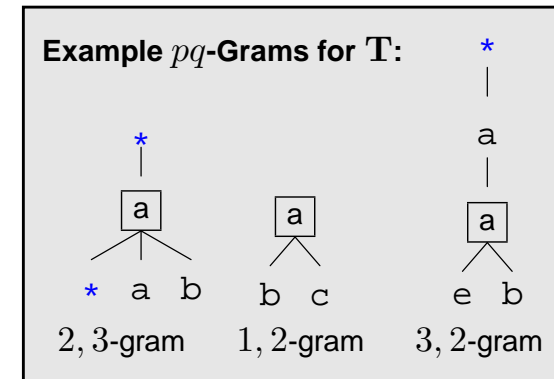
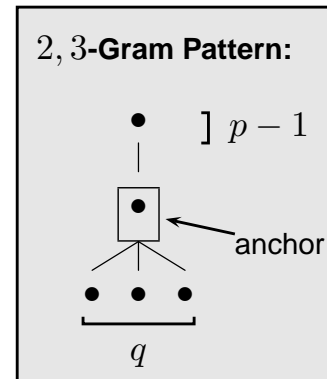
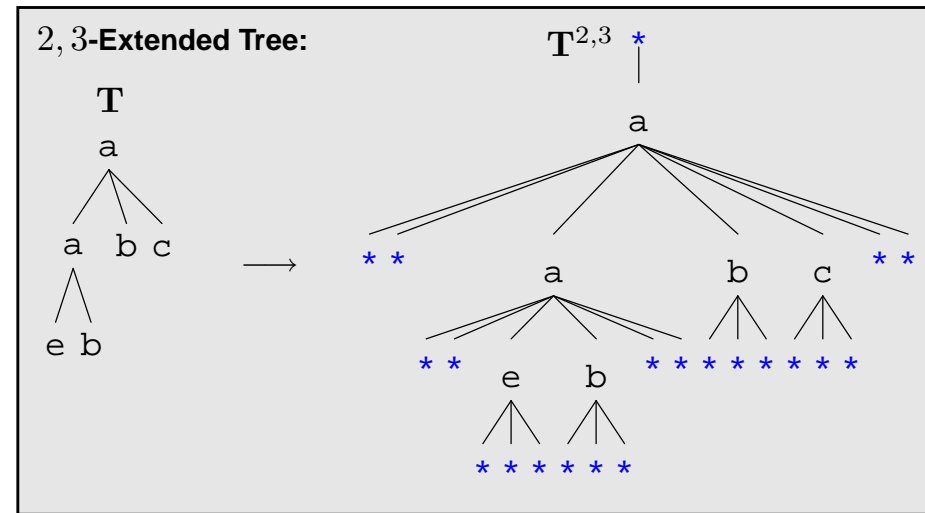
Patch boundaries by adding null nodes (\*):

- $\Rightarrow p - 1$  ancestors to the root
- $\Rightarrow q - 1$  nodes before the first and after the last child of each non-leaf node
- $\Rightarrow q$  children to each leaf

## $pq$ -Gram $G$ : Subtree of $T^{pq}$ .

- $\Rightarrow$  Anchor node
- $\Rightarrow$  with  $p - 1$  ancestors
- $\Rightarrow$  and  $q$  children.

Contiguous siblings in  $G$  are contiguous siblings in  $T^{pq}$ .



# pq-Grams — Subtrees of the pq-Extended Tree

## Extended Tree $T^{pq}$ :

Patch boundaries by adding null nodes (\*):

- $\Rightarrow p - 1$  ancestors to the root
- $\Rightarrow q - 1$  nodes before the first and after the last child of each non-leaf node
- $\Rightarrow q$  children to each leaf

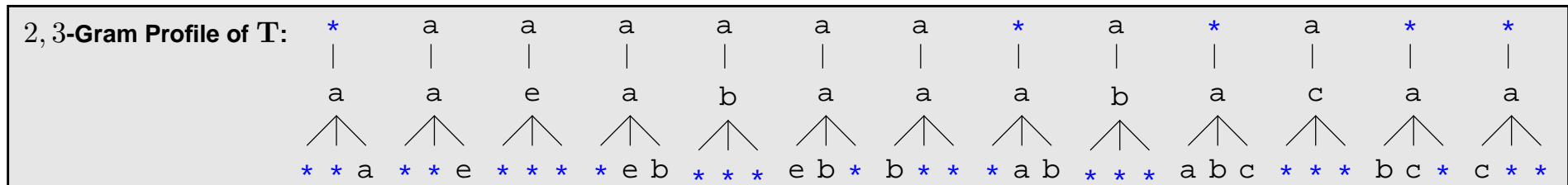
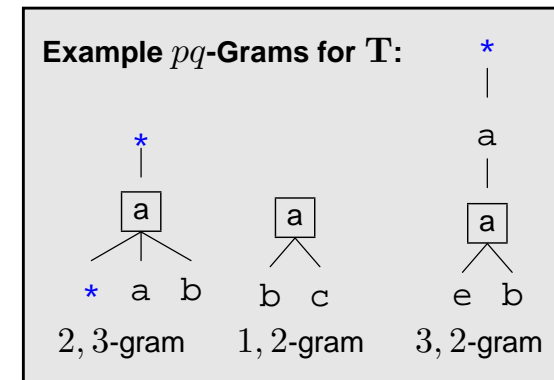
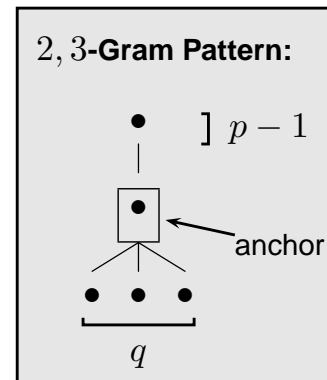
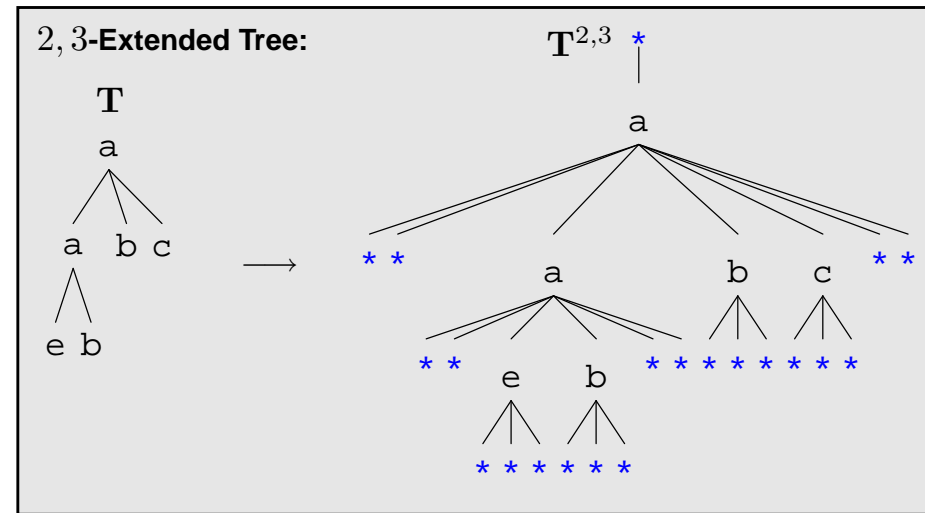
## pq-Gram $G$ : Subtree of $T^{pq}$ .

- $\Rightarrow$  Anchor node
- $\Rightarrow$  with  $p - 1$  ancestors
- $\Rightarrow$  and  $q$  children.

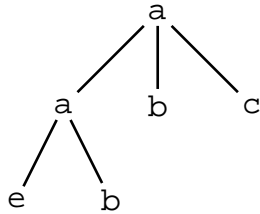
Contiguous siblings in  $G$  are contiguous siblings in  $T^{pq}$ .

## pq-gram Profile $P^{p,q}(T)$ :

- $\Rightarrow$  Bag of all pq-grams of  $T$ .



**T**



*P*

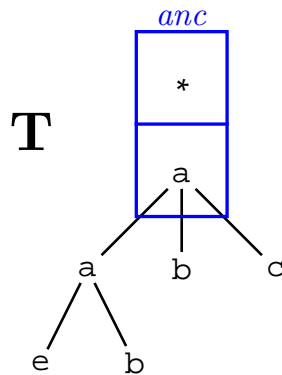
```
1 CREATEPROFILE(T, p, q, P, r, anc)
2   anc := shift(anc, l(r))
3   sib: shift register of size q (initialized with *)
4
5   if r is a leaf then
6     P := P ∪ (anc ∘ sib)
7   else
8     for each child c (from left to right) of r do
9       sib := shift(sib, l(c))
10      P := P ∪ (anc ∘ sib)
11      P := PROFILE(T, p, q, P, c, anc)
12     for k := 1 to q − 1
13       sib := shift(sib, *)
14       P := P ∪ (anc ∘ sib)
15   return P
```



# *pq*-Grams — Algorithm for *pq*-Gram Profile

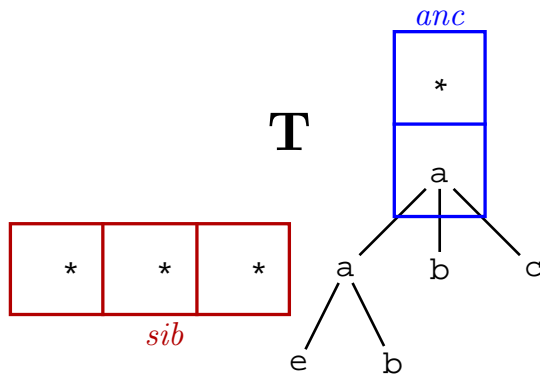


```
1 CREATEPROFILE(T, p, q, P, r, anc)
2   anc := shift(anc, l(r))
3   sib: shift register of size q (initialized with *)
4
5   if r is a leaf then
6     P := P ∪ (anc ∘ sib)
7   else
8     for each child c (from left to right) of r do
9       sib := shift(sib, l(c))
10      P := P ∪ (anc ∘ sib)
11      P := PROFILE(T, p, q, P, c, anc)
12     for k := 1 to q - 1
13       sib := shift(sib, *)
14       P := P ∪ (anc ∘ sib)
15   return P
```



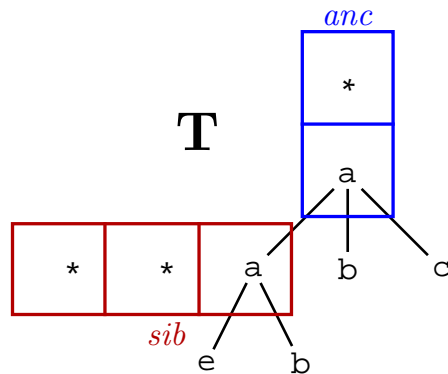
```
1 CREATEPROFILE(T, p, q, P, r, anc)
2 anc := shift(anc, l(r))
3 sib: shift register of size q (initialized with *)
4
5 if r is a leaf then
6   P := P ∪ (anc ∘ sib)
7 else
8   for each child c (from left to right) of r do
9     sib := shift(sib, l(c))
10    P := P ∪ (anc ∘ sib)
11    P := PROFILE(T, p, q, P, c, anc)
12  for k := 1 to q - 1
13    sib := shift(sib, *)
14    P := P ∪ (anc ∘ sib)
15 return P
```

# $pq$ -Grams — Algorithm for $pq$ -Gram Profile



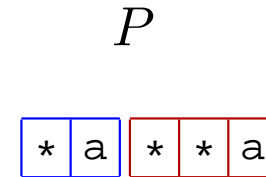
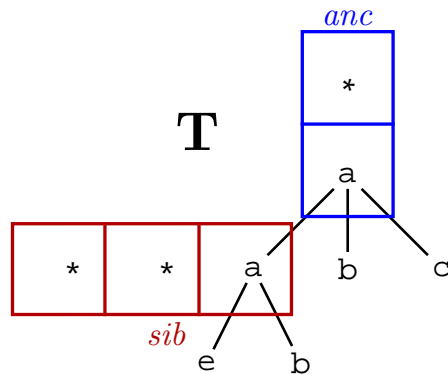
```
1 CREATEPROFILE( $\mathbf{T}$ ,  $p$ ,  $q$ ,  $P$ ,  $r$ ,  $anc$ )
2    $anc := \text{shift}(anc, l(r))$ 
3    $sib$ : shift register of size  $q$  (initialized with  $*$ )
4
5   if  $r$  is a leaf then
6      $P := P \cup (anc \circ sib)$ 
7   else
8     for each child  $c$  (from left to right) of  $r$  do
9        $sib := \text{shift}(sib, l(c))$ 
10       $P := P \cup (anc \circ sib)$ 
11       $P := \text{PROFILE}(\mathbf{T}, p, q, P, c, anc)$ 
12    for  $k := 1$  to  $q - 1$ 
13       $sib := \text{shift}(sib, *)$ 
14       $P := P \cup (anc \circ sib)$ 
15  return  $P$ 
```

# $pq$ -Grams — Algorithm for $pq$ -Gram Profile



```
1 CREATEPROFILE( $\mathbf{T}$ ,  $p$ ,  $q$ ,  $P$ ,  $r$ ,  $anc$ )
2    $anc := \text{shift}(anc, l(r))$ 
3    $sib$ : shift register of size  $q$  (initialized with  $*$ )
4
5   if  $r$  is a leaf then
6      $P := P \cup (anc \circ sib)$ 
7   else
8     for each child  $c$  (from left to right) of  $r$  do
9      $sib := \text{shift}(sib, l(c))$ 
10     $P := P \cup (anc \circ sib)$ 
11     $P := \text{PROFILE}(\mathbf{T}, p, q, P, c, anc)$ 
12    for  $k := 1$  to  $q - 1$ 
13       $sib := \text{shift}(sib, *)$ 
14       $P := P \cup (anc \circ sib)$ 
15  return  $P$ 
```

# *pq*-Grams — Algorithm for *pq*-Gram Profile

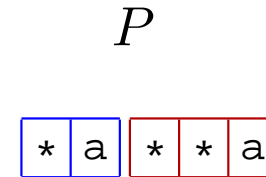
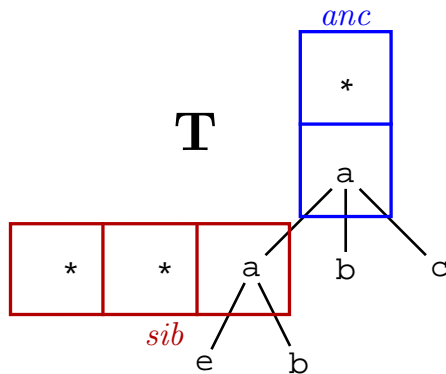


```

1 CREATEPROFILE(T, p, q, P, r, anc)
2   anc := shift(anc, l(r))
3   sib: shift register of size q (initialized with *)
4
5   if r is a leaf then
6     P := P ∪ (anc ∘ sib)
7   else
8     for each child c (from left to right) of r do
9       sib := shift(sib, l(c))
10      P := P ∪ (anc ∘ sib)
11      P := PROFILE(T, p, q, P, c, anc)
12     for k := 1 to q - 1
13       sib := shift(sib, *)
14       P := P ∪ (anc ∘ sib)
15   return P

```

# *pq*-Grams — Algorithm for *pq*-Gram Profile

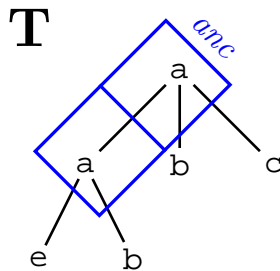


```

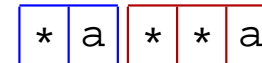
1 CREATEPROFILE(T, p, q, P, r, anc)
2   anc := shift(anc, l(r))
3   sib: shift register of size q (initialized with *)
4
5   if r is a leaf then
6     P := P ∪ (anc ∘ sib)
7   else
8     for each child c (from left to right) of r do
9       sib := shift(sib, l(c))
10      P := P ∪ (anc ∘ sib)
11     P := PROFILE(T, p, q, P, c, anc)
12     for k := 1 to q - 1
13       sib := shift(sib, *)
14       P := P ∪ (anc ∘ sib)
15   return P

```





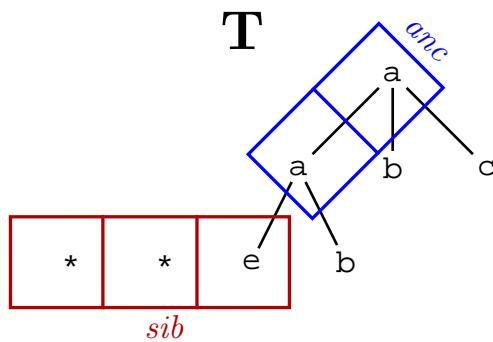
**P**



```

1 CREATEPROFILE(T, p, q, P, r, anc)
2  anc := shift(anc, l(r))
3  sib: shift register of size q (initialized with *)
4
5  if r is a leaf then
6    P := P ∪ (anc ◦ sib)
7  else
8    for each child c (from left to right) of r do
9      sib := shift(sib, l(c))
10     P := P ∪ (anc ◦ sib)
11     P := PROFILE(T, p, q, P, c, anc)
12   for k := 1 to q - 1
13     sib := shift(sib, *)
14     P := P ∪ (anc ◦ sib)
15  return P
    
```

# pq-Grams — Algorithm for pq-Gram Profile

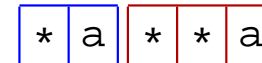


```

1 CREATEPROFILE(T, p, q, P, r, anc)
2   anc := shift(anc, l(r))
3   sib: shift register of size q (initialized with *)
4
5   if r is a leaf then
6     P := P ∪ (anc ∘ sib)
7   else
8     for each child c (from left to right) of r do
9     sib := shift(sib, l(c))
10    P := P ∪ (anc ∘ sib)
11    P := PROFILE(T, p, q, P, c, anc)
12    for k := 1 to q - 1
13      sib := shift(sib, *)
14      P := P ∪ (anc ∘ sib)
15  return P

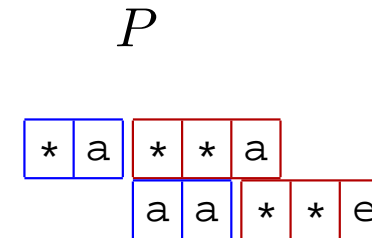
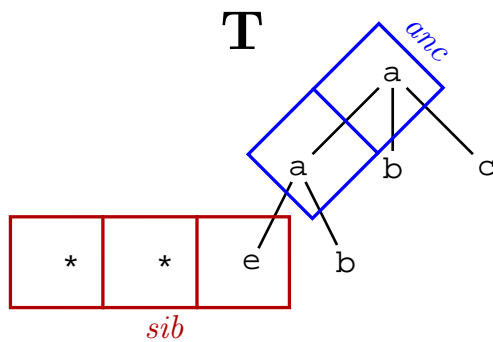
```

*P*





# *pq*-Grams — Algorithm for *pq*-Gram Profile

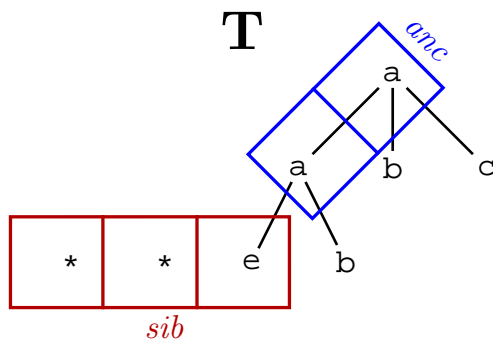


```

1 CREATEPROFILE(T, p, q, P, r, anc)
2   anc := shift(anc, l(r))
3   sib: shift register of size q (initialized with *)
4
5   if r is a leaf then
6     P := P ∪ (anc ∘ sib)
7   else
8     for each child c (from left to right) of r do
9       sib := shift(sib, l(c))
10      P := P ∪ (anc ∘ sib)
11      P := PROFILE(T, p, q, P, c, anc)
12     for k := 1 to q - 1
13       sib := shift(sib, *)
14       P := P ∪ (anc ∘ sib)
15   return P

```

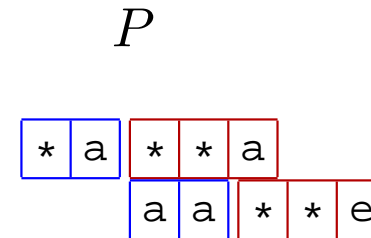
# pq-Grams — Algorithm for pq-Gram Profile

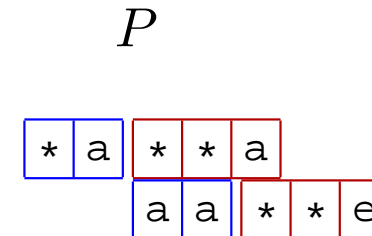
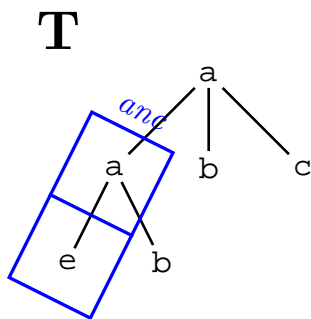


```

1 CREATEPROFILE(T, p, q, P, r, anc)
2   anc := shift(anc, l(r))
3   sib: shift register of size q (initialized with *)
4
5   if r is a leaf then
6     P := P ∪ (anc ∘ sib)
7   else
8     for each child c (from left to right) of r do
9       sib := shift(sib, l(c))
10      P := P ∪ (anc ∘ sib)
11     P := PROFILE(T, p, q, P, c, anc)
12     for k := 1 to q - 1
13       sib := shift(sib, *)
14       P := P ∪ (anc ∘ sib)
15   return P

```

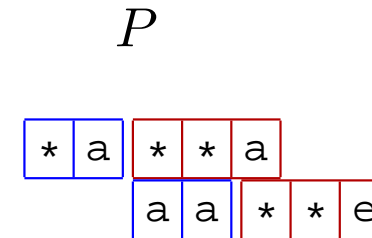
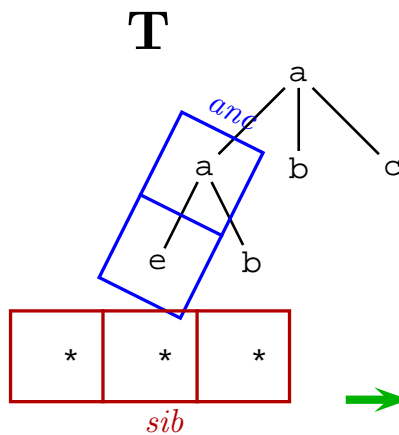




```

1 CREATEPROFILE(T, p, q, P, r, anc)
2  anc := shift(anc, l(r))
3  sib: shift register of size q (initialized with *)
4
5  if r is a leaf then
6    P := P ∪ (anc ∘ sib)
7  else
8    for each child c (from left to right) of r do
9      sib := shift(sib, l(c))
10     P := P ∪ (anc ∘ sib)
11     P := PROFILE(T, p, q, P, c, anc)
12   for k := 1 to q - 1
13     sib := shift(sib, *)
14     P := P ∪ (anc ∘ sib)
15   return P
    
```

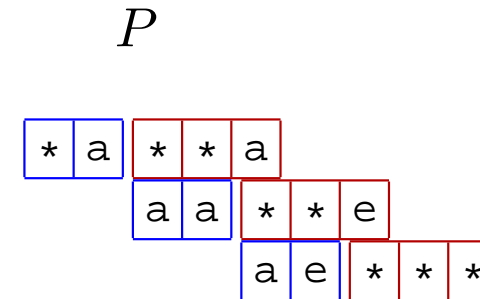
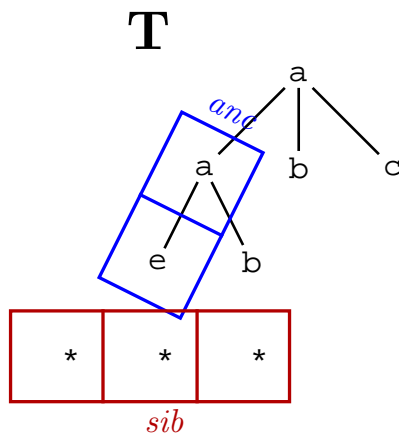
# pq-Grams — Algorithm for pq-Gram Profile



```

1 CREATEPROFILE(T, p, q, P, r, anc)
2   anc := shift(anc, l(r))
3   sib: shift register of size q (initialized with *)
4
5   if r is a leaf then
6     P := P ∪ (anc ∘ sib)
7   else
8     for each child c (from left to right) of r do
9       sib := shift(sib, l(c))
10      P := P ∪ (anc ∘ sib)
11      P := PROFILE(T, p, q, P, c, anc)
12     for k := 1 to q - 1
13       sib := shift(sib, *)
14       P := P ∪ (anc ∘ sib)
15   return P
  
```

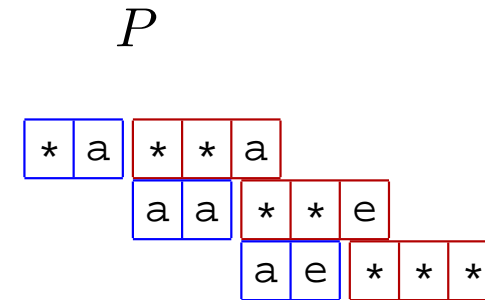
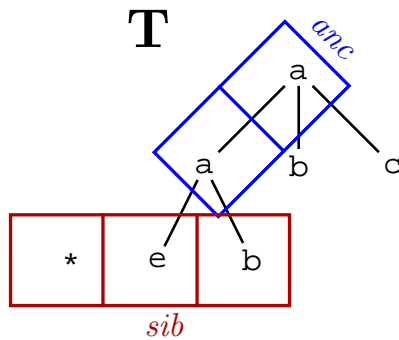
# pq-Grams — Algorithm for pq-Gram Profile



```

1 CREATEPROFILE(T, p, q, P, r, anc)
2   anc := shift(anc, l(r))
3   sib: shift register of size q (initialized with *)
4
5   if r is a leaf then
6     P := P ∪ (anc ∘ sib)
7   else
8     for each child c (from left to right) of r do
9       sib := shift(sib, l(c))
10      P := P ∪ (anc ∘ sib)
11      P := PROFILE(T, p, q, P, c, anc)
12     for k := 1 to q - 1
13       sib := shift(sib, *)
14       P := P ∪ (anc ∘ sib)
15   return P
    
```

# pq-Grams — Algorithm for pq-Gram Profile

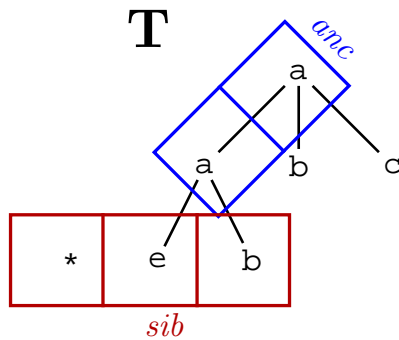


```

1 CREATEPROFILE(T, p, q, P, r, anc)
2   anc := shift(anc, l(r))
3   sib: shift register of size q (initialized with *)
4
5   if r is a leaf then
6     P := P ∪ (anc ∘ sib)
7   else
8     for each child c (from left to right) of r do
9     sib := shift(sib, l(c))
10    P := P ∪ (anc ∘ sib)
11    P := PROFILE(T, p, q, P, c, anc)
12    for k := 1 to q - 1
13      sib := shift(sib, *)
14      P := P ∪ (anc ∘ sib)
15  return P

```

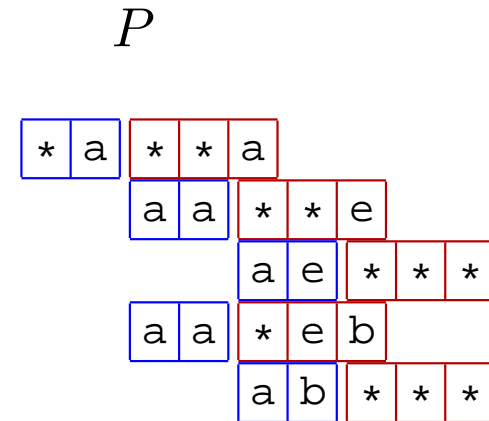
# pq-Grams — Algorithm for pq-Gram Profile



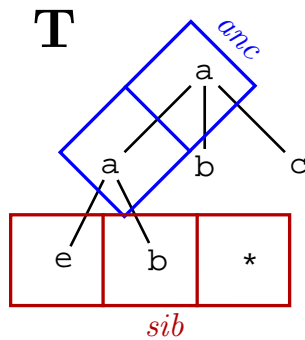
```

1 CREATEPROFILE(T, p, q, P, r, anc)
2   anc := shift(anc, l(r))
3   sib: shift register of size q (initialized with *)
4
5   if r is a leaf then
6     P := P ∪ (anc ∘ sib)
7   else
8     for each child c (from left to right) of r do
9       sib := shift(sib, l(c))
10      P := P ∪ (anc ∘ sib)
11      P := PROFILE(T, p, q, P, c, anc)
12     for k := 1 to q - 1
13       sib := shift(sib, *)
14       P := P ∪ (anc ∘ sib)
15   return P

```



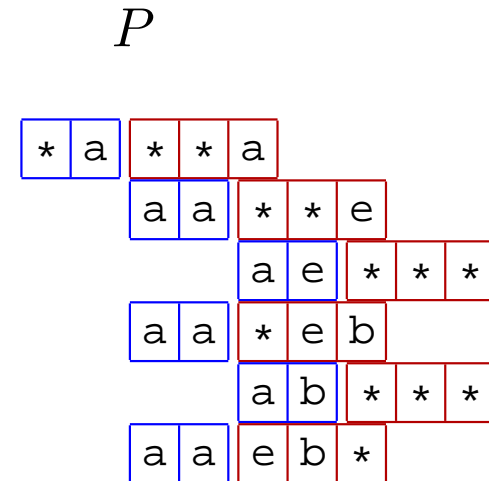
# pq-Grams — Algorithm for pq-Gram Profile



```

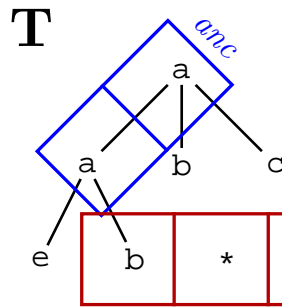
1 CREATEPROFILE(T, p, q, P, r, anc)
2   anc := shift(anc, l(r))
3   sib: shift register of size q (initialized with *)
4
5   if r is a leaf then
6     P := P ∪ (anc ∘ sib)
7   else
8     for each child c (from left to right) of r do
9       sib := shift(sib, l(c))
10      P := P ∪ (anc ∘ sib)
11      P := PROFILE(T, p, q, P, c, anc)
12      for k := 1 to q - 1
13        sib := shift(sib, *)
14      P := P ∪ (anc ∘ sib)
15  return P

```





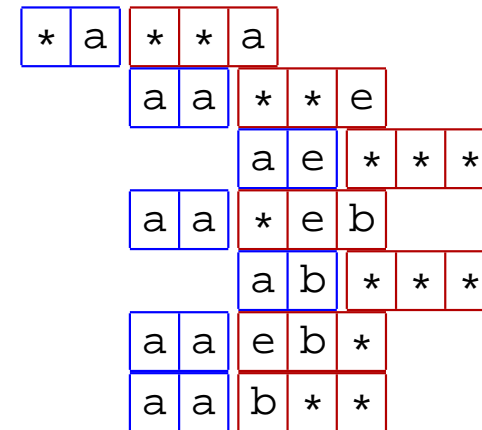
# pq-Grams — Algorithm for pq-Gram Profile



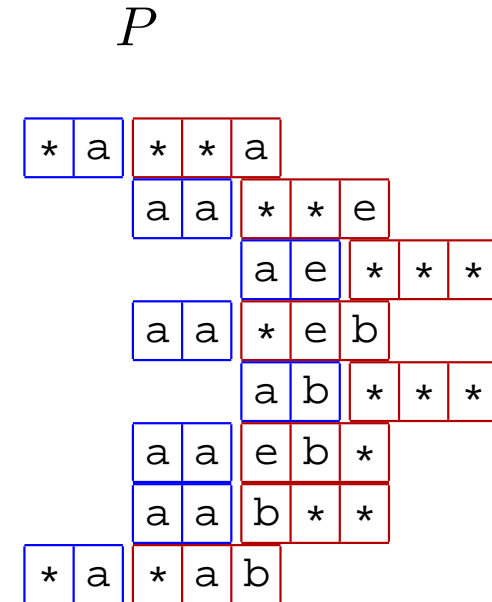
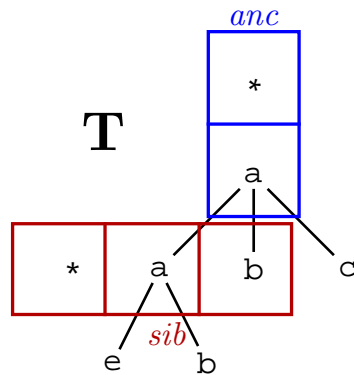
```

1 CREATEPROFILE(T, p, q, P, r, anc)
2   anc := shift(anc, l(r))
3   sib: shift register of size q (initialized with *)
4
5   if r is a leaf then
6     P := P ∪ (anc ∘ sib)
7   else
8     for each child c (from left to right) of r do
9       sib := shift(sib, l(c))
10      P := P ∪ (anc ∘ sib)
11      P := PROFILE(T, p, q, P, c, anc)
12      for k := 1 to q - 1
13        sib := shift(sib, *)
14      P := P ∪ (anc ∘ sib)
15  return P
  
```

*P*



# pq-Grams — Algorithm for pq-Gram Profile



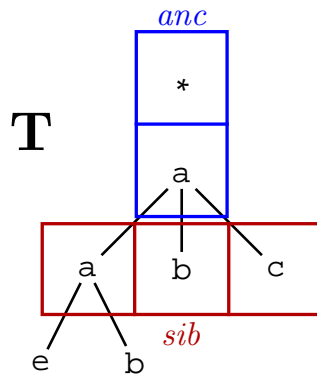
```

1 CREATEPROFILE(T, p, q, P, r, anc)
2   anc := shift(anc, l(r))
3   sib: shift register of size q (initialized with *)
4
5   if r is a leaf then
6     P := P ∪ (anc ∘ sib)
7   else
8     for each child c (from left to right) of r do
9       sib := shift(sib, l(c))
10      P := P ∪ (anc ∘ sib)
11      P := PROFILE(T, p, q, P, c, anc)
12      for k := 1 to q - 1
13        sib := shift(sib, *)
14        P := P ∪ (anc ∘ sib)
15  return P

```



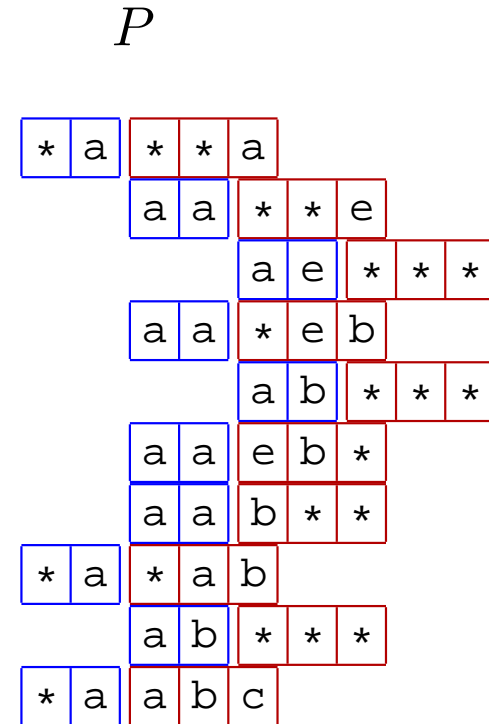
# pq-Grams — Algorithm for pq-Gram Profile



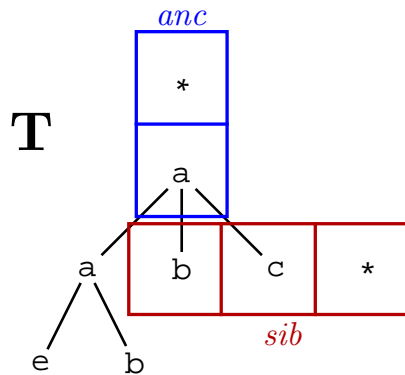
```

1 CREATEPROFILE(T, p, q, P, r, anc)
2   anc := shift(anc, l(r))
3   sib: shift register of size q (initialized with *)
4
5   if r is a leaf then
6     P := P ∪ (anc ∘ sib)
7   else
8     for each child c (from left to right) of r do
9       sib := shift(sib, l(c))
10      P := P ∪ (anc ∘ sib)
11      P := PROFILE(T, p, q, P, c, anc)
12      for k := 1 to q - 1
13        sib := shift(sib, *)
14        P := P ∪ (anc ∘ sib)
15  return P

```



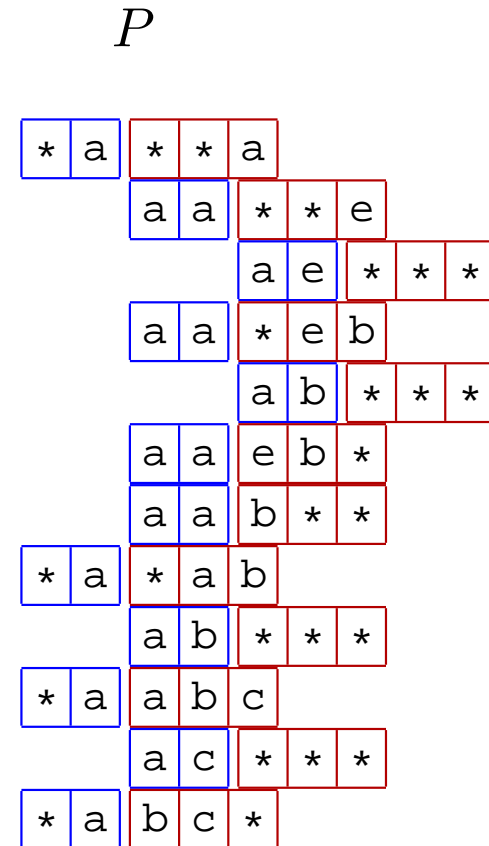
# pq-Grams — Algorithm for pq-Gram Profile



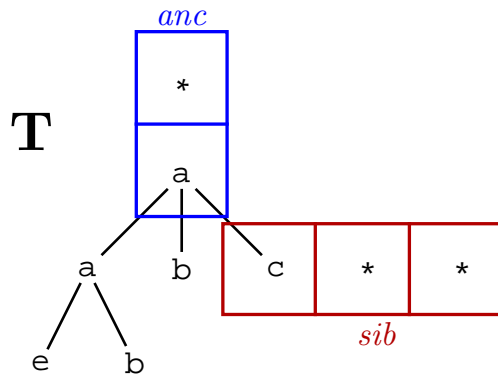
```

1 CREATEPROFILE(T, p, q, P, r, anc)
2   anc := shift(anc, l(r))
3   sib: shift register of size q (initialized with *)
4
5   if r is a leaf then
6     P := P ∪ (anc ∘ sib)
7   else
8     for each child c (from left to right) of r do
9       sib := shift(sib, l(c))
10      P := P ∪ (anc ∘ sib)
11      P := PROFILE(T, p, q, P, c, anc)
12      for k := 1 to q - 1
13        sib := shift(sib, *)
14      P := P ∪ (anc ∘ sib)
15  return P

```

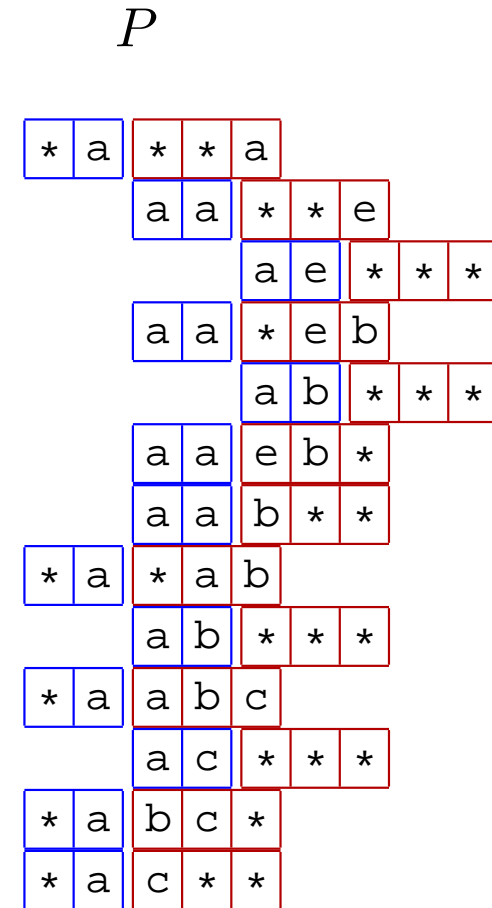


# pq-Grams — Algorithm for pq-Gram Profile

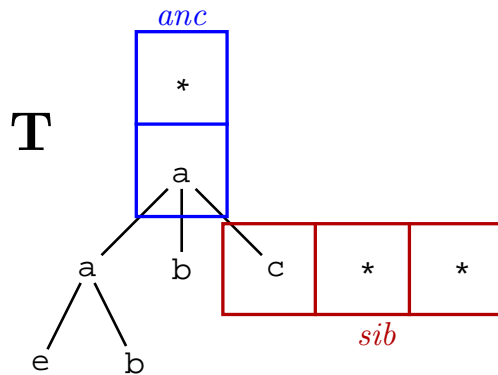


```

1 CREATEPROFILE(T, p, q, P, r, anc)
2   anc := shift(anc, l(r))
3   sib: shift register of size q (initialized with *)
4
5   if r is a leaf then
6     P := P ∪ (anc ∘ sib)
7   else
8     for each child c (from left to right) of r do
9       sib := shift(sib, l(c))
10      P := P ∪ (anc ∘ sib)
11      P := PROFILE(T, p, q, P, c, anc)
12      for k := 1 to q - 1
13        sib := shift(sib, *)
14      P := P ∪ (anc ∘ sib)
15  return P
  
```



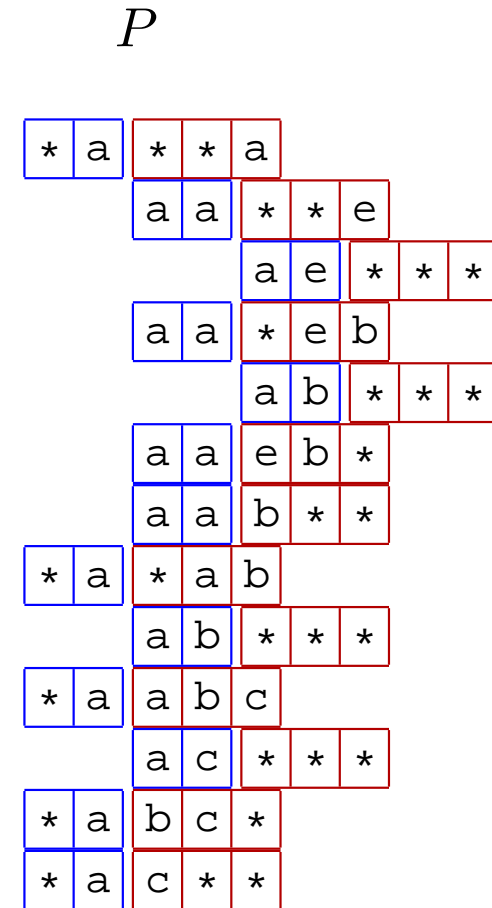
# pq-Grams — Algorithm for pq-Gram Profile



```

1 CREATEPROFILE(T, p, q, P, r, anc)
2   anc := shift(anc, l(r))
3   sib: shift register of size q (initialized with *)
4
5   if r is a leaf then
6     P := P ∪ (anc ∘ sib)
7   else
8     for each child c (from left to right) of r do
9       sib := shift(sib, l(c))
10      P := P ∪ (anc ∘ sib)
11      P := PROFILE(T, p, q, P, c, anc)
12      for k := 1 to q - 1
13        sib := shift(sib, *)
14        P := P ∪ (anc ∘ sib)
15  return P

```



The  $pq$ -gram profile is

☞ **small**  $\rightarrow$  size  $O(n)$

---

**Theorem 1** For tree  $\mathbf{T}$  with  $l$  leaves and  $i$  non-leaves:

$$|\mathbf{P}^{p,q}(\mathbf{T})| = 2l + qi - 1.$$

---

# *pq*-Grams — The *pq*-Gram Profile

The *pq*-gram profile is

☞ **small** → size  $O(n)$

☞ **easy to store**

⇒ represent the *pq*-grams by fingerprint **hash value**

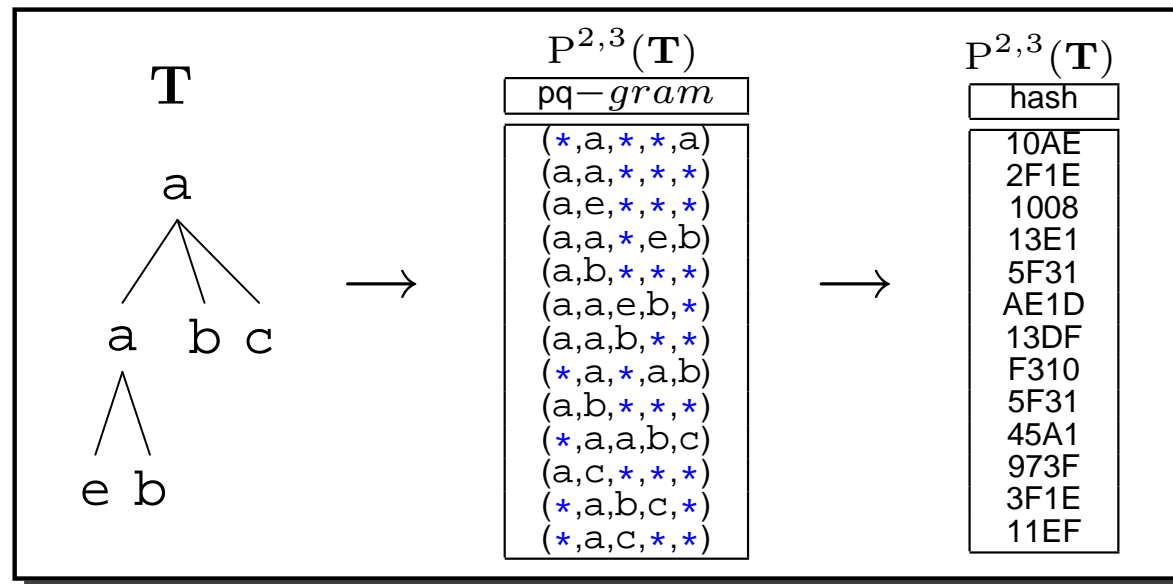
⇒ store profile in **single-attribute relation**

---

**Theorem 1** For tree  $\mathbf{T}$  with  $l$  leaves and  $i$  non-leaves:

$$|P^{p,q}(\mathbf{T})| = 2l + qi - 1.$$


---





# pq-Grams — The pq-Gram Profile

The  $pq$ -gram profile is

☞ **small**  $\rightarrow$  size  $O(n)$

☞ **easy to store**

$\Rightarrow$  represent the  $pq$ -grams by fingerprint **hash value**

$\Rightarrow$  store profile in **single-attribute relation**

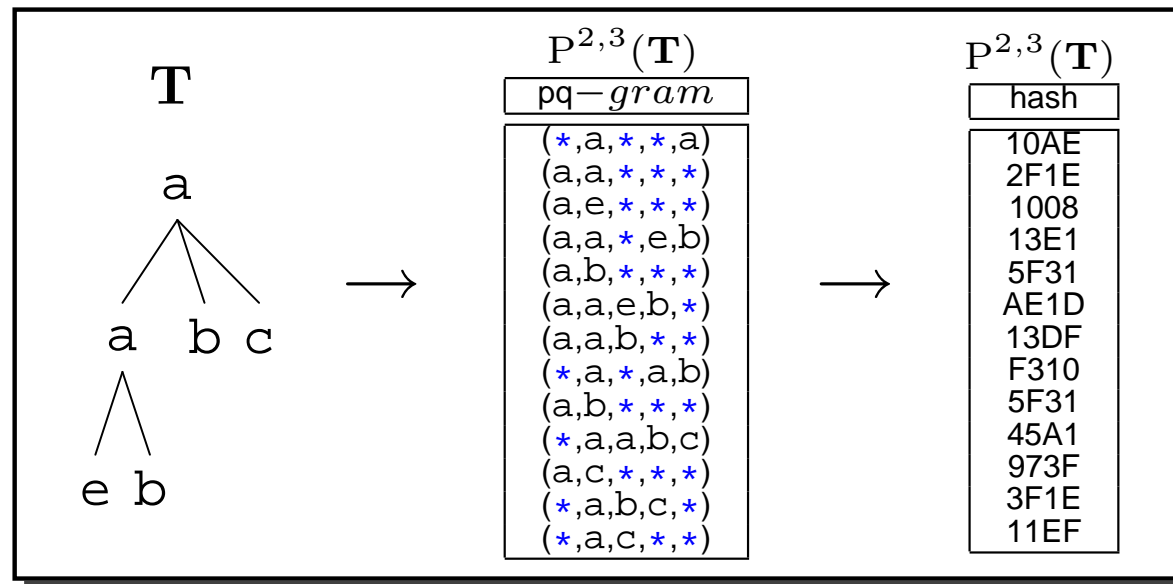
☞ **allows effective distance computation** between trees

---

**Theorem 1** For tree  $\mathbf{T}$  with  $l$  leaves and  $i$  non-leaves:

$$|P^{p,q}(\mathbf{T})| = 2l + qi - 1.$$


---



➡ **Definition 1** For two trees  $\mathbf{T}_1$  and  $\mathbf{T}_2$  the *pq*-gram distance is:

$$\Delta^{p,q}(\mathbf{T}_1, \mathbf{T}_2) = 1 - 2 \frac{|P^{p,q}(\mathbf{T}_1) \cap P^{p,q}(\mathbf{T}_2)|}{|P^{p,q}(\mathbf{T}_1) \cup P^{p,q}(\mathbf{T}_2)|}$$

☞ **Definition 1** For two trees  $\mathbf{T}_1$  and  $\mathbf{T}_2$  the *pq*-gram distance is:

$$\Delta^{p,q}(\mathbf{T}_1, \mathbf{T}_2) = 1 - 2 \frac{|\mathbf{P}^{p,q}(\mathbf{T}_1) \cap \mathbf{P}^{p,q}(\mathbf{T}_2)|}{|\mathbf{P}^{p,q}(\mathbf{T}_1) \cup \mathbf{P}^{p,q}(\mathbf{T}_2)|}$$

☞ can be computed in  $O(n \log n)$  time and  $O(n)$  space (bag intersection of relations)

☞ **Definition 1** For two trees  $\mathbf{T}_1$  and  $\mathbf{T}_2$  the *pq*-gram distance is:

$$\Delta^{p,q}(\mathbf{T}_1, \mathbf{T}_2) = 1 - 2 \frac{|\mathbf{P}^{p,q}(\mathbf{T}_1) \cap \mathbf{P}^{p,q}(\mathbf{T}_2)|}{|\mathbf{P}^{p,q}(\mathbf{T}_1) \cup \mathbf{P}^{p,q}(\mathbf{T}_2)|}$$

☞ can be computed in  $O(n \log n)$  time and  $O(n)$  space (bag intersection of relations)

☞ other terms are constants for **normalization**:

⇒  $\Delta^{p,q}(\mathbf{T}_1, \mathbf{T}_2) = 1$  if trees have no *pq*-grams in common

⇒  $\Delta^{p,q}(\mathbf{T}_1, \mathbf{T}_2) = 0$  if trees have the same *pq*-gram profile

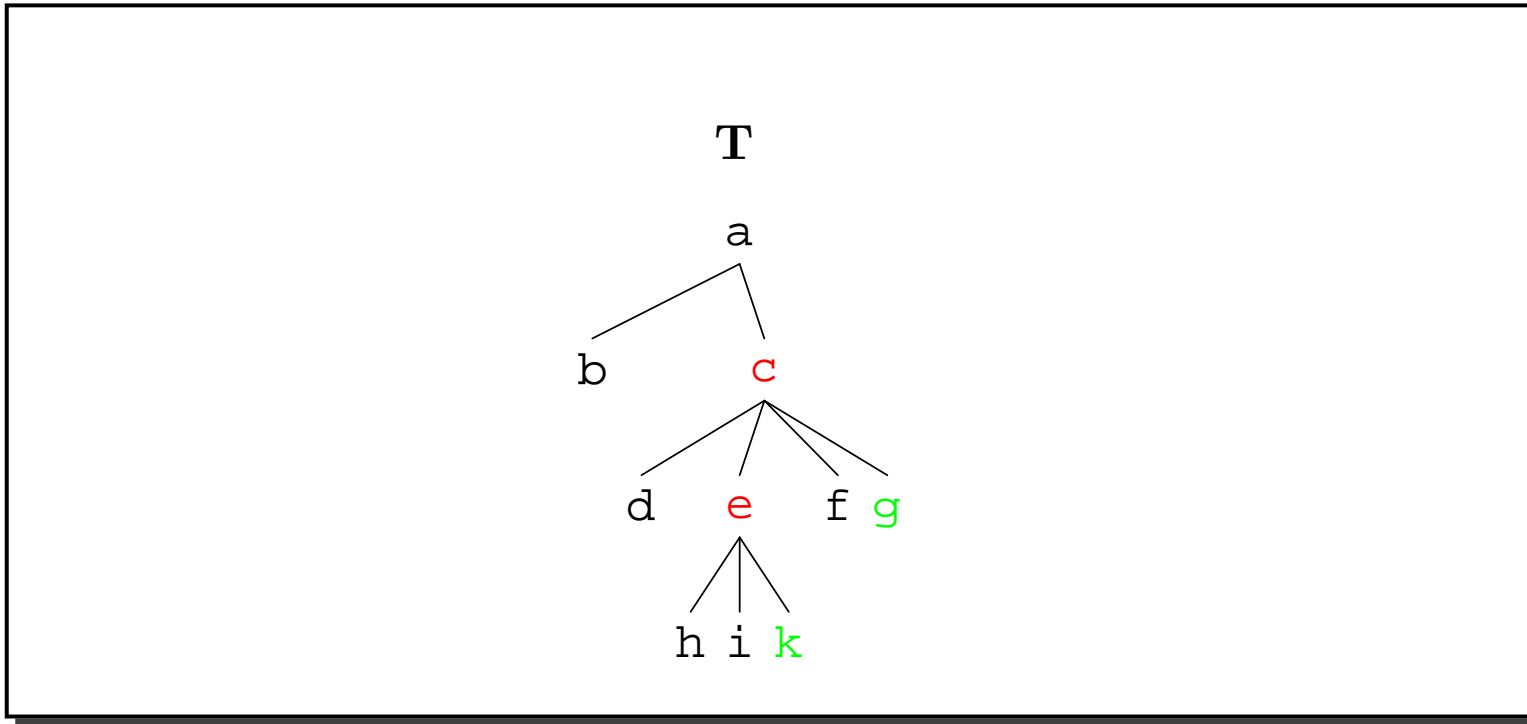
# Properties — Sensitivity to Structure Change

---

☞ **Intuition:** Nodes with structural information → more significant

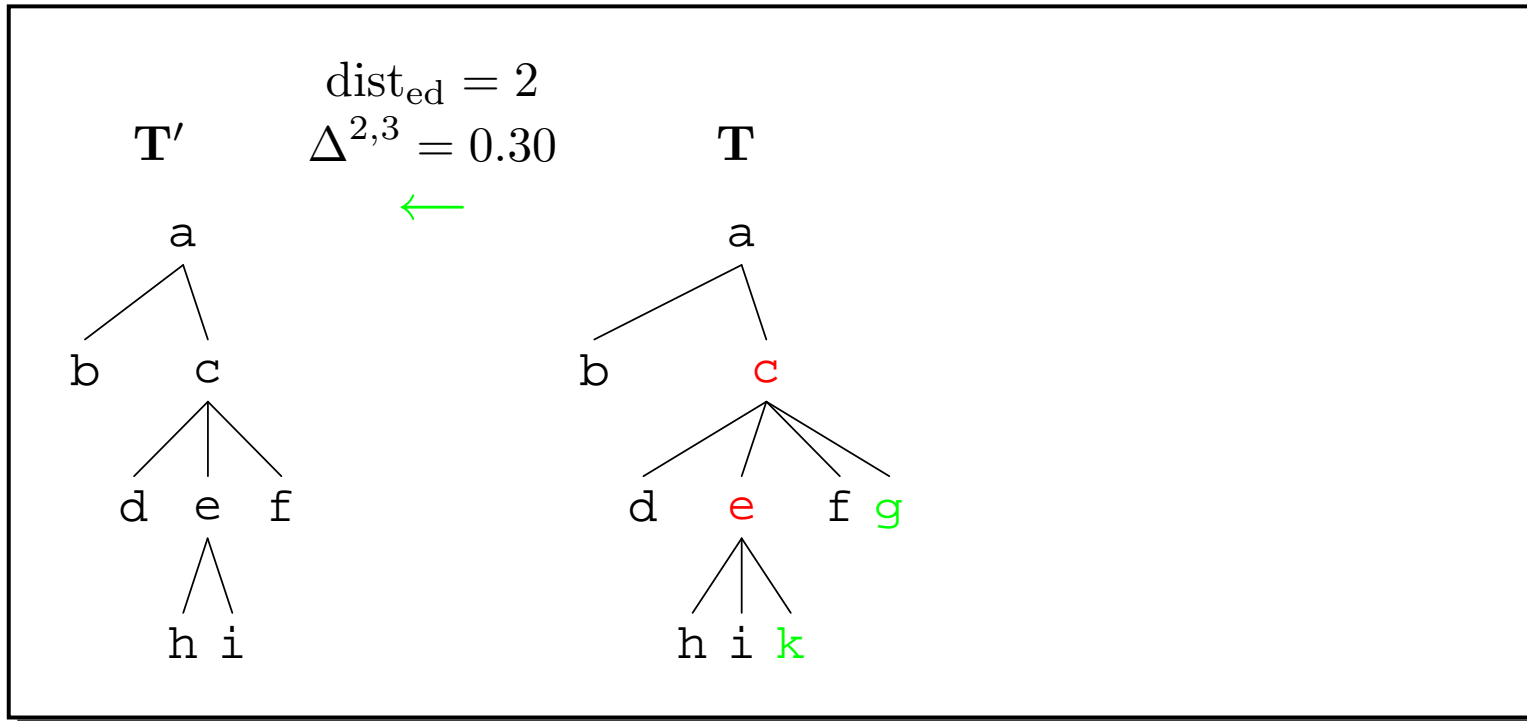
# Properties — Sensitivity to Structure Change

👉 **Intuition:** Nodes with structural information → more significant



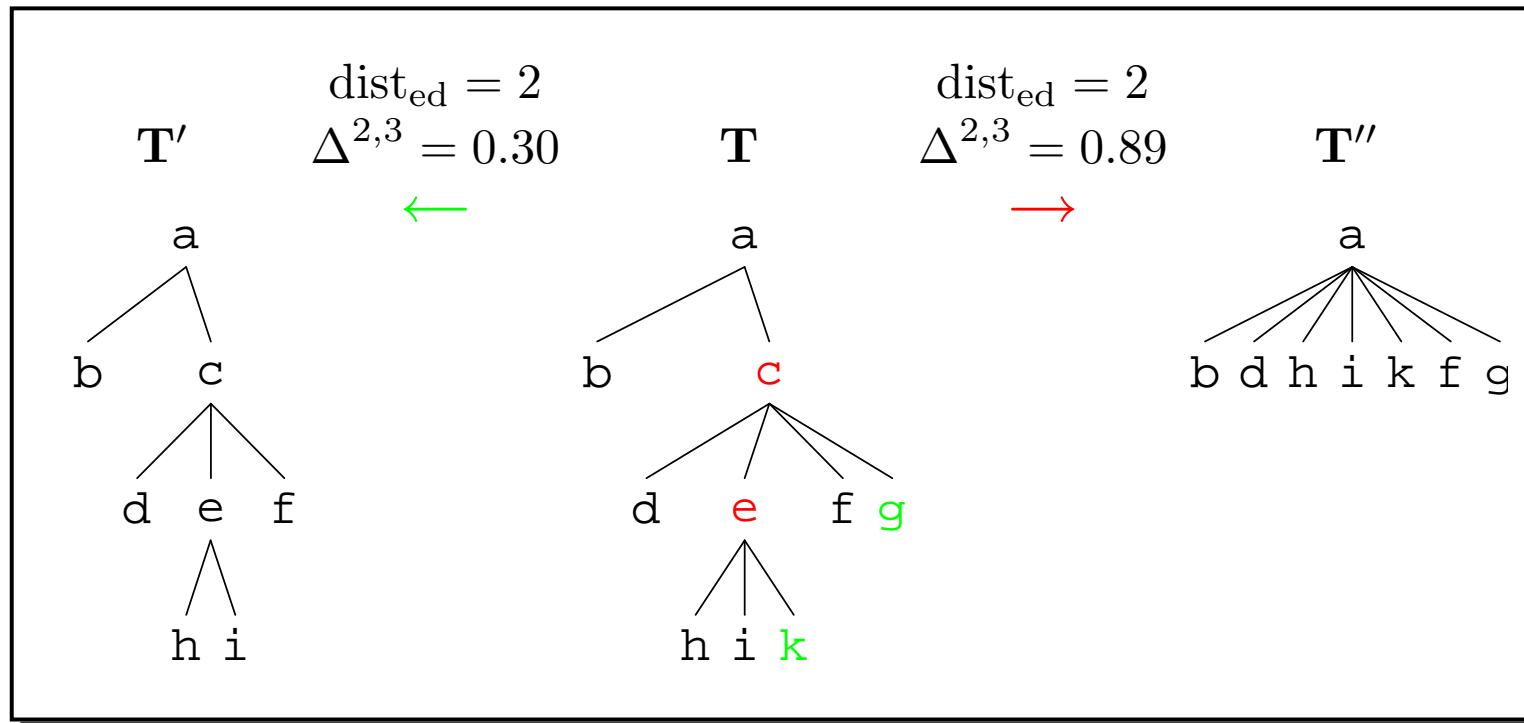
# Properties — Sensitivity to Structure Change

👉 **Intuition:** Nodes with structural information → more significant



# Properties — Sensitivity to Structure Change

👉 **Intuition:** Nodes with structural information → more significant

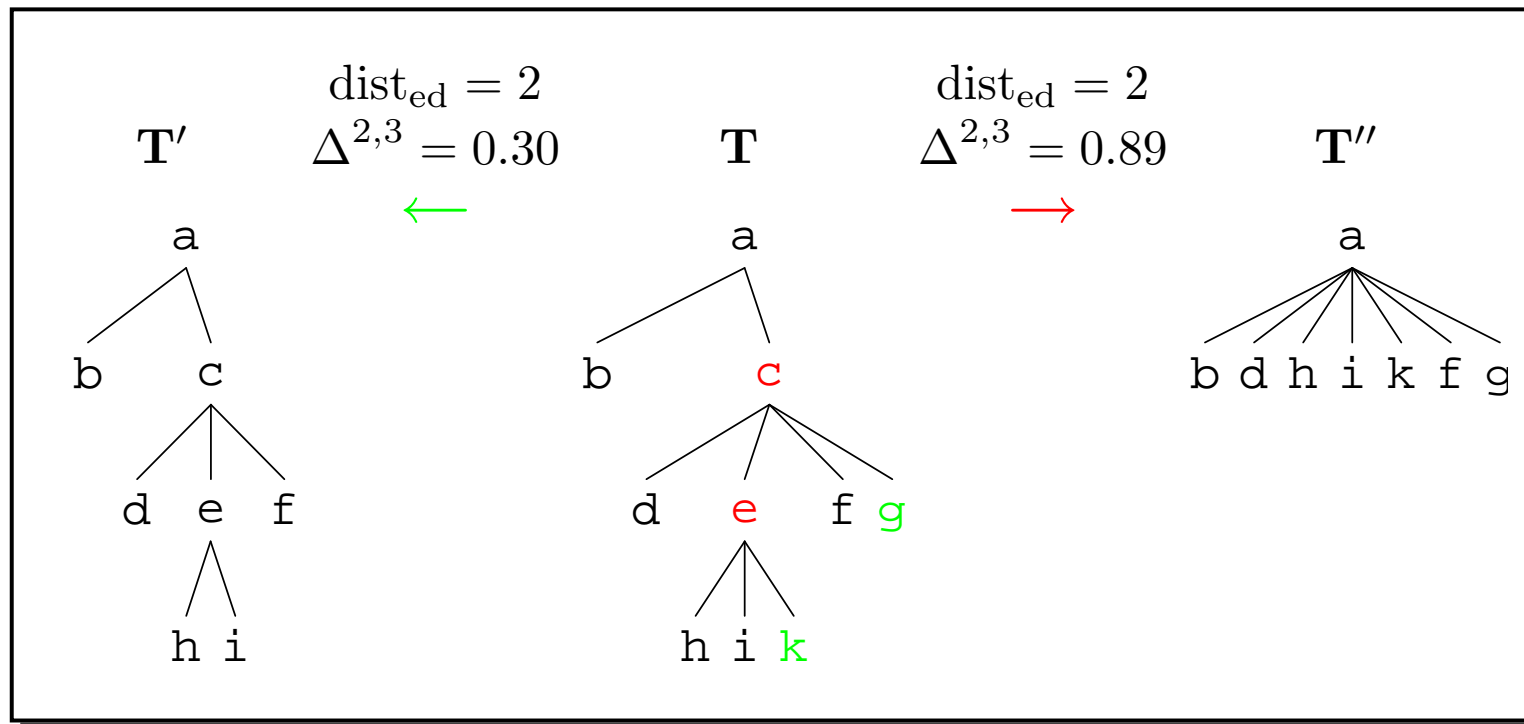




# Properties — Sensitivity to Structure Change

👉 **Intuition:** Nodes with structural information → more significant

👉 **Address application:** Mismatch of houses (with subnumbers and apartment numbers) is more significant than mismatch of apartments.



# Properties — Sensitive to Structure Change

---

☞ node changes  $\Rightarrow$   $pq$ -grams change

☞  $pq$ -grams change  $\Rightarrow$  distance increases

# Properties — Sensitive to Structure Change

---

- ☞ node changes  $\Rightarrow pq$ -grams change
- ☞  $pq$ -grams change  $\Rightarrow$  distance increases
- ☞  $\text{cnt}_{pq}(\mathbf{T}, \mathbf{v}) \approx q + f^p$ 
  - $\Rightarrow$  **leaf** change: cost depends **only on  $q$**
  - $\Rightarrow$  **non-leaf** change:  **$p$  prevalent**

# Properties — Sensitive to Structure Change

---

- ☞ node changes  $\Rightarrow pq$ -grams change
- ☞  $pq$ -grams change  $\Rightarrow$  distance increases
- ☞  $\text{cnt}_{pq}(\mathbf{T}, v) \approx q + f^p$ 
  - $\Rightarrow$  **leaf** change: cost depends **only on  $q$**
  - $\Rightarrow$  **non-leaf** change:  **$p$  prevalent**

---

**Theorem 2** For a complete tree  $\mathbf{T}$  (fanout  $f$ , depth  $d$ ) the number of  $pq$ -grams that contain a node  $v$  of level  $l$  is:

$$\text{cnt}_{pq}(\mathbf{T}, v) = q \text{sgn}(l) + \begin{cases} \frac{f^p - 1}{f - 1} (f + q - 1) & \text{if } p \leq d - l \\ \frac{f^{d-l} - 1}{f - 1} (f + q - 1) + f^{d-l} & \text{if } p > d - l. \end{cases}$$

---

# Properties — Sensitive to Structure Change

---

- ☞ node changes  $\Rightarrow pq$ -grams change
- ☞  $pq$ -grams change  $\Rightarrow$  distance increases
- ☞  $\text{cnt}_{pq}(\mathbf{T}, v) \approx q + f^p$ 
  - $\Rightarrow$  **leaf** change: cost depends **only on  $q$**
  - $\Rightarrow$  **non-leaf** change:  **$p$  prevalent**
  - $\Rightarrow$   **$p$**  controls structure sensitivity

---

**Theorem 2** For a complete tree  $\mathbf{T}$  (fanout  $f$ , depth  $d$ ) the number of  $pq$ -grams that contain a node  $v$  of level  $l$  is:

$$\text{cnt}_{pq}(\mathbf{T}, v) = q \text{sgn}(l) + \begin{cases} \frac{f^p - 1}{f - 1} (f + q - 1) & \text{if } p \leq d - l \\ \frac{f^{d-l} - 1}{f - 1} (f + q - 1) + f^{d-l} & \text{if } p > d - l. \end{cases}$$

---

# Properties — Robust to Local Changes

---

👉 **Intuition:** Weight local changes less than distributed changes!

# Properties — Robust to Local Changes

---

- ➡ **Intuition:** Weight local changes less than distributed changes!
- ➡ **Address application:** missing house with apartment numbers → small difference, but many nodes change

# Properties — Robust to Local Changes

---

- ☞ **Intuition:** Weight local changes less than distributed changes!
- ☞ **Address application:** missing house with apartment numbers → small difference, but many nodes change
- ☞ Local changes → less  $pq$ -grams change
  - ⇒ neighbored nodes “share”  $pq$ -grams
  - ⇒ change counts only once.



# Properties — Robust to Local Changes

---

- ☞ **Intuition:** Weight local changes less than distributed changes!
- ☞ **Address application:** missing house with apartment numbers → small difference, but many nodes change
- ☞ Local changes → less  $pq$ -grams change
  - ⇒ neighbored nodes “share”  $pq$ -grams
  - ⇒ change counts only once.

---

**Theorem 3** *Delete or update all nodes of subtree with  $l$  leaves and  $i$  non-leaves  $\Rightarrow$  only  $2l + iq + q - 1$   $pq$ -grams change*

---

# Properties — Robust to Local Changes

- 👉 **Intuition:** Weight local changes less than distributed changes!
- 👉 **Address application:** missing house with apartment numbers → small difference, but many nodes change
- 👉 Local changes → less  $pq$ -grams change
  - ⇒ neighbored nodes “share”  $pq$ -grams
  - ⇒ change counts only once.

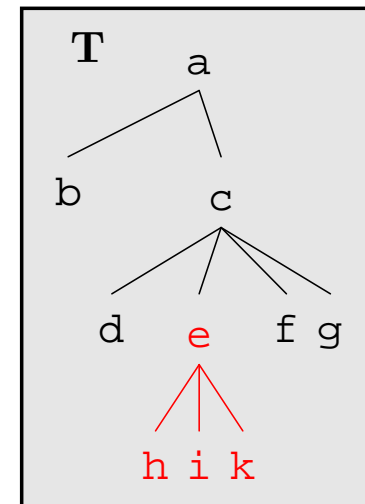
---

**Theorem 3** Delete or update all nodes of subtree with  $l$  leaves and  $i$  non-leaves  $\Rightarrow$  only  $2l + iq + q - 1$   $pq$ -grams change

---

**Example:** Delete subtree rooted in  $e$

- $e \rightarrow$  in 11  $pq$ -grams
- $h, i, k \rightarrow$  in 4  $pq$ -grams each
- actually changing: 11  $pq$ -grams (vs.  $3 \times 4 + 11 = 23$ )



☞ Cost for **leaf change** → depends only on  $q$

# Experiments — Sensitivity to Structure Changes

---

☞ Cost for **leaf change** → depends only on  $q$

☞ **Experiment:**

⇒ delete leaf nodes

⇒ measure edit distance

# Experiments — Sensitivity to Structure Changes

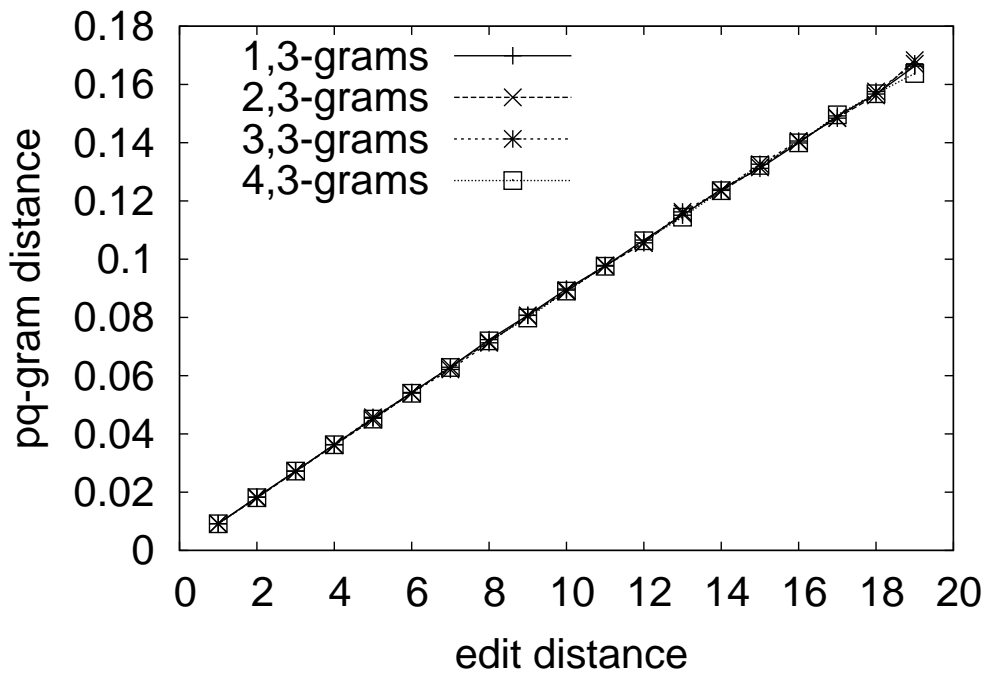
☞ Cost for **leaf change** → depends only on  $q$

☞ **Experiment:**

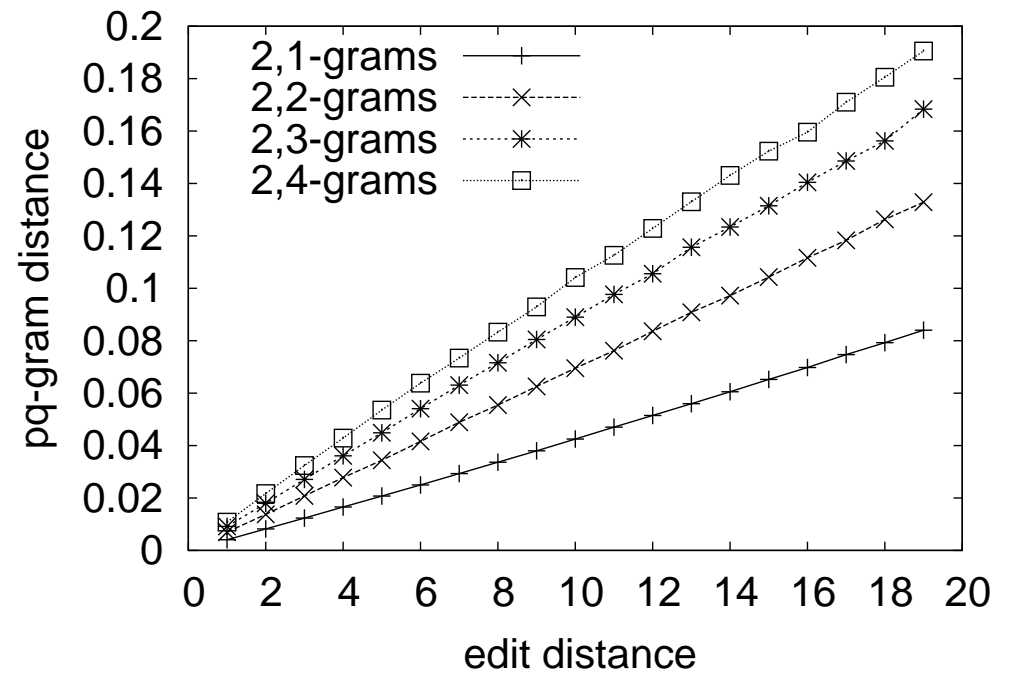
⇒ delete leaf nodes

⇒ measure edit distance

vary  $p$



vary  $q$



(Artificial tree with 144 nodes, 102 leaves, fanout 2–6 and depth 6. Average over 100 runs.)

# Experiments — Sensitivity to Structure Changes

---

☞ Cost for **non-leaf change** → controlled by  $p$

# Experiments — Sensitivity to Structure Changes

---

☞ Cost for **non-leaf change** → controlled by  $p$

☞ **Experiment:**

⇒ delete non-leaf nodes

⇒ measure edit distance

# Experiments — Sensitivity to Structure Changes

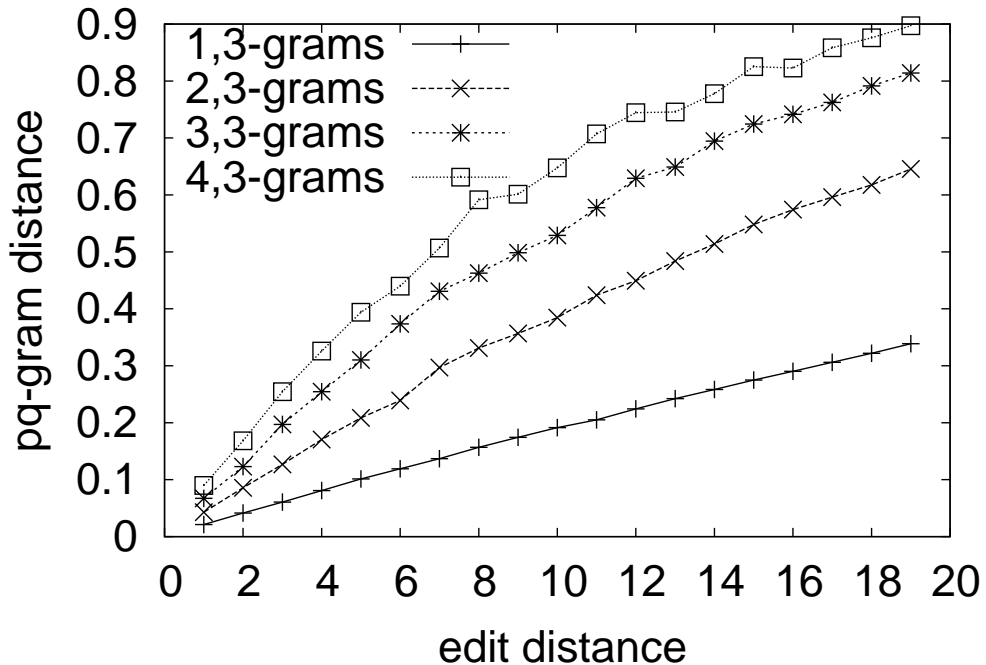
☞ Cost for **non-leaf change** → controlled by  $p$

☞ **Experiment:**

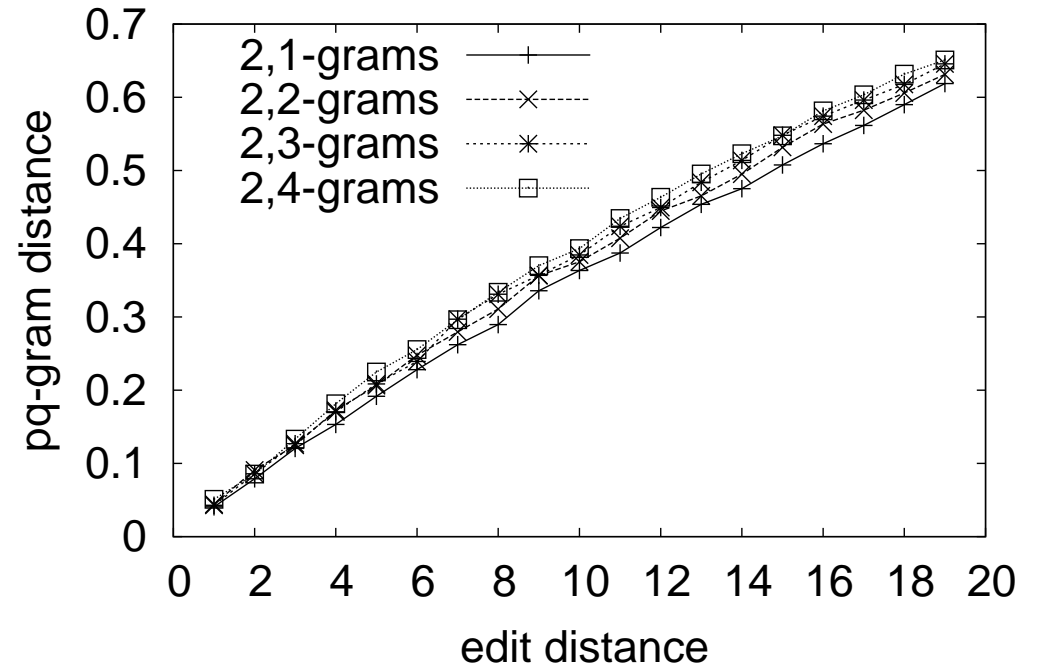
⇒ delete non-leaf nodes

⇒ measure edit distance

vary  $p$



vary  $q$



(Artificial tree with 144 nodes, 102 leaves, fanout 2–6 and depth 6. Average over 100 runs.)



# Experiments — Robustness to Local Changes

---

👉 **Subtree deletions** → **cheaper** than distributed deletions

# Experiments — Robustness to Local Changes

---

☞ **Subtree deletions** → **cheaper** than distributed deletions

☞ **Experiment:**

⇒ delete subtree

⇒ randomly delete same number of distributed nodes

⇒ compare edit distance

# Experiments — Robustness to Local Changes

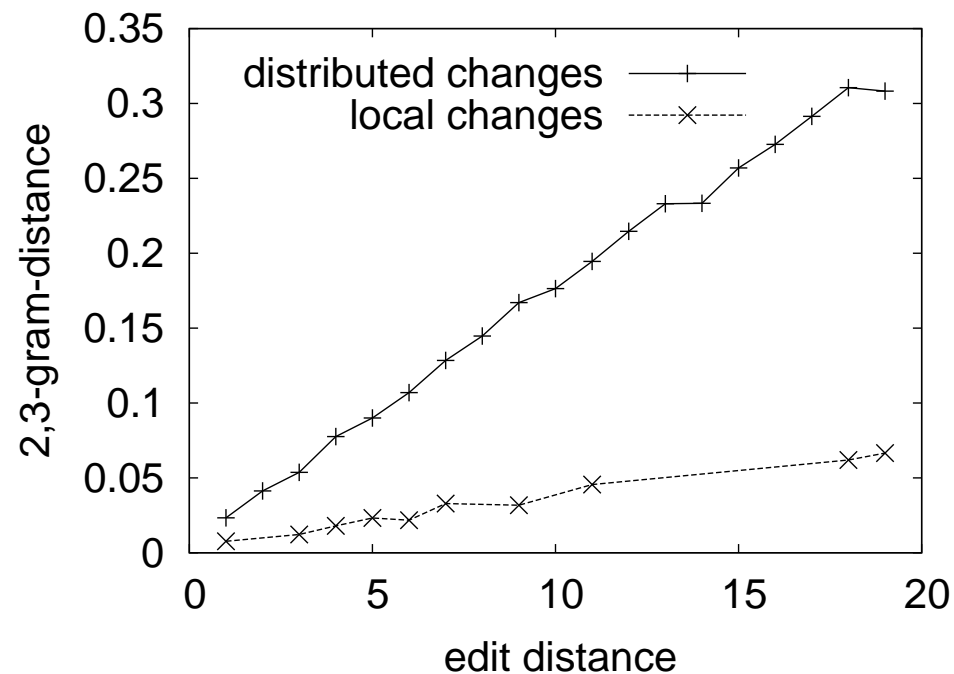
☞ **Subtree deletions** → cheaper than distributed deletions

☞ **Experiment:**

⇒ delete subtree

⇒ randomly delete same number of distributed nodes

⇒ compare edit distance



(Artificial tree with 144 nodes, 102 leaves, fanout 2–6 and depth 6. Average over 100 runs.)

# Experiments — Scalability to Large Trees

---

☞ *pq*-gram distance → **scalable** to large trees

☞ compare with edit distance

---

<sup>a</sup>implementation by Zhang and Shasha (<http://www.cs.nyu.edu/cs/faculty/shasha/papers/tree.html>)

# Experiments — Scalability to Large Trees

---

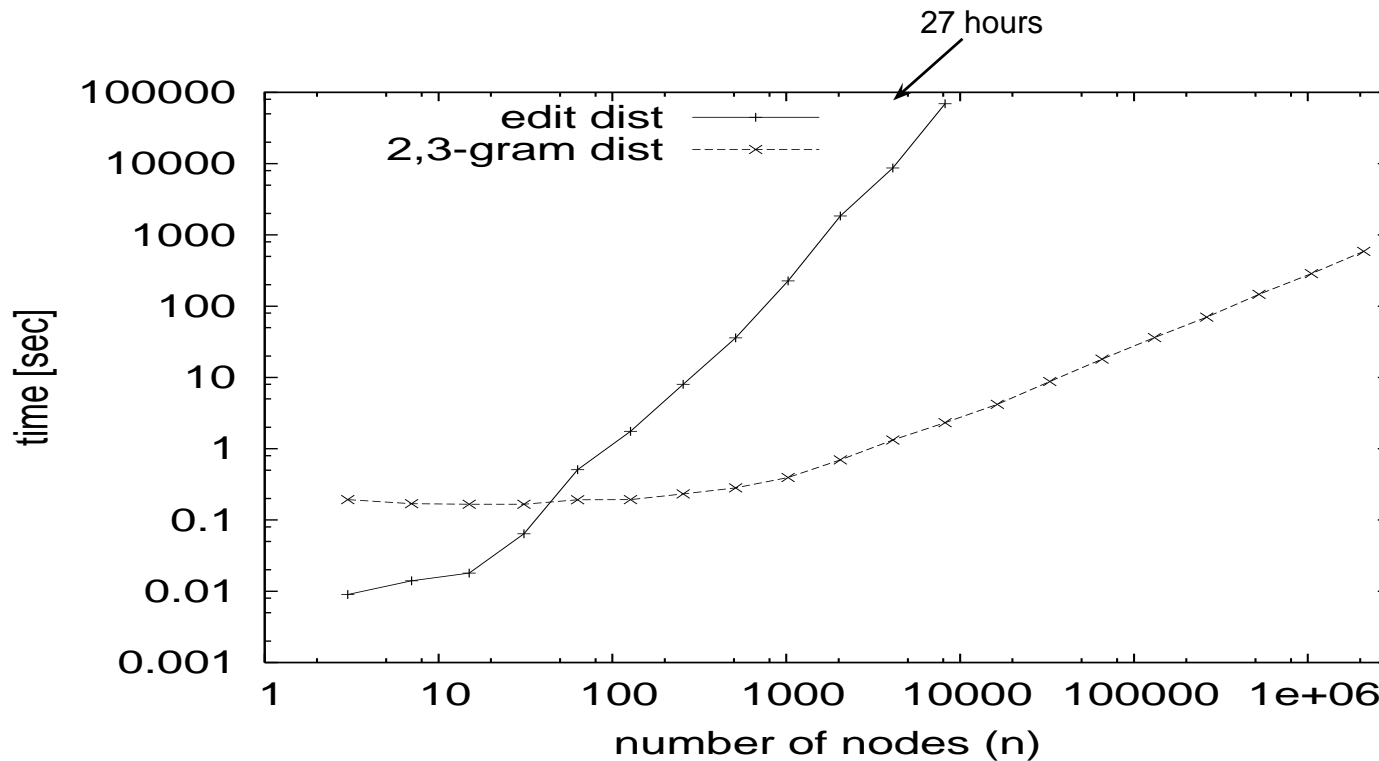
- ☞ *pq*-gram distance → **scalable** to large trees
- ☞ compare with edit distance
- ☞ **Experiment:** For pair of trees
  - ⇒ compute tree edit distance<sup>a</sup> and *pq*-gram distance
  - ⇒ vary tree size: up  $2 \times 10^6$  nodes
  - ⇒ measure wall clock time

---

<sup>a</sup>implementation by Zhang and Shasha (<http://www.cs.nyu.edu/cs/faculty/shasha/papers/tree.html>)

# Experiments — Scalability to Large Trees

- ☞  $pq$ -gram distance → **scalable** to large trees
- ☞ compare with edit distance
- ☞ **Experiment:** For pair of trees
  - ⇒ compute tree edit distance<sup>a</sup> and  $pq$ -gram distance
  - ⇒ vary tree size: up to  $2 \times 10^6$  nodes
  - ⇒ measure wall clock time



<sup>a</sup>implementation by Zhang and Shasha (<http://www.cs.nyu.edu/cs/faculty/shasha/papers/tree.html>)

# Experiments — Influence of $p$ and $q$ on Scalability

---

➡ Scalability independent of  $p$  and  $q$ .

# Experiments — Influence of $p$ and $q$ on Scalability

---

☞ Scalability independent of  $p$  and  $q$ .

☞ **Experiment:** For pair of trees

⇒ compute  $pq$ -gram distance for varying  $p$  and  $q$

⇒ vary tree size: up  $10^6$  nodes

⇒ measure wall clock time



# Experiments — Influence of $p$ and $q$ on Scalability

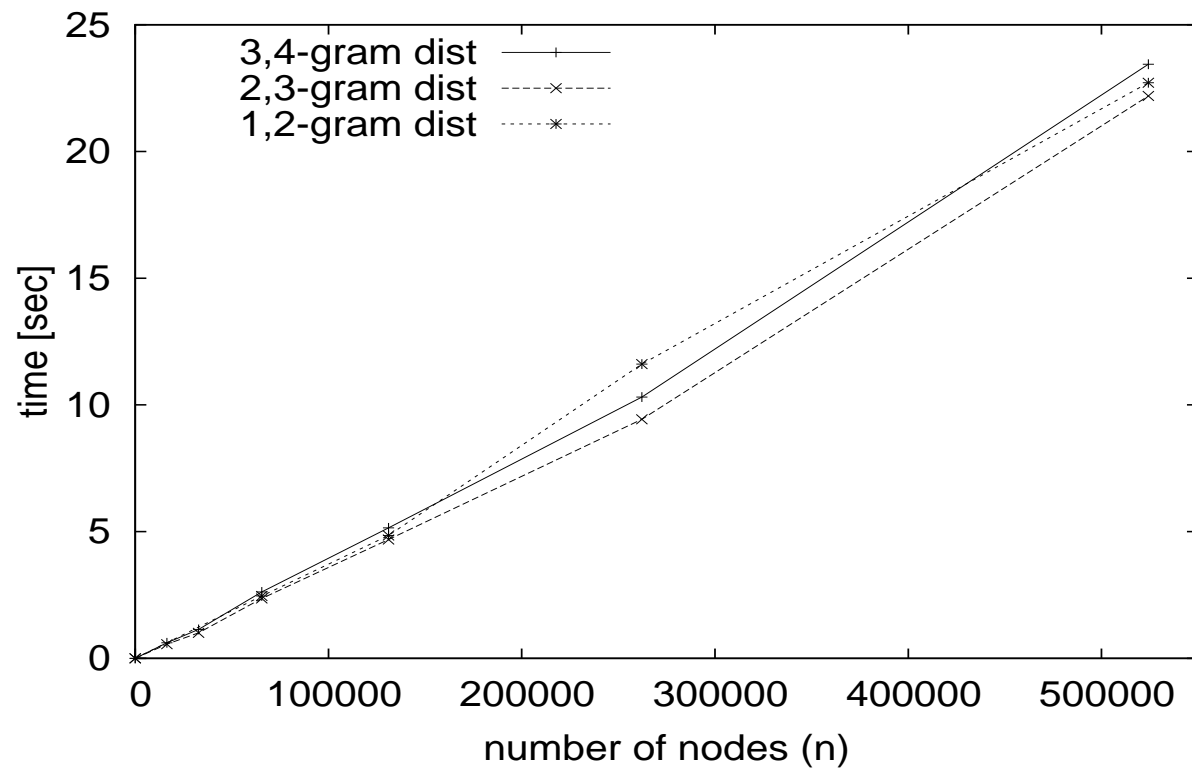
☞ Scalability independent of  $p$  and  $q$ .

☞ **Experiment:** For pair of trees

⇒ compute  $pq$ -gram distance for varying  $p$  and  $q$

⇒ vary tree size: up  $10^6$  nodes

⇒ measure wall clock time



# Experiments — Effectiveness for Real World Dataset

---

☞  $pq$ -gram distance → **effective approximation** of tree edit distance

☞ test on real world data (**address databases**)

☞  $pq$ -gram distance → **effective approximation** of tree edit distance

☞ test on real world data (**address databases**)

☞ **Experiment:** For two sets of address trees

⇒ find matches (closest tree of other set)

⇒ use different distance functions

⇒ count correct matches

# Experiments — Effectiveness for Real World Dataset

☞  $pq$ -gram distance → **effective approximation** of tree edit distance

☞ test on real world data (**address databases**)

☞ **Experiment:** For two sets of address trees

⇒ find matches (closest tree of other set)

⇒ use different distance functions

⇒ count correct matches

	accuracy	correct	false pos.	runtime
edit dist	82.7%	248	9	187,538s
1,2-grams	78.3%	235	5	181s
2,3-grams	77.3%	232	4	204s
3,2-grams	79.3%	238	2	180s
tree-embedding	69.0%	207	8	313s
bottom-up	50.0%	150	12	237s

# Experiments — Tree-Embedding vs. $pq$ -Grams

---

tree edit distance embedding

☞ variable shapes

$pq$ -grams

☞ fixed shape

# Experiments — Tree-Embedding vs. *pq*-Grams

tree edit distance embedding

- ☞ variable shapes
- ☞ elements can be
  - ⇒ single node (no structure)
  - ⇒ chains (only vertical structure)
  - ⇒ contiguous leaves (only horizontal structure)
  - ⇒ subtrees with vertical and horizontal structure

*pq*-grams

- ☞ fixed shape
- ☞ horizontal **and** vertical structure

Typical address tree (nearly complete tree):

Phase	0	1	2	3	4	5	6	tot.	tree-embed	<i>pq</i> -gram
single nodes	29	8	4	2	1	-	-	44	65%	0%
chains	-	1	1	-	-	-	-	2	3%	0%
cont. leaves	-	7	2	-	-	-	-	9	13%	0%
subtrees	-	-	3	4	3	2	1	13	19%	100%

# Experiments — Tree-Embedding vs. *pq*-Grams

tree edit distance embedding	<i>pq</i> -grams
<ul style="list-style-type: none"> <li>☞ variable shapes</li>   <li>☞ elements can be                             <ul style="list-style-type: none"> <li>☞ single node (no structure)</li> <li>☞ chains (only vertical structure)</li> <li>☞ contiguous leaves (only horizontal structure)</li> <li>☞ subtrees with vertical and horizontal structure</li> </ul> </li>   <li>☞ guarantees with respect to tree edit distance</li> </ul>	<ul style="list-style-type: none"> <li>☞ fixed shape</li>   <li>☞ horizontal <b>and</b> vertical structure</li>   <li>☞ emphasizes structure</li> </ul>

Typical address tree (nearly complete tree):

Phase	0	1	2	3	4	5	6	tot.	tree-embed	<i>pq</i> -gram
single nodes	29	8	4	2	1	-	-	44	65%	0%
chains	-	1	1	-	-	-	-	2	3%	0%
cont. leaves	-	7	2	-	-	-	-	9	13%	0%
subtrees	-	-	3	4	3	2	1	13	19%	100%

# Conclusion and Future Work

---

☞ *pq*-gram distance



# Conclusion and Future Work

---

☞ *pq*-gram distance

⇒ scalable to large trees

# Conclusion and Future Work

---

- ☞ *pq*-gram distance
  - ⇒ scalable to large trees
  - ⇒ emphasizes structure

# Conclusion and Future Work

---

- ☞ *pq*-gram distance
  - ⇒ scalable to large trees
  - ⇒ emphasizes structure
  - ⇒ robust to local change

# Conclusion and Future Work

---

☞ *pq*-gram distance

- ⇒ scalable to large trees
- ⇒ emphasizes structure
- ⇒ robust to local change
- ⇒ effective approximation of tree edit distance

# Conclusion and Future Work

---

## ☞ *pq*-gram distance

- ⇒ scalable to large trees
- ⇒ emphasizes structure
- ⇒ robust to local change
- ⇒ effective approximation of tree edit distance

## ☞ Ongoing and future work:

# Conclusion and Future Work

---

## ☞ *pq*-gram distance

- ⇒ scalable to large trees
- ⇒ emphasizes structure
- ⇒ robust to local change
- ⇒ effective approximation of tree edit distance

## ☞ Ongoing and future work:

- ⇒ strict bounds for *pq*-gram approximations

# Conclusion and Future Work

---

## ☞ *pq*-gram distance

- ⇒ scalable to large trees
- ⇒ emphasizes structure
- ⇒ robust to local change
- ⇒ effective approximation of tree edit distance

## ☞ Ongoing and future work:

- ⇒ strict bounds for *pq*-gram approximations
- ⇒ clustering of XML data

# Conclusion and Future Work

---

## ☞ *pq*-gram distance

- ⇒ scalable to large trees
- ⇒ emphasizes structure
- ⇒ robust to local change
- ⇒ effective approximation of tree edit distance

## ☞ Ongoing and future work:

- ⇒ strict bounds for *pq*-gram approximations
- ⇒ clustering of XML data
- ⇒ incremental updates of *pq*-gram profiles



---

## References

- [[Chawathe and Garcia-Molina, 1997](#)] Chawathe, S. S. and Garcia-Molina, H. (1997). Meaningful change detection in structured data. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 26–37, Tucson, Arizona, United States. ACM Press.
- [[Chawathe et al., 1996](#)] Chawathe, S. S., Rajaraman, A., Garcia-Molina, H., and Widom, J. (1996). Change detection in hierarchically structured information. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 493–504, Montreal, Quebec, Canada. ACM Press.
- [[Garofalakis and Kumar, 2003](#)] Garofalakis, M. and Kumar, A. (2003). Correlating XML data streams using tree-edit distance embeddings. In *Proceedings of the Twenty-Second ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems (PODS 2003)*, pages 143–154, San Diego, California. ACM Press.
- [[Garofalakis and Kumar, 2005](#)] Garofalakis, M. and Kumar, A. (2005). XML stream processing using tree-edit distance embeddings. *ACM Transactions on Database Systems*, 30(1):279–332.
- [[Jiang et al., 1995](#)] Jiang, T., Wang, L., and Zhang, K. (1995). Alignment of trees—an alternative to tree edit. *Theoretical Computer Science*, 143(1):137–148.
- [[Klein, 1998](#)] Klein, P. N. (1998). Computing the edit-distance between unrooted ordered trees. In *Proceedings of the 6th European Symposium on Algorithms*, volume 1461 of *Lecture Notes in Computer Science*, pages 91–102, Venice, Italy. Springer.

- 
- [[Lee et al., 2004](#)] Lee, K.-H., Choy, Y.-C., and Cho, S.-B. (2004). An efficient algorithm to compute differences between structured documents. *IEEE Transactions on Knowledge and Data Engineering*, 16(8):965–979.
- [[Navarro, 2001](#)] Navarro, G. (2001). A guided tour to approximate string matching. *ACM Computing Surveys*, 33(1):31–88.
- [[Selkow, 1977](#)] Selkow, S. M. (1977). The tree-to-tree editing problem. *Information Processing Letters*, 6(6):184–186.
- [[Tanaka and Tanaka, 1988](#)] Tanaka, E. and Tanaka, K. (1988). The tree-to-tree editing problem. *Int. Journal of Pattern Recognition and Artificial Intelligence (IJPRAI)*, 2(2):221–240.
- [[Ukkonen, 1992](#)] Ukkonen, E. (1992). Approximate string-matching with  $q$ -grams and maximal matches. *Theoretical Computer Science*, 92(1):191–211.
- [[Valiente, 2001](#)] Valiente, G. (2001). An efficient bottom-up distance between trees. In *Proceedings of the 8th Symposium on String Processing and Information Retrieval*, pages 212–219, Laguna de San Rafael, Chile. IEEE Computer Science Press.
- [[Yang, 1991](#)] Yang, W. (1991). Identifying syntactic differences between two programs. *Software—Practice & Experience*, 21(7):739–755.
- [[Zhang and Shasha, 1989](#)] Zhang, K. and Shasha, D. (1989). Simple fast algorithms for the editing distance between trees and related problems. *SIAM Journal on Computing*, 18(6):1245–1262.