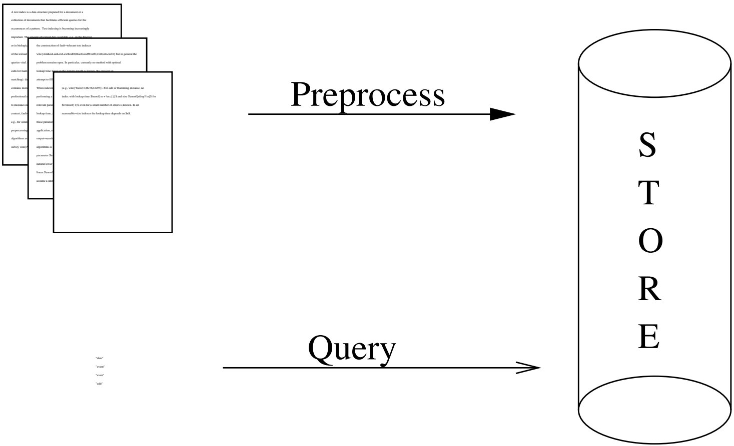
Text Indexing with Errors

Moritz G. Maaß and Johannes Nowak

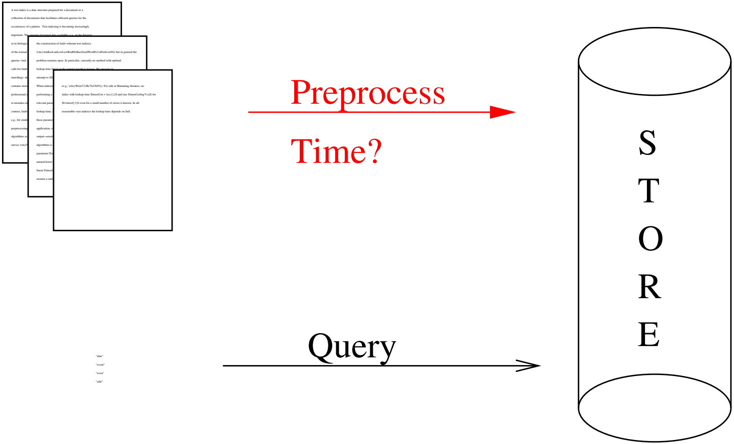
{maass,nowakj}@in.tum.de
Institut für Informatik
Technische Universität München

June 20, 2005

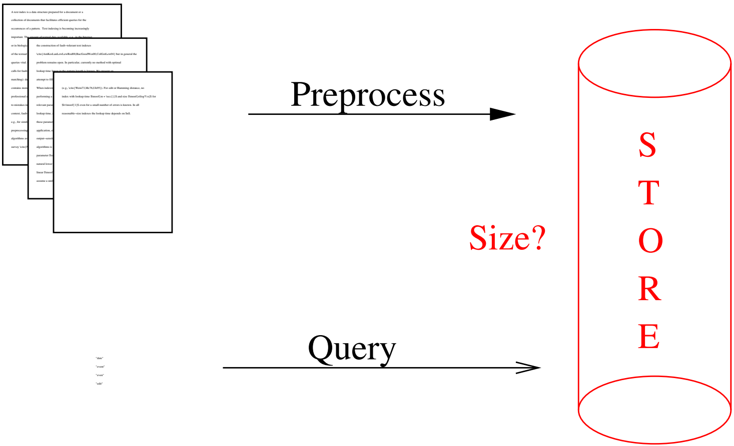
Overview



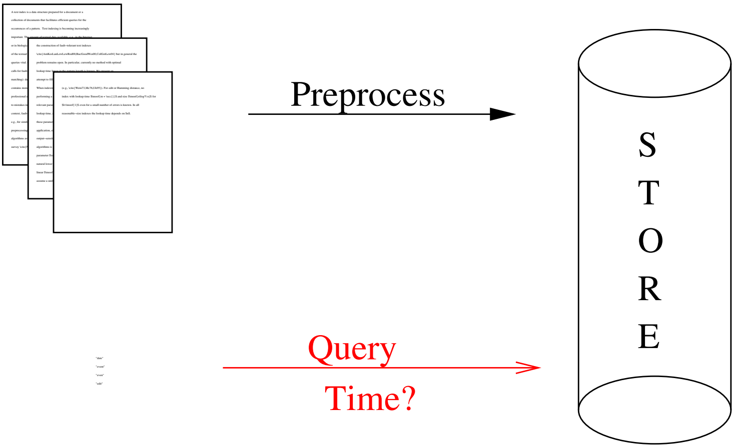
Overview



Overview



Overview



Indexing Problems

One pattern is queried against

- a single document – Text Indexing
- a collection of words
- a collection documents

Indexing Problems

One pattern is queried against

- a single document – Text Indexing
- a collection of words – Dictionary Indexing
- a collection documents

Indexing Problems

One pattern is queried against

- a single document – Text Indexing
- a collection of words – Dictionary Indexing
- a collection documents

Indexing Problems

One pattern is queried against

- a single document – Text Indexing
- a collection of words – Dictionary Indexing
- a collection documents – Document Collection Indexing

Indexing Problems

One pattern is queried against

- a single document – Text Indexing
- a collection of words – Dictionary Indexing
- a collection documents – Document Collection Indexing

Indexing Problems

One pattern is queried against

- a single document – Text Indexing
- a collection of words – Dictionary Indexing
- a collection documents – Document Collection Indexing

Indexing Problems

One pattern is queried against

- a single document – Text Indexing
- a collection of words – Dictionary Indexing
- a collection documents – Document Collection Indexing

Previous Work

Indexing a text of size[1] n and querying a pattern of size m resulting in occ occurrences.

Errors	Model	Query Time	Index Size	Prep. Time	Literature
$k = 0$	exact	$O(m + \text{occ})$	$O(n)$	$O(n)$	Weiner 1973
$k = 1$	edit	$O(m \log n \log\log n + \text{occ})$	$O(n \log^2 n)$	$O(n \log^2 n)$	Amir et. al 2000
$k = 1$	edit	$O(m \log\log n + \text{occ})$	$O(n \log n)$	$O(n \log n)$	Buchsbaum et. al 2000
$k = 1$	Ham.	$O(m + \text{occ})$	$O(n \log n)$ (avg)	$O(n \log n)$ (avg)	N 2004
$k = O(1)$	edit	$O(m + \log^k n + \text{occ})$	$O(n \log^k n)$	$O(n \log^k n)$	Cole et. al 2004
$k = O(1)$	edit	$O(m \log^2 n + m^2 + \text{occ})$ (avg)	$O(n \log n)$ (avg)	$O(n \log^2 n)$ (avg)	Chávez et al. 2002
$k = O(1)$	edit	$O(m \min\{n, m^{k+1}\} + \text{occ})$	$O(\min\{n, m^{k+1}\} + n)$	$O(\min\{n, m^{k+1}\} + n)$	Cobbs 1995 (Ukkonen 1993)
$k = O(1)$	edit	$O(m + \text{occ})$	$O(n \log^k n)$, (avg,whp)	$O(n \log^{k+1} n)$, (avg,whp)	MN 2005
$k = O(1)$	edit	$O(m + \text{occ})$, (avg,whp)	$O(n \log^k n)$	$O(n \log^{k+1} n)$	
$k = O(1)$	Ham.	$O(\log^{k+1} n)$, (avg)	$O(n)$	$O(n)$	M 2004
$k = \alpha \log n$	Ham.	$O(n^\lambda)$, $\lambda < 1$, (avg)	$O(n)$	$O(n)$	
$k = \alpha m$	edit	$O(n^\lambda \log n)$, $\lambda < 1$	$O(n)$	$O(n)$	Navarro et. al 2000
k mismatches in a window of length r		$O(m + \text{occ})$ (avg)	$O(n \log^l n)$ (avg)	$O(n \log^l n)$ (avg)	Gabriele et. al 2003

[1] Uniform Cost Model

Previous Work

Indexing a text of size[1] n and querying a pattern of size m resulting in occ occurrences.

Errors	Model	Query Time	Index Size	Prep. Time	Literature
$k=0$	exact	$O(\mathbf{m} + \text{occ})$	$O(n)$	$O(n)$	Weiner 1973
$k=1$	edit	$O(m \log \mathbf{n} \log\log \mathbf{n} + \text{occ})$	$O(n \log^2 n)$	$O(n \log^2 n)$	Amir et. al 2000
$k=1$	edit	$O(m \log\log \mathbf{n} + \text{occ})$	$O(n \log n)$	$O(n \log n)$	Buchsbaum et. al 2000
$k=1$	Ham.	$O(\mathbf{m} + \text{occ})$	$O(n \log n)$ (avg)	$O(n \log n)$ (avg)	N 2004
$k=O(1)$	edit	$O(m + \log^k \mathbf{n} + \text{occ})$	$O(n \log^k n)$	$O(n \log^k n)$	Cole et. al 2004
$k=O(1)$	edit	$O(m \log^2 \mathbf{n} + m^2 + \text{occ})$ (avg)	$O(n \log n)$ (avg)	$O(n \log^2 n)$ (avg)	Chávez et al. 2002
$k=O(1)$	edit	$O(m \min\{\mathbf{n}, m^{\mathbf{k}+1}\} + \text{occ})$	$O(\min\{\mathbf{n}, m^{k+1}\} + n)$	$O(\min\{n, m^{k+1}\} + n)$	Cobbs 1995 (Ukkonen 1993)
$k=O(1)$	edit	$O(\mathbf{m} + \text{occ})$	$O(n \log^k n)$, (avg,whp)	$O(n \log^{k+1} n)$, (avg,whp)	MN 2005
$k=O(1)$	edit	$O(\mathbf{m} + \text{occ})$, (avg,whp)	$O(n \log^k n)$	$O(n \log^{k+1} n)$	
$k=O(1)$	Ham.	$O(\log^{k+1} \mathbf{n})$, (avg)	$O(n)$	$O(n)$	M 2004
$k=\alpha \log n$	Ham.	$O(\mathbf{n}^\lambda)$, $\lambda < 1$, (avg)	$O(n)$	$O(n)$	
$k=\alpha m$	edit	$O(\mathbf{n}^\lambda \log \mathbf{n})$, $\lambda < 1$	$O(n)$	$O(n)$	Navarro et. al 2000
k mismatches in a window of length r		$O(\mathbf{m} + \text{occ})$ (avg)	$O(n \log^l n)$ (avg)	$O(n \log^l n)$ (avg)	Gabriele et. al 2003

[1] Uniform Cost Model

Worst-Case Optimal Search-Time

We focus on

- indexing a text t of size n,
- querying a pattern p of size m,
- allowing a constant number of k errors (Edit distance),
- achieving worst-case optimal query-time $O(m + \mathrm{occ})$ and size $O(n \log^k n)$ on average and w.h.p.

Worst-Case Optimal Search-Time

We focus on

- indexing a text t of size n,
- querying a pattern p of size m,
- allowing a constant number of k errors (Edit distance),
- achieving worst-case optimal query-time $O(m + \text{occ})$ and size $O(n \log^k n)$ on average and w.h.p.

Worst-Case Optimal Search-Time

We focus on

- indexing a text t of size n,
- querying a pattern p of size m,
- allowing a constant number of k errors (Edit distance),
- achieving worst-case optimal query-time $O(m + \text{occ})$ and size $O(n \log^k n)$ on average and w.h.p.

Worst-Case Optimal Search-Time

We focus on

- indexing a text t of size n,
- querying a pattern p of size m,
- allowing a constant number of k errors (Edit distance),
- achieving worst-case optimal query-time $O(m + \text{occ})$ and size $O(n \log^k n)$ on average and w.h.p.

Worst-Case Optimal Search-Time

We focus on

- indexing a text t of size n,
- querying a pattern p of size m,
- allowing a constant number of k errors (Edit distance),
- achieving worst-case optimal query-time $O(m + \text{occ})$ and size $O(n \log^k n)$ on average and w.h.p.

String Distance

- Hamming Distance
- Edit Distance
- To handle ambiguities:

String Distance

- Hamming Distance
- Edit Distance
- To handle ambiguities:

Definition (k-Minimal Prefix Length)

For two strings $u, v \in \Sigma^*$ with $\mathrm{d}(u, v) = k$ we define

$$\mathrm{minpref}_{k,u}(v) = \min \left\{ l \;\middle|\; \begin{array}{l} \mathrm{d}(\mathrm{pref}_l(u), \mathrm{pref}_{l+|v|-|u|}(v)) = k \quad \text{and} \\ \mathrm{suff}_{l+1}(u) = \mathrm{suff}_{l+|v|-|u|+1}(v) \end{array} \right\}.$$

String Distance

- Hamming Distance
- Edit Distance
- To handle ambiguities:

Definition (k-Minimal Prefix Length)

For two strings $u, v \in \Sigma^*$ with $\mathrm{d}(u,v) = k$ we define

$$\mathrm{minpref}_{k,u}(v) = \min\left\{ l \;\middle|\; \begin{array}{l} \mathrm{d}(\mathrm{pref}_l(u), \mathrm{pref}_{l+|v|-|u|}(v)) = k \quad \text{and} \\ \mathrm{suff}_{l+1}(u) = \mathrm{suff}_{l+|v|-|u|+1}(v) \end{array} \right\}.$$

String Distance

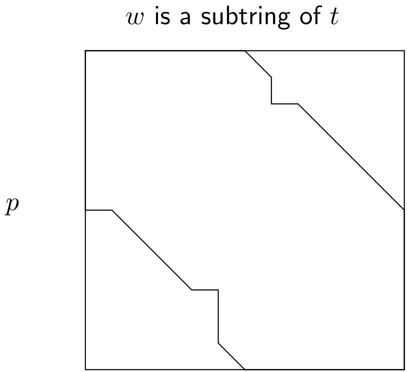
- Hamming Distance
- Edit Distance
- To handle ambiguities:

Definition (k-Minimal Prefix Length)

For two strings $u, v \in \Sigma^*$ with $\mathsf{d}(u,v) = k$ we define

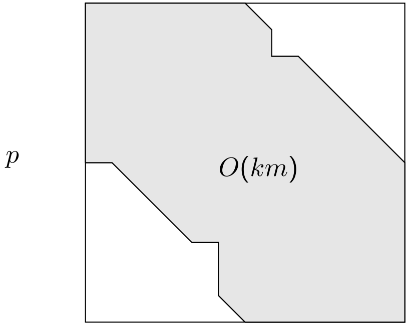
$$\mathsf{minpref}_{k,u}(v) = \min \left\{ l \;\middle|\; \begin{array}{l} \mathsf{d}(\mathsf{pref}_l(u), \mathsf{pref}_{l+|v|-|u|}(v)) = k \quad \text{and} \\ \mathsf{suff}_{l+1}(u) = \mathsf{suff}_{l+|v|-|u|+1}(v) \end{array} \right\} .$$

Simple Methods for Indexing



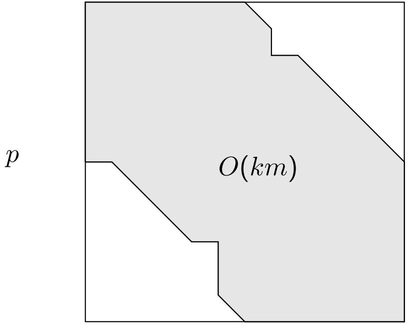
Simple Methods for Indexing

w is a subtring of t

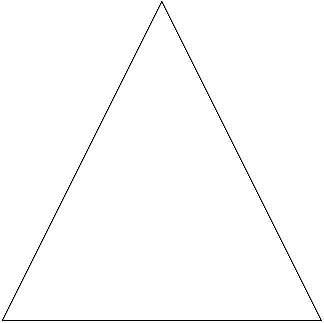


Simple Methods for Indexing

w is a subtring of t



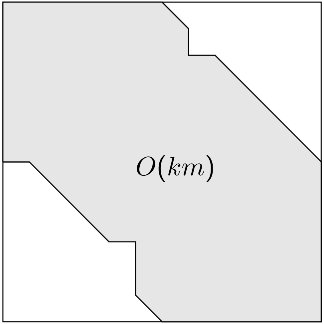
Trie for all strings w within distance k of t



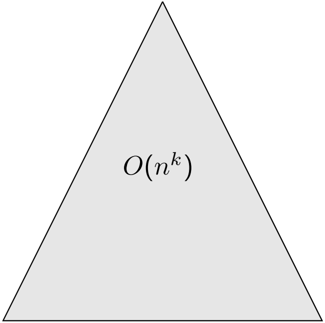
Simple Methods for Indexing

w is a subtring of t

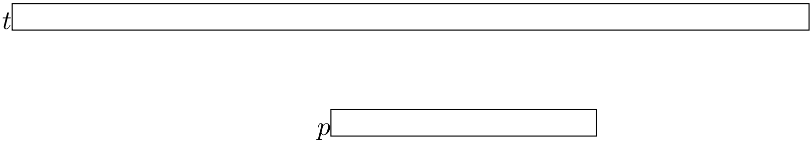
p



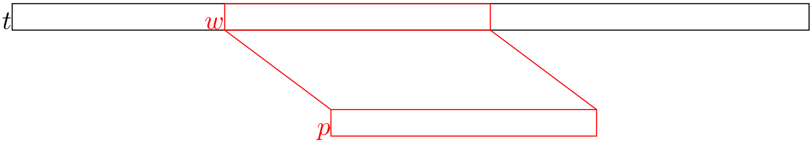
Trie for all strings w within distance k of t



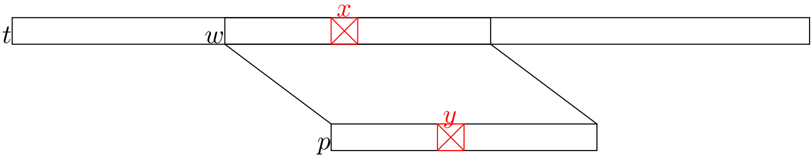
Searching with One Error



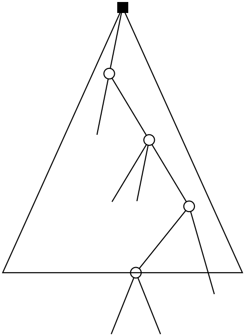
Searching with One Error

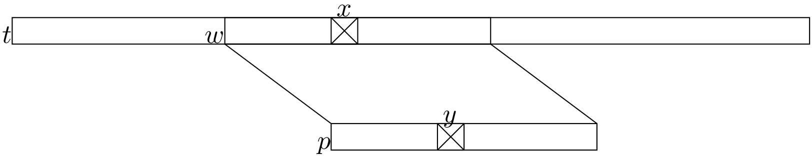


Searching with One Error

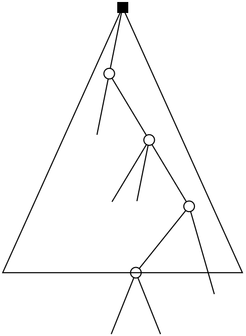


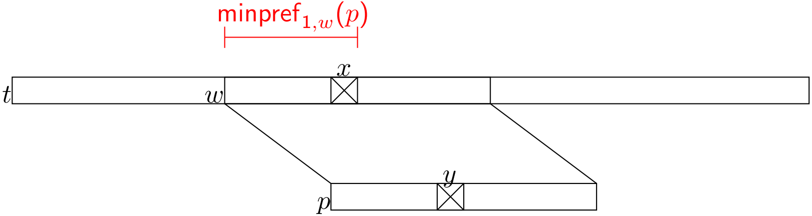
Searching with One Error



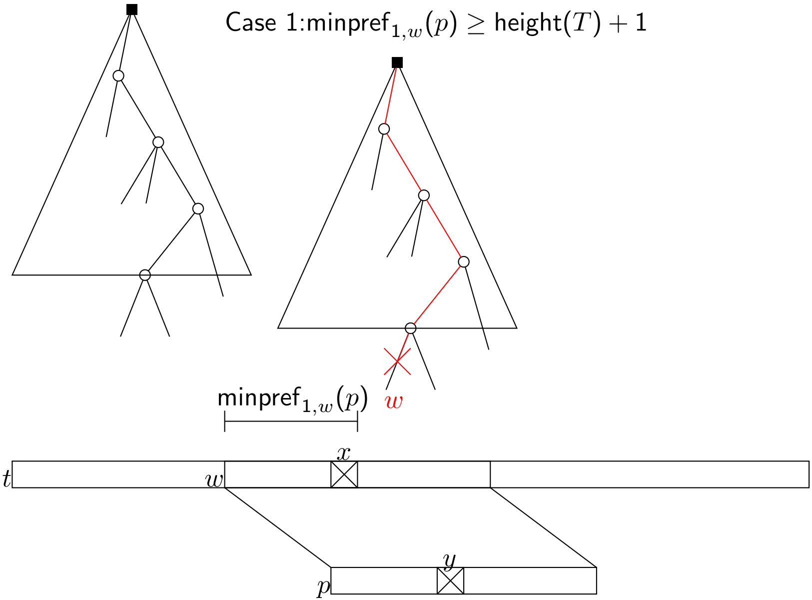


Searching with One Error

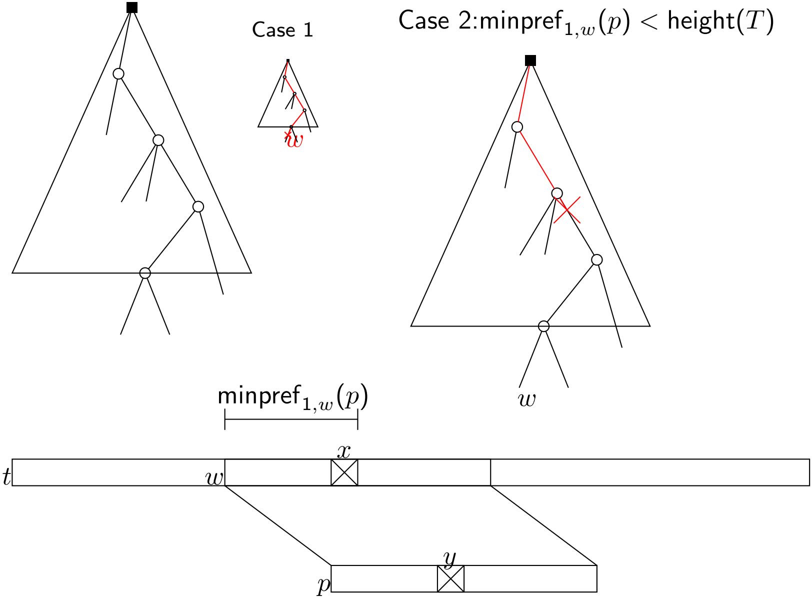




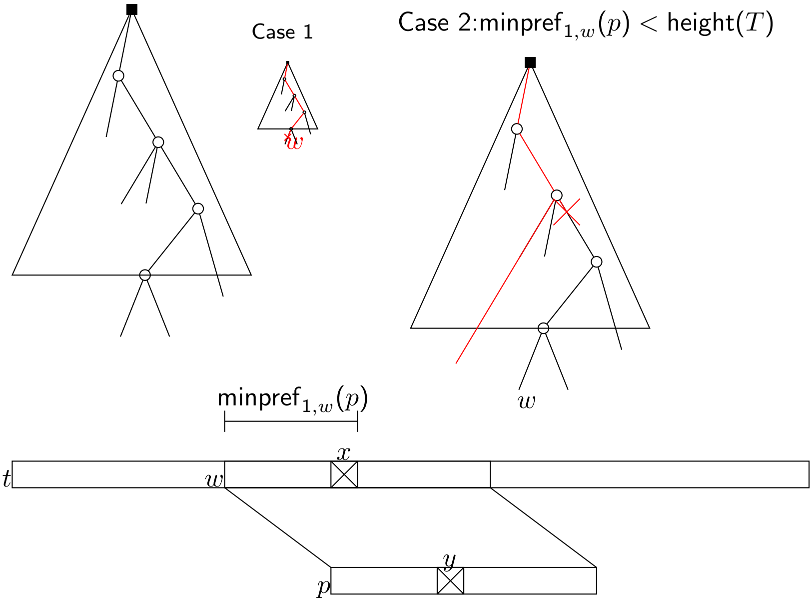
Searching with One Error



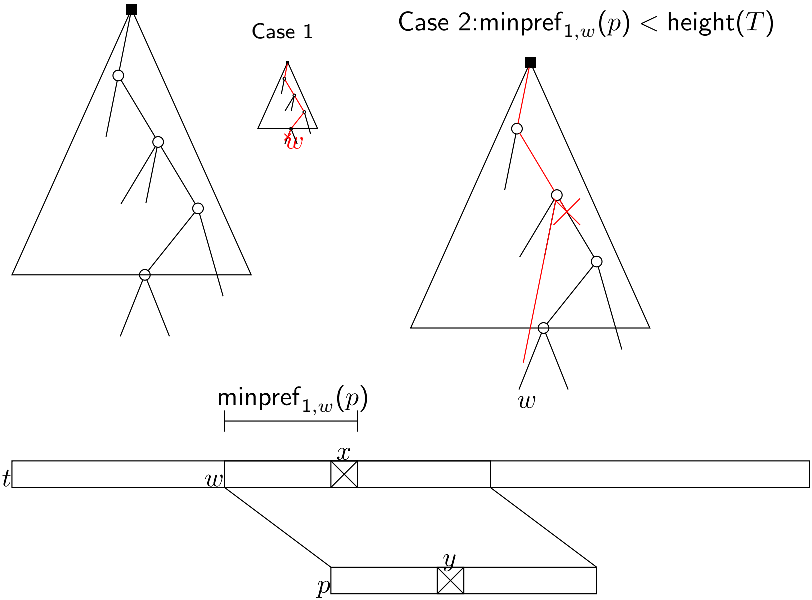
Searching with One Error



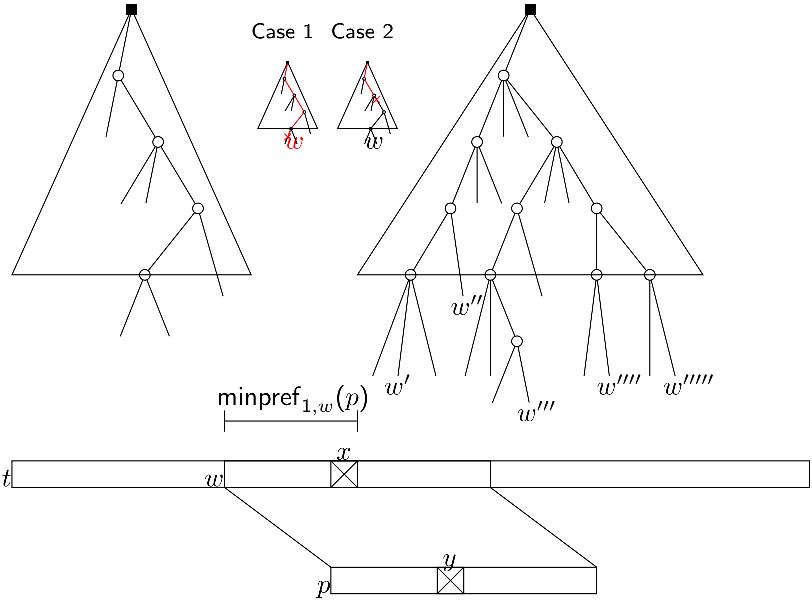
Searching with One Error



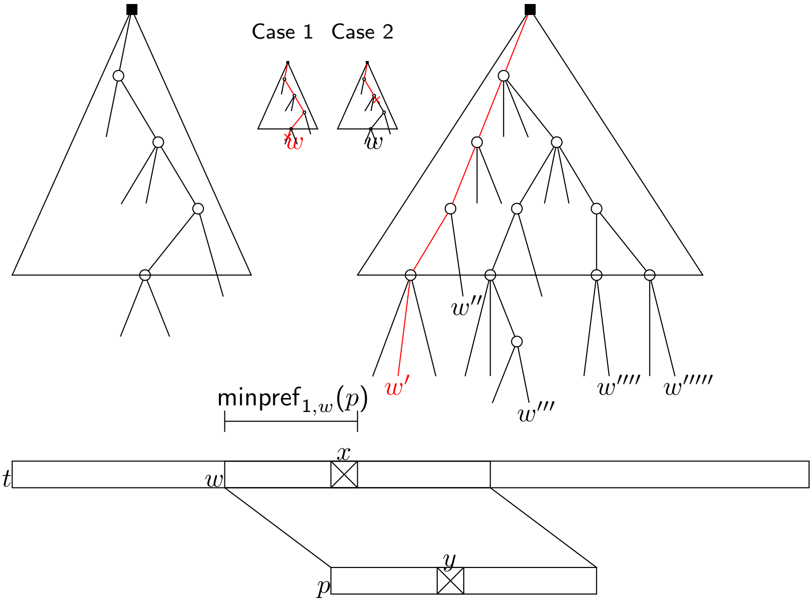
Searching with One Error



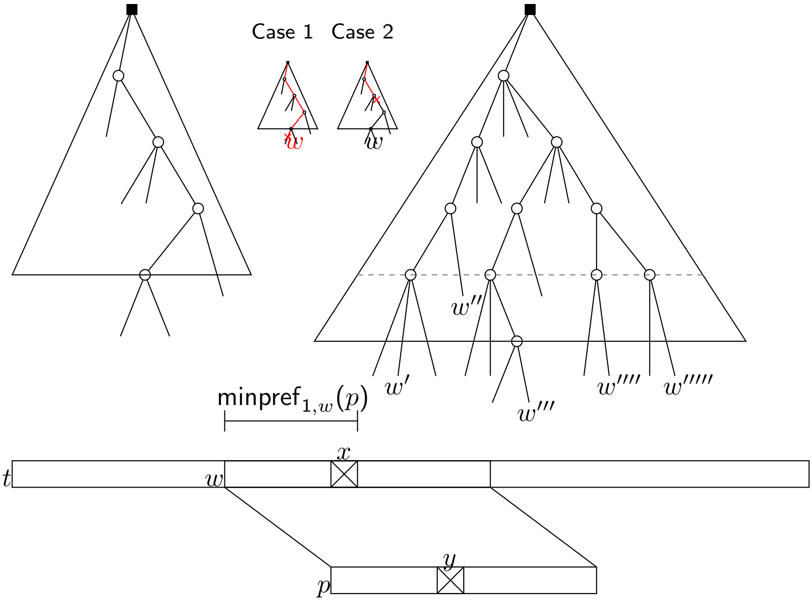
Searching with One Error



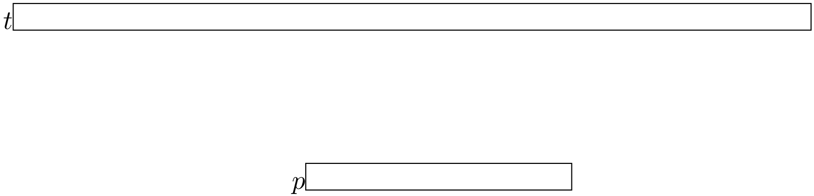
Searching with One Error



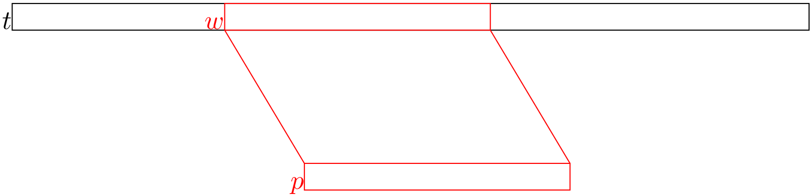
Searching with One Error



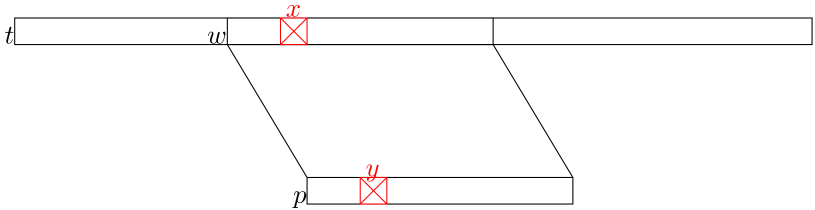
Searching with More Errors



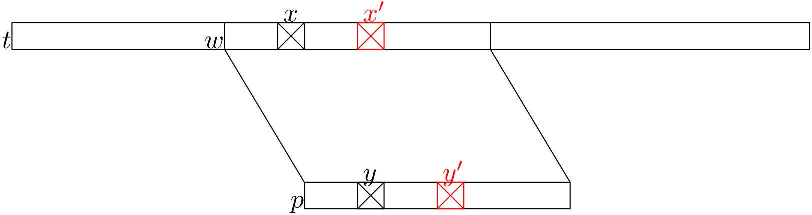
Searching with More Errors



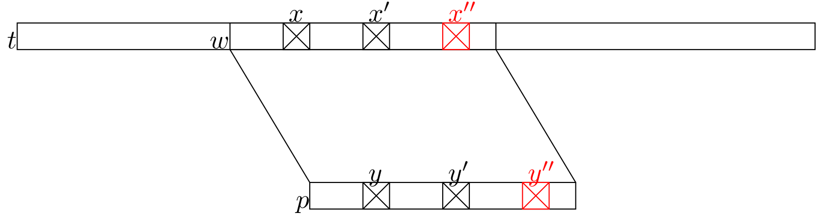
Searching with More Errors



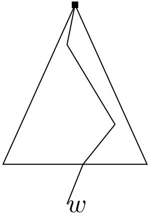
Searching with More Errors

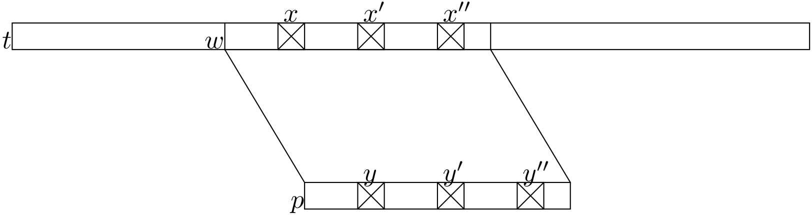


Searching with More Errors

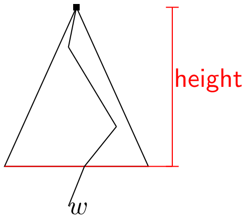


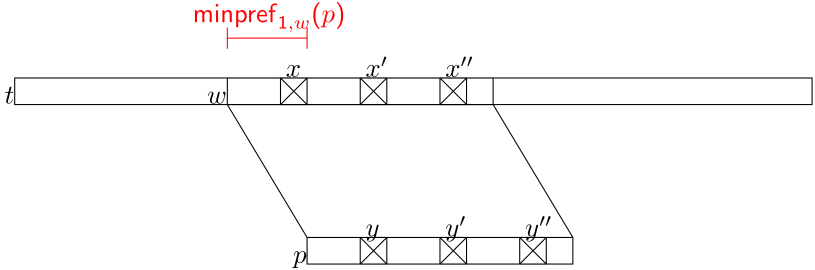
Searching with More Errors



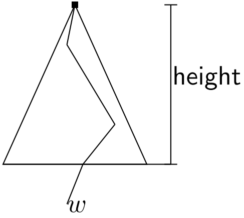


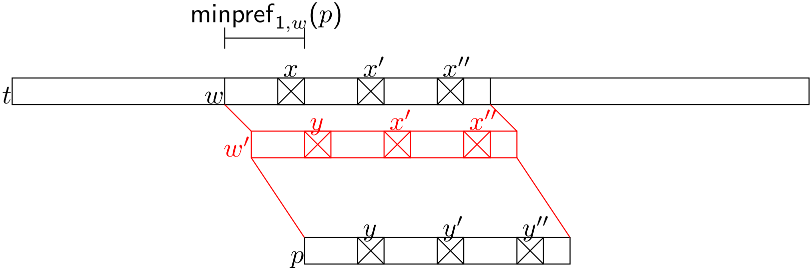
Searching with More Errors



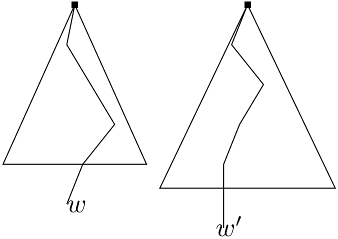


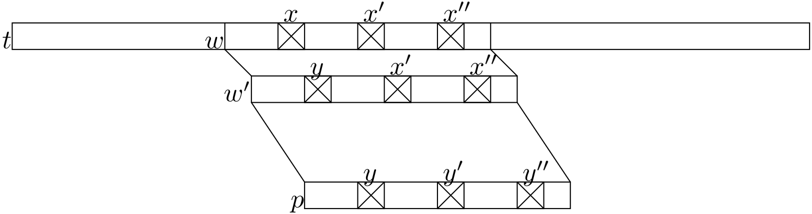
Searching with More Errors



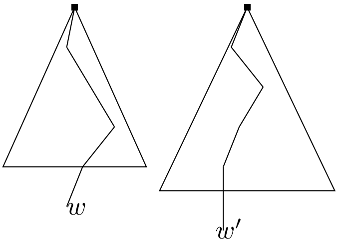


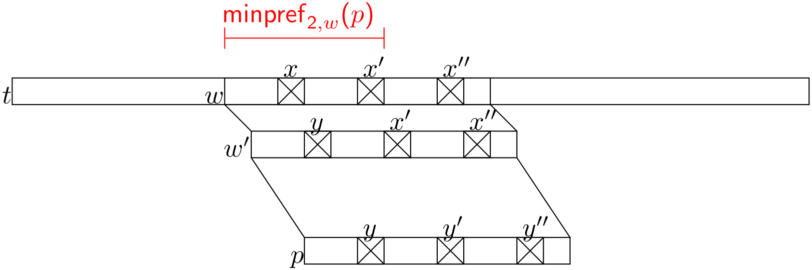
Searching with More Errors



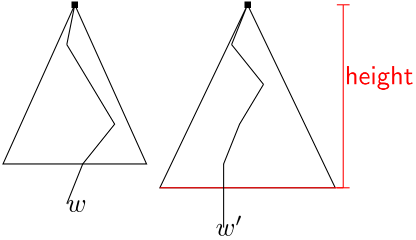


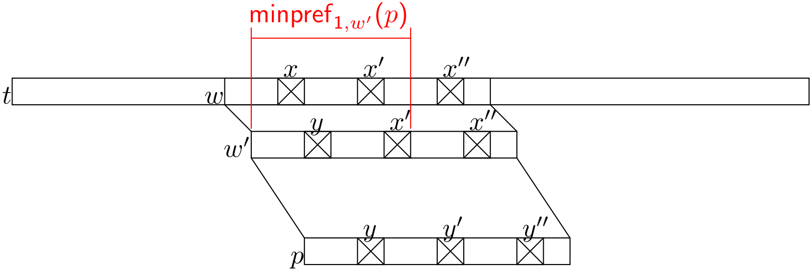
Searching with More Errors



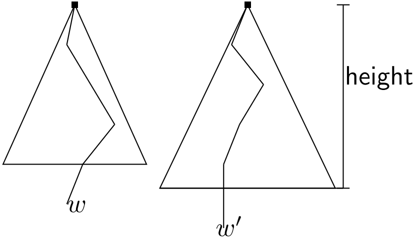


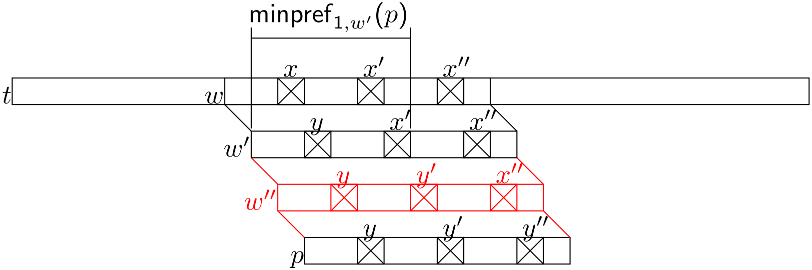
Searching with More Errors



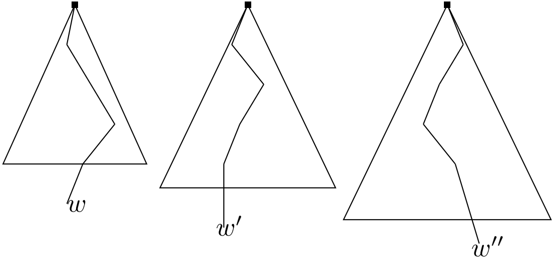


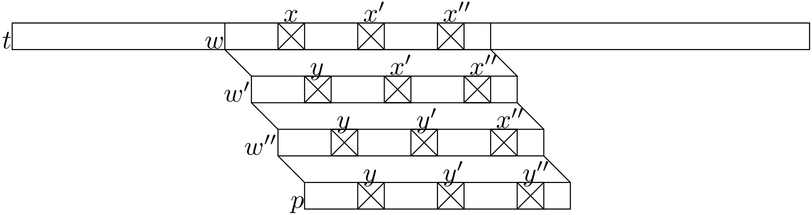
Searching with More Errors



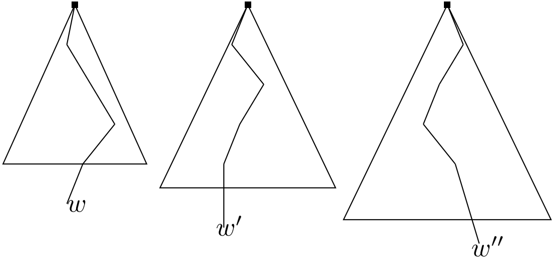


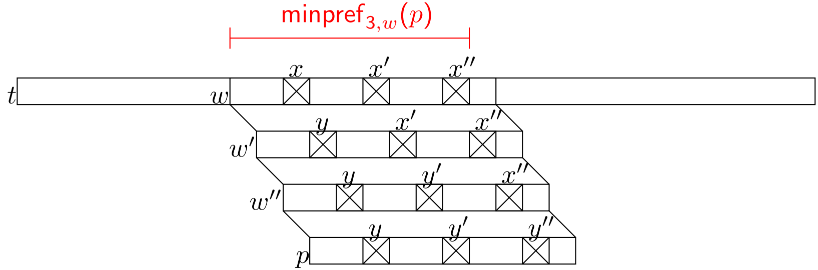
Searching with More Errors



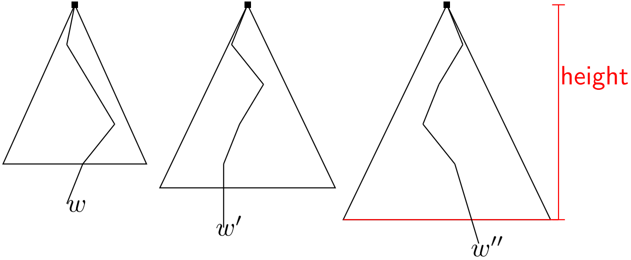


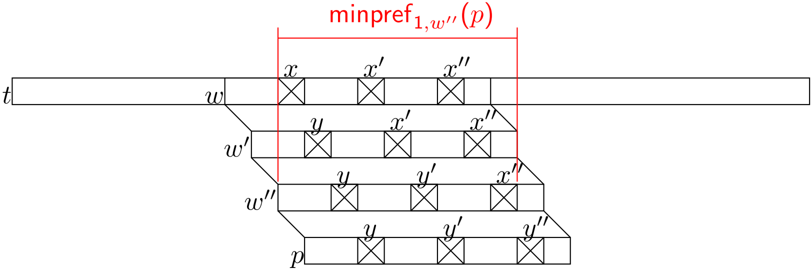
Searching with More Errors



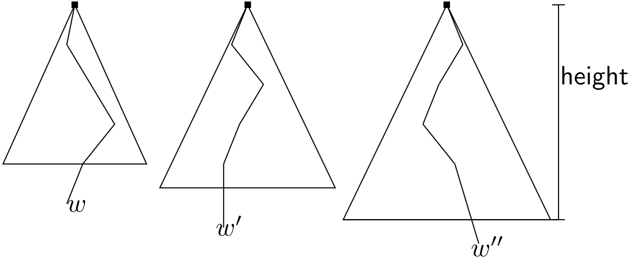


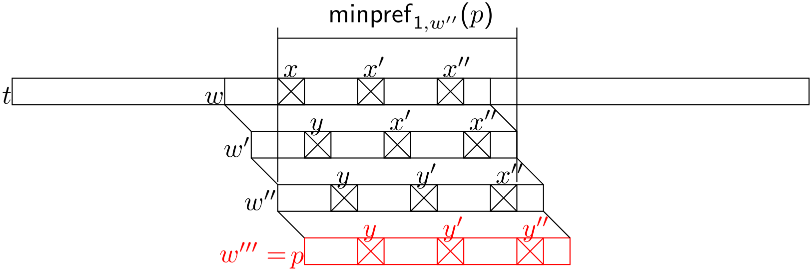
Searching with More Errors



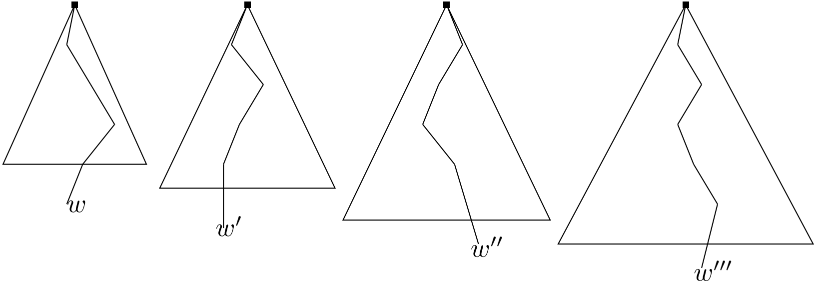


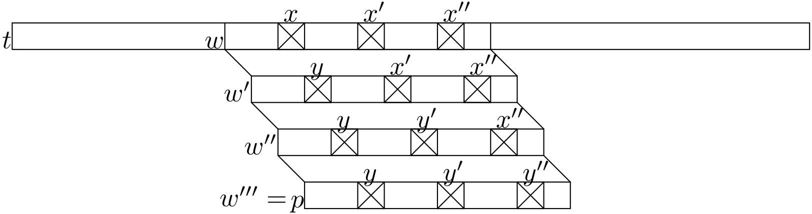
Searching with More Errors





Searching with More Errors





Leaves of error trees corresponding to matches

When matching the pattern in the i-th error tree we may

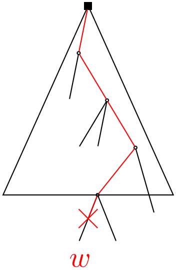
1. reach a leaf-edge
2. reach the end of the pattern

Leaves of error trees corresponding to matches

When matching the pattern in the i-th error tree we may

1. reach a leaf-edge
2. reach the end of the pattern

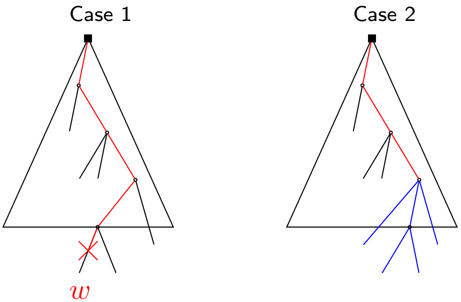
Case 1



Leaves of error trees corresponding to matches

When matching the pattern in the i-th error tree we may

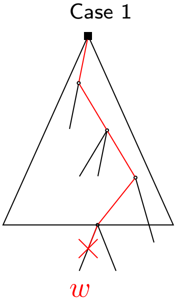
1. reach a leaf-edge, or
2. reach the end of the pattern

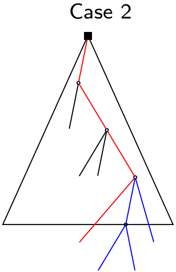


Leaves of error trees corresponding to matches

When matching the pattern in the i-th error tree we may

1. reach a leaf-edge, or
2. reach the end of the pattern and search the complete subtree.

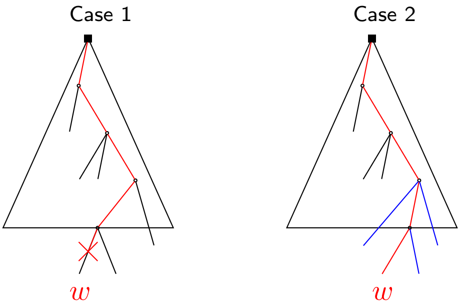




Leaves of error trees corresponding to matches

When matching the pattern in the i-th error tree we may

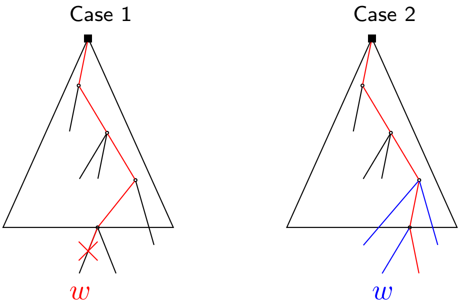
1. reach a leaf-edge, or
2. reach the end of the pattern and search the complete subtree.



Leaves of error trees corresponding to matches

When matching the pattern in the i-th error tree we may

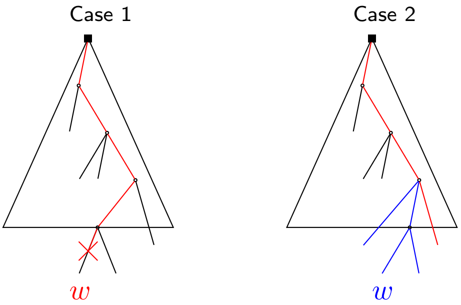
1. reach a leaf-edge, or
2. reach the end of the pattern and search the complete subtree.



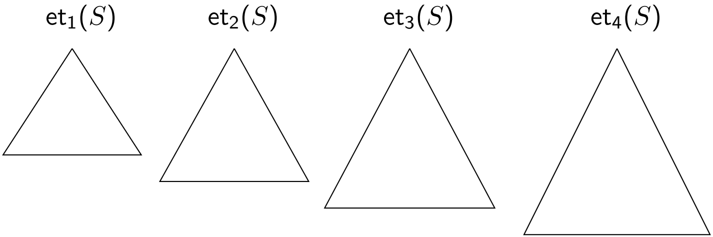
Leaves of error trees corresponding to matches

When matching the pattern in the i-th error tree we may

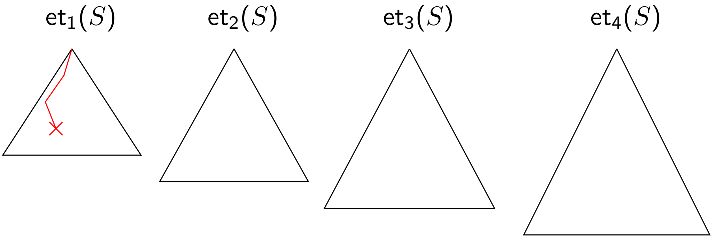
1. reach a leaf-edge, or
2. reach the end of the pattern and search the complete subtree.



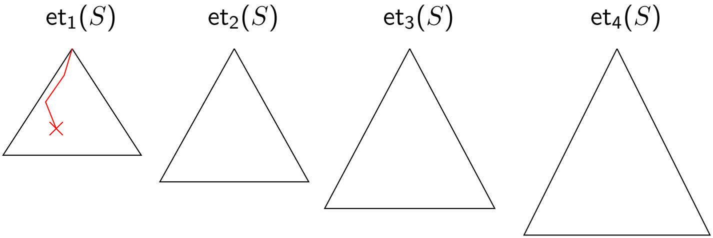
Search Time



Search Time

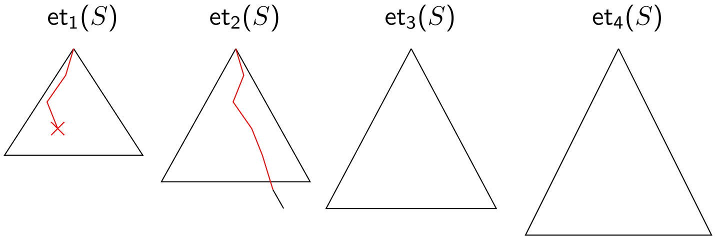


Search Time



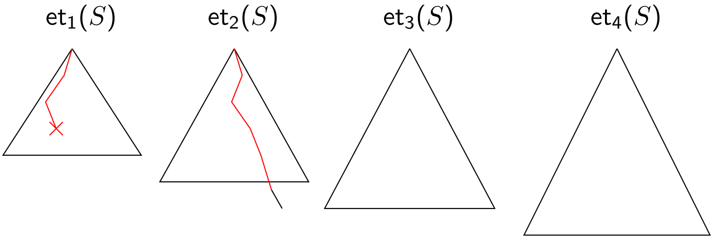
$O(m)$

Search Time



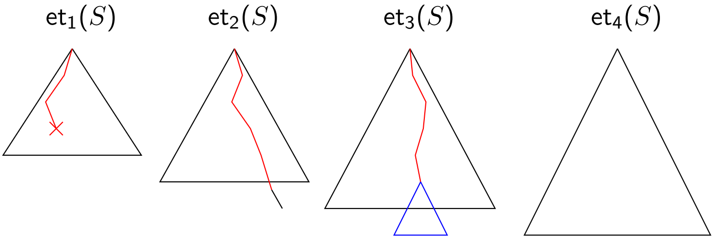
$O(m)$

Search Time



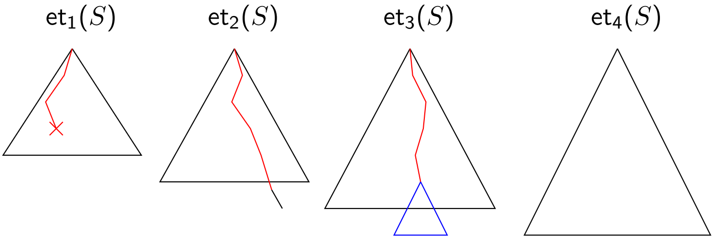
$O(m)$ $O(m + km)$

Search Time



$O(m)$ $\quad$ $O(m + km)$

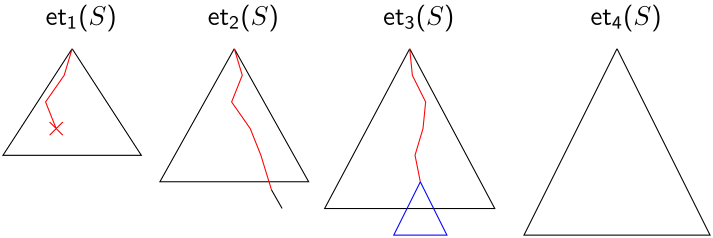
Search Time



Range queries!

$O(m)$ $O(m + km)$

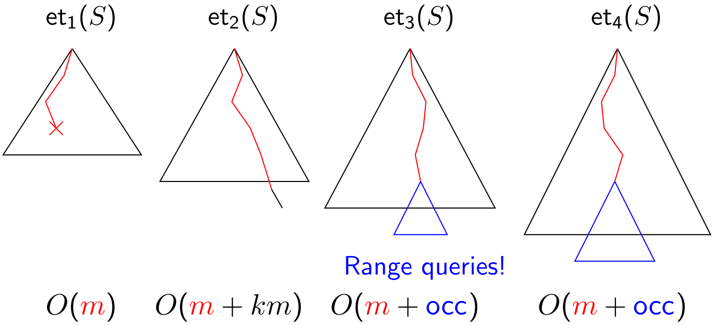
Search Time



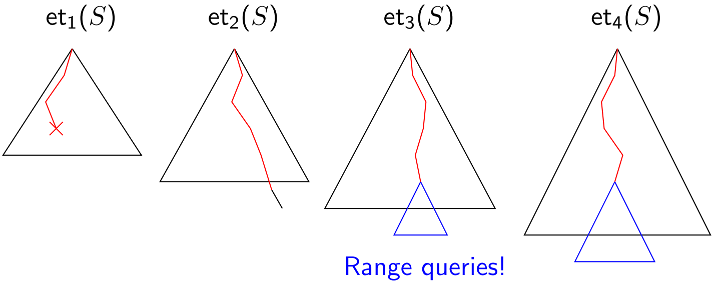
Range queries!

$O(m)$ $O(m + km)$ $O(m + \mathrm{occ})$

Search Time


et$_1(S)$
et$_2(S)$
et$_3(S)$
et$_4(S)$
Range queries!
$O(m)$
$O(m + km)$
$O(m + \text{occ})$
$O(m + \text{occ})$

Search Time



$O(m)$ $O(m + km)$ $O(m + \mathrm{occ})$ $O(m + \mathrm{occ})$

$$\Rightarrow O(m + \mathrm{occ})$$

Range Queries

Input: Array A of size n containing integer values. $< \text{Query}, \text{Preproc.} >$

RMQ: (Range Minimum Query) For (i,j) find index l with $< O(1), O(n) >$

$$A[l] = \min_{i \le k \le j} \{A[k]\}.$$

BVRQ: (Bounded Value Range Query) For (i,j,k) find all indexes $< O(|L|), O(n) >$

$$L = \{l \mid i \le l \le j \text{ and } A[l] \le k\}.$$

CRQ: (Colored Range Query) For (i,j) find the distinct set of numbers $< O(|C|), O(n) >$

$$C = \{A[l] \mid i \le l \le j\}.$$

Range Queries

Input: Array A of size n containing integer values. $< \text{Query}, \text{Preproc.} >$

RMQ: (Range Minimum Query) For (i, j) find index l with $< O(1), O(n) >$

$$A[l] = \min_{i \le k \le j} \{A[k]\}.$$

BVRQ: (Bounded Value Range Query) For (i, j, k) find all indexes $< O(|L|), O(n) >$

$$L = \{l \mid i \le l \le j \text{ and } A[l] \le k\}.$$

CRQ: (Colored Range Query) For (i, j) find the distinct set of numbers $< O(|C|), O(n) >$

$$C = \{A[l] \mid i \le l \le j\}.$$

Range Queries

Input: Array A of size n containing integer values. $< \text{Query}, \text{Preproc.} >$

RMQ: (Range Minimum Query) For (i, j) find index l with $< O(1), O(n) >$

$$A[l] = \min_{i \le k \le j} \{A[k]\}.$$

BVRQ: (Bounded Value Range Query) For (i, j, k) find all indexes $< O(|L|), O(n) >$

$$L = \{l \mid i \le l \le j \text{ and } A[l] \le k\}.$$

CRQ: (Colored Range Query) For (i, j) find the distinct set of numbers $< O(|C|), O(n) >$

$$C = \{A[l] \mid i \le l \le j\}.$$

Range Queries

Input: Array A of size n containing integer values. $< \mathrm{Query}, \mathrm{Preproc.} >$

RMQ: (Range Minimum Query) For (i, j) find index l with $< O(1), O(n) >$

$$A[l] = \min_{i \leq k \leq j} \{A[k]\}.$$

BVRQ: (Bounded Value Range Query) For (i, j, k) find all indexes $< O(|L|), O(n) >$

$$L = \{l \mid i \leq l \leq j \text{ and } A[l] \leq k\}.$$

CRQ: (Colored Range Query) For (i, j) find the distinct set of numbers $< O(|C|), O(n) >$

$$C = \{A[l] \mid i \leq l \leq j\}.$$

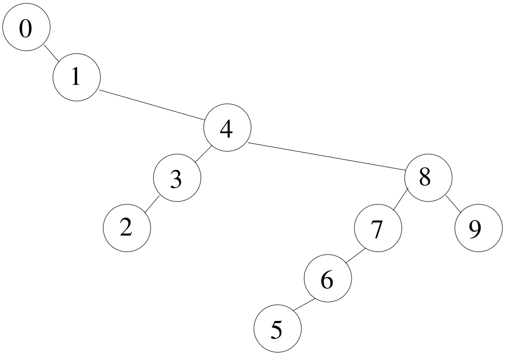
Range Minimum Queries

0	1	2	3	4	5	6	7	8	9
0	1	7	4	2	9	8	5	3	6

RMQ

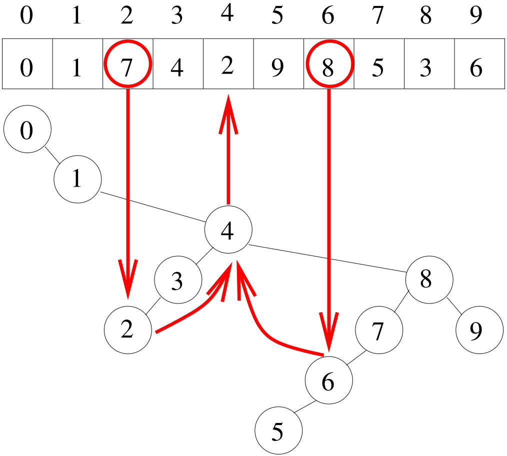
Range Minimum Queries

0	1	2	3	4	5	6	7	8	9
0	1	7	4	2	9	8	5	3	6



RMQ
=
Cartesian Tree
(in $O(n)$)

Range Minimum Queries



RMQ

$=$

Cartesian Tree

(in $O(n)$)

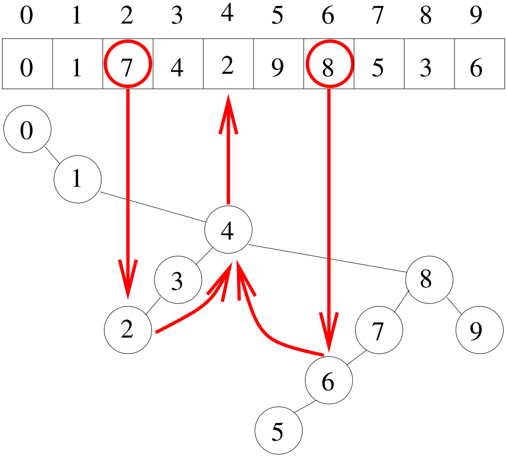
$+$

constant time

LCA

(in $O(n)$)

Range Minimum Queries



RMQ

$=$

Cartesian Tree

(in $O(n)$)

$+$

constant time

LCA

(in $O(n)$)

$\Rightarrow$Query time

is $O(1)$

BVRG

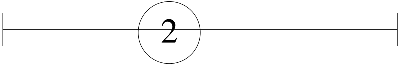
Bounded value range queries are solved through successive RMQs, e.g., querying $(2, 7, 6)$:

The querytree is binary and has $|L|$ inner nodes, hence we need $O(|L|)$ RMQs in total.

BVRG

Bounded value range queries are solved through successive RMQs, e.g., querying $(2, 7, 6)$:

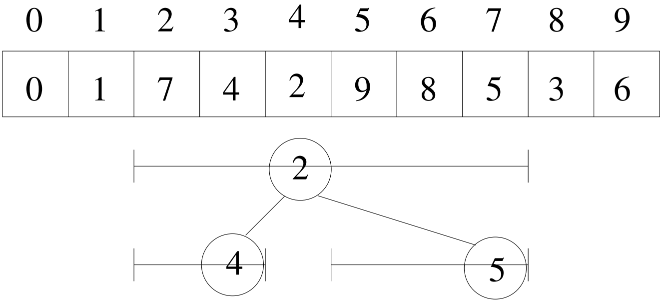
0	1	2	3	4	5	6	7	8	9
0	1	7	4	2	9	8	5	3	6



The querytree is binary and has $|L|$ inner nodes, hence we need $O(|L|)$ RMQs in total.

BVRG

Bounded value range queries are solved through successive RMQs, e.g., querying $(2, 7, 6)$:

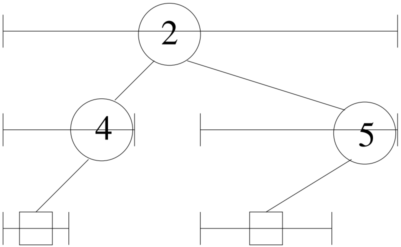


The querytree is binary and has $|L|$ inner nodes, hence we need $O(|L|)$ RMQs in total.

BVRG

Bounded value range queries are solved through successive RMQs, e.g., querying $(2, 7, 6)$:

0	1	2	3	4	5	6	7	8	9
0	1	7	4	2	9	8	5	3	6

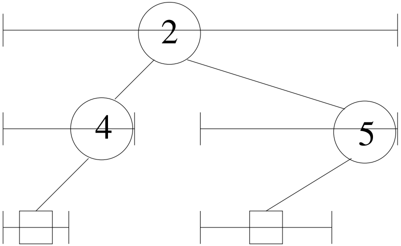


The querytree is binary and has $|L|$ inner nodes, hence we need $O(|L|)$ RMQs in total.

BVRG

Bounded value range queries are solved through successive RMQs, e.g., querying $(2,7,6)$:

0	1	2	3	4	5	6	7	8	9
0	1	7	4	2	9	8	5	3	6



The querytree is binary and has $|L|$ inner nodes, hence we need $O(|L|)$ RMQs in total.

CRQ

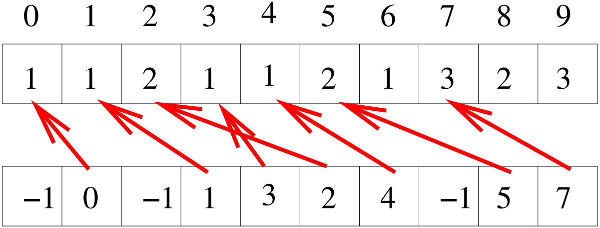
Colored range queries are reduced by using another array which stores the last index of each value. A CRQ (i, j) then transforms into a BVRQ $(i, j, i-1)$ on the new array:

0	1	2	3	4	5	6	7	8	9
1	1	2	1	1	2	1	3	2	3

−1	0	−1	1	3	2	4	−1	5	7

CRQ

Colored range queries are reduced by using another array which stores the last index of each value. A CRQ (i, j) then transforms into a BVRQ $(i, j, i-1)$ on the new array:



Error Sets

Definition (Error Sets)

Defined inductively by $W_0 = S$ and

$$W_i = \Gamma_{h_{i-1}}(W_{i-1}) \cap S_i \quad ,$$

where

- $\Gamma_l = \{op(u)v \mid uv \in A, |op(u)| \le l + 1, op \in \{\mathrm{del}, \mathrm{ins}, \mathrm{sub}\}\}$,
- $S_i = \{r \mid \text{there exists } s \in S \text{ such that } d(s, r) = i\}$,
- and h_i are arbitrary integer parameters.

Error Sets

Definition (Error Sets)

Defined inductively by $W_0 = S$ and

$$W_i = \Gamma_{h_{i-1}}(W_{i-1}) \cap S_i \quad ,$$

where

- $\Gamma_l = \{op(u)v \mid uv \in A, |op(u)| \leq l+1, op \in \{\mathsf{del}, \mathsf{ins}, \mathsf{sub}\}\}$,
- $S_i = \{r \mid \text{there exists } s \in S \text{ such that } \mathsf{d}(s,r) = i\}$,
- and h_i are arbitrary integer parameters.

Error Sets

Definition (Error Sets)

Defined inductively by $W_0 = S$ and

$$W_i = \Gamma_{h_{i-1}}(W_{i-1}) \cap S_i \ ,$$

where

- $\Gamma_l = \{op(u)v \mid uv \in A, |op(u)| \leq l+1, op \in \{\mathsf{del}, \mathsf{ins}, \mathsf{sub}\}\}$,
- $S_i = \{r \mid \text{there exists } s \in S \text{ such that } \mathsf{d}(s,r) = i\}$,
- and h_i are arbitrary integer parameters.

Error Sets

Definition (Error Sets)

Defined inductively by $W_0 = S$ and

$$W_i = \Gamma_{h_{i-1}}(W_{i-1}) \cap S_i \quad ,$$

where

- $\Gamma_l = \{op(u)v \mid uv \in A, |op(u)| \leq l + 1, op \in \{\mathsf{del}, \mathsf{ins}, \mathsf{sub}\}\}$,
- $S_i = \{r \mid \text{there exists } s \in S \text{ such that } \mathsf{d}(s, r) = i\}$,
- and h_i are arbitrary integer parameters.

Error Sets

Definition (Error Sets)

Defined inductively by $W_0 = S$ and

$$W_i = \Gamma_{h_{i-1}}(W_{i-1}) \cap S_i \quad ,$$

where

- $\Gamma_l = \{op(u)v \mid uv \in A, |op(u)| \leq l+1, op \in \{\mathsf{del}, \mathsf{ins}, \mathsf{sub}\}\}$,
- $S_i = \{r \mid \text{there exists } s \in S \text{ such that } \mathsf{d}(s,r) = i\}$,
- and h_i are arbitrary integer parameters.

Error Tree

- Defined as trie for W_i.
- Label leaf x by suffix number and $l = \text{minpref}_{k,s}(\text{path}(x))$.
- Leaf is a hit if pattern length greater than l.

Error Tree

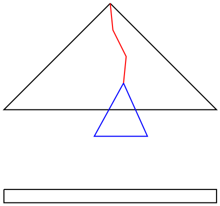
- Defined as trie for W_i.
- Label leaf x by suffix number and $l = \text{minpref}_{k,s}(\text{path}(x))$.
- Leaf is a hit if pattern length greater than l.

Error Tree

- Defined as trie for W_i.
- Label leaf x by suffix number and $l = \text{minpref}_{k,s}(\text{path}(x))$.
- Leaf is a hit if pattern length greater than l.

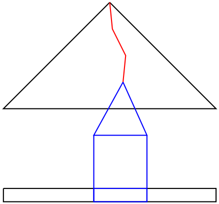
Result Selection using Range Queries

- Array of leaves A, additional array B.
- Subtree ranges at nodes.
- B contains minimal prefix l $\Rightarrow$ occurrences
- B contains starting positions $\Rightarrow$ positions
- B contains string identifiers $\Rightarrow$ documents



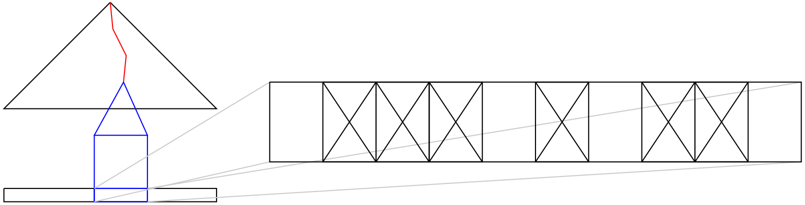
Result Selection using Range Queries

- Array of leaves A, additional array B.
- Subtree ranges at nodes.
- B contains minimal prefix l $\Rightarrow$ occurrences
- B contains starting positions $\Rightarrow$ positions
- B contains string identifiers $\Rightarrow$ documents



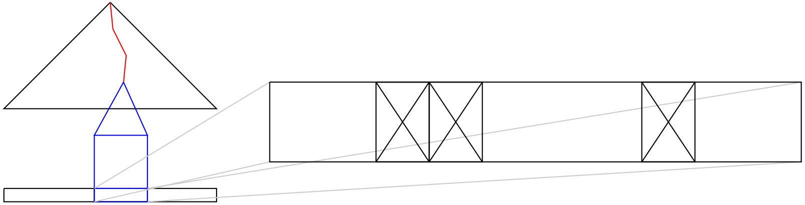
Result Selection using Range Queries

- Array of leaves A, additional array B.
- Subtree ranges at nodes.
- B contains minimal prefix l $\Rightarrow$ occurrences
- B contains starting positions $\Rightarrow$ positions
- B contains string identifiers $\Rightarrow$ documents



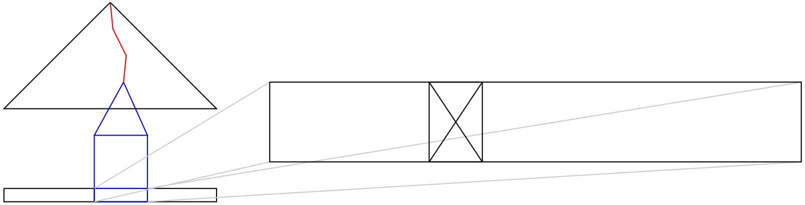
Result Selection using Range Queries

- Array of leaves A, additional array B.
- Subtree ranges at nodes.
- B contains minimal prefix l $\Rightarrow$ occurrences
- B contains starting positions $\Rightarrow$ positions
- B contains string identifiers $\Rightarrow$ documents



Result Selection using Range Queries

- Array of leaves A, additional array B.
- Subtree ranges at nodes.
- B contains minimal prefix l $\Rightarrow$ occurrences
- B contains starting positions $\Rightarrow$ positions
- B contains string identifiers $\Rightarrow$ documents



Basic Bounds

- Set $h_i = \text{maxpref}(W_i)$ (the height of $\text{et}_i(S)$)
- The error sets have size $|W_i| = O(h_0 h_1 \cdots h_{i-1} |S|)$
- Construction time for $\text{et}_i(S)$ is $O(h_0 h_1 \cdots h_i |S|)$
- $h_i = O(\log n)$ in expectation and with high probability

Basic Bounds

- Set $h_i = \text{maxpref}(W_i)$ (the height of $\text{et}_i(S)$)
- The error sets have size $|W_i| = O(h_0 h_1 \cdots h_{i-1} |S|)$
- Construction time for $\text{et}_i(S)$ is $O(h_0 h_1 \cdots h_i |S|)$
- $h_i = O(\log n)$ in expectation and with high probability

Basic Bounds

- Set $h_i = \text{maxpref}(W_i)$ (the height of $\text{et}_i(S)$)
- The error sets have size $|W_i| = O(h_0 h_1 \cdots h_{i-1} |S|)$
- Construction time for $\text{et}_i(S)$ is $O(h_0 h_1 \cdots h_i |S|)$
- $h_i = O(\log n)$ in expectation and with high probability

Basic Bounds

- Set $h_i = \text{maxpref}(W_i)$ (the height of $\text{et}_i(S)$)
- The error sets have size $|W_i| = O(h_0 h_1 \cdots h_{i-1} |S|)$
- Construction time for $\text{et}_i(S)$ is $O(h_0 h_1 \cdots h_i |S|)$
- $h_i = O(\log n)$ in expectation and with high probability

Random Model

- Stationary ergodic source
- *mixing* condition

$$c_1 \Pr\{\mathcal{A}\} \Pr\{\mathcal{B}\} \leq \Pr\{\mathcal{A} \cap \mathcal{B}\} \leq c_2 \Pr\{\mathcal{A}\} \Pr\{\mathcal{B}\}.$$

- Probability of common substring:

$$r_2 = \lim_{n \to \infty} \frac{-\ln\left(\sum_{w \in \Sigma^n} (\Pr\{w\})^2\right)}{2n}.$$

Random Model

- Stationary ergodic source
- *mixing* condition

$$c_1 \mathsf{Pr}\{\mathcal{A}\}\mathsf{Pr}\{\mathcal{B}\} \leq \mathsf{Pr}\{\mathcal{A} \cap \mathcal{B}\} \leq c_2 \mathsf{Pr}\{\mathcal{A}\}\mathsf{Pr}\{\mathcal{B}\}.$$

- Probability of common substring:

$$r_2 = \lim_{n\to\infty} \frac{-\ln\left(\sum_{w\in\Sigma^n} (\mathsf{Pr}\{w\})^2\right)}{2n}.$$

Random Model

- Stationary ergodic source
- *mixing* condition

$$c_1 \mathsf{Pr}\left\{\mathcal{A}\right\}\mathsf{Pr}\left\{\mathcal{B}\right\} \leq \mathsf{Pr}\left\{\mathcal{A} \cap \mathcal{B}\right\} \leq c_2 \mathsf{Pr}\left\{\mathcal{A}\right\}\mathsf{Pr}\left\{\mathcal{B}\right\}.$$

- Probability of common substring:

$$r_2 = \lim_{n\to\infty} \frac{-\ln\left(\sum_{w\in\Sigma^n}\left(\mathsf{Pr}\left\{w\right\}\right)^2\right)}{2n} \ .$$

Height of Error Trees

- Let l be the expected length of a common substring.
- Each step adds one error to the strings.
- Possibly more matching characters afterwards, expected increment is at most l.
- $\Rightarrow h_i \leq (2i+1)l$.
- Thus,

$$\Pr\{l_{max} > (2k+1)(c''h+1)\} \leq c'k^2h^2 e^{-2(1+\epsilon)h}$$

as $h \to \infty$.

Height of Error Trees

- Let l be the expected length of a common substring.
- Each step adds one error to the strings.
- Possibly more matching characters afterwards, expected increment is at most l.
- $\Rightarrow h_i \leq (2i+1)l$.
- Thus,

$$\Pr\{l_{max} > (2k+1)(c''h+1)\} \leq c'k^2h^2e^{-2(1+\epsilon)h}$$

as $h \to \infty$.

Height of Error Trees

- Let l be the expected length of a common substring.
- Each step adds one error to the strings.
- Possibly more matching characters afterwards, expected increment is at most l.
- $\Rightarrow h_i \leq (2i+1)l$.
- Thus,

$$\Pr\{l_{max} > (2k+1)(c''h+1)\} \leq c'k^2h^2e^{-2(1+\epsilon)h}$$

as $h \to \infty$.

Height of Error Trees

- Let l be the expected length of a common substring.
- Each step adds one error to the strings.
- Possibly more matching characters afterwards, expected increment is at most l.
- $\Rightarrow h_i \leq (2i+1)l$.
- Thus,

$$\Pr\left\{l_{max} > (2k+1)(c''h+1)\right\} \leq c'k^2h^2e^{-2(1+\epsilon)h}$$

as $h \longrightarrow \infty$.

Height of Error Trees

- Let l be the expected length of a common substring.
- Each step adds one error to the strings.
- Possibly more matching characters afterwards, expected increment is at most l.
- $\Rightarrow h_i \leq (2i+1)l$.
- Thus,

$$\Pr\left\{l_{max} > (2k+1)(c''h+1)\right\} \leq c'k^2h^2e^{-2(1+\epsilon)h}$$

as $h \to \infty$.

Size and Preprocessing

- the expected height of the first tree is $l = O(\log n)$.
- Conditioning on $l \leq c \log n$ yields the expected size

$$\begin{aligned}
\mathbf{E}[n \cdot h^k] &\leq n(ck \ln n)^k + \sum_{i=ck \ln n}^{\infty} n i^k \Pr\{l_{max} > i\} \\
&< n(ck \ln n)^k + O(1) \\
&= O(n \log^k n).
\end{aligned}$$

Size and Preprocessing

- the expected height of the first tree is $l = O(\log n)$.
- Conditioning on $l \leq c \log n$ yields the expected size

$$
\begin{aligned}
\mathbf{E}[n \cdot h^k] &\leq n(ck \ln n)^k + \sum_{i=ck \ln n}^{\infty} n i^k \Pr\{l_{max} > i\} \\
&< n(ck \ln n)^k + O(1) \\
&= O(n \log^k n).
\end{aligned}
$$

Size and Preprocessing

- the expected height of the first tree is $l = O(\log n)$.
- Conditioning on $l \leq c \log n$ yields the expected size

$$\begin{aligned}
\mathbf{E}[n \cdot h^k] &\leq n(ck \ln n)^k + \sum_{i=ck \ln n}^{\infty} n i^k \Pr\{l_{max} > i\} \\
&< n(ck \ln n)^k + O(1) \\
&= O(n \log^k n).
\end{aligned}$$

Size and Preprocessing

- the expected height of the first tree is $l = O(\log n)$.
- Conditioning on $l \leq c \log n$ yields the expected size

$$\begin{aligned} \mathbf{E}[n \cdot h^k] &\leq n(ck \ln n)^k + \sum_{i=ck \ln n}^{\infty} n i^k \Pr\{l_{max} > i\} \\ &< n(ck \ln n)^k + O(1) \\ &= O(n \log^k n). \end{aligned}$$

Size and Preprocessing

- the expected height of the first tree is $l = O(\log n)$.
- Conditioning on $l \leq c \log n$ yields the expected size

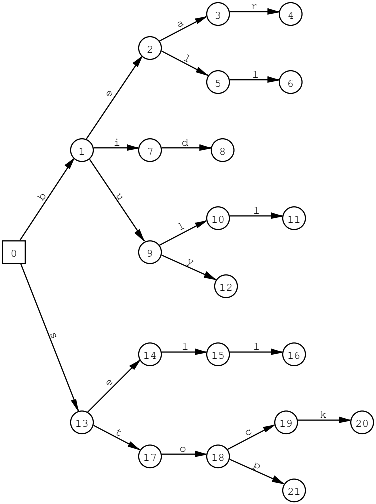
$$\begin{aligned} \mathbf{E}[n \cdot h^k] &\leq n(ck \ln n)^k + \sum_{i=ck \ln n}^{\infty} n i^k \Pr\{l_{max} > i\} \\ &< n(ck \ln n)^k + O(1) \\ &= O(n \log^k n). \end{aligned}$$

Weak Tries

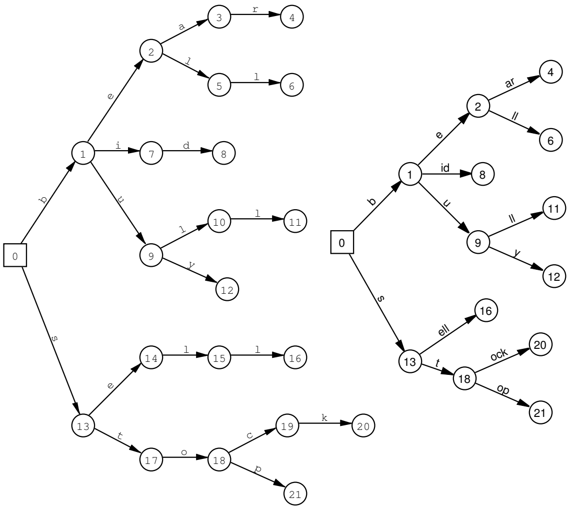
Definition (Weak Trie)

For $l > 0$, the l-weak trie for a set of strings $S \subset \Sigma^*$ is a rooted tree with edges labeled by characters from Σ. For any node with a depth less than l, all outgoing edges are labeled by different characters, and there are no branching nodes with a depth of more than l. Each path from the root to a leaf can be read as a string from S.

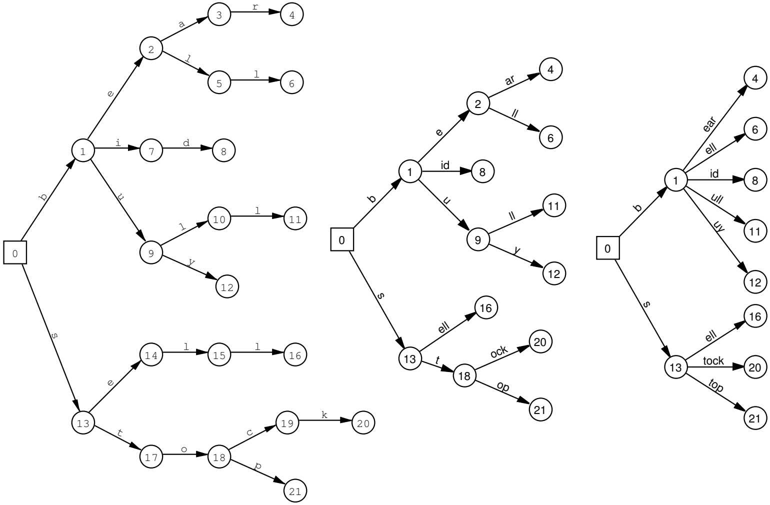
Tries and Weak Tries



Tries and Weak Tries



Tries and Weak Tries



Error Tree

- Define the error trie as h_i-weak trie:

$$\mathsf{et}_i(S) = \mathcal{W}_{h_i}(W_i)$$

- Leaf labels are the same as before: leaf x is labeled by suffix number and $l = \mathsf{minpref}_{k,s}(\mathsf{path}(x))$.
- Leaf is a hit if pattern length greater than l.
- Order of leaves is not important for range queries.

Error Tree

- Define the error trie as h_i-weak trie:

$$\mathsf{et}_i\left(S\right) = \mathcal{W}_{h_i}\left(W_i\right)$$

- Leaf labels are the same as before: leaf x is labeled by suffix number and $l = \text{minpref}_{k,s}(\text{path}(x))$.
- Leaf is a hit if pattern length greater than l.
- Order of leaves is not important for range queries.

Error Tree

- Define the error trie as h_i-weak trie:

$$\mathrm{et}_i\left(S\right) = \mathcal{W}_{h_i}\left(W_i\right)$$

- Leaf labels are the same as before: leaf x is labeled by suffix number and $l = \mathrm{minpref}_{k,s}(\mathrm{path}(x))$.
- Leaf is a hit if pattern length greater than l.
- Order of leaves is not important for range queries.

Error Tree

- Define the error trie as h_i-weak trie:

$$\mathrm{et}_i\left(S\right) = \mathcal{W}_{h_i}\left(W_i\right)$$

- Leaf labels are the same as before: leaf x is labeled by suffix number and $l = \mathrm{minpref}_{k,s}(\mathrm{path}(x))$.
- Leaf is a hit if pattern length greater than l.
- Order of leaves is not important for range queries.

Basic Bounds

Data structure:

- Set $h_i = ck \log n + i$.
- The size is $O(n \log^k n)$.
- The preprocessing takes $O(n \log^{k+1} n)$ (weak tries!).

Searching:

- For $m \leq ck \log n$: as before in $O(m + \text{occ})$ time.
- For $m > ck \log n$: use a generalized suffix tree (as a filter).
- Expected search time: $O(1)$.

Basic Bounds

Data structure:

- Set $h_i = ck \log n + i$.
- The size is $O(n \log^k n)$.
- The preprocessing takes $O(n \log^{k+1} n)$ (weak tries!).

Searching:

- For $m \leq ck \log n$: as before in $O(m + \text{occ})$ time.
- For $m > ck \log n$: use a generalized suffix tree (as a filter).
- Expected search time: $O(1)$.

Basic Bounds

Data structure:

- Set $h_i = ck \log n + i$.
- The size is $O(n \log^k n)$.
- The preprocessing takes $O(n \log^{k+1} n)$ (weak tries!).

Searching:

- For $m \leq ck \log n$: as before in $O(m + \mathrm{occ})$ time.
- For $m > ck \log n$: use a generalized suffix tree (as a filter).
- Expected search time: $O(1)$.

Basic Bounds

Data structure:

- Set $h_i = ck \log n + i$.
- The size is $O(n \log^k n)$.
- The preprocessing takes $O(n \log^{k+1} n)$ (weak tries!).

Searching:

- For $m \leq ck \log n$: as before in $O(m + \text{occ})$ time.
- For $m > ck \log n$: use a generalized suffix tree (as a filter).
- Expected search time: $O(1)$.

Basic Bounds

Data structure:

- Set $h_i = ck \log n + i$.
- The size is $O(n \log^k n)$.
- The preprocessing takes $O(n \log^{k+1} n)$ (weak tries!).

Searching:

- For $m \leq ck \log n$: as before in $O(m + \text{occ})$ time.
- For $m > ck \log n$: use a generalized suffix tree (as a filter).
- Expected search time: $O(1)$.

Basic Bounds

Data structure:

- Set $h_i = ck \log n + i$.
- The size is $O(n \log^k n)$.
- The preprocessing takes $O(n \log^{k+1} n)$ (weak tries!).

Searching:

- For $m \leq ck \log n$: as before in $O(m + \text{occ})$ time.
- For $m > ck \log n$: use a generalized suffix tree (as a filter).
- Expected search time: $O(1)$.

Basic Bounds

Data structure:

- Set $h_i = ck \log n + i$.
- The size is $O(n \log^k n)$.
- The preprocessing takes $O(n \log^{k+1} n)$ (weak tries!).

Searching:

- For $m \leq ck \log n$: as before in $O(m + \text{occ})$ time.
- For $m > ck \log n$: use a generalized suffix tree (as a filter).
- Expected search time: $O(1)$.

Conclusion

- Worst-case optimal search-time $O(m + \text{occ})$ for a *constant* number of errors.
- Average-case index size $O(n \log^k n)$ for k errors.
- Matches average search time $O(m \log^k n)$ with index size $O(n)$ (M 2004).
- Achieved through

Conclusion

- Worst-case optimal search-time $O(m + \text{occ})$ for a *constant* number of errors.
- Average-case index size $O(n \log^k n)$ for k errors.
- Matches average search time $O(m \log^k n)$ with index size $O(n)$ (M 2004).
- Achieved through
  - recursive case-distinction upon the location of the errors
  - range queries to select leaves from subtrees

Conclusion

- Worst-case optimal search-time $O(m + \text{occ})$ for a *constant* number of errors.
- Average-case index size $O(n \log^k n)$ for k errors.
- Matches average search time $O(m \log^k n)$ with index size $O(n)$ (M 2004).
- Achieved through
  - recursive case-distinction upon the location of the error.
  - range queries to select leaves from subtrees.

Conclusion

- Worst-case optimal search-time $O(m + \mathrm{occ})$ for a *constant* number of errors.
- Average-case index size $O(n \log^k n)$ for k errors.
- Matches average search time $O(m \log^k n)$ with index size $O(n)$ (M 2004).
- Achieved through
  - recursive case-distinction upon the location of the error.
  - range queries to select leaves from subtrees

Conclusion

- Worst-case optimal search-time $O(m + \mathrm{occ})$ for a *constant* number of errors.
- Average-case index size $O(n \log^k n)$ for k errors.
- Matches average search time $O(m \log^k n)$ with index size $O(n)$ (M 2004).
- Achieved through
  - recursive case-distinction upon the location of the error.
  - range queries to select leaves from subtrees

Conclusion

- Worst-case optimal search-time $O(m + \mathrm{occ})$ for a *constant* number of errors.
- Average-case index size $O(n \log^k n)$ for k errors.
- Matches average search time $O(m \log^k n)$ with index size $O(n)$ (M 2004).
- Achieved through
  - recursive case-distinction upon the location of the error.
  - range queries to select leaves from subtrees

Open Problems

- Can we close the gap?
  - Our new indexing structure works best for “small” patterns of length $O(\log n)$.
  - On average, “small” patterns are the hardest to find.
  - The case of “large” patterns of length $\Omega(\log^k n)$ can be handled with the data structure of Cole et. al 2004.
  - No lower bounds on the index size for $O(m + \mathrm{occ})$-time lookup are known.
- Can we use it?

Open Problems

- Can we close the gap?
  - Our new indexing structure works best for “small” patterns of length $O(\log n)$.
  - On average, “small” patterns are the hardest to find.
  - The case of “large” patterns of length $\Omega(\log^k n)$ can be handled with the data structure of Cole et. al 2004.
  - No lower bounds on the index size for $O(m + \text{occ})$-time lookup are known.
- Can we use it?

Open Problems

- Can we close the gap?
  - Our new indexing structure works best for "small" patterns of length $O(\log n)$.
  - On average, "small" patterns are the hardest to find.
  - The case of "large" patterns of length $\Omega(\log^k n)$ can be handled with the data structure of Cole et. al 2004.
  - No lower bounds on the index size for $O(m + \mathrm{occ})$-time lookup are known.
- Can we use it?

Open Problems

- Can we close the gap?
  - Our new indexing structure works best for “small” patterns of length $O(\log n)$.
  - On average, “small” patterns are the hardest to find.
  - The case of “large” patterns of length $\Omega(\log^k n)$ can be handled with the data structure of Cole et. al 2004.
  - No lower bounds on the index size for $O(m + \text{occ})$-time lookup are known.
- Can we use it?
  - Constant factors are large!
  - Implementation by (suffix) arrays seems possible and could be more space efficient.
  - Is compression applicable?
  - Construction time?

Open Problems

- Can we close the gap?
  - Our new indexing structure works best for "small" patterns of length $O(\log n)$.
  - On average, "small" patterns are the hardest to find.
  - The case of "large" patterns of length $\Omega(\log^k n)$ can be handled with the data structure of Cole et. al 2004.
  - No lower bounds on the index size for $O(m + \mathrm{occ})$-time lookup are known.
- Can we use it?
  - Constant factors are large!
  - Implementation by (suffix) arrays seems possible and could be more space efficient.
  - Is compression applicable?
  - Construction time?

Open Problems

- Can we close the gap?
  - Our new indexing structure works best for “small” patterns of length $O(\log n)$.
  - On average, “small” patterns are the hardest to find.
  - The case of “large” patterns of length $\Omega(\log^k n)$ can be handled with the data structure of Cole et. al 2004.
  - No lower bounds on the index size for $O(m + \mathrm{occ})$-time lookup are known.
- Can we use it?
  - Constant factors are large!
  - Implementation by (suffix) arrays seems possible and could be more space efficient.
  - Is compression applicable?
  - Construction time?

Open Problems

- Can we close the gap?
  - Our new indexing structure works best for "small" patterns of length $O(\log n)$.
  - On average, "small" patterns are the hardest to find.
  - The case of "large" patterns of length $\Omega(\log^k n)$ can be handled with the data structure of Cole et. al 2004.
  - No lower bounds on the index size for $O(m + \text{occ})$-time lookup are known.
- Can we use it?
  - Constant factors are large!
  - Implementation by (suffix) arrays seems possible and could be more space efficient.
  - Is compression applicable?
  - Construction time?

Open Problems

- Can we close the gap?
  - Our new indexing structure works best for “small” patterns of length $O(\log n)$.
  - On average, “small” patterns are the hardest to find.
  - The case of “large” patterns of length $\Omega(\log^k n)$ can be handled with the data structure of Cole et. al 2004.
  - No lower bounds on the index size for $O(m + \text{occ})$-time lookup are known.
- Can we use it?
  - Constant factors are large!
  - Implementation by (suffix) arrays seems possible and could be more space efficient.
  - Is compression applicable?
  - Construction time?

Open Problems

- Can we close the gap?
  - Our new indexing structure works best for “small” patterns of length $O(\log n)$.
  - On average, “small” patterns are the hardest to find.
  - The case of “large” patterns of length $\Omega(\log^k n)$ can be handled with the data structure of Cole et. al 2004.
  - No lower bounds on the index size for $O(m + \text{occ})$-time lookup are known.
- Can we use it?
  - Constant factors are large!
  - Implementation by (suffix) arrays seems possible and could be more space efficient.
  - Is compression applicable?
  - Construction time?

Open Problems

- Can we close the gap?
  - Our new indexing structure works best for "small" patterns of length $O(\log n)$.
  - On average, "small" patterns are the hardest to find.
  - The case of "large" patterns of length $\Omega(\log^k n)$ can be handled with the data structure of Cole et. al 2004.
  - No lower bounds on the index size for $O(m + \text{occ})$-time lookup are known.
- Can we use it?
  - Constant factors are large!
  - Implementation by (suffix) arrays seems possible and could be more space efficient.
  - Is compression applicable?
  - Construction time?

Thank you!

Questions?

Text Indexing with Errors

Moritz G. Maaß and Johannes Nowak

{maass,nowakj}@in.tum.de
Institut für Informatik
Technische Universität München

June 20, 2005

Example

```
GACTCAAAACGGGTTGTTACCGGGTATGGCTAGAATCATC
CGTACTGCGTGACCGACGGATGACGAATAAAGGAGTTAAC
TTGAGGGCGGCGAGCGACCTACAAACATGTTCGGGAAACT
CTGTCCAAAATAAACCTGAGACCAACCGTTTAGCAAGAAG
```

Pattern : ACAAC

Example - Occurrences

GACTCAAAACGGGTTGTTACCGGGTATGGCTAGAATCATC : (6,9)
CGTACTGCGTGACCGACGGATGACGAATAAAGGAGTTAAC
TTGAGGGCGGCGAGCGACCTACAAACATGTTCGGGAAACT : (20,23),(20,24),(22,25),(35,38)
CTGTCCAAAATAAACCTGAGACCAACCGTTTAGCAAGAAG : (11,14),(22,25)

Pattern : ACAAC

Example - Positions

GACTCAAAACGGGTTGTTACCGGGTATGGCTAGAATCATC : 6
CGTACTGCGTGACCGACGGATGACGAATAAAGGAGTTAAC
TTGAGGGCGGCGAGCGACCTACAAACATGTTCGGGAAACT : 20,22,35
CTGTCCAAAATAAACCTGAGACCAACCGTTTAGCAAGAAG : 11,22

Pattern : ACAAC

Example - Documents

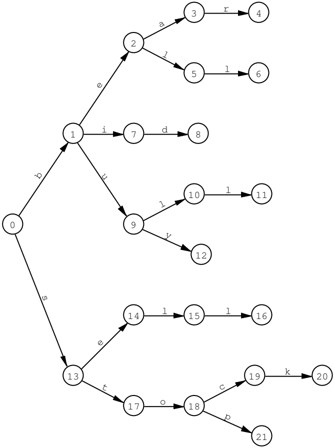
GACTCAAAACGGGTTGTTACCGGGTATGGCTAGAATCATC
CGTACTGCGTGACCGACGGATGACGAATAAAGGAGTTAAC
TTGAGGGCGGCGAGCGACCTACAAACATGTTCGGGAAACT
CTGTCCAAAATAAACCTGAGACCAACCGTTTAGCAAGAAG

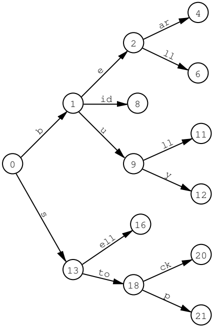
Pattern : ACAAC

Tries and PATRICIA Trees

String set:

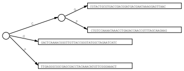
```
bear
bell
bid
bull
buy
sell
stock
stop
```



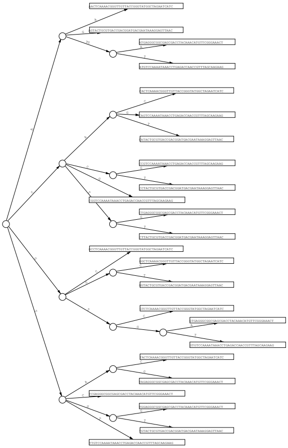


Example

GACTCAAAACGGGTTGTTACCGGGTATGGCTAGAATCATC
CGTACTGCGTGACCGACGGATGACGAATAAAGGAGTTAAC
TTGAGGGCGGCGAGCGACCTACAAACATGTTCGGGAAACT
CTGTCCAAAATAAACCTGAGACCAACCGTTTAGCAAGAAG



AACTCAAAACGGGTTGTTACCGGGTATGGCTAGAATCATC
CACTCAAAACGGGTTGTTACCGGGTATGGCTAGAATCATC
TACTCAAAACGGGTTGTTACCGGGTATGGCTAGAATCATC
GCCTCAAAACGGGTTGTTACCGGGTATGGCTAGAATCATC
GGCTCAAAACGGGTTGTTACCGGGTATGGCTAGAATCATC
GTCTCAAAACGGGTTGTTACCGGGTATGGCTAGAATCATC
AGTACTGCGTGACCGACGGATGACGAATAAAGGAGTTAAC
GGTACTGCGTGACCGACGGATGACGAATAAAGGAGTTAAC
TGTACTGCGTGACCGACGGATGACGAATAAAGGAGTTAAC
CATACTGCGTGACCGACGGATGACGAATAAAGGAGTTAAC
CCTACTGCGTGACCGACGGATGACGAATAAAGGAGTTAAC
CTTACTGCGTGACCGACGGATGACGAATAAAGGAGTTAAC
ATGAGGGCGGCGAGCGACCTACAAACATGTTCGGGAAACT
CTGAGGGCGGCGAGCGACCTACAAACATGTTCGGGAAACT
GTGAGGGCGGCGAGCGACCTACAAACATGTTCGGGAAACT
TAGAGGGCGGCGAGCGACCTACAAACATGTTCGGGAAACT
TCGAGGGCGGCGAGCGACCTACAAACATGTTCGGGAAACT
TGGAGGGCGGCGAGCGACCTACAAACATGTTCGGGAAACT
ATGTCCAAAATAAACCTGAGACCAACCGTTTAGCAAGAAG
GTGTCCAAAATAAACCTGAGACCAACCGTTTAGCAAGAAG
TTGTCCAAAATAAACCTGAGACCAACCGTTTAGCAAGAAG
CAGTCCAAAATAAACCTGAGACCAACCGTTTAGCAAGAAG
CCGTCCAAAATAAACCTGAGACCAACCGTTTAGCAAGAAG
CGGTCCAAAATAAACCTGAGACCAACCGTTTAGCAAGAAG



Edit Distance

- Dynamic programming version:

$$D_{i,j} = \min\{D_{i-1,j} + 1, D_{i-1,j-1} + \delta(i,j), D_{i,j-1} + 1\},$$

for all $1 \leq i \leq |u|$, $1 \leq j \leq |v|$.

$$\delta(i,j) = \begin{cases} 1 & \text{if } u[i] \neq v[j] \\ 0 & \text{if } u[i] = v[j] \end{cases}$$

- As operator $op : \Sigma^* \to \Sigma^*$, i.e.,

$$\begin{aligned} op_{del,2}(\mathtt{ACAAC}) &= \mathtt{AAAC} \\ op_{sub,2,\mathtt{A}}(\mathtt{ACAAC}) &= \mathtt{ACAAA} \end{aligned}$$