Jumbled String Matching: Online, offline, binary or not

Zsuzsanna Lipták

University of Verona (Italy)

McMaster University, 10 Oct. 2014

Jumbled String Matching

Parikh vectors:

Given string t over constant-size ordered alphabet Σ, with $|\Sigma| = \sigma$.
The Parikh vector $p(t)$ counts the multiplicity of characters in t.
Ex.: $p(aabcac) = (3, 1, 2)$.

Jumbled String Matching

Parikh vectors:

Given string t over constant-size ordered alphabet Σ, with $|\Sigma| = \sigma$.
The Parikh vector $p(t)$ counts the multiplicity of characters in t.
Ex.: $p(aabcac) = (3, 1, 2)$.

(a.k.a. jumbled string = compomer = composition = multiplicity vector = statistics = histogram . . . . . .)

Problem Statement

JUMBLED STRING MATCHING

Given string s of length n, and query Parikh vector $q \in \mathbb{N}^\sigma$.
Find all occurrences of substrings t of s s.t. $p(t) = q$.

Ex.: $\Sigma = \{a, b, c\}$, query $q = (3, 1, 2)$

b b a c a c c a b a b b a b c c a a a c

Problem Statement

JUMBLED STRING MATCHING

Given string s of length n, and query Parikh vector $q \in \mathbb{N}^\sigma$.
Find all occurrences of substrings t of s s.t. $p(t) = q$.

Ex.: $\Sigma = \{a, b, c\}$, query $q = (3, 1, 2)$

b b a c $\boxed{\textbf{\textit{a c c a b a}}}$ ***b b*** $\boxed{\textbf{\textit{a b c c a a}}}$ ***a c***

$=$ any permutation of *aabcac*

Problem Statement

JUMBLED STRING MATCHING

Given string s of length n, and query Parikh vector $q \in \mathbb{N}^\sigma$.
Find all occurrences of substrings t of s s.t. $p(t) = q$.

Ex.: $\Sigma = \{a, b, c\}$, query $q = (3, 1, 2)$

b b a c a c c a b a b b a b c c a a a c

$=$ any permutation of *aabcac* $=$ any jumble of *aabcac*

Problem Statement

JUMBLED STRING MATCHING

Given string s of length n, and query Parikh vector $q \in \mathbb{N}^\sigma$.
Find all occurrences of substrings t of s s.t. $p(t) = q$.

Ex.: $\Sigma = \{a, b, c\}$, query $q = (3, 1, 2)$

b b a c a c c a b a b b a b c c a a a c

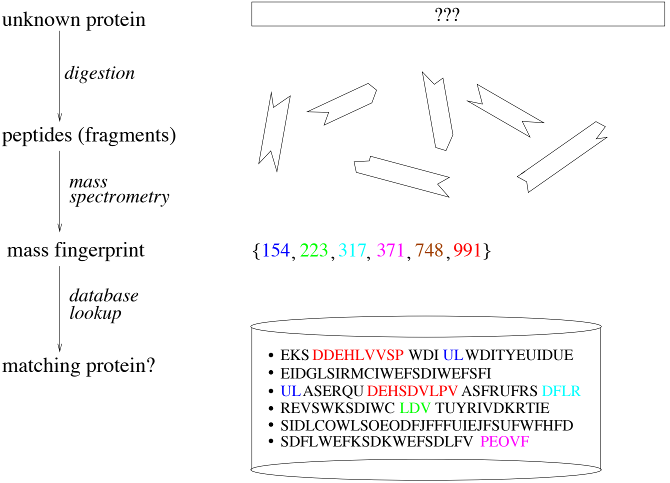
$=$ any permutation of *aabcac* $=$ any jumble of *aabcac*

a.k.a. permutation matching, Parikh vector matching, abelian matching, JPM $=$ jumbled pattern matching, jumbled indexing - see later

Motivations, Applications

- Mass spectrometry
- Gene clusters
- Motif search in graphs and trees
- Filter for exact pattern matching

Protein identification with mass spectrometry (here: PMF)



Takes advantage of different molecular masses of “characters” (AAs, nucleotides, . . .)

Modelling sample identification with MS

Every character has a mass: $\mu : \Sigma \to \mathbb{R}^+$ mass function, $\mu(t) = \sum_i \mu(t_i)$.

Ex: $\Sigma = \{a, b, c\}$ with $\mu(a) = 2, \mu(b) = 3, \mu(c) = 5$. Query $M = 19$

b b a c a c c a b a b b a b c c a a a c

Actually we can also look for all Parikh vectors with query mass M!
$\Longrightarrow$ Jumbled String Matching!

Application 2: Gene clusters

Given: k genomes

Find: maximal blocks consisting of same genes

$$1\ \ 2\ \ \boxed{3\ \ 4\ \ 3\ \ 5\ \ 1\ \ 5\ \ 4}\ \ 7\ \ 2\ \ 5$$

$$\boxed{5\ \ 3\ \ 1\ \ 4\ \ 4}\ \ 7\ \ 1\ \ 7\ \ 2\ \ 2\ \ 1\ \ 3$$

$$1\ \ 2\ \ 6\ \ 7\ \ \boxed{5\ \ 1\ \ 3\ \ 5\ \ 4}\ \ 2\ \ 2\ \ 5$$

gene cluster: $\{1, 3, 4, 5\}$

Application 2: Gene clusters

Given: k genomes
Find: maximal blocks consisting of same genes

$$1\ 2\ \boxed{3\ 4\ 3\ 5\ 1\ 5\ 4}\ 7\ 2\ 5$$

$$\boxed{5\ 3\ 1\ 4\ 4}\ 7\ 1\ 7\ 2\ 2\ 1\ 3$$

$$1\ 2\ 6\ 7\ \boxed{5\ 1\ 3\ 5\ 4}\ 2\ 2\ 5$$

gene cluster: $\{1, 3, 4, 5\}$

Caveat: Problem slightly different (so far).

1. Simple solutions

Window algorithm

Jumbled string matching query $q = (3, 1, 2)$

$\boxed{b\ b\ a\ c\ a\ c}\ c\ a\ b\ a\ b\ b\ a\ b\ c\ c\ a\ a\ a\ c$

- sliding window (of fixed-size $m = |q| = \sum_i q_i$)

Window algorithm

Jumbled string matching query $q = (3, 1, 2)$

$b \; \boxed{b \; a \; c \; a \; c \; c} \; a \; b \; a \; b \; b \; a \; b \; c \; c \; a \; a \; a \; c$

- sliding window (of fixed-size $m = |q| = \sum_i q_i$)

Window algorithm

Jumbled string matching query $q = (3, 1, 2)$

$$\boldsymbol{b \; b \; \boxed{a \; c \; a \; c \; c \; a} \; b \; a \; b \; b \; a \; b \; c \; c \; a \; a \; a \; c}$$

- sliding window (of fixed-size $m = |q| = \sum_i q_i$)

Window algorithm

Jumbled string matching query $q = (3, 1, 2)$

$$\boldsymbol{b\ b\ a\ \boxed{c\ a\ c\ c\ a\ b}\ a\ b\ b\ a\ b\ c\ c\ a\ a\ a\ c}$$

- sliding window (of fixed-size $m = |q| = \sum_i q_i$)

Window algorithm

Jumbled string matching query $q = (3, 1, 2)$

_b b a c a c c a b a b b a b c c a a a c_

- sliding window (of fixed-size $m = |q| = \sum_i q_i$)

Window algorithm

Jumbled string matching query $q = (3, 1, 2)$

b b a c $\boxed{\textit{a c c a b a}}$ **b b a b c c a a a c**

- sliding window (of fixed-size $m = |q| = \sum_i q_i$)
- worst-case optimal for one query: $O(n)$ time, $O(\sigma)$ additional space.

Indexed jumbled string matching

What about many queries? $\rightsquigarrow$ indexed version of problem

simple solutions (K queries):

1. no index: $O(Kn)$ query time
2. store all: $O(n^2)$ index size, $O(K \log n)$ query time.

Indexed jumbled string matching

What about many queries? $\rightsquigarrow$ indexed version of problem

simple solutions (K queries):

1. no index: $O(Kn)$ query time
2. store all: $O(n^2)$ index size, $O(K \log n)$ query time.

For exact string matching, elaborate solutions exist:
suffix trees, suffix arrays, ...

- index has $O(n)$ size (size of text)
- construction time and space $O(n)$
- query time $O(m)$ (size of query) – so $O(Km)$ for K queries

2. General alphabets: Jumping Algorithm

Jumping Algorithm for Jumbled String Matching

Cicalese, Fici, L. (PSC 2009, FUN 2010)

Recall the window algorithm.
fixed size window $\rightarrow$ variable size window

$q = (312)$

$\boxed{b\ b\ a\ c\ a\ c}\ c\ a\ b\ a\ b\ b\ a\ b\ c\ c\ a\ a\ a\ c$

Jumping Algorithm for Jumbled String Matching

Cicalese, Fici, L. (PSC 2009, FUN 2010)

Recall the window algorithm.
fixed size window $\rightarrow$ variable size window

$q = (312)$

$\boxed{b\ b\ a\ c\ a\ c\ c}\ a\ b\ a\ b\ b\ a\ b\ c\ c\ a\ a\ a\ c$

Jumping Algorithm for Jumbled String Matching

Cicalese, Fici, L. (PSC 2009, FUN 2010)

Recall the window algorithm.
fixed size window $\rightarrow$ variable size window

$q = (312)$

$\boxed{b\ b\ a\ c\ a\ c\ c\ a}\ b\ a\ b\ b\ a\ b\ c\ c\ a\ a\ a\ c$

Jumping Algorithm for Jumbled String Matching

Cicalese, Fici, L. (PSC 2009, FUN 2010)

Recall the window algorithm.
fixed size window → variable size window

$q = (312)$

$b \ \boxed{b \ a \ c \ a \ c \ c \ a} \ b \ a \ b \ b \ a \ b \ c \ c \ a \ a \ a \ c$

Jumping Algorithm for Jumbled String Matching

Cicalese, Fici, L. (PSC 2009, FUN 2010)

Recall the window algorithm.
fixed size window $\rightarrow$ variable size window

$q = (312)$

$\boldsymbol{b}\ \boldsymbol{b}\ \boxed{\boldsymbol{a}\ \boldsymbol{c}\ \boldsymbol{a}\ \boldsymbol{c}\ \boldsymbol{c}\ \boldsymbol{a}}\ \boldsymbol{b}\ \boldsymbol{a}\ \boldsymbol{b}\ \boldsymbol{b}\ \boldsymbol{a}\ \boldsymbol{b}\ \boldsymbol{c}\ \boldsymbol{c}\ \boldsymbol{a}\ \boldsymbol{a}\ \boldsymbol{a}\ \boldsymbol{c}$

Jumping Algorithm for Jumbled String Matching

Cicalese, Fici, L. (PSC 2009, FUN 2010)

Recall the window algorithm.
fixed size window $\rightarrow$ variable size window

$q = (312)$

$$\boldsymbol{b\ b\ a\ \boxed{c\ a\ c\ c\ a}\ b\ a\ b\ b\ a\ b\ c\ c\ a\ a\ a\ c}$$

Jumping Algorithm for Jumbled String Matching

Cicalese, Fici, L. (PSC 2009, FUN 2010)

Recall the window algorithm.
fixed size window $\rightarrow$ variable size window

$q = (312)$

b b a c $\boxed{\textbf{\textit{a c c a}}}$ ***b a b b a b c c a a a c***

Jumping Algorithm for Jumbled String Matching

Cicalese, Fici, L. (PSC 2009, FUN 2010)

Recall the window algorithm.
fixed size window $\rightarrow$ variable size window

$q = (312)$

$$b\ b\ a\ c\ \boxed{a\ c\ c\ a}\ b\ a\ b\ b\ a\ b\ c\ c\ a\ a\ a\ c$$

The Jumping Algorithm simulates these moves by jumps.

Jumping algorithm: update rules

$q = (312)$

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
	L	*b*	*b*	*a*	*c*	*a*	*c*	*c*	*a*	*b*	*a*	*b*	*b*	*a*	*b*	*c*	*c*	*a*	*a*	*a*	*c*
a		0	0	1	1	2	2	2	3	3	4	4	4	5	5	5	5	6	7	8	8
b		1	2	2	2	2	2	2	2	3	3	4	5	5	6	6	6	6	6	6	6
c		0	0	0	1	1	2	3	3	3	3	3	3	3	3	4	5	5	5	5	6

Jumping algorithm: update rules

$q = (312)$

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
	L								***R***												
		b	***b***	***a***	***c***	***a***	***c***	***c***	***a***	***b***	***a***	***b***	***b***	***a***	***b***	***c***	***c***	***a***	***a***	***a***	***c***
a		0	0	1	1	2	2	2	3	3	4	4	4	5	5	5	5	6	7	8	8
b		1	2	2	2	2	2	2	2	3	3	4	5	5	6	6	6	6	6	6	6
c		0	0	0	1	1	2	3	3	3	3	3	3	3	3	4	5	5	5	5	6

Jumping algorithm: update rules

$q = (312)$

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
	L								*R*												
		b	***b***	***a***	***c***	***a***	***c***	***c***	***a***	***b***	***a***	***b***	***b***	***a***	***b***	***c***	***c***	***a***	***a***	***a***	***c***
a		0	0	1	1	2	2	2	3	3	4	4	4	5	5	5	5	6	7	8	8
b		1	2	2	2	2	2	2	2	3	3	4	5	5	6	6	6	6	6	6	6
c		0	0	0	1	1	2	3	3	3	3	3	3	3	3	4	5	5	5	5	6

Jumping algorithm: update rules

$q = (312)$

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
	L								*R*												
		b	***b***	***a***	***c***	***a***	***c***	***c***	***a***	***b***	***a***	***b***	***b***	***a***	***b***	***c***	***c***	***a***	***a***	***a***	***c***
a		0	0	1	1	2	2	2	3	3	4	4	4	5	5	5	5	6	7	8	8
b		1	2	2	2	2	2	2	2	3	3	4	5	5	6	6	6	6	6	6	6
c		0	0	0	1	1	2	3	3	3	3	3	3	3	3	4	5	5	5	5	6

update R: $R \leftarrow \max_{a\in\Sigma} select_a \underbrace{(rank_a(L) + q_a)}_{\text{necessary char's}}$

Jumping algorithm: update rules

$q = (312)$

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
					L				***R***												
		b	*b*	*a*	*c*	*a*	*c*	*c*	*a*	*b*	*a*	*b*	*b*	*a*	*b*	*c*	*c*	*a*	*a*	*a*	*c*
a		0	0	1	1	2	2	2	3	3	4	4	4	5	5	5	5	6	7	8	8
b		1	2	2	2	2	2	2	2	3	3	4	5	5	6	6	6	6	6	6	6
c		0	0	0	1	1	2	3	3	3	3	3	3	3	3	4	5	5	5	5	6

update R: $R \leftarrow \max_{a\in\Sigma} select_a \underbrace{\left(rank_a(L) + q_a\right)}_{\text{necessary char's}}$

Jumping algorithm: update rules

$q = (312)$

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
					L				R												
		b	*b*	*a*	*c*	*a*	*c*	*c*	*a*	*b*	*a*	*b*	*b*	*a*	*b*	*c*	*c*	*a*	*a*	*a*	*c*
a		0	0	1	1	2	2	2	3	3	4	4	4	5	5	5	5	6	7	8	8
b		1	2	2	2	2	2	2	2	3	3	4	5	5	6	6	6	6	6	6	6
c		0	0	0	1	1	2	3	3	3	3	3	3	3	3	4	5	5	5	5	6

update R: $R \leftarrow \max_{a\in\Sigma} select_a \underbrace{(rank_a(L) + q_a)}_{\text{necessary char's}}$

update L: $L \leftarrow \max_{a\in\Sigma} select_a \underbrace{(rank_a(R) - q_a)}_{\text{unnecessary char's}}$ (no match),

Jumping algorithm: update rules

$q = (312)$

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
					L						***R***										
		b	***b***	***a***	***c***	***a***	***c***	***c***	***a***	***b***	***a***	***b***	***b***	***a***	***b***	***c***	***c***	***a***	***a***	***a***	***c***
a		0	0	1	1	2	2	2	3	3	4	4	4	5	5	5	5	6	7	8	8
b		1	2	2	2	2	2	2	2	3	3	4	5	5	6	6	6	6	6	6	6
c		0	0	0	1	1	2	3	3	3	3	3	3	3	3	4	5	5	5	5	6

update R: $R \leftarrow \max_{a\in\Sigma} select_a \underbrace{(rank_a(L) + q_a)}_{\text{necessary char's}}$

update L: $L \leftarrow \max_{a\in\Sigma} select_a \underbrace{(rank_a(R) - q_a)}_{\text{unnecessary char's}}$ (no match),

Jumping algorithm: update rules

$q = (312)$

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
						L					R										
		b	*b*	*a*	*c*	*a*	*c*	*c*	*a*	*b*	*a*	*b*	*b*	*a*	*b*	*c*	*c*	*a*	*a*	*a*	*c*
a		0	0	1	1	2	2	2	3	3	4	4	4	5	5	5	5	6	7	8	8
b		1	2	2	2	2	2	2	2	3	3	4	5	5	6	6	6	6	6	6	6
c		0	0	0	1	1	2	3	3	3	3	3	3	3	3	4	5	5	5	5	6

update R: $R \leftarrow \max_{a\in\Sigma} select_a \underbrace{(rank_a(L) + q_a)}_{\text{necessary char's}}$

update L: $L \leftarrow \max_{a\in\Sigma} select_a \underbrace{(rank_a(R) - q_a)}_{\text{unnecessary char's}}$ (no match),

. $L \leftarrow L + 1$ (match).

Jumping algorithm: update rules

$q = (312)$

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
						L								R							
		b	*b*	*a*	*c*	*a*	*c*	*c*	*a*	*b*	*a*	*b*	*b*	*a*	*b*	*c*	*c*	*a*	*a*	*a*	*c*
a		0	0	1	1	2	2	2	3	3	4	4	4	5	5	5	5	6	7	8	8
b		1	2	2	2	2	2	2	2	3	3	4	5	5	6	6	6	6	6	6	6
c		0	0	0	1	1	2	3	3	3	3	3	3	3	3	4	5	5	5	5	6

update R: $R \leftarrow \max_{a \in \Sigma} select_a \underbrace{(rank_a(L) + q_a)}_{\text{necessary char's}}$

update L: $L \leftarrow \max_{a \in \Sigma} select_a \underbrace{(rank_a(R) - q_a)}_{\text{unnecessary char's}}$ (no match),

. $L \leftarrow L + 1$ (match).

Jumping algorithm: update rules

$q = (312)$

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
												L		R							
		b	***b***	***a***	***c***	***a***	***c***	***c***	***a***	***b***	***a***	***b***	***b***	***a***	***b***	***c***	***c***	***a***	***a***	***a***	***c***
a		0	0	1	1	2	2	2	3	3	4	4	4	5	5	5	5	6	7	8	8
b		1	2	2	2	2	2	2	2	3	3	4	5	5	6	6	6	6	6	6	6
c		0	0	0	1	1	2	3	3	3	3	3	3	3	3	4	5	5	5	5	6

update R: $R \leftarrow \max_{a\in\Sigma} select_a \underbrace{(rank_a(L) + q_a)}_{\text{necessary char's}}$

update L: $L \leftarrow \max_{a\in\Sigma} select_a \underbrace{(rank_a(R) - q_a)}_{\text{unnecessary char's}}$ (no match),

. $L \leftarrow L + 1$ (match).

Jumping algorithm: update rules

$q = (312)$

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
												L							R		
		b	b	a	c	a	c	c	a	b	a	b	b	a	b	c	c	a	a	a	c
a		0	0	1	1	2	2	2	3	3	4	4	4	5	5	5	5	6	7	8	8
b		1	2	2	2	2	2	2	2	3	3	4	5	5	6	6	6	6	6	6	6
c		0	0	0	1	1	2	3	3	3	3	3	3	3	3	4	5	5	5	5	6

update R: $R \leftarrow \max_{a\in\Sigma} select_a \underbrace{(rank_a(L) + q_a)}_{\text{necessary char's}}$

update L: $L \leftarrow \max_{a\in\Sigma} select_a \underbrace{(rank_a(R) - q_a)}_{\text{unnecessary char's}}$ (no match),

. $L \leftarrow L + 1$ (match).

Jumping algorithm: update rules

$q = (312)$

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
													L						*R*		
		b	***b***	***a***	***c***	***a***	***c***	***c***	***a***	***b***	***a***	***b***	***b***	***a***	***b***	***c***	***c***	***a***	***a***	***a***	***c***
a		0	0	1	1	2	2	2	3	3	4	4	4	5	5	5	5	6	7	8	8
b		1	2	2	2	2	2	2	2	3	3	4	5	5	6	6	6	6	6	6	6
c		0	0	0	1	1	2	3	3	3	3	3	3	3	3	4	5	5	5	5	6

update R: $R \leftarrow \max_{a\in\Sigma} select_a \underbrace{(rank_a(L) + q_a)}_{\text{necessary char's}}$

update L: $L \leftarrow \max_{a\in\Sigma} select_a \underbrace{(rank_a(R) - q_a)}_{\text{unnecessary char's}}$ (no match),

. $L \leftarrow L + 1$ (match).

Jumping algorithm: update rules

$q = (312)$

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
														L					R		
		b	***b***	***a***	***c***	***a***	***c***	***c***	***a***	***b***	***a***	***b***	***b***	***a***	***b***	***c***	***c***	***a***	***a***	***a***	***c***
a		0	0	1	1	2	2	2	3	3	4	4	4	5	5	5	5	6	7	8	8
b		1	2	2	2	2	2	2	2	3	3	4	5	5	6	6	6	6	6	6	6
c		0	0	0	1	1	2	3	3	3	3	3	3	3	3	4	5	5	5	5	6

update R: $R \leftarrow \max_{a\in\Sigma} select_a \underbrace{(rank_a(L) + q_a)}_{\text{necessary char's}}$

update L: $L \leftarrow \max_{a\in\Sigma} select_a \underbrace{(rank_a(R) - q_a)}_{\text{unnecessary char's}}$ (no match),

. $L \leftarrow L + 1$ (match).

Jumping algorithm: update rules

$q = (312)$

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
														L						R	
		b	***b***	***a***	***c***	***a***	***c***	***c***	***a***	***b***	***a***	***b***	***b***	***a***	***b***	***c***	***c***	***a***	***a***	***a***	***c***
a		0	0	1	1	2	2	2	3	3	4	4	4	5	5	5	5	6	7	8	8
b		1	2	2	2	2	2	2	2	3	3	4	5	5	6	6	6	6	6	6	6
c		0	0	0	1	1	2	3	3	3	3	3	3	3	3	4	5	5	5	5	6

update R: $R \leftarrow \max_{a\in\Sigma} select_a \underbrace{(rank_a(L) + q_a)}_{\text{necessary char's}}$

update L: $L \leftarrow \max_{a\in\Sigma} select_a \underbrace{(rank_a(R) - q_a)}_{\text{unnecessary char's}}$ (no match),

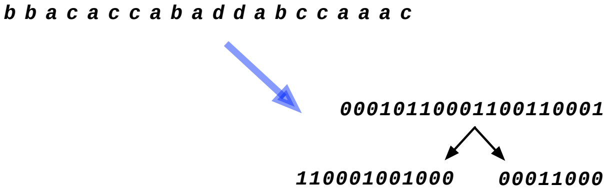
. $L \leftarrow L + 1$ (match).

Jumping algorithm: Analysis

Note that we do not need to store the prefix table, nor the string s.

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
						L								*R*							
		b	*b*	*a*	*c*	*a*	*c*	*c*	*a*	*b*	*a*	*b*	*b*	*a*	*b*	*c*	*c*	*a*	*a*	*a*	*c*
a		0	0	1	1	2	2	2	3	3	4	4	4	5	5	5	5	6	7	8	8
b		1	2	2	2	2	2	2	2	3	3	4	5	5	6	6	6	6	6	6	6
c		0	0	0	1	1	2	3	3	3	3	3	3	3	3	4	5	5	5	5	6

Jumping algorithm: Analysis



Using a wavelet tree [Grossi et al., SODA 2003] as index, we have

- space $O(n)$ (for wavelet tree)
- construction time $O(n \log \sigma)$
- every update (jump) in $O(\sigma)$ time
- query time $O(J\sigma)$, where $J =$ number of jumps (updates)
- expected running time: $O(n\sqrt{\frac{\sigma}{\log \sigma}\frac{1}{\sqrt{m}}})$, where $m = \sum_i q_i$.

Jumbled string matching on general alphabets

Current best result: Kociumaka, Radoszewski, Rytter (ESA 2013)

- for arbitrary constant size alphabet
- $o(n^2)$ index size
- $o(n)$ query time (worst-case)
- $O(n^2)$ construction time

(for any $\delta \in (0,1)$ construct index of size $O(n^{2-\delta})$ with query time $O(m^{\delta(2\sigma-1)})$)

Jumbled string matching on general alphabets

Current best result: Kociumaka, Radoszewski, Rytter (ESA 2013)

- for arbitrary constant size alphabet
- $o(n^2)$ index size
- $o(n)$ query time (worst-case)
- $O(n^2)$ construction time

(for any $\delta \in (0, 1)$ construct index of size $O(n^{2-\delta})$ with query time $O(m^{\delta(2\sigma-1)})$)

Compare to:

1. no index (online): $O(n)$ space, $O(n)$ query time
2. store all: $O(n^2)$ index size, $O(\log n)$ query time
3. Jumping algo: $O(n)$ index size, $o(n)$ query time in expectation

Jumbled string matching on general alphabets

Open problem was: Find something better.

But:
Amir, Chan, Lewenstein, Lewenstein (ICALP 2014) showed that the indexing problem is 3-sum-hard for $\sigma \geq 3$

More precisely, under different versions of 3-sum-hardness assumption,

- for non-constant σ, either preprocessing time is $\Omega(n^{2-\epsilon})$, or query time is $\Omega(n^{1-\delta})$, for all ϵ, δ
- for $\sigma \geq 3$, same except ϵ, δ depend on σ

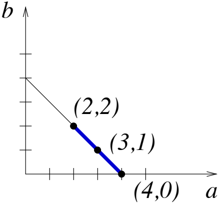
Authors also promised algorithms matching the bounds (forthcoming?)

3. Binary alphabets

Binary alphabets: Interval property

Lemma

If $(x, m - x)$ *and* $(y, m - y)$ *both occur in* s*, then so does* $(z, m - z)$ *for any* $x \leq z \leq y$*.*

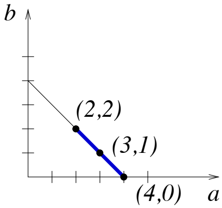


`b a b a` `a b a a b b` `a a a a` `b a`

`amin(4) = 2` `amax(4) = 4`

Lemma

If $(x, m - x)$ and $(y, m - y)$ both occur in s, then so does $(z, m - z)$ for any $x \leq z \leq y$.



$\boxed{\texttt{b a b a}}$ **a b a a b b** $\boxed{\texttt{a a a a}}$ **b a**

amin(4) = 2 **amax(4) = 4**

Corollary

All sub-Parikh vectors of s of length m build a set
$\{(x, m - x) : amin(m) \leq x \leq amax(m)\}$.

Binary alphabets: Interval algorithm for decision queries

- Index: Table of amin(m) and amax(m), for $1 \leq m \leq n$
  size $O(n)$
- Query (x, y) occurs in s iff $\text{amin}(x + y) \leq x \leq \text{amax}(x + y)$.
- Query time $O(1)$.

$$\mathbf{s = ababbaabaabbbaaabbab}$$

m	amin	amax
...	...	...
3	0	3
4	1	3
5	2	4
6	2	4
...	...	...

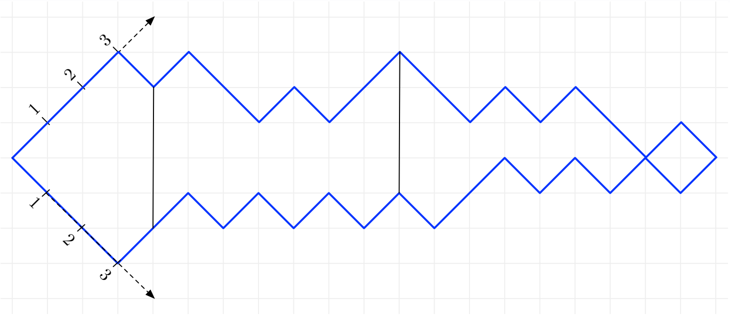
query $(3, 2)$ — YES
query $(1, 4)$ — NO

Construction of index

Goal

Given a binary string of length n, find, for all $1 \leq m \leq n$, the minimum (maximum) number of a's in a window of size m.

- $O(n^2)$ time—Cicalese, Fici, L. (PSC 2009)
- $O(n^2/\log n)$ time
  —Burcsi, Cicalese, Fici, L. (FUN 2010); Moosa, Rahman (IPL 2010)
- $O(n^2/\log^2 n)$ time in word-RAM model
  —Moosa, Rahman (JDA 2012)
- approximate index with one-sided error in $O(n^{1+\epsilon})$ time
  —Cicalese, Laber, Weimann, Yuster (CPM 2012)
- Corner Index: construction time and index size depend on $r =$ runlength enc. of s
  —Badkobeh, Fici, Kroon, L. (IPL 2013); Giaquinta, Grabowski (IPL 2013)
- $n^2/2^{\Omega(\log n/\log\log n)^{1/2}}$—Hermelin, Landau, Rabinovich, Weimann (unpubl.)



m	amin	amax
...	...	...
3	0	3
4	1	3
5	2	4
6	2	4
...	...	...

The Parikh set of $s = ababbaabaabbbaaabbab$. Verticals are fixed length sub-Parikh vectors, for $m = 4, 11$

Construction of index

Goal

Given a binary string of length n, find, for all $1 \leq m \leq n$, the minimum (maximum) number of a's in a window of size m.

- $O(n^2)$ time—Cicalese, Fici, L. (PSC 2009)
- $O(n^2/\log n)$ time
  —Burcsi, Cicalese, Fici, L. (FUN 2010); Moosa, Rahman (IPL 2010)
- $O(n^2/\log^2 n)$ time in word-RAM model
  —Moosa, Rahman (JDA 2012)
- approximate index with one-sided error in $O(n^{1+\epsilon})$ time
  —Cicalese, Laber, Weimann, Yuster (CPM 2012)
- Corner Index: construction time and index size depend on $r =$ runlength enc. of s
  —Badkobeh, Fici, Kroon, L. (IPL 2013); Giaquinta, Grabowski (IPL 2013)
- $n^2/2^{\Omega(\log n/\log\log n)^{1/2}}$—Hermelin, Landau, Rabinovich, Weimann (unpubl.)

Construction of index

Open problem: Faster construction of binary index?

4. Prefix normal words

Prefix normal words

Fici, L. (DLT 2011)

Definition

A word $s \in \{a, b\}^*$ is a prefix normal word (w.r.t. a) if $\forall\ 0 \leq m \leq |s|$ no substring of length m has more a's than the prefix of s of length m.

Example

$$s = ababbaabaabbbaaabbab$$

$$s' = aaababbabaabbababbab$$

Prefix normal words

Fici, L. (DLT 2011)

Definition

A word $s \in \{a, b\}^*$ is a prefix normal word (w.r.t. a) if $\forall\ 0 \leq m \leq |s|$ no substring of length m has more a's than the prefix of s of length m.

Example

$$s = ababbaabaabbbaaabbab \quad NO$$

$$s' = aaababbabaabbababbab \quad YES$$

Prefix normal forms

Recall $amax_a(s, m)$ = maximum number of a's in a substring of s of length m.

$$s = ababbaabaabbbaaabbab$$

m	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
$amax_a$	0	1	2	3	3	4	4	4	5	5	6	7	7	7	8	8	9	9	9	10	10

Theorem

Let $s \in \{a, b\}^*$. Then there exists a unique prefix normal word s' s.t. for all $0 \leq m \leq |s|$, $amax_a(s, m) = amax_a(s', m)$, called its prefix normal form w.r.t. a, $PNF_a(s)$.

Prefix normal forms

$$s = ababbaabaabbbaaabbab$$

m	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
$amax_a$	0	1	2	3	3	4	4	4	5	5	6	7	7	7	8	8	9	9	9	10	10

Prefix normal forms

$$s = ababbaabaabbbaaabbab$$

m	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
$amax_a$	0	1	2	3	3	4	4	4	5	5	6	7	7	7	8	8	9	9	9	10	10

Define the word s' by

$$s'_m = \begin{cases} a & \text{if } amax_a(s,m) = 1 + amax_a(s,m-1) \\ b & \text{if } amax_a(s,m) = amax_a(s,m-1) \end{cases}$$

Prefix normal forms

$$s = ababbaabaabbbaaabbab$$

m	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
$amax_a$	0	1	2	3	3	4	4	4	5	5	6	7	7	7	8	8	9	9	9	10	10

Define the word s' by

$$s'_m = \begin{cases} a & \text{if } amax_a(s,m) = 1 + amax_a(s,m-1) \\ b & \text{if } amax_a(s,m) = amax_a(s,m-1) \end{cases}$$

Prefix normal forms

$$s = ababbaabaabbbaaabbab$$

m	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
$amax_a$	0	1	2	3	3	4	4	4	5	5	6	7	7	7	8	8	9	9	9	10	10

Define the word s' by

$$s'_m = \begin{cases} a & \text{if } amax_a(s,m) = 1 + amax_a(s,m-1) \\ b & \text{if } amax_a(s,m) = amax_a(s,m-1) \end{cases}$$

$$s' = a$$

Prefix normal forms

$$s = ababbaabaabbbaaabbab$$

m	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
$amax_a$	0	1	2	3	3	4	4	4	5	5	6	7	7	7	8	8	9	9	9	10	10

Define the word s' by

$$s'_m = \begin{cases} a & \text{if } amax_a(s,m) = 1 + amax_a(s,m-1) \\ b & \text{if } amax_a(s,m) = amax_a(s,m-1) \end{cases}$$

$$s' = aa$$

Prefix normal forms

$$s = ababbaabaabbbaaabbab$$

m	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
$amax_a$	0	1	2	3	3	4	4	4	5	5	6	7	7	7	8	8	9	9	9	10	10

Define the word s' by

$$s'_m = \begin{cases} a & \text{if } amax_a(s,m) = 1 + amax_a(s,m-1) \\ b & \text{if } amax_a(s,m) = amax_a(s,m-1) \end{cases}$$

$$s' = aaa$$

Prefix normal forms

$$s = ababbaabaabbbaaabbab$$

m	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
$amax_a$	0	1	2	3	3	4	4	4	5	5	6	7	7	7	8	8	9	9	9	10	10

Define the word s' by

$$s'_m = \begin{cases} a & \text{if } amax_a(s,m) = 1 + amax_a(s,m-1) \\ b & \text{if } amax_a(s,m) = amax_a(s,m-1) \end{cases}$$

$$s' = aaab$$

Prefix normal forms

$$s = ababbaabaabbbaaabbab$$

m	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
$amax_a$	0	1	2	3	3	4	4	4	5	5	6	7	7	7	8	8	9	9	9	10	10

Define the word s' by

$$s'_m = \begin{cases} a & \text{if } amax_a(s,m) = 1 + amax_a(s, m-1) \\ b & \text{if } amax_a(s,m) = amax_a(s, m-1) \end{cases}$$

$$s' = aaababbabaabbababbab$$

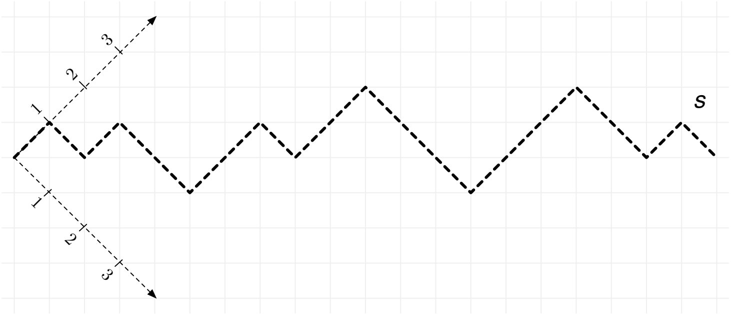
BJPM with prefix normal forms

Theorem

Two strings $s, t \in \{a, b\}^*$ have the same Parikh set if and only if $\mathrm{PNF}_a(s) = \mathrm{PNF}_a(t)$ and $\mathrm{PNF}_b(s) = \mathrm{PNF}_b(t)$.

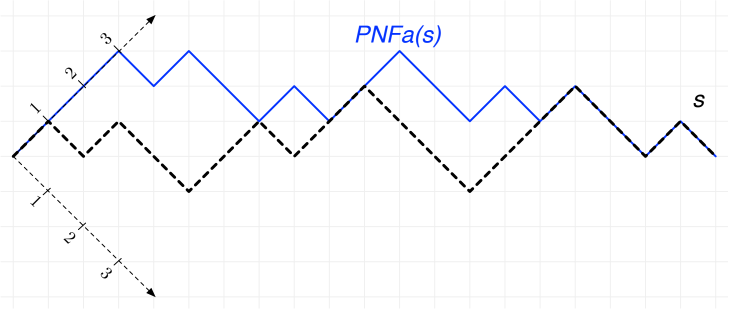
(Parikh set of a word s is the set of Parikh vectors of the substrings of s.
BJPM = binary jumbled pattern matching)

$$s = ababbaabaabbbaaabbab$$



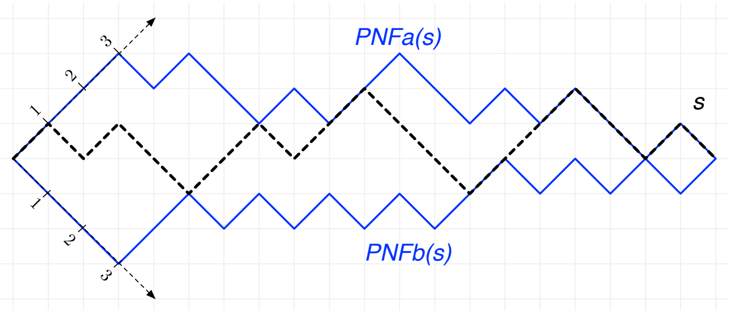
$\diagup = a, \diagdown = b,$

$$s = ababbaabaabbbaaabbab$$



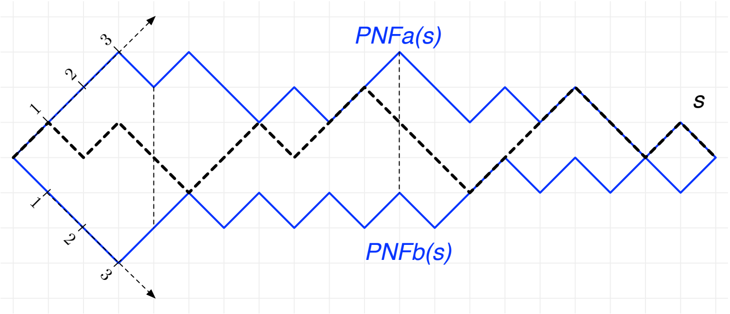
$\diagup = a$, $\diagdown = b$,

$s = ababbaabaabbbaaabbab$



$\diagup = a, \diagdown = b,$

$$s = ababbaabaabbbaaabbab$$



$\diagup = a$, $\diagdown = b$,

BJPM with prefix normal forms

Does $\mathbf{s} = \mathbf{ababbaabaabbbaaabbab}$ have a substring of length 11 containing exactly 5 a's?

$\#$ a's in $\text{pref}(\mathrm{PNF}_a(s), 11) = 7$

$\#$ a's in $\text{pref}(\mathrm{PNF}_b(s), 11) = 5$

$$7 \geq 5 \geq 5 \quad \rightsquigarrow \text{YES}$$

Thus, fast computation of PNFs yields fast solution to BJPM.

BJPM with prefix normal forms

Does $\mathbf{s} = \mathbf{ababbaabaabbbaaabbab}$ have a substring of length 11 containing exactly 5 a's?

$\#\ a$'s in $\text{pref}(\text{PNF}_a(s), 11) = 7$
$\#\ a$'s in $\text{pref}(\text{PNF}_b(s), 11) = 5$

$$7 \geq 5 \geq 5 \quad \rightsquigarrow \text{YES}$$

Thus, fast computation of PNFs yields fast solution to BJPM.
Better understanding p.n. words might be useful for BJPM.

BJPM with prefix normal forms

Does $\mathbf{s} = \mathbf{ababbaabaabbbaaabbab}$ have a substring of length 11 containing exactly 5 a's?

$\#$ a's in $\text{pref}(\mathrm{PNF}_a(s), 11) = 7$
$\#$ a's in $\text{pref}(\mathrm{PNF}_b(s), 11) = 5$

$$7 \geq 5 \geq 5 \quad \rightsquigarrow \text{YES}$$

Thus, fast computation of PNFs yields fast solution to BJPM.
Better understanding p.n. words might be useful for BJPM.
Or might just be fun.

Computation of PNFs

Open problem: Compute the PNFs fast. (Many other open problems on PNFs, e.g. testing.)

Lit:

- Fici, L. (DLT 2011)
- Burcsi, Fici, L., Ruskey, Sawada (CPM 2014): combinatorial generation of all p.n. words, based on them being a bubble language.
- Burcsi, Fici, L., Ruskey, Sawada (FUN 2014): partial results on enumeration of p.n. words
- OEIS seq. no. A194850: no. of p.n. words of length n
- OEIS seq. no. A238109: size of largest equivalence class for length n

Conclusion

We saw: JPM on

1. general alphabets (constant-size)
   - expected: jumping algorithm—sublinear expected time, linear space,
   - worst-case: Kocziumaka et al—just sublinear time and just subquadratic space (tradeoff)
2. binary alphabets (decision queries): linear index, main question is how to compute it
3. prefix normal words / prefix normal forms

Conclusion

We saw: JPM on

1. general alphabets (constant-size)
   - expected: jumping algorithm—sublinear expected time, linear space,
   - worst-case: Kocziumaka et al—just sublinear time and just subquadratic space (tradeoff)
2. binary alphabets (decision queries): linear index, main question is how to compute it
3. prefix normal words / prefix normal forms

Take home message

1. Jumbled pattern matching is interesting.
2. It is definitely hype.
3. Prefix normal words are cool!

AHKNOTUY!

zsuzsanna.liptak@univr.it

APPENDIX

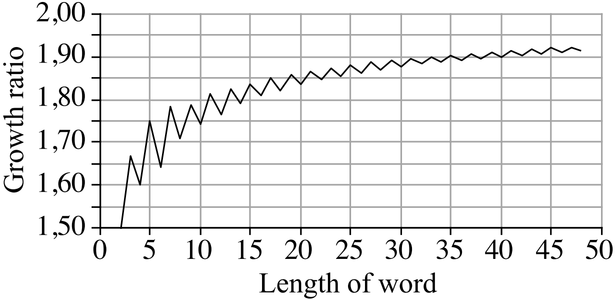
Other variants

- approximate jumbled string matching: e.g. find all occurrences between $(12, 5, 7)$ and $(8, 2, 4)$: variant of jumping algorithm expected sublinear (Burcsi, Cicalese, Fici, L., FUN 2010, ToCS 2012)
- (mostly binary) on trees and graphs with bounded treewidth (Gagie, Hermelin, Landau, Weimann, ESA 2013)
- (mostly binary) locating one occurrence in strings, trees, graphs (Cicalese, Gagie, Giaquinta, Laber, L., Rizzi, Tomescu, SPIRE 2013)
- Jumbled matching on color point sets in the plane (Gagie et al: CPM 2014)
- binary locating queries: included in (Gagie, Hermelin, Landau, Weimann, ESA 2013)

Enumerating p.n. words

OEIS sequence no. A194850

n	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
$pnw(n)$	2	3	5	8	14	23	41	70	125	218	395	697	1273	2279	4185	7568



The prefix normal form PNF_a induces an equivalence relation on Σ^*, namely $u \equiv_{\mathrm{PNF}_a} v$ if and only if $\mathrm{PNF}_a(u) = \mathrm{PNF}_a(v)$.

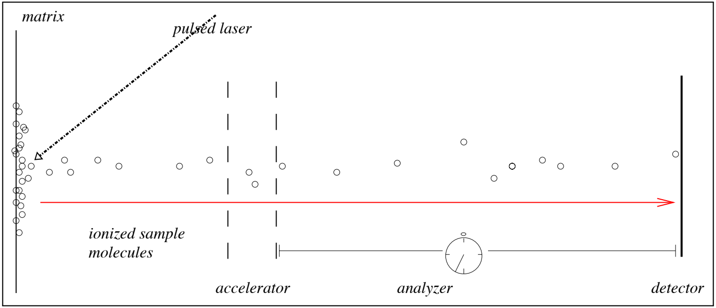
PNF_a	class	card.
$aaaa$	$\{aaaa\}$	1
$aaab$	$\{aaab, baaa\}$	2
$aaba$	$\{aaba, abaa\}$	2
$aabb$	$\{aabb, baab, bbaa\}$	3
$abab$	$\{abab, baba\}$	2
$abba$	$\{abba\}$	1
$abbb$	$\{abbb, babb, bbab, bbba\}$	4
$bbbb$	$\{bbbb\}$	1

Table : The classes of words of length 4 having the same prefix normal form.

PNF_a	card.	PNF_a	card.	PNF_a	card.	PNF_a	card.
aaaaaaaa	1	*aaabaabb*	6	*aabababa*	6	*abababba*	2
aaaaaaab	2	*aaababaa*	2	*aabababb*	9	*abababbb*	4
aaaaaaba	2	*aaababab*	6	*aababbaa*	2	*ababbaba*	1
aaaaaabb	3	*aaababba*	4	*aababbab*	8	*ababbabb*	6
aaaaabaa	2	*aaababbb*	8	*aababbba*	4	*ababbbab*	4
aaaaabab	4	*aaabbaaa*	1	*aababbbb*	10	*ababbbba*	2
aaaaabba	2	*aaabbaab*	4	*aabbaabb*	3	*ababbbbb*	6
aaaaabbb	4	*aaabbaba*	2	*aabbabab*	4	*abbabbab*	2
aaaabaaa	2	*aaabbabb*	6	*aabbabba*	3	*abbabbba*	2
aaaabaab	4	*aaabbbaa*	2	*aabbabbb*	8	*abbabbbb*	5
aaaababa	3	*aaabbbab*	4	*aabbbaab*	2	*abbbabbb*	4
aaaababb	6	*aaabbbba*	2	*aabbbaba*	2	*abbbbabb*	3
aaaabbaa	2	*aaabbbbb*	6	*aabbbabb*	6	*abbbbbab*	2
aaaabbab	4	*aabaabaa*	1	*aabbbbaa*	1	*abbbbbba*	1
aaaabbba	2	*aabaabab*	4	*aabbbbab*	4	*abbbbbbb*	8
aaaabbbb	5	*aabaabba*	2	*aabbbbba*	2	*bbbbbbbb*	1
aaabaaab	2	*aabaabbb*	4	*aabbbbbb*	7		
aaabaaba	4	*aababaab*	2	*abababab*	2		

Table : The cardinalities of the 70 classes of words of length 8 having the same prefix normal form.

What is mass spectrometry?



example: MALDI-TOF mass spectrometer

Reduction to min-plus-convolution

Min-plus-convolution

For real vectors $x = (x_0, \ldots, x_n)$ and $y = (y_0, \ldots, y_n)$, define $x \star y = z$ by

$$z_k = \min_i \{x_i + y_{k-i}\}, \qquad k = 0, 1, \ldots, 2n,$$

(replace *min* by *sum* and $+$ by $\cdot$ to get classic convolution).

Reduction to min-plus-convolution

Min-plus-convolution

For real vectors $x = (x_0, \ldots, x_n)$ and $y = (y_0, \ldots, y_n)$, define $x \star y = z$ by

$$z_k = \min_i \{x_i + y_{k-i}\}, \qquad k = 0, 1, \ldots, 2n,$$

Reduction

Set $x_i := |\mathit{pref}_i(s)|_a$, the number of a's in the prefix of s of length i.
Set $y_i := -x_{n-i}$, $i = 0, 1, \ldots, n$. Then

$$\text{amin}(m) = \min_{i=0}^{n-m}(x_{i+m} - x_i) = \min_{i=0}^{n-m}(x_{i+m} + y_{n-i}) = z_{n+m}.$$

Jumping algo analysis: Estimating no. of jumps

We compute the expectation of J over a random string s (uniform i.i.d.) and balanced query $q = (\frac{m}{\sigma}, \ldots, \frac{m}{\sigma})$ as

$$\mathbb{E}(J) = \frac{n}{\mathbb{E}(\text{length of jump})}, \text{ and}$$

$$\mathbb{E}(\text{length of jump}) = Pr(\text{match}) \cdot 1 + Pr(\text{no match}) \cdot (\mathbb{E}(R_{\text{new}} - L) - m).$$

Jumping algo analysis: Estimating no. of jumps

We compute the expectation of J over a random string s (uniform i.i.d.) and balanced query $q = (\frac{m}{\sigma}, \ldots, \frac{m}{\sigma})$ as

$$\mathbb{E}(J) = \frac{n}{\mathbb{E}(\text{length of jump})}, \text{ and}$$

$$\mathbb{E}(\text{length of jump}) = Pr(\text{match}) \cdot 1 + Pr(\text{no match}) \cdot (\mathbb{E}(R_{\text{new}} - L) - m).$$

- $\mathbb{E}(R_{\text{new}} - L)$ = length of wait before we have seen $\frac{m}{\sigma}$ many of all σ characters $\approx \sqrt{2m\sigma \ln \frac{\sigma}{\sqrt{2\pi}}} + m$
  (R. May, Coupon collecting with quotas, Electr. J Comb., 2008)
- $Pr(\text{match}) = Pr(\text{random string of size } m \text{ has Parikh vector } q) = \frac{\binom{m}{\frac{m}{\sigma}, \ldots, \frac{m}{\sigma}}}{m^\sigma} \approx \sqrt{\frac{\sigma^\sigma}{m^{\sigma-1}}}$ very small for $m >> \sigma$

Jumping algo analysis: Estimating no. of jumps

We compute the expectation of J over a random string s (uniform i.i.d.) and balanced query $q = (\frac{m}{\sigma}, \ldots, \frac{m}{\sigma})$ as

$$\mathbb{E}(J) = \frac{n}{\mathbb{E}(\text{length of jump})}, \text{ and}$$

$$\begin{aligned}\mathbb{E}(\text{length of jump}) &= Pr(\text{match}) \cdot 1 + Pr(\text{no match}) \cdot (\mathbb{E}(R_{\text{new}} - L) - m) \\ &\approx (\mathbb{E}(R_{\text{new}} - L) - m) \approx \sqrt{2m\sigma \ln \frac{\sigma}{\sqrt{2\pi}}}\end{aligned}$$

So,

$$\mathbb{E}(J) = O(\frac{n}{\sqrt{m}\sqrt{\sigma \log \sigma}}).$$

Altogether, we get

$$\mathbb{E}(\text{runtime}) = O(J\sigma) = O\left(\frac{n\sigma}{\sqrt{m}\sqrt{\sigma \log \sigma}}\right) = O\left(n\sqrt{\frac{\sigma}{\log \sigma}}\frac{1}{\sqrt{m}}\right).$$

Recall: We assume a random string s (uniform i.i.d.) and balanced query $q = (\frac{m}{\sigma}, \ldots, \frac{m}{\sigma})$.

These are worst-case, both acc. to our model and our simulations:

- on non-random strings (DNA) J decreases
- on non-balanced Parikh vectors J decreases