



CLUSTERED-CLUMPS IN DEGENERATE STRINGS

- Efficient Algorithm for their Computation

Costas S. Iliopoulos, **Ritu Kundu**, Manal Mohamed

September 17, 2016

Presenting at: **MHDW, AIAI' 2016**

Travel Grants from: **Institute of Mathematics & its Applications; KCL Graduate School**

Introduction

Technical Background

Algorithms

Summary

INTRODUCTION

Definition

A clustered-clump of a given reduced set of strings $\mathcal{P} = \{P_1, \dots, P_r\}$ is a string T such that any two consecutive positions in T are covered by the same occurrence in T of a string $P \in \mathcal{P}$. More formally, W is a clustered-clump for the set \mathcal{P} such that

$$\forall i \in \{1, \dots, |T|\} \exists P \in \mathcal{P}, \exists j \in Pos_W(P) \text{ such that } j \leq i \leq j + |P| - 1,$$

where $Pos_W(P)$ is the set of positions of occurrences of P in T .

Illustration



$$\mathcal{P}: \{ \underline{\text{aba}}, \text{---bba---} \}$$
$$T: \text{b b b a b a b a b a b b b b a b a a b a b b}$$

$$\mathcal{P}: \{ \underline{\text{aba}}, \text{---bba---} \}$$
$$T: \text{b b b a b a b a b a b b b b a b a a b a b b}$$

$$\mathcal{P}: \{ \underline{\text{aba}}, \text{---bba---} \}$$
$$T: \text{b b b a b a b a b a b b b b a b a a b a b b}$$

$$\mathcal{P}: \{ \underline{\text{aba}}, \text{---bba---} \}$$
$$T: \text{b b b a b a b a b a b b b b a b a a b a b b}$$

EXAMPLE

$\mathcal{P}: \{ \underline{\text{aba}}, \text{---bba---} \}$

$T: \text{b } \boxed{\text{b b a b a b a}} \text{ b a b b b b a b a a b a b b}$

EXAMPLE

$\mathcal{P} : \{ \underline{\text{aba}}, \text{---bba---} \}$

$T : \text{b } \boxed{\text{b b a b a b a}} \text{ b a b b b b a b a a b a b b}$

The diagram shows a string T of length 19 characters: $\text{b b b a b a b a b a b b a b a a b a b b}$. The first 7 characters, b b a b a b a , are enclosed in a red box. Below this box, a dashed line is positioned under the first three characters (b b a) and a solid line is positioned under the last four characters (a b a b). To the right of the red box, the remaining 12 characters ($\text{b a b b b b a b a a b a b b}$) are shown in a light gray background. Below these characters, a dashed line is positioned under the 10th and 11th characters (b b).

$$\mathcal{P}: \{ \underline{\text{aba}}, \text{---bba---} \}$$

$$T: \text{b } \boxed{\text{b b a b a b a}} \text{ b a b b b b a b a a b a b b}$$

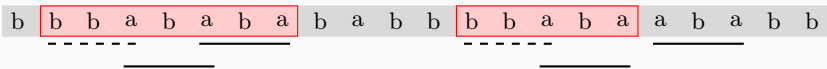
EXAMPLE

$\mathcal{P} : \{ \underline{\text{aba}}, \text{---bba---} \}$

$T : \text{b } \boxed{\text{b b a b a b a}} \text{ b a b b } \boxed{\text{b b a b a}} \text{ a b a b b}$

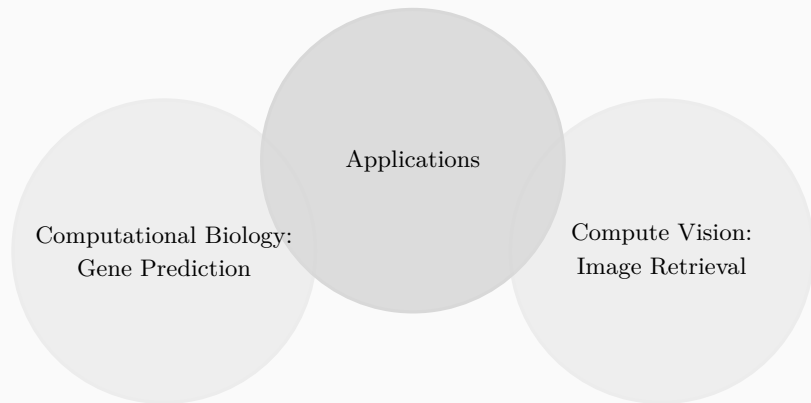
EXAMPLE

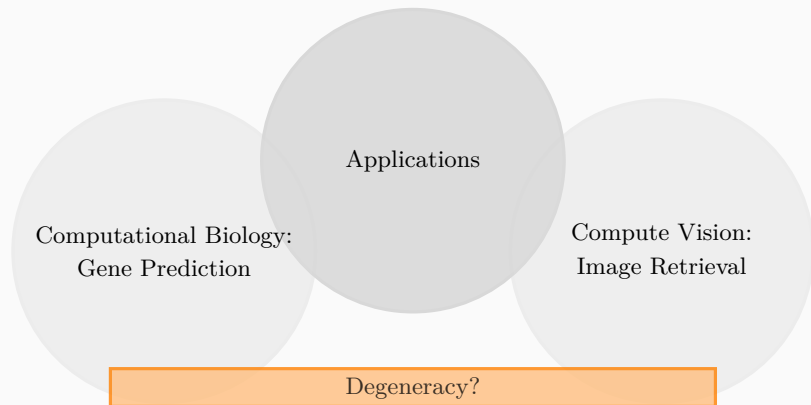
$\mathcal{P} : \{ \underline{\text{aba}}, \text{---bba---} \}$

$T :$ 

$$\mathcal{P}: \{ \underline{\text{aba}}, \text{---bba---} \}$$

$$T: \text{b } \boxed{\text{b b a b a b a}} \text{ b a b b } \boxed{\text{b b a b a}} \boxed{\text{a b a}} \text{ b b}$$





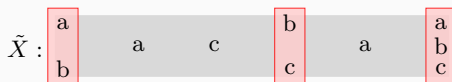
TECHNICAL BACKGROUND

Degenerate symbol

- $\tilde{\sigma}$ over an alphabet Σ is a non-empty subset of Σ .
- $|\tilde{\sigma}|$ denotes the size of the set.

Degenerate string

- $\tilde{X} = \tilde{X}[1..n]$, is a string such that every $\tilde{X}[i]$ is a degenerate symbol, $1 \leq i \leq n$.



Conserved degenerate string

A degenerate string where its number of non-solid symbols is upper-bounded by a fixed positive constant.

Match

- Two degenerate symbols $\tilde{\alpha}_1$ and $\tilde{\alpha}_2$ are said to match (represented as $\tilde{\alpha}_1 \approx \tilde{\alpha}_2$) if $\tilde{\alpha}_1 \cap \tilde{\alpha}_2 \neq \emptyset$.
- Two degenerate strings \tilde{X} and \tilde{Y} match (denoted as $\tilde{X} \approx \tilde{Y}$) if $|\tilde{X}| = |\tilde{Y}|$ and $\tilde{X}[i] \approx \tilde{Y}[i]$, for $i = 1, \dots, |\tilde{X}|$.

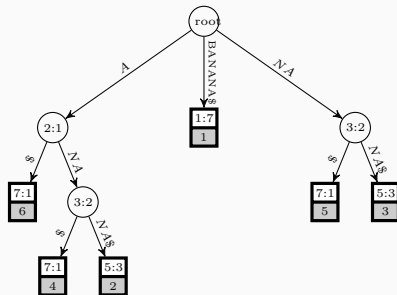
\tilde{X} :	a	a	c	b	a	a
	b			c		c
\tilde{Y} :	a	a	c	b	a	a
		c		c	c	

Occurrence

- A degenerate string \tilde{Y} is said to occur at position i in another degenerate (resp. solid) string \tilde{X} (resp. X) if $\tilde{Y} \approx \tilde{X}[i..i + |\tilde{Y}| - 1]$ (resp. $\tilde{Y} \approx X[i..i + |\tilde{Y}| - 1]$).

Suffix Tree ([Crochemore et al., 2007])

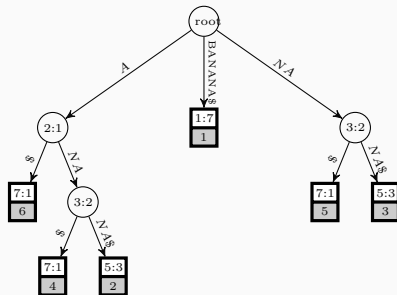
- The suffix tree $\mathcal{S}(X)$ of a non-empty string X of length n is a compact trie representing all the suffixes of X such that $\mathcal{S}(X)$ has n leaves, labelled from 1 to n .
- Example: $X = \text{"BANANA\$"};$



- Linear Time and Space Construction ([Weiner, 1973, McCreight, 1976, Ukkonen, 1995])
- After Linear time pre-processing, constant-time answer to LCP queries.
- Generalised Suffix trees

Suffix Tree ([Crochemore et al., 2007])

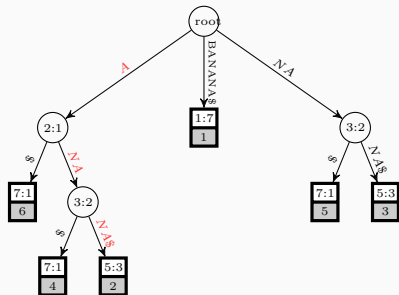
- The suffix tree $\mathcal{S}(X)$ of a non-empty string X of length n is a compact trie representing all the suffixes of X such that $\mathcal{S}(X)$ has n leaves, labelled from 1 to n .
- Example: $X = \text{"BANANA\$"};$ suffixes: { BANANA\$, ANANA\$, NANA\$, ANA\$, NA\$, A\$, \$ }



- Linear Time and Space Construction ([Weiner, 1973, McCreight, 1976, Ukkonen, 1995])
- After Linear time pre-processing, constant-time answer to LCP queries.
- Generalised Suffix trees

Suffix Tree ([Crochemore et al., 2007])

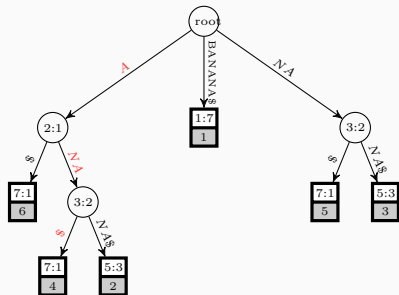
- The suffix tree $\mathcal{S}(X)$ of a non-empty string X of length n is a compact trie representing all the suffixes of X such that $\mathcal{S}(X)$ has n leaves, labelled from 1 to n .
- Example: $X = \text{"BANANA\$"};$ suffixes: { BANANA\$, ANANA\$, NANA\$, ANA\$, NA\$, A\$, \$ }



- Linear Time and Space Construction ([Weiner, 1973, McCreight, 1976, Ukkonen, 1995])
- After Linear time pre-processing, constant-time answer to LCP queries.
- Generalised Suffix trees

Suffix Tree ([Crochemore et al., 2007])

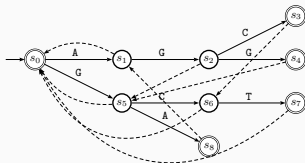
- The suffix tree $\mathcal{S}(X)$ of a non-empty string X of length n is a compact trie representing all the suffixes of X such that $\mathcal{S}(X)$ has n leaves, labelled from 1 to n .
- Example: $X = \text{"BANANA\$"};$ suffixes: { BANANA\$, ANANA\$, NANA\$, ANA\$, NA\$, A\$, \$ }



- Linear Time and Space Construction ([Weiner, 1973, McCreight, 1976, Ukkonen, 1995])
- After Linear time pre-processing, constant-time answer to LCP queries.
- Generalised Suffix trees

Aho-Corasick Automaton ([Crochemore and Rytter, 1994])

- The Aho-Corasick automaton of a set of strings \mathcal{P} , denoted $\mathcal{A}(\mathcal{P})$, is the minimal partial deterministic finite automaton accepting the set of all strings having a string of \mathcal{P} as a suffix
- Example: $\mathcal{P} = \{AGC, AGG, GCT, GA\}$



- Linear Time and Space Construction
- Linear Time Dictionary matching.

ALGORITHMS

Input

A text T of length n and a set of patterns $\mathcal{P} = \{P_1, \dots, P_r\}$, such that $m = \sum_{1 \leq i \leq r} |P_i|$.

Output

All clustered-clumps in T .


Input

A text T of length n and a set of patterns $\mathcal{P} = \{P_1, \dots, P_r\}$, such that $m = \sum_{1 \leq i \leq r} |P_i|$.

Output

All clustered-clumps in T .

Dealing with Degeneracy



Variations

CLUSTERED-CLUMPS ALGORITHMS

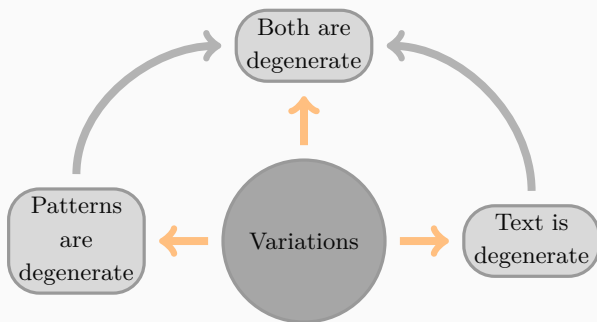
Input

A text T of length n and a set of patterns $\mathcal{P} = \{P_1, \dots, P_r\}$, such that $m = \sum_{1 \leq i \leq r} |P_i|$.

Output

All clustered-clumps in T .

Dealing with Degeneracy



PROBLEM 1: SOLID TEXT & DEGENERATE PATTERNS

Input

A text T of length n , a set of conservative degenerate patterns $\tilde{\mathcal{P}} = \{\tilde{P}_1, \dots, \tilde{P}_r\}$, and integers d and m , such that total number of non-solid symbols in $\tilde{\mathcal{P}} \leq d$, and $m = \sum_{1 \leq i \leq r} |\tilde{P}_i|$.

Output

All clustered-clumps in T .

Example

$$\tilde{\mathcal{P}} : \left\{ \underbrace{A C \begin{bmatrix} T \\ G \end{bmatrix} A A \begin{bmatrix} C \\ G \end{bmatrix} \begin{bmatrix} A \\ C \end{bmatrix} T A A}_{\tilde{P}_1}, \quad \underbrace{A T \begin{bmatrix} C \\ G \end{bmatrix} T T}_{\tilde{P}_2} \right\}$$

T : C G A C T A A C A T A A C G A A G C T A A T C T T A A C
A C $\begin{bmatrix} T \\ G \end{bmatrix}$ A A $\begin{bmatrix} C \\ G \end{bmatrix}$ $\begin{bmatrix} A \\ C \end{bmatrix}$ T A A

PROBLEM 1: SOLID TEXT & DEGENERATE PATTERNS

Input

A text T of length n , a set of conservative degenerate patterns $\tilde{\mathcal{P}} = \{\tilde{P}_1, \dots, \tilde{P}_r\}$, and integers d and m , such that total number of non-solid symbols in $\tilde{\mathcal{P}} \leq d$, and $m = \sum_{1 \leq i \leq r} |\tilde{P}_i|$.

Output

All clustered-clumps in T .

Example

$$\tilde{\mathcal{P}} : \left\{ \underbrace{A C \begin{bmatrix} T \\ G \end{bmatrix} A A \begin{bmatrix} C \\ G \end{bmatrix} \begin{bmatrix} A \\ G \end{bmatrix} T A A}_{\tilde{P}_1}, \quad \underbrace{A T \begin{bmatrix} C \\ G \end{bmatrix} T T}_{\tilde{P}_2} \right\}$$

T : C G A C T A A C A T A A C G A A G C T A A T C T T A A C
A C $\begin{bmatrix} T \\ G \end{bmatrix}$ A A $\begin{bmatrix} C \\ G \end{bmatrix}$ $\begin{bmatrix} A \\ G \end{bmatrix}$ T A A

PROBLEM 1: SOLID TEXT & DEGENERATE PATTERNS

Input

A text T of length n , a set of conservative degenerate patterns $\tilde{\mathcal{P}} = \{\tilde{P}_1, \dots, \tilde{P}_r\}$, and integers d and m , such that total number of non-solid symbols in $\tilde{\mathcal{P}} \leq d$, and $m = \sum_{1 \leq i \leq r} |\tilde{P}_i|$.

Output

All clustered-clumps in T .

Example

$$\tilde{\mathcal{P}} : \left\{ \underbrace{A C \begin{bmatrix} T \\ G \end{bmatrix} A A \begin{bmatrix} C \\ G \end{bmatrix} \begin{bmatrix} A \\ G \end{bmatrix} T A A}_{\tilde{P}_1}, \quad \underbrace{A T \begin{bmatrix} C \\ G \end{bmatrix} T T}_{\tilde{P}_2} \right\}$$

T : C G A C T A A C A T A A C G A A G C T A A T C T T A A C
A T $\begin{bmatrix} C \\ G \end{bmatrix}$ T T

PROBLEM 1: SOLID TEXT & DEGENERATE PATTERNS

Input

A text T of length n , a set of conservative degenerate patterns $\tilde{\mathcal{P}} = \{\tilde{P}_1, \dots, \tilde{P}_r\}$, and integers d and m , such that total number of non-solid symbols in $\tilde{\mathcal{P}} \leq d$, and $m = \sum_{1 \leq i \leq r} |\tilde{P}_i|$.

Output

All clustered-clumps in T .

Example

$$\tilde{\mathcal{P}} : \left\{ \underbrace{A C \begin{bmatrix} T \\ G \end{bmatrix} A A \begin{bmatrix} C \\ G \end{bmatrix} \begin{bmatrix} A \\ C \end{bmatrix} T A A}_{\tilde{P}_1}, \quad \underbrace{A T \begin{bmatrix} C \\ G \end{bmatrix} T T}_{\tilde{P}_2} \right\}$$

T : $C G$ $A C T A A C A T A A C G A A G C T A$ $A T C T T$ $A A C$

Algorithm

- Split in solid subpatterns.
- Find occurrences of solid subpatterns in T .
- Compute the locations of clustered-clumps.

Algorithm

- Split in solid subpatterns.
 - Obtain the set \mathcal{P} .
- Find occurrences of solid subpatterns in T .
- Compute the locations of clustered-clumps.

Illustration



PROBLEM 1: SOLID TEXT & DEGENERATE PATTERNS

Algorithm

- **Split in solid subpatterns.**
 - Obtain the set \mathcal{P} .
- Find occurrences of solid subpatterns in T .
- Compute the locations of clustered-clumps.

Illustration



PROBLEM 1: SOLID TEXT & DEGENERATE PATTERNS

Algorithm

- Split in solid subpatterns.
- Find occurrences of solid subpatterns in T .
 - Build Aho-Corasick automaton of \mathcal{P} .
 - Find and store all the occurrences of the solid subpatterns in the text T (in a boolean matrix of size $|\mathcal{P}| \times n$):
 - Whether previous subpattern occurs in the corresponding position.
 - Whether the non-solid symbols between previous subpattern and itself match the corresponding positions in T .
- Compute the locations of clustered-clumps.

Illustration

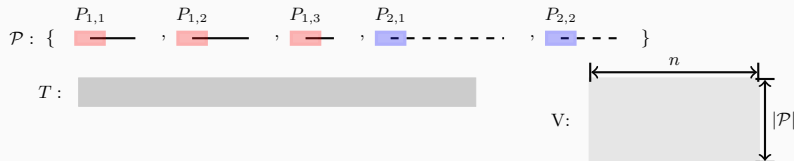
$\mathcal{P} : \{ \overset{P_{1,1}}{\text{—}} , \overset{P_{1,2}}{\text{—}} , \overset{P_{1,3}}{\text{—}} , \overset{P_{2,1}}{\text{---}} , \overset{P_{2,2}}{\text{---}} \}$

PROBLEM 1: SOLID TEXT & DEGENERATE PATTERNS

Algorithm

- Split in solid subpatterns.
- Find occurrences of solid subpatterns in T .
 - Build Aho-Corasick automaton of \mathcal{P} .
 - Find and store all the occurrences of the solid subpatterns in the text T (in a boolean matrix of size $|\mathcal{P}| \times n$):
 - Whether previous subpattern occurs in the corresponding position.
 - Whether the non-solid symbols between previous subpattern and itself match the corresponding positions in T .
- Compute the locations of clustered-clumps.

Illustration

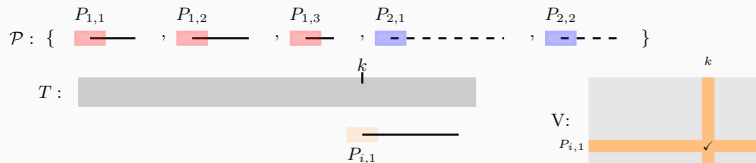


PROBLEM 1: SOLID TEXT & DEGENERATE PATTERNS

Algorithm

- Split in solid subpatterns.
- Find occurrences of solid subpatterns in T .
 - Build Aho-Corasick automaton of \mathcal{P} .
 - Find and store all the occurrences of the solid subpatterns in the text T (in a boolean matrix of size $|\mathcal{P}| \times n$):
 - Whether previous subpattern occurs in the corresponding position.
 - Whether the non-solid symbols between previous subpattern and itself match the corresponding positions in T .
- Compute the locations of clustered-clumps.

Illustration

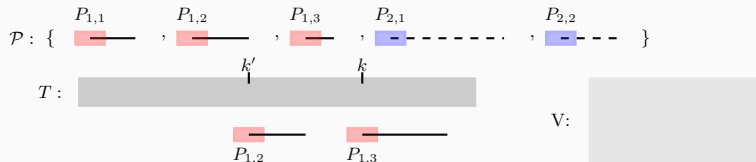


PROBLEM 1: SOLID TEXT & DEGENERATE PATTERNS

Algorithm

- Split in solid subpatterns.
- Find occurrences of solid subpatterns in T .
 - Build Aho-Corasick automaton of \mathcal{P} .
 - Find and store all the occurrences of the solid subpatterns in the text T (in a boolean matrix of size $|\mathcal{P}| \times n$):
 - Whether previous subpattern occurs in the corresponding position.
 - Whether the non-solid symbols between previous subpattern and itself match the corresponding positions in T .
- Compute the locations of clustered-clumps.

Illustration

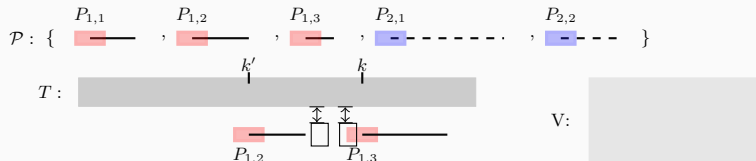


PROBLEM 1: SOLID TEXT & DEGENERATE PATTERNS

Algorithm

- Split in solid subpatterns.
- Find occurrences of solid subpatterns in T .
 - Build Aho-Corasick automaton of \mathcal{P} .
 - Find and store all the occurrences of the solid subpatterns in the text T (in a boolean matrix of size $|\mathcal{P}| \times n$):
 - Whether previous subpattern occurs in the corresponding position.
 - Whether the non-solid symbols between previous subpattern and itself match the corresponding positions in T .
- Compute the locations of clustered-clumps.

Illustration

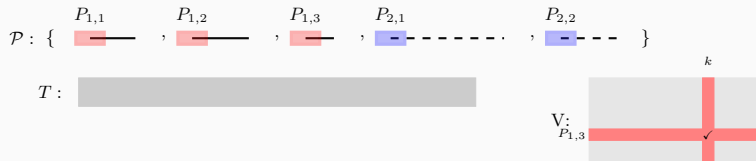


PROBLEM 1: SOLID TEXT & DEGENERATE PATTERNS

Algorithm

- Split in solid subpatterns.
- Find occurrences of solid subpatterns in T .
 - Build Aho-Corasick automaton of \mathcal{P} .
 - Find and store all the occurrences of the solid subpatterns in the text T (in a boolean matrix of size $|\mathcal{P}| \times n$):
 - Whether previous subpattern occurs in the corresponding position.
 - Whether the non-solid symbols between previous subpattern and itself match the corresponding positions in T .
- Compute the locations of clustered-clumps.

Illustration

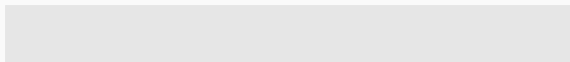


Algorithm

- Split in solid subpatterns.
- Find occurrences of solid subpatterns in T .
- **Compute the locations of clustered-clumps.**
 - Single scan of an array of size n that stores the length of the longest pattern occurring at each position of the text.

Illustration

L:



PROBLEM 1: SOLID TEXT & DEGENERATE PATTERNS

Algorithm

- Split in solid subpatterns.
- Find occurrences of solid subpatterns in T .
- **Compute the locations of clustered-clumps.**
 - Single scan of an array of size n that stores the length of the longest pattern occurring at each position of the text.

Illustration

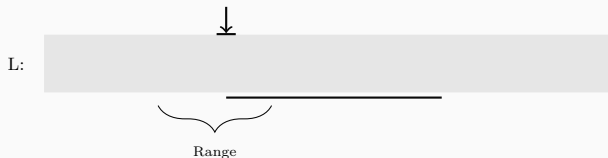


PROBLEM 1: SOLID TEXT & DEGENERATE PATTERNS

Algorithm

- Split in solid subpatterns.
- Find occurrences of solid subpatterns in T .
- **Compute the locations of clustered-clumps.**
 - Single scan of an array of size n that stores the length of the longest pattern occurring at each position of the text.

Illustration

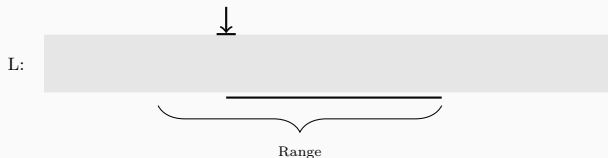


PROBLEM 1: SOLID TEXT & DEGENERATE PATTERNS

Algorithm

- Split in solid subpatterns.
- Find occurrences of solid subpatterns in T .
- **Compute the locations of clustered-clumps.**
 - Single scan of an array of size n that stores the length of the longest pattern occurring at each position of the text.

Illustration

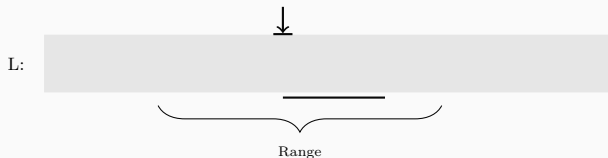


PROBLEM 1: SOLID TEXT & DEGENERATE PATTERNS

Algorithm

- Split in solid subpatterns.
- Find occurrences of solid subpatterns in T .
- **Compute the locations of clustered-clumps.**
 - Single scan of an array of size n that stores the length of the longest pattern occurring at each position of the text.

Illustration

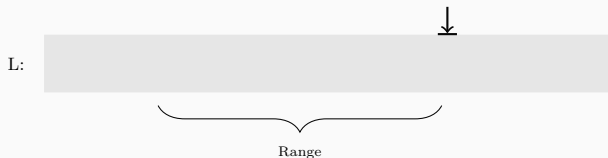


PROBLEM 1: SOLID TEXT & DEGENERATE PATTERNS

Algorithm

- Split in solid subpatterns.
- Find occurrences of solid subpatterns in T .
- **Compute the locations of clustered-clumps.**
 - Single scan of an array of size n that stores the length of the longest pattern occurring at each position of the text.

Illustration

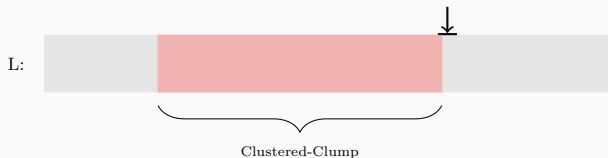


PROBLEM 1: SOLID TEXT & DEGENERATE PATTERNS

Algorithm

- Split in solid subpatterns.
- Find occurrences of solid subpatterns in T .
- **Compute the locations of clustered-clumps.**
 - Single scan of an array of size n that stores the length of the longest pattern occurring at each position of the text.


Illustration





Analysis

- Split in solid subpatterns.
- Find occurrences of solid subpatterns in T .
 - Build Aho-Corasick automaton of \mathcal{P} .
 - Find and store all the occurrences of the solid subpatterns in the text T (in a boolean matrix):
 - Whether previous subpattern occurs in the corresponding position.
 - Whether the non-solid symbols between previous subpattern and itself match the corresponding positions in T .
- Compute the locations of clustered-clumps.





Analysis

- Split in solid subpatterns.  $O(m)$
- Find occurrences of solid subpatterns in T .
 - Build Aho-Corasick automaton of \mathcal{P} .
 - Find and store all the occurrences of the solid subpatterns in the text T (in a boolean matrix):
 - Whether previous subpattern occurs in the corresponding position.
 - Whether the non-solid symbols between previous subpattern and itself match the corresponding positions in T .
- Compute the locations of clustered-clumps.




Analysis

- Split in solid subpatterns.  $O(m)$
- Find occurrences of solid subpatterns in T .
 - Build Aho-Corasick automaton of \mathcal{P} .  $O(m)$
 - Find and store all the occurrences of the solid subpatterns in the text T (in a boolean matrix):
 - Whether previous subpattern occurs in the corresponding position.
 - Whether the non-solid symbols between previous subpattern and itself match the corresponding positions in T .
- Compute the locations of clustered-clumps.

Analysis

- Split in solid subpatterns.  $O(m)$
- Find occurrences of solid subpatterns in T .
 - Build Aho-Corasick automaton of \mathcal{P} .  $O(m)$
 - Find and store all the occurrences of the solid subpatterns in the text T (in a boolean matrix):
 - Whether previous subpattern occurs in the corresponding position.  $O(1)$
 - Whether the non-solid symbols between previous subpattern and itself match the corresponding positions in T .  $O(d)$
- Compute the locations of clustered-clumps.

Analysis

- Split in solid subpatterns.  $O(m)$
- Find occurrences of solid subpatterns in T .
 - Build Aho-Corasick automaton of \mathcal{P} .  $O(m)$
 - Find and store all the occurrences of the solid subpatterns in the text T (in a boolean matrix):  $O(nd)$
 - Whether previous subpattern occurs in the corresponding position.
 - Whether the non-solid symbols between previous subpattern and itself match the corresponding positions in T .
- Compute the locations of clustered-clumps.

Analysis

- Split in solid subpatterns. $\longrightarrow O(m)$
- Find occurrences of solid subpatterns in T .
 - Build Aho-Corasick automaton of \mathcal{P} . $\longrightarrow O(m)$
 - Find and store all the occurrences of the solid subpatterns in the text T (in a boolean matrix): $\longrightarrow O(nd)$
 - Whether previous subpattern occurs in the corresponding position.
 - Whether the non-solid symbols between previous subpattern and itself match the corresponding positions in T .
- Compute the locations of clustered-clumps. $\longrightarrow O(n)$

Analysis

- Split in solid subpatterns.
- Find occurrences of solid subpatterns in T .
 - Build Aho-Corasick automaton of \mathcal{P} .
 - Find and store all the occurrences of the solid subpatterns in the text T (in a boolean matrix):
 - Whether previous subpattern occurs in the corresponding position.
 - Whether the non-solid symbols between previous subpattern and itself match the corresponding positions in T .
- Compute the locations of clustered-clumps.

Running time

Linear: $(O(m + nd))$

PROBLEM 2: DEGENERATE TEXT & SOLID PATTERNS

Input

A conservative degenerate text \tilde{T} of length n , a set of patterns $\mathcal{P} = \{P_1, \dots, P_r\}$, and integers d and m , such that the total number of non-solid symbols in $\tilde{T} \leq d$, and $m = \sum_{1 \leq i \leq r} |P_i|$.

Output

All clustered-clumps in \tilde{T} .

Example

$\mathcal{P} : \{ \underline{AGC} \text{ }_{P_1}, \text{ } \text{---} \underline{AGG} \text{---} \text{ }_{P_2}, \text{ } \text{.....} \underline{GCT} \text{.....} \text{ }_{P_3}, \text{ } \text{ } \underline{\text{GA}} \text{ }_{P_4} \}$

$\tilde{T} : C \ A \ T \ T \ A \ [A] \ G \ A \ G \ C \ [T] \ C \ T \ T$

PROBLEM 2: DEGENERATE TEXT & SOLID PATTERNS

Input

A conservative degenerate text \tilde{T} of length n , a set of patterns $\mathcal{P} = \{P_1, \dots, P_r\}$, and integers d and m , such that the total number of non-solid symbols in $\tilde{T} \leq d$, and $m = \sum_{1 \leq i \leq r} |P_i|$.

Output

All clustered-clumps in \tilde{T} .

Example

$\mathcal{P} : \{ \underbrace{AGC}_{P_1}, \quad \text{---} \underbrace{AGG}_{P_2} \text{---}, \quad \dots \underbrace{GCT}_{P_3} \dots, \quad \underbrace{GA}_{P_4} \}$

$\tilde{T} : C \quad A \quad T \quad T \quad A \quad \left[\begin{array}{c} A \\ G \end{array} \right] \quad G \quad A \quad G \quad C \quad \left[\begin{array}{c} T \\ G \end{array} \right] \quad C \quad T \quad T$

A G G

PROBLEM 2: DEGENERATE TEXT & SOLID PATTERNS

Input

A conservative degenerate text \tilde{T} of length n , a set of patterns $\mathcal{P} = \{P_1, \dots, P_r\}$, and integers d and m , such that the total number of non-solid symbols in $\tilde{T} \leq d$, and $m = \sum_{1 \leq i \leq r} |P_i|$.

Output

All clustered-clumps in \tilde{T} .

Example

$\mathcal{P} : \{ \underbrace{AGC}_{P_1}, \quad \underbrace{AGG}_{P_2}, \quad \underbrace{GCT}_{P_3}, \quad \underbrace{GA}_{P_4} \}$

$\tilde{T} : C \quad A \quad T \quad T \quad A \quad [A] \quad G \quad A \quad G \quad C \quad [T] \quad C \quad T \quad T$

~~~~~

G    A



## PROBLEM 2: DEGENERATE TEXT & SOLID PATTERNS

### Input

A conservative degenerate text  $\tilde{T}$  of length  $n$ , a set of patterns  $\mathcal{P} = \{P_1, \dots, P_r\}$ , and integers  $d$  and  $m$ , such that the total number of non-solid symbols in  $\tilde{T} \leq d$ , and  $m = \sum_{1 \leq i \leq r} |P_i|$ .

### Output

All clustered-clumps in  $\tilde{T}$ .

### Example

$\mathcal{P} : \{ \underbrace{AGC}_{P_1}, \quad \underbrace{AGG}_{P_2}, \quad \underbrace{GCT}_{P_3}, \quad \underbrace{GA}_{P_4} \}$

$\tilde{T} : C \quad A \quad T \quad T \quad A \quad [A] \quad G \quad A \quad G \quad C \quad [T] \quad C \quad T \quad T$

-----  
~~~~~  
G C T

PROBLEM 2: DEGENERATE TEXT & SOLID PATTERNS

Input

A conservative degenerate text \tilde{T} of length n , a set of patterns $\mathcal{P} = \{P_1, \dots, P_r\}$, and integers d and m , such that the total number of non-solid symbols in $\tilde{T} \leq d$, and $m = \sum_{1 \leq i \leq r} |P_i|$.

Output

All clustered-clumps in \tilde{T} .

Example

$\mathcal{P} : \{ \underbrace{AGC}_{P_1}, \quad \underbrace{AGG}_{P_2}, \quad \underbrace{GCT}_{P_3}, \quad \underbrace{GA}_{P_4} \}$

$\tilde{T} : C \quad A \quad T \quad T \quad A \quad \left[\begin{smallmatrix} A \\ G \end{smallmatrix} \right] \quad G \quad A \quad G \quad C \quad \left[\begin{smallmatrix} T \\ G \end{smallmatrix} \right] \quad C \quad T \quad T$

Algorithm

- Substitute.
- Find occurrences of solid patterns in T_λ .
- Compute the locations of clustered-clumps.

Illustration

$\mathcal{P}: \{ \text{—————}, \text{-----} \}$

$\tilde{T}: \text{ [] [] [] [] [] [] }$



Algorithm

- **Substitute.**
 - Obtain T_λ by replacing each non-solid symbol with a unique symbol.
- Find occurrences of solid patterns in T_λ .
- Compute the locations of clustered-clumps.

Illustration



PROBLEM 2: DEGENERATE TEXT & SOLID PATTERNS

Algorithm

- Substitute.
- Find occurrences of solid patterns in T_λ .
 - Use algorithm given by Crochemore et. al [Crochemore et al., 2016]:
 - Build generalised suffix tree of T_λ and the patterns in \mathcal{P} .
 - Checking whether a pattern P_i occurs at a certain position in T is realized by at most d longest common ancestor (LCA) queries.
- Compute the locations of clustered-clumps.

Illustration

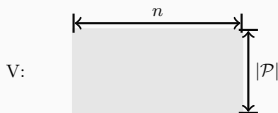


PROBLEM 2: DEGENERATE TEXT & SOLID PATTERNS

Algorithm

- Substitute.
- Find occurrences of solid patterns in T_λ .
- **Compute the locations of clustered-clumps.**
 - Single scan of an array of size n that stores the length of the longest pattern occurring at each position of the text.


Illustration





Analysis

- Substitute.
- Find occurrences of solid patterns in T_λ .
- Compute the locations of clustered-clumps.

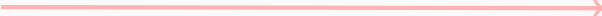
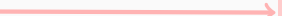
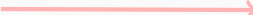
Analysis

- Substitute.  $O(n)$
- Find occurrences of solid patterns in T_λ .
- Compute the locations of clustered-clumps.

Analysis

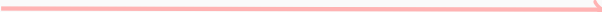

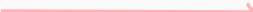
- Substitute.  $O(n)$
- Find occurrences of solid patterns in T_λ .  $O(dn)$
- Compute the locations of clustered-clumps.

Analysis

- Substitute.  $O(n)$
- Find occurrences of solid patterns in T_λ .  $O(dn)$
- Compute the locations of clustered-clumps.  $O(n)$

PROBLEM 2: DEGENERATE TEXT & SOLID PATTERNS

Analysis

- Substitute.  $O(n)$
- Find occurrences of solid patterns in T_λ .  $O(dn)$
- Compute the locations of clustered-clumps.  $O(n)$

Running time

Linear: ($O(m + nd)$)

PROBLEM 3: DEGENERATE TEXT & DEGENERATE PATTERNS

Input

A conservative degenerate text \tilde{T} of length n , a set of conservative degenerate patterns $\tilde{\mathcal{P}} = \{\tilde{P}_1, \dots, \tilde{P}_r\}$, and integers d and m , such that total number of non-solid symbols in both \tilde{T} and $\tilde{\mathcal{P}} \leq d$ and $m = \sum_{1 \leq i \leq r} |\tilde{P}_i|$.

Output

All clustered-clumps in \tilde{T} .

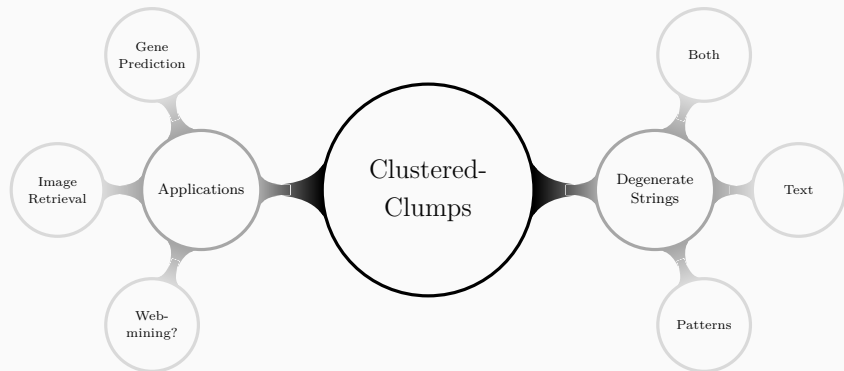
Algorithm

- Substitute to obtain T_λ .
- Split patterns into solid subpatterns.
- Find occurrences of solid subpatterns in T_λ .
- Compute the locations of clustered-clumps.

Running time

Linear: $(O(m + nd))$

SUMMARY





Crochemore, M., Hancart, C., and Lecroq, T. (2007).
Algorithms on Strings.
Cambridge University Press.
392 pages.



Crochemore, M., Iliopoulos, C. S., Kundu, R., Mohamed, M., and Vayani, F. (2016).
Linear algorithm for conservative degenerate pattern matching.
Engineering Applications of Artificial Intelligence, 51:109 – 114.



Crochemore, M. and Rytter, W. (1994).
Text algorithms.
Oxford University Press.



McCreight, E. M. (1976).
A space-economical suffix tree construction algorithm.
Journal of the ACM (JACM), 23(2):262–272.



Ukkonen, E. (1995).
On-line construction of suffix trees.
Algorithmica, 14(3):249–260.



Weiner, P. (1973).
Linear pattern matching algorithms.
In Proceedings of the 14th IEEE Annual Symposium on Switching and Automata Theory,
pages 1–11. Institute of Electrical Electronics Engineer.

Thank You!

Contact: ritu.kundu@kcl.ac.uk