Pseudo-realtime Pattern Matching: Closing the Gap

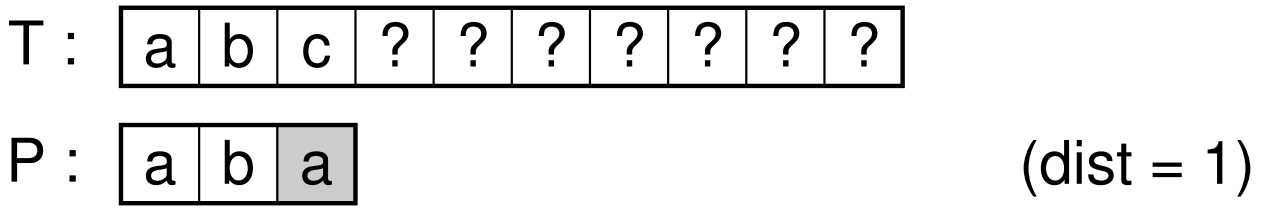
Raphaël Clifford and Benjamin Sach

{clifford,sach}@cs.bris.ac.uk

Department of Computer Science,
University of Bristol

Introduction: Pseudo-realtime pattern matching

- Consider a text, T (length n) and a pattern P (length m)
- We assume we have P in advance but T arrives online...



- Find the **distance** between $T[i, i + m - 1]$ and P for all i.

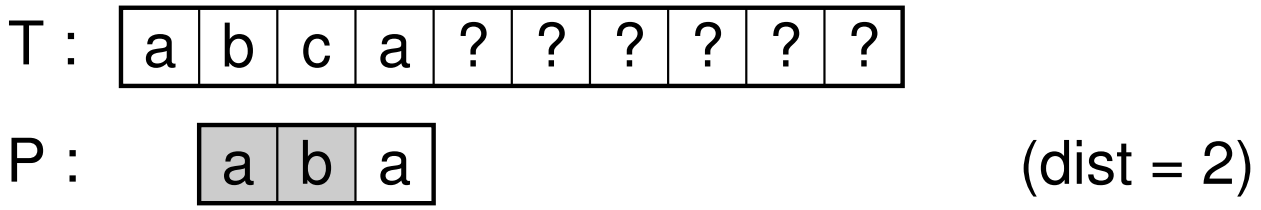
(Hamming distance shown)

- We are concerned with **worst-case** time per text character.

(Pseudo-Realtime)

Introduction: Pseudo-realtime pattern matching

- Consider a text, T (length n) and a pattern P (length m)
- We assume we have P in advance but T arrives online...



- Find the **distance** between $T[i, i + m - 1]$ and P for all i.

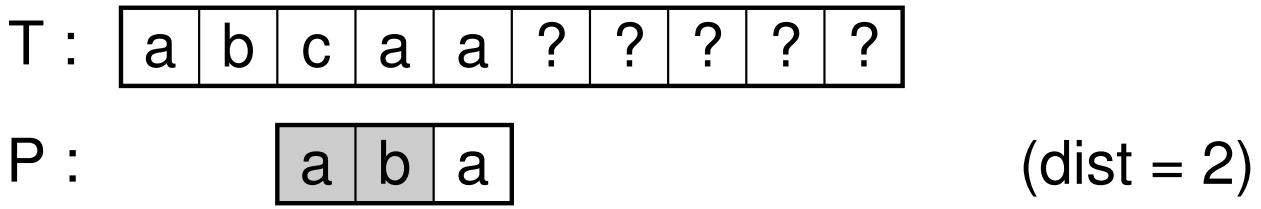
(Hamming distance shown)

- We are concerned with **worst-case** time per text character.

(Pseudo-Realtime)

Introduction: Pseudo-realtime pattern matching

- Consider a text, T (length n) and a pattern P (length m)
- We assume we have P in advance but T arrives online...



- Find the **distance** between $T[i, i + m - 1]$ and P for all i.

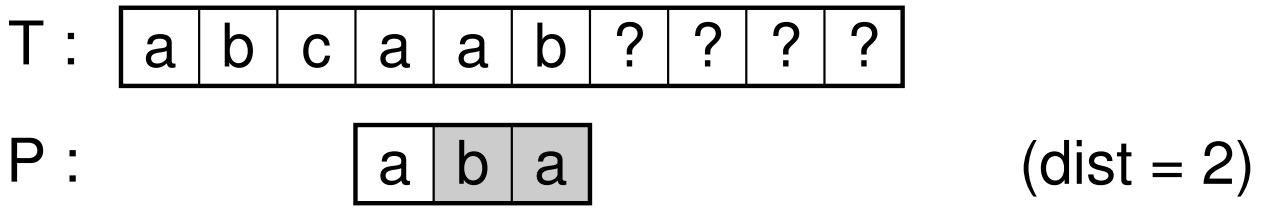
(Hamming distance shown)

- We are concerned with **worst-case** time per text character.

(Pseudo-Realtime)

Introduction: Pseudo-realtime pattern matching

- Consider a text, T (length n) and a pattern P (length m)
- We assume we have P in advance but T arrives online...



- Find the **distance** between $T[i, i + m - 1]$ and P for all i.

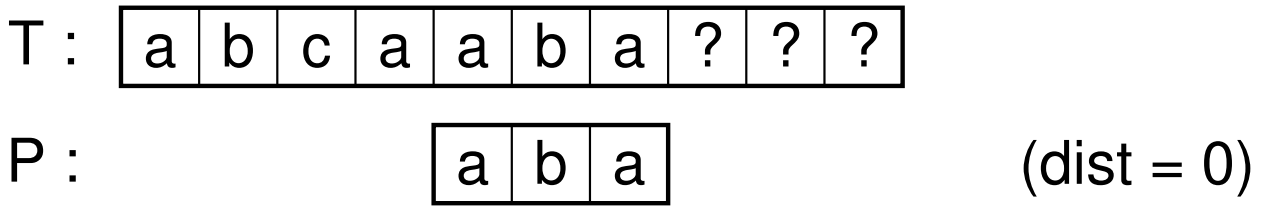
(Hamming distance shown)

- We are concerned with **worst-case** time per text character.

(Pseudo-Realtime)

Introduction: Pseudo-realtime pattern matching

- Consider a text, T (length n) and a pattern P (length m)
- We assume we have P in advance but T arrives online...



- Find the **distance** between $T[i, i + m - 1]$ and P for all i.

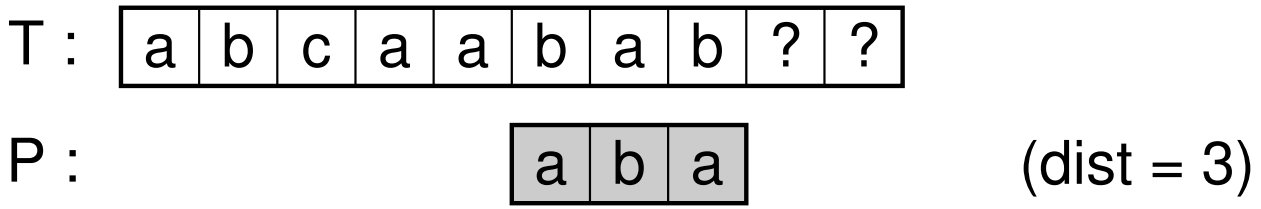
(Hamming distance shown)

- We are concerned with **worst-case** time per text character.

(Pseudo-Realtime)

Introduction: Pseudo-realtime pattern matching

- Consider a text, T (length n) and a pattern P (length m)
- We assume we have P in advance but T arrives online...



- Find the **distance** between $T[i, i + m - 1]$ and P for all i.

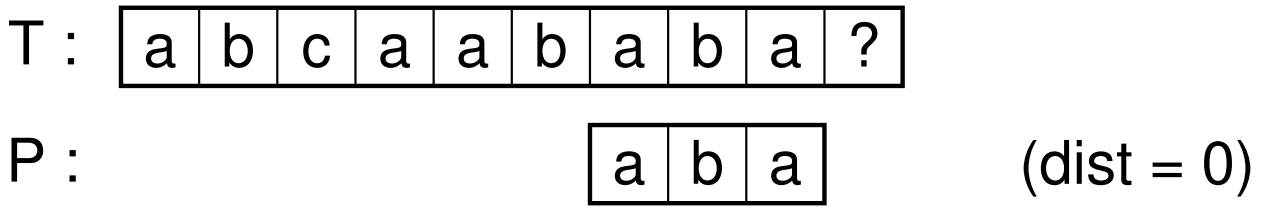
(Hamming distance shown)

- We are concerned with **worst-case** time per text character.

(Pseudo-Realtime)

Introduction: Pseudo-realtime pattern matching

- Consider a text, T (length n) and a pattern P (length m)
- We assume we have P in advance but T arrives online...



- Find the **distance** between $T[i, i + m - 1]$ and P for all i.

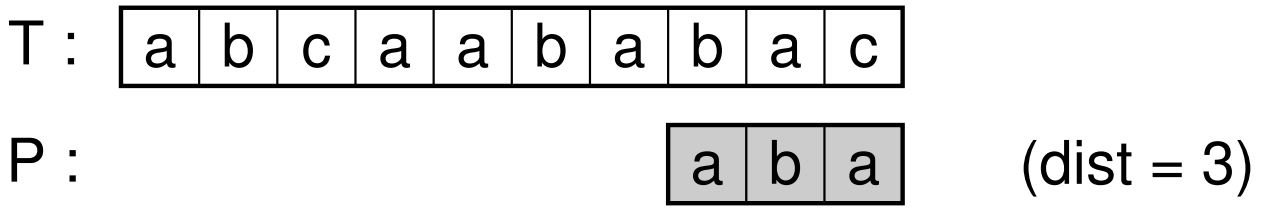
(Hamming distance shown)

- We are concerned with **worst-case** time per text character.

(Pseudo-Realtime)

Introduction: Pseudo-realtime pattern matching

- Consider a text, T (length n) and a pattern P (length m)
- We assume we have P in advance but T arrives online...



- Find the **distance** between $T[i, i + m - 1]$ and P for all i.

(Hamming distance shown)

- We are concerned with **worst-case** time per text character.

(Pseudo-Realtime)

Previous and Related work

Problem	Offline per char time	Online/PsR penalty	Online Space
Local, Deterministic (C., Efremenko, Porat and Porat. CPM 2008)			
Matching with wildcards	$O(\log m)$	$O(\log m)$	$O(m)$
Hamming distance	$O(\sqrt{m \log m})$	$O(1)$	$O(m)$
k-mismatch	$O(\sqrt{k \log k})$	$O(\log m)$	$O(m)$
L_1 distance	$O(\sqrt{m \log m})$	$O(1)$	$O(m)$
L_2 distance	$O(\log m)$	$O(\log m)$	$O(m)$
Non-Local, Deterministic (C. and S. CPM 2009)			
Parameterised	$O(\log \lvert\Sigma_P\rvert)$	$O(1)$	$O(m)$
L_2 rearrangement	$O(\log m)$	$O(\log m)$	$O(m)$
Swap-mismatch	$O(\sqrt{m \log m})$	$O(1)$	$O(m)$
k-differences	$O(k)$	$O(\log m)$	$O(m)$
Local, Randomised (Porat and Porat. FOCS 2009)			
Exact matching	$O(1)$	$O(\log m)$	$O(\log m)$
k-mismatch	$O(\sqrt{k \log k})$	$O(k^{1.5}\text{plog}(m))$	$O(k^3\text{plog}(m))$

(plog(m) is polynomial in $log(m)$)

Previous and Related work

Problem	Offline per char time	Online/PsR penalty	Online Space
Local, Deterministic (C., Efremenko, Porat and Porat. CPM 2008)			
Matching with wildcards	$\mathbf{O}(\log \mathbf{m})$	$O(\log m)$	$O(m)$
Hamming distance	$\mathbf{O}(\sqrt{\mathbf{m} \log \mathbf{m}})$	$O(1)$	$O(m)$
k-mismatch	$\mathbf{O}(\sqrt{\mathbf{k} \log \mathbf{k}})$	$O(\log m)$	$O(m)$
L_1 distance	$\mathbf{O}(\sqrt{\mathbf{m} \log \mathbf{m}})$	$O(1)$	$O(m)$
L_2 distance	$\mathbf{O}(\log \mathbf{m})$	$O(\log m)$	$O(m)$
Non-Local, Deterministic (C. and S. CPM 2009)			
Parameterised	$\mathbf{O}(\log \lvert\Sigma_P\rvert)$	$O(1)$	$O(m)$
L_2 rearrangement	$\mathbf{O}(\log \mathbf{m})$	$O(\log m)$	$O(m)$
Swap-mismatch	$\mathbf{O}(\sqrt{\mathbf{m} \log \mathbf{m}})$	$O(1)$	$O(m)$
k-differences	$\mathbf{O}(\mathbf{k})$	$O(\log m)$	$O(m)$
Local, Randomised (Porat and Porat. FOCS 2009)			
Exact matching	$\mathbf{O}(\mathbf{1})$	$O(\log m)$	$O(\log m)$
k-mismatch	$\mathbf{O}(\sqrt{\mathbf{k} \log \mathbf{k}})$	$O(k^{1.5}\text{plog}(m))$	$O(k^3\text{plog}(m))$

(plog(m) is polynomial in $log(m)$)

Previous and Related work

Problem	Offline per char time	Online/PsR penalty	Online Space
Local, Deterministic (C., Efremenko, Porat and Porat. CPM 2008)			
Matching with wildcards	$O(\log m)$	$\mathbf{O(\log m)}$	$O(m)$
Hamming distance	$O(\sqrt{m \log m})$	$\mathbf{O(1)}$	$O(m)$
k-mismatch	$O(\sqrt{k \log k})$	$\mathbf{O(\log m)}$	$O(m)$
L_1 distance	$O(\sqrt{m \log m})$	$\mathbf{O(1)}$	$O(m)$
L_2 distance	$O(\log m)$	$\mathbf{O(\log m)}$	$O(m)$
Non-Local, Deterministic (C. and S. CPM 2009)			
Parameterised	$O(\log \lvert\Sigma_P\rvert)$	$\mathbf{O(1)}$	$O(m)$
L_2 rearrangement	$O(\log m)$	$\mathbf{O(\log m)}$	$O(m)$
Swap-mismatch	$O(\sqrt{m \log m})$	$\mathbf{O(1)}$	$O(m)$
k-differences	$O(k)$	$\mathbf{O(\log m)}$	$O(m)$
Local, Randomised (Porat and Porat. FOCS 2009)			
Exact matching	$O(1)$	$\mathbf{O(\log m)}$	$O(\log m)$
k-mismatch	$O(\sqrt{k \log k})$	$\mathbf{O(k^{1.5}\text{plog}(m))}$	$O(k^3\text{plog}(m))$

(plog(m) is polynomial in $log(m)$)

Previous and Related work

Problem	Offline per char time	Online/PsR penalty	Online Space
Local, Deterministic (C., Efremenko, Porat and Porat. CPM 2008)			
Matching with wildcards	$O(\log m)$	$O(\log m)$	$\mathbf{O(m)}$
Hamming distance	$O(\sqrt{m \log m})$	$O(1)$	$\mathbf{O(m)}$
k-mismatch	$O(\sqrt{k \log k})$	$O(\log m)$	$\mathbf{O(m)}$
L_1 distance	$O(\sqrt{m \log m})$	$O(1)$	$\mathbf{O(m)}$
L_2 distance	$O(\log m)$	$O(\log m)$	$\mathbf{O(m)}$
Non-Local, Deterministic (C. and S. CPM 2009)			
Parameterised	$O(\log \lvert\Sigma_P\rvert)$	$O(1)$	$\mathbf{O(m)}$
L_2 rearrangement	$O(\log m)$	$O(\log m)$	$\mathbf{O(m)}$
Swap-mismatch	$O(\sqrt{m \log m})$	$O(1)$	$\mathbf{O(m)}$
k-differences	$O(k)$	$O(\log m)$	$\mathbf{O(m)}$
Local, Randomised (Porat and Porat. FOCS 2009)			
Exact matching	$O(1)$	$O(\log m)$	$\mathbf{O}(\log \mathbf{m})$
k-mismatch	$O(\sqrt{k \log k})$	$O(k^{1.5}\text{plog}(m))$	$\mathbf{O(k^3 \text{plog}(m))}$

(plog(m) is polynomial in $log(m)$)

Previous and Related work

Problem	Offline per char time	Online/PsR penalty	Online Space
Local, Deterministic (C., Efremenko, Porat and Porat. CPM 2008)			
Matching with wildcards	$O(\log m)$	$O(\log m)$	$O(m)$
Hamming distance	$O(\sqrt{m \log m})$	$O(1)$	$O(m)$
k-mismatch	$O(\sqrt{k \log k})$	$O(\log m)$	$O(m)$
L_1 distance	$O(\sqrt{m \log m})$	$O(1)$	$O(m)$
L_2 distance	$O(\log m)$	$O(\log m)$	$O(m)$
Non-Local, Deterministic (C. and S. CPM 2009)			
Parameterised	$O(\log \lvert\Sigma_P\rvert)$	$O(1)$	$O(m)$
L_2 rearrangement	$O(\log m)$	$O(\log m)$	$O(m)$
Swap-mismatch	$O(\sqrt{m \log m})$	$O(1)$	$O(m)$
k-differences	$O(k)$	$O(\log m)$	$O(m)$
Local, Randomised (Porat and Porat. FOCS 2009)			
Exact matching	$O(1)$	$O(\log m)$	$O(\log m)$
k-mismatch	$O(\sqrt{k \log k})$	$O(k^{1.5}\text{plog}(m))$	$O(k^3\text{plog}(m))$

(plog(m) is polynomial in $log(m)$)

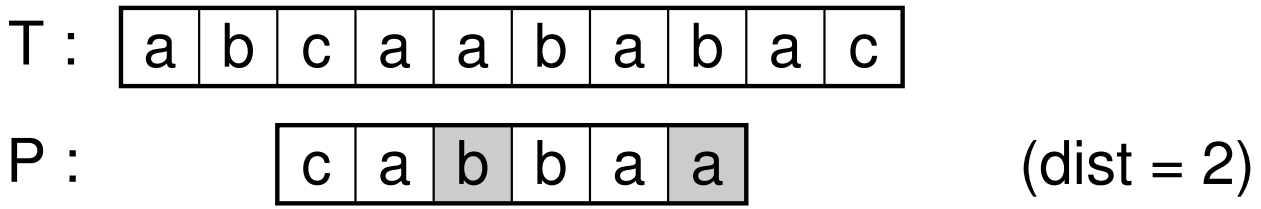
Previous and Related work

Problem	Offline per char time	Online/PsR penalty	Online Space
Local, Deterministic (C., Efremenko, Porat and Porat. CPM 2008)			
Matching with wildcards	$O(\log m)$	$O(\log m)$	$O(m)$
Hamming distance	$O(\sqrt{m \log m})$	$O(1)$	$O(m)$
***k*-mismatch**	$\mathbf{O(\sqrt{k \log k})}$	$\mathbf{O(\log m)}$	$\mathbf{O(m)}$
L_1 distance	$O(\sqrt{m \log m})$	$O(1)$	$O(m)$
L_2 distance	$O(\log m)$	$O(\log m)$	$O(m)$
Non-Local, Deterministic (C. and S. CPM 2009)			
Parameterised	$O(\log \lvert\Sigma_P\rvert)$	$O(1)$	$O(m)$
L_2 rearrangement	$O(\log m)$	$O(\log m)$	$O(m)$
Swap-mismatch	$O(\sqrt{m \log m})$	$O(1)$	$O(m)$
***k*-differences**	$\mathbf{O(k)}$	$\mathbf{O(\log m)}$	$\mathbf{O(m)}$
Local, Randomised (Porat and Porat. FOCS 2009)			
Exact matching	$O(1)$	$O(\log m)$	$O(\log m)$
k-mismatch	$O(\sqrt{k \log k})$	$O(k^{1.5}\text{plog}(m))$	$O(k^3\text{plog}(m))$

(plog(m) is polynomial in $log(m)$)

Can we improve the k-mismatch result?

- The k-mismatch problem is to find all alignments of P with T where the **Hamming distance** is at most k.



- Offline: $O(n\sqrt{k \log k})$ time[1].
- Pseudo-realtime: $O(\sqrt{k \log k} \log m)$ time per character.

Problem: k is often very small in comparison with m.

[1] Amir, Lewenstein, Porat. SODA 2000

How is it done offline? *ALP 2000*

Def: A character is *frequent* if it occurs at least $\sqrt{k}$ times in P.

How is it done offline? *ALP 2000*

Def: A character is *frequent* if it occurs at least $\sqrt{k}$ times in P.

Case 1: Fewer than $\sqrt{k}$ *frequent* characters in P.

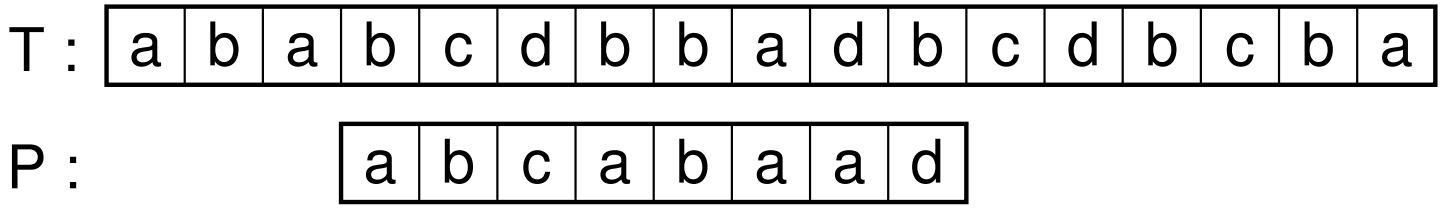
- Here we count *matches* rather than *mismatches*.

How is it done offline? *ALP 2000*

Def: A character is *frequent* if it occurs at least $\sqrt{k}$ times in P.

Case 1: Fewer than $\sqrt{k}$ *frequent* characters in P.

- Here we count *matches* rather than *mismatches*.



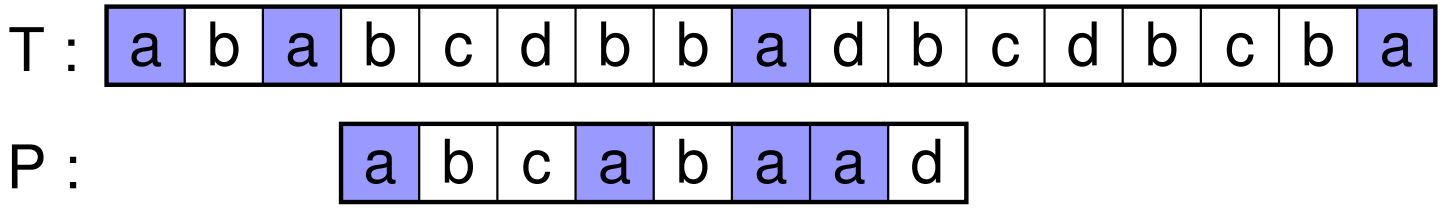
- Matches with a single *frequent* character can be found using cross-correlations in $O(n \log m)$ time.

How is it done offline? *ALP 2000*

Def: A character is *frequent* if it occurs at least $\sqrt{k}$ times in P.

Case 1: Fewer than $\sqrt{k}$ *frequent* characters in P.

- Here we count *matches* rather than *mismatches*.



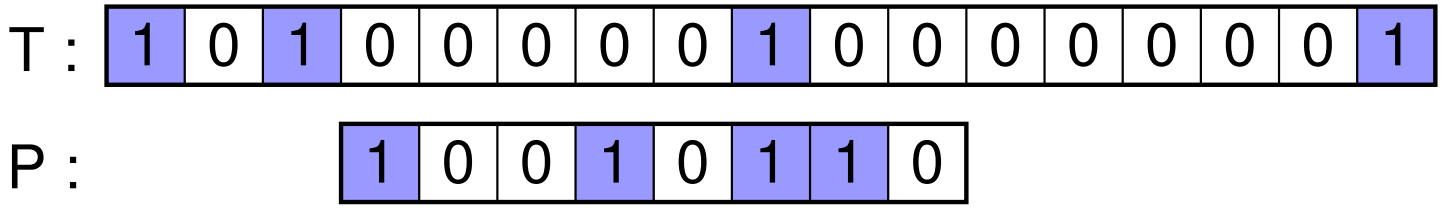
- Matches with a single *frequent* character can be found using cross-correlations in $O(n \log m)$ time.

How is it done offline? *ALP 2000*

Def: A character is *frequent* if it occurs at least $\sqrt{k}$ times in P.

Case 1: Fewer than $\sqrt{k}$ *frequent* characters in P.

- Here we count *matches* rather than *mismatches*.



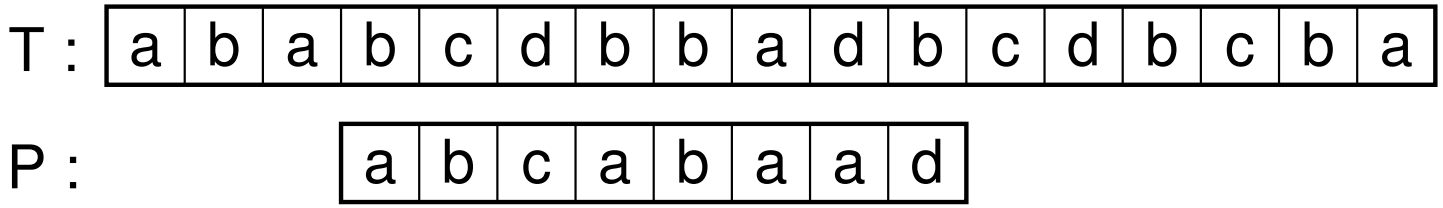
- Matches with a single *frequent* character can be found using cross-correlations in $O(n \log m)$ time.

How is it done offline? *ALP 2000*

Def: A character is *frequent* if it occurs at least $\sqrt{k}$ times in P.

Case 1: Fewer than $\sqrt{k}$ *frequent* characters in P.

- Here we count *matches* rather than *mismatches*.



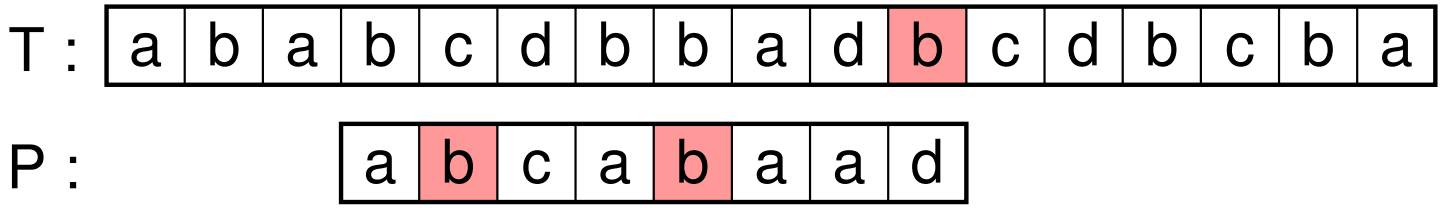
- Matches with a single *frequent* character can be found using cross-correlations in $O(n \log m)$ time.
- Each *infrequent* text character can't match very many times.

How is it done offline? *ALP 2000*

Def: A character is *frequent* if it occurs at least $\sqrt{k}$ times in P.

Case 1: Fewer than $\sqrt{k}$ *frequent* characters in P.

- Here we count *matches* rather than *mismatches*.



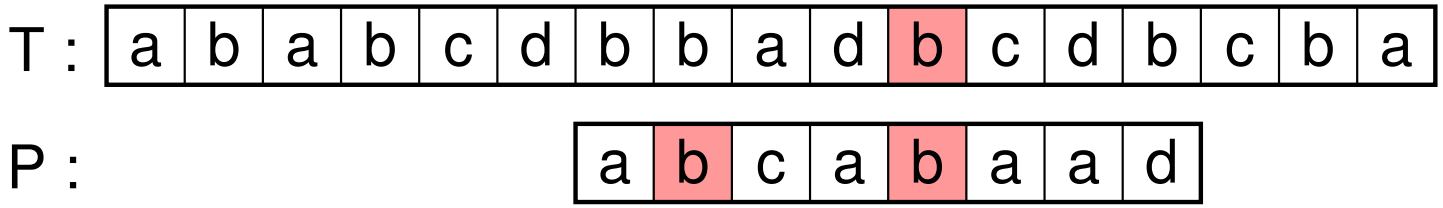
- Matches with a single *frequent* character can be found using cross-correlations in $O(n \log m)$ time.
- Each *infrequent* text character can't match very many times.

How is it done offline? *ALP 2000*

Def: A character is *frequent* if it occurs at least $\sqrt{k}$ times in P.

Case 1: Fewer than $\sqrt{k}$ *frequent* characters in P.

- Here we count *matches* rather than *mismatches*.



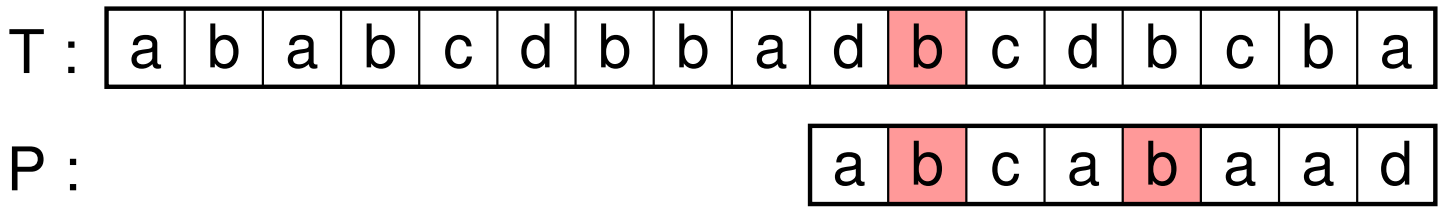
- Matches with a single *frequent* character can be found using cross-correlations in $O(n \log m)$ time.
- Each *infrequent* text character can't match very many times.

How is it done offline? *ALP 2000*

Def: A character is *frequent* if it occurs at least $\sqrt{k}$ times in P.

Case 1: Fewer than $\sqrt{k}$ *frequent* characters in P.

- Here we count *matches* rather than *mismatches*.



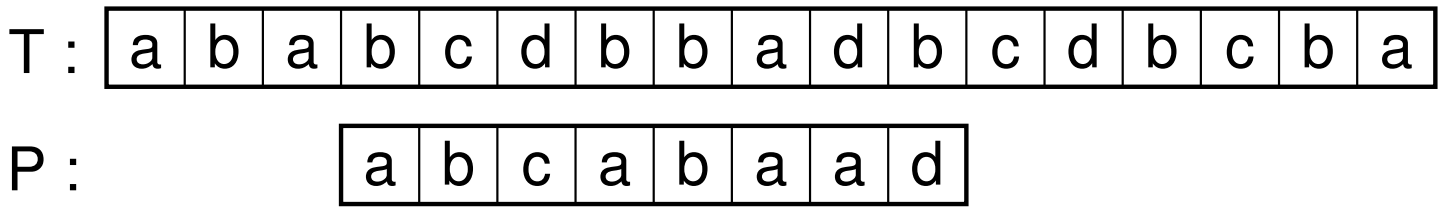
- Matches with a single *frequent* character can be found using cross-correlations in $O(n \log m)$ time.
- Each *infrequent* text character can't match very many times.

How is it done offline? *ALP 2000*

Def: A character is *frequent* if it occurs at least $\sqrt{k}$ times in P.

Case 1: Fewer than $\sqrt{k}$ *frequent* characters in P.

- Here we count *matches* rather than *mismatches*.



- Matches with a single *frequent* character can be found using cross-correlations in $O(n \log m)$ time.
- Each *infrequent* text character can't match very many times.

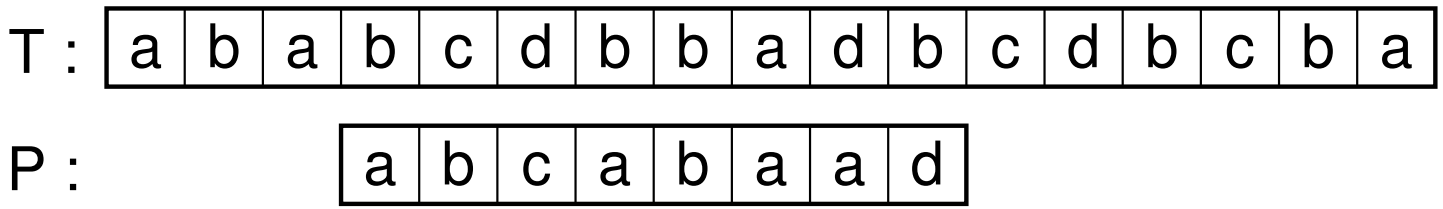
This method gives a time complexity of $O(n\sqrt{k} \log m + n\sqrt{k})$

How is it done offline? *ALP 2000*

Def: A character is *frequent* if it occurs at least $\sqrt{k}$ times in P.

Case 1: Fewer than $\sqrt{k}$ *frequent* characters in P.

- Here we count *matches* rather than *mismatches*.



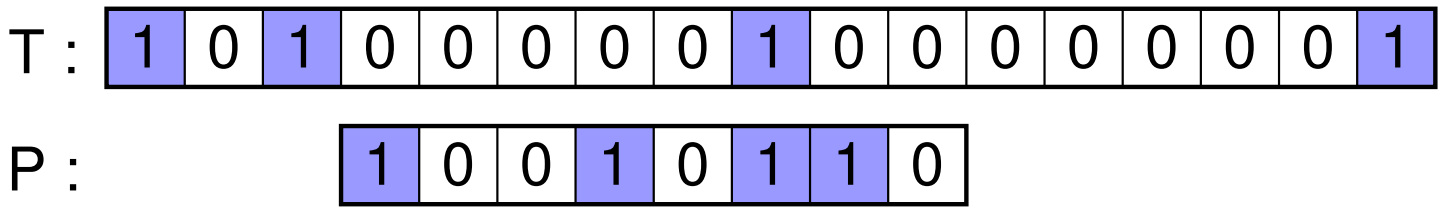
- Matches with a single *frequent* character can be found using cross-correlations in $O(n \log m)$ time.
- Each *infrequent* text character can't match very many times.

This method gives a time complexity of $O(n\sqrt{k \log m})$

How do we do cross-correlations *online*?

The cross-correlation between arrays T and P is defined by:

$$(T \otimes P)[i] = \sum_{j=0}^{m-1} T[i+j]P[j]$$



- Offline: $O(n \log m)$ total time (via FFTs).
- Pseudo-realtime: $O(\log^2 m)$ time per character[2].

[2] via the blackbox method of CEPP 2008

How is it done online? *C. S. 2010*

Def: A character is *frequent* if it occurs at least $\sqrt{k}$ times in P.

Case 1: Fewer than $\sqrt{k}$ *frequent* characters in P.

- Here we count *matches* rather than *mismatches*.

T :	a	b	a	b	c	d	b	b	a	d	b	c	d	b	c	b	a

P :			a	b	c	a	b	a	a	d

- Matches with a single *frequent* character can be found using **PsR cross-correlations** in $O(\log^2 m)$ **time per character**.
- Each *infrequent* text character can't match very many times.

This gives a time complexity of $O(\sqrt{k} \log m + \log^2 m)$ per character.

How is it done offline? *ALP 2000*

Def: A character is *frequent* if it occurs at least $\sqrt{k}$ times in P.

Case 2: At least $\sqrt{k}$ *frequent* characters in P.

How is it done offline? *ALP 2000*

Def: A character is *frequent* if it occurs at least $\sqrt{k}$ times in P.

Case 2: At least $\sqrt{k}$ *frequent* characters in P.

- Filter the text to rule out alignments which must have more than k mismatches. The ALP filter leaves only $n/\sqrt{k}$ locations.

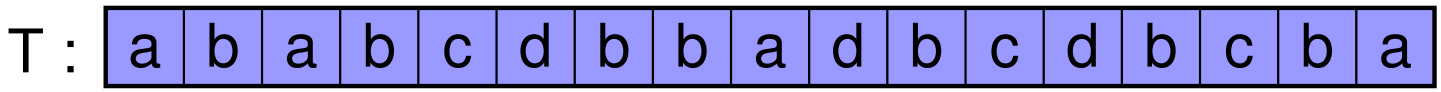
- Check each location using $(k+1)$ *Longest Common Extension* queries. Offline we can preprocess T, P for $O(1)$ time queries.

How is it done offline? *ALP 2000*

Def: A character is *frequent* if it occurs at least $\sqrt{k}$ times in P.

Case 2: At least $\sqrt{k}$ *frequent* characters in P.

- Filter the text to rule out alignments which must have more than k mismatches. The ALP filter leaves only $n/\sqrt{k}$ locations.



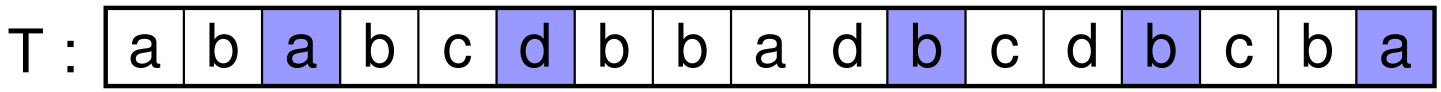
- Check each location using $(k+1)$ *Longest Common Extension* queries. Offline we can preprocess T, P for $O(1)$ time queries.

How is it done offline? *ALP 2000*

Def: A character is *frequent* if it occurs at least $\sqrt{k}$ times in P.

Case 2: At least $\sqrt{k}$ *frequent* characters in P.

- Filter the text to rule out alignments which must have more than k mismatches. The ALP filter leaves only $n/\sqrt{k}$ locations.



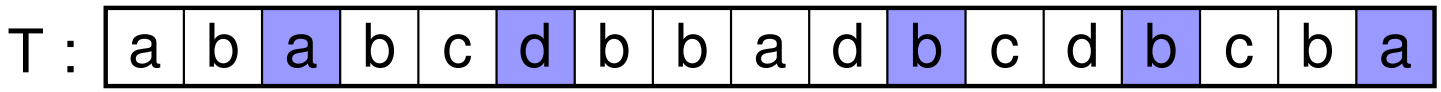
- Check each location using $(k+1)$ *Longest Common Extension* queries. Offline we can preprocess T, P for $O(1)$ time queries.

How is it done offline? *ALP 2000*

Def: A character is *frequent* if it occurs at least $\sqrt{k}$ times in P.

Case 2: At least $\sqrt{k}$ *frequent* characters in P.

- Filter the text to rule out alignments which must have more than k mismatches. The ALP filter leaves only $n/\sqrt{k}$ locations.



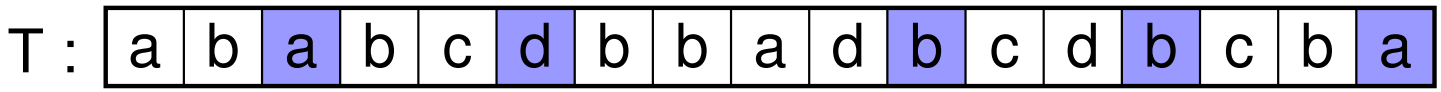
- Check each location using $(k+1)$ *Longest Common Extension* queries. Offline we can preprocess T, P for $O(1)$ time queries.

How is it done offline? *ALP 2000*

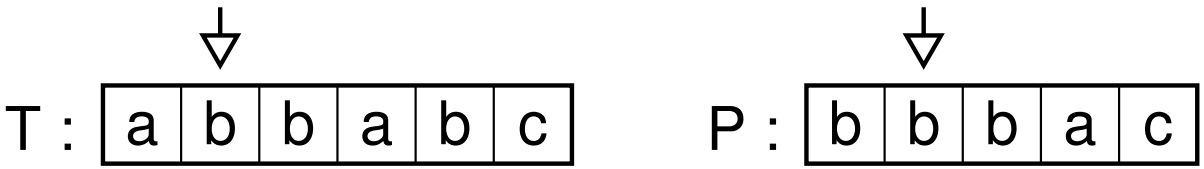
Def: A character is *frequent* if it occurs at least $\sqrt{k}$ times in P.

Case 2: At least $\sqrt{k}$ *frequent* characters in P.

- Filter the text to rule out alignments which must have more than k mismatches. The ALP filter leaves only $n/\sqrt{k}$ locations.



- Check each location using $(k+1)$ *Longest Common Extension* queries. Offline we can preprocess T, P for $O(1)$ time queries.

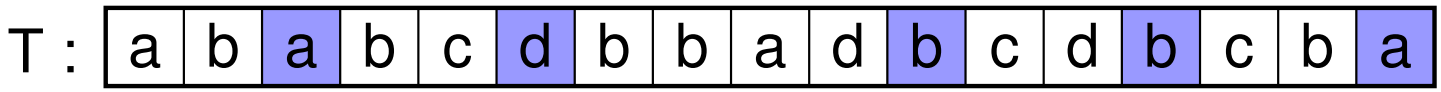


How is it done offline? *ALP 2000*

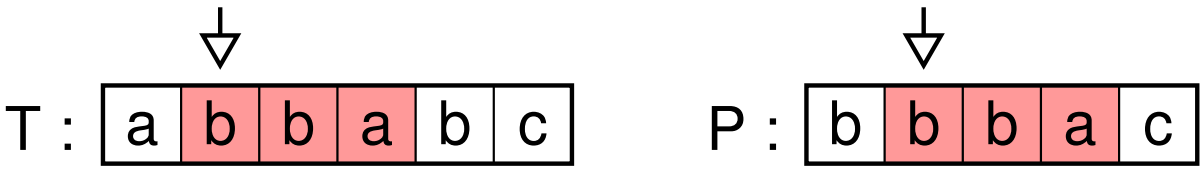
Def: A character is *frequent* if it occurs at least $\sqrt{k}$ times in P.

Case 2: At least $\sqrt{k}$ *frequent* characters in P.

- Filter the text to rule out alignments which must have more than k mismatches. The ALP filter leaves only $n/\sqrt{k}$ locations.

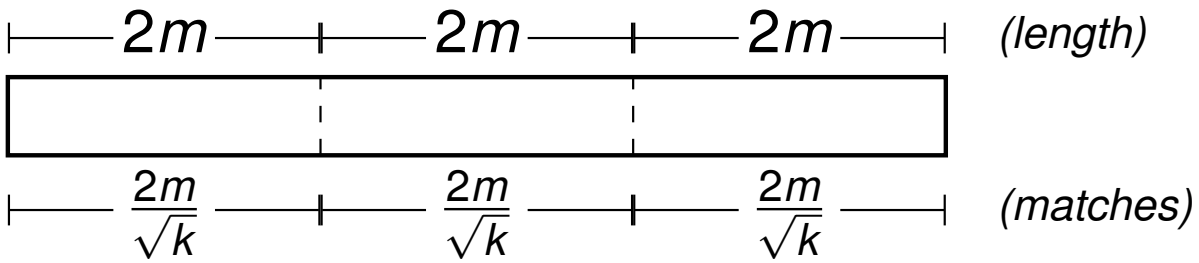


- Check each location using $(k+1)$ *Longest Common Extension* queries. Offline we can preprocess T, P for $O(1)$ time queries.



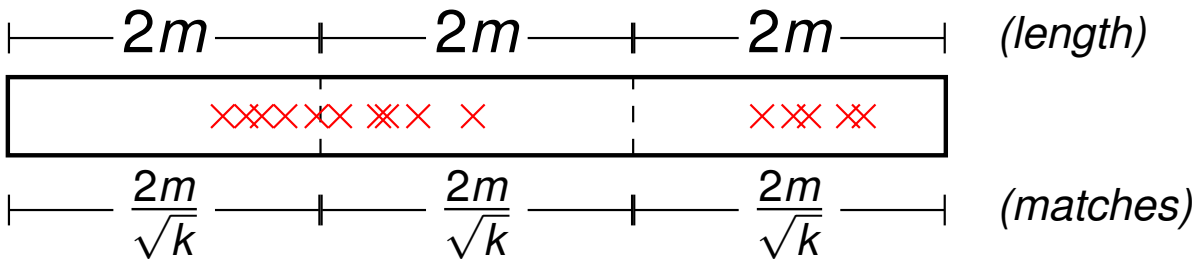
What is wrong with the original filtering?

Original filter: At most $2m/\sqrt{k}$ matches in $2m$ text locations.



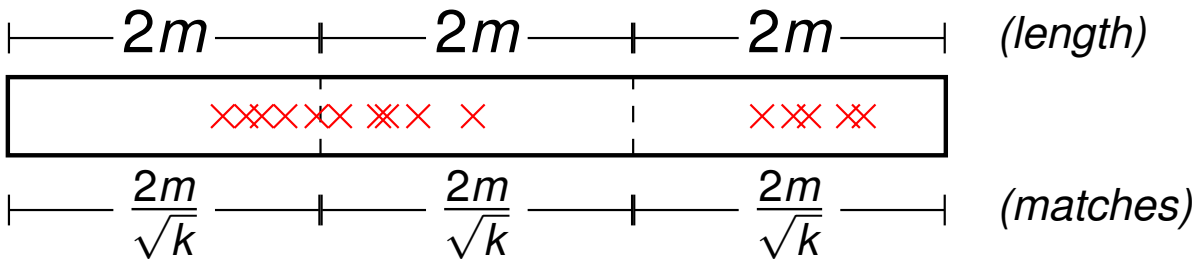
What is wrong with the original filtering?

Original filter: At most $2m/\sqrt{k}$ matches in $2m$ text locations.

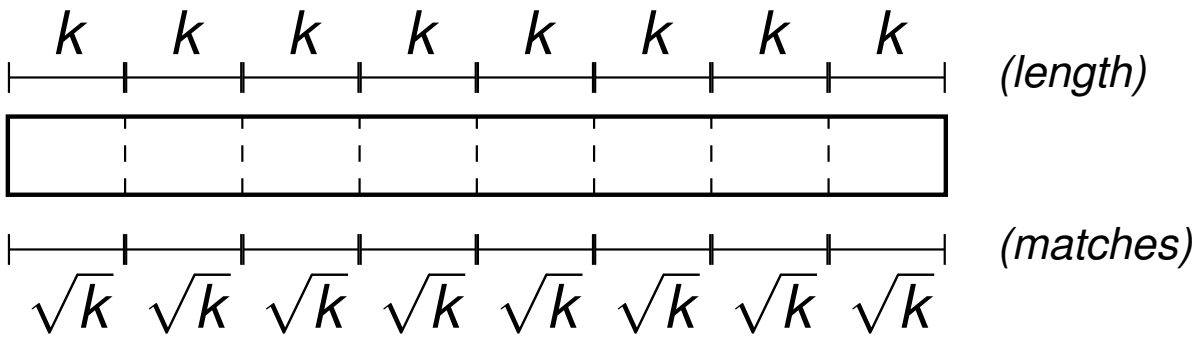


What is wrong with the original filtering?

Original filter: At most $2m/\sqrt{k}$ matches in $2m$ text locations.

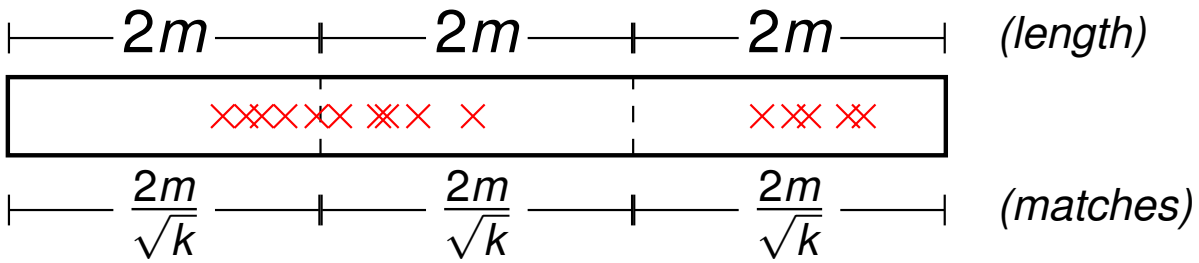


Improved filter: At most $\sqrt{k}$ matches in k text locations.

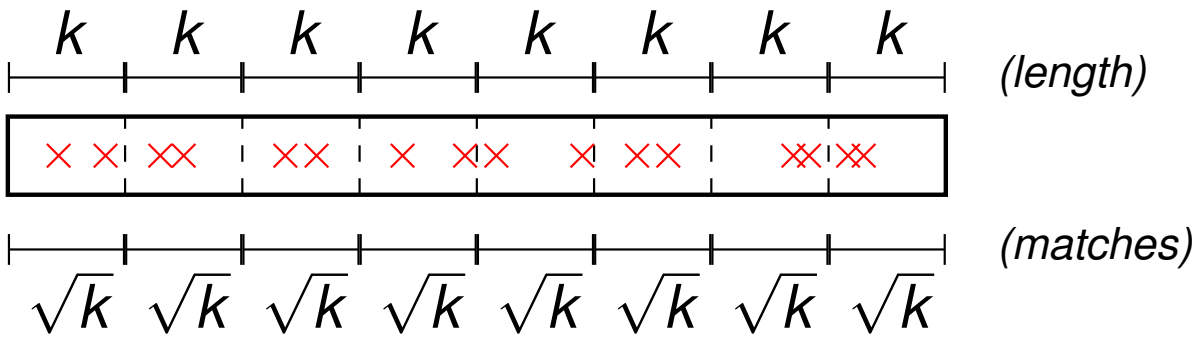


What is wrong with the original filtering?

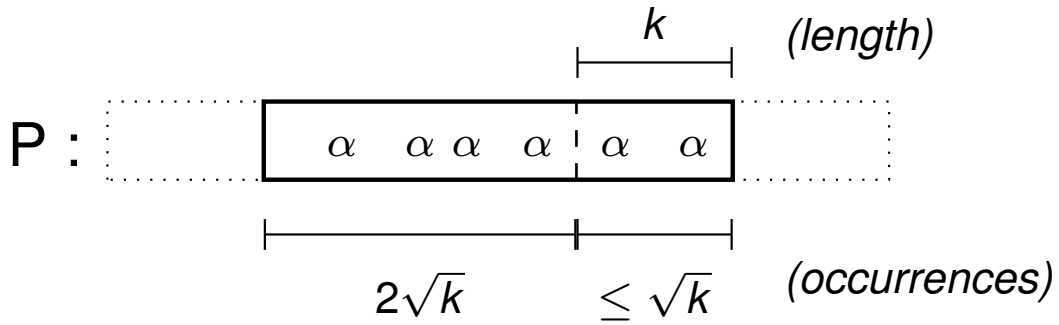
Original filter: At most $2m/\sqrt{k}$ matches in $2m$ text locations.



Improved filter: At most $\sqrt{k}$ matches in k text locations.

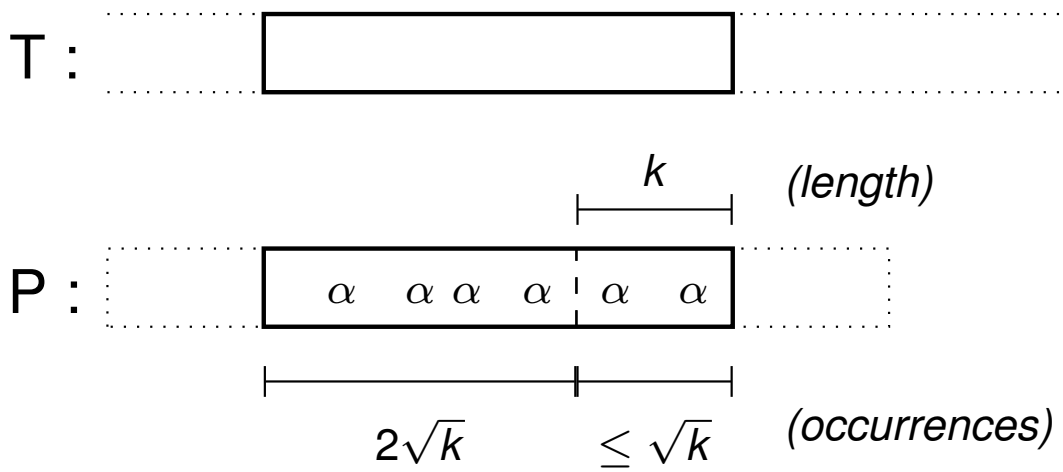


How does the improved filter work?



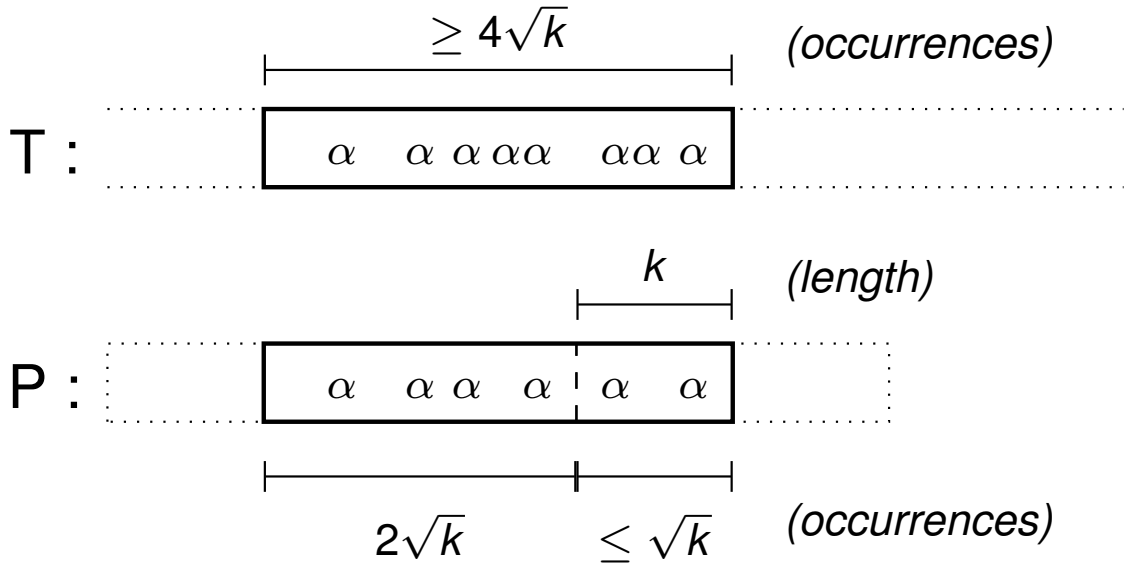
We call this an $\sqrt{k}$-balanced substring (for symbol $\alpha \in \Sigma$).

How does the improved filter work?



We call this an $\sqrt{k}$-balanced substring (for symbol $\alpha \in \Sigma$).

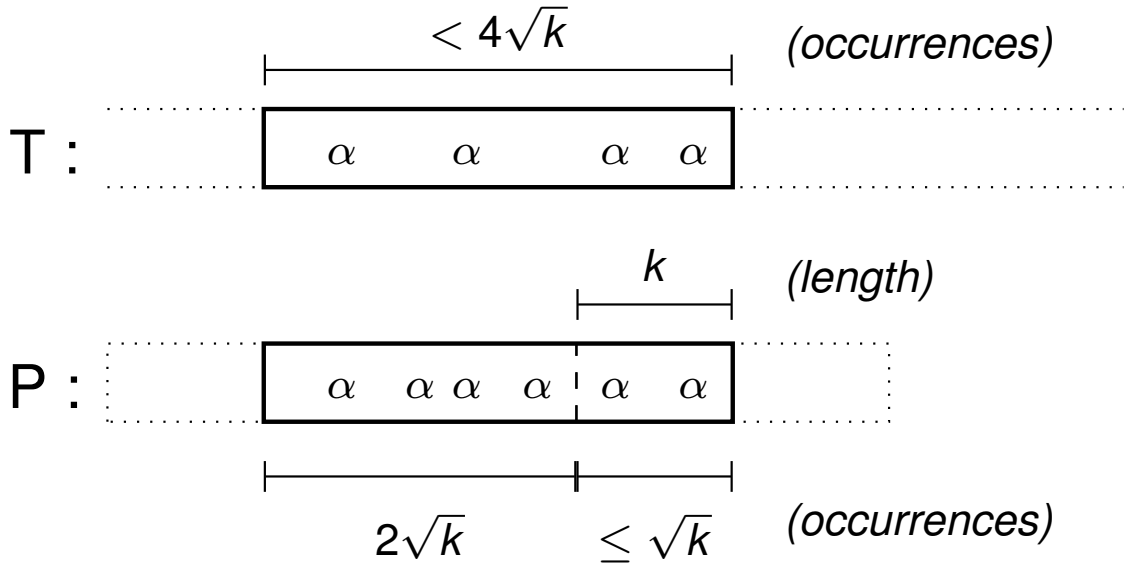
How does the improved filter work?



We call this an $\sqrt{k}$-balanced substring (for symbol $\alpha \in \Sigma$).

1. If the aligned substring of T has more than $4\sqrt{k}$ occurrences of α we have found at least $\sqrt{k}$ mismatches at this alignment.

How does the improved filter work?



We call this an $\sqrt{k}$-balanced substring (for symbol $\alpha \in \Sigma$).

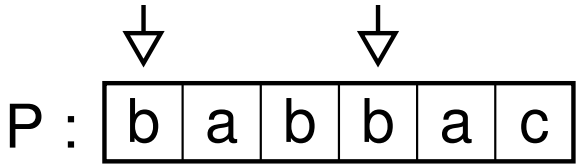
1. If the aligned substring of T has more than $4\sqrt{k}$ occurrences of α we have found at least $\sqrt{k}$ mismatches at this alignment.
2. Otherwise, we have found $2\sqrt{k}$ pattern positions which match at most $4\sqrt{k}$ times in the next k alignments.

What is wrong with the original *LCE* queries?

Offline, complex text preprocessing gives constant *LCE* query times. Instead we preprocess the pattern for *self-LCE* queries. . . .

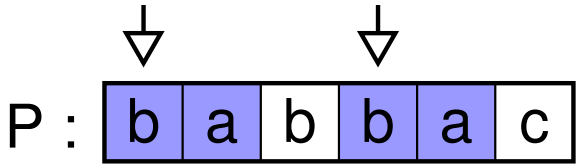
What is wrong with the original *LCE* queries?

Offline, complex text preprocessing gives constant *LCE* query times. Instead we preprocess the pattern for *self-LCE* queries. . . .



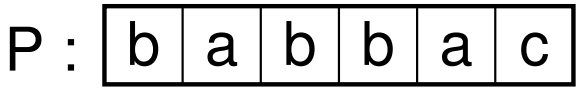
What is wrong with the original *LCE* queries?

Offline, complex text preprocessing gives constant *LCE* query times. Instead we preprocess the pattern for *self-LCE* queries. . . .



What is wrong with the original *LCE* queries?

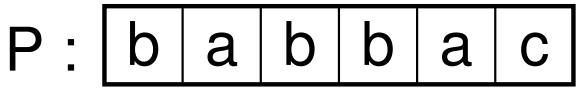
Offline, complex text preprocessing gives constant *LCE* query times. Instead we preprocess the pattern for *self-LCE* queries. . . .



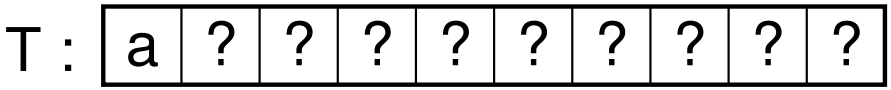
Main Idea: Encode the arriving text in terms of the pattern and use *self-LCE* queries on the pattern to perform pattern/text *LCE* queries.

What is wrong with the original *LCE* queries?

Offline, complex text preprocessing gives constant *LCE* query times. Instead we preprocess the pattern for *self-LCE* queries. . . .

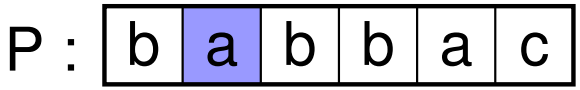


Main Idea: Encode the arriving text in terms of the pattern and use *self-LCE* queries on the pattern to perform pattern/text *LCE* queries.

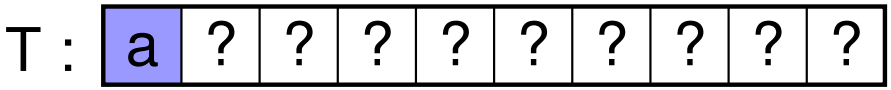


What is wrong with the original *LCE* queries?

Offline, complex text preprocessing gives constant *LCE* query times. Instead we preprocess the pattern for *self-LCE* queries. . . .

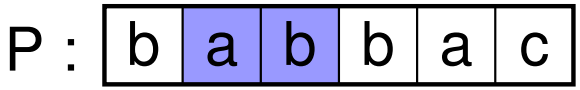


Main Idea: Encode the arriving text in terms of the pattern and use *self-LCE* queries on the pattern to perform pattern/text *LCE* queries.

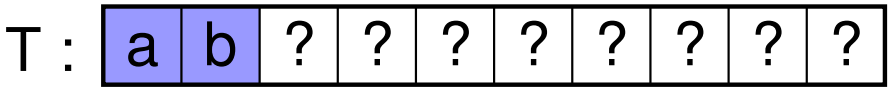


What is wrong with the original *LCE* queries?

Offline, complex text preprocessing gives constant *LCE* query times. Instead we preprocess the pattern for *self-LCE* queries. . . .

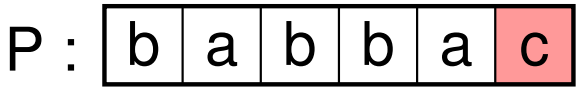


Main Idea: Encode the arriving text in terms of the pattern and use *self-LCE* queries on the pattern to perform pattern/text *LCE* queries.

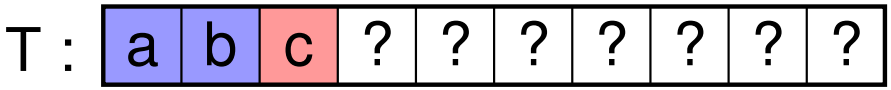


What is wrong with the original *LCE* queries?

Offline, complex text preprocessing gives constant *LCE* query times. Instead we preprocess the pattern for *self-LCE* queries. . . .

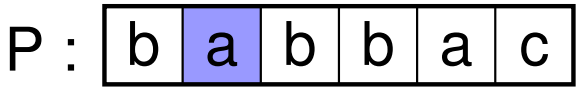


Main Idea: Encode the arriving text in terms of the pattern and use *self-LCE* queries on the pattern to perform pattern/text *LCE* queries.

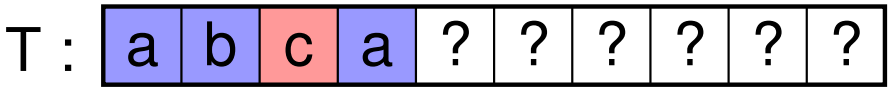


What is wrong with the original *LCE* queries?

Offline, complex text preprocessing gives constant *LCE* query times. Instead we preprocess the pattern for *self-LCE* queries. . . .

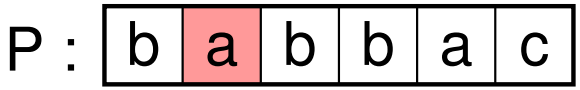


Main Idea: Encode the arriving text in terms of the pattern and use *self-LCE* queries on the pattern to perform pattern/text *LCE* queries.

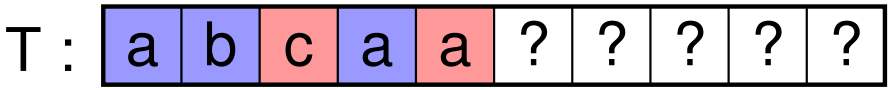


What is wrong with the original *LCE* queries?

Offline, complex text preprocessing gives constant *LCE* query times. Instead we preprocess the pattern for *self-LCE* queries. . . .

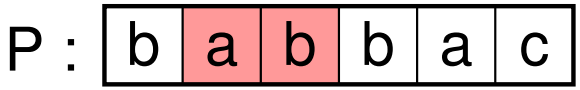


Main Idea: Encode the arriving text in terms of the pattern and use *self-LCE* queries on the pattern to perform pattern/text *LCE* queries.

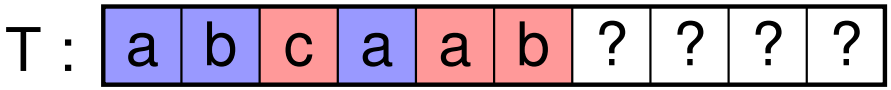


What is wrong with the original *LCE* queries?

Offline, complex text preprocessing gives constant *LCE* query times. Instead we preprocess the pattern for *self-LCE* queries. . . .

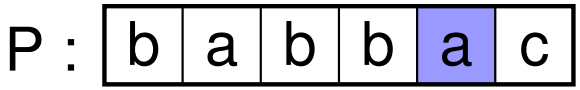


Main Idea: Encode the arriving text in terms of the pattern and use *self-LCE* queries on the pattern to perform pattern/text *LCE* queries.

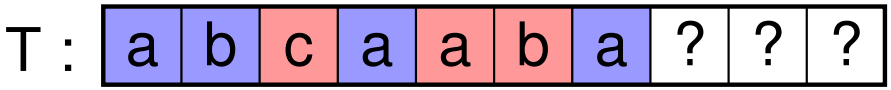


What is wrong with the original *LCE* queries?

Offline, complex text preprocessing gives constant *LCE* query times. Instead we preprocess the pattern for *self-LCE* queries. . . .

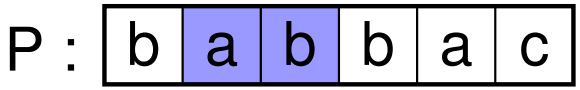


Main Idea: Encode the arriving text in terms of the pattern and use *self-LCE* queries on the pattern to perform pattern/text *LCE* queries.

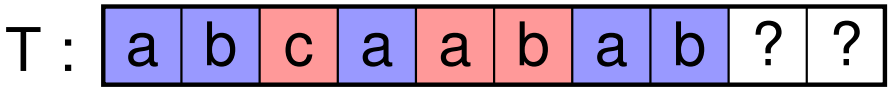


What is wrong with the original *LCE* queries?

Offline, complex text preprocessing gives constant *LCE* query times. Instead we preprocess the pattern for *self-LCE* queries. . . .

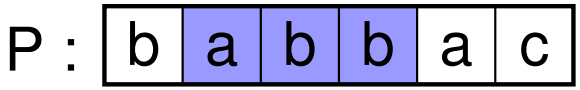


Main Idea: Encode the arriving text in terms of the pattern and use *self-LCE* queries on the pattern to perform pattern/text *LCE* queries.

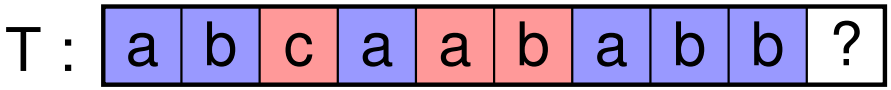


What is wrong with the original *LCE* queries?

Offline, complex text preprocessing gives constant *LCE* query times. Instead we preprocess the pattern for *self-LCE* queries. . . .

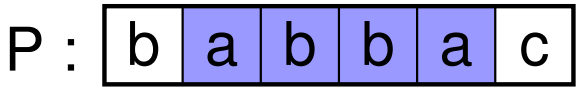


Main Idea: Encode the arriving text in terms of the pattern and use *self-LCE* queries on the pattern to perform pattern/text *LCE* queries.

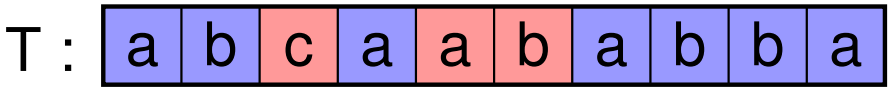


What is wrong with the original *LCE* queries?

Offline, complex text preprocessing gives constant *LCE* query times. Instead we preprocess the pattern for *self-LCE* queries. . . .

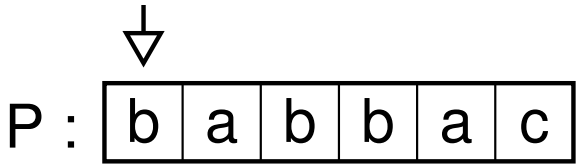


Main Idea: Encode the arriving text in terms of the pattern and use *self-LCE* queries on the pattern to perform pattern/text *LCE* queries.

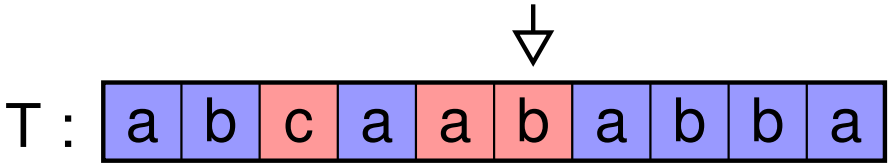


What is wrong with the original *LCE* queries?

Offline, complex text preprocessing gives constant *LCE* query times. Instead we preprocess the pattern for *self-LCE* queries. . . .



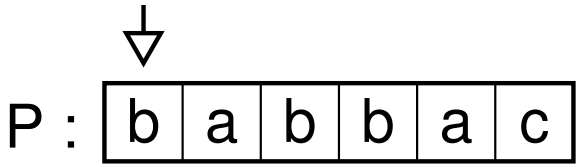
Main Idea: Encode the arriving text in terms of the pattern and use *self-LCE* queries on the pattern to perform pattern/text *LCE* queries.



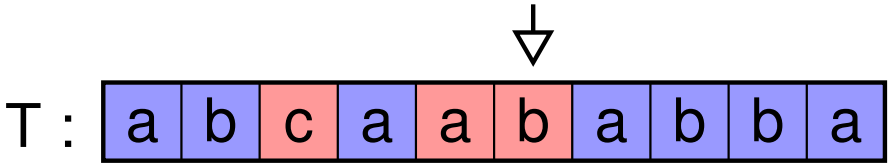
- How many self queries do we need to do?

What is wrong with the original *LCE* queries?

Offline, complex text preprocessing gives constant *LCE* query times. Instead we preprocess the pattern for *self-LCE* queries. . . .



Main Idea: Encode the arriving text in terms of the pattern and use *self-LCE* queries on the pattern to perform pattern/text *LCE* queries.



- How many self queries do we need to do? *At most three!*

Putting it all together

The remainder is conceptually simple but the details are *fiddly*:

Conceptually

- Use the improved filter on each k-length text substring online.
- Check the $\sqrt{k}$ potential matches using the online *LCE* queries.

However, we need to be very careful about the scheduling details to ensure that the work is done in time.

This results in a time complexity per text character of

$$O(\sqrt{k}\log m + \log^2 m)$$

Putting it all together

The remainder is conceptually simple but the details are *fiddly*:

Conceptually

- Use the improved filter on each k-length text substring online.
- Check the $\sqrt{k}$ potential matches using the online *LCE* queries.

However, we need to be very careful about the scheduling details to ensure that the work is done in time.

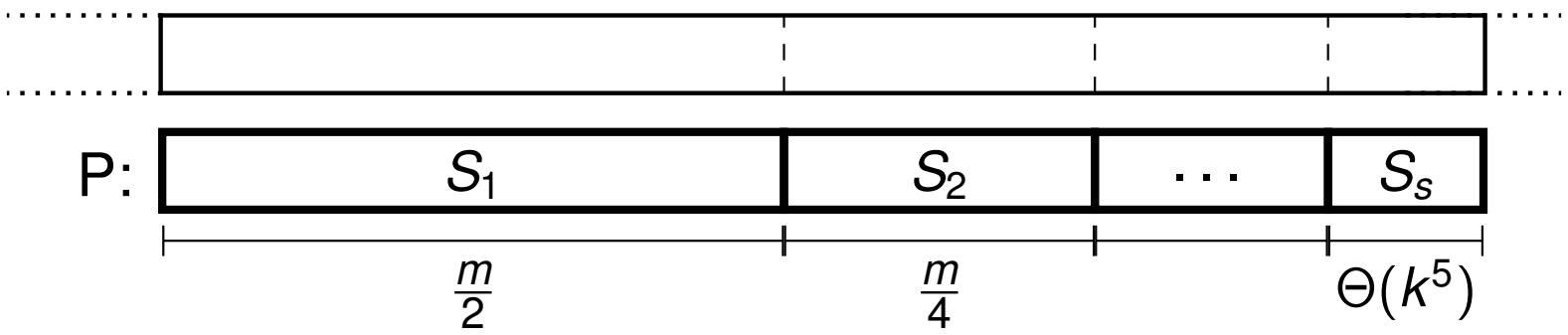
This results in a time complexity per text character of

$$O(\sqrt{k}\log m + \log^2 m)$$

Didn't you say additive $\log m$*?*

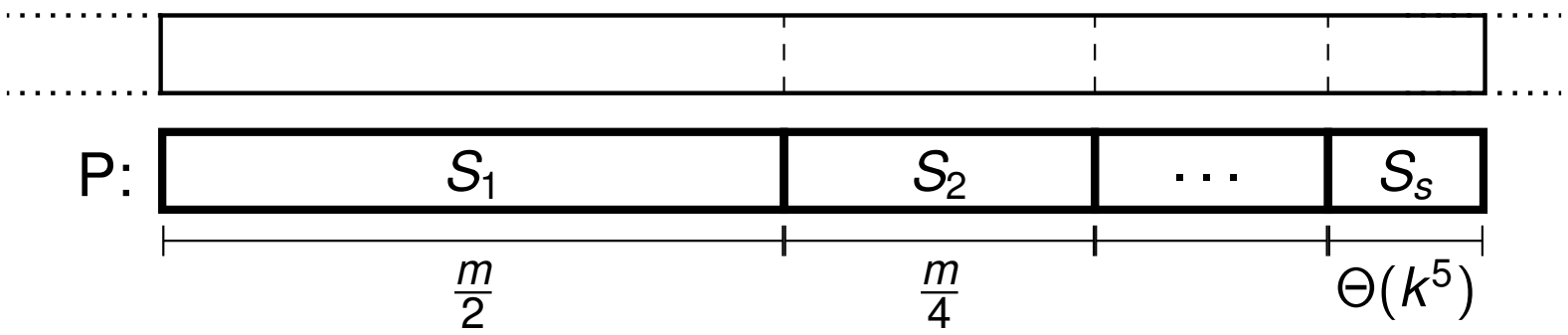
How do we get an additive log m?

- Split the pattern into $s \in O(\log m)$ consecutive subpatterns where each subpattern is half the length of the previous.
- We set s so that $k^5/2 \leq |S_s| < k^5$.



How do we get an additive log m?

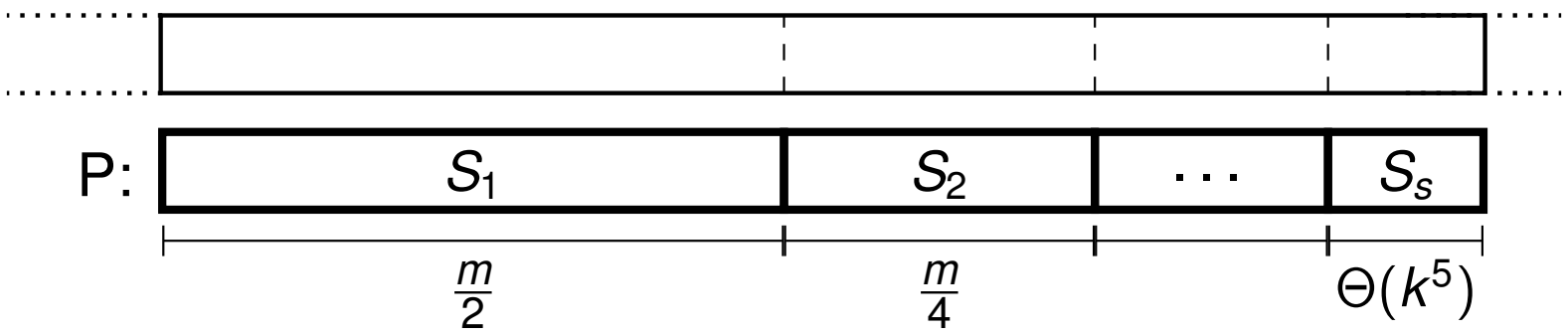
- Split the pattern into $s \in O(\log m)$ consecutive subpatterns where each subpattern is half the length of the previous.
- We set s so that $k^5/2 \leq |S_s| < k^5$.



- Use the $O(\sqrt{k}\log m + \log^2 m)$ algorithm for the final subpattern.

How do we get an additive log m?

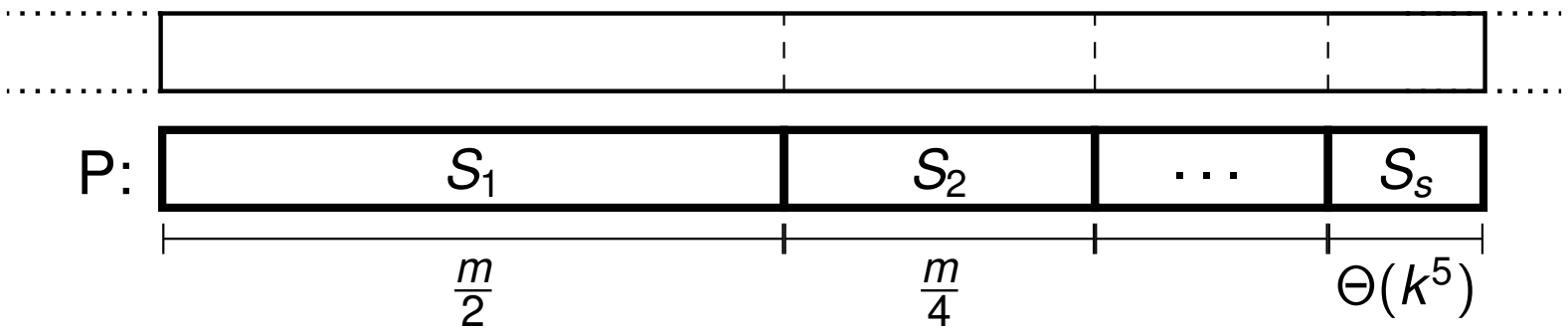
- Split the pattern into $s \in O(\log m)$ consecutive subpatterns where each subpattern is half the length of the previous.
- We set s so that $k^5/2 \leq |S_s| < k^5$.



- Use the $O(\sqrt{k}\log m + \log^2 m)$ algorithm for the final subpattern.
- As $|S_s| \in \Theta(k^5)$ we have $O(\sqrt{k}\log k)$ time per character.

How do we get an additive log m?

- What about mismatches with subpatterns $S_1, S_2, \ldots, S_{s-1}$?
- Each of these subpatterns has length greater than k^5.



- For the case where $m >> k$ there is an offline algorithm[3] with time complexity $O(n + nk^4 \log k/m)$.
- Applied to one of our subpatterns (offline) this would be $O(n)$.
- Using the *black box method* we obtain $O(1)$ time per character.

[3] Amir, Lewenstein and Porat. SODA 2000

Conclusions

k-mismatch

- Summing across all subpatterns we obtain a time complexity of

$$O(\sqrt{k}\log k + \log m) \text{ per character (and } O(m) \text{ space).}$$

- This is tantalisingly close to the $O(n\sqrt{k\log k})$ offline solution.

k-differences

- By combining our realtime *LCE* processing with an incremental k-difference algorithm[4] we obtain a time complexity of

$$O(k) \text{ time per character (and } O(m) \text{ space).}$$

- This is matches the $O(nk)$ offline solution.

[4] Landau, Myers and Schmidt, SIAM J. Comput 1998