String Matching with Mismatches

Some slides are stolen from Moshe Lewenstein (Bar Ilan University)

String Matching with Mismatches

Landau – Vishkin	**1986**
Galil – Giancarlo	**1986**
Abrahamson	**1987**
Amir - Lewenstein - Porat	**2000**

Approximate String Matching

problem: Find all text locations where distance from pattern is sufficiently small.

distance metric: **HAMMING DISTANCE**

Let $S = s_1 s_2 \ldots s_m$
$R = r_1 r_2 \ldots r_m$

Ham(S,R) = The number of locations j where $s_j \neq r_j$

Example: S = ABCABC
R = ABBAAC

Ham(S,R) = 2

Problem 1: Counting mismatches

Input: $T = t_1 \quad . . . \quad t_n$

$P = p_1 \dots p_m$

Output: For each i in T

$Ham(P, t_i t_{i+1} \dots t_{i+m-1})$

Example:

P = A B B A A C

T = A B C A A B C A C...

Counting mismatches

Input: $T = t_1 \quad \ldots \quad t_n$

$P = p_1 \ldots p_m$

Output: For each i in T

$Ham(P, t_i t_{i+1} \ldots t_{i+m-1})$

Example:

P = A B B A A C

T = A B C A A B C A C…

2

$Ham(P,T_1) = 2$

Counting mismatches

Input: $T = t_1 \quad \ldots \quad t_n$

$P = p_1 \ldots p_m$

Output: For each i in T

$Ham(P, t_i t_{i+1} \ldots t_{i+m-1})$

Example:

P =		A	B	B	A	A	C		
T =	A	B	C	A	A	B	C	A	C...
	2, 4								

$Ham(P, T_2) = 4$

Counting mismatches

Input: $T = t_1 \quad . . . \quad t_n$

$P = p_1 \dots p_m$

Output: For each i in T

$Ham(P, t_i t_{i+1} \dots t_{i+m-1})$

Example:

P =			A	B	B	A	A	C		
T =	A	B	C	A	A	B	C	A	C…	

2, 4, 6

$Ham(P, T_3) = 6$

Counting mismatches

Input: $T = t_1 \ \ldots \ t_n$

$P = p_1 \ldots p_m$

Output: For each i in T

$Ham(P, t_i t_{i+1} \ldots t_{i+m-1})$

Example:

P = A B B A A C

T = A B C A A B C A C...

2, 4, 6, 2

$Ham(P, T_4) = 2$

Counting mismatches

Input: $T = t_1 \ \ldots \ t_n$

$P = p_1 \ldots p_m$

Output: For each i in T

$Ham(P, t_i t_{i+1} \ldots t_{i+m-1})$

Example:

P = A B B A A C

T = A B C A A B C A C...

2, 4, 6, 2, ...

Problem 2: k-mismatches

Input: $T = t_1 \quad . . . \quad t_n$

$P = p_1 \ldots p_m$

Output: Every i where

$Ham(P, t_i t_{i+1} \ldots t_{i+m-1}) \leq k$

Example: $k = 2$

P = A B B A A C

T = A B C A A B C A C...

2, 4, 6, 2, ...

Problem 2: k-mismatches

Input: $T = t_1 \quad . . . \quad t_n$

$P = p_1 \ldots p_m$

Output: Every i where

$Ham(P, t_i t_{i+1} \ldots t_{i+m-1}) \leq k$h

Example: $k = 2$

P = A B B A A C

T = A B C A A B C A C...

~~2, 4, 6, 2, ...~~

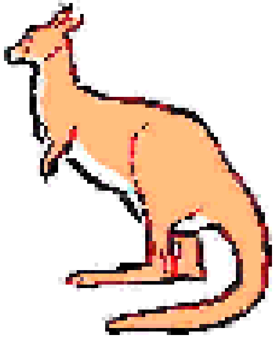
1, 0, 0, 1,

Naïve Algorithm
(for counting mismatches or k-mismatches problem)

- Goto each location of text and compute hamming distance of P and T_i

Running Time: O(nm) $n = |T|$, $m = |P|$

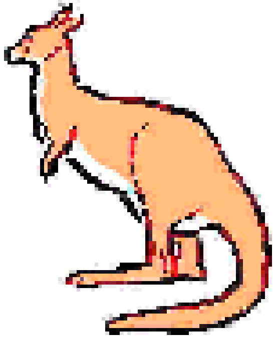
The Kangaroo Method
(for k-mismatches)



Landau – Vishkin 1986

Galil – Giancarlo 1986

The Kangaroo Method (for k-mismatches)



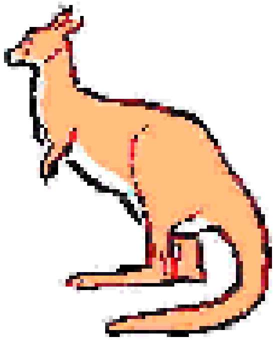
-Create suffix tree (+ lca) for: s = P#T

-Check P at each location i of T by kangrooing

Example:

P = A B A B A A B A C A B
T = A B B A C A B A B A B C A B B C A B C A ...
i

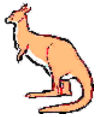
The Kangaroo Method
(for k-mismatches)



- Create suffix tree for: s = P#T

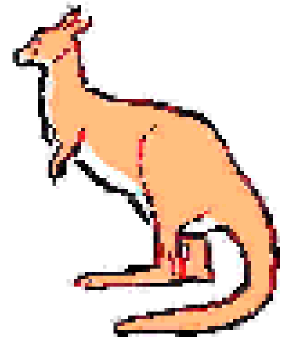
-Check P at each location i of T by kangrooing

Example:



P = A B A B A A B A C A B
T = A B B A C A B A B A B C A B B C A B C A ...
i

The Kangaroo Method (for k-mismatches)



- Create suffix tree for: s = P#T

-Check P at each location i of T by kangrooing

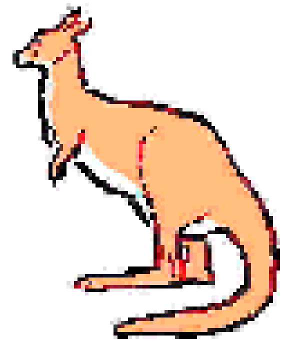
Example:

P = A B A B A A B A C A B

T = A B B A C A B A B A B C A B B C A B C A ...

i

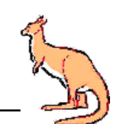
The Kangaroo Method (for k-mismatches)



- Create suffix tree for: s = P#T

-Check P at each location i of T by kangrooing

Example:

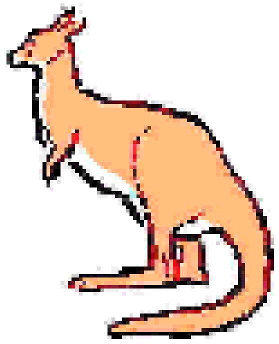


P = A B A B A A B A C A B

T = A B B A C A B A B A B C A B B C A B C A ...

i ↑

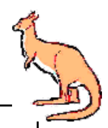
The Kangaroo Method
(for k-mismatches)



- Create suffix tree for: s = P#T

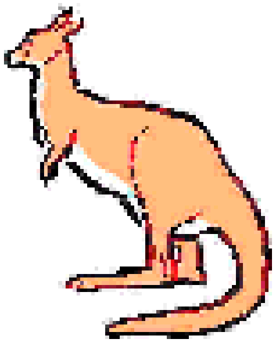
-Check P at each location i of T by kangrooing

Example:



P = A B A B A A B A C A B
T = A B B A C A B A B A B C A B B C A B C A ...
i

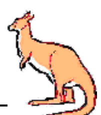
The Kangaroo Method
(for k-mismatches)



- Create suffix tree for: s = P#T

-Check P at each location i of T by kangrooing

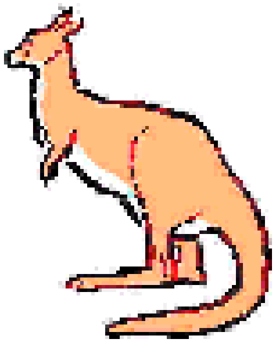
Example:



P = A B A B A A B A C A B
T = A B B A C A B A B A B C A B B C A B C A …

i

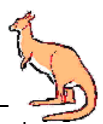
The Kangaroo Method
(for k-mismatches)



- Create suffix tree for: s = P#T

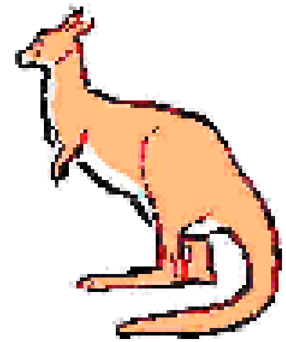
-Check P at each location i of T by kangrooing

Example:



```
P  =               A B A B A A B A C A B
T  =  A B B A C A B A B A B C A B B C A B C A ...
                    i
```

The Kangaroo Method (for k-mismatches)



- Create suffix tree for: s = P#T

- Do up to k LCP queries for every text location

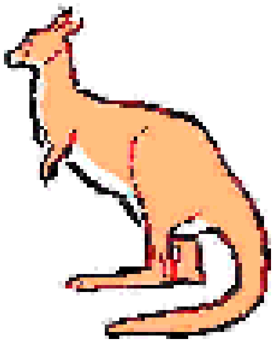
Example:

P = A B A B A A B A C A B

T = A B B A C A B A B A B C A B B C A B C A ...

i

The Kangaroo Method (for k-mismatches)



Preprocess:

Build suffix tree of both P and T - O(n+m) time
LCA preprocessing - O(n+m) time

Check P at given text location

Kangroo jump till next mismatch
- O(k) time

Overall time: O(nk)

How do we do counting in less than O(nm) ?

Lets start with binary strings

P =

0	1	0	1	1	0	1	1

T =

...	0	1	1	1	1	1	1	0	0	0	0	1	0	1	1	...

We can count matches using FFT in O(nlog(m)) time

And if the strings are not binary ?

P =

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

T =

...	a	b	b	b	c	c	c	a	a	a	a	b	a	c	b	...

P =

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

a-mask

a	b	**a**	c	c	**a**	c	b	**a**	c	**a**	b	**a**	c	c

T =

...	a	b	b	b	c	c	c	a	a	a	a	b	a	c	b	...

P =		a	b	a	c	c	a	c	b	a	c	a	b	a	c	c	
a-mask		a	b	a	c	c	a	c	b	a	c	a	b	a	c	c	
P_a		1	0	1	0	0	1	0	0	1	0	1	0	1	0	0	
T =	...	a	b	b	b	c	c	c	a	a	a	a	b	a	c	b	...

P =	a	b	a	c	c	a	c	b	a	c	a	b	a	c	c
Pa	1	0	1	0	0	1	0	0	1	0	1	0	1	0	0

T =	...	a	b	b	b	c	c	c	a	a	a	a	b	a	c	b	...

P =		a	b	a	c	c	a	c	b	a	c	a	b	a	c	c	
Pa		1	0	1	0	0	1	0	0	1	0	1	0	1	0	0	
T =	...	a	b	b	b	c	c	c	a	a	a	a	b	a	c	b	...
not-a mask	...	a	b	b	b	c	c	c	a	a	a	a	b	a	c	b	...

P =		a	b	a	c	c	a	c	b	a	c	a	b	a	c	c	
Pa		1	0	1	0	0	1	0	0	1	0	1	0	1	0	0	
T =	...	a	b	b	b	c	c	c	a	a	a	a	b	a	c	b	...
not-a mask	...	a	b	b	b	c	c	c	a	a	a	a	b	a	c	b	...
$T_{not\ a}$		0	1	1	1	1	1	1	0	0	0	0	1	0	1	1	

P =		a	b	a	c	c	a	c	b	a	c	a	b	a	c	c	
P_a		1	0	1	0	0	1	0	0	1	0	1	0	1	0	0	
T =	...	a	b	b	b	c	c	c	a	a	a	a	b	a	c	b	...
$T_{not\ a}$		0	1	1	1	1	1	1	0	0	0	0	1	0	1	1	

P =	a	b	a	c	c	a	c	b	a	c	a	b	a	c	c
P_a	1	0	1	0	0	1	0	0	1	0	1	0	1	0	0

T =	...	a	b	b	b	c	c	c	a	a	a	a	b	a	c	b	...
$T_{not\ a}$		0	1	1	1	1	1	1	0	0	0	0	1	0	1	1	

Multiply P_a and $T_{(not\ a)}$ to count mismatches using "a"

P_a		1	0	1	0	0	1	0	0	1	0	1	0	1	0	0	
$T_{not\ a}$	...	0	1	1	1	1	1	1	0	0	0	0	1	0	1	1	...

P =		a	b	a	c	c	a	c	b	a	c	a	b	a	c	c	
P_a		1	0	1	0	0	1	0	0	1	0	1	0	1	0	0	
T =	...	a	b	b	b	c	c	c	a	a	a	a	b	a	c	b	...
$T_{not\ a}$		0	1	1	1	1	1	1	0	0	0	0	1	0	1	1	

Multiply P_a and $T_{not\ a}$ to count mismatches using a

P_a		1	0	1	0	0	1	0	0	1	0	1	0	1	0	0	
$T_{not\ a}$	...	0	1	1	1	1	1	1	0	0	0	0	1	0	1	1	...

Boolean Convolutions (FFT) Method

Boolean Convolutions (FFT) Method

Running Time: One boolean convolution - $O(n \log m)$ time

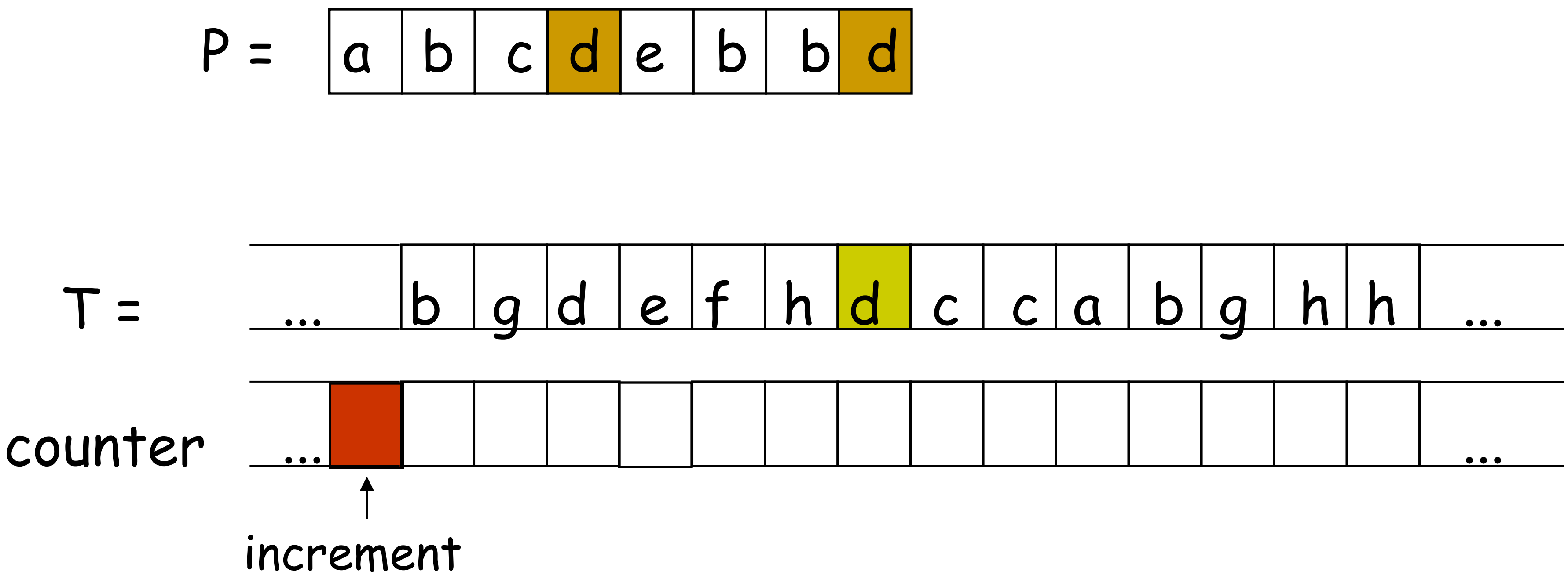
$\Rightarrow$ # of matches of all symbols - $O(n|\Sigma| \log m)$ time

How do we do counting in less than O(nm) ?

Lets count matches rather than mismatches

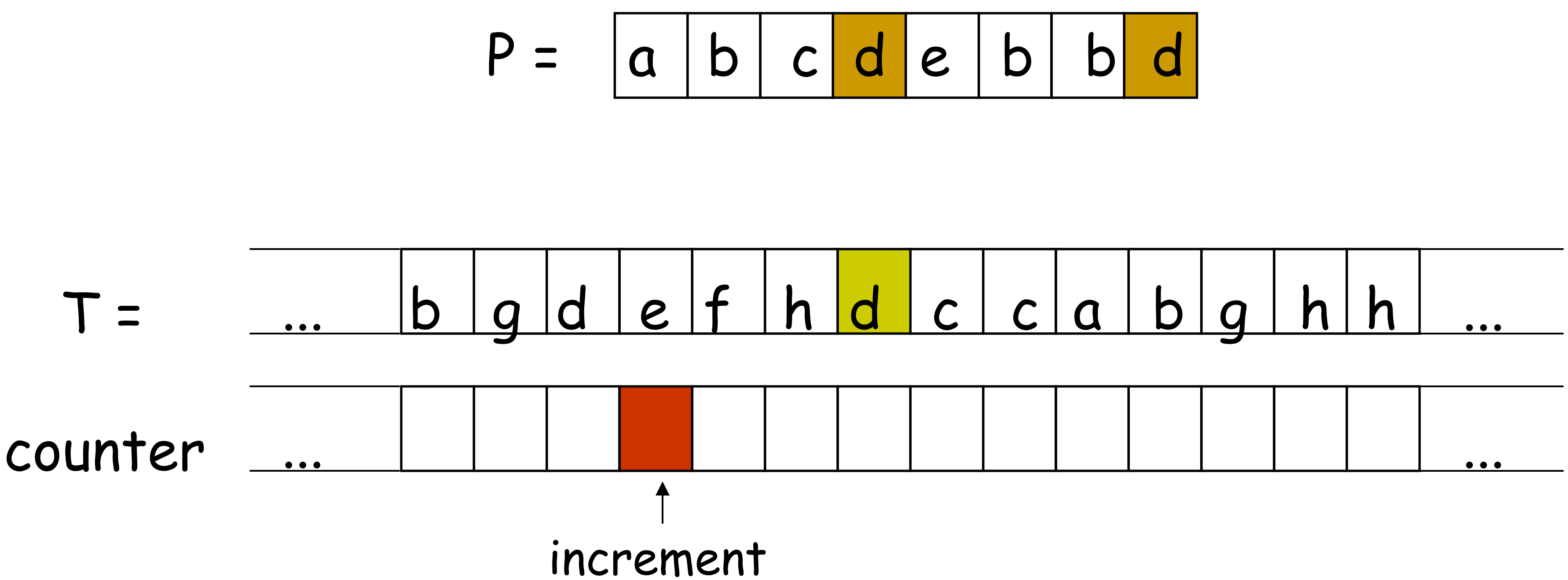
For each character you have a list of offsets where it occurs in the pattern,

When you see the char in the text, you increment the appropriate counters.

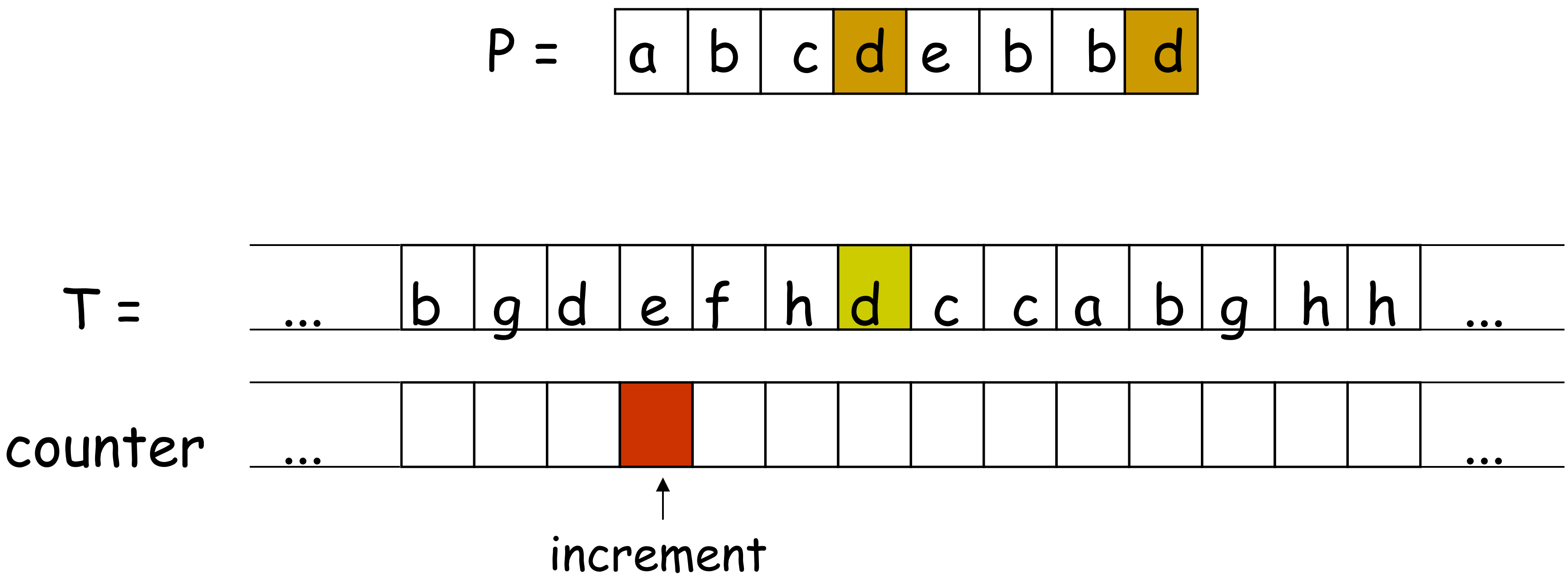


For each character you have a list of offsets where it occurs in the pattern,

When you see the char in the text, you increment the appropriate counters.



This is fast if all characters are "rare"



Partition the characters into rare and frequent

Rare: occurs $\leq c$ times in the pattern

For rare characters run this scheme with the counters

Takes $O(nc)$ time

Frequent characters

You have at most m/c of them

Do a convolution for each

Total cost $O((m/c)n \log(m))$.

Fix c

$$cn = \frac{m}{c} n \cdot \log(m) \Rightarrow$$

$$c^2 = m \cdot \log(m) \Rightarrow$$

$$c = \sqrt{m \cdot \log(m)}$$

Complexity: $O(n\sqrt{m \cdot \log(m)})$

Back to the k-mismatch problem

Want to beat the O(nk) kangaroo bound

Frequent Symbol: a symbol that appears at least $2\sqrt{k}$ times in P.

Few ($\leq\sqrt{k}$) frequent symbols

Do the counters scheme for non-frequent $O(n\sqrt{k})$

Convolve for each frequent $O(n\sqrt{k} \log m)$

($\geq\sqrt{k}$) frequent symbols

Intuition: There cannot be too many places where we match

($\geq\sqrt{k}$) frequent symbols

- Consider $\sqrt{k}$ frequent symbols.

- For each of them consider the first $2\sqrt{k}$ appearances.

Do the counters scheme just for these symbols and occurrences

$k = 4, \quad 2\sqrt{k} = 4$

P =	a	b	a	c	c	a	c	b	a	c	a	b	a	c	c
a-mask	**a**	b	**a**	c	c	**a**	c	b	**a**	c	a	b	a	c	c
c-mask	a	b	a	**c**	**c**	a	**c**	b	a	**c**	a	b	a	c	c

T =	...	a	b	b	b	c	c	c	**a**	a	a	a	b	a	c	b	...

⇕

use a-mask

Example of Masked Counting

k = 4, $2\sqrt{k}$ = 4

P =

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

a-mask

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

c-mask

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

		a	b	a	c	c	a	c	b	a	c	a	b	a	c	c	
T =	...	d	a	b	b	b	c	c	c	a	a	a	a	b	a	c	b	...
counter	...	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	...

Example of Masked Counting

k = 4, $2\sqrt{k}$ = 4

P =

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

a-mask

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

c-mask

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

			a	b	a	c	c	a	c	b	a	c	a	b	a	c	c
T =	…	d	a	b	b	b	c	c	c	a	a	a	a	b	a	c	b	…
counter	…	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	…

Counting Stage:

Run through text and count occurrences of all marks.

Time: $O(n\sqrt{k})$.

Important Observations:

1) Sum of all counters $\leq 2\sqrt{k}\,n$
2) Every counter whose value is less than k already has more than k errors.

   Why? The total # of elements in all masks is $2\sqrt{k}\sqrt{k} = 2k$.

$\Rightarrow$ For location i of T, if $counter_i < k$ then no match at location i.

How many locations remain?

Sum of all counters: $\leq \; 2n\sqrt{k}$

Value of potential matches > k

$\Rightarrow$ # of potential matches: $\leq \dfrac{2n\sqrt{k}}{k} = \dfrac{2n}{\sqrt{k}}$

How do we check these locations?

Use The Kangaroo Method.

Kangaroo method takes: O(k) per location

<u>Overall Time</u>: $O(\frac{n}{\sqrt{k}} k) = O(n\sqrt{k})$

Additional Points

Can reduce to

$$O(\, n \sqrt{k \log k} \,)$$

An alternative presentation of this last result

Back to the k-mismatch problem
Nicolae and Rajasekaran (2013)

Want to beat the O(nk) Kangaroo bound

P =

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

T =

...	d	a	b	b	b	c	c	c	a	a	a	a	b	a	c	b	...

Collect 2k “instances” (=individual chars in the pattern) with cost at most B (> n).
The cost of an “instance” is its frequency in the text.
Greedily put cheap instances first

Back to the k-mismatch problem
Nicolae and Rajasekaran (2013)

Case 1: Managed to collect 2k instances of total cost at most B:

Run the counting procedure for them.

Rule out positions with counter < k

Run kangaroo for the other positions

Back to the k-mismatch problem
Nicolae and Rajasekaran (2013)

Case 2: There aren't 2k instances of total cost at most B ....

Run the counting procedure for the instances in the knapsack

Do convolution for characters out of the knapsack

Analysis

Preparing the Knapsack takes $O(m+n)$

Case 1: Managed to collect 2k instances of total cost at most B:

Run the counting procedure for them. $O(n+B)$

Rule out positions with counter < k $O(n)$

Run kangaroo for the other positions

At most B/k positions with counter > k… $O(B)$ to run the kangaroo on them

Analysis

Do convolution for characters out of the knapsack

We will put instances of chars that occur $\leq B/n$ times in the pattern in the Knapsack

Doing marking for them will take $\leq B$ time

Now there are at most $r=2k/(B/n)$ not in the Knapsack (Otherwise we should have filled the Knapsack taking B/n occurrences of each)

Total cost of convolution $O(n^2 k \log(m)/B)$

Analysis

$$\frac{n^2 k \log(m)}{B} = B$$

$$B = n\sqrt{k \log(m)}$$