PATTERN MATCHING WITH DON'T CARES

Let $\phi \notin \Sigma$

INPUT: $T = t_0 \ldots t_n$

$P = p_0 \cdots p_m \qquad t_i, p_j \in \Sigma \cup \{\phi\}$

OUTPUT: All locations j where

$t_j = p_0$ or $t_j = \phi$ or $p_0 = \phi$

$t_{j+1} = p_1$ or $t_{j+1} = \phi$ or $p_1 = \phi$

$\vdots \qquad \vdots \qquad \vdots$

$t_{j+k} = p_k$ or $t_{j+k} = \phi$ or $p_k = \phi$

$\vdots \qquad \vdots \qquad \vdots$

$t_{j+m} = p_m$ or $t_{j+m} = \phi$ or $p_m = \phi$

-1-

EXAMPLE:

P = B ∅ B

T = A ∅ A ∅ A ∅ B ∅ B

matches

Can witness or automata methods help?

– No. Transitivity fails.

EXAMPLE: P: A ∅ B B

A ∅ B B

↑ here: witness table * automaton would shift

But if text:

... A B B B ...

match A ∅ B B

no match A ∅ B B

-2-

NEW METHOD: Polynomial Multiplication

(Fischer & Paterson 1971)

$$\begin{array}{cccccc} & a_0 & a_1 & a_2 & a_3 & a_4 \\ \times & & & b_2 & b_1 & b_0 \\ \hline & a_0b_0 & a_1b_0 & a_2b_0 & a_3b_0 & a_4b_0 \\ & a_0b_1 & a_1b_1 & a_2b_1 & a_3b_1 & a_4b_1 \\ a_0b_2 & a_1b_2 & a_2b_2 & a_3b_2 & a_4b_2 & \\ \hline \end{array}$$

all comparisons

Define $a \cdot b = \begin{cases} 0 & \text{if } a=b \\ 1 & \text{o/w} \end{cases}$

Then $T \times P^R[i+m-1] = 0$ iff

P matches T in location i.

-3-

How does it solve don't care problem?

- Define $\phi \cdot a = a \cdot \phi = 0 \quad \forall a \in \Sigma$.

Another problem it solves:

Count # of mismatches at every location.

EXAMPLE:

$A\phi A\phi A\phi$ × $B\phi B$

		AB	φB	AB	φB	AB	Bφ
		1	0	1	0	1	0
	Aφ	φφ	Aφ	φφ	Aφ	φφ	
	0	0	0	0	0	0	
AB	φB	AB	φB	AB	φB		
1	0	1	0	1	0		
		2	0	2	0		

BAD NEWS !!!

Time is $O(nm)$

no better than naive algorithm!

GOOD NEWS !!!

Polynomial multiplication can be done in time $O(n \log m)$ using FFT.

BAD NEWS !!!

FFT uses fact that $\mathbb{C}$ algebraically closed field for $\times, +$.

Under our new definition of $\times$, $\mathbb{C}$ not even a field so FFT Does Not Work!

-5-

Promissing News !!!

Use polynomial multiplication over $\mathbb{C}$ to solve our problem.

Assume: $\Sigma = \{0, 1\}$

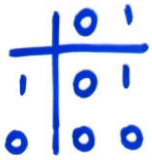
our "x"

T \ P	0	1
0	0	1
1	1	0

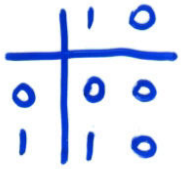
"x" in $\mathbb{C}$

T \ P	0	1
0	0	0
1	0	1

Note: If we exchange rows we are "almost" there



If we exchange columns we are "almost" there



IMPORTANT: In both exchanges, we have "1"s in different locations.

THEREFORE: $\overline{T} \times p^R + T \times \overline{p}^R$ gives desired result in time $O(n \log m)$

EXAMPLE: $T = A \not\emptyset A \not\emptyset A \not\emptyset$

$P = \quad B \not\emptyset B$

Let $A=1 \quad B=0$

$T = 1 \not\emptyset 1 \not\emptyset 1 \not\emptyset$

$P = \quad 0 \not\emptyset 0$

$\overline{T} \times P^R$

$0 \not\emptyset 0 \not\emptyset 0 \not\emptyset$

$0 \not\emptyset 0$

0 0 0 0 0 0 0 0

$T \times \overline{P}^R$

$1 \not\emptyset 1 \not\emptyset 1 \not\emptyset$

$1 \not\emptyset 1$

1 0 1 0 1 0

0 0 0 0 0 0

1 0 1 0 1 0

2 0 2 0

+

2020

What about larger alphabets?

1. Alphabet size bounded by

$$\sigma = \min(m+1, |\Sigma|)$$

Why?

- Let $\Pi = \{x \in \Sigma \mid x \in P\}$

  Let $\alpha \notin \Pi$

  For every text location i

  either $t_i \in \Pi \cup \{\emptyset\}$ or put

  $t_i \leftarrow \alpha$.

This reduces alphabet size to $m+1$

If Π is sorted, this can be done in time $O(n \log m)$

-9-

to next page set

2. Encode every alphabet symbol in binary notation. Need $\log \sigma$ bits.
Encode $\emptyset$ by $\emptyset^{\log \sigma}$ bits.

Pattern Matching with don't cares now solved in time $O(n \log m \log \sigma)$

Mismatch problem not solved!

Errors can not be counted properly.

EXAMPLE: $\log \sigma = 3$

Text:	111	000	111		001	110	001	
Pattern:	000	111	000		000	111	000	
	9 bit errors				3 bit errors			

In both cases 3 "real" mismatches.

-10-

Solution:

For each pattern symbol $a \in \Pi$ and each text location, count number of text elements that mismatch the a's of the pattern.

Formally:

$$\chi_a(b) = \begin{cases} 1 & b = a \\ 0 & b \neq a \end{cases}$$

$$\chi_a(s_1 \cdots s_k) = \chi_a(s_1)\,\chi_a(s_2) \cdots \chi_a(s_k)$$

$$\chi_{\bar{a}}(s_1 \cdots s_k) = \overline{\chi_a(s_1)}\;\overline{\chi_a(s_2)} \cdots \overline{\chi_a(s_k)}$$

Algorithm

for every $a \in \Pi$ do:

$$V_a \leftarrow \chi_{\bar{a}}(T) \times \chi_a(P)^R$$

end

$$V \leftarrow \sum_{a \in \Pi} V_a$$

end Algorithm

Time: $O(n \log m |\Pi|)$

EXAMPLE:

T = AABCC

P = ABC

# of errors

AABCC

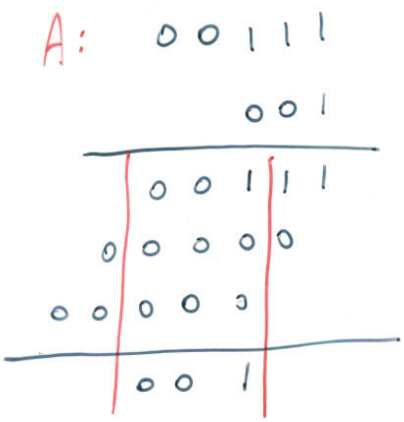
2 ABC

0 ABC

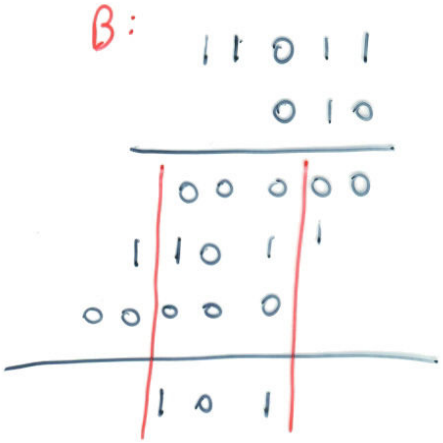
2 ABC

$\Pi = \{A, B, C\}$

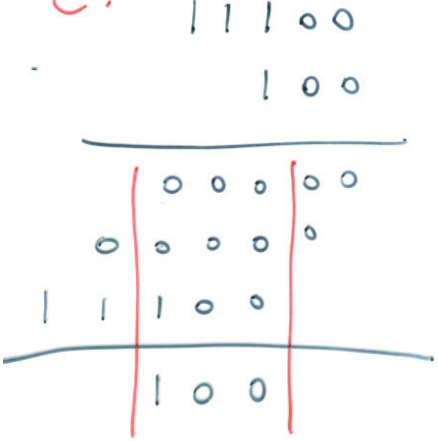
A:



B:



C:



0 0 1

\+ 1 0 1

1 0 0

2 0 2

For bounded alphabets,

e.g. biology $\Sigma = \{A, T, G, U\}$

Our Time: $O(n \log m)$

For unbounded alphabets,

Our Time: $O(mn \log m)$

Worse than naive !!!

SOLUTION:

Divide and Conquer.

Abrahamson- Kosaraju, 1987

- 14 -

ASSUMPTION 1: $n \leq 2m$

i.e. Text length no more than twice pattern length.

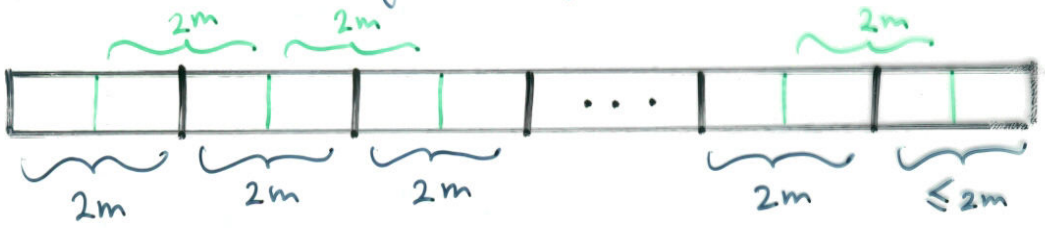
Why can this be assumed?

Claim: If pattern matching with mismatches can be solved in time $O(m f(m))$ for texts of length $\leq 2m$

Then it can be solved in time $O(n f(m))$ for general texts

- 15 -

Proof: Split text to $\frac{n}{m}$ overlapping strings of length $\leq 2m$ each in following way:



In time $O(m\,f(m))$ check pattern appearances in each substring.

P matches T in location i for $i \bmod 2m \leq m$ iff it matches the $\lfloor \frac{i}{2m} \rfloor$ th substring starting at t_0, in location $i \bmod 2m$.

P matches T in location i for $i \bmod 2m > m$ iff it matches the $\lfloor \frac{i-m}{2m} \rfloor$ th substring starting at t_m, in location $(i-m) \bmod 2m$.

Time: $O(\frac{n}{m} m f(m)) = O(n f(m))$

The Problem we are Considering:

Input: $T = t_0 \cdots t_{2m}$

$P = p_0 \cdots p_m$

Output: For every text location i,

the number of mismatches

resulting from trying to

match $p_0 \cdots p_m$ to

$t_i \cdots t_{i+m}$.

-17-

Algorithm Outline

1) Reduce alphabet size to $O(\sqrt{m})$

2) Count mismatches in reduced alphabet in time $O(\sqrt{m}\, m \log m)$

3) "Adjust" to original alphabet in time $O(\sqrt{m}\, m)$.

Total Time: $O(m\sqrt{m}\log m)$

Total Time general texts:

$O(n\sqrt{m}\log m)$

1. Reducing The Alphabet

a) For each of the 3m symbols in T and P construct a triplet $\langle \text{symbol}, \{T, P\}, i \rangle$ where i is the symbol's location in T or P, resp.

b) Sort triplets by symbol.

EXAMPLE: T = A A C B A A A (0 1 2 3 4 5 6)

P = D A D C

sorted triples:

$\langle A,T,0\rangle \langle A,T,1\rangle \langle A,T,4\rangle \langle A,T,5\rangle \langle A,T,6\rangle$

$\langle A,P,1\rangle \langle B,T,3\rangle \langle C,T,2\rangle \langle C,P,3\rangle \langle D,P,0\rangle$

$\langle D,P,2\rangle$

-19-

c) Every symbol that has more than $\sqrt{m}$ triples is called frequent.

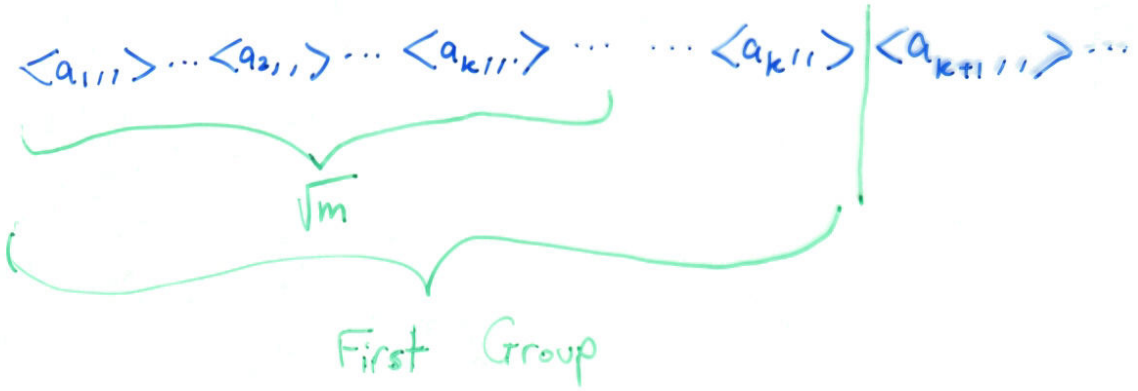
Move all triples of frequent symbols to end of list.

Maximum # of different frequent symbols: $3\sqrt{m}$

d) For triples of non-frequent symbols, split them as follows:

A group is $\sqrt{m}$ consecutive triples, extended (if necessary) until symbol changes.

Maximum Group Size: $2\sqrt{m}$



For frequent symbols - every frequent symbol is a group.

Total # of groups: $O(\sqrt{m})$

e) For every group, the symbol in the leftmost triple is the group representative.

# of representatives: $O(\sqrt{m})$

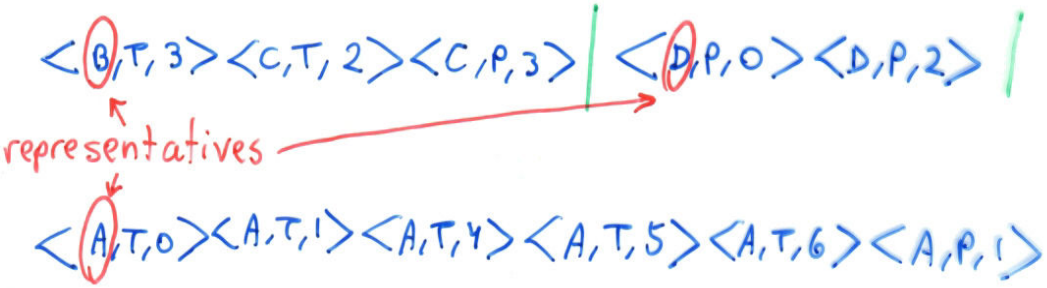
EXAMPLE: $T = AACBAAA$

$P = DADC$

$m = 4$

$\sqrt{m} = 2$

Groups:



f) Construct T', P' from T, P by replacing each symbol by its representative.

in EXAMPLE:

$T' = AABBAAA$

$P' = DADB$

-22-

KEY OBSERVATION: Every mismatch counted in T' and P' is a mismatch in T and P. But we may miss some mismatches in T and P.

in EXAMPLE:

T: A A C B A A A
P: D A D C
3 mismatches

T': A A B B A A A
P': D A D B
2 mismatches.

We now need to adjust errors.

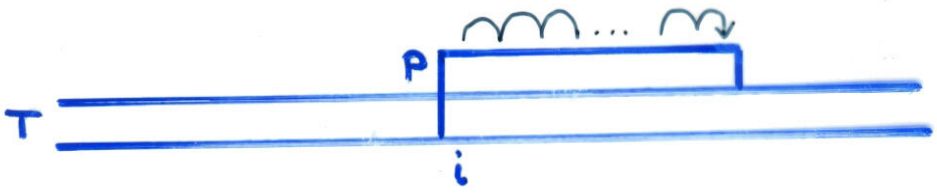
- 23 -

2. Adjustment

Recall naive string matching:

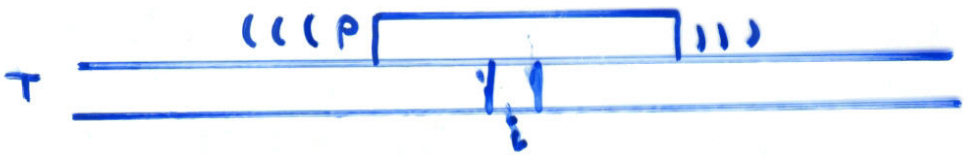
Let $E[0..n]$ be mismatch count

```
E ← 0
for i:=0 to n do
    for j:=0 to m do
        If T[i+j] ≠ P[j] then E[i] ← E[i]+1
    end
end
```



For every location i run forward and count mismatches for occurrence starting at i.

Another View:



For every location i, slide pattern and for every mismatch, update appropriate error counter.

```
E ← 0
for i=0 to n
    for j=0 to m
        If T[i] ≠ P[j] then E[i-j] ← E[i-j]+1
    end
end
```

In our algorithm:

If we slide P over T at location i, every symbol in P which is not in t_i's group already counted its errors.

The only uncounted errors are in t_i's group, where t_i is not a frequent symbol.

So slide only t_i's group, for non-frequent t_i, and add errors appropriately.

Time: $O(\sqrt{m})$ per text location, since non-frequent groups have size $\leq 2\sqrt{m}$

Total Adjustment Time: $O(n\sqrt{m})$

-26-

Total Algorithm Time:

Setting up groups and representatives: $O(m \log m)$

Finding mismatches for reduced alphabet: $O(m\sqrt{m} \log m)$

Adjustment: $O(m\sqrt{m})$

Total for $n \leq 2m$: $O(m\sqrt{m} \log m)$

Total for general texts: $O(n\sqrt{m} \log m)$

-27-

to next page set