CMSC423: Bioinformatic Algorithms, Databases and Tools

Exact string matching: introduction

Sequence alignment: exact matching

```
ACAGGTACAGTTCCCTCGACACCTACTACCTAAG          Text
CCTACT                                       Pattern
 CCTACT
  CCTACT
   CCTACT
```

```
for i = 0 .. len(Text) {
  for j = 0 .. len(Pattern) {
     if (Pattern[j] != Text[i]) go to next i
  }
  if we got there pattern matches at i in Text
}
```

Running time = O(len(Text) * len(Pattern)) = O(mn)

What string achieves worst case?

Worst case?

AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA

AAAAAAAAAAAAT

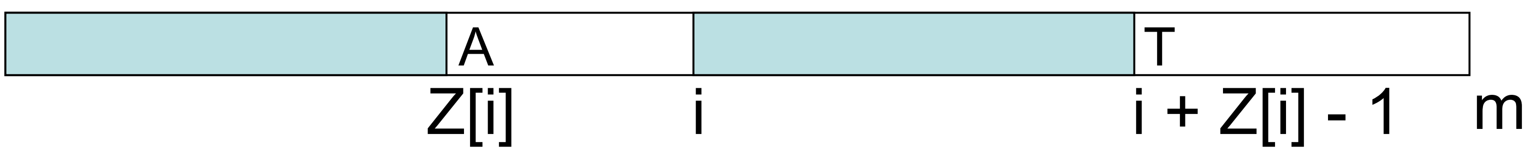
(m – n + 1) * n comparisons

Can we do better?

the Z algorithm (Gusfield)

For a string T, Z[i] is the length of the longest prefix of T[i..m] that matches a prefix of T. Z[i] = 0 if the prefixes don't match.

T[0 .. Z[i]] = T[i .. i+Z[i] -1]

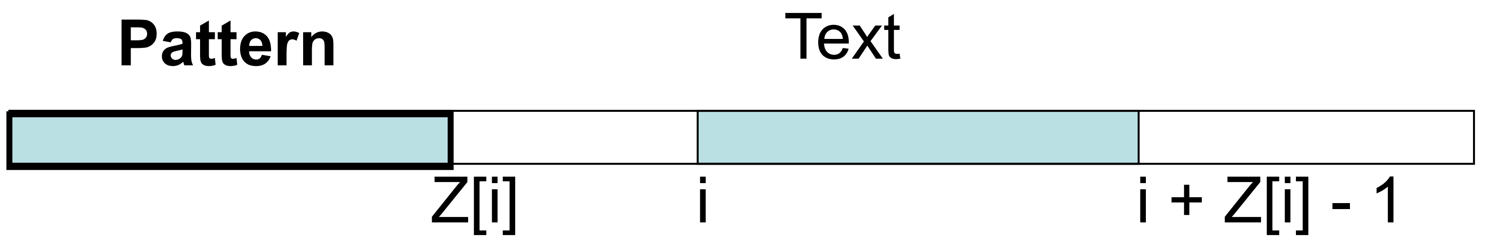


Example Z values

```
ACAGGTACAGTTCCCTCGACACCTACTACCTAAG
0010004010000000003020002002000110
```

Can the Z values help in matching?

Create string   Pattern$Text where $ is not in the alphabet



If there exists i, s.t. Z[i] = length(Pattern)
Pattern occurs in the Text starting at i

example matching

```
CCTACT$ACAGGTACAGTTCCCTCGACACCTACTACCTAAG
0100100010000010000231010010610010041000
```

- What is the largest Z value possible?

Can Z values be computed in linear time?

AAAGGTACAGTTCCCTCGACACCTACTACCTAAG

Z[1]? compare T[1] with T[0], T[2] with T[1], etc. until mismatch

Z[1] = 2

This simple process is still expensive:

T[2] is compared when computing both Z[1] and Z[2].

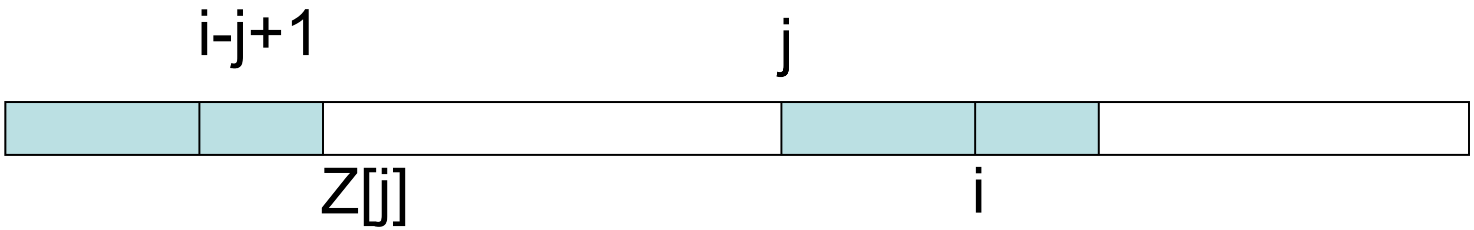
Trick to computing Z values in linear time:

each comparison must involve a character that was not compared before

Since there are only m characters in the string, the overall # of comparisons will be O(m).

Basic idea: 1-D dynamic programming

Can Z[i] be computed with the help of Z[j] for j < i?



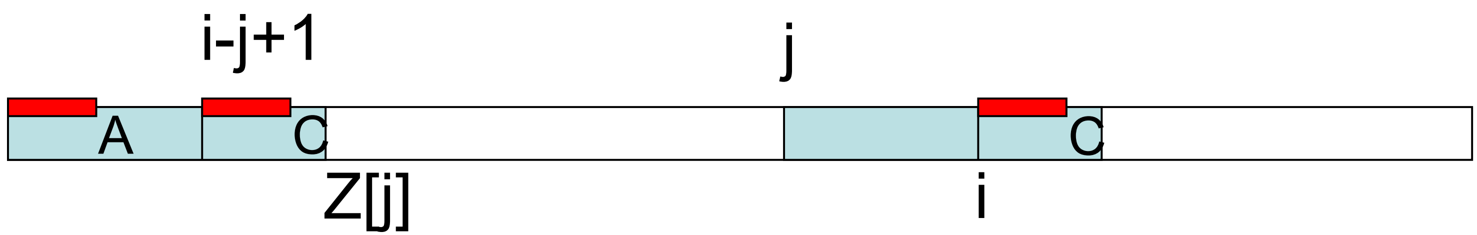
Assume there exists j < i, s.t. j + Z[j] – 1 > i
then Z[i – j + 1] provides information about Z[i]

If there is no such j, simply compare characters T[i..] to T[0..]
since they have not been seen before.

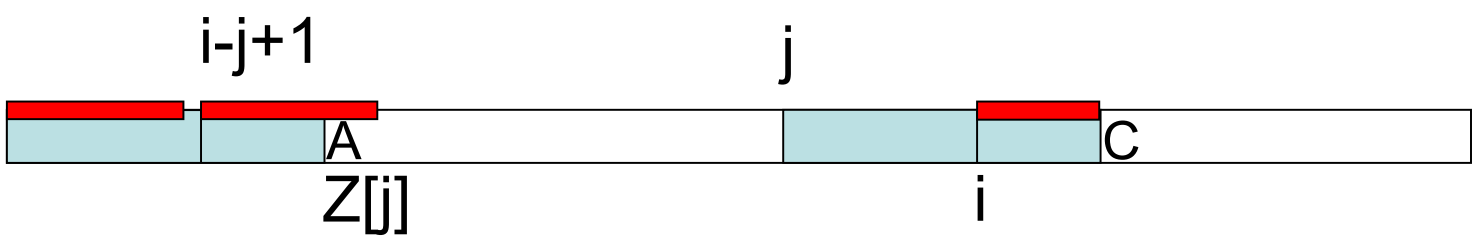
Three cases

Let j < i be the coordinate that maximizes j + Z[j] – 1 (intuitively, the Z[j] that extends the furthest)

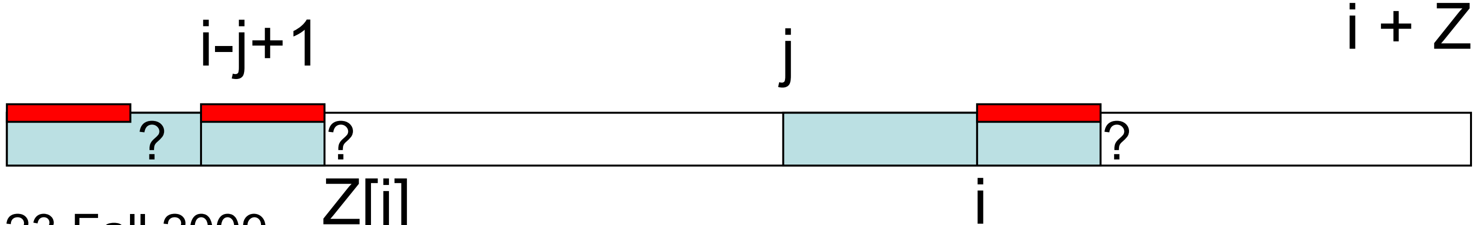
I. Z[i – j + 1] < Z[j] – i + j – 1 => Z[i] = Z[i – j + 1]



II. Z[i – j + 1] > Z[j] – i + j – 1 => Z[i] = Z[j] –i + j - 1



III. Z[i – j + 1] = Z[j] – i + j – 1 => Z[i] = ??, compare from i + Z[i – j + 1]



Time complexity analysis

- Why do these tricks save us time?

1. Cases I and II take constant time per Z-value computed – total time spent in these cases is O(n)
2. Case III might involve 1 or more comparisons per Z-value

   however:

   - every successful comparison (match) shifts the rightmost character that has been visited
   - every unsuccessful comparison terminates the “round” and algorithm moves on to the next Z-value

   total time spent in III cannot be more than # of characters in the text

Overall running time is O(n)

Space complexity?

- If using Z algorithm for matching, how many Z values do we need to store?

PPPPPPPPPP$TTTTTTTTTTTTTTTTTTTTTTTT

- Only need to remember Z-values for pattern and the “farthest reaching Z-value” (Z[j] in what we discussed before)

Some questions

- What are the Z-values for the following string:

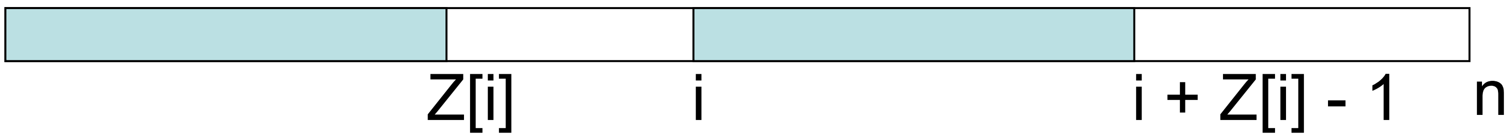
  TTAGGATAGCCATTAGCCTCATTAGGGATTAGGAT

- In the string above, what is the longest prefix that is repeated somewhere else in the string?

- Trace through the execution of the linear-time algorithm for computing the Z values for the string listed above. How many times do rules I, II, and III apply?

Z algorithm, not just for matching

- Lempel-Ziv compression (e.g. gzip)

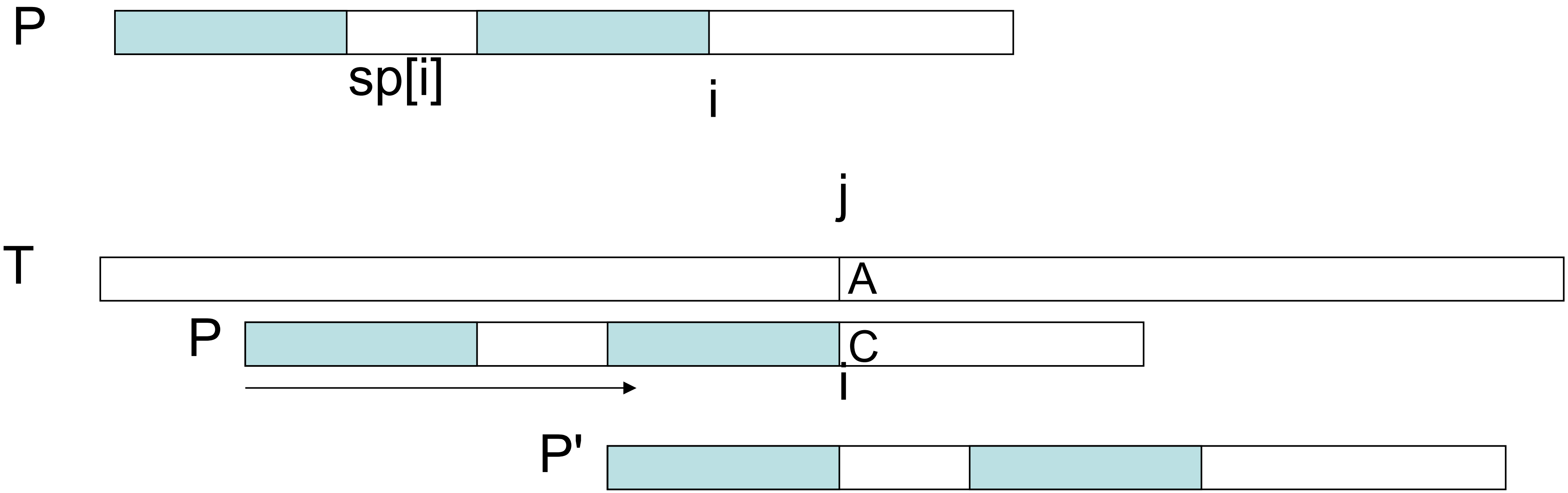


if Z[i] = 0, just send/store the character T[i], otherwise, instead of sending T[i..i+Z[i] – 1] (Z[i] – 1 characters/bytes) simply send Z[i] (one number)

- Note: other exact matching algorithms used for data compression (e.g. Burrows-Wheeler transform relates to suffix arrays)

Knuth-Morris-Pratt algorithm

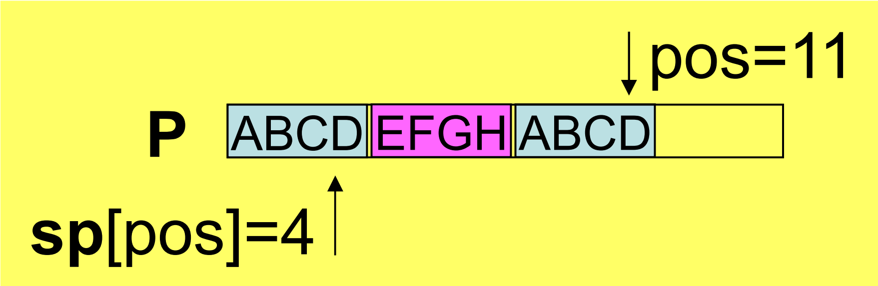
Given a Pattern and a Text, preprocess the Pattern to compute
sp[i] = length of longest prefix of P that matches a suffix of P[0..i]



- Compare P with T until finding a mis-match (at coordinate i + 1 in P and j + 1 in T).
- Shift P such that first sp[i] characters match T[j – sp[i] + 1 .. j].
- Continue matching from T[i+1], P[sp[i]+1]

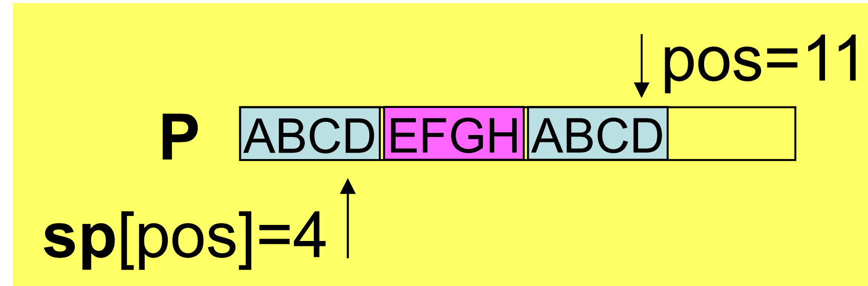
Knuth-Morris-Pratt (KMP) Algorithm

Given a pattern (**P**) and a text block (**T**) you preprocess **P** to compute a zero-indexed array **sp**[] where **sp**[pos] contains the length of longest ***prefix*** of **P** that matches a ***suffix*** of **P**[0..pos]



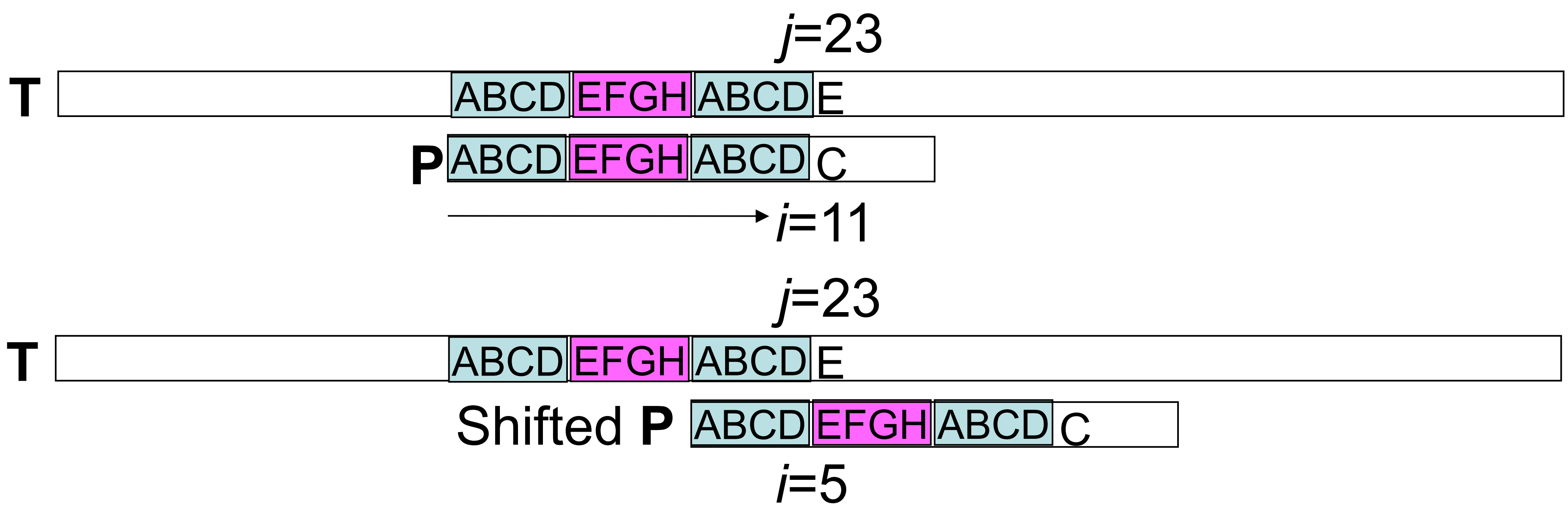
Next 4 slides from Evan Golub

Tracing KMP



We compare **P** with **T** until finding a mismatch.

We'll call that position *i+1* in **P** and *j+1* in **T**.



We then logically shift **P** using the **sp** value.

This allows the first **sp**[*i*] characters to match **T**[(*j*-sp[*i*]+1) .. *j*].

We then continue comparing from **P**[sp[*i*]+1] and **T**[*j+1*]

```
index:   0123456
pattern: AAAAAAA
sp:      0123456

index:   0123456
pattern: AAAAAAB
sp:      0123450
```

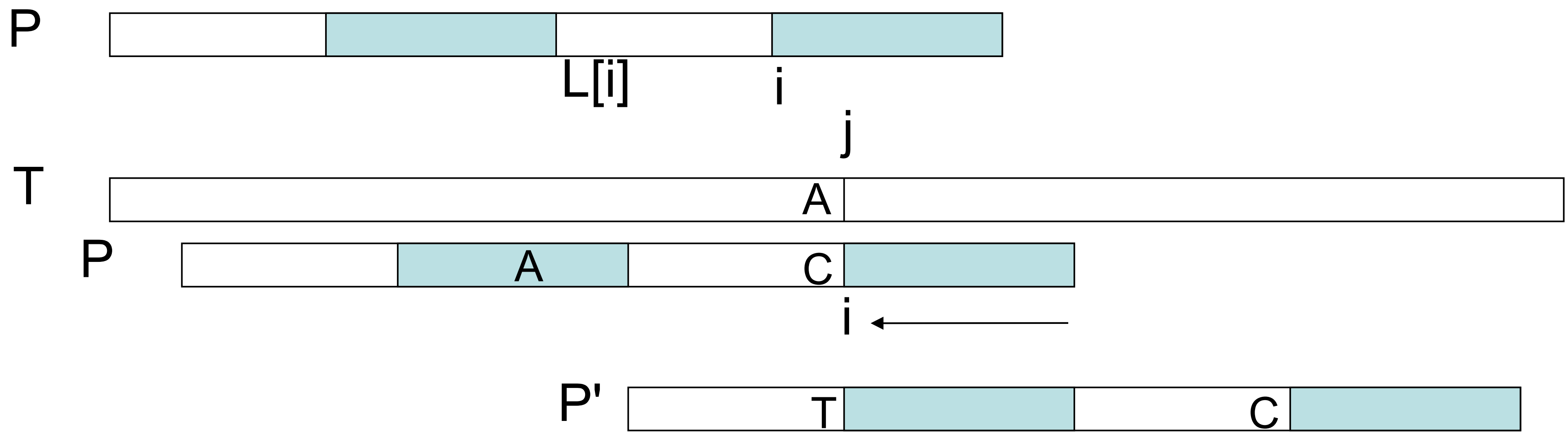
AAAAABAAAAAABAAAAAAA

```
index:   0123456
pattern: ABACABC
sp:      0010120
```

ABABBABAABABACABC

Boyer-Moore algorithm

Preprocess the pattern, computing, for every i, L[i] = largest coordinate < n, s.t. P[i..n] matches a suffix of P[1..L[i]] (inverted Z function)

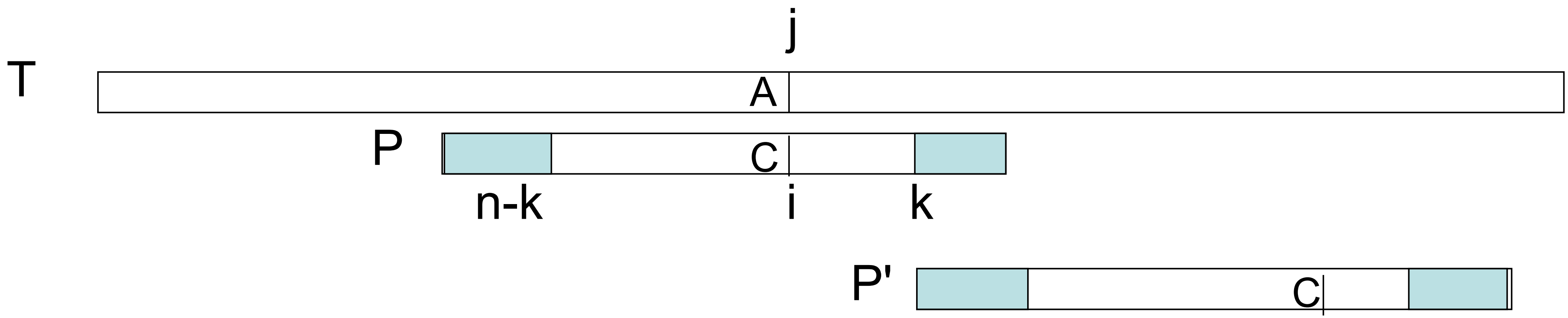


Match the pattern backwards (starting at the right) until mismatch.
Shift the pattern such that P[L[i] – n + i + 1] matches at T[j]
Repeat.

Bad character rule: find character T[j – 1] in P and shift until it matches. Choose the longest shift (btwn. suffix & char. rules)

Boyer-Moore ... cont

- What if P[i..n] does not occur elsewhere prior to i?
- Find k > i s.t. P[1..(n-k)] matches P[k..n]



- Also bad character rule:
  - if P[i] mismatches with T[j] – can shift P until we find a character equal to T[j] in P (above, shift until an A in P lines up to the A in T)
- Putting it all together: compute the shift according to the suffix rules and the bad character rule and pick the largest

Questions

- Can you use the Z-values to efficiently compute the sp() values used in the KMP algorithm?

- How about the values used by the Boyer-Moore algorithm?