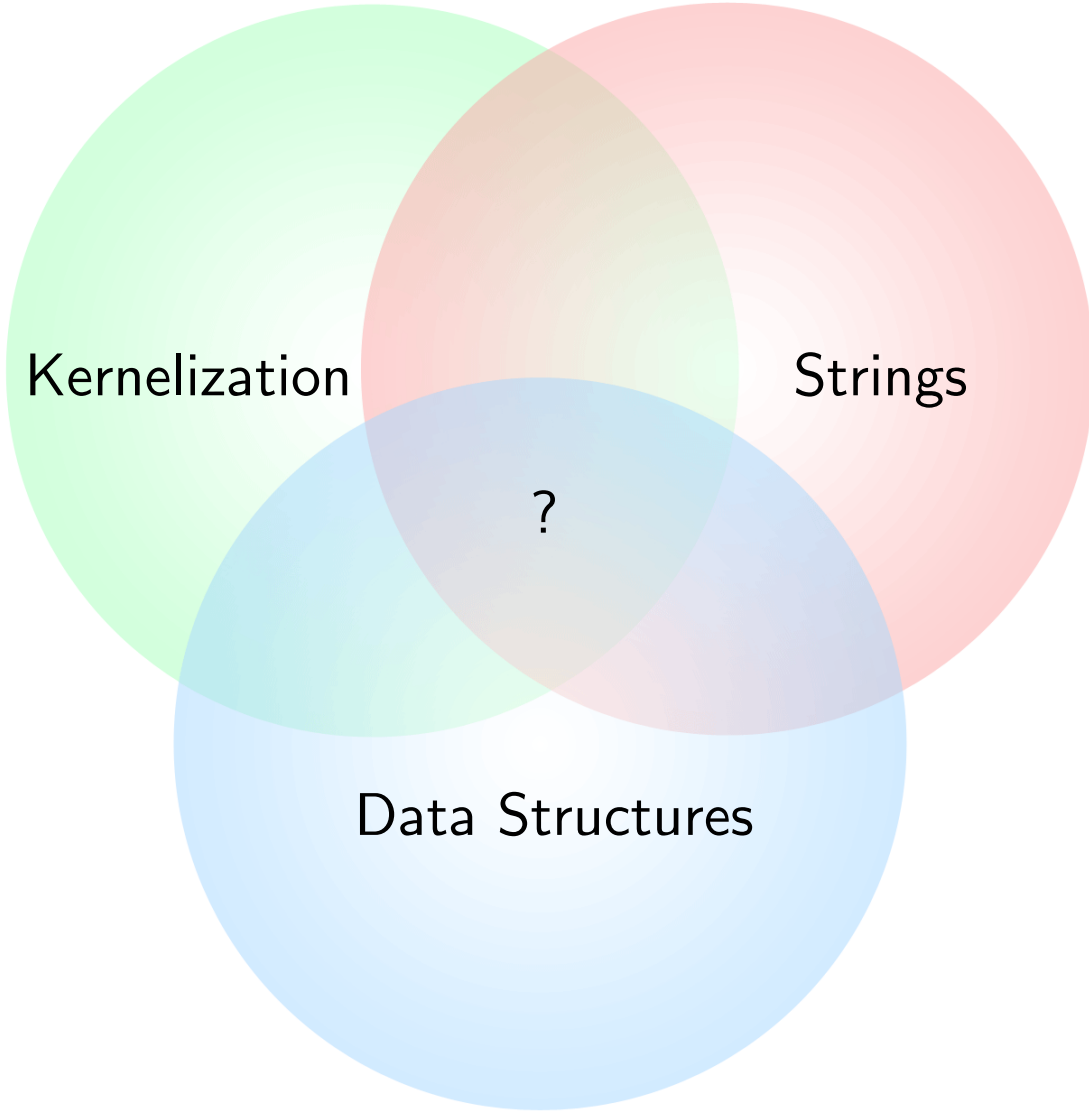
Searching and Indexing Genomic Databases via Kernelization

Travis Gagie and Simon J. Puglisi

Department of Computer Science
University of Helsinki

WorKer 2015


Kernelization
Strings
?
Data Structures

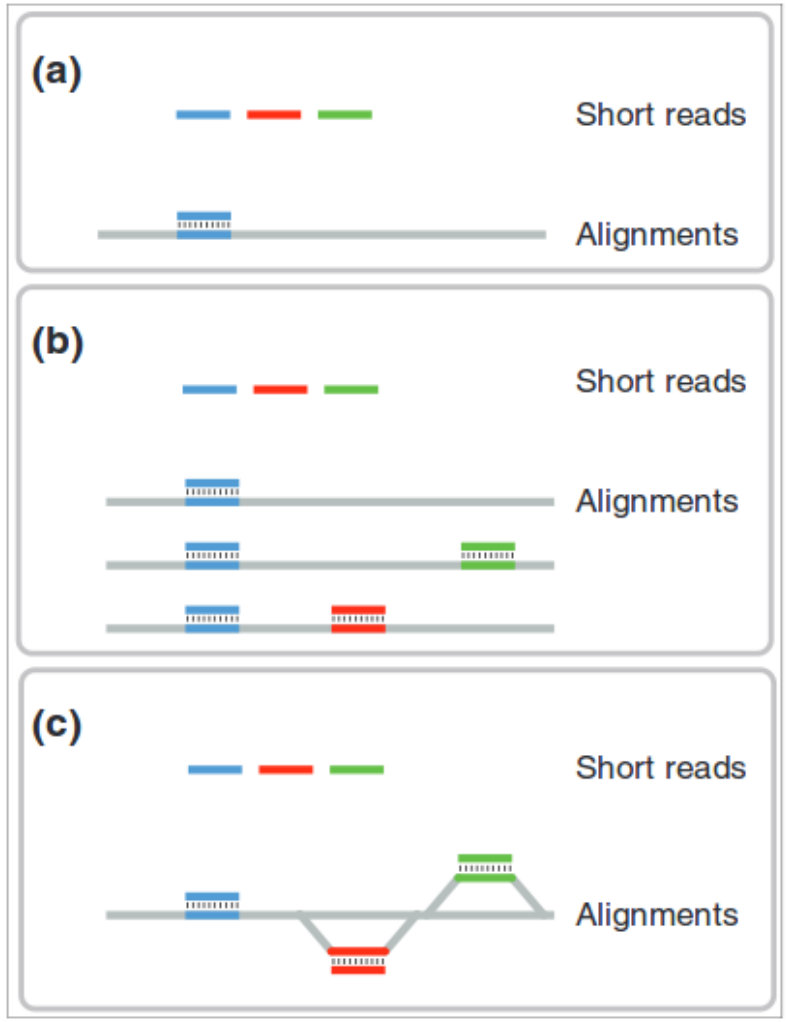
Kernelization $\approx$

Given a big thing X, we build a small thing X'
such that we can reason about X by looking at X'.

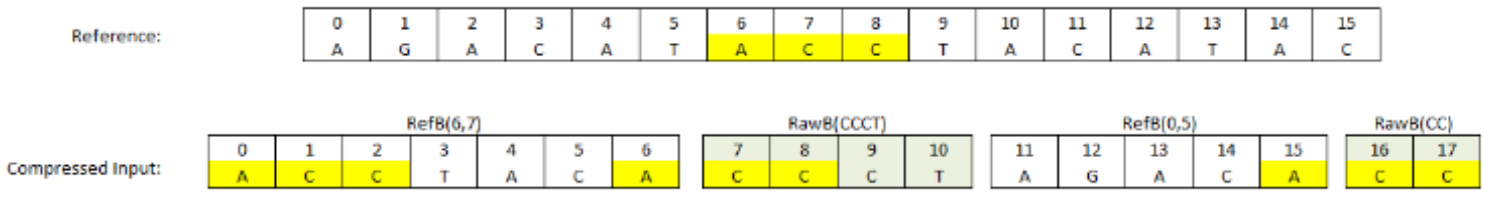
$\approx$ Compressed Data Structures?

For example, how do we search and index massive repetitive datasets?

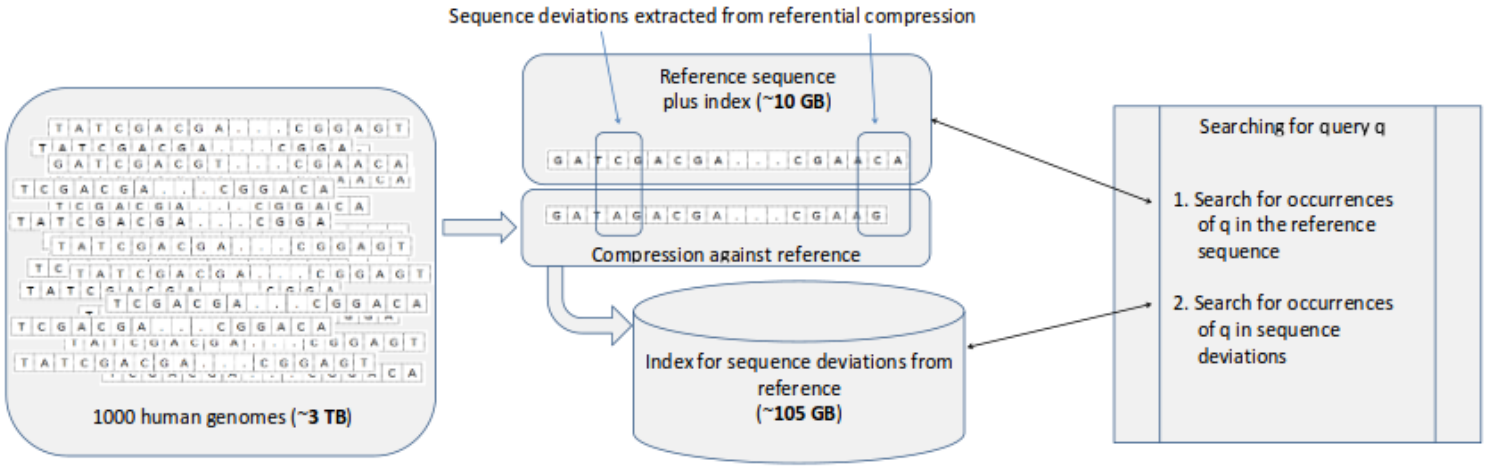
- Wikipedia (2001)
- GitHub (2008)
- genomic databases ($\sim$2008)



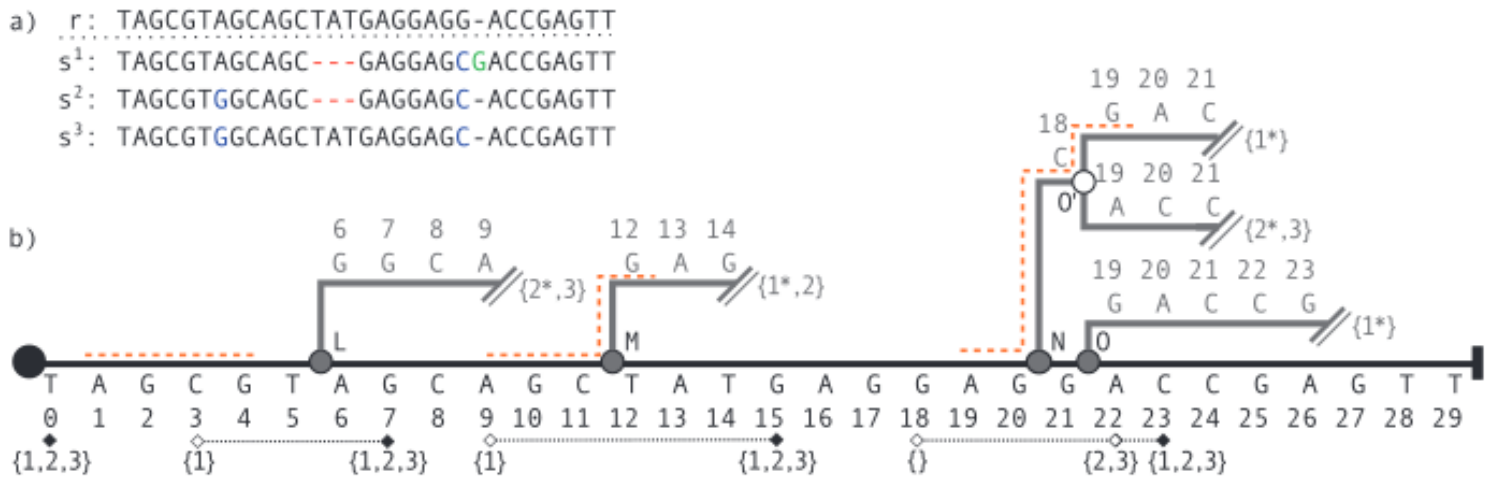
Schneeberger et al. Simultaneous alignment of short reads against multiple genomes, *Genome Biology*, 2009



Wandelt and Leser. String searching in referentially compressed genomes, *Proc. KDIR*, 2012



Wandelt et al. RCSI: scalable similarity search in thousand(s) of genomes, *Proc. VLDB*, 2013



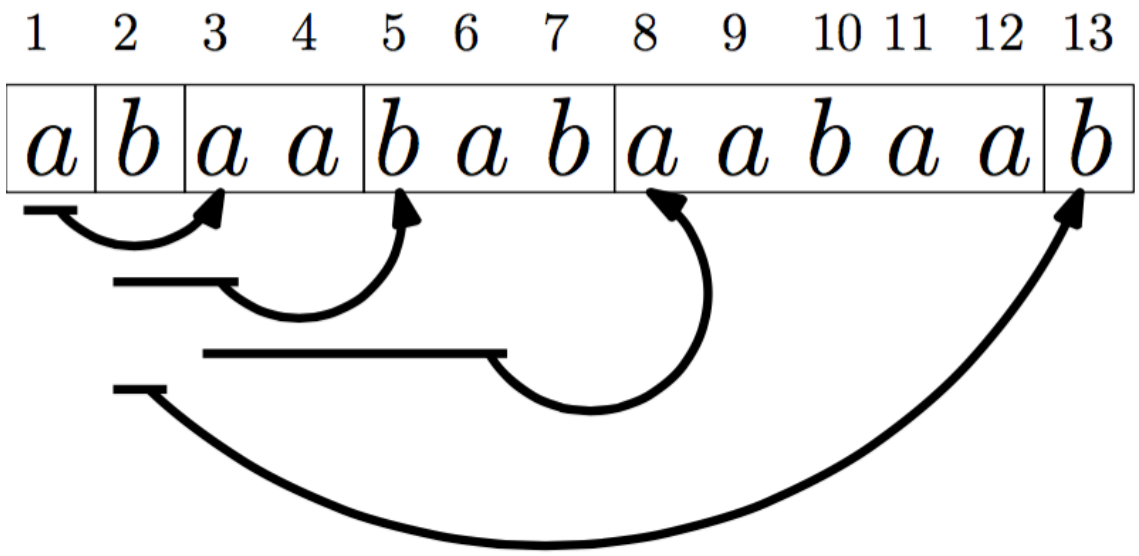
Rahn, Weese and Reinert. Journaled string tree — a scalable data structure for analyzing thousands of similar genomes on your laptop, *Bioinformatics*, 2014

Danek, Deorowicz and Grabowski. Indexes of large genome collections on a PC, *PLoS ONE*, 2014

Ferrada et al. Hybrid indexes for repetitive datasets, *Philosophical Transactions of the Royal Society A*, 2014

Gagie, Gawrychowski and Puglisi. Faster approximate pattern matching in compressed repetitive texts, *Journal of Discrete Algorithms*, 2015

Etc. etc. . . .

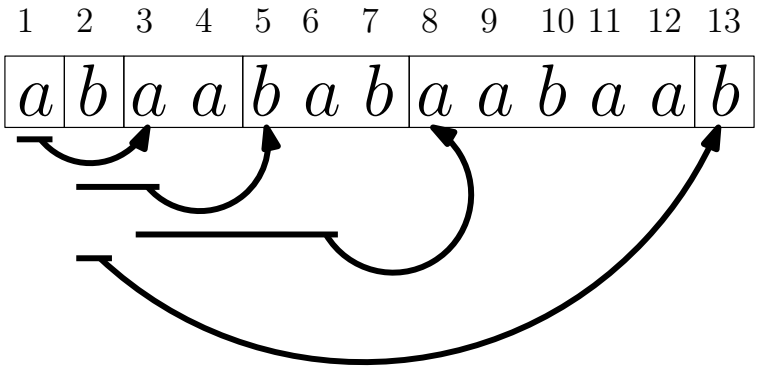

1 2 3 4 5 6 7 8 9 10 11 12 13
a b a a b a b a a b a a b

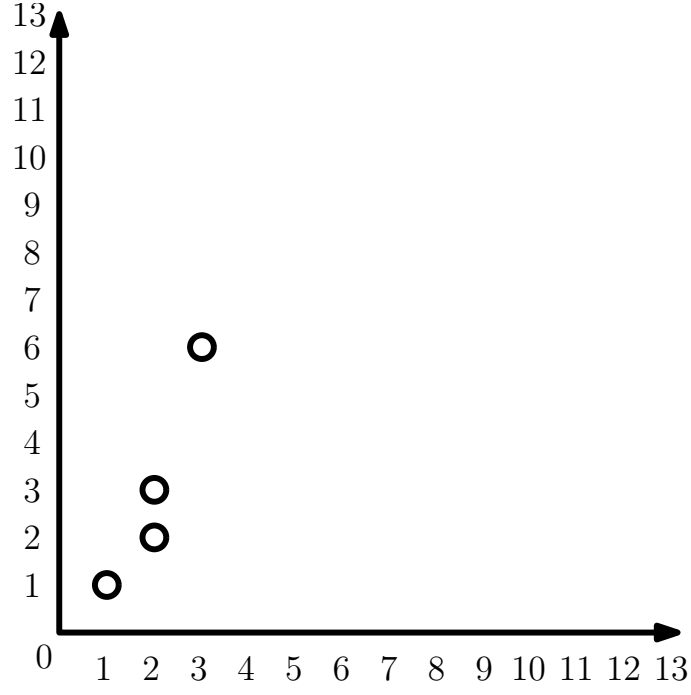
Observation (Farach and Thorup, 1995)

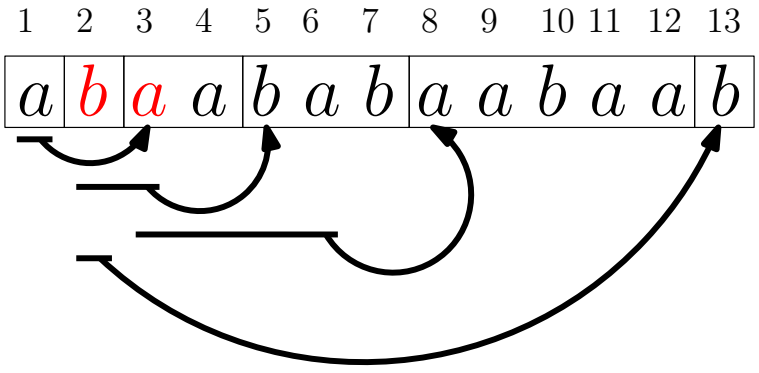
If a substring does not touch a phrase boundary, then it is copied from an earlier occurrence of that substring.

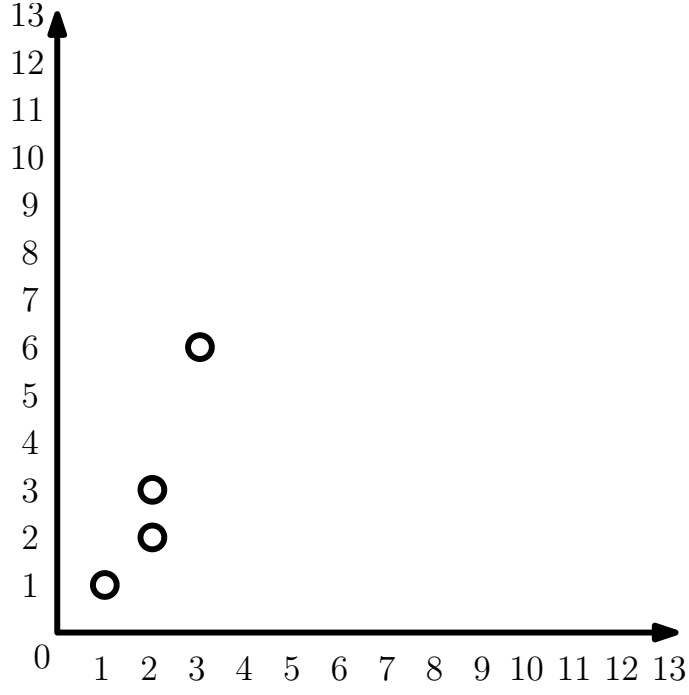
Theorem (Kärkkäinen and Ukkonen, 1996)

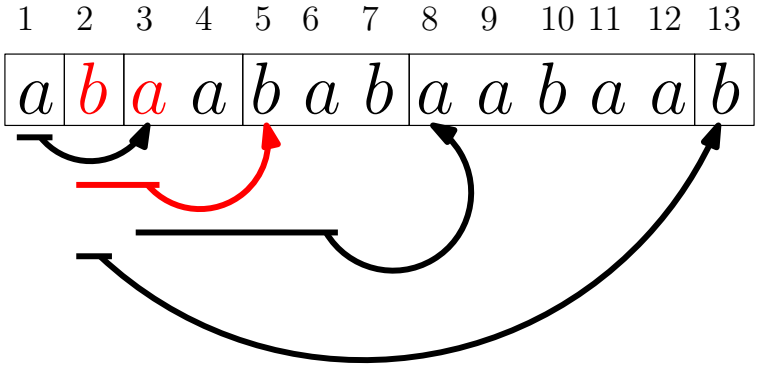
We can quickly infer the positions of all the occurrences of a pattern from the positions of its occurrences touching phrase boundaries, and the structure of the parse.

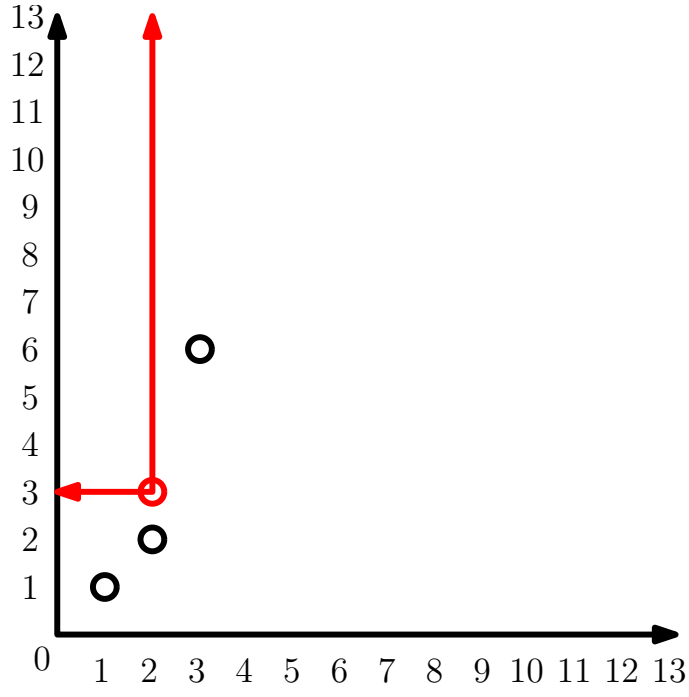

1 2 3 4 5 6 7 8 9 10 11 12 13
a b a a b a b a a b a a b

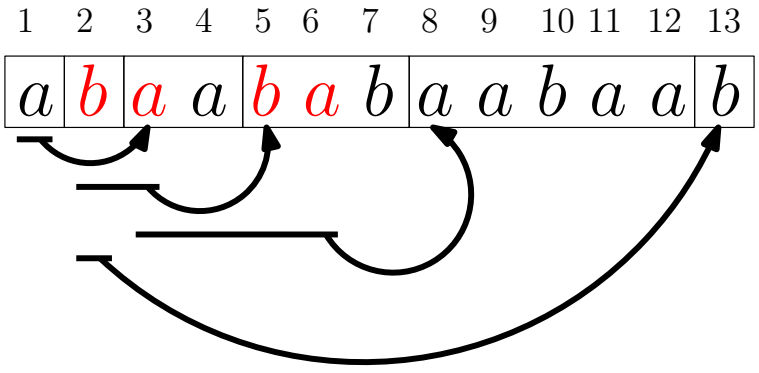

13
12
11
10
9
8
7
6
5
4
3
2
1
0
1 2 3 4 5 6 7 8 9 10 11 12 13

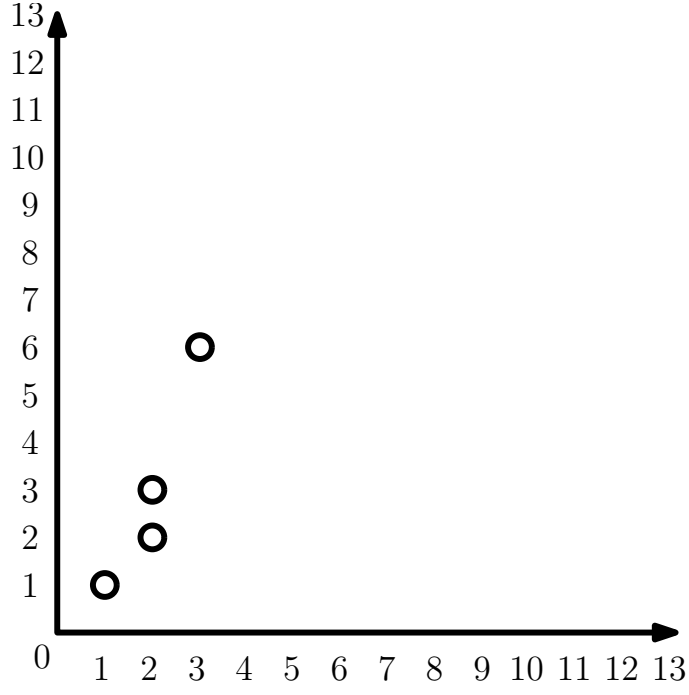

1 2 3 4 5 6 7 8 9 10 11 12 13
a b a a b a b a a b a a b

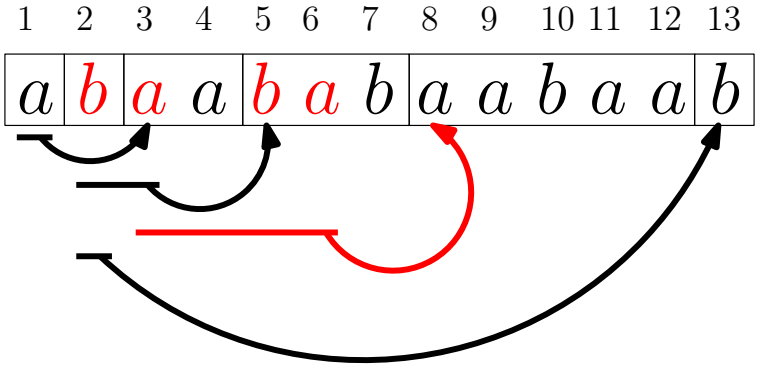

13
12
11
10
9
8
7
6
5
4
3
2
1
0
1 2 3 4 5 6 7 8 9 10 11 12 13

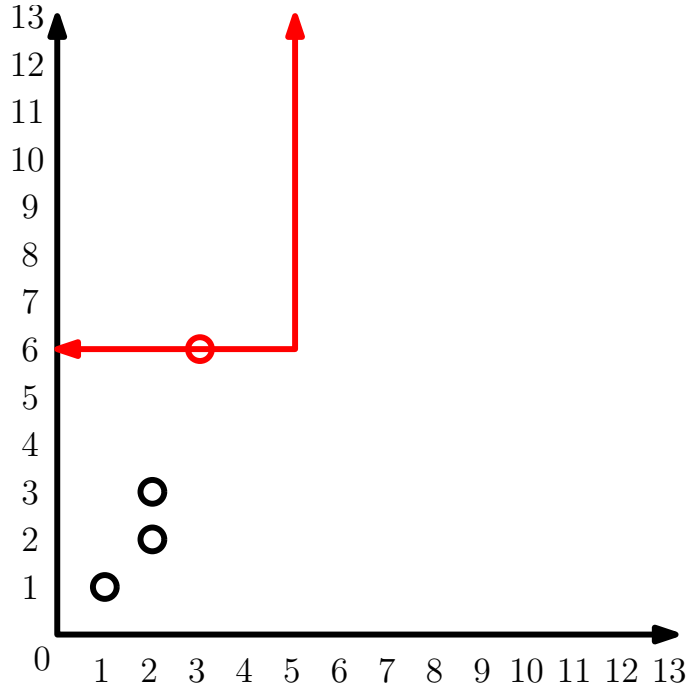

1 2 3 4 5 6 7 8 9 10 11 12 13
a b a a b a b a a b a a b

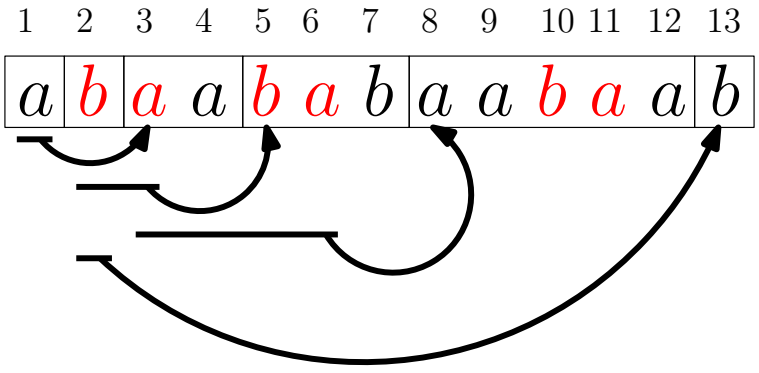

13
12
11
10
9
8
7
6
5
4
3
2
1
0
1 2 3 4 5 6 7 8 9 10 11 12 13

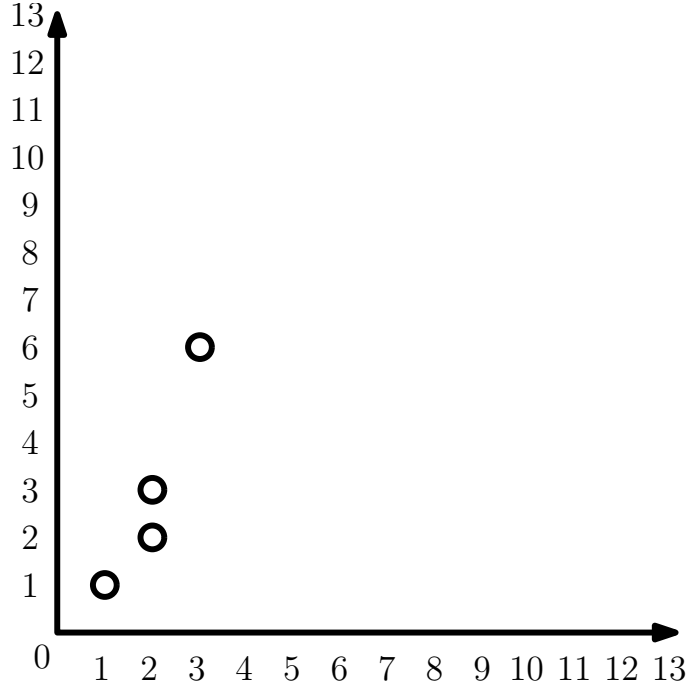

1 2 3 4 5 6 7 8 9 10 11 12 13
a b a a b a b a a b a a b

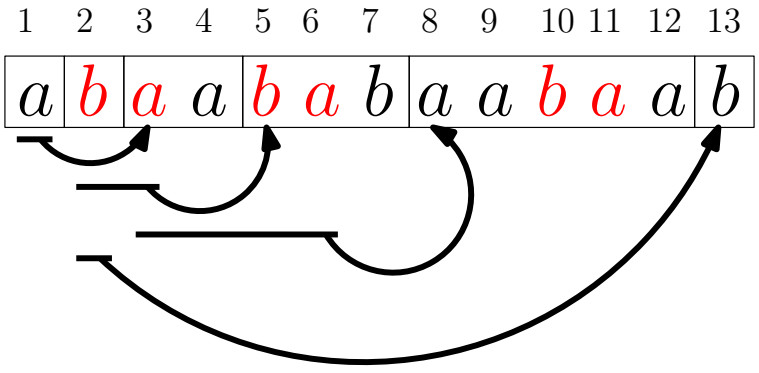

13
12
11
10
9
8
7
6
5
4
3
2
1
0
1 2 3 4 5 6 7 8 9 10 11 12 13

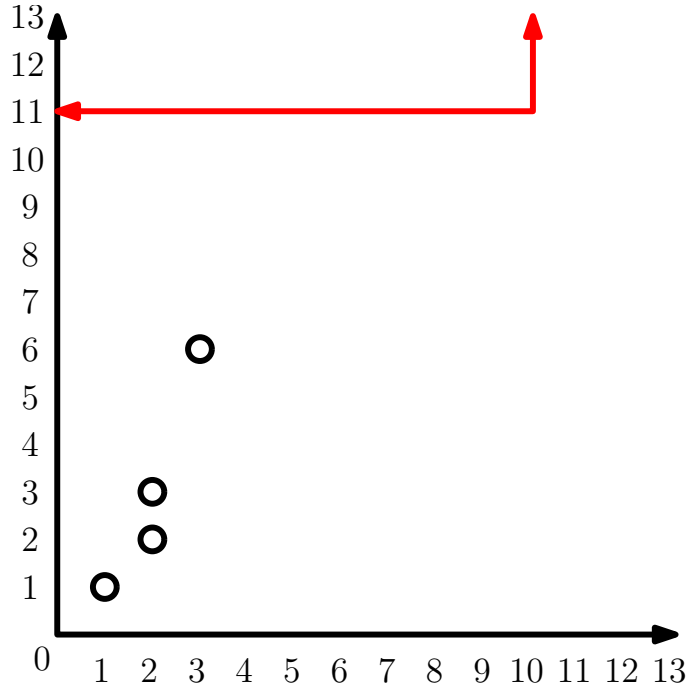

1 2 3 4 5 6 7 8 9 10 11 12 13
a b a a b a b a a b a a b

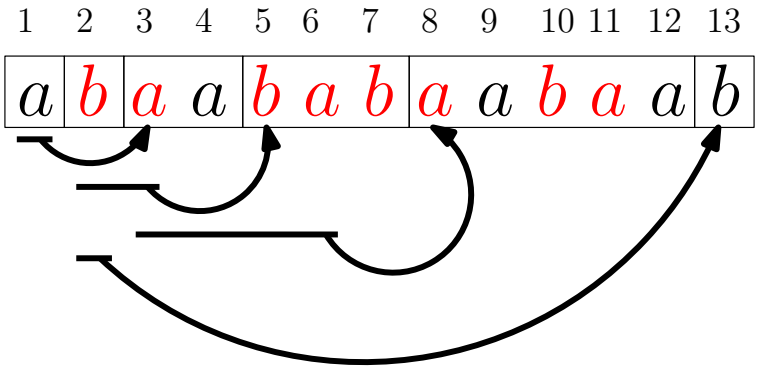

13
12
11
10
9
8
7
6
5
4
3
2
1
0
1 2 3 4 5 6 7 8 9 10 11 12 13

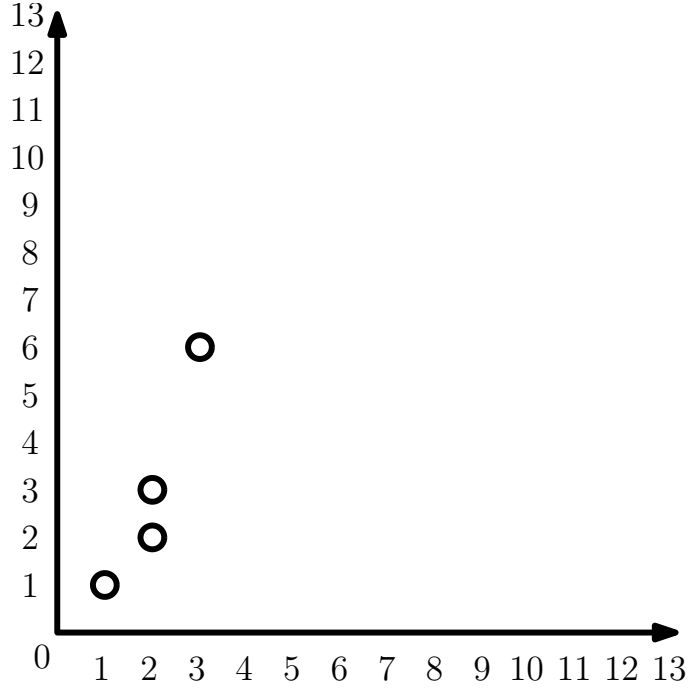

1 2 3 4 5 6 7 8 9 10 11 12 13
a b a a b a b a a b a a b

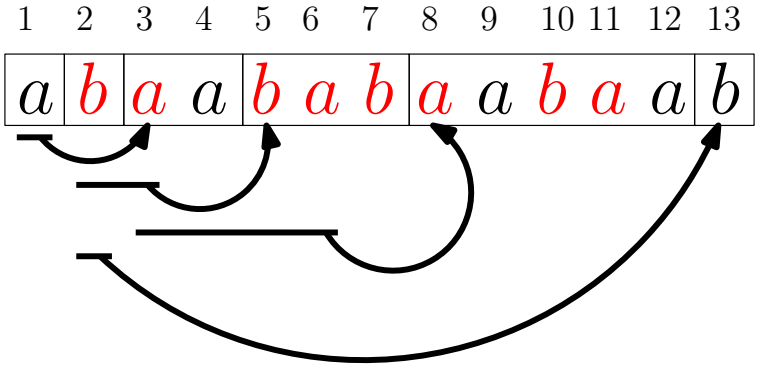

13
12
11
10
9
8
7
6
5
4
3
2
1
0
1 2 3 4 5 6 7 8 9 10 11 12 13

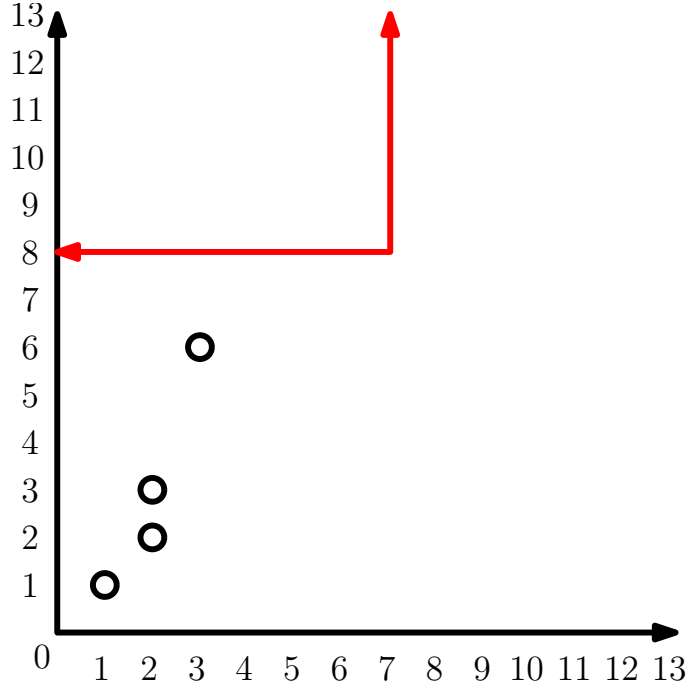

1 2 3 4 5 6 7 8 9 10 11 12 13
a b a a b a b a a b a a b


13
12
11
10
9
8
7
6
5
4
3
2
1
0
1 2 3 4 5 6 7 8 9 10 11 12 13


1 2 3 4 5 6 7 8 9 10 11 12 13
a b a a b a b a a b a a b


13
12
11
10
9
8
7
6
5
4
3
2
1
0
1 2 3 4 5 6 7 8 9 10 11 12 13


1 2 3 4 5 6 7 8 9 10 11 12 13
a b a a b a b a a b a a b


13
12
11
10
9
8
7
6
5
4
3
2
1
0
1 2 3 4 5 6 7 8 9 10 11 12 13

9|9-|b|o|t|tl|e|s|-o|f|-be|er|-on|-t|h|e-|w|a|ll|-9|9-bottles-of-beer-
l|f-o|n|e-o|f-t|ho|se|-bottles-s|hou|ld|-h|ap|pe|n-to|-f|all-
98|-bottles-of-beer-on-the-wall-

98|-bottles-of-beer-on-the-wall-98-bottles-of-beer-
l|f-one-of-those-bottles-should-happen-to-fall-
97|-bottles-of-beer-on-the-wall. . .

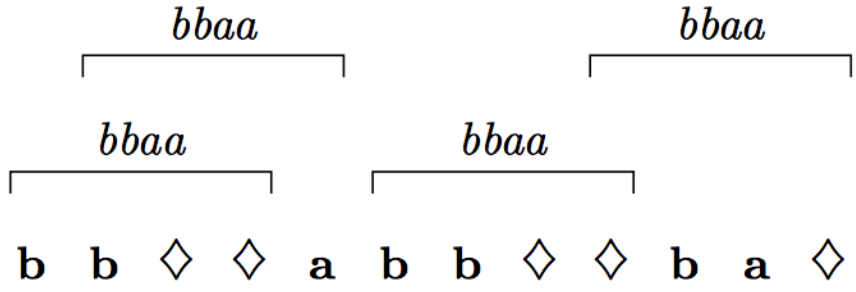
99-bottles-of-beer-on-the-wall-99-bo
eer-If-one-of-those-bottles-should-happen-to-fall-98-bot
ll-98-bot
eer-If-on
ll-97-bot. . .

Corollary

Given a bound on the maximum length of a match, we can build a kernel whose size is proportional to the product of that maximum length and the number of phrases, such that we can answer many pattern-matching problems looking only at that kernel (and perhaps the structure of the parse).

The kernelization is an interesting problem in itself, almost independent of the searching or indexing. Can we

- build optimal kernels?
- kernelize in external memory?
- maintain dynamic kernels?
- kernelize 2D arrays?
- prove lower bounds for compressed pattern matching?
- use these kernels for more than pattern matching?
- find a general framework for turning kernels into compressed data structures?



Crochemore et al. Covering problems for partial words and for indeterminate strings, *arXiv.org*, 2014

"We prove [the] partial word covering problem [is] NP-complete for binary alphabet and show [it is] fixed-parameter tractable with respect to k, the number of [don't-care] symbols [by giving] a $2^{O(\sqrt{k}\log k)} + nk^{O(1)}$-time algorithm. We prove that, unless the Exponential Time Hypothesis is false, no $2^{o(\sqrt{k})}n^{O(1)}$-time solution exists."

Questions?