

Suffix arrays

Suffix trees are $O(n)$ space, but the constant is large.

(≈ 20 bytes per char in good implementation)

10 Gb genome = 200 GB to store.

Suffix arrays are more efficient even than suffix trees & do most of the same ops.

Idea: Sort all suffixes of string lexicographically & keep the indices.

$S = \text{cattcat}\$$

1 cattcat\$
 2 attcat\$
 3 tcat\$
 4 cat\$
 5 cat\$
 6 at\$
 7 t\$
 8 \$

sorted $\rightarrow$

8\$
 6at\$
 2attcat\$
 5cat\$
 1cattcat\$
 7t\$
 4tcat\$
 3ttcat\$

Search substring by binary search of suffixes.

Counting is easy because all suffixes starting with "at" are next to each other.

App: k-mer counting for all k-mers.

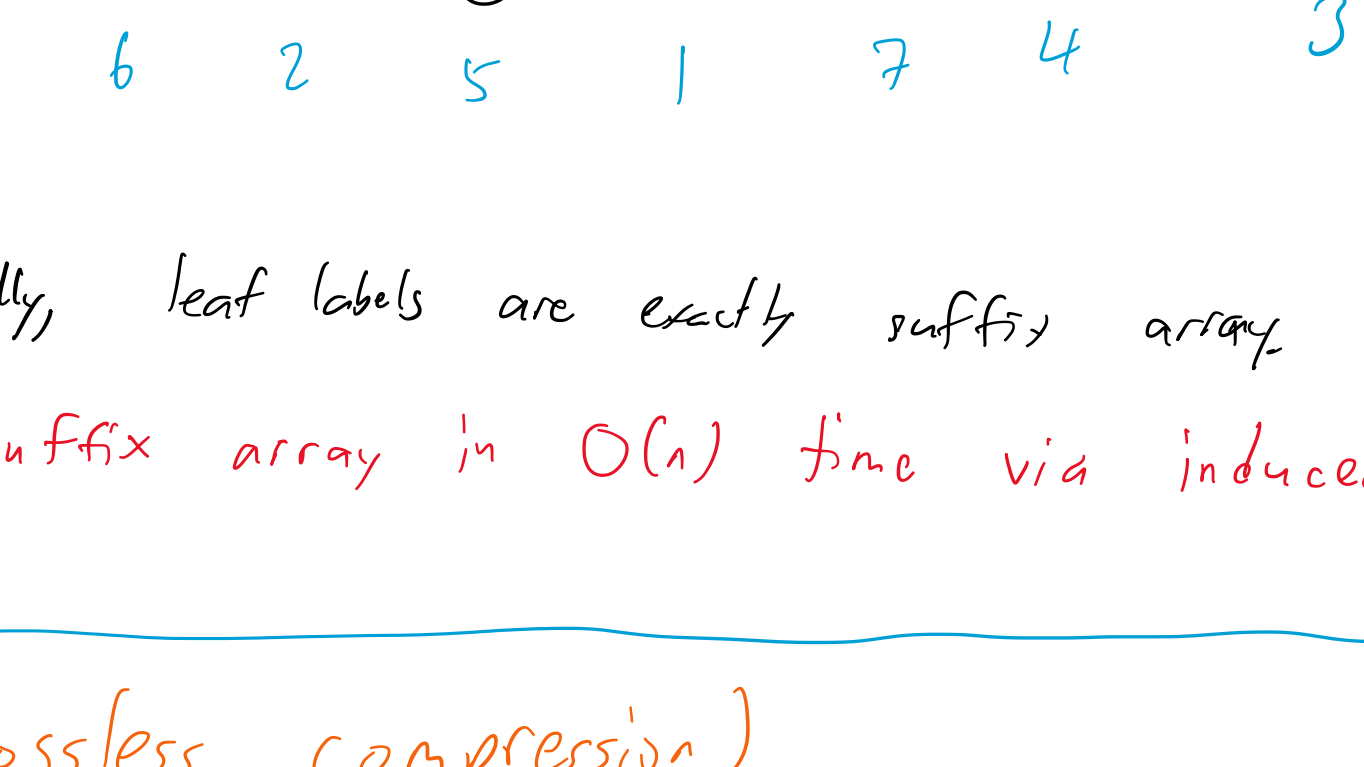
Only keep indices & original str S

Construction: $O(n^2 \log n)$ by sorting suffixes, where each comparison takes $O(n)$ time.

But better $O(n)$ algorithms such as building suffix tree.

$\Sigma = \{a, c, g, t\}$

$S = \text{cattcat}\$$
 12345678



If edges sorted lexicographically, leaf labels are exactly suffix array.

Can also directly craft suffix array in $O(n)$ time via induced sorting & recursion.

Compression basics (lossless compression)

0001111100011111 $\rightarrow$ 03 15 03 15 } Run-Length encoding

16 bytes

8 bytes

(00011111)₂

Lempel-Ziv compression

but need to find repeats, so a bit more complicated

CAGCAGCAG $\rightarrow$ (CAG)₃

Another approach: Permute the string.

Let's split the string into 3 strings by taking every 3rd char & concat

C A G C A G C A G
 C A C A C A
 G G G
 CCC AAA GGG
 C₃ A₃ G₃

Problems with permuting

Need to store the permutation - e.g. every 3rd char & concat

Permutation needs to be reversible

Large space of possible permutations to explore

Wouldn't it be lovely if there was a single magic permutation that always works?

Burrows-Wheeler Transform (1994)

Recall: Suffix arrays index & sort lists of suffixes

end of string delimiter

(text) $T = \text{CAGCAGCAG}\$$

Suffixes:

10\$
 9G\$
 8AG\$
 7CAG\$
 6GCAG\$
 5AGCAG\$
 4CAGCAG\$
 3GCAGCAG\$
 2AGCAGCAG\$
 1CAGCAGCAG\$

Sorted:

10\$
 8AG\$
 5AGCAG\$
 2AGCAGCAG\$
 7CAG\$
 4CAGCAG\$
 1CAGCAGCAG\$
 9G\$
 6GCAG\$
 3GCAGCAG\$

Change suffixes to rotations

$T = \text{CAGCAGCAG}\$$

Suffixes: Rotations

10\$CAGCAGCAG
 9G\$CAGCAGCA
 8AG\$CAGCAGC
 7CAG\$CAGCAGC
 6GCAG\$CAGCA
 5AGCAG\$CAGC
 4CAGCAG\$CAG
 3GCAGCAG\$CA
 2AGCAGCAG\$C
 1CAGCAGCAG\$

Sorted:

10\$CAGCAGCAG
 8AG\$CAGCAGC
 5AGCAG\$CAGC
 2AGCAGCAG\$C
 7CAG\$CAGCAGC
 4CAGCAG\$CAG
 1CAGCAGCAG\$
 9G\$CAGCAGCA
 6GCAG\$CAGCA
 3GCAGCAG\$CA

take last column of rotation matrix

BWT: GCCCG\$AAA

This last column is much more compressible

Why? If there is a repeated pattern CAG, then a string that starts with "AG" will probably end in "C", and we're putting all A-starts (suffixes) together.

Notice: BWT is reversible.

From last col, reconstruct 1st col by sorting.

From first & last col, reconstruct 2nd col by all pairs of letters.

Could even reconstruct suffix array indices by looking for '\$'

Exercise: What's the naive reverse-BWT runtime?

Define: The T-rank of a letter by how many times it has appeared in the Text

$T = C_0 A_0 G_0 C_1 A_1 G_1 C_2 A_2 G_2 \$$

F(1st)

\$ C₀ A₀ G₀ C₁ A₁ G₁ C₂ A₂ G₂
 A₂ G₂ \$ C₀ A₀ G₀ C₁ A₁ G₁ C₂
 A₁ G₁ C₂ A₂ G₂ \$ C₀ A₀ G₀ C₁
 A₀ G₀ C₁ A₁ G₁ C₂ A₂ G₂ \$ C₀
 C₂ A₂ G₂ \$ C₀ A₀ G₀ C₁ A₁ G₁
 C₁ A₁ G₁ C₂ A₂ G₂ \$ C₀ A₀ G₀
 C₀ A₀ G₀ C₁ A₁ G₁ C₂ A₂ G₂ \$
 G₂ \$ C₀ A₀ G₀ C₁ A₁ G₁ C₂ A₂
 G₁ C₂ A₂ G₂ \$ C₀ A₀ G₀ C₁ A₁
 G₀ C₁ A₁ G₁ C₂ A₂ G₂ \$ C₀ A₀

L(last)

Define: The LF-mapping property states that the nth occurrence of a letter in L (the last col) is the same character as the nth occurrence of X in F (the first col)

proof: Consider only rows with X in L. They will be ordered by the prefix of X.

But now consider only rows with X in R. They will be ordered by the suffixes of X.

By construction, the set of prefixes and suffixes correspond, and uniquely identify which X it is.

Notice: It doesn't matter what ranking we give. The rank order in L and F will match. $\square$

Define: Ascending rank: Assign ranks by order in F & L cols

F(1st)

\$ C A G C A G C A G₀
 A₀ G \$ C A G C A G C₀
 A₁ G C A G \$ C A G C₁
 A₂ G C A G C A G \$ C₂
 C₀ A G \$ C A G C A G₁
 C₁ A G C A G \$ C A G₂
 C₂ A G C A G C A G \$
 G₀ \$ C A G C A G C A₀
 G₁ C A G \$ C A G C A₁
 G₂ C A G C A G \$ C A₂

L(last)

ascending rank

Jumping around BWT

Which BWT row starts with G₂?

$\rightarrow$ G₂ C A G C A G \$ C A₂ $\rightarrow$ so A₂ comes before G₂

Fast reverse BWT:

Start in first row with \$ in F

The char X in L is the preceding character.

LF mapping then allows jumps to that occurrence of X in F.

Rise & Repeat.

Let's turn the BWT into a search index.

FM-index

"Full-text Minute-space" by Ferragina & Manzini, FOCS 2000

Basic idea is to store just F & L cols of BWT.

F takes 1 integer per alphabet character.

L is compressible.

Then augment with occurrence table to quickly compute ranks.

Query FM-index

Let Q = AGC

(can't binary search as in suffix array)

Start by looking in F for all C's.

Then use LF to find all G's before C's. (ie. finding CA)

Build backwards to get all AGC instances

(if query doesn't exist, will fail. e.g. GCAT)

Need to store ranks:

Easy for F. Why? All instances of each letter together.

For L, we build a tally table for every letter.

F	L	A	C	G	T
\$	G ₀	0	0	1	0
A	C	0	1	1	0
A	C	0	2	1	0
A	C	0	3	1	0
C	G ₁	0	3	2	0
C	G ₂	0	3	3	0
C	\$	0	3	3	0
G	A	1	3	3	0
G	A	2	3	3	0
G	A	3	3	3	0

which G's come before a C?

On G tally table, we go one before.

The C index to get 1-3 range

$\Rightarrow$ the 2nd & 3rd G's appear before C

$\Rightarrow$ G₁ & G₂

O(1) time

Can save space by only keeping some checkpoints & scanning forward.

Resolving offsets (position)

F	L	SA index	sample
\$	G	10	10
A	C	8	8
A	C	5	
A	C	2	2
C	G	7	
C	G	4	4
C	\$	1	
G	A	9	
G	A	6	6
G	A	3	

we keep every

kth offset,

so to find where

we are, we just

use LF property

to walk along at

most k times.

FM index components

First col (F): $\sim |\Sigma|$ integers

Last col (L): m characters

SA sample: m · a integers, where a is fraction of rows kept

Tally table: m · $|\Sigma|$ · b integers, where b is fraction of rows checkpointed.

HT = 3 billion

Ex: T = human genome, $|\Sigma| = 4$, $a = \frac{1}{32}$, $b = \frac{1}{128}$

FM-index stores human genome in ~ 1.5 GB

Full suffix array takes ≥ 12 GB

Hash table ~ 36 GB