Bioinformatics Algorithms

(Fundamental Algorithms, module 2)

Zsuzsanna Lipták

Masters in Medical Bioinformatics
academic year 2018/19, II. semester

Suffix Trees (and other string indexes)[1]

[1] Some of these slides are based on slides of Jens Stoye's.

Text indexes

Let T be a string of length n over alphabet Σ (which we refer to as text in the following).

A text index (or string index) is a data structure built on the text which allows to answer a certain type of query (e.g. pattern matching) without traversing the whole text. Typically, we want

1. the index not to use too much space (linear or sublinear in n), and
2. the query time to be fast (ideally: independent of n).

A common string problem: Pattern matching

Pattern matching (aka exact string matching) is at the core of almost every text-managing application.

Pattern matching

Given a (typically long) string T (the text), and a (typically much shorter) string P (the pattern) over the same alphabet Σ, find all occurrences of P as substring of T.

Variants:

- output all occurrences of P in T — "all-occurrences version"
- decide whether P occurs in T (yes - no) — "decision version"
- output the number of occurrences of P in T — "counting version"

We usually refer to the number of occurrences of P as occ_P.

Pattern matching

Pattern matching (p.m.)

text: $T = T_1 \ldots T_n$ of length n,
pattern: $P = P_1 \ldots P_m$ of length m

- The best non-index-based algorithms solve this problem in time $O(n + m)$ (e.g. Knuth-Morris-Pratt)
- This is optimal, since one has to read both strings at least once.
- But not tolerable with the data sizes we are seeing now!
- That is why we need text indexes.

The k-mer index

The k-mer index

Recall that a k-mer (or k-gram) is a string of length k.

k-mer index

Earlier in this course, we saw the k-mer profile, $P_k(s)$ (or q-gram profile) of a string s.

Ex.

$s =$ ACAGGGCA,
on the right is $P_2(s)$.

r	u_r	$P_2(s)$
0	AA	0
1	AC	1
2	AG	1
3	AT	0
4	CA	2
5	CC	0
6	CG	0
7	CT	0
8	GA	0
9	GC	1
10	GG	2
11	GT	0
12	TA	0
13	TC	0
14	TG	0
15	TT	0

The k-mer index

Replacing the number of occurrences by the occurrences themselves, we get the k-mer index of s.

Ex.
$s =$ `ACAGGGCA`,
on the right 2-mer index of s.

Analysis (for p.m.)
Space: total space is $O(\sigma^k + n)$, since no. of rows $= \sigma^k$ and total number of entries $= n - k + 1$.
Time (p.m.): $O(k)$ for decision, $O(k + occ_P)$ for all-occurrences.

r	u_r	k-mer index of s
0	AA	
1	AC	1
2	AG	3
3	AT	
4	CA	2, 7
5	CC	
6	CG	
7	CT	
8	GA	
9	GC	6
10	GG	4, 5
11	GT	
12	TA	
13	TC	
14	TG	
15	TT	

The k-mer index

Replacing the number of occurrences by the occurrences themselves, we get the k-mer index of s.

Ex.
$s =$ `ACAGGGCA`,
on the right 2-mer index of s.

Analysis (for p.m.)
Space: total space is $O(\sigma^k + n)$, since no. of rows $= \sigma^k$ and total number of entries $= n - k + 1$.
Time (p.m.): $O(k)$ for decision, $O(k + occ_P)$ for all-occurrences.
N.B.: works only for patterns of length exactly k

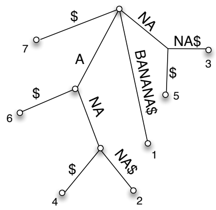
r	u_r	k-mer index of s
0	AA	
1	AC	1
2	AG	3
3	AT	
4	CA	2, 7
5	CC	
6	CG	
7	CT	
8	GA	
9	GC	6
10	GG	4, 5
11	GT	
12	TA	
13	TC	
14	TG	
15	TT	

The suffix tree

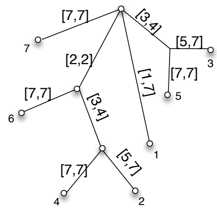
The suffix tree

T = BANANA\$ (add sentinel character $\$ \notin \Sigma$)

labels only conceptual!



two pointers into string



The suffix tree

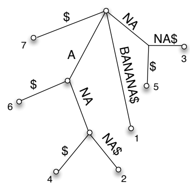
Given T string over Σ (finite ordered alphabet), and $\$ \notin \Sigma$.

Definitions

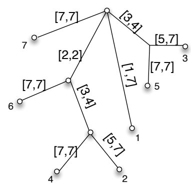
- $ST(T)$ is a rooted tree with edge-labels from $(\Sigma \cup \{\$\})^+$ such that
  - the labels of all edges outgoing from a node begin with different characters;
  - the paths from the root to the leaves of $ST(T)$ spell the suffixes of $T\$$;
  - each node in $ST(T)$ is either the root, a leaf, or a branching node;
- $L(u)$ is the path-label of node u: the concatenation of edge labels on the path from the root to u,
- a leaf v has leaf-label i if and only if $L(v) = T_i \ldots T_n\$$ (i'th suffix),
- $sd(v)$ is the string-depth of a node v is the length of its path-label,
- a locus (u, d) is a position on an edge (v, u) where u is a node of $ST(T)$ and $sd(v) < d \leq sd(u)$: d is the string-depth of locus (u, d).

- N.B.: the edge labels are not stored explicitly:
- they are represented by two pointers $[b, e]$ into T: beginning and end of an occurrence of the edge label;
- this representation is not necessarily unique
- e.g. in the example, any edge with label NA can be represented by $[3, 4]$ or $[5, 6]$

labels only conceptual!



two pointers into string



Suffix tree properties

- The leaves of $ST(T)$ correspond to the suffixes of $T\$$.
- $ST(T)$ represents exactly the substrings of $T\$$: there is a one-to-one correspondence between loci in $ST(T)$ (possibly within an edge) and substrings of $T\$$.
- This allows us to define locus(P) for a substring P of T.
- The leaves in the subtree under a locus(P) correspond to the (beginning positions of) P's occurrences in $T\$$: one-to-one correspondence between leaves in subtree under locus(P) and occurrences of substring P.
- $ST(T)$ requires $O(n)$ space (details next).

MAGIC!
The suffix tree represents a possibly quadratic number of objects (the substrings) in linear space!

Space usage of suffix trees

Lemma:
$ST(T)$ requires $O(n)$ space.

Proof sketch:

1. $ST(T)$ has exactly $n+1$ leaves (one for each suffix).
2. Each internal node is branching, therefore there are at most n internal nodes.
3. A tree with at most $2n+1$ nodes has at most $2n$ edges.
4. Each node can be represented in constant space.
5. Each edge is labeled by a substring of $T\$$ and hence can be represented by a pair of pointers $[i, j]$ into $T\$$.

The suffix array

The suffix array

Definition
The SA is a permutation of $\{1, 2, \ldots, n+1\}$ s.t. $\text{SA}[i] = j$ if the j'th suffix $\text{Suf}_j = T_j \cdots T_n\$$ is the i'th among all suffixes in lexicographic order.

Example: $T =$ `BANANA$`

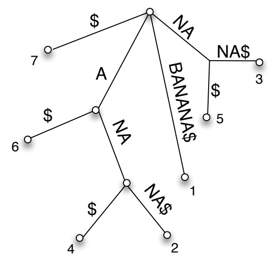
$$\text{SA} = [\overset{1}{7}, \overset{2}{6}, \overset{3}{4}, \overset{4}{2}, \overset{5}{1}, \overset{6}{5}, \overset{7}{3}]$$

i	SA	Suf_i
1	7	`$`
2	6	`A$`
3	4	`ANA$`
4	2	`ANANA$`
5	1	`BANANA$`
6	5	`NA$`
7	3	`NANA$`

Note \$ is smaller than all other characters.

The suffix array

Suffix tree



(Note that children of inner nodes are ordered acc. to the alphabet's order.)

Suffix array
SA = [7,6,4,2,1,5,3]

N.B.
When reading the leaves of the ST from left-to-right, we get the SA.

One can imagine the suffix array as the leaves of the suffix tree that fell down and stayed in order ...

Some Applications of Suffix Trees/Suffix Arrays

- exact string matching
- exact set matching
- text statistics
- DNA contamination problem
- common substrings of more than two strings
- matching statistics
- overlap computation (all-pairs prefix-suffix matching)
- exact repeats and palindromes problem
- tandem repeats problem
- shortest unique substring
- maximal unique matches
- approximate string matching (k-mismatch and k-differences)
- computation of the q-gram distance
- Lempel-Ziv data compression