Suffix Arrays

CMSC 858S

Suffix Arrays

- Even though Suffix Trees are O(n) space, the constant hidden by the big-Oh notation is somewhat "big": ≈ 20 bytes / character in good implementations.

- If you have a 10Gb genome, 20 bytes / character = 200Gb to store your suffix tree. "Linear" but large.

- Suffix arrays are a more efficient way to store the suffixes that can do most of what suffix trees can do, but just a bit slower.

- Slight space vs. time tradeoff.

Example Suffix Array

s = attcatg$

- Idea: lexicographically sort all the suffixes.
- Store the starting indices of the suffixes in an array.

index of suffix	suffix of s
1	attcatg$
2	ttcatg$
3	tcatg$
4	catg$
5	atg$
6	tg$
7	g$
8	$

sort the suffixes alphabetically →

the indices just "come along for the ride"

8	$
5	atg$
1	attcatg$
4	catg$
7	g$
3	tcatg$
6	tg$
2	ttcatg$

Example Suffix Array

- Idea: lexicographically sort all the suffixes.
- Store the starting indices of the suffixes in an array.

s = attcatg$

index of suffix	suffix of s
1	attcatg$
2	ttcatg$
3	tcatg$
4	catg$
5	atg$
6	tg$
7	g$
8	$

sort the suffixes alphabetically

→

the indices just "come along for the ride"

8
5
1
4
7
3
6
2

Another Example Suffix Array

s = cattcat$

- Idea: lexicographically sort all the suffixes.
- Store the starting indices of the suffixes in an array.

index of suffix	suffix of s
1	cattcat$
2	attcat$
3	ttcat$
4	tcat$
5	cat$
6	at$
7	t$
8	$

sort the suffixes alphabetically

→

the indices just "come along for the ride"

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

Another Example Suffix Array

s = cattcat$

- Idea: lexicographically sort all the suffixes.
- Store the starting indices of the suffixes in an array.

index of suffix	suffix of s
1	cattcat$
2	attcat$
3	ttcat$
4	tcat$
5	cat$
6	at$
7	t$
8	$

sort the suffixes alphabetically

→

the indices just "come along for the ride"

8
6
2
5
1
7
4
3

Search via Suffix Arrays

s = cattcat$

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

- Does string "at" occur in s?
- Binary search to find "at".
- What about "tt"?

Counting via Suffix Arrays

s = cattcat$

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

- How many times does "at" occur in the string?
- All the suffixes that start with "at" will be next to each other in the array.
- Find one suffix that starts with "at" (using binary search).
- Then count the neighboring sequences that start with at.

K-mer counting

Problem: Given a string s, an integer k, output all pairs (b, i) such that b is a length-k substring of s that occurs exactly i times.

k = 2

		CurrentCount	
8	$	1	
6	at$	1	
2	attcat$	2	
5	cat$	1	(at,2)
1	cattcat$	2	
7	t$	1	(ca,2)
4	tcat$	1	(t$,1)
3	ttcat$	1	(tc,1)
		1	(tt,1)

1. Build a suffix array.

2. Walk down the suffix array, keeping a CurrentCount count

    If the current suffix has length < k, skip it

    If the current suffix starts with the same length-k string as the previous suffix:

        increment CurrentCount

    else

        output CurrentCount and previous length-k suffix

        CurrentCount := 1

Output CurrentCount & length-k suffix.

Constructing Suffix Arrays

- Easy $O(n^2 \log n)$ algorithm:

  sort the n suffixes, which takes $O(n \log n)$ comparisons, where each comparison takes $O(n)$.

- There are several direct $O(n)$ algorithms for constructing suffix arrays that use very little space.
- The Skew Algorithm is one that is based on divide-and-conquer.

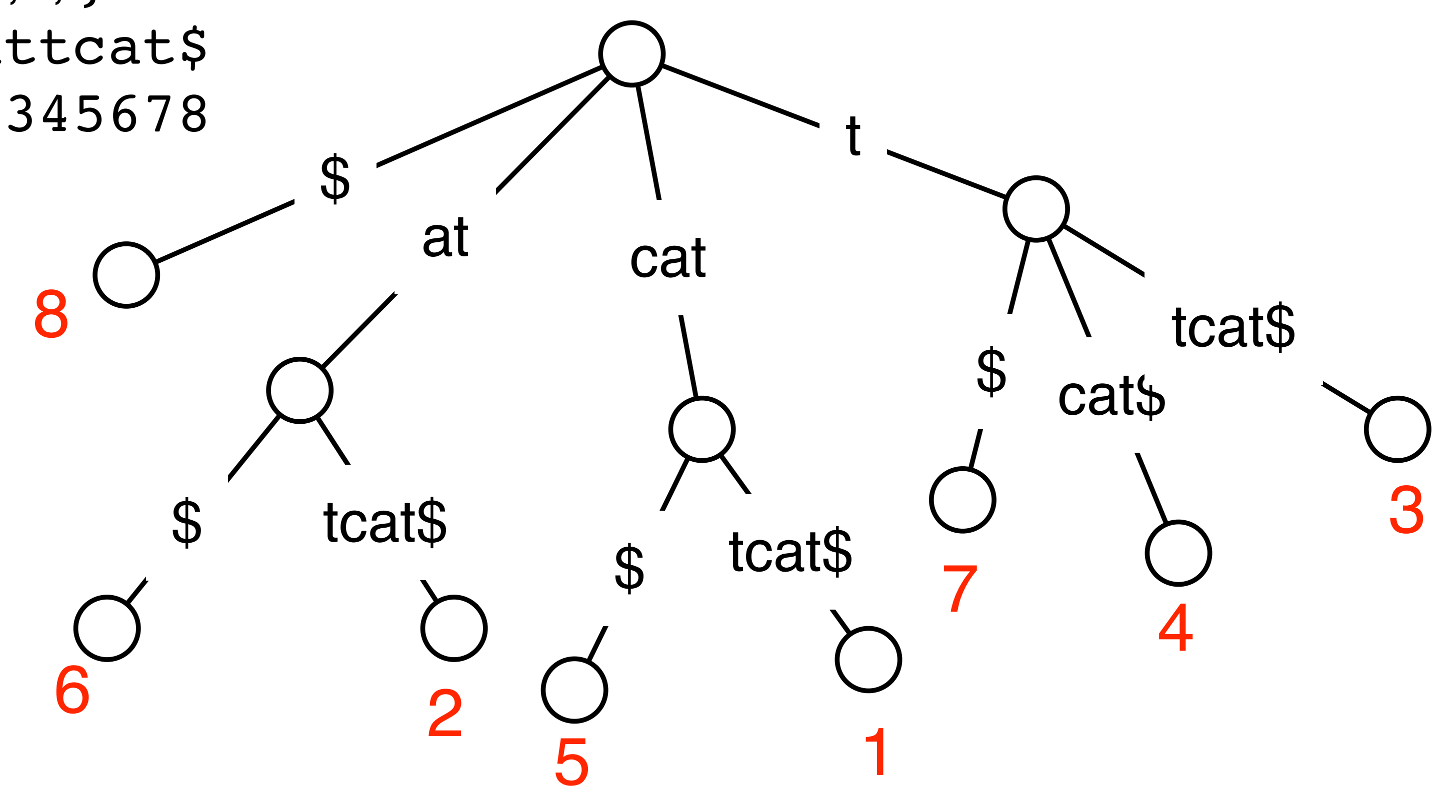
- An simple $O(n)$ algorithm: build the suffix tree, and exploit the relationship between suffix trees and suffix arrays (next slide)

Relationship Between Suffix Trees & Suffix Arrays

Σ = {$,a,c,t}

s = cattcat$

12345678



Red #s = starting position of the suffix ending at that leaf

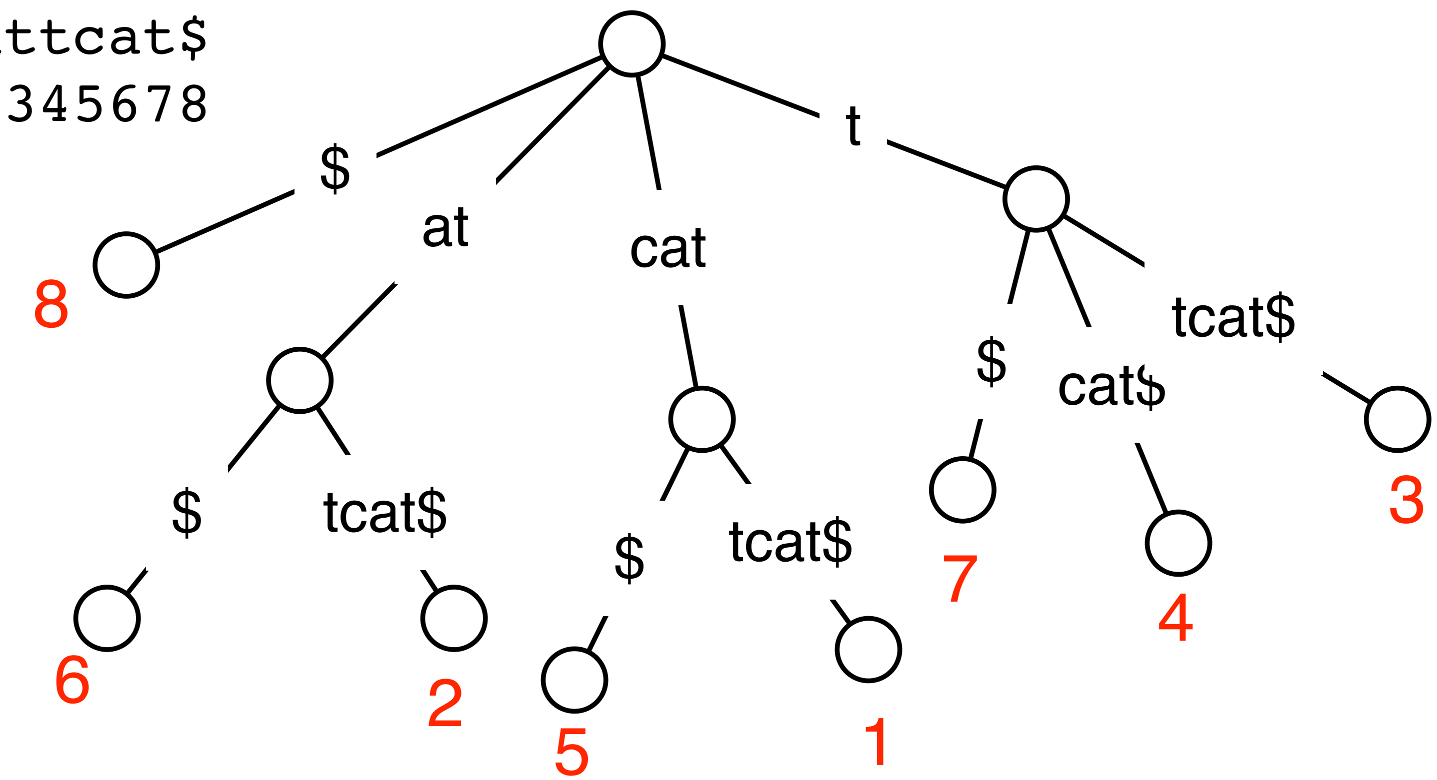
Leaf labels left to right: 86251743

Edges leaving each node are sorted by label (left-to-right).

Relationship Between Suffix Trees & Suffix Arrays

Σ = {$,a,c,t}

```
s = cattcat$
    12345678
```



s = cattcat$

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

Red #s = starting position of the suffix ending at that leaf

Edges leaving each node are sorted by label (left-to-right).

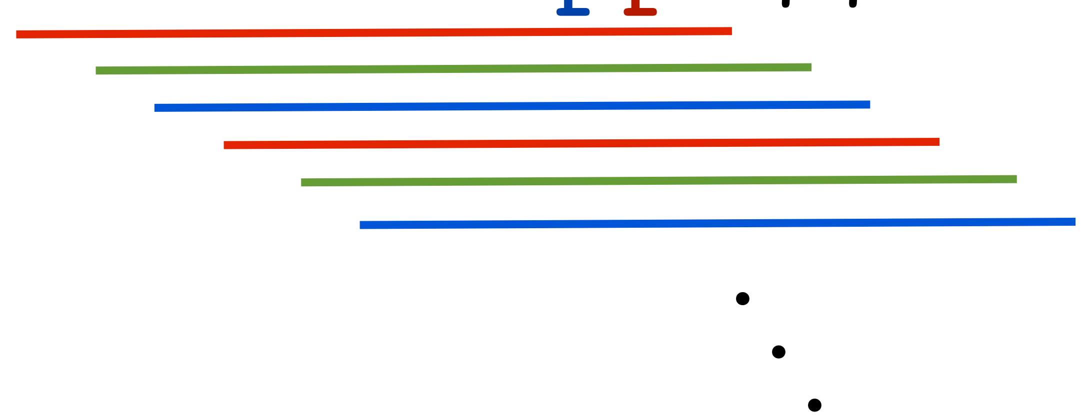
Leaf labels left to right: 86251743

The Skew Algorithm

Kärkkäinen & Sanders, 2003

- **Main idea: Divide suffixes into 3 groups:**
  - Those starting at positions i=0,3,6,9,.... (i mod 3 = 0)
  - Those starting at positions 1,4,7,10,... (i mod 3 = 1)
  - Those starting at positions 2,5,8,11,... (i mod 3 = 2)
- For simplicity, assume text length is a multiple of 3 after padding with a special character.

mississippi$$



Basic Outline:

- Recursively handle suffixes from the i mod 3 = 1 and i mod 3 = 2 groups.
- Merge the i mod 3 = 0 group at the end.

Handing the 1 and 2 groups

s = mississippi$$

iss	iss	ipp	i$$	ssi	ssi	ppi

triples for groups 1 and 2 groups

t = C C B A E E D

assign each triple a token in lexicographical order

AEED	4
BAEED	3
CBAEED	2
CCBAEED	1
D	7
ED	6
EED	5

recursively compute the suffix array for tokenized string

4321765

Every suffix of t corresponds to a suffix of s.

Relationship Between t and s

s = mississippi$$

iss	iss	ipp	i$$	ssi	ssi	ppi
C	C	B	A	E	E	D

t = CCBAEED

t_4

4321765

Key Point #1: The order of the suffixes of t is the same as the order of the group 1 & 2 suffixes of s.

Why?

Every suffix of t corresponds to some suffix of s (perhaps with some extra letters at the end of it --- in this case EED)

Because the tokens are sorted in the same order as the triples, the sort order of the suffix of t matches that of s.

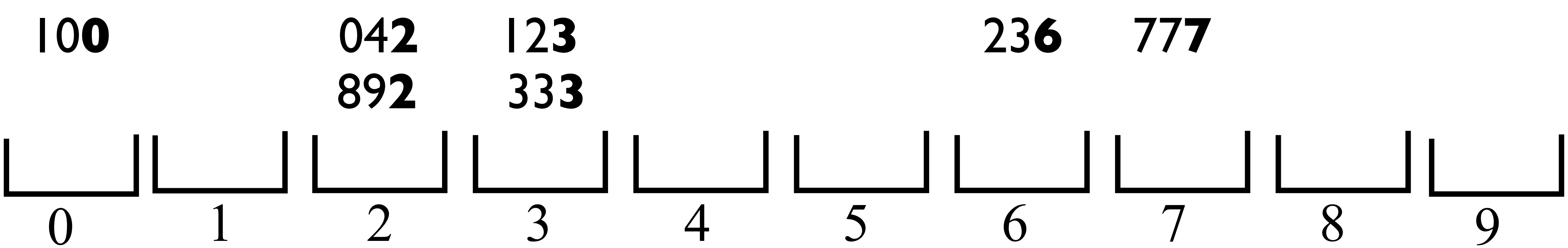
So: The recursive computational of the suffix array for t gives you the ordering of the group 1 and group 2 suffixes.

Radix Sort

- O(n)-time sort for n items when items can be divided into constant # of digits.

- Put into buckets based on least-significant digit, flatten, repeat with next-most significant digit, etc.

- Example items: 10**0** 12**3** 04**2** 33**3** 77**7** 89**2** 23**6**



- # of passes = # of digits
- Each pass goes through the numbers once.

Handling 0 Suffixes

- First: sort the group 0 suffixes, using the representation ($s[i]$, S_{i+1})
  - Since the S_{i+1} suffixes are already in the array sorted, we can just *stably* sort them with respect to $s[i]$, again using radix sort.

1,2-array:	ipp	iss	iss	i$$	ppi	ssi	ssi

0-array:	mis	pi$	sip	sis

- We have to merge the group 0 suffixes into the suffix array for group 1 and 2.
- Given suffix S_i and S_j, need to decide which should come first.
  - If S_i and S_j are both either group 1 or group 2, then the recursively computed suffix array gives the order.
  - If one of i or j is 0 (mod 3), see next slide.

Comparing 0 suffix S_j with 1 or 2 suffix S_i

Represent S_i and S_j using subsequent suffixes:

$i \pmod 3 = 1$:

$(s[i], S_{i+1}) \overset{?}{<} (s[j], S_{j+1})$

S_{i+1}: $\equiv 2 \pmod 3$ S_{j+1}: $\equiv 1 \pmod 3$

$i \pmod 3 = 2$:

$(s[i], s[i+1], S_{i+2}) \overset{?}{<} (s[j], s[j+1], S_{j+2})$

S_{i+2}: $\equiv 1 \pmod 3$ S_{j+2}: $\equiv 2 \pmod 3$

⇒ the suffixes can be compared quickly because the relative order of S_{i+1}, S_{j+1} or S_{i+2}, S_{j+2} is known from the 1,2-array we already computed.

Running Time

$$T(n) = O(n) + T(2n/3)$$

time to sort and merge

array in recursive calls is 2/3rds the size of starting array

Solves to $T(n) = O(n)$:

- Expand big-O notation: $T(n) \leq cn + T(2n/3)$ for some c.
- Guess: $T(n) \leq 3cn$
- Induction step: assume that is true for all $i < n$.
- $T(n) \leq cn + 3c(2n/3) = cn + 2cn = 3cn$ □

Recap

- Suffix arrays can be used to search and count substrings.

- Construction:
  - Easily constructed in $O(n^2 \log n)$
  - Simple algorithms to construct them in O(n) time.
  - More complicated algorithms to construct them in O(n) time using even less space.

- More space efficient than suffix trees: just storing the original string + a list of integers.