The Longest Common Prefix Array

Juha Kärkkäinen

University of Helsinki

Summer School on Bioinformatics Data Strucures
August 12, 2016

Outline

Outline

Suffix Array

$T =$ `ctaataatg`

SA	
2	`aataatg`
5	`aatg`
3	`ataatg`
6	`atg`
0	`ctaataatg`
8	`g`
1	`ctaataatg`
4	`taatg`
7	`tg`

Longest Common Prefix Array

LCP	SA	
	2	aataatg
3		aat
	5	aatg
1		a
	3	ataatg
2		at
	6	atg
0		
	0	ctaataatg
0		
	8	g
0		
	1	taataatg
4		taat
	4	taatg
1		t
	7	tg

Brief History of Full-Text Indexes [1/3]

Full-text index

- Data structure build from a string (text)
- Fast pattern matching
- Other applications: repeat finding, string statistics, ...

1973: Suffix tree [Peter Weiner, 1973]

- Linear size and construction
- Myriad applications
  - Bioinformatics [Gusfield, 1997]

Brief History of Full-Text Indexes [2/3]

1990: Suffix array [Manber & Myers, 1990, 1993]

- Simpler and lighter than suffix tree
- LCP array
  - Speed up pattern matching
- Limited other applications

2001: LCP array [Kasai & al., 2001]

- Simple and fast construction
- Suffix tree simulation
  - Many applications

2002: Enhanced suffix array [Abouelhoda & al., 2002, 2004]

- SA + LCP + ...
- Myriad applications [Ohlebusch, 2013]

Brief History of Full-Text Indexes [3/3]

2000: Compressed indexes [Ferragina & Manzini, 2000, 2005]

- Much smaller index size
- Limited applications at first
- More applications later [Mäkinen et al., 2015]

Which Full-Text Index Should I Use?

Suffix tree

- Conceptually powerful
- Big and slow

Enhanced suffix array

- Fast and simple
- Bigger than compressed indexes
  - External memory (disk based) algorithms often possible
- Conceptually trickier than suffix tree?
  $\Longrightarrow$ this talk

Compressed indexes

- Small but slow
- Very complicated

Which Full-Text Index Should I Use?

Recipe for solving a string analysis problem

1. Develop an algorithm using suffix tree
   - Gain intuition

Which Full-Text Index Should I Use?

Recipe for solving a string analysis problem

1. Develop an algorithm using suffix tree
   - Gain intuition
2. Develop an algorithm using enhanced suffix array
   - And implement it

Which Full-Text Index Should I Use?

Recipe for solving a string analysis problem

1. Develop an algorithm using suffix tree
   - Gain intuition
2. Develop an algorithm using enhanced suffix array
   - And implement it
3. Develop an algorithm using a compressed index
   - If needed because of big data

Construction

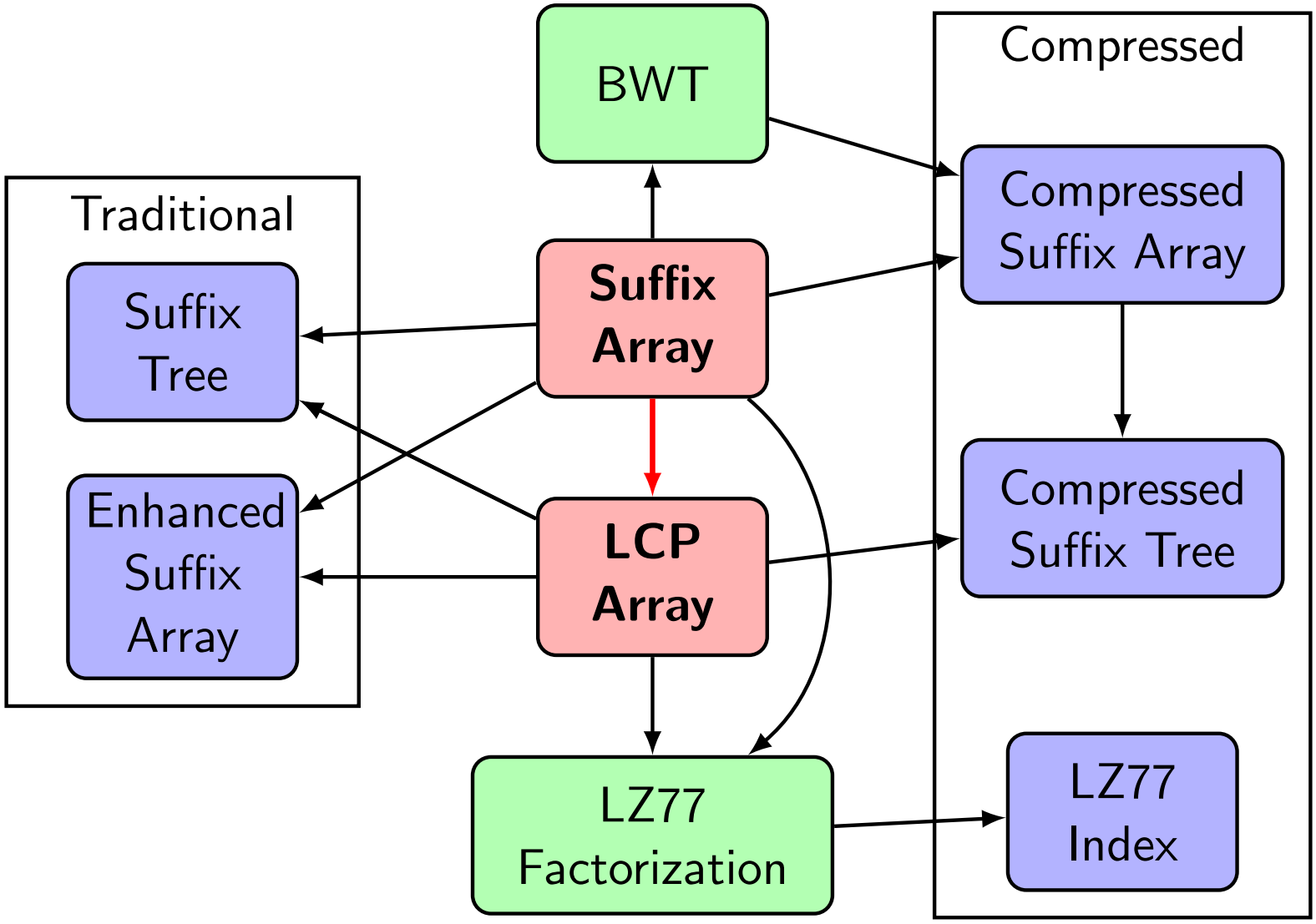
Suffix tree

- Fairly simple
- Big and slow
- Best way?: use suffix and LCP array!

Compressed indexes

- Complicated
  - Especially in small space
- Best way?: use suffix and LCP array

Construction



Construction

Suffix array

- Fast algorithms are complicated
  - Good public implementations
  - Standard output!
- External memory construction possible

LCP array

- Simple and fast
- External memory construction possible

`https://www.cs.helsinki.fi/group/pads/`

- Practical implementations for external memory suffix and LCP array construction

This Lecture

- Several algorithms on enhanced suffix arrays with the LCP array in the central role
- Selected properties of LCP arrays

Goal

- Show that LCP array is a powerful and sophisticated data structure in its own right
- Gain intuition on the LCP array

Outline

Longest Repeat

Problem: Longest repeat

Find the longest string that occurs at least twice in the text

```
 taat
ctaataatg
    taat
```

Longest Repeat

```
 taat
ctaataatg
    taat
```

LCP	SA	
	2	aataatg
3		aat
	5	aatg
1		a
	3	ataatg
2		at
	6	atg
0		
	0	ctaataatg
0		
	8	g
0		
	1	taataatg
4		taat
	4	taatg
1		t
	7	tg

Longest Repeat

taat
ctaataatg
taat

LCP	SA	
	2	aataatg
3		aat
	5	aatg
1		a
	3	ataatg
2		at
	6	atg
0		
	0	ctaataatg
0		
	8	g
0		
	1	taataatg
4		taat
	4	taatg
1		t
	7	tg

Longest Repeat

```
 taat
ctaataatg
    taat
```

LCP	SA	
	2	aataatg
3		aat
	5	aatg
1		a
	3	ataatg
2		at
	6	atg
0		
	0	ctaataatg
0		
	8	g
0		
	1	taataatg
4		taat
	4	taatg
1		t
	7	tg

Longest Repeat

Problem: Longest repeat

Find the longest string that occurs at least twice in the text

Solution

- Find the maximum value in LCP array
- The value is the length of the longest repeat
- The surrounding SA entries give the positions

Longest k-Repeat

Problem: Longest k-repeat

Find the longest string that occurs at least k times in the text

$k = 4$

```
ctaataatg
  aa aa
```

Longest k-Repeat

$k = 4$

ctaataatg
aa aa

LCP	SA	
	2	aataatg
3		aat
	5	aatg
1		a
	3	ataatg
2		at
	6	atg
0		
	0	ctaataatg
0		
	8	g
0		
	1	taataatg
4		taat
	4	taatg
1		t
	7	tg

Longest k-Repeat

$k = 4$

ctaataatg
aa aa

LCP	SA	
	2	aataatg
3		aat
	5	atg
1		a
	3	ataatg
2		at
	6	atg
0		
	0	ctaataatg
0		
	8	g
0		
	1	taataatg
4		taat
	4	taatg
1		t
	7	tg

Longest k-Repeat

$k = 4$

ctaataatg
aa aa

LCP	SA	
	2	aataatg
3		aat
	5	aatg
1		a
	3	ataatg
2		at
	6	atg
0		
	0	ctaataatg
0		
	8	g
0		
	1	taataatg
4		taat
	4	taatg
1		t
	7	tg

Longest k-Repeat

Problem: Longest k-repeat

Find the longest string that occurs at least k times in the text

Solution

- Find the LCP array interval of length $k-1$ with the largest minimum value
- The minum value is the length of the k-repeat
- The corresponding SA interval gives the positions

Longest k-Repeat

Problem: Longest k-repeat

Find the longest string that occurs at least k times in the text

Solution

- Find the LCP array interval of length $k-1$ with the largest minimum value
- The minum value is the length of the k-repeat
- The corresponding SA interval gives the positions

- The interval may be extended as long as the minimum LCP value does not change
  - More than k occurrences

Most Frequent k-Mer

Problem: Most frequent k-mer

Find the k-mer with most occurrences in the text

$k = 2$

```
   ta   ta
ctaatatag
     ta
```

Most Frequent k-Mer

$k = 2$

ta ta
ctaatatag
ta

LCP	SA	
	2	aatatag
1		a
	7	ag
1		a
	5	atag
3		ata
	3	atatag
0		
	0	ctaatatag
0		
	8	g
0		
	1	taatatag
2		ta
	6	tag
2		ta
	4	tatag

Most Frequent k-Mer

$k = 2$

ta ta
ctaatatag
ta

LCP	SA	
	2	aatatag
1		a
	7	ag
1		a
	5	atag
3		ata
	3	atatag
0		
	0	ctaatatag
0		
	8	g
0		
	1	taatatag
2		ta
	6	tag
2		ta
	4	tatag

Most Frequent k-Mer

$k = 2$

ta ta
ctaatatag
ta

LCP	SA	
	2	aatatag
1		a
	7	ag
1		a
	5	atag
3		ata
	3	atatag
0		
	0	ctaatatag
0		
	8	g
0		
	1	taatatag
2		ta
	6	tag
2		ta
	4	tatag

Most Frequent k-Mer

Problem: Most frequent k-mer

Find the k-mer with most occurrences in the text

Solution

- Find the longest LCP array interval where the minimum is at least k

Distinct k-mers

Problem: Distinct k-mers

Count the number of distinct k-mers in the text

Distinct k-mers

$k = 2$ ctaatatag

LCP	SA	
	2	aatatag
1		a
	7	ag
1		a
	5	atag
3		ata
	3	atatag
0		
	0	ctaatatag
0		
	8	g
0		
	1	taatatag
2		ta
	6	tag
2		ta
	4	tatag

Distinct k-mers

$k = 2$ ctaatatag

LCP	SA	
	2	aatatag
1		a
	7	ag
1		a
	5	atag
3		ata
	3	atatag
0		
	0	ctaatatag
0		
	8	g
0		
	1	taatatag
2		ta
	6	tag
2		ta
	4	tatag

Distinct k-mers

Problem: Distinct k-mers

Count the number of distinct k-mers in the text

Solution

- Count the number of LCP array entries smaller than k
- Subtract $k - 2$
- The result is the distinct k-mer count

k-Mer Frequencies

Problem: k-mer frequencies

List all the distinct k-mers in the text and their frequencies

- Exercise

Distinct Substrings

Problem: Distinct substrings

Count the number of distinct substrings in the text

Distinct Substrings

Problem: Distinct substrings

Count the number of distinct substrings in the text

Solution

$$\frac{n(n+1)}{2} - \sum_i LCP[i]$$

Longest Common Substring

Problem: Longest common substring

Given two texts, find the longest string that occurs in both

$$S = \texttt{tatat} \qquad T = \texttt{atg}$$
$$\texttt{at} \qquad\qquad \texttt{at}$$

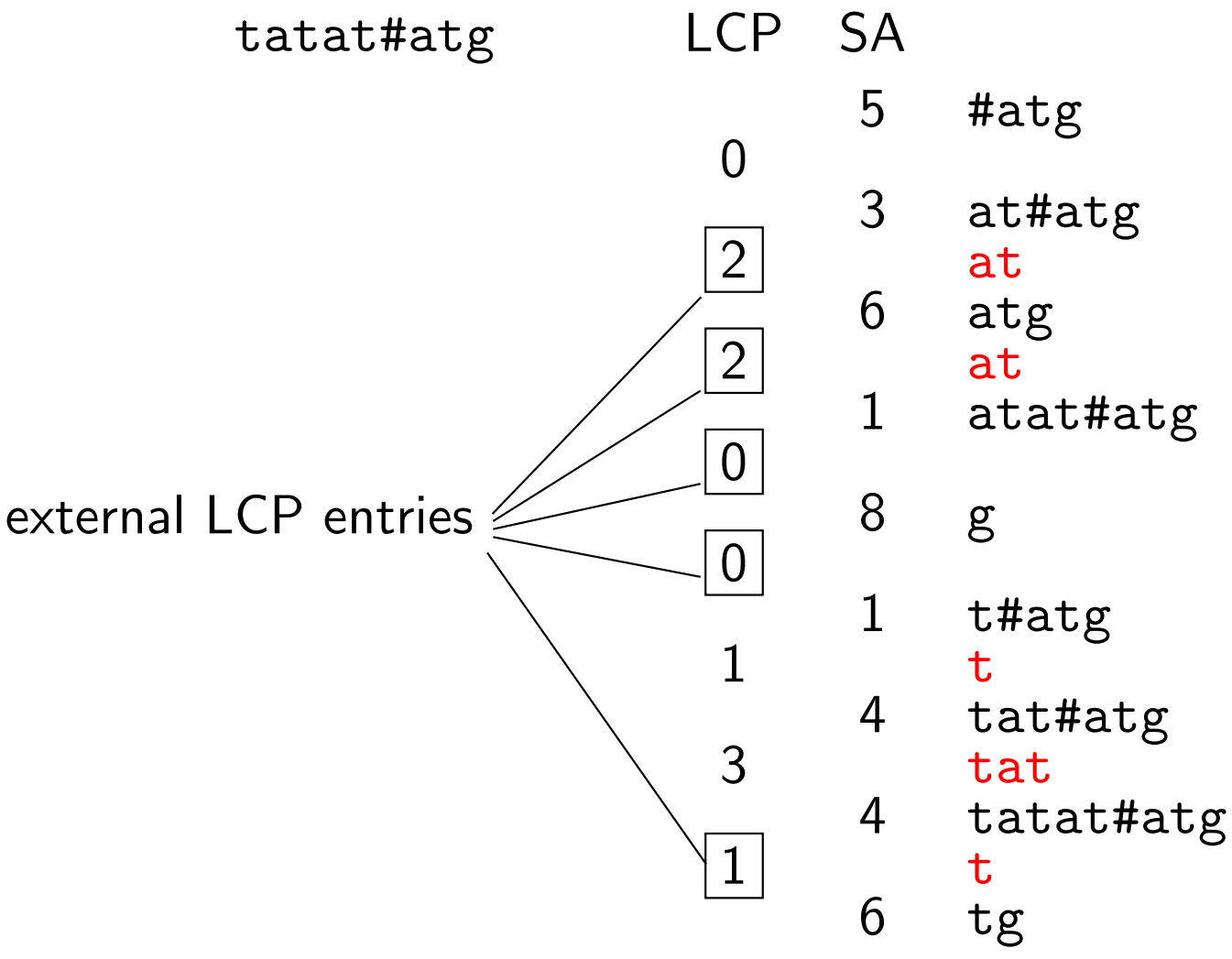
Use the generalized suffix and LCP arrays, i.e.,
SA and LCP for the concatenation $S\#T = \texttt{tatat\#atg}$

Longest Common Substring

tatat#atg

LCP	SA	
	5	#atg
0		
	3	at#atg
2		at
	6	atg
2		at
	1	atat#atg
0		
	8	g
0		
	1	t#atg
1		t
	4	tat#atg
3		tat
	4	tatat#atg
1		t
	6	tg

Longest Common Substring



Longest Common Substring

Problem: Longest common substring

Given two texts, find the longest string that occurs in both

Solution

- Find the biggest external value in the generalized LCP array

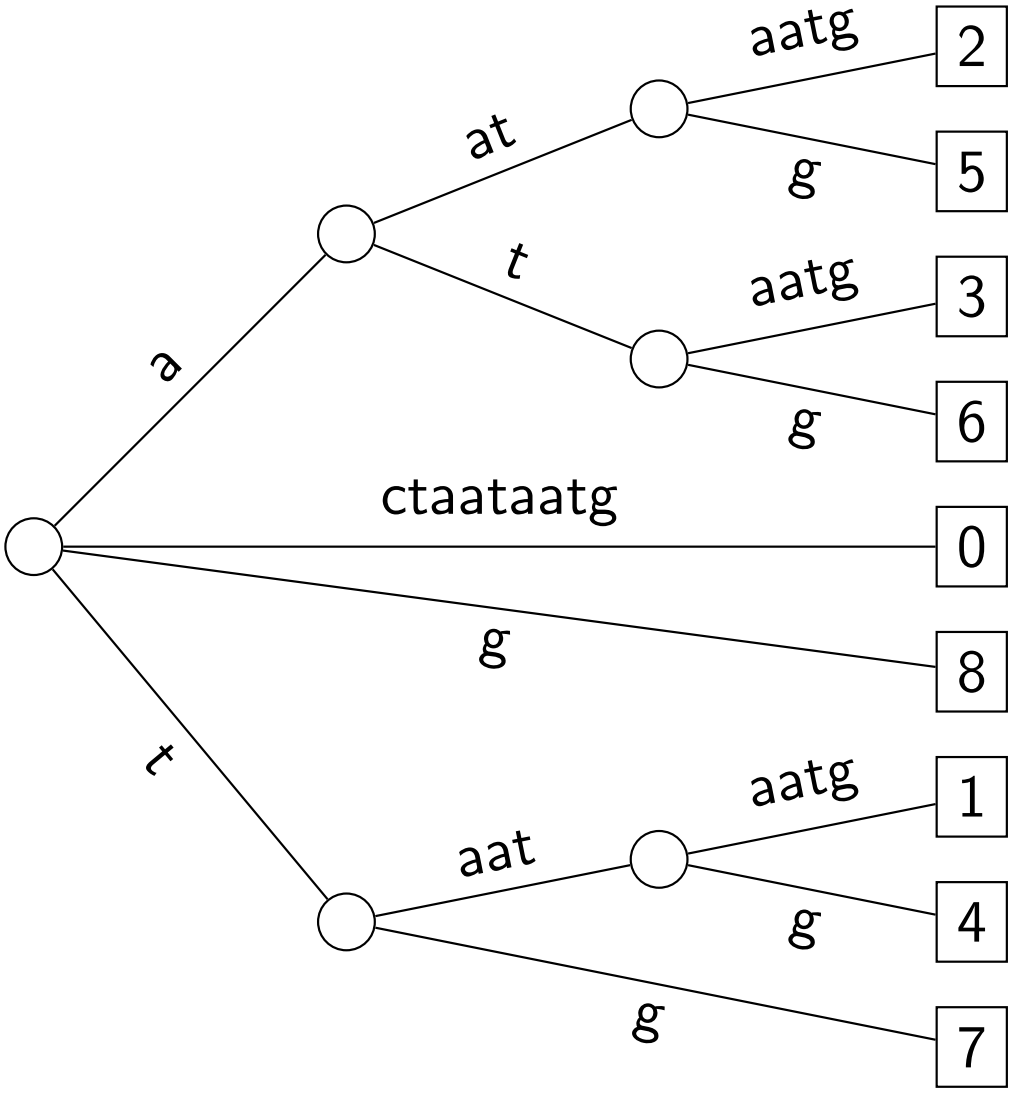
Summary

All the solutions so far

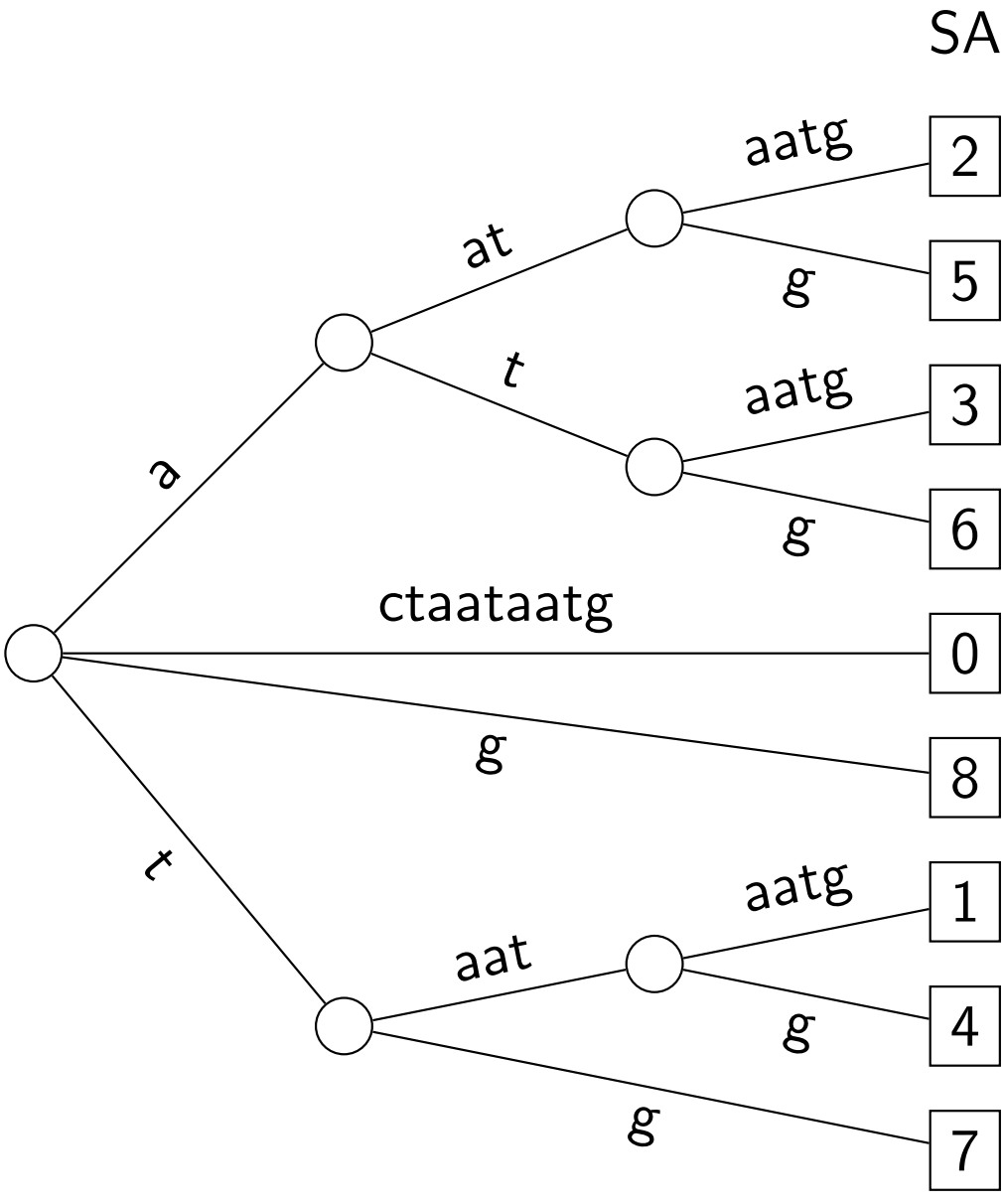
- A single scan over the LCP array (and sometimes SA)
- Constant extra space
- Only a few lines of code

Outline

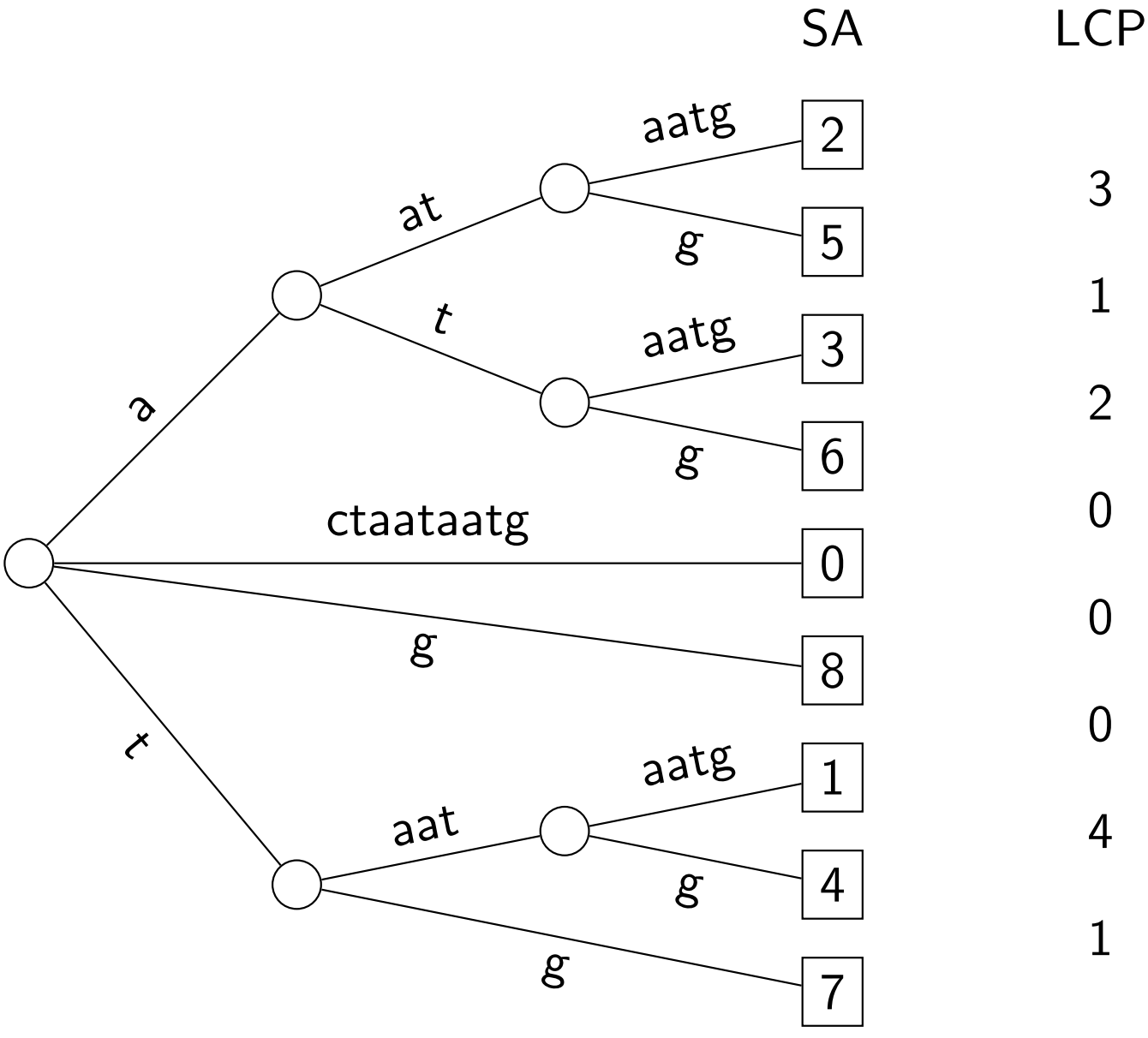
Suffix Tree



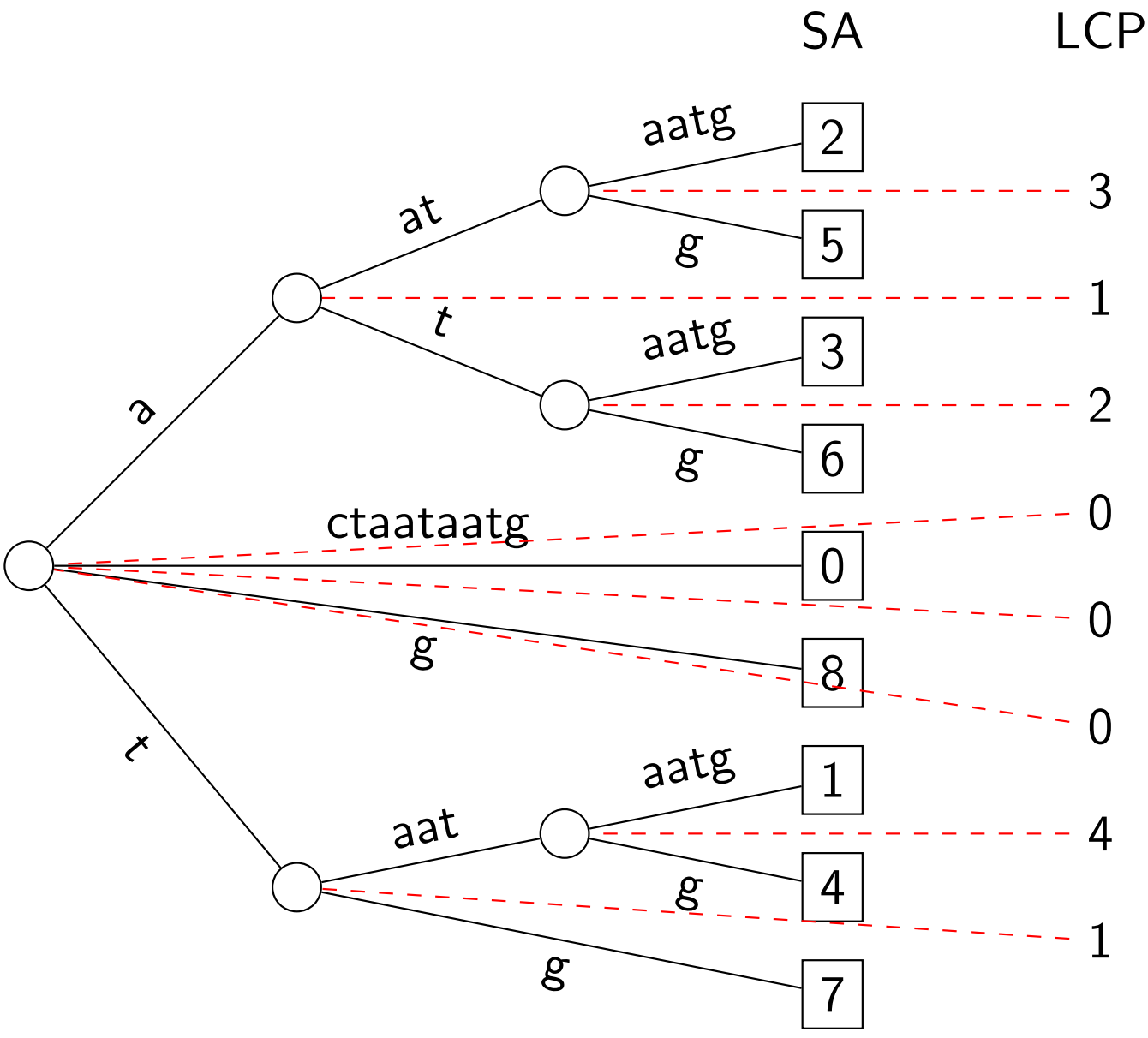
Suffix Tree



Suffix Tree


SA
LCP
aatg
2
3
at
g
5
1
t
aatg
3
2
a
g
6
ctaataatg
0
0
g
8
0
t
aatg
1
4
aat
g
4
1
g
7

Suffix Tree


SA
LCP
aatg
2
3
at
g
5
1
t
aatg
3
2
a
g
6
ctaataatg
0
0
g
8
0
t
aatg
1
4
aat
g
4
1
g
7

Constructing Suffix Tree from SA and LCP

- Insert suffixes in lexicographical order (using SA)
- LCP entries tell the depths of insertion points

Constructing Suffix Tree from SA and LCP

- Insert suffixes in lexicographical order (using SA)
- LCP entries tell the depths of insertion points

SA LCP

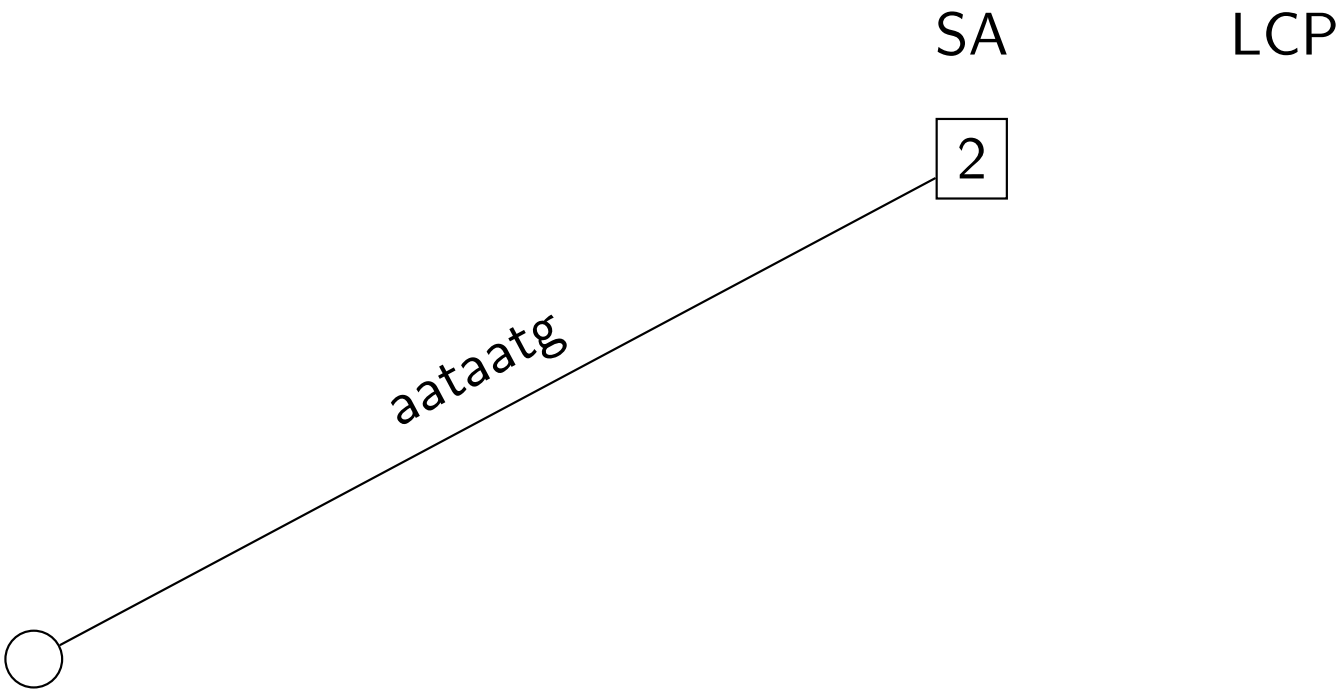
Constructing Suffix Tree from SA and LCP

- Insert suffixes in lexicographical order (using SA)
- LCP entries tell the depths of insertion points

SA	LCP
2	

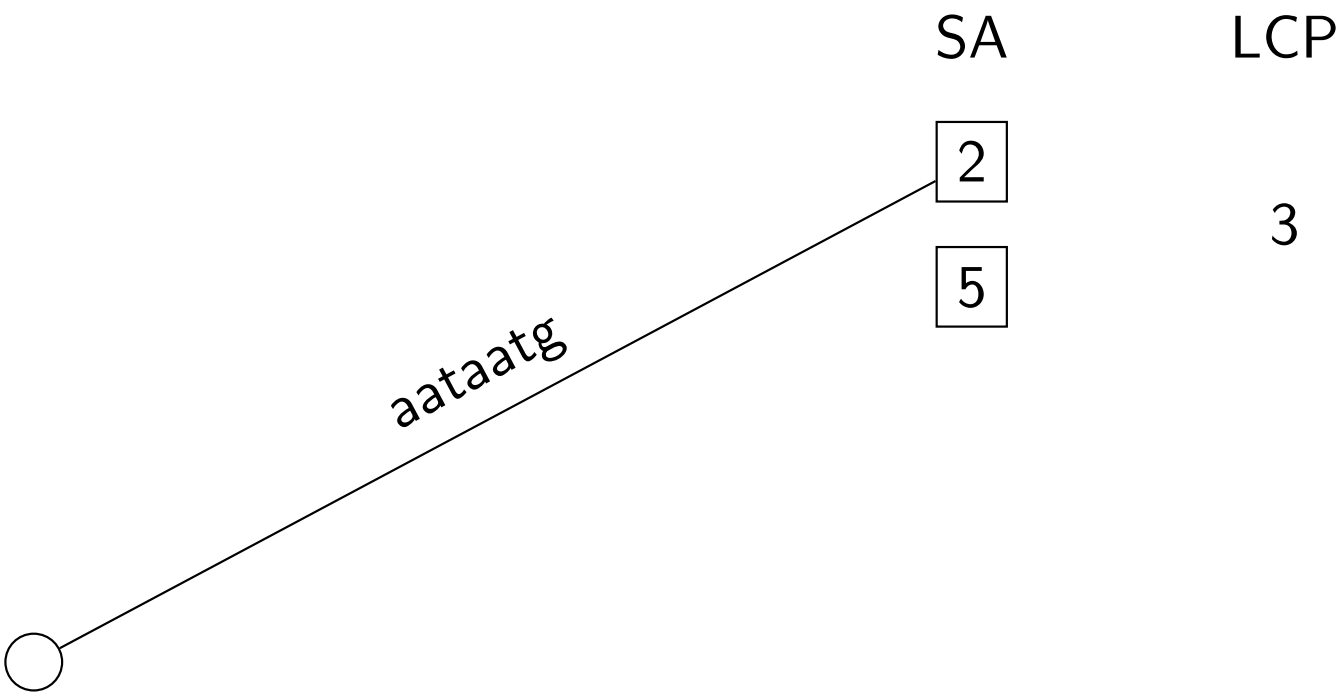
Constructing Suffix Tree from SA and LCP

- Insert suffixes in lexicographical order (using SA)
- LCP entries tell the depths of insertion points



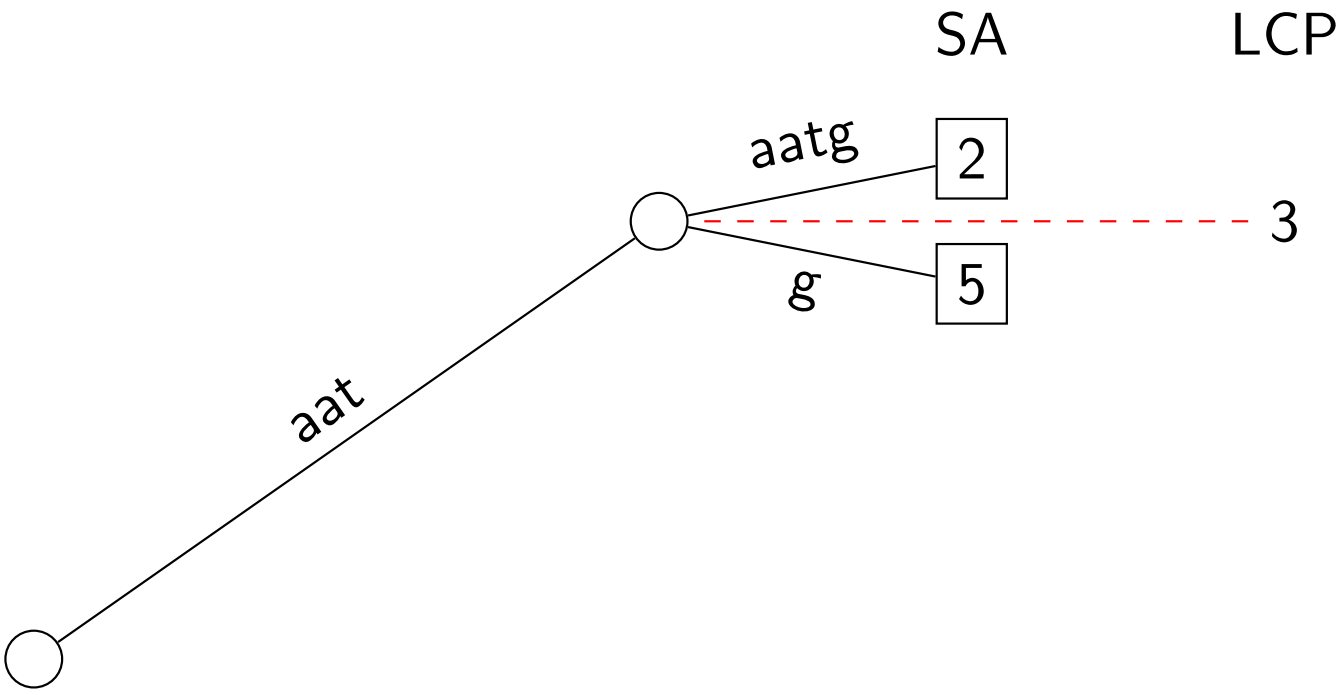
Constructing Suffix Tree from SA and LCP

- Insert suffixes in lexicographical order (using SA)
- LCP entries tell the depths of insertion points



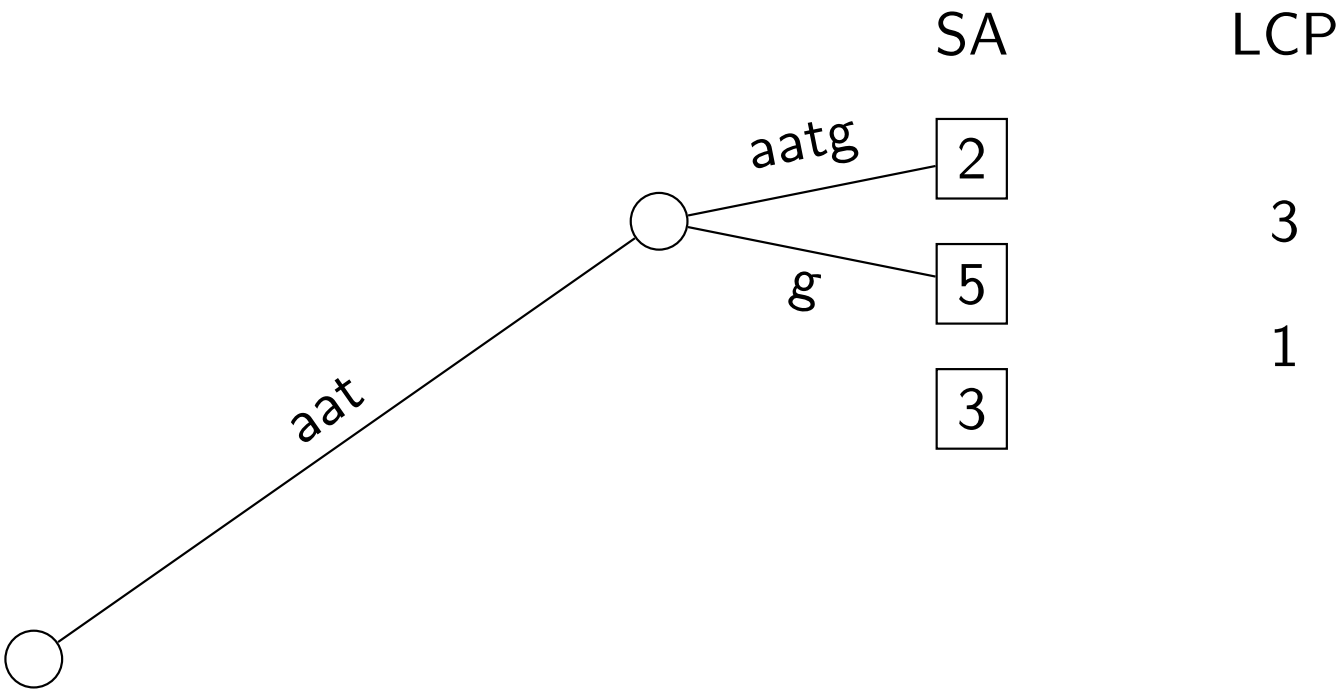
Constructing Suffix Tree from SA and LCP

- Insert suffixes in lexicographical order (using SA)
- LCP entries tell the depths of insertion points



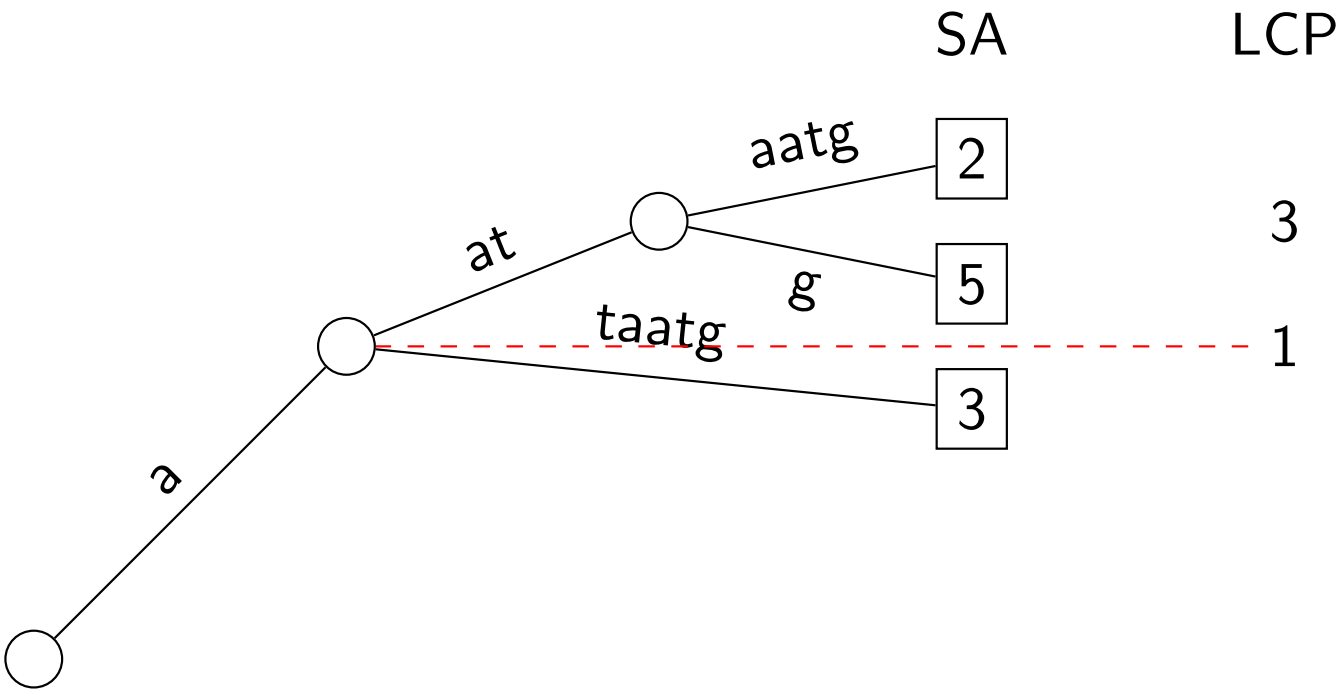
Constructing Suffix Tree from SA and LCP

- Insert suffixes in lexicographical order (using SA)
- LCP entries tell the depths of insertion points



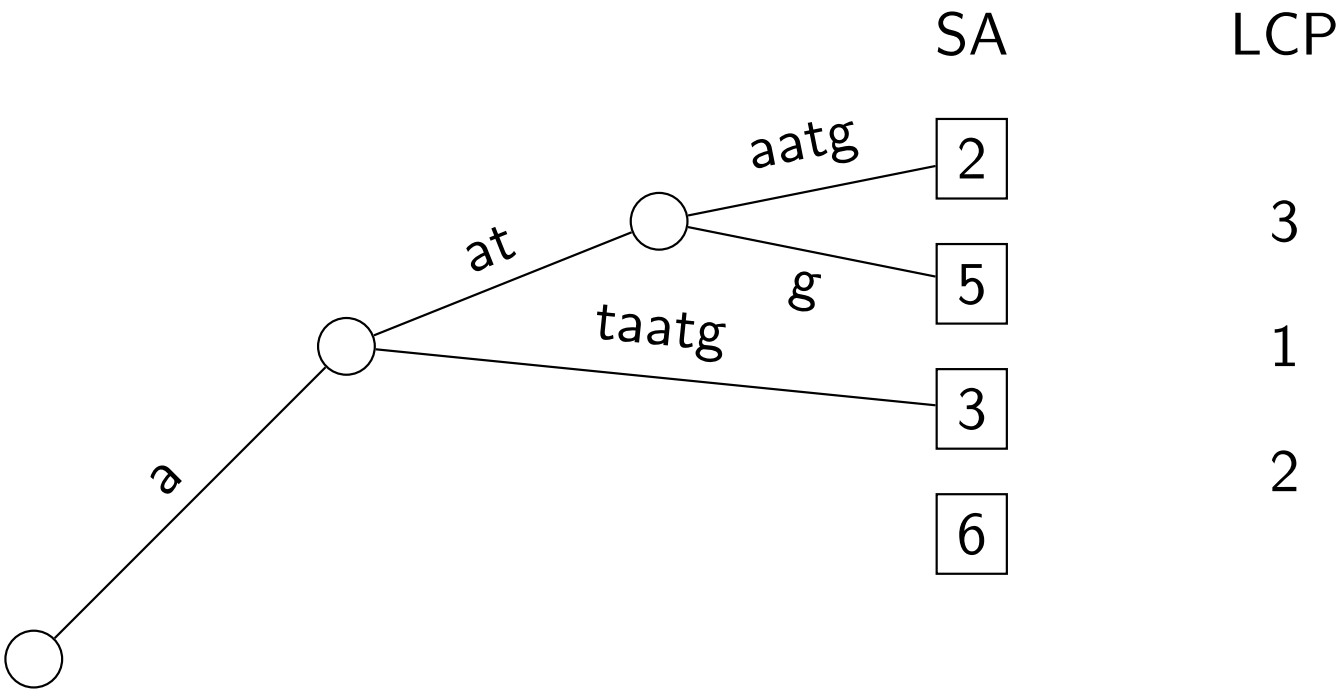
Constructing Suffix Tree from SA and LCP

- Insert suffixes in lexicographical order (using SA)
- LCP entries tell the depths of insertion points



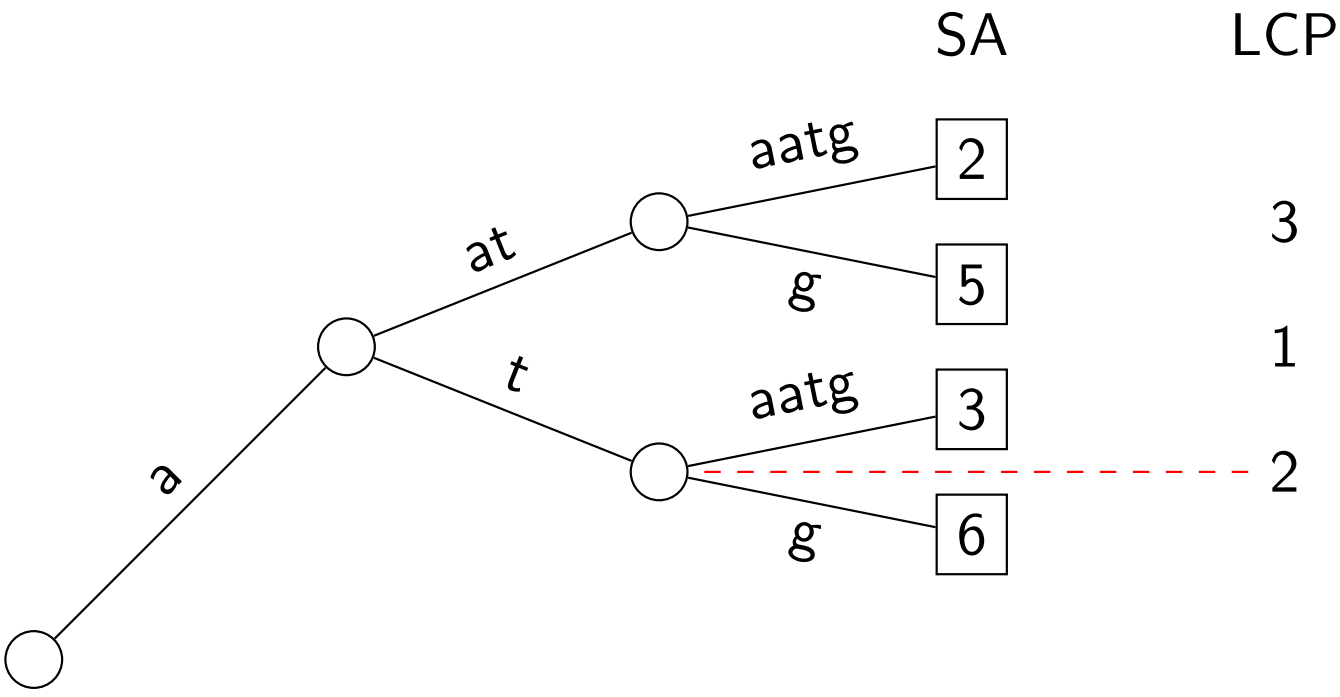
Constructing Suffix Tree from SA and LCP

- Insert suffixes in lexicographical order (using SA)
- LCP entries tell the depths of insertion points



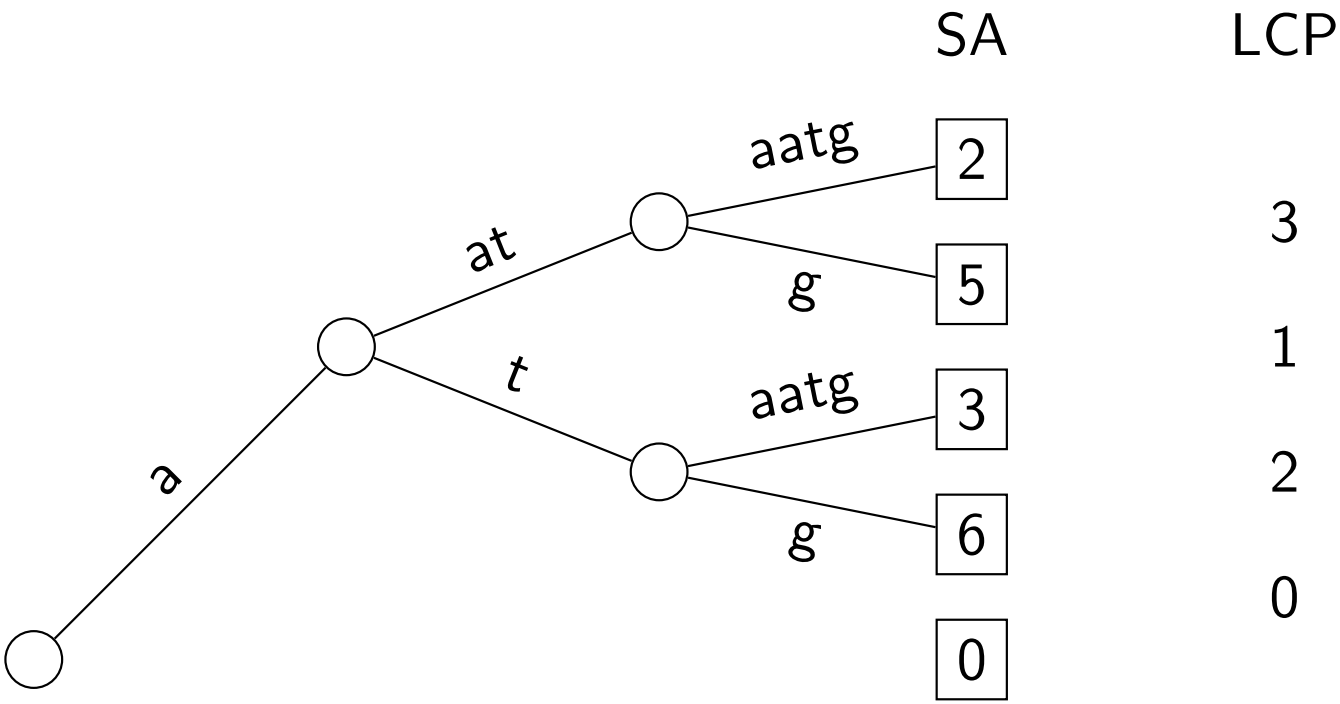
Constructing Suffix Tree from SA and LCP

- Insert suffixes in lexicographical order (using SA)
- LCP entries tell the depths of insertion points



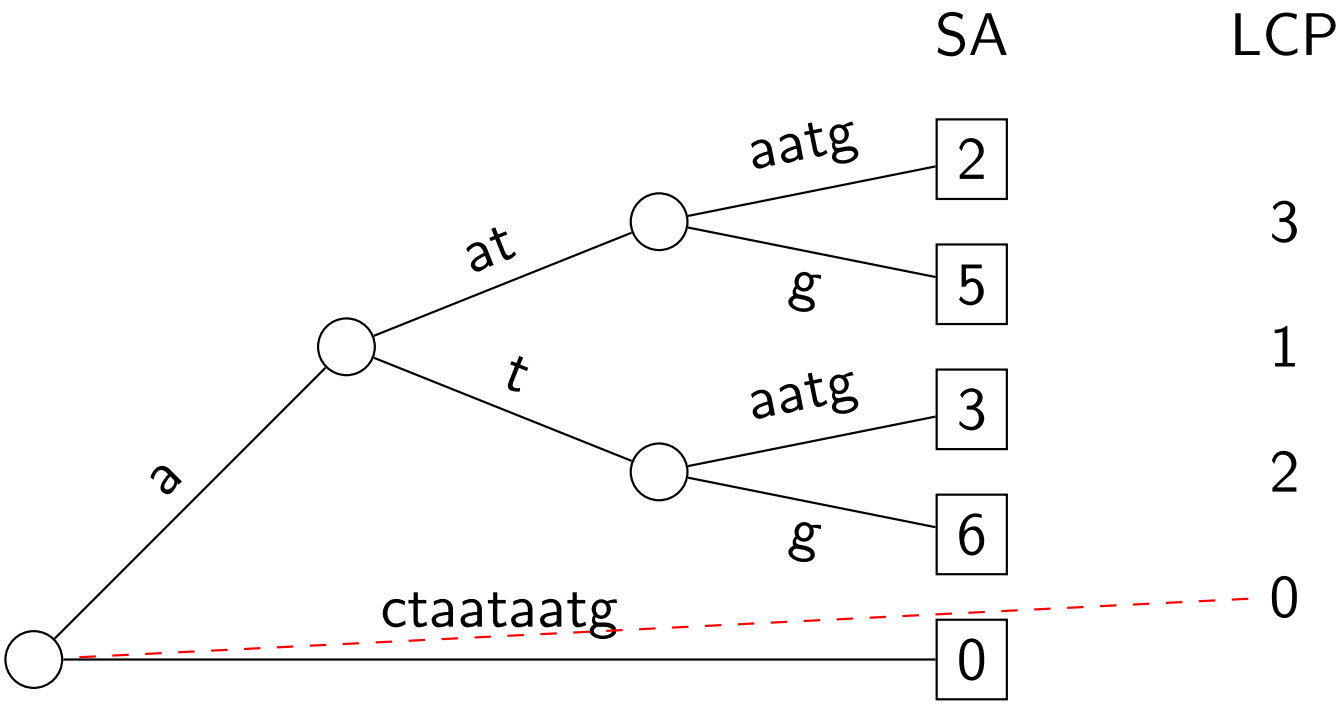
Constructing Suffix Tree from SA and LCP

- Insert suffixes in lexicographical order (using SA)
- LCP entries tell the depths of insertion points



Constructing Suffix Tree from SA and LCP

- Insert suffixes in lexicographical order (using SA)
- LCP entries tell the depths of insertion points



Right-Maximal Repeats

Definition: Right-maximal repeat

A string w is a right-maximal repeat if

1. w occurs at least twice, and
2. for all symbols α, $w\alpha$ has less occurrences than w

- aat is a right-maximal repeat

```
  aata
ctaataatg
    aatg
```

- taa is *not* a right-maximal repeat because taat has the same number of occurrences

```
 taat
ctaataatg
   taat
```

Right-Maximal Repeats

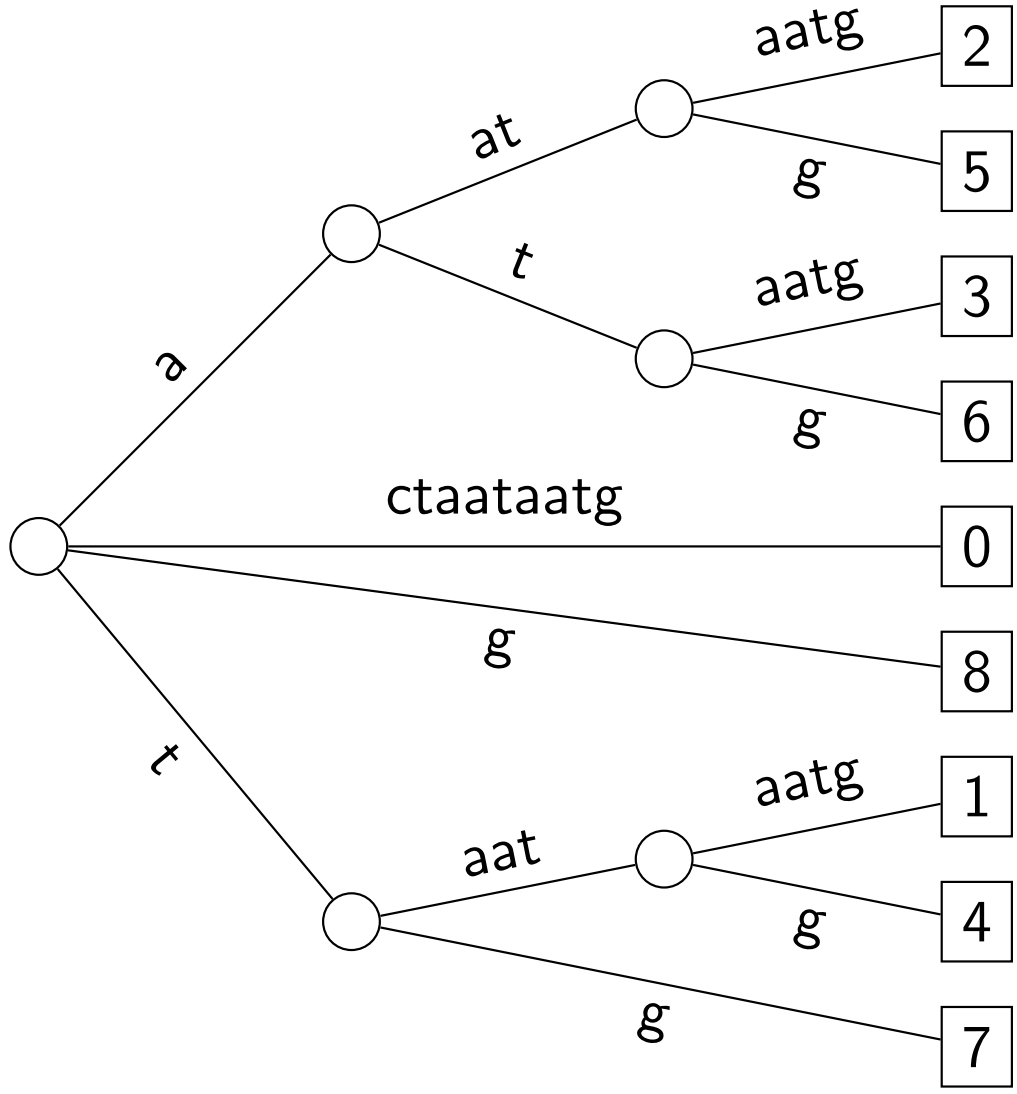
All right-maximal
repeats in `ctaataatgt`

- `aat`
- `at`
- `a`
- `taat`
- `t`
- ϵ (empty string)

Right-Maximal Repeats

All right-maximal repeats in `ctaataatgt`

- `aat`
- `at`
- `a`
- `taat`
- `t`
- ϵ (empty string)



Right-Maximal Repeats

Problem: Right-maximal repeats

List all right-maximal repeats

- Corresponds to enumerating all internal nodes of suffix tree

Right-Maximal Repeats

All right-maximal repeats in ctaataatgt

- aat
- at
- a
- taat
- t
- ϵ (empty string)

LCP	SA	
	2	aataatg
3		aat
	5	aatg
1		a
	3	ataatg
2		at
	6	atg
0		
	0	ctaataatg
0		
	8	g
0		
	1	taataatg
4		taat
	4	taatg
1		t
	7	tg

Right-Maximal Repeats

All right-maximal repeats in ctaataatgt

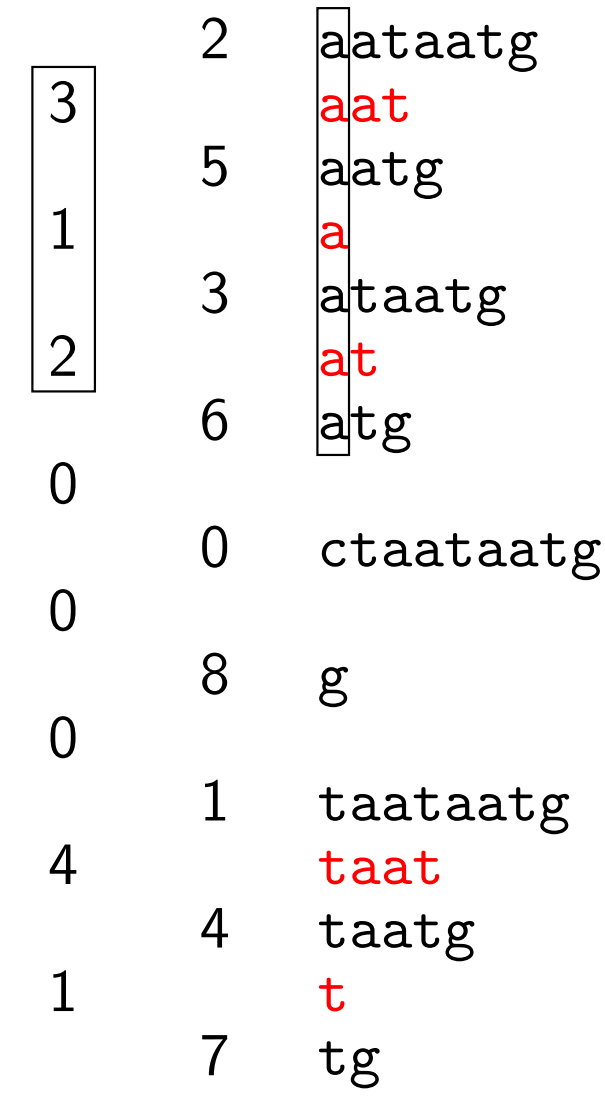
- aat
- at
- a
- taat
- t
- ϵ (empty string)

LCP	SA	
	2	aataatg
3		aat
	5	aatg
1		a
	3	ataatg
2		at
	6	atg
0		
	0	ctaataatg
0		
	8	g
0		
	1	taataatg
4		taat
	4	taatg
1		t
	7	tg

Right-Maximal Repeats

All right-maximal repeats in ctaataatgt

- aat
- at
- a
- taat
- t
- ϵ (empty string)



Right-Maximal Repeats

Definition: ℓ-interval

An interval $[i..j]$ is an ℓ-interval if

1. $\min LCP[i+1..j] = \ell$
2. $LCP[i] < \ell$
3. $LCP[j+1] < \ell$

An ℓ-interval $[i..j]$ corresponds to a right-maximal repeat

- ℓ is the length of the repeat
- $SA[i..j]$ contains the occurrence positions

Right-Maximal Repeats

All right-maximal repeats in `ctaataatgt`

- `aat`
- `at`
- `a`
- `taat`
- `t`
- ϵ (empty string)

LCP	SA	
-1		
	2	aataatg
3		aat
	5	aatg
1		a
	3	ataatg
2		at
	6	atg
0		
	0	ctaataatg
0		
	8	g
0		
	1	taataatg
4		taat
	4	taatg
1		t
	7	tg
-1		

Right-Maximal Repeats

Problem: Right-maximal repeats

List all right-maximal repeats

Solution

- Find all ℓ-intervals
- Can be done by scanning LCP array while maintaining a stack

Right-Maximal Repeats

Compare the next LCP value to top of stack

- if bigger, push and advance
- if smaller, pop and stay
- if equal, advance

Every pop forms an ℓ-interval from top of stack (after pop) to current position

stack
-1,0

LCP	SA	
-1		
*	2	aataatg
3		aat
	5	aatg
1		a
	3	ataatg
2		at
	6	atg
0		
	0	ctaataatg
0		
	8	g
0		
	1	taataatg
4		taat
	4	taatg
1		t
	7	tg
-1		

Right-Maximal Repeats

Compare the next LCP value to top of stack

- if bigger, push and advance
- if smaller, pop and stay
- if equal, advance

Every pop forms an ℓ-interval from top of stack (after pop) to current position

stack
-1,0
3,1

LCP	SA	
-1		
	2	aataatg
3		aat
*	5	aatg
1		a
	3	ataatg
2		at
	6	atg
0		
	0	ctaataatg
0		
	8	g
0		
	1	taataatg
4		taat
	4	taatg
1		t
	7	tg
-1		

Right-Maximal Repeats

Compare the next LCP value to top of stack

- if bigger, push and advance
- if smaller, pop and stay
- if equal, advance

Every pop forms an ℓ-interval from top of stack (after pop) to current position

stack
-1,0

LCP	SA	
-1		
	2	aataatg
3		aat
*	5	aatg
1		a
	3	ataatg
2		at
	6	atg
0		
	0	ctaataatg
0		
	8	g
0		
	1	taataatg
4		taat
	4	taatg
1		t
	7	tg
-1		

Right-Maximal Repeats

Compare the next LCP value to top of stack

- if bigger, push and advance
- if smaller, pop and stay
- if equal, advance

Every pop forms an ℓ-interval from top of stack (after pop) to current position

stack
-1,0

LCP	SA	
-1		
	2	aataatg
3		aat
*	5	aatg
1		a
	3	ataatg
2		at
	6	atg
0		
	0	ctaataatg
0		
	8	g
0		
	1	taataatg
4		taat
	4	taatg
1		t
	7	tg
-1		

Right-Maximal Repeats

Compare the next LCP value to top of stack

- if bigger, push and advance
- if smaller, pop and stay
- if equal, advance

Every pop forms an ℓ-interval from top of stack (after pop) to current position

stack
-1,0
1,2

LCP	SA	
-1		
	2	aataatg
3		aat
	5	aatg
1		a
*	3	ataatg
2		at
	6	atg
0		
	0	ctaataatg
0		
	8	g
0		
	1	taataatg
4		taat
	4	taatg
1		t
	7	tg
-1		

Right-Maximal Repeats

Compare the next LCP value to top of stack

- if bigger, push and advance
- if smaller, pop and stay
- if equal, advance

Every pop forms an ℓ-interval from top of stack (after pop) to current position

stack
-1,0
1,2
2,3

LCP	SA	
-1		
	2	aataatg
3		aat
	5	aatg
1		a
	3	ataatg
2		at
*	6	atg
0		
	0	ctaataatg
0		
	8	g
0		
	1	taataatg
4		taat
	4	taatg
1		t
	7	tg
-1		

Right-Maximal Repeats

Compare the next LCP value to top of stack

- if bigger, push and advance
- if smaller, pop and stay
- if equal, advance

Every pop forms an ℓ-interval from top of stack (after pop) to current position

stack
-1,0
1,2

LCP	SA	
-1		
	2	aataatg
3		aat
	5	aatg
1		a
	3	ataatg
2		at
*	6	atg
0		
	0	ctaataatg
0		
	8	g
0		
	1	taataatg
4		taat
	4	taatg
1		t
	7	tg
-1		

Right-Maximal Repeats

Compare the next LCP value to top of stack

- if bigger, push and advance
- if smaller, pop and stay
- if equal, advance

Every pop forms an ℓ-interval from top of stack (after pop) to current position

stack
-1,0
1,2

LCP	SA	
-1		
	2	aataatg
3		aat
	5	aatg
1		a
	3	ataatg
2		at
*	6	atg
0		
	0	ctaataatg
0		
	8	g
0		
	1	taataatg
4		taat
	4	taatg
1		t
	7	tg
-1		

Right-Maximal Repeats

Compare the next LCP value to top of stack

- if bigger, push and advance
- if smaller, pop and stay
- if equal, advance

Every pop forms an ℓ-interval from top of stack (after pop) to current position

stack
-1,0

LCP	SA	
-1		
	2	aataatg
3		aat
	5	aatg
1		a
	3	ataatg
2		at
*	6	atg
0		
	0	ctaataatg
0		
	8	g
0		
	1	taataatg
4		taat
	4	taatg
1		t
	7	tg
-1		

Right-Maximal Repeats

Compare the next LCP value to top of stack

- if bigger, push and advance
- if smaller, pop and stay
- if equal, advance

Every pop forms an ℓ-interval from top of stack (after pop) to current position

stack

-1,0

LCP	SA	
-1		
	2	aataatg
3		aat
	5	aatg
1		a
	3	ataatg
2		at
*	6	atg
0		
	0	ctaataatg
0		
	8	g
0		
	1	taataatg
4		taat
	4	taatg
1		t
	7	tg
-1		

Right-Maximal Repeats

Compare the next LCP value to top of stack

- if bigger, push and advance
- if smaller, pop and stay
- if equal, advance

Every pop forms an ℓ-interval from top of stack (after pop) to current position

stack
-1,0
0,4

LCP	SA	
-1		
	2	aataatg
3		aat
	5	aatg
1		a
	3	ataatg
2		at
	6	atg
0		
*	0	ctaataatg
0		
	8	g
0		
	1	taataatg
4		taat
	4	taatg
1		t
	7	tg
-1		

Right-Maximal Repeats

Compare the next LCP value to top of stack

- if bigger, push and advance
- if smaller, pop and stay
- if equal, advance

Every pop forms an ℓ-interval from top of stack (after pop) to current position

stack
-1,0
0,4

LCP	SA	
-1		
	2	aataatg
3		aat
	5	aatg
1		a
	3	ataatg
2		at
	6	atg
0		
	0	ctaataatg
0		
*	8	g
0		
	1	taataatg
4		taat
	4	taatg
1		t
	7	tg
-1		

Right-Maximal Repeats

Compare the next LCP value to top of stack

- if bigger, push and advance
- if smaller, pop and stay
- if equal, advance

Every pop forms an ℓ-interval from top of stack (after pop) to current position

stack
-1,0
0,4

LCP	SA	
-1		
	2	aataatg
3		aat
	5	aatg
1		a
	3	ataatg
2		at
	6	atg
0		
	0	ctaataatg
0		
	8	g
0		
*	1	taataatg
4		taat
	4	taatg
1		t
	7	tg
-1		

Summary

- LCP array encodes the tree structure of the suffix tree
- Easy construction of suffix tree
- Enumeration of internal nodes
  - Solves many problems without constructing suffix tree
  - Single scan of LCP array

Outline

Maximal Repeats

Definition: Maximal repeat

A string w is a maximal repeat if

1. w occurs at least twice (repeat)
2. for all symbols α, $w\alpha$ has less occurrences than w (right-maximal)
3. for all symbols α, αw has less occurrences than w (left-maximal)

- `taat` is a maximal repeat

```
ctaata
ctaataatg
    ataatg
```

- `aat` is right-maximal but not left-maximal

```
  taata
ctaataatg
    taatg
```

Maximal Repeats

Problem: Maximal repeats

List all maximal repeats

Solution

- List all right-maximal repeats and report those that are left-maximal too
- How to check for left-maximality?

Burrows-Wheeler Transform

ctaataatg	LCP	BWT	SA	
		t	2	aataatg
	3			aat
		t	5	aatg
	1			a
		a	3	ataatg
	2			at
		a	6	atg
	0			
		#	0	ctaataatg
	0			
		t	8	g
	0			
		c	1	taataatg
	4			taat
		a	4	taatg
	1			t
		a	7	tg

Burrows-Wheeler Transform

ctaataatg

LCP	BWT	SA	
	t	2	aataatg
3			aat
	t	5	aatg
1			a
	a	3	ataatg
2			at
	a	6	atg
0			
	#	0	ctaataatg
0			
	t	8	g
0			
	c	1	taataatg
4			taat
	a	4	taatg
1			t
	a	7	tg

Burrows-Wheeler Transform

ctaataatg

LCP	BWT	SA	
	t	2	aataatg
3			aat
	t	5	aatg
1			a
	a	3	ataatg
2			at
	a	6	atg
0			
	#	0	ctaataatg
0			
	t	8	g
0			
	c	1	taataatg
4			taat
	a	4	taatg
1			t
	a	7	tg

Burrows-Wheeler Transform

ctaataatg

LCP	BWT	SA	
	t	2	aataatg
3			aat
	t	5	aatg
1			a
	a	3	ataatg
2			at
	a	6	atg
0			
	#	0	ctaataatg
0			
	t	8	g
0			
	c	1	taataatg
4			taat
	a	4	taatg
1			t
	a	7	tg

Irreducible LCP Values

LCP[i] is irreducible if BWT[$i-1$] $\neq$ BWT[i]

IL	LCP	BWT	SA	
		t	2	aataatg
N	3			aat
		t	5	aatg
Y	1			a
		a	3	ataatg
N	2			at
		a	6	atg
Y	0			
		#	0	ctaataatg
Y	0			
		t	8	g
Y	0			
		c	1	taataatg
Y	4			taat
		a	4	taatg
N	1			t
		a	7	tg

Irreducible LCP Values

LCP[i] is irreducible if BWT[$i-1$] $\neq$ BWT[i]

IL	LCP	BWT	SA	
		t	2	aataatg
N	3			aat
		t	5	aatg
Y	1			a
		a	3	ataatg
N	2			at
		a	6	atg
Y	0			
		#	0	ctaataatg
Y	0			
		t	8	g
Y	0			
		c	1	taataatg
Y	4			taat
		a	4	taatg
N	1			t
		a	7	tg

Irreducible LCP Values

LCP[i] is irreducible if BWT[$i-1$] $\neq$ BWT[i]

IL	LCP	BWT	SA	
		t	2	aataatg
N	3			aat
		t	5	aatg
Y	1			a
		a	3	ataatg
N	2			at
		a	6	atg
Y	0			
		#	0	ctaataatg
Y	0			
		t	8	g
Y	0			
		c	1	taataatg
Y	4			taat
		a	4	taatg
N	1			t
		a	7	tg

Irreducible LCP Values

LCP[i] is irreducible if BWT[$i-1$] $\neq$ BWT[i]

IL	LCP	BWT	SA	
		t	2	aataatg
N	3			aat
		t	5	aatg
Y	1			a
		a	3	ataatg
N	2			at
		a	6	atg
Y	0			
		#	0	ctaataatg
Y	0			
		t	8	g
Y	0			
		c	1	taataatg
Y	4			taat
		a	4	taatg
N	1			t
		a	7	tg

Irreducible LCP Values

LCP[i] is irreducible if
BWT[$i-1$] $\neq$ BWT[i]

IL	LCP	BWT	SA	
		t	2	aataatg
N	3			aat
		t	5	aatg
Y	1			a
		a	3	ataatg
N	2			at
		a	6	atg
Y	0			
		#	0	ctaataatg
Y	0			
		t	8	g
Y	0			
		c	1	taataatg
Y	4			taat
		a	4	taatg
N	1			t
		a	7	tg

Irreducible LCP Values

LCP[i] is irreducible if
BWT[$i-1$] $\neq$ BWT[i]

IL	LCP	BWT	SA	
		t	2	aataatg
N	3			aat
		t	5	aatg
Y	1			a
		a	3	ataatg
N	2			at
		a	6	atg
Y	0			
		#	0	ctaataatg
Y	0			
		t	8	g
Y	0			
		c	1	taataatg
Y	4			taat
		a	4	taatg
N	1			t
		a	7	tg

Maximal Repeats

Problem: Maximal repeats

List all maximal repeats

Solution

- List all ℓ-intervals (representing right-maximal repeats) and report those that contain an irreducible LCP value
- Since the intervals are generated in order of ending position, we only need to keep track of the most recent irreducible position

Maximal Repeats

Computing irreducible positions

- Precompute IL or BWT, or
- Compute the on the fly: $\mathsf{BWT}[i] = T[\mathsf{SA}[i] - 1]$

Maximal Repeats

Computing irreducible positions

- Precompute IL or BWT, or
- Compute the on the fly: $\mathrm{BWT}[i] = T[\mathrm{SA}[i] - 1]$

Algorithm properties

- Single sequential pass over the arrays
  - Except the text in the on-the-fly computation
- Superconstant additional space needed only for the stack
- Only a few lines of code

Permuted LCP Array

$$\mathrm{PLCP}[\mathrm{SA}[i]] = \mathrm{LCP}[i]$$

T	c	t	a	a	t	a	a	t	g
PLCP	0	0	-	1	4	3	2	1	0

LCP	SA	
	2	aataatg
3		aat
	5	aatg
1		a
	3	ataatg
2		at
	6	atg
0		
	0	ctaataatg
0		
	8	g
0		
	1	taataatg
4		taat
	4	taatg
1		t
	7	tg

Properties of PLCP Array

T	c	t	a	a	t	a	a	t	g
PLCP	0	0	-	1	4	3	2	1	0

Lemma

$\text{PLCP}[i] \geq \text{PLCP}[i-1] - 1$

- Used in all efficient (P)LCP array construction algorithms
- Succinct representation in $2n$ bits

Properties of PLCP Array

T	c	t	a	a	t	a	a	t	g
PLCP	0	0	-	1	4	3	2	1	0

Lemma

$\text{PLCP}[i] \geq \text{PLCP}[i-1] - 1$

- Used in all efficient (P)LCP array construction algorithms
- Succinct representation in $2n$ bits

Lemma

$\text{PLCP}[i] = \text{PLCP}[i-1] - 1$ if $\text{PLCP}[i]$ is reducible

- Reducible values easy to compute given irreducible values

Sum of Irreducible LCP Values

Lemma

The sum of irreducible (P)LCP values is at most $n \log n$.

- Useful in some (P)LCP array construction algorithms

Summary

Irreducible LCP values

- Interesting combinatorial properties
- Useful in several algorithms

Outline

Conclusion

LCP array

- Powerful data structure with many applications
- Often simple and efficient algorithms
- Surprisingly complex combinatorics

Conclusion

LCP array

- Powerful data structure with many applications
- Often simple and efficient algorithms
- Surprisingly complex combinatorics

Many interesting topics not covered here

- Powerful tools such as Range Minimum Queries on LCP array
- Interesting applications such as Lempel–Ziv factorization
- Construction algorithms, compact representations, ...
- Even complexity results: LCP array validation

LCP Array Validation

LCP Array Validation

Given an integer array, find if it is a valid LCP array

Linear time algorithm [K, Piatkowski, Puglisi, 2016]

- generalized LCP array for sets of cyclic strings
- binary alphabet

NP-hard [K, Piatkowski, Puglisi, 2016]

- sets of non-cyclic strings or single (cyclic or non-cyclic) string
- any alphabet size

Open problem

- sets of cyclic strings
- larger than binary alphabet

Thank You!