Lempel-Ziv Factorization Revisited

Enno Ohlebusch Simon Gog

Institute of Theoretical Computer Science
Ulm University

Palermo, June 27th 2011

Outline

Definition

Let $S = S[1..n]$ a string of length n over an alphabet Σ.
LZ-factorization of S is a factorization $S = \omega_1 \omega_2 \cdots \omega_m$ such that each ω_k, $1 \leq k \leq m$, is either

(a) a letter $c \in \Sigma$ that does not occur in $\omega_1 \omega_2 \cdots \omega_{k-1}$ or
(b) the longest substring of S that occurs at least twice in $\omega_1 \omega_2 \cdots \omega_k$.

Definition also know as LZ77

Example

$S =$ acaaacatat

```
                    1
 01 2 3 45 67 8 9 0
  a|c|a|aa|ca|t|a|t
suffix 4   aacatat
suffix 3  aaacatat
```

Encoding: $(a, 0), (c, 0), (1, 1), (3, 2), (2, 2), (t, 0), (7, 2)$

PrevOcc   LPS

$S =$ aa....a is encoded by $(a, 0), (1, n-1)$
i.e. $\mathcal{O}(\log n)$ bits

Applications

- LZ-factorization is used in/for
  - `gzip`, `WinZip`, `PKZIP`, `7zip` with sliding window
  - computing all runs (Kolpakov and Kucherov)
  - repeats with fixed gap (Kolpakov and Kucherov)
  - branching repeats (Gusfield and Stoye)
  - finding sequence alignment (Crochemore et al.)
  - local periods (Duval et al.)
- Space consumption is a bottleneck for finding tandem repeats in DNA (Pokrazywa and Polanski, 2010)

Outline

A list of existing solutions

The construction of the LZ-factorization can be done in *linear time and space*. Solutions using S and

fast
- suffix tree (ST) (Kolpakov and Kucherov)
- suffix array (SA) + longest common prefix array (LCP) (Crochemore et al.)
- ...

space-efficient
- SA + range minimum queries (RMQs) on SA (Chen et al.)
- SA + BWT (Okanohara and Sadakane)
- (C)SA + RMQ + inverse (C)SA + BWT of S^{rev} + binare tree (Kreft and Navarro) for LZend using backward search
- ...

Our contributions

- A fast algorithm which uses S, SA and Φ during the construction
- A simple space-efficient algorithm which uses S^{rev}
  - BWT of S^{rev} + backward search
  - Compressed Suffix Array (CSA)
  - constant time next and previous greater value data structure for SA which takes $2n + o(n)$ bits

Workflow of most fast solutions

First calculate Longest Previous String (LPS) and Previous Occurrence (PrevOcc) for each suffix *i* in text order

$S[i]$	*a*	*c*	*a*	*a*	*a*	*c*	*a*	*t*	*a*	*t*
i	1	2	3	4	5	6	7	8	9	10
LPF[*i*] or LPS[*i*]	0	0	1	2	3	2	1	0	2	1
PrevOcc[*i*]	0	0	1	3	1	2	5	0	7	8

Second calculate LZ factorization from LPS

(a,0)

Workflow of most fast solutions

First calculate Longest Previous String (LPS) and Previous Occurrence (PrevOcc) for each suffix *i* in text order

$S[i]$	*a*	*c*	*a*	*a*	*a*	*c*	*a*	*t*	*a*	*t*
i	1	2	3	4	5	6	7	8	9	10
LPF[*i*] or LPS[*i*]	0	0	1	2	3	2	1	0	2	1
PrevOcc[*i*]	0	0	1	3	1	2	5	0	7	8

Second calculate LZ factorization from LPS

(a,0) (c,0)

Workflow of most fast solutions

First calculate Longest Previous String (LPS) and Previous Occurrence (PrevOcc) for each suffix *i* in text order

$S[i]$	*a*	*c*	*a*	*a*	*a*	*c*	*a*	*t*	*a*	*t*
i	1	2	3	4	5	6	7	8	9	10
LPF[*i*] or LPS[*i*]	0	0	1	2	3	2	1	0	2	1
PrevOcc[*i*]	0	0	1	3	1	2	5	0	7	8

Second calculate LZ factorization from LPS

(a,0) (c,0) (1,1)

Workflow of most fast solutions

First calculate Longest Previous String (LPS) and Previous Occurrence (PrevOcc) for each suffix *i* in text order

$S[i]$	*a*	*c*	*a*	*a*	*a*	*c*	*a*	*t*	*a*	*t*
i	1	2	3	4	5	6	7	8	9	10
LPF[*i*] or LPS[*i*]	0	0	1	2	3	2	1	0	2	1
PrevOcc[*i*]	0	0	1	3	1	2	5	0	7	8

Second calculate LZ factorization from LPS

(a,0) (c,0) (1,1) (3,2)

Workflow of most fast solutions

First calculate Longest Previous String (LPS) and Previous Occurrence (PrevOcc) for each suffix *i* in text order

$S[i]$	a	c	a	a	a	c	a	t	a	t
i	1	2	3	4	5	6	7	8	9	10
LPF[i] or LPS[i]	0	0	1	2	3	2	1	0	2	1
PrevOcc[i]	0	0	1	3	1	2	5	0	7	8

Second calculate LZ factorization from LPS

(a,0) (c,0) (1,1) (3,2) (1,2)

Workflow of most fast solutions

First calculate Longest Previous String (LPS) and Previous Occurrence (PrevOcc) for each suffix *i* in text order

$S[i]$	*a*	*c*	*a*	*a*	*a*	*c*	*a*	*t*	*a*	*t*
i	1	2	3	4	5	6	7	8	9	10
LPF[i] or LPS[i]	0	0	1	2	3	2	1	0	2	1
PrevOcc[i]	0	0	1	3	1	2	5	0	7	8

Second calculate LZ factorization from LPS

(a,0) (c,0) (1,1) (3,2) (1,2) (t,0)

Workflow of most fast solutions

First calculate Longest Previous String (LPS) and Previous Occurrence (PrevOcc) for each suffix *i* in text order

$S[i]$	*a*	*c*	*a*	*a*	*a*	*c*	*a*	*t*	*a*	*t*
i	1	2	3	4	5	6	7	8	9	10
LPF[i] or LPS[i]	0	0	1	2	3	2	1	0	2	1
PrevOcc[i]	0	0	1	3	1	2	5	0	7	8

Second calculate LZ factorization from LPS

(a,0) (c,0) (1,1) (3,2) (1,2) (t,0) (7,2)

Calculating LPS[SA[i]]

i	SA[i]	LCP[i]	$S_{SA[i]}$	PSV[i]	NSV[i]	LPS[SA[i]]	PrevOcc[SA[i]]
0	0		ε				
1	3	0	*aaacatat*	0	3	1	1
2	4	2	*aacatat*	1	3	2	3
3	1	1	*acaaacatat*	0	11	0	0
4	5	3	*acatat*	3	7	3	1
5	9	1	*at*	4	6	2	7
6	7	2	*atat*	4	7	1	5
7	2	0	*caaacatat*	3	11	0	0
8	6	2	*catat*	7	11	2	2
9	10	0	*t*	8	10	1	8
10	8	1	*tat*	8	11	0	0
11	0	0	ε				

Calculating LPS[SA[i]]

i	SA[i]	LCP[i]	$S_{SA[i]}$	PSV[i]	NSV[i]	LPS[SA[i]]	PrevOcc[SA[i]]
0	0		ε				
1	3	0	*aaacatat*	0	3	1	1
2	4	2	*aacatat*	1	3	2	3
3	1	1	*acaaacatat*	0	11	0	0
4	5	3	*acatat*	3	7	3	1
5	9	1	*at*	4	6	2	7
6	7	2	*atat*	4	7	1	5
7	2	0	*caaacatat*	3	11	0	0
8	6	2	*catat*	7	11	2	2
9	10	0	*t*	8	10	1	8
10	8	1	*tat*	8	11	0	0
11	0	0	ε				

PSV[7]

NSV[7]

$$\mathrm{LPS}[7] = \max\{|\mathrm{lcp}(S_{SA[3]}, S_{SA[7]})|, |\mathrm{lcp}(S_{SA[7]}, S_{SA[11]})|\}$$
$$= \max\{0, 0\} = 0$$

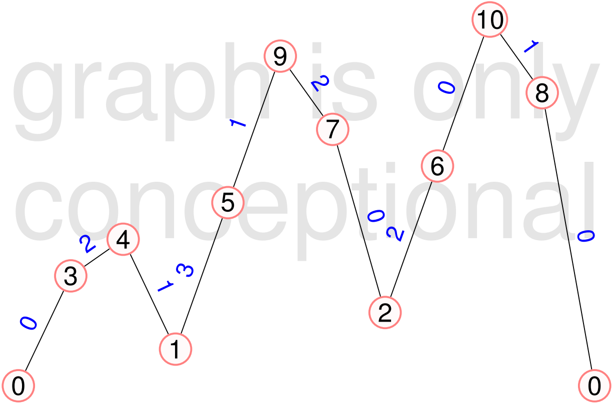
Calculating LPS[SA[i]]

i	SA[i]	LCP[i]	$S_{SA[i]}$	PSV[i]	NSV[i]	LPS[SA[i]]	PrevOcc[SA[i]]
0	0		ε				
1	3	0	*aaacatat*	0	3	1	1
2	4	2	*aacatat*	1	3	2	3
3	1	1	*acaaacatat*	0	11	0	0
4	5	3	*acatat*	3	7	3	1
5	9	1	*at*	4	6	2	7
6	7	2	*atat*	4	7	1	5
7	2	0	*caaacatat*	3	11	0	0
8	6	2	*catat*	7	11	2	2
9	10	0	*t*	8	10	1	8
10	8	1	*tat*	8	11	0	0
11	0	0	ε				

$$\mathrm{LPS}[\mathrm{SA}[i]] = \max\{|\mathrm{lcp}(S_{\mathrm{SA}[\mathrm{PSV}[i]]}, S_{\mathrm{SA}[i]})|, |\mathrm{lcp}(S_{\mathrm{SA}[i]}, S_{\mathrm{SA}[\mathrm{NSV}[i]]})|\}$$

with $|\mathrm{lcp}(S_{\mathrm{SA}[x]}, S_{\mathrm{SA}[y]})| = \mathrm{RMQ}_{\mathrm{LCP}}(x+1, y)$

Peak elimination (Crochemore and Ilie)

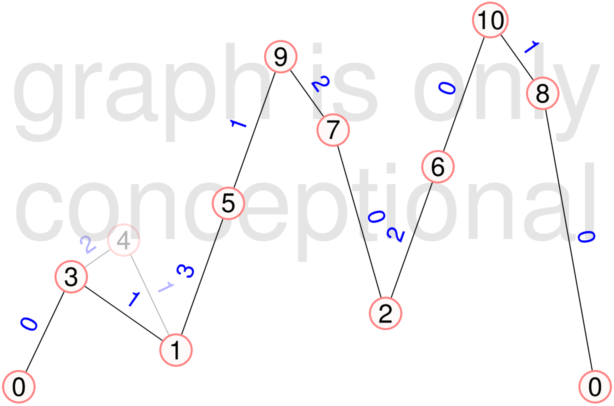


SA[i]

LPS[SA[i]]

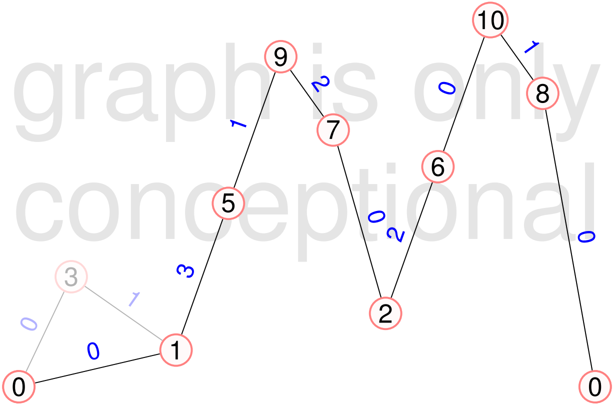
PrevOcc[SA[i]]

Peak elimination (Crochemore and Ilie)



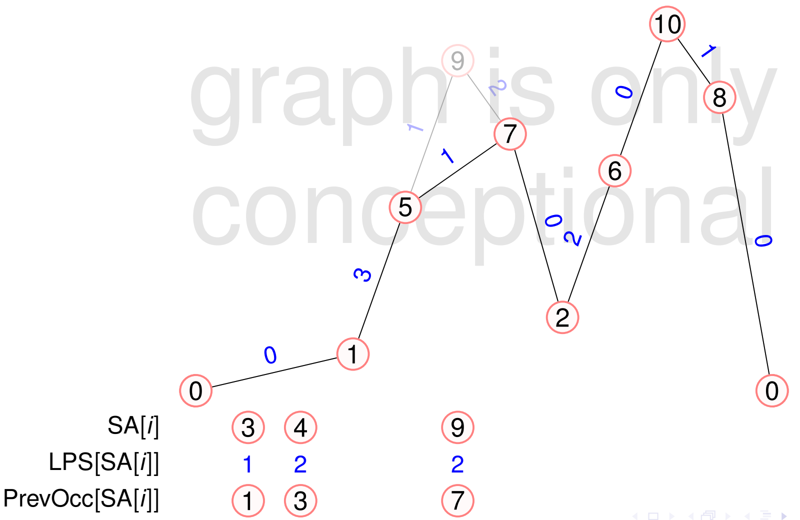
SA[i] 4
LPS[SA[i]] 2
PrevOcc[SA[i]] 3

Peak elimination (Crochemore and Ilie)

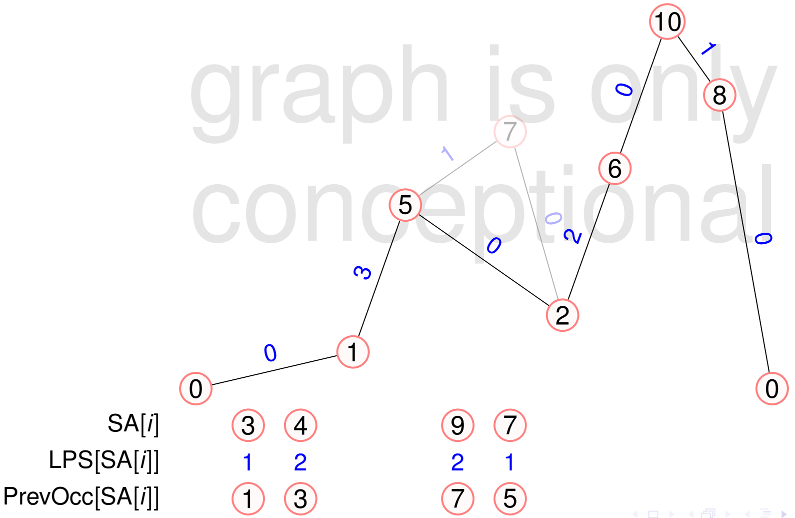


SA[i]	3	4
LPS[SA[i]]	1	2
PrevOcc[SA[i]]	1	3

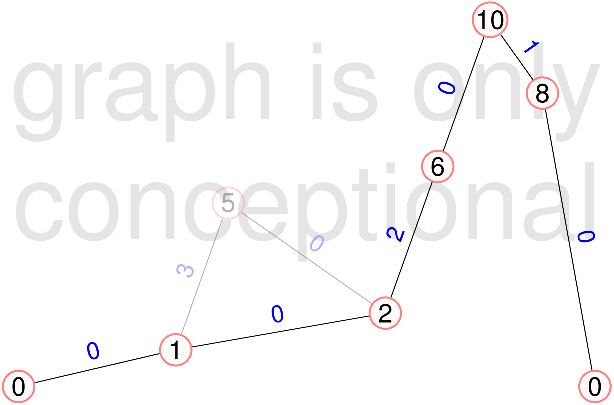
Peak elimination (Crochemore and Ilie)



Peak elimination (Crochemore and Ilie)

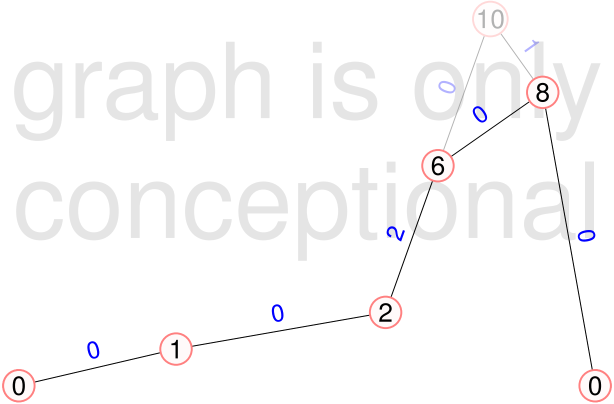


Peak elimination (Crochemore and Ilie)



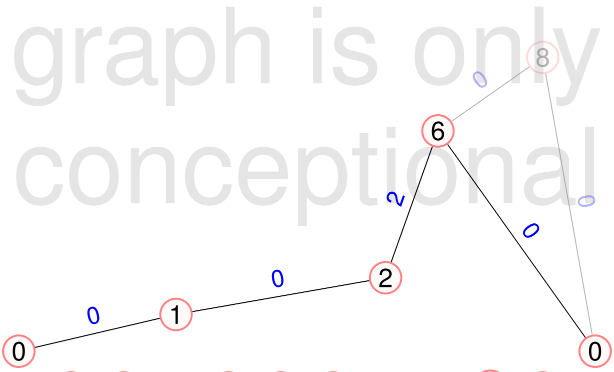
SA[*i*]	3	4		5	9	7
LPS[SA[*i*]]	1	2		3	2	1
PrevOcc[SA[*i*]]	1	3		1	7	5

Peak elimination (Crochemore and Ilie)



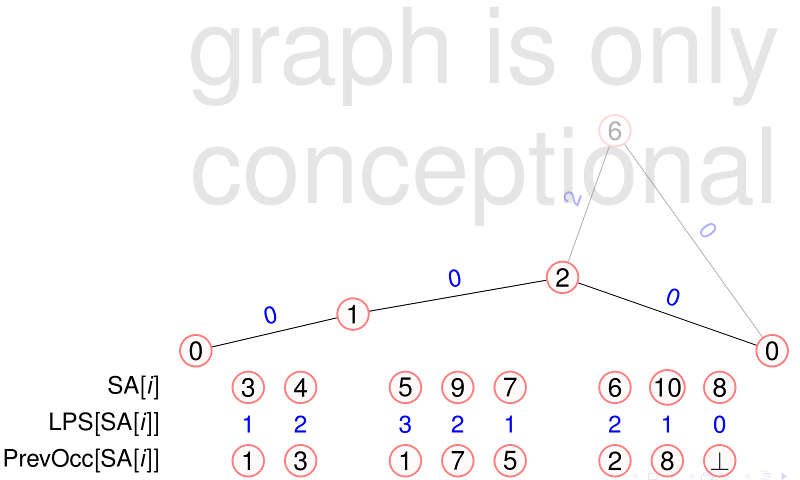
SA[i]	3	4	5	9	7	10
LPS[SA[i]]	1	2	3	2	1	1
PrevOcc[SA[i]]	1	3	1	7	5	8

Peak elimination (Crochemore and Ilie)



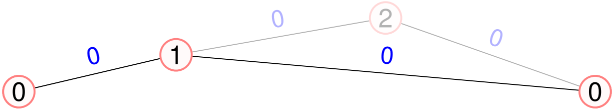
SA[i]	3	4	5	9	7	10	8
LPS[SA[i]]	1	2	3	2	1	1	0
PrevOcc[SA[i]]	1	3	1	7	5	8	⊥

Peak elimination (Crochemore and Ilie)



Peak elimination (Crochemore and Ilie)

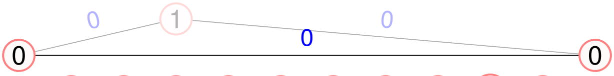
graph is only conceptional



SA[i]	3	4	5	9	7	2	6	10	8
LPS[SA[i]]	1	2	3	2	1	0	2	1	0
PrevOcc[SA[i]]	1	3	1	7	5	⊥	2	8	⊥

Peak elimination (Crochemore and Ilie)

graph is only conceptional



SA[*i*]	3	4	1	5	9	7	2	6	10	8
LPS[SA[*i*]]	1	2	0	3	2	1	0	2	1	0
PrevOcc[SA[*i*]]	1	3	⊥	1	7	5	⊥	2	8	⊥

Summary for peak elimination approach

- LPS is a permutation of LCP
- Most implementations
  - first calculate LCP (SA order!)
  - calculate LPS and PrevOcc also in SA order
  - transform LPS and PrevOcc into text order
  - calculate the LZ-factorization from LPS and PrevOcc in text order

Outline

Overview: The New fast algorithm

Observation:

- Kasai et al.'s LCP algorithm (CPM 2001) produces LCP array in SA order
- Kärkkäinen at al.'s algorithm, called Φ-algorithm, (CPM 2009) produces PLCP (=LCP in text order!) much faster

Our solution:

- Adapt Φ-algorithm to peak elimination
- Eliminate transformation from SA order to text order

Calculating PLCP

Main procedure

for $i \leftarrow 1$ **to** n **do**

  $\Phi[\mathrm{SA}[i]] \leftarrow \mathrm{SA}[i-1]$

$\ell \leftarrow 0$

for $i \leftarrow 1$ **to** n **do**

  $j \leftarrow \Phi[i]$

  while $S[i+\ell] = S[j+\ell]$ **do**

    $\ell \leftarrow \ell + 1$

  $\mathrm{PLCP}[i] \leftarrow \ell$

  $\ell \leftarrow max(\ell - 1, 0)$

The new fast algorithm

Main procedure

```
for i ← 1 to n do
    Φ[SA[i]] ← SA[i − 1]
ℓ ← 0
for i ← 1 to n do
    j ← Φ[i]
    while S[i + ℓ] = S[j + ℓ] do
        ℓ ← ℓ + 1
    if i > j then
        sop(i, ℓ, j)
    else
        sop(j, ℓ, i)
    ℓ ← max(ℓ − 1, 0)
```

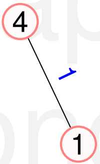
The new fast algorithm

Main procedure
for $i \leftarrow 1$ **to** n **do**
  $\Phi[SA[i]] \leftarrow SA[i-1]$
$\ell \leftarrow 0$
for $i \leftarrow 1$ **to** n **do**
  $j \leftarrow \Phi[i]$
  while $S[i+\ell] = S[j+\ell]$ **do**
    $\ell \leftarrow \ell + 1$
  if $i > j$ **then**
    $sop(i, \ell, j)$
  else
    $sop(j, \ell, i)$
  $\ell \leftarrow max(\ell - 1, 0)$

Procedure $sop(i, \ell, j)$
if $LPS[i] = \perp$ **then**
  $LPS[i] \leftarrow \ell$
  $PrevOcc[i] \leftarrow j$
else
  if $LPS[i] < \ell$ **then**
    if $PrevOcc[i] > j$ **then**
      $sop(PrevOcc[i], LPS[i], j)$
    else
      $sop(j, LPS[i], PrevOcc[i])$
    $LPS[i] \leftarrow \ell$
    $PrevOcc[i] \leftarrow j$
  else $/\star\ LPS[i] \geq \ell \star/$
    if $PrevOcc[i] > j$ **then**
      $sop(PrevOcc[i], \ell, j)$
    else
      $sop(j, \ell, PrevOcc[i])$

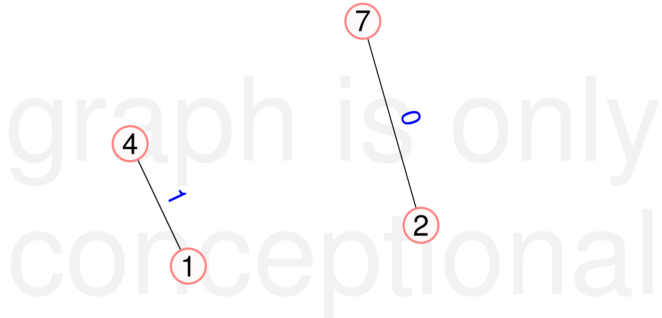
set(1, $\Phi[1]$ = 4)

graph is only conceptional



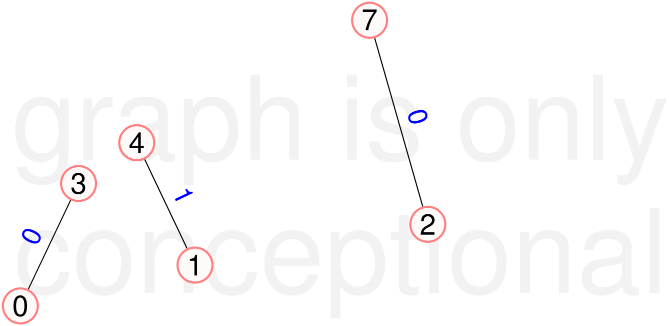
i	0	1	2	3	4	5	6	7	8	9	10	11
LPS[i]					1							
PrevOcc[i]					1							

set(2, $\Phi[2] = 7$)



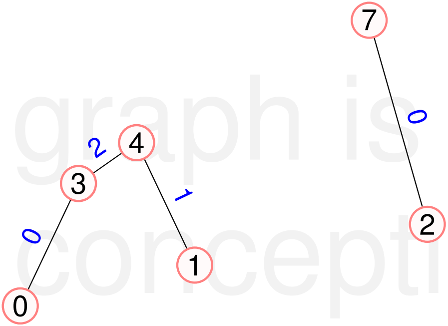
i	0	1	2	3	4	5	6	7	8	9	10	11
LPS[i]					1			0				
PrevOcc[i]					1			2				

set(3, $\Phi[3] = 0$)



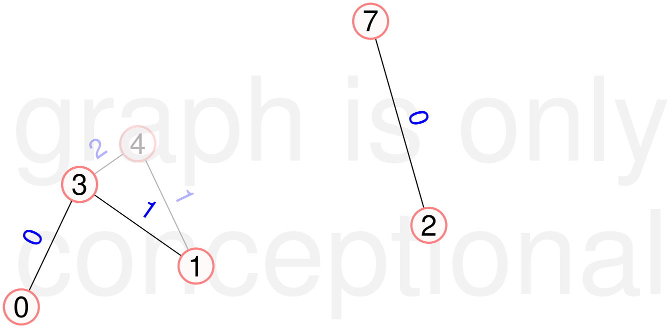
i	0	1	2	3	4	5	6	7	8	9	10	11
LPS[i]				0	1			0				
PrevOcc[i]				0	1			2				

set(4, $\Phi[4] = 3$)



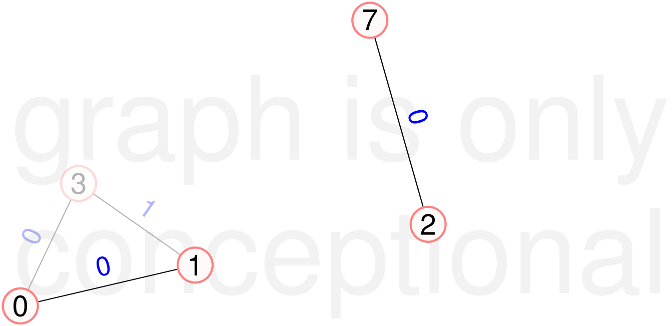
i	0	1	2	3	4	5	6	7	8	9	10	11
LPS[i]				0	2|1			0				
PrevOcc[i]				0	3|1			2				

permute



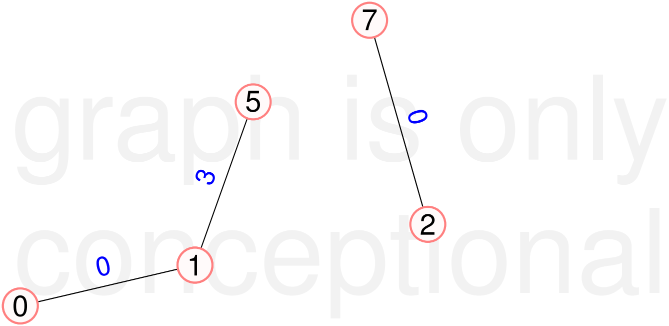
i	0	1	2	3	4	5	6	7	8	9	10	11
LPS[*i*]				0|1	2			0				
PrevOcc[*i*]				0|1	3			2				

permute



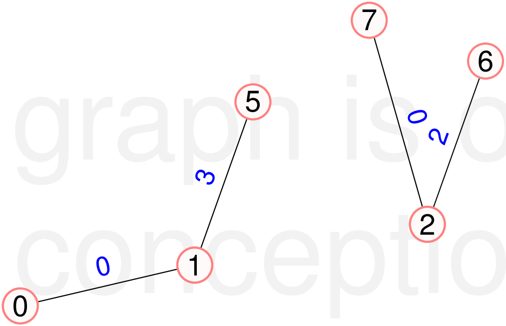
i	0	1	2	3	4	5	6	7	8	9	10	11
LPS[i]		0		1	2			0				
PrevOcc[i]		0		1	3			2				

set(5, $\Phi[5] = 1$)



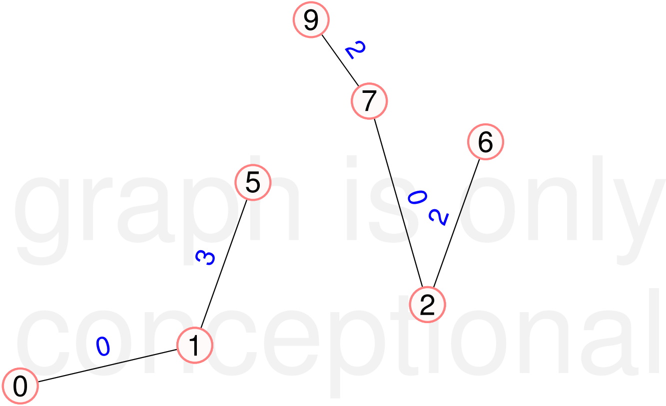
i	0	1	2	3	4	5	6	7	8	9	10	11
LPS[i]		0		1	2	3		0				
PrevOcc[i]		0		1	3	1		2				

set(6, $\Phi[6] = 2$)



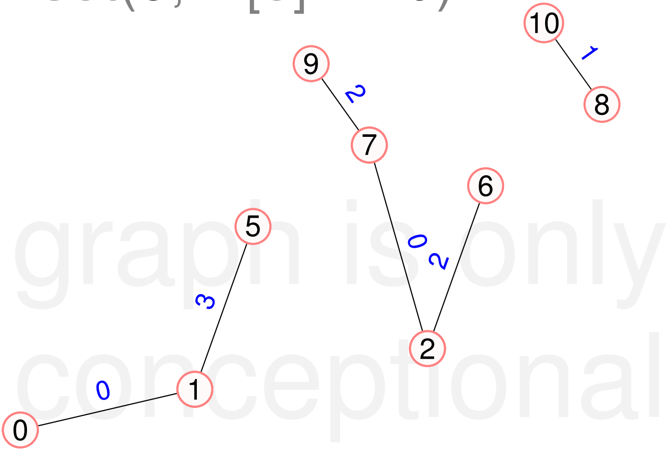
i	0	1	2	3	4	5	6	7	8	9	10	11
LPS[i]		0		1	2	3	2	0				
PrevOcc[i]		0		1	3	1	2	2				

set(7, $\Phi[7] = 9$)



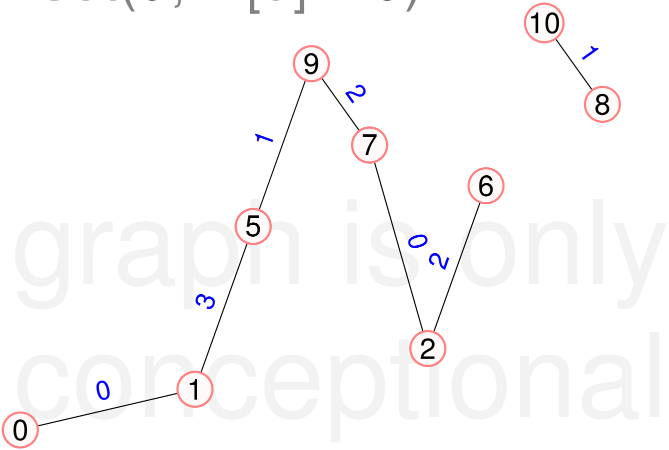
i	0	1	2	3	4	5	6	7	8	9	10	11
LPS[i]		0		1	2	3	2	0		2		
PrevOcc[i]		0		1	3	1	2	2		7		

set(8, $\Phi[8]$ = 10)



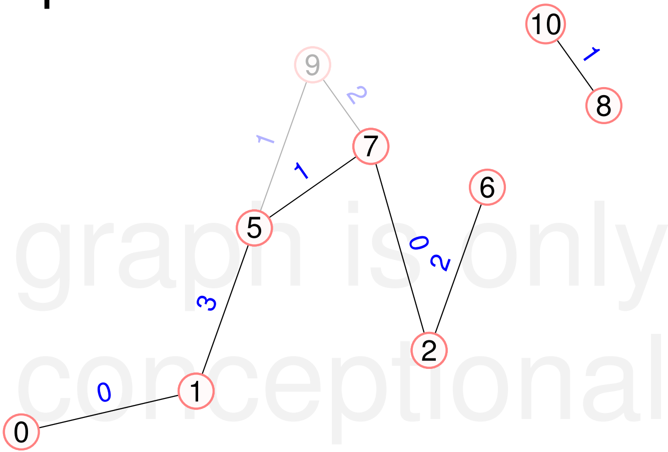
i	0	1	2	3	4	5	6	7	8	9	10	11
LPS[i]		0		1	2	3	2	0		2	1	
PrevOcc[i]		0		1	3	1	2	2		7	8	

set(9, Φ[9] =5)



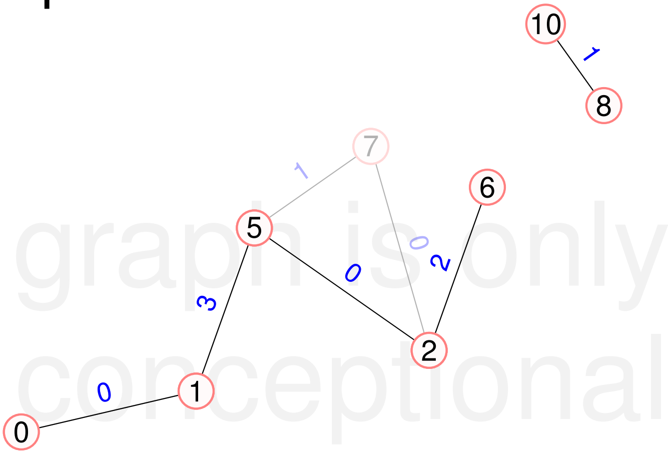
<table>
<tr><td>i</td><td>0</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td><td>10</td><td>11</td></tr>
<tr><td>LPS[i]</td><td></td><td>0</td><td></td><td>1</td><td>2</td><td>3</td><td>2</td><td>0</td><td></td><td>1|2</td><td>1</td><td></td></tr>
<tr><td>PrevOcc[i]</td><td></td><td>0</td><td></td><td>1</td><td>3</td><td>1</td><td>2</td><td>2</td><td></td><td>5|7</td><td>8</td><td></td></tr>
</table>

permute



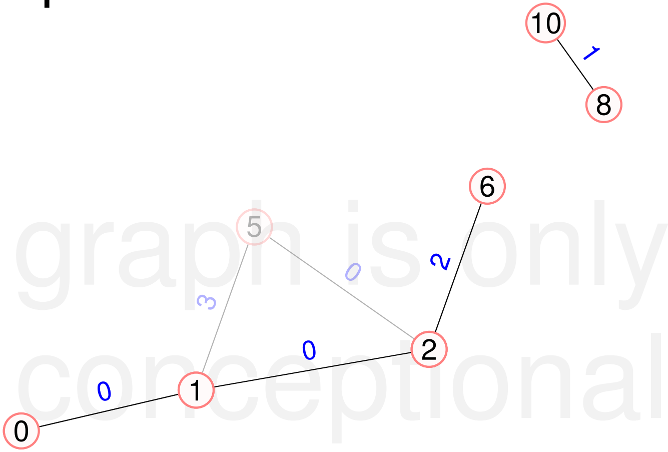
<table>
<tr><td>i</td><td>0</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td><td>10</td><td>11</td></tr>
<tr><td>LPS[i]</td><td></td><td>0</td><td></td><td>1</td><td>2</td><td>3</td><td>2</td><td>0|1</td><td></td><td>2</td><td>1</td><td></td></tr>
<tr><td>PrevOcc[i]</td><td></td><td>0</td><td></td><td>1</td><td>3</td><td>1</td><td>2</td><td>2|5</td><td></td><td>7</td><td>8</td><td></td></tr>
</table>

permute



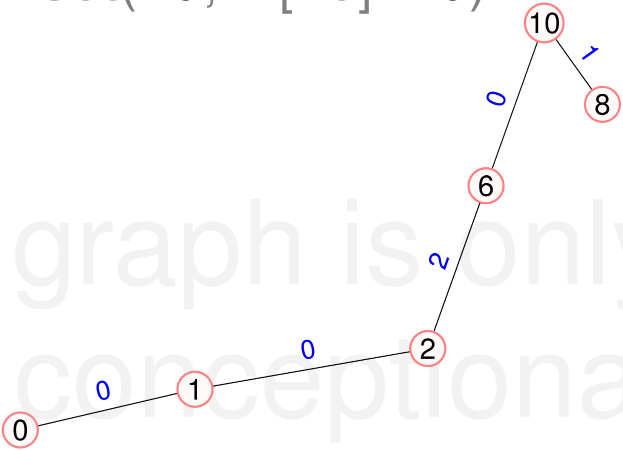
<table>
<tr><td>i</td><td>0</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td><td>10</td><td>11</td></tr>
<tr><td>LPS[i]</td><td></td><td>0</td><td></td><td>1</td><td>2</td><td>0|3</td><td>2</td><td>1</td><td></td><td>2</td><td>1</td><td></td></tr>
<tr><td>PrevOcc[i]</td><td></td><td>0</td><td></td><td>1</td><td>3</td><td>2|1</td><td>2</td><td>5</td><td></td><td>7</td><td>8</td><td></td></tr>
</table>

permute



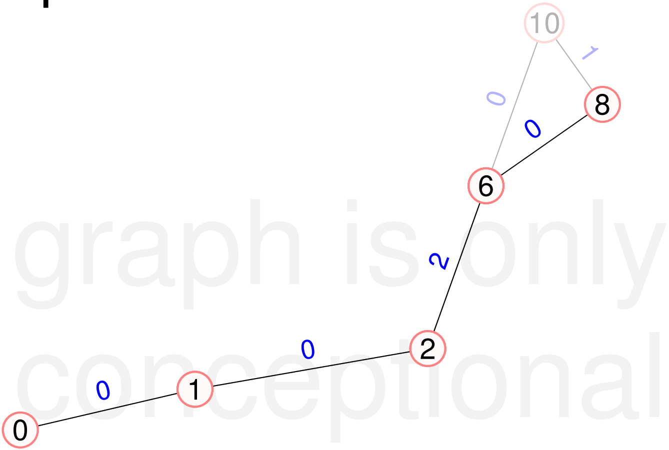
i	0	1	2	3	4	5	6	7	8	9	10	11
LPS[i]		0	0	1	2	3	2	1		2	1	
PrevOcc[i]		0	1	1	3	1	2	5		7	8	

set(10, $\Phi[10] = 6$)



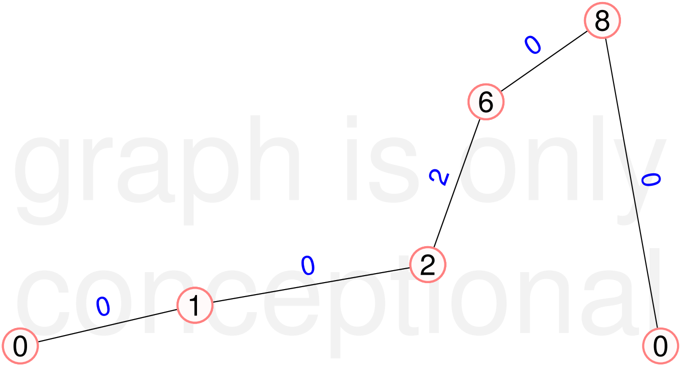
i	0	1	2	3	4	5	6	7	8	9	10	11
LPS[i]		0	0	1	2	3	2	1		2	0|1	
PrevOcc[i]		0	1	1	3	1	2	5		7	6|8	

permute



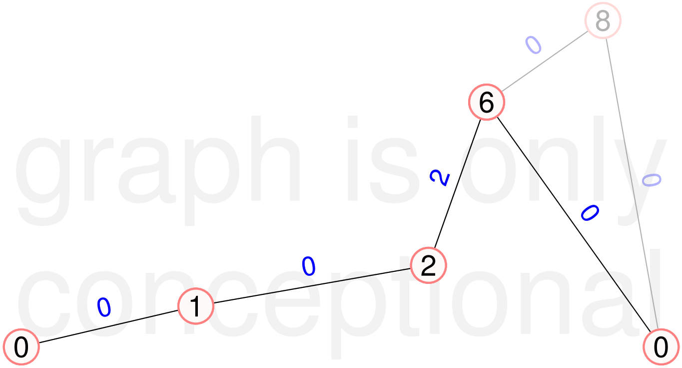
i	0	1	2	3	4	5	6	7	8	9	10	11
LPS[i]		0	0	1	2	3	2	1	0	2	1	
PrevOcc[i]		0	1	1	3	1	2	5	6	7	8	

set(11, $\Phi[11] = 8$)



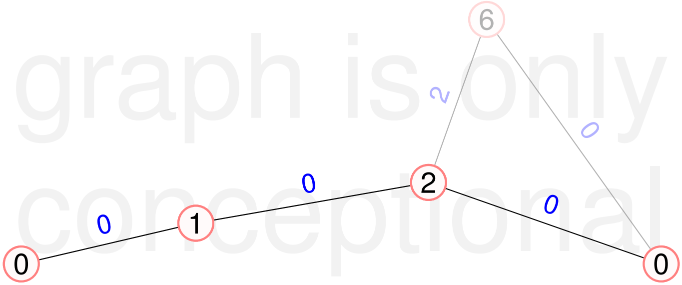
i	0	1	2	3	4	5	6	7	8	9	10	11
LPS[i]		0	0	1	2	3	2	1	0|0	2	1	
PrevOcc[i]		0	1	1	3	1	2	5	6|11	7	8	

permute



i	0	1	2	3	4	5	6	7	8	9	10	11
LPS[*i*]		0	0	1	2	3	0|2	1	0	2	1	
PrevOcc[*i*]		0	1	1	3	1	11|2	5	6	7	8	

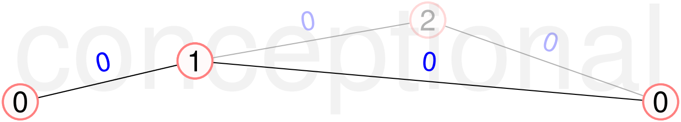
permute



i	0	1	2	3	4	5	6	7	8	9	10	11
LPS[i]		0	0|0	1	2	3	2	1	0	2	1	
PrevOcc[i]		0	1|11	1	3	1	2	5	6	7	8	

permute

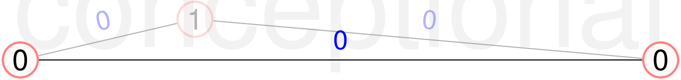
graph is only conceptual



<table>
<tr><td>i</td><td>0</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td><td>10</td><td>11</td></tr>
<tr><td>LPS[i]</td><td></td><td>0|0</td><td>0</td><td>1</td><td>2</td><td>3</td><td>2</td><td>1</td><td>0</td><td>2</td><td>1</td><td></td></tr>
<tr><td>PrevOcc[i]</td><td></td><td>0|11</td><td>1</td><td>1</td><td>3</td><td>1</td><td>2</td><td>5</td><td>6</td><td>7</td><td>8</td><td></td></tr>
</table>

permute

graph is only conceptional



i	0	1	2	3	4	5	6	7	8	9	10	11
LPS[*i*]		0	0	1	2	3	2	1	0	2	1	
PrevOcc[*i*]		0	1	1	3	1	2	5	6	7	8	

Outline

Experimental setup

- SA is already calculated and stored on disk
- Test cases like in Chen et. al
- Other implementations from
  - German Tischler
  - Simon J. Puglisi
- We use data structures of the succinct data structure library (*sdsl*)
  `http://www.uni-ulm.de/in/theo/research/sdsl`

Experimental Results for Fast Algorithms

test case	LCP	LPF_simple	LPF_next_prev	LPF_sorting	LPF_online	LPF_optimal	LZ_OG
chr19.dna4	19.40	46.40	25.80	31.30	26.30	26.30	19.10
chr22.dna4	8.90	24.60	14.80	14.60	12.30	12.20	10.00
E.coli	0.80	2.00	1.20	1.40	1.20	1.20	0.90
bible	0.60	1.60	0.90	1.10	0.90	0.90	0.50
howto	8.10	20.60	11.80	13.40	11.60	11.50	6.70
fib_s14930352	2.20	6.90	3.40	3.80	3.30	3.30	0.80
fib_s9227465	1.40	4.20	2.30	2.40	2.10	2.10	0.50
fss10	1.70	5.60	2.80	3.00	2.70	2.70	0.70
fss9	0.40	1.10	0.60	0.70	0.60	0.60	0.20
p16Mb	3.50	8.70	5.00	5.90	4.90	4.90	3.90
p32Mb	8.50	20.60	11.60	13.90	11.60	11.60	9.70
rndA21_8Mb	1.60	4.00	2.40	2.80	2.30	2.30	1.90
rndA2_8Mb	1.50	3.90	2.20	2.60	2.20	2.20	1.50

Thank you!
Questions?

Experimental Results for Space-Efficient Algorithms

	space in bytes per symbol							time in seconds						
test case	CPS2	LZ_bwd1	LZ_bwd2	LZ_bwd4	LZ_bwd8	LZ_bwd16	LZ_bwd32	CPS2	LZ_bwd1	LZ_bwd2	LZ_bwd4	LZ_bwd8	LZ_bwd16	LZ_bwd32
chr19.dna4	6.0	4.7	2.7	1.7	1.2	1.0	0.8	161.9	75.2	76.5	78.9	83.3	92.1	110.5
chr22.dna4	6.0	4.7	2.7	1.7	1.2	1.0	0.8	78.8	39.0	39.5	40.3	43.0	48.3	58.5
E.coli	6.0	4.7	2.7	1.7	1.2	1.0	0.8	7.6	3.9	4.0	4.3	4.5	5.0	6.2
bible	6.0	5.1	3.1	2.1	1.6	1.3	1.2	3.9	4.6	4.8	5.2	5.6	6.5	8.5
howto	6.0	5.1	3.1	2.1	1.6	1.4	1.2	54.2	60.1	61.7	65.2	72.1	84.9	110.1
fib_s14930352	6.0	4.6	2.6	1.6	1.1	0.8	0.7	2.1	14.6	15.0	14.7	14.7	14.6	14.7
fib_s9227465	6.0	4.6	2.6	1.6	1.1	0.8	0.7	1.3	8.9	9.1	8.9	8.9	9.1	8.9
fss10	6.0	4.6	2.6	1.6	1.1	0.8	0.7	1.8	11.5	11.6	11.6	11.6	11.5	11.6
fss9	6.0	4.6	2.6	1.6	1.1	0.9	0.7	0.4	2.0	2.0	2.0	2.0	2.0	2.0
p16Mb	6.0	5.0	3.0	2.0	1.5	1.3	1.2	33.7	24.8	26.3	28.5	33.8	43.7	63.9
p32Mb	6.0	5.0	3.0	2.0	1.5	1.3	1.1	73.7	52.5	54.5	59.7	69.3	88.1	126.7
rndA21_8Mb	6.0	5.1	3.1	2.1	1.6	1.3	1.2	17.6	13.0	13.9	15.6	19.0	26.2	40.6
rndA2_8Mb	6.0	4.6	2.6	1.6	1.1	0.8	0.7	11.7	6.8	6.8	6.9	7.2	7.5	8.4