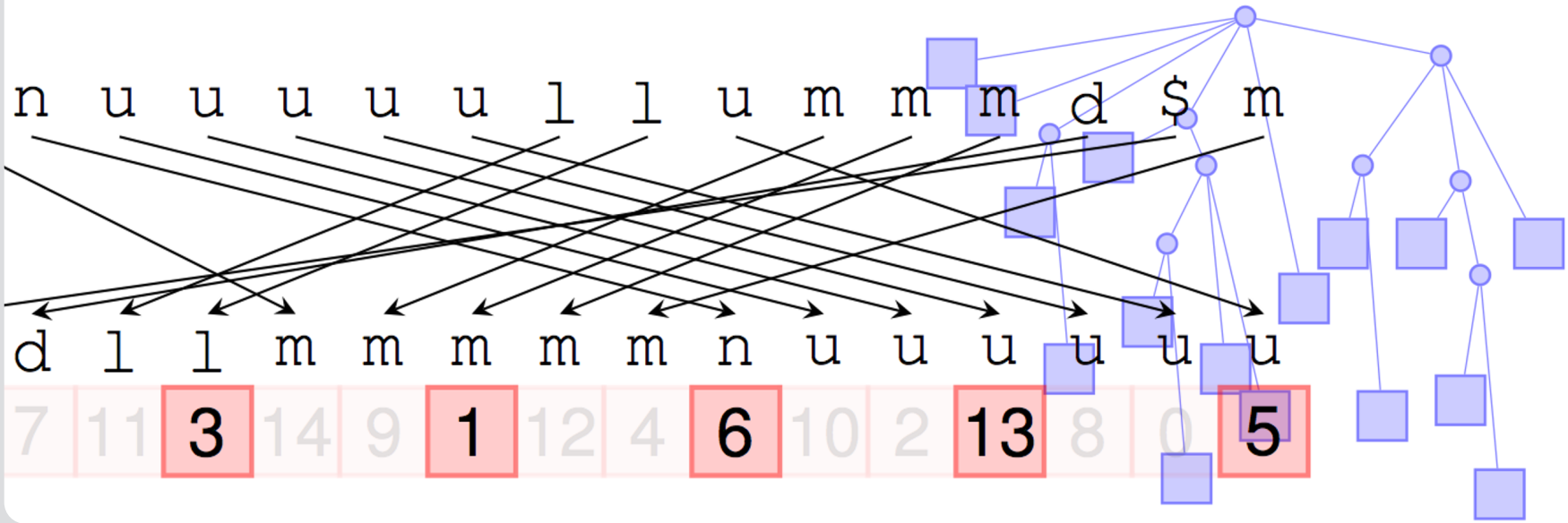
Text Indexing: Lecture 9

Simon Gog – *gog@kit.edu*

Institute of Theoretical Informatics - Algorithmics



LZ Factorization

Let $\mathcal{T}$ be a string of length n over an alphabet Σ.
$\mathcal{T}$ is a factorization $\mathcal{T} = \omega_0\omega_1 \cdots \omega_{m-1}$ such that each ω_k is either

(a) a letter $c \in \Sigma$ that does not occur in $\omega_0\omega_1 \cdots \omega_{k-1}$ or

(b) the longest substring of $\mathcal{T}$ that occurs at least twice in $\omega_0\omega_1 \cdots \omega_k$.

The LZ factorization can be represented by a sequence of pairs $(PrevOcc_0, LPS_0), \ldots, (PrevOcc_{m-1}, LPS_{m-1})$, where in case

(a) $PrevOcc_j = c$ and $LPS_j = 0$

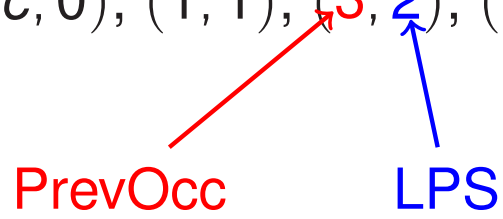
(b) $PrevOcc_j$ is a position in $\omega_0\omega_1 \cdots \omega_{k-1}$ at which an occurrence of ω_k starts and $LPS_j = |\omega_k|$

Example

$\mathcal{T} = \texttt{acaaacatat}$

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

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



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

Reconstruction of $\mathcal{T}$

Karlsruhe Institute of Technology

Exercise

Given the LZ factorization $(PrevOcc_0, LPS_0), \ldots, (PrevOcc_{m-1}, LPS_{m-1})$ of a text $\mathcal{T}$. Devise a linear time algorithm to reconstruct $\mathcal{T}$ from its LZ factorization.

Efficient Calculation of LZ Factorization

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
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)

Efficient Calculation of LZ Factorization

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
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)

Efficient Calculation of LZ Factorization

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
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)

Efficient Calculation of LZ Factorization

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
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)

Efficient Calculation of LZ Factorization

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
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)

Efficient Calculation of LZ Factorization

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
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)

Efficient Calculation of LZ Factorization

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
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)

Next: How can we calculate LPS and PrevOcc?

Calculating LPS[SA[i]]

i	SA[i]	LCP[i]	$S_{\mathrm{SA}[i]}$	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]

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

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				

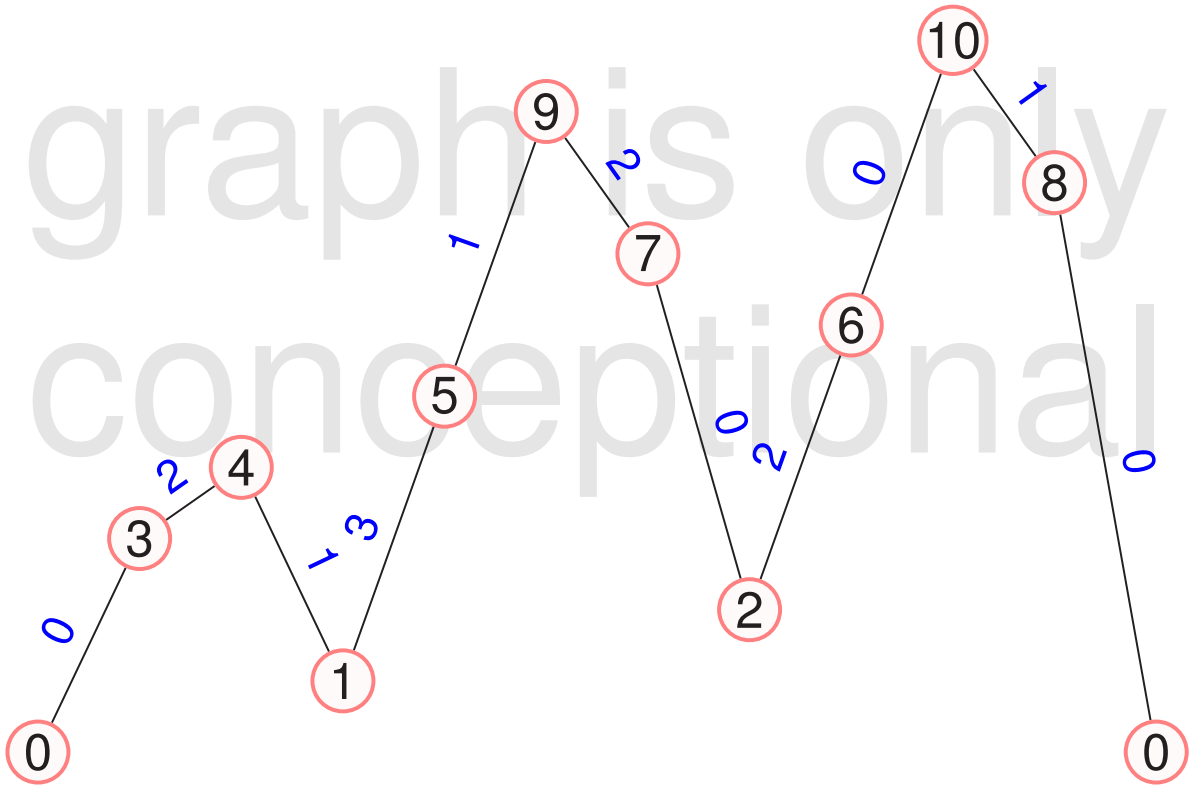
$$\mathrm{LPS}[\mathrm{SA}[i]] = \max\{|lcp(S_{\mathrm{SA}[\mathrm{PSV}[i]]}, S_{\mathrm{SA}[i]})|, |lcp(S_{\mathrm{SA}[i]}, S_{\mathrm{SA}[\mathrm{NSV}[i]]})|\}$$

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

We can construct LPS using PSV, NSV, and RMQ. But we do not need these arrays...

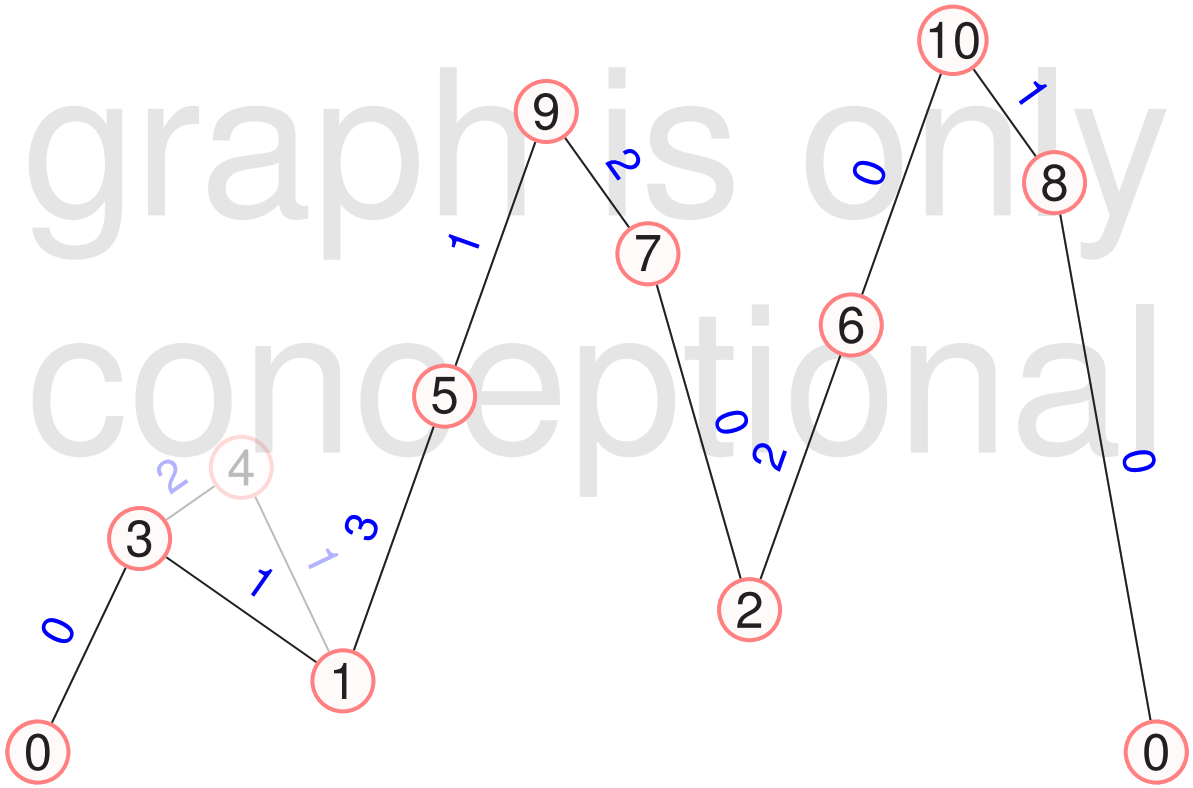
Peak Elimination

(Crochemore and Ilie)



Peak Elimination

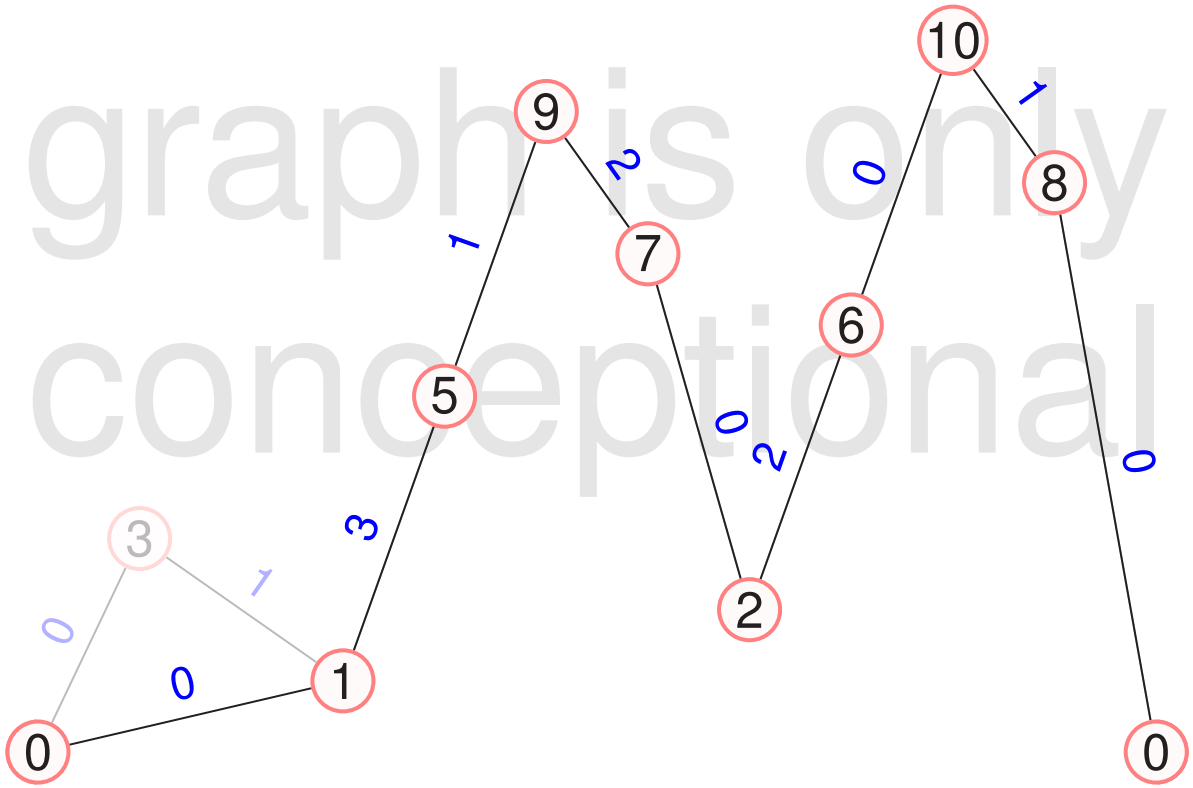
(Crochemore and Ilie)



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

Peak Elimination

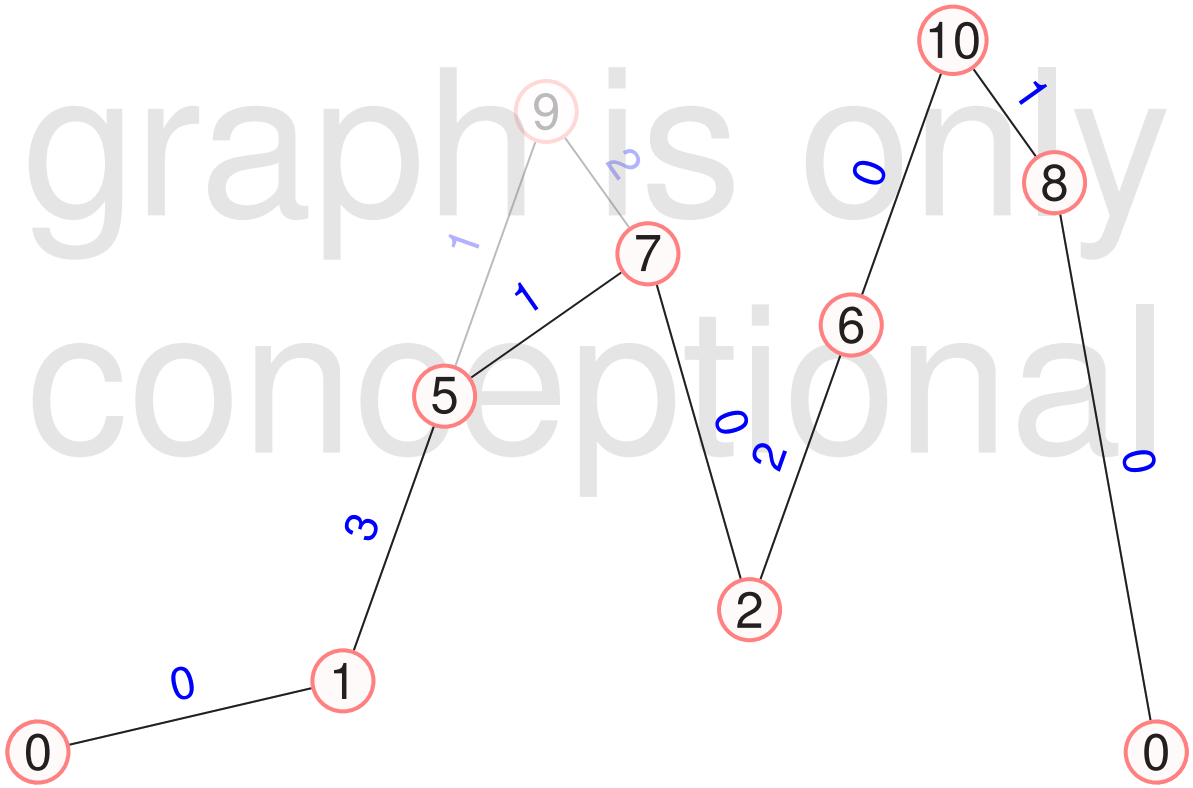
(Crochemore and Ilie)



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

Peak Elimination

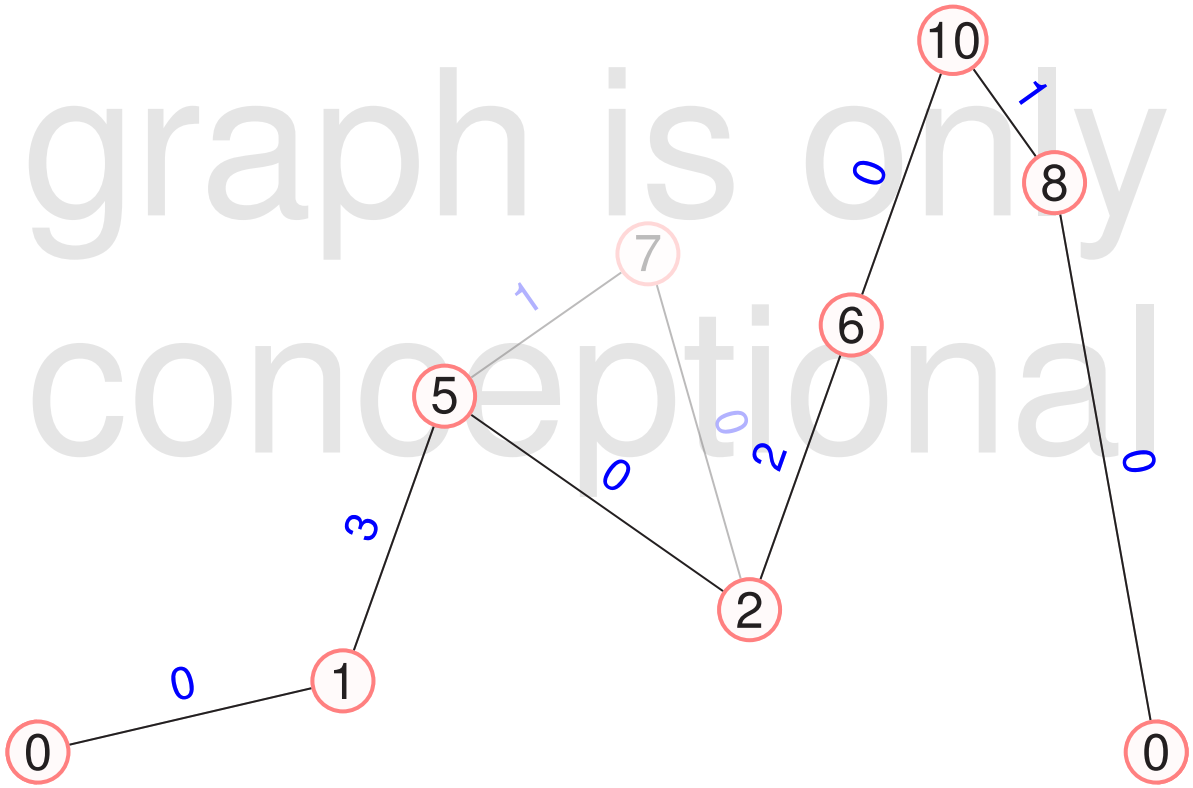
(Crochemore and Ilie)



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

Peak Elimination

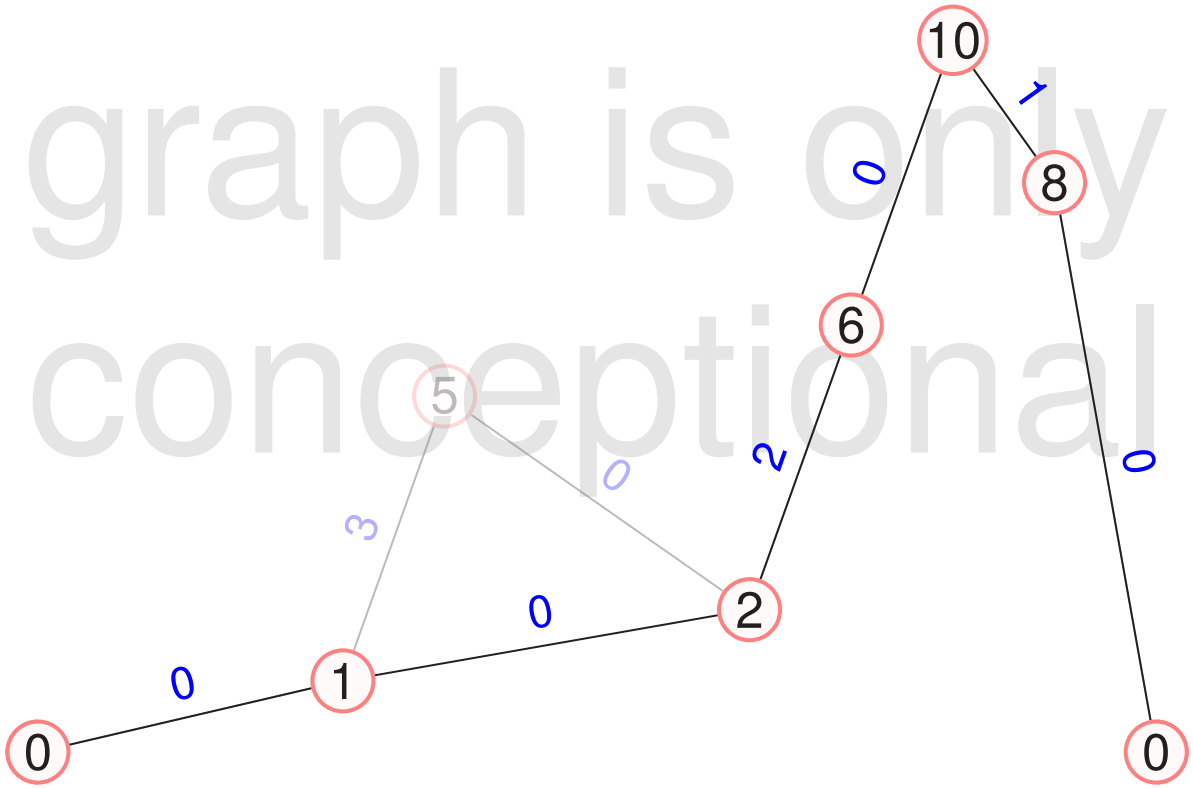
(Crochemore and Ilie)



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

Peak Elimination

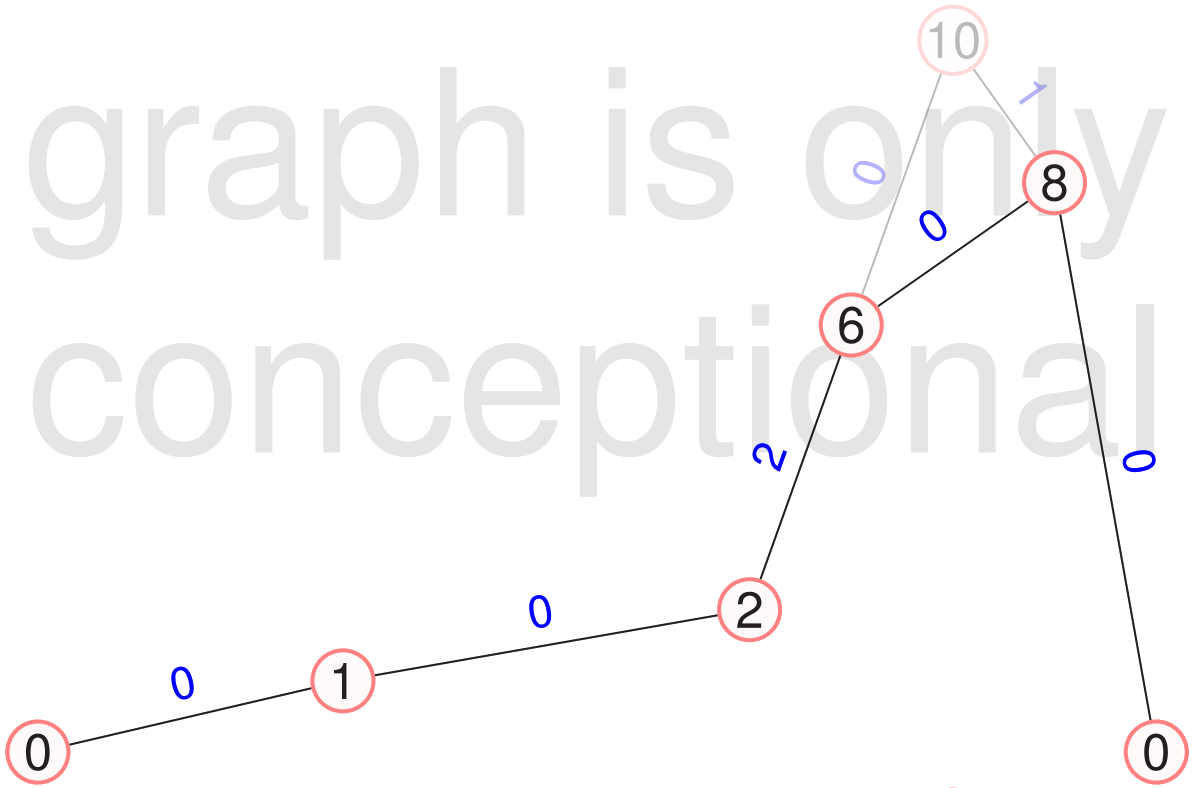
(Crochemore and Ilie)



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

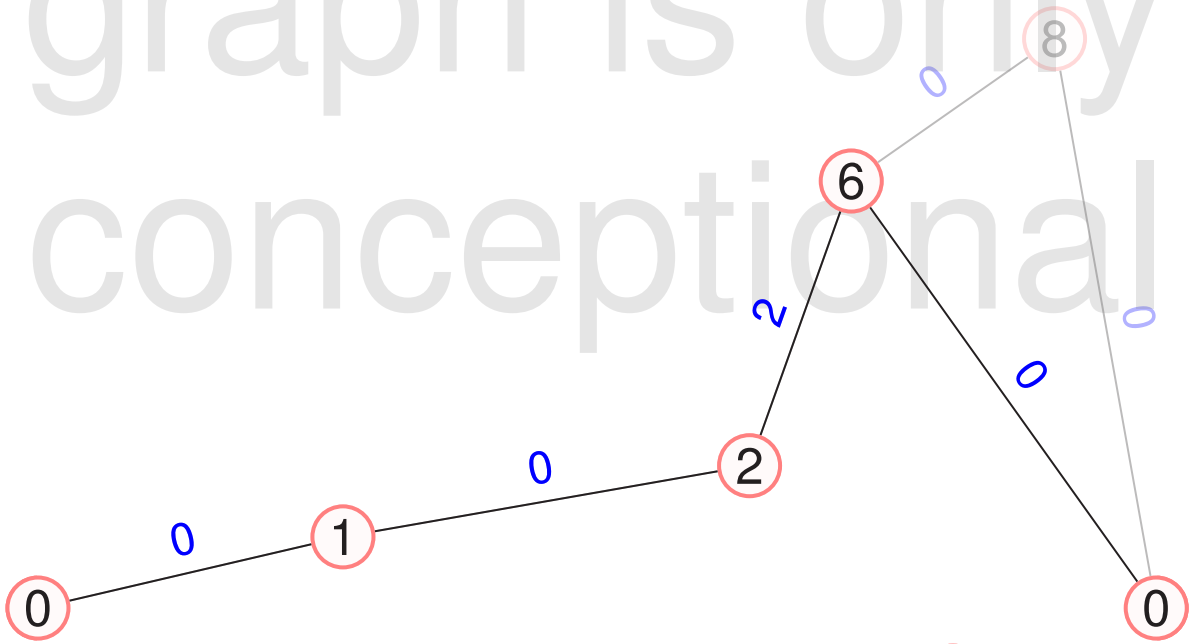
(Crochemore and Ilie)



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

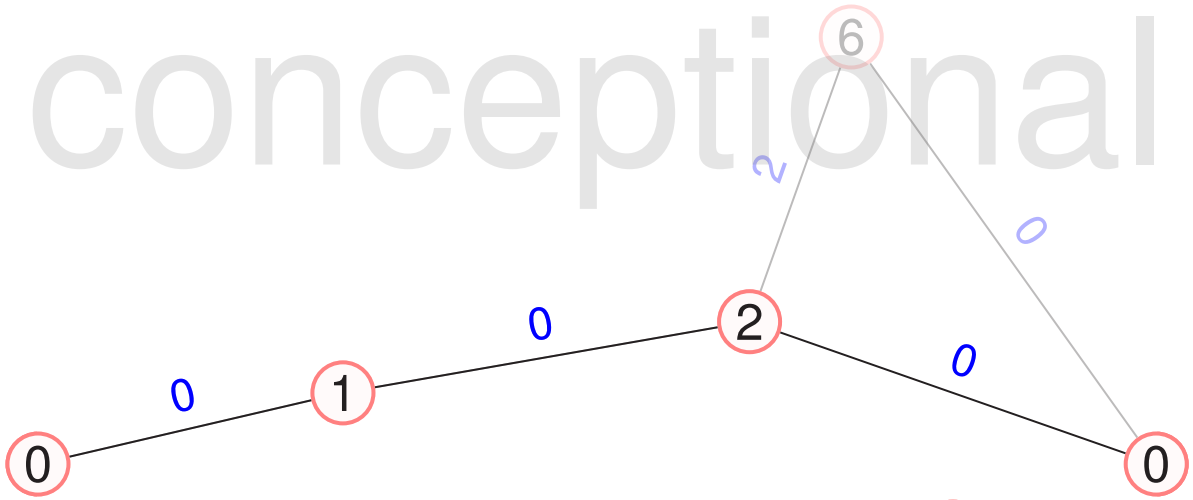
(Crochemore and Ilie)



Peak Elimination

(Crochemore and Ilie)

graph is only conceptional

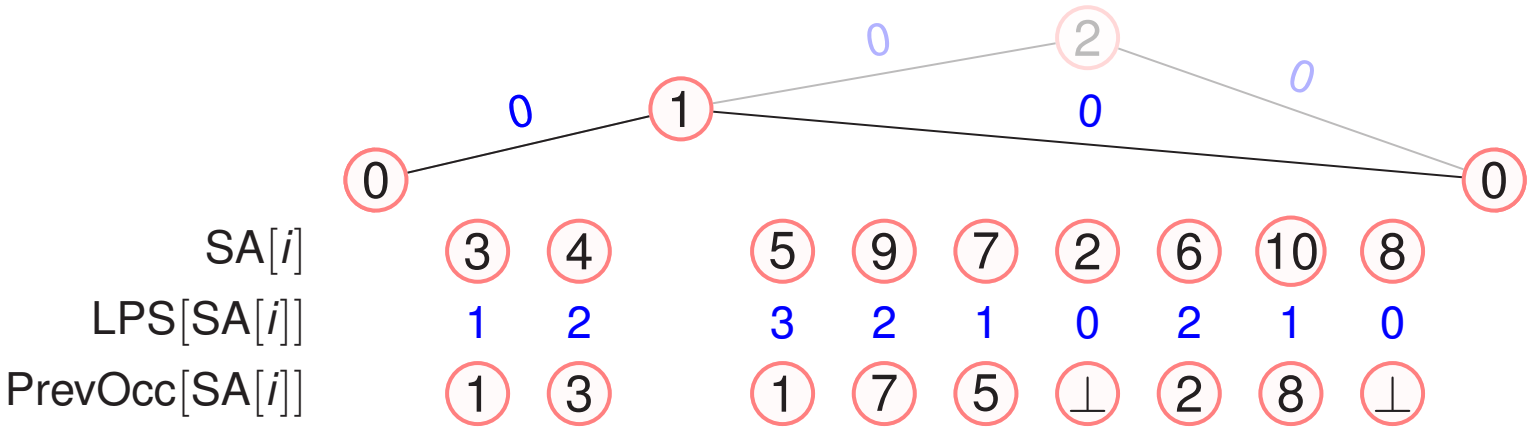


SA[i]	3	4	5	9	7	6	10	8
LPS[SA[i]]	1	2	3	2	1	2	1	0
PrevOcc[SA[i]]	1	3	1	7	5	2	8	$\perp$

Peak Elimination

(Crochemore and Ilie)

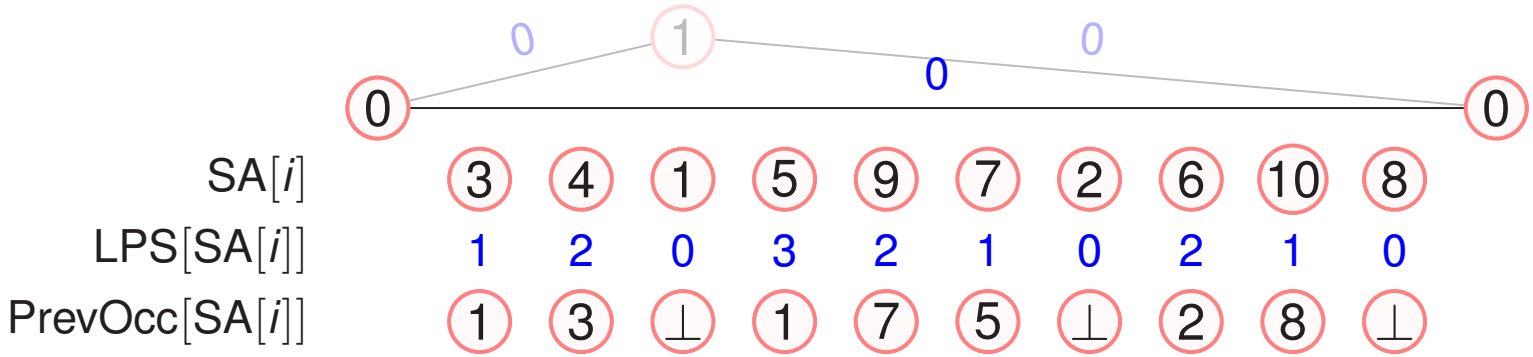
graph is only conceptional



Peak Elimination

(Crochemore and Ilie)

graph is only conceptional



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 of a fast algorithm

Observation:

- Kasai et al.'s LCP algorithm (CPM 2001) produces LCP array in SA order
- Kärkkäinen at al.'s algorithm (Φ-algorithm, see Lecture 7) produces PLCP (=LCP in text order!) much faster

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

A Faster Alternative

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**
      $\ell \leftarrow \ell + 1$
    $\mathrm{PLCP}[i] \leftarrow \ell$
    $\ell \leftarrow max(\ell - 1, 0)$

A Faster Alternative

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**
      $\ell \leftarrow \ell + 1$
    $sop(i, \ell, j)$
    $\ell \leftarrow max(\ell - 1, 0)$

A Faster Alternative

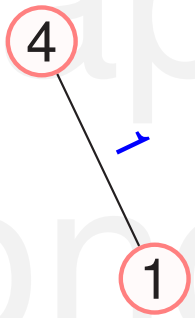
Main procedure

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**
    $\ell \leftarrow \ell + 1$
  sop(i, ℓ, j)
  $\ell \leftarrow \mathit{max}(\ell - 1, 0)$

Procedure *sop*(i, ℓ, j) /* swap or permute */

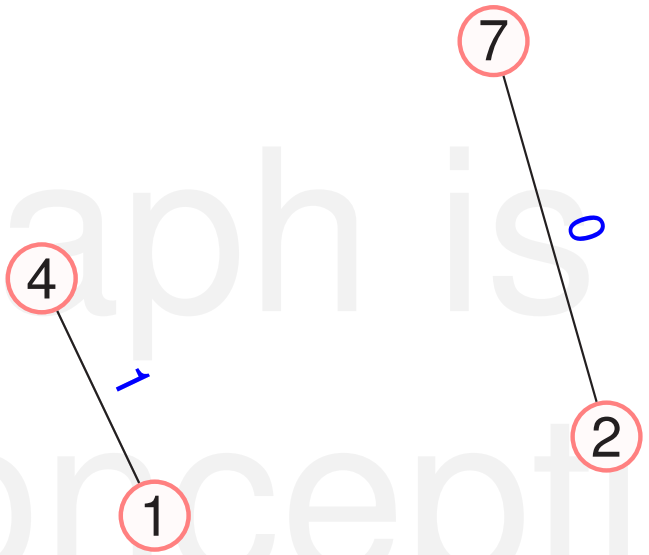
if $j > i$ **then**
  swap(i, j)
if $\mathrm{LPS}[i] = \perp$ **then**
  $\mathrm{LPS}[i] \leftarrow \ell$
  $\mathrm{PrevOcc}[i] \leftarrow j$
else
  if $\mathrm{LPS}[i] < \ell$ **then**
    sop$(\mathrm{PrevOcc}[i], \mathrm{LPS}[i], j)$
    $\mathrm{LPS}[i] \leftarrow \ell$
    $\mathrm{PrevOcc}[i]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

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



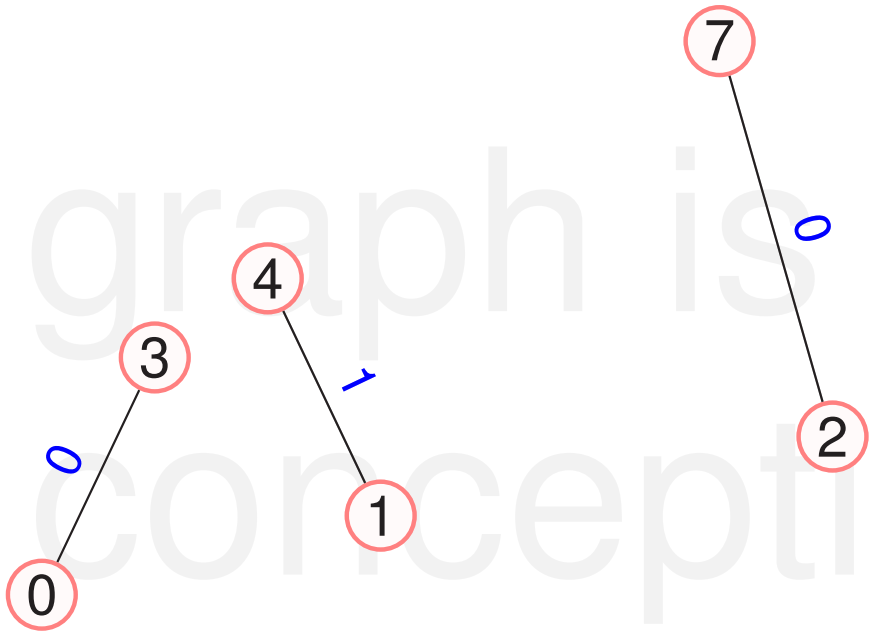
i	0	1	2	3	4	5	6	7	8	9	10	11
LPS$[i]$					1							
PrevOcc$[i]$					1							

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



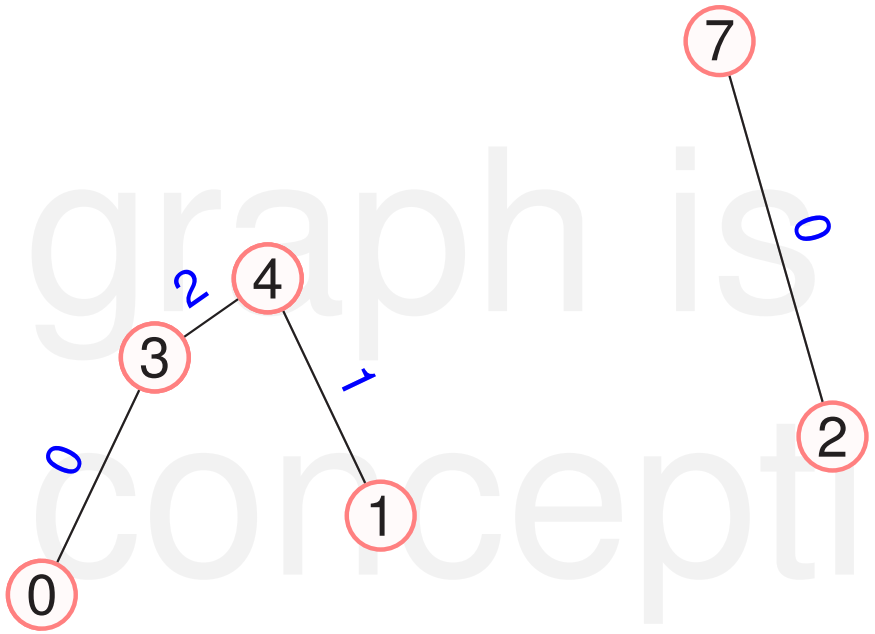
i	0	1	2	3	4	5	6	7	8	9	10	11
LPS$[i]$					1			0				
PrevOcc$[i]$					1			2				

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



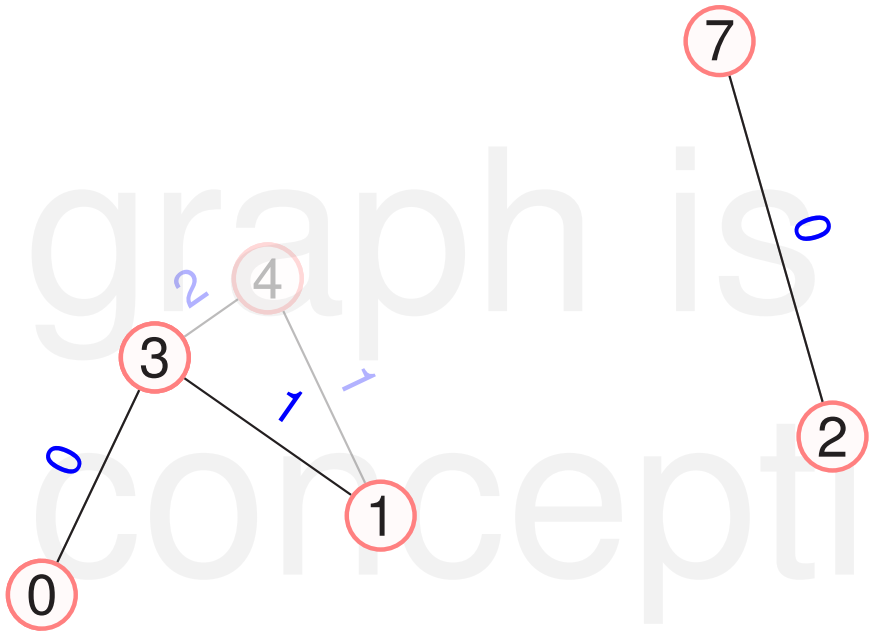
i	0	1	2	3	4	5	6	7	8	9	10	11
LPS[i]				0	1			0				
PrevOcc[i]				0	1			2				

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



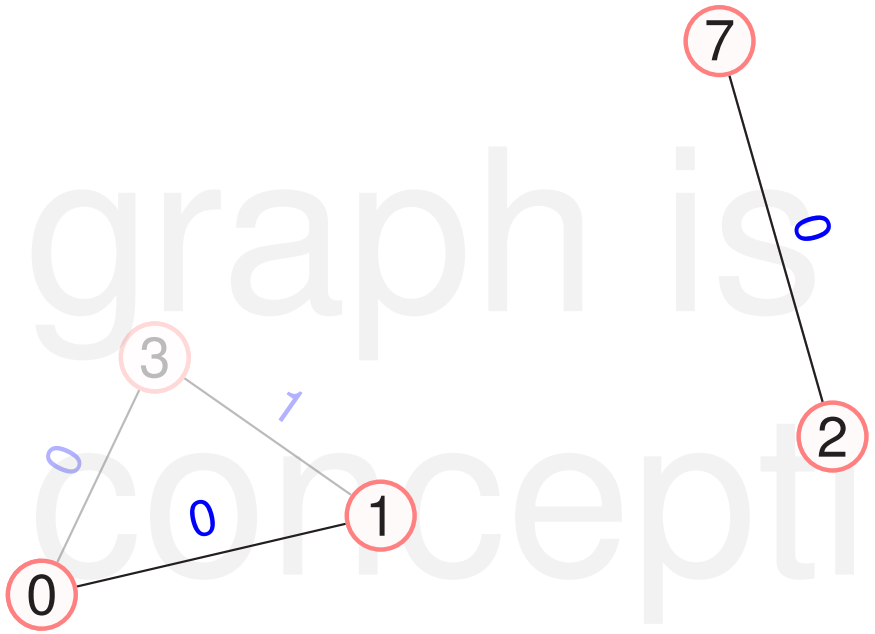
i	0	1	2	3	4	5	6	7	8	9	10	11
LPS$[i]$				0	2|1			0				
PrevOcc$[i]$				0	3|1			2				

permute



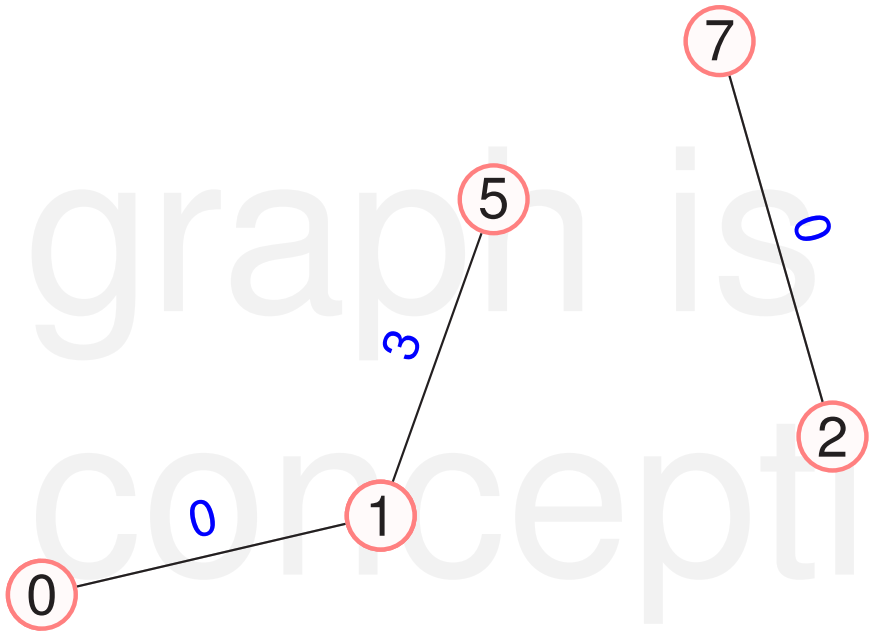
i	0	1	2	3	4	5	6	7	8	9	10	11
LPS$[i]$				0|1	2			0				
PrevOcc$[i]$				0|1	3			2				

permute



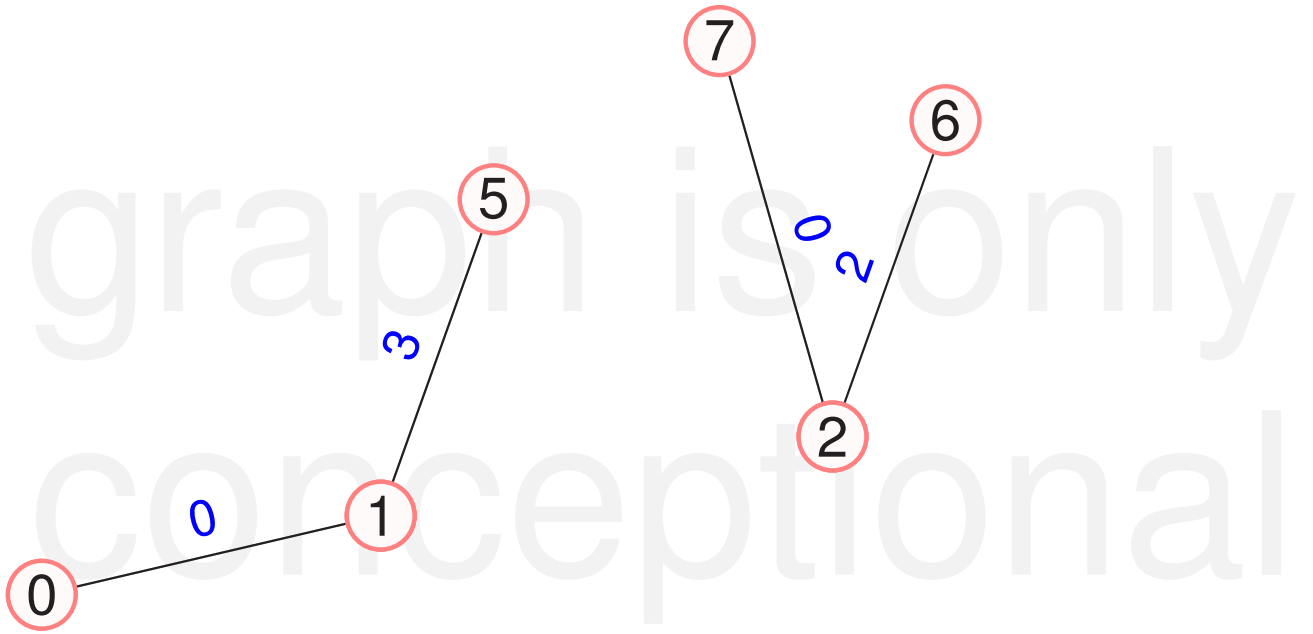
i	0	1	2	3	4	5	6	7	8	9	10	11
LPS$[i]$		0		1	2			0				
PrevOcc$[i]$		0		1	3			2				

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



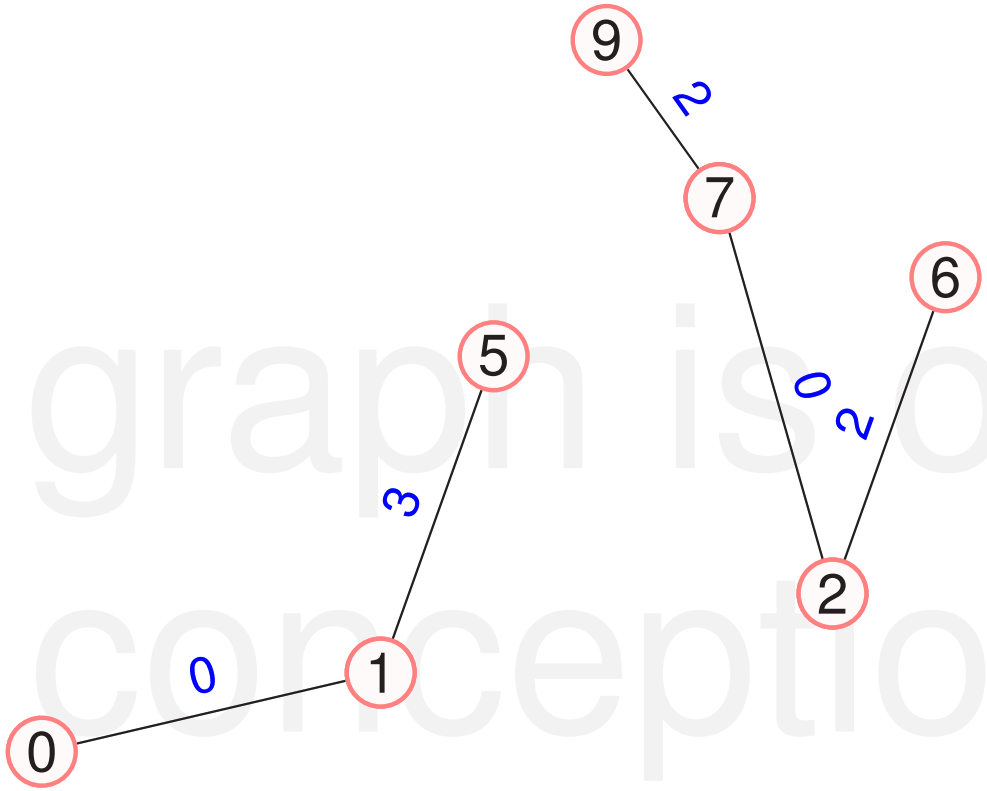
i	0	1	2	3	4	5	6	7	8	9	10	11
LPS$[i]$		0		1	2	3		0				
PrevOcc$[i]$		0		1	3	1		2				

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



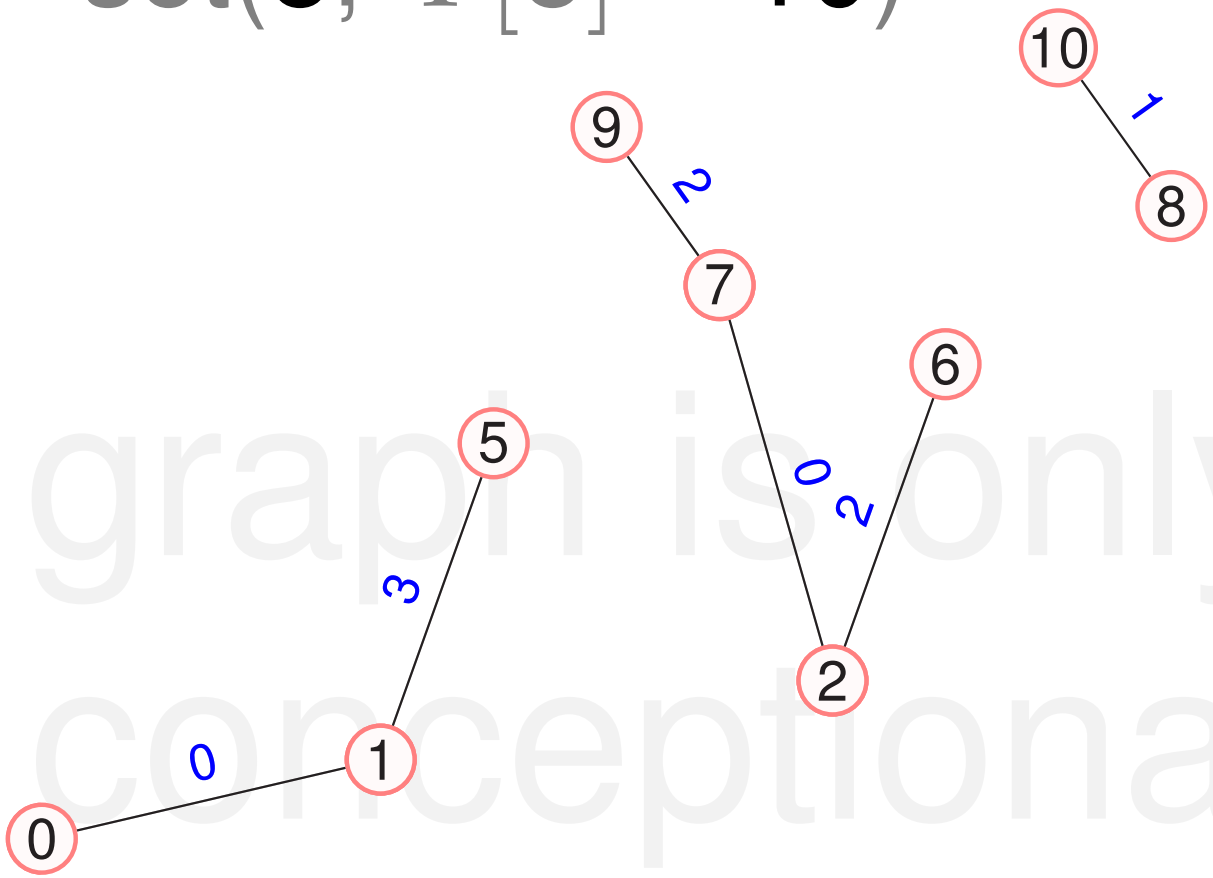
i	0	1	2	3	4	5	6	7	8	9	10	11
LPS$[i]$		0		1	2	3	2	0				
PrevOcc$[i]$		0		1	3	1	2	2				

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



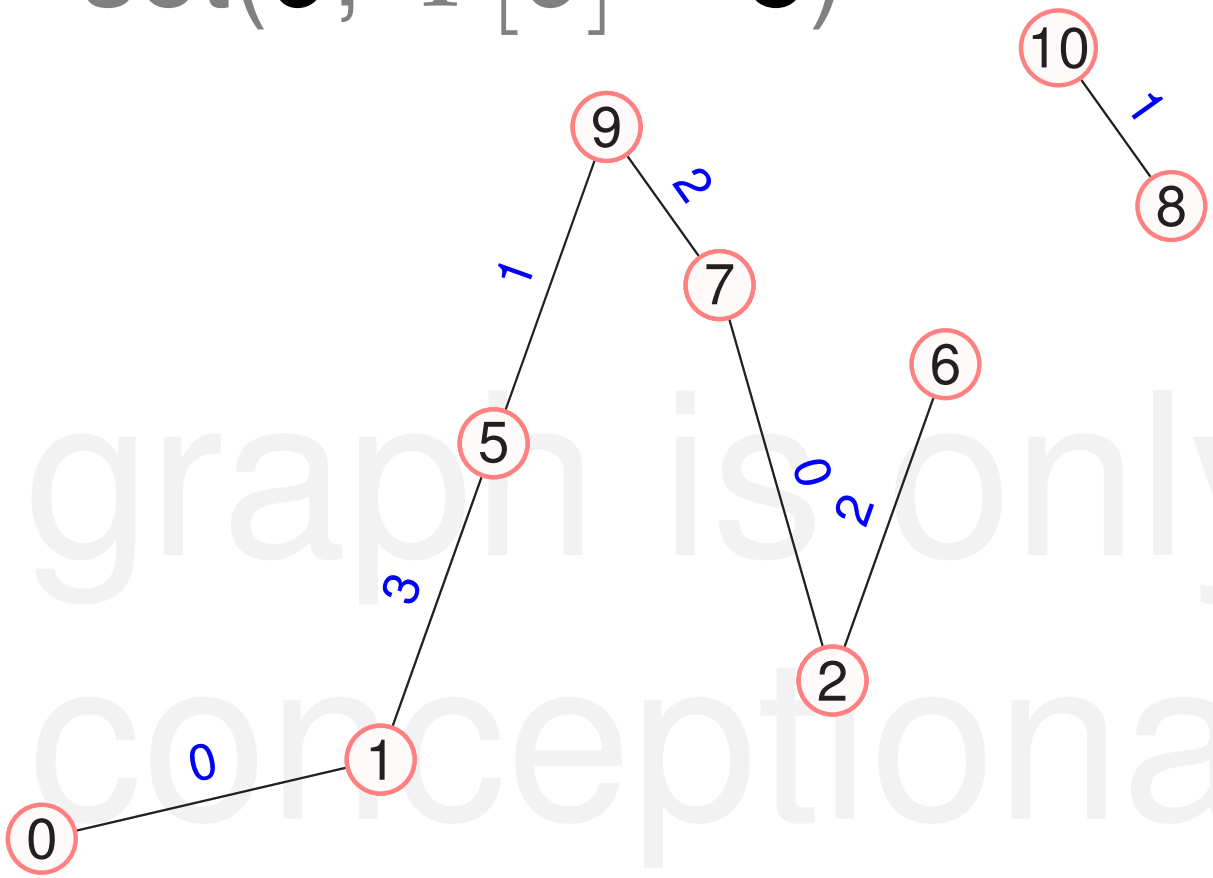
i	0	1	2	3	4	5	6	7	8	9	10	11
LPS[i]		0		1	2	3	2	0		2		
PrevOcc[i]		0		1	3	1	2	2		7		

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



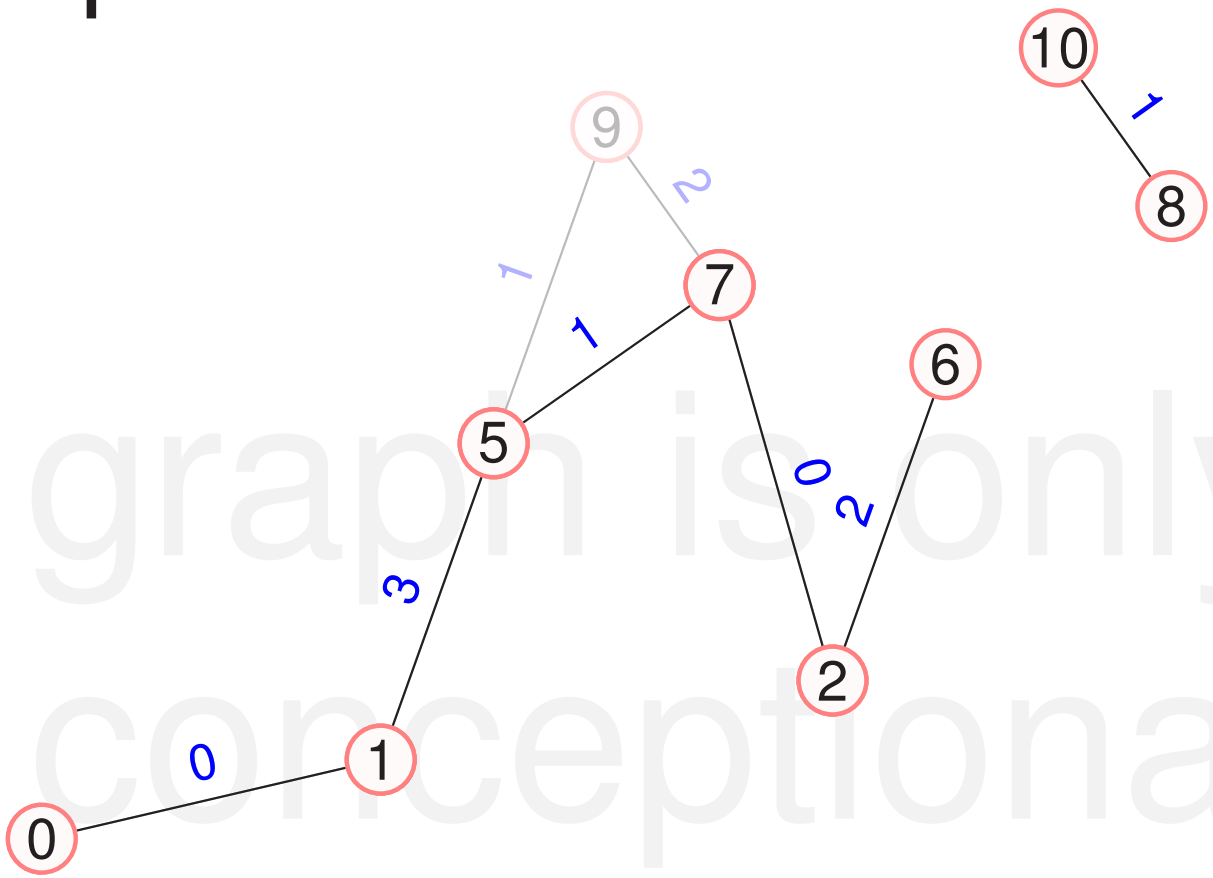
i	0	1	2	3	4	5	6	7	8	9	10	11
LPS$[i]$		0		1	2	3	2	0		2	1	
PrevOcc$[i]$		0		1	3	1	2	2		7	8	

set(9, $\Phi[9] = 5$)



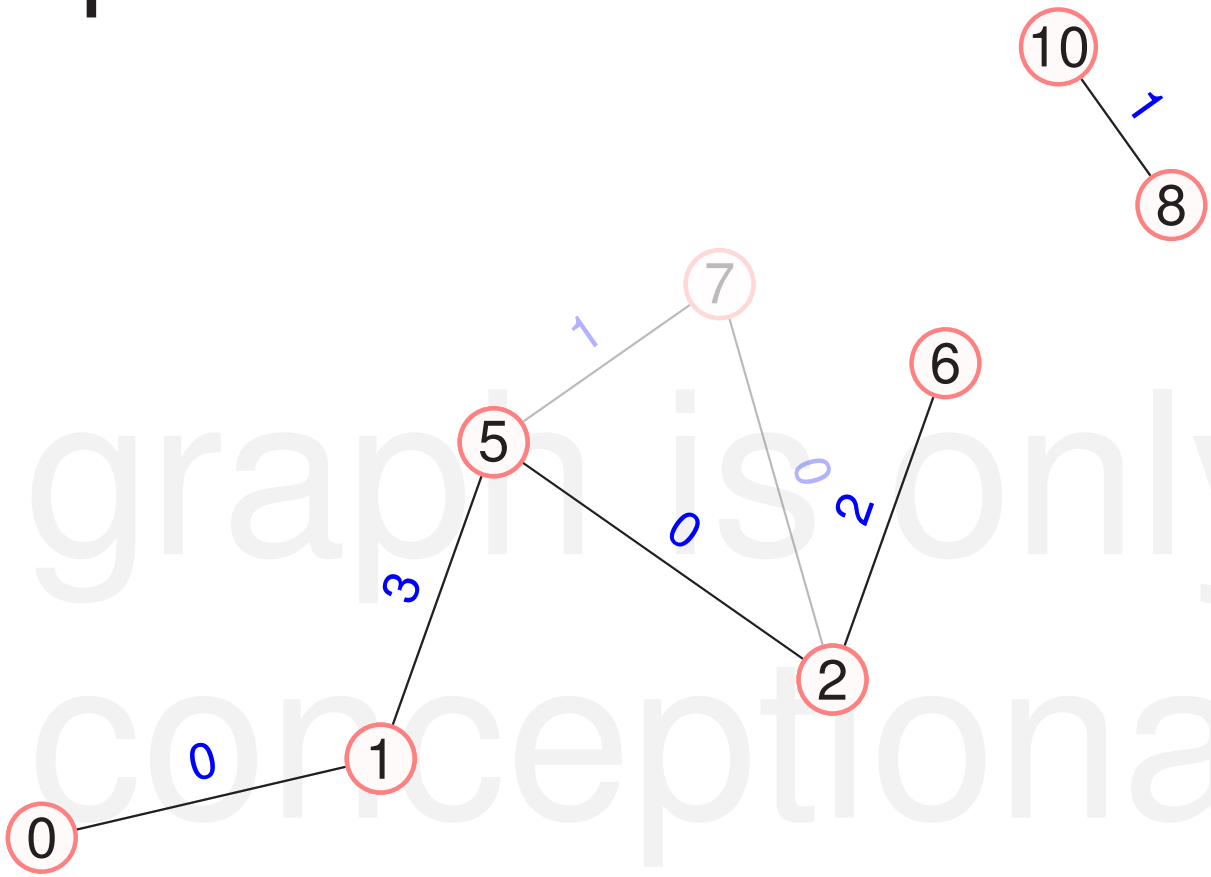
i	0	1	2	3	4	5	6	7	8	9	10	11
LPS$[i]$		0		1	2	3	2	0		1|2	1	
PrevOcc$[i]$		0		1	3	1	2	2		5|7	8	

permute



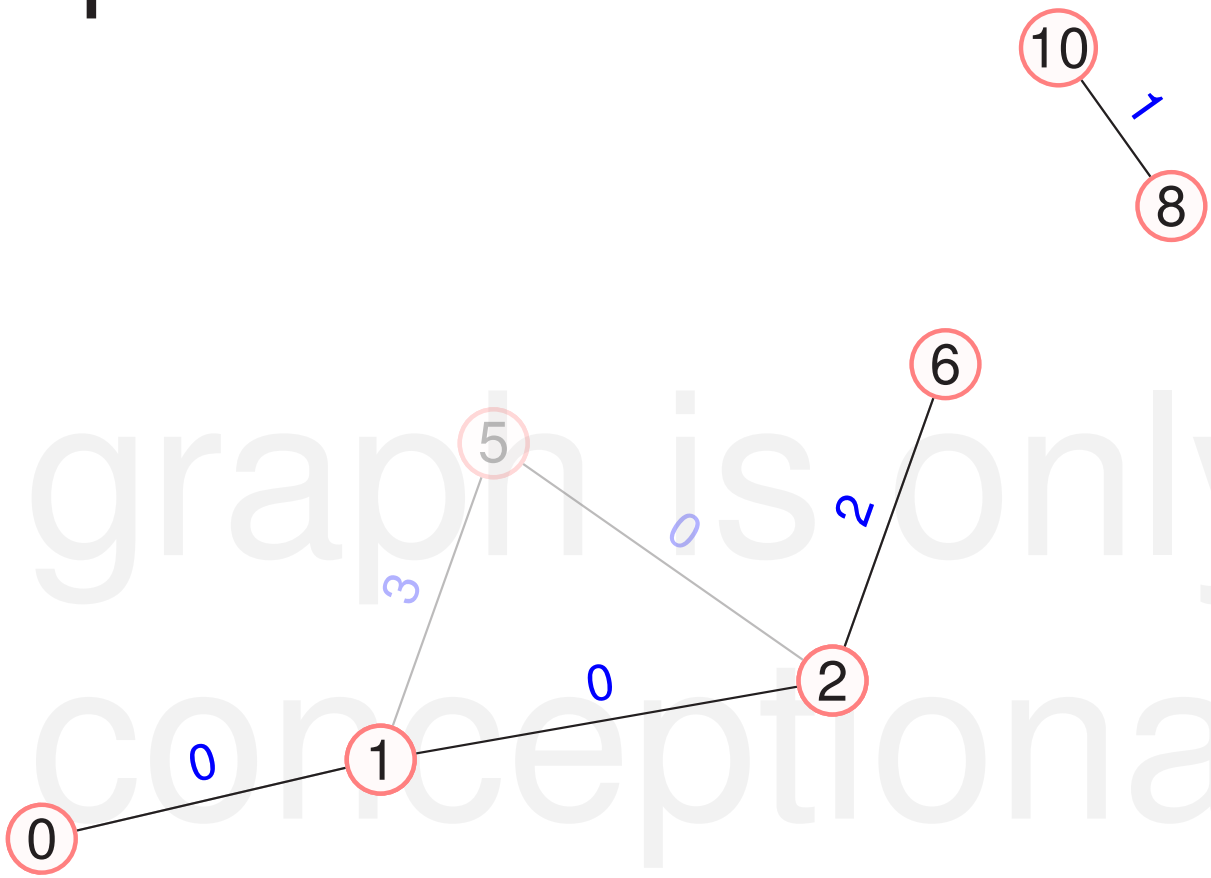
i	0	1	2	3	4	5	6	7	8	9	10	11
LPS$[i]$		0		1	2	3	2	0|1		2	1	
PrevOcc$[i]$		0		1	3	1	2	2|5		7	8	

permute



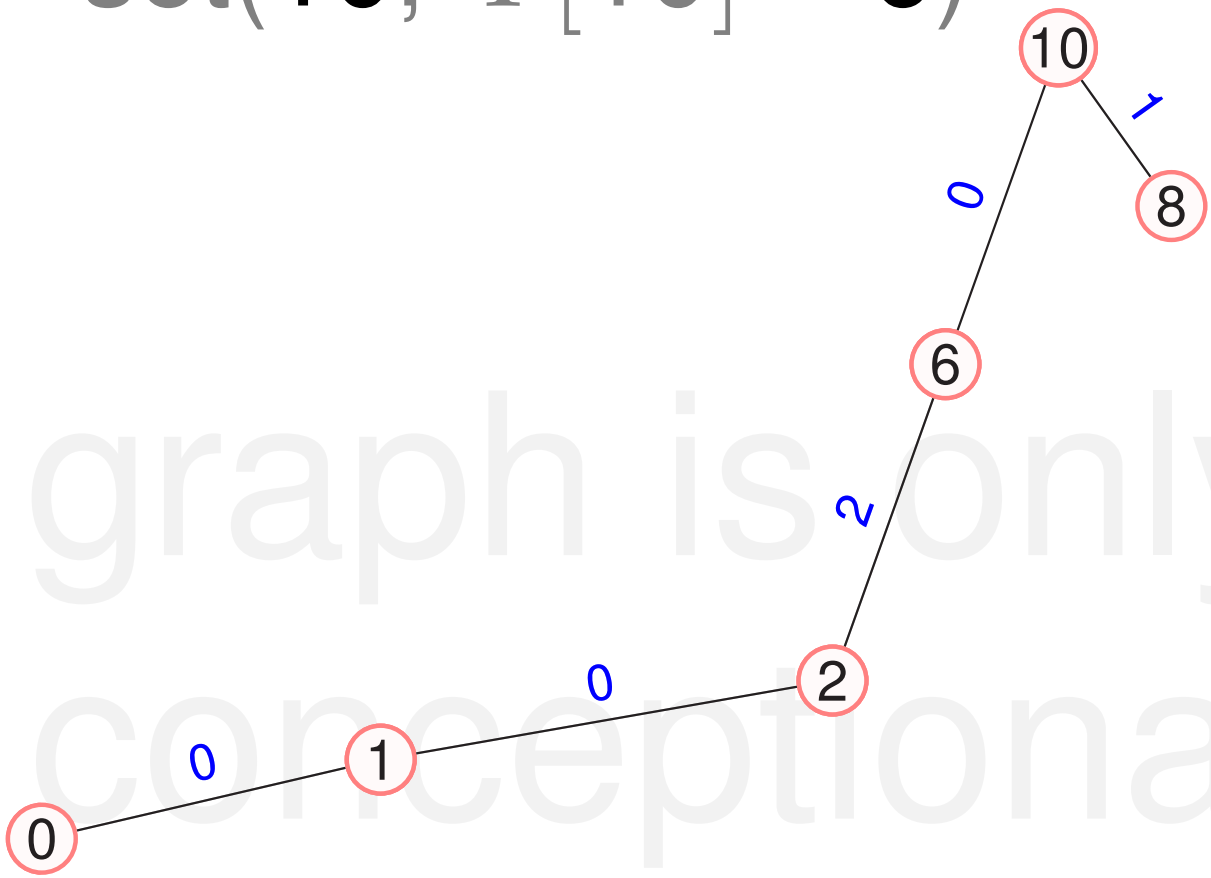
i	0	1	2	3	4	5	6	7	8	9	10	11
LPS$[i]$		0		1	2	0|3	2	1		2	1	
PrevOcc$[i]$		0		1	3	2|1	2	5		7	8	

permute



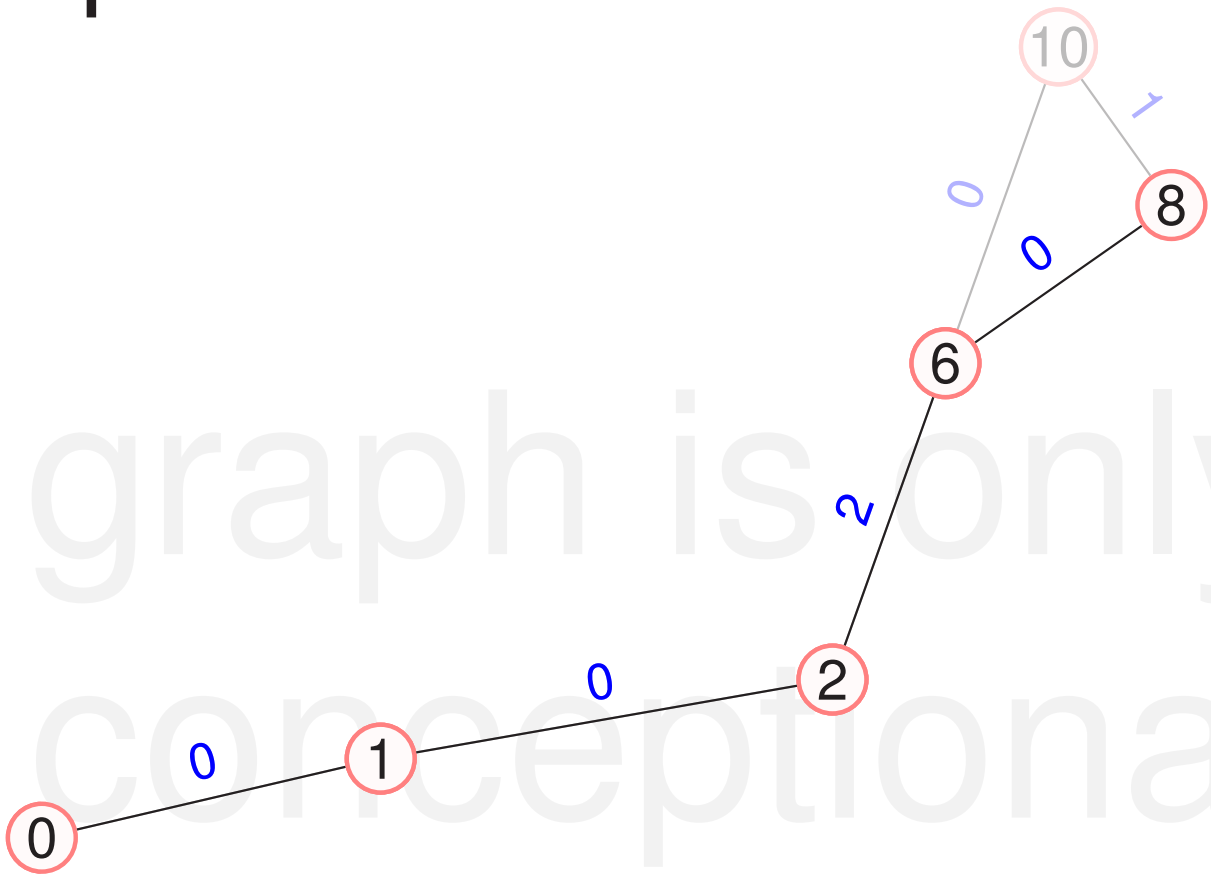
i	0	1	2	3	4	5	6	7	8	9	10	11
LPS$[i]$		0	0	1	2	3	2	1		2	1	
PrevOcc$[i]$		0	1	1	3	1	2	5		7	8	

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



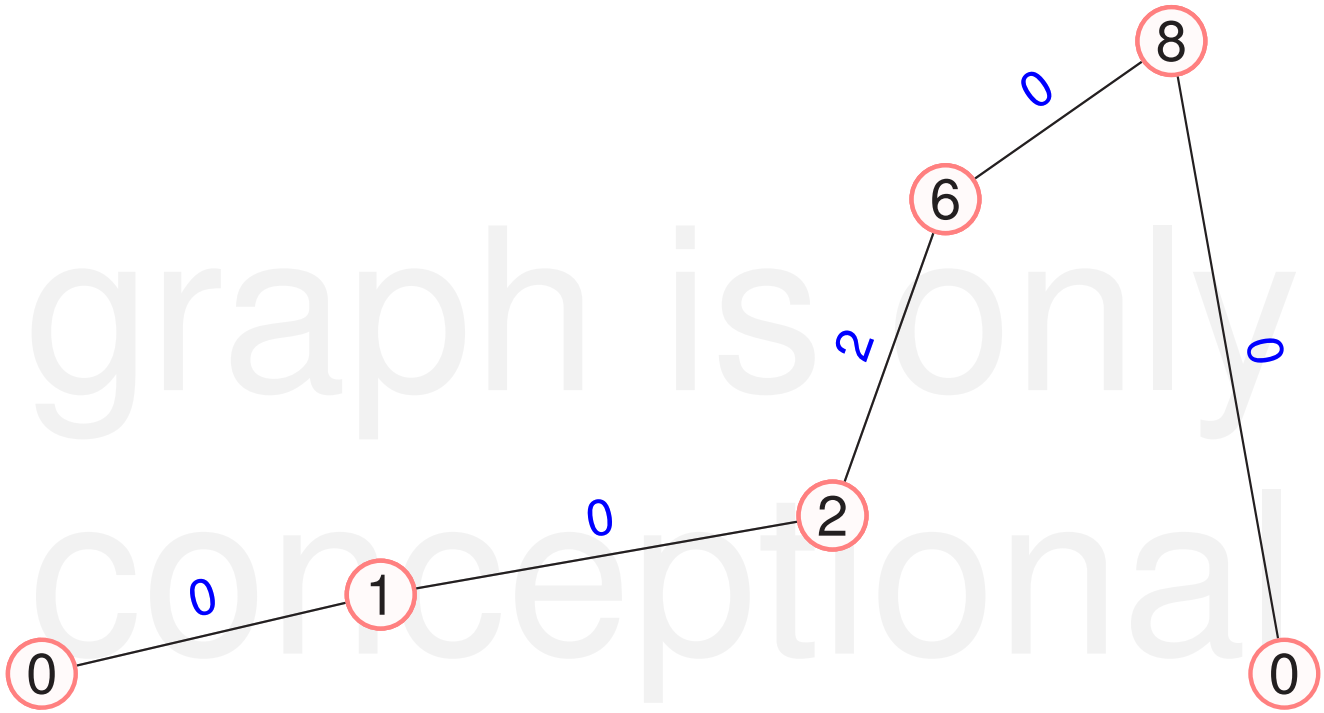
i	0	1	2	3	4	5	6	7	8	9	10	11
LPS$[i]$		0	0	1	2	3	2	1		2	0|1	
PrevOcc$[i]$		0	1	1	3	1	2	5		7	6|8	

permute



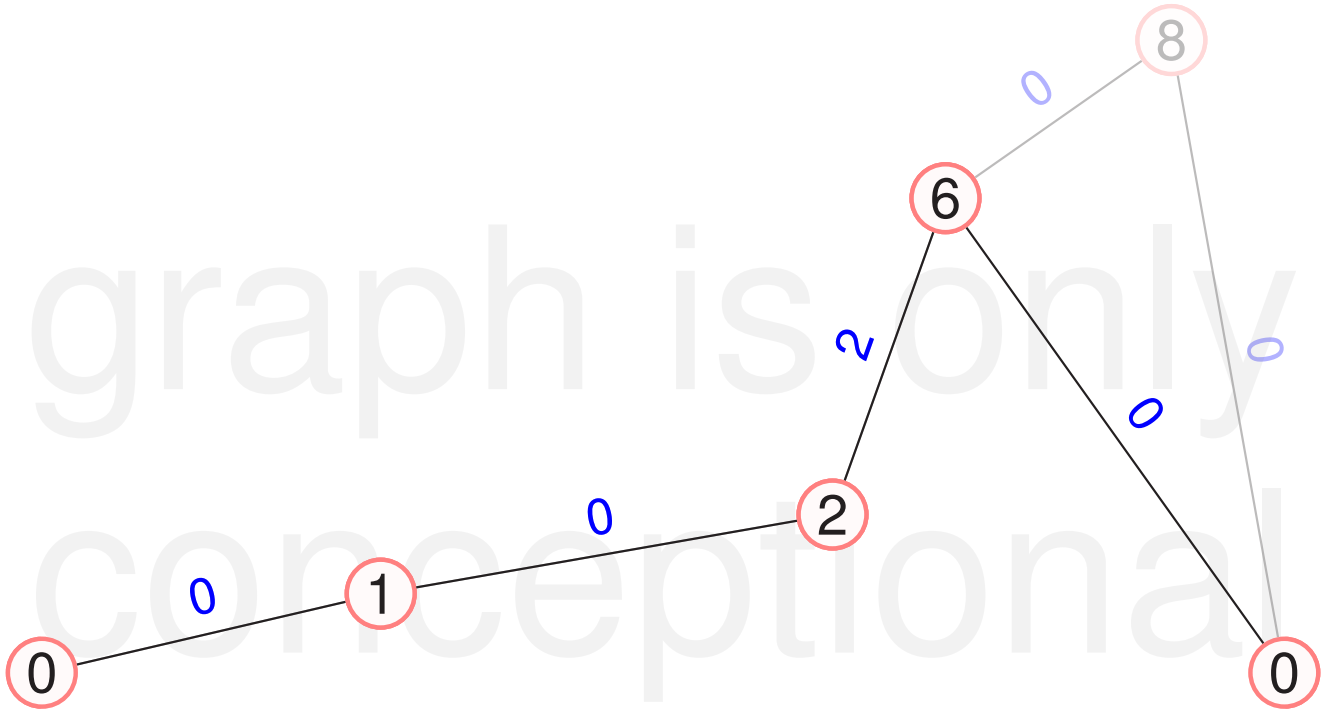
i	0	1	2	3	4	5	6	7	8	9	10	11
LPS$[i]$		0	0	1	2	3	2	1	0	2	1	
PrevOcc$[i]$		0	1	1	3	1	2	5	6	7	8	

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



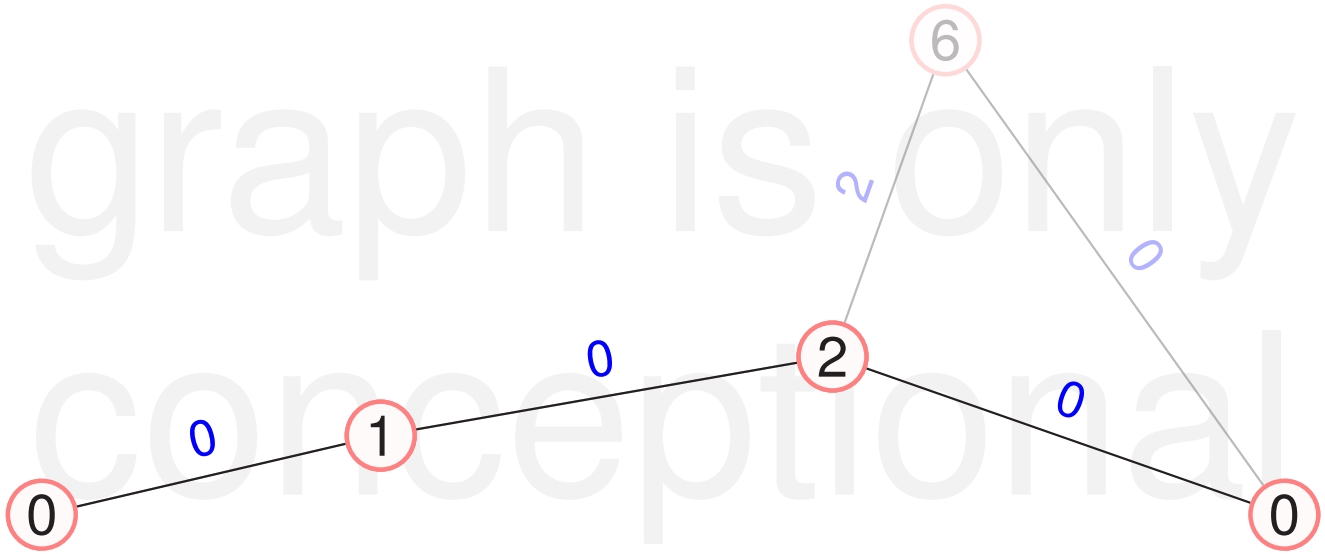
i	0	1	2	3	4	5	6	7	8	9	10	11
LPS$[i]$		0	0	1	2	3	2	1	0|0	2	1	
PrevOcc$[i]$		0	1	1	3	1	2	5	6|11	7	8	

permute



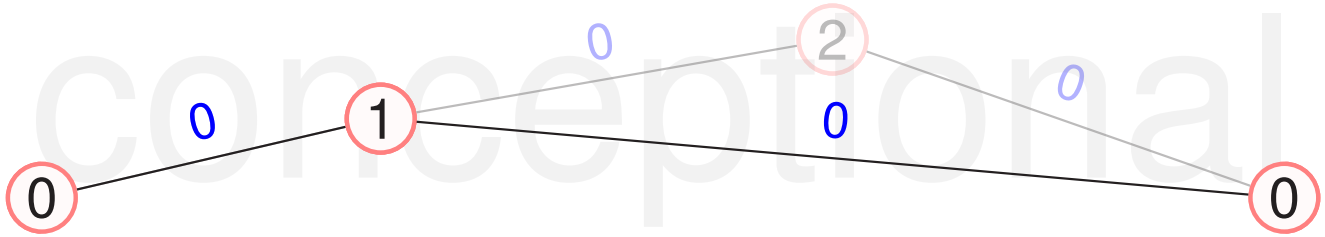
i	0	1	2	3	4	5	6	7	8	9	10	11
LPS$[i]$		0	0	1	2	3	0|2	1	0	2	1	
PrevOcc$[i]$		0	1	1	3	1	11|2	5	6	7	8	

permute



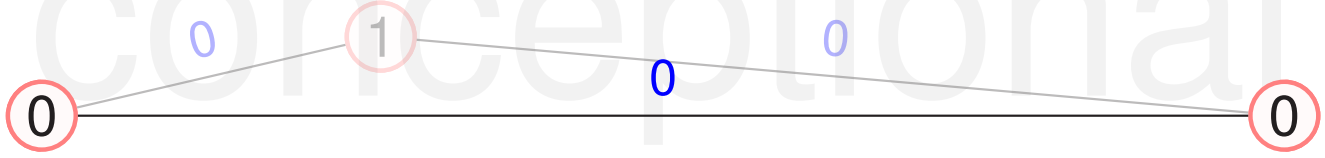
i	0	1	2	3	4	5	6	7	8	9	10	11
LPS$[i]$		0	0|0	1	2	3	2	1	0	2	1	
PrevOcc$[i]$		0	1|11	1	3	1	2	5	6	7	8	

permute



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|11	1	1	3	1	2	5	6	7	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	

Space-Efficient Calculation of LZ Factorization

Karlsruhe Institute of Technology

Exercise

Devise a more space-efficient construction algorithm.

Hint: Use concepts from previous lectures. CSA, backward search, PSV, and NSV.

Literature

[1], Chapter 5.2.

Bibliography

[1] Enno Ohlebusch. *Bioinformatics Algorithms*. Oldenbusch Verlag, 2013.