Compression Boosting

The order zero entropy

Given a string s let

$$H_0(s) = -\sum_i (n_i/n) \log(n_i/n)$$

where n_i is # of occurrences of symbol i, $n=|s|$.

$H_0(s)$ is a lower bound only if we use a fixed codeword for each character.

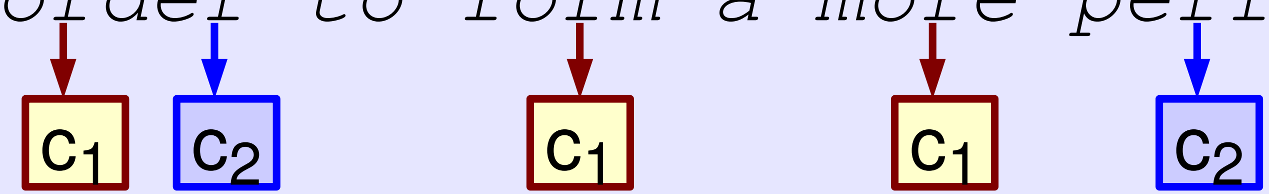
Going further

To compress more, we observe that certain pairs of symbols are more frequent than others...

This suggests new compression algorithms!

Example. For each symbol we use a codeword which depends on the previous symbol.

In order to form a more perfect ...



A lower bound to the compression of such algorithms is

$$H_1(s) = -\sum_j \sum_i (n_{j,i}/n) \log(n_{j,i}/n_j)$$

$n_{j,i}$: # occc pair j,i

n_j: # occ symbol j

Generalization: the k-th order entropy $H_k(s)$ is a lower bound if we use a codeword which depends on the previous k symbols.

Another view of $H_k(s)$ (1)

Let s = *ippississim*,

s^R = *mississippi*

BWT(s^R) =

imississip	p
ippimissis	s
issippimis	s
ississippi	m
mississipp	i
pimississi	p
ppimississ	i
sippimissi	s
sissippimi	s
ssippimiss	i
ssissippim	i

Another view of $H_k(s)$ (1)

Let s = *ippississim*,

s^R = *mississippi*

$BWT(s^R)$ =

F		L
i	mississip	p
i	ppimissis	s
i	ssippimis	s
i	ssissippi	m
m	ississipp	i
p	imississi	p
p	pimississ	i
s	ippimissi	s
s	issippimi	s
s	sippimiss	i
s	sissippim	i

$H_1(s) = (4/11)\, H_0(pssm) + (1/11)\, H_0(i) + (2/11)\, H_0(pi) + (4/11)\, H_0(ssii)$

To compress up to $H_1(s)$ it suffices to compress each segment up to H_0

Another view of $H_k(s)$ (2)

Let s = *ippississim*,

s^R = *mississippi*

BWT(s^R) =

```
imississipp
ippimississ
issippimiss
ississippim
mississippi
pimississip
ppimississi
sippimissis
sissippimis
ssippimissi
ssissippimi
```

To compress up to $H_k(s)$ it suffices to compress each segment up to H_0

Summing up

To compress a string up to $H_k(s)$ it suffices to compress a partition of $BWT(s^R)$ up to H_k

However for each k there is a different overhead, so the choice of the partitioning strategy is non-trivial

Assume now we are a given an Order-0 compressor C such that for any s

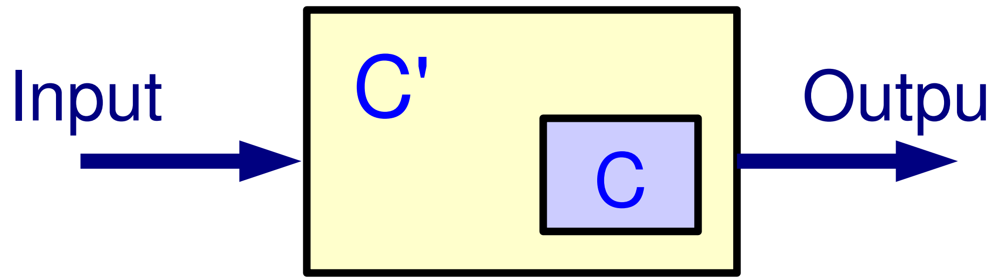
$$|C(s)| \leqslant |s|H_0(s) + v\,|s|$$

We can combine C with the BWT to obtain a new algorithm C’ such that for any s

$$|C'(s)| \leqslant |s|H_k(s) + v\,|s| + \log|s| + g_k$$

for any $k \geq 0$.

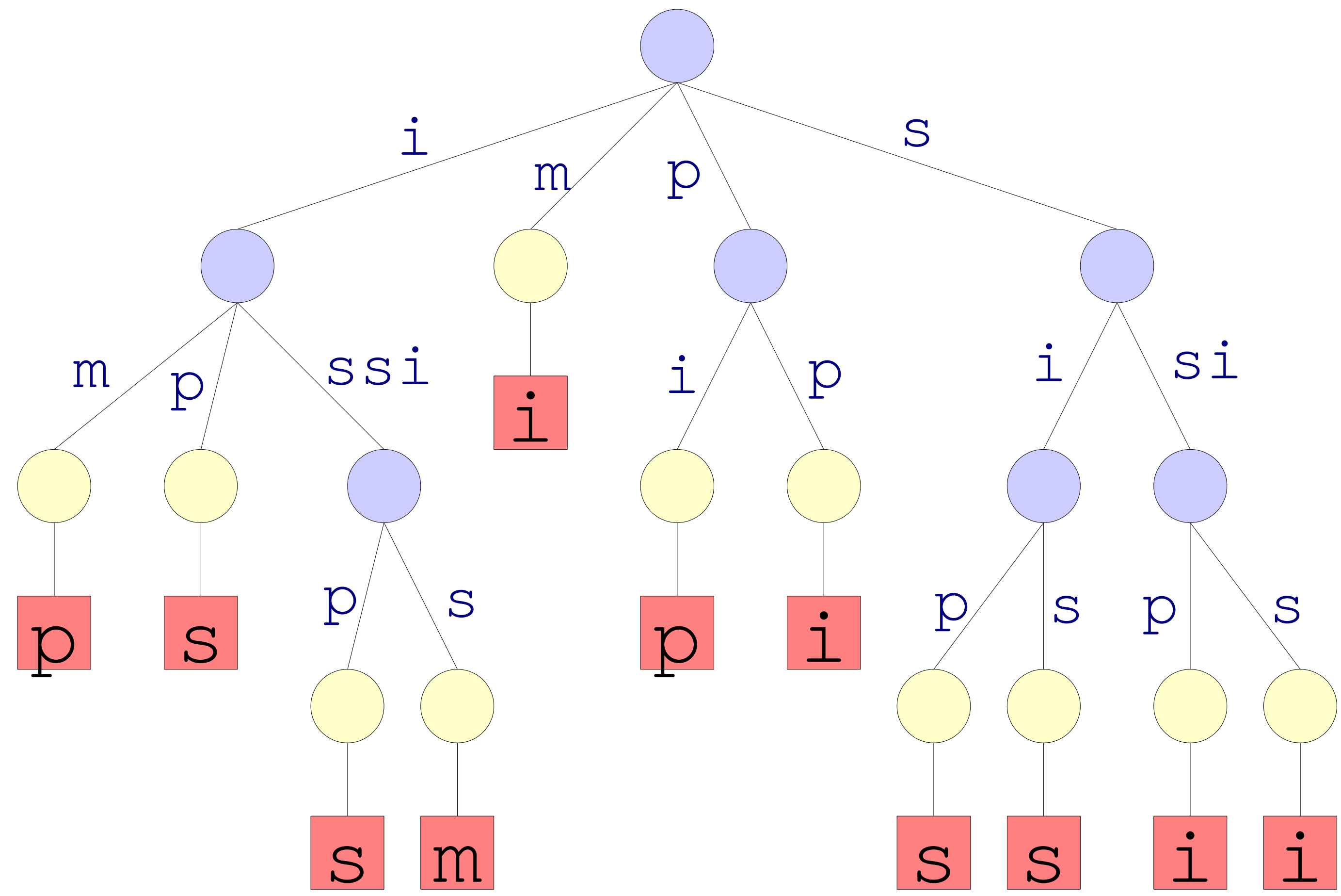
Compression Boosting



Given a memoryless algorithm C we use it as a black-box and we obtain an order-k algorithm C'.

BWT matrix vs Suffix Tree

imississipp
ippimississ
issippimiss
ississippim
mississippi
pimississip
ppimississi
sippimissis
sissippimis
ssippimissi
ssissippimi



Rows (top to bottom) correspond to leaves (left to right)

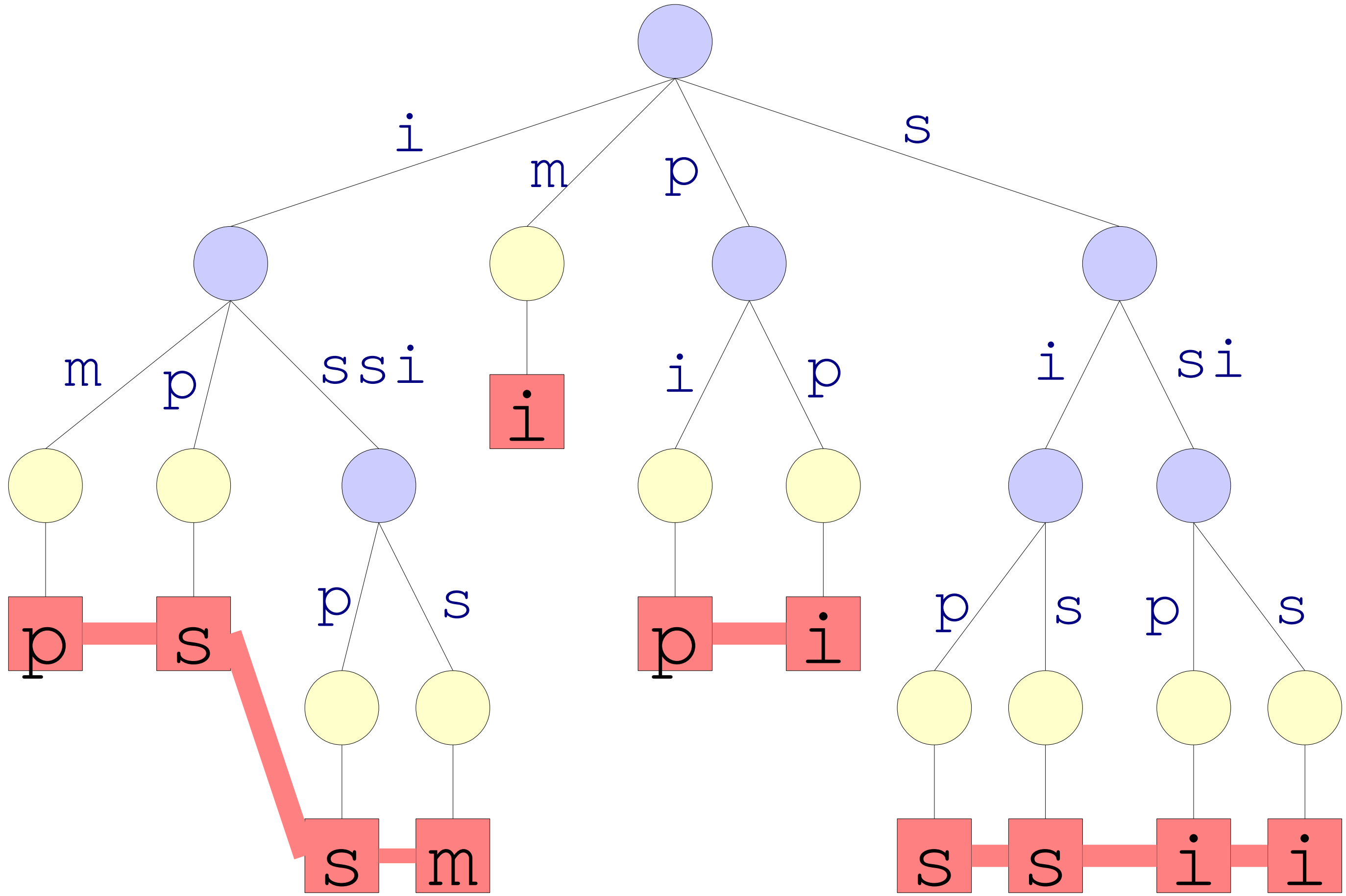
Key observation

We are interested only in the BWT partitions which enter in the definition of H_k for some $k \geq 0$.

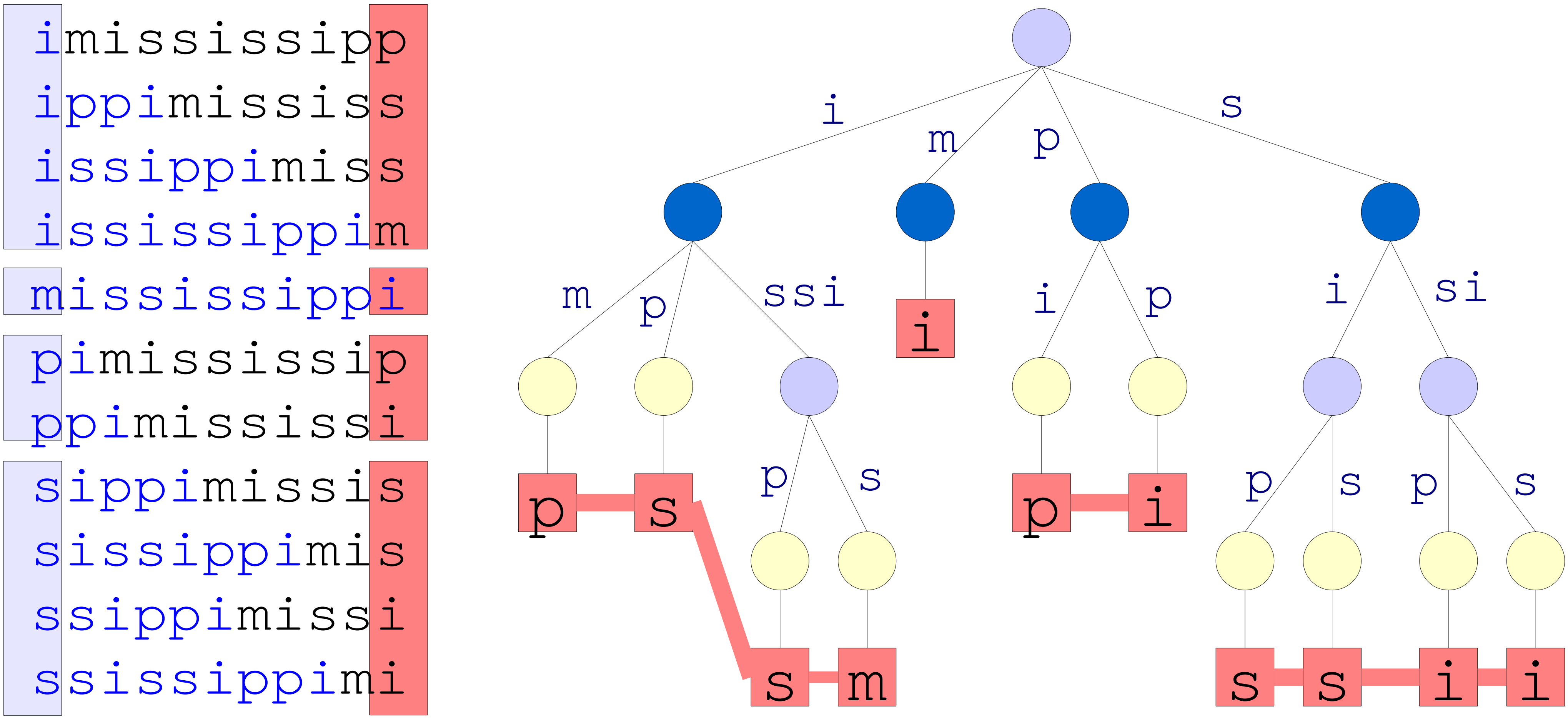
Each "interesting" BWT partition is induced by a set L of suffix tree nodes called a leaf cover.

Leaf cover corresponding to H_1

```
imississipp
ippimississ
issippimiss
ississippim
mississippi
pimississip
ppimississi
sippimissis
sissippimis
ssippimissi
ssissippimi
```



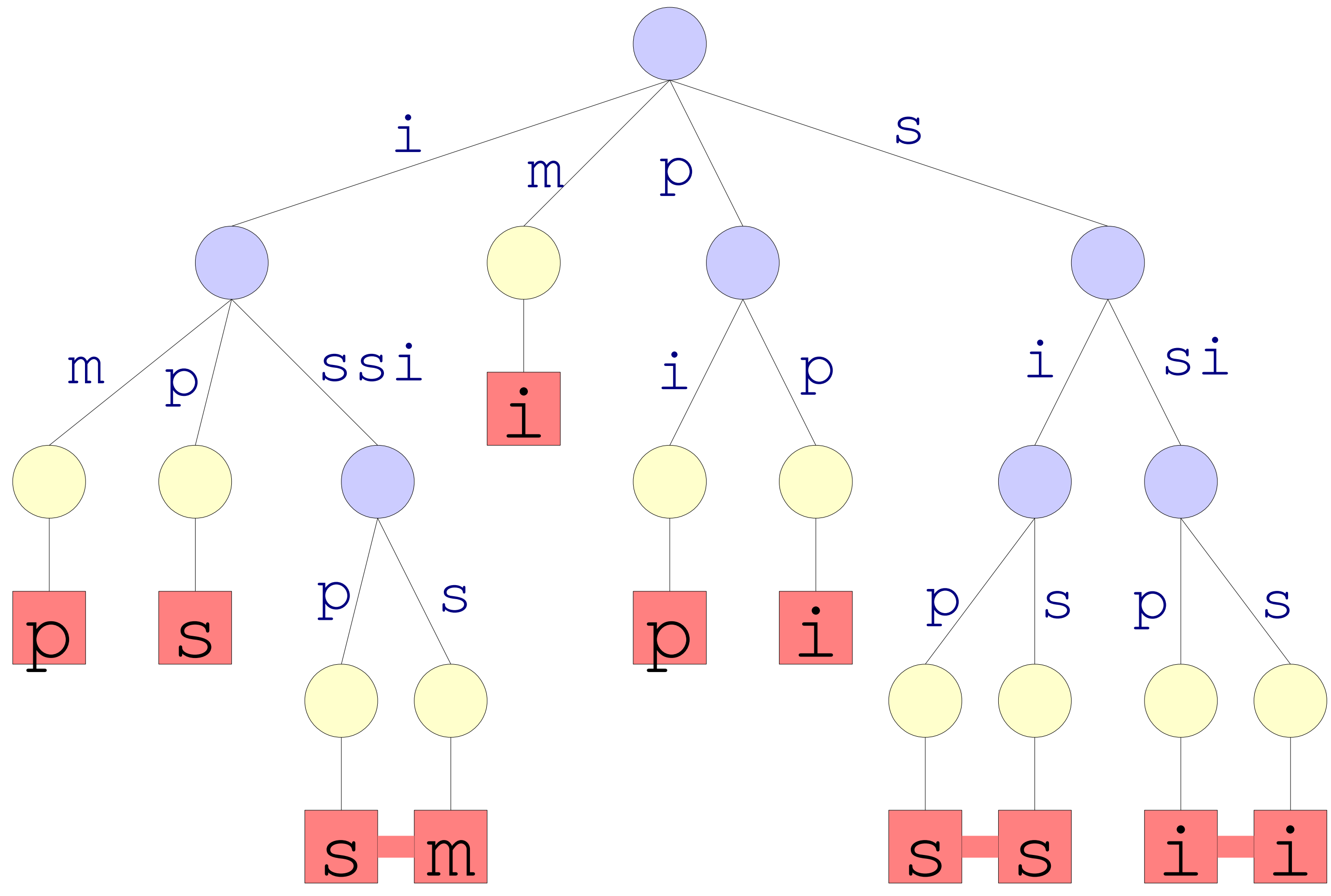
Leaf cover corresponding to H_1



The leaf cover consists in the lowest common ancestors of leaves belonging to the same group.

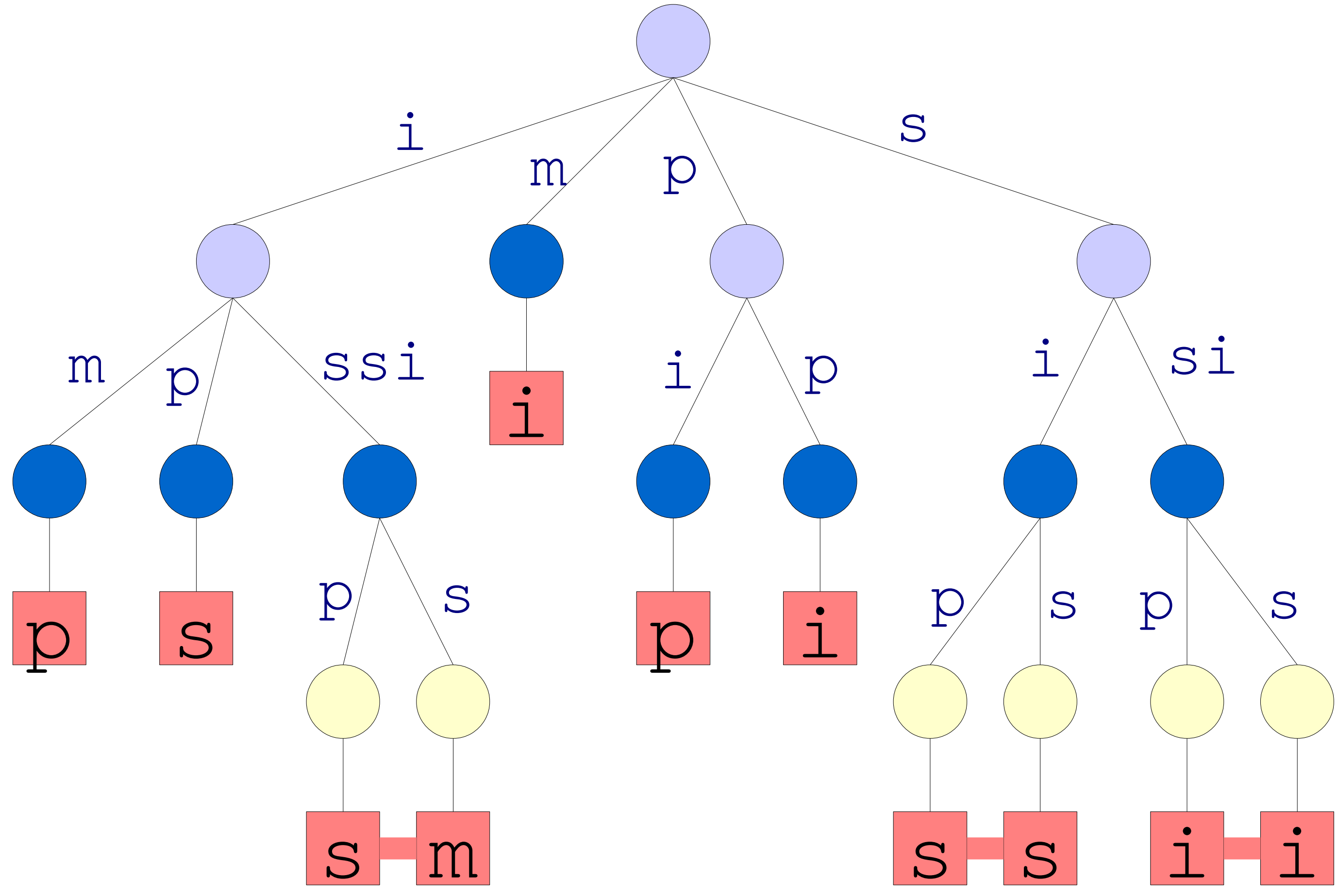
Leaf cover corresponding to H_2

imississipp
ippimississ
issippimiss
ississippim
mississippi
pimississip
ppimississi
sippimissis
sissippimis
ssippimissi
ssissippimi



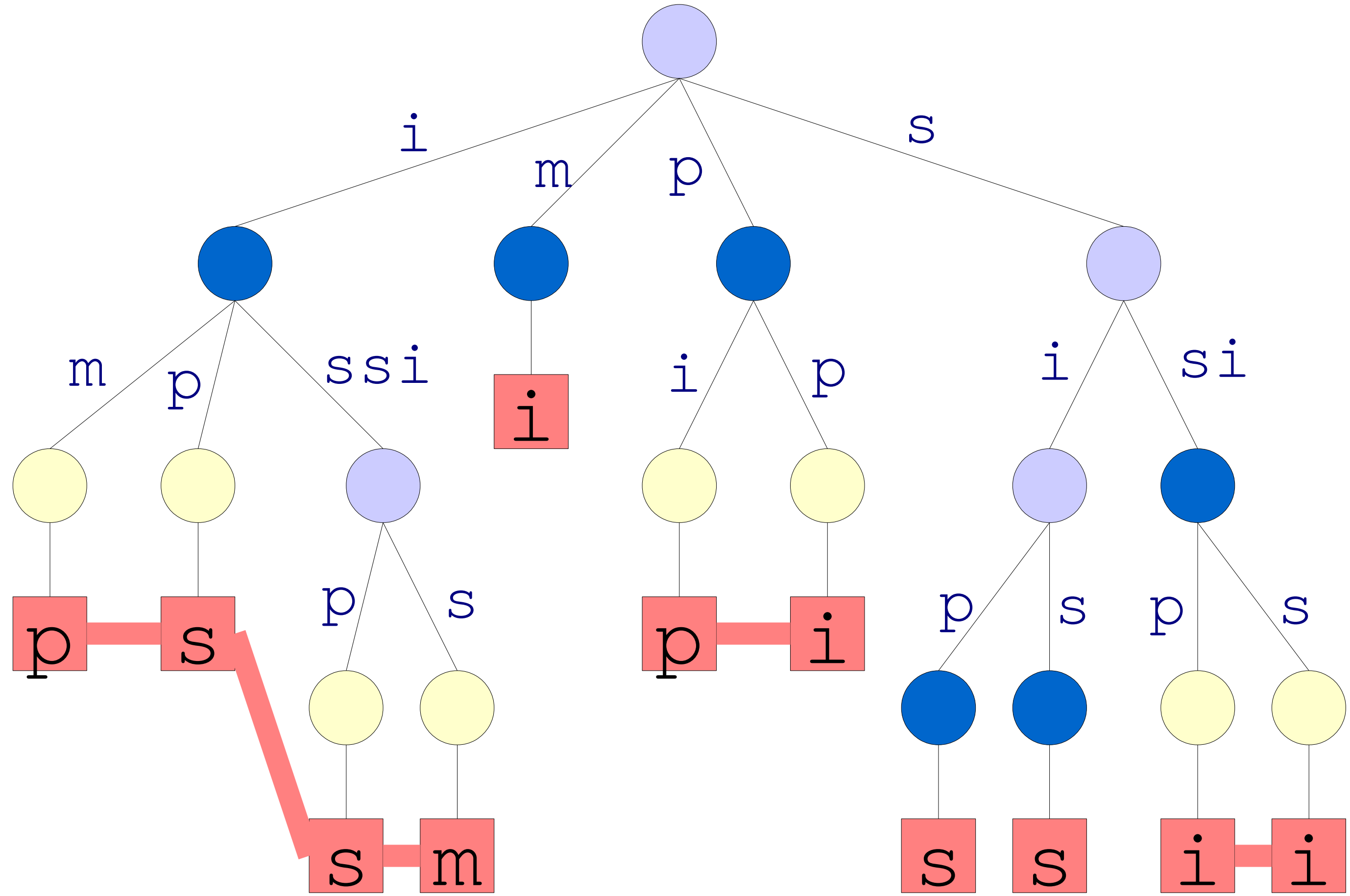
Leaf cover corresponding to H_2

```
imississipp
ippimississ
issippimiss
ississippim
mississippi
pimississip
ppimississi
sippimissis
sissippimis
ssippimissi
ssissippimi
```



Another leaf cover...

imississipp
ippimississ
issippimiss
ississippim
mississippi
pimississip
ppimississi
sippimissis
sissippimis
ssippimissi
ssissippimi



A **leaf cover** has the property that **every leaf has a unique ancestor** in the set.

For any leaf cover L we define

$$\mathrm{Cost}(L) = \sum_i \; |s_i| H_0(s_i) + v\,|s_i|$$

where s_1, s_2, ... is the partition corresponding to L.

We compute the leaf cover L^* of minimum cost;

for any $k \geq 0$ we have

$$\mathrm{Cost}(L^*) \leqslant \mathrm{Cost}(L_k) \leqslant |s| H_k(s) + v|s|$$

Leaf cover defining $H_k(s)$

Surprisingly, the optimal leaf cover *L** can be found with a post-order visit of the suffix tree which takes linear time and linear space.

We do not need to actually build the suffix tree: for the post-order visit we only need two integer arrays of length |s|: the suffix array and lcp array.