# Lecture 2:
# FM-indexes

February 17th, 2016

# Components

With an FM-index we can quickly count the occurrences of patterns in a string and then locate those occurrences. It uses lots of fun things:

- a wavelet tree supporting rank queries on the Burrows-Wheeler Transform (BWT);
- an array storing the partial sums of the characters' frequencies;
- a bitvector marking the cells in the suffix array (SA) that store $\log^{1+\epsilon} n, 2\log^{1+\epsilon} n, 3\log^{1+\epsilon} n, \ldots$;
- an array storing those cells' contents — i.e., $\log^{1+\epsilon} n, 2\log^{1+\epsilon} n, 3\log^{1+\epsilon} n, \ldots$ — in the order they appear in the SA.

# Rank on Binary Strings

## Definition

For a binary string $B[1..n]$, $rank(i)$ is the number of 1s in $B[1..i]$.

(Notice that the number of 0s in $B[1..i]$ is $i - rank(i)$.)

# Succinct Solution

We break $B$ into blocks of length $\log^2 n$. We store $rank(i)$ for the starting position $i$ of every block; this takes a total of $\mathcal{O}\left(\frac{n}{\log^2 n} \cdot \log n\right) = o(n)$ bits. We then break each block into mini-blocks of length $\frac{\log n}{2}$. For each mini-block, we store the number of 1s between the start of the block and the start of the mini-block; this takes a total of $\mathcal{O}\left(\frac{n}{\log n} \cdot \log \log n\right) = o(n)$ bits.

# Succinct Solution

We build a *universal table* mapping (mini-block, position)-pairs to ranks within mini-blocks. This table takes $\mathcal{O}\left(2^{\log(n)/2} \cdot \log n \log\log n\right) = o(n)$ bits. In the RAM model, $\frac{\log n}{2}$ bits fit in $\mathcal{O}(1)$ machine words, so we can look up a rank in a mini-block in $\mathcal{O}(1)$ time.

# Succinct Solution

**Theorem**

*We can store a binary string* $B[1..n]$ *in* $n + o(n)$ *bits such that rank takes* $\mathcal{O}(1)$ *time.*

# Compressed Solution (Sketch)

We encode each mini-block $M$ of $B$ by writing i) the number of 1s in $M$ and ii) $M$'s lexicographic rank among all $\frac{\log n}{2}$-bit binary strings with that many 1s. Suppose there are $b$ 1s in $M$, so we use $\log\log n + \log \binom{\log(n)/2}{b} + \mathcal{O}(1)$ bits.

If there are roughly the same number of 1s and 0s in $M$, then $\log \binom{\log(n)/2}{b} \approx \frac{\log n}{2}$; however, the more skewed the distribution is, the fewer bits we use. Notice we're little pieces of $B$ separately.

# Compressed Solution (Sketch)

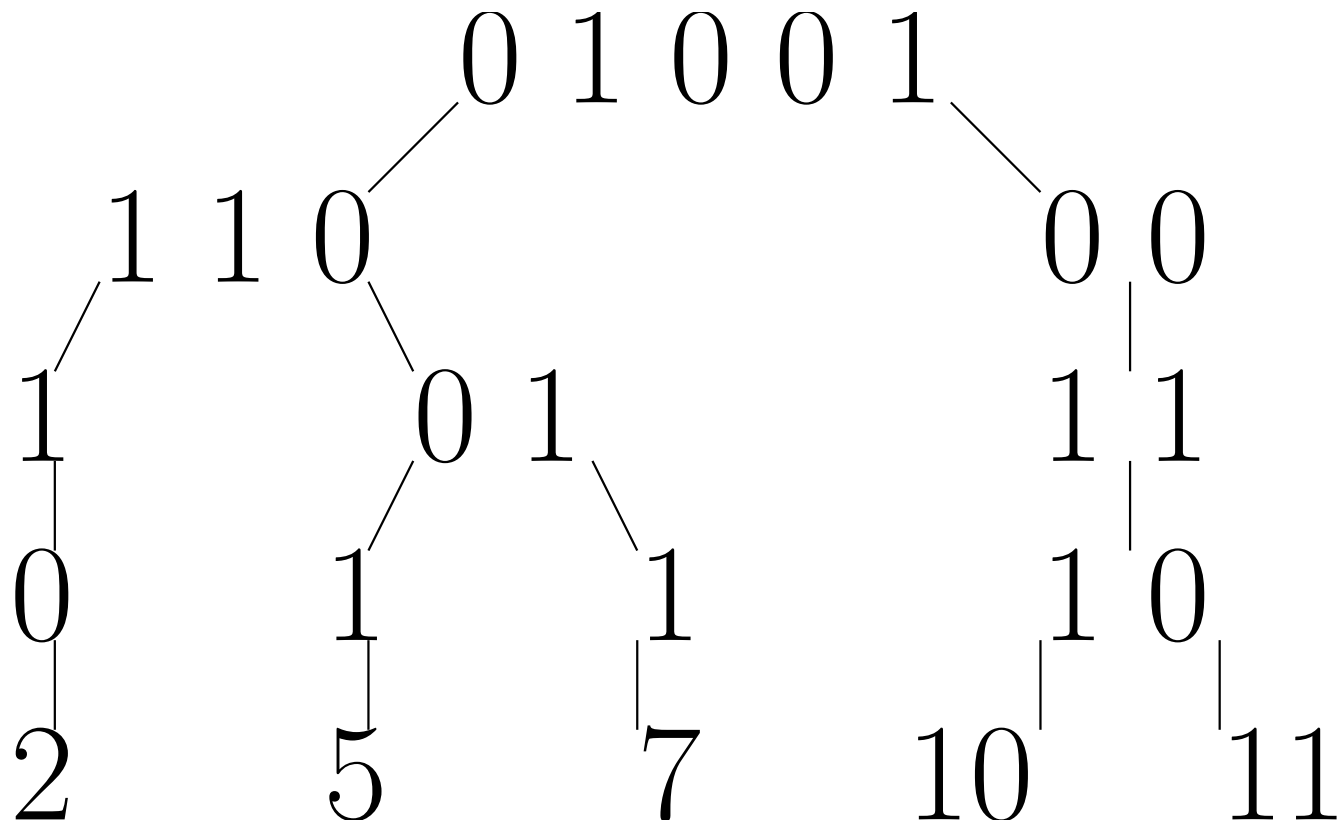
**Theorem**

*We can store a binary string* $B[1..n]$ *in*

$$\sum_i \frac{\log n}{2} \cdot H_0(M_i) + o(n) \quad \leq \quad nH_0(B) + o(n)$$

*bits such that rank takes* $\mathcal{O}(1)$ *time.*

# Wavelet Trees




The wavelet tree for $5, 11, 7, 2, 10$.

# Wavelet Trees

The root stores the first bits of all the numbers. The root's left child stores the second bits of all the numbers that start with a 0; the root's right child stores the second bits of all the numbers that start with a 1. The

Notice the tree has height $\lceil \lg \sigma \rceil$ and all the nodes together store $n\lceil \log \sigma \rceil$ bits unencoded.

In fact, if we encode the bits at all the nodes with our compressed solution for *rank*, then we use only $nH_k(S) + o(n \log \sigma)$ bits.

# Rank for Larger Alphabets

| 3 | 1 | 2 | 0 | 0 | 3 | 1 | 2 | 3 | 3 |
|---|---|---|---|---|---|---|---|---|---|
| 1 | 0 | 1 | 0 | 0 | 1 | 0 | 1 | 1 | 1 |

| 1 | 0 | 0 | 1 | | 3 | 2 | 3 | 2 | 3 | 3 |
|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 0 | 0 | 1 | | 1 | 0 | 1 | 0 | 1 | 1 |

How many 3s are there up to position 7?

# Burrows-Wheeler Transform (BWT)

Recall that BWT[$i$] = S[(SA[$i$] - 1) mod (n + 1)] [1] Thus, if we sort the characters in $BWT(S)$, we obtain a list whose $i$th element is the first character of the lexicographically $i$th suffix of $S$.

| | |
|---|---|
| A | $ |
| R | -A-LA-ALABARDA$ |
| A | -ALABARDA$ |
| A | -LA-ALABARDA$ |
| D | A$ |
| L | A-ALABARDA$ |
| - | A-LA-ALABARDA$ |
| L | ABAR-A-LA-ALABARDA$ |
| L | ABARDA$ |
| $ | ALABAR-A-LA-ALABARDA$ |
| - | ALABARDA$ |
| B | AR-A-LA-ALABARDA$ |
| B | ARDA$ |
| A | BAR-A-LA-ALABARDA$ |
| A | BARDA$ |
| R | DA$ |
| - | LA-ALABARDA$ |
| A | LABAR-A-LA-ALABARDA$ |
| A | LABARDA$ |
| A | R-A-LA-ALABARDA$ |
| A | RDA$ |

[1] Again, subject to off-by-one errors depending on whether you count from 0 and whether $n$ includes $.

BWT(ALABAR-A-LA-ALABARDA) = ARAADL-LL$-BBAAR-AAAA

The partial sums of the frequencies — 1 $, 3 -'s, 9 As, 2 Bs, 1 D, 3 Ls, 2 Rs — are C = 0, 1, 4, 13, 15, 16, 19

Setting $ = 0, - = 1, A = 2, B = 3, D = 4, L = 5, R = 6, we get

2 6 2 2 4 5 1 5 5 0 1 3 3 2 2 6 1 2 2 2 2

# Counting

$\{\$ = 0, \text{-} = 1, A = 2, B = 3, D = 4, L = 5, R = 6\}$

$C = 0, 1, 4, 13, 15, 16, 19$

| 2 | 6 | 2 | 2 | 4 | 5 | 1 | 5 | 5 | 0 | 1 | 3 | 3 | 2 | 2 | 6 | 1 | 2 | 2 | 2 | 2 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 1 | 0 | 0 | 1 | 1 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 |

| 2 | 2 | 2 | 1 | 0 | 1 | 3 | 3 | 2 | 2 | 1 | 2 | 2 | 2 | 2 | | 6 | 4 | 5 | 5 | 5 | 6 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 1 | 1 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 0 | 1 | 1 | 1 | 1 | | 1 | 0 | 0 | 0 | 0 | 1 |

| 1 | 0 | 1 | 1 | | 2 | 2 | 2 | 3 | 3 | 2 | 2 | 2 | 2 | 2 | 2 | | 4 | 5 | 5 | 5 | | 6 | 6 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 0 | 1 | 1 | | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | | 0 | 1 | 1 | 1 | | 0 | 0 |

| 0 | 1 | 1 | 1 | | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 3 | 3 | | 4 | 5 | 5 | 5 | | 6 | 6 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|

How many occurrences of BAR are there? Of LA?

# Locating

Counting occurrences of $P$ and finding their positions in $BWT(S)$ is one thing, but we often want to know where there are in $S$. We can determine that if we have $SA(S)$, of course, but that takes $\Theta(n \log n)$ bits.

Suppose we store a bitvector with 1s marking the cells in $SA(S)$ that contain multiples of $\log^{1+\epsilon} n$ and an array storing those values in the order they appear in $SA(S)$. This takes $o(n)$ bits.

# Locating

Once we've found in $BWT(S)$ the range containing the characters that precede occurrences of $P$ in $S$, for each such character, we can backstep until we reach a position marked by a 1. We perform a rank to determine which 1 we're at, then look in the array to tell which character of $S$ we're at.
Adding the number of backsteps we've taken gives us the position in $S$ of the character we started at. This takes a total of $\mathcal{O}\left(\log^{1+\epsilon}(n) \log \sigma\right)$ time.

# Summary

## Theorem

*We can store a string* $S[1..n]$ *over an alphabet of size* $\sigma$ *in* $nH_k(S) + o(n \log \sigma)$ *bits such that, given a pattern* $P[1..m]$*, we can count the occ occurrences of* $P$ *in* $S$ *in* $\mathcal{O}(m \log \sigma)$ *time and locate each occurrence in* $\mathcal{O}\left(\log^{1+\epsilon} n \cdot \log \sigma\right)$ *time.*[2]

[2] Actually, we can reduce the $\log \sigma$ factors — but not in this course.

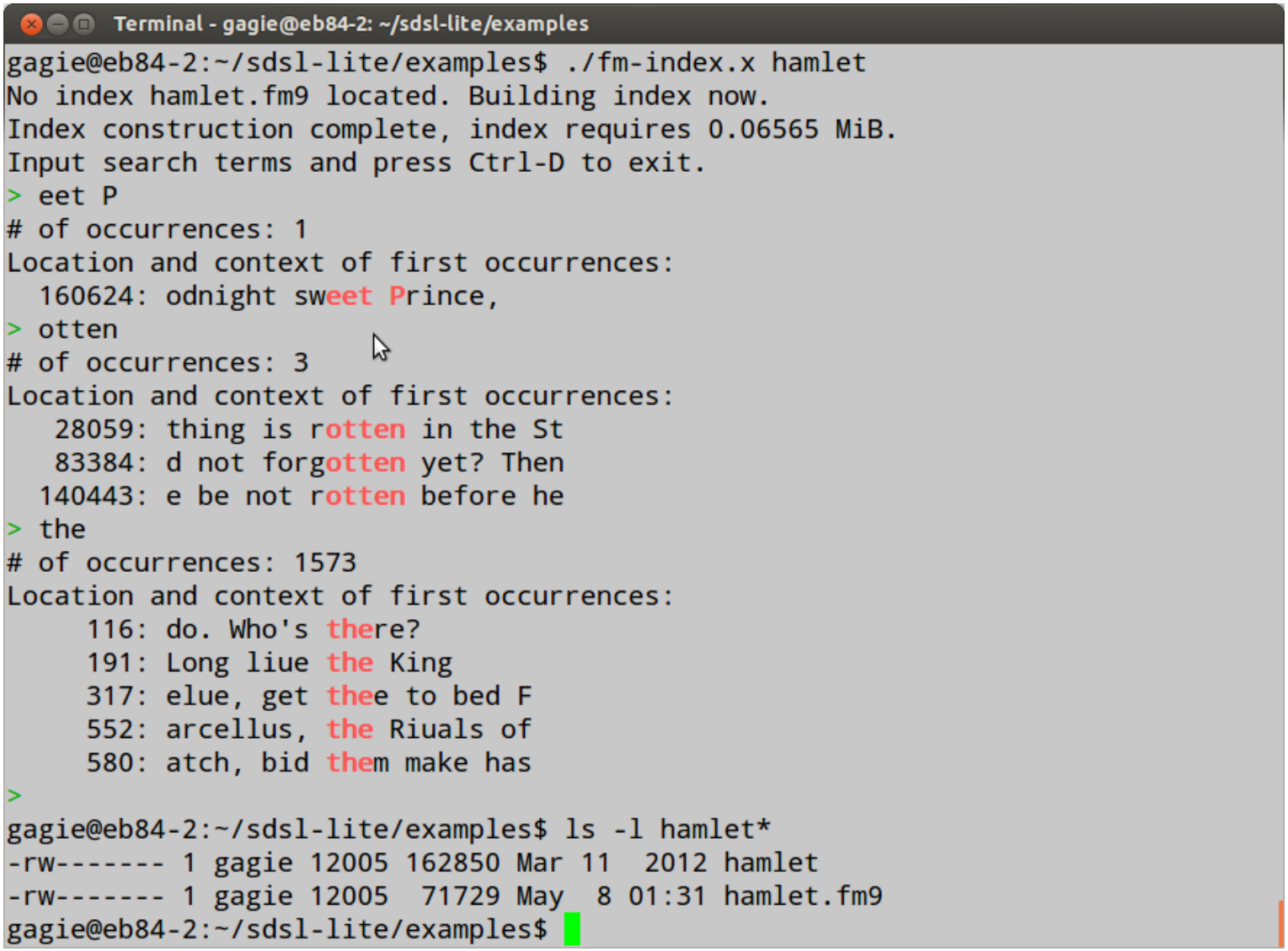
# Implementation

You can experiment with FM-indexes using SDSL. Check you have g++ version 4.7 or later, then install it with:

```
git clone https://github.com/simongog/sdsl-lite.git
cd sdsl-lite
./install.sh
```

Go into `examples` and `make` them, then try `./fm-index text-file`.

# Implementation




The FM-index is 44% as big as the ASCII file; `hamlet.7z` is 37% and `hamlet.bz2` is 33%.