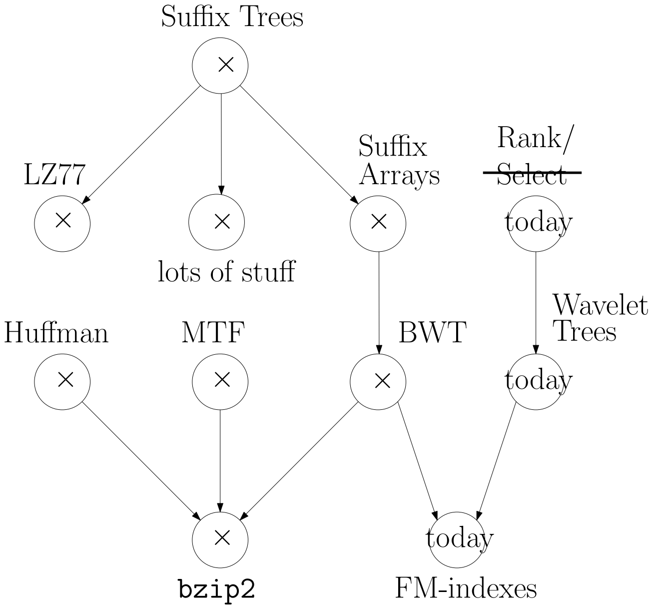
Lecture 6:
FM-indexes

March 19th, 2012

Plan


Suffix Trees
×
LZ77
×
×
lots of stuff
Suffix
Arrays
×
Rank/
~~Select~~
today
Wavelet
Trees
today
Huffman
×
MTF
×
BWT
×
×
bzip2
today
FM-indexes

Rank on Binary Strings

Definition

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

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

Succinct Solution

We break B into blocks of length $\log^2 n$. We store $\mathrm{rank}(i)$ 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. Remember we discussed that as a way to combine Huffman coding with the BWT?

Compressed Solution (Sketch)

Theorem

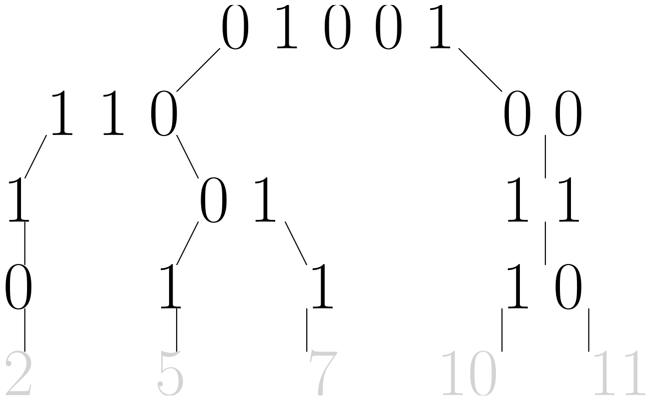
We can store a binary string $B[1..n]$ in

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

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

We don't really have time to discuss H_0 in this course — but don't worry, there's *lots* about it in the data compression course.

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. We don't have enough time to discuss H_k in detail, either, but it's good!

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
1	0	0	1

3	2	3	2	3	3
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]

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

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

FM-Indexes

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

The partial sums of the frequencies — 1 \$, 3 -'s, 8 As, 2 Bs, 1 D, 3 Ls, 2 Rs — are C = 0, 1, 4, 12, 14, 15, 18

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

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

Counting

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

$C = 0, 1, 4, 12, 14, 15, 18$

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

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

How many occurrences of BAR are there? Of LA?

Counting

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

$C = 0, 1, 4, 12, 14, 15, 18$

0 1 0 1 1 0 1 1 0 0 0 0 0 0 1 0 0 0 0 0

1 1 0 0 0 1 1 1 1 0 1 1 1 1   1 0 0 0 0 1

1 0 1 1   0 0 1 1 0 0 0 0 0 0   0 1 1 1   0 0

How many occurrences of BAR are there? Of LA?

Locating

We store a binary string indicating the position in the BWT of every $(\log^{1+\epsilon} n)$th character in S; we also store each of those characters' positions, in the order they appear in the BWT. This takes a total of $\mathcal{O}\left(\frac{n}{\log^{1+\epsilon} n} \cdot \log n\right) = o(n)$ bits.

Given the position of a character in the BWT, we use rank to walk backward until we reach a character whose position in S we have sampled. We then add to that sampled position the number of backward steps we have taken. This takes a total of $\mathcal{O}\big(\log^{1+\epsilon} n \cdot \log \sigma\big)$ time.

End Result

Theorem

We can store a string $S[1..n]$ over an alphabet of size σ 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 $\log \sigma$ to $\log \log \sigma$ in the first bound and get rid of it in the second bound. But not in this course.