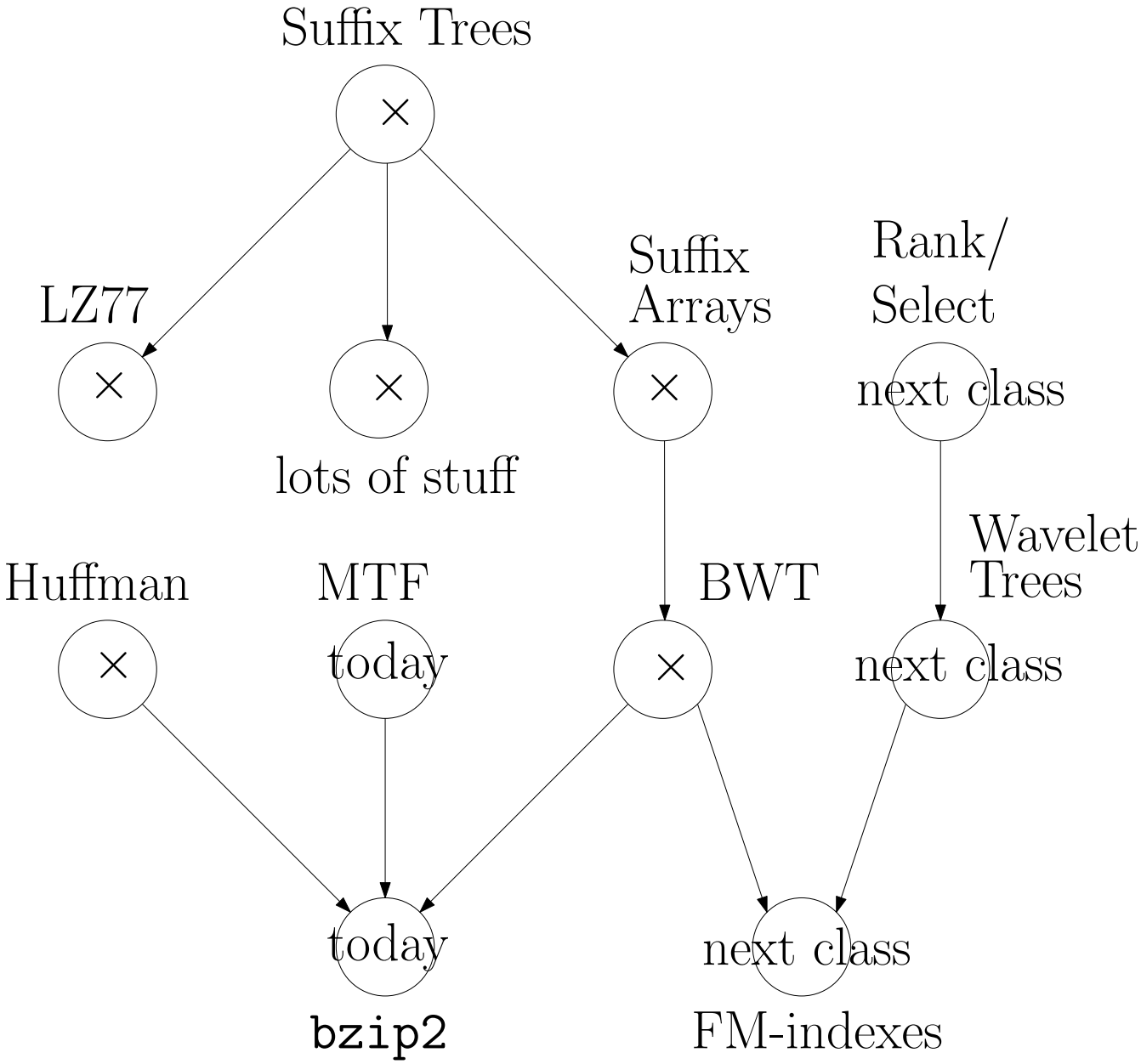
Lecture 5:
BWT-Based Compression

March 12th, 2012

Plan


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

Oops!

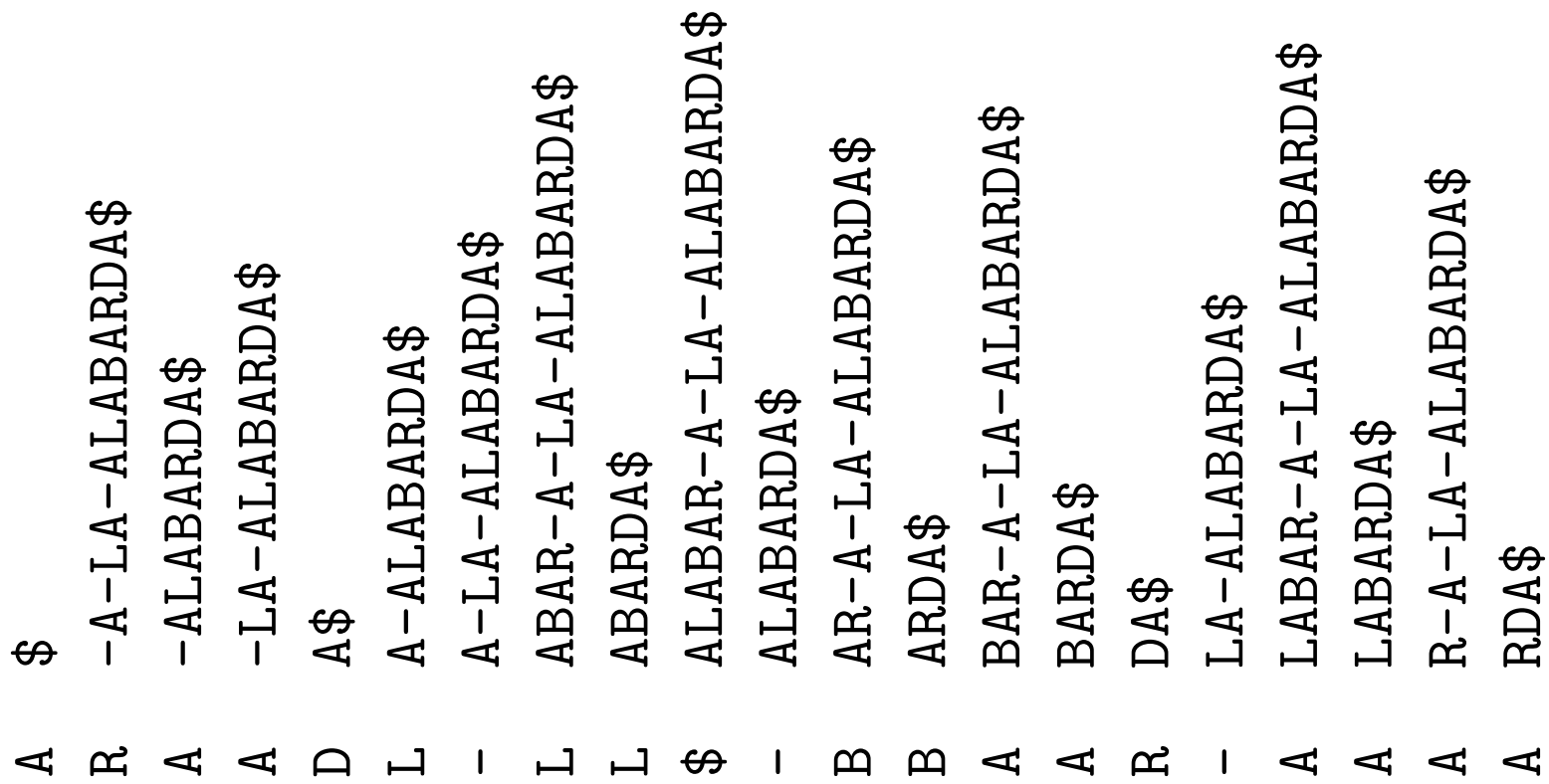
I messed up the homework: 4.5 is the same as part of 4.4, so you can skip it. Sorry!

If you're feeling enthusiastic, you can try 4.7 (optional!):

> *Huffman code MTF(BWT(MISSISSIPPI$)). Compare with what you got for 4.6.*

Burrows-Wheeler Transform (BWT)

Recall that BWT[i] = S[(SA[i] - 1) mod (n + 1)] [1]



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

Local and Global Homogeneity

The BWT moves together characters that have similar contexts. That is, if characters following occurrences of the same context tend to be the same, then the BWT turns *contextual structure* into *local homogeneity*.

However, since the BWT only permutes the characters, it doesn't change the efficacy of Huffman coding, which takes advantage only of *global homogeneity*.

How can we turn *local homogeneity* into *global homogeneity*?

Move-To-Front (MTF)

MTF is described in your exercise:

> *For Move-To-Front (MTF) compression, we are given a string $S[1..n]$ and a list of the characters in the alphabet in some order. For i from 1 to n, we replace the character $S[i]$ by its current position in the list and then move it to the front of the list. For example, with an initial list D-E-H-L-O-R-W, MTF turns HELLOWORLD into 3-3-4-1-5-7-2-7-4-7 (I think). We then encode the resulting list of integers using, say, Huffman coding.*

Example

Let's look at *Hamlet* from Project Gutenberg, BWT(*Hamlet*), MTF(BWT(*Hamlet*)) and a simple prefix-free encoding of the integers in MTF(BWT(*Hamlet*)) that uses about $2 \log x$ bits to write each integer x.[2]

Our final encoding takes up about half a megabyte — three times as much as the original file *Hamlet* — but that's only because it's ASCII instead of binary. As a binary file, it would take up about 62 kilobytes, just a little more than the `7-Zip` encoding.

[2] These files are available on Noppa.

PAUSE TO LOOK AT FILES

bzip2

If you check Sourceforge (or Wikipedia) you'll find that `bzip2` is essentially BWT + MTF + Huffman, with some run-length encoding (RLE) thrown in. For RLE, we replace (sufficiently long) runs of duplicate characters by 1 copy of the character and the length of the run.[3]

To decompress a file, we decode it with Huffman coding, then invert MTF, then invert the BWT.

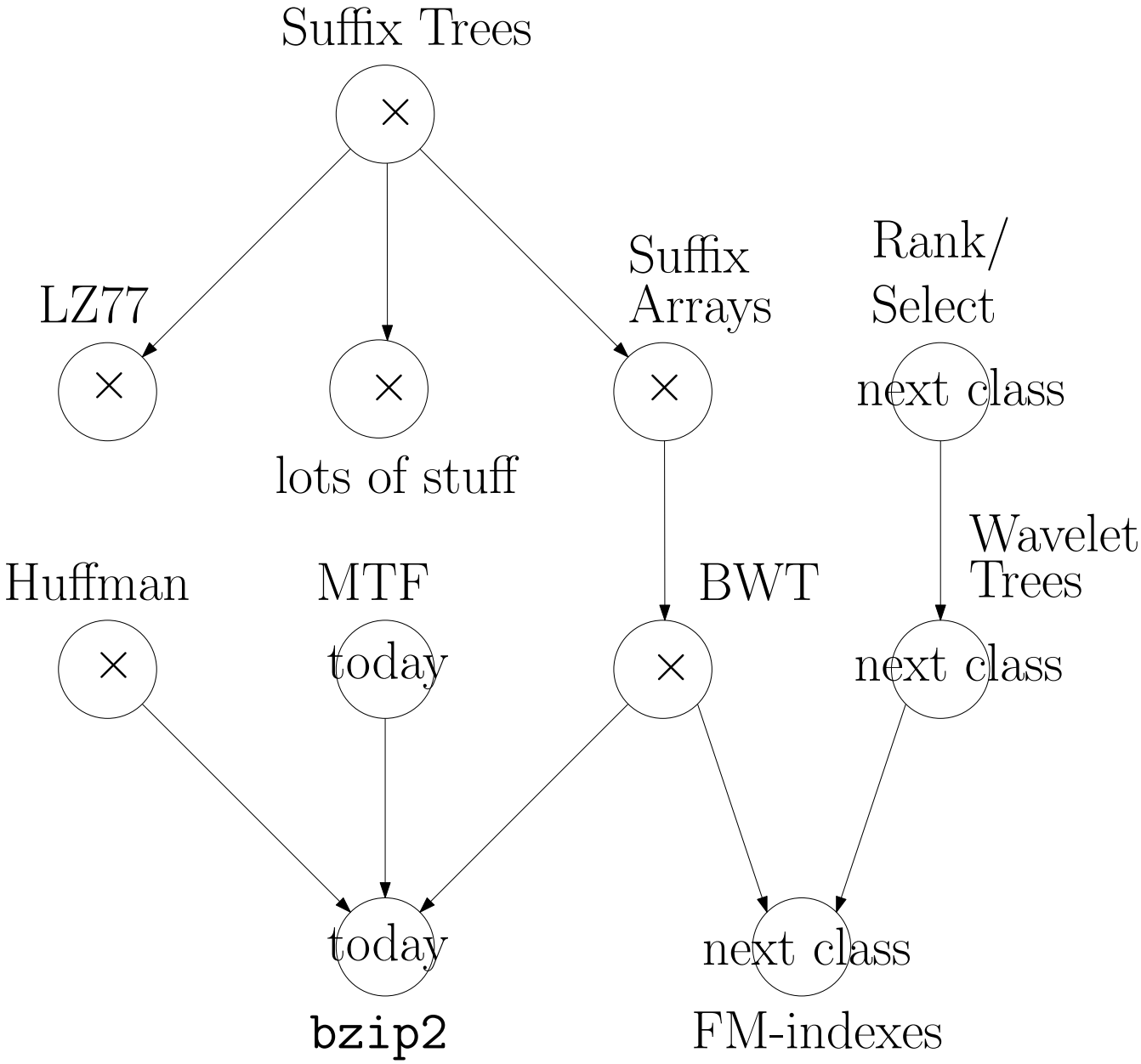
So, you now know how to build one of the most popular compressors! Well, pretty much...

[3] I didn't bother with Huffman or RLE.

Next Time

I mentioned before how the BWT can be used as an index. Next time we'll see how to build a kind of *compressed full-text index* called an *FM-index*. We'll use *compressed rank/select* data structures to build a *wavelet tree*, which is a data structure that's almost as versatile as a suffix tree — and much smaller.

Plan


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