582487 Data Compression Techniques (Spring 2015)

Extra: Wavelet tree – how does it actually work?

****NOTE: This is 100% optional material. You don't need to know it for exam. But it will (hopefully) help understand why the rank operations for backtracking work the way they do.****

In binary space (0s and 1s), we can't directly mark, for example, where are B's in BWT ARD$RCAAAABB, as there are too many "choices" to fit in binary. But, we can partition the alphabet into two sets $\{\$, A, B, C\}$ and $\{D, R\}$. Then for each character in the BWT, we mark the position by 0 if it belongs to first set (going left in the tree), and by 1 if it belongs to second set (going right in the tree).

Then we can again partition the sets, so that $\{\$, A, B, C\} \rightarrow \{\$, A\}$ and $\{B, C\}$. And again mark 0 if the character in BWT belongs to first set, and 1 if it belongs to second set. One more partition will get us single characters, and we have reached our goal!

So to use the BWT ABD$RCAAAABB and the wavelet tree in the exercise sheet as an example, lets say we choose the starting range $[1, 10]$[1] (1-based) and are looking for A.

So we start at root, node v_1, which has the bitvector of 011010000000. We know that A is found in set $\{\$, A, B, C\}$, so we ask that how many occurrences of characters in that set are there, which is $v_1.rank_0(10) = 7$ (because the set is the first set of the partition, we're counting 0s, i.e. going left in the tree).

But we're not done yet! This doesn't mean that there are 7 As, just that there are 7 characters that are either $, A, B or C. So we move to next node, v_2 and get the bitvector 001000011. Please note that the size of this bitvector is 9, the number of 0s in the bitvector of the parent node. So this bitvector is answering the question of "out of these occurrences of either $, A, B or C, which belong to set $\{\$, A\}$ and which to set $\{B, C\}$?".

As we know from the last query that there are 7 occurrences that are either $, A, B or C in our range, we are interested how many characters belonging to set $\{\$, A\}$ are there in the first 7 out of 9 positions, that is $v_2.rank_0(7) = 6$ (it's the first set, so we're counting 0s).

Now we descend to the node for subset $\{\$, A\}$ and examine its bitvector 101111. Please note again that the size of this bitvector is the number of 0s in the parent node's bitvector, so the question is again "out of these occurrences of either $ or A, which are $ and which are A?".

This time A is in the second set, so we are counting 1s. From the previous node we know that we are interested of the first 6 occurrences that are either $ or A, so $v_4.rank_1(6) = 5$. So we got to the conclusion that there are 5 occurrences of A in this range.

I hope that clarified that what are we actually asking from the wavelet tree when we are doing all of those rank queries!

****Even more optional section****

In the exercise session a question was brought up that couldn't we construct a different-shaped wavelet tree for the BWT. Yes, we could. There are many ways to partition the alphabet, which lead to different shapes of the trees. Like with all the codes we have encountered in this course, you have to let the decoder know the rules, so that they can build the same tree. Or you could save the sets for every node explicitly, but that could end up using a lot of space.

The shape of the tree affects the look-up times (=number of rank operations to do). There are for example Huffman-shaped wavelet trees. But that's beyond the scope of this course.

[1] Please note that this is just an example of the rank queries, it does not necessarily correspond to any pattern backtracking

For the sake of simplicity, lets stick with *complete binary trees*[2] like in the slides and exercise sheet. In a complete binary tree, every level, except maybe the last, is completely filled. This has the advantage that we can concatenate the bitvectors for each level and store those (see lecture 9, slide 14, how node for 6 has ”dummy” bitvector ”00” to fill the level). For the trees in the exercise sheet we can imagine similar dummies.

[2]I mistakenly called these balanced binary trees in the exercise, apologies. Complete binary tree is balanced, but balanced binary tree is not necessarily complete.