Burrows Wheeler Transform and FM-index

25 October 2023

For simplicity, in the following exercises you can consider the alphabet $\Sigma = \{\$, A, C, G, N, T\}$.

Suffix array

For the following exercises we will use an existing implementation, available in the file `karkkainen_sanders.py`

Example:

```
import karkkainen_sanders as ks

S = "GGCGGCACCGC$"
SA = ks.simple_kark_sort(S)
print("i\tSA\tSuffix")
for i in range(len(S)):
    print(f"{i}\t{SA[i]}\t{S[SA[i]:]}")
```

Burrows-Wheeler Transform

Q 1. Write (and test) a function `build_bwt(S,SA)` which returns the Burrows-Wheeler Transform `BWT` of a string `S` for which the suffix array `SA` has already been computed.

Let `F` be the first column of the Burrows-Wheeler matrix. In the previous example `F` = `$ACCCCCGGGGG`. More formally, `F[i]` = `S[SA[i]]`.

Q 2. Implement the function `get_count(bwt)` which returns a dictionary `count` such that `count[a]` is the number of characters lexicographically smaller than `a` occurring in the BWT `bwt`. This value also corresponds to the index of the row in which appears the first suffix starting with `a`. In the previous example, `count[G]` = 7.

Q 3. Implement the function `get_rank(bwt)` which outputs a vector `rank` such that `rank[i]` equals to the rank of the i-th character of `bwt`. For example, `rank[3]` = 1 as `bwt[3]` is the second "`G`" appearing in `bwt`.

Q 4. Implement the function `bwt2seq(bwt,count,rank)` that returns (in linear time) the original sequence S from which the Burrows-Wheeler transform `bwt` has been computed.

Q 5. Write a function `test_bwt2seq(S)` that tests the construction and decoding of the BWT on several examples such as an empty sequence, the *E. coli* genome, sequences from previous exercises, etc..

FM-index

Q 6. Implement a function `get_occ(bwt)` that returns a dictionary `occ`, where `occ[a]` is a vector such that `occ[a][i]` is the number of occurrences of `a` in `bwt` up to (and including) index i.

Q 7. Implement the function `contains_pattern(p, bwt, count, occ)` that outputs `True` if `p` is found in the text encoded by the BWT `bwt`, and `False` otherwise.

Q 8. Implement the function `find_pattern(p, bwt, count, occ, sa)` that output the list of occurrences (positions) of `p` in the text econded by the BWT `bwt`.

Q 9. Implement two functions to test `contains_pattern` and `find_pattern` as previously done for `bwt2seq`.

Bonus exercises

Q 10. Propose a sub-sampling of the table `occ` every `step` rows.

- Modify your implementation accordingly (possibly in a new file)
- Validate your modifications with the help of the test functions you already wrote
- Estimate the gain in terms of used memory and the impact on execution time

Q 11. Modify `contains_pattern` and `find_pattern` (and all other relevant functions) to accept an additional parameter k which represents the maximum number of errors allowed in the pattern during the search. Test the functions with small values of k (up to 3 for example).