

Burrows-Wheeler Transform

Antonio Osamu Katagiri Tanaka - A01212611@itesm.mx

Langmead, B. (2013). Introduction to the Burrows-Wheeler Transform and FM Index. Retrieved from http://www.cs.jhu.edu/~langmea/resources/bwt_fm.pdf

Kingsford, C. (2009). Burrows-Wheeler Transform. Carnegie Mellon University. Retrieved from <https://www.cs.cmu.edu/~ckingsf/bioinfo-lectures/bwt.pdf>

```
# Clear all objects (from the workspace)
rm(list = ls())
```

```
# Suppress Warning messages
options(warn = -1)
```

```
# Integrate python to R
library(reticulate)
use_virtualenv("r-reticulate")
py_available(TRUE)
```

```
## [1] TRUE
```

```
# LIBRARIES
from collections import Counter as cnter;
```

```
# CONSTANTS
EOS = "$";
```

Calculate counts for the letters in the sequence.

```
def make_cnt(s, printTbl, alphabet=None):
    # Create alphabet (aka. used characters) if none is provided
    if alphabet is None:
        alphabet = set(s);

    # Count each character repetitions
    c = cnter(s);

    # Sort the alphabet and
    # Get the position where the next character starts
    total = 0;
    result = {};
    for letter in sorted(alphabet):
        result[letter] = total;
        total += c[letter];

    # Print the result
    if printTbl: print("Character score:\n", result, "\n");
    return result;
```

```
make_cnt('banana', printTbl=False);
```

```
## {'a': 0, 'b': 3, 'n': 4}
```

Return the suffix array of the sequence

```
def make_suffixArray(s, printTbl):
    # Iterate s and generate all suffixes
    suffixes = {};
    for i in range(len(s)):
        suffixes.update({s[i:] : i});

    # Sort suffixes by name
    lst_sort = sorted(suffixes.keys());

    # Iterate suffix and get their index
    lst = list(suffixes[suffix] for suffix in lst_sort);

    # Print the list of sorted suffixes
    if printTbl: print("Suffixes:\n", lst_sort, "\n");

    # Return the list of indexes
    return lst;
```

```
make_suffixArray('banana' + EOS, printTbl=True);
```

```
## Suffixes:
## ['$ ', 'a$', 'ana$', 'anana$', 'banana$', 'na$', 'nana$']
##
## [6, 5, 3, 1, 0, 4, 2]
```

Computes the Burrows-Wheeler transform from a suffix array.

```
def make_bwt(s, printTbl, suffixArray=None):
    # Create an suffix array if none is provided
    if suffixArray is None:
        suffixArray = make_suffixArray(s, printTbl);

    # Compute the Burrows-Wheeler Transform
    bwt = '';
    for idx in suffixArray:
        bwt = bwt + s[idx - 1]; # -1 as the last character is EOS

    # Print the the Burrows-Wheeler Transform
    if printTbl:
        print("iBWT:\n", s, "\n");
        print("BWT:\n", bwt, "\n");
    return bwt;
```

```
make_bwt('banana' + EOS, printTbl=False);
```

```
## 'annb$aa'
```

Returns occurrence of the letters in the Burrows-Wheeler transform.

```
def make_occ(bwt, printTbl, letters=None):
    # Create a list with the used characters
    if letters is None:
```

```

        letters = set(bwt);

        # Create a dict including each character with counts eq to zero
        result = {};
        for letter in letters:
            result.update({letter : [0]});

        # Initialize the counter with 1 for the 1st character
        result[bwt[0]] = [1];
        # for each charter in the BWT,
        for letter in bwt[1:]:
            # Add a column to the right, counting the appearances of each character
            for k, v in result.items():
                v.append(v[-1] + (k == letter));

        # Print the occurrences of each character
        if printTbl: print("BWT Occurrence:\n", result, "\n");
        return result;

make_occ('annb' + EOS + 'aa', printTbl=False);

```

```
## {'n': [0, 1, 2, 2, 2, 2, 2], 'a': [1, 1, 1, 1, 1, 2, 3], '$': [0, 0, 0, 0, 1, 1, 1], 'b': [0, 0, 0, 0, 0, 0, 0]}
```

Returns the information tables required to find matches within BWT.

```

def make_all(sequence, printTbl, suffixArray=None, eos=EOS):
    # Create a list with the used characters
    alphabet = set(sequence);

    # Ensure EOS is not a character within the sequence
    assert eos not in alphabet;

    # Get the count of the characters in the sequence
    cnt = make_cnt(sequence, printTbl, alphabet);

    # Concatenate EOS to the sequence
    sequence = sequence + eos;

    # Create an suffix array if none is provided
    if suffixArray is None:
        suffixArray = make_suffixArray(sequence, printTbl);

    # Compute the Burrows-Wheeler Transform
    bwt = make_bwt(sequence, printTbl, suffixArray);

    # Get the occurrences of each character
    letters = alphabet | set([eos])
    occ = make_occ(bwt, printTbl, letters);

    # Make sure the indexes never exceed the sequence limits
    for k, v in occ.items():
        v.extend([v[-1], 0]);

```

```

# Print/Return all tables
if printTbl: print("alphabet:\n", alphabet, "\n");
return alphabet, bwt, occ, cnt, suffixArray;

```

Update the “begin/end” range of a letter within the sorted BWT

```

def update_range(begin, end, letter, occ, cnt, length):
    # Set the new left pointer
    newbegin = cnt[letter] + occ[letter][begin - 1] + 1;

    # Set the new left pointer
    newend = cnt[letter] + occ[letter][end];

    # Return the range limits
    return newbegin, newend;

```

Find all matches of the ‘query’ within the ‘sequence’, with at most some amount of mismatches.

```

def find(query, sequence, printTbl=False, mismatches=0, bwt_data=None, suffixArray=None):
    # Ensure the query to search is long enough
    assert len(query) > 0;

    # Save the query string for later use
    query_str = query;

    # Create all the required tables
    if bwt_data is None:
        bwt_data = make_all(sequence, printTbl, suffixArray=suffixArray);
    alphabet, bwt, occ, cnt, suffixArray = bwt_data;

    # Ensure the alphabet contains at least one character
    assert len(alphabet) > 0;

    # If the query contains more characters than the alphabet,
    # then there are not matches
    if not set(query) <= alphabet:
        return [];

    # Define a 'stack' data structure
    class Partial(object):
        def __init__(self, **kwargs):
            self.__dict__.update(kwargs);

    # Create a 'stack' of partial matches
    length = len(bwt);
    results = [];
    partial_matches = [Partial(
        query      = query,
        begin      = 0,
        end        = len(bwt) - 1,
        mismatches = mismatches
    )];

```

```

# Iterate the query (character by character) to find matches
while len(partial_matches) > 0:
    # Read and remove an element of partial_matches
    p = partial_matches.pop();

    # Search for the query (letter by letter)
    query = p.query[:-1];
    last = p.query[-1];

    # If mismatches are allowed, then use the whole alphabet
    # If no mismatches are allowed, only use the current character/letter
    if p.mismatches == 0: letters = [last];
    else:                 letters = alphabet;

    # Check if the current letter is a match
    for letter in letters:
        # Get the range where the current character appears
        begin, end = update_range(p.begin, p.end, letter, occ, cnt, length);

        # Iterate the calculated range/window
        if begin <= end:
            # Stop comparing if the current letter is last character of the query
            # Store the match location in results
            if len(query) == 0:
                results.extend(suffixArray[begin : end + 1]);
            else:
                # Track the number of mismatches
                mismatchesCnt = p.mismatches;

                # Decrement the number of allowed mismatches if one is founded
                if letter != last:
                    mismatchesCnt = max(0, p.mismatches - 1);

                # Update the 'stack'
                partial_matches.append(Partial(
                    query      = query,
                    begin      = begin,
                    end        = end,
                    mismatches = mismatchesCnt
                ));

    # Sort the match locations
    res = sorted(set(results));

    # Print and Return the result
    print("subSequence appears at possitions:\n", res);
    print('', sequence);
    for i in res: print(' '*i, query_str);
    print();
    return res;

```

Let's test the BWT algorithm to find "ana" in "banana"

```
find('ana', 'banana', printTbl=True);
```

```
## Character score:
## {'a': 0, 'b': 3, 'n': 4}
##
## Suffixes:
## ['$ ', 'a$', 'ana$', 'anana$', 'banana$', 'na$', 'nana$']
##
## iBWT:
## banana$
##
## BWT:
## annb$aa
##
## BWT Occurrence:
## {'n': [0, 1, 2, 2, 2, 2, 2], 'a': [1, 1, 1, 1, 1, 2, 3], '$': [0, 0, 0, 0, 1, 1, 1], 'b': [0, 0, 0,
##
## alphabet:
## {'n', 'a', 'b'}
##
## subSequence appears at positions:
## [1, 3]
## banana
##   ana
##     ana
##
## [1, 3]
```

Now an example with the SARS nucleotide (NC_004718.3 SARS coronavirus, partial genome)

```
find(
    'GATCTCTTA',
    'ATATTAGGTTTTTACCTACCCAGGAAAAGCCAACCAACCTCGATCTCTTGTAGATCTGTTCTCTAAGATCTCTTACGAACTTTA',
    mismatches=1
);
```

```
## subSequence appears at positions:
## [41, 66]
## ATATTAGGTTTTTACCTACCCAGGAAAAGCCAACCAACCTCGATCTCTTGTAGATCTGTTCTCTAAGATCTCTTACGAACTTTA
##                                     GATCTCTTA
##                                     GATCTCTTA
##
## [41, 66]
```

```
options(warn = 0)
```