INTRODUCTION TO FOCUS AREAS IN BIOINFORMATICS - WS19/20

Project 12: Mapping of shotgun reads to a reference genome

Raghavendra Tikare and Stanislav Klein

Full list of author information is available at the end of the article

Abstract

Goal of the project: This week's project was centered on mapping of shotgun reads with index-based and non index-based search algorithms in C++ and Python.

Main result(s) of the project: Implementing the same algorithm into two programming languages can result in very different looking way in terms of run time, although the basic principle remains the same.

Personal key learnings: There are easy ways to implement algorithm into a specific programming language, which in return may have disadvantages in other fields in data science.

Estimation of the time: 16 hours

Project evaluation on a scale of 1-5: 3

Word count: 558

Keywords: search algorithm; Boyer-Moore; C++; Python

Implementation

Implementation of non index based search in C++ and Python

In order to search a pattern in the text or a sequence we decided to the Boyer-Moore algorithm from the appendix of the lecture's slides. Searching the internet for ways to implement the algorithm led us to an article by Atum et al. on https://www.geeksforgeeks.org/boyer-moore-algorithm-for-pattern-searching/. We were provided Python code (contributed by Atul Kumar) and its C++ counterpart (contributed by a user named rathbhupendra). This version of the Boyer-Moore algorithm uses the bad character heuristic approach by shifting the pattern upon mismatch to look for a predetermined pattern. The result of the algorithm in Python and C++ can be seen in figures 1 and 2 respectively.

```
78      txt = "ABABBBAASSSBCDABABBBAASSSBCDABC"
79      pat = "ABC"
80      search(txt, pat)
81
82  if __name__ == '__main__':
83      main()
84
```

```
Pattern occur at shift = 28
```

Figure 1 Boyer Moore algorithm in Python using Python Google Colab; pattern occuring at shift 28

```
int main()
{
    string txt= "ABAAABCD";
    string pat = "ABC";
    search(txt, pat);
    return 0;
}
C:\Users\RAGHAVENDRA\Desktop\1\bin\Debug\1.exe
pattern occurs at shift = 4

Process returned 0 (0x0)   execution time : 0.052 s
Press any key to continue.
```

Figure 2 C++ - Boyer Moore algorithm in C++ using the CodeBlocks software; pattern occuring at shift 4

Implementaion of Index based search using BWT and FM Index in C++ and Python

Burrows-Wheeler Transform (BWT) is the way of permuting the characters of a string t into another string. Generally, it has two main approaches indexing and compressing. Here we started with defining the simple terms in Python like rotations, as initaially we need to list the rotations of the string t, and then sorting the string in the lexcicographic order. Then by the way of BWM it tries to match each letters in the string. We defined the ranks which maps the characters to the number of times they appeared. Finally we define the cols function which returns the map from character to the range of rows prefixed by the character. We even tried with the reverse function which works similarly with effective output. A demonstration of the algorithm is shown in the Figure 3.

```
In [41]: firstCol(tots)
Out[41]: {'$': (0, 1), 'a': (1, 5), 'b': (5, 7)}

In [42]: # confirm that the representation of the first column above is sensible
         print('\n'.join(bwm(t)))

$abaaba
a$abaab
aaba$ab
aba$aba
abaaba$
ba$abaa
baaba$a
```

Figure 3 Index based search algorithm using BWT and FM index showing all possible rotations of a string in the Burrows-Wheeler-Transform

C++ implementation using Burrows wheeler transform and FM Index

Following the nanocourse by Reinert et al. to implement the desired algorithm we were left with code which outputs the number of occurances of a pattern inside a predetermined input. For this to work we needed to change the compiler settings in CodeBlocks to support all commands, as this didn't seem to be the case for the default settings on our computers. One of the steps of BWT was to create bitvectors for each character of the string.

Benchmarks

The wall clock time for all different reads within the chromosome 4 of the drosophila genome using four different algorithms is calculated. The index building time for

Python and C++ is nearly 2 minutes and 30 seconds respectively. When implemented with python and C++ non index method shows that only till 1000 and 10000 reads the run time is less than 2 minutes and for the rest it was greater than 10 minutes. By using BWT and FM with the Python and C++ gives the run time less than 100 minutes for 1000000 reads. We can also know that the C++ is the faster programming language to run with as it gives flexibility with time and speed.

Algorithm	Index build time in seconds (s)
Python – BWT & FM	134.269s
C++ - BWT & FM	26.657s

Figure 4 Benchmark results

Algorithm	Runtime in seconds (s) for 100 reads	Runtime in seconds (s) for 1000 reads	Runtime in seconds (s) for 10000 reads	Runtime in seconds (s) for 100000 reads	Runtime in seconds (s) for 500000 reads	Runtime in seconds (s) for 1000000 reads
Python - no index	9.864s	108.957s	> 10 mins	> 10 mins	> 10 mins	> 10 mins
C++ - no index	0.658s	5.668s	69.358s	> 10 mins	> 10 mins	> 10 mins
Python – BWT & FM	125.563s	> 10 mins	> 10 mins	> 10 mins	> 10 mins	> 10 mins
C++ - BWT & FM	0s	0.012s	0.795s	7.256s	39.358s	81.376s

Figure 5 Benchmark results