CMPE 59H - Bioinformatics
Assignment 2

Abdullah Atakan Guney
2018700069

December 25, 2018

1. Description

In this assignment, I have implemented Burrows-Wheeler transform and inverse Burrows-Wheeler transform, to compress and decompress methods used for genome, protein sequences

2. Burrows - Wheeler Transform

The Burrows–Wheeler transform (BWT, also called block-sorting compression) rearranges a character string into runs of similar characters. This is useful for compression, since it tends to be easy to compress a string that has runs of repeated characters by techniques such as move-to-front transform and run-length encoding. More importantly, the transformation is reversible, without needing to store any additional data except the position of the first original character. The BWT is thus a "free" method of improving the efficiency of text compression algorithms, costing only some extra computation.

It basically appends '$' to the string and make rotations through the string. For example, let the input string is "panamabananas", the list I am considering is following:

- panamabananas$
- $panamabananas
- s$panamabanana
- as$panamabanan
- nas$panamabana
- anas$panamaban
- nanas$panamaba
- ananas$panamab
- bananas$panama
- abananas$panam
- mabananas$pana
- amabananas$pan
- namabananas$pa
- anamabananas$p

Then sort this list:

- $panamabananas

- abananas$panam
- amabananas$pan
- anamabananas$p
- ananas$panamab
- anas$panamaban
- as$panamabanan
- bananas$panama
- mabananas$pana
- namabananas$pa
- nanas$panamaba
- nas$panamabana
- panamabananas$
- s$panamabanana

The last column is Burrows-Wheeler transform: smnpbnnaaaaa$a.

3. Inverse Burrows-Wheeler Transform

To decompress, compressed sequence with BW transform, we should apply inverse transform, thank to BW-transform is bijection for input strings(we consider same letter as with different subscripts). I have done inverse transform as follows:

- I converted given string to a (character, index) list.
    1. ('s', 0)
    2. ('m', 1)
    3. ('n', 2)
    4. ('p', 3)
    5. ('b', 4)
    6. ('n', 5)
    7. ('n', 6)
    8. ('a', 7)
    9. ('a', 8)
    10. ('a', 9)

11. ('a', 10)
12. ('a', 11)
13. ('$', 12)
14. ('a', 13)

- Sort this character list:

1. ('$', 12)
2. ('a', 7)
3. ('a', 8)
4. ('a', 9)
5. ('a', 10)
6. ('a', 11)
7. ('a', 13)
8. ('b', 4)
9. ('m', 1)
10. ('n', 2)
11. ('n', 5)
12. ('n', 6)
13. ('p', 3)
14. ('s', 0)

- After constructing these 2 lists, I constructed a dictionary that maps sorted lists items to unsorted one accordingly.

1. ('$', 12) : ('s', 0)
2. ('a', 7) : ('m', 1)
3. ('a', 8) : ('n', 2)
4. ('a', 9) : ('p', 3)
5. ('a', 10) : ('b', 4)
6. ('a', 11) : ('n', 5)
7. ('a', 13) : ('n', 6)
8. ('b', 4) : ('a', 7)
9. ('m', 1) : ('a', 8)
10. ('n', 2) : ('a', 9)
11. ('n', 5) : ('a', 10)
12. ('n', 6) : ('a', 11)

13. ('p', 3) : ('$', 12)
14. ('s', 0) : ('a', 13)

- Beginning current character as first element in the sorted list, I constructed original string in reverse order as follows, I looked up next character by using the dictionary that I have constructed before, by looking what is next character after current character.

1. ('$', 12)
2. ('s', 0)
3. ('a', 13)
4. ('n', 6)
5. ('a', 11)
6. ('n', 5)
7. ('a', 10)
8. ('b', 4)
9. ('a', 7)
10. ('m', 1)
11. ('a', 8)
12. ('n', 2)
13. ('a', 9)
14. ('p', 3)

- After constructing original string in reversed order, I just reversed the string

1. ('p', 3)
2. ('a', 9)
3. ('n', 2)
4. ('a', 8)
5. ('m', 1)
6. ('a', 7)
7. ('b', 4)
8. ('a', 10)
9. ('n', 5)
10. ('a', 11)
11. ('n', 6)
12. ('a', 13)
13. ('s', 0)
14. ('$', 12)

4. Screenshots

Here is the screenshot of my program that you can compute both BW-transform and inverse BW-Transform

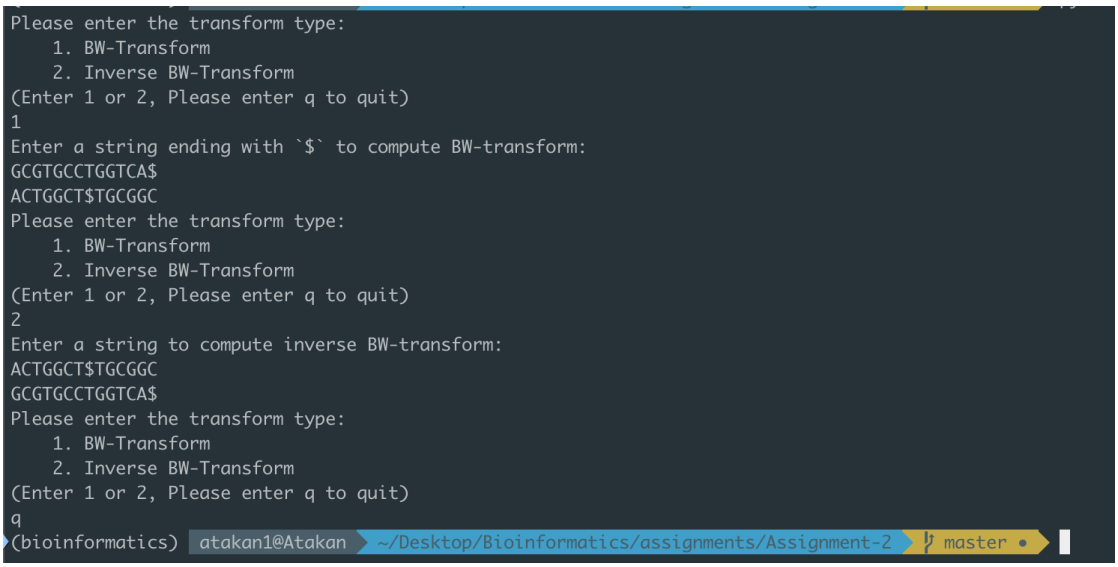

Figure 1: My Program