Saarland University
Jun.-Prof. Dr. Tobias Marschall
Dr. Marcel Schulz

UNIVERSITÄT DES SAARLANDES

4.Assignment - Algorithms for Sequence Analysis, SS 2018

Exercise 1: De Bruijn Graphs (3 Theory)

The De Bruijn graph (DBG) is a commonly used data structure in many *de novo* assemblers. In this task you have to build a DBG from the read set given below. The task is divided into three subtasks:

(a) Draw a DBG from the set of reads given with this sheet using a k-mer size of 4. You should not consider the reverse complement of the sequences.

(b) The set of reads given with the task contains erroneous reads. As a result, the DBG of subtask (a) should have *bubbles* and *tips*. Your task is to remove bubbles (having an edit distance of at most one) and tips in the graph, both by preferring the path with higher coverage. Note that tips have a length of $< 2k$. Draw the simplified version of the graph after error removal and collapsing of linear stretches.

(c) Based on the above two graphs, identify the read(s) which are erroneous and mark the erroneous base(s). Also justify the reason behind your choice.

List of reads for task 1

```
>1
CGACTAACATA
>2
GATCGACTATC
>3
CGACTATCATA
>4
TCGACTATCAT
>5
GCGATCGACAA
>6
GATCGACTGTA
>7
GATCGACTGTA
>8
GCGATCGACTG
>9
GATCGACTGTA
>10
TCGACTATCAT
>11
GATCGACAAAG
>12
GCGATCGACTA
>13
GCGATCGACTA
>14
GATCGACAAAG
```

Exercise : Assembly (15 Programming)
In the lecture, we introduced de novo assembly. In this task, you have to implement a tool for de novo assembly using De Brujin graphs. Beyond that, you have to implement read error correction method to improve the quality of your assembly. On our website, you find three FASTA[1] files containing DNA-Reads for testing your tool. The first two fasta files *reads_simple.fa* and *reads_simple_error.fa* are generated from short sequences of length 110 bps. And *reads_complex.fa* is a real dataset sequenced from Escherichia Coli. The reference sequence of Escherichia Coli is also provided with this assignment. You can use the reference to assess the quality of your assembly generated from *reads_complex.fa* dataset. Of course, you are not allowed to use the reference in your assembly algorithm. Using QUAST[2] , you can learn about the quality of your assembled sequence(s) compared to the reference. For testing, we will use a read dataset and genome similar in size to the one available on our website.

Your program should be able to do the following four sub-tasks:

(a) Given a k-mer and a k-mer size, your program should give its incoming and outgoing k-mer(s) from the original input dataset as output. Note that by original input dataset, we mean input dataset which has not been error corrected. As an example, if the input dataset contains one read say AACGTACGTAGGTACGTCG, the input k-mer size is 5 and the input k-mer is CGTAC then your program should output:

`Incoming` k`-mers - ACGTA`
`Outgoing` k`-mers - GTACG`

If there are no incoming or outgoing k-mers then your program should give an output accordingly. For testing this subtask we would be using the same k-mer size which will be used for assembly.

(b) Your program should be able to assemble a simple linear stretch of sequence of around 110bps. Given a k-mer size, your program should consider all k-mers and also their reverse complements to generate the *de* Bruijn graph. For testing this subtask, you can use the fasta file *read_simple.fa* available on our website and a k-mer size of 21. This dataset does not contain any errors. The program should be able to output the following stretch of sequences:

```
>1
GCATTATCAAAGATAACGCCAGCCAGTCGATTGAGCATCAGGCGGAGGCGGTGAAAGCCGTCGCCGAT
ACGGTGCTGGAAATGCTGGCTTCCGATTACGACATTGTGC
>2
GCACAATGTCGTAATCGGAAGCCAGCATTTCCAGCACCGTATCGGCGACGGCTTTCACCGCCTCCGCC
TGATGCTCAATCGACTGGCTGGCGTTATCTTTGATAATGC.
```

where the second sequence is the reverse complement of the first sequence. Note that we will be using a different input dataset obtained from a similar stretch of sequence (approx 100bps) to test your code.

(c) Error correction is one of the important sub-tasks of your program. To test this sub-task, you can use the fasta file *reads_simple_error.fa* available on our website. This dataset is similar to *reads_simple.fa* but contains mismatch errors compared to

the original reference sequence. Given *reads_simple_error.fa* as input and k-mer size of 21, the program should be able to output the following stretch of sequences:

```
>1
GCATTATCAAAGATAACGCCAGCCAGTCGATTGAGCATCAGGCGGAGGCGGTGAAAGCCGTCGCCGAT
ACGGTGCTGGAAATGCTGGCTTCCGATTACGACATTGTGC
>2
GCACAATGTCGTAATCGGAAGCCAGCATTTCCAGCACCGTATCGGCGACGGCTTTCACCGCCTCCGCC
TGATGCTCAATCGACTGGCTGGCGTTATCTTTGATAATGC
```

(d) One of the measures for the evaluation of an assembler is its ability to assemble large genomes. Given an input dataset, your assembler should be able to error correct and provide sequences corresponding to all linear paths in the graph. To test this task, you can use *read_complex.fa* available on our website.

Submission

- Your submission should have an executable named *assemble*. Given an input dataset and an input k-mer, your program should first display the output of subtask (a) on the screen. After this step, you can error correct the dataset (you can do this either by simplifying the graph or just using the k-mer abundance information). Assembly of the dataset should be the final step of your program and the assembled sequence should be written in a fasta file. The parameters of the executable should be as follows

  `# ./assemble <Input>.fasta <Output>.fasta size` k`-mer`

  $<Input>.fasta$ is the input fasta file containing the reads. $<Output>.fasta$ is the output fasta file which should contain the assembled sequence, $size$ is the size of the k-mer used to build the graph. $k\text{-}mer$ is the query k-mer for subtask (a) for which you have to obtain the incoming and outgoing k-mers . Note that the order of the parameters should remain the same.

- We will evaluate your tool by the quality of the assembled sequence it produces. To this end, we compare your assembly with another one computed by us. We will not evaluate your program with respect to memory consumption. However, we expect your tool to be executable on a machine with 8GB of main memory. Your tool should be able to assemble $reads\_complex.fa$ in less than 30 minutes.

---

[1]See http://www.bioinformatics.nl/tools/crab_fasta.html for format description.
[2]See http://quast.sourceforge.net/quast for details.

Remarks:

- There are 18 points to be earned on this assignment sheet.
- 50% of programing points (18/36) and 50% of theoretical exercises (18/36) are necessary to take the exam.

- You are allowed to work in groups of two.
- Hand in your solutions on paper (except for source code) by putting it in the letter box of Tobias Marschall in E2.1 (ground floor).
- Programming code is to be sent as a *tar.gz* package by mail to:
  - aryan.3264@gmail.com
- Source code is only considered if
  - it is in one of the languages Python, C, C++, or Java,
  - it is reasonably documented, commented, and readable,
  - command-lines for compiling and calling are provided,
  - compilation does not fail, and
  - executables are named according to:

    `lastname1_lastname2_assignment1_exercise3`
- Do not forget to mention your names and matriculation numbers on your solutions!
- Copying between groups will result in zero points for all involved groups!

Hand in date: Thursday, June 28, before 12:00