Bilkent University
CS481 | cs583 - Bioinformatics Algorithms

Spring 2025
Deadline: **28 Feb 2025**

Homework Assignment #1

GENERAL INSTRUCTIONS

FAILURE TO FULFILL ANY OF THE FOLLOWING ITEMS WILL RESULT IN A GRADE SCORE OF 0 (zero) WITHOUT ANY CHANCE OF REDEMPTION.

- You must **write your code yourself**. Sufficient evidence of plagiarism will be treated the same as for plagiarism or cheating.
- **C / C++, Python or Java** will be used as programming language. STL is allowed. The assignments are created with C / C++ in mind, so using other languages would required minimum tweaking.
- Your code must **compile**.
- Your code must **be complete**.
- Your code must **run on dijkstra.cs.bilkent.edu.tr** server.
- Your code must make use of **argument parsing**.
  Refer to: https://docs.google.com/presentation/d/1rU0DhBg6yVXbfEtNuK73pjc1yWSPG9OYnGNH-b-kklQ
- Submit your answers **ONLY** through the Moodle page.
- **Zip** your files and send them in only one zipped file.
- File name format `surname_name_hw#.zip`.
- The zip file must contain the following items:
  - All the source files.
  - `Makefile` to compile the source code and produce the binary. Even if you use Python, include this Makefile.
  - A `README.txt` file that briefly describes how your program works.
- All submissions must be made **strictly before the stipulated deadline**.
- A **bonus** will be given for the fastest code that solves the assignment successfully.
- Your program (ideally) should be able to handle very large inputs while consuming low memory and completing tasks within a **reasonable** time. (Basically, avoid writing code that takes hours or days to complete simple tasks.)

1) MULTIPLE PATTERN MATCHING

Aim: In this assignment, you will get familiar with the data structures associated with string searches in bioinformatics. Given a set of patterns and texts, find all occurrences of any of the patterns in texts.

1.1 Data

- **Input**
  1. A FASTA file containing reference sequences to be searched. Each sequence in the file includes a header line (for the name) followed by multiple lines of sequence data. Note that each reference sequence must be concatenated while reading from the file. The path to this file will be provided as a command-line argument using the `-r` switch.
  2. A FASTA file containing patterns to search. The file consists of a header and one pattern per two lines. The path to this file will be provided as a command-line argument using the `-p` switch.
- **Output**
  1. A text file containing the positions $1 < i < m$ where a substring of t, starting at i, matches p_j for $1 < j < k$. The prefix for this file will be provided as a command-line argument using the `-o` switch. You will create a file named $< prefix > .txt$ and save the output there. The file will contain one line per pattern, followed by the indices of the matches in the reference. If no match is found, no index will be recorded. If multiple matches are found, the indices should be printed in a comma-separated format.
  2. A text file, created as $< prefix > .dot$, containing the definition of your suffix tree in DOT[1] language for visualization. This operation should only be performed if the `-d` switch is provided. Note that this is an optional argument. You should not print the tree to a file if the switch is not provided.

As in many other fields, bioinformatics relies heavily on specific file formats. One such format is FASTA (or FA), which represents a string along with its corresponding ID/header. The sequence can be stored in either a single-line or multi-line format. The format is as follows:

```
>sequence1
ACACTTGCCATCAATACTTTTTCGATGTTAGTGGGCA
CACATTGTCTTCCCTACAAATCCATTTCTTGTCCAGC
GAACTCCTCCGCTGTGGGCCATGAGTGGAGGCTTTAC
GGAACCACTTTGGGACTGGAGGAAGTTCACCT
>sequence2
ATTTAATTGCATACTACCTGAGTGGATTGTGGTTATA
GAGACAGGAAAATATTCACAGGTTTTGAAACCAGAAG
ATTGTACACACAGCACTGCATGGGCGGCGGGAGGCTG
GATCCTACTCTCAGGAGGCTCGTATGCCATATGCCAT
TAAAGGGGGATGTTTTA
```

Alternatively, if a single-line format is preferred for storing sequences:

```
>sequence1
AGTGGGCACACATT
>sequence2
GCACTGCATGGGCGGCGGGAGGCTGGGCCCCTACCAGCTCTAAGGACAATCACCATTG
>sequence3
TTTAATTATCTCTGAAATTTAAA
```

1.2 Data Structures

Implement the data structure based on the information given in class and other resources you might find, but be sure to document them in your README file. Note that you need to implement reading input operations on your own. No library can be used (like SeqIO).

- Suffix Tree

[1] https://graphviz.org/documentation/

1.3 Tasks

Using a data structure, perform pattern matching and write the results to the output files.

1.4 Assumptions

- The reference and patterns will be provided in FASTA files.
- The reference file will use a multi-line structure with a fixed length (60), while each pattern in the file will be stored on a single line (regardless of the string's length). Both input files are valid.
- There is no limit on length. The length of the reference can vary from hundreds to millions of characters. However, the length of the pattern string will be strictly less than or equal to the size of the reference.
- You may assume that the alphabet is $\Sigma = \{A, G, C, T\}$.
- The reference does not end with a terminator ($), but you will need to append it to the end yourself during the construction of the suffix tree.
- In order to keep it simple, for each sequence, create a separate suffix tree. Store the trees in array/vector/list and perform a query for each one of them. Those who successfully represent all reference sequences in a single suffix tree will receive an extra **25 points**.
- Patterns and their corresponding indices can be reported in any order.
- The order of node and edge descriptions in the DOT file does not matter as long as the file is consistent and correctly formatted.

2) EXAMPLE

The numbers presented in the example are not necessarily correct. They just show how the program should output the information.

```
user@dijkstra$ cat reference.fasta
>chr1
ACACTTGCCATCAATACTTTTTCGATGTTAGTGGGCACACATTGTCTTCCCTACAAATCC
ATTTCTTGTCCAGCGAACTCCTCCGCTGTGGGCCATGAGTGGAGGCTTTACGGAACCACT
TTGGGACTGGAGGAAGTTCACCT
>chr2
ATTTAATTGCATACTACCTGAGTGGATTGTGGTTATAGAGACAGGAAAATATTCACAGGT
TTTGAAACCAGAAGATTGTACACACAGCACTGCATGGGCGGCGGGAGGCTGGATCCTACT
CTCAGGAGGCTCGTATGCCATATGCCATTAAAGGGGGATGTTTTA
```

```
user@dijkstra$ cat patterns.fasta
>p1
GAGTGGA
>p2
AGTGGGCACACATT
>p3
ACTAG
>p4
GTA
```

```
user@dijkstra$ cat output.txt
(p1) - chr1:96, chr2:19
(p2) - chr1:29
(p3) -
(p4) - chr2:77, chr2:132
```

```
user@dijkstra$ cat output.dot
digraph suffix_trie {
node_chr1_0 [label=""];
node_chr1_1 [label=""];
node_chr1_2 [label="chr1:54"];
node_chr1_3 [label=""];
node_chr1_4 [label="chr1:113"];
node_chr1_5 [label="chr1:75"];
node_chr1_6 [label="chr1:133"];
node_chr1_7 [label=""];
node_chr1_8 [label="chr1:12"];
node_chr1_9 [label="chr1:55"];
```

```
13
14  ...
15
16  node_chr1_0 -> node_chr2_1 [label="AA"];
17  node_chr1_1 -> node_chr1_2 [label=["
        ATCCATTTCTTGTCCAGCGAACTCCTCCGCTGTGGGCCATGAGTGGAGGCTTTACGGAACCACTTTGGGACTGGAGGAAGTTCACCT$"];
18  node_chr1_1 -> node_chr1_3 [label="C"];
19  node_chr1_3 -> node_chr1_4 [label="CACTTTGGGACTGGAGGAAGTTCACCT$"];
20  node_chr1_3 -> node_chr1_5 [label="
        TCCTCCGCTGTGGGCCATGAGTGGAGGCTTTACGGAACCACTTTGGGACTGGAGGAAGTTCACCT$"];
21  node_chr1_1 -> node_chr1_6 [label="GTTCACCT$"];
22  node_chr1_1 -> node_chr1_7 [label="T"];
23  node_chr1_7 -> node_chr1_8 [label="
        ACTTTTTCGATGTTAGTGGGCACACATTGTCTTCCCTACAAATCCATTTCTTGTCCAGCGAACTCCTCCGCTGTGGGCCATGAGTGGAGGCTTTACGGAACCACT
        "];
24  node_chr1_7 -> node_chr1_9 [label="
        CCATTTCTTGTCCAGCGAACTCCTCCGCTGTGGGCCATGAGTGGAGGCTTTACGGAACCACTTTGGGACTGGAGGAAGTTCACCT$"];
25
26  ...
27
28  }
```

After running the **Graphviz** tool (available in Dijkstra), out of the `output.dot` file, a PNG will be generated with your tree. Follow the example to create the layout for the suffix trees you create.

```
1  user@dijkstra$ dot output.dot -Tpng > output.png
```

The `output.png` will contain a tree similar the one shown below ($chr1 = AGTCAGTA\$$ and $chr2 = TAGTAG\$$).