Ling 473 Project 4
Due 11:59pm on Thursday, Sept. 2, 2021

Bioinformatics refers the application of statistics and computer science to the management and analysis of data from the biosciences. In common with computational linguistics, the field shares the task of detecting and searching for patterns in gigantic datasets. For this project you search for DNA sequences in the 24 chromosomes of the complete hg19 GRCh37 human genome. This base pairs are represented in text form, so the problem is essentially a text processing problem.

There are 4,965 target sequences. For each of these you will identify the chromosome file and the offset within the file for all matches. This problem was selected because this search cannot be performed efficiently without special attention to the algorithm that is used. Using `grep` around five thousand times to search 2.8 gigabytes probably won't end well, even in this simplified exercise.

Instead, you will store the search targets in a **prefix trie** and make a single pass through the genome corpus.

Human Genome Corpus

The following directory on patas contains 3 files, corresponding to human chromosomes 20 through 22. Each file describes an ordered sequence of nucleotide bases A, T, C or G.

`/dropbox/20-21/473/project4/hg19-GRCh37/`

These files come from `http://hgdownload.cse.ucsc.edu/goldenPath/hg19/bigZips/hg19.2bit` (this project only use three files in the entire corpus). You may copy them to your home machine for local development, but ***your final code must run on patas***.

> **Masking**
> In accordance with convention for DNA data, some of the nucleotides are represented by the corresponding lower-case letter. This indicates that they are "masked," meaning that they are of low complexity and thus believed to be less important. For this project, **you will do a case-insensitive search**, so masked areas will be included (the target sequences are specified as all upper-case). **Therefore, you may want to capitalize all letters in the 3 files before processing them.**
>
> **Unknown**
> There is no data available for some parts of the genome. **These areas are indicated by the letter N**. None of the target sequences contain N, and your trie does not need to be able to recognize N. In other words, if the current letter is N, just stop searching through the trie and move to the next letter.

Loading the Target Sequences

You will develop a trie structure. Each trie node makes reference to (up to) **four child nodes**, one for each nucleotide (we don't need a node that represent the letter N, because we don't want to recognize it). Any or all of these references can be *null*. When all four children of a node are *null*, the node is a terminal node. Each node (except the root) has exactly one parent, but nodes do not need to store a reference to their parent

node, because navigation through a trie is always one-way, starting from the root. The elegance of a trie is that every node implicitly represents a unique string merely by its position.

Some trie implementations allow each node to store a "payload." One purpose of the payload is to signal whether the node represents the last character in a target string. In this case, the payload would be *null* if the trie node is not a stopping point. This sort of mechanism (for differentiating valid acceptance points) is required if the trie has to store strings that are exact prefixes of other strings that are also stored. For example, {"cat" "catamaran"}. **In this project, none of the target sequences start with any of the others, so we don't need to use a payload**. Instead, we will know if we are at an acceptance point (i.e. that we found a match) if the node has no child nodes (e.g. the leaf node). **Hint: the pseudo code of a node is available at the end of this instruction. You can refer to it if you don't know where to start.**

After defining the node structure, you will instantiate a single node to represent the trie root. Next, you will populate the trie with each target sequence. The target sequences are in the following file, one per line:

`/dropbox/20-21/473/project4/targets`

Add each target sequence to the trie by starting at the trie root and following the sequence's path of nucleotides. This is done by advancing through the target string and navigating/building the trie at the same time. For example, if you come to a child node that is missing, add it by creating a new node. When you reach the end of the target string, you should be at a node with no children—a terminal node that you just created.

If your trie nodes store a payload (not required here), you would store the payload in this node in order to mark the node as a completed match. In such a design, matching can occur at non-terminal nodes.

The target strings can be discarded now, since they are all stored in the trie implicitly (and explicitly as well, if you chose to store each string as a payload).

Searching

Now that you have a trie that represents the strings that you are searching for, you can scan the corpus and find all the matches. **As you read from the corpus, don't forget to upper-case the letters, since we do wish to search within masked data**.

In the searching algorithm, we need three variables, namely, the current index i, the end index j, and a pointer (denoted by p) to the current node.

For each of the current letter (i.e., the i-th letter), the current node pointer p will start at the trie root, and the end index position j will be initialized to the value of the current position i (i.e., set $j = i$).

Now, similar to how the trie was loaded, the current node pointer p will attempt to navigate through the trie, according to the letters read from the corpus, beginning from j. Of course, this time we don't modify the tree; if further navigation becomes impossible and we are not at a terminal node, there is no match, so increment the overall position i by 1 (i.e., $i = i + 1$), set $j = i$ and $p = root$, and repeat.

If you are able to reach a terminal node by matching the corpus to the trie, then you have found a match. If you didn't store the target sequence as a payload in the terminal node, it's easy enough to recover the string that you've matched: it's the substring of the input from i to j, inclusive. (Note that you don't need to build it as you go.)

Hint: the pseudo code of the above search algorithm is available at the end of this instruction. You can refer to it if you don't know where to start.

Running the code and the output format

The program should accept **two arguments** where

- the first is the absolute path to the target file (i.e., /dropbox/20-21/473/project4/targets);
- the second is the absolute path to the directory that contains the 3 files to be processed (i.e., /dropbox/20-21/473/project4/hg19-GRCh37/)

In this project, print the results to the standard output (which is the same as project 1 and project 2), where each line corresponds to a match in a file. The format of each line should be the following

filename + a tab + **the start index *i* of the match** + a tab + **the target string**

Here, **the index *i* starts from 0**, which means the index of the first letter in a corpus is 0; **the target string should be uppercase**. For example, if a target substring "AAG" is found in the corpus "CaAgT" in file example.dna (the matched substring is underlined), the output should be

example.dna 1 AAG

Herein, the order of the lines in the output file does not matter. I will write a script to read your output file and compare it to the gold standard.

Run your code on Patas with the following command line:

./run.sh /dropbox/20-21/473/project4/targets /dropbox/20-21/473/project4/hg19-GRCh37/ >output

Ideally, your output file should look like this:

```
chr20.dna 103769
GAAGAGAAATTATTGATTCAACCTTAAAATATTTGAAGTCTCTGGTAAGGTGACTTTAGCTAAAGCTGCAACCAGCTCAACTACAAATCAGACTGACTCACTACT
CTAGCAACTTGACAGAAGAACAATACTAGACAGAATATTAATTACCTTTTGGAGGTAATTATTTATTGAAAA
chr20.dna 424108 AACACATACTCAGAGAGAGAAAGCAGCTAATGTAAGGGCCCAAAC
chr20.dna 816630 TTTGGAAACCACACCTCCCACCCCCTCACCAAAGAATAAAAATCACTAACACCTCTTGAGCATTTACTATGTTCTAAGTACTTTACA
etc.
chr21.dna 9738211
TAACCTTGCTAGCTGAAGCTTCTGGCTCTCTTTTTTTCTTCTTCCCGTGCAGGACACTTGCTTCCCTTCTTCCTGCCCTCGGACATGAGACTCCAGGTTCTTATG
CCTTT
chr21.dna 10440880 AACTTTAAACATTAGAAATAAAATAAAAACTAGCCCAATTCCAATATTCCCCATGGATTTCCTAGCAGAAGAAATGTGAAA
chr21.dna 10909048 GACACAAAGGAGGAGGCTTTAAGAAAATGCTACTGCATTCTCT
etc.
chr22.dna 16288780
TCACTGACAGCCTGGTGTTCCCGGCAAACTTCCTGGGAGTCAGCCGAGCTTTCAGGCCATTGAGAAGCCTCTGGTGAAAGAAAAAGCTTCTTGAAGCAGGACT
GGGGCTAAGTGGCTGGAACTTGAGGATGCTGACAGCCTCCTCTGAAGAAAGCCCCCAGGACACTCCT
chr22.dna 16523019
CTCAAATCGTATGTGAGGGACAACACTCATAACCCACCAGCAGTGTTCTGGAATCCTCTGTTTGGGACAAACATTCAGAAACTCATAGCAGTGTTCTGGATTCG
TATGTGAGGGAGAAACACTCCGAACCCAACATCAGT
chr22.dna 16644036
```

TTTATAGTTTCATGTAGTTTTTTAATACTGTTCAGCATCATTATATTTTTCAACATATGGACTCTTTTGGGCATTAAAAAAATAAACAGCATCTCACTATGTTACTCA
GGCTCATCTTGAACTCTTAGCCTCAACTAATCTGCCTGCCTTGGCCTCCCAAGTCTCTGGGATTACAGACATGAGC
etc.

For reference, it takes my code around 20 minutes to process the 3 files when running on Patas.

In addition, there are small example files that you can use to test your code. You can test your code on the small example files by

./run.sh /dropbox/20-21/473/project4/targets /dropbox/20-21/473/project4/example_input/ >ex.output

Ideally, your output should match the output file at /dropbox/20-21/473/project4/example.output (the order of the lines does not matter). For reference, it takes my code around 30 seconds to process the example input when running on Patas.

Submission

hw.tar.gz

Include the following files in your submission:

`run.sh`	The command(s) that run your program.
`output`	The output file which is produced by **./run.sh /dropbox/20-21/473/project4/targets /dropbox/20-21/473/project4/hg19-GRCh37/ >output**
`(source code and binary files)`	All source code and binary files (jar, a.out, etc., if any) required to run and compile your program

Similar to previous projects, gather together all the required files and use the following command to package your files for submission.

`tar -czf hw.tar.gz ./*`

Notice that this command packages all files in the current directory; do not include any top-level directories.

Check whether your submission contains all required file using

`/dropbox/20-21/473/project4/check_project4.sh`

`readme.{pdf, txt}`

Your write-up of the project. Describe your approach, any problems or special features, or anything else you'd like me to review. If you could not complete some or all of the project's goals, please explain what you were able to complete.

Submit **hw.tar.gz** and **readme.{pdf, txt}** to Canvas.

Citation

Human Genome 19, GRCh37 Genome Reference Consortium Human Reference 37, Primary Assembly (GCA_000001405.1), February 2009. GRCh37 is a haploid assembly, constructed from multiple individuals. The primary assembly represents the assembled chromosomes, plus any unlocalized or unplaced sequence that represent the non-redundant, haploid assembly.

The pseudo code is on the next page.

pseudo code of the node

```
class node:
    def __init__():
        self.children = {
            'A': None,
            'C': None,
            'G': None,
            'T': None
        }
        self.target = None

    def set_target(target):
        self.target = target

    def set_child(input):
        self.children[input] = new node()

    def child(letter):
        return self.children[letter]
```

pseudo code of searching the trie

```
root = Trie(targets)

i = 0
while i < length(input):
    current_node = root
    j = i
    while current_node is not None:
        letter = uppercase(input[j])
        if letter is 'N':
            break
        current_node = current_node.child(letter)
        if current_node in none:
            break
        if current_node is a leaf node or current_node.target is not none:
            print file name, i, and the target
            break
        j = j + 1
    i = i + 1
```