CS 321: Data Structures, Spring 2018

Team Programming Project: Bioinformatics

Due Date: 12PM (noon) on 4/27/2018 (Friday), 200 points

1 Introduction

In this assignment, we will solve a problem from the field of Bioinformatics using BTrees. The amount of data that we have to handle is large and any data structure is not likely to fit in memory. Hence a BTree is a good choice for the data structure.

2 Background

Bioinformatics is the field of science in which biology, computer science, and information technology merge to form a single discipline. One of the primary aims of Bioinformatics is to attempt to determine the structure and meaning of the human genome. The *human genome* is a complete set of human DNA. The Human Genome project was started in 1990 by the United States Department of Energy and the U.S. National Institutes of Health. By April 14, 2003 99% of the Human Genome had been sequenced with 99.9% accuracy.

The Human Genome is a big strand of 4 different organic chemicals, known as bases, which are:

Adenine, Cytosine, Thiamine, Guanine

Biologists often call them A, C, T, G for short. The bases A and T are always paired together. Similarly the bases C and G are always paired together. So when we look at the DNA representation, only one side is listed. For example: the DNA sequence: AATGC actually represents two sequences: AATGC and its complement TTACG (replace A by T, T by A, C by G and G by C). Even with only half the bases represented in a DNA sequence, the human genome is about 2.87 billion characters long! See Figure 1 for a image of the DNA as well as the chemical structure of the bases.

The primary source for getting the human genome (as well as all other mapped organisms) is in the National Center for Biotechnology Information (NCBI) website.

`http://www.ncbi.nlm.nih.gov/`

We will be using the GeneBank files from NCBI. The format is described with a sample file at the following web link:

http://www.ncbi.nlm.nih.gov/Sitemap/samplerecord.html

Most of the information in a GeneBank file is of interest to biologists. We will only be

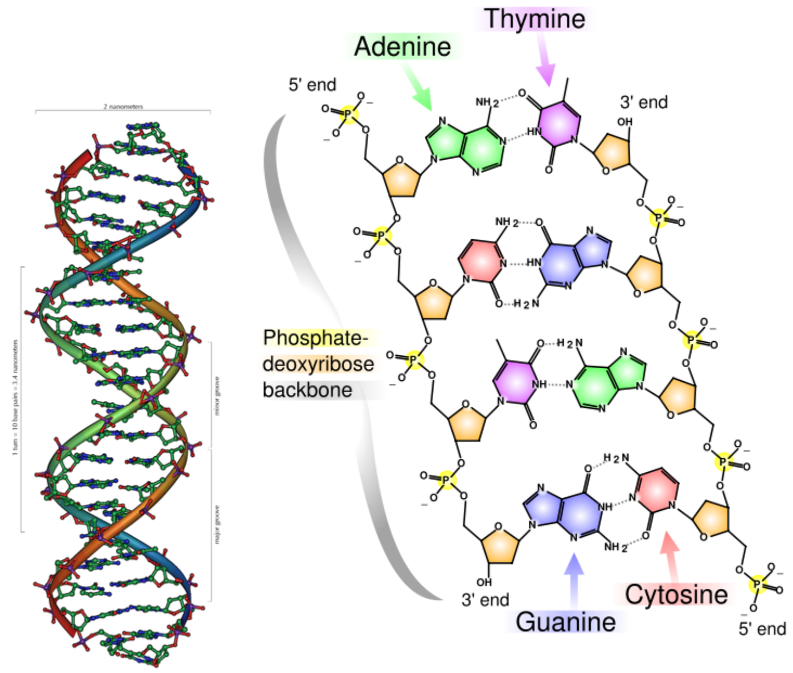

Figure 1: Physical and Chemical Structure of DNA

interested in the actual DNA sequences that are embedded in these files rather than in the intervening annotations.

3 Specifications

3.1 Input Files

The GeneBank files have a bunch of annotations followed by the keyword `ORIGIN`. The DNA sequences start from the next line. Each line has 60 characters (one of A, T, C, G, could be lower/upper case) until the end of sequence, which is denoted by `//` on a line by itself. Sometimes you will see the character `N`, which denotes that the sequence is not known at that character. You would skip these characters. One GeneBank file may have several DNA sequences in it, each marked by `ORIGIN` and `//` tags.

When we reach a character `N`, we assume that the subsequence has ended. Similarly, when we reach `//`, we also assume that the subsequence has ended. So at those points, we reset the subsequence that we were building and start over when we find the next `ORIGIN` or the next valid character after seeing a `N`.

Some sample genome files (.gbk) can be found at:

```
cp ~jhyeh/cs321/labs/lab4/BTree/data/*.gbk ~/cs321/p4/data/.
```

You may simply copy these files as shown above.

3.2 Problem

For a given GeneBank file, we want to convert it into a BTree with each object being a DNA sequence of specified length k (where $1 \leq k \leq 31$). We will take the DNA sequence from the GeneBank file and break it into sequences of length k each. We are interested in all subsequences with length k. For example, in the sequence AATTCG, the subsequences of length three are: AAT, ATT, TTC and TCG. Once we have a BTree for a length k, we want to be able to search for query sequences of length k. The search returns the frequency of occurrence of the query string (which can be zero if it is not found).

The biological motivation behind is to study the frequency of different length subsequences to see if they are random or that some sequences are more likely to be found in the DNA.

4 Design Issues

Saving memory. Since we only have four possible bases (A, C, G and T), we can optimize on space by converting our DNA strings to base 4 numbers. Then we can represent each base by a 2-bit binary number, as shown below:

A	00
T	11
C	01
G	10

Note that we have made the binary representations for complementary bases be binary complements as well. With this compact representation, we can store a 31 length sequence in a 64-bit `long` primitive type in Java.

Key Values. Note that the binary compact representation of the subsequences will result in a unique 64-bit integer value. Hence we can directly use that as our key value.

Class Design. We will need a `BTree` class as well as a `BTreeNode` class (which should be an inner class). The objects that we store in the BTree will be similar to the objects we stored in the previous Hashtable assignment. You may call the relevant class `TreeObject` to represent the objects.

4.1 Implementation

You will have two programs: one that creates a BTree from a given GeneBank file and another for searching in the specified BTree for sequences of given length. The search program assumes that the user specified the proper BTree to use depending upon the query length.

- The main Java classes should be named `GeneBankCreateBTree` and `GeneBankSearch`.

  The required arguments for the two programs are shown below.

  ```
  java GeneBankCreateBTree <0/1(no/with Cache)> <degree> <gbk file> <sequence length>
  [<cache size>] [<debug level>]
  ```

  ```
  java GeneBankSearch <0/1(no/with Cache)> <btree file> <query file> [<cache size>]
  [<debug level>]
  ```

  The degree is the degree to be used for the BTree. If the user specifies 0, then your program should choose the optimum degree based on a disk block size of 4096 bytes and the size of your BTree node on disk.

  The sequence length is an integer that must be between 1 and 31 (inclusive). The query file contains all the DNA strings of a specific sequence length that we want to search for in the specified btree file. The strings are one per line and they all must have the same length as the DNA sequences in the btree file. The DNA strings use a, c, t, and g (either lower or upper case).

  The debug level is an optional argument with a default value of zero. It must at least support the following levels for `GeneBankCreateBTree`:

  0 Any diagnostic messages, help and status messages must be be printed on standard error stream.

  1 The program writes a text file named `dump`, that has the following line format: `<frequency> <DNA string>`. The dump file contains frequency and DNA string (corresponding to the key stored) in an inorder traversal.

  The debug level for `GeneBankSearch` must at least support the following.

  0 The output of the queries should be printed on the standard output stream. Any diagnostic messages, help and status messages must be be printed on standard error stream.

- *Your program should always keep the root node in the memory.* Write the root node to disk file only at the end of the program and read it in when the program starts up. In addition, your program can only hold a few nodes in memory (that is, declare a few BTreeNode variables, including the root, in your program).

- **Metadata storage**. We need to store some metadata about the BTree on disk. For example, we can store the degree of the tree, the byte offset of the root node (so we can find it), the number of nodes etc. This information could be stored in separate metadata file or it can be stored at the beginning of the BTree file.

- The B-Tree should be stored as a binary data file on the disk (and not as a text file). If the name of the GeneBank file is `xyz.gbk`, the sequence length is *k*, the BTree degree is *t*, then the name of the btree file should be `xyz.gbk.btree.data.k.t`.

- You need to turn in a README file that describes the layout of the B-Tree file on disk as well as any other relevant observations.

- Sample query files are available in the directory:

  `~jhyeh/cs321/labs/lab4/BTree/queries/`

  That directory also contains a sample program named `QueryGenerator.java` that generates random queries for testing.

- **Running Tests**. Please be sensitive to other students when you run large tests on your assignments. Preferably, run all your tests at home. If you do need to run them in the lab, please do not run them on `onyx`. Instead run them on one the workstations. If you are log in to onyx remotely via `ssh`, then `ssh` into a idle workstation to run your tests. You can check if anyone is on a workstation with the command `who`. You can also check the load on the system with the command `top`. The workstations are named `node00` through `node32`.

5 Using a Cache (20 points)

- Incorporate the Object Cache from the earlier assignment to improve the performance of your Btree implementation. The size of the Cache should be a command line argument. An entry in the Cache is a Btree node. With the Cache option, the cache size needs to be specified as a command-line argument.

  ```
  java GeneBankCreateBTree <0/1(no/with Cache)> <degree> <gbk file> <sequence length>
  [<cache size>] [<debug level>]
  ```

  ```
  java GeneBankSearch <0/1(no/with Cache)> <btree file> <query file> [<cache size>]
  [<debug level>]
  ```

- Report the improvement in time using a cache of size 100 and 500 in your README file.

6 Submission

You can submit your project via Blackboard.

7 Progress Reports

Each team member is responsible for sending the TA, via email, a short progress report each week. To clarify:

one progress report, per week, per member

A progress report should describe your project-related activities for the week. I expect each team should have at least one meeting every week. You can (and should, if necessary) complain about your teammates, and I'll take their behavior into consideration. Progress reports are confidential.