

Programming Project: Genome Annotation Browser

CMSC 420 — Data Structures — Spring 2008

Introduction.

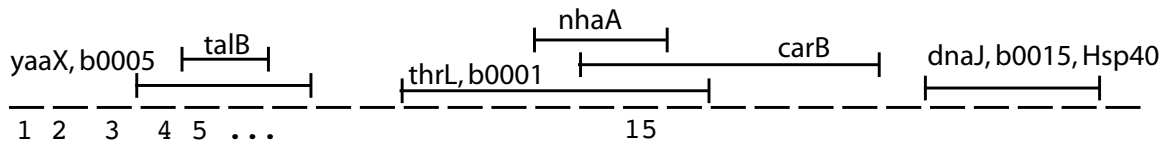
The human genome consists of DNA molecules which can be thought of as a linear sequence of the letters A, C, G, and T. Some regions of the DNA encode for genes. Because the human genome is very large, computers are needed to search and study it. This project will develop a toolkit for storing and querying the locations of genes along the genome. You will not need to know any biology for this project.

In part 1a, you will develop a data structure for mapping gene names to gene locations. In part 1b, you will use your interval tree to perform some analysis about which genes interact with one another.

The general structure of your program will be to read in commands from standard input and perform the required action. Details about the operations that should be supported are given below. Your program must be written in C or C++.

Gene Representation.

We will think of the genome as an integer number line, and genes as ranges $[x_1, x_2]$, where $0 \leq x_1 < x_2$. In addition, each gene will have at least one name associated with it:



To represent each interval, you should create a Gene structure with at least the fields shown below:

```
struct Gene {
    int x1, x2;           // gene from x1 to x2 inclusive
    list<string> names;  // a list of strings
};
```

`names` is a list that holds all the names of this gene. The variables `x1` and `x2` hold the start and end of the interval covered by the gene. For this project, you are not allowed use STL container classes such as `list<>`, `vector<>` **except** for this one use of the `list` class. You can use the STL `string` class.

Your main program should be called `gene_mapper`.

Part 1a. Mapping Names to Genes

Your program should handle the following commands, provided on standard input, that let the user map multiple names to a single gene. These operations represent a standard *dictionary ADT* mapping gene names to **Gene** objects with one wrinkle: several keys may point to the same object. You must implement the dictionary using a **splay tree**. The commands you must support are:

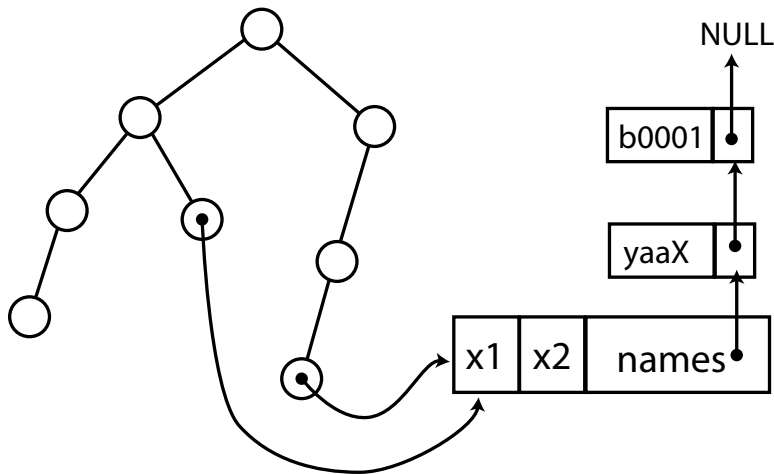
- **AddGene name x1 x2**: create a new **Gene** structure at the coordinates x_1 , x_2 with the given name and insert a mapping between **name** and the **Gene** structure into the dictionary. If a gene already exists with that name, an error should be reported.
- **AddGeneAlias name1 name2**: create a new mapping between **name2** and the **Gene** object associated with **name1**. Also add **name2** to the list of names in the Gene's names list. If **name2** is already assigned to any gene, or if **name1** is not assigned to a gene, an error should be reported and nothing should be changed.
- **RemoveAlias name**: remove **name** from the dictionary and from the appropriate gene's names list. If the names list becomes empty, the associated **Gene** should be deleted (freed). Thus, genes can be deleted by removing all aliases to them. If the given alias does not exist, an error should be reported.
- **PrintGeneInfo name** should print the coordinates and the names for the given gene, in the format:

x_1 x_2 alias1 alias2 alias3 ...

Output for each gene should be contained on a single line, and each element should be separated by a single space.

- **quit** should end the program.

Error messages: When reporting an error, you should print an error message to standard output. The error message should start with the string "Error: ". This should be followed by a descriptive, but short, explanation of the problem. The entire error message should be on a single line.



Part 1b. Investigating the Gene Regulatory Graph

Some genes turn other genes on or off. If gene A turns gene B on or off, we say that A *regulates* B . This ordered, pairwise relationship can be used to define a directed graph, with nodes for each gene and an edge (A, B) if A regulates B . You should implement a data structure suitable for storing sparse, directed graphs in order to respond to the following commands:

- `Regulates name1 name2`: add a directed edge from gene `name1` to the gene with name `name2`. If either `name1` or `name2` are not valid gene names, report an error. If the edge $(name1, name2)$ already exists between them, do nothing.
- `DoesntRegulate name1 name2`: remove a the edge going from the gene with name `name1` to the gene with name `name2`. If either `name1` or `name2` are not valid gene names, report an error. If no such edge is present, do nothing. Your graph should never contain isolated nodes: if a gene is not currently known to regulate any other genes and it is not known to be regulated by any genes, it should not be included in the graph.
- `PrintGraph`: print the graph using the following format. The first line should be

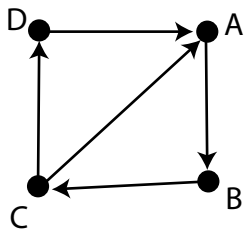
```
digraph G {
```

The last line should contain only a single “}”. In between, there should be a single line of the following format for each edge:

```
name1 -> name2
```

where `name1` and `name2` are names for the two genes connected by the edge. If a gene has more than one name, you can arbitrarily pick one to print, but you must consistently use the same name to refer to the same gene. Remember: the vertex set of the graph is the set of genes not the set of names.

- `PrintFeedBackLoops name k`: Use a BFS to print all directed, simple cycles that return to gene `name` in fewer than $\leq k$ steps. A cycle $A \rightarrow B \rightarrow C \rightarrow D \rightarrow A$ should be printed in the format `A B C D`. Cycles should be separated by semicolons. For example:



```
PrintFeedBackLoops A 3  
A B C  
PrintFeedBackLoops A 4  
A B C ; A B C D
```

In the case that a gene has more than one name, you can use any valid name to refer to it, but you must be consistent within the output of a single `PrintFeedBackLoops` command. (You may need to implement a queue or similar list-like data structure to perform the BFS and to keep track of paths you find.)

Finally, you should modify how you handle `RemoveAlias` to delete nodes (and their incident edges) if a gene is deleted.