Department of Computer Science and Software Engineering
Concordia University

SOEN 6461 Software Design Methodologies
Design Assignments Fall 2015

Design Assignments

The assignments require you to practice creating a design, communicating your design, and analysing your design to demonstrate that the design works, and that the design has the properties — for example, performance, security, usability, etc — required for the task.

You do not program your designs, so your program can not serve as a way to check your design. You must be able to communicate and check your design independently of a program.

Design work requires you to be able to express your ideas about a design to your fellow designers. This includes the key features of the design, why these features are good, or necessary, and what are the consequences of using this design feature (both positive consequences and negative consequences).

These design conversations can happen in formal review meetings, or as informal chats around the water cooler.

These design conversations happen no matter which methodology you are using for software development.

Assignments 2015 — Patterns in Genome Sequences

Overview

The assignments deal with finding short patterns in strings; in this case the strings are genome sequences.

A **genome** sequence is a string over the alphabet of four nucleotides$\{a, c, g, t\}$. A genome can be very long, from 100 thousand to 100 billion nucleotides (nt). For example, the human genome has length about 3.2 $\times 10^9$ nt.

A **pattern** is a k-mer, a string of nucleotides of length k. The values of k that are of interest in pattern finding varies from $k = 1$ to $k = 37$. Note that there are 4^k possible k-mer's, and that 4^{37} is 2^{74} which is approximately 16×10^{21} — because $2^{10} = 1024$, which is approximately $10^3 = 1000$.

When identifying organisms, scientists want to be able to detect a pattern or a combination of patterns that distinguish one organism from all other organisms. For example, in food safety testing, certain strains of common bacteria are harmful to us, while other strains are perfectly safe. Is there a k-mer p in the genome of an unsafe strain X that is unique to that strain?

Sometimes we need several patterns in order to be able to distinguish between strains or organisms. That is, suppose we can identify a list $\vec{P} = (p_1, p_2, \ldots, p_m)$ of patterns p_j, such that no two strains have the same combination of patterns. That is, the binary string $B_{\vec{P}}(X) = b_1 b_2 \ldots b_m$ of zeroes and ones, where a one for b_j indicates that the genome of organism X has pattern p_j, and a zero indicates that the pattern is absent in the genome of X. The list $\vec{P} = (p_1, p_2, \ldots, p_m)$ of patterns are called **biomarkers**, and the binary number $B_{\vec{P}}(X) = b_1 b_2 \ldots b_m$ is called the **signature** of X.

Assignment One

The Problem

You are given a genome as a string S of length n over the alphabet = { a, c, g, t }; and an integer k between the values of 1 and 37.

Your system is to compute the set of k-mers p over the alphabet = { a, c, g, t } that occur in the genome S, together with a count of the number of occurrences of p, and a list of positions i of p in S.

A Small Example

The genome is a string S= ataaaa of length $n = 6$ over the alphabet = { a, c, g, t }. The input value of k is 2; hence, a pattern is a 2-mer, a string of 2 nucleotides. The task is to determine all patterns of size 2, count the number of occurrences of each pattern, and determine the position of each occurrence in the genome S.

For this example, the result would be

Pattern	Count	Positions
aa	3	2,3,4
at	1	0
ta	1	1

Note that the positions in the genome are numbered from 0 to 5.

Guidance

This assignment focuses on the design of classes, both in terms of being real-world concepts, but also in terms of data representations in memory or on disk. In assignment one we really want to avoid disk usage during the computation because of the high performance costs.

For class design, it is important to select appropriate data structures and operations. Furthermore, the classes collaborate in order to perform the computation, so the design must make the operations work together. The choice of data representations for each class must

be consistent, otherwise there is the overhead of transforming from one data representation to another at each step of the algorithm.

For these assignments, you need to be concerned about

- that your program computes the correct result;
- that your program makes efficient use of cpu time;
- that you program makes efficient use of memory;
- that your program scales to handle genomes of size 100 billion nt; and
- that your program scales to handle patterns of size k=37.

When considering scaleability, think about how large a problem — that is, n and k — your program can execute in the following three systems:

- 4 GB of memory for data;
- 128 GB of memory for data; and
- 1024 GB of memory for data.

Data Structures For this assignment, you need to design several key data structures. Your design needs to have an in-memory data representation for

- a nucleotide, which is a member of { a, c, g, t };
- the genome itself, which is a string of length n;
- a pattern, which is a string of length k; and
- the result of the computation, which is a collection of patterns, together with a count, and a list of positions.

You need to be aware of the memory resources used by each data structure. How many bits or bytes or words are required by each element of the data structure? How many entries are there in each data structure? What is the total size of each data structure?

Operations The overall algorithm for the program is a sequential scan of the genome from position i=0 to position i=$n-1$; and storing the information about the pattern $p = S[i \ldots i+k-1]$ in the collection of patterns.

For this assignment, there are key operations associated with the collection of patterns:

- given a pattern, find the pattern in the collection;
- update the count and positions for a pattern, once you have found the pattern in the collection; and
- add a new pattern to the collection, if you could not find the pattern in the collection.

A very important part of these operations, is the data representation of a pattern, and the related operation of

- fetching the pattern at position i of the genome;

and the comparison operator between patterns for the operation of

- compare a pattern p from the genome with each pattern q already in the collection.

You need to be aware of the number of steps used by each operation, including the steps that move data. Remember that the underlying hardware operations will handle data in terms of bytes, 32-bit words, 64-bit words, and that these units of data are at specific locations in a memory's 64-bit word. So think about how data is aligned in a word, and how you pad unused bits of the word, since these affect your operations.

Consider the scenario where the pattern $p = S[i \ldots i+k-1]$ at position i spans the boundary from one word of memory to the next; that is, for some value of j, $S[i \ldots j]$ are bits L..63 for a word and $S[j+1 \ldots i+k-1]$ are bits 0..M of the next word. When $k > 32$, the pattern will always span a 64-bit word boundary, and may even span two boundaries in some circumstances.

You may need to design an example of the Iterator design pattern that iterates along the genome sequence, and also handles the data representation of the pattern at position i.

Classes and Objects You may want separate classes for the **concept** of genome, pattern, and collection of patterns and the **representation** of these concepts. Alternatively, the concept class may have several *auxiliary* or *helper* methods implementing the operations for the data representation of the concept.

Hashing If you decide that a hash index is appropriate for searching the collection of patterns, then you need to be explicit about the key decisions: the choice of **hash function**, how the hash function is computed, and which class has the method for the hash function. Furthermore, you need to be clear on how you resolve collisions, and how you handle overflow of the hash table or bins.

Assignment Two

The Problem

You are given several genomes S_1, S_2, ..., S_M over the alphabet = { a, c, g, t } of lengths n_1, n_2, ..., n_M respectively; and an integer k between the values of 1 and 37.

Your system is to compute the k-mers p_1, p_2, ..., p_M over the alphabet = { a, c, g, t } such that p_i occurs in the genome S_i and does not occur in any other genome. If there is no pattern unique to genome S_i then raise an exception.

For convenience, consider $k \leq 16$ so that the pattern fits into a single 32-bit word; consider the genomes to be bacterial genomes of length about 5 million nucleotides; and consider that there are about $M = 100$ genomes S_1, S_2, ..., S_M.

A Small Example

Consider three genomes of length $n = 6$ over the alphabet = { a, c, g, t }.

- S_1 = ataaaa
- S_1 = atcgta
- S_3 = atagcg

The input value of k is 2.

Consider the k-mers in each genome:

Genome	Patterns
S_1	aa, at, ta
S_2	at, cg, gt, ta, tc
S_3	ag, at, cg, gc, ta

Each genome has a pattern that is unique to that particular genome, namely aa, gt, ag respectively.

Guidance

This assignment builds on assignment 1. It combines object-oriented design and functional design in that key processes in the solution are

- compute the set $\mathcal{P}(S_i)$ of patterns p of length k over the alphabet $A = \{a, c, g, t\}$ in the genome S_i; that is $\mathcal{P}(S_i) = \{\ p \in A^k \mid p \text{ occurs in } S_i\ \}$; and
- compute the set difference $\mathcal{P}(S_i) \setminus \mathcal{P}(S_j)$

because conceptually the set of unique patterns $U(S_i)$ in a genome S_i is defined as

$$U(S_i) = \mathcal{P}(S_i) \setminus \left(\bigcup_{j \neq i} \mathcal{P}(S_j) \right) \tag{1}$$

which are the patterns p that occur in the genome S_i but do not occur in any other genome S_j.

These computational processes can be designed as operations for an appropriate class, or they can be designed to be a class by themselves, much as you would define a web service to provide the service of performing a computation. You should have a class that is the overall algorithm for the problem. Furthermore, you should consider the advantages and disadvantages of representing the two processes above as their own class.

Note that if there are M genomes S_1, S_2, ..., S_M then you may require $\binom{M}{2}$ set differences, which is $M \times (M-1)/2$.

Data Structures In assignment 1 you computed the collection of patterns, together with the counts and the positions of each pattern. In addition to that, this assignment requires many computations with a **set** of patterns. The representation of the set and the implementation of the set operations, especially set difference, are critical to performance and memory usage.

A set can be represented by a list of the patterns in the set.

A set can be represented as a bitmap for A^k where the l-th bit is set to one if and only if the l-th pattern of A^k is in the set.

The bitmap can be compressed using run-length encoding. This may be advantageous when the set is sparse and the size of A^k is very large.

What is the cost in terms of memory and cpu instructions for the set difference operation for each of these representations?

Some Numbers Consider problems working with bacterial genomes of size n about 5×10^6 nucleotides.

Consider the case of $k = 16$ where each pattern requires one 32-bit word for storage.

A genome of size n has $n - k$ substrings of length k. Hence, there are at most $n - k$ distinct patterns p in the genome. So each bacterial genome has at most 5×10^6 distinct patterns.

Storing the collection $C(S) = \{\langle p, \mathit{count}, \mathrm{list}(\mathit{position})\rangle \mid p \in S\}$ of patterns requires for each distinct pattern, the pattern, a count, and a list of positions. The total number of positions is $n - k$. Assume each pattern, count, and position requires a 32-bit word. Then the total storage is $2 \times |C| + n - k$, which is bounded by $3 \times n$ 32-bit words, about 60 MB.

So 100 genomes require about 6 GB of memory for the collection of patterns.

What is the size of A^k? For $k = 16$, there are $4^{16} = 2^{32}$ distinct k-mers; which is about 4×10^9 k-mers.

Therefore, a bacterial genome has at most 1/1000 of the possible patterns. Amongst 100 genomes therefore it is likely to find a unique pattern for each genome.

For k=16, storing every k-mer requires 2^{32} 32-bit words, which is 2^{34} bytes, which is 16 GB.

For k=16, storing a set of k-mers as a bitmap of the 2^{32} k-mers in A^k requires 2^{32} bits, which is 2^{29} bytes, which is 512 MB. Hence, storing the information on patterns for 100 genomes is about 50 GB.

What if you had 10,000 genomes? Then storage would be 5,000 GB of memory. And the likelihood of there being a pattern unique to a single genome is low.

Assignment Three

The Problem

You are given several genomes S_1, S_2, ..., S_M over the alphabet = { a, c, g, t } of lengths n_1, n_2, ..., n_M respectively; and an integer k between the values of 1 and 37.

Your system is to compute an integer m, a biomarker list $\vec{P}$ of k-mers (p_1, p_2, ..., p_m), and the signature $B_{\vec{P}}(X) = b_1 b_2 \ldots b_m$ of each genome X relative to the biomarkers $\vec{P}$.

To be biomarkers, the signature of each genome must be a unique binary number.

Ideally, the number m of biomarkers should be small. However, you are not required to find the smallest possible list of biomarkers.

If there are no biomarkers that distinguish each and every genome then raise an exception.

For convenience, consider $k \leq 16$ so that the pattern fits into a single 32-bit word; consider the genomes to be bacterial genomes of length about 5 million nucleotides; and consider that there are up to $M = 10{,}000$ genomes S_1, S_2, ..., S_M.

A Small Example

Consider five genomes of length $n = 6$ over the alphabet = { a, c, g, t }.

- S_1 = ataaaa
- S_1 = atcgta
- S_3 = atagcg
- S_4 = cgtaag
- S_5 = tagcgt

The input value of k is 2.

Consider the k-mers in each genome:

Genome	Patterns
S_1	aa, at, ta
S_2	at, cg, gt, ta, tc
S_3	ag, at, cg, gc, ta
S_4	aa, ag, cg, gt, ta
S_5	ag, cg, gc, gt, ta

No genome has a pattern that is unique to that particular genome. However, consider the set of patterns across every genome as the biomarkers, and then look at the signature of each genome:

Genome	Patterns							
	aa	ag	at	cg	gc	gt	ta	tc
S_1	1	0	1	0	0	0	1	0
S_2	0	0	1	1	0	1	1	1
S_3	0	1	1	1	1	0	1	0
S_4	1	1	0	1	0	1	1	0
S_5	0	1	0	1	1	1	1	0

Therefore (aa, ag, at, cg, gc, gt, ta, tc) are biomarkers. Note that the first three patterns (aa, ag, at) also form biomarkers that distinguish each genome.

Guidance

This assignment has no simple algorithm to find the solution. You need to explore a heuristic solution where the design has a Strategy class for the heuristic.

For the solution, the heuristic should identify patterns that distinguish between sets of genomes by occuring in some of the genomes but not in others.

A common heuristic is the greedy algorithm. The greedy algorithm solves – finds biomarkers $\mathcal{P}_2$ — the case for $\{S_1, S_2\}$ and then extends the solution $\mathcal{P}_2$ to a solution $\mathcal{P}_2$ for the case for $\{S_1, S_2, S_3\}$. And so on to a solution $\mathcal{P}$ for $\mathcal{S} = \{\ S_1,\ S_2,\ \ldots,\ S_M\ \}$.

You could apply divide-and-conquer to solve the problem as a whole. That is, identify subproblems, solve each subproblem, and then build the final solution from the set of sub-solutions.

You could consider using the blackboard architecture as the basis of the design. However, consider how the use of the blackboard architecture complicates the design. Is it necessary? Is it helpful in making the design understandable?

Often heuristics are difficult to analyse in terms of computation time. This is generally not so for divide-and-conquer heuristics, as they naturally lead to recurrence relations.

Data Structures You need to represent the set of biomarkers.

Membership of a pattern in a $\mathcal{P}(S)$ is required to determine the bits in the signature of S.

Your heuristics might take advantage of the count of a pattern in $C(S)$.

Biomarkers A set of biomarkers $\vec{P}$ of m k-mers determines a signature $B_{\vec{P}}(X)$ which is a binary number with m binary digits. Such a binary number can distinguish at most 2^m genomes. Hence, we need at least $m = \log_2(M)$ biomarkers to distinguish M genomes. So 10,000 genomes requires at least 14 biomarkers to distinguish them.

Here are some properties of biomarkers that may be useful in a divide-and-conquer heuristic.

Lemma 1 If there exist a set of biomarkers for two genomes S_1 and S_2 then there is a set of biomarkers of size 1 consisting of a single pattern p.

Proof If $\vec{P}$ is a set of biomarkers for S_1 and S_2 then there is some bit b_i which is different in the two signatures. Take p to be the corresponding biomarker p_i. □

Corollary 1 If p is a pattern that occurs in genome S_1 but does not occur in genome S_2, then (p) is a set of biomarkers for $\{\ S_1,\ S_2\ \}$.

Lemma 2 If $\vec{P} = (p_1,\ p_2,\ \ldots,\ p_m)$ is a set of biomarkers for the genomes $\mathcal{S} = \{\ S_1,\ S_2,\ \ldots,\ S_M\ \}$, then any permutation of the patterns $(p_1,\ p_2,\ \ldots,\ p_m)$ is also a set of biomarkers for $\mathcal{S}$.

Proof Follows directly from the definition of biomarkers and signature. □

Lemma 3 If $\vec{P} = (p_1, p_2, \ldots, p_m)$ is a set of biomarkers for the genomes $\mathcal{S} = \{ S_1, S_2, \ldots, S_M \}$, and p_{m+1} is a pattern not in $\vec{P}$ then the patterns $(p_1, p_2, \ldots, p_m, p_{m+1})$ are also a set of biomarkers for $\mathcal{S}$.

Proof Follows directly from the definition of biomarkers and signature. □

Lemma 4 If $\vec{P}$ is a set of biomarkers for the genomes $\mathcal{S} = \{ S_1, S_2, \ldots, S_M \}$, and $\mathcal{T}$ is a subsect of the genomes $\mathcal{S}$, then $\vec{P}$ is also a set of biomarkers for $\mathcal{T}$.

Proof Follows directly from the definition of biomarkers and signature. □

Lemma 5 If $\vec{P} = (p_1, p_2, \ldots, p_m)$ is a set of biomarkers for the genomes $\mathcal{S} = \{ S_1, S_2, \ldots, S_M \}$, S is a genome not in $\mathcal{S}$, and p is a pattern in $\mathcal{P}(S)$ that does not occur in any genome in $\mathcal{S}$, then the patterns $(p_1, p_2, \ldots, p_m, p)$ are a set of biomarkers for $\{ S_1, S_2, \ldots, S_M, S \}$.

Proof Follows directly from the definition of biomarkers and signature. □

Not-Lemma If $\vec{P}$ is a set of biomarkers for the genomes $\mathcal{S}$ and $\vec{Q}$ is a set of biomarkers for the genomes $\mathcal{T}$ then the combination of $\vec{P}$ and $\vec{Q}$ is not necessarily a set of biomarkers for $\mathcal{S} \cup \mathcal{T}$.

The counterexample for the proof is that the combined biomarkers do not necessary distinguish between a genome S from $\mathcal{S}$ and a genome T from $\mathcal{T}$.

Your Submission

There are three assignments were you must create, evaluate, and document a design.

The deliverable is a design document of about five (5) pages as a pdf file, submitted to the ENCS electronic submission system. The first line of your document should identify the course, the assignment number, your name, and your student number; do not have a separate title page. Use a 12pt font or larger throughout the document.

The Design Document

The document is meant to be short and concise. The three design documents are practice at communicating. They are also practice in identifying which of your design decisions are obvious decisions to other designers, and which design decisions are important, critical to the success of the design, and not obvious. You must communicate the important information. There is no need to document the obvious design decisions.

Your design document should have subsections:

1. a brief description of the problem (up to one half (0.5) page);
2. a concise description of the design (from two (2) to four (4) pages), where you focus on presenting the most important parts of the design (whatever they may be);
3. a brief description of the major design decisions (one half (0.5) to one (1) page);
4. a brief description on how you reviewed the design (one half (0.5) to one (1) page), and which qualities of the design were considered during the review;
5. a glossary of important things (one half (0.5) to one (1) page in length), such as classes, objects, methods, algorithms, and data structures.

Write your document for a reader at the level of a CS or SOEN graduate like yourself. Use UML diagrams where and when they are appropriate for communicating information about the design, but do not use them if they are not needed.

Marking Scheme

For each design document, the marking scheme will assign a mark out of 2 for each of the subsections listed above, for a total mark out of 10:

- 0 marks for a missing subsection, or very unclear section, or where the section content is not appropriate for the section (that is, it is mis-placed);
- 1 mark for an informative subsection that does not have too many errors;
- 2 marks for a clear, informative, concise, almost error-free subsection.

Length Penalty: There is a penalty of 10% of the assignment mark for each page of your submission over 7 pages.
Late Penalty: There is a penalty of 10% of the assignment mark for each day that your assignment is late.