Repeat Assignment: DNA Sequence Building

SOFT7019

Due date: 17th August 2023

1 Problem outline

DNA sequences are composed of individual building blocks called nucleotides or bases. There are 4 different types of bases: A (adenine), C (cytosine), G (guanine), and T (thymine). The goal of this project is to implement a C application which can create ordered lists of these DNA bases to create DNA sequences.
In this application, DNA sequences will develop, grow, and mutate over time. Time should be represented by iterations of a loop and the DNA sequences should be able to record their age measured in loop iterations.

2 Solution requirements

In your project you will need to implement a data structure that can represent DNA sequences. This structure should be implemented using a doubly linked list, that is a linked list where each node is connected to the 'next' node and the 'previous' node.

- Representing the bases
  - Each base building block should be represented by a struct.
  - The struct should contain a char field representing the type of base A, C, G, or T.
  - Hint: write a function to create new base structs specifying the type of base as a function parameter.
- Building the sequences
  - Each DNA sequence should be represented by a new data structure. As each sequence will be a doubly linked list, each node should have a field that references the next node and a field that references the previous node. There should also be a field that points to a DNA base struct.
  - You should be able to traverse through the list to print the DNA sequence.
  - Each sequence should have a mechanism for recording the following information relating to that sequence:
    - current age
    - maximum lifespan
    - probability of growth
    - probability of deletion

    - probability of mutation
  - These sequence properties can be stored in the head only, or in each node in the list; it is up to you how you choose to record this information.

- Useful functions to implement: the following basic functions will help you when interfacing with linked lists
  - Write a function to add a base at the end of an existing sequence
  - Write a function to insert a base at a specific position in the sequence. The function should take as input the base to be inserted and the position. The resulting sequence should reflect the insertion correctly.
  - Write a function to delete a base from a specific position in the sequence. The function should take as input the position of the base to be deleted. The resulting sequence should reflect the deletion correctly.
  - Write a function that searches for a given DNA pattern (2 or more bases long) within the current sequence. Return the starting position of the matched pattern if found.

- DNA lifecycles - upon startup your application should instantiate a new DNA sequence, follow its lifecycle until its death, and then repeat the process with a new sequence.
  1. Randomly initialise a new DNA sequence:
     - The length of all initial sequences should be 20 bases long
     - The type of each base (A, C, G, or T) should be randomly assigned
     - The max lifespan of the sequence should be a random number between 8 and 50 iterations
     - The current age should be set to 0
     - The probabilities of growth and deletion should each be set to 0.25
     - The probability of mutation should initially be set to 0
  2. Iterate through a lifecycle loop, where each iteration represents an increase in the DNA sequence's current age
  3. Upon each iteration the probability of growth determines the likelihood that the DNA sequence will increase by 1 base - this increase should occur at a randomly selected position in the current sequence and the added base (A, C, G, or T) should be randomly assigned
  4. Upon each iteration the probability of deletion determines the likelihood that the DNA sequence will decrease by 1 base - this deletion should occur at a randomly selected position in the current sequence (if all bases are deleted from a sequence the sequence automatically dies)
  5. After the 5th iteration of any sequence's lifecycle the probability of mutation should be set to 0.1, it should increase by 0.01 upon every successive iteration.
  6. Upon each iteration the probability of mutation determines the likelihood that the DNA sequence will mutate, mutation means that one of the bases in a sequence will be randomly assigned a new value.
  7. If any one base is repeated 3 or more times a special behaviour should follow, these special patterns and their associated effects are listed below. Upon each iteration the application should check for the presence of any of these repeating patterns.

8. The sequence or the updated version or the sequence should be printed upon each iteration of the lifecycle loop.
9. If the sequence reaches the end of its lifecycle or it dies for some other reason a message should be printed and the user is given the option of starting with a new sequence.

- Patterns with special properties
  - If the pattern "AAA" appears in the sequence, the life of that DNA sequence should be extended by 1 iteration. For each additional A in the repeating pattern, the life should be extended by a further iteration.
  - If the pattern "CCC" appears in the sequence, the life of that DNA sequence should be reduced by 1 iteration. For each additional C in the repeating pattern, the life should be reduced by a further iteration.
  - If the pattern "GGG" appears in the sequence, the length of the DNA sequence should be halved with every second base being deleted.
  - If the pattern "TTT" appears in the sequence, the DNA sequence should double in size. A reversed version of the sequence should be concatenated onto the end of the sequence, resulting in a pattern that is a palindrome.

3 Grading (60)

- representing the bases - 2
- building the sequences using a doubly linked list - 4
- basic linked list operations (14)
  - append a node - 3
  - insert a node - 4
  - remove a node - 4
  - update list information - 3
- sequence initialisation - 2
- iterating the lifecycle loop and updating sequence age - 2
- manage growth and deletion based on random likelihoods - 10
- manage mutation probability and implement mutations based on likelihood - 6
- print the sequence - 1
- manage the end of a sequence's lifecycle (4)
  - free all memory - 3
  - user decides if another sequence should be created - 1
- patterns with special properties (15)
  - search for patterns - 5

- extended or reduce lifespan - 2
- reduce the sequence length by half - 4
- double the sequence size - 4