Trabalho 1 de Algoritmos de Bioinformática: Definição de uma classe de manipulação de sequências

Edgar Carneiro
Faculdade de Ciências da Universidade do Porto
(Dated: 6 de Março de 2019)

I. INTRODUÇÃO

Neste relatório, no âmbito da disciplina de Bioinformática, disciplina que aborda temáticas atuais e relevantes na área da investigação, será abordada a implementação de um conjunto de classes com funcionalidades para representar e manipular sequências biológicas. As classes implementadas lidam com três tipos de sequências: *DNA*, representante da sequência biológica ácido desoxirribonucleico, *RNA*, representante do ácido ribonucleico, e *Protein*, representante de cadeias de aminoácidos. Esta implementação trata-se de uma base para futuros trabalhos desenvolvidos na disciplina previamente mencionada.

II. DESCRIÇÃO E ESTRATÉGIAS DE IMPLEMENTAÇÃO

Foram consideradas duas estratégias de implementação do problema apresentado. Por um lado, o desenvolvimento de todas as funcionalidades dentro da classe *BioSeq*: este estratégia tem a vantagem de seguir rigidamente as especificações requisitadas pelo problema, tais como o uso de uma só class e a necessidade de existência de um atributo na classe que guarde qual o tipo de sequência a que se refere uma instância. No entanto, esta estratégia introduz algumas desvantagens, tais como: permite a chamada de métodos em tipos nos quais estes não são adequados, como por exemplo a chamada do método *translation* num sequência do tipo *Protein*. A limitação de termos tudo numa só classe, obriga-nos também a garantir que nunca nos esquecemos de definir como um determinado método irá interagir com uma determinado tipo, forçando assim a um cuidado acrescido por parte do desenvolvedor.

Por outro lado, o desenvolvimento da classe *BioSeq* como uma interface de criação de objetos dos tipos designados, sendo estes baseados num modelo de classes e herança. Este modelo trás vantagens como a completa desagregação dos diferentes tipos de sequências de funcionalidades que não fazem sentido estes implementarem, isto é, com esta representação torna-se impossível chamar o método *translation* na classe *Protein*. Torna-se assim mais fácil para um utilizador posteriormente utilizar o módulo pois deixa de necessitar de conhecimento prévio da matéria, no sentido em que cada classe apenas deixa o utilizador usá-la de formas que esta o permita. Com a implementação de heranças e diversas classes deixa de ser necessária a existência de um atributo *tipo de sequência*, na medida em que cada classe representa o tipo designado.

Após uma deliberação sobre as vantagens de cada modelo, e pelas vantagens demostradas pelo modelo de classes e heranças, optei pela implementação deste último.

Um diagrama de classes simplificado do modelo implementado pode ser visualizado na figura 1.

Através da análise do diagrama na figura 1 podemos visualizar diversas classes inter-relacionados. Explicando brevemente cada uma delas temos:

1. ***Seq***: Classe base abstrata que representa uma sequência biológica. Serve de superclasse à totalidade das outras classes. Implementa funcionalidades genéricos como: a frequência dos símbolos de uma classe, leitura e escrita em ficheiros, entre outras.

2. ***NucleotideChain***: Classe abstrata que implementa as funcionalidades de uma cadeia de nucleotídeos, isto é, funcionalidades comuns quer ao RNA quer ao DNA.

3. ***Dna***: Representa a implementação do DNA incluindo assim funcionalidades exclusivas a esta sequência biológica.

4. ***Rna***: Representa a implementação do RNA incluindo assim funcionalidades exclusivas a esta sequência biológica.

5. ***Protein***: Representa a implementação das Proteínas incluindo assim funcionalidades exclusivas a estas sequências biológicas.

Torna-se ainda importante acrescentar que a classe ***BioSeq***, não presente no diagrama, funciona como uma

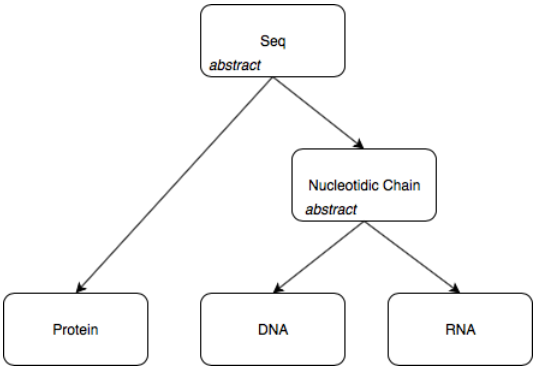

Figura 1. Diagrama de Classes implementadas

fábrica[2] de sequências biológicas na medida em que é através desta que são chamados os construtores que nos permitem gerar instâncias de *Dna*, *Rna* e *Protein*.

Para uma descrição mais detalhada das funcionalidades de cada classe, foi realizada a documentação da totalidade do código.

III. RESULTADOS

Foram implementadas as funcionalidades pedidas na sua totalidade. Cada funcionalidade e respetiva localização no código é apresentada de seguida, bem como breves explicações quando necessário:

Construtor de classe com biótipo: através do método estático *create_bio_seq(seq, seq_type)* presente na classe *BioSeq* que, passando como argumento uma sequência e um tipo de sequência (de entre os tipos válidos ou uma exceção é lançada), cria e retorna um objeto correspondente.

Métodos de visualização da classe: nas três classes que representam os biótipos em questão, foi *overriden* o método *__str__()*.

Validação do tipo de sequências: a classe base *Seq* declara um método abstrato *validade()* forçando assim as suas classes filhas não abstratas a implementar este método.

Frequência de símbolos: método implementado na classe abstrata *Seq* com a assinatura *freq_symbols()*

GC Content: métodos implementados na classe abstrata *NucleotideChain* com as assinaturas *gc_percent()* e *gc_percent_sub_seq(k)*. Estes diferem no facto do segundo aceitar um parâmetro k, de forma a calcular a percentagem de GC em subsequências de tamanho k pertencentes à sequência.

Reverse Complement: método implementado na class abstrata *NucleotideChain* com a assinatura *reverse_complement()*. Este método faz uso do atributo abstrato *switcher*, atributo este que deve ser implementado pelas classes filhas *Rna* e *Dna*. O *switcher* é um dicionário que guarda um mapeamento entre uma base azotada e o seu complemento. As chaves deste dicionário são ainda usadas para a validação das sequências nas respetivas classes.

Transcription: método da implementado na classe *Dna* com a assinatura *transcription()*. Retorna um objeto *Rna*.

Translation: funcionalidade implementada na classe *Dna* com a assinatura *translate(ini_pos=0)*. Aceita um argumento opcional a indicar a posição onde deve começar a translação do DNA. Retorna um objeto *Protein*.

Find all ORFs (by minimum size): funcionalidade implementada na classe *Dna* com a assinatura *all_orfs(minsize=0)*. Aceita um argumento opcional que indica o tamanho mínimo das sequências retornadas. Retorna uma lista de objetos *Protein*.

Pretty-printing da informação da classe: funcionalidade implementada na totalidade das classes não abstratas, através da assinatura *pretty_print()*

Leitura a partir de um ficheiro Fasta (das sequências): implementado através de um método estático na classe *BioSeq*, através da assinatura *read_fasta_file(file_name)*. O dicionário retornado contém nas chaves o descritor das sequências e nos valores as respetivas sequências. Para criar objetos do tipo correspondente os valores devem ser passadas ao método estático *create_bio_seq(seq, seq_type)*, em conjunto com o respetivo tipo.

Função Load e Save que guarda e lê em ficheiro o estado do objeto: Função load implementada sob a forma do método estático *load(file_name*, na classe *BioSeq*, que recebe como argumento o nome do ficheiro e retorna o objeto que foi carregado. A função save encontra-se implementada na superclasse abstrata *Seq* com a assinatura *save(file_name)* que permite guardar o objeto em questão num ficheiro com o nome fornecido. De destacar que no conjunto de testes desta funcionalidade foi utilizado um *fasta file* que representa *TP53 tumor protein p53 [Homo sapiens (human)]* [1].

A. Funcionalidades Extra

Adicionalmente, foi ainda implementado o méotodo *rev_transcription()* na classe *Rna*, que permite realizar a transição reversa do RNA. Retorna um objeto *Dna*.

Todas as funcionalidades foram ainda testadas atingindo perto de 95% de coverage (apenas os métodos *pretty_print()* não foram testados).

Foi também realizada documentação da totalidade do código.

IV. COMENTÁRIOS E CONCLUSÕES

Acredito que o meu conhecimento sobre sequências biológicas foi consideravelmente aprofundado. De facto, a área da bioinformática que manipula sequências biológicas aborda temáticas bastante atuais, contribuindo assim ainda mais para o interesse no desenvolvimento do projeto.

Em suma, encontro-me satisfeito com o resultado final, tendo implementado a totalidade das funcionalidades requisitadas. Considero também que o trabalho desenvolvido, pela estrutura que apresenta, revela-se uma base sólida para possíveis extensões e desenvolvimentos sobre esta.

V. BIBLIOGRAFIA

[1] P53 tumor protein p53 [Homo sapiens (human)], https://www.ncbi.nlm.nih.gov

[2] Factory pattern, https://refactoring.guru/design-patterns/factory-method