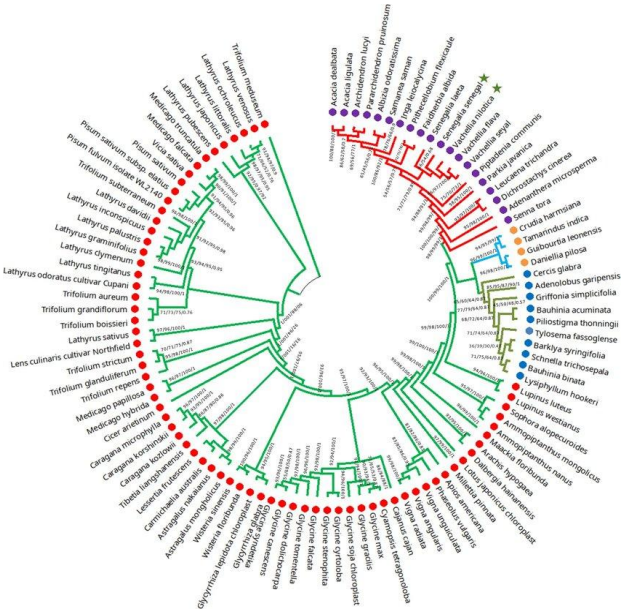
MC102 - Algoritmos e Programação de Computadores

Lab 05

Data da Primeira Chance: 24 de abril de 2023
Peso: 2

A biologia evolutiva e a genômica comparativa são importantes campos de investigação na biologia. Neles, os pesquisados utilizam-se de ferramentas (como as árvores filogenéticas) para entender a origem e a evolução de características particulares de entidades biológicas, como a presença ou ausência de um gene, ou a aquisição de uma função biológica específica. As árvores filogenéticas também são usadas na identificação de relacionamentos filogenéticos complexos entre diferentes grupos de organismos e para reconstruir a história evolutiva de todo o reino vivo.

As árvores filogenéticas são diagramas que representam as relações evolutivas entre diferentes organismos. Elas são baseadas em comparações de características morfológicas, comportamentais, fisiológicas e, cada vez mais comum, em informações genéticas e genômicas: sequências de DNA ou proteínas de diferentes espécies. Essas sequências podem ser comparadas e alinhadas para identificar regiões conservadas e divergentes, usadas para inferir a relação evolutiva entre as espécies.



Árvore filogenética construída sobre todo o conjunto de dados genoma. Fonte: Asaf et al., 2019. DOI: 10.1371/journal.pone.0225469

Algoritmos de alinhamento de sequência são empregados no alinhamento e comparação das sequências de DNA. Estes procuram as regiões de sobreposição (ou "*matches*") entre duas sequências e as alinham, de forma que as bases correspondentes fiquem em colunas verticais. As diferenças (ou "*mismatches*") são indicadas com símbolos para indicar a falta de correspondência.

Neste trabalho, você deverá implementar uma calculadora de genoma. Logo no início, será informado o genoma de referência, representado por uma *string* contendo os caracteres C, G, T e A. Será informado, em seguida, uma sequência de operações a serem realizadas sobre a cadeia de referência, que devem ser implementadas como **funções** no seu programa.

Seu programa deve possibilitar a execução das seguintes funções:

- **reverter:** dado os índices *i* e *j*, reverter a subsequência $[i, j]$ (de *i* até *j*, inclusive) do genoma atual. Por exemplo: Seja i o índice do primeiro C e $j = i + 3$, o genoma…**CGTAGT**… será revertido para …**ATGCGT**…
- **transpor:** transpor o genoma atual considerando os índices *i*, *j*, *k, isto é, trocar a posição da subsequência iniciada em i e terminada em j com a subsequência iniciada em* $j + 1$ *e terminada em k*. Por exemplo: seja $i = 2$, $j = 5$, $k = 9$, então o genoma **CTCGTAAGCTGA** será transposto para **CTAGCTCGTAGA**.
- **combinar:** combinar o genoma atual com um novo genoma informado, inserindo esse na i-ésima posição do atual. Por exemplo: seja **ATGCGT** o genoma atual, **CTAG** o genoma informado e $i = 3$, o genoma atual será atualizado para **ATGCTAGCGT.**
- **concatenar:** adicionar ao final do genoma atual um novo genoma informado. Por exemplo: seja **ATGCGT** o genoma atual, **CTAG** o genoma informado, o genoma atual será atualizado para **ATGCGTCTAG**.
- **remover:** remover a subsequência $[i, j]$ do genoma atual considerando dois índices *i* e *j* informados. Por exemplo: seja o genoma = **GCGTAT**, $i = 2$ e $j = 4$, então o genoma resultante será **GCT**.
- **transpor_e_reverter:** transpor o genoma considerando os índices *i*, *j*, *k* e, em seguida, o reverter considerando os índices *i* e *k*. Por exemplo: seja $i = 2, j = 5, k = 9$, então o genoma **CTCGTAAGCTGA** será transposto para **CTAGCTCGTAGA** e imediatamente revertido para **CTATGCTCGAGA**.
- **buscar:** dado um genoma de busca, exibir na tela o número de vezes em que ele ocorre no genoma atual. O genoma atual não deve sofrer alterações. Por exemplo: `buscar **TCGA** em **CTCGACTCGAGA** deve retornar duas ocorrências (exibir "2"). Porém, você deve se atentar a não considerar as sub-ocorrências: uma busca por **CCC** em **ACCCCT**, por exemplo, deve retornar exatamente uma única ocorrência (exibir "1").
- **buscar_bidirecional:** dado um genoma de interesse, buscar quantas vezes ele ocorre no genoma atual ou em sua reversão. O genoma atual não deve sofrer alterações.
- **mostrar:** exibir o genoma atual para o usuário.

Finalmente, seu programa deve finalizar ao receber o comando "sair".

Entrada

A entrada do seu programa consiste em uma *string*, representativa do genoma de referência, e uma sequência de comandos a serem executados sobre o genoma. Estes são separados por quebras de linha (o caractere *n*), onde cada operação descreve a função a ser executada e seus parâmetros desejados.

A tabela abaixo descreve os comandos que serão entregues ao programa, seus parâmetros e um exemplo de como eles seriam apresentados:

Funções e parâmetros	**Parâmetros**	**Exemplos**
`reverter`	`i:int j:int`	`reverter 2 5`
`transpor`	`i:int j:int k:int`	`transpor 0 2 4`
`combinar`	`g:str i:int`	`combinar CTAG 3`
`concatenar`	`g:str`	`concatenar TG`
`remover`	`i:int j:int`	`remover 2 4`
`transpor_e_reverter`	`i:int j:int k:int`	`transpor_e_reverter 2 5 9`
`buscar`	`g:str`	`buscar TG`
`buscar_bidirecional`	`g:str`	`buscar_bidirecional TG`
`mostrar`		`mostrar`

Além disso, algumas regras são importantes:

1. Os índices *i*, *j* e *k* serão sempre inteiros positivos;
2. Se um dos índices exceder o tamanho da palavra, o maior índice válido será considerado;
3. Nenhuma alteração é feita ao genoma atual se o início (índice *i*) da reversão, transposição ou remoção for maior do que o tamanho da palavra (corolário da regra 2);
4. Nenhuma alteração é feita ao genoma atual se o meio da transposição (índice *j*) for maior do que o tamanho total da palavra (corolário da regra 2);

Saída

Seu programa deve produzir saídas nas (e somente nas) funções **buscar**, **buscar_bidirecional** e **mostrar**. Nas duas primeiras, um número inteiro deve ser impresso na tela, indicando o número de *matches* encontrados. Por outro lado, o genoma atual completo deve ser exibido ao receber o comando **mostrar.**

Exemplos

Exemplo 1:

Entrada

```
ATCGTAGTG
reverter 2 5
mostrar
buscar TG
sair
```

Saída

```
ATATGCGTG
2
```

Exemplo 2:

Entrada

```
ATGCGT
mostrar
combinar CTAG 3
mostrar
buscar TGC
buscar_bidirecional CGT
buscar ATGCTAGCGT
remover 2 6
combinar TTCGG 0
mostrar
buscar TTCGGATC
sair
```

Saída

```
ATGCGT
ATGCTAGCGT
1
2
1
TTCGGATCGT
1
```

Exemplo 3:

Entrada

```
ATGCGT
mostrar
combinar CTAG 3
mostrar
buscar TGC
buscar_bidirecional CGT
buscar ATGCTAGCGT
sair
```

Saída

```
ATGCGT
ATGCTAGCGT
1
2
1
```

Regras e Avaliação

Todos os casos de testes estão disponíveis no seguinte link: testes Lab05. Os arquivos com a extensão ".in" contém as entradas dos testes e nos arquivos com final ".out" as saídas correspondentes. Os arquivos de textos podem ser abertos com qualquer editor de texto.

Neste laboratório, seu código-fonte será analisado e deve conter as funções mencionadas acima: **reverter**, **transpor**, **concatenar**, **remover**, **transpor_e_reverter**, **buscar**, **buscar_bidirecional** e **mostrar**.

Seu código será avaliado não apenas pelos testes do CodePost, mas também pela qualidade. Dentre os critérios subjetivos de qualidade de código, analisaremos neste laboratório: o uso e reuso apropriado de funções; o tamanho das funções; se as funções estão autocontidas; a escolha de bons nomes de funções e variáveis; a ausência de diversos trechos de código repetidos desnecessariamente. Note, porém, que essa não é uma lista exaustiva, pois outros critérios podem ser analisados dependendo do código apresentado visando mostrar ao aluno como o código poderia ser melhor. **Nesse laboratório, você não pode usar bibliotecas (isto é, o comando *import*).**

Submissão

Você deverá submeter no CodePost, na tarefa Lab 05, um arquivo com o nome `lab05.py`. Após a correção da primeira entrega, será aberta uma tarefa Lab 05 - Segunda Chance, com prazo de entrega apropriado.