SCC 0224/0606 - Estrutura de Dados II

Primeiro Projeto

Prof. Robson L. F. Cordeiro
Estagiários PAE: Fernanda T. Marana e Pedro de C. B. Ilídio Silva

16 de maio de 2022

Data de entrega: 16 de junho de 2022.

1 Introdução

A fim de criar mais familiaridade com os conceitos de programação trazidos pela disciplina e demonstrar a aplicabilidade dos novos conhecimentos em problemas científicos do mundo real, é pedida aos alunos a elaboração do projeto proposto a seguir.

O projeto é dividido em quatro tarefas envolvendo ordenação e análise de complexidade de algoritmos, que, de forma conjunta e progressiva, abrem caminho para que o aluno desenvolva uma aplicação em bioinformática. O objetivo de tal aplicação final, de forma breve, será medir a atividade de genes, humanos ou de outros organismos, a partir de dados de sequenciamento de DNA.

Dessa forma, pede-se paciência caso um objetivo prático dos problemas propostos não se faça claro à primeira vista, e espera-se que ao final do quarto problema se torne justificável ao aluno a cautela ao se desenvolver algoritmos de crítica eficiencia, e fique minimamente embasada a importância à humanidade dos métodos que buscam reduzir a complexidade computacional.

O conteúdo deste documento será organizado em duas seções principais. Em um primeiro momento, são descritas em detalhes e com excertos de pseudocódigo as quatro tarefas propostas. A segunda parte, por sua vez, é onde estarão reunidas instruções objetivas de implementação dos programas e elaboração de um relatório documentando seu desenvolvimento. Também é onde estará descrito como se espera que se demonstre aos avaliadores o trabalho realizado: são expostos os critérios a serem levados em conta

ao avaliar cada tarefa, para uma avaliação transparente e consistente.

2 Descrição dos problemas propostos

2.1 Implementação da ordenação

Para o nosso caso em específico, a melhor forma de ordenar valores numéricos é a apresentada pelo algoritmo OrdenaNumeros.

Trataremos, na verdade, um número inteiro como uma sequência de de seus dígitos constituintes, ordenando como se estivéssemos ordenando uma *string*. Ou seja, em cada passo olharemos apenas para um dígito de cada número do conjunto que iremos ordenar. Primeiro, levamos em consideração o dígito mais à direita dos números em questão, por exemplo, 385652 será visto como 2, 860 será tratado como 0, etc. Em seguida, olhamos para o segundo dígito à direita: 860 agora é tratado como 6. E assim por diante, como se estivéssemos ordenando palavras. Então ordenar [3456, 9889, 0191, 99219] considerando-se apenas o segundo dígito à direita significa obter a sequência [00191, 09889, 03456, 99219]. Notem a presença de zeros à esquerda para que os números tenham o mesmo comprimento.

O processo de ordenar os números considerando-se apenas um dígito em uma posição específica (ainda não os ordenando por completo) é realizado pela função OrdenaDigitos. O processo total de ordenação (algoritmo OrdenaNumeros) consiste então apenas em chamar a ordenação de dígitos para cada posição de dígito nos nossos números.

Por fim, para os próximos problemas precisaremos ordenar vetores com, na verdade dois números por linha. Ou seja, ordenaremos as linhas de matrizes na forma $A[n][2]$, em que apenas o primeiro número, $A[i][0]$ de cada linha i deve ser considerado para a ordenação.

Função OrdenaDigitos(A, n, *posicao*)

Entrada: A, matriz com duplas de números a serem ordenadas.
Entrada: n, número de linhas em A.
Entrada: *posicao*, posição do dígito a se considerar (10, 100, 1000, etc.).
Saída: A ordenado a partir do dígito em *posicao*.

```
// B armazenará contagens de números em A com cada dígito, de 0
   a 9, na posição posicao.
```

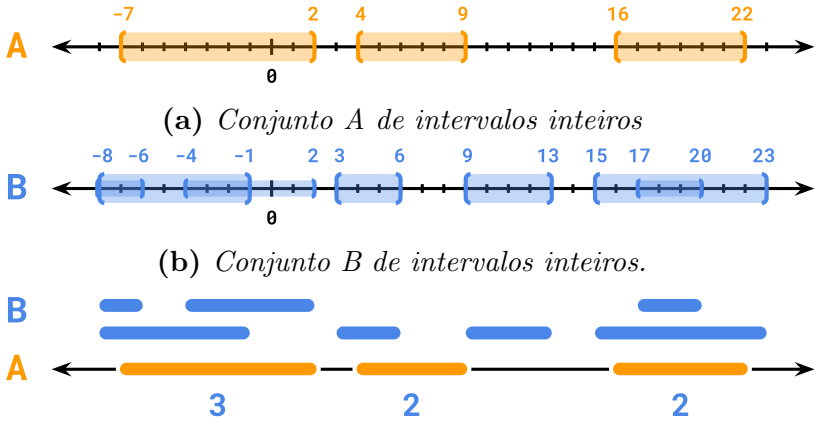
```
1  Inicializar B[10] com zeros ;
2  para i ← 0 até n − 1 faça
3  |  digito ← A[i][0]/posicao;
4  |  digito ← digito mod 10;
5  |  B[digito] ← B[digito] + 1;
6  |  i ← i + 1;
7  fim
8  para i ← 1 até 9 faça
9  |  B[i] ← B[i] + B[i − 1];
10 |  i ← i + 1;
11 fim
12 Inicializar C[n][2];
13 para i ← n − 1 até 0 faça
14 |  digito ← A[i][0]/posicao;
15 |  digito ← digito mod 10;
16 |  B[digito] ← B[digito] − 1;
17 |  C[B[digito]][0] ← A[i][0];
18 |  C[B[digito]][1] ← A[i][1];
19 |  i ← i − 1;
20 fim
21 para i ← 0 até n − 1 faça
22 |  A[i][0] ← C[i][0];
23 |  A[i][1] ← C[i][1];
24 |  i ← i + 1;
25 fim
```

Função OrdenaNumeros(A,n)

Entrada: $A[n][2]$, vetor com pares de números a serem ordenados.
Entrada: n, número de elementos em A.
Saída: A ordenado pelo primeiro elemento de cada par.

```
1 maior ← maior inteiro armazenado em A[i][0];
2 posicao ← 1;
3 enquanto (maior/posicao) > 0 faça
4 |   OrdenaDigitos (A,n,posicao);
5 |   posicao ← posicao * 10;
6 fim
```

2.2 Contagem de interseções



(a) *Conjunto A de intervalos inteiros*

(b) *Conjunto B de intervalos inteiros.*

(c) *Ilustração da contagem de interseções com conjuntos de B para cada conjunto de A.*

Figura 2.2.1: *Exemplos ilustrativos das entradas e saídas esperadas para o problema 2.2.*

Suponha dois arquivos, `A.csv` e `B.csv` (figuras 2.2.1a e 2.2.1b), cada um contendo um conjunto de intervalos na reta dos inteiros. Os intervalos são representados um por linha, por dois números inteiros que determinam seu primeiro e último valores. Esses números de início e fim são separados por um caractere de vírgula (`,`), o que dá nome ao formato `.csv` (*comma-separated values*).

Como representado na figura 2.2.1b, note que os intervalos podem ter interseções e pontos de início e/ou fim coincidentes (o tamanho dos colchetes representados varia para facilitar a visualização). Além disso, não há garantia de ordenação dos intervalos de nenhuma forma nos arquivos de entrada.

O objetivo desta seção, como ilustrado na figura 2.2.1c, é escrever um programa que imprima na saída padrão um inteiro por linha, representando, para cada intervalo

A_i do arquivo A, quantos intervalos de B intersectam A_i. Ou seja, quantos intervalos de B têm ao menos um elemento em comum com A_i, para cada A_i, na ordem cresente em função do inteiro que marca o início do intervalo.

O caso ilustrado na figura 2.2.1 é representado pelos exemplos de entrada (tabela 2.2.1) e saída (tabela 2.2.2).

`A.csv`	`B.csv`
`16,22`	`-7,2`
`4,9`	`3,6`
`17,20`	`9,13`
`-8,-1`	`-4,2`
`15,23`	`-8,-6`

Tabela 2.2.1: *Exemplo de entradas.*

`3`
`2`
`2`

Tabela 2.2.2: *Exemplo de saída esperada.*

Note que a ordenação dos intervalos em cada arquivo em função de alguma de suas extremidades permite o uso de algoritmos com menor complexidade para executar a tarefa descrita. Observe o algoritmo ContagemIntersecoes, que ordena os intervalos nos arquivos `A.txt` e `B.txt` em função de sua posição de início. Ao final do documento, será pedia sua implementação.

Função ContagemIntersecoes(
. *arquivo_A*, *arquivo_B*, *nA*, *nB*, *arquivo_saida*)

Entrada: `A.txt` e `B.txt`, não ordenados
Entrada: nA e nB, números de linhas em A e B, respectivamente.
Saída: `contagens.txt`, contagens de intervalos de B em cada intervalo de A

```
1  Inicializar A[nA][2] e B[nB][2]. Inicializar contagens[nA] com zeros.;
2  para i ← 0 até nA − 1 faça
3  |  Leia uma linha de A.txt, determinando inicio e fim do intervalo;
4  |  A[i][0] ← inicio;
5  |  A[i][1] ← fim;
6  fim
7  para i ← 0 até nB − 1 faça
8  |  Leia uma linha de B.txt, determinando inicio e fim do intervalo;
9  |  B[i][0] ← inicio;
10 |  B[i][1] ← fim;
11 fim
12 OrdenaNumeros (A, nA);
13 OrdenaNumeros (B, nB);
14 primeiro_iB ← 0;
15 para iA ← 0 até nA − 1 faça
16 |  para iB ← primeiro_iB até nB − 1 faça
17 |  |  se A[iA][1] < B[iB][0] ou A[iA][0] > B[iB][1] então
18 |  |  |  se contagens[iA] = 0 então
19 |  |  |  |  primeiro_iB ← iB;
20 |  |  |  fim
21 |  |  senão
22 |  |  |  contagens[iA] ← contagens[iA] + 1;
23 |  |  fim
24 |  fim
25 fim
26 para i ← 0 até nA faça
27 |  Escrever contagens[i] em contagens.txt.
28 fim
```

2.3 Ctrl + F

Para um arquivo de entrada `texto.txt` contendo um texto qualquer, e um outro `trechos.txt` contendo um segmento de texto por linha, seu objetivo é determinar as posições de início e fim, na string em `texto.txt`, da primeira ocorrência de cada *substring* dada por `trechos.txt`.

texto.txt	trechos.txt
Fala pro cliente que o último pull request desse SCRUM superou o desempenho na organização alfanumérico dos arrays multidimensionais. Uma frase intrusa. Desde ontem à noite, a disposição dos elementos HTML complexificou o merge dos parametros passados em funções privadas. Uma frase intrusa. Dado o fluxo de dados atual, a otimização de performance da renderização do DOM complexificou o merge de uma configuração Webpack eficiente nos builds. Explica pro Product Owner que a disposição dos elementos HTML corrigiu o bug na interpolação dinâmica de strings. Uma frase intrusa.	HTML pull request elementos HTML o Uma frase intrusa. complexificou o merge fluxo de dados atual interpolação dinâmica de strings

Tabela 2.3.1: *Exemplo de entradas. Créditos:* `https://lerolero.com`

```
201,204
30,41
191,204
7,7
134,151
206,226
299,318
524,535
```

Tabela 2.3.2: *Exemplo de saída esperada.*

A forma simples de resolver o problema é apresentada pelo algoritmo CtrlF e, para o nosso trabalho, será suficiente sua implementação.

Função CtrlF(*arquivo_texto*, *arquivo_trechos*, *arquivo_saida*)

Entrada: *arquivo_trechos*, *arquivo_texto*
Saída: *arquivo_saida*

```
1  Leia arquivo_texto e atribua a texto.
2  enquanto não chega o fim de arquivo_trechos faça
3  |  Leia trecho em arquivo_trechos.;
4  |  i ← 0;
5  |  enquanto texto[i] ≠ (fim da string) faça
6  |  |  j ← 0;
7  |  |  enquanto (trecho[j] ≠ (fim da string)) e (texto[i + j] = trecho[j])
   |  |   faça
8  |  |  |  j ← j + 1;
9  |  |  fim
10 |  |  se trecho[j] = (fim da string) então
11 |  |  |  Escreva i,i + j − 1 em arquivo_saida;
12 |  |  |  Pare o laço.
13 |  |  fim
14 |  |  i ← i + 1;
15 |  fim
16 fim
```

2.4 Aplicação em bioinformática

Nesta seção, aplica-se a um cenário do mundo real o trabalho desenvolvido para os problemas anteriores.

2.4.1 Contextualização

Desde o sequenciamento completo do genoma humano em 2003, o volume de dados genéticos disponível cresce de forma acelerada. Se anos de esforço e milhares de cientistas de diversos países foram necessários na época, hoje bastam algumas horas para que o genoma de uma espécie esteja disponível à humanidade. O desafio agora então é outro: extrair conhecimento, de forma eficiente, a partir da miríade de dados biológicos a que temos acesso.

Uma das informações que frequentemente se quer obter é o nível de atividade de cada um dos genes que nos compõem, isto é, o nível de influência que esse gene terá nos processos biológicos de que faz parte e o quão vital será o seu papel para o funcionamento adequado do organismo. Calcular esse dado será o objetivo deste problema. Como podemos medir a atividade de um gene? E de que forma isso se relaciona com sequenciamento de DNA?

Antes faz-se necessário o entendimento básico de como se dá o fluxo de informação em uma célula.

Um genoma é o conjunto de toda a sequência de DNA de um ser vivo. Ele nada mais é que uma longa sequência de quatro moléculas, representadas pelas letras `A`, `C`, `G` e `T`, que se apresentam ligadas linearmente umas às outras como miçangas num fio, repetidas e alternadas em arranjos ainda hoje não totalmente compreendidos. Essa cadeia de `A`, `C`, `G` e `T` é o que chamamos de DNA. Ela pode ser representada como uma longa string semelhante a `ACGTGCTAGTGTTATTATTTTGCGGCGAGCATTCGATCGA`..., se estendendo por bilhões (!) de caracteres. Uma cópia dessa sequência (do genoma), é armazenada em cada uma das nossas células.

Os genes, por sua vez, nada mais são que trechos específicos, com geralmente alguns milhares de "caracteres", do enorme genoma. Eles são reconhecidos e interpretados pela célula para a fabricação de proteínas (figura 2.4.1).

Sobre as proteínas, basta termos em mente que são o objetivo final de um gene. São complexas e massivas moléculas responsáveis por quase todas as reações químicas que fazem a vida como conhecemos. De forma simplificada, cada gene é responsável pela produção de uma proteína. Isto é, em cada gene está contida a informação necessária para que a célula produza uma proteína.

O último detalhe é que esse processo de tradução de um gene não ocorre de forma direta. Para que a célula tenha mais controle sobre a quantidade de proteínas produzida

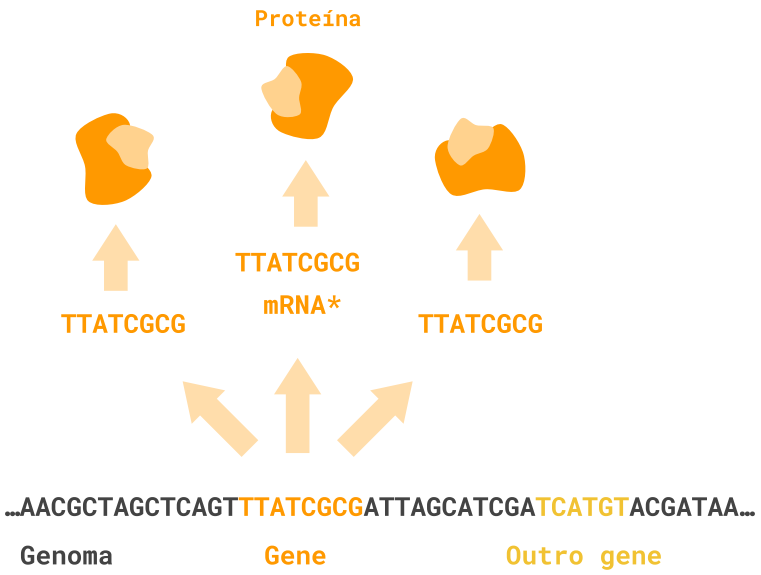


Figura 2.4.1: *Fluxo da informação genética.*

e possa paralelizar a tradução sem encher o genoma, já comprido, com vários genes iguais, ela constantemente transcreve e degrada várias cópias de cada gene, trechos das mesmas[1] letras "`ACGTCGT`..."que se dispersam por toda a célula. São essas cópias que vão ser lidas e traduzidas para a produção de proteínas, cada uma representando um gene e, portanto, produzindo a proteína correspondente a ele (figura 2.4.2).

O ponto-chave é que a quantidade de cópias que a célula transcreve de cada gene não é a mesma. Genes mais ativos são justamente aqueles cujas cópias são transcritas em maior quantidade, produzindo, portanto, mais proteínas e, assim, influenciando as reações químicas da célula de forma mais significativa (figura 2.4.2).

Esses transcritos gênicos são chamados de mRNAs (RNAs mensageiros), e a contagem de quantos mRNAs de cada gene há numa célula nos fornece uma métrica para o nível de atividade de cada gene.

Com a tecnologia moderna, conseguimos coletar e ler as sequências de caracteres dos mRNAs presentes nas células em um determinado momento. Essa leitura de caracteres desse tipo é o que chamamos de sequenciamento.

Identificando posteriormente que trecho do genoma do organismo aquele mRNA representa e cruzando a posição do genoma que foi associada ao mRNA com as posições sabidas de cada gene no genoma, identificamos de que gene cada mRNA é proveniente, e podemos então fazer a contagem de mRNAs.

[1]Na verdade, o `T` é substituído por uma outra molécula, chamada `U`, mas desconsidere essa informação para as coisas ficarem mais simples.

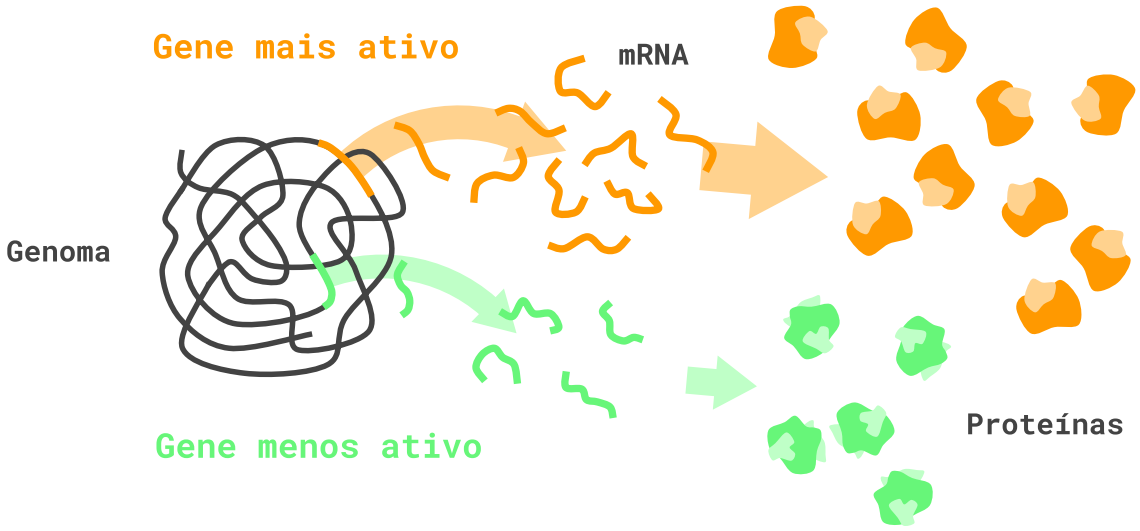


Figura 2.4.2: *Representação do conceito de "nível de atividade" de um gene.*

Um fator que vai dificultar a nossa análise posterior é que o enorme volume de "caracteres" numa célula faz com que o sequenciamento precise ser altamente paralelo, o que leva os cientistas sequenciadores a optar frequentemente por fragmentar os mRNAs em pedaços de algumas dezenas de letras, sequenciando, na verdade, esses fragmentos (figura 2.4.3). Ou seja, não teremos as sequências dos transcritos em si, mas sim as sequências de diversos fragmentos desses transcritos. Faremos então não a contagem de mRNAs, mas sim de fragmentos deles para cada gene, que ainda se trata de um indicador útil do nível de atividade gênica.

O objetivo do problema é então medir a atividade de um conjunto de genes especificados. Em três arquivos (tabela 2.2.1), serão dadas:

- Parte da sequência do genoma humano, apenas a região necessária (você deve desconsiderar as quebras de linha);
- As localizações (início e fim) de genes nessa região do genoma, um gene por linha;
- As sequências de fragmentos resultantes de sequenciamento de mRNA, um fragmento por linha.

Você deve então utilizar o código desenvolvido nos problemas anteriores para obter a contagem de fragmentos por gene, como ilustrado na figura 2.4.4 e demonstrado pelo algoritmo ContagemLeituras.

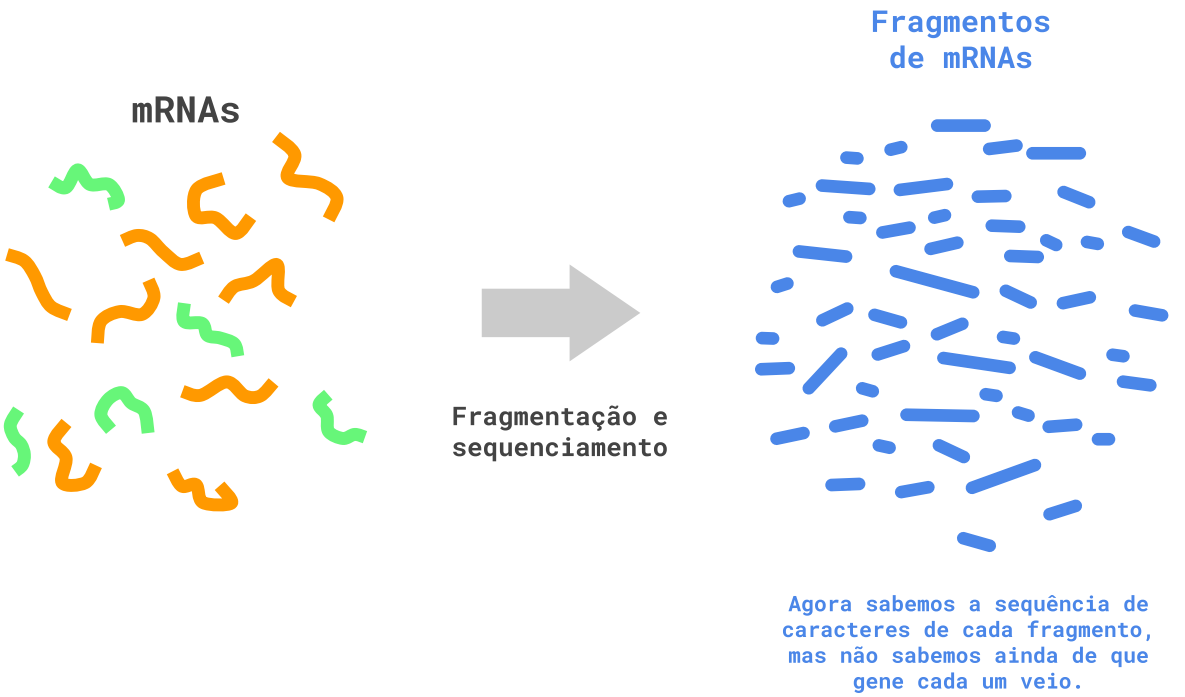


Figura 2.4.3: *Para sequenciar mRNAs, frequentemente se opta por fragmentar as cadeias antes.*

Figura 2.4.4: *Ilustração da contagem de fragmentos de mRNA por gene, a saída requerida no presente problema.*

genoma.txt	pos_genes.csv	fragmentos.txt
AGCTGTACGTACGTCGTTGTACGTAC	3305,4205	GCTTGTGG
ACTGGTATGGTCAGTAGGGTATGTAT	22352,32253	CGTAGCGCGTATACGG
ATGTCATCATGCATGATTCATCCATC	70235,80300	GCTTATG
CGCATTATATGATGATAATTATTTAT	33233,35023	AGGATCGATTACGTAGC
AGCATGACTGACTGCAAATGACTGAC	48712,49388	AGCTTTTTTTAGCGG
CGGAGCGGGCTGCATGAAGCGGGCGG		AGTCGATACGTG
GCATGATTTTCGGAGCTTGATTGATT		AACTGCCGTA
CTGCATGACTGCATGACCATGAATGA		AGCTTAGCCGATCGAT
ATGACGTCATGTCGAAAACGTCCGTC		CGTAGACGTAGC
CGTGTCATGACGTTGTGGTCATTCAT		[...]
CTGCATGCGGCGCGGGCCATGCATGC		
AGGTACTACACTGCTGTTACTAACTA		
TGACTGCATGATTTTACCTGCATGCA		
[...]		

Tabela 2.4.1: *Exemplo de entradas (meramente ilustrativo).*

34
754
12
1231

Tabela 2.4.2: *Exemplo de saída esperada. `[...]` indica que há mais linhas equivalentes no arquivo, não mostradas aqui.*

Função ContagemLeituras(
. *arquivo_genoma*,
. *arquivo_pos_genes*,
. *arquivo_fragmentos*,
. *arquivo_pos_fragmentos*,
. *n_genes*,
. *n_fragmentos*,
. *arquivo_saida*,
)

Entrada: `genoma.txt`
Entrada: `pos_genes.csv`
Entrada: `fragmentos.txt`
Entrada: *n_genes*, número de genes em `pos_genes`.
Entrada: *n_fragmentos*, número de fragmentos de mRNA em `fragmentos.txt`.
Saída: `atividade_genica.txt`, arquivo de saída.

```
1  CtrlF (
2  .    arquivo_genoma,
3  .    arquivo_fragmentos,
4  .    arquivo_pos_fragmentos,
5  );
6  ContagemIntersecoes (
7  .    arquivo_pos_genes,
8  .    arquivo_pos_fragmentos,
9  .    n_genes,
10 .    n_fragmentos,
11 .    arquivo_saida,
12 );
```

2.4.2 Análise empírica da complexidade

Para este último problema, pede-se que se efetue a análise empírica da complexidade do programa desenvolvido. Havendo três variáveis principais a se analisar, deverão ser elaborados três gráficos do tempo de necessário para rodar o programa, em função de cada uma delas.

De forma mais específica, serão providos nove arquivos, sendo três versões para cada arquivo de entrada da função ContagemLeituras, identificadas com os sufixos `_grande`, `_medio` e `_pequeno`. Os sufixos dizem respeito ao comprimento do genoma, à quantidade de genes e à quantidade de fragmentos, respectivamente quando acompanharem os arquivos `genoma_*.txt`, `pos_genes_*.csv` e `fragmentos_*.txt`.

1. Fixar os arquivos `genoma_grande.txt` e `pos_genes_grande.txt`, rodando o programa para as seguintes combinações de entradas:
   - `(genoma_grande.txt, pos_genes_grande.csv, fragmentos_pequeno.txt)`
   - `(genoma_grande.txt, pos_genes_grande.csv, fragmentos_medio.txt)`
   - `(genoma_grande.txt, pos_genes_grande.csv, fragmentos_grande.txt)`
2. De forma semelhante, agora fixar os arquivos `genoma_grande.txt` e `fragmentos_grande.txt`, coletando os tempos de execução para combinações com os diferente arquivos `pos_genes_*.txt`:
   - `(genoma_grande.txt, pos_genes_pequeno.csv, fragmentos_grande.txt)`
   - `(genoma_grande.txt, pos_genes_medio.csv, fragmentos_grande.txt)`
   - `(genoma_grande.txt, pos_genes_grande.csv, fragmentos_grande.txt)`
3. Por fim, fixar os arquivos `pos_genes_grande.csv` e `fragmentos_grande.txt`, coletando os tempos de execução para combinações com os diferente arquivos `genoma_*.txt`:
   - `(genoma_pequeno.txt, pos_genes_grande.csv,fragmentos_grande.txt)`
   - `(genoma_medio.txt, pos_genes_grande.csv, fragmentos_grande.txt)`
   - `(genoma_grande.txt, pos_genes_grande.csv, fragmentos_grande.txt)`

Note que a combinação com todos os arquivos grandes aparece em todos os casos, vocês só precisam coletar o tempo de execução uma vez para essa combinação de entradas.

Apresentem um gráfico para cada uma das três etapas numeradas anteriormente, totalizando três gráficos, cada um com três pontos representando a variável em questão (número de genes, comprimento do genoma ou número de fragmentos) em função do

Valor	Arquivo
30000	fragmentos_grande.txt
3000	fragmentos_medio.txt
300	fragmentos_pequeno.txt
1000000	genoma_grande.txt
100000	genoma_medio.txt
10000	genoma_pequeno.txt
10000	pos_genes_grande.csv
1000	pos_genes_medio.csv
100	pos_genes_pequeno.csv

Tabela 2.4.3: *Valores numéricos das variáveis do problema, para cada arquivo fornecido.*

tempo de execução. o valor das variáveis em cada arquivo é apresentado pela tabela 2.4.3.

3 Relatório e avaliação

Pede-se que os alunos se dividam em duplas ou trios para o desenvolvimento do presente projeto e a elaboração de um relatório documentando sua realização. Os códigos devem ser escritos na linguagem C, na versão que preferirem. Instruções detalhadas de cada passo serão dadas nas próximas subseções.

O relatório em formato PDF deve ser entregue juntamente com os arquivos `.c` e `.h` e quaisquer outros gerados pelos alunos no decorrer do projeto, todos reunidos em um arquivo compactado da forma que preferirem, com nome estritamente composto pelos números USP dos autores separados por *underlines* ("_"). A entrega deve ser feita através da plataforma de Atividades do Tidia-ae (`https://ae4.tidia-ae.usp.br/portal`).

3.1 Critérios de avaliação

Abaixo são apresentadas cada tarefa pedida para a elaboração do relatório. Cada item passível de avaliação terá seus critérios avaliativos enumerados abaixo dele. Os critérios apresentados como "Código organizado"se desdobram nos itens apresentados em 3.1.1, que somados possuem o mesmo valor de um dos demais critérios.

A nota final corresponde à fração de critérios da seção 3.1.2 cumpridos pelo grupo multiplicada por dez. Ou seja, cada critério contibui igualmente, valendo

$$\textit{valor de cada critério} = \frac{10}{\textit{quantidade de critérios em 3.1.2}}$$

pontos na nota final cada um.

Pontos adicionais podem ser obtidos cumprindo-se os critérios em 3.1.3, cada um com o mesmo valor dos critérios na seção 3.1.2. Contudo, os pontos extras obtidos só serão adicionados à nota caso a nota sem os pontos extras seja igual ou superior a 6.

Observe que a nota máxima a ser obtida neste projeto é 10.

A detecção de plágio via ferramenta automatizada e conferência manual imediatamente zera a nota dos grupos envolvidos.

3.1.1 Da organização do código

C1. São empregados espaçamento e identação que facilitam a leitura e entendimento do código do programa.

C2. São empregados comentários pertinentes e elucidativos ao longo do código, descrevendo cada função escrita.

C3. Quando pertinente, é empregada de forma correta a alocação dinâmica de memória.

C4. O conceito de TAD (Tipo Abstrato de Dados) é visivelmente utilizado no desenvolvimento do programa.

C5. É dedicada uma seção breve do relatório indicando quais arquivos de código correspondem ao programa solicitado.

C6. Nesta mesma seção, são indicados eventuais desvios do código implementado em relação às propostas sugeridas no presente documento, explicando, por exemplo, decisões por determinadas estruturas de dados, bibliotecas ou tipos de variáveis.

3.1.2 Da realização do projeto

1. Devem estar contidas na introdução do relatório uma breve descrição do projeto e uma explicação da organização de seu conteúdo.

   1.1 ☐ A organização do trabalho é claramente explicada.

   1.2 ☐ São providos de forma clara os objetivos de cada etapa do projeto.

2. Façam a análise teórica da complexidade assintótica do algoritmo OrdenaNumeros, de ordenamento de sequências. Estimem a complexidade no melhor caso, caso médio e pior caso. Em cada caso, justifiquem o raciocínio por trás das conclusões. Caso lhes sejam úteis, utilizem os números das linhas que aparecem nos algoritmos definidos aqui.

   2.1 ☐ São obtidos os valores corretos para a complexidade nos casos médio,

2.2 ☐ melhor e

2.3 ☐ pior.

2.4 ☐ São apresentadas justificativas corretas, completas e coerentes com os valores obtidos, para os casos médio,

2.5 ☐ melhor e

2.6 ☐ pior.

3. Implementem o algoritmo de ordenação proposto OrdenaNumeros.

   3.1 ☐ O programa implementado funciona de forma correta.

   3.2 ☐ Código organizado (vide 3.1.1).

4. Expliquem em alguns parágrafos, com suas palavras, o funcionamento do algoritmo de ordenação proposto.

   4.1 ☐ A explicação é correta e compreensível.

5. Façam a análise empírica da complexidade do algoritmo OrdenaNumeros.

   5.1 ☐ A análise é efetuada de forma correta.

6. Implementem o algoritmo ContagemIntersecoes para a contagem de interseções, ou outro algoritmo equivalente que você ache mais interessante.

   6.1 ☐ O programa implementado funciona de forma correta.

   6.2 ☐ Código organizado (vide 3.1.1).

7. Implementem o algoritmo CtrlF para determinar as posições de *substrings*, ou outro algoritmo com o mesmo fim que você ache mais interessante.

   7.1 ☐ O programa implementado funciona de forma correta.

   7.2 ☐ Código organizado (vide 3.1.1).

8. Façam um programa para executar o procedimento ContagemLeituras.

   8.1 ☐ O programa implementado funciona de forma correta.

   8.2 ☐ Código organizado (vide 3.1.1).

9. Utilizem o programa para contabilizar fragmentos de mRNA em cada gene a partir dos arquivos que acompanham essa descrição de projeto.

10. Efetuem a análise empírica de complexidade, como explicado na seção 2.4.2.

11. Apresentem a ordem determinada para a complexidade em função de cada uma das três variáveis (cada um dos três gráficos). Uma tabela com os valores empíricos calculados e gráficos demonstrando as medidas devem também estar presentes.

    11.1 ☐ A complexidade em função do comprimento do genoma é estimada de forma correta.

    11.2 ☐ A complexidade em função da quantidade de genes é estimada de forma correta.

    11.3 ☐ A complexidade em função da quantidade de fragmentos de mRNA é estimada de forma correta.

    11.4 ☐ Estão presentes os três gráficos solicitados, de forma coerente com as complexidades mencionadas, cada um com os pontos experimentais e a reta que melhor se aproxima deles.

    11.5 ☐ São providos alguns parágrafos apresentando em texto os resultados.

    11.6 ☐ São providas instruções claras de execução dos programas desenvolvidos, que permitam de forma simples reproduzir os resultados que vocês obtiveram.

12. Façam críticas e propostas de melhorias aos métodos e algoritmos apresentados aqui.

13. Concluam o trabalho, pontuando as maiores dificuldades enfrentadas durante seu desenvolvimento, como foi sua experiência em realizá-lo ou quaisquer outros pensamentos sobre o projeto que queiram expressar.

3.1.3 Dos pontos extras

14. Faça a análise teórica da complexidade do algoritmo ContagemLeituras.

    14.1 ☐ São obtidos os valores corretos para a complexidade nos casos médio, melhor e pior.

    14.2 ☐ Justificativa completa e coerente com o valor obtido.

15. Faça a análise teórica da complexidade do algoritmo ContagemIntersecoes.

    15.1 ☐ São obtidos os valores corretos para a complexidade nos casos médio, melhor e pior.

    15.2 ☐ A justificativa fornecida em cada caso é correta, completa e coerente com

o valor obtido.

16. É possível implementar algoritmos com mesma função de ContagemIntersecoes mas complexidade $O(nA + nB)$ em todos os casos. Nos descrevam, da forma que preferirem, um algoritmo de tal ordem.

    16.1 ☐ O algoritmo proposto de fato é $O(nA + nB)$.

    16.2 ☐ A explicação proposta é correta, completa e coerente com o valor estimado.

    16.3 ☐ A complexidade é demonstrada empiricamente.

17. Faça a análise teórica da complexidade do algoritmo CtrlF.

    17.1 ☐ São obtidos os valores corretos para a complexidade nos casos médio, melhor e pior.

    17.2 ☐ A explicação proposta é correta, completa e coerente com o valor estimado.

18. Proponha uma alternativa ao algoritmo CtrlF, de menor complexidade. É permitido que ele funcione apenas no nosso caso, em que os textos são cadeias de DNA.

    18.1 ☐ O algoritmo proposto de fato é de ordem inferior ao apresentado aqui.

    18.2 ☐ A explicação proposta sobre o funcionamento do algoritmo é correta, completa e coerente com o valor estimado.

    18.3 ☐ A complexidade é demonstrada empiricamente.