PONTIFÍCIA UNIVERSIDADE CATÓLICA DO RIO GRANDE DO SUL
ESCOLA POLITÉCNICA
CURSO DE BACHARELADO EM ENGENHARIA DE SOFTWARE
ALGORITMOS E ESTRUTURA DE DADOS II

CAROLINA MICHEL FERREIRA
MATEUS CAMPOS CAÇABUENA

TRABALHO 1

Porto Alegre
2023

1. Qual o problema sendo resolvido

Cientistas descobriram que o DNA de seres alienígenas é formado por 3 bases (D, N e A) e que este DNA é deteriorado ao longo do tempo. A deterioração funciona da seguinte maneira: duas bases diferentes se unem, formando uma nova base. A partir disto, surge o questionamento sobre qual a menor cadeia possível de ser obtida após as mutações e qual o seu tamanho. Desta forma, buscando resolver este problema, foi desenvolvido um algoritmo que lê as cadeias fornecidas e indica qual a menor cadeia possível e o seu tamanho.

2. Como o problema foi modelado

Para iniciar a desenvolver o algoritmo, buscamos compreender a lógica do problema. Chegamos à seguinte conclusão: a análise da cadeia sempre começa pelas duas bases mais à esquerda e, se estas bases forem diferentes, devem ser fundidas em uma terceira base (diferente das bases “mães”) ao final da cadeia. Com isto em vista, partimos para a modelagem do algoritmo. Optamos por armazenar as cadeias em estruturas LinkedList do próprio Java, criando uma classe chamada “Lista” que é responsável por inserir os elementos na lista e realizar a deterioração dos elementos. Criamos também uma classe chamada “ArchiveReader” que lê os arquivos de entrada, cria uma lista e insere os elementos do arquivo na lista. Por fim, a classe “Controller” controla a execução da aplicação e a classe “App” inicia a execução.

3. Processo de solução

Inicialmente, optamos por criar a nossa própria LinkedList para solucionar o problema. Obtivemos sucesso com esta solução, contudo o algoritmo estava muito extenso e com um alto tempo de execução. Buscando resolver isso, abandonamos a nossa LinkedList e migramos para a LinkedList do Java. A partir de então, desenvolvemos o método “deteriorar()” e “compare(char a, char b)” na classe ‘Lista”.

O método “deteriorar()” inicia verificando se a lista possui elementos para serem deteriorados. Após isso, criamos uma variável “indice” que controla em qual posição da lista estamos. Dentro de um laço *while,* que para quando o índice for maior ou igual que o tamanho da lista menos um, verificamos se o elemento da posição índice é diferente do elemento da posição índice + 1. Se for, comparamos os elementos com o método “compare(char a, char b)” que retorna qual carácter será adicionado na lista. Após isso, removemos os elementos da posição índice, adicionamos o novo elemento na última posição e zeramos novamente o índice. O índice é zerado para que o processo de comparação reinicie pelas primeiras bases da cadeia. Se os elementos das posições índice e índice + 1 forem iguais, apenas atualizamos o valor do índice (indice++) para analisar os elementos da próxima posição.

4. Resultados

A seguir, estão os resultados obtidos em cada execução do algoritmo.

Arquivo ‘ct_10’:

```
Cadeia deteriorada: NN
Tamanho da cadeia: 2
```

Arquivo ‘ct_100’:

```
Cadeia deteriorada: N
Tamanho da cadeia: 1
```

Arquivo ‘ct_1000’:

```
Cadeia deteriorada: DDD
Tamanho da cadeia: 3
```

Arquivo ‘ct_10000’:

```
Cadeia deteriorada: DD
Tamanho da cadeia: 2
```

Arquivo ‘ct_10004’:

```
Cadeia deteriorada: DDDDDDDD
Tamanho da cadeia: 8
```

Arquivo ‘ct_100000’:

```
Cadeia deteriorada: A
Tamanho da cadeia: 1
```

Arquivo ‘ct_1000000’:

```
Cadeia deteriorada: N
Tamanho da cadeia: 1
```

Arquivo ‘ct_2000000’:

```
Cadeia deteriorada: NNNN
Tamanho da cadeia: 4
```

5. Conclusões

Ao longo do processo de solução, enfrentamos desafios iniciais ao criar nossa própria LinkedList, o que resultou em um algoritmo extenso e com alto tempo de execução. No entanto, após a transição para a LinkedList fornecida pela linguagem Java, pudemos otimizar significativamente o algoritmo. Desenvolvemos o método "deteriorar()" na classe "Lista", que itera pela lista, identificando bases diferentes e aplicando a lógica de fusão. No geral, esse projeto resultou em um algoritmo eficaz para determinar a menor cadeia possível após as mutações de DNA e calcular seu tamanho. Ele contribui para a compreensão da deterioração do DNA alienígena e pode ter aplicações em estudos relacionados à genética e à evolução das espécies. Além disso, a abordagem de utilizar estruturas de dados eficientes, como LinkedList, demonstra a importância da escolha adequada de ferramentas de programação para a resolução de problemas complexos.