DCC012

Estrutura de Dados II

Turma A

Relatório

Trabalho Prático
Parte 4

Equipe:

Matrícula	**Nome:**
201735038	André Luís Braga Dutra
201865176A	Hyago Assis de Novais Oliveira
201965237A	Mara de Lemos Gomes
201665219AC	Rômulo Luiz Araujo Souza Soares

Grupo ID:	**06**

Prof. José J. Camata

Data de Entrega: 09/09/2021

SUMÁRIO

INTRODUÇÃO **3**

DESENVOLVIMENTO DO SOFTWARE **3**
Compilação e Execução 3
Divisão do Trabalho 4
Descrição das Estruturas Implementadas 4

EXPERIMENTOS COMPUTACIONAIS **7**
Descrição das Instâncias e Conjuntos de Parâmetros 7
Ambiente Computacional do Experimento 7
Resultados e Análise 8
Resultados Casamento de Padrão 8
Resultados Compressão de Huffman 11

CONCLUSÕES **12**

REFERÊNCIAS **13**

1. INTRODUÇÃO

Esta etapa do trabalho tem como objetivo analisar e comparar os métodos de casamento de padrão (força-bruta, KMP e BFH) e o método de compressão de Huffman. Vamos estar utilizando de padrões de sequências de DNA, gerador pelo site bioinformatics [1], para executar os estudos.

2. DESENVOLVIMENTO DO SOFTWARE

A aplicação foi implementada em linguagem C/C++, orientada a objetos.

2.1. Compilação e Execução

O código é compilado e executado pelo terminal. Para executar o usuário precisa informar o caminhos dos arquivos dnaX.txt e padraoX.txt onde X é um valor entre 1 a S e S é a quantidade de sequências que serão processadas, e um valor entre 1 a N e N é a quantidade de padrões que serão analisados, respectivamente.

Acesse o diretório do projeto em seu computador por linha de comando:

```
$ cd /diretório/contendo/projeto
```

Crie e acesse a pasta build:

```
$ mkdir build && cd build
```

Crie os scripts de compilação da aplicação:

```
$ cmake ../.
```

Para compilar:

```
$ make
```

Para executar:

```
$ ./strings.exe /diretório/dnaX.txt /diretório/padraoX.txt
```

2.2. Divisão do Trabalho

Cada integrante do grupo ficou responsável por programar uma das especificações deste. Ficando a divisão da seguinte forma:

André Luís Braga Dutra: Algoritmo KMP e cálculo da frequência dos símbolos do método de compressão de Huffman.

Hyago Assis de Novais Oliveira: Algoritmo BMH e método de descompressão do método de Huffman.

Mara de Lemos Gomes: Algoritmo de força-bruta, montagem da tabela de código e cálculo da taxa de compressão do método de Huffman.

Rômulo Luiz Araujo Souza Soares: Método para cálculo da função prefixo parte do algoritmo KMP e montagem de árvore binária do método de compressão de Huffman.

2.3. Descrição das Estruturas Implementadas

Abaixo estão listadas as principais funcionalidades da aplicação para esta etapa do trabalho, separadas pelos seus respectivos arquivos.

casamentoDePadrao.h: Arquivo responsável por implementar as funções de casamento de padrão para analisar as sequências de dna.

- A função **forçaBruta** implementa o algoritmo de mesmo nome para busca em cadeia de caracteres. Percorre cada posição da sequência testando se as posições do padrão casam. Ao final, a função imprime na

tela a quantidade de casamentos ocorridos e suas posições no arquivo texto ou a não ocorrência de casamento de padrão.

- A função **kmp** é responsável por aplicar o algoritmo KMP que procura por um padrão em um determinado texto e utiliza uma função prefixo para diminuir a quantidade de leituras de caracteres.
- A função **funcaoPrefixo** é responsável por definir o caminhamento sobre quantas posições deve-se caminhar. Ela recebe uma string para definir o caminho.
- A função **bmh** implementa o algoritmo de mesmo nome, que consiste em pesquisar um padrão em determinado texto percorrendo-o da direita para esquerda em busca de igualdades, e caso encontre realiza o cálculo de deslocamento da direita para esquerda a fim de evitar comparações desnecessárias.

Huffman.h: Arquivo responsável por definir e implementar o método de compressão de Huffman.

- A função **contaFrequencias** é responsável por ler todos os caracteres da sequência de dna e adicionar cada caractere e suas repetições em uma função “map”, após isso, são adicionados ao vetor de nós que ao final é ordenado de forma crescente ao comparar as frequências pela função “sort”.
- A função **criaArvore** é responsável por montar uma árvore binária, utilizada para executar a codificação do método de Huffman. Essa árvore é criada a partir da tabela de frequências calculada na função *contaFrquencias*. Os nós de menor frequência são unidos até que todos estejam conectados.
- O método **montarTabCodigos** têm por objetivo percorrer a árvore binária e montar o dicionário com o novo código de cada símbolo. Utiliza para isso o método auxiliar *preencheTabCodigos* que verifica se um nó é uma folha da árvore, se sim, adiciona ao dicionário o símbolo e seu código correspondente, caso contrário percorre o nó à esquerda adicionando "0" ao seu código e a direita adicionando "1" ao seu código.

- A função **descompressao** é responsável por decodificar um texto passado como parâmetro com base no dicionário montado. A função percorre a árvore começando pela raiz, caso o bit seja 0 se percorre para esquerda e caso seja 1, para direita. Assim, é realizado o percurso até achar uma folha e é repetido o processo até não haver mais bits.

No.h: Arquivo responsável por definir e implementar os atributos e métodos dos nós utilizados na classe Huffman.

- A função **No** é o construtor do objeto *No,* que recebe como parâmetro seu símbolo e sua frequência.
- A função **~No** é o destrutor do objeto No.
- A função **getSimbolo** é responsável por retornar o símbolo referente ao nó.
- A função **getFrequencia** é responsável por retornar a frequência referente ao nó.
- A função **getEsq** é responsável por retornar o filho à esquerda do nó.
- A função **getDir** é responsável por retornar o filho à direita do nó.

3. EXPERIMENTOS COMPUTACIONAIS

3.1. Descrição das Instâncias e Conjuntos de Parâmetros

Foram criados três arquivos de sequências de DNA geradas pelo site bioinformatics[1], denominados dna1.txt, dna2.txt e dna3.txt, com 10000000, 1000000 e 100000 caracteres, respectivamente. Todos estes foram utilizados no método de compressão de Huffman e somente o primeiro também foi utilizado nos métodos força-bruta, KMP e BFH.

Para testes de casamento de padrão foram criados os arquivos padrao1.txt, padrao2.txt, padrao3.txt, padrao4.txt e padrao5.txt com 5, 10, 50, 100, 200 caracteres, respectivamente, sendo os dois primeiros gerados por [1] e os demais retirando-se partes da sequência dna1.txt.

3.2. Ambiente Computacional do Experimento

Os testes foram realizados em quatro computadores com as seguintes configurações:

Computador 1: processador Intel i3 de 4ª geração, 4GB de RAM, Ubuntu 20.04 (Subsistema do Windows 10), compilador GNU GCC compiler.

Computador 2: processador Intel i5 de 10ª geração, 16GB de RAM, Ubuntu 20.04, compilador GNU GCC compiler.

Computador 3: processador Intel i5 de 6ª geração, 8GB de RAM, Ubuntu 20.04 (Subsistema do Windows 10), compilador GNU GCC compiler.

Computador 4: processador Intel i7 de 7ª geração, 8GB de RAM, Ubuntu 20.04.2, compilador GNU GCC compiler.

3.3. Resultados e Análise

3.3.1. Resultados Casamento de Padrão

Tabela 1: Resultados médios do Algoritmo de Força Bruta

Algoritmo Força Bruta				
padrao1.txt				
	Comp. 1	**Comp. 2**	**Comp. 3**	**Comp. 4**
Casamentos	9600	9600	9600	9600
Tempo de Execução	0.258s	0.103s	0.296s	0.168s
padrao2.txt				
Casamentos	9	9	9	9
Tempo de Execução	0.239s	0.086s	0.258s	0.173s
padrao3.txt				
Casamentos	1	1	1	1
Tempo de Execução	0.252s	0.085s	0.255s	0.161s
padrao4.txt				
Casamentos	1	1	1	1

Tempo de Execução	0.296s	0.087s	0.253s	0.158s
padrao5.txt				
Casamentos	1	1	1	1
Tempo de Execução	0.244s	0.081s	0.254s	0.115s

Tabela 2: Resultados médios do Algoritmo de KMP

Algoritmo KMP				
padrao1.txt				
	Comp. 1	**Comp. 2**	**Comp. 3**	**Comp. 4**
Casamentos	9600	9600	9600	9600
Tempo de Execução	0.370s	0.096s	0.307s	0.204s
padrao2.txt				
Casamentos	9	9	9	9
Tempo de Execução	0.279s	0.096s	0.272s	0.158s
padrao3.txt				
Casamentos	1	1	1	1
Tempo de Execução	0.260s	0.093s	0.265s	0.162s
padrao4.txt				
Casamentos	1	1	1	1
Tempo de Execução	0.272s	0.093s	0.262s	0.151s
padrao5.txt				
Casamentos	1	1	1	1
Tempo de Execução	0.292s	0.086s	0.262s	0.144s

Tabela 3: Resultados médios do Algoritmo de BMH

Algoritmo BMH				
padrao1.txt				
	Comp. 1	**Comp. 2**	**Comp. 3**	**Comp. 4**
Casamentos	9600	9600	9600	9600
Tempo de Execução	0.148s	0.061s	0.126s	0.081s
padrao2.txt				
Casamentos	9	9	9	9
Tempo de Execução	0.102s	0.071s	0.117	0.067s
padrao3.txt				
Casamentos	1	1	1	1
Tempo de Execução	0.069s	0.025s	0.079s	0.038s
padrao4.txt				
Casamentos	1	1	1	1
Tempo de Execução	0.130s	0.05s	0.098s	0.085s
padrao5.txt				
Casamentos	1	1	1	1
Tempo de Execução	0.107s	0.073s	0.085s	0.047s

Como as sequências de DNA são únicas, o casamento de padrões aleatórios maiores do que 10 caracteres não ocorreram. Como estratégia foram utilizadas partes do arquivo dna1.txt nos arquivos padrao3.txt, padrao4.txt, padrao5.txt, o que garantiu o casamento ao menos 1 vez.

O algoritmo de Boyer-Moore-Horspool (BMH) se mostrou o mais eficiente para todos os casos com relação ao tempo de execução.

Observa-se que para todos os algoritmos os tempos de execução são relativamente estáveis para todos os padrões analisados.

3.3.2. Resultados Compressão de Huffman

Tabela 4: Resultados médios do Algoritmo de Compressão de Huffman

Total de 10000000 Caracteres				
	Comp. 1	**Comp. 2**	**Comp. 3**	**Comp. 4**
Custo Computacional Compressão	8.393s	3.169s	4.877s	5.768s
Taxa de Compressão	75%	75%	75%	75%
Custo Computacional Descompressão	0.655s	0.275s	0.822s	0.544s
Total de 1000000 Caracteres				
Custo Computacional Compressão	0.803s	0.301s	0.513s	0.559s
Taxa de Compressão	75%	75%	75%	75%
Custo Computacional Descompressão	0.073s	0.026s	0.086s	0.058s
Total de 100000 Caracteres				
Custo Computacional Compressão	0.088s	0.036s	0.083s	0.056s
Taxa de Compressão	75%	75%	75%	75%
Custo Computacional Descompressão	0.007s	0.02s	0.007s	0.005s

O custo computacional para a descompressão é significativamente inferior ao custo computacional para a compressão. Isto porque a descompressão utiliza apenas a árvore binária pré-processada pela compressão, enquanto a compressão processa a tabela de frequência dos caracteres, a árvore binária e a tabela de códigos.

A taxa de compressão para todas as sequências analisadas foi de 75% em razão de todas possuírem os mesmos caracteres com frequências próximas.

4. CONCLUSÕES

As sequências de DNA analisadas possuem frequências dos caracteres muito próximas, o que acaba gerando uma tabela de códigos de comprimento fixo, que reduz a taxa de compressão. O algoritmo de Huffman se torna mais eficiente em aplicações com um desequilíbrio grande entre as probabilidades de ocorrências dos símbolos, assim os caracteres com maior frequência possuem um código mínimo, o que provoca uma boa redução no tamanho do texto.

As análises utilizando os métodos de casamento de padrão demonstram números de casamentos iguais para os algoritmos utilizados, pois os mesmos possuem a mesma precisão para estar encontrando os resultados, se diferenciando somente na otimização de buscas, ocasionando tempos de execuções diferentes para cada.

REFERÊNCIAS

[1] Bioinformatics. **Sequence Manipulation Suite**. Disponível em: <https://www.bioinformatics.org/sms2/random_dna.html>. Acesso em: 01 de setembro de 2021.