UNIVERSIDADE FEDERAL DO ESPÍRITO SANTO

ESTRUTURA DE DADOS II

LUCIANA LEE

TRABALHO PRÁTICO

INTEGRANTES:LUCAS RANIERE, MATHEUS MARCARINI, PAULO ROBERTO.

INTRODUÇÃO

Este trabalho consiste na implementação de um algoritmo para a busca da correspondência de cadeias de caracteres de uma ocorrência de um determinado vírus em uma espécie de animais.

POSSÍVEIS SOLUÇÕES

Algoritmos:

- Algoritmo de força bruta
  - Vantagem: Muito simples.
  - Desvantagem: Muito ineficiente.
- KMP
  - Vantagem: algoritmo eficiente para buscar ocorrência.
  - Desvantagem: implementação um pouco mais complexa.

POSSÍVEIS SOLUÇÕES

Uso de memória

- Carregar todos os dados na memória e processá-los a partir dali
  - Vantagem: muito eficiente considerando que leituras em disco são custosas
  - Desvantagem: cria uma restrição quanto ao tamanho do arquivo lido, deve ser menor do que a memória disponível.
- Carregar os dados aos poucos e processá-los em partes.
  - Vantagem: elimina a restrição quanto ao tamanho do arquivo
  - Desvantagem: como leituras no arquivo são menos eficientes que leituras em memória principal há uma certa perda da eficiência

NOSSA SOLUÇÃO

- Algoritmo KMP:
  - Algoritmo consolidado e eficiente de buscar correspondência

- DNAs de animais podem ser sequências muito grande
  - Com isso em mente pensamos em uma solução que funcionasse para qualquer tamanho de arquivo.

O ALGORITMO

O algoritmo desenvolvido usa uma abordagem de leitura de buffer, considere um buffer de tamanho N.

1. Lê o nome e um padrão de um vírus.
2. Lê os N próximos caracteres do arquivo de DNAs.
3. Para esse padrão lido busca a ocorrência em todo o buffer.
4. Se o buffer não for suficiente para guardar todo o conteúdo do arquivo continuar lendo N caracteres até o fim do arquivo.
5. Repete.

O ALGORITMO

Exemplo com o tamanho do buffer de 30 caracteres, primeiro o nome do primeiro vírus e o seu padrão são carregados na memória.

PadroesVirus.txt

```
1  >variante V5.1
2  TTTTGA
3  >variante B10.2
4  ATTTCG
5  >variante L13.2
6  TCGA
7  >variante P2.3
8  CGATCGA
9  >EOF
```

BaseDadosDNA.txt

```
1  >R531273.1 preguica
2  TTTTTTTTTTTGACACTATCGAGCGCCCACACACTAGCTAGCTAGCTAGCGCGCGGCGCGATCGATCGATCGAA
3  GGAGTAGTAGTCGTACGATCGATCAGCTAGTCGATCGATCGATCATGCATGCTACTCGATCGATCGATCA
4  GCTAGCTAGCTAGCTAGCTAGCAGTCATCGATCGATCGATCGATCGATCGATCGATCGATGCACGATGTA
5  ACAGCATGCTAGCTAGC
6  >A141445.2 jacare
7  AAAAAAAAAAAACGGAGTCGTACTGAGTCAGCGCGCGGCGACTGAGCTAGCTAGCTAGCTAGCTAGCTAC
8  GTGTGTGCTAGCTAGCTAGCATCGTCTCGATCGATCGATCGTGCATGCATCGATCGATCGATCGATCGAT
9  TTATTAATTCATCATCTCTCTCTACTATCTACGATCTACGATGCATGCTAGCTGCGCGCGAGTCGATCGA
10 GCTAGCTAGCTAGCTAGCTAGCTACTAC
11 >EOF
```

O ALGORITMO

O Buffer é preenchido com os 30 primeiros caracteres, então o primeiro nome de DNA é guardado. padrão = TTTTGA

```
>R531273.1 preguica
TTTTTTTTTTGACACTATCGAGCGCCCACACACTAGCTAGCTAGCTAGCGCGCGGCGCGATCGATCGATCGAA
GGAGTAGTAGTCGTACGATCGATCAGCTAGTCGATCGATCGATCATGCATGCTACTCGATCGATCGATCA
GCTAGCTAGCTAGCTAGCTAGCAGTCATCGATCGATCGATCGATCGATCGATCGATCGATGCACGATGTA
ACAGCATGCTAGCTAGC
>A141445.2 jacare
AAAAAAAAAAAACGGAGTCGTACTGAGTCAGCGCGCGGCGACTGAGCTAGCTAGCTAGCTAGCTAGCTAC
GTGTGTGCTAGCTAGCTAGCATCGTCTCGATCGATCGATCGTGCATGCATCGATCGATCGATCGATCGAT
TTATTAATTCATCATCTCTCTCTACTATCTACGATCTACGATGCATGCTAGCTGCGCGCGAGTCGATCGA
GCTAGCTAGCTAGCTAGCTAGCTACTAC
>EOF
```

Após ler o nome do DNA a leitura do padrão é iniciada até encontrar ‘>’ ou acabar o buffer.

O ALGORITMO

Com isso surge um problema, uma ocorrência do vírus pode ocorrer no fim de uma leitura e início de outra, sendo assim antes de fazer a próxima leitura do buffer o ponteiro de arquivo volta o tamanho do padrão menos um para a próxima leitura.

```
int deslocamento = (strlen(padrao))*-1 +1 ;

offset += deslocamento + tamanho;

fseek(arquivoDNAs, deslocamento, SEEK_CUR);

tamanho = lerProximos(buffer, arquivoDNAs);

pos = 0;
```

O ALGORITMO

Dessa forma não perdemos a ocorrência do padrão que acontece no fim da primeira leitura, padrão = TTTTGA

```
>R531273.1 preguica
TTTTTTTTTTGACACTATCGAGCGCCCACACACTAGCTAGCTAGCTAGCGCGCGGCGCGATCGATCGAA
GGAGTAGTAGTCGTACGATCGATCAGCTAGTCGATCGATCGATCATGCATGCTACTCGATCGATCGATCA
GCTAGCTAGCTAGCTAGCTAGCAGTCATCGATCGATCGATCGATCGATCGATCGATCGATGCACGATGTA
ACAGCATGCTAGCTAGC
>A141445.2 jacare
AAAAAAAAAAAACGGAGTCGTACTGAGTCAGCGCGCGGCGACTGAGCTAGCTAGCTAGCTAGCTAGCTAC
GTGTGTGCTAGCTAGCTAGCATCGTCTCGATCGATCGATCGTGCATGCATCGATCGATCGATCGATCGAT
TTATTAATTCATCATCTCTCTCTACTATCTACGATCTACGATGCATGCTAGCTGCGCGCGAGTCGATCGA
GCTAGCTAGCTAGCTAGCTAGCTACTAC
>EOF
```

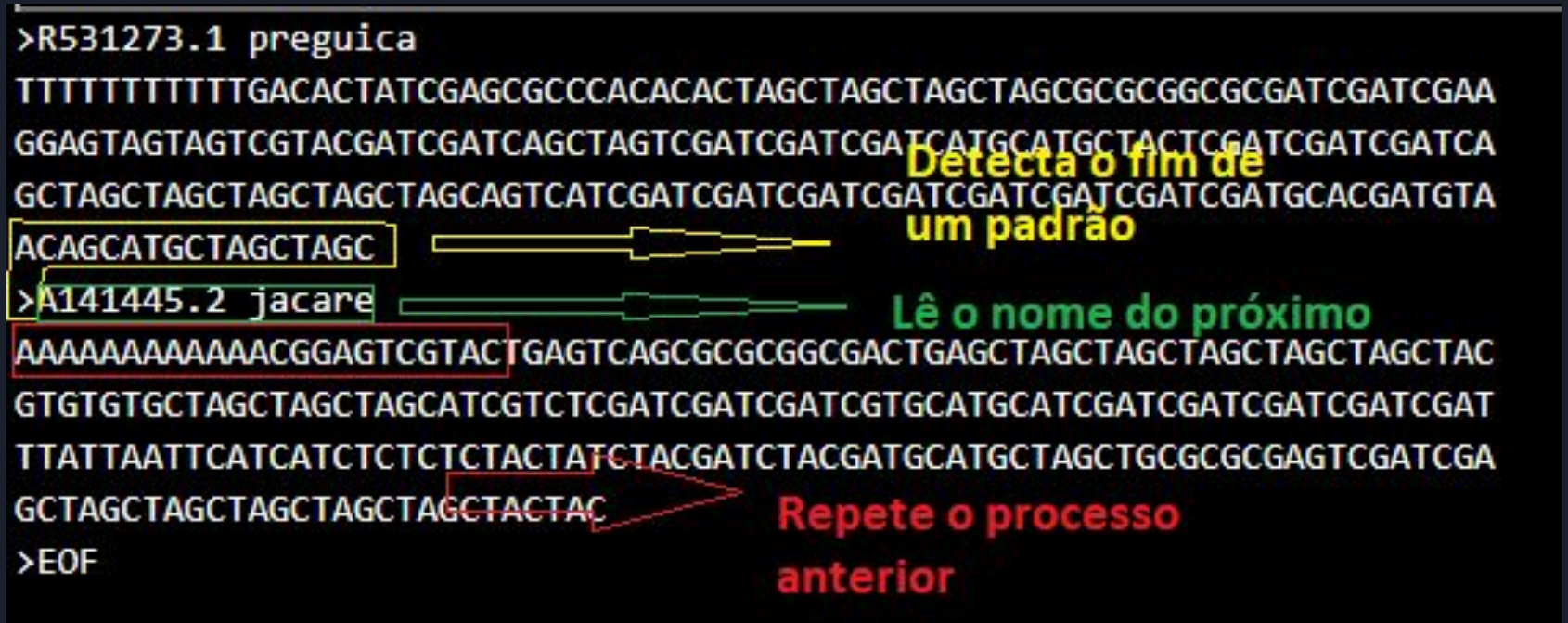
O ALGORITMO

Além disso o algoritmo precisa saber o deslocamento da parte atual de leitura desde o início do padrão então uma variável offset é calculada:

```
int deslocamento = (strlen(padrao))*-1 +1 ;

offset += deslocamento + tamanho;

fseek(arquivoDNAs, deslocamento, SEEK_CUR);

tamanho = lerProximos(buffer, arquivoDNAs);

pos = 0;
```

O ALGORITMO

O algoritmo segue assim até o fim do arquivo e toda vez que encontra o caractere ‘>’ exibe as ocorrências do vírus no DNA que acabou de ler e parte pro próximo.



FUNÇÃO CALCULAR LPS

i	0	1	2	3	4	5	6
π[i]	0	0	1	2	0	0	1
S	A	C	A	C	T	G	A

1º etapa) Considere um valor j=0, e um valor i=1, compare π[i] com π[j]
2º etapa)Ao realizar a comparação se for verdadeira, então π[i] = j+1 na posição i.Caso contrário π[i]=0 na posição i.
3º etapa)Ao fim de toda comparação incremente o i em +1, se for verdadeiro incremente o j em +1, se for falso j = 0.
4º etapa)Quando houver descasamento, o π[i] receberá 0, e j retornará a 0.
5º etapa)Quando i chega ao último caractere é encerrado a função de prefixo após verificar a ocorrência naquela cadeia.

FUNÇÃO DE CALCULAR LPS

```c
void calcularLPS(char *padrao, int m, int lps[]) {
    int len = 0; // Comprimento do prefixo atual
    int i = 1;

    lps[0] = 0; // O primeiro caractere sempre tem LPS igual a zero

    while (i < m) {
        if (padrao[i] == padrao[len]) {
            len++;
            lps[i] = len;
            i++;
        } else {
            if (len != 0) {
                len = lps[len - 1];
            } else {
                lps[i] = 0;
                i++;
            }
        }
    }
}
```

Função com
de execução
(m),usando
método pote
de análise
amortizada

FUNÇÃO BuscaKMP

i	0	1	2	3	4	5	6	7	8	9	10	11	12	13
T	C	G	A	C	T	A	G	T	A	C	A	A	T	A
P	A	C	A	C	T	G	A							

1º etapa) Inicia a Função calcular LPS para encontrar a tabela de prefixos

i	0	1	2	3	4	5	6
π[i]	0	0	1	2	0	0	1
S	A	C	A	C	T	G	A

FUNÇÃO BuscaKMP

2º etapa) Casamento

i	0	1	2	3	4	5	6	7	8	9	10	11	12	13
T	C	G	A	C	T	A	G	T	A	C	A	A	T	A
P	A	C	A	C	T	G	A							
π[i]	0	0	1	2	0	0	1							

3º etapa) Comparação e deslocamento

Realiza a comparação de cadeia com cadeia das strings.

FUNÇÃO BuscaKMP

i	0	1	2	3	4	5	6	7	8	9	10	11	12	13
T	C	G	A	C	T	A	G	T	A	C	A	A	T	A
P	A	C	A	C	T	G	A							
π[i]	0	0	1	2	0	0	1							

C=A? Não i=0, j=0

G=A? Não i=1, j=0

A=A? Sim i=2, j=0

i	0	1	2	3	4	5	6	7	8	9	10	11	12	13
T	C	G	A	C	T	A	G	T	A	C	A	A	T	A
P	A	C	A	C	T	G	A							
π[i]	0	0	1	2	0	0	1							

FUNÇÃO BuscaKMP

i	0	1	2	3	4	5	6	7	8	9	10	11	12	13
T	C	G	A	C	T	A	G	T	A	C	A	A	T	A
P	A	C	A	C	T	G	A							
π[i]	0	0	1	2	0	0	1							

C=C? Sim i=3, j=1 4º etapa) Ocorreu um descasamento,identificamos o valor anterior na tabela prefixo.

T=A? Não i=4, j=2

i	0	1	2	3	4	5	6	7	8	9	10	11	12	13
T	C	G	A	C	T	A	G	T	A	C	A	A	T	A
P	A	C	A	C	T	G	A							
π[i]	0	0	1	2	0	0	1							

FUNÇÃO BuscaKMP

i	0	1	2	3	4	5	6	7	8	9	10	11	12	13
T	C	G	A	C	T	A	G	T	A	C	A	A	T	A
P	A	C	A	C	T	G	A							
π[i]	0	0	1	2	0	0	1							

5º etapa) Como o valor é zero iniciamos a comparação da posição j=0 na tabela de padrões.

T=A? Não i=4, j=0

A=A? Sim i=5, j=0

i	0	1	2	3	4	5	6	7	8	9	10	11	12	13
T	C	G	A	C	T	A	G	T	A	C	A	A	T	A
P	A	C	A	C	T	G	A							
π[i]	0	0	1	2	0	0	1							

FUNÇÃO BuscaKMP

i	0	1	2	3	4	5	6	7	8	9	10	11	12	13
T	C	G	A	C	T	A	G	T	A	C	A	A	T	A
P	A	C	A	C	T	G	A							
π[i]	0	0	1	2	0	0	1							

C=G? Não i=6, j=1

T=A? Não i=7, j=0

A=A? Sim i=8, j=0

i	0	1	2	3	4	5	6	7	8	9	10	11	12	13
T	C	G	A	C	T	A	G	T	A	C	A	A	T	A
P	A	C	A	C	T	G	A							
π[i]	0	0	1	2	0	0	1							

FUNÇÃO BuscaKMP

i	0	1	2	3	4	5	6	7	8	9	10	11	12	13
T	C	G	A	C	T	A	G	T	A	C	A	A	T	A
P	A	C	A	C	T	G	A							
π[i]	0	0	1	2	0	0	1							

C=C? Sim i=9, j=1

A=A? Sim i=10, j=2

A=C? Não i=11, j=3

6º etapa) Um novo descasamento ocorreu, porém agora o valor do LPS é 1.

i	0	1	2	3	4	5	6	7	8	9	10	11	12	13
T	C	G	A	C	T	A	G	T	A	C	A	A	T	A
P	A	C	A	C	T	G	A							
π[i]	0	0	1	2	0	0	1							

FUNÇÃO BuscaKMP

i	0	1	2	3	4	5	6	7	8	9	10	11	12	13
T	C	G	A	C	T	A	G	T	A	C	A	A	T	A
P	A	C	A	C	T	G	A							
π[i]	0	0	1	2	0	0	1							

7ºetapa) Com isto, é possível pular uma casa na string dos padrões, e então continuar o emparelhamento com os próximos valores.

A=C? Não i=11, j=1

i	0	1	2	3	4	5	6	7	8	9	10	11	12	13
T	C	G	A	C	T	A	G	T	A	C	A	A	T	A
P	A	C	A	C	T	G	A							
π[i]	0	0	1	2	0	0	1							

FUNÇÃO BuscaKMP

i	0	1	2	3	4	5	6	7	8	9	10	11	12	13
T	C	G	A	C	T	A	G	T	A	C	A	A	T	A
P	A	C	A	C	T	G	A							
π[i]	0	0	1	2	0	0	1							

Por fim voltamos a verificação inicial

A=A? Sim i=11, j=0

C=T? Não i=12, j=1

A=A? Sim i=13, j=0

i	0	1	2	3	4	5	6	7	8	9	10	11	12	13
T	C	G	A	C	T	A	G	T	A	C	A	A	T	A
P	A	C	A	C	T	G	A							
π[i]	0	0	1	2	0	0	1							

Código da Função BuscarKMP

```c
int buscarKMP(char *texto,int lps[], char *padrao, int *pos, int offsetPos, int *ocorrencias, int *posicoes, int *auxPosicoes) {

    int m = strlen(padrao);
    int n = strlen(texto);


    int posinicial = *pos;
    int i = *pos; // Índice para o texto
    int j = 0; // Índice para o padrão
    int deslocamento = 0;
```

Continuação

```
while (i < n) {
    if(texto[i] == '\n') {
        i++;
    }
    if (texto[i] == '>') {
        //Retorna a posicao em que acabou um padrão para que a próxima leitura
        //do arquivo parta dessa posicao
        *pos = i;
        return LEITURA_PADRAO_COMPLETA;
    }

    if (padrao[j] == texto[i]) {
        i++;
        j++;
    }

    if (j == m) {
        (*ocorrencias)+=1;
        deslocamento = i-posinicial;
        posicoes[*auxPosicoes] = deslocamento - j + offsetPos - (deslocamento - j + offsetPos)/69;
        (*auxPosicoes)+=1;
        j = lps[j - 1];

    } else if (i < n && padrao[j] != texto[i] && texto[i] != '\n') {
        if (j != 0) {
            j = lps[j - 1];
        } else {
            i++;
        }
    }
}
```

PROBLEMAS DO ALGORITMO:

A fim de atingir esse objetivo de funcionar para qualquer tamanho de memória/arquivo um pouco de eficiência é perdida, alguns pontos de perda.

1. Requer mais leituras no arquivo
    a. Como o objetivo era usar menos memória então os dados dos vírus e dos DNAs não são mantidos na memória.
2. O arquivo dos padrões de vírus não são tratados com a abordagem de buffer.
    a. Considerando que os DNAs de animais são muito maiores que os dos vírus então essa abordagem foi considerada apenas no DNA dos animais

CONCLUSÃO

O resultado deste trabalho foi um programa que resolve o problema proposto em grande escala assim como em pequena escala