TRABALHO FINAL
MATEMÁTICA DISCRETA
2020.2

Professor: Diego Nunes Brandão
Alunos: Gabriel Franco Barreto da Silva e Nicolas Vycas Nery
Matrícula: 2012924BCC, 2012383BCC

Questão 1: Considere o Exemplo 6.1

(a) Suponha que decidimos adicionar k espaços vazios em um bloco contínuo. De quantas maneiras podemos fazer isso?

Resposta: Seja X uma cadeia de tamanho n, e Y um cadeia de tamanho $n - k$, $1 \leq k < k$.

Queremos alinhar os elementos em X com os elementos em Y adicionando espaços vazios a Y.

Quando são k espaços vazios em um bloco contínuo, então podemos adicionar o espações k no começo da cadeia Y, ao final da cadeia Y ou entre dois elementos consecutivos em Y.

Se tem $n - k - 1$ pares de elemento consecutivos, assim se tem $n - k - 1 + 2 = n - k + 1$ posições onde o bloco de k espaços vazios podem ser adicionados.

Isso implica que se tem $n - k + 1$maneira de se acionar os espaços vazios na cadeia Y

(b) Suponha que adicionamos dois blocos separados de espaços vazios, um de tamanho ie outro de tamanho $k - i$, para $1 \leq k < k$. De quantas maneiras podemos fazer isso?

Resposta: Utilizando-se da premissa anterior de que se tem $n - k + 1$maneira de se acionar os espaços vazios.

Existem $n - k + 1$possíveis posições para o bloco de tamanho i, e existem $n - k$possíveis posições para o bloco de tamanho $k - i$, já que 2 blocos precisam estar em 2 posições diferentes. E existem k possíveis valores para i

Se um dos eventos pode ocorrer de m maneiras e um segundo evento pode ocorrer de n maneiras, então o número de maneiras que os dois evento podem ocorrer em sequência é $m \cdot n$

$k(n - k + 1)(n - k)$

Temos $k(n - k + 1)(n - k)$ maneiras de fazer isso.

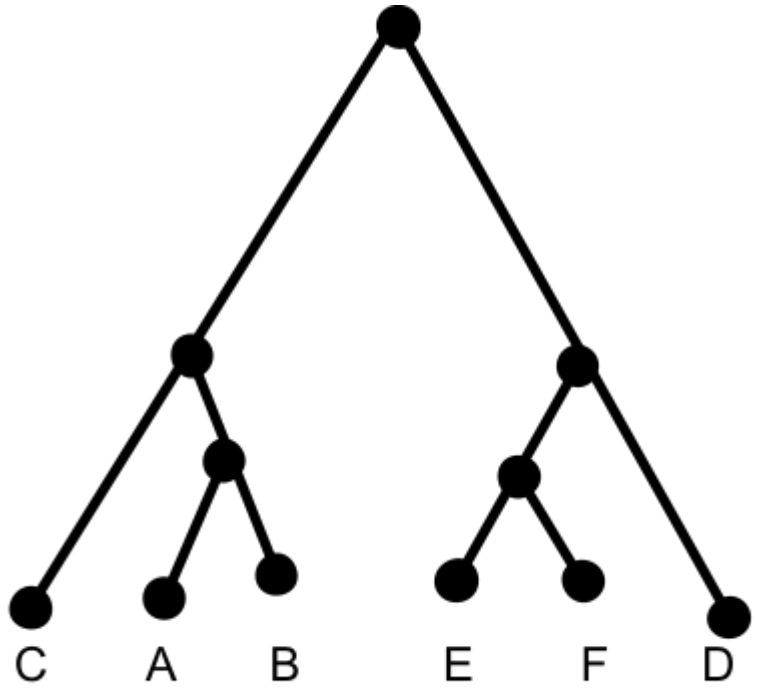
(c) Projete e implemente um algoritmo recursivo que percorra todas as maneiras de adicionar espaço vazios para a cadeia menor. Investigue a complexidade do seu algoritmo.

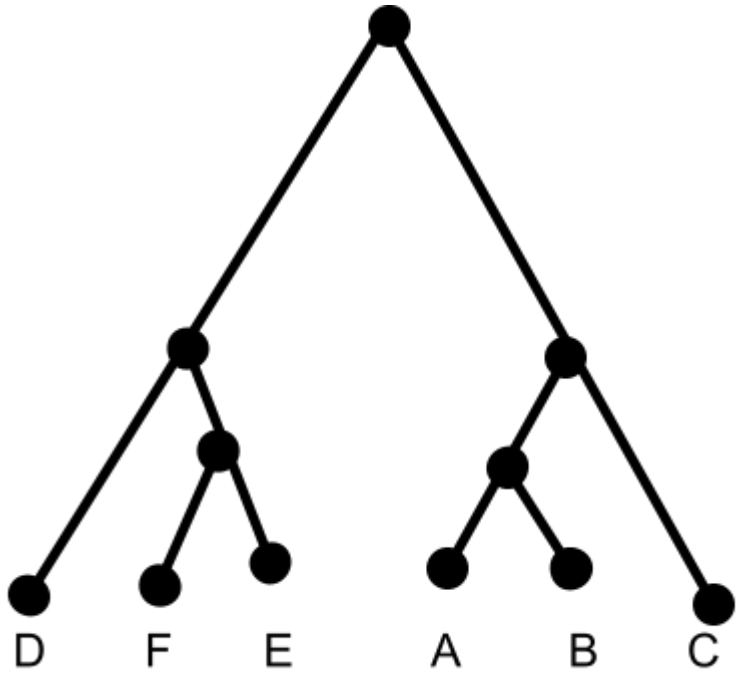
Resposta:

O algoritmo completo está no arquivo *Questão1/questao1.cpp* com as complexidades nos comentários.

Questão 2: Desenhe duas árvores binárias completas diferentes que representem a árvore evolutiva {{{A,B}, C}, {D,{E,F}}}.

Resposta:

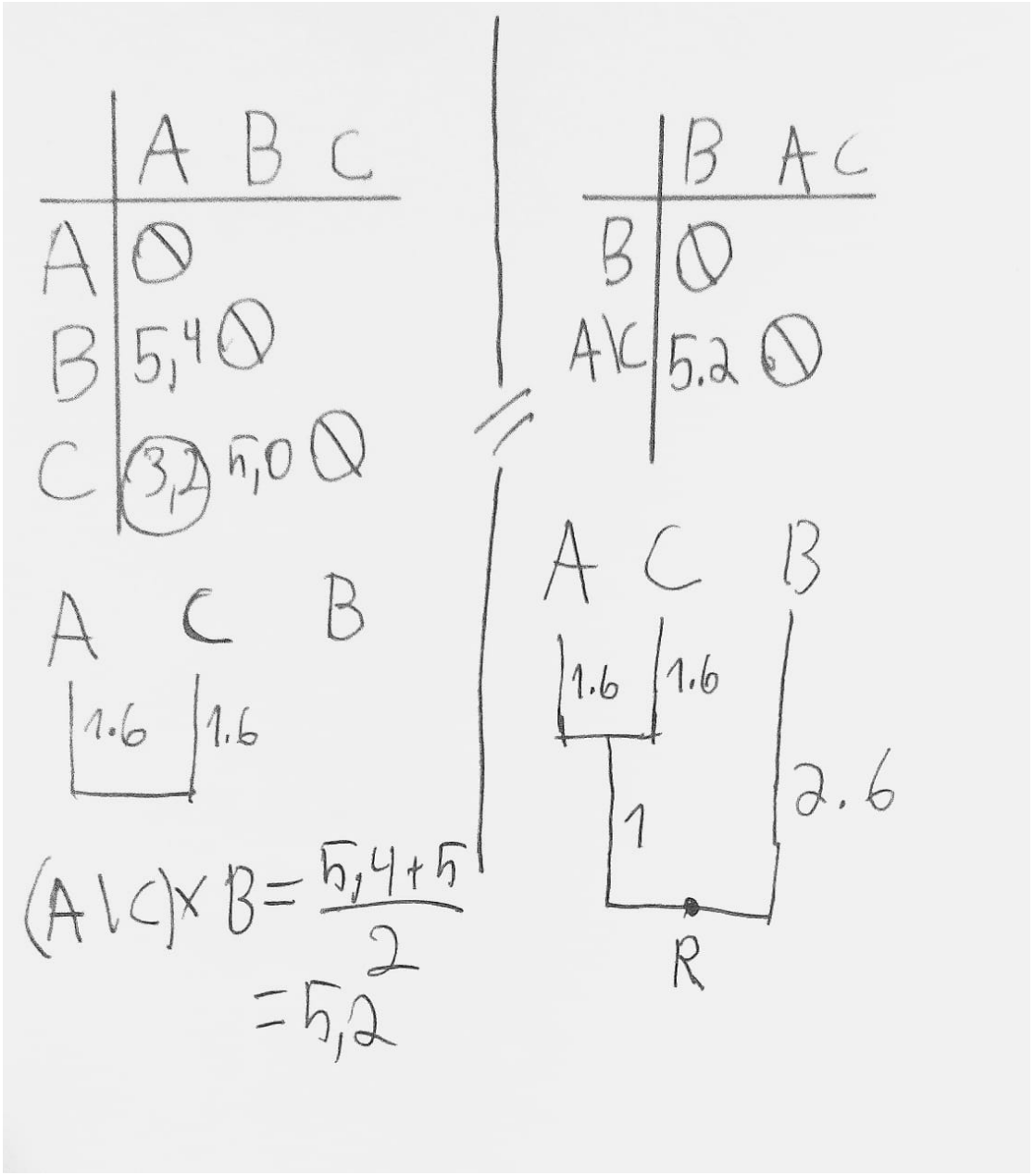




Questão 3: Calcule a árvore UPGMA para os dados na Tabela 6.1. A árvore se ajusta a esses dados de forma exata?

	A	B	C
A	0		
B	5,4	0	
C	3,2	5,0	0

Resposta:



Questão 4: Investigue coincidências de padrões no DNA. Encontre (ou escreva) um algoritmo que busque pela maior subcadeia que duas cadeias (possivelmente de tamanhos diferentes) têm em comum. Modifique o algoritmo para permitir a possibilidade de as subcadeias serem reversas. Implemente sua solução. Proponha uma expansão da sua solução para n-cadeias de DNA. Exemplo:

Exemplo:

```
Entrada:
      Sequência 1: AATCGTAGCTTTCGAA
      Sequência 2:       ATTTAGCTTAGGCGTTAA
Saída:              Tamanho: 6                    Subcadeia: TAGCTT
```

Resposta:

O algoritmo, que está implementado no arquivo *Questão4/questao4.cpp*, o soluciona através da força bruta com complexidade $O(n)$ no melhor caso onde as duas cadeias são iguais ou espelhadas possuindo o mesmo tamanho n e no pior caso é $O(n \cdot k \cdot log\, n)$onde n é o tamanho da *cadeia A* e ké o tamanho da *cadeia B.* Para possibilitar que o programa buscasse por cadeias inversas eu apenas inverti uma das cadeias e utilizo o mesmo algoritmo.

Saída do programa com cadeias definidas pelo usuário em tempo de execução.

```
Cadeia A: AACGTTGACGAG
Cadeia B: ACTGTGATAG
A maior sub cadeia é: TGA
Com o tamanho: 3
A maior sub cadeia reversa é: GA
Com o tamanho: 2

1 - Criar ou Recriar a Cadeia A 2 - Criar ou Recriar a Cadeia B
3 - Imprimir Cadeias            4 - Achar maior sub cadeia
0 - Sair do programa
Digite uma opção:
```

Ao começar a testar o programa comecei a achar tedioso ter que inserir cadeia por cadeia, então implementei um funcionalidade para carregar arquivos a partir dos parâmetros do programa fizemos um pequeno script em python que gera duas cadeia de aleatórias, as salva em 2 arquivos txt separados e executa o programa .

Saída do programa no modo de execução com dois arquivos de entrada

```
Abrindo arquivos CadeiaA.txt e CadeiaB.txt

Cadeia A possui o tamanho: 239
Cadeia B possui o tamanho: 189
A maior sub cadeia é: TCCTTTAA
Com o tamanho: 8
A maior sub cadeia reversa é: TTGTATC
Com o tamanho: 7
```

Uma das maneiras de acelerar o esse algoritmos porém não foi implementada seria reduzir o tamanho das cadeias as compactando as cadeias Ex: Seja A uma cadeia, A = AAATTCCGGTTA e C um função que compacta um cadeia dada como entrada contando a repetições internas da cadeia e substituído pelo número de vezes que o elemento repetiu e a base repetida C(A) = 3A2T2C2G2TA.

Para que o programa funcione com n cadeias e com o algoritmo de busca de maior subcadeia atual seria necessário buscar usando as possíveis subcadeia de todos as as cadeias e buscando

em cada cadeia por essas subcadeia, no caso de que seja encontrado uma maior sub cadeia, essa cadeia será para ser comparada com outras maiores subcadeia originárias das outras sub cadeia, assim achando a maior subcadeia comum entre todas as cadeias.

Abaixo está um pseudocódigo desse algoritmo.

```
// Lista com todas as cadeias
variável Cadeias[]
// Lista que irá guardar todas as maiores subcadeias encontradas
variável MaioresCadeias[]

// função retorna maior subcadeia entre duas cadeias
Função buscarMaiorSubCadeia(cadeia, cadeia)

//Função retorna Lista de subcadeias de uma cadeia e suas versões reversas
Função SubCadeias(cadeia)

// Função imprime um cadeia
Função imprimirCadeia(cadeia)

variável maiorSubcadeia;
Para cadeia em cadeias{
        variável maiorSubcadeiaAtual;
        Para subcadeia em SubCadeias(cadeia){
        //variável existe guarda se uma sua cadeia é comum a todas as outras cadeias
                variável existe = true;
                Para cadeiaAtual em cadeias{
                        se (contemSubcadeia(subcadeia, cadeiaAtual) == false){
                                existe = false;
                                break;
                        }
                }
                se existe{
                        maiorSubcadeiaAtual = subcadeia;
                }
        }
        se maiorSubcadeiaAtual ≠ null{
                maioresCadeias.adiciona(maiorSubcadeiaAtual);
        }
}


variável MaiorSubcadeia = CadeiaVazia;
Para subcadeia em MaioresCadeias{
        se subcadeia.tamnaho > MaiorSubcadeia{
                MaiorSubcadeia = subcadeia;
        }
}
imprimirCadeia(MaiorSubcadeia);
```