NOME: GABRIEL FERREIRA DE SOUZA ARAUJO **NUSP: 12718100**

RELATÓRIO EP3 - Laboratório de Métodos Numéricos

1. COMO COMPILAR E EXECUTAR

- Certifique-se de que o arquivo que deseja ser lido está com nome: “arquivo.txt” e está no mesmo diretório que o programa.

- Para imprimir o erro, é necessário passar na linha 10 (#define fita_original “”) entre as áspas a fita original; por exemplo: #define fita_original “TAGCAT” (as fitas originais dos arquivos de testes estão presentes na sessão 3 do relatório, basta copiar e colar na linha citada).

- Os arquivos precisam estar no formato abaixo:

Número de fragmentos N

Frag1

Frag2

FragN

- É necessário durante a execução do programa passar o parâmetro K.

- Compile no terminal da seguinte forma: gpp EP.cpp -o EP

- Execute no terminal da seguinte forma: ./EP

2. EXPLICANDO O PROGRAMA

2.1 Função “criarGrafo()”

Essa função recebe como parâmetros um vetor de strings chamado vertices, que representa os trechos da sequência genética, e um inteiro K, que define a quantidade mínima de caracteres em comum entre dois trechos para que exista uma conexão entre eles no grafo. A função re**torna um par de matrizes grafo e grafo_k.**

A função criarGrafo é responsável por construir o grafo com base nos vértices fornecidos. Ela percorre todos os pares de vértices e verifica se eles possuem uma quantidade mínima K de caracteres em comum. Se possuírem, são adicionados arcos entre os vértices no grafo e a quantidade de caracteres em comum é armazenada na matriz grafo_k (importante guardar esses números para concatenar os fragmentos).

2.2 Função “maiorCaminho()”

Essa função realiza uma busca em profundidade a partir de um vértice currentNode no grafo representado pela matriz de adjacência grafo. Ela utiliza a recursão para explorar todos os caminhos possíveis no grafo. A função mantém um vetor de visitados visitado, um vetor caminho que armazena o caminho percorrido até o momento e um vetor caminhoMaisLongo que armazena o maior caminho encontrado até o momento.

A função marca o vértice atual como visitado, o adiciona ao caminho e verifica se todos os vizinhos já foram visitados. Se algum vizinho não tiver sido visitado, a função é chamada recursivamente para esse vizinho. Caso contrário, verifica se o caminho atual é maior do que o caminho mais longo encontrado até o momento e atualiza o caminhoMaisLongo se necessário.

Após explorar todos os caminhos possíveis a partir do vértice atual, a função desmarca o vértice como visitado e remove o último vértice do caminho, permitindo a exploração de outros caminhos a partir de vértices anteriores.

2.3 Função “concatenar()”

Essa função recebe duas strings str1 e str2 como parâmetros e retorna a concatenação dessas duas strings, removendo o último caractere de str1.

2.4 Função “calcularErro()”

Essa função calcula o erro entre a sequência genética original e a sequência genética reconstruída. Ela recebe as duas sequências como parâmetros e retorna o erro calculado como um valor entre 0% e 100%.

A função percorre cada caractere da sequência original e compara com a sequência reconstruída. Para cada caractere em comum, a função verifica quantos caracteres subsequentes também são iguais nas duas sequências. O ponto de início do próximo

trecho em comum é armazenado em um vetor pontos. O maior valor desse vetor é considerado como o tamanho do maior trecho em comum entre as sequências.

Essa função tem uma implementação de complexidade vertices_seq_original*(vertices_seq_reconstruída)^2, podendo ser um pouco lenta para casos onde o tamanho da fita é grande, neste EP não teremos preocupação com isso. O erro é calculado como a diferença entre o maior trecho em comum e o tamanho máximo entre as sequências, normalizado para um valor entre 0% e 100%.

2.5 Função “main()”

Essa é a função principal do programa. Ela lê um arquivo chamado "arquivo.txt" que contém os dados de entrada necessários para a reconstrução da sequência genética. A primeira parte do código verifica se o arquivo foi aberto corretamente. Caso contrário, exibe uma mensagem de erro e encerra o programa.

Em seguida, o número de vértices é lido do arquivo e armazenado na variável numVertices. Depois disso, os vértices são lidos do arquivo e armazenados no vetor vertices. O valor de K é solicitado ao usuário através da entrada padrão. A função criarGrafo é chamada passando o vetor vertices e K, e o resultado é armazenado nas variáveis grafo e grafo_k. Em seguida, a matriz de adjacência grafo é exibida na saída padrão.

A função procuraCaminhoMaisLongo é chamada passando o grafo e o resultado é armazenado no vetor caminhoMaisLongo. O caminho mais longo encontrado é exibido na saída padrão, seguido pela sequência genética original e a sequência reconstruída. O erro entre as sequências é calculado utilizando a função calcularErro e exibido na saída padrão.

3. TESTES E CONCLUSÕES

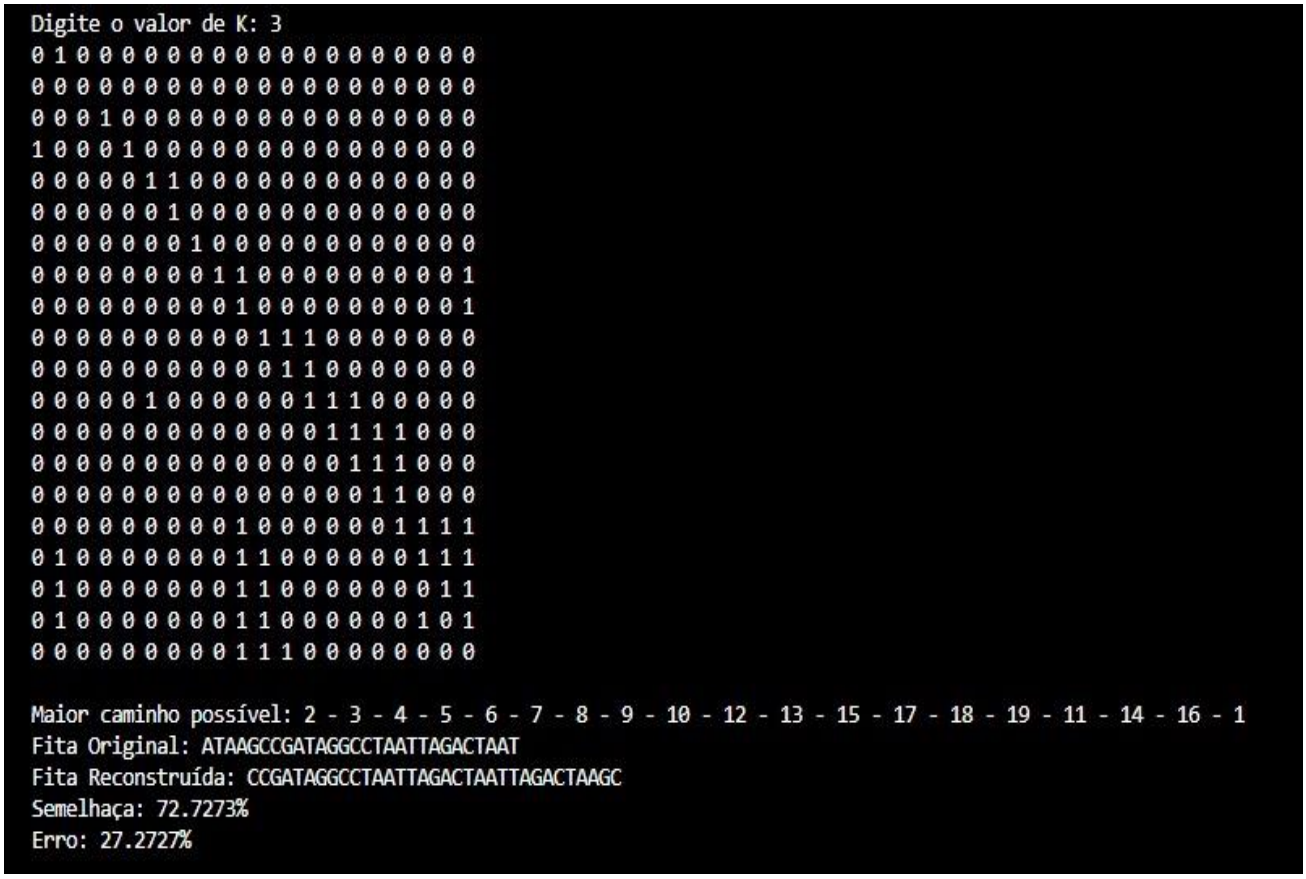
Os testes foram gerados sem uso de programas, apenas por criação própria.Ao realizar os testes foi possível notar que o valor do parâmetro K e a ordem em que os fragmentos eram arranjados no arquivo eram fatores que se relacionavam com o erro. O erro poderia ser maior ou menor de acordo com K de uma sequência específica, além disso, nota-se que quanto maior o K, maior a velocidade de execução do programa. Para teste com mais de 20 fragmentos e K=2, o programa apresentou

dificuldades no tempo de execução, mas aumentando K, a velocidade tem uma taxa razoável. O fator da ineficiência de execução ocorre por causa da implementação da função “maiorCaminho()” (veja mais na sessão 2.2), nela contém a heurística de eliminar arestas que estão presentes num circuito. Veja as saídas dos testes abaixo:

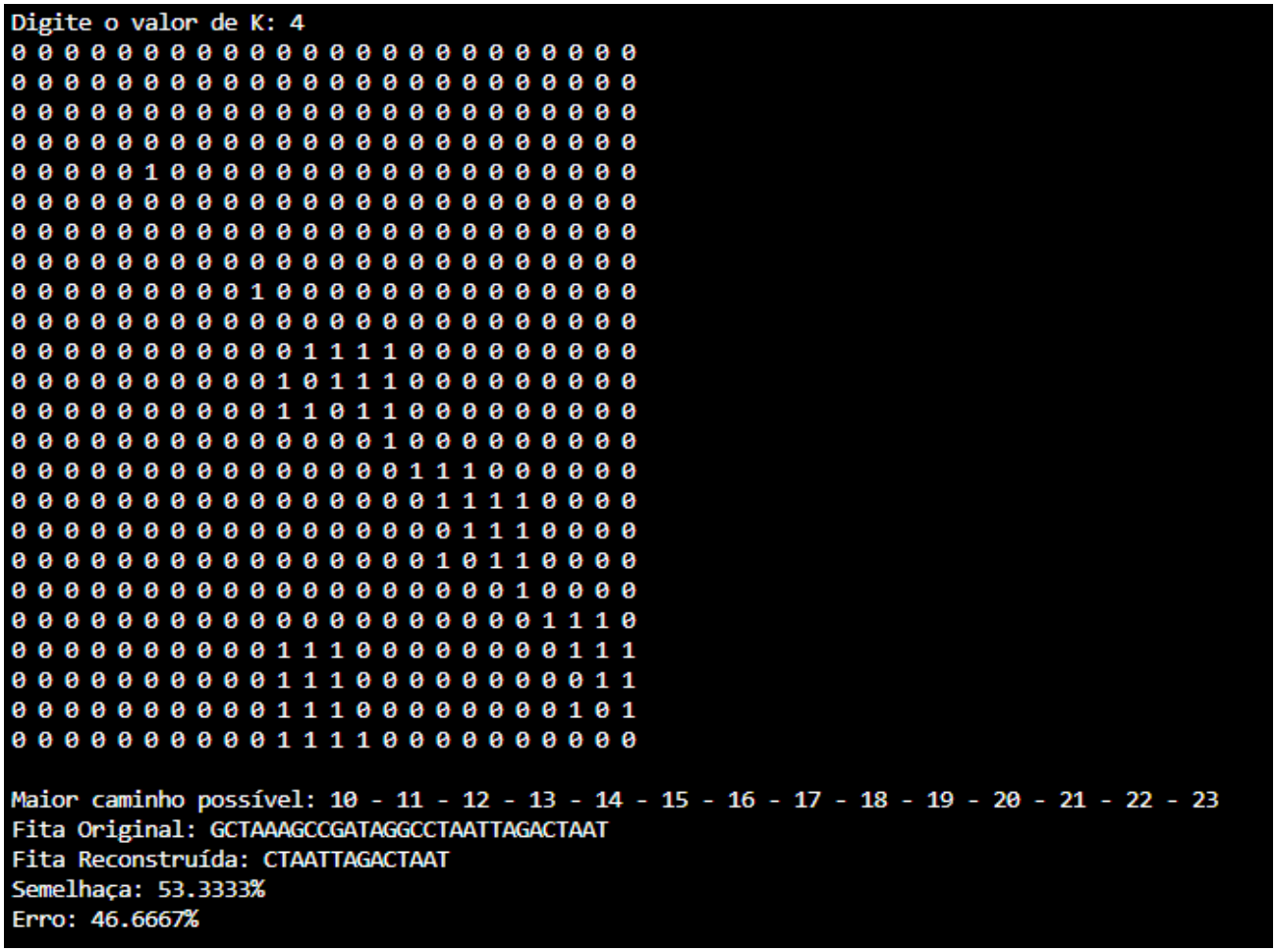
```
Digite o valor de K: 2
0 0 0 0 1 0 0 0 1 0 0 0
0 0 1 0 0 1 1 0 0 1 0 0
0 0 0 1 0 0 0 1 0 0 1 1
0 1 0 0 0 0 1 1 0 0 1 1
0 0 1 0 0 0 0 0 1 0 0 0
0 1 1 0 0 0 1 0 0 0 0 0
1 0 1 1 0 0 0 0 0 1 0 0
0 1 0 0 0 0 1 0 0 0 1 1
0 1 1 0 0 1 1 0 0 0 0 0
0 0 1 0 0 1 1 0 0 0 0 0
1 1 0 1 0 0 0 0 0 1 0 1
1 1 0 1 0 0 0 0 0 1 1 0

Maior caminho possível: 0 - 4 - 8 - 1 - 2 - 3 - 7 - 10 - 11 - 9 - 5 - 6
Fita Original: ACTCGTAAATACATAACGATAC
Fita Reconstruída: ACTCGTAAATACATAACGATACATAAATAAC
Semelhaça: 74.1935%
Erro: 25.8065%
```

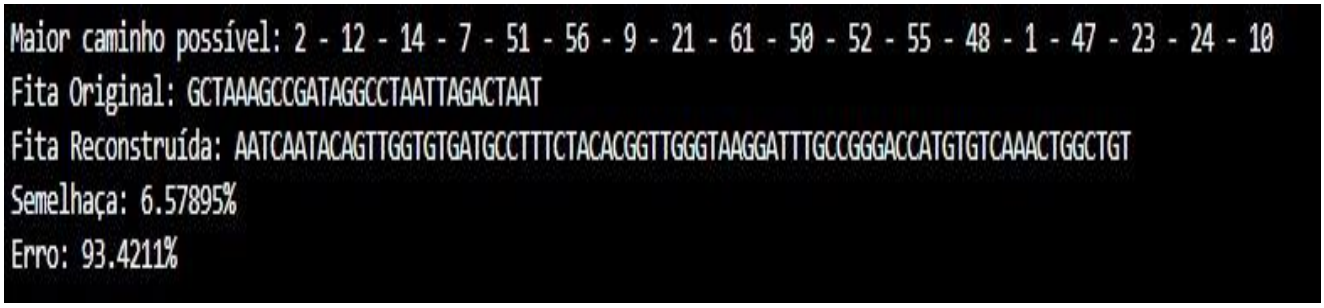
• Fragmentos = 12, K = 2, fita original = ACTCGTAAATACATAACGATAC, Erro = 25.80%



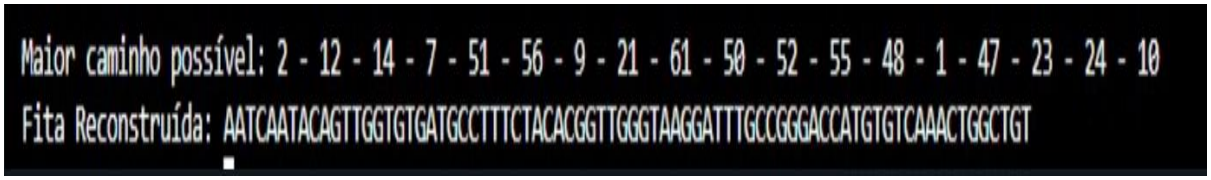
• Fragmentos = 20, K = 3, fita original = ATAAGCCGATAGGCCTAATTAGACTAAT, Erro = 27.27%



• Fragmentos=24, K=4, fita original = GCTAAAGCCGATAGGCCTAATTAGACTAAT, Erro = 46.66%



• Fragmentos = 20, K = 3, fita Original = ATAAGCCGATAGGCCTAATTAGACTAAT, Erro = 27.27%



• Fragmentos = 62, K = 30, fita original = Desconhecida, Erro = NA

OBS: Caso for usar os testes descritos aqui e anexados com o programa, lembre-se de definir corretamente a **fita original** dada aqui no relatório abaixo das imagens dos testes e leia com atenção a sessão 1.