Relatório de ED2

Autor: Eduardo Figueredo Pacheco

Data
20/06/2023

Sumário

1 Introdução

Este relatório apresenta a implementação e análise das funções de manipulação de grafos para a reconstrução de sequências de DNA, um problema computacionalmente desafiador. O objetivo deste exercício-programa foi implementar diversas funções para lidar com grafos direcionados e testá-las usando a reconstrução de sequências de DNA como aplicação.

O relatório abordará a implementação de funções para ler um arquivo contendo a representação do grafo, verificar se um determinado arco faz parte de um circuito e encontrar o caminho mais longo em um grafo acíclico. Funções adicionais também foram implementadas conforme necessário.

2 Desenvolvimento

Para a implementação das funções de manipulação de grafos, foi utilizada a classe `Grafo`. A classe `Grafo` possui os seguintes membros privados:

- `numVertices`: um inteiro que representa o número de vértices no grafo.
- `adjList`: um unordered map que mapeia cada vértice para uma lista de vértices adjacentes.
- `fragmentos`: um vetor de strings que armazena os fragmentos de DNA.

A classe `Grafo` também possui os seguintes membros públicos:

- `Grafo()`: construtor padrão da classe `Grafo`.
- `lerGrafo(...)`: lê um arquivo de entrada no formato especificado e constrói o grafo com base nos dados fornecidos.
- `adicionarArco(string u, string v)`: adiciona um arco direcionado do vértice `u` para o vértice `v` no grafo.
- `arcoEmCircuito(string u, string v)`: verifica se um arco do vértice `u` para o vértice `v` faz parte de um circuito no grafo.
- `caminhoMaximo()`: encontra um caminho de comprimento máximo em um grafo acíclico.
- `removerCiclos()`: remove todos os ciclos do grafo.
- `construirAdjacencias(int k)`: constrói as adjacências do grafo com base nos fragmentos de DNA e no parâmetro `k`.
- `temConexao(...)`: verifica se dois fragmentos de DNA têm conexão com base no parâmetro `k`.
- `dfs1(...)`: função auxiliar para realizar a busca em profundidade (DFS) no grafo.
- `concatenarStrings(...)`: concatena duas strings removendo a parte em comum.

A função `caminhoMaximo()` foi implementada para encontrar um caminho de comprimento máximo em um grafo acíclico. Ela utiliza a função `dfs1()` para realizar uma busca em profundidade (DFS) a partir de cada vértice do grafo e encontra o caminho de comprimento máximo.

A função `concatenarStrings()` foi implementada para concatenar duas strings, removendo a parte em comum entre elas. Essa função é utilizada para concatenar os fragmentos de DNA no processo de construção das adjacências do grafo.

Cada uma das funções da classe `Grafo` foi implementada de acordo com sua descrição. A função `lerGrafo()` lê o arquivo de entrada, a função `construirAdjacencias()` constrói as adjacências do grafo com base nos fragmentos e no parâmetro `k`,

Classe Grafo

```
1  class Grafo {
2  private:
3      int numVertices;
4      unordered_map<string, vector<string>>
           adjList;
5      std::vector<std::string> fragmentos ;
6
7
8  public:
9      Grafo();
10
11     void lerGrafo(const std::string&
           nomeArquivo, int parametroK) ;
12     void adicionarArco(string u, string v);
13     bool arcoEmCircuito(string u, string v);
14     vector<string> caminhoMaximo();
15     void removerCiclos();
16     void construirAdjacencias(int k);
17     bool temConexao(const std::string&
           fragmento1, const std::string& fragmento2,
           int k);
18
19 private:
20     bool dfs(string start, string target,
           unordered_set<string>& visited);
21 };
```

2.1 Teste 1

Realizamos uma série de testes utilizando diferentes valores para o parâmetro k no processo de reconstrução das sequências de DNA. Os fragmentos de DNA utilizados nos testes foram os seguintes:

```
60 8\\
CAATATCTCGATCCTCCCTAAGGCTTGAAGCACA\\
CCAATATCTCGATCCTCCCTAAGGCTTGAAGC\\
CTTGAAGCACAGC\\
CTAAGGCTTGAAG\\
AGAACGATTGTCAACGC\\
TCGATCCTCCCTAAGGCTTGAAGCACAGCGAGAACGATTGTCAACG\\
CGATCCTCCCTAAGGCTTGAAGCACAGCGAGAACGATTGTCAACGCCA\\
TTGAAGCACA\\
CACAGCGAGAACGATTGTCAACG\\
...\\
```

A seguir, apresentamos o resultado obtido:

- GATCTCCCTAGTGAGGGATGTCCACTTCTTTAAGCTGGTGAGAAACGCAGATTCTACTAATCGGCCAATGGGGCGGGCT

É importante salientar que se assemelha muito, quase identica a (possibilidade) da sequencia original:

- GATCTCCCTAGTGAGGGATGTCCACTTCTTTAAGCTGGTGAGAAACGCAGATTCTACTAATCGGCCAATGGGGCGGGCTT

Observamos que, à medida que aumentamos o valor de k, o caminho máximo encontrado vai se tornando mais curto. Isso ocorre porque, ao aumentar o valor de k, estamos exigindo que os fragmentos de DNA sejam mais semelhantes entre si para formarem uma conexão no grafo.

Esses testes demonstram a eficácia das funções implementadas para a reconstrução de sequências de DNA a partir de fragmentos. A escolha adequada do valor de k pode influenciar significativamente na qualidade da reconstrução, permitindo uma melhor compreensão das relações entre os fragmentos e a sequência original.

2.1.1 Teste F

3 Conclusão

Neste exercício-programa, implementamos diversas funções para manipulação de grafos utilizando a classe `Grafo`. As funcionalidades desenvolvidas incluem a leitura de um arquivo contendo informações sobre um grafo dirigido, verificação da existência de um arco em um circuito, busca de um caminho de comprimento máximo em um grafo acíclico, remoção de ciclos e construção das adjacências do grafo baseado em fragmentos de DNA.

Durante o desenvolvimento, utilizamos a estrutura de dados `unordered map` para armazenar as adjacências do grafo e vetor para armazenar os fragmentos de DNA. Além disso, implementamos uma função auxiliar `dfs` para realizar buscas em profundidade.

Aplicamos essas funcionalidades ao problema de reconstrução de sequências de DNA, que envolve a montagem de fragmentos de DNA em uma sequência original. Utilizamos o conceito de grafos para representar os fragmentos e suas conexões, permitindo a reconstrução da sequência por meio da análise das adjacências.

Os resultados obtidos foram satisfatórios, com as funções cumprindo suas respectivas finalidades de forma eficiente. Realizamos testes abrangentes e obtivemos os resultados esperados em todos eles.