Relatório da Implementação de Projeto e Análise de Algoritmos

Camila Lopes[1], **Victor Colen**[1]

[1]Instituto de CIências Exatas e Informática
Pontifícia Universidade Catórila de Minas Gerais (PUC-MG)
30535-901 – Coração Eucarístico – Belo Horizonte – MG

{lopes.camila, victor.costa}@sga.pucminas.br

1. Introdução

Este relatório explora e implementa duas classes para a manipulação de grafos direcionados, e outra para grafos não-direcionados. O grafo direcionado foi implementado a partir de uma lista de adjacência, e assim realizada uma busca em profundidade, rotulando a aresta "de árvore", "de retorno", "de avanço" e "de cruzamento". Já o grafo não-direcionado, foi implementado a partir da matriz de adjacência e, desse modo, realizado uma busca em largura encontrando a excentricidade de um determinado vértice.

2. Implementação

2.1. Grafo Direcionado

A lista de adjacência é uma estrutura de dados eficiente em termos de memória, especialemente para representar grafos esparsos, permitindo uma fácil representação e iteração sobre os vértices conectados.

2.1.1. Estrutura de Dados

No grafo direcionado é necessário observar que para cada vértice na lista de adjacência é associado a uma lista de vértices adjacentes.

Algorithm 1 Estrutura de Dados

$numVertices =$ O número de vértices do grafo
$adjacencyList =$ Lista de adjacência

2.1.2. Métodos Principais

Foram classificados dois métodos importantes, sendo eles o método de *Adição de Aresta* e o método de *Busca em Profundidade*.

Algorithm 2 Adicionar Aresta

Require: Um vértice de origem $sourceVertex$ e de destino $destinationVertex$
Adiciona $destinationVertex$ à lista de adjacência de $sourceVertex$

Algorithm 3 Busca em Profundidade (DFS)

Require: Um grafo com $numVertices$ vértices

Inicializa um vetor $visited$ de tamanho $numVertices$ com **falso**, indicando que nenhum vértice foi visitado
Inicializa um vetor $discoveryTime$ de tamanho $numVertices$ com -1, indicando o tempo de descoberta dos vértices
Inicializa um vetor $lowTime$ de tamanho $numVertices$ com -1, indicando o menor tempo acessível de cada vértice
Inicializa um vetor $parent$ de tamanho $numVertices$ com -1, indicando o pai de cada vértice na árvore DFS
Define uma variável $time$ como 0 para rastrear o tempo de descoberta
for cada $vertex$ de 0 a $numVertices - 1$ **do**
    if $vertex$ não foi visitado **then**
        Classificar a aresta como "Árvore".
        Continuar a DFS recursivamente.
    else
        Classificar a aresta como "Retorno", "Avanço" ou "Cruzamento" com base nas condições de tempo de descoberta.
    end if
end for

2.2. Grafo Não-Direcionado

A matriz de adjacência é uma estrutura de dados útil para a verificação rápida de conexões entre quaisquer dois vértices, se destancando pela sua simplicidade de acesso e modificação, ainda que apresente um custo maior em termos de armazenamento, especialmente em grafos de alta densidade.

2.2.1. Estrutura de Dados

No grafo não direcionado temos a matriz da adjacência, a qual cada célula indica a presença ou ausência de uma aresta entre dois vértices.

Algorithm 4 Estrutura de Dados

$numVertices =$ O número de vértices do grafo
$adjacencyMatrix =$ Matriz de Adjacência

2.2.2. Métodos Principais

Foram classificados dois métodos importantes, sendo eles o método de *Adição de Aresta* e o método de *Busca em Largura e Cálculo de Excentricidade*.

Algorithm 5 Adicionar Aresta

Require: Dois vértices $vertex1$ e $vertex2$
  Defina $adjacencyMatrix[vertex1][vertex2] \leftarrow 1$
  Defina $adjacencyMatrix[vertex2][vertex1] \leftarrow 1$

Algorithm 6 Busca em Largura (BFS) e Cálculo da Excentricidade

Require: Um vértice inicial $startVertex$
  Inicializa o vetor $distance$ de tamanho $numVertices$, com os valores iguais a $+\infty$
  Inicialize uma fila $bfsQueue$
  Defina o vértice inicial $distance[startVertex] \leftarrow 0$
  Adicione $startVertex$ à fila $bfsQueue$
  while $bfsQueue$ não estiver vazia **do**
    Retira o vértice da frente da fila e explora seus vizinhos
    for $i = 0$ **até** $numVertices - 1$ **do**
      if $adjacencyMatrix[currentVertex][i]$ **e** $distance[i] = +\infty$ **then**
        tualiza a distância e adiciona o vizinho à fila
      end if
    end for
  end while
  Inicialize $excentricity \leftarrow 0$
  for $i = 0$ **até** $numVertices - 1$ **do**
    if $distance[i] \neq +\infty$ **e** $distance[i] > excentricity$ **then**
      Defina $excentricity \leftarrow distance[i]$
    end if
  end for
  return $excentricity$

2.3. Criação de Grafos Aleatórios

Para auxiliar na análise das classes criadas foram implementadas funções para gerar grafos direcionados e não-direcionados de tamanho aleatório. Dessa forma, essas funções permitem que novos grafos sejam criados para análise a cada execução do código.

2.3.1. Pseudocódigo para Criar Grafos Direcionados Aleatórios

Algorithm 7 Grafo Direcionado Aleatório

Require: Um número aleatório de vértices $maxVertices$
  Gera um número de vértices $numVertices$ aleatório entre 1 e $maxVertices$
  Cria um grafo direcionado com $numVertices$ vértices
  Gera um número $numEdges$ entre 0 e $numVertices \times (numVertices - 1)$
  for $i = 0$ to $numEdges - 1$ **do**
    Selecione um vértice $source$ aleatório entre 0 e $numVertices - 1$
    Selecione um vértice $destination$ aleatório entre 0 e $numVertices - 1$
    if $source \neq destination$ **then**
      Adicionar uma aresta do vértice $source$ para o vértice $destination$
    end if
  end for

2.3.2. Pseudocódigo para Criar Grafos Não-Direcionados Aleatórios

Algorithm 8 Grafo Não-Direcionado Aleatório

Require: Um número aleatório de vértices $maxVertices$
  Gera um número de vértices $numVertices$ aleatório entre 1 e $maxVertices$
  Cria um grafo direcionado com $numVertices$ vértices
  Gera um número $numEdges$ entre 0 e $(numVertices \times (numVertices - 1)) \div 2$
  for $i = 0$ to $numEdges - 1$ **do**
    Selecione um vértice $source$ aleatório entre 0 e $numVertices - 1$
    Selecione um vértice $destination$ aleatório entre 0 e $numVertices - 1$
    if $source \neq destination$ e $\neg\ (graph.hasEdge(source, destination))$ **then**
      Adicionar uma aresta do vértice $source$ para o vértice $destination$
    end if
  end for

3. Conclusão

Neste trabalho, foi explorado a manipulação de grafos direcionados e grafos-não direcionados e implementado os requisitos propostos, fornecendo uma estrutura eficiente para manipulação de grafos e permitindo análises variadas a cada execução por meio da criação de grafos aleatórios. Ademais, a modularidade das classes facilita tanto a manutenção quanto a expansão futura do código se exigido novos requisitos.