Grafos

Arthur Gregório Leal e Ian Guelman

Março 2021

1 Introducão

O trabalho fora desenvolvido usando a linguagem Java e seguindo explicações encontradas no trabalho de Rodrigues (2018), e implementando a estrutura de dados *Bidimensional Array*, também conhecida como matriz, mais especificamente: matriz de adjacência. O trabalho levou em consideração a construção de uma interface de linha de comando que permite a inserção de vértices rotulados, e não rotulados.

2 Metodologia

2.1 Codificação

Para facilitar o desenvolvimento, criou-se uma interface de linha de comando (CLI) que permite a inserção de vértices e arestas de forma facilitada. As matrizes foram implementas em três classes distintas: Matriz ponderada, Matriz ponderada não direcionada e Matriz não ponderada, todas implementam a interface: Matriz.

A interface Matriz exige a implementação dos seguintes métodos:

boolean addVertex(String vertex) que verifica e adiciona um novo vértice ao grafo (Matriz).

public boolean addEdge(String vertex,String vertex2, String label) que verifica e adiciona uma nova aresta ao grafo (Matriz).

public void increaseMatrix() que verifica e incrementa a Matriz para suportar o grafo.

public boolean print() que imprime a Matriz de Adjacência como uma tabela, e ajuda-nos a entender o grafo.

public void fill(String string) que preenche os campos vazios com string vazias, um método meramente decorativo.

2.2 Interface de Usuário

Para permitir e validar múltiplas situações na construção de grafos fora feita uma interface de linha de comando chamada Graphg(Graph Graphical), aplicação tem como objetivo criar matrizes de adjacências que representam grafos direcionado e não direcionados, com e sem rótulos. A aplicação desenvolvida em Java possuí um pré-lançamento disponível no Github (<https://github.com/ogregorio/graphg>).

Na tabela a seguir estão listados os comandos principais aceitos.

Create: comandos de criação.	
CREATE WITH_LABEL	Cria um grafo ponderado direcionado.
CREATE NO_LABEL	Cria um grafo não-ponderado.
CREATE NO_LABEL_UNDIRECTED	Cria um grafo não-ponderado não-direcionado.
Add (No_Label): comandos de adição de elementos sem rótulo.	
ADD VERTEX [valor]	Adiciona um vértice dado um valor numérico.
ADD EDGE [vértice1] [vértice2]	Adiciona uma aresta dados dois vértices.
Add (Label): comandos de adição de elementos rotulados.	
ADD VERTEX [rótulo]	Adiciona um vértice dado um rótulo escrito.
ADD EDGE [vértice1] [vértice2] [rótulo]	Adiciona uma aresta dados dois vértices e um rótulo.
Count: comandos de contagem de elementos/eventos e interações.	
COUNT CYCLES	Conta o número total de ciclos do grafo.
COUNT CYCLES [tamanho]	Conta o número de ciclos do grafo de um dado tamanho.
Print: comandos de impressão.	
PRINT	Imprime o grafo com matriz de adjacência.
PGM: comandos de segmentação de imagem.	
PGM [origem] [destino]	Realiza a segmentação da imagem em plano de fundo e primeiro plano, utilizando fluxo de rede e corte mínimo
PGMALT [origem] [destino]	Realiza a segmentação da imagem em plano de fundo e primeiro plano, utilizando método procedural
PGMDEMO	Exibe exemplo dos dois métodos citados acima

2.3 Amostras

As imagens dos grafos amostrados nas figuras 1, 2, 3 foram extraídas diretamente do artigo de Rodrigues (2018) demonstração de um grafo direcionado ponderado. Ao lado das representações dos grafos, consta as capturas da saída do código obtido em relação ao grafo respectivo.

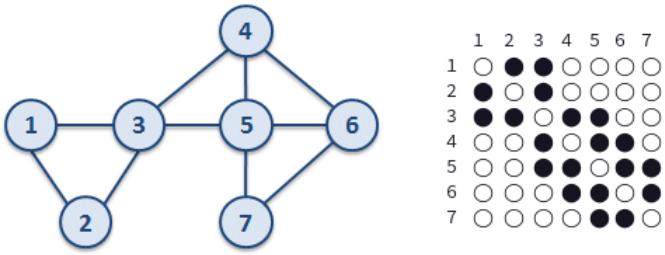

Figure 1: Grafo não-direcionado.

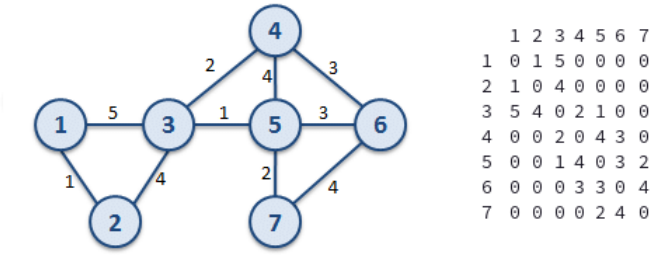

Figure 2: Grafo não-direcionado ponderado.

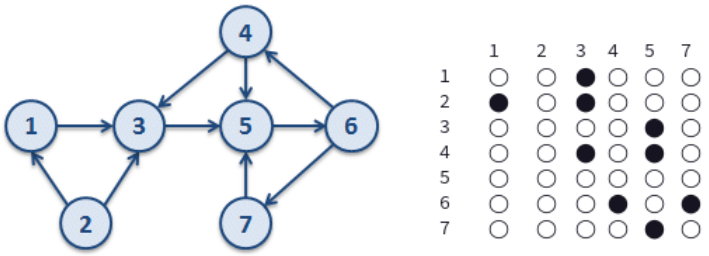

Figure 3: Grafo direcionado não-ponderado.

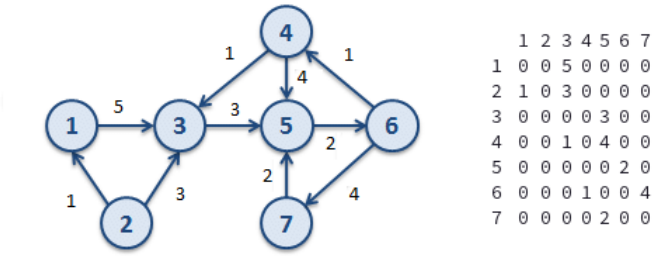

Figure 4: Grafo direcionado ponderado.

3 Ciclos

Um ciclo em um grafo não-direcionado simples é um caminho fechado sem vértices repetidos. Assim, um ciclo se dá pela sequência$(v_0, a_1, v_1, a_2, v_2, \cdots, a_k, v_k)$ com k > 2 em que $v_k = v_0$ mas $v_0, v_1, v_2, \cdots, v_{k1}$ são distintos dois a dois.

O cálculo do número de ciclos em um grafo pode ser realizado utilizando-se de alguns métodos, destaca-se: caminhamento e permutação. A análise nesse documento refere-se a essas duas respectivas formas de implementação.

3.1 Caminhamento

O caminhamento foi implementado utilizando um algoritmo guloso.

Para execução do algoritmo, utiliza-se uma matriz de adjacência, da qual extraímos a sua dimensão para compor um vetor de marcação (booleano) que servirá para gravar aqueles vértices que já foram visitados.

Para cada vértice da matriz, realizamos uma busca em profundidade através do algoritmo DFS (*Depth-first search*), em busca de um ciclo de tamanho K. Ao encontra-lo, incrementamos o marcador de ciclos. Repetimos esse procedimento com o valor de K sendo inicializado como 3 e o incrementamos por 1 até que seu valor seja igual ao tamanho da matriz. Por fim, devolvemos o marcador a *Command Line Interface* através da chamada *CYCLE COUNT*.

A complexidade teórica do algoritmo desenvolvido, em notação *Big O*, é de O(N^3) para se calcular todos os possíveis ciclos de quaisquer tamanhos, e de O(N^2) para calcular o numero de ciclos de um tamanho K dado, sendo N o número de vértices do grafo.

3.2 Permutação

A permutação trata-se de uma abordagem que utiliza de um algoritmo de força bruta. Para tal, cada vértice é analisado e são calculados todos os caminhos proveniente deste de si próprio e de seus subsequentes, esta ação é repetida para cada nó do grafo. Sendo assim, esta abordagem representa um algoritmo de complexidade teórica, em notação *Big O*, de O(N!), sendo N o número de vértices do grafo.

4 Operações de PGM

De acordo com Poskanzer (1991) na documentação oficial do PGM, PGM é um acrônimo derivado de "Portable Gray Map", e é um formato de arquivo em tons de cinza de menor denominador comum. Ele é projetado para ser extremamente fácil de aprender e escrever programas para. Uma imagem PGM representa uma imagem gráfica em tons de cinza. Uma imagem PGM pode ser considerada apenas uma matriz de inteiros arbitrários.

4.1 Fluxo em Rede

Primeiramente, transformamos a matriz que representa uma imagem PGM em um grafo de fluxo de rede, a partir dele utilizamos o algoritmo de fluxo máximo para gerar um corte mínimo. Dado o corte mínimo, separamos os vertices pertencentes a eles e o destacamos na matriz inicial.

4.2 Método procedural

Através do método procedural, percorremos uma matriz que representa uma imagem PGM qualquer, e detectamos o elemento que vem a se repetir na maior quantidade possível, a ele atribuímos a propriedade de *background*, posteriormente, atribuimos um valor que o distingue de todo os demais *pixels* da imagem (o *foreground*). Nomeamos esse método de highlight, que é de atribuir uma cor de muito mais elevada que a do segundo pixel de valor mais alto da imagem.

References

POSKANZER, J. *pgm*. 1991. Disponível em: <http://netpbm.sourceforge.net/doc/pgm.html>. page.66

RODRIGUES, P. G. *Extração de Características em Interfaces Cérebro-Máquina Utilizando Métricas de Redes Complexas*. Tese (Doutorado), 02 2018. page.11, page.44