Pedro Inácio Rodrigues Pontes

Prática 14: Grafos
Laboratório de AEDS

Belo Horizonte, Brasil

2024

1 Introdução

Esta prática teve os seguintes imperativos como objetivos no quesito prático:

1. Implemente o algoritmo de Dijkstra para encontrar o menor caminho entre dois vértices. O retorno deve ser um array com a sequência de vértices a ser percorrida
2. Atualize a função desenhar para receber um array de vértices e, ao desenhar o grafo, caso a aresta desenhada seja composta por vértices em posições consecutivas no array, essa aresta deve ser desenhada em vermelho.

Dentro do quesito do apredizado, o objetivo foi implementar os conhecimentos adquiridos sobre o Algoritmo de Dijkstra e lógica da programação, além de fomentar as habilidades de interpretação de código.

O código em no qual foi necessário implementar os imperativos era um modelo visual de grafos, o qual possuia as classes main e grafo. Ele já possuia o algoritmo de Dijkstra semi-implementado, faltando apenas a criação do array de vértices a ser percorrido para o menor caminho de certa origem a certo destino.

2 Desenvolvimento

Para concluir os objetivos em relação ao código dado, foi utilizada a estrátegia de concluir cada parte separadamente e em ordem. Assim, na primeira versão do código foi feita a resolução do problema 1, e na segunda e última versão foi feita a resolução do problema 2.

2.1 Versão 1

Com vista ao comando "Implemente o algoritmo de Dijkstra para encontrar o menor caminho entre dois vértices. O retorno deve ser um array com a sequência de vértices a ser percorrida", foram implementados os seguintes trechos de código:

2.1.1 Classe Grafo

```
+    import java.util.List;

-----------------------------------------

-    void dijkstra(int origem, int destino){
+    ArrayList<Integer> dijkstra(int origem, int destino){
```

```
----------------------------------------

+    ArrayList <Integer> menorCaminho = new ArrayList<>();
+    for(int i = destino; i != -1; i = anterior[i]){
+        menorCaminho.add(i);
+    }
+
+    for(int i =0; i< menorCaminho.size(); i++)
+        print(menorCaminho.get(i) + " ");
+    print("\n");
+
-    return;
+    return menorCaminho;

----------------------------------------

-        for(int i =0; i< menorCaminho.size(); i++)
-            print(menorCaminho.get(i) + " ");
-        print("\n");
```

2.1.2 Classe main

```
+  grafo.dijkstra(0,8);
```

2.1.3 Explicação

Na classe grafo, foi importada a biblioteca java.util.List, para permitir o uso de ArrayLists, estas foram escolhidas para guardar o array do menor caminho pelo fato de serem dinamicamente alocadas, com tamanho variável de acordo com o número de elementos inseridos, além disso, apresentam uma boa eficiência em tempo de execução e memória usada. Foi alterado o corpo da função dijkstra para ela retornar ArrayLists. Foi criada a ArrayList menorCaminho, onde são armazenados os vértices para o menor caminho de uma dada origem a um dado destino. menorCaminho é preenchido a partir de um for que percorre o resultado do algoritmo de dijkistra do destino até a origem.

O for funciona assim: i é inicializado com o valor do destino, que é o primeiro a ser adicionado na ArrayList. Após isso, o valor é atualizado para anterior de i, que é o vértice com menor caminho até o vértice i. Essa propriedade anterior está dentro da função dijkstra. O for continua até i ser igual a -1. Isso ocorrerá quando a origem for alcançada, pois no código do algoritmo, o anterior do vértice origem é -1. Assim,

o array menorCaminho é preenchido corretamente, contudo, ele é ordenado na direção Destino->Origem

A seção do for que itera o menorCaminho o printa é feita para o teste do funcionamento correto do código, tal seção também ajudou no teste do código implementado na parte 2.

Na main, foi apenas chamado a função dijkstra para verificar funcionamento do código implementado.

2.2 Versão 2

Com vista ao comando "Atualize a função desenhar para receber um array de vértices e, ao desenhar o grafo, caso a aresta desenhada seja composta por vértices em posições consecutivas no array, essa aresta deve ser desenhada em vermelho.", foram implementados os seguintes trechos de código:

2.2.1 Classe Grafo

```
-    void desenhar() {
+    void desenhar(ArrayList<Integer> caminho) {

---------------------------

+        //Desenha as arestas do menor caminho em vermelho
+        stroke(255, 0, 0);
+        for (int i = 0; i < caminho.size() - 1; i++) {
+            int atual = caminho.get(i);
+            int proximo = caminho.get(i + 1);
+            strokeWeight(matrizAdj[atual][proximo]);
+            line(posicoes[atual].x, posicoes[atual].y, posicoes[proximo].x, posico
+        }

---------------------------

-        for(int i =0; i< menorCaminho.size(); i++)
-            print(menorCaminho.get(i) + " ");
-        print("\n");
```

2.2.2 Classe main

```
-  grafo.desenhar();
```

```
-  grafo.dijkstra(0,8);
+  grafo.desenhar(grafo.dijkstra(0,8));
```

2.2.3 Explicação

Primeiramente, foi atualzada a função desenhar para ela receber um array de vértices - nesse caso foi uma ArrayLIst -. Após, foi implementado o trecho de código responsável por desenhar os vértices consecutivos do array em vermelho. O funcionamento é simples:

Stroke é definido com o RGB do vermelho, é criado um loop for que começa do 0 e é incrementado de um em um enquanto ele for menor que o tamanho do array. São criadas os ints atual e proximo, os quais recebem o elemento i do caminho e o elemento posterior a este, respectivamente. É reutilizada a parte da função desenhar que define a grossura dos vértices, *strokeWeight(matrizAdj[atual][proximo])*, e é criada uma linha com as posições iniciais x e y do elemento atual e as posições finais x e y do elemento posterior a tal. Esta linha de código também foi reutilizada da função desenhar, mas com mínimas alterações (i -> atual, j-> próximo).

Foi retirada a parte em que era printada a ArrayLIst menorCaminho, por, como já explicado, ela apenas ter servido de auxiliar para os testes do funcionamento correto do programa.

Na classe main, desenhar recebeu o array dijkstra como parâmetro, assim, o menor caminho de certa origem a certo destino será desenhado (0 a 8, no caso).

3 Resultados

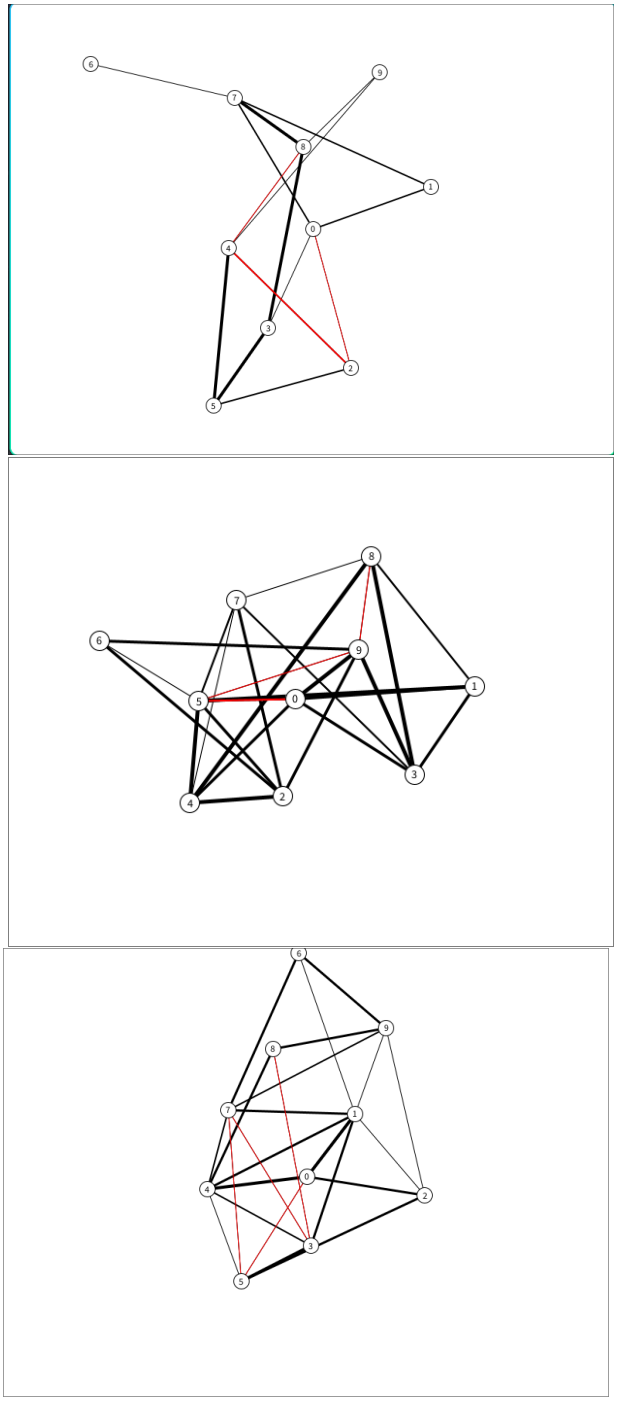

Figura 1 – Visulização do Algoritmo de Dijkstra encontrando o menor caminho entre os vértices 0 e 8 (*colorido em vermelho*)

Os resultados alcançaram todos os objetivos especificados, alcançando a correta implementação do algoritmo de Dijkstra e o correto destaque em vermelho do caminho do array de vértices fornecido à função *desenha*.

4 Conclusão

- Os resultados foram totalmente alcançados e estão de acordo com o esperado.
- Implementar o código para colorir os trechos em vermelho foi particularmente difícil, por ter havido a tentativa de reaproveitar o código que criava as arestas nele. A questão é que ele não funcionava tão bem no contexto da ArrayLIst utilizada para armazenar os vértices. Entender quais parâmetros usar no *for* para guardar o array com o menor caminho também foi complicado, por necessitar de um entendimento quase pleno do restante da função do algoritmo de Dijkstra.
- Foi aprendida a lógica do algoritmo de Dijkstra, como lidar com os resultados dele (para criar o array menorCaminho) e leitura de código (para conseguir entender o que estava sendo feito e como implementar as funcionalidades desejadas).