Vítor Paixão Damasceno

Prática 14
Laboratório de AEDS

Belo Horizonte, Brasil

2024

1 Introdução

Neste laboratório, exploramos a aplicação do algoritmo de Dijkstra em grafos. Partimos de uma classe `Grafo` já implementada e uma simulação que imprime o grafo na tela. Nossos objetivos foram:

- Implementar o algoritmo de Dijkstra, conforme o pseudocódigo fornecido.
- Modificar o algoritmo para retornar um array contendo os vértices do menor caminho entre a origem e o destino.
- Alterar a função `desenhar()` para destacar o caminho mínimo em vermelho na tela.

Concluindo essas etapas, buscamos demonstrar visualmente os resultados do algoritmo, aprofundando o entendimento sobre grafos e algoritmos de menor caminho.

2 Desenvolvimento

Inicialmente, modificamos a simulação para que o grafo fosse estático na tela, eliminando o movimento das arestas em tempo de execução. Transferimos os métodos `atualizar()` e `desenhar()` da função `draw()` para a função `setup()`, rodando `atualizar()` em um loop antes da primeira exibição. A cor de fundo foi alterada para azul, conforme o código abaixo:

```
for (int i = 0; i < 500; i++)
    g.atualizar();

background(#D3EEFF);
g.desenhar(g.dijkstra(0,8));
```

A implementação do método `dijkstra()` foi baseada no pseudocódigo fornecido, conforme segue:

```
int[] dijkstra(int origem, int destino){
    int[] dist = new int[numVertices];
    int[] anterior = new int[numVertices];

    for (int v = 0; v < numVertices; v++) {
        dist[v] = 1000000;
        anterior[v] = -1;
    }

    dist[origem] = 0;
```

```
11      int[] Q = new int[numVertices];
12      for (int k = 0; k < numVertices; k++) {
13          int u = -1;
14          int udist = 1000000000;
15          for (int v = 0; v < numVertices; v++) {
16              if (Q[v] == 0 && dist[v] < udist) {
17                  u = v;
18                  udist = dist[v];
19              }
20          }
21
22          Q[u] = 1;
23
24          for (int v = 0; v < numVertices; v++) {
25              if (u == v || matrizAdj[u][v] == 0) continue;
26
27              int alt = udist + matrizAdj[u][v];
28
29              if (alt < dist[v]) {
30                  dist[v] = alt;
31                  anterior[v] = u;
32              }
33          }
34      }
35
36      int[] caminho = new int[numVertices];
37      for (int v = destino, i = numVertices - 1; v != -1; v =
            anterior[v], i--) {
38          caminho[i] = v;
39      }
40
41      return caminho;
42  }
```

Para a função `desenhar()`, alteramos os parâmetros para incluir o array `caminho[]` e utilizamos variáveis booleanas para controlar o destaque das arestas e vértices no menor caminho, conforme segue:

```
1   // Desenhar arestas
2   for (int i = 0; i < numVertices; i++) {
3       for (int j = i + 1; j < numVertices; j++) {
4           boolean arestaNoCaminho = false;
```

```
5           for (int k = 0; k < caminho.length - 1; k++) {
6               if ((caminho[k] == i && caminho[k + 1] == j) ||
7                   (caminho[k] == j && caminho[k + 1] == i)) {
8                   arestaNoCaminho = true;
9                   break;
10              }
11          }
12
13          if (arestaNoCaminho) {
14              stroke(255, 0, 0); // Vermelho para o caminho
15          } else {
16              stroke(0); // Preto caso contrario
17          }
18
19          strokeWeight(matrizAdj[i][j]);
20          if (matrizAdj[i][j] > 0) {
21              line(posicoes[i].x, posicoes[i].y, posicoes[j].x,
                    posicoes[j].y);
22          }
23      }
24  }
25
26  // Desenhar vertices
27  for (int i = 0; i < numVertices; i++) {
28      boolean verticeNoCaminho = false;
29      for (int k = 0; k < caminho.length; k++) {
30          if (caminho[k] == i) {
31              verticeNoCaminho = true;
32              break;
33          }
34      }
35
36      if (verticeNoCaminho) {
37          fill(255, 0, 0); // Vermelho para caminho
38      } else {
39          fill(#529BFF); // Azul caso contrario
40      }
41
42      stroke(1);
43      strokeWeight(2);
44      ellipse(posicoes[i].x, posicoes[i].y, raio * 2, raio * 2);
45      fill(0);
```

```
46      textSize(20);
47      text(str(i), posicoes[i].x, posicoes[i].y + 7);
48  }
```

3 Resultados

Após as alterações, o grafo e o algoritmo de Dijkstra foram representados visualmente conforme as imagens abaixo.

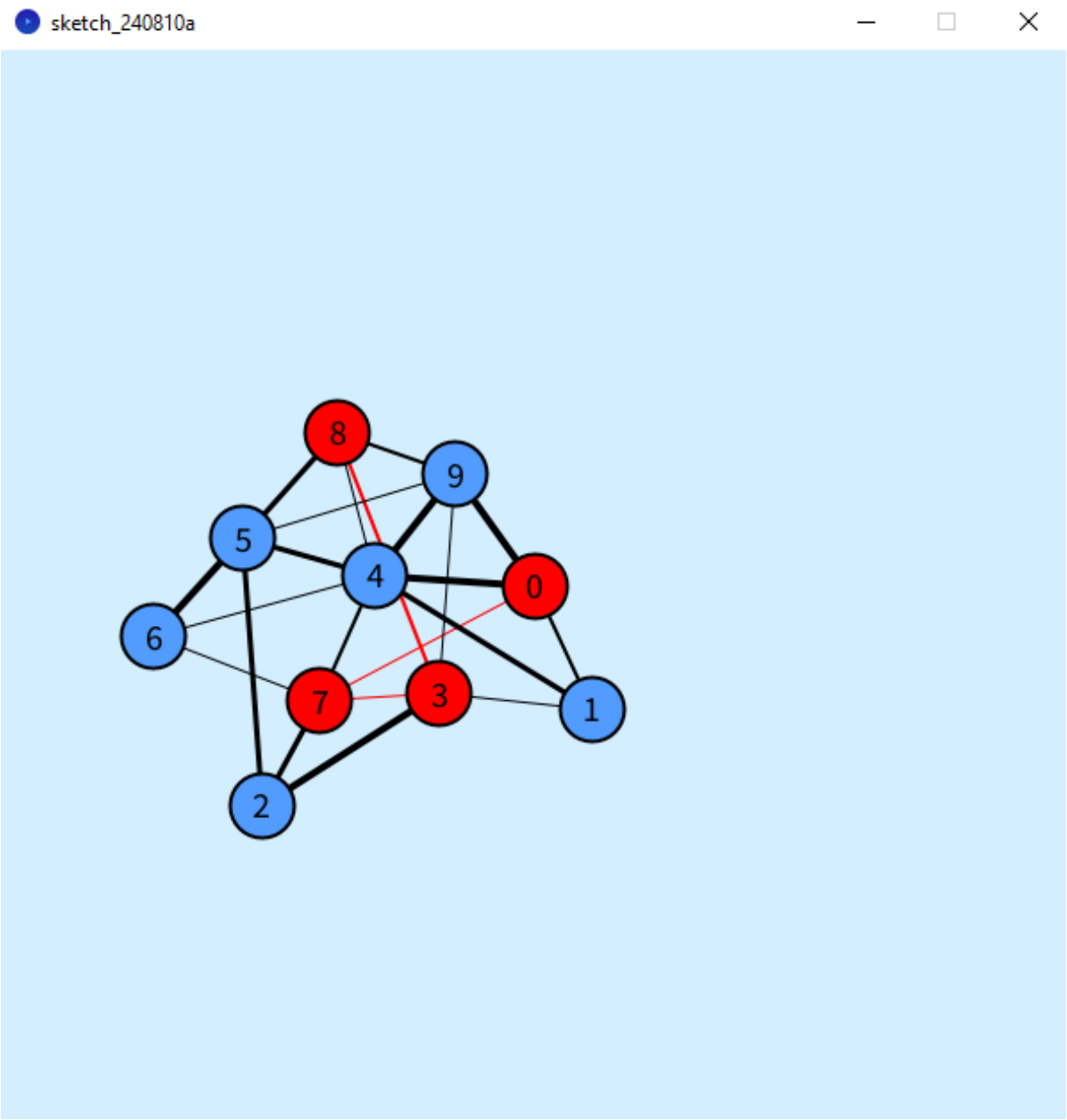

Figura 1 – Grafo com caminho mínimo destacado

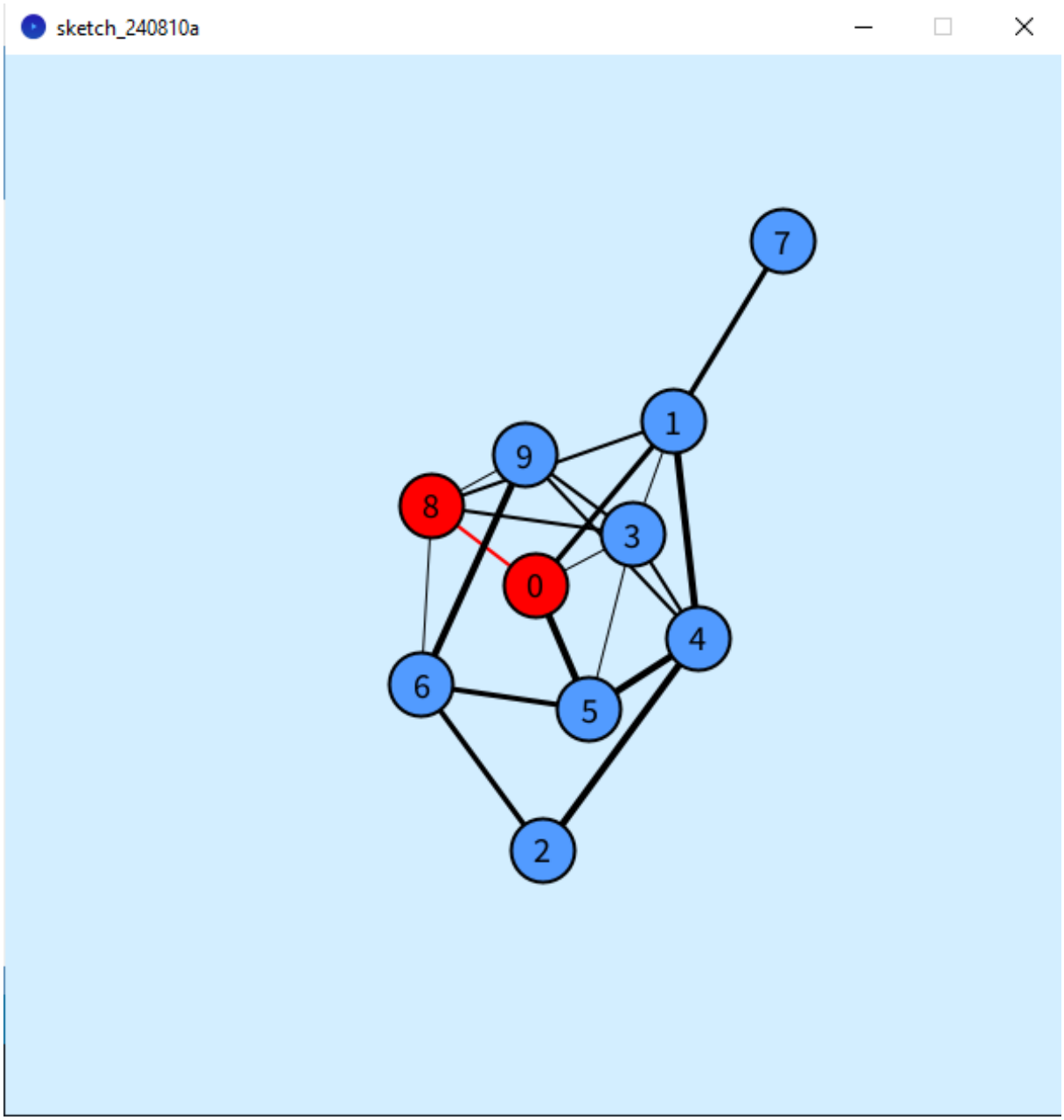

Figura 2 – Outra vista do grafo

4 Conclusão

Os objetivos do experimento foram alcançados com sucesso. O principal desafio foi implementar a lógica para identificar as arestas no caminho mínimo, exigindo várias tentativas. A prática foi valiosa, proporcionando um entendimento completo do algoritmo de Dijkstra e grafos.