

Instituto de Ciências Matemáticas
e de Computação
Departamento de Ciências de Computação
SCC0503 - Algoritmos e Estruturas de Dados II

Relatório Exercício 04

Alunos:

Heitor Pupim A. Toledo n^oUSP: 11372858

Pedro Zambrozi Brunhara - n^oUSP: 12608664

Professor: Leonardo Tórtoro Pereira

Junho
2022

Conteúdo

1	Introdução	1
2	Desenvolvimento	2
3	Resultados	4
3.1	Entrada 1	4
3.2	Entrada 2	4
3.3	Entrada 3	5
3.4	Entrada 4	5
4	Referências	7

1 Introdução

O exercício resolvido pede que dados de quests(missões) de um jogo, um nome(String), uma descrição(String) e um ID(int), sejam lidos. Depois, um dígrafo contendo essas informações seja construído, usando a representação que preferir e as arestas terão peso 1. Por fim, executar uma busca em profundidade a partir de um vértice passado na entrada e imprimir o resultado na tela.

A entrada é dados da entrada padrão (System.in). Primeiro será passado o Número de vértices que serão lidos a seguir (int), depois será passado Nome da quest (String) e Descrição da quest (String) de cada vértice, após isso Número de arestas que serão lidas a seguir (int) , que é a quantidade de vezes que o ID do vértice de origem da aresta(int) e o ID do vértice de destino da aresta(int) serão passados, por fim será passado o ID do vértice inicial da travessia (int).

Ao fim da execução, o programa de imprimir na saída padrão (System.out) o caminho feito pela busca em profundidade ea impressão deve conter o nome, descrição e id das quests no seguinte formato:

```
Quest{\n\tID= 'questID'\n\tname= 'questName'\n\tdescription= 'questDescription'\n}
```

2 Desenvolvimento

Para a realização deste trabalho os códigos fornecidos pelo Prof. Leonardo Tórtoro Pereira foram usados como base, para ser mais preciso AbstractGraph.java, DigraphList.java, Edge.java, GraphInterface.java, TraversalStrategy.java, Vertex.java foram usados na íntegra e sem alteração, já o BreadthFirstTraversal.java foi usado de base para a criação do BuscaEmProfundidade.java.

Foi escolhido o dígrafo de lista, pois esse tipo de dígrafo permite aumentar o número de vértices sem muita dificuldade, além de ser $O(1)$ para achar o primeiro adjacente e o próximo adjacente.

Para transformar o código de busca em largura, imagem 1, em busca em profundidade, imagem 2, foi preciso criar uma recursão que engloba o while que para quando a lista vertexesToVisit está vazia (linha 26 a 44 da imagem 1) que é chamado após o adjacentVertex ser adicionado a vertexesToVisit (linha 39 da imagem 1), assim, gerando o código da imagem 2.

Desse modo, quando um vértice adjacente (VA) ao vértice raiz (VR) é adicionado à lista vertexesToVisit, a recursão faz com que, se o VA possuir quests adjacentes, elas sejam inseridas e não as adjacentes ao VR, mas se o VA não possuir adjacentes, a recursão retorna e depois de retornar a próximo adjacente ao VR é adicionado.

```
21 Queue<Vertex> vertexesToVisit = new LinkedList<>();
22 vertexesToVisit.add(source);
23
24 Vertex currentVisitedVertex;
25 int currentVisitedVertexIndex;
26 while(!vertexesToVisit.isEmpty())
27 {
28     currentVisitedVertex = vertexesToVisit.poll();
29     currentVisitedVertexIndex = getGraph().getVertices().indexOf(currentVisitedVertex);
30     if (currentVisitedVertex != null)
31     {
32         var adjacentVertex = getGraph().getFirstConnectedVertex(currentVisitedVertex);
33         while(adjacentVertex != null)
34         {
35             int adjacentVertexIndex = getGraph().getVertices().indexOf(adjacentVertex);
36             if(!hasVertexBeenVisited(adjacentVertexIndex))
37             {
38                 updateTraversalInfoForVertex(adjacentVertexIndex, currentVisitedVertexIndex);
39                 vertexesToVisit.add(adjacentVertex);
40             }
41             adjacentVertex = getGraph().getNextConnectedVertex(currentVisitedVertex, adjacentVertex);
42         }
43     }
44 }
45 printPath();
46 }
```

Figura 1: Imagem 1

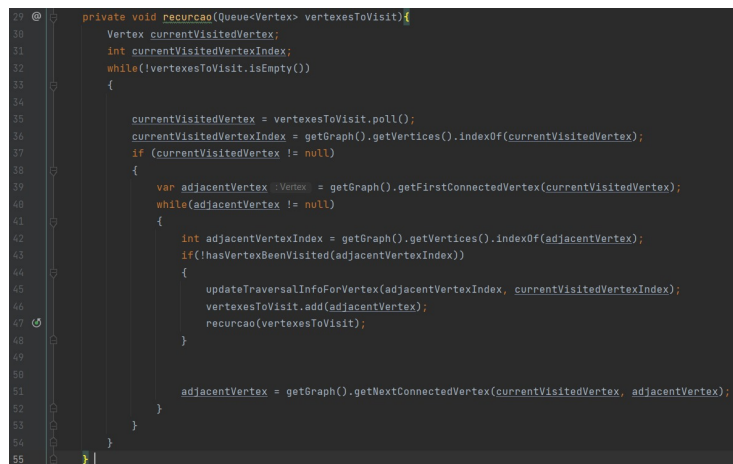
```
Queue<Vertex> vertexesToVisit = new LinkedList<>();
vertexesToVisit.add(source);

Vertex currentVisitedVertex;
int currentVisitedVertexIndex;
while(!vertexesToVisit.isEmpty()){
    currentVisitedVertex = vertexesToVisit.poll();
    currentVisitedVertexIndex = getGraph().getVertices().indexOf(currentVisitedVertex);
    if (currentVisitedVertex != null){
        var adjacentVertex = getGraph().getFirstConnectedVertex(currentVisitedVertex);
        while(adjacentVertex != null){
            int adjacentVertexIndex = getGraph().getVertices().indexOf(adjacentVertex);
```

```

        if(!hasVertexBeenVisited(adjacentVertexIndex)){
            updateTraversalInfoForVertex(adjacentVertexIndex, currentVisitedVertexIndex);
            vertexesToVisit.add(adjacentVertex);
        }
        adjacentVertex = getGraph().getNextConnectedVertex(currentVisitedVertex, adjacentVertex);
    }
}
printPath();
}

```



```

17 private void recursao(Queue<Vertex> vertexesToVisit){
18     Vertex currentVisitedVertex;
19     int currentVisitedVertexIndex;
20     while(!vertexesToVisit.isEmpty()){
21         {
22             currentVisitedVertex = vertexesToVisit.poll();
23             currentVisitedVertexIndex = getGraph().getVertices().indexOf(currentVisitedVertex);
24             if (currentVisitedVertex != null)
25             {
26                 var adjacentVertex :Vertex = getGraph().getFirstConnectedVertex(currentVisitedVertex);
27                 while(adjacentVertex != null)
28                 {
29                     int adjacentVertexIndex = getGraph().getVertices().indexOf(adjacentVertex);
30                     if(!hasVertexBeenVisited(adjacentVertexIndex))
31                     {
32                         updateTraversalInfoForVertex(adjacentVertexIndex, currentVisitedVertexIndex);
33                         vertexesToVisit.add(adjacentVertex);
34                         recursao(vertexesToVisit);
35                     }
36                 }
37                 adjacentVertex = getGraph().getNextConnectedVertex(currentVisitedVertex, adjacentVertex);
38             }
39         }
40     }
41 }
42 }
43 }
44 }
45 }

```

Figura 2: Imagem 2

```

private void recursao(Queue<Vertex> vertexesToVisit){
    Vertex currentVisitedVertex;
    int currentVisitedVertexIndex;
    while(!vertexesToVisit.isEmpty()){
        currentVisitedVertex = vertexesToVisit.poll();
        currentVisitedVertexIndex = getGraph().getVertices().indexOf(currentVisitedVertex);
        if (currentVisitedVertex != null){
            var adjacentVertex = getGraph().getFirstConnectedVertex(currentVisitedVertex);
            while(adjacentVertex != null){
                int adjacentVertexIndex = getGraph().getVertices().indexOf(adjacentVertex);
                if(!hasVertexBeenVisited(adjacentVertexIndex)){
                    updateTraversalInfoForVertex(adjacentVertexIndex, currentVisitedVertexIndex);
                    vertexesToVisit.add(adjacentVertex);
                    recursao(vertexesToVisit);
                }
                adjacentVertex = getGraph().getNextConnectedVertex(currentVisitedVertex, adjacentVertex);
            }
        }
    }
}

```

3 Resultados

3.1 Entrada 1

A saída printada do caminho feito pela busca proveniente da entrada 1.in é igual a saída esperada do 1.out, e a imagem do dígrafo é parecida com a esperada, só diferenciando a posição da quest Slime's Revenge, que em vez de o filho do meio da quest Slime Killer, ele é o filho mais a esquerda, essa mudança é apenas estética e não muda o significado do Digrafo.

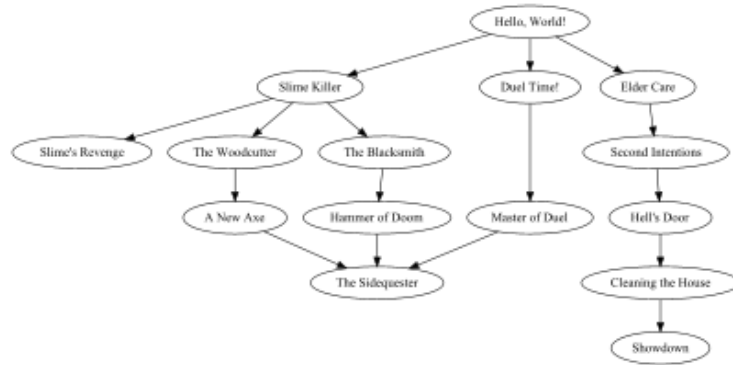


Figura 3: Digrafo 1

3.2 Entrada 2

A saída printada do caminho feito pela busca da entrada 2.in foi diferente do exemplo do professor, mas não está errado. A saída printada foi:

Slime Killer, Slime's Revenge, The Woodcutter, A New Axe, The Blacksmith, Hammer of Doom Essa diferença ocorreu porque depois de passar por Slime Killer a função `getFirstConnectedVertex` retornou a quest The Woodcutter e seguiu até a folha desse galho, depois a função `getNextConnectedVertex` retornou The Blacksmith e seguiu até a folha desse galho e, por fim, a função `getNextConnectedVertex` retornou Slime's Revenge chegando ao fim da execução.

Já no meu código a função `getFirstConnectedVertex` retornou Slime's Revenge, depois a função `getNextConnectedVertex` retornou The Woodcutter e na segunda vez que foi chamada retornou The Blacksmith. Assim, só é diferente, pois as funções retornam quests diferentes, o que não é um erro.

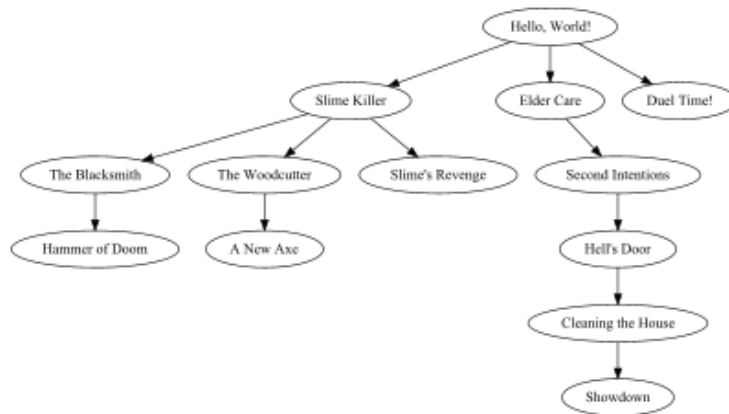


Figura 4: Digrafo 2

3.3 Entrada 3

A saída printada do caminho feito pela busca da entrada 3.in foi diferente do exemplo do professor, mas não está errado. A saída printada foi:

Hello, World!, Slime Killer, Slime's Revenge, The Woodcutter, A New Axe, The Sidequester, The Blacksmith, Hammer of Doom, Duel Time!, Master of Duel, Elder Care, Second Intentions, Hell's Door, Cleaning the House, Showdown Essa diferença ocorre pelo mesmo motivo que ocorreu na entrada 2, a função `getFirstConnectedVertex` retornou uma quest diferente da que está no exemplo do professor, o que não faz o caminho errado apenas diferente.

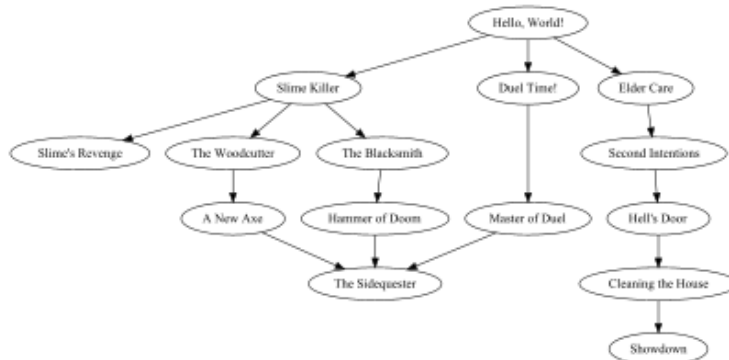


Figura 5: Digrafo 3

3.4 Entrada 4

A saída printada do caminho feito pela busca da entrada 4.in foi diferente do exemplo do professor, mas não está errado. A saída printada foi: Hello, World!, Slime Killer, Slime's Revenge, The Woodcutter, A New Axe, The Blacksmith, Hammer of Doom, Duel Time!, Elder Care, Second Intentions, Hell's Door, Cleaning the House, Showdown. Essa diferença ocorre pelo mesmo motivo que

ocorreu na entrada 2, a função `getFirstConnectedVertex` e `getNextConnectedVertex` retornou uma quest diferente da que está no exemplo do professor, o que não faz o caminho errado apenas diferente.

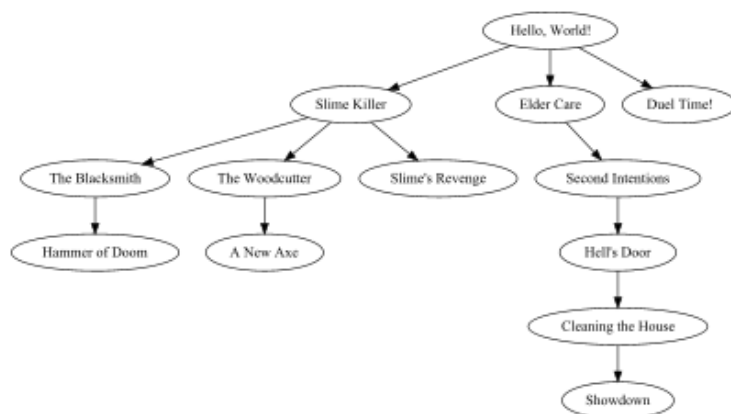


Figura 6: Dagrafo 4

4 Referências

Referências

Link contendo alguns materiais de grafos do professor Leonardo Tortoro:
[Github/LeonardoTPereira/Graph2022](https://github.com/LeonardoTPereira/Graph2022)