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

Relatório Exercício 04

Alunos: João Otávio da Silva, Leonardo Gonçalves Chahud
Professor: Leonardo Tórtoro Pereira

Junho
2022

Conteúdo

1 Introdução

O exercício proposto possui como objetivo principal a construção de um grafo direcionado, sem peso, para a representação de conexões entre *quests* de um jogo, bem como a simulação de caminhos (possíveis ordens de conclusão) através das mesmas, por meio da travessias em profundidade no grafo.

Ao final, para quatro casos de teste, foram obtidos os caminhos da seguinte forma: para cada vértice visitado foram impressas as informações (ID, Nome e Descrição) da *quest* correspondente, na ordem do algoritmo de busca em profundidade.

2 Desenvolvimento

2.1 Dígrafo

A representação utilizada foi a implementação de dígrafo com lista de adjacência, a mesma vista em em aula, *DigraphList*. Com uma única alteração no dado presente nos vértices, que será citada posteriormente.

- **Vértice:** Elemento fundamental do grafo, possui um dado, no caso desse exercício, uma *quest*.
- **Aresta:** Conexão entre dois vértices, possui a referência para o vértice de destino e um peso, no caso desse exercício, todos os pesos são 1.
- **Lista de Vértices:** Uma lista que contém as referências para todos os vértices pertencentes ao grafo.
- **Lista de Adjacência:** Uma lista onde para cada vértice há outra lista, contendo referências para todos as arestas que saem do mesmo.

2.2 Alteração no Vértice

Anteriormente havia uma *string* representando o vértice, agora, cada vértice passa a ter uma instância de uma classe arbitrária que implementa a interface *Data*. Essa, por sua vez, possuí a assinatura de duas funções:

- **stringToTerminal:** Exige uma *string* que será utilizada para impressões no terminal, por exemplo, o caminho das travessias.
- **stringToGraphviz:** Exige uma *string* que será utilizada para impressão da imagem do grafo pela biblioteca *graphviz*. Idealmente é uma representação mais mais concisa do dado, por exmplo, o título da *quest*.

```
public interface Data {
    String stringToTerminal();
    String stringToGraphviz();
}
```

Figura 1: Interface Data

Dessa forma, qualquer classe externa pode utilizar essa representação de grafo, bastando para isso, implementar as duas funções da interface.

2.3 Busca em Profundidade

O algoritmo de Busca em Profundidade faz uso das seguintes estruturas de sua classe mãe *TraversalStrategy* (vista em aula):

- **visitedVertices:** Lista de booleanos, o valor da lista na posição i indica se o vértice de índice i foi visitado ou não.
- **traversalPath:** Lista de vértices, armazena a ordem em que os mesmos foram visitados, é utilizada para a impressão da travessia no terminal.

O algoritmo é implementado através das seguintes funções:

- **traverseGraph:** Função mais abstrata da busca. Chama a busca recursiva partindo do vértice de origem, recebido como parâmetro, e ao final imprime os vértices na ordem da busca através da lista *traversalPath:*.
- **traverseGraphRecursion:** Implementação de fato da busca. Começa marcando o vértice atual como visitado e adicionando-o na lista do caminho. Após isso, para cada vértice adjacente ao vértice atual, é verificado se o mesmo já foi visitado, se não, chama a função recursivamente passando ele como novo vértice atual.

```
public void traverseGraph(Vertex source) {
    traverseGraphRecursion(source);
    printPath();
}

private void traverseGraphRecursion(Vertex currentVisitedVertex) {
    int currentVisitedVertexIndex = getGraph().getVertices().indexOf(currentVisitedVertex);
    addToPath(currentVisitedVertex);
    markVertexAsVisited(currentVisitedVertexIndex);

    Vertex currentAdjacentVertex = getGraph().getFirstConnectedVertex(currentVisitedVertex);
    while (currentAdjacentVertex != null) {
        int currentAdjacentVertexIndex = getGraph().getVertices().indexOf(currentAdjacentVertex);
        if (!hasVertexBeenVisited(currentAdjacentVertexIndex)) {
            traverseGraphRecursion(currentAdjacentVertex);
        }
        currentAdjacentVertex = getGraph().getNextConnectedVertex(currentVisitedVertex, currentAdjacentVertex);
    }
}
```

Figura 2: Busca em Profundidade

2.4 Leitura e Inserção

1. É lido um inteiro n correspondente ao número de vértices.
2. São lidas e inseridas n *quests*. Nome e descrição são inseridos pelo usuário, o ID é o valor do iterador do *loop*.
3. É lido um inteiro m correspondente ao número de arestas.
4. São lidas e inseridas m arestas. O usuário passa os IDs das *quests* de origem e destino, respectivamente.
5. É lido o ID da *quest* de início da busca.

Os métodos de inserção de aresta e travessia, precisam da referência para os vértices para serem executados, porém, a entrada é recebida do usuário na forma do ID da *quest*. Portanto, a fim de facilitar a chamada desses métodos, durante a inserção dos vértices, são guardadas em um *HashMap* as relações *[Id:Vértice]*.

```
private void readVertices(int numberOfVertices) {
    for (int i = 0; i < numberOfVertices; i++) {
        String name = in.nextLine();
        String description = in.nextLine();

        Vertex newVertex = new Vertex(new Quest(name, description, i));
        idVertexMap.put(i, newVertex);
        graph.addVertex(newVertex);
    }
}

private void readEdges(int numberOfEdges) {
    for (int i = 0; i < numberOfEdges; i++) {
        int sourceId = in.nextInt();
        int destinationId = in.nextInt();
        graph.addEdge(idVertexMap.get(sourceId), idVertexMap.get(destinationId));
    }
}
```

Figura 3: Leitura de Vértices e Arestas

3 Resultados

Os números fora dos nós, representam a ordem de descoberta dos vértices.

3.1 Caso 1

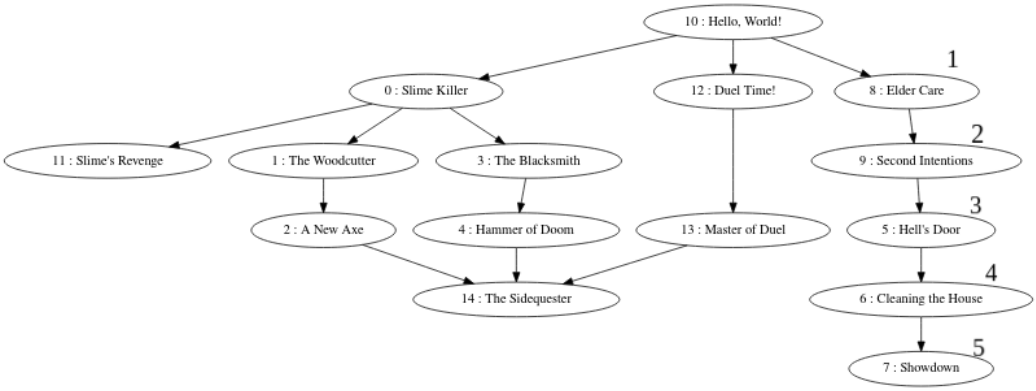

Figura 4: Grafo Caso 1

3.2 Caso 2

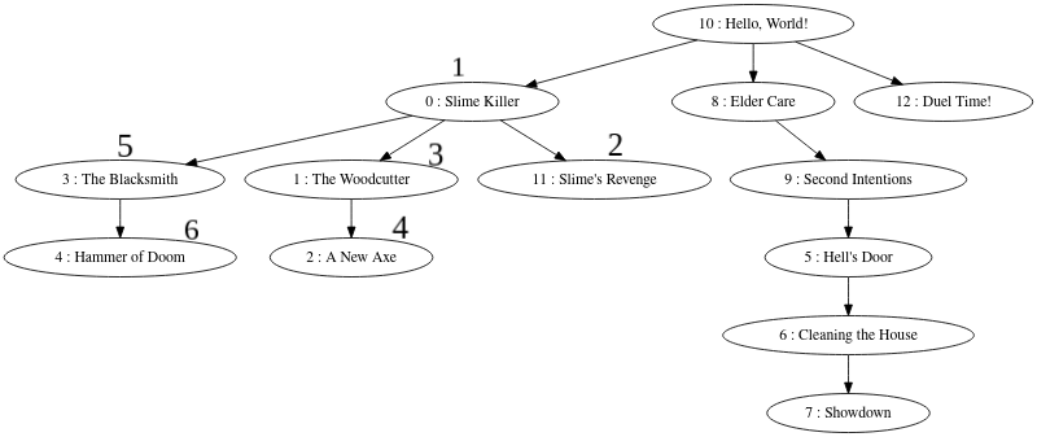

Figura 5: Grafo Caso 2

3.3 Caso 3

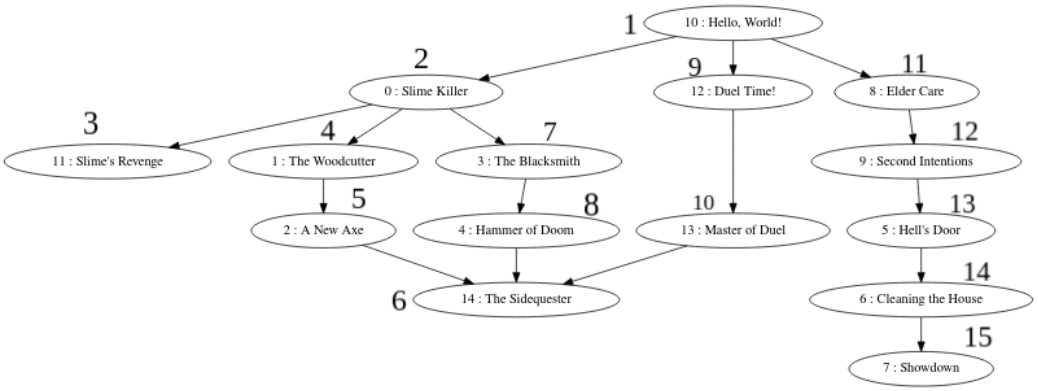

Figura 6: Grafo Caso 3

3.4 Caso 4

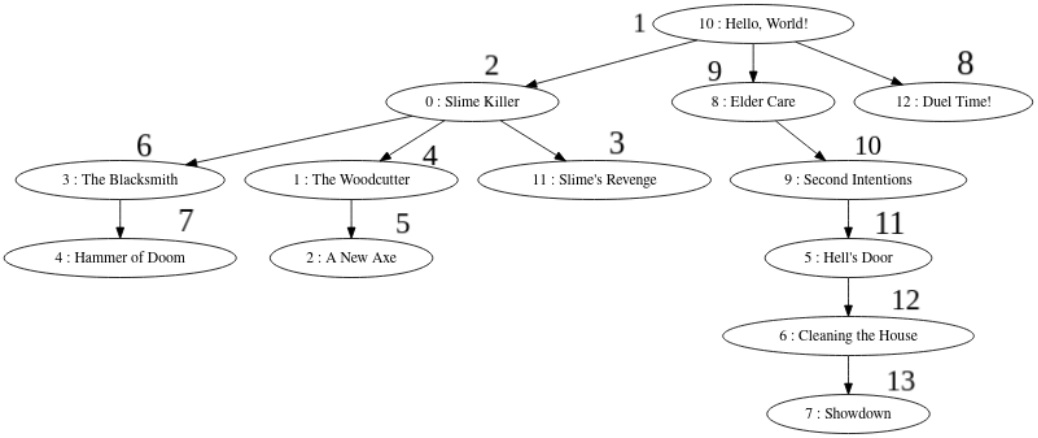

Figura 7: Grafo Caso 4

3.5 Finalização

Os resultados obtidos, mostram possíveis caminhos nos dígrafos, seguindo o algoritmo de busca em profundidade. Esses caminhos, são basicamente a ordem de descoberta dos vértices alcançáveis a partir da origem, seguindo a seguinte estratégia: sempre visitar o vértice mais profundo possível, depois ir retrocedendo e seguindo a busca da mesma forma.

No contexto desse exercício, por exemplo, esses caminhos poderiam representar a ordem que um jogador com um perfil de "avançar o máximo possível no jogo e depois voltar pra concluir o resto", completaria as *quets*.

Vale notar que alguns resultados diferem da resposta esperada. Isso ocorre pois, a depender da implementação, as funções *getFirstConnectedVertex* e *getNextConnectedVertex* podem retornar vértices diferentes, assim tornando os resultados de dois tipos de implementações, diferentes. Porém, os resultados obtidos são corretos e condizentes com a estratégia de travessia em profundidade.