Faculdade de Engenharia de Universidade do Porto

FEUP

Tema 4 – Deslocação do Herói no Labirinto

João Pedro Milano Silva Cardoso
200900579 - ee09063@fe.up.pt

Rui Filipe Laranjeira da Costa
199600952 – ei12150@fe.up.pt

Joel Alexandre Ezequiel Dinis
120509064 – ei12064@fe.up.pt

Relatório realizado no âmbito do Mestrado Integrado de Informática e Computação
Concepção e Análise de Algoritmos

28 de Abril de 2014

Índice de Conteúdos

Índice de Ilustrações

Introdução

O relatório apresentado foi realizado no contexto do 1º trabalho prático da Unidade Curricular de Concepção e Análise de Algoritmos.

Num jogo há um herói, uma espada e vários dragões localizados num labirinto.

O herói tem primeiro de encontrar a espada evitando os dragões (pois, caso se encontre com um dragão sem estar armado, morre).

De seguida, o herói deve dirigir-se aos dragões para os matar e finalmente deve dirigir-se para a saída.

O objectivo é determinar o caminho mais curto que deve ser seguido pelo herói (primeiro até à espada sem passar nos dragões e depois até à saída passando por todos os dragões), assumindo que os dragões são imóveis.

Numa segunda versão, assumir que os dragões se podem movimentar aleatoriamente. Neste caso, em cada jogada, replaneia-se a estratégia do herói considerando a posição corrente dos dragões.

O labirinto deve ser representado por um grafo planar, em que os vértices representam posições no labirinto (onde se podem localizar dragões, o herói, a espada e a saída) e as arestas representam movimentações unitárias possíveis.

Princípio de Funcionamento

Dados de Entrada e Construção de Grafo

Para a construção do grafo, o programa lê um ficheiro de texto, extensão .txt. Nesse ficheiro, o caractér X representa uma parede, N um espaço livre, E espada, D dragões, S a saída e H o herói.

O caracter ";" serve como separador.

Por exemplo, um labirinto de 7x8 poderia ser:

```
H;N;N;N;N;N;N;N;
N;X;X;N;X;N;X;E;
N;N;N;N;N;N;N;N;
D;X;X;N;X;N;X;N;
N;X;X;N;X;N;X;N;
N;N;N;N;N;N;N;N;
N;X;X;N;X;S;X;N;
```

Ilustração 1 – Exemplo de um Labirinto

Da leitura do ficheiro de texto obtém-se um grafo G(V,E), em que:

- $V = \{v_1, v_2, v_3, \dots, v_n\}$, em que $v_1, \dots, v_n$ são os vértices do grafo
- $E = \{e_1, e_2, e_3, \dots, e_k\}$, em que $e_1, \dots, e_k$ são as arestas do grafo

Cada vértice tem os seguintes atributos:

- Vector<Edge> adj – Vector onde são guardadas as arestas que têm esse vértice como origem.
- String space – Indica qual é o caractér que o vértice tem, indicando que tipo de nó é.
- Int dist – Distância do vértice a um determinado nó origem. Usado no cálculo do caminho mais curto.
- Int x, int y – Coordenadas do vértice.
- String Color – Cor do vértice. Usado para a representação usando Graph Viewer.

Cada aresta tem os seguintes atributos:

- Vertex dest* - Vértice que a aresta tem como destino.
- Double weight – Peso da aresta. Na nossa aplicação, todas as arestas têm peso 1.
- String Color – Cor da aresta.

O grafo G tem como principais funções:

Vector<Vertex*> ShortestPath(Vertex* source, Vertex* dest) – função de G -> V

- Esta função calcula o caminho mais curto entre um vértice fonte, que será o herói e um vértice destino. Sempre que a função encontra em dragão esse vértice deixa de poder fazer parte de um caminho, evitando assim os dragões.

Vector<Vertex*> ShortestPathDragon(Vertex* source) – função G -> V

- Esta função calcula o caminho entre um vértice fonte, que será o herói e o primeiro dragão que encontrar. Ao contrário da versão anterior, esta versão pára de construir caminhos quando enconta um dragão, o qual estará mais perto do vértice fonte.

Funcionamento do Programa

A lógica do programa funciona por turnos.

Primeiro move-se o herói. Dependendo se de este estar ou não armado, o seu objectivo será a espada ou o dragão mais próximo. Se não existirem mais dragões, o seu objectivo será a saída. Visto que a saída pode estar colocada em qualquer nó, o herói pode passar por cima da saída enquanto se dirige para outro objectivo.

De seguida movimentam-se todos os dragões numa direcção aleatória. Se o dragão escolher uma direção na qual não se pode movimentar, não se move. Se se mover para cima do herói desarmado o herói morre e o jogo acaba.

Os dragões não podem passar por cima da espada, da saída ou de outros dragões.

Limites da Aplicação/Restrições

De modo que o programa funcione correctamente o labirinto tem de conter todos os elementos necessários – herói, espada, saída, espaços livres e pelo menos um dragão. O labirinto não tem de ser quadrado, mas todos os espaços devem estar preenchidos no ficheiro de texto.

Existe a possibilidade de um labirinto ser impossível de resolver se um dragão imóvel estiver entre o herói e a espada, o que impede o herói de a alcançar.

Se o caminho para a espada estiver bloqueado não é possível a construção de um caminho. Como tal o herói não se move.

Em relação a confrontos entre o herói e um dragão, um herói é morto por um dragão se este se mover para o nó onde o herói se encontra e este tiver desarmado. Neste caso, o jogo termina.

De modo semelhante, o herói mata um dragão se estiver armado e se mover para um nó onde se encontra um dragão. O jogo termina quando o herói alcançar a saída depois de ter morto todos os dragões.

Resultado Esperado

Estando o programa a funcionar correctamente é de esperar que seja possível visualizar o grafo usando a biblioteca Graph Viewer. Cada elemento terá uma cor diferente – azul para o herói, vermelho para os dragões, verde para a espada e amarelo para a saída - e será possível ver o herói a deslocar-se no labirinto, usando o caminho mais curto entre a sua posição inicial e o seu objectivo, sendo esse objectivo primeiro a espada, depois cada dragão e finalmente a saída.

Principal Algoritmo Aplicado

O principal algoritmo aplicado é o algoritmo de caminho mais curto em grafos não pesados. O mesmo é usado nas funções ShortestPath e ShortestPathDragon, com critérios de paragem diferentes.

```
//inicialização dos nós
∀ v ∈ G(V,E)                    //O(|V|)
    dist(v) ← ∞
    path(v) ← NULL

    dist(source) = 0
    queue Q = Q ∪ source

    while(!Q.empty)             //O(|V|)
    v = Q.front
    Q.pop
    ∀ e ∈ v                     //O(|E|)
        w = dest(e)
        if(symbol(w) != "D") //de modo a evitar dragões
            if ( dist(w) == ∞ )
                dist(w) = dist(w) + 1
                path(w) = v
                Q = Q ∪ w
```

Usos da Aplicação

A aplicação é capaz de calcular o caminho mais curto entre dois vértices num grafo, sendo capaz de evitar que o caminho passe por vértices que possam ser considerados indesejáveis por algum motivo.

Embora no caso específico deste programa não seja necessário atribuir um peso às arestas, visto que cada conta como um passo, o algoritmo é capaz de funcionar com aresta que tenham um peso positivo com poucas alterações, em qualquer tipo de grafo.

Diagrama UML

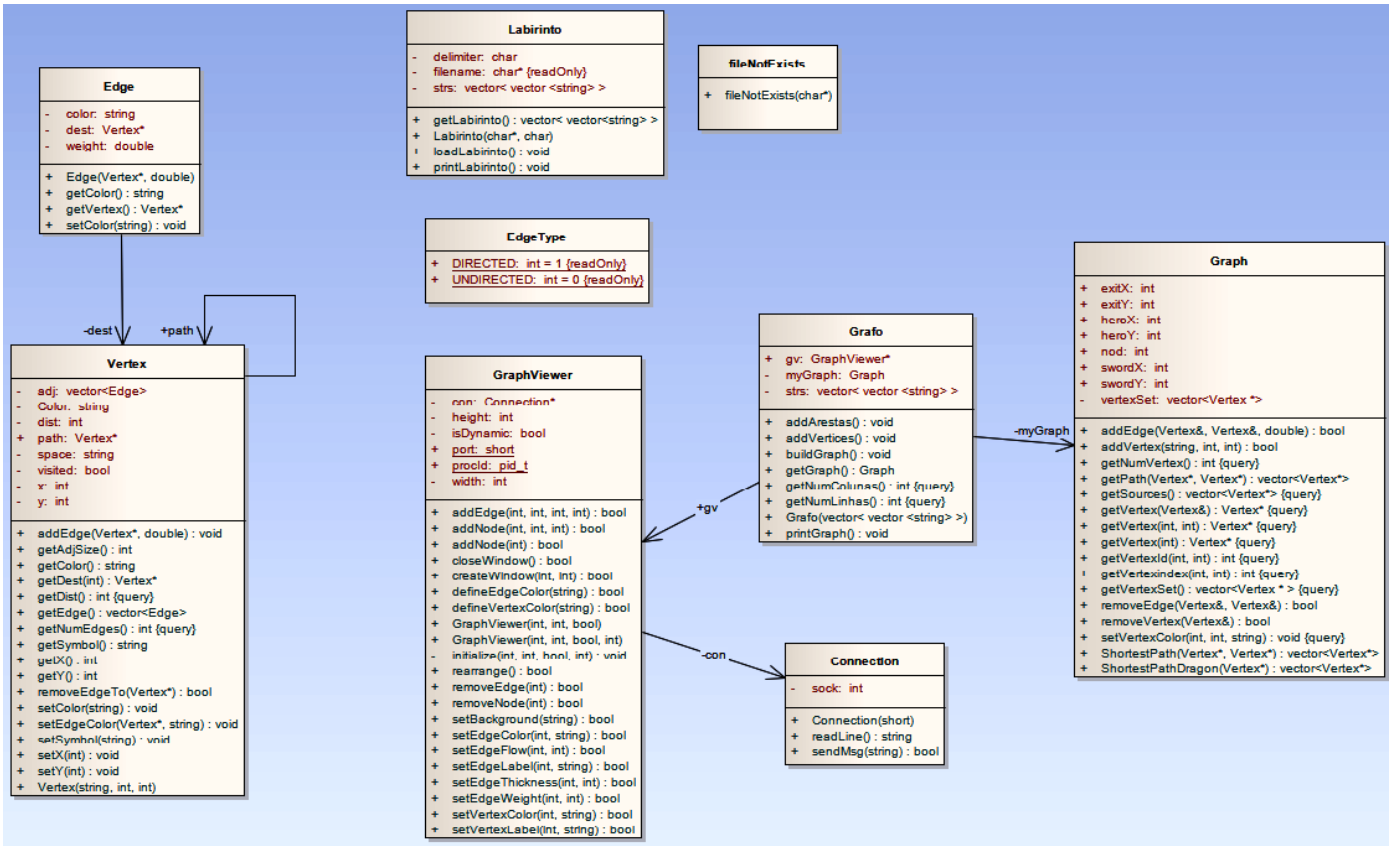

Ilustração 2 – Diagrama UML

Contribuição dos Elementos do Grupo

Leitura de Ficheiro para Matriz – Rui Costa
Elaboração do Programa sem Graph Viewer – João Cardoso
Adaptação do Programa para Graph Viewer – Rui Costa
Elaboração do Relatório – Joel Dinis

Conclusões

É possível concluir que a complexidade do algoritmo é O(|V| + |E|) e que a complexidade do espaço auxiliar, usado para inicialização dos nós é O(|V|).

No pior caso possível é necessário percorrer todos os vértices para encontrar o objectivo do herói.

Algumas melhorias a implementar poderiam ser permitir aos dragões moverem-se por cima de outros tipos de nós, permitir que vários elementos podessem estar num só nó ao mesmo tempo e mudar lógica de encontro com um dragão. O herói entraria em confronto com um dragão se estivesse num nó adjacente em vez de no mesmo nó. Neste caso, seria necessário mudar também o algoritmo de construção de caminhos.