Uma imagem com gráficos de vetor

Descrição gerada automaticamente

ADA TP2

Tiago Fernandes 57677 | António Ferreira 58330

# Explicação:

Para resolver o desafio, decidimos usar o algoritmo de Bellman-Ford, com o objetivo de determinar o caminho do challenge inicial para o final (wizard), calculando os melhores pesos de todos os edges a partir do vértice inicial.

O algoritmo de Bellman-ford é feito de forma semelhante ao lecionado nas aulas teóricas, embora tenha código adicional para o casos em que um nó consiga chegar ao wizard a partir de um caminho.

Foi criado um set canReachFinal (de modo a não existir nós repetidos) que guarda os nós que podem chegar ao nó final. O set tem inicialmente o nó final.

Sempre que o updateLengths é invocado, verifica-se se o secondNode da edge corrente existe no set e, neste caso, o firstNode é adicionado ao canReachFinal, pois se o secondNode chega ao nó final e se o firstNode se encontra numa edge direcionada para o secondNode, então o firstNode também consegue chegar ao nó final.

No final do algoritmo Bellman-Ford, aquando da verificação de ciclos de peso negativo, caso o peso de algum nó tenha mudado e este consiga chegar ao wizard, é lançada a exceção NegativeWeigthCycleException, que devolve ‘Full of energy’.

O algoritmo devolve o peso final do nó correspondente ao wizard. Caso este valor seja menor ou igual a zero, é lançada a exceção NegativeWeigthCycleException. Caso contrário, a resposta é o máximo entre zero e a subtração entre a energia inicial (intialEnergy) e energia consumida durante o caminho (resultado do algoritmo).

# Complexidade temporal

**AwesomeWarriorGame.AwesomeWarriorGame:**

No construtotor a complexidade temporal é O(1), pois faz-se apenas inicialização de variáveis, todas com complexidade O(1).

**AwesomeWarriorGame.handleConnection:**

É feita uma inicialização de uma variável e uma atribuição a uma variável, ambas com complexidade O(1), logo o método tem complexidade **O(1).**

**AwesomeWarriorGame.processFinalLine:**

São feitas 3 inicializações de variáveis, todas com complexidade O(1), logo o método tem complexidade **O(1).**

**AwesomeWarriorGame.updateLengths:**

Como optámos por utilizar uma lista de arcos para representar o grafo, o ciclo terá uma complexidade temporal de **O(|decisions|),** sendo decisions o equivalente ao número de edges.

**AwesomeWarriorGame.bellmanFord:**

É feito o algoritmo de Bellman-Ford.

O ciclo que preenche o vetor length com long.MAX\_VALUE tem complexidade O(|numNodes|), pois o ciclo for é feito numNodes vezes.

O ciclo que invoca o método updateLengths tem complexidade O(|numNodes| \* |decisions|), pois o ciclo corre numNodes vezes e é invocado um método com complexidade temporal O(|decisions|), sendo que decisions é o equivalente ao número de edges.

Como todas as outras operação têm complexidade O(1), a complexidade temporal será O(|numNodes|) + O(|numNodes| \* |decisions|) = **O(|numNodes| \* |decisions|).**