Relatório Trabalho de ADA – Lost

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| * Ano Letivo | * 2020/2021 | * Semestre | * 2 | * Cadeira | * ADA |
| * Alunos | * Gonçalo Martins Lourenço nº55780 | | | | |
| * Joana Soares Faria nº 55754 | | | | |

## Complexidade Temporal

Na análise da complexidade temporal consideramos as seguintes variáveis:

- número de caminhos possíveis (arcos do grafo)

- número de localizações existentes (células da ilhas)

Para computar a solução foi escolhido o algoritmo de Bellman-Ford. Para este algoritmo começamos com a inicialização de um array de tamanho igual ao total de número de posições na ilha (). Este array tem que ser inicializado em todas as posições com , pelo que temos um custo associado a este passo de .

Depois, temos um ciclo executado para todos os vértices do grafo, mas que pode parar antes, pelo que é executado, no máximo, vezes. Dentro deste ciclo temos uma série de passos constantes, como testes de condições e depois todos os arcos a considerar serão iterados. Os arcos a iterar estão divididos por tipos (arcos associados a células de água, a células de erva e a rodas mágicas) e apenas serão iterados os arcos possíveis de serem utilizados pelo jogador em questão. Sendo assim este ciclo é executado, no máximo, vezes. Concluímos então que este passo têm uma complexidade de .

O algoritmo é executado duas vezes, uma vez para cada jogador, uma vez que o grafo que representa os movimentos possíveis de cada jogador difere ligeiramente. Obtemos assim uma complexidade final de , que simplifica para uma complexidade final de .

# Complexidade Espacial

Na análise da complexidade espacial consideramos as seguintes variáveis:

- número de caminhos possíveis (arcos do grafo)

- número de localizações existentes (células da ilhas) (que corresponde a )

– número de linhas da ilha

– número de colunas da ilha

– número de rodas mágicas

Para a complexidade espacial identificamos os seguintes elementos:

* Variável – uma matriz de dimensão , para guardar o tipo de cada célula/posição da ilha. Estas variáveis é apenas utilizada na construção do grafo, para que os arcos do grafo possam ser adicionados corretamente ao conjunto a que pertencem (os arcos encontram-se divididos por tipos), ou não serem adicionados arcos para células correspondentes a obstáculos. Temos assim .
* Seguidamente temos três variáveis: , que totalizam o número de caminhos possíveis na ilha, dividido pelo tipo de caminho que são. Temos assim .
* A variável , também utilizada para a construção do grafo, que guarda a informação da posição das rodas mágicas, dá-nos .
* A variáveis guarda a informação sobre a codificação de cada posição da ilha, usando como chave a posição ) da célula e como valor a codificação correspondente, que varia de . Dados é que necessário uma entrada por posição da ilha temos uma complexidade espacial de .
* Por fim necessitamos, para a computação do algoritmo de um vetor com tamanho igual ao total de número de vértices do grafo, , que acresce uma complexidade temporal de .

Sendo assim temos uma complexidade espacial de , dado que , temos uma complexidade espacial simplificada de . Dado que cada posição da ilha pode ter 4 ligações, uma a cada célula adjacente, mais as ligações provenientes das rodas mágicas, sabemos que o número total de arcos do grafo é superior ao número total de vértices.

# Conclusões

A nossa solução responde ao problema de forma bastante eficiente e foi implementada de uma forma bastante legível na nossa opinião. No entanto, temos consciência de que não escolhemos a implementação mais eficiente e por isso sentimos a obrigação de justificar a nossa decisão.

Escolhemos resolver o problema implementado o algoritmo de Bellman-Ford, por conseguir suportar todas as especificações do problema: arcos pesados e arcos de pesos negativos, com uma complexidade temporal de . Em comparação com o algoritmo de Floyd-Warshall, cuja complexidade temporal seria algo como , é vantajoso. Podemos limitar superiormente o número de arcos como sendo , pois mesmo existindo alguns vértices com 5 arcos (rodas), os da fronteira da ilha têm menos de 4 e compensam o número de arcos a mais das rodas, assim sendo: (nas condições do problema).

Primeiramente, a outra solução (ou soluções) em que pensamos consiste(m) em fazer uma avaliação à *priori* da situação específica daquele problema, e, a partir dessa avaliação identificar a existência de alternativas mais eficientes do que o algoritmo de Bellman-Ford para a procura do caminho.

Assim, com a solução geral solucionada, pensámos que: uma vez que o Jonh não nada, se não existirem rodas mágicas, a procura feita para este caso pode ser uma procura em largura, pois todos os arcos têm o mesmo custo. Podemos ainda verificar, se no caso de existirem rodas mágicas, se estas têm custo 1, o que as tornaria esta solução igualmente válida.

Por outro lado, e agora uma solução um tanto quanto mais abrangente, pelo algoritmo de Dijkstra podemos procurar o caminho em todos os casos que não envolvam pesos negativos, o que nos permite descobrir o caminho da Kate, e, se não existirem pesos negativos nos arcos das rodas, do Jonh.

É de referir ainda que a implementação dos diferentes algoritmos só traria vantagens se as estruturas usadas para o grafo fossem as mais adequadas para eficiência de cada algoritmo. Como nem todos os algoritmos referidos usam a mesma representação do grafo, este teria que ser guardado de diferentes maneiras o que levaria a um aumento da complexidade espacial.

Dito isto, por interpretação do enunciado, compreendemos que seriam poucos os casos em que estas condições se verificariam, e, assim sendo, deduzimos não ser objetivo deste trabalho a implementação de diversos algoritmos, que tornariam sim a complexidade temporal mais baixa, embora apenas em circunstâncias específicas a diferença se mantivesse após simplificação, a custo de muito mais tempo de implementação, testes e cálculos explicativos do nosso raciocínio. Por estes motivos decidimo-nos pela implementação usando o algoritmo Bellman-Ford para evitar maior complexidade espacial e menor legibilidade do código.

Anexo – Código Main

Anexo – Código Class Lost