UNIVERSIDADE DE ÉVORA

3º Trabalho - Mazy Luck

Estrutura de Dados e Algoritmos II

Professora: Vasco Pedro
Realizado por: Filipe Alfaiate (43315), Miguel de Carvalho (43108) **Grupo:** g308

5 de junho de 2021

1 Algoritmo

O nosso algoritmo consiste na utilização de um **grafo orientado cíclico com pesos**, pertencentes aos **números inteiros**. O grafo contém **vértices**, que representam as **salas**, e **arcos**, que representam os **corredores entre as salas**.

Cada **sala** contem o **identificador da sala**, a **distância** a que ela se encontra da sala inicial e a **sala que a precede**.

Cada **corredor** contem duas salas, a **sala de origem** e a **sala de destino**, e um **custo** para o poder atravessar. O **custo é positivo** quando se encontra um **saco** com uma certa quantia de dinheiro e **é negativo** quando se encontra um **crocodilo**, sendo necessário gastar uma certa quantia para baixar a ponte e atravessar o corredor em segurança, essa quantia é descontada em **débito** ou **crédito**.

A utilização deste algoritmo baseia-se no caminho mais curto (apresenta um menor custo) desde da **sala inicial** até à **sala final**, obtendo como resultado a possibilidade do Dirk **perder dinheiro ao percorrer o labirinto**.

2 Análise do Grafo

Tendo o labirinto pronto a ser percorrido, colocamos a distância da **sala inicial** a zero.

Em seguida, percorremos todos os **corredores** o mesmo número de vezes quanto as salas existentes no labirinto. Para cada corredor verificamos se a **sala de origem** tem uma distância conhecida, caso apresente, somamos a distância com o custo do corredor, comparando se o resultado é menor que a distância entre a **sala inicial** e a **sala de destino**, caso se verifique a condição a **sala de destino** fica com o resultado da soma e um precedente que é a sala de origem. Com o término desta pesquisa percorremos todos os corredores uma última vez para verificar se existe um **ciclo** cujo o saldo ao percorrer o mesmo é negativo, ou seja, ganha 3 quantias e perde 6, logo tem um saldo negativo de 3.

Segue-se uma ilustração do problema:

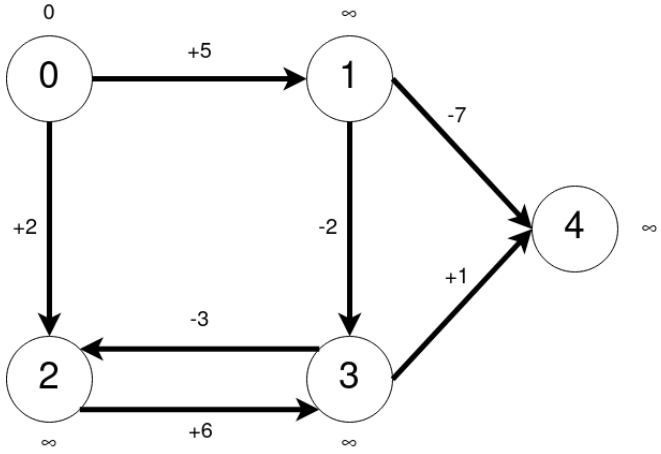

Figura 1: Labirinto inicial

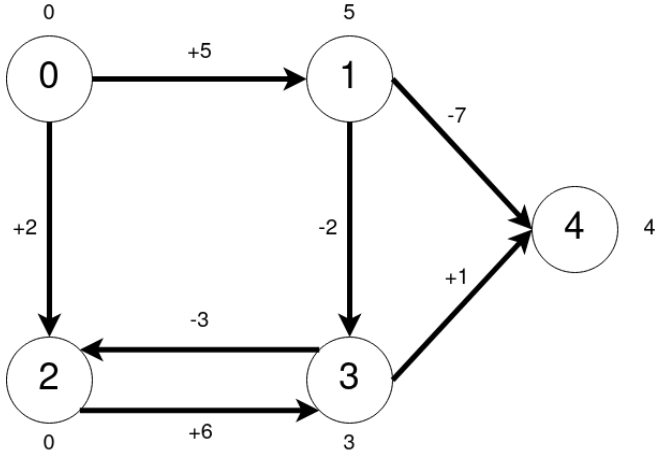

Figura 2: Labirinto com a distância mais curta desde o vértice 0 até aos outros

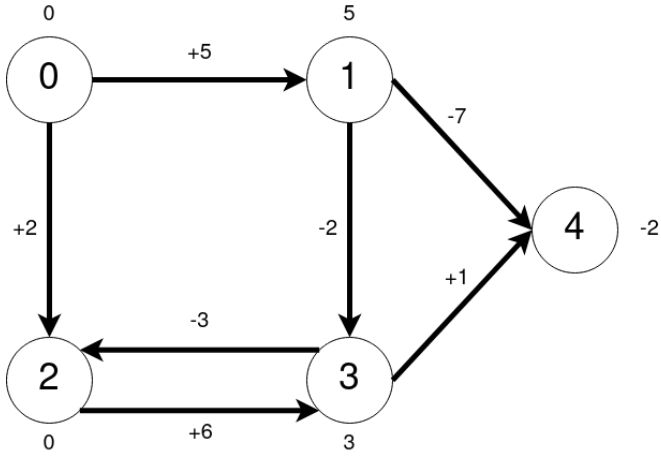

Figura 3: Labirinto com o caminho mais curto para todos os vértices com origem no vértice 0

3 Complexidade

3.1 Temporal

Começámos por proceder à leitura de uma `string` com 2 números, **número de salas**, **número de corredores**, respetivamente, o que origina uma complexidade constante `O(1)`.

$$\mathbf{O(1) = O(1)}$$

De seguida criámos dois vetores para representar todas as salas e todos os corredores, que apresenta um custo constante `O(1)`.

$$\mathbf{O(1) = O(1)}$$

Posteriormente procedemos à leitura do resto das linhas de input com um ciclo com um número de iterações igual ao número de corredores, que apresenta, 3 números e uma letra, **sala de origem**, **sala de destino**, a representação de um número positivo ou negativo (**B** ou **C**) e o **custo**. Tanto a leitura de input, as afetações realizadas e a condição apresentam um custo constante `O(1)`.

Ainda dentro do mesmo ciclo, após a afetação do input as variáveis correspondentes, verificou-se se a **sala de origem e de destino** existem, caso não existam são criadas com um número e uma distância que representa o infinito, colocado-as num vetor de salas. Ou seja, apresenta um custo constante `O(1)`. Para além da verificação da existência de sala é criado também um corredor com o custo, a **sala de origem** e a **sala de destino**, colocando-o no vetor de todos os corredores. Tal como as afetações realizadas acima, esta também tem custo constante `O(1)`.

$$\mathbf{O(1) + O(1) + O(1) = O(1)}$$

Com o término da leitura dos inputs podemos afirmar que tem uma complexidade linear `O(E)`, onde `E` é o número de corredores do labirinto.

$$\mathbf{O(1)\ x\ O(E) = O(E)}$$

Afetámos a distância da **sala inicial**, o que origina um custo constante `O(1)`.

$$\mathbf{O(1) = O(1)}$$

Com tudo preparado para realizar o algoritmo **Bellman-Ford** procedeu-se à sua execução. Nessa execução irão ser realizados dois ciclos, onde o ciclo exterior percorre todas as **salas** e o interior todos os **corredores**. A cada iteração do ciclo interior serão realizadas condições, afetações e operações todas com uma complexidade constante `O(1)`, originando uma complexidade `O(VE)`, onde `V` é o número de **salas** do labirinto e `E` é o número de **corredores** do labirinto.

$$\mathbf{O(1) + O(V)\ x\ O(E) = O(VE)}$$

Por fim, percorremos uma última vez todos os **corredores** verificando se existe algum caminho mais curto, obtendo uma complexidade `O(E)`, onde `E` é o número de **corredores**.

$$\mathbf{O(E) = O(E)}$$

Complexidade final:

$$\mathbf{O(1) + O(1) + O(E) + O(1) + O(VE) + O(E) = O(VE)}$$

3.2 Espacial

Para ler os inputs é necessário criar um array de strings onde o seu tamanho varia consoante o número de palavras lidas durante a inserção do input, ocupando assim $\theta(L)$, onde `L` é o número de elementos lidos entre espaços.

Todas as variáveis do **tipo inteiro** terão uma ocupação em memória constante `O(1)`.

Durante a inicialização dos vetores terão de ser reservados espaços na memória, um com tamanho `V`, que ocupa em memória `O(V)`, onde `V` é o número de **salas** do labirinto, e outro com tamanho `E`, que ocupa em memória `O(E)`, onde `E` é o número de **corredores** do labirinto.

Na execução do ciclo onde lê o resto dos inputs é criado um array de strings onde o seu tamanho varia consoante o número de palavras lidas durante a inserção do input, ocupando assim $\theta(L)$, onde `L` é o número de elementos lidos entre espaços. Como em cada ciclo cria um novo array a complexidade espacial multiplica pelo número de corredores, tendo então um complexidade de `O(EL)`, onde `E` é o número de corredores.

Ainda no mesmo ciclo, são criados **salas** e **corredores** onde as **salas** e os **corredores** terão uma complexidade `O(1)`.

4 Comentários Adicionais

No decorrer do projeto existiram diversas fases de desenvolvimento, uma fase de entendimento, uma fase de pesquisas e dúvidas, uma fase de desenvolvimento do trabalho e uma última fase de debug.

Nesta última fase deparámos-nos com um problema que permitiu evoluir o nosso conhecimento. Quando somado o maior número inteiro de 32 bits com outro número natural é criado um overflow, isto significa que os 32 bits não são suficientes para representar esta soma. Como solução o processador, para continuar a executar corretamente o programa, coloca todos os bits a zero e continua a somar o restante a partir desse ponto, originando um número errado. Para solucionar este problema bastou apenas fazer uma verificação onde é verificado se a distância da respetiva sala é igual ao número inteiro máximo de 32 bits.