Report for Programming Problem 3

Team: 2019214567_2019219068
Student ID: 2019214567 Name Miguel Antunes Carvalho Portugal Ferreira
Student ID: 2019219068 Name Tiago Miguel Gomes Marques Oliveira

1. Algorithm description

Para a estatística 0, após a leitura do input, contamos o número de nós com 0 filhos e o número de nós com 0 pais. Se algum destes for maior que 1 quer dizer que existe mais que um nó final ou mais que um nó inicial, respetivamente, e printamos "**INVALID**", terminando o programa. Se o input passar nesta segurança vamos verificar se existem ciclos através da função ***hasCycle()***. Esta função usa uma *priority queue*, que inicialmente contém o nó inicial. Enquanto esta *queue* não estiver vazia damos *pop* do elemento do topo, colocamos a sua flag ***processed*** a *true* e este é adicionado ao *vector* ***processedOp***, sendo todos os seus filhos adicionados à *queue*. De seguida, o array dos filhos é percorrido e o array de pais de cada um destes filhos é também percorrido incrementando uma variável auxiliar *(j)* por cada pai que já tenha sido processado. Se *j* for igual ao número de pais indicado no input, este filho é adicionado à queue. Quando a queue ficar vazia verifica-se se o tamanho do ***processedOp*** é igual ao tamanho do ***operations*** (*vector* que contém todos os nós) e, caso isto não seja verdade, é retornado -1, indicando que existe um ciclo no input. É printado "**INVALID**".

A estatística 1 é processada durante a execução do ***hasCycle()***. À medida que os nós são adicionados ao *vector* ***processedOp***, o seu tempo de processamento é somado à variável time. Quando a queue ficar vazia a função acaba e o *vector* ***processedOp*** é percorrido e printado o ID de cada um dos nós.

A estatística 2 é efetuada enquanto é feita a deteção de ciclos e a estatística 1. A diferença desta estatística para a estatística 1 é que em vez de ser somado o tempo de cada nó a uma variável *time* é verificado qual dos pais desse nó tem um valor de tempo total maior (***totalT***) e é somado o tempo do próprio nó com o tempo do maior pai à variável *totalT*. Isso permite-nos dar *track* do necessário para chegar a cada nó. No fim desta função é dado *print* do tempo total do nó final.

A estatística 3 é processada através da função ***statistic3()***, que percorre o *vector* ***processedOp*** e, para cada um dos nós chama a função ***isBottleneck()*** e ***isBottleneckAux()***. A primeira percorre todos os filhos do nó enviado e os filhos dos filhos, recursivamente, até chegar ao nó final, somando 1 a um contador e passando a posição no array ***aux*** correspondente ao seu ID a true (o array *aux* indica se o nó já foi visitado). No final da execução desta função é retornado o contador. A segunda função tem um comportamento semelhante, diferindo na chamada recursiva, que é feita para os pais em vez de para os filhos do nó. Se a soma do *return* destas duas funções com 1 (de forma a contar com o nó inicial) for igual ao número total de nós, quer dizer que estamos perante um ***bottleneck***, visto que todos os nós do grafo são acessíveis através deste, e o ID deste nó é printado.

2. Data structures

A classe ***Operation*** contém o inteiro **T**, que representa o tempo necessário para processar esta operação; o inteiro **D**, representa o número de nós dos quais esta operação depende (pais); booleano **processed**, indica se esta operação já foi processada, inicializado a false; inteiro **totalT**, usado para auxiliar na contagem do tempo para a estatística 2; *vector* de inteiros **sons**, contém os IDs das operações que dependem desta operação (filhos); *vector* de inteiros **parents**, contém os IDs dos nós dos quais esta operação depende.

O *unordered_map* ***operations*** contém na sua chave o ID da operação e no valor a ***Operation*** correspondente.

Os *pairs* ***initialOperation*** e ***lastOperation*** indicam o ID e a ***Operation*** da primeira e última operação, respetivamente.

O *vector* de inteiros ***processedOp*** guarda os IDs dos nós que já foram processados e o *vector* de booleanos ***aux*** é usado na estatística 3 para indicar quais dos nós foram visitados na recursividade. O inteiro ***nOp*** indica o número total de operações.

3. Correctness

O algoritmo funciona corretamente para todos os casos do enunciado. No entanto, para a estatística 3 ele é bastante lento e por essa razão apenas recebemos 166 pontos. Para conseguir os restantes pontos poderíamos ter implementado um algoritmo de *reachability*, como o de *Floyd-Warshall*, que deteta o caminho mais curto entre dois vértices.

4. Algorithm Analysis

A complexidade espacial vai ser O(n) porque apenas é preciso guardar os nós num vetor de dimensão n.

Os algoritmos da estatística 0, 1 e 2 têm complexidade temporal O(n), visto que cada nó é percorrido apenas uma vez. Garantimos que um nó nunca é visitado duas vezes através da presença da flag ***processed*** e do *vector* ***processedOp***.

A estatística 3 tem uma complexidade de $O(n^2)$, visto que cada nó é percorrido uma vez por um ciclo e o *vector* dos nós é percorrido n vezes durante as chamadas recursivas.

5. References

GeeksforGeeks. 2021. “Longest Path in a Directed Acyclic Graph.” GeeksforGeeks. https://www.geeksforgeeks.org/find-longest-path-directed-acyclic-graph/.

GeeksforGeeks. 2022. “Dijsktra's algorithm.” GeeksforGeeks. https://www.geeksforgeeks.org/dijkstras-shortest-path-algorithm-greedy-algo-7/?ref=lbp.

GeeksforGeeks. 2022. “Detect Cycle in a Directed Graph.” GeeksforGeeks. https://www.geeksforgeeks.org/detect-cycle-in-a-graph/.