
# Estratégias Algorítmicas

---

## Problema de Programação 3

## Analyzing a Data Pipeline

**Equipa:**

N° Estudante: 2019218981 Nome: Marco da Cruz Pais

N° Estudante: 2019215412 Nome: Simão Carvalho Monteiro

## 1. Descrição do Algoritmo

**Estatística 0:**

Durante a leitura do input, é contado o número de nós iniciais verificando as dependências existentes. Se as dependências forem zero é considerado um nó inicial. Caso exista apenas um nó inicial, é chamada a função *isValid()*, que verifica o número de nós terminais e se o grafo é conectado e não tem ciclos. Caso contrário, é imprimido "INVALID".

Para verificar se existem ciclos é usado um vetor booleano, que durante um depth first search verifica se numa chamada recursiva se passa duas vezes pelo mesmo nó. Para verificar se o grafo é conectado são contados, no depth first search, o número de nós diferentes percorrido. Caso esse valor seja igual ao valor total, guardado inicialmente, o grafo é conectado.

**Estatística 1:**

Nesta estatística, é usado o breadth first search e uma fila de prioridade do mais pequeno para o maior. Primeiramente, é iniciado o vetor indegree com as respectivas profundidades de cada nó. De seguida, é usado um ciclo para percorrer o grafo e adicionar nós à fila de prioridade, sempre que o indegree desse fosse zero, até a fila de prioridade estar vazia. No final, imprime-se o caminho percorrido.

**Estatística 2:**

Sendo possível efetuar um número infinito de operações em simultâneo, o custo mínimo possível nesta estatística seria o caminho mais custoso, pois nesse caminho estariam os nós pelos quais todos os outros nós teriam de esperar depois de percorridos.

Para resolver este problema é chamada a função isValid() para verificar se o grafo é válido. Sendo que o vetor *order* guarda a ordem em que os nós são acedidos, este é usado para percorrer os nós nessa ordem e ser calculado o caminho mais custoso. Em cada nó, é verificado se a distância do pai em questão mais o custo do filho é maior que a distância atual do filho. Caso seja maior o valor da distância do filho, passa a ser o custo dele mesmo mais o do pai. Este processo ajuda a obter sempre o máximo custo de um caminho, visto que existe sempre um nó terminal, ou seja, as distâncias máximas irão ser acumuladas até ao índice do nó terminal ficando, então, no final o nó terminal com o custo total do caminho mais custoso.

**Estatística 3:**

A estatística 3 foi relativamente simples de resolver. Primeiramente chama-se a função isValid() para verificar se o grafo é válido. De seguida, são percorridos todos os nós, na ordem guardada no vetor order, e, em cada nó, são percorridos o caminho normal até ao nó terminal e o caminho reverso até ao nó inicial, através de depth first search, contando o número de nós diferentes que são visitados. Caso o número de nós percorridos nos dois caminhos seja igual ao número total de nós é considerado um nó bottleneck, caso contrário não o é.

## 2. Estruturas de Dados

Foi usada uma classe, *Pipeline*, e 5 vetores muito importantes .

**Classes:**

- *Pipeline* – Contêm toda a informação sobre uma pipeline assim como todas as funções necessárias para a resolução das quatro estatísticas como as funções *isValid()*, *dfs()* e *bfs()*.

**Vetores:**

- *operations* – Contém as conexões adjacentes seguintes de cada nó;
- *reverseOperations* – Contém as conexões adjacentes anteriores de cada nó;
- *visited* – Utilizado para guardar os nós que já foram visitados na pesquisa em específico;
- *path* – Utilizado para a deteção de ciclos;
- *order* – Utilizado para guardar a ordem de processamento dos nós da pipeline.

## 3. Assertividade

A nossa solução está correta visto que é testado sempre se um grafo é válido, ou seja, se o grafo é conectado, acíclico e tem apenas um nó inicial e terminal. Dito isto e visto que não encontrámos nenhum caso de teste onde o algoritmo falhasse, achamos que o nosso algoritmo está correto.

## 4. Análise do Algoritmo

Sendo **V** os vértices e **E** as edges de um grafo:

- A Complexidade Espacial é **O(V)** visto que o espaço ocupado pelo algoritmo aumenta linearmente com o aumento de nós adicionados ao grafo.
- A Complexidade Temporal é **O(V + E)** visto que para todas as estatísticas é preciso percorrer todos os vértices e as suas respetivas ligações.

## 5. Referências

As seguintes referências foram utilizadas com intuito de perceber alguns detalhes e funcionalidades sobre estruturas e funções do C++.


GeeksforGeeks

C++ Reference

Slides Teóricos e Teórico-Práticos de Estratégias Algorítmicas