Relatório 2º projeto ASA 2021/2022

Grupo: al012 | Alunos: Valentim Santos (99343), Tiago Santos (99333)

Descrição da solução

A solução encontrada consiste em percorrer (visitar) todos os ancestrais dos dois vértices recebidos como input, ou seja, visitar o vértice em si e de seguida os seus "pais". São feitas duas destas "visitas", originando assim um subgrafo contendo apenas os ancestrais comuns a ambos os vértices, de onde então retiramos o/s vértice/s mais "leve/s" (solução).

Análise teórica

LEITURA E PROCESSAMENTO DE INPUT:

A leitura de dados de entrada é feita em tempo linear sendo a sua complexidade **$\Theta(n)$**, onde ***n*** representa o tamanho do input. A complexidade do processamento do input dependerá do preenchimento da tabela (explicado em baixo).

PREENCHIMENTO DA TABELA:

A tabela utilizada tem tamanho **4 × número de vértices do grafo** e é preenchida da esquerda para a direita e de cima para baixo, o seu preenchimento tem complexidade **O(4n_vértices)**, e qualquer acesso à mesma é feito em tempo linear, complexidade **O(1)**, e por isso ignorado para o cálculo da complexidade final. Em baixo está representada uma tabela de exemplo, como a que é usada.

	Pai1	Pai2	Nº de pais	Cor
1	-1	-1	0	WHITE
2	1	3	2	WHITE
...	...	...	...	...
n_vértices	-1	-1	0	WHITE

APLICAÇÃO DO ALGORITMO:

O algoritmo utilizado tem três passos:

- Verificar se existem ciclos no grafo. Entramos no grafo por cada um dos seus vértices e percorremo-lo, colorindo cada vértice sempre que o visitamos. Se encontrarmos algum vértice já colorido sabemos então que existe um ciclo. Este passo tem então complexidade **O(E)**, onde ***E*** representa o número de arcos.

```
checkCycle(graph, node)
    visit node;
    for(parent in node.parents)
        if(parent not visited)
            if(checkCycle(graph, parent))
                return true;
        if(parent visited)
            return true;
    return false;
```

- Percorrer o grafo duas vezes, a começar em cada um dos vértices, visitando cada nó até ser encontrado um que não tenha pais (no fundo um algoritmo **DFS** de filhos para pais). A complexidade será então **O(V+E)**, onde ***V*** representa o número de vértices.

$$DFS(node) = \begin{cases} visit(node), & \text{if node has no parents.} \\ visit(node), DFS(parent) & \text{for each of the node's parents.} \end{cases}$$

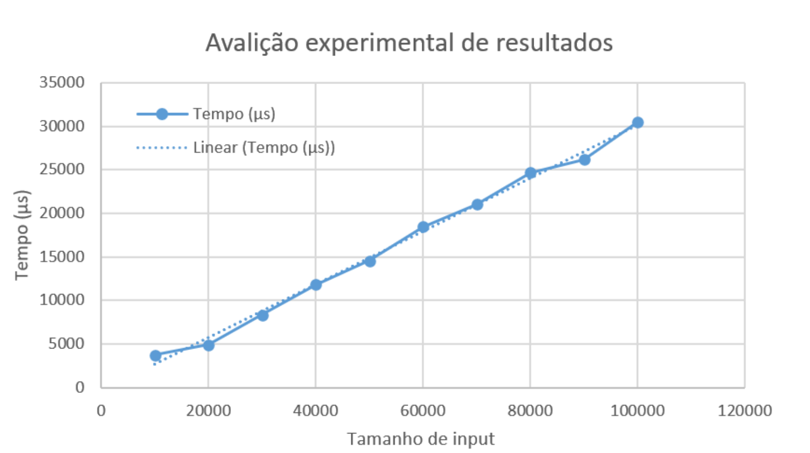
  A função **visit(node)** limita-se a "colorir" (visitar) um vértice, sendo a sua complexidade linear **O (1),** e por isso, ignorada para a complexidade final.

  - Do passo anterior temos agora um subgrafo contendo apenas os ancestrais comuns a ambos os vértices, para descobrirmos o ancestral comum mais próximo, percorremos estes mesmos vértices comuns e "colorimos" cada um dos seus pais a **BLACK**, no fim, os vértices cuja cor não for **BLACK** são aqueles que procuramos. Descobrir quais os vértices em comum tem complexidade **O(V)**, determinar a solução terá complexidade **O(2V_comuns).**

```
for(node in common_nodes)
    for(parent in node.parents)
        parent.color = BLACK;
```

```
for(node in common_nodes)
    if(node.color != BLACK)
        node belongs to solution;
```

Assim, a complexidade final será: **O(E)** + **O(V+E) + O(V)** + **O(2V_comuns) = O(V+E)**.



Analisando o gráfico podemos então verificar que este assume uma curva linear, e então, confirmamos que a complexidade o algoritmo é linear, em **O(V+E)**, assumindo que, em média, **V + E = tamanho de input**.