Relatório 2º projecto ASA 2021/2022

Grupo: al047

Aluno(s): Miguel Eleutério (99287) e Raquel Cardoso (99314)

Descrição do Problema e da Solução

Para resolver o problema proposto, começámos por guardar apenas o grafo transposto ao original, num vetor de vetores. Deste modo, os adjacentes de um vértice seriam os seus pais, caso existissem. Para complementar a informação dos dados de leitura, utilizámos um vetor de inteiros, colors, onde em cada índice correspondente a um vértice do graph, guardávamos a sua "cor" consoante o seu estado. Na nossa resolução utilizámos 6 cores: branco, 0, para quando um vértice não foi ainda visitado; cinzento, 1, para quando um vértice é visitado; preto, 2, para quando um vértice está bloqueado; verde, 3, para quando um vértice é uma solução; vermelho, 4, para quando um vértice é inválido; azul, 5, para quando um vértice fica novamente bloqueado. Inicialmente, todos os vértices são brancos. O algoritmo é constituído por quatro partes:

A primeira, onde após a leitura de dados, verificamos se o grafo é acíclico e o percorremos uma vez. Para tal, utilizámos uma DFS, que ao visitar um vértice inicialmente, o pinta de cinzento e confirma a existência se o visitar novamente. Se a DFS não encontrar nenhum ciclo, prosseguimos e todos os vértices estarão pretos.

A segunda, onde limpamos os nossos vértices para aplicarmos novamente uma DFS com o intuito de encontrar todas os ascendentes do vértice 1 dado nos dados de entrada. Estes ficaram coloridos a preto no final desta interação.

A terceira, onde caso o vértice 2 seja um dos ascendentes do vértice 1, marcamo-lo como resposta e avançamos para a quarta parte, ou, em em caso contrário, o colorimos de cinzento e utilizamos novamente uma DFS para procurar os ancestrais comuns mais próximos. Para tal, temos uma função auxiliar get_ancestors_number que nos possibilita invalidar um vértice e todos os seus ancestrais, caso este seja uma solução ou um vértice inválido.

A quarta, onde apresentamos os resultados. Percorremos o nosso vetor colors à procura das soluções para as imprimir. Se não encontrarmos qualquer solução, imprimimos "-".

Análise Teórica

Assumindo a função dfs_visit como muito semelhante à dada nas aulas:

```
main() {
        readInput()
        if (checkCycle() or moreThan2Parents()) print(0);
        find_closest_common_ancestors();
}
find_closest_common_ancestors() {
```

Relatório 2º projecto ASA 2021/2022

Grupo: al047

Aluno(s): Miguel Eleutério (99287) e Raquel Cardoso (99314)

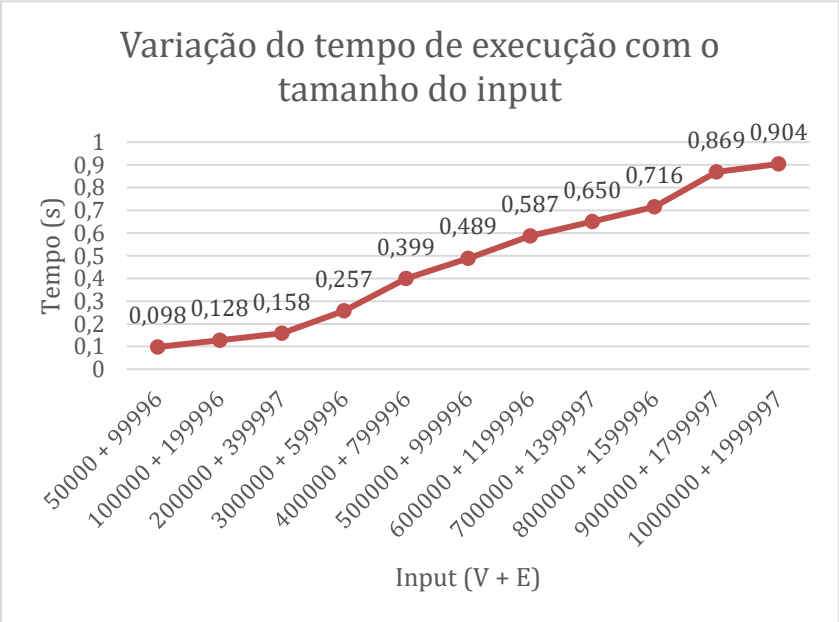
```
    let v1 be the first given node and v2 the second;

    let answers be a new int = 0;
    dfs_visit(v1);
    if (v2.visit == false) {
        v2.visit = true;
        dfs_visit(v2);
    }
    else v2.solution = true;
    for (node in graph) {
        if (node.solution = true) print(node);
        answers += 1;
    }
    if (answers == 0) print("-");
}
```

- Leitura dos dados de entrada: simples leitura do input, com um ciclo a depender linearmente do número de arcos, logo Θ(E).
- Processamento da instância para verificar se o grafo dado é cíclico, dependendo linearmente do número de arcos, logo Θ(E).
- Aplicação do algoritmo de forma a obter os ancestrais comuns. O(V + E).
- Obtenção das respostas e apresentação dos dados. Θ(V).

Complexidade global da solução: O(V + E).

Avaliação Experimental dos Resultados

Pelo gráfico, conseguimos observar um crescimento mais ou menos linear com o tamanho do input, podendo eventuais diferenças dever-se a por exemplo variações no input que facilitam/dificultam o problema, estando assim de acordo com a previsão feita na análise teórica.



Relatório 2º projecto ASA 2021/2022

Grupo: al047

Aluno(s): Miguel Eleutério (99287) e Raquel Cardoso (99314)

Bibliografia

https://fenix.tecnico.ulisboa.pt/disciplinas/ASA/2021-2022/1-semestre/material-de-apoio