Relatório 2º projecto ASA 2021/2022

Grupo: al003

Aluno(s): Luís Marques (99265) e Renato Martins (102314)

Descrição do Problema e da Solução

O problema consiste, a partir de uma árvore genealógica, (um grafo dirigido em que cada nó representa uma pessoa e os vizinhos diretos correspondem aos seus filhos), na procura dos ancestrais comuns de dois nós escritos por input. Para os determinar, é lido o grafo e o seu respetivo transposto, sendo que cada pai só pode apontar para dois filhos. Isto acontece, pois no grafo normal cada filho só pode ter no máximo dois pais. São feitas duas *dfs_visit,* uma para cada nó que escrevemos no input e no percurso usa-se em cada uma das *dfs_visit* um vetor de estados, cujo tamanho corresponde ao número de nós, (1 se tiver sido percorrido e 0 caso não tenha). No fim das duas *dfs_visit* comparamos esses vetores e caso haja índices com o estado 1, colocamos noutro vetor de registo de estados finais o valor 2. São também colocados os elementos em que tal acontece num novo vetor. Os restantes que não são comuns, ou seja, que o seu estado não é 1 em ambos os vetores, fazem com que seja colocado no vetor estado final o estado 1(o que quer dizer que o elemento foi visitado uma única vez numa das duas *dfs_visit*). Seguidamente, percorremos o vetor com os ancestrais comuns aos dois nós e verificamos em cada filho deles no grafo normal se um deles também é um ancestral comum. Caso um deles seja um ancestral comum, não se coloca no vetor com o resultado final. Caso contrário, colocam-se os elementos no vetor final, sendo que depois são devolvidos os elementos que estão neste vetor.

Análise Teórica

1- Leitura dos nós, do número de nós do grafo e o número de arcos.
2- Leitura dos arcos, colocando-os num vetor, cujo primeiro elemento é o pai e o segundo é o filho. É colocado o vetor num vetor de vetor de inteiros (grafo normal) e o seu inverso também é colocado noutro vetor de vetor de inteiros (grafo transposto).
3- É verificada a existência de ciclos no grafo transposto utilizando um vetor de inteiros (trace) que guarda a cor do nó da DFS (preto (2), cinzento (1) e branco (0)). É inicializado tudo a branco. É executada a *dfs_visit* a partir do primeiro elemento do grafo. No início da execução da dfs_visit, neste elemento, ele é colocado a cinzento. Caso um dos filhos desse nó seja cinzento, significa que existe um ciclo e a função retorna true, o que devolve 0 na função principal. Caso este seja branco, continua a procurar até encontrar um nó cujos filhos sejam cinzentos. Caso não haja filhos cinzentos, não existe nenhum ciclo.
4- É feita a dfs_visit para cada nó dado como input inicial, sendo usado um vetor acumulador de estados (1 visitado e 0 não visitado) em cada dfs_visit. É comparado o resultado dos vetores como descrito na Descrição do Problema e Solução. A partir do vetor resultante dessa comparação, cria-se um vetor onde se coloca os elementos cujo seu resultado como índice no vetor resultante é 2, ou seja, esse elemento é percorrido em ambas as *dfs*. Caso não haja comuns, a função principal devolve "-\n". É percorrido esse

Relatório 2º projecto ASA 2021/2022

Grupo: al003

Aluno(s): Luís Marques (99265) e Renato Martins (102314)

---

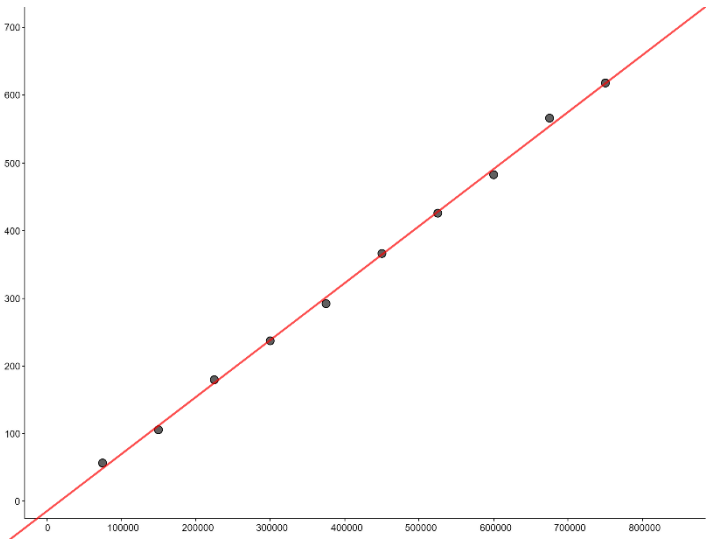
vetor, sendo que é verificado para cada elemento se um dos filhos também lhe pertence. Caso os filhos de um determinado elemento não pertençam ao vetor, ou seja, caso os filhos como índice do vetor de estados finais devolvam um valor inferior a 2, então o elemento é devolvido.

- Leitura dos dados de entrada: leitura do input na função *readVector,* guardando os valores inseridos num vetor. Logo, a complexidade é O(1).
- Aplicação da função *readgraph* que lê os grafos. Como é utilizado um ciclo *for,* a complexidade desta função é O(N).
- Aplicação da função *verify_rest_for_cycles* que verifica a existência de ciclos no grafo, com complexidade O(V+E).
- Aplicação das funções *dfs_visit, dfs_visit1* e *dfs_visit2,* cuja complexidade também é O(V+E).
- Aplicação das funções *getCommons* de complexidade O(V) e *closestCommons* de complexidade O(E).
- Aplicação da função *generationTree,* onde são chamadas diversas funções para determinar o output pretendido. Sendo assim, a sua complexidade será O(V+E).

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

Avaliação Experimental dos Resultados

O exercício foi executado com inputs aleatórios de tamanho de 25 000 a 250 000 vértices. O tamanho dos inputs e o respetivo tempo de execução está representado no gráfico a seguir:



Os valores no eixo dos *yy* dizem respeito ao tempo de execução em ms e os valores no eixo dos *xx* referem-se ao número de vértices mais o número de arcos do grafo do input. Dado que foi obtida uma reta crescente, foi comprovada experimentalmente a complexidade de O(V+E).