Relatório 2º projecto ASA 2021/2022

Grupo: al088

Aluno(s): Francisco Lopes (99220) e Nuno Martins (99292)

Descrição do Problema e da Solução

O problema consiste em achar o ancestral comum mais próximo (LCA) de dois nós de uma árvore geneológica representada por um grafo dirigido. Recebemos de input os dois nós (que correspondem a inteiros), o número de vértices e de arestas do grafo e os pares pai-filho que correspondem às relações de parentesco da árvore geneológica. Durante a etapa de leitura de input, verificamos se a árvore geneológica é válida, isto é, se o máximo de progenitores é 2, se nenhum nó é pai/filho de si próprio e se nenhum nó é pai e filho simultaneamente de outro nó (que criaria um ciclo). No final da etapa de leitura de input, é originado um mapa em que a key é o inteiro de cada nó e o value é uma lista de inteiros relativas aos pais desse mesmo nó, mapa esse que é dado como parâmetro, assim como o número total de nós e os 2 nós que queremos descobrir o LCA à função 'solve'.

Dentro da função 'solve' irá acontecer duas BFS (breadth first search) com origem em cada um dos dois nós aos quais queremos obter o LCA. São inicializados 2 vetores que correspondem às posições visitadas por ambas as BFS e uma queue que irá ser usada para realizar as BFS. A BFS irá ser realizada com tuplos (nó,origem), em que a origem corresponde a um dos 2 nós originais aos quais se quer encontrar o LCA. No caso de o próximo tuplo a analisar que saia da queue tenha origem da BFS com origem no nó1, verificamos se por acaso esse nó já foi encontrado pela BFS com origem no nó2, se for então esse nó é candidato a ser LCA e colocamos ele no unordered_map 'possible'. Se o simétrico acontecer, isto é, se o tuplo a analisar tiver origem na BFS com origem no nó 2 e já tiver sido encontrado pela BFS de origem no nó1, então também é candidato. Após este passo é colocado o nó encontrado no vetor que corresponde aos nós encontrados pela BFS correspondente e iteramos pelos pais do nó e se o pai ainda não tiver sido encontrado pela BFS correspondente à origem do nó, então colocarmos esse nó no final da queue para ser posteriormente processado.

No final quando a queue estiver vazia, significa que o unordered_map contem todos os nós possiveis de serem LCA, porém, para realmente ser LCA, não podem ter filhos que também estejam incluidos nesse mesmo unordered_map. Para isso, criamos uma cópia desse unordered_map para outro unordered_map 'res' (cópia essa feita durante as BFS anteriores) e iteramos pelos pais de cada nó no 'possible' e se o pai estiver no possible então temos de retirar esse pai do 'res', pois nenhum pai LCA pode ter um filho que seja LCA.

Após esta etapa, no unordered_map 'res' iremos ter apenas os LCA filhos que não têm mais LCA filhos, ou seja, todos os LCA corretos, pelo que é só necessário iterar por eles e imprimi-los para o stdout para o utilizador ver. No caso de não haver nós no 'res', então não existem LCA entre os 2 nós originalmente obtidos, pelo que apenas imprimimos '-'.

Análise Teórica

- Leitura dos dados de entrada: simples leitura do input, com um ciclo a depender linearmente do tamanho do input, logo O(E), em que E=numero de arestas entre vértices no grafo. Verificação se a aresta entre vértices já foi observada é realizada em O(logE) pois a função 'find' em set é realizada em O(logE), pelo que complexidade total da leitura do input é O(ElogE).

- Uma vez que após a verificação da interceção das 2 BFS para obter os nós candidatos a ser LCA, usamos 'continue', isto é ele pára de verificar posteriores pais desse nó, e uma vez que nunca visitamos nós repetidos em cada BFS, no total apenas iremos visualizar cada nó da árvore uma única vez, pelo que complexidade temporal é O(V), V=número de nós/vértices no grafo.

- Verificação de quais os LCA dentro do unordered_map de candidatos 'possible' é feita em O(n), incrementando os counters dos pais e só escolhendo os nós com counter igual a 0, isto é que não têm filhos LCA.

- Complexidade espacial é O(V+E) uma vez que são alocados na leitura do input um mapa 'parents' que tem tamanho do número de arestas, isto é O(E) e na função 'solve' são guardados vetores de tamanho n+1 em que n=V (número de vertices no grafo), logo O(E).

Complexidade Temporal final do algoritmo = O(ElogE + V)

Complexidade Espacial final do algoritmo = O(V+E)

Avaliação Experimental dos Resultados: foram testados em ambos os programas 12 inputs de valor entre 1000 e 2048000 vértices, com probabilidade de criar aresta entre 2 vértices de 55%, e no final verificámos que de facto o gráfico está em concordância com a análise teórica relativa à complexidade temporal prevista.