

Universidade de Lisboa - Instituto Superior Técnico  
Licenciatura em Engenharia Informática e de Computadores  
Análise e Síntese de Algoritmos

## 1º Projeto

Nuno Amaro, 81824

Afonso Tinoco, 81861

### Introdução

Com este projeto pretendemos expor um algoritmo em tempo linear para o problema proposto, explicar a sua implementação e fazer uma análise teórica e experimental da complexidade temporal e espacial deste.

### Descrição do problema

O problema descrito pedia para, dentro de uma rede social onde a informação pode ser partilhada entre todas as pessoas da rede, encontrar pessoas que são fundamentais para a transmissão de informação. Isto é, pessoas que ao serem removidas da rede social tornam impossível a transmissão de informação entre todos os membros da rede.

Ora este problema pode ser descrito como um problema num grafo não direcionado, em que:

- cada pessoa é um nó da rede
- uma ligação entre pessoas corresponde a uma aresta
- o grafo é conexo. ("existe sempre uma forma de partilha de informação entre qualquer par de pessoas")

Assim, o problema descrito é equivalente ao de encontrar num grafo conexo  $G(V, E)$  (com  $N = \#V$  e  $L = \#E$ ) nós que se forem removidos tornam o grafo desconexo, ou seja, de encontrar nós de corte (articulation points).

### Algoritmo utilizado

Tendo solução do problema sido estudada nas aulas, foi utilizado o algoritmo estudado - o algoritmo de Tarjan para encontrar nós de corte - descrito como *biconnected components algorithm* em [2]. A descrição do algoritmo será baseada na noção de tempo de descoberta, mas não será utilizada a pilha auxiliar, uma vez que não é necessário encontrar componentes fortemente conexos.

## Estruturas utilizadas:

- **G[u][i]** - O grafo  $G$  foi representado como uma lista de adjacências (foi utilizado um `std::vector` (array dinâmica) em vez de `std::list` (lista duplamente ligada), pelo facto de a implementação do `std::vector` ser bastante mais eficiente que a de `std::list` [1]).
- **disco[u]** - tempo de descoberta do nó  $u$
- **low[u]** - tempo do mínimo valor de descoberta alcançável sem usar a aresta que retorna ao pai na árvore DFS a partir do nó  $u$
- **parent[u]** - nó pai do nó  $u$  na árvore DFS
- **AP[u]** - o nó  $u$  é Articulation Point?

Todas estas estruturas foram implementadas com o `std::vector` de C++ que tem tempo de inserção no fim  $O(1+)$  e de acesso  $O(1)$ . Com tempo de inserção  $O(1+)$  (constante amortizado), queremos dizer que para inserir  $N$  elementos a complexidade de pior caso é  $O(N)$ , apesar da complexidade de pior caso de inserção de 1 elemento não ser  $O(1)$  (de facto é também  $O(N)$ ). [1]

## Explicação do algoritmo

Para o caso de um grafo conexo não-direcionado com pelo menos 2 nós, é executada uma pesquisa em profundidade ao longo dos nós, a partir do nó 1. Para cada nó visitado na DFS é guardada informação relativa ao tempo de descoberta (vetor `disco`) e ao valor do tempo de descoberta do nó com menor tempo de descoberta acessível a partir do nó atual utilizando apenas arestas ainda não visitadas (vetor `low`). Se após uma chamada recursiva retornar, o valor de `low` do filho visitado for superior ao do pai então o pai é um nó de corte.

Após todas as chamadas recursivas há um caso degenerado a considerar, que é o caso da raiz da árvore DFS. Esta apenas será um nó de corte se o seu número de filhos na árvore DFS for superior a 1.

## Prova de correção do algoritmo

A demonstração do algoritmo é simples. Para cada nó  $u$  que não seja a raiz da árvore, e  $v$  qualquer nó tal que  $\text{parent}[v] = u$ ,  $\text{low}[v]$  representa o mínimo valor de disco atingível a partir de  $v$  utilizando apenas arestas ainda não visitadas quando  $v$  começa a ser visitado. Ora se no momento em que a chamada de DFS com parametro  $v$  vai retornar,  $\text{low}[v] \geq \text{disco}[u]$ , isto implica que não existe nenhum caminho entre  $v$  e os nós visitados antes de se visitar  $v$  que não utilize  $u$ , como tal  $u$  é nó de articulação. De igual modo, se para todos os  $v$  a condição se cumpre, então para todo o  $v$  filho de  $u$  na árvore DFS existe um caminho para um nó antes de  $u$  na árvore DFS que não utiliza  $u$ , e vice-versa e como tal  $u$  não é nó de corte uma vez que a sua remoção não afeta a conectividade do grafo. Por fim, falta-nos analisar o caso em que  $u$  é a raiz da árvore. Neste caso, se quando

o primeiro filho de  $u - v$  - retornar da chamada DFS os nós ainda não tiver sido todos visitados, então é porque não existe nenhum caminho de  $v$  para qualquer nó que ainda não tenha sido visitado que não passe por  $u$ , logo  $u$  é nó de corte. No caso contrário, se quando a visita a  $v$  retornar os nós já tiverem sido todos visitados, então existe um caminho de todos os nós (excepto  $u$ ) para todos os nós (excepto  $u$ ) que não passa por  $u$ . Logo  $u$  não é nó de corte. Assim a condição equivalente para a raiz ser nó de corte é o número de filhos na árvore DFS ser superior a 1, e temos a correção do algoritmo demonstrada.

## Análise assintótica temporal teórica do algoritmo

O algoritmo visita cada nó exatamente uma vez, sendo a função de visita chamada exatamente  $N$  vezes. A função de visita tem complexidade local (sem incluir as chamadas recursivas) de  $\#(G[u])$ . Assim, a complexidade de todas as chamadas pode ser descrita como  $\sum_{i=1}^N \#G[i] = L$ . Assim, a complexidade do algoritmo é  $O(\max(N, L)) = O(N + L)$

## Análise assintótica temporal experimental do algoritmo

Para se realizar a análise experimental temporal do algoritmo, foi utilizada a aproximação que cada instrução de C corresponde em média a um número fixo de instruções de CPU. Assim, em primeiro lugar procedeu-se à geração de  $K$  diferentes casos de teste aleatórios (para cada caso foi escolhido aleatoriamente um  $N$  e um  $L \geq N - 1$  e depois gerado um grafo conexo aleatório para esses parâmetros). De seguida, para cada caso de teste foi utilizada a ferramenta perf para contar o número de instruções de CPU utilizadas em modo utilizador durante a execução do program -  $I$ . Por fim, fez-se o plot do gráfico de  $I$  em função de  $N + L$ . Obteve-se assim os resultados da Figura 1.

Sendo o quociente de correlação linear muito próximo de 1 para todos os exemplos, a análise experimental comprova o resultado teórico esperado.

Encontra-se também em anexo testes executados mantendo a densidade do grafo aproximadamente constante, mas variando  $N$  [anexo A].

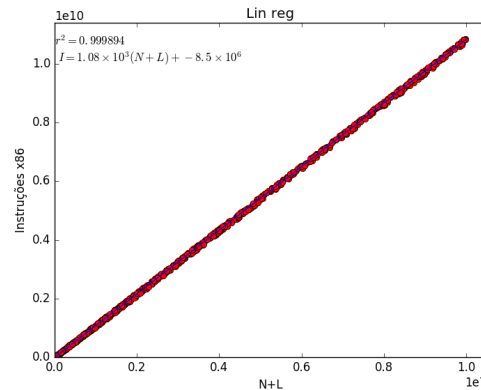


Figura 1: Análise experimental com parâmetros  $2 \leq N \leq 10000$  (1000 pontos)

## Análise assintótica espacial

Como se verifica pelas estruturas utilizadas, o algoritmo utiliza 3 estruturas com  $N$  inteiros (disco, low, parent), 1 estrutura com  $N$  booleanos (AP), e uma estrutura com  $N$  vetores, mas com no total  $L$  inteiros no interior. Durante a recursividade, nunca é atingido um nível de profundidade superior a  $N$ , pelo que a memória de stack tem complexidade de pior caso  $O(N)$  (num grafo linear). Assim a complexidade espacial será  $O(N + L)$ . Se não for contabilizada a memória utilizada para representar o grafo, o algoritmo tem complexidade espacial  $O(N)$ .

## Prova de otimalidade do algoritmo

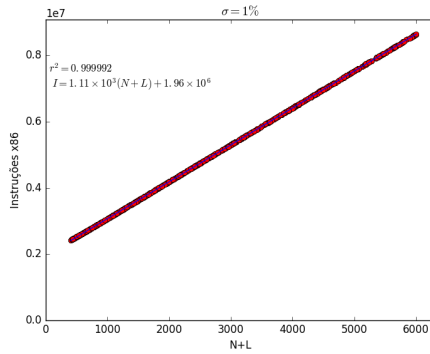
Fácilmente se verifica que o algoritmo é ótimo com base no lema "o algoritmo ótimo para verificar se um nó é nó de corte tem complexidade de pior caso  $\Omega(N + L)$ ". Se para encontrar num grafo todos os nós de corte se tivesse complexidade assintoticamente inferior a  $O(N + L)$ , então seria possível utilizar esse algoritmo para verificar se um nó é nó de corte em tempo de pior caso inferior a  $O(N + L)$ , o que contradiz o lema, logo a complexidade estará em  $\Omega(N + L)$ . Pela nossa prova de correção e análise assintótica, o algoritmo de Tarjan corre em pior caso em  $O(N + L)$ , sendo portanto uma solução ótima ■

Uma demonstração do lema baseia-se em não ser possível verificar para um grafo geral que  $u$  é um nó de corte sem de facto percorrer todos os nós e arestas pelo menos uma vez, o que implica a complexidade de pior caso de  $\Omega(N + L)$ .

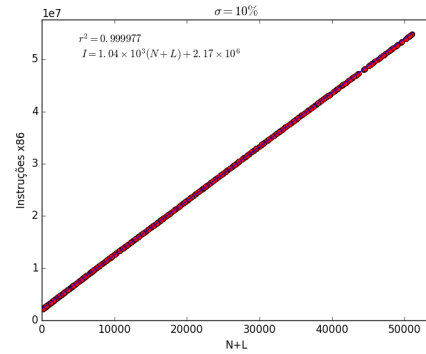
## Referências

- [1] Programming languages - c++. *ISO/IEC 14882:2003*, page 879, 2003.
- [2] John Hopcroft and Robert Tarjan. Algorithm 447: Efficient algorithms for graph manipulation. *Commun. ACM*, 16(6):372–378, June 1973.

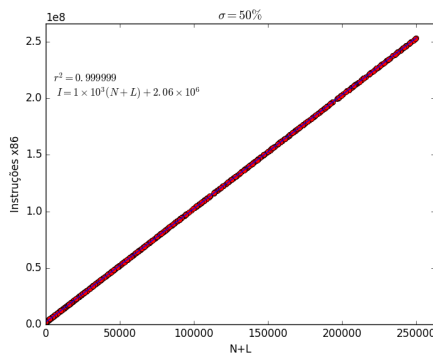
## ANEXO A - Análise experimental para $\sigma$ fixo



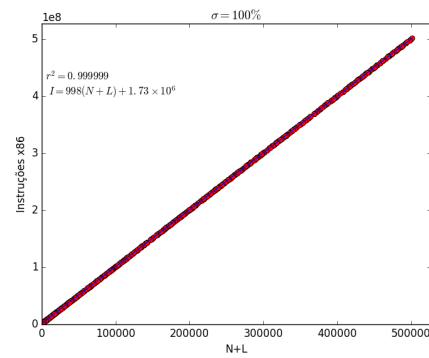
(a) Análise experimental com parâmetros  $202 \leq N \leq 1000$  (1000 pontos) e  $\sigma = 1\%$



(b) Análise experimental com parâmetros  $22 \leq N \leq 1000$  (1000 pontos) e  $\sigma = 10\%$



(c) Análise experimental com parâmetros  $6 \leq N \leq 1000$  (1000 pontos) e  $\sigma = 50\%$



(d) Análise experimental com parâmetros  $4 \leq N \leq 1000$  (1000 pontos) e  $\sigma = 100\%$