1º Projecto ASA: Relatório
Grupo 22
João Miguel P. Campos, nº 75785

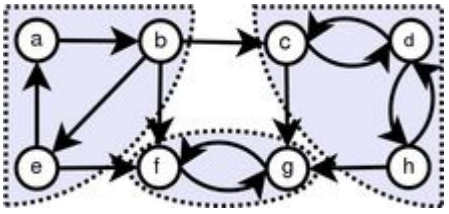
Introdução:

O Sr. João Caracol pretende dividir a sua rede de supermercados, que actualmente é enorme, em várias sub-redes regionais, de forma a que numa região seja possível qualquer ponto de distribuição enviar produtos para qualquer outro ponto da rede regional.

Se houver um ponto *u* da rede de distribuição com uma rota para um ponto *v*, e o ponto *v* tem uma rota para o ponto *u*, então ambos pertencem à mesma sub-rede regional.

Descrição da Solução:

A forma mais fácil de abordar, e mais lógica, foi criar um grafo, onde cada ponto da distribuição é considerado como sendo um vértice (*V*), cada rota é considerada como uma aresta (*E*) e procurar sub-redes existentes na rede de supermercados é equivalente a encontrar componentes fortemente ligados (*SCC*) num grafo.



Um componente é considerado um *SCC*, se todos os vértices são atingíveis a partir de todos os outros vértices.

Com esta analogia, decidi utilizar o algoritmo de Tarjan, que utiliza uma pesquisa em profundidade (*DFS*) um pouco modificada. À medida que o algoritmo vai correndo, são devolvidos os *SCC,* ou sub-redes regionais. Para poder dar o resultado pedido, tive que verificar depois se haviam ligações entre sub-redes regionais, ou seja, verificar se haviam arestas a conectar um *SCC* a outro *SCC*.

Análise Teórica:

Para realizar o algoritmo, criei uma estrutura de um vértice ***u*** que continha informação importante para o algoritmo, nomeadamente o valor de ***low*** - guarda o vŕrtice com menor índice atingível a partir ***u*** - e de ***discovery*** - guarda o tempo de descoberta.

No algoritmo de Tarjan, começamos por visitar o vértice inicial (foi sempre considerado o vértice 1) ***u***. Sempre que se visita um vértice, este é colocado numa *stack* do tipo *LIFO (Last In First Out)* e, caso o vértice adjacente, ***v,*** ainda não tenha sido descoberto, continua a procura*,* e $u.low$ é atualizado segundo a condição $min(v.low,\ u.low)$**.** Caso contrário, se o vértice ***v*** já estiver na nossa *stack,* significa que já foi visitado e apenas se atualiza $u.low\ =\ min(v.low,\ u.low)$**.** Todo este procedimento é apenas uma *DFS* com alguma manipulação e, como tal, tem uma complexidade de $O(|V|+|E|)$, onde V representa o número de vértices do grafo e E representa o número de arestas do grafo.

Depois de acabar esta procura, o algoritmo vai começar a retirar elementos da *stack* até encontrar um elemento que tenha $w.low\ ==\ w.discovery$**,** fazendo também pop deste. Todos os vértices retirados desta forma pertencem à mesma sub-rede, ou seca, são um *SCC.* Esta acção é feita em tempo constante $O(1)$.

Todo este processo é repetido para todos os vértices do grafo, ou seja, tem uma complexidade de $O(V)$.

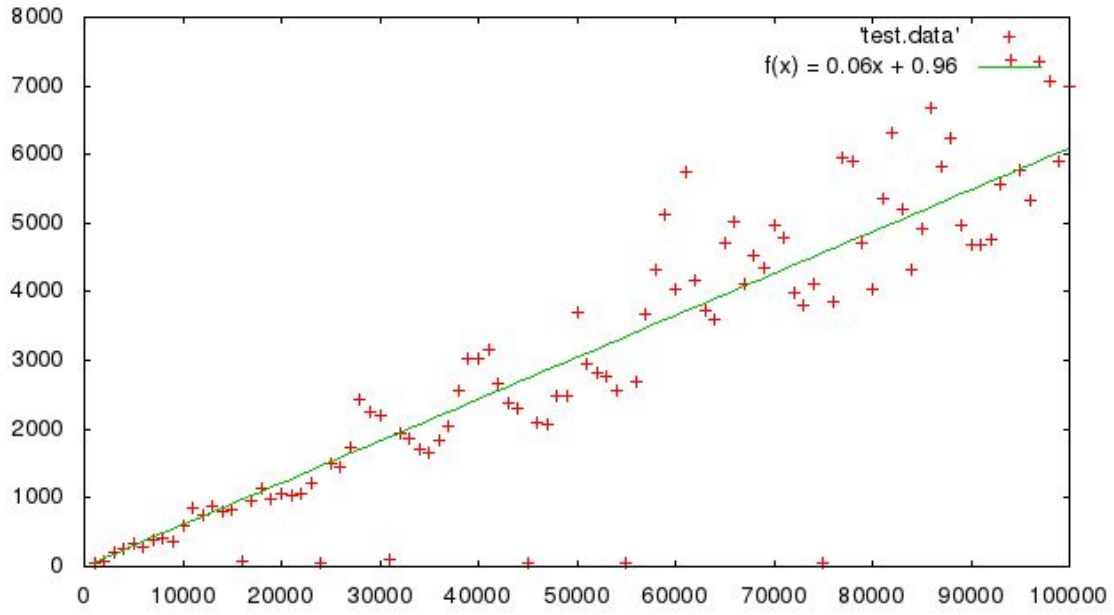
Também foi necessário, após conclusão do algoritmo de Tarjan, procurar possíveis ligações entre os vários *SCC*, algo que foi conseguido em $O(V)$.

Após obter as ligações entre sub-redes, foi executado um algoritmo nativo do c++ para ordenação, que ordena com complexidade $O(n\ log(n))$**.**

Assim, todo o programa corre com complexidade total $O(|V|+|E|)$.

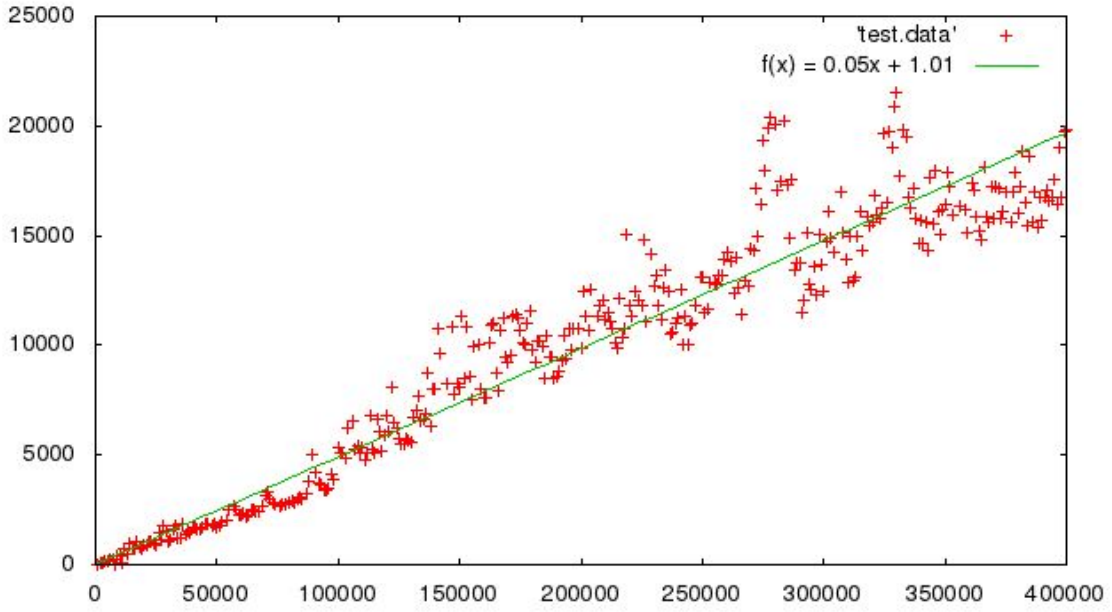
Análise Prática:

Todos os gráficos aqui mostrados têm, no eixo dos yy, o tempo de execução (em ms) e no eixo dos xx, o nº de vértices usado para gerar os testes. O número de arestas foi um número aleatório entre dado pela fórmula $\#arestas = (\#vertices * 5) + \#vertices$ apenas para haver um número mais diverso de arestas. O número de sub-redes é um número aleatório entre $\frac{\#vertices}{\#arestas} < \#SCC < \#vertices$. O número máximo de vértices foi 400000 e foi gerado de 1000 em 1000. Cada teste foi executado 5 vezes, para ter um tempo de execução mais correcto.



Teste de execução de 1000 em 1000, até chegar aos 100000 vértices.

Há alguns desvios devido a utilização do computador, que retira potencialidades computacionais ao mesmo, e outros testes que, possivelmente resultaram em erro, então o seu tempo de execução é bastante mais reduzido que o dos restantes.



Teste de execução de 1000 em 1000, até chegar aos 400000 vértices.

Os desvios devem-se, mais uma vez, à utilização do computador e do processador. Neste teste, filtrei alguns erros emitidos pelos testes, pois eram os únicos com tempos de execução na ordem do 40ms, relativamente ao número bastante grande de vértices que estava a ser testado.

Penso que, de uma forma geral, é possível comprovar os resultados teóricos, ou seja, a complexidade do programa é linear, mais concretamente, $O(|V|+|E|)$.