Análise e Síntese de Algoritmos – 1º Projecto

Grupo 118 (Alameda)
Pedro Centeno – 86501
Vladyslav Shumanskyy – 86526

Introdução

Este relatório aborda a solução encontrada para o problema proposto pelo corpo docente da UC como primeiro projecto no ano curricular 2017/18.

Problema: Considerando as actuais rotas de distribuição de produtos de uma cadeia de supermercados, pretende-se identificar todas as sub-redes regionais desta e as ligações entre as mesmas. Dois pontos, *u* e *v*, entendem-se como pontos de distribuição de uma sub-rede caso exista, pelo menos, uma rota de distribuição entre o primeiro e o segundo, e vice-versa.

Descrição da solução

A implementação do algoritmo foi elaborada em linguagem C++.

Sucintamente, interpretou-se a rede como um **grafo dirigido, não pesado**, sendo cada ponto de distribuição um vértice do mesmo. As sub-redes corresponderão às componentes fortemente ligadas – doravante denominadas *SCCs* (*strongly connected components*) – do grafo e as ligações entre estas serão arestas entre os vértices identificadores de cada SCC (os de menor índice).

Considerando um grafo G como G = (V, E), com V o conjunto de vertices e E o conjunto de arestas, este foi representado internamente como 3 arrays: um de arestas ordenado (pares de vértices, de saída e de entrada) suplementado por um array de vértices (para reutilização dos objetos e execução do algoritmo) e por um array de índices - denominado por adjacência – que, dada uma aresta (u, v), guarda o índice inicial para qualquer u pertencente a V com *degree*(u) > 0 e o seu grau. Foi escolhida esta representação porque depois de lido, o grafo não seria modificado, caso contrário, uma lista de adjacências seria mais flexível.

Este algoritmo foi feito com base numa aplicação de algoritmo de Tarjan seguida de uma atualização de índices e reordenação do input. Formalmente, os passos incluem:

1. A leitura do input, alocação de memória e inserção de vértices e arestas;
2. Ordenação do array de arestas usando *CountingSort,* visto que o tamanho do universo do input sabia-se *a priori;*
3. Uma passagem do algoritmo de Tarjan pelo grafo, de modo a encontrar as SCCs do mesmo;
4. O cálculo do identificador da cada SCC – que trataremos por *minKey* – durante a remoção de vértices da pilha. Ao mesmo tempo são alterados permanentemente os identificadores dos vértices dessa SCC;
5. Ordenação 2 vezes (RadixSort) usando o mesmo algoritmo do ponto (2), desta vez pelo vértice de entrada e de seguida pelo vértice de saída;
6. Percorrer o array de arestas para calcular o número de ligações úteis entre SCCs;
7. Escrita do output e libertação da memória alocada.

O algortimo **não** preserva o input original.

Análise e Síntese de Algoritmos – 1º Projecto

Grupo 118 (Alameda)
Pedro Centeno – 86501
Vladyslav Shumanskyy – 86526

Análise teórica

Passos (desc. da solução)	Tempo	Espaço (memória)
(1)	O(V) → Inserção de vértices; O(E) → Leitura e inserção de arestas; **Total: O(V + E)**	O(V) → Array de vértices; O(E) → Array de arestas; O(V) → Adjacência; **Total: O(V + E)**
(2)	CountingSort: - size(array a ordenar) = #E - universeSize() = #V **Total: O(V + E)**	O(E) → Array auxiliar; O(V) → Array de contagem; **Total: O(V+E)**
(3)	Algoritmo de Tarjan provado como sendo O(V + E) **Total: O(V + E)**	O(V) → TarjanStack, quando #SCC = 1 e size(SCC) = #V; **Total: O(V)**
(4)	O(1) → Cálculo da minKey – simples comparacão; O(V) → Atribuição da minKey – feita em conjunção com a stack (cada *pop* da *stack* adiciona um elemento numa lista, percorrendo dita lista e atribuindo o valor) – no pior dos casos será igual ao da stack; **Total: O(V), feito com Tarjan pelo que O(V + E)**	O(V) → Lista auxiliar, com tamanho = size(SCC); **Total: O(V)**
(5)	Mesma situação que no passo (2), excepto que o algoritmo é corrido 2 vezes. **Total: O(V + E)**	Mesma situação que no passo (2); **Total: O(V + E)**
(6)	O(E) → Percorrer o array de arestas; **Total: O(E)**	N/A
(7)	O(E) → Escrita do output (percorrer arestas); O(V) → Libertação de vértices; O(E) → Libertação de arestas; **Total: O(V + E)**	N/A

Complexidade temporal da solução: O(V + E)
Complexidade espacial da solução: O(V + E)

Análise e Síntese de Algoritmos – 1º Projecto

Grupo 118 (Alameda)
Pedro Centeno – 86501
Vladyslav Shumanskyy – 86526

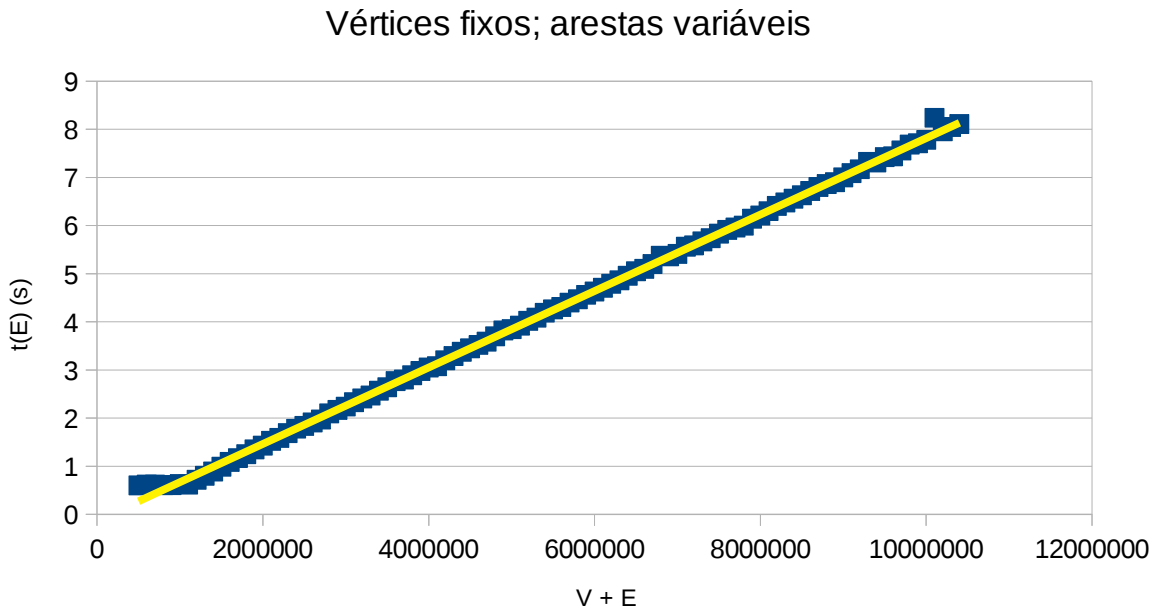
Análise experimental dos resultados

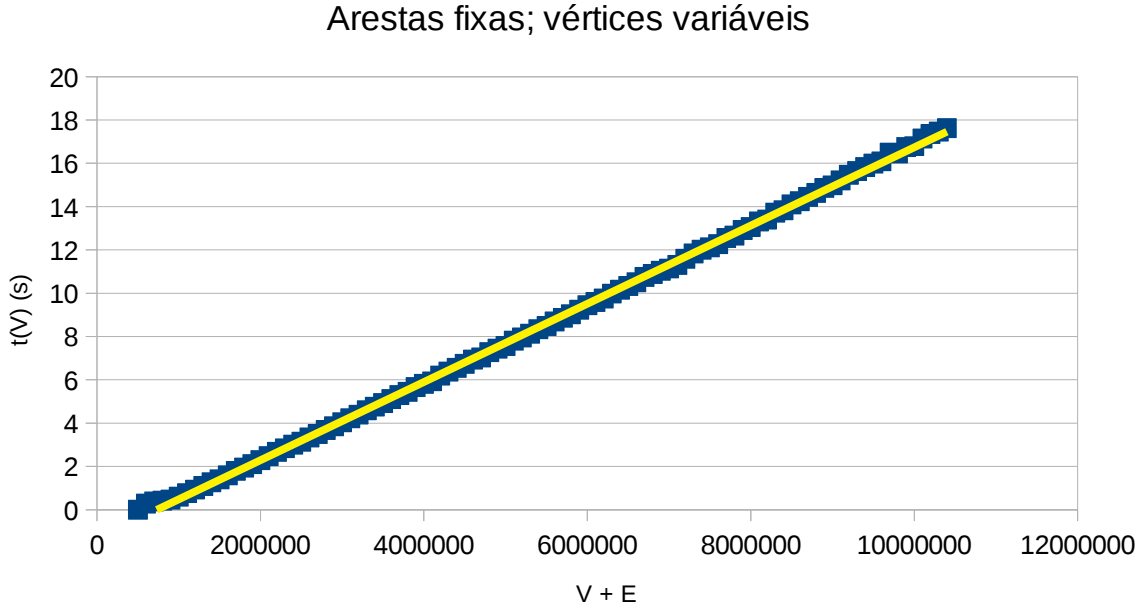
Os testes foram realizados em 3 tipos de ambiente:

→ Com o #V fixo e #E a variar linearmente (entre 100 a 10.000.000);
→ Com o #E fixo e #V a variar linearmente (entre 100 a 10.000.000);
→ Com ambos a variar linearmente, sendo #V = #E (entre 100 a 10.000.000);

O binário utilizado não escrevia output, para calcular apenas o tempo de computação.
Os testes foram realizados em modo linha de comando.

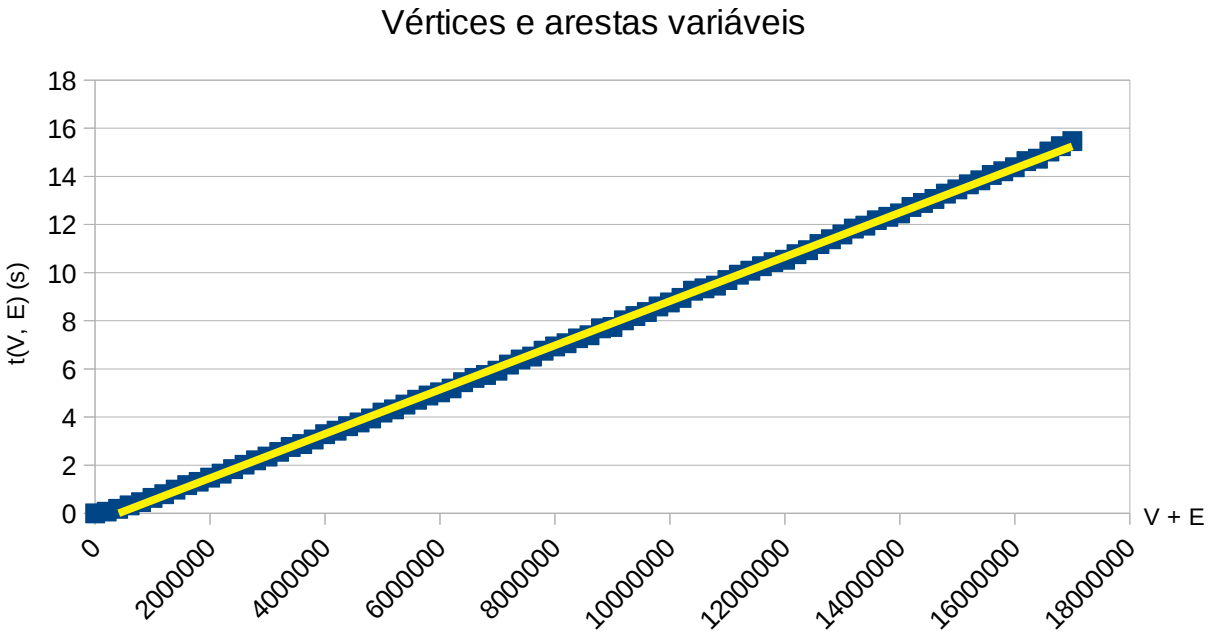
Todos os testes foram automatizados através de um *script* e correram na mesma máquina, cujas especificações estão listadas abaixo:
CPU → AMD Ryzen 7 1700X @ 3.8 GHz (8C / 16T)
RAM → DDR4 2 x 8GB @ 3200 MHz 14-14-14-36

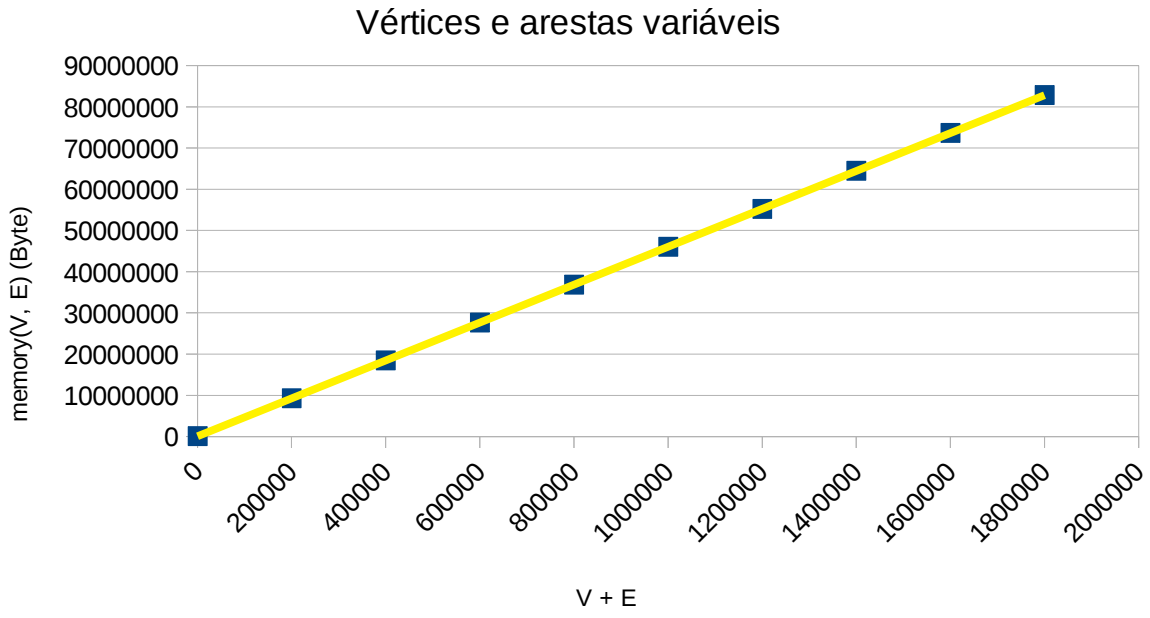




Análise e Síntese de Algoritmos – 1º Projecto

Grupo 118 (Alameda)
Pedro Centeno – 86501
Vladyslav Shumanskyy – 86526





Referências:

https://en.wikipedia.org/wiki/Tarjan%27s_strongly_connected_components_algorithm

http://en.cppreference.com/w/

http://www.cplusplus.com/