INSTITUTO SUPERIOR TÉCNICO

Análise e Síntese de Algoritmos

Relatório 1º Projeto (2017/2018)

Grupo: 142 (Alameda) | Francisco Sena 86420

23 de março de 2018

Introdução

O problema que foi proposto consiste em dividir a rede de distribuição da cadeia de supermercados do Sr. João em sub-redes regionais de forma a que numa sub-região seja possível qualquer ponto de distribuição "u" enviar produtos para qualquer outro ponto "v" da rede regional.

O objetivo é ajudar o Sr. João nesta tarefa desenvolvendo um programa que ao receber do *standard input* uma linha com o número de pontos de distribuição da rede N (N>=2), uma linha com o número de ligações na rede de distribuição M (M>=1) e por fim uma lista de M linhas, em que cada linha contém dois inteiros "u" e "v" (indicando que os produtos podem ser transportados do ponto "u" para o ponto "v" da rede de distribuição), calcula as ligações entre sub-redes regionais.

Descrição da Solução

O algoritmo utilizado na resolução do problema foi o algoritmo de *Tarjan*. Tendo em conta que o outro algoritmo estudado nas aulas teóricas tem a mesma complexidade assimptótica que o escolhido, o critério de escolha foi ao pensar no contexto da implementação dos algoritmos num sistema real, pois na prática, o *Tarjan* tem resultados melhores que o outro algoritmo, no que é respetivo ao tempo de execução.

Utilizou-se uma lista de adjacências como estrutura de dados para representar o grafo dirigido, em que cada vértice V representa um ponto de distribuição da rede e as arestas E representam caminhos de "v1" para "v2". São inicializadas as seguintes estruturas de dados com respetiva descrição antes da execução do algoritmo de *Tarjan*:

- Vetor de *vertexes* (ponteiros para "struct vertex_v") que tem como campos: boolean, d, h, scc_val. Os três primeiros campos contêm o estado do vértice ao longo do algoritmo de *Tarjan*: se está ou não na pilha, tempo de descoberta e valor da heurística. O último campo irá guardar o representante da componente fortemente ligada a que o vértice pertence.
- Vetor *edges* de estruturas "struct Edge", que consiste em 2 inteiros "u" e "v", ou seja, representa uma ligação do vértice "u" para "v";
- Vetor de "links" de tamanho N - lista de adjacências. É utilizada apenas para executar o algoritmo de *Tarjan* em tempo linear;
- Uma pilha "*stack*" onde irão estar os vértices que são visitados durante a execução do algoritmo.

Para resolver o problema de representar cada componente fortemente ligada pelo valor mínimo dos vértices que pertencem à componente, prosseguiu-se da seguinte forma: quando é descoberta uma componente fortemente ligada, declara-se uma variável "min" (onde irá estar o representante da componente) e aloca-se um vetor *temp* para "n" inteiros, onde "n" é o valor de vértices na stack (pode ocorrer que todos os vértices presentes na pilha formem uma SCC). No ciclo em que são executados "pops" até encontrar o vértice que é a raiz da componente, são guardados, em cada iteração, os vértices que são libertados da pilha no vetor alocado e de seguida atualiza-se o valor de "min". Quando este ciclo terminar, executa-se outro ciclo onde todos os vértices indexados pelos valores guardados em *temp*, é-lhes atribuído no campo *scc_val* o representante da componente a que pertencem. Esta operação não altera a complexidade do algoritmo de *Tarjan* pois é O(V).

Depois da execução do algoritmo de *Tarjan*, aloca-se um vetor *scc_connections* de "struct Edge", onde irão estar representadas as ligações entre componentes fortemente ligadas. Os campos "u" e "v" de cada ligação são os representantes das componentes respetivas que têm uma ligação entre elas.

De seguida, executa-se um algoritmo de ordenação ao vetor de *edges*. Na verdade, o vetor não irá ficar ordenado usando como critério de comparação os seus campos "u" e "v", mas sim o valor do campo *scc_val* dos vértices que constituem as componentes (EDGE_1(u), EDGE_2(u)). Uma ligação é "menor" que outra se o campo *scc_val* do vértice correspondente ao campo "u" dessa ligação for menor que o campo *scc_val* do vértice "u" da ligação em comparação. Em caso de empate, é escolhido o menor usando a mesma ideia mas recorrendo ao campo "v". Para encontrar todas as ligações entre componentes fortemente ligadas, basta percorrer o vetor de vértices indexado pelos campos "u" e "v" das M arestas e comparar se um par de vértices têm um *scc_val* diferente, se sim, indica pertencem a componentes fortemente ligadas diferentes mas têm uma ligação. Com o algoritmo de ordenação antes executado, é fácil de não guardar em *scc_connections* ligações entre componentes repetidas.

Análise Teórica

Para analisar a complexidade do algoritmo implementado, procede-se à análise das várias secções que o constituem.

A inicialização e construção do grafo implica percorrer V + E + V, ou seja, O(V+E), sendo que na a inserção de uma aresta na lista de adjacências é efetuada em tempo constante, O(1).

A execução do algoritmo *Tarjan* é em tempo linear, O(V+E) porque basta percorrer todos os vértices e, para cada vértice, percorrer todas a suas adjacências. A solução utilizada para encontrar o representante de cada componente e atualização do campo *scc_val* de todos os vértices tem complexidade Θ(V) pois tanto quando o grafo tem apenas uma componente fortemente ligada ou quando tem "N" componentes fortemente ligadas, o ciclo onde se irá atualizar o campo de todos os vértices com o representante da sua componente, na totalidade, irão executar-se "N" iterações, ou seja, Θ(N). Portanto, não altera a complexidade do algoritmo.

O algoritmo de ordenação utilizado foi o *Quicksort* da "stdlib" do C. Este algoritmo tem complexidade Ω(E*log(E)) e Θ(E*log(E)) no caso médio e no melhor caso possível, respetivamente. Tem, no entanto, no pior caso possível complexidade $O(E^2)$. Não é interessante estudar a complexidade do *Quicksort* em conjunto com o resto do

programa pois o impacto do mesmo varia com a ordem do input e das próprias ligações e consequentes componentes fortemente ligadas).

Por fim, imprimir as ligações entre sub-redes regionais ordenadas tem o custo de percorrer o vetor *scc_connections*, que no pior caso tem tamanho E, ou seja, O(E), que não altera a complexidade do algoritmo.

Conclui-se assim, com base na complexidade das diferentes secções, que o programa é executado em tempo linear O(V+E) com a pequena nuance do *Quicksort*.

Inspeção da eficiência da solução em função de diferentes tipos de inputs

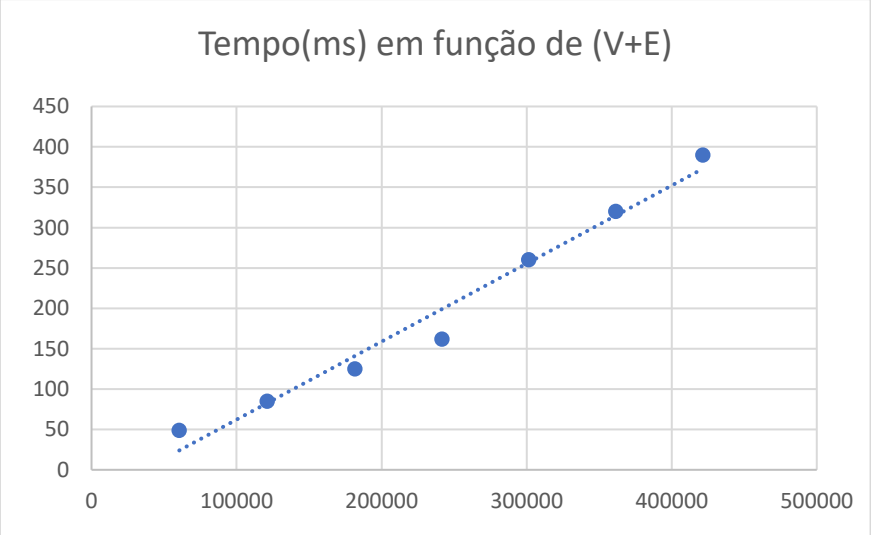
- Grafos densos - |V| muito menor que |E|, prevê-se que o que domine o tempo de computação seja a execução do algoritmo de ordenação que tem complexidade Ω(E*log(E)), pois E*logE > (V + E) que se pode considerar, para o estudo assimptótico da execução do programa, apenas (E), ou seja, E*logE > E. Prevê-se que uma aproximação de função linear não irá ser satisfatória.
- Grafos esparsos - |V| muito maior que |E| é esperado, contrariamente ao caso anterior, que o que domine o tempo de computação seja o algoritmo de *Tarjan*, ou seja, O(V+E).

Análise Experimental dos Resultados

Procedendo à análise do programa, realizaram-se vários testes utilizando o gerador disponibilizado. Foi utilizado o comando *time* e foi extraído o "real time" 5 a 8 vezes para cada input, sendo calculada a média aritmética entre eles para a obtenção dos valores apresentados. De seguida, construíram-se gráficos cuja curva é a que melhor se ajusta aos mesmos.

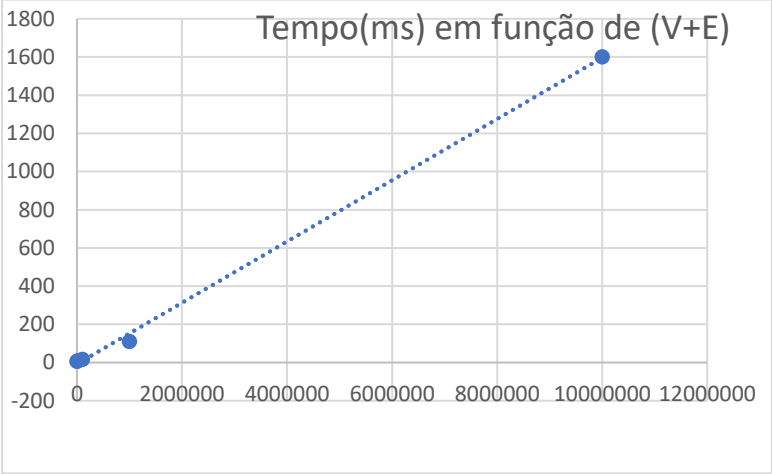
Para o estudo de grafos densos variou-se muito pouco um valor de V e aumentou-se E tanto quanto foi possível no gerador. A partir do 3º ponto, fixou-se V com o valor de 1500 e aumentou-se o E de 6000 em 6000 unidades.

A partir do 4º ponto (inclusive) verifica-se que a aproximação de crescimento linear usada passar a ser maior que linear, pelo que deixa de ser satisfatória como se previu na analise teórica.

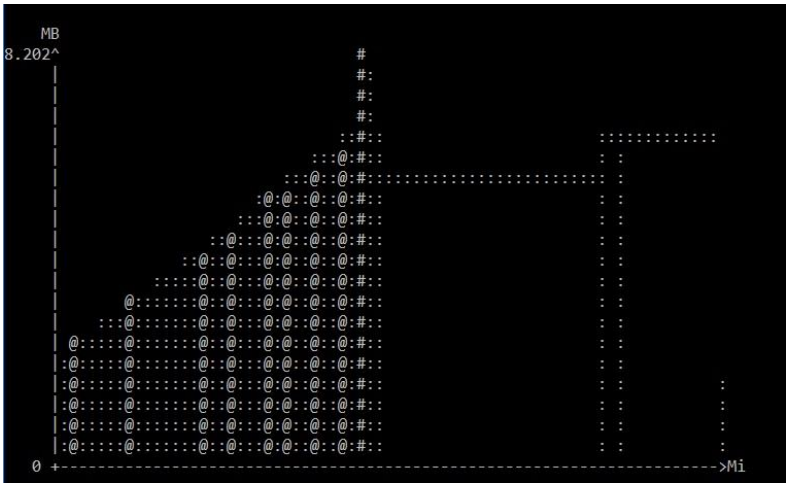


No caso de grafos esparsos fixou-se um valor de E e aumentou-se V tanto quanto foi possível no gerador. Fixou-se o valor de 1000 para E aumentou-se 10 vezes o V de ponto em ponto. Era esperado, pela análise teórica, obter-se uma aproximação de um gráfico de uma função linear, o que não se verificou. O desvio que se pode verificar deve-se provavelmente ou a fatores externos não controláveis, ou aos tipos de inputs gerados pelo gerador de modo a que o algoritmo de ordenação se comportasse de formas imprevisíveis.

Vértices (V)	Arestas (E)	Tempo (ms)
1000	100	7
10000	100	8
100000	100	17
1000000	100	110
10000000	100	1600
100000000	100	21195



Recorreu-se à ferramenta *valgrind massif* para a análise da memória do programa. Verifica-se que a evolução de ocupação de memória em função do número de instruções executadas, com um ficheiro de 2.2 KB o máximo de memoria alocada é de 8.202MB como se pode observar no gráfico.



Comentários e Referências

Na escolha da linguagem a ser utilizada, teve-se em conta o controlo que seria desejável ter sobre a memória utilizada e sobre todas as estruturas necessárias na implementação do projeto. Como tal, foi escolhida a linguagem de programação C.

A implementação da estrutura de dados “Grafo” foi baseada na implementação sugerida nos slides da cadeira de IAED 16/17. A implementação do algoritmo de *Tarjan* foi baseado no pseudocódigo sugerido nas aulas teóricas.