Trabalho Prático 2
A estrutura de "Gravata Borboleta" da Web

Luís Eduardo Oliveira Lizardo

[1]Projeto e Análise de Algoritmos
Departamento de Ciência da Computação
Universidade Federal de Minas Gerais (UFMG)

lizardo@dcc.ufmg.br

1. Introdução

A Web pode ser vista como um grafo no qual, na visão mais simples, os nodos representam páginas individuais e as arestas representam links entre as páginas. A estrutura deste grafo tem sido estudada extensivamente, e seu estudo mais completo, conduzido por [Broder et al. 2000], comparou a topologia do grafo com uma gravata-borboleta em que o maior componente fortemente conexo atua no papel do nó central da gravata.

No total, Broder et al identificou 7 tipos de componentes na estrutura da Web que dão uma visão em alto-nível de sua topologia. Os componentes identificados, ilustrados na Figura 1, foram os seguintes:

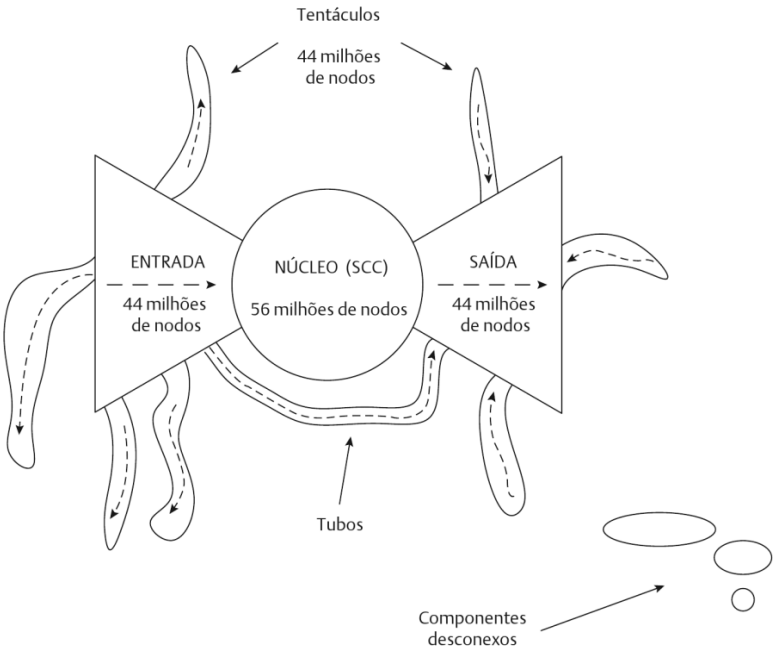

Figura 1. Estrutura de "Gravata Borboleta" da Web (fonte [de Oliveira Borges and Mourão 2013]).

- NÚCLEO (SCC): páginas que compõem o maior componente fortemente conectado do grafo. Por definição, pode-se navegar a partir de qualquer página do Núcleo para qualquer outra página do Núcleo.
- ENTRADA (IN): páginas que podem alcançar o Núcleo, mas não podem ser alcançadas por páginas nele.

- SAÍDA (OUT): páginas que podem ser alcançadas pelas páginas do núcleo, mas em um caminho para voltar ao Núcleo.
- TENTÁCULOS: São três tipos. Páginas que podem ser alcançadas a partir de páginas da Entrada (TENDRILS A); Páginas que somente alcançam sites da Saída (TENDRILS B), mas que não pertencem aos componentes anteriores; Páginas que conectam a Entrada à Saída, mas sem conexão com o Núcleo (TENDRILS C).
- DESCONEXOS (DISCONNECTED): páginas desconectadas, cujos componentes conectados podem ter uma estrutura similar a da Web como um todo.

O objetivo deste trabalho é implementar um programa para classificar links de páginas Web nos 7 componentes descritos. A entrada do programa é um grafo direcionado, onde os vértices são números inteiros que representam páginas Web e as arestas indicam se há um *link* de uma página para a outra. A saída do programa são 7 arquivos textos, um para cada componente, com a lista ordenada dos vértices pertencentes ao componente.

O restante deste relatório está organizado da seguinte forma: a Seção 2 explica o algoritmo de Busca em Profundidade, único algoritmo clássico utilizado no programa. A Seção 3 descreve a solução e faz uma análise de complexidade do programa. A Seção 4 apresenta diversas análises empíricas e a Seção 5 conclui o trabalho.

2. Referencial Teórico - Busca em Profundidade

Neste trabalho é utilizado o algoritmo de Busca em Profundidade (DFS do inglês Depth-First Search) para classificar os vértices nos 7 diferentes componentes da Web. O DFS começa por um vértice e expande através do primeiro nó filho da árvore de busca, e se aprofunda cada vez mais, até que o alvo da busca seja encontrado ou até que ele se depare com um nó folha. Então a busca retrocede (backtrack) e avança para o próximo nó filho.

O DFS apresenta complexidade de tempo igual a $O(E + V)$, onde E é o número de arestas do grafo e V o número de vértices. Mais detalhes sobre o algoritmo podem ser encontrados em [Cormen et al. 2001].

3. Solução Proposta

Esta seção explica como o programa representa o grafo da Web, identifica as estruturas da Web no grafo e classifica seus vértices.

3.1. Representação da Web

O grafo da Web foi implementado com duas listas de adjacências. A primeira registra as adjacências de cada vértice conforme a entrada. Já a segunda registra as adjacências da transposta do grafo. Essa forma de representação, com as duas listas, permite que os algoritmos de classificação das estruturas da Web possam usar o grafo normal, sua transposta e também uma representação não direcionada do grafo, ao recuperar as duas listas.

As listas de adjacências do grafo registram vértices de 0 a V inclusive, onde V é o maior vértice inserido no grafo. Dessa forma, o grafo se comporta com maior robustez em relação à falta de padrão dos arquivos de entrada sobre o domínio dos vértices (alguns grafos começam com o vértice 0 e outros do 1). Um vetor booleano registra quais os vértices foram inseridos no grafo e evita assim que esses sejam utilizados pelos algoritmos.

As listas de adjacências gastam $V + E$ cada uma, e o vetor booleano gasta apenas V bits. V é o número de vértices do grafo e E o número de arestas. Dessa forma a complexidade de espaço dessa representação é $\theta(V + E)$.

3.2. Vetor de classificação vértices

Um vetor de vértices é utilizado para marcar a qual estrutura o vértice classificado pertence. Os 7 tipos possíveis são SCC, IN, OUT, TENDRILS_A, TENDRILS_B, TENDRILS_C e DISCONNECTED. Antes de serem classificados, todos os vértices são do tipo DISCONNECTED. Esse vetor ocupa $\theta(V)$ de espaço.

3.3. Identificação do maior SCC (NÚCLEO)

Para determinar os componentes fortemente conectados do grafo (SCC), é utilizado o algoritmo de Kosaraju (Algoritmo 1). Este algoritmo executa dois DFS não recursivos, sendo o segundo no grafo transposto. O algoritmo de Kosaraju é linear, com complexidade de tempo igual $O(V + E)$ para grafos representados como lista de adjacências.

Durante a execução do algoritmo, todos os componentes fortemente conectados identificados são registrados em uma lista. Ao termino, todos os vértices do maior componente são marcados como sendo do tipo SCC e um vértice arbitrário do componente é retornado para ser utilizado na identificação das demais estruturas do grafo.

Algoritmo 1: Algoritmo de Kosaraju para encontrar os componentes fortemente conexos do grafo.

Seja G um grafo direcionado, G^T o grafo transposto de G e S uma pilha vazia;
while *S não contém todos os vértices* **do**
    Escolha um vértice arbitrário v que não esteja em S;
    Faça um DFS em G começando no vértice v. Toda vez que o DFS terminar a expansão de um vértice u, empilhe u em S;
end
while *S não está vazia* **do**
    Remova o vértice v do topo da pilha S;
    Faça um DFS em G^T começando em v;
    O conjunto de vértices visitados é um componente fortemente conexo contendo v;
    Salve o componente e remova todos os seus vértices de G e da pilha S.
end

3.4. Classificação das estruturas IN, OUT e TENDRILS

A classificação das estruturas IN, OUT e TENDRILS é feita numa única função do programa. Esta função recebe o grafo da Web, uma lista de vértices de partida e o tipo a ser classificado (IN, OUT, TENDRILS A ou TENDRILS B) e executa um DFS não recursivo sobre o grafo a partir dos vértices de partida. Dessa forma, a função descobre todos os vértices que são atingíveis a partir dos vértices de partida e classifica-os, caso não tenham sido classificados antes. Uma lista com os vértices classificados é retornada ao final da execução.

O comportamento da função depende do tipo da estrutura a ser classificada. No caso de ser IN, a função utiliza a lista de adjacências transposta do grafo. Para OUT é utilizado o grafo original. Já no caso dos TENDRILS, as duas listas de adjacências são utilizadas para que o grafo se comporte como não direcionado.

Para classificar as estruturas IN e OUT, é preciso passar para a função uma lista contendo pelo menos um vértice do NÚCLEO. Na classificação dos TENDRILS A, a função precisa receber a lista com todos os vértices pertencentes a estrutura IN. No caso de TENDRILS B, ela precisa da lista dos vértices de OUT. A classificação de TENDRILS C é feita durante a classificação de TENDRILS A ou B, quando o vértice a ser classificado como TENDRILS B já está classificado como TENDRILS A e vice versa.

O Algoritmo 3.4 mostra, de forma simplificada, a função de classificação das estruturas. Esta função executa apenas um DFS e por isso sua complexidade de tempo é, no pior caso, $O(E + V)$.

Algoritmo 2: Algoritmo para classificação das estruturas IN, OUT e TENDRILS do grafo.

```
Seja G o grafo da Web, W uma lista com os vértices de partida, T o tipo a
ser classificado e S uma pilha;
while existir um vértice w em W ainda não explorado do
    Empilhe w em S;
    while S não está vazia do
        Remova o vértice v do topo da pilha S;
        if v não foi explorado then
            Marque v como explorado;
            if v não foi classificado then
                Classifique V como tipo T;
            end
        end
        Empilhe em S, os vértices adjacentes a v;
    end
    Retorne uma lista com os vértices classificados;
end
```

3.5. Execução do programa

O programa inicia a execução lendo a entrada padrão para gerar o grafo com as duas listas de adjacências. Depois ele executa o algoritmo de Kosaraju para identificar o maior componente fortemente conectado (SCC) e classificar seus vértices. Após, o programa chama a função de classificação de estruturas e utiliza um vértice arbitrário do núcleo para calcular o IN, utilizando a lista de adjacência transposta, e o OUT, com lista original. Com os vértices das estruturas IN e OUT classificados, ele inicia a classificação dos TENDRILS utilizando as duas listas de adjacências simultaneamente. Os arquivos de saída são gerados percorrendo o vetor de classificação dos vértices e imprimindo cada vértice no arquivo correspondente a sua classificação.

O programa demanda um tempo proporcional a $V + E$ para calcular o SCC, mais um tempo proporcional a $4 \times (E + V)$ correspondente as 4 chamadas da função

de classificação das demais estruturas. Dessa forma, a complexidade de tempo do programa é $O(V + E)$, onde V é o número de vértices do grafo e E o número de arestas.

4. Avaliação Experimental

Nesta seção o programa é avaliado experimentalmente em relação ao seu tempo de execução e a qualidade das respostas.

Os testes foram realizados no sistema operacional Ubuntu 14.04, rodando num notebook com processador 3rd Generation Intel® Core™ i7-3630QM (2.40GHz 6MB Cache), 8GB RAM, HHD 1 TB S-ATA (5,400 rpm). Para cada teste, foram feitas 10 execuções e retirado a média.

Para os experimentos foram utilizadas 6 bases de dados do Projeto SNAP [1]: *soc-pokec-relationships*, *web-BerkStan*, *web-Google*, *web-NotreDame*, *web-Stanford*, *WikiTalk*; e o grafo de entrada da especificação: *example*. O número de vértices e arestas de cada grafo estão listados na tabela 1.

Tabela 1. Tamanho dos grafos utilizados no exemplo.

	Vértices	Arestas
example	30	38
soc-pokec-relationships	1.632.803	30.622.564
web-BerkStan	685.230	7.600.595
web-Google	875.713	5.105.039
web-NotreDame	325.729	1.497.134
web-Stanford	281.903	2.312.497
WikiTalk	2.394.385	5.021.410

4.1. Tamanho das estruturas

A Tabela 2 mostra o número de nodos classificados como membro de cada uma das 7 estruturas da Web. Todos os resultados, para o SCC, correspondem aos valores apontados no Projeto SNAP. O grafo de maior tamanho é o *WikiTalk*, porém àquele com o maior NÚCLEO é o *soc-pokec-relationships*.

Tabela 2. Número de nodos em cada componente da estrutura do grafo.

	SCCs	INs	OUTs	TEND. A	TEND. B	TEND. C	DISCO.
example	8	4	7	3	3	3	2
soc-pokec-relationships	1.304.537	113.341	199.758	9.090	5.839	238	0
web-BerkStan	334.857	159.709	124.974	13.530	19.041	2.671	30.448
web-Google	434.818	180.902	165.675	30.157	35.888	8.362	19.911
web-NotreDame	53.968	0	271.761	0	0	0	0
web-Stanford	150.532	32.785	67.516	1.690	2.688	54	26.638
WikiTalk	111.881	18.872	2.242.435	10.222	5.481	62	5.432

[1] http://snap.stanford.edu/data/index.html

4.2. Tempo de execução do programa

A Tabela 3 apresenta o tempo de execução do programa para cada uma das bases utilizadas. A primeira coluna mostra o tempo de *clock* total da execução, e as demais, o tempo de execução com a leitura do arquivo, classificação do SCC, classificação das estruturas IN, OUT e TENDRILS e com a geração dos arquivos de saída contendo os vértices de cada estrutura.

Pelos resultados apresentados na tabela, podemos observar que o *soc-pokec-relationships* apresentou um tempo de execução muito superior aos demais grafos, cerca de 21 segundos. Apesar de não ter o maior número de vértices, o *soc-pokec-relationships* é o grafo com o maior número de arestas entre os grafos utilizados e também o grafo com o maior SCC. Essas características explicam o porquê do tempo de execução apresentado ser tão alto em relação aos demais.

Tabela 3. Tempo médio de execução, em segundos, de cada parte do programa para diferentes entradas.

	Total	Input	SCC	In	Out	Tendrils	Output
example	0,001	0,000	0,000	0,000	0,000	0,000	0,000
soc-pokec-relationships	21,894	9,789	6,023	3,955	1,820	0,241	0,144
web-BerkStan	4,002	1,896	0,987	0,533	0,242	0,296	0,058
web-Google	3,561	1,512	0,974	0,463	0,305	0,239	0,082
web-NotreDame	0,836	0,397	0,195	0,022	0,080	0,115	0,027
web-Stanford	1,444	0,649	0,385	0,179	0,130	0,077	0,028
WikiTalk	3,833	1,535	1,039	0,177	0,430	0,456	0,205

A Figura 2 apresenta o tempo de execução do programa em comparação ao número de vértices mais arestas. Pelo grafo podemos observar que o tempo de execução do programa é linearmente proporcional a $V + E$, o que prova a análise de complexidade apresentada na Seção 3.5.

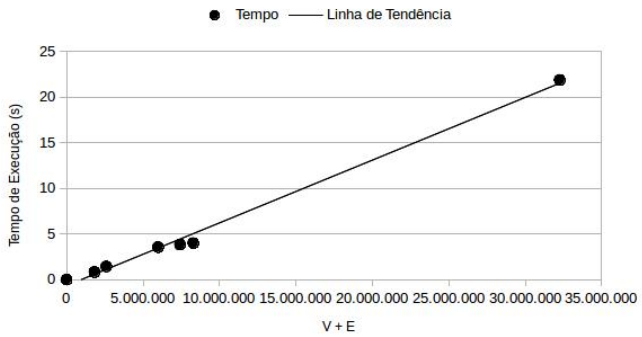

Figura 2. Relação linear do tempo de execução e (V + E)

4.3. Tempo de execução de cada parte do programa

O gráfico da Figura 3 mostra o percentual médio do tempo de execução de cada parte do programa. Pelo gráfico podemos observar que a leitura dos dados de entrada consome

40% do tempo total de execução, sendo o maior gargalo do programa. A classificação do SCC é o segundo maior gargalo, com aproximadamente 23% do tempo, seguido da escrita nos arquivos de saída, que apresenta 13% do tempo total de execução.

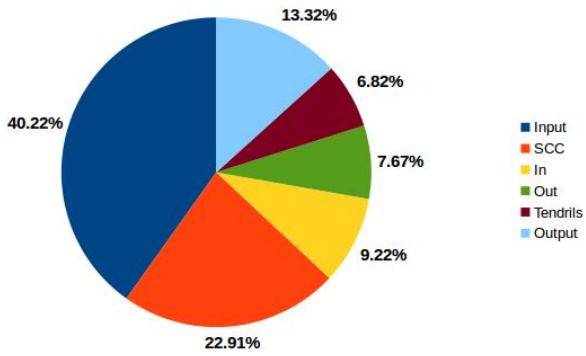

Figura 3. Percentual médio do tempo de execução por parte do programa.

5. Conclusão

Neste trabalho foi apresentado um programa para identificar estruturas em um grafo da Web. As 7 estruturas são: núcleo, entrada, saída, tentáculos (A, B, C) e vértices desconexos.

No programa, o grafo da Web foi representado com listas de adjacências, de forma que o mesmo grafo possui também a sua lista de adjacências das arestas transpostas. A classificação foi feita utilizando o algoritmo de Kosaraju para descobrir o maior componente fortemente conectado do grafo (Núcleo), e com DFS para os demais componentes.

O programa foi implementado em C++ e apresentou tempos de execuções muito rápidos. Para um grafo com 30 milhões de arestas, o programa gastou menos que 22 segundos para classificar todos os seus componentes.

Referências

Broder, A., Kumar, R., Maghoul, F., Raghavan, P., Rajagopalan, S., Stata, R., Tomkins, A., and Wiener, J. (2000). Graph structure in the web. *Computer networks*, 33(1):309–320.

Cormen, T. H., Leiserson, C. E., Rivest, R. L., Stein, C., et al. (2001). *Introduction to algorithms*, volume 2. MIT press Cambridge.

de Oliveira Borges, L. and Mourão, L. (2013). *Recuperação de Informação-: Conceitos e Tecnologia das Máquinas de Busca*. Bookman Editora.