Universidade Estadual de Montes Claros
Programa de Pós-graduação em Modelagem Computacional e Sistemas (PPGMCS)
Prof. Renê R. Veloso
Acadêmico Jorge Gustavo dos Santos Pinho

Relatório do Trabalho Prático

Montes Claros - MG,
Dezembro de 2015

Introdução

O presente trabalho propõe se a apresenta um experimento prático usando métodos de buscas tradicionais presente em algoritmos com grafos, são estes a busca em largura (BFS) e a busca em profundidade (DFS). Que serão usados para resolver o problema de alcançabilidade em um grafo real, isto é, dado o par de vértices (u, v) responder se u alcança v.

O trabalho está estruturado em três partes, a implementação, os experimentos e a conclusão, a primeira parte, implementação, descreve como foram desenvolvidas as estruturas de dados usadas para resolver o problema de alcançabilidade, na segunda parte, experimentos, são apresentados o testes aos quais o algoritmo foi submetido e as estatísticas que foram geradas com os resultados dos testes, também uma comparação entre os resultados do BFS e DFS, por fim a conclusão, que mostra se os objetivos propostos foram alcançados.

Implementação

Na implementação foi usado a linguagem de programação Python na versão 2.7.10 para codificação das estruturas de dados. O grafo foi representado como lista de adjacência usando um dicionário, que é uma estrutura de dados da linguagem Python, assim cada índice do dicionário representa um vértice e seu valor ou melhor uma lista com as adjacência, representa as arestas que saem de cada vértice.

Para os algoritmos foi criada classe de nome GTools com os principais métodos usados para o pré-processamento do grafo e também para as consultas. A GTools foi implementada com os seguintes métodos:

- ➔ load_g(self, name): usado para carregar o grafo a partir de um arquivo e retornar o grafo carregado;
- ➔ is_acicle(self, G): usado para verificar se um grafo possui ciclos, retorna True se sim e False se não;
- ➔ tarjan_scc(self, G): usado para construir o conjunto de componentes fortemente conectados do grafo G=(V, E), esse método retorna o conjunto de componentes fortemente conectados (SCCS) criados;

- ➔ dag_associate(self, G, SCCS): usado para construir o *directed acyclic graph* (DAG) associado i.e um novo grafo em que os vértices são os representantes de cada SCC e suas arestas são as ligações entre cada SCC, esse metódo retorna um $G_{DAG}=(V_{DAG}, E_{DAG})$;
- ➔ bfs(self, G, su, sv): usado para verificar a alcançabilidade entre su e sv i.e se su alcança sv usando a busca em largura (BFS), retorna True se sim e False se não;
- ➔ dfs(self, G, su, sv): também usado para verificar a alcançabilidade entre su, sv só que nesse caso usa a busca em profundidade (DFS), retorna True se alcança e False se não.
- ➔ scc_index(self, SCCS): usado para construir um índice que indica em qual componente cada elemento pertence, retorna o índice.

A primeira etapa ao usar a classe GTools é a construção do grafo, para isso é chamado o método "load_g(name)"**,** passando como parâmetro o nome do arquivo em que o grafo está armazenado. Esse método lê o arquivo linha por linha e adiciona seu conteúdo em um grafo, quando o processo é finalizado, isto é, não possui mais linhas é retornado um G(V, E).

A segunda etapa é a verificação de ciclos no grafo, para isso é usada um adaptação do algoritmo DFS, o algoritmo marca com a cor branca todas as arestas do grafo antes de começar a visita, ao iniciar a visita assim que uma aresta é descoberta ela é marcada com a cor cinza e quando é finalizada é marcada com a cor preta. Usando essas marcações a verificação de ciclos é feita verificando se a aresta que está sendo visitada já foi descoberta, isto é, se a aresta já estiver marcada com a cor cinza quando tento visita la, desta forma consigo saber se um grafo G possui ciclos. Ao tempo que um ciclo é encontrado o método interrompe o processo e retorna True, em contrapartida se nenhum ciclo for encontrado ele continua até o fim e retorna False.

Caso o grafo possua ciclos a terceira etapa é a construção do conjunto SCCS, para isso foi usado o algoritmo de Tarjan sua estrutura também usa uma DFS para determinar os componentes fortemente conectados, no final da construção ela retorna um conjunto de SCCS.

A quarta etapa é a construção do DAG associado onde os vértices são gerados a partir do representante de cada SCC e as arestas são criadas usando as as ligações de um SCC com outro.

A quinta etapa é a construção do índice para retornar em tempo constante qual o componente um vértice qualquer pertence. O uso do índice é feito para otimizar o processo de alcançabilidade, que nesse caso usa o representante de cada SCC para a verificação.

Por fim na sexta etapa são feitas as consultas de alcançabilidade usando BFS e DFS, o processo é feito da seguinte forma, são passados dois vértices u, v para BFS e para a DFS, e ambas devem responder se u alcança v. Assim a busca não é feita diretamente no grafo G, primeiro busca se o representante de u e o de v no índice depois testa se o representante de u alcança o representante de v no DAG associado, um detalhe importante é que se os representantes forem iguais não precisa ser realizada a busca pois vértices do mesmo componente são acessíveis em ambas as direções, i.e, u alcança v e v alcança u.

As buscas BFS e DFS são buscas tradicionais e que mostra um tempo razoável para consultas em DAGs associados para grafos não muito grandes, no entanto para aplicações práticas podem não ser uma boa escolha, como esse trabalho é de caráter experimental não é seu escopo analisar ou comparar outras estruturas de dados usadas em verificação de alcançabilidade.

No próximo capítulo são apresentados os testes submetidos nos algoritmos, os tempos de cada consulta, o tempo para carregar o grafo, os SCCS, a criação do DAG associado e a criação dos índice.

Experimentos

Os testes realizados foram feitos com o grafo web-NotreDame que possui 325729 vértices e 1497134 arestas. Seus vértices são formados pelas páginas da *University of Notre Dame* (domínio nd.edu), e suas arestas são os *hiperlinks* entre as páginas. O grafo foi construído a partir de uma coleta feita em 1999 por Albert, Jeong e Barabasi, disponível em http://snap.stanford.edu/data/web-NotreDame.html.

Para realização dos testes foi usado um computados com processador Core i5-4670K, 3.40 GHz, com 4 núcleos, 16GB de memória RAM, 2 HDs SSD de 120GB em RAID 0 e Sistema operacional Windows 7 64 bits. Não foi usado threads para implementação dos algoritmos, a linguagem usada, como já citado, foi a Python em sua versão 2.7.10.

Para realização dos testes foi feito um pré-processamento do grafo, que seguiu as etapas: carregar o grafo a partir de um arquivo de texto; verificar ciclos (como o grafo apresentou ciclos, foi usado o algoritmo de Tarjan para construído os seus SCCS); criar SCCS; construir DAG associado; e criar o índice usado para buscar o representante de cada vértice.

Os tempos gastos nas etapas citadas são apresentados na Tabela 1.

Tabela 1 : Tempo gasto para pré-processamento do grafo

Operação	**Tempo**
Carregar o web-NotreDame	1.072999s
Verificar ciclos	0.059999s
Construir SCCS	672.868000s
Construir DAG associado	162.126000s
Verificar ciclos no DAG	0.215000s
Construir índices	0.068000s
Total Total	**836.409998s**

Todo o tempo gasto no pré-processamento do grafo levou cerca de 14 minutos, tendo o etapa de construir o conjunto de SCCS e de construir o DAG associado como as operações que mais consumiram tempo.

O pré-processamento gerou um conjunto de SCCS com 203609 componentes, a construção do DAG gerou um grafo com 203609 vértices e 239438 arestas. A verificação de ciclos no DAG foi feita para validação , isto é, verificar se a implementação do algoritmo de Tarjan estava ou não correta.

A próxima etapa dos testes foi a realização das consultas, para isso foi fornecido um arquivo de texto composto por 100000 consultas de alcançabilidade, cada linha do arquivo era composta por dois valores separados por espaço que representam respectivamente u e v. O teste então era verificar se u alcança v usando o BFS e o DFS, para isso seleciona se o

representante de u e o de v usando o vetor de índice e aplica se as buscas passando como parametro os representantes de cada componente e o DAG associado.

A Tabela 2 apresenta os resultados das consultas realizadas para o BFS e para o DFS, mostra o número de consultas positivas e negativas, o tempo médio gasto para as consultas positivas e o tempo médio para as consultas negativas, o tempo médio das consultas no geral é o tempo total gasto.

Tabela 2 : Resultados da consultas

Método	C. Positivas	C. Negativas	T. M. Positivas	T. M. Negativas	T. M. Total	T. Total
BFS	14688	85312	0.477885s	0.000033s	0.070220s	7022.055999s
DFS	14688	85312	0.053173s	0.000185s	0.007968s	796.875000s

C. Positivas: Consultas Positivas

C. Negativas: Consultas Negativas

T. M. Positivas: Tempo Médio Consultas Positivas

T. M. Negativas: Tempo Médio Consultas Negativas

T. M. Total: Tempo Médio Consultas

T. Total: Tempo Total

Ao analisar os dados dos experimento podemos ver que a quantidade de consultas positivas e negativas tanto para o BFS como para o DFS são os mesmo, como era de se esperar.

Em relação ao tempo gasto para realizar as consultas, o BFS apresentou um tempo muito bom para consultas negativas, cerca de 0.000033 segundos, no entanto para as consultas positivas seu tempo chegou a quase meio segundo, cerca de 0.477885 segundos. O DFS também apresentou um tempo muito bom para as consultas negativas, maior um pouco que o tempo do BFS mais não muito, cerca de 0.000185 segundos. Ao compara o tempo de DFS com o BFS para consultas positivas podemos notar uma diferença significativa, enquanto o BFS leva cerca de quase meio segundo o DFS leva apenas 0.053173 segundos. Essas diferenças podem ser melhor visualizadas no Gráfico 1 que apresenta o tempo médio para consultas positivas BFS e DFS, e no Gráfico 2 que mostra o tempo médio para consultas negativas.

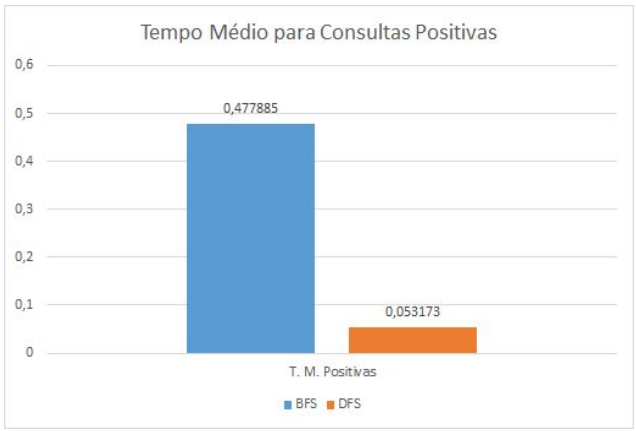

Gráfico 1: Tempo médio para consultas positivas BFS e DFS

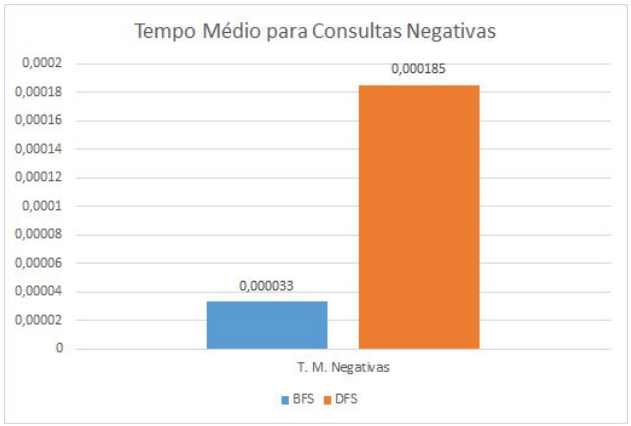

Gráfico 2: Tempo médio para consultas negativas BFS e DFS

Na comparação do tempo médio total por consulta dos dois métodos, o DFS apresenta uma média significativamente melhor do que o BFS, ficando o DFS em cerca de 0.007968 segundos e BFS em cerca de 0.070220 segundos por consulta, isso pode ser visualizado no Gráfico 3.

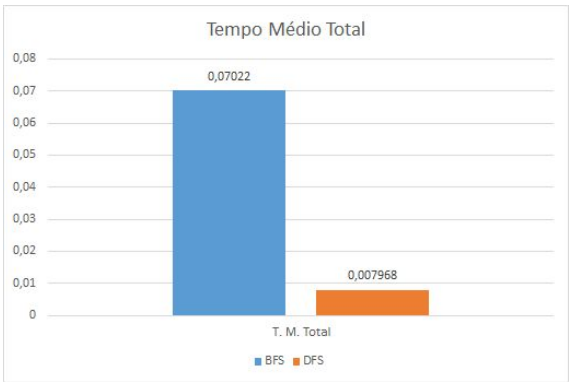

Gráfico 3: Tempo médio total das consultas BFS e DFS

Por fim a diferença entre os tempos dos dois métodos pode então ser vista com maior clareza quando comparado o tempo total gasto para realizar as 100000 consultas de alcançabilidade. Enquanto a BFS gastou cerca de 1,9 horas para realizar todas as consultas o DFS gastou cerca de 0, 22 horas, essa diferença pode ser visualizada no Gráfico 4.

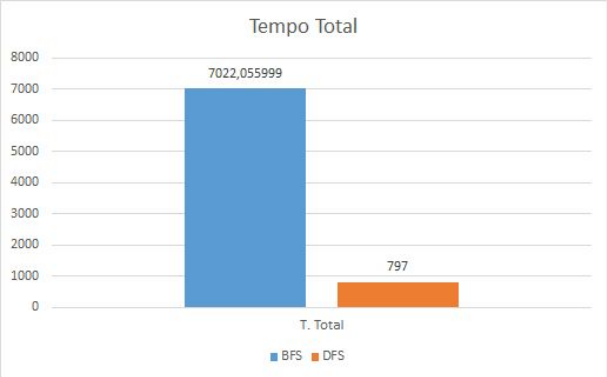

Gráfico 4: Tempo total das consultas BFS e DFS

A conclusão a que se chega é que o DFS se saiu bem melhor nos testes de alcançabilidade, apresentando um desempenho muito superior ao BFS.

Conclusão

Como proposto, o trabalho apresentou a implementação de uma estrutura de dados para responder aos testes de alcançabilidade usando buscas BFS e DFS, mostrou também os testes que foram realizados e o seus resultados. Assim foi possível constatar a possibilidade de responder à pergunta de alcançabilidade usando as buscas BFS e DFS, no entanto dependendo do tamanho do grafo e do tempo necessário para a resposta pode não ser viável o uso desses métodos.

A implementação dos algoritmos usados no trabalho pode ser visualizada no endereço https://github.com/gustavopinho/G-Tools, lá também está presente o arquivo do web-NotreDame e o arquivo com as consultas usadas no trabalho.

Assim conclui-se que o trabalho cumpriu com exito os seus objetivos de: codificar a estruturas de dados Tarjan, codificar os métodos BFS e DFS; executar experimentos e apresentar os resultados.

Referências

CORMEN, Thomas H. **Algoritmos: teoria e prática.** 2.ed. Rio de Janeiro: Elsevier, 2002.