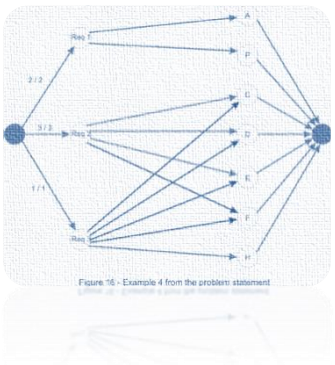
Análise e Síntese de Algoritmos

Relatório Projecto 2 – *Jobs*



Autor: João Pinho

Número: 66047

Introdução

O problema deste segundo projecto, consistiu na implementação de um algoritmo que permite maximizar o número de empregos atribuídos a estudantes de acordo com a sua preferência/interesse em cada emprego oferecido.

Descrição do Algoritmo

A implementação deste algoritmo, implica a criação de um grafo bipartido G, implementado como uma lista de adjacências.

No grafo G, foi criado um vértice para cada estudante e para cada oferta de emprego, correspondendo os primeiros ao conjunto U (dos estudantes) e os segundos ao conjunto V (das ofertas de emprego).

Para n ofertas de emprego e m estudantes, os vértices do grafo de 0 até N representam as ofertas de emprego e os vértices de n até n+m representam os estudantes, os índices foram organizados desta forma por mera conveniência já que os números das empresas começam em zero e os dos estudantes começam num número arbitrário.

Para determinar o início de cada partição existe um *array* de dois inteiros na estrutura do grafo que permite identificar o início da primeira e segunda partição, U e V, respectivamente.

O algoritmo de fluxos máximos, determina a cardinalidade do número máximo de atribuições de estudantes a empregos, por etapas. Cada etapa, consiste na realização dos seguintes passos:

1. Uma BFS separa os vértices do grafo em camadas. Os vértices livres (i.e, sem correspondências/atribuições) em U, são utilizados como vértices fonte no início de cada procura em largura, e formam a primeira da camada do conjunto U. No primeiro nível da procura só podem ser percorridos os arcos que ainda não tenham sido percorridos previamente. Quanto efectuamos uma procura nas adjacências (sucessores) de um vértice em U, apenas os arcos que ainda não estão correspondidos a um vértice de V podem ser percorridos, enquanto que, a partir de um vértice em V

apenas os arcos correspondidos (*matched*) podem ser percorridos. A procura termina na primeira camada K, onde um ou mais vértices livres em V são atingidos.

2. Todos os vértices livres em V na camada K são agrupados num conjunto F, ou seja, todo o vértice v pertencente a F, termina um caminho de aumento mais curto;

3. O algoritmo encontra o maior conjunto de caminhos de aumento de tamanho K, que são disjuntos de um conjunto F constituído por v vértices. O conjunto é calculado através do uso de uma DFS que parte de F para todos os vértices livres de U, para guiar esta pesquisa são utilizadas as camadas descobertas previamente pela BFS. Desta forma a DFS só pode seguir por arcos que conduzam a vértices não utilizados na camada anterior, e os caminhos na árvore da DFS têm de alternar entre os arcos correspondidos (*matched*) e os não correspondidos (*unmatched*). Assim que é encontrado um caminho de aumento que envolva um dos vértices em F, a DFS continua a partir do vértice seguinte em F.

4. Cada um dos caminhos descobertos desta forma é utilizado para aumentar o conjunto o conjunto M, constituído por U e V.

O algoritmo termina quando já não são encontrados mais caminhos de aumento pela BFS, na parte da procura de uma das etapas.

Este algoritmo foi baseado numa publicação conhecida, designada por algoritmo de Hopcroft-Karp, algoritmo que escolhi por ser aquele que permite obter melhores condições em termos de tempo e espaço, face às alternativas: Ford-Fulkerson, Edmonds-Karp ou uma minha que tive de modificar para a implementação actual, pois no algoritmo que implementei inicialmente de forma recreativamente, uma BFS que encontrava vários caminhos de aumento em cada iteração não era suficiente, até que após inúmeras investidas, ocorreu-me que uma DFS sobre os caminhos encontrados pela BFS poderia optimizar a solução, altura esta portanto, em que decidi implementar o algoritmo de Hocroft-Karp por ter conhecimento por alto que o principio seguido seria o mesmo.

Referências:
http://en.wikipedia.org/wiki/Hopcroft%E2%80%93Karp_algorithm#Analysis
http://books.google.pt/books/about/Algorithms_in_C_Part_5.html?id=QXHx5rEhI_AC&redir_esc=y

Análise do Algoritmo

Cada etapa consiste numa BFS e numa DFS. O que significa que, uma etapa pode ser executada em tempo linear. Portanto as primeiras $\sqrt{|V|}$ etapas, num grafo com $|V|$ vértices e $|E|$ arcos, demora O $\left(|E| . \sqrt{|V|}\right)$.

Esta análise já foi efectuada e comprovada para o algoritmo de Hopcroft-Karp, no entanto, e para nos certificarmos de que a implementação é fiel a essa análise, apresenta-se de seguida um conjunto de testes com os quais validei o algoritmo:

Foram executados no total 12 testes, que pretendem testar vários cenários e valores de *input* e se resumem sumariamente nos seguintes:

{ nº alunos, nº empregos, tipo de teste }

{ 10000, 10000, SIMPLE_PAIR },
{ 10000, 10000, CROSSED_MULT2 },
{ 10000, 10000, RANDOM_EDGES4 },

{ 10000, 80000, SIMPLE_PAIR },
{ 10000, 80000, CROSSED_MULT2 },
{ 10000, 80000, RANDOM_EDGES4 },

{ 10000, 20000, SIMPLE_PAIR },
{ 10000, 20000, CROSSED_MULT2 },
{ 10000, 20000, RANDOM_EDGES4 },

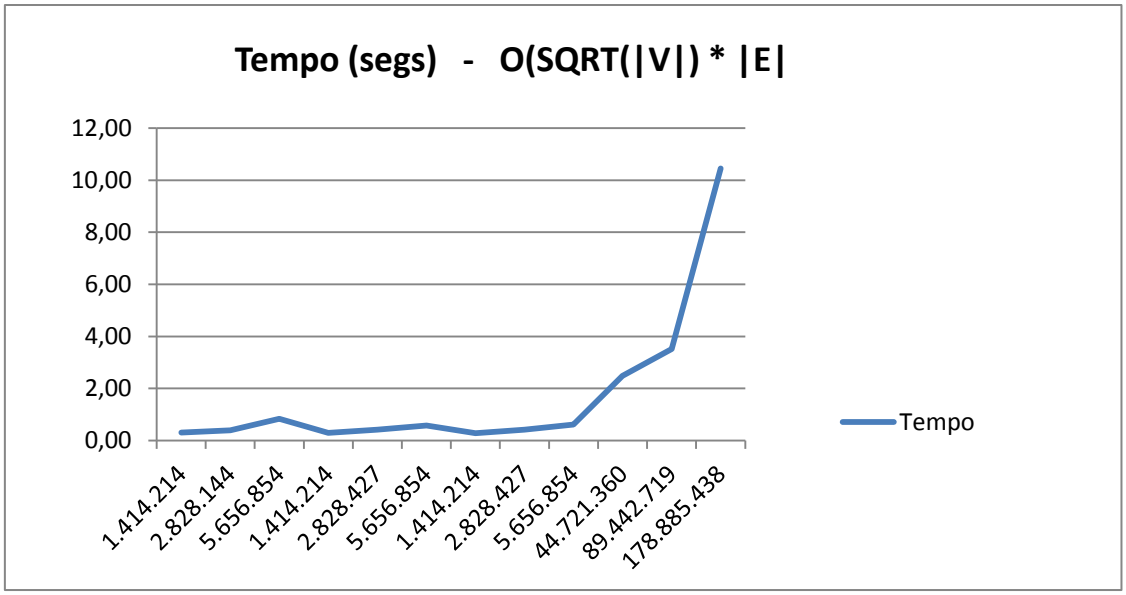
{ 100000, 100000, SIMPLE_PAIR },
{ 100000, 100000, CROSSED_MULT2 },
{ 100000, 100000, RANDOM_EDGES4 }

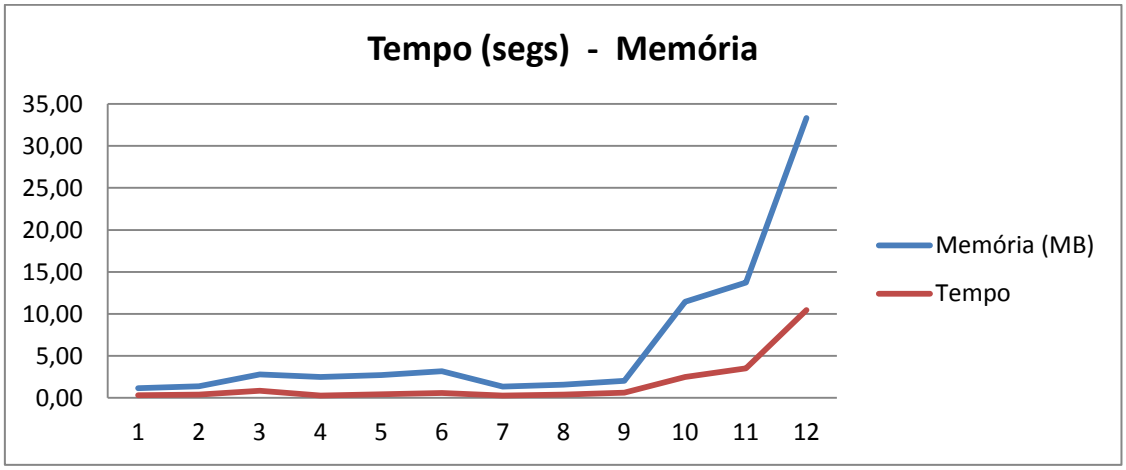
Por motivos de optimização de espaço, os resultados detalhados, encontram-se em anexo, assim como, a descrição do significado de cada tipo de teste.

Dos testes realizados, foi extraída a seguinte informação:

T	Memória (MB)	Tempo (s)	# Estudantes	# Empregos	Tipo Teste	# V	# E
1	1,14	0,31	10000	10000	SIMPLE_PAIR	20000	10000
2	1,37	0,39	10000	10000	CROSSED_MULT2	20000	19998
3	2,78	0,84	10000	10000	RANDOM_EDGES4	20000	40000
4	2,48	0,29	10000	80000	SIMPLE_PAIR	20000	10000
5	2,71	0,42	10000	80000	CROSSED_MULT2	20000	20000
6	3,17	0,58	10000	80000	RANDOM_EDGES4	20000	40000
7	1,34	0,28	10000	20000	SIMPLE_PAIR	20000	10000
8	1,56	0,41	10000	20000	CROSSED_MULT2	20000	20000
9	2,04	0,61	10000	20000	RANDOM_EDGES4	20000	40000
10	11,44	2,48	100000	100000	SIMPLE_PAIR	200000	100000
11	13,73	3,51	100000	100000	CROSSED_MULT2	200000	200000
12	33,32	10,45	100000	100000	RANDOM_EDGES4	200000	400000

Graficamente, podemos observar:





Nos testes realizados o tempo demorado, foi bastante mais rápido do que o que se encontra especificados em tempos de *big-O*, o que é bom. Isto acontece porque em média (Motwani 1994) para grafos espaços bipartidos, existe alta probabilidade para os caminhos de aumento derivados de nós *non-matched* demorarem O $(\log|V|)$ etapas e portanto, neste caso o tempo total é O $(|\mathrm{E}|\,.\log|V|)$.

Referências:

Motwani, Rajeev (1994), "Average-case analysis of algorithms for matchings and related problems", Journal of the ACM *41 (6): 1329–1356, doi:10.1145/195613.195663.*

Apresentam-se de seguida os respectivos resultados à pilha de testes:

[Teste: K01]

Estudantes: 10000
Empregos: 10000
Descrição: Teste SIMLE_PAIR, consiste num teste onde cada estudante quer apenas o emprego correspondente ao seu número, exemplo: s1-e1; s2-e2. Este teste produz caminhos de aumento de nível 2.

Resultados:

- Total heap usage: 40,007 allocs, 0 frees, 1,200,084 bytes allocated
- Took 0.31 seconds to run.

Teste: K02

Estudantes: 10000
Empregos: 10000
Descrição: Teste CROSSED_MULT2, consiste num teste cruzado de multiplicidade 2, ou seja, cada estudante quer o emprego correspondente ao seu número, ou o emprego do colega do lado, e o colega do lado quer o seu, ou então quer o que corresponde ao seu número, exemplo: s1-e1,e2; s2-e1,e2. Este teste obriga a que todos os estudantes sejam empurrados para a sua segunda escolha, por forma a darem oportunidade ao estudante seguinte. Este teste produz caminhos de aumento de nível 2.

Resultados:

- Total heap usage: 50,007 allocs, 0 frees, 1,440,084 bytes allocated
- Took 0.39 seconds to run.

Teste: K03

Estudantes: 10000
Empregos: 10000
Descrição: Teste RANDOM_EDGES4, consiste num teste onde todos os estudantes escolhem pelo menos quatro empregos aleatórios

Resultados:

- Total heap usage: 132,219 allocs, 0 frees, 2,915,476 bytes allocated
- Took 0.84 seconds to run.

Teste: K04

Estudantes: 10000
Empregos: 80000

Descrição: Teste SIMLE_PAIR (…)

Resultados:

- Total heap usage: 40,007 allocs, 0 frees, 2,600,084 bytes allocated
- Took 0.29 seconds to run.

Teste: K05

Estudantes: 10000
Empregos: 80000
Descrição: Teste CROSSED_MULT2 (…)

Resultados:

- Total heap usage: 50,007 allocs, 0 frees, 2,840,084 bytes allocated
- Took 0.42 seconds to run.

Teste: K06

Estudantes: 10000
Empregos: 80000
Descrição: Teste RANDOM_EDGES4 (…)

Resultados:

- Total heap usage: 70,014 allocs, 0 frees, 3,320,196 bytes allocated
- Took 0.58 seconds to run.

Teste: K07

Estudantes: 10000
Empregos: 20000
Descrição: Teste SIMLE_PAIR (…)

Resultados:

- Total heap usage: 40,007 allocs, 0 frees, 1,400,084 bytes allocated
- Took 0.28 seconds to run.

Teste: K08

Estudantes: 10000
Empregos: 20000
Descrição: Teste CROSSED_MULT2 (…)

Resultados:

- Total heap usage: 50,007 allocs, 0 frees, 1,640,084 bytes allocated
- Took 0.41 seconds to run.

Teste: K09

Estudantes: 10000
Empregos: 20000
Descrição: Teste RANDOM_EDGES4 (…)

Resultados:

- Total heap usage: 71,206 allocs, 0 frees, 2,139,268 bytes allocated
- Took 0.61 seconds to run.

Teste: K10

Estudantes: 100000
Empregos: 100000
Descrição: Teste SIMLE_PAIR (…)

Resultados:

- Total heap usage: 400,007 allocs, 0 frees, 12,000,084 bytes allocated
- Took 2.48 seconds to run.

Teste: K11

Estudantes: 100000
Empregos: 100000
Descrição: Teste CROSSED_MULT2 (…)

Resultados:

- total heap usage: 500,007 allocs, 0 frees, 14,400,084 bytes allocated
- took 3.51 seconds to run.

Teste: K12

Estudantes: 100000
Empregos: 100000
Descrição: Teste RANDOM_EDGES4 (…)

Resultados:

- Total heap usage: 1,683,767 allocs, 0 frees, 34,940,244 bytes allocated
- Took 10.45 seconds to run.