UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
PROGRAMA DE PÓS-GRADUAÇÃO EM COMPUTAÇÃO

Pablo Felipe Leonhart

Emparelhamentos
(Trabalho 4)

1. Tarefa

- Implementar o algoritmo de Hopcroft-Karp que resolve o problema do emparelhamento máximo em grafos bipartidos.
- Documentar a implementação, em particular as estruturas de dados para a representação do problema.
- Conduzir testes, que demonstram que a complexidade do algoritmo é O(sqrt(n)(n+m)). Em particular: complexidades O(sqrt(n)) para o número de fases e O(n+m) para a extração de um conjunto maximal de caminhos aumentantes.
- Decidir experimentalmente: a abordagem por redução para um problema de fluxo é mais eficiente?

2. Solução

Foi implementado o algoritmo Hopcroft-Karp na linguagem Python. Inicialmente é realizada a leitura do arquivo de teste e gerado o grafo bipartido. Esse grafo é representado na forma de dicionário, onde uma chave indica um vértice do lado esquerdo do grafo, e a lista de valores dessa chave indica os vértices adjacentes que estão no lado direito do grafo. A execução do método ocorre enquanto existir um caminho aumentante, e para encontrar isso é necessário executar uma busca em largura para encontrar os vértices livres, e então uma busca em profundidade para encontrar os caminhos.

Na busca em largura é percorrida a lista dos vértices do lado esquerdo, todos que estiverem livres são adicionados à uma camada, e essa inserida em uma lista. Então começa uma iteração sobre essa lista, sempre considerando a última camada (o mais recente). Cada vértice percorrido é marcado como visitado. Se ele for do conjunto da esquerda, cada um de seus adjacentes (do lado direito) que não for visitado e se o vértice e a aresta entre eles não for combinada, o vértice do lado direito é então adicionado à uma nova camada. Se o vértice a ser tratado está no lado direito e não for visitado, cada um de seus adjacentes (do lado esquerdo) que for combinado e cuja aresta também esteja combinada, então o adjacente será adicionado à nova camada.

Se existir ao menos uma camada é porque existem vértices livres, então para cada um desses vértices é executada, de forma recursiva, a busca em profundidade para encontrar um conjunto máximo de caminhos aumentantes. Se encontrado algum caminho, suas arestas combinadas são então marcadas no grafo, considerando que o primeiro e o último vértices do caminho sejam livres. O processo encerra quando não há mais caminhos aumentantes.

3. Ambiente de testes

Os resultados foram obtidos numa Intel Core i7-5500U, com 4 processadores de 2.40 GHz e 8 GB de RAM.

4. Resultados

Conduzir testes, que demonstram que a complexidade do algoritmo é O(sqrt(n)(n+m)):

Para verificar a complexidade do algoritmo implementado foram gerados 2 tipos de casos de teste, o primeiro 'A' onde o número de vértices foi definido em 2^{15}, e o número de arestas variou entre 2^{1} e 2^{18}, e o segundo 'B', onde o número de vértices foi fixado em 2^{20} e o número de arestas variou entre 2^{15} e 2^{20}.

As Figuras 1 e 2 representam os valores obtidos para o número de fases e o número de caminhos aumentantes encontrados respectivamente, para as instâncias do teste A. E as Figuras 3 e 4 representam tais informações para as instâncias do teste B.

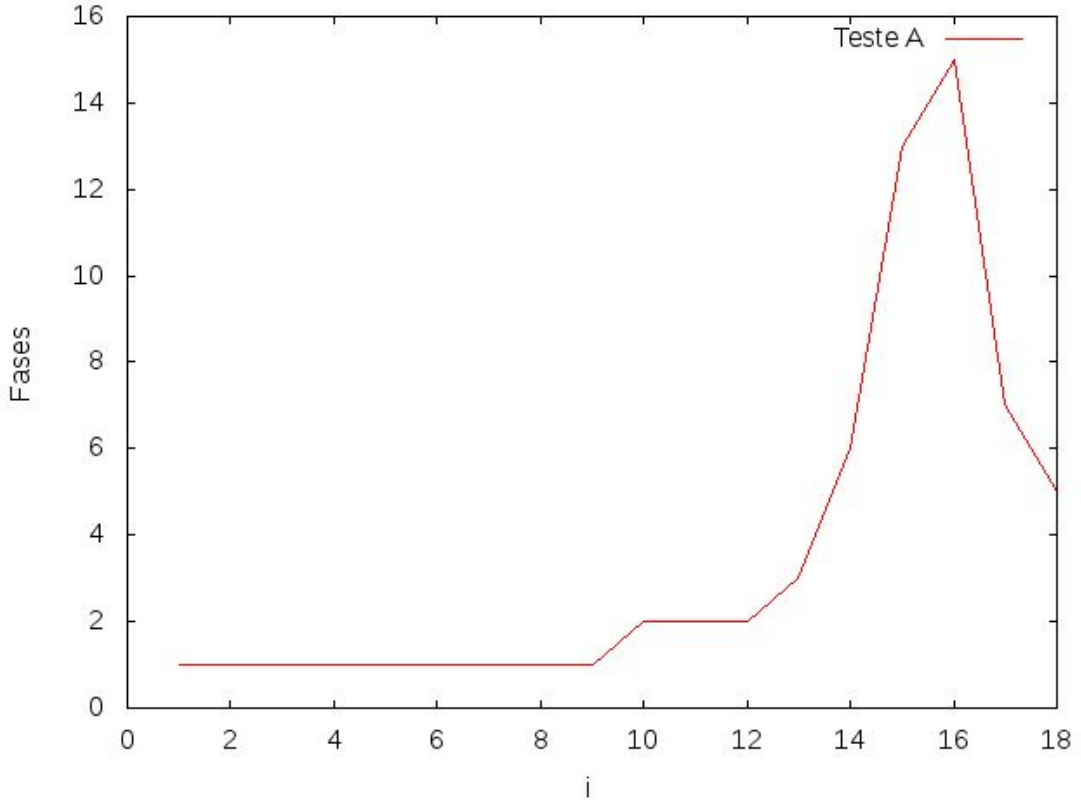

Figura 1: Número de fases necessárias para resolver os casos do teste A

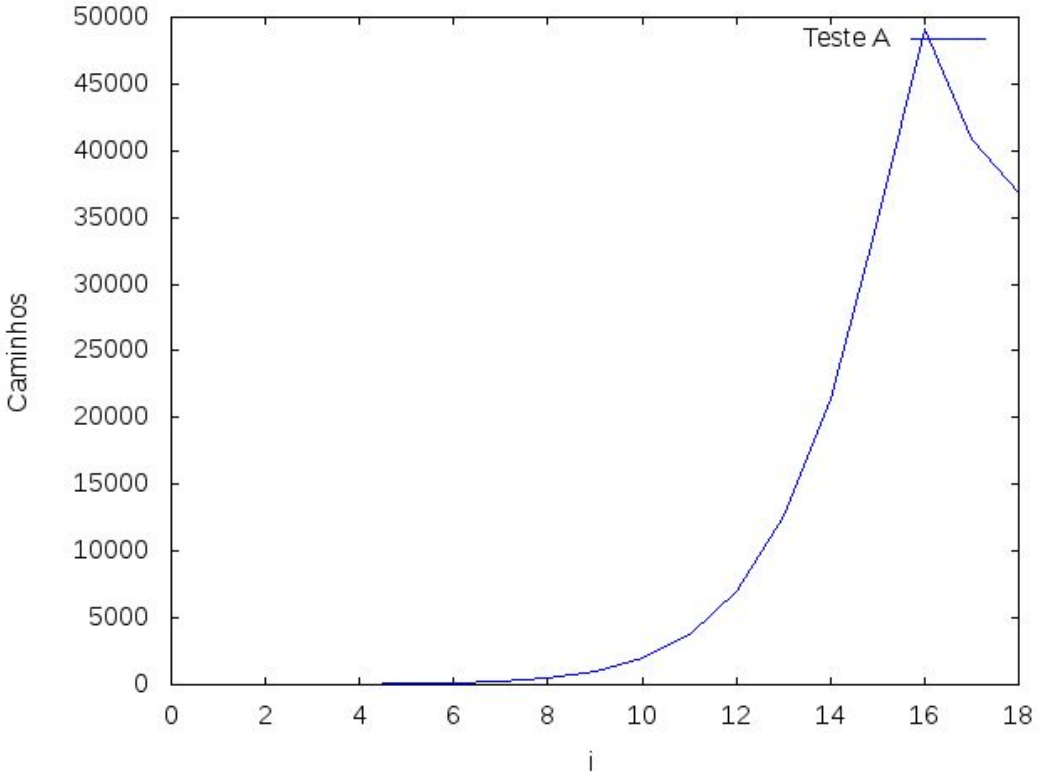

Figura 2: Número de caminhos aumentantes encontrados nos casos do teste A

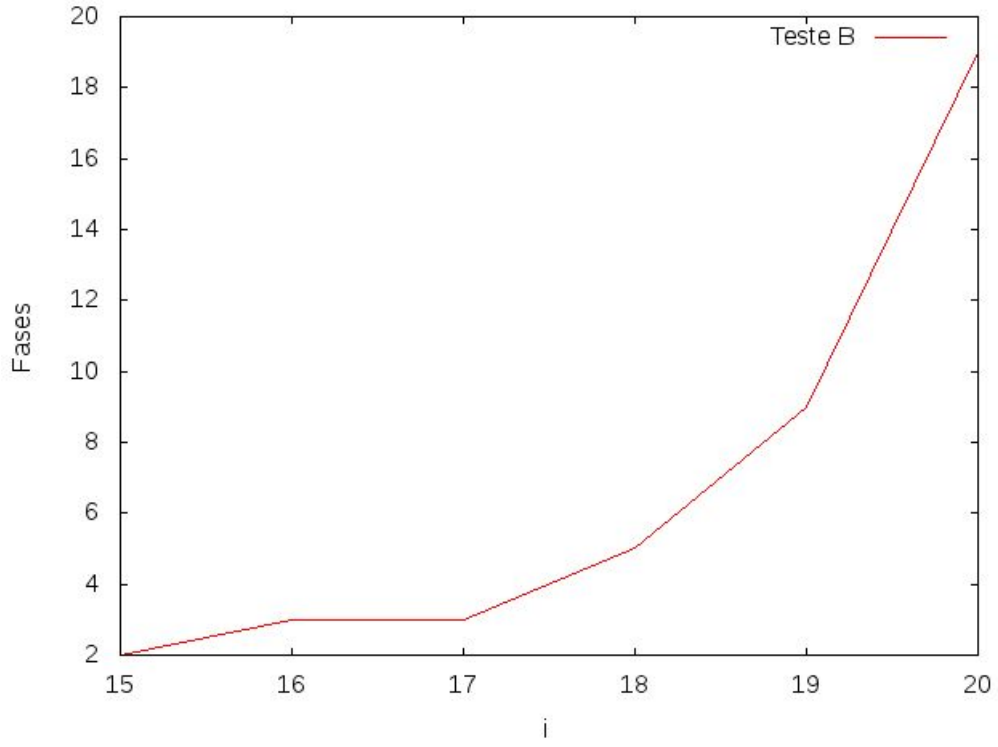

Figura 3: Número de fases necessárias para resolver os casos do teste B

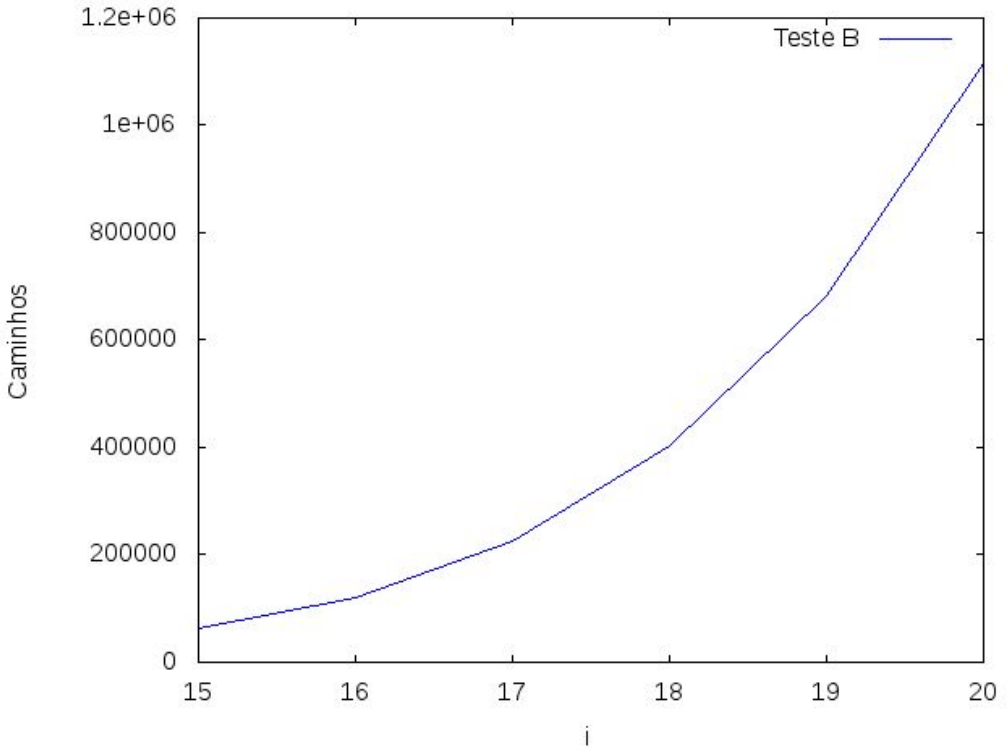

Figura 4: Número de caminhos aumentantes encontrados nos casos do teste B

É possível observar que para as instâncias do teste A, onde o número de vértices foi definido em 2^{15}, há um pico no número de fases e de caminhos aumentantes encontrados quando o número de arestas está entre 2^{14} e 2^{16}. Isso pode ser explicado pois quando há um número relativamente pequeno de arestas, existem menos caminhos aumentantes para serem buscados, e quando há um número maior de arestas do que de vértices, é possível encontrar um número maior de caminhos aumentantes por fase.

Para as instâncias do teste B isso não é observado porque o número máximo de arestas verificadas é igual ao número de vértices. Mas pode ser visto que os gráficos seguem uma curva, assim como no teste A.

Decidir experimentalmente: a abordagem por redução para um problema de fluxo é mais eficiente?

Para fazer a verificação desta abordagem como problema de fluxo foi utilizado o algoritmo de Ford-Fulkerson. Os casos de testes foram gerados com o gerador modificado de forma que foram gerados 2 formatos de arquivo, um para o algoritmo Hopcroft-Karp e outro no formato de entrada para Ford-Fulkerson. No caso do último, foram adicionados 2 vértices (*s* e *t*), de modo que *s* foi conectado a todos os vértices do primeiro conjunto do grafo, e *t* foi ligado a todos os vértices do segundo conjunto. Todos os pesos foram setados com o valor 1.

O número de vértices foi definido em 2^{15}, e o número de arestas variou entre 2^{1} e 2^{18}. Para cada caso de teste foram medidos o tempo de execução em milissegundos e o consumo de memória em kilobytes. Os resultados obtidos estão na Tabela 1.

Tabela 1: Comparação dos resultados obtidos entre o algoritmo Ford-Fulkerson e Hopcroft-Karp

Vértices	Arestas	Ford-Fulkerson		Hopcroft-Karp	
		Tempo (ms)	Memória (kB)	Tempo (ms)	Memória (kB)
2^{15}	2^{1}	75	11192	0,030	7708
2^{15}	2^{2}	128	13568	0,038	7672
2^{15}	2^{3}	243	18848	0,051	7668
2^{15}	2^{4}	480	29144	0,074	7668
2^{15}	2^{5}	945	49472	0,113	7668
2^{15}	2^{6}	1881	90392	0,362	7672
2^{15}	2^{7}	3700	171176	0,374	7672
2^{15}	2^{8}	7283	332488	0,746	7884
2^{15}	2^{9}	17499	649820	2,082	8148
2^{15}	2^{10}	35254	1243968	2,971	8960
2^{15}	2^{11}	64889	2343432	6,091	9800
2^{15}	2^{12}	117058	4259872	12,255	12696
2^{15}	2^{13}	209407	6710576	26,882	16152
2^{15}	2^{14}	375928	7184728	69,454	18972
2^{15}	2^{15}	555116	7270360	207,746	28724
2^{15}	2^{16}	769702	7245680	547,203	32692
2^{15}	2^{17}	899971	7288236	416,462	41144
2^{15}	2^{18}	1048459	7284188	508,910	49652

É possível observar que a abordagem por redução para um problema de fluxo empregando o algoritmo Ford-Fulkerson é de longe mais ineficiente, visto que o fluxo cresce somente uma unidade a cada iteração.

5. Conclusão

O algoritmo de Hopcroft-Karp se mostrou bastante eficiente nos casos de testes propostos. Obteve os resultados em baixo tempo de execução e com pouco consumo de memória, apesar de não serem grandes instâncias de testes. E foi possível provar de forma experimental que a abordagem por redução para um problema de fluxo é menos eficiente que o algoritmo aqui proposto.