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

Pablo Felipe Leonhart

Hollow heaps e algoritmo de Dijkstra
(Trabalho 2)

1. Tarefa

- Implementar um heap com nós vazios e verificar a complexidade das operações experimentalmente.
- Verificar a complexidade do algoritmo de Dijkstra usando um heap com nós vazios experimentalmente.
- Comparar as complexidade empírica do heap binário com a do heap com nós vazios.
- Comparar as complexidades empíricas do algoritmo de Dijkstra com os dois heaps.

2. Solução

Foi implementado o algoritmo hollow heap utilizando a estrutura de dados apresentada no trabalho de Hansen (HANSEN et. al, 2015), seguindo a ideia dos "heaps preguiçosos". Foram implementadas classes para o item, nodo e o heap, e a partir desta última implementadas todas as operações necessárias. Como estrutura de apoio, foi utilizado um vetor, cujas posições representam o valor dos vértices e o conteúdo de cada uma delas contém o ponteiro para o nó correspondente. Essa estrutura é necessária para a operação de decrease-key, onde dado um vértice é possível localizá-lo pela posição nesse vetor.

E o algoritmo de Dijkstra foi implementado utilizando listas de adjacências (pair), e o hollow heap para controle da fila de prioridades.

3. Ambiente de teste

Os resultados foram obtidos numa Intel Core i3-2120, com 4 processadores de 3.30 GHz e 4 GB de RAM.

4. Resultados

Complexidade das operações no hollow heap:

Na Figura 1 pode ser observado a razão entre o número real de operações elementares (links) pelo número empírico destas operações para o hollow heap. Esta medição foi realizada considerando a operação insert, onde foram inseridas $2^{23} - 1$ chaves, e o número de operações por lote de 2^i.

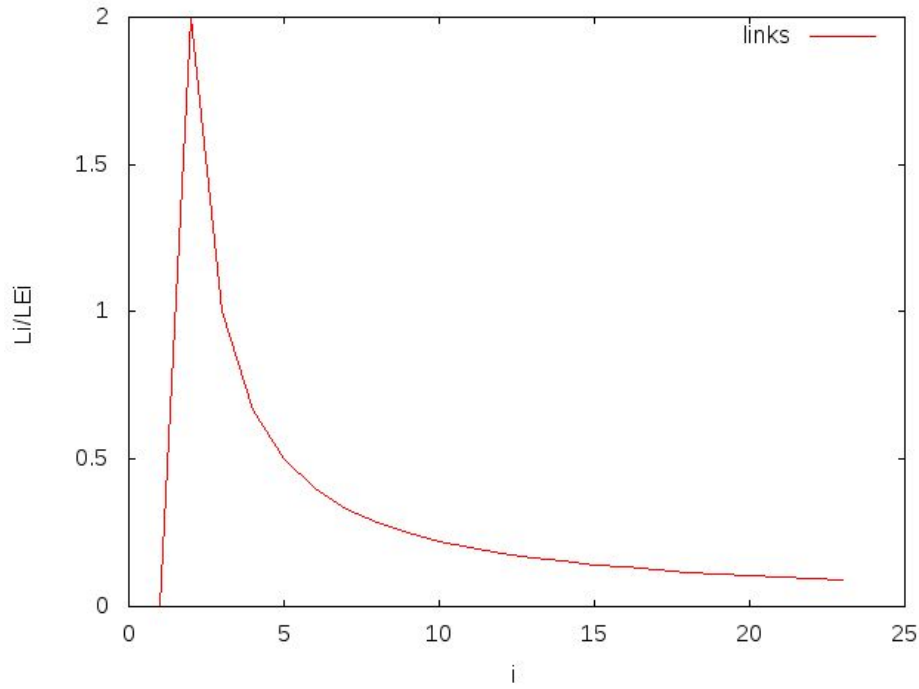

Figura 1: Número real de operações ‘link’ dividido pelo número empírico do pior caso no insert.

Já na Figura 2, temos a razão entre o tempo experimental e o tempo empírico das operações elementares para o hollow heap e o heap binário.

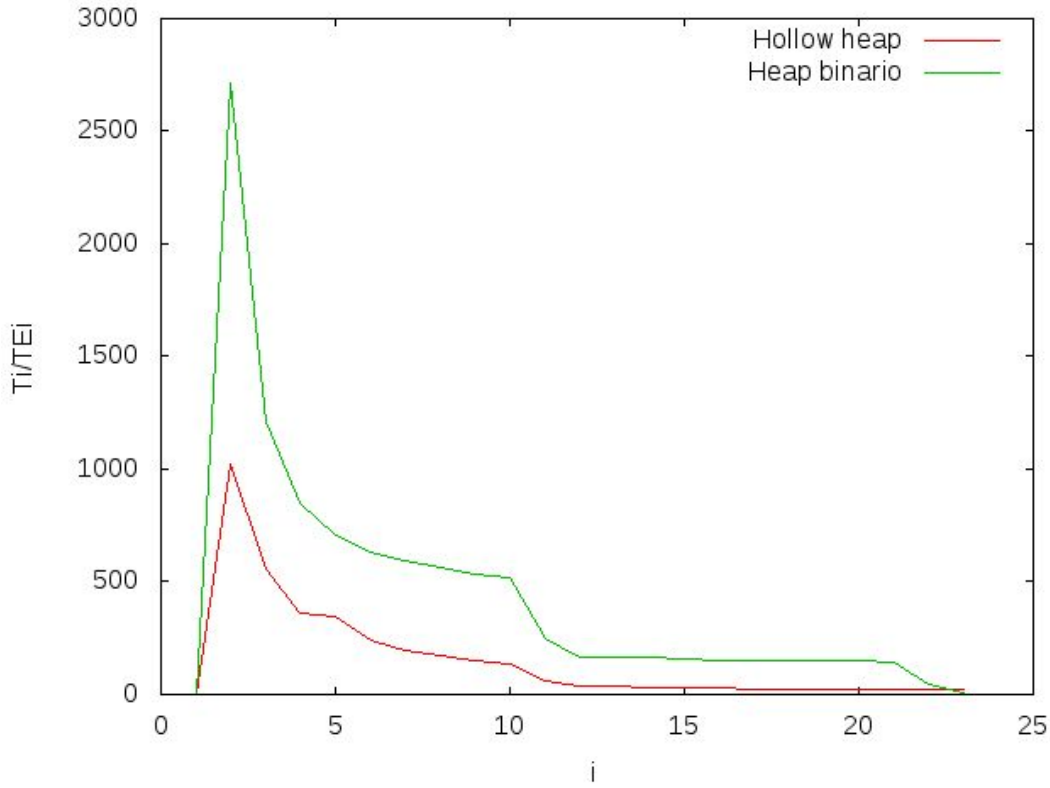

Figura 2: Comparação de desempenho dos heaps para o pior caso no insert.

A Figura 3 representa as medições do número de operações links no heap para o update, onde é mensurada a razão entre os valores coletados e empíricos para i = 1, …, 20.

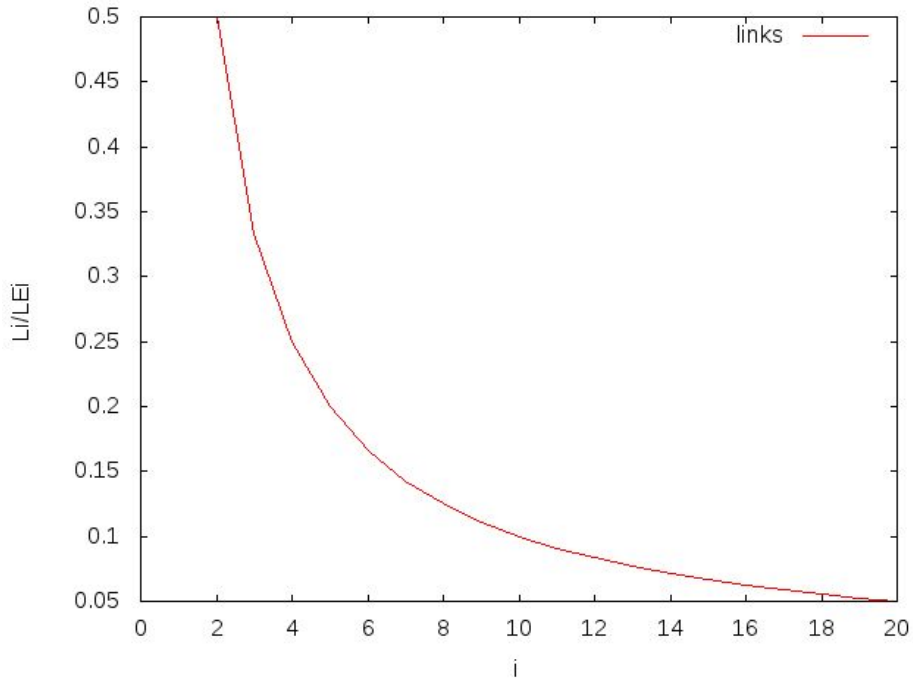

Figura 3: Número real de operações ‘link’ dividido pelo número empírico do pior caso no update.

Na Figura 4 temos a razão entre o tempo experimental e o tempo empírico das operações elementares para o hollow heap e o heap binário para a operação update.

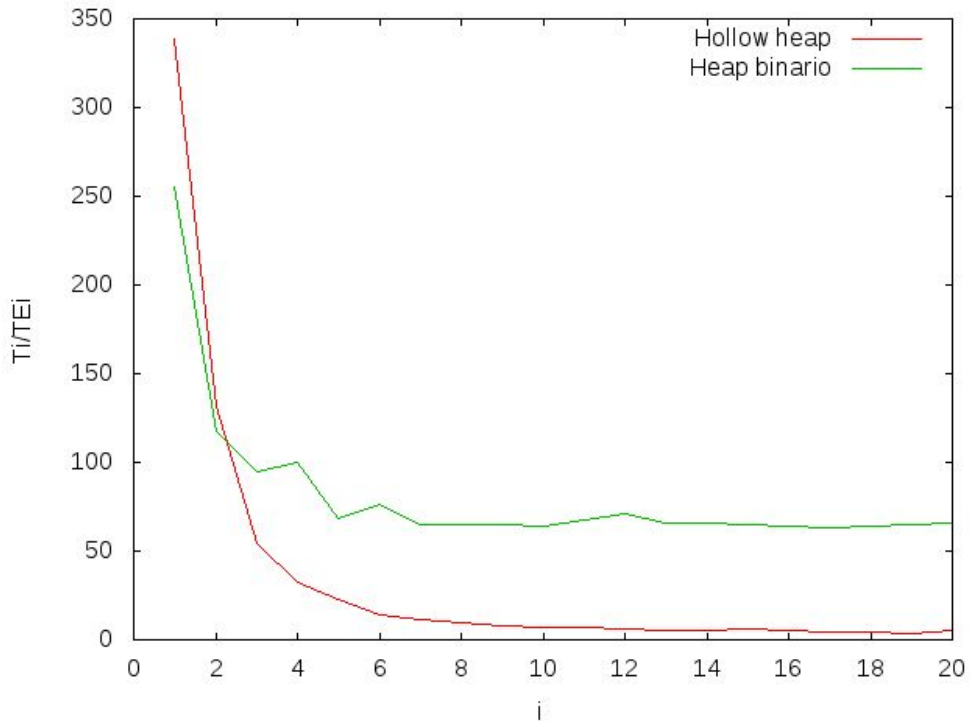

Figura 4: Comparação de desempenho dos heaps para o pior caso no update.

A Figura 5 representa as medições do número de operações links no heap para o delete, onde é mensurada a razão entre os valores coletados e empíricos para i = 1, …, 20.

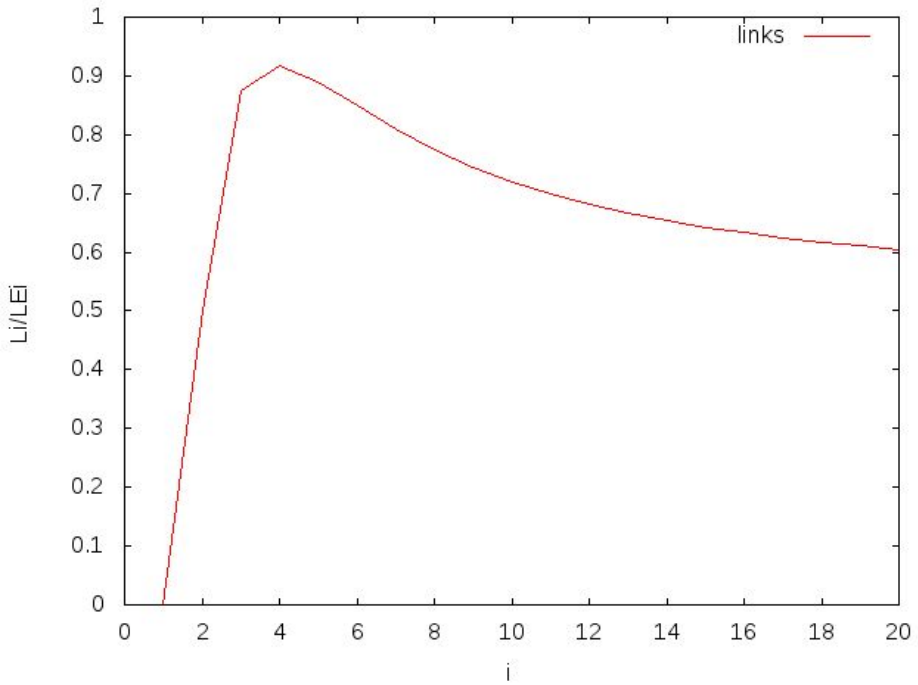

Figura 5: Número real de operações ‘link’ dividido pelo número empírico do pior caso no delete.

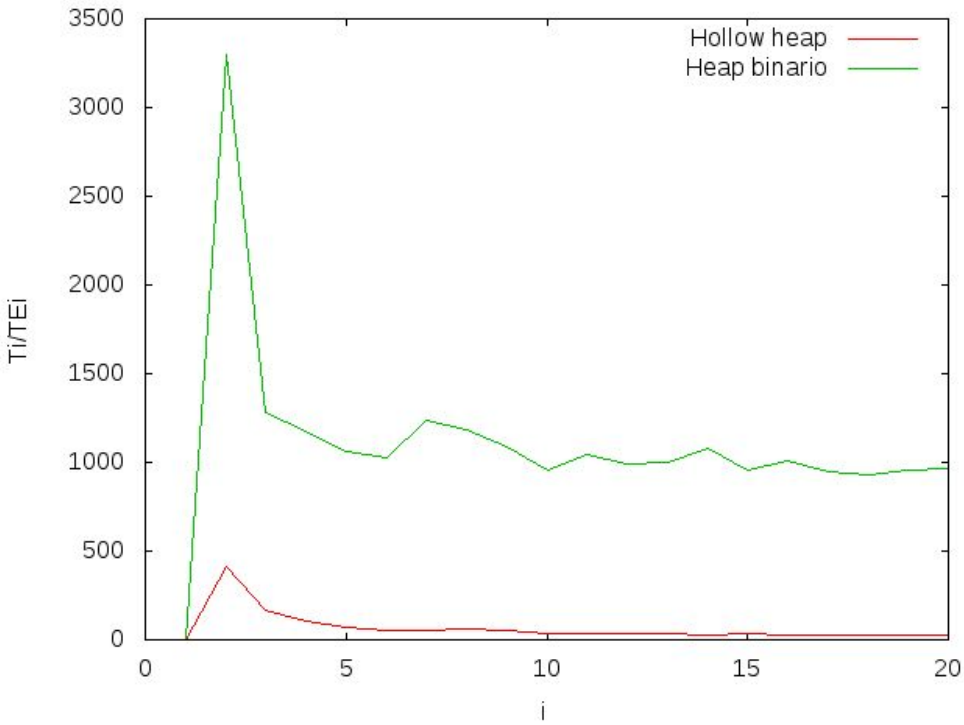

Figura 6: Comparação de desempenho dos heaps para o pior caso no delete.

Na Figura 4 temos a razão entre o tempo experimental e o tempo empírico das operações elementares para o hollow heap e o heap binário para a operação delete.

Complexidade do algoritmo de Dijkstra:

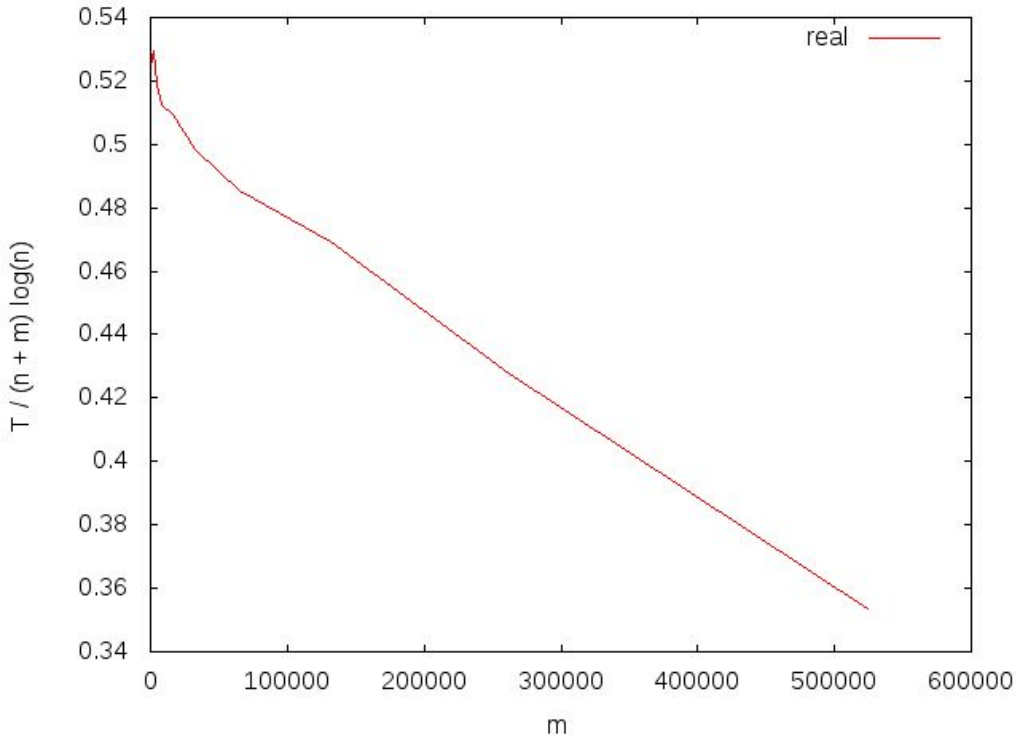

Figura 7: Razão entre as complexidades real e teórica com o número de vértices fixo.

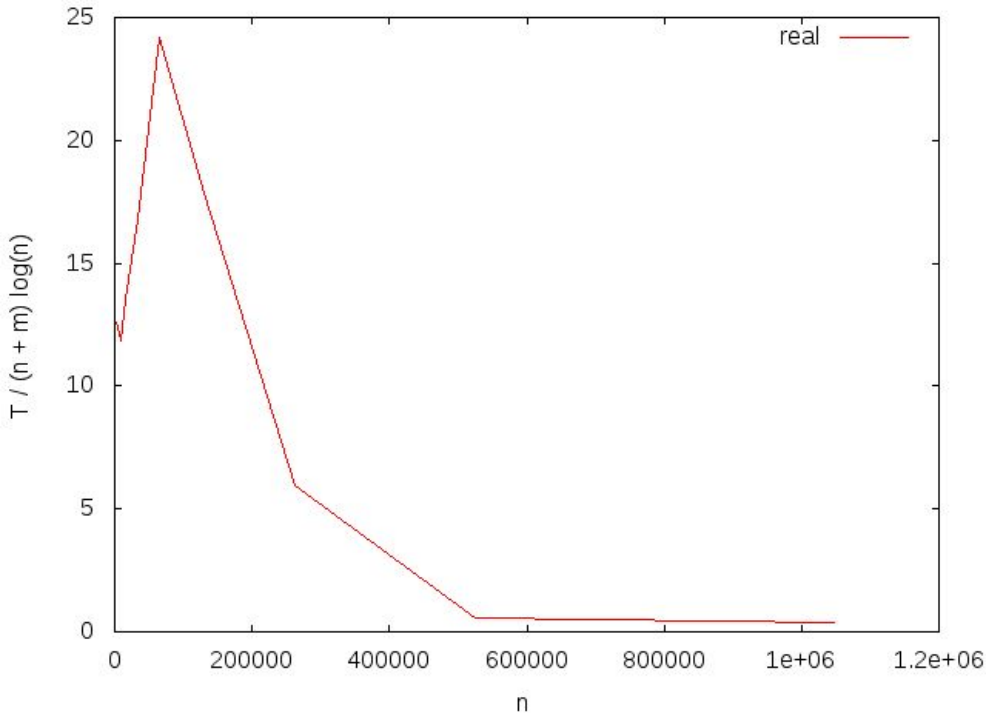

Figura 8: Razão entre as complexidades real e teórica com o número de arestas fixo.

Na Figura 7 pode ser observada a razão entre as complexidades do algoritmo em que o número de vértices foi fixado em 2^{20} e o número de arestas variou de 2^{0} a 2^{19}. E na Figura 8, temos a razão entre as complexidades onde o número de arestas foi fixado em 2^{20} e o número de vértices variou de 2^{11} a 2^{20}.

Comportamento em instâncias grandes:

Foram realizados testes rodando o algoritmo Dijkstra com o uso do hollow heap implementado, por 30 vezes, com um vértice inicial aleatório, nos grafos de NY e USA. Nas Figuras 9 e 10 podem ser observados o tempo de execução em milissegundos e o consumo de memória em kilobytes respectivamente, para cada execução do algoritmo. A média do tempo ficou em 209 ms, e a média do consumo em 40120 kB.

Nas Figuras 11 e 12 são relatados o tempo de execução em segundos e o consumo de memória em gigabytes respectivamente, para cada execução do algoritmo. A média do tempo ficou em 26,3 s, e a média do consumo em 3,22 GB.

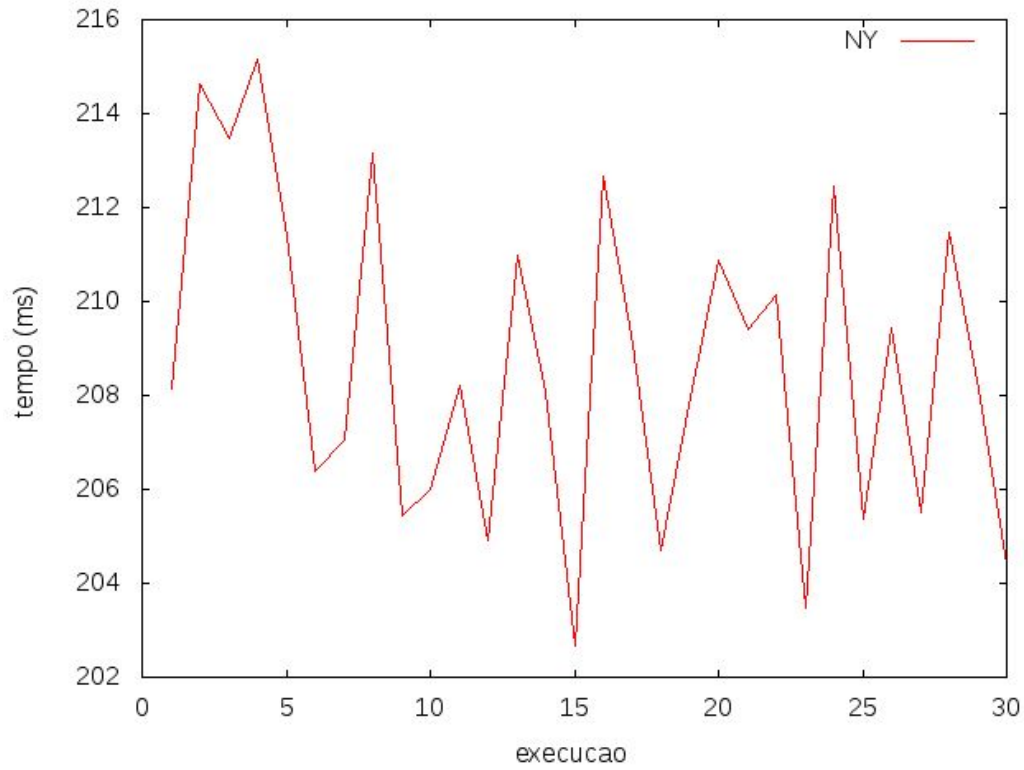

Figura 9: Tempo de execução em milissegundos para percorrer todo o grafo NY a partir de um vértice aleatório.

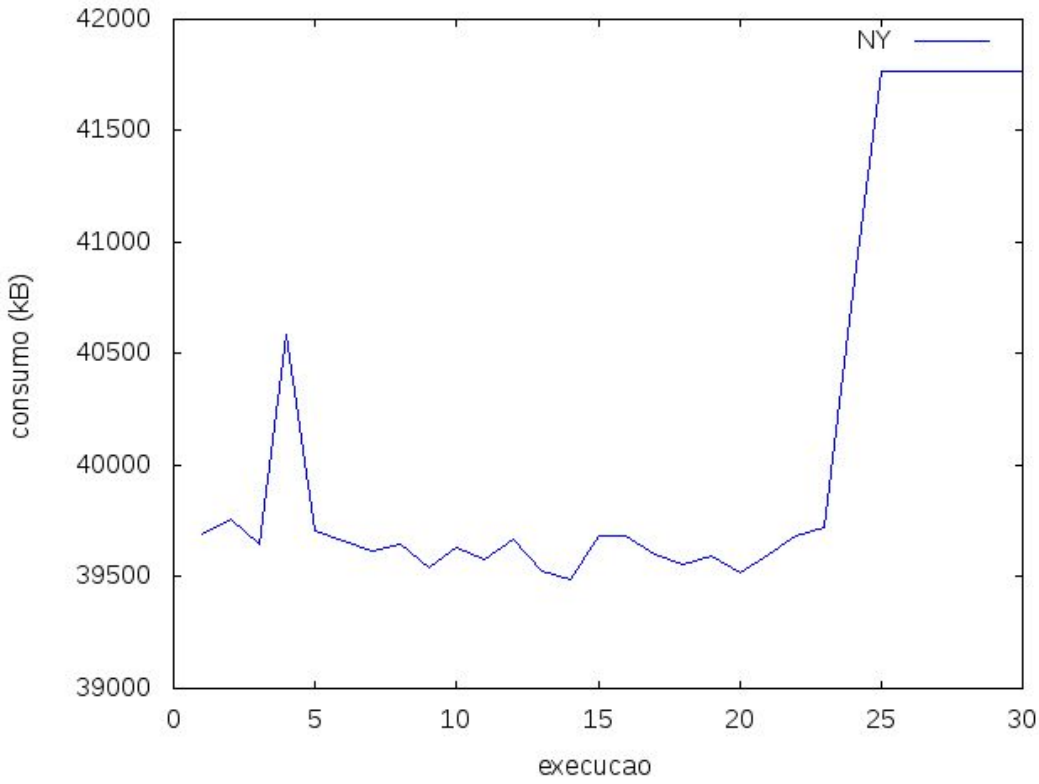

Figura 10: Consumo de memória em kilobytes para as execuções no grafo de NY.

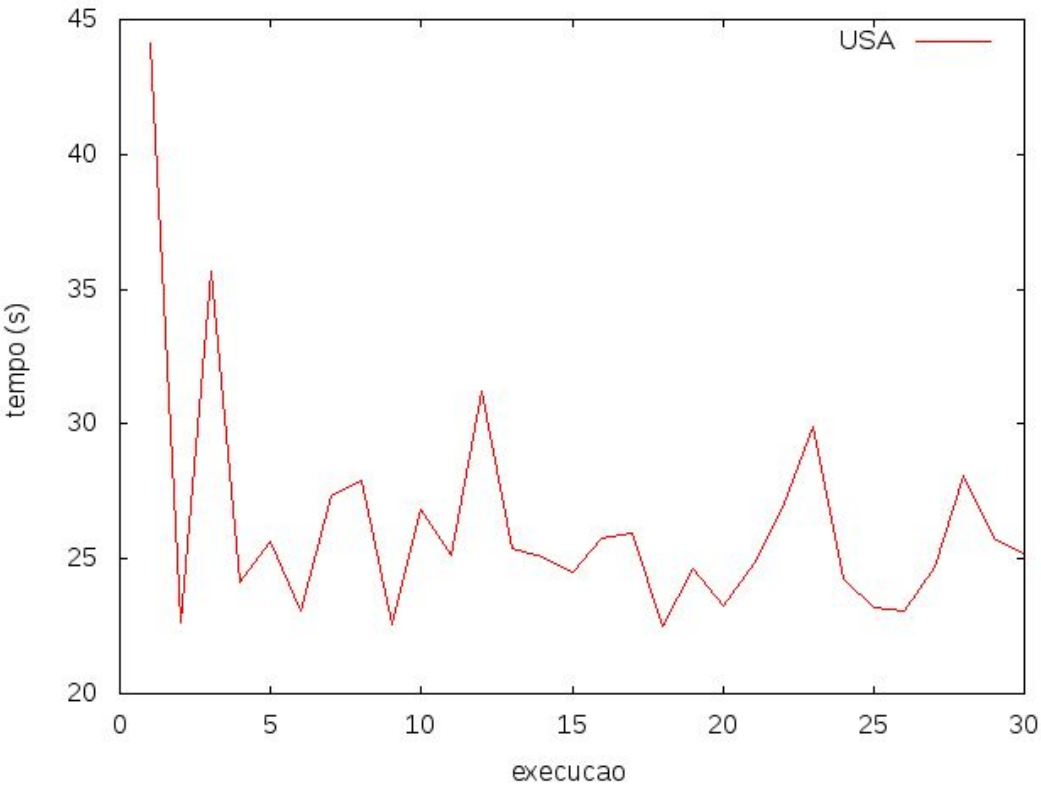

Figura 11: Tempo de execução em segundos para percorrer todo o grafo USA a partir de um vértice aleatório.

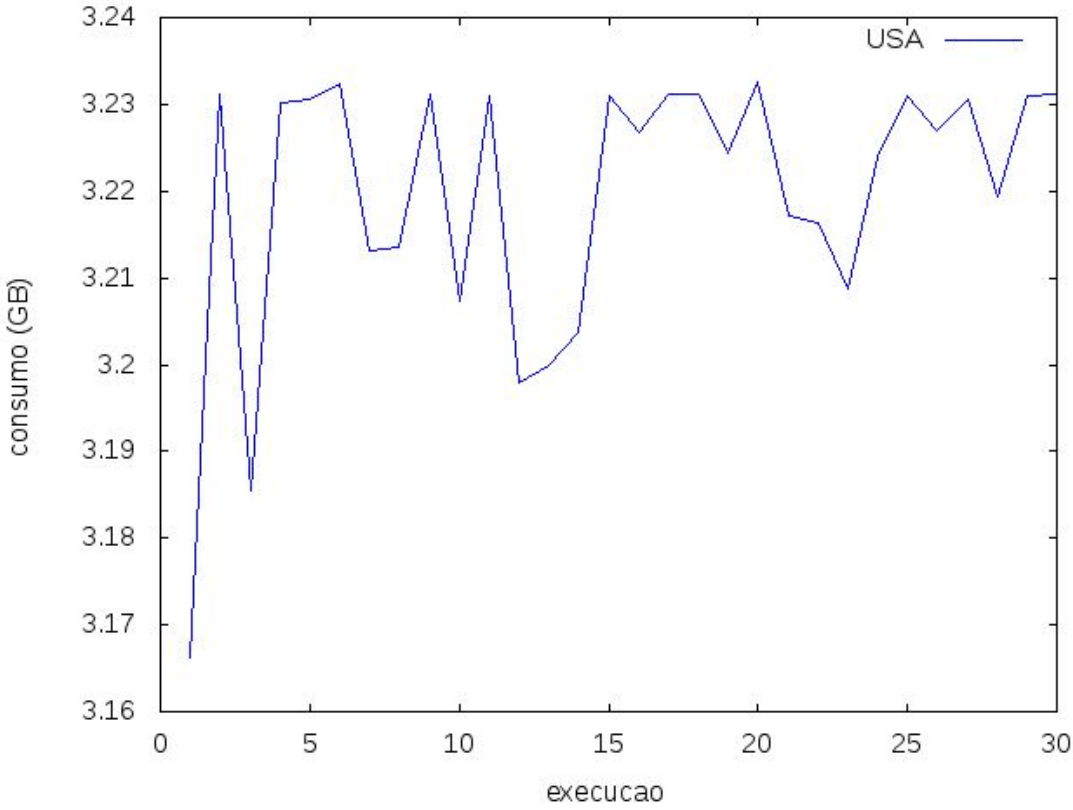

Figura 12: Consumo de memória em gigabytes para as execuções no grafo de USA.

5. Conclusão

Com os testes realizados podemos observar que o hollow heap implementado apresenta uma complexidade linear em suas operações elementares (links). Comparado ao heap binário, apresenta um bom desempenho em tempo de execução. Mostrando que à medida que o número de chaves cresce, há um maior ganho em relação ao heap binário, isso se deve ao fato de que no hollow heap não há operações de swap, e as deleções de nodos ocorrem ocasionalmente no delete-min.

Para os testes realizados com instâncias maiores o algoritmo se mostrou eficiente. Para o caso de testes de NY, o tempo de execução e consumo de memória foram relativamente baixos. E para o caso de testes dos USA, as métricas também foram aceitáveis, levando em conta o poder computacional da máquina.

Referências

HANSEN, T.D., KAPLAN, H., TARJAN, R.E. *Hollow Heaps*. ICALP, 2015. Disponível em: <https://arxiv.org/abs/1510.06535>