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

THIAGO BELL

Laboratório 1 - Binary Heaps e Dijkstra

Porto Alegre, 2017

1.1 Tarefa

Implementar uma *Hollow Heap* e usá-la ao implementar o algoritmo de Dijkstra. Verificar desempenho da *Hollow Heap* e do algoritmo de Dijsktra e compará-los com as previsões teóricas. Os testes presentes no plano de teste sugerido foram utilizados neste trabalho.

1.2 Implementação

Os algoritmos foram implementados em C++. A *Hollow Heap* utiliza memória alocada dinâmicamente para armazenar os seus nodos. Os grafos são representados por lista de adjacências. Um vetor armazena os vértices, e outro, as arestas.

1.3 Ambiente de Teste

Os experimentos foram realizados usando um processador Intel i7 2600k acompanhado de 8 GiB de RAM. O sistema operacional utilizado foi Ubuntu Linux 16.10.

1.4 Análise de Complexidade da *Heap*

A complexidade teórica das operações da *Hollow Heap* foram comparadas com os resultados de experimentos. Conclui-se que a implementação respeita as previsões teóricas.

1.4.1 Avaliação das Operações de Inserção

Fixando-se o valor $n = 2^{27} - 1$, adicionou-se n elementos na *Hollow Heap* com chaves no intervalo de $[n, 1]$. Para cada bloco com o intervalo de inserções $[2^{i-1}, 2^i - 1]$ para $i \in [0, 26]$, foi medido o número de nodos criados C_i, o número de melds M_i e tempo de execução T_i. Para cada valor inserido na *Hollow Heap*, um meld é executado. Um meld é a união de duas *heaps*. Onde um nodo único também é considerado uma heap. Com experimentos, verificou-se que o número de chamadas as rotinas de inserção e *meld* foram chamadas uam vez para cada elemento inserido como era esperado. A inserção de um elemento na heap tem custo $O(1)$ (*meld* do elemento com a *heap*). Logo, inserir n nodos terá custo $O(n)$. Comparando o número de inserções com o número de *melds* percebe-se que eles são os mesmos indicando que a implementaçao respeita a definição do algoritmo. Comparando-se o tempo de execuçao com o nüumero de operações esperadas pela complexidade, vê-se uma relação constante como pode ser vista na figura 1.1

Considerando as duas comparações, conclui-se que a implementação da inserção na *Hollow Heap* respeita a complexidade do algoritmo.

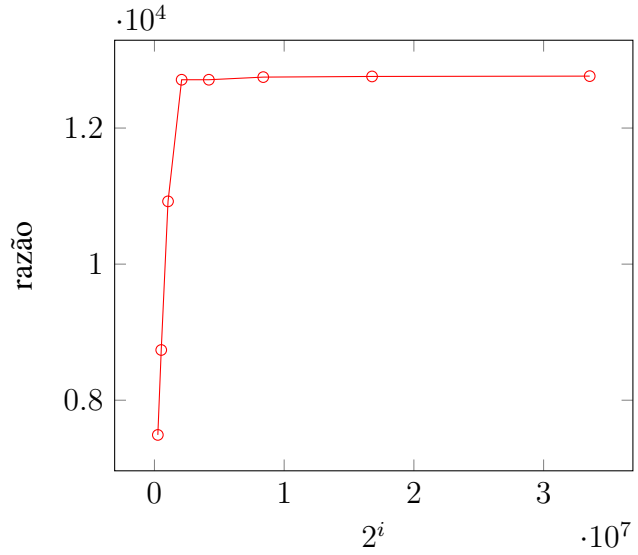

Figura 1.1: Razão entre tempo de execução e o custo esperado.

1.4.2 Avaliação das Operações de Decrescimento de Chaves

Considerando-se uma variável i, $2^i - 1$ chaves são inseridas com valor de $2^i + 1$. Em seguida, 2^i chaves com valor $2^i + 2$ são adicionadas. Medindo-se o tempo, as chaves com valor, $2^i + 2$ são atualizadas para valores decrescentes no intervalo $[2^i, 1]$. As operações de atualização começam por 2^i e a cada operação seguinte esse valor é decrescido por 1. A complexidade para a operação de *decrease key* é $O(1)$. Logo para atualizar as 2^i chaves a complexidade é de $O(2^i)$. Ao comparar essa complexidade com o tempo de execuçao, nota-se a convergência da razão desses valores como mostra a figura 1.2. Assim, pode-se concluir que o algoritmo escala como previsto.

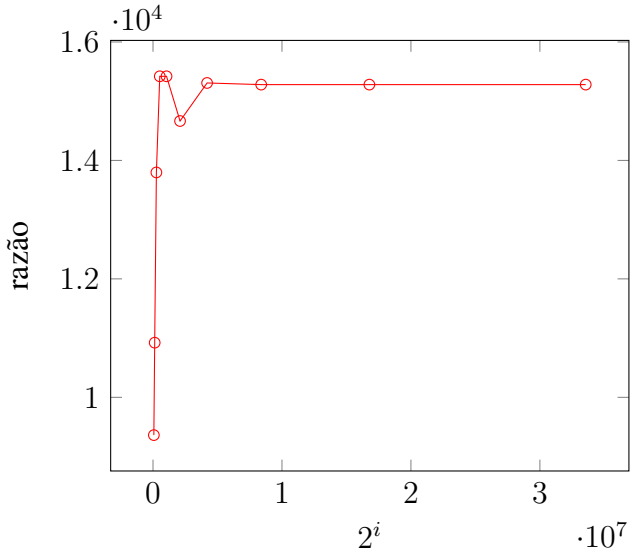

Figura 1.2: Mostra a razão entre o tempo de execução e o custo teórico esperado

1.4.3 Avaliação das Operações de Remoção Mínima

Para um valor i, $n = 2^i - 1$ chaves com valor aleatórios são adicionados. Depois, $m = 2^{i-1}$ chaves são removidas. O tempo de execução e o número de swaps foram

medidos para cada i. A complexidade da operação de *delete min* é de $O(logn)$ onde n é o número de nós na heap. Para remover m nodos, a complexidade é $O(logn * m)$ Na figura 1.3 é comparado esse custo com o tempo de execução. A convergência a um valor indica que o tempo de execução cresce conforme a complexidade do algoritmo.

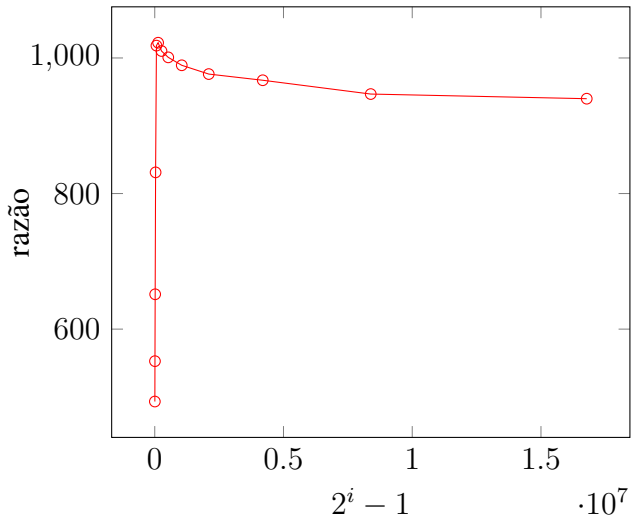

Figura 1.3: Mostra a razão entre o tempo de execução e o custo teórico esperado

1.5 Análise de Complexidade do Algoritmo de Dijkstra

1.5.1 Variando o número de Arestas

Fixando-se o número de vértices em 2^{15}, variou-se o número de arestas entre 2^{18} e 2^{28}. Ao comparar o tempo de execução com o custo teórico de $E = (n+m) * log(n)$, obteve-se uma convergência a um valor constante como pode ser visto na figura 1.4. Os números de *inserts* e *deletes* foram menores ou iguais ao de vértices e o de *updates* menor ou igual ao de arestas. Uma regressão linear na figura 1.5 sobre o os dados obtidos mostrou que o tempo de execução cresce exponencialmente de acordo com a soma do número de arestas.

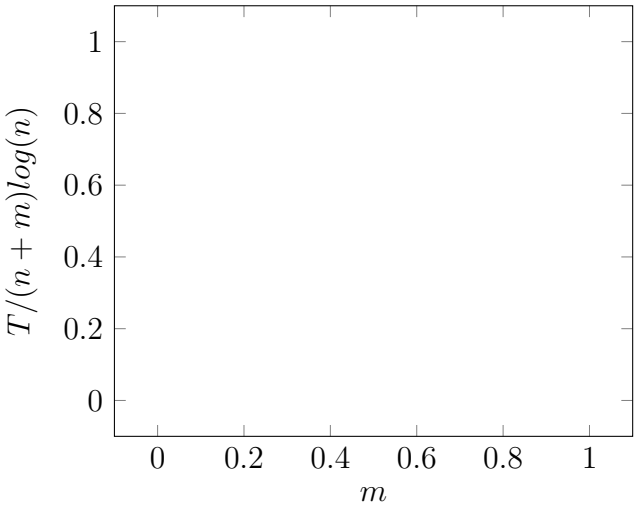

Figura 1.4: Mostra a razão entre o tempo de execução e o custo teórico esperado

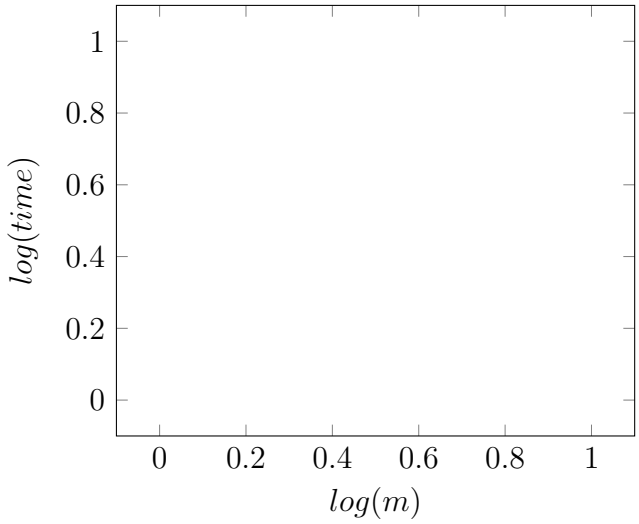

Figura 1.5: Regressão linear. O número de vértices foi desconsiderado pois é constante.

1.5.2 Variando o número de Vértices

Fixou-se o número de arestas em 2^{20} e variou-se o número de vértices entre 2^{11} e 2^{18}. O custo teórico é o mesmo do caso anterior, $E = (n + m) * log(n)$. Encontrou-se um problema onde o grafo se tornavo excessivamente esparso e o algoritmo de dijkstra encerrava sem percorrer parte significante do grafo. Poucas amostras foram obtidas como fica evidente na figura 1.6.

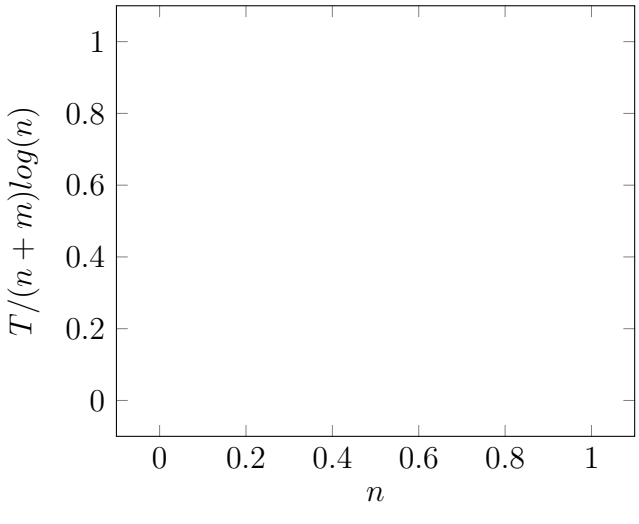

Figura 1.6: Mostra a razão entre o tempo de execução e o custo teórico esperado

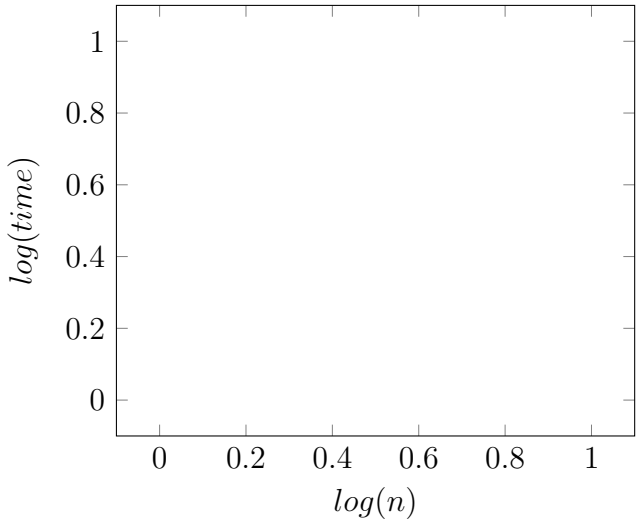

Figura 1.7: Regressão linear. O número de arestas foi desconsiderado pois é constante.

1.6 Verificando a implementação de Dijkstra com grandes instâncias

Testou-se a implementação do algoritmo com duas instâncias conforme recomendado no plano de teste. Para o caso de teste consistindo da rede completa dos Estados Unidos provida na definição do trabalho, obteve-se um tempo médio de execução de 169 segundos e consumo de memómia máximo de 3.64 GiB. Para a rede de Nova York, foram 1.83 segundos e 43.7 MB de memória.

1.7 Testando Grau Ótimo da Heap

Testou-se a implementação da heap variando-se o grau da mesma e usando as redes de Nova York e dos Estados Unidos assim como outras geradas aleatóriamente com uma versão modificada do algoritmo gerador oferecido. Um comportamento interessante foi obtido. Para as redes reais, os resultados foram melhores para graus 4 e 8. Entretando, para as artificiais a tendência foi de quanto maior o grau da heap, melhor o desempenho ao executar o algoritmo de Dijkstra. A figura 1.8 mostra os resultados. Os tempos de execução de cada grafo foram normalizados em relação ao tempo médio obtido para aquela rede. Acredita-se que esse comportamento possa ser explicado pela aleatoriedade na inserção de arestas o que resulta numa topologia diferente daquelas encontradas em redes reais.

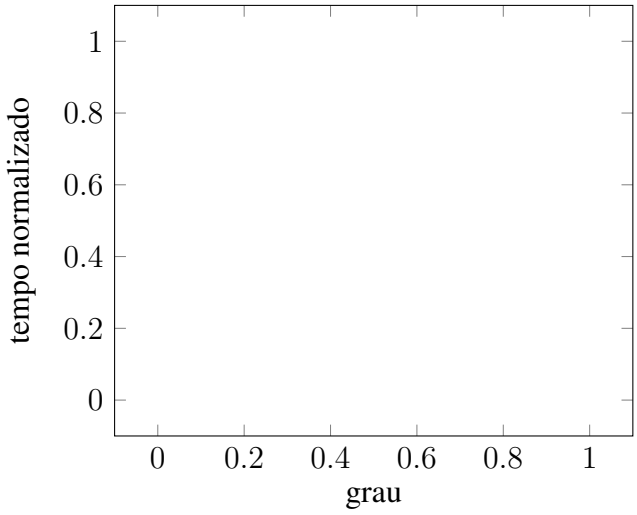

Figura 1.8: Tempo normalizado de execução do algoritmo de Dijkstra

1.8 Conclusão

Excluindo-se o caso do tempo de execução das atualizações quando comparado com a expectativa teórica, as implementações dos algoritmos respeitaram a complexidade teórica dos mesmos. Procurou-se, também, o grau da heap com o valor mais adequado ao algoritmo de Dijkstra. Chegou-se a conclusão que a rede com a qual se testa pode influenciar os resultados desse teste.