Fibonacci Heap

Mauro Nunes Weber [1]

[1]Instituto de Computação – Universidade Federal Fluminense (UFF)
Caixa Postal 24210-346 – São Domingos – Niterói – RJ – Brazil

Abstract. *This article presents the definition and properties of the so-called Fibonacci Heap Tree. Its main operations and their respective complexities are also described. In the end, we show that the main application of this structure is in improving the complexity of the Dijkstra algorithm, where the Fibonacci Heap-based implementation will run at the fastest speed.*

Resumo. *Este artigo apresenta a definição e propriedades da árvore alcunhada de Heap de Fibonacci. Também são descritos suas principais operações e suas respectivas complexidades. No fim, mostramos que a principal aplicação desta estrutura é no aprimoramento da complexidade do algoritmo de Dijkstra, onde a implementação baseada em Heap de Fibonacci será executada na velocidade mais rápida.*

1. Introdução

Heap de Fibonacci é uma estrutura de dados avançada, introduzida por Fredman e Tarjan (1987). É amplamente utilizada para implementar filas de prioridade. A fila de prioridade é uma das estruturas de dados mais usadas para implementar vários algoritmos como algoritmo de caminho mais curto de fonte única, problema de vértice etc. O algoritmo sequencial de heap de Fibonacci é conhecido por ser o algoritmo mais eficiente para a implementação de filas de prioridade.

2. Estrutura de Dados Heap

Uma Estrura de Dados é chamada de Heap quando ela é uma Árvore Binária que satisfaz as propriedades de Heap. Isto é, dado um nó desta Árvore,

- ele é sempre maior que seu nó filho e a raiz da Árvore é o maior nó existente. Esta propriedade é chamada de Max-Heap (Figura 1);
- ele é sempre menor que seu nó filho e a raiz da Árvore é o menor nó existente. Esta propriedade é chamada de Min-Heap (Figura 2).

Esta estrutura também é chamada de Binary Heap.

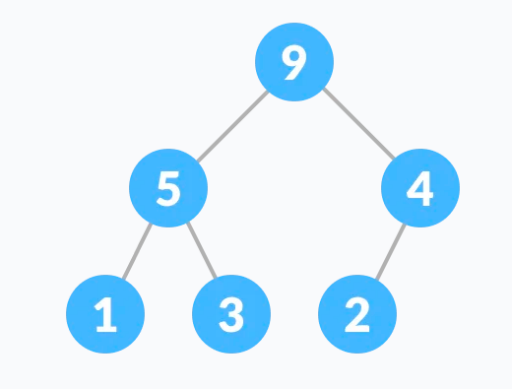

Figura 1. Max Heap.

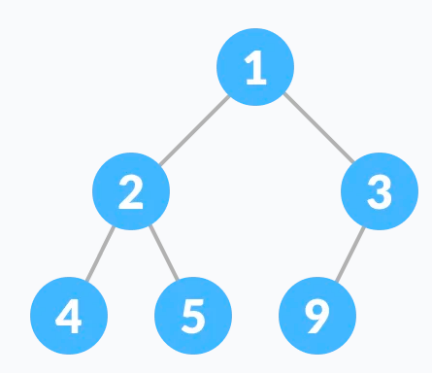

Figura 2. Min Heap.

3. Estrutura Heap de Fibonacci

O Heap de Fibonacci é chamado de Heap de Fibonacci porque as árvores são construídas de forma que uma árvore de ordem n tenha pelo menos F_{n+2} nós, onde F_{n+2} é o $n+2$-ésimo número de Fibonacci.

Um Heap de Fibonacci é uma estrutura de dados que consiste em uma coleção de árvores que possuem a propriedade Min-Heap; Um ponteiro é sempre mantido no nó do menor elemento; Nós podem ser marcados; E as árvores dentro do Heap de Fibonacci possuem suas raízes conectadas, mas não ordenadas.

As raízes das árvores são armazenadas em uma lista circular duplamente ligada. Veja um exemplo de um Heap de Fibonacci abaixo (Figura 3).

3.1. Operações

Nesta sessão será introduzido as principais operações realizadas na estrutura Heap de Fibonacci.

3.1.1. Inserir

A operação de inserir um elemento no Heap de Fibonacci possui custo $O(1)$. Nesta operação tem-se apenas dois passos:

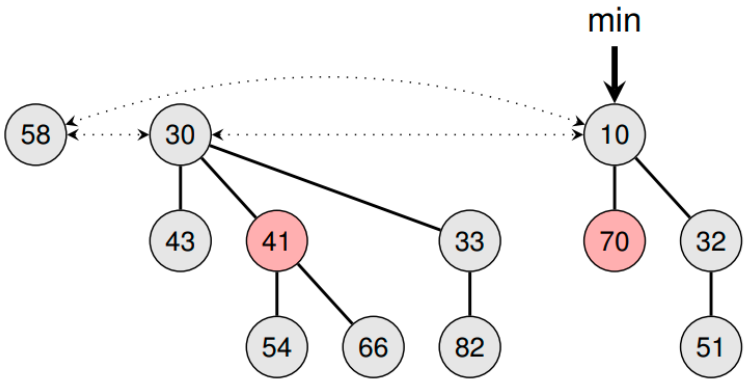

Figura 3. Heap de Fibonacci.

Passo-1: Criar uma Árvore com um elemento. (Figura 4)

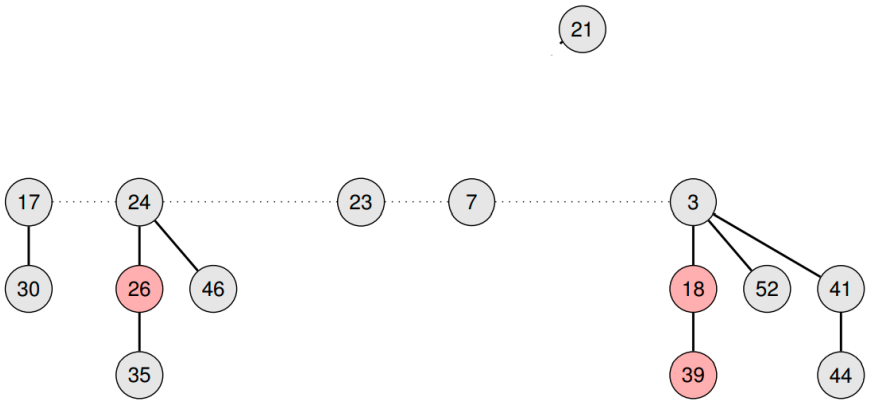

Figura 4. Passo-1 Inserir.

Passo-2: Conectá-lo as raízes do Heap de Fibonacci. Se for necessário atualiza-se o ponteiro para o menor elemento. (Figura 5)

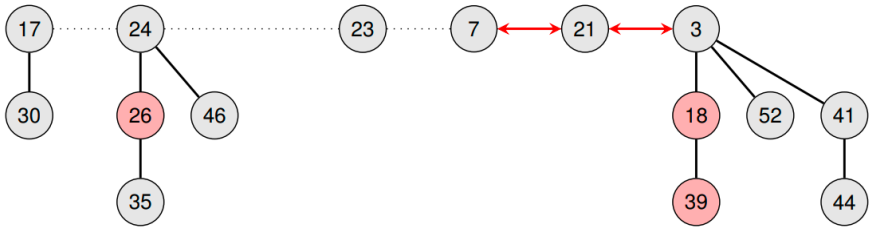

Figura 5. Passo-2 Inserir.

3.1.2. Extrair Mínimo

A operação de Extrair Mínimo possui custo $O(tree(H)) + O(d(n))$, onde $tree(H)$ é a quantidade de árvores do Heap de Fibonacci e $d(n)$ é o grau máximo de uma árvore em

um Heap de Fibonacci de tamanho n. A seguir será apresentado a sequência de passos desta operação:

Passo-1: Deletar o elemento mínimo. (Figura 6 e Figura 7)

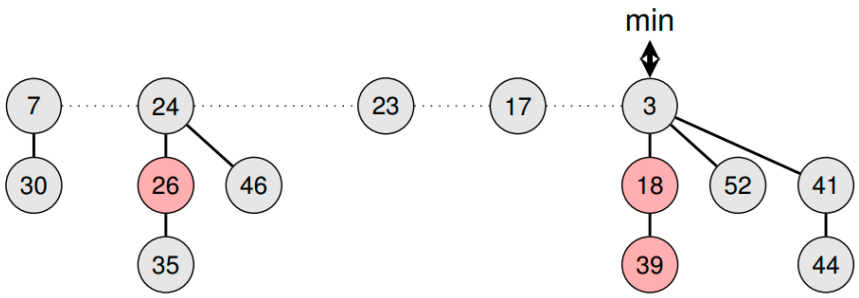

Figura 6. Passo-1 Extrair Mínimo.

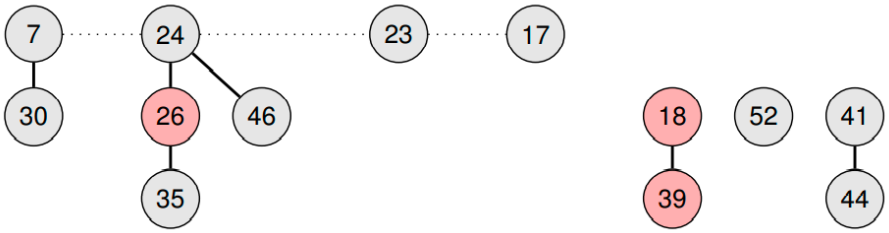

Figura 7. Passo-1 Extrair Mínimo.

Passo-2: Inserir os elementos filhos do nó deletado a lista de raízes. Se um nó estiver marcado, desmarca-se o nó. (Figura 8)

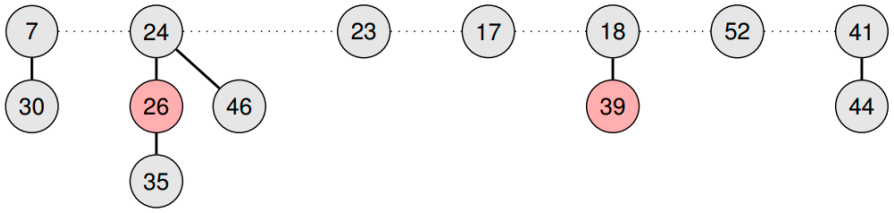

Figura 8. Passo-2 Extrair Mínimo.

Passo-3: Rearranjar as árvores de modo que não se tenha árvores de mesmo grau. (Devido a grande quantidade de passos, mostraremos apenas o resultado final. Os passos completos podem ser consultados na referência desse artigo) (Figura 9).

Passo-4: Atualizar o mínimo (Figura 10).

3.1.3. Diminuir Chave

Para a operação Diminuir Chave tem-se três casos diferentes: Ao diminuir a chave, a estrutura heap não é violada; Ao diminuir a chave, a estrutura heap é violada e o nó pai

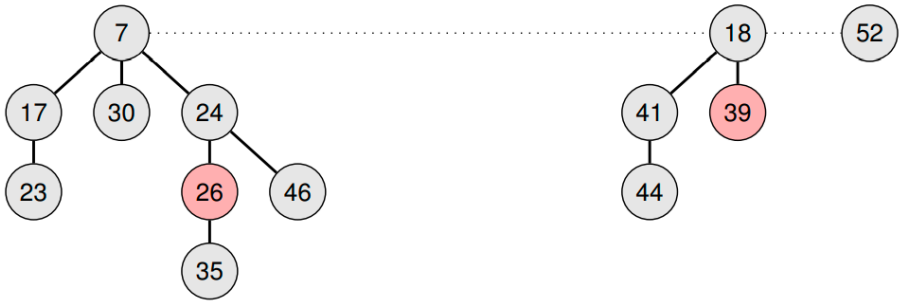

Figura 9. Passo-3 Extrair Mínimo.

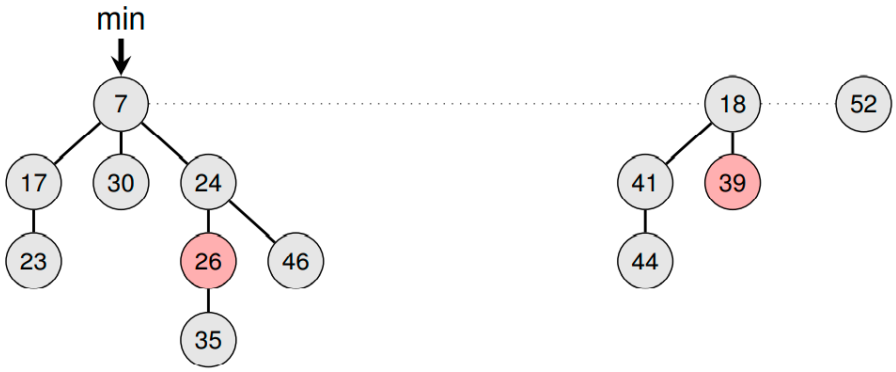

Figura 10. Passo-4 Extrair Mínimo.

do elemento diminuido não é marcado; E ao diminuir a chave, a estrutura heap é violada e o nó pai do elemento diminuido é marcado. Esta operação tem custo $O(cortes)$.

Para o caso ”ao diminuir a chave, a estrutura heap não é violada”, não há alteração da estrutura. Portanto temos dois passos.

Passo-1: Diminuir chave (Figura 11).

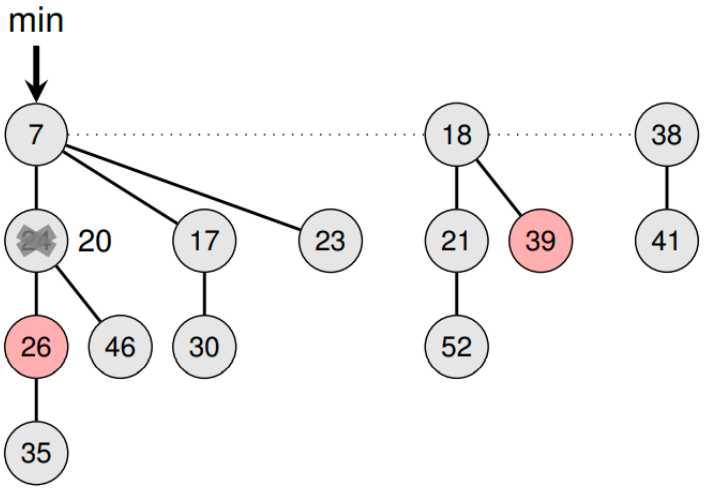

Figura 11. Passo-1 Diminuir Chave.

Passo-2: Atualizar Mínimo (quando necessário)

Para o caso ”ao diminuir a chave, a estrutura heap é violada e o nó pai do elemento diminuido não é marcado”, temos os passos:

Passo-1: Diminuir Chave (Figura 12).

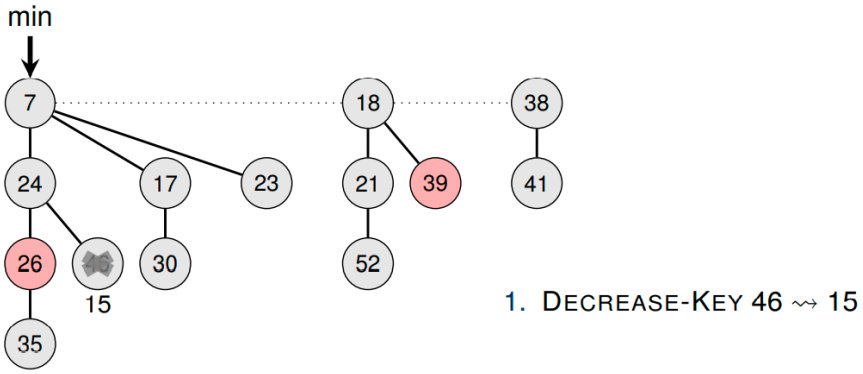

Figura 12. Passo-1 Diminuir Chave.

Passo-2: Agregar o elemento diminuído a lista de raízes (atualizar mínimo quando necessário).

Passo-3: Marcar o antigo nó pai do elemento diminuído (Figura 13).

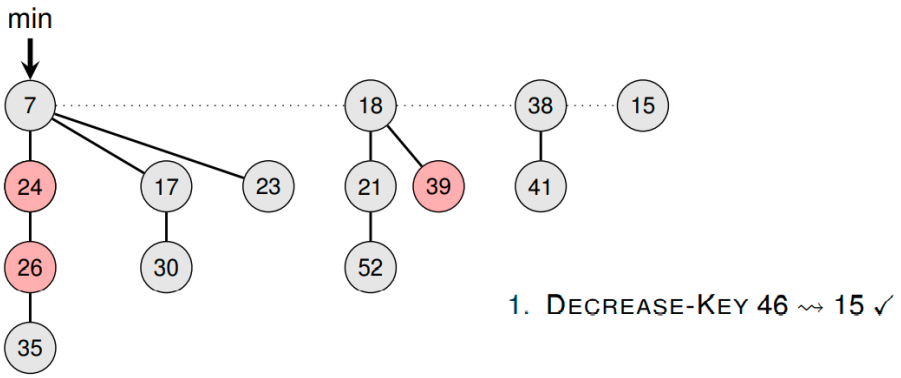

Figura 13. Passo-2 e Passo-3 Diminuir Chave.

Para o caso ”ao diminuir a chave, a estrutura heap é violada e o nó pai do elemento diminuido é marcado”, temos os passos:

Passo-1: Diminuir Chave (Figura 14).

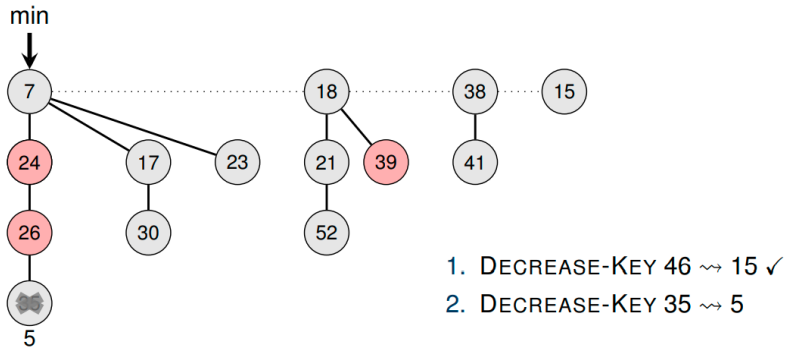

Figura 14. Passo-1 Diminuir Chave.

Passo-2: Agregar o elemento diminuído a lista de raízes (atualizar mínimo quando necessário).

Passo-3: Agregar o antigo nó pai a lista de raízes (Figura 15).

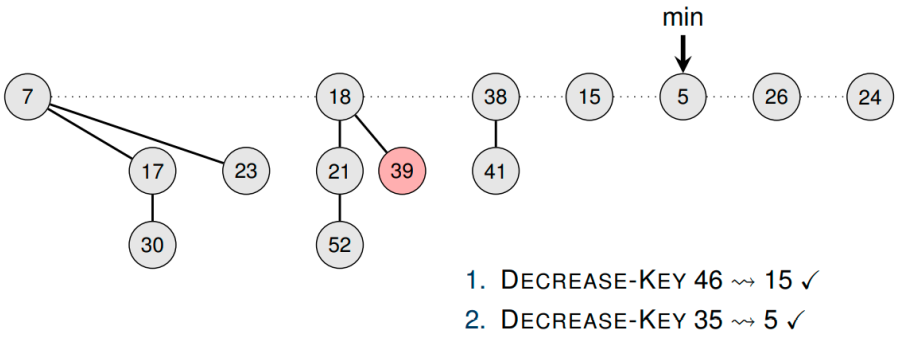

Figura 15. Passo-2 e Passo-3 Diminuir Chave.

3.1.4. Encontrar Mínimo

Como vimos, das propriedades do Heap de Fibonacci, o menor elemento está sempre sendo apontado pelo ponteiro mínimo. O custo dessa operação é $O(1)$.

3.1.5. União

A União é realizada em duas etapas. Esta operação também possui custo $O(1)$.

Passo-1: Unir as raízes das duas árvores (Figura 16).

Passo-2: Atualizar o Mínimo (Figura 17).

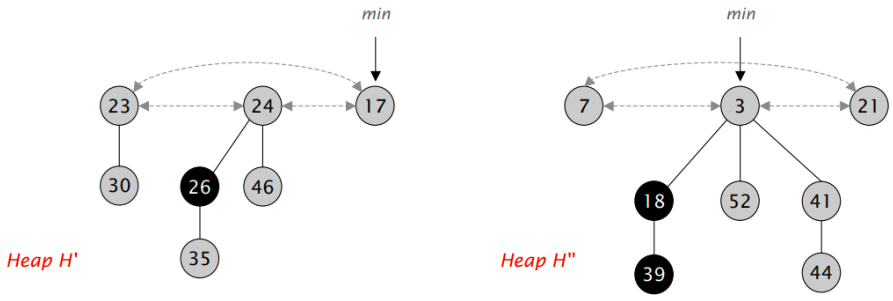

Figura 16. Passo-1 União.

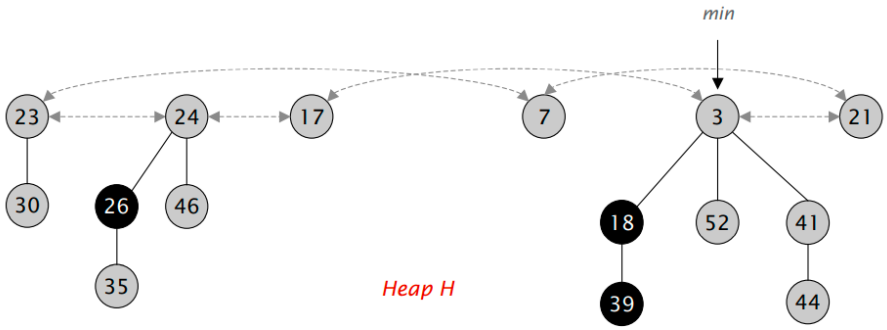

Figura 17. Passo-2 União.

3.2. Complexidade

Analisando todas as operações exemplificadas acima temos a seguinte tabela com as operações e suas respectivas complexidades assintóticas.

Inserir	O(1)
Encontrar Min	O(1)
União	O(1)
Extrair Min	O(log n)
Diminuir Chave	O(1)

4. Aplicação

A principal aplicação do Heap de Fibonacci é aprimorar o tempo de execução assintótico do algoritmo de Dijkstra.

O algoritmo de Dijkstra é usado para encontrar o caminho mais curto de um nó inicial para um nó alvo em um grafo ponderado. O algoritmo existe em muitas variações, que foram originalmente usadas para encontrar o caminho mais curto entre dois nós dados. Agora eles são mais comumente usados para encontrar o caminho mais curto da fonte para todos os outros nós no grafo, produzindo uma árvore de caminho mais curto.

4.1. Condições

É importante observar os seguintes pontos no algoritmo de Dijkstra:

- O algoritmo de Dijkstra funciona apenas para grafos conectados;
- Funciona apenas para grafos que não contêm arestas com peso negativo;
- Fornece apenas o valor ou custo dos caminhos mais curtos;
- E o algoritmo funciona para grafos direcionados e não direcionados.

4.2. Tempo de Execução

A figura a seguir (Figura 18) ilustra a comparação do tempo de execução entre seis variantes quando o número de nós está aumentando:

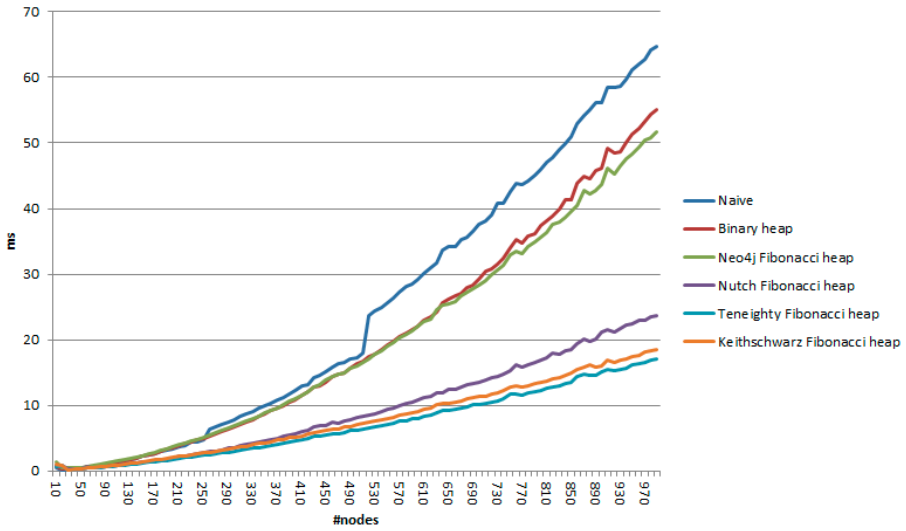

Figura 18. Gráfico de Tempo de Execução

No geral, a implementação baseada em Heap de Fibonacci será executada na velocidade mais rápida. Por outro lado, a versão mais lenta será a versão de fila de prioridade baseada em lista não ordenada.

5. Conclusão

Através deste trabalho pudemos ver a definição de um Heap de Fibonacci e suas principais operação. Além disso, mostramos as complexidades assintóticas de cada uma das operações mostradas no decorrer do texto.

Pudemos concluir que o Heap de Fibonacci possui um tempo de execução muito favorável ao algoritmo de Dijkstra, justamente onde se encontra sua principal utilização. Numa comparação com outras variantes, o Heap de Fibonacci se sobressai no tempo de execução, sendo o melhor dentre todos.

6. References

STAJANO, Frank; SAUERWALD, Thomas. Amortized Analysis. Department of Computer Science and Technology, 2022. Disponível em: https://www.cl.cam.ac.uk/teaching/1415/Algorithms/2015-sauerwald-algs-students-slides.pdf, acessado em 25/06/2022