Estrutura de Dados

Brayan Martins, Carlos Daniel

E-Mail: carlosdaniemds32@gmail.com

E-mail: brayamartins9@gmail.com

Sintemas de Informação - Amf

índice

Introdução

- Vamos explorar conceitos teóricos fundamentais sobre filas de prioridade, especificamente os heaps mínimo, máximo e o desafiador heap de Fibonacci. Vamos não apenas entender suas estruturas e operações, mas também ver exemplos práticos de como são aplicados em diversas áreas, desde a hierarquização de dados na saúde até otimizações em biologia computacional. Além disso, teremos uma demonstração prática em código para ilustrar como essas estruturas funcionam na prática. Vamos mergulhar juntos nesse universo fascinante das filas de prioridade e seus impactos na eficiência computacional.

Implementação em python

- **Python é uma escolha popular para implementar algoritmos complexos como filas de prioridade devido à sua legibilidade, flexibilidade, disponibilidade de bibliotecas e suporte ativo da comunidade, facilitando o desenvolvimento e compreensão dessas estruturas críticas em computação.**

- **Existem várias razões pelas quais Python é frequentemente escolhido para implementar algoritmos complexos como os de filas de prioridade, como heaps mínimo, máximo e de Fibonacci:**

Heap Binários

- **Heap binário é uma estrutura eficiente que permite acesso rápido ao elemento de maior ou menor prioridade, adequando-se a uma variedade de problemas computacionais que requerem ordenação e manipulação de dados baseados em prioridade.**

Heap Fibonacci

- **O heap de Fibonacci é uma estrutura de dados sofisticada e eficiente que combina as propriedades de ordenação de heaps com a estrutura flexível e rápida das árvores de Fibonacci, oferecendo desempenho superior em muitas operações importantes.**

Heap Mínimo

- Heap mínimo é uma estrutura de dados eficiente e poderosa para manter e manipular coleções de elementos onde a prioridade ou o valor mínimo é uma consideração importante. Ele oferece operações rápidas de inserção, remoção do menor elemento e acesso ao elemento mínimo, sendo amplamente utilizado em algoritmos que exigem eficiência na manipulação de prioridades.

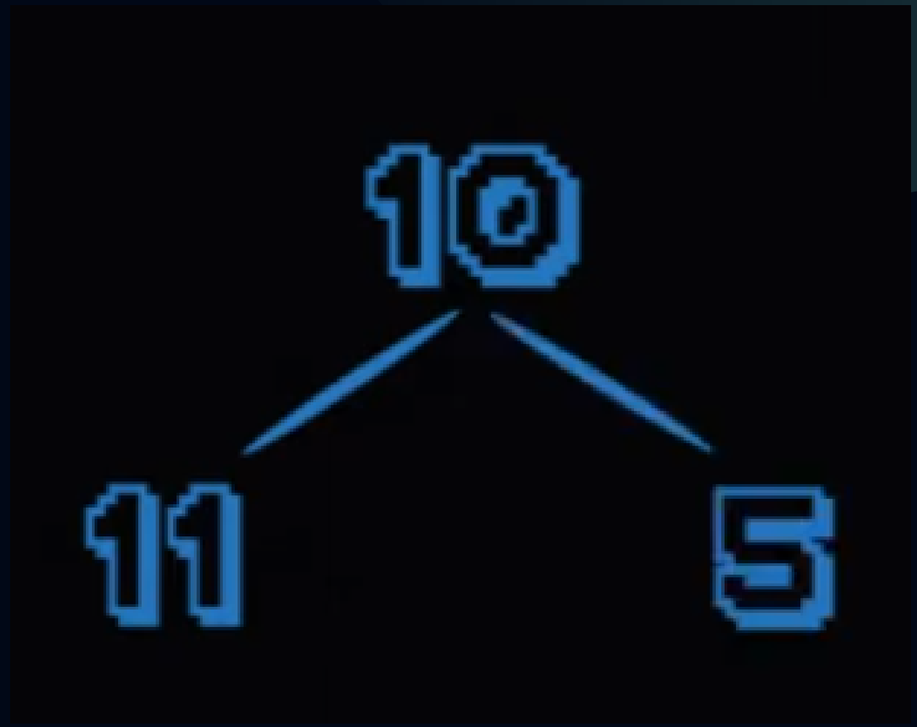
Heap Mínimo

Heap Mínimo

10 11 5 13 19

Heap Mínimo

10 11 5 13 19

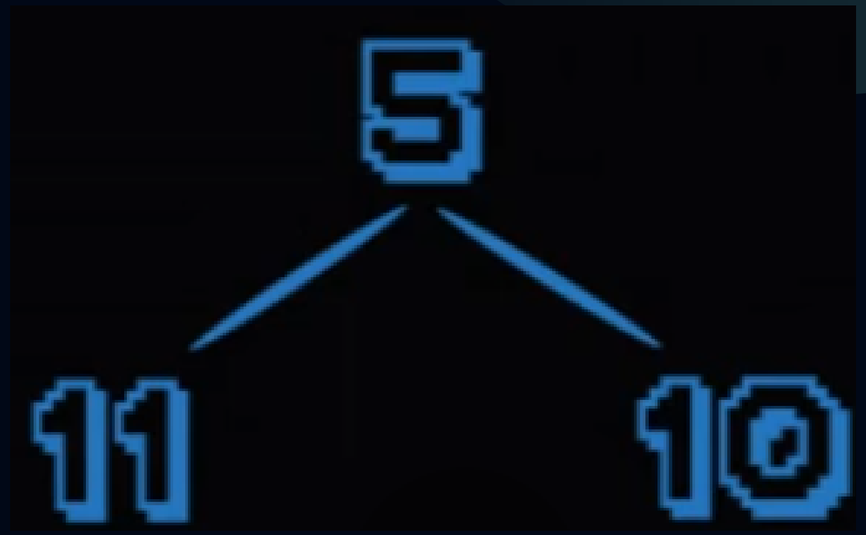


Maior

Menor

Heap Mínimo

10 11 5 13 19

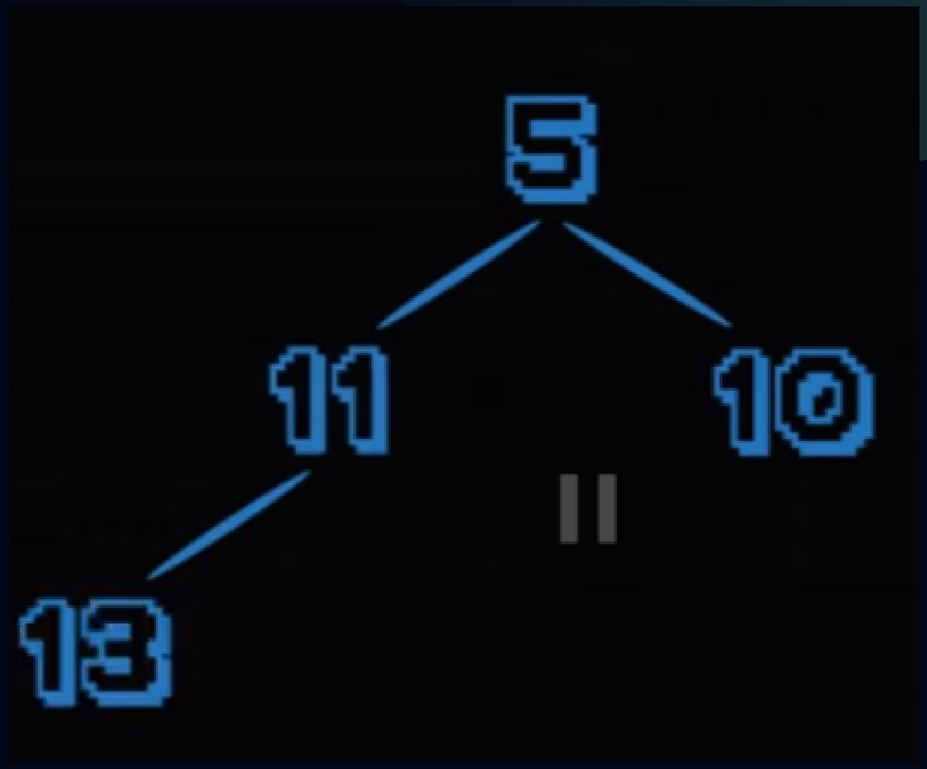


Menor

Maior

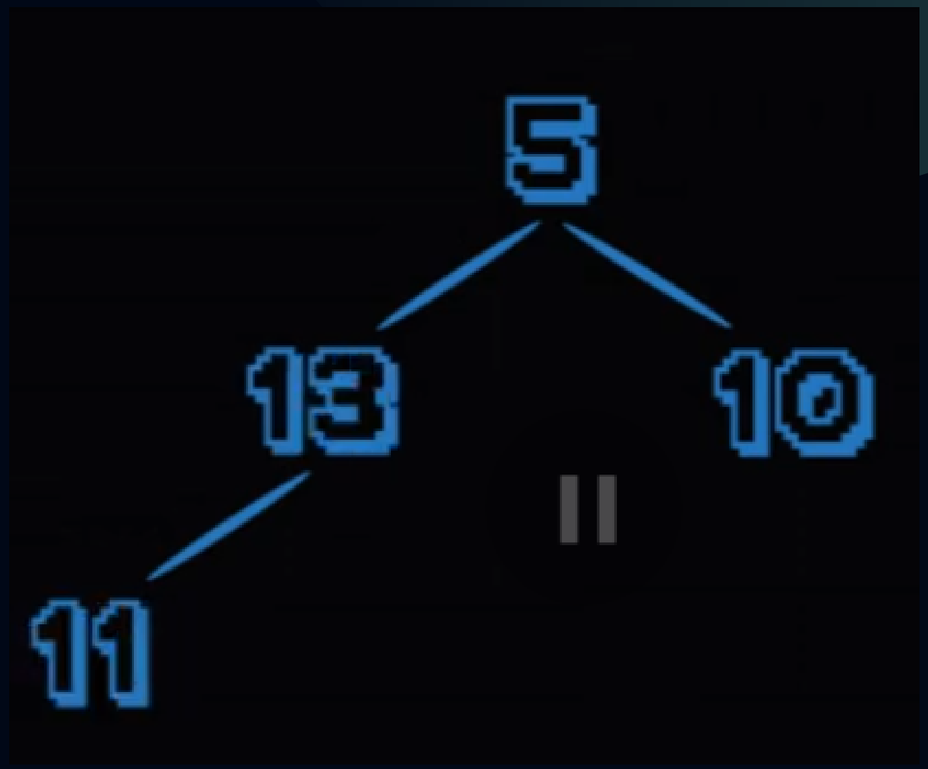
Heap Mínimo

10 11 5 13 19



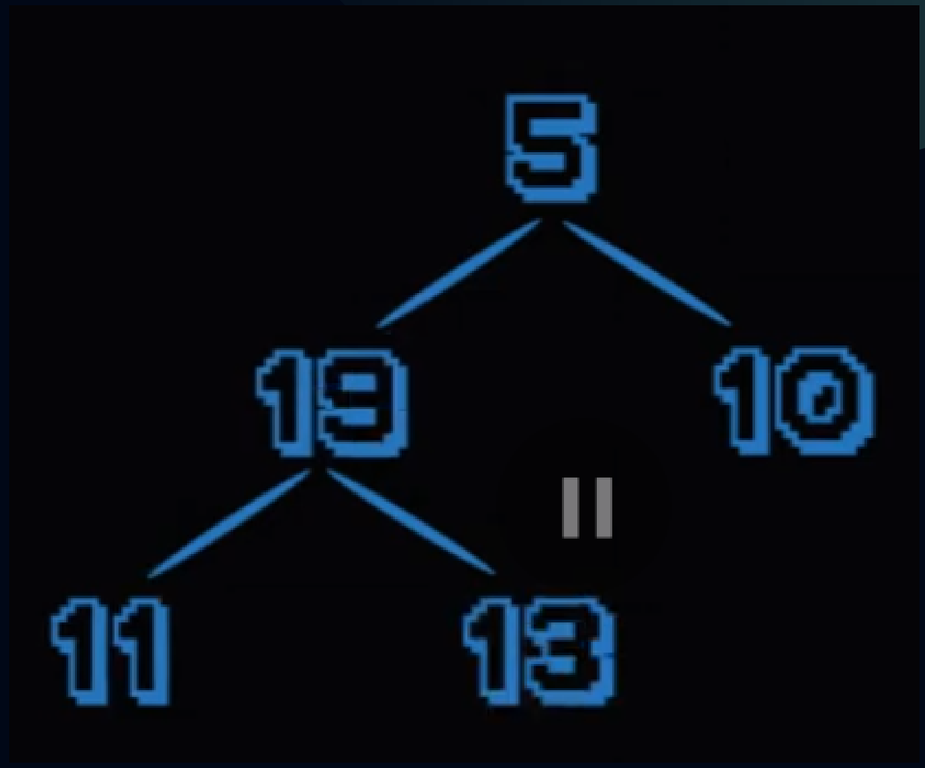
Heap Mínimo

10 11 5 13 19



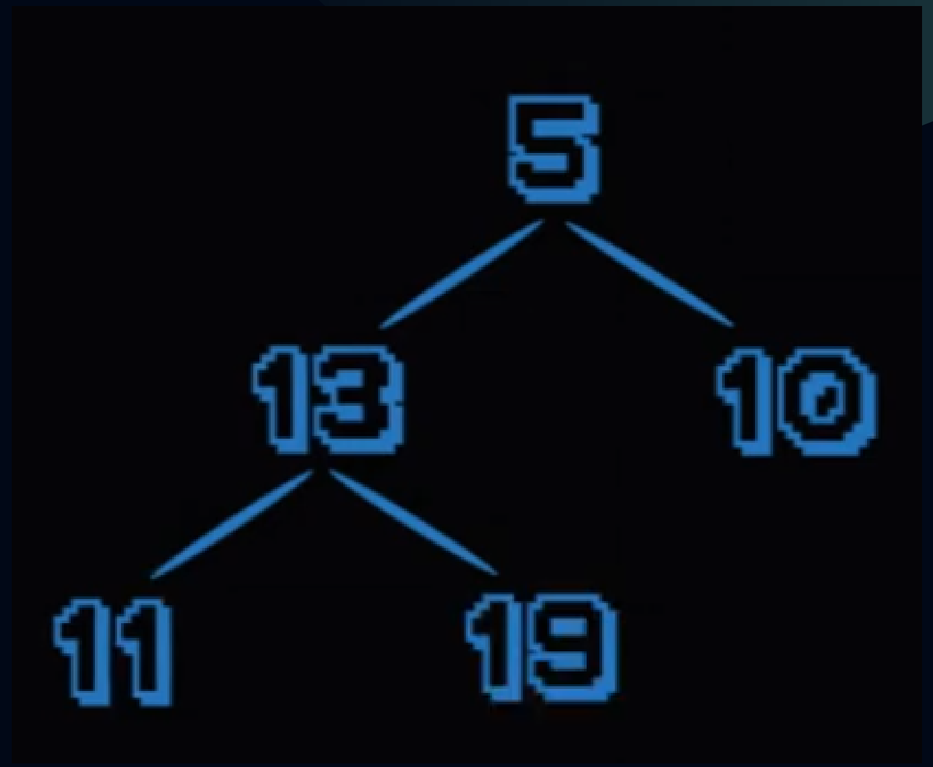
Heap Mínimo

10 11 5 13 19

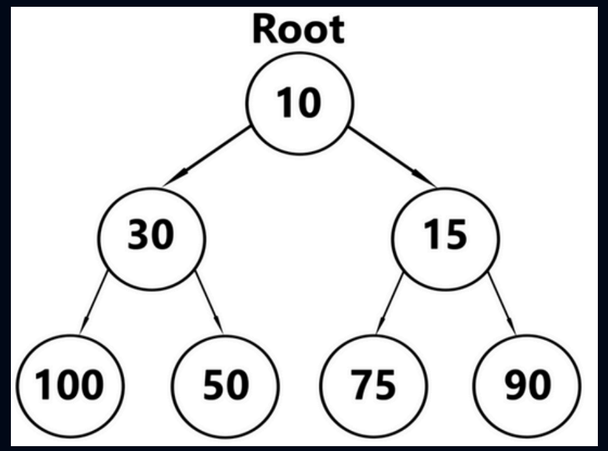


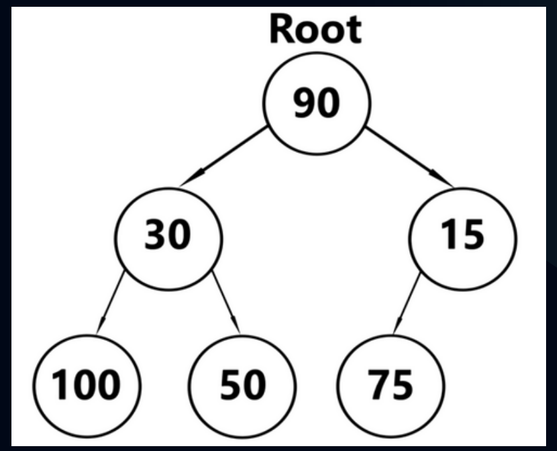
Heap Mínimo

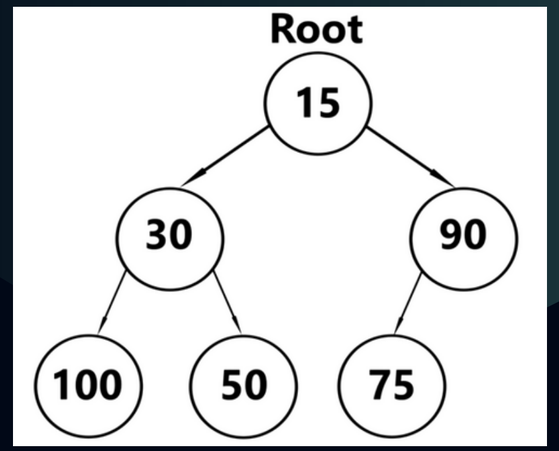
10 11 5 13 19

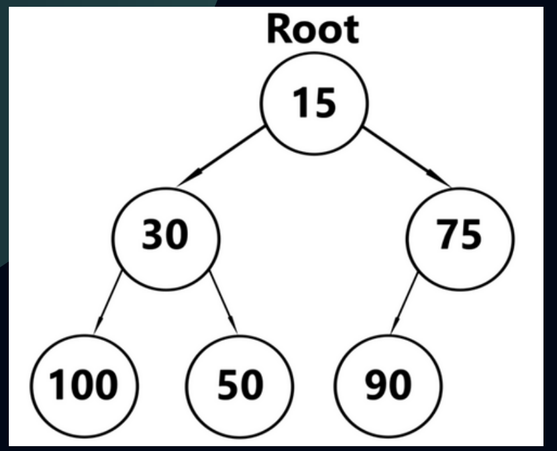


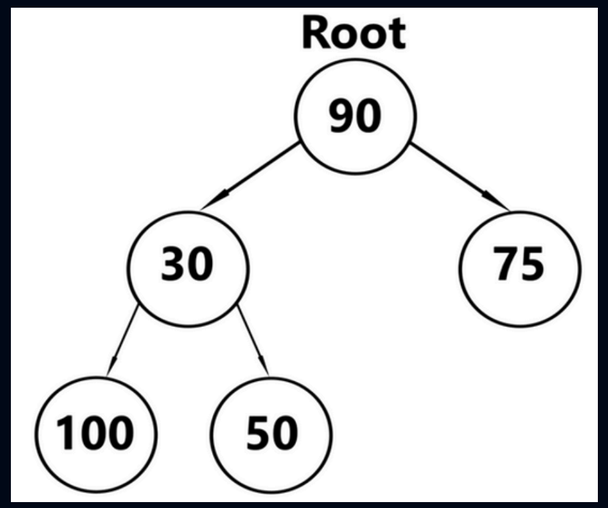
Heap Mínimo

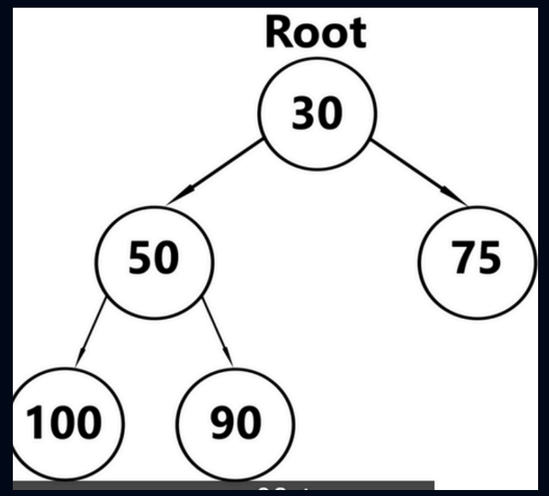

Root
10
30
15
100
50
75
90

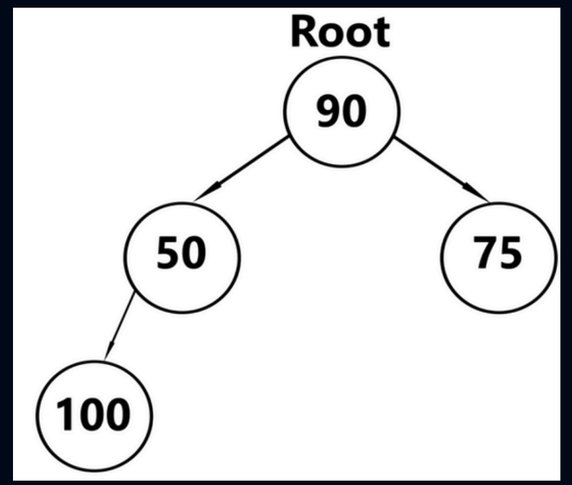

Root
90
30
15
100
50
75

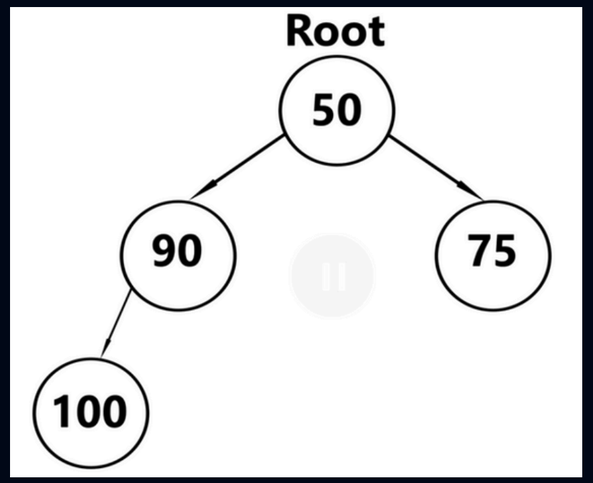

Root
15
30
90
100
50
75


Root
15
30
75
100
50
90


Root
90
30
75
100
50


Root
30
50
75
100
90

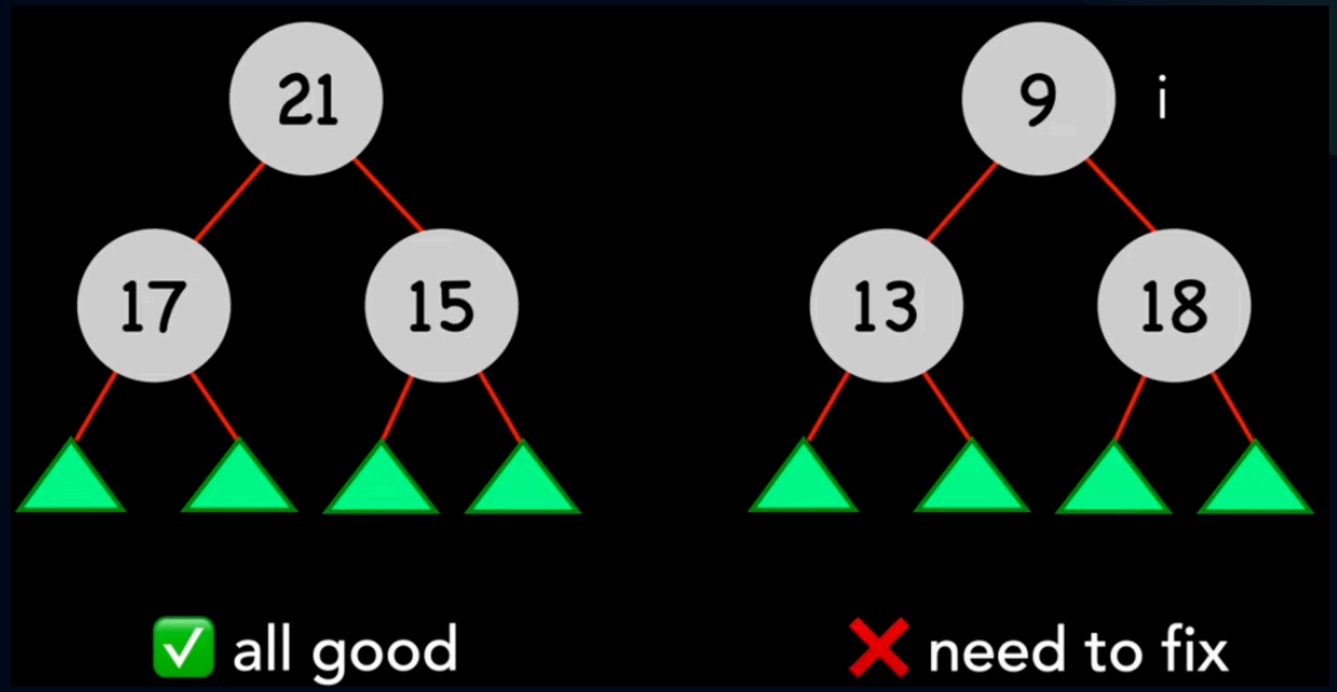

Root
90
50
75
100


Root
50
90
75
100

Heap Máximo

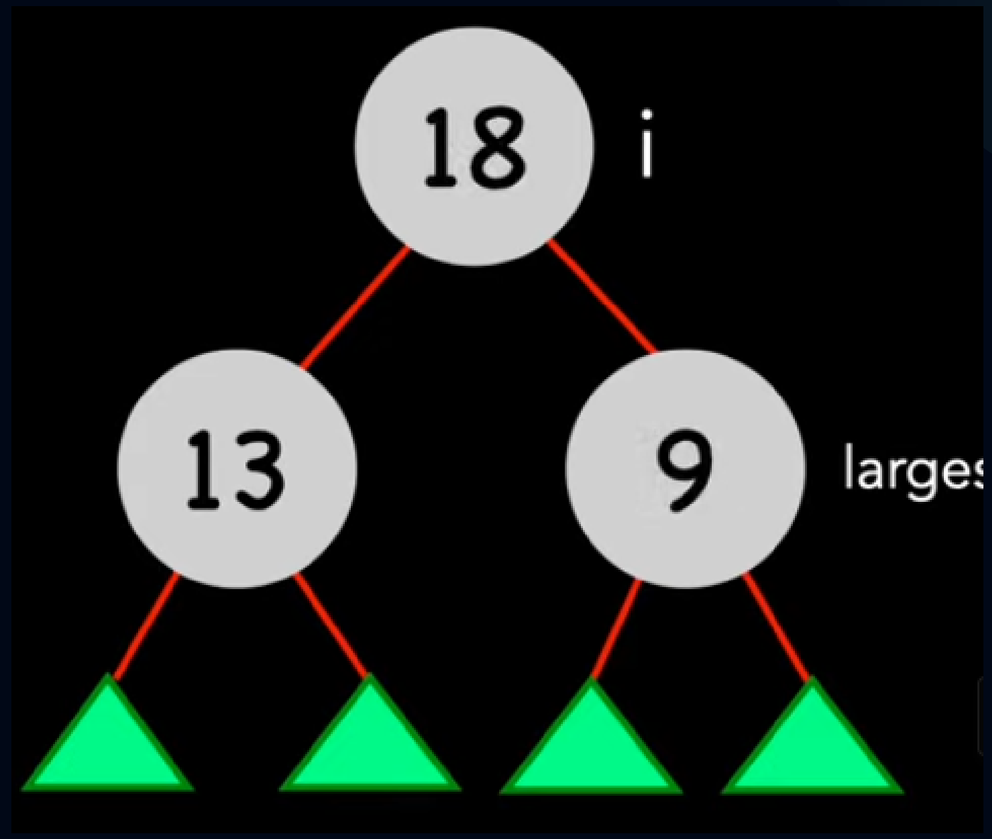
- Heap máximo é uma estrutura de dados eficiente e poderosa para manter e manipular coleções de elementos onde a prioridade ou o valor máximo é uma consideração importante. Ele oferece operações rápidas de inserção, remoção do maior elemento e acesso ao elemento máximo, sendo amplamente utilizado em algoritmos que exigem eficiência na manipulação de prioridades de forma descendente.

exemplo máximo




```python
def max_heapify(a, heap_size, i):
    l = 2*i
    r = 2*i + 1

    largest = i

    if l < heap_size and a[l] > a[i]:
        largest = l

    if r < heap_size and a[r] > a[largest]:
        largest = r

    if largest != i:
        # swap elements
        a[i], a[largest] = a[largest], a[i]
        max_heapify(a, heap_size, largest)
```

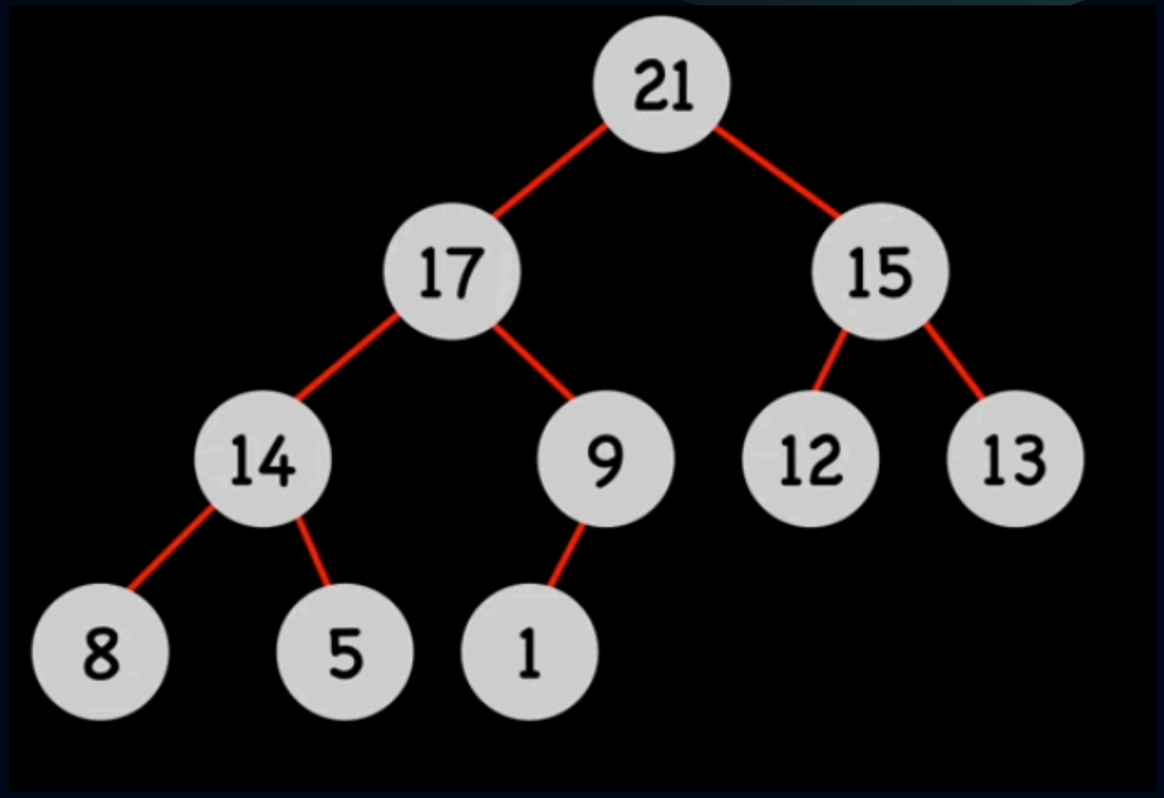
exemplo máximo




```python
def max_heapify(a, heap_size, i):
    l = 2*i
    r = 2*i + 1

    largest = i

    if l < heap_size and a[l] > a[i]:
        largest = l

    if r < heap_size and a[r] > a[largest]:
        largest = r

    if largest != i:
        # swap elements
        a[i], a[largest] = a[largest], a[i]
        max_heapify(a, heap_size, largest)
```

Heap Máximo

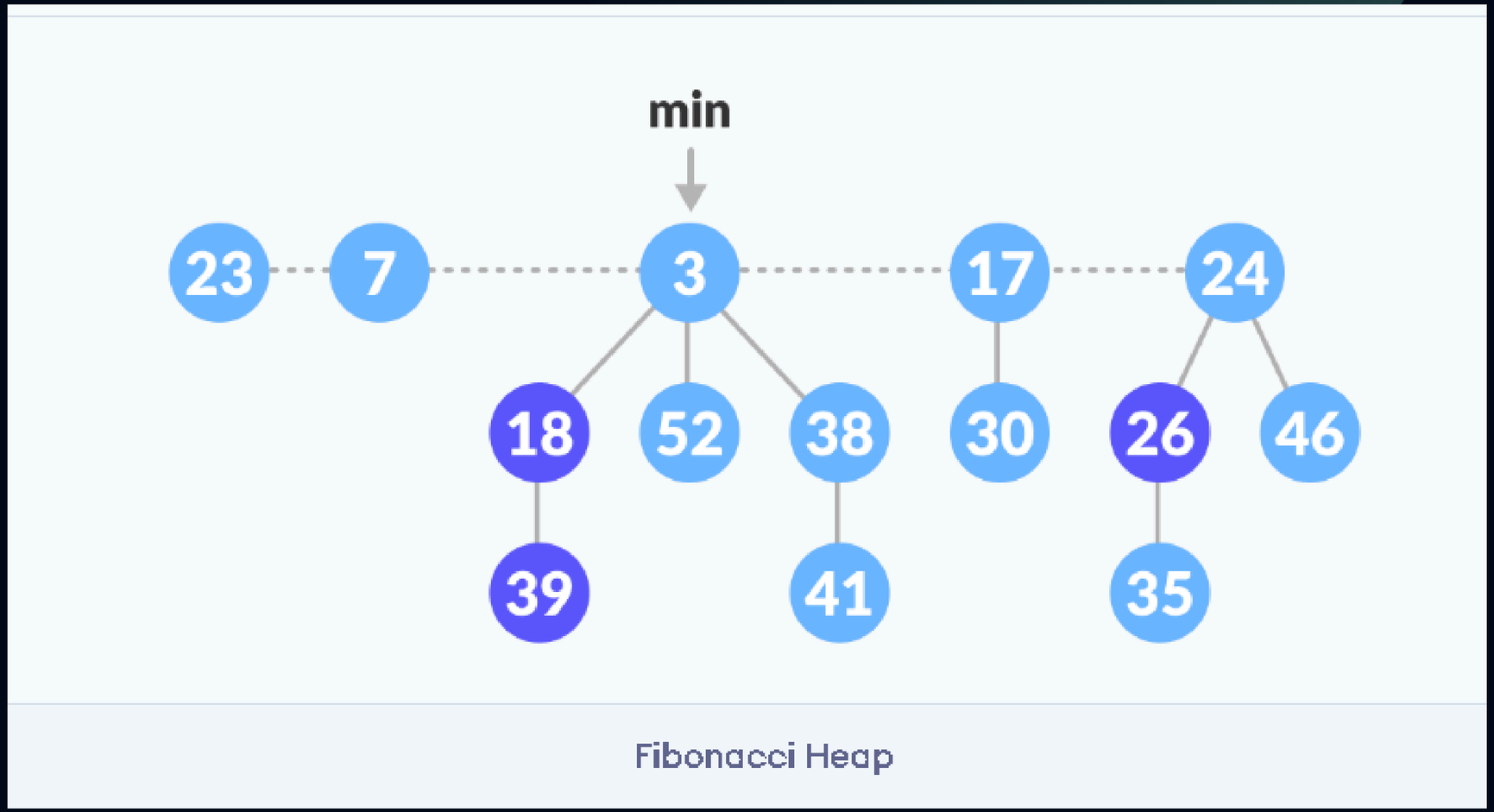
21	17	15	14	9	12	13	8	5	1
1	2	3	4	5	6	7	8	9	10



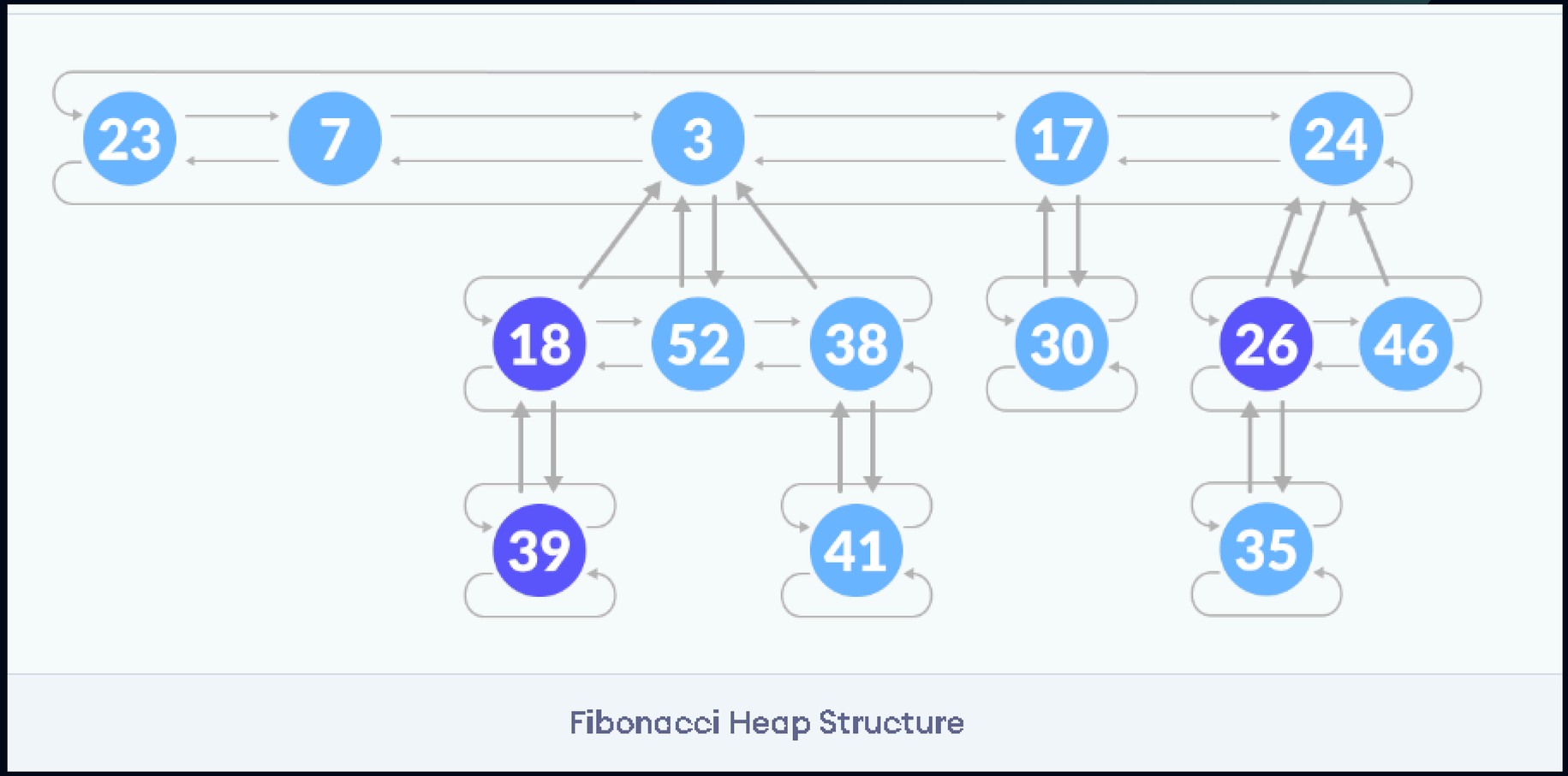
Heap de Fibonacci

- **Heap de Fibonacci é uma estrutura de dados avançada que combina eficiência com flexibilidade, ideal para aplicações que exigem manipulação dinâmica de prioridades com alto desempenho. Ele representa uma escolha poderosa em algoritmos que requerem operações eficientes de filas de prioridade dinâmicas em tempo real.**

Heap de Fibonacci

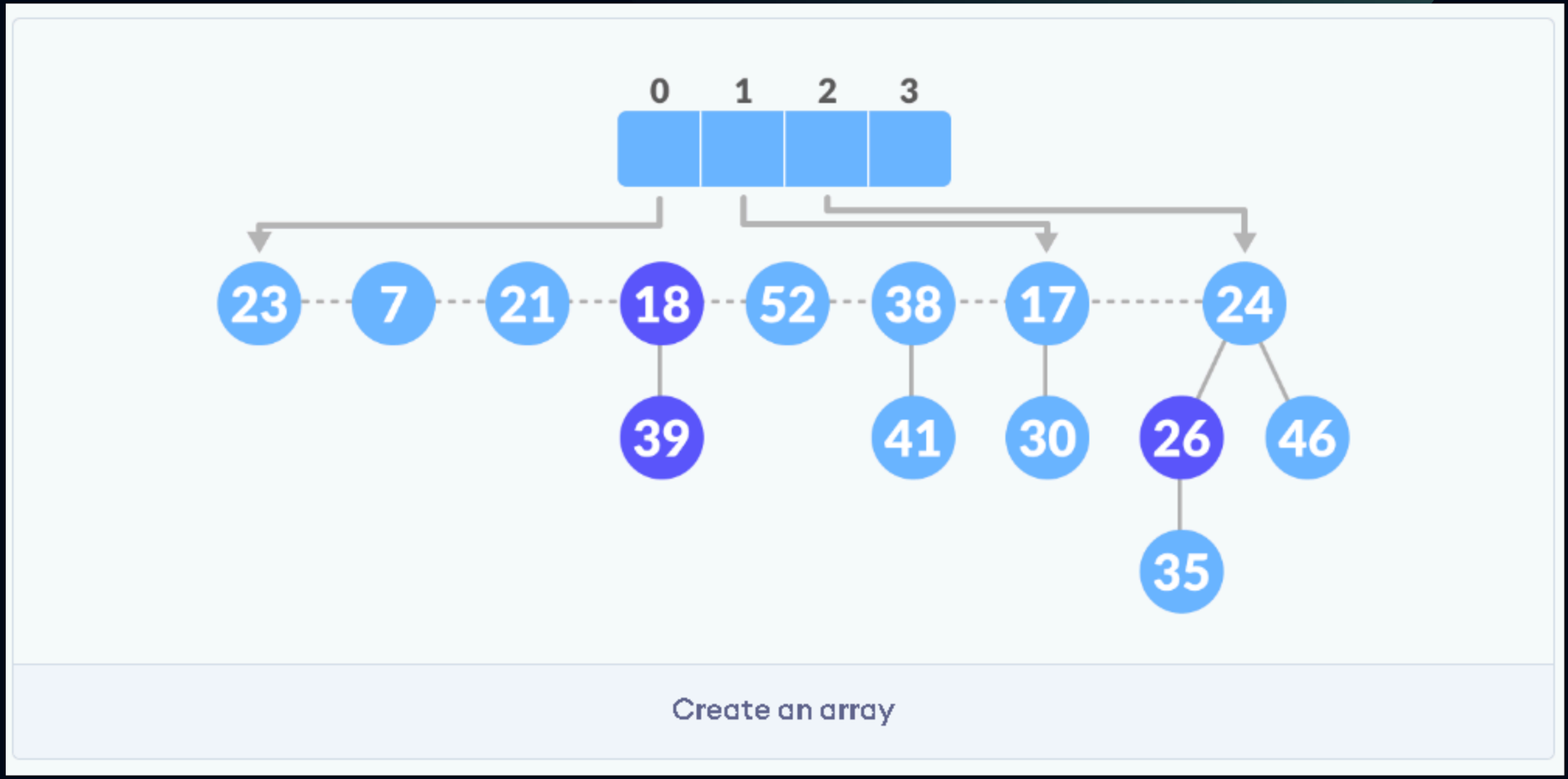


Heap de Fibonacci

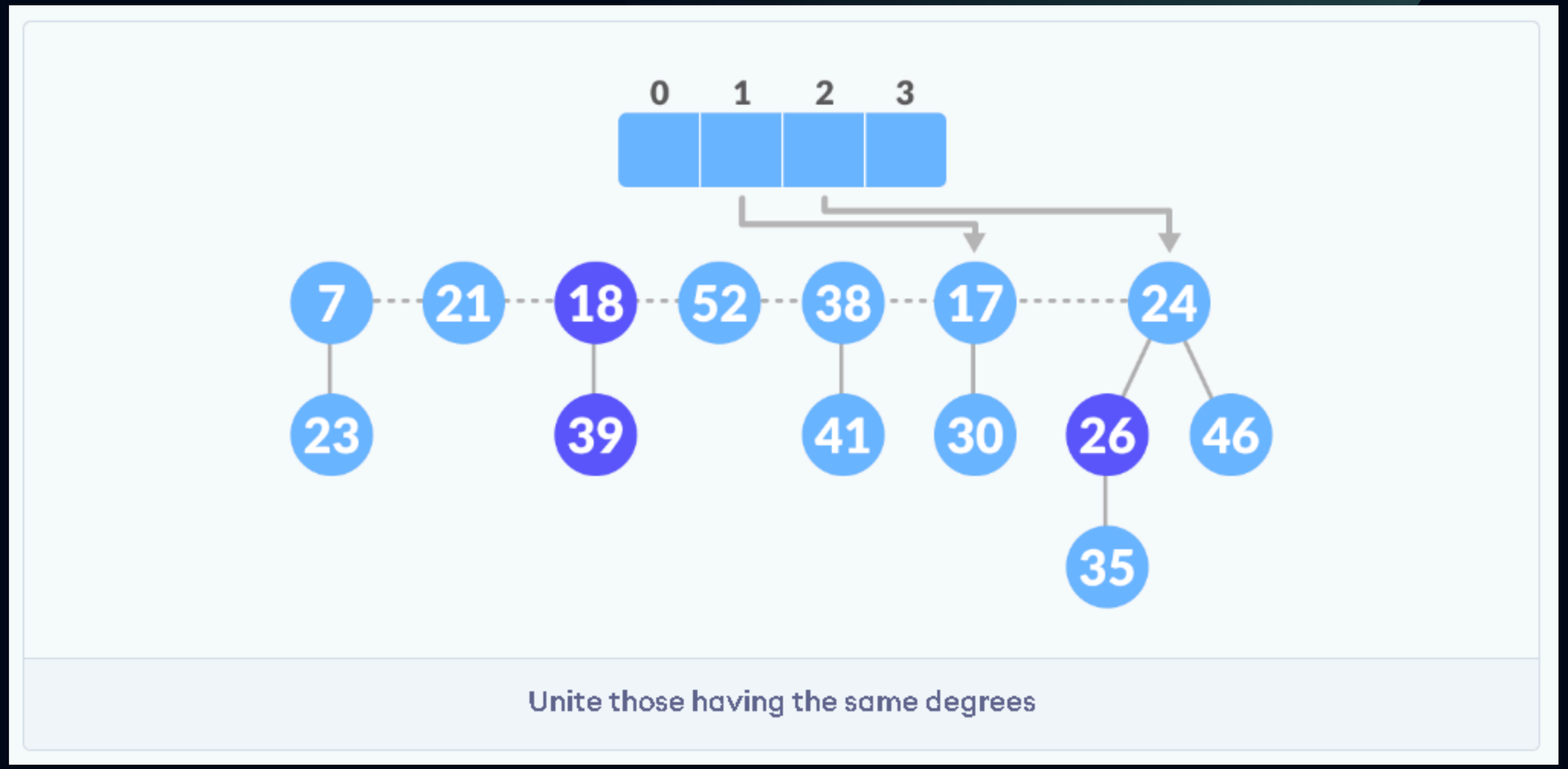


Fibonacci Heap Structure

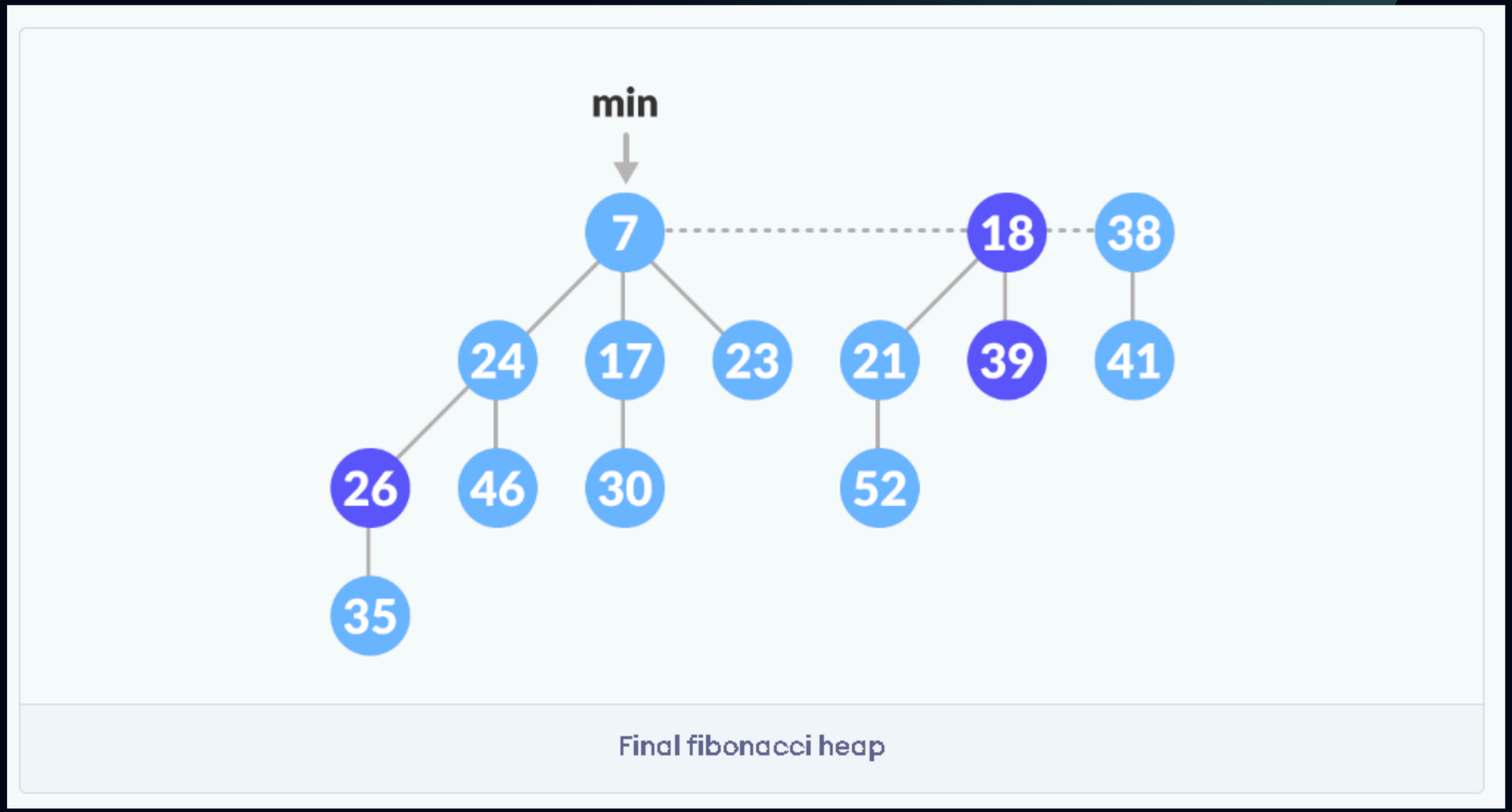
Heap de Fibonacci



Heap de Fibonacci



Heap de Fibonacci



Obrigado!