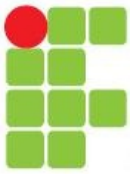
INSTITUTO FEDERAL DE ALAGOAS – IFAL / CAMPUS MACEIÓ
Curso Bacharelado em Sistemas de Informação

Prof. MSc. Ricardo Nunes
Ricardo (arroba) ifal.edu.br



Árvores (3)

Objetivos

- Implementar uma fila de prioridade usando um heap

Filas de prioridade (priority queues)

- Variação das filas
- Dequeue: igual as filas
- A ordem interna é determinada pela prioridade
  - Prioridade alta fica no início da fila
  - Prioridade baixa fica no final da fila

Desempenho fila prioridade

- Uma fila de prioridade poderia ser implementada com uma lista, mas inserir seria O(n) e ordenar O(n log(n))
- Fila de prioridade com Binary Heap permite enqueue e dequeue em O(log(n))

Binary Heap

- Se parece muito com uma árvore
- Mas quando sua implementação usa apenas uma única lista como representação interna
- Variações
  - heap mínimo: o menor valor de chave está sempre na frente
  - heap máximo: o maior valor de chave está sempre na frente

Operações Binary Heap

- BinaryHeap()
- insert(k)
- findMin()
- delMin()
- isEmpty()
- size()
- buildHeap(list)

Exemplo de uso de binary heap

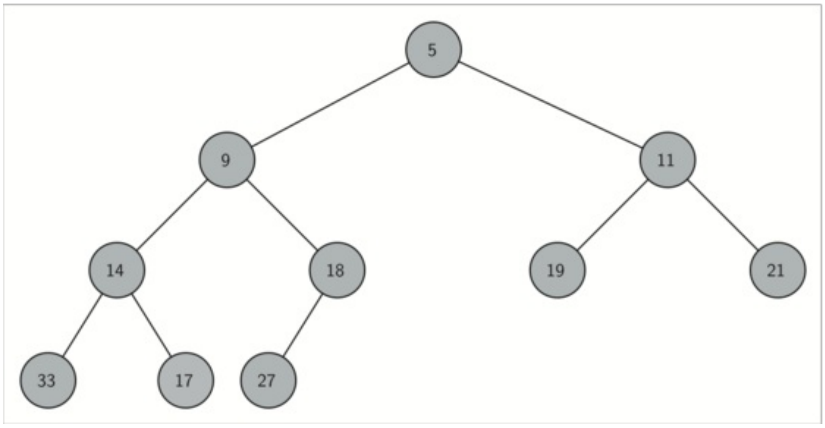
```
bh = BinHeap()
bh.insert(5)
bh.insert(7)
bh.insert(3)
bh.insert(11)

print(bh.delMin())
print(bh.delMin())
print(bh.delMin())
print(bh.delMin())
```

Propriedades estruturais

- Para garantir o desempenho, é preciso deixar a árvore balançada
- uma árvore balançada tem o mesmo número de nós na esquerda e direita, com excessão do último nível

Exemplo

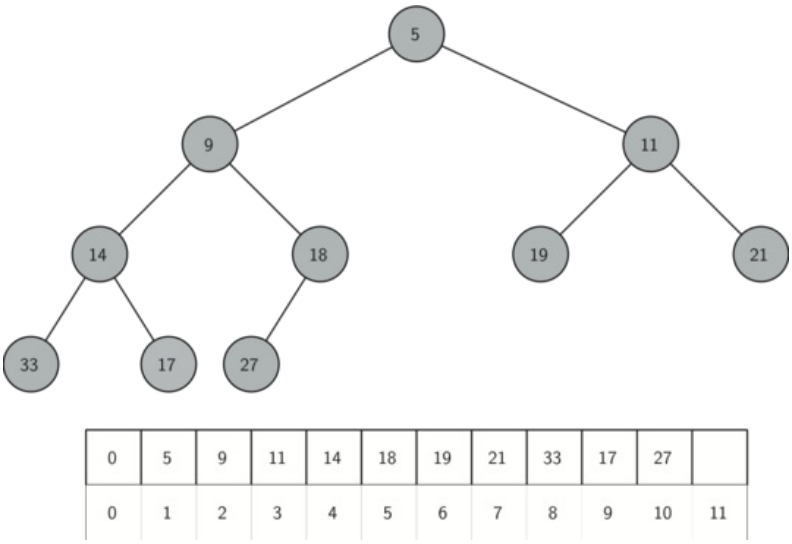


Propriedade estruturais (2)

- Pode ser representada em uma única lista (sem nós e referências ou lista de listas)
- O filho esquerdo de uma pai (na posição p) é um nó que está na posição 2p na lista
- O filho direito de uma pai (na posição p) é um nó que está na posição 2p+1 na lista
- Encontrar o pai de um nó: (n)//2

Propriedade de ordem de um Heap

- Para todo nó **x** com pai **p**, a chave em **p** é menor ou igual a chave em **x**



0	5	9	11	14	18	19	21	33	17	27	
0	1	2	3	4	5	6	7	8	9	10	11

Operações com Heap

Criando um Heap

```
class BinHeap:
    def __init__(self):
        self.heapList = [0]
        self.currentSize = 0
```

Inserindo (1)

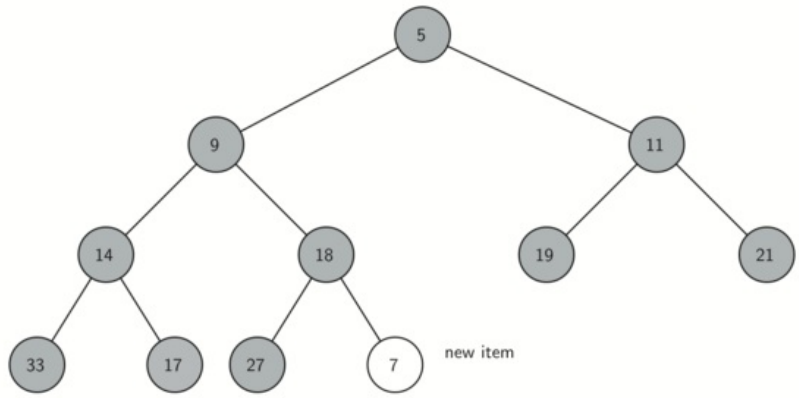
- A maneira mais fácil e eficiente de adicionar um item a uma lista é simplesmente anexar o item ao final da lista
    - Essa operação garante que manteremos as propriedades da árvore

- Provavelmente a propriedade de estrutura de heap será *violada*

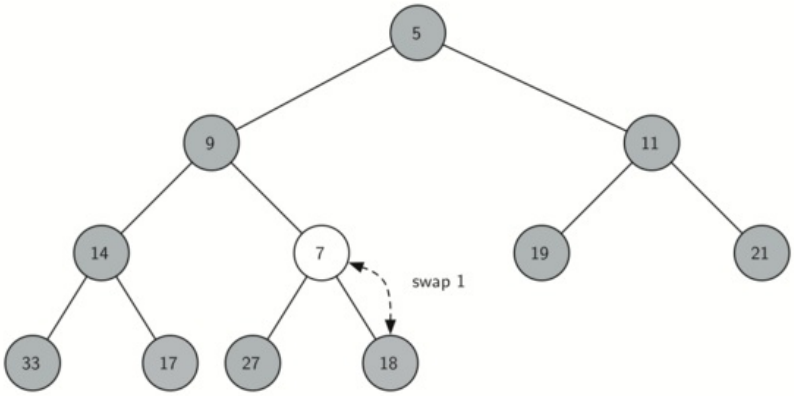
Passos

- Fazer um append no final da lista
- Reordenar o elemento inserido comparando com seus pais
  - Se ele for menor que o pai, troca de posição

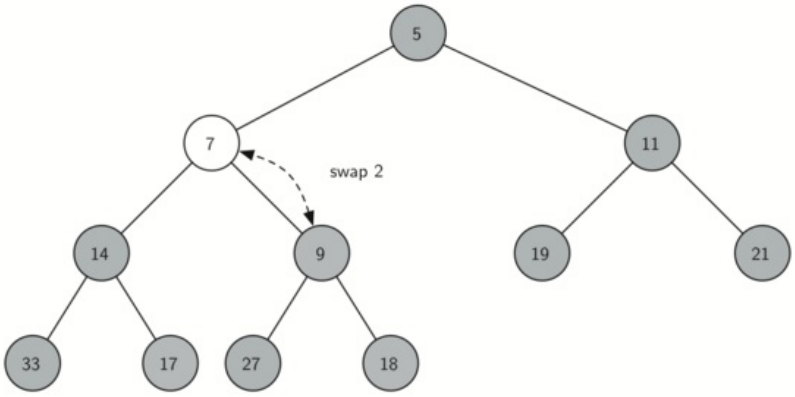
Inserindo (2)



Inserindo (3)



Inserindo (4)



Inserindo (3)

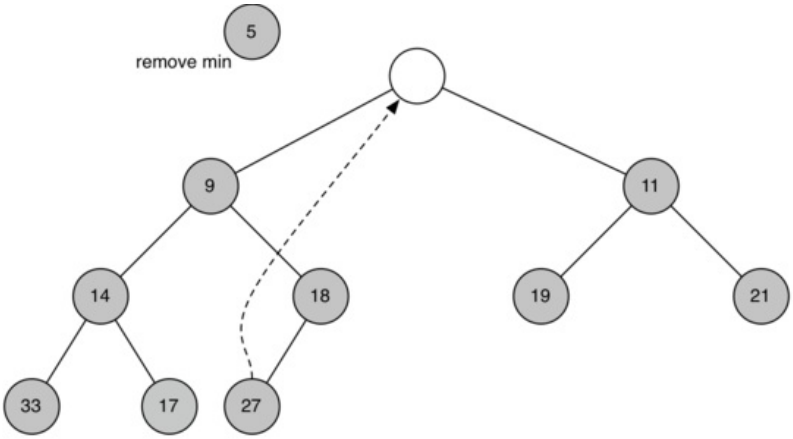
```
def insert(self,k):
    self.heapList.append(k)
    self.currentSize = self.currentSize + 1
    self.percUp(self.currentSize)

def percUp(self,i):
    while i // 2 > 0:
      if self.heapList[i] < self.heapList[i // 2]:
         tmp = self.heapList[i // 2]
         self.heapList[i // 2] = self.heapList[i]
         self.heapList[i] = tmp
      i = i // 2
```

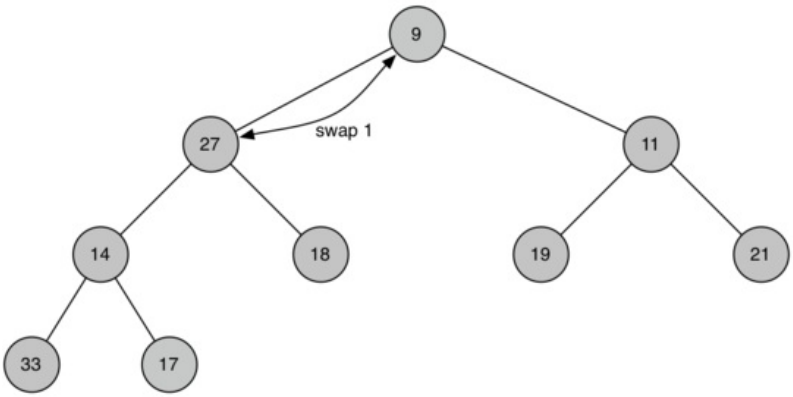
Removendo (o menor valor)

- O menor valor está no início da lista
- Será preciso restaurar as propriedades
- Passos:
  - Mover o último item (para raiz) para manter a estrutura
  - trocar de posição os nós para manter a ordem adequada

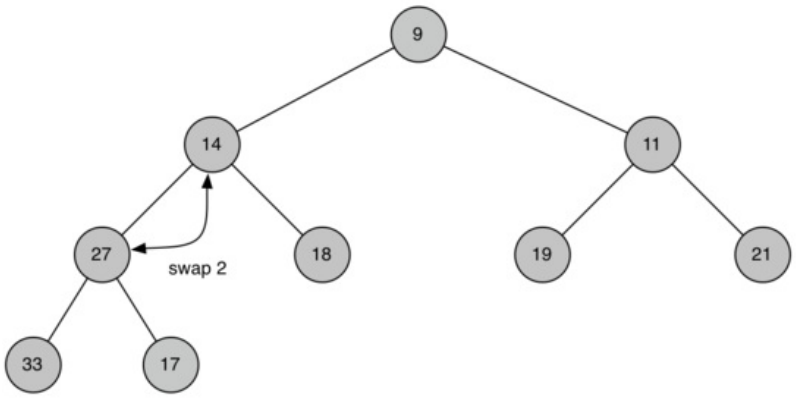
Removendo (2)



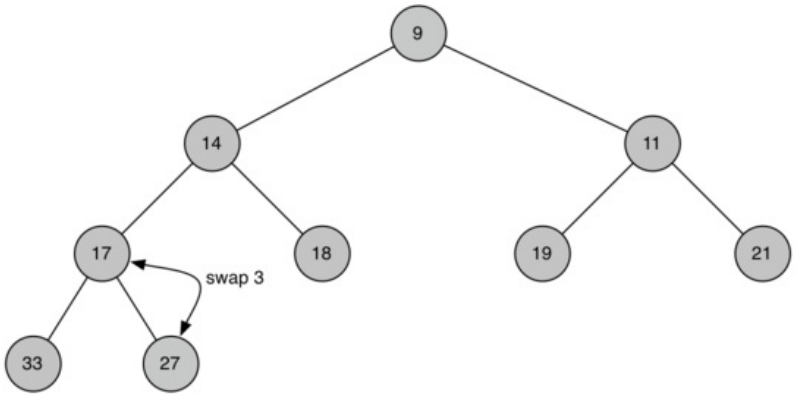
Removendo (3)



Removendo (4)



Removendo (5)



Removendo (3)

```
def percDown(self,i):
    while (i * 2) <= self.currentSize:
        mc = self.minChild(i)
        if self.heapList[i] > self.heapList[mc]:
            tmp = self.heapList[i]
            self.heapList[i] = self.heapList[mc]
            self.heapList[mc] = tmp
        i = mc

def minChild(self,i):
    if i * 2 + 1 > self.currentSize:
        return i * 2
    else:
        if self.heapList[i*2] < self.heapList[i*2+1]:
            return i * 2
        else:
            return i * 2 + 1
```

Removendo (4)

```
def delMin(self):
    retval = self.heapList[1]
    self.heapList[1] = self.heapList[self.currentSize]
    self.currentSize = self.currentSize - 1
    self.heapList.pop()
    self.percDown(1)
    return retval
```

Construindo um Heap a partir de uma lista (1)

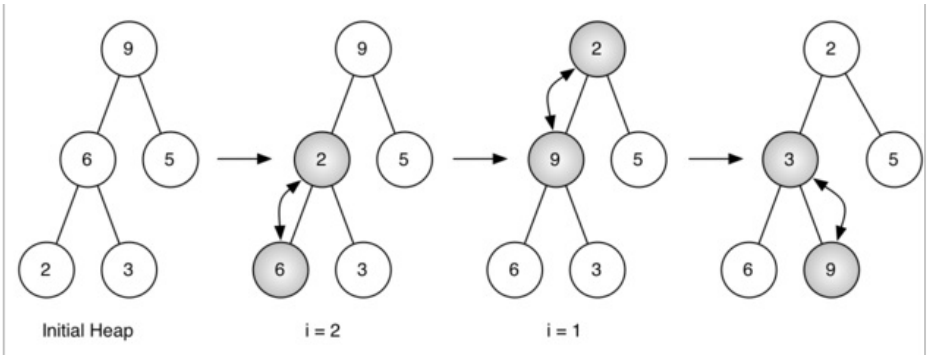
Método 1

- Dada uma lista de chaves, insira uma chave de cada vez
- Como é iniciado a partir de uma lista (heap) vazio, pode-se usar a pesquisa binária para encontrar a posição correta para inserir a próxima chave a um custo de O(log n)
  - Mas deslocar os elementos na lista pode exigir O(n) deslocamentos,
  - custo total será O (n log(n))

Construindo um Heap a partir de uma lista (2)

Método 2: Começar a partir da lista inteira

```
def buildHeap(self,alist):
    i = len(alist) // 2
    self.currentSize = len(alist)
    self.heapList = [0] + alist[:]
    while (i > 0):
        self.percDown(i)
        i = i - 1
```



Para estudar

- Árvores
  - Seções 7.8 a 7.10 do livro [6] https://runestone.academy/ns/books/published//pythonds/Trees/toctree.html (em inglês)
  - Capítulo sobre árvores em qualquer livro sobre estruturas de dados

Referências

1. Tradução do livro *How to Think Like a Computer Scientist: Interactive Version*, de Brad Miller e David Ranum. link: https://panda.ime.usp.br/pensepy/static/pensepy/index.html
2. Allen Downey, Jeff Elkner and Chris Meyers. *Aprenda Computação com Python 3.0*. link: https://chevitarese.files.wordpress.com/2009/09/aprendacomputaocompython3k.pdf
3. SANTOS, A. C. *Algoritmo e Estrutura de Dados I*. 2014. Disponível em http://educapes.capes.gov.br/handle/capes/176522
4. SANTOS, A. C. *Algoritmo e Estrutura de Dados II*. 2014. Disponível em 2014. Disponível em https://educapes.capes.gov.br/handle/capes/176557
5. Tradução do livro [6] *Problem Solving with Algorithms and Data Structures using Python* de Brad Miller and David Ranum. link: https://panda.ime.usp.br/pythonds/static/pythonds_pt/index.html
6. Brad Miller and David Ranum. *Problem Solving with Algorithms and Data Structures using Python* link: https://runestone.academy/ns/books/published//pythonds/index.html
7. Caelum. Algoritmos e Estruturas Dados em Java. Disponível em https://www.caelum.com.br/download/caelum-algoritmos-estruturas-dados-java-cs14.pdf

That's all Folks