UFMS DIGITAL

Estrutura de Dados

Prof. Dr. Gedson Faria

Prof.ª Dr.ª Graziela Santos de Araújo

Prof. Dr. Jonathan de Andrade Silva

UFMS DIGITAL

Módulo 1 - Hash e Heap

Unidade 2 - Heap

UFMS DIGITAL

Conceitos e Definições

Motivação

- Até agora vimos diferentes estruturas de dados com o objetivo de melhorar os custos de inserção, remoção ou busca de elementos em um conjunto;
  - Seja com Array (vetor), Lista, Pilha, Fila e Tabela de dispersão.
- Mesmo se conseguirmos o menor custo para buscar um elemento no conjunto, esse custo se manteria caso quiséssemos o menor ou o maior elemento do conjunto?
  - Não, precisaríamos ordenar o conjunto de dados, caro!

Motivação

- Poderíamos então explorar a estrutura de dados Fila de prioridades.
  - Permite organizar/ordenar os elementos do conjunto por meio de seus valores de prioridades (ou chaves).

Motivação

- Imagine em um posto de saúde em que uma triagem é feita para realizar os atendimentos dos pacientes.
  - A triagem determina uma prioridade ou chave (fita verde, amarela ou vermelha) para cada paciente;
  - Os atendimentos realizam uma busca pelo paciente de maior prioridade de atendimento (fita vermelha, primeiro).

Heaps

- Heap pode ser visto como uma estratégia para manter ordenado (mesmo após inserção ou remoção) os elementos em uma fila de prioridades “ordenado” com custo menor que O(N), onde N é o tamanho do conjunto;
- Podemos ter duas maneiras de ordenar as prioridades: do maior para o menor, ou vice e versa.
  - Max-Heap: a prioridade está com os maiores valores.
  - Min-Heap: a prioridade está com os menores valores.

Heaps

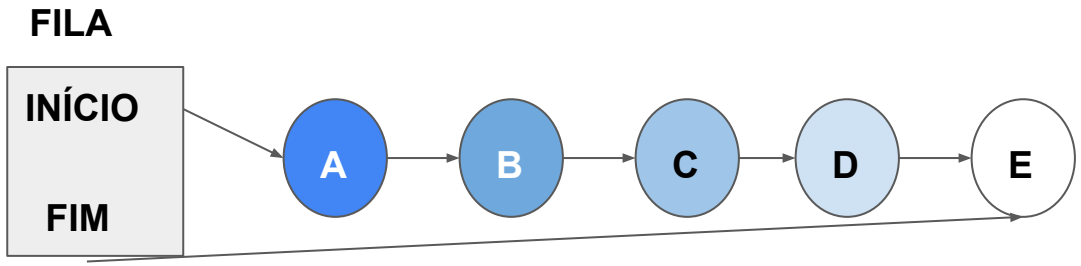
- A ideia de organização dos elementos na Heap, envolve organizar os elementos em uma estrutura conhecida como árvore binária, ou alternativamente, heap binária (*binary heap*).
  - Cada elemento é um nó dessa estrutura de dados;
  - Porém, não são organizados (conectados) linearmente como é feito em uma lista ou fila linear.

Heaps

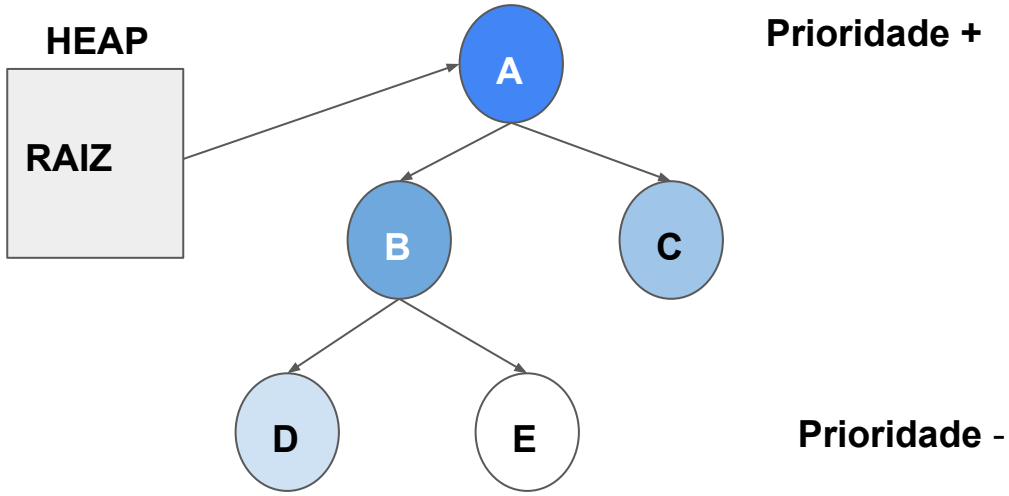
- A organização segue um padrão de árvore binária em que um elemento A pode estar conectado a mais dois outros elementos desde que ambos sejam menores ou maiores que A.

Heap Binária vs Fila Ligada

- Na Fila ligada os nós estão organizados como:

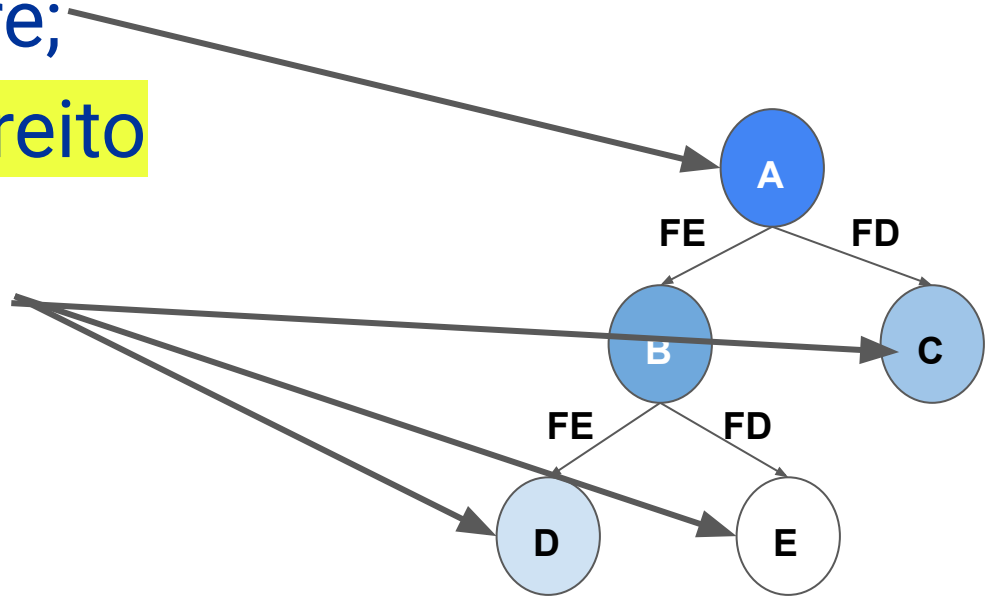


- Na Heap binária os nós estão organizados como:



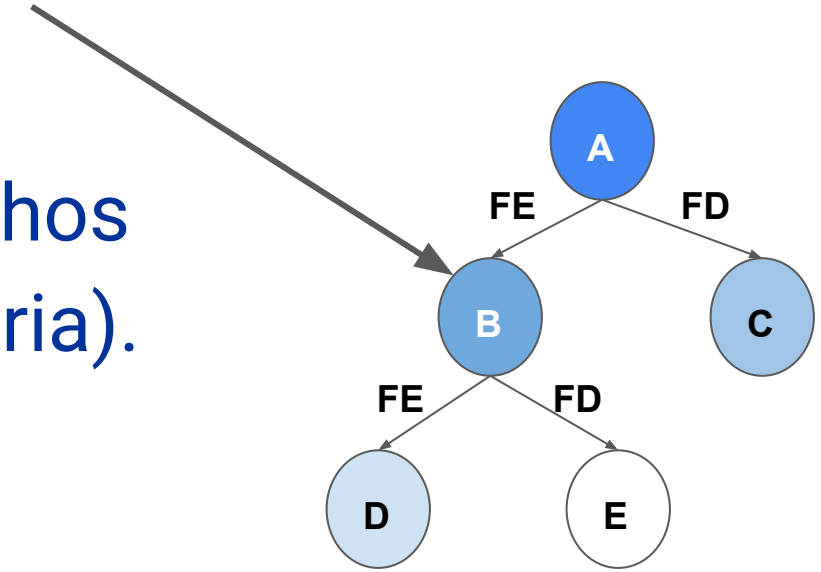
Árvore Binária (definições)

- Na árvore binária temos:
  - Nó raiz no topo da árvore;
  - Filho esquerdo (FE) e direito (FD) de um nó;
  - Nós folhas (sem filhos);



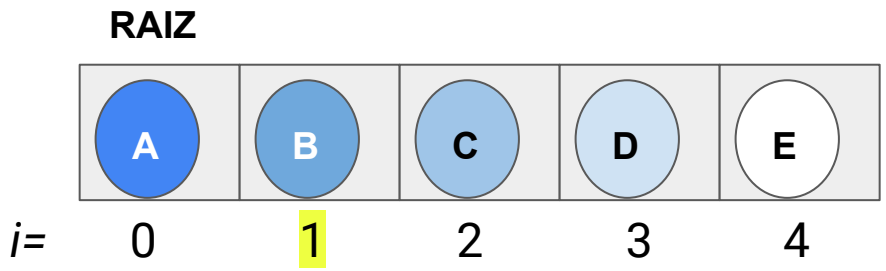
Árvore Binária (definições)

- Na árvore binária temos:
  - Nós internos (nem raiz, nem folha);
  - Cada nó tem no máximo 2 filhos (fator de ramificação=2, binária).



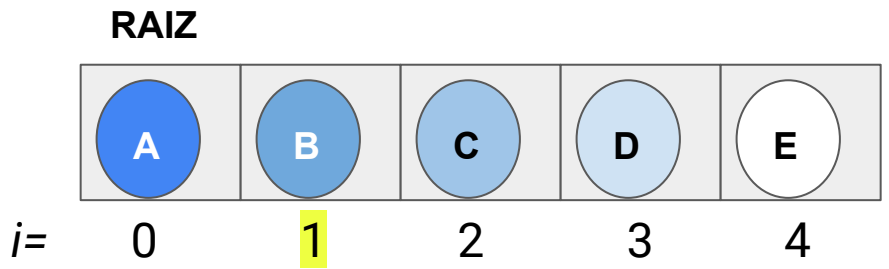
Heap Binária (Array/Vetor)

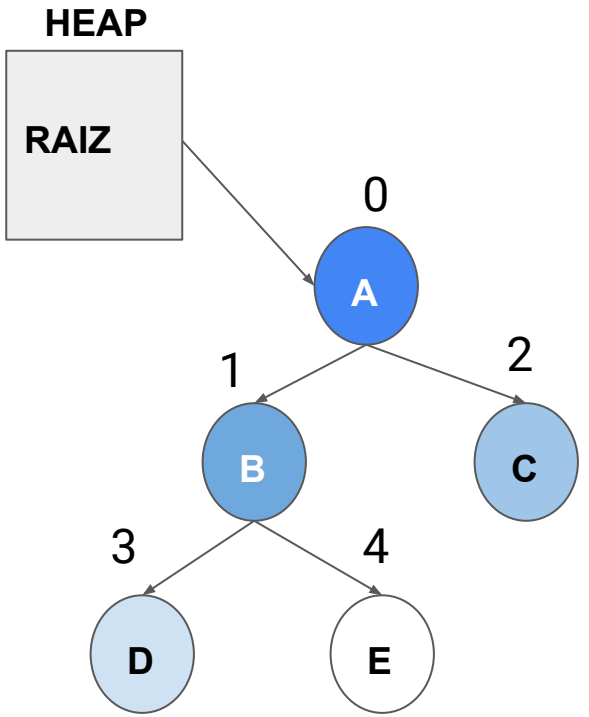
- Podemos também representar essa árvore binária quase completa por meio de vetor/array, mas por que?
  - Lembra da Hash? Acesso direto, O(1);



Heap Binária (Array/Vetor)

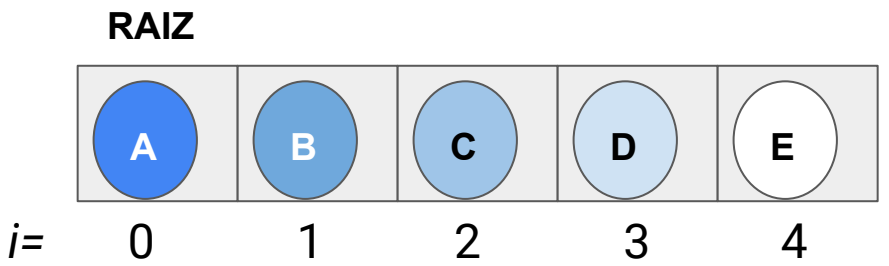
- Um nó na posição *i* do vetor tem filhos nas posições FE=2*i+1 e FD=2*i + 2 e ancestral em (i-1)/2; Veja para o nó B, i=1.





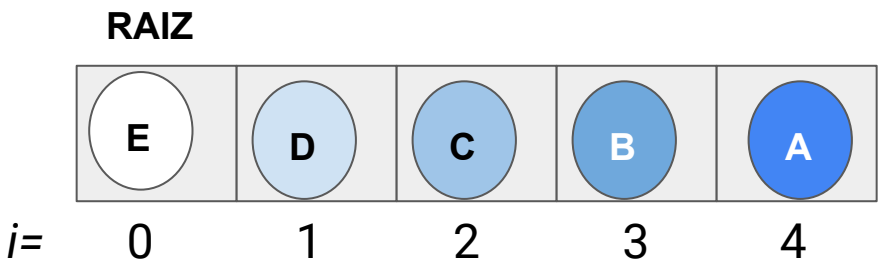
Heap Binária (Array/Vetor)

- Agora que sabemos como encontrar os elementos da árvore no vetor, temos que definir algumas restrições de ordem:
  - Prioridade do nó ancestral é ≥ que dos filhos (Max-Heap);



Heap Binária (Array/Vetor)

- Agora que sabemos como encontrar os elementos da árvore no vetor, temos que definir algumas restrições de ordem:
  - Prioridade do nó ancestral é ≤ que dos filhos (Min-Heap);

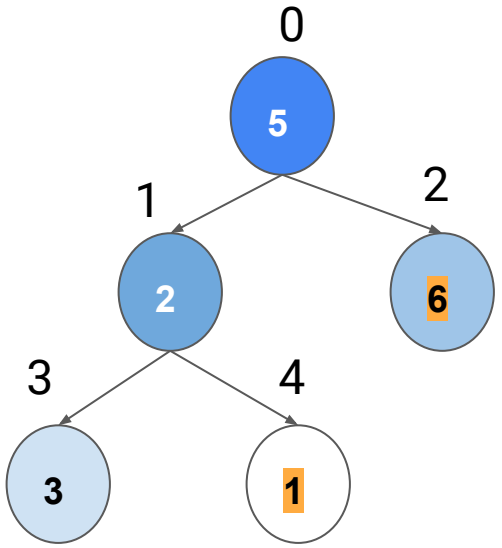


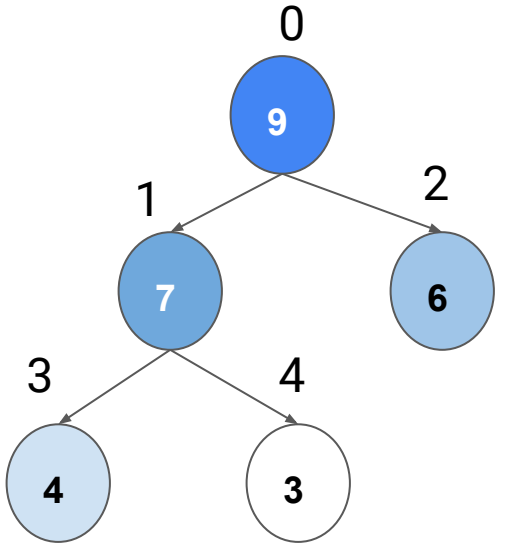
Heap Binária (Exemplos)

- Analise os 2 vetores abaixo e veja qual respeita as propriedades da Heap:

5	2	6	3	1
i= 0	1	2	3	4

9	7	6	4	3
i= 0	1	2	3	4





Heap Binária (Exemplos)

Quem é o pai de 6?

anc(6) = (2-1)/2 = 1/2 = 0; H[0] = 9.

Quem é o filho esquerdo de 6?

FE(6) = 2*2+1 = 5; Não tem H[5]!

Quem é o filho direito de 9?

FD(9) = 2*0+2 = 2; H[2]=6.

H	9	7	6	4	3
i=	0	1	2	3	4

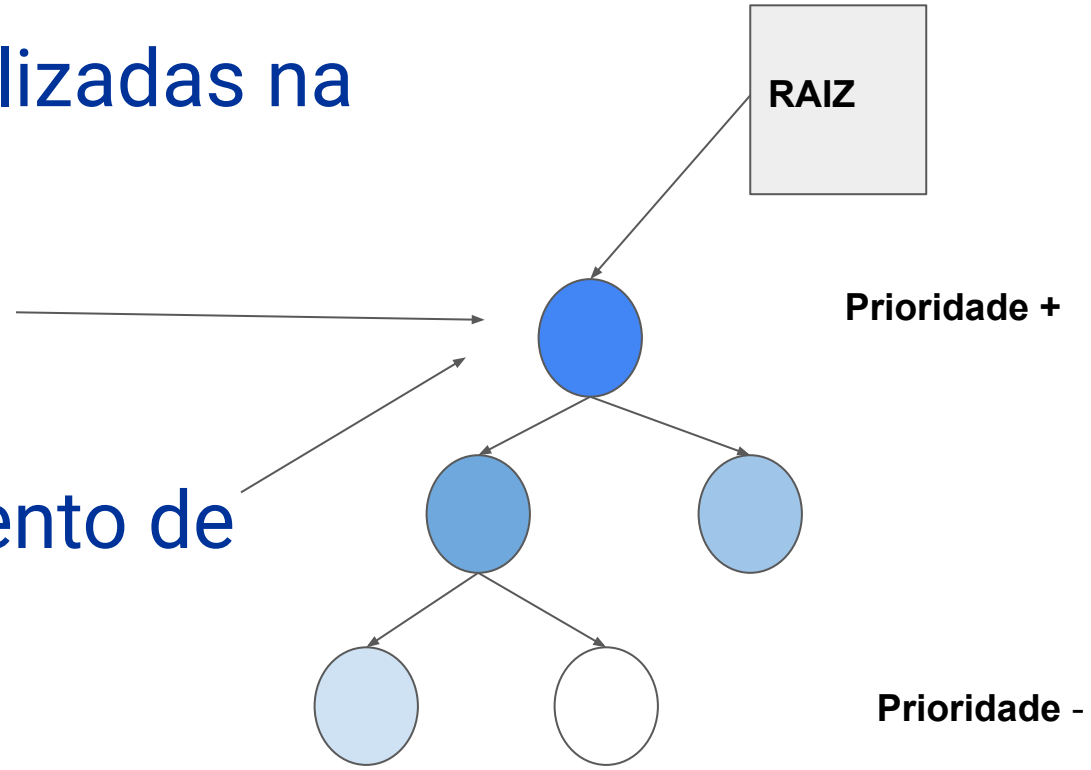
anc(*i*)= $\lfloor (i-1)/2 \rfloor$;

FE(*i*)=2**i*+1;

FD(*i*)=2**i*+2;

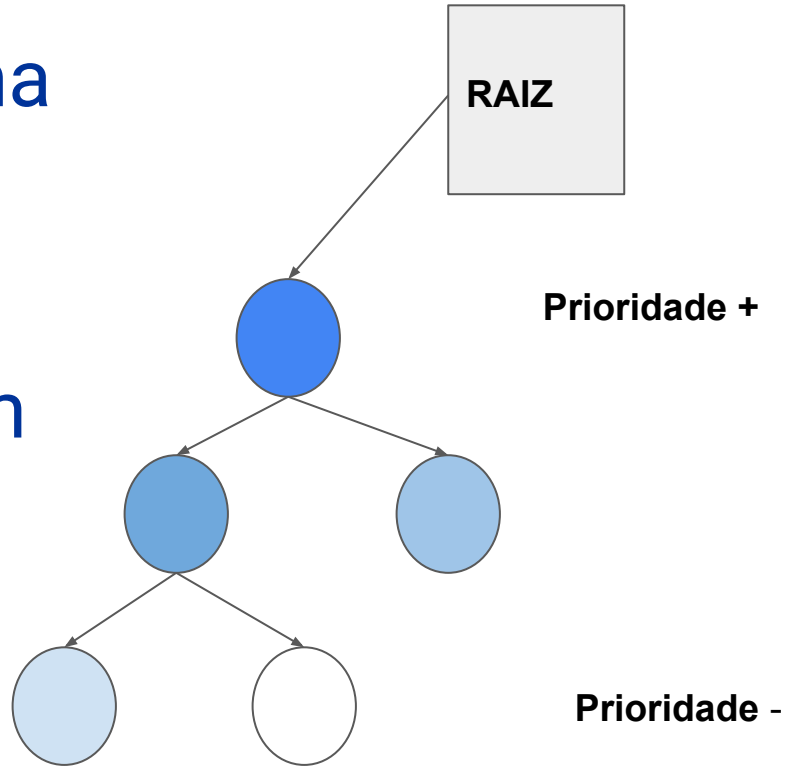
Heap Binária (Operações)

- Operações básicas a serem realizadas na heap binária:
  - max(): busca o elemento de prioridade +;
  - extrairMax(): remove o elemento de prioridade +;



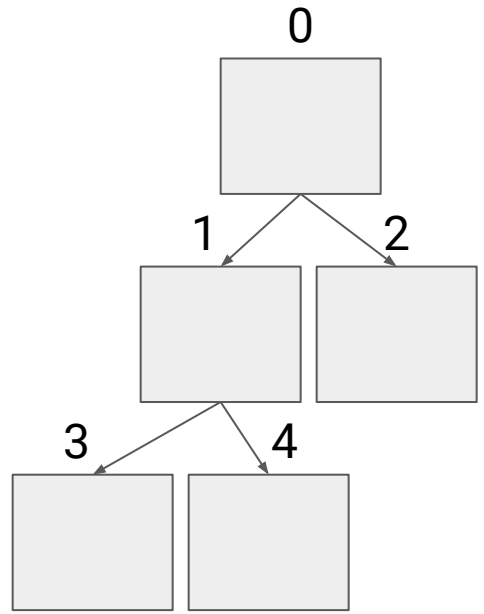
Heap Binária (Operações)

- Operações básicas a serem realizadas na heap binária:
  - max(), extrairMax();
  - inserir(x,p): insere um elemento x com prioridade p;
- Qual dessas operações é a mais trivial?

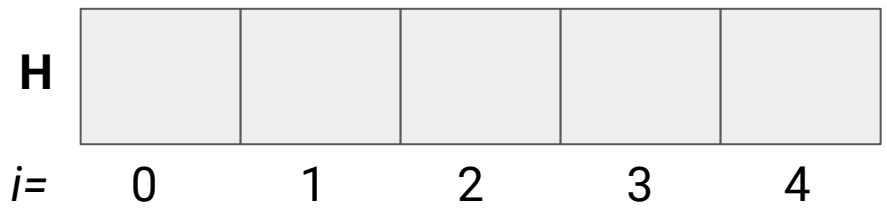


Heap Binária (Inserção - MaxHeap)

- Inserção na Max-Heap.
  - Iniciando com a Heap vazia, inserimos o primeiro valor na raiz (posição 0);
  - Os demais serão inseridos no fim do vetor e verificando as violações até a raiz;

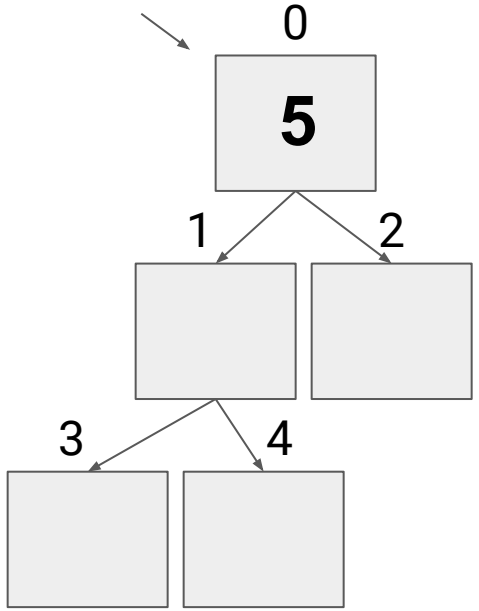


5 2 6 3 7

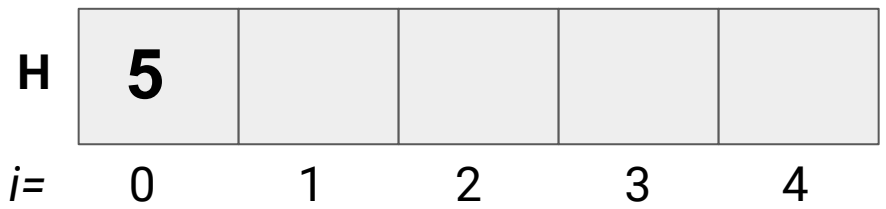


Heap Binária (Inserção - MaxHeap)

- Inserção na Max-Heap.
  - Iniciando com a Heap vazia, inserimos o primeiro valor na raiz (posição 0);
  - Os demais serão inseridos no fim do vetor e verificando as violações até a raiz;

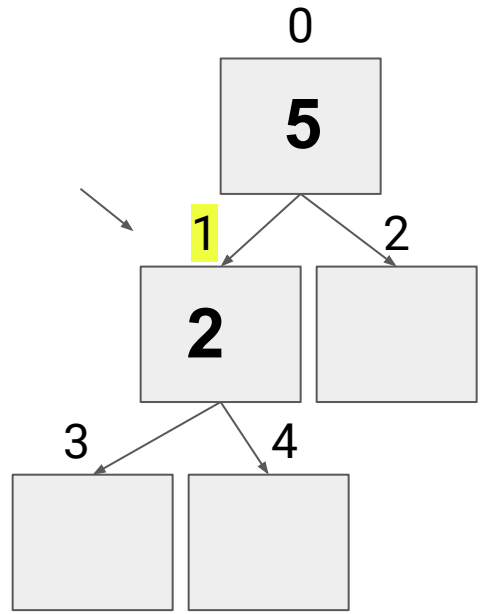


2 6 3 7

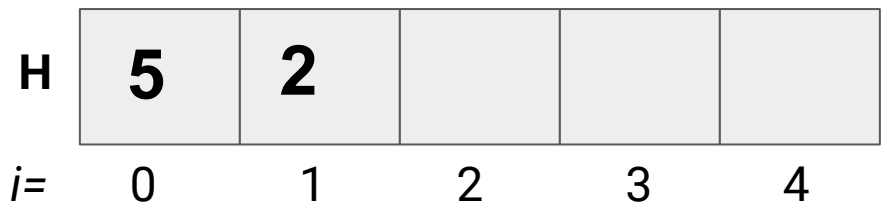


Heap Binária (Inserção - MaxHeap)

- Inserção na Max-Heap.
  - Iniciando com a Heap vazia, inserimos o primeiro valor na raiz (posição 0);
  - Os demais serão inseridos no fim do vetor e verificando as violações até a raiz;



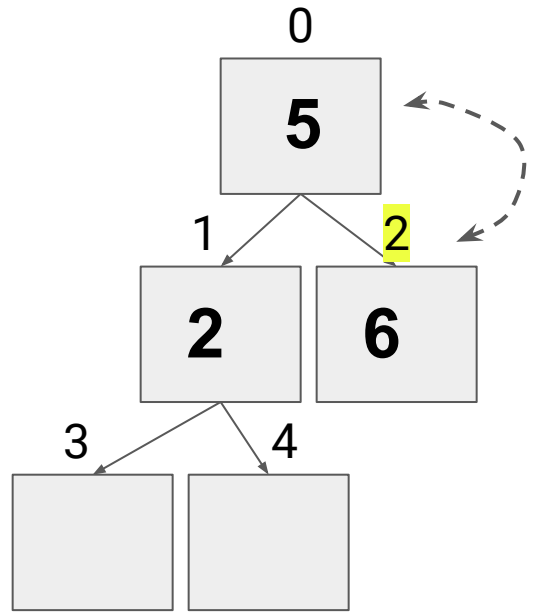
6 3 7



anc(1)=(1-1)/2)=0
H[0] > H[1] ? Sim

Heap Binária (Inserção - MaxHeap)

- Inserção na Max-Heap.
  - Iniciando com a Heap vazia, inserimos o primeiro valor na raiz (posição 0);
  - Os demais serão inseridos no fim do vetor e verificando as violações até a raiz;



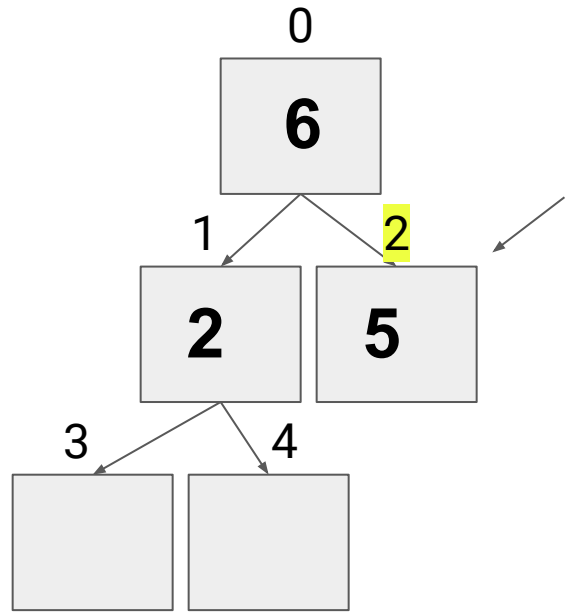
3 7

H	5	2	6		
i=	0	1	2	3	4

anc(2)=(2-1)/2)=0
H[0] > H[2] ? Não
trocar(H[2],H[0])

Heap Binária (Inserção - MaxHeap)

- Inserção na Max-Heap.
  - Iniciando com a Heap vazia, inserimos o primeiro valor na raiz (posição 0);
  - Os demais serão inseridos no fim do vetor e verificando as violações até a raiz;

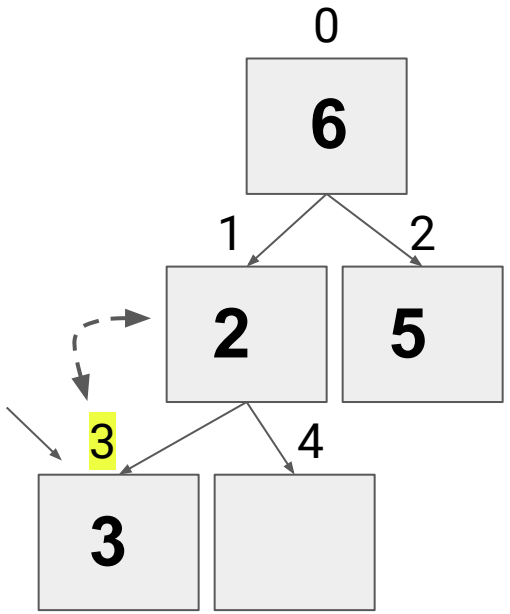


3 7

H	**6**	**2**	**5**		
i=	0	1	2	3	4

Heap Binária (Inserção - MaxHeap)

- Inserção na Max-Heap.
  - Iniciando com a Heap vazia, inserimos o primeiro valor na raiz (posição 0);
  - Os demais serão inseridos no fim do vetor e verificando as violações até a raiz;



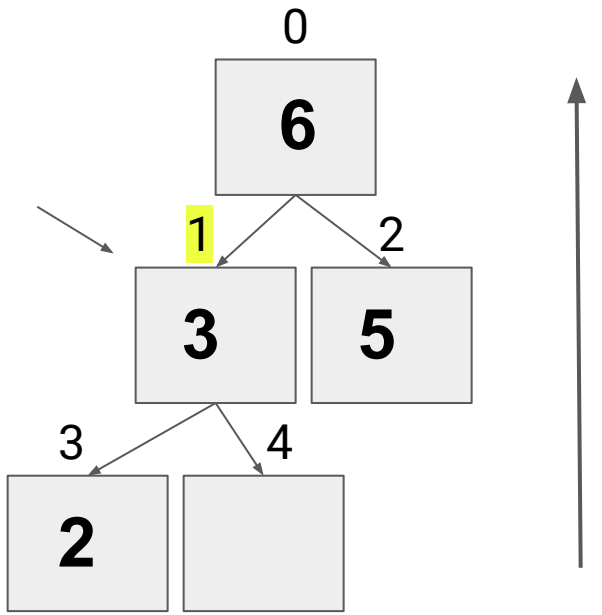
7

H	6	2	5	3	
i=	0	1	2	3	4

anc(3)=(3-1)/2)=1
H[1] > H[3] ? Não
trocar(H[3],H[1])

Heap Binária (Inserção - MaxHeap)

- Inserção na Max-Heap.
  - Iniciando com a Heap vazia, inserimos o primeiro valor na raiz (posição 0);
  - Os demais serão inseridos no fim do vetor e verificando as violações até a raiz;



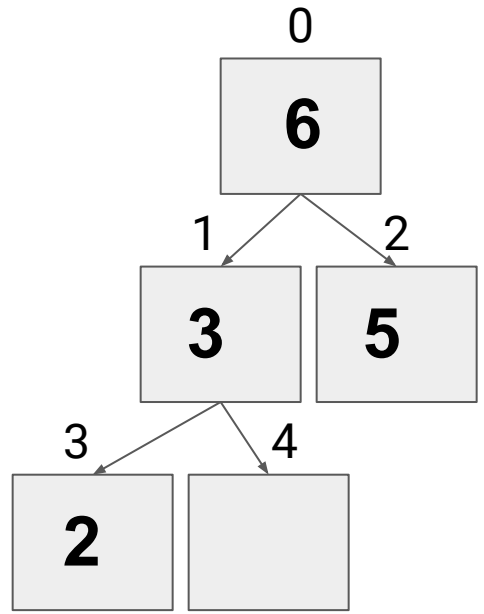
anc(1)=(1-1)/2)=0
H[0] > H[1] ? Sim

7

H	6	3	5	2	
i=	0	1	2	3	4

Heap Binária (Inserção - MaxHeap)

- Inserção na Max-Heap.
  - Iniciando com a Heap vazia, inserimos o primeiro valor na raiz (posição 0);
  - Os demais serão inseridos no fim do vetor e verificando as violações até a raiz;

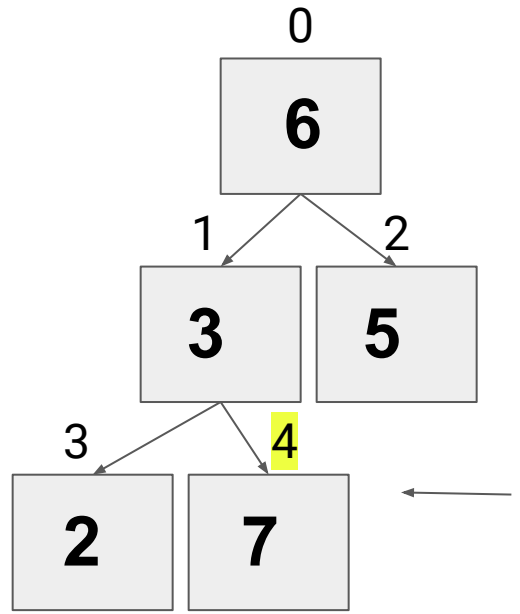


7

H	6	3	5	2	
i=	0	1	2	3	4

Heap Binária (Inserção - MaxHeap)

- Inserção na Max-Heap.
  - Iniciando com a Heap vazia, inserimos o primeiro valor na raiz (posição 0);
  - Os demais serão inseridos no fim do vetor e verificando as violações até a raiz;

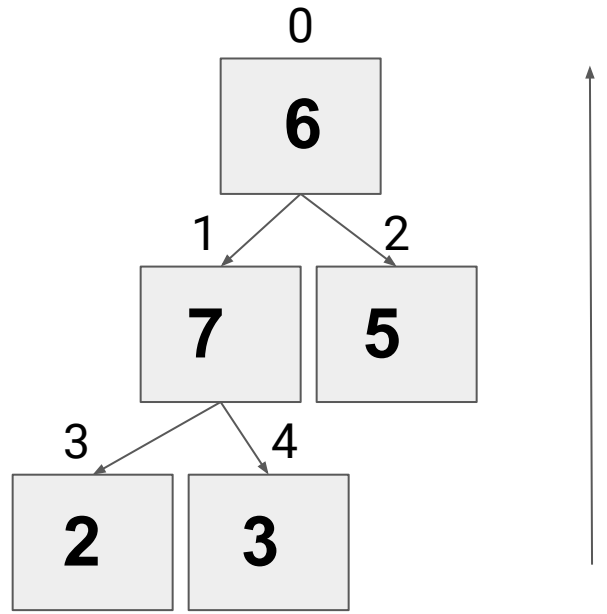


H	6	3	5	2	7
i=	0	1	2	3	4

anc(4)=(4-1)/2)=1
H[1] > H[4] ? Não
trocar(H[4],H[1])

Heap Binária (Inserção - MaxHeap)

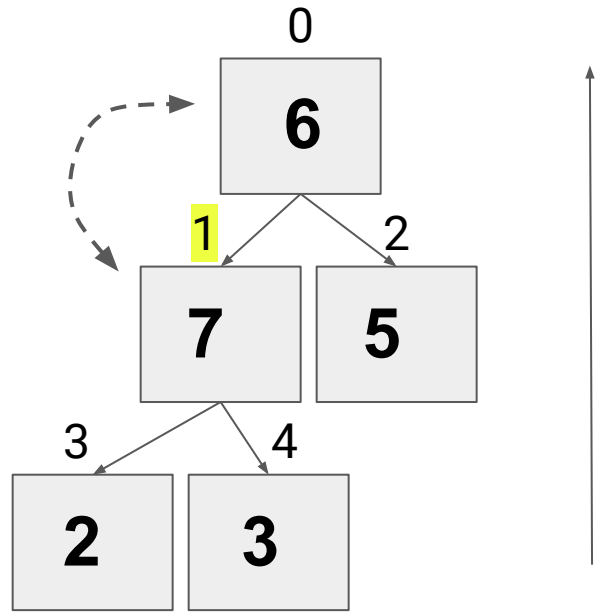
- Inserção na Max-Heap.
  - Iniciando com a Heap vazia, inserimos o primeiro valor na raiz (posição 0);
  - Os demais serão inseridos no fim do vetor e verificando as violações até a raiz;



H	6	7	5	2	3
i=	0	1	2	3	4

Heap Binária (Inserção - MaxHeap)

- Inserção na Max-Heap.
  - Iniciando com a Heap vazia, inserimos o primeiro valor na raiz (posição 0);
  - Os demais serão inseridos no fim do vetor e verificando as violações até a raiz;

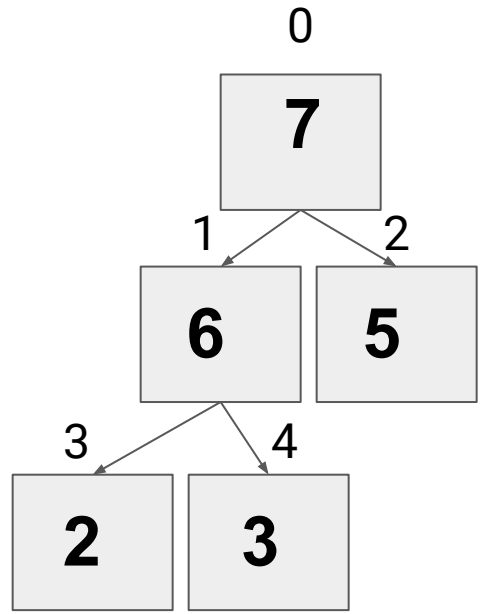


H	7	6	5	2	3
i=	0	1	2	3	4

anc(1)=(1-1)/2)=0
H[0] > H[1] ? Não
trocar(H[1],H[0])

Heap Binária (Inserção - MaxHeap)

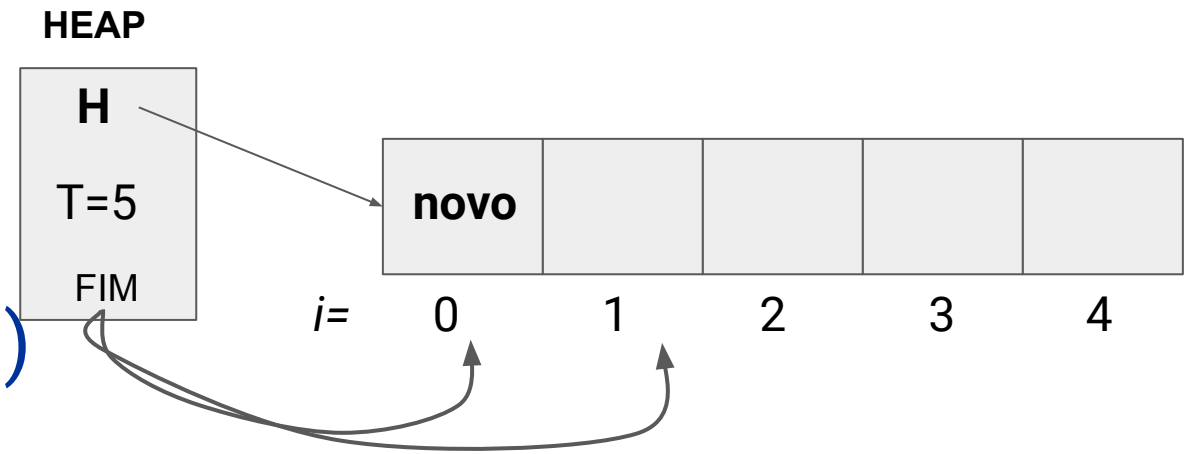
- Inserção na Max-Heap.
  - Iniciando com a Heap vazia, inserimos o primeiro valor na raiz (posição 0);
  - Os demais serão inseridos no fim do vetor e verificando as violações até a raiz;



H	7	6	5	2	3
i=	0	1	2	3	4

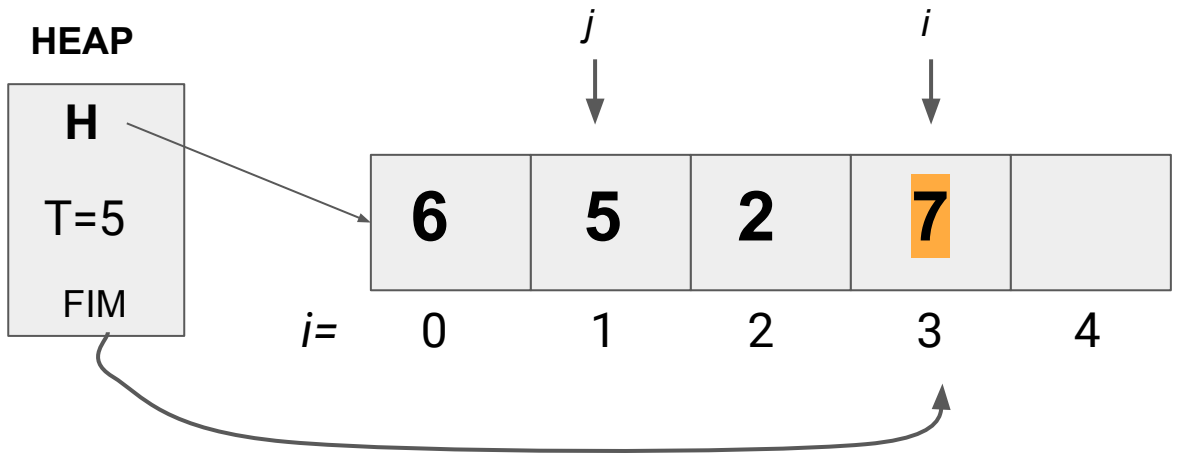
Heap Binária (Inserção - MaxHeap)

- Inserir(**H**, novo):
  - T = tamanho(**H**)
  - **SE** FIM < T **Então**
    - **H**[FIM] = novo
    - VerificarInsercao(**H**,FIM)
    - FIM = FIM+1
  - **Senão**
    - “MaxHeap Cheia!”



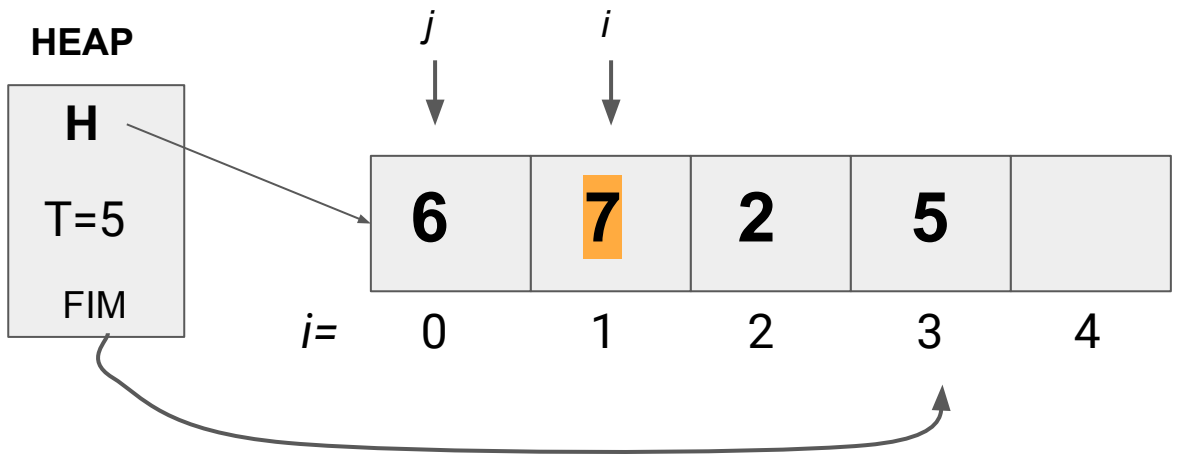
Heap Binária (Inserção - MaxHeap)

- VerificarInsercao(**H**, *i*):
  - *j* = anc(*i*) = ⌊(*i*-1)/2⌋
  - **Se** *i* > 0 **Então**
    - **Se H[*i*] > H[*j*] Então**
      - trocar(**H[*i*],H[*j*]**)
      - VerificarInsercao(**H**,*j*)



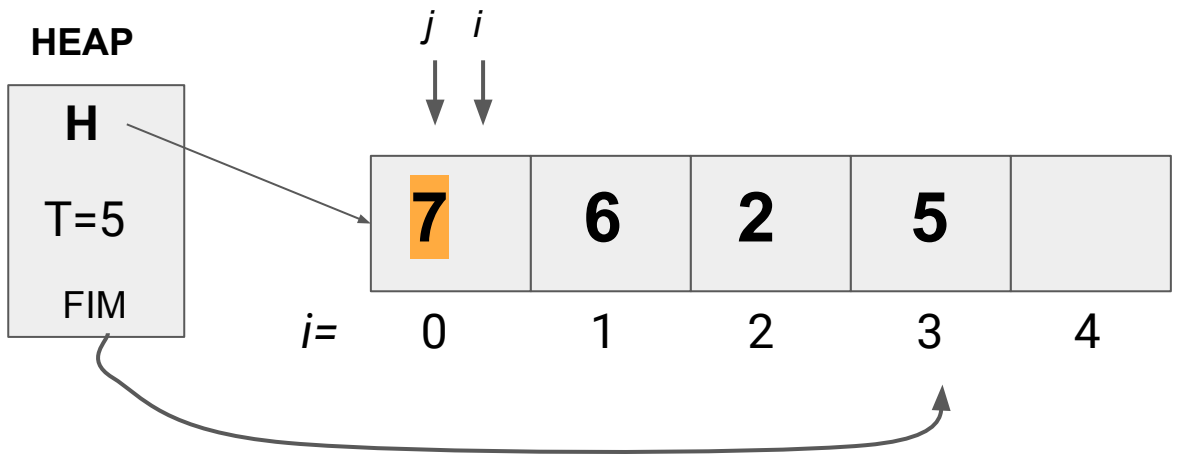
Heap Binária (Inserção - MaxHeap)

- VerificarInsercao(**H**, *i*):
  - *j* = anc(*i*) = ⌊(*i*-1)/2⌋
  - **Se** *i* > 0 **Então**
    - **Se H[*i*] > H[*j*] Então**
      - trocar(**H[*i*],H[*j*]**)
      - VerificarInsercao(**H**,*j*)



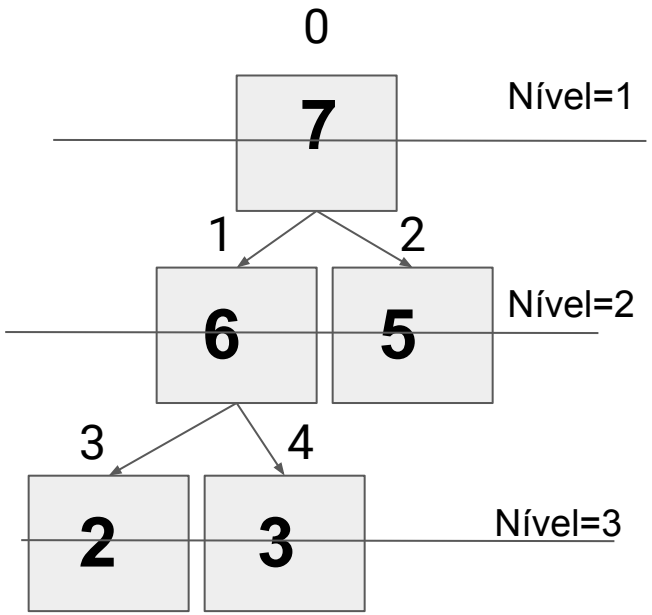
Heap Binária (Inserção - MaxHeap)

- VerificarInsercao(**H**, *i*):
  - *j* = anc(*i*) = ⌊(*i*-1)/2⌋
  - **Se** *i* > 0 **Então**
    - **Se H[*i*] > H[*j*] Então**
      - trocar(**H[*i*],H[*j*]**)
      - VerificarInsercao(**H**,*j*)



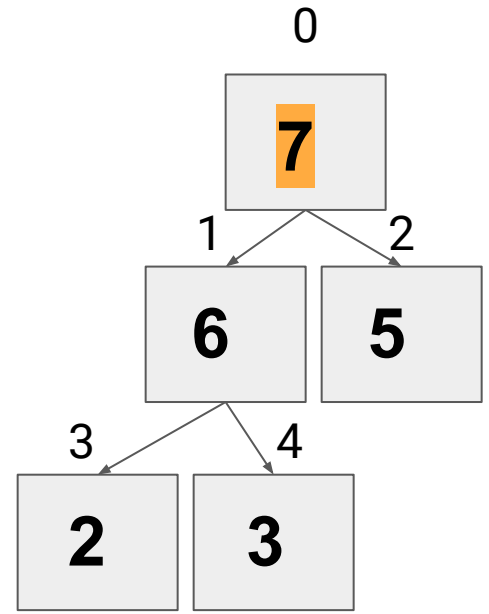
Heap Binária (Inserção - MaxHeap)

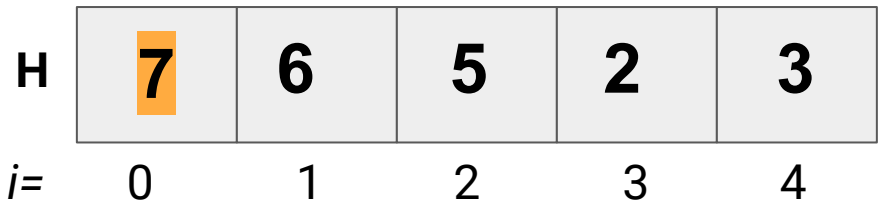
- Inserção na Max-Heap.
  - Inserir em uma posição no vetor H tem custo O(1);
  - Verificar violações até a raiz tem custo proporcional a altura da árvore.
    - Uma heap binária com N nós tem $\lfloor \log_2(N) \rfloor + 1$ níveis.
  - Logo o custo da inserção é $O(\log_2(N))$.



Heap Binária (Remoção)

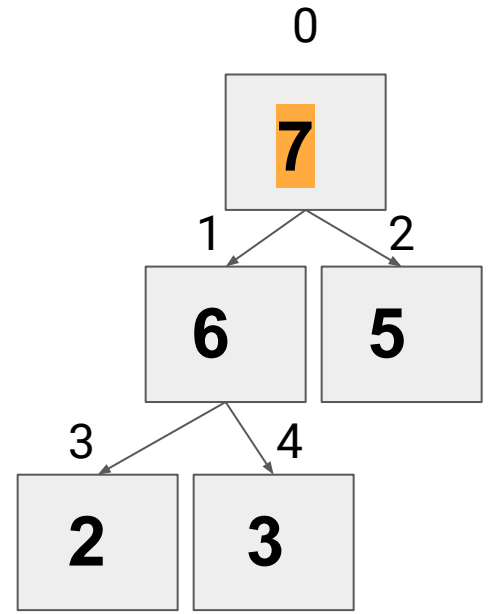
- ExtrairMax ou Remoção na Max-Heap.
  - Retirar o elemento máximo da raiz;
  - Preencher o espaço vazio na raiz (posição 0) com o último elemento do vetor.

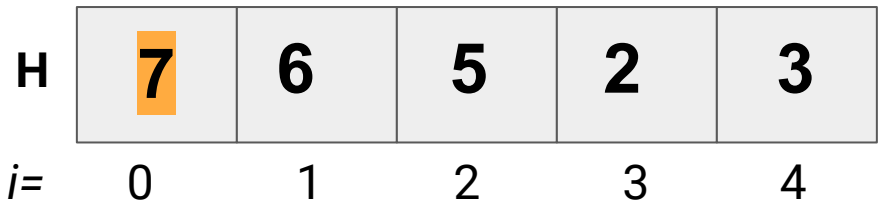




Heap Binária (Remoção)

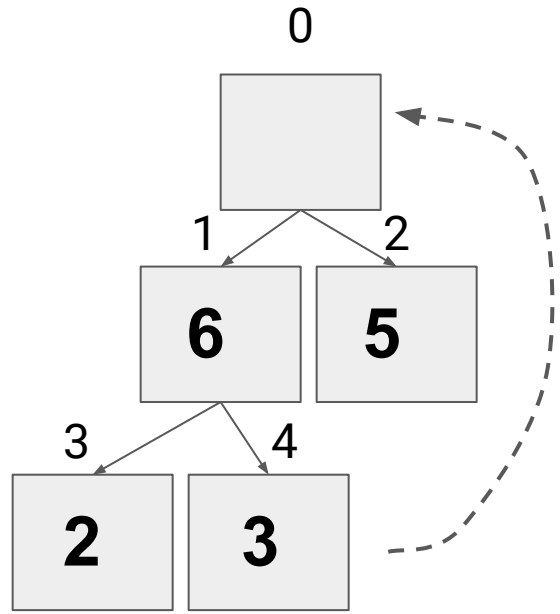
- ExtrairMax ou Remoção na Max-Heap.
  - Verificando da raiz até as folhas as violações:
    - O maior entre: ancestral, FE e FD e trocar pelo ancestral;
    - Descer a árvore na direção do maior elemento trocado.

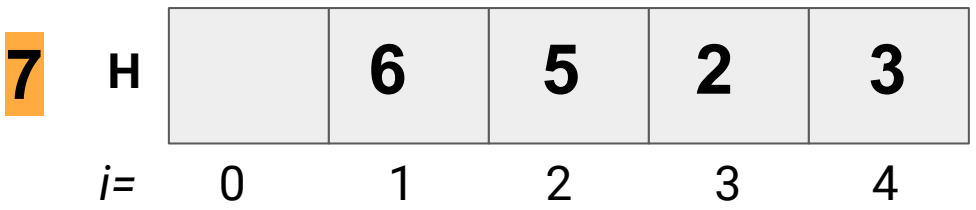




Heap Binária (Remoção)

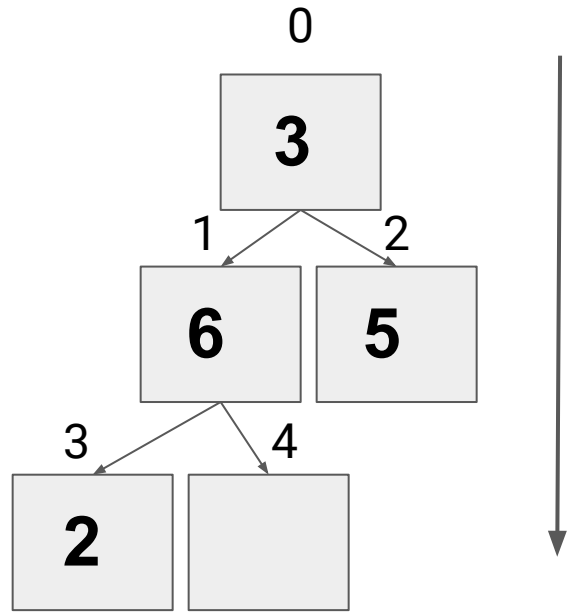
- ExtrairMax ou Remoção na Max-Heap.
  - Retirar o elemento máximo da raiz;
  - Preencher o espaço vazio na raiz (posição 0) com o último elemento do vetor;
  - Verificando da raiz até as folhas as violações.

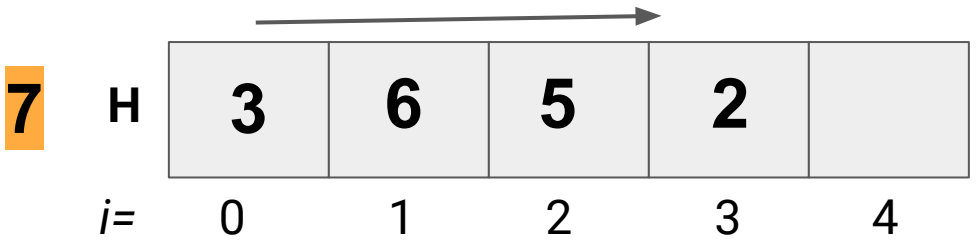




Heap Binária (Remoção)

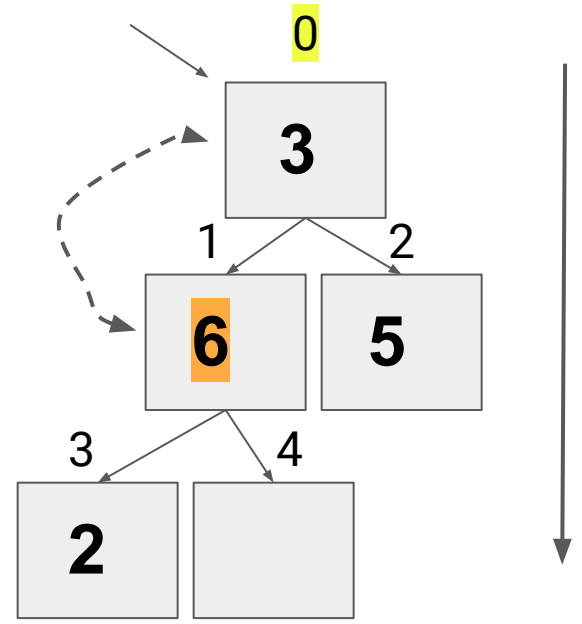
- ExtrairMax ou Remoção na Max-Heap.
  - Retirar o elemento máximo da raiz;
  - Preencher o espaço vazio na raiz (posição 0) com o último elemento do vetor;
  - Verificando da raiz até as folhas as violações.





Heap Binária (Remoção)

- ExtrairMax ou Remoção na Max-Heap.
  - Verificando da raiz até as folhas as violações.

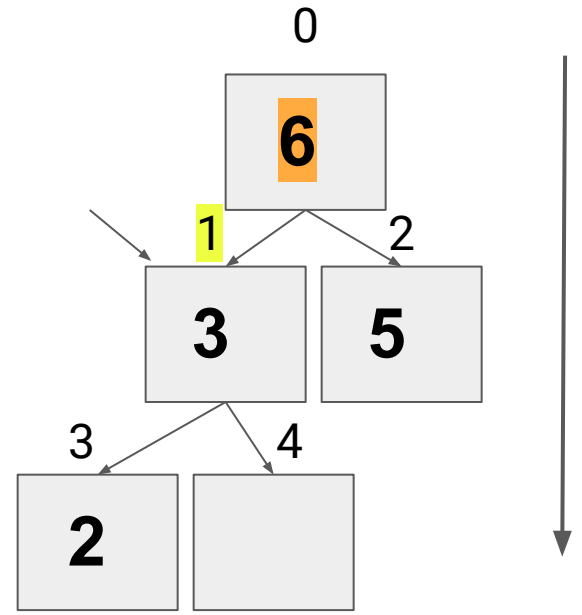


H	3	6	5	2	
i=	0	1	2	3	4

FE(0)=(2*0+1)=1
FD(0)=(2*0+2)=2
maior(H[FE],H[FD])=6
H[0] > 6 ? Não

Heap Binária (Remoção)

- ExtrairMax ou Remoção na Max-Heap.
  - Verificando da raiz até as folhas as violações.

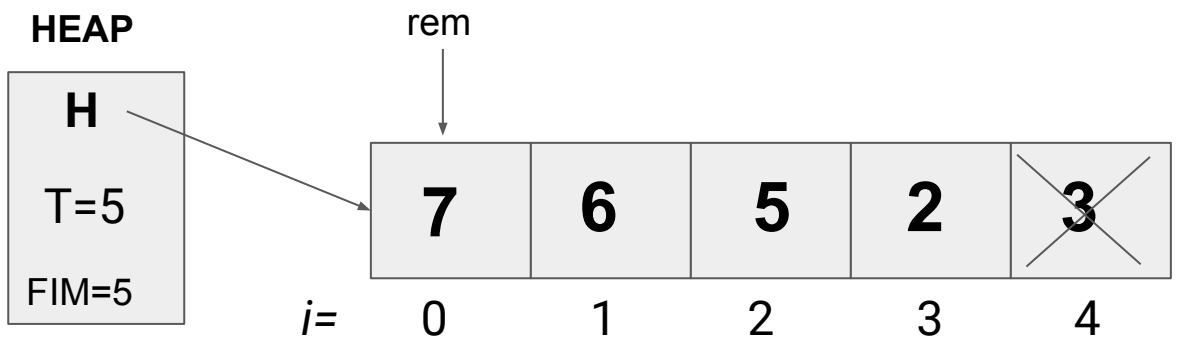


H	6	3	5	2	
i=	0	1	2	3	4

FE(1)=(2*1+1)=3
FD(1)=(2*1+2)=4
maior(H[FE],H[FD])=2
H[1] > 2 ? Sim

Heap Binária (Remoção - MaxHeap)

- ExtrairMax(**H**):
  - rem = **H**[0]
  - **H**[0] = **H**[FIM-1]
  - FIM = FIM-1
    - VerificarRemocao(**H**,0)



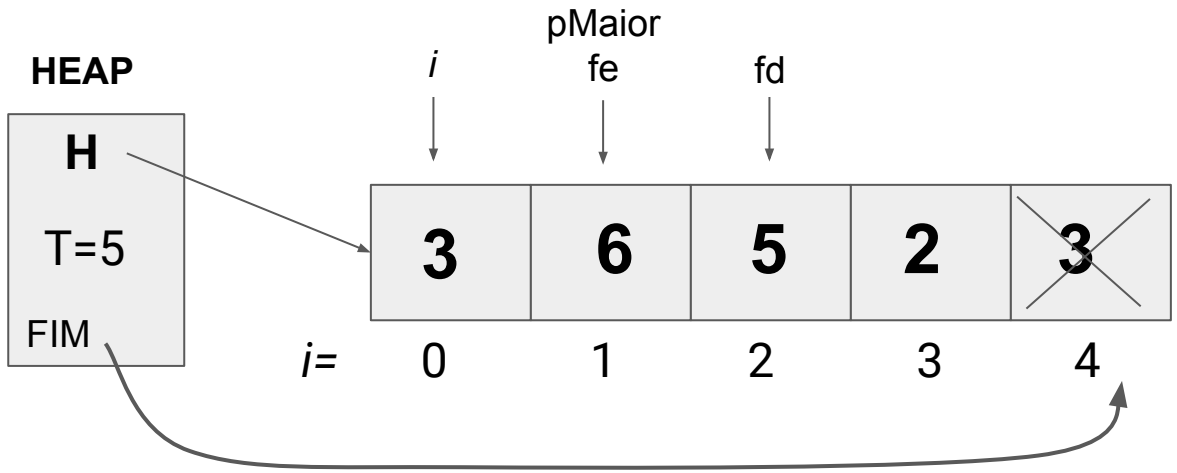
Heap Binária (Remoção - MaxHeap)

- VerificarRemocao(**H,***i*):
  - fe = esq(*i*), fd = fe+1
  - pMaior = EncMaior(**H**,i,fe,fd)
  - **Se** pMaior ≠ *i* **Então**
    - trocar(**H**[*i*], **H**[pMaior])
    - VerificarRemocao(**H**, pMaior)

- EncMaior(**H,***i,fe,fd*):
  - **Se** fe<T **e H**[fe] > **H**[*i*] **Então**
    - pMaior=fe
    - **Senão** pMaior=*i*
  - **Se** fd<T **e H**[fd] > **H**[pMaior] **Então**
    - pMaior=fd
  - **retornar** pMaior

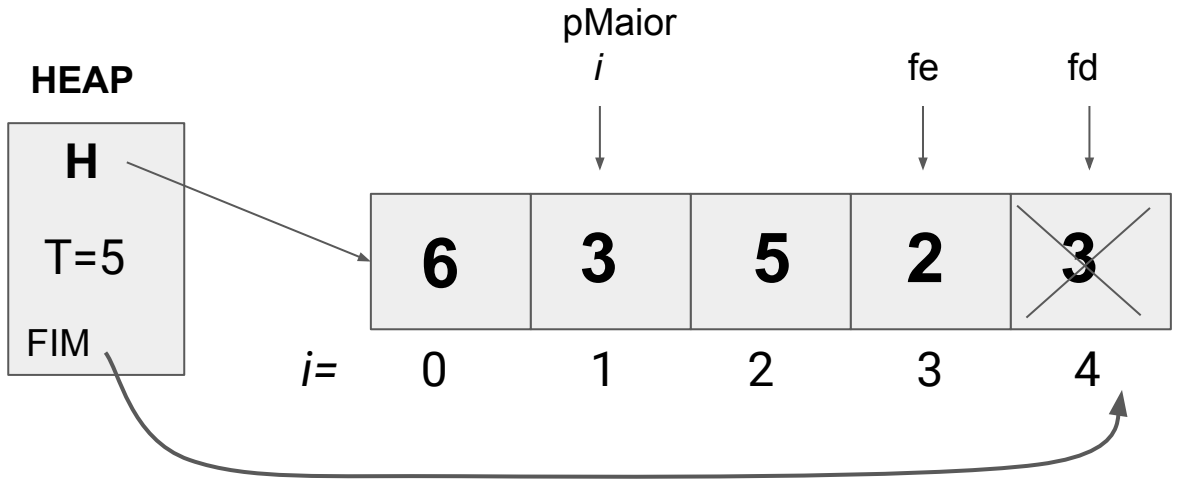
Heap Binária (Remoção - MaxHeap)

- VerificarRemocao(**H**,*i*):
  - fe = esq(*i*), fd = fe+1
  - pMaior = EncMaior(**H**,i,fe,fd)
  - **Se** pMaior ≠ *i* **Então**
    - trocar(**H**[*i*], **H**[pMaior])
    - VerificarRemocao(**H**, pMaior)



Heap Binária (Remoção - MaxHeap)

- VerificarRemocao(**H**,*i*):
  - fe = esq(*i*), fd = fe+1
  - pMaior = EncMaior(**H**,i,fe,fd)
  - **Se** pMaior ≠ *i* **Então**
    - trocar(**H**[*i*], **H**[pMaior])
    - VerificarRemocao(**H**, pMaior)



Heap Binária (Remoção)

- ExtrairMax ou Remoção na Max-Heap.
  - Custo de retirar o maior elemento na raiz é O(1);
  - Porém, descer a árvore e ir verificando as violações depende do número de níveis da árvore (como na inserção);
  - Portanto, o custo da remoção de um elemento é $O(\log_2(N))$.

Referências

- SZWARCFITER, Jayme Luiz; MARKENZON, Lilian. **Estruturas de dados e seus algoritmos**. 3. ed. Rio de Janeiro, RJ: LTC, 2010. ISBN 9788521629955.
- CORMEN, Thomas. **Algoritmos: teoria e prática**. Rio de Janeiro: GEN LTC, 2013. ISBN 9788595158092.

Licenciamento