Fila de Prioridades e Heaps

Prof.: Leonardo Tórtoro Pereira
leonardop@usp.br

Baseado nos slides do Prof. Rudinei Goularte

Conteúdo

- TAD Fila de Prioridade
- Heaps
- Implementação em Arranjo

Fila de Prioridade

- Relembrando:
  - **Pilhas e filas** => Elementos processados em função da sequência na qual foram inseridos
- **Filas de Prioridade**
  - Elementos são processados de acordo com sua importância, independente do momento em que entraram na fila

Fila de Prioridade

- **Filas de Prioridade**
  - Exemplos:
    - atendimento preferencial em estabelecimentos em geral
    - importância de processos em sistemas operacionais
    - substituição de páginas menos utilizadas na memória
    - filas de atendimento para análise de sinais em sistemas de controle (sensores prioritários)

TAD Fila de Prioridade

- Armazena Itens
- Item: par (chave, informação)
- Operações principais
  - desenfileirar(F): remove e retorna o item com maior (menor) prioridade da fila F
  - enfileirar(F, x): insere um item x = (k,i) com chave k
- Operações auxiliares
  - proximo(F): retorna o item com maior (menor) chave da fila F, sem removê-lo
  - vazia(F), cheia(F)

TAD Fila de Prioridade

- Diferentes Realizações (implementações)
  - Estáticas
    - Lista sequencial (arranjo) ordenada
    - Lista sequencial (arranjo) não ordenada
    - **Heap em arranjo**
  - Dinâmicas
    - Lista encadeada ordenada
    - Lista encadeada não ordenada
    - Heap encadeada
- Cada realização possui vantagens e desvantagens

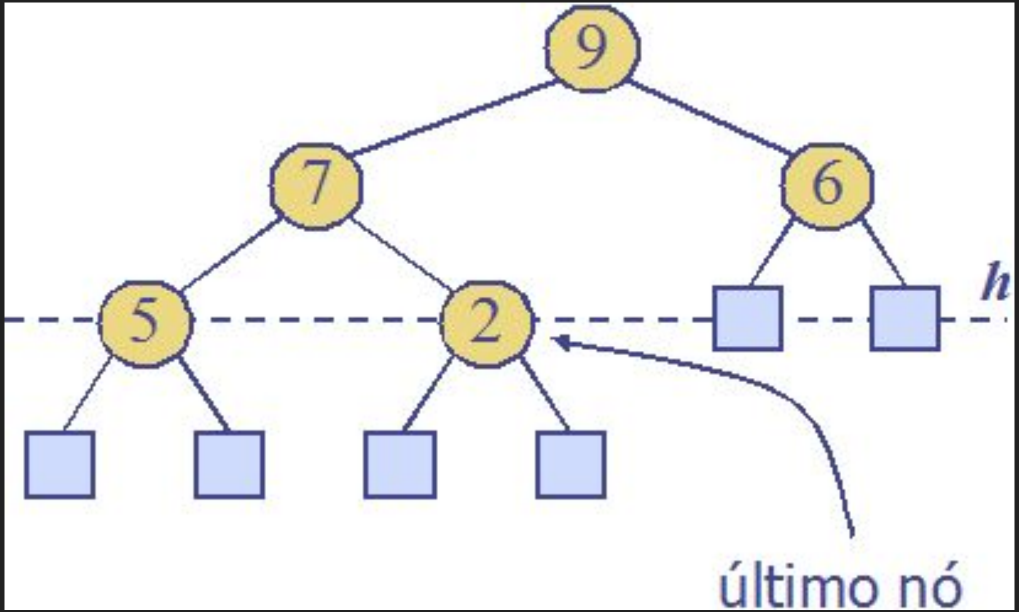
TAD Fila de Prioridade

- Uma das escolhas diretas seria usar uma fila ordenada
  - inserção é O(n)
  - remoção é O(1)
  - próximo é O(1)
- Outra seria usar uma fila não-ordenada
  - inserção é O(1)
  - remoção é O(n)
  - próximo é O(n)
- Portanto uma abordagem mais rápida precisa ser pensada quando grandes conjuntos de dados são considerados

Heaps

- Uma heap é uma árvore binária que satisfaz as propriedades
  - **Ordem**: para cada nó v, exceto o nó raiz, tem-se que
    - chave(v) < = chave(pai(v)) - heap máxima
    - chave(v) > = chave(pai(v)) - heap mínima

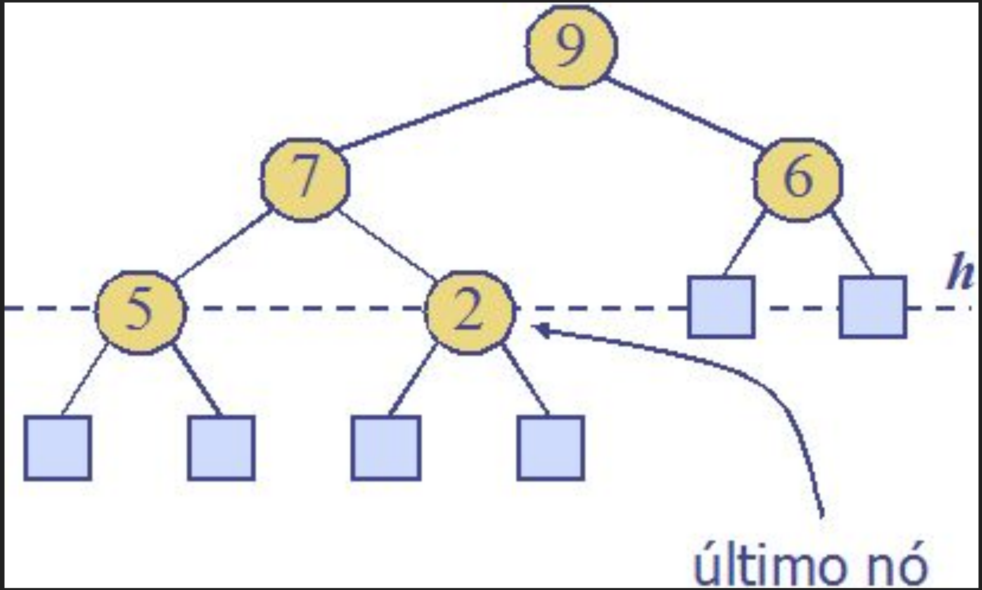
Heaps



Heaps

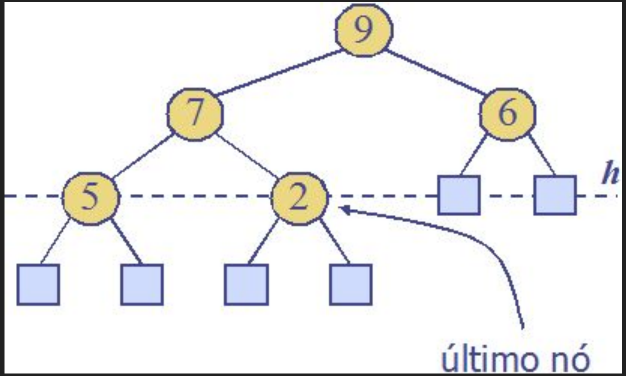
- Uma heap é uma árvore binária que satisfaz as propriedades
  - **Completude:** é completa, i.e., se h é a altura
    - Todo nó folha está no nível h ou h - 1
    - O nível h - 1 está totalmente preenchido
    - As folhas do nível h estão todas mais a esquerda

Heaps



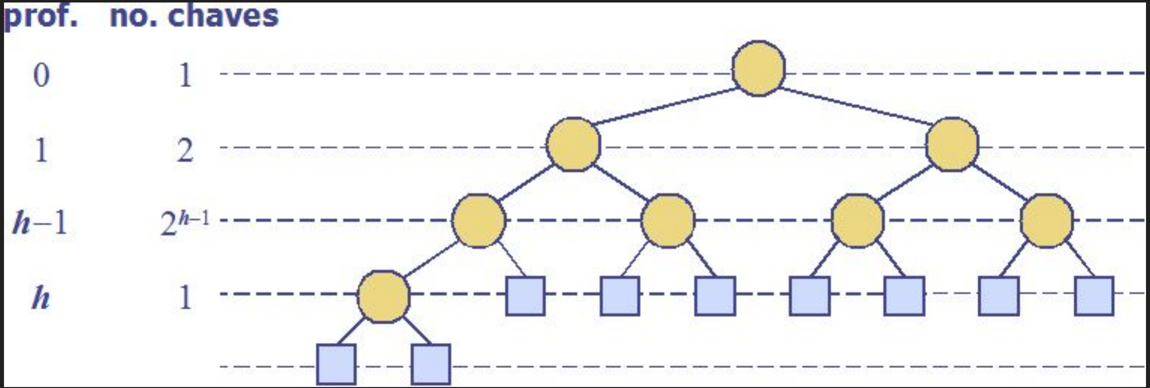
Heaps

- Convenciona-se aqui
  - Último nó: nó interno mais à direita de profundidade h



Altura de uma Heap

- Teorema
  - Uma heap armazenando n nós possui altura h de ordem O(log n).



Altura de uma Heap

- Prova
  - Dado que existem 2^i chaves na profundidade $i = 0, \ldots, h-1$

  e ao menos 1 chave na profundidade h, tem-se:

$$n \geq 1 + 2 + 4 + \ldots + 2^{h-1} + 1$$

Altura de uma Heap

- Prova
- Isso é uma Progressão Geométrica (PG) com razão q = 2. Dado que a soma de um PG pode ser calculada por

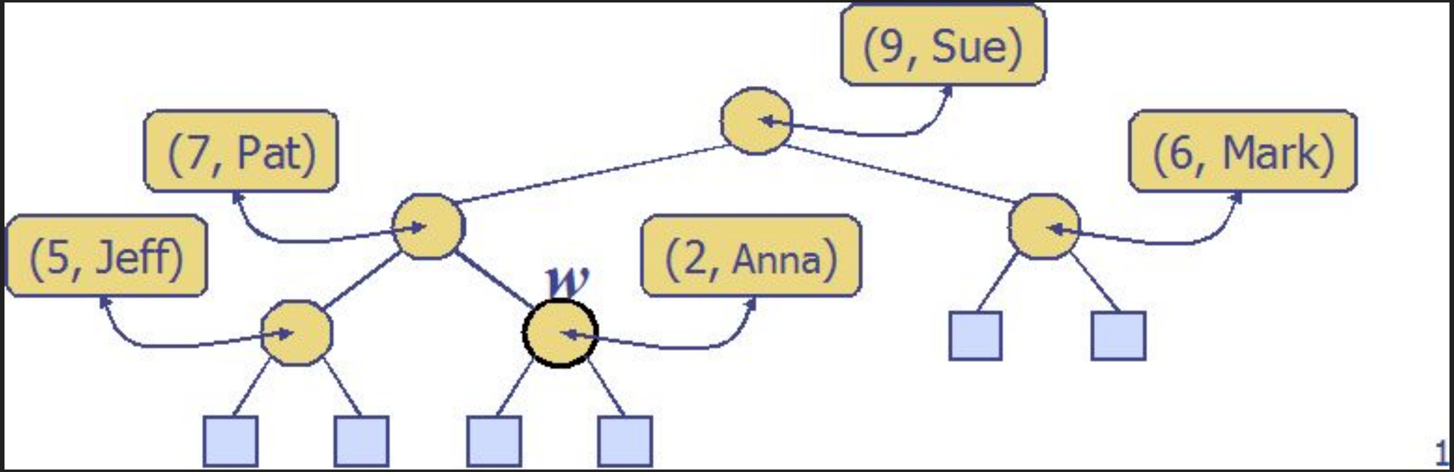
$$S_k = \frac{a^k \times q - a_1}{q-1}$$, temos $$n \geq (2^{h-1} \times 2 - 1) + 1 = 2^h$$

- Logo, $n \geq 2^h$, i.e., $h \leq \log_2 n$ □ h é O(log n)

Filas de Prioridade com Heaps

- Armazena-se um Item (chave, informação) em cada nó
- Mantém-se o controle sobre a localização do último nó (w)
- Remove-se sempre o Item armazenado na raiz, devido à propriedade de ordem da heap
  - Heap mínima: menor chave na raiz da heap
  - Heap máxima: maior chave na raiz da heap

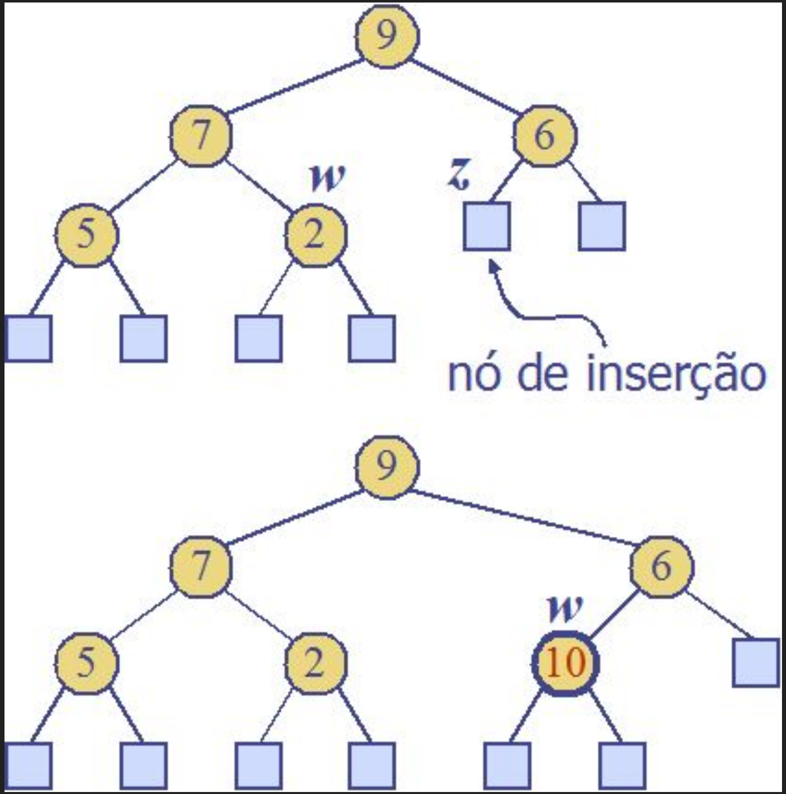
Filas de Prioridade com Heaps



Inserção

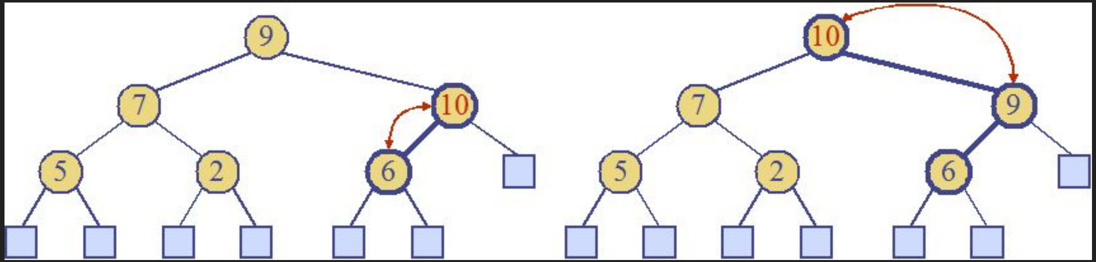
- Método insere do TAD fila de prioridade corresponde à inserção de um Item na heap
- O algoritmo consiste de 3 passos
  1. Encontrar e criar nó de inserção z (novo último nó depois de w)
  2. Armazenar o Item com chave k em z
  3. Restaurar ordem da heap (discutido a seguir)

Inserção



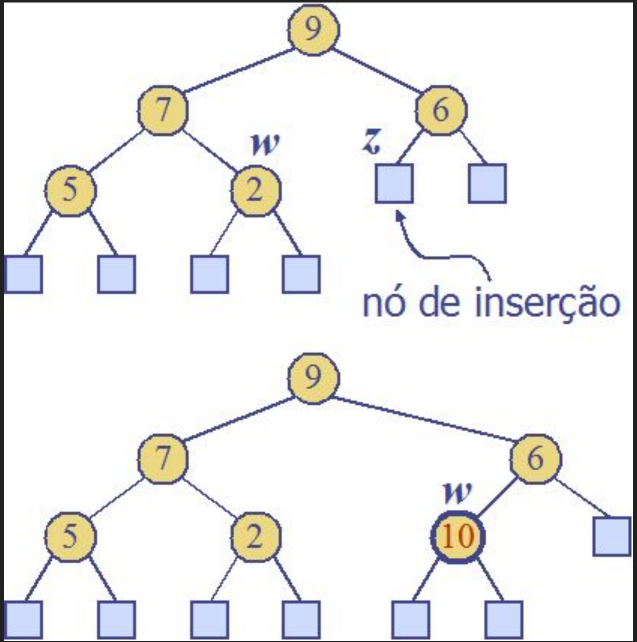
Restauração da Ordem (fix-up)

- Após a inserção de um novo Item, a propriedade de ordem da heap pode ser violada
- A ordem da heap é restaurada trocando os itens caminho acima a partir do nó de inserção
- Termina quando o Item inserido alcança a raiz ou um nó cujo pai possui uma chave maior (ou menor)



Inserção

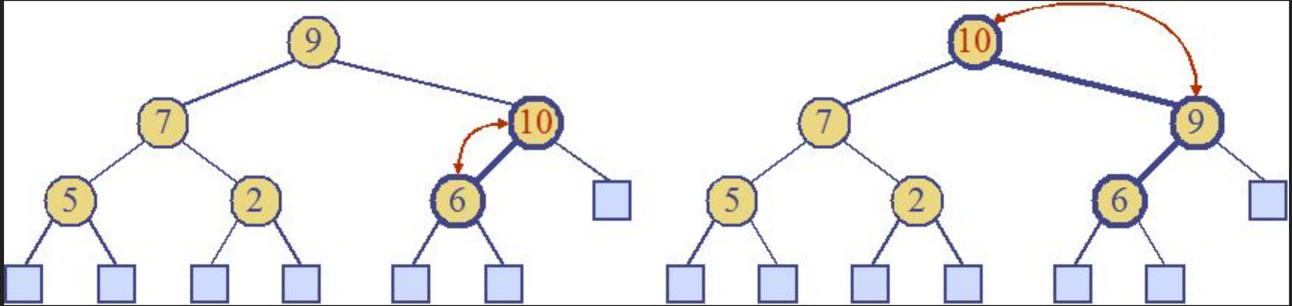
```
Algoritmo Inserir(F,x)
  inserirNoFim(F) //insere na última posição
  fix_up(F) //restaura ordem do heap
```



Restauração da Ordem (fix-up)

- Para uma heap máxima, temos

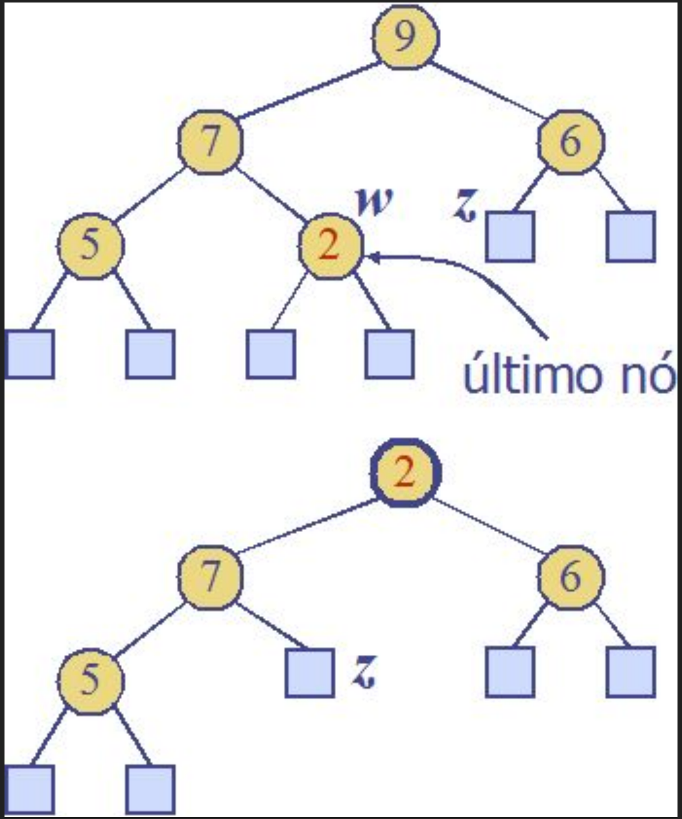
```
Algoritmo fix_up(F)
  w = F.ultimo
  while(!isRoot(F,w)) && (key(F,w) > key(F,parent(F,w))) {
    swap(F,w,parent(F,w))
    w = parent(F,w) //sobe
  }
```



Remoção

- Método remove do TAD fila de prioridade corresponde à remoção do Item da raiz
- O algoritmo de remoção consiste de 3 passos
  1. Armazenar o conteúdo do nó raiz da heap (para retorno)
  2. Copiar o conteúdo de w no nó raiz e remover o nó w
  3. Restaurar ordem da heap (discutido a seguir)

Remoção

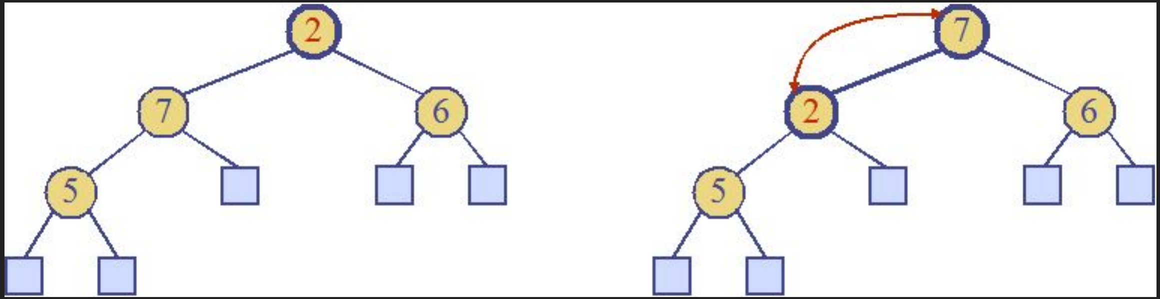


Remoção

- ➔ Utilizar o último nó para substituição da raiz na remoção possui várias vantagens, entre elas
  - ◆ Completude garantida (passo 2)
  - ◆ Implementação em tempo constante através de arranjo (discutida posteriormente)

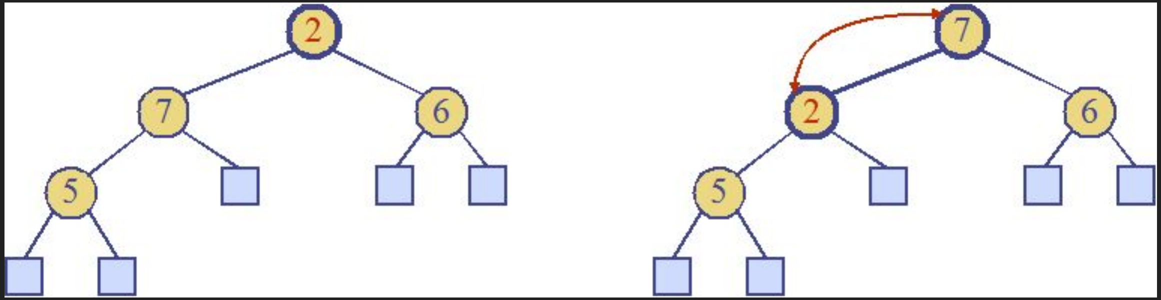
Restauração da Ordem (fix-down)

- Após a remoção, a propriedade de ordem da heap pode ser violada
- A ordem da heap é restaurada trocando os itens caminho abaixo a partir da raiz



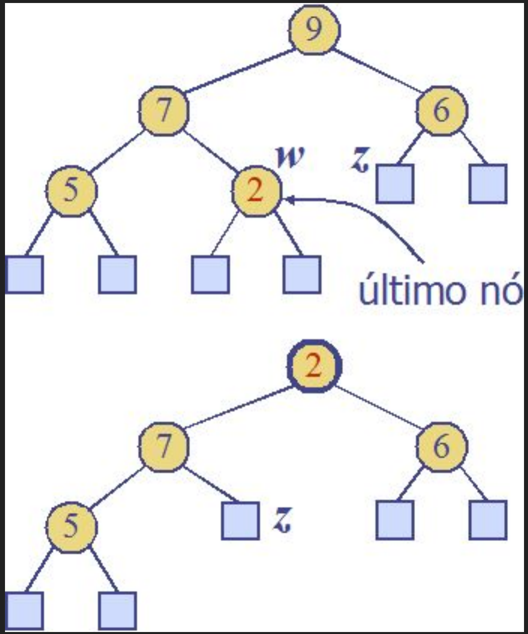
Restauração da Ordem (fix-down)

- O algoritmo fix-down
  - Termina quando o Item movido para a raiz alcança um nó que não possui filho com chave maior que sua
  - Quando ambos os filhos possuem chave maior que o Item inserido, a troca é feita com o filho de maior chave



Remoção

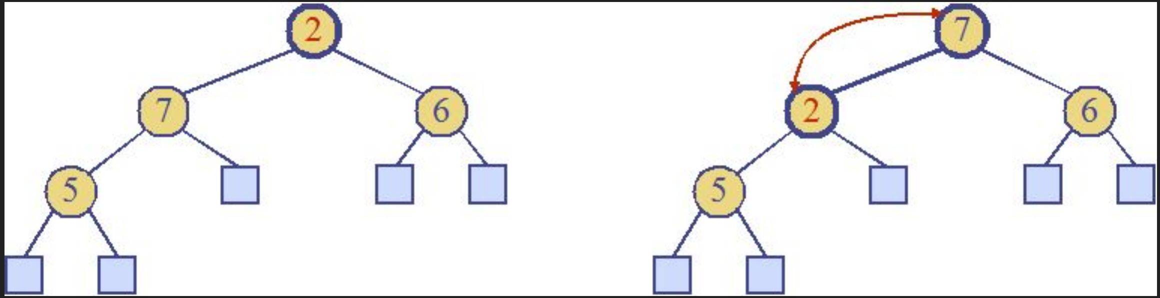
```
Algoritmo Remover(F,x)
  x = inicio(F) //retorna o primeiro nó
  inicio(F) = fim(F) //copia fim no início
  fix_down(F) //restaura ordem do heap
```



Restauração da Ordem (fix-down)

➔ Para uma heap máxima, temos

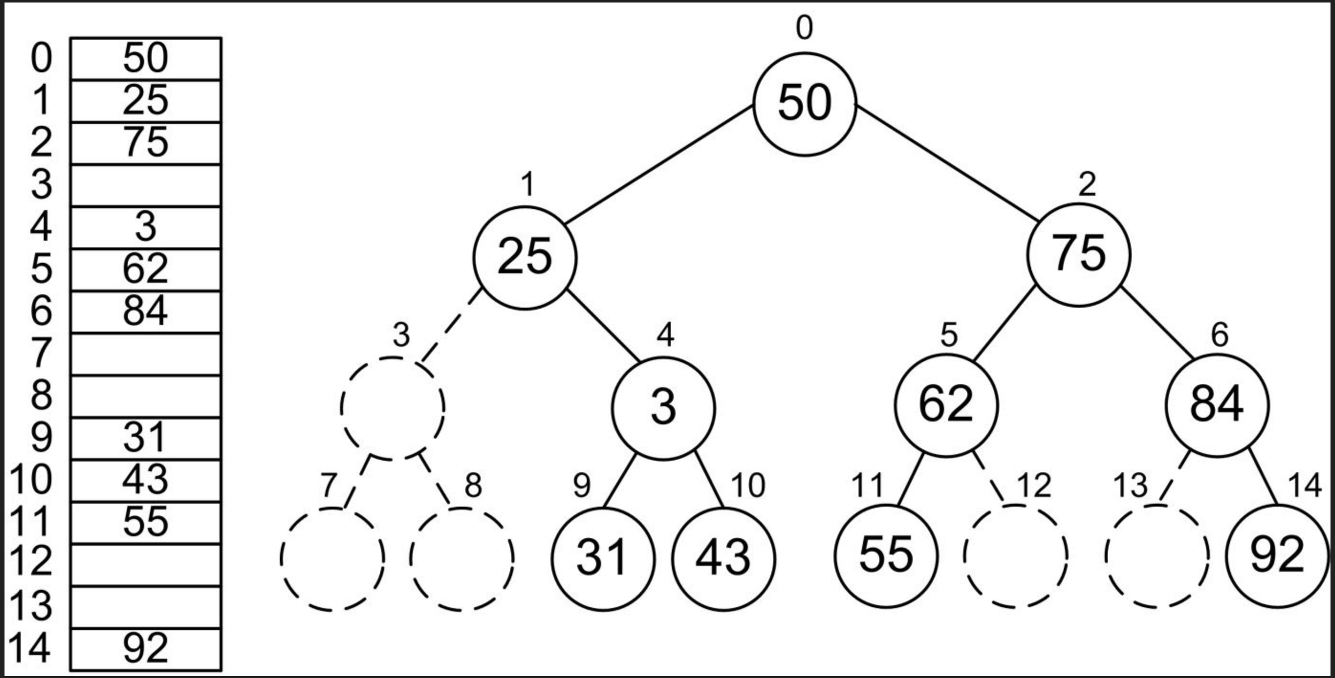
```
Algoritmo fix_down(F)
  w = inicio(F)
  while(tem_filho(w)) {
    m = maior_filho(w)
    if(chave(w) >= chave(m)) break
    swap(F,w,m)
    w = m //desce
  }
```



Implementação em Arranjo

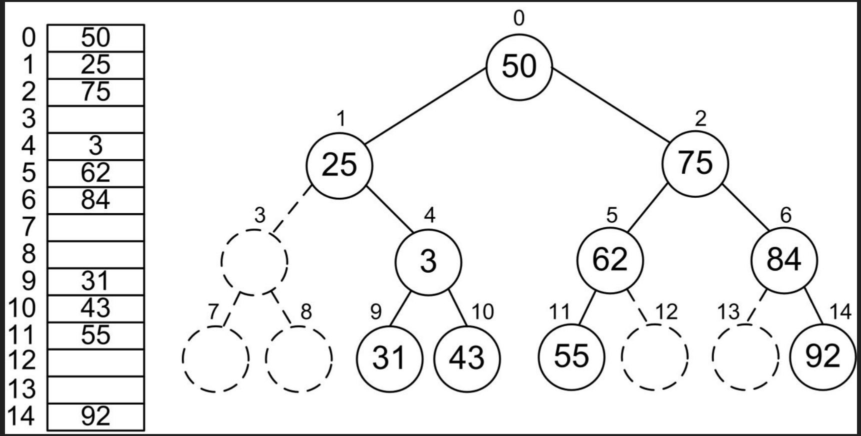
- Vetores podem ser empregados para representar árvores binárias
- Caminha-pela árvore nível por nível, da esquerda para direita armazenando os nós no vetor
  - O primeiro nó fica na posição 0 do vetor, seu filho a esquerda fica na posição 1, e assim por diante...

Implementação em Arranjo



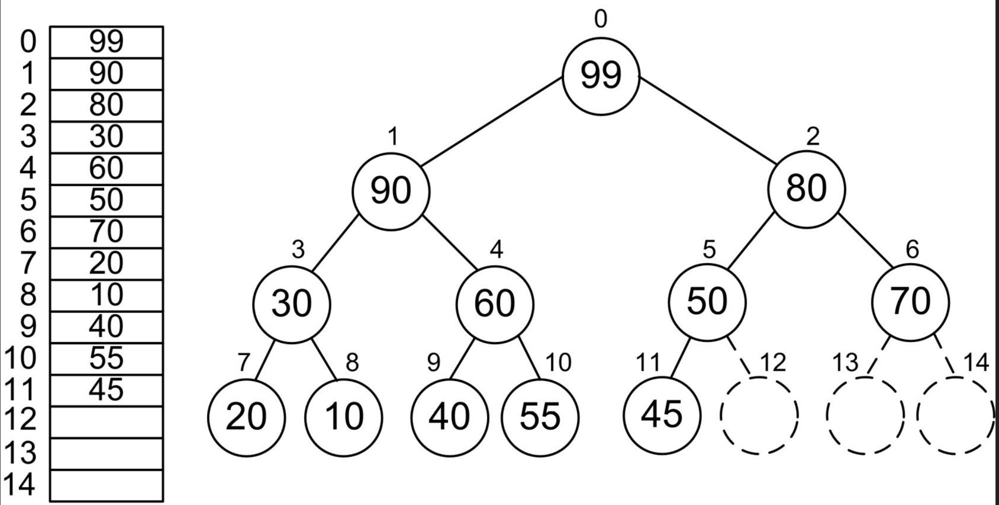
Implementação em Arranjo

- Nessa definição, dado o índice de um item, podemos encontrar seu
  - Filho esquerdo : 2 * índice + 1
  - Filho direito : 2 * índice + 2
  - pai: (índice - 1)/2



Implementação em Arranjo

- Como a Heap é uma árvore completa, o vetor não vai ter "buracos" faltando itens
- Os itens que faltam sempre ficam no fim do vetor



Implementação em Arranjo - Estrutura

```c
//.h
 typedef struct heap_sequencial HEAP_SEQUENCIAL;
 #define TAM 100

//.c
 struct heap_sequencial {
   ITEM * vetor[TAM];
   int fim;
 };

//main.c
HEAP_SEQUENCIAL *Heap;
```

Implementação em Arranjo - Métodos Básicos

```c
HEAP_SEQUENCIAL *hep_criar() {
  HEAP_SEQUENCIAL *heap =
(HEAP_SEQUENCIAL*)malloc(sizeof(HEAP_SEQUENCIAL));
  if (heap != NULL) {
    heap->fim = -1;
  }
  return heap;
}

int heap_cheia(HEAP_SEQUENCIAL *heap) {
  return (heap->fim == TAM - 1);
}

int heap_vazia(HEAP_SEQUENCIAL *heap) {
  return (heap->fim == -1);
}
```

Implementação em Arranjo - Inserção

```
int heap_enfileirar(HEAP_SEQUENCIAL *heap, ITEM *item) {
  if (!heap_cheia(heap)) {
    heap->fim++;
    heap->vetor[heap->fim] = item;
    heap_fix_up(heap);
    return 1;
  }

  return 0;
}
```

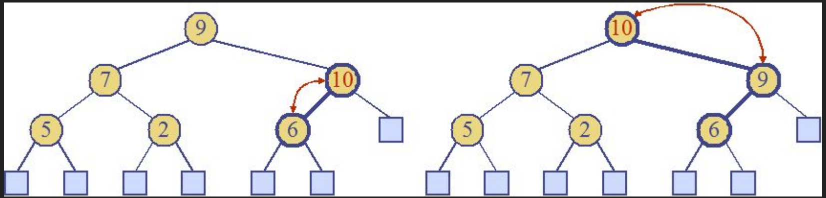
Implementação em Arranjo - Inserção

```c
void heap_swap(HEAP_SEQUENCIAL *heap, int i, int j) {
  ITEM *tmp = heap->vetor[i];
  heap->vetor[i] = heap->vetor[j];
  heap->vetor[j] = tmp;
}

void heap_fix_up(HEAP_SEQUENCIAL *heap) {
  int w = heap->fim;
  int pai = (w - 1) / 2;

  while (w > 0 && item_get_chave(heap->vetor[w]) >
                  item_get_chave(heap->vetor[pai])) {
    heap_swap(heap, w, pai);
    w = pai;
    pai = (pai - 1) / 2;
  }
}
```

```
Algoritmo fix_up(F)
  w = F.ultimo
  while(!isRoot(F,w)) && (key(F,w) >
        key(F,parent(F,w))) {
    swap(F,w,parent(F,w))
    w = parent(F,w) //sobe
  }
```



Comparação: Filas de Prioridade

→ Via fila ordenada → inserção é O(n) → remoção é O(1) → próximo é O(1)	→ Via fila não-ordenada → inserção é O(1) → remoção é O(n) → próximo é O(n)	→ Via Heap → inserção é O(log n) → remoção é O(log n) → próximo é O(1)

Heap e filas de prioridade

- Heap é utilizado como estrutura de apoio a algoritmos clássicos. Ex:
  - Ordenação
    - Heapsort
    - Mergesort
  - Algoritmos em grafos
    - ALGORITMO DE DIJKSTRA busca de menor caminho em grafo ponderado
    - ALGORITMO DE PRIM geração de MST (*Minimal Spaning Tree* - Árvore Geradora Mínima)

Exercícios

- Implementar a remoção de um item em uma heap sequencial
- Implementar o TAD fila de prioridades utilizando uma heap encadeada

Referências

- Material baseado no originais produzidos pelos professores Rudinei Gularte, Gustavo E. de A. P. A. Batista e Fernando V. Paulovich
- SZWARCFITER, J. L.; MARKENZON, L. Estruturas de Dados e seus Algoritmos, Livros Técnicos e Científicos, 1994.
- TENEMBAUM, A.M., e outros Data Structures Using C, Prentice-Hall, 1990.
- ZIVIANI, N. Projeto de Algoritmos, Thomson, 2a. Edição, 2004.