

Filas de Prioridade

Estrutura de Dados Avançada — QXD0015



UNIVERSIDADE
FEDERAL DO CEARÁ
CAMPUS QUIXADÁ

Prof. Atílio Gomes Luiz
gomes.atilio@ufc.br

Universidade Federal do Ceará

1º semestre/2024



Introdução

- Temos uma lista de tarefas a serem realizadas com uma certa prioridade.

Introdução

- Temos uma lista de tarefas a serem realizadas com uma certa prioridade.
 - Deseja-se que as tarefas sejam realizadas em ordem **decrecente** de prioridades.

Introdução

- Temos uma lista de tarefas a serem realizadas com uma certa prioridade.
 - Deseja-se que as tarefas sejam realizadas em ordem **decrecente** de prioridades.
 - As **prioridades das tarefas podem variar** ao longo do tempo.

- Temos uma lista de tarefas a serem realizadas com uma certa prioridade.
 - Deseja-se que as tarefas sejam realizadas em ordem **decrecente** de prioridades.
 - As **prioridades das tarefas podem variar** ao longo do tempo.
 - Novas tarefas podem ingressar na tabela a cada instante.

- Temos uma lista de tarefas a serem realizadas com uma certa prioridade.
 - Deseja-se que as tarefas sejam realizadas em ordem **decrecente** de prioridades.
 - As **prioridades das tarefas podem variar** ao longo do tempo.
 - Novas tarefas podem ingressar na tabela a cada instante.
- Para encontrar a ordem desejada de execução das tarefas, um algoritmo deve:

Introdução

- Temos uma lista de tarefas a serem realizadas com uma certa prioridade.
 - Deseja-se que as tarefas sejam realizadas em ordem **decrecente** de prioridades.
 - As **prioridades das tarefas podem variar** ao longo do tempo.
 - Novas tarefas podem ingressar na tabela a cada instante.
- Para encontrar a ordem desejada de execução das tarefas, um algoritmo deve:
 - sucessivamente, escolher o dado de maior prioridade e retirá-lo da lista.

- Temos uma lista de tarefas a serem realizadas com uma certa prioridade.
 - Deseja-se que as tarefas sejam realizadas em ordem **decrecente** de prioridades.
 - As **prioridades das tarefas podem variar** ao longo do tempo.
 - Novas tarefas podem ingressar na tabela a cada instante.
- Para encontrar a ordem desejada de execução das tarefas, um algoritmo deve:
 - sucessivamente, escolher o dado de maior prioridade e retirá-lo da lista.
 - introduzir novos dados no momento adequado.

Fila de Prioridades

Uma **fila de prioridades** é uma estrutura de dados na qual a cada um de seus elementos está associada uma **prioridade**.

Fila de Prioridades

Uma **fila de prioridades** é uma estrutura de dados na qual a cada um de seus elementos está associada uma **prioridade**.

Prioridade: um valor numérico armazenado em algum dos campos de um elemento da estrutura de dados.

Fila de Prioridades

Uma **fila de prioridades** é uma estrutura de dados na qual a cada um de seus elementos está associada uma **prioridade**.

Prioridade: um valor numérico armazenado em algum dos campos de um elemento da estrutura de dados.

As filas de prioridades possuem três operações básicas:

Fila de Prioridades

Uma **fila de prioridades** é uma estrutura de dados na qual a cada um de seus elementos está associada uma **prioridade**.

Prioridade: um valor numérico armazenado em algum dos campos de um elemento da estrutura de dados.

As filas de prioridades possuem três operações básicas:

- **Selecionar (acessar)** o elemento com maior prioridade.

Fila de Prioridades

Uma **fila de prioridades** é uma estrutura de dados na qual a cada um de seus elementos está associada uma **prioridade**.

Prioridade: um valor numérico armazenado em algum dos campos de um elemento da estrutura de dados.

As filas de prioridades possuem três operações básicas:

- **Selecionar (acessar)** o elemento com maior prioridade.
- **Inserir** um novo elemento.

Fila de Prioridades

Uma **fila de prioridades** é uma estrutura de dados na qual a cada um de seus elementos está associada uma **prioridade**.

Prioridade: um valor numérico armazenado em algum dos campos de um elemento da estrutura de dados.

As filas de prioridades possuem três operações básicas:

- **Selecionar (acessar)** o elemento com maior prioridade.
- **Inserir** um novo elemento.
- **Remover** o elemento com maior prioridade.

Fila de Prioridades

Uma **fila de prioridades** é uma estrutura de dados na qual a cada um de seus elementos está associada uma **prioridade**.

Prioridade: um valor numérico armazenado em algum dos campos de um elemento da estrutura de dados.

As filas de prioridades possuem três operações básicas:

- **Selecionar (acessar)** o elemento com maior prioridade.
- **Inserir** um novo elemento.
- **Remover** o elemento com maior prioridade.

Outras operações auxiliares podem existir:

Fila de Prioridades

Uma **fila de prioridades** é uma estrutura de dados na qual a cada um de seus elementos está associada uma **prioridade**.

Prioridade: um valor numérico armazenado em algum dos campos de um elemento da estrutura de dados.

As filas de prioridades possuem três operações básicas:

- **Selecionar (acessar)** o elemento com maior prioridade.
- **Inserir** um novo elemento.
- **Remover** o elemento com maior prioridade.

Outras operações auxiliares podem existir:

- **Alterar** a prioridade de um elemento.

Fila de Prioridades

Uma **fila de prioridades** é uma estrutura de dados na qual a cada um de seus elementos está associada uma **prioridade**.

Prioridade: um valor numérico armazenado em algum dos campos de um elemento da estrutura de dados.

As filas de prioridades possuem três operações básicas:

- **Selecionar (acessar)** o elemento com maior prioridade.
- **Inserir** um novo elemento.
- **Remover** o elemento com maior prioridade.

Outras operações auxiliares podem existir:

- **Alterar** a prioridade de um elemento.
- **Construir** a fila a partir de um dado grupo de elementos.

Fila de Prioridades

Uma **fila de prioridades** é uma estrutura de dados na qual a cada um de seus elementos está associada uma **prioridade**.

Prioridade: um valor numérico armazenado em algum dos campos de um elemento da estrutura de dados.

As filas de prioridades possuem três operações básicas:

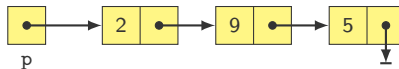
- **Selecionar (acessar)** o elemento com maior prioridade.
- **Inserir** um novo elemento.
- **Remover** o elemento com maior prioridade.

Outras operações auxiliares podem existir:

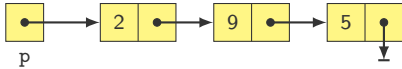
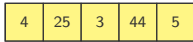
- **Alterar** a prioridade de um elemento.
- **Construir** a fila a partir de um dado grupo de elementos.

Como implementar esta estrutura de dados?

Primeira ideia — Lista não ordenada

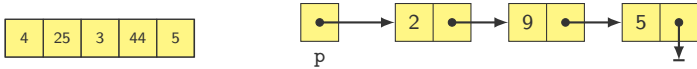


Primeira ideia — Lista não ordenada



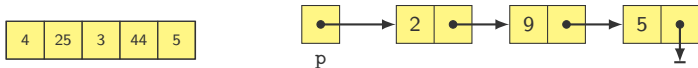
- **Inserção e construção:** o novo elemento pode ser colocado em qualquer posição conveniente dependendo do tipo de alocação utilizada: sequencial ou encadeada.

Primeira ideia — Lista não ordenada



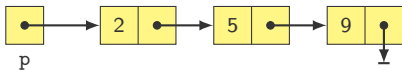
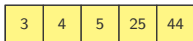
- **Inserção e construção:** o novo elemento pode ser colocado em qualquer posição conveniente dependendo do tipo de alocação utilizada: sequencial ou encadeada.
- **Remoção, alteração e seleção:** implica percorrer a lista em busca do elemento de maior prioridade.

Primeira ideia — Lista não ordenada

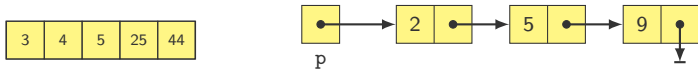


- **Inserção e construção:** o novo elemento pode ser colocado em qualquer posição conveniente dependendo do tipo de alocação utilizada: sequencial ou encadeada.
- **Remoção, alteração e seleção:** implica percorrer a lista em busca do elemento de maior prioridade.
- seleção: $O(n)$
- inserção $O(1)$
- remoção: $O(n)$
- alteração: $O(n)$
- construção: $O(n)$

Segunda ideia — Lista ordenada

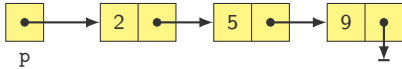


Segunda ideia — Lista ordenada



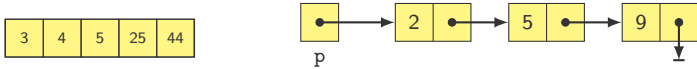
- **Seleção e remoção:** imediatas. **Por quê?**

Segunda ideia — Lista ordenada



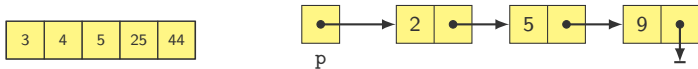
- **Seleção e remoção:** imediatas. **Por quê?**
- **Inserção e alteração:** percorrer a lista até encontrar posição correta do elemento. (pode ou não ocorrer deslocamento)

Segunda ideia — Lista ordenada



- **Seleção e remoção:** imediatas. **Por quê?**
- **Inserção e alteração:** percorrer a lista até encontrar posição correta do elemento. (pode ou não ocorrer deslocamento)
- **Construção:** exige ordenação prévia da lista.

Segunda ideia — Lista ordenada



- **Seleção e remoção:** imediatas. **Por quê?**
- **Inserção e alteração:** percorrer a lista até encontrar posição correta do elemento. (pode ou não ocorrer deslocamento)
- **Construção:** exige ordenação prévia da lista.
- seleção: $O(1)$
- inserção $O(n)$
- remoção: $O(1)$
- alteração: $O(n)$
- construção: $O(n \lg n)$

Heap Binário



Estrutura de dados Heap Binário

- Existem dois tipos de heap: **heap máximo** e **heap mínimo**.
- Nesta aula analisamos o heap binário máximo.
- Para simplificar, chamaremos apenas de **heap**.

Estrutura de dados Heap Binário

- Existem dois tipos de heap: **heap máximo** e **heap mínimo**.
- Nesta aula analisamos o heap binário máximo.
- Para simplificar, chamaremos apenas de **heap**.
- Os heaps têm uma **propriedade estrutural** e uma **propriedade de ordem**.
 - Uma operação em um heap pode destruir uma das propriedades e não deve terminar até que todas as propriedades do heap estejam satisfeitas.

Definição de Heap — Propriedade de ordem

- Um **heap** é um vetor $A[1 \dots n]$ que satisfaz a propriedade:

$$A[\lfloor i/2 \rfloor] \geq A[i], \text{ para } 2 \leq i \leq n.$$

1	2	3	4	5	6	7	8	9	10	11	12	13	14
999	888	777	555	666	777	555	222	333	444	111	333	666	333

Definição de Heap — Propriedade de ordem

- Um **heap** é um vetor $A[1 \dots n]$ que satisfaz a propriedade:

$$A[\lfloor i/2 \rfloor] \geq A[i], \text{ para } 2 \leq i \leq n.$$

1	2	3	4	5	6	7	8	9	10	11	12	13	14
999	888	777	555	666	777	555	222	333	444	111	333	666	333

- Adoto a mesma convenção dos livros:** supponho que os índices do vetor são $1 \dots n$ e não $0 \dots n - 1$.

Definição de Heap — Propriedade de ordem

- Um **heap** é um vetor $A[1 \dots n]$ que satisfaz a propriedade:

$$A[\lfloor i/2 \rfloor] \geq A[i], \text{ para } 2 \leq i \leq n.$$

1	2	3	4	5	6	7	8	9	10	11	12	13	14
999	888	777	555	666	777	555	222	333	444	111	333	666	333

- Adoto a mesma convenção dos livros:** supponho que os índices do vetor são $1 \dots n$ e não $0 \dots n - 1$.
- Observação:** Segue imediatamente da definição que $A[1]$ é um elemento máximo do heap.

Definição de Heap — Propriedade de ordem

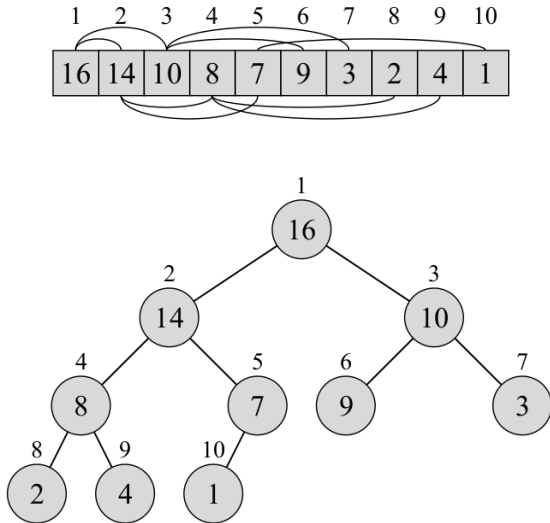
- Um **heap** é um vetor $A[1 \dots n]$ que satisfaz a propriedade:

$$A[\lfloor i/2 \rfloor] \geq A[i], \text{ para } 2 \leq i \leq n.$$

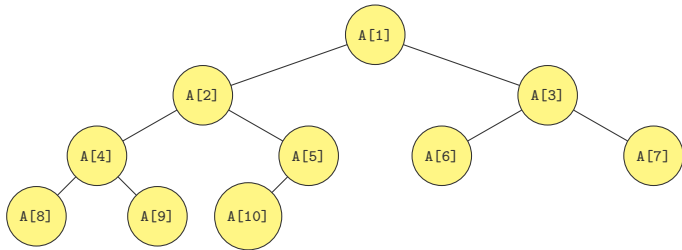
1	2	3	4	5	6	7	8	9	10	11	12	13	14
999	888	777	555	666	777	555	222	333	444	111	333	666	333

- Adoto a mesma convenção dos livros:** supponho que os índices do vetor são $1 \dots n$ e não $0 \dots n - 1$.
- Observação:** Segue imediatamente da definição que $A[1]$ é um elemento máximo do heap.
- Assim como o Cormen et al. , supponho que o array **A** que representa o heap é um objeto que possui dois atributos:
 - A.length:** a capacidade total do array.
 - A.heapSize:** quantos elementos do array A são elementos do heap.
Note que $0 \leq A.heapSize \leq A.length$.

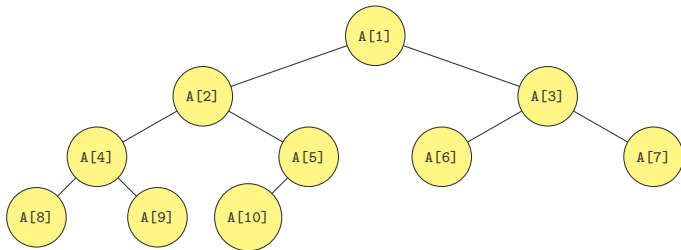
Definição de Heap — Propriedade estrutural



Ver heap como árvore binária completa

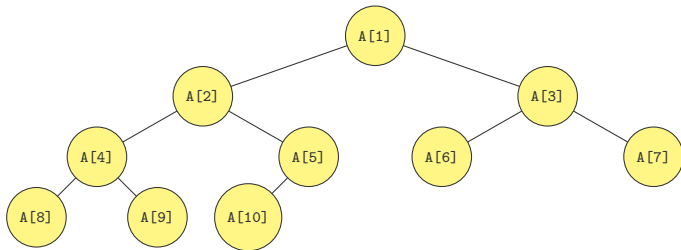


Ver heap como árvore binária completa



- O nó **A[1]** é a raiz da árvore.

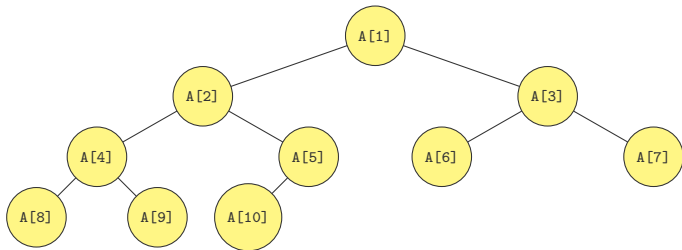
Ver heap como árvore binária completa



- O nó **A[1]** é a raiz da árvore.

Em relação a **A[i]**:

Ver heap como árvore binária completa

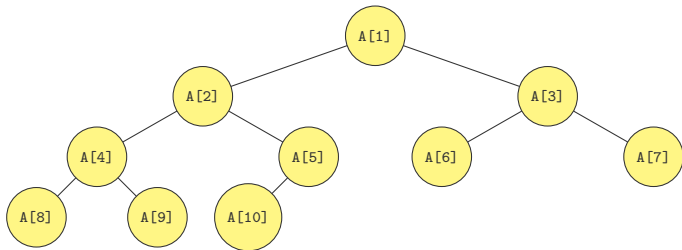


- O nó **A[1]** é a raiz da árvore.

Em relação a **A[i]**:

- o filho esquerdo é **A[2i]** e o filho direito é **A[2i+1]**

Ver heap como árvore binária completa

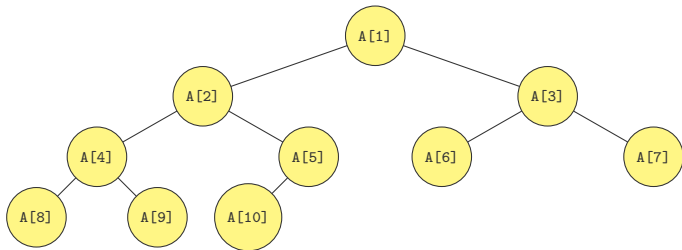


- O nó **A[1]** é a raiz da árvore.

Em relação a **A[i]**:

- o filho esquerdo é **A[2i]** e o filho direito é **A[2i+1]**
- o pai é **A[i/2]**

Ver heap como árvore binária completa



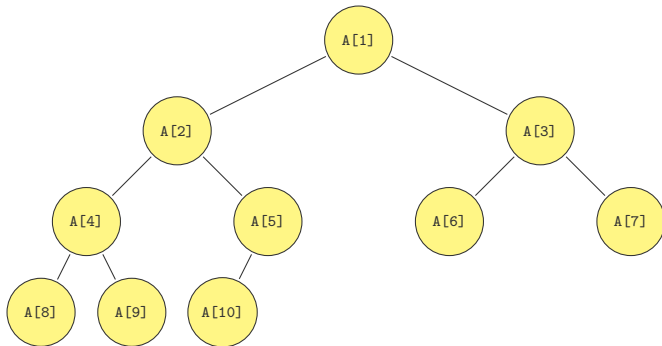
- O nó **A[1]** é a raiz da árvore.

Em relação a **A[i]**:

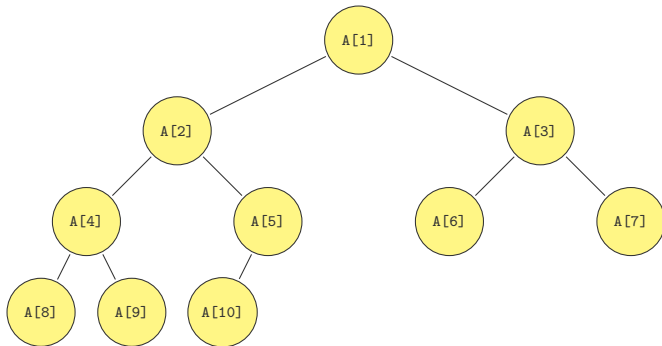
- o filho esquerdo é **A[2i]** e o filho direito é **A[2i+1]**
- o pai é **A[i/2]**

Atenção: A[1] não tem pai, o filho esquerdo de A[i] só existe se $2i \leq A.\text{heapSize}$ e o filho direito de A[i] só existe se $2i+1 \leq A.\text{heapSize}$.

Ver heap como árvore binária completa

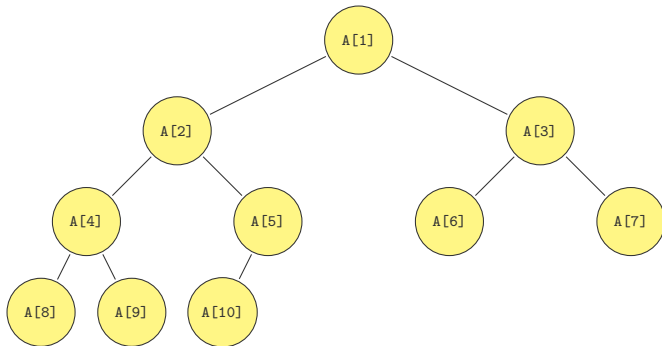


Ver heap como árvore binária completa



Da propriedade de ordem do heap, segue que:

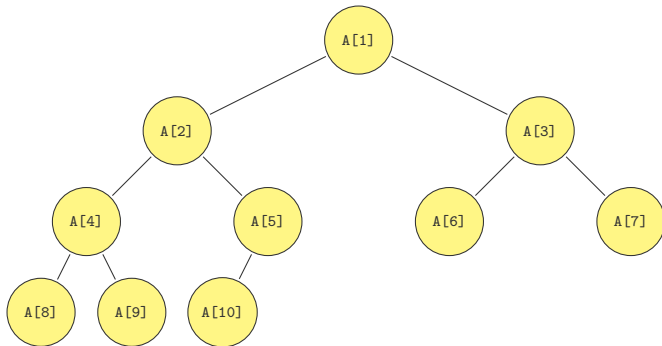
Ver heap como árvore binária completa



Da propriedade de ordem do heap, segue que:

- Os filhos são menores ou iguais ao pai.

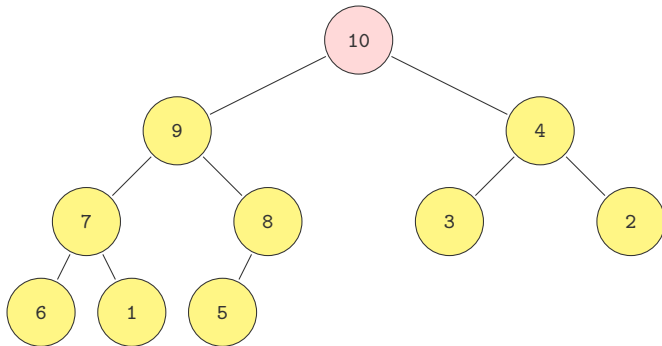
Ver heap como árvore binária completa



Da propriedade de ordem do heap, segue que:

- Os filhos são menores ou iguais ao pai.
- Ou seja, a raiz é o máximo (o elemento $A[1]$).

Ver heap como árvore binária completa



Da propriedade de ordem do heap, segue que:

- Os filhos são menores ou iguais ao pai.
- Ou seja, a raiz é o máximo (o elemento $A[1]$).

Resumo

- Todo vetor pode ser visto como uma árvore binária completa.

Resumo

- Todo vetor pode ser visto como uma árvore binária completa.
- O conjunto de índices de qualquer vetor $A[1..n]$ pode ser encarado como uma árvore binária da seguinte maneira:

- Todo vetor pode ser visto como uma árvore binária completa.
- O conjunto de índices de qualquer vetor $A[1..n]$ pode ser encarado como uma árvore binária da seguinte maneira:
 - o índice 1 é a **raiz da árvore**;

- Todo vetor pode ser visto como uma árvore binária completa.
- O conjunto de índices de qualquer vetor $A[1..n]$ pode ser encarado como uma árvore binária da seguinte maneira:
 - o índice 1 é a **raiz da árvore**;
 - o **pai** de qualquer índice f é $f/2$ (1 não tem pai);

- Todo vetor pode ser visto como uma árvore binária completa.
- O conjunto de índices de qualquer vetor $A[1..n]$ pode ser encarado como uma árvore binária da seguinte maneira:
 - o índice 1 é a **raiz da árvore**;
 - o **pai** de qualquer índice f é $f/2$ (1 não tem pai);
 - o **filho esquerdo** de f é $2f$ (esse filho só existe se $2f \leq n$);

- Todo vetor pode ser visto como uma árvore binária completa.
- O conjunto de índices de qualquer vetor $A[1..n]$ pode ser encarado como uma árvore binária da seguinte maneira:
 - o índice 1 é a **raiz da árvore**;
 - o **pai** de qualquer índice f é $f/2$ (1 não tem pai);
 - o **filho esquerdo** de f é $2f$ (esse filho só existe se $2f \leq n$);
 - o **filho direito** de f é $2f + 1$ (ele só existe se $2f + 1 \leq n$).

TAD Fila de Prioridade Máxima

Estamos interessados em implementar as seguintes operações do Tipo Abstrato de Dados **Fila de Prioridade Máxima**:

- **Operações do TAD:**

- Seleção do elemento máximo
- Inserção de um novo elemento
- Remoção do elemento máximo
- Construção de um heap máximo a partir de um conjunto de elementos
- Aumento da prioridade de um elemento
- Redução da prioridade de um elemento

- **Dados do TAD:**

- Um heap binário de máximo $A[1..n]$

Seleção do Máximo



Pseudocódigo – Seleção do máximo

getMax(A)

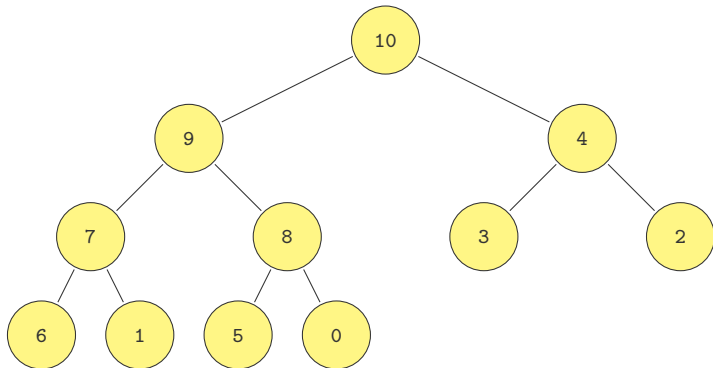
```
1  if A.heapSize > 0
2      return A[1]
3  else
4      error "underflow error"
```

Complexidade: $O(1)$

Alteração de Prioridades

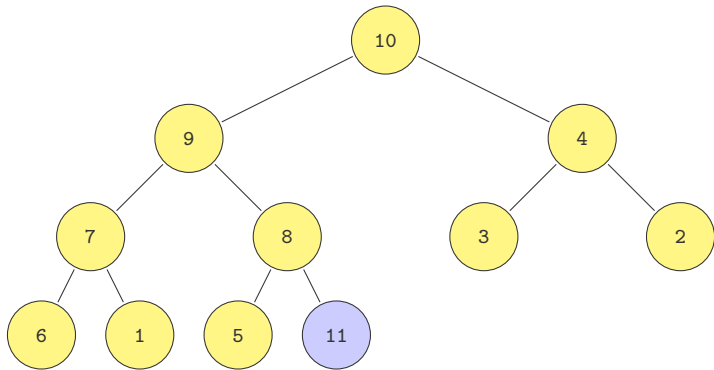


Aumento da prioridade



- Aumentar a prioridade do elemento com prioridade 0 para 11.

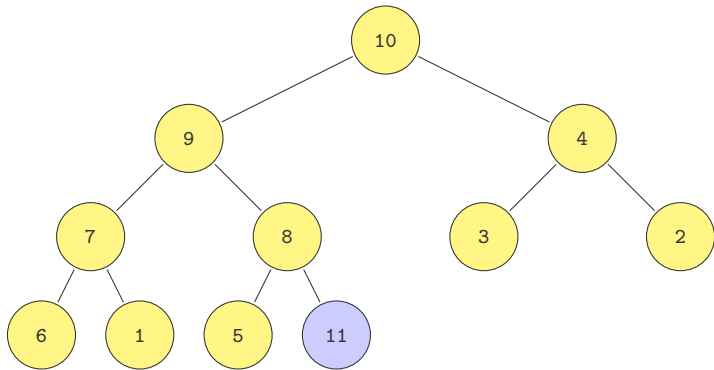
Aumento da prioridade



Note que o vetor não é mais um heap. Por quê?

- É possível consertar? Como?

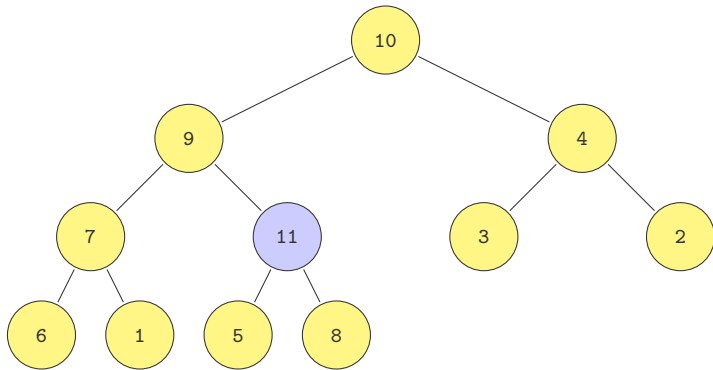
Aumento da prioridade



Basta ir subindo no heap, trocando com o pai se necessário

- O irmão já é menor que o pai, não precisamos mexer nele

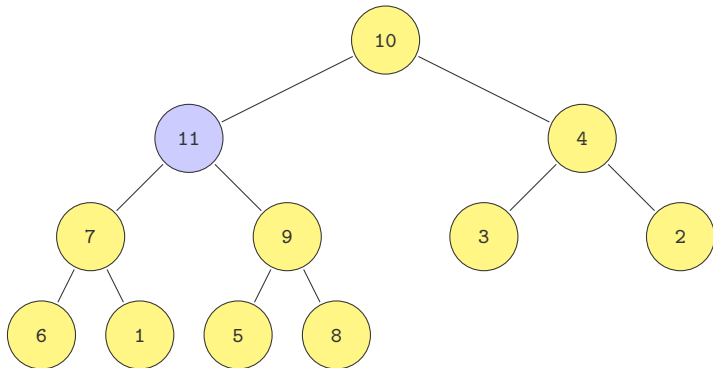
Aumento da prioridade



Basta ir subindo no heap, trocando com o pai se necessário

- O irmão já é menor que o pai, não precisamos mexer nele

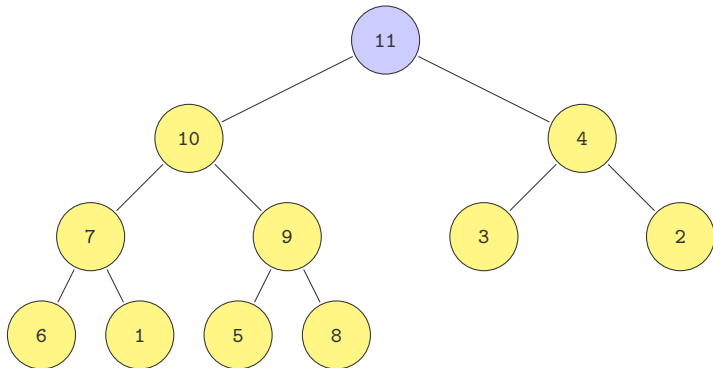
Aumento da prioridade



Basta ir subindo no heap, trocando com o pai se necessário

- O irmão já é menor que o pai, não precisamos mexer nele

Aumento da prioridade



Basta ir subindo no heap, trocando com o pai se necessário

- O irmão já é menor que o pai, não precisamos mexer nele

Pseudocódigo – Aumento da prioridade

maxHeap-increaseKey(A, i, newKey)

```
1  if newKey < A[i] then
2      error "invalid key"
3  A[i] = newKey
4  maxFixUp(A,i)
```

Pseudocódigo – Aumento da prioridade

Pseudocódigo – Aumento da prioridade

maxFixUp(A, i)

```
1  p = i/2
2  while p >= 1 and A[i] > A[p]
3      aux = A[i]
4      A[i] = A[p]
5      A[p] = aux
6      i = p
7      p = p/2
```

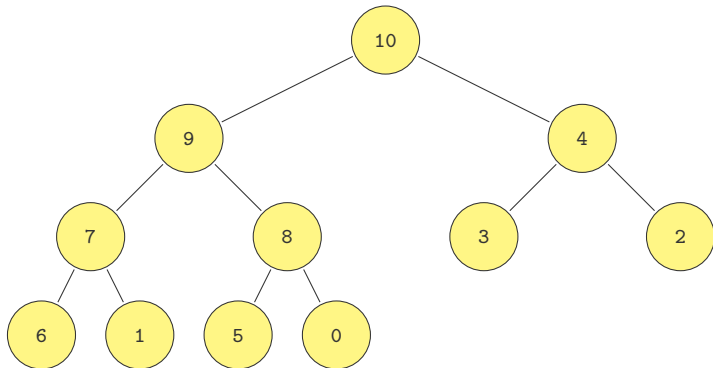
Pseudocódigo – Aumento da prioridade

maxFixUp(A, i)

```
1  p = i/2
2  while p >= 1 and A[i] > A[p]
3      aux = A[i]
4      A[i] = A[p]
5      A[p] = aux
6      i = p
7      p = p/2
```

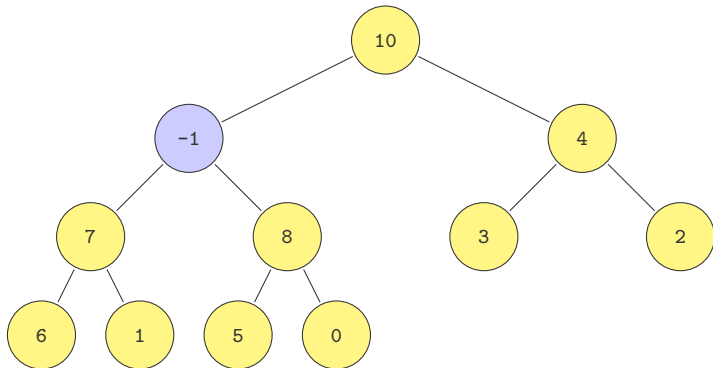
Qual a complexidade de pior caso deste algoritmo?

Redução da prioridade



- Reduzir a prioridade do elemento com prioridade 9 para -1 .

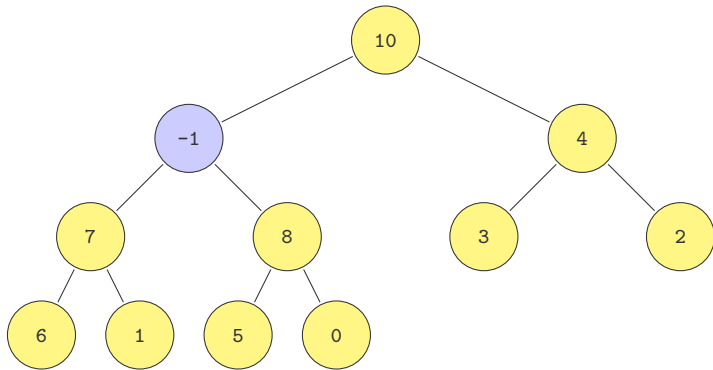
Redução da prioridade



Note que o vetor não é mais um heap. Por quê?

- É possível consertar? Como?

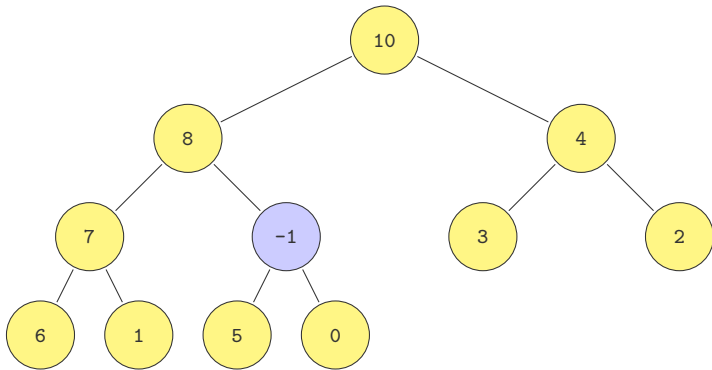
Redução da prioridade



Basta ir descendo no heap, trocando com o pai com o maior filho se necessário

- O menor filho já é menor que o irmão, não precisamos mexer nele

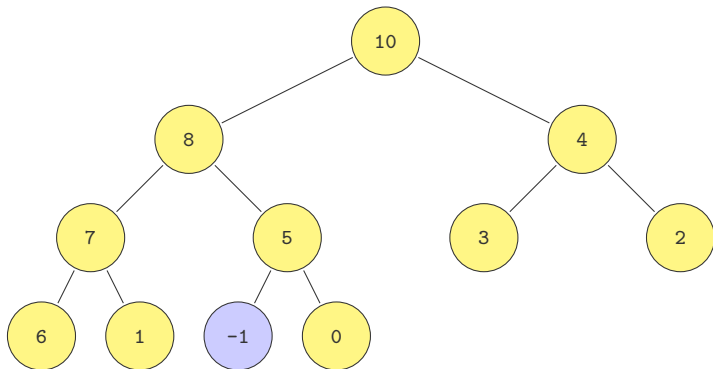
Redução da prioridade



Basta ir descendo no heap, trocando com o pai com o maior filho se necessário

- O menor filho já é menor que o irmão, não precisamos mexer nele

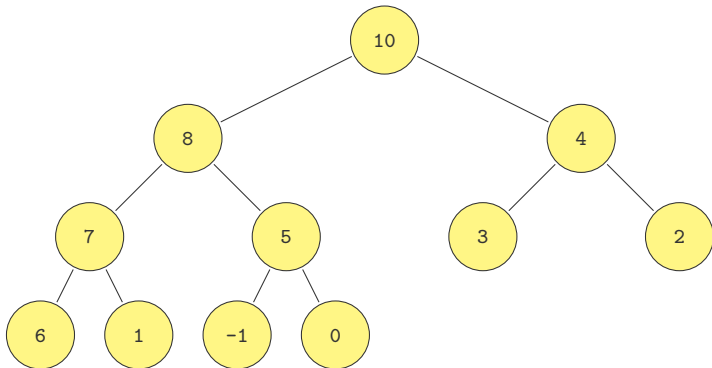
Redução da prioridade



Basta ir descendo no heap, trocando com o pai com o maior filho se necessário

- O menor filho já é menor que o irmão, não precisamos mexer nele

Redução da prioridade



Basta ir descendo no heap, trocando com o pai com o maior filho se necessário

- O menor filho já é menor que o irmão, não precisamos mexer nele

Pseudocódigo – Redução da prioridade

Pseudocódigo – Redução da prioridade

maxHeap-decreaseKey(A, i, newKey)

```
1  if newKey > A[i] then
2      error "invalid key"
3  A[i] = newKey
4  maxFixDown(A,i)
```

Pseudocódigo – Redução da prioridade

Pseudocódigo – Redução da prioridade

maxFixDown(A, i)

```
1  while  $2i \leq A.\text{heapSize}$ 
2       $l = 2i$ 
3       $r = 2i+1$ 
4       $\text{largest} = i$ 
5      if  $l \leq A.\text{heapSize}$  and  $A[l] > A[\text{largest}]$  then
6           $\text{largest} = l$ 
7      if  $r \leq A.\text{heapSize}$  and  $A[r] > A[\text{largest}]$  then
8           $\text{largest} = r$ 
9      if  $\text{largest} \neq i$  then
10          $\text{aux} = A[i]$ 
11          $A[i] = A[\text{largest}]$ 
12          $A[\text{largest}] = \text{aux}$ 
13          $i = \text{largest}$ 
14     else
15         break
```

Pseudocódigo – Redução da prioridade

maxFixDown(A, i)

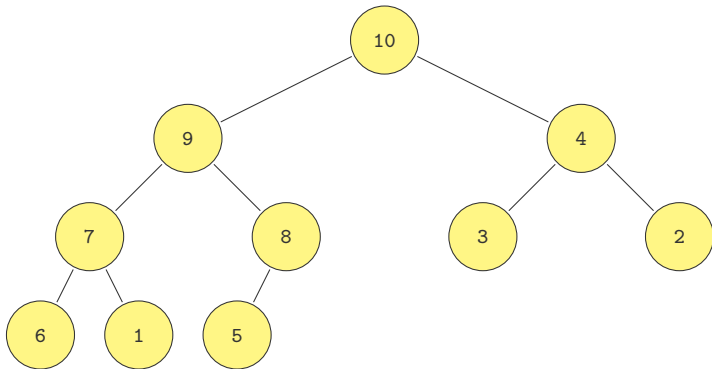
```
1  while  $2i \leq A.\text{heapSize}$ 
2       $l = 2i$ 
3       $r = 2i+1$ 
4       $\text{largest} = i$ 
5      if  $l \leq A.\text{heapSize}$  and  $A[l] > A[\text{largest}]$  then
6           $\text{largest} = l$ 
7      if  $r \leq A.\text{heapSize}$  and  $A[r] > A[\text{largest}]$  then
8           $\text{largest} = r$ 
9      if  $\text{largest} \neq i$  then
10          $\text{aux} = A[i]$ 
11          $A[i] = A[\text{largest}]$ 
12          $A[\text{largest}] = \text{aux}$ 
13          $i = \text{largest}$ 
14     else
15         break
```

Qual a complexidade de pior caso deste algoritmo?

Inserção

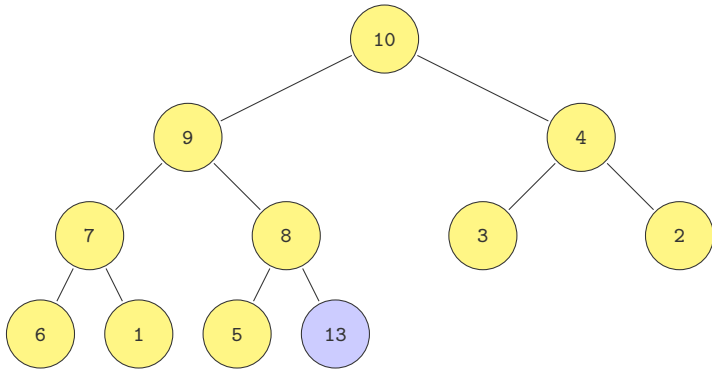


Inserção de novo elemento



- Inserir um elemento com chave 13.

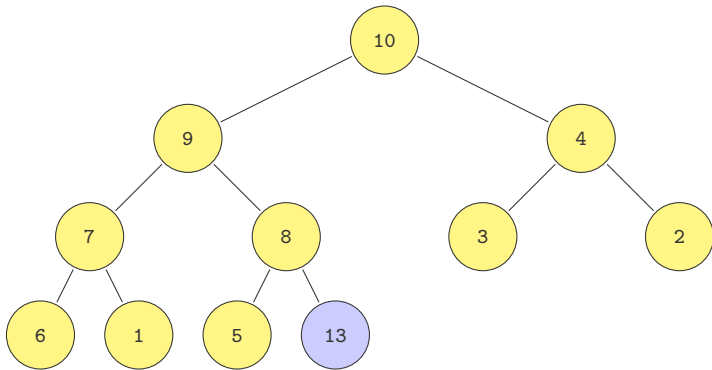
Inserção de novo elemento



Note que o vetor não é mais um heap. Por quê?

- É possível consertar? Como?

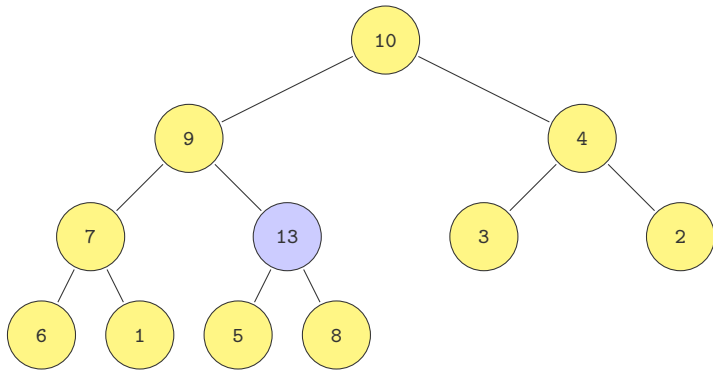
Inserção de novo elemento



Basta ir subindo no heap, trocando com o pai se necessário

- O irmão já é menor que o pai, não precisamos mexer nele

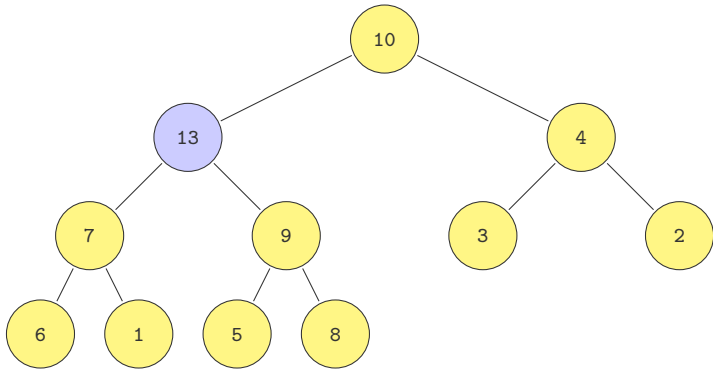
Inserção de novo elemento



Basta ir subindo no heap, trocando com o pai se necessário

- O irmão já é menor que o pai, não precisamos mexer nele

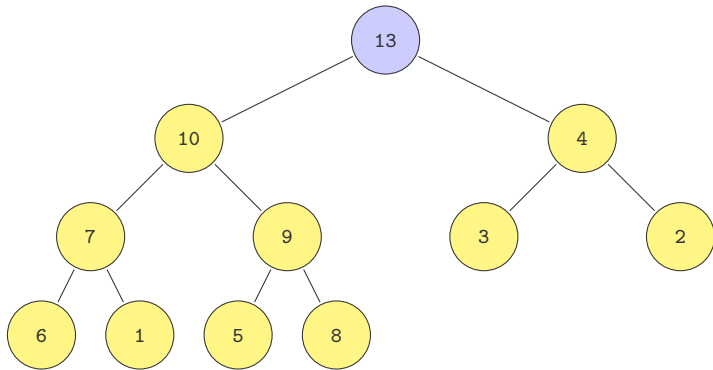
Inserção de novo elemento



Basta ir subindo no heap, trocando com o pai se necessário

- O irmão já é menor que o pai, não precisamos mexer nele

Inserção de novo elemento



Basta ir subindo no heap, trocando com o pai se necessário

- O irmão já é menor que o pai, não precisamos mexer nele

Pseudocódigo – Inserção

Pseudocódigo – Inserção

maxHeapInsert(A, newKey)

```
1  if A.heapSize >= A.length then  
2      error "heap overflow"  
3  A.heapSize = A.heapSize + 1  
4  A[A.heapSize] = newKey  
5  MAXFIXUP(A, A.heapSize)
```

Pseudocódigo – Inserção

maxHeapInsert(A, newKey)

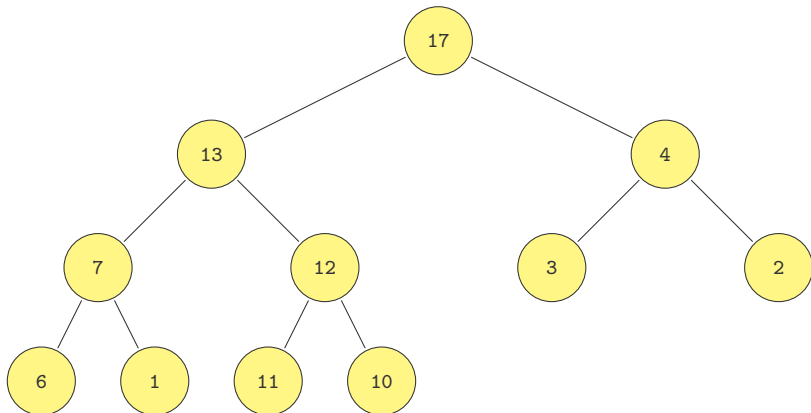
```
1  if A.heapSize >= A.length then  
2      error "heap overflow"  
3  A.heapSize = A.heapSize + 1  
4  A[A.heapSize] = newKey  
5  MAXFIXUP(A, A.heapSize)
```

Qual a complexidade de pior caso deste algoritmo?

Remoção

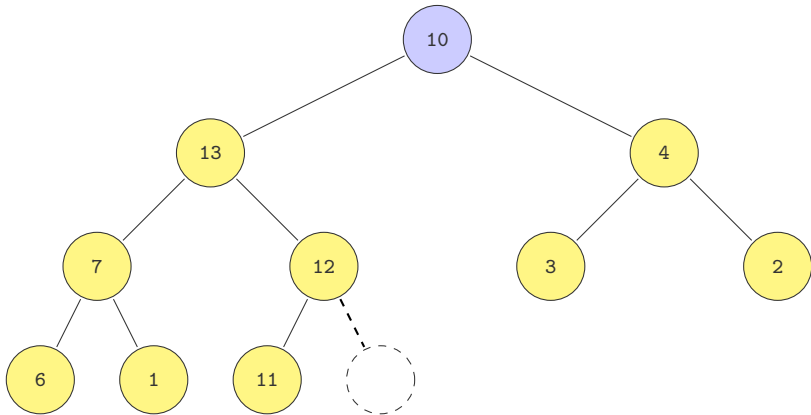


Extrair o elemento de máxima prioridade



- Extrair o elemento de maior prioridade.
- O que fazer inicialmente?

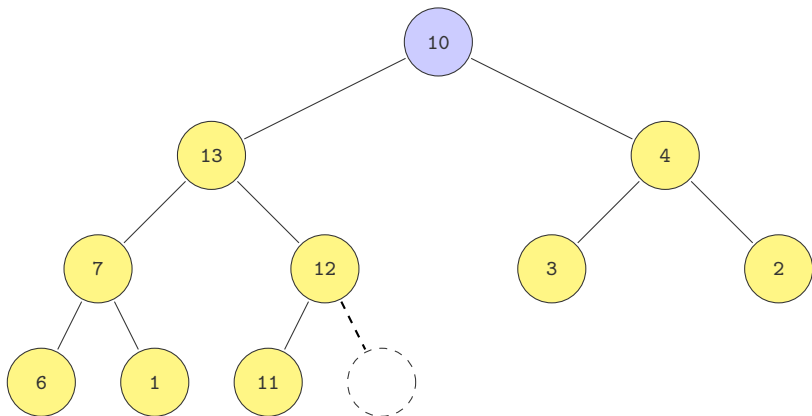
Extrair o elemento de máxima prioridade



Note que o vetor não é mais um heap. Por quê?

- É possível consertar? Como?

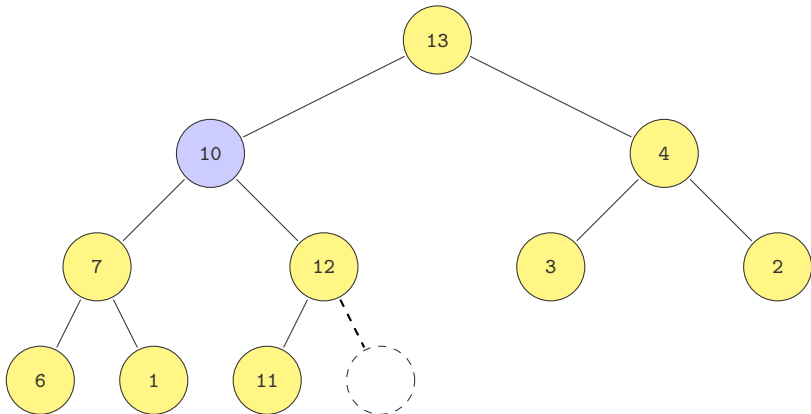
Extrair o elemento de máxima prioridade



Basta descer no heap, trocando com o pai com o maior filho se necessário.

- O menor filho já é menor que o irmão, não precisamos mexer nele.

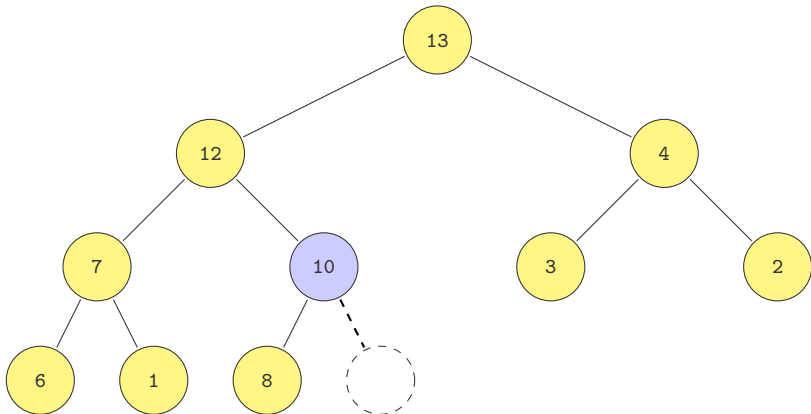
Extrair o elemento de máxima prioridade



Basta descer no heap, trocando com o pai com o maior filho se necessário.

- O menor filho já é menor que o irmão, não precisamos mexer nele.

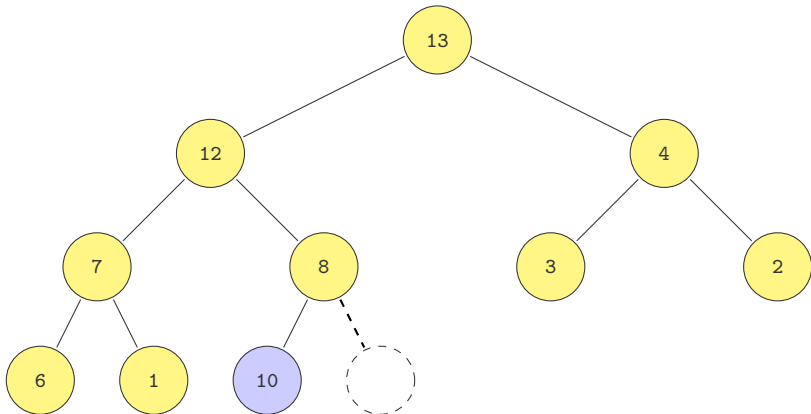
Extrair o elemento de máxima prioridade



Basta descer no heap, trocando com o pai com o maior filho se necessário.

- O menor filho já é menor que o irmão, não precisamos mexer nele.

Extrair o elemento de máxima prioridade



Basta descer no heap, trocando com o pai com o maior filho se necessário.

- O menor filho já é menor que o irmão, não precisamos mexer nele.

Pseudocódigo – Extrair o máximo

maxHeap-extractMax(A)

```
1  if A.heapSize < 1 then  
2      error "heap underflow"  
3  dados = A[1]  
4  A[1] = A[A.heapSize]    ▷ copia elemento  
5  A.heapSize = A.heapSize - 1  
6  MAXFixDOWN(A, 1)  
7  return dados
```

Pseudocódigo – Extrair o máximo

maxHeap-extractMax(A)

```
1  if A.heapSize < 1 then  
2      error "heap underflow"  
3  dados = A[1]  
4  A[1] = A[A.heapSize]    ▷ copia elemento  
5  A.heapSize = A.heapSize - 1  
6  MAXFIXDOWN(A, 1)  
7  return dados
```

Qual a complexidade de pior caso deste algoritmo?

Comparações

Diferentes Implementações de Fila de Prioridades

	lista não ordenada	lista ordenada	heap
seleção	$O(n)$	$O(1)$	$O(1)$
inserção	$O(1)$	$O(n)$	$O(\lg n)$
remoção	$O(n)$	$O(1)$	$O(\lg n)$
alteração	$O(n)$	$O(n)$	$O(\lg n)$
construção	$O(n)$	$O(n \lg n)$	$O(n)$

Comparações

Diferentes Implementações de Fila de Prioridades

	lista não ordenada	lista ordenada	heap
seleção	$O(n)$	$O(1)$	$O(1)$
inserção	$O(1)$	$O(n)$	$O(\lg n)$
remoção	$O(n)$	$O(1)$	$O(\lg n)$
alteração	$O(n)$	$O(n)$	$O(\lg n)$
construção	$O(n)$	$O(n \lg n)$	$O(n)$

Vamos ver a construção do heap agora!

Construção de um heap máximo



Transformando um vetor em um heap

- Suponha que nos seja dado de início um vetor com $n > 0$ elementos.
- Como transformar este vetor em um heap máximo?

1	2	3	4	5	6	7	8	9	10
4	8	2	7	6	3	5	1	9	10

Transformando um vetor em um heap

- **Exercício 6.1-7 [Cormen]:** As folhas de um heap com n elementos são os elementos com índices $\lfloor n/2 \rfloor + 1, \lfloor n/2 \rfloor + 2, \dots, n$.

1	2	3	4	5	6	7	8	9	10
4	8	2	7	6	3	5	1	9	10

Transformando um vetor em um heap

- **Exercício 6.1-7 [Cormen]:** As folhas de um heap com n elementos são os elementos com índices $\lfloor n/2 \rfloor + 1, \lfloor n/2 \rfloor + 2, \dots, n$.

1	2	3	4	5	6	7	8	9	10
4	8	2	7	6	3	5	1	9	10

10

A[10]

Transformando um vetor em um heap

- Exercício 6.1-7 [Cormen]:** As folhas de um heap com n elementos são os elementos com índices $\lfloor n/2 \rfloor + 1, \lfloor n/2 \rfloor + 2, \dots, n$.

1	2	3	4	5	6	7	8	9	10
4	8	2	7	6	3	5	1	9	10



A[9]



A[10]

Transformando um vetor em um heap

- **Exercício 6.1-7 [Cormen]:** As folhas de um heap com n elementos são os elementos com índices $\lfloor n/2 \rfloor + 1, \lfloor n/2 \rfloor + 2, \dots, n$.

1	2	3	4	5	6	7	8	9	10
4	8	2	7	6	3	5	1	9	10

1

A[8]

9

A[9]

10

A[10]

Transformando um vetor em um heap

- Exercício 6.1-7 [Cormen]:** As folhas de um heap com n elementos são os elementos com índices $\lfloor n/2 \rfloor + 1, \lfloor n/2 \rfloor + 2, \dots, n$.

1	2	3	4	5	6	7	8	9	10
4	8	2	7	6	3	5	1	9	10

1

A[8]

9

A[9]

10

A[10]

5

A[7]

Transformando um vetor em um heap

- Exercício 6.1-7 [Cormen]:** As folhas de um heap com n elementos são os elementos com índices $\lfloor n/2 \rfloor + 1, \lfloor n/2 \rfloor + 2, \dots, n$.

1	2	3	4	5	6	7	8	9	10
4	8	2	7	6	3	5	1	9	10



A[8]



A[9]



A[10]



A[6]

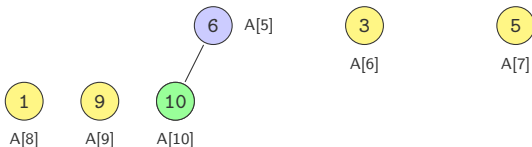


A[7]

Transformando um vetor em um heap

- Exercício 6.1-7 [Cormen]:** As folhas de um heap com n elementos são os elementos com índices $\lfloor n/2 \rfloor + 1, \lfloor n/2 \rfloor + 2, \dots, n$.

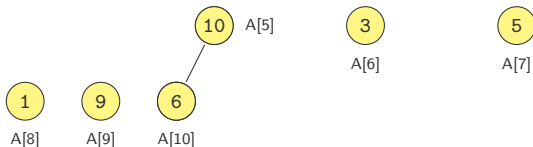
1	2	3	4	5	6	7	8	9	10
4	8	2	7	6	3	5	1	9	10



Transformando um vetor em um heap

- Exercício 6.1-7 [Cormen]:** As folhas de um heap com n elementos são os elementos com índices $\lfloor n/2 \rfloor + 1, \lfloor n/2 \rfloor + 2, \dots, n$.

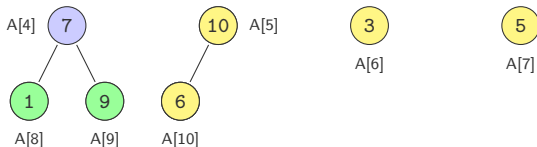
1	2	3	4	5	6	7	8	9	10
4	8	2	7	6	3	5	1	9	10



Transformando um vetor em um heap

- Exercício 6.1-7 [Cormen]:** As folhas de um heap com n elementos são os elementos com índices $\lfloor n/2 \rfloor + 1, \lfloor n/2 \rfloor + 2, \dots, n$.

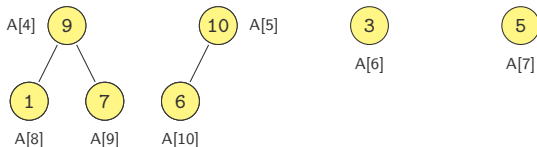
1	2	3	4	5	6	7	8	9	10
4	8	2	7	6	3	5	1	9	10



Transformando um vetor em um heap

- Exercício 6.1-7 [Cormen]:** As folhas de um heap com n elementos são os elementos com índices $\lfloor n/2 \rfloor + 1, \lfloor n/2 \rfloor + 2, \dots, n$.

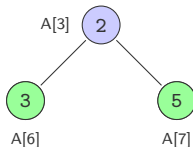
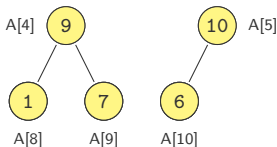
1	2	3	4	5	6	7	8	9	10
4	8	2	7	6	3	5	1	9	10



Transformando um vetor em um heap

- Exercício 6.1-7 [Cormen]:** As folhas de um heap com n elementos são os elementos com índices $\lfloor n/2 \rfloor + 1, \lfloor n/2 \rfloor + 2, \dots, n$.

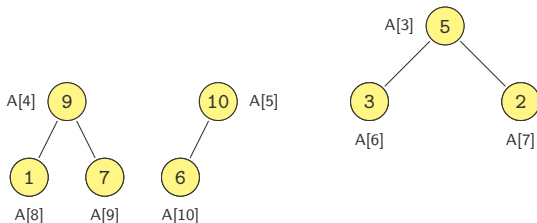
1	2	3	4	5	6	7	8	9	10
4	8	2	7	6	3	5	1	9	10



Transformando um vetor em um heap

- Exercício 6.1-7 [Cormen]:** As folhas de um heap com n elementos são os elementos com índices $\lfloor n/2 \rfloor + 1, \lfloor n/2 \rfloor + 2, \dots, n$.

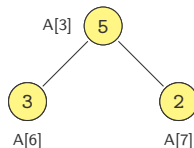
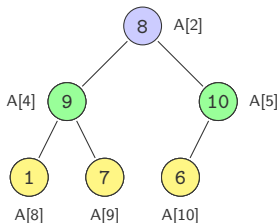
1	2	3	4	5	6	7	8	9	10
4	8	2	7	6	3	5	1	9	10



Transformando um vetor em um heap

- Exercício 6.1-7 [Cormen]:** As folhas de um heap com n elementos são os elementos com índices $\lfloor n/2 \rfloor + 1, \lfloor n/2 \rfloor + 2, \dots, n$.

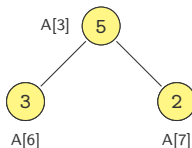
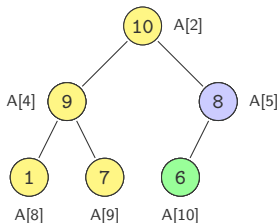
1	2	3	4	5	6	7	8	9	10
4	8	2	7	6	3	5	1	9	10



Transformando um vetor em um heap

- Exercício 6.1-7 [Cormen]:** As folhas de um heap com n elementos são os elementos com índices $\lfloor n/2 \rfloor + 1, \lfloor n/2 \rfloor + 2, \dots, n$.

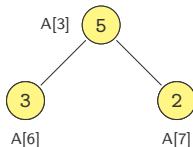
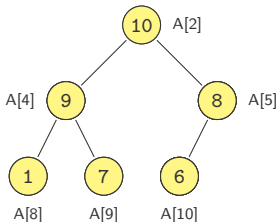
1	2	3	4	5	6	7	8	9	10
4	8	2	7	6	3	5	1	9	10



Transformando um vetor em um heap

- Exercício 6.1-7 [Cormen]:** As folhas de um heap com n elementos são os elementos com índices $\lfloor n/2 \rfloor + 1, \lfloor n/2 \rfloor + 2, \dots, n$.

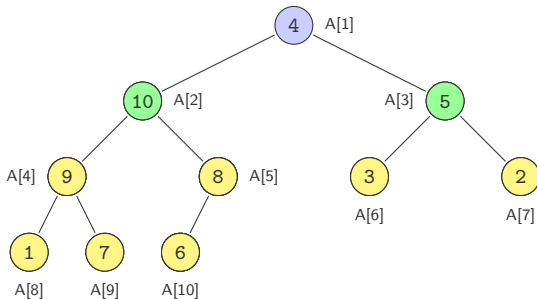
1	2	3	4	5	6	7	8	9	10
4	8	2	7	6	3	5	1	9	10



Transformando um vetor em um heap

- Exercício 6.1-7 [Cormen]:** As folhas de um heap com n elementos são os elementos com índices $\lfloor n/2 \rfloor + 1, \lfloor n/2 \rfloor + 2, \dots, n$.

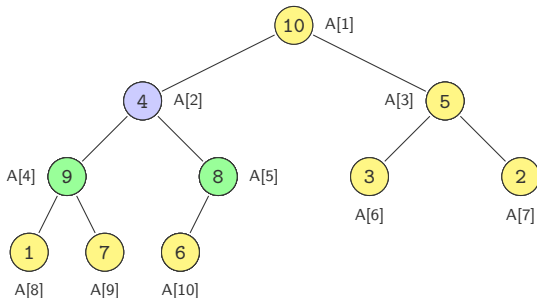
1	2	3	4	5	6	7	8	9	10
4	8	2	7	6	3	5	1	9	10



Transformando um vetor em um heap

- Exercício 6.1-7 [Cormen]:** As folhas de um heap com n elementos são os elementos com índices $\lfloor n/2 \rfloor + 1, \lfloor n/2 \rfloor + 2, \dots, n$.

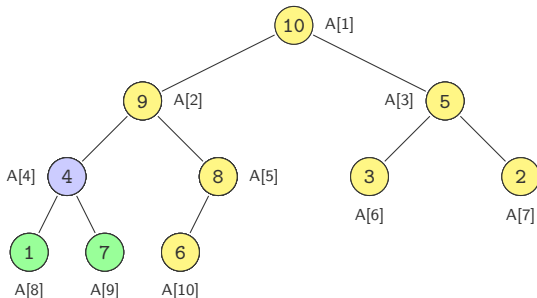
1	2	3	4	5	6	7	8	9	10
4	8	2	7	6	3	5	1	9	10



Transformando um vetor em um heap

- Exercício 6.1-7 [Cormen]:** As folhas de um heap com n elementos são os elementos com índices $\lfloor n/2 \rfloor + 1, \lfloor n/2 \rfloor + 2, \dots, n$.

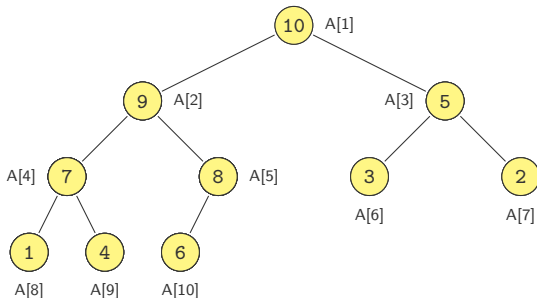
1	2	3	4	5	6	7	8	9	10
4	8	2	7	6	3	5	1	9	10



Transformando um vetor em um heap

- Exercício 6.1-7 [Cormen]:** As folhas de um heap com n elementos são os elementos com índices $\lfloor n/2 \rfloor + 1, \lfloor n/2 \rfloor + 2, \dots, n$.

1	2	3	4	5	6	7	8	9	10
4	8	2	7	6	3	5	1	9	10



Pseudocódigo – Construir heap máximo

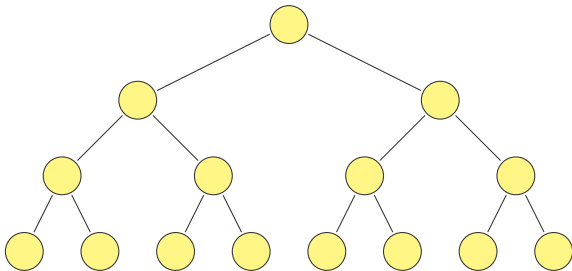
Pseudocódigo – Construir heap máximo

buildMaxHeap(A)

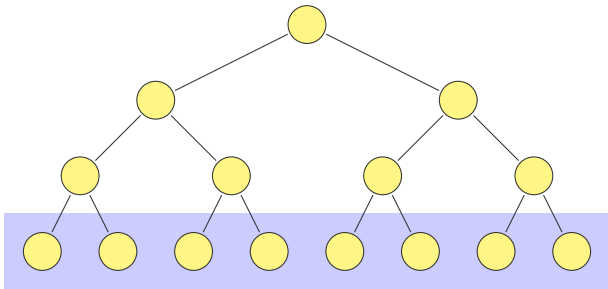
```
1  A.heapSize = A.length
2  for i =  $\lfloor A.length/2 \rfloor$  downto 1 then
3      maxFixDown(A, i)
```

Quanto tempo demora?

Tempo da construção para $n = 2^h - 1$

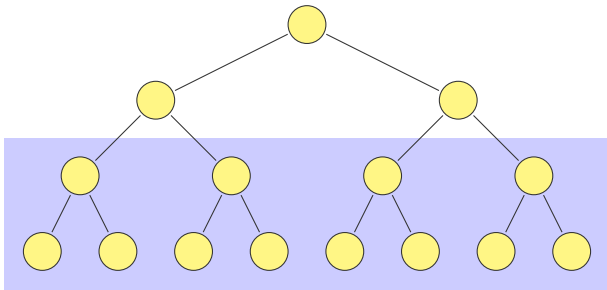


Tempo da construção para $n = 2^h - 1$



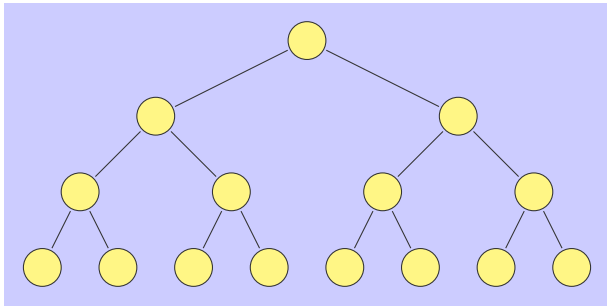
- Temos 2^{h-1} heaps de altura 1

Tempo da construção para $n = 2^h - 1$



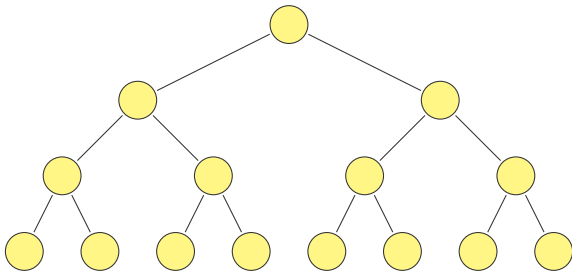
- Temos 2^{h-1} heaps de altura 1
- Temos 2^{h-2} heaps de altura 2

Tempo da construção para $n = 2^h - 1$



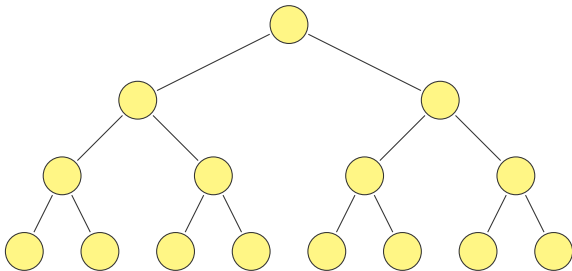
- Temos 2^{h-1} heaps de altura 1
- Temos 2^{h-2} heaps de altura 2
- Temos 2^{h-h} heaps de altura h

Tempo da construção para $n = 2^h - 1$



- Temos 2^{h-1} heaps de altura 1
- Temos 2^{h-2} heaps de altura 2
- Temos 2^{h-h} heaps de altura h
- Cada heap de altura k consome tempo $c \cdot k$

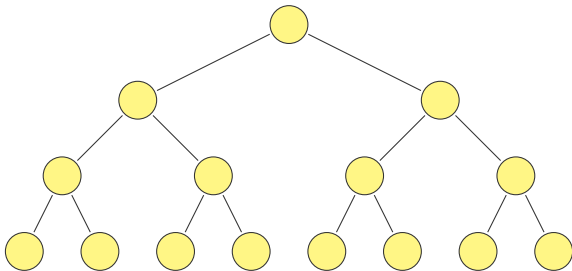
Tempo da construção para $n = 2^h - 1$



- Temos 2^{h-1} heaps de altura 1
- Temos 2^{h-2} heaps de altura 2
- Temos 2^{h-h} heaps de altura h
- Cada heap de altura k consome tempo $c \cdot k$

$$T(n) = \sum_{k=1}^h c \cdot k \cdot 2^{h-k}$$

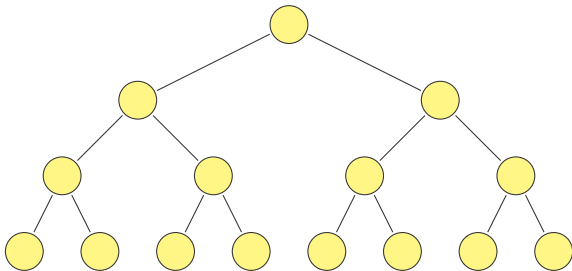
Tempo da construção para $n = 2^h - 1$



- Temos 2^{h-1} heaps de altura 1
- Temos 2^{h-2} heaps de altura 2
- Temos 2^{h-h} heaps de altura h
- Cada heap de altura k consome tempo $c \cdot k$

$$T(n) = \sum_{k=1}^h c \cdot k \cdot 2^{h-k} = \sum_{k=1}^h c \cdot k \cdot \frac{2^h}{2^k}$$

Tempo da construção para $n = 2^h - 1$



- Temos 2^{h-1} heaps de altura 1
- Temos 2^{h-2} heaps de altura 2
- Temos 2^{h-h} heaps de altura h
- Cada heap de altura k consome tempo $c \cdot k$

$$T(n) = \sum_{k=1}^h c \cdot k \cdot 2^{h-k} = \sum_{k=1}^h c \cdot k \cdot \frac{2^h}{2^k} = c \cdot 2^h \sum_{k=1}^h \frac{k}{2^k}$$

Cálculo de somatório auxiliar

Note que
$$\sum_{k=1}^h \frac{k}{2^k} = \frac{1}{2^1} + \frac{2}{2^2} + \frac{3}{2^3} + \cdots + \frac{h}{2^h}$$

Cálculo de somatório auxiliar

Note que $\sum_{k=1}^h \frac{k}{2^k} = \frac{1}{2^1} + \frac{2}{2^2} + \frac{3}{2^3} + \cdots + \frac{h}{2^h}$

Assim,

$$\sum_{k=1}^h \frac{k}{2^k} =$$

Cálculo de somatório auxiliar

Note que $\sum_{k=1}^h \frac{k}{2^k} = \frac{1}{2^1} + \frac{2}{2^2} + \frac{3}{2^3} + \cdots + \frac{h}{2^h}$

Assim,

$$\begin{aligned} \sum_{k=1}^h \frac{k}{2^k} &= \frac{1}{2} + \frac{1}{2^2} + \frac{1}{2^3} + \cdots + \frac{1}{2^h} \\ &\quad + \frac{1}{2^2} + \frac{1}{2^3} + \cdots + \frac{1}{2^h} \\ &\quad + \frac{1}{2^3} + \cdots + \frac{1}{2^h} \end{aligned}$$

Soma de PG finita:

$$S_n = \frac{a_1(1 - q^n)}{1 - q}$$

...

$$+ \frac{1}{2^h}$$

Cálculo de somatório auxiliar

Note que $\sum_{k=1}^h \frac{k}{2^k} = \frac{1}{2^1} + \frac{2}{2^2} + \frac{3}{2^3} + \cdots + \frac{h}{2^h}$

Assim,

$$\begin{aligned} \sum_{k=1}^h \frac{k}{2^k} &= \frac{1}{2} + \frac{1}{2^2} + \frac{1}{2^3} + \cdots + \frac{1}{2^h} = 1 - \frac{1}{2^h} < 1 \\ &\quad + \frac{1}{2^2} + \frac{1}{2^3} + \cdots + \frac{1}{2^h} \\ &\quad + \frac{1}{2^3} + \cdots + \frac{1}{2^h} \end{aligned}$$

Soma de PG finita:

$$S_n = \frac{a_1(1 - q^n)}{1 - q}$$

...

$$+ \frac{1}{2^h}$$

Cálculo de somatório auxiliar

Note que $\sum_{k=1}^h \frac{k}{2^k} = \frac{1}{2^1} + \frac{2}{2^2} + \frac{3}{2^3} + \cdots + \frac{h}{2^h}$

Assim,

$$\begin{aligned} \sum_{k=1}^h \frac{k}{2^k} &= \frac{1}{2} + \frac{1}{2^2} + \frac{1}{2^3} + \cdots + \frac{1}{2^h} = 1 - \frac{1}{2^h} < 1 \\ &\quad + \frac{1}{2^2} + \frac{1}{2^3} + \cdots + \frac{1}{2^h} = \frac{1}{2} - \frac{1}{2^h} < \frac{1}{2} \\ &\quad + \frac{1}{2^3} + \cdots + \frac{1}{2^h} \end{aligned}$$

...

$$+ \frac{1}{2^h}$$

Soma de PG finita:

$$S_n = \frac{a_1(1 - q^n)}{1 - q}$$

Cálculo de somatório auxiliar

Note que $\sum_{k=1}^h \frac{k}{2^k} = \frac{1}{2^1} + \frac{2}{2^2} + \frac{3}{2^3} + \cdots + \frac{h}{2^h}$

Assim,

$$\begin{aligned} \sum_{k=1}^h \frac{k}{2^k} &= \frac{1}{2} + \frac{1}{2^2} + \frac{1}{2^3} + \cdots + \frac{1}{2^h} = 1 - \frac{1}{2^h} < 1 \\ &\quad + \frac{1}{2^2} + \frac{1}{2^3} + \cdots + \frac{1}{2^h} = \frac{1}{2} - \frac{1}{2^h} < \frac{1}{2} \\ &\quad + \frac{1}{2^3} + \cdots + \frac{1}{2^h} = \frac{1}{4} - \frac{1}{2^h} < \frac{1}{4} \end{aligned}$$

...

$$+ \frac{1}{2^h}$$

Soma de PG finita:

$$S_n = \frac{a_1(1 - q^n)}{1 - q}$$

Cálculo de somatório auxiliar

Note que $\sum_{k=1}^h \frac{k}{2^k} = \frac{1}{2^1} + \frac{2}{2^2} + \frac{3}{2^3} + \cdots + \frac{h}{2^h}$

Assim,

$$\begin{aligned} \sum_{k=1}^h \frac{k}{2^k} &= \frac{1}{2} + \frac{1}{2^2} + \frac{1}{2^3} + \cdots + \frac{1}{2^h} = 1 - \frac{1}{2^h} < 1 \\ &\quad + \frac{1}{2^2} + \frac{1}{2^3} + \cdots + \frac{1}{2^h} = \frac{1}{2} - \frac{1}{2^h} < \frac{1}{2} \\ &\quad + \frac{1}{2^3} + \cdots + \frac{1}{2^h} = \frac{1}{4} - \frac{1}{2^h} < \frac{1}{4} \\ &\quad \dots = \dots \end{aligned}$$

Soma de PG finita:

$$S_n = \frac{a_1(1 - q^n)}{1 - q}$$

$$+ \frac{1}{2^h}$$

Cálculo de somatório auxiliar

Note que $\sum_{k=1}^h \frac{k}{2^k} = \frac{1}{2^1} + \frac{2}{2^2} + \frac{3}{2^3} + \cdots + \frac{h}{2^h}$

Assim,

$$\begin{aligned} \sum_{k=1}^h \frac{k}{2^k} &= \frac{1}{2} + \frac{1}{2^2} + \frac{1}{2^3} + \cdots + \frac{1}{2^h} = 1 - \frac{1}{2^h} < 1 \\ &\quad + \frac{1}{2^2} + \frac{1}{2^3} + \cdots + \frac{1}{2^h} = \frac{1}{2} - \frac{1}{2^h} < \frac{1}{2} \\ &\quad + \frac{1}{2^3} + \cdots + \frac{1}{2^h} = \frac{1}{4} - \frac{1}{2^h} < \frac{1}{4} \\ &\quad \dots = \dots \\ &\quad + \frac{1}{2^h} = \frac{1}{2^h} \leq \frac{1}{2^h} \end{aligned}$$

Ou seja, $\sum_{k=1}^h \frac{k}{2^k}$

Cálculo de somatório auxiliar

Note que $\sum_{k=1}^h \frac{k}{2^k} = \frac{1}{2^1} + \frac{2}{2^2} + \frac{3}{2^3} + \cdots + \frac{h}{2^h}$

Assim,

$$\begin{aligned} \sum_{k=1}^h \frac{k}{2^k} &= \frac{1}{2} + \frac{1}{2^2} + \frac{1}{2^3} + \cdots + \frac{1}{2^h} = 1 - \frac{1}{2^h} < 1 \\ &\quad + \frac{1}{2^2} + \frac{1}{2^3} + \cdots + \frac{1}{2^h} = \frac{1}{2} - \frac{1}{2^h} < \frac{1}{2} \\ &\quad + \frac{1}{2^3} + \cdots + \frac{1}{2^h} = \frac{1}{4} - \frac{1}{2^h} < \frac{1}{4} \\ &\quad \dots = \dots \\ &\quad + \frac{1}{2^h} = \frac{1}{2^h} \leq \frac{1}{2^h} \end{aligned}$$

Ou seja, $\sum_{k=1}^h \frac{k}{2^k} < 1 + \frac{1}{2} + \frac{1}{4} + \cdots + \frac{1}{2^h}$

Cálculo de somatório auxiliar

Note que $\sum_{k=1}^h \frac{k}{2^k} = \frac{1}{2^1} + \frac{2}{2^2} + \frac{3}{2^3} + \cdots + \frac{h}{2^h}$

Assim,

$$\begin{aligned}
 \sum_{k=1}^h \frac{k}{2^k} &= \frac{1}{2} + \frac{1}{2^2} + \frac{1}{2^3} + \cdots + \frac{1}{2^h} = 1 - \frac{1}{2^h} < 1 \\
 &\quad + \frac{1}{2^2} + \frac{1}{2^3} + \cdots + \frac{1}{2^h} = \frac{1}{2} - \frac{1}{2^h} < \frac{1}{2} \\
 &\quad + \frac{1}{2^3} + \cdots + \frac{1}{2^h} = \frac{1}{4} - \frac{1}{2^h} < \frac{1}{4} \\
 &\quad \dots = \dots \\
 &\quad + \frac{1}{2^h} = \frac{1}{2^h} \leq \frac{1}{2^h}
 \end{aligned}$$

Ou seja, $\sum_{k=1}^h \frac{k}{2^k} < 1 + \frac{1}{2} + \frac{1}{4} + \cdots + \frac{1}{2^h} < \sum_{k=0}^{\infty} \left(\frac{1}{2}\right)^k$

Cálculo de somatório auxiliar

Note que $\sum_{k=1}^h \frac{k}{2^k} = \frac{1}{2^1} + \frac{2}{2^2} + \frac{3}{2^3} + \cdots + \frac{h}{2^h}$

Assim,

$$\begin{aligned} \sum_{k=1}^h \frac{k}{2^k} &= \frac{1}{2} + \frac{1}{2^2} + \frac{1}{2^3} + \cdots + \frac{1}{2^h} = 1 - \frac{1}{2^h} < 1 \\ &\quad + \frac{1}{2^2} + \frac{1}{2^3} + \cdots + \frac{1}{2^h} = \frac{1}{2} - \frac{1}{2^h} < \frac{1}{2} \\ &\quad + \frac{1}{2^3} + \cdots + \frac{1}{2^h} = \frac{1}{4} - \frac{1}{2^h} < \frac{1}{4} \\ &\quad \dots = \dots \\ &\quad + \frac{1}{2^h} = \frac{1}{2^h} \leq \frac{1}{2^h} \end{aligned}$$

Ou seja, $\sum_{k=1}^h \frac{k}{2^k} < 1 + \frac{1}{2} + \frac{1}{4} + \cdots + \frac{1}{2^h} < \sum_{k=0}^{\infty} \left(\frac{1}{2}\right)^k = \frac{1}{1 - \frac{1}{2}}$

Cálculo de somatório auxiliar

Note que $\sum_{k=1}^h \frac{k}{2^k} = \frac{1}{2^1} + \frac{2}{2^2} + \frac{3}{2^3} + \cdots + \frac{h}{2^h}$

Assim,

$$\begin{aligned} \sum_{k=1}^h \frac{k}{2^k} &= \frac{1}{2} + \frac{1}{2^2} + \frac{1}{2^3} + \cdots + \frac{1}{2^h} = 1 - \frac{1}{2^h} < 1 \\ &\quad + \frac{1}{2^2} + \frac{1}{2^3} + \cdots + \frac{1}{2^h} = \frac{1}{2} - \frac{1}{2^h} < \frac{1}{2} \\ &\quad + \frac{1}{2^3} + \cdots + \frac{1}{2^h} = \frac{1}{4} - \frac{1}{2^h} < \frac{1}{4} \\ &\quad \dots = \dots \\ &\quad + \frac{1}{2^h} = \frac{1}{2^h} \leq \frac{1}{2^h} \end{aligned}$$

Ou seja, $\sum_{k=1}^h \frac{k}{2^k} < 1 + \frac{1}{2} + \frac{1}{4} + \cdots + \frac{1}{2^h} < \sum_{k=0}^{\infty} \left(\frac{1}{2}\right)^k = \frac{1}{1 - \frac{1}{2}} = 2.$

Tempo da construção para $n = 2^h - 1$

Ou seja,
$$T(n) = c \cdot 2^h \sum_{k=1}^h \frac{k}{2^k}$$

Portanto, construir um heap com n vértices leva tempo $O(n)$

Tempo da construção para $n = 2^h - 1$

Ou seja,
$$T(n) = c \cdot 2^h \sum_{k=1}^h \frac{k}{2^k} \leq c \cdot 2^h \cdot 2$$

Portanto, construir um heap com n vértices leva tempo $O(n)$

Tempo da construção para $n = 2^h - 1$

Ou seja,
$$T(n) = c \cdot 2^h \sum_{k=1}^h \frac{k}{2^k} \leq c \cdot 2^h \cdot 2 = O(2^h)$$

Portanto, construir um heap com n vértices leva tempo $O(n)$

Tempo da construção para $n = 2^h - 1$

Ou seja,
$$T(n) = c \cdot 2^h \sum_{k=1}^h \frac{k}{2^k} \leq c \cdot 2^h \cdot 2 = O(2^h) = O(n + 1)$$

Portanto, construir um heap com n vértices leva tempo $O(n)$

Tempo da construção para $n = 2^h - 1$

Ou seja,
$$T(n) = c \cdot 2^h \sum_{k=1}^h \frac{k}{2^k} \leq c \cdot 2^h \cdot 2 = O(2^h) = O(n + 1) = O(n).$$

Portanto, construir um heap com n vértices leva tempo $O(n)$

Aplicação de heap: Ordenação de vetor



Ordenação: Heapsort

Problema: ordenar um vetor de inteiros em ordem crescente.

Ordenação: Heapsort

Problema: ordenar um vetor de inteiros em ordem crescente.

Heapsort(A)

```
1  A.heapSize = A.length
2  BUILDMAXHEAP(A)
3  while A.heapSize > 1 do
4      aux = A[1]
5      A[1] = A[A.heapSize]
6      A[A.heapSize] = aux
7      A.heapSize = A.heapSize - 1
8      MAXFIXDOWN(A, 1)
```

Ordenação: Heapsort

Problema: ordenar um vetor de inteiros em ordem crescente.

Heapsort(A)

```
1  A.heapSize = A.length
2  BUILDMAXHEAP(A)
3  while A.heapSize > 1 do
4      aux = A[1]
5      A[1] = A[A.heapSize]
6      A[A.heapSize] = aux
7      A.heapSize = A.heapSize - 1
8      MAXFIXDOWN(A, 1)
```

Complexidade: $O(n \log n)$

Exercícios



Exercícios

- Implementar uma fila de prioridades em C++.
 - Implemente a fila de prioridades como uma classe chamada **PriorityQueue** usando **template de classe**.
 - **Dica:** Ao invés de implementar o heap usando um array clássico, use a classe **std::vector** nativa do C++. Esta classe pertence à biblioteca de templates STL do C++ e pode ser incluída através do cabeçalho `#include<vector>`
 - A vantagem de usar **std::vector** é que ele é um array “redimensionável”.
 - Um exemplo de template inicial que você pode usar para começar a implementação é dado no próximo slide. Lembre-se de implementar tudo no arquivo de cabeçalho .h

priority_queue.h

```
1  template <typename T>
2  class priority_queue {
3  private:
4      std::vector<T> m_queue; // array
5      size_t m_size;         // numero de elementos
6
7      void move_up(size_t i);
8      void move_down(size_t i);
9      void build_max_heap();
10
11 public:
12     priority_queue(const priority_queue&) = delete;
13     priority_queue& operator=(const priority_queue&) = delete;
14     priority_queue();
15     priority_queue(const std::vector<T>& v);
16     bool empty() const { return m_size == 0; }
17     size_t size() const { return m_size; }
18     const T& top() const;
19     void push(const T& val);
20     void pop();
21 };
```

FIM

