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 decrescente de prioridades.

Introdução

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

Introdução

Universidade Federal do Ceará
Campus Quixadá

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

Introdução

UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ

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

UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ

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

Introdução

UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ

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

UNIVERSIDADE FEDERAL DO CEARÁ CAMPUS QUIXADÁ

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

UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ

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

UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ

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:

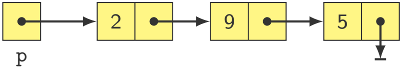
- Selecionar (acessar) o elemento com maior prioridade.
- Inserir um novo elemento.
- Remover o elemento com maior prioridade.

Outras operações auxiliares podem existir:

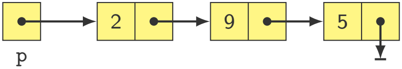
- Alterar a prioridade de um elemento.
- Construir a fila a partir de um dado grupo de elementos.

Como implementar esta estrutura de dados?

Primeira ideia — Lista não ordenada

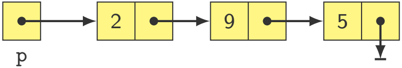


Primeira ideia — Lista não ordenada



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

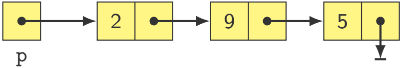
Primeira ideia — Lista não ordenada



- 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



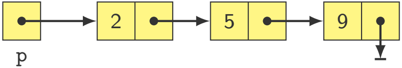
- Inserção e construção: o novo elemento pode ser colocado em qualquer posição conveniente dependendo do tipo de alocação utilizada: sequencial ou encadeada.

- 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: $O(n)$

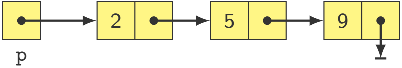
Segunda ideia — Lista ordenada

3	4	5	25	44



Segunda ideia — Lista ordenada

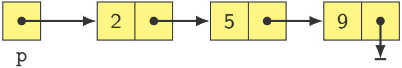
3	4	5	25	44



- Seleção e remoção: imediatas. Por quê?

Segunda ideia — Lista ordenada

UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ

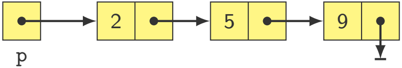


- Seleção e remoção: imediatas. Por quê?

- Inserção e alteração: percorrer a lista até encontrar posição correta do elemento. (pode ou não ocorrer deslocamento)

Segunda ideia — Lista ordenada

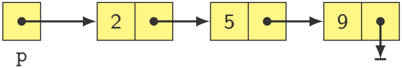
3	4	5	25	44



- 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

3	4	5	25	44



- 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

UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ

- 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: suponho 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 $A[1 \ldots n]$ 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: suponho que os índices do vetor são $1 \ldots n$ e não $0 \ldots n-1$.

- Observação: Segue imediatamente da definição que $A[1]$ é um elemento máximo do heap.

Definição de Heap — Propriedade de ordem

UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ

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

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

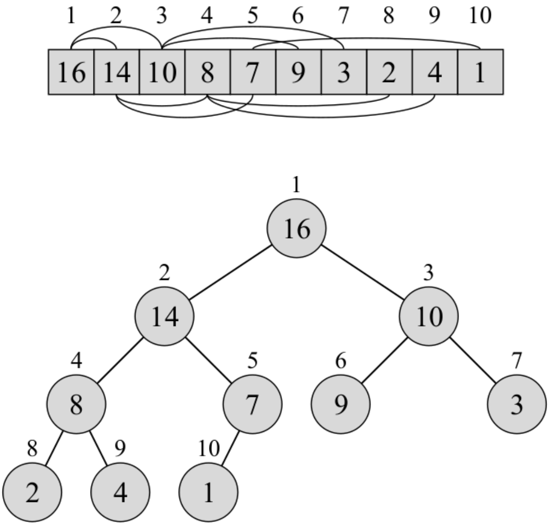
1	2	3	4	5	6	7	8	9	10	11	12	13	14
999	888	777	555	666	777	555	222	333	444	111	333	666	333

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

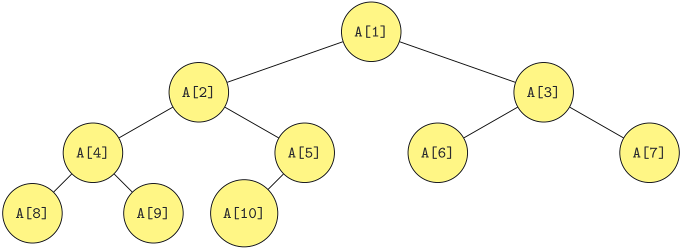
- Observação: Segue imediatamente da definição que $A[1]$ é um elemento máximo do heap.

- Assim como o Cormen et al. , suponho 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`.

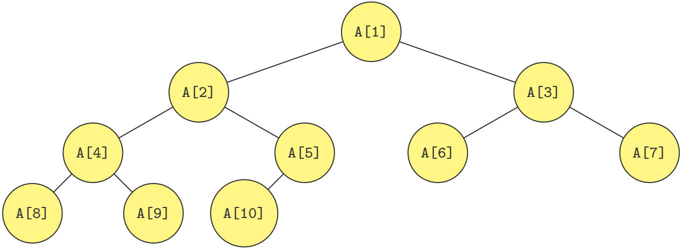
Definição de Heap — Propriedade estrutural



Ver heap como árvore binária completa

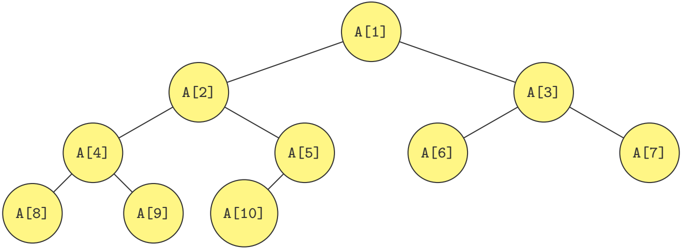


Ver heap como árvore binária completa



- O nó `A[1]` é a raiz da árvore.

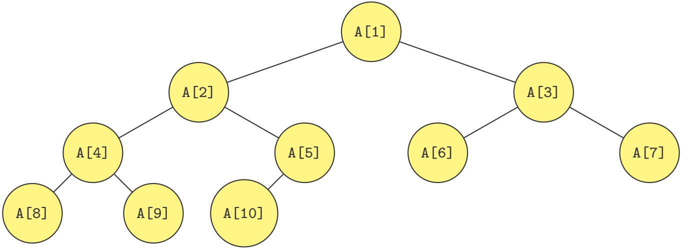
Ver heap como árvore binária completa



- O nó `A[1]` é a raiz da árvore.

Em relação a `A[i]`:

Ver heap como árvore binária completa

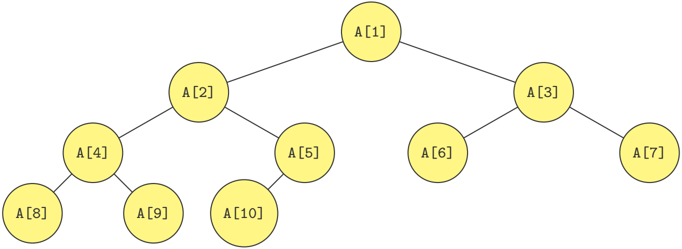


- O nó `A[1]` é a raiz da árvore.

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

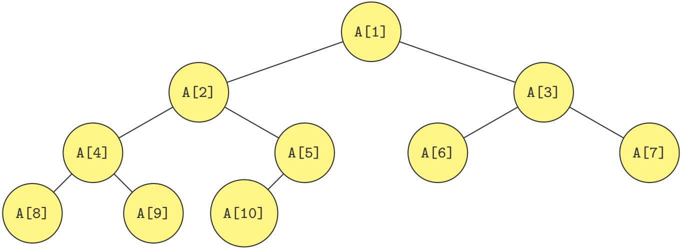


- O nó `A[1]` é a raiz da árvore.

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



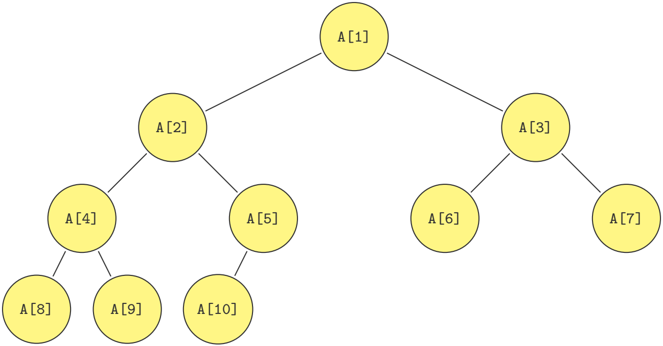
- O nó `A[1]` é a raiz da árvore.

Em relação a `A[i]`:

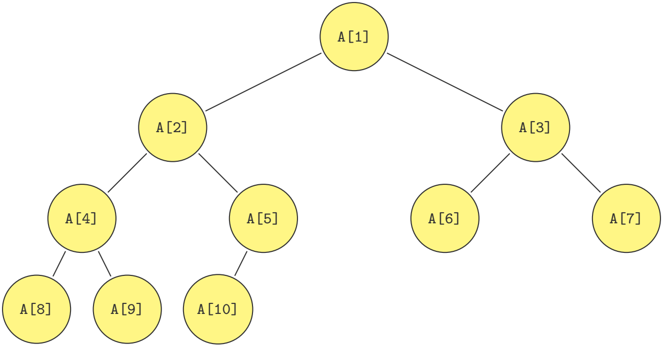
- o filho esquerdo é `A[2i]` e o filho direito é `A[2i+1]`
- o pai é `A[i/2]`

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

Ver heap como árvore binária completa

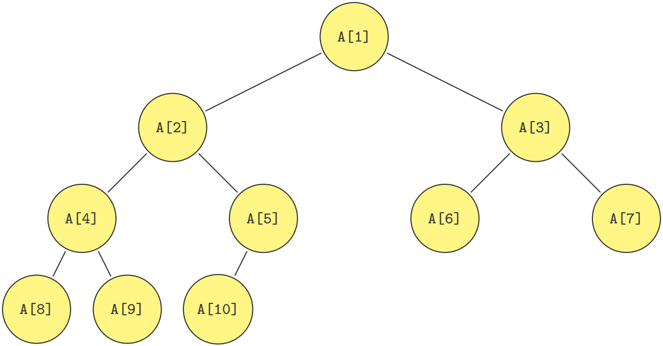


Ver heap como árvore binária completa



Da propriedade de ordem do heap, segue que:

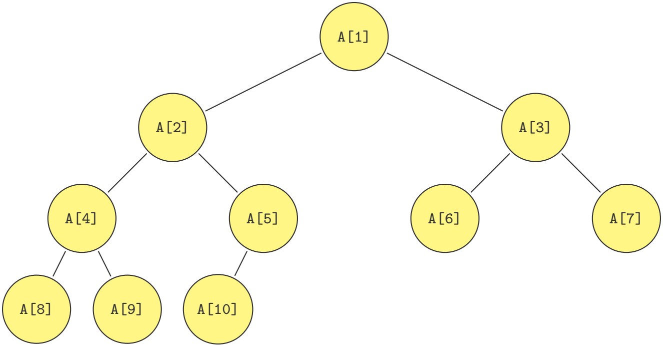
Ver heap como árvore binária completa



Da propriedade de ordem do heap, segue que:

- Os filhos são menores ou iguais ao pai.

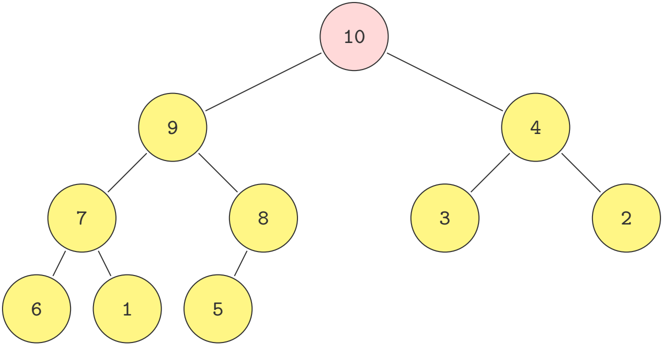
Ver heap como árvore binária completa



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:

Resumo

UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ

- 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;

Resumo

UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ

- 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);

Resumo

UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ

- 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 $2f \leq n$);

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:
  - 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 Priority Queue

Estamos interessados em implementar as seguintes operações do Tipo Abstrato de Dados Priority Queue:

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

Seleção do Máximo

Pseudocódigo – Seleção do máximo

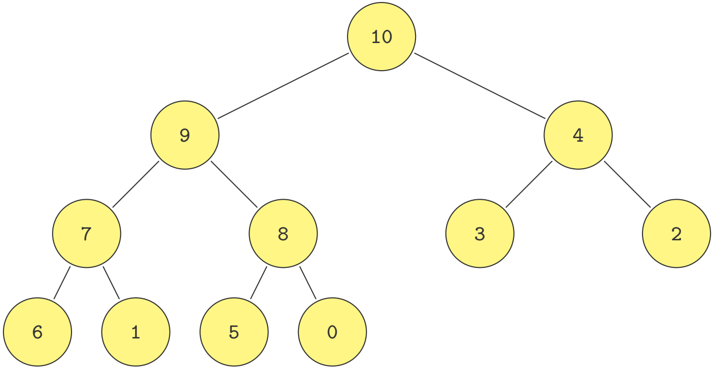
UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ

```
getMax(A)
if A.heapSize > 0
    return A[1]
else
    error "underflow error"
```

Complexidade: $O(1)$

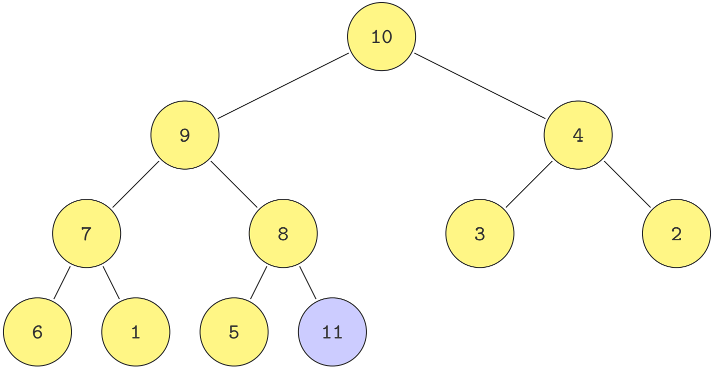
Alteração de Prioridades

Aumento da prioridade



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

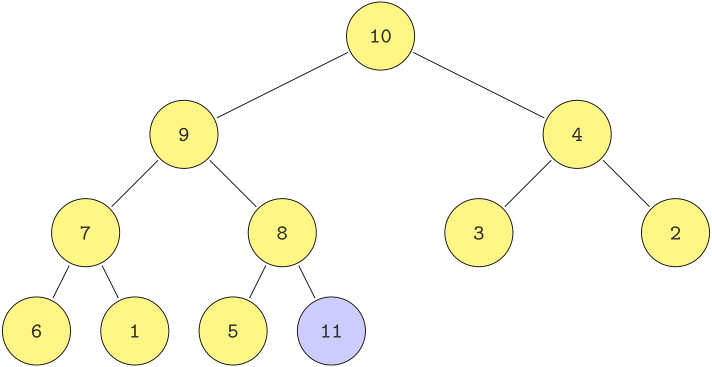
Aumento da prioridade



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

- É possível consertar? Como?

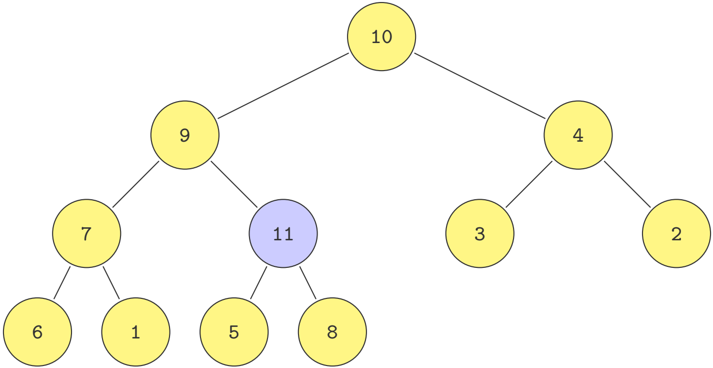
Aumento da prioridade



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

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

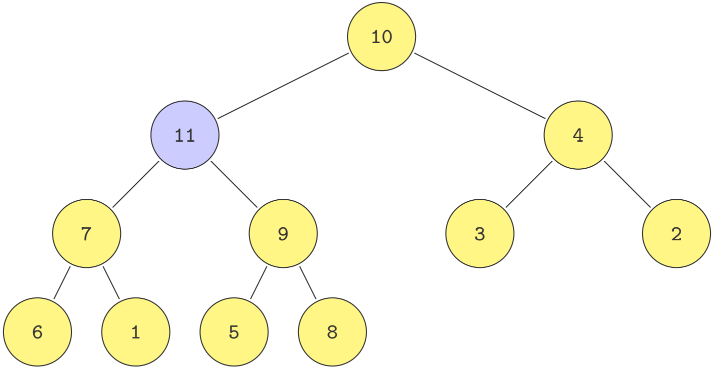
Aumento da prioridade



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

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

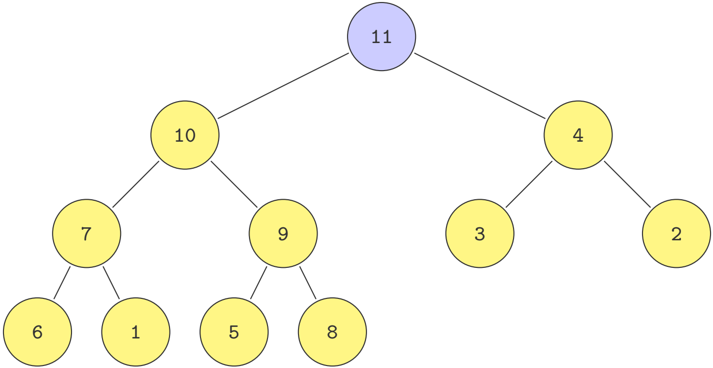
Aumento da prioridade



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

Pseudocódigo – Aumento da prioridade

UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ

Pseudocódigo – Aumento da prioridade

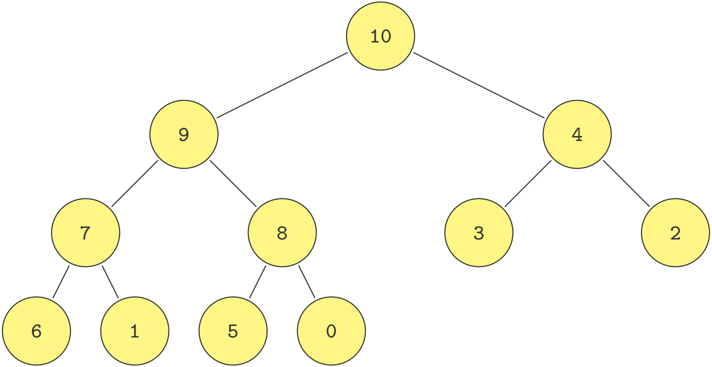
```
maxFixUp(A, i)
p = i/2
while p >= 1 and A[i] > A[p]
    aux = A[i]
    A[i] = A[p]
    A[p] = aux
    i = p
    p = p/2
```

Pseudocódigo – Aumento da prioridade

```
maxFixUp(A, i)
p = i/2
while p >= 1 and A[i] > A[p]
    aux = A[i]
    A[i] = A[p]
    A[p] = aux
    i = p
    p = p/2
```

Qual a complexidade de pior caso deste algoritmo?

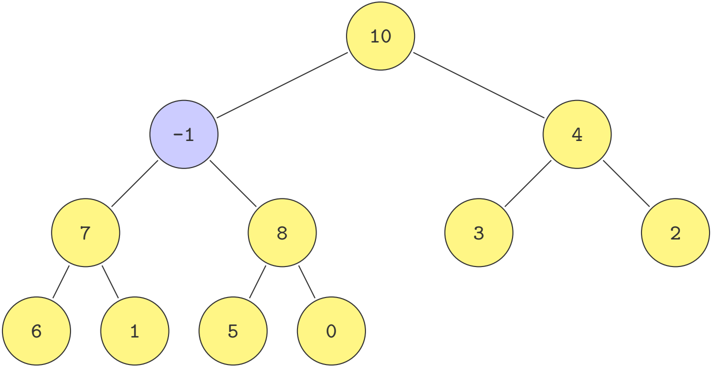
Redução da prioridade



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

Redução da prioridade

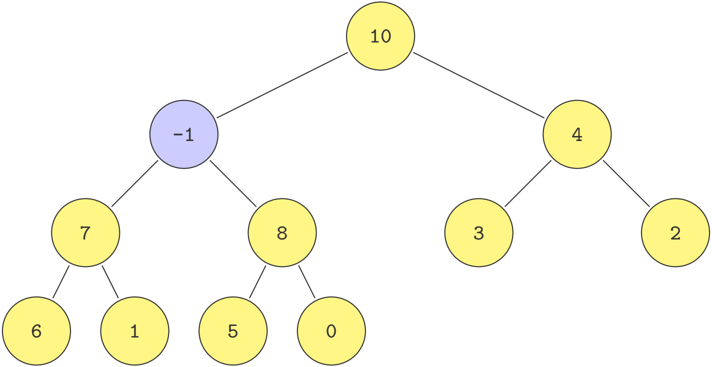
UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ



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

- É possível consertar? Como?

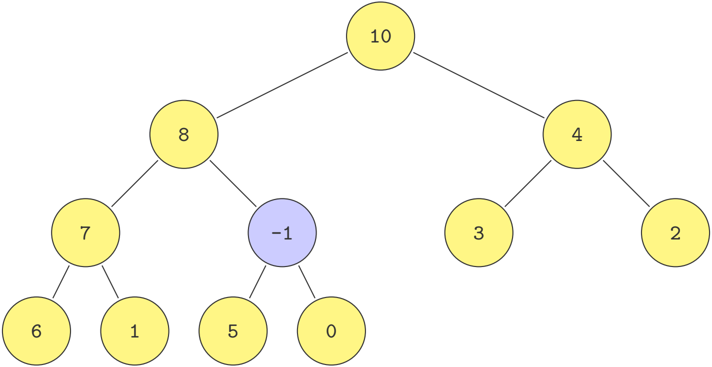
Redução da prioridade



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

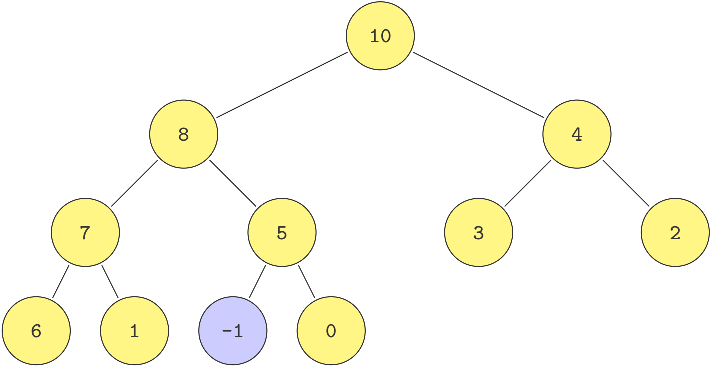
Redução da prioridade



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

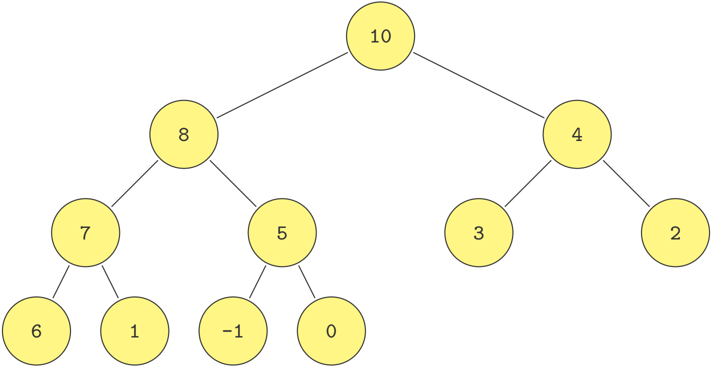
Redução da prioridade



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

Universidade Federal do Ceará
Campus Quixadá

```
maxHeap-decreaseKey(A, i, newKey)
if newKey > A[i] then
    error "invalid key"
A[i] = newKey
maxFixDown(A,i)
```

Pseudocódigo – Redução da prioridade

Universidade Federal do Ceará
Campus Quixadá

Pseudocódigo – Redução da prioridade

`maxFixDown(A, i)`

```
while 2i ≤ A.heapSize
    l = 2i
    r = 2i+1
    largest = i
    if l ≤ A.heapSize and A[l] > A[largest] then
        largest = l
    if r ≤ A.heapSize and A[r] > A[largest] then
        largest = r
    if largest ≠ i then
        aux = A[i]
        A[i] = A[largest]
        A[largest] = aux
        i = largest
    else
        break
```

Pseudocódigo – Redução da prioridade

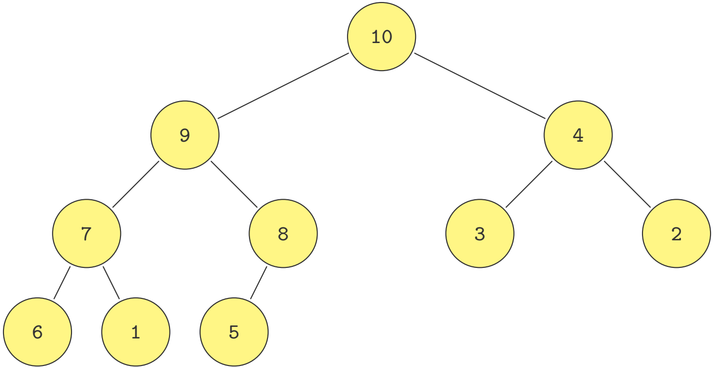
`maxFixDown(A, i)`

```
while 2i ≤ A.heapSize
    l = 2i
    r = 2i+1
    largest = i
    if l ≤ A.heapSize and A[l] > A[largest] then
        largest = l
    if r ≤ A.heapSize and A[r] > A[largest] then
        largest = r
    if largest ≠ i then
        aux = A[i]
        A[i] = A[largest]
        A[largest] = aux
        i = largest
    else
        break
```

Qual a complexidade de pior caso deste algoritmo?

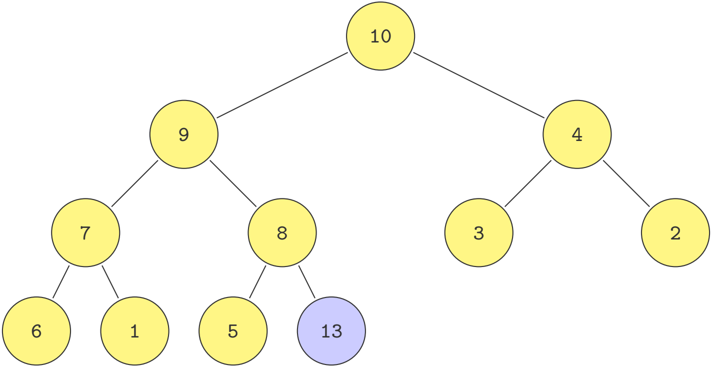
Inserção

Inserção de novo elemento



- Inserir um elemento com chave 13.

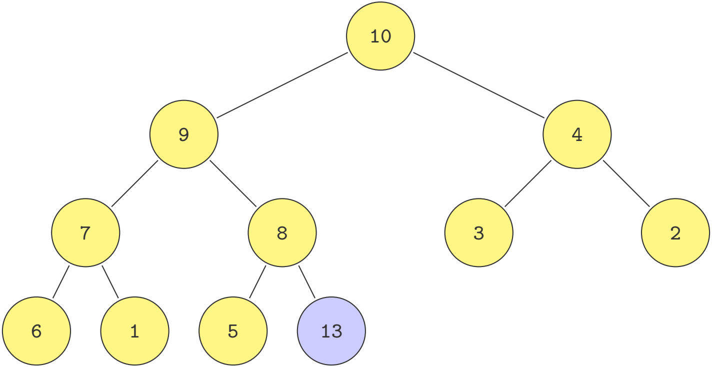
Inserção de novo elemento



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

- É possível consertar? Como?

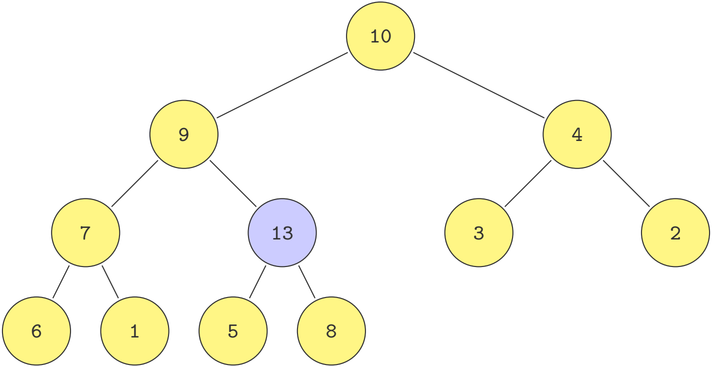
Inserção de novo elemento



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

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

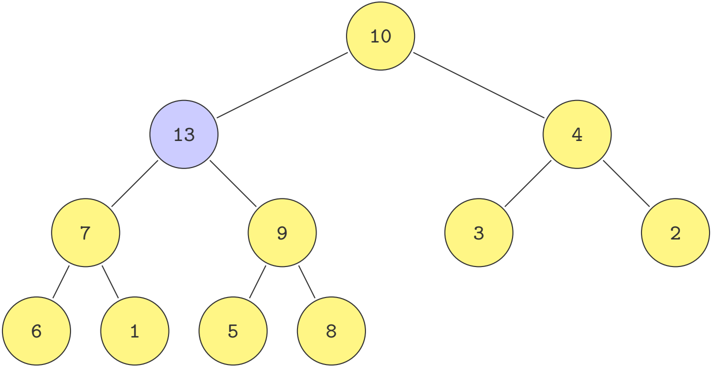
Inserção de novo elemento



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

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

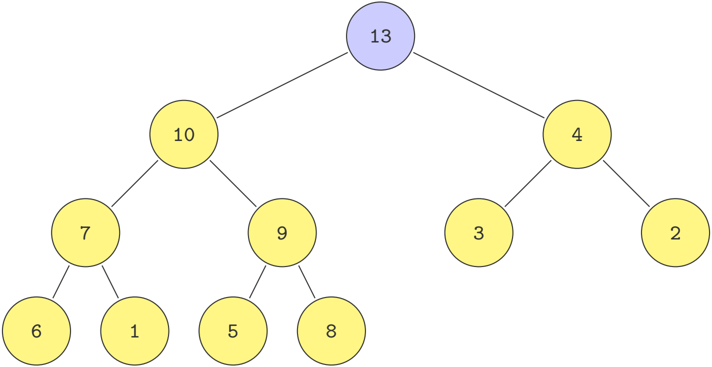
Inserção de novo elemento



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

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

Pseudocódigo – Inserção

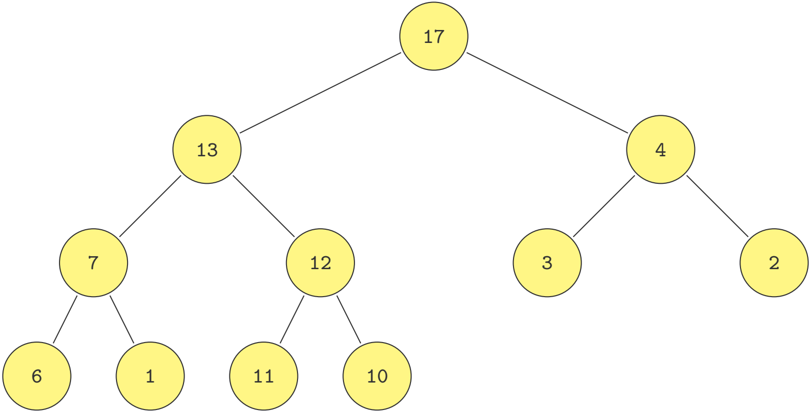
UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ

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

Qual a complexidade de pior caso deste algoritmo?

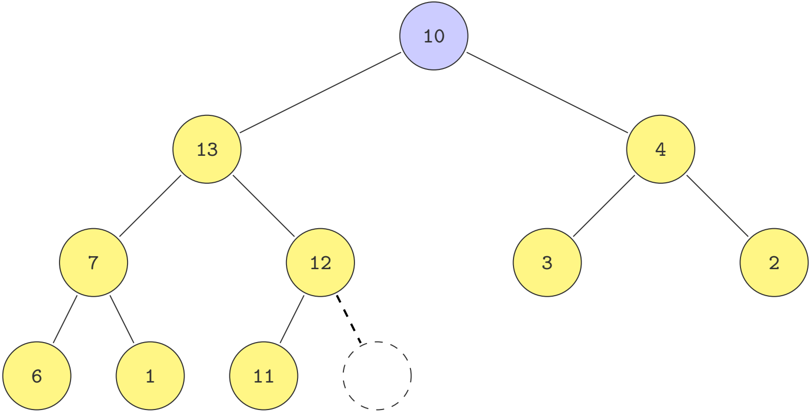
Remoção

Extrair o elemento de máxima prioridade



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

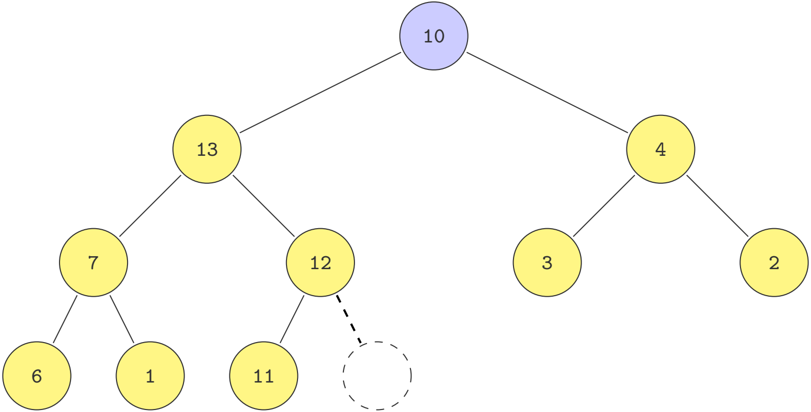
Extrair o elemento de máxima prioridade



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

- É possível consertar? Como?

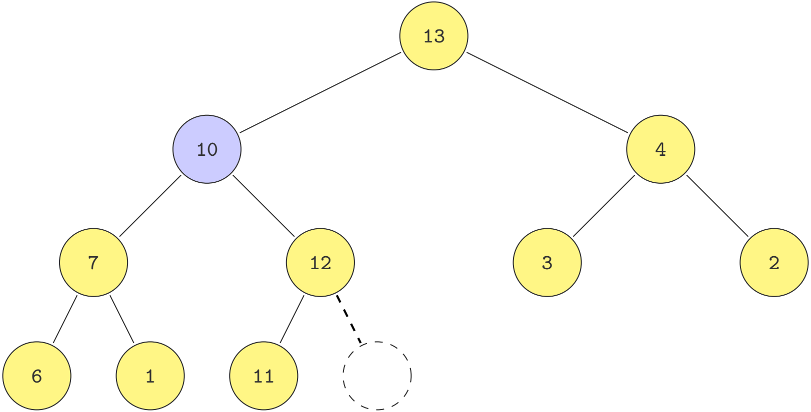
Extrair o elemento de máxima prioridade



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.

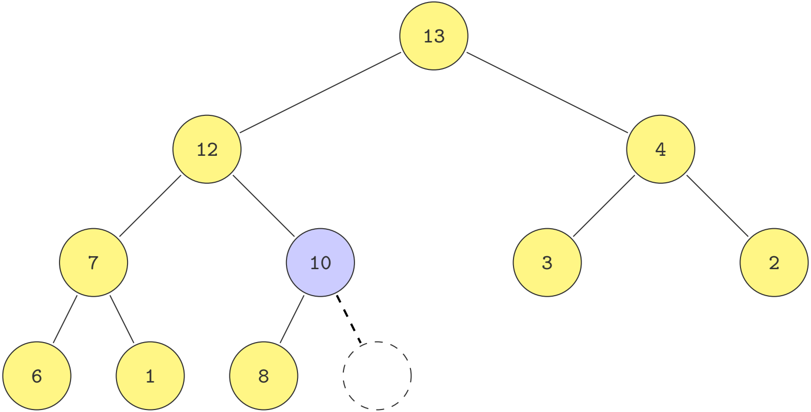
Extrair o elemento de máxima prioridade



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.

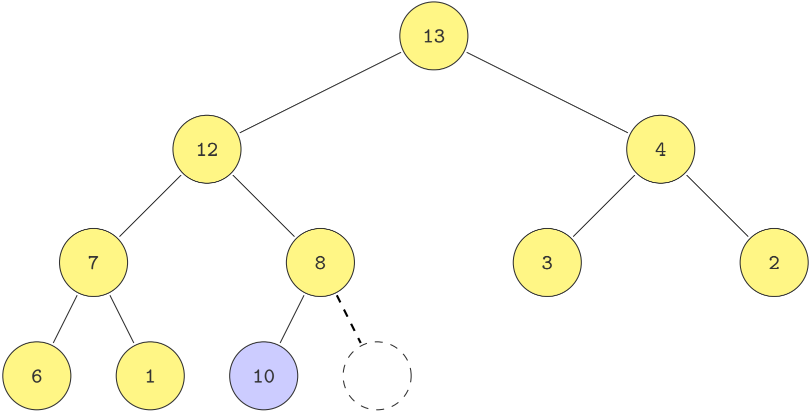
Extrair o elemento de máxima prioridade



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

Universidade Federal do Ceará
Campus Quixadá

```
maxHeap-extractMax(A)
if A.heapSize < 1 then
    error "heap underflow"
dados = A[1]
A[1] = A[A.heapSize]   ▷ copia elemento
A.heapSize = A.heapSize - 1
maxFixDown(A, 1)
return dados
```

Pseudocódigo – Extrair o máximo

UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ

```
maxHeap-extractMax(A)
if A.heapSize < 1 then
    error "heap underflow"
dados = A[1]
A[1] = A[A.heapSize]    ▷ copia elemento
A.heapSize = A.heapSize - 1
MAXFIXDOWN(A, 1)
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	$O(n)$	$O(n \lg n)$	$O(n)$

Construção de um heap máximo

Transformando um vetor em um heap

Universidade Federal do Ceará
Campus Quixadá

- 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

UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ

- 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

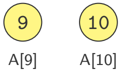
- Exercício 6.1-7 [Cormen]: As folhas de um heap com n elementos são os elementos com índices $\lfloor n/2 \rfloor + 1, \lfloor n/2 \rfloor + 2, \ldots, n$.

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 $\lfloor n/2 \rfloor + 1, \lfloor n/2 \rfloor + 2, \ldots, n$.

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 $\lfloor n/2 \rfloor + 1, \lfloor n/2 \rfloor + 2, \ldots, n$.

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

UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ

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

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

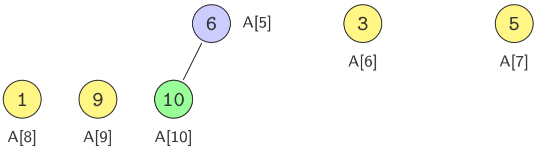
- Exercício 6.1-7 [Cormen]: As folhas de um heap com n elementos são os elementos com índices $\lfloor n/2 \rfloor + 1, \lfloor n/2 \rfloor + 2, \ldots, n$.

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 $\lfloor n/2 \rfloor + 1, \lfloor n/2 \rfloor + 2, \ldots, n$.

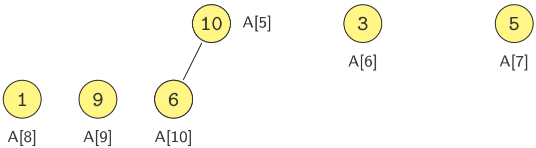
1	2	3	4	5	6	7	8	9	10
4	8	2	7	6	3	5	1	9	10



Transformando um vetor em um heap

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

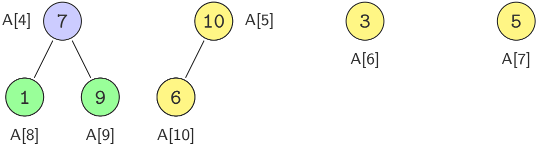
1	2	3	4	5	6	7	8	9	10
4	8	2	7	6	3	5	1	9	10



Transformando um vetor em um heap

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

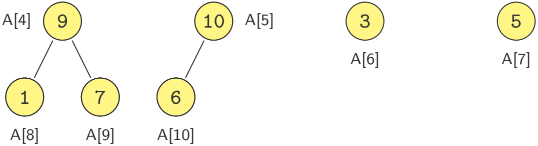
1	2	3	4	5	6	7	8	9	10
4	8	2	7	6	3	5	1	9	10



Transformando um vetor em um heap

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

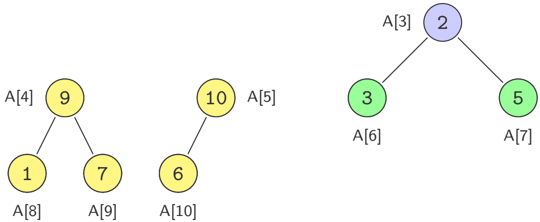
1	2	3	4	5	6	7	8	9	10
4	8	2	7	6	3	5	1	9	10



Transformando um vetor em um heap

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

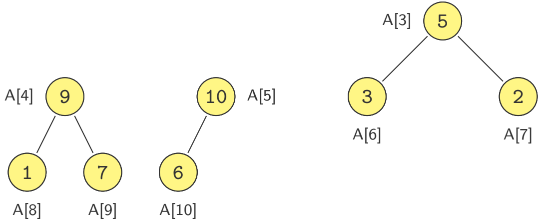
1	2	3	4	5	6	7	8	9	10
4	8	2	7	6	3	5	1	9	10



Transformando um vetor em um heap

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

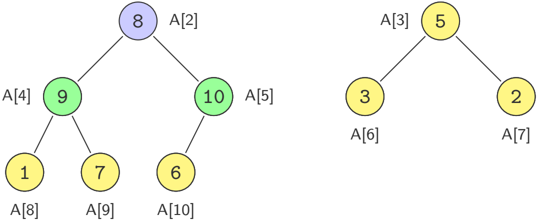
1	2	3	4	5	6	7	8	9	10
4	8	2	7	6	3	5	1	9	10



Transformando um vetor em um heap

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

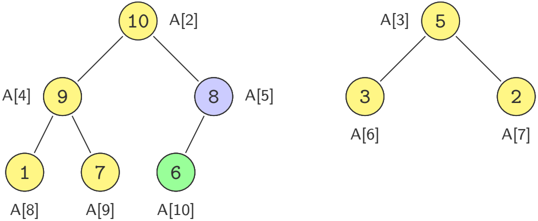
1	2	3	4	5	6	7	8	9	10
4	8	2	7	6	3	5	1	9	10



Transformando um vetor em um heap

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

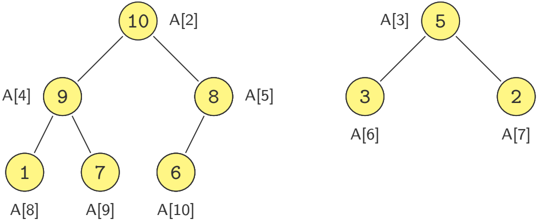
1	2	3	4	5	6	7	8	9	10
4	8	2	7	6	3	5	1	9	10



Transformando um vetor em um heap

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

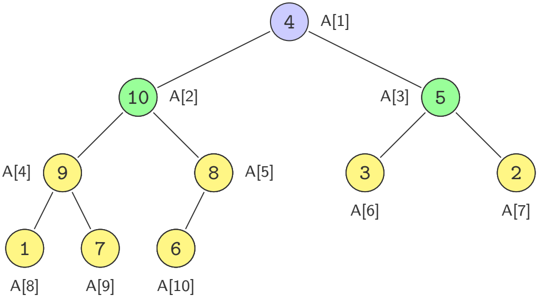
1	2	3	4	5	6	7	8	9	10
4	8	2	7	6	3	5	1	9	10



Transformando um vetor em um heap

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

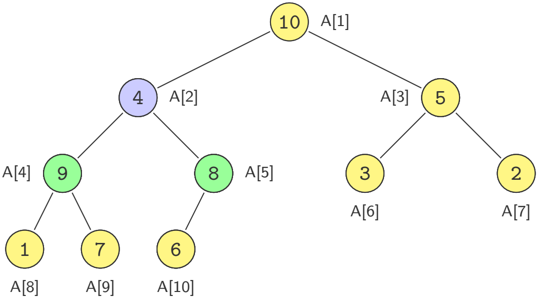
1	2	3	4	5	6	7	8	9	10
4	8	2	7	6	3	5	1	9	10



Transformando um vetor em um heap

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

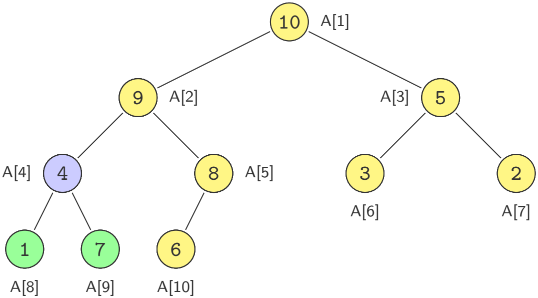
1	2	3	4	5	6	7	8	9	10
4	8	2	7	6	3	5	1	9	10



Transformando um vetor em um heap

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

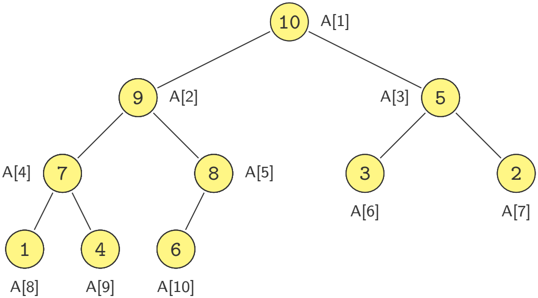
1	2	3	4	5	6	7	8	9	10
4	8	2	7	6	3	5	1	9	10



Transformando um vetor em um heap

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

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



Pseudocódigo – Construir heap máximo

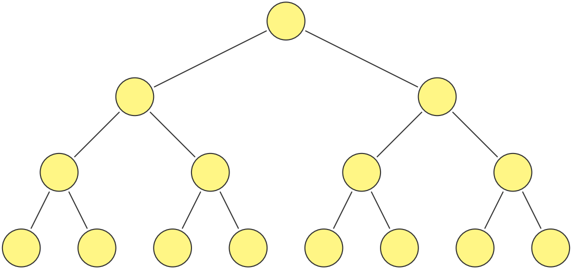
```
buildMaxHeap(A)
A.heapSize = A.length
for i = ⌊ A.length/2 ⌋ downto 1 then
    maxFixDown(A, i)
```

Pseudocódigo – Construir heap máximo

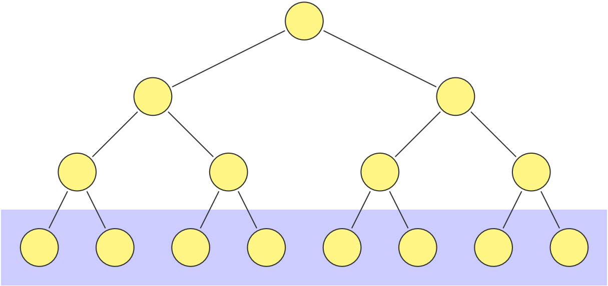
```
buildMaxHeap(A)
A.heapSize = A.length
for i = ⌊ A.length/2 ⌋ downto 1 then
    maxFixDown(A, i)
```

Quanto tempo demora?

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

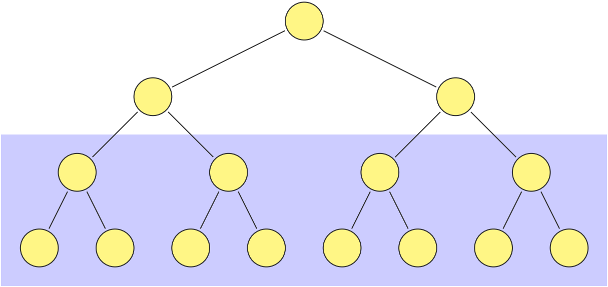


Tempo da construção para $n = 2^h - 1$



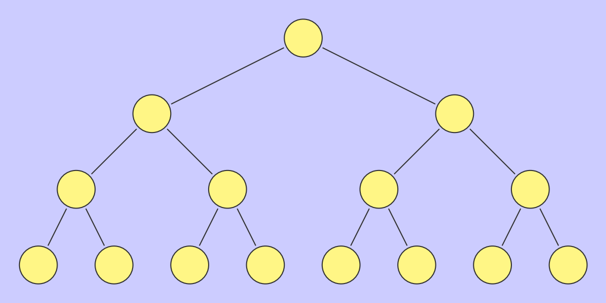
- Temos 2^{h-1} heaps de altura 1

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



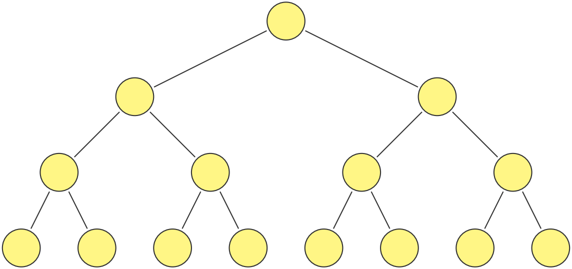
- Temos 2^{h-1} heaps de altura 1
- Temos 2^{h-2} heaps de altura 2

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



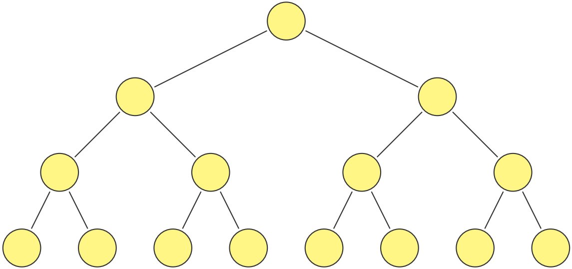
- Temos 2^{h-1} heaps de altura 1
- Temos 2^{h-2} heaps de altura 2
- Temos 2^{h-h} heaps de altura h

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 $c \cdot k$

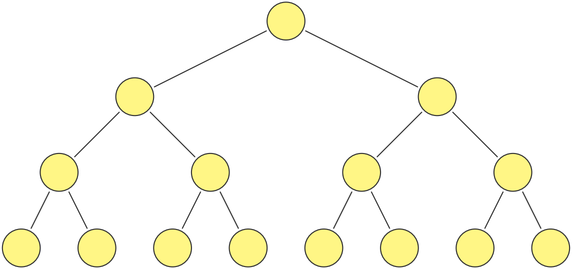
Tempo da construção para $n = 2^h - 1$



- 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 $c \cdot k$

$$\sum_{k=1}^{h} c \cdot k \cdot 2^{h-k}$$

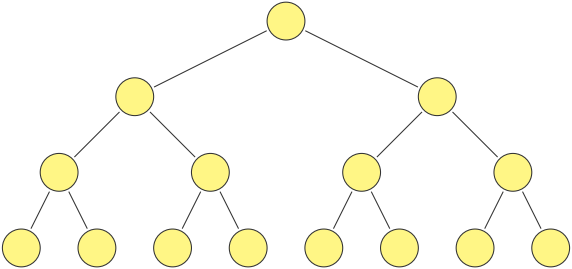
Tempo da construção para $n = 2^h - 1$



- 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 $c \cdot k$

$$\sum_{k=1}^{h} c \cdot k \cdot 2^{h-k} \; = \; \sum_{k=1}^{h} c \cdot k \cdot \frac{2^h}{2^k}$$

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 $c \cdot k$

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

Cálculo de somatório auxiliar

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

Cálculo de somatório auxiliar

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

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,

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

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,

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

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,

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

Soma de PG finita:

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

Cálculo de somatório auxiliar

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

Assim,

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

Soma de PG finita:

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

Cálculo de somatório auxiliar

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

Assim,

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

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,

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

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

Cálculo de somatório auxiliar

UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ

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

Assim,

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

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

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,

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

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,

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

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,

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

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 $n = 2^h - 1$

UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ

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

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

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

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

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

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

UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ

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

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

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

UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ

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

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

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

Universidade Federal do Ceará
Campus Quixadá

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

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

Aplicação de heap: Ordenação de vetor

Ordenação: Heapsort

UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ

Problema: ordenar um vetor de inteiros em ordem crescente.

Ordenação: Heapsort

Problema: ordenar um vetor de inteiros em ordem crescente.

```
Heapsort(A)
A.heapSize = A.length
BuildMaxHeap(A)
while A.heapSize > 1 do
    aux = A[1]
    A[1] = A[A.heapSize]
    A[A.heapSize] = aux
    A.heapSize = A.heapSize - 1
    maxFixDown(A, 1)
```

Ordenação: Heapsort

Problema: ordenar um vetor de inteiros em ordem crescente.

```
Heapsort(A)
A.heapSize = A.length
BuildMaxHeap(A)
while A.heapSize > 1 do
    aux = A[1]
    A[1] = A[A.heapSize]
    A[A.heapSize] = aux
    A.heapSize = A.heapSize - 1
    maxFixDown(A, 1)
```

Complexidade: $O(n \log n)$

Aplicação de heaps: Seleção do k-ésimo maior

Seleção do k-ésimo maior

Problema: Dado um vetor $A[1..n]$ com n elementos, encontre o k-ésimo maior elemento nesse vetor.

Exemplos:

- Input: $A = \{20, 15, 18, 8, 10, 5, 17\}, k = 4$
  Output: 15

- Input: $A = \{100, 50, 80, 10, 25, 20, 75\}, k = 2$
  Output : 80

Seleção do k-ésimo maior

UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ

Abordagem 1:

- Ordene os elementos do vetor $A[1..n]$ em ordem descrescente e pegue o elemento na posição k.
- Qual a complexidade?

Seleção do k-ésimo maior

Abordagem 1:

- Ordene os elementos do vetor $A[1..n]$ em ordem descrescente e pegue o elemento na posição k.
- Qual a complexidade? $O(n) + O(n \lg n)$

Seleção do k-ésimo maior

Abordagem 1:

- Ordene os elementos do vetor $A[1..n]$ em ordem descrescente e pegue o elemento na posição k.
- Qual a complexidade? $O(n) + O(n \lg n)$

Abordagem 2:

- Transforme $A[1..n]$ em um max-heap. Então, extraia o máximo do heap exatamente k vezes.
- Qual a complexidade?

Seleção do k-ésimo maior

Universidade Federal do Ceará
Campus Quixadá

Abordagem 1:

- Ordene os elementos do vetor $A[1..n]$ em ordem descrescente e pegue o elemento na posição k.
- Qual a complexidade? $O(n) + O(n \lg n)$

Abordagem 2:

- Transforme $A[1..n]$ em um max-heap. Então, extraia o máximo do heap exatamente k vezes.
- Qual a complexidade? $O(n) + O(k \lg n)$

Seleção do k-ésimo maior

UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ

Abordagem 3:

- Transforme $A[1..n]$ em um max-heap.
- Note que: um elemento x no i-ésimo nível tem $i-1$ ancestrais. Pela propriedade de max-heaps, esses ancestrais são maiores que x. Isso implica que x não pode estar entre os maiores $i-1$ elementos do heap.
- Usando esta propriedade, concluímos que o k-ésimo maior elemento pode estar em um nível no máximo k.
- Reduzimos o tamanho do heap máximo de forma que ele tenha apenas k níveis.
- Então obtemos o k-ésimo maior elemento pela nossa estratégia anterior de extrair o elemento máximo k vezes.
- Qual a complexidade?

Seleção do k-ésimo maior

UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ

Abordagem 3:

- Transforme $A[1..n]$ em um max-heap.
- Note que: um elemento x no i-ésimo nível tem $i-1$ ancestrais. Pela propriedade de max-heaps, esses ancestrais são maiores que x. Isso implica que x não pode estar entre os maiores $i-1$ elementos do heap.
- Usando esta propriedade, concluímos que o k-ésimo maior elemento pode estar em um nível no máximo k.
- Reduzimos o tamanho do heap máximo de forma que ele tenha apenas k níveis.
- Então obtemos o k-ésimo maior elemento pela nossa estratégia anterior de extrair o elemento máximo k vezes.
- Qual a complexidade? $O(n) + O(k^2)$

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

UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ

```
template <typename T>
class priority_queue {
private:
    std::vector<T> m_queue; // array
    size_t m_size;          // numero de elementos

    void move_up(size_t i);
    void move_down(size_t i);
    void build_max_heap();

public:
    priority_queue(const priority_queue&) = delete;
    priority_queue& operator=(const priority_queue&) = delete;
    priority_queue();
    priority_queue(const std::vector<T>& v);
    bool empty() const { return m_size == 0; }
    size_t size() const { return m_size; }
    const T& top() const;
    void push(const T& val);
    void pop();
};
```

UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ

FIM