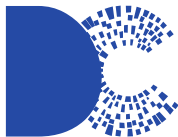




UNIVERSIDADE
FEDERAL DE
SERGIPE



DEPARTAMENTO
DE COMPUTAÇÃO

Árvore *heap*

Estruturas de Dados

Bruno Prado

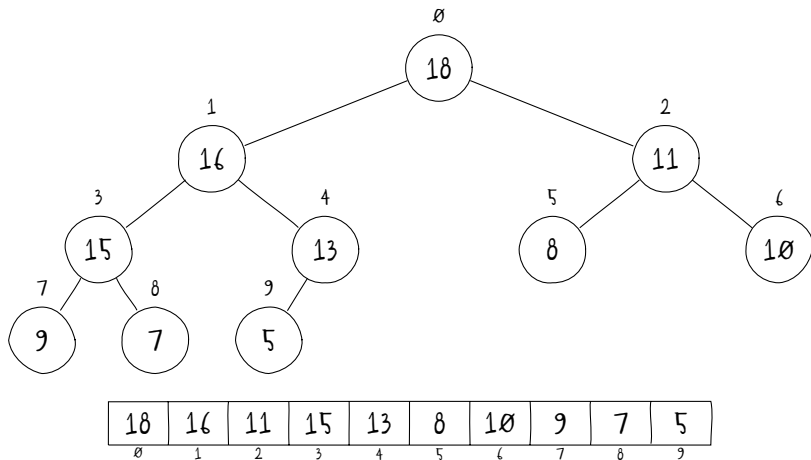
Departamento de Computação / UFS

Introdução

- ▶ O que é uma árvore *heap*?
 - ▶ Árvore binária de prioridade
 - ▶ Representação implícita em vetor
 - ▶ Percursos por indexação dos nós

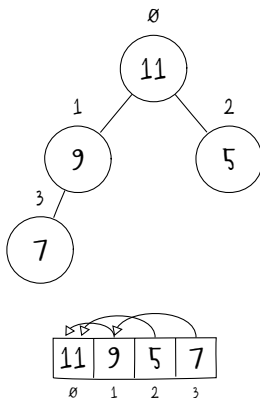
Introdução

► Árvore binária *heap*



Árvore heap

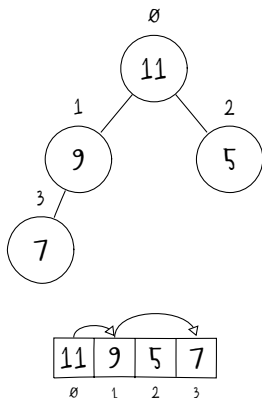
- Representação e indexação
 - Nó pai



$$Pai(i) = \frac{i-1}{2}$$

Árvore *heap*

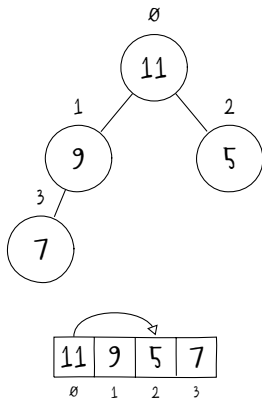
- Representação e indexação
 - Nó filho esquerdo



$$\text{Esquerdo}(i) = 2i + 1$$

Árvore *heap*

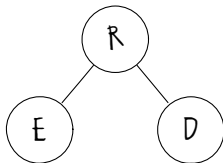
- Representação e indexação
 - Nó filho direito



$$\text{Direito}(i) = 2i + 2$$

Árvore *heap*

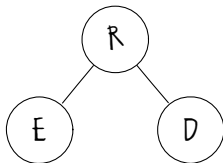
- ▶ Tipos de árvores *heap*
 - ▶ *Heap* mínimo



Propriedade $R \leq E$ e $R \leq D$

Árvore *heap*

- ▶ Tipos de árvores *heap*
 - ▶ *Heap* máximo



Propriedade $R \geq E$ e $R \geq D$

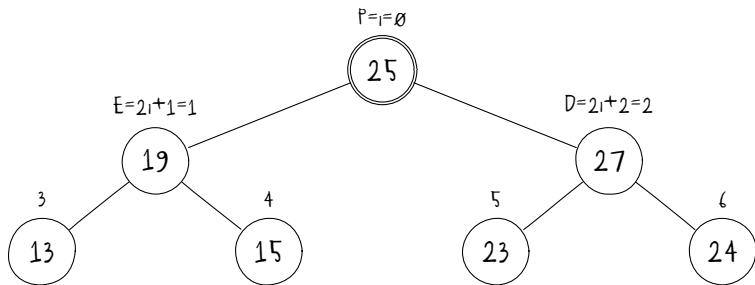
Árvore *heap*

► Aplicação da propriedade de *heap*

```
1 // Padrão de tipos por tamanho
2 #include <stdint.h>
3 // Heapify recursivo
4 void heapify(int32_t* V, uint32_t T, uint32_t i) {
5     // Declaração dos índices
6     uint32_t P = i, E = esquerdo(i), D = direito(i);
7     // Filho da esquerda é maior
8     if(E < T && V[E] > V[P])
9         P = E;
10    // Filho da direita é maior
11    if(D < T && V[D] > V[P])
12        P = D;
13    // Troca e chamada recursiva
14    if(P != i) {
15        trocar(V, P, i);
16        heapify(V, T, P);
17    }
18 }
19 ...
```

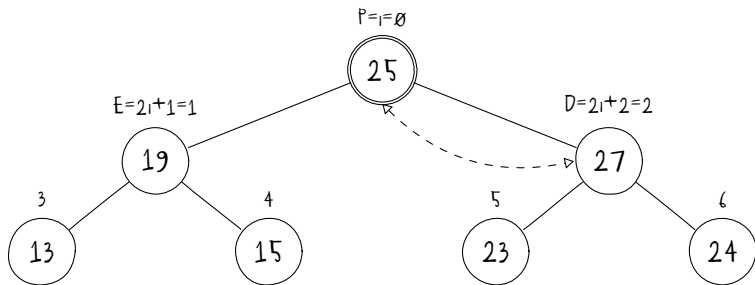
Árvore *heap*

- ▶ Aplicação da propriedade de *heap*
 - ▶ Procedimento *heapify* na raiz



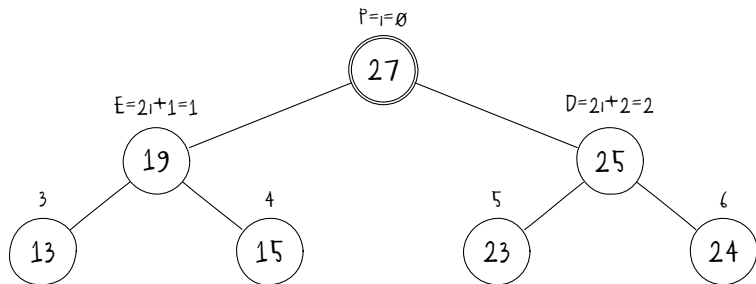
Árvore *heap*

- ▶ Aplicação da propriedade de *heap*
 - ▶ Procedimento *heapify* na raiz



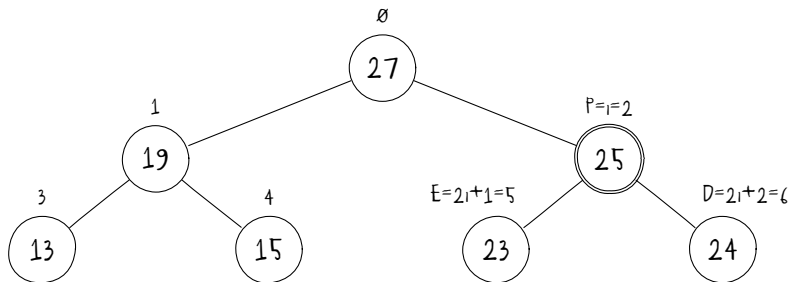
Árvore *heap*

- ▶ Aplicação da propriedade de *heap*
 - ▶ Procedimento *heapify* na raiz



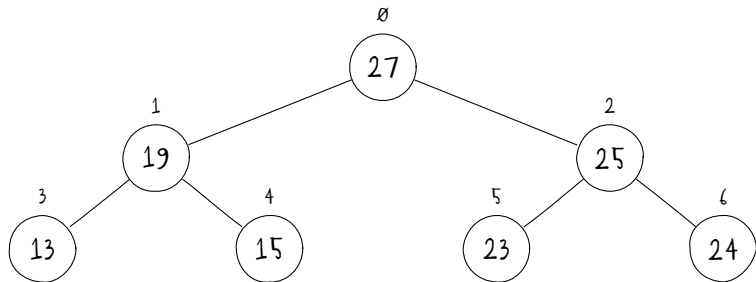
Árvore heap

- ▶ Aplicação da propriedade de *heap*
 - ▶ Procedimento *heapify* na raiz



Árvore *heap*

- ▶ Aplicação da propriedade de *heap*
 - ▶ Procedimento *heapify* finalizado

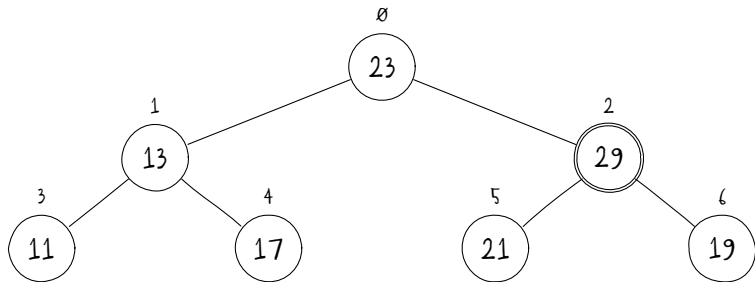


Árvore *heap*

- ▶ Complexidade do procedimento *heapify*
 - ▶ A altura h da árvore é $\log_2 n$
 - ▶ Espaço e tempo: $\Omega(1)$ e $O(h)$

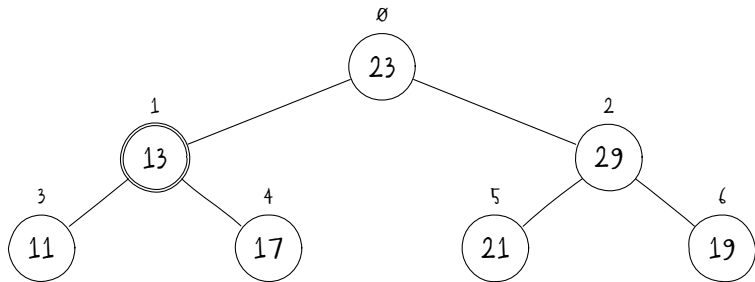
Árvore *heap*

- ▶ Construção da árvore *heap*
 - ▶ Começa pelo último nó com filhos
 - ▶ *Heapify* no índice $i = \frac{(Tamanho-1)-1}{2}$



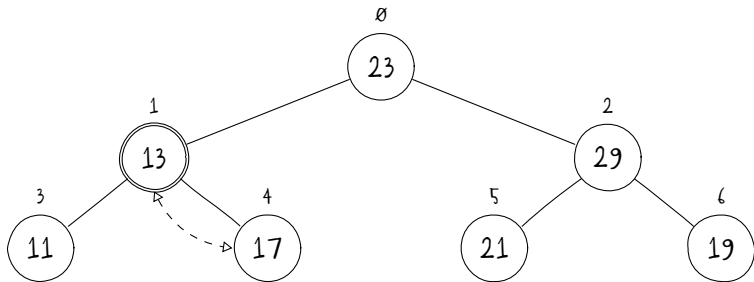
Árvore *heap*

- ▶ Construção da árvore *heap*
 - ▶ O índice é decrementado até atingir a raiz
 - ▶ *Heapify* no índice $i = \frac{(Tamanho-1)-1}{2}$



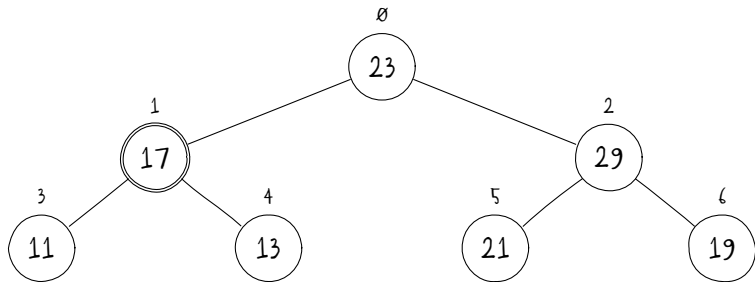
Árvore *heap*

- ▶ Construção da árvore *heap*
 - ▶ O índice é decrementado até atingir a raiz
 - ▶ *Heapify* no índice $i = \frac{(Tamanho-1)-1}{2}$



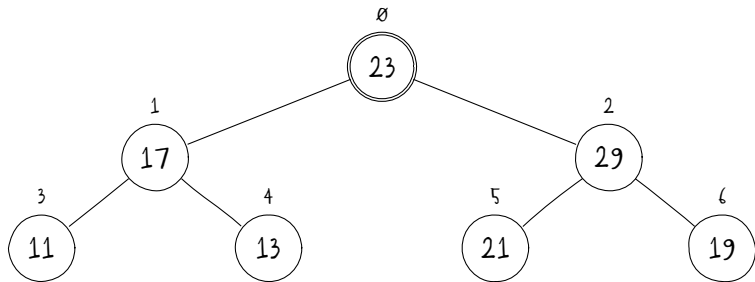
Árvore *heap*

- ▶ Construção da árvore *heap*
 - ▶ O índice é decrementado até atingir a raiz
 - ▶ *Heapify* no índice $i = \frac{(Tamanho-1)-1}{2}$



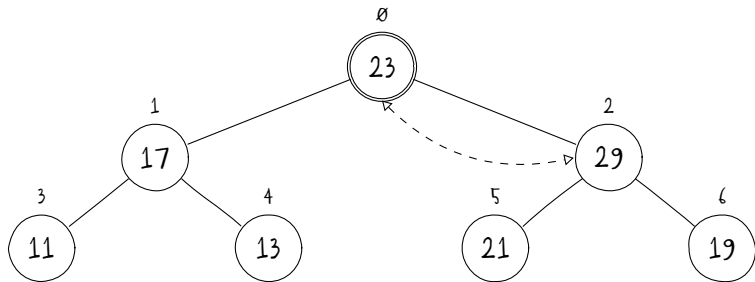
Árvore *heap*

- ▶ Construção da árvore *heap*
 - ▶ O índice é decrementado até atingir a raiz
 - ▶ *Heapify* no índice $i = \frac{(Tamanho-1)-1}{2}$



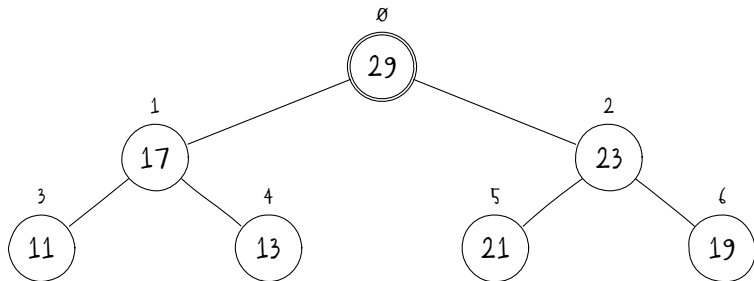
Árvore *heap*

- ▶ Construção da árvore *heap*
 - ▶ O índice é decrementado até atingir a raiz
 - ▶ *Heapify* no índice $i = \frac{(Tamanho-1)-1}{2}$



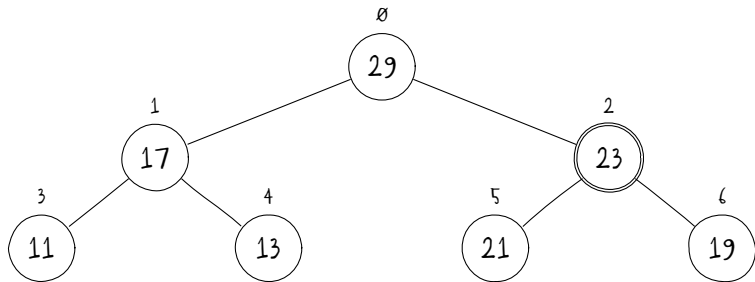
Árvore *heap*

- ▶ Construção da árvore *heap*
 - ▶ O índice é decrementado até atingir a raiz
 - ▶ *Heapify* no índice $i = \frac{(Tamanho-1)-1}{2}$



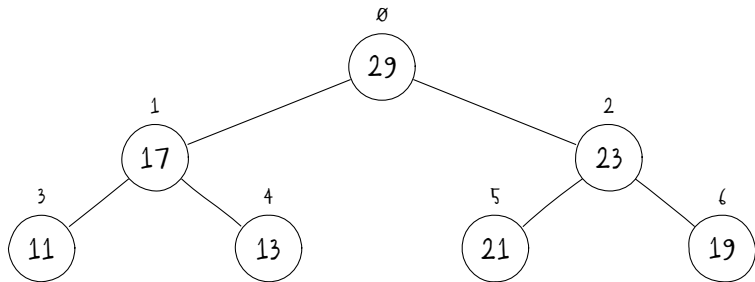
Árvore *heap*

- ▶ Construção da árvore *heap*
 - ▶ O índice é decrementado até atingir a raiz
 - ▶ *Heapify* no índice $i = \frac{(Tamanho-1)-1}{2}$



Árvore *heap*

- ▶ Construção da árvore *heap*
 - ▶ A construção do *heap* foi finalizada



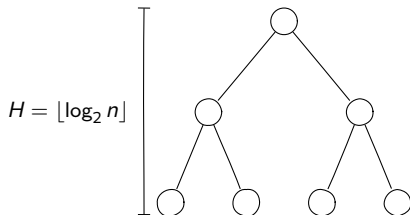
Árvore *heap*

- ▶ Análise de complexidade da construção
 - ▶ São feitas $\frac{n}{2}$ iterações do *heapify*: $\Omega(1)$ e $O(\log_2 n)$
 - ▶ Espaço: $\Omega(1)$ e $O(\log_2 n)$
 - ▶ Tempo: $\Omega(n)$ e $O(\frac{n}{2} \times h) = O(n \log_2 n)$

Árvore heap

- ▶ Análise de complexidade da construção

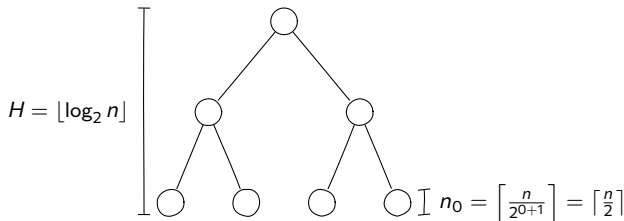
- ▶ O número de nós na profundidade h é $\left\lceil \frac{2^{\lfloor \log_2 n \rfloor}}{2^{h+1}} \right\rceil = \left\lceil \frac{n}{2^{h+1}} \right\rceil$



Árvore heap

- ▶ Análise de complexidade da construção

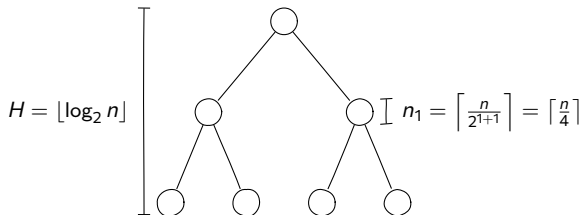
- ▶ O número de nós na profundidade h é $\left\lceil \frac{2^{\lfloor \log_2 n \rfloor}}{2^{h+1}} \right\rceil = \left\lceil \frac{n}{2^{h+1}} \right\rceil$



Árvore heap

- ▶ Análise de complexidade da construção

- ▶ O número de nós na profundidade h é $\left\lceil \frac{2^{\lfloor \log_2 n \rfloor}}{2^{h+1}} \right\rceil = \left\lceil \frac{n}{2^{h+1}} \right\rceil$



Árvore *heap*

- ▶ Análise de complexidade da construção
 - ▶ O tempo de execução do *heapify* é $O(h)$

$$\text{construir_heap}(n) = O\left(\sum_{h=0}^{\lfloor \log_2 n \rfloor} \frac{n}{2^{h+1}}\right) \times O(h)$$

Árvore *heap*

- ▶ Análise de complexidade da construção
 - ▶ O tempo de execução do *heapify* é $O(h)$

$$\begin{aligned} \text{construir_heap}(n) &= O\left(\sum_{h=0}^{\lfloor \log_2 n \rfloor} \frac{n}{2^{h+1}}\right) \times O(h) \\ &= O\left(\frac{n}{2} \times \sum_{h=0}^{\lfloor \log_2 n \rfloor} \frac{h}{2^h}\right) \end{aligned}$$

Árvore *heap*

- ▶ Análise de complexidade da construção
 - ▶ O tempo de execução do *heapify* é $O(h)$

$$\begin{aligned} \text{construir_heap}(n) &= O\left(\sum_{h=0}^{\lfloor \log_2 n \rfloor} \frac{n}{2^{h+1}}\right) \times O(h) \\ &= O\left(\frac{n}{2} \times \sum_{h=0}^{\lfloor \log_2 n \rfloor} \frac{h}{2^h}\right) \\ &= O\left(\frac{n}{2} \times \sum_{h=0}^{\infty} \frac{h}{2^h}\right)^2 \end{aligned}$$

Árvore *heap*

- ▶ Análise de complexidade da construção
 - ▶ O tempo de execução do *heapify* é $O(h)$

$$\begin{aligned} \text{construir_heap}(n) &= O\left(\sum_{h=0}^{\lfloor \log_2 n \rfloor} \frac{n}{2^{h+1}}\right) \times O(h) \\ &= O\left(\frac{n}{2} \times \sum_{h=0}^{\lfloor \log_2 n \rfloor} \frac{h}{2^h}\right) \\ &= O\left(\frac{n}{2} \times \sum_{h=0}^{\infty} \frac{h}{2^h}\right) \\ &= O(n) \end{aligned}$$

Árvore *heap*

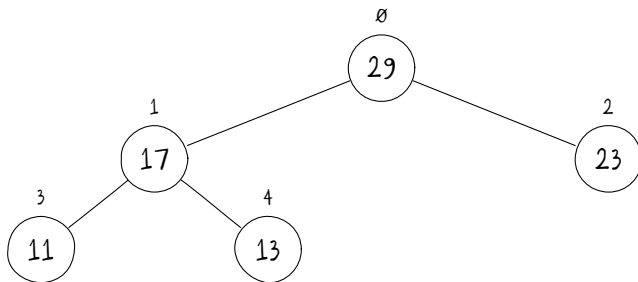
- ▶ Análise de complexidade da construção
 - ▶ Espaço: $\Omega(1)$ e $O(\log_2 n)$
 - ▶ Tempo: $\Theta(n)$

Árvore *heap*

- ▶ Operações básicas
 - ▶ Inserção
 - ▶ Remoção

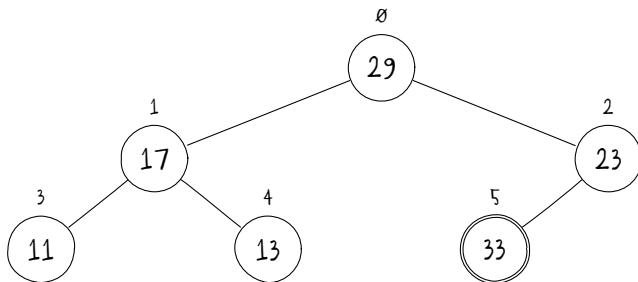
Árvore *heap*

- ▶ Operação de inserção
 - ▶ Parâmetro: 33
 - ▶ A inserção sempre é feita na última posição, aplicando o procedimento *heapify* no pai



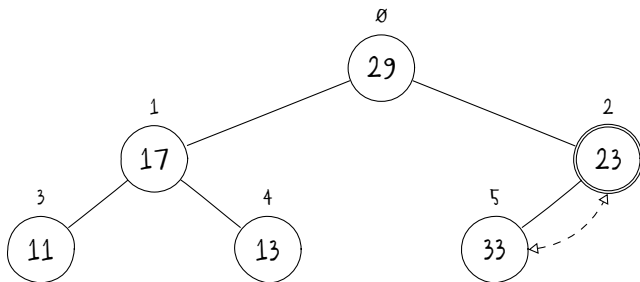
Árvore *heap*

- ▶ Operação de inserção
 - ▶ Parâmetro: 33
 - ▶ A inserção sempre é feita na última posição, aplicando o procedimento *heapify* no pai



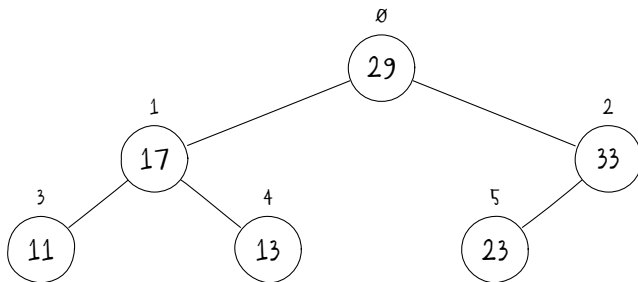
Árvore *heap*

- ▶ Operação de inserção
 - ▶ Parâmetro: 33
 - ▶ A inserção sempre é feita na última posição, aplicando o procedimento *heapify* no pai



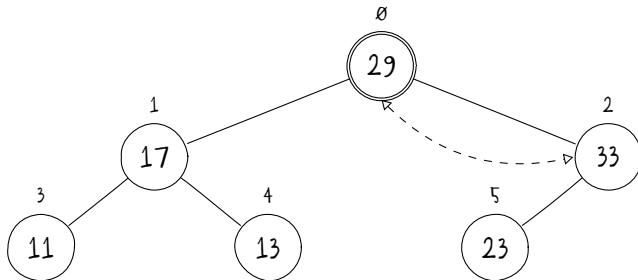
Árvore *heap*

- ▶ Operação de inserção
 - ▶ Parâmetro: 33
 - ▶ A inserção sempre é feita na última posição, aplicando o procedimento *heapify* no pai



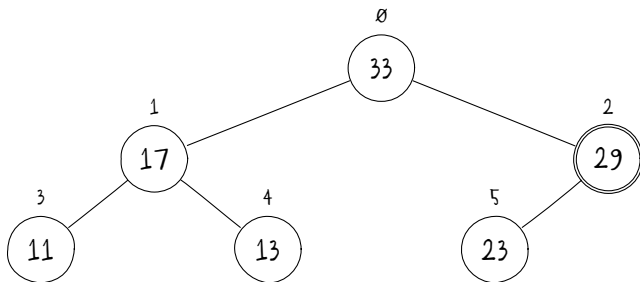
Árvore *heap*

- ▶ Operação de inserção
 - ▶ Parâmetro: 33
 - ▶ A inserção sempre é feita na última posição, aplicando o procedimento *heapify* no pai



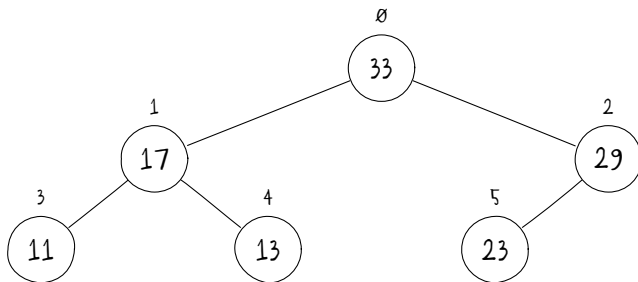
Árvore *heap*

- ▶ Operação de inserção
 - ▶ Parâmetro: 33
 - ▶ A inserção sempre é feita na última posição, aplicando o procedimento *heapify* no pai



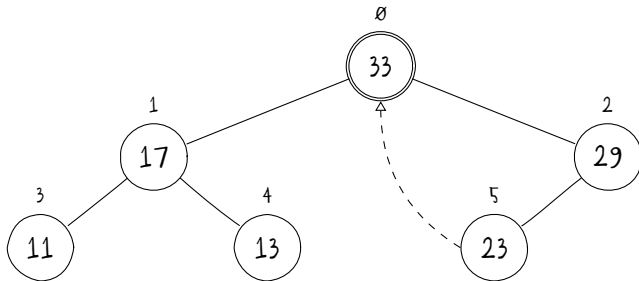
Árvore *heap*

- ▶ Operação de inserção
 - ▶ Parâmetro: 33
 - ▶ A inserção sempre é feita na última posição, aplicando o procedimento *heapify* no pai



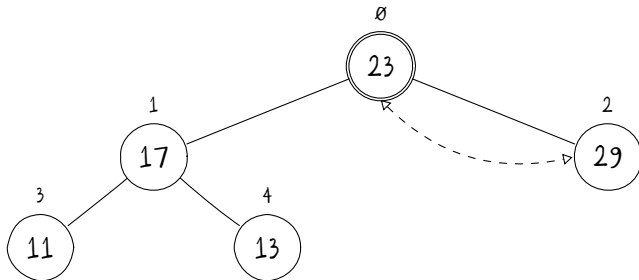
Árvore heap

- ▶ Operação de remoção
 - ▶ Sempre é removido o nó raiz da árvore (mínimo ou máximo), sendo substituído pelo último elemento



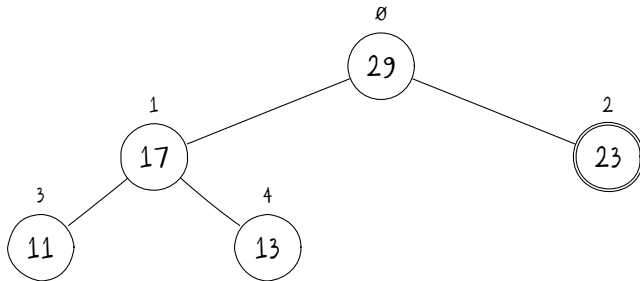
Árvore heap

- ▶ Operação de remoção
 - ▶ Sempre é removido o nó raiz da árvore (mínimo ou máximo), sendo substituído pelo último elemento



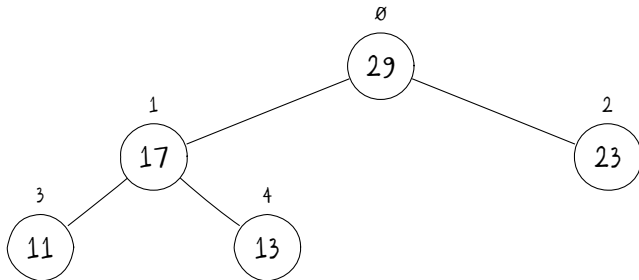
Árvore heap

- ▶ Operação de remoção
 - ▶ Sempre é removido o nó raiz da árvore (mínimo ou máximo), sendo substituído pelo último elemento



Árvore heap

- ▶ Operação de remoção
 - ▶ Sempre é removido o nó raiz da árvore (mínimo ou máximo), sendo substituído pelo último elemento



Árvore *heap*

- ▶ Análise de complexidade das operações
 - ▶ Espaço e tempo: $\Omega(1)$ e $O(\log_2 n)$

Exemplo

- ▶ Construa uma árvore *heap* mínimo e máximo
 - ▶ Considere os números do vetor abaixo
 - ▶ Ilustre a construção passo a passo

13	2	34	11	7	43	9
0	1	2	3	4	5	6

Exercício

- ▶ A empresa de tecnologia Poxim Tech e a empresa de capitalização Banana Cap estão desenvolvendo um sistema para apuração eficiente dos resultados dos concursos de loteria realizados
 - ▶ Os apostadores podem escolher 15 números dentre os valores de 1 até 50, sendo igualmente premiados por faixa as apostas com maior e menor número de acertos, impedindo a acumulação do prêmio
 - ▶ Em cada concurso são sorteados 10 números distintos que permitem aos apostadores obterem entre 0 e 10 acertos para cada aposta
 - ▶ O código da aposta é representado por um número hexadecimal único de 128 bits

Exercício

- ▶ Formato de arquivo de entrada
 - ▶ [*Prêmio em reais*]
 - ▶ [*#Quantidade de apostas(n)*]
 - ▶ [*Sorteado₁*] ... [*Sorteado₁₀*]
 - ▶ [*Código₁*] [*Número₁*] ... [*Número₁₅*]
 - ▶ ⋮
 - ▶ [*Código_n*] [*Número_n*] ... [*Número_{n15}*]

Exercício

- ▶ Formato de arquivo de entrada

```

1 3000
2 5
3 1_2_3_5_8_13_25_33_42_48
4 1234567890ABCDEF1234567890ABCDEF _1_2_3_9_11_17_19_
    20_21_30_34_38_39_40_44
5 1223334444555556666667777777ABCD _2_3_5_9_13_14_15_
    17_18_20_33_35_40_41_42
6 AAAAAABBBBBBCCCCCDDDDDEEEEEFFFFFFF _1_2_5_8_9_11_15_16_
    19_21_27_33_35_42_49
7 0A1B2C3D4E5F6A7B8C9DAEBFCEDFE0F1 _3_4_5_7_11_16_18_
    20_24_25_31_34_35_42_50
8 F0F0F0F0F0F0F0F0F0F0F0F0F0F0F0F0 _3_4_7_9_15_18_23_
    24_26_31_32_38_41_43_48

```

Exercício

- ▶ Formato de arquivo de saída
 - ▶ São exibidos as duas faixas de acerto com a quantidade de apostas premiadas, o número de acertos e o valor do prêmio individual, além da listagem dos códigos das apostas em cada faixa

```
1 [2:6:750]
2 1223334445555566666677777777ABCD
3 AAAAAABBBBBBCCCCDDDDDEEEEEFFFFFFF
4 [1:2:1500]
5 F0F0F0F0F0F0F0F0F0F0F0F0F0F0F0F0
```