Aula 19 – Heaps

Norton T. Roman & Luciano A. Digiampietri
digiampietri@usp.br
@digiampietri

2023

Heaps

Definição

- Estrutura de dados definida como uma sequência S de itens com valores $S[1], S[2], \ldots, S[n]$ tais que

$$S[i] \geq S[2i]$$
$$S[i] \geq S[2i+1]$$

para todo $i = 1, 2, \ldots, n/2$

Heaps

Definição

- Estrutura de dados definida como uma sequência S de itens com valores $S[1], S[2], \ldots, S[n]$ tais que

$$S[i] \geq S[2i]$$
$$S[i] \geq S[2i+1]$$

  para todo $i = 1, 2, \ldots, n/2$

- **Heap máximo**

Heaps

Definição

- Se, contudo, quisermos um **heap mínimo**, basta mudarmos a definição para

$$S[i] \leq S[2i]$$
$$S[i] \leq S[2i+1]$$

para todo $i = 1, 2, \ldots, n/2$

Heaps

Visualização

- Para entender o heap, considere uma uma **árvore binária completa**

Heaps

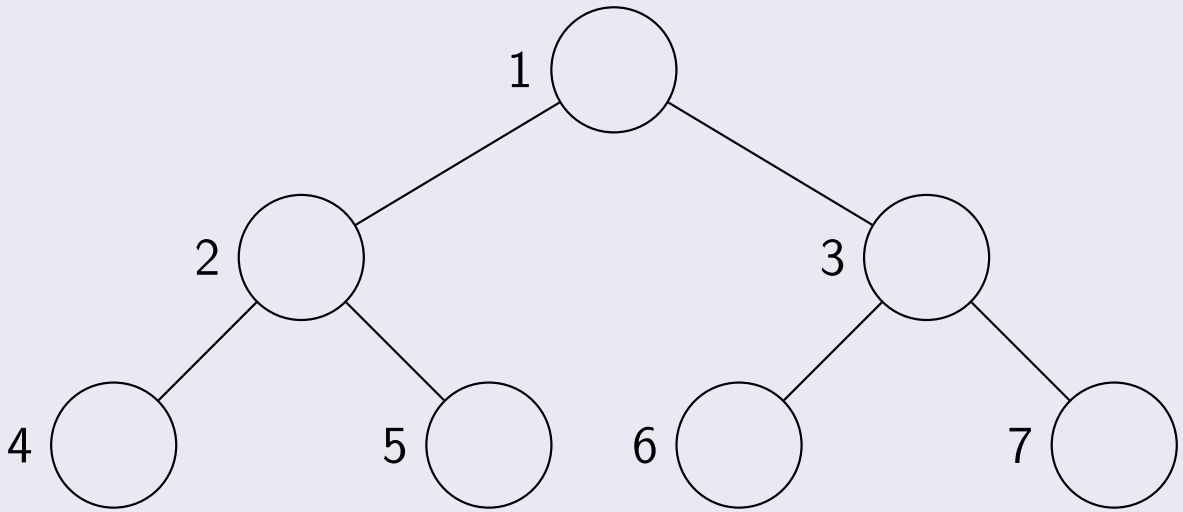
Visualização

- Para entender o heap, considere uma uma **árvore binária completa**
  - Ou seja, a árvore binária em que todas as folhas têm a mesma profundidade e todos os nós internos têm 2 filhos

Heaps

Visualização

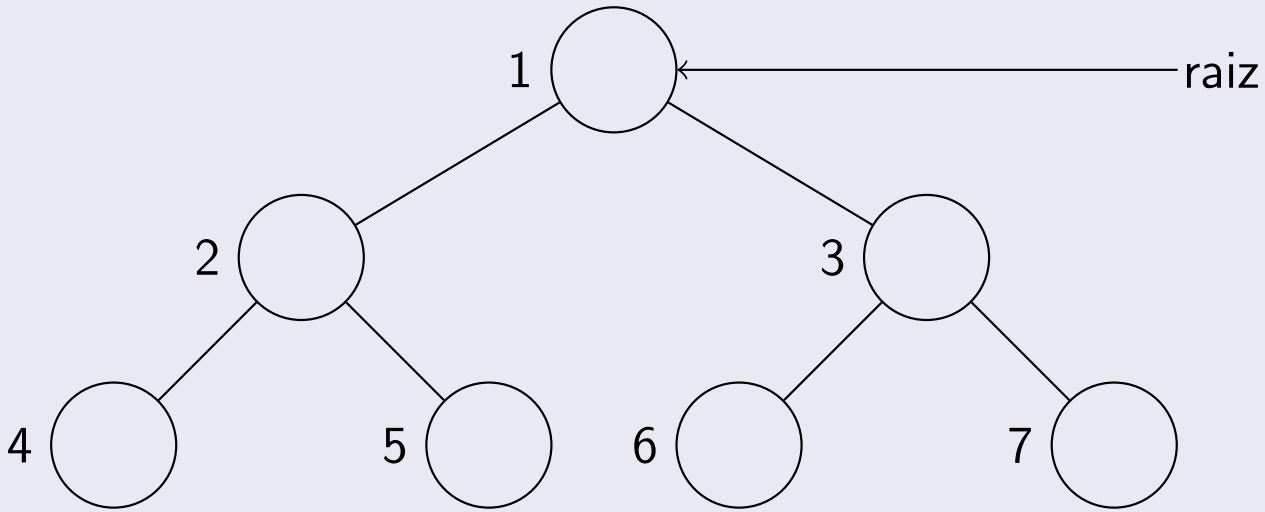
- Para entender o heap, considere uma uma **árvore binária completa**
  - Ou seja, a árvore binária em que todas as folhas têm a mesma profundidade e todos os nós internos têm 2 filhos



Heaps

Visualização

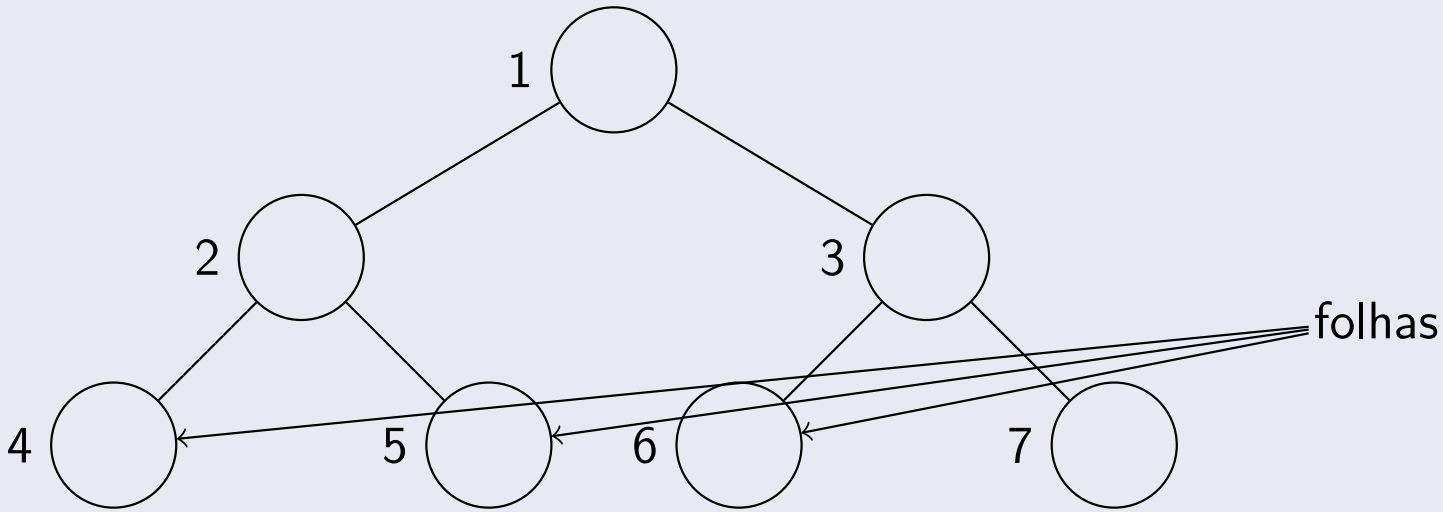
- Para entender o heap, considere uma uma **árvore binária completa**
  - Ou seja, a árvore binária em que todas as folhas têm a mesma profundidade e todos os nós internos têm 2 filhos



Heaps

Visualização

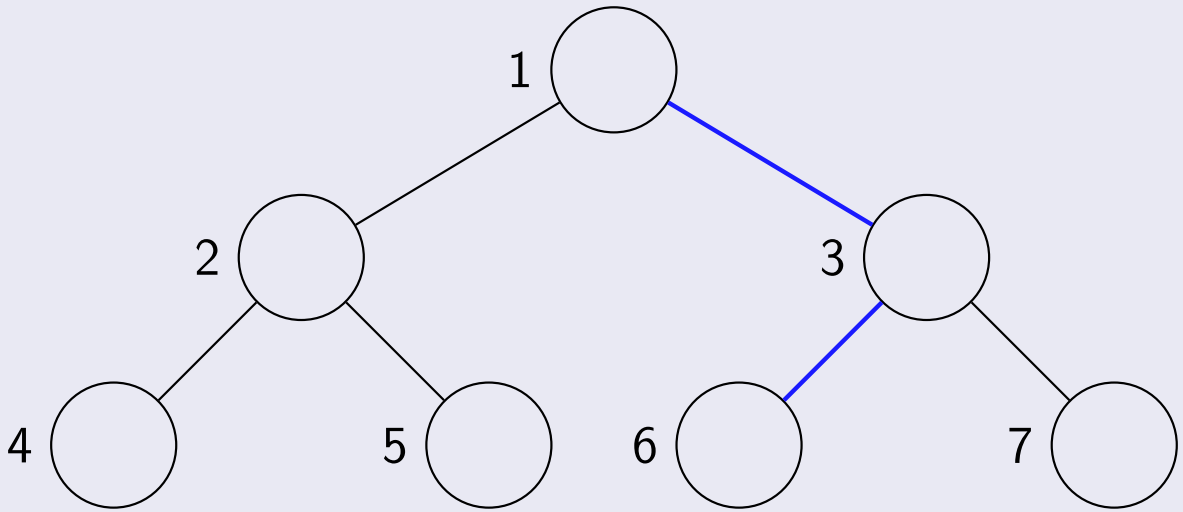
- Para entender o heap, considere uma uma **árvore binária completa**
  - Ou seja, a árvore binária em que todas as folhas têm a mesma profundidade e todos os nós internos têm 2 filhos



Heaps

Visualização

- Para entender o heap, considere uma uma **árvore binária completa**
  - Ou seja, a árvore binária em que todas as folhas têm a mesma profundidade e todos os nós internos têm 2 filhos

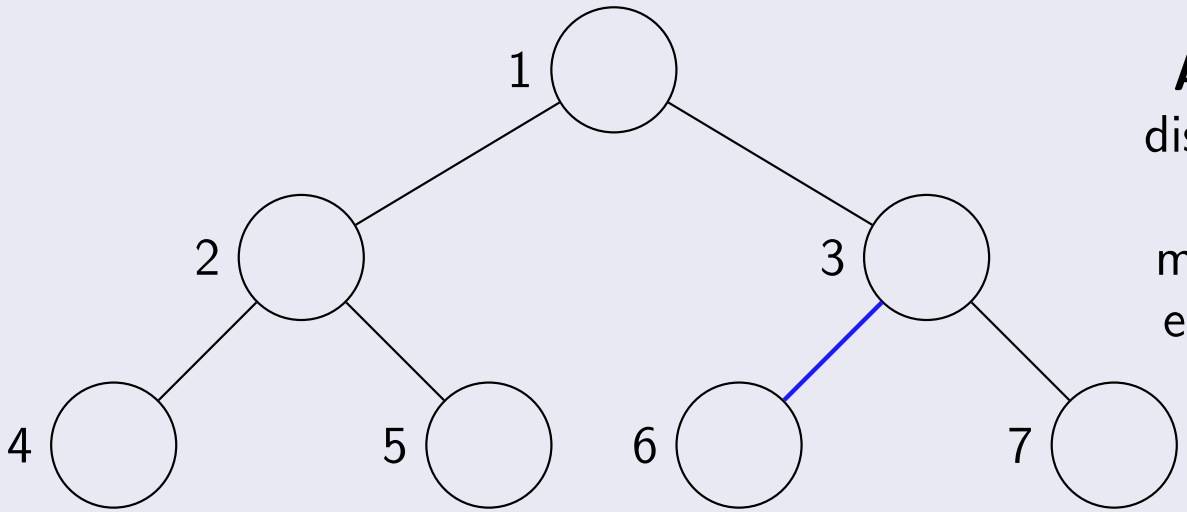


Profundidade de um nó: distância do nó à raiz, em número de arestas

Heaps

Visualização

- Para entender o heap, considere uma uma **árvore binária completa**
  - Ou seja, a árvore binária em que todas as folhas têm a mesma profundidade e todos os nós internos têm 2 filhos

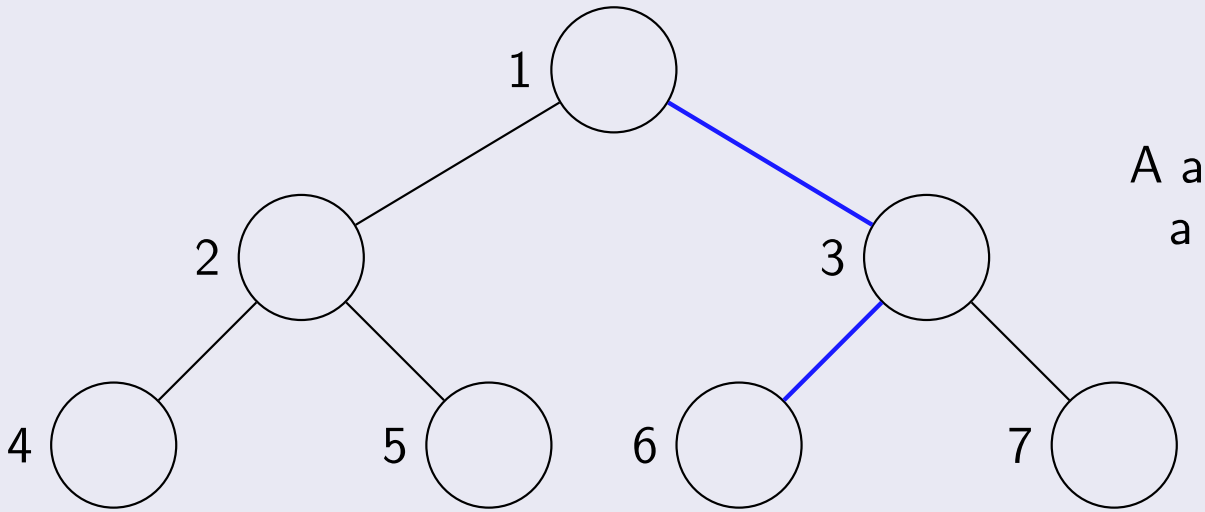


Altura de um nó: distância descendente (em arestas) do maior caminho entre esse nó e uma folha

Heaps

Visualização

- Para entender o heap, considere uma uma **árvore binária completa**
  - Ou seja, a árvore binária em que todas as folhas têm a mesma profundidade e todos os nós internos têm 2 filhos



A altura da árvore será a altura de sua raiz

Heaps

Visualização

- Relaxemos agora para uma árvore binária quase completa

Heaps

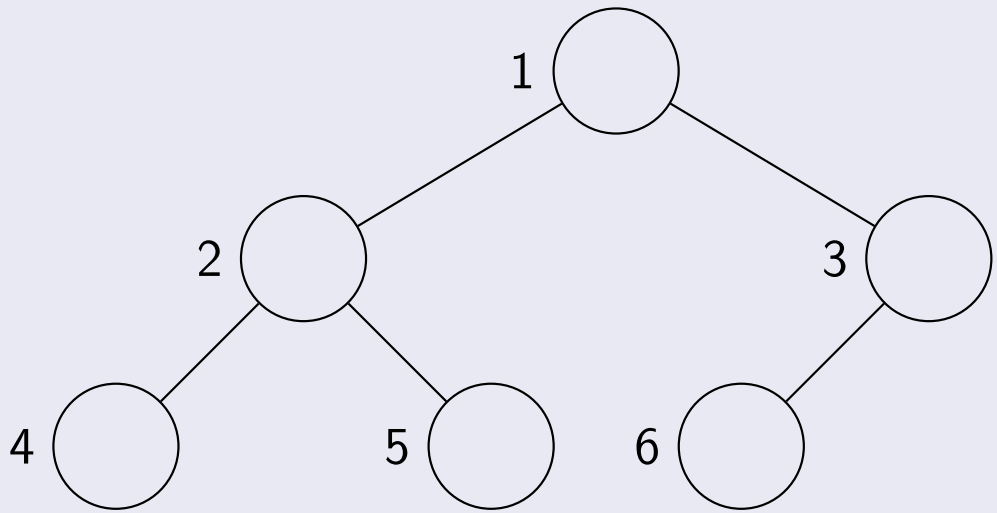
Visualização

- Relaxemos agora para uma árvore binária quase completa
  - Ou seja, uma árvore binária completa, com possível exceção do nível mais baixo, que é preenchido da esquerda até um certo ponto

Heaps

Visualização

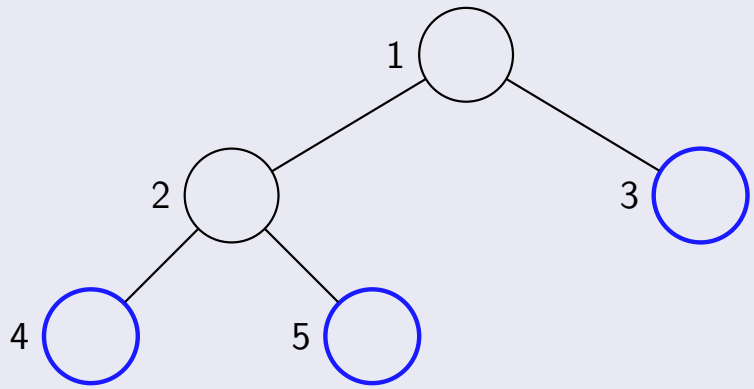
- Relaxemos agora para uma árvore binária quase completa
  - Ou seja, uma árvore binária completa, com possível exceção do nível mais baixo, que é preenchido da esquerda até um certo ponto



Heaps

Visualização

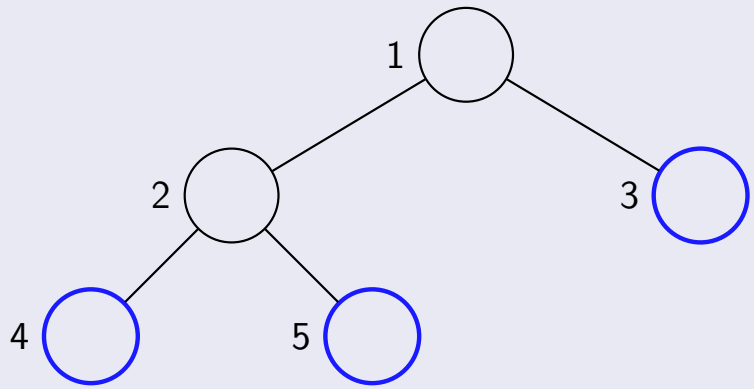
- Note que, se seguirmos essa ordem de criação da árvore (1 a n) e o último nível da árvore não estiver cheio, então as folhas aparecerão em 2 níveis adjacentes



Heaps

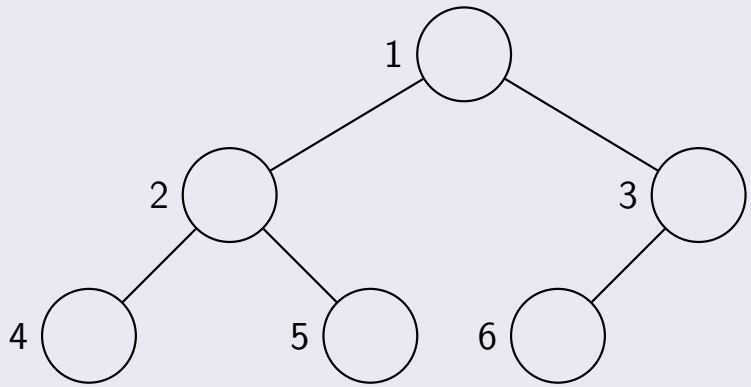
Visualização

- Note que, se seguirmos essa ordem de criação da árvore (1 a n) e o último nível da árvore não estiver cheio, então as folhas aparecerão em 2 níveis adjacentes
  - Com as mais baixas à esquerda



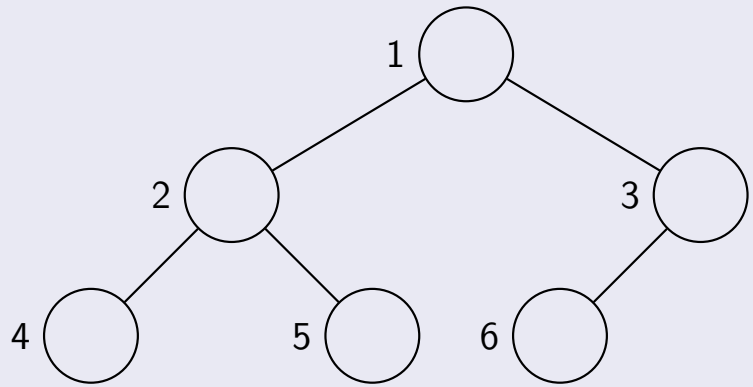
Visualização

- Também note que:



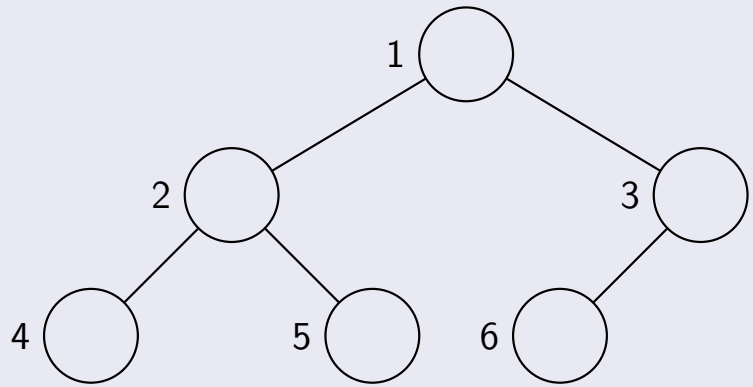
Visualização

- Também note que:
  - O nó $\lfloor k/2 \rfloor$ é pai do nó k, $1 < k \leq n$



Visualização

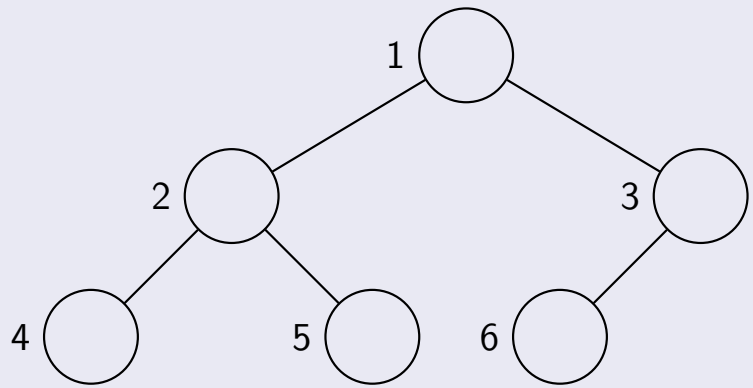
- Também note que:
  - O nó $\lfloor k/2 \rfloor$ é pai do nó k, $1 < k \leq n$
  - Os nós $2k$ e $2k + 1$ são os filhos à esquerda e à direita do nó k, $1 \leq k \leq \lfloor n/2 \rfloor$



Heaps

Visualização

- Também note que:
  - O nó $\lfloor k/2 \rfloor$ é pai do nó k, $1 < k \leq n$
  - Os nós $2k$ e $2k + 1$ são os filhos à esquerda e à direita do nó k, $1 \leq k \leq \lfloor n/2 \rfloor$

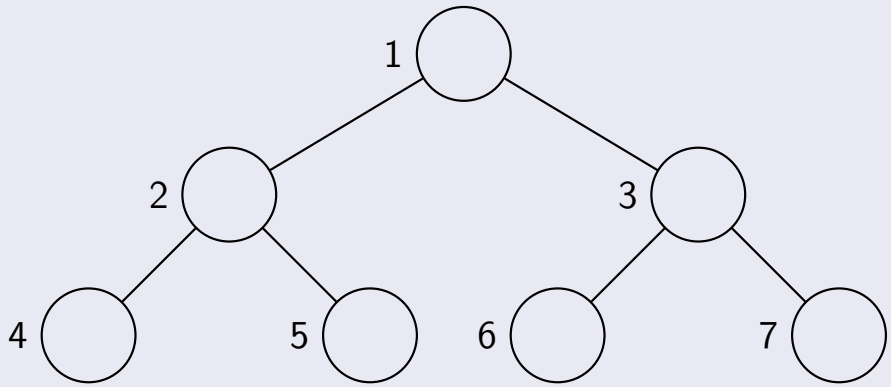


- E que relação isso tem com os valores permitidos em um heap?

Visualização: Heap Máximo

$$S[i] \geq S[2i]$$
$$S[i] \geq S[2i+1]$$

$$i = 1, 2, \ldots, n/2$$

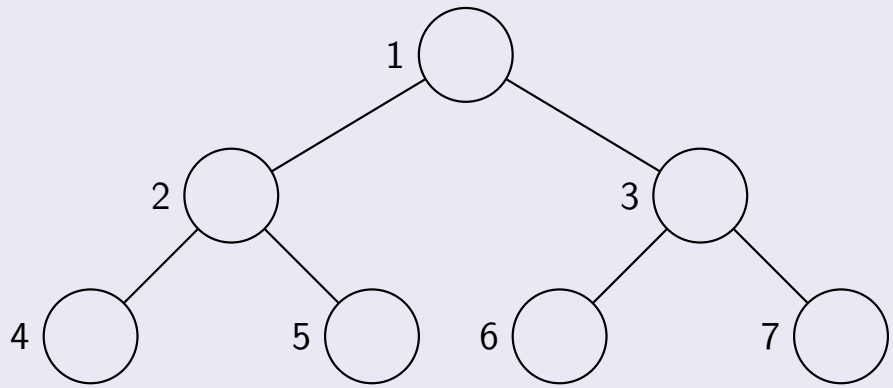


Heaps

Visualização: Heap Máximo

$$S[i] \geq S[2i]$$
$$S[i] \geq S[2i+1]$$

$$i = 1, 2, \ldots, n/2$$



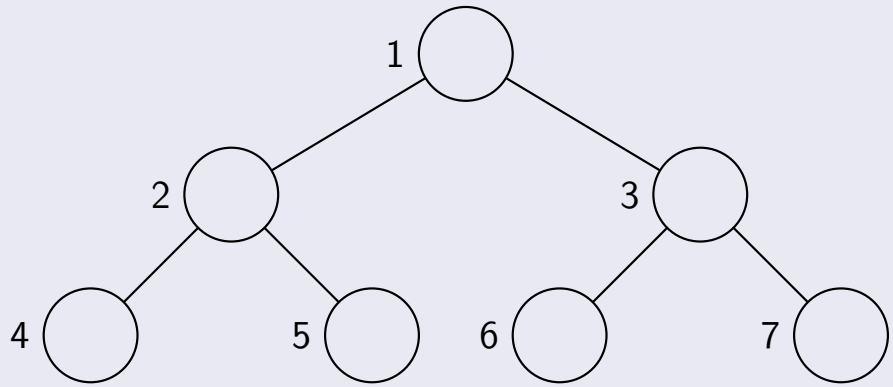
- Os nós com valores $2i$ e $2i+1$ serão os filhos à esquerda e à direita do nó com valor i

Heaps

Visualização: Heap Máximo

$$S[i] \geq S[2i]$$
$$S[i] \geq S[2i+1]$$

$$i = 1, 2, \ldots, n/2$$



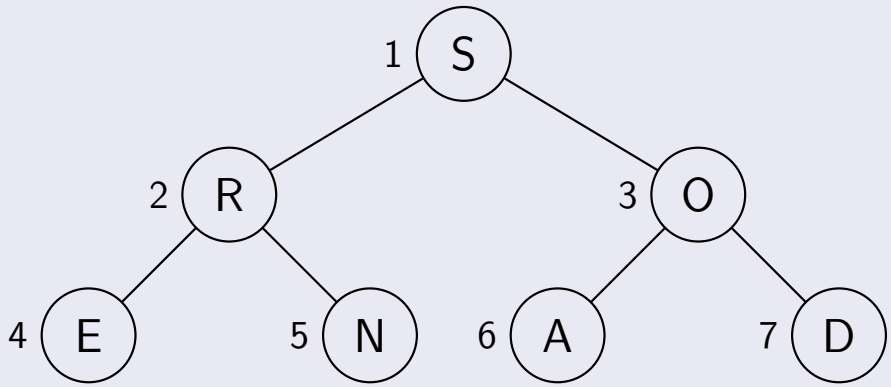
- Os nós com valores $2i$ e $2i+1$ serão os filhos à esquerda e à direita do nó com valor i
- Montando a árvore dessa maneira, em que o valor de cada nó é maior ou igual ao valor dos filhos, ela satisfará a condição do **heap máximo**

Heaps

Visualização: Heap Máximo

$$S[i] \geq S[2i]$$
$$S[i] \geq S[2i+1]$$

$$i = 1, 2, \ldots, n/2$$



- Os nós com valores $2i$ e $2i+1$ serão os filhos à esquerda e à direita do nó com valor i
- Montando a árvore dessa maneira, em que o valor de cada nó é maior ou igual ao valor dos filhos, ela satisfará a condição do **heap máximo**

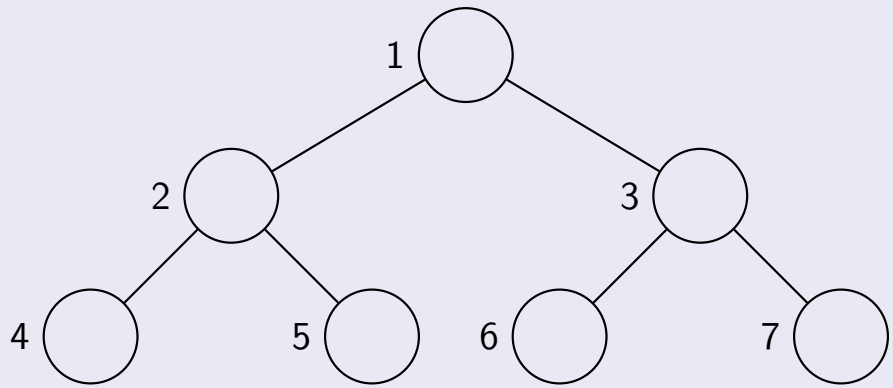
Heaps

Visualização: Heap Mínimo

- Para o **heap mínimo**, basta fazermos com que o valor de cada nó seja menor ou igual ao de seus filhos

$$S[i] \leq S[2i]$$
$$S[i] \leq S[2i+1]$$

$$i = 1, 2, \ldots, n/2$$

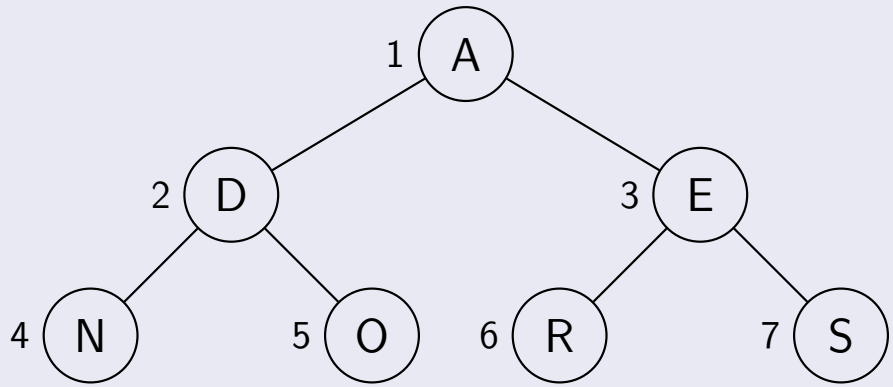


Visualização: Heap Mínimo

- Para o **heap mínimo**, basta fazermos com que o valor de cada nó seja menor ou igual ao de seus filhos

$$S[i] \leq S[2i]$$
$$S[i] \leq S[2i+1]$$

$$i = 1, 2, \ldots, n/2$$



Heaps Máximo e Mínimo

Propriedade dos Heaps

- A representação usando uma árvore nos permite redefinir a propriedade dos heaps:

Heaps Máximo e Mínimo

Propriedade dos Heaps

- A representação usando uma árvore nos permite redefinir a propriedade dos heaps:
- **Heap Máximo**:
  - Aquele em que, para todo nó i diferente da raiz, $S[pai(i)] \geq S[i]$

Heaps Máximo e Mínimo

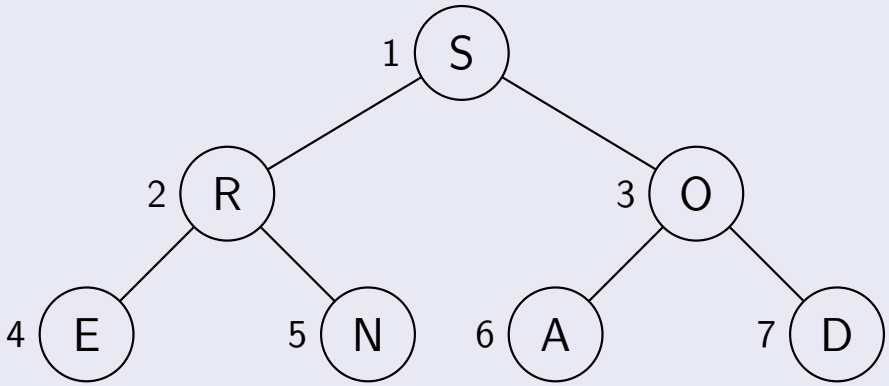
Propriedade dos Heaps

- A representação usando uma árvore nos permite redefinir a propriedade dos heaps:
- **Heap Máximo**:
  - Aquele em que, para todo nó i diferente da raiz, $S[pai(i)] \geq S[i]$
- **Heap Mínimo**:
  - Aquele em que, para todo nó i diferente da raiz, $S[pai(i)] \leq S[i]$

Heaps

Representação

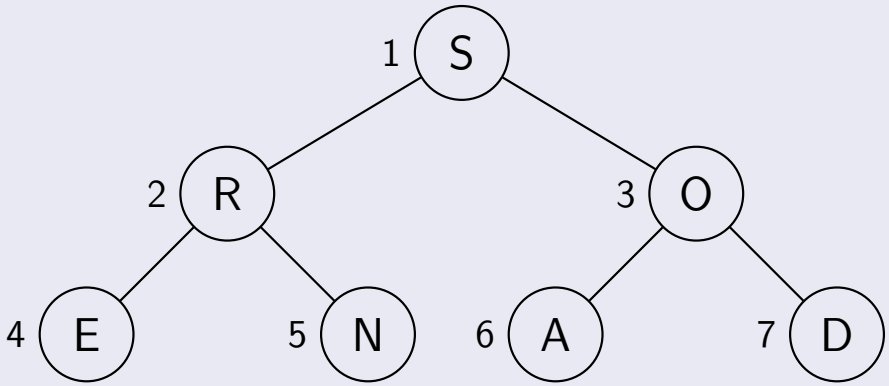
- E como representamos essa árvore computacionalmente?



Heaps

Representação

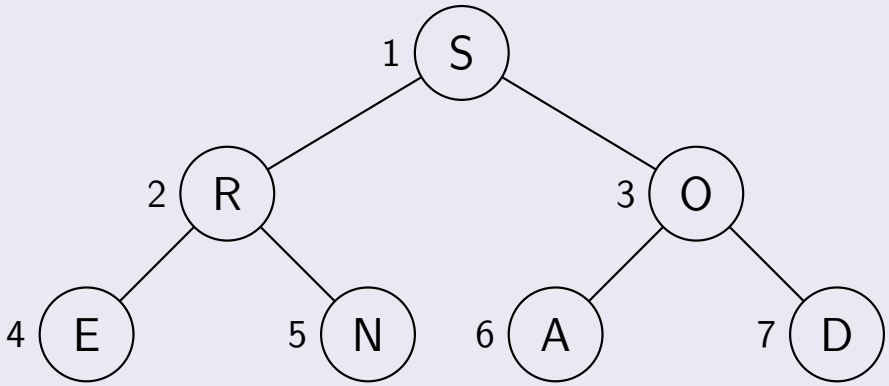
- E como representamos essa árvore computacionalmente?
- Com um arranjo



Heaps

Representação

- E como representamos essa árvore computacionalmente?
- Com um arranjo

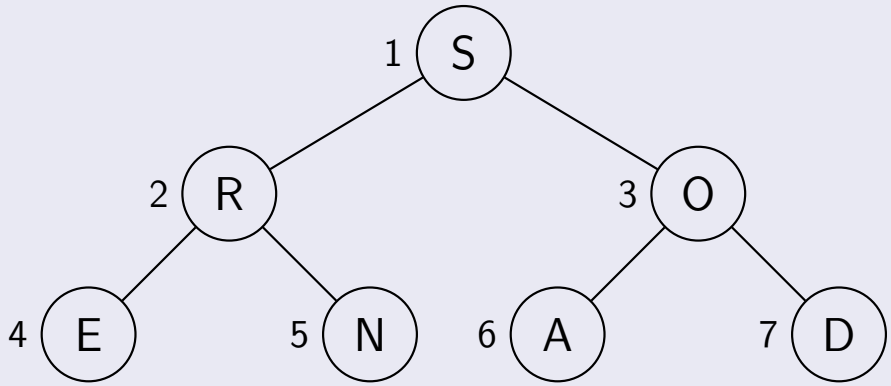


S	R	O	E	N	A	D
1	2	3	4	5	6	7

Heaps

Representação

- E como representamos essa árvore computacionalmente?
- Com um arranjo
  - Os filhos do nó i ficarão nas posições $2i$ e $2i + 1$, caso existam

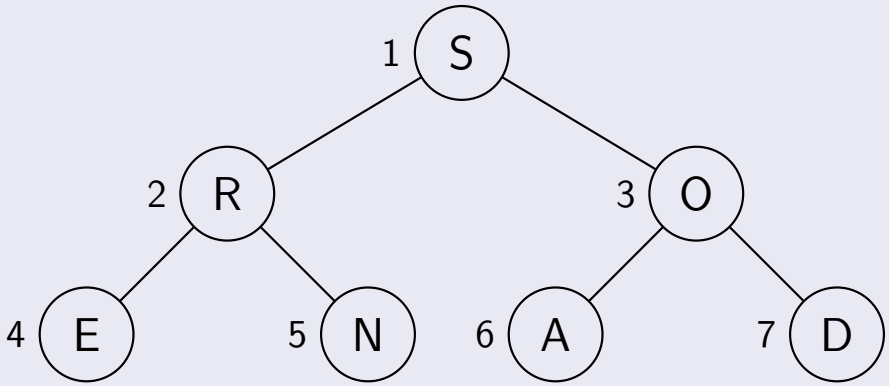


S	R	O	E	N	A	D
1	2	3	4	5	6	7

Heaps

Representação

- E como representamos essa árvore computacionalmente?
- Com um arranjo
  - Os filhos do nó i ficarão nas posições $2i$ e $2i + 1$, caso existam
  - O pai de um nó i estará na posição $i/2$



S	R	O	E	N	A	D
1	2	3	4	5	6	7

Heaps

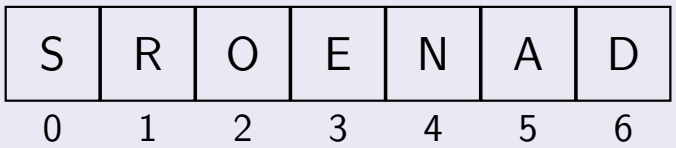
Representação

- Mas, em muitas linguagens o arranjo começa em 0

Heaps

Representação

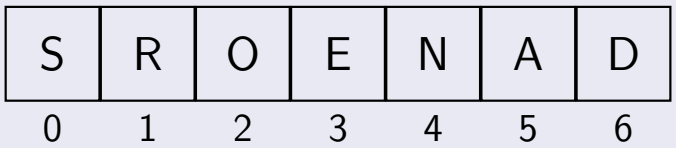
- Mas, em muitas linguagens o arranjo começa em 0
- Sem problemas, basta adaptar



Heaps

Representação

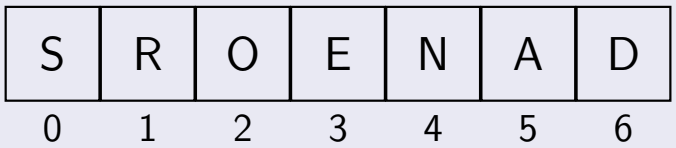
- Mas, em muitas linguagens o arranjo começa em 0
- Sem problemas, basta adaptar
  - Os filhos do nó i ficarão nas posições $2i + 1$ e $2i + 2$, caso existam



Heaps

Representação

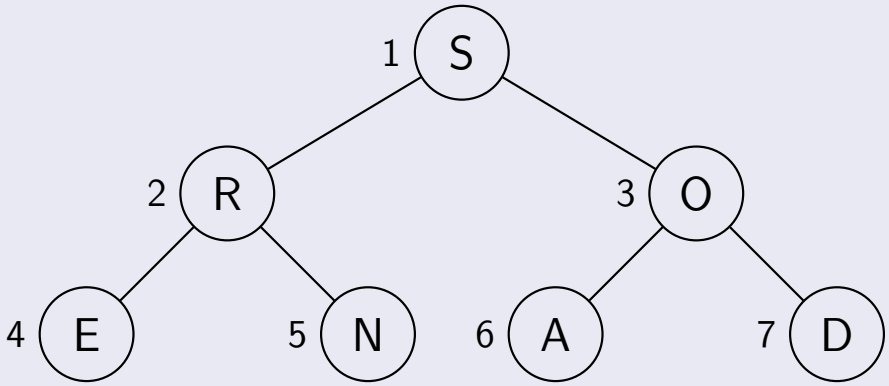
- Mas, em muitas linguagens o arranjo começa em 0
- Sem problemas, basta adaptar
  - Os filhos do nó i ficarão nas posições $2i + 1$ e $2i + 2$, caso existam
  - O pai de um nó i estará na posição $(i - 1)/2$



Heaps

Representação

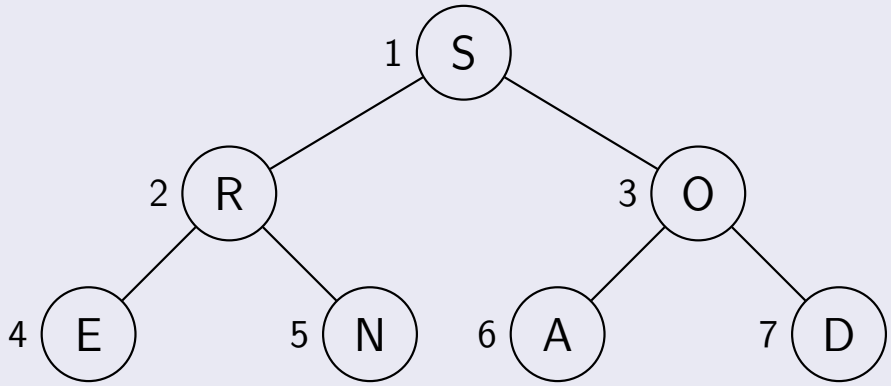
- Um heap é então uma árvore binária quase completa na qual cada nó satisfaz a condição do heap



Heaps

Representação

- Um heap é então uma árvore binária quase completa na qual cada nó satisfaz a condição do heap
- No caso do heap máximo, a maior chave estará sempre na primeira posição



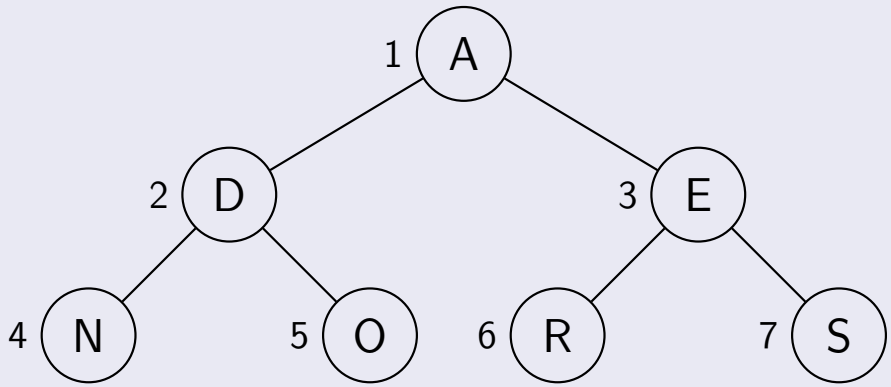
S	R	O	E	N	A	D
1	2	3	4	5	6	7

(heap máximo)

Heaps

Representação

- Já no heap mínimo, a menor chave é que estará sempre na primeira posição



A	D	E	N	O	R	S
1	2	3	4	5	6	7

(heap mínimo)

Heaps

Operações no Heap

- Algoritmos que operam sobre o heap o fazem ao longo de algum caminho na árvore

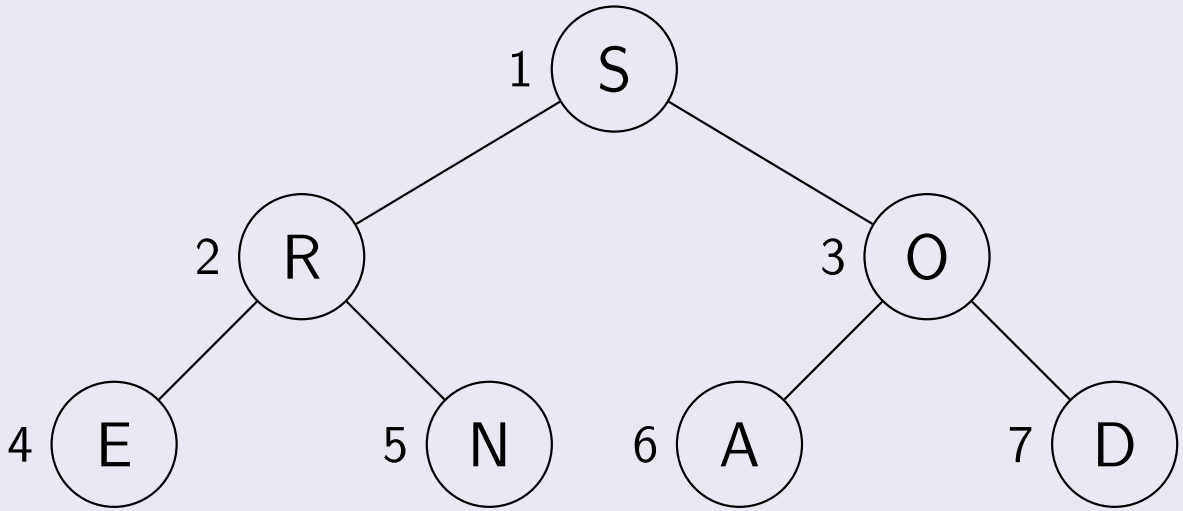
Heaps

Operações no Heap

- Algoritmos que operam sobre o heap o fazem ao longo de algum caminho na árvore
  - A partir da raiz até uma folha

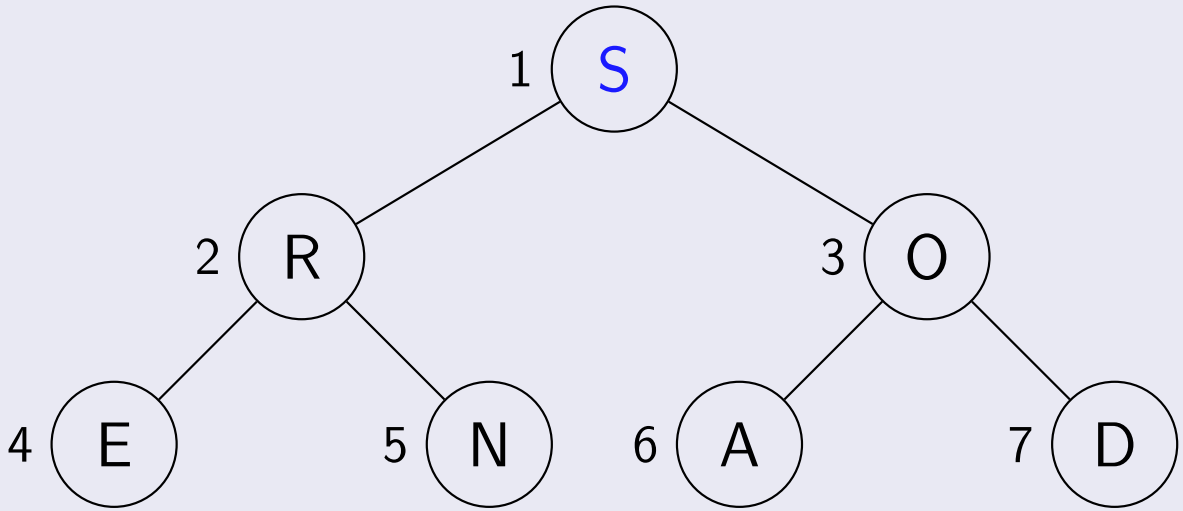
Operações no Heap

- Algoritmos que operam sobre o heap o fazem ao longo de algum caminho na árvore
  - A partir da raiz até uma folha



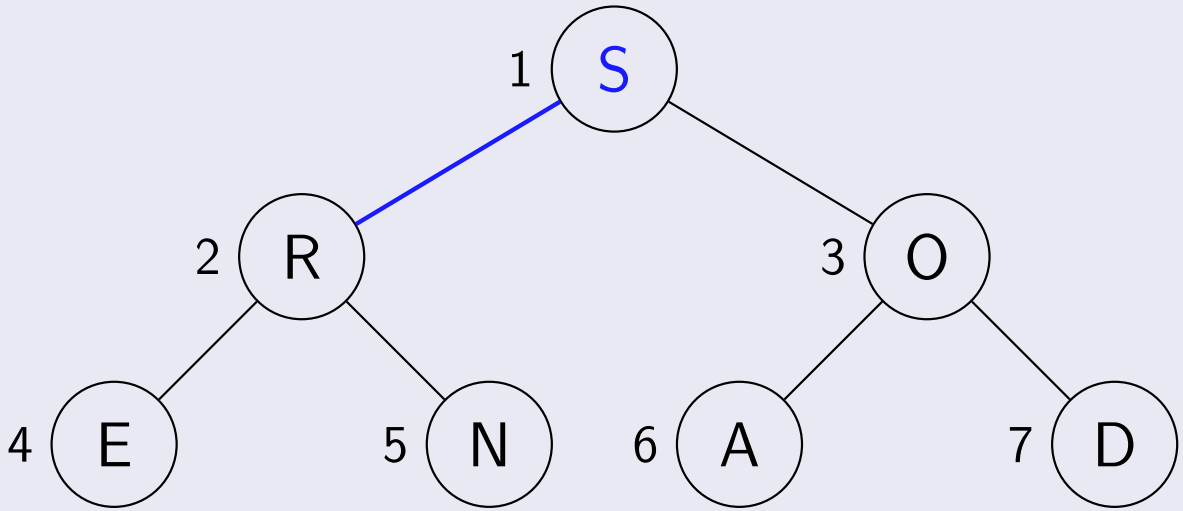
Operações no Heap

- Algoritmos que operam sobre o heap o fazem ao longo de algum caminho na árvore
  - A partir da raiz até uma folha



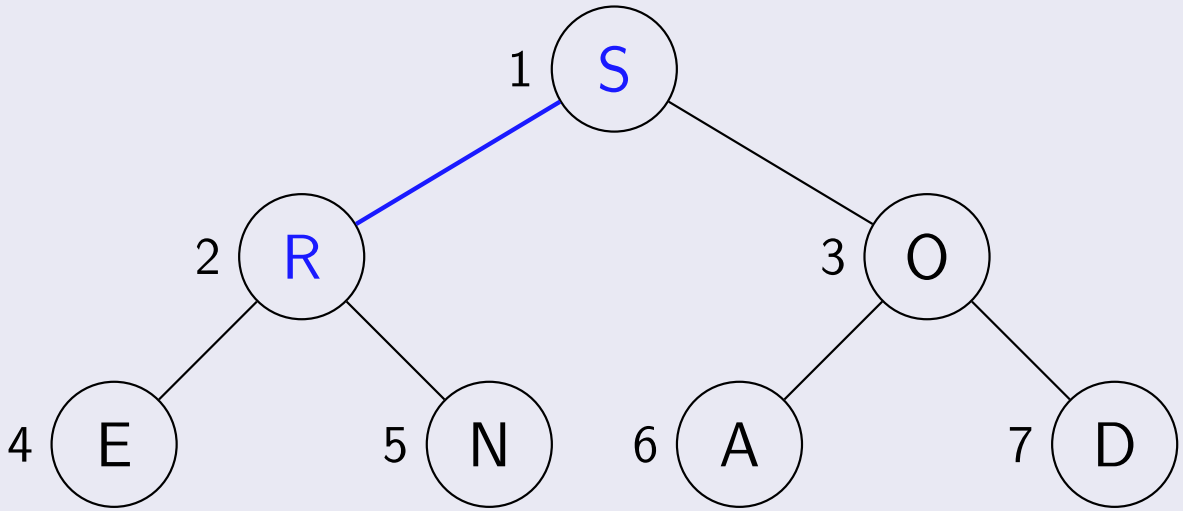
Operações no Heap

- Algoritmos que operam sobre o heap o fazem ao longo de algum caminho na árvore
  - A partir da raiz até uma folha



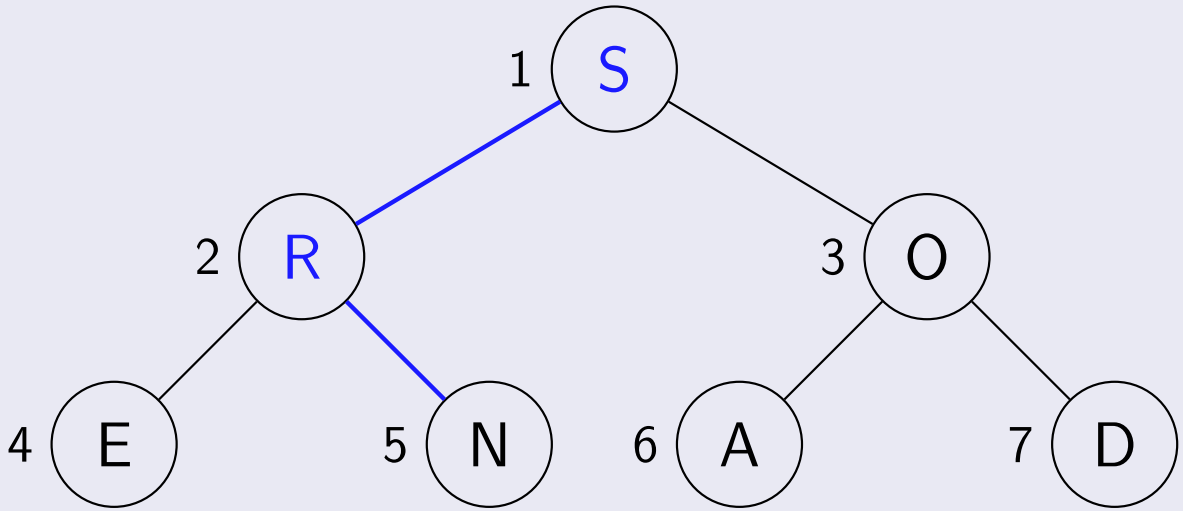
Operações no Heap

- Algoritmos que operam sobre o heap o fazem ao longo de algum caminho na árvore
  - A partir da raiz até uma folha



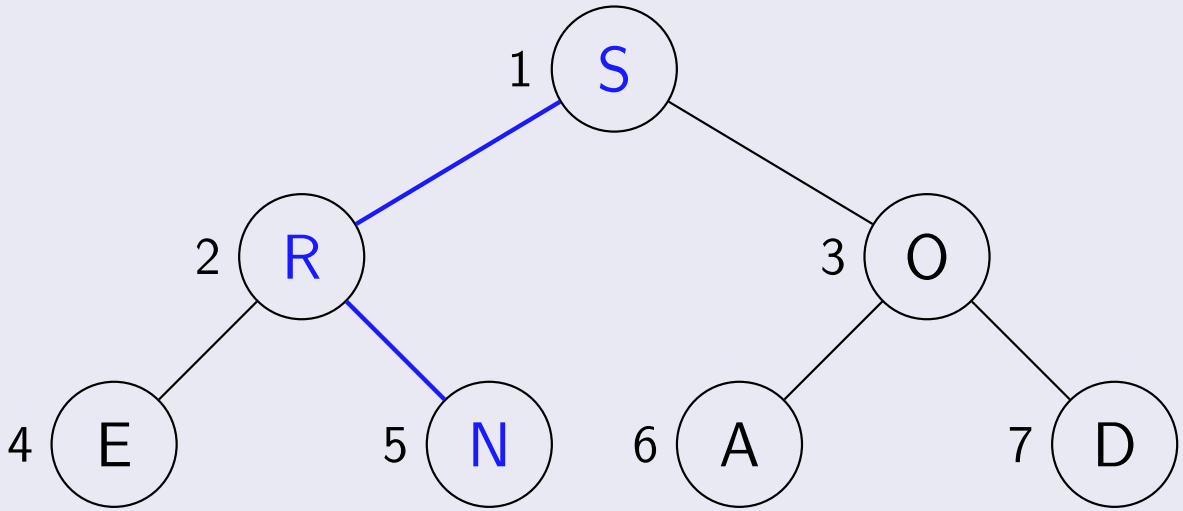
Operações no Heap

- Algoritmos que operam sobre o heap o fazem ao longo de algum caminho na árvore
  - A partir da raiz até uma folha



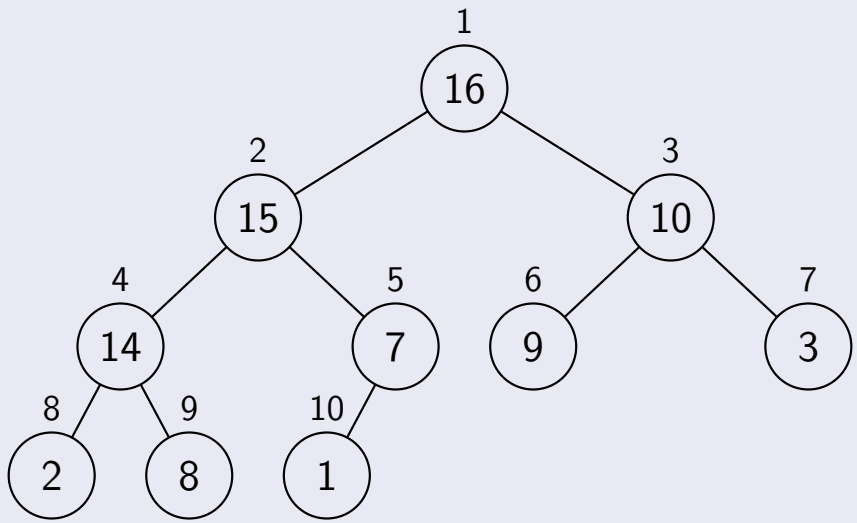
Operações no Heap

- Algoritmos que operam sobre o heap o fazem ao longo de algum caminho na árvore
  - A partir da raiz até uma folha



Mantendo a Propriedade do Heap

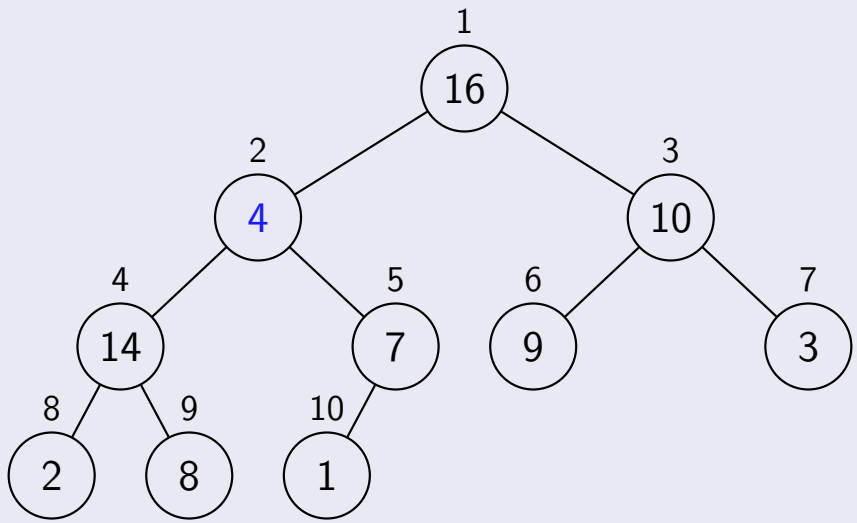
- Imagine que temos o seguinte heap máximo



Heaps

Mantendo a Propriedade do Heap

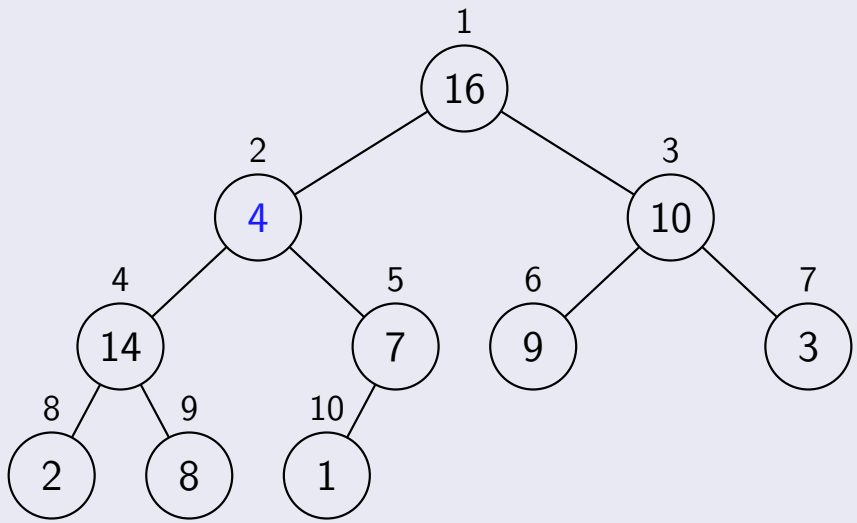
- Imagine que temos o seguinte heap máximo
- Imagine agora que houve uma alteração que pode violar a propriedade do heap



Heaps

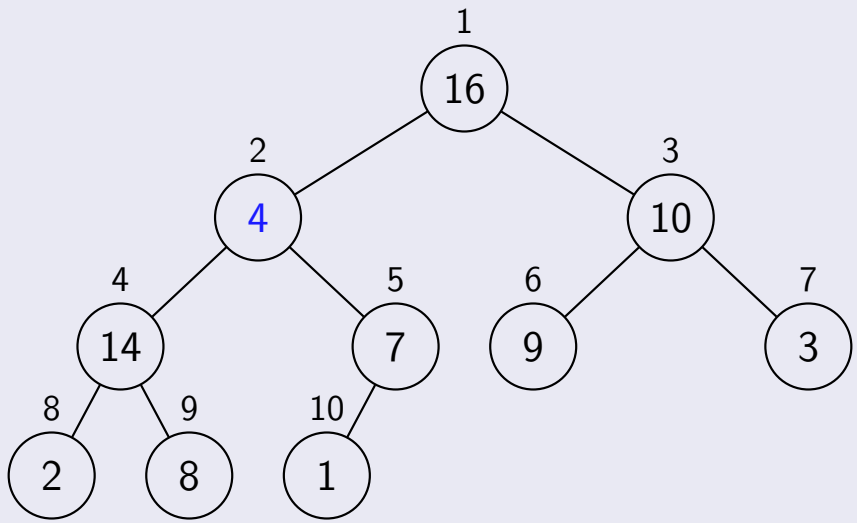
Mantendo a Propriedade do Heap

- Imagine que temos o seguinte heap máximo
- Imagine agora que houve uma alteração que pode violar a propriedade do heap
  - Pode deixar valor do pai menor que o dos filhos



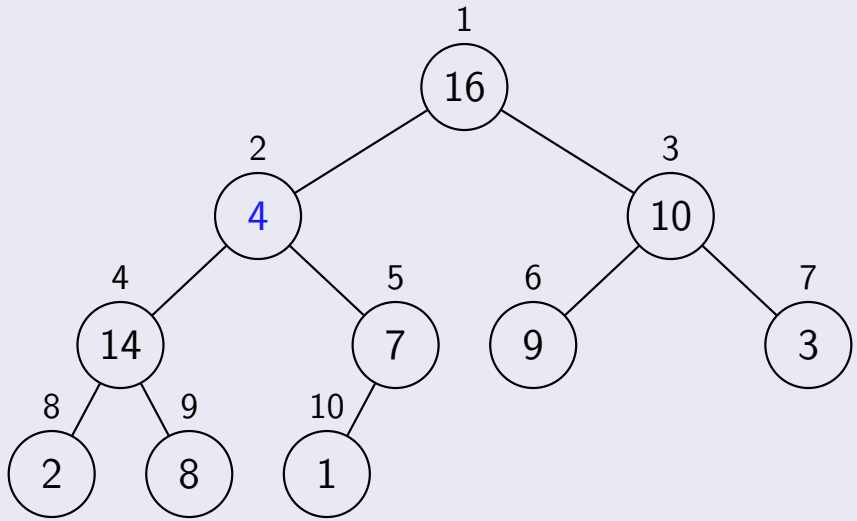
Mantendo a Propriedade do Heap

- Imagine que temos o seguinte heap máximo
- Imagine agora que houve uma alteração que pode violar a propriedade do heap
  - Pode deixar valor do pai menor que o dos filhos
- Como restaurar essa propriedade?



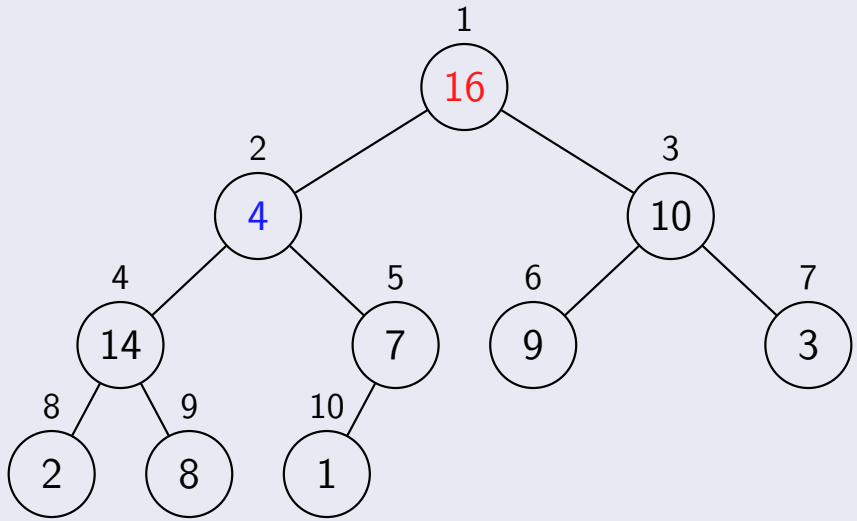
Mantendo a Propriedade do Heap

- Antes de mais nada, verificamos no heap onde houve essa violação



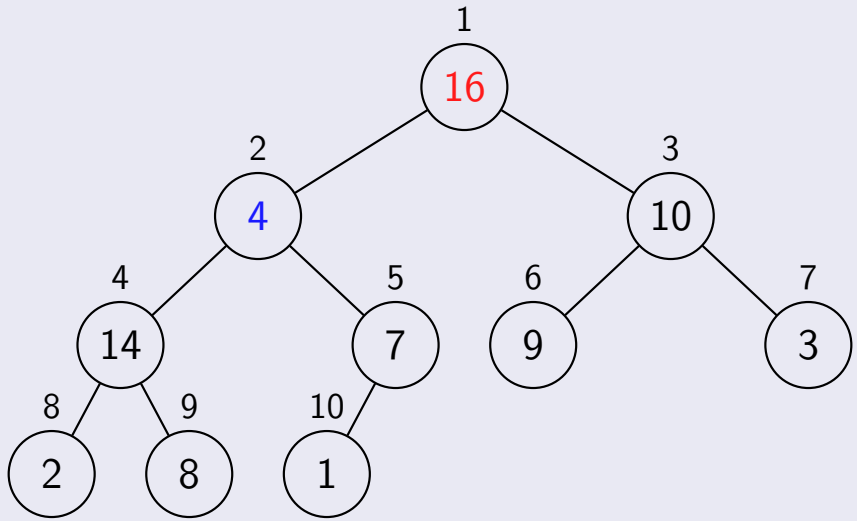
Mantendo a Propriedade do Heap

- Antes de mais nada, verificamos no heap onde houve essa violação
  - Ou ela acontece no pai do nó alterado



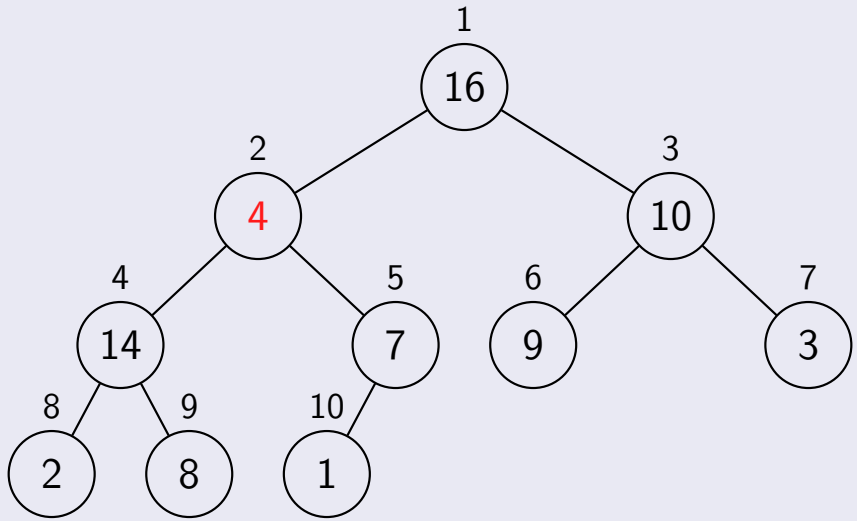
Mantendo a Propriedade do Heap

- Antes de mais nada, verificamos no heap onde houve essa violação
  - Ou ela acontece no pai do nó alterado – está ok



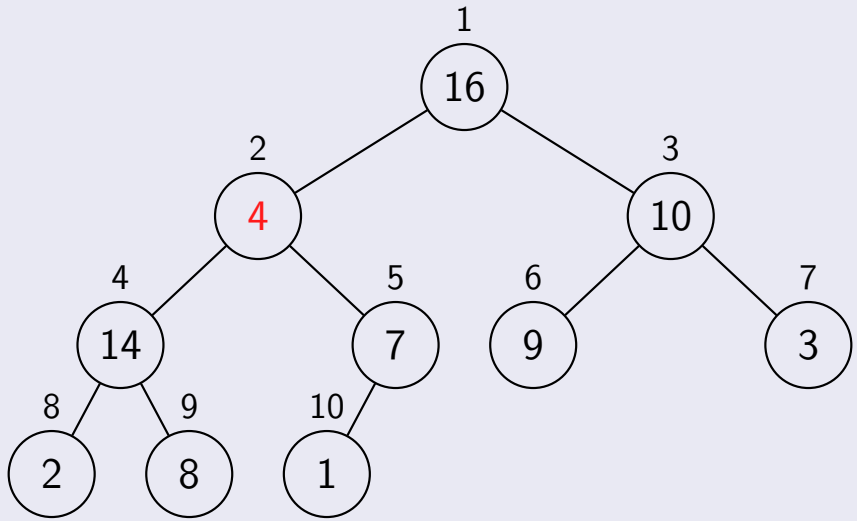
Mantendo a Propriedade do Heap

- Antes de mais nada, verificamos no heap onde houve essa violação
  - Ou ela acontece no pai do nó alterado – está ok
  - Ou no próprio nó



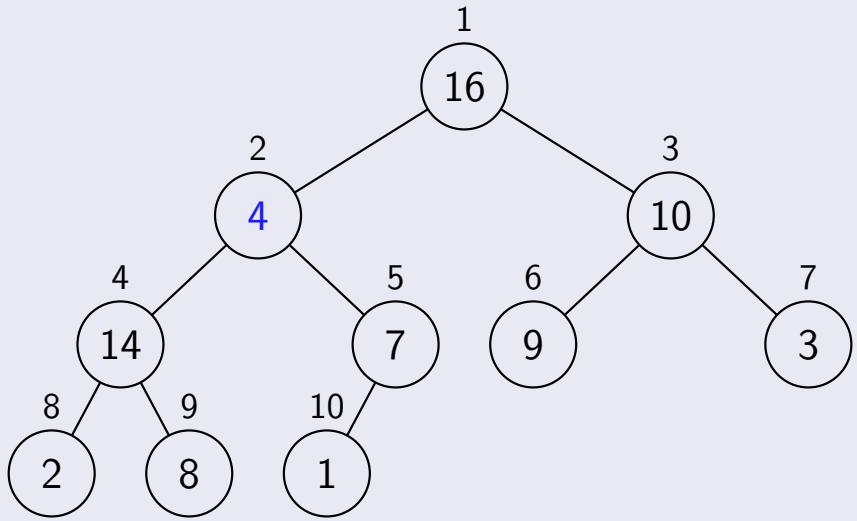
Mantendo a Propriedade do Heap

- Antes de mais nada, verificamos no heap onde houve essa violação
  - Ou ela acontece no pai do nó alterado – está ok
  - Ou no próprio nó – falhou aqui



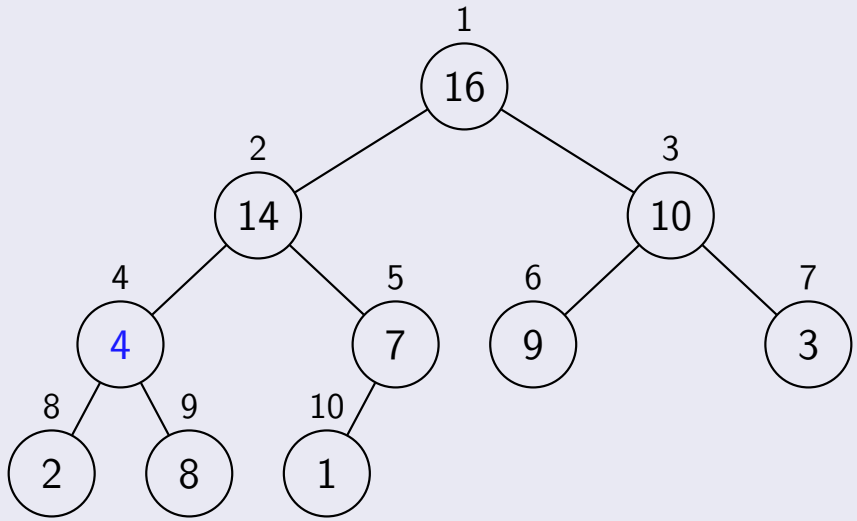
Mantendo a Propriedade do Heap

- Feito isso, trocamos o elemento problemático de lugar com o maior filho



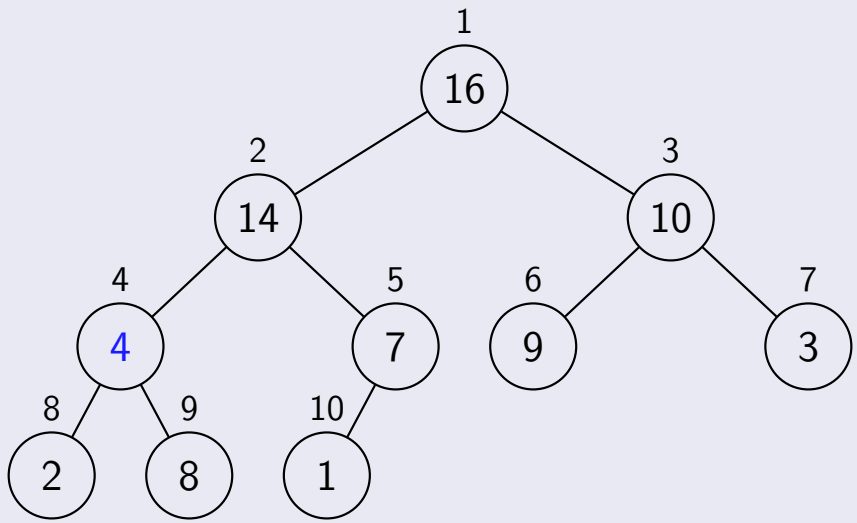
Mantendo a Propriedade do Heap

- Feito isso, trocamos o elemento problemático de lugar com o maior filho



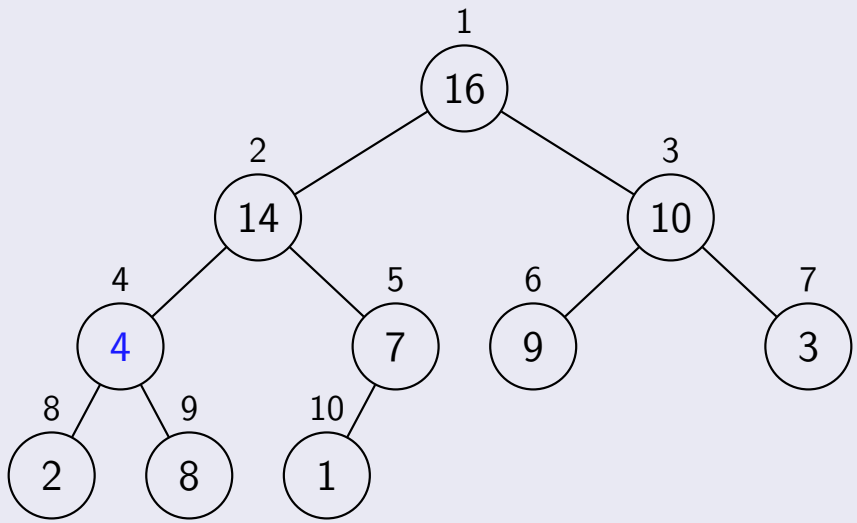
Mantendo a Propriedade do Heap

- Feito isso, trocamos o elemento problemático de lugar com o maior filho
  - Note que isso não viola a propriedade do heap



Mantendo a Propriedade do Heap

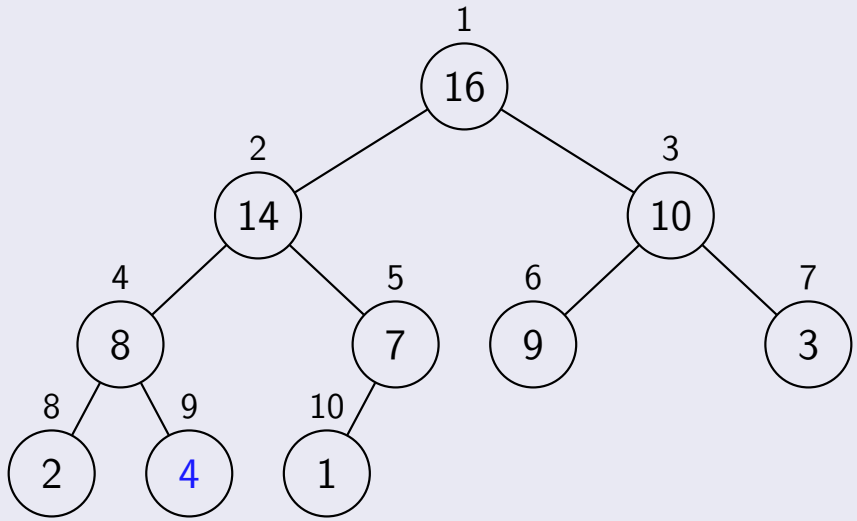
- Feito isso, trocamos o elemento problemático de lugar com o maior filho
  - Note que isso não viola a propriedade do heap



- E repetimos o procedimento enquanto houver violação da propriedade

Mantendo a Propriedade do Heap

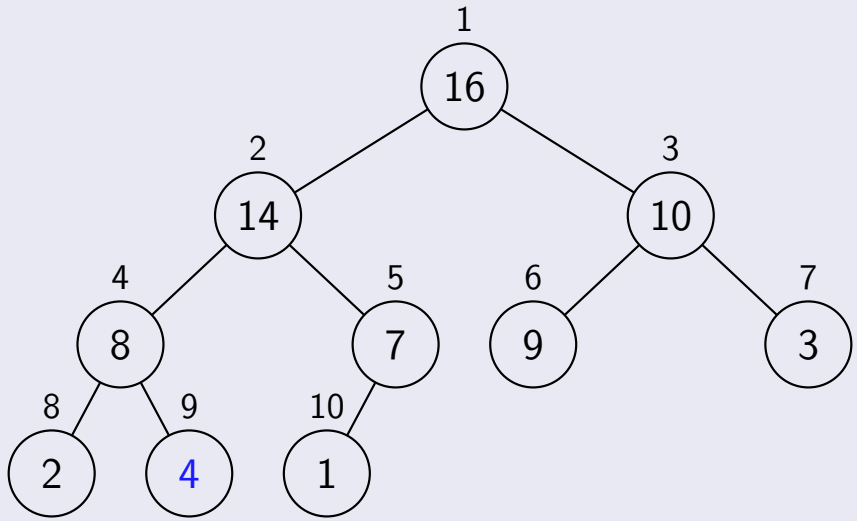
- Feito isso, trocamos o elemento problemático de lugar com o maior filho
  - Note que isso não viola a propriedade do heap



- E repetimos o procedimento enquanto houver violação da propriedade

Mantendo a Propriedade do Heap

- Feito isso, trocamos o elemento problemático de lugar com o maior filho
  - Note que isso não viola a propriedade do heap



- E repetimos o procedimento enquanto houver violação da propriedade
  - Empurramos assim o elemento problemático para baixo no heap, até seu devido lugar

Mantendo a Propriedade do Heap (Máximo)

```
void refazHeapMax(int A[], int i, int compHeap) {
  int esq, dir, maior, temp;

  esq = esquerda(i);
  dir = direita(i);
  if((esq<compHeap) && (A[esq]>A[i])) maior = esq;
  else maior = i;
  if((dir<compHeap) && (A[dir]>A[maior])) maior = dir;
  if(maior != i) {
    temp = A[i];
    A[i] = A[maior];
    A[maior] = temp;
    refazHeapMax(A, maior, compHeap);
  }
}
```

Heaps

Mantendo a Propriedade do Heap (Máximo)

```
void refazHeapMax(int A[], int i, int compHeap) {
  int esq, dir, maior, temp;

  esq = esquerda(i);
  dir = direita(i);
  if((esq<compHeap) && (A[esq]>A[i])) maior = esq;
  else maior = i;
  if((dir<compHeap) && (A[dir]>A[maior])) maior = dir;
  if(maior != i) {
    temp = A[i];
    A[i] = A[maior];
    A[maior] = temp;
    refazHeapMax(A, maior, compHeap);
  }
}
```

Arranjo representando o heap

Heaps

Mantendo a Propriedade do Heap (Máximo)

```
void refazHeapMax(int A[], int i, int compHeap) {
  int esq, dir, maior, temp;

  esq = esquerda(i);
  dir = direita(i);
  if((esq<compHeap) && (A[esq]>A[i])) maior = esq;
  else maior = i;
  if((dir<compHeap) && (A[dir]>A[maior])) maior = dir;
  if(maior != i) {
    temp = A[i];
    A[i] = A[maior];
    A[maior] = temp;
    refazHeapMax(A, maior, compHeap);
  }
}
```

Índice do elemento potencialmente problemático

Heaps

Mantendo a Propriedade do Heap (Máximo)

```
void refazHeapMax(int A[], int i, int compHeap) {
  int esq, dir, maior, temp;

  esq = esquerda(i);
  dir = direita(i);
  if((esq<compHeap) && (A[esq]>A[i])) maior = esq;
  else maior = i;
  if((dir<compHeap) && (A[dir]>A[maior])) maior = dir;
  if(maior != i) {
    temp = A[i];
    A[i] = A[maior];
    A[maior] = temp;
    refazHeapMax(A, maior, compHeap);
  }
}
```

Tamanho do heap (arranjo)

Heaps

Mantendo a Propriedade do Heap (Máximo)

```
void refazHeapMax(int A[], int i, int compHeap) {
  int esq, dir, maior, temp;

  esq = esquerda(i);
  dir = direita(i);
  if((esq<compHeap) && (A[esq]>A[i])) maior = esq;
  else maior = i;
  if((dir<compHeap) && (A[dir]>A[maior])) maior = dir;
  if(maior != i) {
    temp = A[i];
    A[i] = A[maior];
    A[maior] = temp;
    refazHeapMax(A, maior, compHeap);
  }
}
```

Retorna o índice no arranjo da subárvore à esquerda de i

Mantendo a Propriedade do Heap (Máximo)

```
void refazHeapMax(int A[], int i, int compHeap) {
  int esq, dir, maior, temp;

  esq = esquerda(i);
  dir = direita(i);
  if((esq<compHeap) && (A[esq]>A[i])) maior = esq;
  else maior = i;
  if((dir<compHeap) && (A[dir]>A[maior])) maior = dir;
  if(maior != i) {
    temp = A[i];
    A[i] = A[maior];
    A[maior] = temp;
    refazHeapMax(A, maior, compHeap);
  }
}
```

Retorna o índice no arranjo da subárvore à direita de i

Heaps

Mantendo a Propriedade do Heap (Máximo)

```
void refazHeapMax(int A[], int i, int compHeap) {
  int esq, dir, maior, temp;

  esq = esquerda(i);
  dir = direita(i);
  if((esq<compHeap) && (A[esq]>A[i])) maior = esq;
  else maior = i;
  if((dir<compHeap) && (A[dir]>A[maior])) maior = dir;
  if(maior != i) {
    temp = A[i];
    A[i] = A[maior];
    A[maior] = temp;
    refazHeapMax(A, maior, compHeap);
  }
}
```

refazHeapMax assume que essas subárvores correspondem a heaps máximos, e que A[i] pode ser menor que seus filhos

Mantendo a Propriedade do Heap (Máximo)

```
void refazHeapMax(int A[], int i, int compHeap) {
  int esq, dir, maior, temp;

  esq = esquerda(i);
  dir = direita(i);
  if((esq<compHeap) && (A[esq]>A[i])) maior = esq;
  else maior = i;
  if((dir<compHeap) && (A[dir]>A[maior])) maior = dir;
  if(maior != i) {
    temp = A[i];
    A[i] = A[maior];
    A[maior] = temp;
    refazHeapMax(A, maior, compHeap);
  }
}
```

Vê se o filho à esquerda existe, e se é maior que seu pai i

Mantendo a Propriedade do Heap (Máximo)

```
void refazHeapMax(int A[], int i, int compHeap) {
  int esq, dir, maior, temp;

  esq = esquerda(i);
  dir = direita(i);
  if((esq<compHeap) && (A[esq]>A[i])) maior = esq;
  else maior = i;
  if((dir<compHeap) && (A[dir]>A[maior])) maior = dir;
  if(maior != i) {
    temp = A[i];
    A[i] = A[maior];
    A[maior] = temp;
    refazHeapMax(A, maior, compHeap);
  }
}
```

Vê se o filho à direita existe, e se é maior que seu pai ou irmão

Mantendo a Propriedade do Heap (Máximo)

```
void refazHeapMax(int A[], int i, int compHeap) {
  int esq, dir, maior, temp;

  esq = esquerda(i);
  dir = direita(i);
  if((esq<compHeap) && (A[esq]>A[i])) maior = esq;
  else maior = i;
  if((dir<compHeap) && (A[dir]>A[maior])) maior = dir;
  if(maior != i) {
    temp = A[i];
    A[i] = A[maior];
    A[maior] = temp;
    refazHeapMax(A, maior, compHeap);
  }
}
```

Troca i de lugar com o filho maior que ele, se houver

Mantendo a Propriedade do Heap (Máximo)

```
void refazHeapMax(int A[], int i, int compHeap) {
  int esq, dir, maior, temp;

  esq = esquerda(i);
  dir = direita(i);
  if((esq<compHeap) && (A[esq]>A[i])) maior = esq;
  else maior = i;
  if((dir<compHeap) && (A[dir]>A[maior])) maior = dir;
  if(maior != i) {
    temp = A[i];
    A[i] = A[maior];
    A[maior] = temp;
    refazHeapMax(A, maior, compHeap);
  }
}
```

Vê se essa troca não gerou nova violação do heap nessa subárvore

Mantendo a Propriedade do Heap (Máximo)

```
void refazHeapMax(int A[], int i, int compHeap) {
  int esq, dir, maior, temp;

  esq = esquerda(i);
  dir = direita(i);

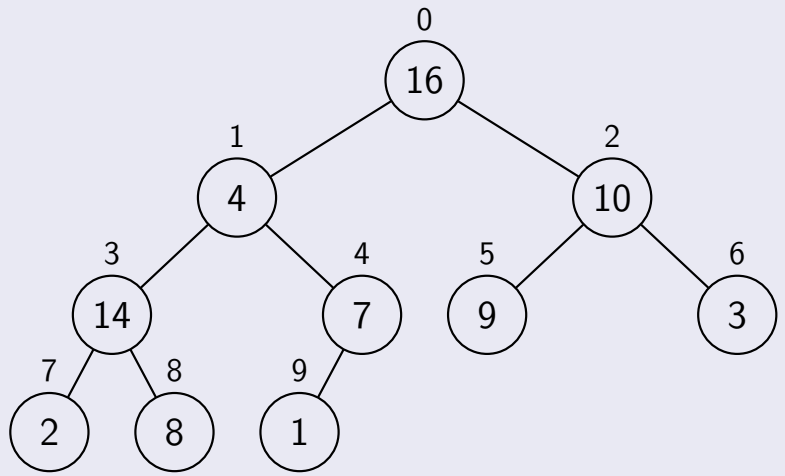
  if((esq<compHeap) && (A[esq]>A[i]))
    maior = esq;
  else maior = i;
  if((dir<compHeap) && (A[dir]>A[maior]))
    maior = dir;
  if(maior != i) {
    temp = A[i];
    A[i] = A[maior];
    A[maior] = temp;
    refazHeapMax(A, maior, compHeap);
  }
}
```

Mantendo a Propriedade do Heap (Máximo)

```
void refazHeapMax(int A[], int i, int compHeap) {
  int esq, dir, maior, temp;

  esq = esquerda(i);
  dir = direita(i);

  if((esq<compHeap) && (A[esq]>A[i]))
    maior = esq;
  else maior = i;
  if((dir<compHeap) && (A[dir]>A[maior]))
    maior = dir;
  if(maior != i) {
    temp = A[i];
    A[i] = A[maior];
    A[maior] = temp;
    refazHeapMax(A, maior, compHeap);
  }
}
```



16	4	10	14	7	9	3	2	8	1
0	1	2	3	4	5	6	7	8	9

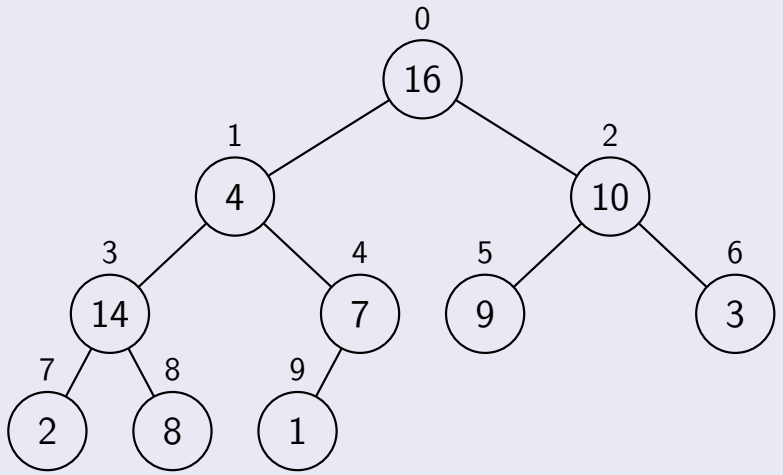
i_1

Mantendo a Propriedade do Heap (Máximo)

```
void refazHeapMax(int A[], int i, int compHeap) {
  int esq, dir, maior, temp;

  esq = esquerda(i);
  dir = direita(i);

  if((esq<compHeap) && (A[esq]>A[i]))
    maior = esq;
  else maior = i;
  if((dir<compHeap) && (A[dir]>A[maior]))
    maior = dir;
  if(maior != i) {
    temp = A[i];
    A[i] = A[maior];
    A[maior] = temp;
    refazHeapMax(A, maior, compHeap);
  }
}
```



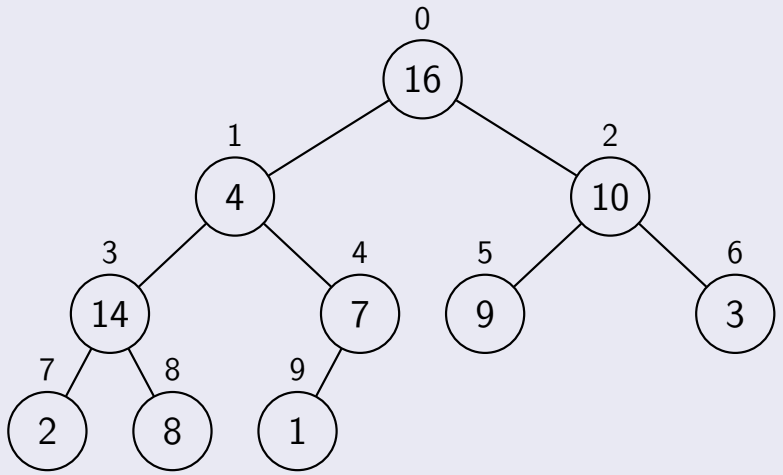
16	4	10	14	7	9	3	2	8	1
0	1	2	3	4	5	6	7	8	9
	i_1		esq_1						

Mantendo a Propriedade do Heap (Máximo)

```
void refazHeapMax(int A[], int i, int compHeap) {
  int esq, dir, maior, temp;

  esq = esquerda(i);
  dir = direita(i);

  if((esq<compHeap) && (A[esq]>A[i]))
    maior = esq;
  else maior = i;
  if((dir<compHeap) && (A[dir]>A[maior]))
    maior = dir;
  if(maior != i) {
    temp = A[i];
    A[i] = A[maior];
    A[maior] = temp;
    refazHeapMax(A, maior, compHeap);
  }
}
```



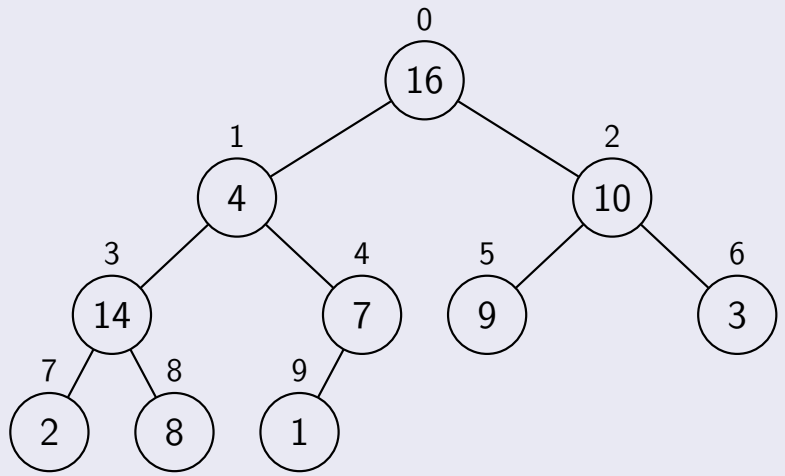
16	4	10	14	7	9	3	2	8	1
0	1	2	3	4	5	6	7	8	9
	i_1		esq_1	dir_1					

Mantendo a Propriedade do Heap (Máximo)

```
void refazHeapMax(int A[], int i, int compHeap) {
  int esq, dir, maior, temp;

  esq = esquerda(i);
  dir = direita(i);

  if((esq<compHeap) && (A[esq]>A[i]))
    maior = esq;
  else maior = i;
  if((dir<compHeap) && (A[dir]>A[maior]))
    maior = dir;
  if(maior != i) {
    temp = A[i];
    A[i] = A[maior];
    A[maior] = temp;
    refazHeapMax(A, maior, compHeap);
  }
}
```



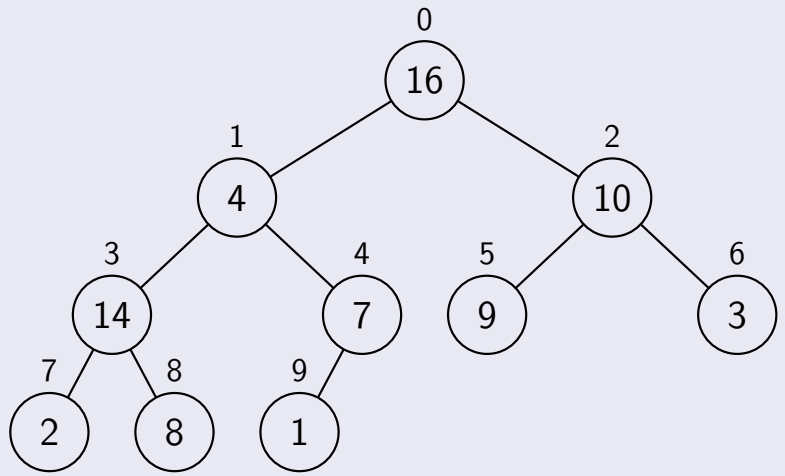
16	4	10	14	7	9	3	2	8	1
0	1	2	3	4	5	6	7	8	9
	i_1		esq_1	dir_1					

Mantendo a Propriedade do Heap (Máximo)

```
void refazHeapMax(int A[], int i, int compHeap) {
  int esq, dir, maior, temp;

  esq = esquerda(i);
  dir = direita(i);

  if((esq<compHeap) && (A[esq]>A[i]))
    maior = esq;
  else maior = i;
  if((dir<compHeap) && (A[dir]>A[maior]))
    maior = dir;
  if(maior != i) {
    temp = A[i];
    A[i] = A[maior];
    A[maior] = temp;
    refazHeapMax(A, maior, compHeap);
  }
}
```



16	4	10	14	7	9	3	2	8	1
0	1	2	3	4	5	6	7	8	9

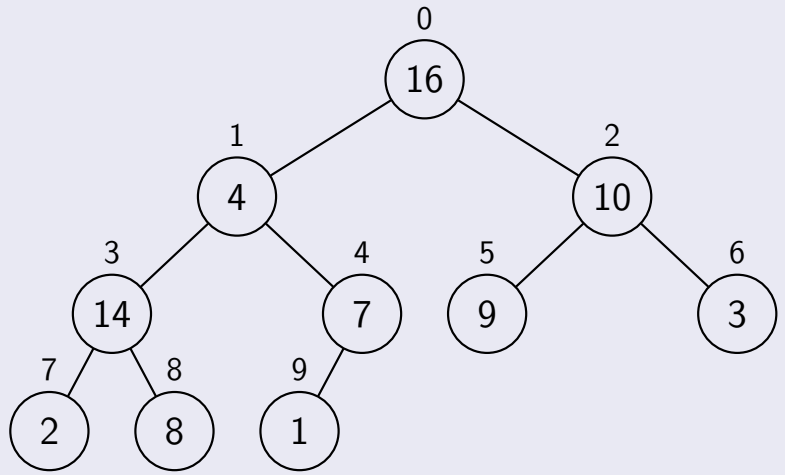
i_1 esq_1 dir_1 $maior_1$

Mantendo a Propriedade do Heap (Máximo)

```
void refazHeapMax(int A[], int i, int compHeap) {
  int esq, dir, maior, temp;

  esq = esquerda(i);
  dir = direita(i);

  if((esq<compHeap) && (A[esq]>A[i]))
    maior = esq;
  else maior = i;
  if((dir<compHeap) && (A[dir]>A[maior]))
    maior = dir;
  if(maior != i) {
    temp = A[i];
    A[i] = A[maior];
    A[maior] = temp;
    refazHeapMax(A, maior, compHeap);
  }
}
```



16	4	10	14	7	9	3	2	8	1
0	1	2	3	4	5	6	7	8	9

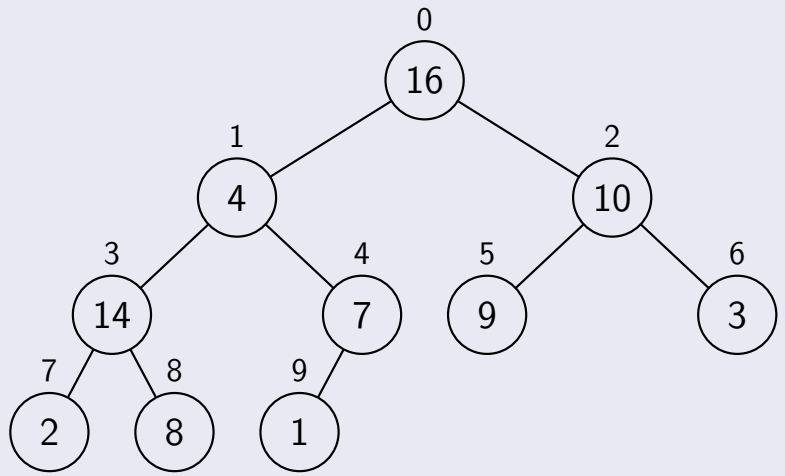
i_1 esq_1 dir_1 $maior_1$

Mantendo a Propriedade do Heap (Máximo)

```
void refazHeapMax(int A[], int i, int compHeap) {
  int esq, dir, maior, temp;

  esq = esquerda(i);
  dir = direita(i);

  if((esq<compHeap) && (A[esq]>A[i]))
    maior = esq;
  else maior = i;
  if((dir<compHeap) && (A[dir]>A[maior]))
    maior = dir;
  if(maior != i) {
    temp = A[i];
    A[i] = A[maior];
    A[maior] = temp;
    refazHeapMax(A, maior, compHeap);
  }
}
```



16	4	10	14	7	9	3	2	8	1
0	1	2	3	4	5	6	7	8	9

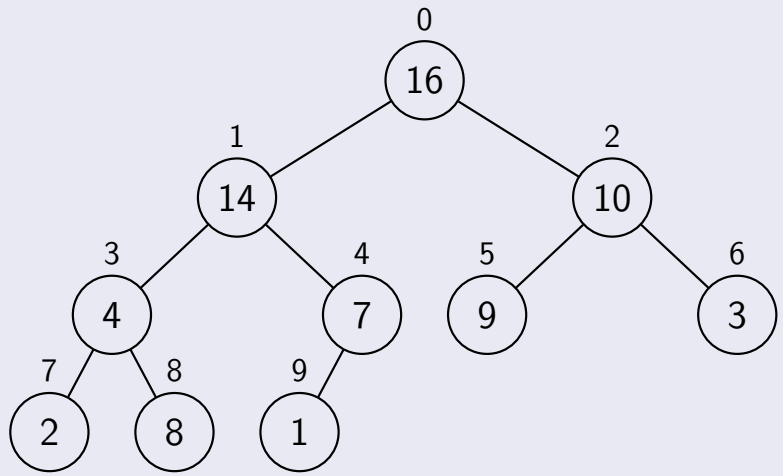
i_1 esq_1 dir_1 $maior_1$

Mantendo a Propriedade do Heap (Máximo)

```
void refazHeapMax(int A[], int i, int compHeap) {
  int esq, dir, maior, temp;

  esq = esquerda(i);
  dir = direita(i);

  if((esq<compHeap) && (A[esq]>A[i]))
    maior = esq;
  else maior = i;
  if((dir<compHeap) && (A[dir]>A[maior]))
    maior = dir;
  if(maior != i) {
    temp = A[i];
    A[i] = A[maior];
    A[maior] = temp;
    refazHeapMax(A, maior, compHeap);
  }
}
```



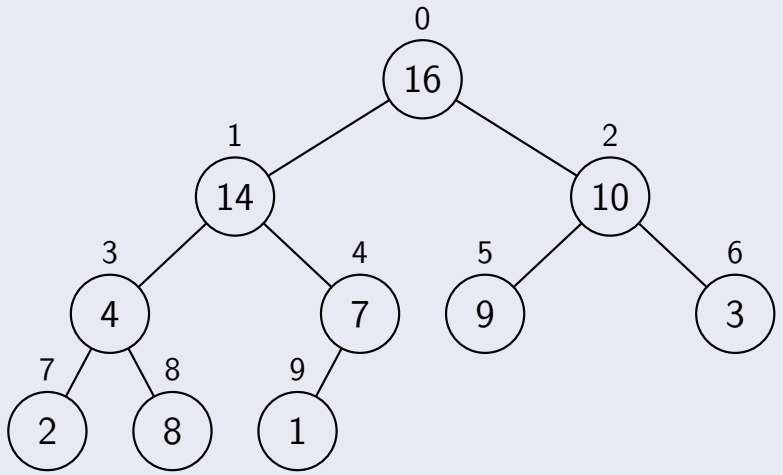
16	14	10	4	7	9	3	2	8	1
0	1	2	3	4	5	6	7	8	9
	i_1		esq_1	dir_1					
			$maior_1$						

Mantendo a Propriedade do Heap (Máximo)

```
void refazHeapMax(int A[], int i, int compHeap) {
  int esq, dir, maior, temp;

  esq = esquerda(i);
  dir = direita(i);

  if((esq<compHeap) && (A[esq]>A[i]))
    maior = esq;
  else maior = i;
  if((dir<compHeap) && (A[dir]>A[maior]))
    maior = dir;
  if(maior != i) {
    temp = A[i];
    A[i] = A[maior];
    A[maior] = temp;
    refazHeapMax(A, maior, compHeap);
  }
}
```



16	14	10	4	7	9	3	2	8	1
0	1	2	3	4	5	6	7	8	9

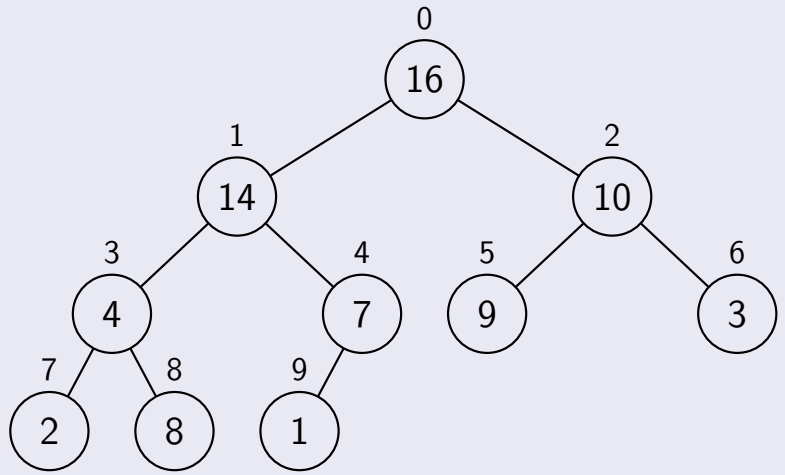
i_2

Mantendo a Propriedade do Heap (Máximo)

```
void refazHeapMax(int A[], int i, int compHeap) {
  int esq, dir, maior, temp;

  esq = esquerda(i);
  dir = direita(i);

  if((esq<compHeap) && (A[esq]>A[i]))
    maior = esq;
  else maior = i;
  if((dir<compHeap) && (A[dir]>A[maior]))
    maior = dir;
  if(maior != i) {
    temp = A[i];
    A[i] = A[maior];
    A[maior] = temp;
    refazHeapMax(A, maior, compHeap);
  }
}
```



16	14	10	4	7	9	3	2	8	1
0	1	2	3	4	5	6	7	8	9

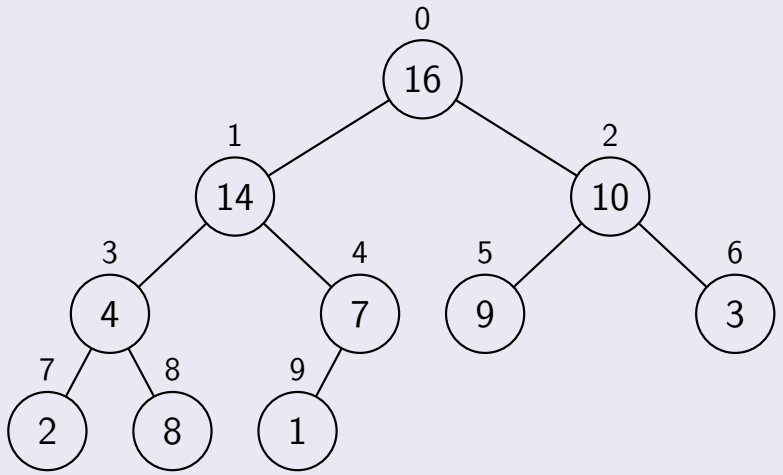
i_2 esq_2

Mantendo a Propriedade do Heap (Máximo)

```
void refazHeapMax(int A[], int i, int compHeap) {
  int esq, dir, maior, temp;

  esq = esquerda(i);
  dir = direita(i);

  if((esq<compHeap) && (A[esq]>A[i]))
    maior = esq;
  else maior = i;
  if((dir<compHeap) && (A[dir]>A[maior]))
    maior = dir;
  if(maior != i) {
    temp = A[i];
    A[i] = A[maior];
    A[maior] = temp;
    refazHeapMax(A, maior, compHeap);
  }
}
```



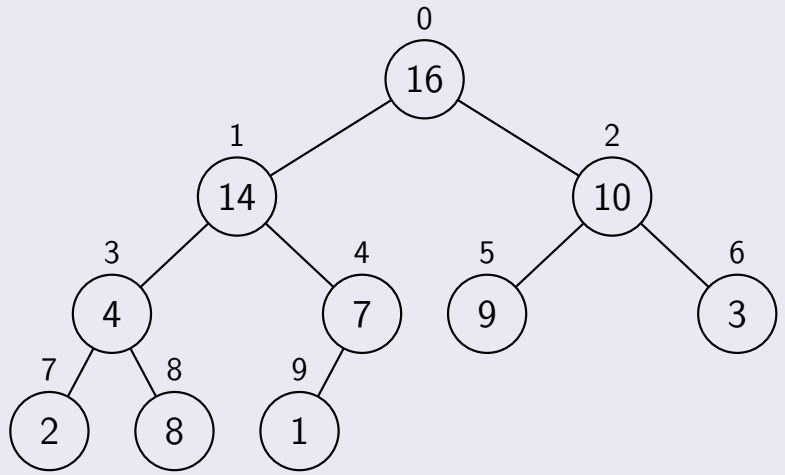
16	14	10	4	7	9	3	2	8	1
0	1	2	3	4	5	6	7	8	9
			i_2				esq_2	dir_2	

Mantendo a Propriedade do Heap (Máximo)

```
void refazHeapMax(int A[], int i, int compHeap) {
  int esq, dir, maior, temp;

  esq = esquerda(i);
  dir = direita(i);

  if((esq<compHeap) && (A[esq]>A[i]))
    maior = esq;
  else maior = i;
  if((dir<compHeap) && (A[dir]>A[maior]))
    maior = dir;
  if(maior != i) {
    temp = A[i];
    A[i] = A[maior];
    A[maior] = temp;
    refazHeapMax(A, maior, compHeap);
  }
}
```



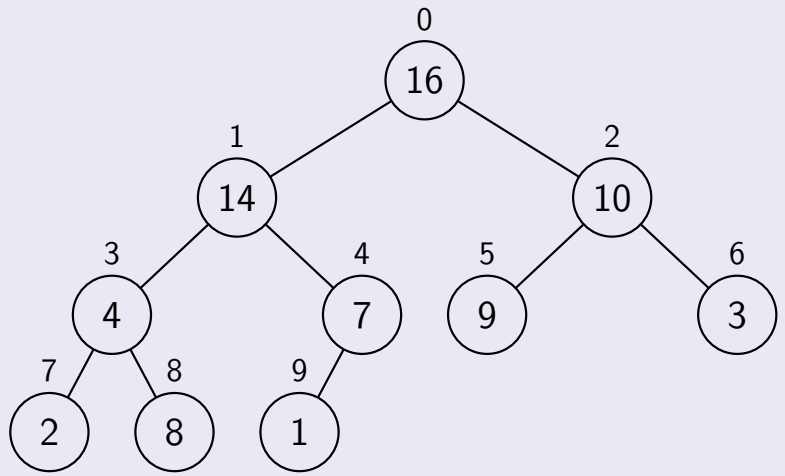
16	14	10	4	7	9	3	2	8	1
0	1	2	3	4	5	6	7	8	9
			i_2				esq_2	dir_2	

Mantendo a Propriedade do Heap (Máximo)

```
void refazHeapMax(int A[], int i, int compHeap) {
  int esq, dir, maior, temp;

  esq = esquerda(i);
  dir = direita(i);

  if((esq<compHeap) && (A[esq]>A[i]))
    maior = esq;
  else maior = i;
  if((dir<compHeap) && (A[dir]>A[maior]))
    maior = dir;
  if(maior != i) {
    temp = A[i];
    A[i] = A[maior];
    A[maior] = temp;
    refazHeapMax(A, maior, compHeap);
  }
}
```



16	14	10	4	7	9	3	2	8	1
0	1	2	3	4	5	6	7	8	9

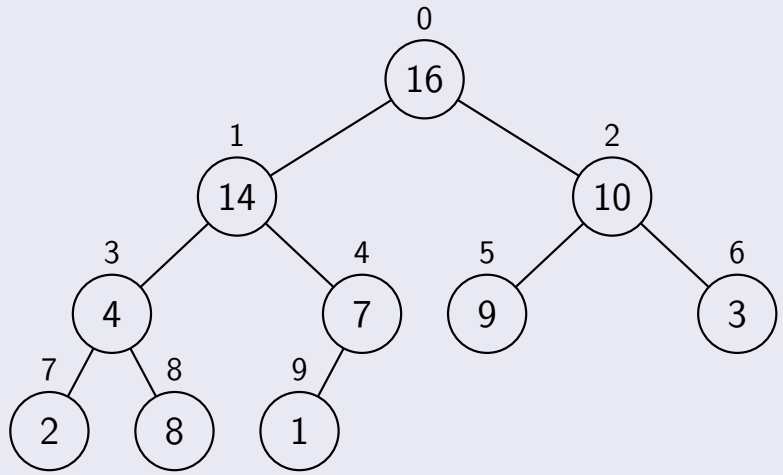
i_2 / $maior_2$ — esq_2 — dir_2

Mantendo a Propriedade do Heap (Máximo)

```
void refazHeapMax(int A[], int i, int compHeap) {
  int esq, dir, maior, temp;

  esq = esquerda(i);
  dir = direita(i);

  if((esq<compHeap) && (A[esq]>A[i]))
    maior = esq;
  else maior = i;
  if((dir<compHeap) && (A[dir]>A[maior]))
    maior = dir;
  if(maior != i) {
    temp = A[i];
    A[i] = A[maior];
    A[maior] = temp;
    refazHeapMax(A, maior, compHeap);
  }
}
```



16	14	10	4	7	9	3	2	8	1
0	1	2	3	4	5	6	7	8	9

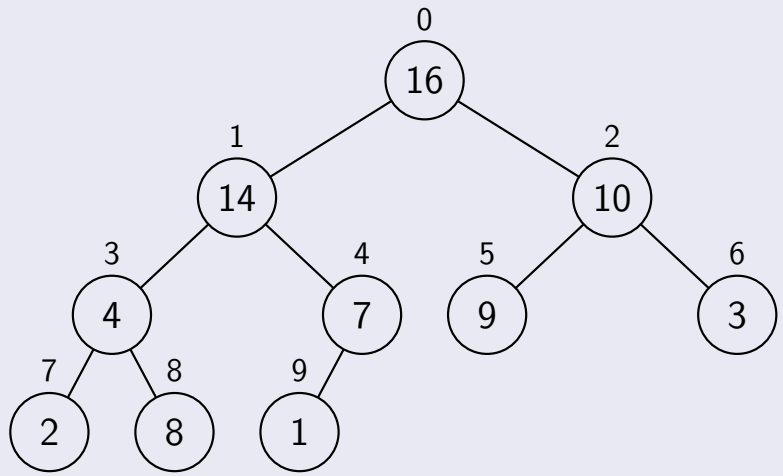
i_2 / $maior_2$ — esq_2 — dir_2

Mantendo a Propriedade do Heap (Máximo)

```
void refazHeapMax(int A[], int i, int compHeap) {
  int esq, dir, maior, temp;

  esq = esquerda(i);
  dir = direita(i);

  if((esq<compHeap) && (A[esq]>A[i]))
    maior = esq;
  else maior = i;
  if((dir<compHeap) && (A[dir]>A[maior]))
    maior = dir;
  if(maior != i) {
    temp = A[i];
    A[i] = A[maior];
    A[maior] = temp;
    refazHeapMax(A, maior, compHeap);
  }
}
```



16	14	10	4	7	9	3	2	8	1
0	1	2	3	4	5	6	7	8	9

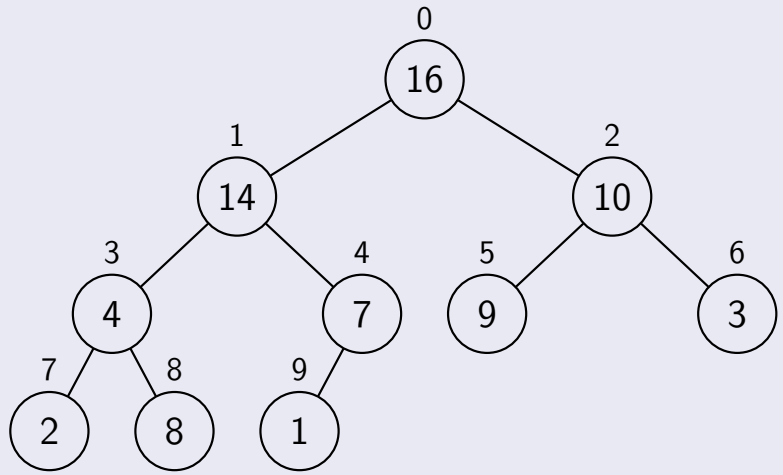
i_2 esq_2 dir_2 $maior_2$

Mantendo a Propriedade do Heap (Máximo)

```
void refazHeapMax(int A[], int i, int compHeap) {
  int esq, dir, maior, temp;

  esq = esquerda(i);
  dir = direita(i);

  if((esq<compHeap) && (A[esq]>A[i]))
    maior = esq;
  else maior = i;
  if((dir<compHeap) && (A[dir]>A[maior]))
    maior = dir;
  if(maior != i) {
    temp = A[i];
    A[i] = A[maior];
    A[maior] = temp;
    refazHeapMax(A, maior, compHeap);
  }
}
```



16	14	10	4	7	9	3	2	8	1
0	1	2	3	4	5	6	7	8	9

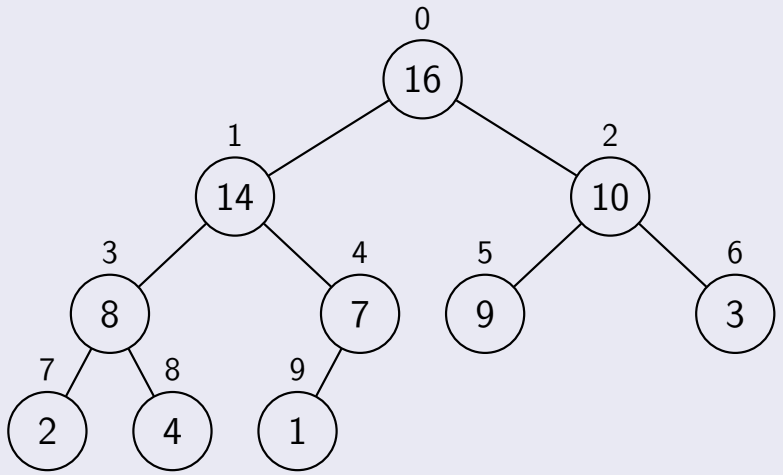
i_2 esq_2 dir_2 $maior_2$

Mantendo a Propriedade do Heap (Máximo)

```
void refazHeapMax(int A[], int i, int compHeap) {
  int esq, dir, maior, temp;

  esq = esquerda(i);
  dir = direita(i);

  if((esq<compHeap) && (A[esq]>A[i]))
    maior = esq;
  else maior = i;
  if((dir<compHeap) && (A[dir]>A[maior]))
    maior = dir;
  if(maior != i) {
    temp = A[i];
    A[i] = A[maior];
    A[maior] = temp;
    refazHeapMax(A, maior, compHeap);
  }
}
```



16	14	10	8	7	9	3	2	4	1
0	1	2	3	4	5	6	7	8	9

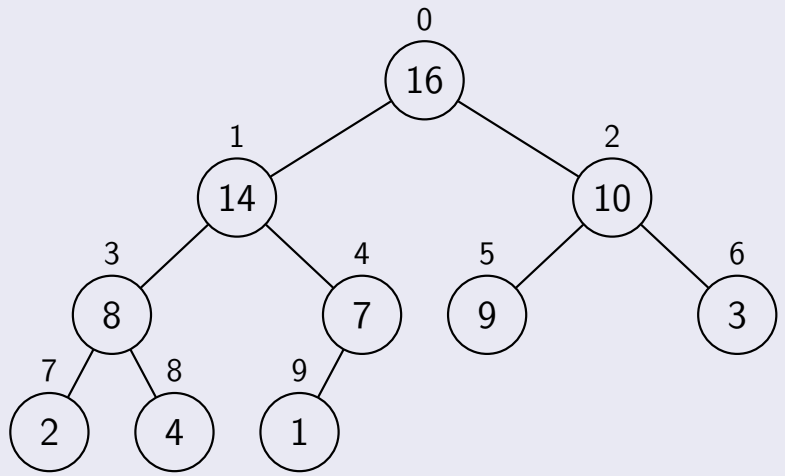
i_2 esq_2 dir_2 $maior_2$

Mantendo a Propriedade do Heap (Máximo)

```
void refazHeapMax(int A[], int i, int compHeap) {
  int esq, dir, maior, temp;

  esq = esquerda(i);
  dir = direita(i);

  if((esq<compHeap) && (A[esq]>A[i]))
    maior = esq;
  else maior = i;
  if((dir<compHeap) && (A[dir]>A[maior]))
    maior = dir;
  if(maior != i) {
    temp = A[i];
    A[i] = A[maior];
    A[maior] = temp;
    refazHeapMax(A, maior, compHeap);
  }
}
```



16	14	10	8	7	9	3	2	4	1
0	1	2	3	4	5	6	7	8	9

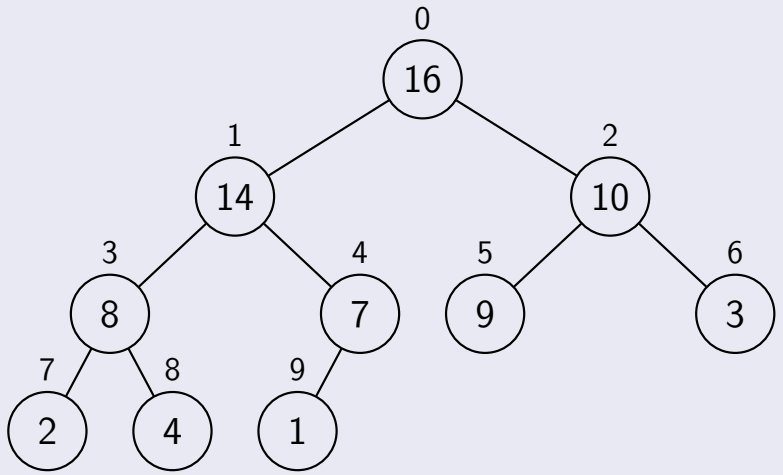
i_3

Mantendo a Propriedade do Heap (Máximo)

```
void refazHeapMax(int A[], int i, int compHeap) {
  int esq, dir, maior, temp;

  esq = esquerda(i);
  dir = direita(i);

  if((esq<compHeap) && (A[esq]>A[i]))
    maior = esq;
  else maior = i;
  if((dir<compHeap) && (A[dir]>A[maior]))
    maior = dir;
  if(maior != i) {
    temp = A[i];
    A[i] = A[maior];
    A[maior] = temp;
    refazHeapMax(A, maior, compHeap);
  }
}
```



16	14	10	8	7	9	3	2	4	1
0	1	2	3	4	5	6	7	8	9

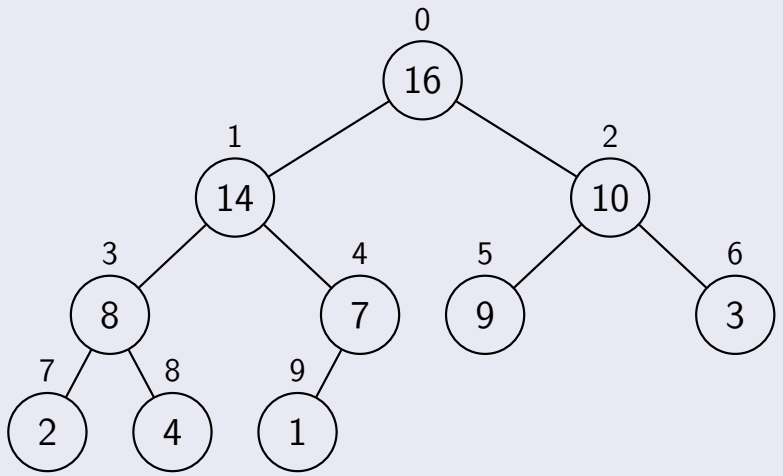
i_3

Mantendo a Propriedade do Heap (Máximo)

```
void refazHeapMax(int A[], int i, int compHeap) {
  int esq, dir, maior, temp;

  esq = esquerda(i);
  dir = direita(i);

  if((esq<compHeap) && (A[esq]>A[i]))
    maior = esq;
  else maior = i;
  if((dir<compHeap) && (A[dir]>A[maior]))
    maior = dir;
  if(maior != i) {
    temp = A[i];
    A[i] = A[maior];
    A[maior] = temp;
    refazHeapMax(A, maior, compHeap);
  }
}
```



16	14	10	8	7	9	3	2	4	1
0	1	2	3	4	5	6	7	8	9

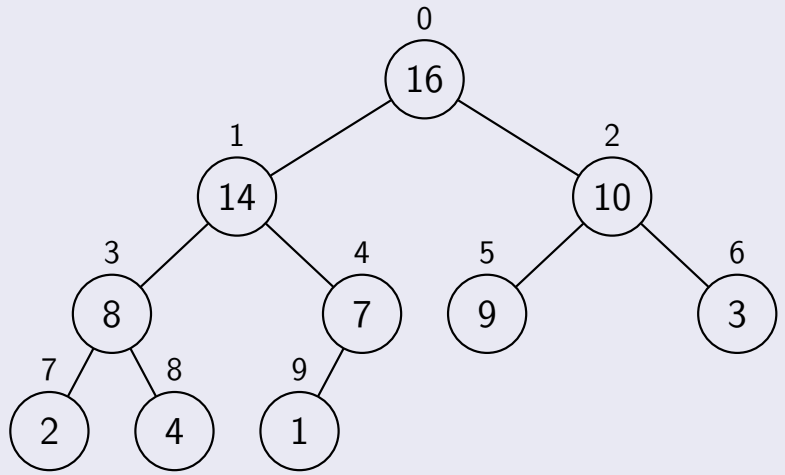
i_3

Mantendo a Propriedade do Heap (Máximo)

```
void refazHeapMax(int A[], int i, int compHeap) {
  int esq, dir, maior, temp;

  esq = esquerda(i);
  dir = direita(i);

  if((esq<compHeap) && (A[esq]>A[i]))
    maior = esq;
  else maior = i;
  if((dir<compHeap) && (A[dir]>A[maior]))
    maior = dir;
  if(maior != i) {
    temp = A[i];
    A[i] = A[maior];
    A[maior] = temp;
    refazHeapMax(A, maior, compHeap);
  }
}
```



16	14	10	8	7	9	3	2	4	1
0	1	2	3	4	5	6	7	8	9

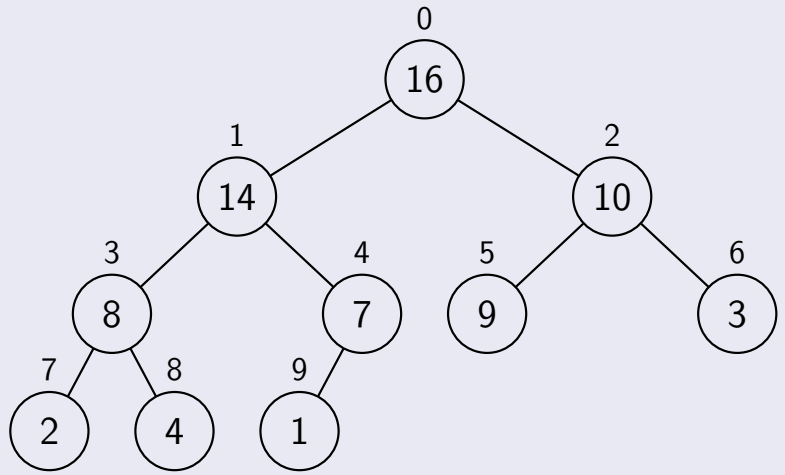
i_3

Mantendo a Propriedade do Heap (Máximo)

```
void refazHeapMax(int A[], int i, int compHeap) {
  int esq, dir, maior, temp;

  esq = esquerda(i);
  dir = direita(i);

  if((esq<compHeap) && (A[esq]>A[i]))
    maior = esq;
  else maior = i;
  if((dir<compHeap) && (A[dir]>A[maior]))
    maior = dir;
  if(maior != i) {
    temp = A[i];
    A[i] = A[maior];
    A[maior] = temp;
    refazHeapMax(A, maior, compHeap);
  }
}
```



16	14	10	8	7	9	3	2	4	1
0	1	2	3	4	5	6	7	8	9

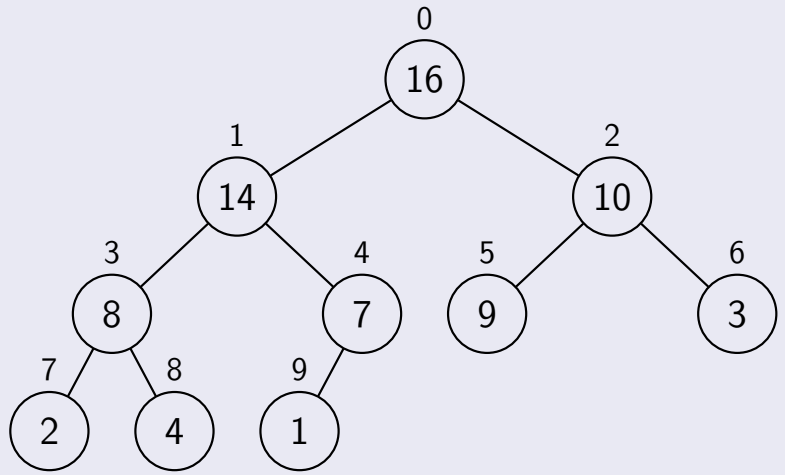
i_3
$maior_3$

Mantendo a Propriedade do Heap (Máximo)

```
void refazHeapMax(int A[], int i, int compHeap) {
  int esq, dir, maior, temp;

  esq = esquerda(i);
  dir = direita(i);

  if((esq<compHeap) && (A[esq]>A[i]))
    maior = esq;
  else maior = i;
  if((dir<compHeap) && (A[dir]>A[maior]))
    maior = dir;
  if(maior != i) {
    temp = A[i];
    A[i] = A[maior];
    A[maior] = temp;
    refazHeapMax(A, maior, compHeap);
  }
}
```



16	14	10	8	7	9	3	2	4	1
0	1	2	3	4	5	6	7	8	9

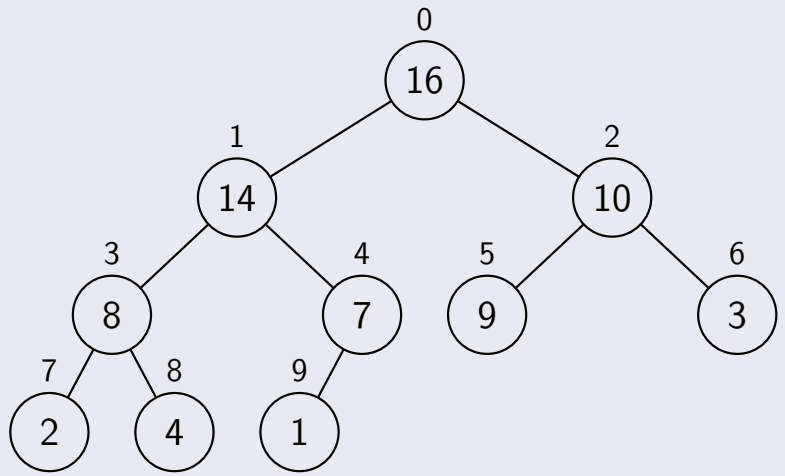
i_3
$maior_3$

Mantendo a Propriedade do Heap (Máximo)

```
void refazHeapMax(int A[], int i, int compHeap) {
  int esq, dir, maior, temp;

  esq = esquerda(i);
  dir = direita(i);

  if((esq<compHeap) && (A[esq]>A[i]))
    maior = esq;
  else maior = i;
  if((dir<compHeap) && (A[dir]>A[maior]))
    maior = dir;
  if(maior != i) {
    temp = A[i];
    A[i] = A[maior];
    A[maior] = temp;
    refazHeapMax(A, maior, compHeap);
  }
}
```



16	14	10	8	7	9	3	2	4	1
0	1	2	3	4	5	6	7	8	9

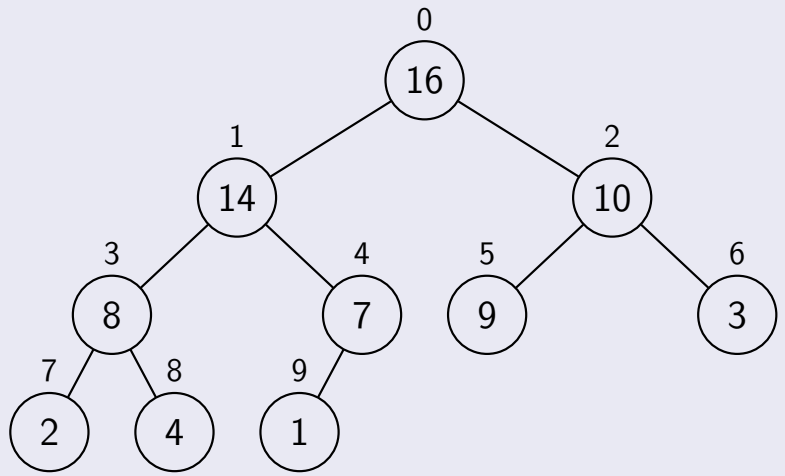
i_3
$maior_3$

Mantendo a Propriedade do Heap (Máximo)

```
void refazHeapMax(int A[], int i, int compHeap) {
  int esq, dir, maior, temp;

  esq = esquerda(i);
  dir = direita(i);

  if((esq<compHeap) && (A[esq]>A[i]))
    maior = esq;
  else maior = i;
  if((dir<compHeap) && (A[dir]>A[maior]))
    maior = dir;
  if(maior != i) {
    temp = A[i];
    A[i] = A[maior];
    A[maior] = temp;
    refazHeapMax(A, maior, compHeap);
  }
}
```



16	14	10	8	7	9	3	2	4	1
0	1	2	3	4	5	6	7	8	9

i_3
$maior_3$

Mantendo a Propriedade do Heap (Máximo)

- Em nosso algoritmo, usamos `esquerda(i)` e `direita(i)`

Heaps

Mantendo a Propriedade do Heap (Máximo)

- Em nosso algoritmo, usamos `esquerda(i)` e `direita(i)`
- Como implementá-las?

Mantendo a Propriedade do Heap (Máximo)

- Em nosso algoritmo, usamos `esquerda(i)` e `direita(i)`
- Como implementá-las?
  - ```
    int esquerda(int i) {
      return 2*i + 1;
    }
    ```
    ```

Heaps

Mantendo a Propriedade do Heap (Máximo)

- Em nosso algoritmo, usamos `esquerda(i)` e `direita(i)`
- Como implementá-las?
  - ```
    int esquerda(int i) {
      return 2*i + 1;
    }
    ```
  - ```
    int direita(int i) {
      return 2*i + 2;
    }
    ```

Mantendo a Propriedade do Heap (Máximo)

```
void refazHeapMax(int A[], int i, int compHeap) {
  int esq, dir, maior, temp;

  esq = esquerda(i);
  dir = direita(i);

  if((esq<compHeap) && (A[esq]>A[i]))
    maior = esq;
  else maior = i;
  if((dir<compHeap) && (A[dir]>A[maior]))
    maior = dir;
  if(maior != i) {
    temp = A[i];
    A[i] = A[maior];
    A[maior] = temp;
    refazHeapMax(A, maior, compHeap);
  }
}
```

- E qual a complexidade do `refazHeapMax`?

Mantendo a Propriedade do Heap (Máximo)

```
void refazHeapMax(int A[], int i, int compHeap) {
  int esq, dir, maior, temp;

  esq = esquerda(i);
  dir = direita(i);

  if((esq<compHeap) && (A[esq]>A[i]))
    maior = esq;
  else maior = i;
  if((dir<compHeap) && (A[dir]>A[maior]))
    maior = dir;
  if(maior != i) {
    temp = A[i];
    A[i] = A[maior];
    A[maior] = temp;
    refazHeapMax(A, maior, compHeap);
  }
}
```

- E qual a complexidade do `refazHeapMax`?
  - $\Theta(1)$

Mantendo a Propriedade do Heap (Máximo)

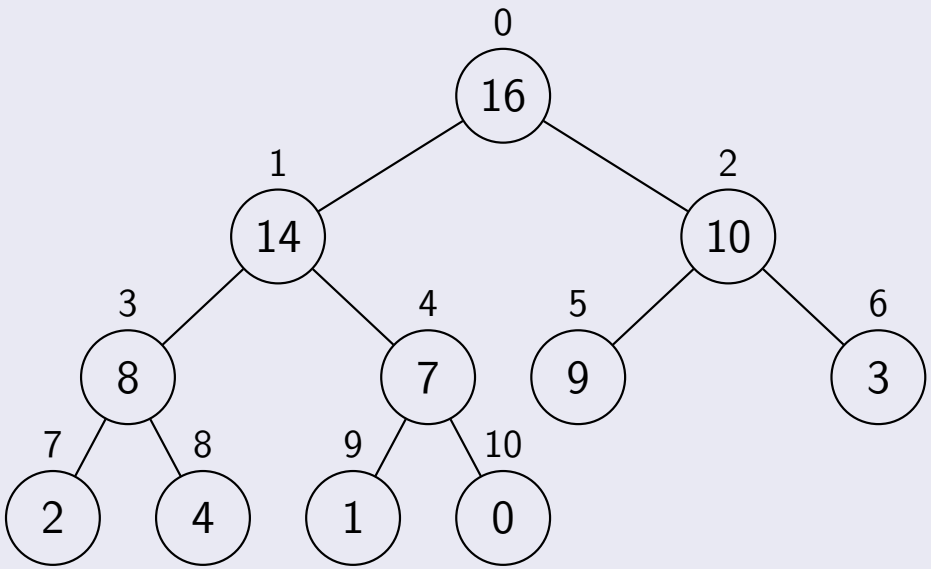
```
void refazHeapMax(int A[], int i, int compHeap) {
  int esq, dir, maior, temp;

  esq = esquerda(i);
  dir = direita(i);

  if((esq<compHeap) && (A[esq]>A[i]))
    maior = esq;
  else maior = i;
  if((dir<compHeap) && (A[dir]>A[maior]))
    maior = dir;
  if(maior != i) {
    temp = A[i];
    A[i] = A[maior];
    A[maior] = temp;
    refazHeapMax(A, maior, compHeap);
  }
}
```

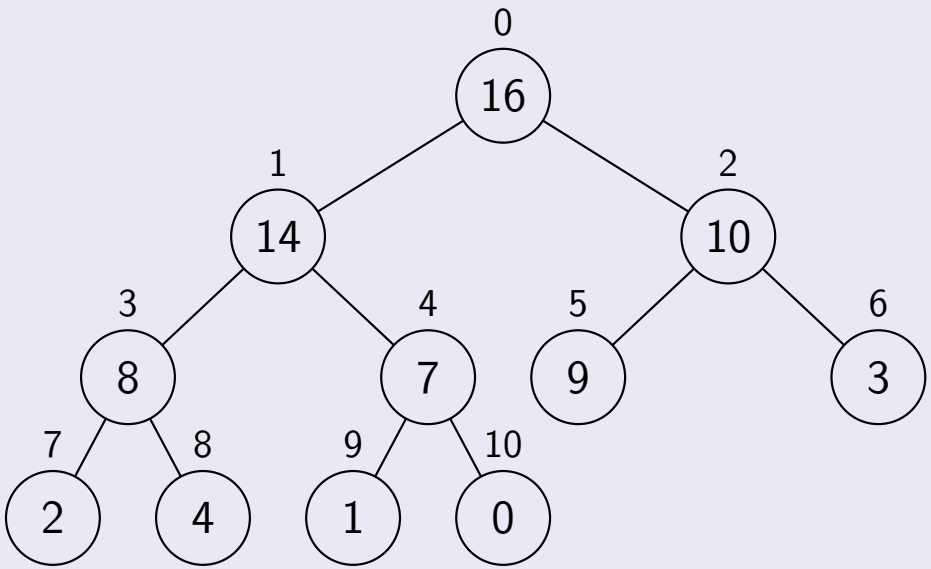
- E qual a complexidade do `refazHeapMax`?
  - $\Theta(1)$
  - $T(???)$

Mantendo a Propriedade do Heap (Máximo)



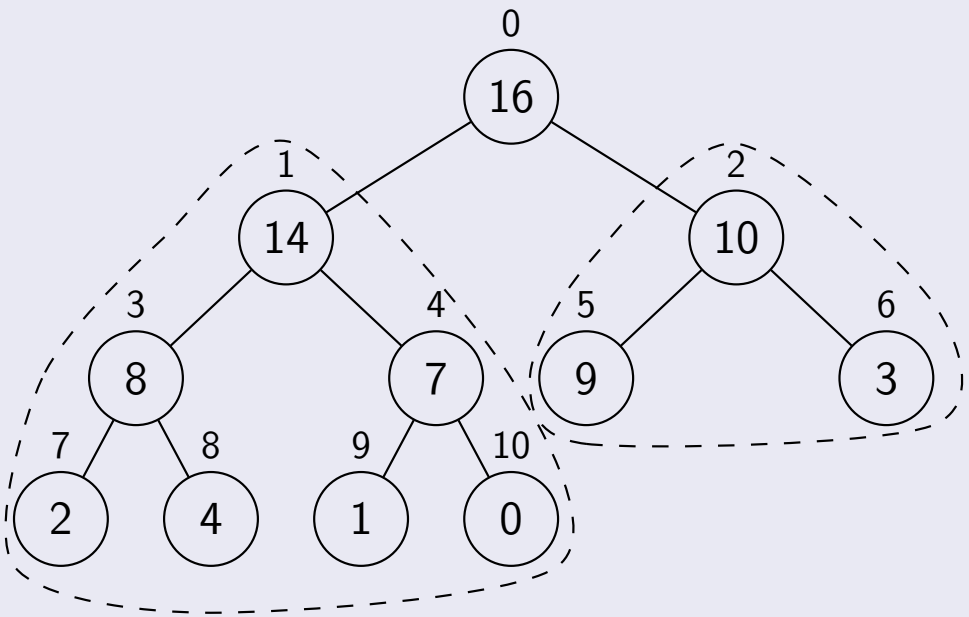
- Note que abaixo de cada nó há uma subárvore de, no máximo, $2n/3$ nós

Mantendo a Propriedade do Heap (Máximo)



- Note que abaixo de cada nó há uma subárvore de, no máximo, $2n/3$ nós
  - O pior caso sendo quando a última camada está metade cheia

Mantendo a Propriedade do Heap (Máximo)



- Note que abaixo de cada nó há uma subárvore de, no máximo, $2n/3$ nós
  - O pior caso sendo quando a última camada está metade cheia
  - Caso em que uma subárvore terá $1/3$ e a outra $2/3$ dos nós

Mantendo a Propriedade do Heap (Máximo)

```
void refazHeapMax(int A[], int i,
                            int compHeap) {
  int esq, dir, maior, temp;

  esq = esquerda(i);
  dir = direita(i);

  if((esq<compHeap) && (A[esq]>A[i]))
    maior = esq;
  else maior = i;
  if((dir<compHeap) && (A[dir]>A[maior]))
    maior = dir;
  if(maior != i) {
    temp = A[i];
    A[i] = A[maior];
    A[maior] = temp;
    refazHeapMax(A, maior, compHeap);
  }
}
```

- Então, temos que a chamada recursiva é $T(2n/3)$

Mantendo a Propriedade do Heap (Máximo)

```
void refazHeapMax(int A[], int i,
                          int compHeap) {
  int esq, dir, maior, temp;

  esq = esquerda(i);
  dir = direita(i);

  if((esq<compHeap) && (A[esq]>A[i]))
    maior = esq;
  else maior = i;
  if((dir<compHeap) && (A[dir]>A[maior]))
    maior = dir;
  if(maior != i) {
    temp = A[i];
    A[i] = A[maior];
    A[maior] = temp;
    refazHeapMax(A, maior, compHeap);
  }
}
```

- Então, temos que a chamada recursiva é $T(2n/3)$
  - E a relação de recorrência fica $T(n) = T(2n/3) + \Theta(1)$

Mantendo a Propriedade do Heap (Máximo)

```
void refazHeapMax(int A[], int i,
                          int compHeap) {
  int esq, dir, maior, temp;

  esq = esquerda(i);
  dir = direita(i);

  if((esq<compHeap) && (A[esq]>A[i]))
    maior = esq;
  else maior = i;
  if((dir<compHeap) && (A[dir]>A[maior]))
    maior = dir;
  if(maior != i) {
    temp = A[i];
    A[i] = A[maior];
    A[maior] = temp;
    refazHeapMax(A, maior, compHeap);
  }
}
```

- Então, temos que a chamada recursiva é $T(2n/3)$
  - E a relação de recorrência fica $T(n) = T(2n/3) + \Theta(1)$
  - Que, pelo Teorema Mestre (caso 2), tem solução $T(n) = O(log\ n)$

Construindo um Heap (Máximo)

- Vimos então como representar um heap

Construindo um Heap (Máximo)

- Vimos então como representar um heap
- Também vimos como manter a propriedade do heap, quando alguma alteração é feita nele

Construindo um Heap (Máximo)

- Vimos então como representar um heap
- Também vimos como manter a propriedade do heap, quando alguma alteração é feita nele
- E como fazemos para construir um heap?

Heaps

Construindo um Heap (Máximo)

- Vimos então como representar um heap
- Também vimos como manter a propriedade do heap, quando alguma alteração é feita nele
- E como fazemos para construir um heap?
  - Na próxima aula...

Referências

- Ziviani, Nivio. Projeto de Algoritmos: com implementações em Java e C++. Cengage. 2007.
- Cormen, Thomas H., Leiserson, Charles E., Rivest, Ronald L., Stein, Clifford. Introduction to Algorithms. 2a ed. MIT Press, 2001.
- Slides dos professores Delano Beder e Marcos Chain

Aula 19 – Heaps

Norton T. Roman & Luciano A. Digiampietri
digiampietri@usp.br
@digiampietri

2023