UniCesumar

Estrutura de Dados II

Prof. Me. Pietro M. de Oliveira

Heapsort

Heap

Estrutura de dados muito comum na implementação de filas de prioridade

- Agrupar elementos nos quais cada um tem maior ou menor importância
- Inserir elementos a qualquer instante e em qualquer posição, de acordo com a prioridade
- Remoção apenas no elemento de maior prioridade

Heap: permite inserção e remoção em tempo logarítmico (muito eficiente)

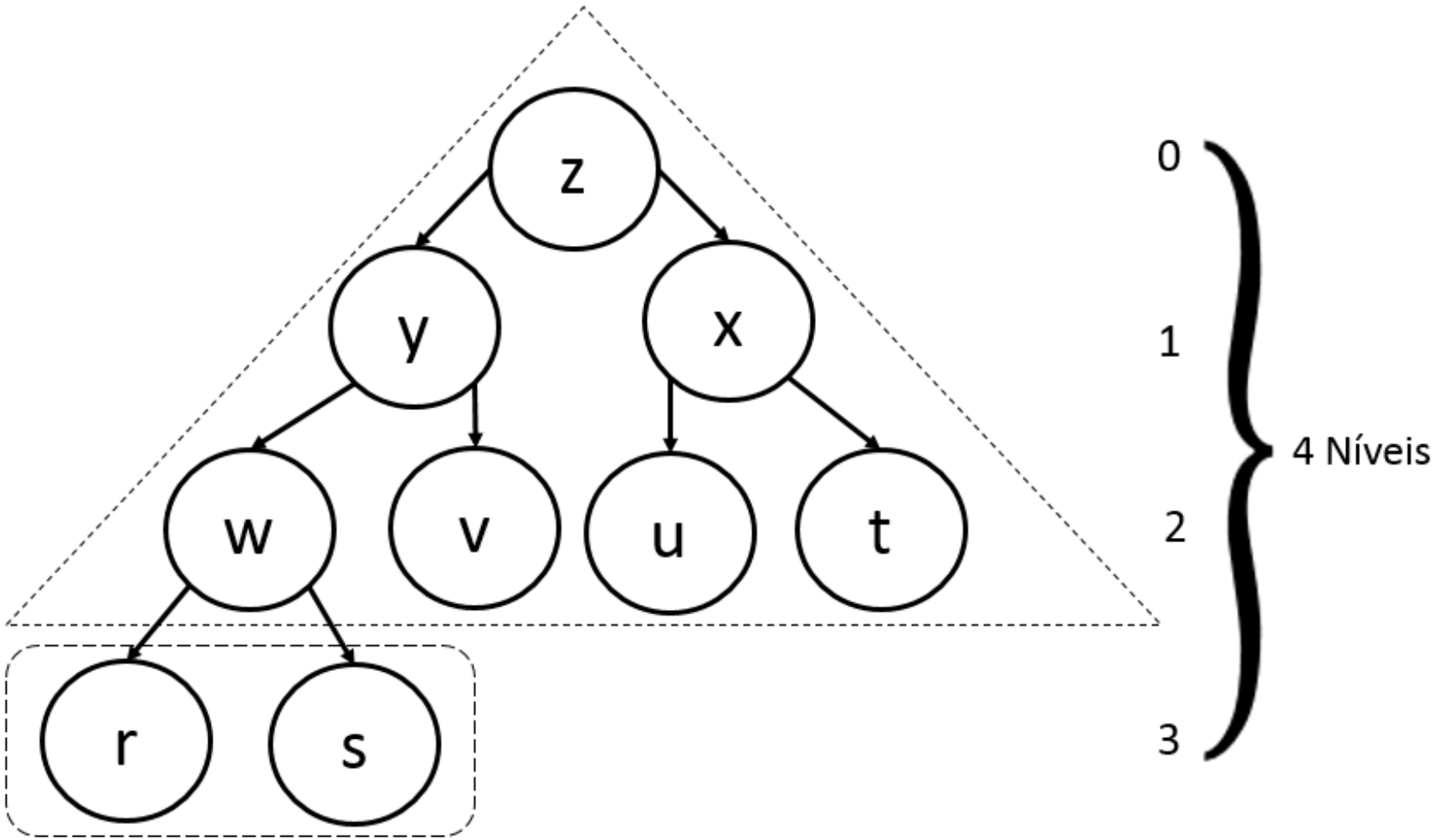
- "Transformação" de um "vetor linear" em uma "árvore binária"

Heap

Definição: *heap* é uma árvore binária com propriedades adicionas

- ***N*** níveis, **de 0 a *N-1***
- **Árvore completa** até, pelo menos, o nível *N-2* (completa **até o penúltimo nível**)
- **Nós** no **nível *N-1*** (**último**) devem estar "**tão à esquerda quanto possível**"
- ***Heap* Máxima** (*max-heap*): um **nó pai** sempre é **maior ou igual** aos seus **filhos**
- ***Heap* Mínima** (*min-heap*): um **nó pai** sempre é **menor ou igual** aos seus **filhos**

Heap



Heap

Para evitar a utilização de alocação dinâmica e memória auxiliar:

$$F_e = P*2 + 1$$

$$F_d = P*2 + 2$$

Considere o seguinte conjunto de dados armazenados em um vetor *A*:

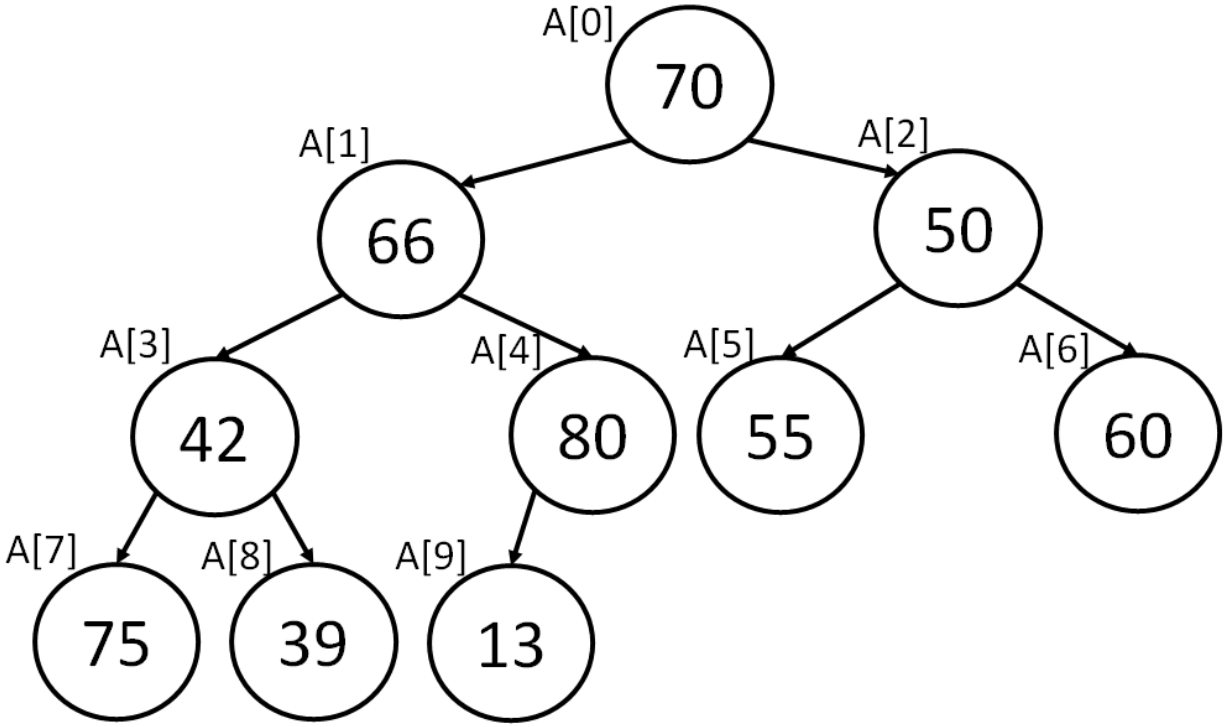
A

0	1	2	3	4	5	6	7	8	9
70	66	50	42	80	55	60	75	39	13

Heap

Aplicando-se as fórmulas, teríamos algo como:

i	0	1	2	3	4	5	6	7	8	9
A[i]	70	66	50	42	80	55	60	75	39	13



UniCesumar

Heapsort

1. **Formatar** o **vetor** como uma ***heap***
   - ConstroiHeap() → Heapifica(): nós "não-folha"
   - Assumir a seguinte **consideração**:
     - **Porções iniciais**: porção do **vetor** "**desordenado**" (**dentro da *heap***)
     - **Porções finais**: porção **vetor** "**ordenado**" (**fora da *heap***)
2. O **maior elemento** da porção "desordenada": **raiz**
   - Para ordenar esse elemento basta troca-lo para a última posição da porção desordenada: porção desordenada diminui & porção ordenada cresce em tamanho.
3. *Heap* "estragada" (pela **troca da raiz**): Heapifica()

Heapsort

Heapsort(arranjo ***A, fim***)
1. ***n*** ← ***fim***
2. ***ConstroiHeap***(***A, fim***)
3. Para ***i*** de ***n*** até ***1*** faça
4. troca ***A[1]*** ↔ ***A[i]***
5. ***n*** ← ***n-1***
6. ***Heapifica***(***A, n, 0***)

ConstroiHeap(arranjo ***A, fim***)
1. Para ***i*** de ***fim /2*** até ***0*** faça
2. ***Heapifica***(***A, fim, i***)

Vetor de tamanho *TAM* cujas posições vão de 0 a *TAM*-1 (*fim*)

Heapsort

Heapifica(arranjo ***A, fim, i***)

1. ***e*** ← ***2*i + 1***
2. ***d*** ← ***2*i + 2***
3. Se ***e ≤ fim*** e ***A[e] > A[i]*** então
4. ***maior*** ← ***e***
5. Senão
6. ***maior*** ← ***i***
7. Se ***d ≤ fim*** e ***A[d] > A[maior]*** então
8. ***maior*** ← ***d***
9. Se ***maior ≠ i*** então
10. troca ***A[i]*** ↔ ***A[maior]***
11. ***Heapifica***(***A, fim, maior***)

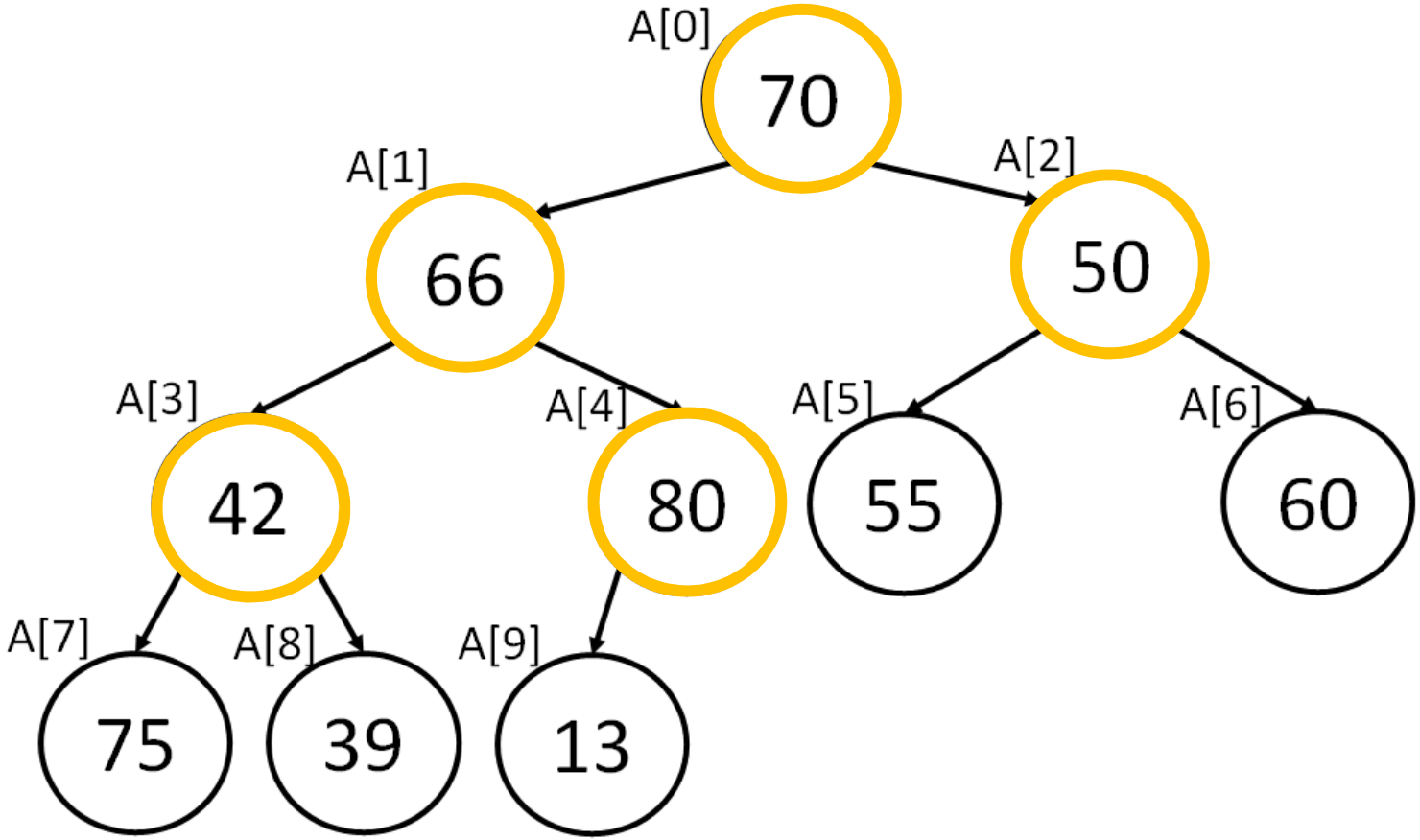
UniCesumar

Primeira Etapa: Formatando o vetor como uma Heap

Heapifica() + ConstroiHeap()

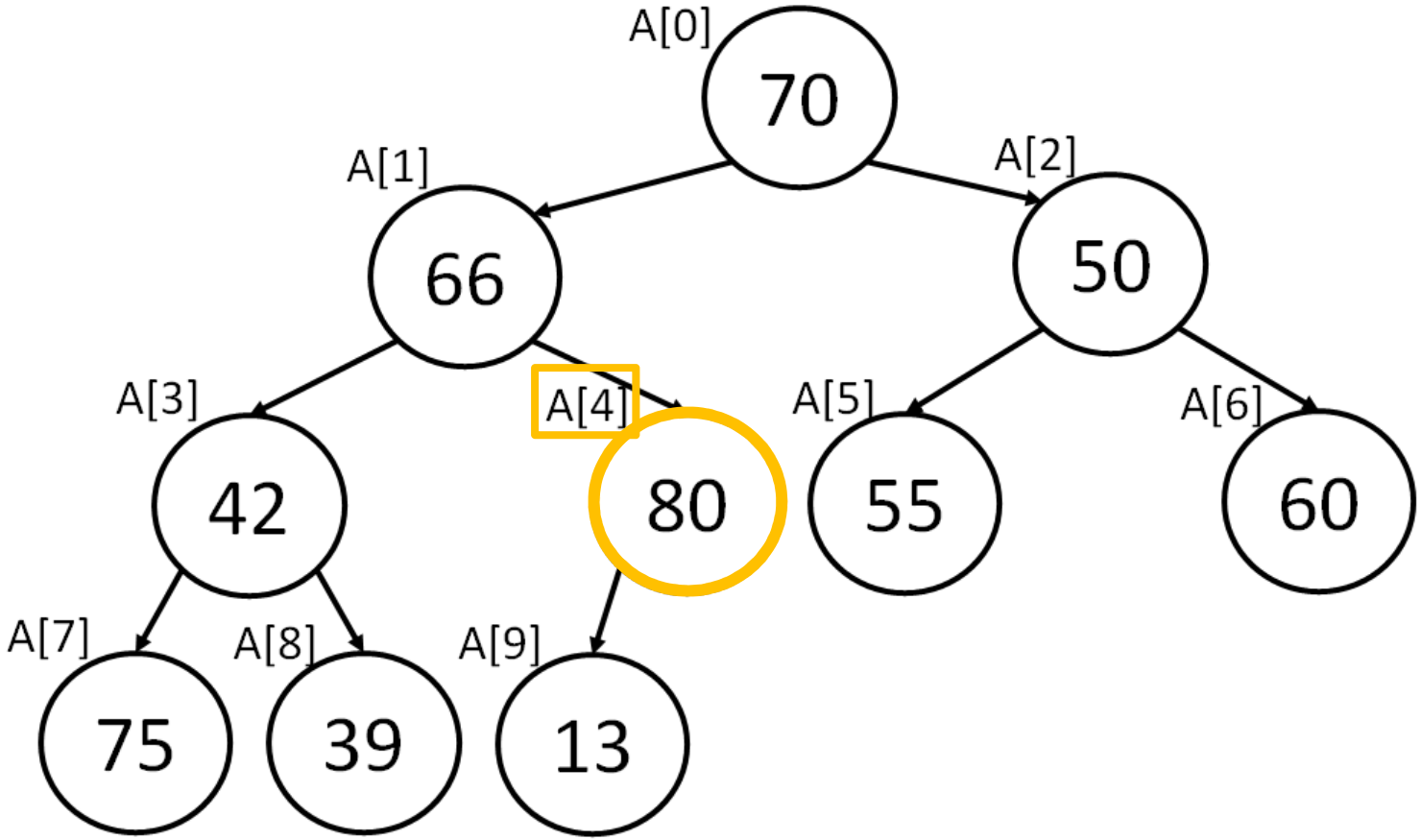
Construindo a Heap: método ConstroiHeap()

i	0	1	2	3	4	5	6	7	8	9
A[i]	70	66	50	42	80	55	60	75	39	13



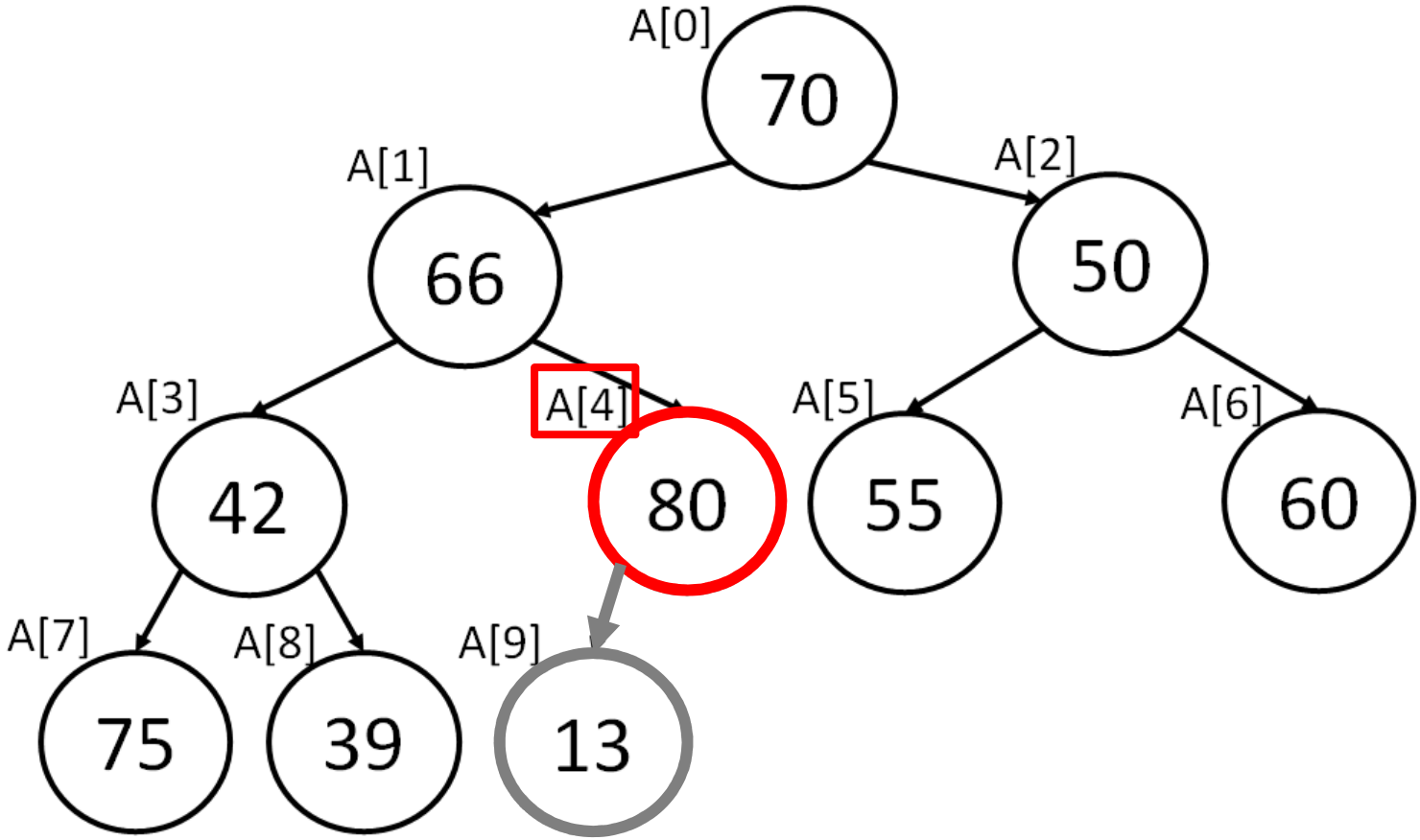
Construindo a Heap: método ConstroiHeap()

i	0	1	2	3	4	5	6	7	8	9
A[i]	70	66	50	42	80	55	60	75	39	13



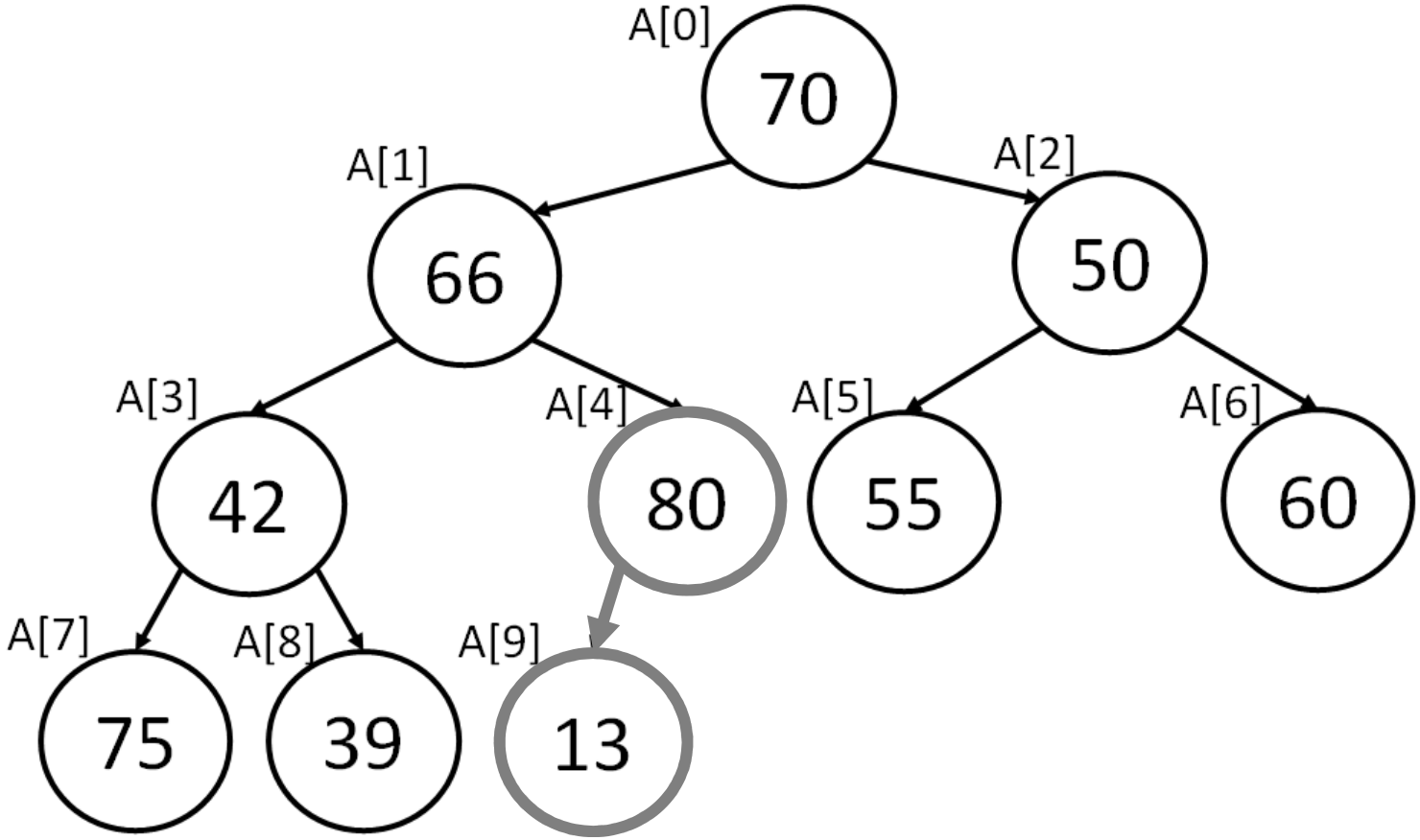
Construindo a Heap: método Heapifica()

i	0	1	2	3	4	5	6	7	8	9
A[i]	70	66	50	42	80	55	60	75	39	13



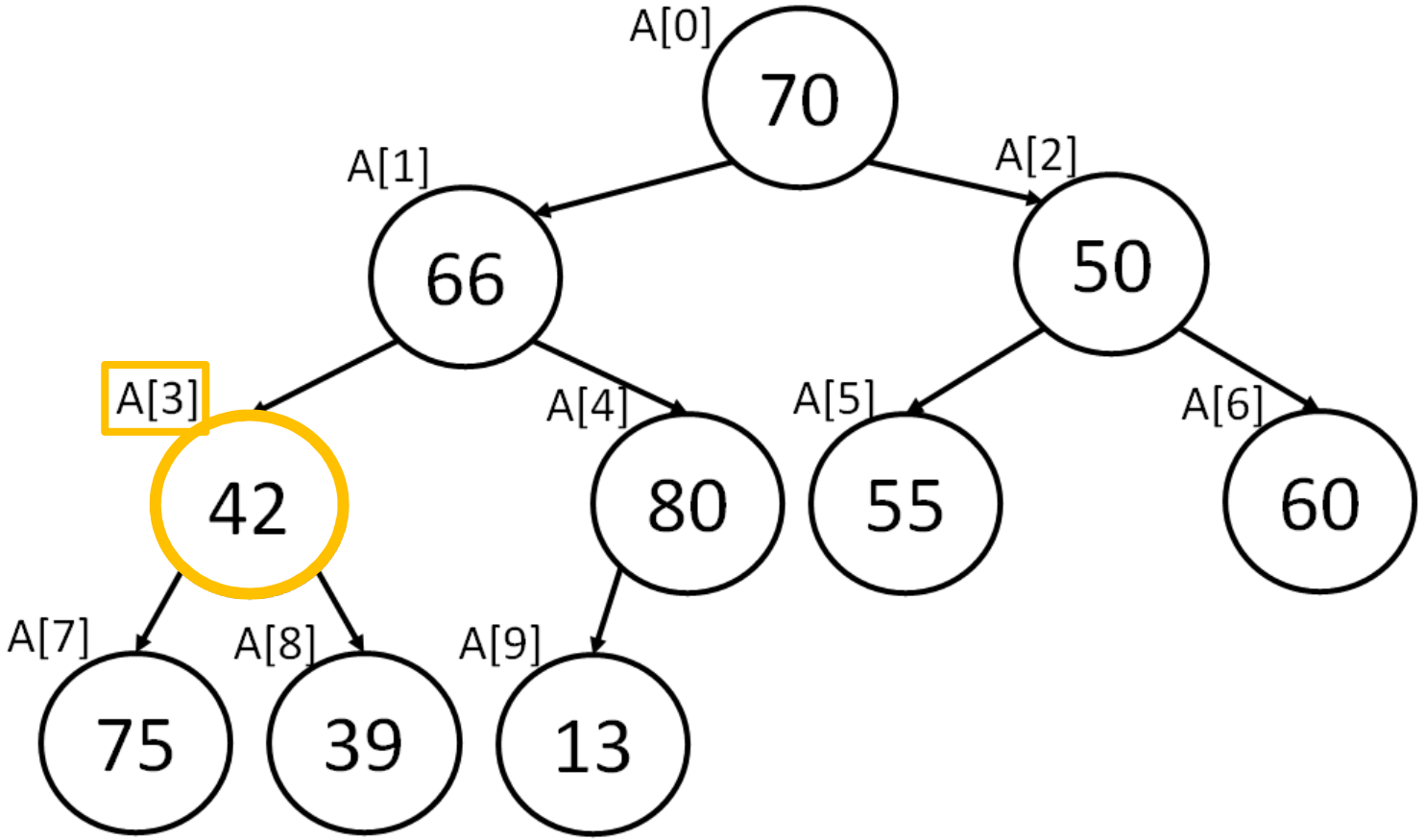
Construindo a Heap: método Heapifica()

i	0	1	2	3	4	5	6	7	8	9
A[i]	70	66	50	42	80	55	60	75	39	13



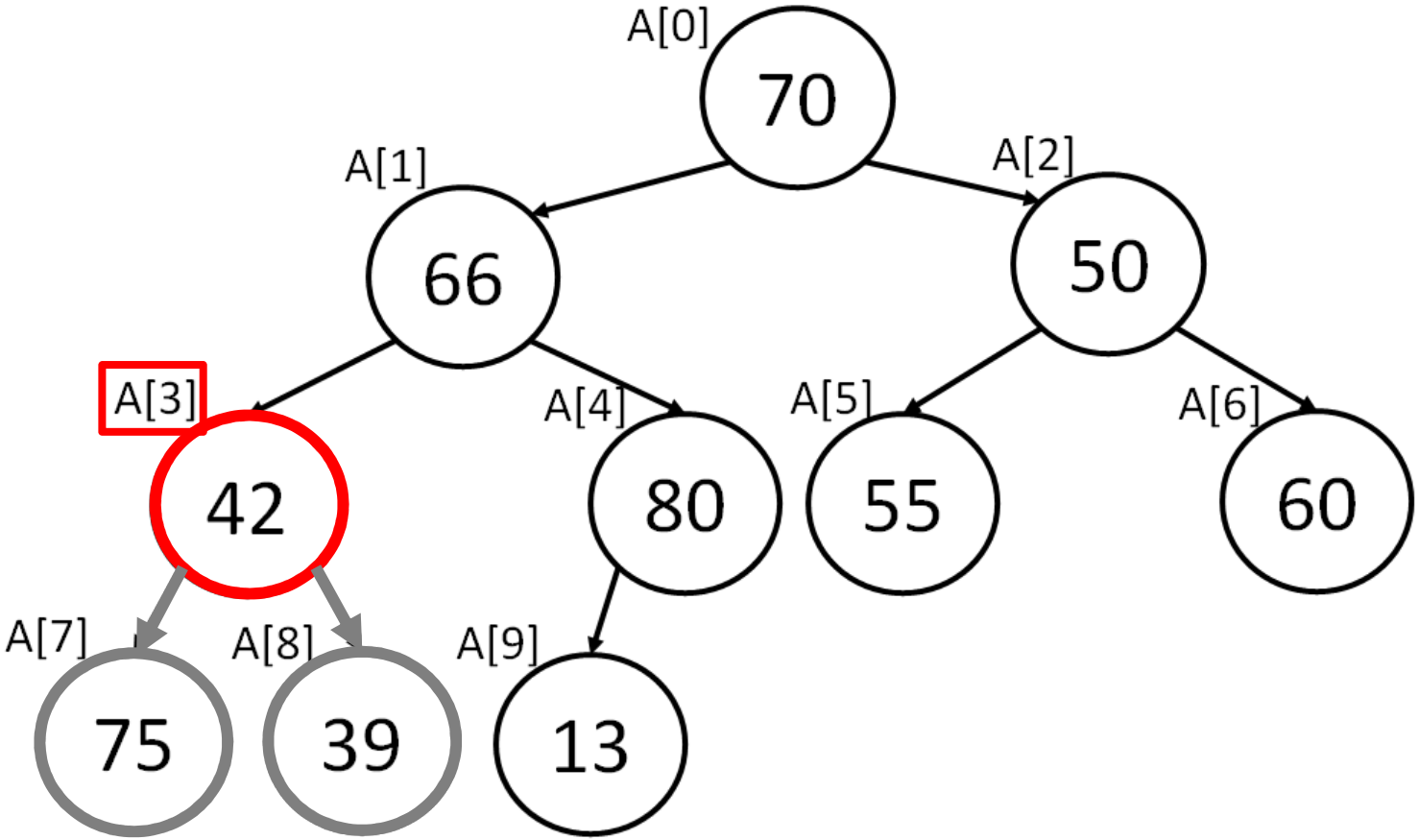
Construindo a Heap: método ConstroiHeap()

i	0	1	2	3	4	5	6	7	8	9
A[i]	70	66	50	42	80	55	60	75	39	13



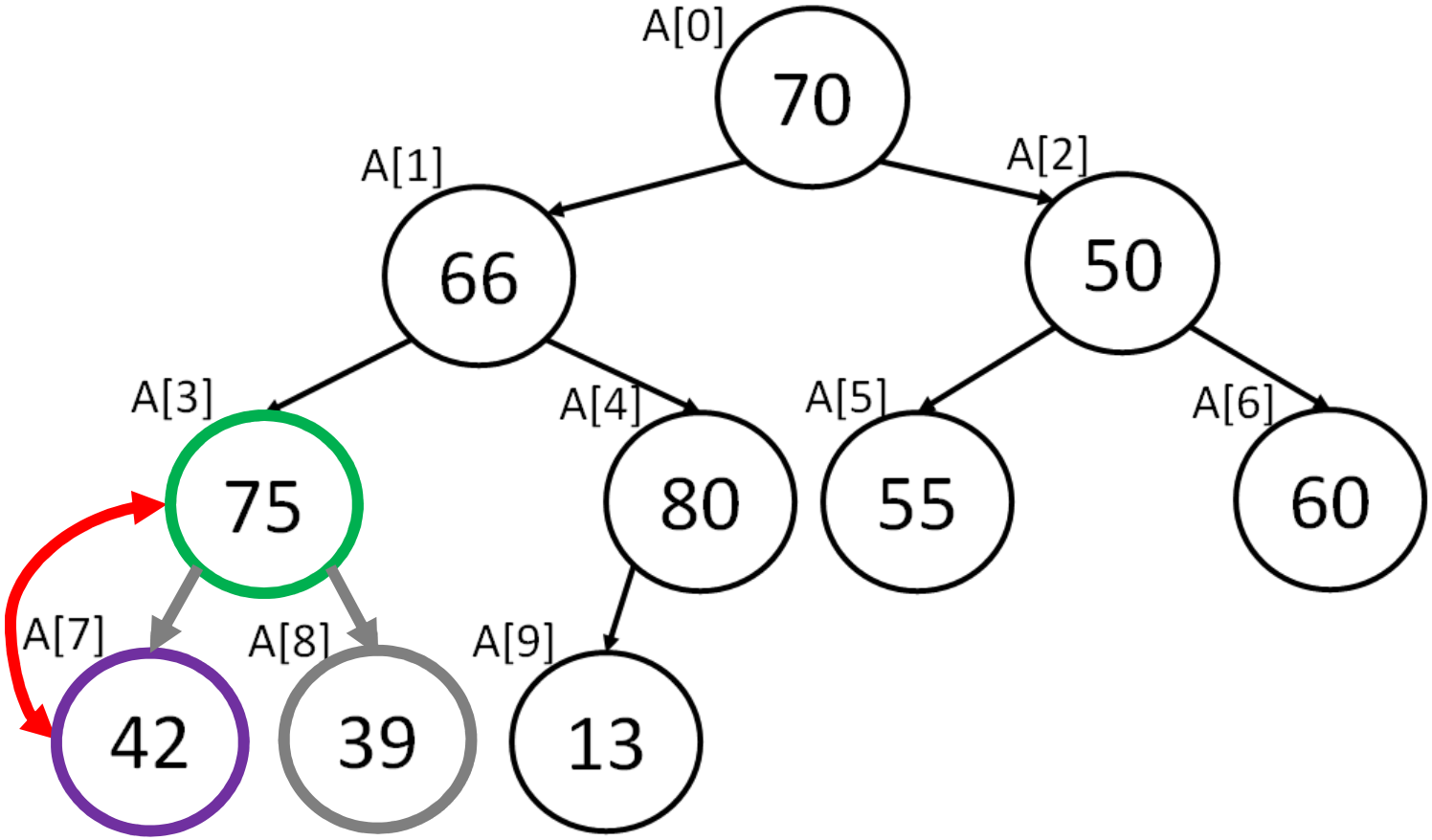
Construindo a Heap: método Heapifica()

i	0	1	2	3	4	5	6	7	8	9
A[i]	70	66	50	42	80	55	60	75	39	13



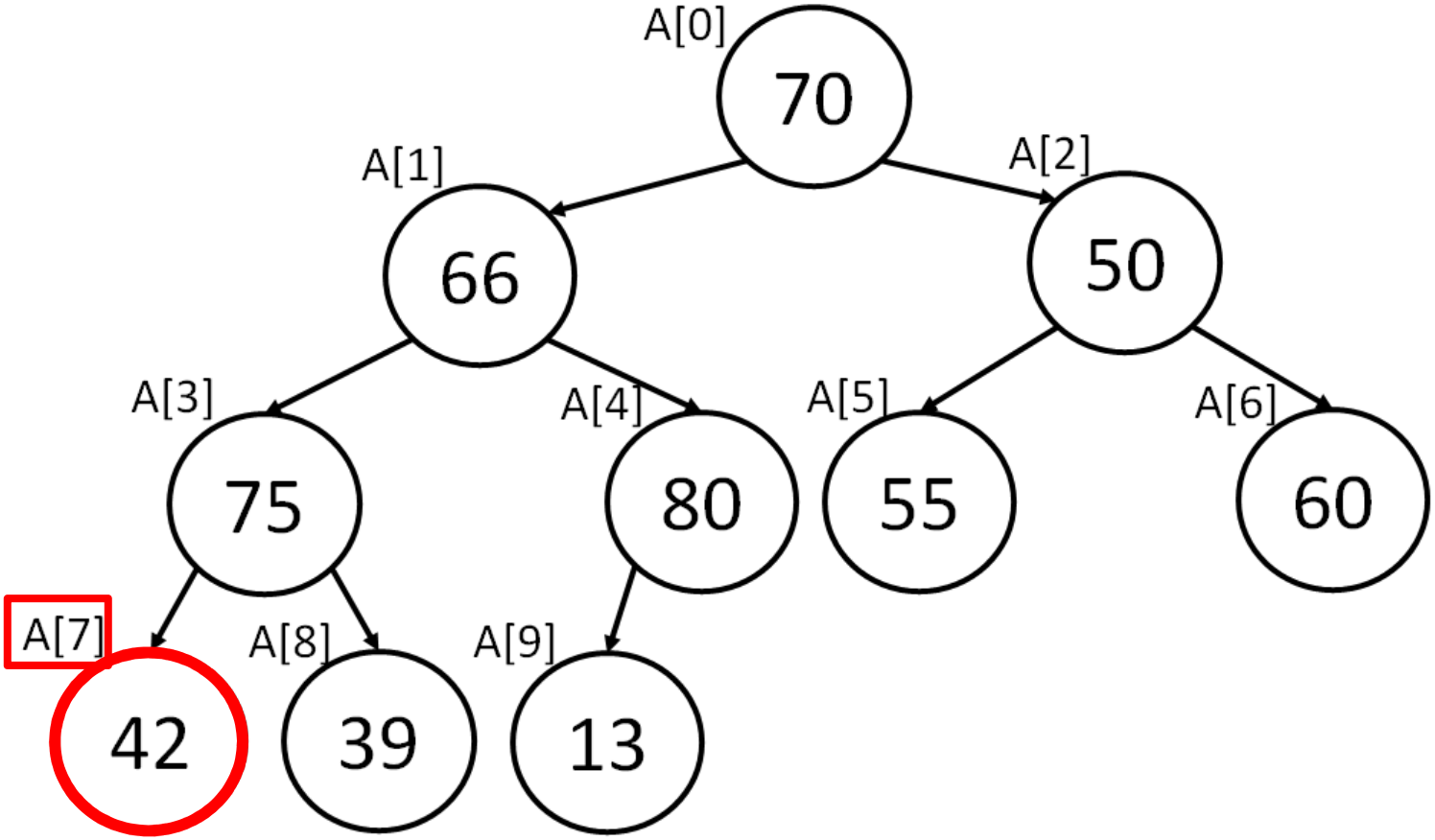
Construindo a Heap: método Heapifica()

i	0	1	2	3	4	5	6	7	8	9
A[i]	70	66	50	75	80	55	60	42	39	13



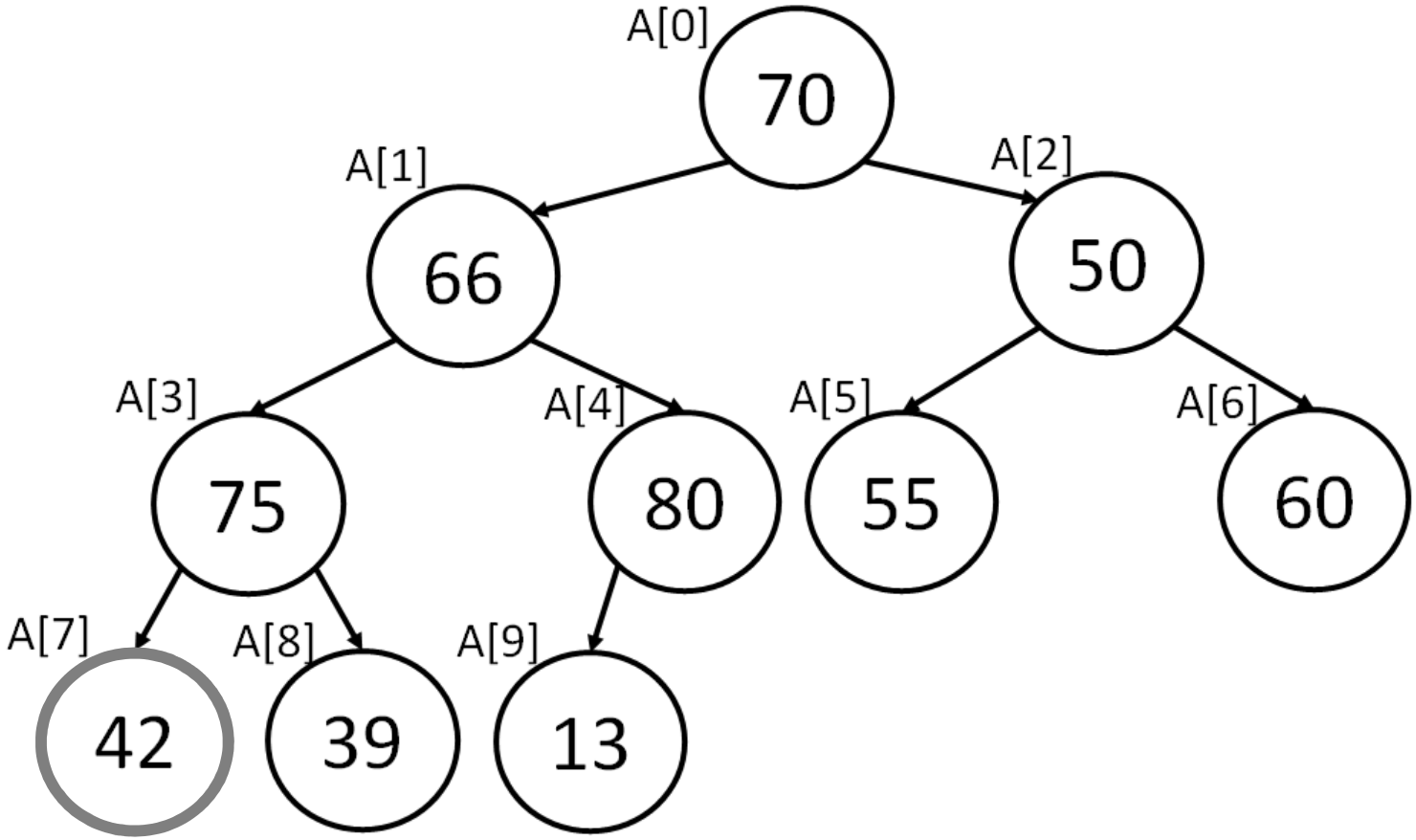
Construindo a Heap: método Heapifica()

i	0	1	2	3	4	5	6	7	8	9
A[i]	70	66	50	75	80	55	60	42	39	13



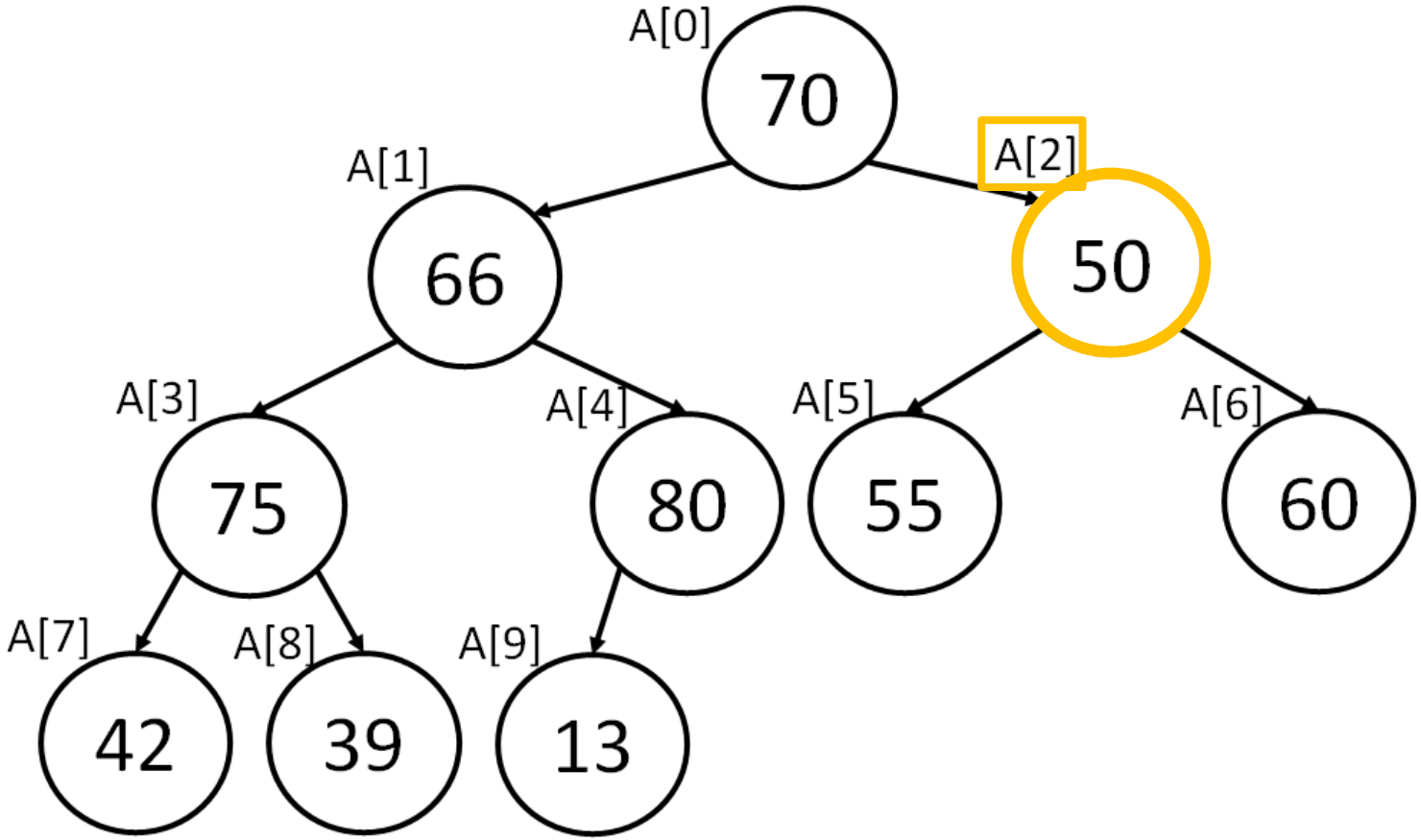
Construindo a Heap: método Heapifica()

i	0	1	2	3	4	5	6	7	8	9
A[i]	70	66	50	75	80	55	60	42	39	13



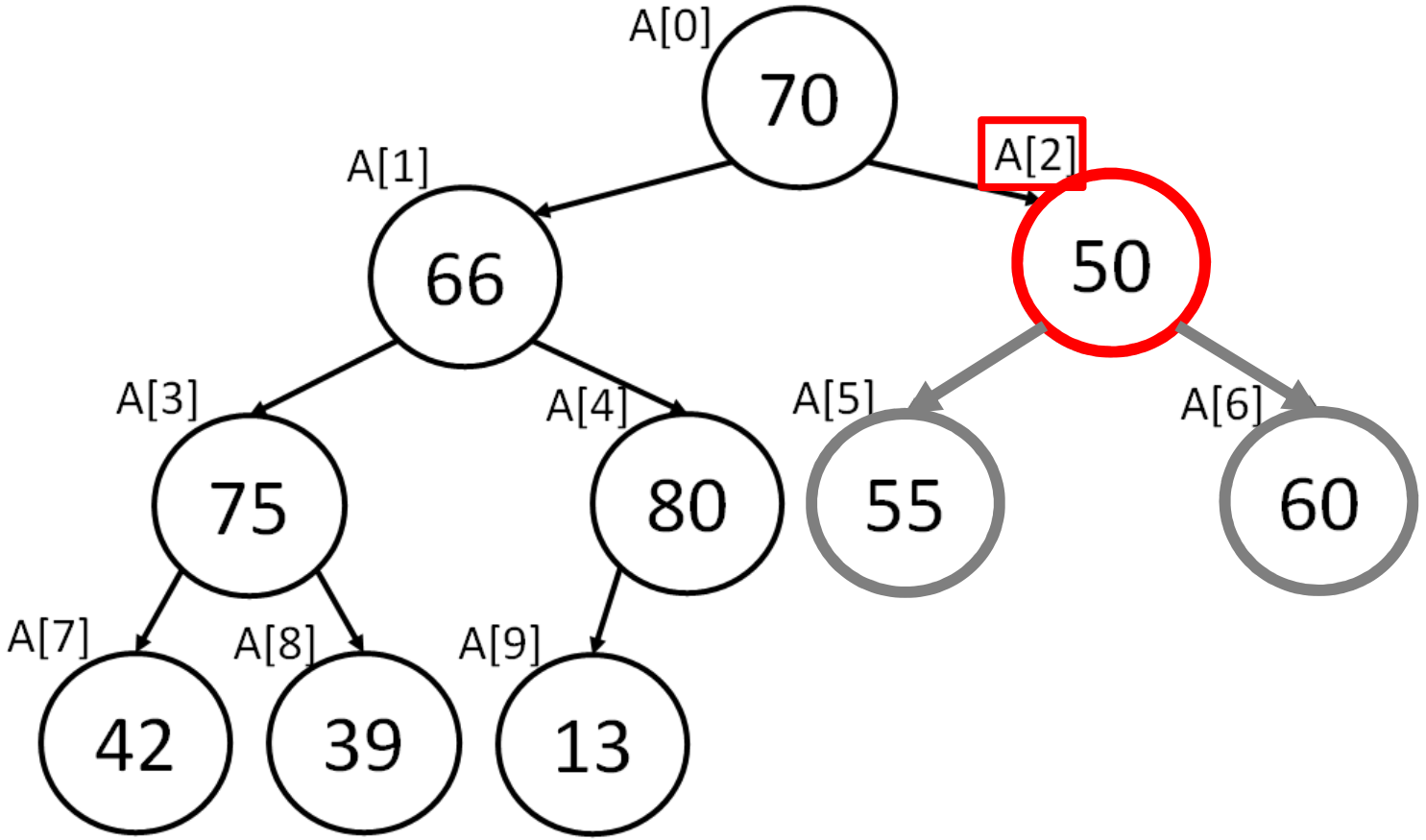
Construindo a Heap: método ConstroiHeap()

i	0	1	2	3	4	5	6	7	8	9
A[i]	70	66	50	75	80	55	60	42	39	13



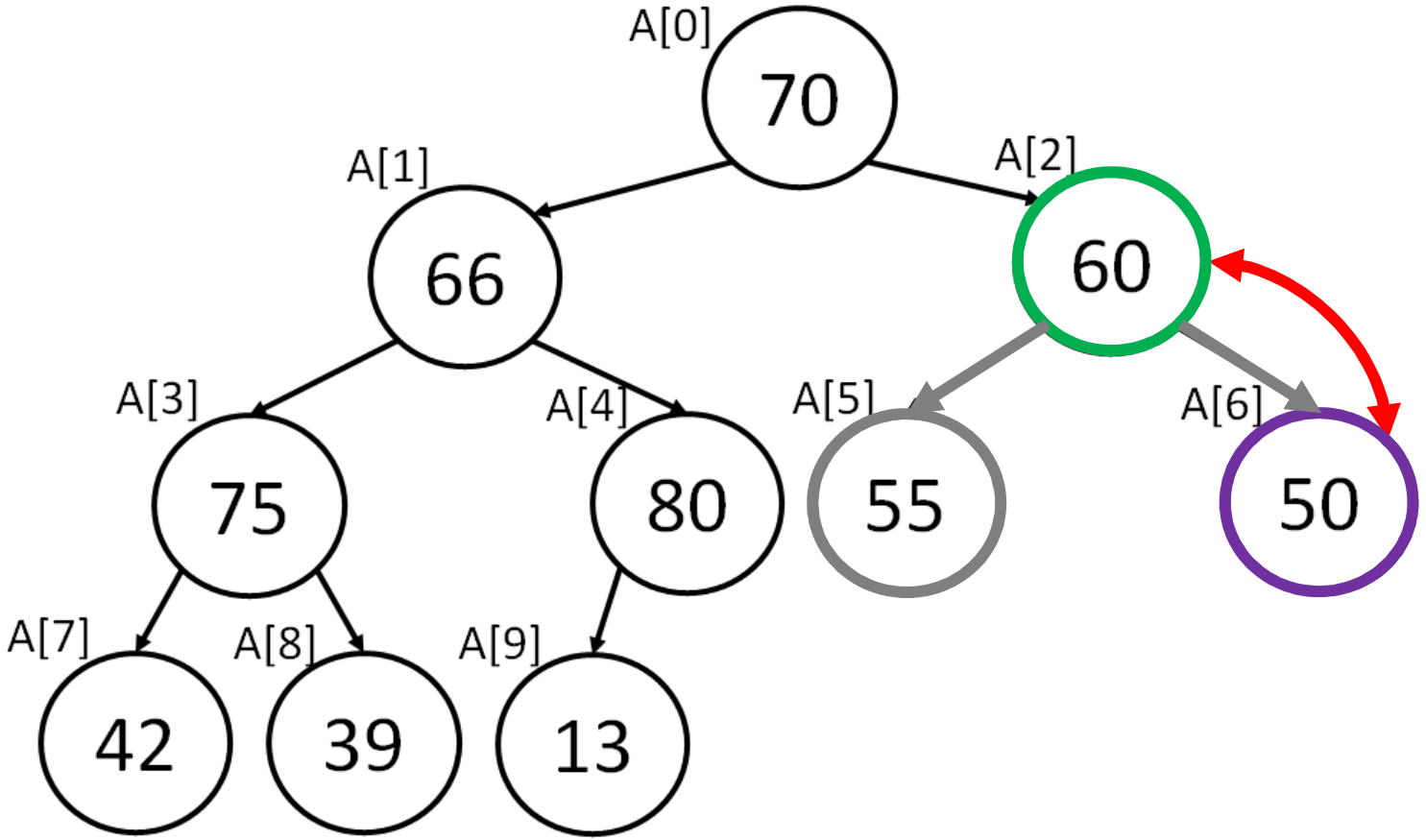
Construindo a Heap: método Heapifica()

i	0	1	2	3	4	5	6	7	8	9
A[i]	70	66	50	75	80	55	60	42	39	13



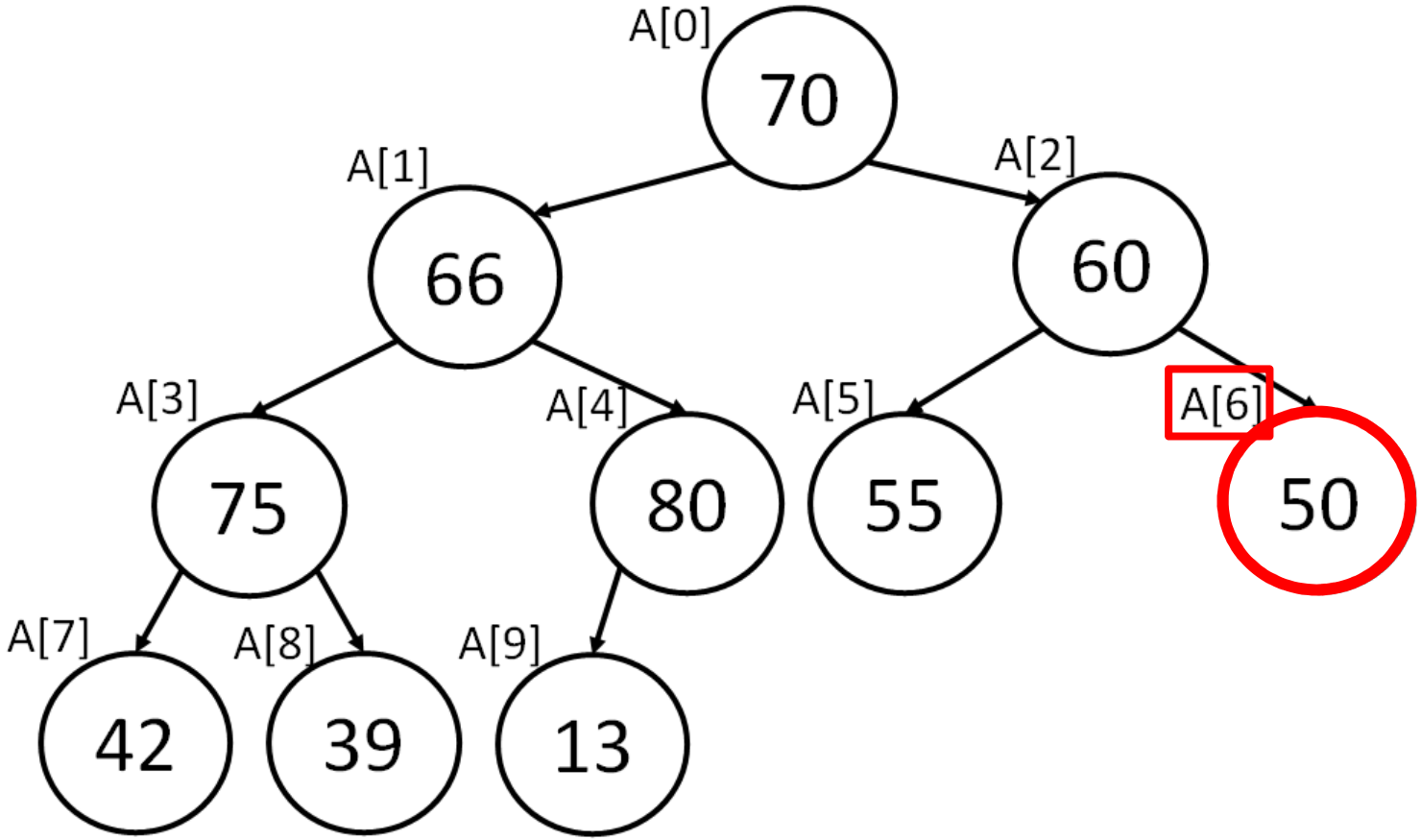
Construindo a Heap: método Heapifica()

i	0	1	2	3	4	5	6	7	8	9
A[i]	70	66	60	75	80	55	50	42	39	13



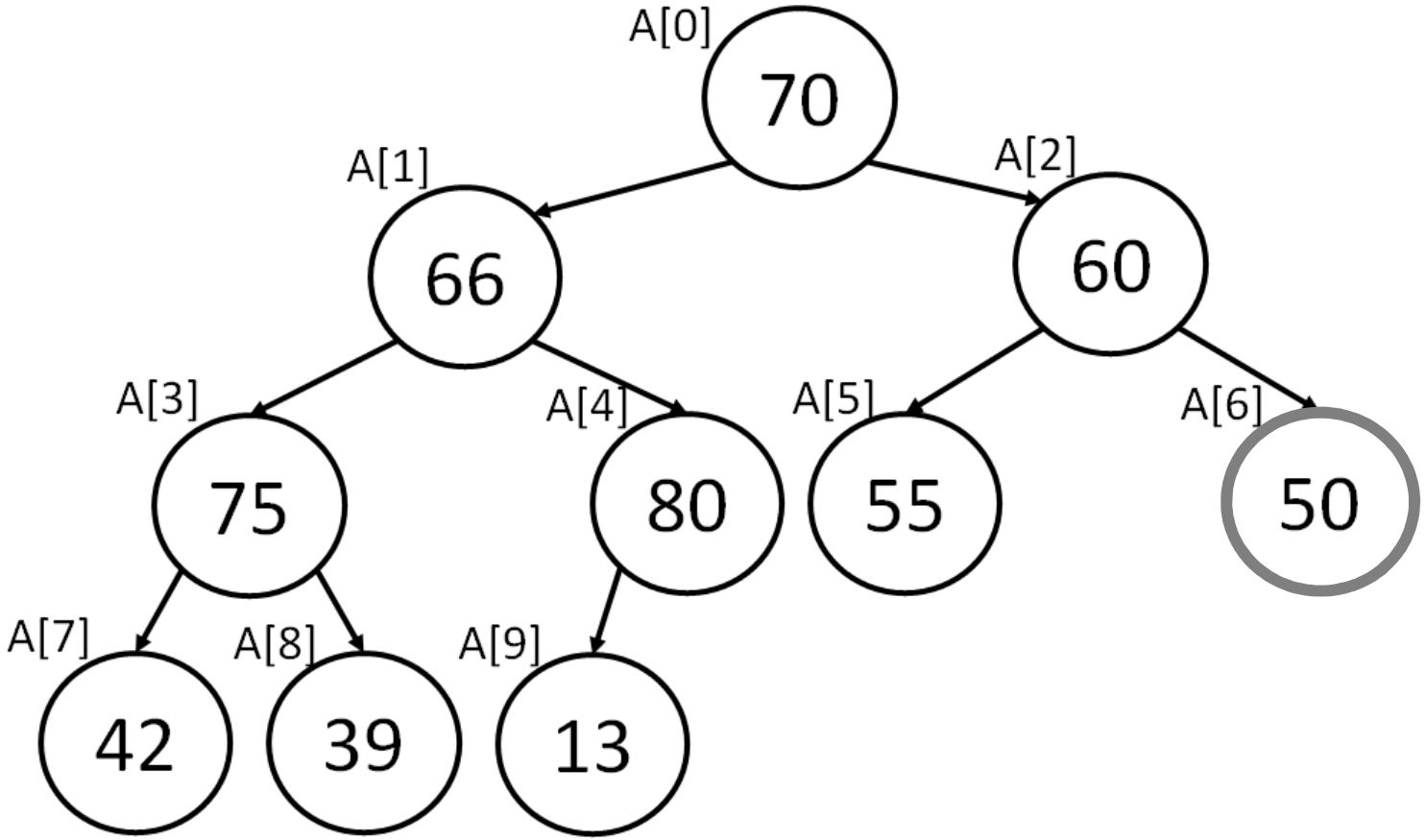
Construindo a Heap: método Heapifica()

i	0	1	2	3	4	5	6	7	8	9
A[i]	70	66	60	75	80	55	50	42	39	13



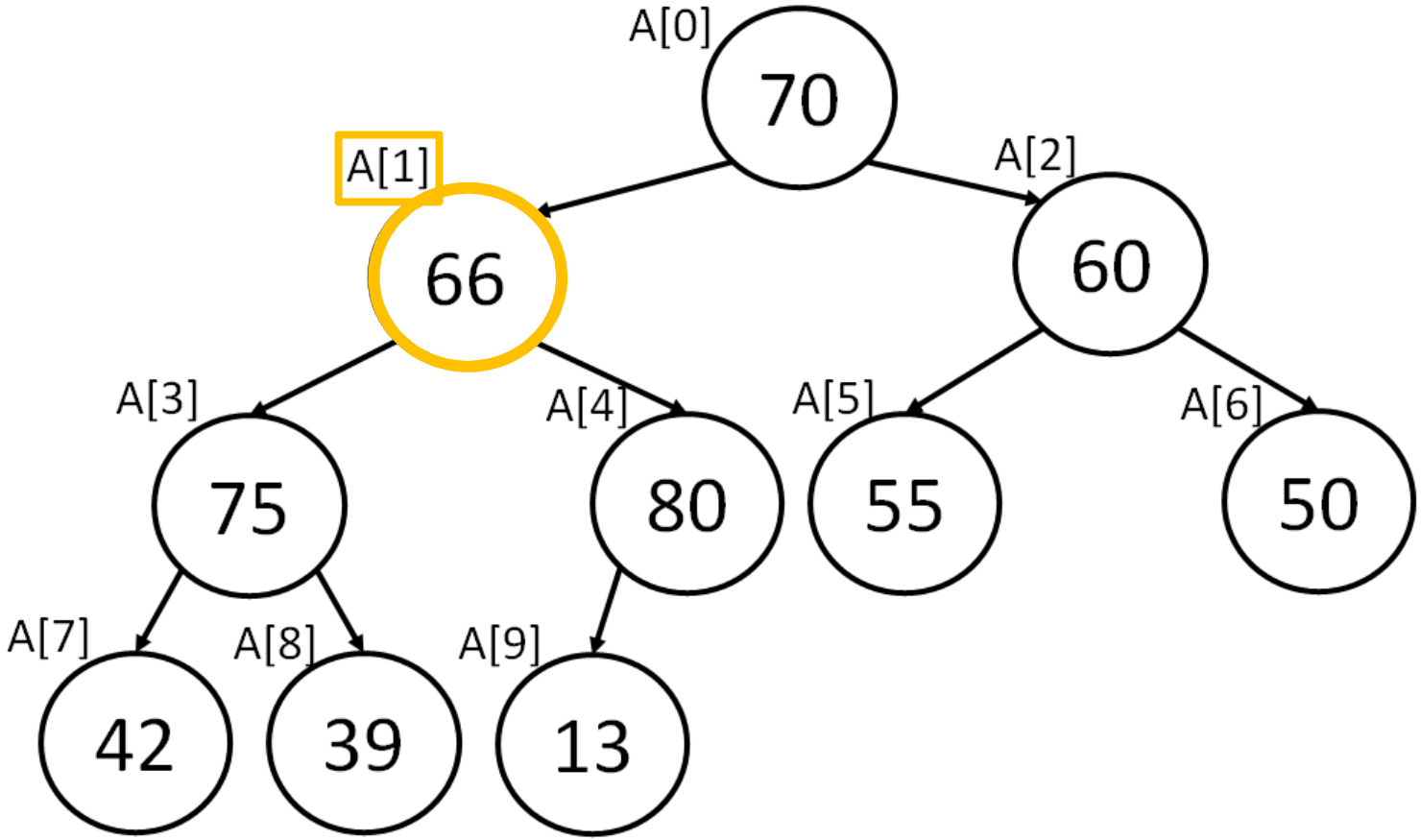
Construindo a Heap: método Heapifica()

i	0	1	2	3	4	5	6	7	8	9
A[i]	70	66	60	75	80	55	50	42	39	13



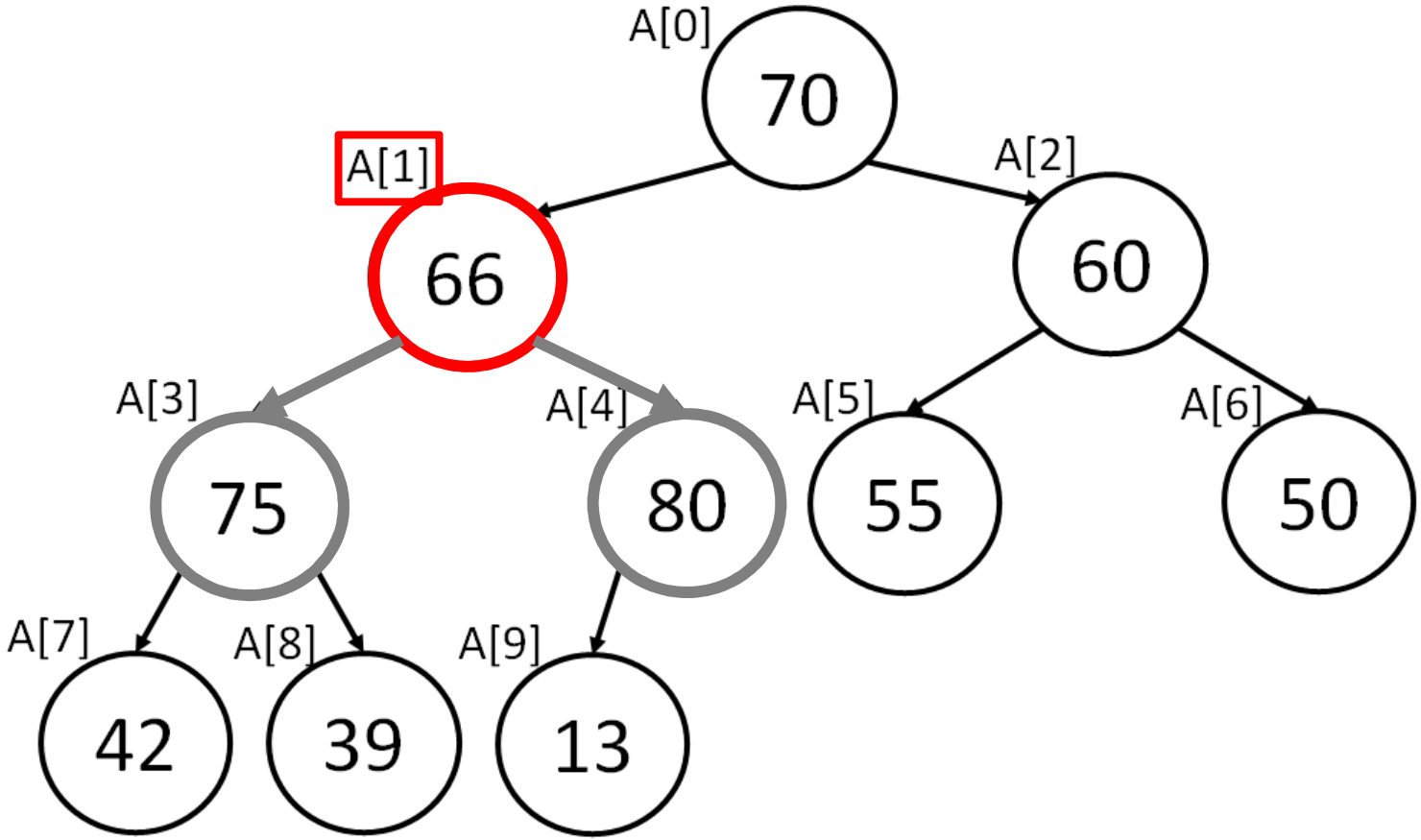
Construindo a Heap: método ConstroiHeap()

i	0	1	2	3	4	5	6	7	8	9
A[i]	70	66	60	75	80	55	50	42	39	13



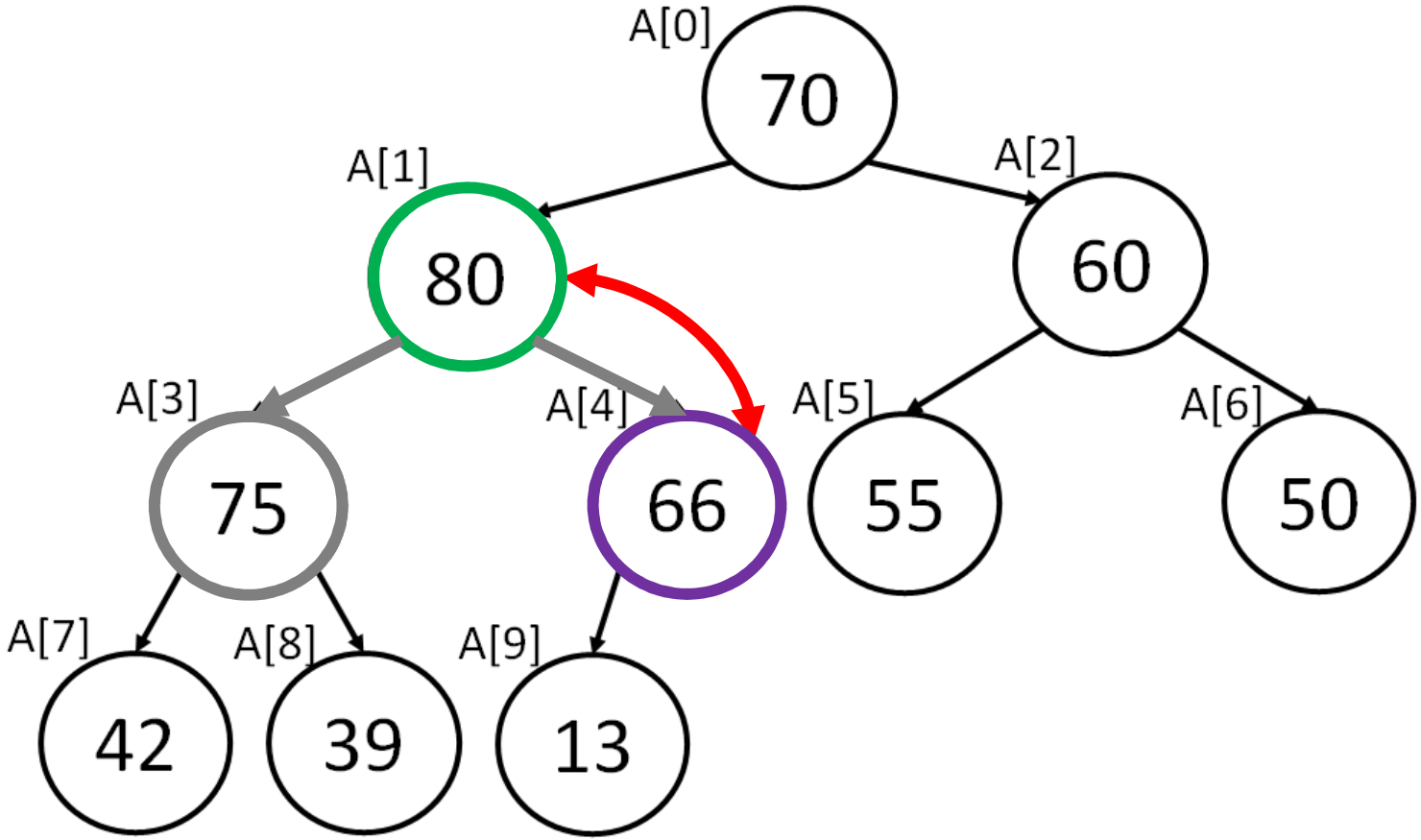
Construindo a Heap: método Heapifica()

i	0	1	2	3	4	5	6	7	8	9
A[i]	70	66	60	75	80	55	50	42	39	13



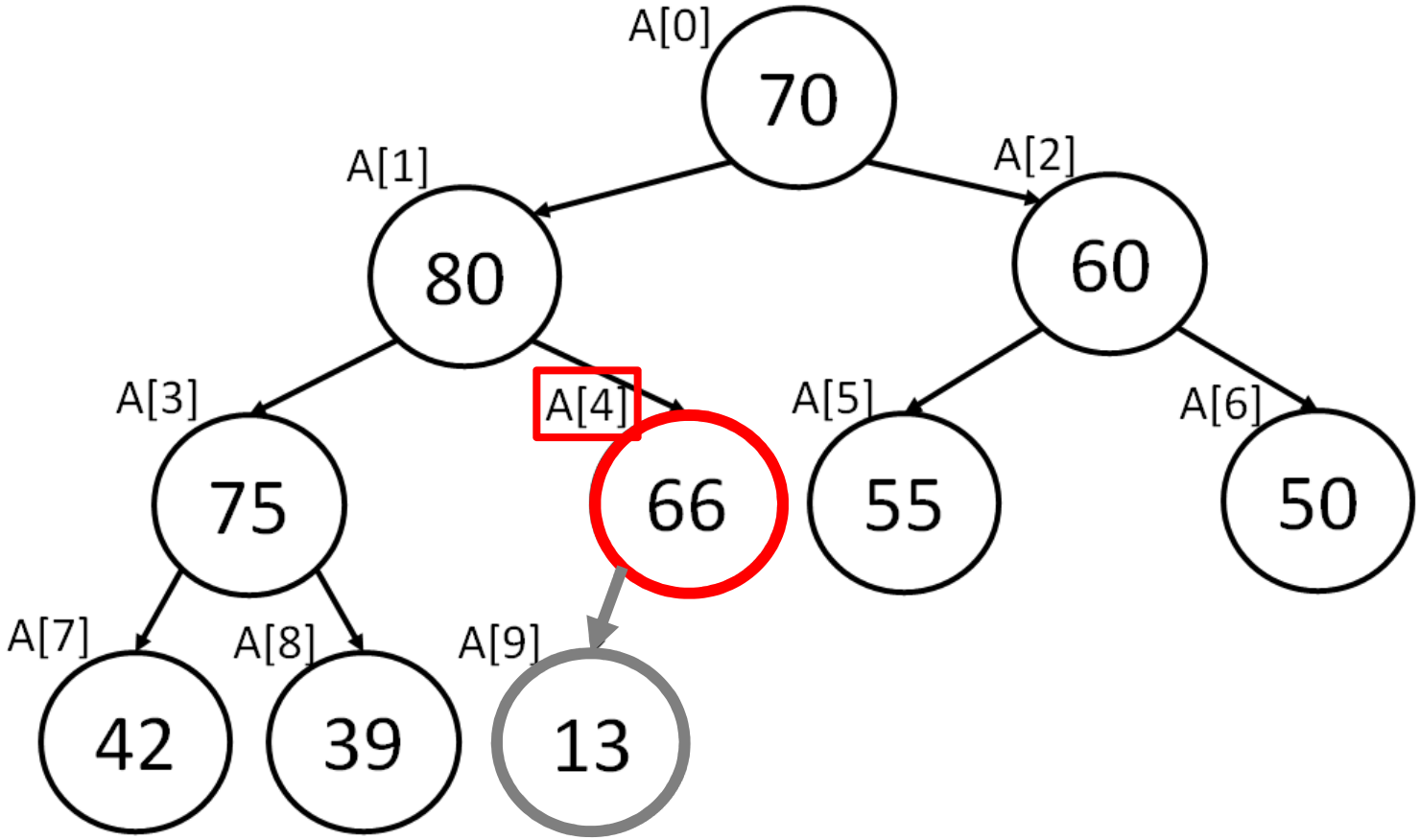
Construindo a Heap: método Heapifica()

i	0	1	2	3	4	5	6	7	8	9
A[i]	70	80	60	75	66	55	50	42	39	13



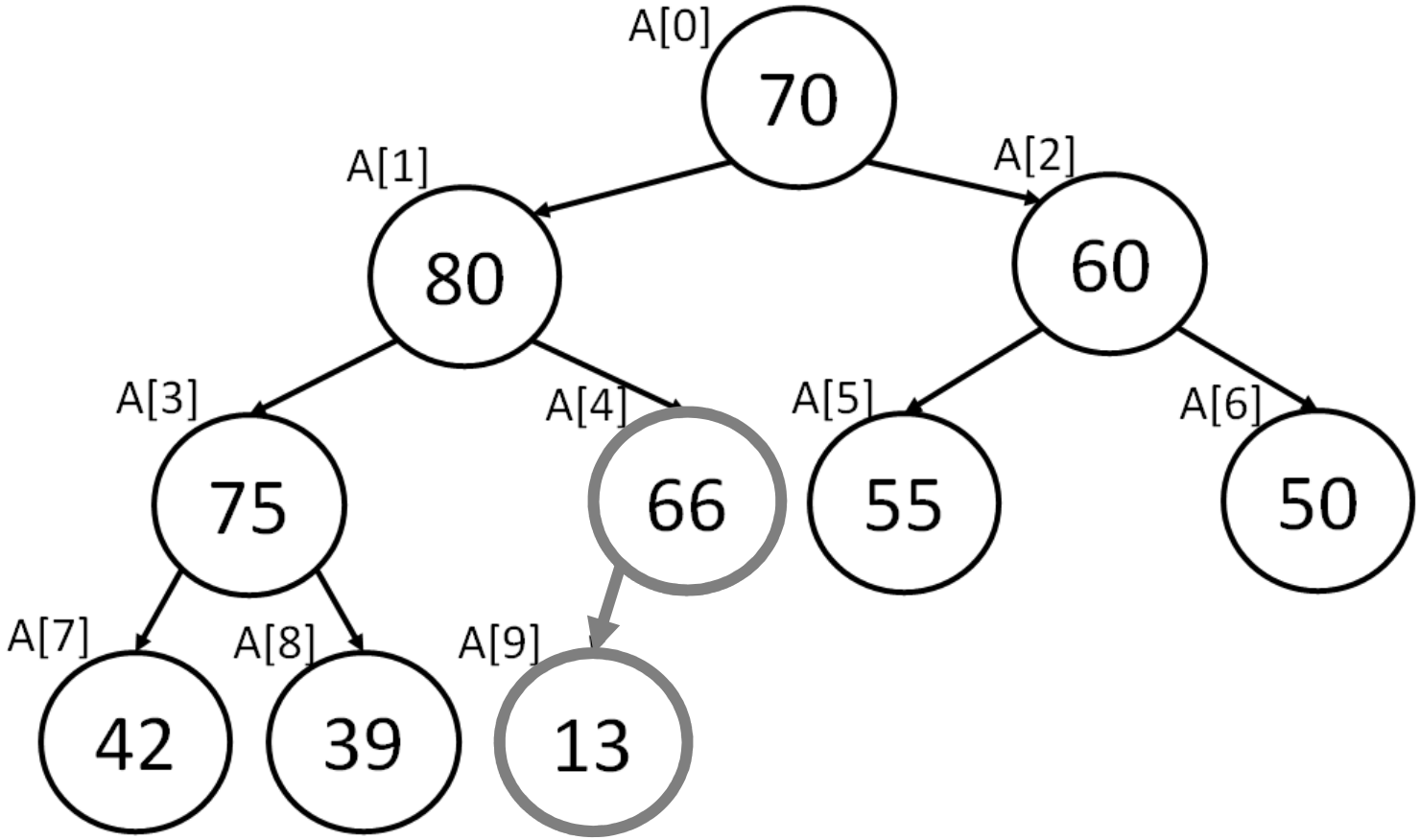
Construindo a Heap: método Heapifica()

i	0	1	2	3	4	5	6	7	8	9
A[i]	70	80	60	75	66	55	50	42	39	13



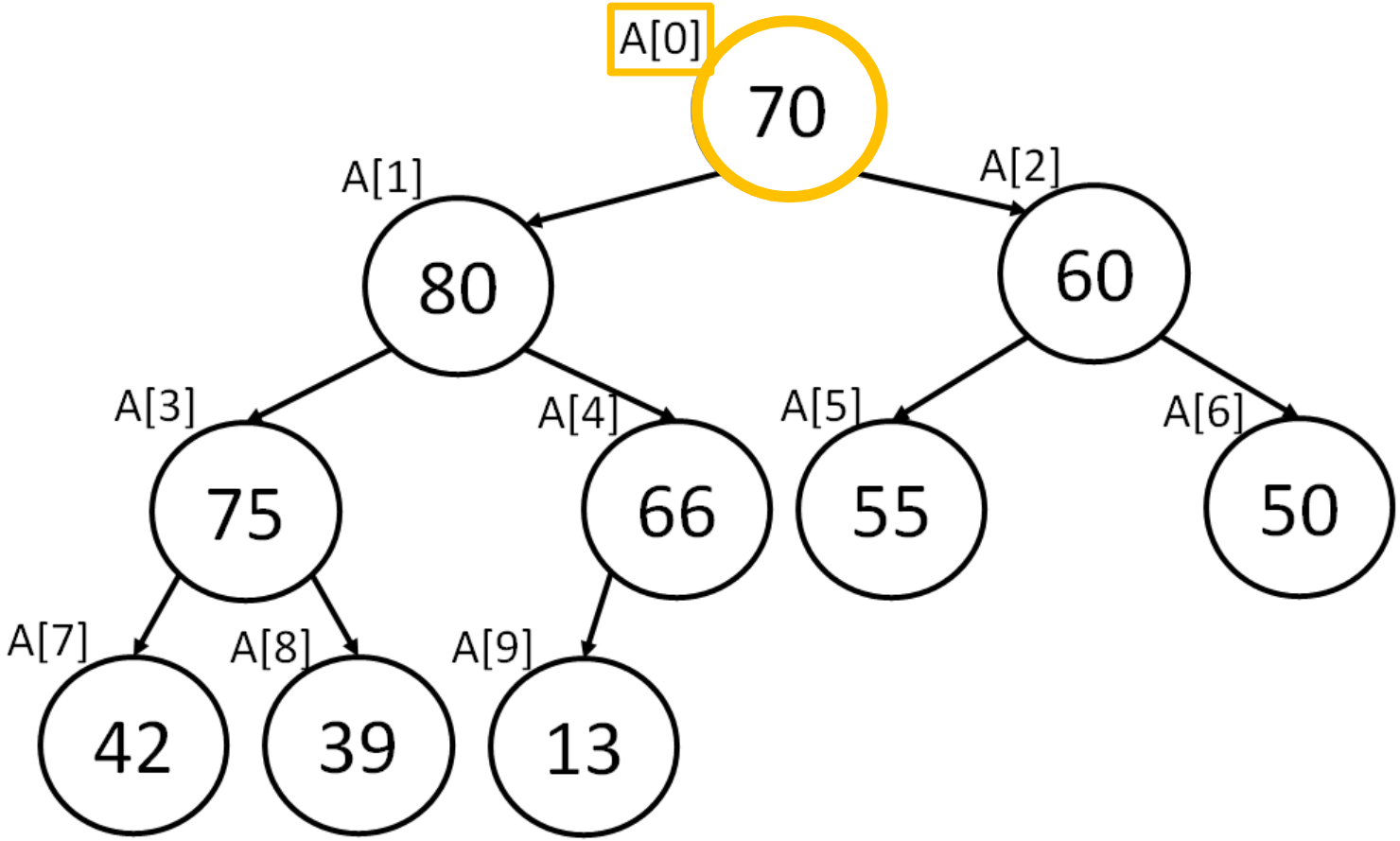
Construindo a Heap: método Heapifica()

i	0	1	2	3	4	5	6	7	8	9
A[i]	70	80	60	75	66	55	50	42	39	13



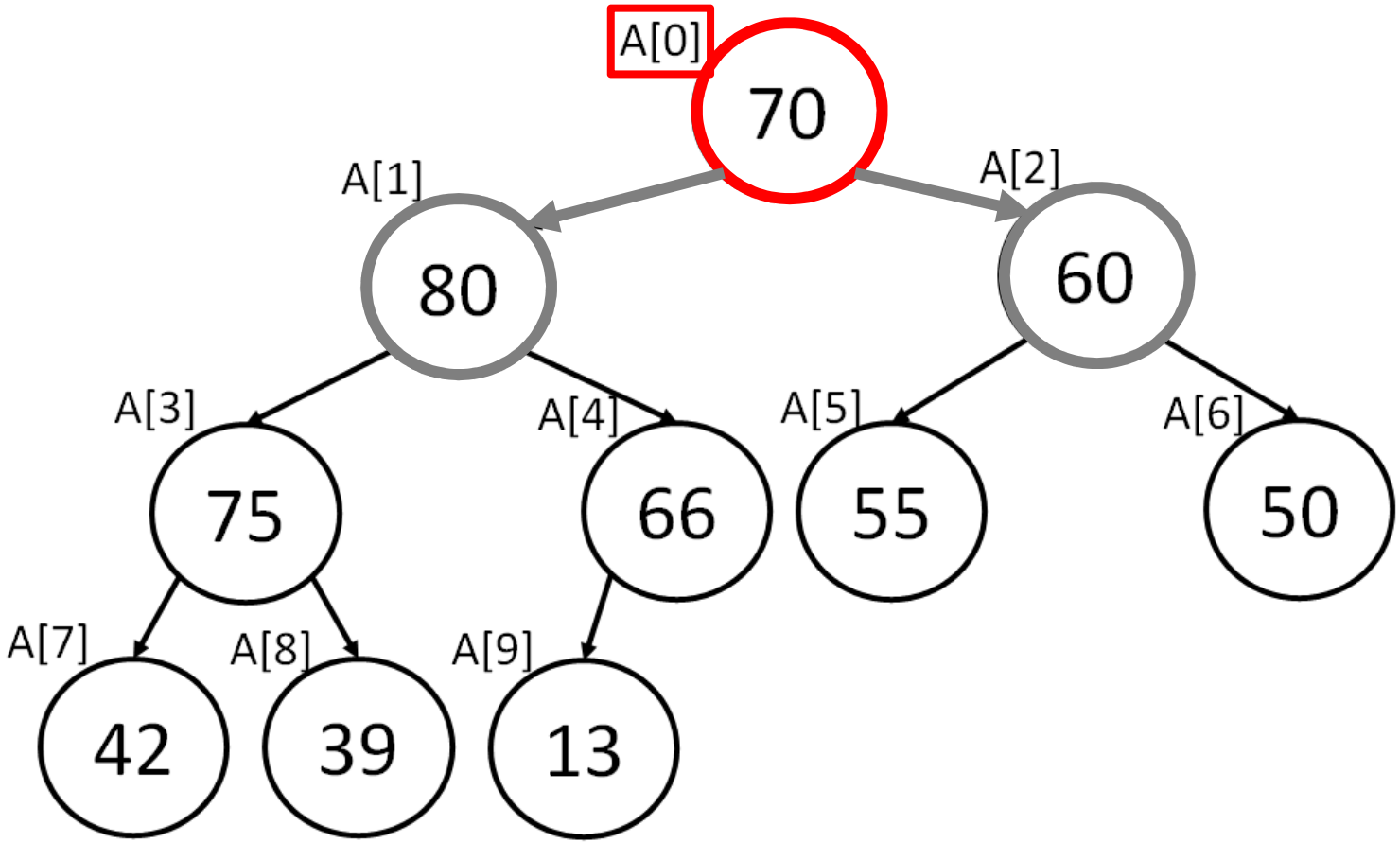
Construindo a Heap: método ConstroiHeap()

i	0	1	2	3	4	5	6	7	8	9
A[i]	70	80	60	75	66	55	50	42	39	13



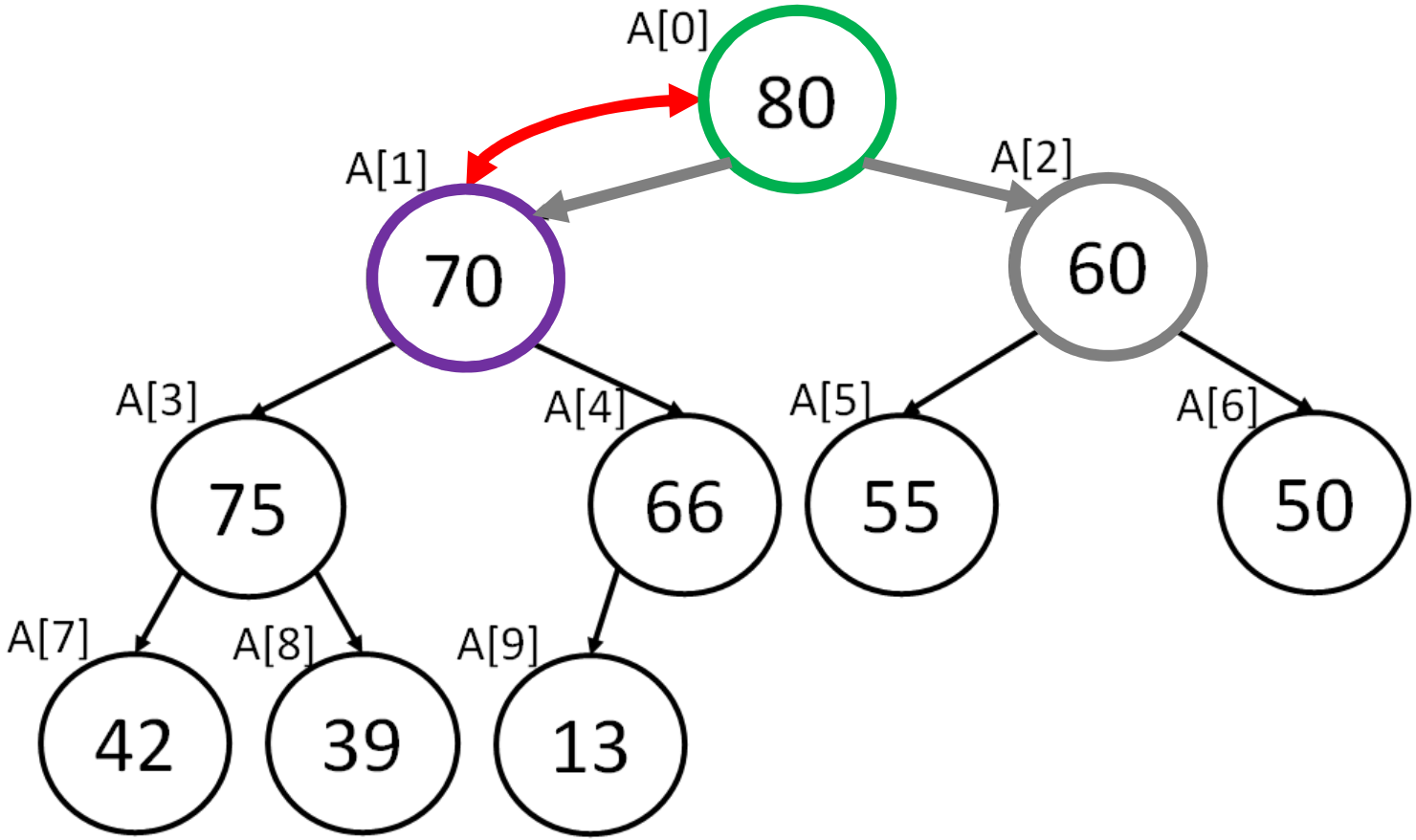
Construindo a Heap: método Heapifica()

i	0	1	2	3	4	5	6	7	8	9
A[i]	70	80	60	75	66	55	50	42	39	13



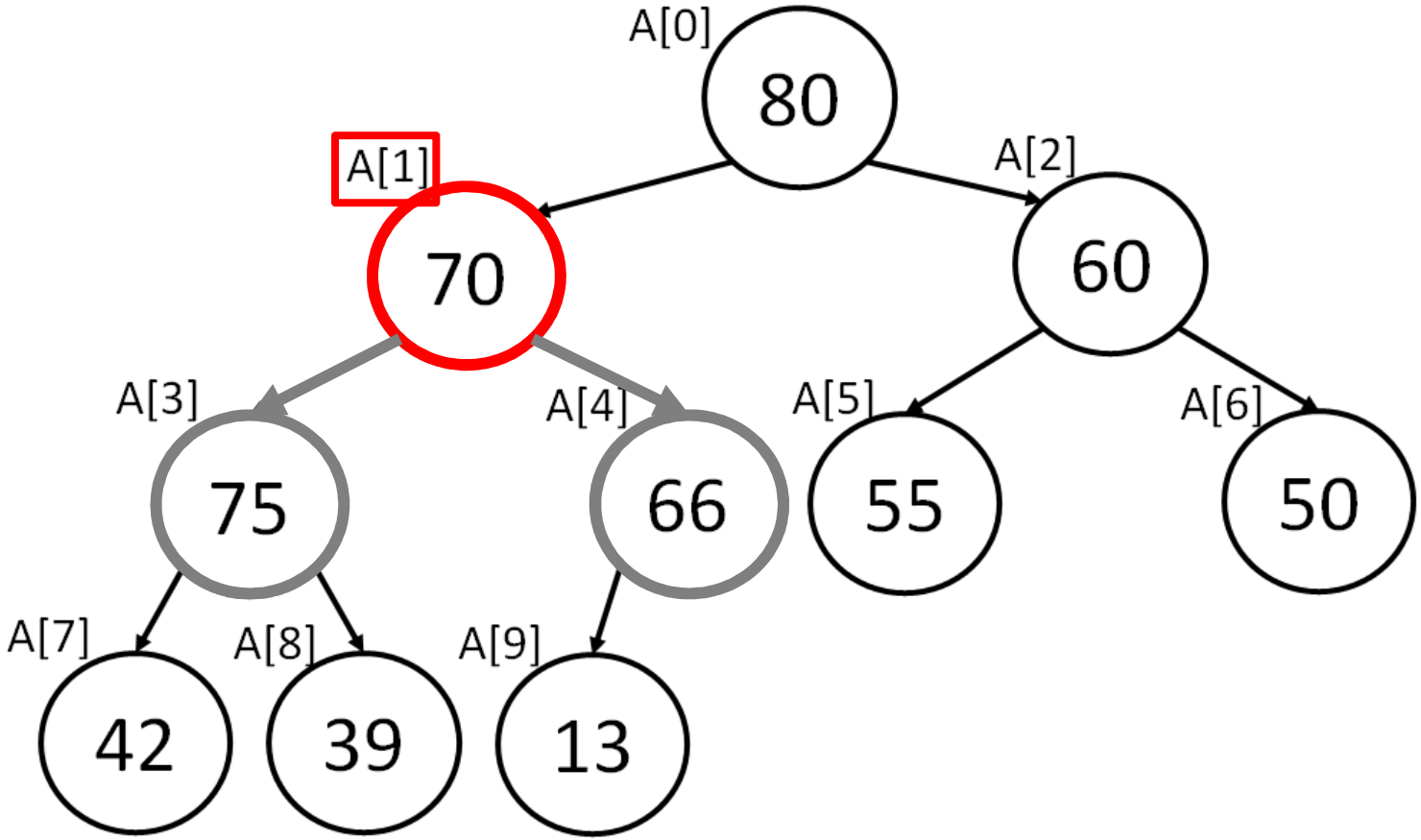
Construindo a Heap: método Heapifica()

i	0	1	2	3	4	5	6	7	8	9
A[i]	80	70	60	75	66	55	50	42	39	13



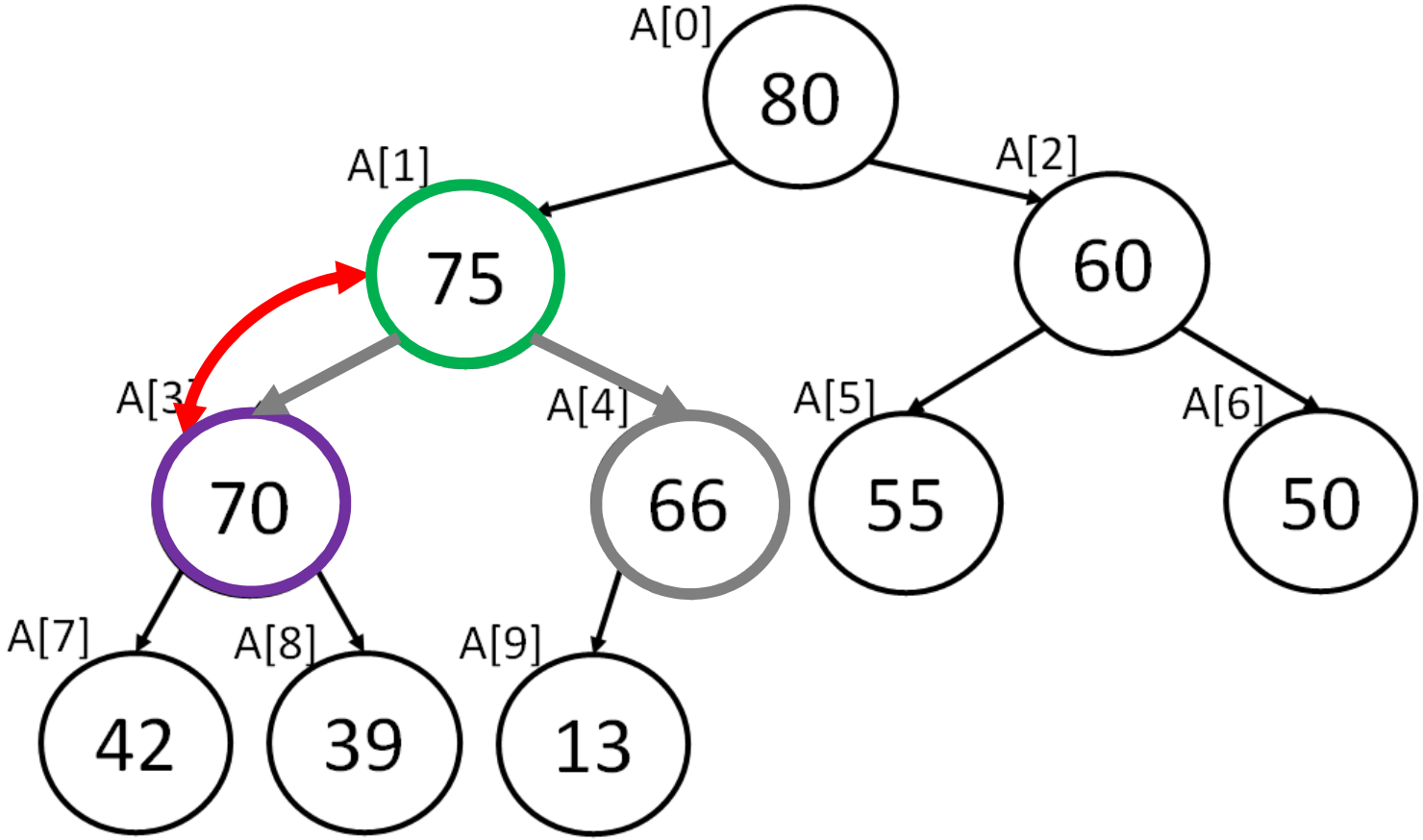
Construindo a Heap: método Heapifica()

i	0	1	2	3	4	5	6	7	8	9
A[i]	80	70	60	75	66	55	50	42	39	13



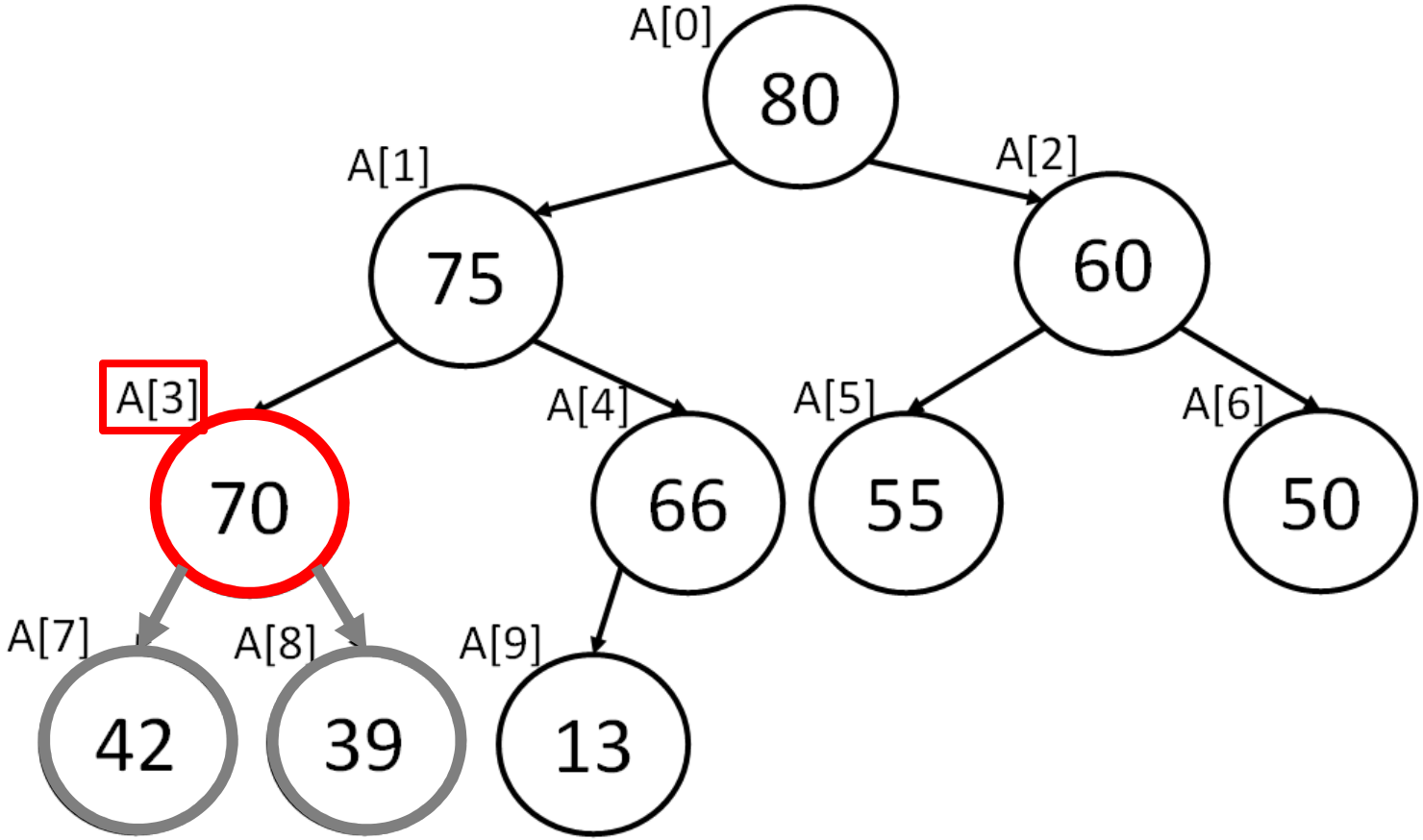
Construindo a Heap: método Heapifica()

i	0	1	2	3	4	5	6	7	8	9
A[i]	80	75	60	70	66	55	50	42	39	13



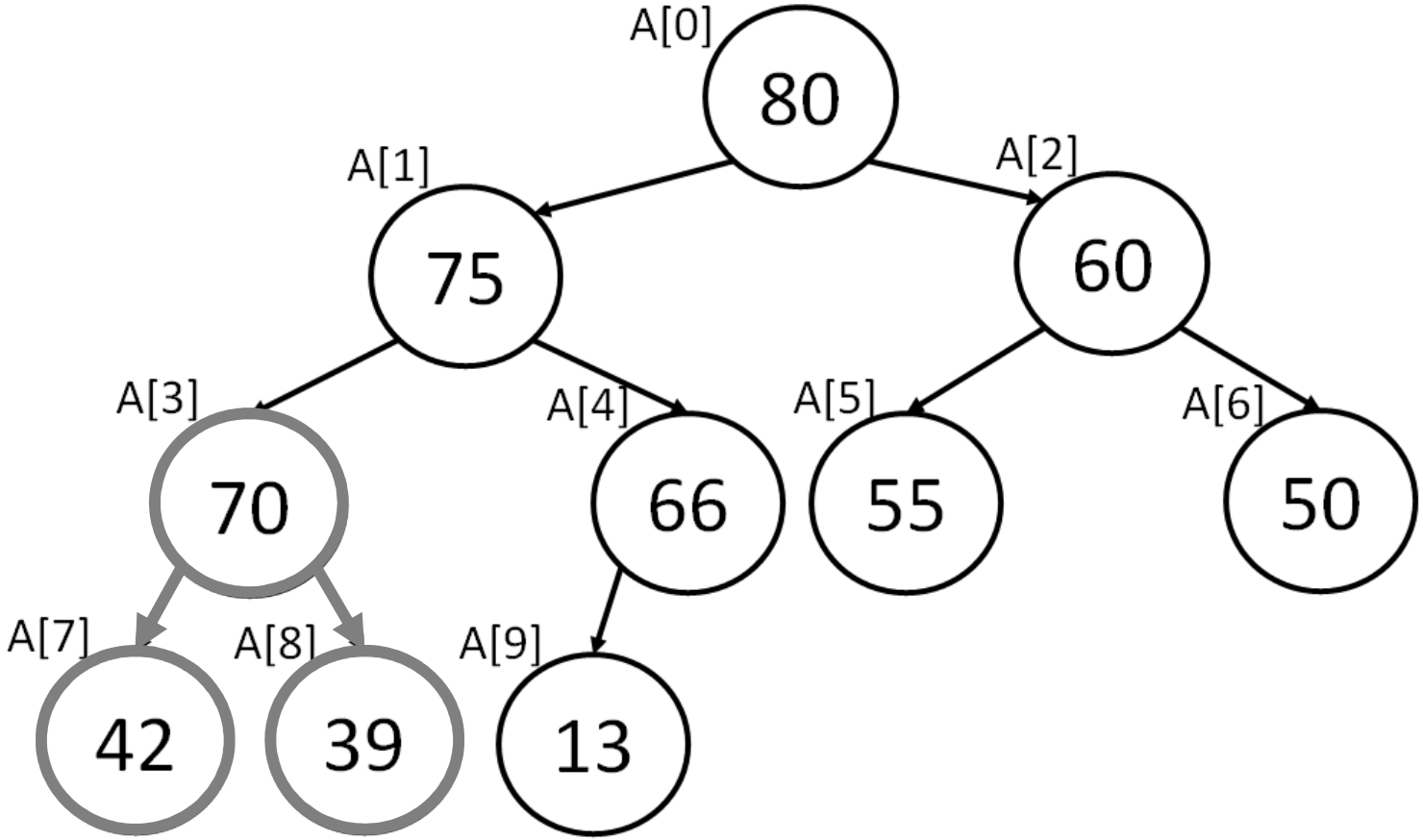
Construindo a Heap: método Heapifica()

i	0	1	2	3	4	5	6	7	8	9
A[i]	80	75	60	70	66	55	50	42	39	13



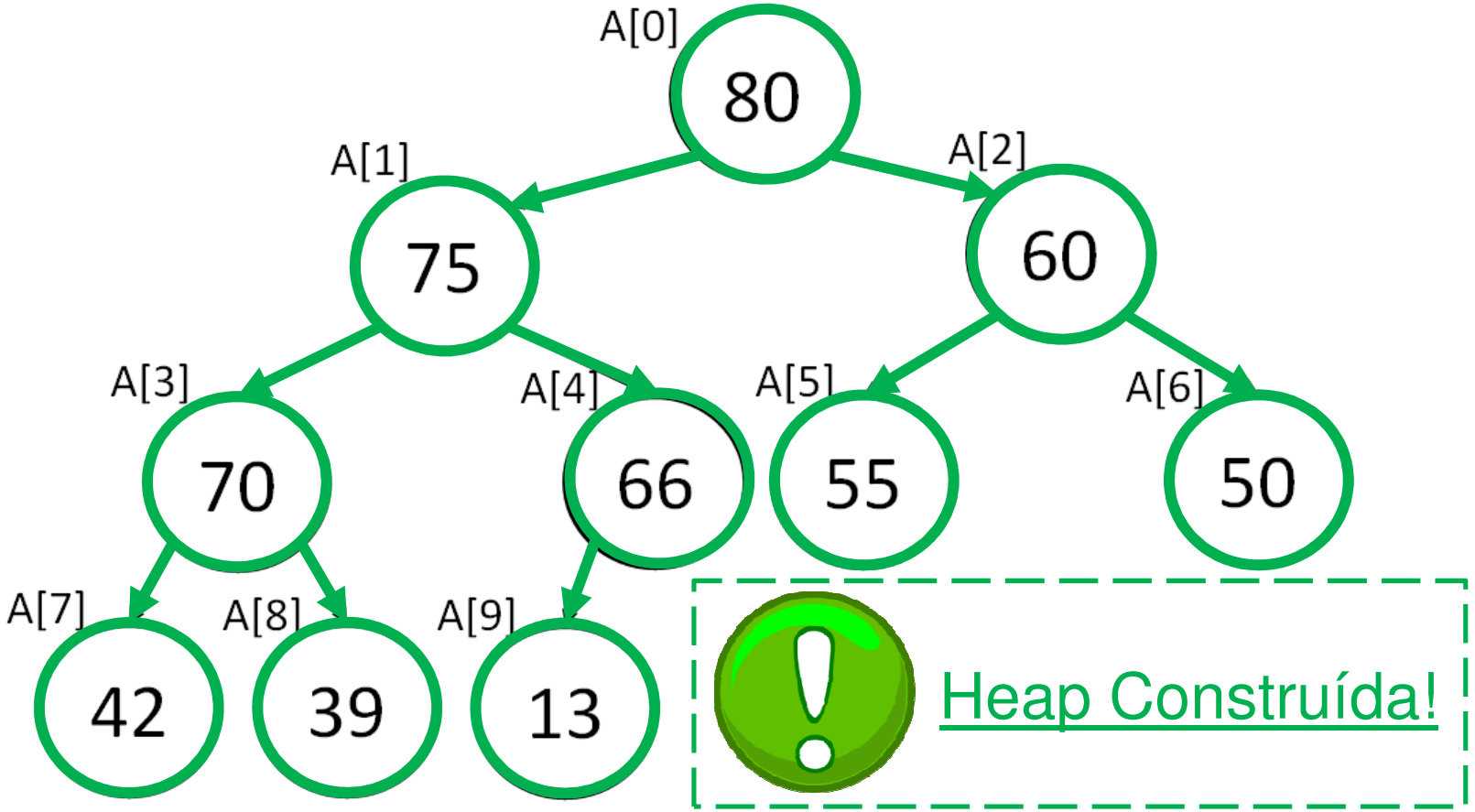
Construindo a Heap: método Heapifica()

i	0	1	2	3	4	5	6	7	8	9
A[i]	80	75	60	70	66	55	50	42	39	13



Primeiro Passo Completo: Heap Construída

i	0	1	2	3	4	5	6	7	8	9
A[i]	80	75	60	70	66	55	50	42	39	13



Heap Construída!

ConstroiHeap() + Heapifica()

Teste de Mesa: repetindo a execução dos algoritmos, de maneira detalhada

Heapsort: Início do algoritmo

Heapsort(arranjo ***A, fim***)

1. ***n*** ← ***fim***
2. ***ConstroiHeap***(***A, fim***)
3. Para ***i*** de ***n*** até ***1*** faça
4. troca ***A[1]*** ↔ ***A[i]***
5. ***n*** ← ***n-1***
6. ***Heapifica***(***A, n, 0***)

Heapsort: Construindo a Heap

Heapsort(arranjo ***A, fim***)

1. ***n*** ← ***fim***
2. ***ConstroiHeap***(***A, fim***)
3. Para ***i*** de ***n*** até ***1*** faça
4. troca ***A[1]*** ↔ ***A[i]***
5. ***n*** ← ***n-1***
6. ***Heapifica***(***A, n, 0***)

ConstroiHeap(arranjo ***A, fim***)

1. Para ***i*** de ***fim /2*** até ***0*** faça
2. ***Heapifica***(***A, fim, i***)

Heapsort: Construindo a Heap

Heapsort(arranjo ***A, fim***)

1. ***n*** ← ***fim***
2. ***ConstroiHeap***(***A, fim***)
3. Para ***i*** de ***n*** até ***1*** faça
4. troca ***A[1]*** ↔ ***A[i]***
5. ***n*** ← ***n-1***
6. ***Heapifica***(***A, n, 0***)

ConstroiHeap(arranjo ***A, fim***)

1. Para ***i*** de ***fim /2*** até ***0*** faça
2. ***Heapifica***(***A, fim, i***)

UniCesumar

Heapsort: Construindo a Heap

Heapsort(arranjo ***A, fim***)

1. ***n*** ← ***fim***
2. ***ConstroiHeap***(***A, fim***)
3. Para ***i*** de ***n*** até ***1*** faça
4. troca ***A[1]*** ↔ ***A[i]***
5. ***n*** ← ***n-1***
6. ***Heapifica***(***A, n, 0***)

ConstroiHeap(arranjo ***A, fim***)

1. Para ***i*** de ***fim /2*** até ***0*** faça
2. ***Heapifica***(***A, fim, i***)

Heapsort: Construindo a Heap

Heapsort(arranjo ***A, fim***)

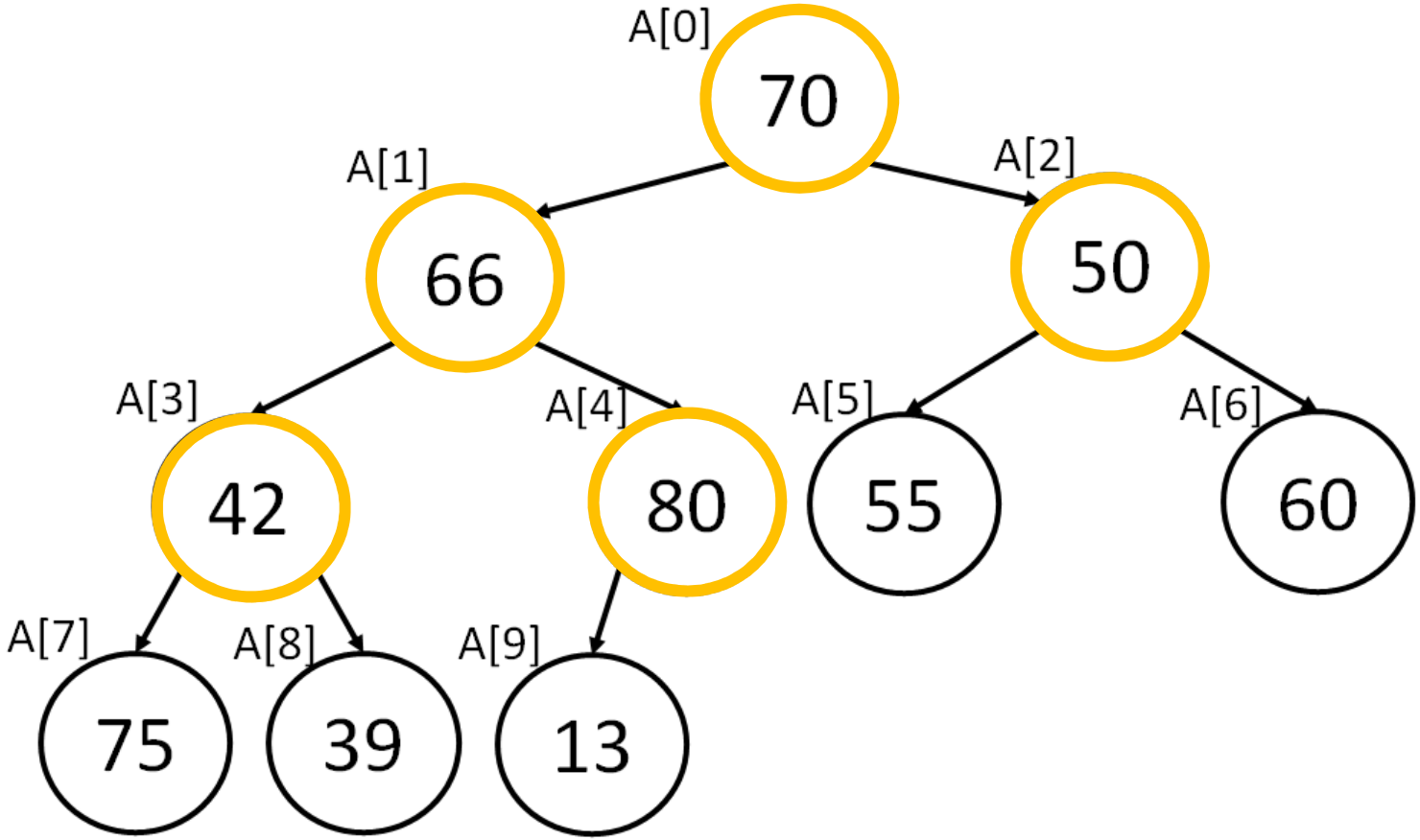
1. ***n*** ← ***fim***
2. ***ConstroiHeap***(***A, fim***)
3. Para ***i*** de ***n*** até ***1*** faça
4. troca ***A[1]*** ↔ ***A[i]***
5. ***n*** ← ***n-1***
6. ***Heapifica***(***A, n, 0***)

ConstroiHeap(arranjo ***A, fim***)

1. Para ***i*** de ***fim /2*** até ***0*** faça
2. ***Heapifica***(***A, fim, i***)

Construindo a Heap: método ConstroiHeap()

i	0	1	2	3	4	5	6	7	8	9
A[i]	70	66	50	42	80	55	60	75	39	13



Construindo a Heap: método ConstroiHeap()

i	0	1	2	3	4	5	6	7	8	9
A[i]	70	66	50	42	80	55	60	75	39	13

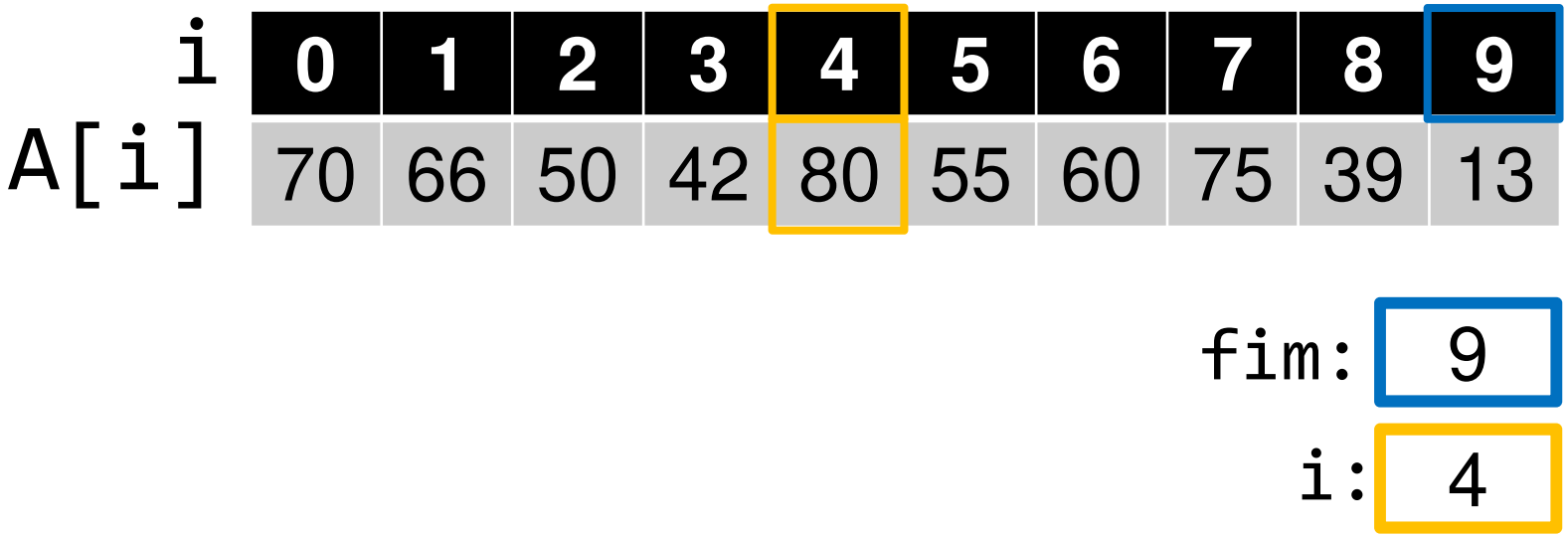
fim: 9

i:

ConstroiHeap(arranjo ***A, fim***)

1. Para ***i*** de ***fim /2*** até ***0*** faça
2. ***Heapifica***(***A, fim, i***)

Construindo a Heap: método ConstroiHeap()

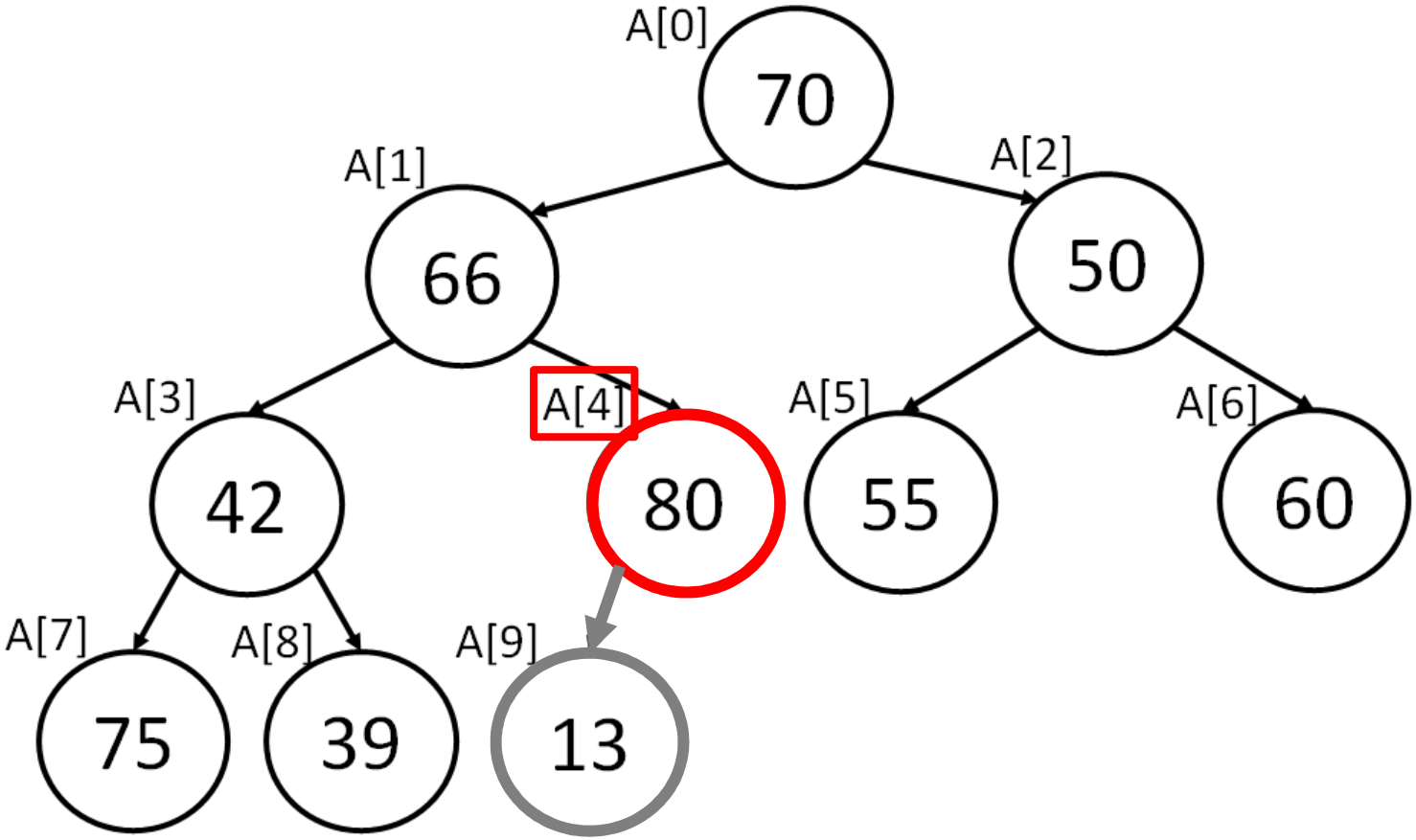


ConstroiHeap(arranjo ***A, fim***)

1. Para ***i*** de ***fim /2*** até ***0*** faça
2. ***Heapifica***(***A, fim, i***)

Construindo a Heap: método Heapifica()

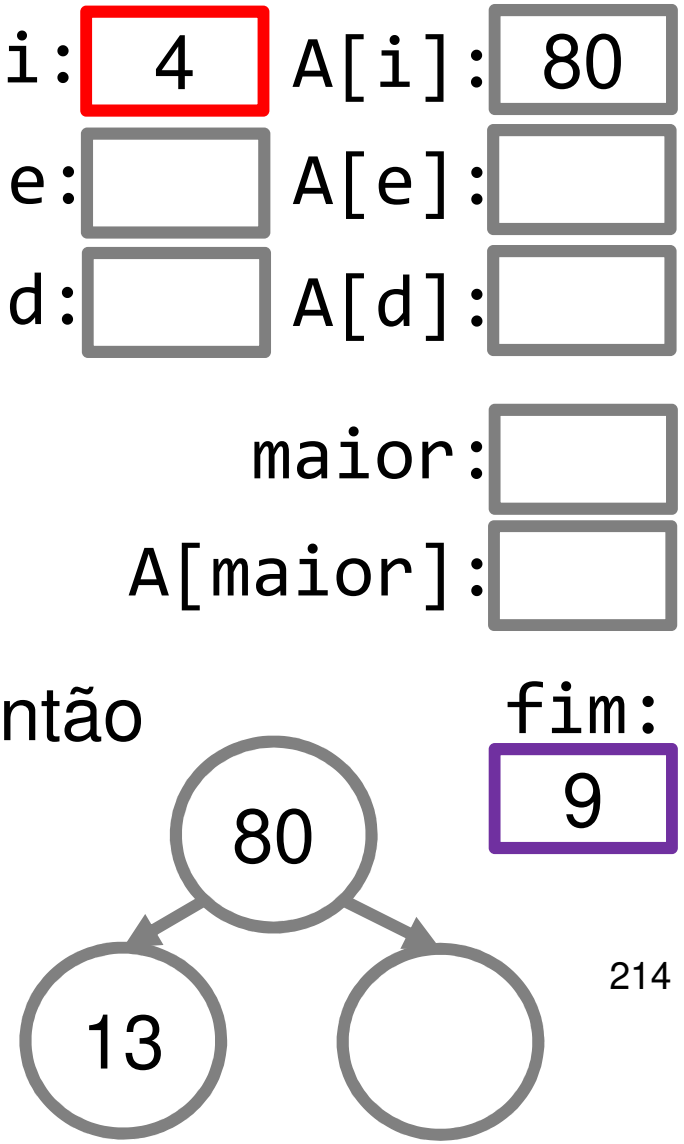
i	0	1	2	3	4	5	6	7	8	9
A[i]	70	66	50	42	80	55	60	75	39	13



Construindo a Heap: método Heapifica()

Heapifica(arranjo ***A, fim, i***)

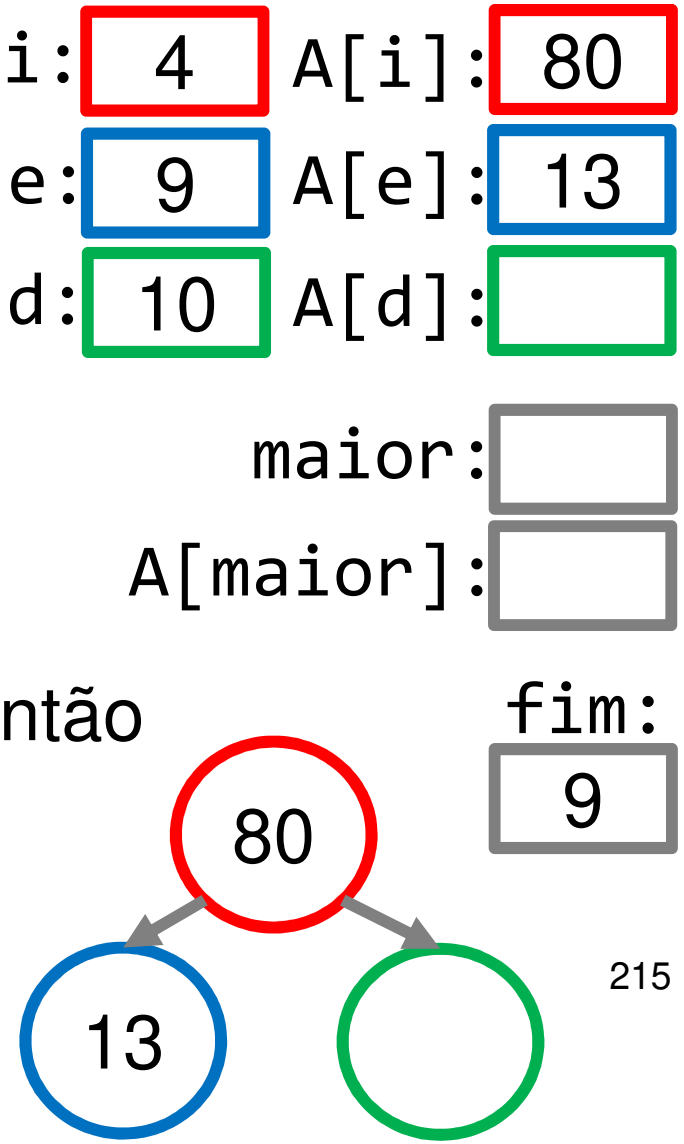
1. ***e*** ← ***2*i + 1***
2. ***d*** ← ***2*i + 2***
3. Se ***e ≤ fim*** e ***A[e] > A[i]*** então
4. ***maior*** ← ***e***
5. Senão
6. ***maior*** ← ***i***
7. Se ***d ≤ fim*** e ***A[d] > A[maior]*** então
8. ***maior*** ← ***d***
9. Se ***maior ≠ i*** então
10. troca ***A[i]*** ↔ ***A[maior]***
11. ***Heapifica***(***A, fim, maior***)



Construindo a Heap: método Heapifica()

Heapifica(arranjo ***A, fim, i***)

1. ***e*** ← ***2*i + 1***
2. ***d*** ← ***2*i + 2***
3. Se ***e ≤ fim*** e ***A[e] > A[i]*** então
4. ***maior*** ← ***e***
5. Senão
6. ***maior*** ← ***i***
7. Se ***d ≤ fim*** e ***A[d] > A[maior]*** então
8. ***maior*** ← ***d***
9. Se ***maior ≠ i*** então
10. troca ***A[i]*** ↔ ***A[maior]***
11. ***Heapifica***(***A, fim, maior***)

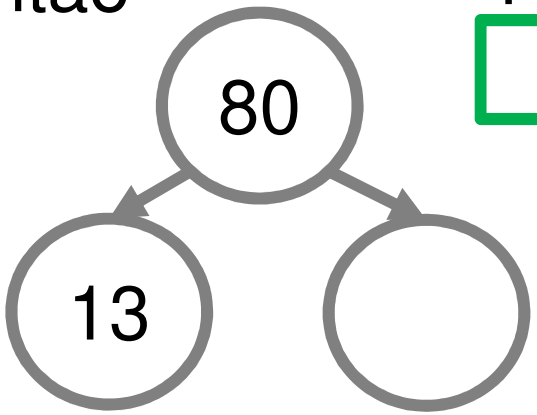


Construindo a Heap: método Heapifica()

Heapifica(arranjo ***A, fim, i***)

1. ***e*** ← ***2*i + 1***
2. ***d*** ← ***2*i + 2***
3. Se ***e ≤ fim*** e ***A[e] > A[i]*** então
4. ***maior*** ← ***e***
5. Senão
6. ***maior*** ← ***i***
7. Se ***d ≤ fim*** e ***A[d] > A[maior]*** então
8. ***maior*** ← ***d***
9. Se ***maior ≠ i*** então
10. troca ***A[i]*** ↔ ***A[maior]***
11. ***Heapifica***(***A, fim, maior***)

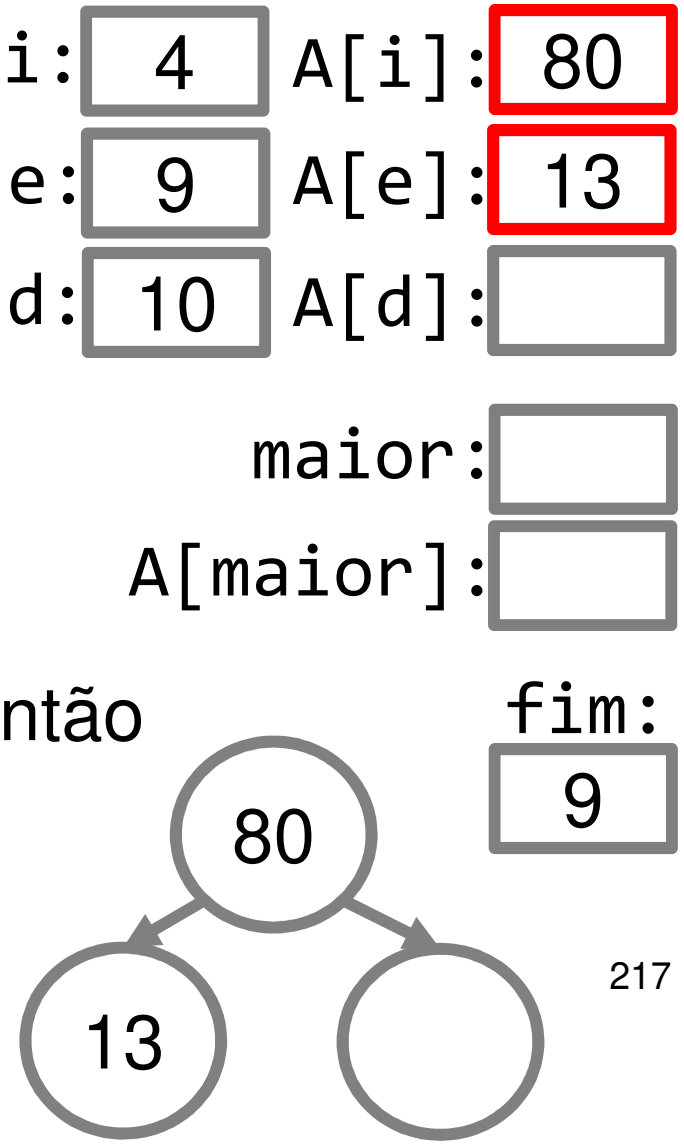
i:	4	A[i]:	80
e:	9	A[e]:	13
d:	10	A[d]:	
maior:			
A[maior]:			
fim:	9		



Construindo a Heap: método Heapifica()

Heapifica(arranjo ***A, fim, i***)

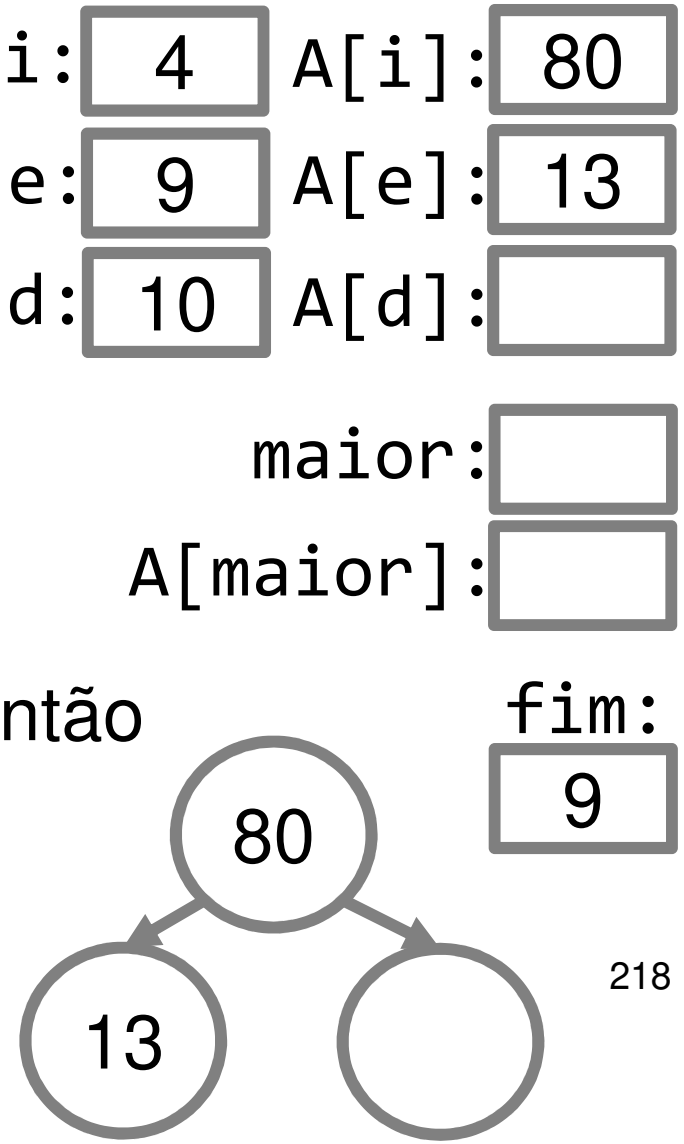
1. ***e*** ← ***2*i + 1***
2. ***d*** ← ***2*i + 2***
3. Se ***e ≤ fim*** e ***A[e] > A[i]*** então
4. ***maior*** ← ***e***
5. Senão
6. ***maior*** ← ***i***
7. Se ***d ≤ fim*** e ***A[d] > A[maior]*** então
8. ***maior*** ← ***d***
9. Se ***maior ≠ i*** então
10. troca ***A[i]*** ↔ ***A[maior]***
11. ***Heapifica***(***A, fim, maior***)



Construindo a Heap: método Heapifica()

Heapifica(arranjo ***A, fim, i***)

1. ***e*** ← ***2*i + 1***
2. ***d*** ← ***2*i + 2***
3. Se ***e ≤ fim*** e ***A[e] > A[i]*** então
4. ***maior*** ← ***e***
5. Senão
6. ***maior*** ← ***i***
7. Se ***d ≤ fim*** e ***A[d] > A[maior]*** então
8. ***maior*** ← ***d***
9. Se ***maior ≠ i*** então
10. troca ***A[i]*** ↔ ***A[maior]***
11. ***Heapifica***(***A, fim, maior***)

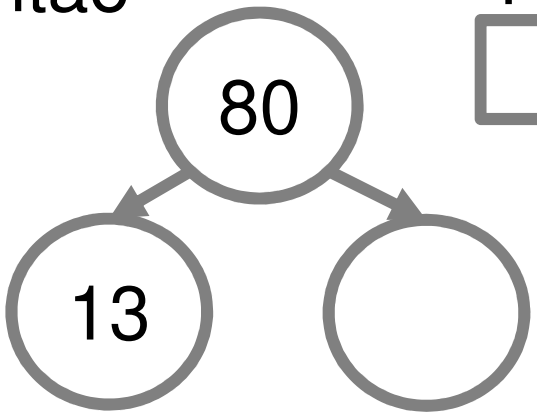


Construindo a Heap: método Heapifica()

Heapifica(arranjo ***A, fim, i***)

1. ***e*** ← ***2*i + 1***
2. ***d*** ← ***2*i + 2***
3. Se ***e ≤ fim*** e ***A[e] > A[i]*** então
4. ***maior*** ← ***e***
5. Senão
6. ***maior*** ← ***i***
7. Se ***d ≤ fim*** e ***A[d] > A[maior]*** então
8. ***maior*** ← ***d***
9. Se ***maior ≠ i*** então
10. troca ***A[i]*** ↔ ***A[maior]***
11. ***Heapifica***(***A, fim, maior***)

i:	4	A[i]:	80
e:	9	A[e]:	13
d:	10	A[d]:	
maior:	4		
A[maior]:	80		
fim:	9		

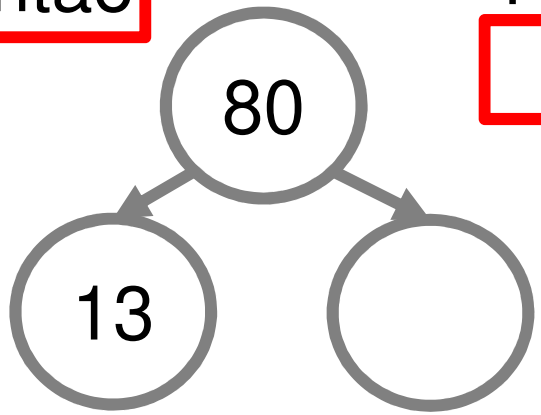


Construindo a Heap: método Heapifica()

Heapifica(arranjo ***A, fim, i***)

1. ***e*** ← ***2*i + 1***
2. ***d*** ← ***2*i + 2***
3. Se ***e ≤ fim*** e ***A[e] > A[i]*** então
4. ***maior*** ← ***e***
5. Senão
6. ***maior*** ← ***i***
7. Se ***d ≤ fim*** e ***A[d] > A[maior]*** então
8. ***maior*** ← ***d***
9. Se ***maior ≠ i*** então
10. troca ***A[i]*** ↔ ***A[maior]***
11. ***Heapifica***(***A, fim, maior***)

i:	4	A[i]:	80
e:	9	A[e]:	13
d:	10	A[d]:	
maior:	4		
A[maior]:	80		
fim:	9		



Construindo a Heap: método Heapifica()

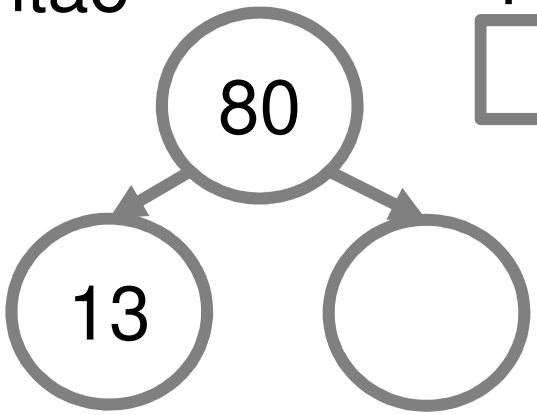
Heapifica(arranjo ***A, fim, i***)

1. ***e*** ← ***2*i + 1***
2. ***d*** ← ***2*i + 2***
3. Se ***e ≤ fim*** e ***A[e] > A[i]*** então
4. ***maior*** ← ***e***
5. Senão
6. ***maior*** ← ***i***
7. Se ***d ≤ fim*** e ***A[d] > A[maior]*** então
8. ***maior*** ← ***d***
9. Se ***maior ≠ i*** então
10. troca ***A[i]*** ↔ ***A[maior]***
11. ***Heapifica***(***A, fim, maior***)

i:	4	A[i]:	80
e:	9	A[e]:	13
d:	10	A[d]:	

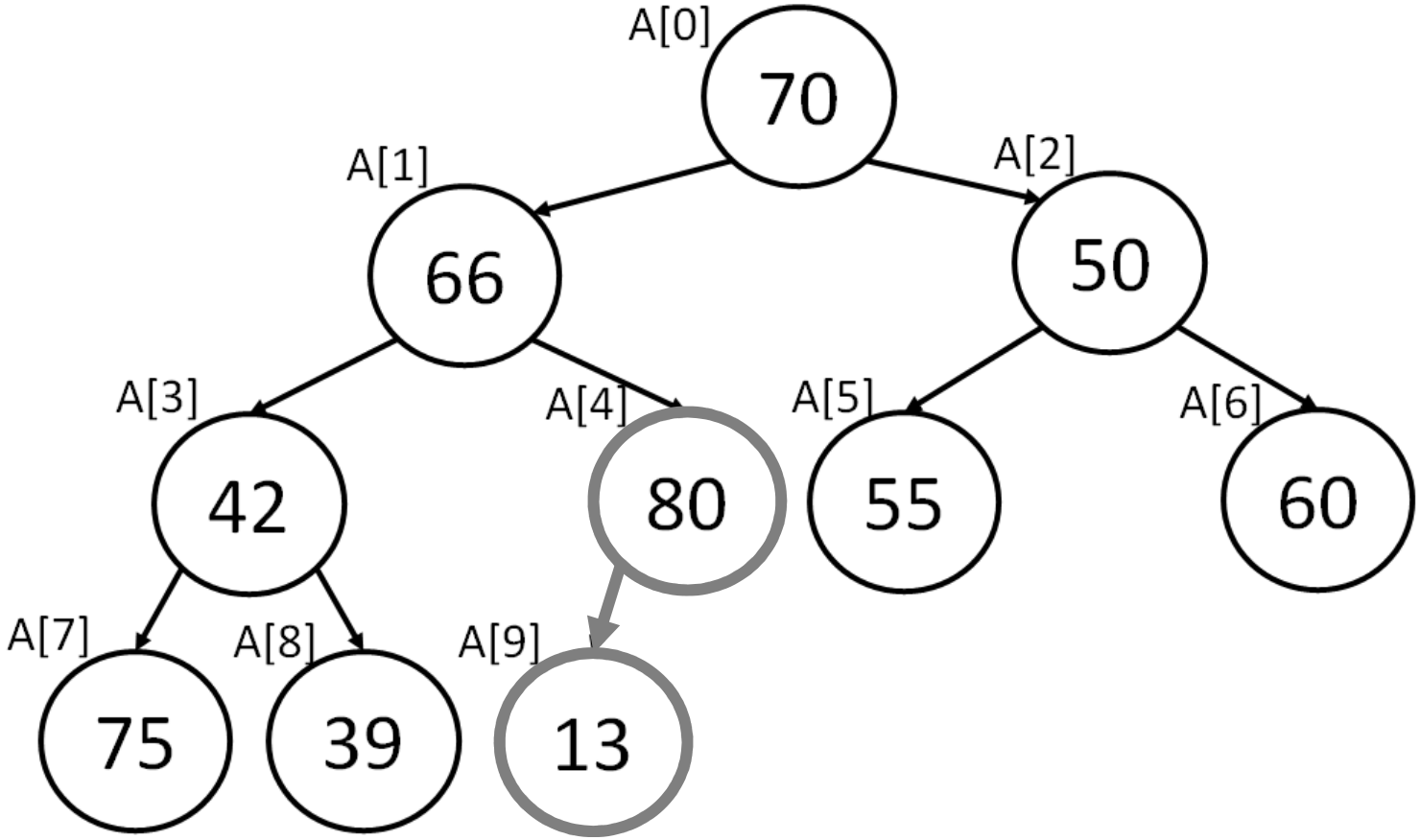
maior:	4
A[maior]:	80

fim: 9



Construindo a Heap: método Heapifica()

i	0	1	2	3	4	5	6	7	8	9
A[i]	70	66	50	42	80	55	60	75	39	13



Construindo a Heap: método ConstroiHeap()

i	0	1	2	3	4	5	6	7	8	9
A[i]	70	66	50	42	80	55	60	75	39	13

fim: 9

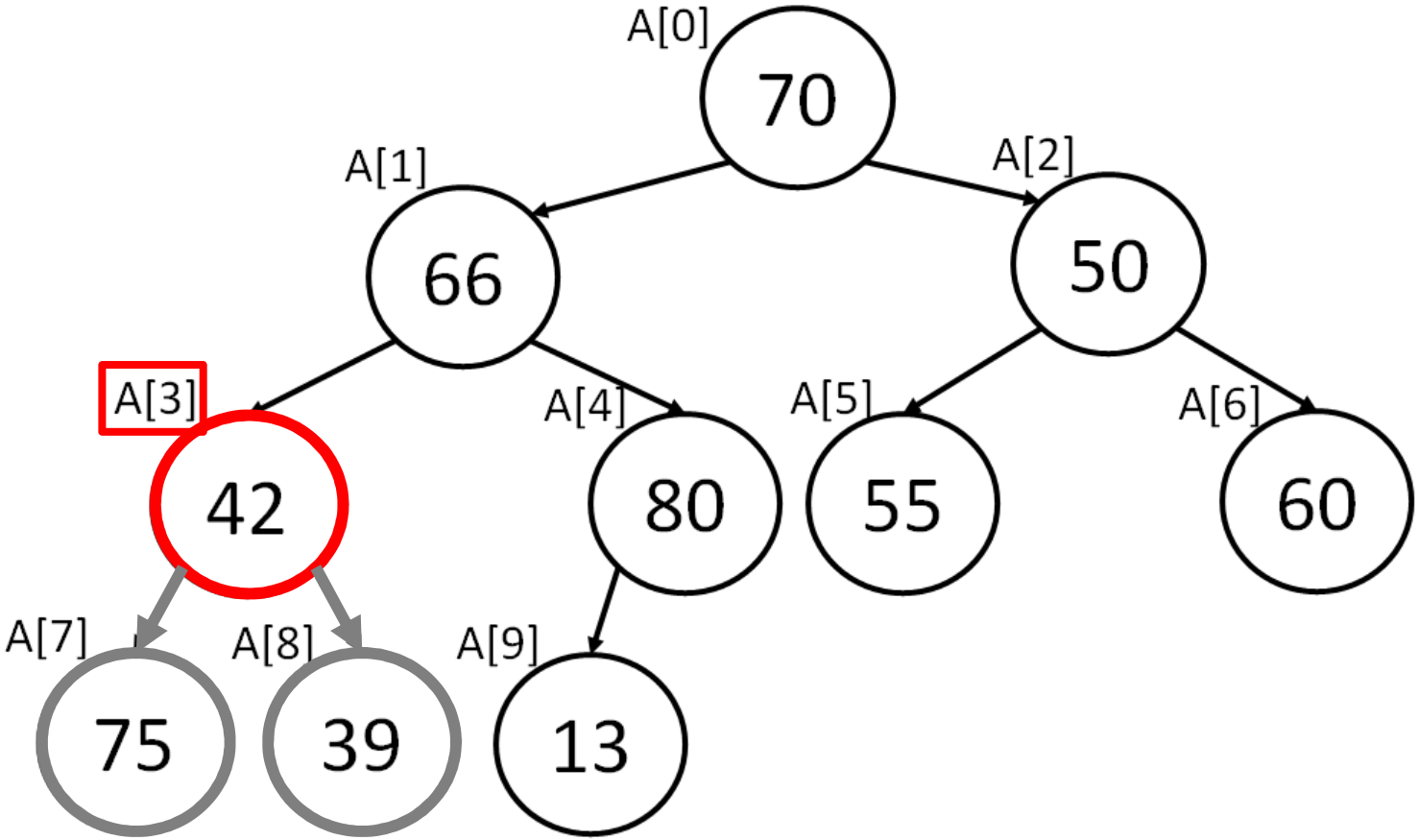
i: 3

ConstroiHeap(arranjo ***A, fim***)

1. Para ***i*** de ***fim /2*** até ***0*** faça
2. ***Heapifica***(***A, fim, i***)

Construindo a Heap: método Heapifica()

i	0	1	2	3	4	5	6	7	8	9
A[i]	70	66	50	42	80	55	60	75	39	13



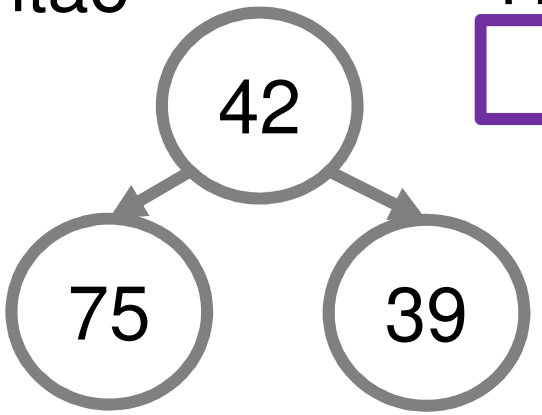
Construindo a Heap: método Heapifica()

Heapifica(arranjo ***A***, ***fim***, ***i***)

1. ***e*** ← ***2*i + 1***
2. ***d*** ← ***2*i + 2***
3. Se ***e*** ≤ ***fim*** e ***A[e]*** > ***A[i]*** então
4. ***maior*** ← ***e***
5. Senão
6. ***maior*** ← ***i***
7. Se ***d*** ≤ ***fim*** e ***A[d]*** > ***A[maior]*** então
8. ***maior*** ← ***d***
9. Se ***maior*** ≠ ***i*** então
10. troca ***A[i]*** ↔ ***A[maior]***
11. ***Heapifica***(***A, fim, maior***)

i:	3	A[i]:	42
e:		A[e]:	
d:		A[d]:	
maior:			
A[maior]:			

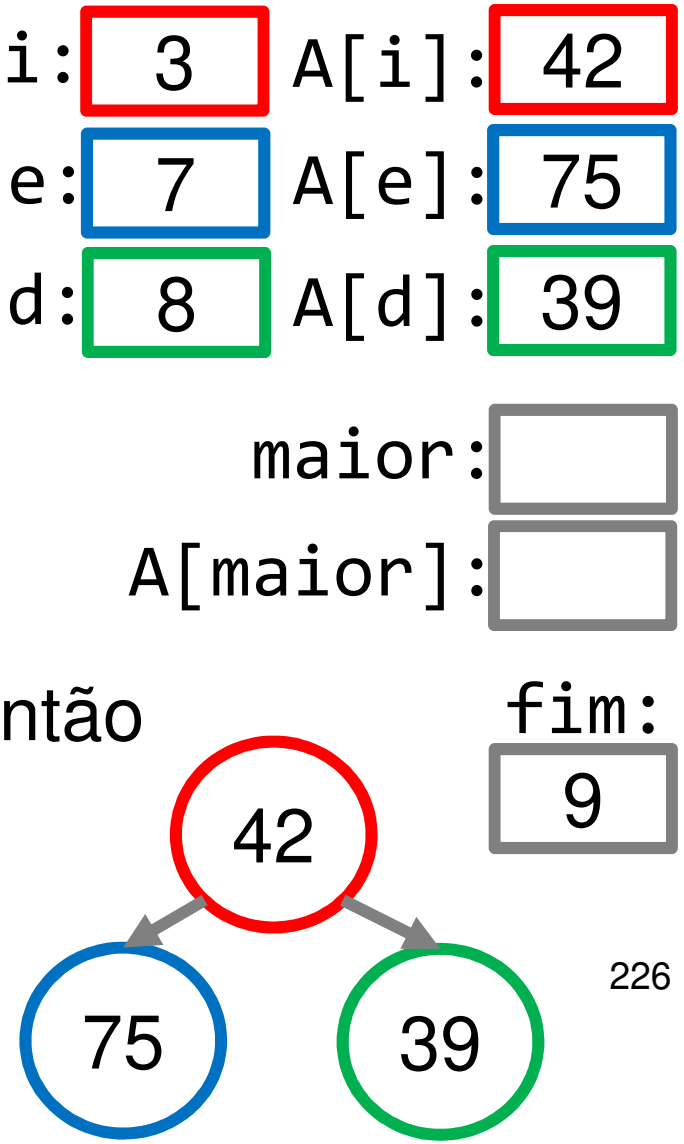
fim: 9



Construindo a Heap: método Heapifica()

Heapifica(arranjo ***A, fim, i***)

1. ***e*** ← ***2*i + 1***
2. ***d*** ← ***2*i + 2***
3. Se ***e ≤ fim*** e ***A[e] > A[i]*** então
4. ***maior*** ← ***e***
5. Senão
6. ***maior*** ← ***i***
7. Se ***d ≤ fim*** e ***A[d] > A[maior]*** então
8. ***maior*** ← ***d***
9. Se ***maior ≠ i*** então
10. troca ***A[i]*** ↔ ***A[maior]***
11. ***Heapifica***(***A, fim, maior***)

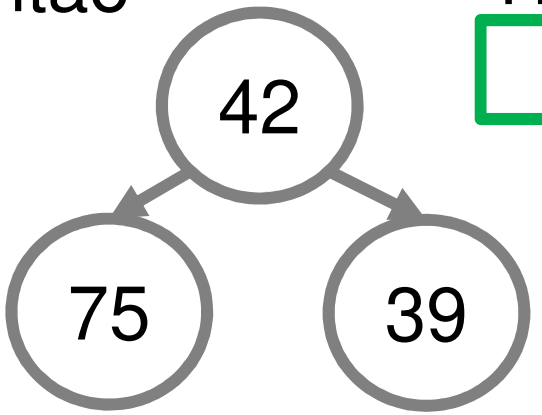


Construindo a Heap: método Heapifica()

Heapifica(arranjo ***A, fim, i***)

1. ***e*** ← ***2*i + 1***
2. ***d*** ← ***2*i + 2***
3. Se ***e ≤ fim*** e ***A[e] > A[i]*** então
4. ***maior*** ← ***e***
5. Senão
6. ***maior*** ← ***i***
7. Se ***d ≤ fim*** e ***A[d] > A[maior]*** então
8. ***maior*** ← ***d***
9. Se ***maior ≠ i*** então
10. troca ***A[i]*** ↔ ***A[maior]***
11. ***Heapifica***(***A, fim, maior***)

i:	3	A[i]:	42
e:	7	A[e]:	75
d:	8	A[d]:	39
maior:			
A[maior]:			
fim:	9		

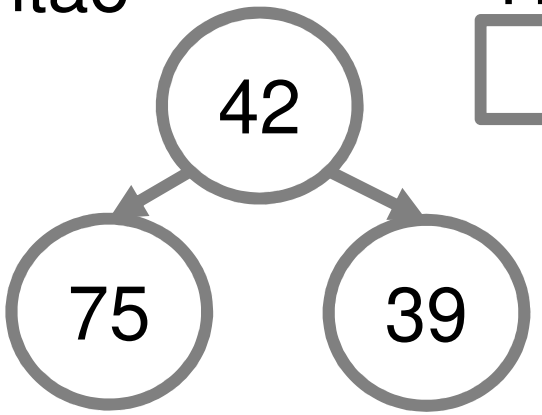


Construindo a Heap: método Heapifica()

Heapifica(arranjo ***A, fim, i***)

1. ***e*** ← ***2*i + 1***
2. ***d*** ← ***2*i + 2***
3. Se ***e ≤ fim*** e ***A[e] > A[i]*** então
4. ***maior*** ← ***e***
5. Senão
6. ***maior*** ← ***i***
7. Se ***d ≤ fim*** e ***A[d] > A[maior]*** então
8. ***maior*** ← ***d***
9. Se ***maior ≠ i*** então
10. troca ***A[i]*** ↔ ***A[maior]***
11. ***Heapifica***(***A, fim, maior***)

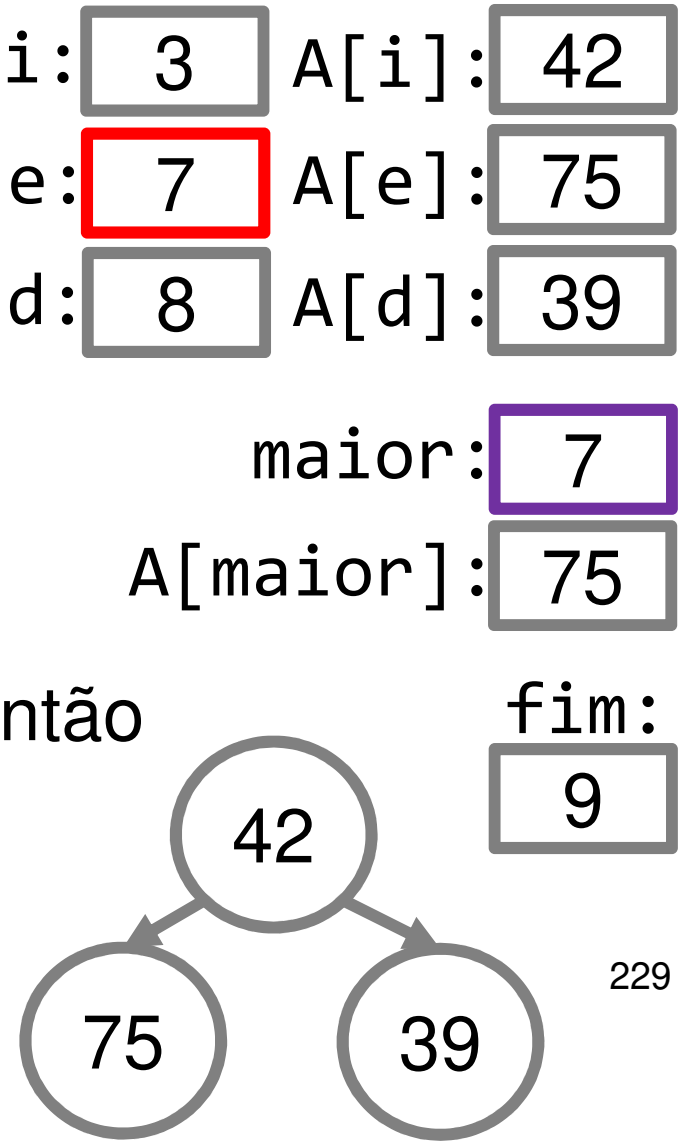
i:	3	A[i]:	42
e:	7	A[e]:	75
d:	8	A[d]:	39
maior:			
A[maior]:			
fim:	9		



Construindo a Heap: método Heapifica()

Heapifica(arranjo ***A, fim, i***)

1. ***e*** ← ***2*i + 1***
2. ***d*** ← ***2*i + 2***
3. Se ***e ≤ fim*** e ***A[e] > A[i]*** então
4. ***maior*** ← ***e***
5. Senão
6. ***maior*** ← ***i***
7. Se ***d ≤ fim*** e ***A[d] > A[maior]*** então
8. ***maior*** ← ***d***
9. Se ***maior ≠ i*** então
10. troca ***A[i]*** ↔ ***A[maior]***
11. ***Heapifica***(***A, fim, maior***)

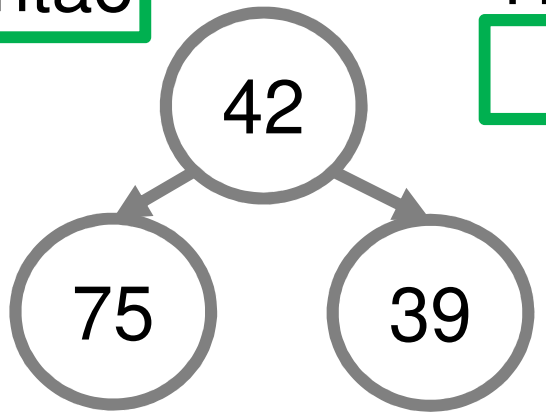


Construindo a Heap: método Heapifica()

Heapifica(arranjo ***A, fim, i***)

1. ***e*** ← ***2*i + 1***
2. ***d*** ← ***2*i + 2***
3. Se ***e ≤ fim*** e ***A[e] > A[i]*** então
4. ***maior*** ← ***e***
5. Senão
6. ***maior*** ← ***i***
7. Se ***d ≤ fim*** e ***A[d] > A[maior]*** então
8. ***maior*** ← ***d***
9. Se ***maior ≠ i*** então
10. troca ***A[i]*** ↔ ***A[maior]***
11. ***Heapifica***(***A, fim, maior***)

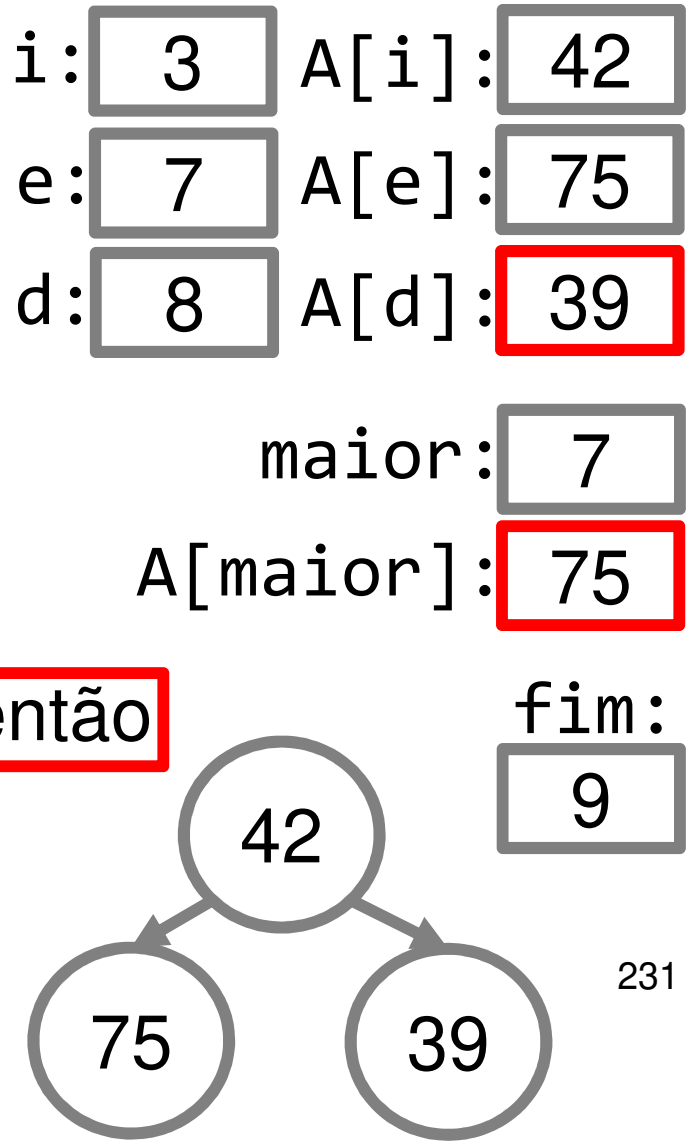
i:	3	A[i]:	42
e:	7	A[e]:	75
d:	8	A[d]:	39
maior:	7		
A[maior]:	75		
fim:	9		



Construindo a Heap: método Heapifica()

Heapifica(arranjo ***A, fim, i***)

1. ***e*** ← ***2*i + 1***
2. ***d*** ← ***2*i + 2***
3. Se ***e*** ≤ ***fim*** e ***A[e] > A[i]*** então
4. ***maior*** ← ***e***
5. Senão
6. ***maior*** ← ***i***
7. Se ***d*** ≤ ***fim*** e ***A[d] > A[maior]*** então
8. ***maior*** ← ***d***
9. Se ***maior*** ≠ ***i*** então
10. troca ***A[i]*** ↔ ***A[maior]***
11. ***Heapifica***(***A, fim, maior***)



Construindo a Heap: método Heapifica()

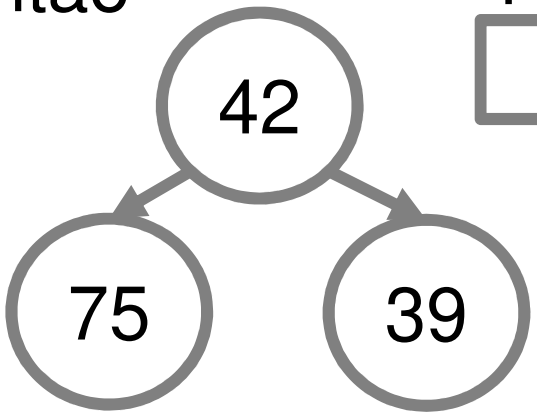
Heapifica(arranjo ***A, fim, i***)

1. ***e*** ← ***2*i + 1***
2. ***d*** ← ***2*i + 2***
3. Se ***e ≤ fim*** e ***A[e] > A[i]*** então
4. ***maior*** ← ***e***
5. Senão
6. ***maior*** ← ***i***
7. Se ***d ≤ fim*** e ***A[d] > A[maior]*** então
8. ***maior*** ← ***d***
9. Se ***maior ≠ i*** então
10. troca ***A[i]*** ↔ ***A[maior]***
11. ***Heapifica***(***A, fim, maior***)

i:	3	A[i]:	42
e:	7	A[e]:	75
d:	8	A[d]:	39

maior:	7
A[maior]:	75

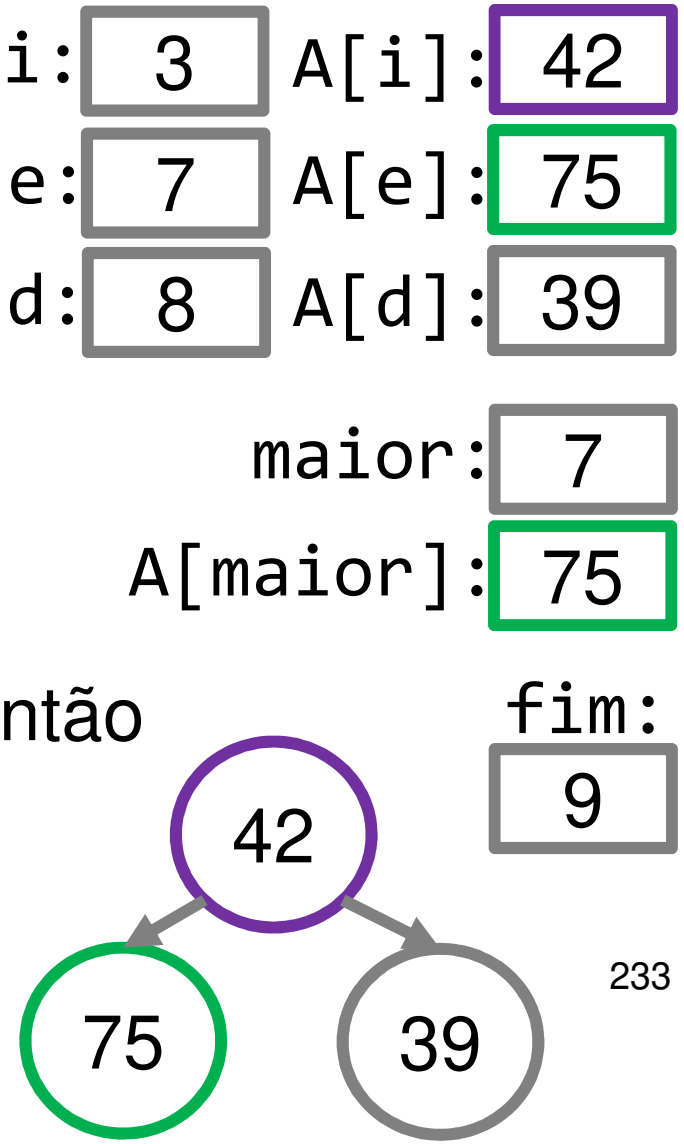
fim: 9



Construindo a Heap: método Heapifica()

Heapifica(arranjo ***A, fim, i***)

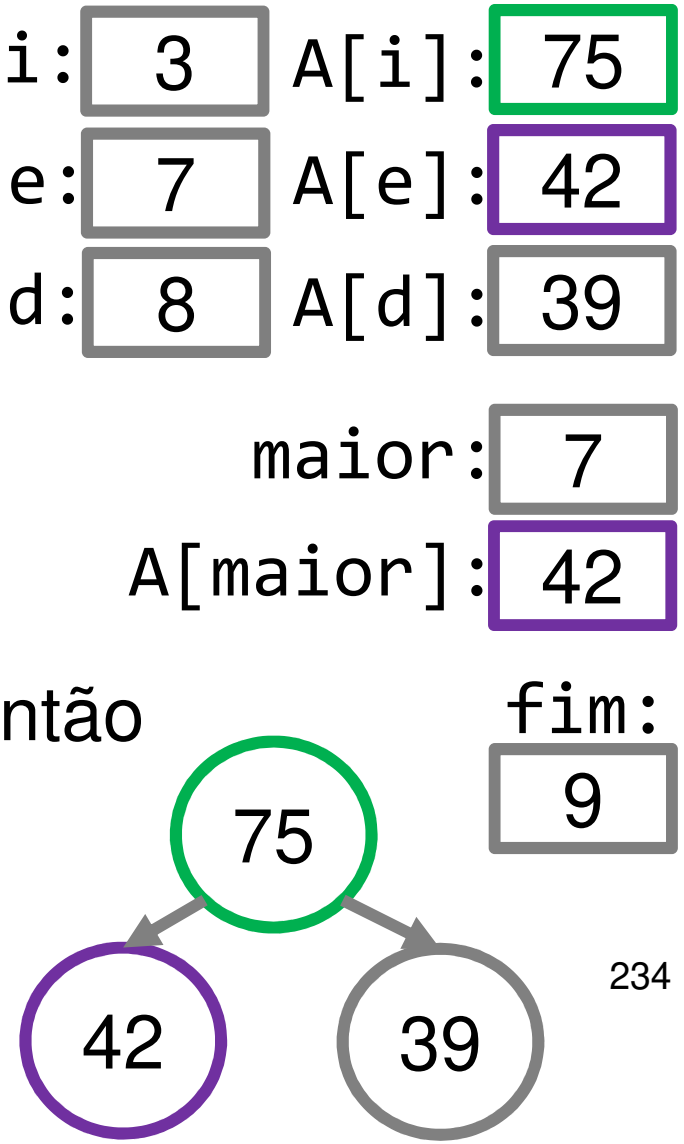
1. ***e*** ← ***2*i + 1***
2. ***d*** ← ***2*i + 2***
3. Se ***e ≤ fim*** e ***A[e] > A[i]*** então
4. ***maior*** ← ***e***
5. Senão
6. ***maior*** ← ***i***
7. Se ***d ≤ fim*** e ***A[d] > A[maior]*** então
8. ***maior*** ← ***d***
9. Se ***maior ≠ i*** então
10. troca ***A[i]*** ↔ ***A[maior]***
11. ***Heapifica***(***A, fim, maior***)



Construindo a Heap: método Heapifica()

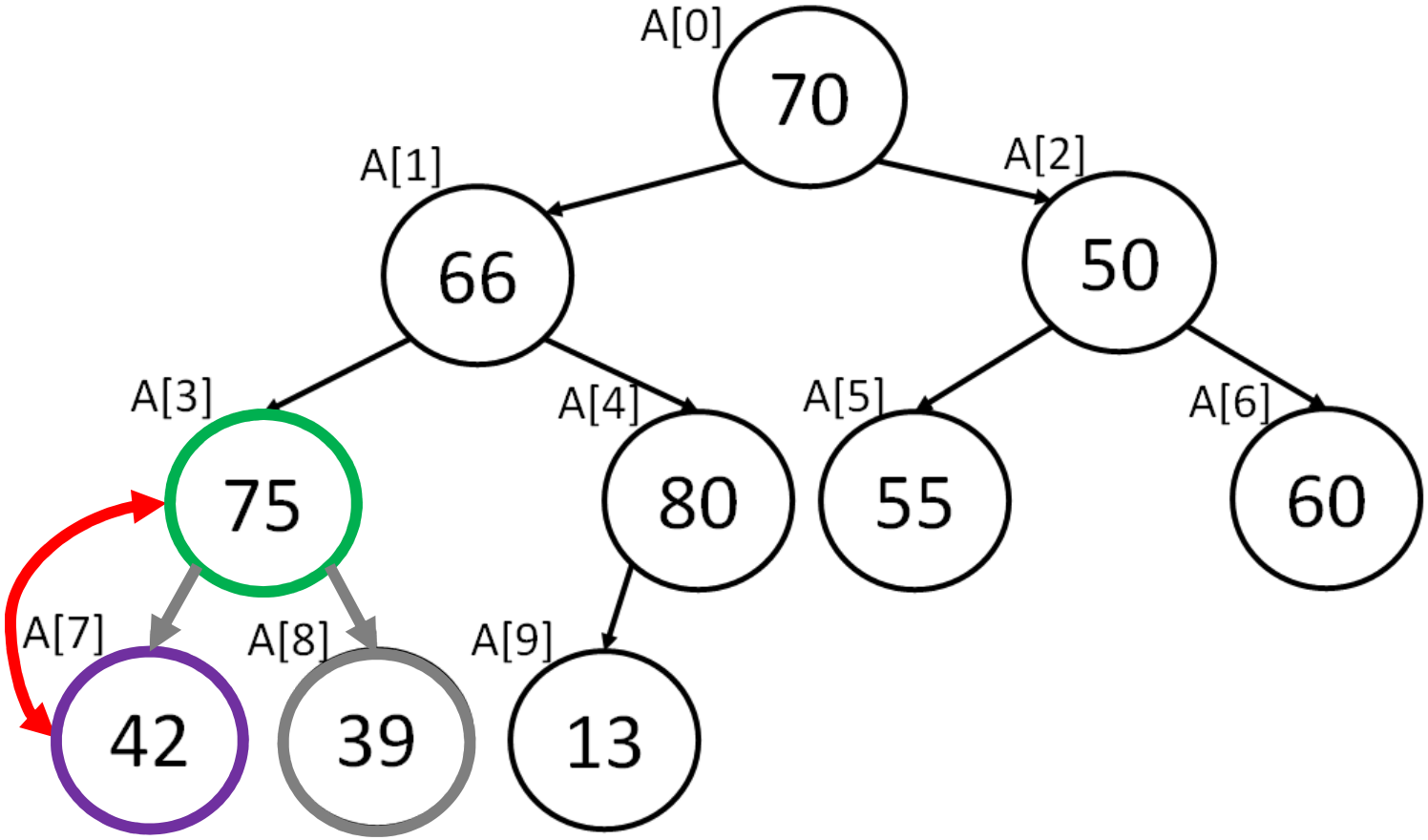
Heapifica(arranjo ***A, fim, i***)

1. ***e*** ← ***2*i + 1***
2. ***d*** ← ***2*i + 2***
3. Se ***e ≤ fim*** e ***A[e] > A[i]*** então
4. ***maior*** ← ***e***
5. Senão
6. ***maior*** ← ***i***
7. Se ***d ≤ fim*** e ***A[d] > A[maior]*** então
8. ***maior*** ← ***d***
9. Se ***maior ≠ i*** então
10. troca ***A[i]*** ↔ ***A[maior]***
11. ***Heapifica***(***A, fim, maior***)



Construindo a Heap: método Heapifica()

i	0	1	2	3	4	5	6	7	8	9
A[i]	70	66	50	75	80	55	60	42	39	13



Construindo a Heap: método Heapifica()

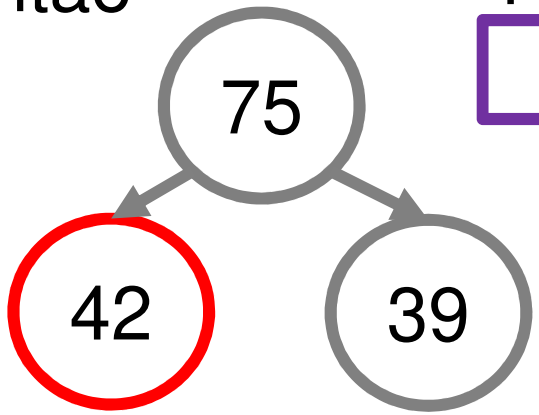
Heapifica(arranjo ***A, fim, i***)

1. ***e*** ← ***2*i + 1***
2. ***d*** ← ***2*i + 2***
3. Se ***e ≤ fim*** e ***A[e] > A[i]*** então
4. ***maior*** ← ***e***
5. Senão
6. ***maior*** ← ***i***
7. Se ***d ≤ fim*** e ***A[d] > A[maior]*** então
8. ***maior*** ← ***d***
9. Se ***maior ≠ i*** então
10. troca ***A[i]*** ↔ ***A[maior]***
11. ***Heapifica***(***A, fim, maior***)

i:	3	A[i]:	75
e:	7	A[e]:	42
d:	8	A[d]:	39

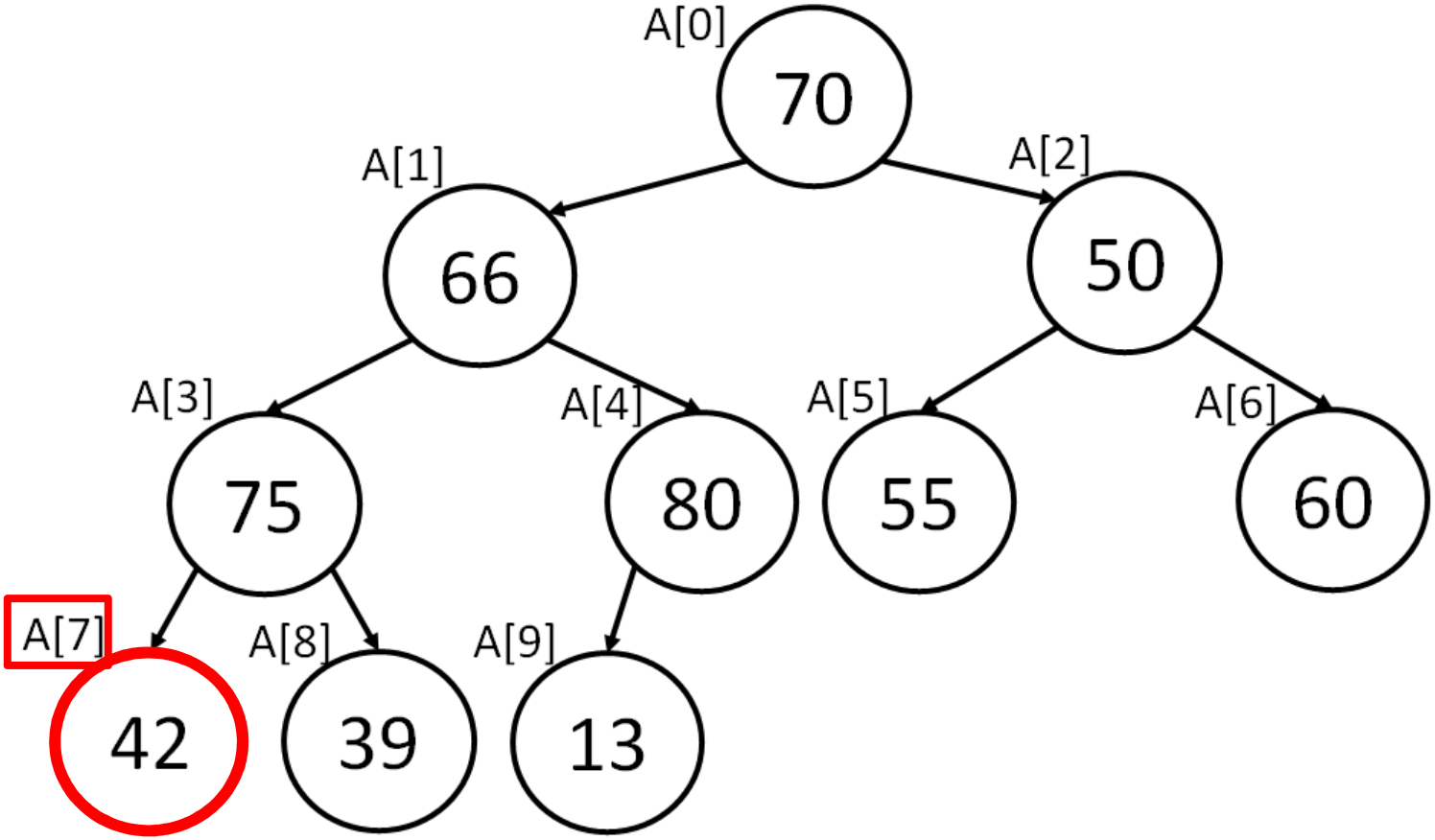
maior:	7
A[maior]:	42

fim: 9



Construindo a Heap: método Heapifica()

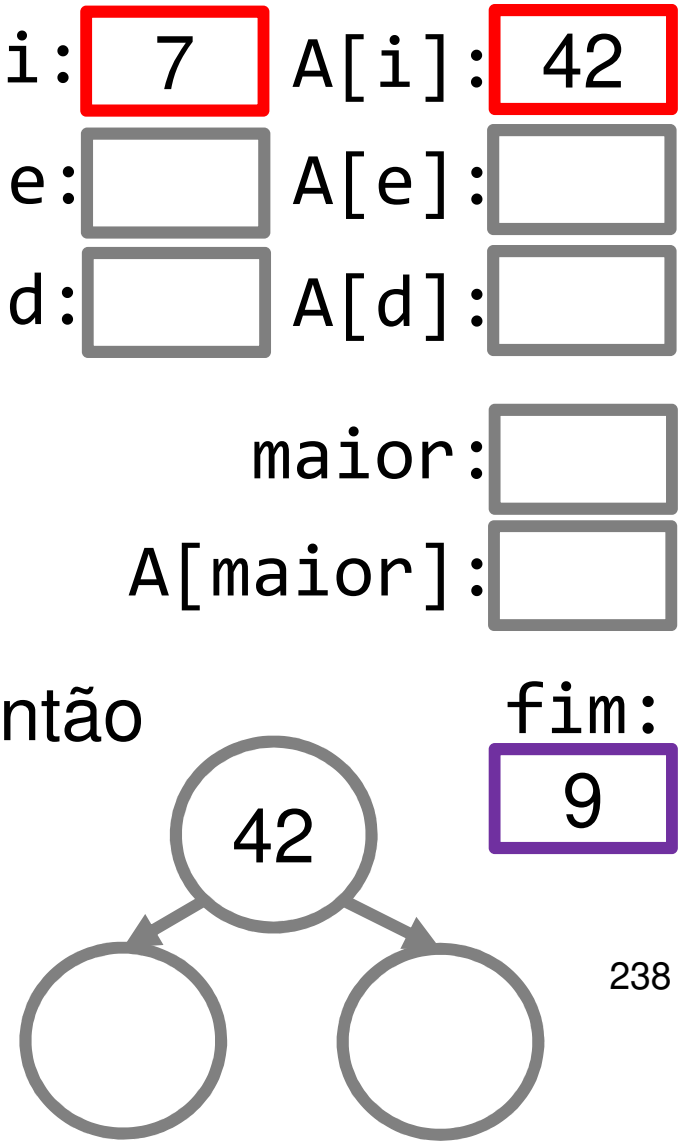
i	0	1	2	3	4	5	6	7	8	9
A[i]	70	66	50	75	80	55	60	42	39	13



Construindo a Heap: método Heapifica()

Heapifica(arranjo ***A***, ***fim***, ***i***)

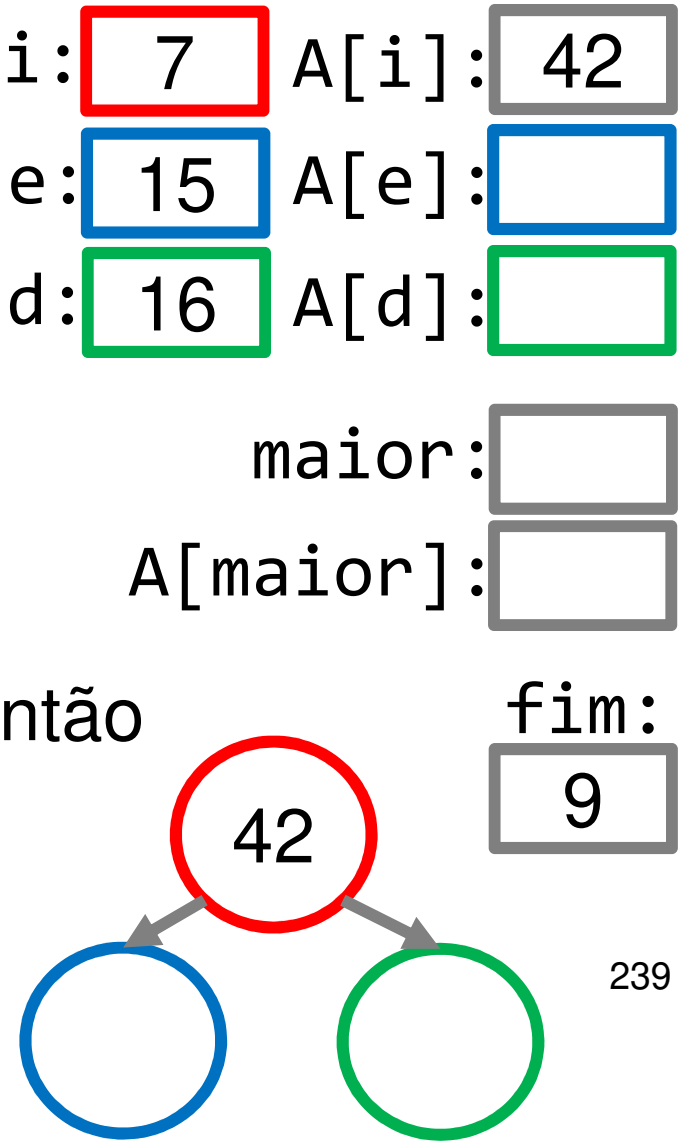
1. ***e*** ← ***2*i + 1***
2. ***d*** ← ***2*i + 2***
3. Se ***e*** ≤ ***fim*** e ***A[e]*** > ***A[i]*** então
4. ***maior*** ← ***e***
5. Senão
6. ***maior*** ← ***i***
7. Se ***d*** ≤ ***fim*** e ***A[d]*** > ***A[maior]*** então
8. ***maior*** ← ***d***
9. Se ***maior*** ≠ ***i*** então
10. troca ***A[i]*** ↔ ***A[maior]***
11. ***Heapifica***(***A, fim, maior***)



Construindo a Heap: método Heapifica()

Heapifica(arranjo ***A, fim, i***)

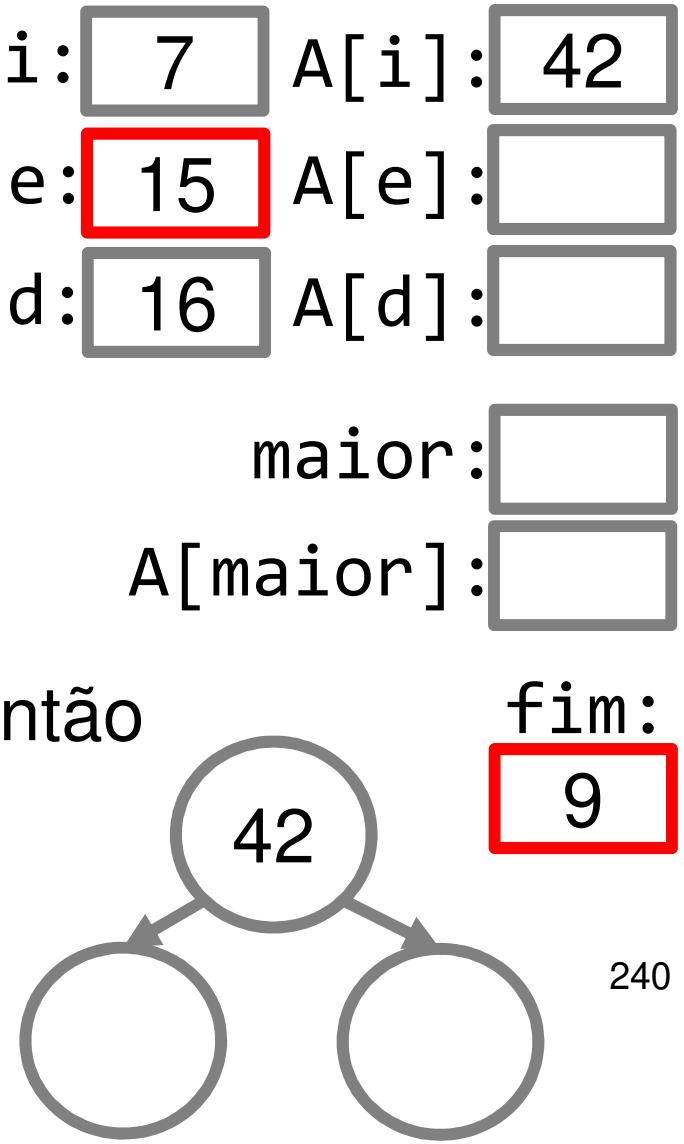
1. ***e*** ← ***2*i + 1***
2. ***d*** ← ***2*i + 2***
3. Se ***e ≤ fim*** e ***A[e] > A[i]*** então
4. ***maior*** ← ***e***
5. Senão
6. ***maior*** ← ***i***
7. Se ***d ≤ fim*** e ***A[d] > A[maior]*** então
8. ***maior*** ← ***d***
9. Se ***maior ≠ i*** então
10. troca ***A[i]*** ↔ ***A[maior]***
11. ***Heapifica***(***A, fim, maior***)



Construindo a Heap: método Heapifica()

Heapifica(arranjo ***A, fim, i***)

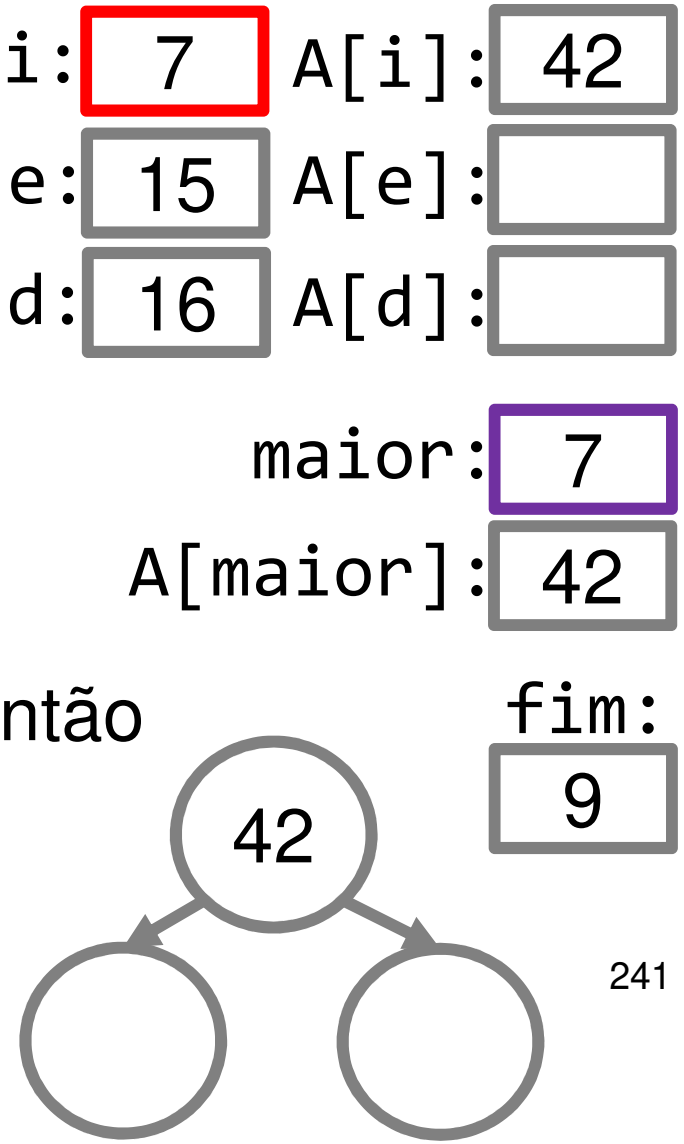
1. ***e*** ← ***2*i + 1***
2. ***d*** ← ***2*i + 2***
3. Se ***e ≤ fim*** e ***A[e] > A[i]*** então
4. ***maior*** ← ***e***
5. Senão
6. ***maior*** ← ***i***
7. Se ***d ≤ fim*** e ***A[d] > A[maior]*** então
8. ***maior*** ← ***d***
9. Se ***maior ≠ i*** então
10. troca ***A[i]*** ↔ ***A[maior]***
11. ***Heapifica***(***A, fim, maior***)



Construindo a Heap: método Heapifica()

Heapifica(arranjo ***A, fim, i***)

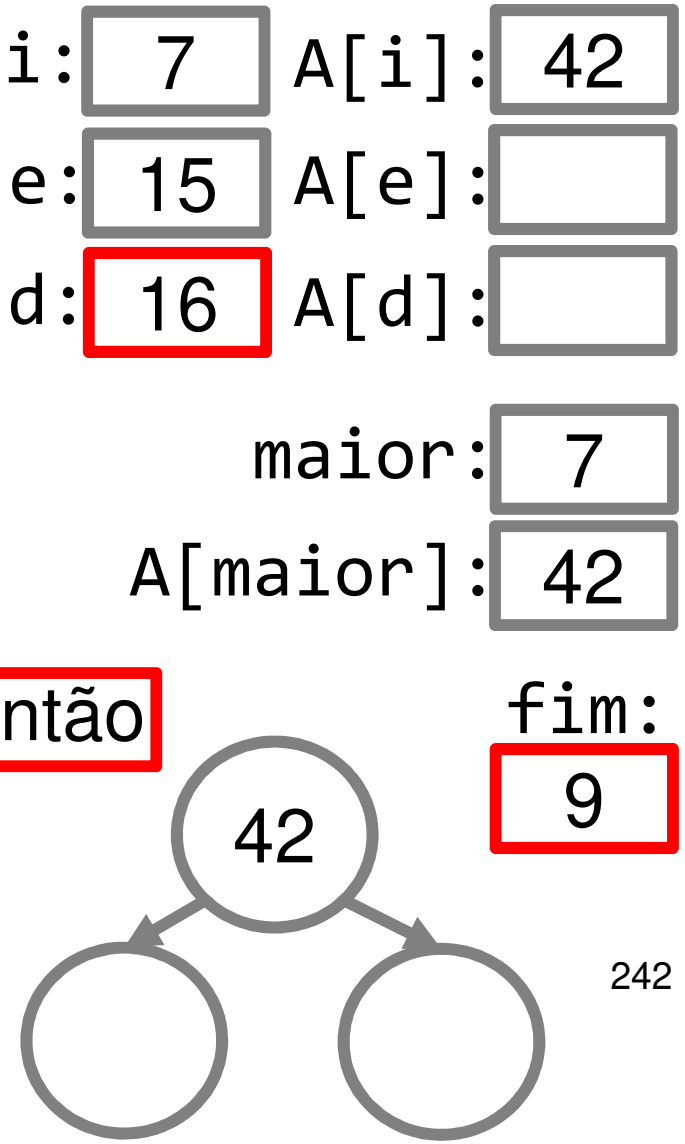
1. ***e*** ← ***2*i + 1***
2. ***d*** ← ***2*i + 2***
3. Se ***e ≤ fim*** e ***A[e] > A[i]*** então
4. ***maior*** ← ***e***
5. Senão
6. ***maior*** ← ***i***
7. Se ***d ≤ fim*** e ***A[d] > A[maior]*** então
8. ***maior*** ← ***d***
9. Se ***maior ≠ i*** então
10. troca ***A[i]*** ↔ ***A[maior]***
11. ***Heapifica***(***A, fim, maior***)



Construindo a Heap: método Heapifica()

Heapifica(arranjo ***A, fim, i***)

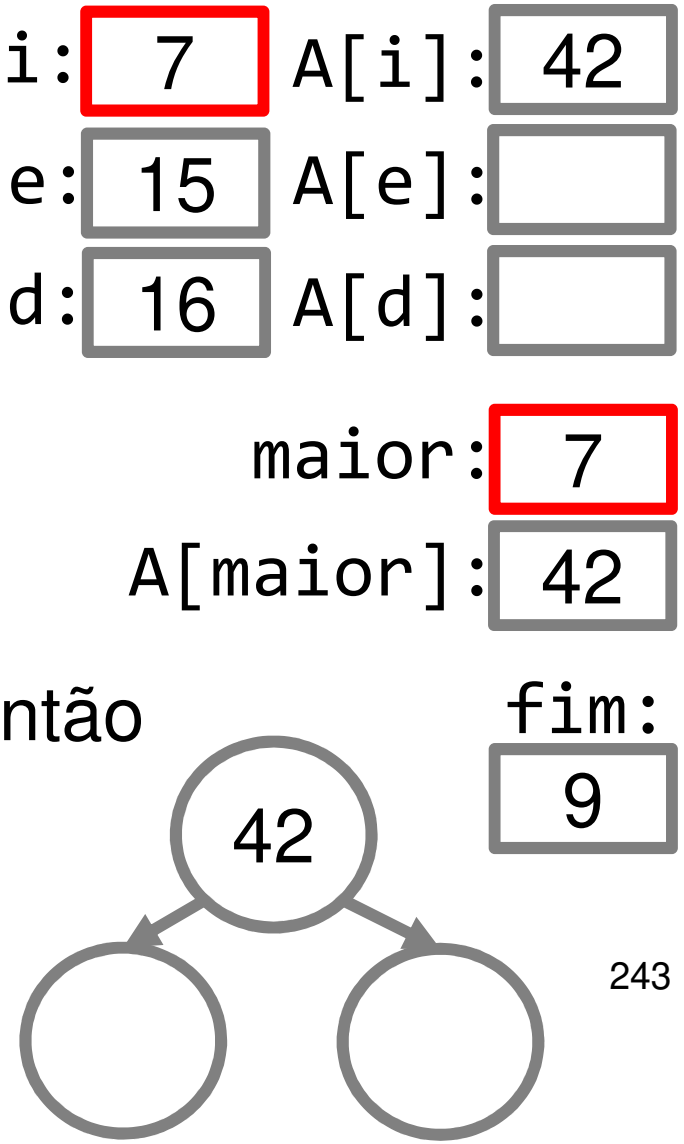
1. ***e*** ← ***2*i + 1***
2. ***d*** ← ***2*i + 2***
3. Se ***e ≤ fim*** e ***A[e] > A[i]*** então
4. ***maior*** ← ***e***
5. Senão
6. ***maior*** ← ***i***
7. Se ***d ≤ fim*** e ***A[d] > A[maior]*** então
8. ***maior*** ← ***d***
9. Se ***maior ≠ i*** então
10. troca ***A[i]*** ↔ ***A[maior]***
11. ***Heapifica***(***A, fim, maior***)



Construindo a Heap: método Heapifica()

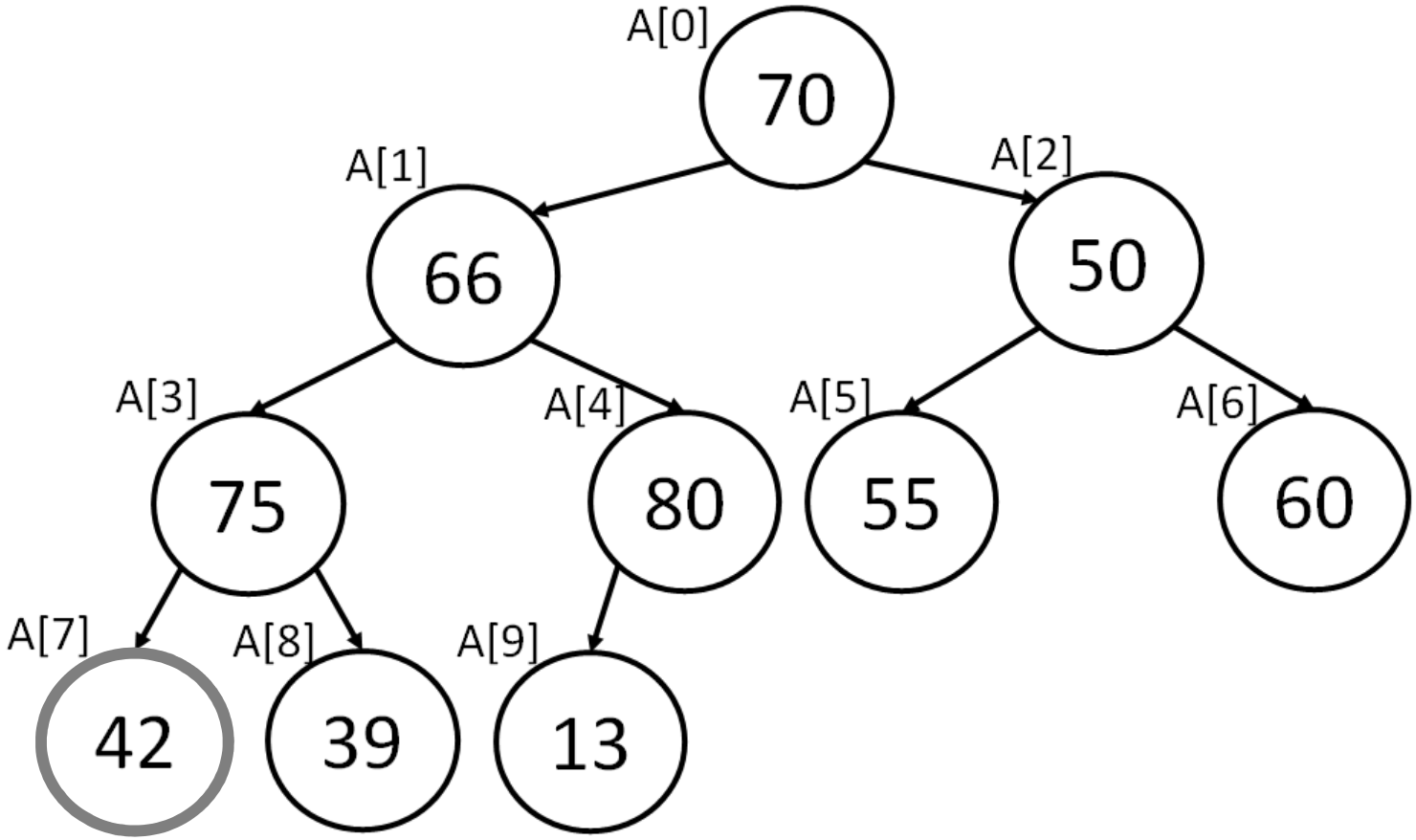
Heapifica(arranjo ***A, fim, i***)

1. ***e*** ← ***2*i + 1***
2. ***d*** ← ***2*i + 2***
3. Se ***e ≤ fim*** e ***A[e] > A[i]*** então
4. ***maior*** ← ***e***
5. Senão
6. ***maior*** ← ***i***
7. Se ***d ≤ fim*** e ***A[d] > A[maior]*** então
8. ***maior*** ← ***d***
9. Se ***maior ≠ i*** então
10. troca ***A[i]*** ↔ ***A[maior]***
11. ***Heapifica***(***A, fim, maior***)



Construindo a Heap: método Heapifica()

i	0	1	2	3	4	5	6	7	8	9
A[i]	70	66	50	75	80	55	60	42	39	13



Construindo a Heap: método ConstroiHeap()

i	0	1	2	3	4	5	6	7	8	9
A[i]	70	66	50	75	80	55	60	42	39	13

fim: 9

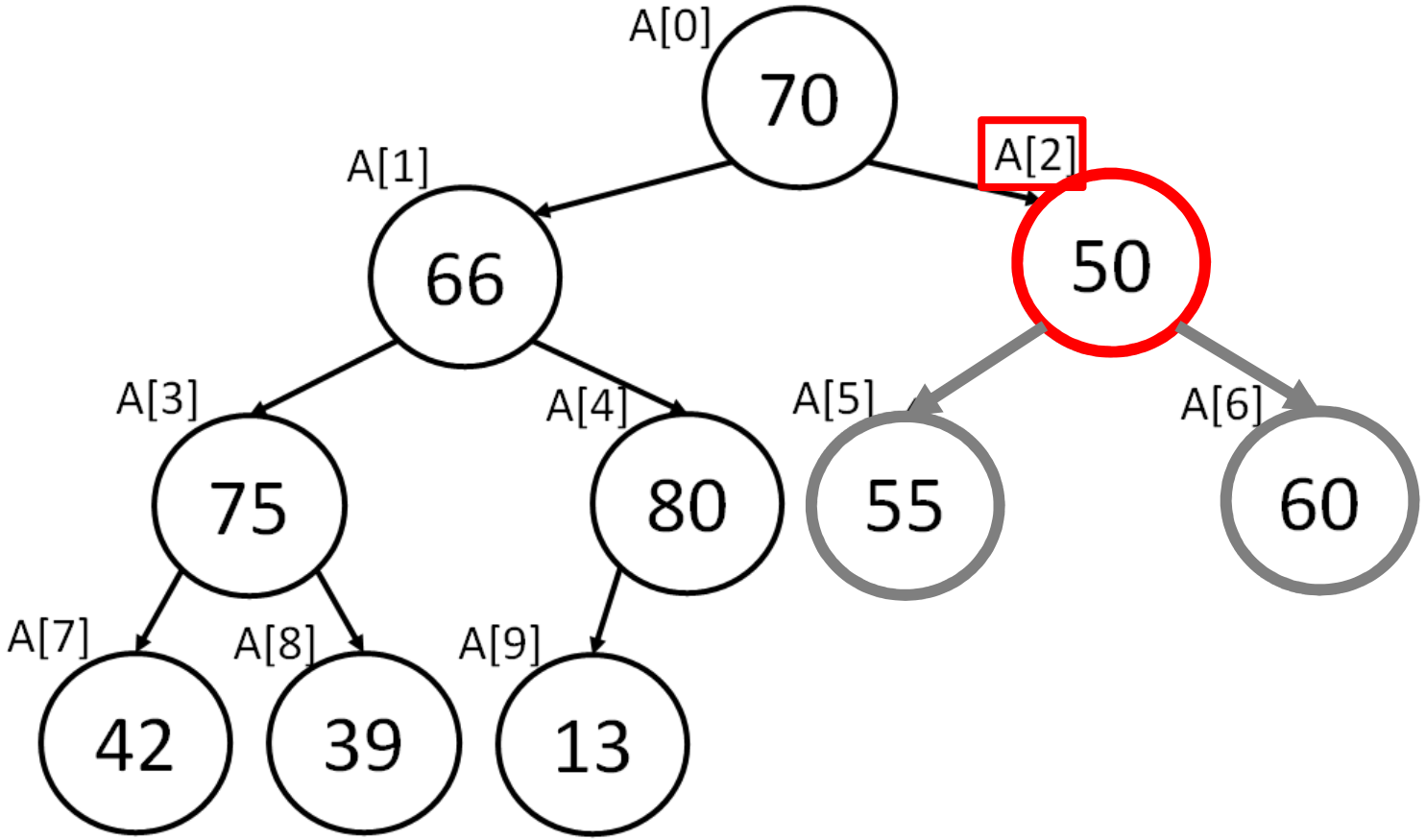
i: 2

ConstroiHeap(arranjo ***A, fim***)

1. Para ***i*** de ***fim /2*** até ***0*** faça
2. ***Heapifica***(***A, fim, i***)

Construindo a Heap: método Heapifica()

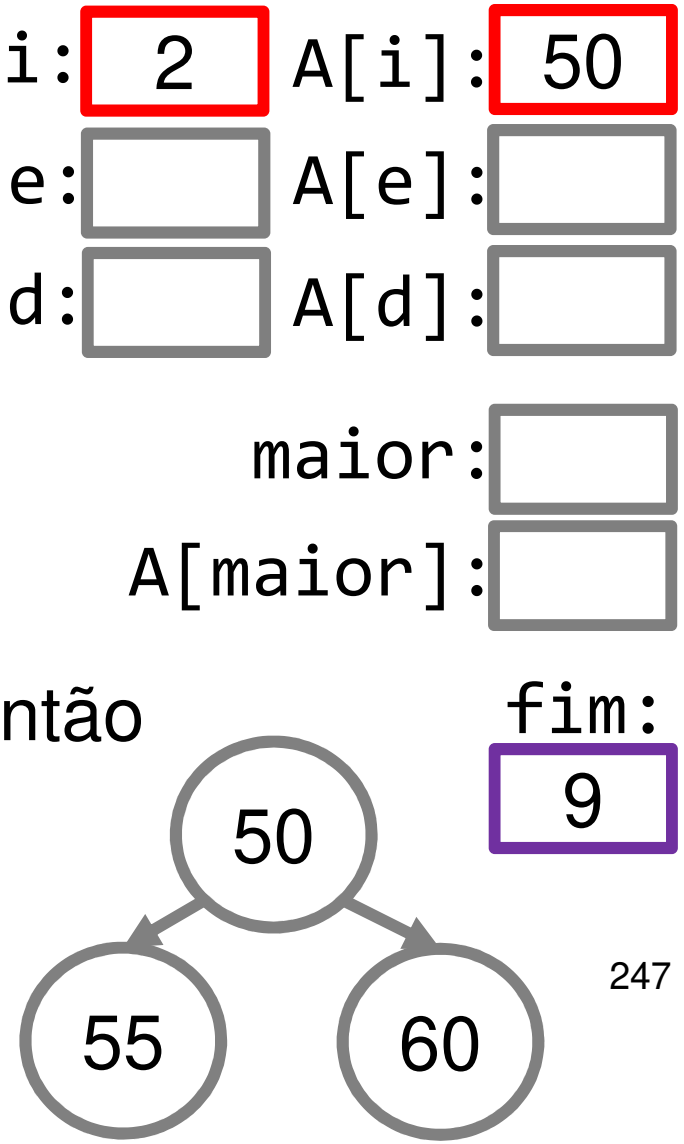
i	0	1	2	3	4	5	6	7	8	9
A[i]	70	66	50	75	80	55	60	42	39	13



Construindo a Heap: método Heapifica()

Heapifica(arranjo ***A, fim, i***)

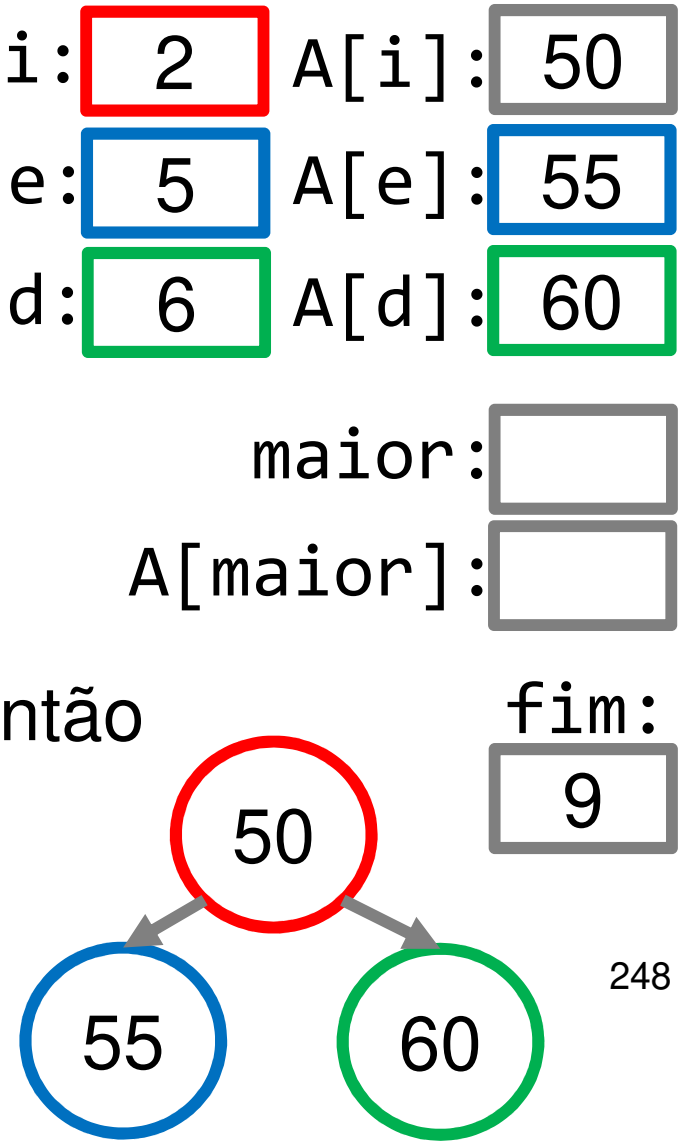
1. ***e*** ← ***2*i + 1***
2. ***d*** ← ***2*i + 2***
3. Se ***e ≤ fim*** e ***A[e] > A[i]*** então
4. ***maior*** ← ***e***
5. Senão
6. ***maior*** ← ***i***
7. Se ***d ≤ fim*** e ***A[d] > A[maior]*** então
8. ***maior*** ← ***d***
9. Se ***maior ≠ i*** então
10. troca ***A[i]*** ↔ ***A[maior]***
11. ***Heapifica***(***A, fim, maior***)



Construindo a Heap: método Heapifica()

Heapifica(arranjo ***A, fim, i***)

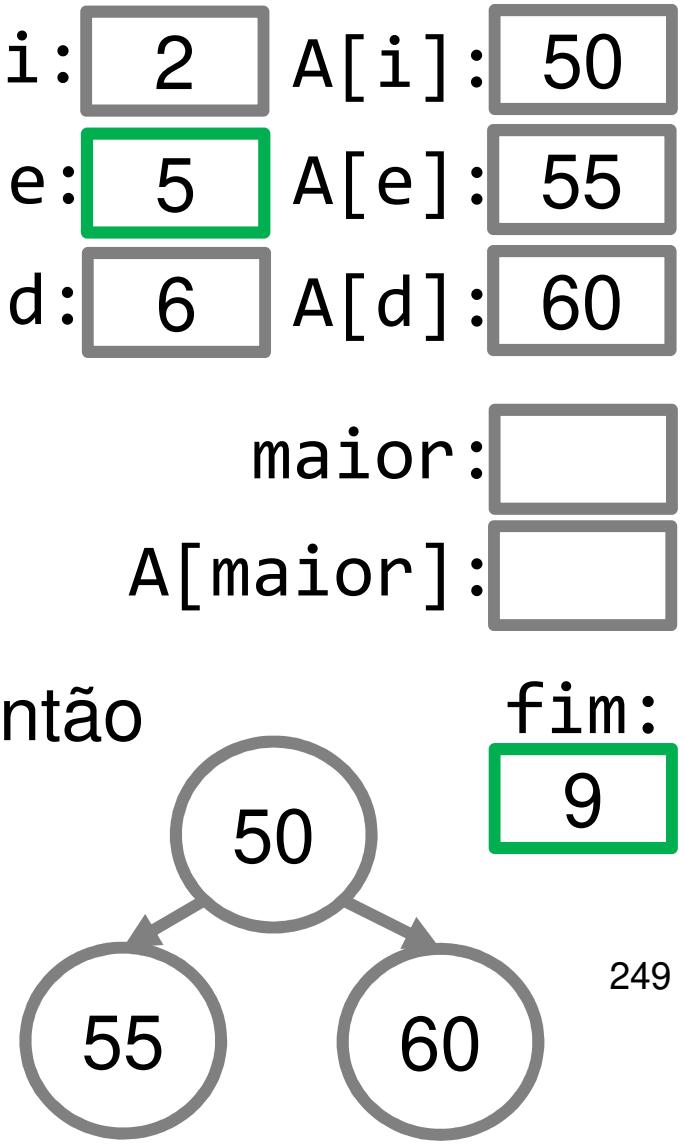
1. ***e*** ← ***2*i + 1***
2. ***d*** ← ***2*i + 2***
3. Se ***e ≤ fim*** e ***A[e] > A[i]*** então
4. ***maior*** ← ***e***
5. Senão
6. ***maior*** ← ***i***
7. Se ***d ≤ fim*** e ***A[d] > A[maior]*** então
8. ***maior*** ← ***d***
9. Se ***maior ≠ i*** então
10. troca ***A[i]*** ↔ ***A[maior]***
11. ***Heapifica***(***A, fim, maior***)



Construindo a Heap: método Heapifica()

Heapifica(arranjo ***A, fim, i***)

1. ***e*** ← ***2*i + 1***
2. ***d*** ← ***2*i + 2***
3. Se ***e ≤ fim*** e ***A[e] > A[i]*** então
4. ***maior*** ← ***e***
5. Senão
6. ***maior*** ← ***i***
7. Se ***d ≤ fim*** e ***A[d] > A[maior]*** então
8. ***maior*** ← ***d***
9. Se ***maior ≠ i*** então
10. troca ***A[i]*** ↔ ***A[maior]***
11. ***Heapifica***(***A, fim, maior***)

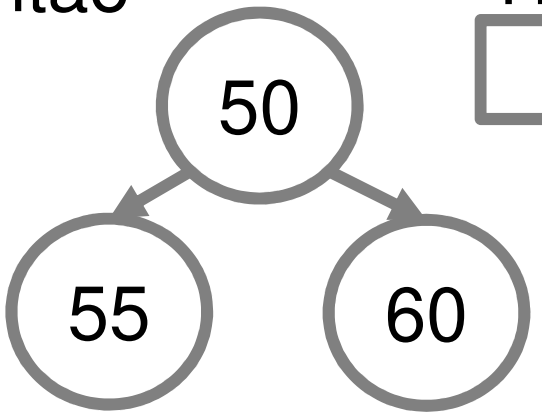


Construindo a Heap: método Heapifica()

Heapifica(arranjo ***A, fim, i***)

1. ***e*** ← ***2*i + 1***
2. ***d*** ← ***2*i + 2***
3. Se ***e ≤ fim*** e ***A[e] > A[i]*** então
4. ***maior*** ← ***e***
5. Senão
6. ***maior*** ← ***i***
7. Se ***d ≤ fim*** e ***A[d] > A[maior]*** então
8. ***maior*** ← ***d***
9. Se ***maior ≠ i*** então
10. troca ***A[i]*** ↔ ***A[maior]***
11. ***Heapifica***(***A, fim, maior***)

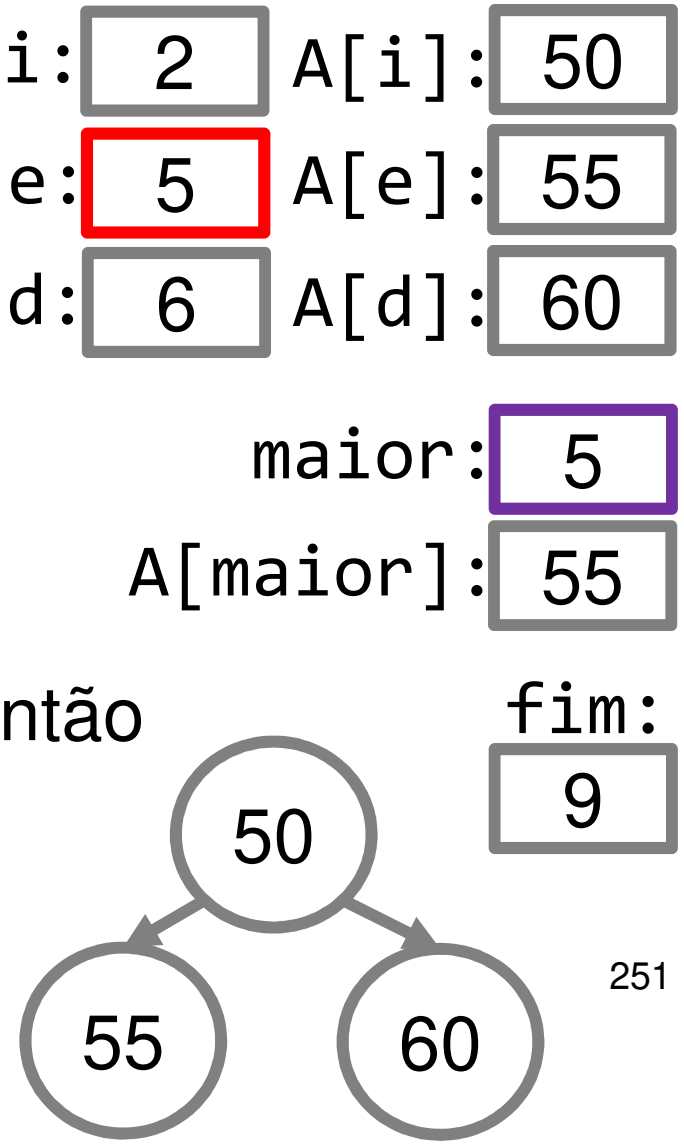
i:	2	A[i]:	50
e:	5	A[e]:	55
d:	6	A[d]:	60
maior:			
A[maior]:			
fim:	9		



Construindo a Heap: método Heapifica()

Heapifica(arranjo ***A, fim, i***)

1. ***e*** ← ***2*i + 1***
2. ***d*** ← ***2*i + 2***
3. Se ***e ≤ fim*** e ***A[e] > A[i]*** então
4. ***maior*** ← ***e***
5. Senão
6. ***maior*** ← ***i***
7. Se ***d ≤ fim*** e ***A[d] > A[maior]*** então
8. ***maior*** ← ***d***
9. Se ***maior ≠ i*** então
10. troca ***A[i]*** ↔ ***A[maior]***
11. ***Heapifica***(***A, fim, maior***)

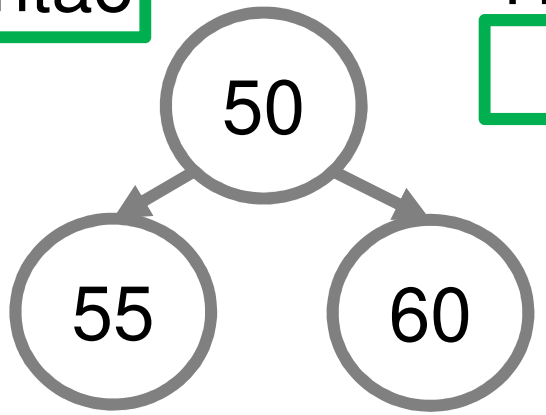


Construindo a Heap: método Heapifica()

Heapifica(arranjo ***A, fim, i***)

1. ***e*** ← ***2*i + 1***
2. ***d*** ← ***2*i + 2***
3. Se ***e ≤ fim*** e ***A[e] > A[i]*** então
4. ***maior*** ← ***e***
5. Senão
6. ***maior*** ← ***i***
7. Se ***d ≤ fim*** e ***A[d] > A[maior]*** então
8. ***maior*** ← ***d***
9. Se ***maior ≠ i*** então
10. troca ***A[i]*** ↔ ***A[maior]***
11. ***Heapifica***(***A, fim, maior***)

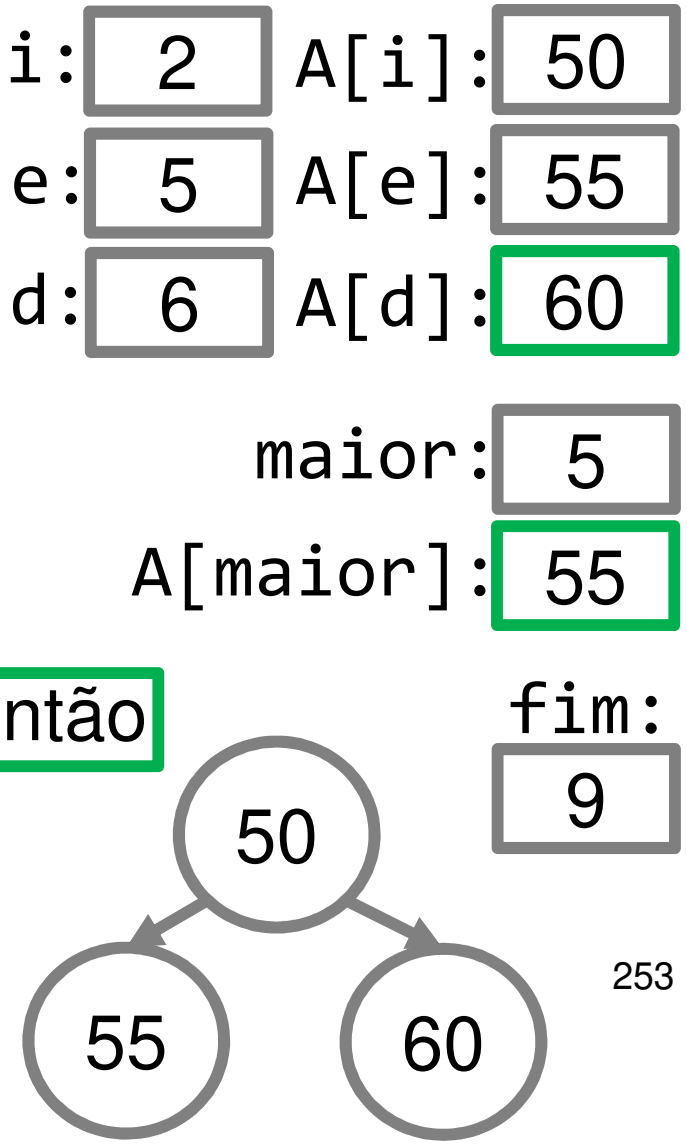
i:	2	A[i]:	50
e:	5	A[e]:	55
d:	6	A[d]:	60
maior:	5		
A[maior]:	55		
fim:	9		



Construindo a Heap: método Heapifica()

Heapifica(arranjo ***A, fim, i***)

1. ***e*** ← ***2*i + 1***
2. ***d*** ← ***2*i + 2***
3. Se ***e ≤ fim*** e ***A[e] > A[i]*** então
4. ***maior*** ← ***e***
5. Senão
6. ***maior*** ← ***i***
7. Se ***d ≤ fim*** e ***A[d] > A[maior]*** então
8. ***maior*** ← ***d***
9. Se ***maior ≠ i*** então
10. troca ***A[i]*** ↔ ***A[maior]***
11. ***Heapifica***(***A, fim, maior***)

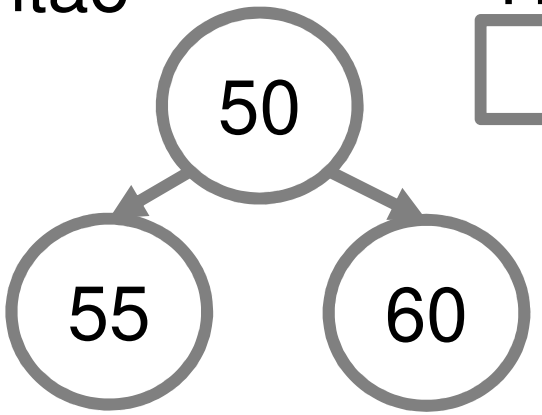


Construindo a Heap: método Heapifica()

Heapifica(arranjo ***A, fim, i***)

1. ***e*** ← ***2*i + 1***
2. ***d*** ← ***2*i + 2***
3. Se ***e ≤ fim*** e ***A[e] > A[i]*** então
4. ***maior*** ← ***e***
5. Senão
6. ***maior*** ← ***i***
7. Se ***d ≤ fim*** e ***A[d] > A[maior]*** então
8. ***maior*** ← ***d***
9. Se ***maior ≠ i*** então
10. troca ***A[i]*** ↔ ***A[maior]***
11. ***Heapifica***(***A, fim, maior***)

i:	2	A[i]:	50
e:	5	A[e]:	55
d:	6	A[d]:	60
maior:	6		
A[maior]:	60		
fim:	9		



Construindo a Heap: método Heapifica()

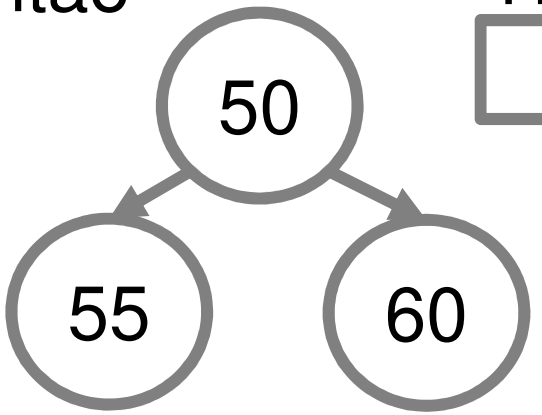
Heapifica(arranjo ***A, fim, i***)

1. ***e*** ← ***2*i + 1***
2. ***d*** ← ***2*i + 2***
3. Se ***e ≤ fim*** e ***A[e] > A[i]*** então
4. ***maior*** ← ***e***
5. Senão
6. ***maior*** ← ***i***
7. Se ***d ≤ fim*** e ***A[d] > A[maior]*** então
8. ***maior*** ← ***d***
9. Se ***maior ≠ i*** então
10. troca ***A[i]*** ↔ ***A[maior]***
11. ***Heapifica***(***A, fim, maior***)

i:	2	A[i]:	50
e:	5	A[e]:	55
d:	6	A[d]:	60

maior:	6
A[maior]:	60

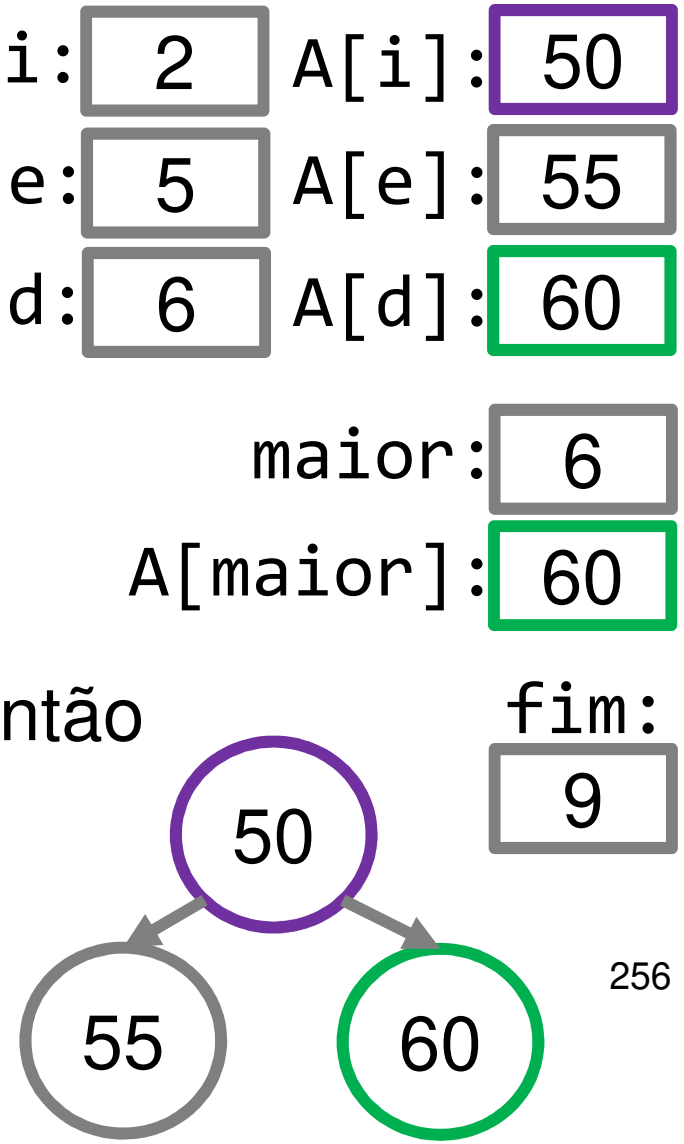
fim: 9



Construindo a Heap: método Heapifica()

Heapifica(arranjo ***A, fim, i***)

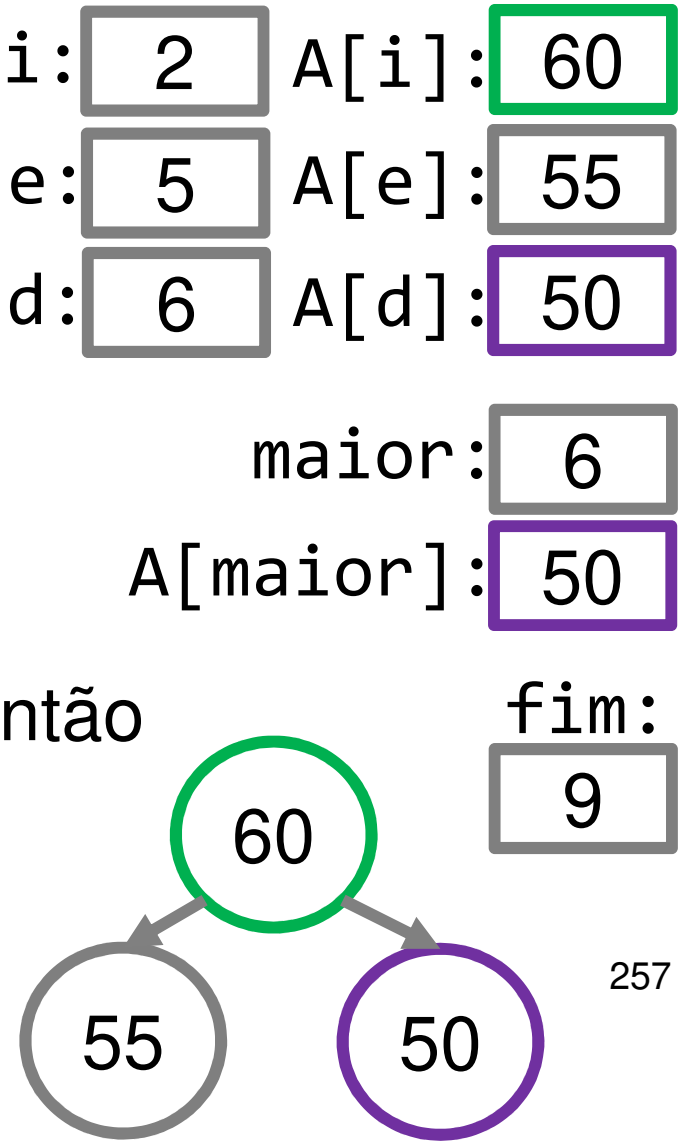
1. ***e*** ← ***2*i + 1***
2. ***d*** ← ***2*i + 2***
3. Se ***e ≤ fim*** e ***A[e] > A[i]*** então
4. ***maior*** ← ***e***
5. Senão
6. ***maior*** ← ***i***
7. Se ***d ≤ fim*** e ***A[d] > A[maior]*** então
8. ***maior*** ← ***d***
9. Se ***maior ≠ i*** então
10. troca ***A[i]*** ↔ ***A[maior]***
11. ***Heapifica***(***A, fim, maior***)



Construindo a Heap: método Heapifica()

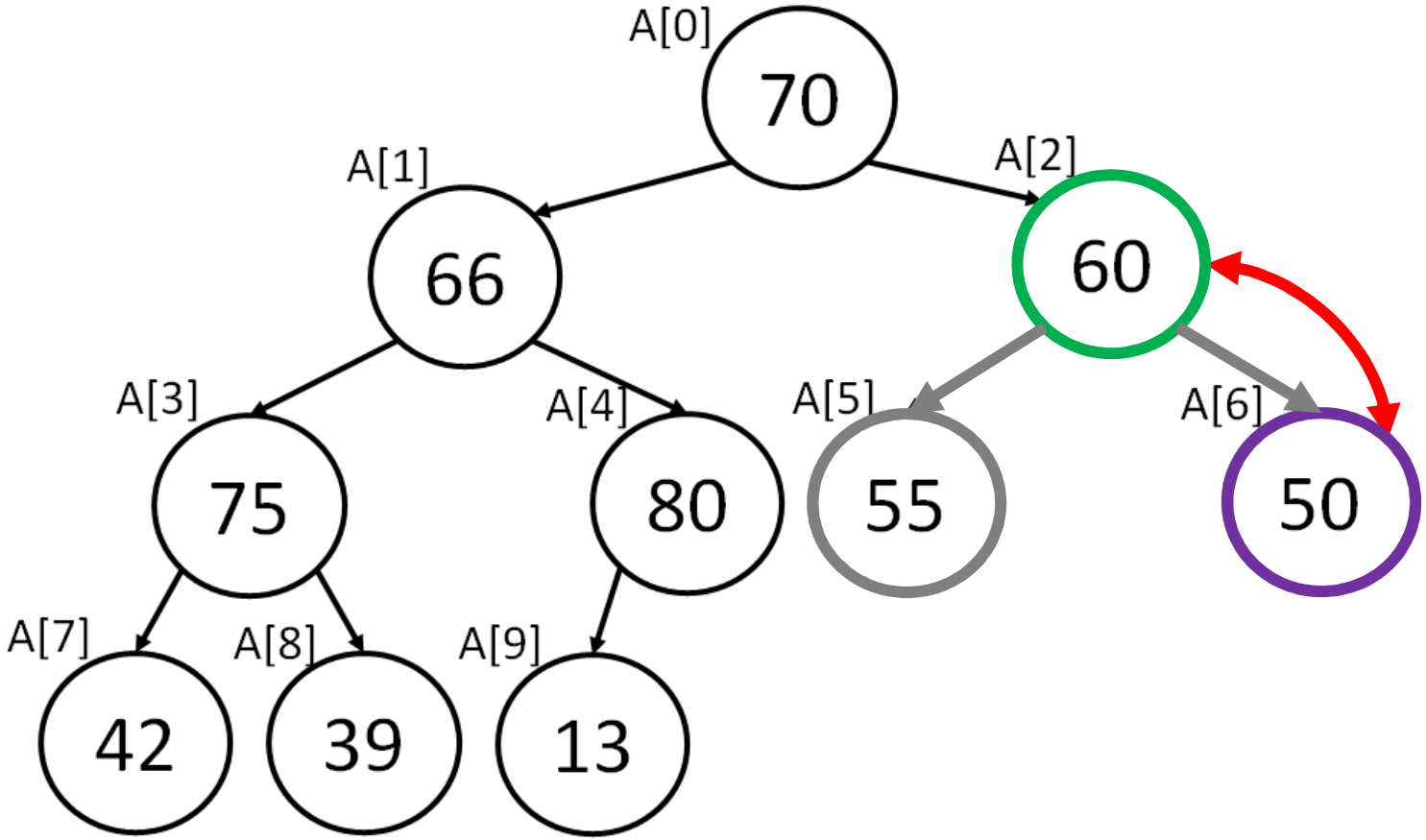
Heapifica(arranjo ***A, fim, i***)

1. ***e*** ← ***2*i + 1***
2. ***d*** ← ***2*i + 2***
3. Se ***e ≤ fim*** e ***A[e] > A[i]*** então
4. ***maior*** ← ***e***
5. Senão
6. ***maior*** ← ***i***
7. Se ***d ≤ fim*** e ***A[d] > A[maior]*** então
8. ***maior*** ← ***d***
9. Se ***maior ≠ i*** então
10. troca ***A[i]*** ↔ ***A[maior]***
11. ***Heapifica***(***A, fim, maior***)



Construindo a Heap: método Heapifica()

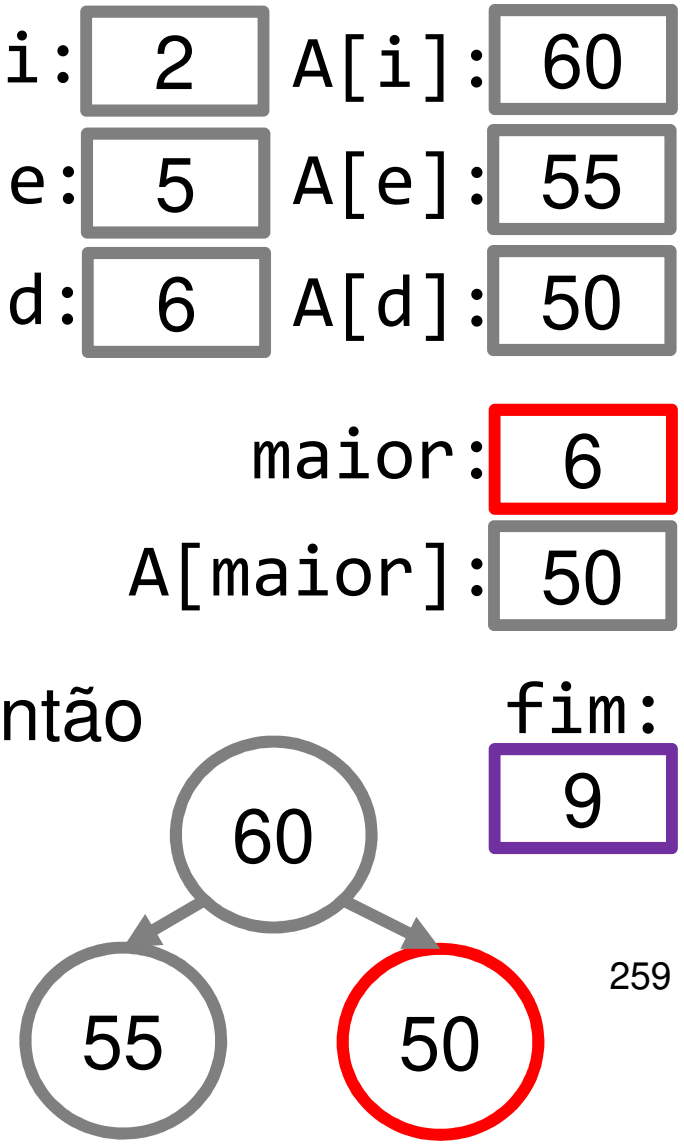
i	0	1	2	3	4	5	6	7	8	9
A[i]	70	66	60	75	80	55	50	42	39	13



Construindo a Heap: método Heapifica()

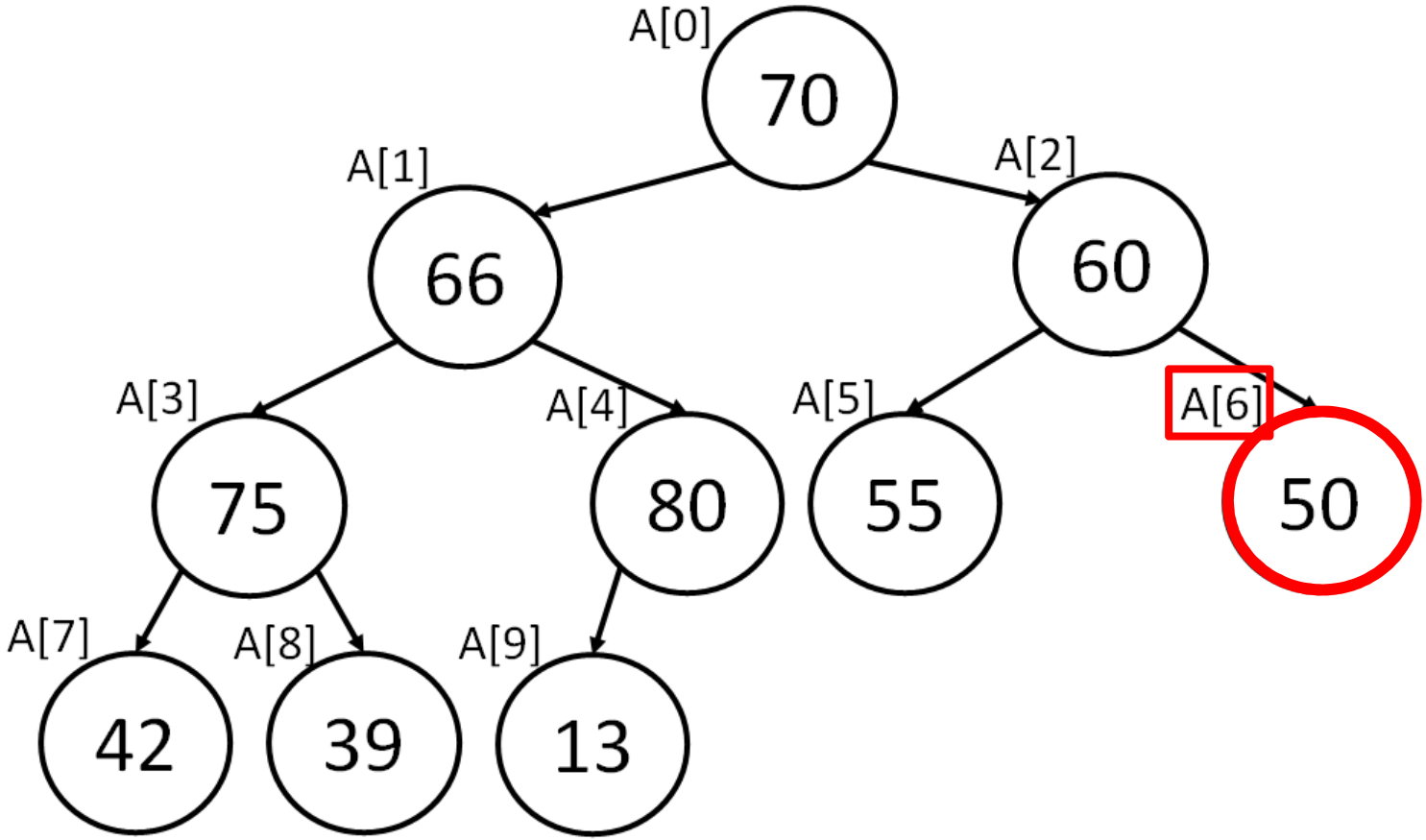
Heapifica(arranjo ***A, fim, i***)

1. ***e*** ← ***2*i + 1***
2. ***d*** ← ***2*i + 2***
3. Se ***e ≤ fim*** e ***A[e] > A[i]*** então
4. ***maior*** ← ***e***
5. Senão
6. ***maior*** ← ***i***
7. Se ***d ≤ fim*** e ***A[d] > A[maior]*** então
8. ***maior*** ← ***d***
9. Se ***maior ≠ i*** então
10. troca ***A[i]*** ↔ ***A[maior]***
11. ***Heapifica***(***A, fim, maior***)



Construindo a Heap: método Heapifica()

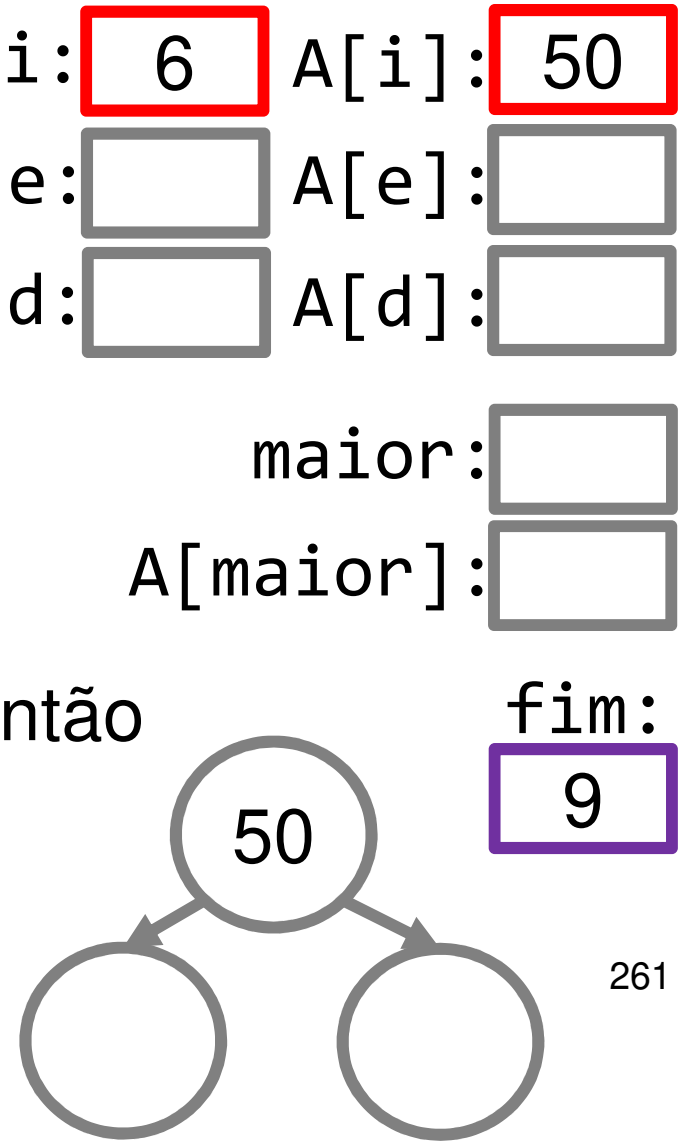
i	0	1	2	3	4	5	6	7	8	9
A[i]	70	66	60	75	80	55	50	42	39	13



Construindo a Heap: método Heapifica()

Heapifica(arranjo ***A***, ***fim***, ***i***)

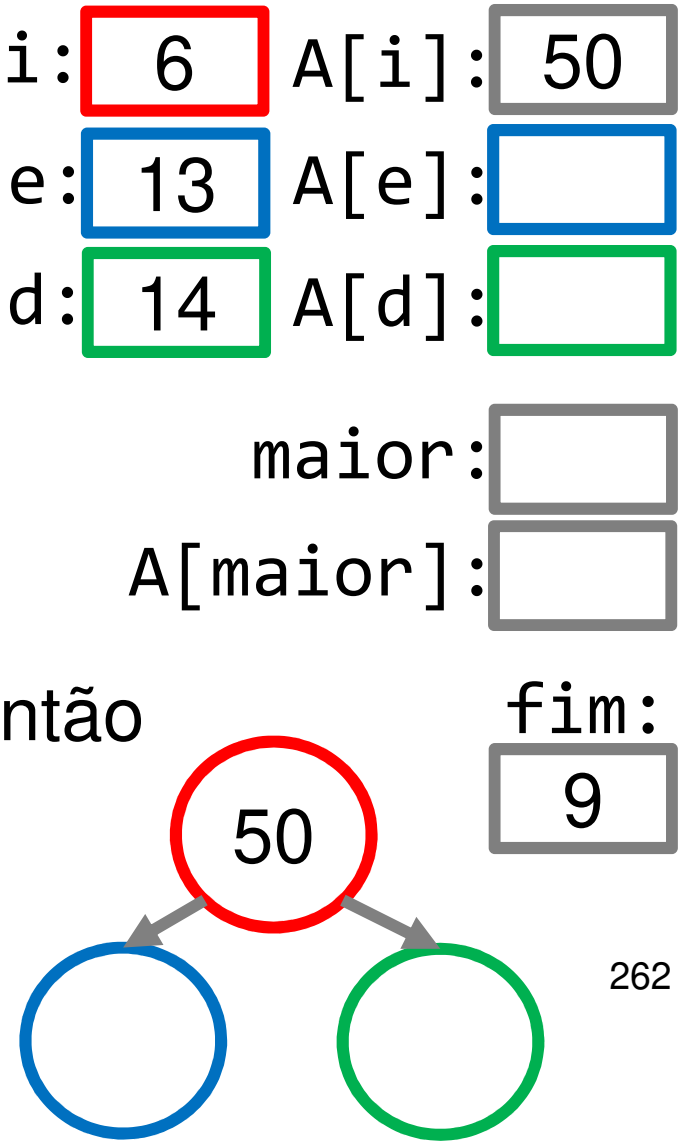
1. ***e*** ← ***2*i + 1***
2. ***d*** ← ***2*i + 2***
3. Se ***e*** ≤ ***fim*** e ***A[e]*** > ***A[i]*** então
4. ***maior*** ← ***e***
5. Senão
6. ***maior*** ← ***i***
7. Se ***d*** ≤ ***fim*** e ***A[d]*** > ***A[maior]*** então
8. ***maior*** ← ***d***
9. Se ***maior*** ≠ ***i*** então
10. troca ***A[i]*** ↔ ***A[maior]***
11. ***Heapifica***(***A, fim, maior***)



Construindo a Heap: método Heapifica()

Heapifica(arranjo ***A, fim, i***)

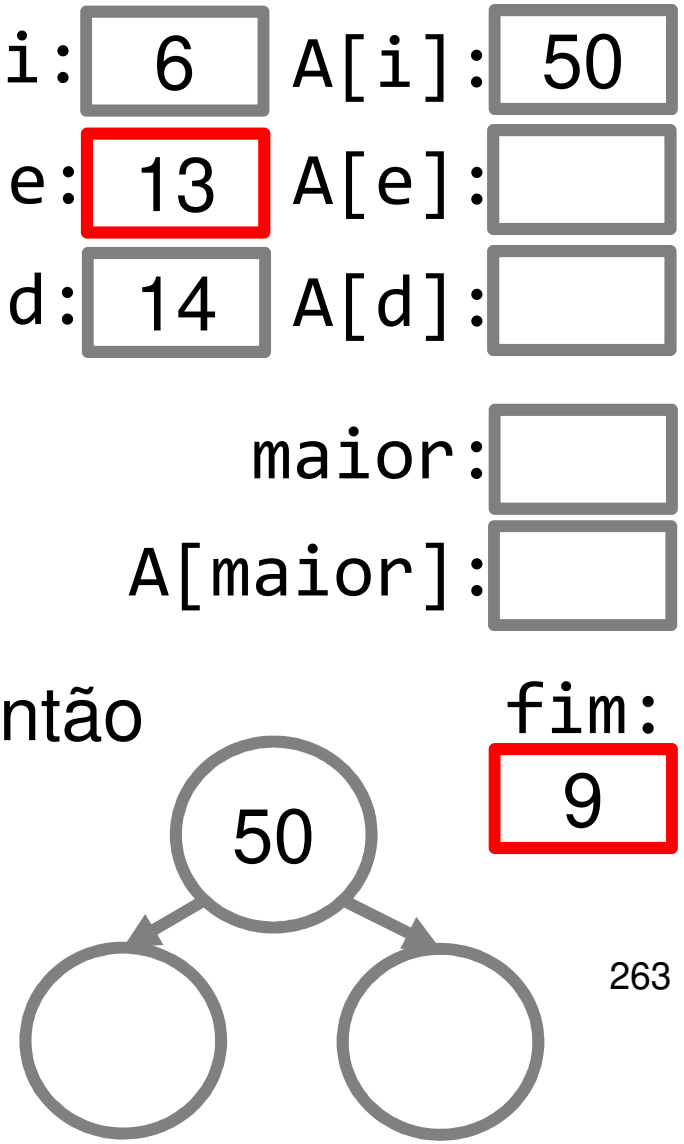
1. ***e*** ← ***2*i + 1***
2. ***d*** ← ***2*i + 2***
3. Se ***e ≤ fim*** e ***A[e] > A[i]*** então
4. ***maior*** ← ***e***
5. Senão
6. ***maior*** ← ***i***
7. Se ***d ≤ fim*** e ***A[d] > A[maior]*** então
8. ***maior*** ← ***d***
9. Se ***maior ≠ i*** então
10. troca ***A[i]*** ↔ ***A[maior]***
11. ***Heapifica***(***A, fim, maior***)



Construindo a Heap: método Heapifica()

Heapifica(arranjo ***A, fim, i***)

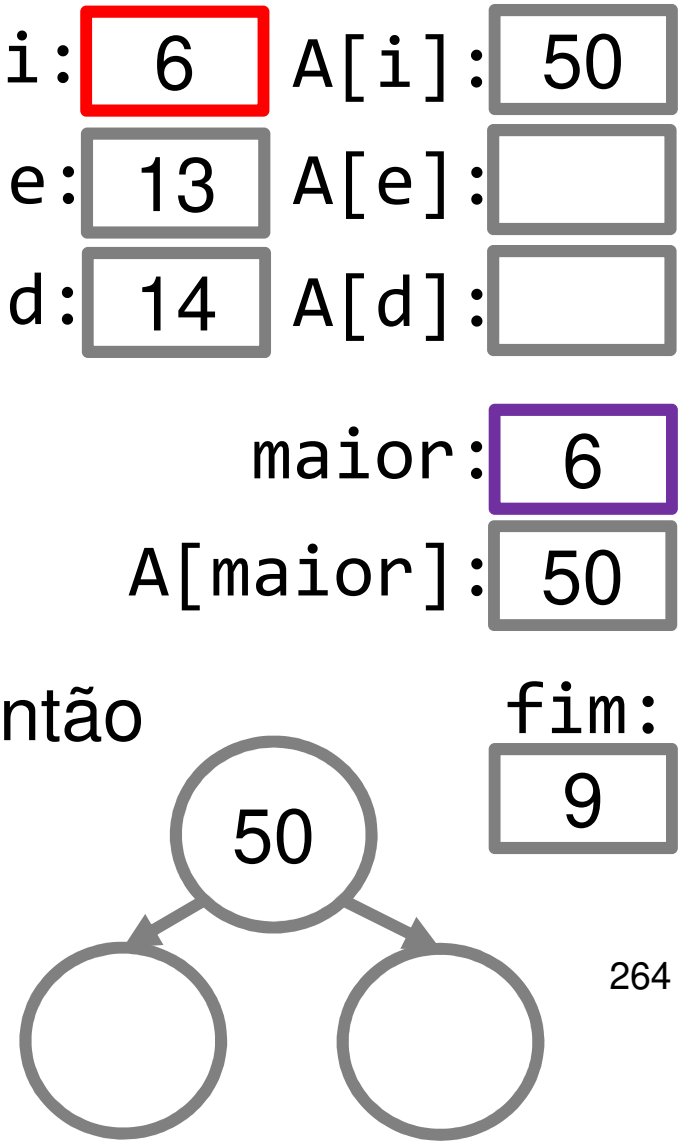
1. ***e*** ← ***2*i + 1***
2. ***d*** ← ***2*i + 2***
3. Se ***e ≤ fim*** e ***A[e] > A[i]*** então
4. ***maior*** ← ***e***
5. Senão
6. ***maior*** ← ***i***
7. Se ***d ≤ fim*** e ***A[d] > A[maior]*** então
8. ***maior*** ← ***d***
9. Se ***maior ≠ i*** então
10. troca ***A[i]*** ↔ ***A[maior]***
11. ***Heapifica***(***A, fim, maior***)



Construindo a Heap: método Heapifica()

Heapifica(arranjo ***A, fim, i***)

1. ***e*** ← ***2*i + 1***
2. ***d*** ← ***2*i + 2***
3. Se ***e ≤ fim*** e ***A[e] > A[i]*** então
4. ***maior*** ← ***e***
5. Senão
6. ***maior*** ← ***i***
7. Se ***d ≤ fim*** e ***A[d] > A[maior]*** então
8. ***maior*** ← ***d***
9. Se ***maior ≠ i*** então
10. troca ***A[i]*** ↔ ***A[maior]***
11. ***Heapifica***(***A, fim, maior***)

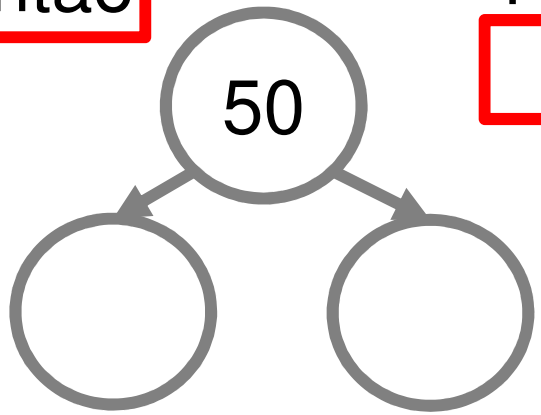


Construindo a Heap: método Heapifica()

Heapifica(arranjo ***A, fim, i***)

1. ***e*** ← ***2*i + 1***
2. ***d*** ← ***2*i + 2***
3. Se ***e ≤ fim*** e ***A[e] > A[i]*** então
4. ***maior*** ← ***e***
5. Senão
6. ***maior*** ← ***i***
7. Se ***d ≤ fim*** e ***A[d] > A[maior]*** então
8. ***maior*** ← ***d***
9. Se ***maior ≠ i*** então
10. troca ***A[i]*** ↔ ***A[maior]***
11. ***Heapifica***(***A, fim, maior***)

i:	6	A[i]:	50
e:	13	A[e]:	
d:	14	A[d]:	
maior:	6		
A[maior]:	50		
fim:	9		

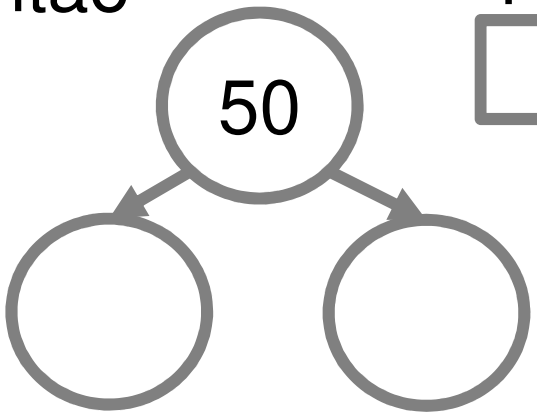


Construindo a Heap: método Heapifica()

Heapifica(arranjo ***A, fim, i***)

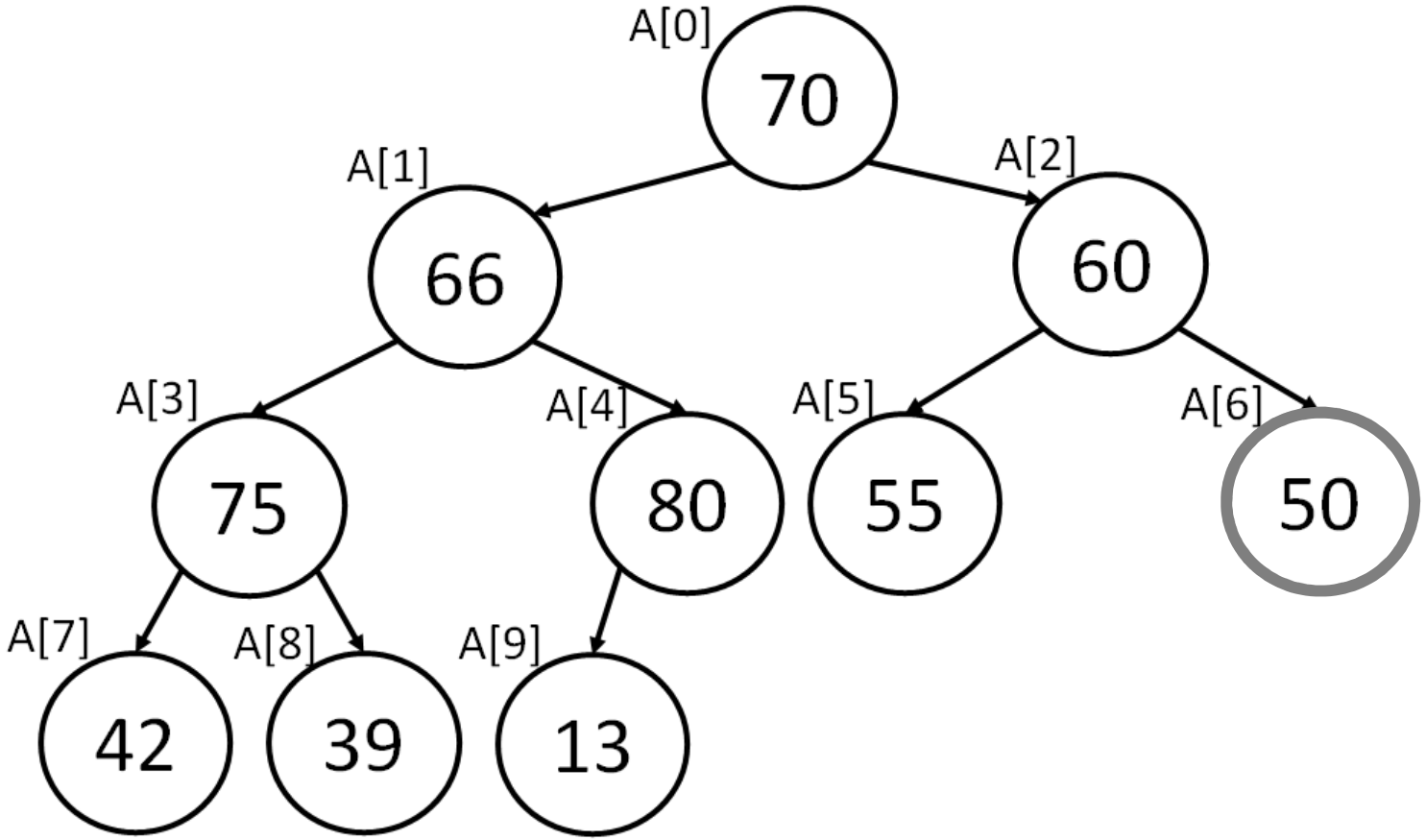
1. ***e*** ← ***2*i + 1***
2. ***d*** ← ***2*i + 2***
3. Se ***e*** ≤ ***fim*** e ***A[e] > A[i]*** então
4. ***maior*** ← ***e***
5. Senão
6. ***maior*** ← ***i***
7. Se ***d*** ≤ ***fim*** e ***A[d] > A[maior]*** então
8. ***maior*** ← ***d***
9. Se ***maior*** ≠ ***i*** então
10. troca ***A[i]*** ↔ ***A[maior]***
11. ***Heapifica***(***A, fim, maior***)

i:	6	A[i]:	50
e:	13	A[e]:	
d:	14	A[d]:	
maior:	6		
A[maior]:	50		
fim:	9		



Construindo a Heap: método Heapifica()

i	0	1	2	3	4	5	6	7	8	9
A[i]	70	66	60	75	80	55	50	42	39	13



Construindo a Heap: método ConstroiHeap()

i	0	1	2	3	4	5	6	7	8	9
A[i]	70	66	60	75	80	55	50	42	39	13

fim: 9

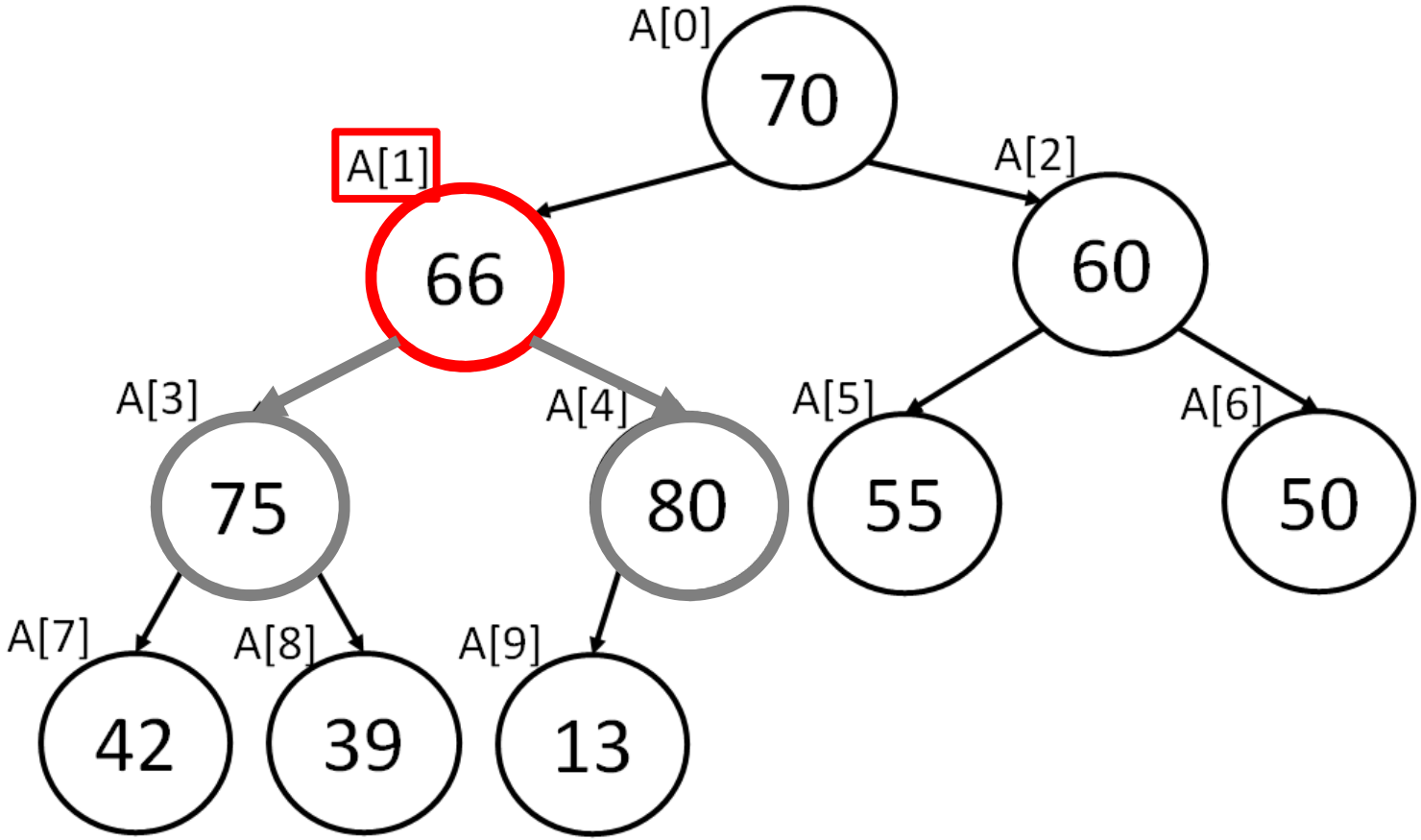
i: 1

ConstroiHeap(arranjo ***A, fim***)

1. Para ***i*** de ***fim /2*** até ***0*** faça
2. ***Heapifica***(***A, fim, i***)

Construindo a Heap: método Heapifica()

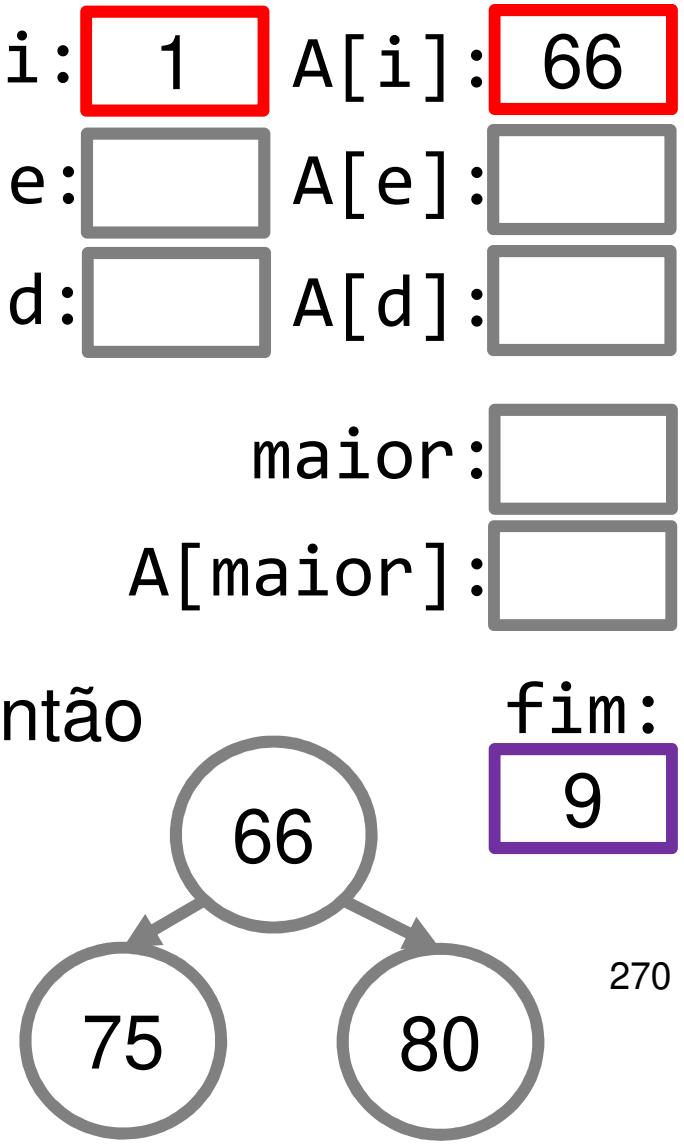
i	0	1	2	3	4	5	6	7	8	9
A[i]	70	66	60	75	80	55	50	42	39	13



Construindo a Heap: método Heapifica()

Heapifica(arranjo ***A, fim, i***)

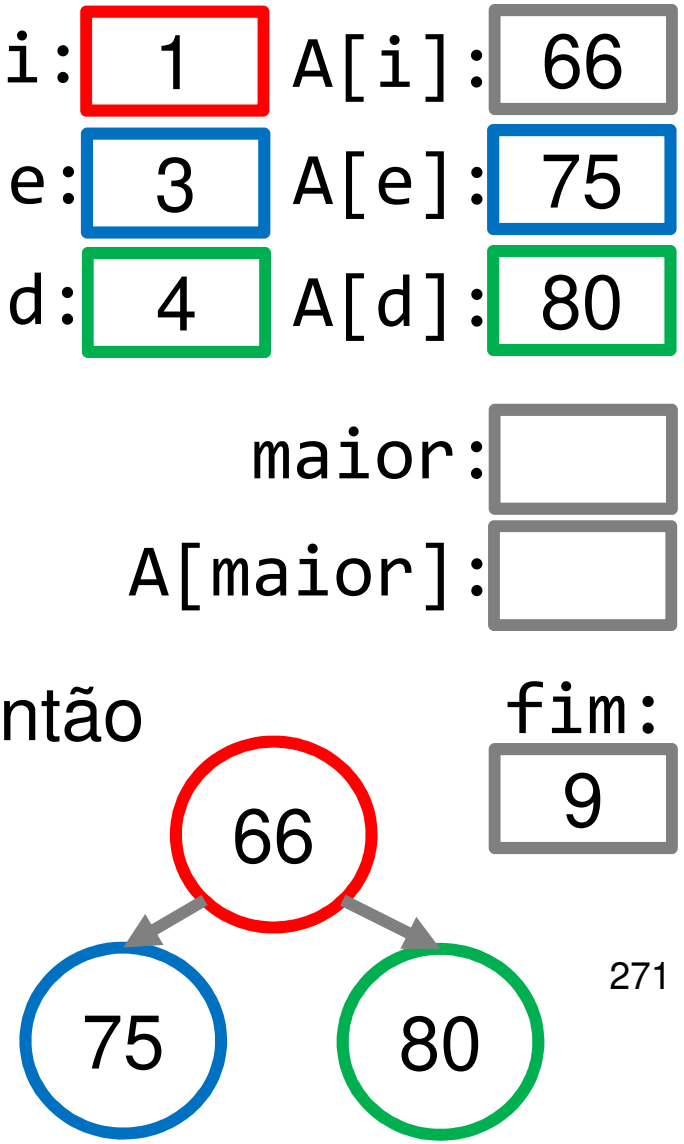
1. ***e*** ← ***2*i + 1***
2. ***d*** ← ***2*i + 2***
3. Se ***e ≤ fim*** e ***A[e] > A[i]*** então
4. ***maior*** ← ***e***
5. Senão
6. ***maior*** ← ***i***
7. Se ***d ≤ fim*** e ***A[d] > A[maior]*** então
8. ***maior*** ← ***d***
9. Se ***maior ≠ i*** então
10. troca ***A[i]*** ↔ ***A[maior]***
11. ***Heapifica***(***A, fim, maior***)



Construindo a Heap: método Heapifica()

Heapifica(arranjo ***A, fim, i***)

1. ***e*** ← ***2*i + 1***
2. ***d*** ← ***2*i + 2***
3. Se ***e ≤ fim*** e ***A[e] > A[i]*** então
4. ***maior*** ← ***e***
5. Senão
6. ***maior*** ← ***i***
7. Se ***d ≤ fim*** e ***A[d] > A[maior]*** então
8. ***maior*** ← ***d***
9. Se ***maior ≠ i*** então
10. troca ***A[i]*** ↔ ***A[maior]***
11. ***Heapifica***(***A, fim, maior***)

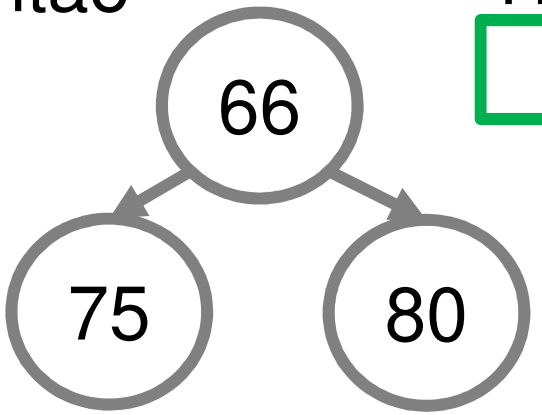


Construindo a Heap: método Heapifica()

```
Heapifica(arranjo A, fim, i)
1.  e ← 2*i + 1
2.  d ← 2*i + 2
3.  Se e ≤ fim e A[e] > A[i] então
4.      maior ← e
5.      Senão
6.          maior ← i
7.  Se d ≤ fim e A[d] > A[maior] então
8.      maior ← d
9.  Se maior ≠ i então
10.     troca A[i] ↔ A[maior]
11.     Heapifica(A, fim, maior)
```

i:	1	A[i]:	66
e:	3	A[e]:	75
d:	4	A[d]:	80
maior:			
A[maior]:			

fim: 9

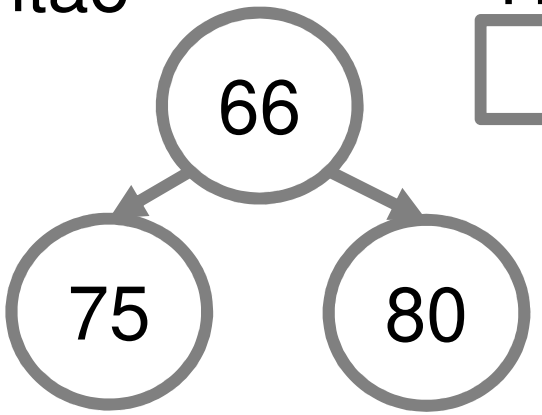


Construindo a Heap: método Heapifica()

Heapifica(arranjo ***A, fim, i***)

1. ***e*** ← ***2*i + 1***
2. ***d*** ← ***2*i + 2***
3. Se ***e ≤ fim*** e ***A[e] > A[i]*** então
4. ***maior*** ← ***e***
5. Senão
6. ***maior*** ← ***i***
7. Se ***d ≤ fim*** e ***A[d] > A[maior]*** então
8. ***maior*** ← ***d***
9. Se ***maior ≠ i*** então
10. troca ***A[i]*** ↔ ***A[maior]***
11. ***Heapifica***(***A, fim, maior***)

i:	1	A[i]:	66
e:	3	A[e]:	75
d:	4	A[d]:	80
maior:			
A[maior]:			
fim:	9		



Construindo a Heap: método Heapifica()

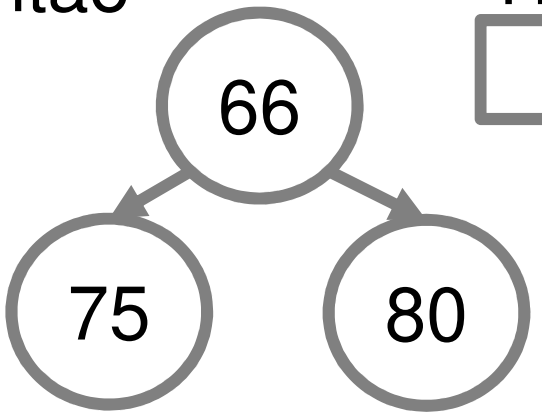
Heapifica(arranjo ***A, fim, i***)

1. ***e*** ← ***2*i + 1***
2. ***d*** ← ***2*i + 2***
3. Se ***e ≤ fim*** e ***A[e] > A[i]*** então
4. ***maior*** ← ***e***
5. Senão
6. ***maior*** ← ***i***
7. Se ***d ≤ fim*** e ***A[d] > A[maior]*** então
8. ***maior*** ← ***d***
9. Se ***maior ≠ i*** então
10. troca ***A[i]*** ↔ ***A[maior]***
11. ***Heapifica***(***A, fim, maior***)

i:	1	A[i]:	66
e:	3	A[e]:	75
d:	4	A[d]:	80

maior:	3
A[maior]:	75

fim: 9

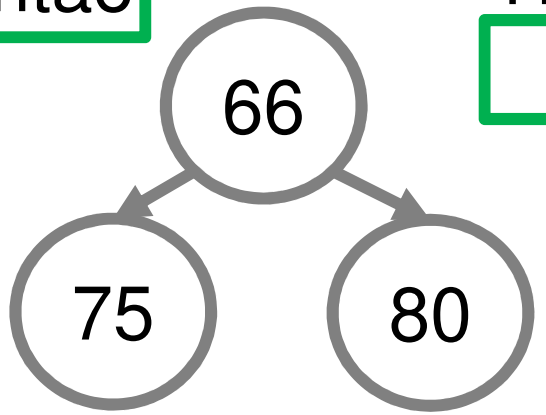


Construindo a Heap: método Heapifica()

Heapifica(arranjo ***A, fim, i***)

1. ***e*** ← ***2*i + 1***
2. ***d*** ← ***2*i + 2***
3. Se ***e ≤ fim*** e ***A[e] > A[i]*** então
4. ***maior*** ← ***e***
5. Senão
6. ***maior*** ← ***i***
7. Se ***d ≤ fim*** e ***A[d] > A[maior]*** então
8. ***maior*** ← ***d***
9. Se ***maior ≠ i*** então
10. troca ***A[i]*** ↔ ***A[maior]***
11. ***Heapifica***(***A, fim, maior***)

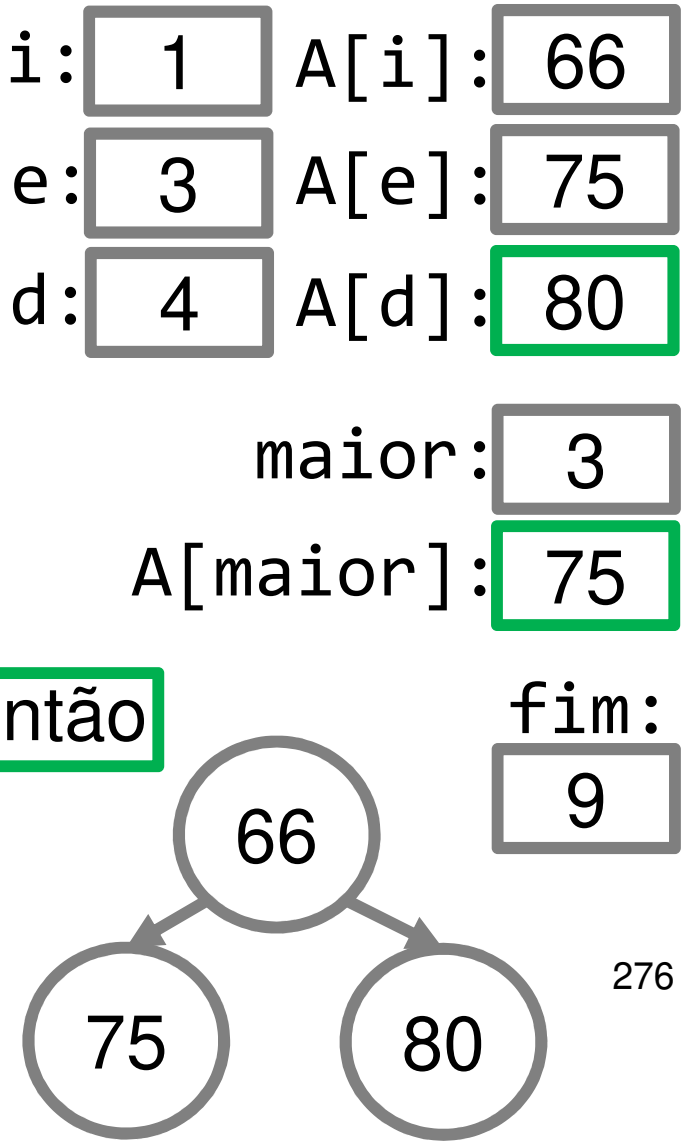
i:	1	A[i]:	66
e:	3	A[e]:	75
d:	4	A[d]:	80
maior:	3		
A[maior]:	75		
fim:	9		



Construindo a Heap: método Heapifica()

Heapifica(arranjo ***A, fim, i***)

1. ***e*** ← ***2*i + 1***
2. ***d*** ← ***2*i + 2***
3. Se ***e ≤ fim*** e ***A[e] > A[i]*** então
4. ***maior*** ← ***e***
5. Senão
6. ***maior*** ← ***i***
7. Se ***d ≤ fim*** e ***A[d] > A[maior]*** então
8. ***maior*** ← ***d***
9. Se ***maior ≠ i*** então
10. troca ***A[i]*** ↔ ***A[maior]***
11. ***Heapifica***(***A, fim, maior***)

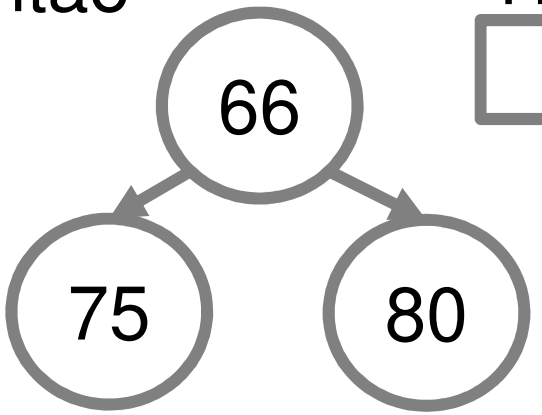


Construindo a Heap: método Heapifica()

Heapifica(arranjo ***A, fim, i***)

1. ***e*** ← ***2*i + 1***
2. ***d*** ← ***2*i + 2***
3. Se ***e ≤ fim*** e ***A[e] > A[i]*** então
4. ***maior*** ← ***e***
5. Senão
6. ***maior*** ← ***i***
7. Se ***d ≤ fim*** e ***A[d] > A[maior]*** então
8. ***maior*** ← ***d***
9. Se ***maior ≠ i*** então
10. troca ***A[i]*** ↔ ***A[maior]***
11. ***Heapifica***(***A, fim, maior***)

i:	1	A[i]:	66
e:	3	A[e]:	75
d:	4	A[d]:	80
maior:	4		
A[maior]:	80		
fim:	9		



Construindo a Heap: método Heapifica()

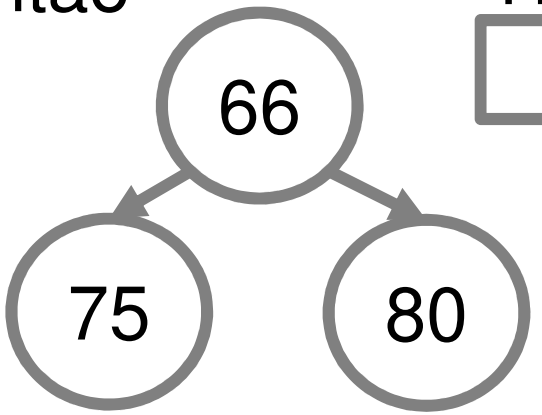
Heapifica(arranjo ***A, fim, i***)

1. ***e*** ← ***2*i + 1***
2. ***d*** ← ***2*i + 2***
3. Se ***e*** ≤ ***fim*** e ***A[e]*** > ***A[i]*** então
4. ***maior*** ← ***e***
5. Senão
6. ***maior*** ← ***i***
7. Se ***d*** ≤ ***fim*** e ***A[d]*** > ***A[maior]*** então
8. ***maior*** ← ***d***
9. Se ***maior*** ≠ ***i*** então
10. troca ***A[i]*** ↔ ***A[maior]***
11. ***Heapifica***(***A, fim, maior***)

i:	1	A[i]:	66
e:	3	A[e]:	75
d:	4	A[d]:	80

maior:	4
A[maior]:	80

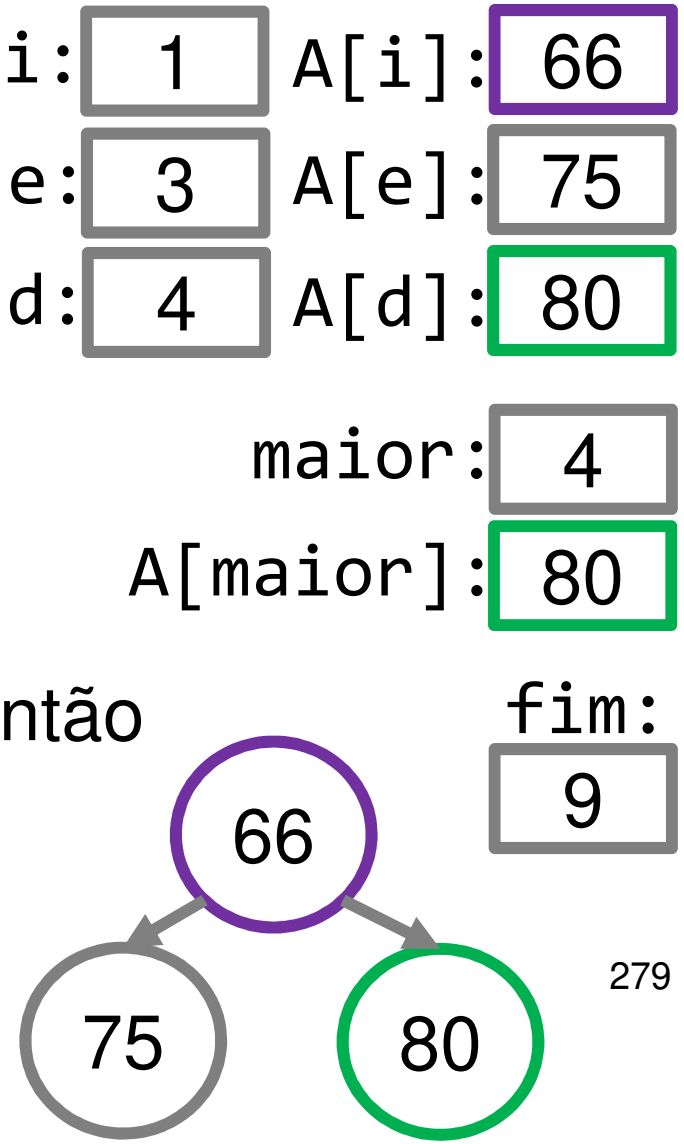
fim: 9



Construindo a Heap: método Heapifica()

Heapifica(arranjo ***A, fim, i***)

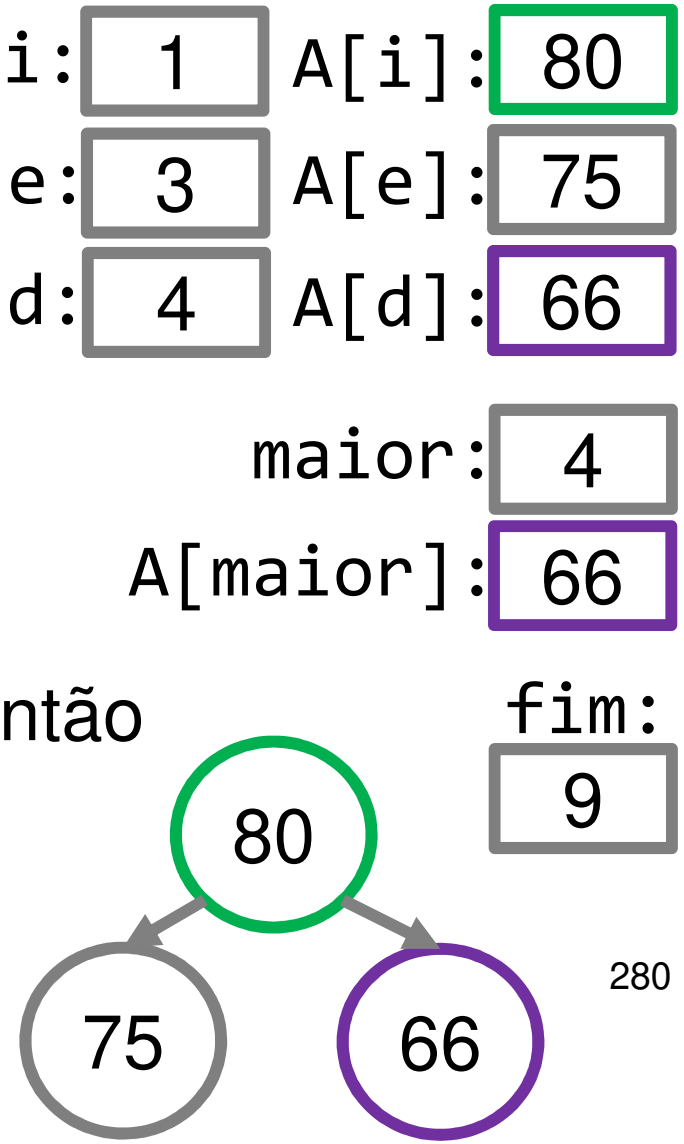
1. ***e*** ← ***2*i + 1***
2. ***d*** ← ***2*i + 2***
3. Se ***e ≤ fim*** e ***A[e] > A[i]*** então
4. ***maior*** ← ***e***
5. Senão
6. ***maior*** ← ***i***
7. Se ***d ≤ fim*** e ***A[d] > A[maior]*** então
8. ***maior*** ← ***d***
9. Se ***maior ≠ i*** então
10. troca ***A[i]*** ↔ ***A[maior]***
11. ***Heapifica***(***A, fim, maior***)



Construindo a Heap: método Heapifica()

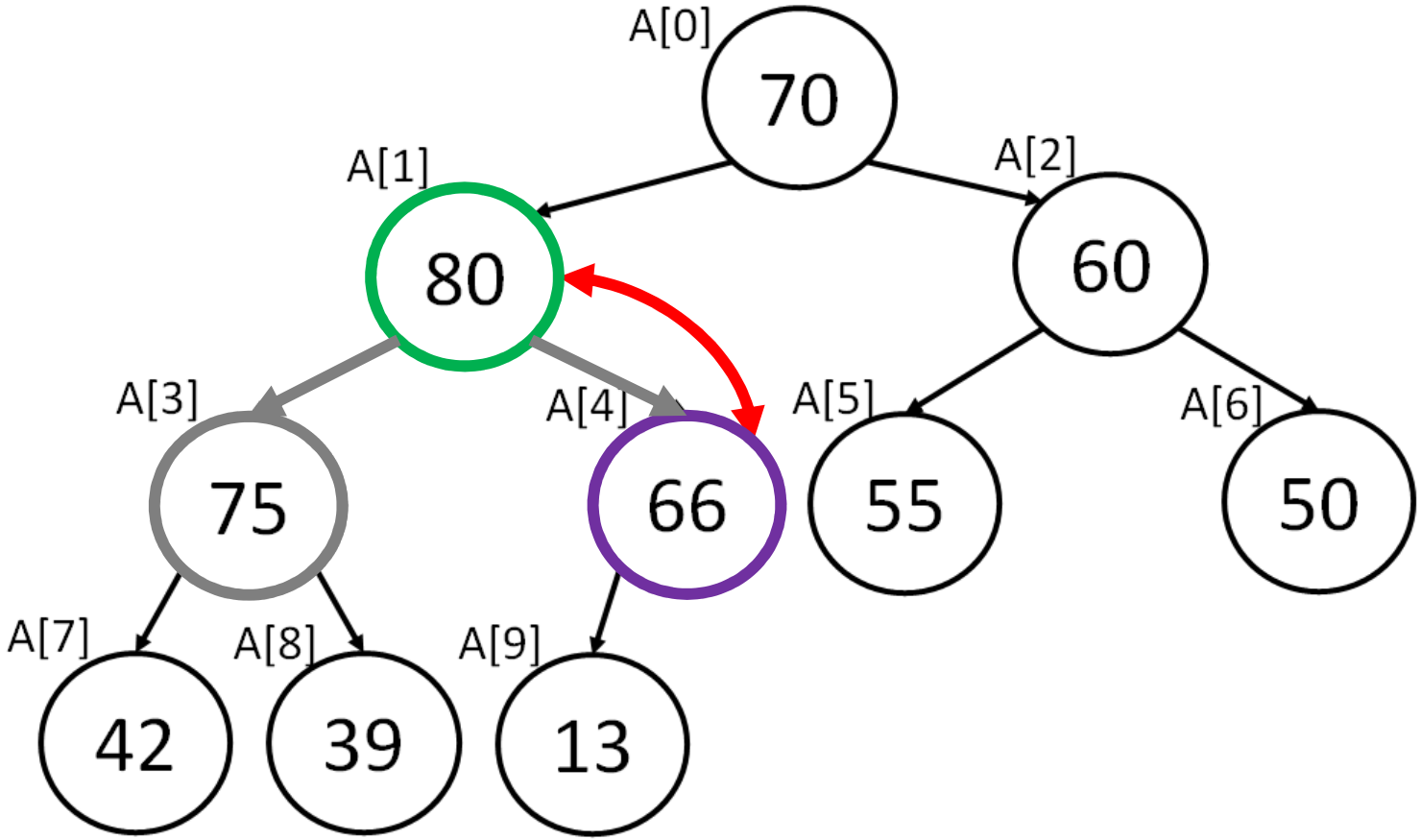
Heapifica(arranjo ***A, fim, i***)

1. ***e*** ← ***2*i + 1***
2. ***d*** ← ***2*i + 2***
3. Se ***e ≤ fim*** e ***A[e] > A[i]*** então
4. ***maior*** ← ***e***
5. Senão
6. ***maior*** ← ***i***
7. Se ***d ≤ fim*** e ***A[d] > A[maior]*** então
8. ***maior*** ← ***d***
9. Se ***maior ≠ i*** então
10. troca ***A[i]*** ↔ ***A[maior]***
11. ***Heapifica***(***A, fim, maior***)



Construindo a Heap: método Heapifica()

i	0	1	2	3	4	5	6	7	8	9
A[i]	70	80	60	75	66	55	50	42	39	13



Construindo a Heap: método Heapifica()

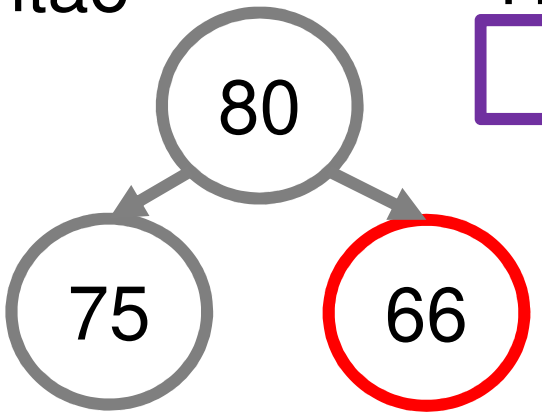
Heapifica(arranjo ***A, fim, i***)

1. ***e*** ← ***2*i + 1***
2. ***d*** ← ***2*i + 2***
3. Se ***e ≤ fim*** e ***A[e] > A[i]*** então
4. ***maior*** ← ***e***
5. Senão
6. ***maior*** ← ***i***
7. Se ***d ≤ fim*** e ***A[d] > A[maior]*** então
8. ***maior*** ← ***d***
9. Se ***maior ≠ i*** então
10. troca ***A[i]*** ↔ ***A[maior]***
11. ***Heapifica***(***A, fim, maior***)

i:	1	A[i]:	80
e:	3	A[e]:	75
d:	4	A[d]:	66

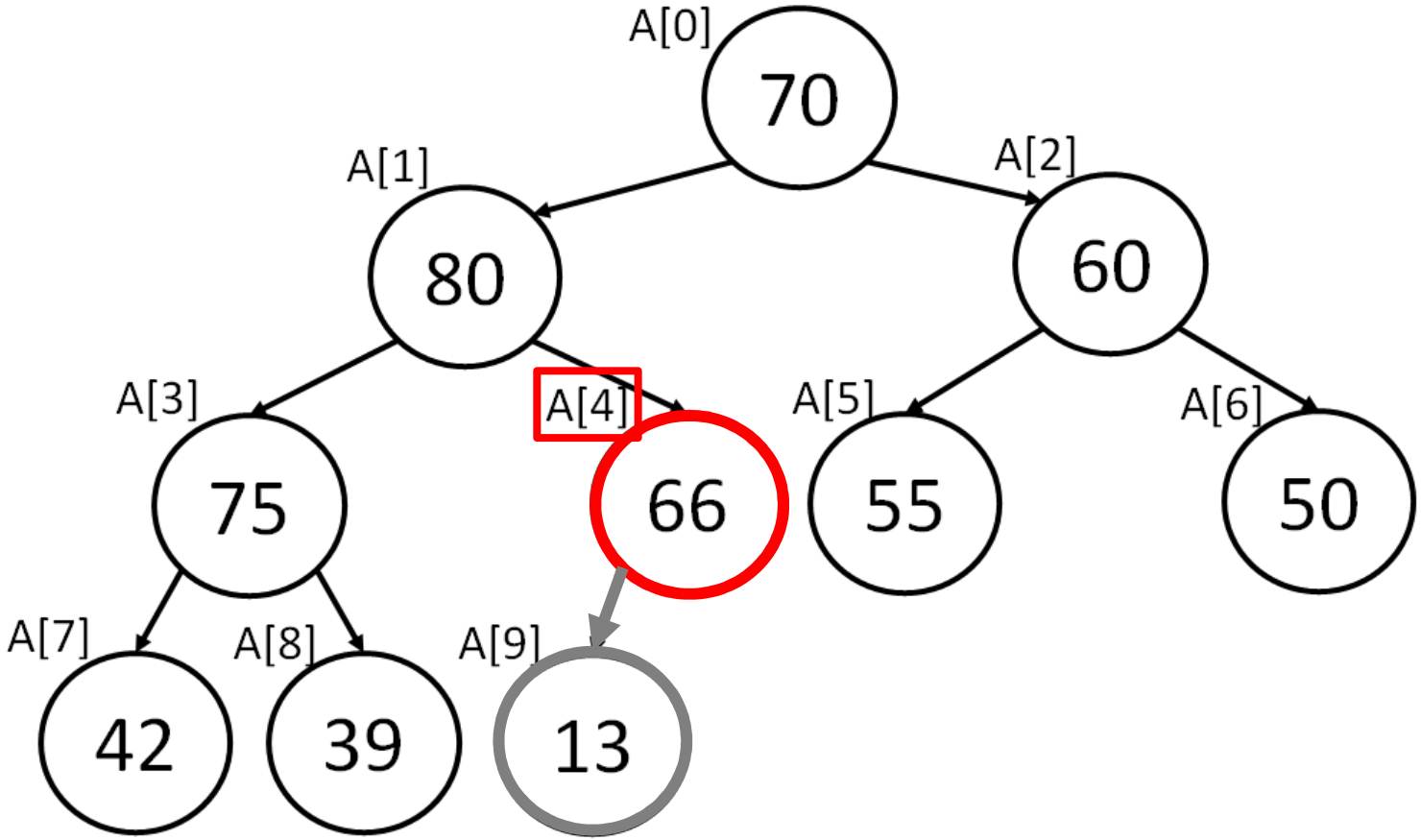
maior:	4
A[maior]:	66

fim: 9



Construindo a Heap: método Heapifica()

i	0	1	2	3	4	5	6	7	8	9
A[i]	70	80	60	75	66	55	50	42	39	13

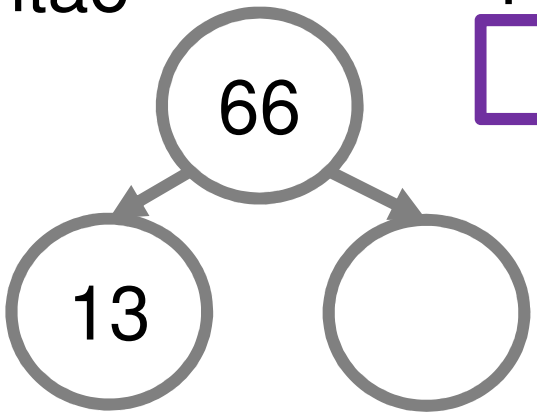


Construindo a Heap: método Heapifica()

Heapifica(arranjo ***A, fim, i***)

1. ***e*** ← ***2*i + 1***
2. ***d*** ← ***2*i + 2***
3. Se ***e ≤ fim*** e ***A[e] > A[i]*** então
4. ***maior*** ← ***e***
5. Senão
6. ***maior*** ← ***i***
7. Se ***d ≤ fim*** e ***A[d] > A[maior]*** então
8. ***maior*** ← ***d***
9. Se ***maior ≠ i*** então
10. troca ***A[i]*** ↔ ***A[maior]***
11. ***Heapifica***(***A, fim, maior***)

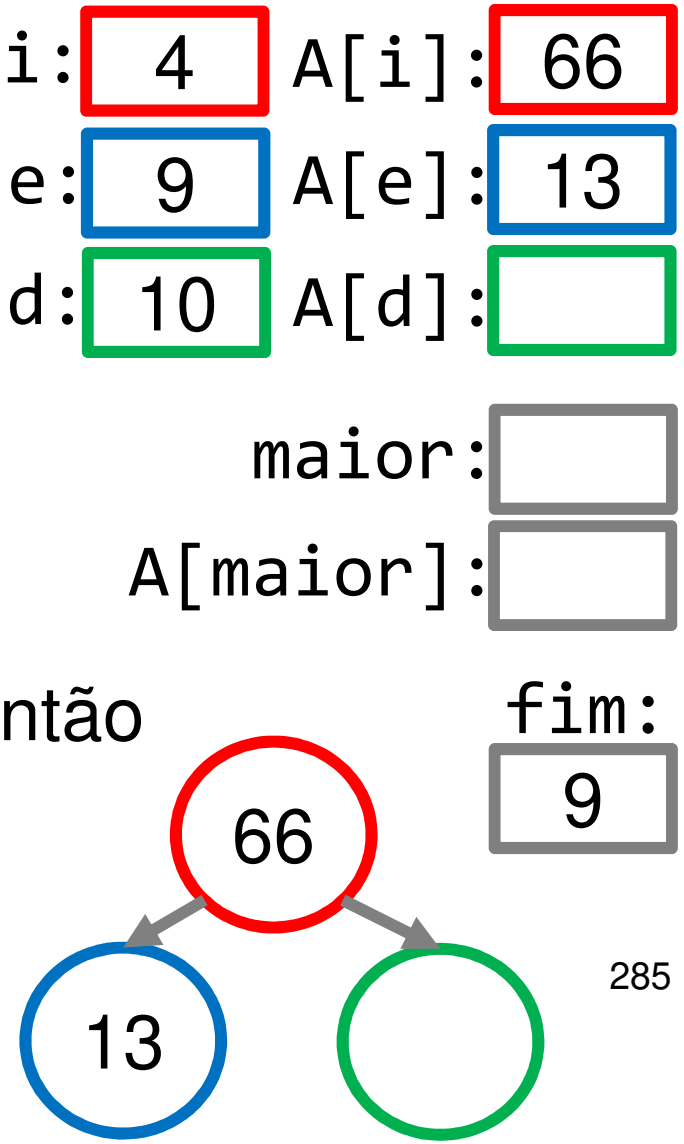
i:	4	A[i]:	66
e:		A[e]:	
d:		A[d]:	
maior:			
A[maior]:			
fim:	9		



Construindo a Heap: método Heapifica()

Heapifica(arranjo ***A, fim, i***)

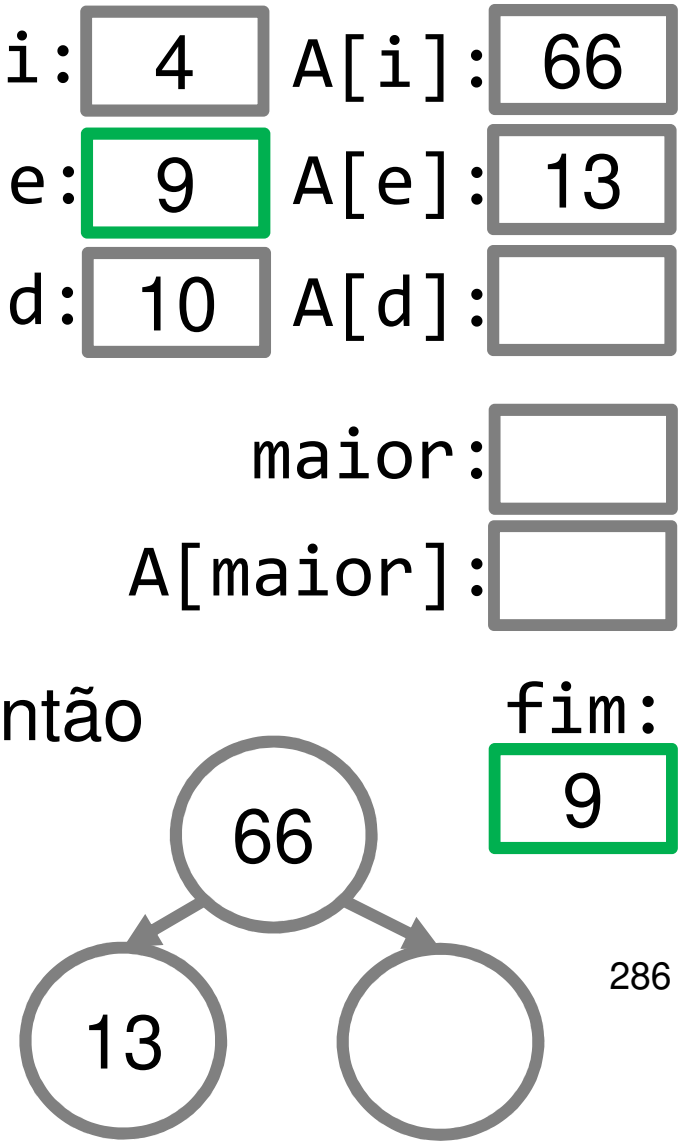
1. ***e*** ← ***2*i + 1***
2. ***d*** ← ***2*i + 2***
3. Se ***e ≤ fim*** e ***A[e] > A[i]*** então
4. ***maior*** ← ***e***
5. Senão
6. ***maior*** ← ***i***
7. Se ***d ≤ fim*** e ***A[d] > A[maior]*** então
8. ***maior*** ← ***d***
9. Se ***maior ≠ i*** então
10. troca ***A[i]*** ↔ ***A[maior]***
11. ***Heapifica***(***A, fim, maior***)



Construindo a Heap: método Heapifica()

Heapifica(arranjo ***A, fim, i***)

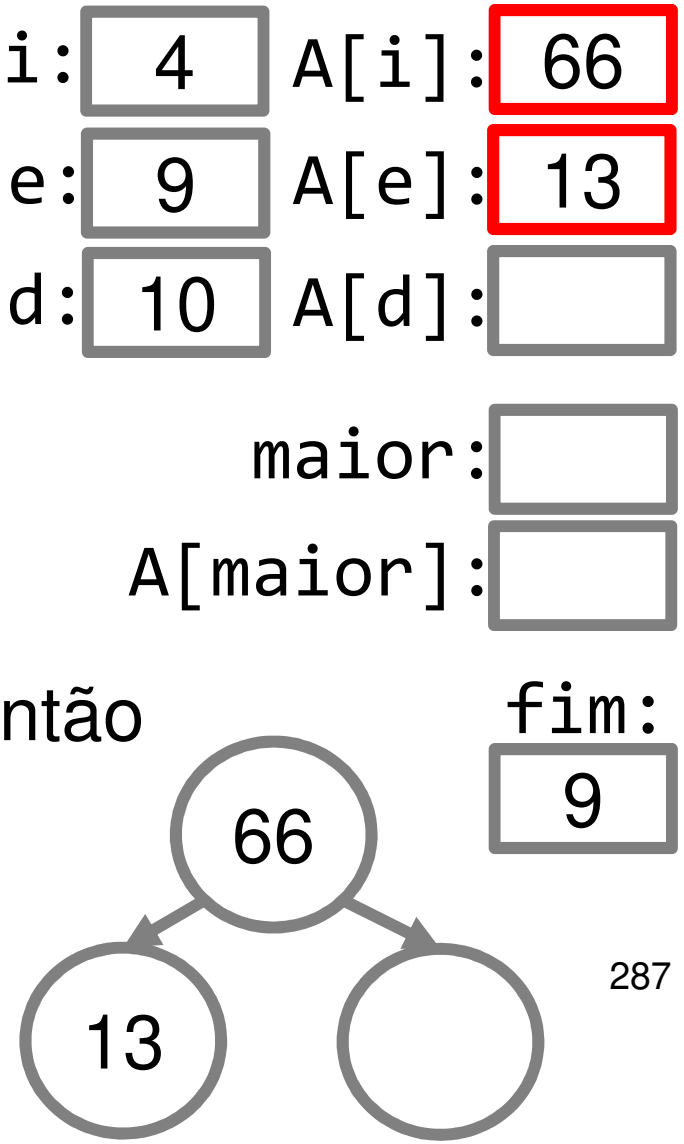
1. ***e*** ← ***2*i + 1***
2. ***d*** ← ***2*i + 2***
3. Se ***e ≤ fim*** e ***A[e] > A[i]*** então
4. ***maior*** ← ***e***
5. Senão
6. ***maior*** ← ***i***
7. Se ***d ≤ fim*** e ***A[d] > A[maior]*** então
8. ***maior*** ← ***d***
9. Se ***maior ≠ i*** então
10. troca ***A[i]*** ↔ ***A[maior]***
11. ***Heapifica***(***A, fim, maior***)



Construindo a Heap: método Heapifica()

Heapifica(arranjo ***A, fim, i***)

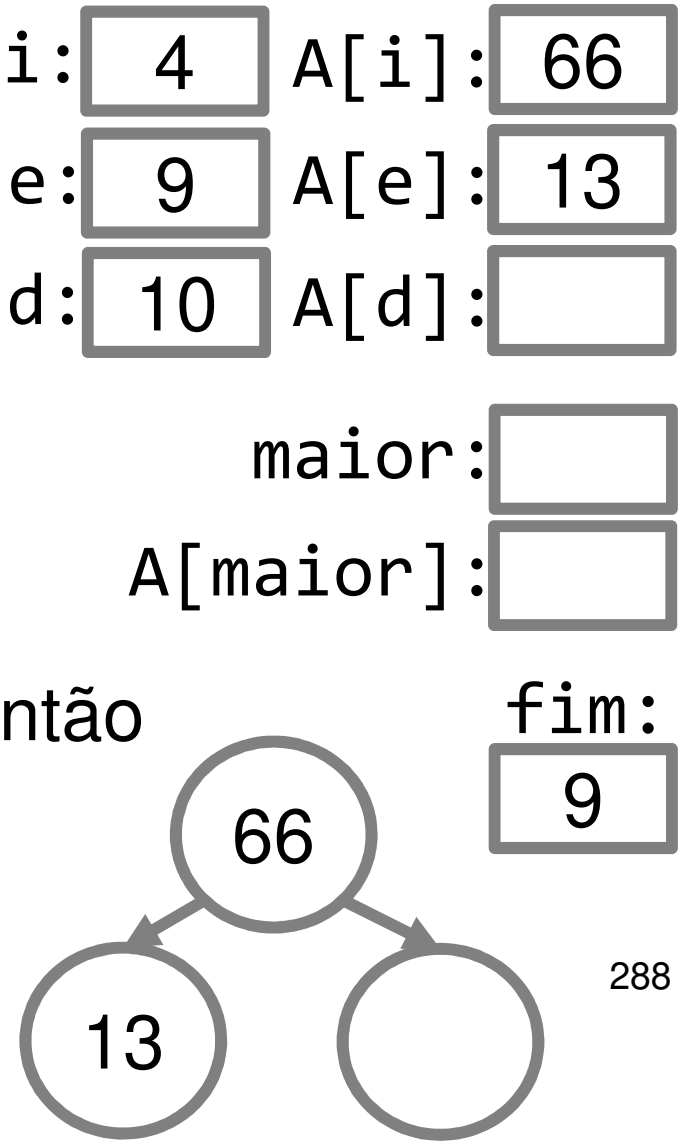
1. ***e*** ← ***2*i + 1***
2. ***d*** ← ***2*i + 2***
3. Se ***e*** ≤ ***fim*** e ***A[e] > A[i]*** então
4. ***maior*** ← ***e***
5. Senão
6. ***maior*** ← ***i***
7. Se ***d*** ≤ ***fim*** e ***A[d] > A[maior]*** então
8. ***maior*** ← ***d***
9. Se ***maior*** ≠ ***i*** então
10. troca ***A[i]*** ↔ ***A[maior]***
11. ***Heapifica***(***A, fim, maior***)



Construindo a Heap: método Heapifica()

Heapifica(arranjo ***A, fim, i***)

1. ***e*** ← ***2*i + 1***
2. ***d*** ← ***2*i + 2***
3. Se ***e ≤ fim*** e ***A[e] > A[i]*** então
4. ***maior*** ← ***e***
5. Senão
6. ***maior*** ← ***i***
7. Se ***d ≤ fim*** e ***A[d] > A[maior]*** então
8. ***maior*** ← ***d***
9. Se ***maior ≠ i*** então
10. troca ***A[i]*** ↔ ***A[maior]***
11. ***Heapifica***(***A, fim, maior***)

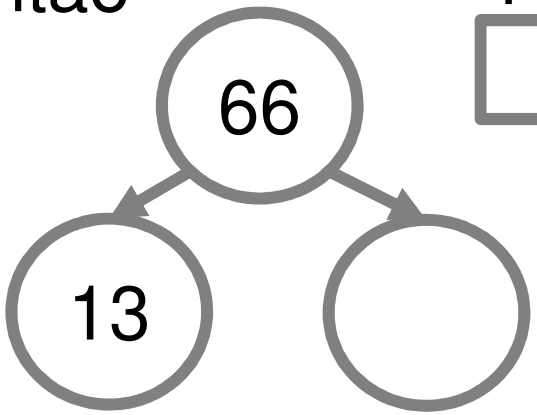


Construindo a Heap: método Heapifica()

Heapifica(arranjo ***A, fim, i***)

1. ***e*** ← ***2*i + 1***
2. ***d*** ← ***2*i + 2***
3. Se ***e ≤ fim*** e ***A[e] > A[i]*** então
4. ***maior*** ← ***e***
5. Senão
6. ***maior*** ← ***i***
7. Se ***d ≤ fim*** e ***A[d] > A[maior]*** então
8. ***maior*** ← ***d***
9. Se ***maior ≠ i*** então
10. troca ***A[i]*** ↔ ***A[maior]***
11. ***Heapifica***(***A, fim, maior***)

i:	4	A[i]:	66
e:	9	A[e]:	13
d:	10	A[d]:	
maior:	4		
A[maior]:	66		
fim:	9		

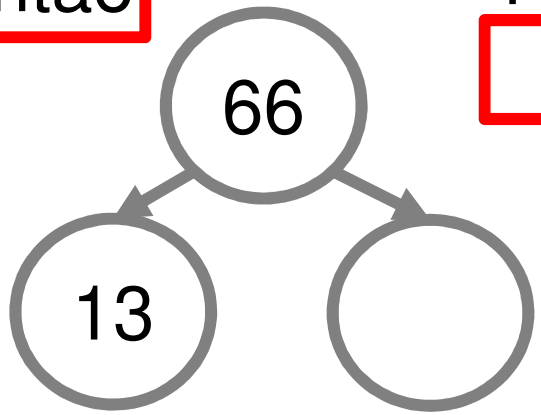


Construindo a Heap: método Heapifica()

Heapifica(arranjo ***A, fim, i***)

1. ***e*** ← ***2*i + 1***
2. ***d*** ← ***2*i + 2***
3. Se ***e ≤ fim*** e ***A[e] > A[i]*** então
4. ***maior*** ← ***e***
5. Senão
6. ***maior*** ← ***i***
7. Se ***d ≤ fim*** e ***A[d] > A[maior]*** então
8. ***maior*** ← ***d***
9. Se ***maior ≠ i*** então
10. troca ***A[i]*** ↔ ***A[maior]***
11. ***Heapifica***(***A, fim, maior***)

i:	4	A[i]:	66
e:	9	A[e]:	13
d:	10	A[d]:	
maior:	4		
A[maior]:	66		
fim:	9		

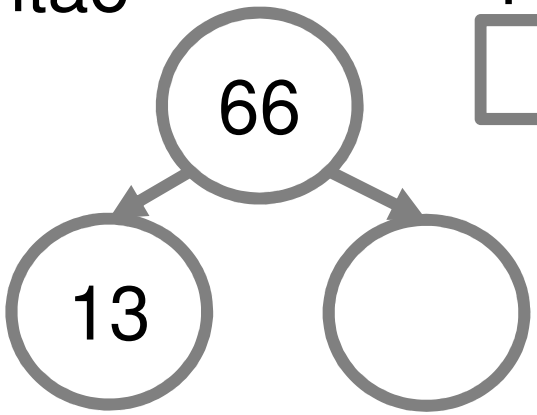


Construindo a Heap: método Heapifica()

Heapifica(arranjo ***A, fim, i***)

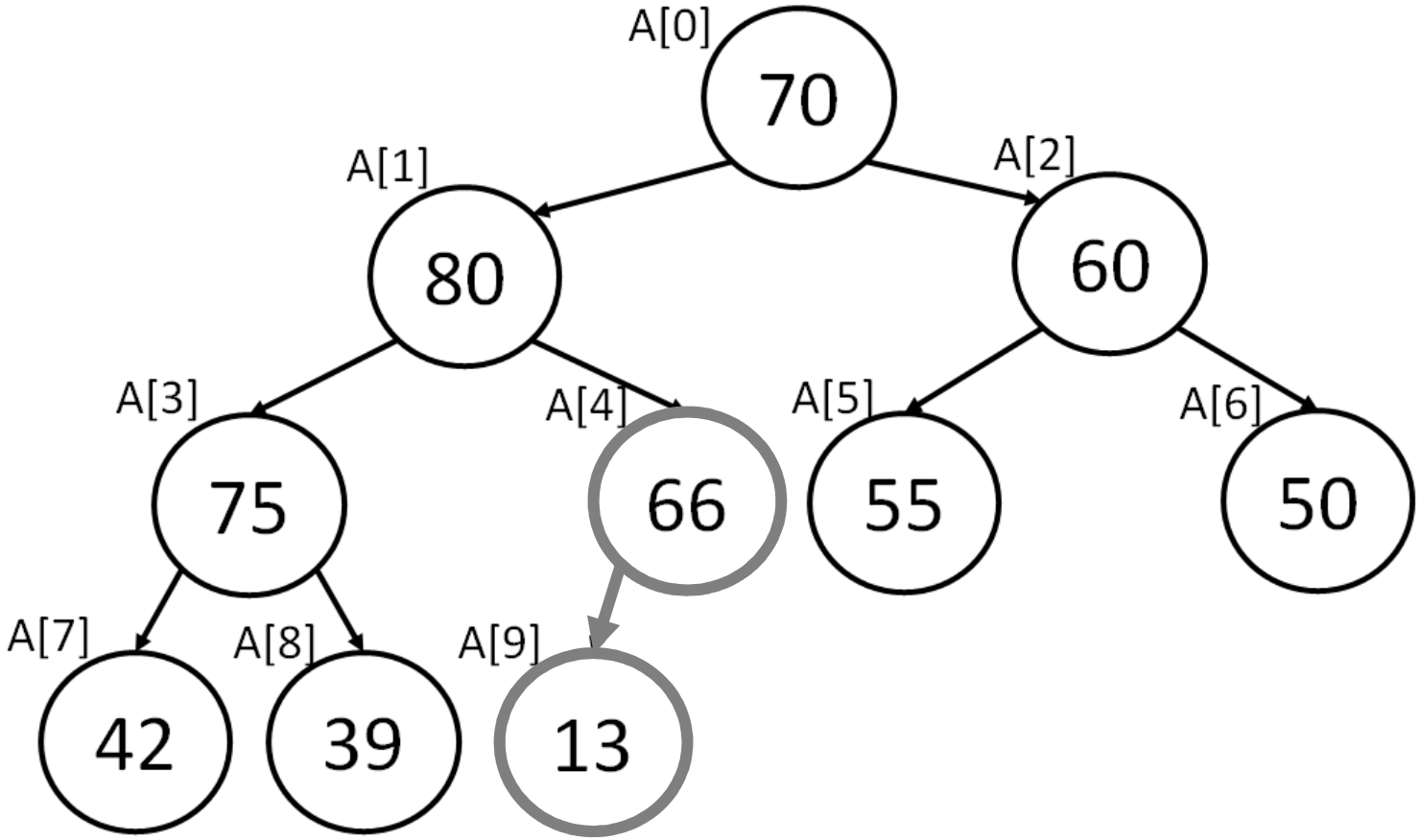
1. ***e*** ← ***2*i + 1***
2. ***d*** ← ***2*i + 2***
3. Se ***e ≤ fim*** e ***A[e] > A[i]*** então
4. ***maior*** ← ***e***
5. Senão
6. ***maior*** ← ***i***
7. Se ***d ≤ fim*** e ***A[d] > A[maior]*** então
8. ***maior*** ← ***d***
9. Se ***maior ≠ i*** então
10. troca ***A[i]*** ↔ ***A[maior]***
11. ***Heapifica***(***A, fim, maior***)

i:	4	A[i]:	66
e:	9	A[e]:	13
d:	10	A[d]:	
maior:	4		
A[maior]:	66		
fim:	9		



Construindo a Heap: método Heapifica()

i	0	1	2	3	4	5	6	7	8	9
A[i]	70	80	60	75	66	55	50	42	39	13



Construindo a Heap: método ConstroiHeap()

i	0	1	2	3	4	5	6	7	8	9
A[i]	70	80	60	75	66	55	50	42	39	13

fim: 9

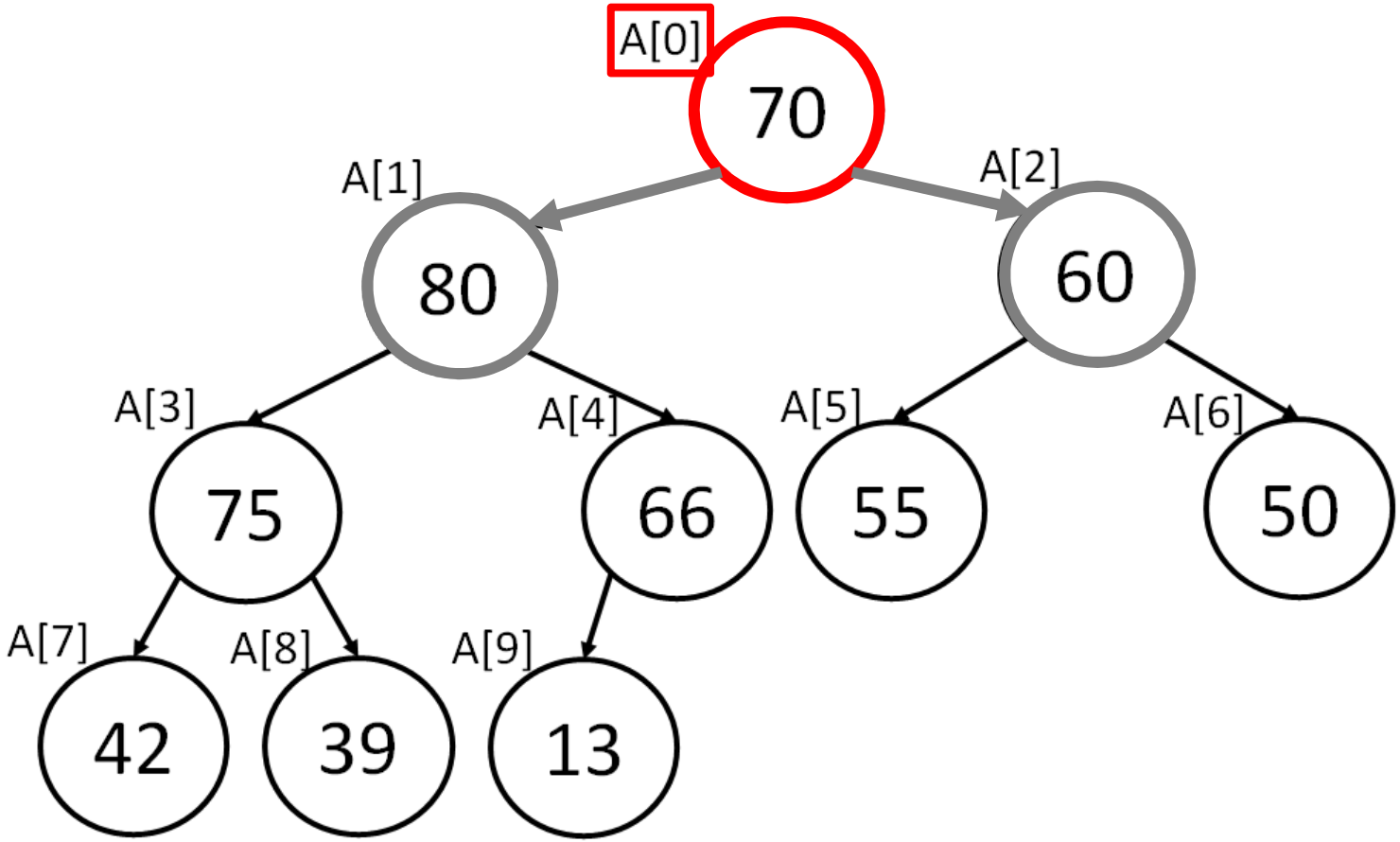
i: 0

ConstroiHeap(arranjo ***A, fim***)

1. Para ***i*** de ***fim /2*** até ***0*** faça
2. ***Heapifica***(***A, fim, i***)

Construindo a Heap: método ConstroiHeap()

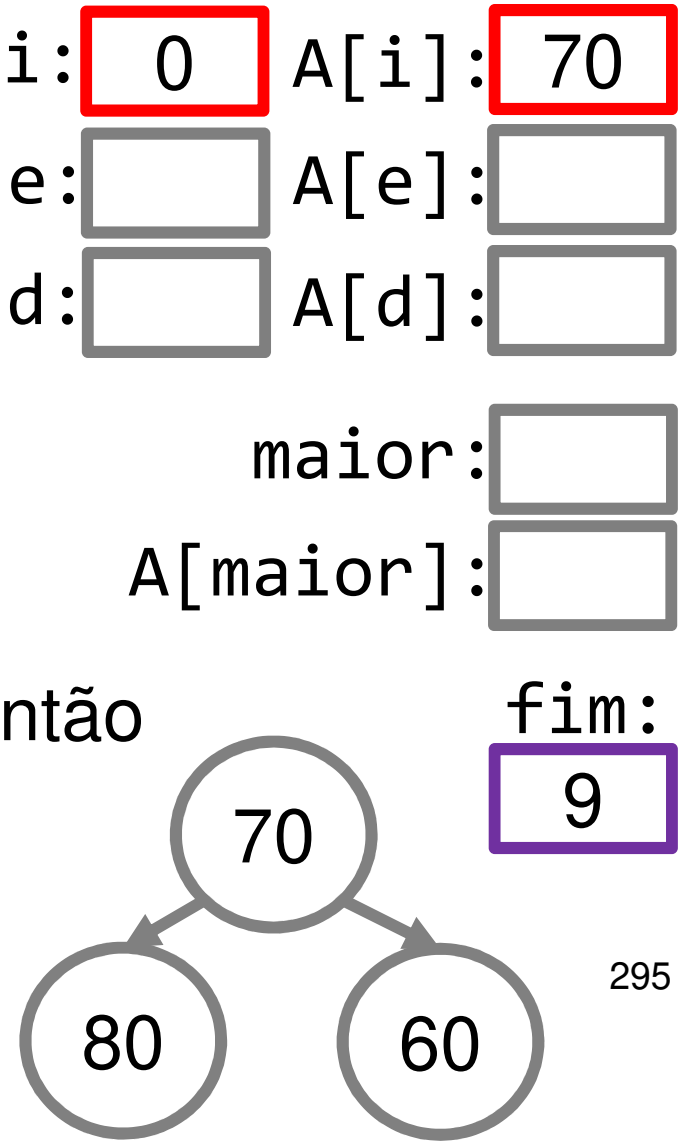
i	0	1	2	3	4	5	6	7	8	9
A[i]	70	80	60	75	66	55	50	42	39	13



Construindo a Heap: método Heapifica()

Heapifica(arranjo ***A, fim, i***)

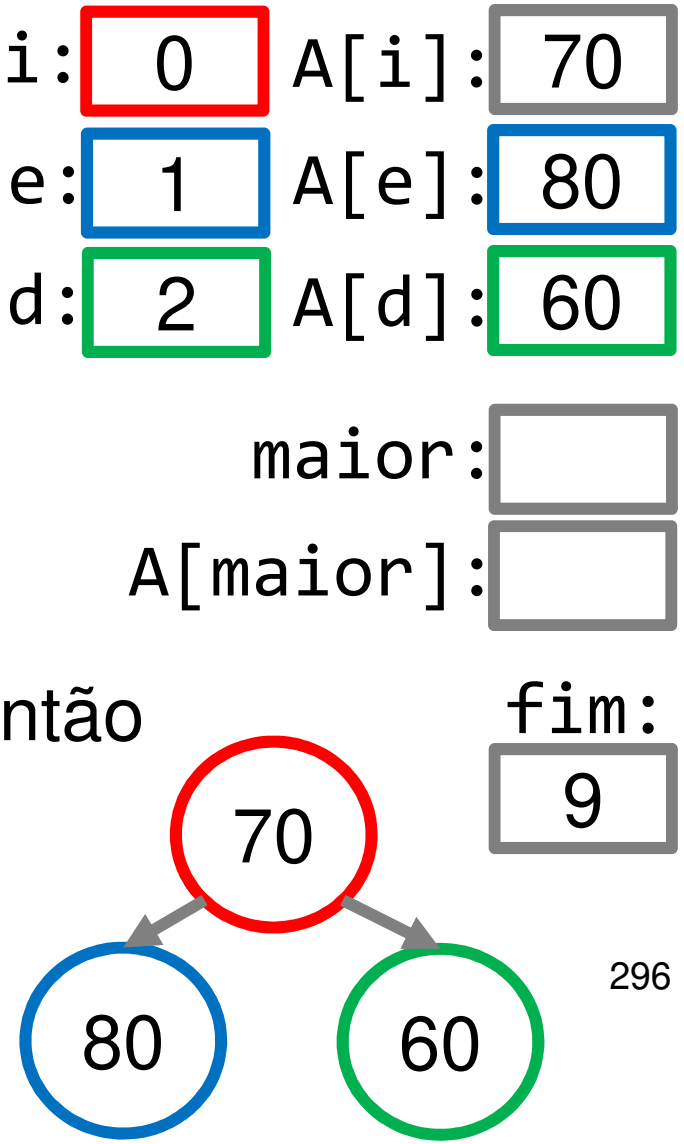
1. ***e*** ← ***2*i + 1***
2. ***d*** ← ***2*i + 2***
3. Se ***e ≤ fim*** e ***A[e] > A[i]*** então
4. ***maior*** ← ***e***
5. Senão
6. ***maior*** ← ***i***
7. Se ***d ≤ fim*** e ***A[d] > A[maior]*** então
8. ***maior*** ← ***d***
9. Se ***maior ≠ i*** então
10. troca ***A[i]*** ↔ ***A[maior]***
11. ***Heapifica***(***A, fim, maior***)



Construindo a Heap: método Heapifica()

Heapifica(arranjo ***A, fim, i***)

1. ***e*** ← ***2*i + 1***
2. ***d*** ← ***2*i + 2***
3. Se ***e ≤ fim*** e ***A[e] > A[i]*** então
4. ***maior*** ← ***e***
5. Senão
6. ***maior*** ← ***i***
7. Se ***d ≤ fim*** e ***A[d] > A[maior]*** então
8. ***maior*** ← ***d***
9. Se ***maior ≠ i*** então
10. troca ***A[i]*** ↔ ***A[maior]***
11. ***Heapifica***(***A, fim, maior***)

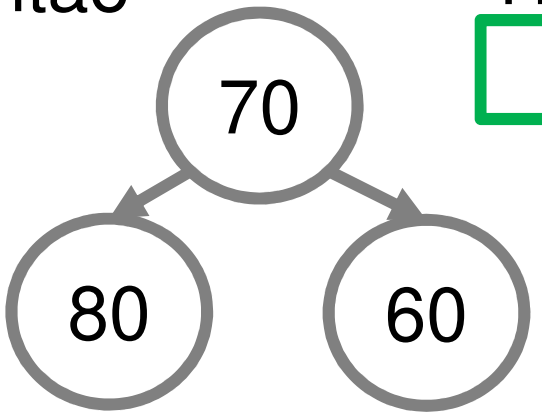


Construindo a Heap: método Heapifica()

Heapifica(arranjo ***A, fim, i***)

1. ***e*** ← ***2*i + 1***
2. ***d*** ← ***2*i + 2***
3. Se ***e ≤ fim*** e ***A[e] > A[i]*** então
4. ***maior*** ← ***e***
5. Senão
6. ***maior*** ← ***i***
7. Se ***d ≤ fim*** e ***A[d] > A[maior]*** então
8. ***maior*** ← ***d***
9. Se ***maior ≠ i*** então
10. troca ***A[i]*** ↔ ***A[maior]***
11. ***Heapifica***(***A, fim, maior***)

i:	0	A[i]:	70
e:	1	A[e]:	80
d:	2	A[d]:	60
maior:			
A[maior]:			
fim:	9		

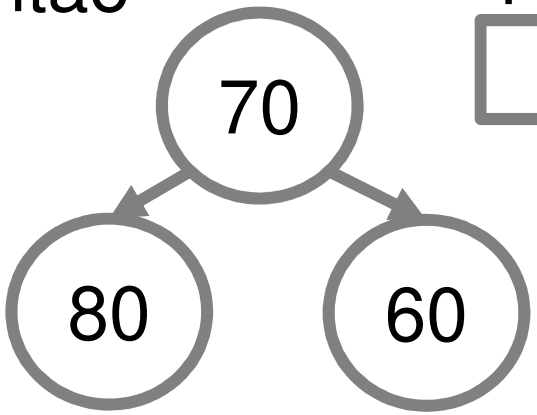


Construindo a Heap: método Heapifica()

Heapifica(arranjo ***A, fim, i***)

1. ***e*** ← ***2*i + 1***
2. ***d*** ← ***2*i + 2***
3. Se ***e ≤ fim*** e ***A[e] > A[i]*** então
4. ***maior*** ← ***e***
5. Senão
6. ***maior*** ← ***i***
7. Se ***d ≤ fim*** e ***A[d] > A[maior]*** então
8. ***maior*** ← ***d***
9. Se ***maior ≠ i*** então
10. troca ***A[i]*** ↔ ***A[maior]***
11. ***Heapifica***(***A, fim, maior***)

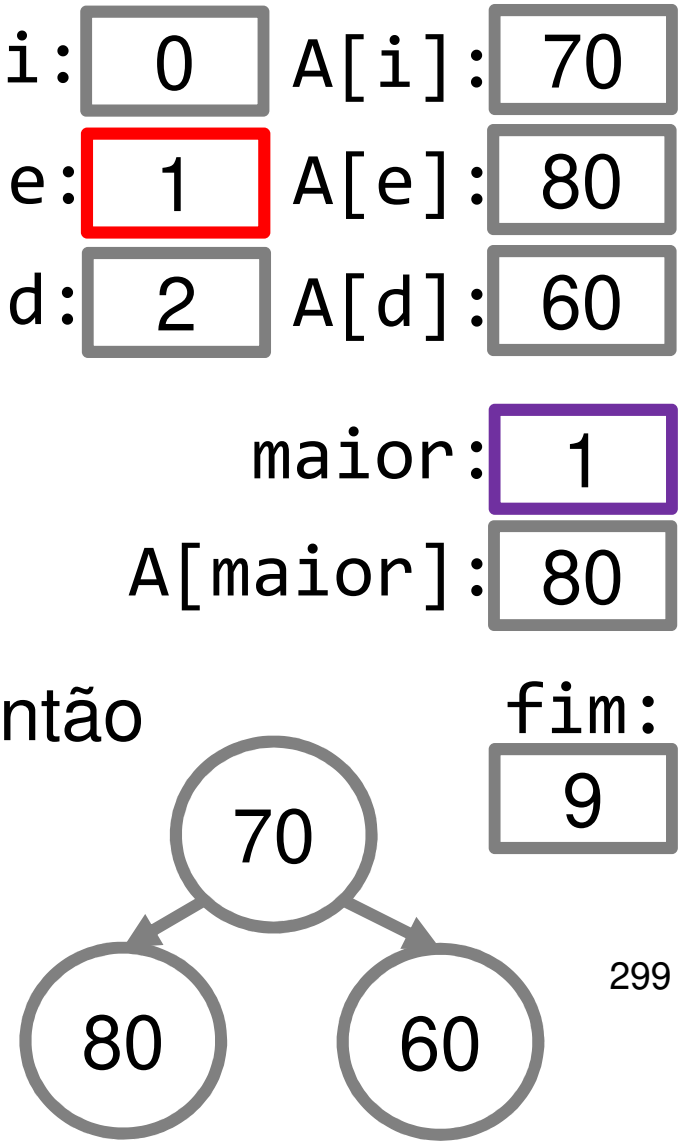
i:	0	A[i]:	70
e:	1	A[e]:	80
d:	2	A[d]:	60
maior:			
A[maior]:			
fim:	9		



Construindo a Heap: método Heapifica()

Heapifica(arranjo ***A, fim, i***)

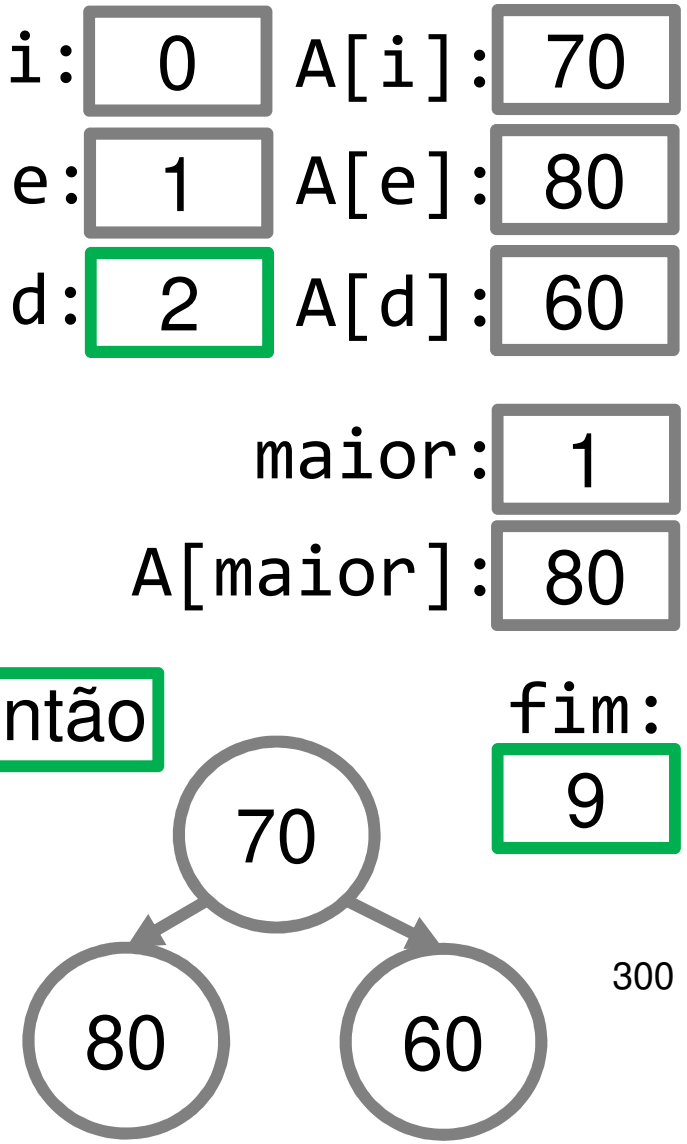
1. ***e*** ← ***2*i + 1***
2. ***d*** ← ***2*i + 2***
3. Se ***e ≤ fim*** e ***A[e] > A[i]*** então
4. ***maior*** ← ***e***
5. Senão
6. ***maior*** ← ***i***
7. Se ***d ≤ fim*** e ***A[d] > A[maior]*** então
8. ***maior*** ← ***d***
9. Se ***maior ≠ i*** então
10. troca ***A[i]*** ↔ ***A[maior]***
11. ***Heapifica***(***A, fim, maior***)



Construindo a Heap: método Heapifica()

Heapifica(arranjo ***A, fim, i***)

1. ***e*** ← ***2*i + 1***
2. ***d*** ← ***2*i + 2***
3. Se ***e ≤ fim*** e ***A[e] > A[i]*** então
4. ***maior*** ← ***e***
5. Senão
6. ***maior*** ← ***i***
7. Se ***d ≤ fim*** e ***A[d] > A[maior]*** então
8. ***maior*** ← ***d***
9. Se ***maior ≠ i*** então
10. troca ***A[i]*** ↔ ***A[maior]***
11. ***Heapifica***(***A, fim, maior***)



Construindo a Heap: método Heapifica()

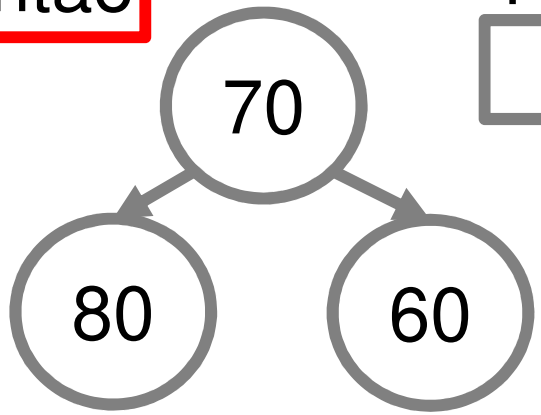
Heapifica(arranjo ***A, fim, i***)

1. ***e*** ← ***2*i + 1***
2. ***d*** ← ***2*i + 2***
3. Se ***e*** ≤ ***fim*** e ***A[e] > A[i]*** então
4. ***maior*** ← ***e***
5. Senão
6. ***maior*** ← ***i***
7. Se ***d*** ≤ ***fim*** e ***A[d] > A[maior]*** então
8. ***maior*** ← ***d***
9. Se ***maior*** ≠ ***i*** então
10. troca ***A[i]*** ↔ ***A[maior]***
11. ***Heapifica***(***A, fim, maior***)

i:	0	A[i]:	70
e:	1	A[e]:	80
d:	2	A[d]:	60

maior:	1
A[maior]:	80

fim: 9



Construindo a Heap: método Heapifica()

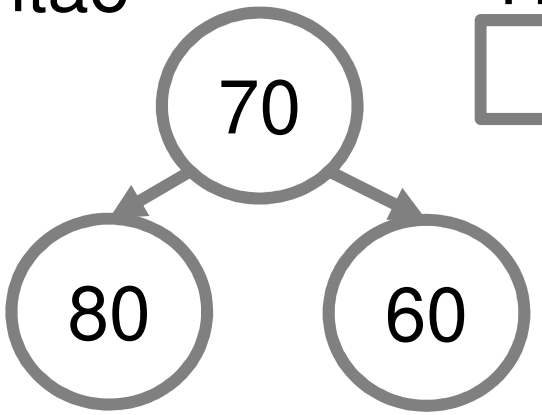
Heapifica(arranjo ***A, fim, i***)

1. ***e*** ← ***2*i + 1***
2. ***d*** ← ***2*i + 2***
3. Se ***e ≤ fim*** e ***A[e] > A[i]*** então
4. ***maior*** ← ***e***
5. Senão
6. ***maior*** ← ***i***
7. Se ***d ≤ fim*** e ***A[d] > A[maior]*** então
8. ***maior*** ← ***d***
9. Se ***maior ≠ i*** então
10. troca ***A[i]*** ↔ ***A[maior]***
11. ***Heapifica***(***A, fim, maior***)

i:	0	A[i]:	70
e:	1	A[e]:	80
d:	2	A[d]:	60

maior:	1
A[maior]:	80

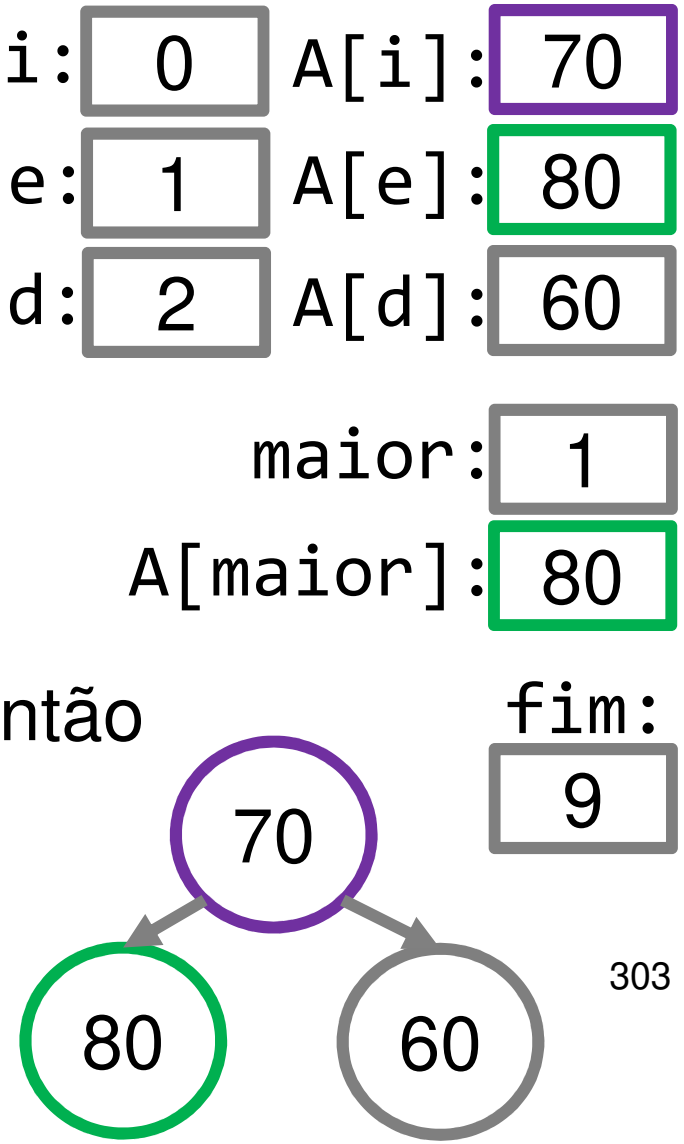
fim: 9



Construindo a Heap: método Heapifica()

Heapifica(arranjo ***A, fim, i***)

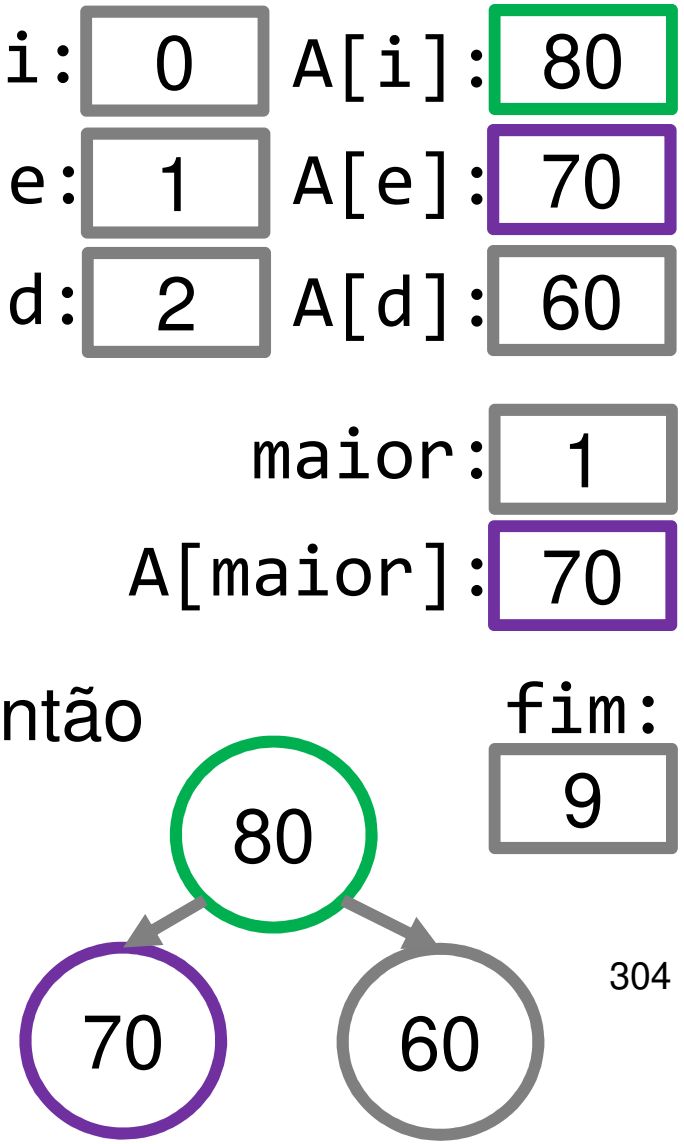
1. ***e*** ← ***2*i + 1***
2. ***d*** ← ***2*i + 2***
3. Se ***e ≤ fim*** e ***A[e] > A[i]*** então
4. ***maior*** ← ***e***
5. Senão
6. ***maior*** ← ***i***
7. Se ***d ≤ fim*** e ***A[d] > A[maior]*** então
8. ***maior*** ← ***d***
9. Se ***maior ≠ i*** então
10. troca ***A[i]*** ↔ ***A[maior]***
11. ***Heapifica***(***A, fim, maior***)



Construindo a Heap: método Heapifica()

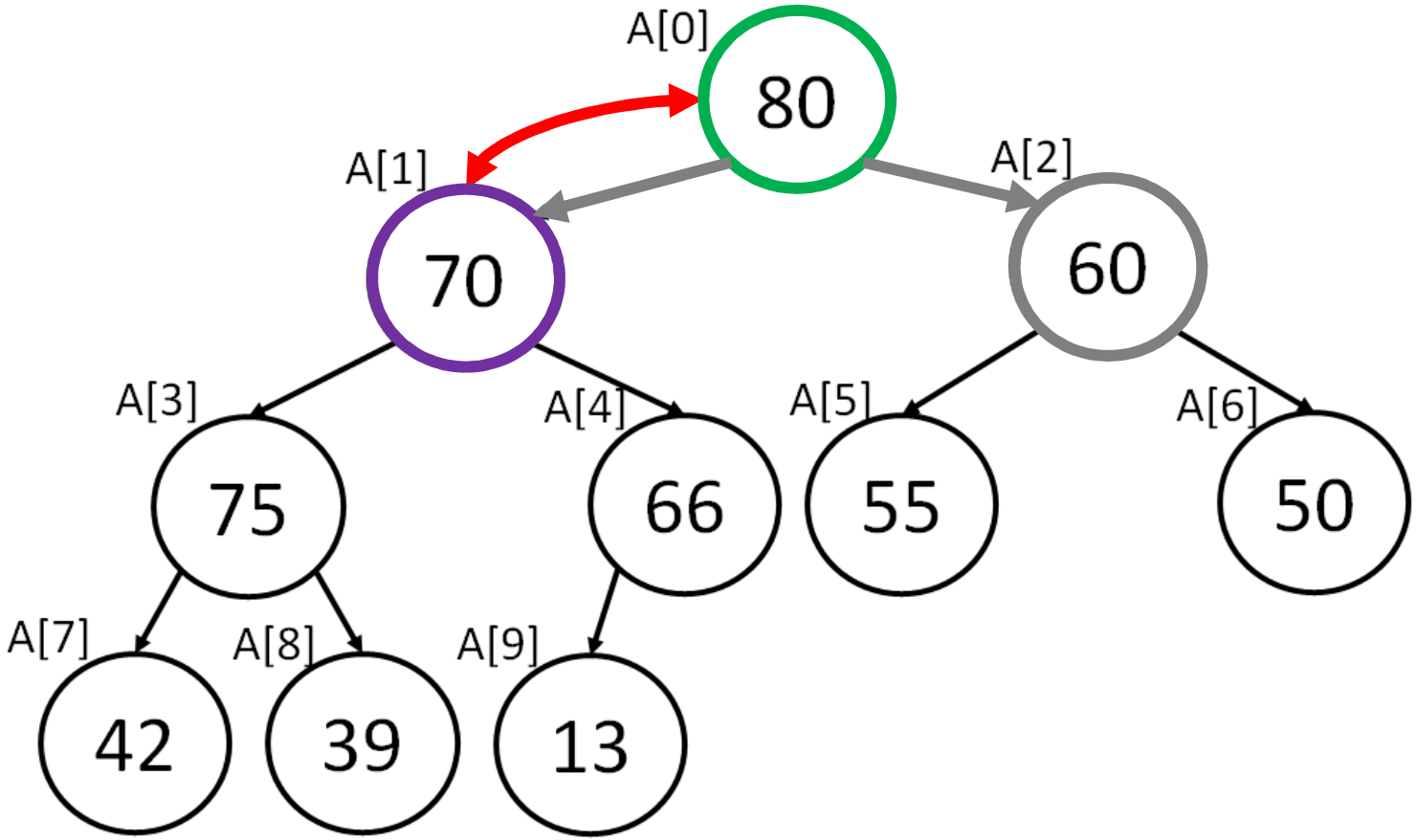
Heapifica(arranjo ***A, fim, i***)

1. ***e*** ← ***2*i + 1***
2. ***d*** ← ***2*i + 2***
3. Se ***e ≤ fim*** e ***A[e] > A[i]*** então
4. ***maior*** ← ***e***
5. Senão
6. ***maior*** ← ***i***
7. Se ***d ≤ fim*** e ***A[d] > A[maior]*** então
8. ***maior*** ← ***d***
9. Se ***maior ≠ i*** então
10. troca ***A[i]*** ↔ ***A[maior]***
11. ***Heapifica***(***A, fim, maior***)



Construindo a Heap: método Heapifica()

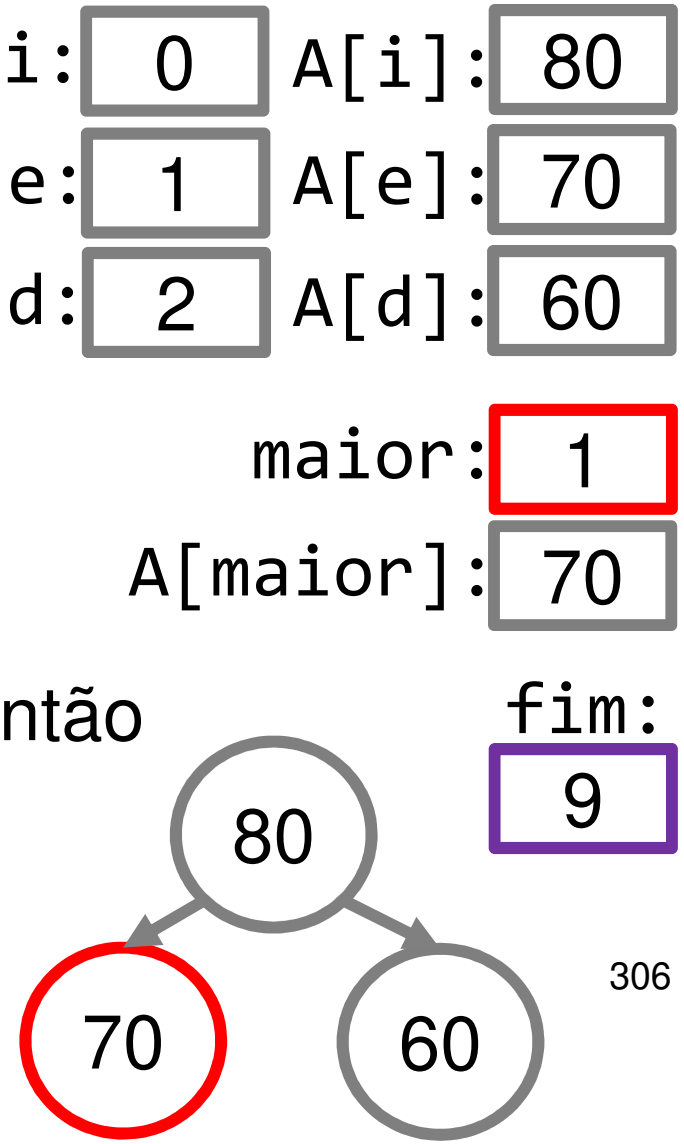
i	0	1	2	3	4	5	6	7	8	9
A[i]	80	70	60	75	66	55	50	42	39	13



Construindo a Heap: método Heapifica()

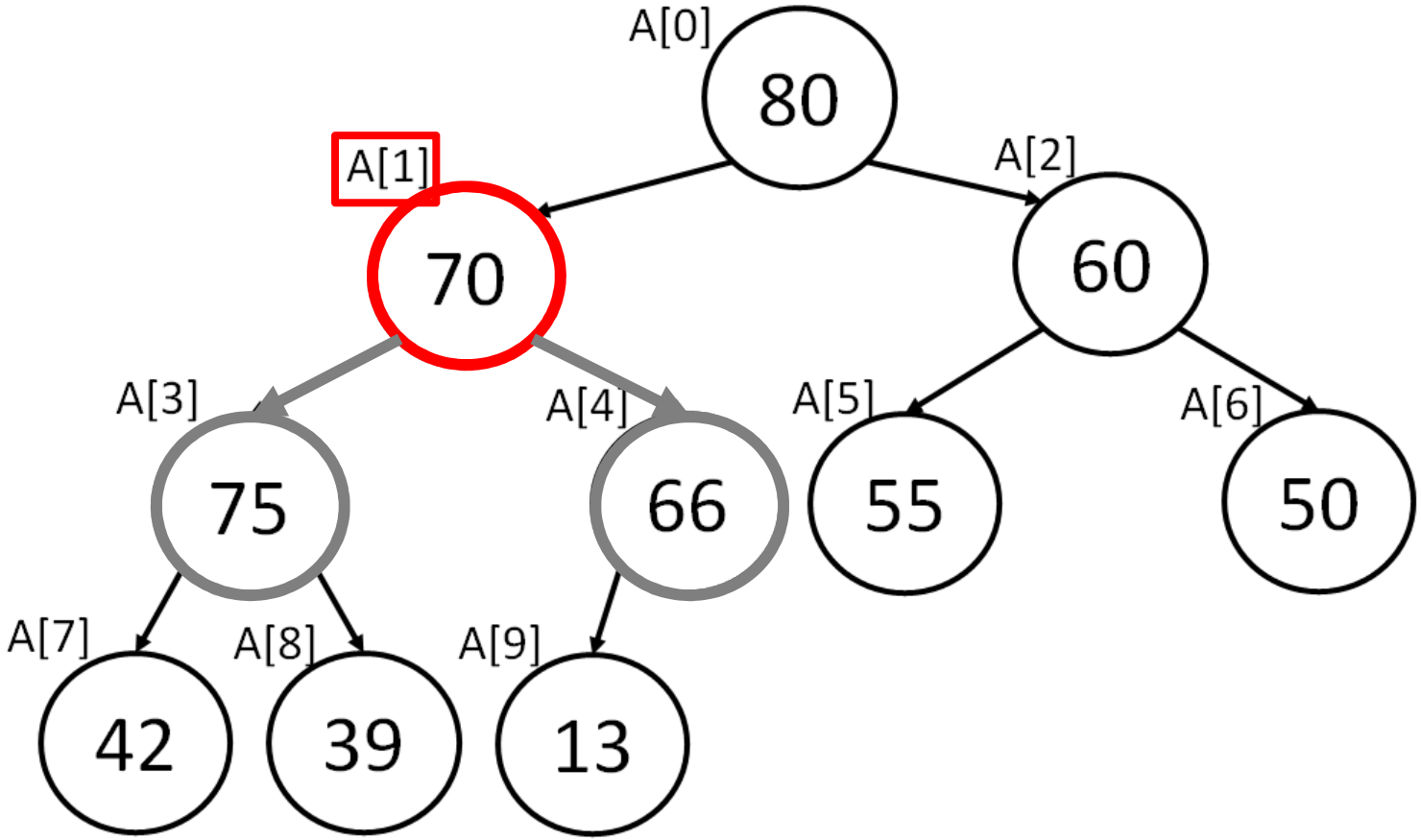
Heapifica(arranjo ***A, fim, i***)

1. ***e*** ← ***2*i + 1***
2. ***d*** ← ***2*i + 2***
3. Se ***e ≤ fim*** e ***A[e] > A[i]*** então
4. ***maior*** ← ***e***
5. Senão
6. ***maior*** ← ***i***
7. Se ***d ≤ fim*** e ***A[d] > A[maior]*** então
8. ***maior*** ← ***d***
9. Se ***maior ≠ i*** então
10. troca ***A[i]*** ↔ ***A[maior]***
11. ***Heapifica***(***A, fim, maior***)



Construindo a Heap: método Heapifica()

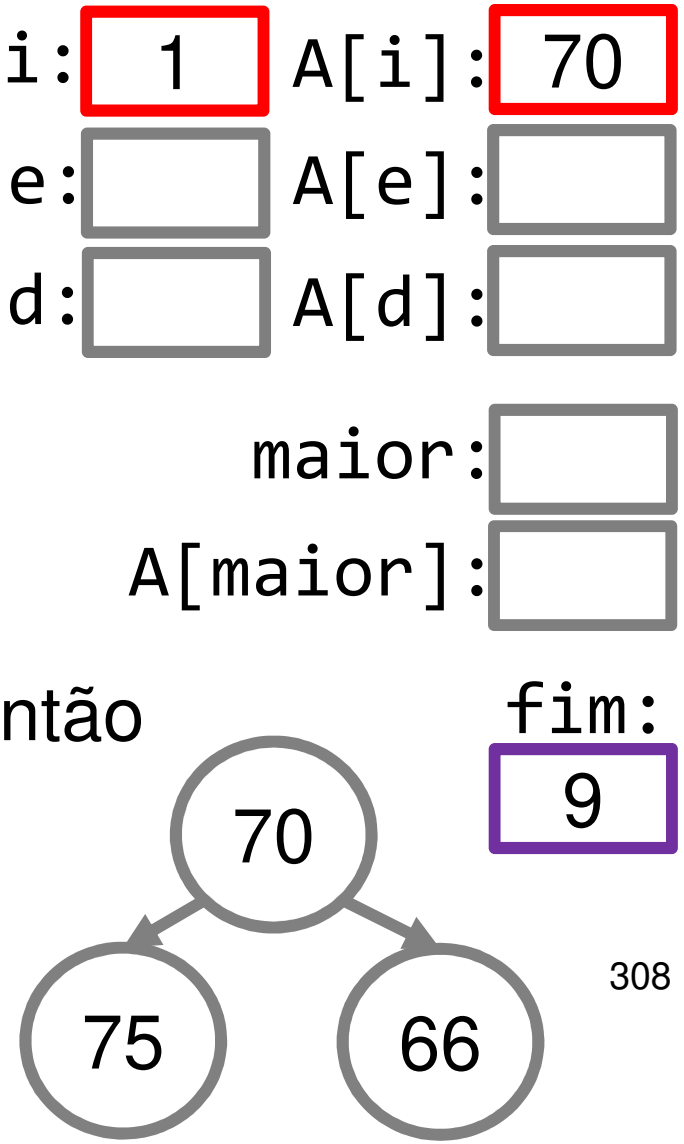
i	0	1	2	3	4	5	6	7	8	9
A[i]	80	70	60	75	66	55	50	42	39	13



Construindo a Heap: método Heapifica()

Heapifica(arranjo ***A***, ***fim***, ***i***)

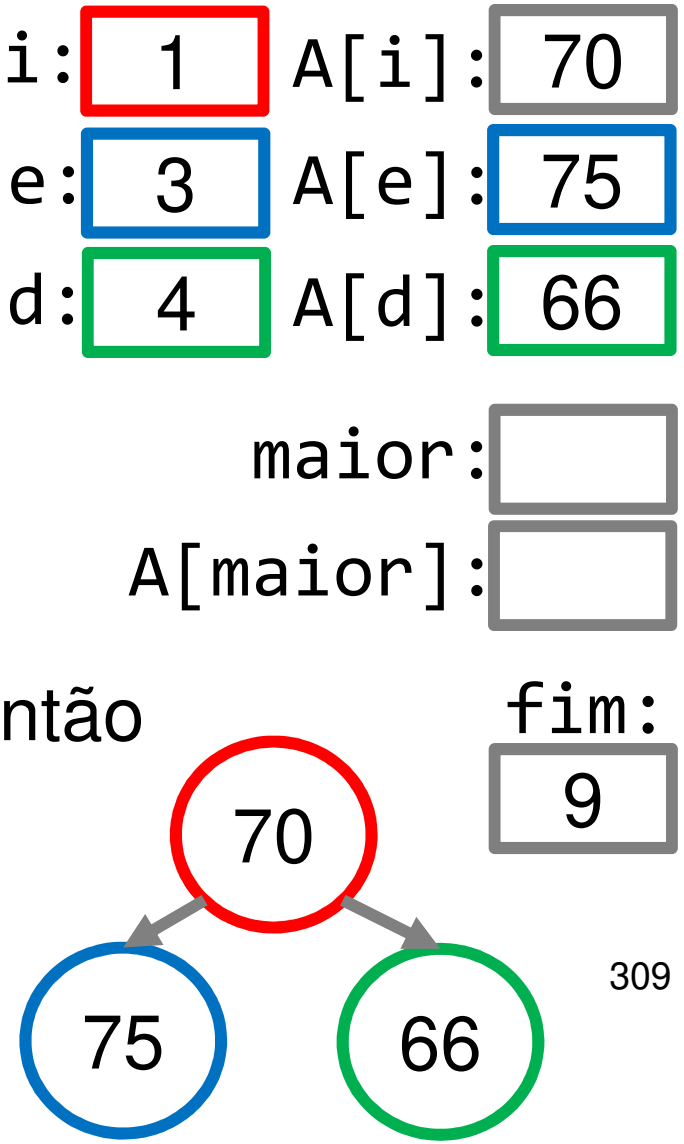
1. ***e*** ← ***2*i + 1***
2. ***d*** ← ***2*i + 2***
3. Se ***e*** ≤ ***fim*** e ***A[e]*** > ***A[i]*** então
4. ***maior*** ← ***e***
5. Senão
6. ***maior*** ← ***i***
7. Se ***d*** ≤ ***fim*** e ***A[d]*** > ***A[maior]*** então
8. ***maior*** ← ***d***
9. Se ***maior*** ≠ ***i*** então
10. troca ***A[i]*** ↔ ***A[maior]***
11. ***Heapifica***(***A, fim, maior***)



Construindo a Heap: método Heapifica()

Heapifica(arranjo ***A, fim, i***)

1. ***e*** ← ***2*i + 1***
2. ***d*** ← ***2*i + 2***
3. Se ***e ≤ fim*** e ***A[e] > A[i]*** então
4. ***maior*** ← ***e***
5. Senão
6. ***maior*** ← ***i***
7. Se ***d ≤ fim*** e ***A[d] > A[maior]*** então
8. ***maior*** ← ***d***
9. Se ***maior ≠ i*** então
10. troca ***A[i]*** ↔ ***A[maior]***
11. ***Heapifica***(***A, fim, maior***)

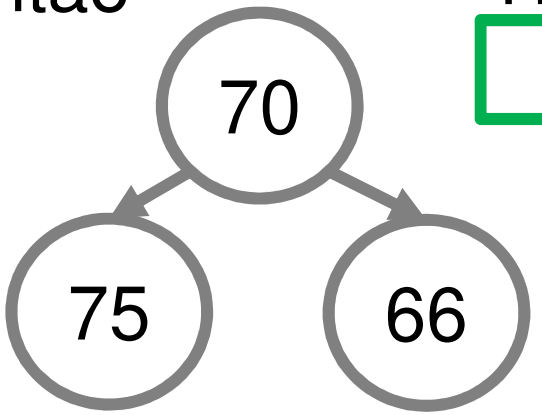


Construindo a Heap: método Heapifica()

Heapifica(arranjo ***A, fim, i***)

1. ***e*** ← ***2*i + 1***
2. ***d*** ← ***2*i + 2***
3. Se ***e ≤ fim*** e ***A[e] > A[i]*** então
4. ***maior*** ← ***e***
5. Senão
6. ***maior*** ← ***i***
7. Se ***d ≤ fim*** e ***A[d] > A[maior]*** então
8. ***maior*** ← ***d***
9. Se ***maior ≠ i*** então
10. troca ***A[i]*** ↔ ***A[maior]***
11. ***Heapifica***(***A, fim, maior***)

i:	1	A[i]:	70
e:	3	A[e]:	75
d:	4	A[d]:	66
maior:			
A[maior]:			
fim:	9		

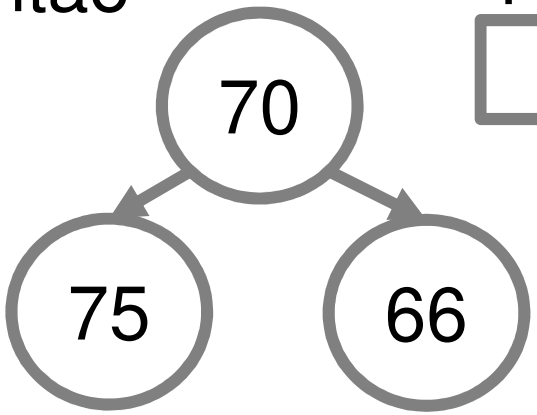


Construindo a Heap: método Heapifica()

Heapifica(arranjo ***A, fim, i***)

1. ***e*** ← ***2*i + 1***
2. ***d*** ← ***2*i + 2***
3. Se ***e ≤ fim*** e ***A[e] > A[i]*** então
4. ***maior*** ← ***e***
5. Senão
6. ***maior*** ← ***i***
7. Se ***d ≤ fim*** e ***A[d] > A[maior]*** então
8. ***maior*** ← ***d***
9. Se ***maior ≠ i*** então
10. troca ***A[i]*** ↔ ***A[maior]***
11. ***Heapifica***(***A, fim, maior***)

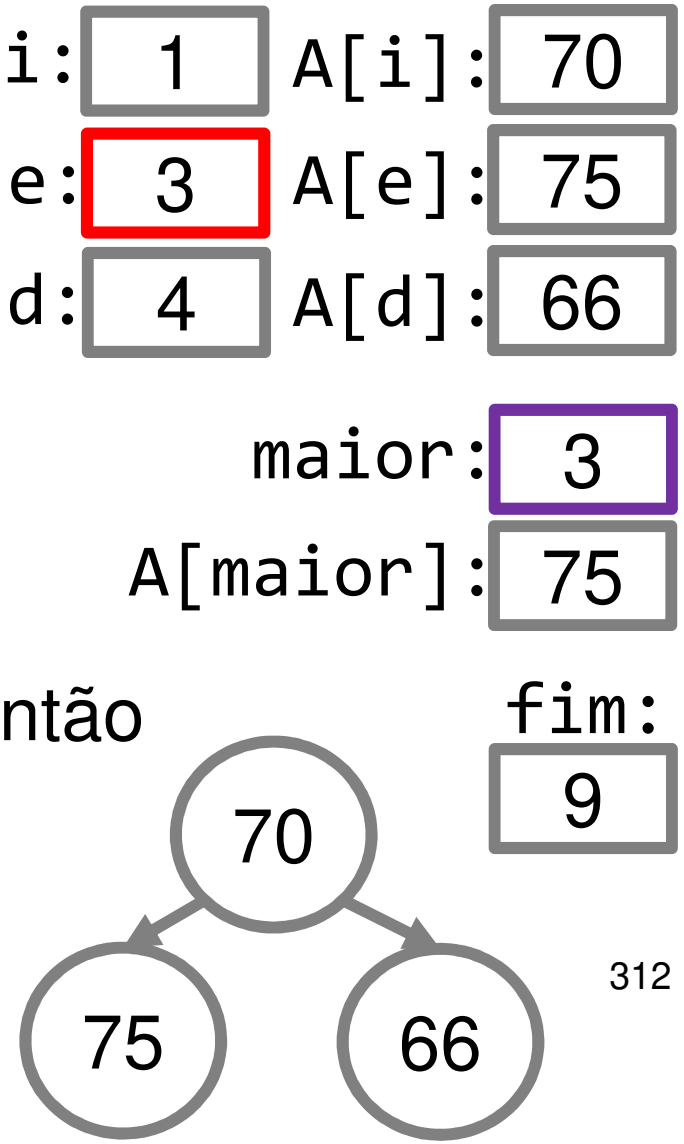
i:	1	A[i]:	70
e:	3	A[e]:	75
d:	4	A[d]:	66
maior:			
A[maior]:			
fim:	9		



Construindo a Heap: método Heapifica()

Heapifica(arranjo ***A, fim, i***)

1. ***e*** ← ***2*i + 1***
2. ***d*** ← ***2*i + 2***
3. Se ***e ≤ fim*** e ***A[e] > A[i]*** então
4. ***maior*** ← ***e***
5. Senão
6. ***maior*** ← ***i***
7. Se ***d ≤ fim*** e ***A[d] > A[maior]*** então
8. ***maior*** ← ***d***
9. Se ***maior ≠ i*** então
10. troca ***A[i]*** ↔ ***A[maior]***
11. ***Heapifica***(***A, fim, maior***)

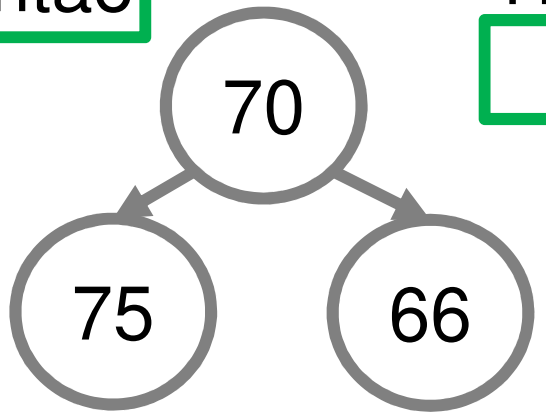


Construindo a Heap: método Heapifica()

Heapifica(arranjo ***A, fim, i***)

1. ***e*** ← ***2*i + 1***
2. ***d*** ← ***2*i + 2***
3. Se ***e ≤ fim*** e ***A[e] > A[i]*** então
4. ***maior*** ← ***e***
5. Senão
6. ***maior*** ← ***i***
7. Se ***d ≤ fim*** e ***A[d] > A[maior]*** então
8. ***maior*** ← ***d***
9. Se ***maior ≠ i*** então
10. troca ***A[i]*** ↔ ***A[maior]***
11. ***Heapifica***(***A, fim, maior***)

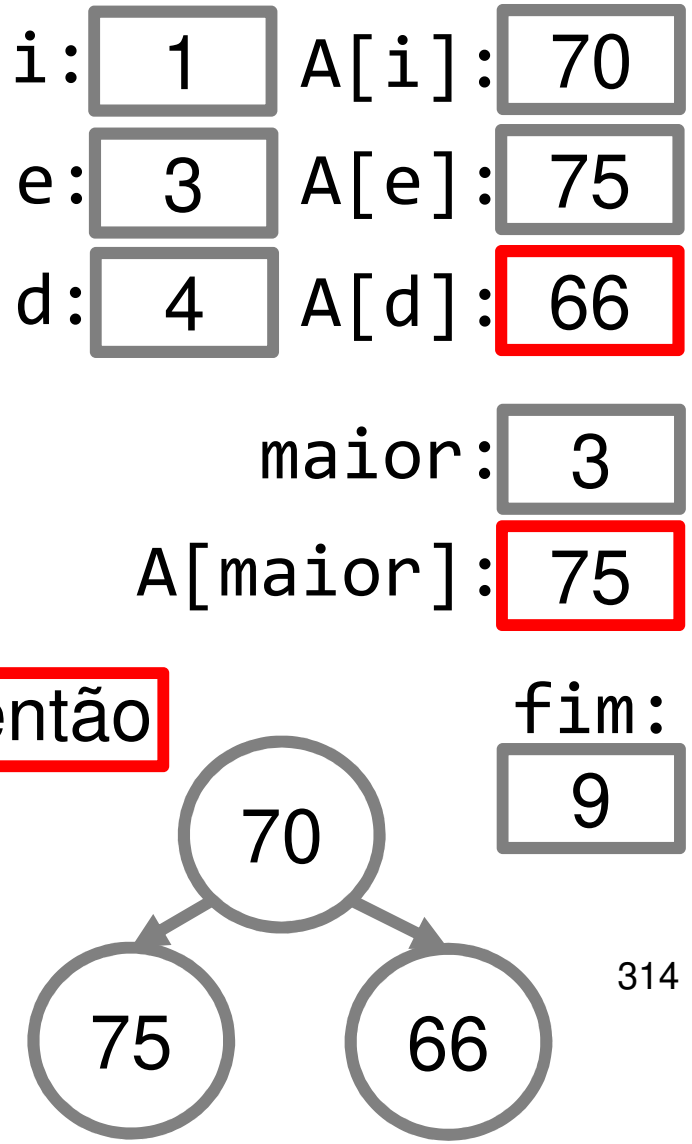
i:	1	A[i]:	70
e:	3	A[e]:	75
d:	4	A[d]:	66
maior:	3		
A[maior]:	75		
fim:	9		



Construindo a Heap: método Heapifica()

Heapifica(arranjo ***A, fim, i***)

1. ***e*** ← ***2*i + 1***
2. ***d*** ← ***2*i + 2***
3. Se ***e*** ≤ ***fim*** e ***A[e] > A[i]*** então
4. ***maior*** ← ***e***
5. Senão
6. ***maior*** ← ***i***
7. Se ***d*** ≤ ***fim*** e ***A[d] > A[maior]*** então
8. ***maior*** ← ***d***
9. Se ***maior*** ≠ ***i*** então
10. troca ***A[i]*** ↔ ***A[maior]***
11. ***Heapifica***(***A, fim, maior***)



Construindo a Heap: método Heapifica()

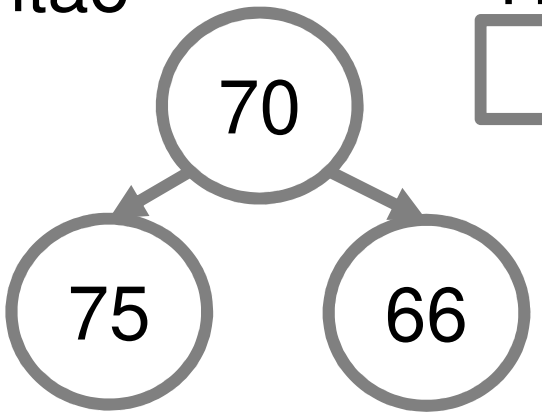
Heapifica(arranjo ***A, fim, i***)

1. ***e*** ← ***2*i + 1***
2. ***d*** ← ***2*i + 2***
3. Se ***e ≤ fim*** e ***A[e] > A[i]*** então
4. ***maior*** ← ***e***
5. Senão
6. ***maior*** ← ***i***
7. Se ***d ≤ fim*** e ***A[d] > A[maior]*** então
8. ***maior*** ← ***d***
9. Se ***maior ≠ i*** então
10. troca ***A[i]*** ↔ ***A[maior]***
11. ***Heapifica***(***A, fim, maior***)

i:	1	A[i]:	70
e:	3	A[e]:	75
d:	4	A[d]:	66

maior:	3
A[maior]:	75

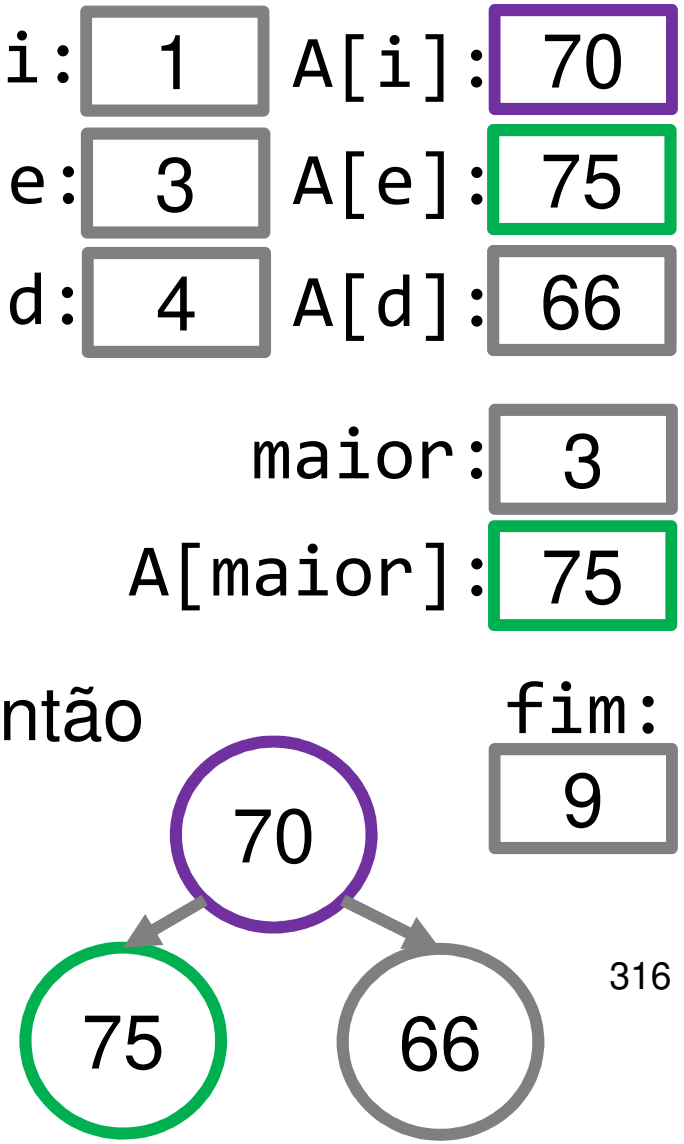
fim: 9



Construindo a Heap: método Heapifica()

Heapifica(arranjo ***A, fim, i***)

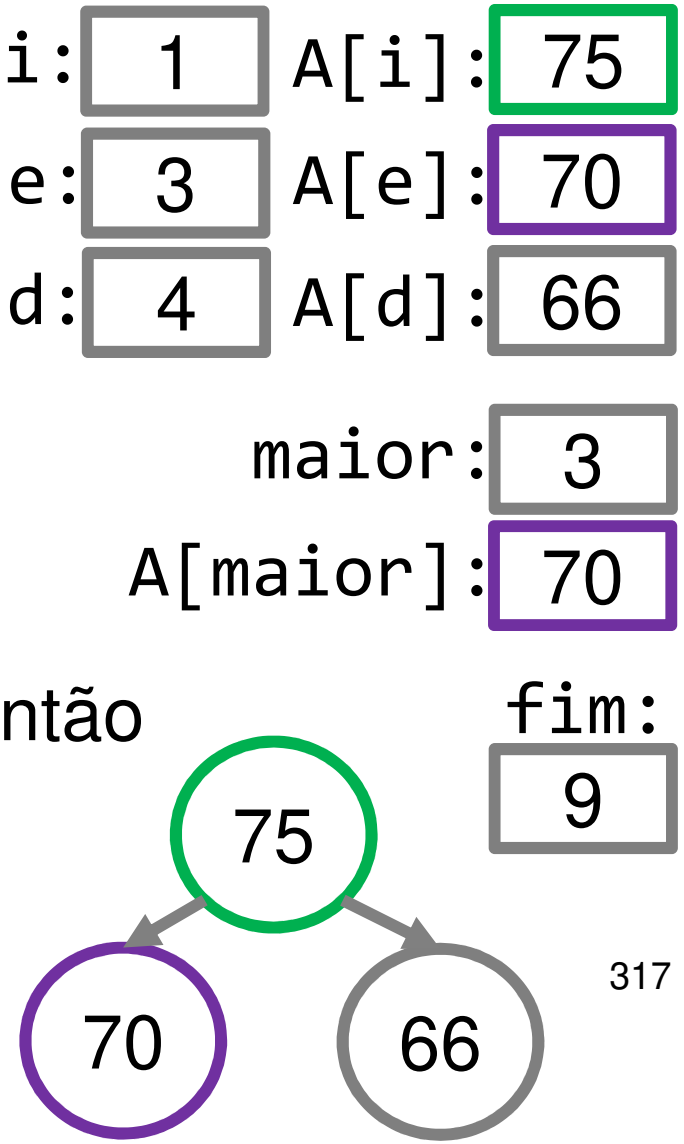
1. ***e*** ← ***2*i + 1***
2. ***d*** ← ***2*i + 2***
3. Se ***e ≤ fim*** e ***A[e] > A[i]*** então
4. ***maior*** ← ***e***
5. Senão
6. ***maior*** ← ***i***
7. Se ***d ≤ fim*** e ***A[d] > A[maior]*** então
8. ***maior*** ← ***d***
9. Se ***maior ≠ i*** então
10. troca ***A[i]*** ↔ ***A[maior]***
11. ***Heapifica***(***A, fim, maior***)



Construindo a Heap: método Heapifica()

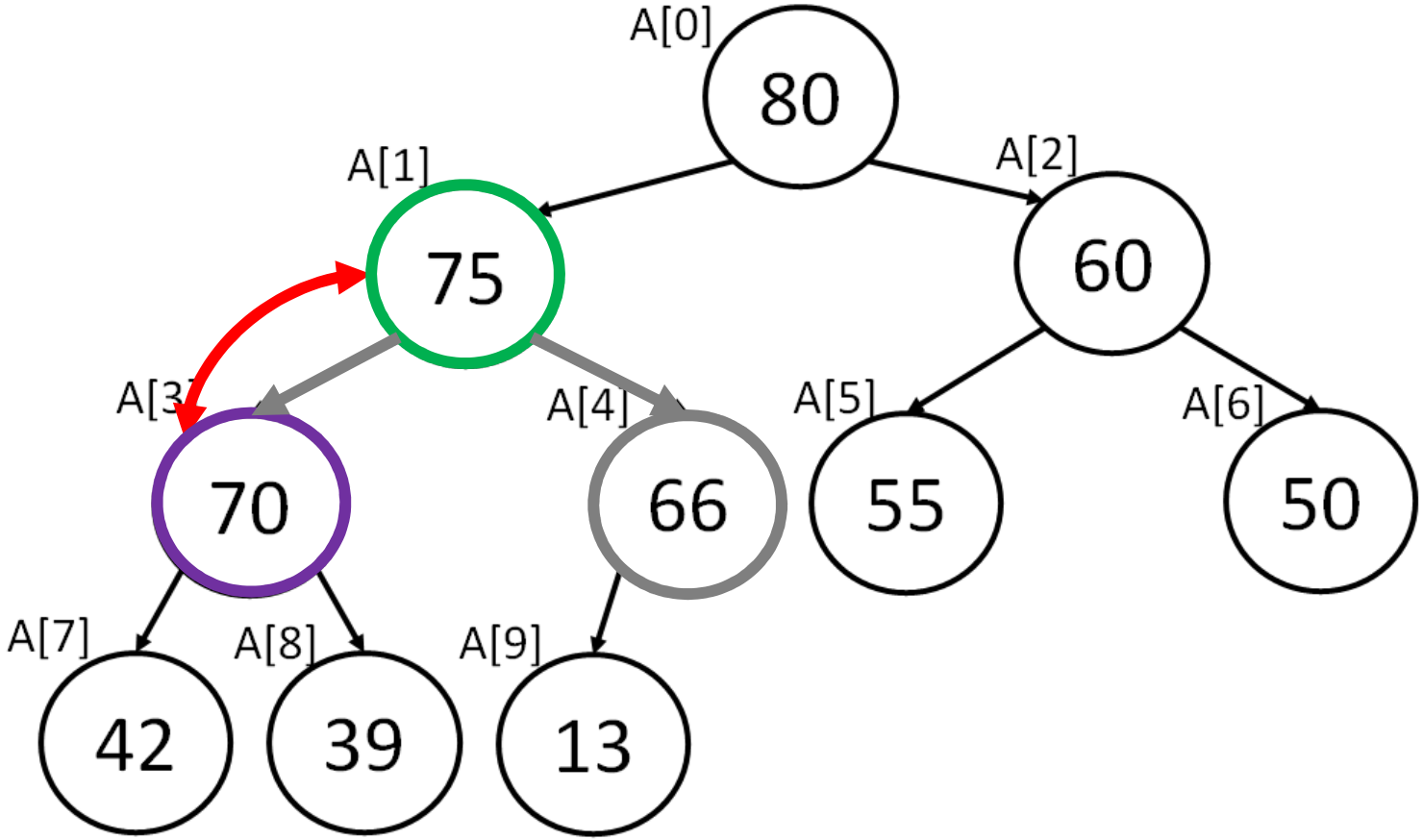
Heapifica(arranjo ***A, fim, i***)

1. ***e*** ← ***2*i + 1***
2. ***d*** ← ***2*i + 2***
3. Se ***e ≤ fim*** e ***A[e] > A[i]*** então
4. ***maior*** ← ***e***
5. Senão
6. ***maior*** ← ***i***
7. Se ***d ≤ fim*** e ***A[d] > A[maior]*** então
8. ***maior*** ← ***d***
9. Se ***maior ≠ i*** então
10. troca ***A[i]*** ↔ ***A[maior]***
11. ***Heapifica***(***A, fim, maior***)



Construindo a Heap: método Heapifica()

i	0	1	2	3	4	5	6	7	8	9
A[i]	80	75	60	70	66	55	50	42	39	13



Construindo a Heap: método Heapifica()

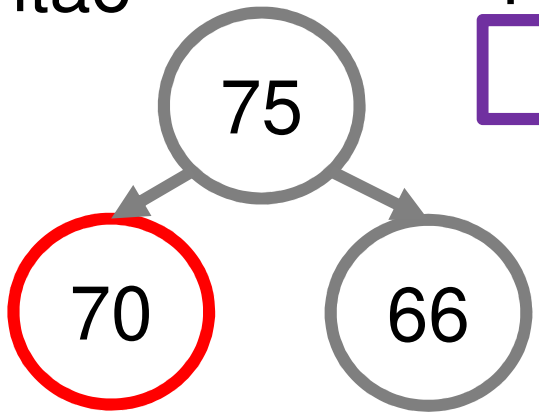
Heapifica(arranjo ***A, fim, i***)

1. ***e*** ← ***2*i + 1***
2. ***d*** ← ***2*i + 2***
3. Se ***e ≤ fim*** e ***A[e] > A[i]*** então
4. ***maior*** ← ***e***
5. Senão
6. ***maior*** ← ***i***
7. Se ***d ≤ fim*** e ***A[d] > A[maior]*** então
8. ***maior*** ← ***d***
9. Se ***maior ≠ i*** então
10. troca ***A[i]*** ↔ ***A[maior]***
11. ***Heapifica***(***A, fim, maior***)

i:	1	A[i]:	75
e:	3	A[e]:	70
d:	4	A[d]:	66

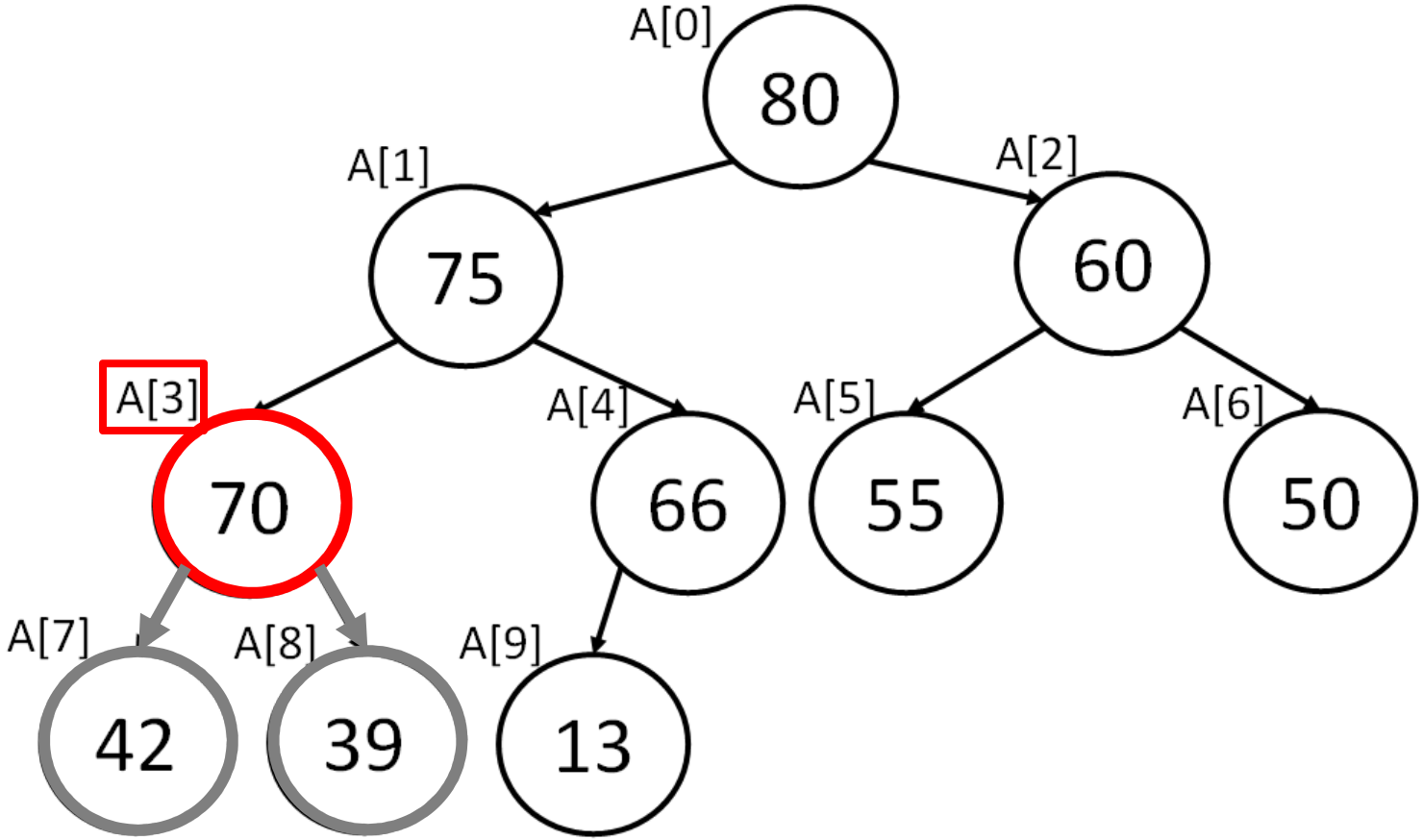
maior:	3
A[maior]:	70

fim: 9



Construindo a Heap: método Heapifica()

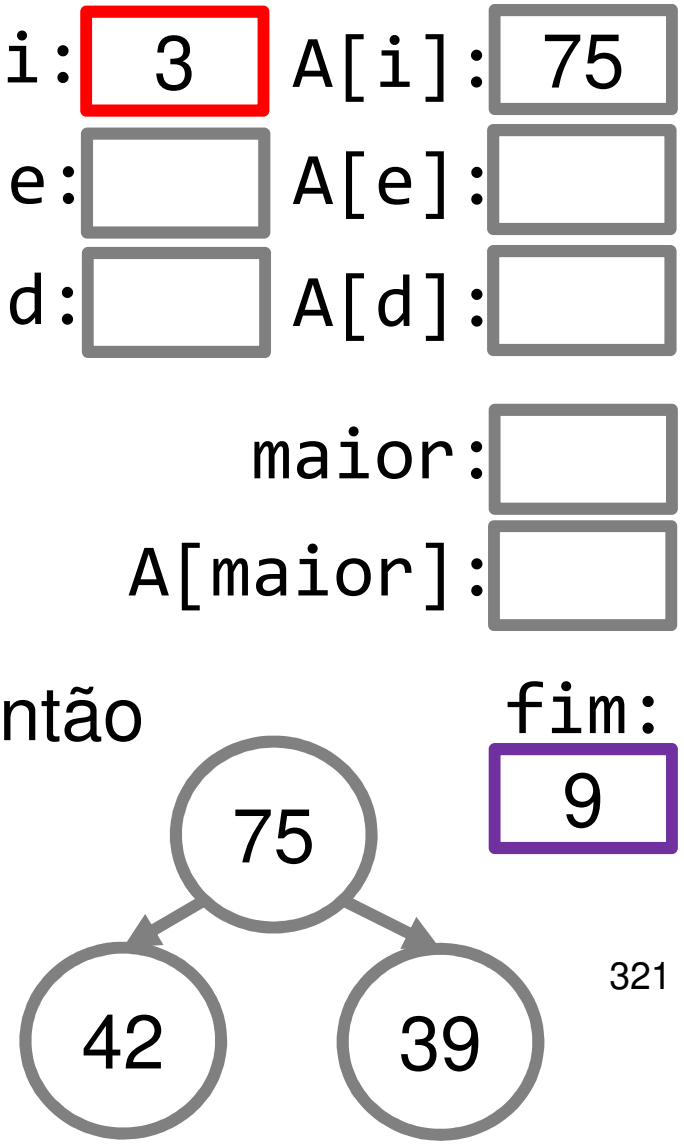
i	0	1	2	3	4	5	6	7	8	9
A[i]	80	75	60	70	66	55	50	42	39	13



Construindo a Heap: método Heapifica()

Heapifica(arranjo ***A, fim, i***)

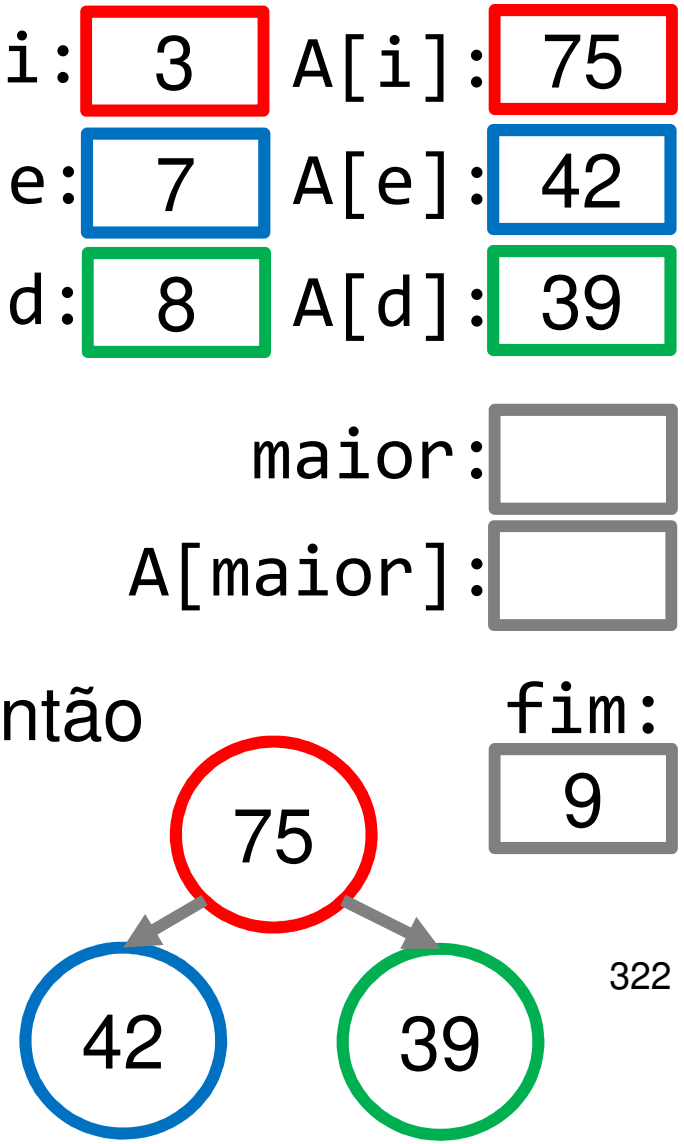
1. ***e*** ← ***2*i + 1***
2. ***d*** ← ***2*i + 2***
3. Se ***e ≤ fim*** e ***A[e] > A[i]*** então
4. ***maior*** ← ***e***
5. Senão
6. ***maior*** ← ***i***
7. Se ***d ≤ fim*** e ***A[d] > A[maior]*** então
8. ***maior*** ← ***d***
9. Se ***maior ≠ i*** então
10. troca ***A[i]*** ↔ ***A[maior]***
11. ***Heapifica***(***A, fim, maior***)



Construindo a Heap: método Heapifica()

Heapifica(arranjo ***A, fim, i***)

1. ***e*** ← ***2*i + 1***
2. ***d*** ← ***2*i + 2***
3. Se ***e ≤ fim*** e ***A[e] > A[i]*** então
4. ***maior*** ← ***e***
5. Senão
6. ***maior*** ← ***i***
7. Se ***d ≤ fim*** e ***A[d] > A[maior]*** então
8. ***maior*** ← ***d***
9. Se ***maior ≠ i*** então
10. troca ***A[i]*** ↔ ***A[maior]***
11. ***Heapifica***(***A, fim, maior***)

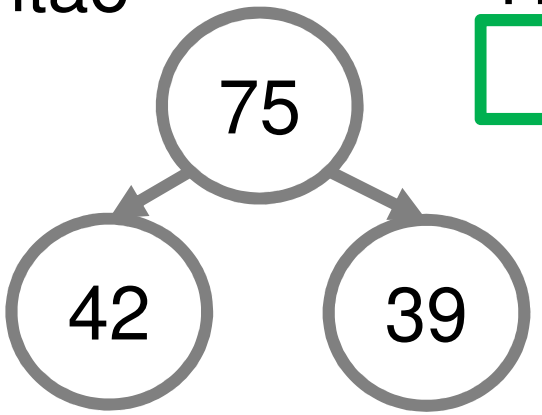


Construindo a Heap: método Heapifica()

Heapifica(arranjo ***A, fim, i***)

1. ***e*** ← ***2*i + 1***
2. ***d*** ← ***2*i + 2***
3. Se ***e ≤ fim*** e ***A[e] > A[i]*** então
4. ***maior*** ← ***e***
5. Senão
6. ***maior*** ← ***i***
7. Se ***d ≤ fim*** e ***A[d] > A[maior]*** então
8. ***maior*** ← ***d***
9. Se ***maior ≠ i*** então
10. troca ***A[i]*** ↔ ***A[maior]***
11. ***Heapifica***(***A, fim, maior***)

i:	3	A[i]:	75
e:	7	A[e]:	42
d:	8	A[d]:	39
maior:			
A[maior]:			
fim:	9		

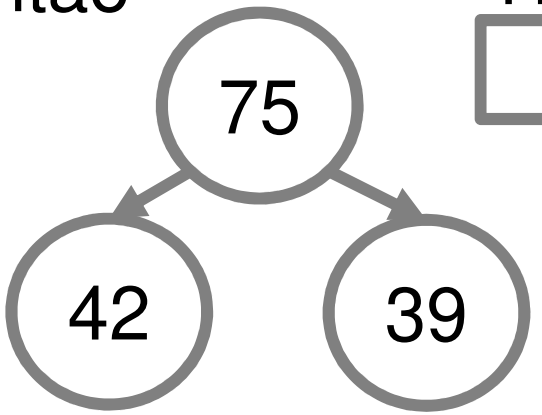


Construindo a Heap: método Heapifica()

Heapifica(arranjo ***A, fim, i***)

1. ***e*** ← ***2*i + 1***
2. ***d*** ← ***2*i + 2***
3. Se ***e ≤ fim*** e ***A[e] > A[i]*** então
4. ***maior*** ← ***e***
5. Senão
6. ***maior*** ← ***i***
7. Se ***d ≤ fim*** e ***A[d] > A[maior]*** então
8. ***maior*** ← ***d***
9. Se ***maior ≠ i*** então
10. troca ***A[i]*** ↔ ***A[maior]***
11. ***Heapifica***(***A, fim, maior***)

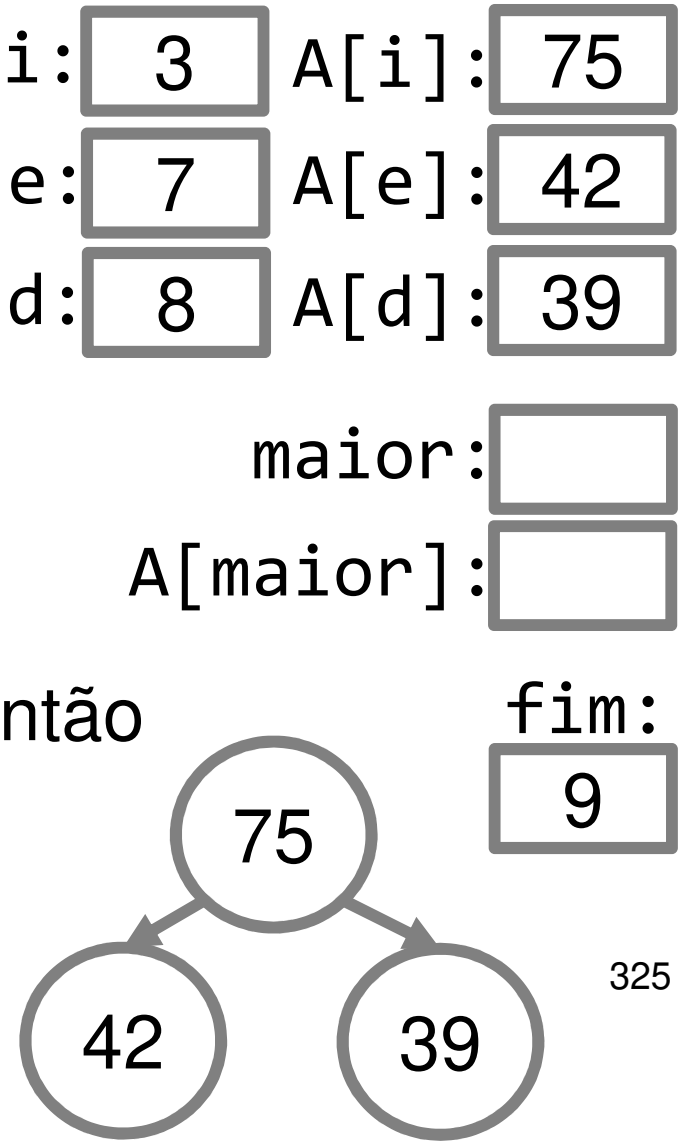
i:	3	A[i]:	75
e:	7	A[e]:	42
d:	8	A[d]:	39
maior:			
A[maior]:			
fim:	9		



Construindo a Heap: método Heapifica()

Heapifica(arranjo ***A, fim, i***)

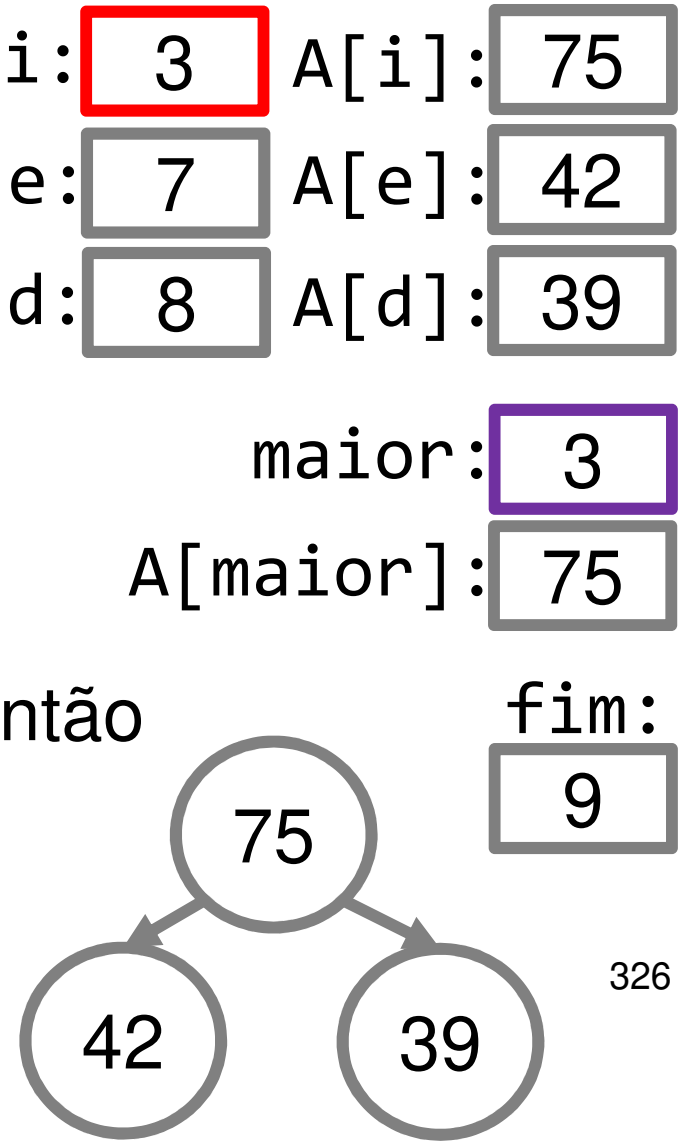
1. ***e*** ← ***2*i + 1***
2. ***d*** ← ***2*i + 2***
3. Se ***e*** ≤ ***fim*** e ***A[e]*** > ***A[i]*** então
4. ***maior*** ← ***e***
5. Senão
6. ***maior*** ← ***i***
7. Se ***d*** ≤ ***fim*** e ***A[d]*** > ***A[maior]*** então
8. ***maior*** ← ***d***
9. Se ***maior*** ≠ ***i*** então
10. troca ***A[i]*** ↔ ***A[maior]***
11. ***Heapifica***(***A, fim, maior***)



Construindo a Heap: método Heapifica()

Heapifica(arranjo ***A, fim, i***)

1. ***e*** ← ***2*i + 1***
2. ***d*** ← ***2*i + 2***
3. Se ***e ≤ fim*** e ***A[e] > A[i]*** então
4. ***maior*** ← ***e***
5. Senão
6. ***maior*** ← ***i***
7. Se ***d ≤ fim*** e ***A[d] > A[maior]*** então
8. ***maior*** ← ***d***
9. Se ***maior ≠ i*** então
10. troca ***A[i]*** ↔ ***A[maior]***
11. ***Heapifica***(***A, fim, maior***)

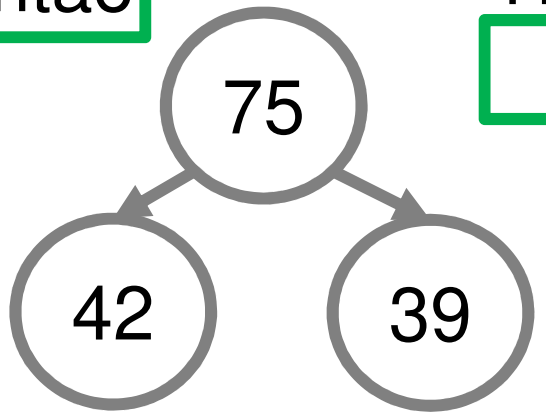


Construindo a Heap: método Heapifica()

Heapifica(arranjo ***A, fim, i***)

1. ***e*** ← ***2*i + 1***
2. ***d*** ← ***2*i + 2***
3. Se ***e ≤ fim*** e ***A[e] > A[i]*** então
4. ***maior*** ← ***e***
5. Senão
6. ***maior*** ← ***i***
7. Se ***d ≤ fim*** e ***A[d] > A[maior]*** então
8. ***maior*** ← ***d***
9. Se ***maior ≠ i*** então
10. troca ***A[i]*** ↔ ***A[maior]***
11. ***Heapifica***(***A, fim, maior***)

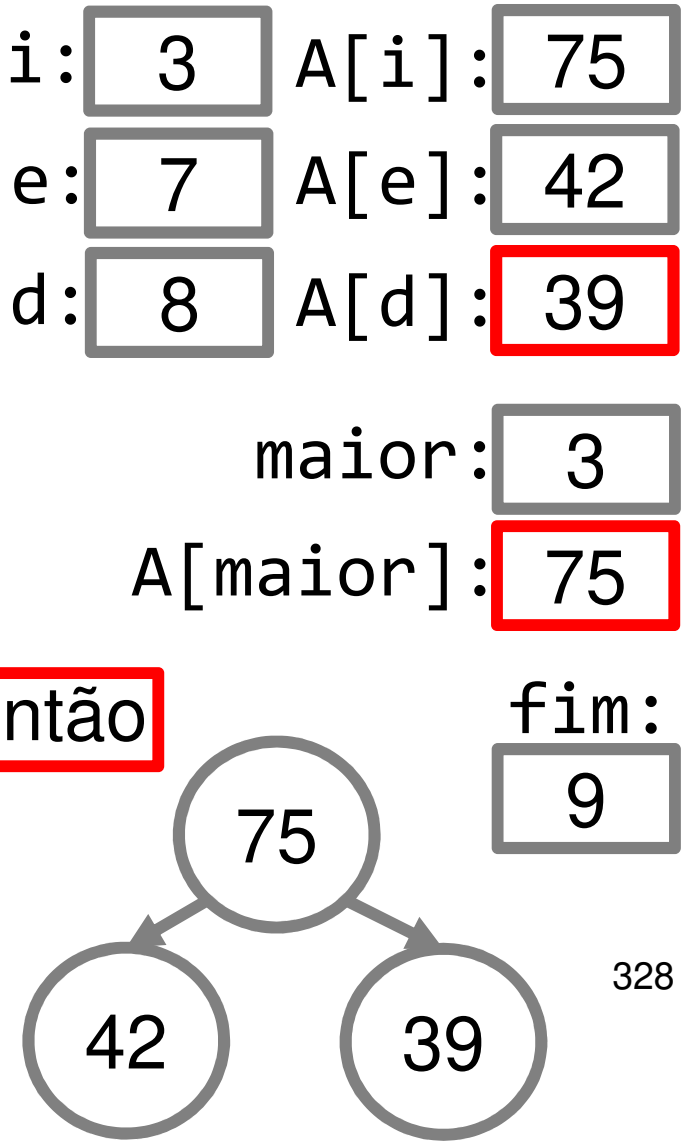
i:	3	A[i]:	75
e:	7	A[e]:	42
d:	8	A[d]:	39
maior:	3		
A[maior]:	75		
fim:	9		



Construindo a Heap: método Heapifica()

Heapifica(arranjo ***A, fim, i***)

1. ***e*** ← ***2*i + 1***
2. ***d*** ← ***2*i + 2***
3. Se ***e ≤ fim*** e ***A[e] > A[i]*** então
4. ***maior*** ← ***e***
5. Senão
6. ***maior*** ← ***i***
7. Se ***d ≤ fim*** e ***A[d] > A[maior]*** então
8. ***maior*** ← ***d***
9. Se ***maior ≠ i*** então
10. troca ***A[i]*** ↔ ***A[maior]***
11. ***Heapifica***(***A, fim, maior***)



Construindo a Heap: método Heapifica()

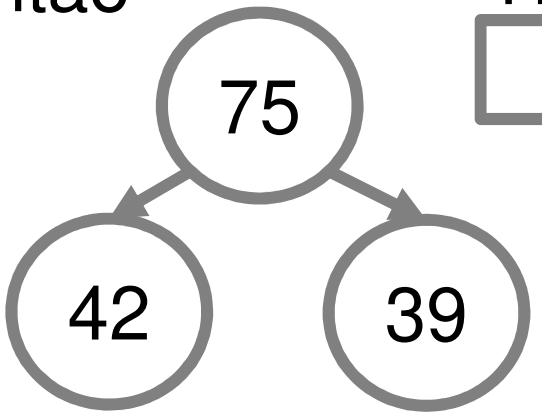
Heapifica(arranjo ***A, fim, i***)

1. ***e*** ← ***2*i + 1***
2. ***d*** ← ***2*i + 2***
3. Se ***e*** ≤ ***fim*** e ***A[e] > A[i]*** então
4. ***maior*** ← ***e***
5. Senão
6. ***maior*** ← ***i***
7. Se ***d*** ≤ ***fim*** e ***A[d] > A[maior]*** então
8. ***maior*** ← ***d***
9. Se ***maior*** ≠ ***i*** então
10. troca ***A[i]*** ↔ ***A[maior]***
11. ***Heapifica***(***A, fim, maior***)

i:	3	A[i]:	75
e:	7	A[e]:	42
d:	8	A[d]:	39

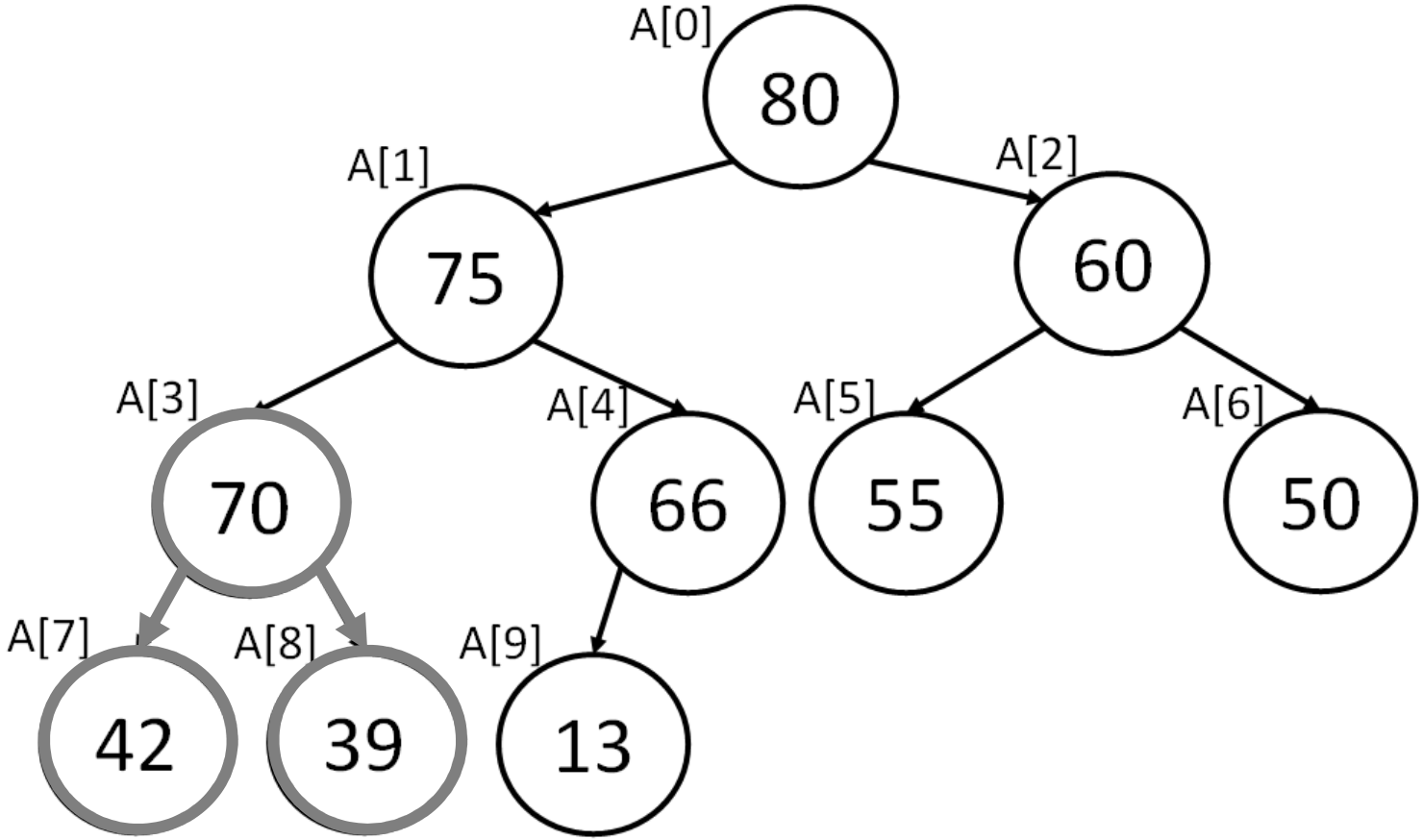
maior:	3
A[maior]:	75

fim: 9



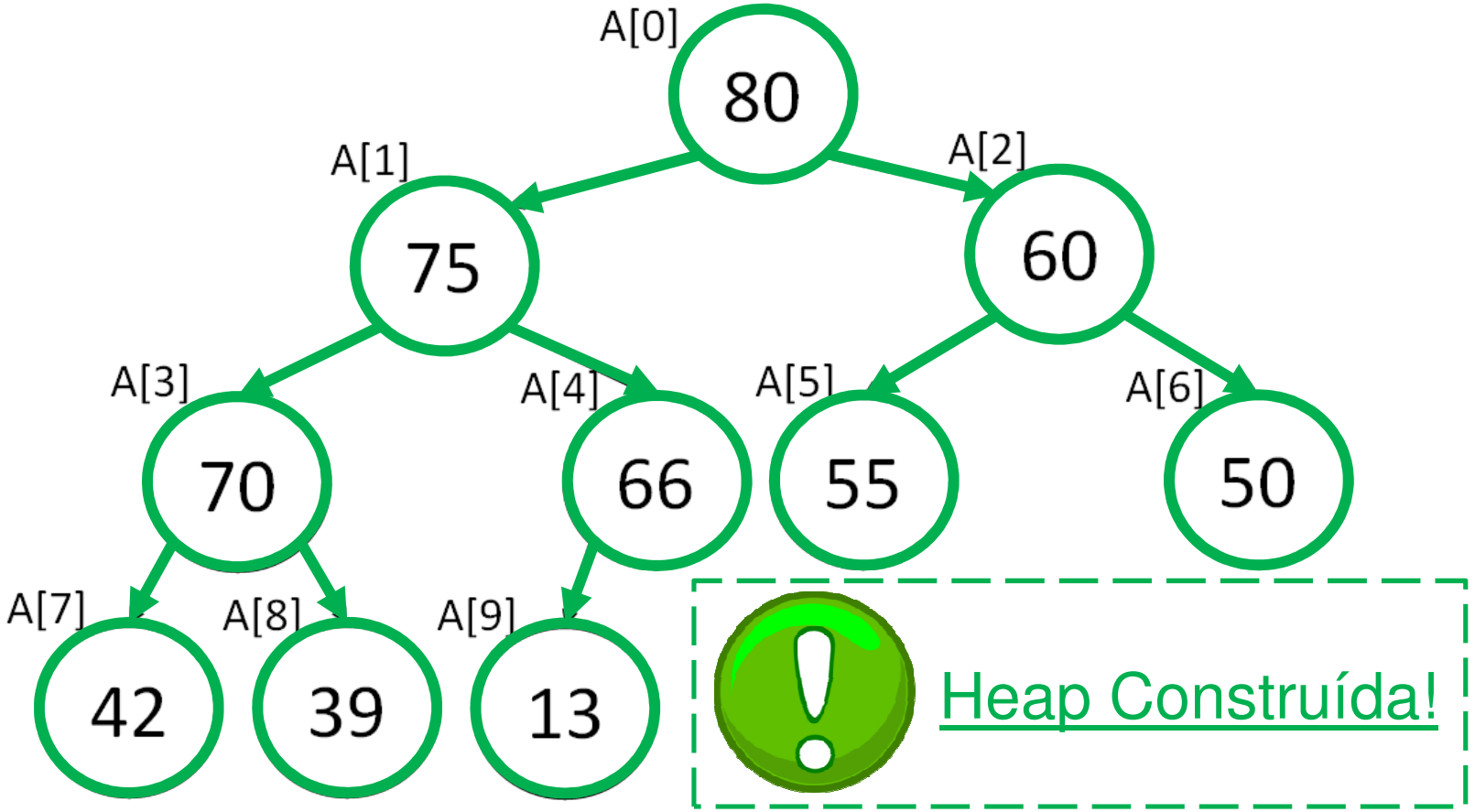
Construindo a Heap : método Heapifica()

i	0	1	2	3	4	5	6	7	8	9
A[i]	80	75	60	70	66	55	50	42	39	13



Primeiro Passo Completo: Heap Construída!

i	0	1	2	3	4	5	6	7	8	9
A[i]	80	75	60	70	66	55	50	42	39	13



UniCesumar

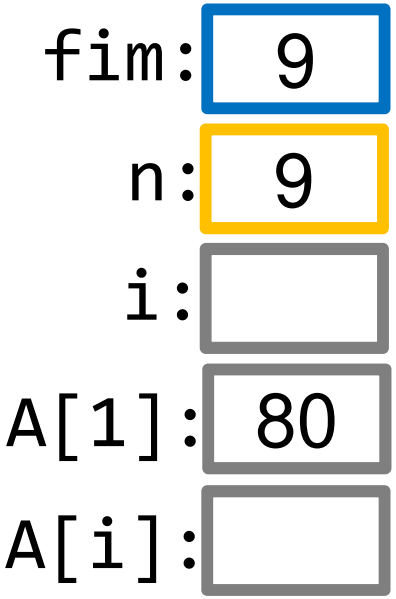
Segunda Etapa: Ordenando Pela Raiz da Heap

Heapsort() + Heapifica()

Segundo Passo: Ordenando Pela Raiz

Heapsort(arranjo ***A, fim***)

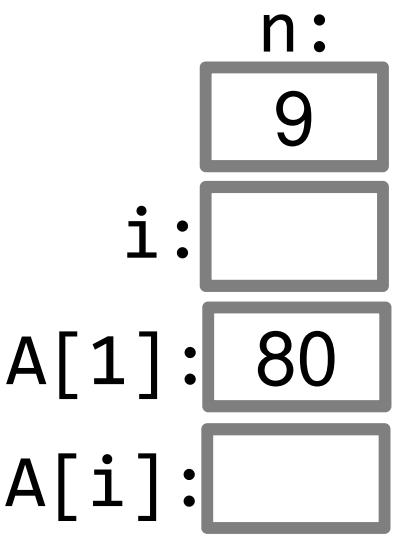
1. ***n*** ← ***fim***
2. ***ConstroiHeap***(***A, fim***)
3. Para ***i*** de ***n*** até ***1*** faça
4. troca ***A[1]*** ↔ ***A[i]***
5. ***n*** ← ***n-1***
6. ***Heapifica***(***A, n, 0***)



Segundo Passo: Ordenando Pela Raiz

Heapsort(arranjo ***A, fim***)

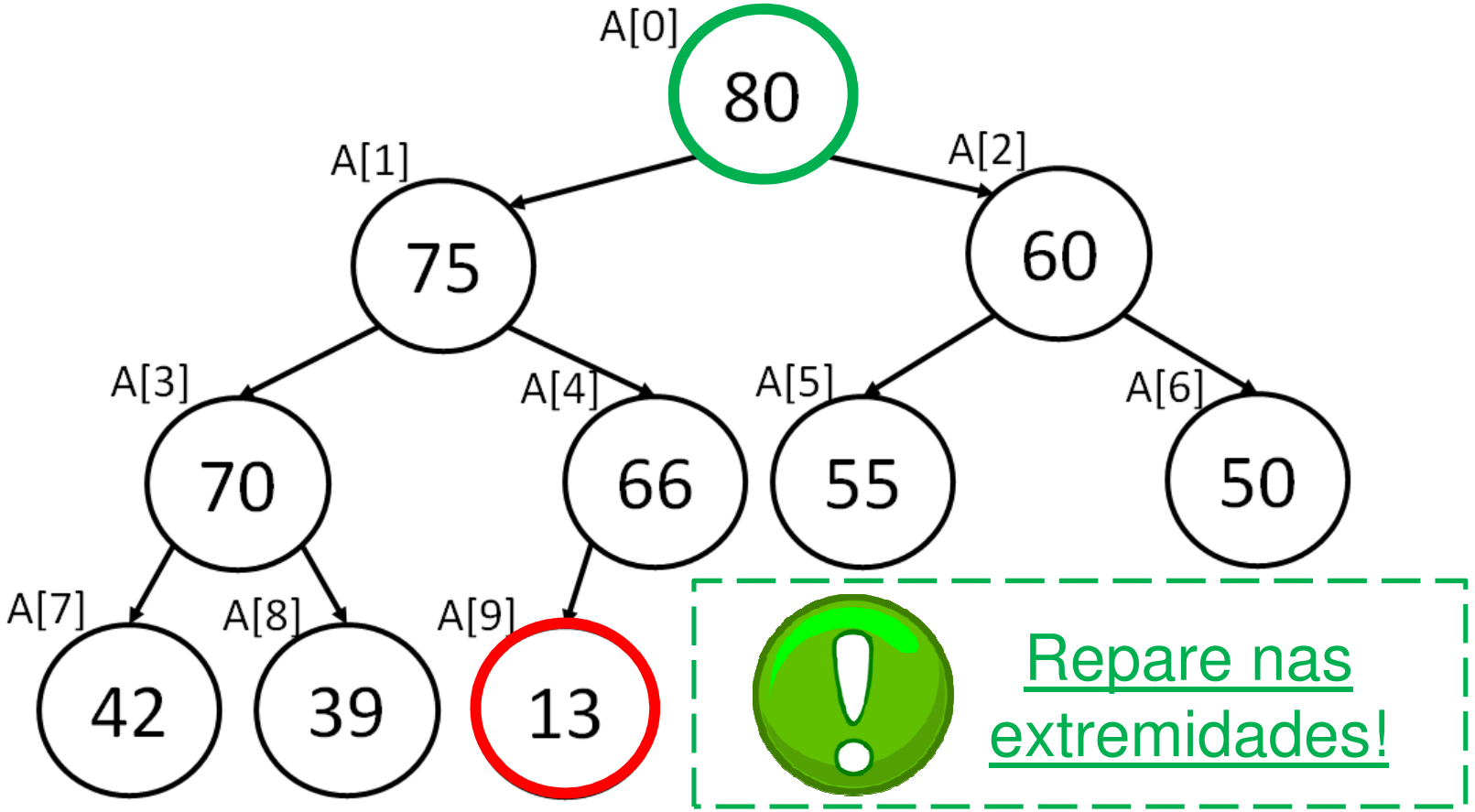
1. ***n*** ← ***fim***
2. ***ConstroiHeap***(***A, fim***)
3. Para ***i*** de ***n*** até ***1*** faça
4. troca ***A[1]*** ↔ ***A[i]***
5. ***n*** ← ***n-1***
6. ***Heapifica***(***A, n, 0***)



O ConstroiHeap() foi recém demonstrado, nos slides anteriores

Ordenando Pela Raiz: Método Heapsort()

i	0	1	2	3	4	5	6	7	8	9
A[i]	80	75	60	70	66	55	50	42	39	13

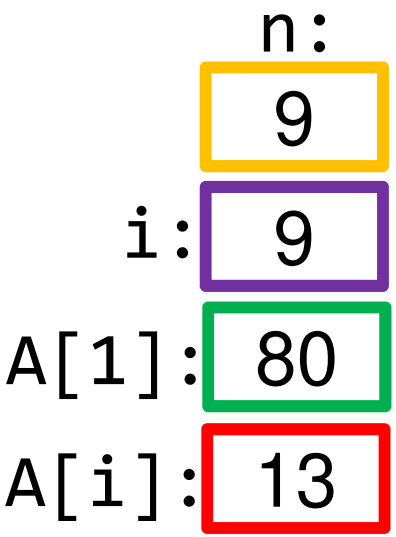


Repare nas extremidades!

Ordenando Pela Raiz: Método Heapsort()

Heapsort(arranjo ***A, fim***)

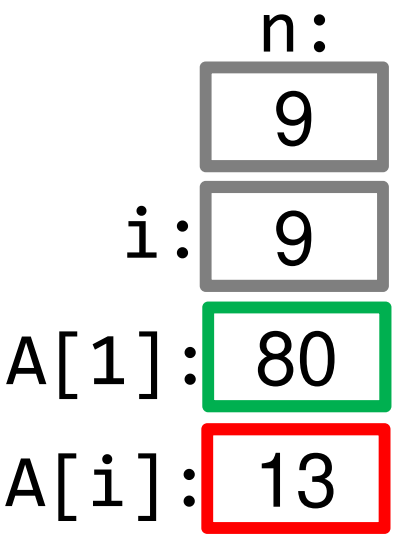
1. ***n*** ← ***fim***
2. ***ConstroiHeap***(***A, fim***)
3. Para ***i*** de ***n*** até ***1*** faça
4. troca ***A[1]*** ↔ ***A[i]***
5. ***n*** ← ***n-1***
6. ***Heapifica***(***A, n, 0***)



Ordenando Pela Raiz: Método Heapsort()

Heapsort(arranjo ***A, fim***)

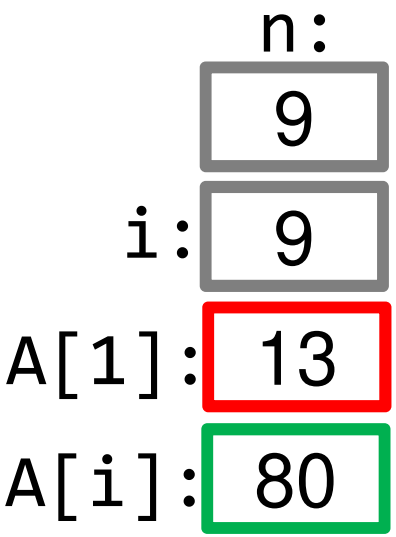
1. ***n*** ← ***fim***
2. ***ConstroiHeap***(***A, fim***)
3. Para ***i*** de ***n*** até ***1*** faça
4. troca ***A[1]*** ↔ ***A[i]***
5. ***n*** ← ***n-1***
6. ***Heapifica***(***A, n, 0***)



Ordenando Pela Raiz: Método Heapsort()

Heapsort(arranjo ***A, fim***)

1. ***n*** ← ***fim***
2. ***ConstroiHeap***(***A, fim***)
3. Para ***i*** de ***n*** até ***1*** faça
4. troca ***A[1]*** ↔ ***A[i]***
5. ***n*** ← ***n-1***
6. ***Heapifica***(***A, n, 0***)



Ordenando Pela Raiz: Método Heapsort()

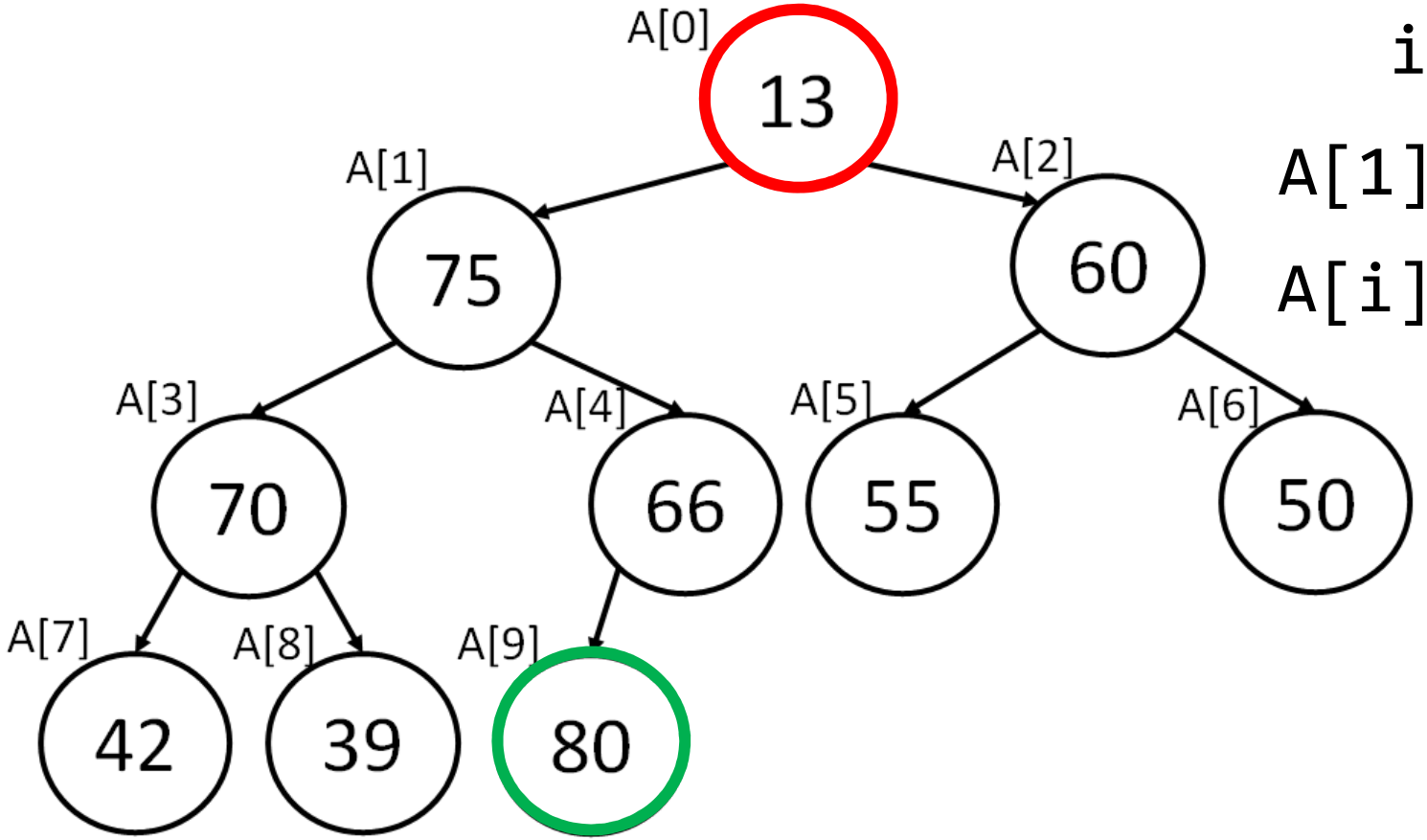
i	0	1	2	3	4	5	6	7	8	9
A[i]	13	75	60	70	66	55	50	42	39	80

n: 9

i: 9

A[1]: 13

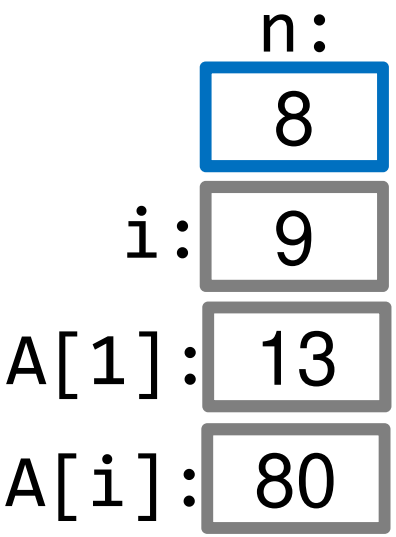
A[i]: 80



Ordenando Pela Raiz: Método Heapsort()

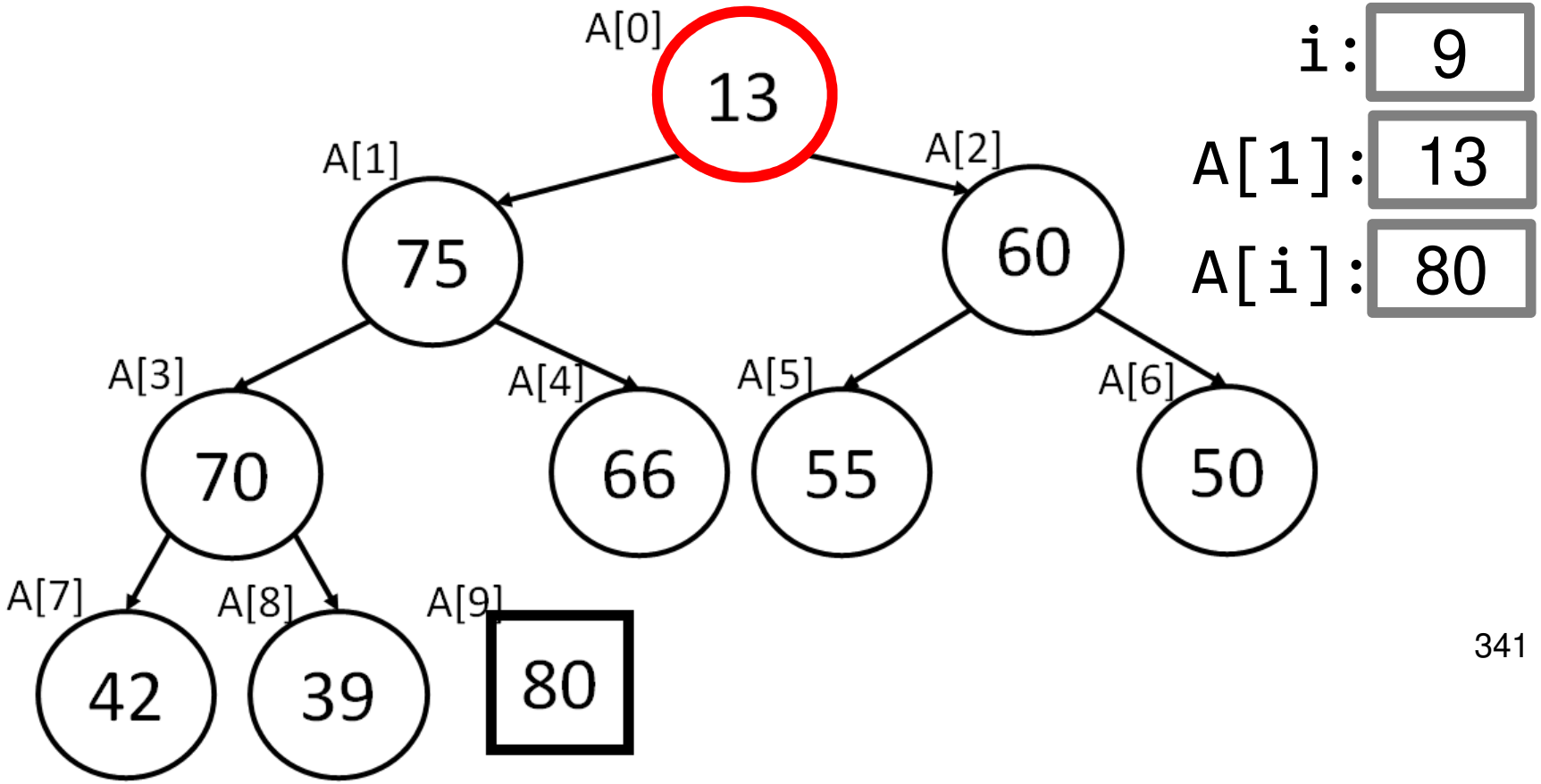
Heapsort(arranjo ***A, fim***)

1. ***n*** ← ***fim***
2. ***ConstroiHeap***(***A, fim***)
3. Para ***i*** de ***n*** até ***1*** faça
4. troca ***A[1]*** ↔ ***A[i]***
5. ***n*** ← ***n-1***
6. ***Heapifica***(***A, n, 0***)



Ordenando Pela Raiz: Método Heapsort()

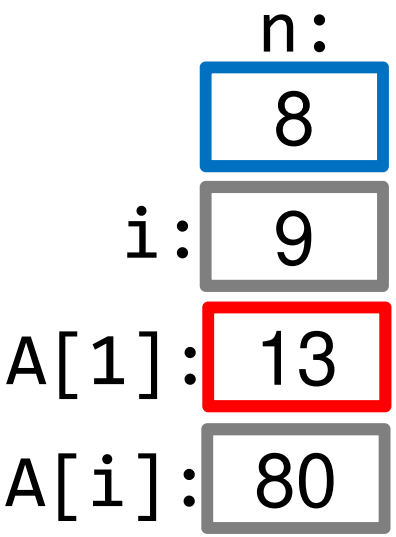
i	0	1	2	3	4	5	6	7	8	9
A[i]	13	75	60	70	66	55	50	42	39	80



Ordenando Pela Raiz: Método Heapsort()

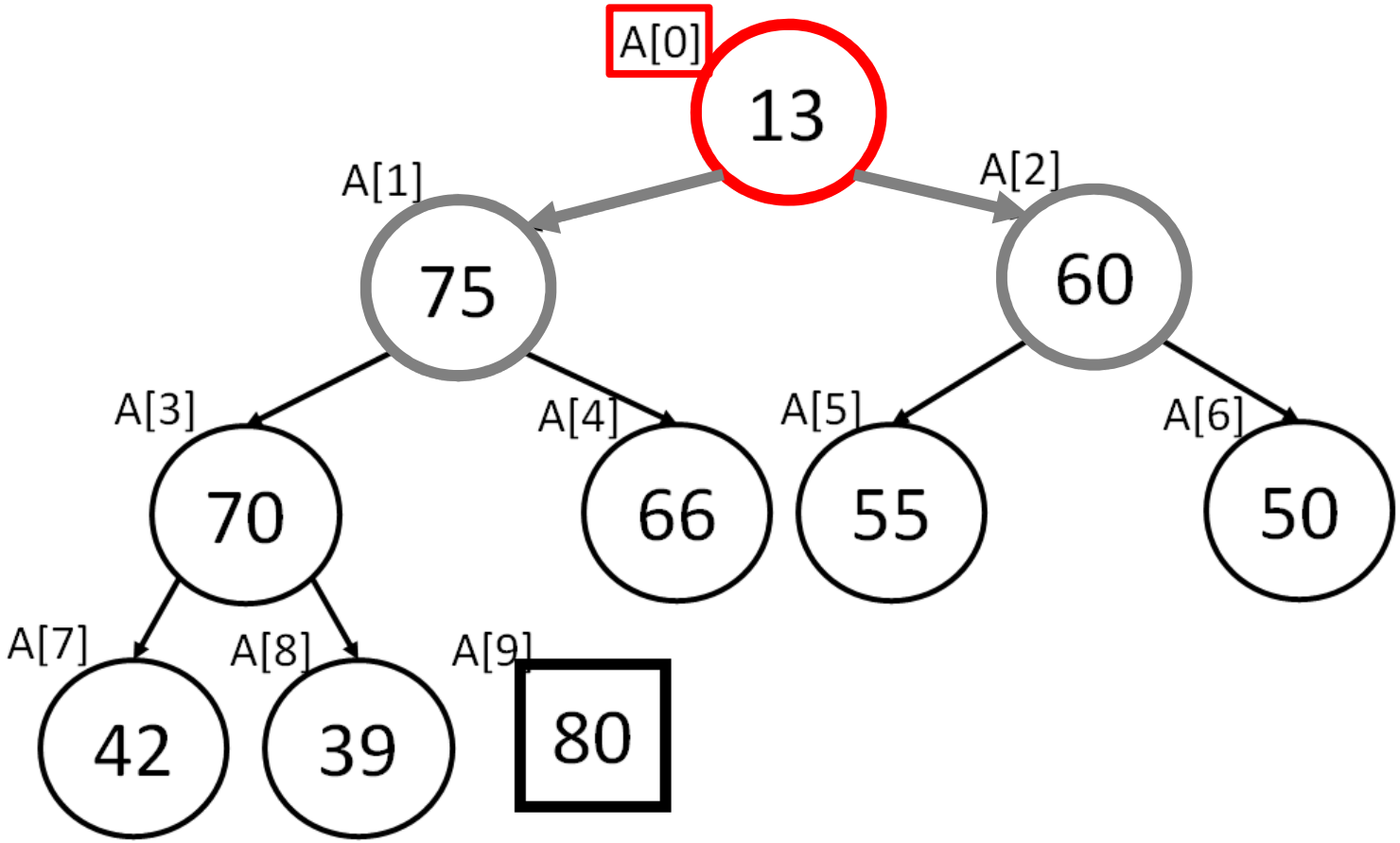
Heapsort(arranjo ***A, fim***)

1. ***n*** ← ***fim***
2. ***ConstroiHeap***(***A, fim***)
3. Para ***i*** de ***n*** até ***1*** faça
4. troca ***A[1]*** ↔ ***A[i]***
5. ***n*** ← ***n-1***
6. ***Heapifica***(***A, n, 0***)



Ordenando Pela Raiz: Método Heapifica()

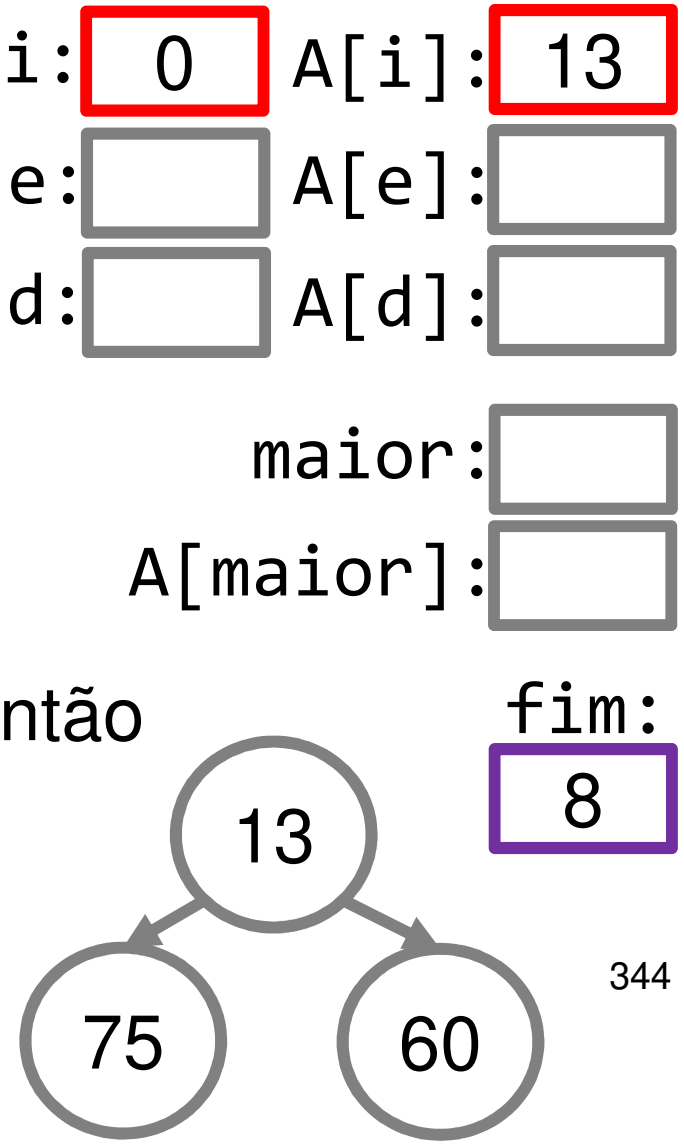
i	0	1	2	3	4	5	6	7	8	9
A[i]	13	75	60	70	66	55	50	42	39	80



Construindo a Heap: método Heapifica()

Heapifica(arranjo ***A, fim, i***)

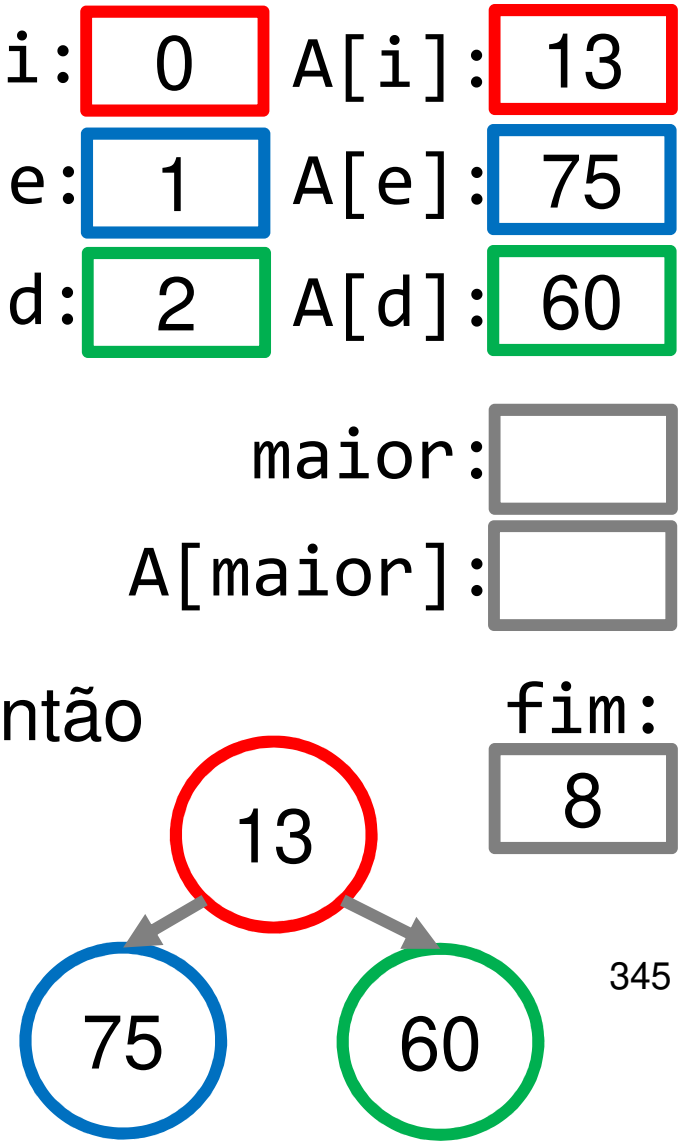
1. ***e*** ← ***2*i + 1***
2. ***d*** ← ***2*i + 2***
3. Se ***e ≤ fim*** e ***A[e] > A[i]*** então
4. ***maior*** ← ***e***
5. Senão
6. ***maior*** ← ***i***
7. Se ***d ≤ fim*** e ***A[d] > A[maior]*** então
8. ***maior*** ← ***d***
9. Se ***maior ≠ i*** então
10. troca ***A[i]*** ↔ ***A[maior]***
11. ***Heapifica***(***A, fim, maior***)



Construindo a Heap: método Heapifica()

Heapifica(arranjo ***A, fim, i***)

1. ***e*** ← ***2*i + 1***
2. ***d*** ← ***2*i + 2***
3. Se ***e ≤ fim*** e ***A[e] > A[i]*** então
4. ***maior*** ← ***e***
5. Senão
6. ***maior*** ← ***i***
7. Se ***d ≤ fim*** e ***A[d] > A[maior]*** então
8. ***maior*** ← ***d***
9. Se ***maior ≠ i*** então
10. troca ***A[i]*** ↔ ***A[maior]***
11. ***Heapifica***(***A, fim, maior***)

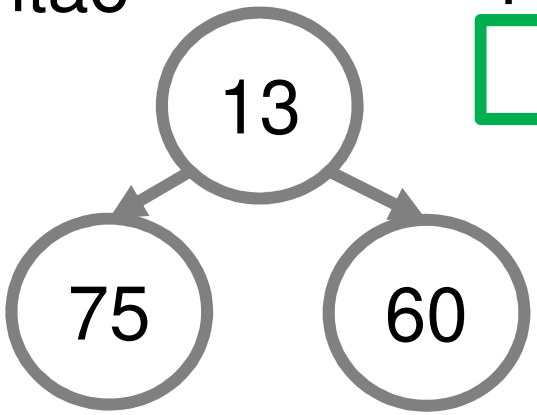


Construindo a Heap: método Heapifica()

Heapifica(arranjo ***A, fim, i***)

1. ***e*** ← ***2*i + 1***
2. ***d*** ← ***2*i + 2***
3. Se ***e ≤ fim*** e ***A[e] > A[i]*** então
4. ***maior*** ← ***e***
5. Senão
6. ***maior*** ← ***i***
7. Se ***d ≤ fim*** e ***A[d] > A[maior]*** então
8. ***maior*** ← ***d***
9. Se ***maior ≠ i*** então
10. troca ***A[i]*** ↔ ***A[maior]***
11. ***Heapifica***(***A, fim, maior***)

i:	0	A[i]:	13
e:	1	A[e]:	75
d:	2	A[d]:	60
maior:			
A[maior]:			
fim:	8		



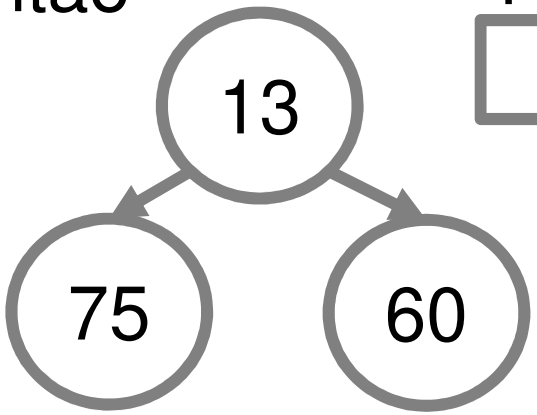
Construindo a Heap: método Heapifica()

Heapifica(arranjo ***A, fim, i***)

1. ***e*** ← ***2*i + 1***
2. ***d*** ← ***2*i + 2***
3. Se ***e ≤ fim*** e ***A[e] > A[i]*** então
4. ***maior*** ← ***e***
5. Senão
6. ***maior*** ← ***i***
7. Se ***d ≤ fim*** e ***A[d] > A[maior]*** então
8. ***maior*** ← ***d***
9. Se ***maior ≠ i*** então
10. troca ***A[i]*** ↔ ***A[maior]***
11. ***Heapifica***(***A, fim, maior***)

i:	0	A[i]:	13
e:	1	A[e]:	75
d:	2	A[d]:	60
maior:			
A[maior]:			

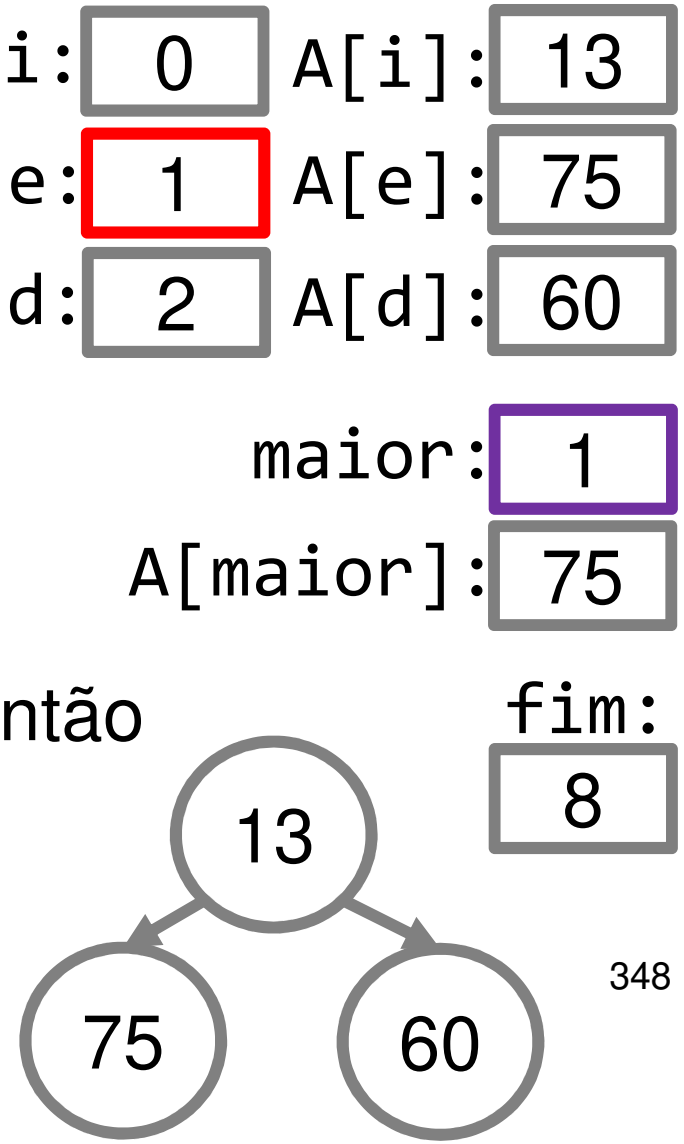
fim: 8



Construindo a Heap: método Heapifica()

Heapifica(arranjo ***A, fim, i***)

1. ***e*** ← ***2*i + 1***
2. ***d*** ← ***2*i + 2***
3. Se ***e ≤ fim*** e ***A[e] > A[i]*** então
4. ***maior*** ← ***e***
5. Senão
6. ***maior*** ← ***i***
7. Se ***d ≤ fim*** e ***A[d] > A[maior]*** então
8. ***maior*** ← ***d***
9. Se ***maior ≠ i*** então
10. troca ***A[i]*** ↔ ***A[maior]***
11. ***Heapifica***(***A, fim, maior***)

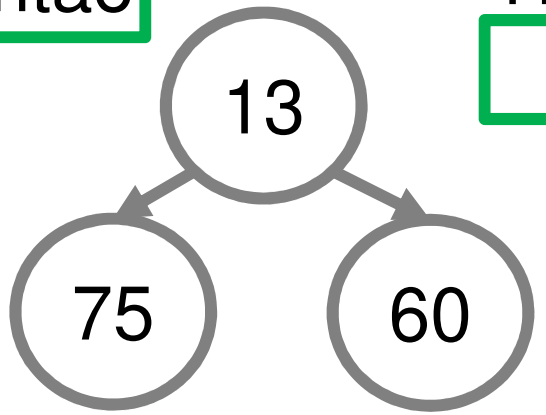


Construindo a Heap: método Heapifica()

Heapifica(arranjo ***A, fim, i***)

1. ***e*** ← ***2*i + 1***
2. ***d*** ← ***2*i + 2***
3. Se ***e ≤ fim*** e ***A[e] > A[i]*** então
4. ***maior*** ← ***e***
5. Senão
6. ***maior*** ← ***i***
7. Se ***d ≤ fim*** e ***A[d] > A[maior]*** então
8. ***maior*** ← ***d***
9. Se ***maior ≠ i*** então
10. troca ***A[i]*** ↔ ***A[maior]***
11. ***Heapifica***(***A, fim, maior***)

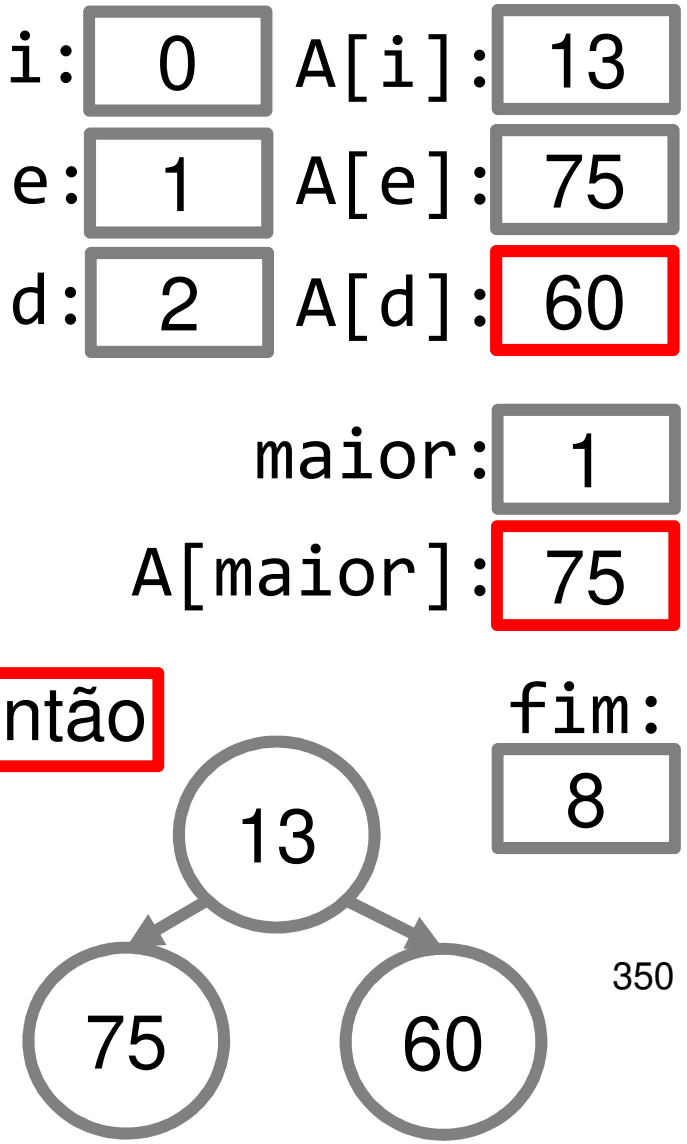
i:	0	A[i]:	13
e:	1	A[e]:	75
d:	2	A[d]:	60
maior:	1		
A[maior]:	75		
fim:	8		



Construindo a Heap: método Heapifica()

Heapifica(arranjo ***A, fim, i***)

1. ***e*** ← ***2*i + 1***
2. ***d*** ← ***2*i + 2***
3. Se ***e*** ≤ ***fim*** e ***A[e] > A[i]*** então
4. ***maior*** ← ***e***
5. Senão
6. ***maior*** ← ***i***
7. Se ***d*** ≤ ***fim*** e ***A[d] > A[maior]*** então
8. ***maior*** ← ***d***
9. Se ***maior*** ≠ ***i*** então
10. troca ***A[i]*** ↔ ***A[maior]***
11. ***Heapifica***(***A, fim, maior***)



Construindo a Heap: método Heapifica()

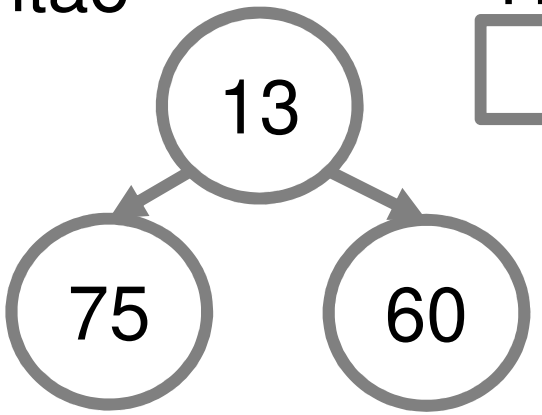
Heapifica(arranjo ***A, fim, i***)

1. ***e*** ← ***2*i + 1***
2. ***d*** ← ***2*i + 2***
3. Se ***e*** ≤ ***fim*** e ***A[e] > A[i]*** então
4. ***maior*** ← ***e***
5. Senão
6. ***maior*** ← ***i***
7. Se ***d*** ≤ ***fim*** e ***A[d] > A[maior]*** então
8. ***maior*** ← ***d***
9. Se ***maior*** ≠ ***i*** então
10. troca ***A[i]*** ↔ ***A[maior]***
11. ***Heapifica***(***A, fim, maior***)

i:	0	A[i]:	13
e:	1	A[e]:	75
d:	2	A[d]:	60

maior:	1
A[maior]:	75

fim: 8

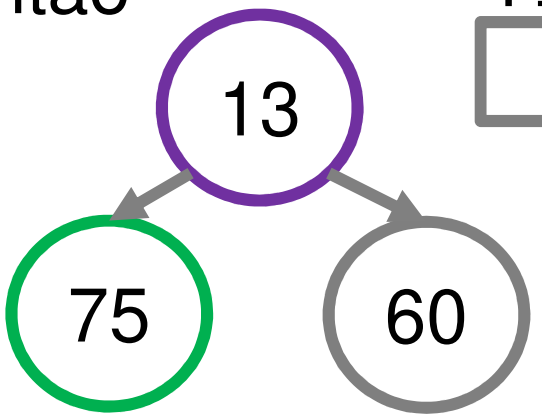


Construindo a Heap: método Heapifica()

Heapifica(arranjo ***A, fim, i***)

1. ***e*** ← ***2*i + 1***
2. ***d*** ← ***2*i + 2***
3. Se ***e ≤ fim*** e ***A[e] > A[i]*** então
4. ***maior*** ← ***e***
5. Senão
6. ***maior*** ← ***i***
7. Se ***d ≤ fim*** e ***A[d] > A[maior]*** então
8. ***maior*** ← ***d***
9. Se ***maior ≠ i*** então
10. troca ***A[i]*** ↔ ***A[maior]***
11. ***Heapifica***(***A, fim, maior***)

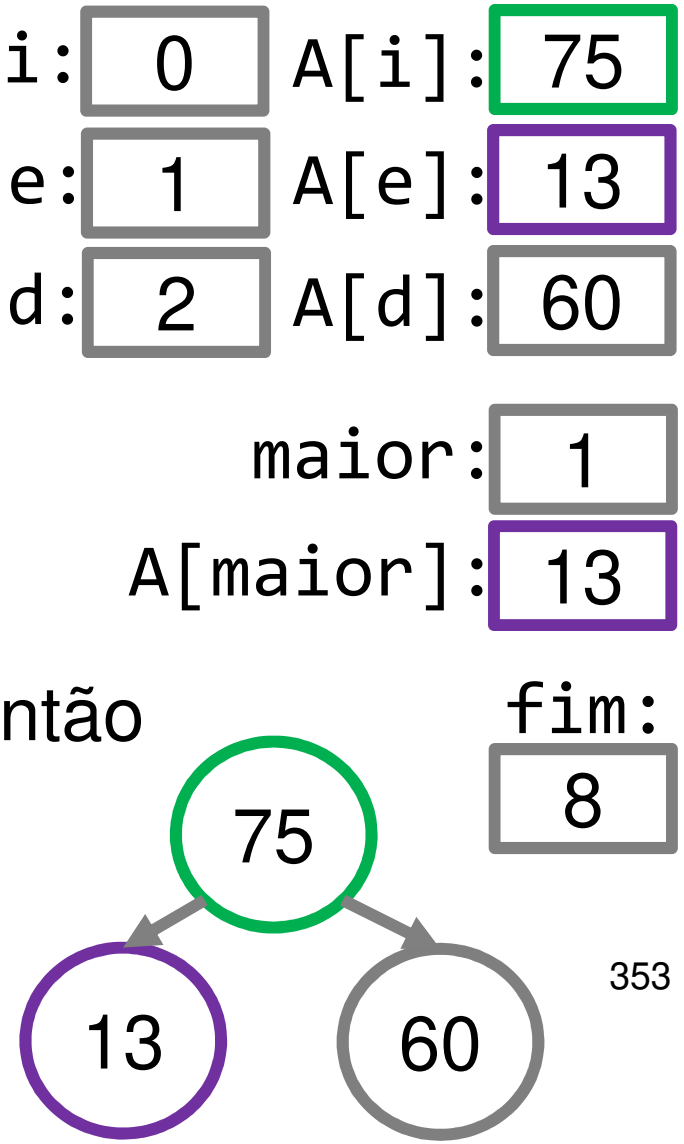
i:	0	A[i]:	13
e:	1	A[e]:	75
d:	2	A[d]:	60
maior:	1		
A[maior]:	75		
fim:	8		



Construindo a Heap: método Heapifica()

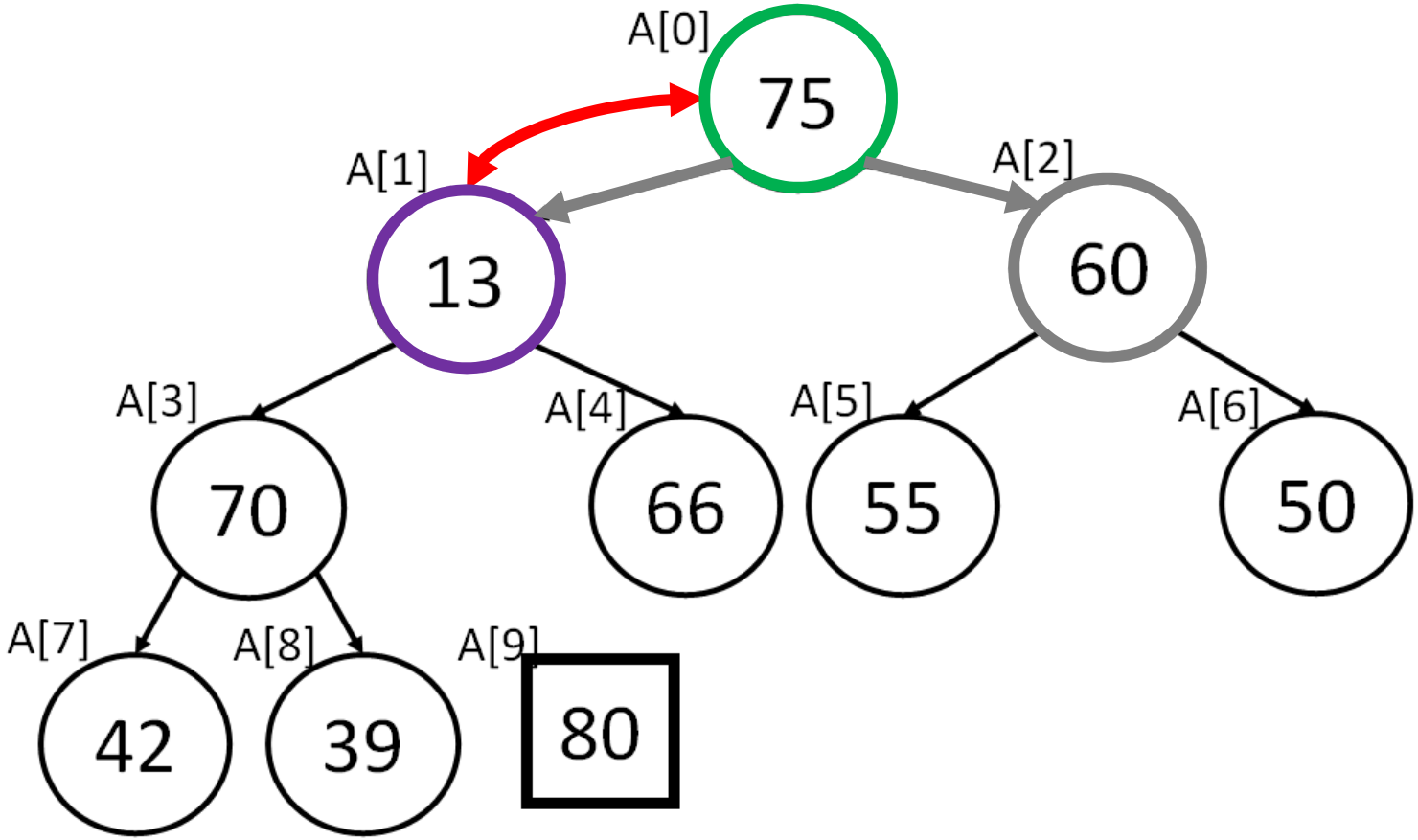
Heapifica(arranjo ***A, fim, i***)

1. ***e*** ← ***2*i + 1***
2. ***d*** ← ***2*i + 2***
3. Se ***e ≤ fim*** e ***A[e] > A[i]*** então
4. ***maior*** ← ***e***
5. Senão
6. ***maior*** ← ***i***
7. Se ***d ≤ fim*** e ***A[d] > A[maior]*** então
8. ***maior*** ← ***d***
9. Se ***maior ≠ i*** então
10. troca ***A[i]*** ↔ ***A[maior]***
11. ***Heapifica***(***A, fim, maior***)



Construindo a Heap: método Heapifica()

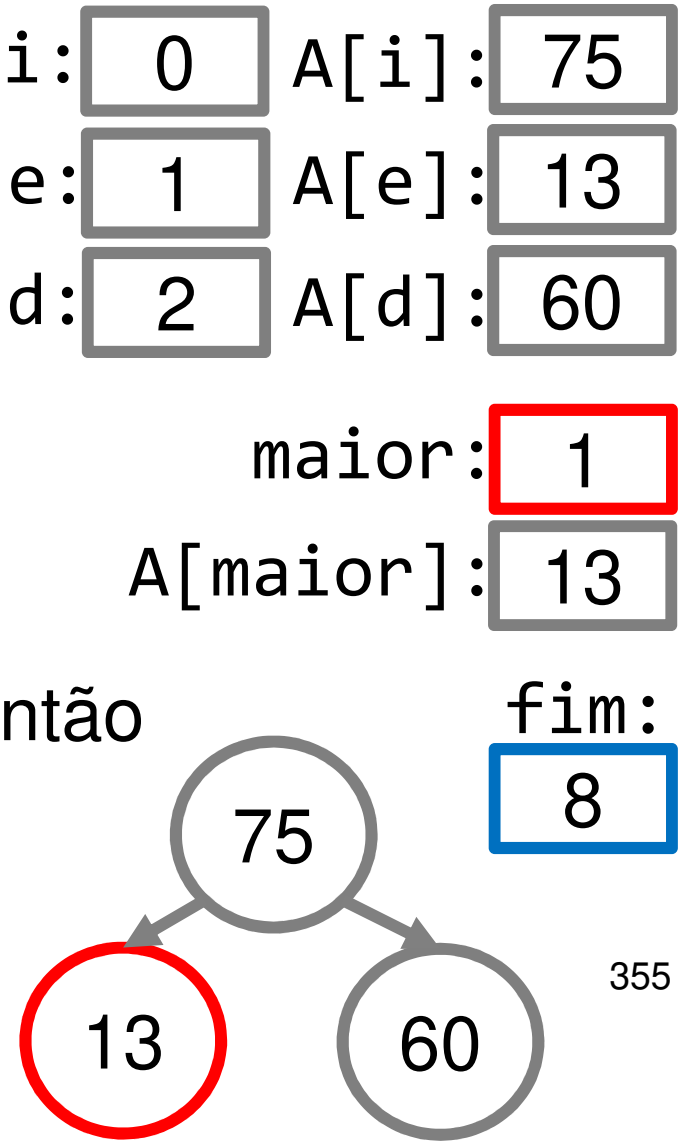
i	0	1	2	3	4	5	6	7	8	9
A[i]	75	13	60	70	66	55	50	42	39	80



Construindo a Heap: método Heapifica()

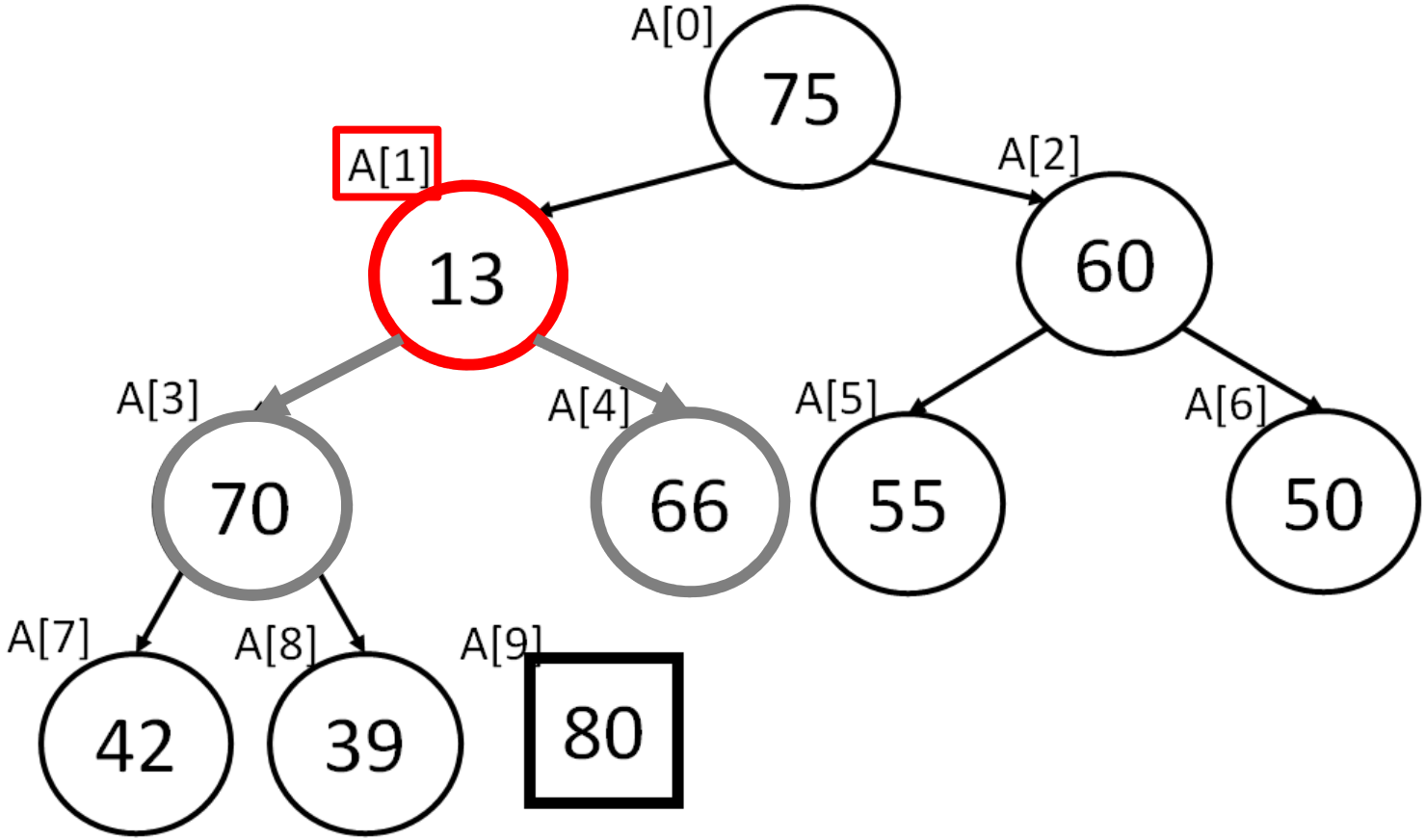
Heapifica(arranjo ***A, fim, i***)

1. ***e*** ← ***2*i + 1***
2. ***d*** ← ***2*i + 2***
3. Se ***e ≤ fim*** e ***A[e] > A[i]*** então
4. ***maior*** ← ***e***
5. Senão
6. ***maior*** ← ***i***
7. Se ***d ≤ fim*** e ***A[d] > A[maior]*** então
8. ***maior*** ← ***d***
9. Se ***maior ≠ i*** então
10. troca ***A[i]*** ↔ ***A[maior]***
11. ***Heapifica***(***A, fim, maior***)



Construindo a Heap: método Heapifica()

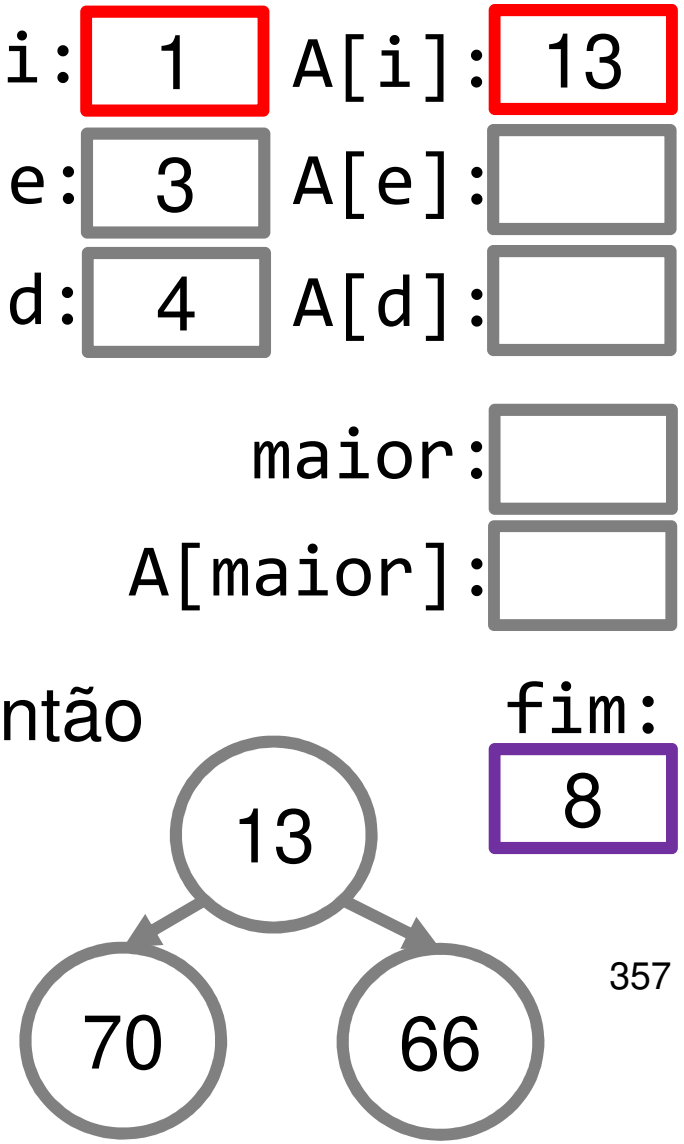
i	0	1	2	3	4	5	6	7	8	9
A[i]	75	13	60	70	66	55	50	42	39	80



Construindo a Heap: método Heapifica()

Heapifica(arranjo ***A, fim, i***)

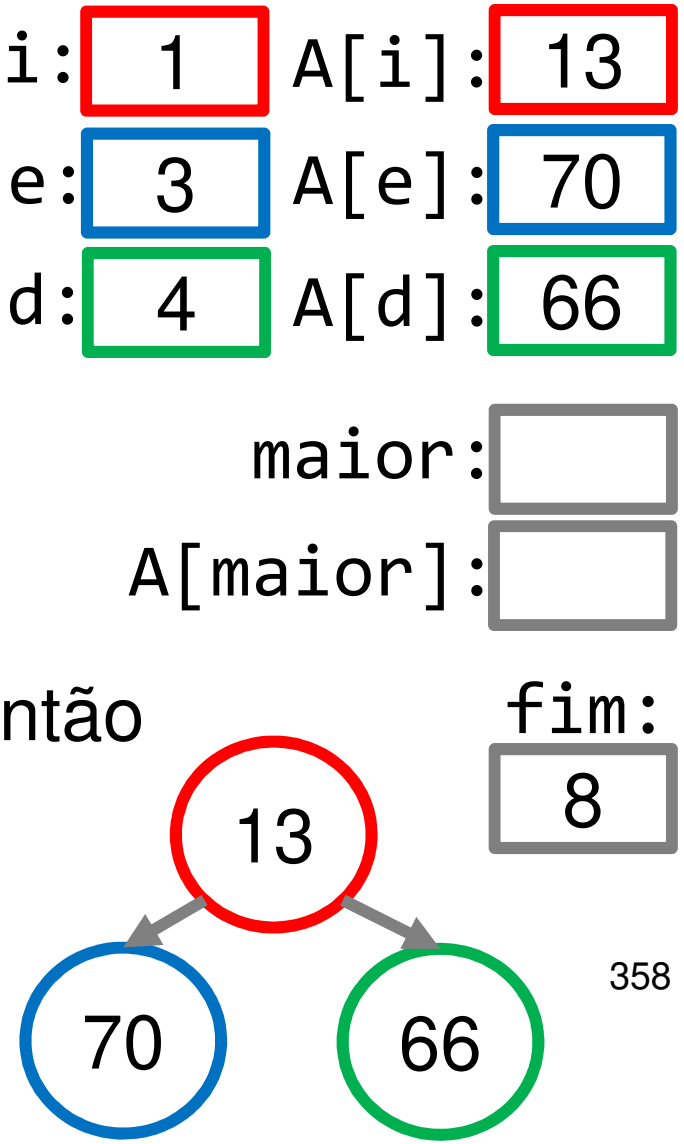
1. ***e*** ← ***2*i + 1***
2. ***d*** ← ***2*i + 2***
3. Se ***e ≤ fim*** e ***A[e] > A[i]*** então
4. ***maior*** ← ***e***
5. Senão
6. ***maior*** ← ***i***
7. Se ***d ≤ fim*** e ***A[d] > A[maior]*** então
8. ***maior*** ← ***d***
9. Se ***maior ≠ i*** então
10. troca ***A[i]*** ↔ ***A[maior]***
11. ***Heapifica***(***A, fim, maior***)



Construindo a Heap: método Heapifica()

Heapifica(arranjo ***A, fim, i***)

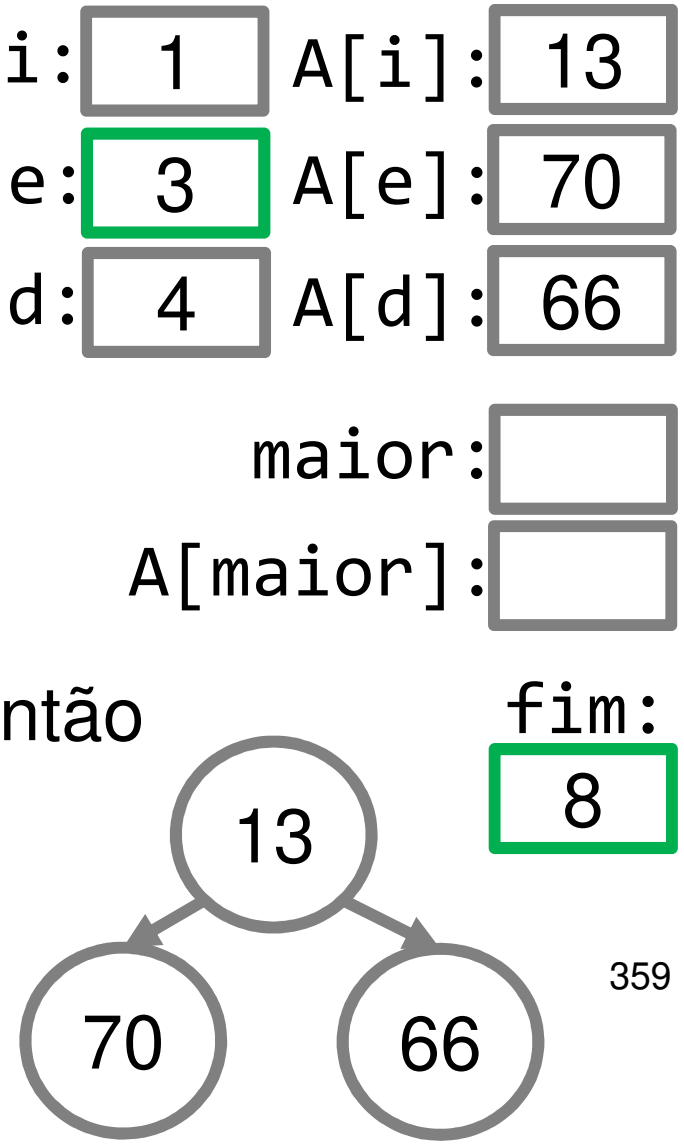
1. ***e*** ← ***2*i + 1***
2. ***d*** ← ***2*i + 2***
3. Se ***e ≤ fim*** e ***A[e] > A[i]*** então
4. ***maior*** ← ***e***
5. Senão
6. ***maior*** ← ***i***
7. Se ***d ≤ fim*** e ***A[d] > A[maior]*** então
8. ***maior*** ← ***d***
9. Se ***maior ≠ i*** então
10. troca ***A[i]*** ↔ ***A[maior]***
11. ***Heapifica***(***A, fim, maior***)



Construindo a Heap: método Heapifica()

Heapifica(arranjo ***A, fim, i***)

1. ***e*** ← ***2*i + 1***
2. ***d*** ← ***2*i + 2***
3. Se ***e ≤ fim*** e ***A[e] > A[i]*** então
4. ***maior*** ← ***e***
5. Senão
6. ***maior*** ← ***i***
7. Se ***d ≤ fim*** e ***A[d] > A[maior]*** então
8. ***maior*** ← ***d***
9. Se ***maior ≠ i*** então
10. troca ***A[i]*** ↔ ***A[maior]***
11. ***Heapifica***(***A, fim, maior***)



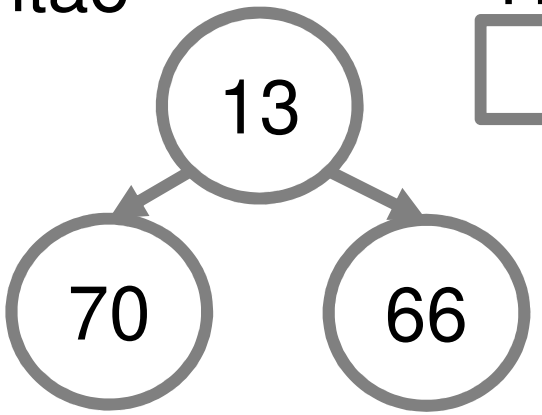
Construindo a Heap: método Heapifica()

Heapifica(arranjo ***A, fim, i***)

1. ***e*** ← ***2*i + 1***
2. ***d*** ← ***2*i + 2***
3. Se ***e ≤ fim*** e ***A[e] > A[i]*** então
4. ***maior*** ← ***e***
5. Senão
6. ***maior*** ← ***i***
7. Se ***d ≤ fim*** e ***A[d] > A[maior]*** então
8. ***maior*** ← ***d***
9. Se ***maior ≠ i*** então
10. troca ***A[i]*** ↔ ***A[maior]***
11. ***Heapifica***(***A, fim, maior***)

i:	1	A[i]:	13
e:	3	A[e]:	70
d:	4	A[d]:	66
maior:			
A[maior]:			

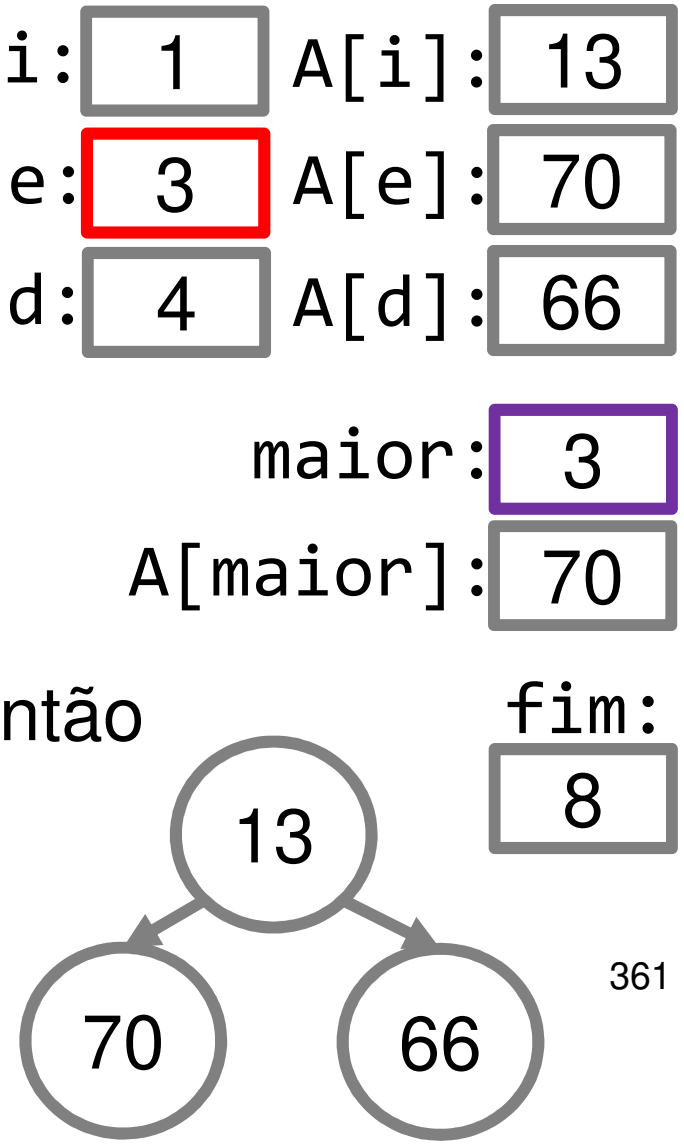
fim: 8



Construindo a Heap: método Heapifica()

Heapifica(arranjo ***A, fim, i***)

1. ***e*** ← ***2*i + 1***
2. ***d*** ← ***2*i + 2***
3. Se ***e ≤ fim*** e ***A[e] > A[i]*** então
4. ***maior*** ← ***e***
5. Senão
6. ***maior*** ← ***i***
7. Se ***d ≤ fim*** e ***A[d] > A[maior]*** então
8. ***maior*** ← ***d***
9. Se ***maior ≠ i*** então
10. troca ***A[i]*** ↔ ***A[maior]***
11. ***Heapifica***(***A, fim, maior***)

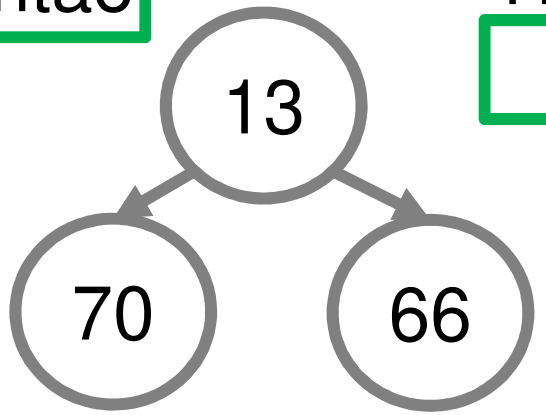


Construindo a Heap: método Heapifica()

Heapifica(arranjo ***A, fim, i***)

1. ***e*** ← ***2*i + 1***
2. ***d*** ← ***2*i + 2***
3. Se ***e ≤ fim*** e ***A[e] > A[i]*** então
4. ***maior*** ← ***e***
5. Senão
6. ***maior*** ← ***i***
7. Se ***d ≤ fim*** e ***A[d] > A[maior]*** então
8. ***maior*** ← ***d***
9. Se ***maior ≠ i*** então
10. troca ***A[i]*** ↔ ***A[maior]***
11. ***Heapifica***(***A, fim, maior***)

i:	1	A[i]:	13
e:	3	A[e]:	70
d:	4	A[d]:	66
maior:	3		
A[maior]:	70		
fim:	8		

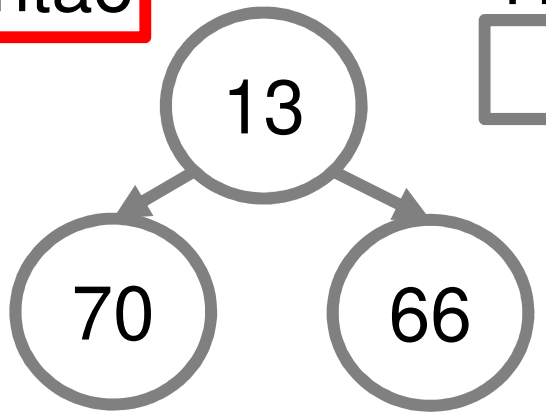


Construindo a Heap: método Heapifica()

Heapifica(arranjo ***A, fim, i***)

1. ***e*** ← ***2*i + 1***
2. ***d*** ← ***2*i + 2***
3. Se ***e*** ≤ ***fim*** e ***A[e] > A[i]*** então
4. ***maior*** ← ***e***
5. Senão
6. ***maior*** ← ***i***
7. Se ***d*** ≤ ***fim*** e ***A[d] > A[maior]*** então
8. ***maior*** ← ***d***
9. Se ***maior*** ≠ ***i*** então
10. troca ***A[i]*** ↔ ***A[maior]***
11. ***Heapifica***(***A, fim, maior***)

i:	1	A[i]:	13
e:	3	A[e]:	70
d:	4	A[d]:	66
maior:	3		
A[maior]:	70		
fim:	8		



Construindo a Heap: método Heapifica()

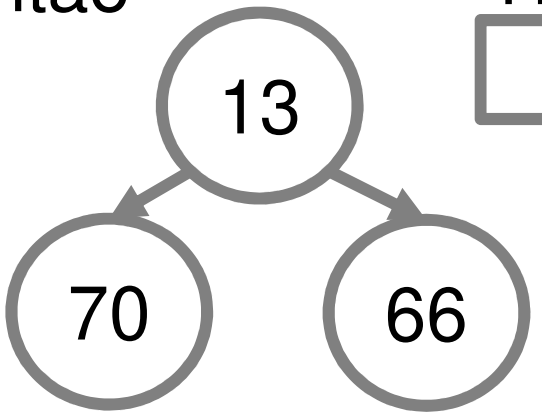
Heapifica(arranjo ***A, fim, i***)

1. ***e*** ← ***2*i + 1***
2. ***d*** ← ***2*i + 2***
3. Se ***e ≤ fim*** e ***A[e] > A[i]*** então
4. ***maior*** ← ***e***
5. Senão
6. ***maior*** ← ***i***
7. Se ***d ≤ fim*** e ***A[d] > A[maior]*** então
8. ***maior*** ← ***d***
9. Se ***maior ≠ i*** então
10. troca ***A[i]*** ↔ ***A[maior]***
11. ***Heapifica***(***A, fim, maior***)

i:	1	A[i]:	13
e:	3	A[e]:	70
d:	4	A[d]:	66

maior:	3
A[maior]:	70

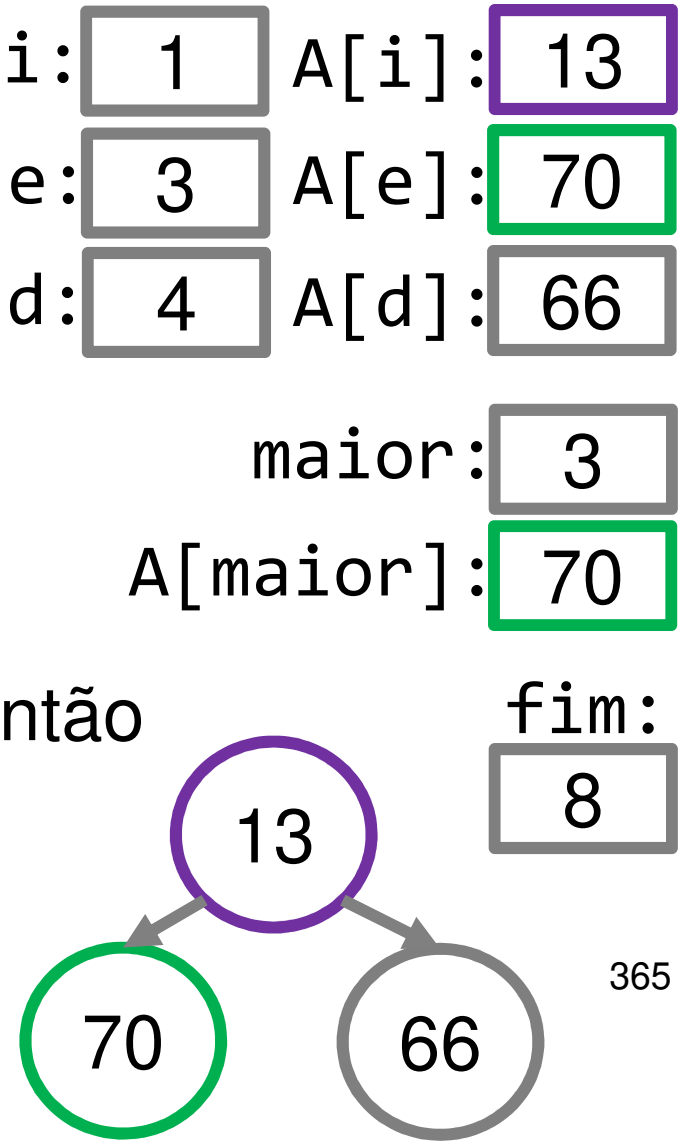
fim: 8



Construindo a Heap: método Heapifica()

Heapifica(arranjo ***A, fim, i***)

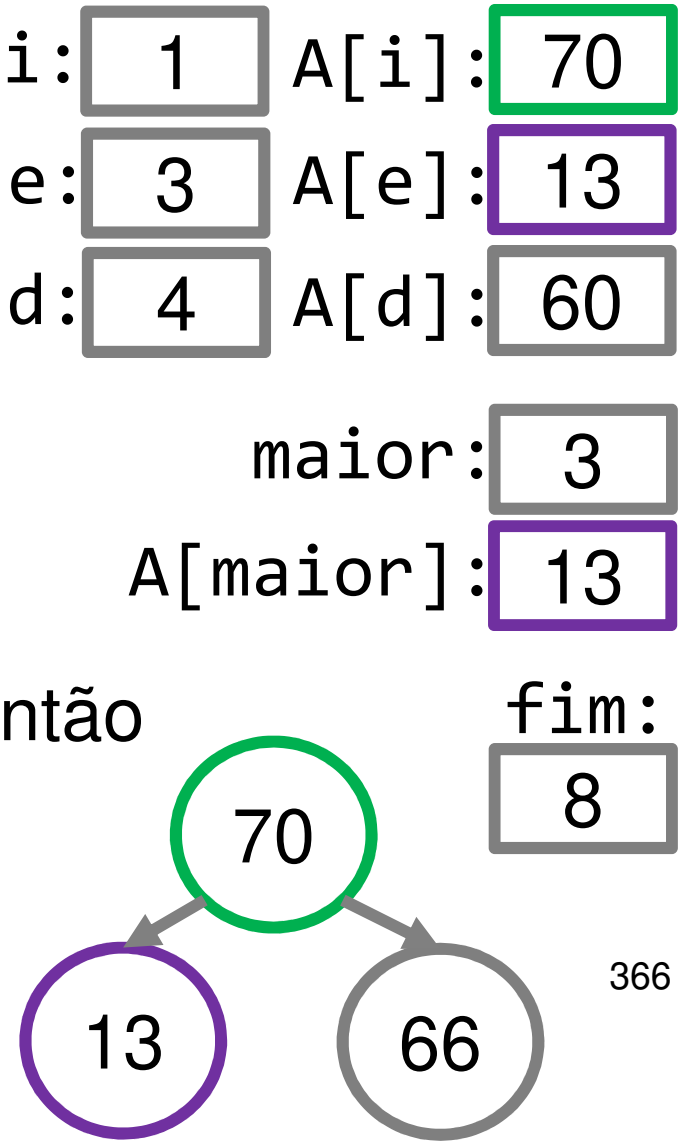
1. ***e*** ← ***2*i + 1***
2. ***d*** ← ***2*i + 2***
3. Se ***e ≤ fim*** e ***A[e] > A[i]*** então
4. ***maior*** ← ***e***
5. Senão
6. ***maior*** ← ***i***
7. Se ***d ≤ fim*** e ***A[d] > A[maior]*** então
8. ***maior*** ← ***d***
9. Se ***maior ≠ i*** então
10. troca ***A[i]*** ↔ ***A[maior]***
11. ***Heapifica***(***A, fim, maior***)



Construindo a Heap: método Heapifica()

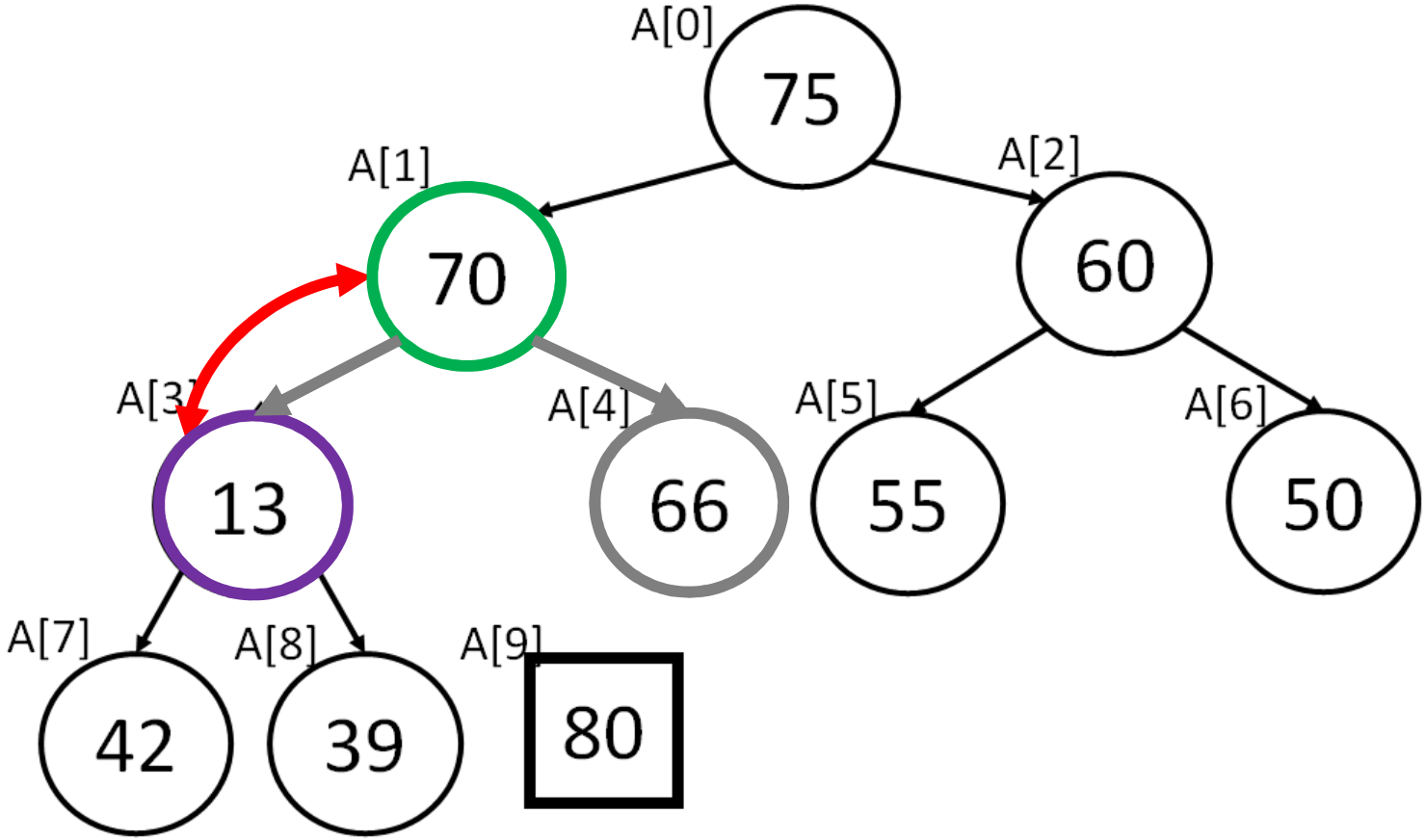
Heapifica(arranjo ***A, fim, i***)

1. ***e*** ← ***2*i + 1***
2. ***d*** ← ***2*i + 2***
3. Se ***e ≤ fim*** e ***A[e] > A[i]*** então
4. ***maior*** ← ***e***
5. Senão
6. ***maior*** ← ***i***
7. Se ***d ≤ fim*** e ***A[d] > A[maior]*** então
8. ***maior*** ← ***d***
9. Se ***maior ≠ i*** então
10. troca ***A[i]*** ↔ ***A[maior]***
11. ***Heapifica***(***A, fim, maior***)



Construindo a Heap: método Heapifica()

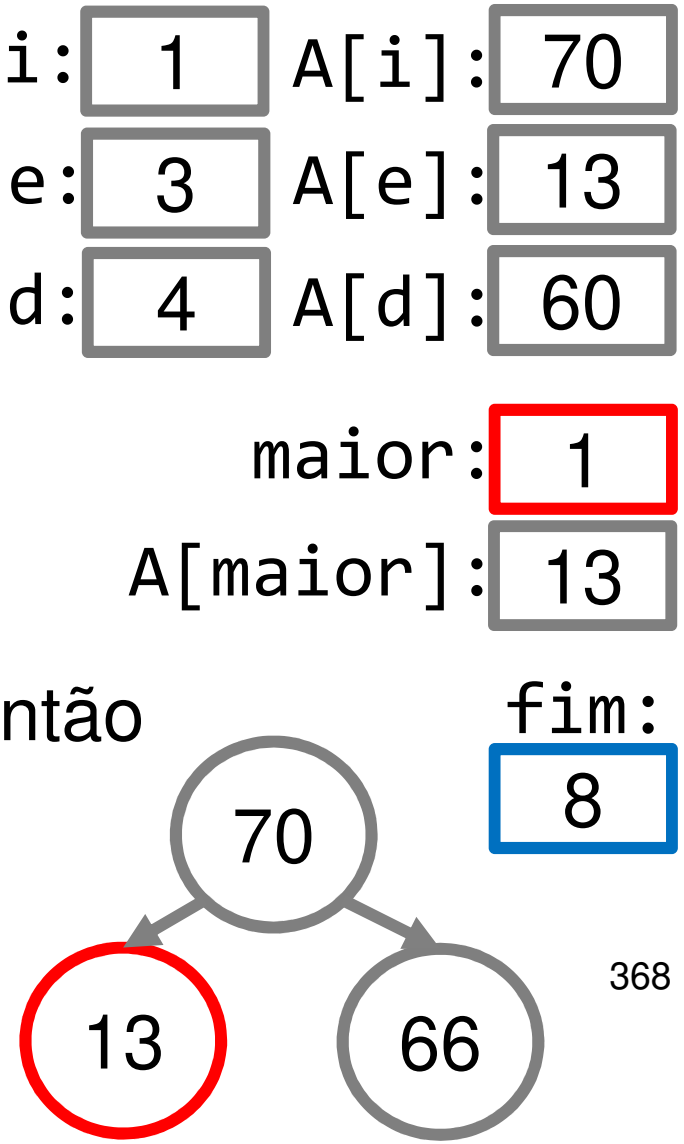
i	0	1	2	3	4	5	6	7	8	9
A[i]	75	70	60	13	66	55	50	42	39	80



Construindo a Heap: método Heapifica()

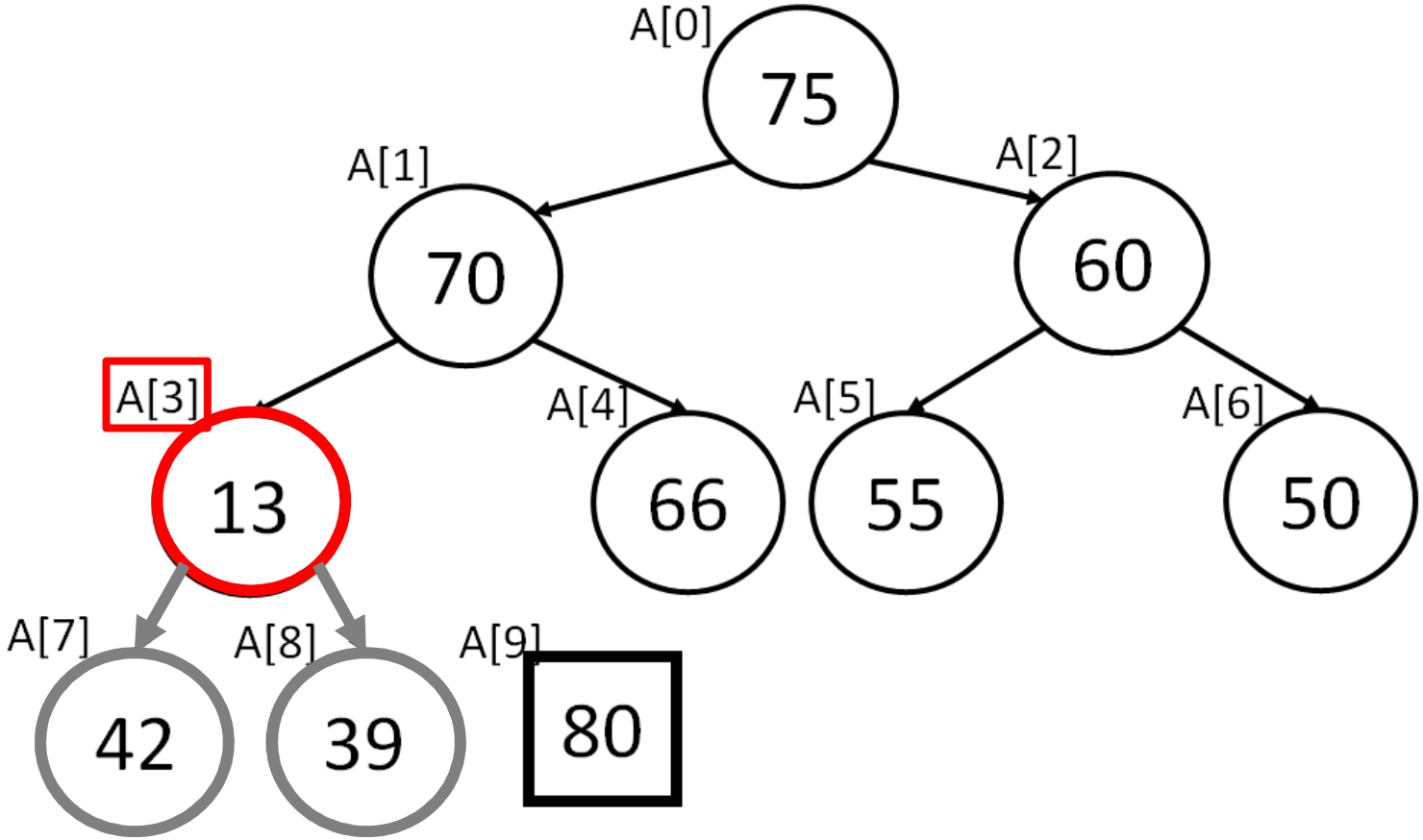
Heapifica(arranjo ***A, fim, i***)

1. ***e*** ← ***2*i + 1***
2. ***d*** ← ***2*i + 2***
3. Se ***e ≤ fim*** e ***A[e] > A[i]*** então
4. ***maior*** ← ***e***
5. Senão
6. ***maior*** ← ***i***
7. Se ***d ≤ fim*** e ***A[d] > A[maior]*** então
8. ***maior*** ← ***d***
9. Se ***maior ≠ i*** então
10. troca ***A[i]*** ↔ ***A[maior]***
11. ***Heapifica***(***A, fim, maior***)



Construindo a Heap: método Heapifica()

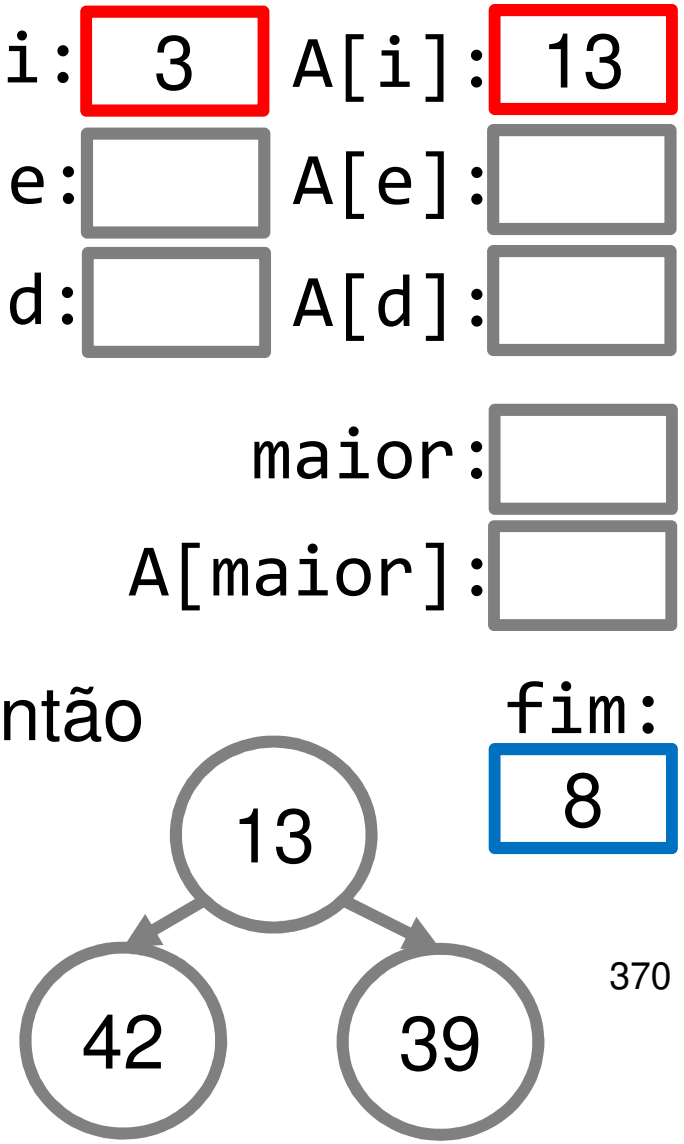
i	0	1	2	3	4	5	6	7	8	9
A[i]	75	70	60	13	66	55	50	42	39	80



Construindo a Heap: método Heapifica()

Heapifica(arranjo ***A***, ***fim***, ***i***)

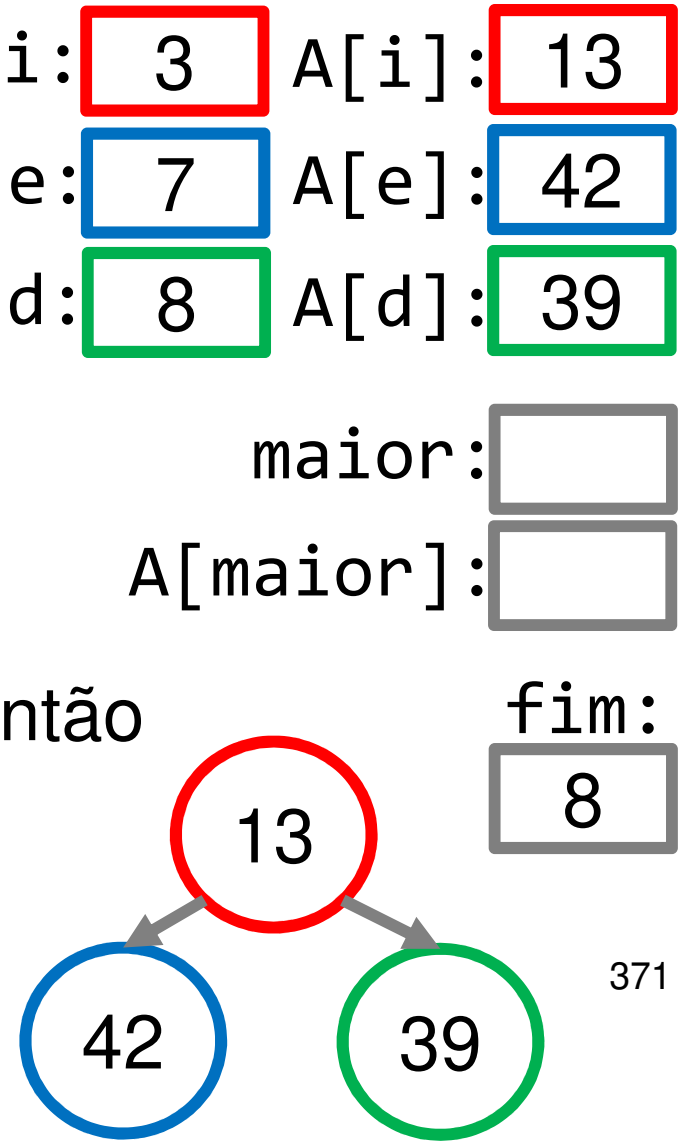
1. ***e*** ← ***2*i + 1***
2. ***d*** ← ***2*i + 2***
3. Se ***e*** ≤ ***fim*** e ***A[e]*** > ***A[i]*** então
4. ***maior*** ← ***e***
5. Senão
6. ***maior*** ← ***i***
7. Se ***d*** ≤ ***fim*** e ***A[d]*** > ***A[maior]*** então
8. ***maior*** ← ***d***
9. Se ***maior*** ≠ ***i*** então
10. troca ***A[i]*** ↔ ***A[maior]***
11. ***Heapifica***(***A, fim, maior***)



Construindo a Heap: método Heapifica()

Heapifica(arranjo ***A, fim, i***)

1. ***e*** ← ***2*i + 1***
2. ***d*** ← ***2*i + 2***
3. Se ***e ≤ fim*** e ***A[e] > A[i]*** então
4. ***maior*** ← ***e***
5. Senão
6. ***maior*** ← ***i***
7. Se ***d ≤ fim*** e ***A[d] > A[maior]*** então
8. ***maior*** ← ***d***
9. Se ***maior ≠ i*** então
10. troca ***A[i]*** ↔ ***A[maior]***
11. ***Heapifica***(***A, fim, maior***)



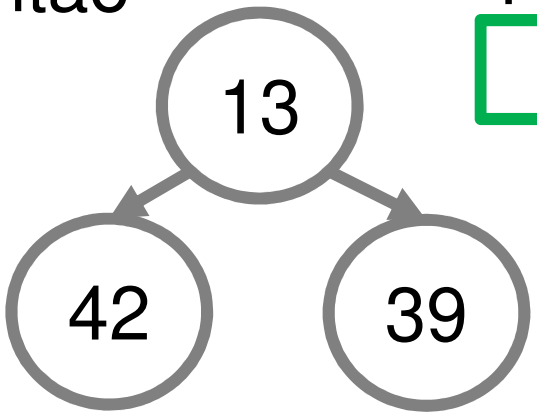
Construindo a Heap: método Heapifica()

Heapifica(arranjo ***A, fim, i***)

1. ***e*** ← ***2*i + 1***
2. ***d*** ← ***2*i + 2***
3. Se ***e ≤ fim*** e ***A[e] > A[i]*** então
4. ***maior*** ← ***e***
5. Senão
6. ***maior*** ← ***i***
7. Se ***d ≤ fim*** e ***A[d] > A[maior]*** então
8. ***maior*** ← ***d***
9. Se ***maior ≠ i*** então
10. troca ***A[i]*** ↔ ***A[maior]***
11. ***Heapifica***(***A, fim, maior***)

i:	3	A[i]:	13
e:	7	A[e]:	42
d:	8	A[d]:	39
		maior:	
		A[maior]:	

fim: 8



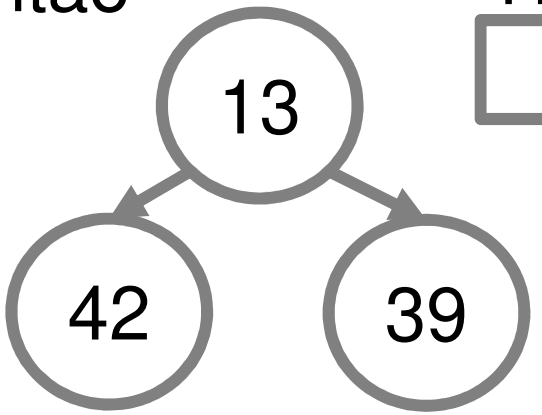
Construindo a Heap: método Heapifica()

Heapifica(arranjo ***A, fim, i***)

1. ***e*** ← ***2*i + 1***
2. ***d*** ← ***2*i + 2***
3. Se ***e ≤ fim*** e ***A[e] > A[i]*** então
4. ***maior*** ← ***e***
5. Senão
6. ***maior*** ← ***i***
7. Se ***d ≤ fim*** e ***A[d] > A[maior]*** então
8. ***maior*** ← ***d***
9. Se ***maior ≠ i*** então
10. troca ***A[i]*** ↔ ***A[maior]***
11. ***Heapifica***(***A, fim, maior***)

i:	3	A[i]:	13
e:	7	A[e]:	42
d:	8	A[d]:	39
maior:			
A[maior]:			

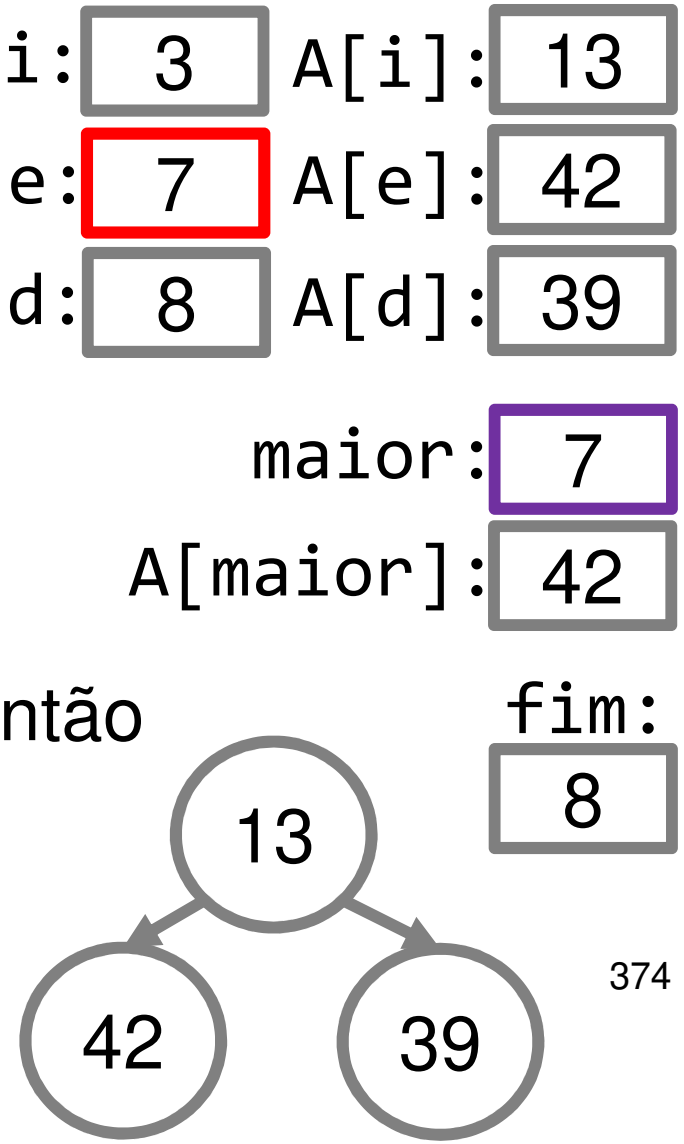
fim: 8



Construindo a Heap: método Heapifica()

Heapifica(arranjo ***A, fim, i***)

1. ***e*** ← ***2*i + 1***
2. ***d*** ← ***2*i + 2***
3. Se ***e ≤ fim*** e ***A[e] > A[i]*** então
4. ***maior*** ← ***e***
5. Senão
6. ***maior*** ← ***i***
7. Se ***d ≤ fim*** e ***A[d] > A[maior]*** então
8. ***maior*** ← ***d***
9. Se ***maior ≠ i*** então
10. troca ***A[i]*** ↔ ***A[maior]***
11. ***Heapifica***(***A, fim, maior***)

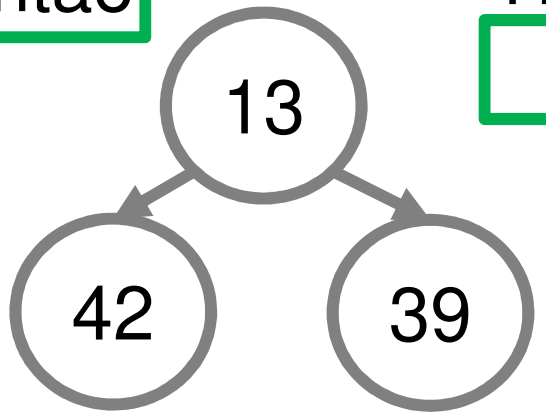


Construindo a Heap: método Heapifica()

Heapifica(arranjo ***A, fim, i***)

1. ***e*** ← ***2*i + 1***
2. ***d*** ← ***2*i + 2***
3. Se ***e ≤ fim*** e ***A[e] > A[i]*** então
4. ***maior*** ← ***e***
5. Senão
6. ***maior*** ← ***i***
7. Se ***d ≤ fim*** e ***A[d] > A[maior]*** então
8. ***maior*** ← ***d***
9. Se ***maior ≠ i*** então
10. troca ***A[i]*** ↔ ***A[maior]***
11. ***Heapifica***(***A, fim, maior***)

i:	3	A[i]:	13
e:	7	A[e]:	42
d:	8	A[d]:	39
maior:	7		
A[maior]:	42		
fim:	8		

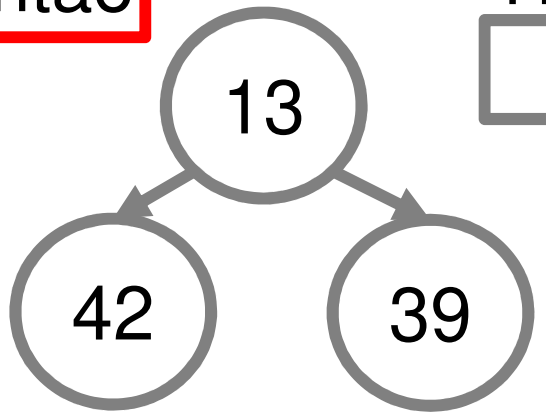


Construindo a Heap: método Heapifica()

Heapifica(arranjo ***A, fim, i***)

1. ***e*** ← ***2*i + 1***
2. ***d*** ← ***2*i + 2***
3. Se ***e*** ≤ ***fim*** e ***A[e] > A[i]*** então
4. ***maior*** ← ***e***
5. Senão
6. ***maior*** ← ***i***
7. Se ***d*** ≤ ***fim*** e ***A[d] > A[maior]*** então
8. ***maior*** ← ***d***
9. Se ***maior*** ≠ ***i*** então
10. troca ***A[i]*** ↔ ***A[maior]***
11. ***Heapifica***(***A, fim, maior***)

i:	3	A[i]:	13
e:	7	A[e]:	42
d:	8	A[d]:	39
maior:	7		
A[maior]:	42		
fim:	8		



Construindo a Heap: método Heapifica()

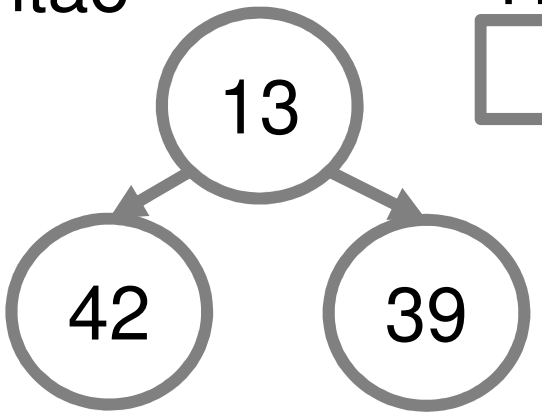
Heapifica(arranjo ***A, fim, i***)

1. ***e*** ← ***2*i + 1***
2. ***d*** ← ***2*i + 2***
3. Se ***e*** ≤ ***fim*** e ***A[e] > A[i]*** então
4. ***maior*** ← ***e***
5. Senão
6. ***maior*** ← ***i***
7. Se ***d*** ≤ ***fim*** e ***A[d] > A[maior]*** então
8. ***maior*** ← ***d***
9. Se ***maior*** ≠ ***i*** então
10. troca ***A[i]*** ↔ ***A[maior]***
11. ***Heapifica***(***A, fim, maior***)

i:	3	A[i]:	13
e:	7	A[e]:	42
d:	8	A[d]:	39

maior:	7
A[maior]:	42

fim: 8

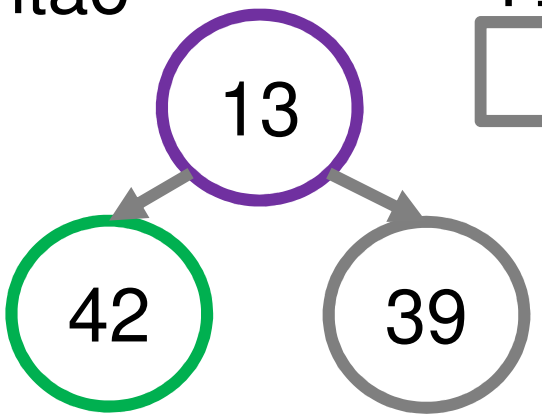


Construindo a Heap: método Heapifica()

Heapifica(arranjo ***A, fim, i***)

1. ***e*** ← ***2*i + 1***
2. ***d*** ← ***2*i + 2***
3. Se ***e ≤ fim*** e ***A[e] > A[i]*** então
4. ***maior*** ← ***e***
5. Senão
6. ***maior*** ← ***i***
7. Se ***d ≤ fim*** e ***A[d] > A[maior]*** então
8. ***maior*** ← ***d***
9. Se ***maior ≠ i*** então
10. troca ***A[i]*** ↔ ***A[maior]***
11. ***Heapifica***(***A, fim, maior***)

i:	3	A[i]:	13
e:	7	A[e]:	42
d:	8	A[d]:	39
maior:	7		
A[maior]:	42		
fim:	8		

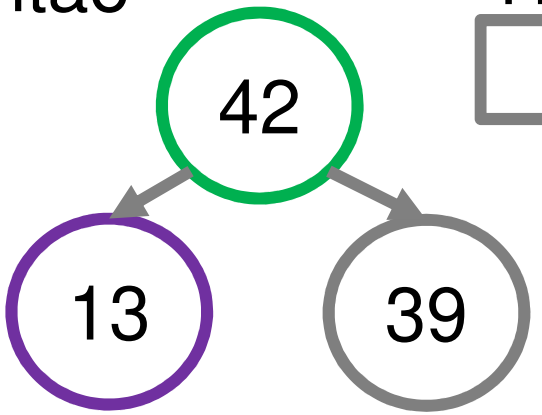


Construindo a Heap: método Heapifica()

Heapifica(arranjo ***A, fim, i***)

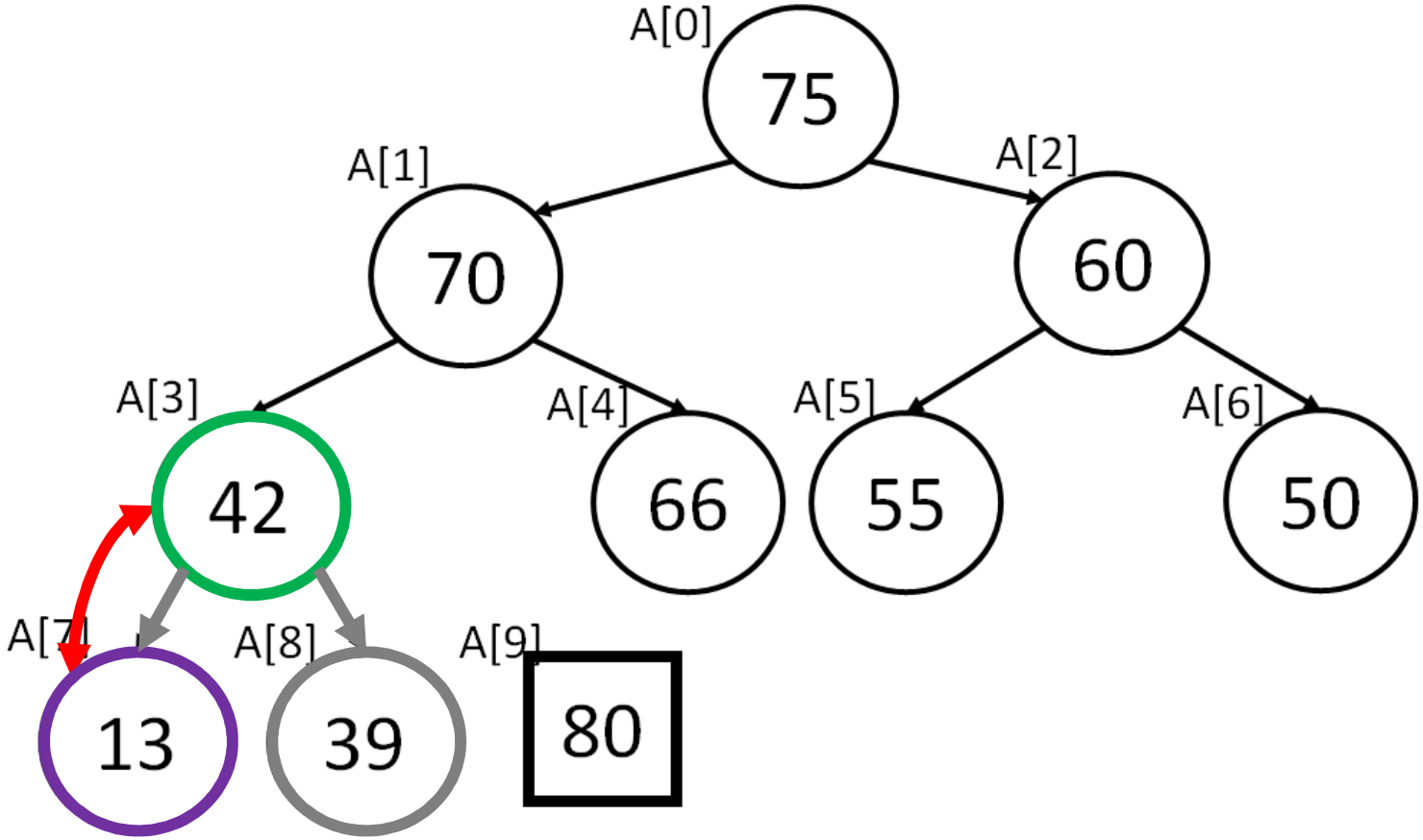
1. ***e*** ← ***2*i + 1***
2. ***d*** ← ***2*i + 2***
3. Se ***e ≤ fim*** e ***A[e] > A[i]*** então
4. ***maior*** ← ***e***
5. Senão
6. ***maior*** ← ***i***
7. Se ***d ≤ fim*** e ***A[d] > A[maior]*** então
8. ***maior*** ← ***d***
9. Se ***maior ≠ i*** então
10. troca ***A[i]*** ↔ ***A[maior]***
11. ***Heapifica***(***A, fim, maior***)

i:	3	A[i]:	42
e:	7	A[e]:	13
d:	8	A[d]:	39
maior:	7		
A[maior]:	13		
fim:	8		



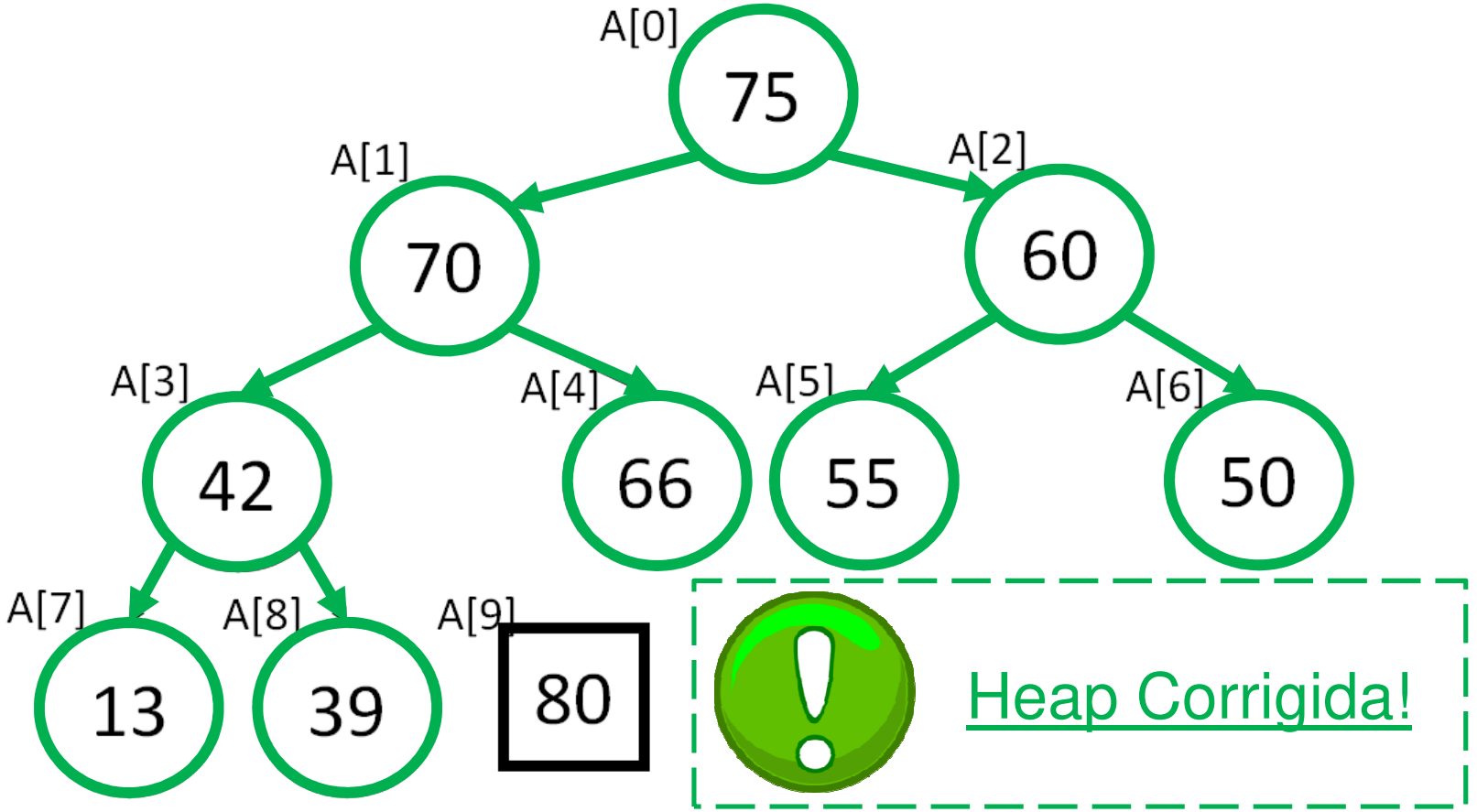
Construindo a Heap: método Heapifica()

i	0	1	2	3	4	5	6	7	8	9
A[i]	75	70	60	42	66	55	50	13	39	80



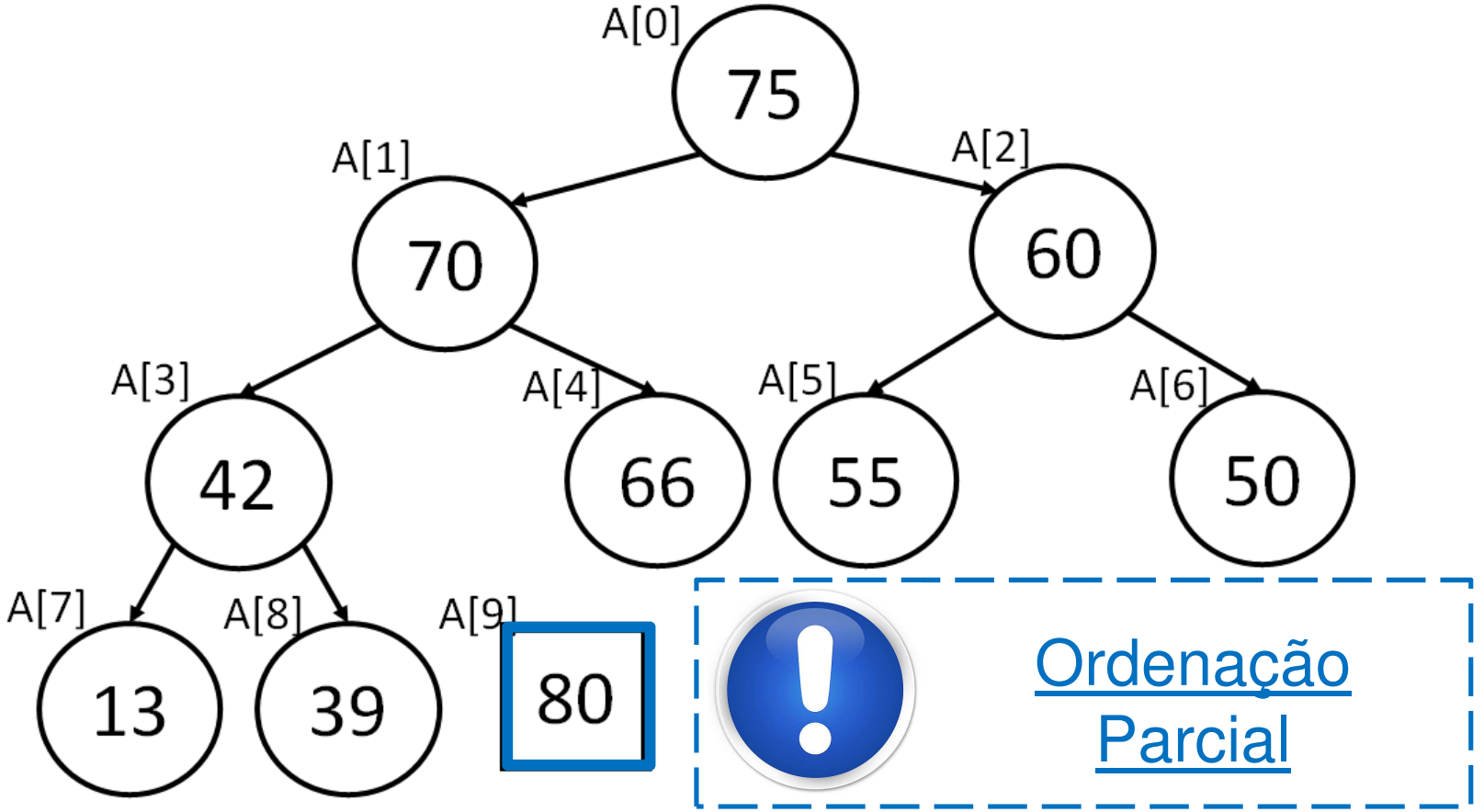
Construindo a Heap: método Heapifica()

i	0	1	2	3	4	5	6	7	8	9
A[i]	75	70	60	42	66	55	50	13	39	80



Construindo a Heap: método Heapifica()

i	0	1	2	3	4	5	6	7	8	9
A[i]	75	70	60	42	66	55	50	13	39	80

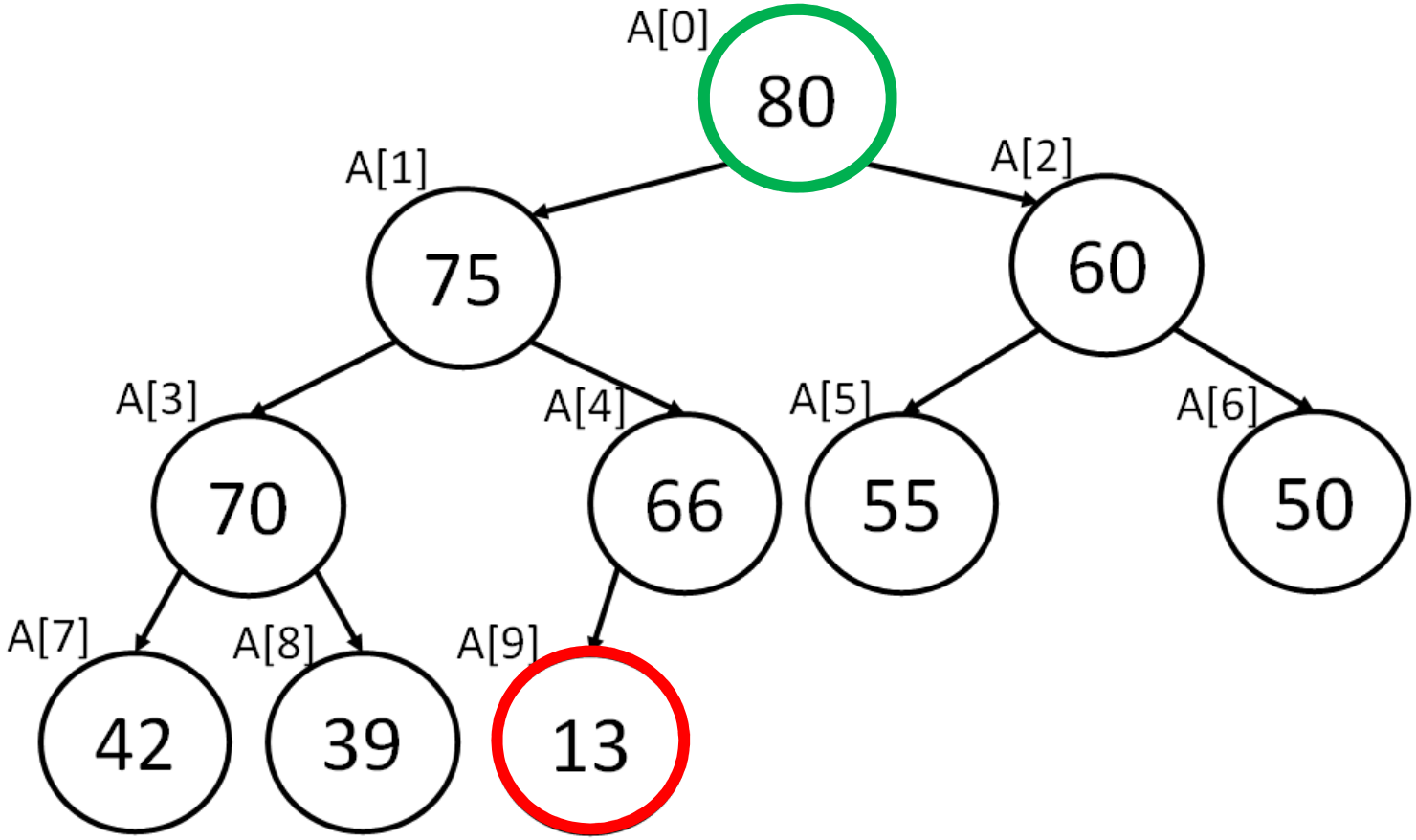


Ordenando Pela Raiz da Heap...

Em suma, o que ocorreu desde a troca da raiz, até a correção da heap?

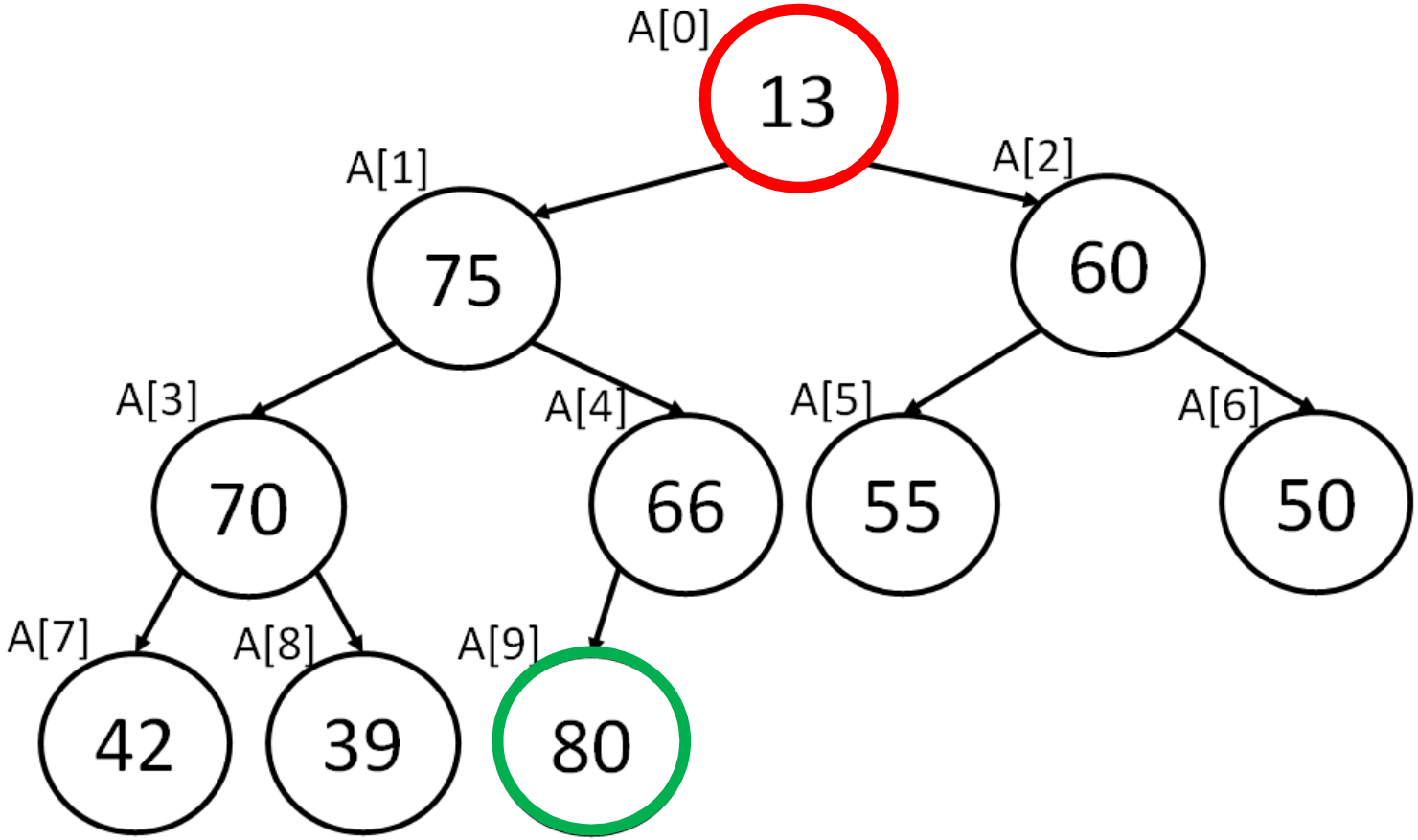
Ordenando Pela Raiz: Método Heapsort()

i	0	1	2	3	4	5	6	7	8	9
A[i]	80	75	60	70	66	55	50	42	39	13



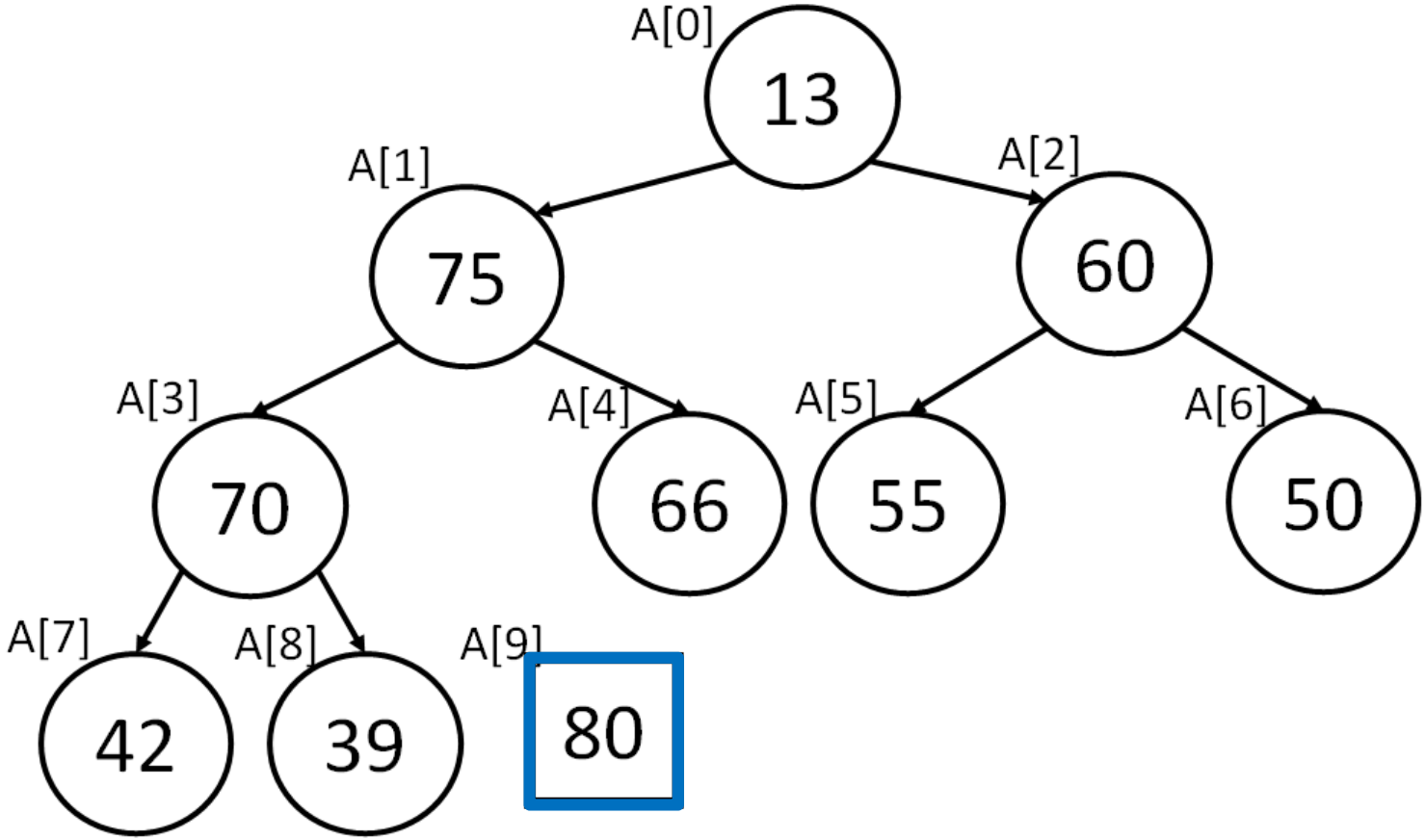
Ordenando Pela Raiz: Método Heapsort()

i	0	1	2	3	4	5	6	7	8	9
A[i]	13	75	60	70	66	55	50	42	39	80



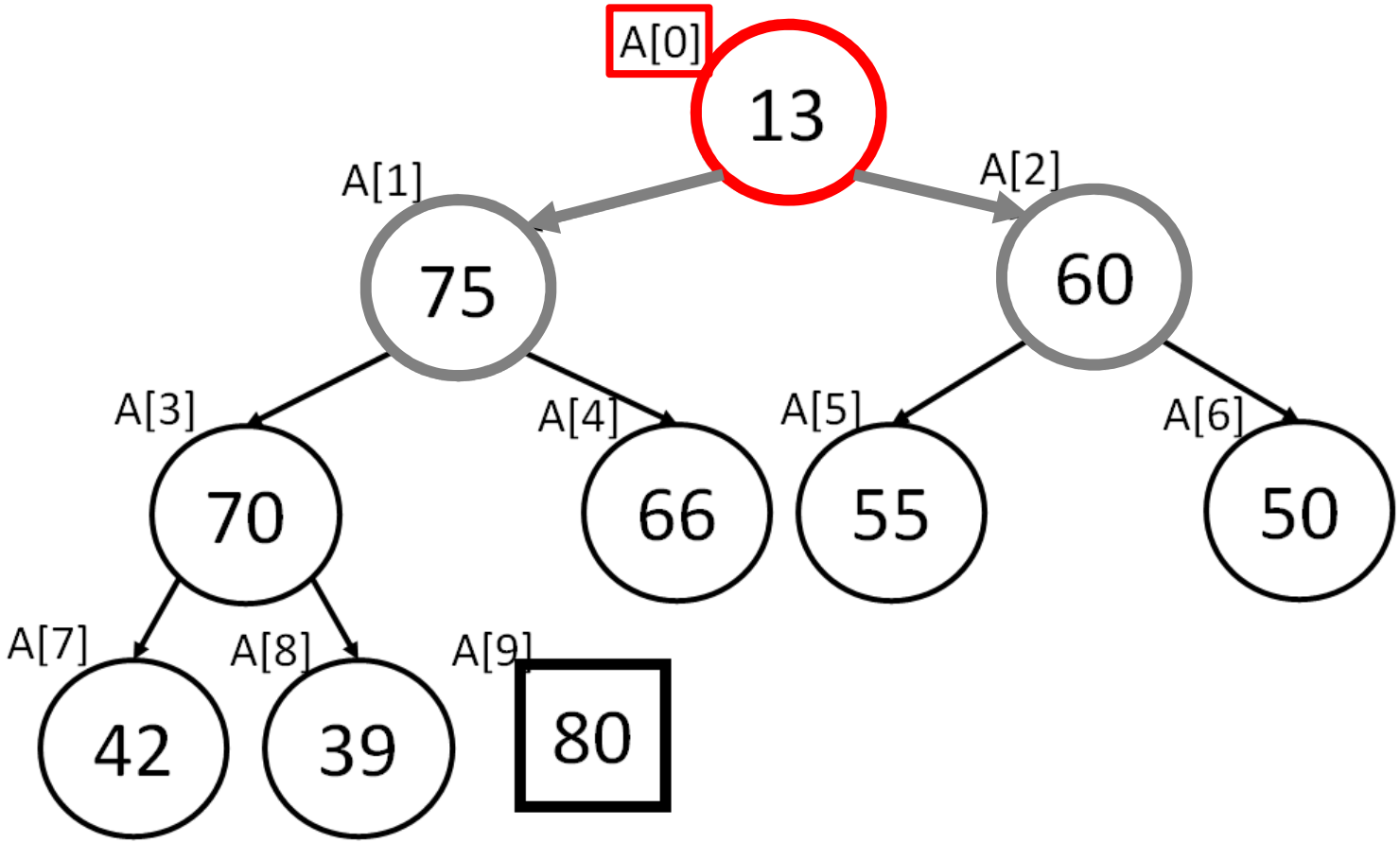
Ordenando Pela Raiz: Método Heapsort()

i	0	1	2	3	4	5	6	7	8	9
A[i]	13	75	60	70	66	55	50	42	39	80



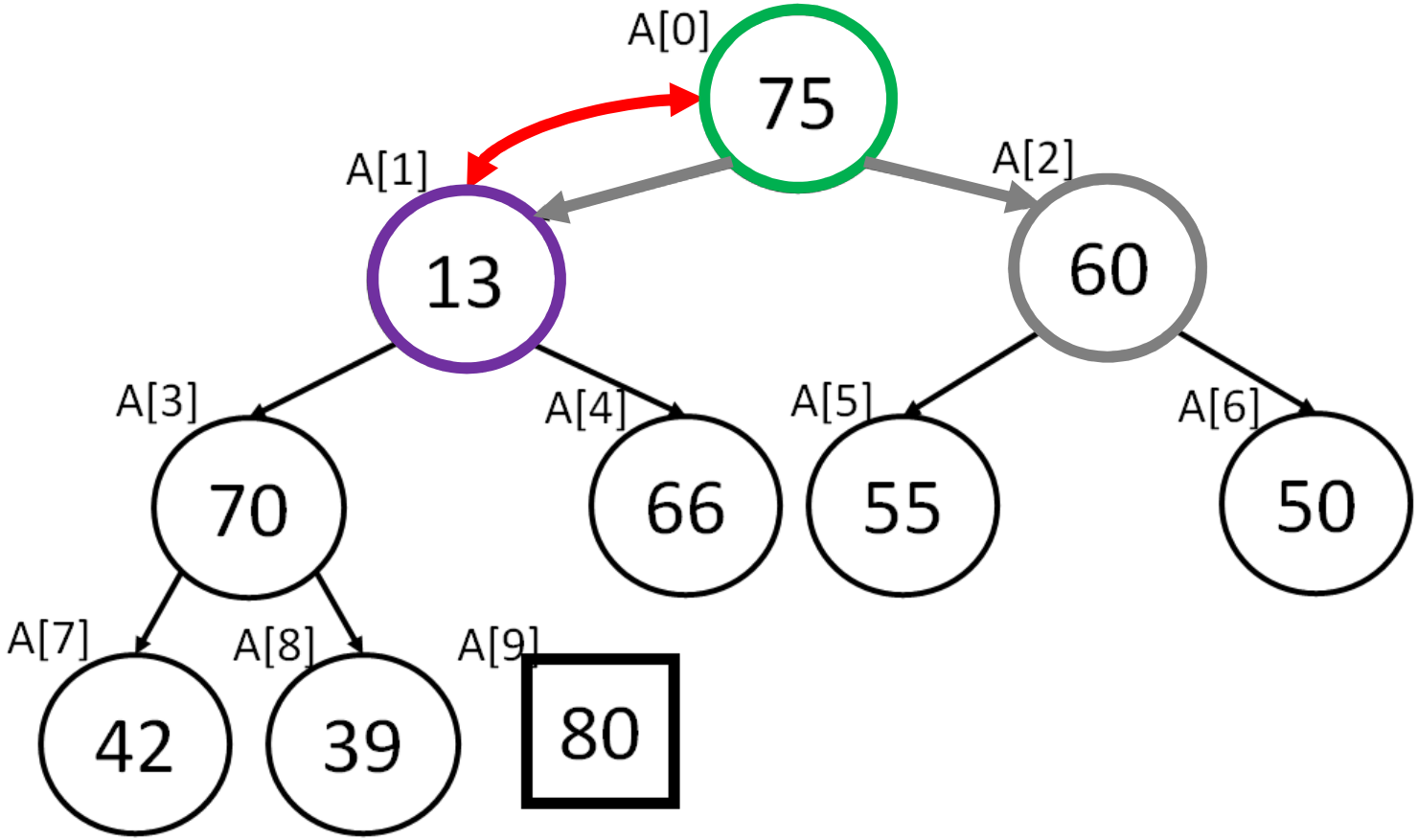
Ordenando Pela Raiz: Método Heapifica()

i	0	1	2	3	4	5	6	7	8	9
A[i]	13	75	60	70	66	55	50	42	39	80



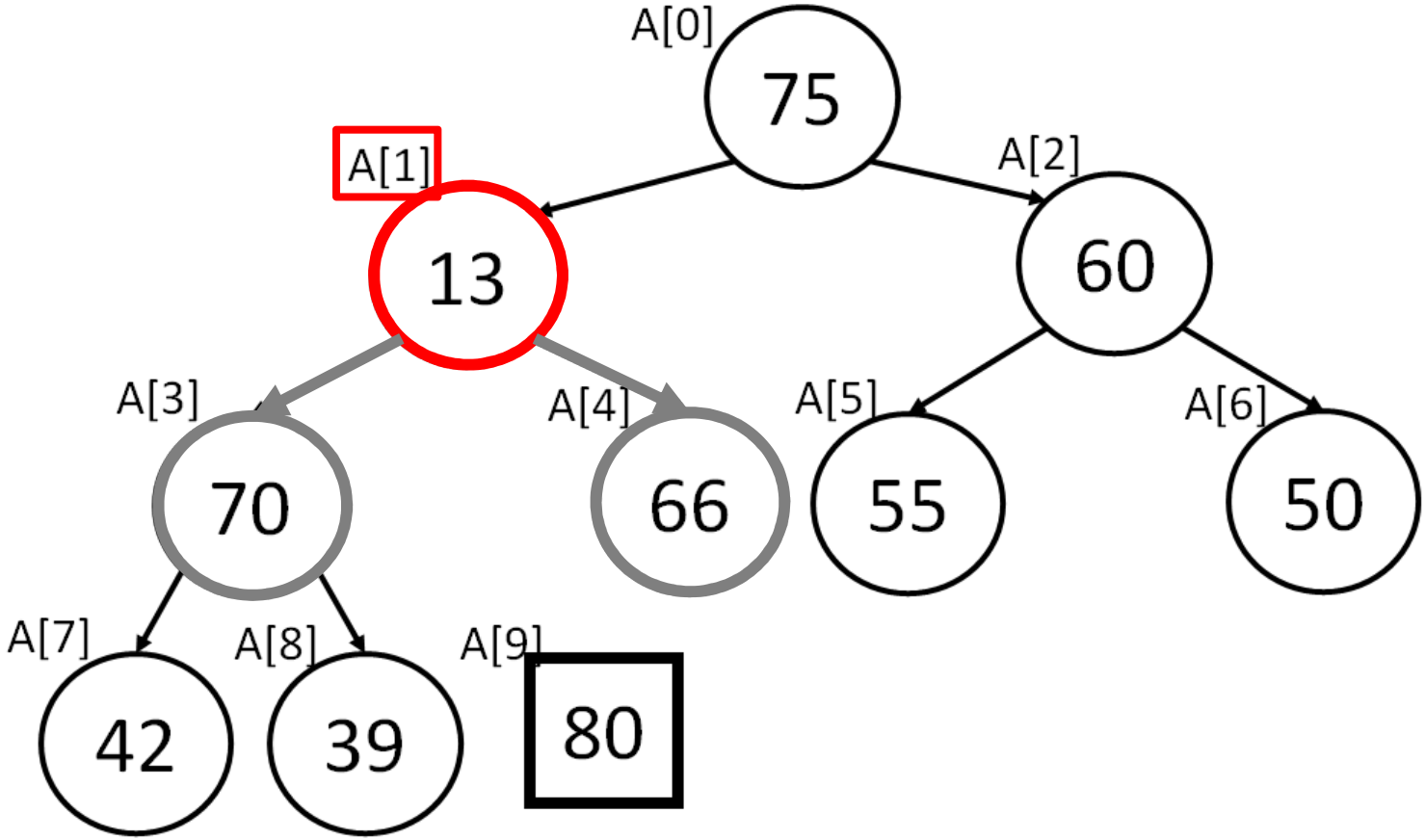
Construindo a Heap: método Heapifica()

i	0	1	2	3	4	5	6	7	8	9
A[i]	75	13	60	70	66	55	50	42	39	80



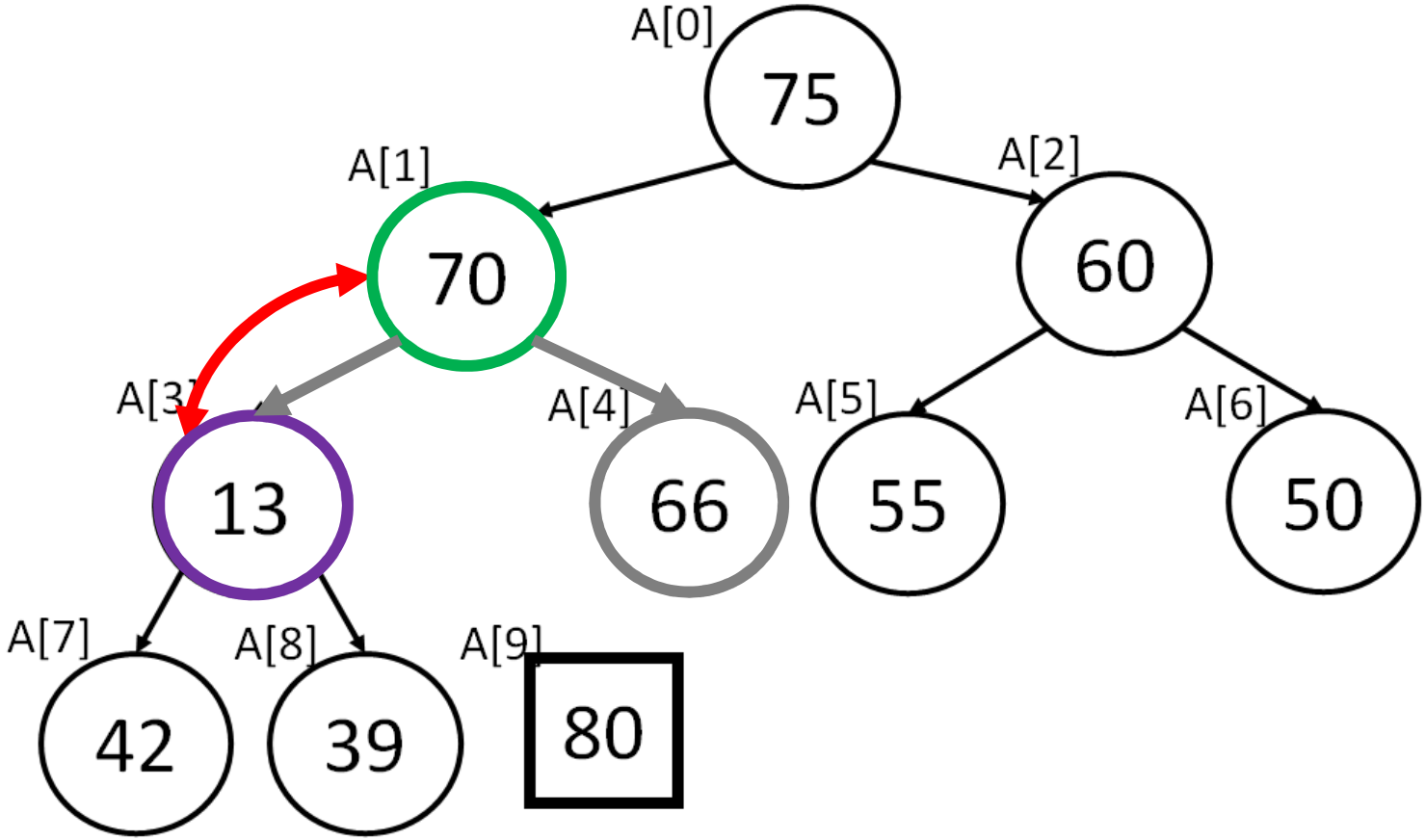
Construindo a Heap: método Heapifica()

i	0	1	2	3	4	5	6	7	8	9
A[i]	75	13	60	70	66	55	50	42	39	80



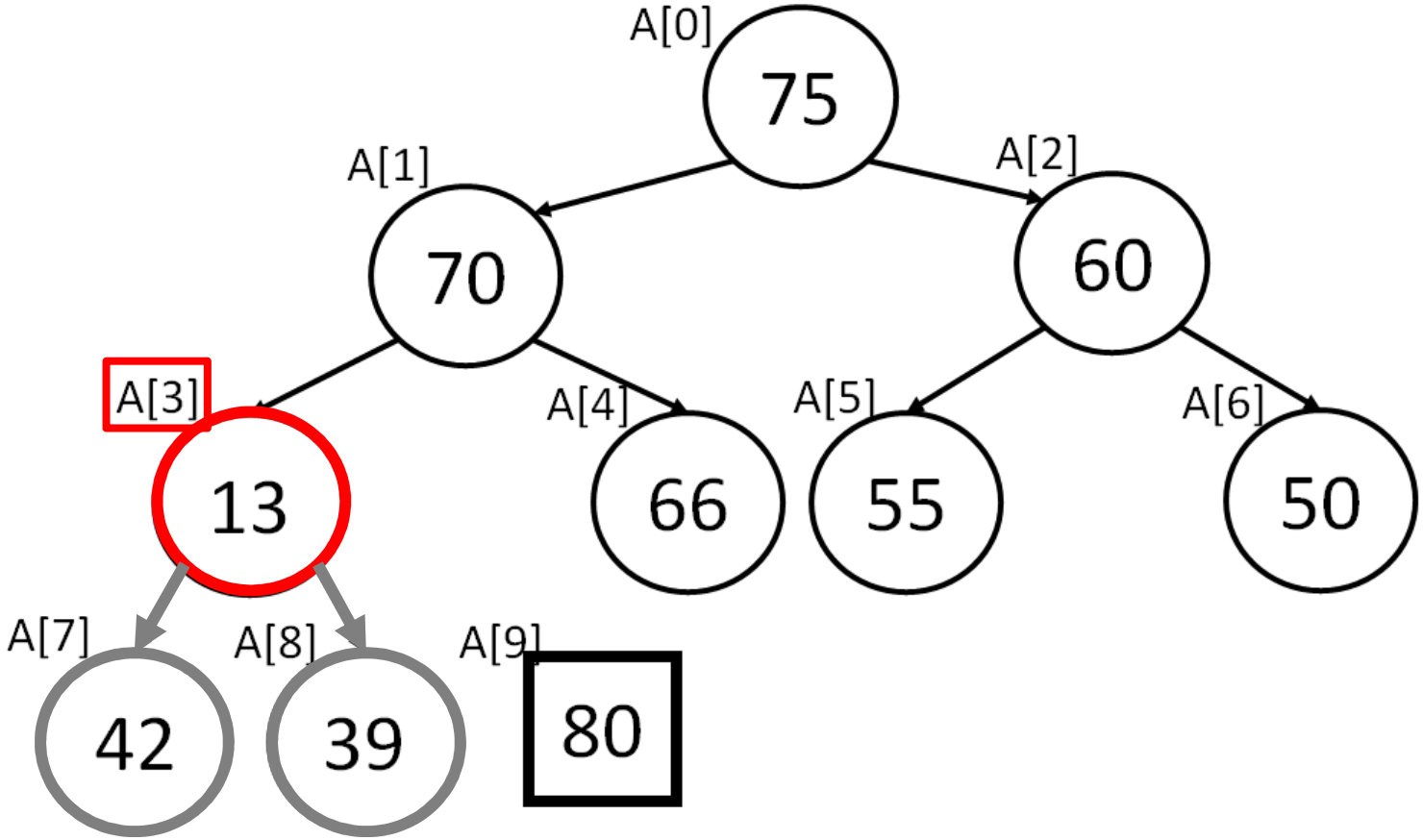
Construindo a Heap: método Heapifica()

i	0	1	2	3	4	5	6	7	8	9
A[i]	75	70	60	13	66	55	50	42	39	80



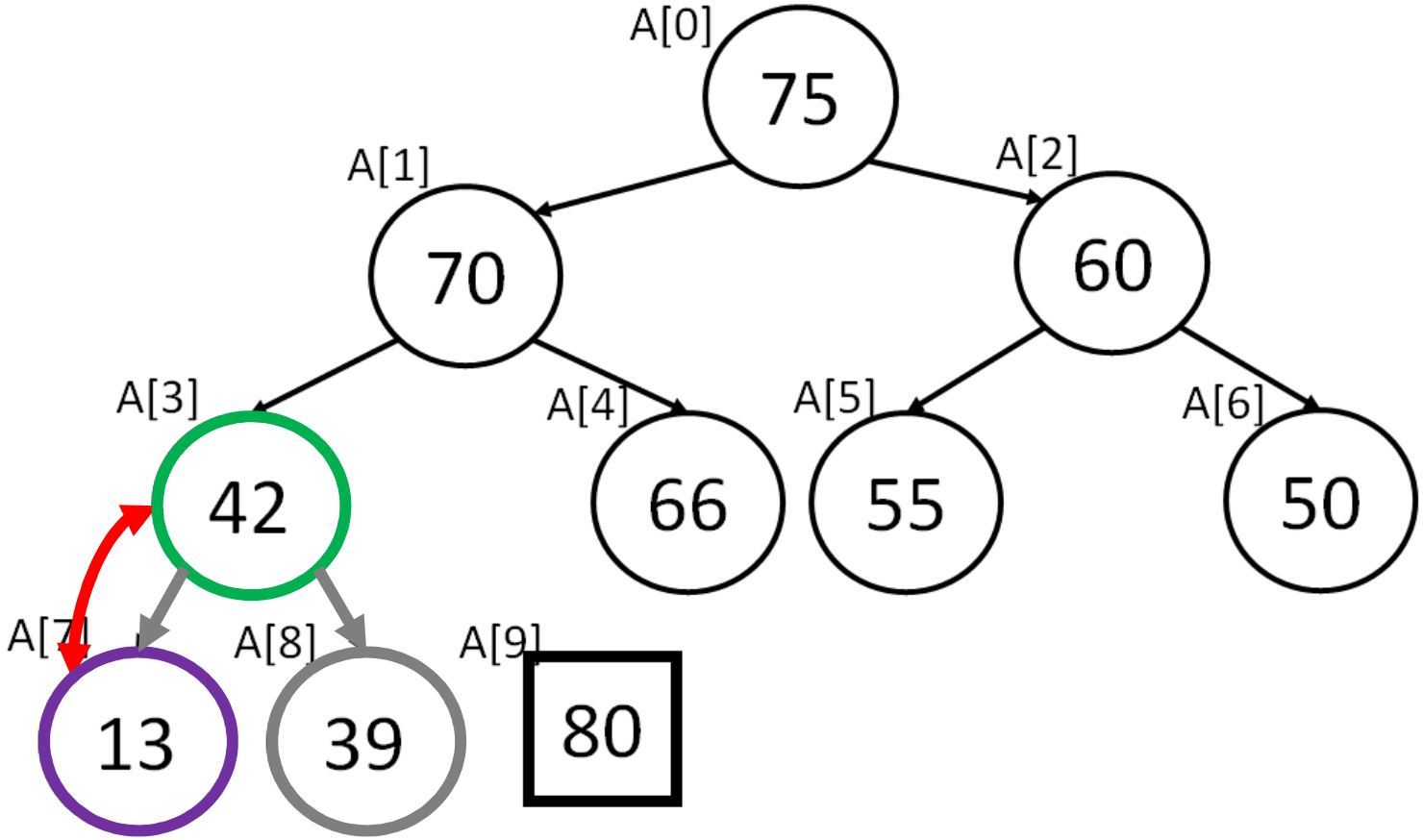
Construindo a Heap: método Heapifica()

i	0	1	2	3	4	5	6	7	8	9
A[i]	75	70	60	13	66	55	50	42	39	80



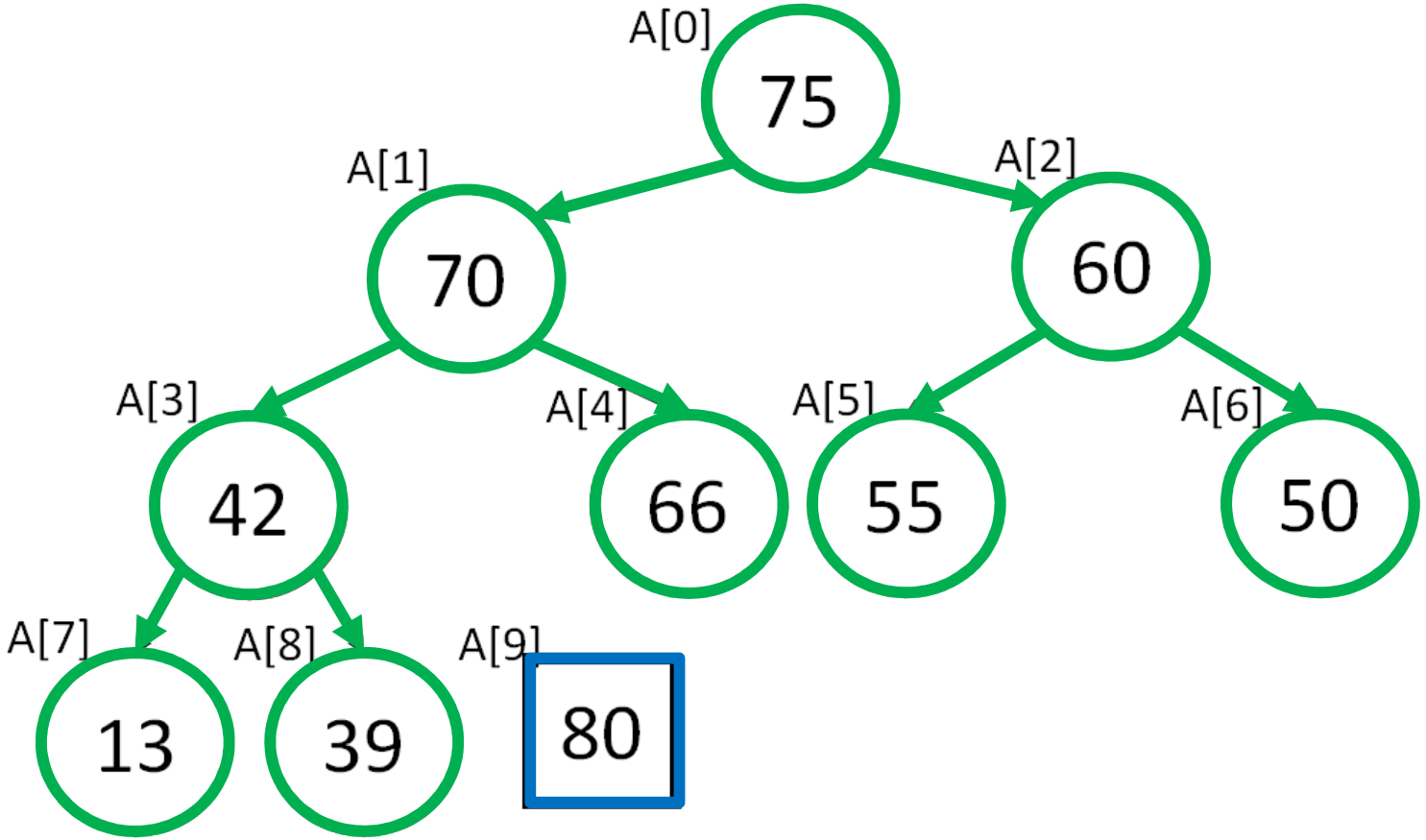
Construindo a Heap: método Heapifica()

i	0	1	2	3	4	5	6	7	8	9
A[i]	75	70	60	42	66	55	50	13	39	80



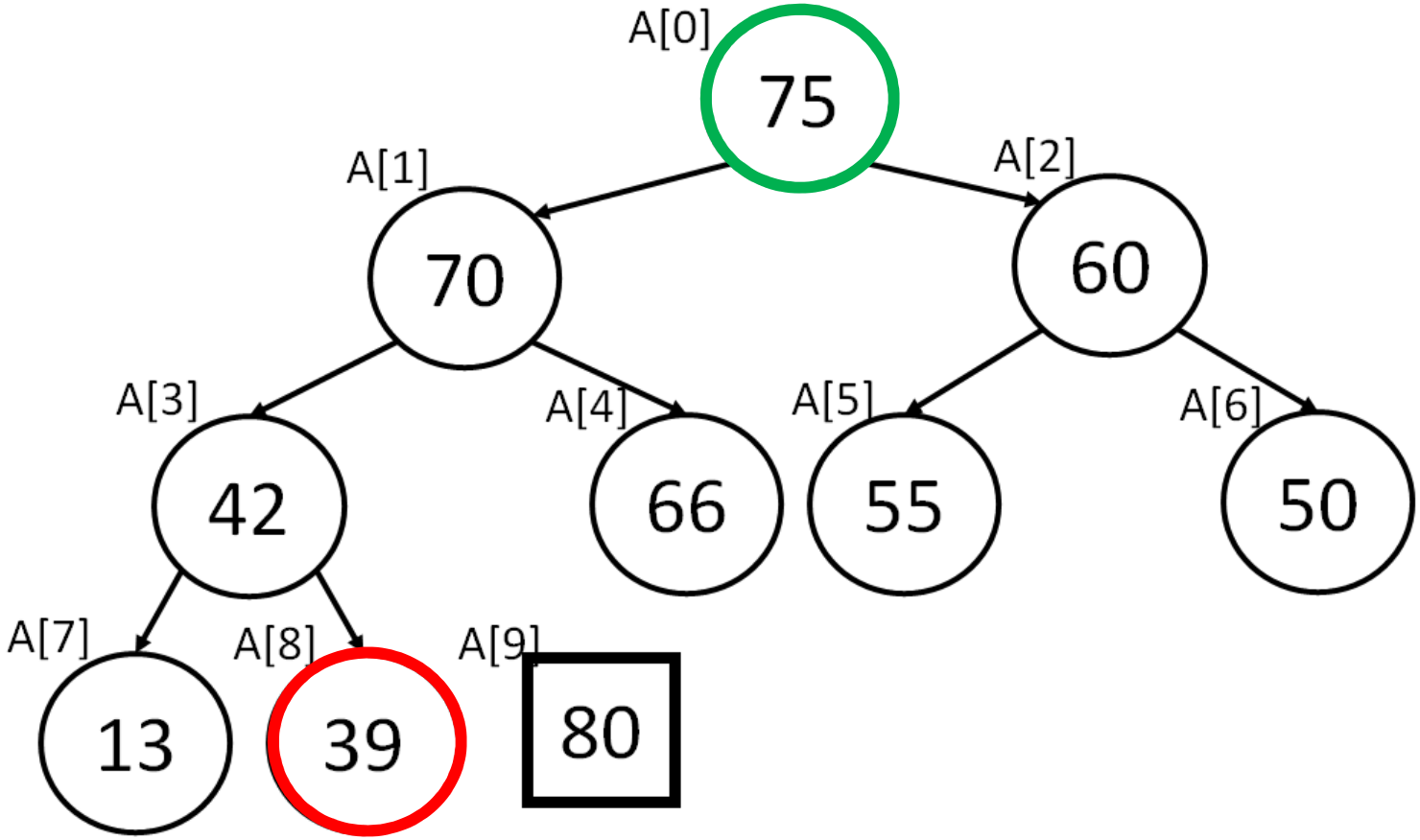
Construindo a Heap: método Heapifica()

i	0	1	2	3	4	5	6	7	8	9
A[i]	75	70	60	42	66	55	50	13	39	80



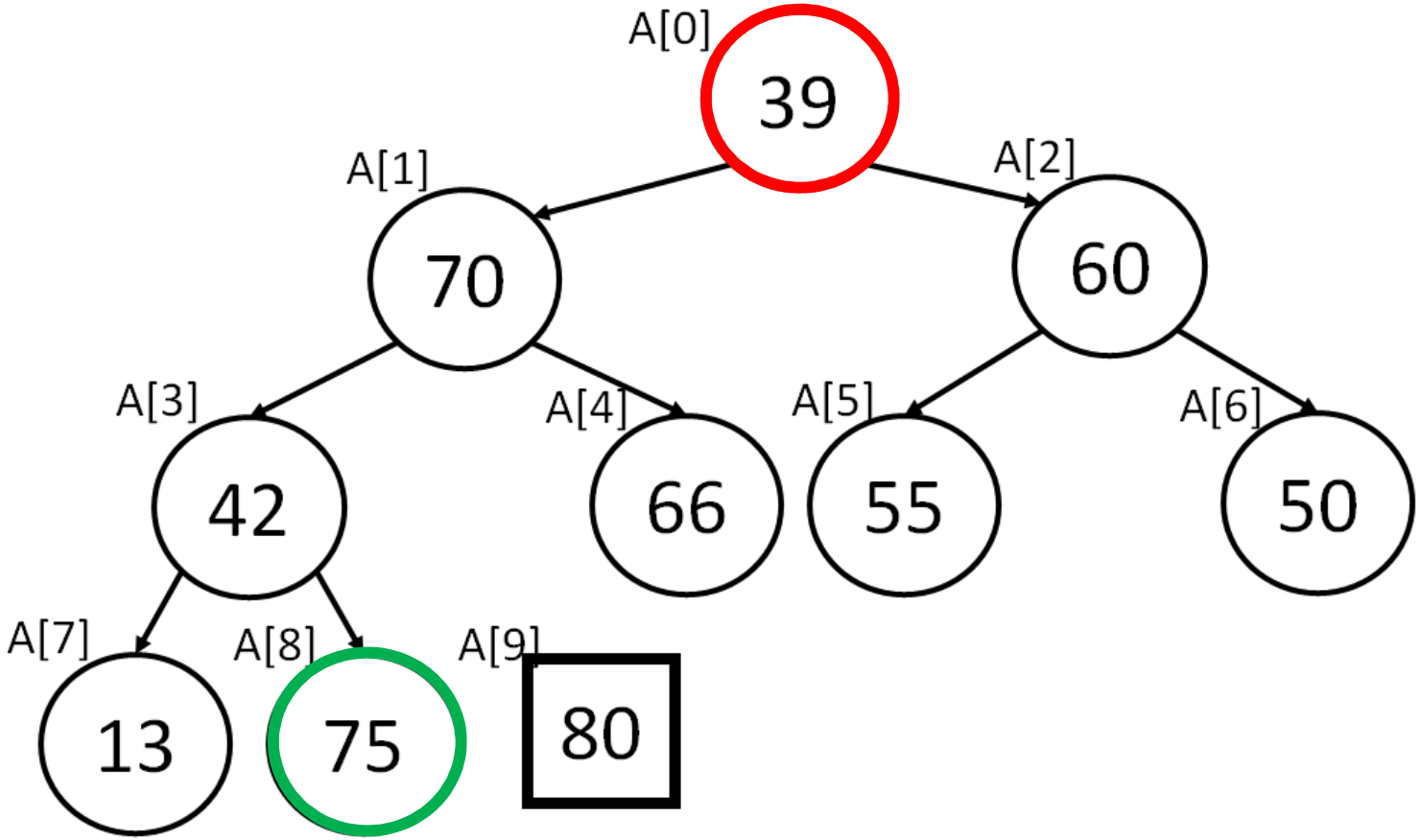
Ordenando Pela Raiz: Método Heapsort()

i	0	1	2	3	4	5	6	7	8	9
A[i]	75	70	60	42	66	55	50	13	39	80



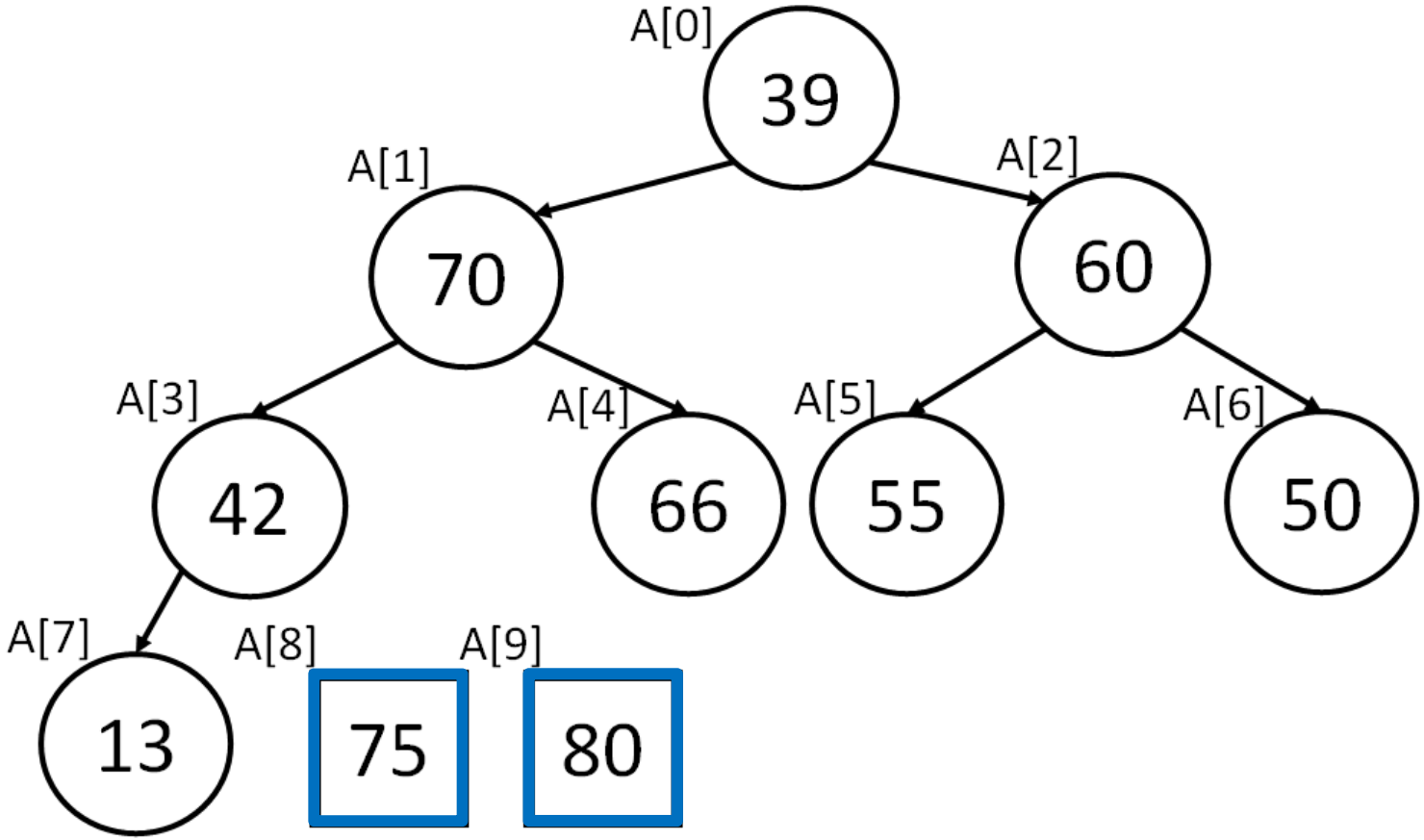
Ordenando Pela Raiz: Método Heapsort()

i	0	1	2	3	4	5	6	7	8	9
A[i]	39	70	60	42	66	55	50	13	75	80



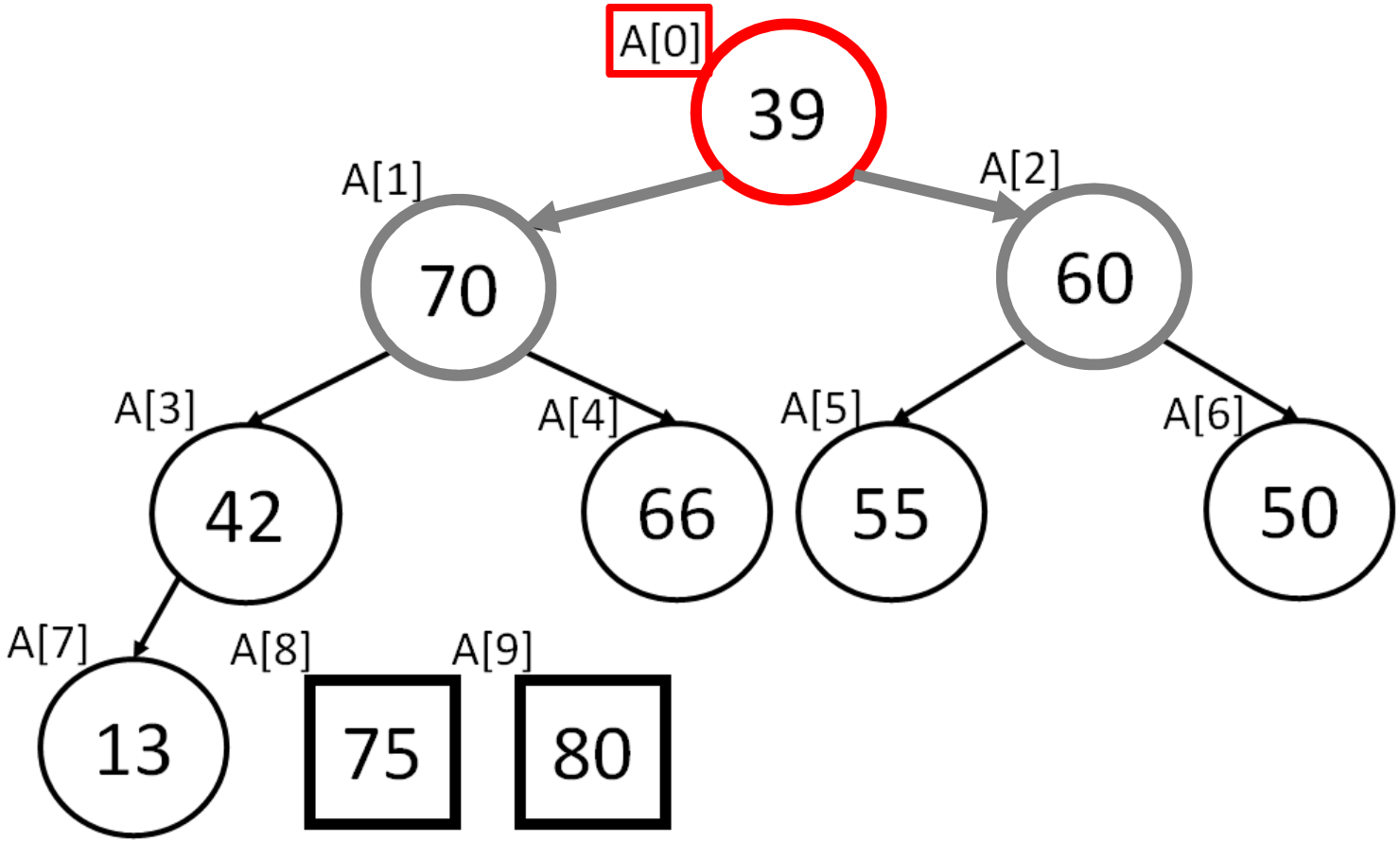
Ordenando Pela Raiz: Método Heapsort()

i	0	1	2	3	4	5	6	7	8	9
A[i]	39	70	60	42	66	55	50	13	75	80



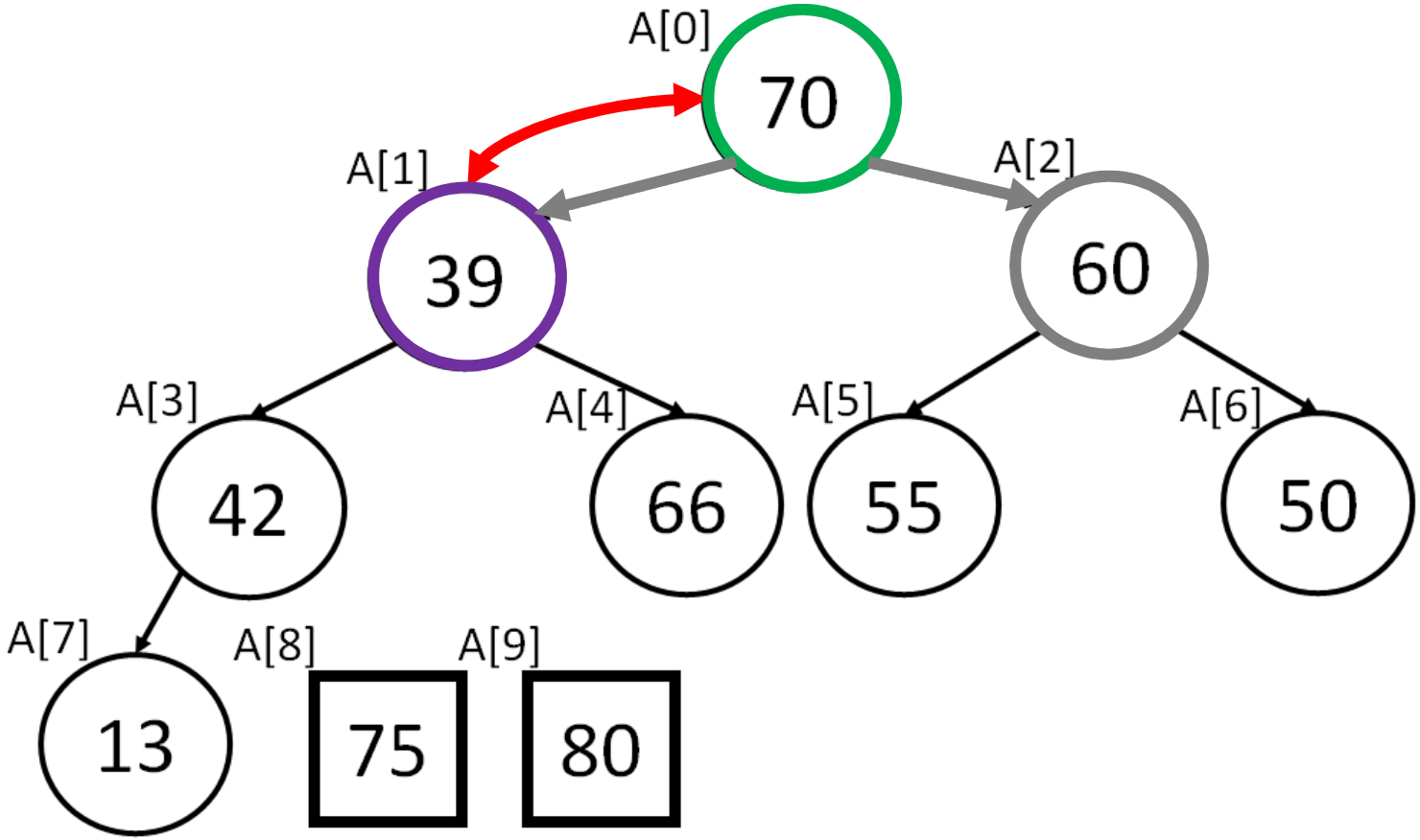
Ordenando Pela Raiz: Método Heapifica()

i	0	1	2	3	4	5	6	7	8	9
A[i]	39	70	60	42	66	55	50	13	75	80



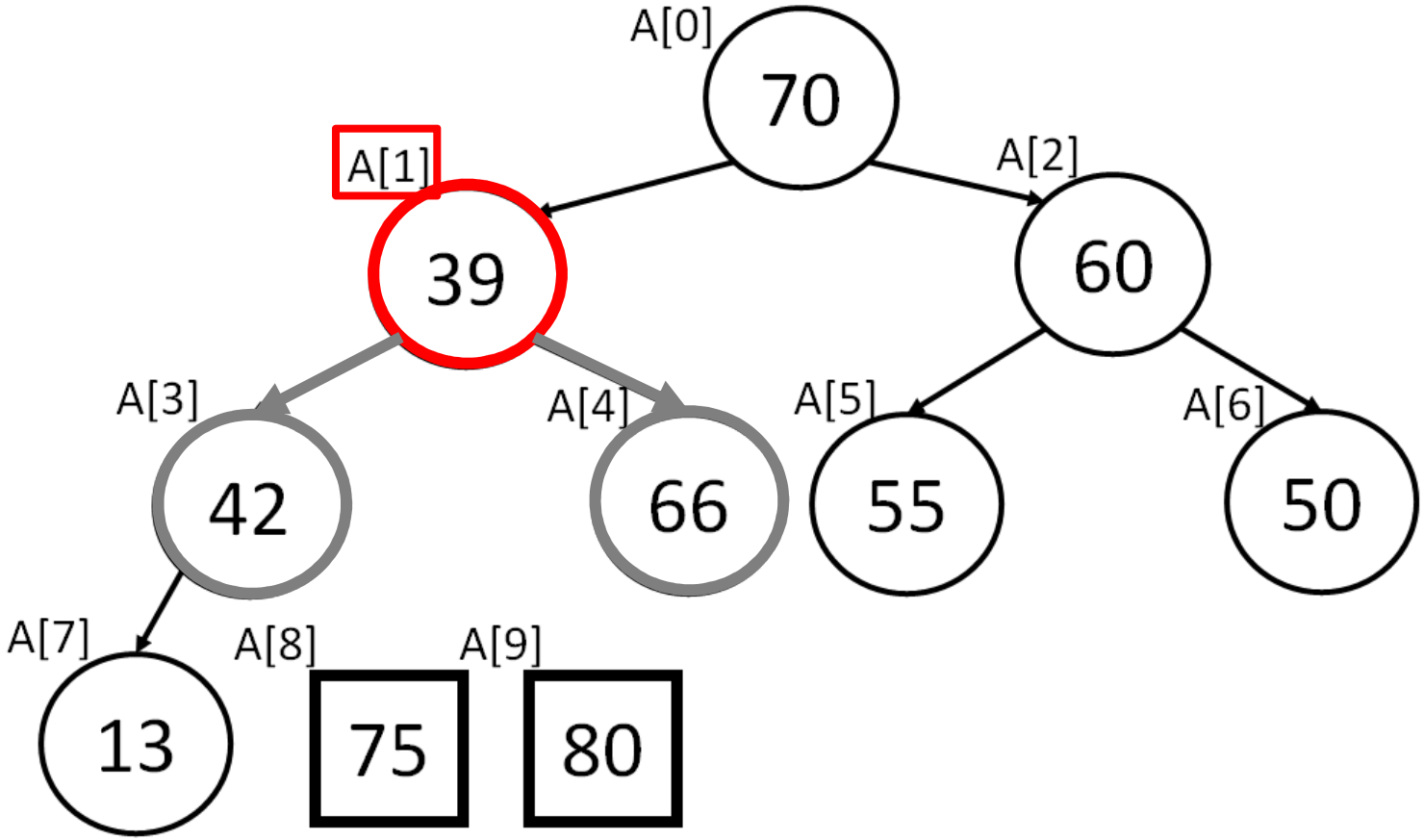
Ordenando Pela Raiz: Método Heapifica()

i	0	1	2	3	4	5	6	7	8	9
A[i]	70	39	60	42	66	55	50	13	75	80



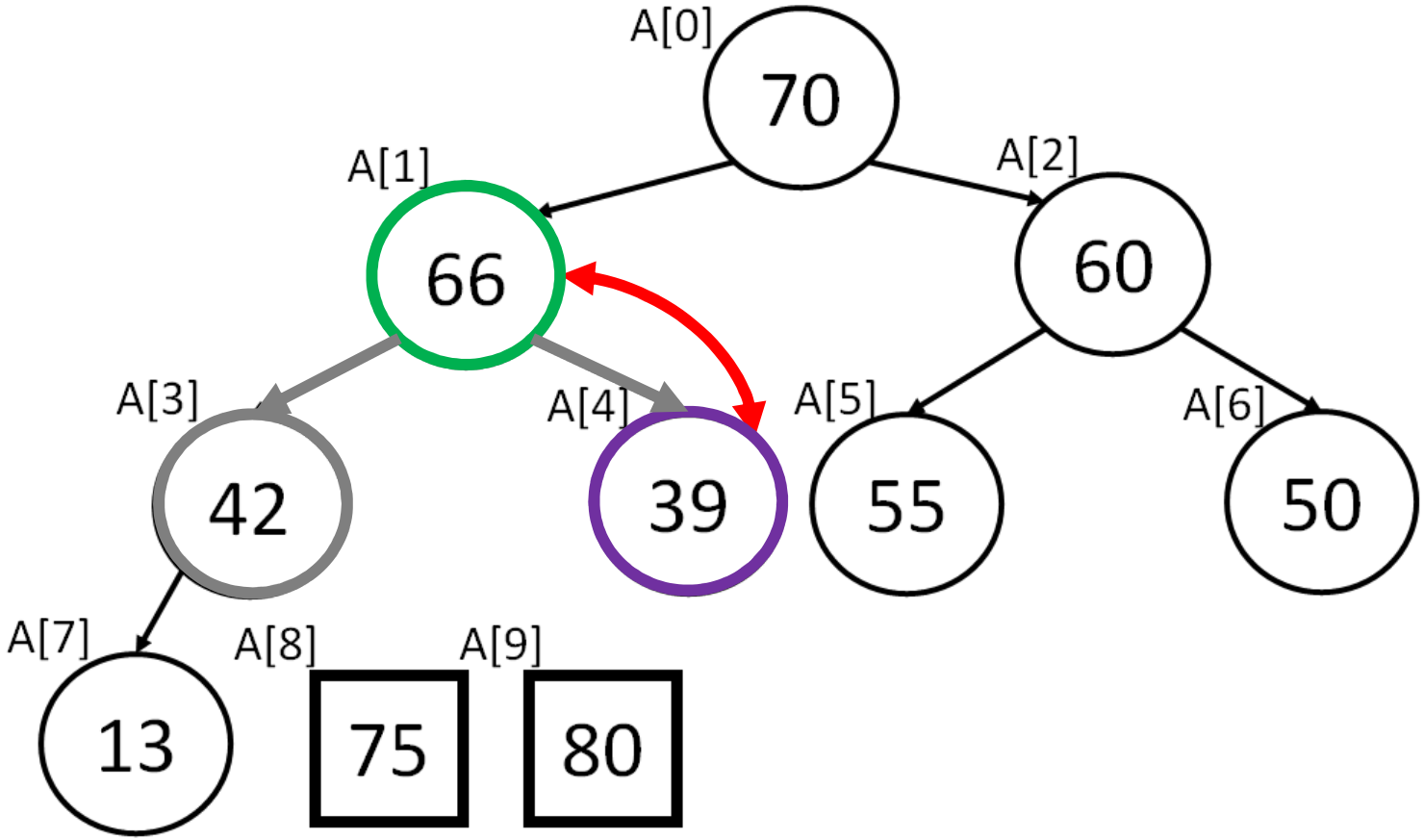
Ordenando Pela Raiz: Método Heapifica()

i	0	1	2	3	4	5	6	7	8	9
A[i]	70	39	60	42	66	55	50	13	75	80



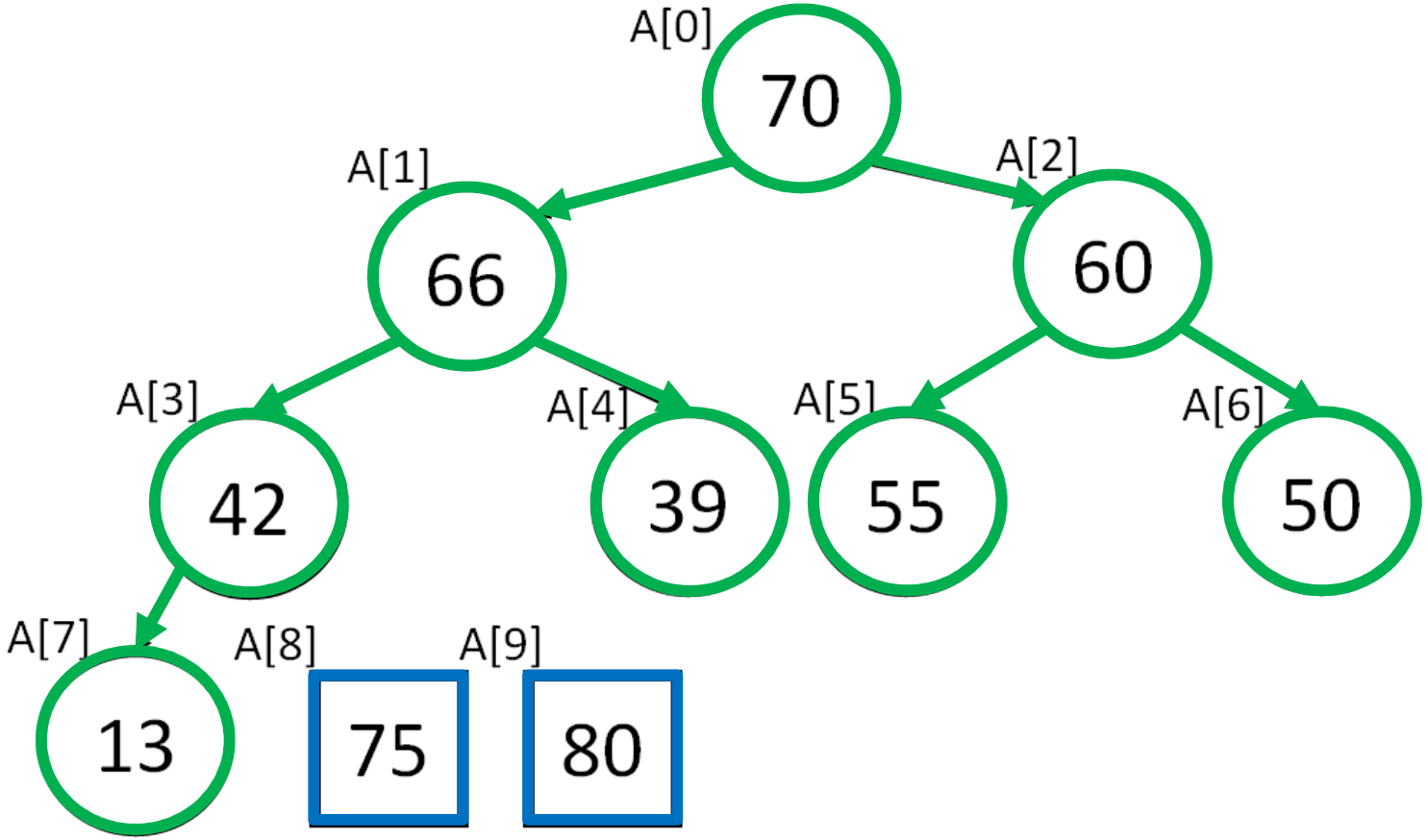
Ordenando Pela Raiz: Método Heapifica()

i	0	1	2	3	4	5	6	7	8	9
A[i]	70	66	60	42	39	55	50	13	75	80



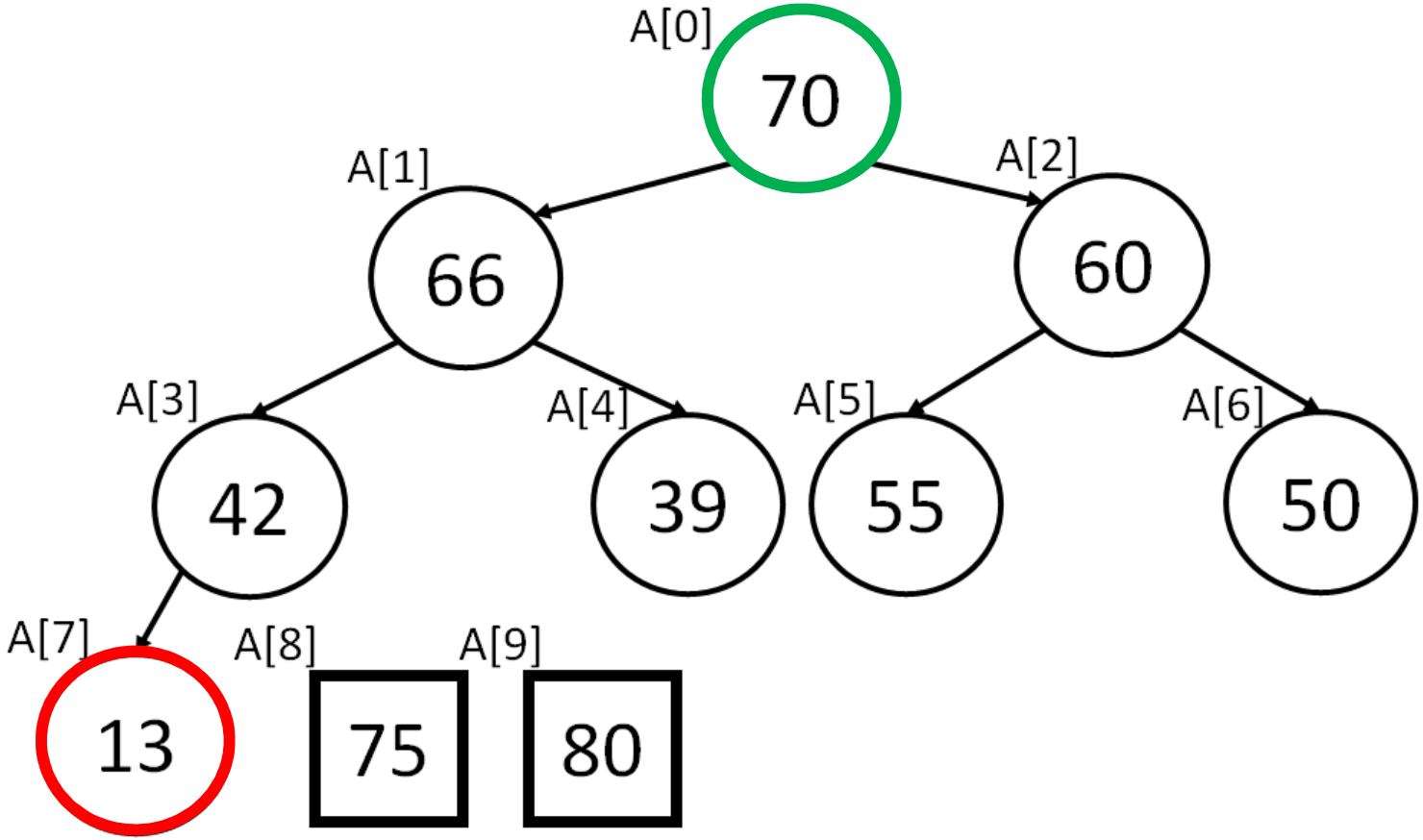
Construindo a Heap: método Heapifica()

i	0	1	2	3	4	5	6	7	8	9
A[i]	70	66	60	42	39	55	50	13	75	80



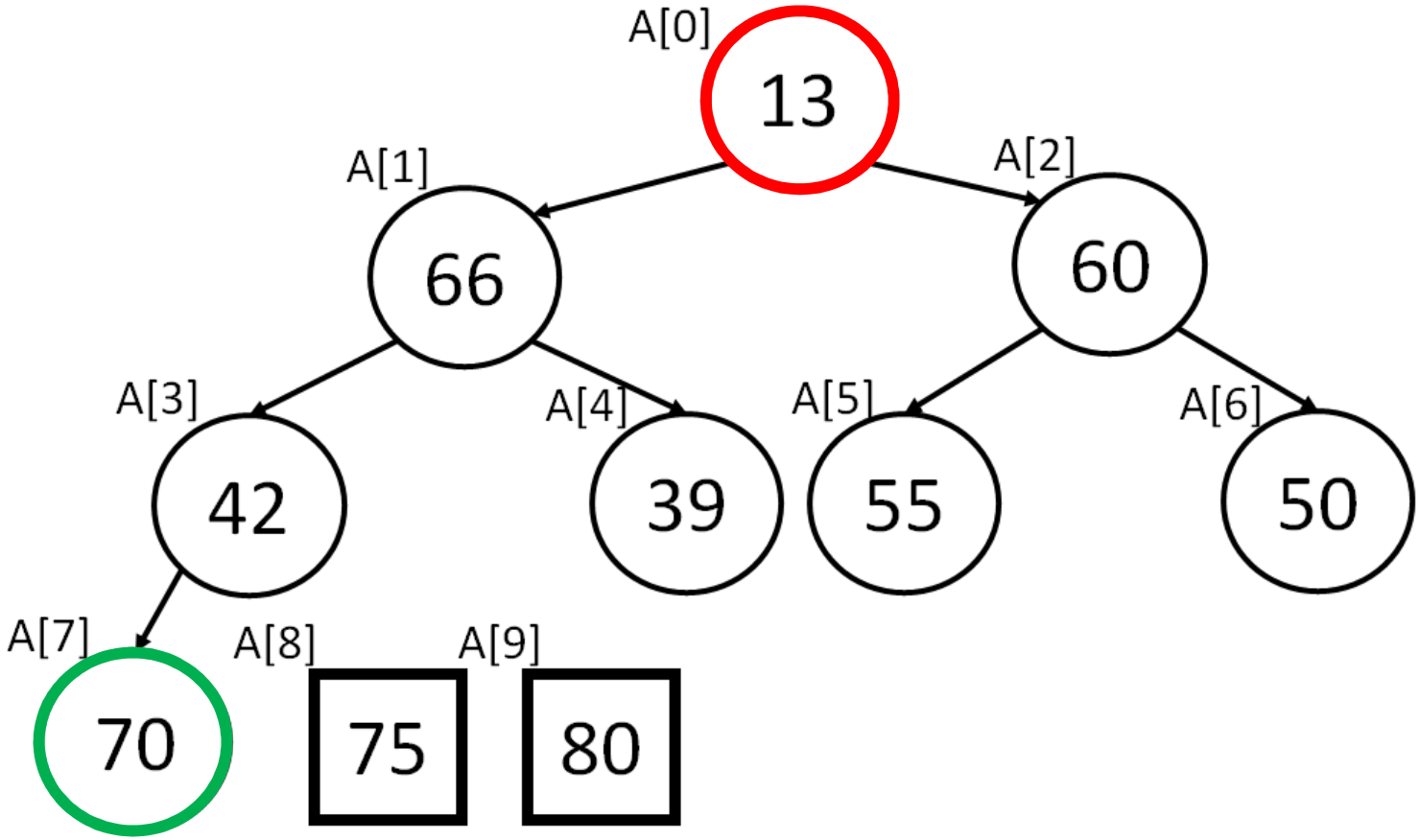
Construindo a Heap: método Heapsort()

i	0	1	2	3	4	5	6	7	8	9
A[i]	70	66	60	42	39	55	50	13	75	80



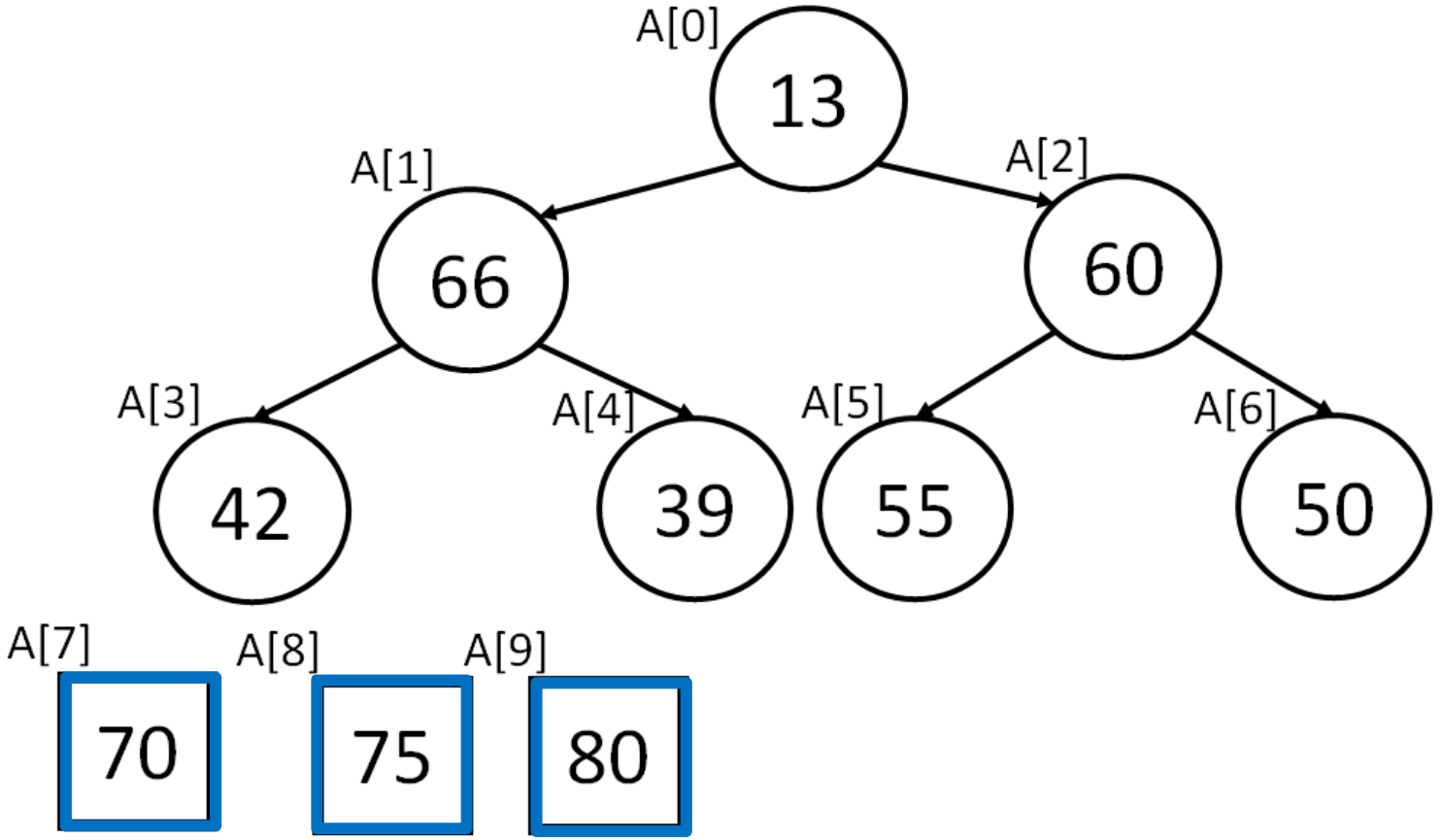
Construindo a Heap: método Heapsort()

i	0	1	2	3	4	5	6	7	8	9
A[i]	13	66	60	42	39	55	50	70	75	80



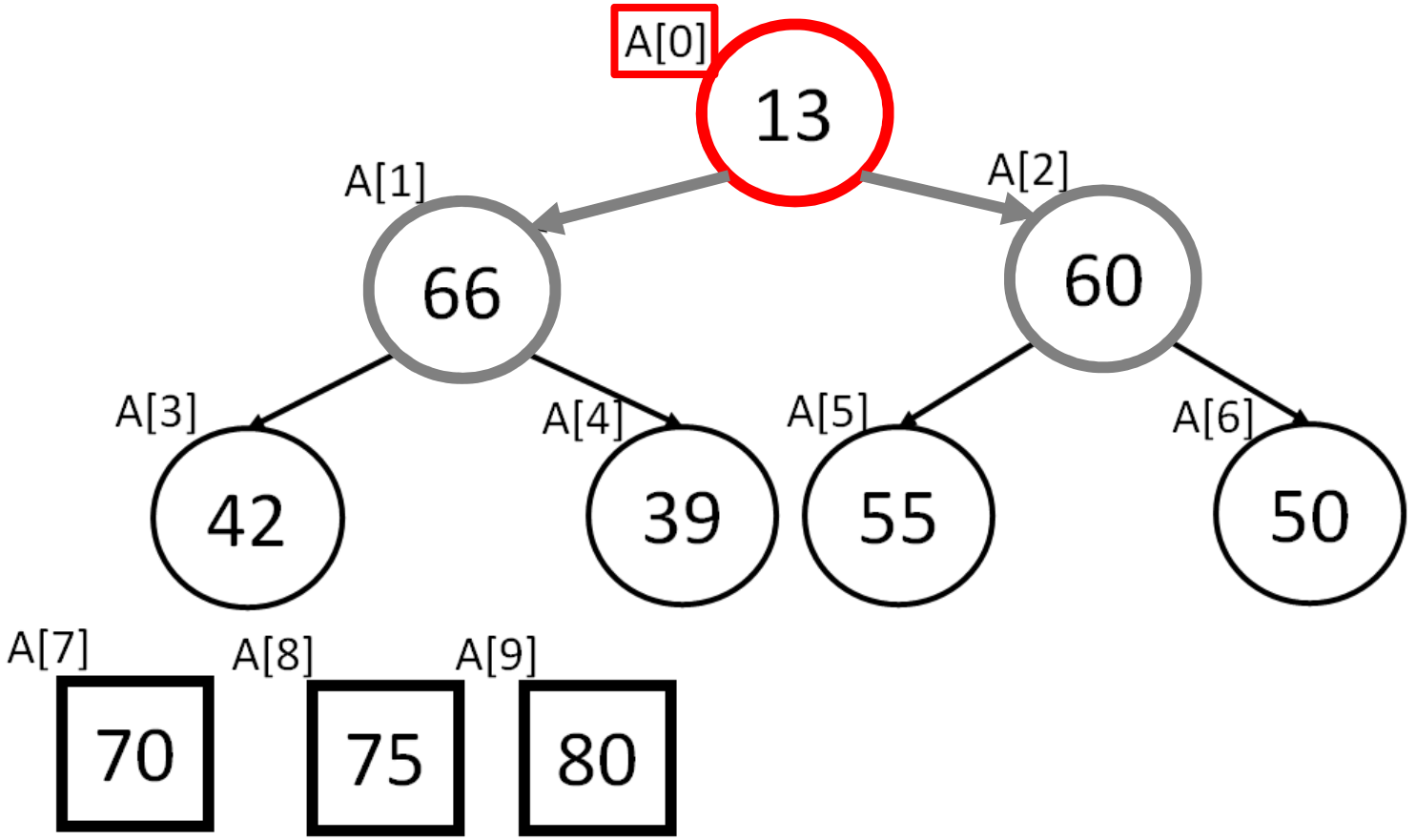
Construindo a Heap: método Heapsort()

i	0	1	2	3	4	5	6	7	8	9
A[i]	13	66	60	42	39	55	50	70	75	80



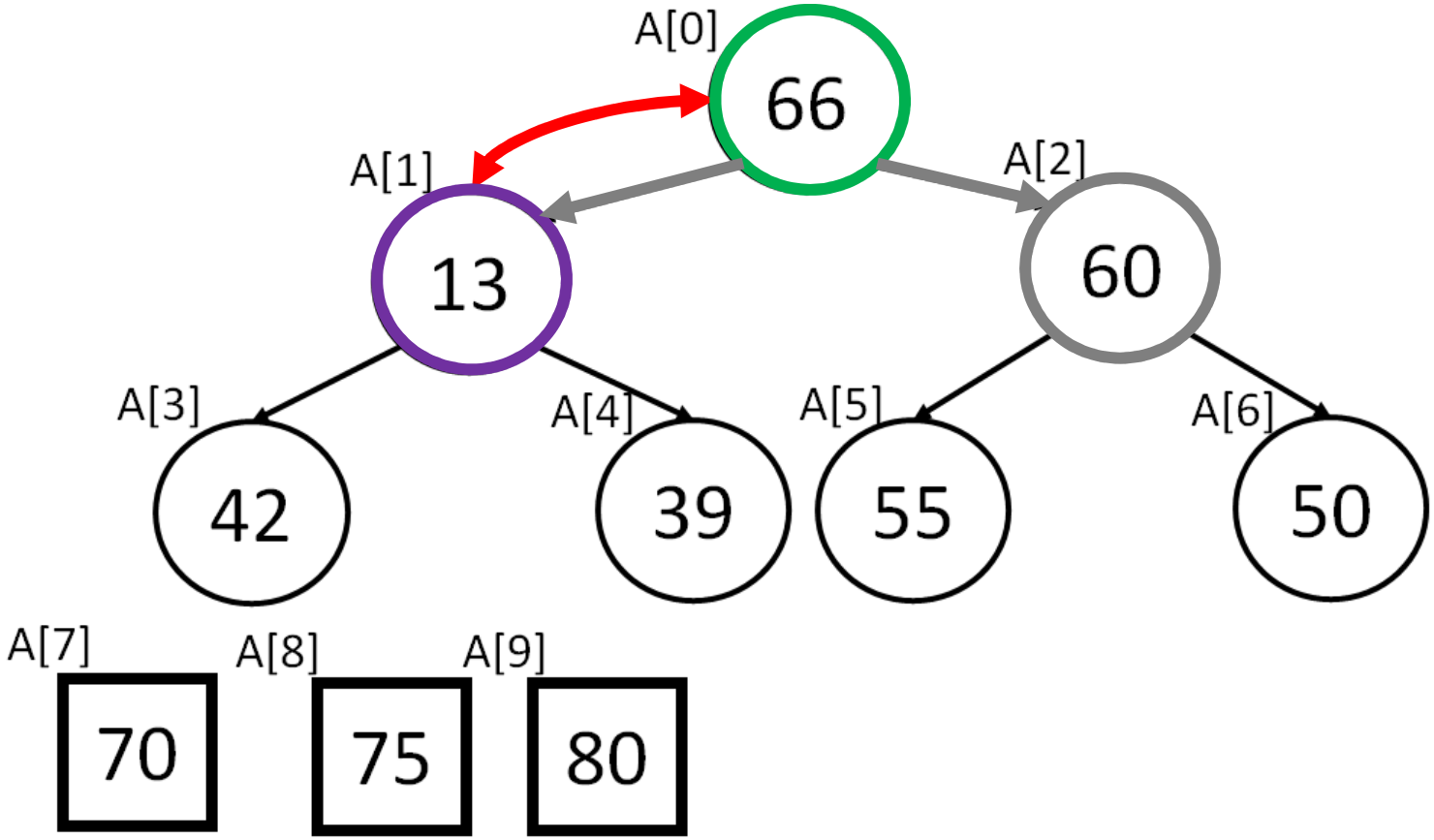
Construindo a Heap: método Heapifica()

i	0	1	2	3	4	5	6	7	8	9
A[i]	13	66	60	42	39	55	50	70	75	80



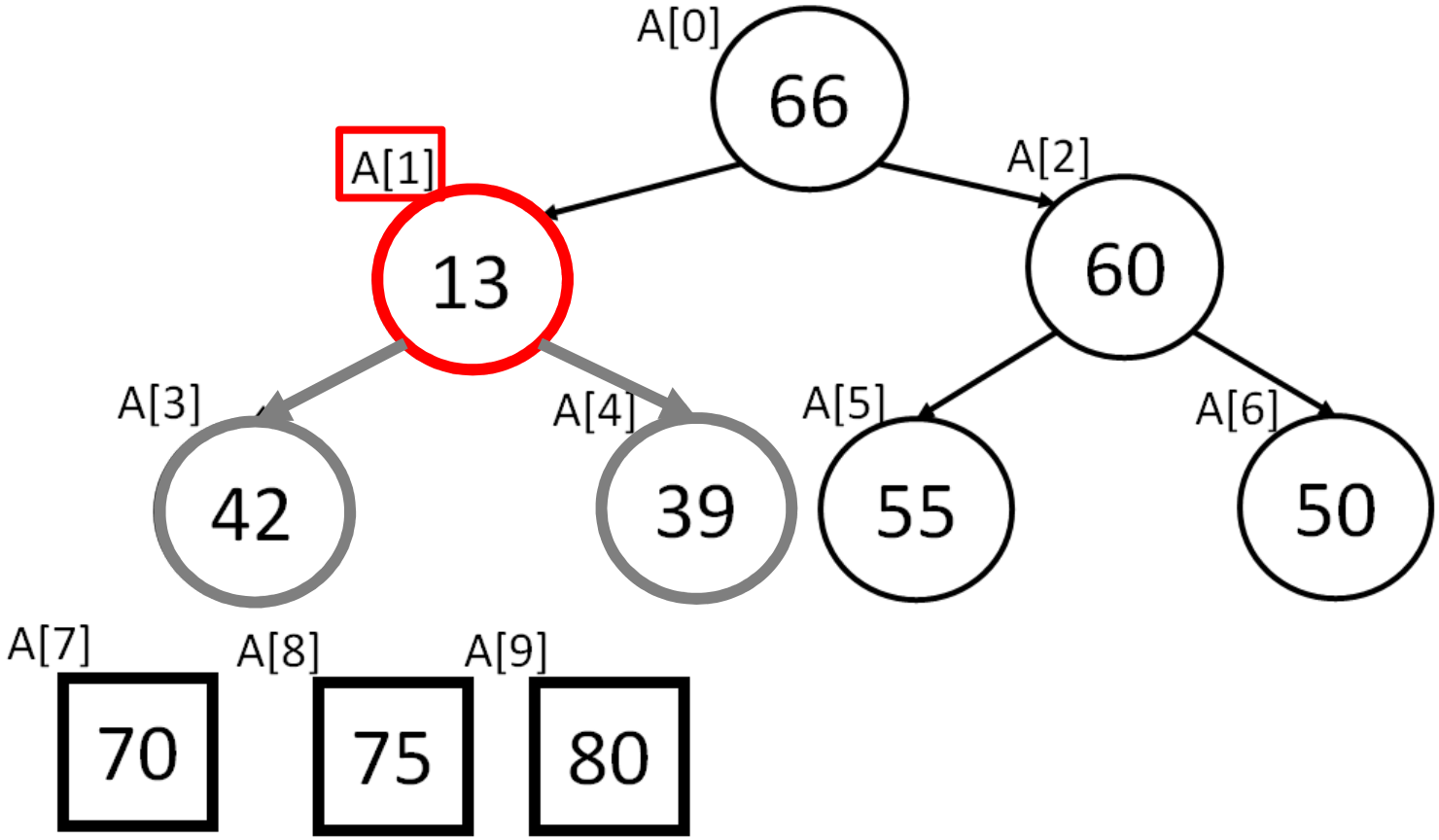
Construindo a Heap: método Heapifica()

i	0	1	2	3	4	5	6	7	8	9
A[i]	66	13	60	42	39	55	50	70	75	80



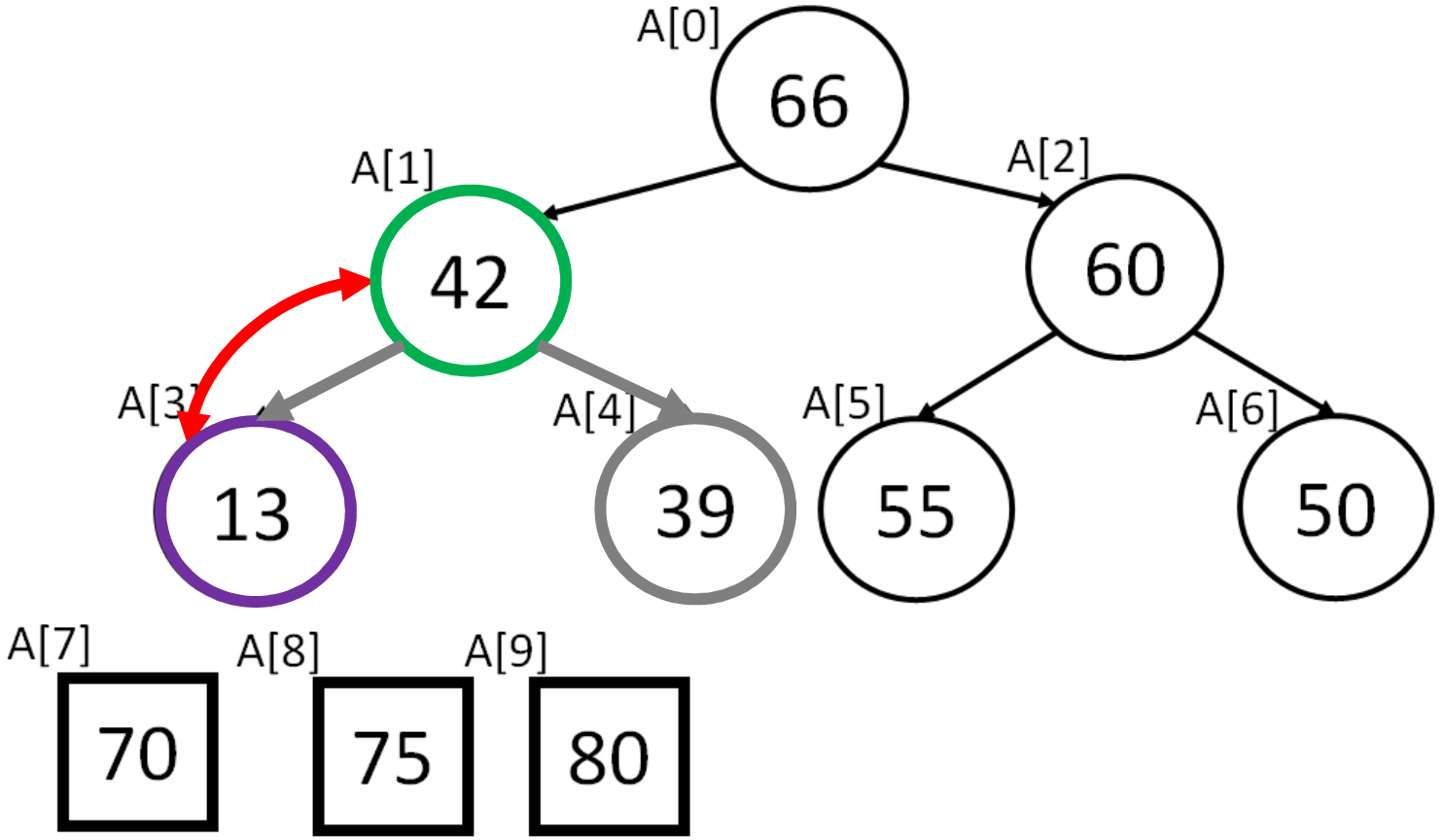
Construindo a Heap: método Heapifica()

i	0	1	2	3	4	5	6	7	8	9
A[i]	66	13	60	42	39	55	50	70	75	80



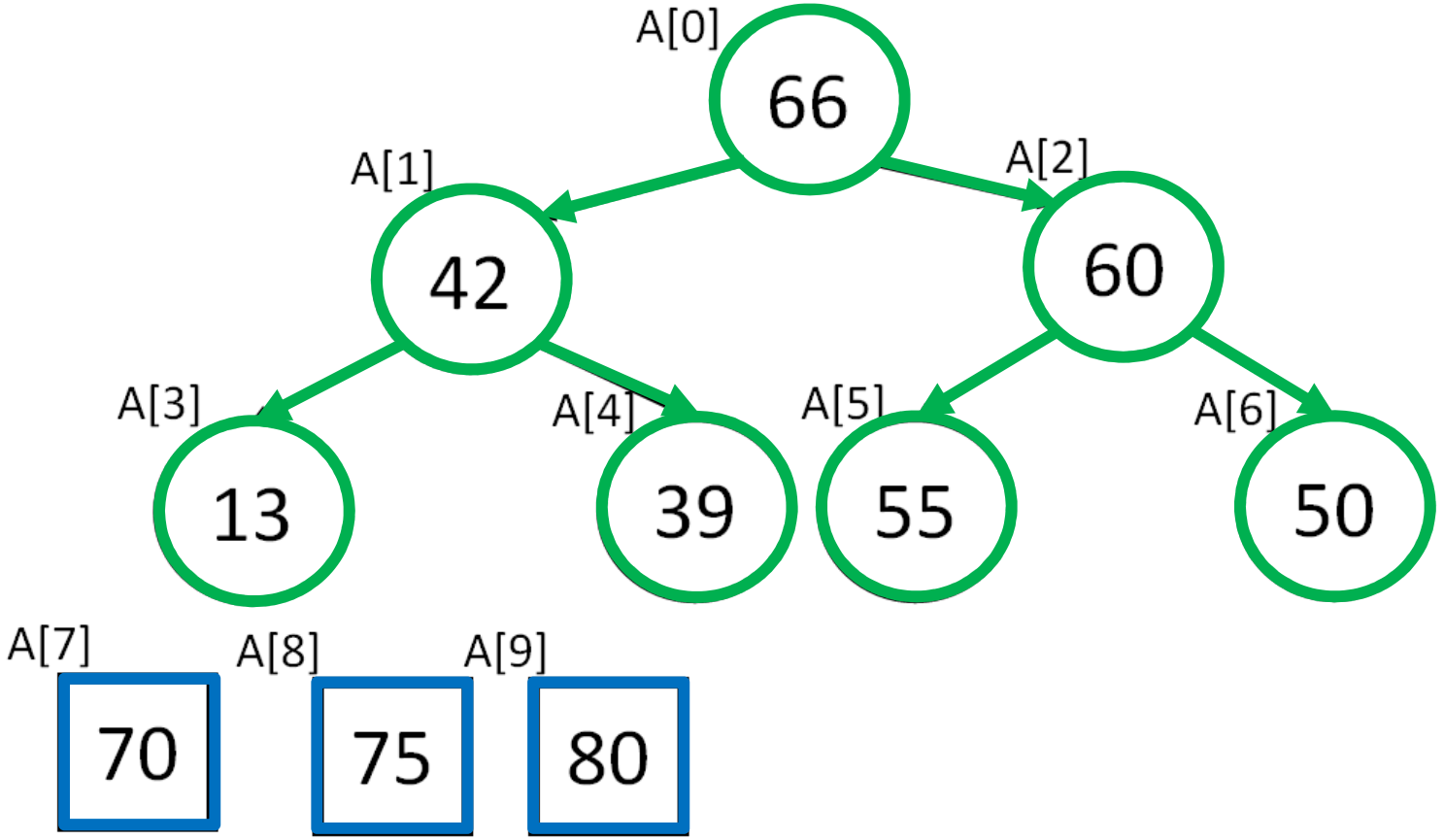
Construindo a Heap: método Heapifica()

i	0	1	2	3	4	5	6	7	8	9
A[i]	66	42	60	13	39	55	50	70	75	80



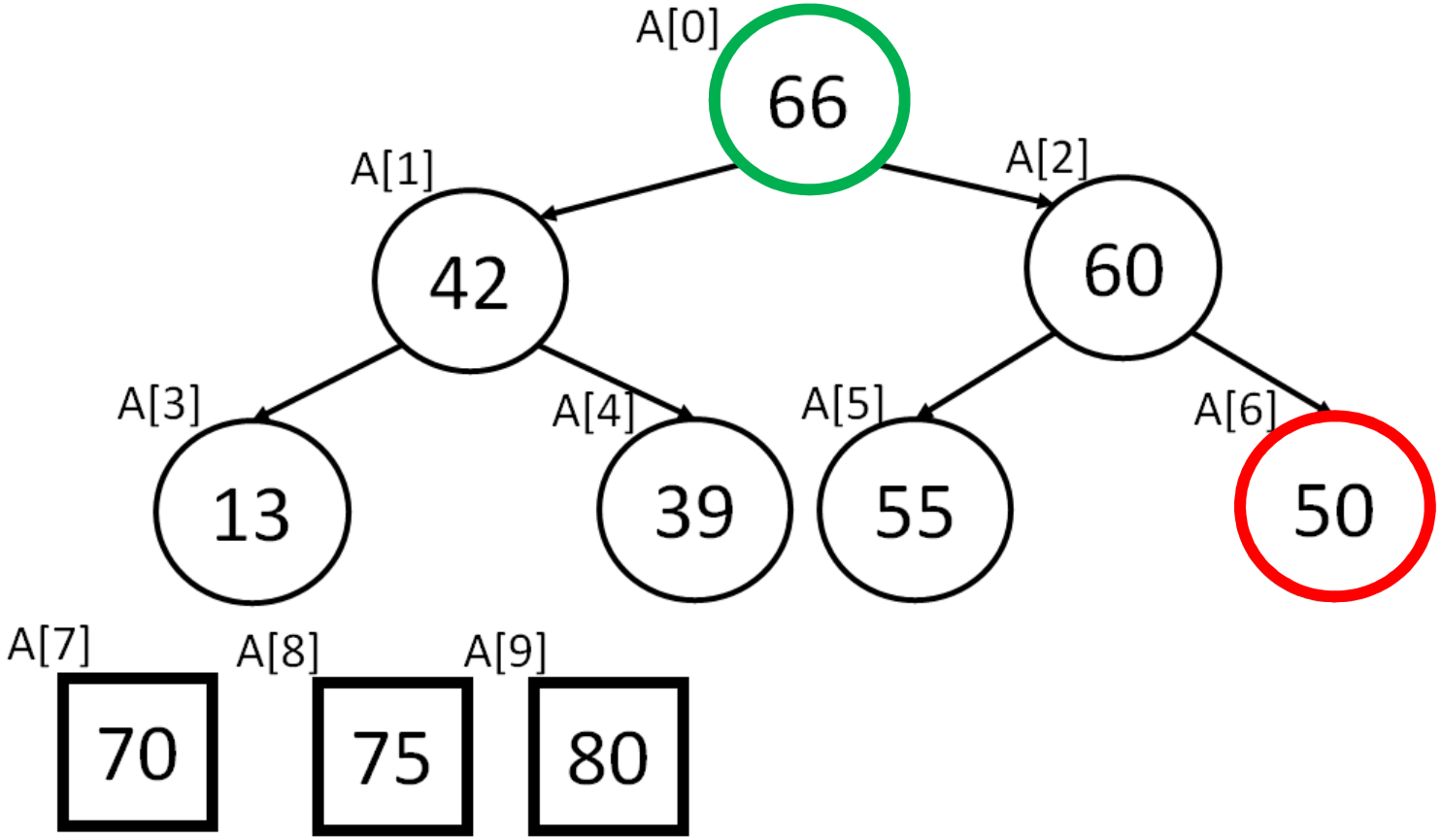
Construindo a Heap: método Heapifica()

i	0	1	2	3	4	5	6	7	8	9
A[i]	66	42	60	13	39	55	50	70	75	80



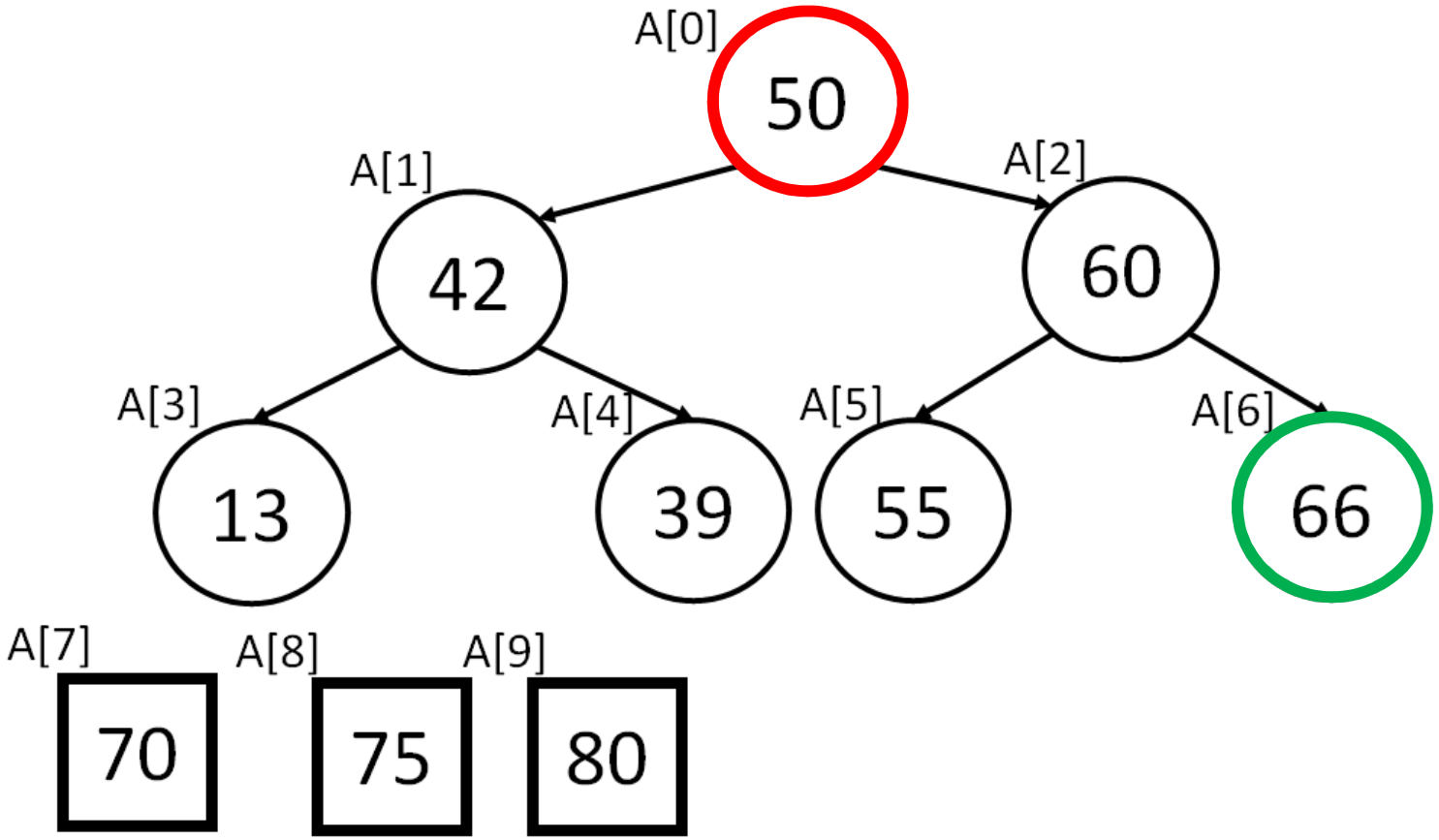
Construindo a Heap: método Heapsort()

i	0	1	2	3	4	5	6	7	8	9
A[i]	66	42	60	13	39	55	50	70	75	80



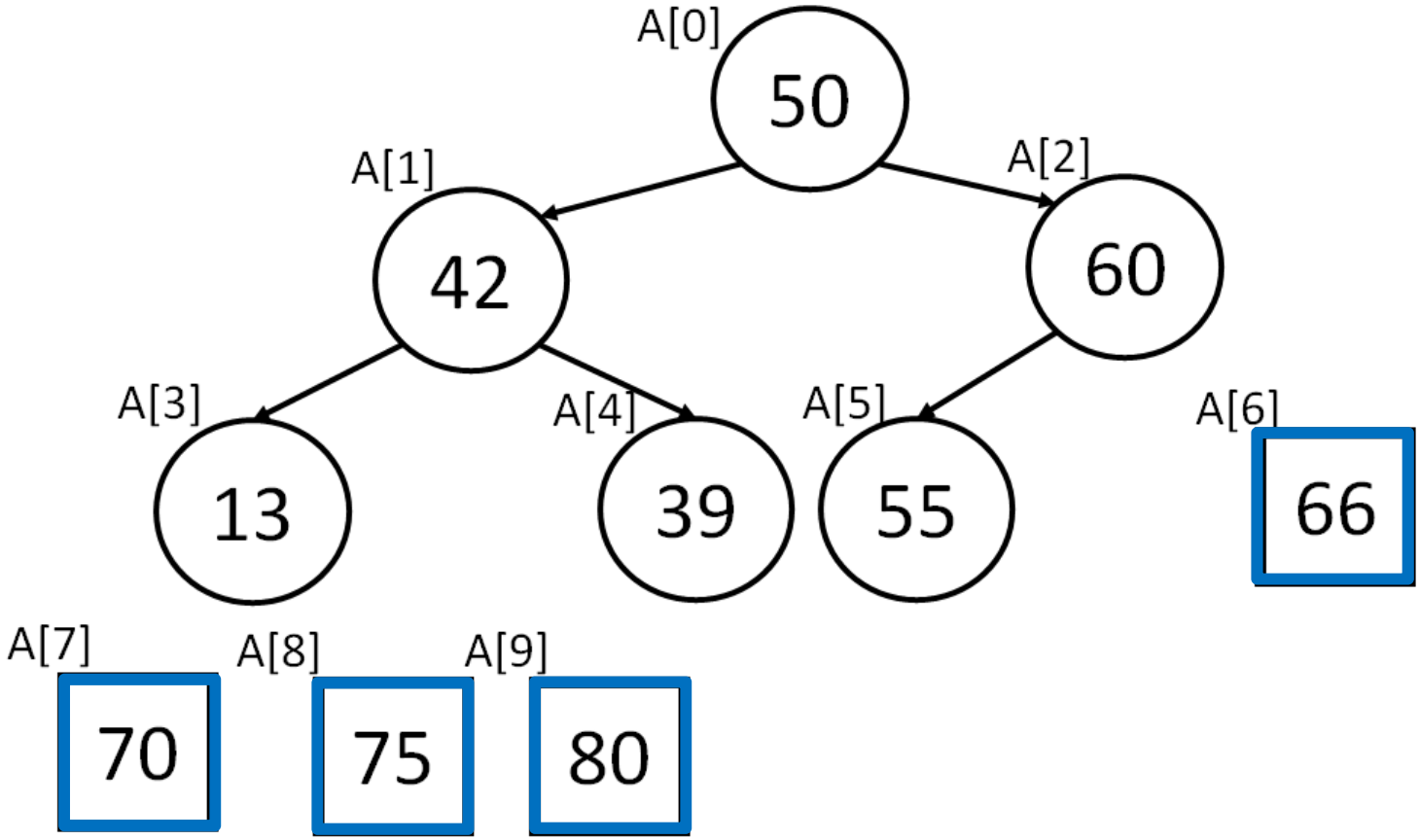
Construindo a Heap: método Heapsort()

i	0	1	2	3	4	5	6	7	8	9
A[i]	50	42	60	13	39	55	66	70	75	80



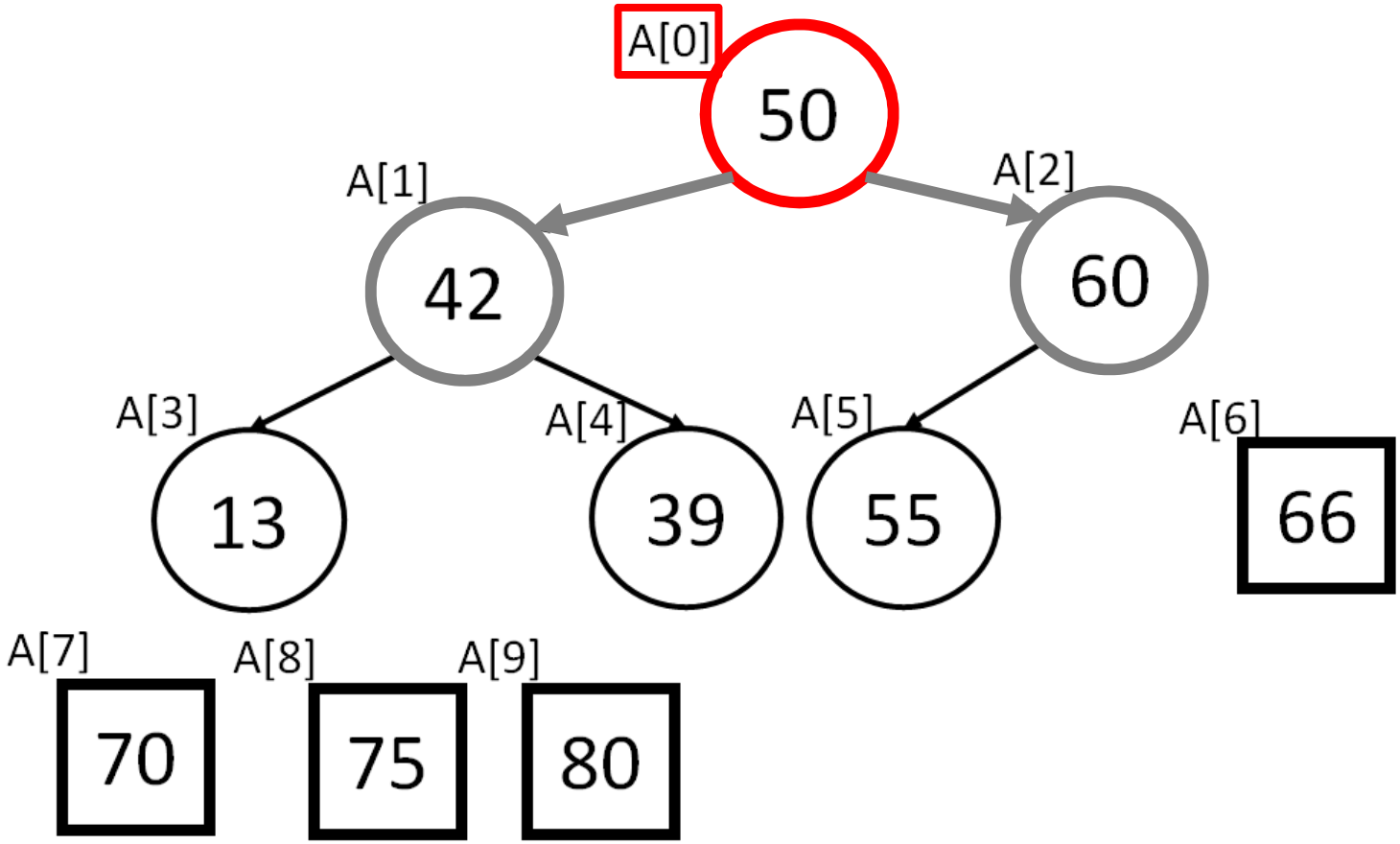
Construindo a Heap: método Heapsort()

i	0	1	2	3	4	5	6	7	8	9
A[i]	50	42	60	13	39	55	66	70	75	80



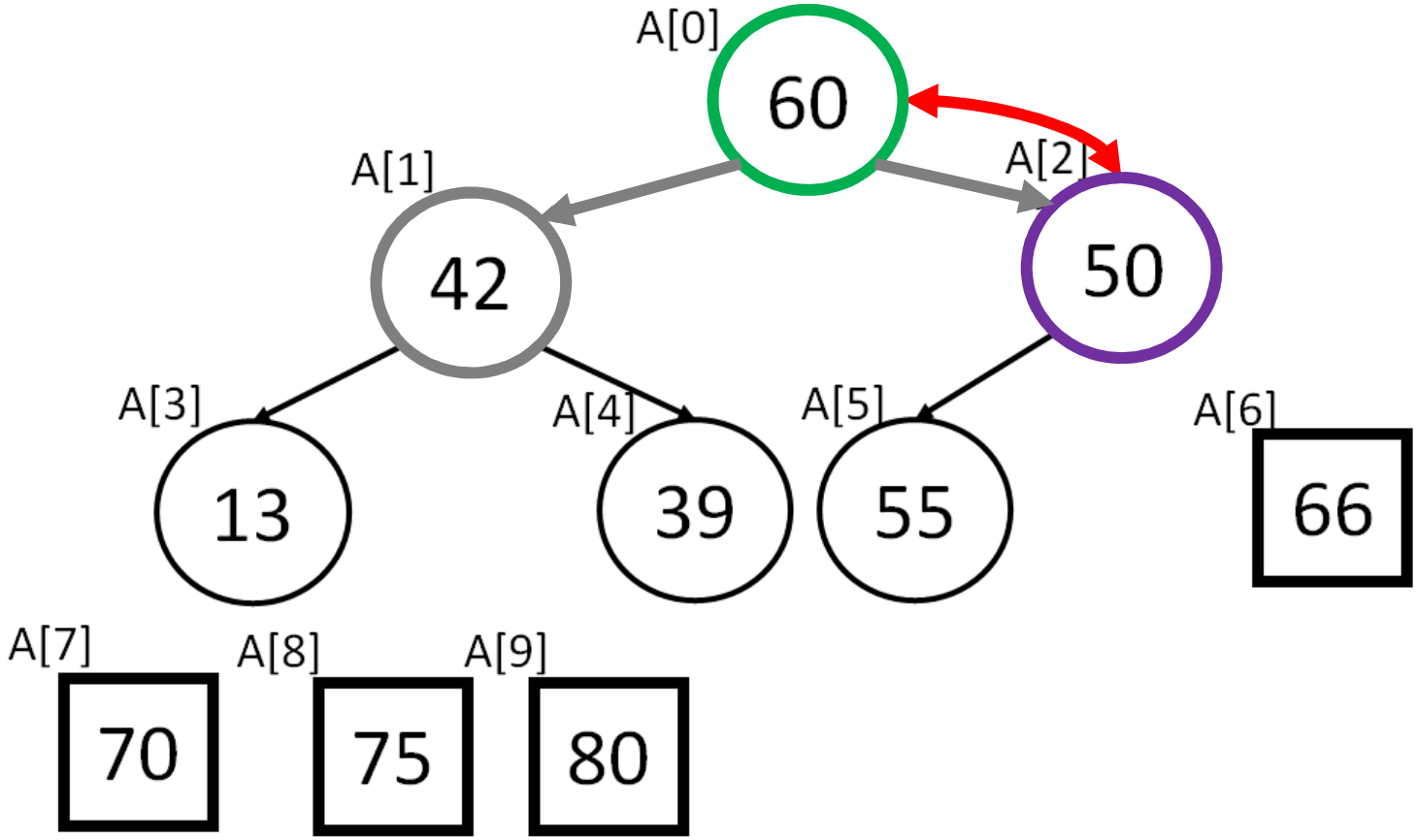
Construindo a Heap: método Heapifica()

i	0	1	2	3	4	5	6	7	8	9
A[i]	50	42	60	13	39	55	66	70	75	80



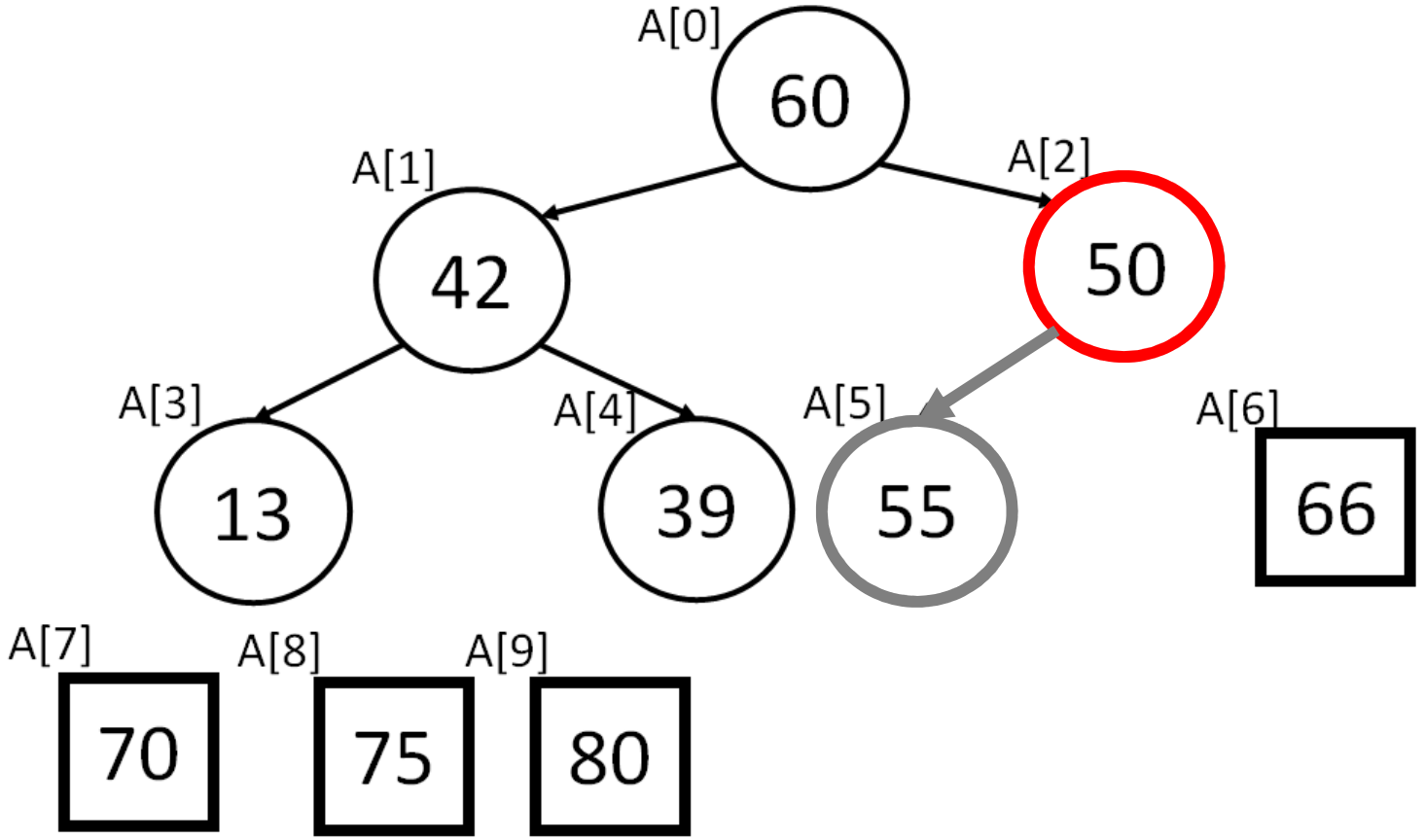
Construindo a Heap: método Heapifica()

i	0	1	2	3	4	5	6	7	8	9
A[i]	60	42	50	13	39	55	66	70	75	80



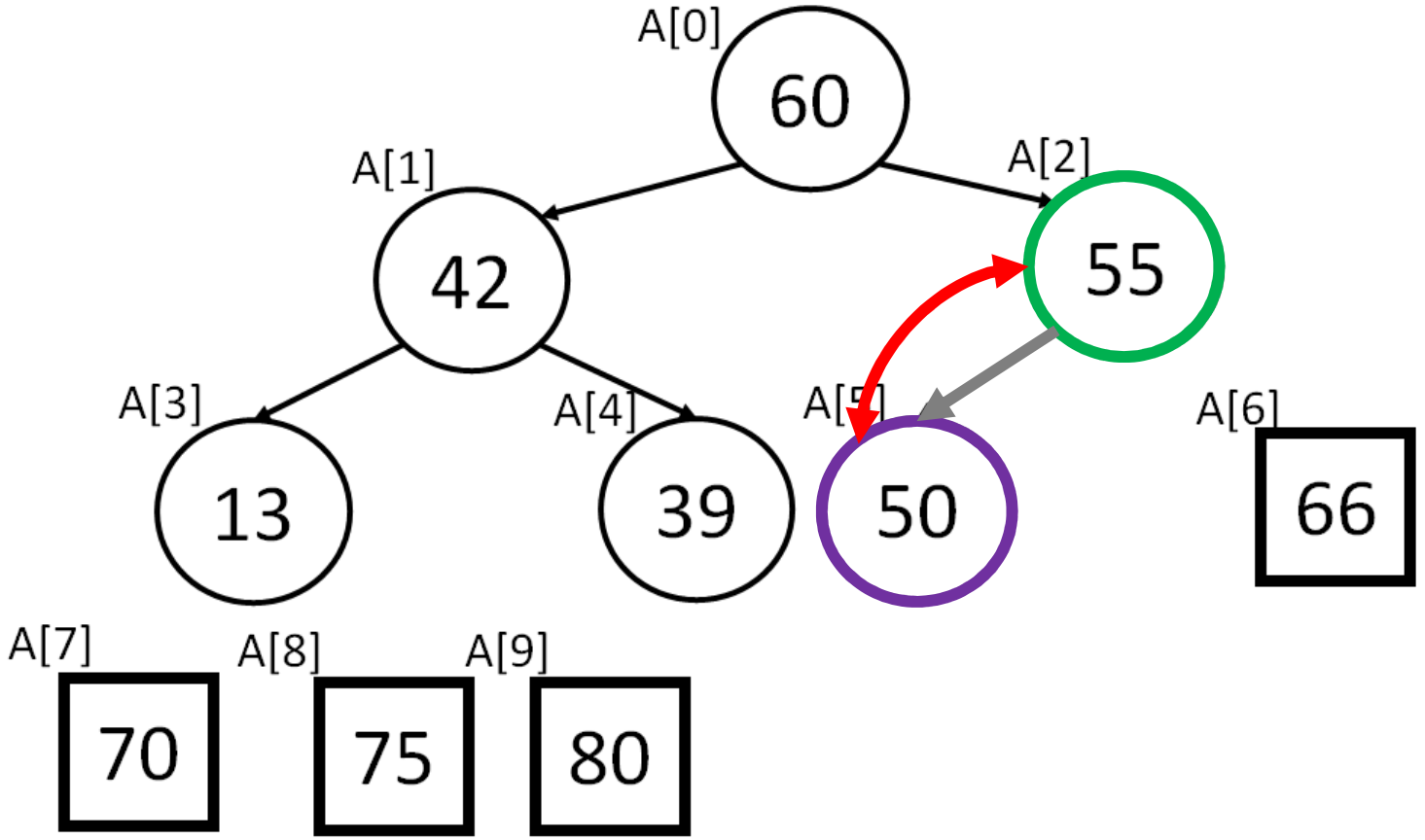
Construindo a Heap: método Heapifica()

i	0	1	2	3	4	5	6	7	8	9
A[i]	60	42	50	13	39	55	66	70	75	80



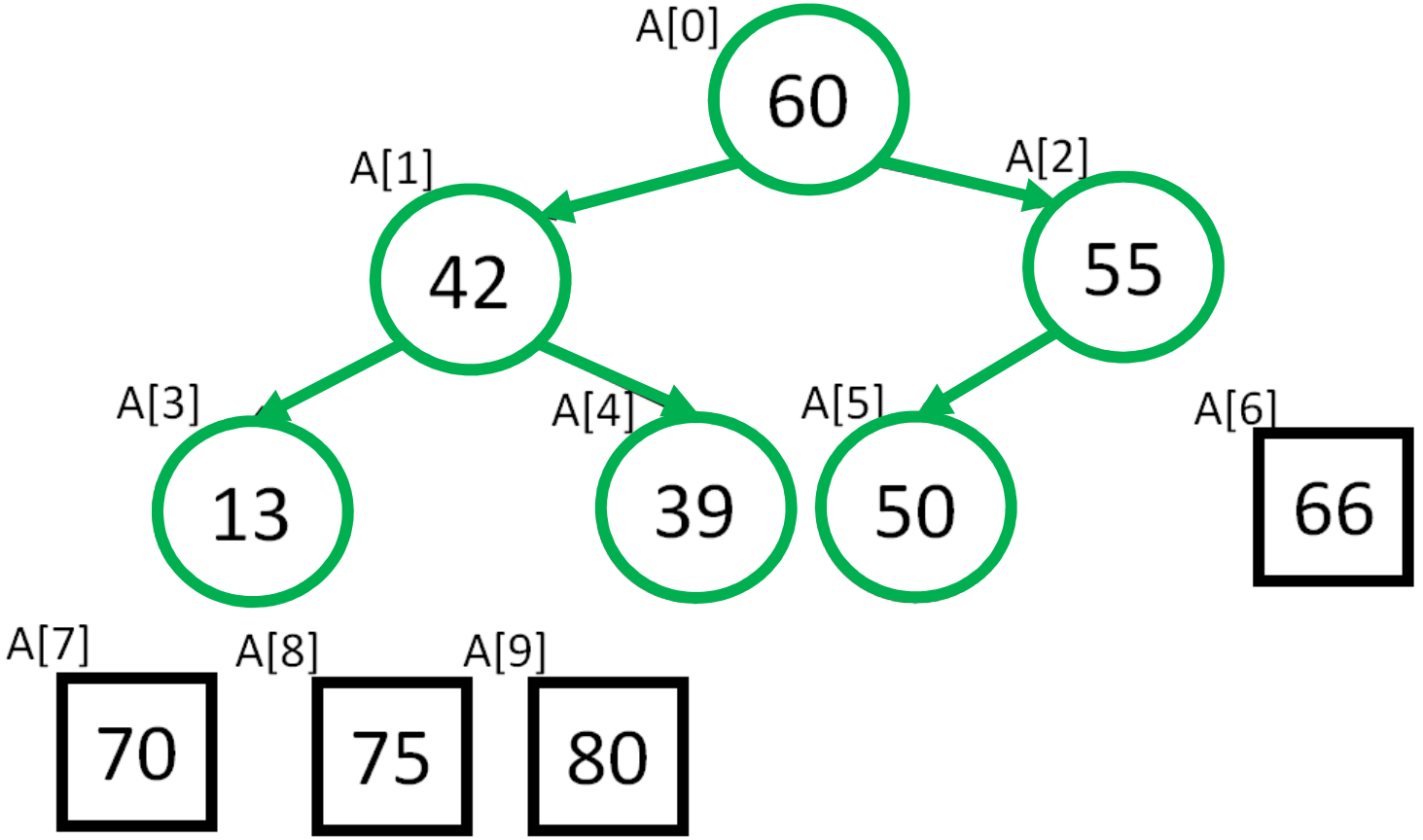
Construindo a Heap: método Heapifica()

i	0	1	2	3	4	5	6	7	8	9
A[i]	60	42	55	13	39	50	66	70	75	80



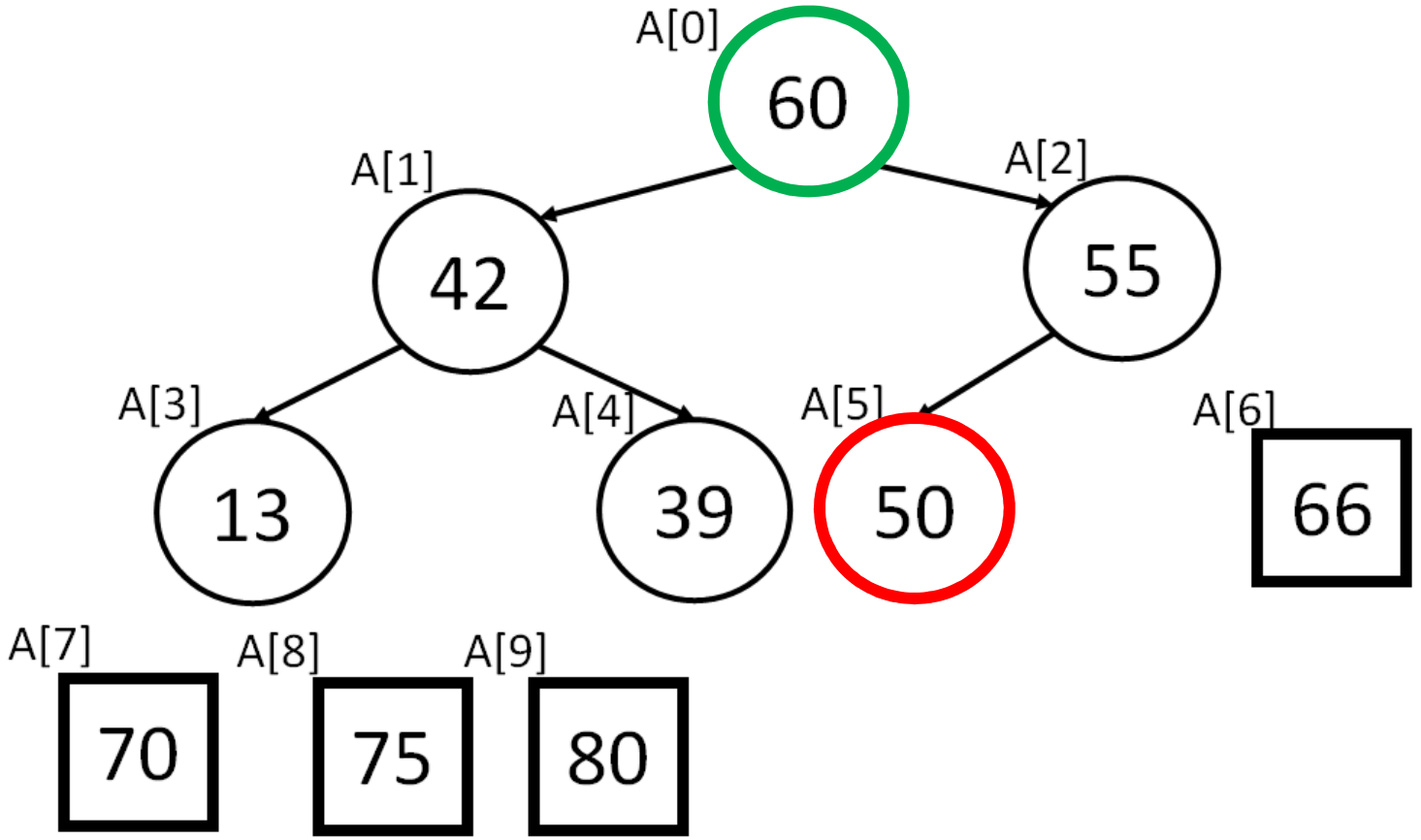
Construindo a Heap: método Heapifica()

i	0	1	2	3	4	5	6	7	8	9
A[i]	60	42	55	13	39	50	66	70	75	80



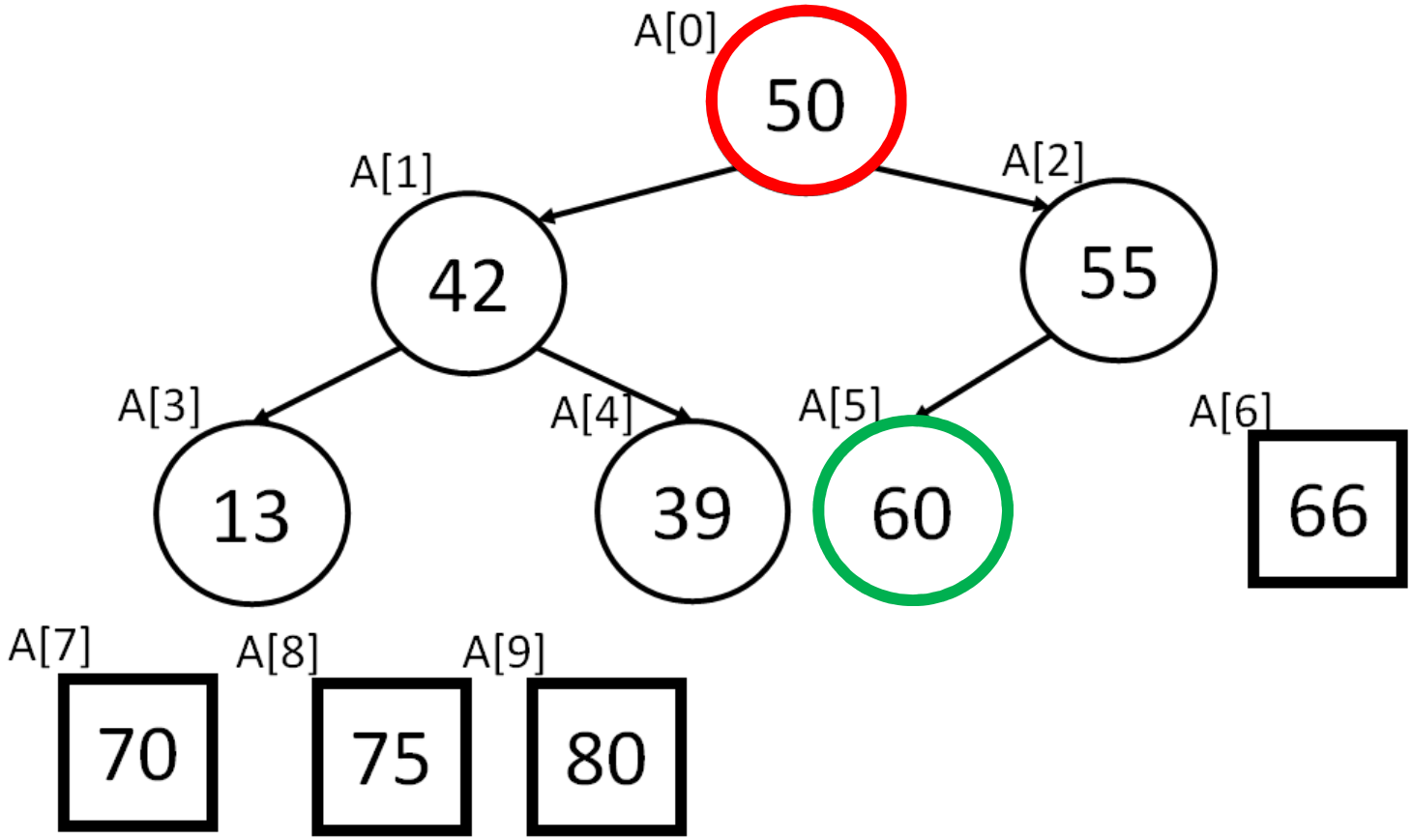
Construindo a Heap: método Heapsort()

i	0	1	2	3	4	5	6	7	8	9
A[i]	60	42	55	13	39	50	66	70	75	80



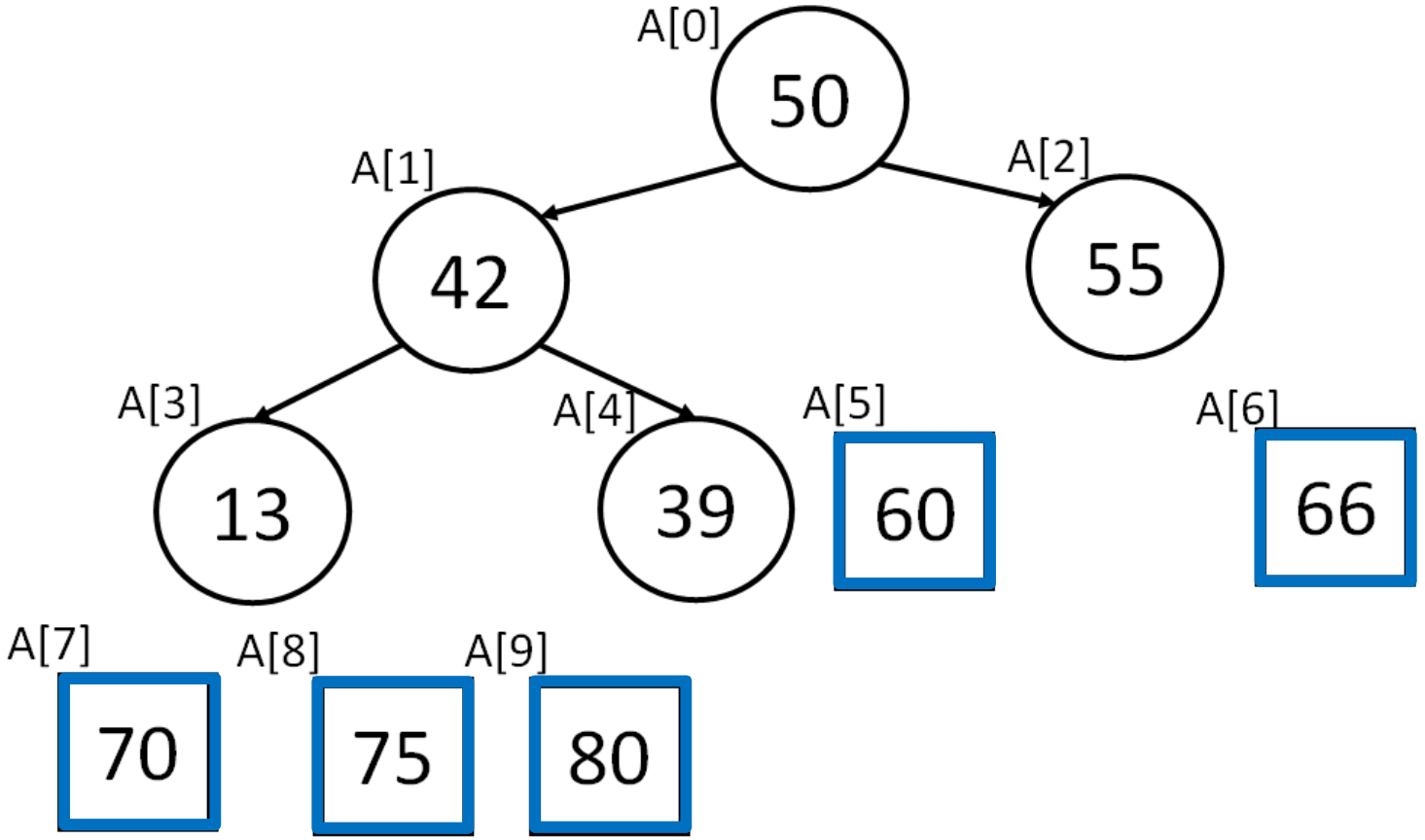
Construindo a Heap: método Heapsort()

i	0	1	2	3	4	5	6	7	8	9
A[i]	50	42	55	13	39	60	66	70	75	80



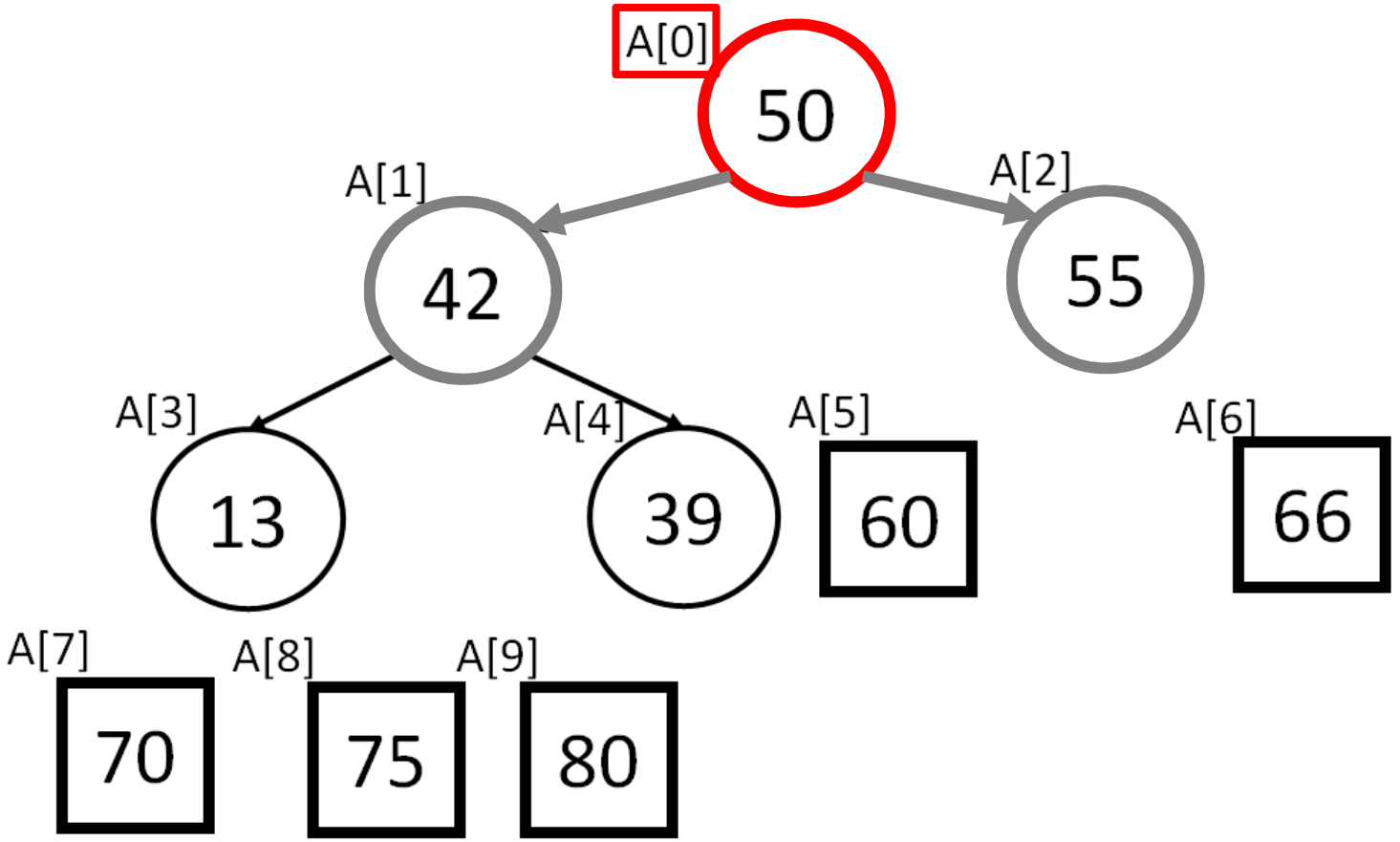
Construindo a Heap: método Heapsort()

i	0	1	2	3	4	5	6	7	8	9
A[i]	50	42	55	13	39	60	66	70	75	80



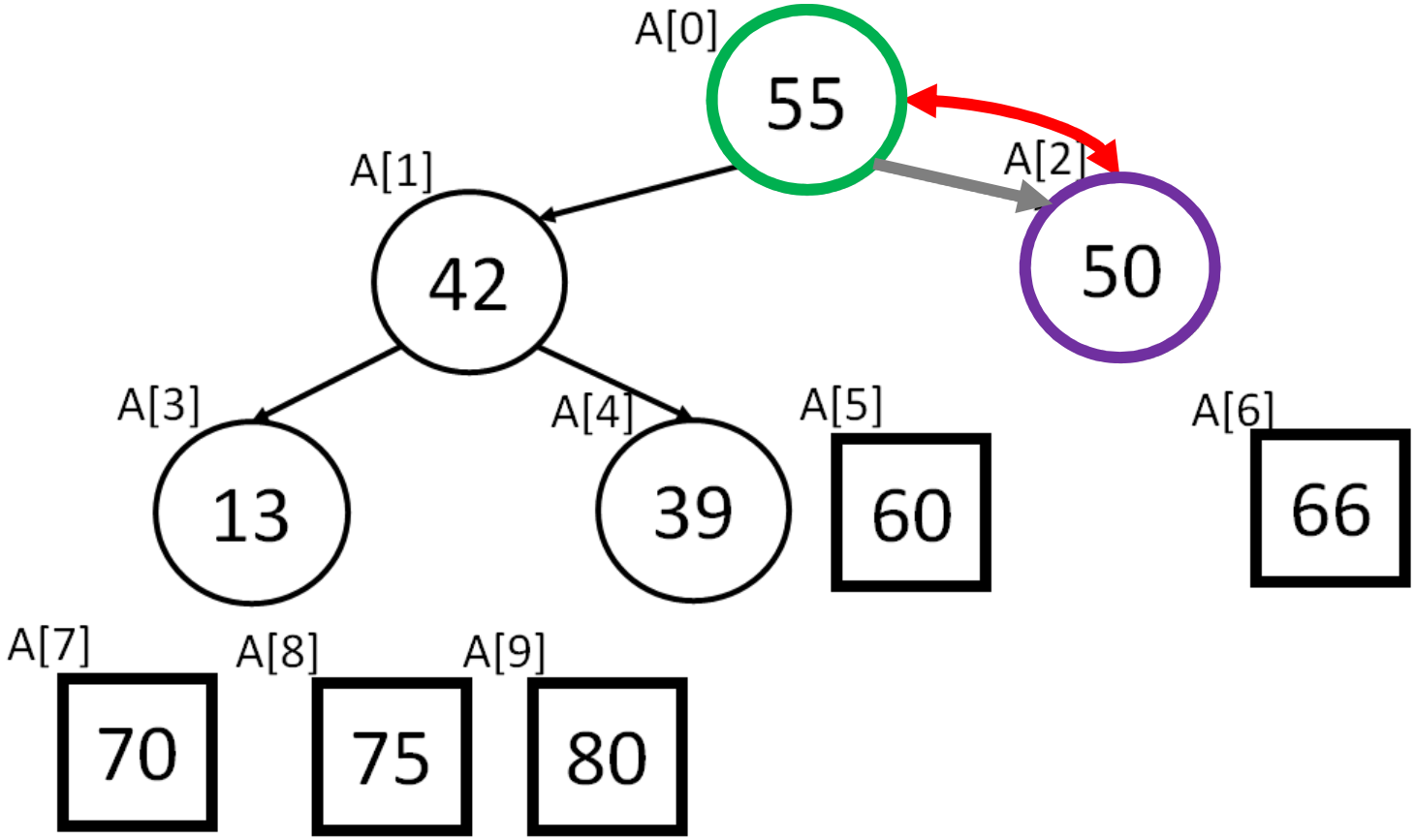
Construindo a Heap: método Heapifica()

i	0	1	2	3	4	5	6	7	8	9
A[i]	50	42	55	13	39	60	66	70	75	80



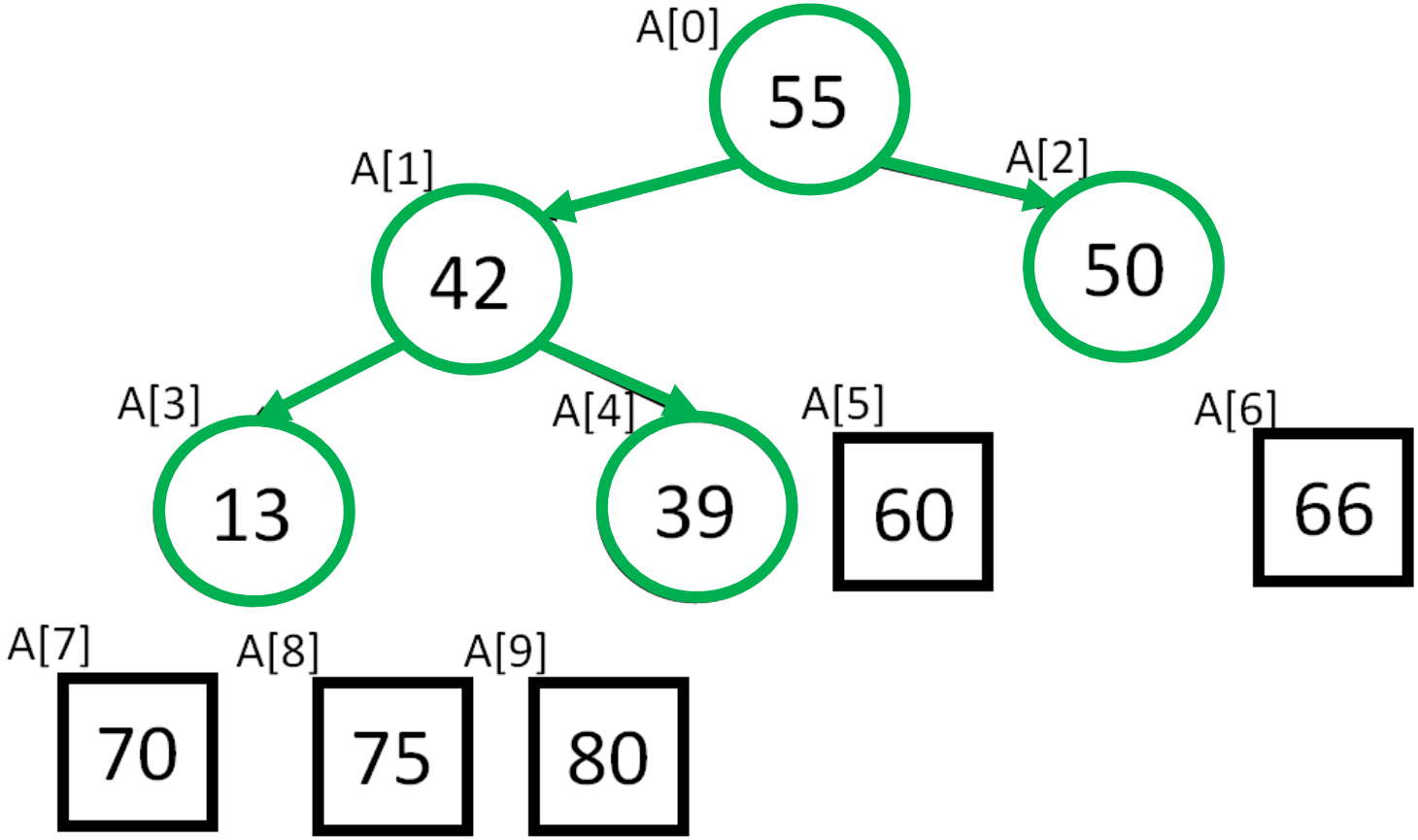
Construindo a Heap: método Heapifica()

i	0	1	2	3	4	5	6	7	8	9
A[i]	55	42	50	13	39	60	66	70	75	80



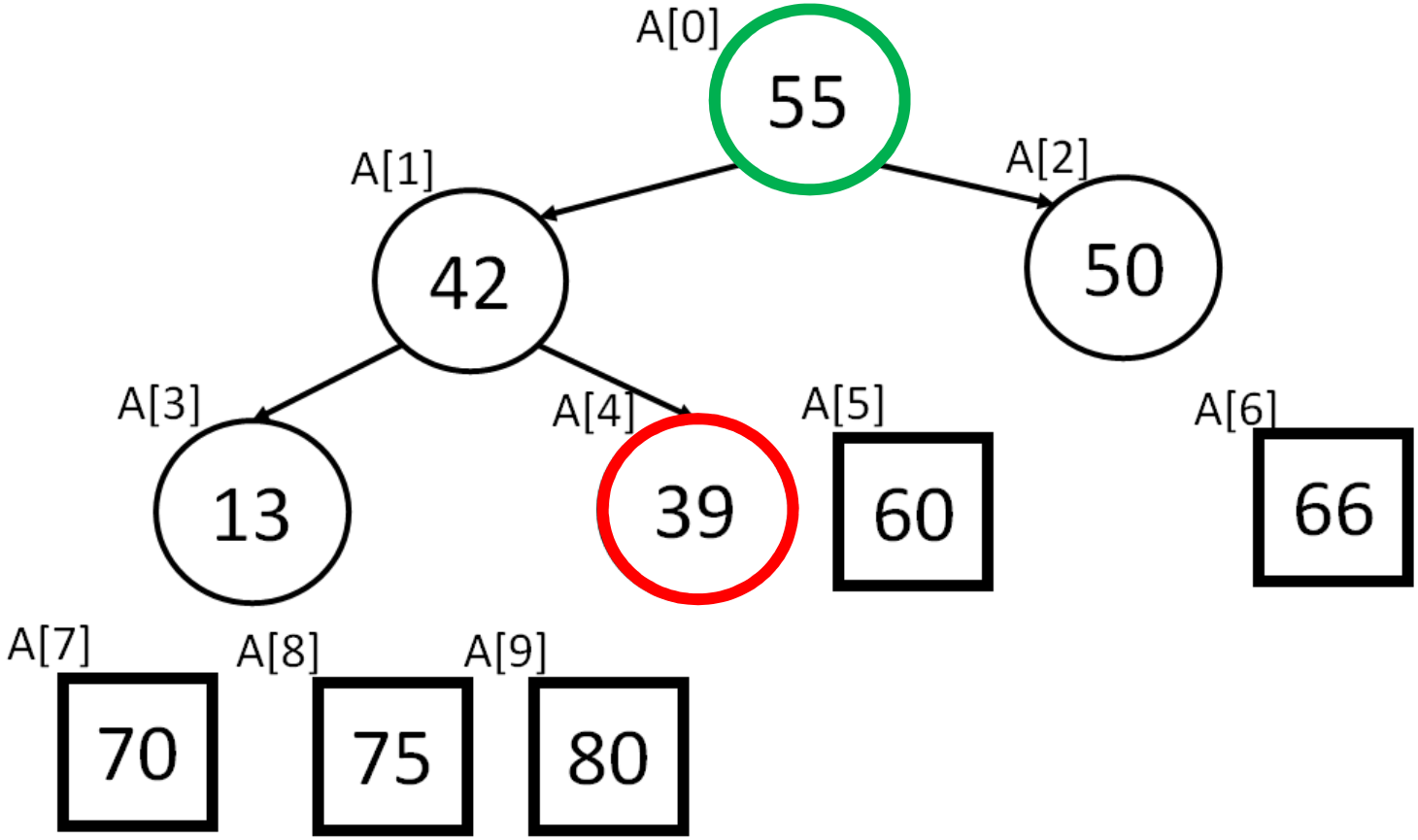
Construindo a Heap: método Heapifica()

i	0	1	2	3	4	5	6	7	8	9
A[i]	55	42	50	13	39	60	66	70	75	80



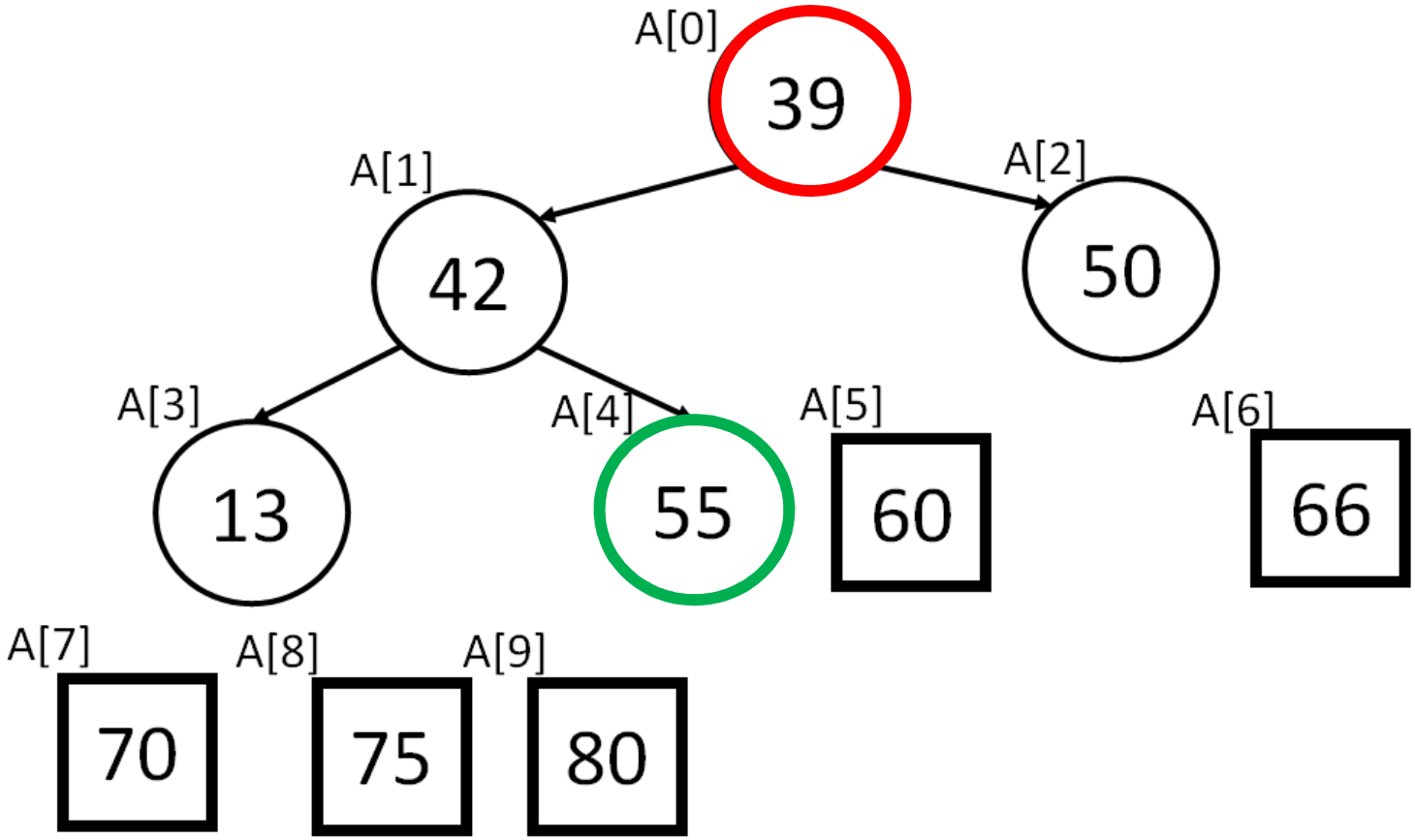
Construindo a Heap: método Heapsort()

i	0	1	2	3	4	5	6	7	8	9
A[i]	55	42	50	13	39	60	66	70	75	80



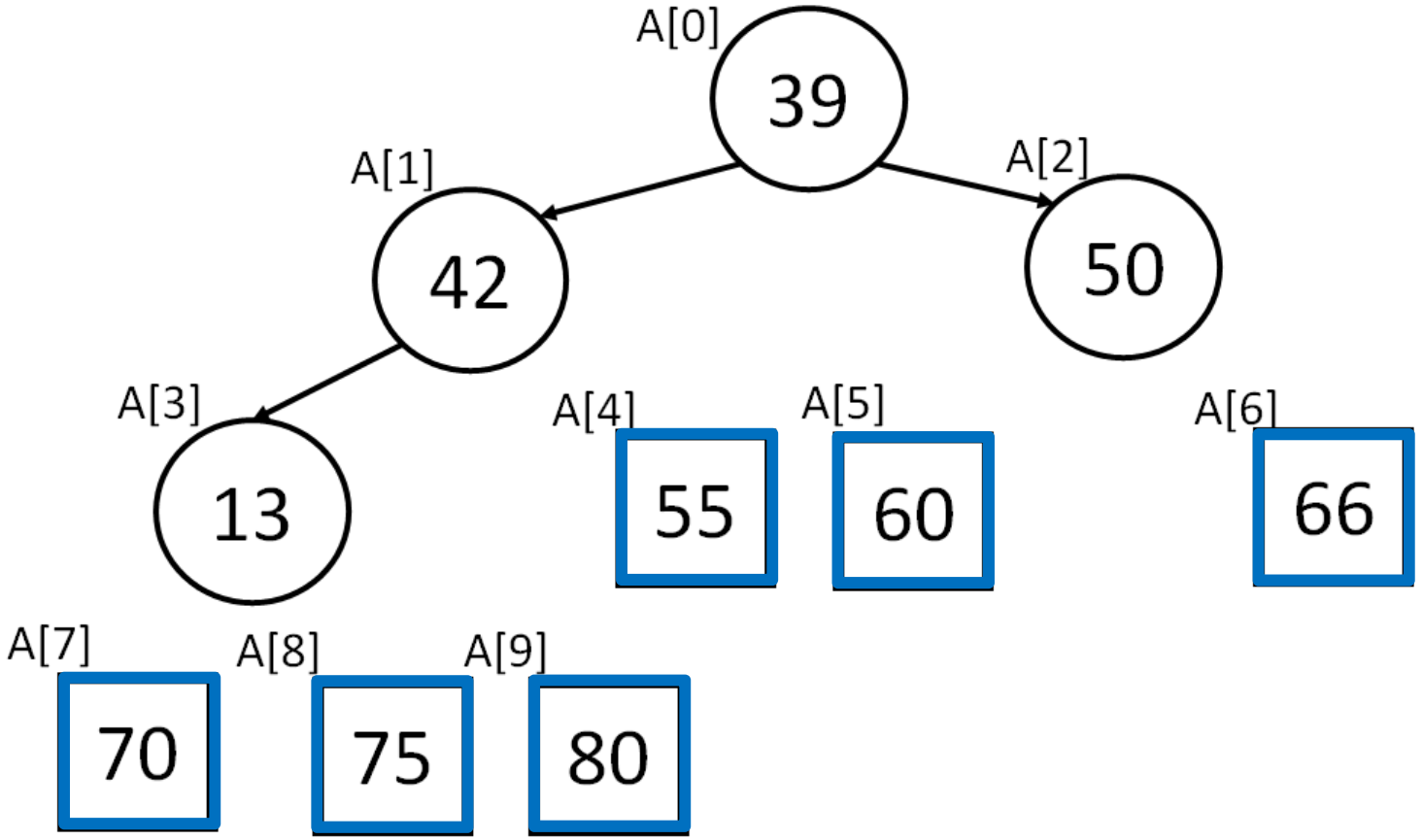
Construindo a Heap: método Heapsort()

i	0	1	2	3	4	5	6	7	8	9
A[i]	39	42	50	13	55	60	66	70	75	80



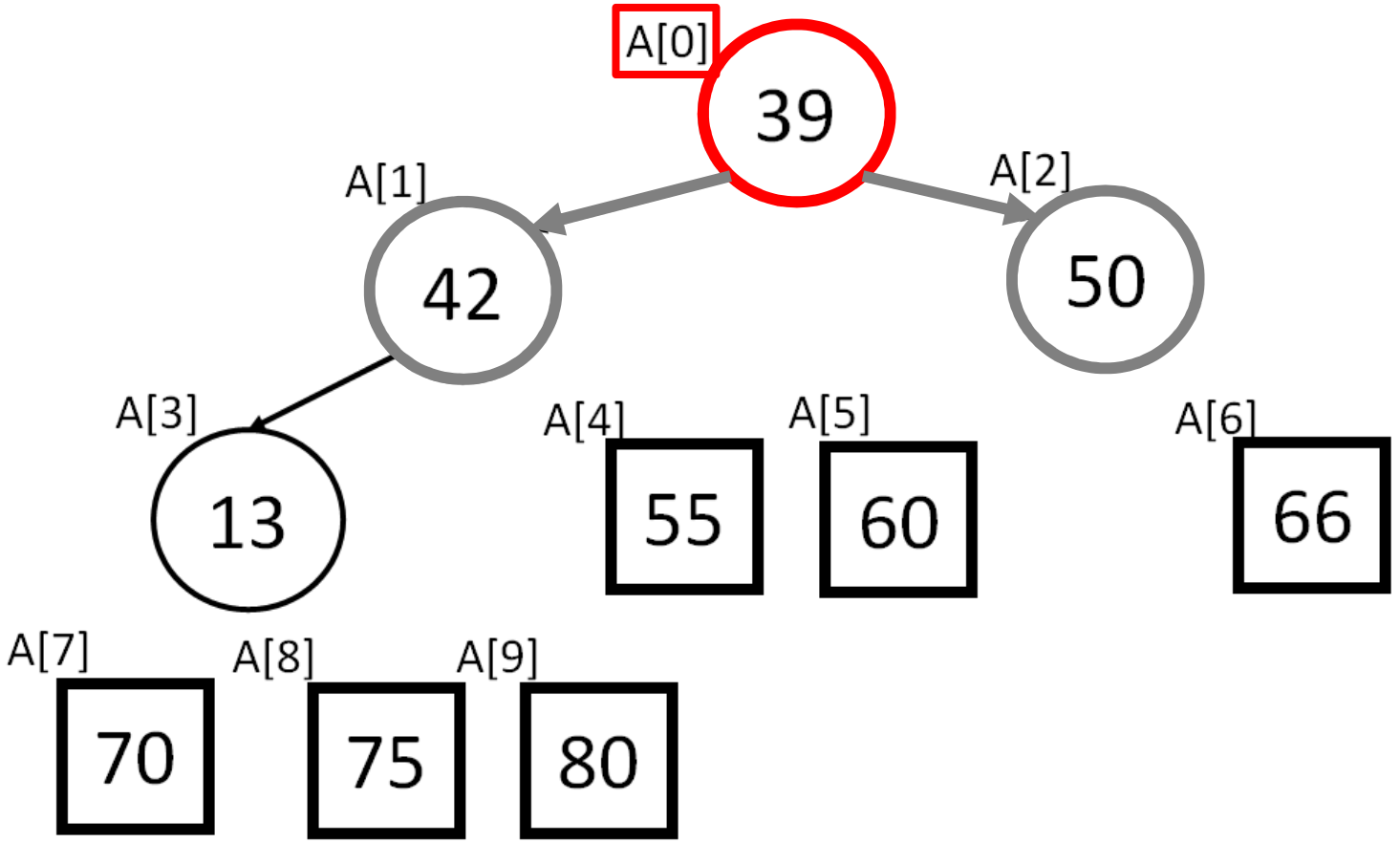
Construindo a Heap: método Heapsort()

i	0	1	2	3	4	5	6	7	8	9
A[i]	39	42	50	13	55	60	66	70	75	80



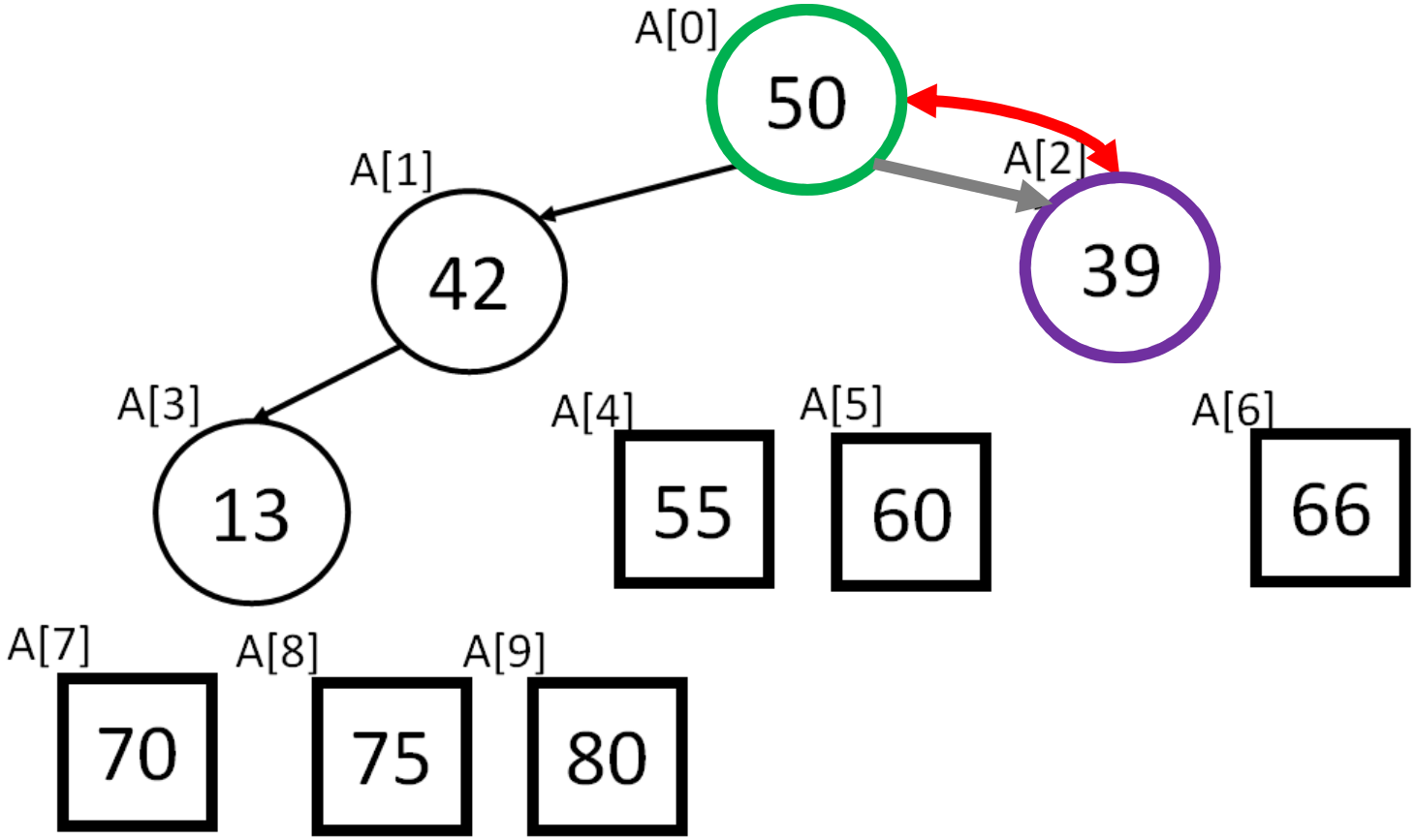
Construindo a Heap: método Heapifica()

i	0	1	2	3	4	5	6	7	8	9
A[i]	39	42	50	13	55	60	66	70	75	80



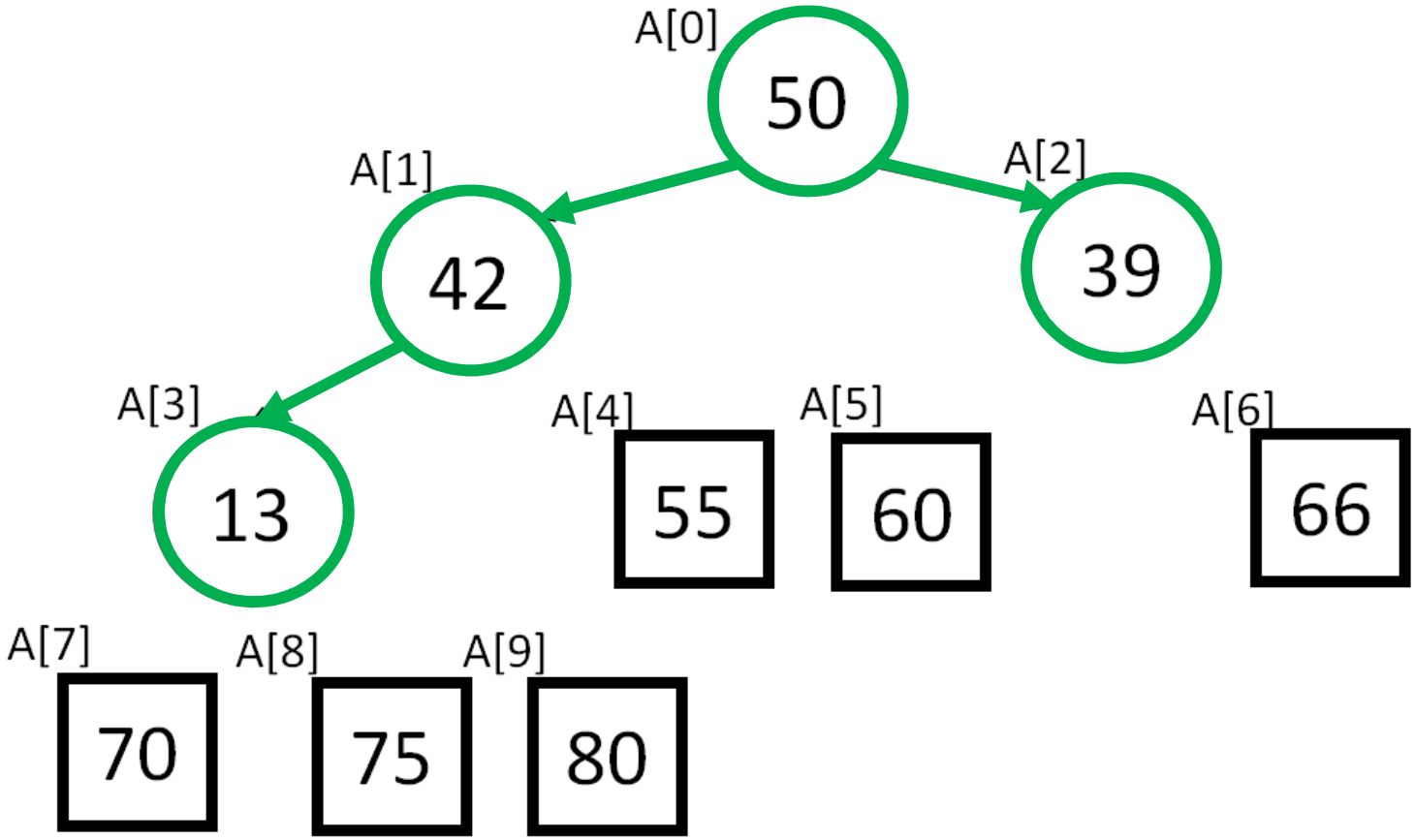
Construindo a Heap: método Heapifica()

i	0	1	2	3	4	5	6	7	8	9
A[i]	50	42	39	13	55	60	66	70	75	80



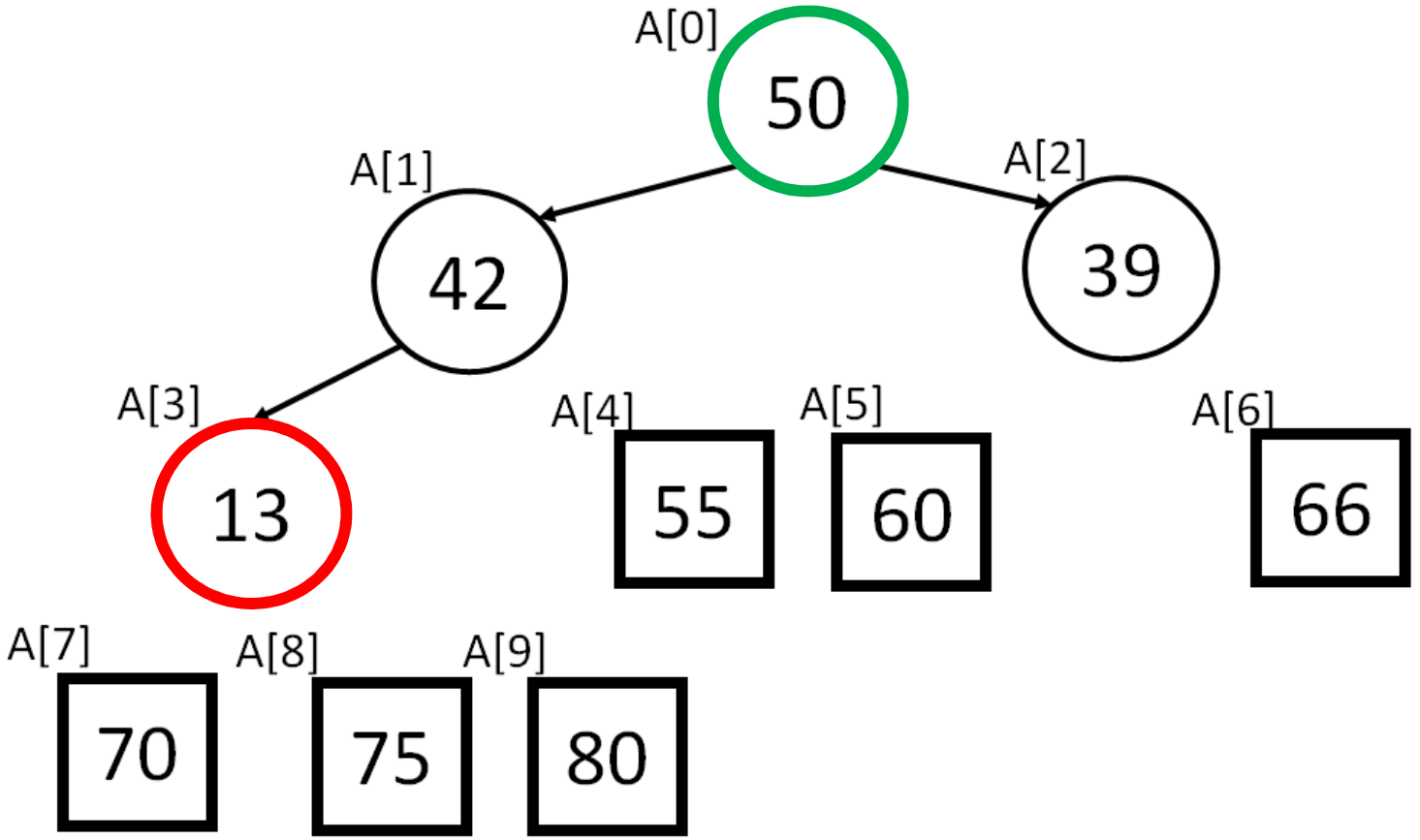
Construindo a Heap: método Heapifica()

i	0	1	2	3	4	5	6	7	8	9
A[i]	50	42	39	13	55	60	66	70	75	80



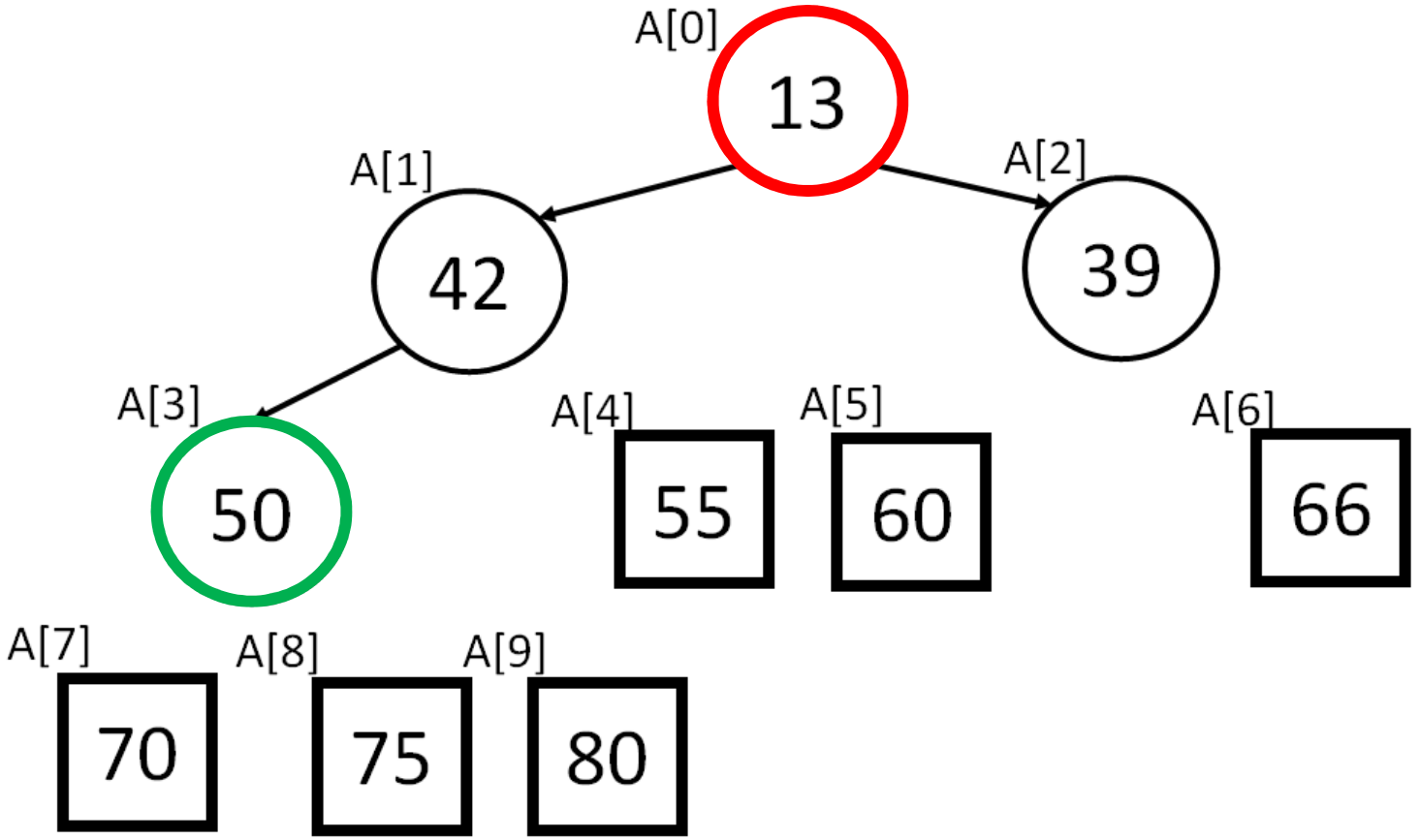
Construindo a Heap: método Heapsort()

i	0	1	2	3	4	5	6	7	8	9
A[i]	50	42	39	13	55	60	66	70	75	80



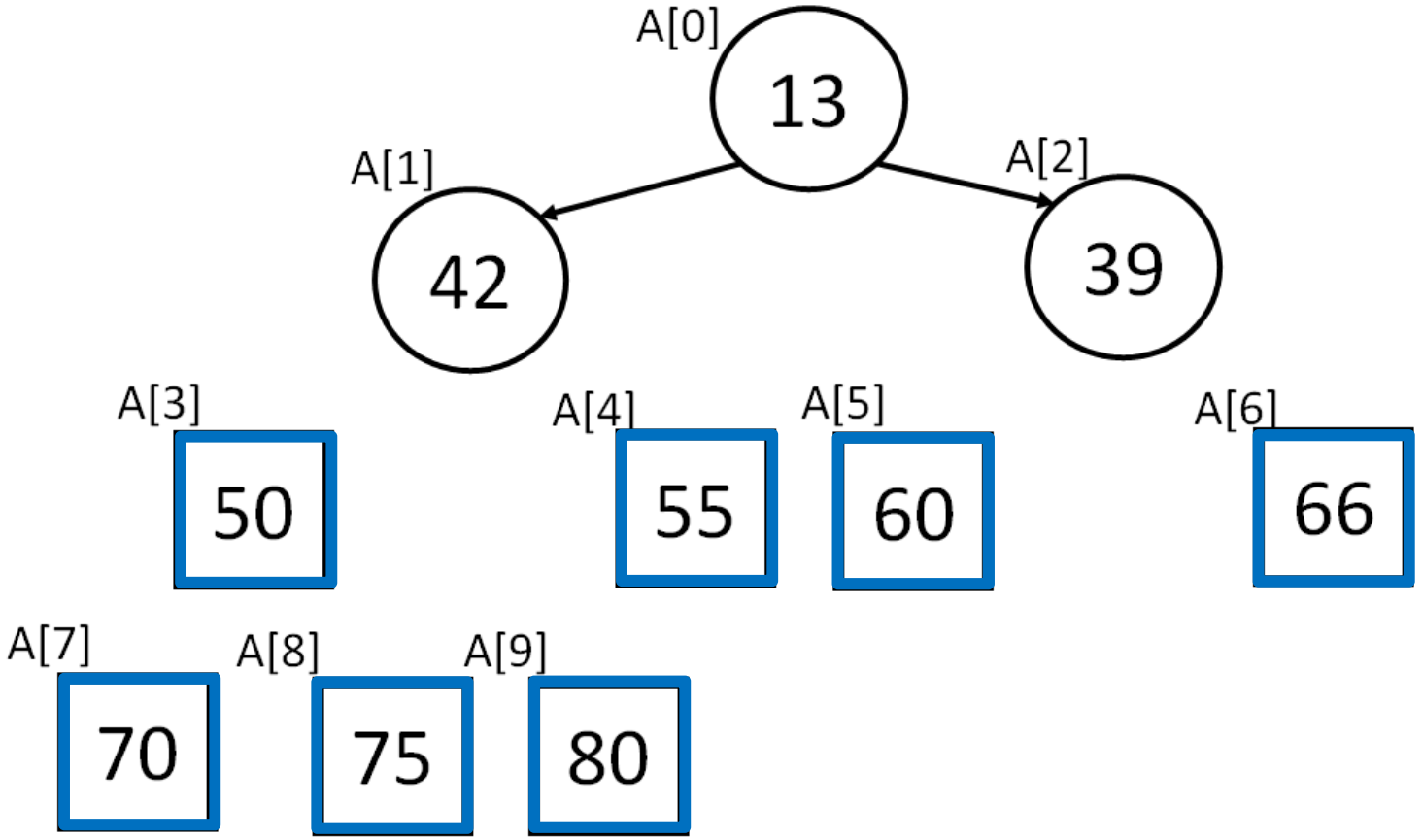
Construindo a Heap: método Heapsort()

i	0	1	2	3	4	5	6	7	8	9
A[i]	13	42	39	50	55	60	66	70	75	80



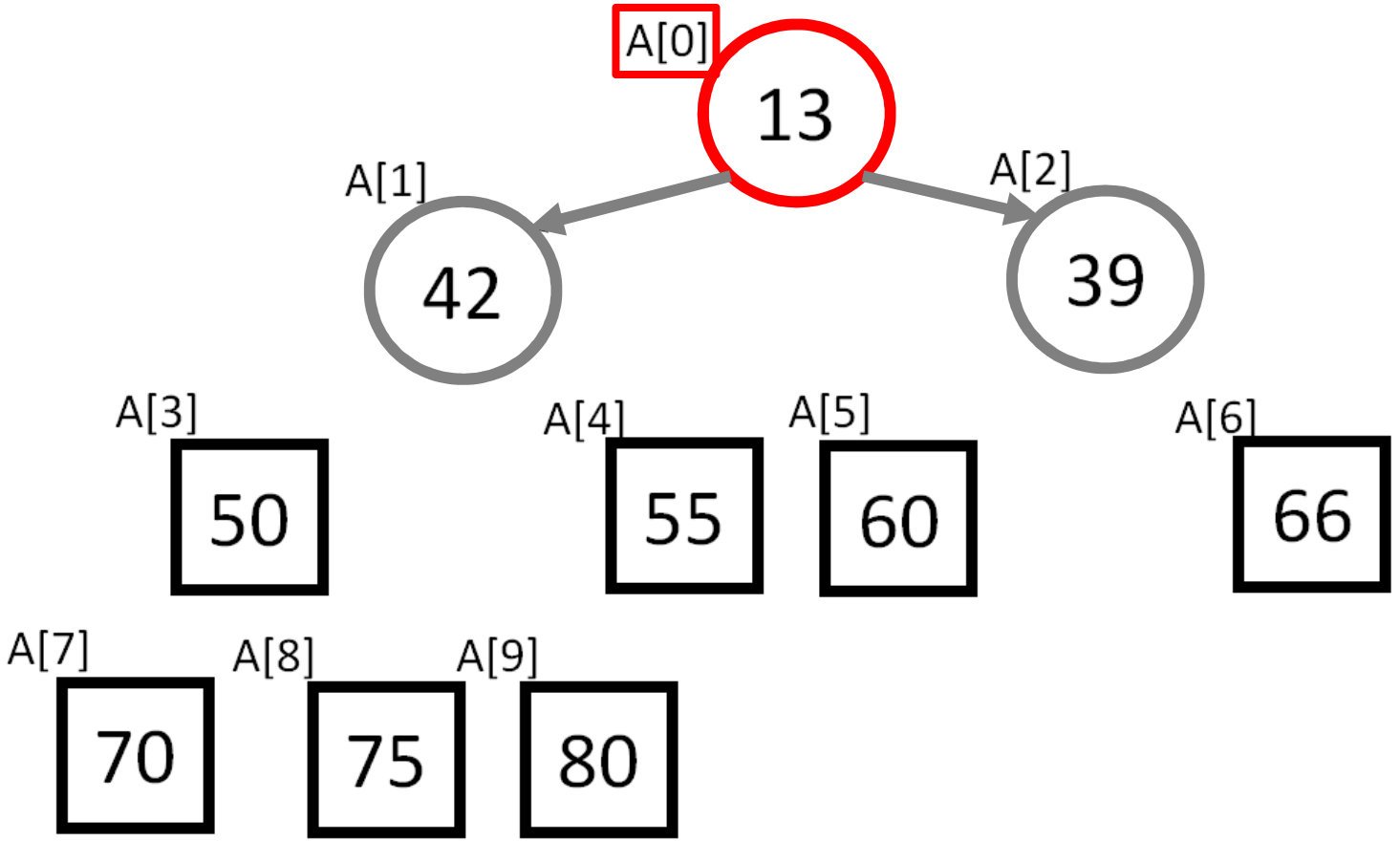
Construindo a Heap: método Heapsort()

i	0	1	2	3	4	5	6	7	8	9
A[i]	13	42	39	50	55	60	66	70	75	80



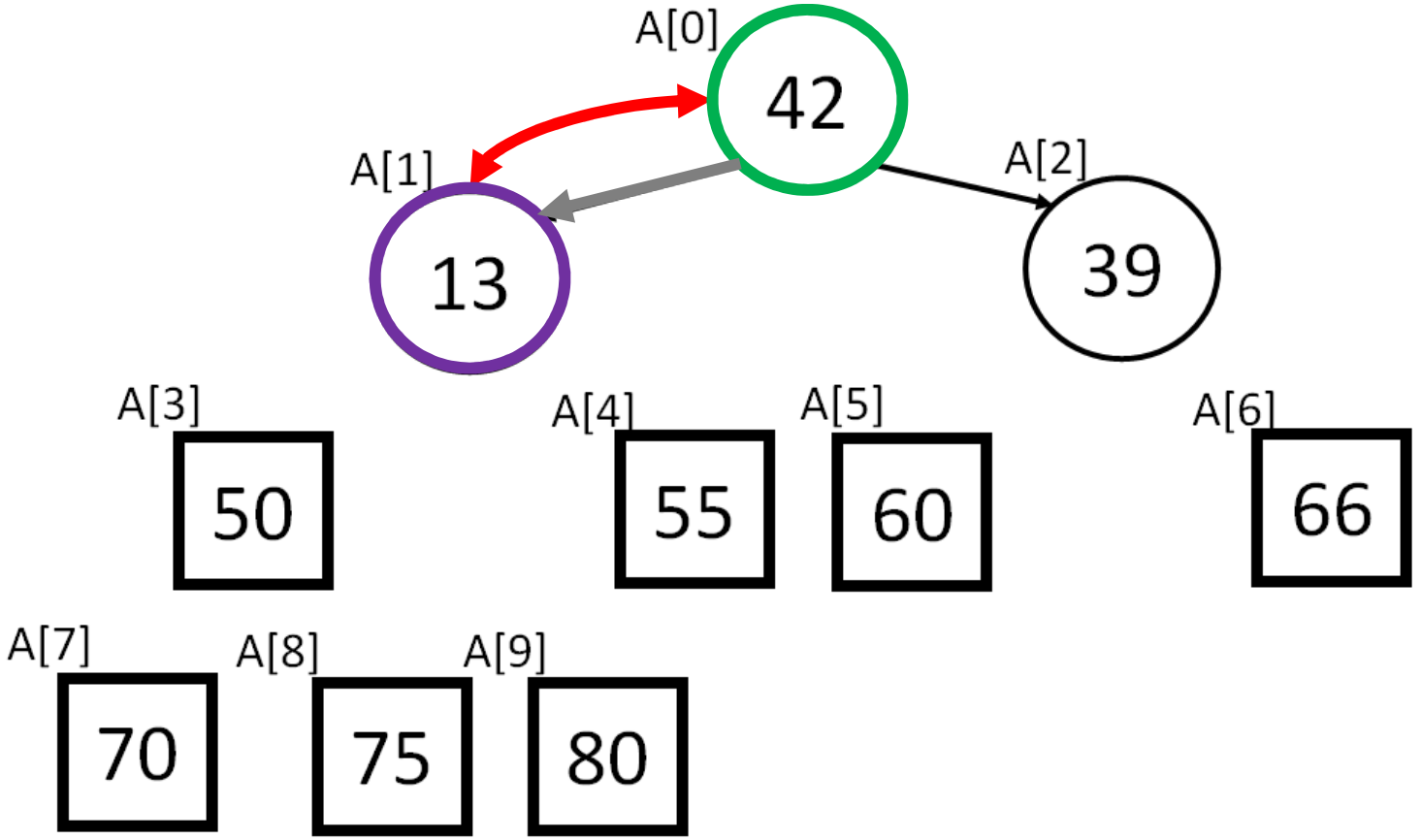
Construindo a Heap: método Heapifica()

i	0	1	2	3	4	5	6	7	8	9
A[i]	13	42	39	50	55	60	66	70	75	80



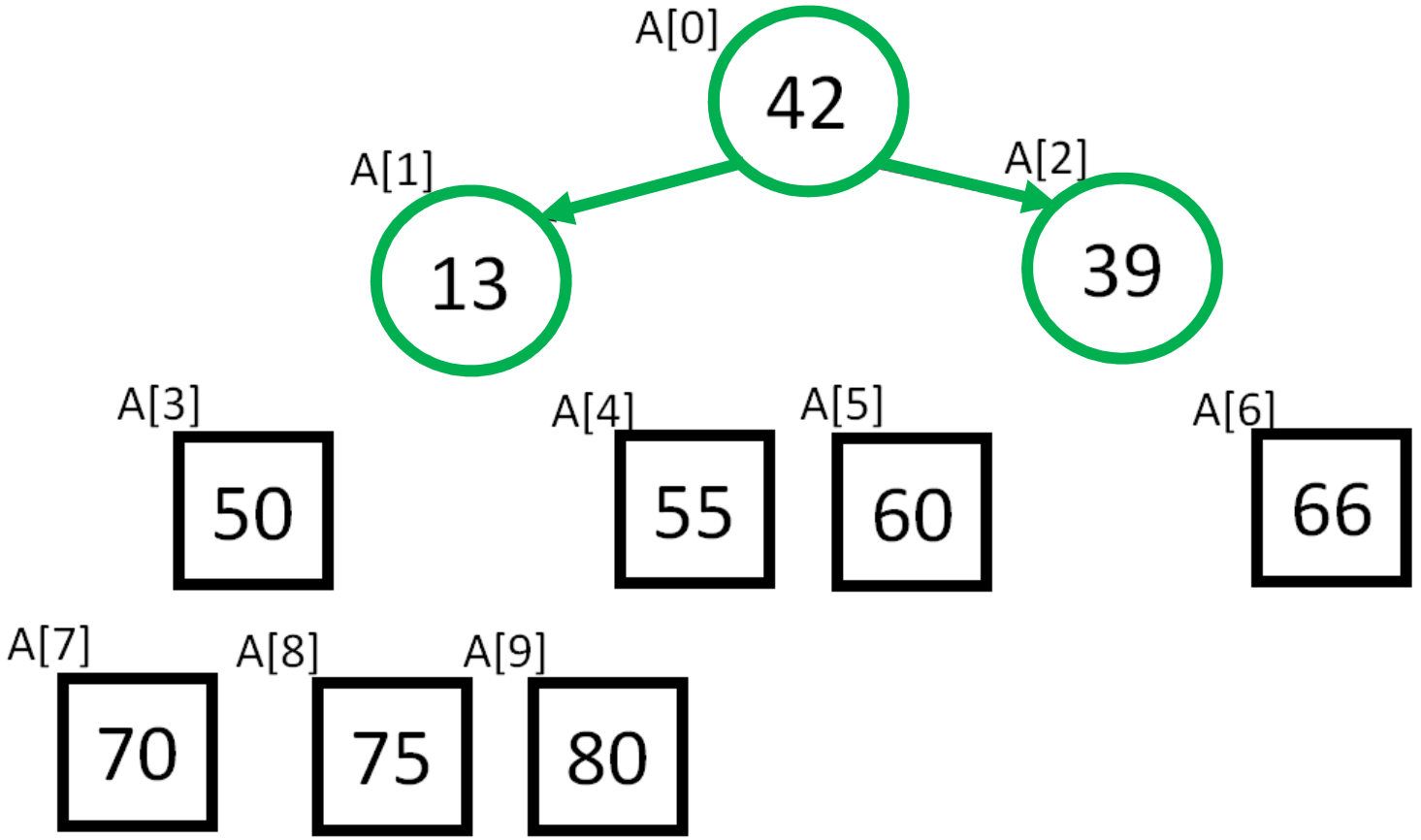
Construindo a Heap: método Heapifica()

i	0	1	2	3	4	5	6	7	8	9
A[i]	42	13	39	50	55	60	66	70	75	80



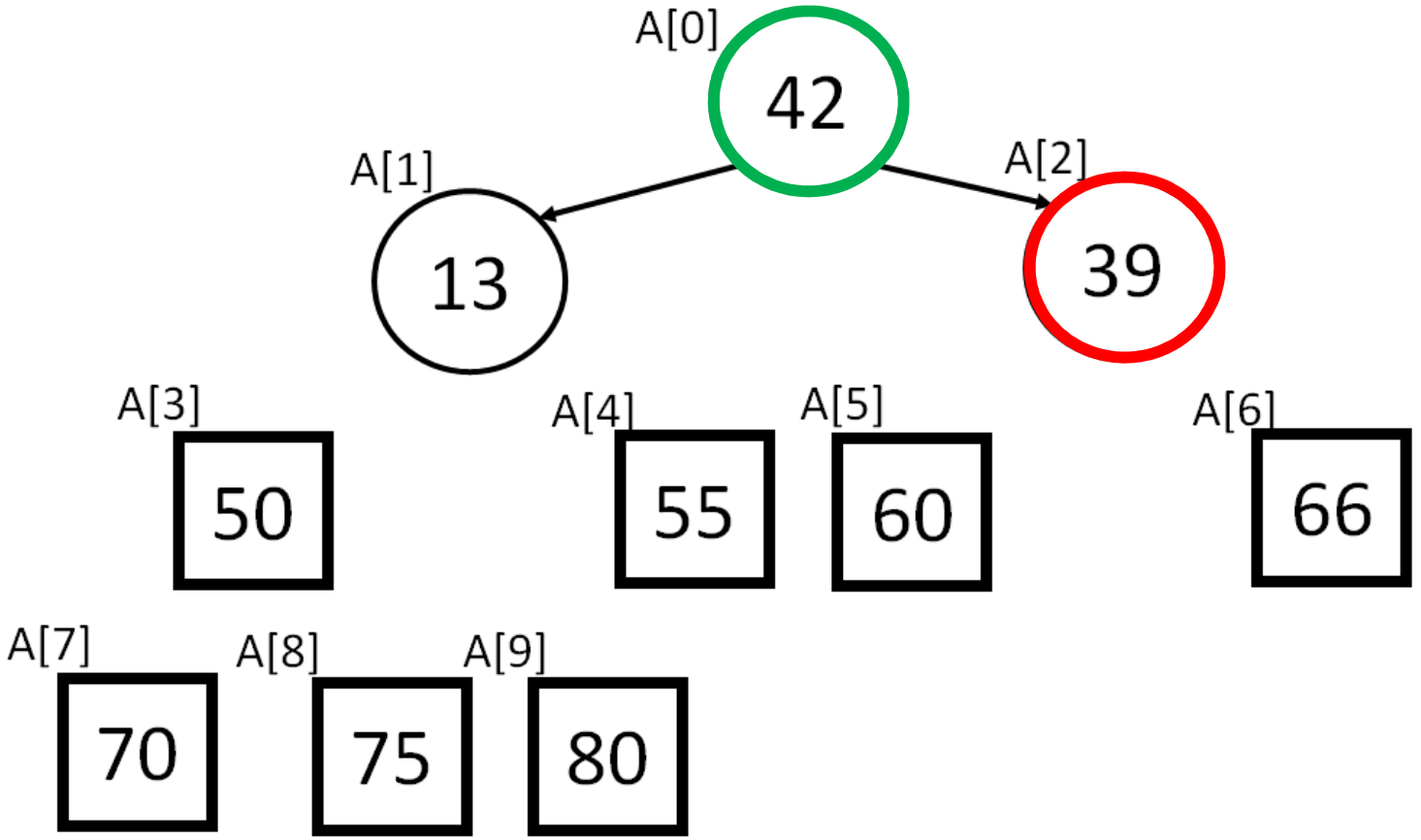
Construindo a Heap: método Heapifica()

i	0	1	2	3	4	5	6	7	8	9
A[i]	42	13	39	50	55	60	66	70	75	80



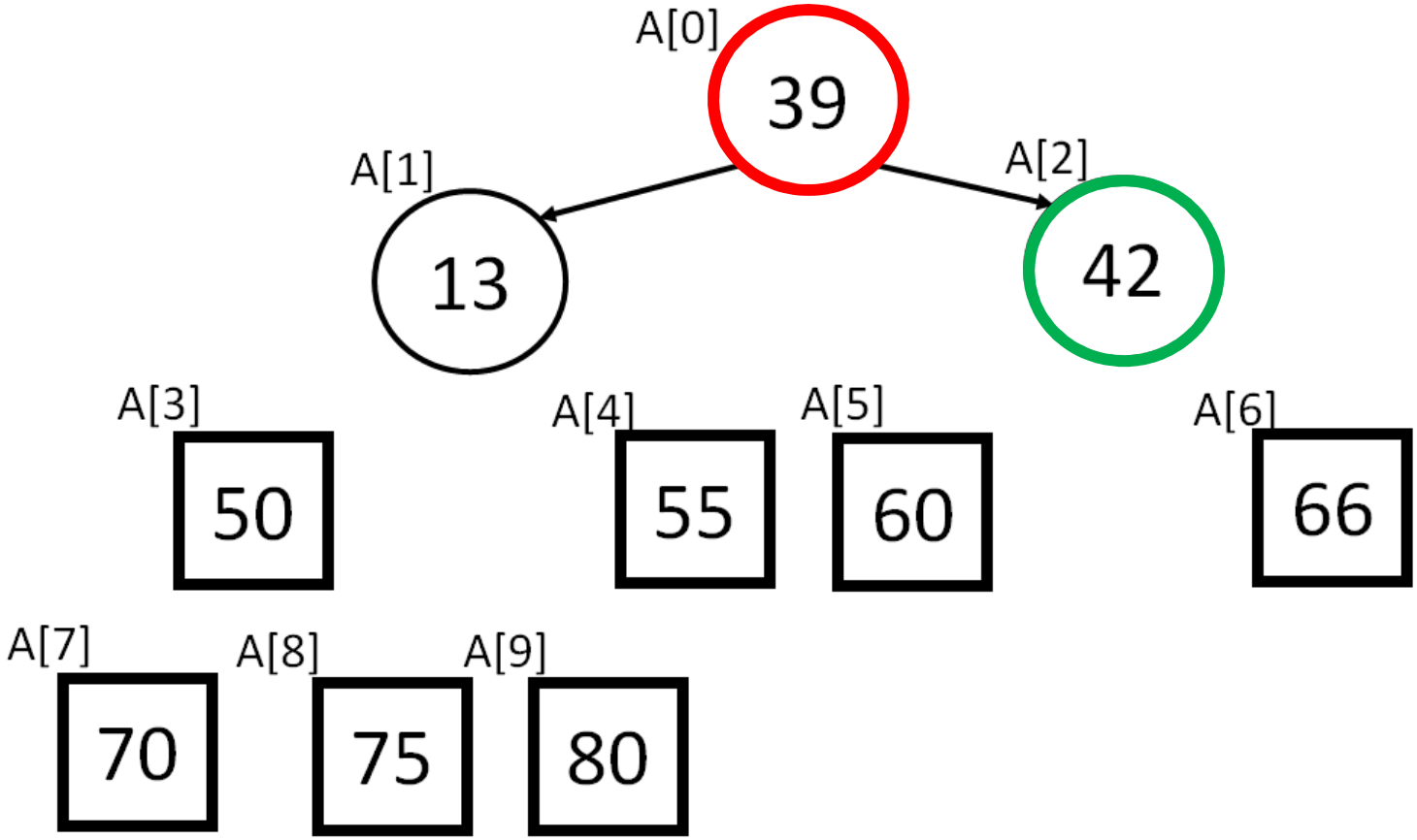
Construindo a Heap: método Heapsort()

i	0	1	2	3	4	5	6	7	8	9
A[i]	42	13	39	50	55	60	66	70	75	80



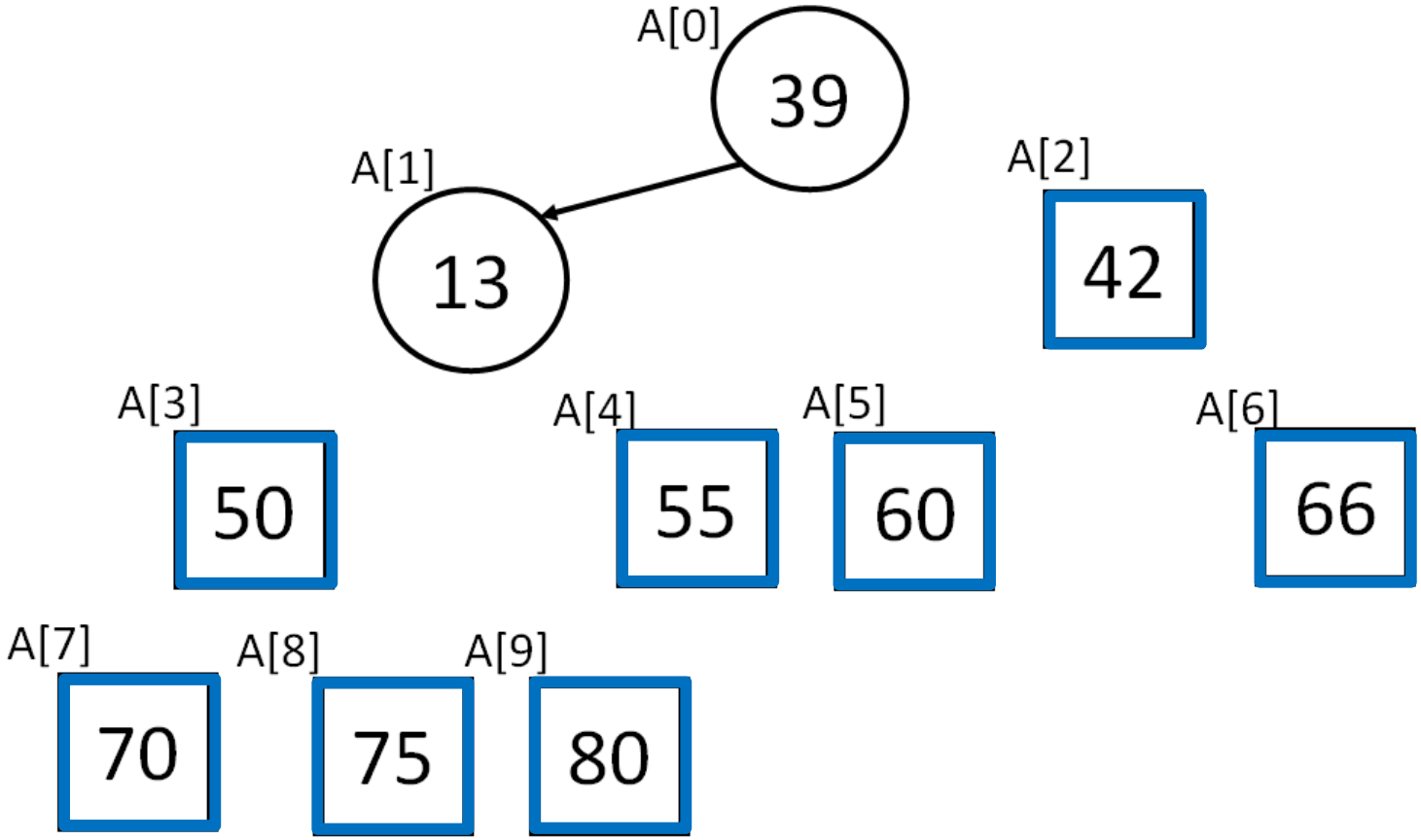
Construindo a Heap: método Heapsort()

i	0	1	2	3	4	5	6	7	8	9
A[i]	39	13	42	50	55	60	66	70	75	80



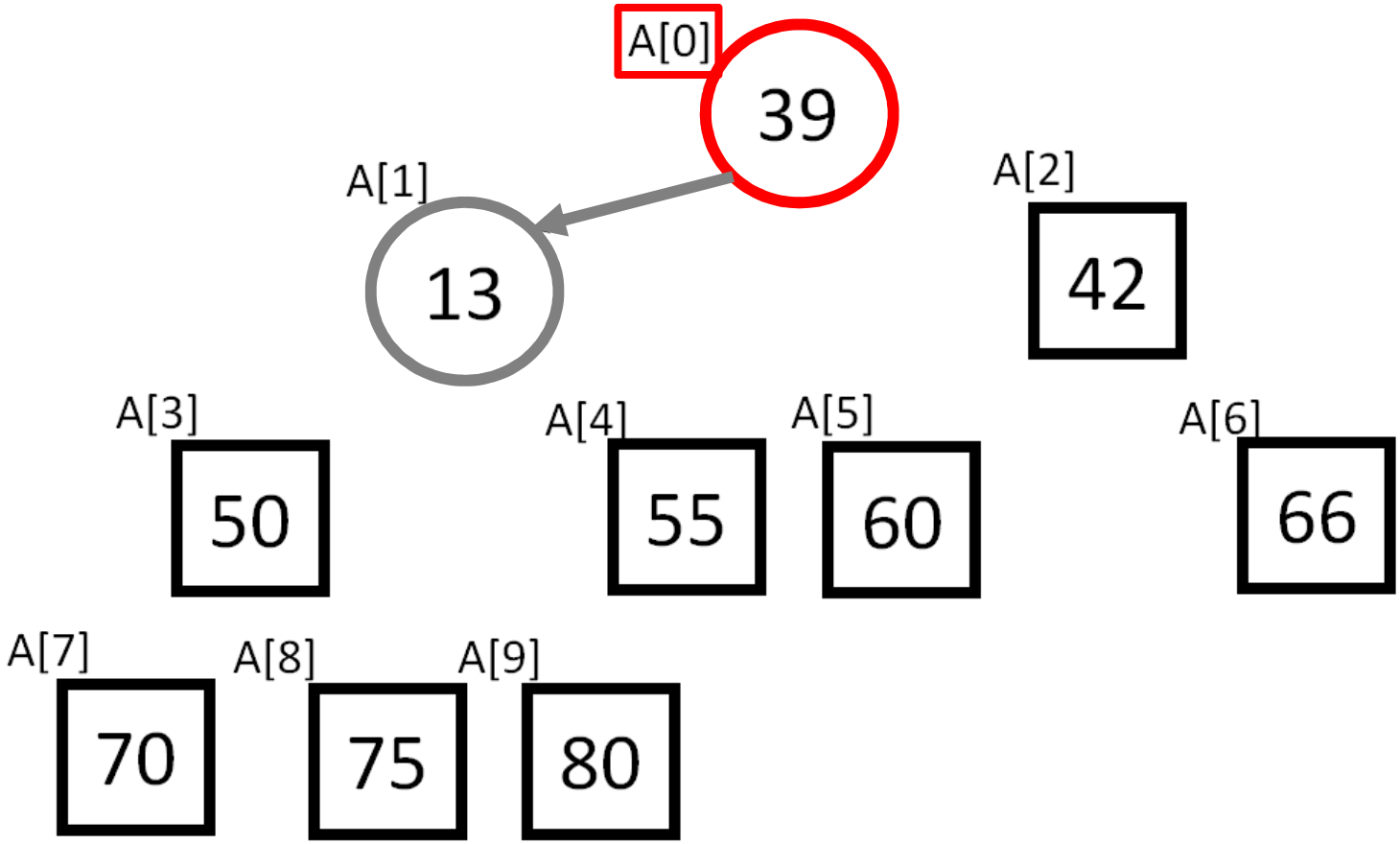
Construindo a Heap: método Heapsort()

i	0	1	2	3	4	5	6	7	8	9
A[i]	39	13	42	50	55	60	66	70	75	80



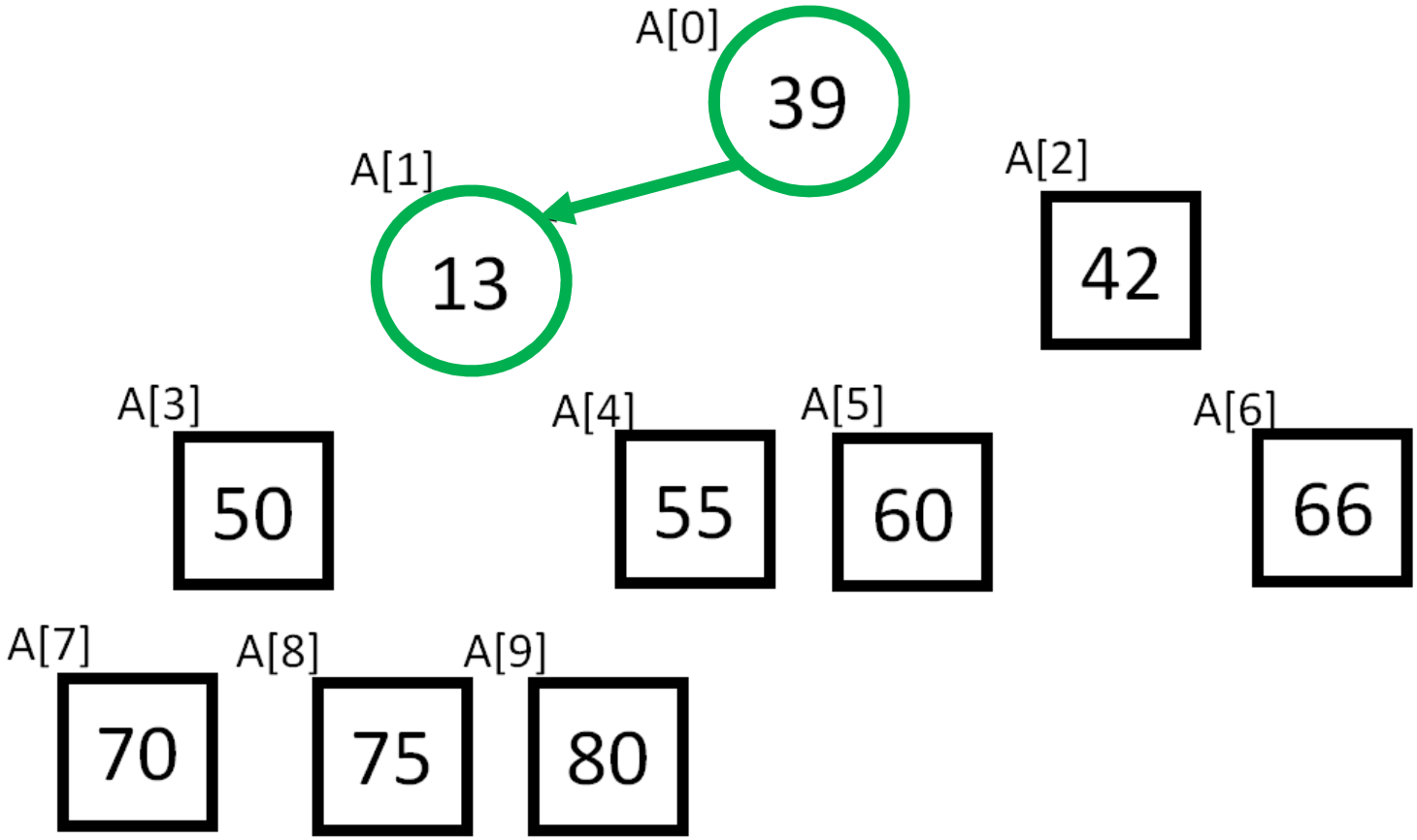
Construindo a Heap: método Heapifica()

i	0	1	2	3	4	5	6	7	8	9
A[i]	39	13	42	50	55	60	66	70	75	80



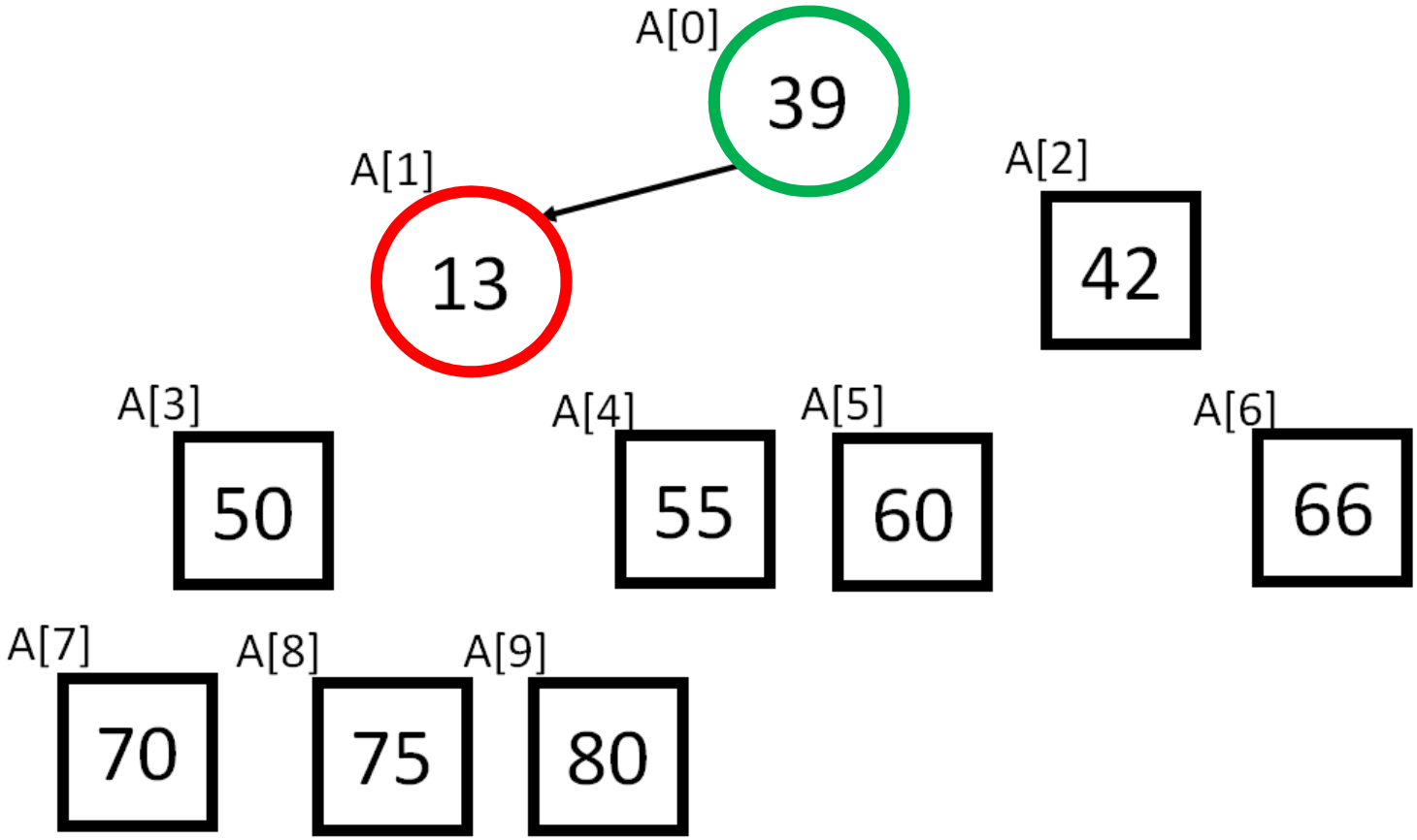
Construindo a Heap: método Heapifica()

i	0	1	2	3	4	5	6	7	8	9
A[i]	39	13	42	50	55	60	66	70	75	80



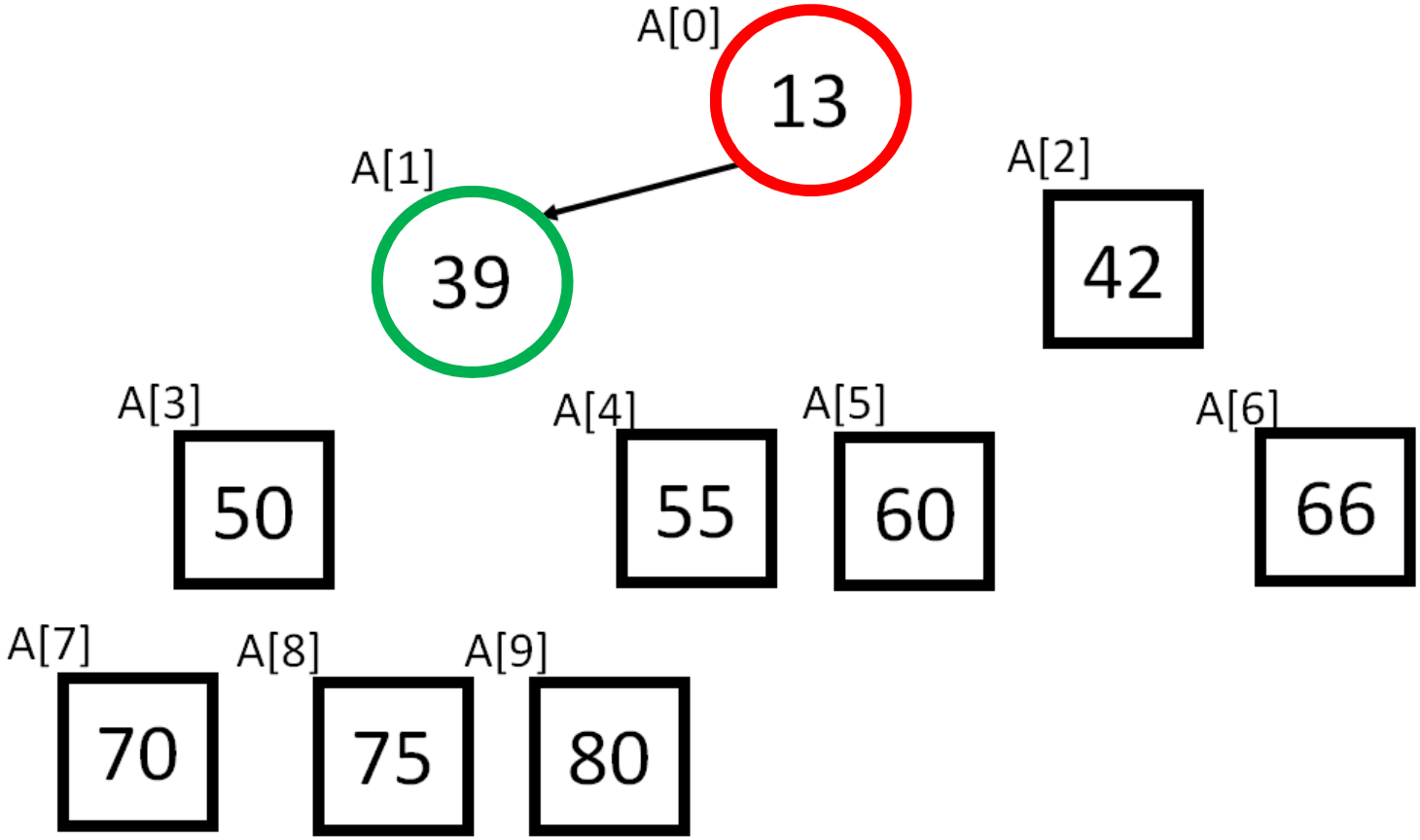
Construindo a Heap: método Heapsort()

i	0	1	2	3	4	5	6	7	8	9
A[i]	39	13	42	50	55	60	66	70	75	80



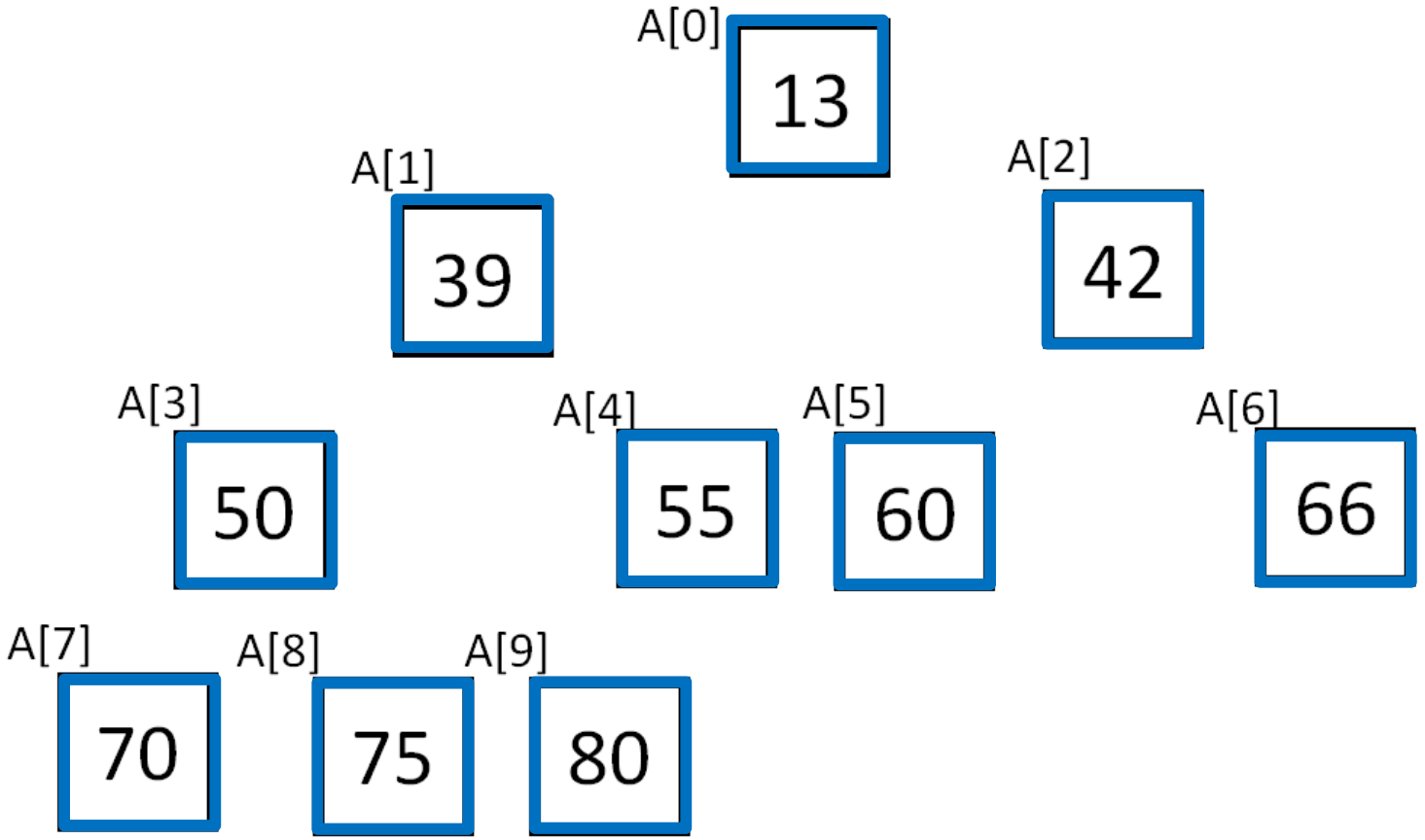
Construindo a Heap: método Heapsort()

i	0	1	2	3	4	5	6	7	8	9
A[i]	13	39	42	50	55	60	66	70	75	80



Construindo a Heap: método Heapsort()

i	0	1	2	3	4	5	6	7	8	9
A[i]	13	39	42	50	55	60	66	70	75	80



UniCesumar

Estrutura de Dados II

Prof. Me. Pietro M. de Oliveira