"Nada na vida deve ser temido, apenas compreendido" (Marie Curie).

Ordenação Externa

Paulo Ricardo Lisboa de Almeida

DSBD

UFPR

O Problema

Considere que temos um arquivo com 2TB de dados.

Precisamos ordenar.

Problema?

O Problema

Considere que temos um arquivo com 2TB de dados.

Precisamos ordenar.

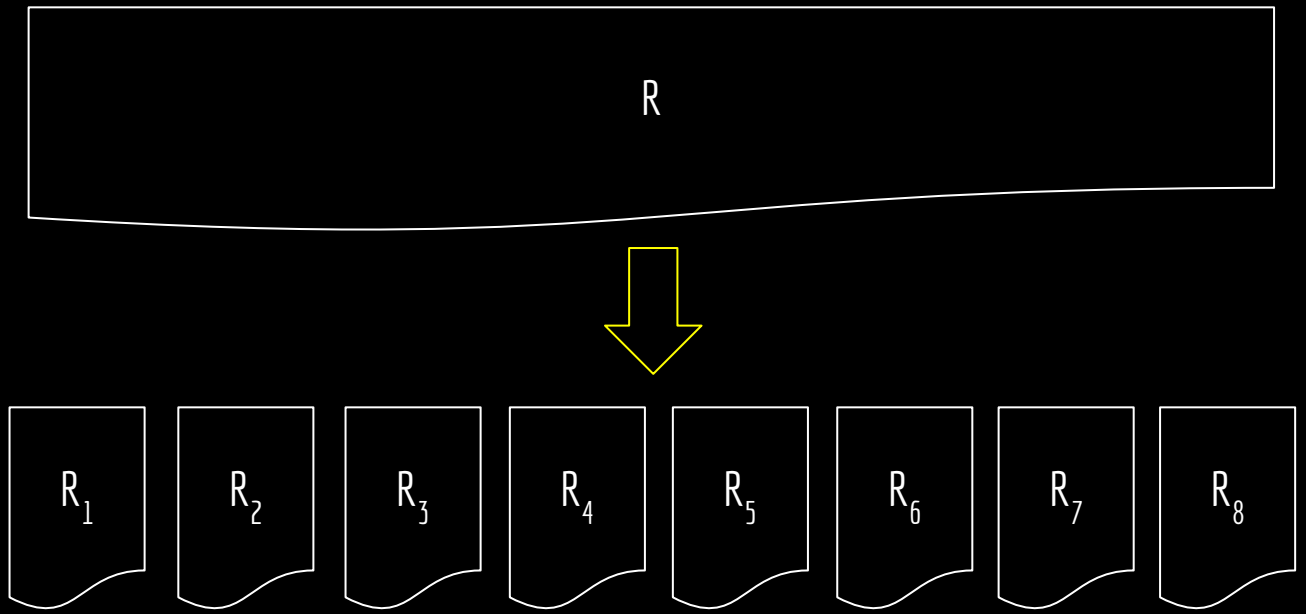
Provavelmente não teremos memória principal o suficiente para ordenar.

Resolvendo

Como resolver?

P-way merging

Particionar o arquivo *R* em *p* partes aproximadamente iguais R_1, R_2, ..., R_p, onde cada R_x cabe na memória principal.

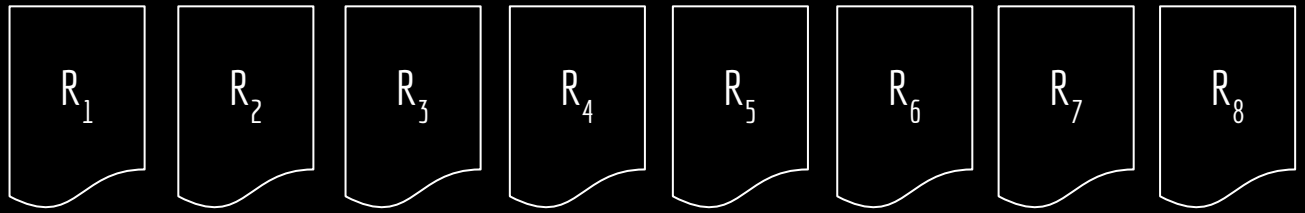


Exemplo com P=8

P-way merging

Particionar o arquivo *R* em *p* partes aproximadamente iguais R_1, R_2, ..., R_p, onde cada R_x cabe na memória principal.

Carregar cada R_x para a memória principal, ordenar, e armazenar novamente na memória secundária.



P-way merging

Particionar o arquivo *R* em *p* partes aproximadamente iguais R_1, R_2, ..., R_p, onde cada R_x cabe na memória principal.

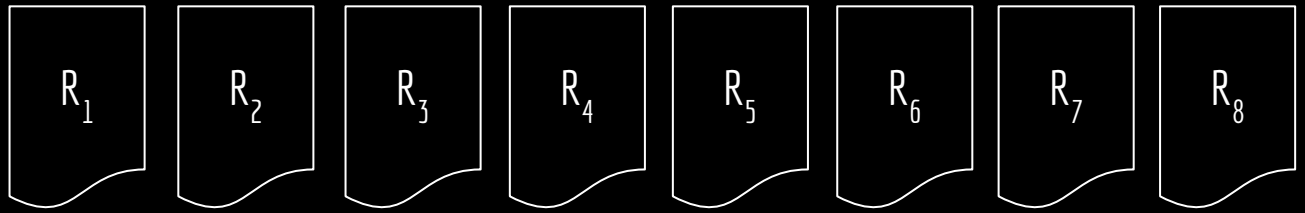
Carregar cada R_x para a memória principal, ordenar, e armazenar novamente na memória secundária.

Abrir ponteiros para os *p* arquivos R_1, ..., R_p.

Abrir um ponteiro *S* para um arquivo que receberá os dados ordenados.

Olhar para a cabeça dos arquivos R_1, ..., R_p, e escrever em S o dado do arquivo com o menor valor.

Incrementar o ponteiro do arquivo, e repetir o processo até esgotar todos os arquivos.



Exemplo

70	58	62	5	53	76	10	22	31	39	34	73	71	21	55	0	60	11	41	27	96	90	86	97	63	12	8	84	3	2	85	77

Arquivo grande.

Exemplo

70	58	62	5	53	76	10	22	31	39	34	73	71	21	55	0	60	11	41	27	96	90	86	97	63	12	8	84	3	2	85	77

Dividir em arquivos que podemos tratar na memória principal.

70
58
62
5

53
76
10
22

31
39
34
73

71
21
55
0

60
11
41
27

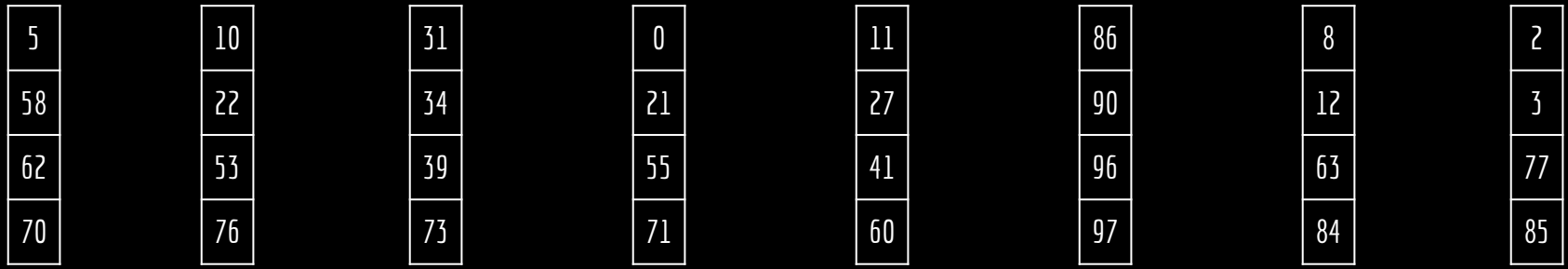
96
90
86
97

63
12
8
84

3
2
85
77

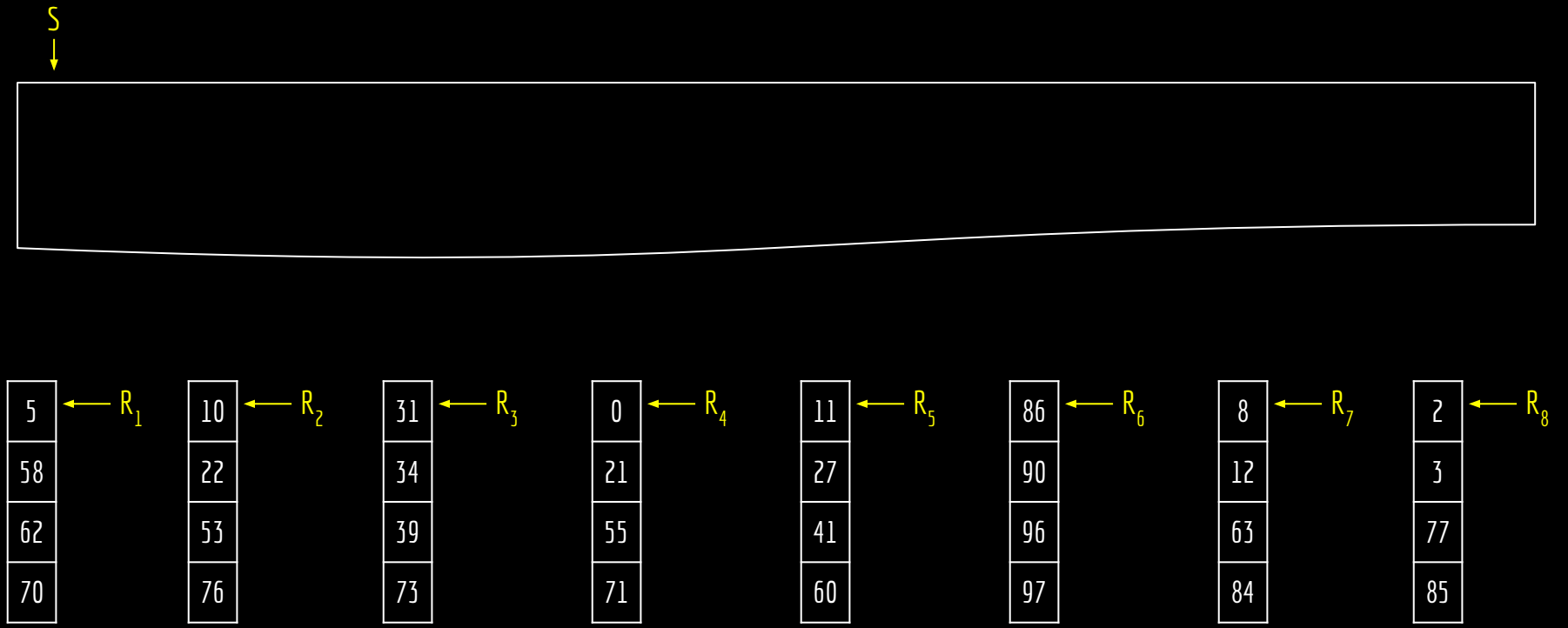
Exemplo

Ordenar cada arquivo utilizando algum algoritmo de ordenação (ex.: Merge Sort).



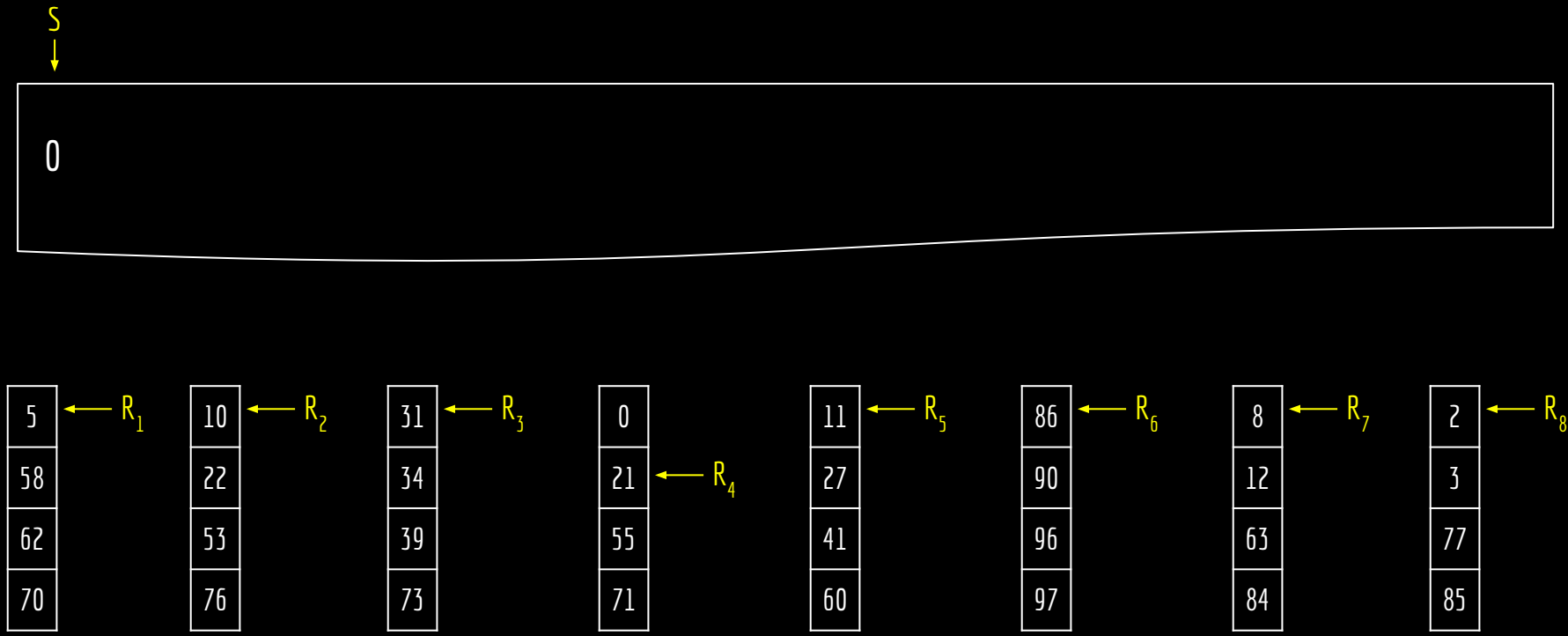
Exemplo

Abrir um ponteiro para cada arquivo, e um ponteiro para o arquivo ordenado.



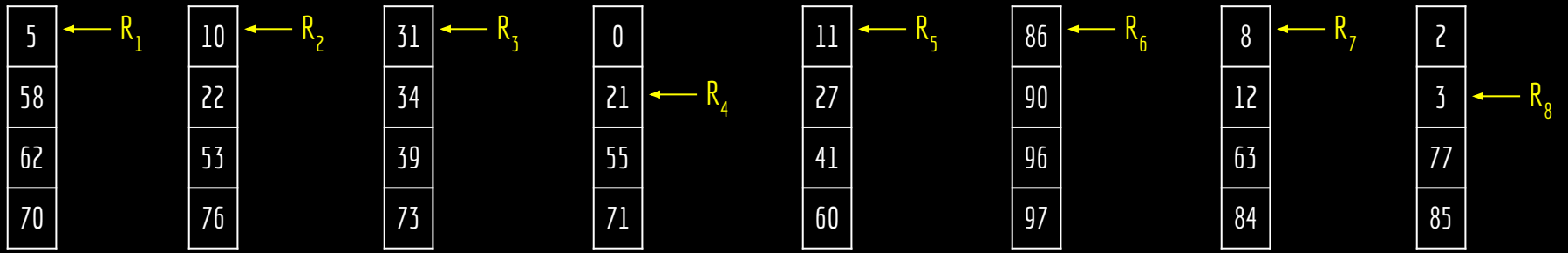
Exemplo

Verificar qual dos ponteiros escrever a seguir no arquivo.
Utilizar um Selection Sort, tomando *P-1* comparações a cada iteração.



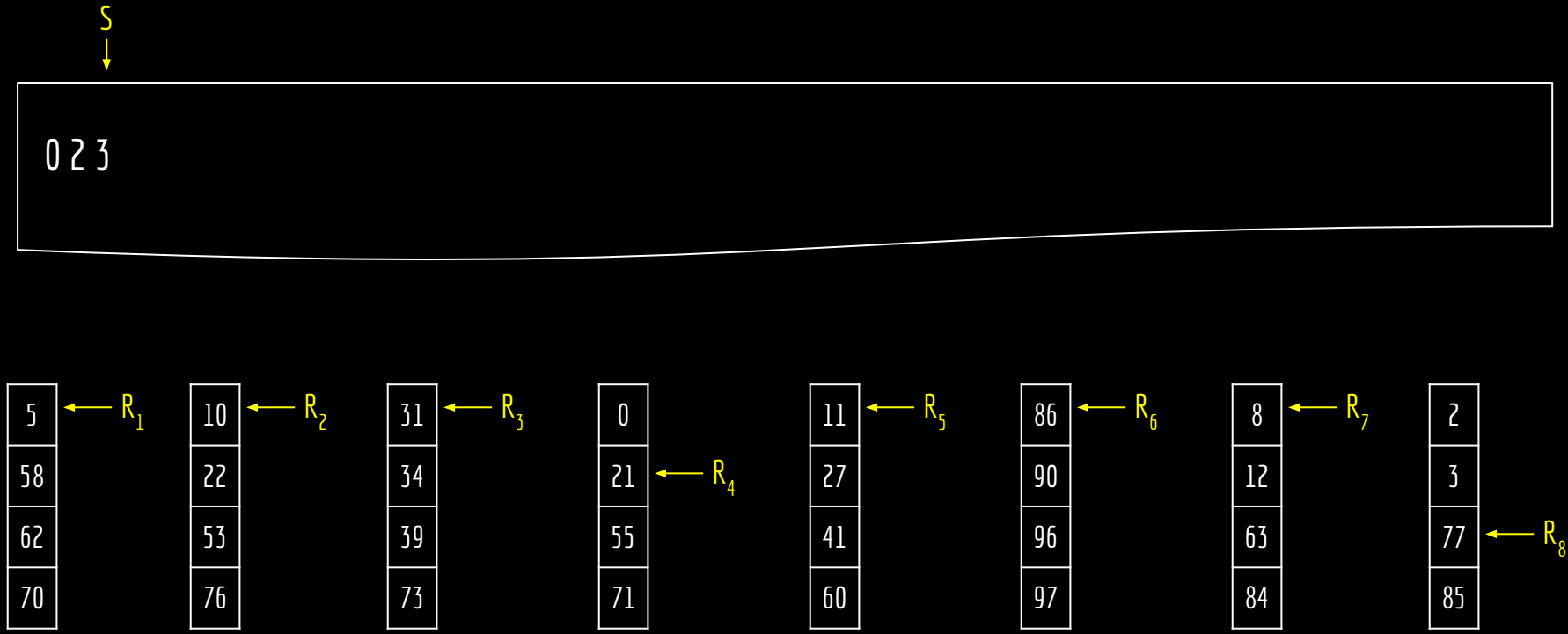
Exemplo

Verificar qual dos ponteiros escrever a seguir no arquivo.
Utilizar um Selection Sort, tomando *P-1* comparações a cada iteração.



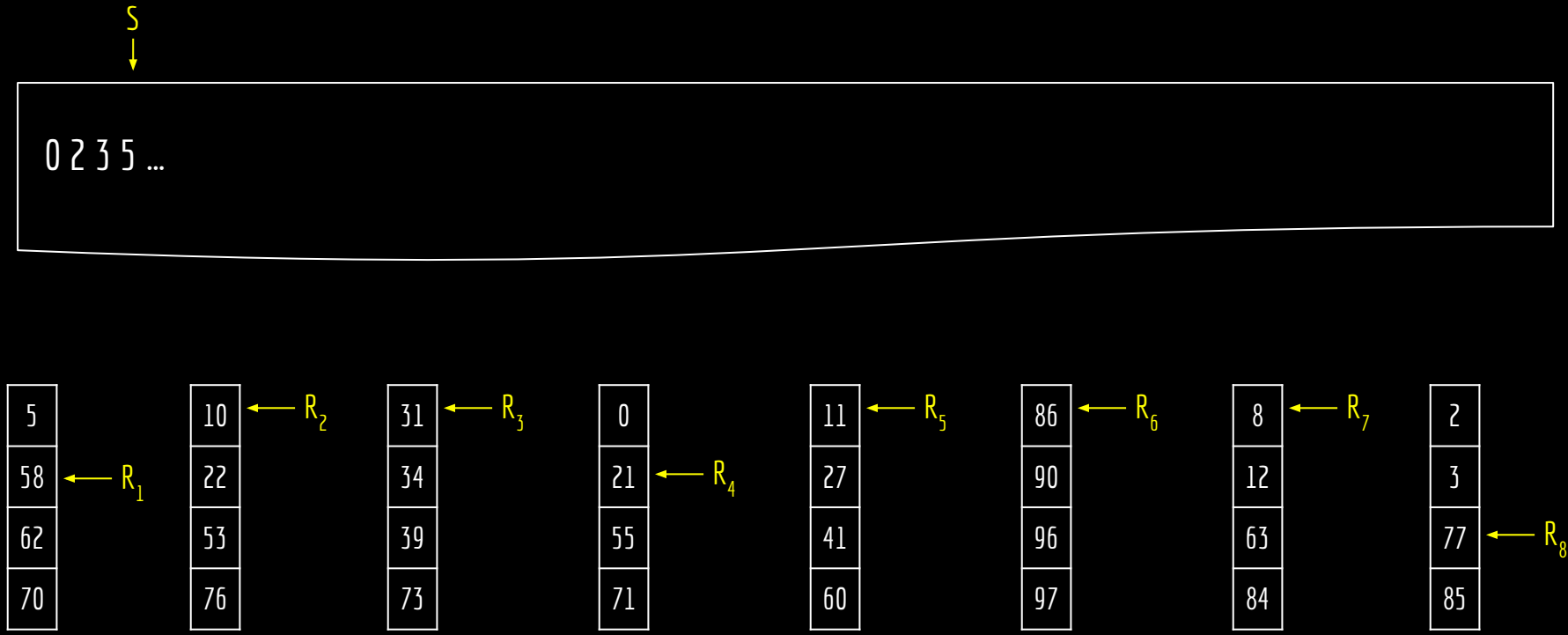
Exemplo

Verificar qual dos ponteiros escrever a seguir no arquivo.
Utilizar um Selection Sort, tomando *P-1* comparações a cada iteração.



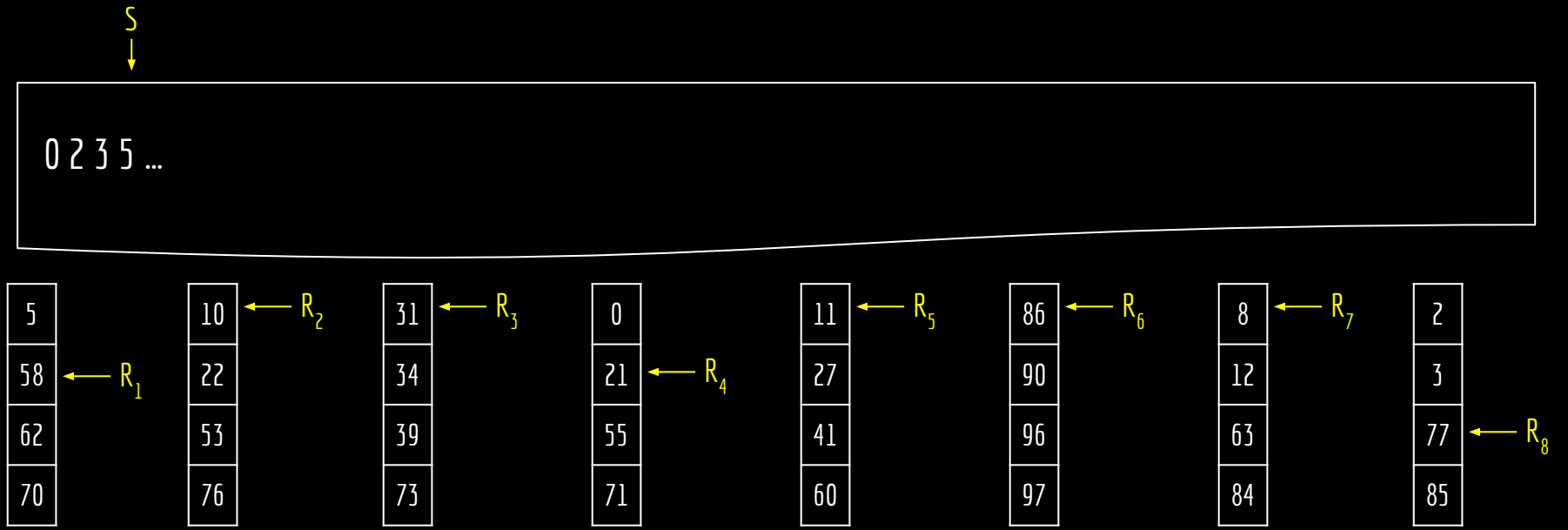
Exemplo

Verificar qual dos ponteiros escrever a seguir no arquivo.
Utilizar um Selection Sort, tomando *P-1* comparações a cada iteração.



Problemas

Problemas com essa estratégia?



Problemas

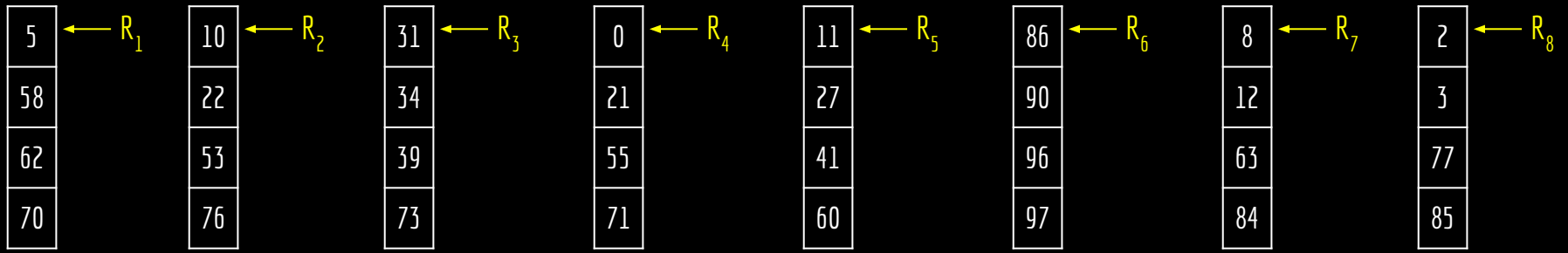
Problemas com essa estratégia?

- Funciona bem para valores de *P* pequenos.
- Valores grandes de *P* levam a um grande número de comparações a cada iteração pelo *Selection Sort*.

Resolvendo

Precisamos de uma estrutura que garantidamente mantenha a cabeça do arquivo vencedor prontamente acessível, e seja simples de atualizar.

O que podemos usar?

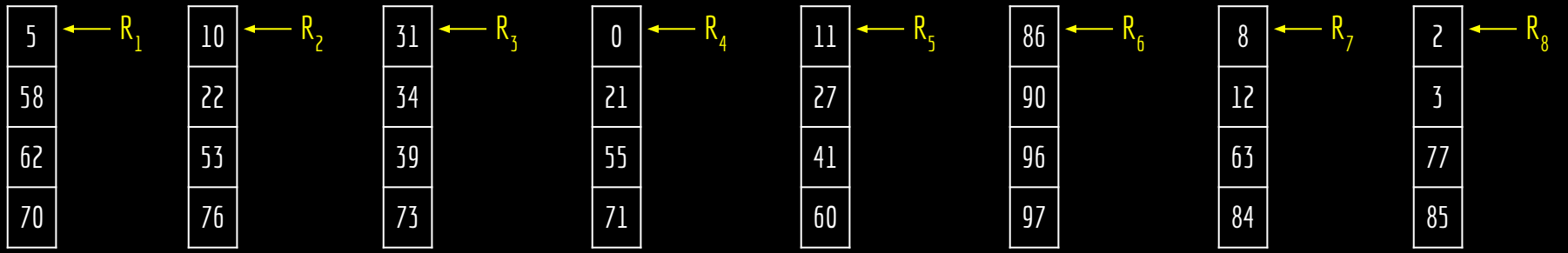


Heaps - Relembrando

Propriedades da Min-Heap.

1. A árvore é completa ou quase completa.
2. Uma heap de mínimo (min-heap) deve satisfazer a propriedade da min-heap.
   $h[pai(i)] \leq h[i], \ \forall i > 1$

Usando Heaps

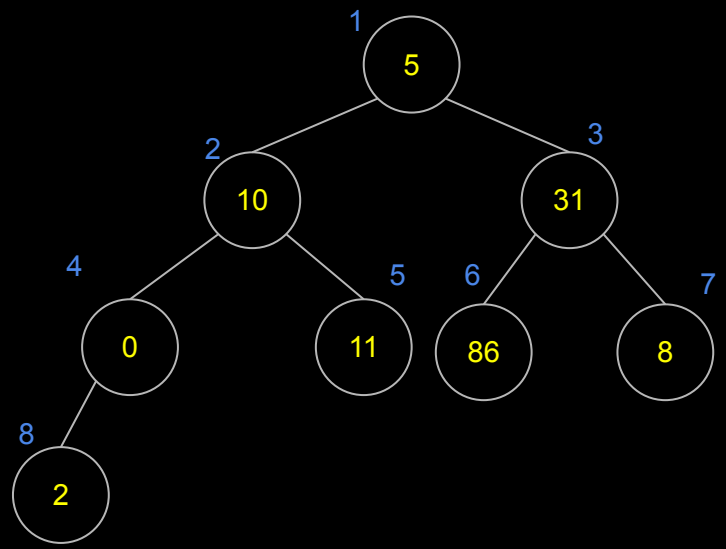


```
função construir-min-heap(v,n)
para i = n/2 até 1 passo -1
      min-heapify(v,i,n)
```

```
função min-heapify(h,i,n)
l = esquerda(i)
r = direita(i)
se l ≤ n e h[l] < h[i]
      menor = l
senão
      menor = i
se r ≤ n e h[r] < h[menor]
      menor = r
se menor ≠ i
      trocar(h,i,menor)
      min-heapify(h,menor,n)
```

construir-min-heap(v,8)

n	i
8	



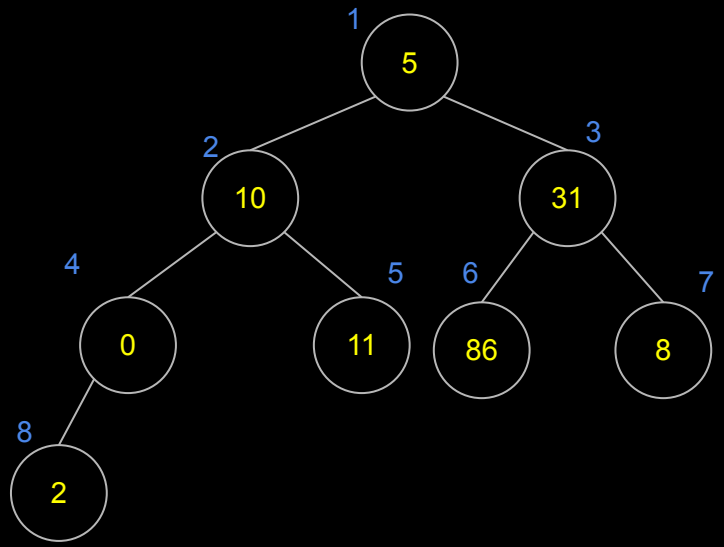
R_1	R_2	R_3	R_4	R_5	R_6	R_7	R_8
5	10	31	0	11	86	8	2
58	22	34	21	27	90	12	3
62	53	39	55	41	96	63	77
70	76	73	71	60	97	84	85

```
função construir-min-heap(v,n)
para i = n/2 até 1 passo -1
      min-heapify(v,i,n)
```

construir-min-heap(v,8)

n	i
8	4

```
função min-heapify(h,i,n)
l = esquerda(i)
r = direita(i)
se l ≤ n e h[l] < h[i]
      menor = l
senão
      menor = i
se r ≤ n e h[r] < h[menor]
      menor = r
se menor ≠ i
      trocar(h,i,menor)
      min-heapify(h,menor,n)
```



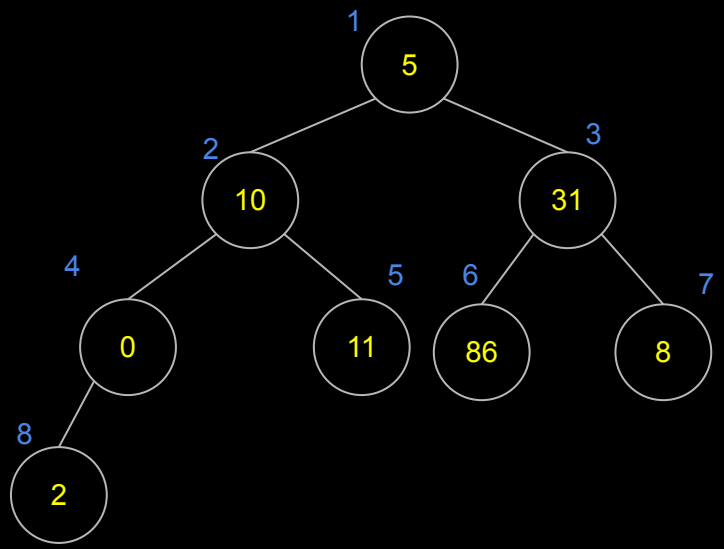
R_1	R_2	R_3	R_4	R_5	R_6	R_7	R_8
5	10	31	0	11	86	8	2
58	22	34	21	27	90	12	3
62	53	39	55	41	96	63	77
70	76	73	71	60	97	84	85

```
função construir-min-heap(v,n)
para i = n/2 até 1 passo -1
      min-heapify(v,i,n)
```

construir-min-heap(v,8)

n	i
8	4

```
função min-heapify(h,i,n)
l = esquerda(i)
r = direita(i)
se l ≤ n e h[l] < h[i]
      menor = l
senão
      menor = i
se r ≤ n e h[r] < h[menor]
      menor = r
se menor ≠ i
      trocar(h,i,menor)
      min-heapify(h,menor,n)
```



R_1	R_2	R_3	R_4	R_5	R_6	R_7	R_8
5	10	31	0	11	86	8	2
58	22	34	21	27	90	12	3
62	53	39	55	41	96	63	77
70	76	73	71	60	97	84	85

```
função construir-min-heap(v,n)
para i = n/2 até 1 passo -1
      min-heapify(v,i,n)
```

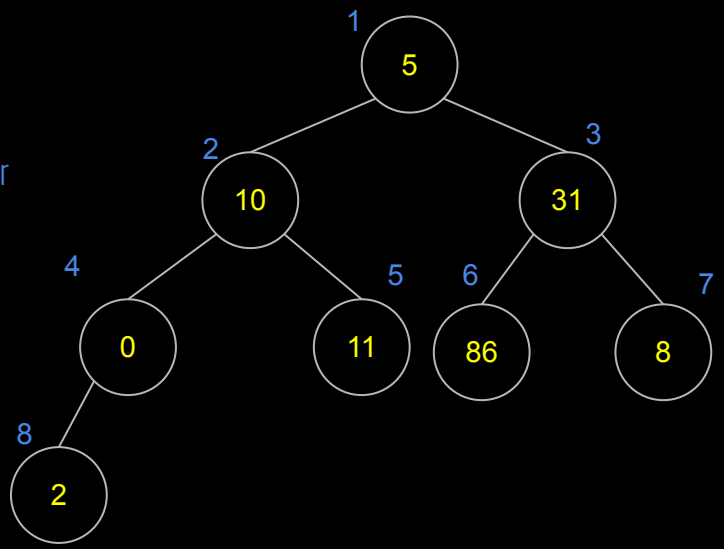
```
função min-heapify(h,i,n)
l = esquerda(i)
r = direita(i)
se l ≤ n e h[l] < h[i]
      menor = l
senão
      menor = i
se r ≤ n e h[r] < h[menor]
      menor = r
se menor ≠ i
      trocar(h,i,menor)
      min-heapify(h,menor,n)
```

construir-min-heap(v,8)

n	i
8	4

min-heapify(v,4,8)

n	i	l	r	menor
8	4	8		



R_1	R_2	R_3	R_4	R_5	R_6	R_7	R_8
5	10	31	0	11	86	8	2
58	22	34	21	27	90	12	3
62	53	39	55	41	96	63	77
70	76	73	71	60	97	84	85

```
função construir-min-heap(v,n)
para i = n/2 até 1 passo -1
      min-heapify(v,i,n)
```

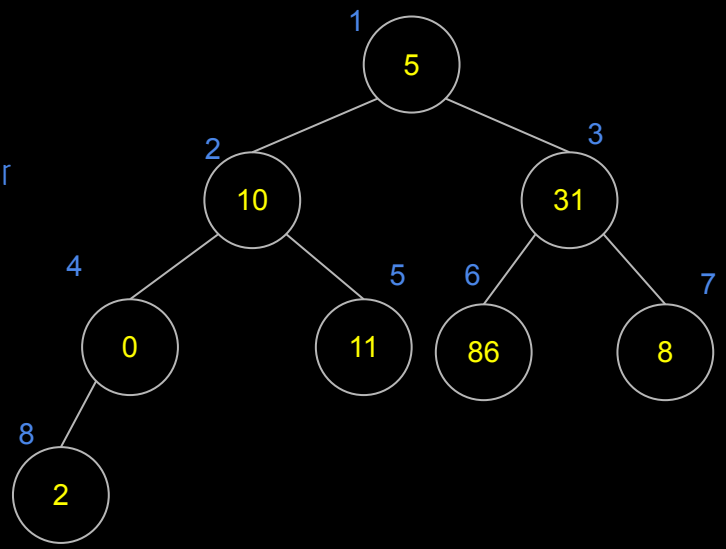
```
função min-heapify(h,i,n)
l = esquerda(i)
r = direita(i)
se l ≤ n e h[l] < h[i]
      menor = l
senão
      menor = i
se r ≤ n e h[r] < h[menor]
      menor = r
se menor ≠ i
      trocar(h,i,menor)
      min-heapify(h,menor,n)
```

construir-min-heap(v,8)

n	i
8	4

min-heapify(v,4,8)

n	i	l	r	menor
8	4	8	9	



R_1	R_2	R_3	R_4	R_5	R_6	R_7	R_8
5	10	31	0	11	86	8	2
58	22	34	21	27	90	12	3
62	53	39	55	41	96	63	77
70	76	73	71	60	97	84	85

```
função construir-min-heap(v,n)
para i = n/2 até 1 passo -1
      min-heapify(v,i,n)
```

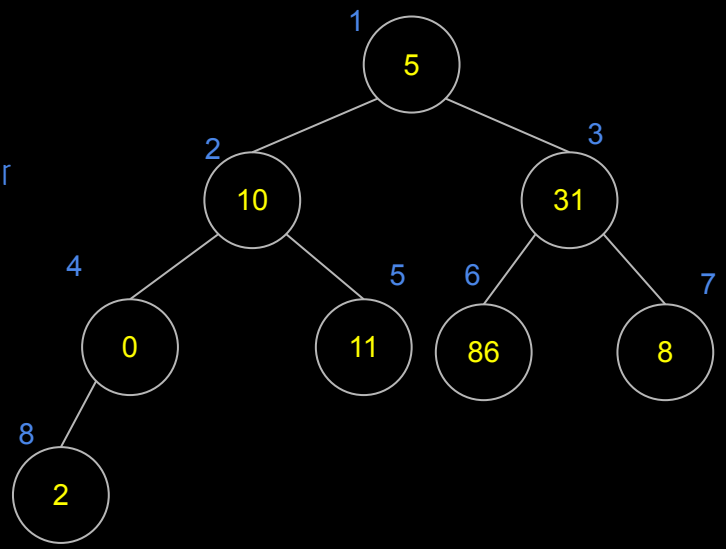
```
função min-heapify(h,i,n)
l = esquerda(i)
r = direita(i)
se l ≤ n e h[l] < h[i]
      menor = l
senão
      menor = i
se r ≤ n e h[r] < h[menor]
      menor = r
se menor ≠ i
      trocar(h,i,menor)
      min-heapify(h,menor,n)
```

construir-min-heap(v,8)

n	i
8	4

min-heapify(v,4,8)

n	i	l	r	menor
8	4	8	9	4



R_1	R_2	R_3	R_4	R_5	R_6	R_7	R_8
5	10	31	0	11	86	8	2
58	22	34	21	27	90	12	3
62	53	39	55	41	96	63	77
70	76	73	71	60	97	84	85

```
função construir-min-heap(v,n)
para i = n/2 até 1 passo -1
      min-heapify(v,i,n)
```

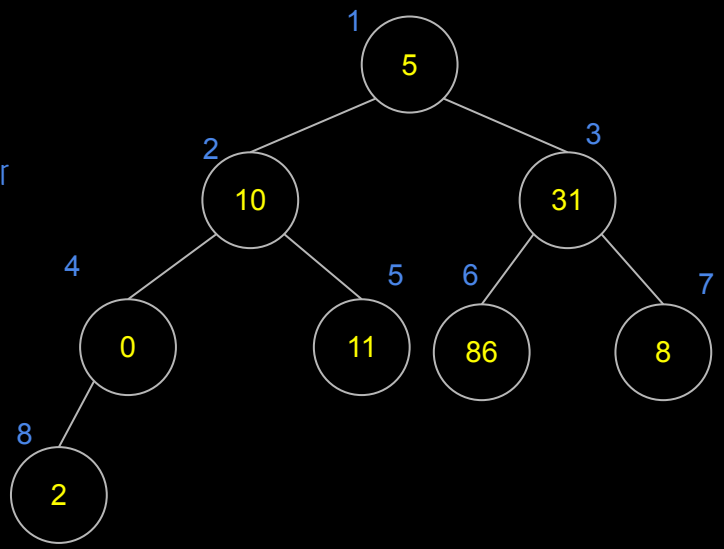
```
função min-heapify(h,i,n)
l = esquerda(i)
r = direita(i)
se l ≤ n e h[l] < h[i]
      menor = l
senão
      menor = i
se r ≤ n e h[r] < h[menor]
      menor = r
se menor ≠ i
      trocar(h,i,menor)
      min-heapify(h,menor,n)
```

construir-min-heap(v,8)

n	i
8	4

min-heapify(v,4,8)

n	i	l	r	menor
8	4	8	9	4



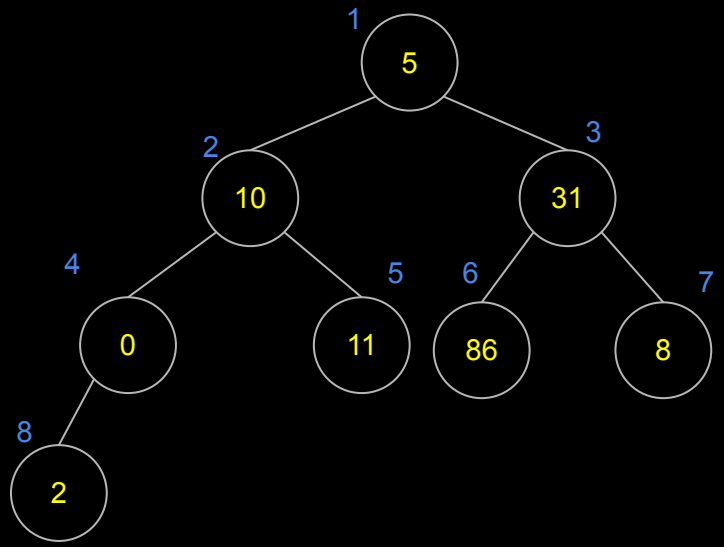
R_1	R_2	R_3	R_4	R_5	R_6	R_7	R_8
5	10	31	0	11	86	8	2
58	22	34	21	27	90	12	3
62	53	39	55	41	96	63	77
70	76	73	71	60	97	84	85

```
função construir-min-heap(v,n)
para i = n/2 até 1 passo -1
      min-heapify(v,i,n)
```

construir-min-heap(v,8)

n	i
8	3

```
função min-heapify(h,i,n)
l = esquerda(i)
r = direita(i)
se l ≤ n e h[l] < h[i]
      menor = l
senão
      menor = i
se r ≤ n e h[r] < h[menor]
      menor = r
se menor ≠ i
      trocar(h,i,menor)
      min-heapify(h,menor,n)
```



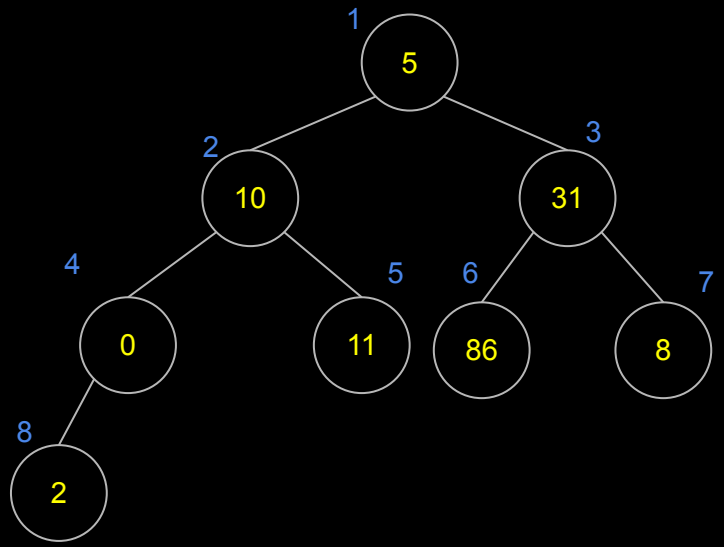
R_1	R_2	R_3	R_4	R_5	R_6	R_7	R_8
5	10	31	0	11	86	8	2
58	22	34	21	27	90	12	3
62	53	39	55	41	96	63	77
70	76	73	71	60	97	84	85

```
função construir-min-heap(v,n)
para i = n/2 até 1 passo -1
      min-heapify(v,i,n)
```

construir-min-heap(v,8)

n	i
8	3

```
função min-heapify(h,i,n)
l = esquerda(i)
r = direita(i)
se l ≤ n e h[l] < h[i]
      menor = l
senão
      menor = i
se r ≤ n e h[r] < h[menor]
      menor = r
se menor ≠ i
      trocar(h,i,menor)
      min-heapify(h,menor,n)
```



R_1	R_2	R_3	R_4	R_5	R_6	R_7	R_8
5	10	31	0	11	86	8	2
58	22	34	21	27	90	12	3
62	53	39	55	41	96	63	77
70	76	73	71	60	97	84	85

```
função construir-min-heap(v,n)
para i = n/2 até 1 passo -1
      min-heapify(v,i,n)
```

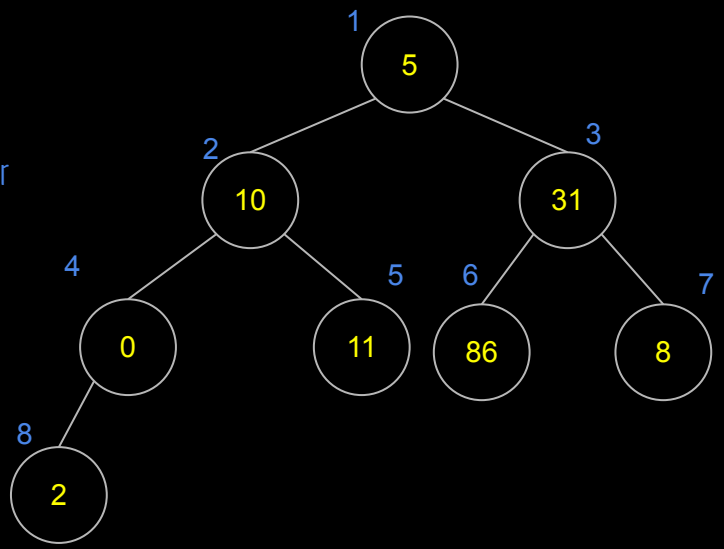
```
função min-heapify(h,i,n)
l = esquerda(i)
r = direita(i)
se l ≤ n e h[l] < h[i]
      menor = l
senão
      menor = i
se r ≤ n e h[r] < h[menor]
      menor = r
se menor ≠ i
      trocar(h,i,menor)
      min-heapify(h,menor,n)
```

construir-min-heap(v,8)

n	i
8	3

min-heapify(v,3,8)

n	i	l	r	menor
8	3	6	7	



R_1	R_2	R_3	R_4	R_5	R_6	R_7	R_8
5	10	31	0	11	86	8	2
58	22	34	21	27	90	12	3
62	53	39	55	41	96	63	77
70	76	73	71	60	97	84	85

```
função construir-min-heap(v,n)
para i = n/2 até 1 passo -1
      min-heapify(v,i,n)
```

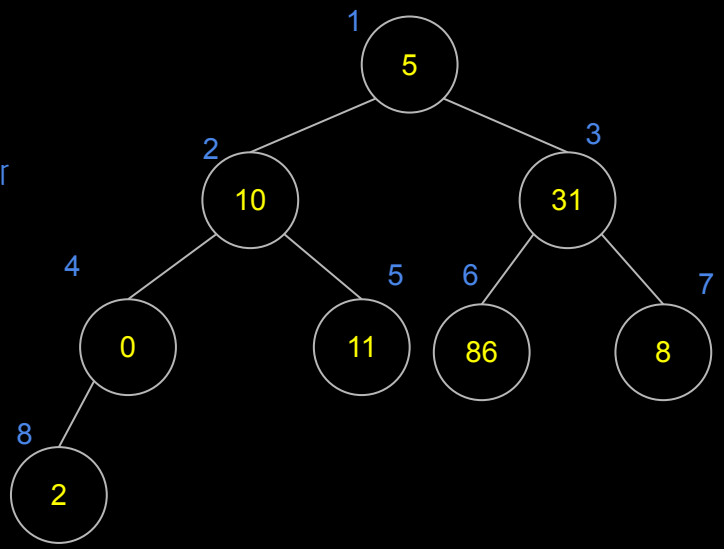
construir-min-heap(v,8)

n	i
8	3

```
função min-heapify(h,i,n)
l = esquerda(i)
r = direita(i)
se l ≤ n e h[l] < h[i]
      menor = l
senão
      menor = i
se r ≤ n e h[r] < h[menor]
      menor = r
se menor ≠ i
      trocar(h,i,menor)
      min-heapify(h,menor,n)
```

min-heapify(v,3,8)

n	i	l	r	menor
8	3	6	7	7



R_1	R_2	R_3	R_4	R_5	R_6	R_7	R_8
5	10	31	0	11	86	8	2
58	22	34	21	27	90	12	3
62	53	39	55	41	96	63	77
70	76	73	71	60	97	84	85

```
função construir-min-heap(v,n)
para i = n/2 até 1 passo -1
      min-heapify(v,i,n)
```

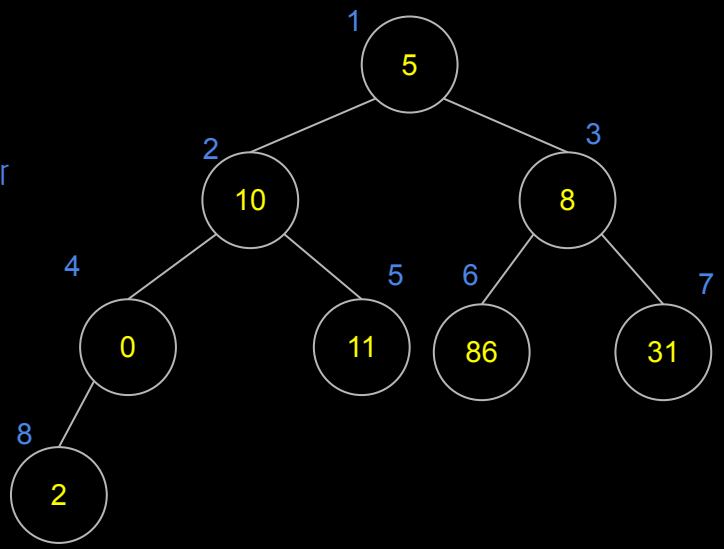
```
função min-heapify(h,i,n)
l = esquerda(i)
r = direita(i)
se l ≤ n e h[l] < h[i]
      menor = l
senão
      menor = i
se r ≤ n e h[r] < h[menor]
      menor = r
se menor ≠ i
      trocar(h,i,menor)
      min-heapify(h,menor,n)
```

construir-min-heap(v,8)

n	i
8	3

min-heapify(v,3,8)

n	i	l	r	menor
8	3	6	7	7



R_1	R_2	R_3	R_4	R_5	R_6	R_7	R_8
5	10	31	0	11	86	8	2
58	22	34	21	27	90	12	3
62	53	39	55	41	96	63	77
70	76	73	71	60	97	84	85

```
função construir-min-heap(v,n)
para i = n/2 até 1 passo -1
      min-heapify(v,i,n)
```

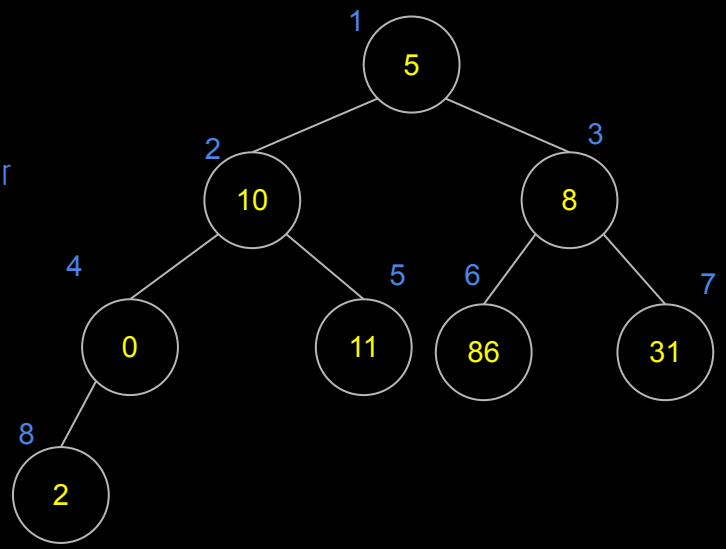
construir-min-heap(v,8)

n	i
8	3

```
função min-heapify(h,i,n)
l = esquerda(i)
r = direita(i)
se l ≤ n e h[l] < h[i]
      menor = l
senão
      menor = i
se r ≤ n e h[r] < h[menor]
      menor = r
se menor ≠ i
      trocar(h,i,menor)
      min-heapify(h,menor,n)
```

min-heapify(v,3,8)

n	i	l	r	menor
8	3	6	7	7



R_1	R_2	R_3	R_4	R_5	R_6	R_7	R_8
5	10	31	0	11	86	8	2
58	22	34	21	27	90	12	3
62	53	39	55	41	96	63	77
70	76	73	71	60	97	84	85

```
função construir-min-heap(v,n)
para i = n/2 até 1 passo -1
      min-heapify(v,i,n)
```

```
função min-heapify(h,i,n)
l = esquerda(i)
r = direita(i)
se l ≤ n e h[l] < h[i]
      menor = l
senão
      menor = i
se r ≤ n e h[r] < h[menor]
      menor = r
se menor ≠ i
      trocar(h,i,menor)
      min-heapify(h,menor,n)
```

construir-min-heap(v,8)

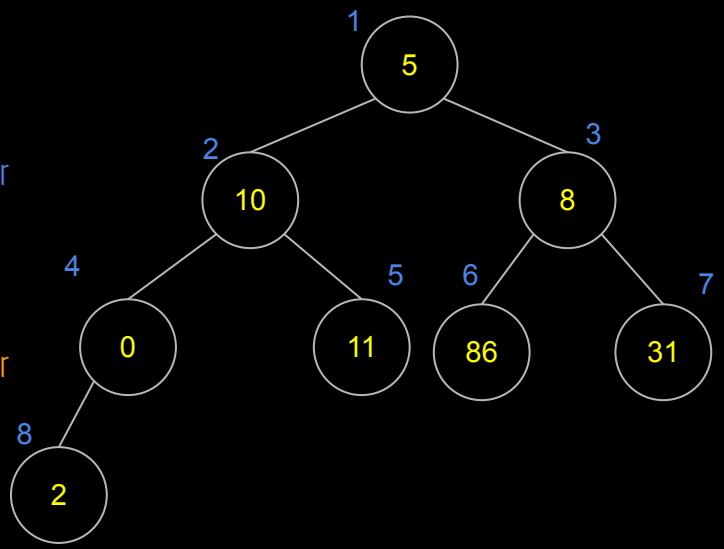
n	i
8	3

min-heapify(v,3,8)

n	i	l	r	menor
8	3	6	7	7

min-heapify(v,3,7)

n	i	l	r	menor
8	7	14	15	



R_1	R_2	R_3	R_4	R_5	R_6	R_7	R_8
5	10	31	0	11	86	8	2
58	22	34	21	27	90	12	3
62	53	39	55	41	96	63	77
70	76	73	71	60	97	84	85

```
função construir-min-heap(v,n)
para i = n/2 até 1 passo -1
      min-heapify(v,i,n)
```

```
função min-heapify(h,i,n)
l = esquerda(i)
r = direita(i)
se l ≤ n e h[l] < h[i]
      menor = l
senão
      menor = i
se r ≤ n e h[r] < h[menor]
      menor = r
se menor ≠ i
      trocar(h,i,menor)
      min-heapify(h,menor,n)
```

construir-min-heap(v,8)

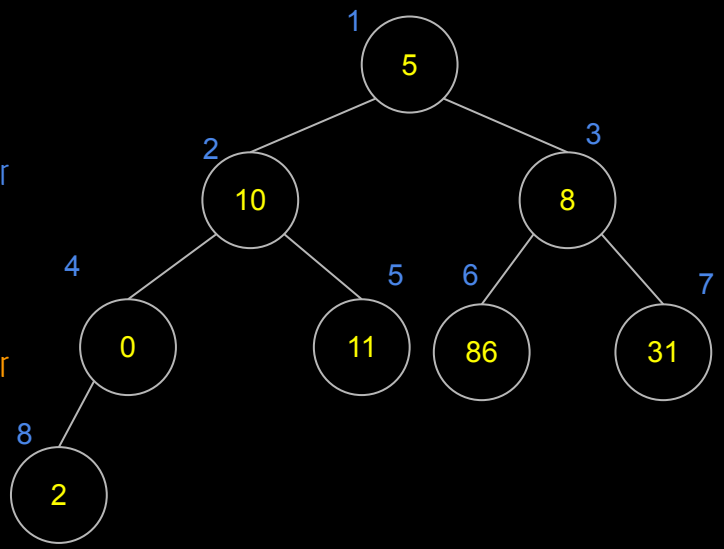
n	i
8	3

min-heapify(v,3,8)

n	i	l	r	menor
8	3	6	7	7

min-heapify(v,3,7)

n	i	l	r	menor
8	7	14	15	7



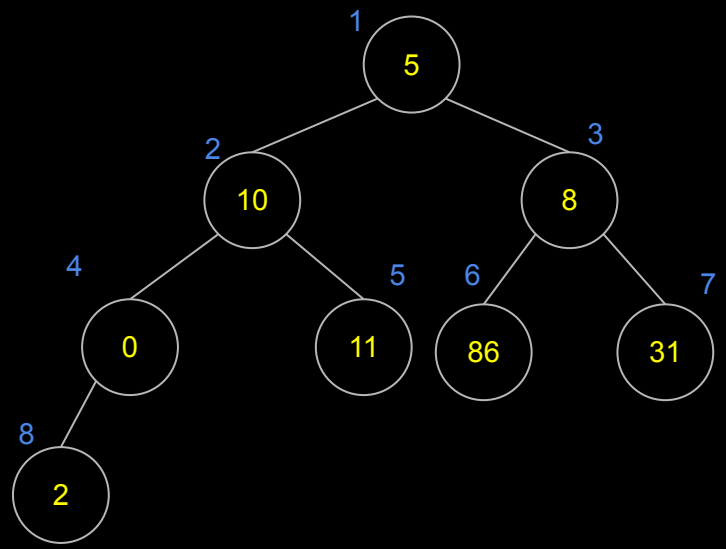
R_1	R_2	R_3	R_4	R_5	R_6	R_7	R_8
5	10	31	0	11	86	8	2
58	22	34	21	27	90	12	3
62	53	39	55	41	96	63	77
70	76	73	71	60	97	84	85

```
função construir-min-heap(v,n)
para i = n/2 até 1 passo -1
      min-heapify(v,i,n)
```

construir-min-heap(v,8)

n	i
8	2

```
função min-heapify(h,i,n)
l = esquerda(i)
r = direita(i)
se l ≤ n e h[l] < h[i]
      menor = l
senão
      menor = i
se r ≤ n e h[r] < h[menor]
      menor = r
se menor ≠ i
      trocar(h,i,menor)
      min-heapify(h,menor,n)
```



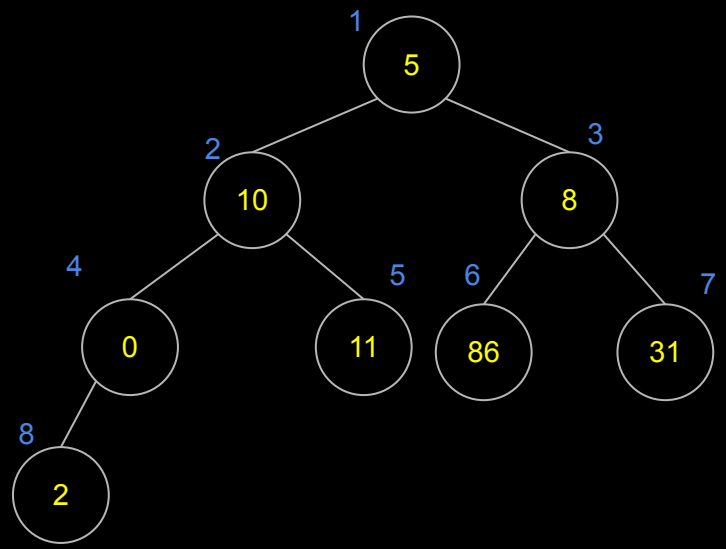
R_1	R_2	R_3	R_4	R_5	R_6	R_7	R_8
5	10	31	0	11	86	8	2
58	22	34	21	27	90	12	3
62	53	39	55	41	96	63	77
70	76	73	71	60	97	84	85

```
função construir-min-heap(v,n)
para i = n/2 até 1 passo -1
      min-heapify(v,i,n)
```

construir-min-heap(v,8)

n	i
8	2

```
função min-heapify(h,i,n)
l = esquerda(i)
r = direita(i)
se l ≤ n e h[l] < h[i]
      menor = l
senão
      menor = i
se r ≤ n e h[r] < h[menor]
      menor = r
se menor ≠ i
      trocar(h,i,menor)
      min-heapify(h,menor,n)
```



R_1	R_2	R_3	R_4	R_5	R_6	R_7	R_8
5	10	31	0	11	86	8	2
58	22	34	21	27	90	12	3
62	53	39	55	41	96	63	77
70	76	73	71	60	97	84	85

```
função construir-min-heap(v,n)
para i = n/2 até 1 passo -1
      min-heapify(v,i,n)
```

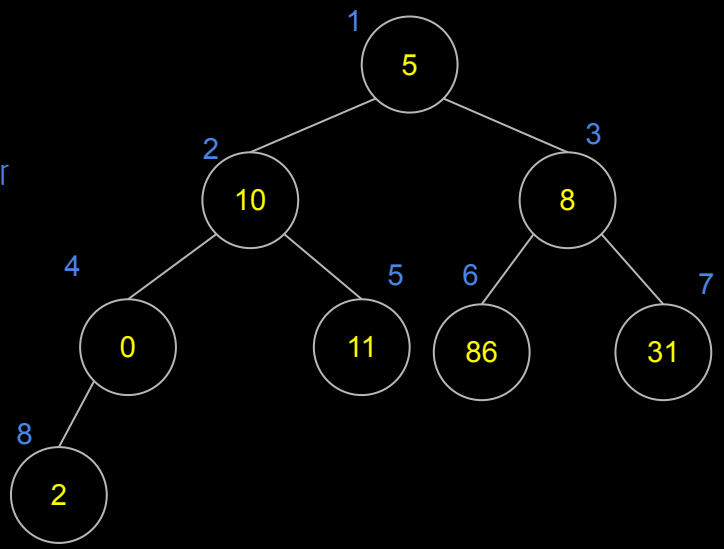
```
função min-heapify(h,i,n)
l = esquerda(i)
r = direita(i)
se l ≤ n e h[l] < h[i]
      menor = l
senão
      menor = i
se r ≤ n e h[r] < h[menor]
      menor = r
se menor ≠ i
      trocar(h,i,menor)
      min-heapify(h,menor,n)
```

construir-min-heap(v,8)

n	i
8	2

min-heapify(v,2,8)

n	i	l	r	menor
8	2	4	5	



R_1	R_2	R_3	R_4	R_5	R_6	R_7	R_8
5	10	31	0	11	86	8	2
58	22	34	21	27	90	12	3
62	53	39	55	41	96	63	77
70	76	73	71	60	97	84	85

```
função construir-min-heap(v,n)
para i = n/2 até 1 passo -1
      min-heapify(v,i,n)
```

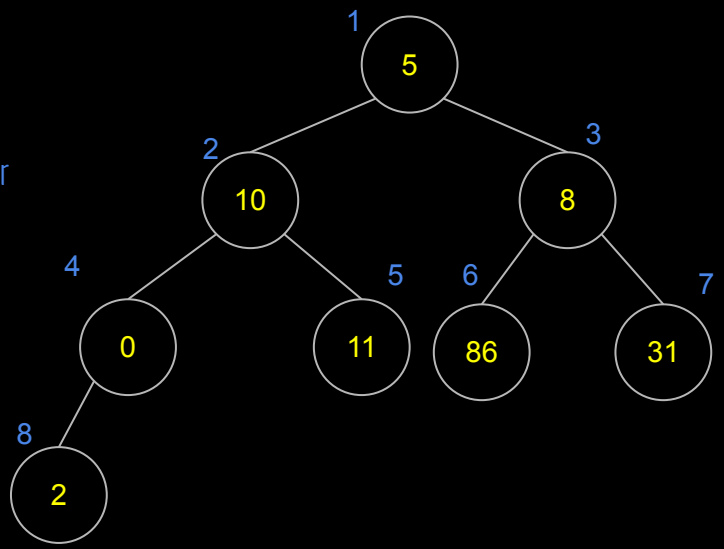
```
função min-heapify(h,i,n)
l = esquerda(i)
r = direita(i)
se l ≤ n e h[l] < h[i]
      menor = l
senão
      menor = i
se r ≤ n e h[r] < h[menor]
      menor = r
se menor ≠ i
      trocar(h,i,menor)
      min-heapify(h,menor,n)
```

construir-min-heap(v,8)

n	i
8	2

min-heapify(v,2,8)

n	i	l	r	menor
8	2	4	5	4



R_1	R_2	R_3	R_4	R_5	R_6	R_7	R_8
5	10	31	0	11	86	8	2
58	22	34	21	27	90	12	3
62	53	39	55	41	96	63	77
70	76	73	71	60	97	84	85

```
função construir-min-heap(v,n)
para i = n/2 até 1 passo -1
      min-heapify(v,i,n)
```

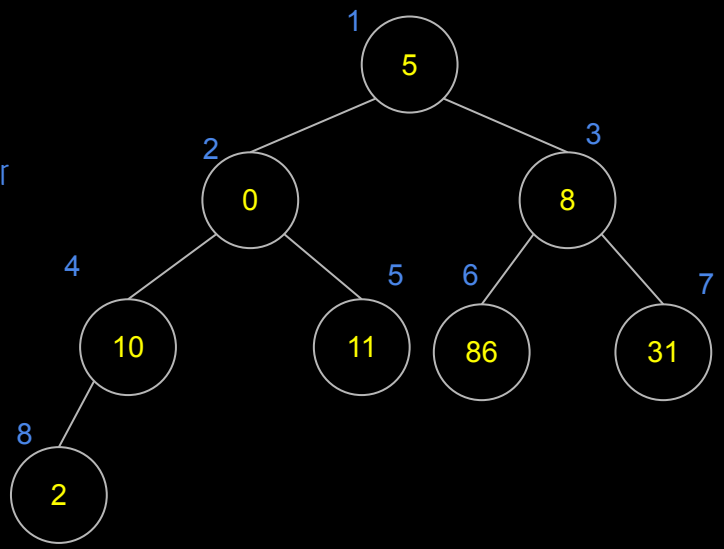
```
função min-heapify(h,i,n)
l = esquerda(i)
r = direita(i)
se l ≤ n e h[l] < h[i]
      menor = l
senão
      menor = i
se r ≤ n e h[r] < h[menor]
      menor = r
se menor ≠ i
      trocar(h,i,menor)
      min-heapify(h,menor,n)
```

construir-min-heap(v,8)

n	i
8	2

min-heapify(v,2,8)

n	i	l	r	menor
8	2	4	5	4



R_1	R_2	R_3	R_4	R_5	R_6	R_7	R_8
5	10	31	0	11	86	8	2
58	22	34	21	27	90	12	3
62	53	39	55	41	96	63	77
70	76	73	71	60	97	84	85

```
função construir-min-heap(v,n)
para i = n/2 até 1 passo -1
      min-heapify(v,i,n)
```

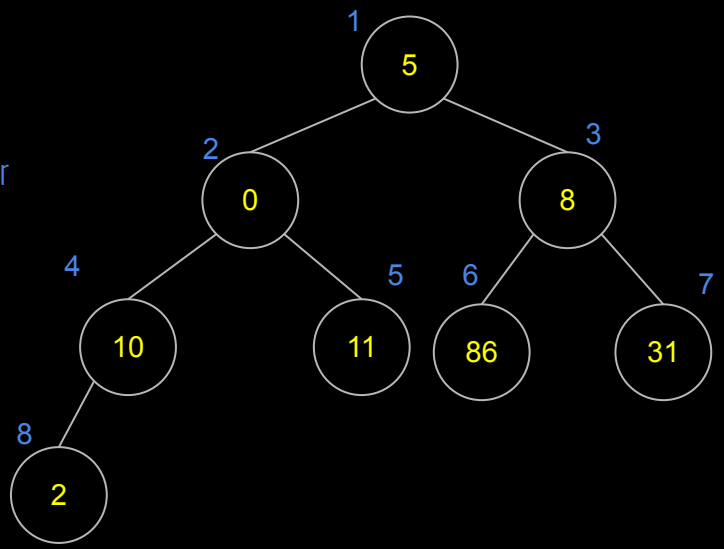
```
função min-heapify(h,i,n)
l = esquerda(i)
r = direita(i)
se l ≤ n e h[l] < h[i]
      menor = l
senão
      menor = i
se r ≤ n e h[r] < h[menor]
      menor = r
se menor ≠ i
      trocar(h,i,menor)
      min-heapify(h,menor,n)
```

construir-min-heap(v,8)

n	i
8	2

min-heapify(v,2,8)

n	i	l	r	menor
8	2	4	5	4



R_1	R_2	R_3	R_4	R_5	R_6	R_7	R_8
5	10	31	0	11	86	8	2
58	22	34	21	27	90	12	3
62	53	39	55	41	96	63	77
70	76	73	71	60	97	84	85

```
função construir-min-heap(v,n)
para i = n/2 até 1 passo -1
      min-heapify(v,i,n)
```

```
função min-heapify(h,i,n)
l = esquerda(i)
r = direita(i)
se l ≤ n e h[l] < h[i]
      menor = l
senão
      menor = i
se r ≤ n e h[r] < h[menor]
      menor = r
se menor ≠ i
      trocar(h,i,menor)
      min-heapify(h,menor,n)
```

construir-min-heap(v,8)

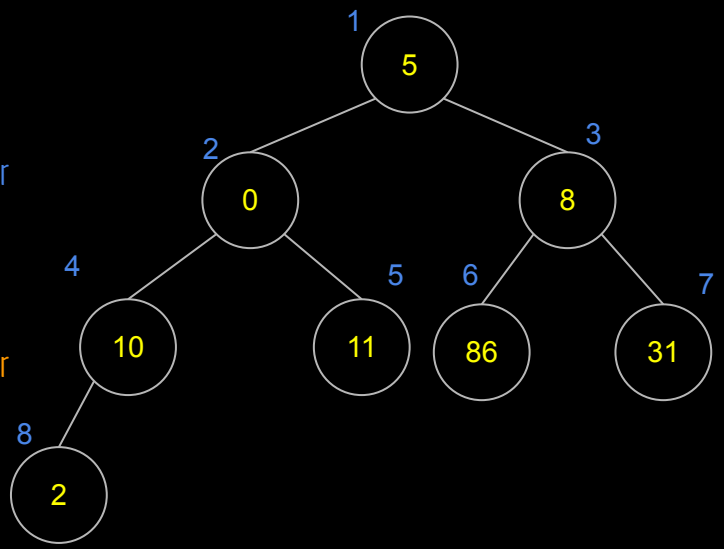
n	i
8	2

min-heapify(v,2,8)

n	i	l	r	menor
8	2	4	5	4

min-heapify(v,4,8)

n	i	l	r	menor
8	4	8	9	



R_1	R_2	R_3	R_4	R_5	R_6	R_7	R_8
5	10	31	0	11	86	8	2
58	22	34	21	27	90	12	3
62	53	39	55	41	96	63	77
70	76	73	71	60	97	84	85

```
função construir-min-heap(v,n)
para i = n/2 até 1 passo -1
      min-heapify(v,i,n)
```

```
função min-heapify(h,i,n)
l = esquerda(i)
r = direita(i)
se l ≤ n e h[l] < h[i]
      menor = l
senão
      menor = i
se r ≤ n e h[r] < h[menor]
      menor = r
se menor ≠ i
      trocar(h,i,menor)
      min-heapify(h,menor,n)
```

construir-min-heap(v,8)

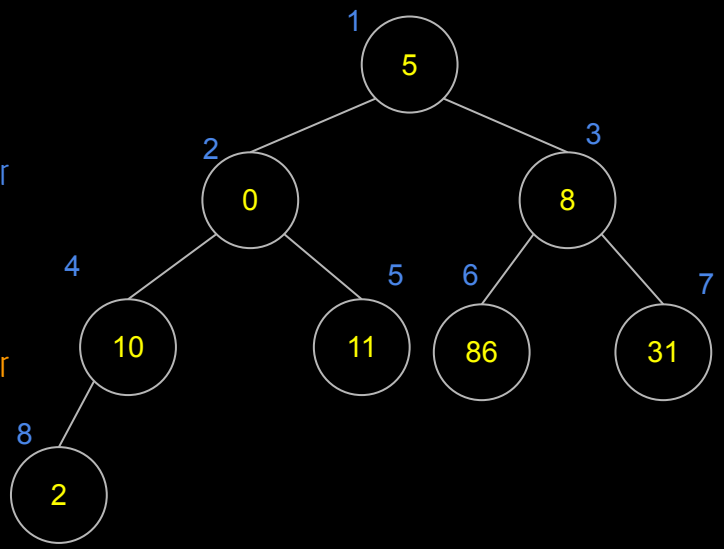
n	i
8	2

min-heapify(v,2,8)

n	i	l	r	menor
8	2	4	5	4

min-heapify(v,4,8)

n	i	l	r	menor
8	4	8	9	8



R_1	R_2	R_3	R_4	R_5	R_6	R_7	R_8
5	10	31	0	11	86	8	2
58	22	34	21	27	90	12	3
62	53	39	55	41	96	63	77
70	76	73	71	60	97	84	85

```
função construir-min-heap(v,n)
para i = n/2 até 1 passo -1
      min-heapify(v,i,n)
```

```
função min-heapify(h,i,n)
l = esquerda(i)
r = direita(i)
se l ≤ n e h[l] < h[i]
      menor = l
senão
      menor = i
se r ≤ n e h[r] < h[menor]
      menor = r
se menor ≠ i
      trocar(h,i,menor)
      min-heapify(h,menor,n)
```

construir-min-heap(v,8)

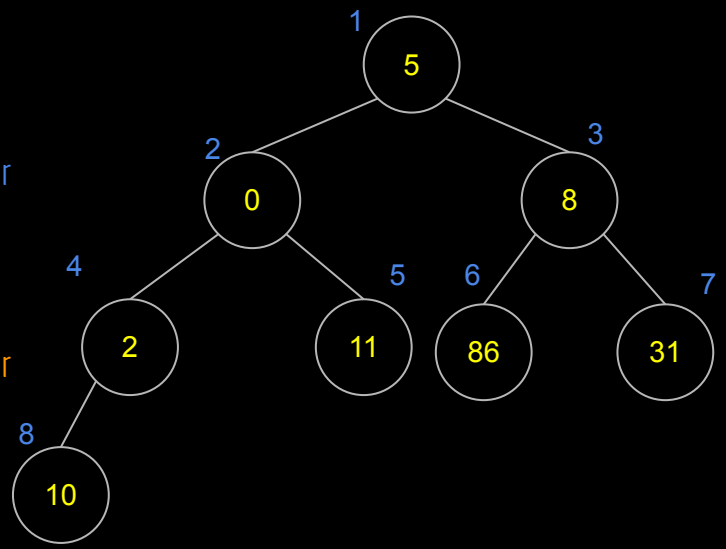
n	i
8	2

min-heapify(v,2,8)

n	i	l	r	menor
8	2	4	5	4

min-heapify(v,4,8)

n	i	l	r	menor
8	4	8	9	8



R_1	R_2	R_3	R_4	R_5	R_6	R_7	R_8
5	10	31	0	11	86	8	2
58	22	34	21	27	90	12	3
62	53	39	55	41	96	63	77
70	76	73	71	60	97	84	85

```
função construir-min-heap(v,n)
para i = n/2 até 1 passo -1
      min-heapify(v,i,n)
```

```
função min-heapify(h,i,n)
l = esquerda(i)
r = direita(i)
se l ≤ n e h[l] < h[i]
      menor = l
senão
      menor = i
se r ≤ n e h[r] < h[menor]
      menor = r
se menor ≠ i
      trocar(h,i,menor)
      min-heapify(h,menor,n)
```

construir-min-heap(v,8)

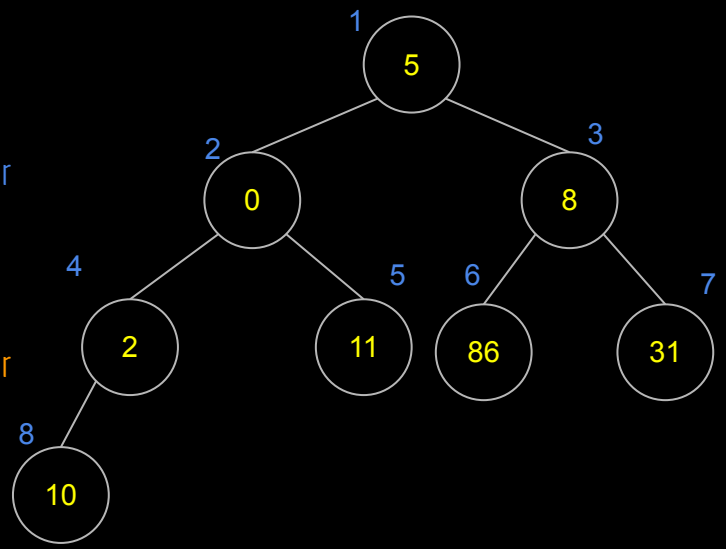
n	i
8	2

min-heapify(v,2,8)

n	i	l	r	menor
8	2	4	5	4

min-heapify(v,4,8)

n	i	l	r	menor
8	4	8	9	8



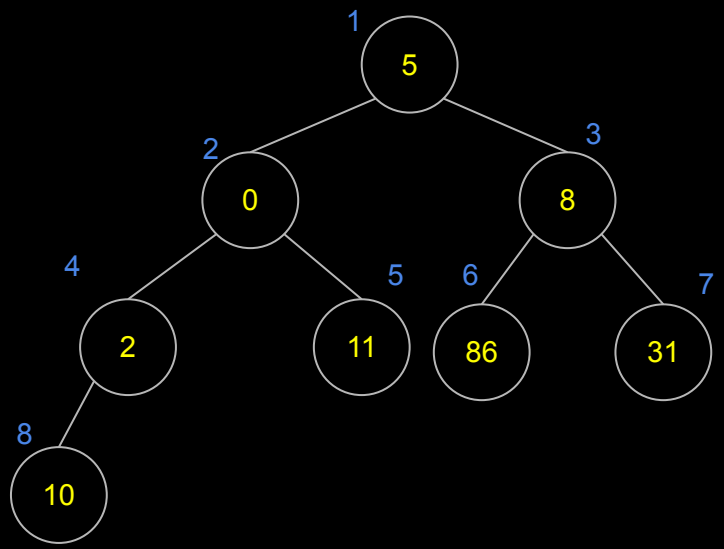
R_1	R_2	R_3	R_4	R_5	R_6	R_7	R_8
5	10	31	0	11	86	8	2
58	22	34	21	27	90	12	3
62	53	39	55	41	96	63	77
70	76	73	71	60	97	84	85

```
função construir-min-heap(v,n)
para i = n/2 até 1 passo -1
      min-heapify(v,i,n)
```

construir-min-heap(v,8)

n	i
8	1

```
função min-heapify(h,i,n)
l = esquerda(i)
r = direita(i)
se l ≤ n e h[l] < h[i]
      menor = l
senão
      menor = i
se r ≤ n e h[r] < h[menor]
      menor = r
se menor ≠ i
      trocar(h,i,menor)
      min-heapify(h,menor,n)
```



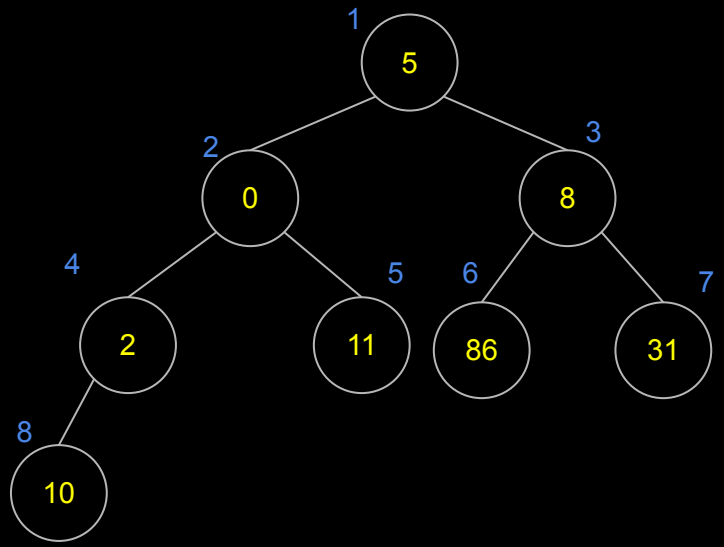
R_1	R_2	R_3	R_4	R_5	R_6	R_7	R_8
5	10	31	0	11	86	8	2
58	22	34	21	27	90	12	3
62	53	39	55	41	96	63	77
70	76	73	71	60	97	84	85

```
função construir-min-heap(v,n)
para i = n/2 até 1 passo -1
      min-heapify(v,i,n)
```

construir-min-heap(v,8)

n	i
8	1

```
função min-heapify(h,i,n)
l = esquerda(i)
r = direita(i)
se l ≤ n e h[l] < h[i]
      menor = l
senão
      menor = i
se r ≤ n e h[r] < h[menor]
      menor = r
se menor ≠ i
      trocar(h,i,menor)
      min-heapify(h,menor,n)
```



R_1	R_2	R_3	R_4	R_5	R_6	R_7	R_8
5	10	31	0	11	86	8	2
58	22	34	21	27	90	12	3
62	53	39	55	41	96	63	77
70	76	73	71	60	97	84	85

```
função construir-min-heap(v,n)
para i = n/2 até 1 passo -1
      min-heapify(v,i,n)
```

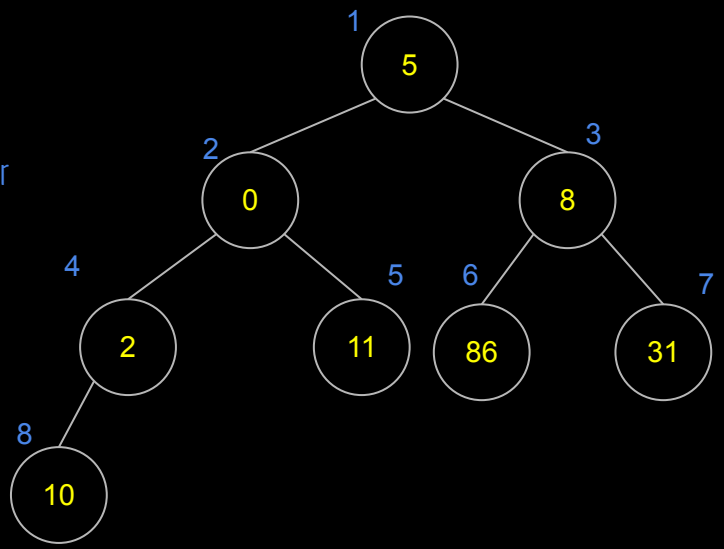
```
função min-heapify(h,i,n)
l = esquerda(i)
r = direita(i)
se l ≤ n e h[l] < h[i]
      menor = l
senão
      menor = i
se r ≤ n e h[r] < h[menor]
      menor = r
se menor ≠ i
      trocar(h,i,menor)
      min-heapify(h,menor,n)
```

construir-min-heap(v,8)

n	i
8	1

min-heapify(v,1,8)

n	i	l	r	menor
8	1	2	3	



R_1	R_2	R_3	R_4	R_5	R_6	R_7	R_8
5	10	31	0	11	86	8	2
58	22	34	21	27	90	12	3
62	53	39	55	41	96	63	77
70	76	73	71	60	97	84	85

```
função construir-min-heap(v,n)
para i = n/2 até 1 passo -1
      min-heapify(v,i,n)
```

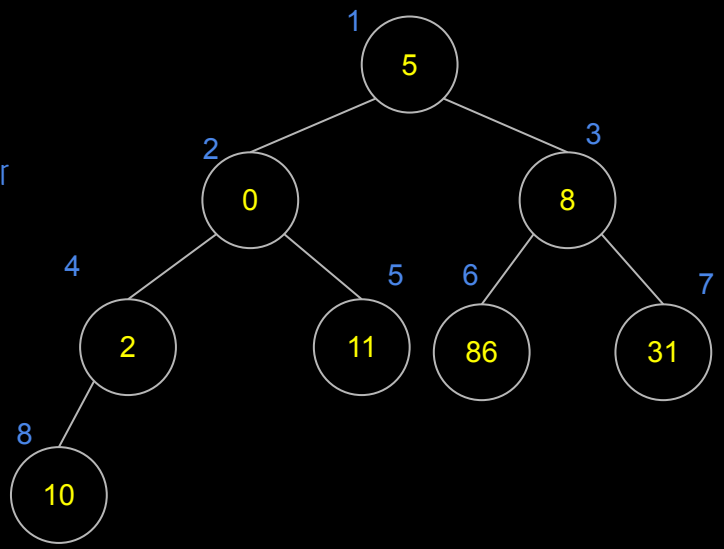
```
função min-heapify(h,i,n)
l = esquerda(i)
r = direita(i)
se l ≤ n e h[l] < h[i]
      menor = l
senão
      menor = i
se r ≤ n e h[r] < h[menor]
      menor = r
se menor ≠ i
      trocar(h,i,menor)
      min-heapify(h,menor,n)
```

construir-min-heap(v,8)

n	i
8	1

min-heapify(v,1,8)

n	i	l	r	menor
8	1	2	3	2



R_1	R_2	R_3	R_4	R_5	R_6	R_7	R_8
5	10	31	0	11	86	8	2
58	22	34	21	27	90	12	3
62	53	39	55	41	96	63	77
70	76	73	71	60	97	84	85

```
função construir-min-heap(v,n)
para i = n/2 até 1 passo -1
      min-heapify(v,i,n)
```

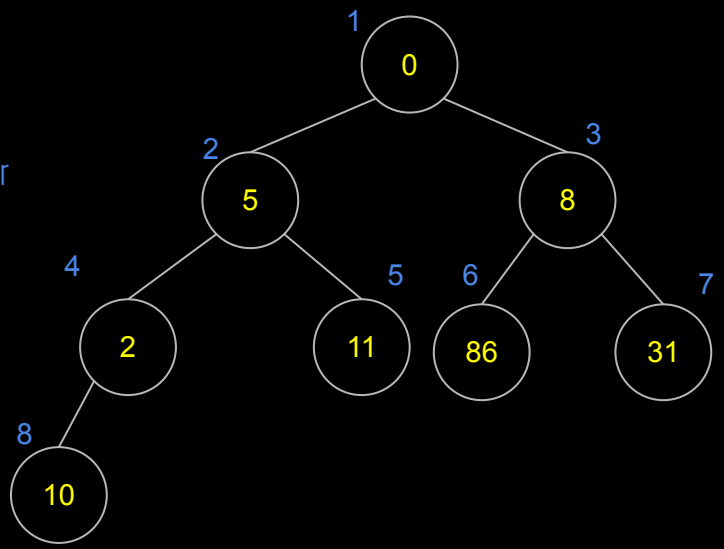
```
função min-heapify(h,i,n)
l = esquerda(i)
r = direita(i)
se l ≤ n e h[l] < h[i]
      menor = l
senão
      menor = i
se r ≤ n e h[r] < h[menor]
      menor = r
se menor ≠ i
      trocar(h,i,menor)
      min-heapify(h,menor,n)
```

construir-min-heap(v,8)

n	i
8	1

min-heapify(v,1,8)

n	i	l	r	menor
8	1	2	3	2



R_1	R_2	R_3	R_4	R_5	R_6	R_7	R_8
5	10	31	0	11	86	8	2
58	22	34	21	27	90	12	3
62	53	39	55	41	96	63	77
70	76	73	71	60	97	84	85

```
função construir-min-heap(v,n)
para i = n/2 até 1 passo -1
      min-heapify(v,i,n)
```

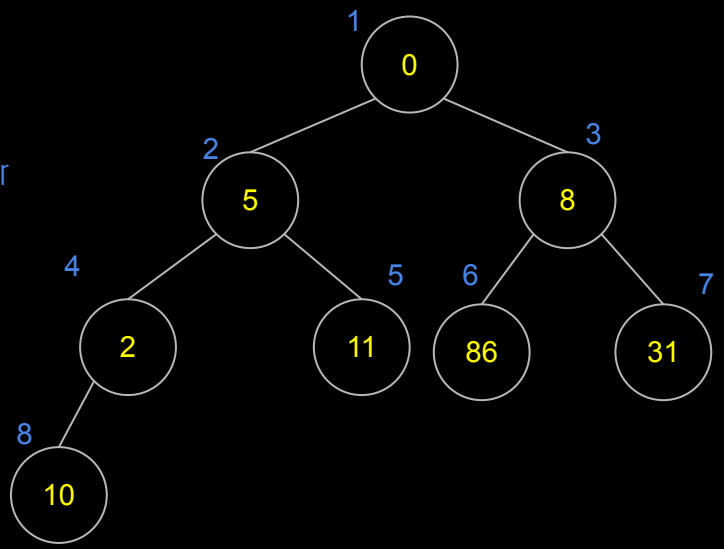
```
função min-heapify(h,i,n)
l = esquerda(i)
r = direita(i)
se l ≤ n e h[l] < h[i]
      menor = l
senão
      menor = i
se r ≤ n e h[r] < h[menor]
      menor = r
se menor ≠ i
      trocar(h,i,menor)
      min-heapify(h,menor,n)
```

construir-min-heap(v,8)

n	i
8	1

min-heapify(v,1,8)

n	i	l	r	menor
8	1	2	3	2



R_1	R_2	R_3	R_4	R_5	R_6	R_7	R_8
5	10	31	0	11	86	8	2
58	22	34	21	27	90	12	3
62	53	39	55	41	96	63	77
70	76	73	71	60	97	84	85

```
função construir-min-heap(v,n)
para i = n/2 até 1 passo -1
      min-heapify(v,i,n)
```

```
função min-heapify(h,i,n)
l = esquerda(i)
r = direita(i)
se l ≤ n e h[l] < h[i]
      menor = l
senão
      menor = i
se r ≤ n e h[r] < h[menor]
      menor = r
se menor ≠ i
      trocar(h,i,menor)
      min-heapify(h,menor,n)
```

construir-min-heap(v,8)

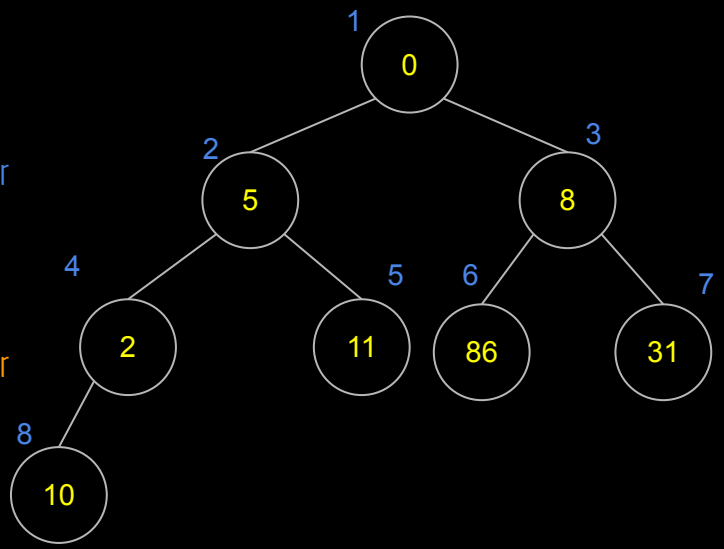
n	i
8	1

min-heapify(v,1,8)

n	i	l	r	menor
8	1	2	3	2

min-heapify(v,2,8)

n	i	l	r	menor
8	2	4	5	



R_1	R_2	R_3	R_4	R_5	R_6	R_7	R_8
5	10	31	0	11	86	8	2
58	22	34	21	27	90	12	3
62	53	39	55	41	96	63	77
70	76	73	71	60	97	84	85

```
função construir-min-heap(v,n)
para i = n/2 até 1 passo -1
      min-heapify(v,i,n)
```

```
função min-heapify(h,i,n)
l = esquerda(i)
r = direita(i)
se l ≤ n e h[l] < h[i]
      menor = l
senão
      menor = i
se r ≤ n e h[r] < h[menor]
      menor = r
se menor ≠ i
      trocar(h,i,menor)
      min-heapify(h,menor,n)
```

construir-min-heap(v,8)

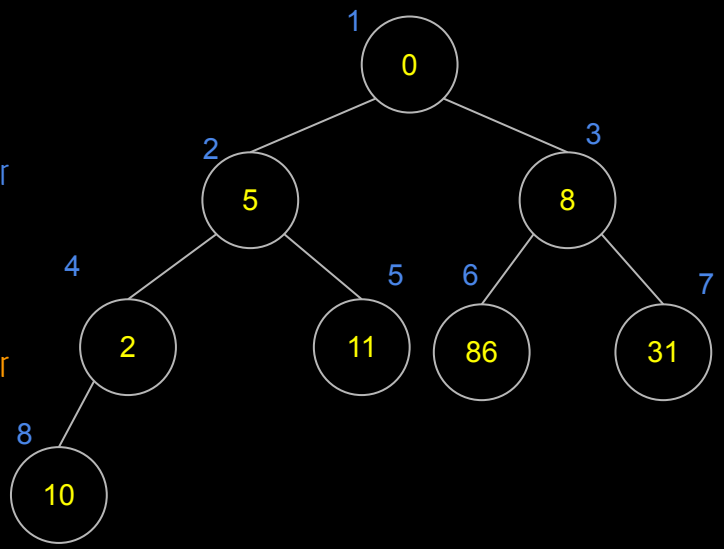
n	i
8	1

min-heapify(v,1,8)

n	i	l	r	menor
8	1	2	3	2

min-heapify(v,2,8)

n	i	l	r	menor
8	2	4	5	4



R_1	R_2	R_3	R_4	R_5	R_6	R_7	R_8
5	10	31	0	11	86	8	2
58	22	34	21	27	90	12	3
62	53	39	55	41	96	63	77
70	76	73	71	60	97	84	85

```
função construir-min-heap(v,n)
para i = n/2 até 1 passo -1
      min-heapify(v,i,n)
```

```
função min-heapify(h,i,n)
l = esquerda(i)
r = direita(i)
se l ≤ n e h[l] < h[i]
      menor = l
senão
      menor = i
se r ≤ n e h[r] < h[menor]
      menor = r
se menor ≠ i
      trocar(h,i,menor)
      min-heapify(h,menor,n)
```

construir-min-heap(v,8)

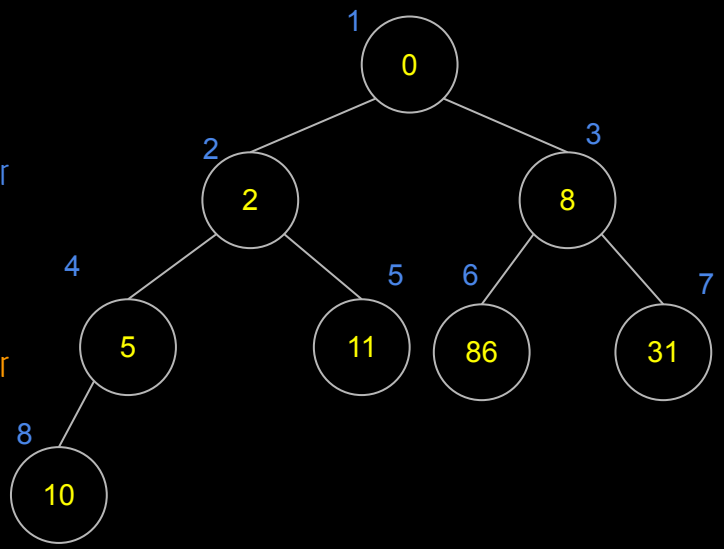
n	i
8	1

min-heapify(v,1,8)

n	i	l	r	menor
8	1	2	3	2

min-heapify(v,2,8)

n	i	l	r	menor
8	2	4	5	4



R_1	R_2	R_3	R_4	R_5	R_6	R_7	R_8
5	10	31	0	11	86	8	2
58	22	34	21	27	90	12	3
62	53	39	55	41	96	63	77
70	76	73	71	60	97	84	85

```
função construir-min-heap(v,n)
para i = n/2 até 1 passo -1
      min-heapify(v,i,n)
```

```
função min-heapify(h,i,n)
l = esquerda(i)
r = direita(i)
se l ≤ n e h[l] < h[i]
      menor = l
senão
      menor = i
se r ≤ n e h[r] < h[menor]
      menor = r
se menor ≠ i
      trocar(h,i,menor)
      min-heapify(h,menor,n)
```

construir-min-heap(v,8)

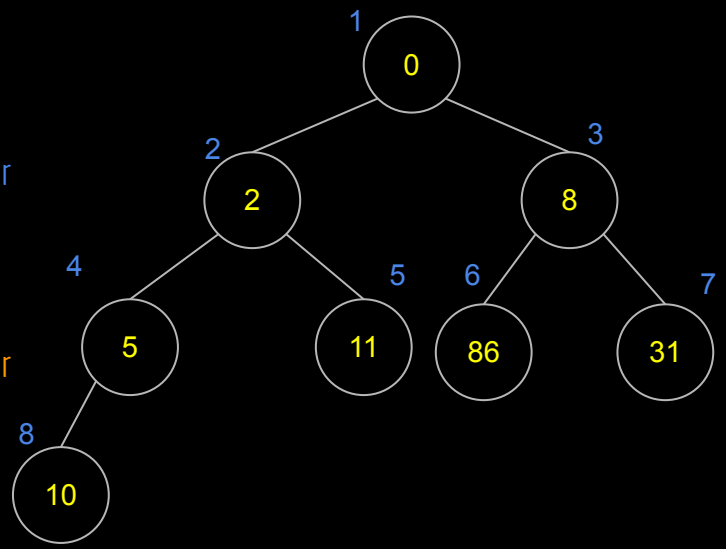
n	i
8	1

min-heapify(v,1,8)

n	i	l	r	menor
8	1	2	3	2

min-heapify(v,2,8)

n	i	l	r	menor
8	2	4	5	4

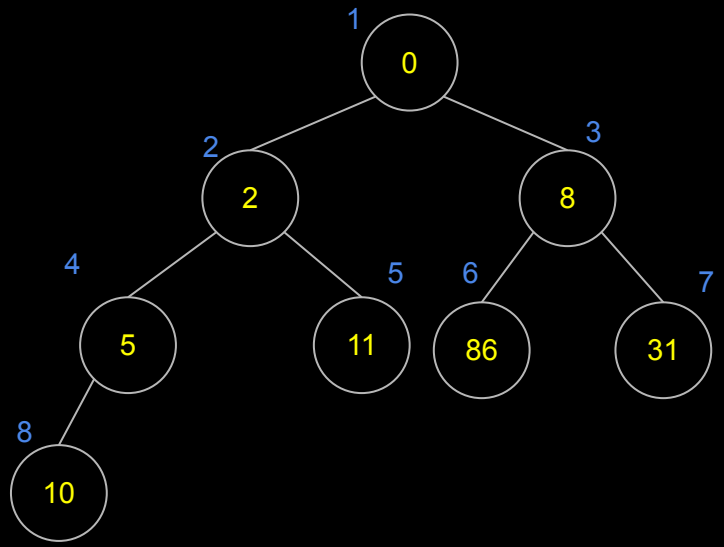


R_1	R_2	R_3	R_4	R_5	R_6	R_7	R_8
5	10	31	0	11	86	8	2
58	22	34	21	27	90	12	3
62	53	39	55	41	96	63	77
70	76	73	71	60	97	84	85

Custo

Min-Heap construída.

Qual foi o custo?



5	10	31	0	11	86	8	2
58	22	34	21	27	90	12	3
62	53	39	55	41	96	63	77
70	76	73	71	60	97	84	85

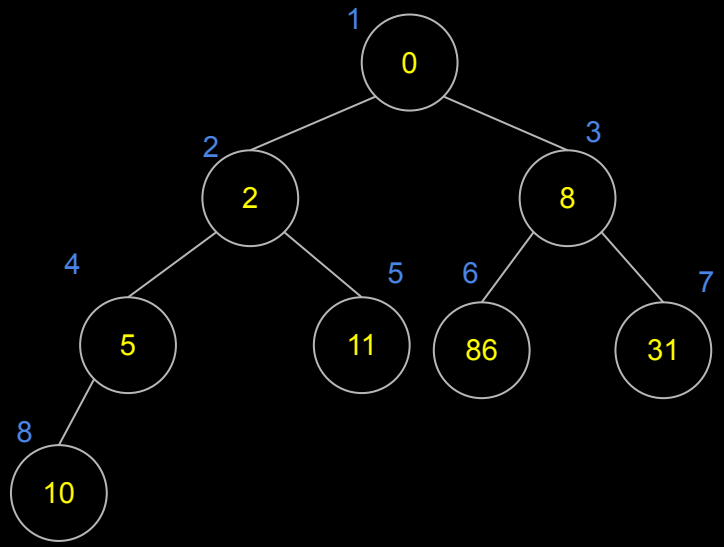
R_1 R_2 R_3 R_4 R_5 R_6 R_7 R_8

Custo

Min-Heap construída.

Qual foi o custo?

Custou P para copiar a cabeça de cada R_x para a heap,
+ P para construir a Min-Heap.
Logo, $C(P) = 2P$.



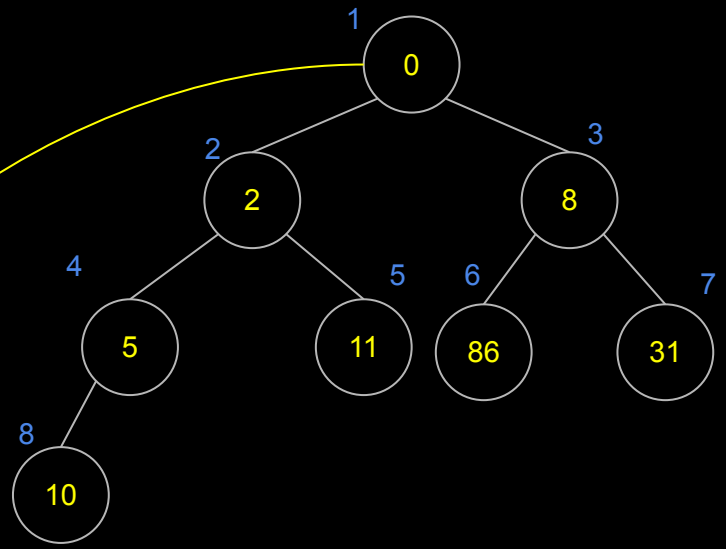
5	10	31	0	11	86	8	2
58	22	34	21	27	90	12	3
62	53	39	55	41	96	63	77
70	76	73	71	60	97	84	85
← R_1	← R_2	← R_3	← R_4	← R_5	← R_6	← R_7	← R_8

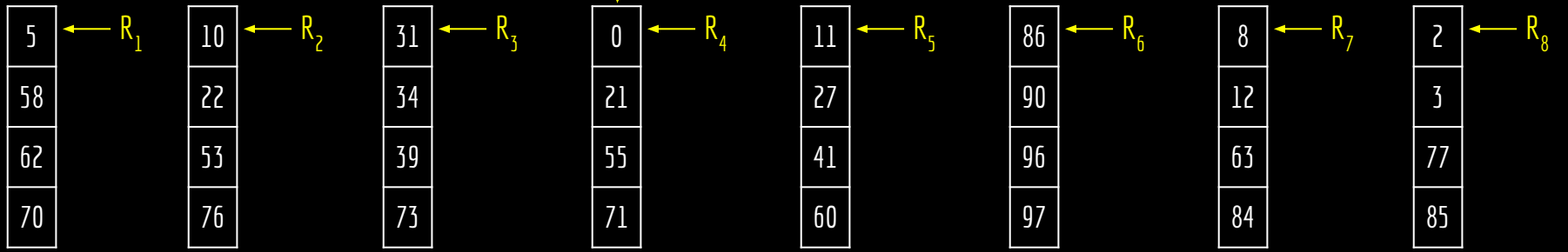
E agora?

Cada nodo da Min-Heap aponta para o seu arquivo.

Copiar a cabeça da Min-Heap para o arquivo final, mover o ponteiro do arquivo R_x correspondente, copiar a nova cabeça de R_x para a cabeça da Min-Heap, e chamar Min-Heapify com um custo $O(log_2 P)$.

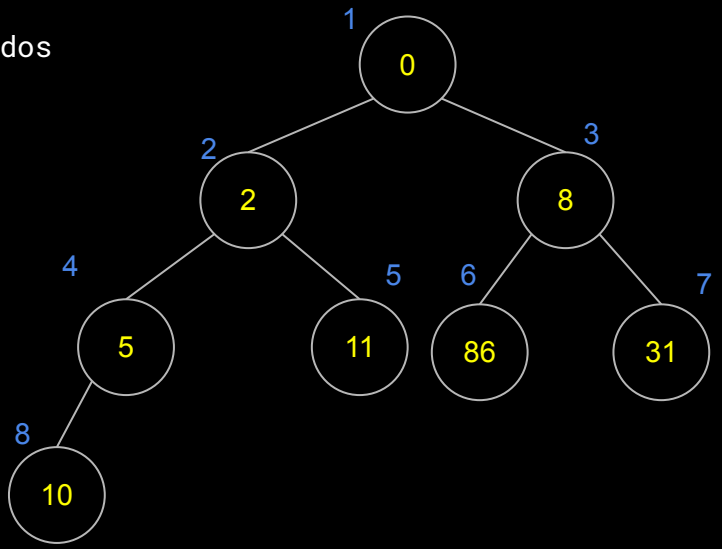
Repetir o processo.





E agora?

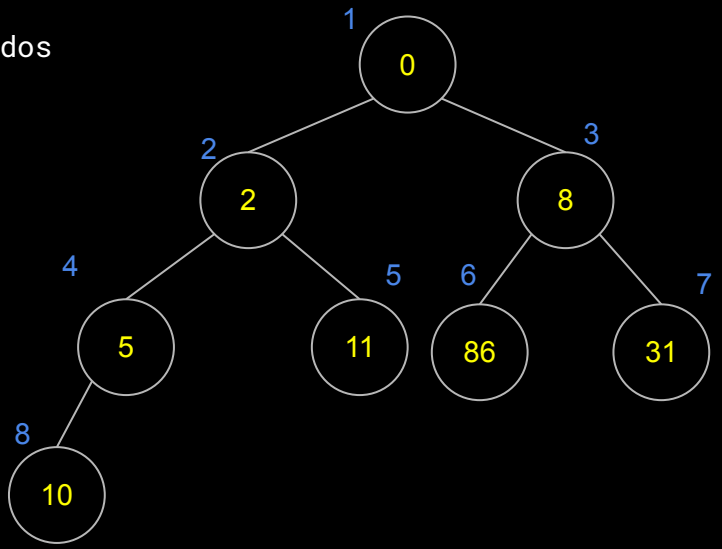
Copiar dado da cabeça da heap
Ajustar Ponteiros
Substituir cabeça da heap
Min-Heapify
repetir enquanto existirem dados

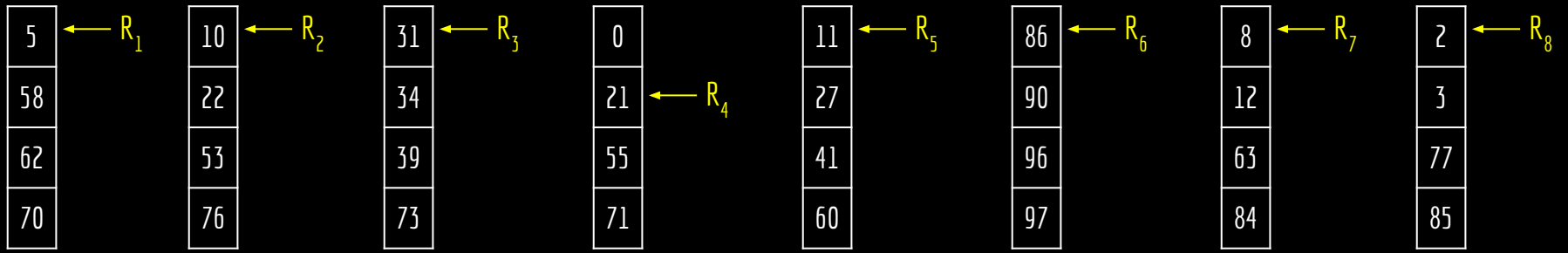


5 ← R_1	10 ← R_2	31 ← R_3	0 ← R_4	11 ← R_5	86 ← R_6	8 ← R_7	2 ← R_8
58	22	34	21	27	90	12	3
62	53	39	55	41	96	63	77
70	76	73	71	60	97	84	85

E agora?

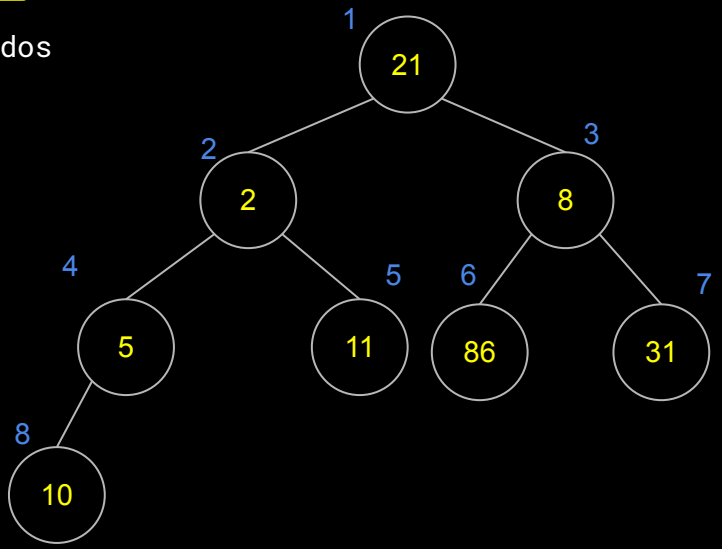
Copiar dado da cabeça da heap
Ajustar Ponteiros
Substituir cabeça da heap
Min-Heapify
repetir enquanto existirem dados

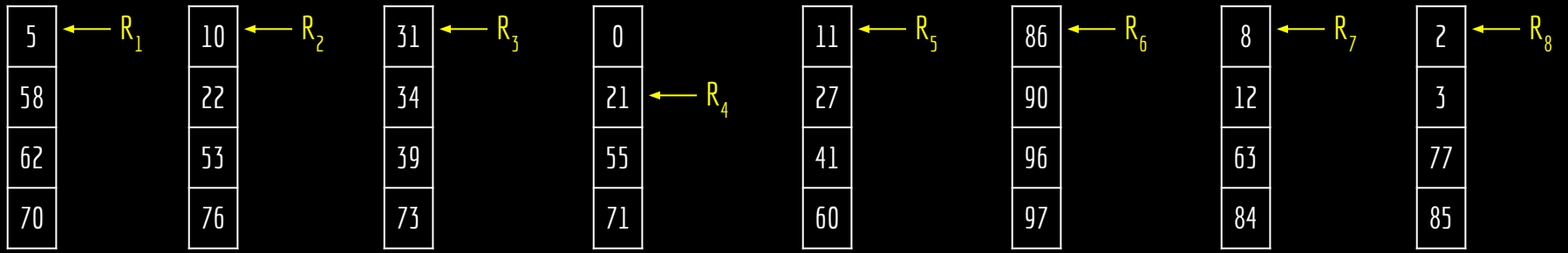




E agora?

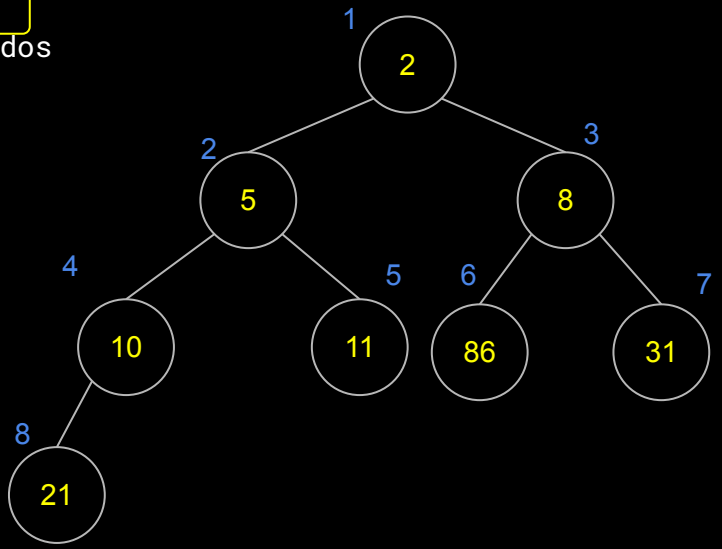
Copiar dado da cabeça da heap
Ajustar Ponteiros
Substituir cabeça da heap
Min-Heapify
repetir enquanto existirem dados

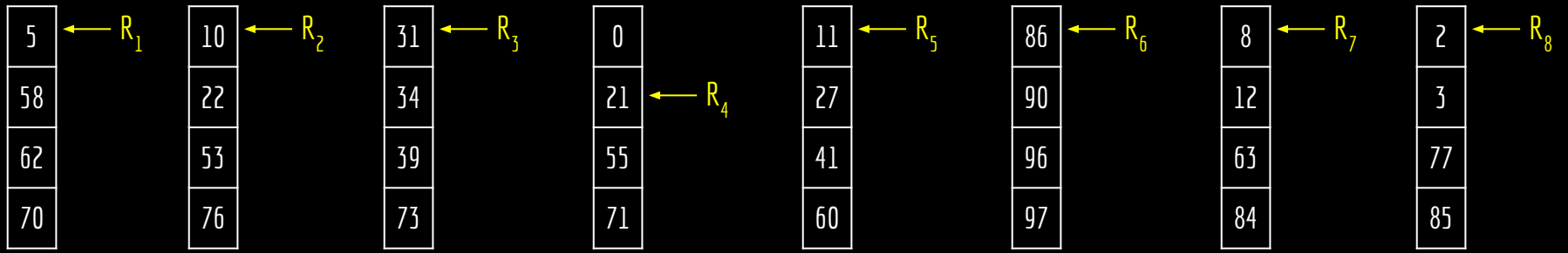




E agora?

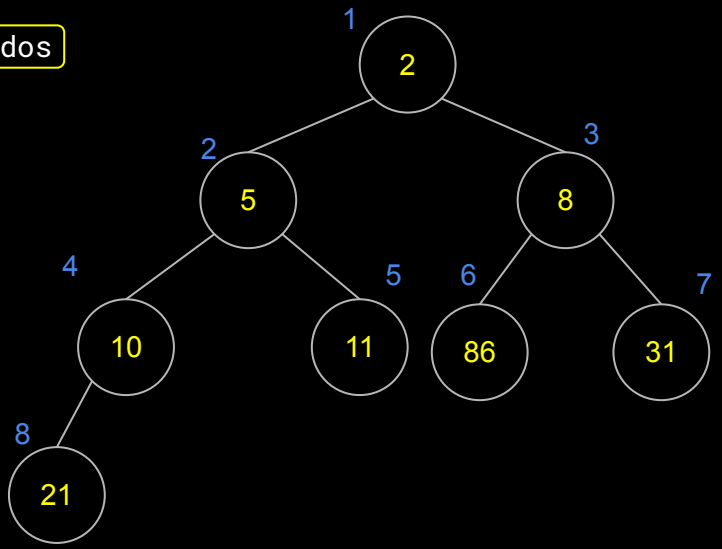
Copiar dado da cabeça da heap
Ajustar Ponteiros
Substituir cabeça da heap
Min-Heapify
repetir enquanto existirem dados





E agora?

Copiar dado da cabeça da heap
Ajustar Ponteiros
Substituir cabeça da heap
Min-Heapify
repetir enquanto existirem dados



5 ← R_1	10 ← R_2	31 ← R_3	0	11 ← R_5	86 ← R_6	8 ← R_7	2 ← R_8
58	22	34	21 ← R_4	27	90	12	3
62	53	39	55	41	96	63	77
70	76	73	71	60	97	84	85

Observações

A ordenação externa vista em aula está disponível em Knuth (1998).

- No livro é discutido que podemos usar uma Árvore de Seleção no lugar da Heap.
  - Mais eficiente quando as chaves são estruturas grandes.

Existem diversos outros algoritmos e formas de se fazer a ordenação externa.

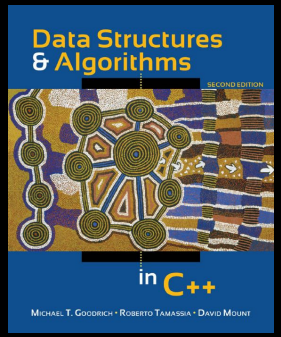
Exemplos:

- Usar diretamente um Merge Sort.
- Merge Polifásico.
- Quicksort externo.

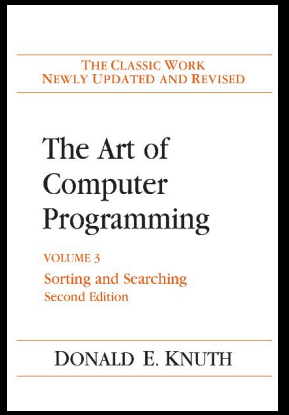
Exercícios

1. Implemente os algoritmos apresentados em C.

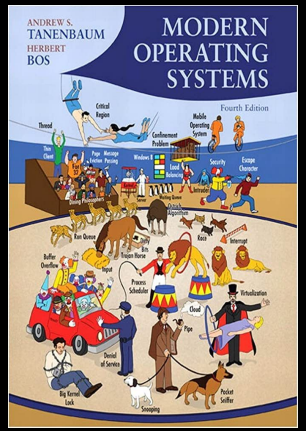
Referências



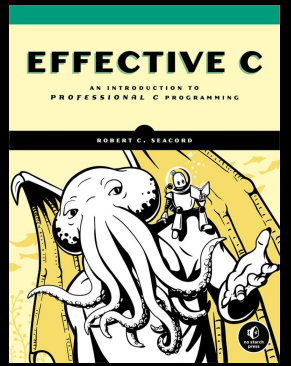
Mount, Goodrich, Tamassia. Data Structures and Algorithms in C++, 2011.



Knuth, D. The Art of Computer Programming: Volume 3: Sorting and Searching. 1998.



Tanenbaum, Bos. Sistemas Operacionais Modernos. 4a Ed. 2016.



Seacord, R. C. Effective C: An ntroduction to Professional C Programming. 2020.

Licença