

Fila de Prioridades

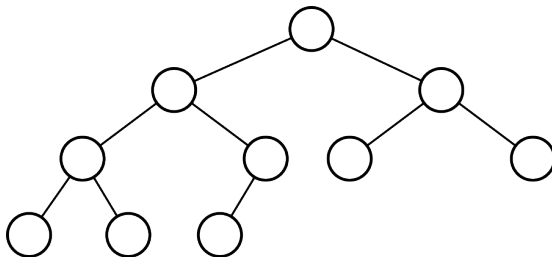
Prof^a. Rose Yuri Shimizu

Fila de Prioridades - TAD

- É um tipo abstrato de dados (TAD)
 - ▶ Estrutura de dados com comportamentos específicos
 - ▶ Acessadas por um conjunto de operações (interface)
- Existem operações que envolvem um grande volume de informações que precisam de alguma ordenação
 - ▶ Não necessariamente precisam estar totalmente ordenados
 - ★ O importante é saber qual tem a maior prioridade
 - ▶ Não necessariamente precisam processar todos os dados
 - ★ Conforme novos dados forem coletados, atualiza-se a fila de prioridades
 - ▶ Dados que são rankeados conforme um critério em que o mais importante é saber quem está no topo
 - ▶ Exemplos: mineração de dados, caminhos em grafos (verificar adjacentes na decisão do caminho)
- A fila pode ser com prioridade máxima (maior chave, maior prioridade) ou mínima (menor valor, maior prioridade)

Fila de Prioridades - Heap

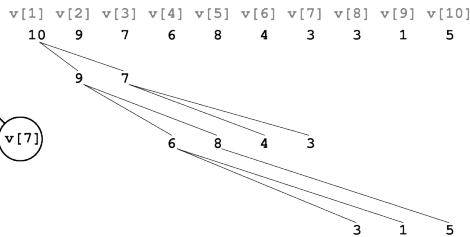
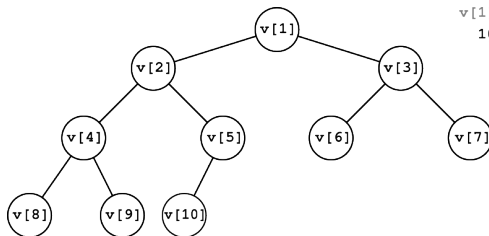
- Forma uma árvore binária quase completa, com as seguintes características:
 - ▶ Todos as folhas estão no nível “d” ou “d-1”
 - ▶ Todos os níveis exceto o último estão cheios
 - ▶ Os nós do último nível estão o mais a esquerda possível



- Raiz: chave de maior prioridade
- Não ordena por completo, só garante-se que:
 - ▶ Quanto mais próximo à raiz, maior a prioridade
 - ▶ Cada nó possui filhos com valores menores ou iguais

Fila de Prioridades - Heap

- Representada por um vetor:
 - ▶ Eficiente para as operações básicas (logarítmico) da fila de prioridades
 - ▶ Representação sequencial da árvore: facilidade em deixá-la completa
 - ▶ Acesso direto aos nós
 - ▶ Níveis da árvore acessada pelos seus índices
 - ★ Raiz: posição 1
 - ★ Filhos: 2 e 3
 - ★ Netos: 4, 5, 6 e 7
 - ★ E assim por diante.



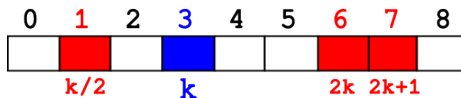
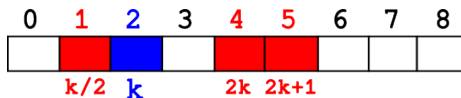
Fila de Prioridades - Heap

- Navegação trivial para cima e baixo:

- ▶ Simples operação aritmética
- ▶ Sendo um nó na posição k

- ★ pai: $\lfloor \frac{k}{2} \rfloor$

- ★ filhos: $2k$ e $2k + 1$



- Tamanho N em um vetor $pq[]$

- ▶ $pq[N+1]$: $pq[1..N]$
- ▶ Não utiliza-se a posição $pq[0]$ (??)
- ▶ E se utilizar?

- ★ pai: $\lfloor \frac{k-1}{2} \rfloor$

- ★ filhos: $2k + 1$ e $2k + 2$

Fila de Prioridades - Heap

- Interface (manipulação da fila):
 - ▶ PQinit(int maxN): criar uma fila de prioridades com capacidade máxima inicial
 - ▶ PQempty(): testar se está vazia
 - ▶ PQinsert(Item v): inserir uma chave
 - ▶ PQdelmax(): retornar e remover (maior prioridade)
 - ▶ Busca??

Fila de Prioridades - Heap

```
1 #define less(A, B) ((A) < (B))
2 #define exch(A, B) { Item t=A; A=B; B=t; }
3
4 //static: acessível somente no arquivo
5 static Item *pq;
6 static int N;
7
8 void PQinit(int maxN) {
9     pq = malloc(sizeof(Item)*(maxN+1));
10    N = 0;
11 }
12
13 int PQempty() {
14     return N==0;
15 }
```

Fila de Prioridades - Heap - Inserção

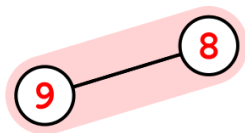
- Inserção nas folhas da heap
- Restauração/conserto: subindo na heap (maior que o pai?)



Inserir 8 [8]

Fila de Prioridades - Heap - Inserção

- Inserção nas folhas da heap
- Restauração/conserto: subindo na heap (maior que o pai?)

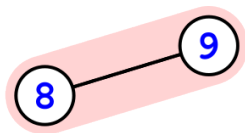


Inserir 9

[8, 9]

Fila de Prioridades - Heap - Inserção

- Inserção nas folhas da heap
- Restauração/conserto: subindo na heap (maior que o pai?)

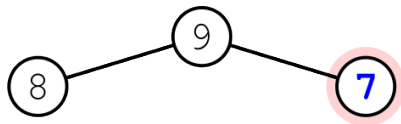


Restaurando 9

[9, 8]

Fila de Prioridades - Heap - Inserção

- Inserção nas folhas da heap
- Restauração/conserto: subindo na heap (maior que o pai?)

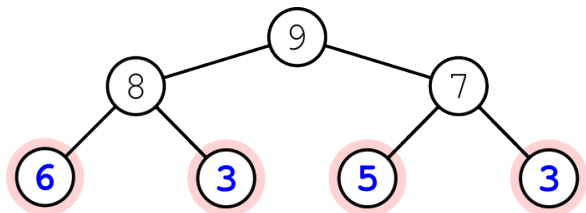


Inserir 7

[9, 8, 7]

Fila de Prioridades - Heap - Inserção

- Inserção nas folhas da heap
- Restauração/conserto: subindo na heap (maior que o pai?)

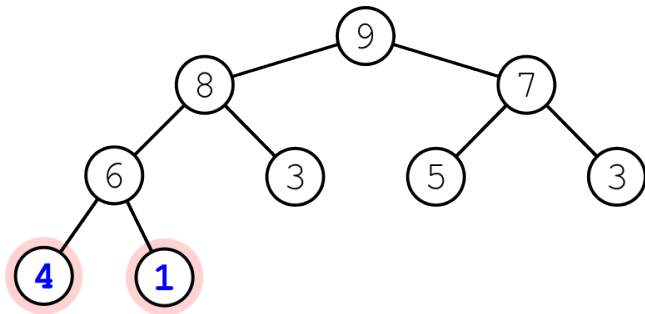


Inserir 6,3,5,3

[9, 8, 7, 6, 3, 5, 3]

Fila de Prioridades - Heap - Inserção

- Inserção nas folhas da heap
- Restauração/conserto: subindo na heap (maior que o pai?)

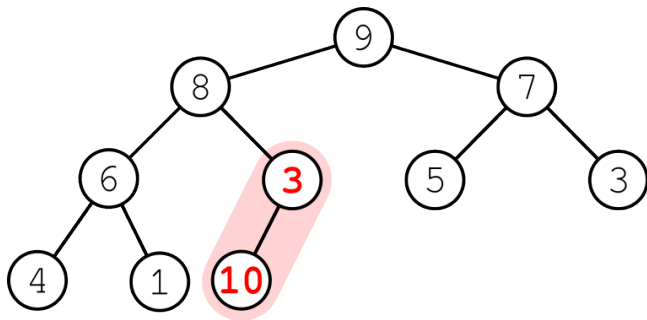


Inserir 4,1

[9, 8, 7, 6, 3, 5, 3, 4, 1]

Fila de Prioridades - Heap - Inserção

- Inserção nas folhas da heap
- Restauração/conserto: subindo na heap (maior que o pai?)

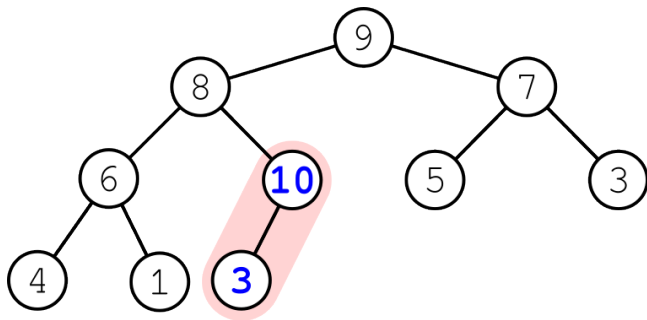


Inserir 10

[9, 8, 7, 6, 3, 5, 3, 4, 1, 10]

Fila de Prioridades - Heap - Inserção

- Inserção nas folhas da heap
- Restauração/conserto: subindo na heap (maior que o pai?)

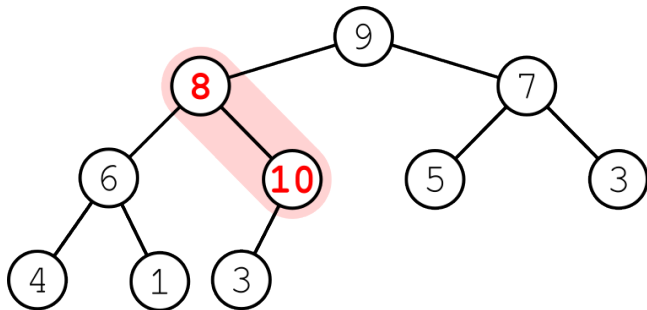


Restaurando 10

[9, 8, 7, 6, 3, 5, 3, 4, 1, 10]

Fila de Prioridades - Heap - Inserção

- Inserção nas folhas da heap
- Restauração/conserto: subindo na heap (maior que o pai?)

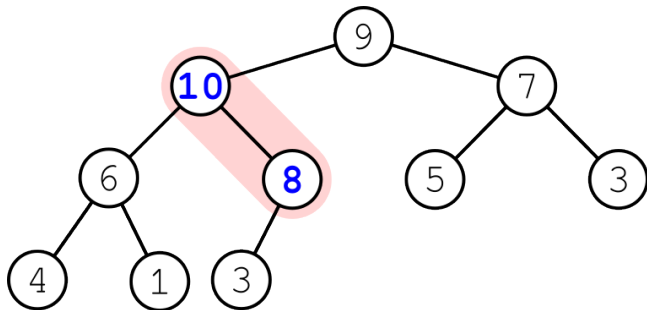


Restaurando 10

[9, 8, 7, 6, 10, 5, 3, 4, 1, 3]

Fila de Prioridades - Heap - Inserção

- Inserção nas folhas da heap
- Restauração/conserto: subindo na heap (maior que o pai?)

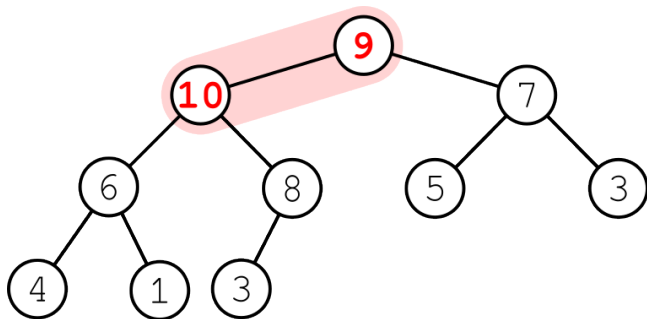


Restaurando 10

[9, 8, 7, 6, 10, 5, 3, 4, 1, 3]

Fila de Prioridades - Heap - Inserção

- Inserção nas folhas da heap
- Restauração/conserto: subindo na heap (maior que o pai?)

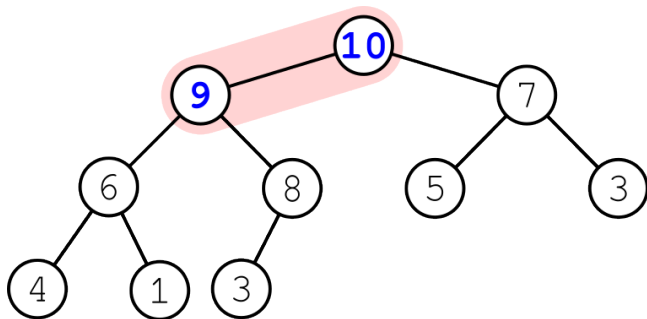


Restaurando 10

[9, 10, 7, 6, 8, 5, 3, 4, 1, 3]

Fila de Prioridades - Heap - Inserção

- Inserção nas folhas da heap
- Restauração/conserto: subindo na heap (maior que o pai?)

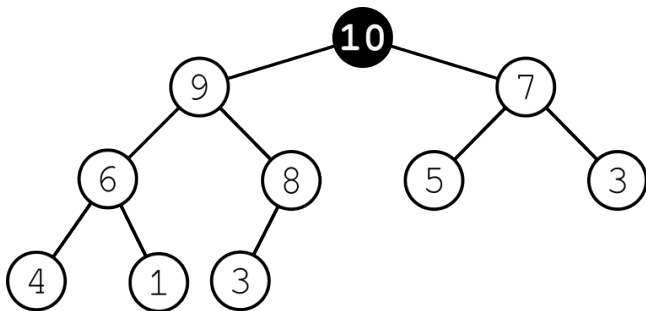


Restaurando 10

[10, 9, 7, 6, 8, 5, 3, 4, 1, 3]

Fila de Prioridades - Heap - Inserção

- Inserção nas folhas da heap
- Restauração/conserto: subindo na heap (maior que o pai?)



Fila de Prioridades - Heap - Inserção

- Adicionar uma nova chave no fim do vetor

```
1 void PQinsert(Item v) {  
2     //adicionar no fim
```

Fila de Prioridades - Heap - Inserção

- Adicionar uma nova chave no fim do vetor

```
1 void PQinsert(Item v) {  
2     //adicionar no fim  
3     pq[++N] = v;  
4  
5     //consertar o elemento na posição N  
6     fixUp(N);  
7 }
```

Fila de Prioridades - Heap - Inserção

- Restaura a ordenação da heap: bottom-up (swim - fixUp)
 - ▶ Flutue (swap) caso a chave seja maior que seu pai
 - ▶ Repetidamente, flutue até pai maior ou raiz

```
1 void fixUp(int k)
2 {
3     //se não for raiz e maior que o pai
```

Fila de Prioridades - Heap - Inserção

- Restaura a ordenação da heap: bottom-up (swim - fixUp)
 - ▶ Flutue (swap) caso a chave seja maior que seu pai
 - ▶ Repetidamente, flutue até pai maior ou raiz

```
1 void fixUp(int k)
2 {
3     //se não for raiz e maior que o pai
4     while(k>1 &&
```


Fila de Prioridades - Heap - Inserção

- Restaura a ordenação da heap: bottom-up (swim - fixUp)
 - ▶ Flutue (swap) caso a chave seja maior que seu pai
 - ▶ Repetidamente, flutue até pai maior ou raiz

```
1 void fixUp(int k)
2 {
3     //se não for raiz e maior que o pai
4     while(k>1 && less(pq[k/2],pq[k]))
5     {
6         //troque
```

Fila de Prioridades - Heap - Inserção

- Restaura a ordenação da heap: bottom-up (swim - fixUp)
 - ▶ Flutue (swap) caso a chave seja maior que seu pai
 - ▶ Repetidamente, flutue até pai maior ou raiz

```
1 void fixUp(int k)
2 {
3     //se não for raiz e maior que o pai
4     while(k>1 && less(pq[k/2],pq[k]))
5     {
6         //troque
7         exch(pq[k], pq[k/2]);
8
9         //subir: atualizar k
```

Fila de Prioridades - Heap - Inserção

- Restaura a ordenação da heap: bottom-up (swim - fixUp)
 - ▶ Flutue (swap) caso a chave seja maior que seu pai
 - ▶ Repetidamente, flutue até pai maior ou raiz

```
1 void fixUp(int k)
2 {
3     //se não for raiz e maior que o pai
4     while(k>1 && less(pq[k/2],pq[k]))
5     {
6         //troque
7         exch(pq[k], pq[k/2]);
8
9         //subir: atualizar k
10        k = k/2;
11    }
12 }
```

Fila de Prioridades - Heap - Inserção

- Complexidade: $1 + \log N$ comparações - $O(\log N)$

```
1 void fixUp(int k)
2 {
3     //k até 1 - reduzindo metade por iteração
4     //altura da árvore  $\approx \log k$ 
5     while(k > 1 && less(pq[k/2], pq[k]))
6     {
7         exch(pq[k], pq[k/2]);
8         k = k/2;
9     }
10 }
```

Fila de Prioridades - Heap - Remoção

- 1 Remover qual elemento??

Fila de Prioridades - Heap - Remoção

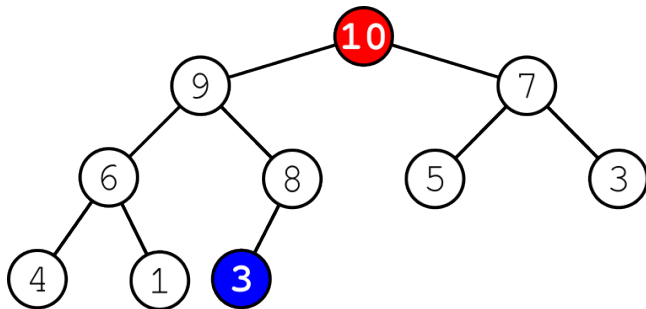
- 1 Remover qual elemento??
- 2 Substituir a raiz por uma folha

Fila de Prioridades - Heap - Remoção

- 1 Remover qual elemento??
- 2 Substituir a raiz por uma folha
- 3 Restauração/conserto: descendo na heap

Fila de Prioridades - Heap - Remoção

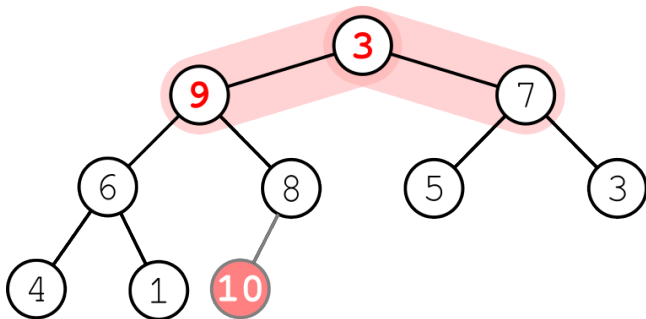
Substituir a raiz por uma folha



Remover 10 [**10**, 9, 7, 6, 8, 5, 3, 4, 1, **3**]

Fila de Prioridades - Heap - Remoção

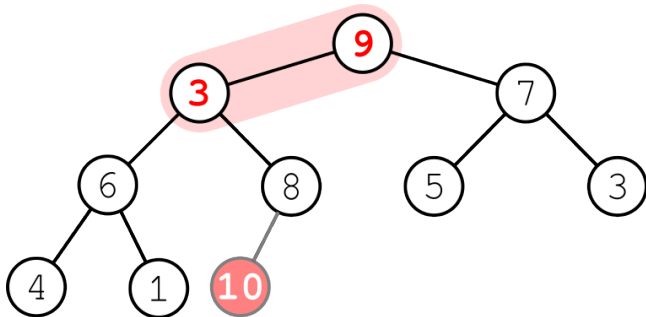
Restauração/conserto: descendo na heap (maior filho?)



Restaurando 3 [3, 9, 7, 6, 8, 5, 3, 4, 1, 10]

Fila de Prioridades - Heap - Remoção

Restauração/conserto: descendo na heap (maior filho?)

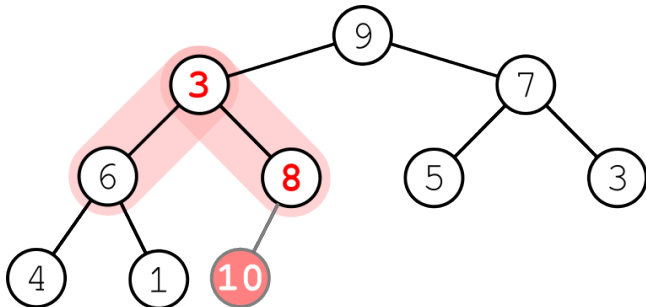


Restaurando 3

[9, 3, 7, 6, 8, 5, 3, 4, 1, 10]

Fila de Prioridades - Heap - Remoção

Restauração/conserto: descendo na heap (maior filho?)

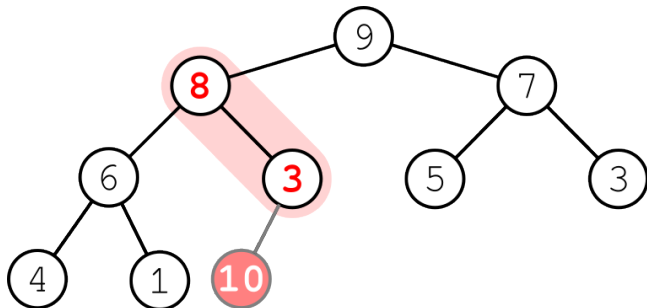


Restaurando 3

[9, 3, 7, 6, 8, 5, 3, 4, 1, 10]

Fila de Prioridades - Heap - Remoção

Restauração/conserto: descendo na heap (maior filho?)

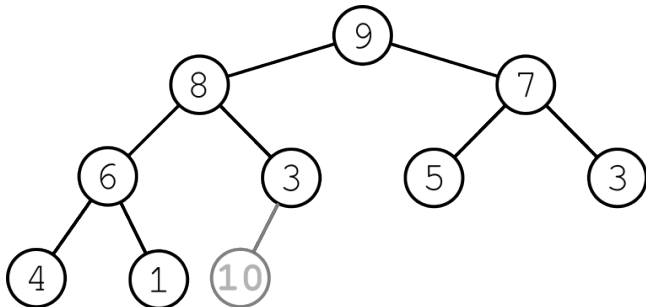


Restaurando 3

[9, **8**, 7, 6, **3**, 5, 3, 4, 1, 10]

Fila de Prioridades - Heap - Remoção

Restauração/conserto: descendo na heap (maior filho?)



Restaurando 3

[9, 8, 7, 6, 3, 5, 3, 4, 1, 10]

Fila de Prioridades - Heap - Remoção

- Troca a raiz com o último elemento

```
1 Item PQdelmax() {  
2     //troque topo -> ultimo
```

Fila de Prioridades - Heap - Remoção

- Troca a raiz com o último elemento

```
1 Item PQdelmax() {  
2     //troque topo -> ultimo  
3     exch(pq[1], pq[N]);  
4  
5     //reposicione
```

Fila de Prioridades - Heap - Remoção

- Troca a raiz com o último elemento

```
1 Item PQdelmax() {  
2     //troque topo -> ultimo  
3     exch(pq[1], pq[N]);  
4  
5     //reposicione  
6     fixDown(1, N-1);  
7  
8     //retorne o removido
```


Fila de Prioridades - Heap - Remoção

- Troca a raiz com o último elemento

```
1 Item PQdelmax() {  
2     //troque topo -> ultimo  
3     exch(pq[1], pq[N]);  
4  
5     //reposicione  
6     fixDown(1, N-1);  
7  
8     //retorne o removido  
9     return pq[N--];  
10 }
```

- Restaura a ordenação da heap: top-down (sink - fixDown)
 - ▶ Afunde caso a chave seja menor que os filhos
 - ★ Swap com o maior filho
 - ▶ Repetidamente, afunde até ser maior ou igual que os filhos ou ser folha

```
1 void fixDown(int k, int N) { //afunda k
2   //enquanto tive filho (?)
```

- Restaura a ordenação da heap: top-down (sink - fixDown)
 - ▶ Afunde caso a chave seja menor que os filhos
 - ★ Swap com o maior filho
 - ▶ Repetidamente, afunde até ser maior ou igual que os filhos ou ser folha

```
1 void fixDown(int k, int N) { //afunda k
2     //enquanto tive filho (?)
3     while(2*k <= N) {
4         int j = 2*k; //filho da esquerda
5
6         //se tiver filho da direita e for maior?
```

- Restaura a ordenação da heap: top-down (sink - fixDown)
 - ▶ Afunde caso a chave seja menor que os filhos
 - ★ Swap com o maior filho
 - ▶ Repetidamente, afunde até ser maior ou igual que os filhos ou ser folha

```
1 void fixDown(int k, int N) { //afunda k
2     //enquanto tive filho (?)
3     while(2*k <= N) {
4         int j = 2*k; //filho da esquerda
5
6         //se tiver filho da direita e for maior?
7         if(j<N && less(pq[j], pq[j+1])) j++;
8
9         //pq[k] maior que o maior filho?
```

- Restaura a ordenação da heap: top-down (sink - fixDown)
 - ▶ Afunde caso a chave seja menor que os filhos
 - ★ Swap com o maior filho
 - ▶ Repetidamente, afunde até ser maior ou igual que os filhos ou ser folha

```
1 void fixDown(int k, int N) { //afunda k
2     //enquanto tive filho (?)
3     while(2*k <= N) {
4         int j = 2*k; //filho da esquerda
5
6         //se tiver filho da direita e for maior?
7         if(j<N && less(pq[j], pq[j+1])) j++;
8
9         //pq[k] maior que o maior filho?
10        if(!less(pq[k], pq[j])) break;
11
12        //senão, afunde (troque com o filho)
```

- Restaura a ordenação da heap: top-down (sink - fixDown)
 - ▶ Afunde caso a chave seja menor que os filhos
 - ★ Swap com o maior filho
 - ▶ Repetidamente, afunde até ser maior ou igual que os filhos ou ser folha

```
1 void fixDown(int k, int N) { //afunda k
2     //enquanto tive filho (?)
3     while(2*k <= N) {
4         int j = 2*k; //filho da esquerda
5
6         //se tiver filho da direita e for maior?
7         if(j<N && less(pq[j], pq[j+1])) j++;
8
9         //pq[k] maior que o maior filho?
10        if(!less(pq[k], pq[j])) break;
11
12        //senão, afunde (troque com o filho)
13        exch(pq[k], pq[j]);
14
15        //atualiza k para o maior filho
```

- Restaura a ordenação da heap: top-down (sink - fixDown)
 - ▶ Afunde caso a chave seja menor que os filhos
 - ★ Swap com o maior filho
 - ▶ Repetidamente, afunde até ser maior ou igual que os filhos ou ser folha

```
1 void fixDown(int k, int N) { //afunda k
2     //enquanto tive filho (?)
3     while(2*k <= N) {
4         int j = 2*k; //filho da esquerda
5
6         //se tiver filho da direita e for maior?
7         if(j<N && less(pq[j], pq[j+1])) j++;
8
9         //pq[k] maior que o maior filho?
10        if(!less(pq[k], pq[j])) break;
11
12        //senão, afunde (troque com o filho)
13        exch(pq[k], pq[j]);
14
15        //atualiza k para o maior filho
16        k = j;
17    }
18 }
```

Fila de Prioridades - Heap - Remoção

- Complexidade: $2 \log N$ comparações - $O(\log N)$

```
1 void fixDown(int k, int N) {  
2     int j;  
3     //2*k até N - dobrando a cada iteração  
4     //altura da árvore  $\approx \log k$   
5     while(2*k <= N) {  
6         j = 2*k;  
7         if(j < N && less(pq[j], pq[j+1])) j++; //1  
8         if(!less(pq[k], pq[j])) break; //1  
9         exch(pq[k], pq[j]);  
10        k = j;  
11    }  
12 }
```


Fila de Prioridades - Heap - Várias filas

```
1  /*****  
2  /* Implementacao com array */  
3  /* Varias filas          */  
4  *****/  
5  typedef int Item;  
6  typedef struct {  
7      Item *pq;  
8      int N;  
9  }PQueue;  
10  
11 PQueue *PQinit(int);  
12 int PQempty(PQueue*);  
13 void PQinsert(PQueue*, Item);  
14 Item PQdelmax(PQueue*);  
15  
16 void fixUp(PQueue*, int);  
17 void fixDown(PQueue*, int);
```

Fila de Prioridades - Heap - Alterar prioridade

- Se temos o índice na fila de prioridades é trivial

```
1 void PQchange(int k, int valor)
2 {
3     v[k] = valor;
4     fixUp(k);
5     fixDown(k, N);
6 }
```

Fila de Prioridades - Heap - Alterar prioridade

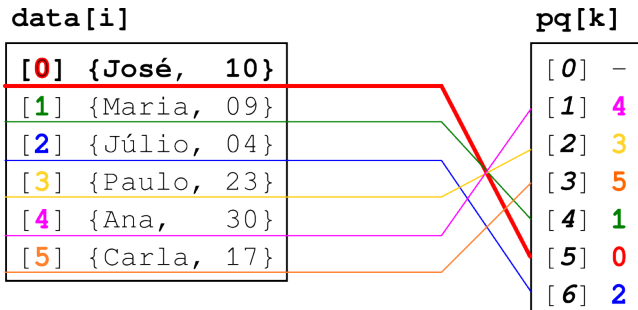
- Se não tem o acesso direto?
- Alterar a chave de valor 10 por 50?

pq[k]

[0]	—
[1]	30
[2]	23
[3]	17
[4]	09
[5]	10
[6]	04

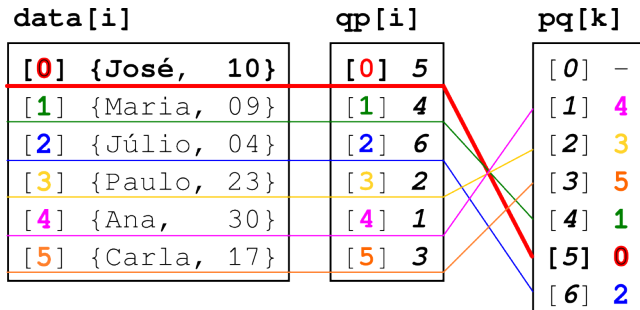
Fila de Prioridades - Heap - Alterar prioridade

- Base de dados: `data[i]`
- Fila de prioridades: posições da base de dados
 - ▶ `pq[k] = i`
 - ▶ `i` o índice em `data[i]`
 - ▶ `k` sua prioridade (posição na fila)
- Se alterar algum dado de `data`?
 - ▶ Se `data[0].chave = 50`?



Fila de Prioridades - Heap - Lista de índices - qp[]

- Lista de posições de data em pq
- Sendo $qp[i] = k \leftrightarrow data[i]$ está em $pq[k]$
 - ▶ $pq[k] = i$
 - ▶ $pq[qp[i]] = i$



```

1 typedef struct {
2     char nome[20];
3     int chave;
4 } Item;
5
6 static int *pq;
7 static int *qp;
8 static int N;
9
10 void PQinit(int);
11 void PQinsert(int);
12 void PQchange(int);
13 int PQdelmax();
14
15 int main(){
16     Item data[6] = {"José", 10},
17                    {"Maria", 9},
18                    {"Júlio", 4},
19                    {"Paulo", 23},
20                    {"Ana", 30},
21                    {"Carla", 17}};
22
23     PQinit(6);
24     for(int i=0; i<6; i++) PQinsert(i);
25
26 }

```

```

1 void PQinit(int maxN)
2 {
3     pq = malloc(sizeof(int)*(maxN+1));
4     qp = malloc(sizeof(int)*(maxN+1));
5     N = 0;
6 }
7
8 //data[k]
9 void PQinsert(int k)
10 {
11     pq[++N] = k; //inserir na última posição
12     qp[k] = N;  //lista de índices
13
14     fixUp(N);    //consertar a heap
15                 //pq[N/2].chave < pq[N].chave
16 }

```

Fila de Prioridades - Heap - Alterar prioridade

- `data[0].chave = 50`
- Atualizar fila de prioridades:
 - ▶ `PQchange(0)`
 - ▶ Encontrar sua posição na fila através da lista
 - ★ `data[0] → qp[0]=5 → pq[qp[0]] → pq[5]=0`

<code>data[i]</code>	<code>qp[i]</code>	<code>pq[k]</code>
[0] {José, 50}	[0] 5	[0] -
[1] {Maria, 09}	[1] 4	[1] 4
[2] {Júlio, 04}	[2] 6	[2] 3
[3] {Paulo, 23}	[3] 2	[3] 5
[4] {Ana, 30}	[4] 1	[4] 1
[5] {Carla, 17}	[5] 3	[5] 0
		[6] 2

Fila de Prioridades - Heap - Alterar prioridade

- `data[0].chave = 50`
- Atualizar fila de prioridades:
 - ▶ `PQchange(0)`
 - ▶ Encontrar sua posição na fila através da lista
 - ★ `data[0] → qp[0]=5 → pq[qp[0]] → pq[5]=0`
 - ▶ Consertar a heap
 - ★ `fixUp(qp[0])`: motivo?
 - ★ `fixDown(qp[0], N)`: motivo?

<code>data[i]</code>	<code>qp[i]</code>	<code>pq[k]</code>
[0] {José, 50}	[0] 5	[0] -
[1] {Maria, 09}	[1] 4	[1] 4
[2] {Júlio, 04}	[2] 6	[2] 3
[3] {Paulo, 23}	[3] 2	[3] 5
[4] {Ana, 30}	[4] 1	[4] 1
[5] {Carla, 17}	[5] 3	[5] 0
		[6] 2

Fila de Prioridades - Heap - Alterar prioridade

- `data[0].chave = 50`
- Atualizar fila de prioridades:
 - ▶ `PQchange(0)`
 - ▶ Encontrar sua posição na fila através da lista
 - ★ `data[0] → qp[0]=5 → pq[qp[0]] → pq[5]=0`
 - ▶ Consertar a heap
 - ★ `fixUp(qp[0])`: motivo?
 - ★ `fixDown(qp[0], N)`: motivo?

<code>data[i]</code>	<code>qp[i]</code>	<code>pq[k]</code>
[0] {José, 50}	[0] 2	[0] -
[1] {Maria, 09}	[1] 4	[1] 4
[2] {Júlio, 04}	[2] 6	[2] 0
[3] {Paulo, 23}	[3] 5	[3] 5
[4] {Ana, 30}	[4] 1	[4] 1
[5] {Carla, 17}	[5] 3	[5] 3
		[6] 2

Fila de Prioridades - Heap - Alterar prioridade

- `data[0].chave = 50`
- Atualizar fila de prioridades:
 - ▶ `PQchange(0)`
 - ▶ Encontrar sua posição na fila através da lista
 - ★ `data[0] → qp[0]=5 → pq[qp[0]] → pq[5]=0`
 - ▶ Consertar a heap
 - ★ `fixUp(qp[0])`: motivo?
 - ★ `fixDown(qp[0], N)`: motivo?

<code>data[i]</code>	<code>qp[i]</code>	<code>pq[k]</code>
[0] {José, 50}	[0] 2	[0] -
[1] {Maria, 09}	[1] 4	[1] 4
[2] {Júlio, 04}	[2] 6	[2] 0
[3] {Paulo, 23}	[3] 5	[3] 5
[4] {Ana, 30}	[4] 1	[4] 1
[5] {Carla, 17}	[5] 3	[5] 3
		[6] 2

Fila de Prioridades - Heap - Alterar prioridade

- `data[0].chave = 50`
- Atualizar fila de prioridades:
 - ▶ `PQchange(0)`
 - ▶ Encontrar sua posição na fila através da lista
 - ★ `data[0] → qp[0]=5 → pq[qp[0]] → pq[5]=0`
 - ▶ Consertar a heap
 - ★ `fixUp(qp[0])`: motivo?
 - ★ `fixDown(qp[0], N)`: motivo?

<code>data[i]</code>	<code>qp[i]</code>	<code>pq[k]</code>
[0] {José, 50}	[0] 1	[0] -
[1] {Maria, 09}	[1] 4	[1] 0
[2] {Júlio, 04}	[2] 6	[2] 4
[3] {Paulo, 23}	[3] 5	[3] 5
[4] {Ana, 30}	[4] 2	[4] 1
[5] {Carla, 17}	[5] 3	[5] 3
		[6] 2

Fila de Prioridades - Heap - Alterar prioridade

- `data[0].chave = 50`
- Atualizar fila de prioridades:
 - ▶ `PQchange(0)`
 - ▶ Encontrar sua posição na fila através da lista
 - ★ `data[0] → qp[0]=5 → pq[qp[0]] → pq[5]=0`
 - ▶ Consertar a heap
 - ★ `fixUp(qp[0])`: motivo?
 - ★ `fixDown(qp[0], N)`: motivo?

<code>data[i]</code>	<code>qp[i]</code>	<code>pq[k]</code>
[0] {José, 50}	[0] 1	[0] -
[1] {Maria, 09}	[1] 4	[1] 0
[2] {Júlio, 04}	[2] 6	[2] 4
[3] {Paulo, 23}	[3] 5	[3] 5
[4] {Ana, 30}	[4] 2	[4] 1
[5] {Carla, 17}	[5] 3	[5] 3
		[6] 2

```

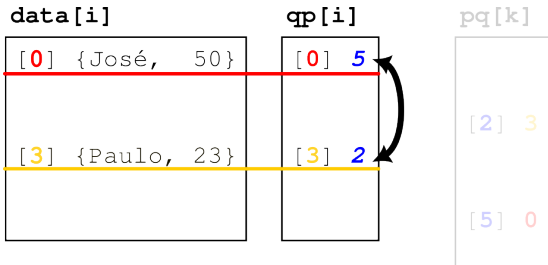
1 void PQchange(int i) {
2     //atualizar data[i] na fila
3     //data[i] está na posição qp[i]
4     fixUp(qp[i]);
5     fixDown(qp[i], N);
6 }
7
8 void fixUp(int k) {
9     while(k>1 && less(pq[k/2],pq[k])) comparação?
10    {
11        exch(pq[k], pq[k/2]); ALTERAÇÃO!!!!
12        k = k/2;
13    }
14 }
15
16 void fixDown(int k, int N) {
17     int j;
18     while(2*k <= N) {
19         j = 2*k;
20         if(j<N && less(pq[j], pq[j+1])) j++;
21         if(!less(pq[k], pq[j])) break; comparação?
22         exch(pq[k], pq[j]); ALTERAÇÃO!!!!
23         k = j;
24     }
25 }

```

```

1 //swap das posições p/ data[a] e data[b]
2 void exch(int a, int b) {
3     //atualizar lista de índices
4
5
6
7
8     //atualizar fila de prioridades
9
10
11 }

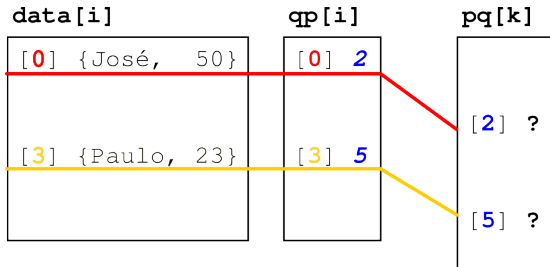
```



```

1 //swap das posições p/ data[a] e data[b]
2 void exch(int a, int b) {
3     //atualizar lista de índices
4     int k = qp[a];
5     qp[a] = qp[b]; //troca a posição: data[a] ↔ data[b]
6     qp[b] = k;     //troca a posição: data[b] ↔ data[a]
7
8     //atualizar fila de prioridades
9
10
11 }

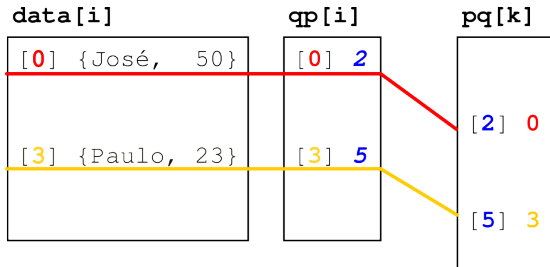
```




```

1 //swap das posições p/ data[a] e data[b]
2 void exch(int a, int b) {
3     //atualizar lista de índices
4     int k = qp[a];
5     qp[a] = qp[b];
6     qp[b] = k;
7
8     //atualizar fila de prioridades
9     pq[qp[a]] = a; //na fila pq, posição qp[a], está data[a]
10    pq[qp[b]] = b; //na fila pq, posição qp[b], está data[b]
11 }

```



Fila de Prioridades - Heap - Alterar prioridade

```
1 int main(){
2     Item data[6] = {"José", 10},
3                     {"Maria", 9},
4                     {"Júlio", 4},
5                     {"Paulo", 23},
6                     {"Ana", 30},
7                     {"Carla", 17}};
8
9     PQinit(6);
10    for(int i=0; i<6; i++) PQinsert(i);
11
12    data[0].chave = 50;
13    PQchange(0);
14
15    int k = PQdelmax();
16    printf("%d %s\n", data[k].chave, data[k].nome);
17 }
```