

# Filas de Prioridade

Estruturas de Dados e Algoritmos – Ciência da Computação



**INSTITUTO  
FEDERAL**  
Brasília

Prof. Daniel Saad Nogueira  
Nunes

IFB – Instituto Federal de Brasília,  
Campus Taguatinga



# Sumário

---

- 1 Introdução
- 2 Filas de Prioridade
- 3 Exemplos



# Sumário

---

## 1 Introdução



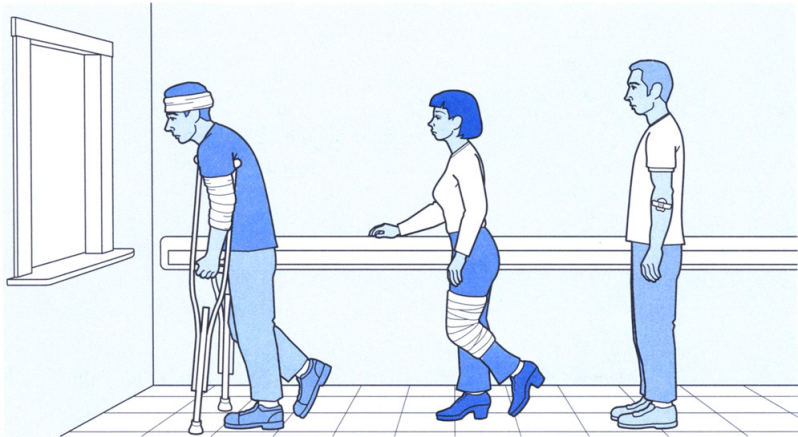
# Filas de Prioridade

---

- Filas de prioridade são TADs que também generalizam filas.
- Neste TAD, cada elemento tem sua prioridade.
- Os elementos com maior prioridade tem precedência sobre o menor, e portanto são retirados primeiro, independente da ordem de inserção.
- Vocês já viram isso na vida real?



# Filas de Prioridade





# Operações em Filas de Prioridade

---

- Algumas das operações suportadas por uma fila devem ser:
  - ▶ Enfileiramento de elementos;
  - ▶ Desenfileiramento de elementos com maior prioridade;
  - ▶ Verificar o elemento com maior prioridade;



# Operações em Filas de Prioridade

---

- Conhecemos alguma estrutura que realiza estas operações eficientemente?



# Operações em Filas de Prioridade

---

- Sim, uma **heap**!
- Mas temos que adaptá-la em sua versão dinâmica.
- Temos que usar vetores dinâmicos.





# Sumário

---

## 2 Filas de Prioridade



# Sumário

---

## 2 Filas de Prioridade

- Definição
- Inicialização
- Funções auxiliares
- Inserção
- Acesso ao elemento de maior prioridade
- Remoção
- Limpeza
- Análise



## Filas de Prioridade: Definição

---

```
7 typedef void* (*priority_queue_element_constructor_fn) (void*);  
8 typedef void (*priority_queue_element_destructor_fn)(void *);  
9 typedef int (*priority_queue_element_compare_fn)(const void*,const void*);
```



## Filas de Prioridade: Definição

---

```
12 typedef struct priority_queue_t{
13     void** data;
14     priority_queue_element_constructor_fn constructor;
15     priority_queue_element_destructor_fn destructor;
16     priority_queue_element_compare_fn comparator;
17     size_t size;
18     size_t capacity;
19 }priority_queue_t;
```



# Sumário

---

## 2 Filas de Prioridade

- Definição
- **Inicialização**
- Funções auxiliares
- Inserção
- Acesso ao elemento de maior prioridade
- Remoção
- Limpeza
- Análise



## Filas de Prioridade: Inicialização

---

```
42 void priority_queue_initialize(priority_queue_t** pq,
43                               priority_queue_element_constructor_fn constructor,
44                               priority_queue_element_destructor_fn destructor,
45                               priority_queue_element_compare_fn comparator){
46     (*pq) = malloc(sizeof(priority_queue_t));
47     (*pq)->data = NULL;
48     (*pq)->constructor = constructor;
49     (*pq)->destructor = destructor;
50     (*pq)->comparator = comparator;
51     (*pq)->size = 0;
52     (*pq)->capacity = 0;
53 }
```



# Sumário

---

## 2 Filas de Prioridade

- Definição
- Inicialização
- **Funções auxiliares**
- Inserção
- Acesso ao elemento de maior prioridade
- Remoção
- Limpeza
- Análise



## Filas de Prioridade: Inserção

---

```
97 size_t priority_queue_size(priority_queue_t* pq){  
98     return pq->size;  
99 }
```





## Filas de Prioridade: Inserção

---

```
101 size_t priority_queue_empty(priority_queue_t* pq){
102     return priority_queue_size(pq) == 0 ? 1: 0;
103 }
```



## Filas de Prioridade: Funções Auxiliares

---

```
6 static void priority_queue_heapify_bottom_up(priority_queue_t* pq, size_t i){
7     size_t p;
8     for(p=(i-1)/2; i!=0; p=(p-1)/2){
9         if(pq->comparator(pq->data[i], pq->data[p])<=0){
10             break;
11         }
12         void* aux = pq->data[i];
13         pq->data[i] = pq->data[p];
14         pq->data[p] = aux;
15         i = p;
16     }
17 }
```



## Filas de Prioridade: Funções Auxiliares

```
20 static void priority_queue_heapify_top_down(priority_queue_t* pq, size_t i){
21     size_t l,r;
22     size_t largest = i;
23     while(i < priority_queue_size(pq)){
24         i = largest;
25         l = 2*i + 1;
26         r = 2*i + 2;
27         if(l < priority_queue_size(pq) && pq->comparator(pq->data[i], pq->data[l]) <= 0){
28             largest = l;
29         }
30         if(r < priority_queue_size(pq) && pq->comparator(pq->data[largest], pq->data[r]) < 0){
31             largest = r;
32         }
33         if(largest == i){
34             break;
35         }
36         void* aux = pq->data[i];
37         pq->data[i] = pq->data[largest];
38         pq->data[largest] = aux;
39     }
40 }
```



# Sumário

---

## 2 Filas de Prioridade

- Definição
- Inicialização
- Funções auxiliares
- **Inserção**
- Acesso ao elemento de maior prioridade
- Remoção
- Limpeza
- Análise



# Filas de Prioridade: Inserção

---

## Inserção em Heap Dinâmica

- A inserção de um novo elemento é feito no final do vetor.
- Se o vetor não apresenta espaço suficiente, ele deverá ser relocado.
- A propriedade de Heap deve ser restaurada usando comparações debaixo para cima (bottom-up).
- O tamanho do vetor dinâmico aumenta de um.



## Filas de Prioridade: Inserção

```
63 void priority_queue_push(priority_queue_t* pq, void* data){
64     if(pq->size == pq->capacity){
65         if(pq->capacity == 0){
66             pq->capacity = 1;
67         }
68         else{
69             pq->capacity*=2;
70         }
71         pq->data = reallocx(pq->data, pq->capacity * sizeof(void*));
72     }
73     pq->data[pq->size] = pq->constructor(data);
74     priority_queue_heapify_bottom_up(pq, pq->size);
75     pq->size++;
76 }
```



# Sumário

---

## 2 Filas de Prioridade

- Definição
- Inicialização
- Funções auxiliares
- Inserção
- **Acesso ao elemento de maior prioridade**
- Remoção
- Limpeza
- Análise



## Filas de Prioridade: Acesso

---

### Consulta do Elemento de Maior Prioridade em Heap Dinâmica

- Pela propriedade de Heap, o elemento com maior prioridade ocupa a posição 0.
- Basta acessá-lo.





## Filas de Prioridade: Inserção

---

```
78 void* priority_queue_front(priority_queue_t* pq){
79     assert(!priority_queue_empty(pq));
80     return(pq->data[0]);
81 }
```



# Sumário

---

## 2 Filas de Prioridade

- Definição
- Inicialização
- Funções auxiliares
- Inserção
- Acesso ao elemento de maior prioridade
- **Remoção**
- Limpeza
- Análise



# Operações em Filas de Prioridade

---

## Remoção em Heap Dinâmica

- Pela propriedade de Heap, o elemento de maior prioridade ocupa a posição 0.
- A retirada é simulada ao colocar o elemento que ocupa a última posição do vetor na posição 0.
- A propriedade de heap deve ser restaurada ao utilizar comparações de cima para baixo (top-down).
- O tamanho do vetor dinâmico diminui de um.
- Se o tamanho for muito pequeno em comparação à área ocupada, o vetor dinâmico deverá ser relocado.



## Filas de Prioridade: Remoção

```
83 void priority_queue_pop(priority_queue_t* pq){
84     assert(!priority_queue_empty(pq));
85     pq->size--;
86     pq->destructor(pq->data[0]);
87     if(!priority_queue_empty(pq)){
88         pq->data[0] = pq->data[pq->size];
89         priority_queue_heapify_top_down(pq,0);
90     }
91     if(pq->size == pq->capacity/2){
92         pq->data = reallocx(pq->data,pq->capacity/2 * sizeof(void*));
93         pq->capacity = pq->capacity/2;
94     }
95 }
```



# Sumário

---

## 2 Filas de Prioridade

- Definição
- Inicialização
- Funções auxiliares
- Inserção
- Acesso ao elemento de maior prioridade
- Remoção
- **Limpeza**
- Análise



## Filas de Prioridade: Limpeza

---

```
55 void priority_queue_delete(priority_queue_t** pq){
56     while(!priority_queue_empty(*pq)){
57         priority_queue_pop(*pq);
58     }
59     free(*pq);
60     *pq = NULL;
61 }
```



# Sumário

---

## 2 Filas de Prioridade

- Definição
- Inicialização
- Funções auxiliares
- Inserção
- Acesso ao elemento de maior prioridade
- Remoção
- Limpeza
- Análise



## Operações em Filas de Prioridade

---

Operação	Complexidade
Consulta do elemento de maior prioridade	$\Theta(1)$
Remoção do elemento de maior prioridade	$\Theta(\lg n)$
Inserção de um novo elemento	$\Theta(\lg n)$





# Sumário

---

## 3 Exemplos



## Exemplo de Uso da Biblioteca

---

```
8 typedef struct aluno{
9     char nome[50];
10    double p1,p2,p3;
11 } aluno;
```



## Exemplo de Uso da Biblioteca

---

```
13 void* aluno_constructor(void* data){  
14     void* ptr = mallocx(sizeof(aluno));  
15     memcpy(ptr,data,sizeof(aluno));  
16     return ptr;  
17 }
```



## Exemplo de Uso da Biblioteca

---

```
19 void aluno_destructor(void* data){  
20     free(data);  
21 }
```



## Exemplo de Uso da Biblioteca

```
23 int aluno_comparator(const void* a,const void* b){
24     aluno *aluno1, *aluno2;
25     aluno1 = a;
26     aluno2 = b;
27     double media1,media2;
28     media1 = (aluno1->p1 + aluno1->p2 + aluno1->p3)/3.0;
29     media2 = (aluno2->p1 + aluno2->p2 + aluno2->p3)/3.0;
30     if(media1<media2){
31         return -1;
32     }
33     else if(media1>media2){
34         return 1;
35     }
36     if(strcmp(aluno1->nome,aluno2->nome) < 0){
37         return 1;
38     }
39     else if(strcmp(aluno1->nome,aluno2->nome) > 0){
40         return -1;
41     }
42     return 0;
43 }
```



## Exemplo de Uso da Biblioteca

```
47 int main(void){
48     aluno a;
49     aluno* aluno_ptr;
50     int nro_alunos = 4;
51     int i;
52     priority_queue_t* pq;
53     priority_queue_initialize(&pq,aluno_constructor,aluno_destructor,aluno_comparator);
54     for(i=1;i<=nro_alunos;i++){
55         printf("Aluno %d\n",i);
56         printf("Nome: ");
57         scanf("%s",a.nome);
58         printf("Nota P1: ");
59         scanf("%lf",&a.p1);
60         printf("Nota P2" );
61         scanf("%lf",&a.p2);
62         printf("Nota P3: ");
63         scanf("%lf",&a.p3);
64         priority_queue_push(pq,&a);
65     }
66
67     printf("\n\n");
```



## Exemplo de Uso da Biblioteca

---

```
69 while(!priority_queue_empty(pq)){
70     aluno_ptr = priority_queue_front(pq);
71     printf("Nome: %s\n",aluno_ptr->nome);
72     printf("P1: %lf\n",aluno_ptr->p1);
73     printf("P2: %lf\n",aluno_ptr->p2);
74     printf("P3: %lf\n",aluno_ptr->p3);
75     priority_queue_pop(pq);
76 }
77 priority_queue_delete(&pq);
78 return 0;
79 }
```