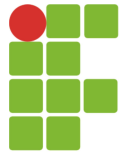
Filas de prioridade

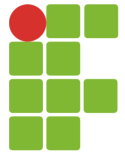
Estrutura de Dados e Algoritmos

Prof. Daniel Saad Nogueira Nunes

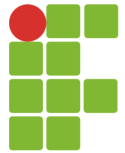
IFB – Instituto Federal de Brasília,
Campus Taguatinga



Sumário



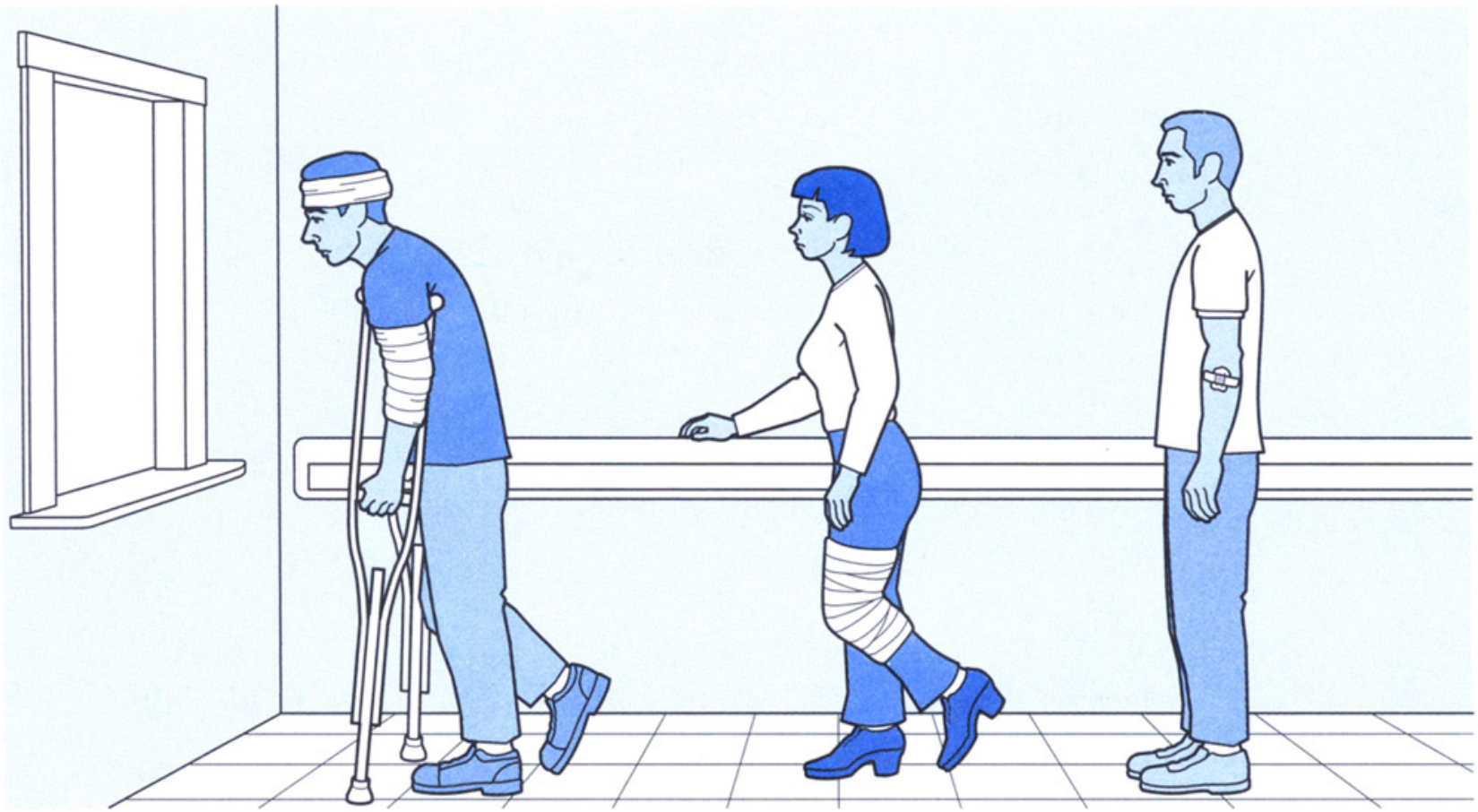
Sumário

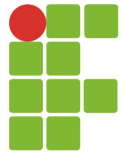


Filas de prioridade

- 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.

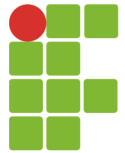
Filas de prioridade





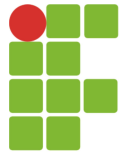
Operações em Filas de prioridade

- Algumas das operações suportadas por uma fila de prioridade devem ser:
  - Enfileiramento de elementos;
  - Desenfileiramento de elementos com maior prioridade;
  - Verificar o elemento com maior prioridade;
  - Obter o tamanho da fila.
  - Verificar se a fila está vazia.



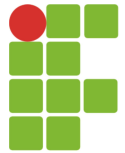
Operações em Filas de prioridade

- Conhecemos alguma estrutura que realiza estas operações eficientemente?

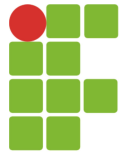


Operações em Filas de prioridade

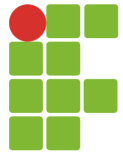
- Sim, uma **heap**!
- Mas temos que adaptá-la em sua versão dinâmica.
- Temos que usar vetores dinâmicos.



Sumário

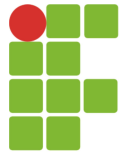


Sumário

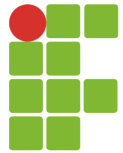


Filas de prioridade: Definição

```
7  typedef struct priority_queue_t {
8      int *pqueue;
9      size_t size;
10     size_t capacity;
11 } priority_queue_t;
```

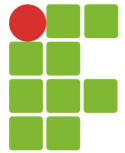


Sumário

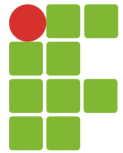


Filas de prioridade: Inicialização

```
39 void priority_queue_initialize(priority_queue_t **pq) {
40     (*pq) = mallocx(sizeof(priority_queue_t));
41     (*pq)->size = 0;
42     (*pq)->capacity = 4;
43     (*pq)->pqueue = mallocx(sizeof(int) * (*pq)->capacity);
44 }
```

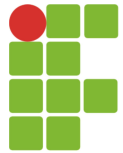


Sumário



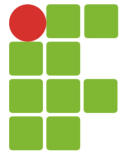
Filas de prioridade: obter tamanho

```
80  size_t priority_queue_size(priority_queue_t *pq) {
81      return pq->size;
82  }
```

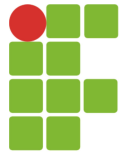


Filas de prioridade: verificar se a fila está vazia

```
84  bool priority_queue_empty(priority_queue_t *pq) {
85      return priority_queue_size(pq) == 0;
86  }
```



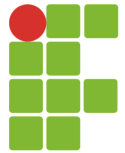
Sumário



Filas de prioridade: Inserção

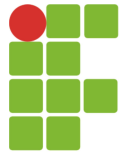
Inserção em Heap Dinâmica

- 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.



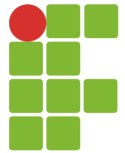
Filas de prioridade: Inserção

```
52  void priority_queue_push(priority_queue_t *pq, int data) {
53      if (pq->size == pq->capacity) {
54          pq->capacity *= 2;
55          pq->pqueue = reallocx(pq->pqueue, sizeof(int) * pq->capacity);
56      }
57      pq->pqueue[pq->size] = data;
58      priority_queue_heapify_bottom_up(pq, pq->size);
59      pq->size++;
60  }
```

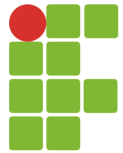


Filas de prioridade: Inserção

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



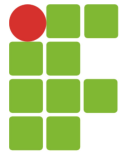
Sumário



Filas de prioridade: Acesso

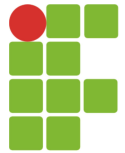
Consulta do Elemento de Maior Prioridade em Heap Dinâmica

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

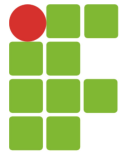


Filas de prioridade: acesso

```
75  int priority_queue_front(priority_queue_t *pq) {
76      assert(!priority_queue_empty(pq));
77      return pq->pqueue[0];
78  }
```



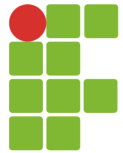
Sumário



Operações em Filas de prioridade

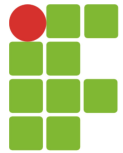
Remoção em Heap Dinâmica

- Pela propriedade de Heap, o elemento de maior prioridade ocupa a posição 0.
- A retirada é realizada 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.



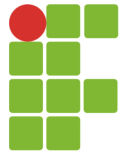
Filas de prioridade: remoção

```
62  void priority_queue_pop(priority_queue_t *pq) {
63      assert(!priority_queue_empty(pq));
64      if (pq->size == pq->capacity / 4 && pq->capacity > 4) {
65          pq->capacity /= 2;
66          pq->pqueue = reallocx(pq->pqueue, sizeof(int) * pq->capacity);
67      }
68      pq->size--;
69      if (!priority_queue_empty(pq)) {
70          pq->pqueue[0] = pq->pqueue[pq->size];
71          priority_queue_heapify_top_down(pq, 0);
72      }
73  }
```

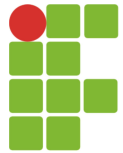


Filas de prioridade: remoção

```
17  static void priority_queue_heapify_top_down(priority_queue_t *pq, size_t i) {
18      size_t l, r;
19      size_t largest = i;
20      while (i < pq->size) {
21          i = largest;
22          l = 2 * i + 1;
23          r = 2 * i + 2;
24          if (l < pq->size && pq->pqueue[i] < pq->pqueue[l]) {
25              largest = l;
26          }
27          if (r < pq->size && pq->pqueue[largest] < pq->pqueue[r]) {
28              largest = r;
29          }
30          if (largest == i) {
31              break;
32          }
33          int aux = pq->pqueue[i];
34          pq->pqueue[i] = pq->pqueue[largest];
35          pq->pqueue[largest] = aux;
36      }
37  }
```

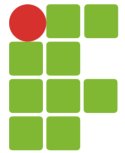


Sumário

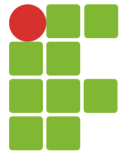


Filas de prioridade: limpeza

```
46 void priority_queue_delete(priority_queue_t **pq) {
47     free((*pq)->pqueue);
48     free(*pq);
49     (*pq) = NULL;
50 }
```

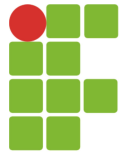


Sumário

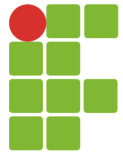


Operações em Filas de prioridade

Operação	Complexidade
Acesso	$\Theta(1)$
Remoção	$\Theta(\lg n)$ amortizado
Inserção	$\Theta(\lg n)$ amortizado



Sumário



Filas de prioridade: exemplo

```
1  #include "alloc.h"
2  #include "priority_queue.h"
3  #include <stdio.h>
4  #include <string.h>
5  #include <time.h>
6
7  int main(void) {
8      srand(time(NULL));
9      priority_queue_t *pq;
10     priority_queue_initialize(&pq);
11     int i;
12     for (i = 0; i < 1000; i++) {
13         int value = rand() % 10000;
14         printf("Inserindo %d na fila de prioridades.\n", value);
15         priority_queue_push(pq, value);
16     }
17     while (!priority_queue_empty(pq)) {
18         printf("Valor retirado da fila de prioridades: %d\n",
19                priority_queue_front(pq));
20         priority_queue_pop(pq);
21     }
22     priority_queue_delete(&pq);
23     return 0;
24 }
```