

Deques

Estruturas de Dados e Algoritmos – Ciência da Computação



Prof. Daniel Saad Nogueira
Nunes

IFB – Instituto Federal de Brasília,
Campus Taguatinga



Sumário

1 Introdução

2 Deques

3 Exemplos



Sumário

1 Introdução

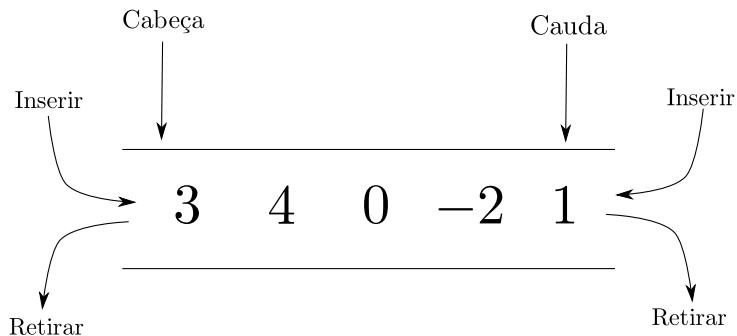


Deques

- Deque: Double-Ended-Queue.
- Deques são TADs que generalizam filas.
- Em deques, elementos podem ser adicionados tanto no início quando no fim da fila.
- Pode-se retirar elementos nas duas extremidades também.



Deques



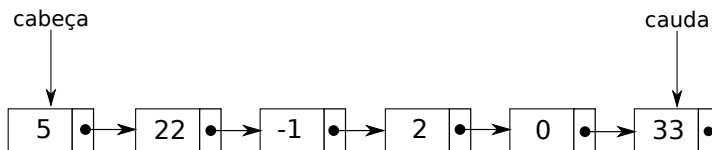


Implementação de Deques

- Que estruturas de dados podemos usar para implementar um deque?
- Esta estrutura deve suportar inserção/remoção na cabeça e na cauda em $\Theta(1)$.
- Será que uma lista encadeada simples funciona?



Implementação de Deques





Listas Encadeadas: Análise

Operação	Complexidade
Inserção na cabeça	$\Theta(1)$
Inserção na cauda	$\Theta(1)$
Remoção da cabeça	$\Theta(1)$
Remoção da cauda	$\Theta(n)$
Acesso à cabeça	$\Theta(1)$
Acesso à cauda	$\Theta(1)$



Implementação de Deques

- Listas encadeadas simples não permitem remoção da cauda em tempo constante.
- Temos que recorrer às listas duplamente encadeadas!

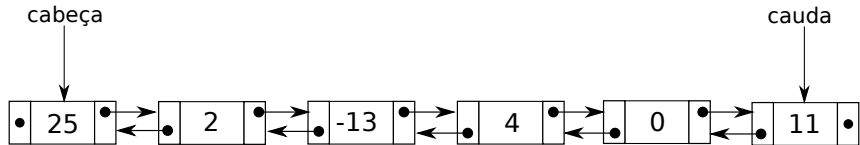


Listas Duplamente Encadeadas: Análise

Operação	Complexidade
Inserção na cabeça	$\Theta(1)$
Inserção na cauda	$\Theta(1)$
Remoção da cabeça	$\Theta(1)$
Remoção da cauda	$\Theta(1)$
Acesso à cabeça	$\Theta(1)$
Acesso à cauda	$\Theta(1)$



Implementação de Deques





Implementação de Deques

- Listas duplamente encadeadas se mostram uma escolha certa para implementação de Deques!



Sumário

2 Deques



Sumário

2 Deques

- Definição
- Inicialização
- Funções auxiliares
- Inserção na frente
- Inserção atrás
- Acesso na frente
- Acesso atrás
- Remoção na frente
- Remoção atrás
- Limpeza
- Análise



Deques: Definição

```
10 typedef void* (*deque_node_constructor_fn) (void*);  
11 typedef void (*deque_node_destructor_fn)(void *);
```



Deques: Definição

```
22 typedef struct deque_node_t{  
23     struct deque_node_t* next;  
24     struct deque_node_t* prev;  
25     void* data;  
26 }deque_node_t;
```




Deques: Definição

```
40 typedef struct deque_t{
41     struct deque_node_t* front;
42     struct deque_node_t* back;
43     deque_node_constructor_fn constructor;
44     deque_node_destructor_fn destructor;
45     size_t size;
46 }deque_t;
```



Sumário

2 Deques

- Definição
- **Inicialização**
- Funções auxiliares
- Inserção na frente
- Inserção atrás
- Acesso na frente
- Acesso atrás
- Remoção na frente
- Remoção atrás
- Limpeza
- Análise



Dequeues: Inicialização

```
6 void deque_initialize(deque_t** d, deque_node_constructor_fn constructor,  
7 deque_node_destructor_fn destructor){  
8     (*d) = mallocx(sizeof(deque_t));  
9     (*d)->front = NULL;  
10    (*d)->back = NULL;  
11    (*d)->constructor = constructor;  
12    (*d)->destructor = destructor;  
13    (*d)->size = 0;  
14 }
```



Sumário

2 Deques

- Definição
- Inicialização
- **Funções auxiliares**
- Inserção na frente
- Inserção atrás
- Acesso na frente
- Acesso atrás
- Remoção na frente
- Remoção atrás
- Limpeza
- Análise



Deques: Funções Auxiliares

```
97  size_t deque_size(deque_t* d){  
98      return d->size;  
99  }
```



Deque: Funções Auxiliares

```
102 size_t deque_empty(deque_t* d){  
103     return deque_size(d) == 0 ? 1 : 0;  
104 }
```



Sumário

2 Deques

- Definição
- Inicialização
- Funções auxiliares
- **Inserção na frente**
- Inserção atrás
- Acesso na frente
- Acesso atrás
- Remoção na frente
- Remoção atrás
- Limpeza
- Análise



Dequeues: Inserção na Frente

```
26 void deque_push_front(deque_t* d, void* data){
27     deque_node_t* new_node = malloc(sizeof(deque_node_t));
28     new_node->data = d->constructor(data);
29     new_node->next = d->front;
30     new_node->prev = NULL;
31     if(deque_empty(d)){
32         d->back = new_node;
33     }
34     else{
35         d->front->prev = new_node;
36     }
37     d->front = new_node;
38     d->size++;
39 }
```




Sumário

2 Deques

- Definição
- Inicialização
- Funções auxiliares
- Inserção na frente
- **Inserção atrás**
- Acesso na frente
- Acesso atrás
- Remoção na frente
- Remoção atrás
- Limpeza
- Análise



Deques: Inserção Atrás

```
43 void deque_push_back(deque_t* d, void* data){
44     deque_node_t* new_node = malloc(sizeof(deque_node_t));
45     new_node->data = d->constructor(data);
46     new_node->next = NULL;
47     new_node->prev = d->back;
48     if(deque_empty(d)){
49         d->front = new_node;
50     }
51     else{
52         d->back->next = new_node;
53     }
54     d->back = new_node;
55     d->size++;
56 }
```



Sumário

2 Deques

- Definição
- Inicialização
- Funções auxiliares
- Inserção na frente
- Inserção atrás
- **Acesso na frente**
- Acesso atrás
- Remoção na frente
- Remoção atrás
- Limpeza
- Análise



Deques: Acesso na Frente

```
85 void* deque_front(deque_t* d){  
86     assert(!deque_empty(d));  
87     return(d->front->data);  
88 }
```



Sumário

2 Deques

- Definição
- Inicialização
- Funções auxiliares
- Inserção na frente
- Inserção atrás
- Acesso na frente
- **Acesso atrás**
- Remoção na frente
- Remoção atrás
- Limpeza
- Análise



Deques: Acesso Atrás

```
91 void* deque_back(deque_t* d){  
92     assert(!deque_empty(d));  
93     return(d->back->data);  
94 }
```



Sumário

2 Deques

- Definição
- Inicialização
- Funções auxiliares
- Inserção na frente
- Inserção atrás
- Acesso na frente
- Acesso atrás
- **Remoção na frente**
- Remoção atrás
- Limpeza
- Análise



Deques: Acesso na Frente

```
59 void deque_pop_front(deque_t* d){  
60     assert(!deque_empty(d));  
61     deque_iterator_t it = d->front;  
62     d->front = d->front->next;  
63     if(deque_size(d)==1){  
64         d->back = NULL;  
65     }  
66     d->destructor(it->data);  
67     free(it);  
68     d->size--;  
69 }
```




Sumário

2 Deques

- Definição
- Inicialização
- Funções auxiliares
- Inserção na frente
- Inserção atrás
- Acesso na frente
- Acesso atrás
- Remoção na frente
- **Remoção atrás**
- Limpeza
- Análise



Deques: Acesso na Frente

```
72 void deque_pop_back(deque_t* d){  
73     assert(!deque_empty(d));  
74     deque_iterator_t it = d->back;  
75     d->back = d->back->prev;  
76     if(deque_size(d)==1){  
77         d->front = NULL;  
78     }  
79     d->destructor(it->data);  
80     free(it);  
81     d->size--;  
82 }
```



Sumário

2 Deques

- Definição
- Inicialização
- Funções auxiliares
- Inserção na frente
- Inserção atrás
- Acesso na frente
- Acesso atrás
- Remoção na frente
- Remoção atrás
- **Limpeza**
- Análise



Dequeues: Limpeza

```
17 void deque_delete(deque_t** d){  
18     while(!deque_empty(*d)){  
19         deque_pop_front(*d);  
20     }  
21     free(*d);  
22     (*d) = NULL;  
23 }
```



Sumário

2 Deques

- Definição
- Inicialização
- Funções auxiliares
- Inserção na frente
- Inserção atrás
- Acesso na frente
- Acesso atrás
- Remoção na frente
- Remoção atrás
- Limpeza
- **Análise**



Deques: Análise

Operação	Complexidade
Inserção na frente	$\Theta(1)$
Inserção atrás	$\Theta(1)$
Remoção da frente	$\Theta(1)$
Remoção atrás	$\Theta(1)$
Acesso à frente	$\Theta(1)$
Acesso atrás	$\Theta(1)$



Sumário

3 Exemplos



Exemplo de Utilização da Biblioteca

```
6 typedef struct pessoa{
7     char nome[30];
8     char cpf[20];
9     int idade;
10 }pessoa;
```




Exemplo de Utilização da Biblioteca

```
12 void* constructor_pessoa(void* data){  
13     void* ptr = malloc(sizeof(pessoa));  
14     memcpy(ptr,data,sizeof(pessoa));  
15     return ptr;  
16 }
```



Exemplo de Utilização da Biblioteca

```
35 void destructor_pessoa(void* data){  
36     free(data);  
37 }
```



Exemplo de Utilização da Biblioteca

```
18 void my_getline(char* str,size_t size){
19     int i;
20     char c;
21     for(i=0;i<size-1;i++){
22         c = getchar();
23         if(c=='\n'){
24             str[i] = '\0';
25             break;
26         }
27         str[i] = c;
28     }
29     str[size-1]='\0';
30     while(c!='\n'){
31         c = getchar();
32     }
33 }
```



Exemplo de Utilização da Biblioteca

```
39 void cadastra_pessoa(pessoa* p){
40     printf("Nome: ");
41     my_getline(p->nome,30);
42     printf("CPF: ");
43     my_getline(p->cpf,20);
44     printf("Idade: ");
45     scanf("%d%c",&p->idade);
46 }
```



Exemplo de Utilização da Biblioteca

```
48 void imprime_pessoa(const pessoa* p){  
49     printf("Nome: ");  
50     printf("%s\n",p->nome);  
51     printf("CPF: ");  
52     printf("%s\n",p->cpf);  
53     printf("Idade: ");  
54     printf("%d\n",p->idade);  
55 }
```



Exemplo de Utilização da Biblioteca

```
57 int main(void){
58     int i;
59     deque_t* d;
60     pessoa p;
61     deque_initialize(&d,constructor_pessoa,destructor_pessoa);
62     for(i=0;i<5;i++){
63         printf("Cadastrando pessoa %d\n",i+1);
64         cadastra_pessoa(&p);
65         if(i%2==0){
66             deque_push_front(d,&p);
67         }
68         else{
69             deque_push_back(d,&p);
70         }
71     }
72     i=0;
```



Exemplo de Utilização da Biblioteca

```
73 while(!deque_empty(d)){
74     printf("\n**Imprimindo pessoa**\n");
75     if(i%2==0){
76         p = *(pessoa*) deque_front(d);
77         deque_pop_front(d);
78     }
79     else{
80         p = *(pessoa*) deque_back(d);
81         deque_pop_back(d);
82     }
83     imprime_pessoa(&p);
84     printf("\n");
85     i++;
86 }
87 deque_delete(&d);
88 return 0;
89 }
```