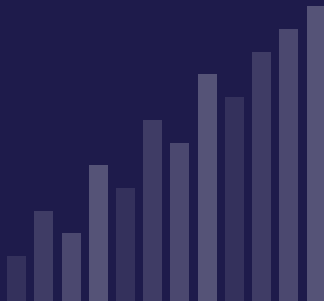

AULA 15 · BLOCO 2: DICIONÁRIOS E ESTRUTURAS LINEARES

Listas duplamente ligadas e listas circulares

Seção 10.2

Estruturas de dados e seus algoritmos · José Dihego · IFBA



Roteiro da aula

Ao final, você será capaz de

- ▶ Implementar a lista duplamente ligada e seus casos especiais sem sentinela
- ▶ Remover um nó dado em $O(1)$
- ▶ Fechar a lista num círculo e percorrê-la sem fim
- ▶ Resolver o problema de Josefo e comparar lista circular com vetor



Teoria

Ponteiro *ant*, LIST-INSERT, LIST-DELETE, listas circulares



Prática em C

Lista dupla sem sentinela e o problema de Josefo em C



Experimento

Josefo: $(n - 1)(k - 1)$ passos na lista contra $\Theta(n^2)$ no vetor



Conexões

Sistemas operacionais, navegadores, caches e jogos

Lista duplamente ligada

Cada nó tem *chave*, *prox* e *ant*. Andar para os dois lados e, principalmente, **remover um nó dado em $O(1)$** : o anterior está em $x.ant$.

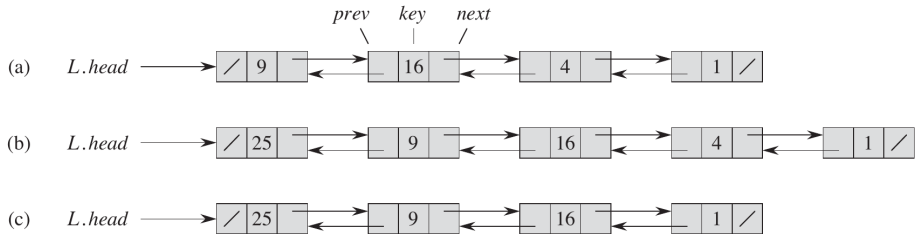


Figura 10.3 de Cormen et al. (2009) — (a) lista $\langle 9, 16, 4, 1 \rangle$; (b) após LIST-INSERT de 25; (c) após LIST-DELETE do 4.

Inserir e remover sem sentinela

- ▶ Ambas $O(1)$.
- ▶ Repare nos **se**: lista vazia, x na cabeça, x na cauda. Cada um é uma chance de erro.
- ▶ Na aula 16, uma **sentinela** elimina todos esses testes.

LIST-INSERT(L, x) e LIST-DELETE(L, x)

```
1                                     // LIST-INSERT:  $x$  na cabeça
2  $x.prox \leftarrow L.cabeça$ 
3 se  $L.cabeça \neq \text{NIL}$  então
4    $L.cabeça.ant \leftarrow x$ 
5  $L.cabeça \leftarrow x$ ;  $x.ant \leftarrow \text{NIL}$ 
6
7                                     // LIST-DELETE
8 se  $x.ant \neq \text{NIL}$  então
9    $x.ant.prox \leftarrow x.prox$ 
10 senão
11    $L.cabeça \leftarrow x.prox$ 
12 se  $x.prox \neq \text{NIL}$  então
13    $x.prox.ant \leftarrow x.ant$ 
```

Listas circulares

- ▶ Na lista **circular**, o *prox* da cauda aponta para a cabeça (e, na dupla, o *ant* da cabeça aponta para a cauda).
- ▶ Não há fim: de qualquer nó se chega a todos. Basta guardar um ponteiro para **um** nó.
- ▶ Na remoção não há extremos, logo não há casos especiais (desde que reste mais de um nó).
- ▶ Usos: escalonamento *round robin*, *buffers* circulares, jogadores em turnos.

Problema de Josefo: n pessoas em roda; contando a partir da primeira, cada k -ésima sai, até restar uma. Quem sobrevive? Com lista circular: $(n - 1)(k - 1)$ passos e $O(1)$ por remoção.

Para $k = 2$ há fórmula fechada: se $n = 2^m + \ell$ com $\ell < 2^m$, sobrevive $2\ell + 1$ (problema 1.3 de *Matemática Concreta*).



Prática em C

- ▶ Lista dupla sem sentinela: os casos especiais explícitos
- ▶ A mesma lista fechada em círculo
- ▶ Problema de Josefo: ordem de saída e sobrevivente

Lista dupla e circular em C (1/3)

```
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  /* Lista duplamente ligada SEM sentinela: cabeça e cauda podem ser NULL, então
5     inserir e remover têm casos especiais (compare com a sentinela da aula 16). */
6  typedef struct No { int chave; struct No *ant, *prox; } No;
7
8  No *insere_fim(No **cabeca, No **cauda, int k) {
9     No *x = malloc(sizeof(No));
10     x->chave = k;
11     x->prox = NULL;
12     x->ant = *cauda;
13     if (*cauda) (*cauda)->prox = x;           // caso geral
14     else *cabeca = x;                          // caso especial: lista vazia
15     *cauda = x;
16     return x;
```

Lista dupla e circular em C (2/3)

```
17 }
18
19 void remove_no(No **cabeca, No **cauda, No *x) {
20     if (x->ant) x->ant->prox = x->prox; else *cabeca = x->prox;    // x era a cabeça
21     if (x->prox) x->prox->ant = x->ant; else *cauda = x->ant;      // x era a cauda
22     free(x);
23 }
24
25 int main(void) {
26     int n, k;
27     if (scanf("%d %d", &n, &k) != 2 || n < 1 || k < 1) return 1;
28     No *cabeca = NULL, *cauda = NULL;
29     for (int i = 1; i <= n; i++) insere_fim(&cabeca, &cauda, i);
30     /* Problema de Josefo: n pessoas em roda, cada k-ésima sai. Fechamos a lista
31        num CÍRCULO (cauda->prox = cabeça): andar é só seguir prox, sem testar o fim. */
32     cauda->prox = cabeca;
```

Lista dupla e circular em C (3/3)

```
33     cabeca->ant = cauda;
34     No *x = cauda;                                // começa "antes" da pessoa 1
35     printf("ordem de saída:");
36     for (int restam = n; restam > 1; restam--) {
37         for (int passo = 0; passo < k; passo++) x = x->prox;
38         printf(" %d", x->chave);
39         No *sai = x;
40         x = x->ant;                                // continua a partir do anterior
41         sai->ant->prox = sai->prox;                  // na lista circular não há extremos:
42         sai->prox->ant = sai->ant;                  // a remoção não tem casos especiais
43         free(sai);
44     }
45     printf("\nsobrevivente: %d\n", x->prox->chave);
46     free(x->prox == x ? x : x->prox);
47     return 0;
48 }
```



Experimento

- ▶ $k = 3$; n de 2.000 a 32.000
- ▶ Lista: $(n - 1)(k - 1)$ passos
- ▶ Vetor: acha a posição em $O(1)$, mas remover desloca

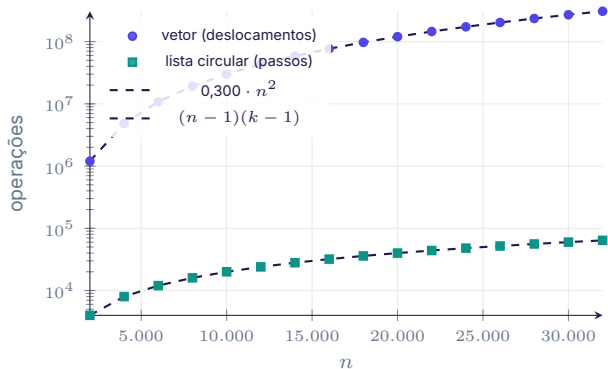
Josefo: lista circular contra vetor

Hipótese. Na lista circular, cada eliminação anda $k - 1$ nós e remove em $O(1)$: exatamente $(n - 1)(k - 1)$ passos, **linear**. No vetor, a posição sai de uma conta $((pos + k - 1) \% tam)$, mas remover desloca os elementos à direita: $\Theta(n^2)$ no total.

Método

1. Lista circular simples com os nós num vetor de struct
2. Vetor com remoção por deslocamento
3. Os dois sobreviventes são conferidos
4. Contagem exata de passos e deslocamentos

Passos e deslocamentos



n	medido	razão
2.000	1.199.342	0,300
8.000	19.196.162	0,300
14.000	58.790.924	0,300
20.000	119.993.562	0,300
26.000	202.798.972	0,300
32.000	307.183.368	0,300

razão = medido $\div n^2$

✓ desvio máx. 0,0%

Lista: exatamente $2(n - 1)$ passos.
 Vetor: cerca de $0,3 n^2$ deslocamentos;
 com 32 mil pessoas, 307 milhões
 contra 64 mil.

EXPERIMENTO

O que o experimento mostrou

A lista circular resolve Josefo em exatamente $(n - 1)(k - 1)$ passos; o vetor, em $\approx 0,3n^2$ deslocamentos: **4.800 vezes** mais trabalho para $n = 32.000$. Quando a operação dominante é remover no meio, a lista ligada vence; quando é acessar por posição, o vetor vence.

➤ Reproduza: `gcc -O2 aula15_experimento.c -o exp -lm && ./exp`



Conexões

- ▶ Sistemas operacionais
- ▶ Navegadores
- ▶ Caches LRU
- ▶ Jogos

Onde isso aparece na computação

Sistemas operacionais

O kernel Linux usa listas duplas circulares (`list_head`) em toda parte; o escalonador *round robin* percorre uma lista circular de processos.

Navegadores

O histórico com voltar e avançar e as abas em ordem são listas duplas: sair de uma aba no meio é remover um nó em $O(1)$.

Caches LRU

Um cache LRU combina uma tabela hash com uma lista dupla: o item usado vai para a frente, o mais antigo sai do fim, tudo em $O(1)$.

Jogos

Turnos de jogadores, animações em laço e trilhas de áudio em repetição são percorridos como listas circulares.

Para casa

Exercícios

1. (10.2-5) Implemente INSERT, DELETE e SEARCH numa lista circular simplesmente ligada. Custos?
2. (10.2-6) Uma dois conjuntos disjuntos guardados em listas em $O(1)$. Que lista usar?
3. (10.2-8) Lista dupla com um só campo $x.np = x.prox$ XOR $x.ant$: implemente busca, inserção e remoção.
4. Resolva Josefo para $k = 2$ e confira a fórmula $2\ell + 1$ para vários n .
5. Implemente um cache LRU de capacidade fixa com lista dupla (e, na aula 19, com tabela hash).

Leitura

Cormen et al., seção 10.2 (listas duplamente ligadas e circulares).

Código

josedihego.net/estruturas