ARRAYS E LISTAS

Gustavo Carvalho
(ghpc@cin.ufpe.br)

Universidade Federal de Pernambuco
Centro de Informática, 50740-560, Brazil

Agenda

Estático *vs.* dinâmico

Alocação memória: compilação | pilha *vs.* execução | heap
Memória após executar as linhas: 4, 7, 8, 9, 10, 11, 12 e 14

```
1  int m = 2;
2  int fun(int d, int *h) {
3    int e = *h;
4    return (d * e * m) / 5;
5  }
6  void main() {
7    int a = 1; int *b; int *c; int *d; int **e; int **f;
8    b = &a; *b = *b * 10;
9    c = malloc(sizeof(int)); *c = *b; d = c;
10   e = malloc(sizeof(int*)); *e = malloc(sizeof(int)); **e = *d;
11   f = malloc(sizeof(int*)); *f = b; **f = *b * **f;
12   **f = fun(a, b); printf("Valor de a = %d\n", a);
13   free(c); free(*e); free(e); free(f);
14   // free(b); free(*f); // errado
15 }
```

Arrays estáticos *vs.* arrays dinâmicos

Arrays alocados estaticamente *vs.* alocados dinamicamente

- Capacidade fixa (não é possível adicionar | remover elementos)

Arrays dinâmicos: uma possível implementação

- Crie um array com capacidade para n elementos
- Inserção provoca um deslocamento para a direita
  - Sem espaço: aumentar (e.g., dobrar) tamanho e copiar dados
- Remoção prova um deslocamento para a esquerda

Arrays estáticos *vs.* arrays dinâmicos

Arrays alocados estaticamente *vs.* alocados dinamicamente

- Capacidade fixa (não é possível adicionar | remover elementos)

Arrays dinâmicos: uma possível implementação

- Crie um array com capacidade para n elementos
- Inserção provoca um deslocamento para a direita
  - Sem espaço: aumentar (e.g., dobrar) tamanho e copiar dados
- Remoção prova um deslocamento para a esquerda

Em Java = Vector, ArrayList

Em C++ = std::vector

Agenda

Tipos abstratos de dados e estruturas de dados

Tipo: um conjunto de valores

- $bool = \{false, true\}$
- $byte = \{-128 \,..\, 127\}$

Tipo simples: seus valores não possuem “subpartes”

Tipo composto: um conjunto de valores compostos

```
1  typedef struct b2 {
2    bool first;
3    bool second;
4  } B2;
```

- $B2 = \{(false, false), (false, true), (true, false), (true, true)\}$

Tipos abstratos de dados e estruturas de dados

Tipo de dados: um tipo e operações associadas

- Uma variável inteira é um elemento do tipo de dados inteiro
- Adição é uma operação associada ao tipo de dados inteiro

Tipos abstratos de dados e estruturas de dados

Tipo de dados: um tipo e operações associadas

- Uma variável inteira é um elemento do tipo de dados inteiro
- Adição é uma operação associada ao tipo de dados inteiro

Tipo de dados: representação lógica e representação física

- Tipo abstrato de dados (ADT): representação lógica (o quê)
- Estrutura de dados: representação física (como)

Tipos abstratos de dados e estruturas de dados

Tipo de dados: um tipo e operações associadas

- Uma variável inteira é um elemento do tipo de dados inteiro
- Adição é uma operação associada ao tipo de dados inteiro

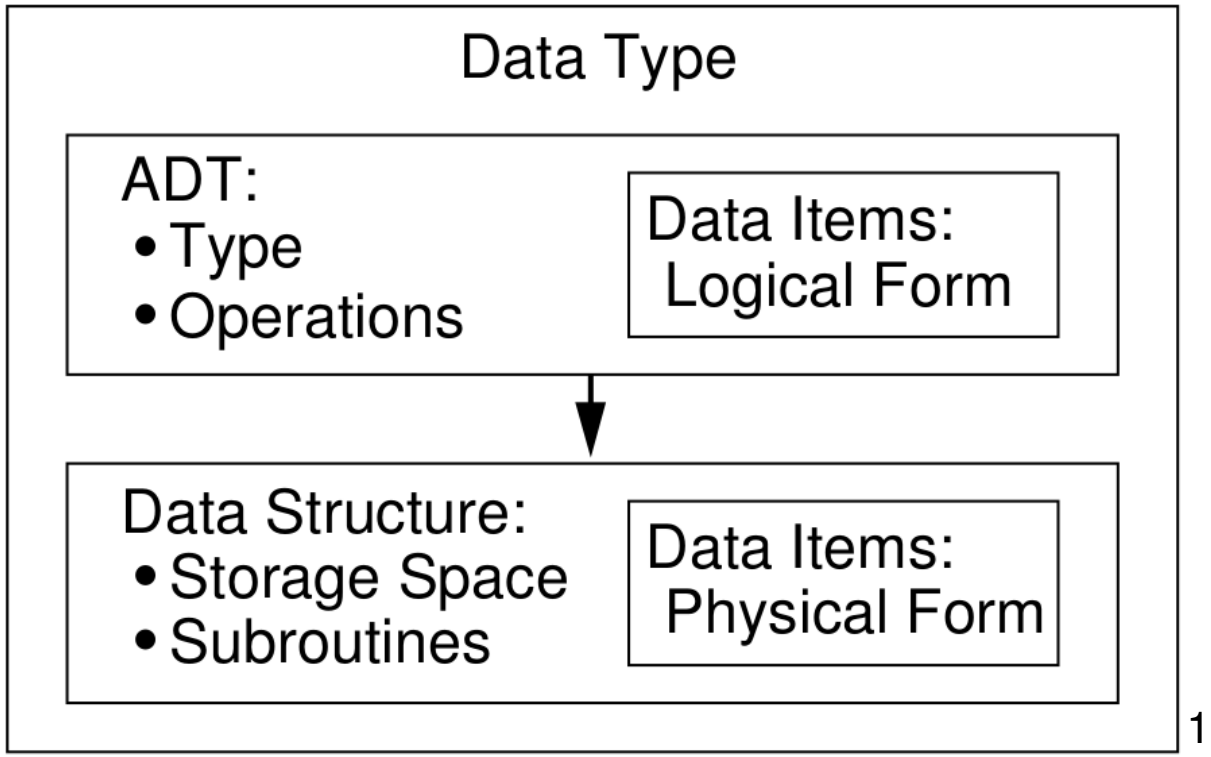
Tipo de dados: representação lógica e representação física

- Tipo abstrato de dados (ADT): representação lógica (o quê)
- Estrutura de dados: representação física (como)

Exemplo:

- ADT: lista com operações de inserir, buscar e remover
- Estrutura de dados: listas baseadas em arrays, listas ligadas
  - Por que duas implementações para uma mesma ADT?

Tipos abstratos de dados e estruturas de dados



[1]

[1] Fonte: C. Shaffer. Data Structures and Algorithm Analysis. 2013.

Agenda

List ADT

Lista com n elementos: $\langle a_0, a_1, ..., a_{n-1} \rangle$

- Abstração matemática: sequências
- Início da lista, posição corrente (símbolo $|$) e final da lista;
- Ao inserir 10 em $\langle 20, 23 \mid 12, 15 \rangle$, obtém-se $\langle 20, 23 \mid 10, 12, 15 \rangle$

List ADT

Operações:

- void clear(List l);
- void insert(List l, E item);
- void append(List l, E item);
- E remove(List l);
- void moveToStart(List l);
- void moveToEnd(List l);
- void prev(List l);
- void next(List l);
- int length(List l);
- int currPos(List l);
- void moveToPos(List l, int pos);
- E getValue(List l);

List ADT

Percorrendo uma lista l:

```
1  moveToStart(l);
2  while currPos(l) < length(l) do
3      it ← getValue(l);
4      doSomething(it);
5      next(l);
```

List ADT

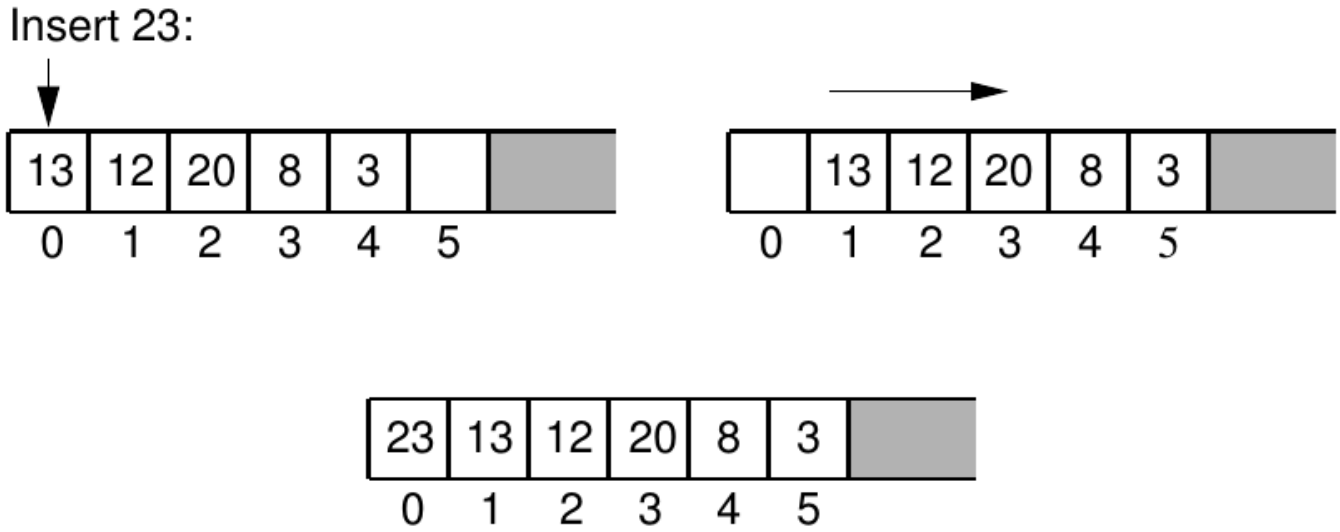
Procurando um elemento na lista:

Algoritmo: boolean find(List l, int k)

```
1  moveToStart(l);
2  while currPos(l) < length(l) do
3      if k = getValue(l) then
4          return true;
5      next(l);
6  return false;
```

Implementação baseada em array

Inserção



[2]

Remoção: implementação análoga

[2] Fonte: C. Shaffer. Data Structures and Algorithm Analysis. 2013.

Implementação baseada em array

Estrutura de dados (List):

```
1  int maxSize ;                          // capacidade
2  int listSize ;                  // quantidade de elementos
3  int curr ;                           // posição corrente
4  E[] listArray ;                       // array com dados
```

Algoritmo: List create_list(int size)

1. $l.maxSize \leftarrow size$;
2. $l.listSize \leftarrow l.curr \leftarrow 0$;
3. $l.listArray \leftarrow new\ E[size]$;
4. **return** l;

Implementação baseada em array (continuação)

Algoritmo: void clear(List l)

```
1  l.listSize ← l.curr ← 0;
```

Algoritmo: void insert(List l, E it)

```
1  if l.listSize ≥ l.maxSize then error ;
2  i ← l.listSize;
3  while i > l.curr do
4      l.listArray[i] ← l.listArray[i − 1] ;        // deslocamento direita
5      i--;
6  l.listArray[l.curr] ← it;
7  l.listSize++;
```

Implementação baseada em array (continuação)

Algoritmo: void append(List l, E it)

```
1  if l.listSize ≥ l.maxSize then error ;
2  l.listArray[l.listSize++] ← it;
```

Algoritmo: E remove(List l)

```
1  if l.curr < 0 ∨ curr ≥ listSize then return NULL ;
2  E it ← l.listArray[l.curr];
3  i ← l.curr;
4  while i < l.listSize − 1 do
5      l.listArray[i] ← l.listArray[i + 1] ;    // deslocamento esquerda
6      i++;
7  l.listSize--;
8  return it;
```

Implementação baseada em array (continuação)

Algoritmo: void moveToStart(List l)

```
1  l.curr ← 0;
```

Algoritmo: void moveToEnd(List l)

```
1  l.curr ← l.listSize;
```

Algoritmo: void prev(List l)

```
1  if l.curr ≠ 0 then l.curr--;
```

Algoritmo: void next(List l)

```
1  if l.curr < l.listSize then curr++;
```

Implementação baseada em array (continuação)

Algoritmo: int length(List l)

```
1  return l.listSize;
```

Algoritmo: int currPos(List l)

```
1  return l.curr;
```

Algoritmo: void moveToPos(List l, int pos)

```
1  if pos < 0 ∨ pos > l.listSize then error ;
2  l.curr ← pos;
```

Algoritmo: E getValue(List l)

```
1  if l.curr < 0 ∨ l.curr ≥ l.listSize then error ;
2  return l.listArray[l.curr];
```

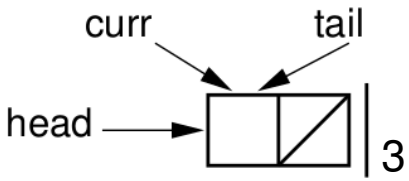
Implementação baseada em array: exercício

1. Desenhar a rep. lógica após as linhas 2, 3, 4, 5, 6, 7 e 10
2. Desenhar a rep. física após as linhas 2, 3, 4, 5, 6, 7 e 10
3. Indicar os valores impressos nas linhas 8, 9 e 11

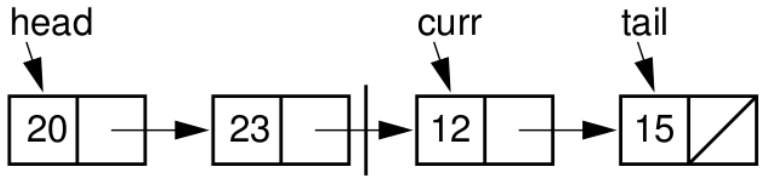
```
1  l ← create_list(5);
2  insert(l, 15);
3  insert(l, 12);
4  insert(l, 23);
5  insert(l, 20);
6  next(l);
7  next(l);
8  print(length(l));
9  print(currPos(l));
10 prev(l);
11 print(remove(l));
```

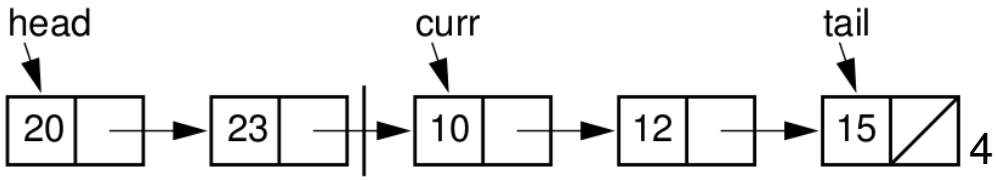
Implementação baseada em lista ligada

Lista vazia: nó *header vs.* casos especiais em *insert* e *remove*



Cuidado: para onde *curr* aponta

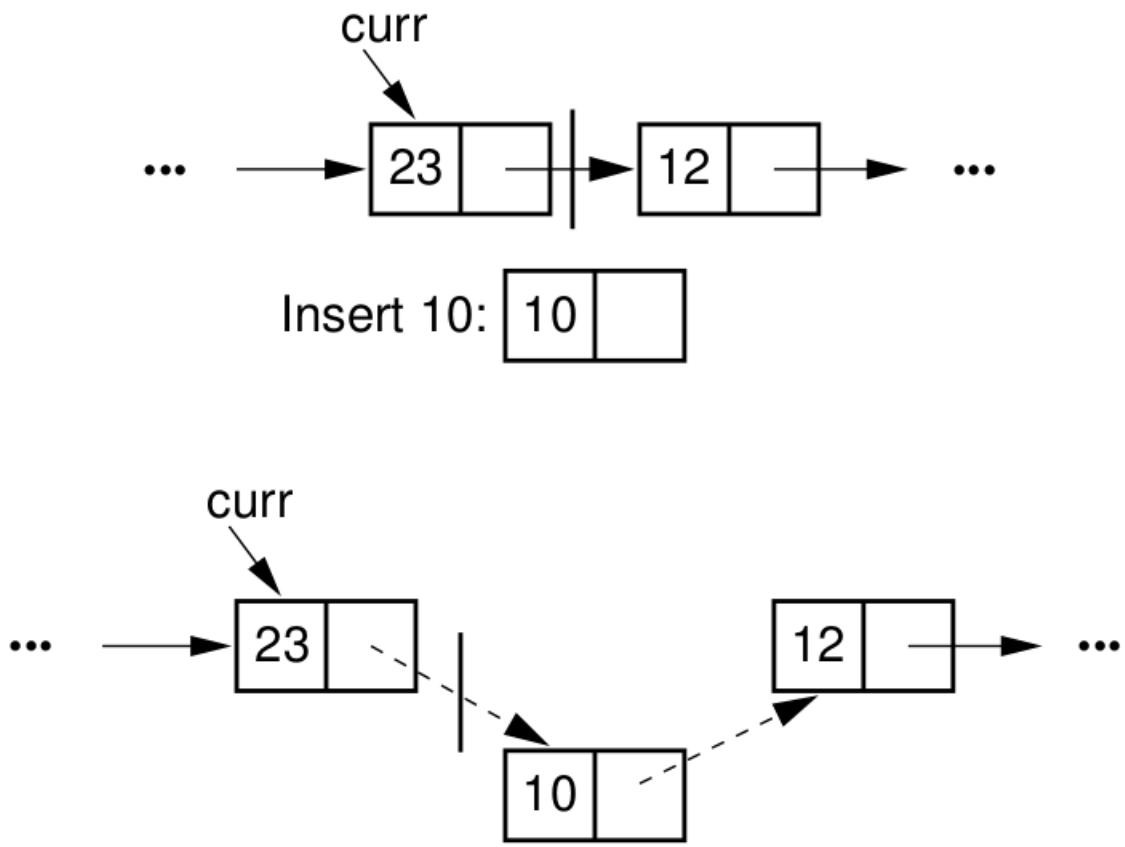




[3] Fonte: C. Shaffer. Data Structures and Algorithm Analysis. 2013.

[4] Fonte: C. Shaffer. Data Structures and Algorithm Analysis. 2013.

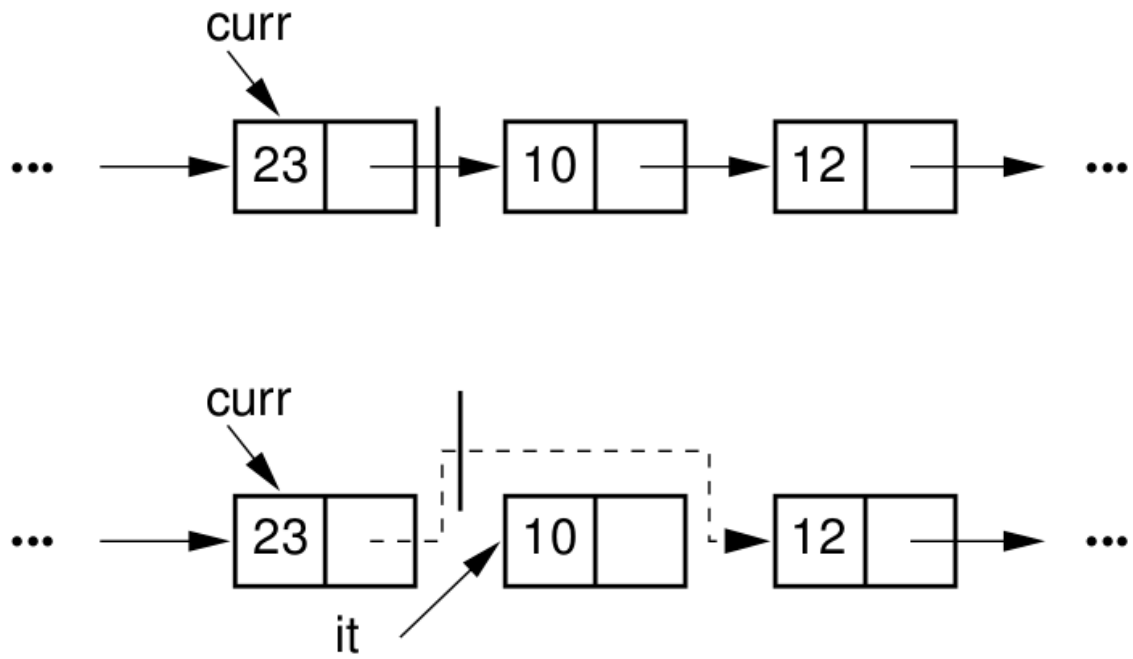
Implementação baseada em lista ligada: inserção



5

[5] Fonte: C. Shaffer. Data Structures and Algorithm Analysis. 2013.

Implementação baseada em lista ligada: remoção



[6]

Importante: preocupação com desalocação do nó removido

- Linguagens sem GC: do usuário da estrutura
- Linguagens com GC: do *garbage collector*

[6] Fonte: C. Shaffer. Data Structures and Algorithm Analysis. 2013.

Implementação baseada em lista ligada

Estrutura de dados (Link):

```
1  E element ;                                  // valor deste nó
2  Link next ;                          // referência para o próximo nó
```

Algoritmo: Link create_link(E it, Link nextval)

1 $n.element \leftarrow it$;
2 $n.next \leftarrow nextval$;
3 **return** n;

Algoritmo: Link create_link(Link nextval)

1 $n.next \leftarrow nextval$;
2 **return** n;

Implementação baseada em lista ligada (cont.)

Estrutura de dados (List):

```
1  Link head;
2  Link tail;
3  Link curr;
4  int cnt ;                                  // tamanho da lista
```

Algoritmo: List create_list()

```
1  l.curr ← l.tail ← l.head ← create_link(NULL) ; // cria nó header
2  l.cnt ← 0;
3  return l;
```

Implementação baseada em lista ligada (cont.)

Algoritmo: void clear(List l)

```
1  l.curr ← l.tail ← l.head ← create_link(NULL) ; // cria nó header
2  l.cnt ← 0;
3  return l;
```

Algoritmo: void insert(List l, E it)

```
1  l.curr.next ← create_link(it, l.curr.next);
2  if l.tail = l.curr then l.tail ← l.curr.next;
3  l.ctn++;
```

Algoritmo: void append(List l, E it)

```
1  l.tail.next ← create_link(it, NULL);
2  l.tail ← l.tail.next;
3  l.ctn++;
```

Implementação baseada em lista ligada (cont.)

Algoritmo: E remove(List l)

```
    // nada a remover
1   if l.curr.next = NULL then  return NULL ;
2   E it ← l.curr.next.element;
    // removido o último
3   if l.tail = l.curr.next then l.tail ← l.curr;
4   l.curr.next ← l.curr.next.next ;            // removendo da lista
5   ctn--;
6   return it;
```

Algoritmo: void moveToStart(List l)

```
1   l.curr ← l.head;
```

Implementação baseada em lista ligada (cont.)

Algoritmo: void moveToEnd(List l)

```
1  l.curr ← l.tail;
```

Algoritmo: void prev(List l)

```
   // não há anterior
1  if l.curr = l.head then return ;
2  Link temp ← l.head;
3  while temp.next ≠ l.curr do
4  |    temp ← temp.next;
5  l.curr ← temp;
```

Algoritmo: void next(List l)

```
1  if l.curr ≠ l.tail then l.curr ← l.curr.next ;
```

Implementação baseada em lista ligada (cont.)

Algoritmo: int length(List l)

```
1  return l.ctn
```

Algoritmo: int currPos(List l)

```
1  Link temp ← l.head;
2  i ← 0;
3  while l.curr ≠ temp do
4  |    temp ← temp.next;
5  |    i++;
6  return i;
```

Implementação baseada em lista ligada (cont.)

Algoritmo: void moveToPos(List l, int pos)

```
1  if pos < 0 ∨ pos > l.ctn then error ;
2  l.curr ← l.head;
3  i ← 0;
4  while i < pos do
5      l.curr ← l.curr.next;
6      i++;
```

Algoritmo: E getValue(List l)

```
1  if l.curr.next = NULL then return NULL;
2  return curr.next.element;
```

Implementação baseada em lista ligada: exercício

1. Desenhar a rep. lógica após as linhas 2, 3, 4, 5, 6, 7 e 10
2. Desenhar a rep. física após as linhas 2, 3, 4, 5, 6, 7 e 10
3. Indicar os valores impressos nas linhas 8, 9 e 11

```
1  l ← create_list();
2  insert(l, 15);
3  insert(l, 12);
4  insert(l, 23);
5  insert(l, 20);
6  next(l);
7  next(l);
8  print(length(l));
9  print(currPos(l));
10 prev(l);
11 print(remove(l));
```

Única mudança: linha 1 (abstração: ADT)

Comparação das implementações

Array-based lists

- $\pm$ Tamanho máximo predefinido (alternativa: arrays dinâmicos)
- $+$ Não consome espaço com ponteiros (links)
- $+$ Mais rápido para acesso aleatório
- $-$ Custo computacional do *insert* e do *remove*

Linked lists

- $+$ Sem tamanho predefinido (limite: memória disponível)
- $+$ Só consome espaço para elementos na lista
- $-$ Custo computacional do *prev*, *currPos* e *moveToPos*
- $+$ Custo computacional do *insert* e do *remove*

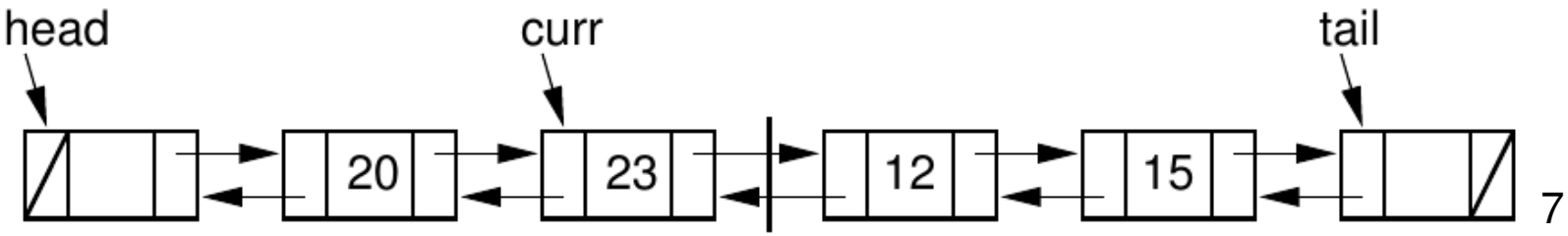
Otimizações para listas ligadas

Gerenciamento de memória (custo new e GC): freelist

- Ao remover: adiciona no início da freelist
- Ao inserir: usa um nó da freelist; se vazia, cria novo nó

Doubly linked lists

- + Melhor custo computacional para *prev*
- − Consome mais memória (armazenar dois links por nó)



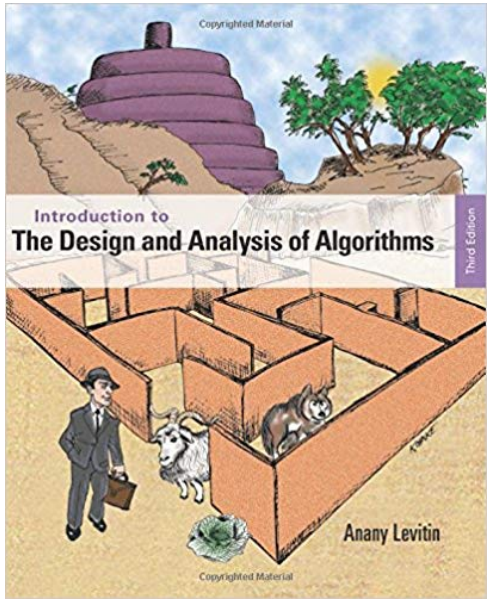
[7]

Outra variação de listas: listas circulares

[7] Fonte: C. Shaffer. Data Structures and Algorithm Analysis. 2013.

Agenda

Bibliografia + leitura recomendada



Capítulo 1 (pp. 25–28).
Anany Levitin.
Introduction to the Design and Analysis of Algorithms.
3a edição. Pearson. 2011.

Capítulo 1 (pp. 8–12)
Capítulo 4 (pp. 93–117)
Clifford Shaffer.
Data Structures and Algorithm Analysis.
Dover, 2013.

ARRAYS E LISTAS

Gustavo Carvalho
(ghpc@cin.ufpe.br)

Universidade Federal de Pernambuco
Centro de Informática, 50740-560, Brazil