Listas circulares e Listas Duplamente Encadeadas

Estrutura de Dados — QXD0010

Prof. Atílio Gomes Luiz

gomes.atilio@ufc.br

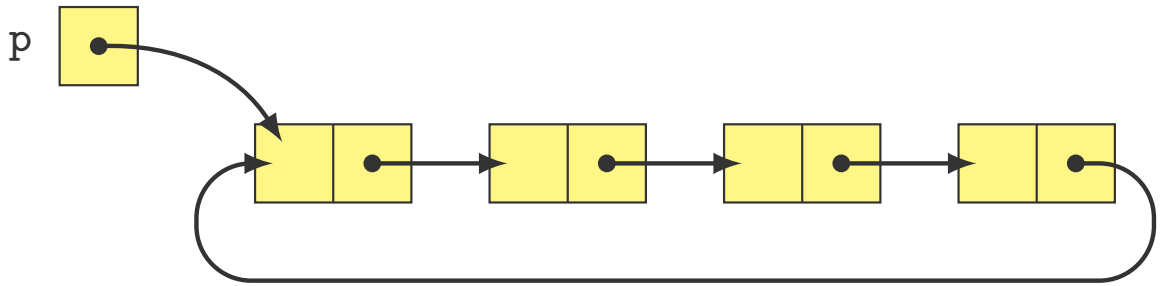
Universidade Federal do Ceará

1º semestre/2020

Introdução

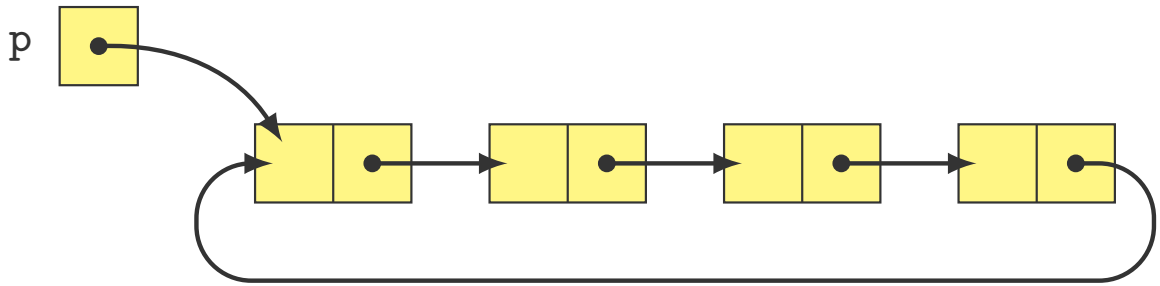
Lista circular simplesmente encadeada

Lista circular (sem nó cabeça):

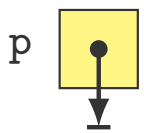


Lista circular simplesmente encadeada

Lista circular (sem nó cabeça):

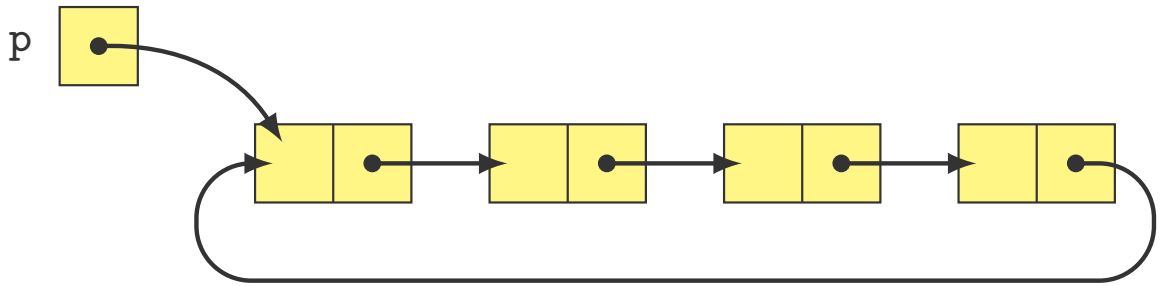


Lista circular vazia: ponteiro `p` é nulo.

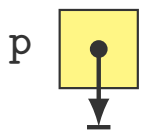


Lista circular simplesmente encadeada

Lista circular (sem nó cabeça):



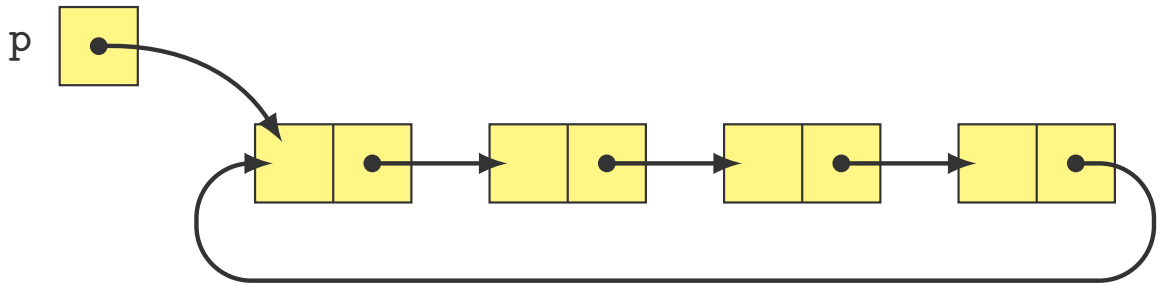
Lista circular vazia: ponteiro `p` é nulo.



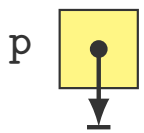
Exemplo de aplicações:

Lista circular simplesmente encadeada

Lista circular (sem nó cabeça):



Lista circular vazia: ponteiro `p` é nulo.



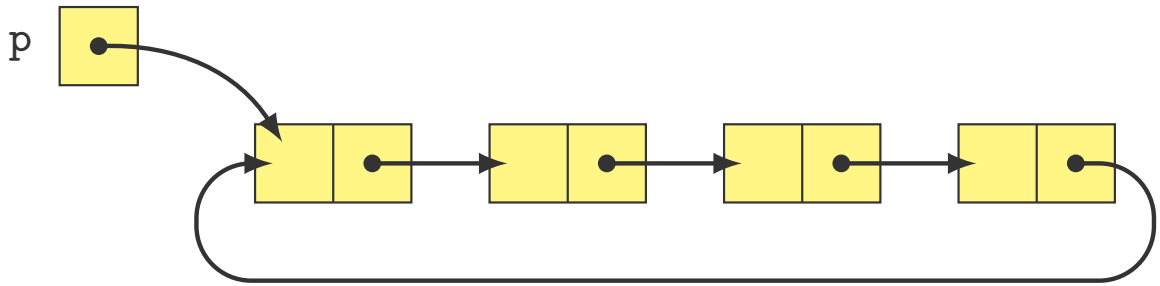
Exemplo de aplicações:

- Execução de processos no sistema operacional

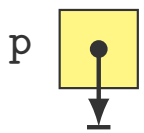
Lista circular simplesmente encadeada

UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ

Lista circular (sem nó cabeça):



Lista circular vazia: ponteiro `p` é nulo.



Exemplo de aplicações:

- Execução de processos no sistema operacional
- Controlar de quem é a vez em um jogo de tabuleiro

Implementação em C++

Arquivo `List.h` – Interface da Classe


```
1  #ifndef LISTCIRC_H
2  #define LISTCIRC_H
3  struct Node;
4
5  class List {
6  private:
7      Node *head; // Ponteiro para o primeiro no da lista
8      //Operacao auxiliar: retorna o no com valor x,
9      // ou nullptr caso x nao esteja presente
10     Node *search(int x);
11 public:
12       List(); // Construtor
13       ~List(); // Destrutor: libera memoria alocada
14       void add_back(int x); // Insere x ao final da lista
15       void add_front(int x); // Insere x no inicio da lista
16       void remove(int x); // remove o primeiro no com valor x
17       void removeAll(int x); // remove todo no com valor x
18       bool contains(int x); // lista contem x?
19       void print(); // Imprime elementos da lista
20       bool empty(); // Esta vazia?
21       int size(); // retorna numero de nos
22       void clear(); // deixa a lista vazia, sem nenhum no
23 };
24 #endif
```

List.cpp — Criando uma lista circular

UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ

```
1 #include <iostream>
2 #include "List.h"
3 using std::cout;
4 using std::endl;
5
6 struct Node {
7     int value;
8     Node *next;
9 };
```

List.cpp — Criando uma lista circular

UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ

```
1  #include <iostream>
2  #include "List.h"
3  using std::cout;
4  using std::endl;
5
6  struct Node {
7      int value;
8      Node *next;
9  };
10
11 List::List() {
12     head = nullptr; // lista vazia
13 }
```

List.cpp — Imprimindo uma lista circular

UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ

```
1 void List::print() {
2     // Se a lista for vazia, retorna
3     if (head == nullptr) return;
```

List.cpp — Imprimindo uma lista circular

UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ

```
1  void List::print() {
2      // Se a lista for vazia, retorna
3      if (head == nullptr) return;
4
5      // Senao, percorre a lista imprimindo
6      Node *aux = head;
7      do {
8          cout << aux->value << " ";
9          aux = aux->next;
10     } while (aux != head);
11     cout << endl;
12 }
```

List.cpp — Inserindo no final da lista


```
1 void List::add_back(int x) {
2     // Cria um novo no e inicializa o seu valor
3     Node *novo = new Node;
4     novo->value= x;
```

List.cpp — Inserindo no final da lista


```
1  void List::add_back(int x) {
2      // Cria um novo no e inicializa o seu valor
3      Node *novo = new Node;
4      novo->value= x;
5      // Caso 1: lista vazia
6      if(head == nullptr) {
7          novo->next = novo;
8          head = novo;
9      }
10     // Caso 2: lista nao vazia
```

List.cpp — Inserindo no final da lista

Universidade Federal do Ceará
Campus Quixadá

```
1  void List::add_back(int x) {
2      // Cria um novo no e inicializa o seu valor
3      Node *novo = new Node;
4      novo->value= x;
5      // Caso 1: lista vazia
6      if(head == nullptr) {
7          novo->next = novo;
8          head = novo;
9      }
10     // Caso 2: lista nao vazia
11     else {
12         Node *aux = head;
13         while(aux->next != head)
14             aux = aux->next;
15         novo->next = head;
16         aux->next = novo;
17     }
18 }
```

List.cpp — Inserindo no final da lista


```
1  void List::add_back(int x) {
2      // Cria um novo no e inicializa o seu valor
3      Node *novo = new Node;
4      novo->value= x;
5      // Caso 1: lista vazia
6      if(head == nullptr) {
7          novo->next = novo;
8          head = novo;
9      }
10     // Caso 2: lista nao vazia
11     else {
12         Node *aux = head;
13         while(aux->next != head)
14             aux = aux->next;
15         novo->next = head;
16         aux->next = novo;
17     }
18 }
```

- Qual o tempo de execução dessa função?

List.cpp — Inserindo no início da lista

UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ

```
1 void List::add_front(int x) {
2     // Cria um novo no e inicializa seu valor
3     Node *novo = new Node;
4     novo->value= x;
```

List.cpp — Inserindo no início da lista

UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ

```
1 void List::add_front(int x) {
2     // Cria um novo no e inicializa seu valor
3     Node *novo = new Node;
4     novo->value= x;
5     // Caso 1: lista vazia
6     if(head == nullptr) {
7         novo->next = novo;
8     }
9     // Caso 2: lista nao vazia
```

List.cpp — Inserindo no início da lista

UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ

```
1  void List::add_front(int x) {
2      // Cria um novo no e inicializa seu valor
3      Node *novo = new Node;
4      novo->value= x;
5      // Caso 1: lista vazia
6      if(head == nullptr) {
7          novo->next = novo;
8      }
9      // Caso 2: lista nao vazia
10     else {
11         Node *aux = head;
12         while(aux->next != head)
13             aux = aux->next;
14         novo->next = head;
15         aux->next = novo;
16     }
17     head = novo; // atualiza ponteiro para o primeiro
18 }
```

List.cpp — Inserindo no início da lista

UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ

```
1  void List::add_front(int x) {
2      // Cria um novo no e inicializa seu valor
3      Node *novo = new Node;
4      novo->value= x;
5      // Caso 1: lista vazia
6      if(head == nullptr) {
7          novo->next = novo;
8      }
9      // Caso 2: lista nao vazia
10     else {
11         Node *aux = head;
12         while(aux->next != head)
13             aux = aux->next;
14         novo->next = head;
15         aux->next = novo;
16     }
17     head = novo; // atualiza ponteiro para o primeiro
18 }
```

- Qual o tempo de execução dessa função?

List.cpp — Liberando todos os nós

UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ

```
1  void List::clear() {
2      if(head != nullptr) {
3          Node *aux = head->next;
4          while(aux != head) {
5              Node *t = aux;
6              aux = aux->next;
7              cout << "Removendo " << t->value << endl;
8              delete t;
9          }
10         cout << "Removendo " << head->value << endl;
11         delete head;
12         head = nullptr;
13     }
14 }
```

List.cpp — Liberando todos os nós

UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ

```
1  void List::clear() {
2      if(head != nullptr) {
3          Node *aux = head->next;
4          while(aux != head) {
5              Node *t = aux;
6              aux = aux->next;
7              cout << "Removendo " << t->value << endl;
8              delete t;
9          }
10         cout << "Removendo " << head->value << endl;
11         delete head;
12         head = nullptr;
13     }
14 }
```

```
1 List::~List() {
2     clear();
3 }
```

List.cpp — Buscando um nó com certo valor

UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ

```
1 //Operacao auxiliar: retorna o no com valor x,
2 // ou nullptr caso x nao esteja presente
3 Node *List::search(int x) {
4     if(head == nullptr) // Lista vazia
5         return nullptr;
```

List.cpp — Buscando um nó com certo valor

UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ

```
1  //Operacao auxiliar: retorna o no com valor x,
2  // ou nullptr caso x nao esteja presente
3  Node *List::search(int x) {
4      if(head == nullptr) // Lista vazia
5          return nullptr;
6      Node *aux = head;
7      do {
8          if(aux->value == x)
9              return aux;
10         aux = aux->next;
11     } while (aux != head);
12     return nullptr;
13 }
```

List.cpp — Removendo de lista circular

UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ

```
1 void List::remove(int x) {
2     Node *noRem = search(x);
3     if(noRem == nullptr) return; // lista vazia ou nao tem x
```

List.cpp — Removendo de lista circular

UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ

```
1  void List::remove(int x) {
2      Node *noRem = search(x);
3      if(noRem == nullptr) return; // lista vazia ou nao tem x
4
5      // A lista contem apenas o no a ser removido
6      if(noRem == noRem->next) {
7          delete noRem;
8          head = nullptr;
9          return; // finaliza remocao neste caso
10     }
```

List.cpp — Removendo de lista circular

UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ

```
1  void List::remove(int x) {
2      Node *noRem = search(x);
3      if(noRem == nullptr) return; // lista vazia ou nao tem x
4
5      // A lista contem apenas o no a ser removido
6      if(noRem == noRem->next) {
7          delete noRem;
8          head = nullptr;
9          return; // finaliza remocao neste caso
10     }
11
12     // Percorrer a lista ate achar o antecessor do no
13     Node *aux = noRem;
14     while(aux->next != noRem)
15         aux = aux->next;
16     // Ajusta ponteiros e remove o no com valor x
17     aux->next = noRem->next;
18     if(head == noRem) // Se no removido for head
19         head = noRem->next;
20     delete noRem;
21 }
```

List.cpp — Removendo de lista circular

UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ

```
1 // Remove todos os nos com valor x
2 void List::removeAll(int x) {
```

List.cpp — Removendo de lista circular


```
1 // Remove todos os nos com valor x
2 void List::removeAll(int x) {
3     while(contains(x))
4         remove(x);
5 }
```

List.cpp — Removendo de lista circular

UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ

```
1 // Remove todos os nos com valor x
2 void List::removeAll(int x) {
3     while(contains(x))
4         remove(x);
5 }
```

- Qual a complexidade desta função?

List.cpp — Demais funções


```
1 bool List::contains(int x) {
2     return (search(x) != nullptr);
3 }
```

List.cpp — Demais funções


```
1 bool List::contains(int x) {
2     return (search(x) != nullptr);
3 }
4
5 bool List::empty() {
6     return (head == nullptr);
7 }
```

List.cpp — Demais funções

UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ

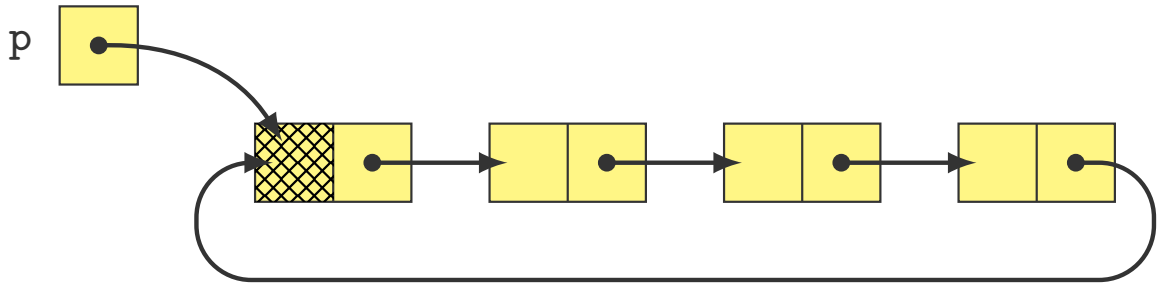
```
1  bool List::contains(int x) {
2      return (search(x) != nullptr);
3  }
4
5  bool List::empty() {
6      return (head == nullptr);
7  }
8
9  int List::size() {
10     if (head == nullptr) return 0;
11     Node *aux = head->next;
12     int s = 1;
13     while(aux != head) {
14         s++;
15         aux = aux->next;
16     }
17     return s;
18 }
```

UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ

Variações

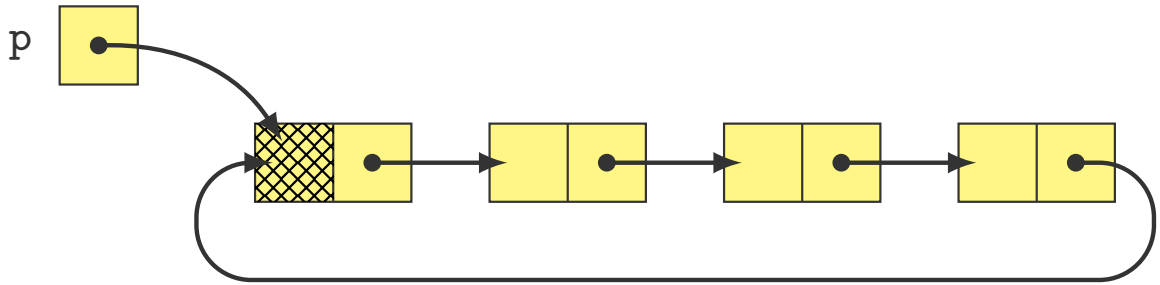
Variações — Listas circulares com nó cabeça

Lista circular com cabeça:

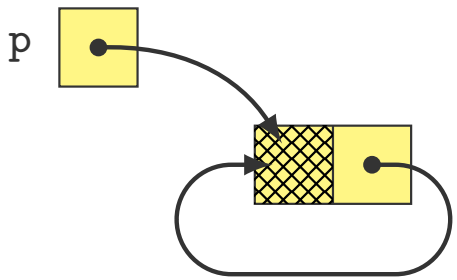


Variações — Listas circulares com nó cabeça

Lista circular com cabeça:

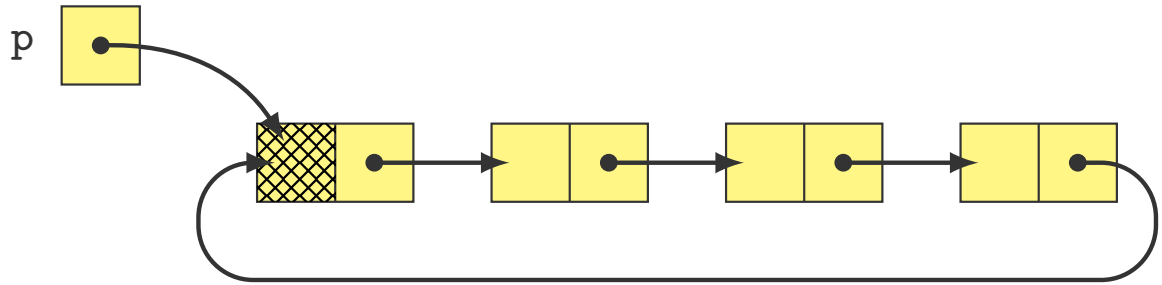


Lista circular com cabeça vazia:

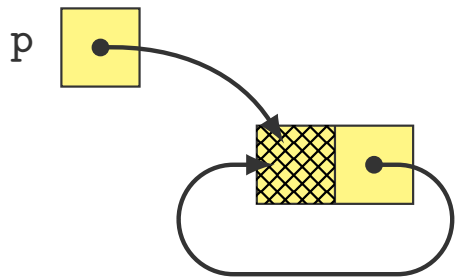


Variações — Listas circulares com nó cabeça

Lista circular com cabeça:



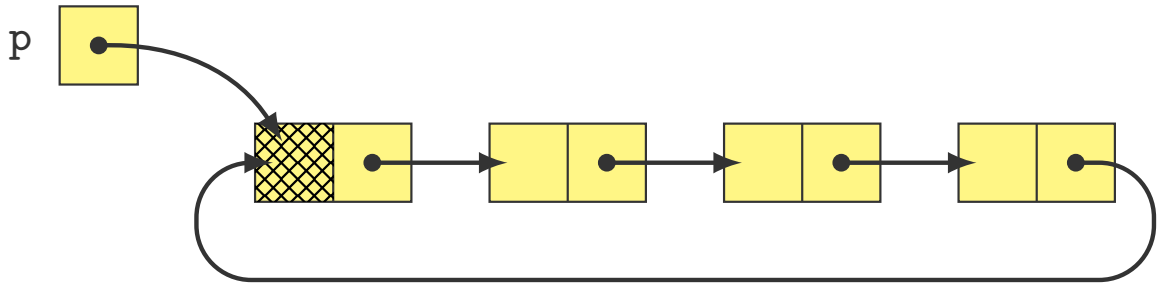
Lista circular com cabeça vazia:



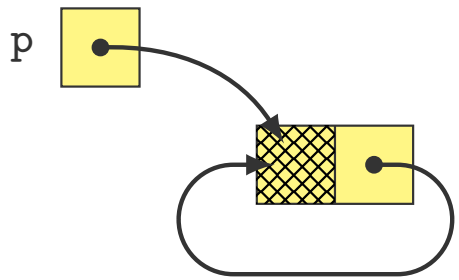
Diferenças para a versão sem cabeça:

Variações — Listas circulares com nó cabeça

Lista circular com cabeça:



Lista circular com cabeça vazia:



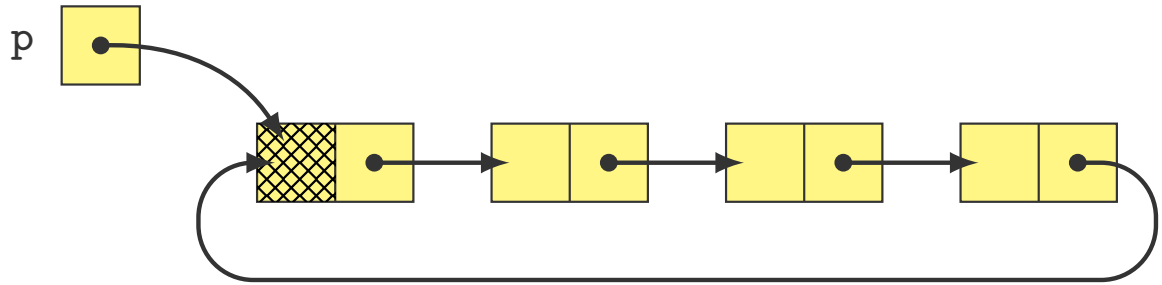
Diferenças para a versão sem cabeça:

- lista sempre aponta para o nó cabeça

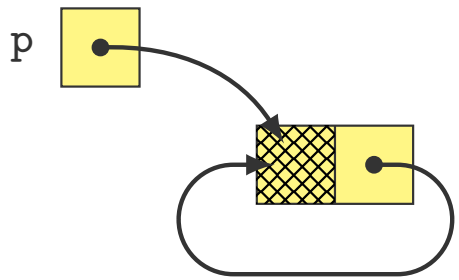
Variações — Listas circulares com nó cabeça

UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ

Lista circular com cabeça:



Lista circular com cabeça vazia:



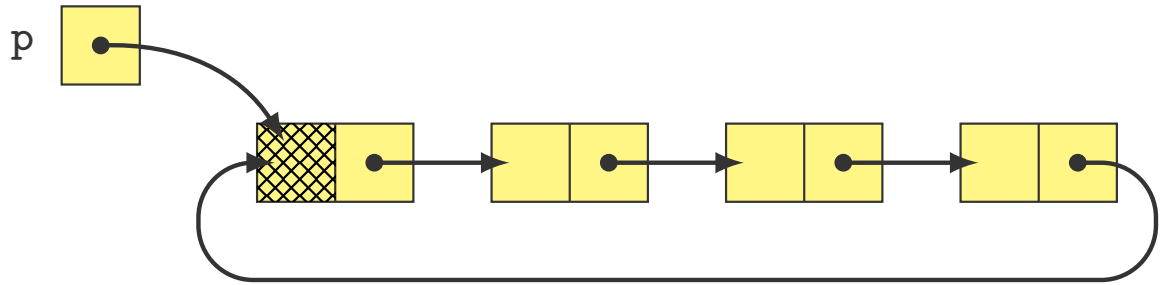
Diferenças para a versão sem cabeça:

- lista sempre aponta para o nó cabeça
- código de inserção e de remoção mais simples

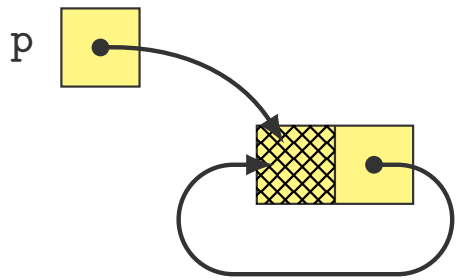
Variações — Listas circulares com nó cabeça

Universidade Federal do Ceará
Campus Quixadá

Lista circular com cabeça:



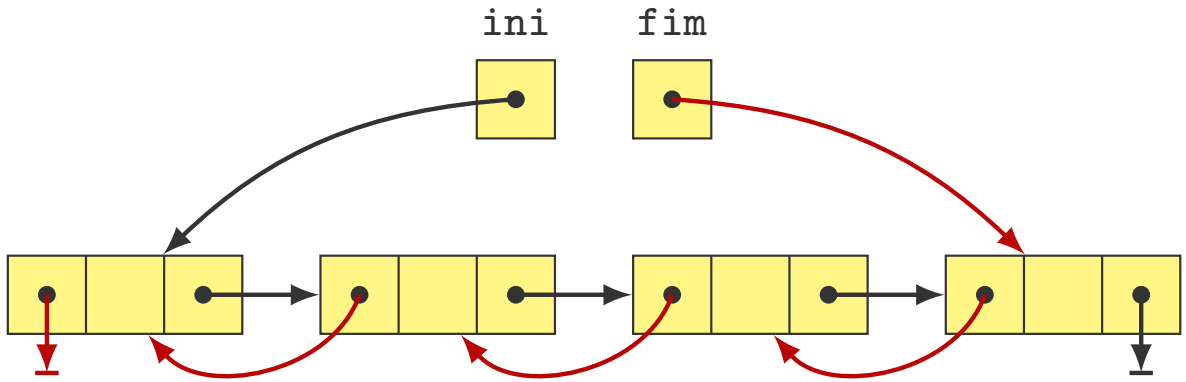
Lista circular com cabeça vazia:



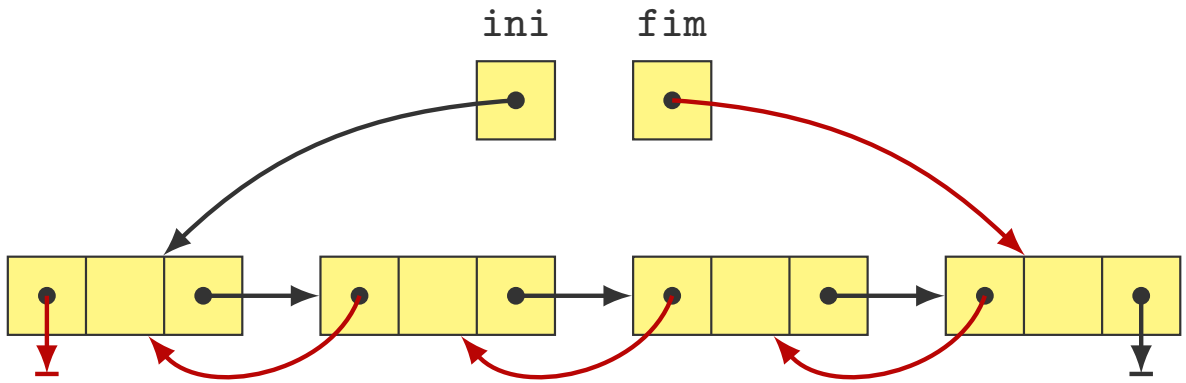
Diferenças para a versão sem cabeça:

- lista sempre aponta para o nó cabeça
- código de inserção e de remoção mais simples
- ao percorrer tem que ignorar cabeça

Variações - Lista duplamente encadeada



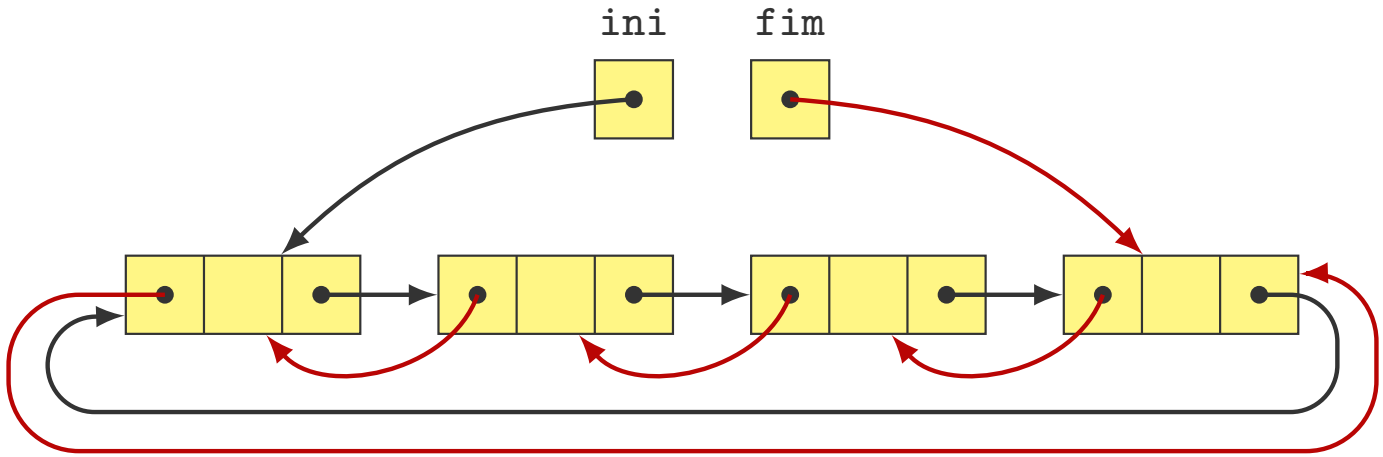
Variações - Lista duplamente encadeada



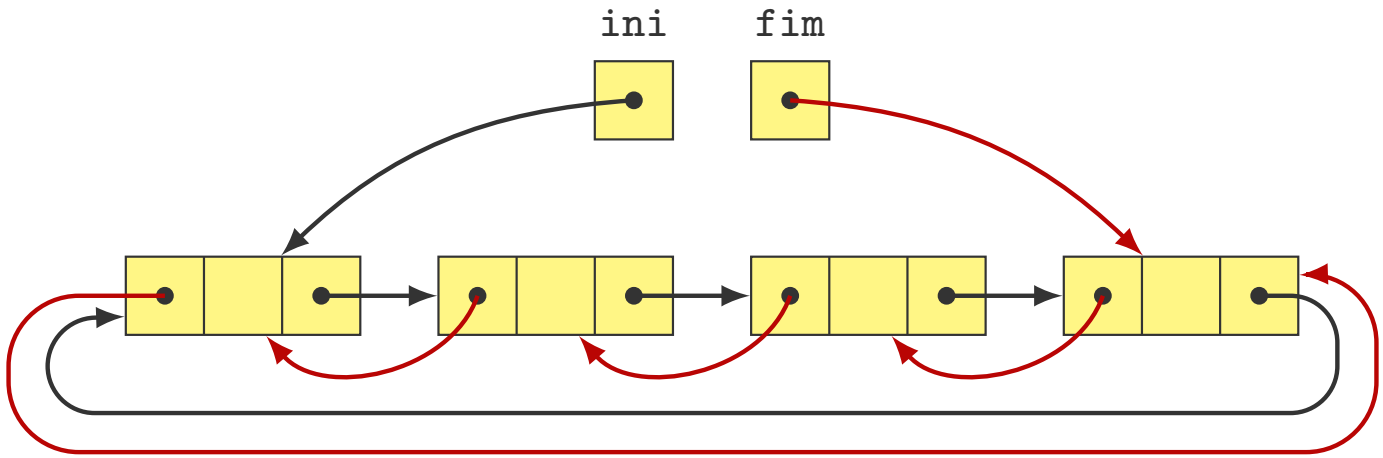
- Cada nó tem um ponteiro para o próximo nó e para o nó anterior.
- Se tivermos um ponteiro para o último elemento da lista, podemos percorrer a lista em ordem inversa, bastando percorrer a lista até alcançar o primeiro elemento da lista, que não tem elemento anterior (seu ponteiro vale nullptr).

Variações - Lista circular duplamente encadeada

UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ



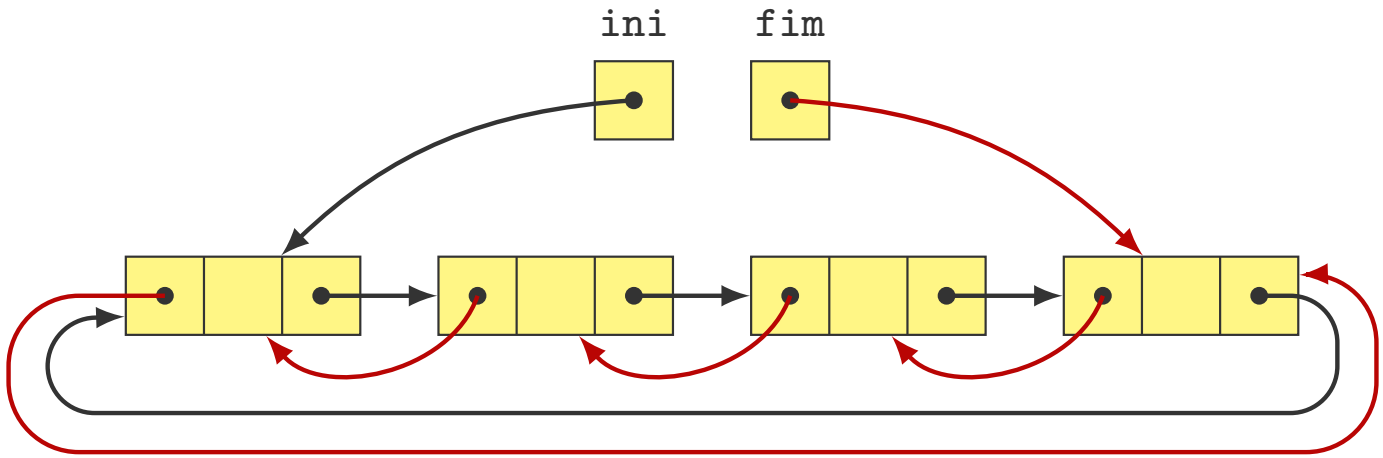
Variações - Lista circular duplamente encadeada



Permite inserção e remoção em $O(1)$

Variações - Lista circular duplamente encadeada

UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ

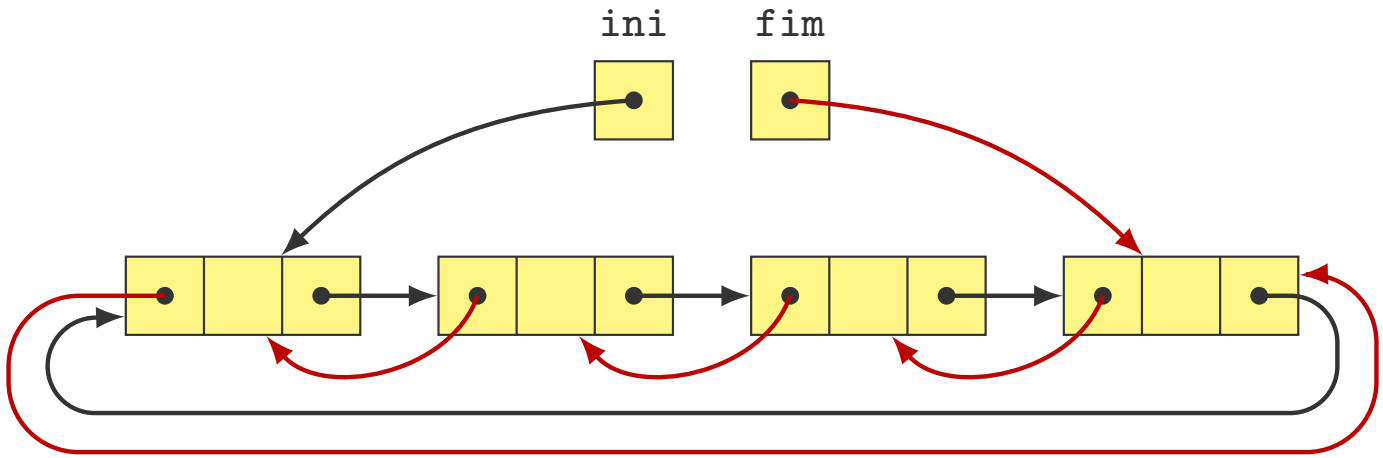


Permite inserção e remoção em $O(1)$

- Variável `fim` é opcional (`fim == ini->ant`)

Variações - Lista circular duplamente encadeada

UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ



Permite inserção e remoção em $O(1)$

- Variável `fim` é opcional (`fim == ini->ant`)

Podemos ter uma lista dupla circular com cabeça também...

Exercícios

Exercícios

- Implemente uma lista duplamente encadeada com as operações:
  - inserir nó
  - remover nó
  - saber se há nó com dado valor
  - tamanho da lista
  - concatenar duas listas
  - imprimir lista de frente para trás ou reversamente

- Implemente uma lista circular duplamente encadeada com as operações:
  - inserir nó
  - remover nó
  - saber se há nó com dado valor
  - tamanho da lista
  - concatenar duas listas
  - imprimir lista de frente para trás ou reversamente

UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ

FIM