Listas em alocação encadeada

Programação II – Engenharia de Telecomunicações

Prof. Emerson Ribeiro de Mello

mello@ifsc.edu.br

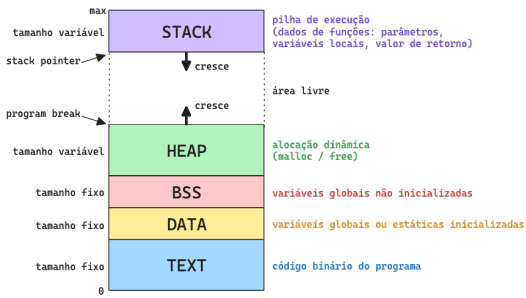
INSTITUTO FEDERAL
Santa Catarina
Campus São José

Licenciamento

Listas lineares em alocação encadeada

Linked list

- Estrutura linear com **alocação dinâmica** (*heap memory*)
  - Em C é necessário fazer uso de `malloc` ou `calloc`
- Nós não precisam estar em posições contíguas da memória (*stack memory*)
- Operações de **inserção e exclusão mais eficientes** se comparadas com vetores

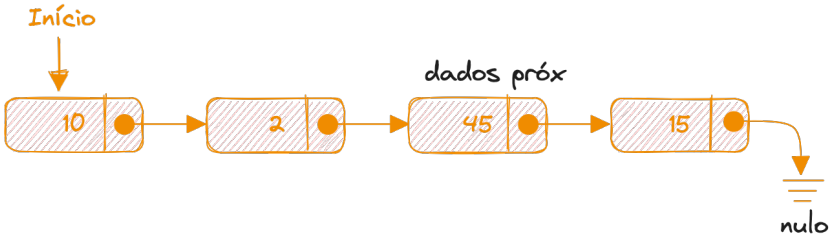


Listas lineares em alocação encadeada

Exemplos de uso em sistemas computacionais

- Filas, pilhas, árvores e grafos
- Caminhos em uma rede de computadores
- Gerenciamento de memória, escalonamento de processos e sistemas de arquivos
- Lista de músicas em um reprodutor de mídia
- Lista de *links* na navegação em páginas HTML

Lista encadeada



- **Cada nó** é composto por uma **área de dados** e por um **ponteiro que aponta para** o endereço de memória do **próximo nó**
- A partir de um nó só é possível ir para o próximo
- É necessário ter um ponteiro para o primeiro nó (**início**)

Operações de inserção e remoção

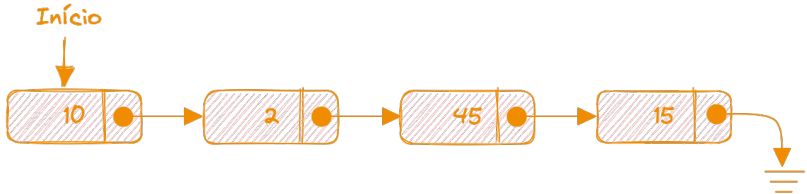
- **Lista encadeada não ordenada**
  - Pode inserir onde está apontando o ponteiro `início`
  - Ao remover, só precisa atualizar os ponteiros. Não há deslocamento
- **Lista encadeada ordenada**
  - Ao inserir, só precisa atualizar os ponteiros. Não há deslocamento
  - Ao remover, só precisa atualizar os ponteiros. Não há deslocamento

---

- Para inserir ou remover é necessário buscar pelo elemento
- A **inserção** e **remoção** tem complexidade $O(1)$,
- A **busca** tem complexidade $\Theta(n)$ no pior caso

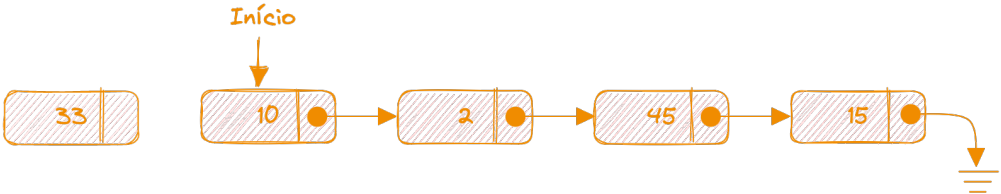
Lista encadeada

Inserção de um novo nó – complexidade $O(1)$



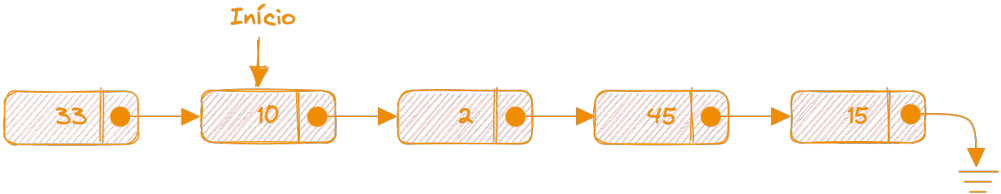
Lista encadeada

Inserção de um novo nó – complexidade $O(1)$



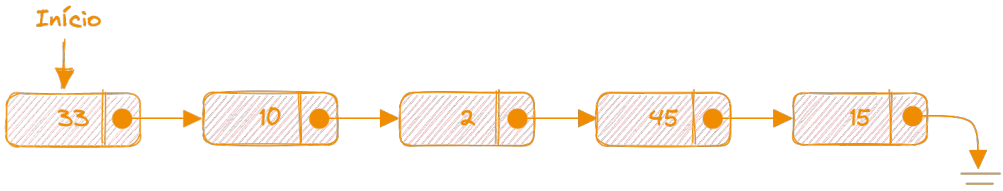
Lista encadeada

Inserção de um novo nó – complexidade $O(1)$



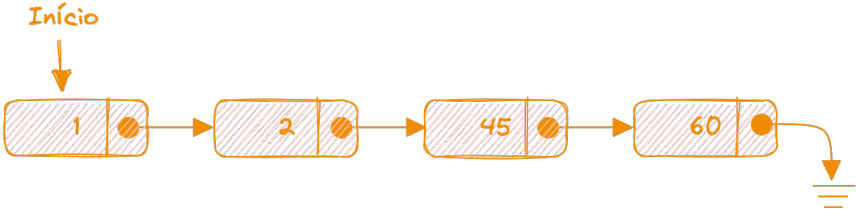
Lista encadeada

Inserção de um novo nó – complexidade $O(1)$



Lista encadeada ordenada

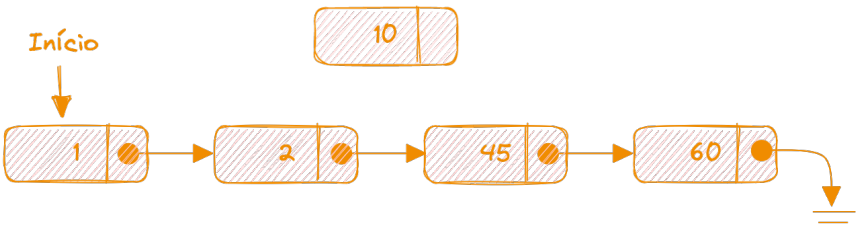
Inserção de um novo nó – complexidade $O(1)$ + complexidade da busca $O(n)$



- O dado de cada nó indica a ordem do nó na lista

Lista encadeada ordenada

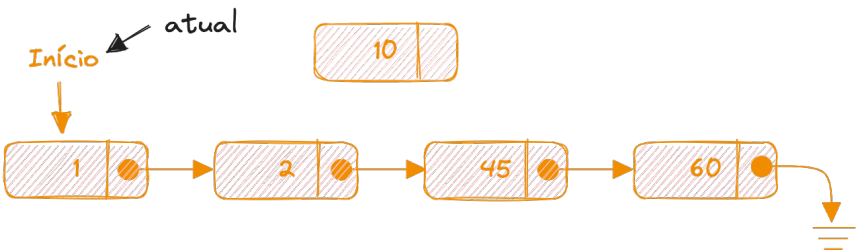
Inserção de um novo nó – complexidade $O(1)$ + complexidade da busca $O(n)$



- O dado de cada nó indica a ordem do nó na lista

Lista encadeada ordenada

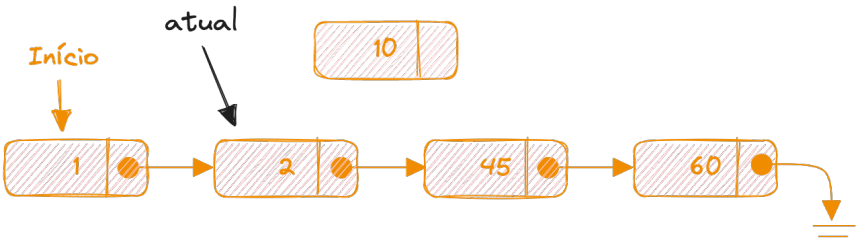
Inserção de um novo nó – complexidade $O(1)$ + complexidade da busca $O(n)$



- O dado de cada nó indica a ordem do nó na lista

Lista encadeada ordenada

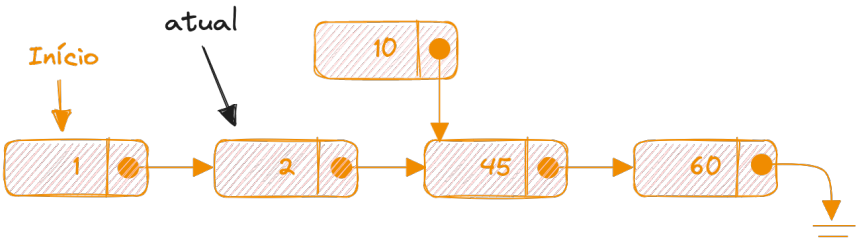
Inserção de um novo nó – complexidade $O(1)$ + complexidade da busca $O(n)$



- O dado de cada nó indica a ordem do nó na lista

Lista encadeada ordenada

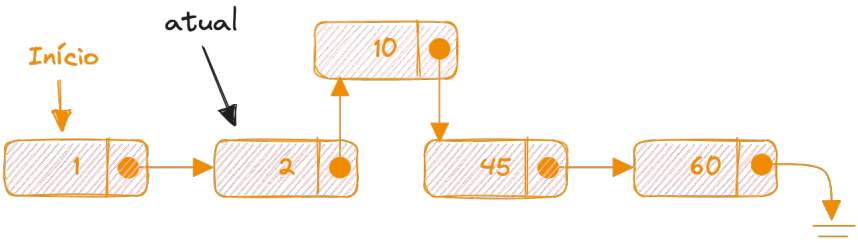
Inserção de um novo nó – complexidade $O(1)$ + complexidade da busca $O(n)$



- O dado de cada nó indica a ordem do nó na lista

Lista encadeada ordenada

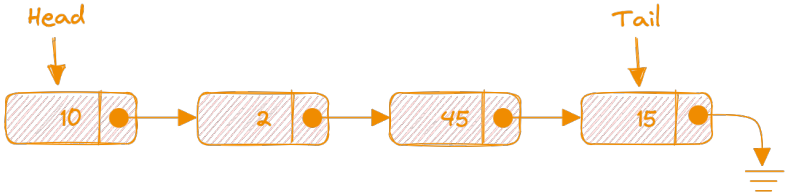
Inserção de um novo nó – complexidade $O(1)$ + complexidade da busca $O(n)$



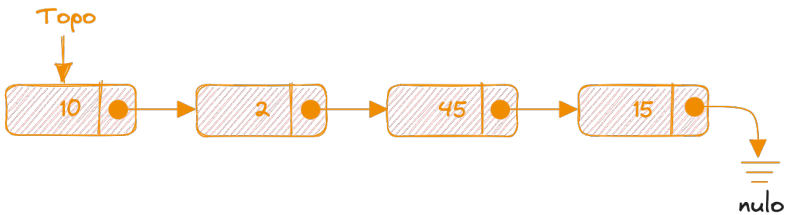
- O dado de cada nó indica a ordem do nó na lista

Fila e pilha por meio de lista encadeada

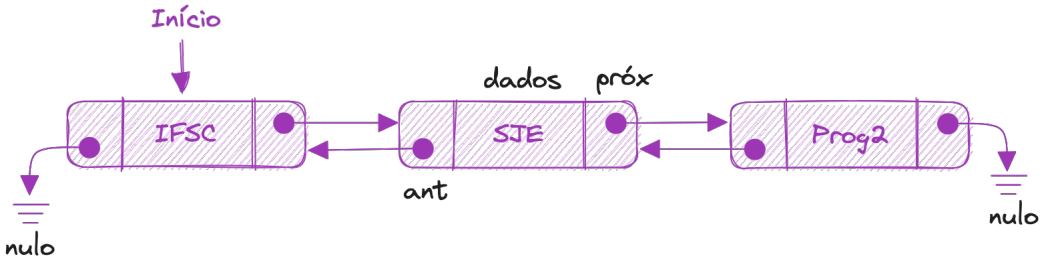
- Fila



- Pilha

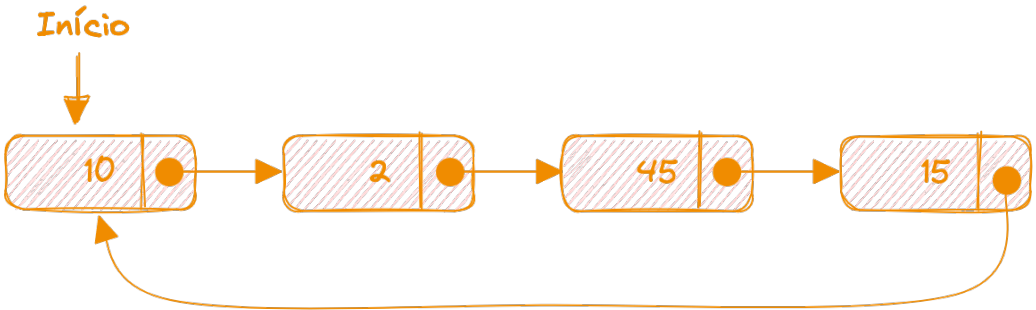


Lista duplamente encadeada

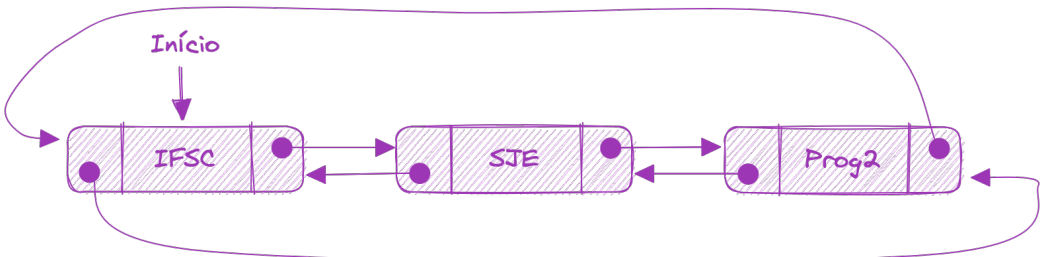


- Cada nó é composto por uma **área de dados** e por **dois ponteiros** que apontam para os endereços de memória do nó anterior e do próximo nó
- A partir de um nó é possível ir navegar para o próximo ou para o anterior

Lista circular encadeada



Lista circular duplamente encadeada



Implementação de lista encadeada na linguagem C

```c
typedef struct no{
    int dado;
    struct no* proximo;
} no_t;

// inserir novo nó. O parâmetro inicio é um ponteiro para
      um ponteiro de no_t
void adicionar(no_t** inicio, int dado){
    no_t* novo = (no_t*) malloc(sizeof (no_t));
    novo->dado = dado;
    novo->proximo = *inicio;
    *inicio = novo;
}

// buscar por dados
no_t* buscar(no_t* inicio, int dado){
    // ...
}
```

```c
bool remover(no_t** inicio, int dado){
    no_t* atual = *inicio;
    no_t* anterior = NULL;
    while(atual != NULL){
        if (atual->dado == dado){
            if (anterior == NULL) {
                *inicio = atual->proximo;
            } else {
                anterior->proximo = atual->proximo;
            }
            free(atual);
            return true;
        }
        anterior = atual;
        atual = atual->proximo;
    }
    return false;
}
```

Em C, ao remover um nó, garanta que não resultará em vazamento de memória (*memory leak*)

Implementação de lista encadeada na linguagem C

```c
void destruir(no_t** inicio){
    no_t* atual = *inicio;
    no_t* prox;
    while(atual != NULL){
        prox = atual->proximo;
        free(atual);
        atual = prox;
    }
    *inicio = NULL;
}
```

```c
int main(){
    // inicio precisa ser um ponteiro
    no_t* inicio = NULL;

    // adicionando
    adicionar(&inicio, 1);
    adicionar(&inicio, 2);
    adicionar(&inicio, 3);

    // removendo
    remover(&inicio, 2);

    // listando
    no_t* atual = inicio;
    while(atual != NULL){
        printf("%d ", atual->dado);
        atual = atual->proximo;
    }

    destruir(&inicio);

    return 0;
}
```

Listas de alocação sequencial *vs* Listas encadeadas

Arranjos

- Inserção e remoção são custosas
- Acesso aleatório (busca pelo índice) com tempo constante
- Simples de ordenar
- Tamanho fixo

Listas encadeadas

- Inserção e remoção são simples
- Busca é custosa (linear)
- Custoso para ordenar
- Tamanho dinâmico

Exercício 1

- Evolua a biblioteca `libprg`, criada por você na aula de lista de sequencial, para ofertar as seguintes estruturas de dados (para armazenar números inteiros):
  a. Lista circular encadeada (não ordenada e ordenada)
  b. Lista circular duplamente encadeada (não ordenada e ordenada)
  c. Fila representada em uma lista encadeada
  d. Pilha representada em uma lista encadeada
- As definições das funções devem ser feitas obrigatoriamente no arquivo de cabeçalho `libprg.h`
- As implementações das funções obrigatoriamente no arquivo com o nome `lista_encadeada.c`

Exercício 2

- Crie um projeto com CMake de uma aplicação em C que dependa da biblioteca `libprg`, criada no exercício anterior
  - Faça uso do `FetchContent` no CMake para baixar a dependência
- Faça um menu interativo que permita ao usuário executar as seguintes operações
  1. Criar qualquer uma das estruturas de dados do exercício anterior
  2. Inserir elemento na estrutura de dados
  3. Remover elemento da estrutura de dados
  4. Buscar por um elemento

Referências

CORMEN, Thomas H. et al. **Algoritmos: teoria e prática**. LTC, 2012. Disponível em: <https://app.minhabiblioteca.com.br/reader/books/9788595158092>.