Universidade Federal do ABC

CMCC Centro de Matemática, Computação e Cognição

Algoritmos e Estruturas de Dados I

Estruturas Lineares: Listas, Pilhas e Filas

Mirtha Lina Fernández Venero
Sala 529-2, Bloco A
mirtha.lina@ufabc.edu.br
http://professor.ufabc.edu.br/~mirtha.lina/eadi.html

15 de março de 2019

Agenda

Aulas Anteriores: Estruturas Lineares

Tipos de dados abstratos que permitem armazenar uma sequência de elementos de dados do mesmo tipo de forma **linear**, i.e.

- Todo elemento exceto o primeiro tem **um** elemento anterior
- Todo elemento exceto o último tem **um** elemento sucessor

Podem ser implementadas de forma **sequencial** ou **encadeada**

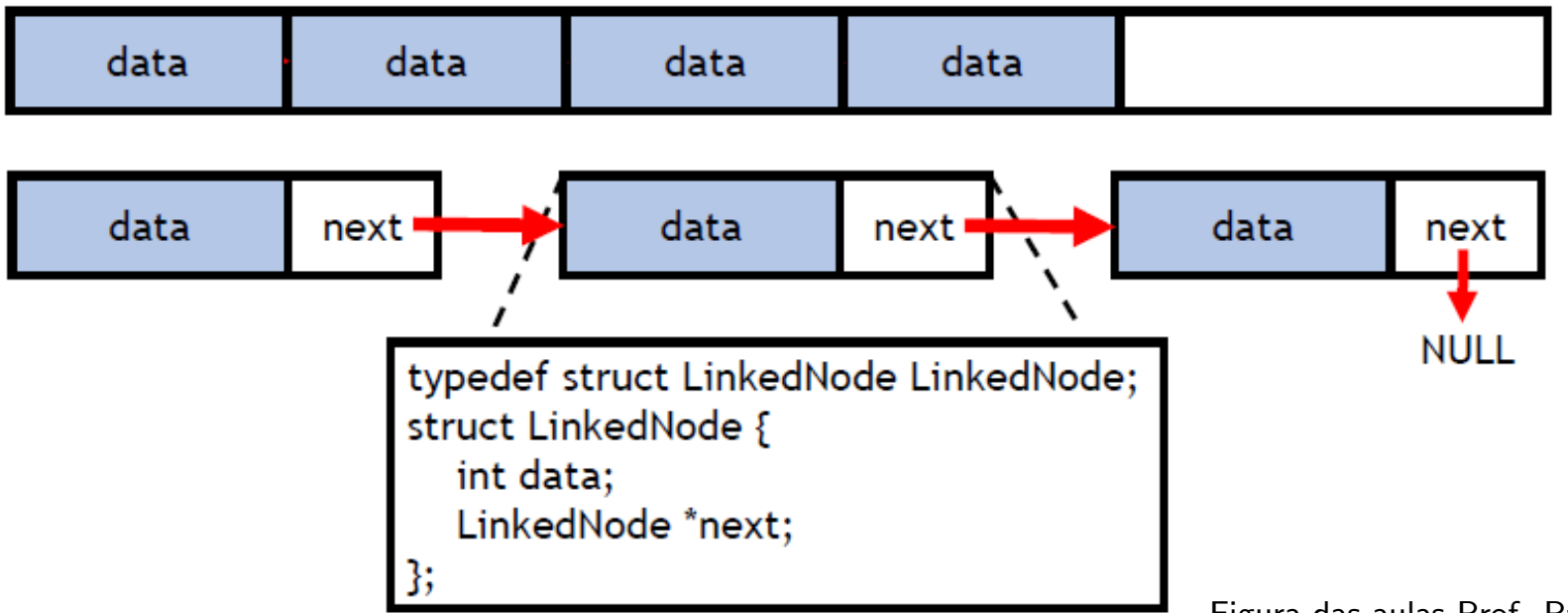


Figura das aulas Prof. Paulo Pisani

Universidade Federal do ABC

CMCC Centro de Matemática, Computação e Cognição

Operações Básicas sobre Estruturas Lineares

- Percorrer todos os elementos para e.g. mostrar
- **inserir** um elemento: dependendo dum índice ou uma condição
- **remover** um elemento: dependendo dum índice ou uma condição

Outras: copiar, unir, dividir, remover todos os elementos...

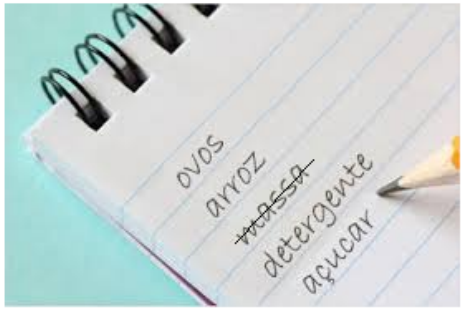
Operações Básicas sobre Estruturas Lineares

- Percorrer todos os elementos para e.g. mostrar
- **inserir** um elemento: dependendo dum índice ou uma condição
- **remover** um elemento: dependendo dum índice ou uma condição

Outras: copiar, unir, dividir, remover todos os elementos...

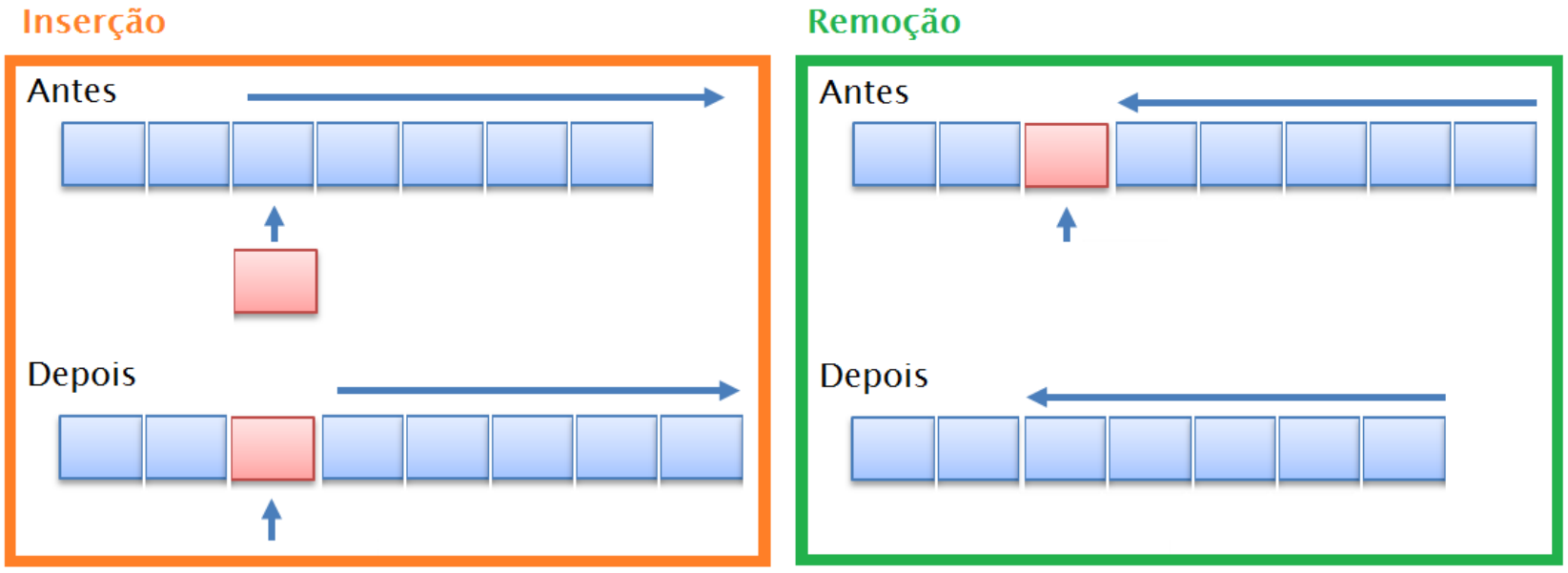
Lista: Estrutura linear que permite

- permite acessar o elemento na posição k
- inserção e remoção em qualquer posição!



Operações em Listas com implementação sequencial

- Acessar elemento na posição k: Direto
- Inserção e remoção: precisa de deslocamentos



Universidade Federal do ABC

CMCC Centro de Matemática, Computação e Cognição

O que fazer quando o vetor está cheio?

Solução 1: Reportar um erro ou lançar uma exceção

Solução 2: Em muitos casos há memória disponível no sistema; por que não auto-ajustar?

Quanto e quando ajustar? Como garantir que o auto-ajuste não aconteça frequentemente? Como evitar ter muita memória sem usar?

Universidade Federal do ABC

CMCC Centro de Matemática, Computação e Cognição

O que fazer quando o vetor está cheio?

Solução 1: Reportar um erro ou lançar uma exceção

Solução 2: Em muitos casos há memória disponível no sistema; por que não auto-ajustar?

Quanto e quando ajustar? Como garantir que o auto-ajuste não aconteça frequentemente? Como evitar ter muita memória sem usar?

- *inserir*: dobrar o tamanho do vetor se a lista está na capacidade máxima
- *remover*: reduzir à metade o tamanho do vetor se a lista está num quarto da capacidade

O que fazer quando o vetor está cheio?

Solução 1: Reportar um erro ou lançar uma exceção

Solução 2: Em muitos casos há memória disponível no sistema; por que não auto-ajustar?

Quanto e quando ajustar? Como garantir que o auto-ajuste não aconteça frequentemente? Como evitar ter muita memória sem usar?

- *inserir*: dobrar o tamanho do vetor se a lista está na capacidade máxima
- *remover*: reduzir à metade o tamanho do vetor se a lista está num quarto da capacidade

Exercício para casa: Implementação de um vetor auto-ajustável

Operações em Listas com implementação encadeada

- Acessar elemento na posição k: Precisa percorrer $k-1$ nós
- Inserção e remoção: É necessário
  1. Distinguir o caso em que a lista está vazia ou for o primeiro
  2. Achar o nó anterior à posição desejada (se existir)

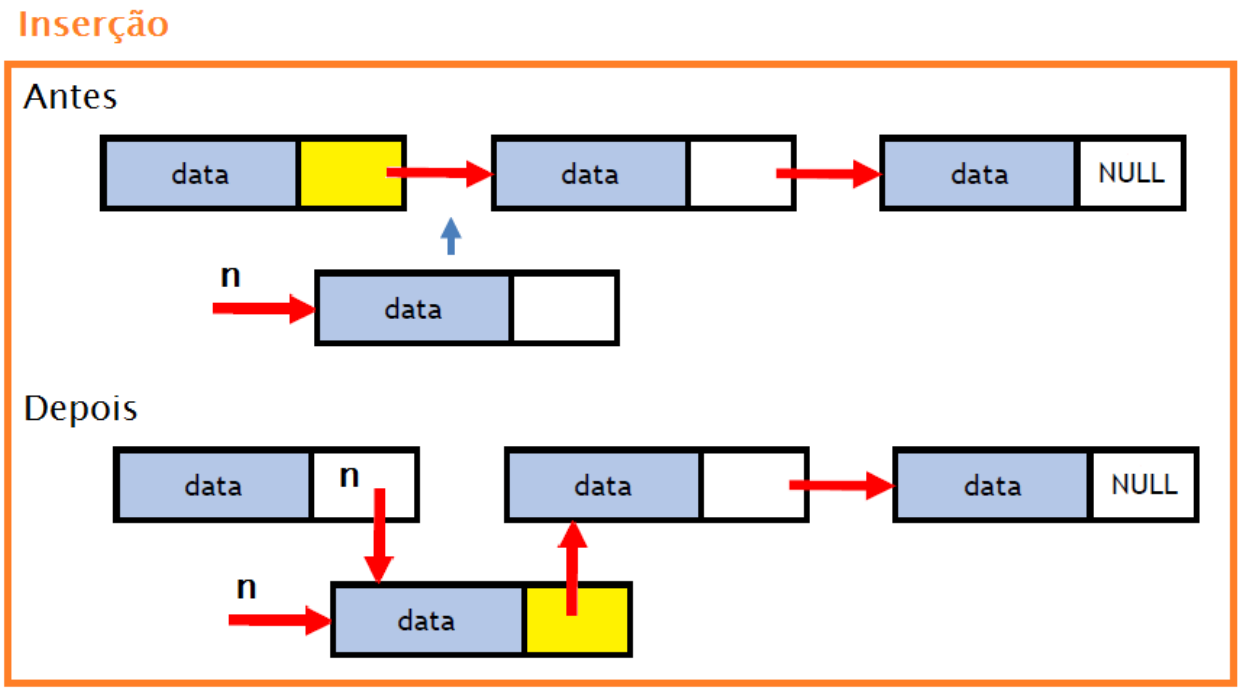


Figura das aulas Prof. Paulo Pisani

Operações em Listas com implementação encadeada

- Acessar elemento na posição k: Precisa percorrer $k-1$ nós
- Inserção e remoção: É necessário
  1. Distinguir o caso em que a lista está vazia ou for o primeiro
  2. Achar o nó anterior à posição desejada (se existir)

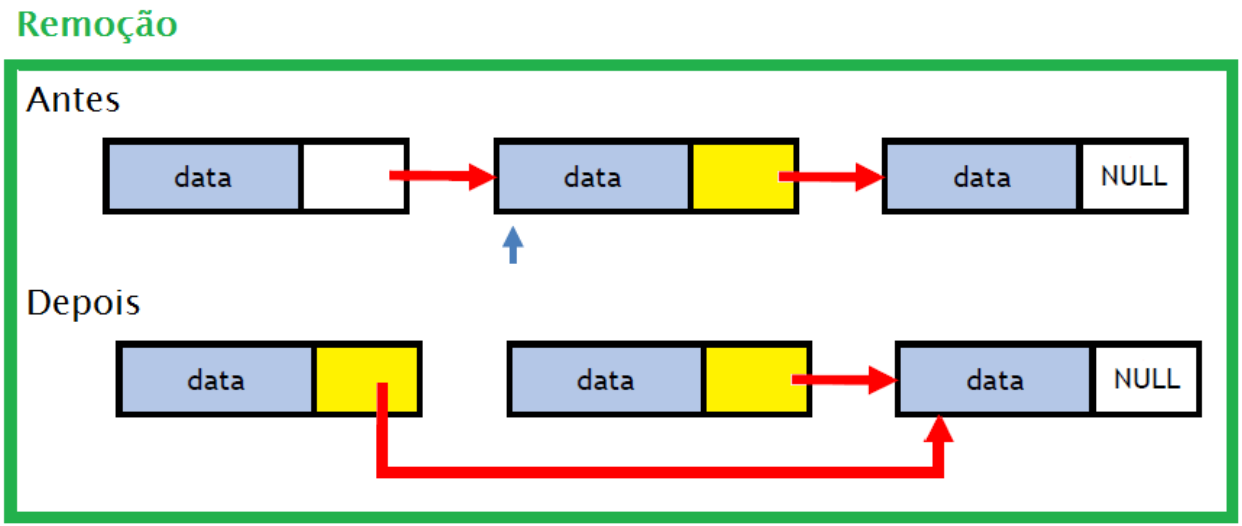
Exemplo: Mude a seguinte função para inserir após um nó

```
linked_node *append_node_1(linked_node *ultimo, int valor) {
    linked_node *novo = malloc(sizeof(linked_node));
    if (novo == NULL) return NULL;
    novo->data = valor;
    novo->next = NULL;

    if (ultimo != NULL) ultimo->next = novo;

    return novo;
}
```

Figura das aulas Prof. Paulo Pisani

Operações em Listas com implementação encadeada

- Acessar elemento na posição k: Precisa percorrer $k-1$ nós
- Inserção e remoção: É necessário
  1. Distinguir o caso em que a lista está vazia ou for o primeiro
  2. Achar o nó anterior à posição desejada (se existir)



3. Em linguagens sem coleta automática de lixo, **importante** não esquecer liberar o nó ao remover.

Figura das aulas Prof. Paulo Pisani

Operações em Listas com implementação encadeada

- Acessar elemento na posição k: Precisa percorrer $k-1$ nós
- Inserção e remoção: É necessário
  1. Distinguir o caso em que a lista está vazia ou for o primeiro. **Pode ser evitado usando uma lista com nó fantasma como primeiro!**

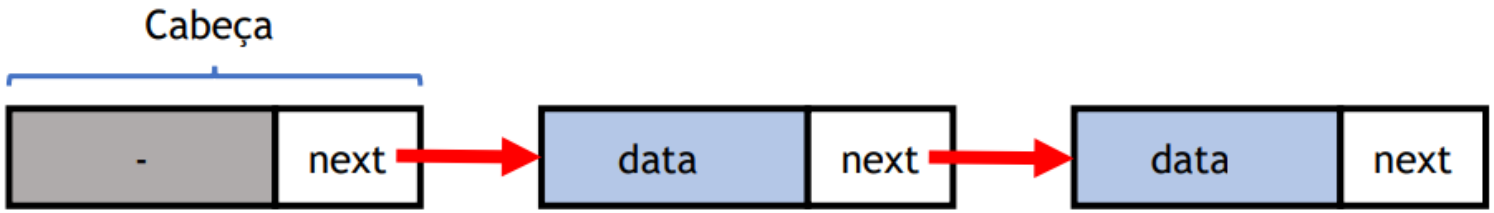


Figura das aulas Prof. Paulo Pisani

Operações em Listas com implementação encadeada

- Acessar elemento na posição k: Precisa percorrer $k-1$ nós
- Inserção e remoção: É necessário
  1. Distinguir o caso em que a lista está vazia ou for o primeiro. **Pode ser evitado usando uma lista com nó fantasma como primeiro!**
  2. Achar o nó anterior à posição desejada (se existir). **Pode ser "aliviado"com nós duplamente enlaçados!**

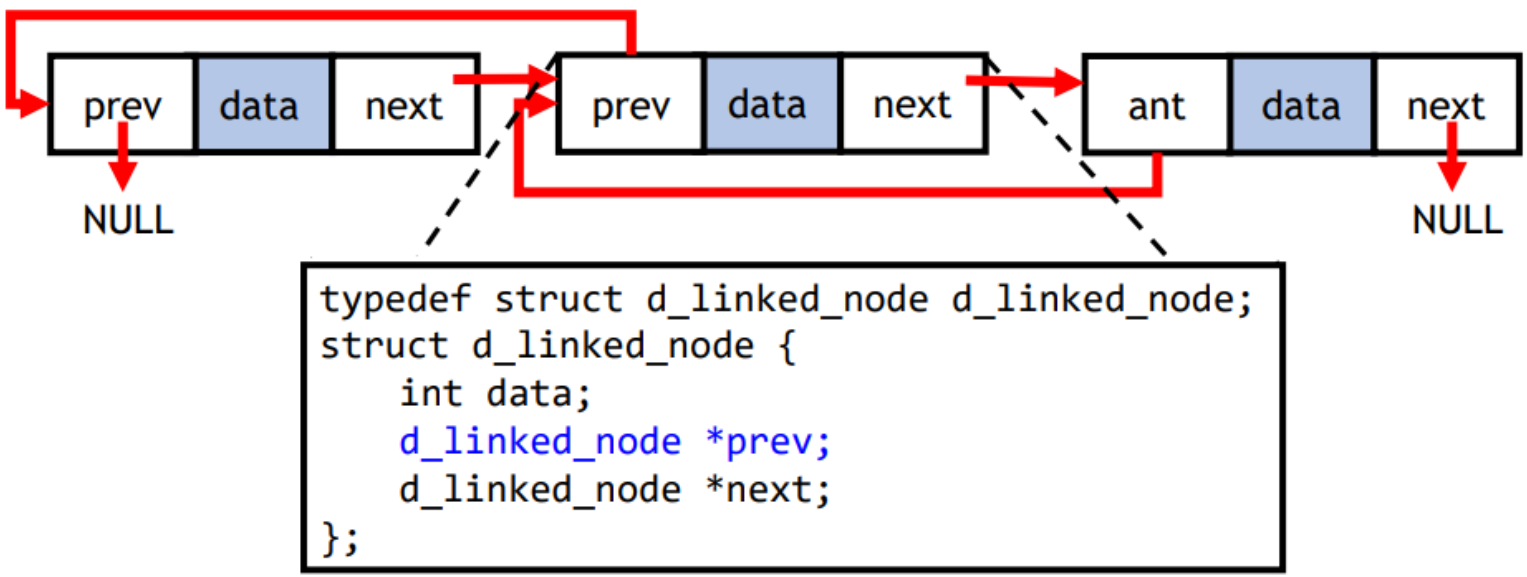


Figura das aulas Prof. Paulo Pisani

Resumo Listas

Listas com arranjos

- Simples para usar
- Alocação em bloco contínuo
- Acesso a um item em tempo constante

- Requer saber a quantidade de itens previamente (para alocaçao)
- Inserção/Remoção requer deslocamentos
- Expansão custosa (realocar e copiar)

Listas ligadas/encadeadas/enlaçadas

- Não requer conhecer a quantidade de itens previamente
- Inserção e remoção não requer deslocamentos

- Acesso a uma posição necessita percorrer a lista
- Memória extra para os ponteiros

Figura das aulas Prof. Paulo Pisani

Resumo Listas

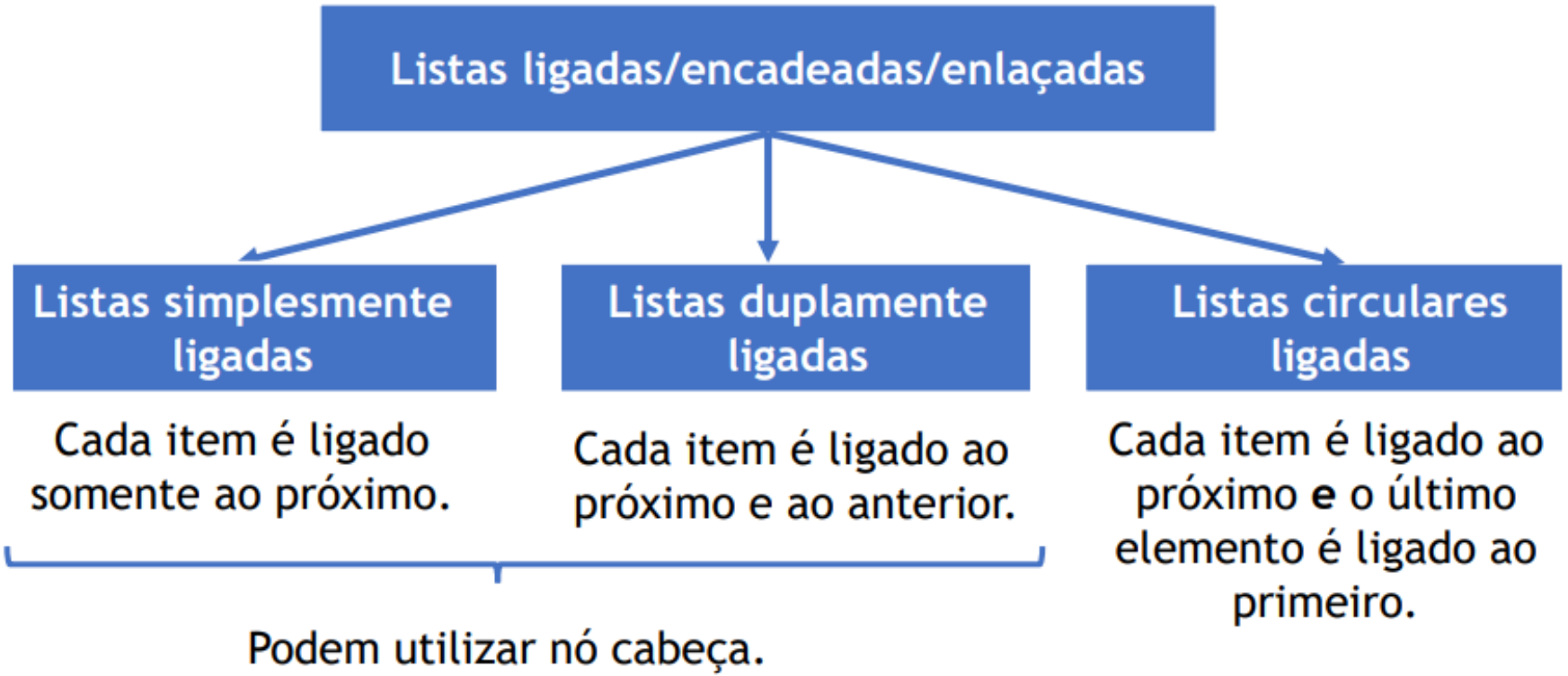


Figura das aulas Prof. Paulo Pisani

Outros tipos de Estruturas Lineares: Pilhas e Filas

Duas importantes variantes de menor complexidade que resultam de restringir as operações de inserção e remoção

Pilha: Tipo abstrato de dados que inclui duas operações fundamentais: a inserção (*push* - empilhar) e a remoção (*pop* - desempilhar)

- ambas operações são sempre realizadas por um único extremo (início/final=*topo* - *top*) $\Rightarrow$ sempre é removido o elemento **mais** recentemente inserido
- também conhecidas como listas LIFO = "*last in first out*"

Outros tipos de Estruturas Lineares: Pilhas e Filas

Duas importantes variantes de menor complexidade que resultam de restringir as operações de inserção e remoção

Fila: Tipo abstrato de dados que inclui duas operações fundamentais: a inserção (*enqueue* - enfileirar) e a remoção (*dequeue* - desenfileirar)

- a inserção é sempre realizada pelo final enquanto a remoção é sempre realizada pelo início $\Rightarrow$ sempre é removido o elemento **menos** recentemente inserido
- também conhecidas como listas FIFO = "*first in first out*"

Universidade Federal do ABC

CMCC
Centro de Matemática, Computação e Cognição

Agenda

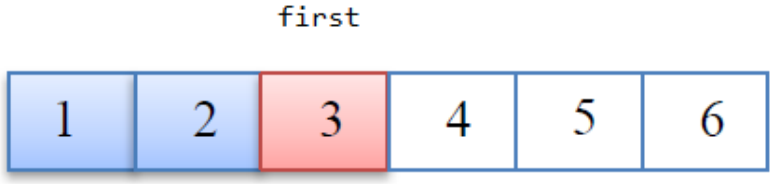
Exemplos de Pilhas



- nos compiladores: análise sintática, semântica, para lidar com funções recursivas (exemplo fatorial e imprimir lista)
- balanceamento de símbolos auxiliares em expressões, tags em HTML and XML,
- implementar botões de avançar/voltar dos navegadores, desfazer/refazer dos editores

Implementação de pilha usando vetores

- Mesma estrutura de dados dum vetor; por isso é preciso armazenar a capacidade e uma variável que indica a posição do topo da pilha

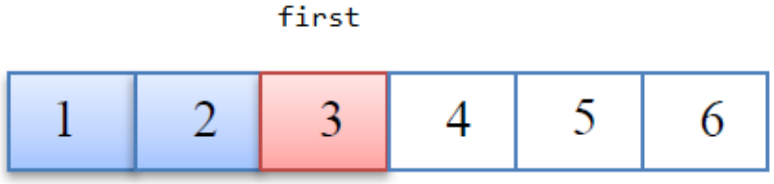


Alguns cuidados

- caso o tamanho dos dados for maior do que `sizeof(*void)`, o tipo base do vetor deve ser um ponteiro a eles

Implementação de pilha usando vetores

- Mesma estrutura de dados dum vetor; por isso é preciso armazenar a capacidade e uma variável que indica a posição do topo da pilha



Alguns cuidados

- caso o tamanho dos dados for maior do que `sizeof(*void)`, o tipo base do vetor deve ser um ponteiro a eles
- Ao desempilhar, considerar o caso da pilha estar vazia.
- Ao empilhar, considerar o caso da pilha estar cheia
- Liberar toda a memória ao terminar

Exemplo de implementação de pilha usando vetores

```c
#include "stdio.h"
#include "stdlib.h"

#define MINCAPACITY  15

typedef struct arrStack arrStack;

struct arrStack{
  int top;
  int capacity;
  int *data;
};

arrStack *createArrStack()
{
  arrStack *temp = malloc(sizeof(arrStack));
  if (!temp)
    return NULL;
  temp->data =  malloc(MINCAPACITY * sizeof(int));
  if (!temp->data){
    free(temp);
    return NULL;
  }
  temp->top = 0;
  temp->capacity = MINCAPACITY;
  return temp;
}
```

```c
int isEmpty(arrStack *stack)
{
  return !stack || !stack->top;
}

int isFull(arrStack *stack)
{
  return stack->top == stack->capacity;
}

int push(arrStack *stack, int elem)
{
  if(isFull(stack))
    return 0;                 //return 0 if full
  stack->data[stack->top] = elem;
  return ++stack->top;        // returns the new size
}

int pop(arrStack *stack, int *elem)
{
  if(isEmpty(stack))
    return -1;                //return -1 if empty
  *elem = stack->data[--stack->top];
  return stack->top;          // returns the new size
}

void deleteStack(arrStack  *stack){ ... }
```

Implementação básica de pilha enlaçada

- Mesma estrutura de dados duma lista simplesmente ligada


```
#include <stdio.h>
#include <stdlib.h>

typedef struct LinkedNode LinkedNode;
struct LinkedNode {
    int data;
    LinkedNode *next;
};

    ...

    LinkedNode *first = NULL;

    ...
```

O ponteiro para o primeiro item deve ser salvo.

Implementação básica de pilha enlaçada push(1)

- Mesma estrutura de dados duma lista simplesmente ligada
  Como empilhar e desempilhar?

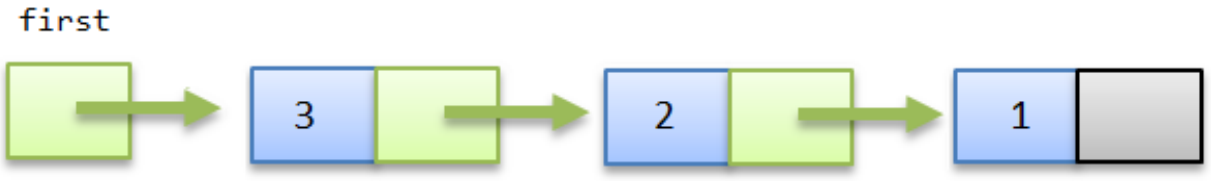
Implementação básica de pilha enlaçada push(2)

- Mesma estrutura de dados duma lista simplesmente ligada
  Como empilhar e desempilhar?

- Ao **empilhar**, considerar se há memória disponível, criar o novo nó e inserir no topo

Implementação básica de pilha enlaçada push(3)

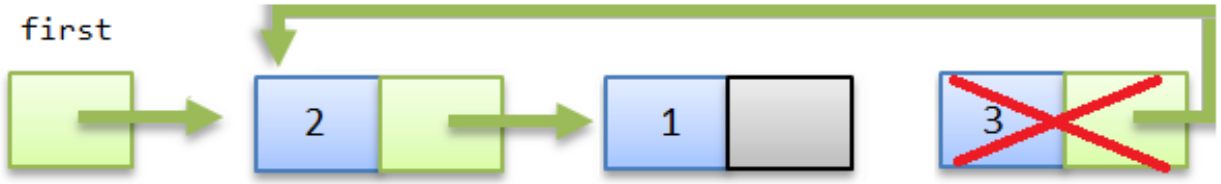
- Mesma estrutura de dados duma lista simplesmente ligada
  Como empilhar e desempilhar?



- Ao **empilhar**, considerar se há memória disponível, criar o novo nó e inserir no topo

Implementação básica de pilha enlaçada pop()

- Mesma estrutura de dados duma lista simplesmente ligada
  Como empilhar e desempilhar?



- Ao **empilhar**, considerar se há memória disponível, criar o novo nó e inserir no topo
- Ao **desempilhar**, considerar o caso da pilha estar vazia e sinalizar (retornar `NULL`, booleano ou valor apropriado, lançar exceção, etc). Em outro caso, salvar/retornar o dado no topo e atualizar o `first`. Em linguagens sem coleta automática de lixo, **não esquecer liberar o nó**.

Implementação básica de pilha enlaçada push(4)

- Mesma estrutura de dados duma lista simplesmente ligada
  Como empilhar e desempilhar?

- Ao **empilhar**, considerar se há memória disponível, criar o novo nó e inserir no topo
- Ao **desempilhar**, considerar o caso da pilha estar vazia e sinalizar (retornar `NULL`, booleano ou valor apropriado, lançar exceção, etc). Em outro caso, salvar/retornar o dado no topo e atualizar o `first`. Em linguagens sem coleta automática de lixo, **não esquecer liberar o nó**.

Exemplo de implementação de pilha usando listas ligadas

```
11  LinkedNode *push(LinkedNode* top, int elem) {
12      LinkedNode *tmp = malloc(sizeof(LinkedNode));
13      if (tmp == NULL)
14        return NULL;       // not enough memory
15
16      tmp->data = elem;
17      if(top == NULL)      // stack is empty
18        tmp->next = NULL;
19      else
20        tmp->next = top;
21      return tmp;          // returns the new top
22  }
23
24  LinkedNode *pop(LinkedNode* top, int *elem) {
25      if (top == NULL)
26        return NULL;
27      *elem = top->data;
28      LinkedNode *tmp = top->next;
29      free(top);
30      return tmp;        // returns the new top
31  }
```

```
33  void deleteStack(LinkedNode* top) {
34    LinkedNode *tmp;
35    while(top != NULL){
36      tmp = top;
37      top = top->next;
38      free(tmp);
39    }
40  }
41
42  int main(void) {
43    LinkedNode *stack = NULL; int i;
44
45    stack = push(stack, 1);
46    stack = push(stack, 145);
47    stack = pop(stack, &i);
48    printf("%d\n", i);
49    printf("%d\n", stack->data);
50    deleteStack(stack);
51    return 0;
52  }
```

Análise das Operações

- **Implementação enlaçada**: O custo das operações é constante. Precisa memória extra para os apontadores.
- **Implementação sequencial**: O custo das operações é constante. Capacidade máxima é um valor fixo
- **Implementação usando vetor auto-ajustável**: Menos espaço desperdiçado. No pior caso, o custo das operações é linear; porém

Análise das Operações

- **Implementação enlaçada**: O custo das operações é constante. Precisa memória extra para os apontadores.
- **Implementação sequencial**: O custo das operações é constante. Capacidade máxima é um valor fixo
- **Implementação usando vetor auto-ajustável**: Menos espaço desperdiçado. No pior caso, o custo das operações é linear; porém

  Análise amortizado: Cálculo do custo médio por operação ao longo duma sequência de operações no caso pior.

  A partir de uma pilha autoajustável vazia, qualquer sequência de M operações *push* e *pop* tem um custo proporcional a M. Logo, cada operação tem custo amortizado constante

Robert Sedgewick, Algorithms, 4th Edition, Addison-Wesley, 2011

Universidade Federal do ABC

CMCC
Centro de Matemática, Computação e Cognição

Agenda

Exemplos de Filas

Nos sistemas operacionais são usadas e.g. para armazenar tarefas ainda não processadas ou concluídas

- os caracteres lidos do teclado
- documentos esperando para serem impressos
- processos esperando para usar um recurso
- instruções para serem executadas na CPU
- pacotes para serem encaminhados

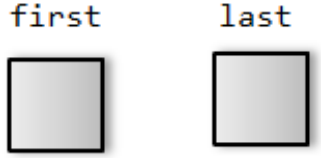
Fila: Implementação enlaçada

Mesma estrutura duma lista simplesmente ligada porém com ponteiros ao primeiro e último elementos, inicialmente nulos

```
struct Queue{
  LinkedNode *first, *last;
};
```

Fila: Implementação enlaçada

Mesma estrutura duma lista simplesmente ligada porém com ponteiros ao primeiro e último elementos, inicialmente nulos



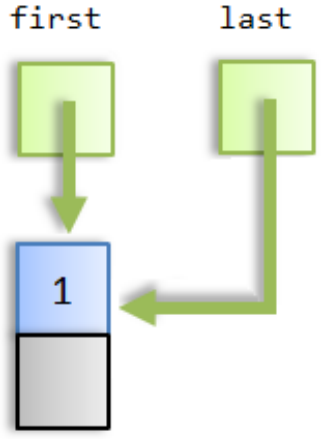

```
struct Queue{
  LinkedNode *first, *last;
};
```

- **Enfileirar** $\Leftrightarrow$ Inserir no final

  `enqueue(Queue * q, int v);`

Fila: Implementação enlaçada

enqueue(1)



Mesma estrutura duma lista simplesmente ligada porém com ponteiros ao primeiro e último elementos, inicialmente nulos

```
struct Queue{
  LinkedNode *first, *last;
};
```

- **Enfileirar** $\Leftrightarrow$ Inserir no final

  `enqueue(Queue * q, int v);`

Fila: Implementação enlaçada

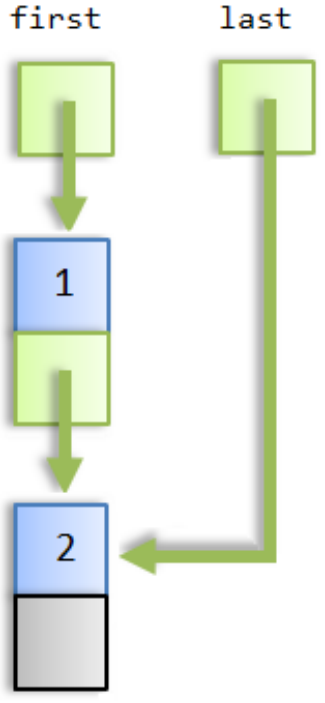
Mesma estrutura duma lista simplesmente ligada porém com ponteiros ao primeiro e último elementos, inicialmente nulos

```
struct Queue{
  LinkedNode *first, *last;
};
```

- **Enfileirar** ⇔ Inserir no final

  `enqueue(Queue * q, int v);`

enqueue(2)



Fila: Implementação enlaçada

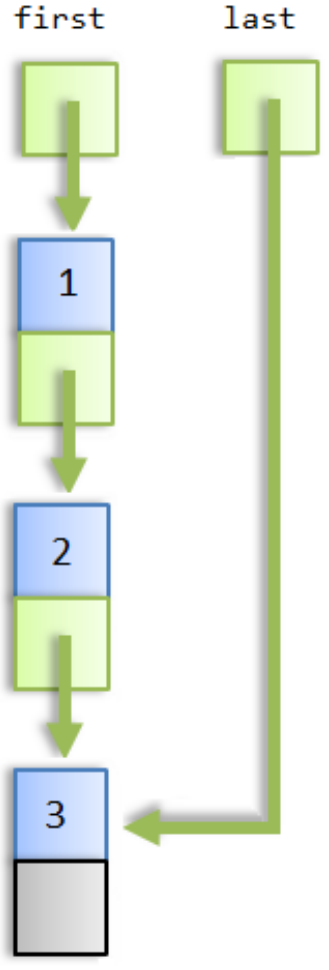
Mesma estrutura duma lista simplesmente ligada porém com ponteiros ao primeiro e último elementos, inicialmente nulos

```
struct Queue{
  LinkedNode *first, *last;
};
```

- **Enfileirar** ⇔ Inserir no final

  `enqueue(Queue * q, int v);`

enqueue(3)

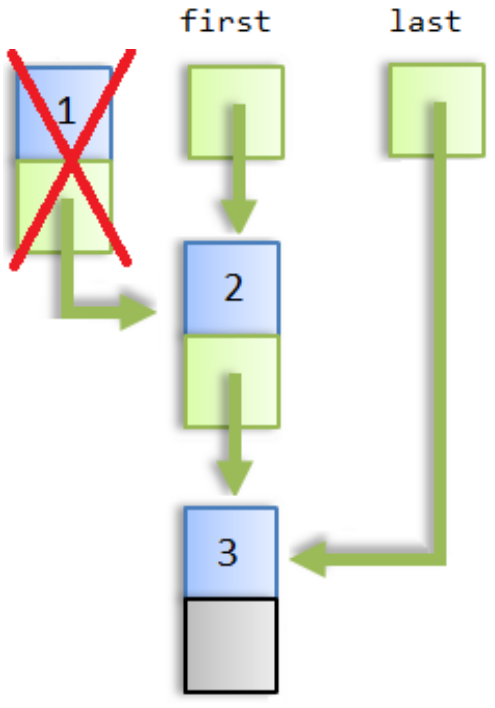


Fila: Implementação enlaçada

dequeue()

Mesma estrutura duma lista simplesmente ligada porém com ponteiros ao primeiro e último elementos, inicialmente nulos

```
struct Queue{
  LinkedNode *first, *last;
};
```



- **Enfileirar** ⇔ Inserir no final

  `enqueue(Queue * q, int v);`

- **Desenfileirar** ⇔ Inserir no inicio (*push*)

  `int enqueue(Queue * q);`

Fila: Implementação enlaçada

Mesma estrutura duma lista simplesmente ligada porém com ponteiros ao primeiro e último elementos, inicialmente nulos

```
struct Queue{
  LinkedNode *first, *last;
};
```

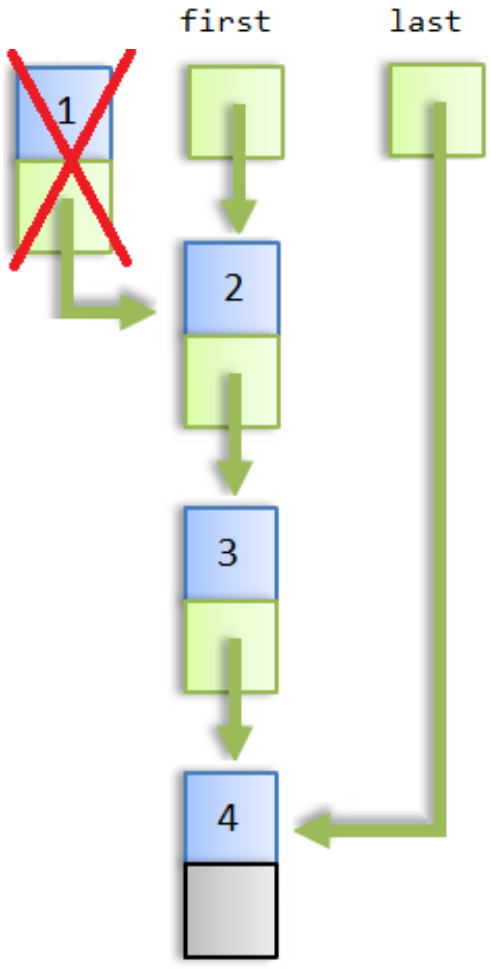
- **Enfileirar** ⇔ Inserir no final

  `enqueue(Queue * q, int v);`

- **Desenfileirar** ⇔ Inserir no inicio (*push*)

  `int enqueue(Queue * q);`

enqueue(4)

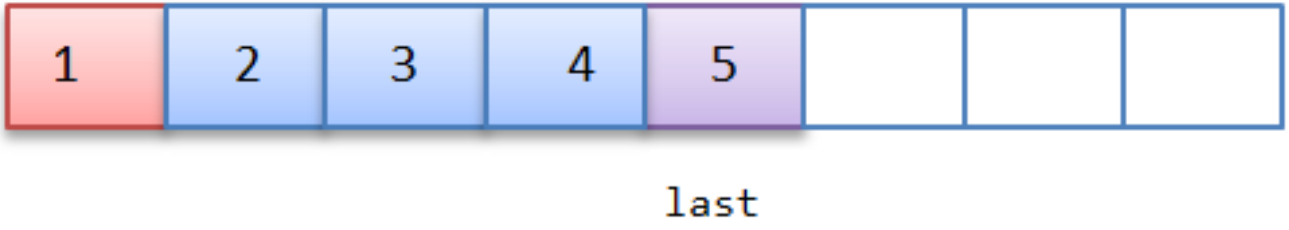


Importante: Atualizar ambos ponteiros e liberar o lixo em linguagens sem coleta automática de memória

Fila: Implementação usando vetores

- da mesma forma que as pilhas, as filas podem ser implementadas usando um único cursor dentro do vetor

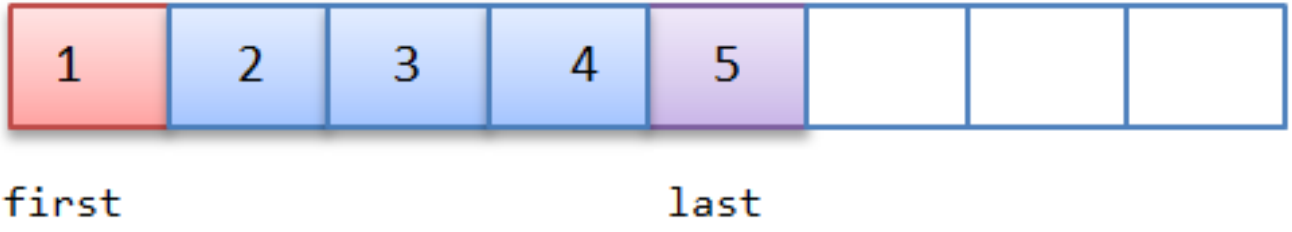
  Exemplo: após enfileirar os inteiros 1, 2, 3, 4, 5



Fila: Implementação usando vetores

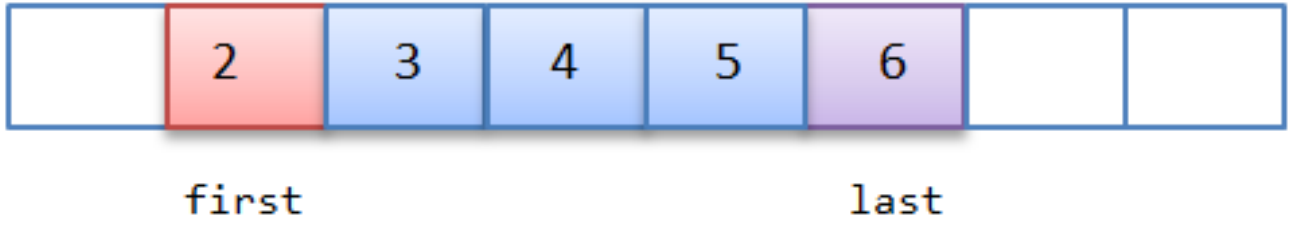
- da mesma forma que as pilhas, as filas podem ser implementadas usando um único cursor dentro do vetor

  Exemplo: após enfileirar os inteiros 1, 2, 3, 4, 5



- isso faz com que uma das operações tenha custo linear. É melhor usar dois cursores e deixar a fila se deslocar pelo vetor

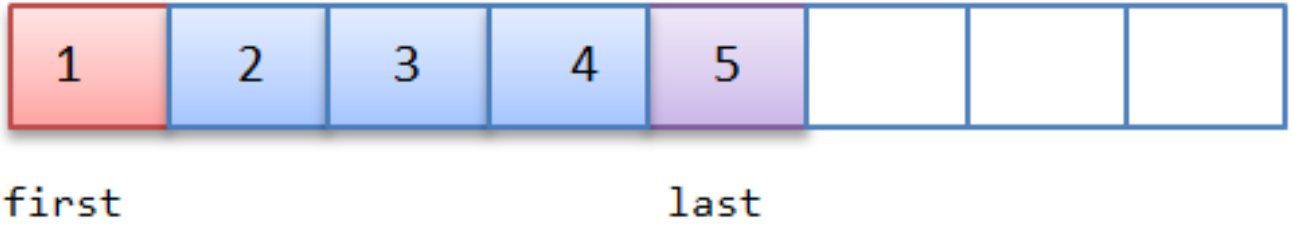
  Exemplo: após desenfileirar 1



Fila: Implementação usando vetores

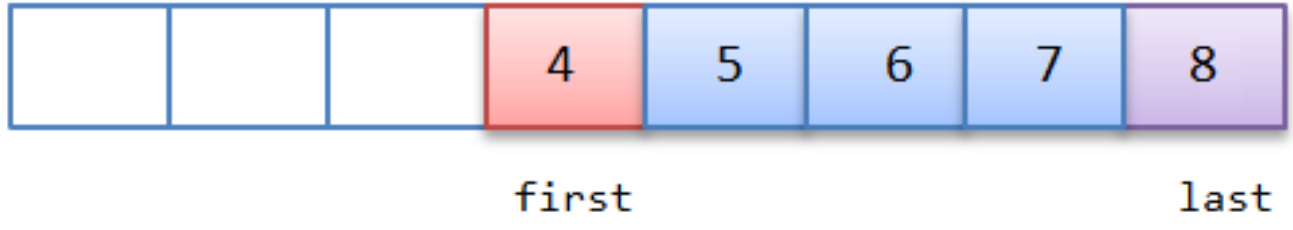
- da mesma forma que as pilhas, as filas podem ser implementadas usando um único cursor dentro do vetor

  Exemplo: após enfileirar os inteiros 1, 2, 3, 4, 5



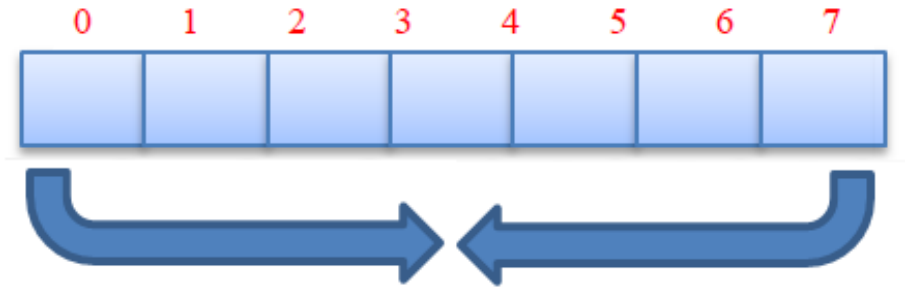
- isso faz com que uma das operações tenha custo linear. É melhor usar dois cursores e deixar a fila se deslocar pelo vetor

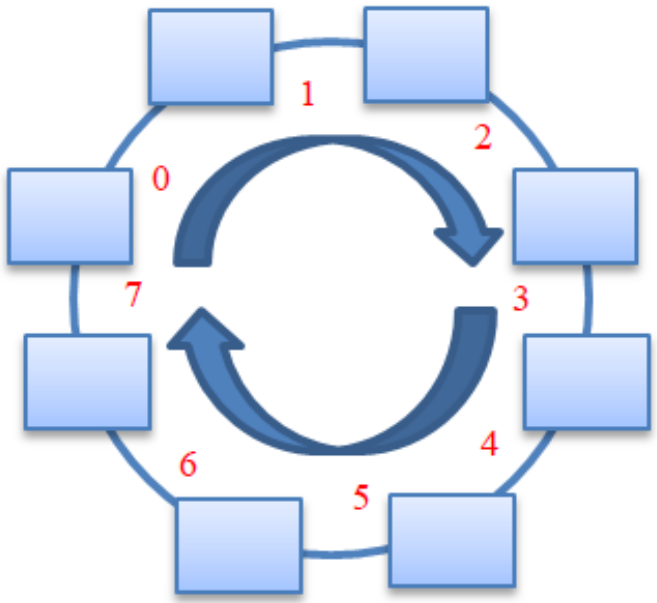
  Exemplo: após desenfileirar 1, 2, 3 e enfileirar 6, 7, 8



Fila: Implementação usando vetores

- Além disso, para garantir o custo constante, o deslocamento deve ser circular

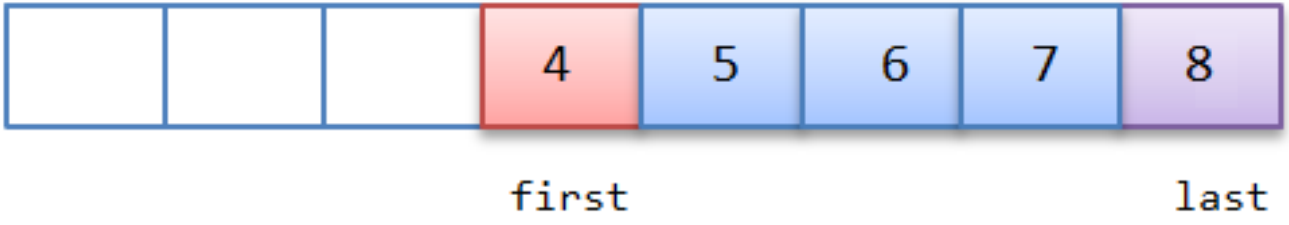




Fila: Implementação usando vetores

- Além disso, para garantir o custo constante, o deslocamento deve ser circular

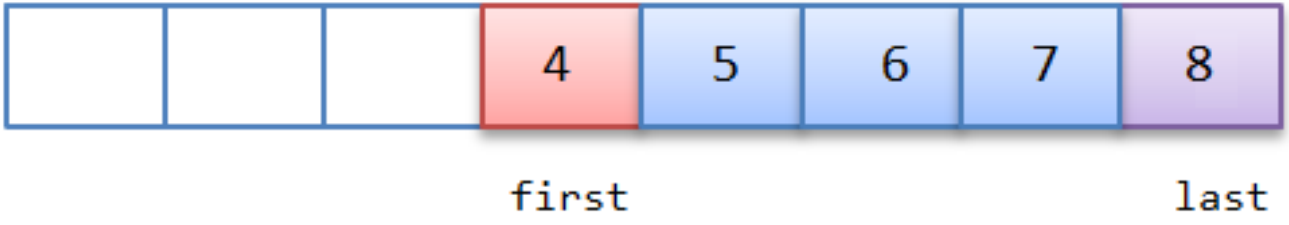
Exemplo: Antes



Fila: Implementação usando vetores

- Além disso, para garantir o custo constante, o deslocamento deve ser circular

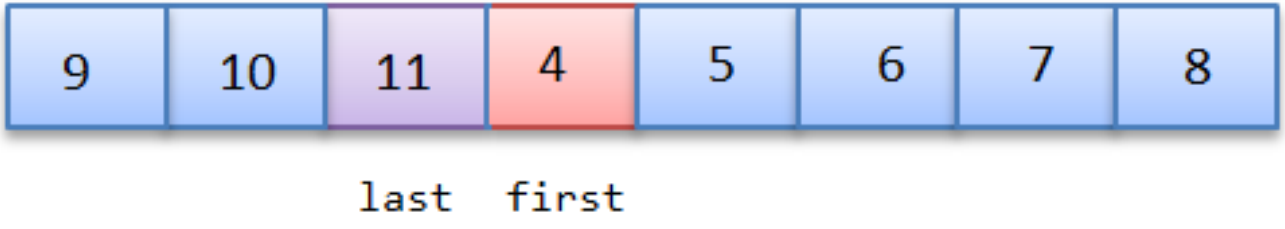
Exemplo: Antes



Após enfileirar 9, 10, 11 e desenfileirar 4, 5

Fila: Implementação usando vetores, alguns cuidados

- incrementar os cursores de forma apropriada i.e. se estiverem na última posição do vetor, a nova posição é 0;
- onde colocar o cursor `last`: na posição do último elemento ou na seguinte vazia? **Exemplo anterior**: Sem desenfileirar 4, 5



- verificar se a lista está vazia ou cheia: verificar os índices ou usar um contador auxiliar para o tamanho?
- Igualmente válidas as considerações sobre a implementação de listas e pilhas usando vetores

Resumo Pilhas e Filas

- As pilhas e filas são casos especiais de estruturas lineares amplamente usadas na computação
- Podem ser implementadas de forma sequencial ou encadeada
- O custo das operações é constante. Nas versões autoajustáveis o custo do redimensionamento é amortizado nas operações seguintes
- Outras variantes de filas: fila simétrica (Deque, Dequeue, Double-Ended Queues), inserções e remoções por ambos extremos; fila de prioridades (priority queue); insere em qualquer posição, remove o elemento de maior prioridade

Universidade Federal do ABC

CMCC Centro de Matemática, Computação e Cognição

Agenda

Referências Bibliográficas

- Introduction to Algorithms, 3rd Edition. Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest and Clifford Stein, 2009
- Algorithms, Robert Sedgewick and Kevin Wayne, 4th Edition, 2011, Slides http://algs4.cs.princeton.edu/lectures/
- The Art of Computer Programming 3rd Edition,Donald Knuth, Section 6.1: Sequential Searching, 1997
- Projeto de Algoritmos, 2da Edição, Nivio Ziviani, 2007
- Estruturas de Dados e seus Algoritmos, 3ra edição, Jayme L. Szwarcfiter and Lilian Markezon, 2010