LABORATÓRIO DE ALGORITMOS E ESTRUTURAS DE DADOS I

Prof. Caio César de Freitas Dantas

Tipos Abstratos de Dados - TAD

Variação de implementação

- Há diferentes implementações possíveis para o mesmo tipo de dado
- Todas definem o mesmo domínio e não mudam o significado das operações

Tipos Abstratos de Dados - TAD

Substituição de implementações

- Em programas reais, as implementações dos tipos de dados são modificadas constantemente para melhorar a:
    - Velocidade
    - Eficiência
    - Clareza

- Essas mudanças têm grande impacto nos programas usuários do tipo de dado. Por exemplo:
    - Re-implementação de código
    - Mais suscetível a erros
    - CUSTO MUITO ALTO!

Tipos Abstratos de Dados - TAD

Substituição de implementações

- Como podemos modificar as implementações dos tipos de dados com o menor impacto possível para os programas?

- Como podemos encapsular (esconder) de quem usa um determinado tipo de dado a forma concreta como este tipo foi implementado?

    - TIPOS ABSTRATOS DE DADOS (TAD)

Tipos Abstratos de Dados - TAD

- Um TAD especifica o tipo de dado (domínio e operações) sem referência a detalhes da implementação

- Minimiza código do programa que usa detalhes de implementação
    - Dando mais liberdade para mudar implementação com menor impacto nos programas
    - Minimiza custos

- Os programas que usam o TAD não “conhecem” as implementações dos TADs
    - Fazem uso do TAD através de operações

Tipos Abstratos de Dados - TAD

TAD Fracao (operações principais)

- Cria_fracao(N,D)

  Pega dois inteiros e retorna a fração N/D.
- Acessa_numerador(F)

  Pega a fração e retorna o numerador.
- Acessa_denominador(F)

  Pega a fração e retorna o denominador.
- Fracao Soma(fracao F1, fracao F2) {

```
int n1 = get_numerador(F1);
n2 = acessa_numerador(F2);
d1 = acessa_denominador(F1);
d2 = acessa_denominador(F2);
return cria_fracao( n1*d2+n2*d1 , d1*d2 ); }
```

Tipos Abstratos de Dados - TAD

Resumindo (TAD)

- Um TAD especifica tudo que se precisa saber para usar um determinado tipo de dado.

- Não faz referência à maneira com a qual o tipo de dado será (ou é) implementado.

Tipos Abstratos de Dados - TAD

Resumindo (TAD)

- Um TAD especifica tudo que se precisa saber para usar um determinado tipo de dado.

- Não faz referência à maneira com a qual o tipo de dado será (ou é) implementado.

Exemplos de TADs:

- Listas
- Pilhas
- Filas

TAD - Listas

- Em uma estrutura de dados tipo lista, para cada novo elemento inserido na lista, alocamos um espaço de memória para armazená-lo.
    - Dessa forma, o espaço total de memória gasto pela estrutura é proporcional ao número de elementos armazenados na lista.

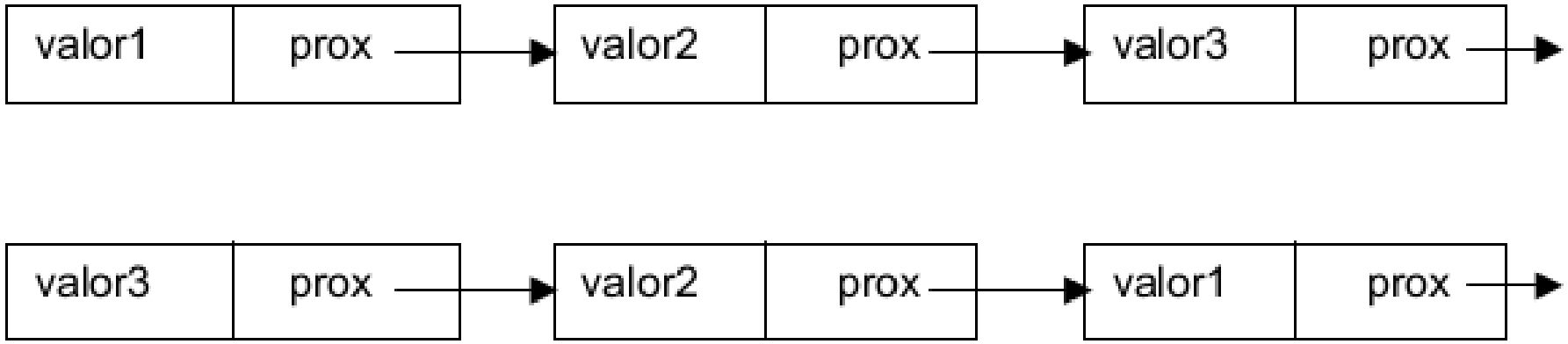
- No entanto, não podemos garantir que os elementos armazenados na lista ocuparão um espaço de memória contíguo, pois a alocação é feita dinamicamente;

TAD - Listas

- Para percorrer todos os elementos da lista, devemos explicitamente guardar o endereço de cada elemento da lista.
- Isto é feito armazenando-se, junto com a informação de cada elemento, um ponteiro com o endereço para o próximo elemento da lista.
- Por isso, os elementos da lista estão ligados uns aos outros, encadeados e por conta desta característica a lista é também conhecida como lista encadeada.

TAD - Listas

- Arranjo da memória de uma estrutura de dados lista.



TAD - Listas

- A estrutura de dados Lista consiste de uma sequência encadeada de elementos, em geral chamados de nós da lista.

- Um nó da lista é representado por uma estrutura que contém, conceitualmente, dois campos: a informação armazenada e o ponteiro para o próximo elemento da lista.

- A lista inteira é representada por um ponteiro para o primeiro elemento (ou nó). Do primeiro elemento, podemos alcançar o segundo.

- O último elemento da lista armazenada, não possui o ponteiro para o próximo elemento mas sim um ponteiro inválido, com valor NULL.

TAD - Listas

Características das Listas

- As listas são implementadas através de variáveis dinâmicas que são criadas em tempo de execução com alocação dinâmica de memória;
- O acesso aos nós componentes da lista é sempre sequencial (para se acessar o 4º elemento, é necessário passar antes pelo 3º, para se acessar o 3º elemento, é necessário passar antes pelo 2º e assim sucessivamente);
- As estruturas de representação de Listas Ligadas devem obrigatoriamente suportar conceitos como ponteiros e alocação dinâmica;

TAD - Listas

Características das Listas

- Os nós que compõem a lista devem ser tipos abstratos de dados do tipo struct contendo, pelo menos, dois tipos de informação:
  - Um campo ou mais campos de tipo simples ou struct para abrigar as informações de cada elemento armazenado na lista e
  - Um outro campo do tipo ponteiro para abrigar o endereço do próximo elemento, ou nó, da lista;

TAD - Listas

As vantagens e desvantagens do uso das Listas

- Maior complexidade inerente à manipulação os elementos da lista devido ao uso de ponteiros; (desvantagem)

- A flexibilidade de se trabalhar com listas de tamanhos indefinidos, que podem crescer e decrescer conforme a necessidade (vantagem);

- Maior facilidade para a realização de operações de inserção e remoção, que consistem basicamente no rearranjo de alguns ponteiros (vantagem).

TAD – Listas Duplamente encadeadas

- A forma de encadeamento simples permite o acesso aos elementos de uma lista apenas em uma direção.
- Desta forma, não temos como percorrer eficientemente os elementos em ordem inversa, isto é, do final para o início da lista.
- Sendo este um fator que limita o uso das listas encadeadas em determinados tipos de aplicação.

TAD – Listas Duplamente encadeadas

- O encadeamento simples também dificulta a retirada de um elemento da lista.

- Mesmo se tivermos o ponteiro do elemento que desejamos retirar, temos que percorrer a lista, elemento por elemento, para encontrarmos o elemento anterior, pois, dado um determinado elemento, não temos como acessar diretamente seu elemento anterior.

TAD – Listas Duplamente encadeadas

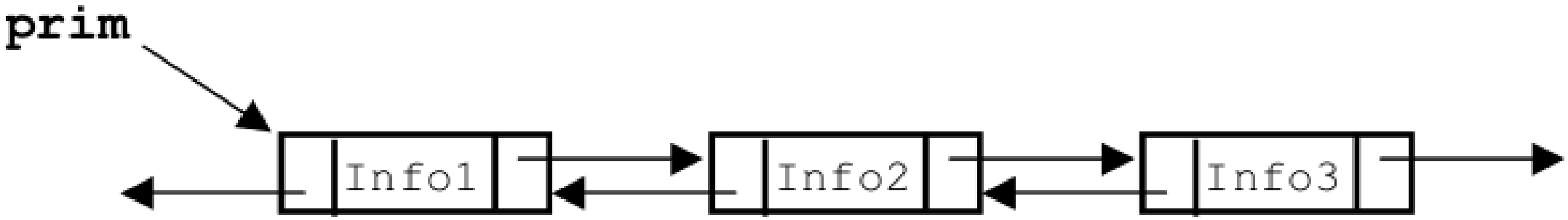
- Cada elemento tem um ponteiro para o próximo elemento e um ponteiro para o elemento anterior.

- Desta forma, dado um elemento, podemos acessar ambos os elementos adjacentes: o próximo e o anterior.

- Se tivermos um ponteiro para o último elemento da lista, podemos percorrer a lista em ordem inversa, bastando acessar continuamente o elemento anterior, até alcançar o primeiro elemento da lista, que não tem elemento anterior (o ponteiro do elemento anterior vale NULL).

TAD – Listas Duplamente encadeadas

Cada elemento possui dois ponteiros, um que aponta para o próximo elemento da lista e outro ponteiro que aponta para o elemento anterior da lista.



TAD – Listas Circulares

- Numa lista circular, o último elemento tem como próximo o primeiro elemento da lista, formando um ciclo.

- A rigor, neste caso, não faz sentido falarmos em primeiro ou último elemento. A lista pode ser representada por um ponteiro para um elemento inicial qualquer da lista.

- Para percorrer os elementos de uma lista circular, visitamos todos os elementos a partir do ponteiro do elemento inicial até alcançarmos novamente esse mesmo elemento.

- Uma lista circular também pode ser construída com encadeamento duplo

TAD – Filas

- Na estrutura de fila, os acessos aos elementos também seguem uma regra.
- Na fila “o primeiro que entra é o primeiro que sai” (a sigla FIFO – first in, first out – é usada para descrever essa estratégia).
- A ideia fundamental da fila é que só podemos inserir um novo elemento no final da fila e só podemos retirar o elemento do início.
- A estrutura de fila é uma analogia natural com o conceito de fila que usamos no nosso dia a dia: quem primeiro entra numa fila é o primeiro a ser atendido (a sair da fila).

TAD – Filas

- Se uma impressora é compartilhada por várias máquinas, deve-se adotar uma estratégia para determinar que documento será impresso primeiro.

- A estratégia mais simples é tratar todas as requisições com a mesma prioridade e imprimir os documentos na ordem em que foram submetidos – o primeiro submetido é o primeiro a ser impresso.

- Para implementar uma fila, devemos ser capazes de inserir novos elementos em uma extremidade, o fim, e retirar elementos da outra extremidade, o início.

TAD – Filas

As operações de uma Fila

Independente da estratégia de implementação, uma estrutura de fila deve ser composta pelas seguintes operações:

- Criar uma estrutura de fila;
- Inserir um elemento no fim;
- Retirar o elemento do início;
- Verificar se a fila está vazia;
- Liberar a fila.

TAD – Filas

As operações de uma Fila

O trecho de código abaixo representa a estrutura fila, o ponteiro para um elemento da estrutura e as operações possíveis com uma fila:

```
typedef struct fila Fila;   // Estrutura para a fila
Fila* cria (void);              // Ponteiro para a fila
void insere (Fila* f, float v);  // Funcao para incluir um elemento no final da fila
float retira (Fila* f);          // Funcao para remover um elemento do inicio da fila
int vazia (Fila* f);             // Funcao para verificar se a fila esta vazia
void libera (Fila* f);           // Funcao para liberar a fila da memoria
```

TAD – Filas

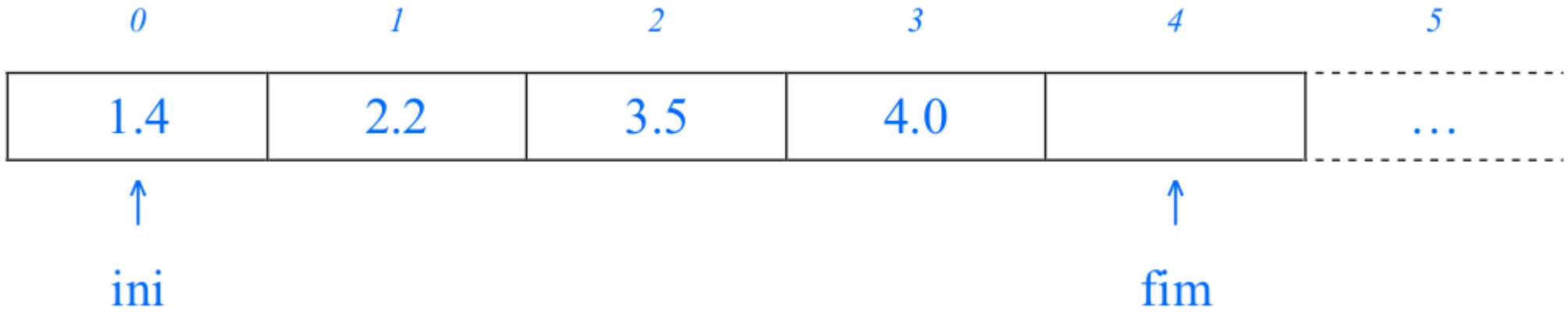
- Uma fila pode ser implementada com um vetor. Para isso, deve-se fixar o número máximo N de elementos na fila.

- Podemos observar que o processo de inserção e remoção em extremidades opostas fará com que a fila “ande” no vetor.

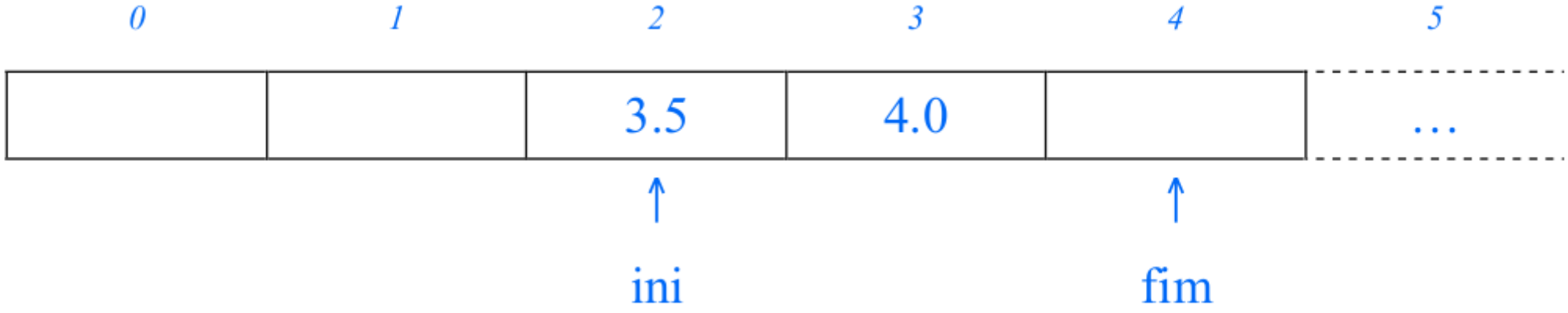
- Por exemplo, se inserirmos os elementos 1.4, 2.2, 3.5, 4.0 e depois retirarmos dois elementos, a fila não estará mais nas posições iniciais do vetor.

TAD – Filas

- A configuração da fila após a inserção dos primeiros quatro elementos.



- A configuração da fila após a remoção de dois elementos.



TAD – Pilhas

- Uma pilha é um conjunto ordenado de itens, no qual novos itens podem ser inseridos e a partir do qual podem ser eliminados itens de uma extremidade, chamada topo da pilha.

- Onde todas as inserções e eliminações são feitas em apenas uma das extremidades, chamada topo.

- A estrutura de dados em pilha tem como característica que a última informação a entrar é a primeira a sair (LIFO - last in first out).

TAD – Pilhas

A estrutura de dados em pilha tem as seguintes operações implementadas pelas funções abaixo:

- push - coloca uma informação na pilha (empilha).
- pop - retira uma informação da pilha (desempilha).
- size - retorna o tamanho da pilha.
- stackpop - retorna o elemento superior da pilha sem removê-lo (equivalente às operações de pop e um push).
- empty - verifica se a pilha está vazia.

TAD – Pilhas

A estrutura de dados em pilha tem as seguintes operações implementadas pelas funções abaixo:

- push - coloca uma informação na pilha (empilha).
- pop - retira uma informação da pilha (desempilha).
- size - retorna o tamanho da pilha.
- stackpop - retorna o elemento superior da pilha sem removê-lo (equivalente às operações de pop e um push).
- empty - verifica se a pilha está vazia.

FIM!