Tipos abstratos de dados

Em computação, um tipo abstrato de dado (TAD) é o modelo matemático de um determinado tipo de dado que define seus valores possíveis e as operações sobre ele.

É importante observar que a definição de um TAD não determina a forma de implementá-lo. O TAD separa a **especificação** do dado (que define como ele funciona) de sua **implementação** (como ele está codificado). Isso possibilita:

- usar um TAD sem precisar conhecer os detalhes internos de sua implementação;
- alterar a implementação do TAD sem afetar os programas que a usam.

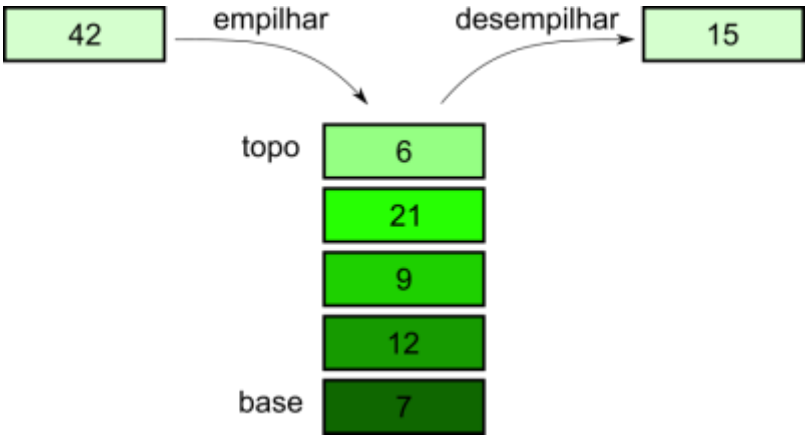
TADs são geralmente usados para representar estruturas de dados de uso frequente em computação, como filas, pilhas e conjuntos, mas podem ser definidos para qualquer tipo de dado frequentemente manipulado em uma aplicação.

Pilha

Uma pilha (ou *stack*) armazena uma sequência de valores “empilhados” como em uma pilha de pratos ou de caixas. Uma pilha tem uma **base** e um **topo**:

- A **base** contém o **primeiro** elemento que entrou na pilha (e que será o último a sair dela);
- O **topo** contém o **último** elemento que entrou na pilha (e que será o primeiro a sair dela).

Da mesma forma que a fila, podemos ter pilhas de inteiros, de racionais, de strings, etc. A principal característica da pilha é que os elementos entram e saem sempre **pelo topo** da pilha. Por isso, pilhas são também conhecidas como LIFO, do inglês *Last In, First Out* (o último a entrar é o primeiro a sair).



Algumas operações tipicamente definidas sobre uma pilha são:

- `cria (pilha, capacidade)` : cria uma nova pilha, informando o número máximo de elementos que ela pode comportar
- `destroi (pilha)` : esvazia/destrói a pilha
- `insere (pilha, elemento)` : insere um novo elemento **no topo** da pilha
- `elemento = retira (pilha)` : retira o elemento do topo da pilha
- `elemento = topo (pilha)` : devolve o elemento do topo da pilha sem removê-lo
- `tamanho (pilha)` : informa o número atual de elementos na pilha

- ...

Uma interface possível para o TAD “pilha de números inteiros” em C seria:

pilha.h

```
#ifndef PILHA
#define PILHA

struct pilha_t
{
  ...  // conteúdo depende da implementação em pilha.c
} ;

// Cria uma pilha vazia com a capacidade informada e a retorna;
// Retorna NULL em caso de erro
struct pilha_t *pilha_cria (int capacidade);

// Remove todos os elementos da pilha, libera a memória e retorna NULL
struct pilha_t *pilha_destroi (struct pilha_t *pilha);

// Insere o elemento no topo da pilha (politica LIFO);
// Retorna o número de elementos na pilha após a operação
// ou -1 em caso de erro
int pilha_insere (struct pilha_t *pilha, int elem);

// Retira o elemento do topo da pilha (politica LIFO) e o devolve;
// Retorna o número de elementos na pilha após a operação
// ou -1 em caso de erro
int pilha_retira (struct pilha_t *pilha, int *elem);

// devolve o elemento no topo da pilha, sem removê-lo da pilha
// Retorna o número de elementos na pilha ou -1 em caso de erro
int pilha_topo (struct pilha_t *pilha, int *elem);

// Retorna o tamanho da pilha (número de elementos na pilha)
// ou -1 em caso de erro
int pilha_tamanho (struct pilha_t *pilha);

// Retorna a capacidade da pilha (número de elementos que ela aceita)
// ou -1 em caso de erro
int pilha_capacidade (struct pilha_t *pilha);

// Imprime o conteúdo da pilha, do topo à base
void pilha_imprime (struct pilha_t *pilha);

#endif
```

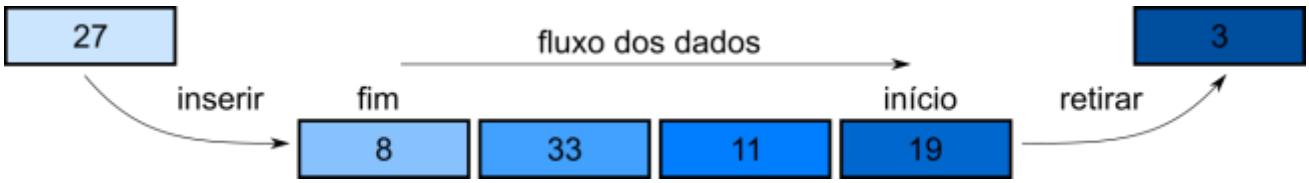
Pilhas são usualmente implementadas com vetores ou listas encadeadas simples.

Fila

Uma fila (ou *queue*) armazena uma sequência de valores. Por exemplo:

- uma fila de inteiros: 11, 7, 4, 5, 9, -2, 16, 0, 4
- uma fila de racionais: 1/4, -3/2, 5/8, 0, 3/7
- uma fila de strings: “uva”, “pera”, “abacaxi”, “melão”

A principal característica da fila é que os elementos entram **no fim** da fila e saem **do início** dela, da mesma forma que em filas do mundo real. Por isso, filas são também conhecidas como FIFO, do inglês *First In, First Out* (o primeiro a entrar é o primeiro a sair):



Filas são entidades muito usadas em computação. Por exemplo, filas são usadas para:

- Organizar pacotes de rede
- Gerenciar processos que querem usar a CPU
- Sequenciar pedidos de acesso a dados ou serviços
- ...

Algumas operações tipicamente definidas sobre uma fila são:

- `cria (fila, capacidade)` : cria uma nova fila, informando o número máximo de elementos que ela pode comportar; uma fila pode ser considerada infinita, se não tiver capacidade máxima definida.
- `destroi (fila)` : esvazia/destrói a fila
- `insere (fila, elemento)` : insere um novo elemento **no fim** da fila
- `elemento = retira (fila)` : retira **o primeiro** elemento da fila
- `elemento = primeiro (fila)` : informa qual é o primeiro elemento, mas não o retira da fila
- `tamanho (fila)` : informa o número atual de elementos na fila
- ...

Uma interface possível para o TAD “fila de números inteiros” em C seria:

fila.h

```
#ifndef FILA
#define FILA

struct fila_t {
  ...  // conteúdo depende da implementação em fila.c
} ;

// Cria uma fila vazia com a capacidade informada e a retorna;
// Retorna NULL em caso de erro
struct fila_t *fila_cria (int capacidade);

// Remove todos os elementos da fila, libera memória e retorna NULL
struct fila_t *fila_destroi (struct fila_t *f);

// Insere o elemento no final da fila (politica FIFO);
// Retorna o número de elementos na fila após a operação
// ou -1 em caso de erro
int fila_insere (struct fila_t *f, int elem);
```

```
// Retira o elemento do inicio da fila (politica FIFO) e o devolve;
// Retorna o número de elementos na fila após a operação
// ou -1 em caso de erro
int fila_retira (struct fila_t *f, int *elem);

// Devolve o primeiro da fila, sem removê-lo
// Retorna o número de elementos na fila ou -1 em caso de erro
int fila_primeiro (struct fila_t *f, int *elem);

// Retorna o tamanho da fila (número de elementos presentes)
int fila_tamanho (struct fila_t *f);

// Retorna a capacidade da fila (número máximo de elementos)
int fila_capacidade (struct fila_t *f);

// Imprime o conteúdo da fila, do inicio ao fim
void fila_imprime (struct fila_t *f);

#endif
```

Filas podem ser implementadas de diversas formas, as mais usuais são:

- vetor linear simples
- vetor linear otimizado (só move elementos se o fim do vetor for atingido)
- vetor circular
- lista encadeada simples
- lista encadeada dupla
- lista encadeada circular simples
- lista encadeada circular dupla

Observe que, embora o conteúdo do *struct* `fila_t` dependa da forma de implementação da fila, os protótipos das funções não dependem desse conteúdo. Isso torna o uso da fila independente de sua implementação.

Deque

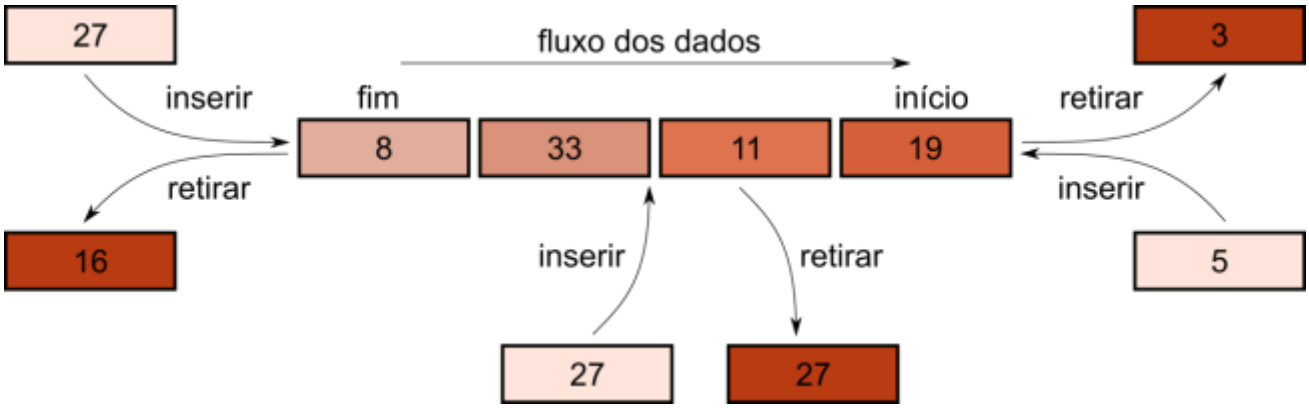
Um deque (contração do inglês *double-ended queue*) é similar a uma fila, mas permite operações (inserir, retirar, consultar) em ambas as pontas (início e fim). O deque pode ser visto como uma generalização de fila e pilha, porque pode funcionar como qualquer uma dessas estruturas.

Algumas operações tipicamente definidas sobre um deque são:

- `cria (deque, capacidade)` : cria um novo deque, informando o número máximo de elementos que ele pode comportar
- `destroi (deque)` : esvazia/destrói o deque
- `insere_inicio (deque, elemento)` : insere um novo elemento **no início**
- `insere_final (deque, elemento)` : insere um novo elemento **no fim**
- `elemento = retira_inicio (deque)` : retira **o primeiro** elemento
- `elemento = retira_final (deque)` : retira **o último** elemento
- `elemento = primeiro (deque)` : informa qual é o primeiro elemento, mas não o retira
- `elemento = ultimo (deque)` : informa qual é o último elemento, mas não o retira
- `tamanho (deque)` : informa o número atual de elementos no deque
- ...

Lista

Uma lista é similar a uma fila (FIFO), mas permite operações (inserir, retirar, consultar) sobre qualquer um de seus elementos. A lista pode ser vista como uma generalização dos TADs fila, pilha e deque, porque pode funcionar como qualquer uma dessas estruturas.



Algumas operações tipicamente definidas sobre uma lista são:

- `cria (lista, capacidade)` : cria uma nova lista, informando o número máximo de elementos que ela pode comportar
- `destroi (lista)` : esvazia/destrói a lista
- `insere_inicio (lista, elemento)` : insere um novo elemento **no início**
- `insere_fim (lista, elemento)` : insere um novo elemento **no fim**
- `insere_pos (lista, elemento, pos)` : insere um novo elemento em uma posição qualquer
- `elemento = retira_inicio (lista)` : retira o elemento do início

- `elemento = retira_fim (lista)` : retira o elemento do fim
- `elemento = retira_pos (lista)` : retira o elemento de uma posição qualquer
- `pertence (lista, elemento)` : informa se o elemento aparece na lista
- `posição (lista, elemento)` : informa a posição do elemento na lista
- `tamanho (lista)` : informa o número atual de elementos na lista
- ...

Conjunto

Um TAD conjunto (ou *set*) é uma coleção de itens similar aos conjuntos matemáticos.

Algumas operações tipicamente definidas sobre um conjunto são:

- `cria (conjunto, capacidade)` : cria um novo conjunto, informando o número máximo de elementos que ele pode comportar.
- `destroi (conjunto)` : esvazia/destrói o conjunto
- `insere (conjunto, elemento)` : insere o elemento no conjunto
- `retira (conjunto, elemento)` : retira o elemento do conjunto
- `pertence (conjunto, elemento)` : informa se o elemento pertence ao conjunto
- `uniao (c1, c2)` : calcula a união de dois conjuntos
- `intersecao (c1, c2)` : calcula a interseção de dois conjuntos
- `diferenca (c1, c2)` : calcula a diferença de dois conjuntos
- `igual (c1, c2)` : informa se os conjuntos c1 e c2 são iguais
- `contem (c1, c2)` : informa se c1 contém c2 (c2 é subconjunto de c1)
- `tamanho (conjunto)` : informa o número atual de elementos no conjunto (cardinalidade)
- ...

Fila de prioridade

Um TAD fila de prioridade é uma fila na qual cada elemento é associado a uma chave (prioridade) e a fila é mantida ordenada segundo os valores das chaves. Por exemplo, considerando uma fila de prioridades `fp` inicialmente vazia:

operação (valor, chave)	**conteúdo da fila**
insere (5, 3)	[**(5, 3)**]
insere (0, 6)	[(5, 3) **(0, 6)**]
insere (1, 1)	[**(1, 1)** (5, 3) (0, 6)]
insere (-2, 2)	[(1, 1) **(-2, 2)** (5, 3) (0, 6)]
insere (8, 2)[1)]	[(1, 1) (-2, 2) **(8, 2)** (5, 3) (0, 6)]
insere (9, 4)	[(1, 1) (-2, 2) (8, 2) (5, 3) **(9, 4)** (0, 6)]
retira min	[(-2, 2) (8, 2) (5, 3) (9, 4) (0, 6)]
retira min	[(8, 2) (5, 3) (9, 4) (0, 6)]
retira max	[(8, 2) (5, 3) (9, 4)]

Algumas operações tipicamente definidas sobre uma fila de prioridades são:

- `cria (fila, capacidade)` : cria uma nova fila, informando o número máximo de elementos que ela pode comportar; uma fila pode ser considerada infinita, se não tiver capacidade máxima definida.
- `destroi (fila)` : esvazia/destrói a fila
- `insere (fila, elemento, chave)` : insere um novo elemento na fila, posicionado de acordo com o valor da chave informada (prioridade)
- `elemento = retira_min (fila)` : retira o elemento da fila que tem a **menor** chave/prioridade

- `elemento = retira_max (fila)` : retira o elemento da fila que tem a **maior** chave/prioridade
- `valor_min (fila)` : informa o valor do elemento que tem a menor prioridade, sem alterar a fila
- `valor_max (fila)` : informa o valor do elemento que tem a maior prioridade, sem alterar a fila
- `chave_min (fila)` : informa o valor da menor prioridade, sem alterar a fila
- `chave_max (fila)` : informa o valor da maior prioridade, sem alterar a fila
- `tamanho (fila)` : informa o número atual de elementos na fila
- ...

Filas de prioridades têm muitos usos em computação, como na implementação de simuladores, no processamento de pacotes de rede, em algoritmos de compressão de dados e no cálculo de rotas em mapas.

Exercícios

1. Implemente o TAD “fila de inteiros” usando um vetor alocado dinamicamente para armazenar os inteiros.
2. Idem, usando uma lista de *struct*s alocadas dinamicamente e ligadas por ponteiros (lista encadeada).
3. Compare as duas implementações sob os seguintes aspectos:
    1. simplicidade da implementação;
    2. rapidez das operações para inserir e remover elementos;
    3. uso total de memória;
    4. flexibilidade no uso da memória.
4. Repita os itens acima para os TADs pilha, lista e conjunto.

[1)]
Se dois valores têm a mesma chave, devem ser inseridos e retirados em ordem FIFO.

From:
https://wiki.inf.ufpr.br/maziero/ - **Prof. Carlos Maziero**

Permanent link:
https://wiki.inf.ufpr.br/maziero/doku.php?id=prog1:tipos_abstratos_de_dados

Last update: **2024/10/24 18:27**