INSTITUTO FEDERAL
Sudeste de Minas Gerais
Campus Manhuaçu

Estruturas de Dados I
Pilhas

Prof. Leonardo C. R. Soares - *leonardo.soares@ifsudestemg.edu.br*
Instituto Federal do Sudeste de Minas Gerais

23 de setembro de 2023

Pilhas

Descrição

- Uma pilha é um contêiner de objetos que são inseridos e retirados de acordo com o princípio de que o **último que entra é o primeiro que sai** (do inglês, Last In First Out, LIFO).

Pilhas

Descrição

- Uma pilha é um contêiner de objetos que são inseridos e retirados de acordo com o princípio de que o **último que entra é o primeiro que sai** (do inglês, Last In First Out, LIFO).
- O nome pilha deriva-se da metáfora de uma pilha de pratos em uma cantina.

Pilhas

Descrição

- Uma pilha é um contêiner de objetos que são inseridos e retirados de acordo com o princípio de que o **último que entra é o primeiro que sai** (do inglês, Last In First Out, LIFO).
- O nome pilha deriva-se da metáfora de uma pilha de pratos em uma cantina.
- As pilhas são estruturas de dados fundamentais sendo utilizadas em muitas aplicações, por exemplo:

Pilhas - Exemplos

- Navegadores *web* armazenam os endereços mais recentemente visitados em uma pilha. Cada vez que o navegador visita um novo site, o endereço do site é armazenado na pilha de endereços (*push*). Usando a operação de retorno, *back*, o navegador permite que o usuário retorne ao último site visitado, retirando o endereço do topo da pilha (*pop*).

Pilhas - Exemplos

- Navegadores *web* armazenam os endereços mais recentemente visitados em uma pilha. Cada vez que o navegador visita um novo site, o endereço do site é armazenado na pilha de endereços (*push*). Usando a operação de retorno, *back*, o navegador permite que o usuário retorne ao último site visitado, retirando o endereço do topo da pilha (*pop*).
- Editores de texto geralmente oferecem um mecanismo de reversão de operações (*undo*) que cancela as operações recentes e reverte um documento ao estado anterior à operação. O mecanismo de reversão é implementado mantendo-se as alterações no texto em uma pilha.

Operações

O tipo abstrato de dados (TAD) pilha deve, obrigatoriamente, suportar os métodos:

- push(o): Insere o objeto o no topo da pilha.

Operações

O tipo abstrato de dados (TAD) pilha deve, obrigatoriamente, suportar os métodos:

- push(o): Insere o objeto o no topo da pilha.
- pop(): Retira o objeto no topo da pilha e o retorna; se a pilha estiver vazia, ocorre um erro.

Operações

O tipo abstrato de dados (TAD) pilha deve, obrigatoriamente, suportar os métodos:

- push(o): Insere o objeto o no topo da pilha.
- pop(): Retira o objeto no topo da pilha e o retorna; se a pilha estiver vazia, ocorre um erro.

Adicionalmente, podemos definir os seguintes métodos auxiliares:

- tamanho(): Retorna o número de objetos na pilha.

Operações

O tipo abstrato de dados (TAD) pilha deve, obrigatoriamente, suportar os métodos:

- push(o): Insere o objeto o no topo da pilha.
- pop(): Retira o objeto no topo da pilha e o retorna; se a pilha estiver vazia, ocorre um erro.

Adicionalmente, podemos definir os seguintes métodos auxiliares:

- tamanho(): Retorna o número de objetos na pilha.
- vazia(): Retorna um *boolean* indicando se a pilha está vazia;

Operações

O tipo abstrato de dados (TAD) pilha deve, obrigatoriamente, suportar os métodos:

- push(o): Insere o objeto o no topo da pilha.
- pop(): Retira o objeto no topo da pilha e o retorna; se a pilha estiver vazia, ocorre um erro.

Adicionalmente, podemos definir os seguintes métodos auxiliares:

- tamanho(): Retorna o número de objetos na pilha.
- vazia(): Retorna um *boolean* indicando se a pilha está vazia;
- top(): Retorna o elemento no topo da pilha sem removê-lo.

Formas de implementação

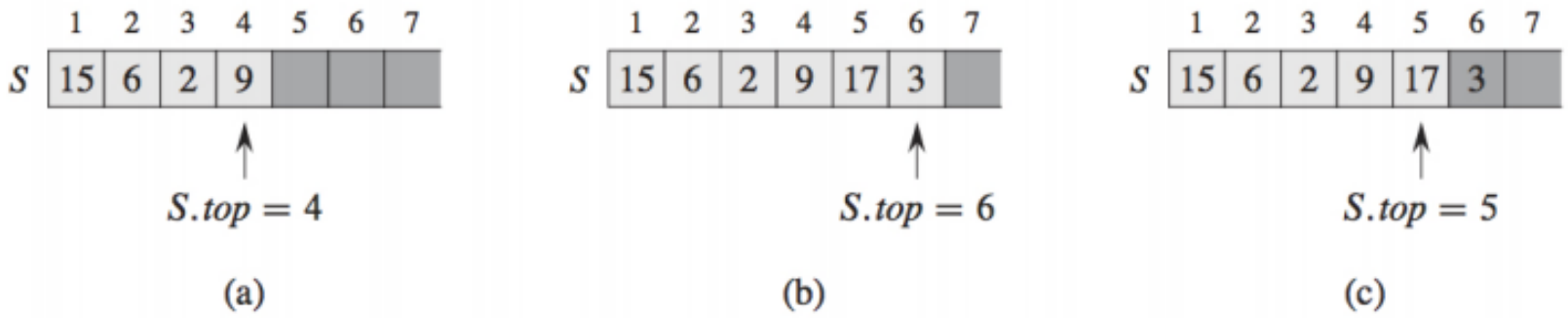
Existem várias opções de estruturas de dados que podem ser usadas para representar pilhas. As duas representações mais utilizadas são:

- Por meio de arranjos.
- Por meio de referências.

Independente da forma de implementação, uma pilha é uma lista com restrições quanto às formas de inserção e remoção, o que permite a reusabilidade de código.

Implementação por arranjos

Os itens da pilha são armazenados em posições contíguas de memória.
Como as inserções e as remoções ocorrem no topo da pilha, um campo chamado **topo** é utilizado para controlar a posição do item no topo da pilha.



(a) Uma pilha com quatro elementos. (b) Após empilhar (push) dois elementos. (c) Após desempilhar (pop) um elemento.

Implementação por referência

- Cada célula de uma pilha contém um item da pilha e um apontador para outra célula.

Implementação por referência

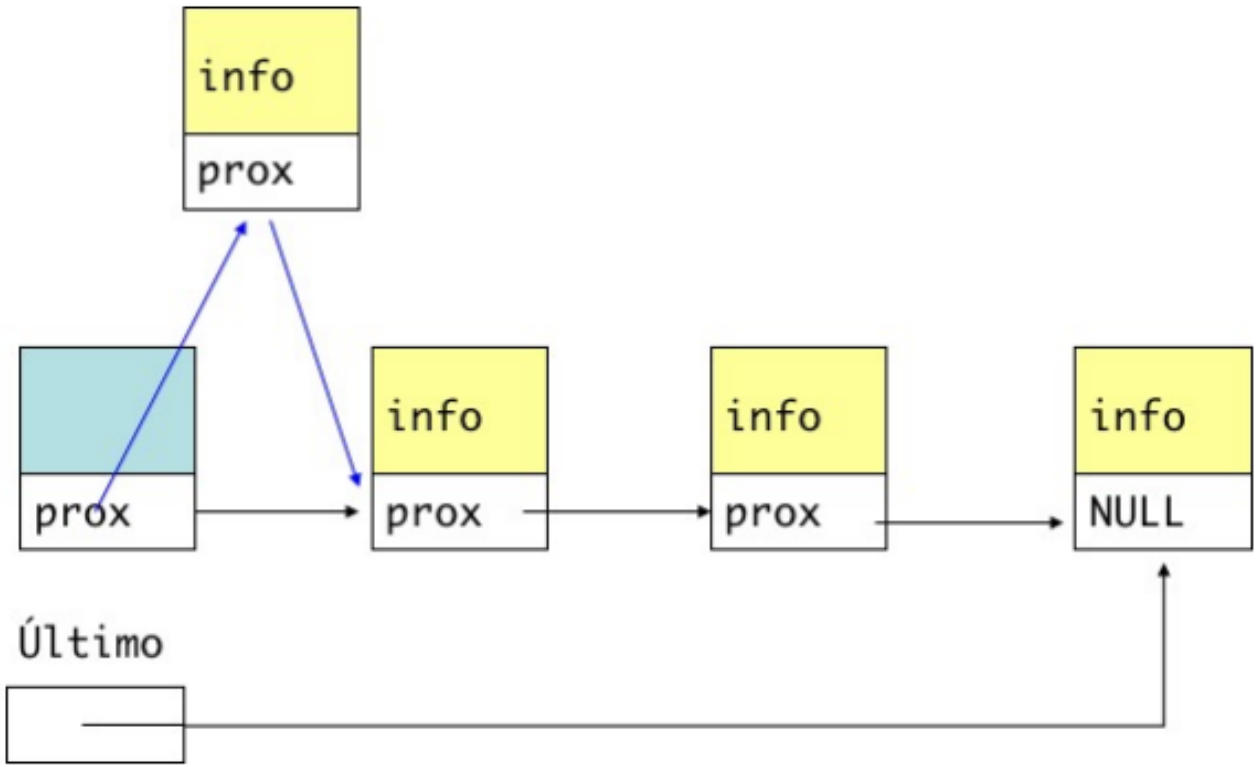
- Cada célula de uma pilha contém um item da pilha e um apontador para outra célula.
- A estrutura contém um apontador para o topo da pilha (célula cabeça).

Implementação por referência

- Cada célula de uma pilha contém um item da pilha e um apontador para outra célula.
- A estrutura contém um apontador para o topo da pilha (célula cabeça).
- Criar um campo tamanho evita a contagem do número de itens na função tamanho.

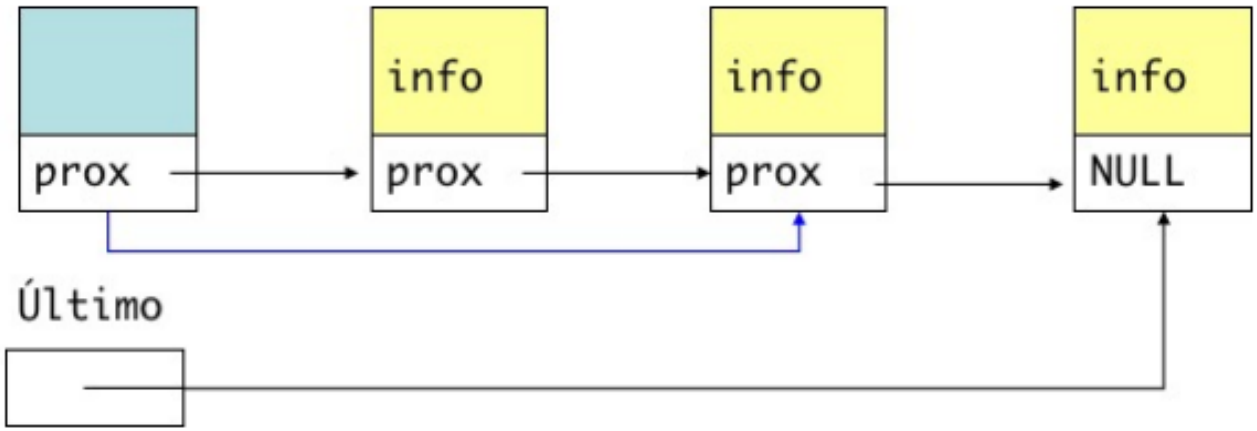
Implementação por referência - Inserção

De acordo com a política **LIFO**, há apenas uma opção de posição onde podemos inserir elementos: o topo da pilha (ou seja, primeira posição).



Implementação por referência - Remoção

De acordo com a política **LIFO**, há apenas uma opção de posição onde podemos remover elementos: o topo da pilha (ou seja, primeira posição).



Complexidade

A complexidade de todas as operações é mantida da implementação de Lista:

- Empilhar: $\theta(1)$
- Desempilhar: $\theta(1)$

Perguntas?

```java
public class Processo {
    int codigo;
    String responsavel;
    String cliente;
    public Processo(int codigo, String responsavel, String cliente) {
        this.codigo = codigo;
        this.responsavel = responsavel;
        this.cliente = cliente;
    }
    public Processo() {
    }
}
```

```java
public class Pilha {
    static final int MAX_TAM = 100;
    Processo[] pilha = new Processo[MAX_TAM];
    int topo = -1;
    public boolean isVazia(){
        return topo == -1;
    }
    public int getTamanho(){
        return topo+1;
    }
    public void push(Processo p) throws Exception{
        if (topo==MAX_TAM-1)
            throw new Exception ("Não há espaço disponível");
        pilha[++topo] = p;
    }
    public Processo pop() throws Exception{
        if (isVazia())
            throw new Exception ("Lista vazia");
        return pilha[topo--];
        // Atenção ao operador de pós-decremento
    }
}
```

```java
public class Main {

    public static void main(String[] args) throws Exception{
        Pilha p = new Pilha();
        Processo proc = new Processo();
        p.push(new Processo(1, "Rosimeire", "Acme"));
        p.push(new Processo(2, "Afonso", "Sansung"));
        p.push(new Processo(3, "Rosimeire", "Lenovo"));
        p.push(new Processo(4, "Ana", "Lenovo"));
        p.push(new Processo(5, "Afonso", "Acme"));
        p.push(new Processo(6, "Rosimeire", "Lenovo"));

        System.out.println("Lista de processos a serem executados:");
        while (!p.isVazia()){
            proc = p.pop();
            System.out.printf("Responsável: %s\t\t Código: %d\tCliente: %s\n",
                    proc.responsavel, proc.codigo, proc.cliente);
        }
    }

}
```

Exercício

Após implementar o exemplo, altere-o de forma que, após o cadastro inicial de processos, os mesmos sejam desempilhados e re-empilhados nas pilhas específicas de cada responsável (considere que a empresa possui apenas os três funcionários utilizados no exemplo). Após a re-empilhagem, imprima a pilha de cada responsável.

Acrescente um método à estrutura pilha que retorne a posição que um determinado processo (busque o processo pelo código) se encontra na pilha. Se o processo não for encontrado, lance uma exceção.

Exercício

Utilizando uma pilha de caracteres, desenvolva uma aplicação que verifique se os parenteses de uma determinada expressão estão ou não corretos. Por exemplo:

- **a+(b*c)-2-a** está correto
- **(a+b*(2-c)-2+a)*2** está correto
- **(a*b-(2+c)** está incorreto
- **2*(3-a))** está incorreto
- **)3+b*(2-c)(** está incorreto


рахмат
danke
謝謝
ngiyabonga
teşekkür ederim
Баярлалаа
спасибо
faafetai lava
mersi
barka
welalin
tack
dank je
misaotra
matondo
paldies
grazzi
mahalo
tapadh leat
vinaka
blagodaram
thank you
gracias
хвала
kiitos
dankie
dhanyavad
nanni
nandri
asante
manana
obrigada
tenki
enkosi
bedankt
hvala
koszonom
dankon
djiere dieuf
mochchakkeram
murakoze
dziękuję
chnorakaloutioun
gratias ago
gràcies
sulpay
go raibh maith agat
chokrane
sobodi
dėkuji
obrigado
sukriya
kop khun krap
taiku
arigatô
takk
dakujem
trugarez
mèsi
sagolun
ありがとう
tanemirt
grazie
najis tuke
terima kasih
rahmet
diolch
dhanyavadagalu
shukriya
merce
мерси
kam sah hamnida
rahmat
감사합니다
xiexie
ευχαριστώ
merci

Referências

- CARVALHO, Marco Antonio Moreira de. **Projeto e análise de algoritmos**. 01 mar. 2018, 15 jun. 2018. Notas de Aula. PPGCC. UFOP
- GOODRICH, Michael T.; TAMASSIA, Roberto. **Estruturas de Dados & Algoritmos em Java.** Bookman Editora, 2013.
- ZIVIANI, Nivio. **Projeto de Algoritmos com implementações em Java e C++**, 2007.