Estruturas de Dados I
Pilhas

Prof. Leonardo C. R. Soares - *leonardo.soares@ifsudestemg.edu.br*

Instituto Federal do Sudeste de Minas Gerais

5 de dezembro de 2024

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

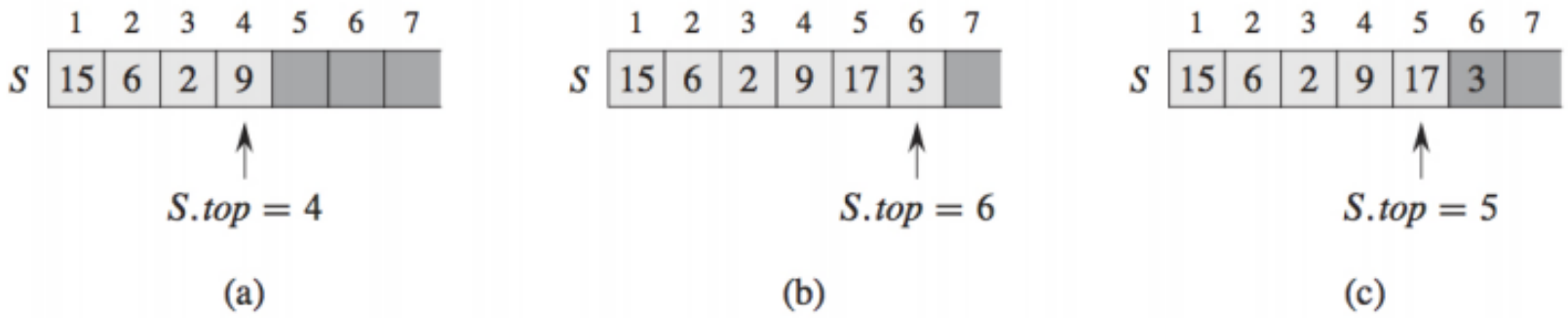
Existem várias opções de estruturas de dados que podem ser usadas para representar pilhas. As duas representações mais utilizadas são:

- 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

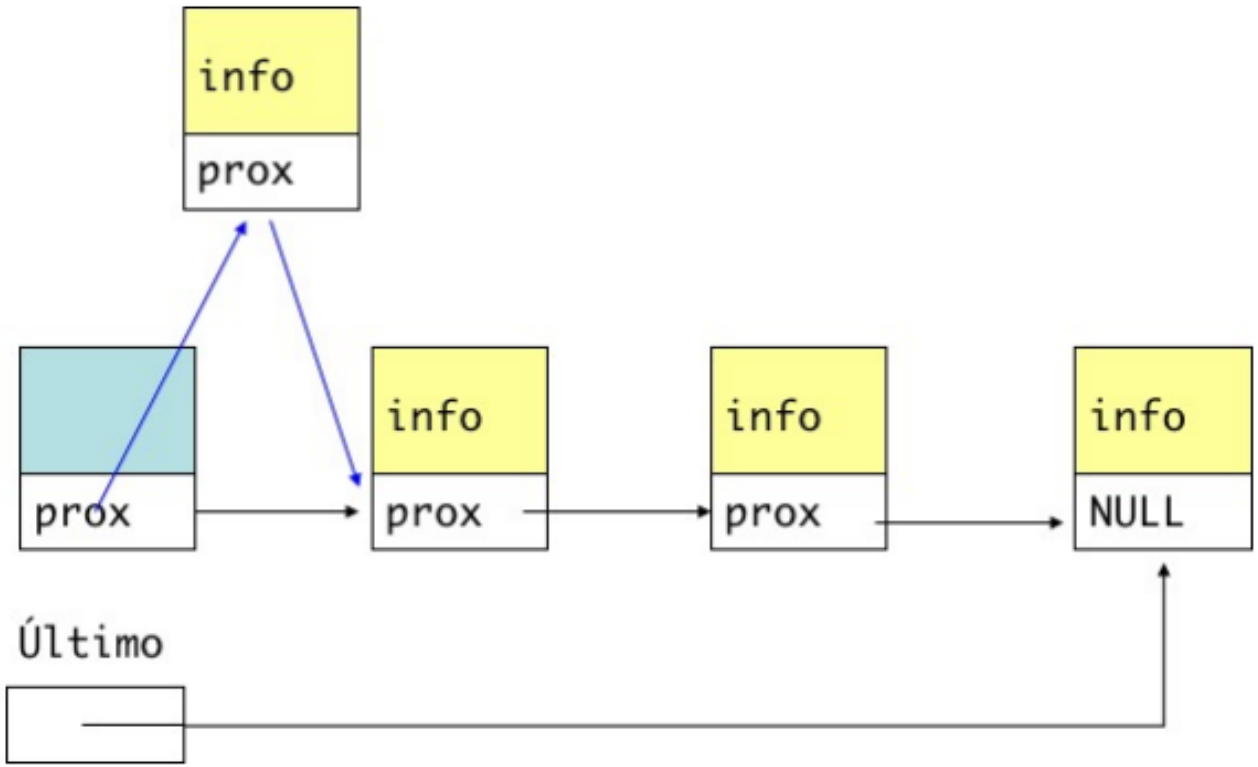
- Cada célula de uma pilha contém um item da pilha e um apontador para outra célula.
- A estrutura contém um apontador para o topo da pilha (célula cabeça).

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.

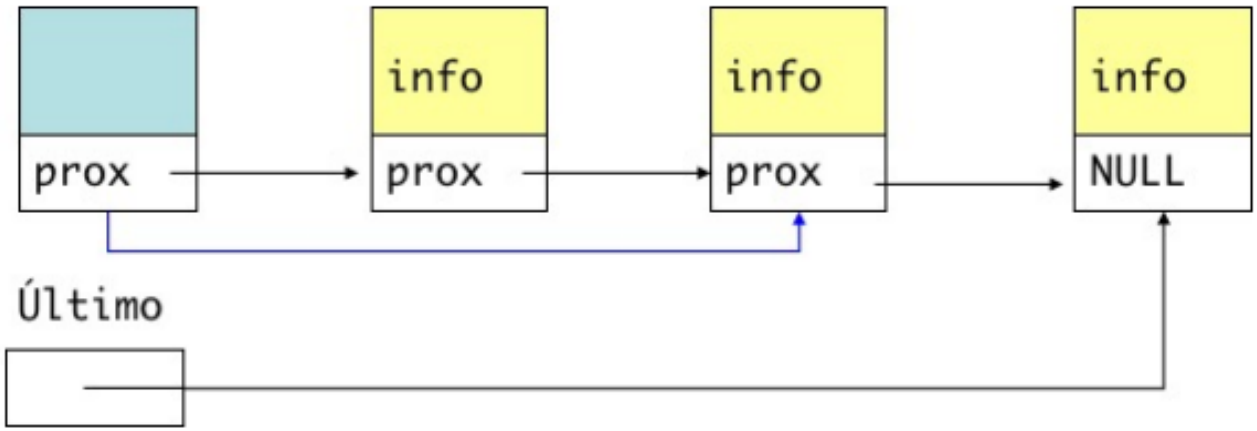
Implementação por referência - Inserção

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
    }
}
```

```
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);
        }
    }

}
```

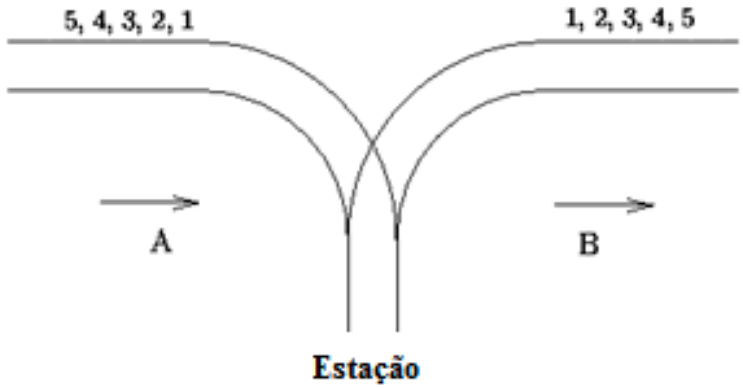
Exercício de fixação (não entra no GitHub)

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, retorne -1.

GitHub (entregar antes da próxima aula)

Há uma famosa estação de trem na cidade PopPush. Esta cidade fica em um país incrivelmente acidentado e a estação foi criada no último século. Infelizmente os fundos eram extremamente limitados naquela época. Foi possível construir somente uma pista. Além disso, devido a problemas de espaço, foi feita uma pista apenas até a estação (veja figura abaixo).



A tradição local é que todos os comboios que chegam vindo da direção A continuam na direção B com os vagões reorganizados, de alguma forma. Suponha que o trem que está chegando da direção A tem N <= 1000 vagões numerados **sempre** em ordem crescente **1,2, ..., N**. O primeiro que chega é o **1** e o último que chega é o **N**. Existe um chefe de reorganizações de trens que quer saber se é possível reorganizar os vagões para que os mesmos saiam na direção B na ordem a_1, a_2, a_n..

GitHub (entregar antes da próxima aula)

O chefe pode utilizar qualquer estratégia para obter a saída desejada. No caso do desenho ilustrado acima, por exemplo, basta o chefe deixar todos os vagões entrarem na estação (do 1 ao 5) e depois retirar um a um: retira o 5, retira o 4, retira o 3, retira o 2 e por último retira o 1. Desta forma, se o chefe quer saber se a saída 5,4,3,2,1 é possível em **B,** a resposta seria **Yes**. Vagão que entra na estação **só pode sair para a direção B** e é possível incluir quantos forem necessários para retirar o primeiro vagão desejado.

Entrada

O arquivo de entrada consiste de um bloco de linhas, cada bloco, com exceção do último, descreve um trem e possivelmente mais do que uma requisição de reorganização. Na primeira linha de cada bloco há um inteiro N que é a quantidade de vagões. Em cada uma das próximas linhas de entrada haverá uma permutação dos valores **1,2, ..., N**. A última linha de cada bloco contém apenas 0. Um bloco iniciando com zero (0) indica o final da entrada.

Saída

O arquivo de saída contém a quantidade de linhas correspondente às linhas com permutações no arquivo de entrada. Cada linha de saída deve ser **Yes** se for possível organizar os vagões da forma solicitada e **No**, caso contrário. Há também uma linha em branco após cada bloco de entrada. No exemplo abaixo, O primeiro caso de teste tem 3 permutações para 5 vagões. O ultimo zero dos testes de entrada não devem ser processados.

GitHub (entregar antes da próxima aula)

Exemplo de Entrada	Exemplo de Saída
5 5 4 3 2 1 1 2 3 4 5 5 4 1 2 3 0 6 1 3 2 5 4 6 0 0	Yes Yes No Yes

Tradução & Dataset por Neilor

Exercício (GitHub ao vivo)


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

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.