Algoritmos e Programação Estruturada

Pilhas

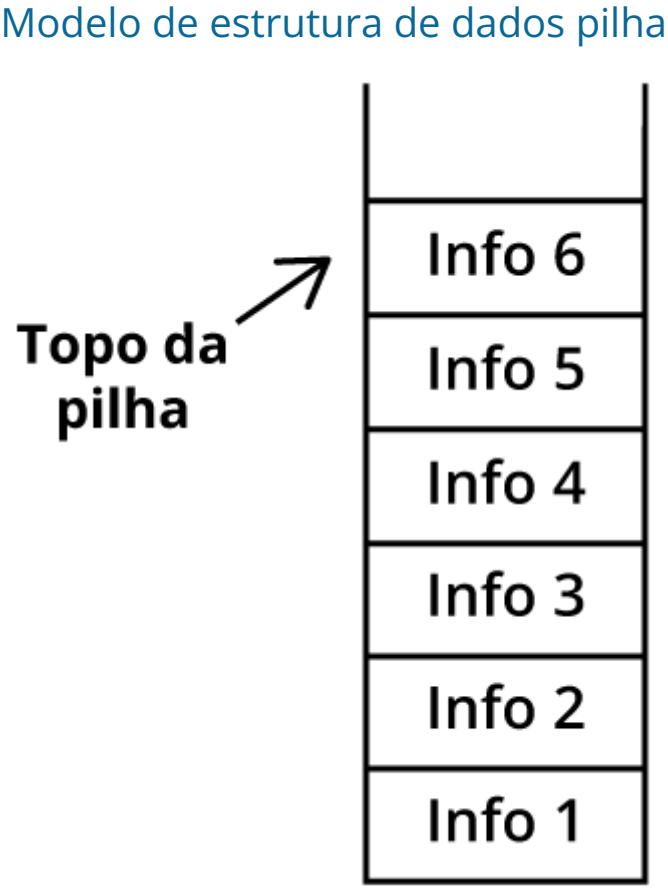
Você sabia que seu material didático é interativo e multimídia? Isso significa que você pode interagir com o conteúdo de diversas formas, a qualquer hora e lugar. Na versão impressa, porém, alguns conteúdos interativos ficam desabilitados. Por essa razão, fique atento: sempre que possível, opte pela versão digital. Bons estudos!

Nesta webaula, vamos conhecer o funcionamento das pilhas.

Pilhas

Uma pilha tem como definição básica ser um conjunto de elementos ordenados que permite a inserção e a remoção de mais elementos em apenas uma das extremidades da estrutura denominada topo da pilha. (TENENBAUM, LANGSAM e AUGENSTEIN, 2007). Assim, um novo elemento que é inserido passa a ser o topo da pilha, e o único elemento que pode ser removido da pilha é o elemento que está no topo.

Modelo de estrutura de dados pilha

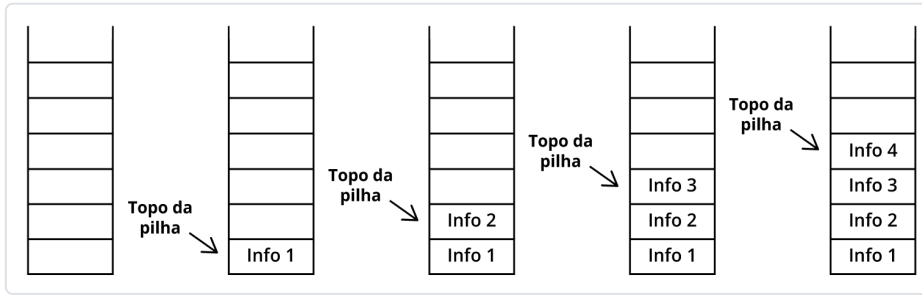


Fonte: elaborada pelo autor.

Os elementos inseridos em uma pilha apresenta, uma sequência de inserção. O primeiro elemento que entra na pilha só pode ser removido por último, após todos os outros elementos serem removidos.

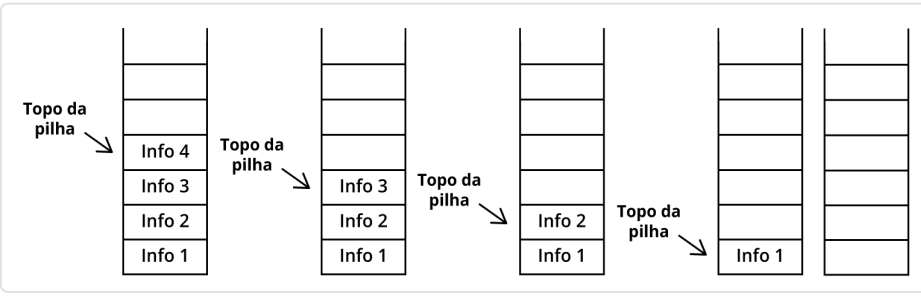
Segundo Celes, Cerqueira e Rangel (2004), os elementos da pilha só podem ser retirados na ordem inversa da ordem em que nela foram inseridos. Isso é conhecido como LIFO (*last in*, *first out*, ou seja, o último que entra é o primeiro a sair) ou FILO (*first in*, *last out*, ou seja, primeiro que entra é o último a sair).

Inserindo elemento na pilha



Fonte: elaborada pelo autor.

Removendo um elemento da pilha



Fonte: elaborada pelo autor.

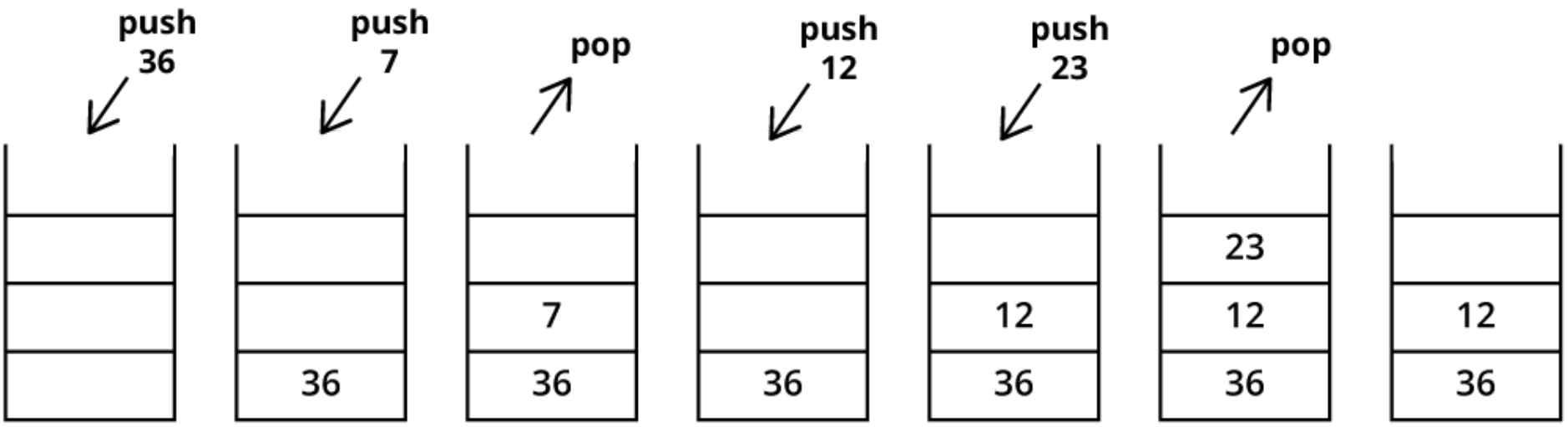
Operações de pilhas

Uma pilha também pode estar no estado de pilha vazia quando não houver elementos. Segundo Celes, Cerqueira e Rangel (2004), os passos para a criação de uma pilha são:

- Criar uma pilha vazia.
- Inserir um elemento no topo.
- Remover o elemento do topo.
- Verificar se a pilha está vazia.
- Liberar a estrutura de pilha.

Empilhar e desempilhar

A operação de empilhar um novo elemento, segundo Lorenzi, Mattos, Carvalho (2015), tem a função de inserir um novo elemento na pilha, e definida na programação em C++ como `push()`. Já a operação de desempilhar , tem a função de remover um elemento do topo da Pilha e utilizada na programação em C++ como `pop()`.



Fonte: elaborada pelo autor.

Conforme Drozdek(2016), outras operações que podem ser implementadas na pilha são as funções `clear()`, para limpar a
pilha e `isEmpty()`, para verificar se a pilha está vazia.

Conforme Tenenbaum (2007), uma pilha possui uma estrutura que pode ser declarada contendo dois objetos:

- Um ponteiro, que irá armazenar o endereçamento inicial da pilha.
- Um valor inteiro, que irá indicar a posição no topo da pilha.

A seguir, veja as implementações de pilha:

Declaração da estrutura inicial

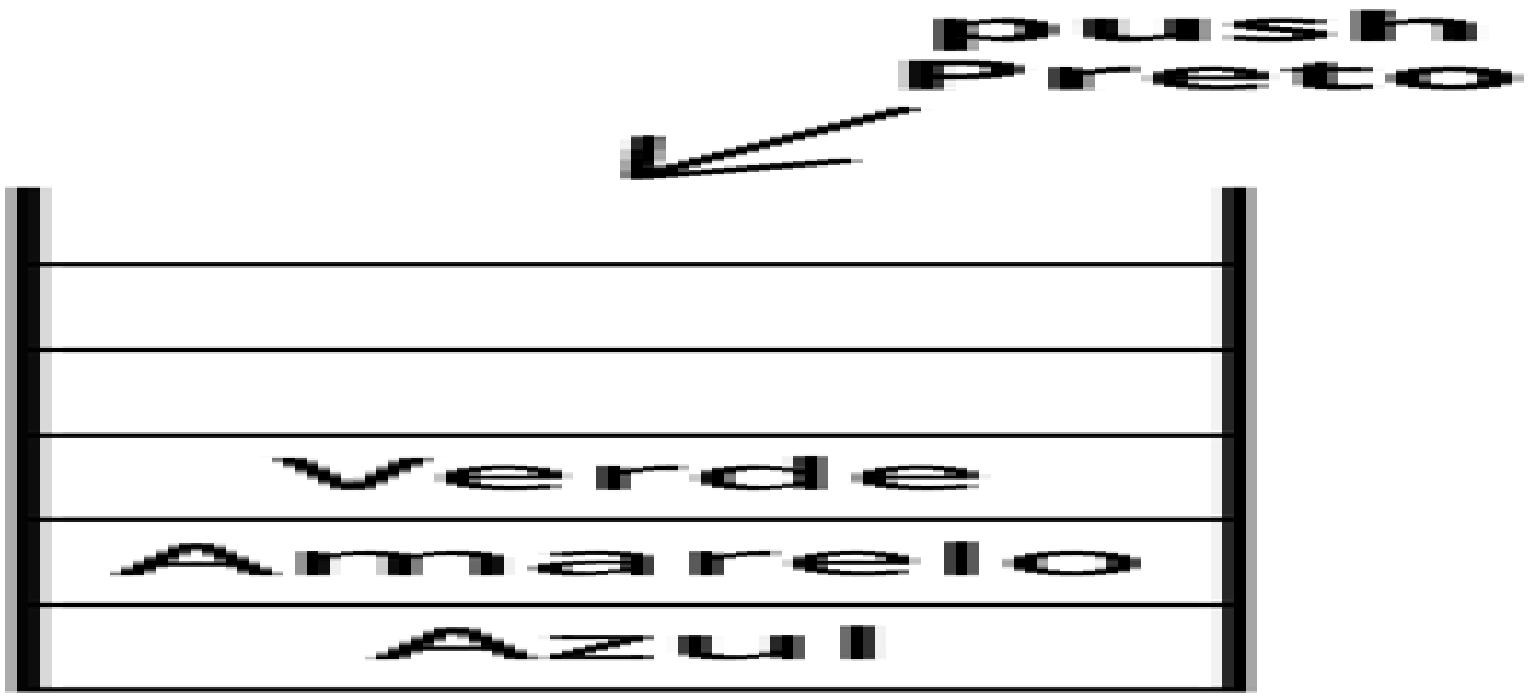
```
struct Pilha {
        int topo;
        int capacidade;
        float * proxElem;
};
struct Pilha minhaPilha;
```

Criar a pilha

```
void cria_pilha(struct Pilha *p, int c ){
   p -> proxElem = (float*) malloc (c * sizeof(float));
   p -> topo = -1;
   p -> capacidade = c;
}
```

Inserir elemento na pilha

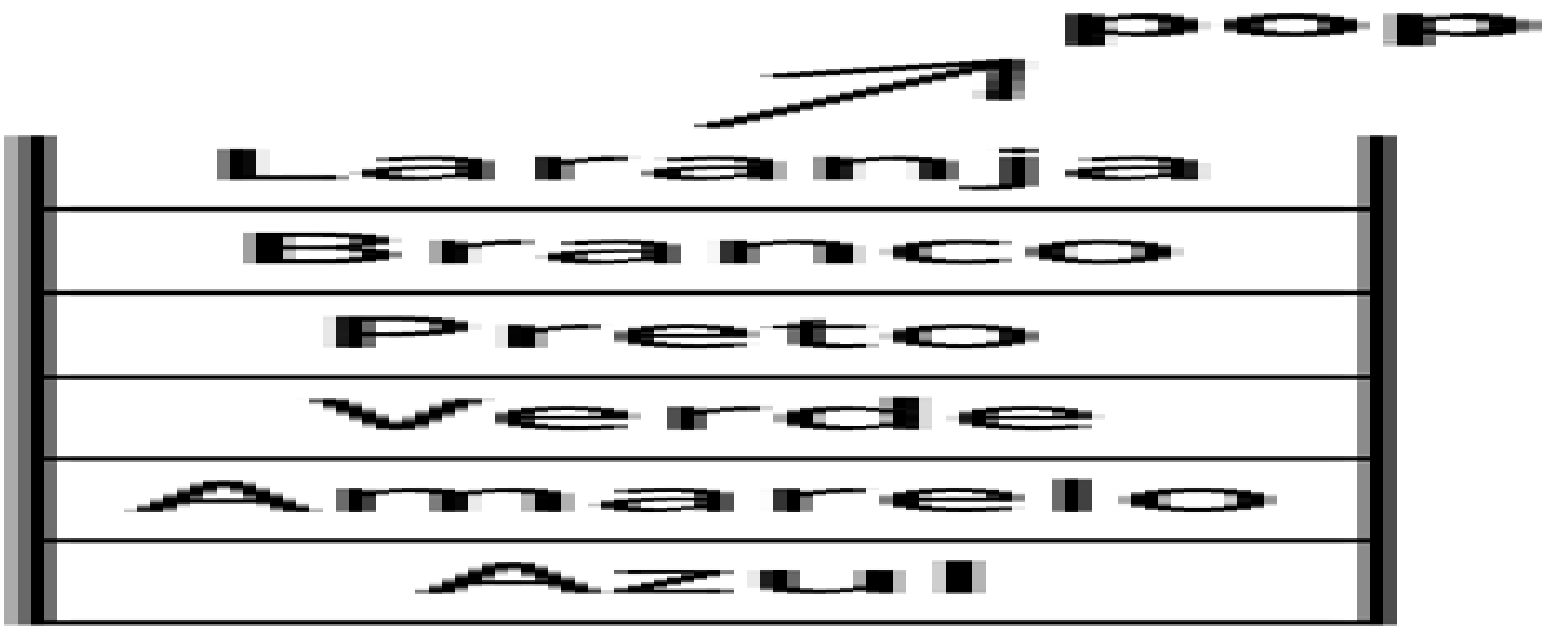
```
void push_pilha(struct Pilha *p, float v){
        p -> topo++;
        p -> proxElem [p -> topo] = v;
}
```



Fonte: elaborada pelo autor

Remover elemento da pilha

```
float pop_pilha (struct Pilha *p){
   float aux = p -> proxElem [p -> topo];
     p -> topo--;
   return aux;
}
```



Fonte: elaborada pelo autor

Informar se a pilha está vazia

```
/*Declaração da função pilha_vazia com passagem da pilha por parâmetro*/
int pilha_vazia (struct Pilha *p ){
   if( p -> topo ==  -1 )
      return 1; /*Sendo o topo igual a -1, a função retorna verdadeiro*/
   else
      return 0; /*Caso contrário, a função retorna verdadeiro*/
}
```

Verificar se a pilha está cheia

```
int pilha_cheia ( struct Pilha *p ){

        if (p -> topo == p -> capacidade - 1)
                return 1;
        else
                return 0;
}
```

Problema do Labirinto

Os labirintos são desafios criados como problematização de estrutura de dados.

Em um labirinto, por exemplo, para encontrar um caminho correto, podemos andar pelo labirinto até que se encontre uma divisão neste caminho. Caso o caminho escolhido não possua uma saída, é removido o ponto anterior da pilha, voltando ao último ponto em que o labirinto se dividiu e recomeçamos por um outro caminho ainda não escolhido, adicionando na pilha o novo caminho.

A seguir, um trecho de implementação de criação de uma solução para labirinto:

```
/*Chama a função inicLabirinto, passando o labirinto, a pilha, o valor da linha e da coluna como passagem de
parâmetros*/
void inicLabirinto(Labirinto *l, Pilha *p_l, int linha, int coluna){
  int i, j, flag = 0;
  char aux;
  elem_t_pilha origem;
/*Aplicamos uma rotina em matriz para verificar se a posição foi visitada (1) ou não (0)*/
  for(i = 0 ; i < linha ; i++){
    for(j = 0 ; j < coluna ; j++){
      if(l->p[i][j].tipo == '0'){
        l->p[i][j].visitado = 1; //Visitado
        origem.x = i;
        origem.y = j;
        /*Insere na pilha a posição de origem*/
        push(p_l, origem);
      }
    }
  }
}
```

Backtracking

As pilhas podem ser aplicadas também no uso de algoritmos de *Backtracking*, que consiste em criar marcações para onde o algoritmo pode retornar. Podem ser aplicados também como operação de desfazer, existente em diversas aplicações de usuários, como a utilização deste algoritmo em sistema de GPS. Quando o motorista utiliza uma rota não indicada pelo programa, o algoritmo de *Backtracking* é aplicado para redefinir a nova rota.

O algoritmo de *Backtracking* tem como meta resolver o problema no menor intervalo de tempo possível, sem levar em consideração o esforço de operações para alcançar a solução do problema.