Pilhas

Juliana Kaizer Vizzotto

DLSC/PPGI
Universidade Federal de Santa Maria

Pilhas: Introdução

- Uma das estruturas de dados mais simples e mais utilizadas em programação.
- Implementada pelo hardware da maioria das máquinas modernas.
- Ideia fundamental: todo acesso é feito através do seu topo.
- Estratégida de acesso: LIFO (*Last In, First Out*).
- Operações básicas: *push* (empilha) e *pop* (desempilha).

Pilhas: Introdução

- Exemplo de utilização: pilha de execução da linguagem C. As variáveis locais das funções são dispostas numa pilha e uma função só tem acesso às variáveis que estão no topo (não é possível acessar as variáveis da função locais às outras funções).
- Implementação: há várias implementações possíveis de uma pilha, vamos estudar duas em aula: pilha como vetor e pilha como lista encadeada.

Pilhas: Implementação como vetor

- Quando sabemos de antemão o tamanho da pilha.
- Teremos um vetor, `vet` para armazenar os elementos da pilha. Os elementos inseridos ocupam as primeiras posições do vetor. Assim, se temos n elementos armazenados na pilha, o elemento `vet[n-1]` representa o elemento do topo da pilha.

```
#define MAX 50

struct pilha{
   int n;
   float vet[MAX];
};
```

Pilhas: Implementação como vetor

- Criar a pilha: aloca dinamicamente e inicializa a pilha como vazia, i.e., com o número de elementos igual a zero.

```
Pilha* cria(void){
    Pilha* p =(Pilha*) malloc(sizeof(Pilha));
    p->n=0;
    return p;
}
```

Pilhas: Implementação como vetor

- Inserir um elemento na pilha: usamos a próxima posição do vetor.

```
void push(Pilha* p, float v){
    if (p->n ==MAX){
        printf("Capacidade da pilha estourou.\n");
        exit(1); //aborta programa
    }
  p->vet[p->n] = v;
  p->n++;
}
```

Pilhas: Implementação como vetor

- Retirar um elemento da pilha,

```
float pop(Pilha* p){
   float v;
   if (vazia(p)){
      printf("Pilha vazia.\n");
      exit(1); //aborta programa
   }
  v = p->vet[p->n-1];
  p->n--;
  return v;
}
```

Pilhas: Implementação como vetor

- Vazia e Libera

```
int vazia(Pilha* p){
   return (p->n==0);
}

void libera(Pilha* p){
   free(p)
}
```

Pilhas: Implementação como lista encadeada

- Os elementos são armazenados na lista e a pilha pode ser representada por um ponteiro para o primeiro nó da lista.

```
struct no {
    float info;
    struct no* prox;
};
typedef struct no No;

struct pilha{
    No* prim;
};
typedef struct pilha Pilha;
```

Pilhas: Implementação como lista encadeada

- Cria: aloca a estrutura pilha e inicializa a lista vazia

```
Pilha* cria(void){
  Pilha* p = (Pilha*) malloc(sizeof(Pilha));
  p->prim ==NULL;
  return p;
}
```

- O primeiro elemento da lista representa o topo da pilha. Cada novo elemento é inserido no início da lista. Também retira-se elementos do início da lista.

```
No* ins_ini(No* l,float v){
  No* p = (No*) malloc(sizeof(No));
  p->info = v;
  p->prox = l;
  return p;

}
```

Pilhas: Implementação como lista encadeada

- Retira início:

```
No* ret_ini(No* l){
  No* p = l->prox;
  free(l);
  return p;
}
```

- Push

```
void push(Pilha* p, float v){
  p->prim = ins_ini(p->prim,v);
}
```

Pilhas: Implementação como lista encadeada

- Pop:

```
float pop(Pilha* p){
  float v;
  if (vazia(p)){
    printf("Pilha vazia.\n");
    exit(1);
  }
  v = p->prim->info;
  p->prim=retu_ini(p->prim);
  return v;
}
```

- Vazia?

```
int vazia(Pilha* p){
  return (p->prim==NULL);
}
```

Pilhas: Implementação como lista encadeada

- Libera:

```
void libera(Pilha* p){
  No* q = p->prim;
  While (q!=NULL){
    No*t = q->prox;
    free(q);
    q=t;
 }
 free(p);
}
```

- Imprimir:

```
void imprime(Pilha* p){
  No* q;
  for(q=p->prim;q!=NULL;q=q->prox)
    printf("%f\n",q->info);
}
```

Pilhas: Exercícios

- Considere o problema de decidir se uma dada sequência de parênteses e colchetes está bem-formada (ou seja, parênteses e colchetes são fechados na ordem inversa aquela em que foram abertos).