Algoritmos
Estructuras de Datos I

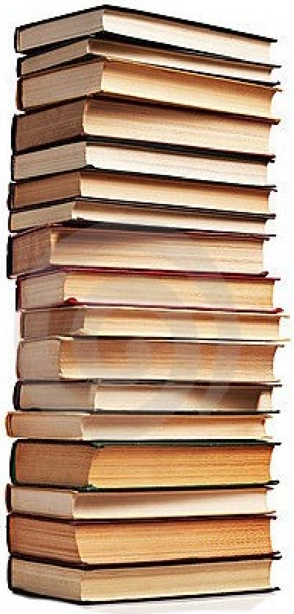
Facultad de Ciencias Exactas y Tecnología

Universidad Nacional de Tucumán

2023

Universidad Nacional de Tucumán
facet
Facultad de Ciencias Exactas y Tecnología

PILA-STACK(1)



PILA o STACK

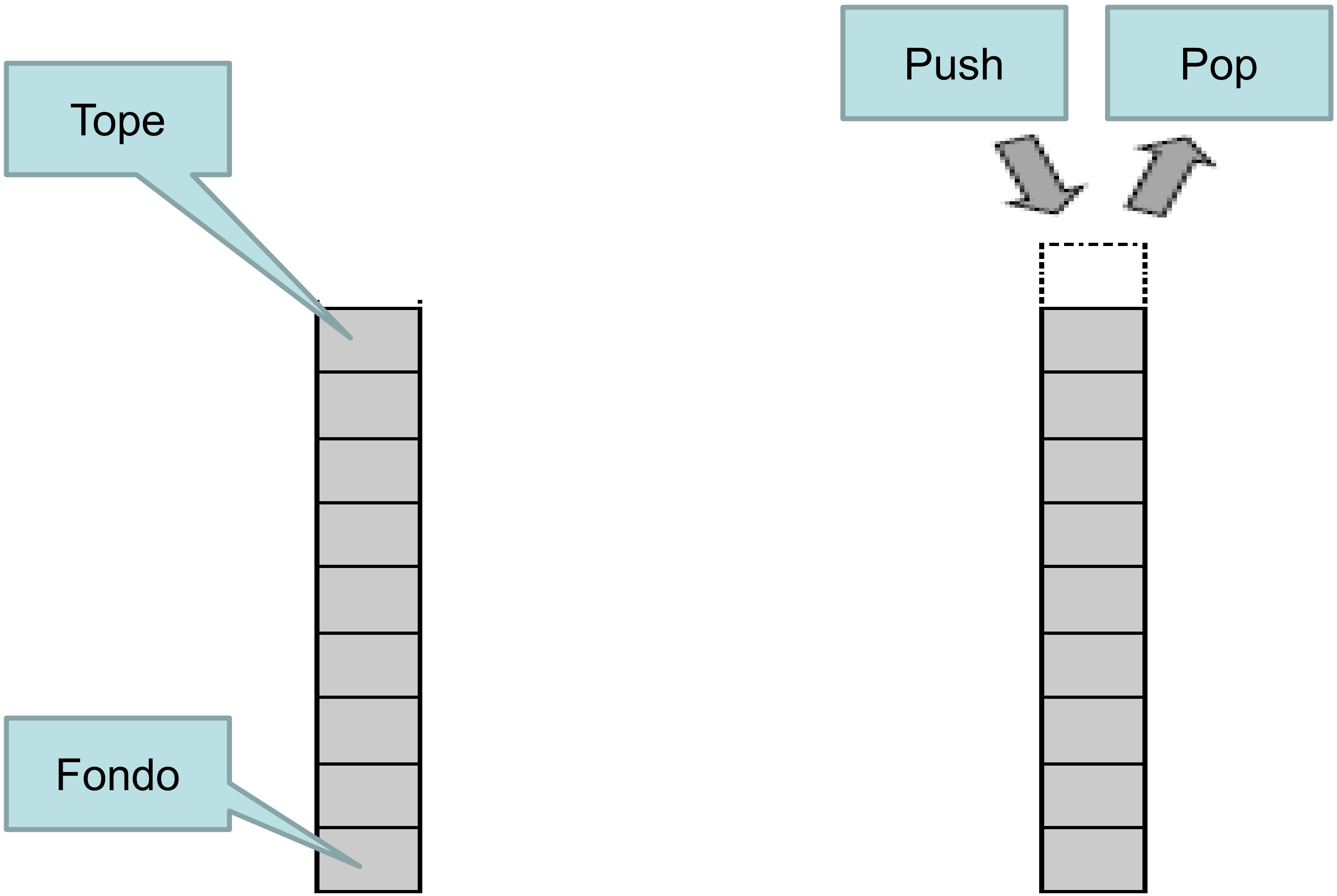
Una pila es una colección ordenada de cero o mas *objetos de un mismo tipo* que solamente puede crecer o decrecer por uno solo de sus extremos. Ese extremo se llama **tope** o **cabeza** de la pila. El otro extremo que es inaccesible se llama el **fondo** de la pila.

Una pila se llama también una estructura de tipo **LIFO** (last in, first out), que posee la siguiente propiedad: el ultimo elemento que llega a una pila es el primero en ser sacado.

Las pilas son estructuras de datos fáciles de definir y de implementar, y además son muy eficientes.

PILA

LIFO (last in, first out)



PILA(ITEM)
Especificación Algebraica

OPERACIONES

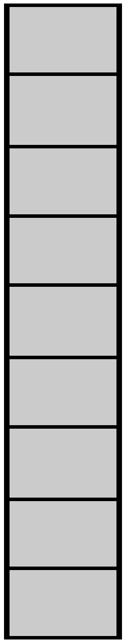
A) Sintaxis:

PILAVACIA	: → PILA
ESPILAVACIA	: PILA → BOOL
TOP	: PILA → ITEM U {indefinido}
POP	: PILA → PILA
PUSH	: PILA x ITEM → PILA

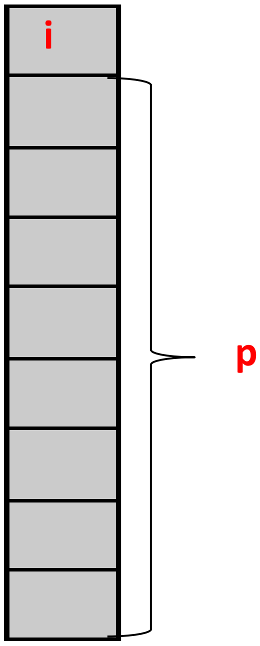
PILA

Constructoras:

PILAVACIA



PUSH (p, i)



PILA(ITEM)
Especificación Algebraica

OPERACIONES

B) Semántica: Para todo $p \in$ PILA , $\forall$ $i \in$ ITEM

ESPILAVACIA(PILAVACIA) $\equiv$ TRUE

ESPILAVACIA(PUSH(p,i)) $\equiv$ FALSE

POP(PILAVACIA) $\equiv$ PILAVACIA

POP(PUSH(p,i)) $\equiv$ p

TOP(PILAVACIA) $\equiv$ indefinido

TOP(PUSH(p,i)) $\equiv$ i

PILA
Aplicación: altura de una Pila

1) Dentro de la Especificación Algebraica

Sintaxis:

ALTURA : PILA $\rightarrow$ entero ≥ 0

Semántica: Para todo p $\in$ PILA, $\forall$ i $\in$ ITEM

ALTURA (PILAVACIA) $\equiv$ 0
ALTURA (PUSH (p,i)) $\equiv$ 1 + ALTURA(p)

PILA

Aplicación: altura de una Pila

2) **Como usuario** **del ADT PILA,** función *recursiva*

```
Funcion ALTURA(p) : PILA  → entero ≥ 0
        Si ESPILAVACIA (p) entonces
                Retorna 0
        Sino
                Retorna 1 + ALTURA(POP(p))
Fin
```

NOTA: la función POP modifica la pila.

PILA
Aplicación: altura de una Pila

3) Como usuario del ADT PILA, función *iterativa*

```
Funcion ALTURA(p) : PILA  → entero ≥ 0
Auxiliar: h ∈ entero
          h ← 0
          Mientras NOT ESPILAVACIA(p) hacer
                    h ← h+1
                    POP (p)
          Retorna h
 Fin
```

PILA
Aplicación: igualdad de pilas

1) Dentro de la Especificación Algebraica (versión 1)

Sintaxis:

IGUALP **:** PILA x PILA → BOOL

Semántica: $\forall$ p,q $\in$ PILA, $\forall$ i,j $\in$ ITEM

IGUALP(PILAVACIA, PILAVACIA) $\equiv$ TRUE

IGUALP(PILAVACIA, PUSH(p,i)) $\equiv$ FALSE

IGUALP(PUSH(p,i),PILAVACIA) $\equiv$ FALSE

IGUALP(PUSH(p,i),PUSH(q,j)) $\equiv$ Si NOT i**=**j entonces

FALSE

Sino IGUALP(p,q)

donde **=** *es la operación* ***IGUALITEM*** *que permite comparar 2 objetos del tipo item*

PILA
Aplicación: igualdad de pilas

1) Dentro de la Especificación Algebraica (versión 2)

Sintaxis:

IGUALP **:** PILA x PILA $\rightarrow$ BOOL

Semántica: $\forall$ p,q $\in$ PILA, $\forall$ i,j $\in$ ITEM

IGUALP(PILAVACIA, PILAVACIA) $\equiv$ TRUE

IGUALP(PILAVACIA, PUSH(p,i)) $\equiv$ FALSE

IGUALP(PUSH(p,i),PILAVACIA) $\equiv$ FALSE

IGUALP(PUSH(p,i),PUSH(q,j)) $\equiv$ i**=**j AND IGUALP(p,q)

donde **=** *es la operación* ***IGUALITEM*** *que permite comparar 2 objetos del tipo item*

PILA
Aplicación: igualdad de pilas

2) Como usuario del ADT PILA, función *recursiva*

Funcion **IGUALP** (p1,p2) : PILA x PILA → BOOL

```
Si ESPILAVACIA(p1) AND ESPILAVACIA(p2) entonces
    Retorna TRUE
Sino
     Si ESPILAVACIA(p1) OR ESPILAVACIA(p2) entonces
          Retorna FALSE
    Sino
          Retorna
          TOP(p1)=TOP(p2) AND IGUALP(POP(p1),POP(p2))
Fin
```

PILA

Aplicación: igualdad de pilas

3) Como usuario del ADT PILA, función *iterativa*

Funcion **IGUALP** (p1,p2) : PILA x PILA → BOOL

```
    Mientras   NOT ESPILAVACIA(p1) AND
               NOT ESPILAVACIA(p2) AND
               TOP(p1)=TOP(p2)          hacer
                     POP(p1)
                     POP(p2)
    Retorna ESPILAVACIA(p1) AND ESPILAVACIA(p2)
Fin
```

PILA
Aplicación: apilar 2 pilas

Una operación que dadas dos pilas apile los objetos de la 2da sobre la 1ª conservando el orden.

Por ejemplo:

p1		p2		p1+p2
				z
				y
				x
d				d
c		z		c
b	+	y	→	b
a		x		a

<u>PILA</u>

Aplicación: apilar 2 pilas

1) Dentro de la Especificación Algebraica

Sintaxis:

APILA : PILA x PILA → PILA

Semántica: ∀ p1,p2 ∈ PILA, ∀ i,j ∈ ITEM

APILA (PILAVACIA , PILAVACIA) ≡ PILAVACIA

APILA (PUSH(p1,i) , PILAVACIA) ≡ PUSH(p1,i)

APILA (PILAVACIA , PUSH(p2,j)) ≡ PUSH(p2,j)

APILA (PUSH(p1,i) , PUSH(p2,j)) ≡

PUSH (APILA (PUSH(p1,i),p2) , j)

PILA
Aplicación: apilar 2 pilas

1) Dentro de la Especificación Algebraica con 2 axiomas

Sintaxis:

APILA : PILA x PILA $\rightarrow$ PILA

Semántica: $\forall$ p1,p2 $\in$ PILA, $\forall$ j $\in$ ITEM

APILA (p1 , PILAVACIA) $\equiv$ p1

APILA (p1 , PUSH(p2,j)) $\equiv$ PUSH (APILA (p1,p2) , j)

PILA
Aplicación: apilar 2 pilas

2) Como usuario del ADT PILA, función *recursiva*

Funcion **APILA** (p1,p2) : PILA x PILA → PILA

Si ESPILAVACIA(p2) entonces

Retorna p1

Sino

Retorna PUSH (**APILA** (p1,POP(p2)) ,TOP(p2))

Fin

PILA
Aplicación: apilar 2 pilas

2) Como usuario del ADT PILA, función *recursiva* con aux

Funcion **APILA** (p1,p2) : PILA x PILA → PILA

Auxiliar: t∈ITEM, paux∈PILA

Si ESPILAVACIA(p2) entonces

Retorna p1

Sino

t←TOP(p2)

POP(p2)

paux←**APILA** (p1, p2)

Retorna PUSH (paux,t)

Fin

PILA
Aplicación: apilar 2 pilas

3) Como usuario del ADT PILA, función *iterativa*

Funcion **APILA** (p1,p2) : PILA x PILA → PILA

Auxiliar: paux∈PILA, paux ← PILAVACIA

Mientras NOT ESPILAVACIA(p2) hacer

PUSH(paux, TOP(p2))

POP(p2)

Mientras NOT ESPILAVACIA(paux) hacer

PUSH(p1, TOP(paux))

POP(paux)

Retorna p1

Fin

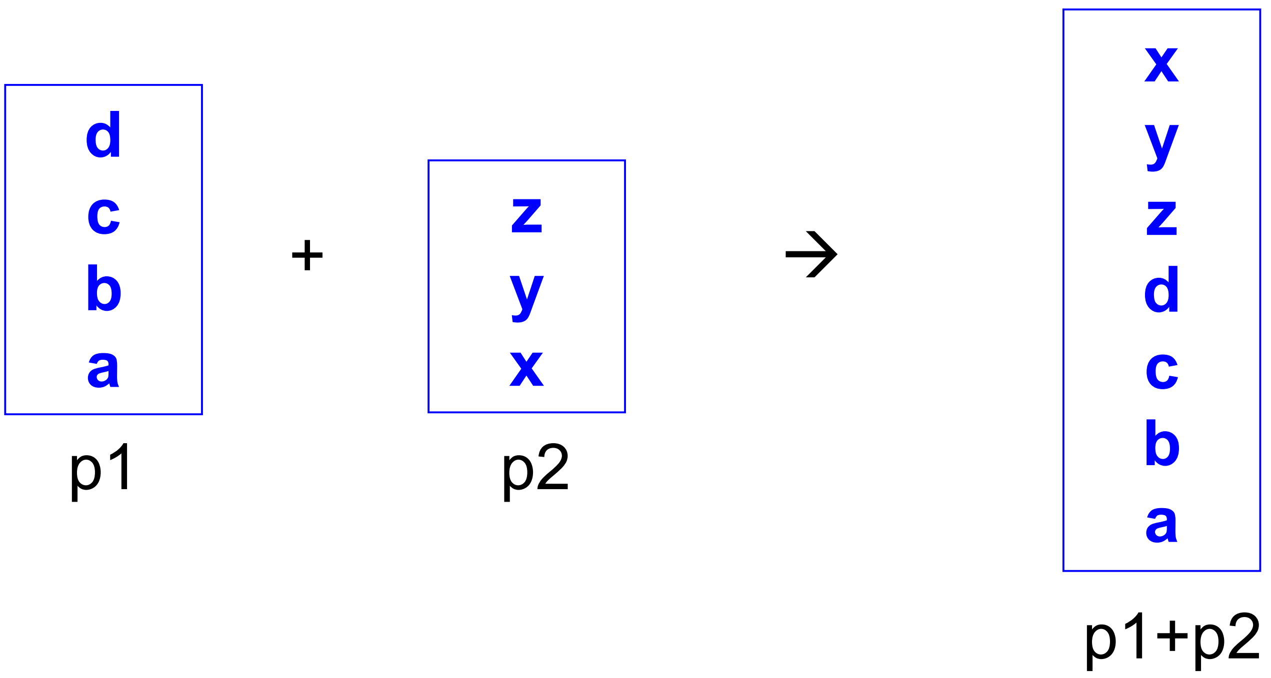
PILA

Aplicación: apilar 2 pilas

Una operación que dadas dos pilas apile la 2da invertida sobre la 1a.

EJERCITACION

Por ejemplo:



PILA
Aplicación: parentesis balanceados

Escribir un algoritmo que dada una expresión aritmética permita ***decidir si tiene paréntesis balanceados.***

Dada una expresión aritmética, para decir que tiene paréntesis balanceados se tiene que cumplir:

- hay el mismo numero de paréntesis izquierdos que derechos
- todo paréntesis derecho **)** esta precedido por su correspondiente izquierdo **(** .

PILA
Aplicación: parentesis balanceados

Se puede considerar la expresión aritmética como una tira de caracteres con marca final.

Para chequear los paréntesis balanceados se la recorre secuencialmente de izquierda a derecha.

La tira tendrá paréntesis balanceados si:

- cada vez que se encuentra) ya se ha encontrado un correspondiente (.
- cuando se alcanza el fin de la tira, cada (encontró su correspondiente).

PILA

Aplicación: parentesis balanceados

La solución requiere que se lleve la cuenta de cada **(** no cerrado y se lo descarte cada vez que se encuentre su par **)**.

Para ello, se puede guardar los **(** en una pila y cada vez que se encuentra **)** sacarlo de la pila. Al final la pila debe quedar vacía.

Es claro que no hace falta usar una *pila* para hacer este control sino simplemente un *contador*.

- Comienza con cont←0
- Llega un **(** entonces cont++
- Llega un **)** entonces si cont>0 cont- - sino error
- Debe finaliza con cont=0 sino error

PILA

Aplicación: delimitadores balanceados

Escribir un algoritmo que dada una expresión aritmética permita ***decidir si tiene delimitadores balanceados.***

Suponga que existen tres tipos diferentes de ***delimitadores de ámbito***, los cuales se indican mediante:

- paréntesis: **( )**
- corchetes: **[ ]**
- llaves: **{ }**

IMPORTANTE: *el tipo de delimitador que cierra el ámbito debe ser el mismo que lo abre.*

PILA
Aplicación: delimitadores balanceados

Por ejemplo las siguientes cadenas serán validas:

(A+B) ✓

[(A-B)/2] ✓

{A*[-B]} ✓

({A*B}/[C+D]) ✓

Por ejemplo las siguientes cadenas no serán validas:

(A+B] ✘

[(A-B)]) ✘

{A*(-B]} ✘

(A/(B-4))) ✘

ALGORITMO BALANCE

ENTRADA: tira de caracteres con MF

SALIDA: Valida ∈ BOOL

Auxiliar: p ∈ PILA, c , t ∈ CHAR, Valida ∈ BOOL

```
P1. Leer ( c )
P2. PILAVACIA(p)
P3. Valida ← TRUE
P4. Mientras c ≠ MF AND Valida hacer
        Si   c='('   OR  c='['   OR    c='{'     entonces       PUSH(p,c)
        sino Si  c=')'   OR    c=']'    OR    c='}'      entonces
                Si ESPILAVACIA(p) entonces     Valida ← FALSE
                sino
                    t←TOP(p)
                  Si (t ='(' AND c=')') OR  (t='[' AND c=']' ) OR (t='{' AND c='}' )
                      entonces      POP(p)
                  sino
                      Valida ← FALSE
        Leer ( c )
P5. Si NOT ESPILAVACIA(p) entonces        Valida ← FALSE
P6. Escribir (Valida)
P7. Fin
```