TIPOS ABSTRACTOS DE DATOS
T.A.D. PILA

ESTRUCTURAS DE DATOS
y ALGORITMOS
LCC - LSI - TUPW

Objetivos

- Construir los TADs Pila, Cola y Lista.
- Analizar, evaluar y comparar distintas alternativas de representación.
- Evaluar, clasificar en el marco de análisis de eficiencia de algoritmos, y comparar distintas soluciones algorítmicas.
- Resolver problemas típicos.

PILAS

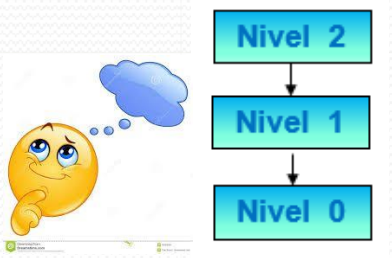
- Las Pilas son secuencias de elementos que pueden crecer y contraerse siguiendo la política: *Ultimo en Entrar, Primero en Salir*

- Se las suele llamar también Listas LIFO- Last In First Out- , o Listas "último en entrar primero en salir", o Stack.

- Entre los elementos existe un *orden temporal*

T.A.D. PILA

Especificación



Pila: Secuencia de cero o mas elementos de un tipo determinado, que crece y se contrae según la política LIFO.

P = (a1, a2, ..., an) , n>=0

El orden temporal de inserciones en este conjunto determina completamente el orden en el cual los elementos serán recuperados.

- a1: primer elemento ingresado.
- an: último elemento ingresado, primer elemento a ser retirado. (Se dice que an es el elemento que se encuentra en el tope o cima de la pila)

T.A.D. PILA

Especificación

P = (a1, a2, ..., an) , n>=0

Insertar en P, el elemento X

P = (a1, a2, ..., an, X) , n>=0

P = (a1, a2, ..., an) , n>0

Suprimir de P el elemento X

P = (a1, a2, ..., an-1) , X=an y n>=0

T.A.D. PILA

Especificación

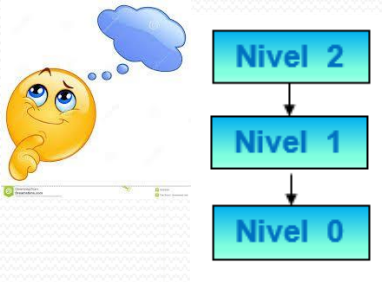
Operaciones Abstractas

Sean **P**: Pila y **X**: elemento

NOMBRE	*ENCABEZADO*	*FUNCION*	*ENTRADA*	*SALIDA*
Insertar	Insertar (P,X)	Ingresa el elemento X en la pila P	P y X	$P=(a_1,a_2,...,a_n,X)$
Suprimir	Suprimir(P,X)	Si P no está vacía, elimina el elemento que fue insertado mas recientemente	P	$P=(a_1,a_2,...a_{n-1})$ y $X= a_n$ si n>0; Error en caso contrario
Recorrer	Recorrer(P)	Procesa todos los elementos de P siguiendo la política LIFO	P	Está sujeta al proceso que se realiza sobre los elementos de P
Crear *	Crear(P)	Inicializa P	P	P=( )
Vacía *	Vacía(P)	Evalúa si P tiene elementos	P	Verdadero si P No tiene elementos, Falso en caso contrario.

T.A.D. PILA

Aplicación



Se desea controlar la correspondencia de ´**[**´ / ´**]**´ , ´**{**´ / ´**}**´ y ´**(**´ / ´**)**´ en una expresión aritmética, en la que los identificadores de los operandos están formados por un solo carácter

Expresión de Entrada	Mensaje de Salida
• {[(A-B)*C]^D}	CORRESPONDENCIA
• {([A-B)*C]^D}	ERROR DE CORRESPONDENCIA
• {[(A-B)*C]^D	ERROR DE CORRESPONDENCIA
• {[A-B)*C]^D}	ERROR DE CORRESPONDENCIA

T.A.D. PILA

Aplicación (2)

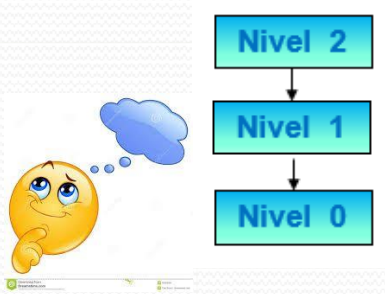
```
Crear(P)
Recuperar ( Expresión, X )
MIENTRAS  (No fin de Expresión)  y  (No Error)
    SI  ( X = " [ " ó  X = " { "  ó   X = "(" )
                ENTONCES
                        Insertar ( P , X)
    FIN SI
    SI ( X = "]"  ó X = " }"  ó   X = ")" )
                ENTONCES
                        Suprimir (P, aux)
                        SI (No Error)
                                ENTONCES
                                    SI (X = " ] "  y   aux ≠ "[" )  ó
                                        (X = " } "  y   aux ≠ "{ " )  ó
                                                                (X = " ) "  y   aux ≠ " (" )
                                        ENTONCES
                                                Error
                                    FIN SI
                         FIN SI
     FIN SI
     Recuperar ( Expresión, X )
FIN MIENTRAS
SI ( No (Vacía (P)) ó (Error)
                        ENTONCES
                                " ERROR DE CORRESPONDENCIA"
                        SINO
                                " CORRESPONDENCIA"
FIN SI
```

Ver Video Ejemplo de uso TAD PILA.mp4

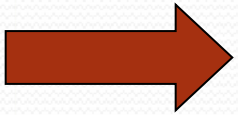
T.A.D. PILA

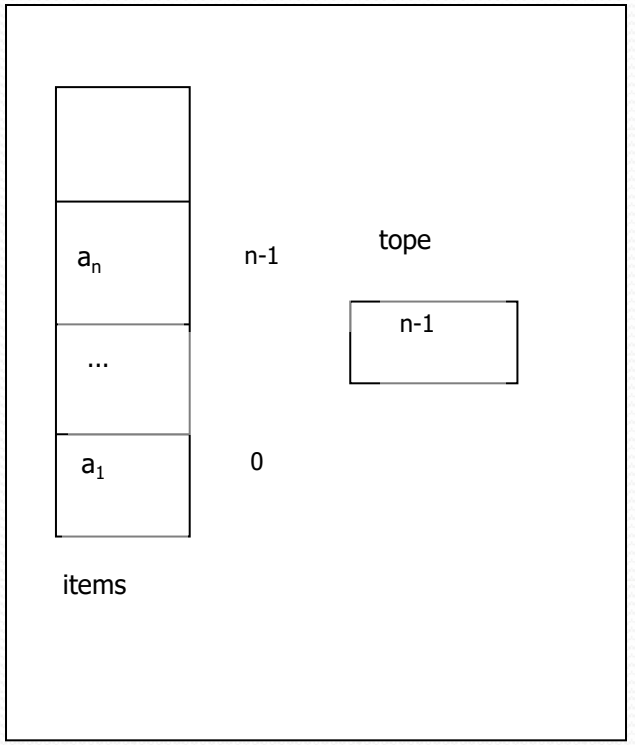
Representación

Representación secuencial



P = (a1, a2, ..., an)



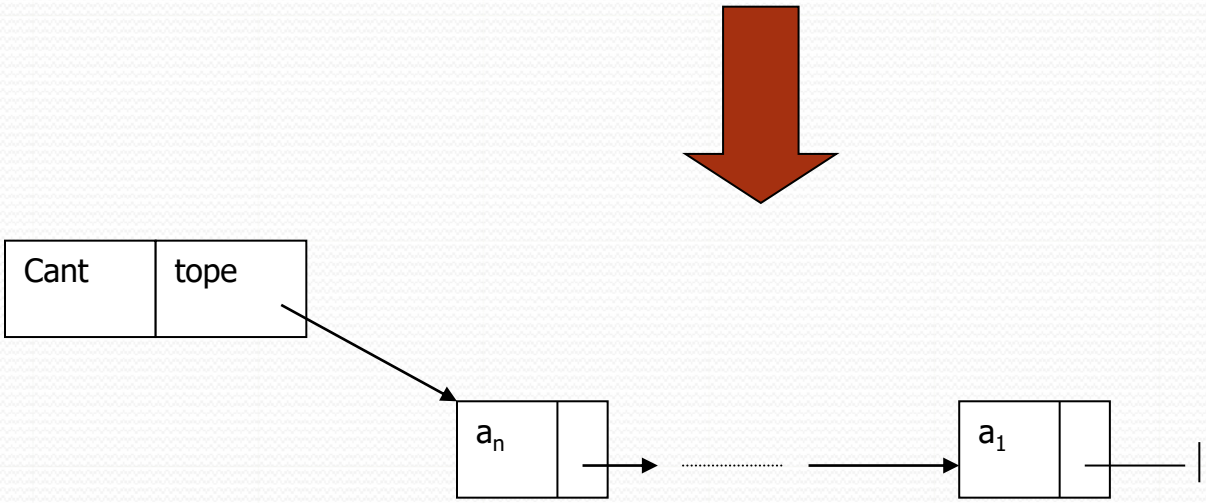


T.A.D. PILA

Representación (2)

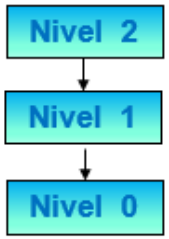
Representación encadenada:

P = (a1, a2, ..., an)



T.A.D. PILA –

Construcción de operaciones abstractas (1)



REPRESENTACIÓN SECUENCIAL

```
class pila
{   int *items;
   int tope;
   int cant;
   public:
   pila(int xcant=0):
   cant(xcant)
   {  tope=-1;
      items=new int[cant];
    }
   int vacia(void)
   {    return (tope==-1);
    }
   int insertar(int x)
  {
   if (tope<cant-1)
     { items[++tope]=x;
       return (x);
     }
     else return (0);
   }
```

```
int suprimir(void)
 {    int x;
   if (vacia())
   {     printf("%s","Pila vacia");
         return(0);
   }
 else
   {  x=items[tope--];
      return(x);
    }
 }
     void mostrar(void)
 { int i;
   if (!vacia())
   {  for (i=tope; i>=0; i--)
      {     cout<<items[i]<<endl;
      }
   }
 }
};
```

T.A.D. PILA

Construcción de operaciones abstractas (1)

```
class Stack:
def ____init_____(self): self.items = []

def isEmpty(self):
return self.items == []

def push(self, item): self.items.append(item)

def pop(self):
return self.items.pop()

def size(self):
return len(self.items)
```

T.A.D. PILA

Construcción de operaciones abstractas (2)

REPRESENTACIÓN ENCADENADA

```
class celda
{
   int item;
   celda *sig;
   public:
   int obteneritem(void)
   {
     return(item);
   }
   void cargaritem(int xitem)
   {
     item=xitem;
   }
   void cargarsig(celda* xtope)
   {
     sig=xtope;
   }
   celda* obtenersig(void)
   {
     return(sig);
   }
};
```

T.A.D. PILA

Construcción de operaciones abstractas (3)

REPRESENTACIÓN ENCADENADA

```
class pila
{
   int cant;
   celda *tope;
   public:
   pila(celda* xtope=NULL,int xcant=0):
   tope(xtope),cant(xcant)
   { }
   int vacia(void)
   {
   return (cant==0);
    }
   int insertar(int x)
  {
   celda *ps1;
   ps1=new(celda);
   ps1->cargaritem(x);
   ps1->cargarsig(tope);
   tope=ps1;
   cant++;
   return(ps1->obteneritem());
   }
```

```
int suprimir(void)
  { celda *aux;
  int x;
  if (vacia())
    {    printf("%s","Pila vacia");
         return(0);
    }
  else
  {  aux=tope;
     x=tope->obteneritem();
     tope=tope->obtenersig();
     cant--;
     free(aux);
     return(x);
   }
  }
```

Complejidad de Algoritmos

Análisis Amortizado

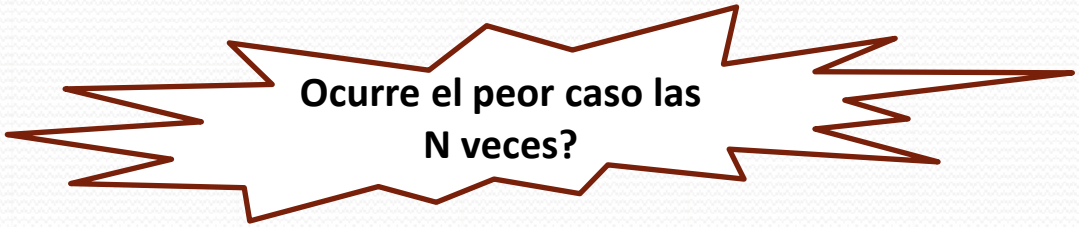
El análisis amortizado estudia el tiempo requerido para ejecutar una secuencia de operaciones sobre una estructura de datos.

Amortizar:
Recuperar o compensar los fondos in vertidos en alguna empresa. (RAE)

En el análisis normal en el peor caso, ejecutar N operaciones sobre una estructura de datos de n elementos lleva tiempo en O(f (n)), donde f (n) es el tiempo en el peor caso de la operación.

Complejidad de Algoritmos

Análisis Amortizado



- Las técnicas de análisis amortizado procuran obtener una cota menor para la secuencia de operaciones.

- Los resultados del análisis amortizado sirven para optimizar el diseño de la estructuras de datos, produciendo entonces estructuras de datos avanzadas

Cota ajustada

Notación Theta

$$\Theta(f)=\{t: N\rightarrow R^{+} / \exists c,d\in R^{+}, \exists n0\in N, \forall\, n\geq n0 : c\cdot f(n) \leq t(n) \leq d\cdot f(n)\}$$

$$\Theta\,(f(n)) = O(f(n)) \cap \Omega\,(f(n))$$