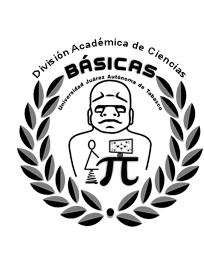
**UNIVERSIDAD JUÁREZ AUTONOMA DE TABASCO**

**DIVISIÓN ACADÉMICA DE CIENCIAS BÁSICAS**

PROGRAMA EDUCATIVO

**LIC. CIENCIAS COMPUTACIONALES**

PROFESOR

**DR. ABDIEL EMILIO CACERES GONZALEZ**

EXPERIENCIA EDUCATIVA

**ANALIZIS DE ALGORITMOS**

TRABAJO

**TAREA 5**

ESTUDIANTE

**RODRIGUEZ TORRES KEVIN NICK**

**CARDENAS, TAB. 26 DE ABRIL DEL 2021**

“EJERCICIOS”

Explica mediante un algoritmo como puedes implementar dos pilas en un mismo

arreglo, sin que las pilas se sobrepongan. Las pilas deben considerarse llenas cuando

la cantidad de elementos almacenados en ambas pilas sea igual al tamaño del arreglo. Cada pila puede almacenar elementos mientras haya algún espacio en el arreglo.

Partiendo del concepto de pila con sus métodos push, pop y view;

Tenemos dos pilas con una misma lista donde se guardarán los valores ingresados en el método push y serán extraídos en el método pop.

Array lista = [None]\*n declaración de la lista siendo n el tamaño de la lista

Stack1():

En esta sección del constructor inicializamos los valores para la pila 1 la cual usara desde la base de la lista hasta una cantidad n de la misma.

Constructor():

This.top = 0

This.lower = 0

Global lista

Push (value):

Si !Full():

Método push el cual ingresa un elemento a la pila el cual lo guarda en la lista.

This.lista[This.top] = value

This.top = This.top + 1

Pop(Lista):

Si !Empty():

Método pop el cual saca un elemento de la pila el cual lo elimina de la lista.

This.lista[This.Top – 1] = None

This.top = This.top – 1

View(Lista):

Para elemento de range (This.lower hasta This.top):

Imprimir elemento

Imprime la pila.

Empty():

Retorna true o false si la pila está vacía.

Return This.top == this.lower

Full():

Para elementos de This.lista:

If elementos == None:

Return Flase

Retorna true o false si la pila está Llena.

Return True

Stack2():

En esta sección del constructor inicializamos los valores para la pila 1 la cual usara desde la base de la lista hasta una cantidad n de la misma.

Constructor():

Global lista

This.top = lista.lengh

This.lower = lista.lengh

Push (value):

Si !Full():

Método push el cual ingresa un elemento a la pila el cual lo guarda en la lista.

This.lista[This.top] = value

This.top = This.top - 1

Pop(Lista):

Si !Empty():

Método pop el cual saca un elemento de la pila el cual lo elimina de la lista.

This.lista[This.Top + 1] = None

This.top = This.top + 1

View(Lista):

Para elemento de range (This.lower hasta This.top):

Imprimir elemento

Imprime la pila.

Empty():

Return This.top == This.lower

Retorna true o false si la pila está vacía.

Full():

Para elementos de This.lista:

If elementos == None:

Return Flase

Retorna true o false si la pila está Llena.

Return True

De esta forma podemos usar dos pilas distintas con una sola lista nótese que emos- considerado a la lista para ingresar los elementos en el inicio y fin de la misma así no se traslaparan los datos ni abra problemas de rendimiento de la lista.

Explica mediante un algoritmo como puedes implementar una pila utilizando dos

colas. Analiza el tiempo de ejecución de las operaciones Push y Pop de esta nueva

implementación.

Stack():

Constructor(sz):

This.Queue1 = [None]\*sz

This.Queue2 = [None]\*sz

Push(value):

Si !This.Queue1.Full():

This.Queue1.push(value)

La idea de este método es usar la segunda cola para almacenar todos los elementos menos el ultimo el cual se eliminara de la pila nótese que cala cola tiene una estructura FIFO así que sus métodos son contrarios a la pila. El cual es de tipo LIFO.

Pop():

Value = This.Queue1.pop()

If !This.Queue1.Empty():

This.Queue2.push(value)

This.Pop()

This.Queue1 = This.Queue2

This.Queue2 = Empty()

View():

Imprimir This.Queue1

Tomando en cuenta que cada línea de código en 1 unidad de tiempo para el método push tenemos que consta de 2 intrusiones lineales por lo que su tiempo de ejecución es 2 unidades de tiempo, el método pop consta de 6 líneas sin embargo una de ellas es recursiva por lo que es tiempo de ejecución es el cuadrado de la cantidad de elementos en la lista por lo que su tiempo de ejecución es siendo c la cantidad de elementos en la cola.

Explica mediante algoritmos, como se puede implementar una pila utilizando una

lista simplemente ligada. Debes escribir los algoritmos para las operaciones Pop y

Push de la pila en esta nueva implementación.

Teniendo en cuenta el concepto de lista simplemente ligadas el cual tenemos un conjunto de nodos que están enlazados solo con el nodo siguiente de tal forma que si queremos recorrer la colección lo haremos del primero hasta el último, pero no podremos regresar.

Stack():

Constructor(sz):

This.Top=0

This.Lista= [None]\*sz

Push(value):

SI !This.Lista.Full():

This.Lista.append(value)

This.Top = This.Top + 1

Pop():

Si !This.Lista.Empty():

This.Lista[This.Top] = None

This.Top = This.Top - 1

View():

Imprimir Thsi.Lista

Dibuja el árbol binario que tiene raiz en el nodo con índice 6 que está representado

por los siguientes atributos:

|  |  |  |  |
| --- | --- | --- | --- |
| ´ındice | *key* | *left* | *right* |
| 1 | 12 | 7 | 3 |
| 2 | 15 | 8 | NIL |
| 3 | 4 | 10 | NIL |
| 4 | 10 | 5 | 9 |
| 5 | 2 | NIL | NIL |
| 6 | 18 | 1 | 4 |
| 7 | 7 | NIL | NIL |
| 8 | 14 | 6 | 2 |
| 9 | 21 | NIL | NIL |
| 10 | 5 | NIL | NIL |