Linear Data Structures

(Estructuras de datos lineales)

# Que es un ADT?

Abstract Data Type 🡪 Un conjunto de objetos que de igual manera tienen un conjunto de operaciones.

Es una abstracción (modelo) matemática: Por lo tanto, tiene una definición y operaciones.

## Tipos de datos abstractos

Listas, colas, arboles, grafos, conjuntos, pila, array

Operaciones: insertar, reordenar, extraer, quitar, editar…

En POO, se utilizan por medio de clases, pues me permite ocultar los detalles de la implementación.

Estas operaciones se realizan mediante llamada a métodos

En POO, un ADT es una instancia

No existe un estándar para las operaciones en un ADT

No se exponen los detalles de la implementación (usando un método auxiliar). Además, se deben usar setters y getters.

## Tipos de datos (data type)

Todos los tipos de datos tienen operaciones

entero (int), double, String, Array,bool [tipos de datos])

Son todas representaciones inexactas.

Arreglos

Son un tipo de dato abstracto.

Los arreglos tienen un tamaño limitado.

Ejemplo:

Int[] a; o Int a[];

Int[] a = new int[10];

Int [] b = {5,4,2,1} 🡪 se vuelve implícita la cantidad de posiciones.

Primer índice = 0

Ultimo índice = largo - 1

Tiene acceso indexado

Para acceder a una posición 🡪 a[indx]

Para asignar un valor a una posición 🡪 a[indx]= 3 o x = a[3]

(nota: b2logic)

# Limitaciones

Para cambiar el tamaño debe eliminarse el arreglo anterior y copiarlo en el arreglo con el nuevo tamaño.

Se ordena secuencialmente en memoria, por lo que si quiero poner un numero en medio de otros es necesario correr los demás valores.

# Matriz

Es un arreglo bidimensional (un arreglo de arreglos)

Int [][] nums = new int [5][4] (en orden: filas y columas ) 🡪 matriz para enteros de 5x4

# Listas enlazadas

Es una estructura de datos lineal donde cada elemento (nodo) es un objeto separado y no están necesariamente secuencialmente en memoria.

Es una estructura de datos dinámica 🡪 la cantidad de nodos no es fija, es decir, se puede agregar y quitar nodos (independientemente de la cantidad o la posición)

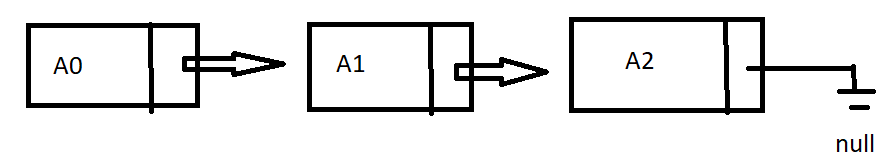
Funciona para gestionar datos cuando van a generar un número desconocido de objetos.

Cada nodo tiene dos datos (dos campos):

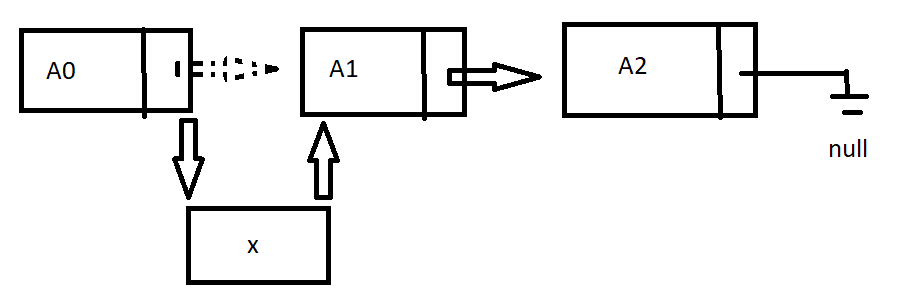
El dato que almaceno (objetos, enteros, etc)

La referencia al siguiente nodo (a quien apunto)

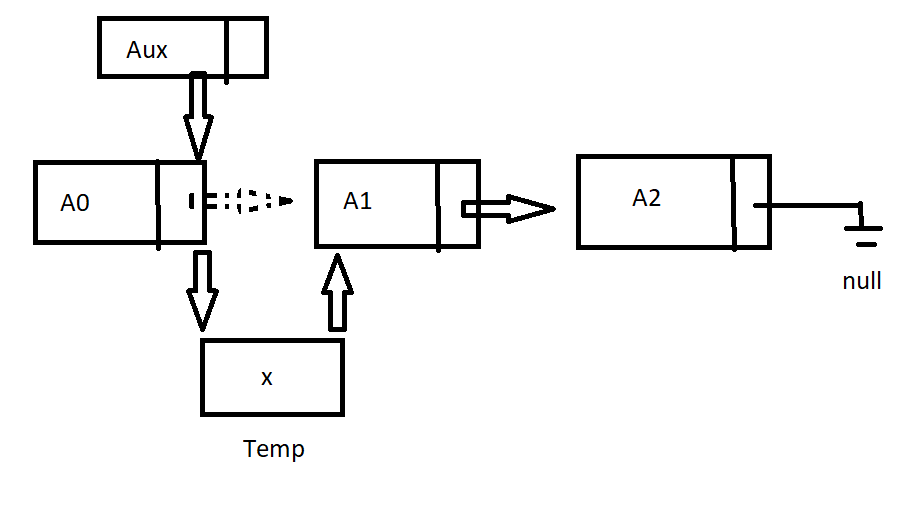
Cuando se encuentra un nodo que apunta a null significa que el ultimo



Insertar un elemento en una dada posición:



* Se crea un nodo sin valor y que apunta a null (temp)
* Se crea un nodo sin valor que apunta a la posición donde quiero agregar el nodo (aux)
* Temp apunta a su siguiente deseado a partir de aux
* Y el anterior se para a apuntar al nuevo nodo



Eliminar un elemento en una dada posición:

* Se crea un nodo aux en la primera posición y se pregunta : el síguente es el que quiero eliminar? Y lo recorre hasta encontrarlo y apunta al anterior.
* Procede a cambiar la referencia del siguente al siguente.

Recolector de basura: Si hay un espacio en memoria que no referencio se elimina

Solo el head de la lista no se borra por no estar referenciada.

# Código de nodo

(foto)

class Node

# Codigo linkled list

## Agregar al inicio

analizar

## Eliminar al inicio

Que el primero pase a ser el segundo

(Averiguar si llamo un atributo de un obejto de mi clase pero es creado de otra clase)

## Imprimir datos

public void retornarValores(){

Node current = this.head;

while (current != null){

System.out.println(current.getDato());

current = current.getNext();

}

}

## Retornar un dato especifico

codigo

## Eliminar un dato especifico

public Node retornarUnValor(object x){

Node current = this.head;

if (current.getData.equals(x)){

this.head = current.getNext();

return current;

}

while (current.getNext() != null){

if (current.getNext.getData() == x){

current.setNext(current.getNext.getNext)

return current.getNext();

}

}

else{

currernt = current.getNext();

}

return null;

}

Otro codigo:

# Double-ended list

Tener un head y un tail.

Codigo:

# Lista doblemente enlazada (buscar imágenes)

Cada nodo tiene una referencia a su nodo anterior y posterior.

Es una variación de la lista simple.

Ventaja es que se puede recorrer hacia atrás

Hay que tener un apuntador first, y se puede tener un last

Son mas complicados de controlar, pues tienen que mantener referencia del anterioir

## Insertar el inicio

1. Anterior es nuevo nodo (old node)
2. Sigueinte es old node (nuevo node)
3. New node es first (lista)

## Insertar en x

Current en el anterior al que quiero agregar

1. Anterior es nuevo nodo (old node)
2. Sigueinte es old node (nuevo node)
3. (buscar imagen)

## Eliminar en X

Current en el que elimino

1. Al siguiente lo apunto a al anterior
2. Al anterior lo apunto al siguiente

# Lista circular

El ultimo apunta al primero, igualmente podría ser doblemente enlazada.

No tenemos final.

El apuntador va al último si quiero eliminar al inicio, y en el primero si elimino mucho al final.

Es bueno cuando quiero acceder tanto al inicio y al final, y si la lista esta ordenada.

# Pilas (stacks)

Es una estructura de datos que solo permite el acceso al último ítem que se agrega de ultimo, y no se puede acceder al segundo si no se “trata” al primero.

LIFO = último que entra es el primero en salir.

Una pila para compiladores --> procesamiento de instrucciones, así las interpreta

Tienen 3 operaciones (siempre en el tope, el último en ser agregado):

* Agregar 🡪 Push
* Consultar 🡪 Peek
* Extraer 🡪 Pop

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | Push(15) | Push(7) | Peek(92) | Pop(7) | Pop(92) | Peek(); |  |  |
| 49 | 15 | 7 | 7 | 7 | 7 | 7 |  |  |
| 27 | 49 | 15 | 15 | 15 | 15 | 15 |  |  |
| 14 | 27 | 49 | 49 | 49 | 49 | 49 |  |  |
| 3 | 14 | 27 | 27 | 27 | 27 | 27 |  |  |
| 92 | 3 | 14 | 14 | 14 | 14 | 14 |  |  |
| 64 | 92 | 3 | 3 | 3 | 3 | 3 |  |  |
|  | 64 | 92 | 92 | 92 | 92 | 92 |  |  |
|  |  | 64 | 64 | 64 | 64 | 64 |  |  |
|  |  |  | Error | Error | Error | Retorna 7 |  |  |

\*Peek(); no lleva parametros de entrada

\*Pop(); no lleva parámetros de entrada

Una pila es útil en stack frame (llamadas a metodos). Pues solo trabajo el método que estoy llamando o bien el método que llame ese método.

## Pilas utilizando arrays

Top es el indice

maxSize el tamaño maximo del array

this.stackArray[++top]

(++x 🡪 incremente y luego use)

(x-- 🡪 Use y luego reste)

Para elementos delimitados, conozco el tamaño

## Pilas utilizando ListasEnlazada

Pop 🡪 insertarAlInicio();

Push🡪 eliminarAlInicio();

Peek🡪getHead();

O bien con los últimos y tener un apuntador al primero

Crecimiento dinámico, desconozco el tamaño

# Colas (Queues)

Como la fila del comedor.

Respeta quien tiene mas tiempo de estar montado en esa estructura.

FIFO = First in First out

Al interno un sistema operativo utiliza colas 🡪 colas de impresión FE

También tiene 3 operaciones

Push 🡪 Enqueue (agrego al final)

Pop 🡪 Dequeue (saco al primero, es igual que pila)

Peek 🡪 Peek :3 (veo el primero)

Al usar Array:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  |  | Enqueue(x); | Enqueue(y); | Enqueue(z); | Enqueue(j); | Enqueue(f); |
| 9 | - | - | y | y | y | y |
| 8 | - | x | x | x | x | x |
| 7 | 79 | 79 | 79 | 79 | 79 | 79 |
| 6 | 32 | 32 | 32 | 32 | 32 | 32 |
| 5 | 6 | 6 | 6 | 6 | 6 | 6 |
| 4 | 80 | 80 | 80 | 80 | 80 | 80 |
| 3 | 12 | 12 | 12 | 12 | 12 | 12 |
| 2 | - | - | - | - | - | f |
| 1 | - | - | - | - | j | j |
| 0 | - | - | - | z | z | z |

Si se intenta agregar otro da un error o se pasa la solicitud a una lista de espera.

“¿Esta disponible la siguente?” (asi como 2 🡪3 entonces 9 🡪0)

Otra solución es mover toda la estructura para abajo, como una fila.

## Cola con arreglo

maxSize

Object[] queueArray

Int front

Int rear

Int nIteams

## Cola con listaEnlazada

Peek get head

# Priority Queue

Funciona igual que una cola.

Pero al encolar se ponen en una llave (un sistema de prioridad)

En el frente del la pila tengo al elemnteo con mayor prioridad.

Por orden de tamaño por ejempo

El dequeue es normal