Arreglos

Conceptos básicos

Verónica E. Arriola-Rios

Facultad de Ciencias, UNAM

3 de junio de 2021

Definición

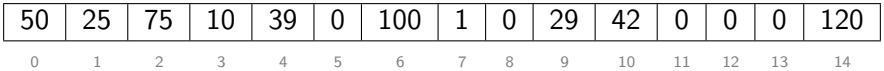
Temas

Arreglos como tipo abstracto de datos

Definición (Arreglo)

Un TAD *arreglo* es una estructura de datos que contiene un conjunto de **elementos** *del mismo tipo*, un conjunto de **índices** y un conjunto de operaciones que se utilizan para definir, manipular y abstraer estos elementos de datos.

Sengupta y Korobkin 1994



- **NOTA:** No todos los autores consideran el tipo de dato abstracto arreglo. En estos casos, sólo se considera a los arreglos como estructuras básicas provistas por la arquitectura de la computadora.
- Para distinguir entre el TAD y la implementación, también se puede utilizar el nombre *Vector* para el TAD y *arreglo* para la implementación.

Arreglo multidimensional

Definición (Arreglo)

Un arreglo de

1. dimensión n,
2. de elementos de tipo X y
3. de tamaño $T = t_1 \times t_2 \times ... \times t_n$, donde t_j es el tamaño del arreglo en la j-ésima dimensión,

es un conjunto de T elementos de tipo X, en el que cada uno de ellos es identificado **unívocamente** por un **vector coordenado de n índices** $(i_1, i_2, ..., i_n)$, con $0 \leqslant i_j < t_j$.

	0	1	2	3	4
0	50	25	75	10	39
1	0	100	1	0	29
2	42	0	0	0	120

Temas

Operaciones para un arreglo unidimensional

Nota: La definición de las operaciones siguientes utilizan un lenguaje que asume que no hay cambios de estado cada vez que una función es llamada, sino que las funciones que modifican al arreglo devuelven un arreglo nuevo.

Operaciones básicas:

- construir: $\emptyset \rightarrow$ Arreglo
- construir: $\{$Datos$\} \rightarrow$ Arreglo
- destruir: Arreglo $\rightarrow \emptyset$
- almacenar: Arreglo, Int, Elemento $\rightarrow$ Arreglo
- actualizar: Arreglo, Int, Elemento $\rightarrow$ Arreglo
- recuperar: Arreglo, Int $\rightarrow$ Elemento

Manipulación avanzada:

- insertarAntes: Arreglo, Elemento, Elemento $\rightarrow$ Arreglo
- insertarDespués: Arreglo, Elemento, Elemento $\rightarrow$ Arreglo
- buscar: Arreglo, Elemento $\rightarrow$ Int
- borrar: Arreglo, Elemento $\rightarrow$ Arreglo
- ordenar: Arreglo $\rightarrow$ Arreglo
- imprimir: Arreglo $\rightarrow$ Cadena

Temas

Restricciones

Las operaciones almacenar, insertarAntes, insertarDespués, borrar, actualizar, buscar, ordenar **son tales que para todo** [a:Arreglo, e, e_i:Elemento, i:$\mathbb{N}$] **:**

1. *Al **almacenar/actualizar** el elemento e en la posición* i *de* a*, e deberá ser recuperable desde esa misma posición:*

$$\text{Si } \mathrm{recuperar}(a, i) = \emptyset \Rightarrow \mathrm{recuperar}(\mathrm{almacenar}(a, i, e), i) = e$$
$$\text{Si } \mathrm{recuperar}(a, i) \neq \emptyset \Rightarrow \mathrm{recuperar}(\mathrm{actualizar}(a, i, e), i) = e$$

Restricciones: Buscar y borrar

1. *La función* ***buscar*** *devuelve el índice donde fue almacenado el elemento* e

$$\text{Sea } e_i = e \Rightarrow \text{buscar}(a, e) = i$$

2. *Después de* ***borrar*** *el elemento* e *del arreglo* a*, ya no debe aparecer en* a.[1].

$$\text{buscar}(\text{borrar}(a, e), e) = -1$$

3. *El índice de cada elemento* e_j *a la derecha del elemento borrado* e_i *debe ser disminuído en* 1*, mientras que los demás no son modificados.*

$$\text{buscar}(\text{borrar}(a, e_i), e_j) = \text{buscar}(a, e_j) - 1 \qquad \text{para } j > i$$
$$\text{buscar}(\text{borrar}(a, e_i), e_j) = \text{buscar}(a, e_j) \qquad \text{para } j <= i$$

[1] Recuérdese que, en este caso, se dijo que a cada elemento e corresponde un sólo índice.

Restricciones: Insertar, buscar y ordenar

1. *Al **insertar** el elemento e antes de e_i, e tomará la posición de e_i.*

$$\text{buscar}(\text{insertarAntes}(a, e_i, e), e) = \text{buscar}(a, e_i)$$

2. *Al **insertar** el elemento e antes de e_i, los índices de todos los elementos, a partir de e_i serán incrementados en uno.*

$$\text{Sea } j > i \Rightarrow \text{buscar}(\text{insertarAntes}(a, e_i, e), e_j) = \text{buscar}(a, e_j) + 1$$

3. *Después de **ordenar** el arreglo, todo elemento a la izquierda deberá ser menor que cualquier elemento a su derecha.*

$$\forall i, j \mid 0 \leqslant i < j < T(a), \text{recuperar}(\text{ordenar}(a), i) < \text{recuperar}(\text{ordenar}(a), j)$$

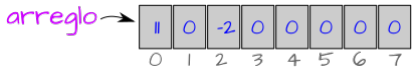
Ejercicio: dar las restricciones para la operación `insertarDespués`.

Arreglos en Java

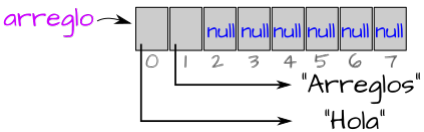
Temas

Representación

Tipos primitivos:

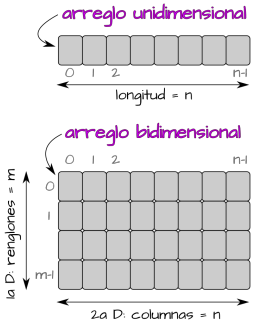


Objetos y referencias:

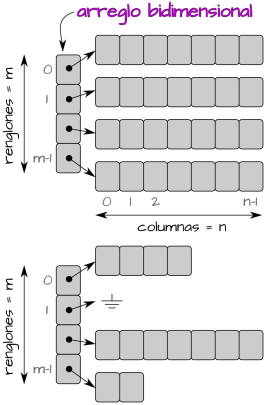


Representación 2

Conceptualmente:



En memoria: **Vectores de Iliffe**



Representación 3

Conceptualmente:

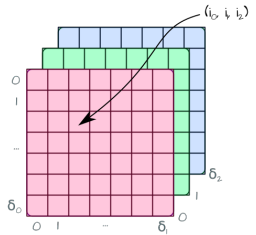
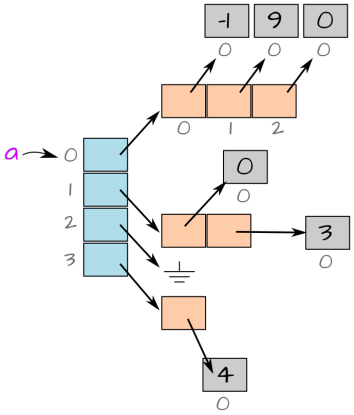


Figura: Arreglo 3D

En memoria: **Vectores de Iliffe**



Temas

Implementación de las operaciones del TDA

Ejemplos con un arreglo de cadenas `String` en Java.

- `construir: new String[n];`
- `construir: String[] s = {"Lápiz", "Pluma", "Plumón"};`
- `destruir: s = null; // Remover referencias.`
- `almacenar: s[i] = "Lapicero";`
- `recuperar: temp = s[i];`

Arreglos de 1D en Java

Se declaran e instancian con:

```
1  String array[];
2  array = new String[2];
```

Ojo: Sólo se reservan espacios para referencias a cadenas. Ahora hay que crear los objetos:

```
1  array[0] = "Hola";
2  array[1] = new String("arreglos");
```

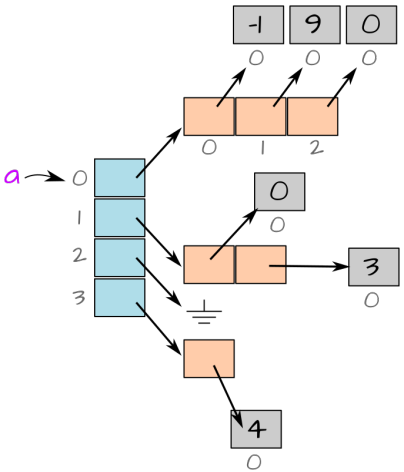
Creación abreviada

```java
public class PrintCadenas{
  public static void main(String args[]){
    String lista[] = {"Hola", "Arreglos"};

    for(int i = 0; i < lista.length; i++){
        System.out.println(lista[i]);
    }
  }
}
```

```
>> java PrintCadenas
Hola
Arreglos
```

Ejemplo de arreglo irregular




```java
int[][][] a = new int[4][][];

// Renglón 0
a[0] = new int[][]{{-1}, {9}, {0}};

// Renglón 1
int[][] a1 = new int[2][1];
a[1] = a1;
a1[0][0] = 0;
a1[1][0] = 3;

// Renglón 3
a[3] = new int[1][];
a[3][0] = new int[1];
a[3][0][0] = 4;
```

```java
int[][][] a = {{{-1},{9},{0}},
               {{0},{3}},null,{{4}}};
```

Ejemplo de arreglo regular

	0	1	2	3	4	5
0	0	1	2	3	4	5
1	-1	0	1	2	3	4
2	-2	-1	0	1	2	3
2	-3	-2	-1	0	1	2

```
1  public static int[][] porZona(int rens, int cols) {
2      if(rens <= 0 || cols <= 0) throw new IllegalArgumentException();
3      int[][] res = new int[rens][cols];
4      for(int i = 0; i < rens; i++) {
5          for(int j = 0; j < cols; j++) {
6              if(j < i) res[i][j] = -(j - i);
7              else if (i == j) res[i][j] = 0;
8              else res[i][j] = (j - i);
9          }
10     }
11     return res;
12 }
```

Arreglos empacados

Arreglos de bits

- Si desea un arreglo de valores binarios, como los `boolean` se desperdicia mucho espacio, porque incluso un `True` o `False` pueden requerir un byte entero[2], así que

$$\texttt{False} = 00000000$$
$$\texttt{True} = 00000001$$

  Esto debido a que la unidad de memoria que maneja la máquina más eficientemente es **una palabra**.
- Un arreglo de este tipo (`boolean[]`) desperdiciaría aún más espacio.
- Es común entonces utilizar los **bits** dentro de un tipo entero como `int` para almacenar los datos binarios y a esto se le llama *arreglos empacados*.

[2] La cantidad de memoria exacta puede variar desde 1 bit hasta 1 int (4 bytes) dependiendo de la implementación de la máquina virtual.

Empacando y desempacando

- *Representación.* Un entero, cada bit contiene un valor almacenado. La longitud del arreglo es el tamaño en bits de la representación del tipo primitivo (e.g. 32 bits).

$$\texttt{arr} = 10011010101110111001101010111011$$

- *Leer.* Para leer el valor almacenado en la posición `i` es necesario utilizar operaciones sobre bits.

$$\texttt{arr} = 10011010101110111001\mathbf{1}01010111011$$

`lee((byte)12)`

```
1  public boolean lee(byte pos) {
2    int máscara = 1 << pos;
3    return (arr & máscara) != 0;
4  }
```

- *Escribir.* Escribir el valor requiere el uso de máscaras, enteros con los bits adecuados para colocar el valor binario en la posición requerida.

$$\texttt{arr} = 10011010101110111000101010111011$$

`escribe(false, (byte)12)`

```
1  public void escribe(boolean val, byte pos) {
2      int pack = 1 << pos;
3      if (val) {
4          arr |= pack;
5      } else {
6          pack = Integer.MAX_VALUE - pack;
7          arr &= pack;
8      }
9  }
```

Bibliografía

Bibliografía I


Sengupta, Saumyendra y Carl Philip Korobkin (1994). «C++ Object-Oriented Data Structures». En: pág. 51.

Licencia