Arreglos

Conceptos básicos

Verónica E. Arriola-Rios

Facultad de Ciencias, UNAM

8 de agosto de 2024

Definición

1 Definición

2 Arreglos en C#

3 Arreglos en .NET

4 Arreglos empacados

5 Bibliografía

Temas

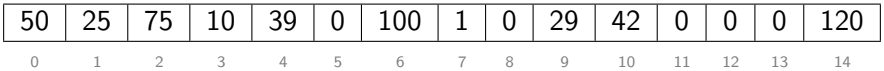
1. Definición
   - Datos
   - Operaciones
   - Ejemplos de restricciones para las operaciones

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

1. Definición
   - Datos
   - Operaciones
   - Ejemplos de restricciones para las operaciones

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

1. Definición
   - Datos
   - Operaciones
   - Ejemplos de restricciones para las operaciones

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:*

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

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 C#

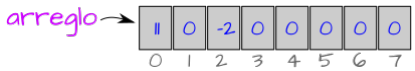
1. Definición
2. Arreglos en C#
3. Arreglos en .NET
4. Arreglos empacados
5. Bibliografía

Temas

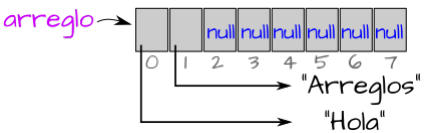
2 Arreglos en C#
- Representación
- Código

Representación

Tipos valor:

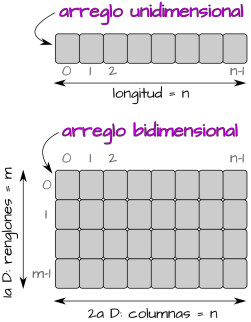


Tipos referencia:

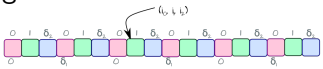


Representación 2

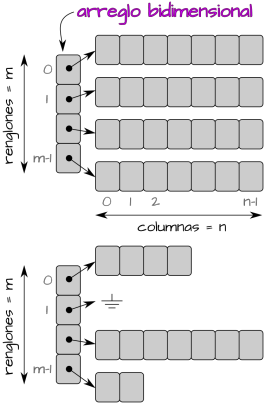
Conceptualmente:



Arreglos *multidimensionales*:



Arreglos *dentados (Jagged array)*:
Vectores de Iliffe



Representación 3

Conceptualmente:

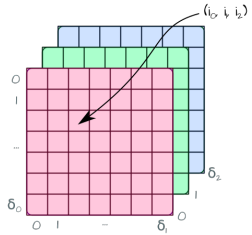
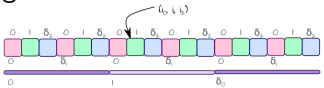


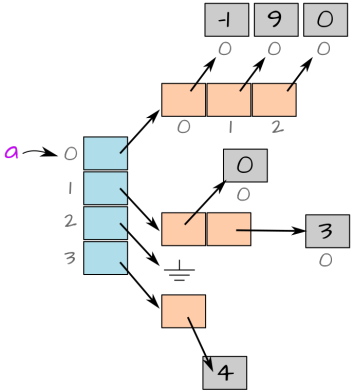
Figura: Arreglo 3D

Arreglos *multidimensionales*:



Arreglos dentados *(Jagged array)*:

Vectores de Iliffe



Temas

2 Arreglos en C#
- Representación
- Código

Implementación de las operaciones del TDA

Ejemplos con un arreglo de cadenas `string` en C#.

- `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 C#

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");  // string y String son válidos.
```

Creación abreviada

```
1  public class PrintCadenas
2  {
3    public static void Main()
4    {
5      String lista[] = {"Hola", "Arreglos"};
6
7      for(int i = 0; i < lista.Length; i++)
8      {
9          Console.WriteLine(lista[i]);
10     }
11   }
12 }
```

```
1 >> dotnet run
2 Hola
3 Arreglos
```

Arreglos y Genéricos (Creación)

```
1  namespace Arreglo
2  {
3      class Arreglo<T>
4      {
5          private T[] _arr;
6          public Arreglo(int tam)
7          {
8              _arr = new T[tam];
9          }
10         public T this[int i]
11         {
12             get { return _arr[i]; }
13             set { _arr[i] = value; }
14         }
15     }
16 }
```

Uso

```
1  namespace Arreglo
2  {
3      class Programa
4      {
5          static void Main(string[] args)
6          {
7              Arreglo<int> ag = new Arreglo<int>(5);
8              ag[0] = 10;
9              Console.WriteLine("¡Ahhh!" + ag[0]);
10         }
11     }
12 }
```

Arreglos y Genéricos (Iteración)

```
1  public class EjemplosArreglos
2  {
3      private static void ImprimeArreglo<T>(T[] a)
4      {
5          for (int i = 0; i < a.Length; i++)
6          {
7              Console.WriteLine($"{i}: {a[i]}");
8          }
9      }
10
11     private static void ImprimeArreglo2<T>(T[] a)
12     {
13         foreach(var e in a)
14         {
15             Console.WriteLine(e);
16         }
17     }
18 }
```

Arreglos y Genéricos (Uso iteración)

```
1  public class EjemplosArreglos
2  {
3      public static void Main()
4      {
5          char?[] cadenas = new char?[10];
6          cadenas[3] = 'c';
7          cadenas[2] = 'b';
8          cadenas[5] = 'c';
9          cadenas[8] = '¿';
10
11         imprimeArreglo(cadenas);
12         imprimeArreglo2(cadenas);
13     }
14 }
```

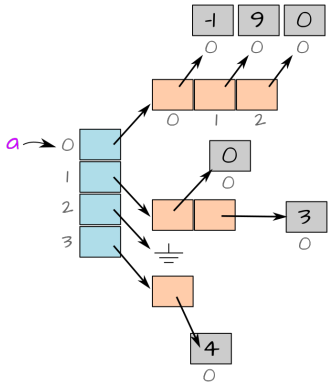
```
0:
1:
2: b
3: c
4:
5: c
6:
7:
8: ¿
9:


b
c

c


¿
```

Ejemplo de arreglo irregular




```
1  int[][][] a = {{{-1},{9},{0}},
2                 {{0},{3}},null,{{4}}};
```

```
1  int[][][] a = new int[4][][];
2
3  // Renglón 0
4  a[0] = new int[][]{new int[]{-1}, new
       ->int[]{9}, new int[]{0}};
5
6  // Renglón 1
7  int[][] a1 = new int[2][];
8  a1[0] = new int[1];
9  a1[1] = new int[1];
10 a[1] = a1;
11 a1[0][0] = 0;
12 a1[1][0] = 3;
13
14 // Renglón 3
15 a[3] = new int[1][];
16 a[3][0] = new int[1];
17 a[3][0][0] = 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
3	-3	-2	-1	0	1	2

```
1  public static int[,] PorZona(int rens, int cols)
2  {
3      if(rens <= 0 || cols <= 0) throw new ArgumentException();
4      int[,] res = new int[rens,cols];
5      for(int i = 0; i < rens; i++)
6      {
7          for(int j = 0; j < cols; j++)
8          {
9              if(j < i) res[i,j] = -(j - i);
10             else if (i == j) res[i,j] = 0;
11             else res[i,j] = (j - i);
12         }
13     }
14     return res;
15 }
```

Arreglos en .NET

1 Definición

2 Arreglos en C#

3 Arreglos en .NET

4 Arreglos empacados

5 Bibliografía

Clase Array

- La clase base `Array` se encuentra definida en la plataforma `.NET` y cada lenguaje extiende esta clase.
- Por lo tanto, cuenta con métodos generales para crear y manipular arreglos de varias dimensiones.
- Admite un máximo de:
  - 32 dimensiones,
  - índice máximo de 0X7FEFFFFF en cada dimensión (0X7FFFFFC7 si cada dato mide un `byte`)
  - 4 billones de elementos
  - 2 gigabytes (GB) de capacidad, esta restricción se puede eliminar en ambientes de 64bits (*Array Class - Reference* s.f.).

Imprimir arreglo multidimensional I

```
1  public class EjemploArreglos
2  {
3      private static void ImprimeArreglo(Array arr,
4                                         int[] índices,
5                                         int dimensión)
6      {
7          if (dimensión == arr.Rank - 1)
8          {
9              for(int i = 0; i < arr.GetLength(dimensión); i++)
10             {
11                 índices[dimensión] = i;
12                 Console.WriteLine($"{String.Join(",␣índices)}:␣{arr.GetValue
                       ->(índices)}");
13             }
14         }
15         else
16         {
17             for(int i = 0; i < arr.GetLength(dimensión); i++)
```

Imprimir arreglo multidimensional II

```
18              {
19                  índices[dimensión] = i;
20                  ImprimeArreglo(arr, índices, dimensión + 1);
21              }
22          }
23      }
24      public static void ImprimeArreglo(Array arr)
25      {
26          Console.WriteLine("Arreglo:␣");
27          Console.WriteLine($"␣␣{arr}␣con␣rango={arr.Rank}");
28          int[] espacios = new int[arr.Rank];
29          ImprimeArreglo(arr, espacios, 0);
30      }
31      public static void Main()
32      {
33          int[,,] ar = new int[3,4,5];
34          ar[2,3,4] = 1;
35          Console.WriteLine(typeof(int[,,]));
36          ImprimeArreglo(ar);
```

Imprimir arreglo multidimensional III

```
37          }
38  }
```

Arreglos empacados

1 Definición

2 Arreglos en C#

3 Arreglos en .NET

4 Arreglos empacados

5 Bibliografía

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.

$$\texttt{arr} = 10011010101110111001101010111011$$

`lee((byte)12)`

```
1  public bool Lee(byte pos)
2  {
3      int máscara = 1 << pos;
4      return (_arr & máscara) != 0;
5  }
```

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

$$arr = 10011010101110111000101010111011$$

escribe(false, (byte)12)

```
1  public void Escribe(bool val, byte pos)
2  {
3      int pack = 1 << pos;
4      if (val)
5      {
6          _arr |= pack;
7      }
8      else
9      {
10         pack = Int32.MaxValue - pack;
11         _arr &= pack;
12     }
13 }
```

Bibliografía

1 Definición

2 Arreglos en C#

3 Arreglos en .NET

4 Arreglos empacados

5 Bibliografía

Bibliografía I

- *Array Class - Reference* (s.f.). Microsoft. URL: https://learn.microsoft.com/en-us/dotnet/api/system.array?view=net-7.0.
- Sengupta, Saumyendra y Carl Philip Korobkin (1994). «C++ Object-Oriented Data Structures». En: pág. 51.

Licencia