

Programación II. Algoritmos y EDD II. 2024

AGENDA 8-4

TDA DICCIONARIO SIMPLE.

Especificación

Implementación estructura estática

TDA DICCIONARIO MULTIPLE

Especificación

Implementación estructura estática

EVALUACION DE COSTOS EN ALGORITMOS

Ejercitación

TDA Diccionarios

Diccionario. Se caracteriza porque cada valor ingresa a la estructura asociado a una clave, y estas claves existen siempre que tengan valor asociado y son únicas.

Analizaremos dos comportamientos diferentes de la estructura diccionario.

Primero: en donde cada clave puede tener asociado un único valor, estructura llamada **Diccionario Simple (dicc. Palabra, sinónimo)**,

Segundo: en donde cada clave tiene asociado un conjunto de valores, estructura que denominaremos **Diccionario Múltiple (dic Curso, alumnos)**.

TDA Diccionario Simple

Operaciones:

Agregar , dada una clave y un valor, agrega al diccionario el valor quedando asociado a la clave. Si ya existe la misma clave con otro valor se **sobreescribe** el valor, dejando el nuevo ingresado. Para poder agregar un par clave/valor la estructura debe estar inicializada.

TDA Diccionario Simple

Operaciones:

Agregar, dada una clave y un valor, agrega al diccionario el valor quedando asociado a la clave. Si ya existe la misma clave con otro valor se **sobreescribe** el valor, dejando el nuevo ingresado. Para poder agregar un par clave/valor la estructura debe estar inicializada.

Eliminar, dada una clave elimina el valor asociado a la clave, y por consiguiente la clave, ya que el diccionario no puede contener claves sin valores asociados. Si la clave no existe no se hace nada. Se supone que el diccionario esta inicializado.

Recuperar, dada una clave devuelve el valor asociado a la clave. La clave dada debe pertenecer al diccionario. Se supone que el diccionario esta inicializado.

TDA Diccionario Simple

Operaciones:

Agregar, dada una clave y un valor, agrega al diccionario el valor quedando asociado a la clave. Si ya existe la misma clave con otro valor se **sobreescribe** el valor, dejando el nuevo ingresado. Para poder agregar un par clave/valor la estructura debe estar inicializada.

Eliminar, dada una clave elimina el valor asociado a la clave, y por consiguiente la clave ya que el diccionario no puede contener claves sin valores asociados. Si la clave no existe no se hace nada. Se supone que el diccionario esta inicializado.

Recuperar, dada una clave devuelve el valor asociado a la clave. La clave dada debe pertenecer al diccionario. Se supone que el diccionario esta inicializado.

Claves, devuelve el conjunto de todas las claves definidas en el diccionario. Se supone que el diccionario esta inicializado.

InicializarDiccionario, permite inicializar la estructura del diccionario.

TDA Diccionario Simple. Interfaz

```
1. public interface DiccionarioSimpleTDA; {  
2.     void InicializarDiccionario();           // sin precondiciones  
3.     void Agregar(int clave, int valor);      // diccionario inicializado  
4.     void Eliminar(int clave);               // diccionario inicializado  
5.     int Recuperar(int clave);               // diccionario inicializado y clave existente  
6.     ConjuntoTDA Claves();                  // diccionario inicializado  
7. }
```

TDA Diccionario Simple. Interfaz

```
1.  public interface DiccionarioSimpleTDA; {  
2.      void InicializarDiccionario();           // sin precondiciones  
3.      void Agregar(int clave, int valor);       // diccionario inicializado  
4.      void Eliminar(int clave);                 // diccionario inicializado  
5.      int Recuperar(int clave);                 // diccionario inicializado y clave existente  
6.      ConjuntoTDA Claves();                     // diccionario inicializado  
7.  }
```

Ejemplo: pasaje de todos los valores de un diccionario simple *Dic* a una pila *Valores*.

```
1.  public void Pasaje (DiccionarioSimpleTDA Dic, PilaTDA Valores) {  
2.      ConjuntoTDA claves = new ConjuntoTDA();  
3.      claves.InicializarConjunto();  
4.      claves = Dic.Claves();  
5.      while (!claves.ConjuntoVacio()) {  
6.          int x = claves.Elegir();  
7.          Valores.Apilar(Dic.Recuperar(x));  
8.          claves.Sacar(x);  
9.      }  
10. }
```

TDA Diccionario Simple. Implementación

Implementación TDA de Diccionario Simple a partir de la utilización de arreglos.

- clase **Elemento**, que contendrá una clave entera y un valor entero. Los objetos de éste contendrán las entradas individuales del diccionario.
- el diccionario simple se representará entonces como un ***arreglo elementos de objetos de tipo Elemento***.

Se definirá un **método privado** para uso exclusivo de la implementación: **Clave2Indice** que, dada una clave, devuelve el índice correspondiente en elementos.

Se utilizará la **variable entera cant** para indicar la cantidad de claves que hay en elementos.

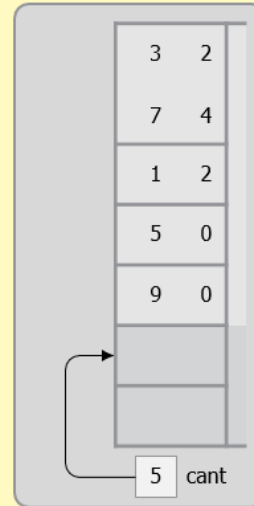
Cuando se elimina una clave (o un valor), se reemplaza el elemento eliminado con el de la última posición.

TDA Diccionario Simple. Implementación

Agregado (clave que no existe)

La clave no existe; se crea una nueva entrada con clave 9 y valor 0 y se incrementa *cant*

Agregar (9, 0)

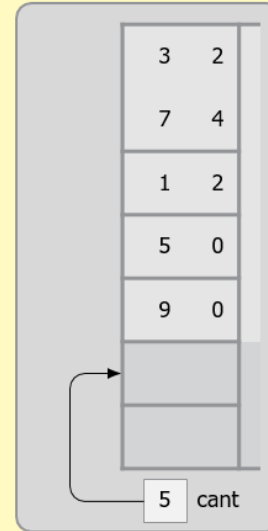


TDA Diccionario Simple. Implementación

Agregado (clave que existe)

La clave existe; el nuevo valor (7)
sobrescribe al antiguo valor (0)

Agregar(5, 7)

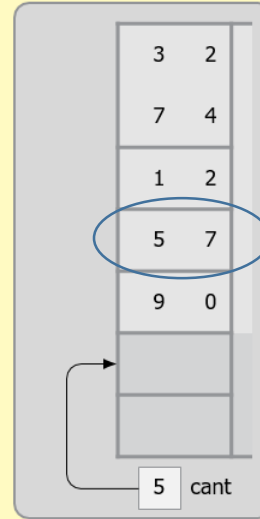


TDA Diccionario Simple. Implementación

Agregado (clave que existe)

La clave existe; el nuevo valor (7)
sobrescribe al antiguo valor (0)

Agregar(5, 7)

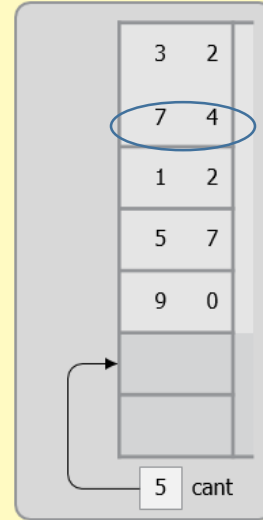


TDA Diccionario Simple. Implementación

Eliminar

La entrada con clave 7 se reemplaza
con la última (clave 9)
y se decrementa *cant*

Eliminar(7)

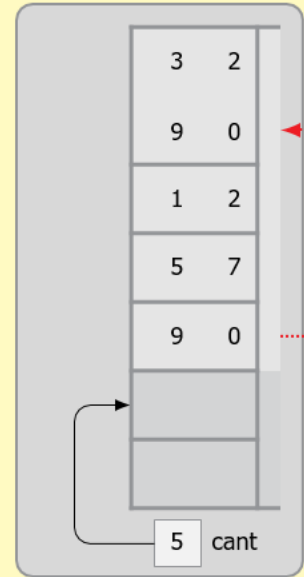


TDA Diccionario Simple. Implementación

Eliminar

La entrada con clave 7 se reemplaza
con la última (clave 9)
y se decrementa *cant*

Eliminar(7)

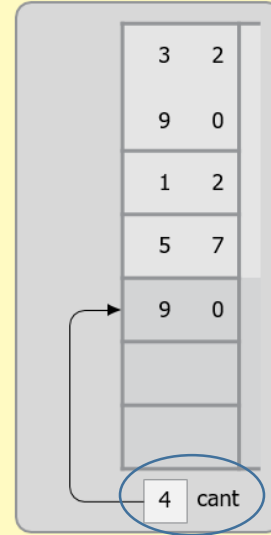


TDA Diccionario Simple. Implementación

Eliminar

La entrada con clave 7 se reemplaza
con la última (clave 9)
y se decrementa *cant*

Eliminar(7)




TDA Diccionario Simple. Implementación

```
public class DicSimpleA implements DiccionarioSimpleTDA {  
    class Elemento{  
        int clave;  
        int valor;  
    }  
    Elemento[] elementos;  
    int cant ;
```

TDA Diccionario Simple. Implementación

```
public void InicializarDiccionario () {  
    cant = 0;  
    elementos = new Elemento [100];  
}
```

```
public void Agregar( in t clave , in t valor){  
    int pos = Clave2Indice( clave);  
    if ( pos == -1) {  
        pos = cant ;  
        elementos[ pos ]= new Elemento ();  
        elementos[ pos ]. clave = clave;  
        cant ++;  
    }  
    elementos[ pos ]. valor = valor;  
}
```



```
private int Clave2Indice( int clave){  
    int i= cant -1;  
    while (i >=0 && elementos[i]. clave!= clave)  
        i --;  
    return i;  
}
```


TDA Diccionario Simple. Implementación

```
private int Clave2Indice( int clave){  
    int i= cant -1;  
    while (i >=0 && elementos[i]. clave!= clave)  
        i --;  
    return i;  
}
```

```
public void Eliminar( int clave) {  
    int pos = Clave2Indice( clave);  
    if ( pos != -1) {  
        elementos[ pos ] = elementos[ cant -1];  
        cant --;  
    }  
}
```

TDA Diccionario Simple. Implementación

```
public int Recuperar( int clave) {  
    in t pos = Clave2Indice( clave);  
    return elementos[ pos ]. valor;  
}  
  
public ConjuntoTDA Claves() {  
    ConjuntoTDA c=new ConjuntoLD() ;  
    c. InicializarConjunto () ;  
    for ( int i =0; i < cant ; i ++ ) {  
        c. Agregar( elementos[i ]. clave);  
    }  
    return c;  
}  
}
```

TDA Diccionario Múltiple

El diccionario múltiple tiene como característica principal que cada clave del diccionario puede tener asociado un **conjunto** de valores.

TDA Diccionario Múltiple

Operaciones:

Agregar , dada una clave y un valor, agrega al diccionario el valor quedando asociado a la clave. Una misma clave puede tener asociada un conjunto de valores, pero esos valores no se pueden repetir. Para poder agregar un par clave/valor la estructura debe estar inicializada.

Eliminar, dada una clave elimina todos los valores asociados a la clave, y por consiguiente la clave, ya que el diccionario no puede contener claves sin valores asociados. Si la clave no existe no se hace nada. Se supone que el diccionario esta inicializado.

TDA Diccionario Múltiple

Operaciones:

Agregar , dada una clave y un valor, agrega al diccionario el valor quedando asociado a la clave. Una misma clave puede tener asociada un conjunto de valores, pero esos valores no se pueden repetir. Para poder agregar un par clave/valor la estructura debe estar inicializada.

Eliminar, dada una clave elimina todos los valores asociados a la clave, y por consiguiente la clave ya que el diccionario no puede contener claves sin valores asociados. Si la clave no existe no se hace nada. Se supone que el diccionario esta inicializado.

EliminarValor, dada una clave y un valor se elimina el valor asociado a la clave, en caso de que la clave o el valor no existan no se hace nada. Si al eliminar el valor, la clave no tiene otros valores asociados se debe eliminar la misma. Se supone que el diccionario esta inicializado.

TDA Diccionario Múltiple

Operaciones:

Agregar, dada una clave y un valor, agrega al diccionario el valor quedando asociado a la clave. Una misma clave puede tener asociada un conjunto de valores, pero esos valores no se pueden repetir. Para poder agregar un par clave/valor la estructura debe estar inicializada.

Eliminar, dada una clave elimina todos los valores asociados a la clave, y por consiguiente la clave ya que el diccionario no puede contener claves sin valores asociados. Si la clave no existe no se hace nada. Se supone que el diccionario esta inicializado.

EliminarValor, dada una clave y un valor se elimina el valor asociado a la clave, y en caso de que la clave o el valor no existan no se hace nada. Si al eliminar el valor, la clave no tiene otros valores asociados se debe eliminar la misma. Se supone que el diccionario esta inicializado.

Recuperar, dada una clave devuelve el **conjunto de valores** asociados a la misma. Si la clave dada no pertenece al diccionario, se debe devolver un conjunto vacío. Se supone que el diccionario esta inicializado.

TDA Diccionario Múltiple

Operaciones:

Agregar, dada una clave y un valor, agrega al diccionario el valor quedando asociado a la clave. Una misma clave puede tener asociada un conjunto de valores, pero esos valores no se pueden repetir. Para poder agregar un par clave/valor la estructura debe estar inicializada.

Eliminar, dada una clave elimina todos los valores asociados a la clave, y por consiguiente la clave ya que el diccionario no puede contener claves sin valores asociados. Si la clave no existe no se hace nada. Se supone que el diccionario esta inicializado.

EliminarValor, dada una clave y un valor se elimina el valor asociado a la clave, y en caso de que la clave o el valor no existan no se hace nada. Si al eliminar el valor, la clave no tiene otros valores asociados se debe eliminar la misma. Se supone que el diccionario esta inicializado.

Recuperar: dada una clave devuelve el conjunto de valores asociados a la misma. Si la clave dada no pertenece al diccionario, se debe devolver un conjunto vacío. Se supone que el diccionario esta inicializado.

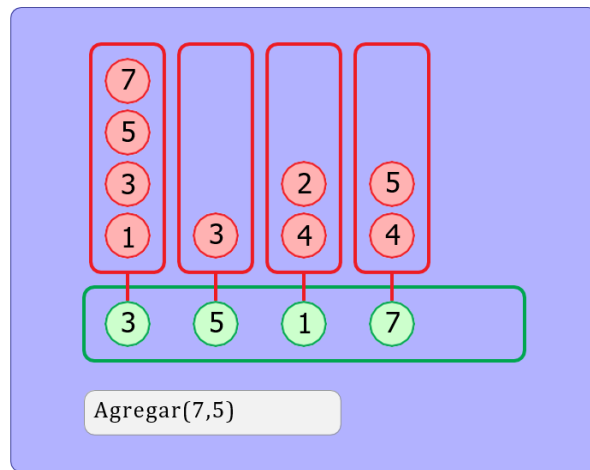
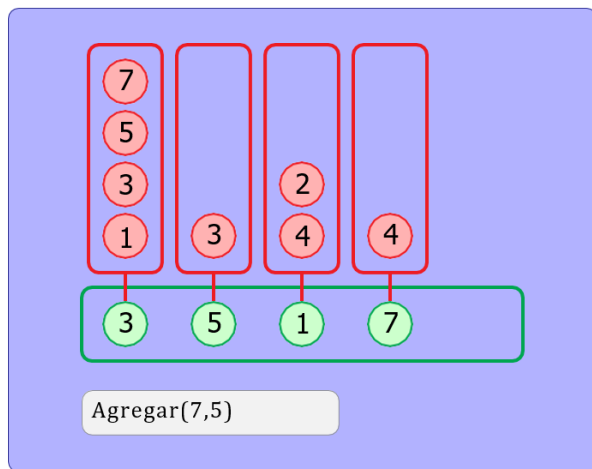
Claves: devuelve el conjunto de todas las claves definidas en el diccionario. Se supone que el diccionario esta inicializado.

InicializarDiccionario: permite inicializar la estructura del diccionario.

TDA Diccionario Múltiple

Operaciones:

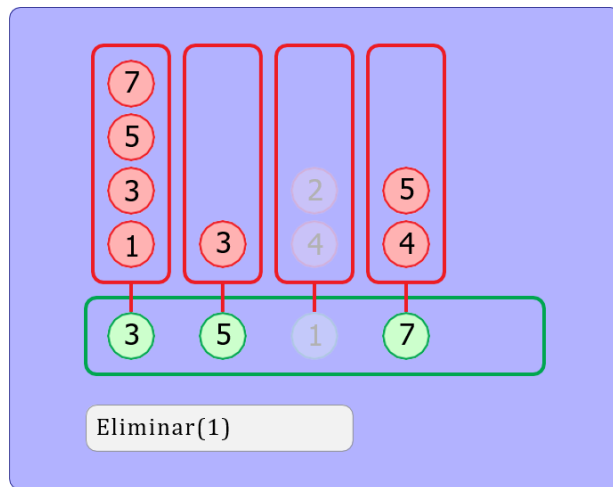
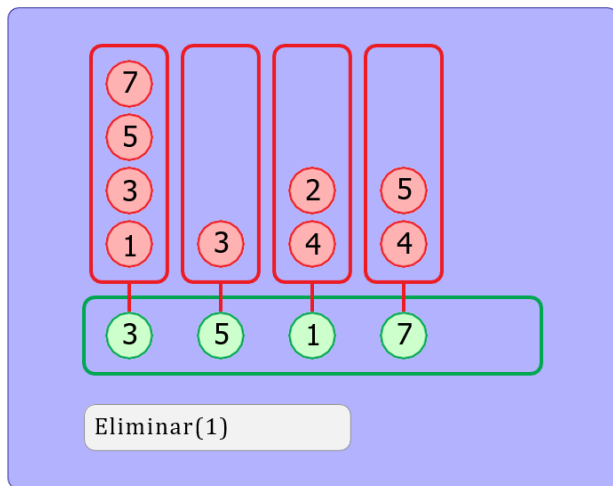
Agregar , dada una clave y un valor, agrega al diccionario el valor quedando asociado a la clave. Una misma clave puede tener asociada un conjunto de valores, pero esos valores no se pueden repetir. Para poder agregar un par clave/valor la estructura debe estar inicializada.



TDA Diccionario Múltiple

Operaciones:

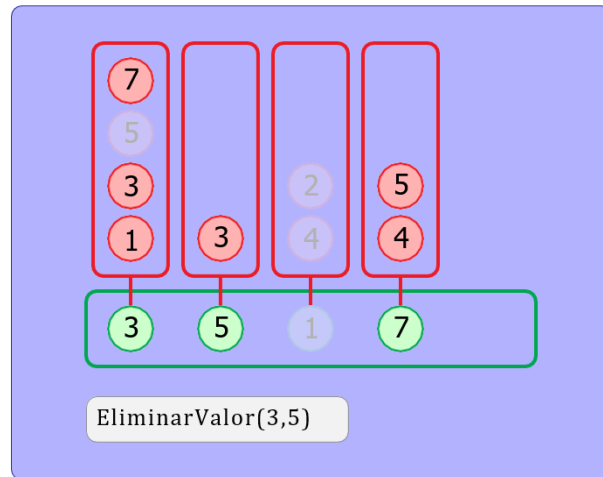
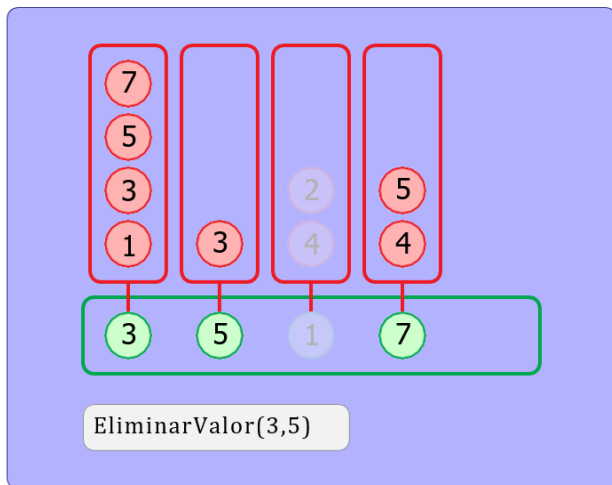
Eliminar, dada una clave elimina todos los valores asociados a la clave, y por consiguiente la clave ya que el diccionario no puede contener claves sin valores asociados. Si la clave no existe no se hace nada. Se supone que el diccionario esta inicializado.



TDA Diccionario Múltiple

Operaciones:

EliminarValor, dada una clave y un valor se elimina el valor asociado a la clave, en caso de que la clave o el valor no existan no se hace nada. Si al eliminar el valor, la clave no tiene otros valores asociados se debe eliminar la misma. Se supone que el diccionario esta inicializado.



TDA Diccionario Múltiple. Interfaz

```
1. public interface DiccionarioMultipleTDA; {  
2.     void InicializarDiccionario();           // sin precondiciones  
3.     void Agregar(int clave, int valor);      // diccionario inicializado  
4.     void Eliminar(int clave); // diccionario inicializado  
5.     void EliminarValor(int clave, int valor); // diccionario inicializado  
6.     ConjuntoTDA Recuperar(int clave); // diccionario inicializado  
7.     ConjuntoTDA Claves(); // diccionario inicializado  
8. }
```

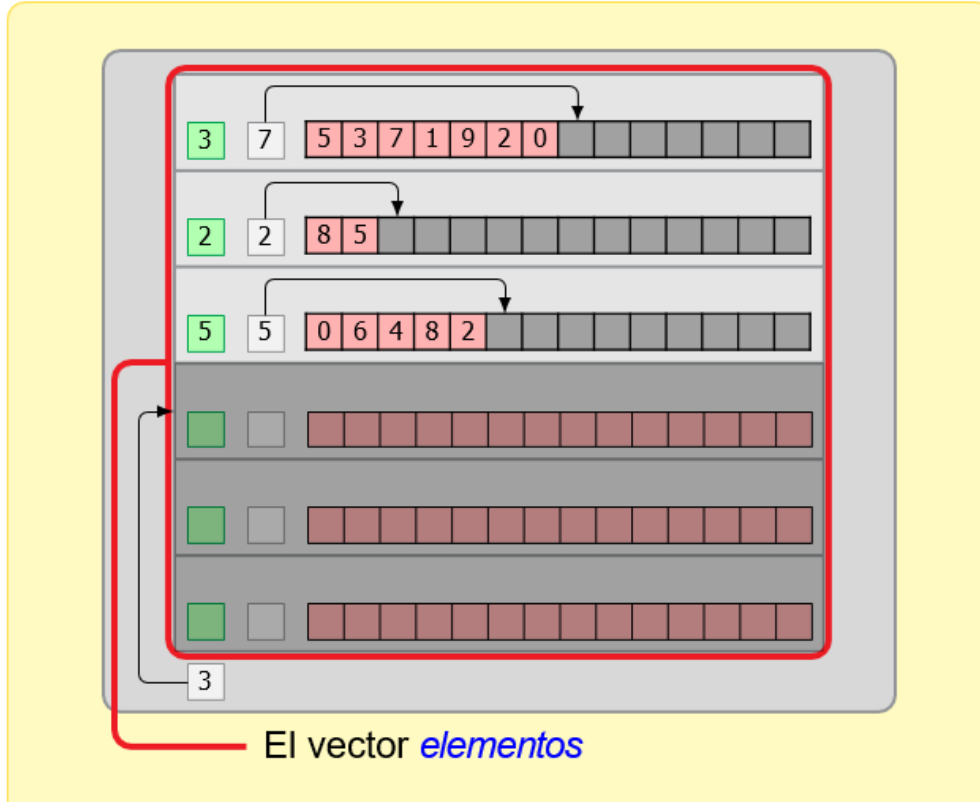
TDA Diccionario Múltiple. Pasaje de elementos de un diccionario simple *DicSim* a uno múltiple *DicMul*.

```
1. public void Pasaje (DiccionarioSimpleTDA DicSim,  
   DiccionarioMultipleTDA DicMul) {  
2.     ConjuntoTDA claves = new ConjuntoTDA();  
3.     claves.InicializarConjunto();  
4.     claves = DicSim.Claves();  
5.     while (!claves.ConjuntoVacio()) {  
6.         int clave = claves.Elegir();  
7.         int valor = DicSim.Recuperar(clave);  
8.         DicMul.Agregar(clave, valor);  
9.         claves.Sacar;  
10.    }  
11. }
```

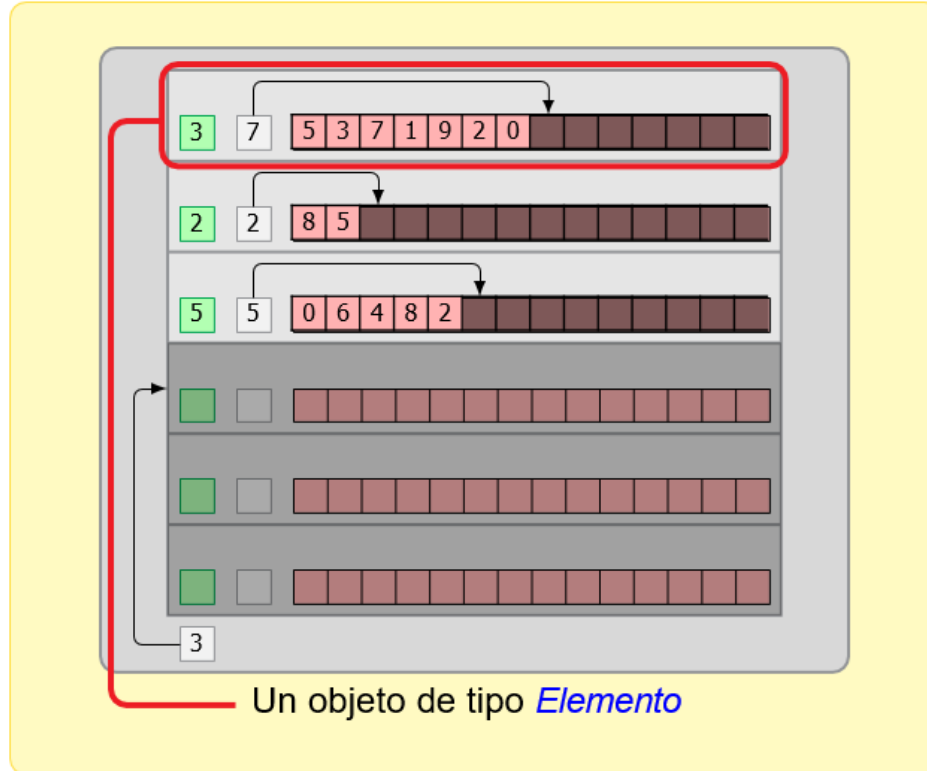
TDA Diccionario Múltiple. Implementación

- Clase **Elemento**, que contendrá una clave entera, un arreglo de números enteros (los valores) y una variable entera que nos dará (como para las anteriores colecciones), la cantidad de valores de la colección de la clave.
- El diccionario múltiple se representará entonces como un arreglo *elementos* de objetos de tipo **Elemento**.
 - Se definirán dos métodos privados (para uso exclusivo de la implementación): *Clave2Indice* que, dada una clave, devuelve el índice correspondiente en *elementos* y *Valor2Indice* que, dado un valor, devuelve el índice correspondiente en *valores*.
 - Se utilizará la variable entera *cantClaves* para indicar la cantidad de claves que hay en *elementos*.
 - Cuando se elimina una clave (o un valor), se reemplaza el elemento eliminado con el de la última posición.

TDA Diccionario Múltiple. Implementación



TDA Diccionario Múltiple. Implementación



TDA Diccionario Múltiple. Implementación

```
public class DicMultipleA implements DiccionarioMultipleTDA {
```

```
    class Elemento{
```

```
        int clave;
```

```
        int [] valores;
```

```
        int cantValores;
```

```
    }
```

```
    Elemento[] elementos;
```

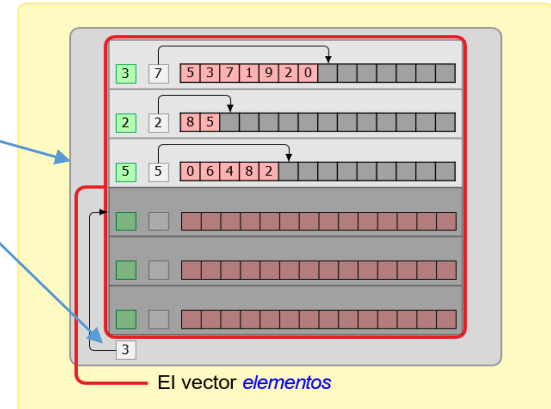
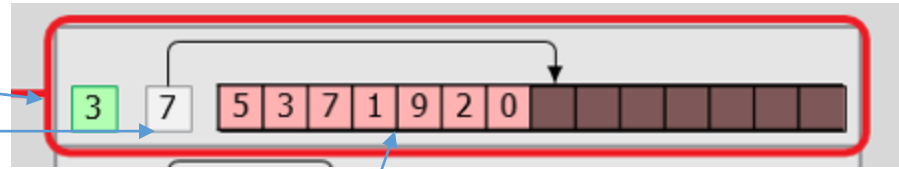
```
    int cantClaves;
```

```
    public void InicializarDiccionario () {
```

```
        elementos = new Elemento [100];
```


```
        cantClaves = 0;
```

```
    }
```



TDA Diccionario Múltiple. Implementación

```
public void Agregar( in t clave , in t valor){  
    int posC = Clave2Indice( clave);  
    if ( posC == -1) {  
        posC = cantClaves;  
        elementos[ posC ]= new Elemento();  
        elementos[ posC ]. clave = clave;  
        elementos[ posC ]. cantValores = 0;  
        elementos[ posC ]. valores = new in t [100];  
        cantClaves ++;  
    }  
    Elemento e = elementos[ posC ];  
    in t posV = Valor2Indice(e , valor);  
    if ( posV == -1) {  
        e. valores[e. cantValores] = valor;  
        e. cantValores ++;  
    }  
}
```




```
private int Clave2Indice( int clave){  
    int i = cantClaves -1;  
    while(i >=0 && elementos[i]. clave!= clave)  
        i --;  
    return i;  
}
```

dada una clave, devuelve el índice correspondiente en elementos

TDA Diccionario Múltiple. Implementación

```
private int Clave2Indice( int clave){  
    int i = cantClaves -1;  
    while(i >=0 && elementos[i ]. clave!= clave)  
        i --;  
    return i;  
}
```




```
private int Clave2Indice( int clave){  
    int i = cantClaves -1;  
    while(i >=0 && elementos[i ]. clave!= clave)  
        i --;  
    return i;  
}
```

```
public void Eliminar( in t clave) {  
    int pos = Clave2Indice( clave);  
    if ( pos != -1) {  
        elementos[ pos ] = elementos[ cantClaves -1];  
        cantClaves --;  
    }  
}
```

TDA Diccionario Múltiple. Implementación

```
public void EliminarValor( int clave , int valor) {  
    int posC = Clave2Indice( clave) ;  
    if ( posC != -1) {  
        Elemento e = elementos[ posC ];  
        int posV = Valor2Indice(e, valor);  
        if ( posV != -1) {  
            e.valores[ posV ] = e.valores[ e.cantValores -1];  
            e.cantValores --;  
            if (e.cantValores ==0) {  
                Eliminar( clave);  
            }  
        }  
    }  
}
```



```
private int Valor2Indice( Elemento e , int valor)  
{  
    int i = e . cantValores -1;  
    while(i >=0 && e.valores[i] != valor)  
        i --;  
    return i;  
}
```

dado un valor, devuelve el índice correspondiente en *valores*

TDA Diccionario Múltiple. Implementación

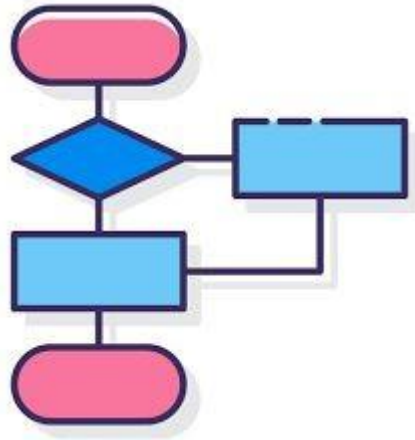
```
private int Valor2Indice( Elemento e , in t valor) {  
    int i = e . cantValores -1;  
    while(i >=0 && e. valores[i ]!= valor)  
        i --;  
    return i;  
}  
  
public ConjuntoTDA Recuperar( in t clave) {  
    ConjuntoTDA c=new ConjuntoLD() ;  
    c. InicializarConjunto () ;  
    in t pos = Clave2Indice( clave);  
    if ( pos != -1) {  
        Elemento e= elementos[ pos ];  
        for ( int i =0; i <e. cantValores; i ++ ) {  
            c. Agregar(e. valores[i ]) ;  
        }  
    }  
    return c;  
}
```

TDA Diccionario Múltiple. Implementación

```
public ConjuntoTDA Claves() {  
    ConjuntoTDA c=new ConjuntoLD() ;  
    c. InicializarConjunto () ;  
    for ( int i =0; i < cantClaves; i ++ ) {  
        c. Agregar( elementos[i ]. clave);  
    }  
    return c;  
}
```

Costos de algoritmos

¿Cómo evaluar un algoritmo?



Costos de algoritmos

Cumple la especificación y funciona para toda entrada

Es fácil de codificar y entender

No existe otro que resuelva lo mismo usando menos **recursos**

Costos de algoritmos

No existe otro que resuelva lo mismo usando menos **recursos**

Que son los recursos?

Memoria

Estática: Sumar la memoria de todas las variables

Dinámica: Es más complejo. Similar a la evaluación de tiempo

Costos de algoritmos

No existe otro que resuelva lo mismo usando menos **recursos**

Procesador

Depende de la computadora

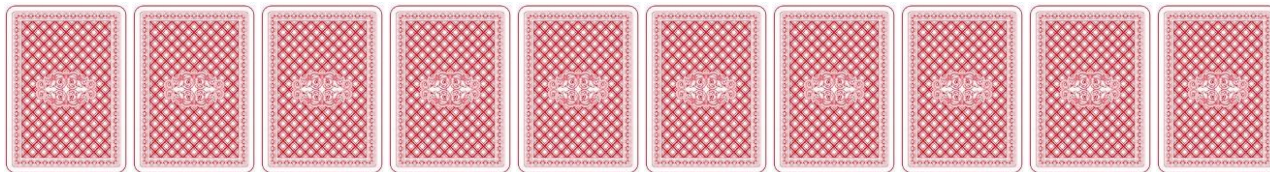
Tiempo que demora en terminar un algoritmo

Costos de algoritmos

Tiempo que demora en terminar un algoritmo

- No se puede medir con cronómetro
- Otros programas pueden alterar la medida
- Se debe buscar una medida objetiva Por ejemplo relativa al tamaño de la entrada

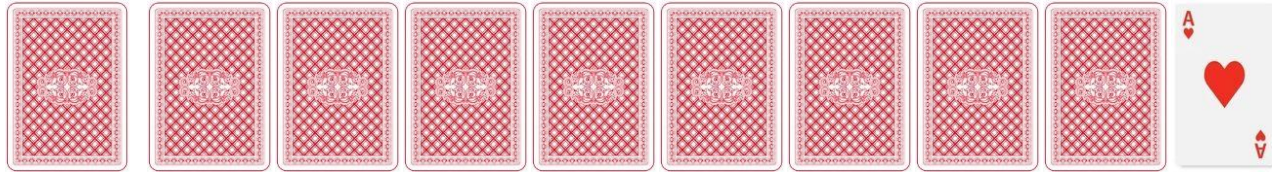
Buscamos la carta en un conjunto aleatorio



Mejor caso. Costo de búsqueda: 1 operación



Peor caso. Costo de búsqueda: 9 operaciones



Caso promedio. Costo de búsqueda: 4 operaciones



Costos de algoritmos

- Mejor Caso: Costo 1
- Peor Caso: Costo n , donde n es la cantidad de cartas
- Caso medio: Costo $n/2$

Definimos $t(n)$ como el costo en tiempo, donde n es el tamaño de la entrada

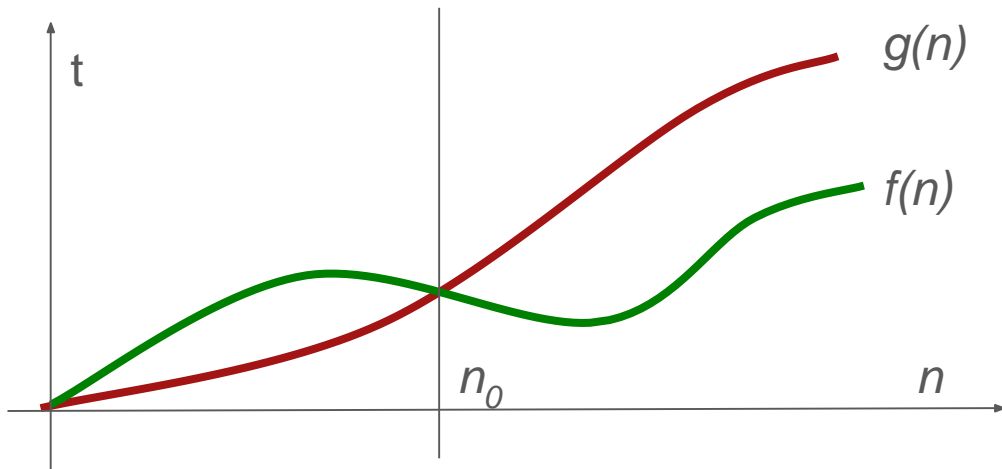
No es relativo al **tiempo**, sino al **peor** caso

Notación $O(n)$. Big Oh

$O(n)$ es una cota de una función $f(n)$

Quiere decir que a partir de un cierto valor $f(n)$ se comporta como $O(g(n))$

$$\frac{f(n)}{g(n)} \leq c, \forall n \geq n_0$$



Notación $O(n)$

Ejemplo

$$t(n) = 4n^3 + 2n^2$$

$$g(n) = n^3$$

$$O(4n^3 + 2n^2) = O(n^3)$$

$$\frac{f(n)}{g(n)} \leq c, \forall n \geq n_0$$

$$\frac{4n^3 + 2n^2}{n^3} \leq c$$

$$4 + \frac{2}{n} \leq c$$

Notación $O(n)$

Ejemplo

$$t(n) = 4n^3 + 2n^2$$

$$g(n) = n^3$$

$$O(4n^3 + 2n^2) = O(n^3)$$

$$\frac{f(n)}{g(n)} \leq c, \forall n \geq n_0 \quad 4 + \frac{2}{n} \leq c$$

$$\text{Si } n_0 = 1 \rightarrow 6 \leq c$$

$$\text{Si } n_0 = 2 \rightarrow 5 \leq c$$

$$\text{Si } n_0 = 3 \rightarrow 4,66.. \leq c$$

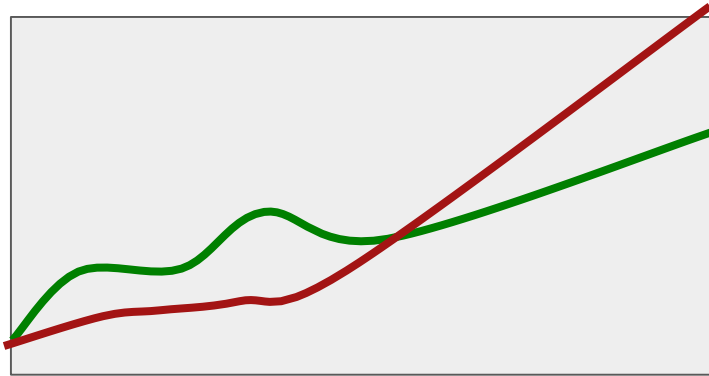
$$\lim_{n \rightarrow \infty} \left(4 + \frac{2}{n}\right) = 4$$

$4 \leq c$ es decir existe una c

$O(g(n))$ es un “resumen” de $f(n)$

$$\frac{f(n)}{g(n)} \leq c, \forall n \geq n_0$$

- Quiere decir que existe un c , que es cota de $f(n)/g(n)$ a partir de un cierto valor n_0 .
- Refleja las irregularidades iniciales del algoritmo y la estabilidad que alcanzan a partir de un valor



Propiedades $O(n)$

- $c O(f(n)) \equiv O(f(n))$
- $O(f(n)) + O(f(n)) \equiv O(f(n))$
- $O(f(n)) + O(g(n)) \equiv O(f(n) + g(n))$
- $O(f(n)) * O(g(n)) \equiv O(f(n) * g(n))$
- $O(O(f(n))) \equiv O(f(n))$

Complejidad de las operaciones

- Asignaciones, lecturas, escrituras y comparaciones

- `x = a + 1;`

`System.out.println(x);`

$O(1)$

Complejidad de las operaciones

- Complejidad de un bloque de operaciones

{

$O(a_1)$

$O(a_2)$

..

$O(a_n)$

}

Es la suma de las
complejidades de cada
instrucción

$$O(a_1)+O(a_2)+..+O(a_n)$$

Complejidad de las operaciones

- Sentencia *if*

```
if (O(a1)) {
```

```
    O(a2)
```

$O(a_1) + \max(O(a_2), O(a_3))$

```
} else {
```

```
    O(a3)
```

```
}
```

Complejidad de las operaciones

- Sentencia *case*

```
switch (expresion) { O(a1)
```

```
    case a2:    O(a2)
```

```
    case a3:    O(a3)
```

```
    case an-1:    O(an-1)
```

```
    default an O(an)
```

$O(a_1) + \max(\{ O(a_2), \dots, O(a_n) \})$

```
}
```


Complejidad de las operaciones

- Bucles

```
for (int i=1; i<n; i++) {  
    a_i  
}
```

$O(a_1)+O(a_2)+\dots+O(a_n)$

Si a_i es $O(1) \rightarrow n \cdot O(1) = O(n)$

Complejidad de las operaciones

- Bucles

<pre>for (int j=1; j<n; j++) { for (int i=1; i<n; i++) { a_i } }</pre>	<pre>for (int j=1; j<n; j++) { O(n) }</pre>
	$n \cdot O(n) = O(n \cdot n) = O(n^2)$

Si tenemos k ciclos anidados tendremos $O(n^k)$

Comparativa

n	log(n)	n	n . log(n)	n ²	n ³	2 ^{**} n
1	0,00	1	0,00	1	1	2
5	0,70	5	3,49	25	125	32
10	1,00	10	10,00	100	1000	1024
20	1,30	20	26,02	400	8000	1048576
50	1,70	50	84,95	2500	125000	1,1259E+15
100	2,00	100	200,00	10000	1000000	1,26765E+30
200	2,30	200	460,21	40000	8000000	1,60694E+60
500	2,70	500	1349,49	250000	125000000	3,2734E+150
1000	3,00	1000	3000,00	1000000	1000000000	1,0715E+301

Resumen

Hemos visto una forma analítica de comparación de algoritmos.

Si el algoritmo se usará poco, se recomienda utilizar el algoritmo más sencillo de escribir.

Si utilizará una cantidad reducida de datos, usar un caso medio o empírico.

Eficiente, pero muy complejo: puede no ser útil

Los accesos a disco, o red, son muy lentos, por lo tanto se requiere reducirlos

En algoritmos de precisión matemática, puede requerirse algoritmo complejos, el análisis es mas minucioso.

Ejercicio

Calcular la complejidad del siguiente fragmento

```
if (n % 2 == 0) {  
    for (int i = 0; i < n; i++) {  
        x = x+1;  
    }  
} else
```

```
{  
    i = 1  
    while ( i<= n) {  
        x = x+1;  
        i = i +1;  
    }  
}
```