

# Arboles B

Ing. Max Alejandro Antonio Cerna  
Flores

# Agenda

- Definición
- Historia
- Componentes
- Coste Computacional
- Operaciones
  - Búsqueda
  - Inserción
  - Borrado
  - dividir/reestructurar\*
- Ventajas y Desventajas

# Definición

En informática, un árbol B es una estructura de datos de árbol auto equilibrado, que mantiene datos ordenados y permite búsquedas, acceso secuencial, inserciones y eliminaciones en tiempo logarítmico.

Tiene un coste menor de acceso a datos que otras estructuras como los árboles AVL y árboles binarios, reduciendo la necesidad de utilizar el almacenamiento secundario.

# Historia

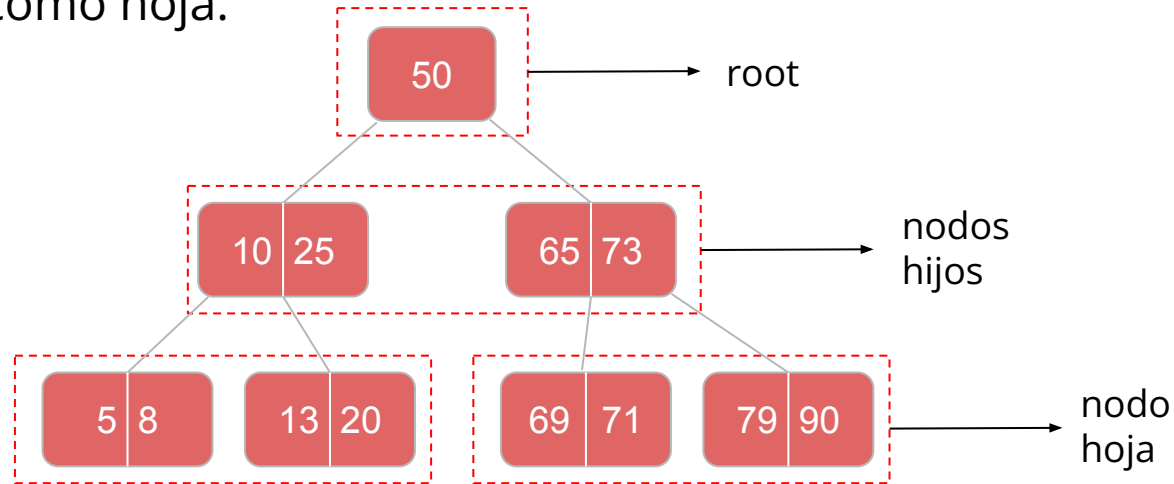
Los árboles B fueron inventados en los años 70 por Rudolf Bayer y Edward McCreight quienes trabajaban en Boeing Research Labs.

Querían diseñar una mejor versión de los árboles binarios, al abordar cargas de datos almacenados en discos.

El árbol B es ideal para sistemas de almacenamiento que leen y escriben bloques de datos relativamente grandes, como bases de datos y sistemas de archivos.

# Componentes

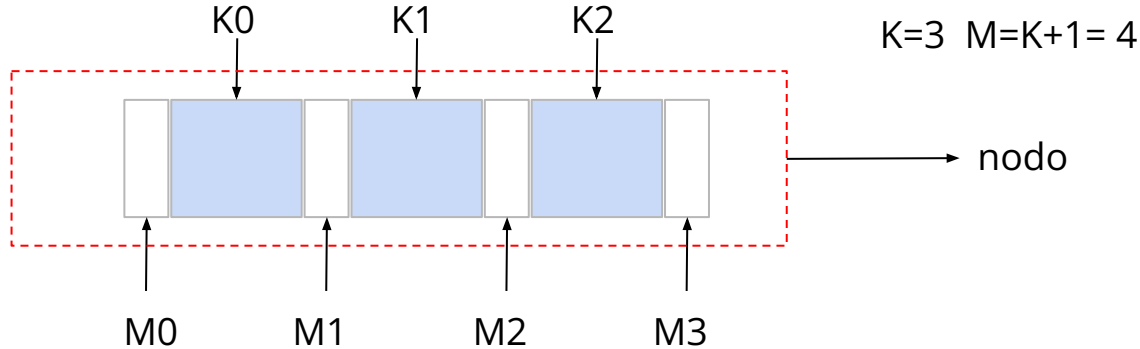
Los árboles B, están compuestos por nodos, teniendo un nodo origen conocido como raíz (root) y sus nodos hijos, cuando un nodo no tiene más hijos se conoce como hoja.



# Componentes

## Nodos en árboles B

- Cada nodo está compuesto de K claves (key).
- Cada nodo tendrá M hijos (  $M = K + 1$  ).



# Componentes

## Nodos

- Un nodo de árbol-B también es llamado Pagina (Page)
- Cada nodo (excepto root) tiene como mínimo  $(M)/2$  hijos.
- Todos los nodos hoja, están en el mismo nivel.
- El orden de las claves en cada nodo debe estar ordenado.

$$k_1 > k_2 > k_3 \dots > k_n$$

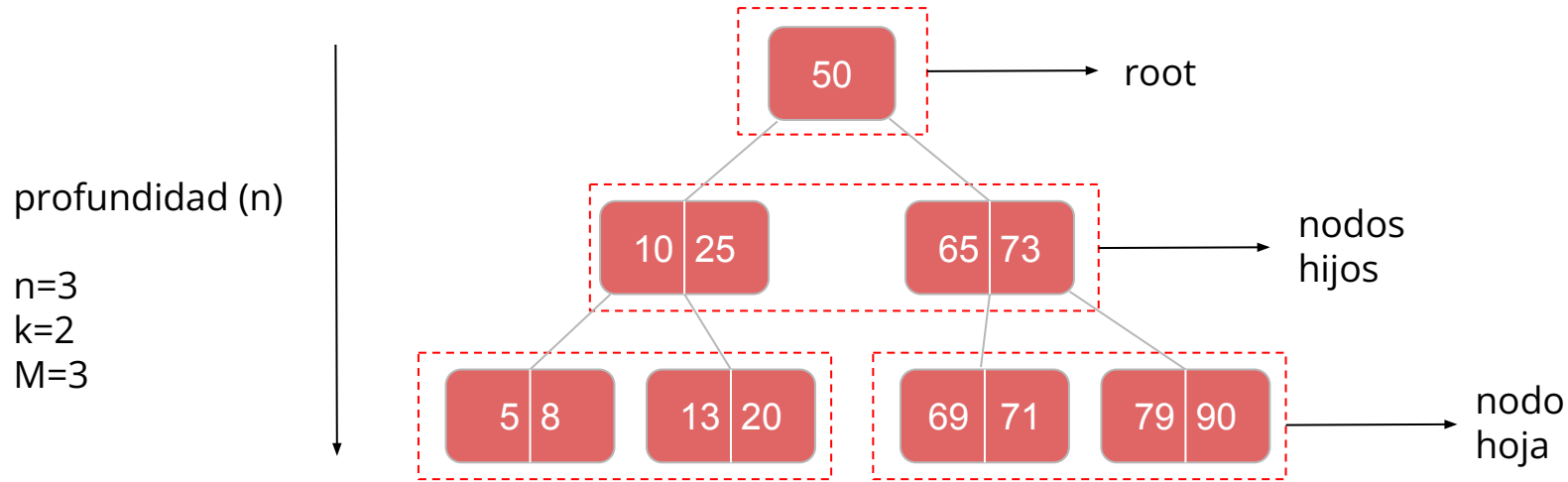
# Componentes

## Árbol

- El árbol está ordenado.
- El grado de un árbol-B es el número máximo de hijos que pueden partir de un nodo (M).
- La profundidad es el número máximo de consultas para encontrar una clave.



# Componentes



Cuanto mayor sea el grado, menor será la profundidad.

# Coste Computacional

Altura/Profundidad (mejor escenario/altura mínima)

$$\text{altura} = \log_m (k+1)$$

Altura/Profundidad (peor escenario/altura máxima) tomando  $d = M/2$

$$\text{altura} = \log_d ((k+1)/2)$$

# Coste Computacional

Operación	Promedio	Peor escenario
<b>Búsqueda</b>	$O(\log n)$	$O(\log n)$
<b>Inserción</b>	$O(\log n)$	$O(\log n)$
<b>Borrado</b>	$O(\log n)$	$O(\log n)$

# Operaciones

## **Búsqueda**

La búsqueda es similar a la búsqueda en un árbol de búsqueda binaria.

Comenzando en la raíz, el árbol se recorre recursivamente de arriba a abajo.

En cada nivel, la búsqueda reduce su campo de visión al subárbol.

# Operaciones

## **Búsqueda**

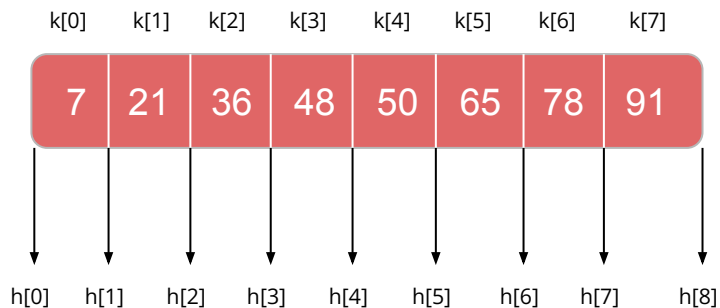
La búsqueda implica:

- Buscar entre las claves contenidas del nodo
- Sino no se encuentra, se salta al nodo hijo donde puede estar.
- Si se llega a un nodo hoja, notificar de no encontrado.

# Búsqueda

Buscar: 71

Grado: 9



# Búsqueda

## Estructura de un Nodo (PSEUDO CÓDIGO)

```
estructura nodo {  
  
    numClaves: Entero;  
  
    claves: Entero[];  
  
    numHijos: Entero;  
  
    hijos: Nodo[];  
  
    esHoja: booleano;  
  
    clavesUsadas: Entero;  
  
}
```

## ArbolB (PSEUDO CÓDIGO)

```
arbol {  
  
    grado: entero;  
  
    raiz: nodo;  
  
}
```

# Búsqueda

## PSEUDO-CODIGO

```
Busqueda(nodo, valor){  
    i : Entero;  
  
    i = 0;  
  
    mientras(i < nodo.numClaves && nodo.clave[i] < valor)  
        i = i + 1;  
  
    if(i < nodo.numClaves && nodo.clave[i] == valor)  
        return (nodo,i);  
  
    if(nodo.hoja)  
        return (nodo, null);  
  
    else  
        return busqueda(nodo, valor);  
}
```



# Operaciones

## Inserción

Consiste en introducir elementos en el árbol B.

La aceptación de duplicados es un criterio al momento de diseñar el árbol B.

**INSERTAR**(nodo, valor)

# Inserción

Una inserción implica:

- Salvo ciertas excepciones, los elementos serán insertados en una hoja.
  - Si el nodo no es una hoja, se debe dirigir al hijo más prometedor e intentarlo de nuevo.
  - La inserción en una hoja se hace metiendo el elemento de manera ordenada en el arreglo de claves.
- Se debe respetar el grado del árbol. Si se tienen demasiadas claves hay que ***dividir*** el nodo.

# Inserción

## Parámetros

Árbol B de grado M.

Máximo

- Como mucho M descendientes.
- Como mucho M-1 claves.

Mínimo

- Como mínimo  $M/2$  descendientes.
- Como mínimo  $(M/2)-1$  claves.

# Inserción

## **Arbol de grado 7**

$$N = 7$$

$$\text{Min. punteros: } N/2 = 3.5 \Rightarrow 4.$$

$$\text{Min. claves: } (N/2) - 1 = 2.5 \Rightarrow 3.$$

$$\text{Max. punteros: } 7$$

$$\text{Max. claves: } 6$$

# Inserción

Analicemos el siguiente nodo hoja:

Elemento a insertar: 29



# Inserción

Analicemos el siguiente nodo hoja:

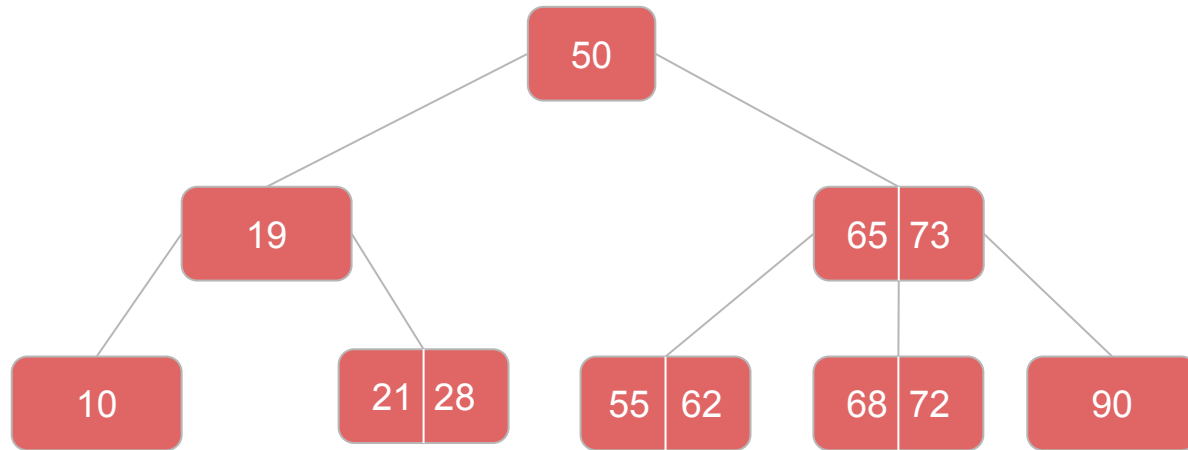
Elemento a insertar: 29



# Inserción

Analicemos el siguiente árbol B:

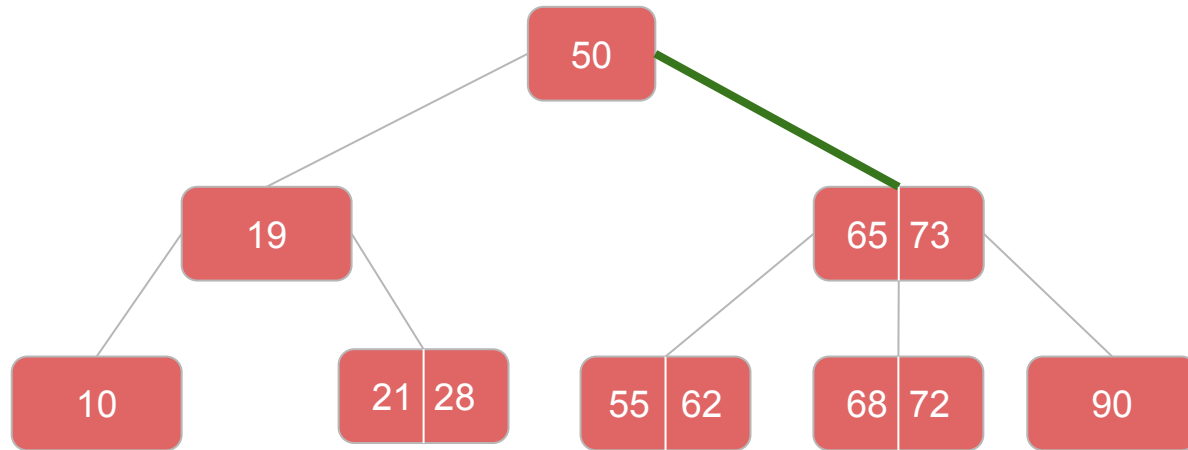
Elemento a insertar: 83



# Inserción

Analicemos el siguiente árbol B:

Elemento a insertar: 83

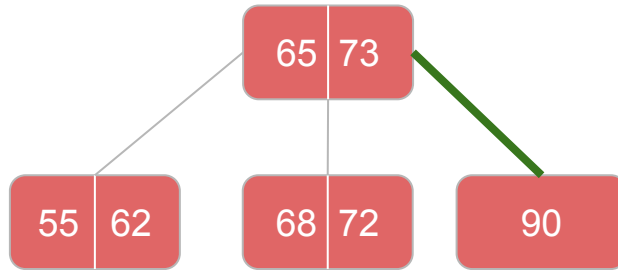




# Inserción

Analicemos el siguiente árbol B:

Elemento a insertar: 83



# Inserción

Analicemos el siguiente árbol B:

Elemento a insertar: 83



# Inserción

Se definirá U (Upper) al número de máximo de claves entonces:

$$U = M - 1$$

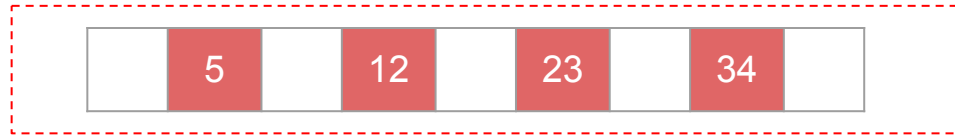
Se definirá L (Lower) al número mínimo de claves entonces:

$$L = (M/2) - 1$$

# Inserción

**M= 5** (L = 2, U = 4)

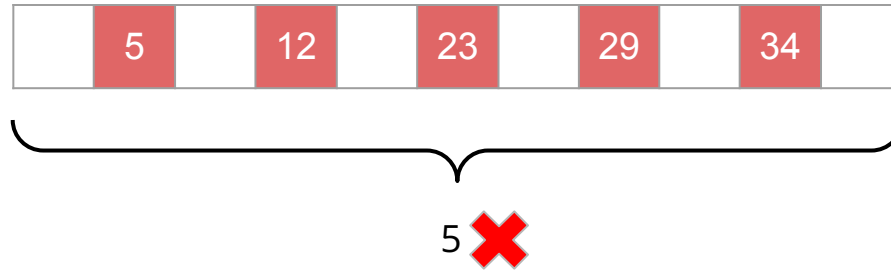
Inserción 29



nodo hoja

# Inserción

**M= 5** (L = 2, U = **4**)



# Inserción

**M= 5** (L = 2, U = 4)

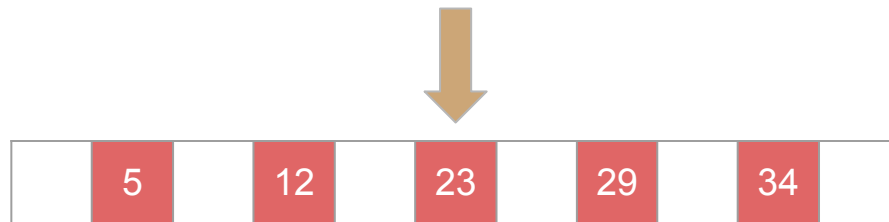
Inserción 29

**Solución:** será necesario dividir.

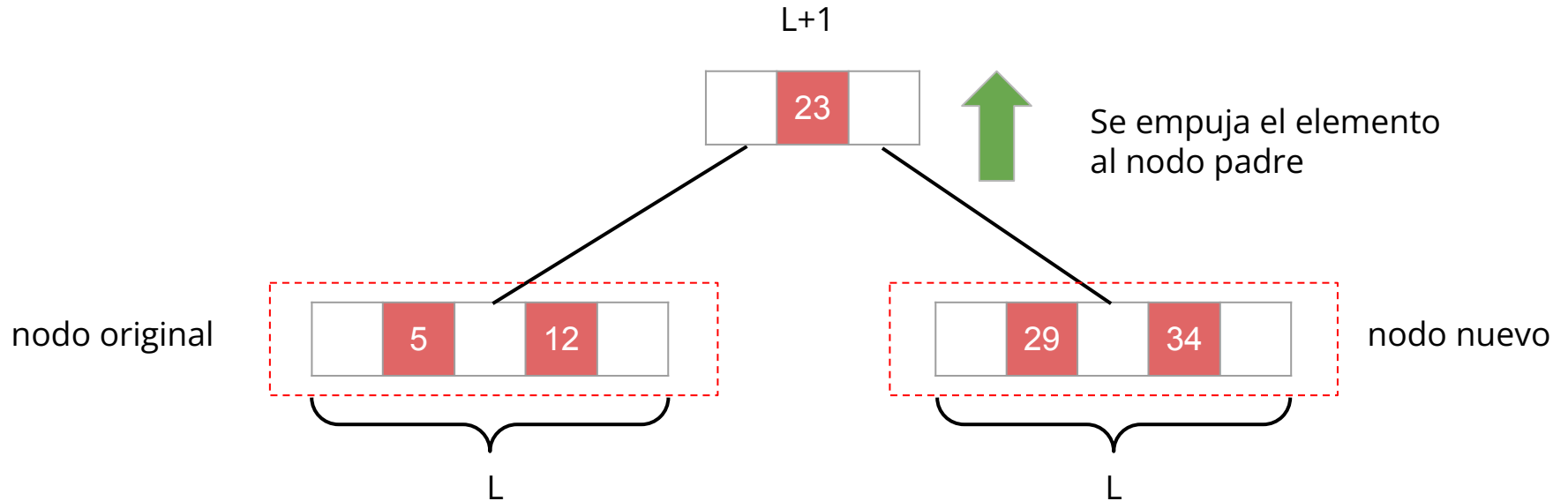
	5		12		23		34	
--	---	--	----	--	----	--	----	--

# Inserción - División

Buscar elemento pivote



# Inserción - División





# Inserción - División

## Otro ejemplo

**M= 6** (L = 2, U = 5)

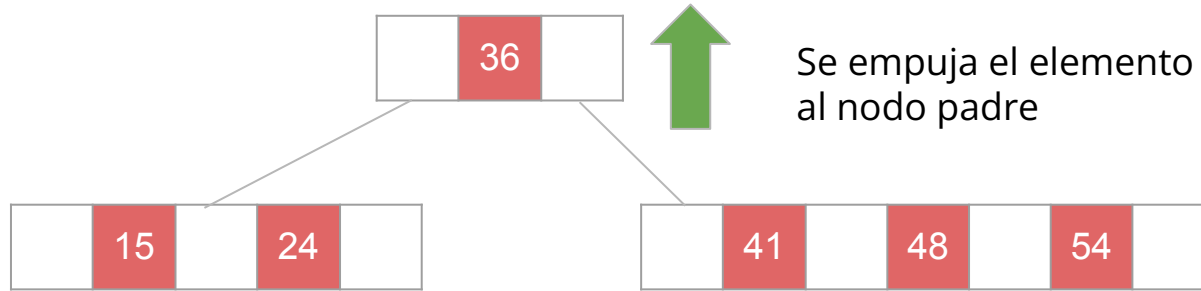
Nodo sobrepasado



nodo hoja

# Inserción - División

**M= 6** (L = 2, U = 5)



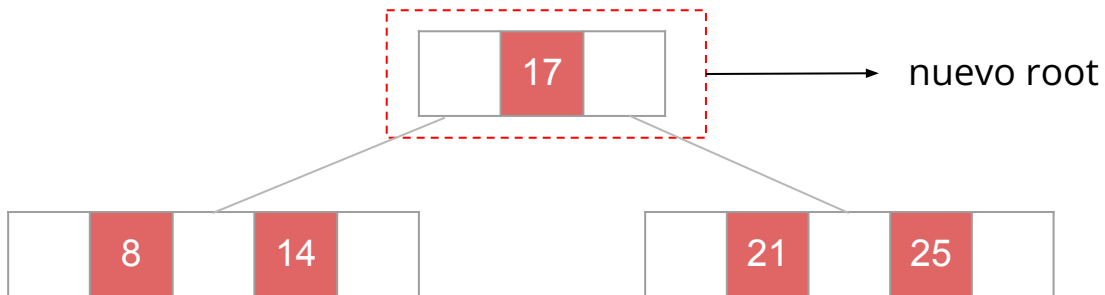
# Inserción - División en Raíz

**M= 5** (L = 2, U = 4)



# Inserción - División en Raíz

**M= 5** (L = 2, U = 4)



# Resumen División

- Si luego de hacer una inserción un nodo tiene más claves que lo permitido, entonces se divide en dos subnodos el nodo.
- Se toma un elemento pivote que estará en la mediana (posición del arreglo de claves).
- Se traslada las claves e hijos mayores que L a un nuevo nodo y se retiran del nodo viejo, que será el otro subnodo.
- Se empuja el pivote hacia arriba en el árbol.
- El nodo padre que recibe el pivote puede colapsar también, pidiendo una división.
- Como en el nodo raíz no hay padre al que empujar el pivote, se redefine la raíz del árbol.

# Inserción

## Estructura de un Nodo (PSEUDO CÓDIGO)

```
estructura nodo {  
  
    numClaves: Entero;  
  
    claves: Entero[];  
  
    numHijos: Entero;  
  
    hijos: Nodo[];  
  
    esHoja: booleano;  
  
    clavesUsadas: Entero;  
  
}
```

## ArbolB (PSEUDO CÓDIGO)

```
arbol {  
  
    t : entero;  
  
    raiz: nodo;  
  
}
```

**NOTA:** llamaremos **t** al grado mínimo (o sea **grado/2**)

# Inserción

## Constructor de un Nodo (PSEUDO CÓDIGO)

```
nodo {  
  
    constructor nodo(t) {  
  
        clavesUsadas = 0;  
  
        esHoja = verdadero;  
  
        clave[(2*t - 1)];  
  
        hijo[2*t];  
  
    }  
  
}
```

# Inserción en Nodo

```
insercionEnNodo(nodo,valor) {  
    if(nodo.esHoja) {  
        i: Entero;  
        i = nodo.clavesUsadas;  
        mientras(i>=1 && valor < nodo.clave[i-1]){  
            nodo.clave[i] = nodo.clave[i-1];  
            i --;  
        }  
        nodo.clave[i] = valor;  
        nodo.clavesUsadas++;  
    }  
}
```

```
else { //el nodo no es hoja  
    j: Entero; j = 0;  
    mientras(j < nodo.clavesUsadas && valor > nodo.clave[j]){  
        j++;  
    }  
    if(nodo.hijo[j].clavesUsadas == (2 * t - 1) ){// ( U - maximo de claves)  
        dividir(nodo, j, nodo.hijo[j]);  
        if(valor > nodo.clave[j]) { j++; }  
    }  
    insercionEnNodo(nodo.hijo[j], valor);  
}
```



# Inserción

```
insertar( valor) {  
    r : Nodo;  
    r = raiz;  
    if(r.clavesUsadas == (2*t - 1)) { // 2*t - 1 ( U - maximo de claves)  
        s: Nodo(t);  
        raiz = s;  
        s.esHoja = falso;  
        s.clavesUsadas = 0;  
        s.hijo[0] = r;  
        dividir(s,0,r); // es lo mismo decir dividir(s,0,s.hijo[0]);  
        insercionEnNodo(s, valor);  
    } else {  
        insercionEnNodo(s, valor);  
    }  
}
```

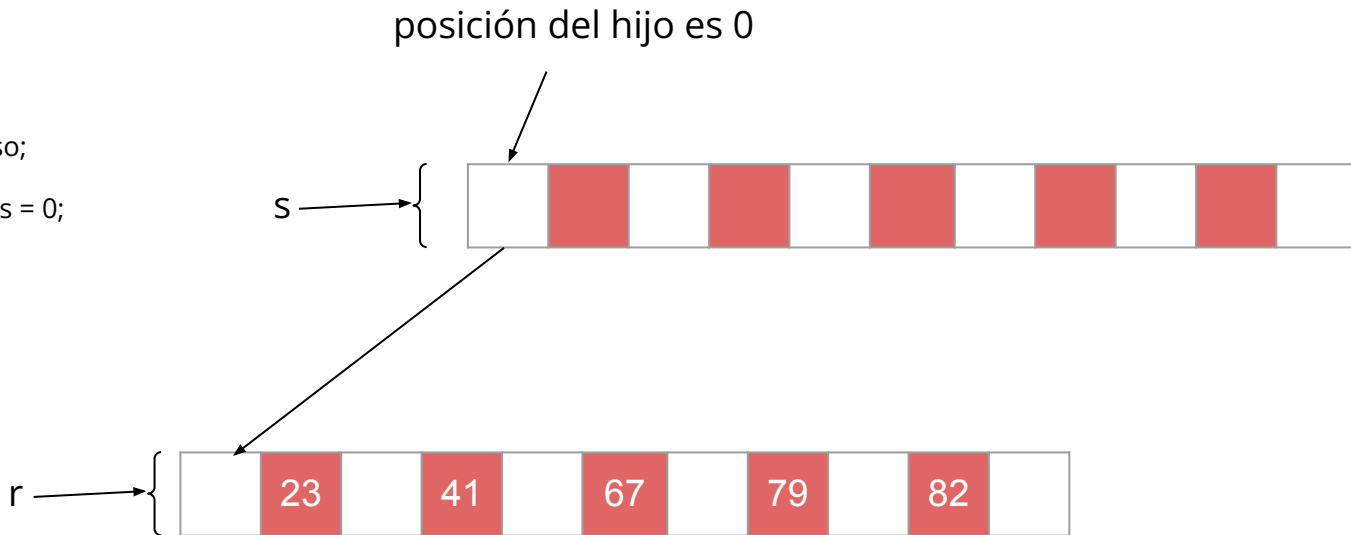
# División

si  $t = 4$

si  $\text{raiz} = \{ 23, 41, 67, 79, 82 \}$

```
s: Nodo(t);  
raiz = s;  
s.esHoja = falso;  
s.clavesUsadas = 0;  
s.hijo[0] = r;
```

**dividir(s,0,r);**



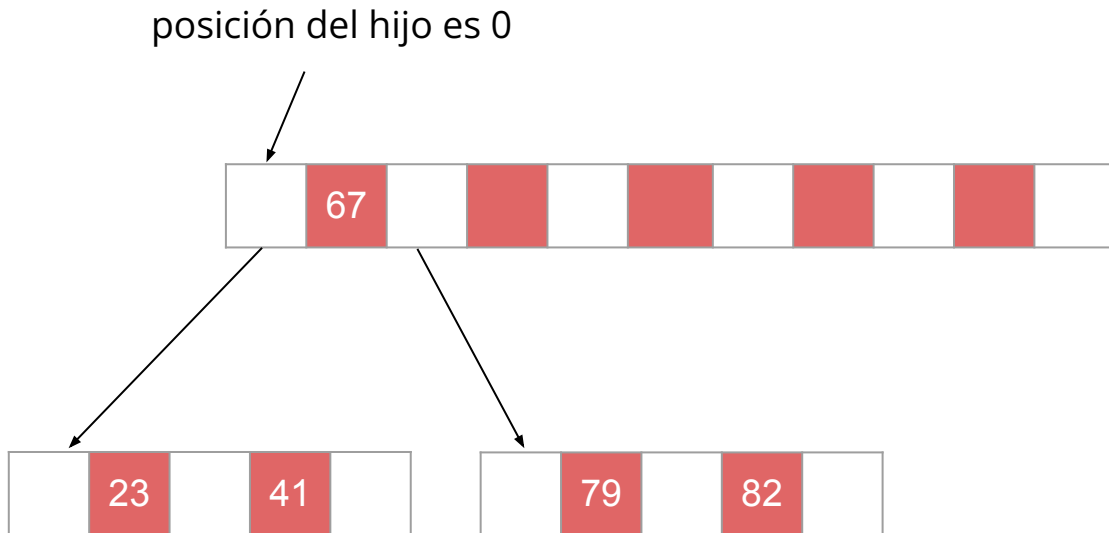
# División

si  $t = 3$

si  $\text{raiz} = \{ 23, 41, 67, 79, 82 \}$

```
s: Nodo(t);  
raiz = s;  
s.esHoja = falso;  
s.clavesUsadas = 0;  
s.hijo[0] = r;
```

**dividir(s,0,r);**



# División

```
dividir(npadre, posicion, nhijo) {  
    newNode : Nodo;  
    newNode.esHoja = nhijo.esHoja;  
    newNode.clavesUsadas = (t - 1); // L - minimo de claves  
    para(int j = 0; j < (t - 1); j++) { // copia ultimas claves del nodo hijo al newNode  
        newNode.clave[j] = nhijo.clave[(j+t)];  
    }  
    if(nhijo.esHoja == falso){  
        para(int k=0; k < t; k++) {  
            newNode.hijo[k] = nhijo.hijo[(k+t)];  
        }  
    }  
}
```

```
    nhijo.clavesUsadas = t - 1;  
    para(int j = npadre.clavesUsadas; j < posicion ; j++) {  
        npadre.hijo[(j+1)] = npadre.hijo[(j)];  
    }  
    npadre.hijo[(posicion+1)] = newNode;  
    para(int j = npadre.clavesUsadas; j < posicion ; j--) {  
        npadre.clave[(j+1)] = npadre.clave[j];  
    }  
    npadre.clave[posicion] = nhijo.clave[(t-1)];  
    npadre.clavesUsadas++;  
}
```

# Borrado

Proceso similar a una inserción, el borrado implica:

- En lugar de dividir , se realiza una unión o redistribución.
- El borrado puede ocurrir en cualquier lugar del árbol, contrario a la inserción que ocurría únicamente en las hojas.

# Borrado

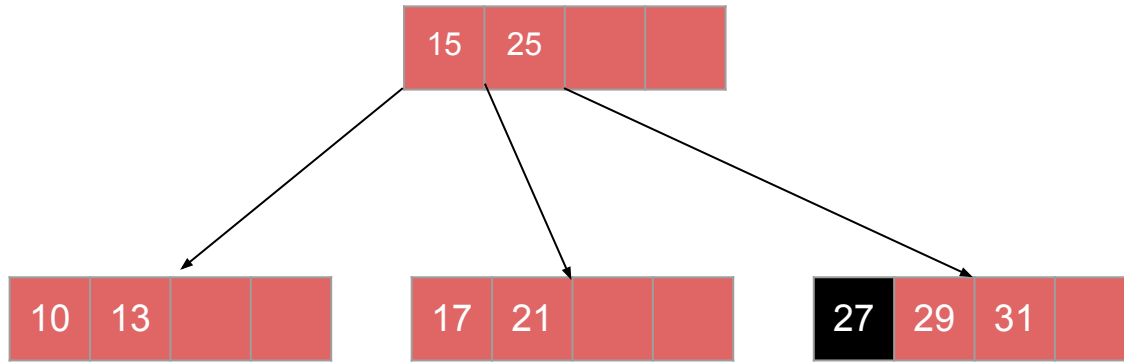
- Si la clave a eliminar se encuentra en una hoja, se elimina directamente.
  - Si al realizar la eliminación, el nodo mantiene el mínimo número de claves ( $M/2 - 1$ ), finaliza.
  - De lo contrario se debe hacer una redistribución.
- Si se encuentra dentro de un nodo interno, no se puede suprimir de forma directa
  - Se debe “subir” la clave que se encuentra más a la derecha en el subárbol izquierdo o más a la izquierda en el subárbol derecho.
- Si al subir la clave, en el nodo respectivo no se cumple el mínimo número de claves ( $M/2 - 1$ ), se debe realizar una redistribución.

# Borrado - Redistribución

- **NOTA:**  $d = ((M/2) - 1)$
- Si un nodo vecino tiene suficiente número de claves ( $>d$ ), realiza un ***préstamo***.
  - La clave que se encuentra más a la izquierda “sube”.
  - La clave del nodo padre “baja”.
- Si un nodo vecino no tiene suficiente número de claves ( $\leq d$ ) , se realiza una unión entre el nodo donde se realiza el borrado (**nodo de supresión**), su vecino y su padre.

# Ejemplo 1- Borrado

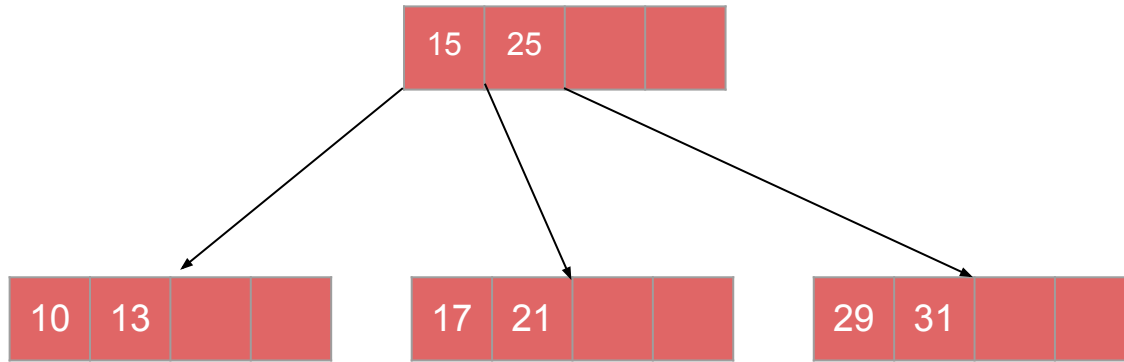
Eliminar valor 27 (M=5, U=4, L=2)





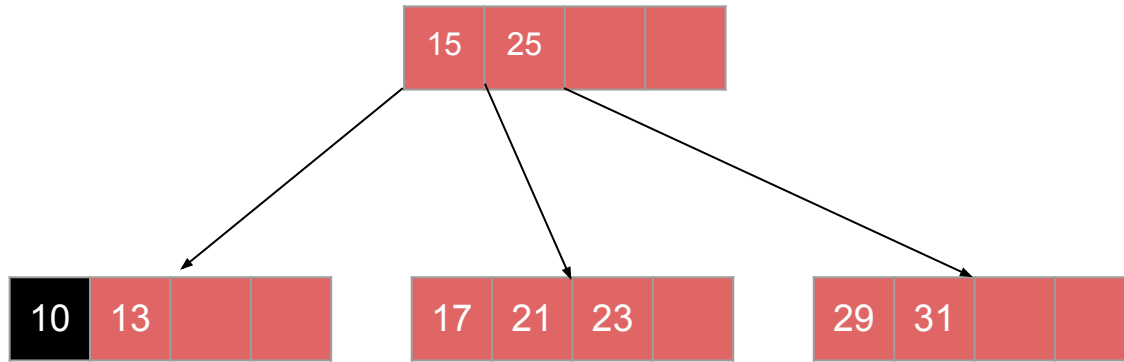
# Ejemplo 1- Borrado

Eliminar valor 27 (M=5, U=4, L=2)



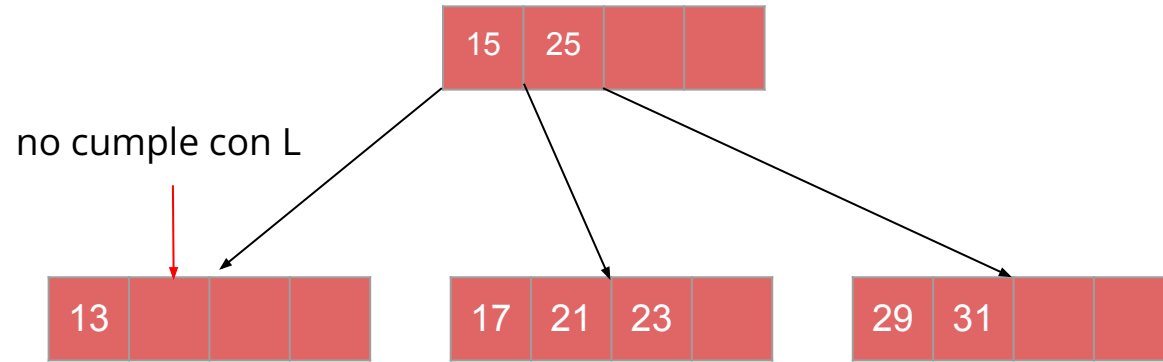
# Ejemplo 2- Borrado

Eliminar valor 10 ( $M=5$ ,  $U=4$ ,  $L=2$ )



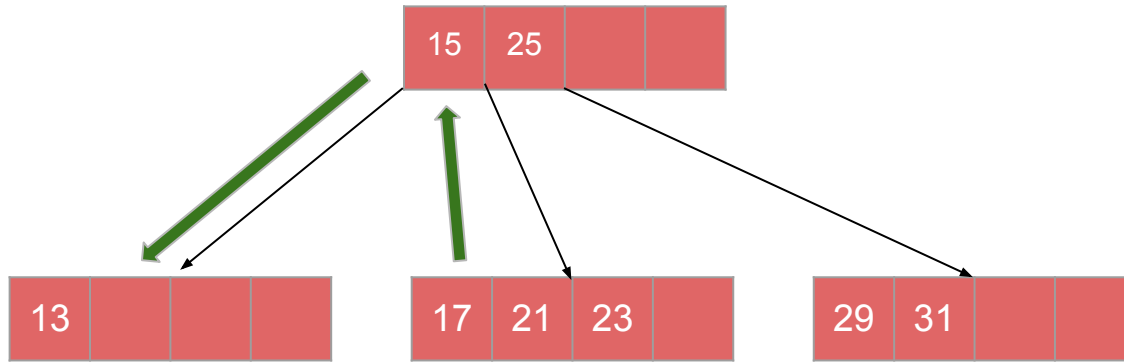
# Ejemplo 2- Borrado

Eliminar valor 10 ( $M=5$ ,  $U=4$ ,  $L=2$ )



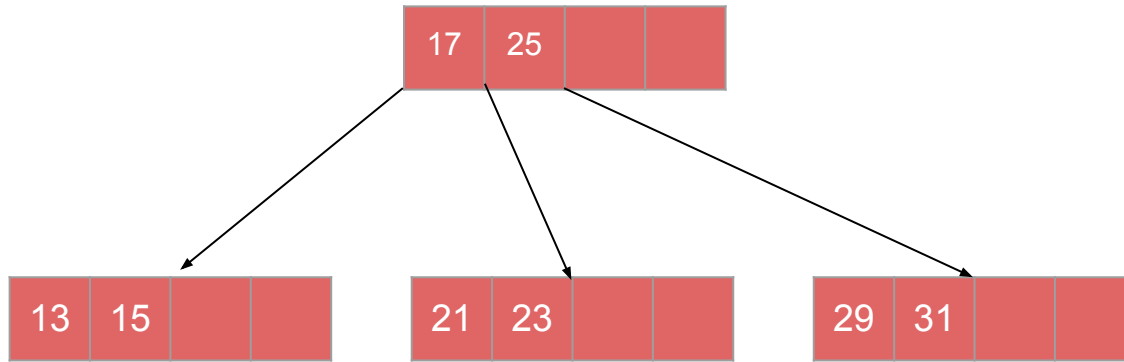
# Ejemplo 2- Borrado

Eliminar valor 10 (M=5, U=4, L=2)



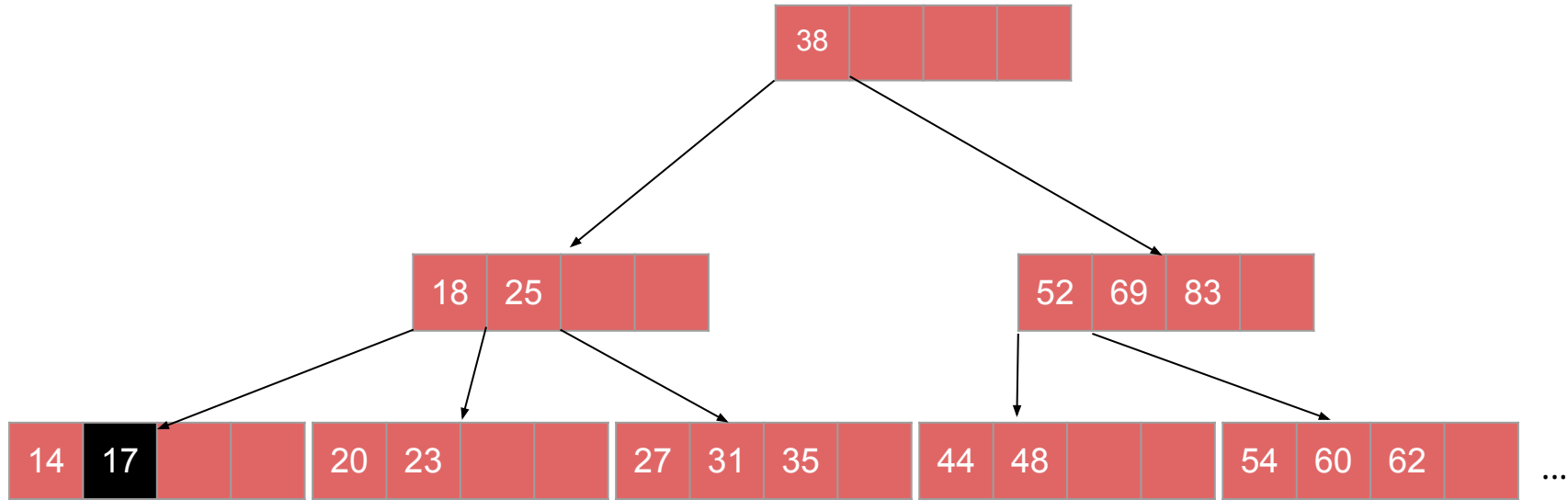
# Ejemplo 2- Borrado

Eliminar valor 10 ( $M=5$ ,  $U=4$ ,  $L=2$ )



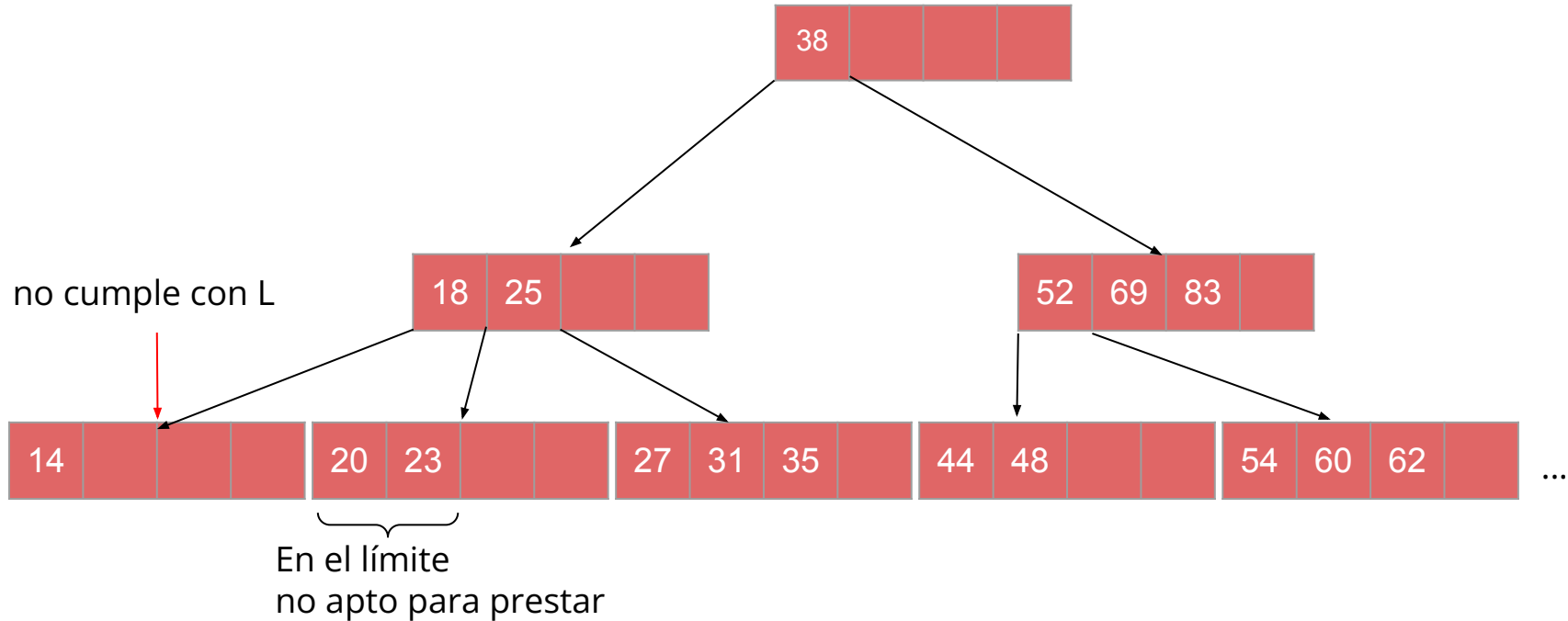
# Ejemplo 3- Borrado

Eliminar valor 17 (M=5, U=4, L=2)



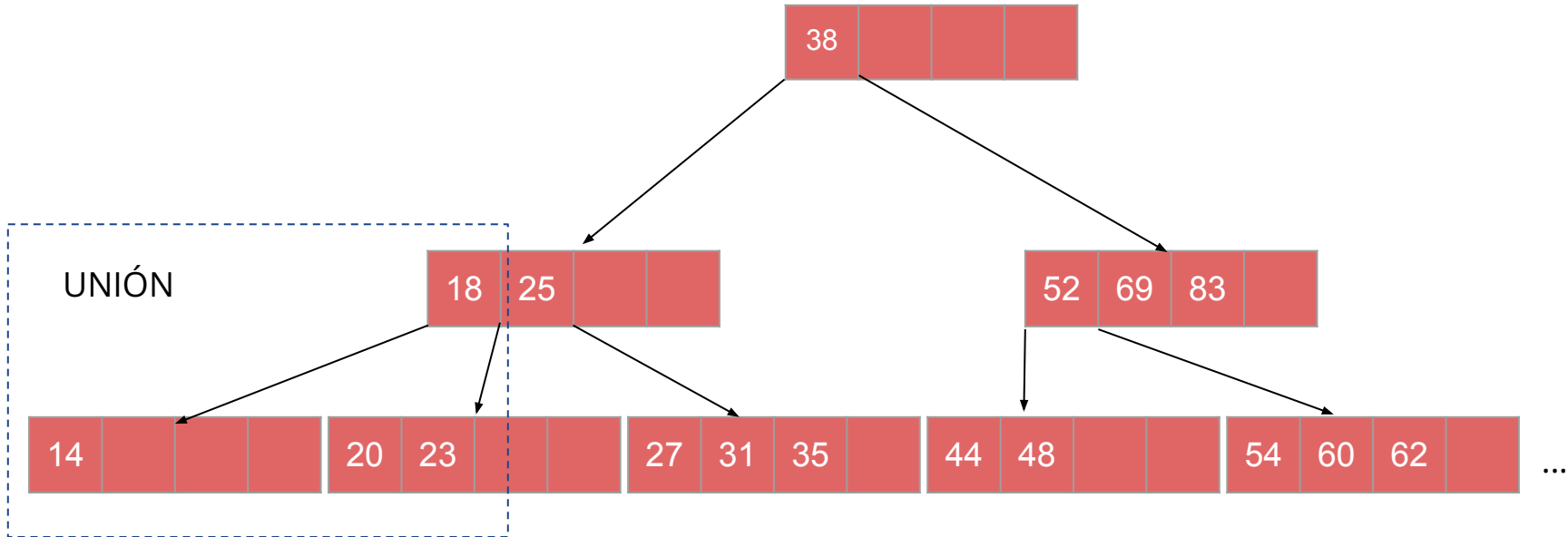
# Ejemplo 3- Borrado

Eliminar valor 17 ( $M=5$ ,  $U=4$ ,  $L=2$ )



# Ejemplo 3- Borrado

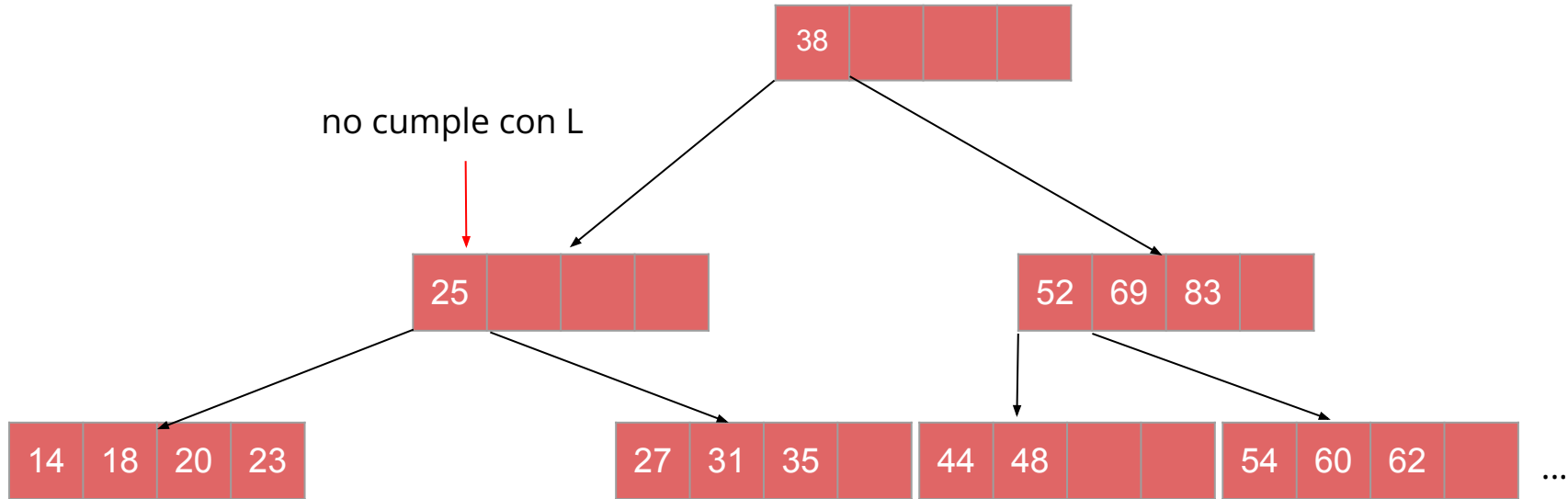
Eliminar valor 17 (M=5, U=4, L=2)





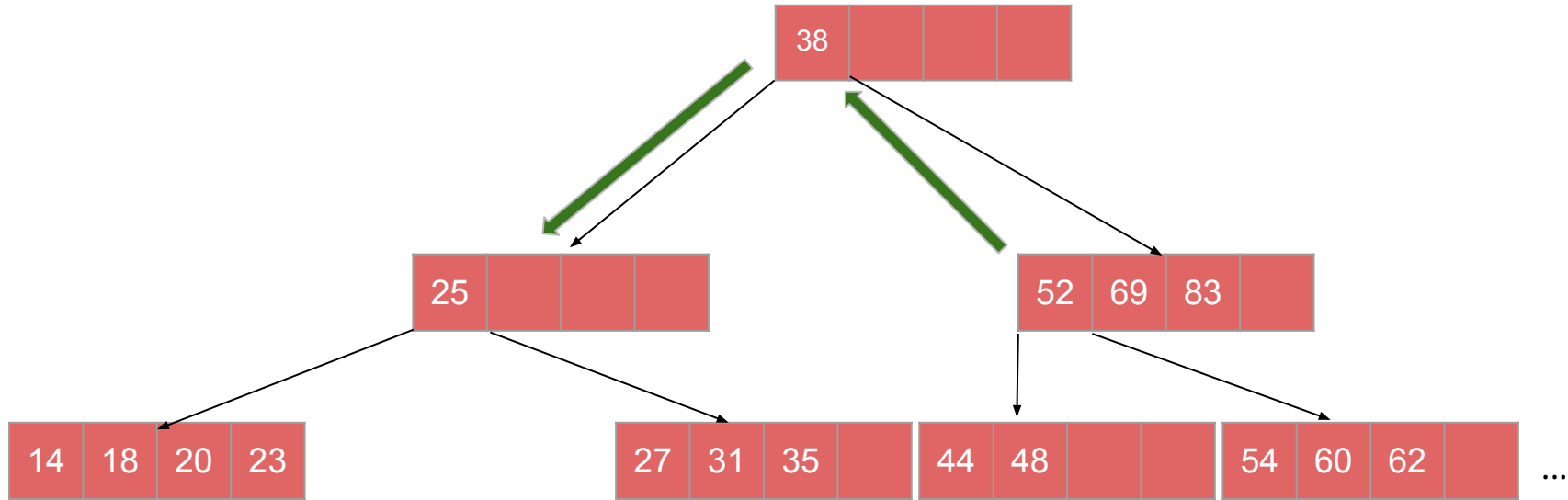
# Ejemplo 3- Borrado

Eliminar valor 17 ( $M=5$ ,  $U=4$ ,  $L=2$ )



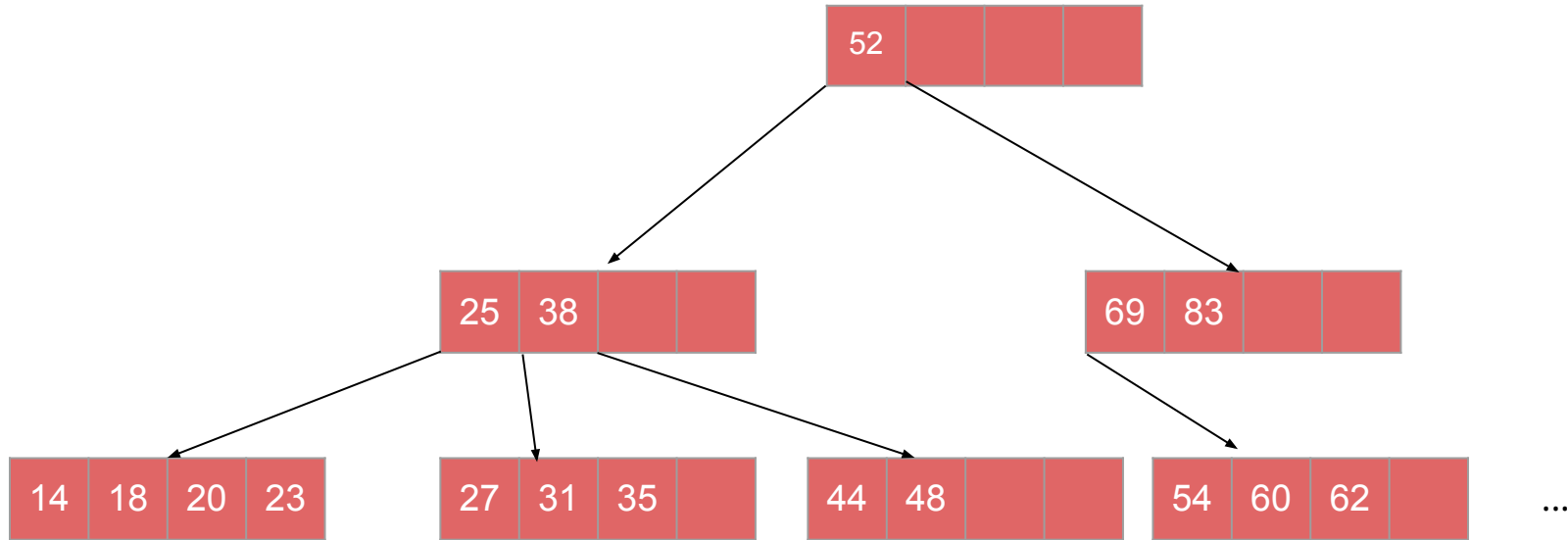
# Ejemplo 3- Borrado

Eliminar valor 17 (M=5, U=4, L=2)



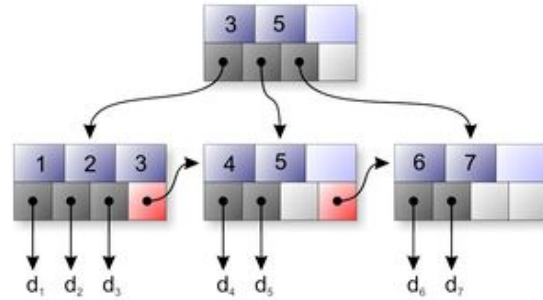
# Ejemplo 3- Borrado

Eliminar valor 17 (M=5, U=4, L=2)



# Ventajas

- Todos los nodos hoja están al mismo nivel, la recuperación de cualquier registro toma el mismo tiempo.
- Permanecen balanceados automáticamente.
- Proveen un buen desempeño para consultas exactas y por rangos.
- Inserciones, modificaciones y eliminaciones son eficientes, y se mantiene el orden de las claves para una recuperación rápida.



# Arbol B<sup>+</sup>

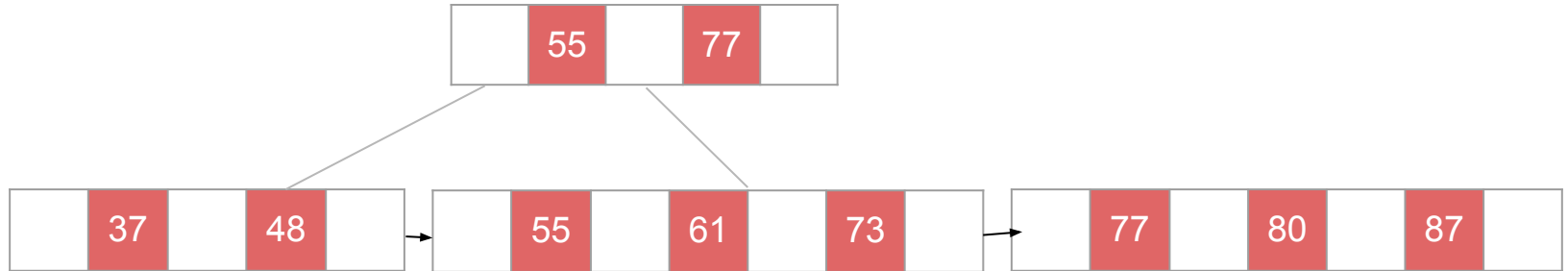
Ing. Max Alejandro Antonio Cerna  
Flores

# Agenda

- Definición
- Historia
- Características
- Operaciones
  - Búsqueda
  - Inserción
  - Borrado
- Implementaciones de árboles B y B+
- Ventajas y Desventajas

# Definición

Son una variante de los árboles B, se diferencian en que los árboles B+ toda la información se encuentra almacenada en las hojas.



# Historia

No hay artículo que presente el concepto de árbol B+.

La idea de mantener todos los datos en nodos hoja se menciona repetidamente como una variante interesante de los árboles B.

IBM utilizó árboles B+ por primera vez en 1973 en VSAM (Virtual Storage Access Method).



# Características

Maneja un **grado (M)** que indica el número máximo de hijos soportados en un nodo.

La raíz almacena como máximo  $M-1$  datos (claves).

Todos los nodos hoja tienen la misma altura.

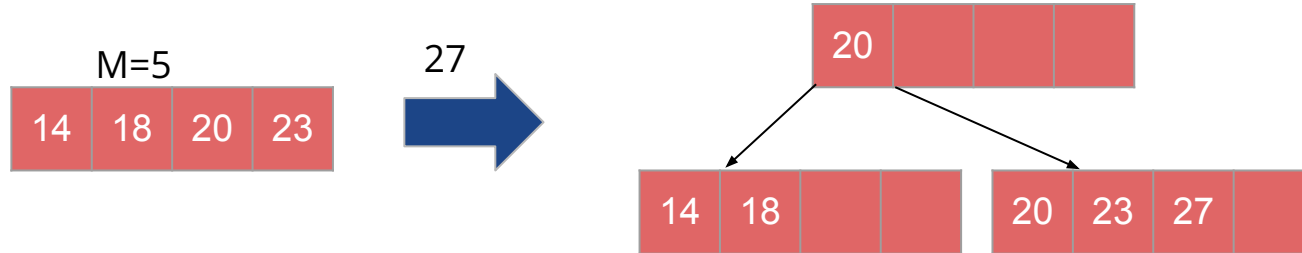
La información se encuentra ordenada.

**Los nodos raíz o interiores (no hojas) se utilizan como índices.**

# Inserción

Similar al árbol B.

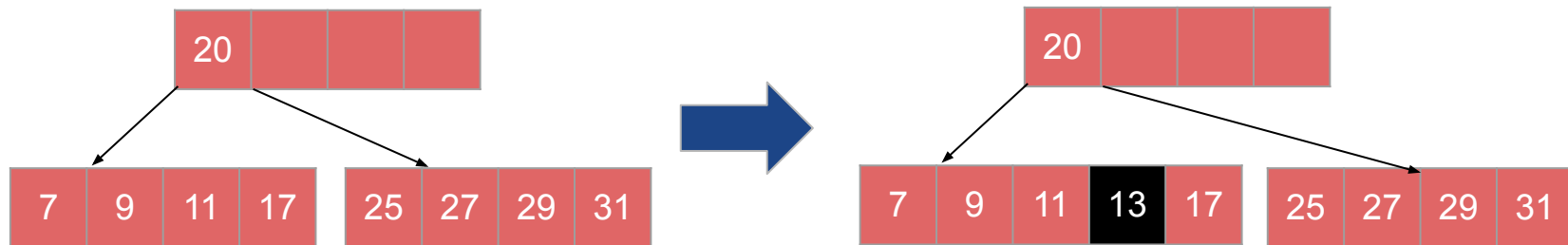
La principal diferencia ocurre cuando se inserta una key en un nodo que se encuentra lleno, se debe hacer una copia de la key que se sube al siguiente nivel en la hoja correspondiente.



# Inserción

M=5

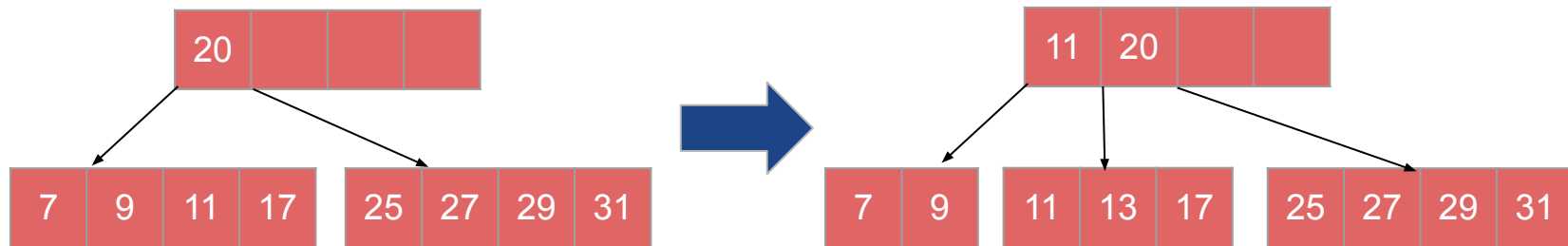
Insertar 13



# Inserción

M=5

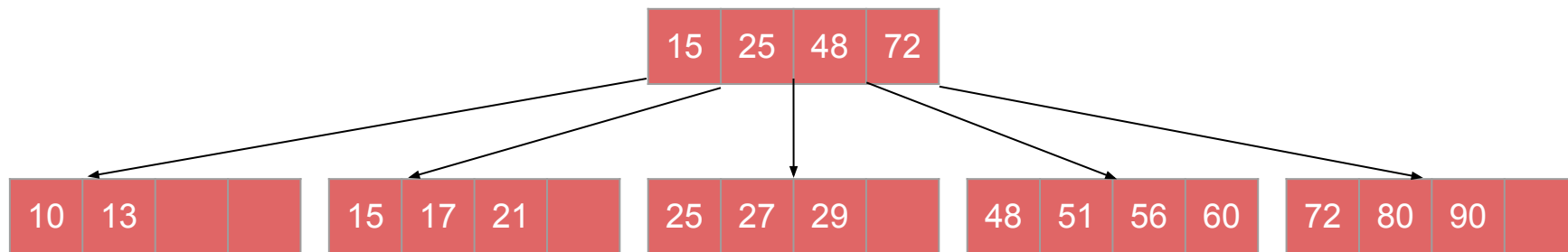
Insertar 13



# Inserción - Ejemplo 2

M=5

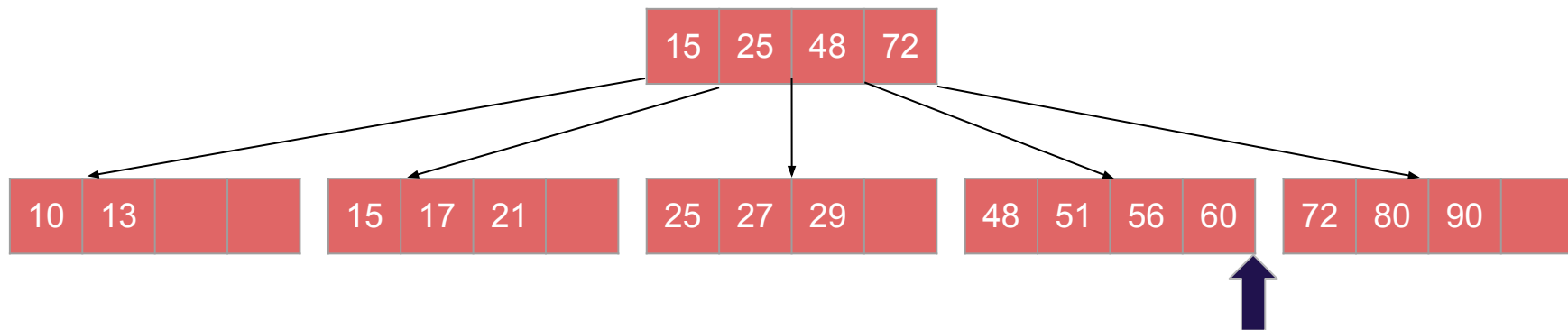
Insertar 67



# Inserción - Ejemplo 2

M=5

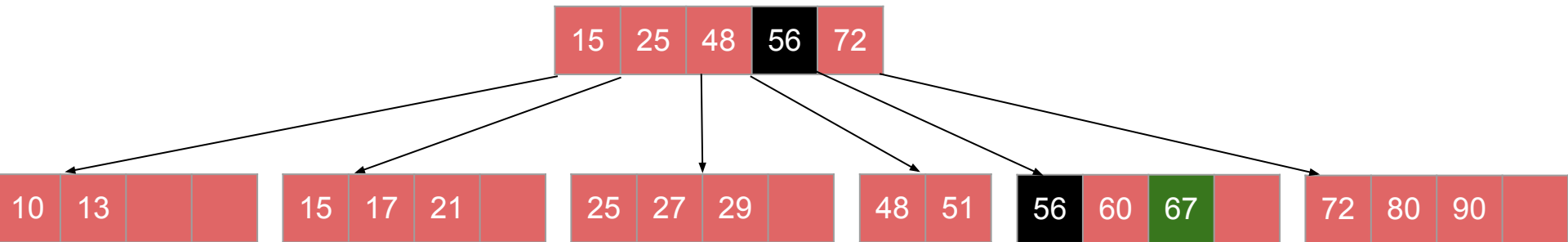
Insertar 67



# Inserción - Ejemplo 2

M=5

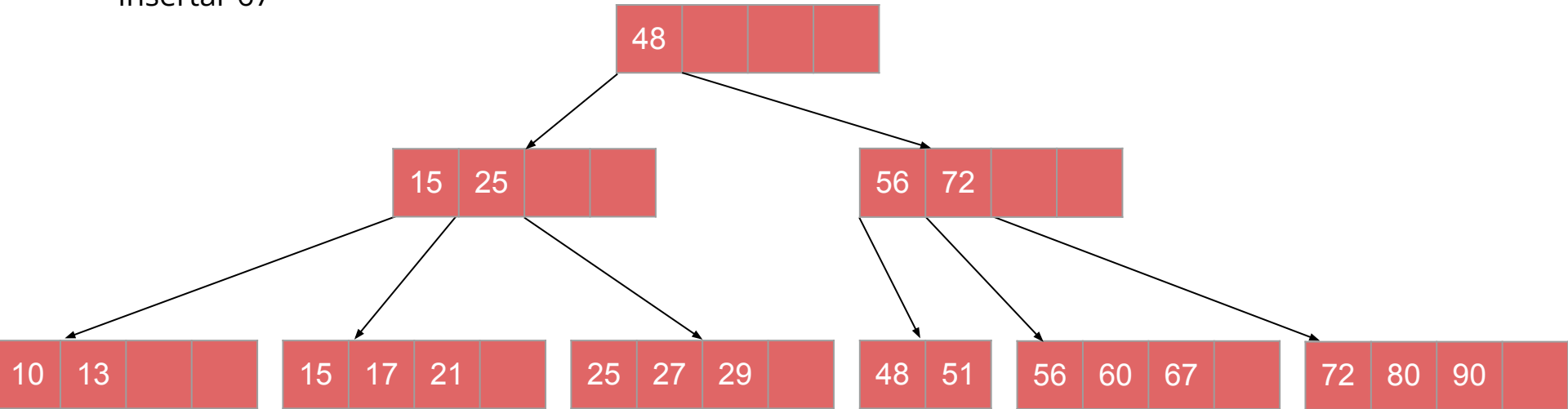
Insertar 67



# Inserción - Ejemplo 2

M=5

Insertar 67





# Eliminación

La eliminación es un proceso relativamente más sencillo que el de un árbol B.

El dato siempre está en el nodo hoja.

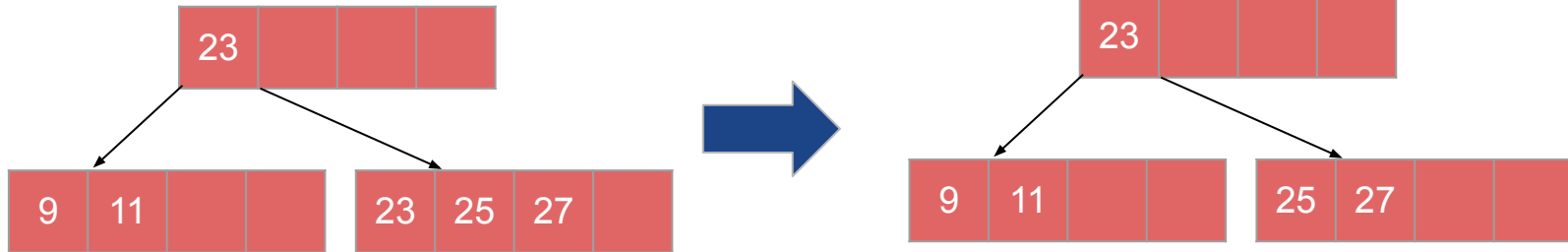
Si al borrar se conserva la regla de  $(M/2)-1$  claves, no se debe hacer ningún proceso adicional.

Si al borrar se rompe la regla del mínimo (lower), es necesario realizar una redistribución considerando las hojas y los índices.

# Eliminación - Ejemplo 1

M=5

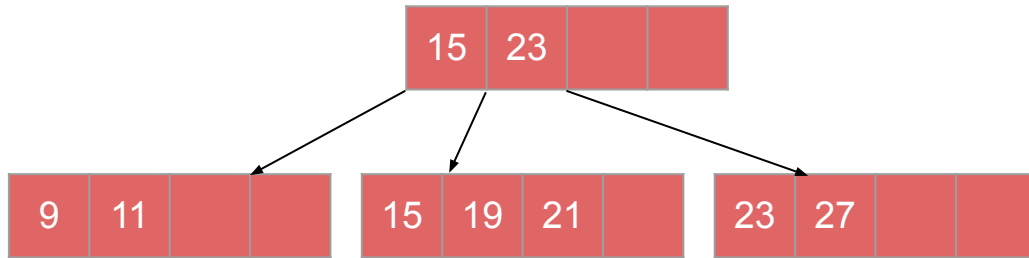
Eliminar 23



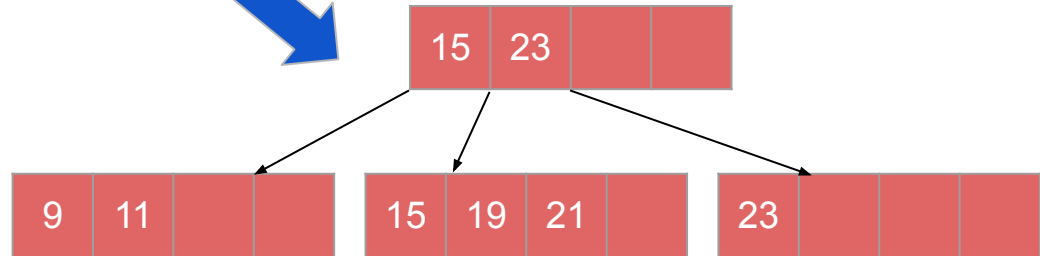
# Eliminación - Ejemplo 2

M=5

Eliminar 27



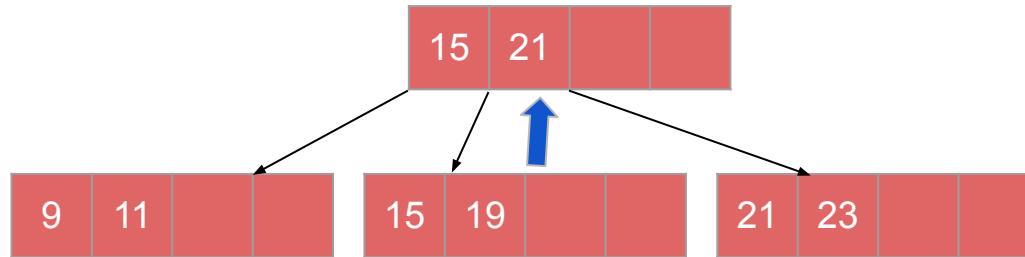
debemos prestar



# Eliminación - Ejemplo 2

M=5

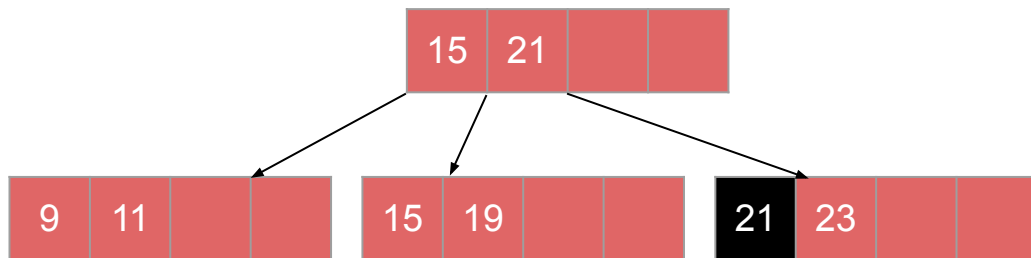
Eliminar 27



# Eliminación - Ejemplo 3

M=5

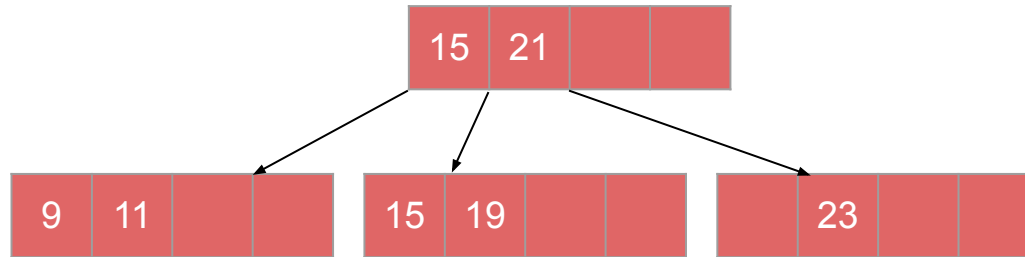
Eliminar 21



# Eliminación - Ejemplo 3

M=5

Eliminar 21

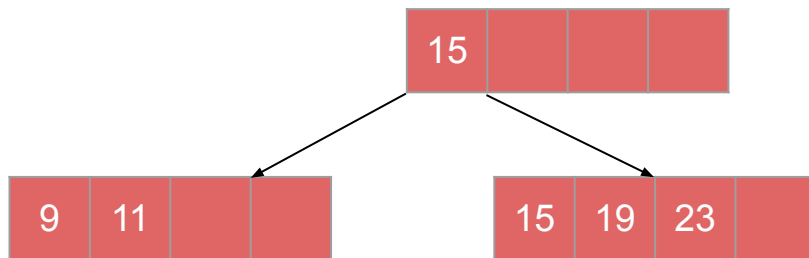


necesitamos fusionar

# Eliminación - Ejemplo 3

M=5

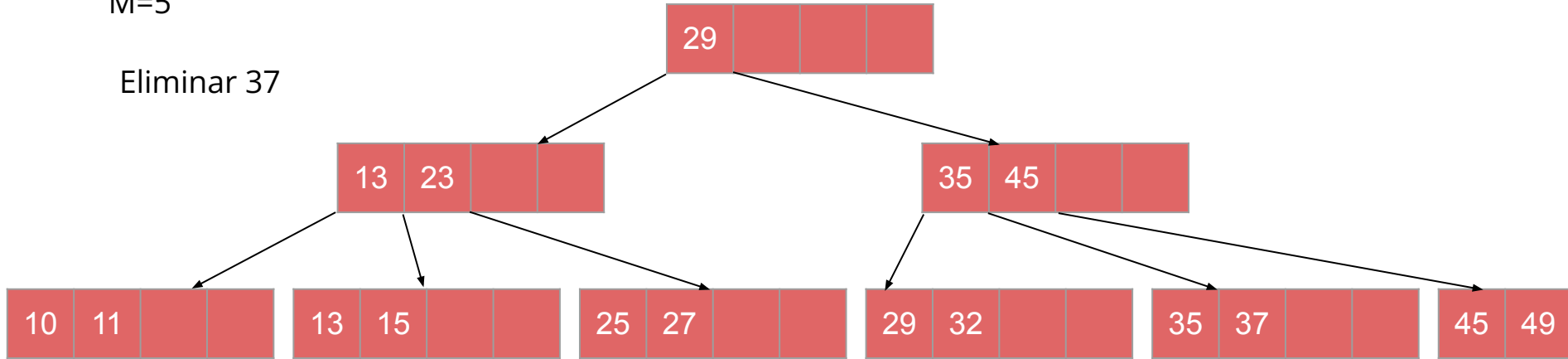
Eliminar 21



# Eliminación - Ejemplo 4

M=5

Eliminar 37

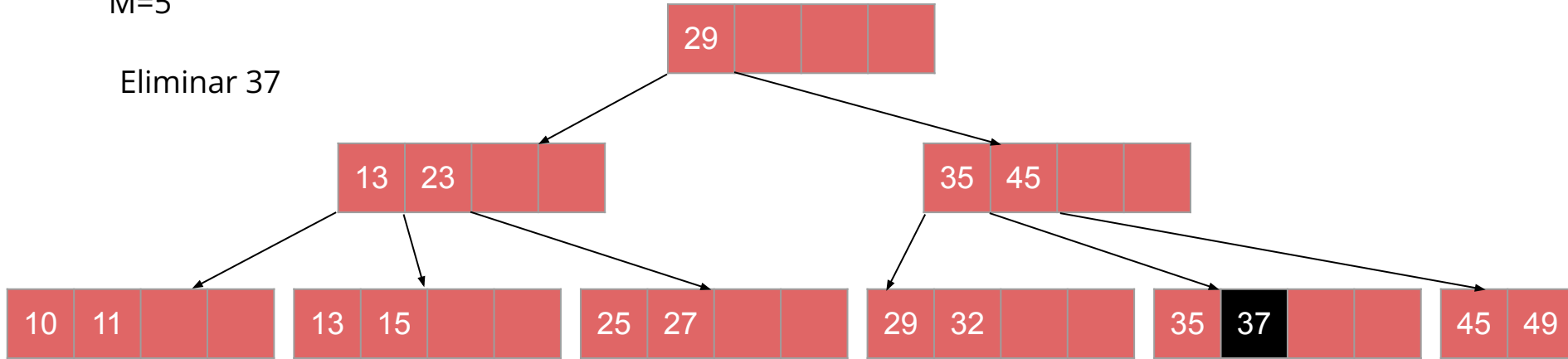




# Eliminación - Ejemplo 4

M=5

Eliminar 37



los vecinos no tienen suficientes  
datos para hacer un préstamo  
se necesita fusionar

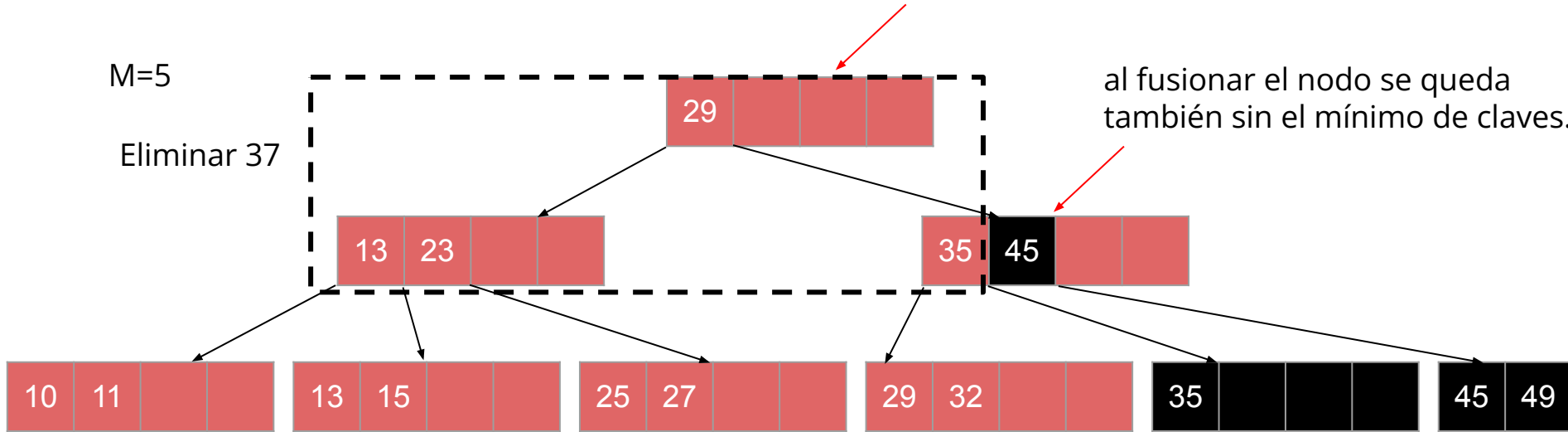
# Eliminación - Ejemplo 4

M=5

Eliminar 37

se necesita  
fusionar

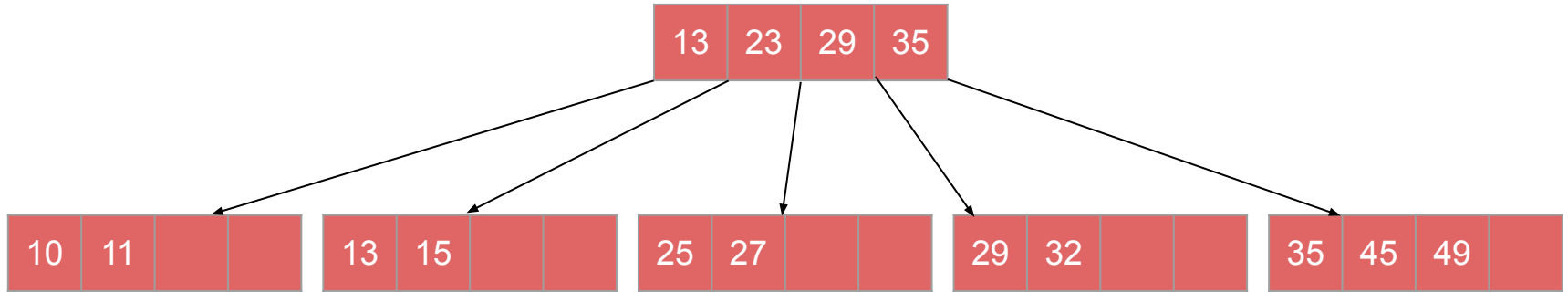
al fusionar el nodo se queda  
también sin el mínimo de claves.



# Eliminación - Ejemplo 4

M=5

Eliminar 37



# Nodos

*estructura NodoHoja {*

*numClaves: Entero;*

*valores: Objeto[];*

*claves: Entero[];*

*clavesUsadas: Entero;*

*nodolzq : NodoHoja;*

*nodoDer : NodoHoja;*

*}*

## **Constructor de un Nodo Hoja (PSEUDO CÓDIGO)**

*NodoHoja {*

*constructor NodoHoja(t) {*

*valores[(2\*t - 1)];*

*claves[(2\*t - 1)];*

*nodolzq = null;*

*nodoDer = null;*

*}*

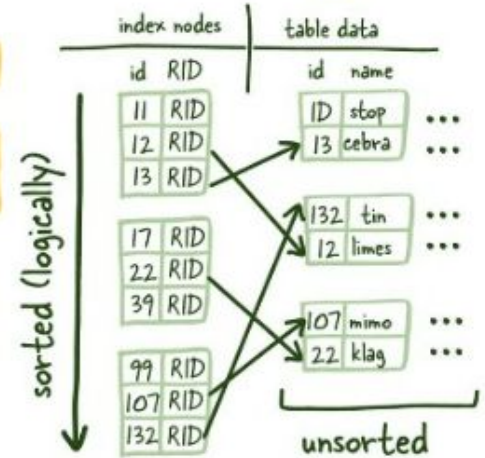
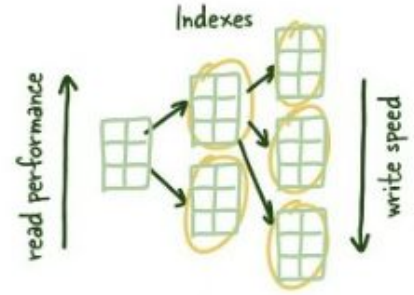
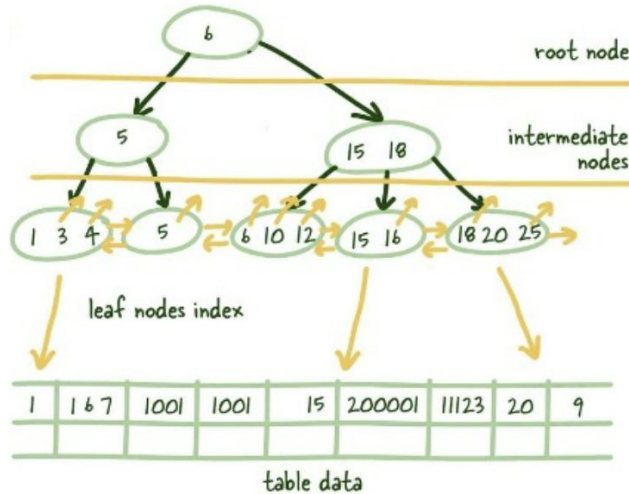
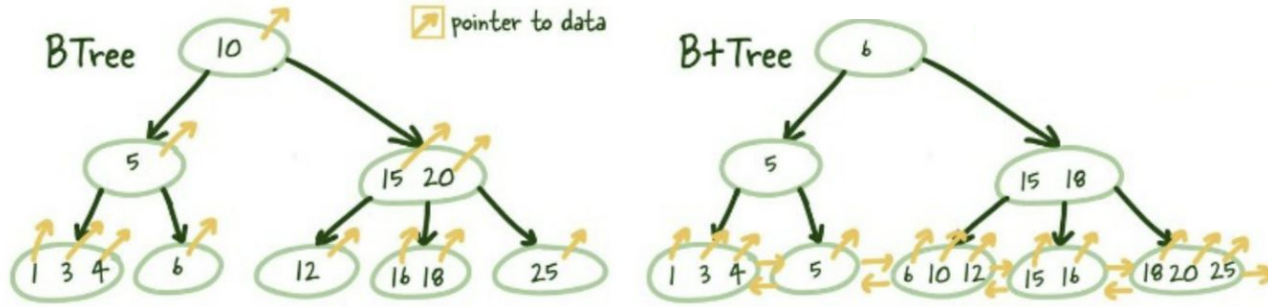
*}*

# Nodos

```
estructura NodoInterno {  
  
    numClaves: Entero;  
  
    claves: Entero[];  
  
    clavesUsadas: Entero;  
  
    nodoPadre : NodoInterno;  
  
    nodosHijo : NodoInterno[];  
  
}
```

```
Constructor de un Nodo Interno NodoInterno {  
  
    constructor NodoInterno(t) {  
  
        clave[(2*t - 1)];  
  
        nodoHijos[(2*t)];  
  
        clavesUsadas=0;  
  
        nodoPadre=null;  
  
    }  
  
}
```

# Implementación de Árboles B y B+



# Ventajas y Desventajas

Los nodos hoja del árbol están conectados entre sí a través de una lista enlazada.

Esto aumenta el coste de inserción.

Mejora la eficiencia en la búsqueda.

# Hoja de Trabajo

Implemente un árbol B+ que realice las siguientes operaciones:

Inserte: 10,17,3,29,4,5,18,6,22,1,33,35

Posteriormente

elimine: 18,33, 29

Grado: 5



# Cifrado de Llave Pública

Ing. Max Alejandro Antonio Cerna Flores

# Agenda

— — —

Definición

Casos de Uso

Cifrado

Firma Digital

Desventaja

# Definición

---

Es un sistema criptográfico en el que las claves vienen en pares.

Una clave (la clave privada) se mantiene secreta mientras que la otra se hace pública.

Es común que un sistema de cifrado utilice un algoritmo simétrico para cifrar el mensaje, y luego un sistema de clave pública para cifrar la clave simétrica.

# Casos de Uso

---

**Firmas digitales:** la clave privada se usa para firmar y la clave pública para verificar.

**Cifrado:** la clave pública se utiliza para cifrar y la clave privada se utiliza para descifrar.

Los sistemas de cifrado de clave pública más utilizados son RSA (para firma y cifrado), DSA (para firma) y Diffie-Hellman (para acuerdo de clave).

## CIFRADO

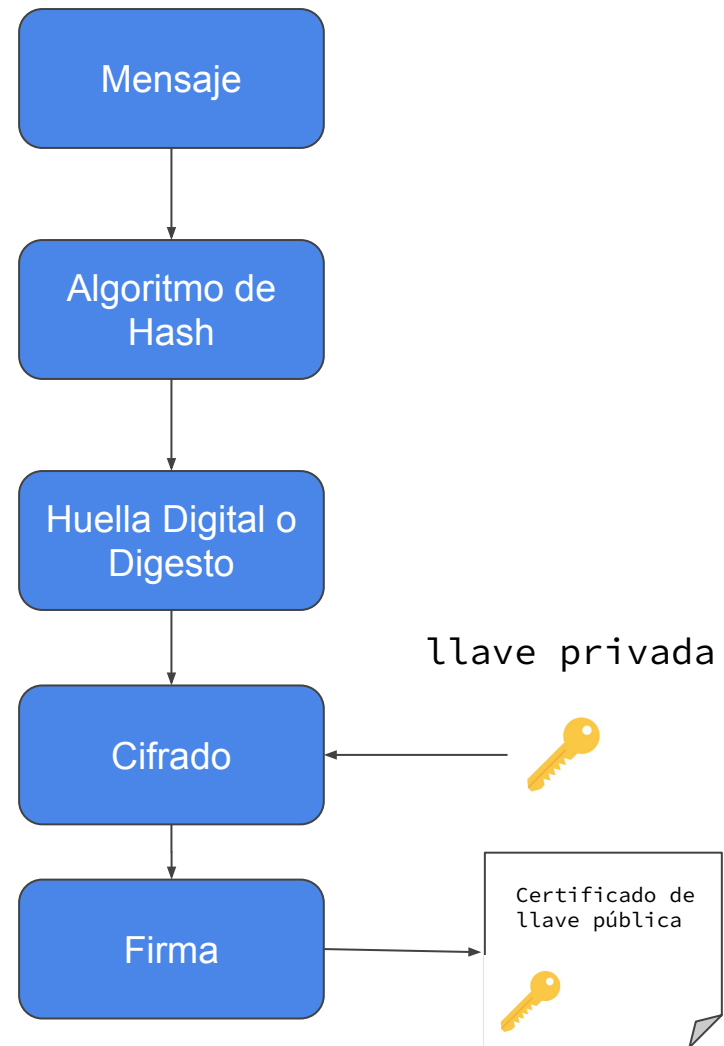
Cualquier persona con una clave pública puede cifrar un mensaje, generando un texto cifrado, pero solo aquellos que conocen la clave privada correspondiente pueden descifrar el texto cifrado para obtener el mensaje original.



## FIRMA DIGITAL

Un remitente puede usar una clave privada junto con un mensaje para crear una firma.

Cualquier persona con la clave pública correspondiente puede verificar si la firma coincide con el mensaje, pero un falsificador que no conoce la clave privada no puede encontrar ningún mensaje/firma que pase la verificación con la clave pública.



# Desventaja

---

Suelen ser mucho más lentos que los algoritmos simétricos y el tamaño del mensaje que pueden cifrar es proporcional al tamaño de la clave, por lo que no se ajustan bien a los mensajes largos.

La mala elección de un algoritmo de clave asimétrica o una longitud de clave demasiado corta conlleva un riesgo de seguridad que es que se conozca la clave privada de un par.

# Desventaja

---

Todos los esquemas de clave pública son, en teoría, susceptibles a un ataque de “fuerza bruta”.

Sin embargo, tal ataque no es práctico si la cantidad de computación necesaria para tener éxito, denominada "factor de trabajo" por Claude Shannon, está fuera del alcance de todos los atacantes potenciales.

Se puede aumentar la seguridad simplemente eligiendo una clave más larga.



# Algoritmos de Cifrado de Llave Pública

— — —

Diffie-Hellman

RSA

DSA

SSL/TLS

Cifrado ElGamal

Cripto Sistema Kramer-Shoup

Funciones Hash

Curvas Elípticas

# RSA

---

Significa Rivest, Shamir y Adleman quienes primero lo describieron públicamente.

Se conoce como el primer algoritmo apropiado tanto para firmar como para cifrar, y fue el primer gran avance en criptografía de llave pública.

Es utilizado extensamente en protocolos de comercio electrónico y se considera seguro dado a la longitud suficiente de las claves y del uso de implementaciones actualizadas.

# DSA

---

Es un estándar para firmas digitales.

Un estándar del gobierno federal de los Estados Unidos para firmas digitales.

Es para firmas únicamente y no es un algoritmo de cifrado.

# TLS/SSL

---

Transport Layer Security (TLS) y su predecesor, Secure Sockets Layer (SSL), son protocolos criptográficos que ofrecen seguridad para comunicaciones en redes como la Internet.

TLS y SSL cifran los segmentos de conexiones de redes en la capa de transporte de extremo a extremo.

# Cifrado ElGamal

---

Es un algoritmo de cifrado de llave asimétrico para criptografía de llave pública basado en el acuerdo de llave Diffie-Hellman.

Fue descrito por Taher Elgamal en 1985.

Se utiliza en software libre de Guardián de Privacidad, GNU, versiones recientes de PGP, y otros cripto-sistemas.

# Cripto Sistema Kramer-Shoup

---

Es un algoritmo de llave asimétrica y fue el primer esquema eficiente y seguro demostrado contra el ataque de texto de cifrado mediante supuestos estándares criptográficos.

Desarrollado por Ronald Cramer y Victor Shoup en 1998, es una extensión del criptosistema Elgamal.

# Funciones Hash

---

Las funciones Hash tienen gran importancia, ya que se enfocan principalmente a solventar los problemas de la integridad de los mensajes, así como la autenticidad tanto de mensajes como de su origen.

Son funciones matemáticamente que se realizan el resumen de un documento a firmar, de manera que para ellos comprimen el documento en un único bloque de longitud fija, bloque cuyo contenido resulta ilegible y no tiene ningún sentido real.

Ejemplo: SHA-1, MD5, SHA-256

# Curvas Elípticas

---

Es un tipo específico de criptografía de clave pública, que utiliza diferentes problemas matemáticos para generar una clave pública y otra privada con las que realizar operaciones. En el caso particular de las curvas elípticas, su utilización en algoritmos criptográficos fue propuesta por primera vez en 1985 por Victor Miller y Neil Koblitz.

En la actualidad, podemos encontrar curvas elípticas en protocolos como Bitcoin o Ethereum mediante el uso del algoritmo de firma digital ECDSA.



# **RSA**

## **(Rivest, Shamir y Adleman)**

**Ing. Max Alejandro Antonio Cerna Flores**

# Agenda

— — —

Definición

Seguridad

Factorización de Números Enteros Grandes

El problema RSA

Proceso de Generación de claves

Proceso de Cifrado y Descifrado

# Definición

---

El nombre RSA proviene de las iniciales de sus tres creadores, Rivest, Shamir y Adleman, allá por 1977.

Es un sistema criptográfico de clave pública, utiliza factorización de números primos enteros. Es el primer y más utilizado algoritmo de este tipo y es válido tanto para cifrar como para firmar digitalmente.

El funcionamiento se basa en el producto, conocido, de dos números primos grandes elegidos al azar y mantenidos en secreto.

# Seguridad

---

Está basado en dos problemas matemáticos: el problema de factorizar números grandes y el problema RSA.

Si el valor  $N$  es lo suficientemente grande el algoritmo RSA es seguro. Si  $N$  tiene 256 bits o menos, puede ser factorizado en pocas horas con un ordenador personal.

# Factorización de Números Enteros Grandes

---

Consiste en descomponer un número compuesto entero positivo no primo que tiene una única descomposición en números primos o factores primos.

Cuando los números son muy grandes no se conoce ningún algoritmo que resuelva eficientemente este problema.

Los números primos son aquellos que solo son divisibles entre 1 y ellos mismos.

# Factorización de Números Enteros Grandes

— — —

2	3	5	7	11	13	17	19	23	29	31	37	41	43	47	53	59	61	67
71	73	79	83	89	97	101	103	107	109	113	127	131	137	139	149	151	157	163
167	173	179	181	191	193	197	199	211	223	227	229	233	239	241	251	257	263	269
271	277	281	283	293	307	311	313	317	331	337	347	349	353	359	367	373	379	383
389	397	401	409	419	421	431	433	439	443	449	457	461	463	467	479	487	491	499
503	509	521	523	541	547	557	563	569	571	577	587	593	599	601	607	613	617	619
631	641	643	647	653	659	661	673	677	683	691	701	709	719	727	733	739	743	751
757	761	769	773	787	797	809	811	821	823	827	829	839	853	857	859	863	877	881
883	887	907	911	919	929	937	941	947	953	967	971	977	983	991	997			

# Factorización de Números Enteros Grandes

---

## *Números Coprimos*

Son dos números enteros “a” y “b” que no tienen ningún factor primo en común.

O sea son coprimos, si y sólo si, su máximo común divisor (MCD) es igual a 1.

# Factorización de Números Enteros Grandes

---

## *Ejemplos*

10 y 12 **no** son coprimos ya que el 2 es su divisor en común.

2 y 3 son coprimos porque el máximo común divisor es 1.

67 tiene de divisores 1,67

12 tiene de divisores 1,2,3,4,6,12

67 y 12 solo tienen el 1 en común por tanto son coprimos.



# Problema RSA

---

Se refiere a la dificultad de efectuar una operación de clave privada mediante el sistema criptográfico RSA conociendo tan solo la clave pública.

Para números suficientemente grandes mayores de 1024 bits no se conoce un método eficiente de factorización.

# Proceso

---

Generación de claves:

1. Seleccionar dos números primos **p** y **q** (ej:  $p=3$ ,  $q=11$ )
2. Calcular **n** =  $p * q$  (ej:  $n = 3 * 11 = 33$ )
3. Calcular **z** =  $(p-1) * (q-1)$  (ej:  $z = (3-1)*(11-1) = 20$ )
4. Se elige un número primo **k**, tal que k sea coprimo a z.

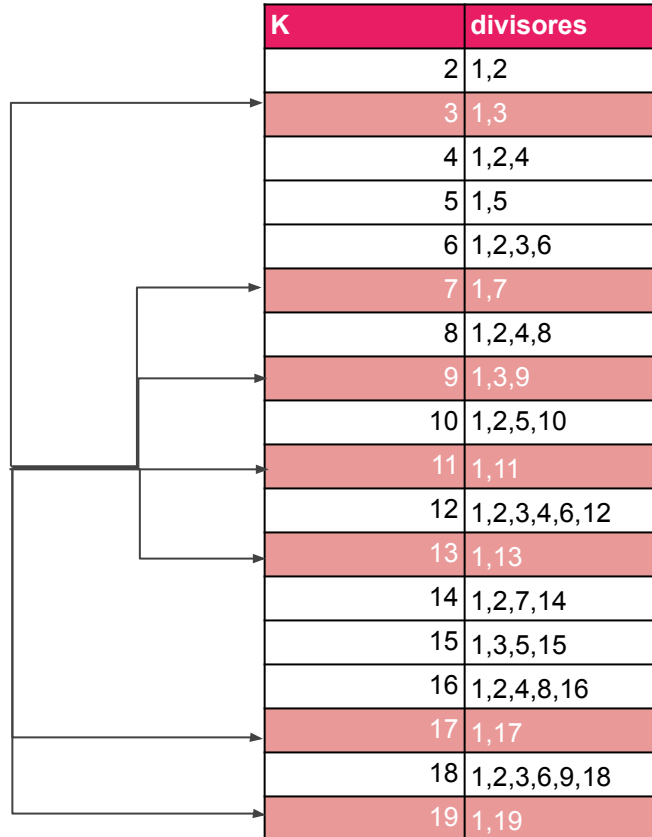
$$\text{MCD}(k, z) = 1$$

$$1 < k < z$$

# Proceso

---

Coprimos



K	divisores
2	1,2
3	1,3
4	1,2,4
5	1,5
6	1,2,3,6
7	1,7
8	1,2,4,8
9	1,3,9
10	1,2,5,10
11	1,11
12	1,2,3,4,6,12
13	1,13
14	1,2,7,14
15	1,3,5,15
16	1,2,4,8,16
17	1,17
18	1,2,3,6,9,18
19	1,19

$$z = 20 = \{1,2,4,5,10\}$$

K posibles =>

3, 7, 11, 13, 17 o 19

# Proceso

---

Generación de claves:

5. clave pública (n,k) (ej: {n=33,k=7})
6. calcular clave privada (j)

donde j es un número entero grande que debe cumplir con:

$$(k*j) \bmod z = 1$$

$$k = 7$$

$$z = 20$$

# Proceso

— — —

$$k = 7$$

$$z = 20$$

j	k*j	(k*j)mod z
1	7	7
2	14	14
3	21	1
4	28	8
5	35	15
6	42	2
7	49	9
8	56	16
9	63	3
10	70	10
11	77	17
12	84	4
13	91	11
14	98	18
15	105	5
16	112	12
17	119	19

$$(k*j) \bmod z = 1$$

$$j = 3$$

# Cifrado

---

Fórmula de cifrado (emisor):

$$C = M^k \bmod n$$

clave pública (n,k) ( $\{n=33, k=7\}$ ) y M es el mensaje a enviar,  
por ejemplo:

$$M = 17$$

$$C = 17^7 \bmod 33$$

$$C = 8$$

# Descifrado

---

Fórmula de descifrado (receptor):

$$M = C^j \bmod n$$

clave privada  $\{n=33, j=3\}$  y mensaje cifrado  $C=8$ :

$$M = 8^3 \bmod 33$$

$$C = 17$$



# Protocolo Diffie-Hellman

Ing. Max Alejandro Antonio Cerna Flores



# Agenda

## Protocolo Diffie-Hellman

Historia

Descripción básica

Complejidad de Logaritmo Discreto

¿Cómo funciona?

Ataques de hombre en medio (MiTM)

Solución a ataques MiTM

# Historia

El intercambio de claves Diffie-Hellman debe su nombre a sus creadores Whitfield Diffie y Martin Hellman, publicaron su artículo **New Directions in Cryptography** en 1976.

Permite acordar una clave secreta entre dos máquinas, a través de un canal inseguro y enviando únicamente dos mensajes.

Actualmente se conoce que es vulnerable a ataques de hombre en medio.

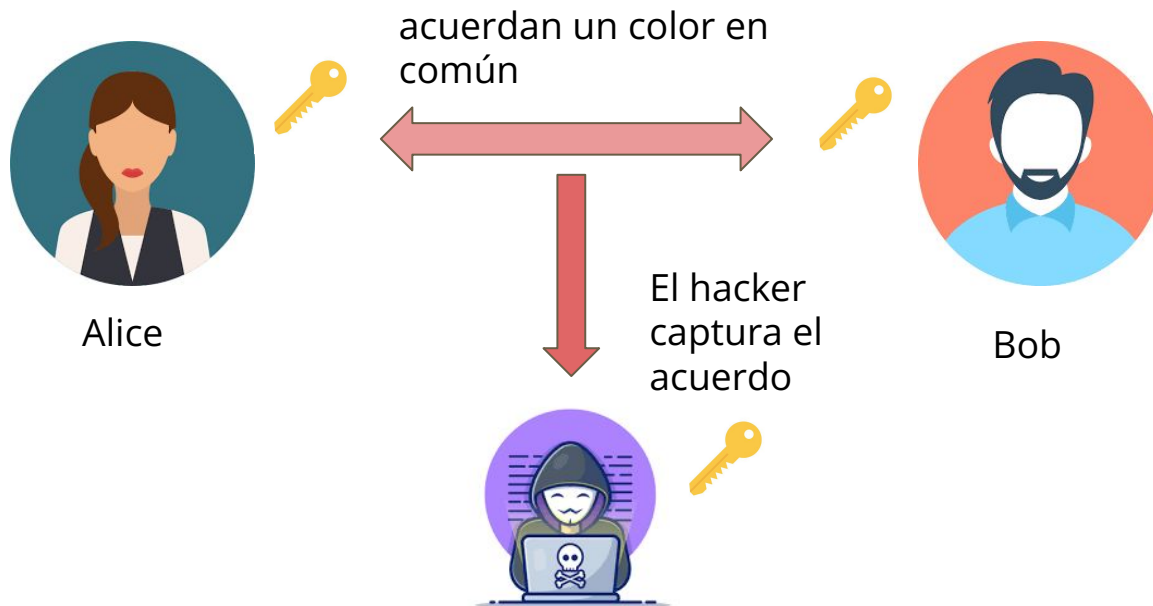
Whitfield Diffie y Martin Hellman recibieron el premio A.M. Turing de 2015 de la Association for Computer Machinery en 2016.

<https://awards.acm.org/about/2015-turing>

# Descripción Básica



# Descripción Básica



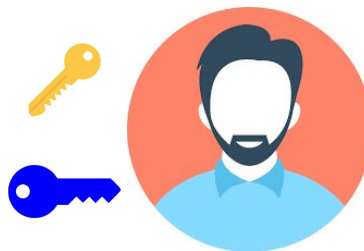
# Descripción Básica



Alice



Bob y Alice escogen  
un nuevo segundo  
color el cual solo  
ellos conocen



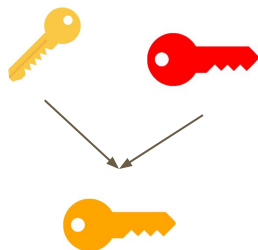
Bob



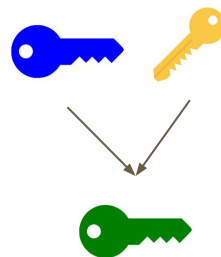
# Descripción Básica



Alice



Mezclan la común  
con la secreta  
generando un  
nuevo color



Bob



# Descripción Básica



Alice



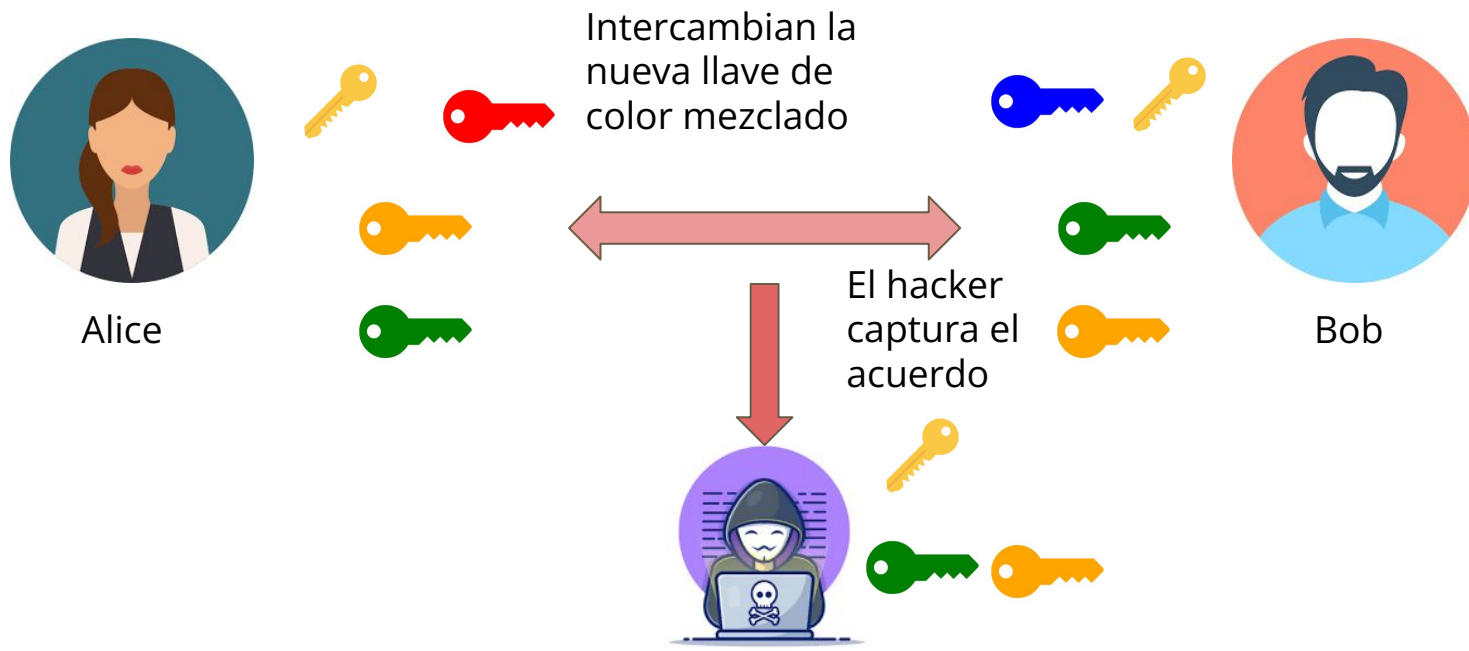
Intercambian la  
nueva llave de  
color mezclado



Bob

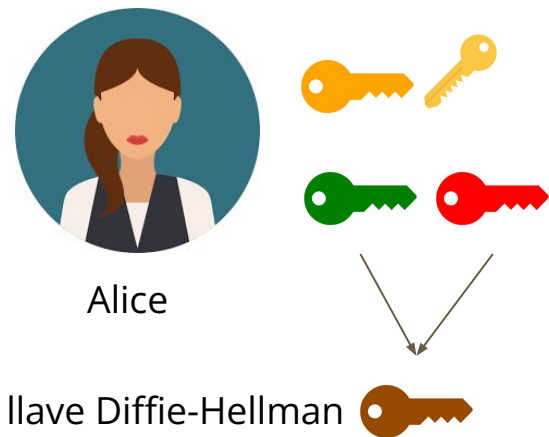


# Descripción Básica

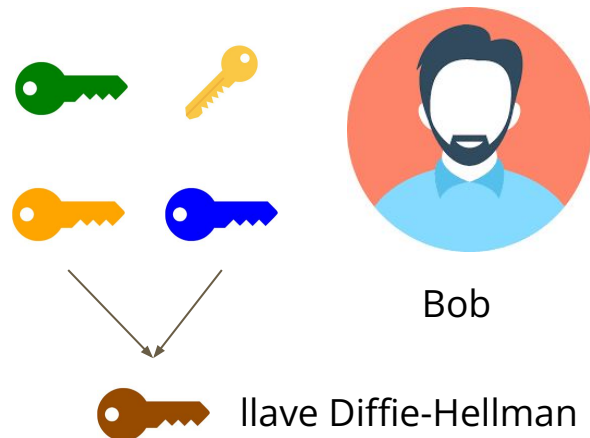




# Descripción Básica



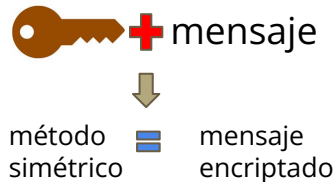
se mezclan la última llave compartida entre ellos con la llave privada de cada uno y se genera una llave igual para ambas partes



# Descripción Básica



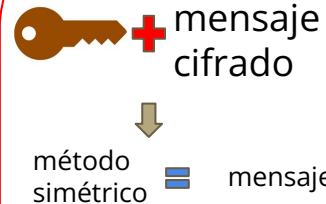
Alice



llaves públicas



llave privada



Bob

llaves públicas



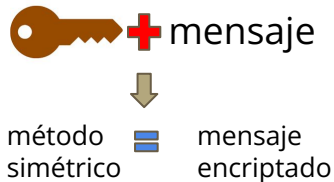
llave privada



# Descripción Básica



Alice



llaves públicas



llave privada



Hacker captura el mensaje cifrado e intenta descifrar usando las llaves.



Bob

llaves públicas



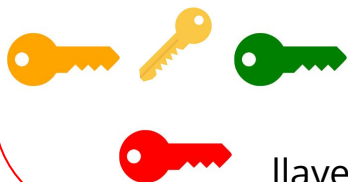
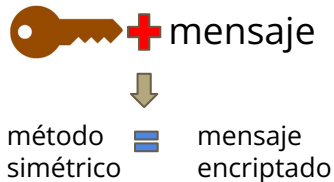
llave privada



# Descripción Básica



Alice

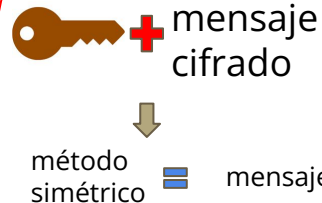


llaves públicas

llave privada



Sin las llaves privadas no puede generar la clave Diffie-Hellman



Bob



llaves públicas

llave privada

# Complejidad de Logaritmo Discreto

La seguridad del algoritmo creado por Whitfield Diffie y Martin Hellman se basa en la dificultad de solucionar el problema del **logaritmo discreto**.

El algoritmo se basa en las propiedades de la **exponenciación modular**.

La exponenciación modular es un tipo de exponenciación realizada sobre un módulo.

$$A = k^a \bmod p$$

# Complejidad de Logaritmo Discreto

**¿Cual es el problema del logaritmo discreto?**

Conociendo  $k, a$  y  $p$  (este último siendo un número primo) y realizando exponenciación modular:

$$A = k^a \bmod p$$

es muy fácil calcular  $A$ , supongamos  $k=3, a=2$  y  $p = 11$  entonces:

$$A = 3^2 \bmod 11 = 9$$

# Complejidad de Logaritmo Discreto

¿Cual es el problema del logaritmo discreto?

Pero, qué pasaría si conocemos  $A$ ,  $k$  y  $p$  pero desconocemos  $a$ :

$$A = k^a \bmod p$$

entonces:

$$a = \log_k A \bmod p$$

# Complejidad de Logaritmo Discreto

¿Cual es el problema del logaritmo discreto?

Calcular  $\log_k A$  **SIMPLE**

Al resultado anterior aplicarle  $\text{mod } p$  **MUY COMPLEJO**

Esto se puede notar a mayor grado conforme los números utilizados son más grandes.



# ¿Cómo funciona?



Alice



Bob

acuerdan clave pública  
( $k, p$ ) en común

$p$  es un número primo  
muy grande.

$k$  es un número menor  
que  $p$  ( $0 < k < p$ )

Supongamos  $p=17$  y  $k=4$

$$A = k^a \bmod p$$

# ¿Cómo funciona?

Alice calcula su llave privada  $a$



$a$  es un número menor que  $p$   
( $0 < a < p$ )



Alice

(4,17)

$$A = k^a \bmod p$$

Supongamos  $a=3$  y  $b=6$

(4,17)



Bob

Bob calcula su llave privada  $b$



$b$  es un número menor que  $p$   
( $0 < b < p$ )

$$B = k^b \bmod p$$

# ¿Cómo funciona?

Calculamos A  
sabiendo

$a=3$   
 $k=4$   
 $p=17$



Alice

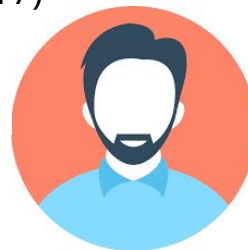
$(4,17)$



$$A = 4^3 \bmod 17$$

$$A = 13$$
 

$(4,17)$



Bob

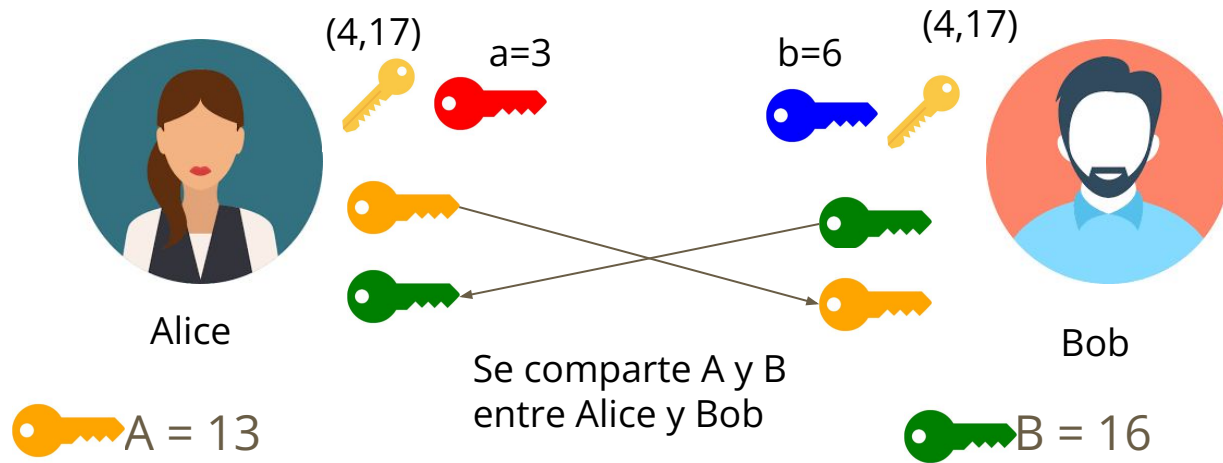
Calculamos B  
sabiendo

$b=6$   
 $k=4$   
 $p=17$

$$B = 4^6 \bmod 17$$


$$B = 16$$
 

# ¿Cómo funciona?



# ¿Cómo funciona?

(4,17)  $a=3$



Alice

Alice is represented by a circular icon of a woman with brown hair. To her right are four keys: a yellow key, a red key, an orange key, and a green key. Above the keys is the text '(4,17)' and 'a=3'.

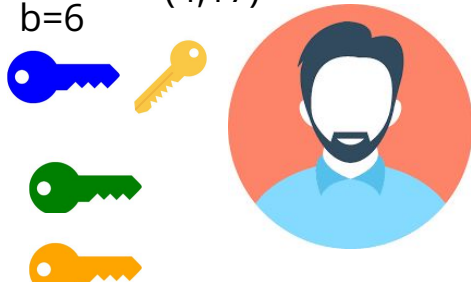
  $B = 16$    $a=3$

$K = B^a \bmod p$



$K = 16^3 \bmod 17 = 16$

(4,17)  $b=6$



Bob is represented by a circular icon of a man with a beard. To his left are four keys: a blue key, a yellow key, a green key, and an orange key. Above the keys is the text '(4,17)' and 'b=6'.

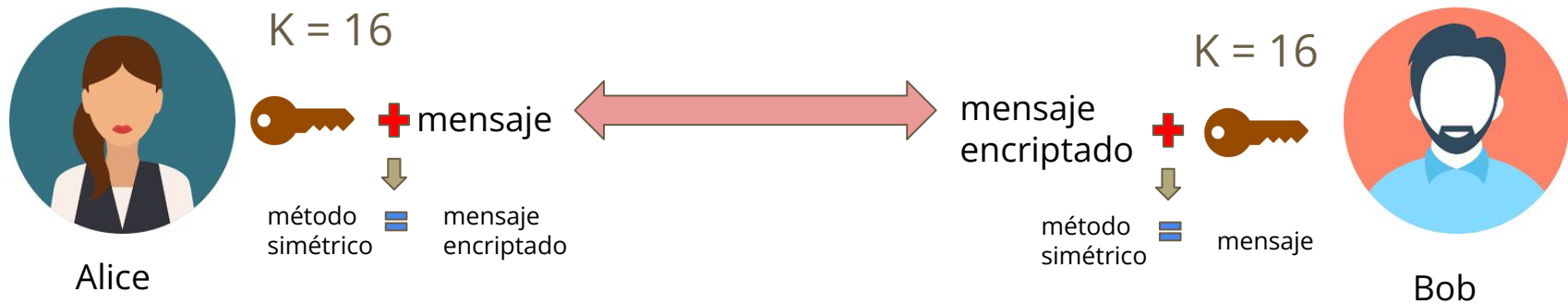
  $A = 13$    $b=6$

$K = A^b \bmod p$



$K = 13^6 \bmod 17 = 16$

# ¿Cómo funciona?



# Ataques de Hombre en Medio (MiTM)

El atacante puede ser un oyente pasivo en tu conversación robando los mensajes mientras escucha, o un participante activo, alterando el mensaje.

En el protocolo Diffie-Hellman un atacante podría situarse entre ambas máquinas y acordar una clave simétrica con cada una de las partes.

Una vez establecidas las 2 claves simétricas, el atacante haría de puente entre los 2 hosts, descifrando toda la comunicación y volviéndola a cifrar para enviársela al otro host.

# Solución a los ataques MiTM

Diffie-Hellman no proporciona autenticación, no tiene forma de verificar si la otra parte en una conexión es realmente quien dice ser.

La autenticación del mensaje es necesaria para evitar que intermediarios utilicen un ataque hombre en medio.

Generalmente se implementa junto con algunos medios de autenticación. Esto a menudo implica el uso de certificados digitales y un algoritmo de clave pública.



# Firma Digital

**Ing. Max Alejandro Antonio Cerna Flores**

# Agenda

— — —

Definición

Proceso

Componentes de la Firma Digital

Casos de Uso

# Definición

---

Es un esquema matemático para verificar la autenticidad de mensajes o documentos digitales.

Válida cuando se cumplen los requisitos previos.

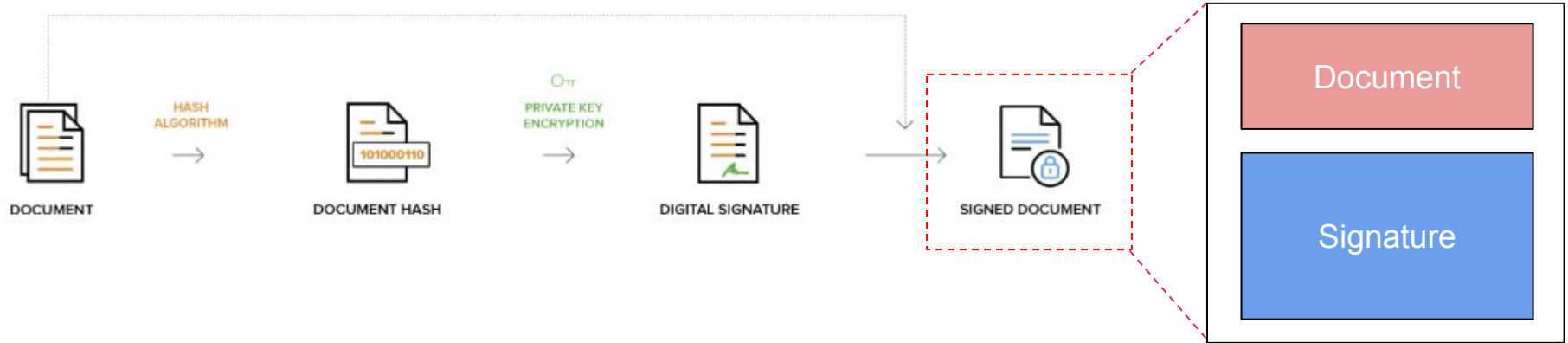
Otorga al destinatario una gran confianza en que el mensaje fue creado por un remitente conocido.

Válida que el mensaje no se modificó durante el tránsito.

# Proceso

---

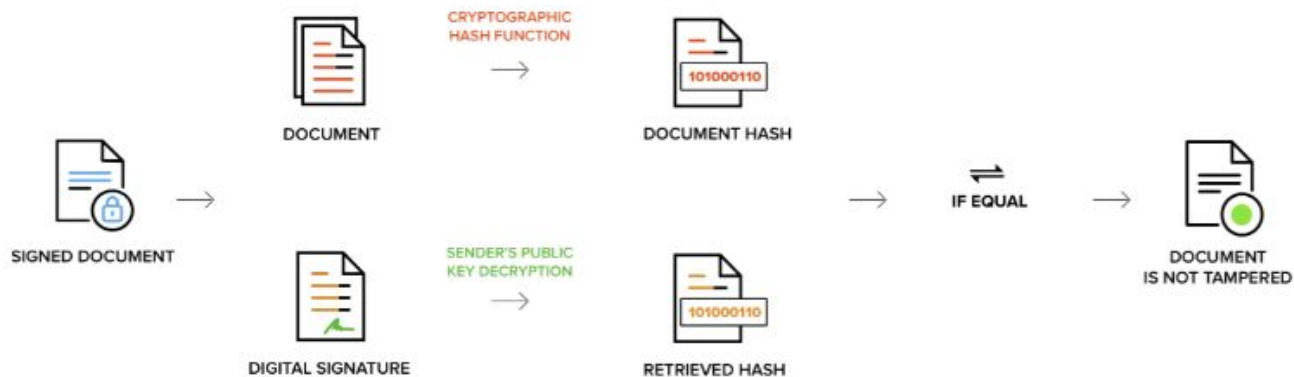
## Generación de firma



# Proceso

---

## Validación de firma

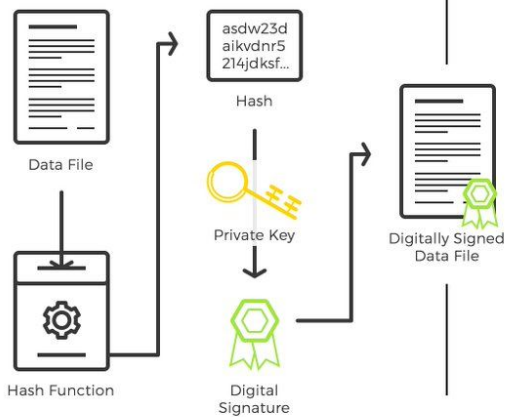


# Proceso

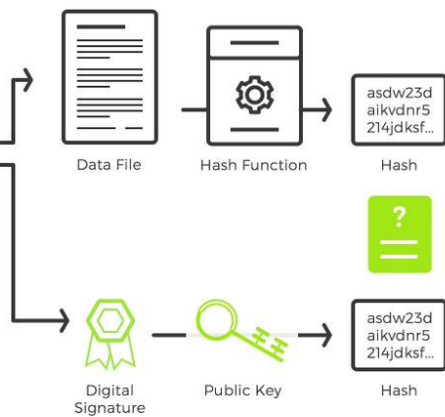
— — —

## Common Public Key Digital Signature

### Signing



### Verification



# Componentes de la Firma Digital

---

Un esquema de firma digital normalmente consta de tres algoritmos:

- 1) Un algoritmo de generación de claves que selecciona una clave privada uniformemente al azar de un conjunto de posibles claves privadas.

El algoritmo genera la clave privada y una clave pública correspondiente.

# Componentes de la Firma Digital

---

- 2) Un algoritmo de firma que dado el mensaje y una clave privada, produce una firma.
- 3) Un algoritmo de verificación de firma que dado el mensaje, la clave pública y la firma, acepta o rechaza la afirmación de autenticidad del mensaje.



## Casos de Uso

### *Autenticación*

Cuando la propiedad de una clave secreta de firma digital está vinculada a un usuario específico, una firma válida muestra que el mensaje fue enviado por ese usuario.

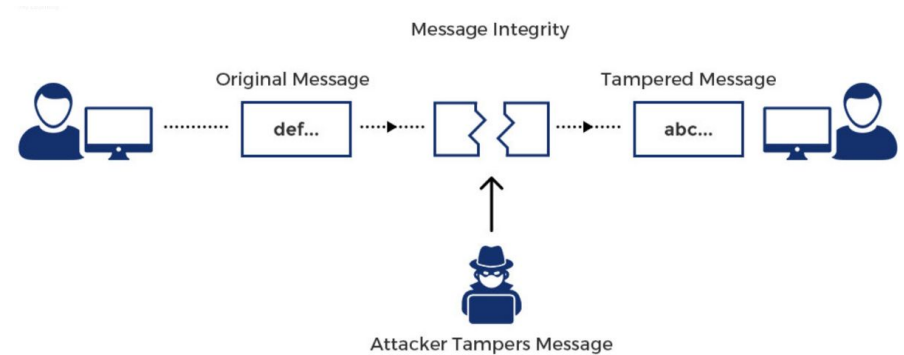


## Caso de Uso

### *Integridad*

Si un mensaje está firmado digitalmente, cualquier cambio en el mensaje después de la firma invalida la firma.

No existe una forma eficiente de modificar un mensaje y su firma para producir un nuevo mensaje con una firma válida, porque la mayoría de las funciones hash criptográficas aún lo consideran computacionalmente inviable.



## Caso de Uso

### *No repudio*

Una entidad que ha firmado alguna información no puede en un momento posterior negar haberla firmado.



# Electronic Codebook (ECB)

Ing. Max Alejandro Antonio Cerna Flores

A dark blue diagonal gradient bar that starts from the bottom left corner and extends towards the top right corner, covering the lower half of the slide.

# Agenda

- ¿Que es un cifrado en bloque?
- ¿Que es el modo de operación de cifrado en bloque?
- Historia
- Vector de Inicialización
- Padding (Rellenado)
- ¿Qué es el modo de cifrado ECB?

# ¿Que es un cifrado en bloque?

Es un algoritmo determinista que opera en grupos de bits de longitud fija, llamados bloques, siendo componentes elementales definidos en el diseño de muchos protocolos criptográficos y se utilizan ampliamente para cifrar grandes cantidades de datos.

Aunque son aptos para cifrar un bloque con una llave fija a la vez cuentan con una variedad de modos de operación que permiten su uso repetido de forma segura para lograr los objetivos de seguridad de confidencialidad y autenticidad.

Operan con bloques completos, por lo que es necesario que la última parte del bloque sea rellena.

# ¿Que es un cifrado en bloque?

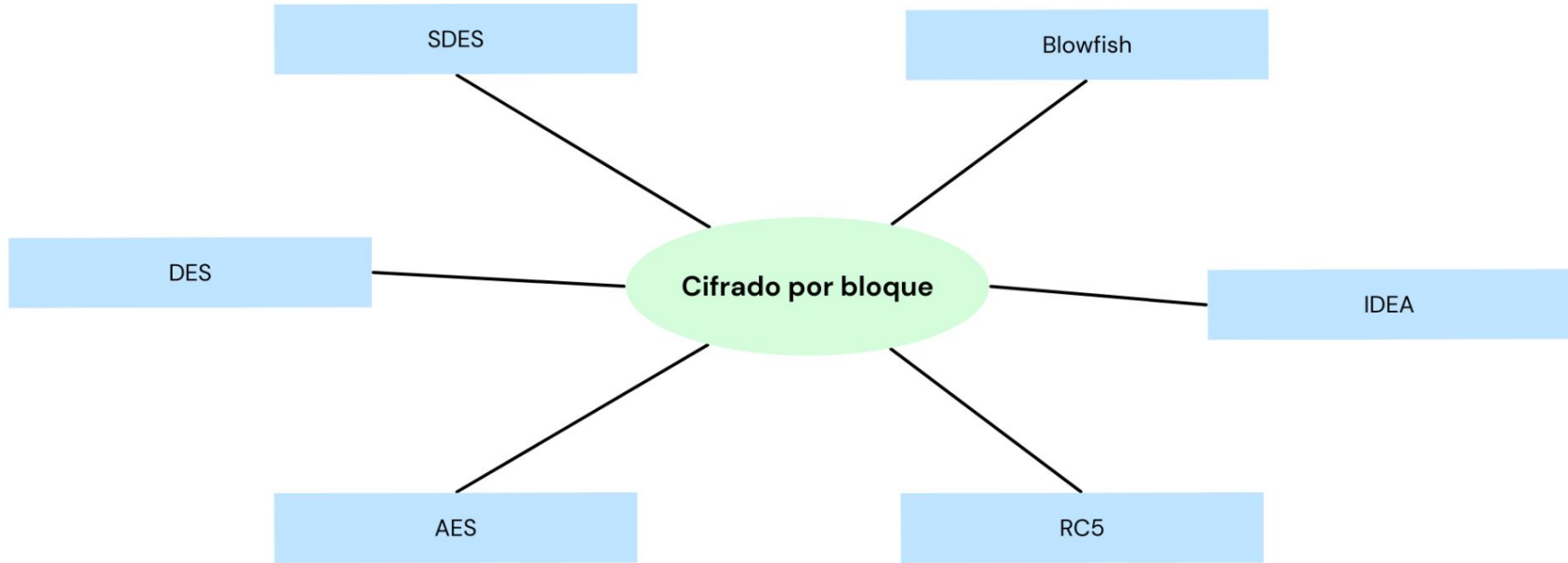
El diseño moderno de este se basa en el concepto de un cifrado de producto por iteración.

Claude Shannon analizó estos y los sugirió como un medio para mejorar la seguridad mediante la combinación de operaciones simples como sustituciones y permutaciones.

Los cifrados de productos por iteración llevan a cabo el cifrado en varias rondas.

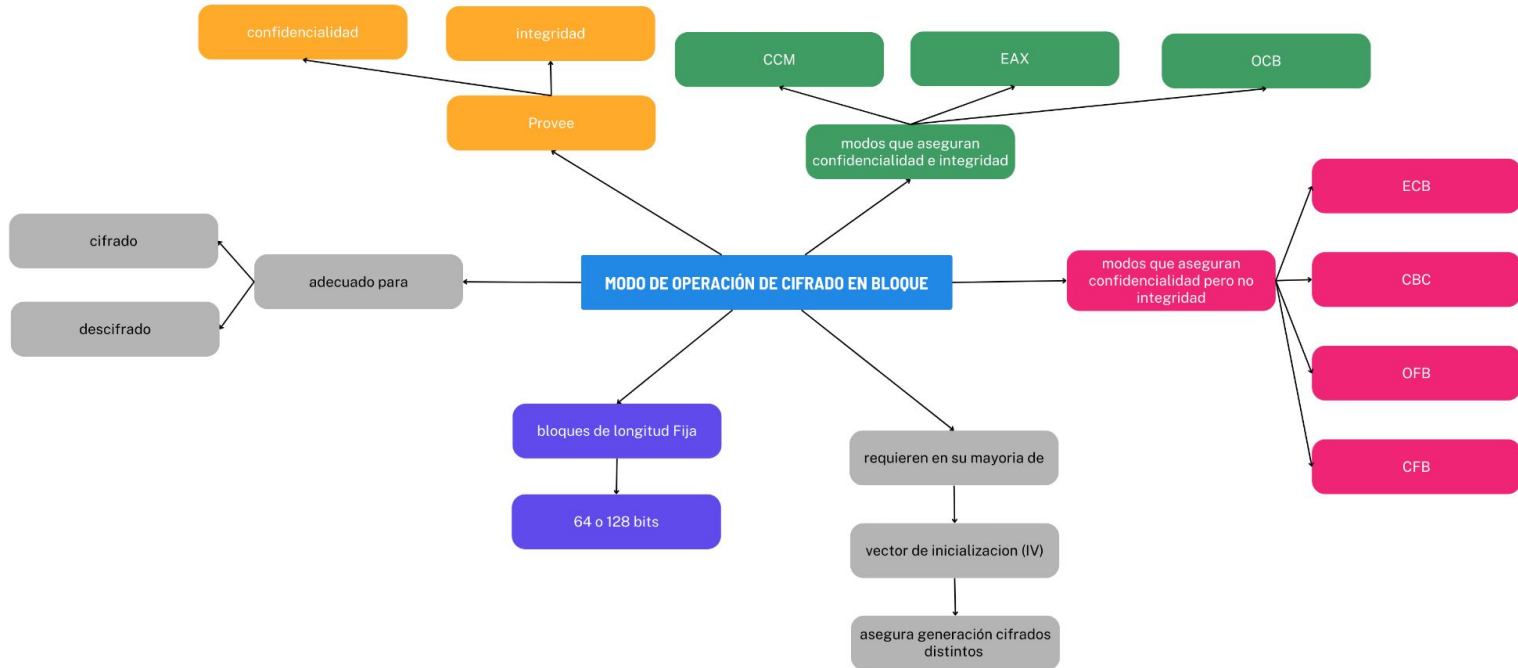
Una implementación generalizada de tales cifrados denominada red Feistel en honor a Horst Feistel.

# ¿Que es un cifrado en bloque?





# ¿Que es el modo de operación de cifrado en bloque?



# Historia

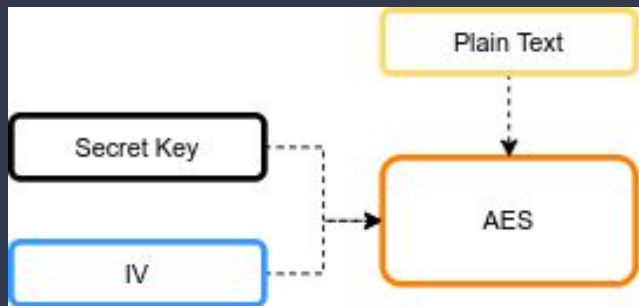
Los primeros modos de operación, ECB, CBC, OFB y CFB, datan de 1981 y se especificaron en FIPS 81(Federal Information Processing Standards Publications).

La manipulación se puede detectar con un código de autenticación de mensaje separado como la firma digital.

La comunidad criptográfica observó que combinar un modo de confidencialidad con un modo de autenticidad podría ser difícil y propenso a errores.

Los modos de operación están definidos por una serie de organismos de normalización reconocidos a nivel nacional e internacional.

# Vector de Inicialización



Es un bloque de bits que es utilizado por varios modos de operación para hacer aleatorio el proceso de cifrado y por lo tanto generar distintos textos cifrados incluso cuando el mismo texto plano es cifrado varias veces, sin la necesidad de regenerar la clave, siendo la regeneración un proceso lento.

# Vector de Inicialización

## ● ● ● Vector de Inicialización (IV)

Tiene requerimientos de seguridad diferentes a los de la clave

No necesita mantenerse en secreto

Tiene que ser no repetitivo y dependiendo del modo también debe aleatorio

Sin embargo, es importante que el IV no sea reutilizado con la misma clave.

**Modos CBC y CFB**  
Si se vuelve a usar el IV permite conocer la información contenida en el primer bloque del texto plano

**Modos OFB y CTR**  
Pone en riesgo la seguridad al crear flujo de bits entre el IV y el texto plano mediante una función XOR.

# Padding

Dado que los algoritmos de cifrado por bloque utilizan bloques de tamaño fijo (ej. DES con 64 bits), pero los mensajes son de longitud variable, es necesario rellenar el espacio del bloque.

En algunos modos de operación como ECB y CBC el último bloque de texto debe ser rellenado antes del proceso de cifrado.

El relleno más simple consiste en agregar null bytes al texto plano.

Debe ser posible recuperar la longitud del texto original.

Los esquemas CBC como *ciphertext stealing* o *residual block termination* agregando complejidad adicional en el proceso.

# Tipos de Padding

## Bit Padding

### Características

Puede ser aplicado a bloques de cualquier tamaño.

Se agrega un bit de bandera para indicar donde empieza el padding generalmente 1.

Se agrega tantos bit de relleno como sean necesarios (quizá ninguno) generalmente 0.

### Implementaciones

Se utiliza en muchas funciones hash, incluidas MD5 y SHA.

## Byte Padding

### Características

Se puede aplicar a los mensajes que se pueden codificar como un número entero de bytes.

### Implementaciones

Implementación en aplicaciones a la medida.

## Zero Padding

### Características

Todos los bytes que deben rellenarse se rellenan con cero.

No se ha estandarizado para el cifrado.

Puede no ser reversible si el archivo original

### Implementaciones

Se especifica para hashes y MAC como método de relleno

## PKCS#5 y PKCS#7

### Características

El valor de los bytes agregar depende de la cantidad de bytes agregados.

Ejemplo: si se agregan 2 bytes, los bytes agregados serían 02 02

Si el dato es múltiplo entero del tamaño del bloque, se agrega un bloque con los valores como el ejemplo

### Implementaciones

Variedad de implementaciones

# Otros tipos de Padding

ANSI X9.23

ISO 10126

ISO/IEC 7816-4

# ¿Qué es el modo de cifrado ECB?

Es el modo más simple de todos.

Se considera inseguro.

No usa el vector de inicialización.

ECB marcó el hito de ser uno de los primeros modos de cifrado por bloques, que permiten la encriptación de cantidades mayores de información.

El principal fallo de seguridad de este método es que los mismos bloques de texto plano siempre producen los mismos textos cifrados.

El mensaje es dividido en bloques, cada uno de los cuales es cifrado de manera separada.

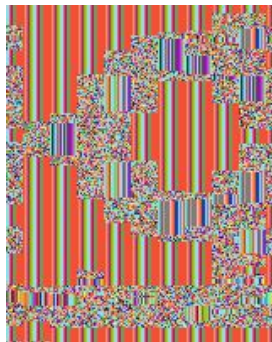
No proporciona una auténtica confidencialidad y no es recomendado para protocolos criptográficos.



# ¿Qué es el modo de cifrado ECB?



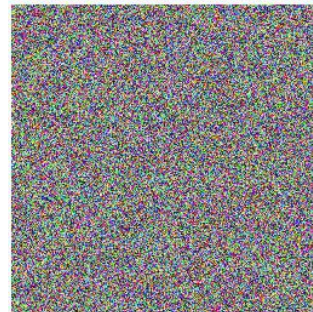
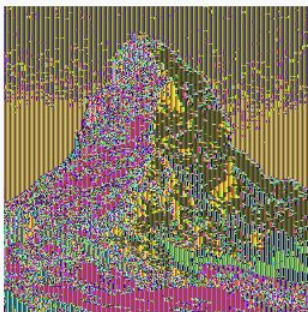
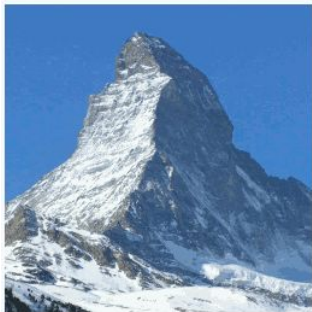
original



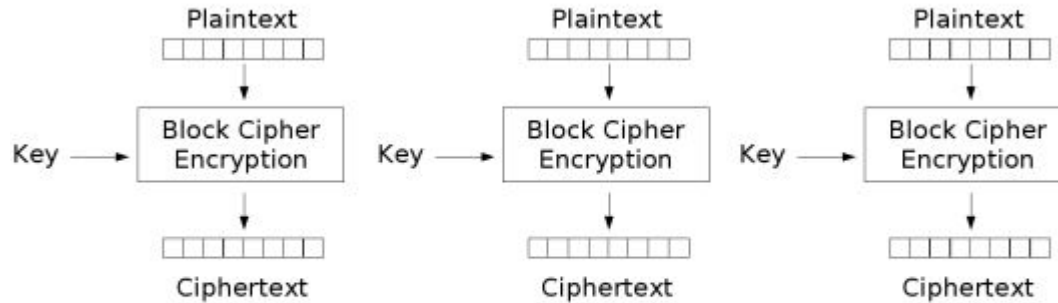
ECB



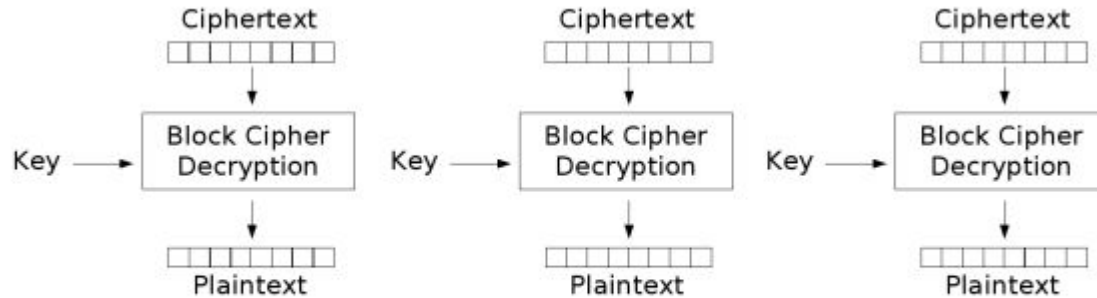
otro modo



# ¿Qué es el modo de cifrado ECB?



# ¿Qué es el modo de cifrado ECB?



Electronic Codebook (ECB) mode decryption

# Cipher-block chaining (CBC)

Ing. Max Alejandro Antonio Cerna Flores

A dark blue diagonal gradient bar that starts from the bottom left corner and extends towards the top right corner, covering the lower half of the slide.

# Agenda

- Elementos requeridos
- ¿Cómo funciona?
- Desventajas y Vulnerabilidades

# Elementos Requeridos

Operación XOR

Vector de Inicialización

Clave

Texto plano o mensaje

Método de cifrado de bloque.

# ¿Cómo funciona?

1. Se define el tamaño del bloque. (ej: 64 bits - 8 caracteres en ASCII)
2. Se define mensaje y clave:

**Mensaje:** batallas anonimas      **clave:** integras - 69 6E 74 65 67 72 61 73

3. Se separa el mensaje en bloques de longitud definida.

**bloque 1:** batallas - 62 61 74 61 6C 6C 61 73

**bloque 2:** anonimas - 61 6E 6F 6E 69 6D 61 73

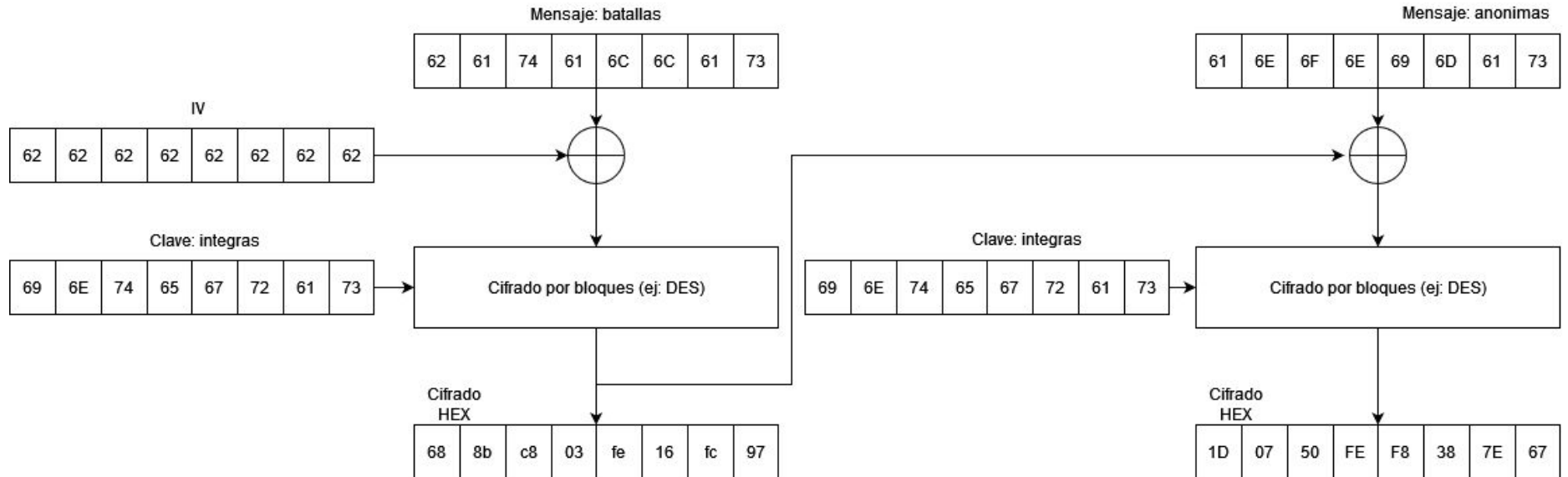
4. Se define IV aleatorio de longitud del bloque (ej: 64 bits).

62 62 62 62 62 62 62 62

5. Escoger método de cifrado por bloques: **DES**

# ¿Cómo funciona?

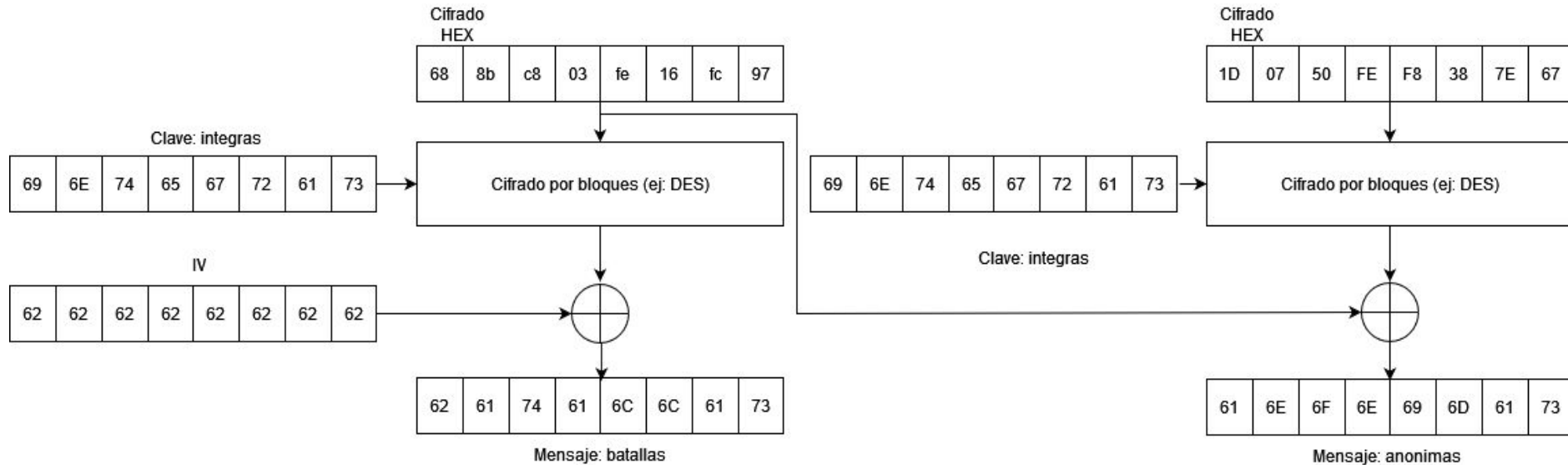
## Cifrado





# ¿Cómo funciona?

## Descifrado



# Desventajas y Vulnerabilidades

- Al ser un método secuencial, no es funcional para ser resuelto en paralelo.
- No permite hacer cambios rápidos en la información cifrada.
- Propagación de errores: Un solo bit erróneo durante la transmisión de un bloque provocará el descifrado incorrecto del bloque siguiente.
- Vulnerable a ataques de oráculo de relleno.

# Desventajas y Vulnerabilidades

*“Un atacante puede usar un oráculo de relleno, en combinación con la manera de estructurar los datos de CBC, para enviar mensajes ligeramente modificados al código que expone el oráculo y seguir enviando datos hasta que el oráculo indique que son correctos. Desde esta respuesta, el atacante puede descifrar el mensaje byte a byte.” - **Vulnerabilidades de temporalización con descifrado simétrico en modo CBC al usar el relleno***

<https://learn.microsoft.com/es-es/dotnet/standard/security/vulnerabilities-cbc-mode>

# Cipher Feedback (CFB)

Ing. Max Alejandro Antonio Cerna Flores

A dark blue diagonal gradient bar that starts from the bottom left and extends towards the top right, covering the lower half of the slide.

# Agenda

- Elementos requeridos
- ¿Cómo funciona?
- Desventajas

# Elementos Requeridos

Operación XOR

Vector de Inicialización

Clave

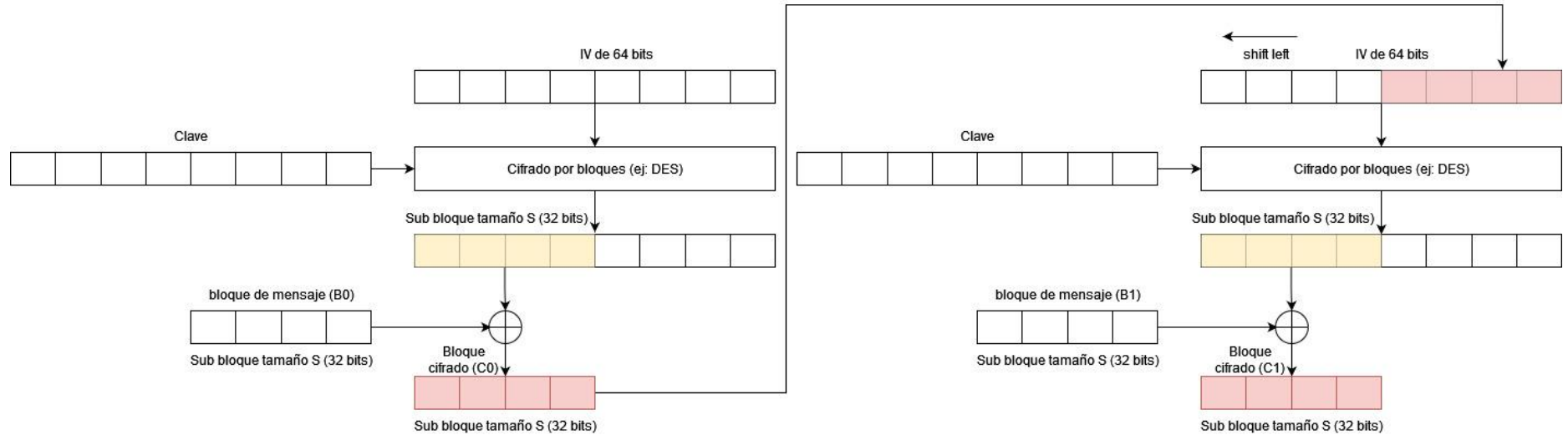
Texto plano o mensaje

Método de cifrado de bloque.

# ¿Cómo funciona?

1. Se define el tamaño del bloque. (ej: 64 bits - 8 caracteres en ASCII).
2. Se define un tamaño del sub-bloque “s” ( $1 < s < \text{bloque}$ ) ej: 32 bits.
3. Se define mensaje y clave a utilizar.
4. Se separa el mensaje en bloques de longitud “s” (ej: 32 bits).
5. Se define IV aleatorio de longitud del bloque (ej: 64 bits).
6. Escoger método de cifrado por bloques: **DES**

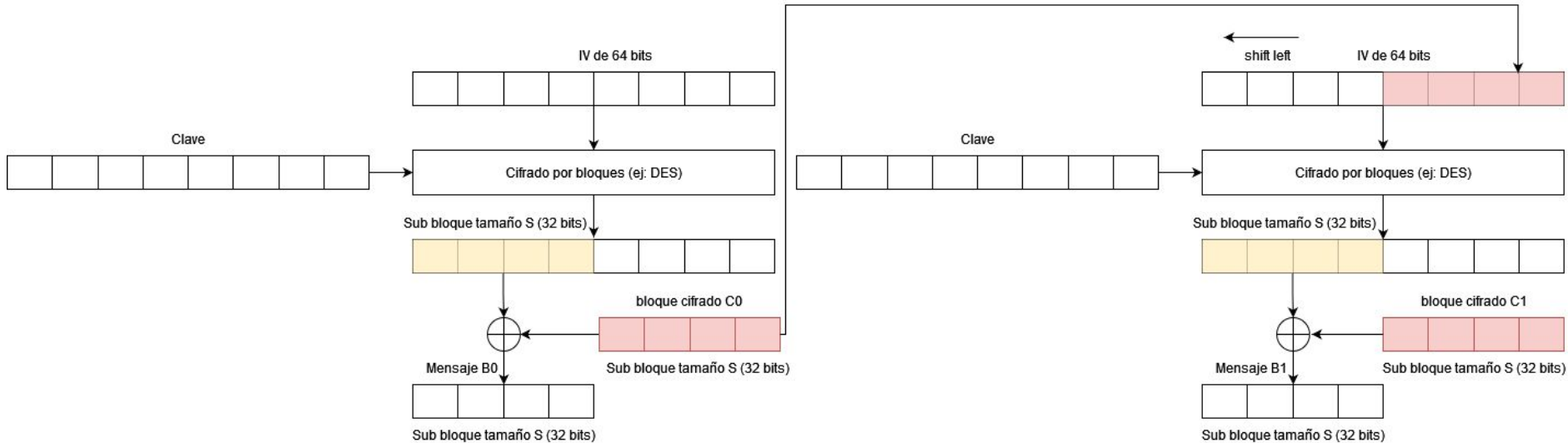
# ¿Cómo funciona?





# ¿Cómo funciona?

## Descifrado



# Ventajas y Desventajas

- Dado que existe cierta pérdida de datos debido al uso del registro de desplazamiento, es difícil aplicar el criptoanálisis.
- El error de transmisión se propaga debido al cambio de bloques.

# Cipher Feedback (CFB) Full-Block

Ing. Max Alejandro Antonio Cerna Flores

A dark blue diagonal gradient bar that starts from the bottom left and extends towards the top right, covering the lower half of the slide.

# Agenda

- Elementos requeridos
- ¿Cómo funciona?
- Desventajas

# Elementos Requeridos

Operación XOR

Vector de Inicialización

Clave

Texto plano o mensaje

Método de cifrado de bloque.

# ¿Cómo funciona?

1. Se define el tamaño del bloque. (ej: 64 bits - 8 caracteres en ASCII)
2. Se define mensaje y clave:

**Mensaje:** cuaderno continuo      **clave:** practico - 70 72 61 63 74 69 63 6F

3. Se separa el mensaje en bloques de longitud definida.

**bloque 1:** cuaderno - 63 75 61 64 65 72 6E 6F

**bloque 2:** continuo - 63 6F 6E 74 69 6E 75 6F

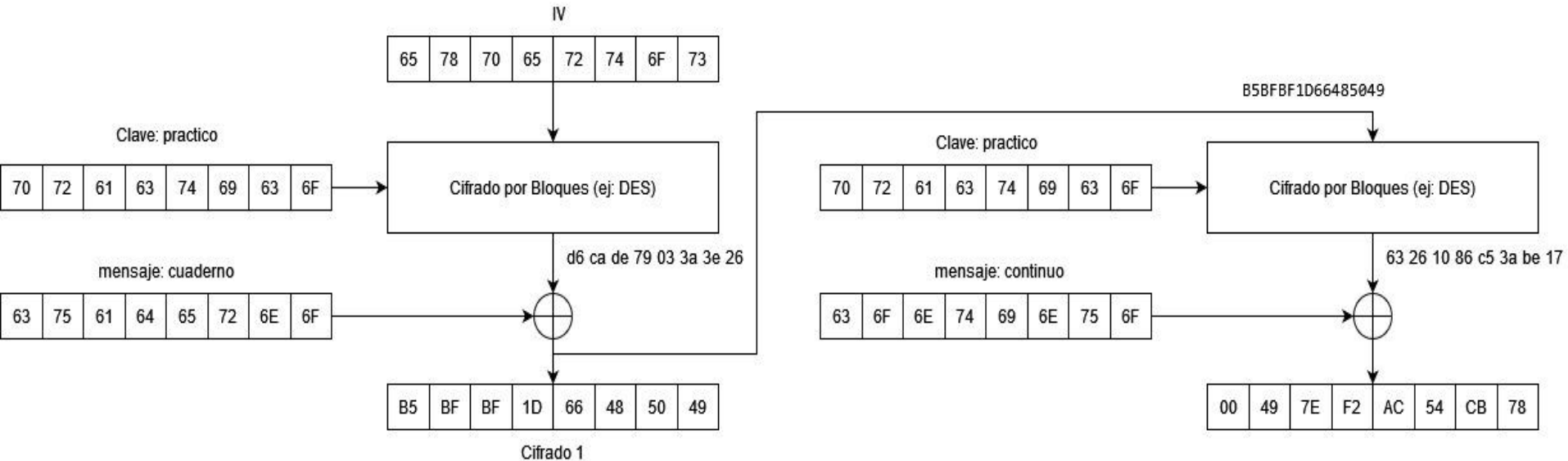
4. Se define IV aleatorio de longitud del bloque (ej: 64 bits).

65 78 70 65 72 74 6F 73

5. Escoger método de cifrado por bloques: **DES**

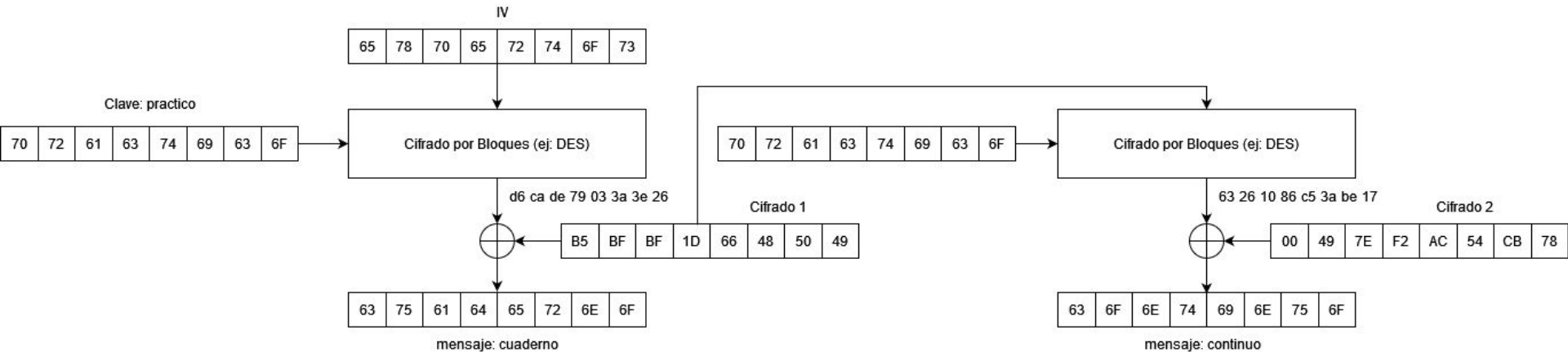
# ¿Cómo funciona?

## Cifrado



# ¿Cómo funciona?

## Descifrado





# Desventajas

- Al ser un método secuencial, no es funcional para ser resuelto en paralelo.
- No permite hacer cambios rápidos en la información cifrada.
- Propagación de errores: Un solo bit erróneo durante la transmisión de un bloque provocará el descifrado incorrecto del bloque siguiente.

# Output Feedback (OFB)

Ing. Max Alejandro Antonio Cerna Flores

A dark blue diagonal gradient bar that starts from the bottom left and extends towards the top right, covering the lower half of the slide.

# Agenda

- Elementos requeridos
- ¿Cómo funciona?
- Desventajas

# Elementos Requeridos

Operación XOR

Vector de Inicialización

Clave

Texto plano o mensaje

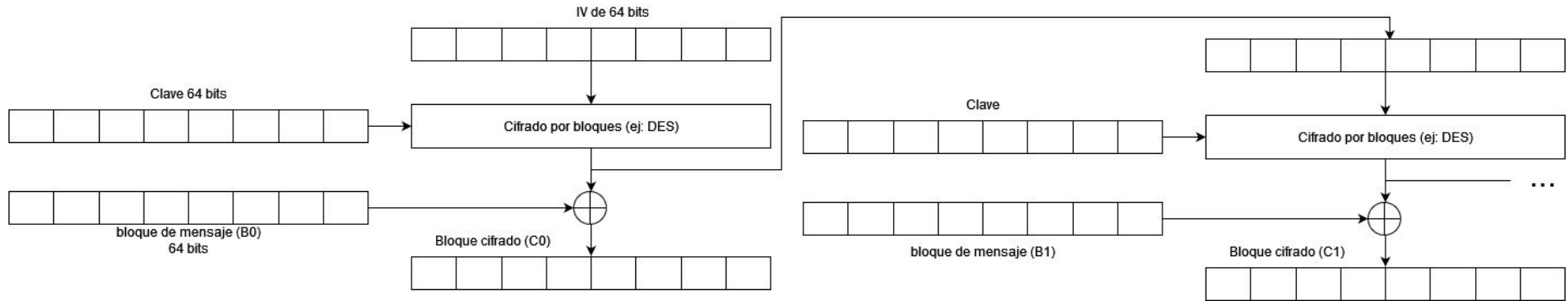
Método de cifrado de bloque.

# ¿Cómo funciona?

1. Se define el tamaño del bloque. (ej: 64 bits - 8 caracteres en ASCII).
2. Se define mensaje y clave a utilizar.
3. Se separa el mensaje en bloques (ej: 64 bits).
4. Se define IV aleatorio de longitud del bloque (ej: 64 bits).
5. Escoger método de cifrado por bloques: **DES**

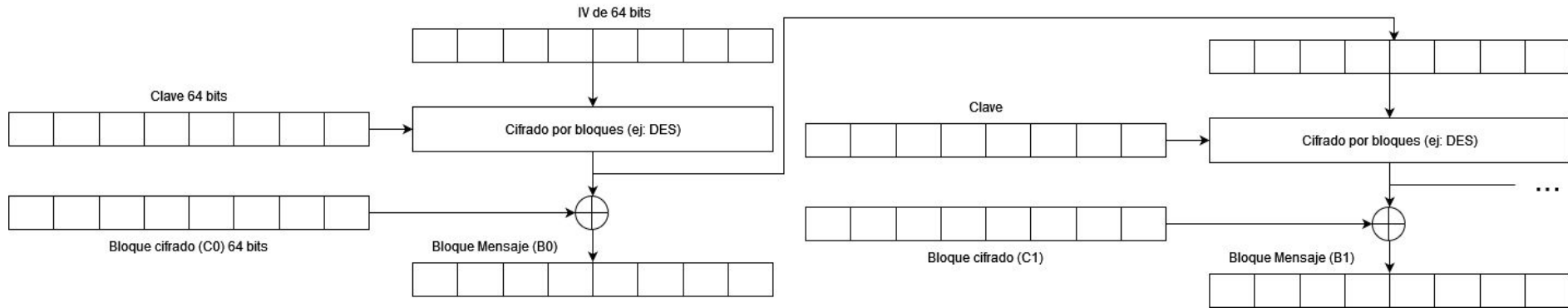
# ¿Cómo funciona?

## Cifrado



# ¿Cómo funciona?

## Descifrado



# Ventajas y Desventajas

- El error de propagación de bit corrupto se resuelve en este modo, ya que está libre de errores de bit en el bloque de texto plano.
- No se puede realizar en paralelo.
- Las operaciones de cifrado de bloques se pueden realizar con anticipación, lo que permite que el paso final se realice en paralelo una vez que el texto o el texto cifrado esté disponible.



---

---

# Codificacion de Huffman

— Ing. Max Alejandro Antonio  
Cerna Flores —

---

---

# Agenda

Historia

Definición

¿Cómo trabaja?

Decodificación

Complejidad de codificación de Huffman

Aplicaciones

# Historia

En 1952, David Huffman propuso un método estadístico que permitía asignar un código binario a los diversos símbolos a comprimir.

Fue desarrollado por David A. Huffman mientras era estudiante de doctorado en el MIT, y publicado en "A Method for the Construction of Minimum-Redundancy Codes".

Huffman superó a su profesor Robert. M. Fano, quien había trabajado con el inventor de la teoría de la información Claude Shannon con el fin de desarrollar un código similar.

# Definición

Es un algoritmo para realizar la compresión de datos y forma la idea básica detrás de la compresión de archivos.

La longitud de cada código no es idéntica para todos los símbolos mientras que los símbolos menos frecuentes reciben códigos binarios más largos.

Huffman solucionó la mayor parte de los errores en el algoritmo de codificación Shannon-Fano.

La solución se basaba en el proceso de construir el árbol de ***abajo a arriba*** en vez de al contrario.

# ¿Cómo trabaja?

Supongamos un string enviado a través de la red como se muestra a continuación:



cada carácter representado  
por 8 bits

El string mostrado contiene 15 caracteres.

$$8 * 15 = 120 \text{ bits requeridos para el envio}$$

# Pasos

Se puede comprimir el string a un tamaño menor.

1. Calcule la frecuencia de cada carácter en el string.

1	6	5	3
B	C	A	D

2. Ordene los caracteres en orden creciente de frecuencia.

1	3	5	6
B	D	A	C

# Pasos

Se puede comprimir el string a un tamaño menor.

3. Hacer que cada carácter único sea representado como un nodo hoja.

4. Crear un nodo vacío:

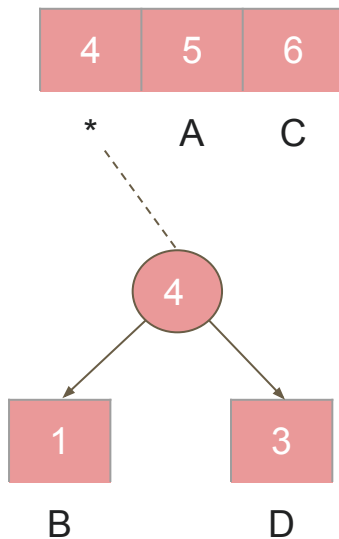
4.1 Asignar al nodo hoja izquierdo el de menor frecuencia.

4.2 Asignar al nodo hoja derecho el siguiente menor.

4.3 Asignar al nodo vacío el valor de la suma de las frecuencias.

# Pasos

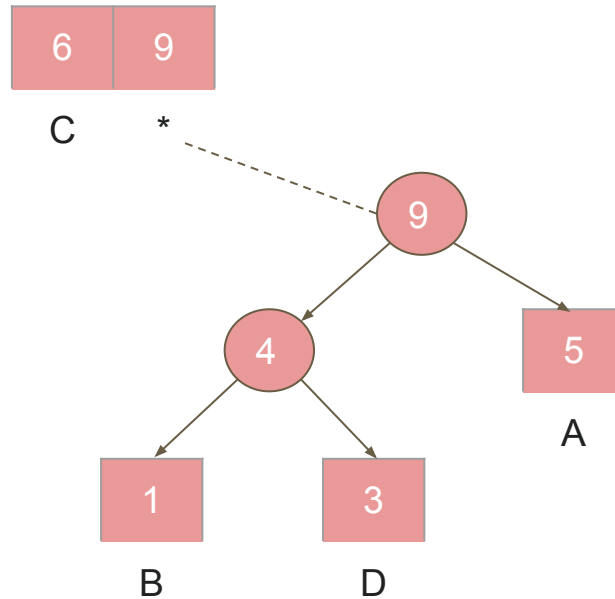
5. Se remueven los valores usados en los nodos hoja de la cola y se agrega a la cola el nuevo nodo con la suma de las frecuencias de los nodos hoja.



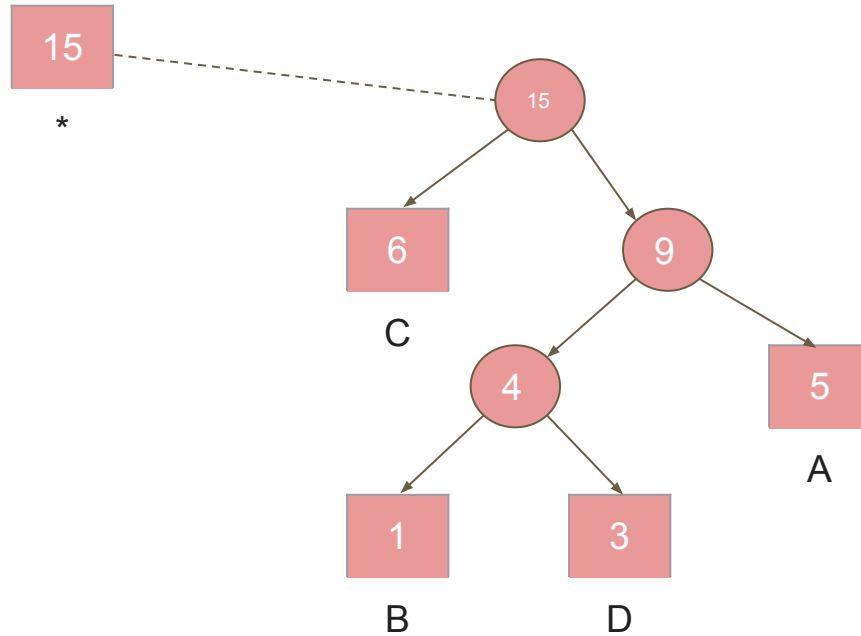


# Pasos

6. Repetir paso 3 y 5 para todos los caracteres sobrantes.

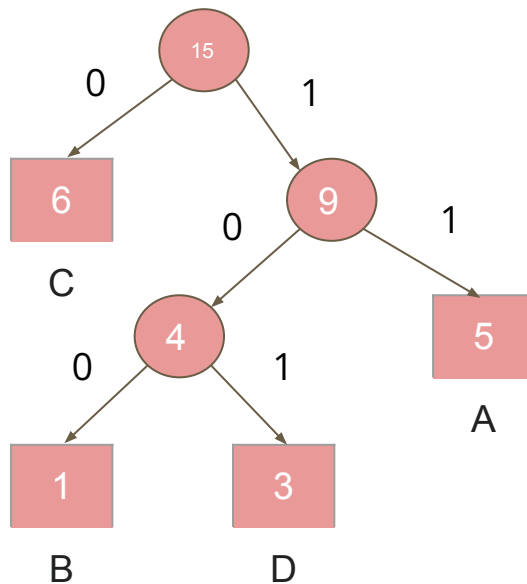


# Pasos



# Pasos

7. Para cada no hijo, asignar 0 a la rama izquierda y 1 a la rama derecha.



# Tabla de codificación

Carácter	Frecuencia	Código	Tamaño
A	5	11	$5 \cdot 2 = 10$
B	1	100	$1 \cdot 3 = 3$
C	6	0	$6 \cdot 1 = 6$
D	3	101	$3 \cdot 3 = 9$
4*8= 32 bits	15 bits		28 bits

Diccionario de datos

Caracteres a utilizar tras codificación =  $32 + 15 + 28 = 75$  bits

# Pasos

O	T	O	R	R	I	N	O	L	A	R	I	N	G	O	L	O	G	I	A
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---



5	1	3	3	2	2	2	2
---	---	---	---	---	---	---	---

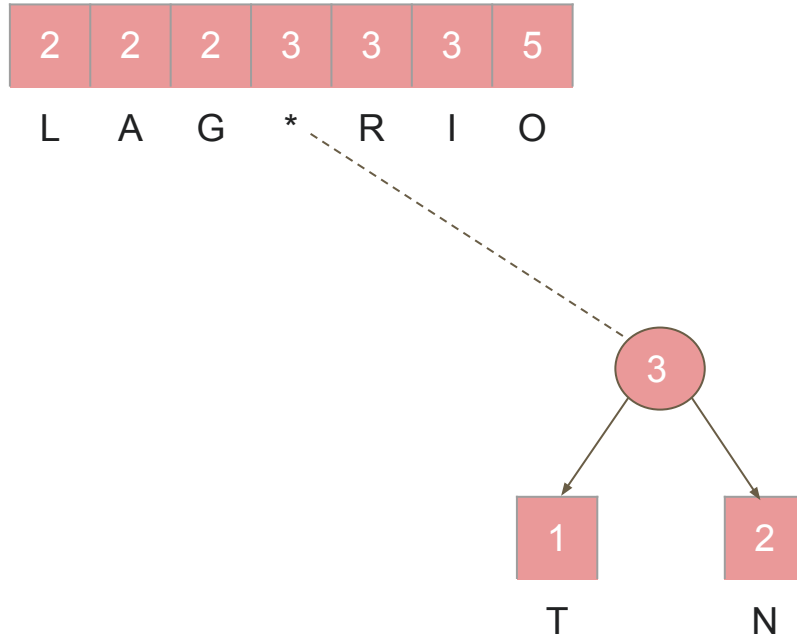
O T R I N L A G



1	2	2	2	2	3	3	5
---	---	---	---	---	---	---	---

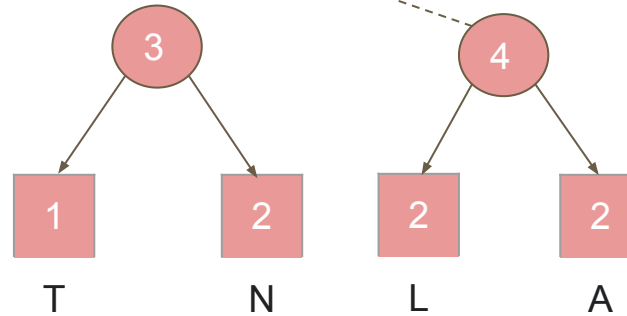
T N L A G R I O

# Pasos



# Pasos

2	3	3	3	4	5
G	*	R	I	*	O



# Pasos

3	3	4	5	5
---	---	---	---	---

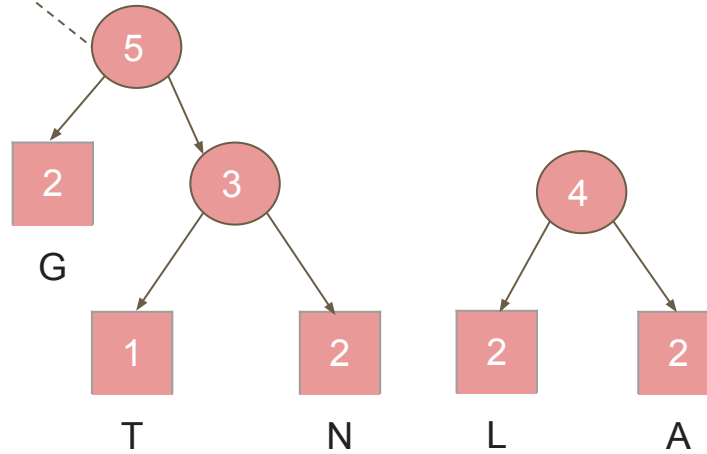
R

I

\*

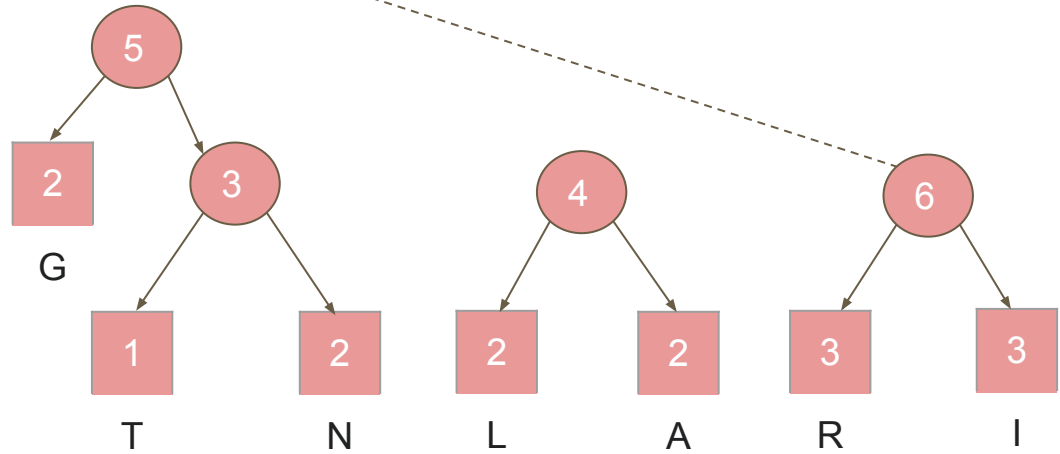
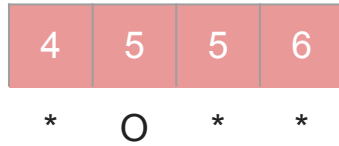
O

\*

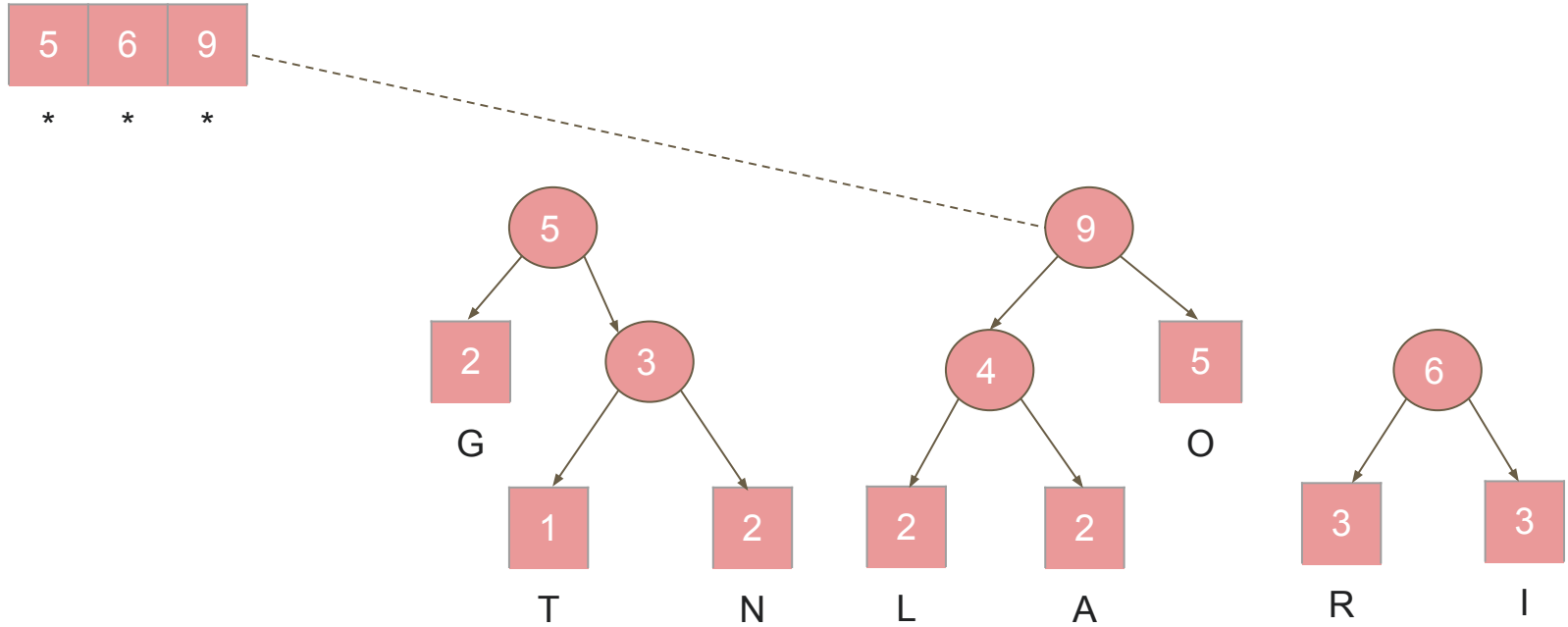




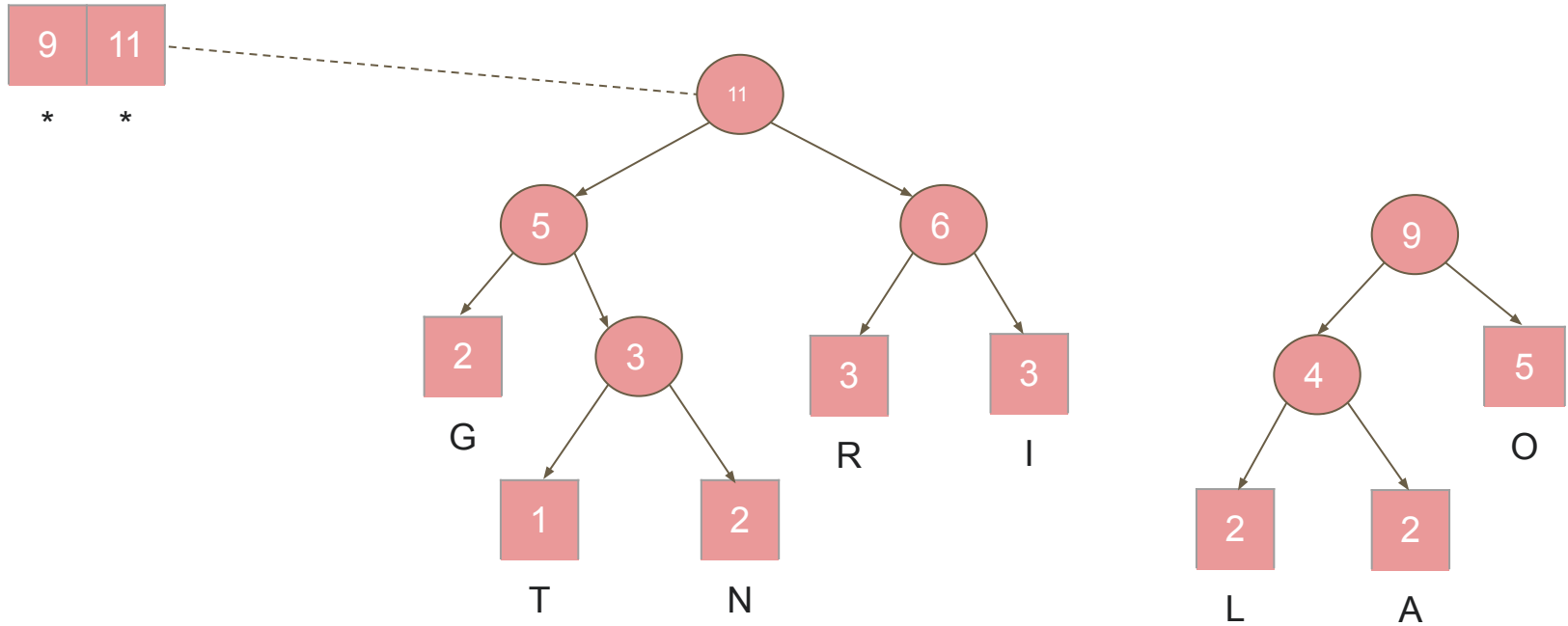
# Pasos



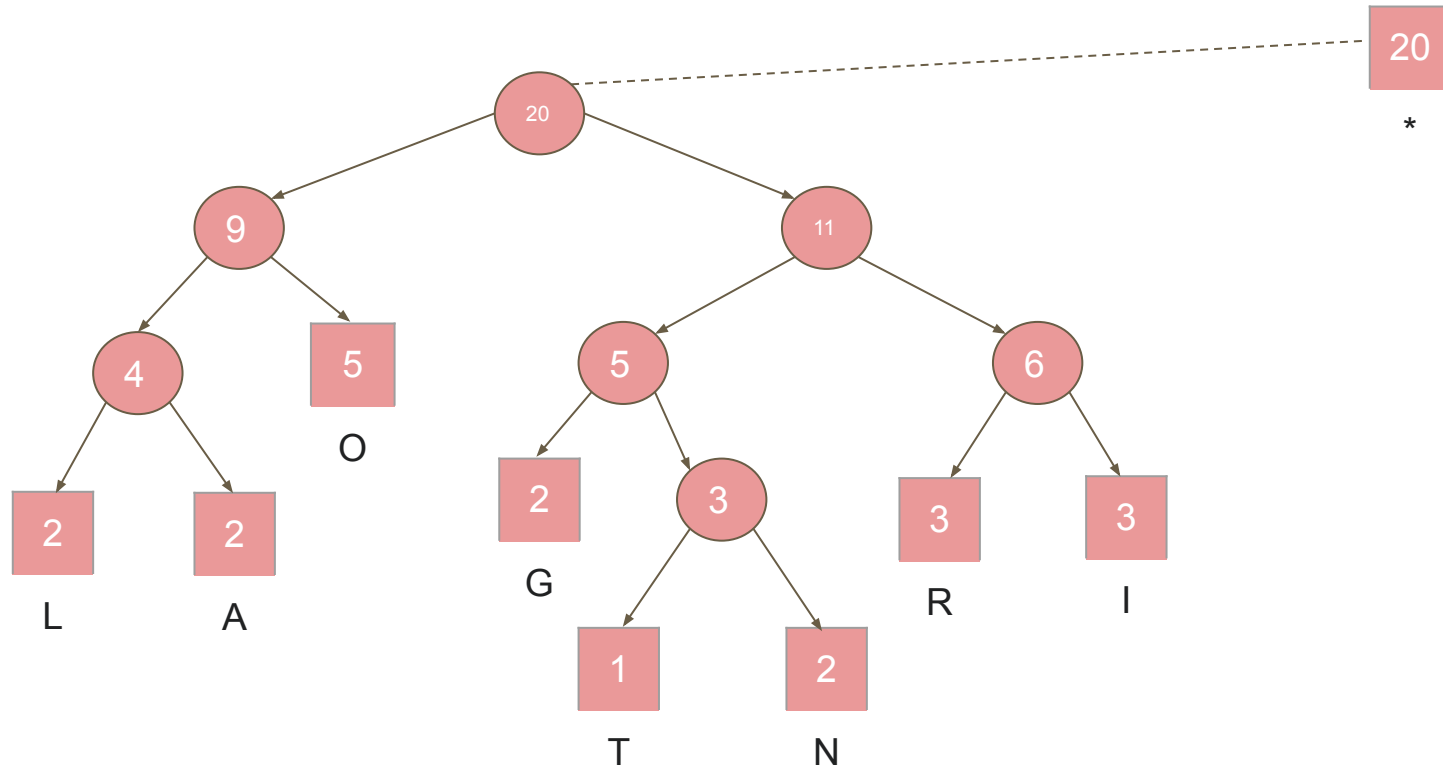
# Pasos



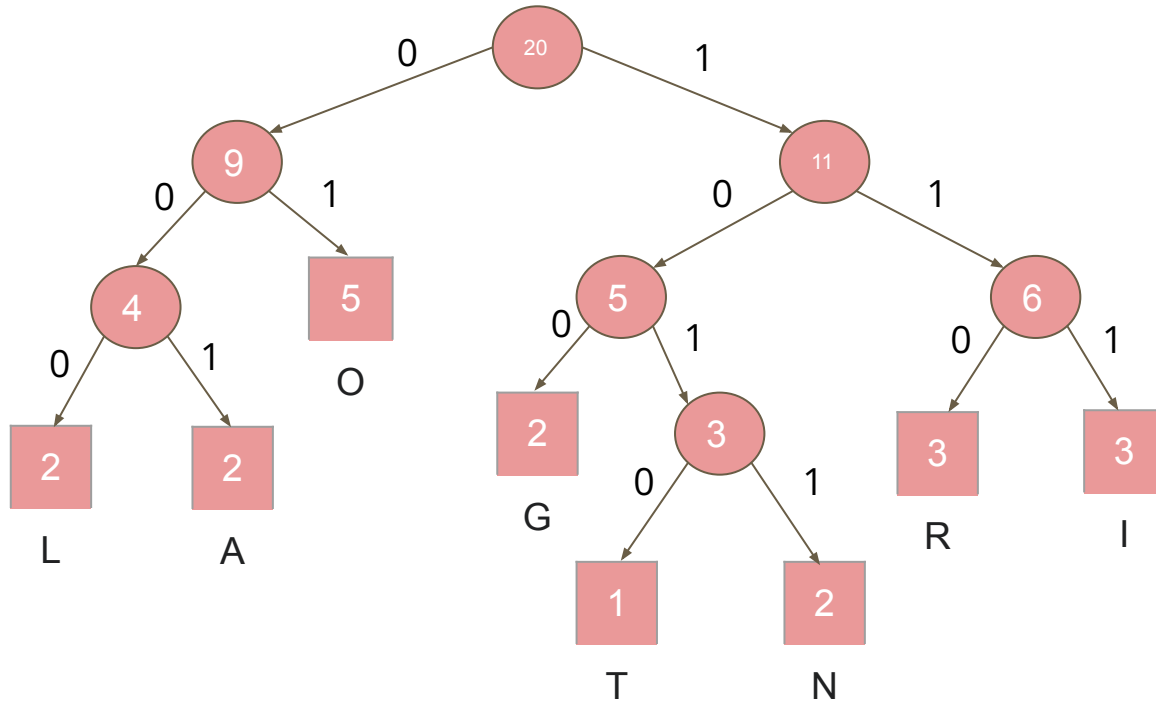
# Pasos



# Pasos



# Pasos



# Tabla de codificación

Carácter	Frecuencia	Código	Tamaño
L	2	000	$2 \times 3 = 6$
A	2	001	$2 \times 3 = 6$
O	5	01	$5 \times 2 = 10$
G	2	100	$2 \times 3 = 6$
T	1	1010	$1 \times 4 = 4$
N	2	1011	$2 \times 4 = 8$
R	3	110	$3 \times 3 = 9$
I	3	111	$3 \times 3 = 9$
8*8= 64 bits	20 bits		58 bits

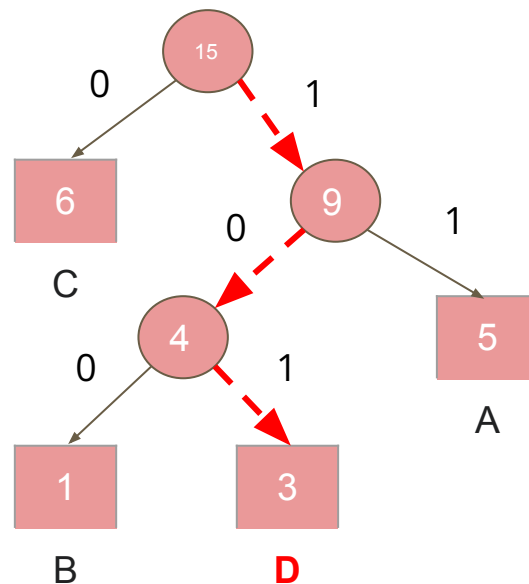
total codificado:  
**142 bits**

total codificado:  
**20\*8=160 bits**

# Decodificación

Es simplemente una cuestión de traducir el flujo de códigos de prefijo a valores de bytes individuales, generalmente atravesando el árbol de Huffman nodo por nodo a medida que se lee cada bit del flujo de entrada.

Decodificar de 101 del primer ejemplo.



# Complejidad de codificación de Huffman

La complejidad de tiempo para codificar cada carácter único en función de su frecuencia es  **$O(n \log n)$** .

Extraer la frecuencia mínima de la cola de prioridad se realiza  $2 \cdot (n-1)$  veces y su complejidad es  **$O(\log n)$** .

Por tanto, la complejidad global es  **$O(n \log n)$** .



---

---

# Algoritmo LZ77

— Ing. Max Alejandro Antonio  
Cerna Flores —

---

---

# Agenda

Historia

Definición

¿Cómo trabaja?

Decodificación

# Historia

En 1977 fue publicado el artículo titulado "Un algoritmo universal para la compresión secuencial de datos".

El artículo fue escrito por Abraham Lempel y Jacob Ziv, donde presentaron su modelo de compresión basado en diccionario.

LZ son las iniciales de sus creadores y 77 el año en que se creó.

# Definición

Es un compresor basado en algoritmos sin pérdida.

Es un algoritmo basado en diccionario que codifica cadenas largas (también conocidas como frases) en tokens cortos y reemplaza frases en el diccionario con tokens pequeños para lograr el propósito de la compresión.

Usa un búfer de avance y una ventana deslizante.

Este algoritmo es muy usado porque es fácil de implementar y bastante eficiente en una relación costo/beneficio.

# ¿Cómo trabaja?

- Se define un buffer de avance de tamaño dado.
- Se define un diccionario
- Ejemplo:
  - Buffer = 4 

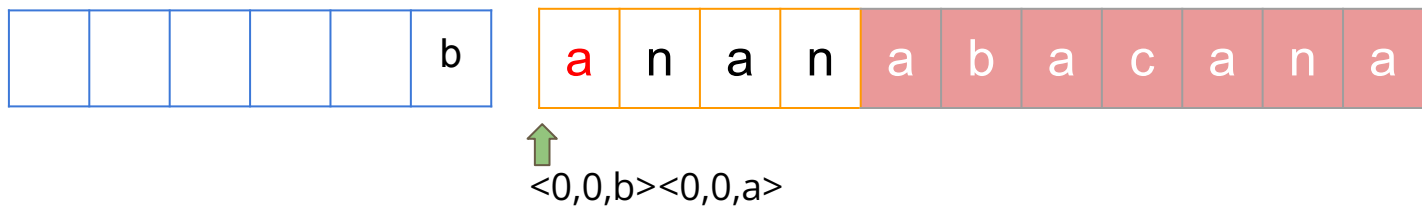
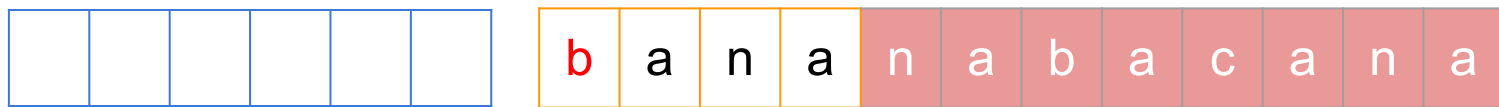
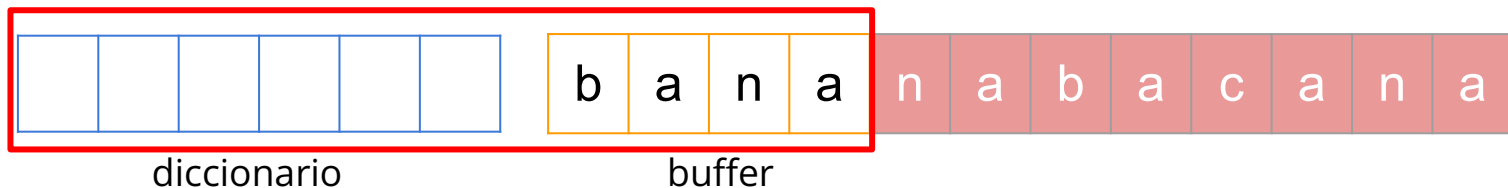
--	--	--	--
  - diccionario = 6 

--	--	--	--	--	--
  - Puntero ↑
  - tripleta [posición, longitud, carácter siguiente]

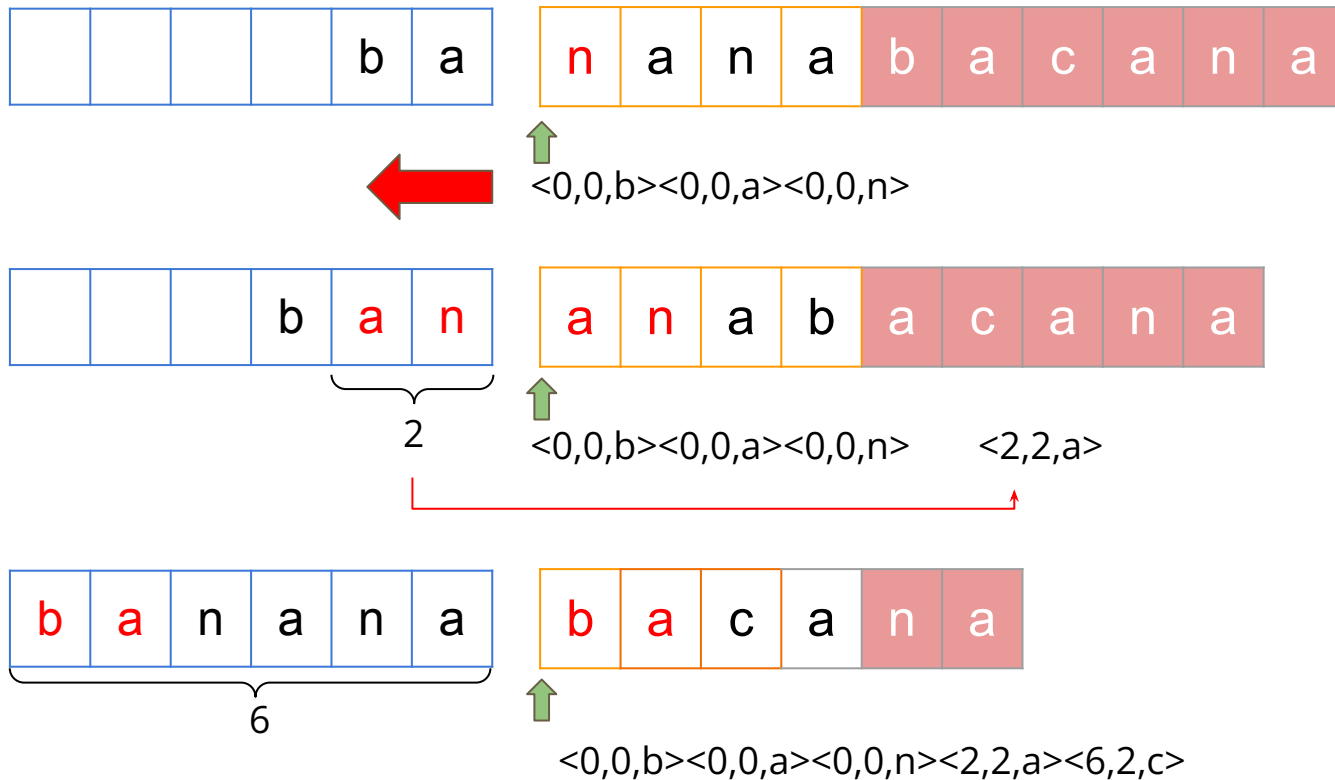
b	a	n	a	n	a	b	a	c	a	n	a
---	---	---	---	---	---	---	---	---	---	---	---

# ¿Cómo trabaja?

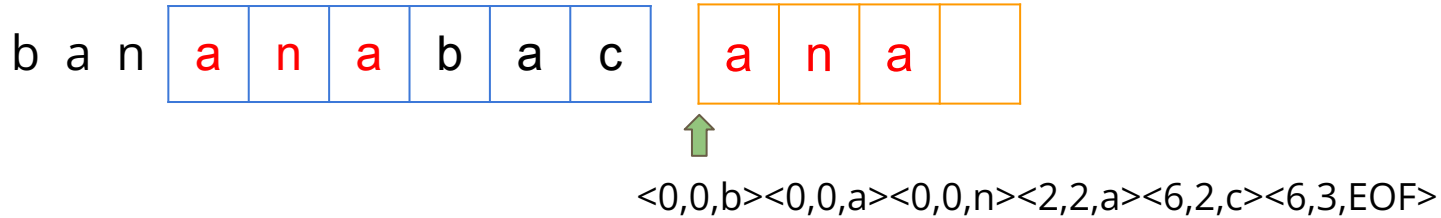
Ventana



# ¿Cómo trabaja?



# ¿Cómo trabaja?



b a n a n a 

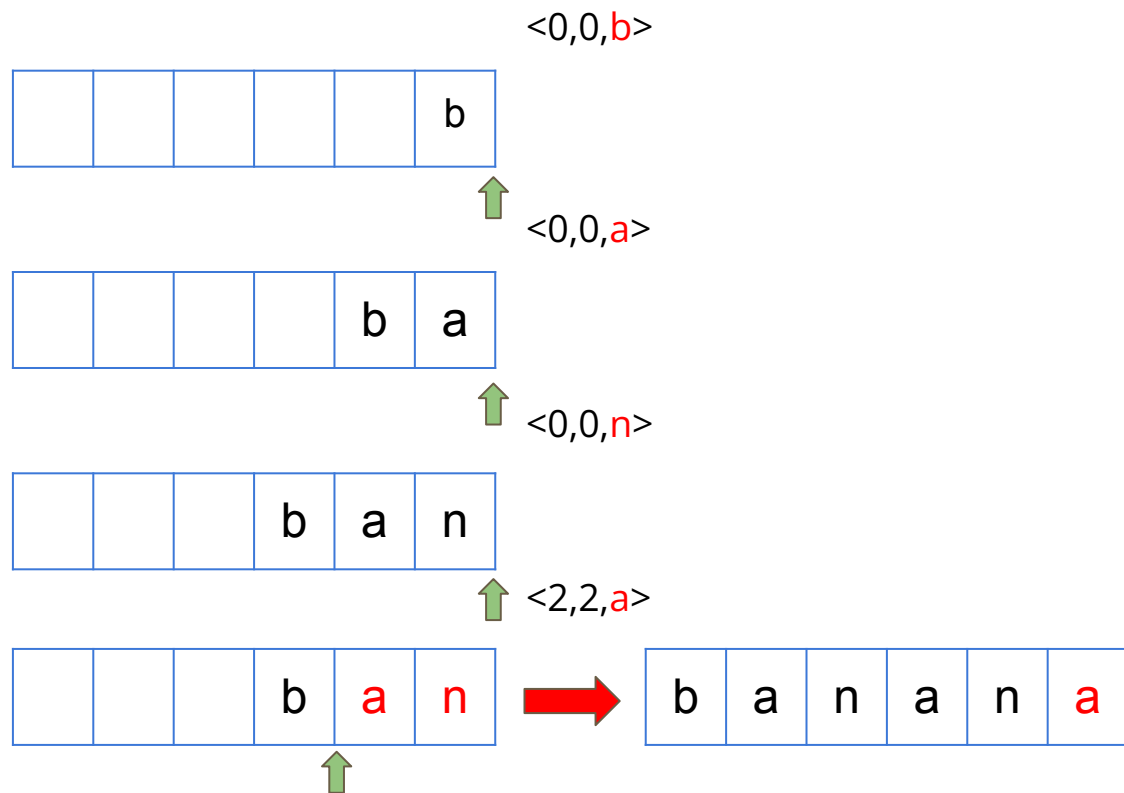
b	a	c	a	n	a
---	---	---	---	---	---

**Mensaje codificado**

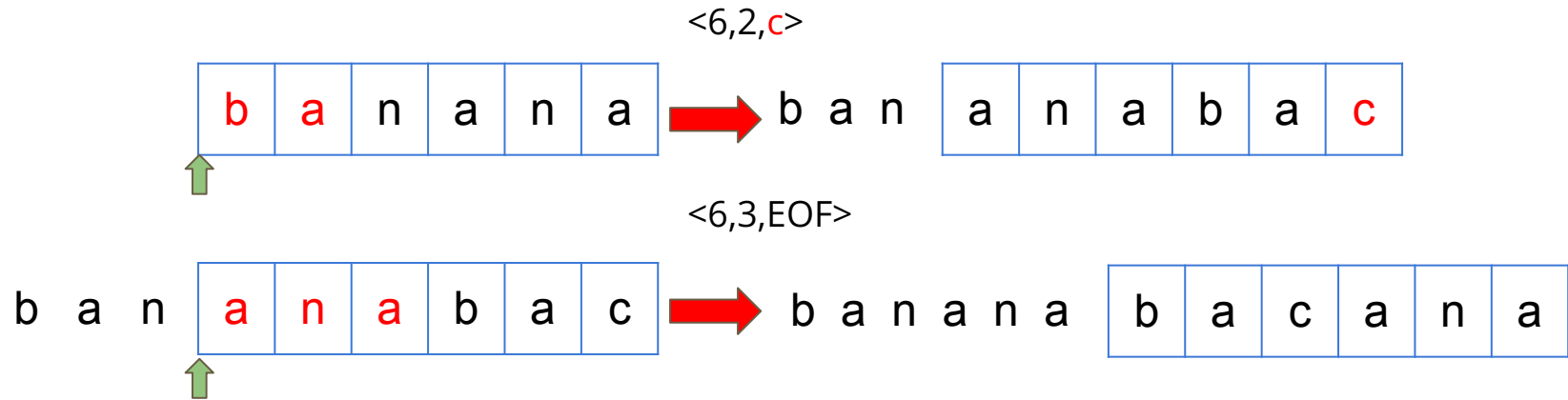
$\langle 0,0,b \rangle \langle 0,0,a \rangle \langle 0,0,n \rangle \langle 2,2,a \rangle \langle 6,2,c \rangle \langle 6,3,EOF \rangle$



# Decodificación



# Decodificación



Mensaje decodificado

**bananabacana**

---

---

# Algoritmo LZ78

— Ing. Max Alejandro Antonio  
Cerna Flores —

---

---

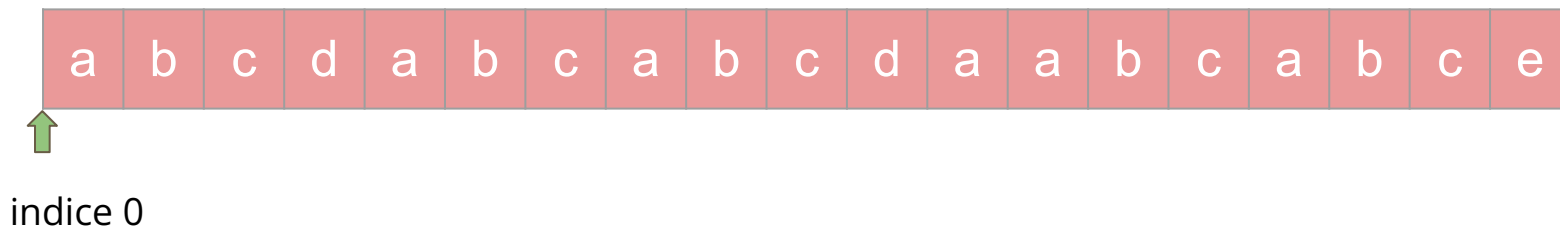
# Agenda

¿Cómo trabaja?

Decodificación

# ¿Cómo trabaja?

- Se define un diccionario basado en frases.
- Ejemplo:



# ¿Cómo trabaja?

a b c d a b c a b c d a a b c a b c e



<0,a>

a b c d a b c a b c d a a b c a b c e



<0,b>

a b c d a b c a b c d a a b c a b c e



<0,c>

i	sal	entry
1	<0,a>	a
2	<0,b>	b
3	<0,c>	c

# ¿Cómo trabaja?

a b c d a b c a b c d a a b c a b c e



<0,d>

a b c d a b c a b c d a a b c a b c e



<1,b>

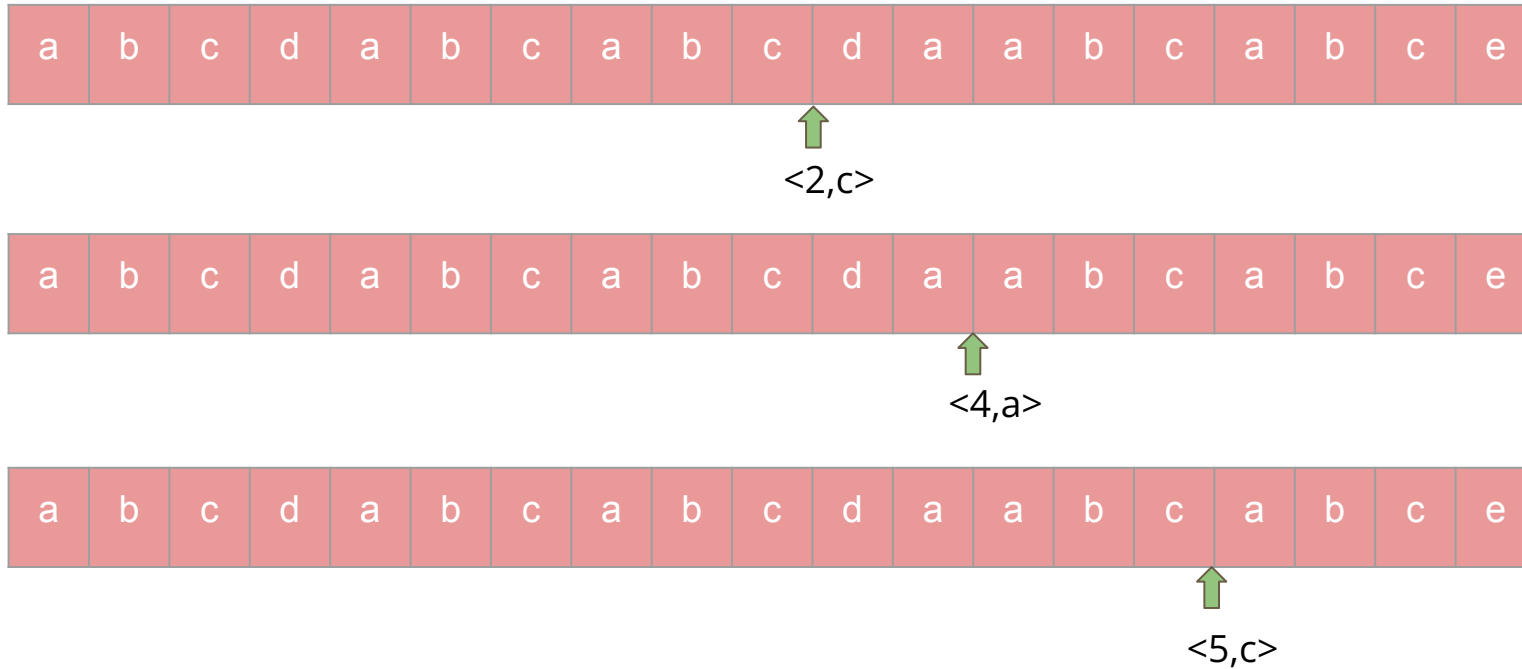
a b c d a b c a b c d a a b c a b c e



<3,a>

i	sal	entry
1	<0,a>	a
2	<0,b>	b
3	<0,c>	c
4	<0,d>	d
5	<1,b>	ab
6	<3,a>	ca

# ¿Cómo trabaja?



i	sal	entry
1	$\langle 0, a \rangle$	a
2	$\langle 0, b \rangle$	b
3	$\langle 0, c \rangle$	c
4	$\langle 0, d \rangle$	d
5	$\langle 1, b \rangle$	ab
6	$\langle 3, a \rangle$	ca
7	$\langle 2, c \rangle$	bc
8	$\langle 4, a \rangle$	da
9	$\langle 5, c \rangle$	abc



# ¿Cómo trabaja?

a	b	c	d	a	b	c	a	b	c	d	a	a	b	c	a	b	c	e
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

Resultado codificado:

**<0,a><0,b><0,c><0,d><1,b><3,a><2,c><4,a><5,c><9,e>**

↑  
<9,e>

i	sal	entry
1	<0,a>	a
2	<0,b>	b
3	<0,c>	c
4	<0,d>	d
5	<1,b>	ab
6	<3,a>	ca
7	<2,c>	bc
8	<4,a>	da
9	<5,c>	abc
10	<9,e>	abce

# Decodificación

código	índice	entrada
<0,a>	1	
<0,b>	2	
<0,c>	3	
<0,d>	4	
<1,b>	5	
<3,a>	6	
<2,c>	7	
<4,a>	8	
<5,c>	9	
<9,e>	10	

# Decodificación

código	índice	entrada
<0,a>	1	a
<0,b>	2	b
<0,c>	3	c
<0,d>	4	d
<1,b>	5	
<3,a>	6	
<2,c>	7	
<4,a>	8	
<5,c>	9	
<9,e>	10	

# Decodificación

código	índice	entrada
<0,a>	1	a
<0,b>	2	b
<0,c>	3	c
<0,d>	4	d
<1,b>	5	ab
<3,a>	6	
<2,c>	7	
<4,a>	8	
<5,c>	9	
<9,e>	10	

# Decodificación

código	índice	entrada
<0,a>	1	a
<0,b>	2	b
<0,c>	3	c
<0,d>	4	d
<1,b>	5	ab
<3,a>	6	ca
<2,c>	7	
<4,a>	8	
<5,c>	9	
<9,e>	10	

# Decodificación

código	índice	entrada
<0,a>	1	a
<0,b>	2	b
<0,c>	3	c
<0,d>	4	d
<1,b>	5	ab
<3,a>	6	ca
<2,c>	7	bc
<4,a>	8	
<5,c>	9	
<9,e>	10	

# Decodificación

código	índice	entrada
<0,a>	1	a
<0,b>	2	b
<0,c>	3	c
<0,d>	4	d
<1,b>	5	ab
<3,a>	6	ca
<2,c>	7	bc
<4,a>	8	da
<5,c>	9	
<9,e>	10	

# Decodificación

código	índice	entrada
<0,a>	1	a
<0,b>	2	b
<0,c>	3	c
<0,d>	4	d
<1,b>	5	ab
<3,a>	6	ca
<2,c>	7	bc
<4,a>	8	da
<5,c>	9	abc
<9,e>	10	



# Decodificación

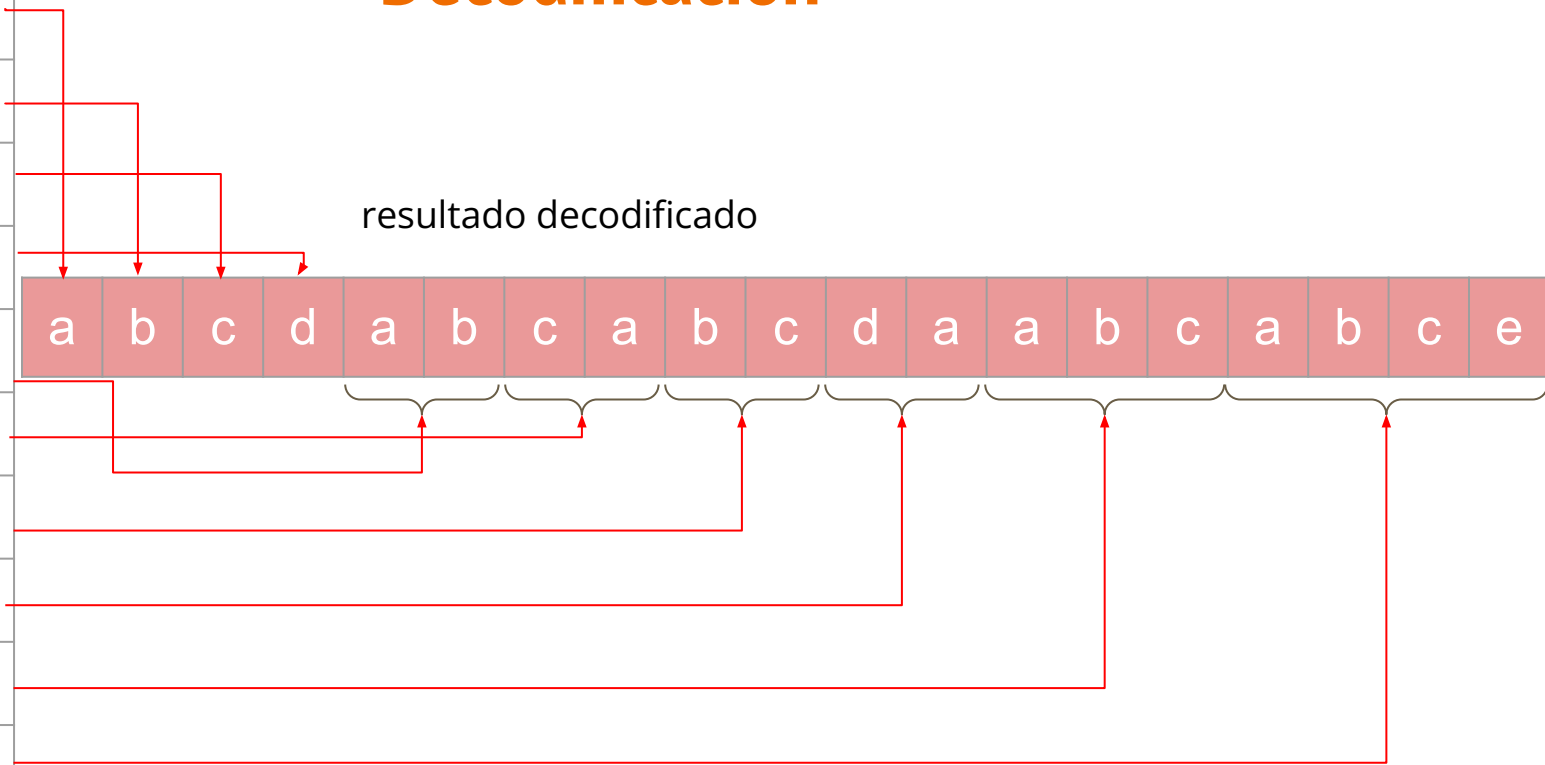
código	índice	entrada
<0,a>	1	a
<0,b>	2	b
<0,c>	3	c
<0,d>	4	d
<1,b>	5	ab
<3,a>	6	ca
<2,c>	7	bc
<4,a>	8	da
<5,c>	9	abc
<9,e>	10	abce

# Decodificación

cod	i	entry
<0,a>	1	a
<0,b>	2	b
<0,c>	3	c
<0,d>	4	d
<1,b>	5	ab
<3,a>	6	ca
<2,c>	7	bc
<4,a>	8	da
<5,c>	9	abc
<9,e>	10	abce

resultado decodificado

a b c d a b c a b c d a a b c a b c e



---

---

# Algoritmo LZW

## (Lempel – Ziv – Welch)

— Ing. Max Alejandro Antonio  
Cerna Flores —

---

---

# Agenda

Definición

¿Cómo trabaja?

Decodificación

# Definición

Compresión basada en diccionario, que no requiere información previa sobre el flujo de datos de entrada.

Se basa en patrones recurrentes para ahorrar espacio de datos, solo necesita un simple código o símbolo para representar una subcadena.

No tiene pérdidas, lo que significa que no se pierden datos al comprimir.

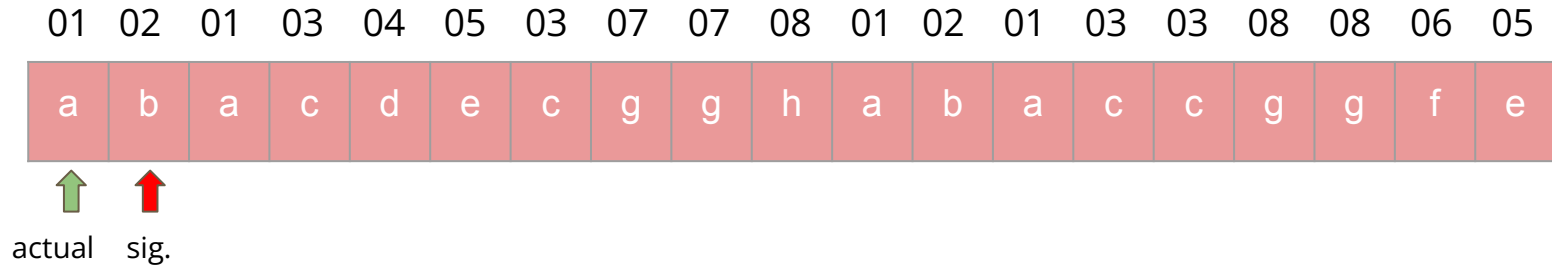
Se utiliza normalmente en GIF y, opcionalmente, en PDF y TIFF.

# ¿Cómo trabaja?

- Se define un diccionario dado, por ejemplo ASCII (8 bits - 256 caracteres)
- Para el ejemplo se trabajara con un diccionario de 8 caracteres (3 bits)

Bin	Dec	Sim
000	01	a
001	02	b
010	03	c
011	04	d
100	05	e
101	06	f
110	07	g
111	08	h

# ¿Cómo trabaja?



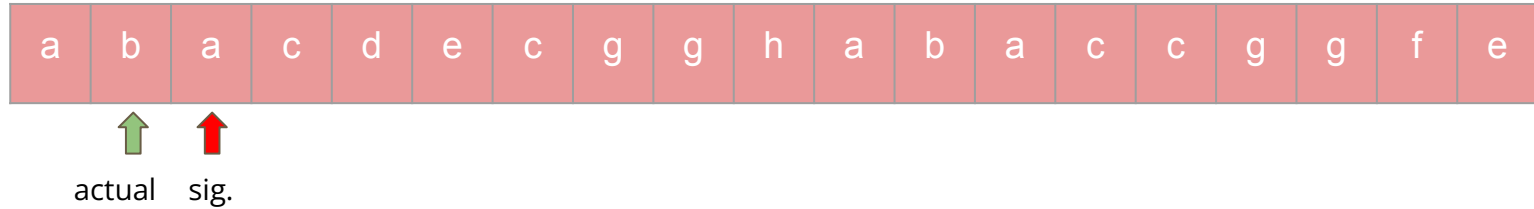
## Compresión (output)

01

## Diccionario (en memoria)

09 -> ab

# ¿Cómo trabaja?



## Compresión (output)

01 02

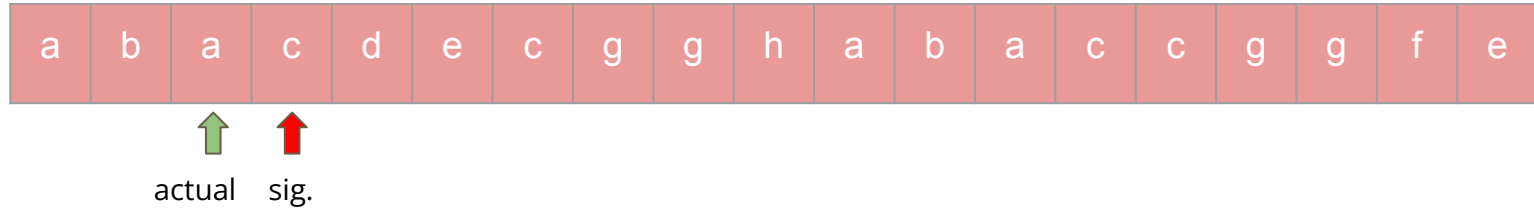
## Diccionario (en memoria)

09 -> ab

10 -> ba



# ¿Cómo trabaja?



## Compresión (output)

01 02 01

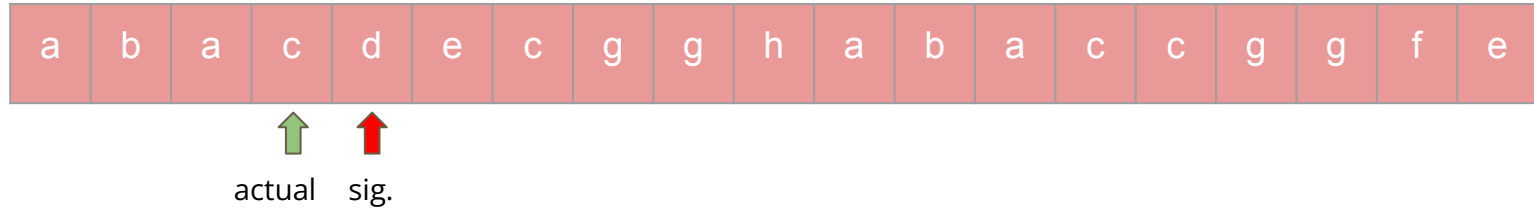
## Diccionario (en memoria)

09 -> ab

10 -> ba

11 -> ac

# ¿Cómo trabaja?



## Compresión (output)

01 02 01 03

## Diccionario (en memoria)

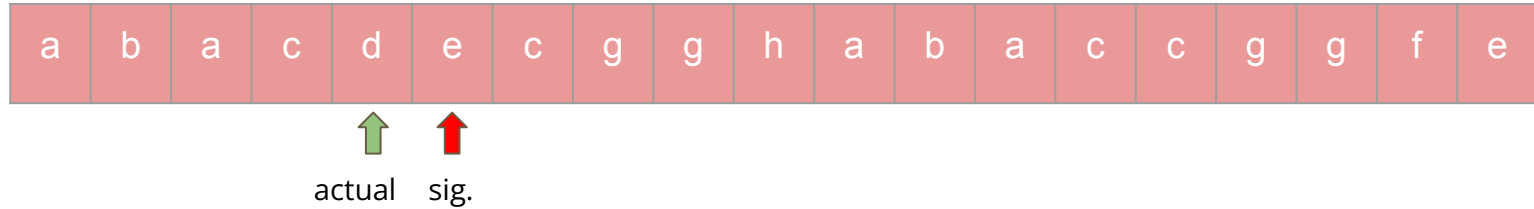
09 -> ab

10 -> ba

11 -> ac

12 -> cd

# ¿Cómo trabaja?



## Compresión (output)

01 02 01 03 04

## Diccionario (en memoria)

09 -> ab

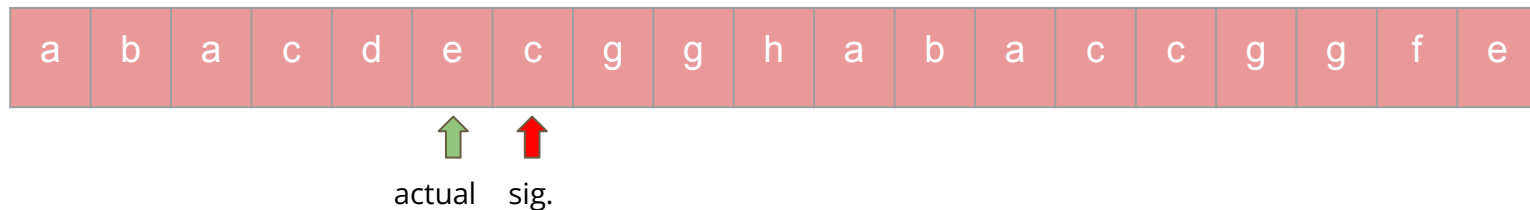
10 -> ba

11 -> ac

12 -> cd

13 -> de

# ¿Cómo trabaja?



## Compresión (output)

01 02 01 03 04 05

## Diccionario (en memoria)

09 -> ab

10 -> ba

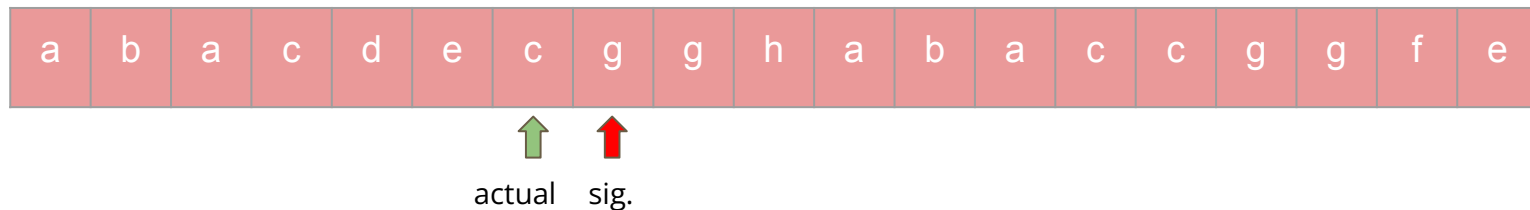
11 -> ac

12 -> cd

13 -> de

14 -> ec

# ¿Cómo trabaja?



## Compresión (output)

01 02 01 03 04 05 03

## Diccionario (en memoria)

09 -> ab

10 -> ba

11 -> ac

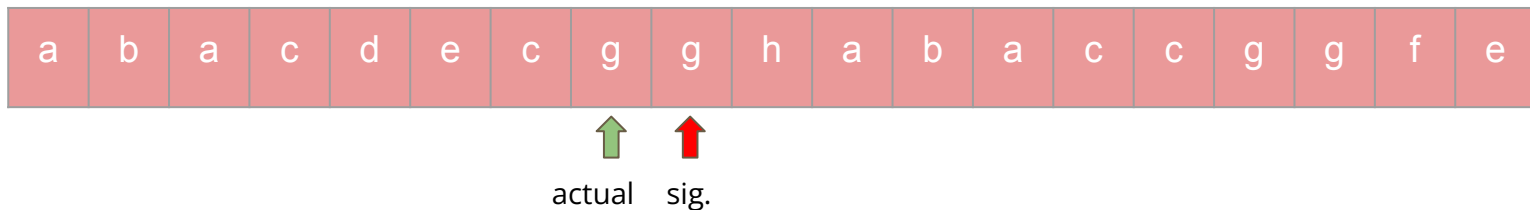
12 -> cd

13 -> de

14 -> ec

15 -> cg

# ¿Cómo trabaja?



## Compresión (output)

01 02 01 03 04 05 03 07

## Diccionario (en memoria)

09 -> ab

10 -> ba

11 -> ac

12 -> cd

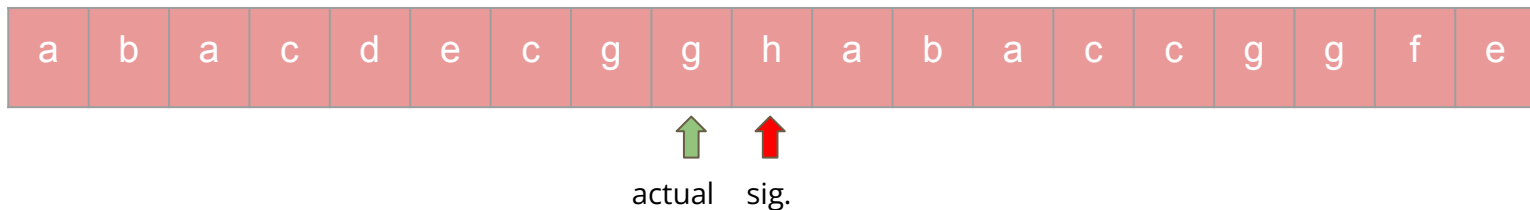
13 -> de

14 -> ec

15 -> cg

16 -> gg

# ¿Cómo trabaja?



## Compresión (output)

01 02 01 03 04 05 03 07 07

## Diccionario (en memoria)

09 -> ab

10 -> ba

11 -> ac

12 -> cd

13 -> de

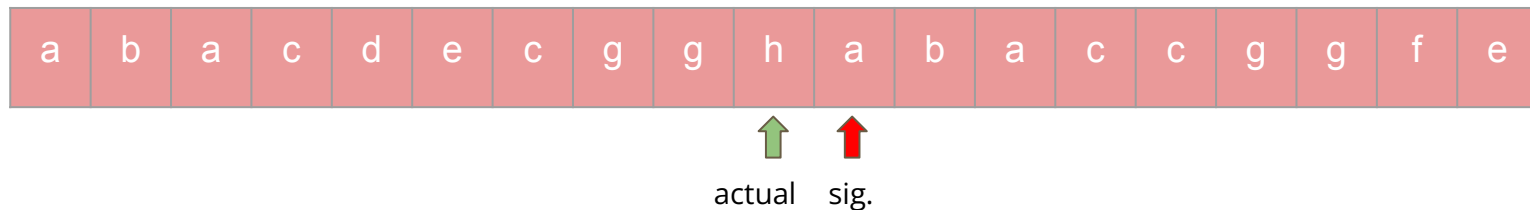
14 -> ec

15 -> cg

16 -> gg

17 -> gh

# ¿Cómo trabaja?



## Compresión (output)

01 02 01 03 04 05 03 07 07 08

## Diccionario (en memoria)

09 -> ab

10 -> ba

11 -> ac

12 -> cd

13 -> de

14 -> ec

15 -> cg

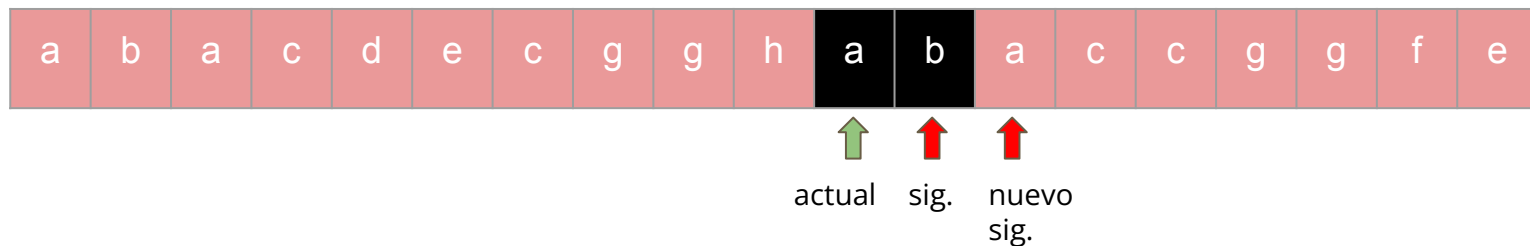
16 -> gg

17 -> gh

19 -> ha



# ¿Cómo trabaja?



## Compresión (output)

01 02 01 03 04 05 03 07 07 08 09

## Diccionario (en memoria)

09 -> ab

10 -> ba

11 -> ac

12 -> cd

13 -> de

14 -> ec

15 -> cg

16 -> gg

17 -> gh

18 -> ha

19 -> aba

# ¿Cómo trabaja?



## Compresión (output)

01 02 01 03 04 05 03 07 07 08 09 **11**

## Diccionario (en memoria)

09 -> ab

10 -> ba

**11 -> ac**

12 -> cd

13 -> de

14 -> ec

15 -> cg

16 -> gg

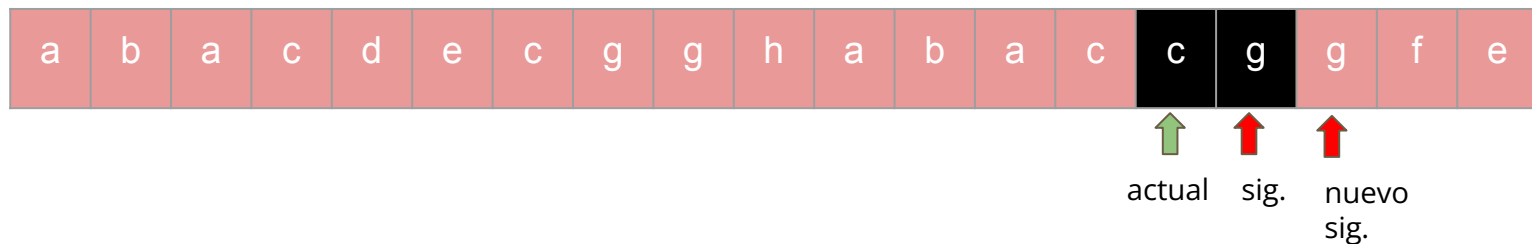
17 -> gh

18 -> ha

19 -> aba

**20 -> acc**

# ¿Cómo trabaja?



## Compresión (output)

01 02 01 03 04 05 03 07 07 08 09 11 15

## Diccionario (en memoria)

09 -> ab

10 -> ba

11 -> ac

12 -> cd

13 -> de

14 -> ec

15 -> **cg**

16 -> gg

17 -> gh

18 -> ha

29 -> aba

20 -> acc

21 -> **cgg**

# ¿Cómo trabaja?



## Compresión (output)

01 02 01 03 04 05 03 07 07 08 09 11 15 07

## Diccionario (en memoria)

09 -> ab

10 -> ba

11 -> ac

12 -> cd

13 -> de

14 -> ec

15 -> cg

16 -> gg

17 -> gh

18 -> ha

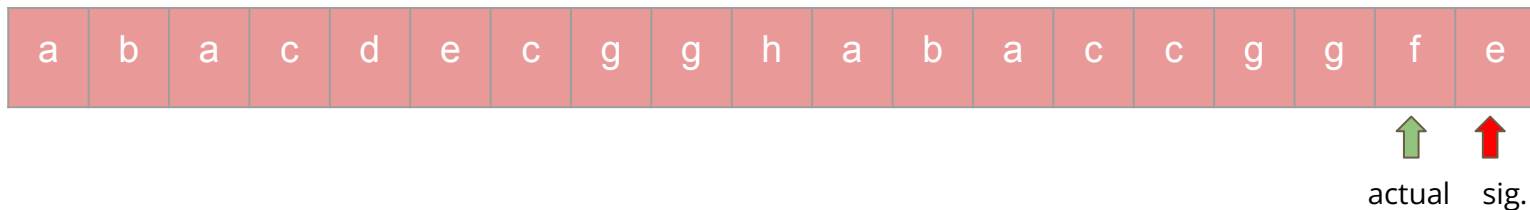
19 -> aba

20 -> acc

21 -> cgg

22 -> gf

# ¿Cómo trabaja?



## Compresión (output)

01 02 01 03 04 05 03 07 07 08 09 11 15 07 06

## Diccionario (en memoria)

09 -> ab

10 -> ba

11 -> ac

12 -> cd

13 -> de

14 -> ec

15 -> cg

16 -> gg

17 -> gh

18 -> ha

19 -> aba

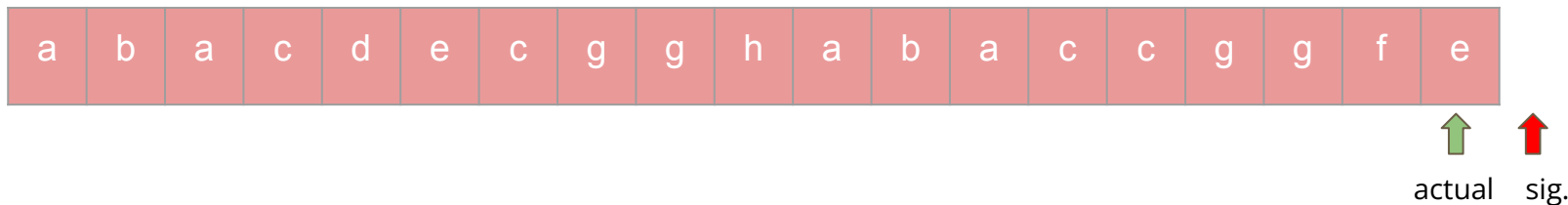
20 -> acc

21 -> cgg

22 -> gf

23 -> fe

# ¿Cómo trabaja?



## Compresión (output)

01 02 01 03 04 05 03 07 07 08 09 11 15 07 06 05 eof

## Diccionario (en memoria)

09 -> ab

10 -> ba

11 -> ac

12 -> cd

13 -> de

14 -> ec

15 -> cg

16 -> gg

17 -> gh

18 -> ha

19 -> aba

20 -> acc

21 -> cgg

22 -> gf

23 -> fe

# ¿Cómo trabaja?

## **input**

01 02 01 03 04 05 03 07 07 08 01 02 01 03 03 08 08 06 05 eof

## **Compresión (output)**

01 02 01 03 04 05 03 07 07 08 09 11 15 07 06 05 eof

# Descompresión

01	02	01	03	04	05	03	07	07	08	09	11	15	07	06	05
----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----



actual sig.

**output:**

01



**Diccionario (en memoria)**

09 -> 01 02 (ab)



# Descompresión

01	02	01	03	04	05	03	07	07	08	09	11	15	07	06	05
----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----

   
actual sig.

**output:**

01 02

**Diccionario (en memoria)**

09 -> 01 02 (ab)

10 -> 02 01 (ba)

# Descompresión



**output:**

01 02 01

**Diccionario (en memoria)**



09 -> 01 02 (ab)

10 -> 02 01 (ba)

11 -> 01 03 (ac)

# Descompresión

01	02	01	03	04	05	03	07	07	08	09	11	15	07	06	05
----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----

   
actual sig.

**output:**

01 02 01 03

## **Diccionario (en memoria)**

09 -> 01 02 (ab)

10 -> 02 01 (ba)

11 -> 01 03 (ac)

12 -> 03 04 (cd)

# Descompresión

01	02	01	03	04	05	03	07	07	08	09	11	15	07	06	05
----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----



actual



sig.

**output:**

01 02 01 03 04

**Diccionario (en memoria)**

09 -> 01 02 (ab)

10 -> 02 01 (ba)

11 -> 01 03 (ac)

12 -> 03 04 (cd)

13 -> 04 05

# Descompresión

01	02	01	03	04	05	03	07	07	08	09	11	15	07	06	05
----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----



actual

sig.

**output:**

01 02 01 03 04 05

## Diccionario (en memoria)

09 -> 01 02 (ab)

10 -> 02 01 (ba)

11 -> 01 03 (ac)

12 -> 03 04 (cd)

13 -> 04 05 (de)

14 -> 05 03 (ec)

# Descompresión

01	02	01	03	04	05	03	07	07	08	09	11	15	07	06	05
----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----



actual



sig.

**output:**

01 02 01 03 04 05 03

## Diccionario (en memoria)

09 -> 01 02 (ab)

10 -> 02 01 (ba)

11 -> 01 03 (ac)

12 -> 03 04 (cd)



13 -> 04 05 (de)

14 -> 05 03 (ec)

15 -> 03 07 (cg)

# Descompresión

01	02	01	03	04	05	03	07	07	08	09	11	15	07	06	05
----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----

   
actual sig.

**output:**

01 02 01 03 04 05 03 07

## Diccionario (en memoria)

09 -> 01 02 (ab)

10 -> 02 01 (ba)

11 -> 01 03 (ac)

12 -> 03 04 (cd)

13 -> 04 05 (de)



14 -> 05 03 (ec)

15 -> 03 07 (cg)

16 -> 07 07 (gg)

# Descompresión

01	02	01	03	04	05	03	07	07	08	09	11	15	07	06	05
----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----

   
actual sig.

**output:**

01 02 01 03 04 05 03 07 07

## Diccionario (en memoria)

09 -> 01 02 (ab)

10 -> 02 01 (ba)

11 -> 01 03 (ac)

12 -> 03 04 (cd)

13 -> 04 05 (de)

14 -> 05 03 (ec)

15 -> 03 07 (cg)



16 -> 07 07 (gg)

17 -> 07 08 (gh)



# Descompresión

01	02	01	03	04	05	03	07	07	08	09	11	15	07	06	05
----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----

   
actual sig.

**output:**

01 02 01 03 04 05 03 07 07 08

## Diccionario (en memoria)

09 -> 01 02 (ab)

10 -> 02 01 (ba)

11 -> 01 03 (ac)

12 -> 03 04 (cd)

13 -> 04 05 (de)

14 -> 05 03 (ec)

15 -> 03 07 (cg)

16 -> 07 07 (gg)

17 -> 07 08 (gh)

18 -> 08 09 (hab)

# Descompresión

01	02	01	03	04	05	03	07	07	08	09	11	15	07	06	05
----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----



actual

sig.

**output:**

01 02 01 03 04 05 03 07 07 08 **01 02**

## Diccionario (en memoria)

**09 -> 01 02 (ab)**

10 -> 02 01 (ba)

11 -> 01 03 (ac)

12 -> 03 04 (cd)

13 -> 04 05 (de)

14 -> 05 03 (ec)

15 -> 03 07 (cg)

16 -> 07 07 (gg)

17 -> 07 08 (gh)

18 -> 08 09 (hab)

19 -> 09 11 (abac)

# Descompresión

01	02	01	03	04	05	03	07	07	08	09	11	15	07	06	05
----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----



actual



sig.

**output:**

01 02 01 03 04 05 03 07 07 08 01 02 **01 03**

## Diccionario (en memoria)

09 -> 01 02 (ab)

10 -> 02 01 (ba)

**11 -> 01 03 (ac)**

12 -> 03 04 (cd)

13 -> 04 05 (de)

14 -> 05 03 (ec)

15 -> 03 07 (cg)

16 -> 07 07 (gg)

17 -> 07 08 (gh)

18 -> 08 09 (hab)

19 -> 09 11 (abac)

20 -> 11 15 (accg)

# Descompresión

01	02	01	03	04	05	03	07	07	08	09	11	15	07	06	05
----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----



actual



sig.

**output:**

01 02 01 03 04 05 03 07 07 08 01 02 01 03 **03 07**

## Diccionario (en memoria)

09 -> 01 02 (ab)

10 -> 02 01 (ba)

11 -> 01 03 (ac)

12 -> 03 04 (cd)

13 -> 04 05 (de)

14 -> 05 03 (ec)

**15 -> 03 07 (cg)**

16 -> 07 07 (gg)

17 -> 07 08 (gh)

18 -> 08 09 (hab)

19 -> 09 11 (abac)

20 -> 11 15 (accg)

21 -> 15 07 (cgg)

# Descompresión

01	02	01	03	04	05	03	07	07	08	09	11	15	07	06	05
----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----



actual



sig.

**output:**

01 02 01 03 04 05 03 07 07 08 01 02 01 03 03 07 07

## Diccionario (en memoria)

09 -> 01 02 (ab)

10 -> 02 01 (ba)

11 -> 01 03 (ac)

12 -> 03 04 (cd)

13 -> 04 05 (de)

14 -> 05 03 (ec)

15 -> 03 07 (cg)

16 -> 07 07 (gg)

17 -> 07 08 (gh)

18 -> 08 09 (hab)

19 -> 09 11 (abac)

20 -> 11 15 (accg)

21 -> 15 07 (cgg)

22 -> 07 06 (gf)

# Descompresión

01	02	01	03	04	05	03	07	07	08	09	11	15	07	06	05
----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----



actual



sig.

**output:**

01 02 01 03 04 05 03 07 07 08 01 02 01 03 03 07 07 06

## Diccionario (en memoria)

09 -> 01 02 (ab)

10 -> 02 01 (ba)

11 -> 01 03 (ac)

12 -> 03 04 (cd)

13 -> 04 05 (de)

14 -> 05 03 (ec)

15 -> 03 07 (cg)

16 -> 07 07 (gg)

17 -> 07 08 (gh)

18 -> 08 09 (hab)

19 -> 09 11 (abac)

20 -> 11 15 (accg)

21 -> 15 07 (cgg)

22 -> 07 06 (gf)

23 -> 06 05 (fe)

# Descompresión

01	02	01	03	04	05	03	07	07	08	09	11	15	07	06	05
----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----



actual



sig.

**output:**

01 02 01 03 04 05 03 07 07 08 01 02 01 03 03 07 07 06 05

## Diccionario (en memoria)

09 -> 01 02 (ab)

10 -> 02 01 (ba)

11 -> 01 03 (ac)

12 -> 03 04 (cd)

13 -> 04 05 (de)

14 -> 05 03 (ec)

15 -> 03 07 (cg)

16 -> 07 07 (gg)

17 -> 07 08 (gh)

18 -> 08 09 (hab)

19 -> 09 11 (abac)

20 -> 11 15 (accg)

21 -> 15 07 (cgg)

22 -> 07 06 (gf)

23 -> 06 05 (fe)

# Descompresión

## compresión

01 02 01 03 04 05 03 07 07 08 09 11 15 07 06 05

## resultado

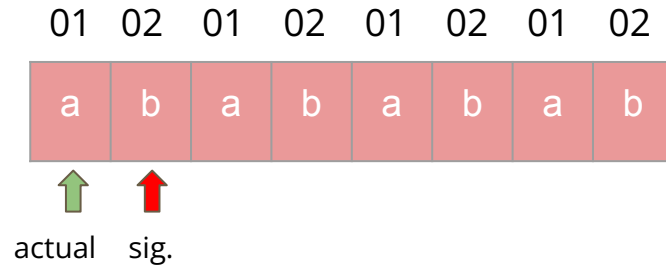
01 02 01 03 04 05 03 07 07 08 01 02 01 03 03 07 07 06 05  
a b a c d e c g g h a b a c c g g f e

## mensaje original

a	b	a	c	d	e	c	g	g	h	a	b	a	c	c	g	g	f	e
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---



# Otro ejemplo



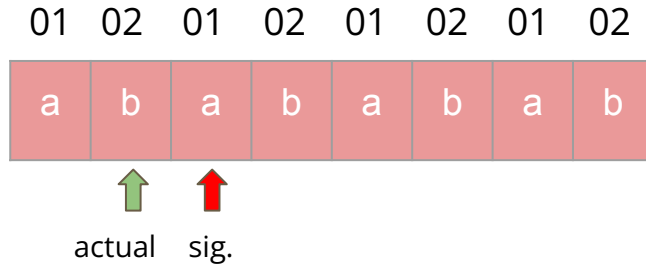
## Compresión (output)

01

## Diccionario (en memoria)

09 -> ab

# Otro ejemplo



## Compresión (output)

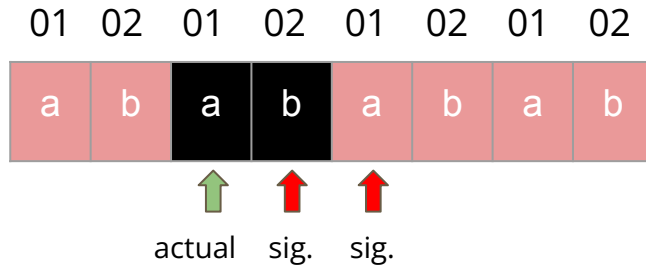
01 02

## Diccionario (en memoria)

09 -> ab

10 -> ba

## Otro ejemplo



### Compresión (output)

01 02 09

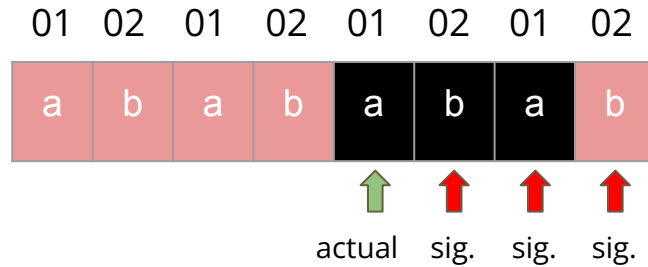
### Diccionario (en memoria)

09 -> ab

10 -> ba

11 -> aba

## Otro ejemplo



### Compresión (output)

01 02 09 11

### Diccionario (en memoria)

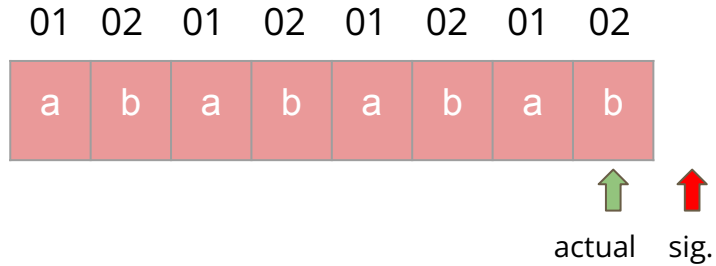
09 -> ab

10 -> ba

11 -> aba

12 -> abab

# Otro ejemplo



## Compresión (output)

01 02 09 11 02

## Diccionario (en memoria)

09 -> ab

10 -> ba

11 -> aba

12 -> abab

# Otro ejemplo

**input**

01 02 01 02 01 02 01 02

**Compresión (output)**

01 02 09 11 02

# Descompresión

01	02	09	11	02
----	----	----	----	----



actual sig.

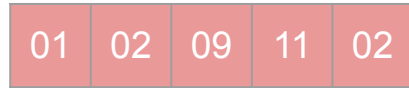
**output:**

01

**Diccionario (en memoria)**

09 -> 01 02 (ab)

# Descompresión



**output:**

01 02

**Diccionario (en memoria)**

09 -> 01 02 (ab)

10 -> 02 01 (ba)



# Descompresión

01	02	09	11	02
----	----	----	----	----

↑    ↑  
actual   sig.

**output:**

01 02 **01 02**

**Diccionario (en memoria)**

09 -> 01 02 (ab)

10 -> 02 01 (ba)



11 -> 09 11 ?

**Solución**

09 (ab) + 01 (a) -> 11 => aba

# Descompresión

01	02	09	11	02
----	----	----	----	----

   
actual sig.

**output:**

01 02 01 02 **01 02 01**

**Diccionario (en memoria)**

09 -> 01 02 (ab)

10 -> 02 01 (ba)

**11 -> 09 01 (aba)**

# Descompresión

01	02	09	11	02
----	----	----	----	----



actual



sig.

**output:**

01 02 01 02 01 02 01 **02**

**Diccionario (en memoria)**

09 -> 01 02 (ab)

10 -> 02 01 (ba)

11 -> 09 01 (aba)

# Descompresión

## compresión

01 02 09 11 02

## resultado

01 02 01 02 01 02 01 02  
a b a b a b a b

## mensaje original

a	b	a	b	a	b	a	b
---	---	---	---	---	---	---	---

---

---

# Algoritmo LZSS

## (Lempel – Ziv – Storer – Szymanski)

— Ing. Max Alejandro Antonio  
Cerna Flores —

---

---

# Agenda

Definición

Diferencia con LZ77

¿Cómo trabaja?

Decodificación

# Definición

Es un algoritmo de compresión de datos sin pérdidas.

Un derivado de LZ77, que fue creado en 1982 por James A. Storer y Thomas Szymanski.

Es una técnica de codificación por diccionario.

Intenta reemplazar una cadena de símbolos con una referencia a una ubicación en el diccionario de la misma cadena.

# Diferencia entre LZSS y LZ77

La principal diferencia entre LZ77 y LZSS es que en LZ77 la referencia del diccionario en realidad podría ser más larga que la cadena que estaba reemplazando.

Si ese fuera el caso, dicha referencia será omitida.



# ¿Cómo trabaja?

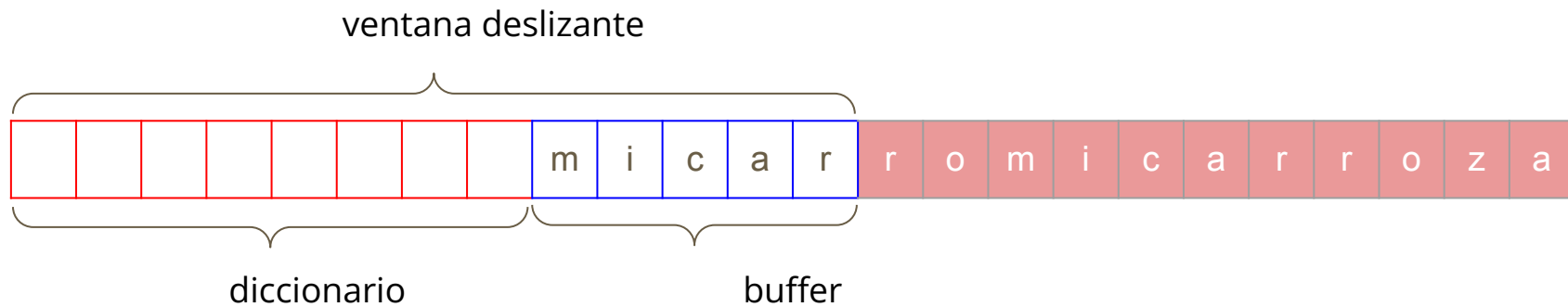
- Se define un buffer de avance de tamaño dado.
- Se define un diccionario
- Ejemplo:
  - Buffer = 5 

--	--	--	--	--
  - diccionario = 8 

--	--	--	--	--	--	--	--
  - Puntero ↑
  - dupla [posición, longitud]

m	i	c	a	r	r	o	m	i	c	a	r	r	o	z	a
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

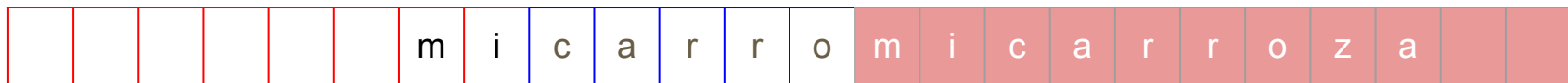
# ¿Cómo trabaja?



# ¿Cómo trabaja?

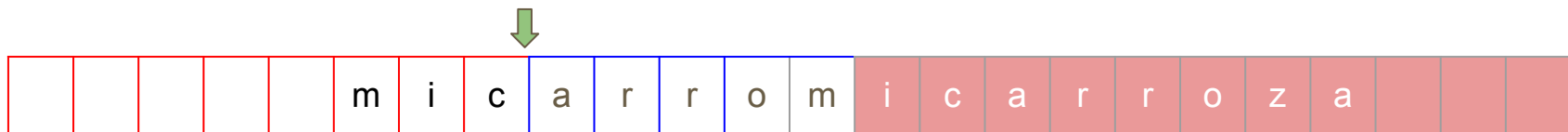


salida: m

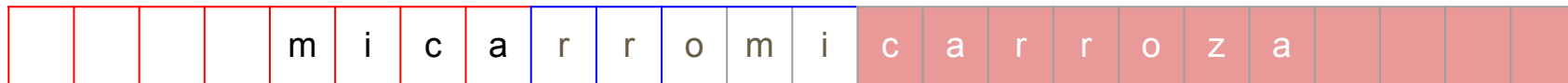


salida: mi

# ¿Cómo trabaja?



salida: mic

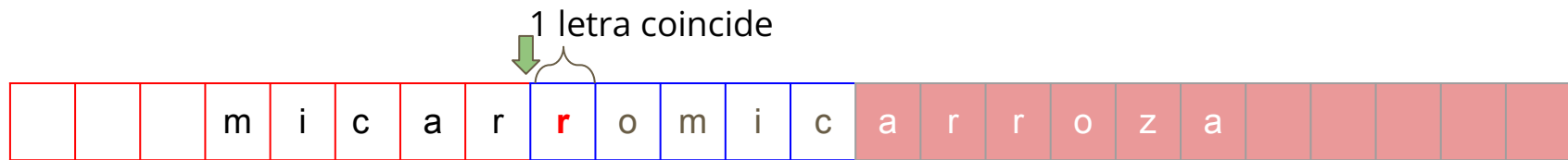


salida: mica

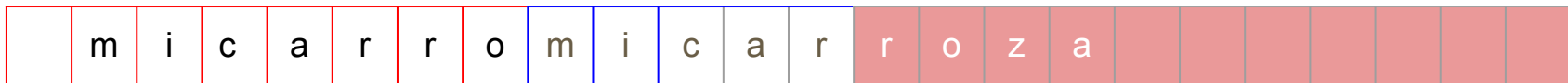
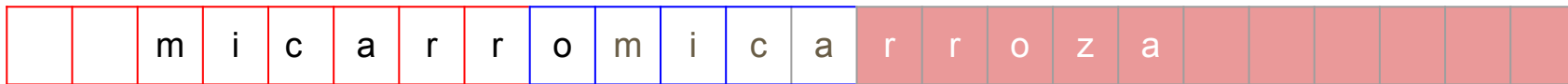


salida: micar

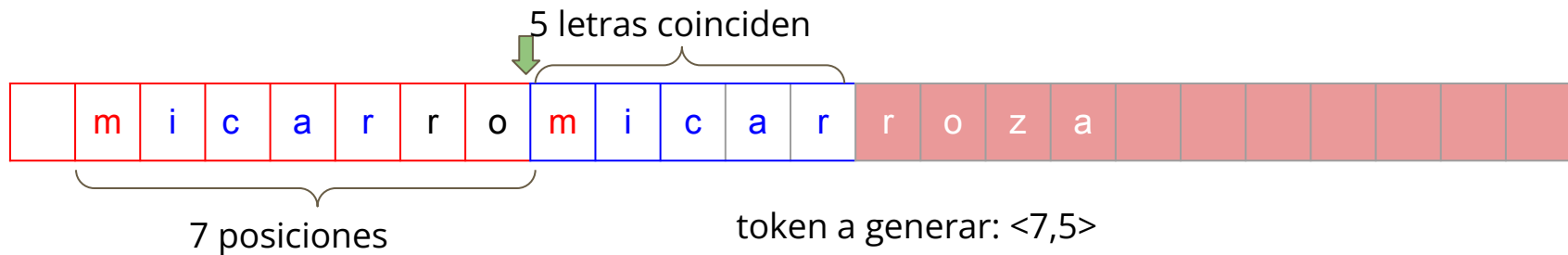
# ¿Cómo trabaja?



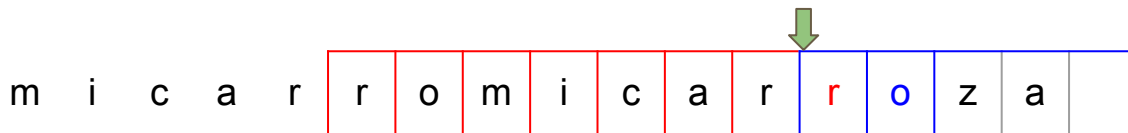
token a generar:  $\langle 1, 1 \rangle$  **X (más largo que el texto a reemplazar)**



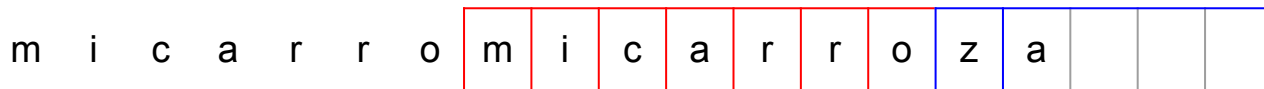
# ¿Cómo trabaja?



salida: micarro<7,5>



salida: micarro<7,5>ro



# ¿Cómo trabaja?


m i c a r r o m 

i	c	a	r	r	o	z	a				
---	---	---	---	---	---	---	---	--	--	--	--

salida: micarro<7,5>roz

m i c a r r o m 

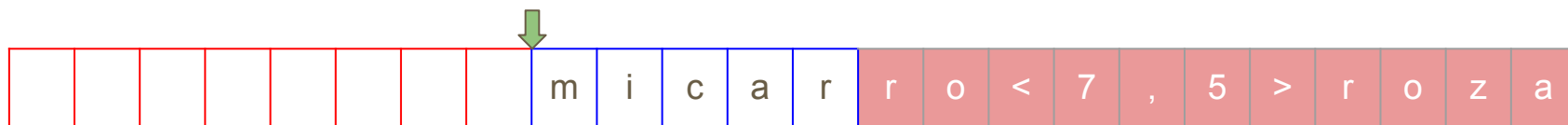
i	c	a	r	r	o	z	a				
---	---	---	---	---	---	---	---	--	--	--	--



salida: **micarro<7,5>roza**

# Descompresión

entrada: micarro<7,5>rosa



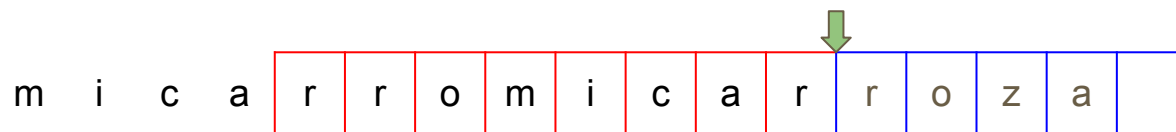
...



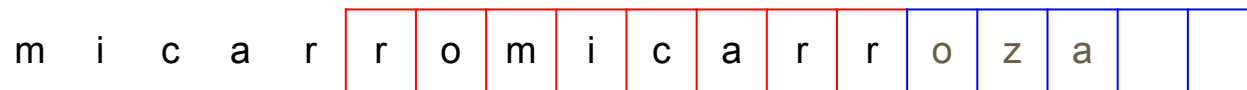
inicio de token



# Descompresión



salida: micarromicar

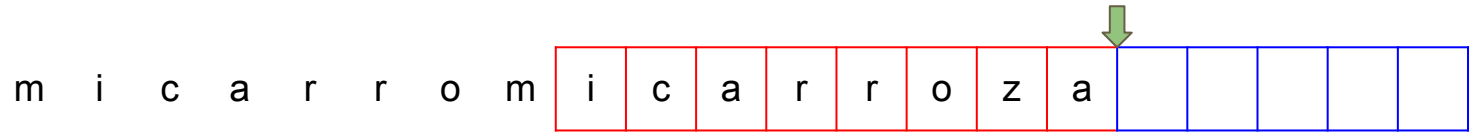


salida: micarromicarr

...

# Descompresión

...



salida: micarromicarroza

# Hoja de Trabajo (LZW y LZSS)

Comprimir el mensaje con LZW:

COMPADRE\_NO\_COMPRO\_COCO

considere el diccionario:

Bin	Dec	Sim
0000	00	a
0001	01	c
0010	02	d
0011	03	e
0100	04	m
0101	05	n
0110	06	o
0111	07	p
1000	08	r
1001	09	—

# Hoja de Trabajo (LZW y LZSS)

Comprimir el mensaje con LZSS (buffer: 6, diccionario 8)

YO\_SOY\_YO\_Y\_YO\_SOY\_ESTUDIANTE\_QUE\_ESTUDIA