7. Mapas: Manejo de colisiones en tablas de dispersión

Carlos Zerón Martínez

Universidad Nacional Autónoma de México

zeron@ciencias.unam.mx

Martes 5 de Enero de 2021

Introducción

La dispersión de entradas en una tabla de dispersión consiste en asociar una entrada (k, v) a una localidad en un arreglo A de cajones, donde k es una clave y v un elemento. Cuando se tienen dos claves k_1 y k_2 tales que $h(k_1) = h(k_2)$, se dice que hay una **colisión**.

La existencia de colisiones nos impide asociar directamente la entrada (k, v) en la localidad $A[h(k)]$, por lo que las implementaciones de mapa con tablas de dispersión, requieren de que sepamos cómo actuar cuando se presenten colisiones.

Técnicas de manejo de colisiones

Tenemos dos enfoques:

- **Encadenamiento separado**. Se cambia la estructura de la tabla de dispersión de modo que cada localidad pueda ahora tener un contenedor para varias claves.

- **Direccionamiento abierto**. Se emplea una localidad alternativa dentro de la misma tabla de dispersión al efectuar búsquedas, inserciones y eliminaciones de entradas por clave, sin requerir modificar la estructura de la tabla de dispersión, es decir, se mantiene el arreglo de cajones intacto.

Encadenamiento separado

Consiste en almacenar por cada cajón $A[i]$ una lista que tenga a todas las entradas mapeadas al cajón por la función de dispersión, esto es, los elementos (k, v) tales que $h(k) = i$.

$\Rightarrow$ La tabla hash queda entonces como un arreglo de listas.

Si cada cajón $A[i]$ inicia con una lista vacía, podemos implementar las operaciones más importantes como sigue:

- get(k). Se ubica el cajón al cual se asignan las entradas con clave k, a partir de la función de asociación y luego se hace una búsqueda sobre la lista correspondiente.
- put(k, v). Se localiza el cajón al cual se asignan las entradas con clave k, a partir de la función de asociación. Si existe una entrada con clave k en la lista correspondiente al cajón, reemplaza el valor original con v, de otro modo, se inserta la entrada nueva en la lista.

Encadenamiento separado

Consiste en almacenar por cada cajón $A[i]$ una lista que tenga a todas las entradas mapeadas al cajón por la función de dispersión, esto es, los elementos (k, v) tales que $h(k) = i$.

$\Rightarrow$ La tabla hash queda entonces como un arreglo de listas.

Si cada cajón $A[i]$ inicia con una lista vacía, podemos implementar las operaciones más importantes como sigue:

- remove(k). Lo mismo que la operación put pero en cuanto se encuentre una entrada con clave k, se elimina de la lista correspondiente.

Encadenamiento separado

En el peor caso, las operaciones sobre un cajón específico toman tiempo proporcional al número de entradas que tenga. Si la función de dispersión distribuye adecuadamente n entradas en una tabla de dispersión de tamaño N, se *espera* que el tamaño de la lista de cada cajón sea n/N (este radio es conocido como el **factor de carga** de la tabla de dispersión).

Idealmente $n/N < 1$ y en esas condiciones se *espera* que las operaciones se ejecuten en tiempo $O(1)$. Si $n > N$ y la función de dispersión no es buena, las operaciones en el peor caso se ejecutan en tiempo $O(n)$ (todas las entradas quedan en el mismo cajón).

Implementación sencilla pero tiene desventaja en programas que se ejecutan en dispositivos que caben en una mano, donde se requiere el mayor ahorro de memoria posible que no se tiene por la alteración a la estructura como arreglo de listas.

Encadenamiento separado

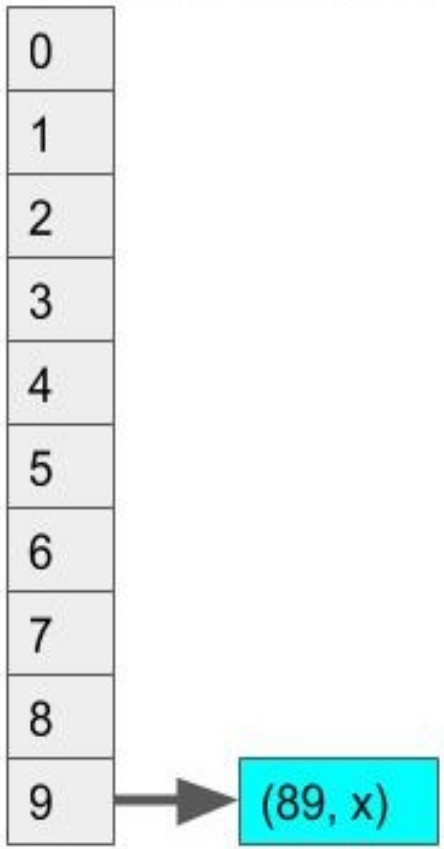
Ejemplo con una tabla de tamaño 10, una función de dispersión que consiste únicamente en una función de compresión de una clave entera.

Entradas = [(**89**,x), (**18**,y) ,(**49**, a), (**58**, e), (**69**, r)]

```
put(89, x)
```

h(k) = k mod 10

h(89) = 89 mod 10 = 9



Encadenamiento separado

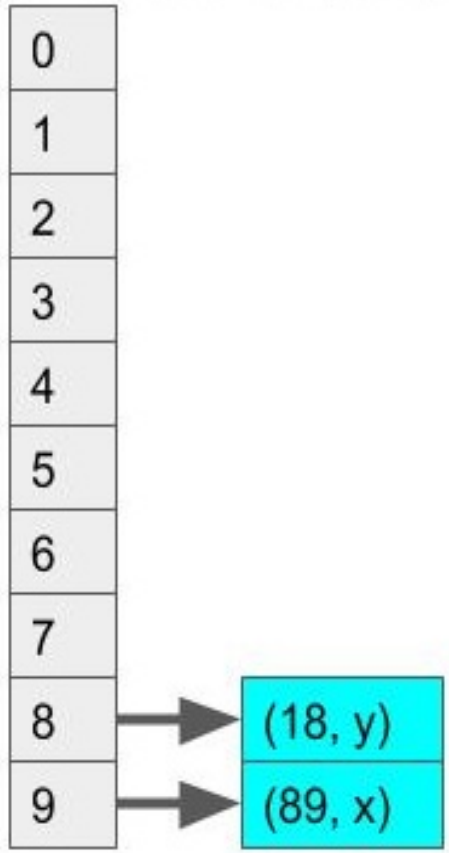
Ejemplo con una tabla de tamaño 10, una función de dispersión que consiste únicamente en una función de compresión de una clave entera.

Entradas = [(**89**,x), (**18**,y) ,(**49**, a), (**58**, e), (**69**, r)]

```
put(18, y)
```

h(k) = k mod 10

h(18) = 18 mod 10 = 8

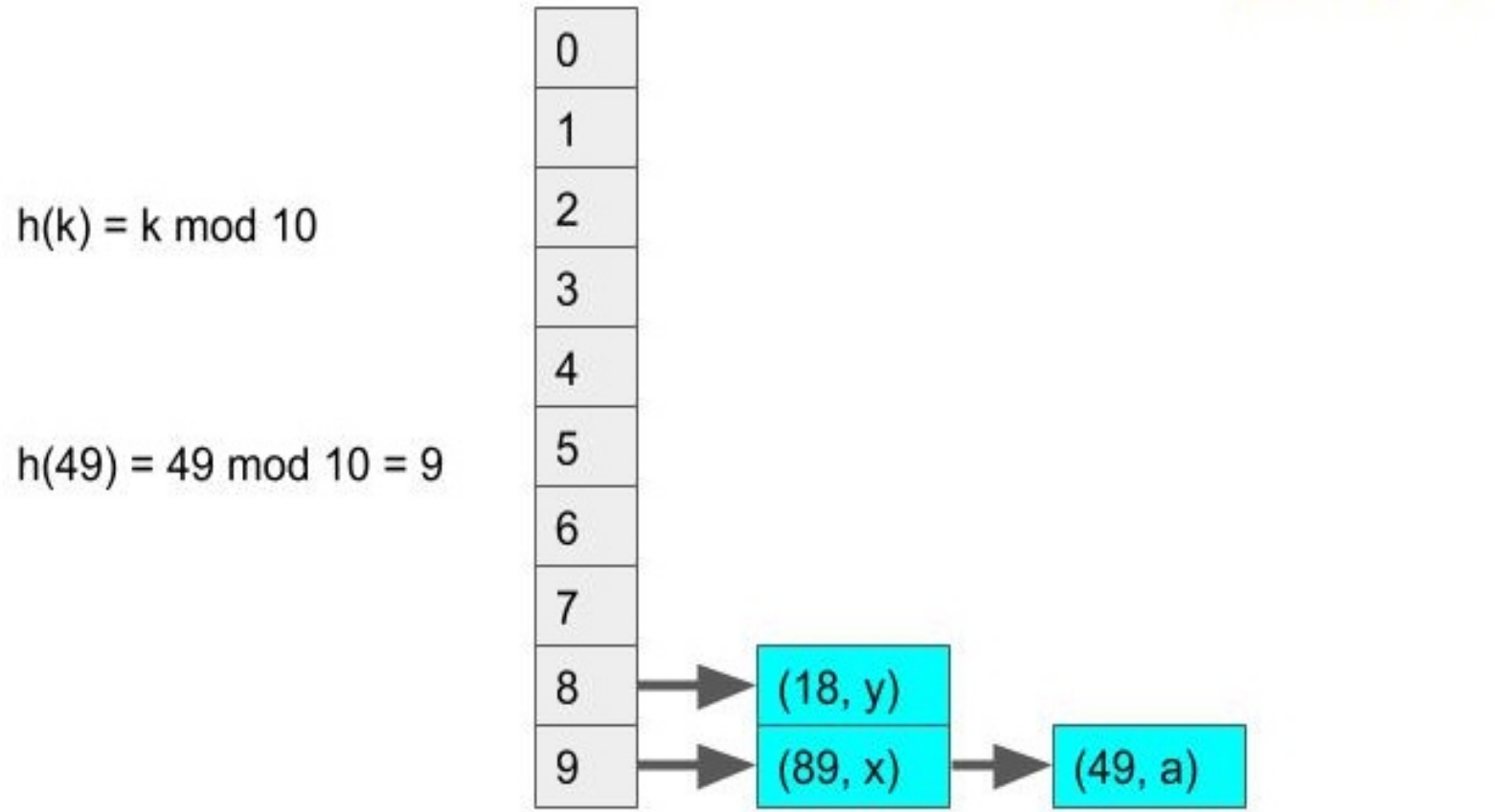


Encadenamiento separado

Ejemplo con una tabla de tamaño 10, una función de dispersión que consiste únicamente en una función de compresión de una clave entera.

Entradas = [(**89**,x), (**18**,y) ,(**49**, a), (**58**, e), (**69**, r)]

```
put(49, a)
```

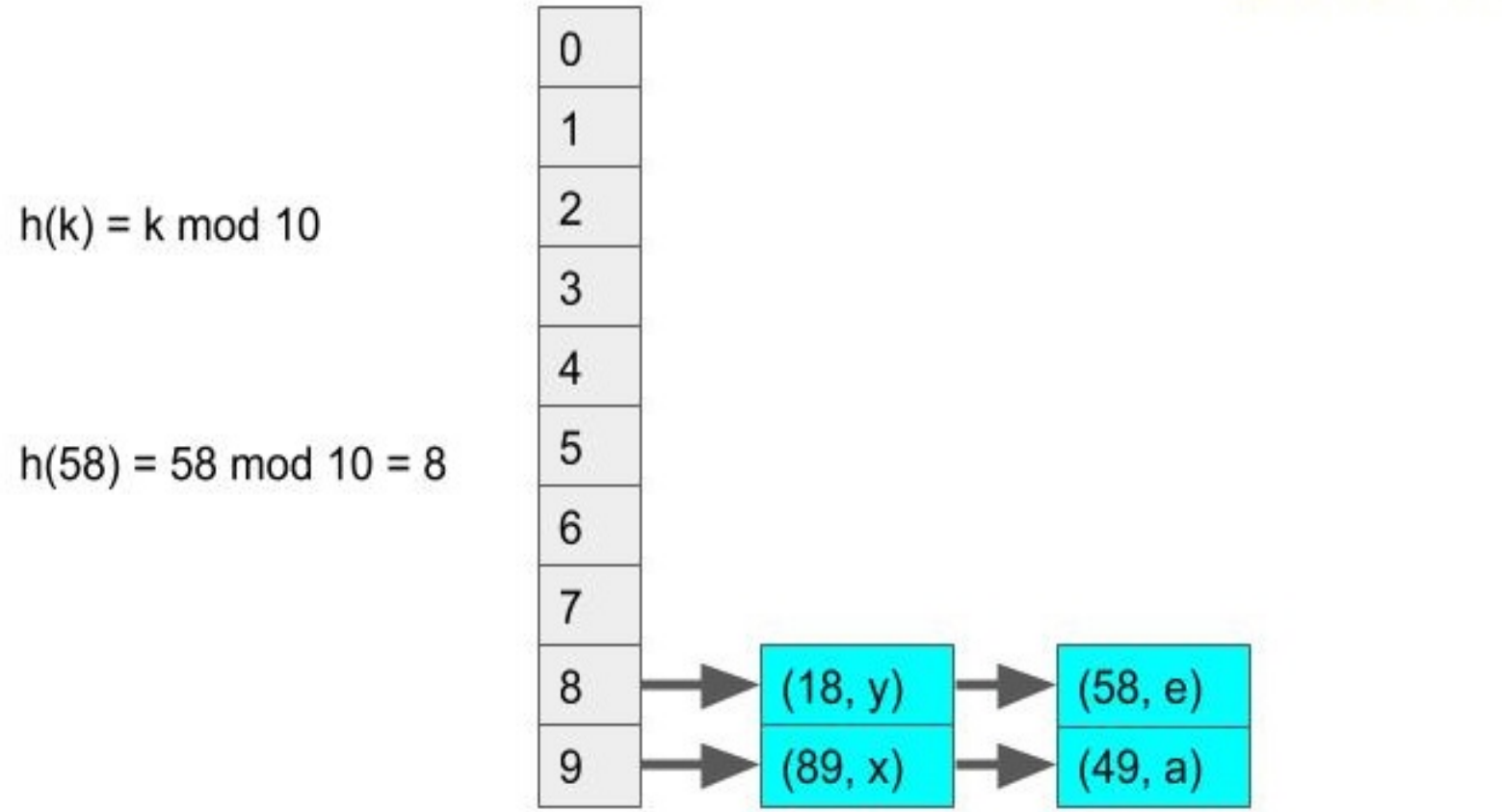


Encadenamiento separado

Ejemplo con una tabla de tamaño 10, una función de dispersión que consiste únicamente en una función de compresión de una clave entera.

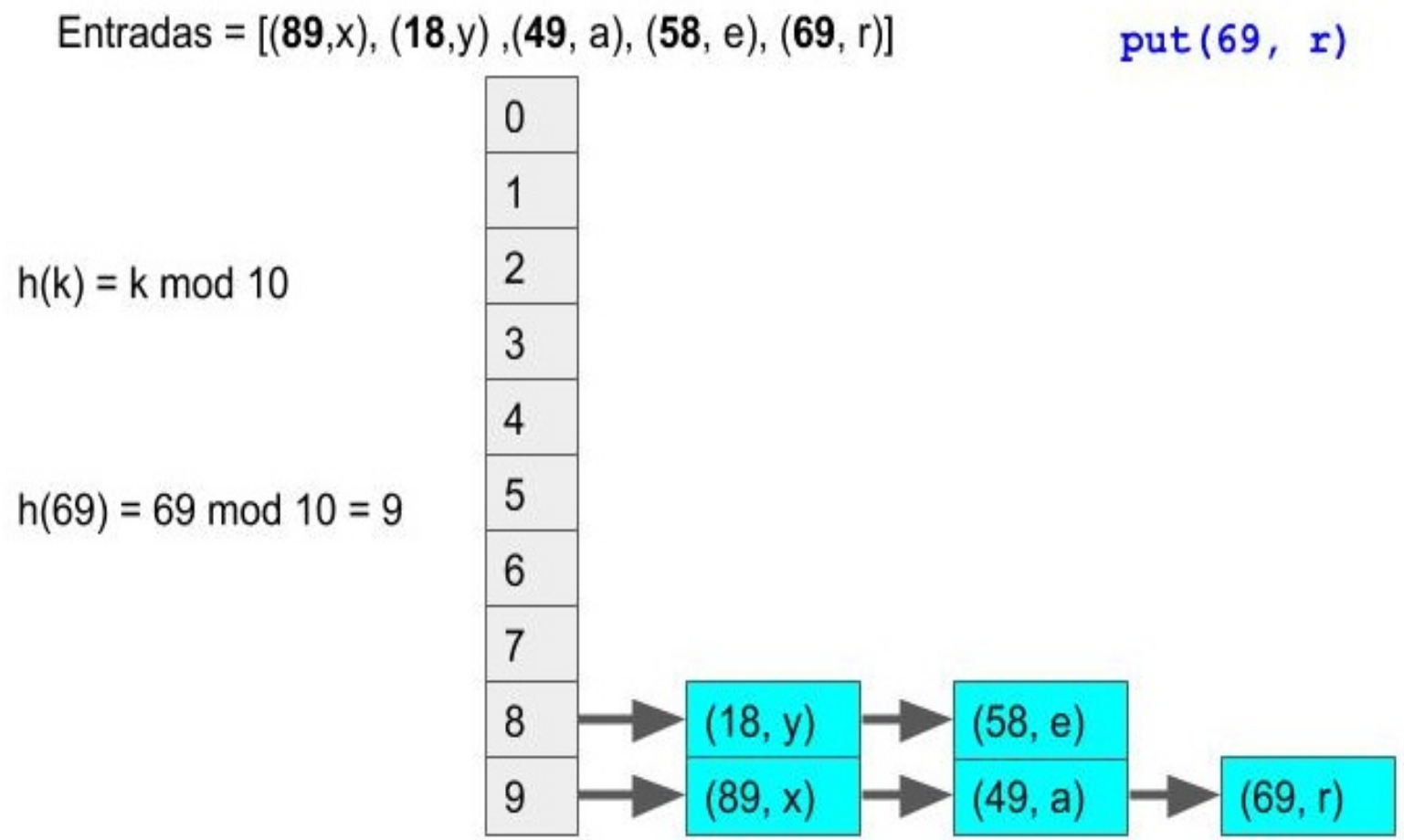
Entradas = [(**89**,x), (**18**,y) ,(**49**, a), (**58**, e), (**69**, r)]

```
put(58, e)
```



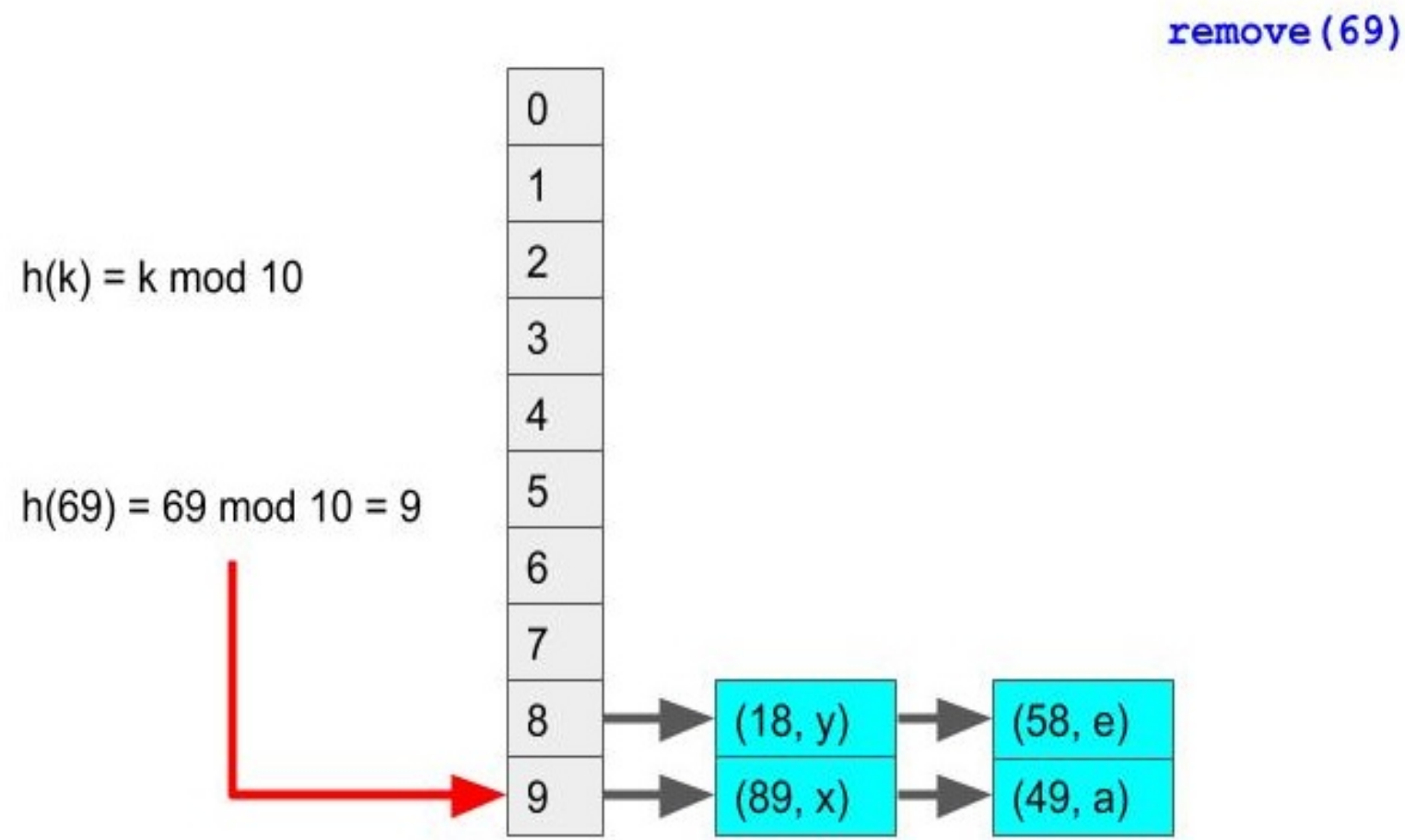
Encadenamiento separado

Ejemplo con una tabla de tamaño 10, una función de dispersión que consiste únicamente en una función de compresión de una clave entera.



Encadenamiento separado

Ejemplo con una tabla de tamaño 10, una función de dispersión que consiste únicamente en una función de compresión de una clave entera.



Direccionamiento abierto

Consiste en encontrar una localidad disponible dentro de la tabla de dispersión, proceso que recibe el nombre de secuencia de tanteos o pruebas. Veremos tres variantes de este enfoque. Se recomienda que el factor de carga de la tabla n/N sea siempre a lo más $1/2$.

En la inserción de una entrada, si se presenta una colisión, se prueban otros cajones hasta que se encuentra uno vacío para colocarla, o bien, el número de pruebas sobrepasa el tamaño de la tabla (la entrada ya no cabe).

Formalmente, si k es una clave, se prueba una sucesión de cajones

$$s_i(k) = \big(h(k) + f(i)\big) \ mod \ N$$

donde $i <= N$, $h(k)$ es la función de dispersión y f es una función de solución de colisiones (dependiendo de la estrategia de direccionamiento abierto que se escoja). Naturalmente $f(0) = 0$.

Direccionamiento abierto

Para localizar una entrada por clave se hace la misma sucesión de pruebas que en la inserción y termina de forma exitosa si encuentra un elemento con clave k y fracasa si el número de pruebas ha sobrepasado el tamaño de la tabla o el cajón examinado es nulo. Esto último porque la inserción busca el primer cajón vacio. Este enfoque es recomendable seguirlo si **no hay eliminaciones de claves**.

La eliminación se complica: no es posible dejar marcado el cajón como vacío o nulo porque podría ocasionar el fracaso en la búsqueda de claves que sí se encuentren en la tabla. Una solución es manejar estados para cada cajón: con entrada nula o con estrada no nula y este caso se divide en dos: el cajón está ocupado o tiene una entrada borrada (cajón borrado).

Direccionamiento abierto

- Inserción: Un cajón borrado se trata como vacío y se puede volver a insertar en él: si se presenta una colisión, se prueban otros cajones hasta que se encuentra uno vacío para colocarla, o bien, el número de pruebas sobrepasa el tamaño de la tabla. Al insertar la entrada, el cajón se marca como ocupado.

- Búsqueda: No requiere modificarse, puesto que los cajones marcados como borrados no detendrían la búsqueda, sólo la detienen los cajones con entrada nula.

- Eliminación: Igual que la búsqueda, pero en cuanto se encuentre una entrada con la clave a eliminar, se quita la entrada y se marca el cajón como borrado.

Tanteo lineal (linear probing)

- Estrategia de direccionamiento abierto que intenta situar a la entrada que colisiona en la siguiente localidad disponible (de forma circular): Si $h(k) = i$ y la localidad $A[i]$ está ocupada, el tanteo i-ésimo ocurre en la posición $s_i(k) = (h(k) + i)\ mod\ N$, donde $i <= N$.

- Se ahorra espacio con respecto a la implementación del encadenamiento separado, pero hay tendencia a agrupar las entradas en posiciones contiguas que alentan la eficiencia práctica de las operaciones considerablemente, especialmente cuando más de la mitad de los cajones se encuentran ocupados. Estos agrupamientos se conocen como**primarios**.

Tanteo lineal (linear probing)

Ejemplo con una tabla de tamaño 10, una función de dispersión que consiste únicamente en una función de compresión de una clave entera.

```
put(89, x)
```

h(k) = k mod 10

h(89) = 89 mod 10 = 9

0	
1	
2	
3	
4	
5	
6	
7	
8	
9	(89, x)

Tanteo lineal (linear probing)

Ejemplo con una tabla de tamaño 10, una función de dispersión que consiste únicamente en una función de compresión de una clave entera.

```
put(18, y)
```

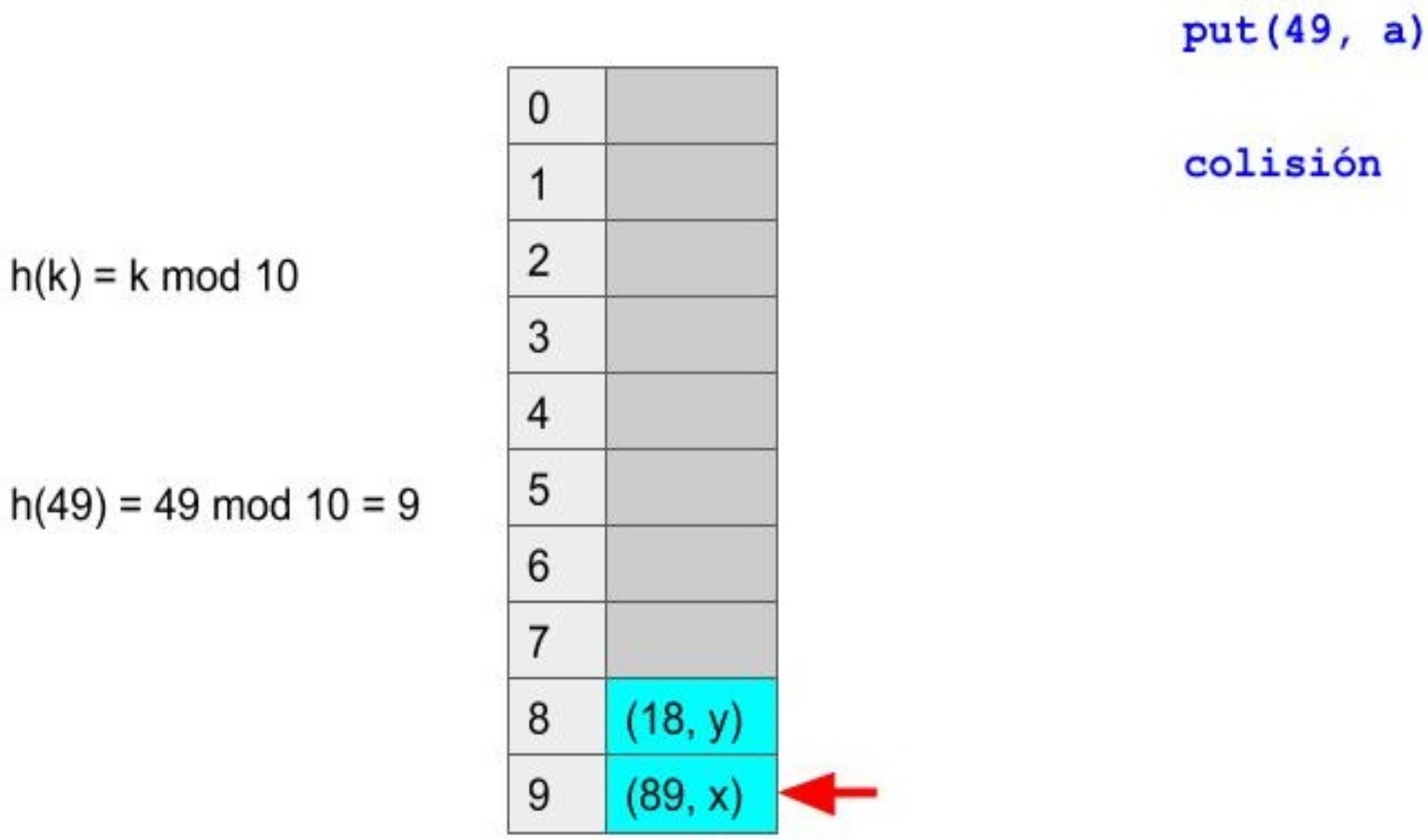
h(k) = k mod 10

h(18) = 18 mod 10 = 8

0	
1	
2	
3	
4	
5	
6	
7	
8	(18, y)
9	(89, x)

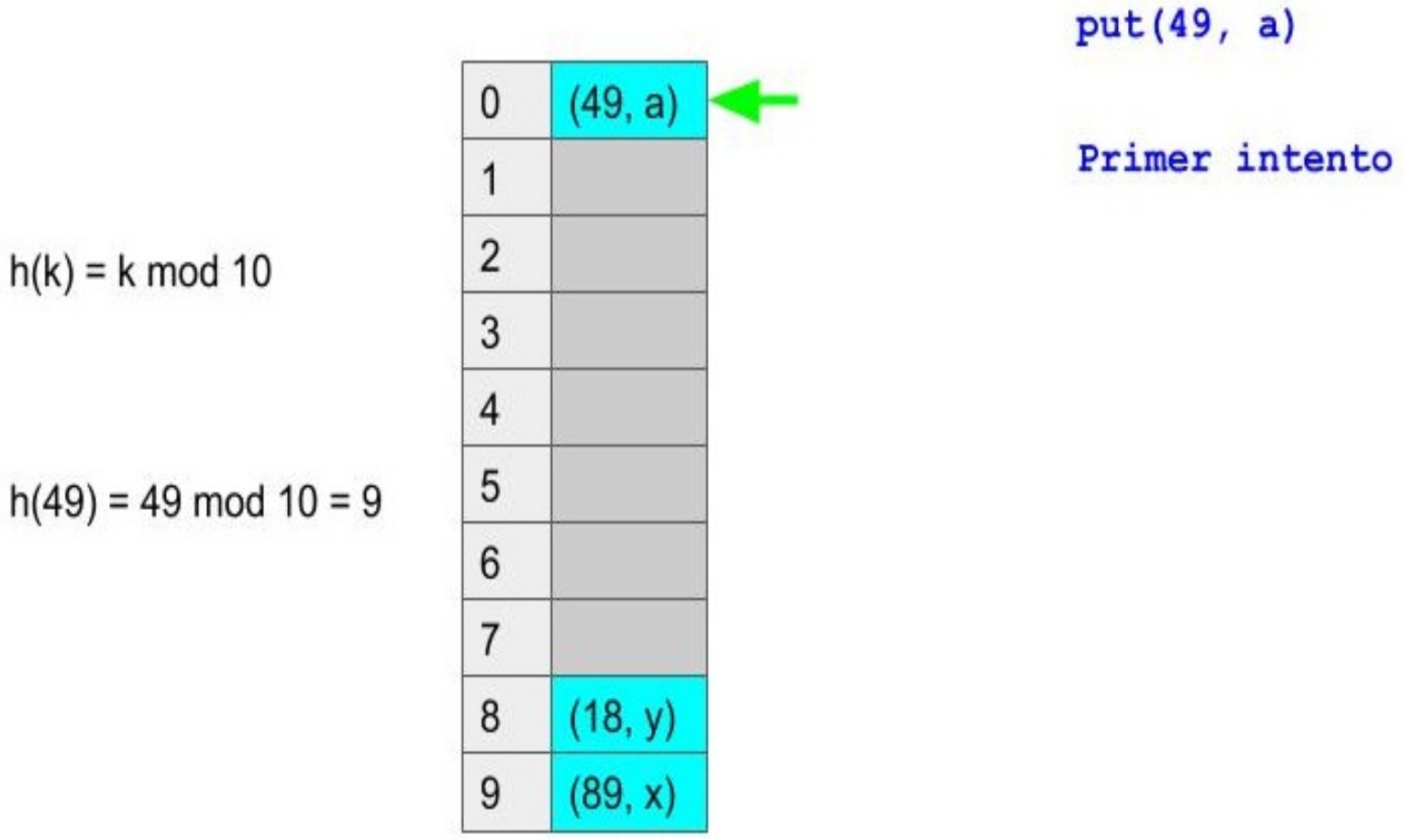
Tanteo lineal (linear probing)

Ejemplo con una tabla de tamaño 10, una función de dispersión que consiste únicamente en una función de compresión de una clave entera.



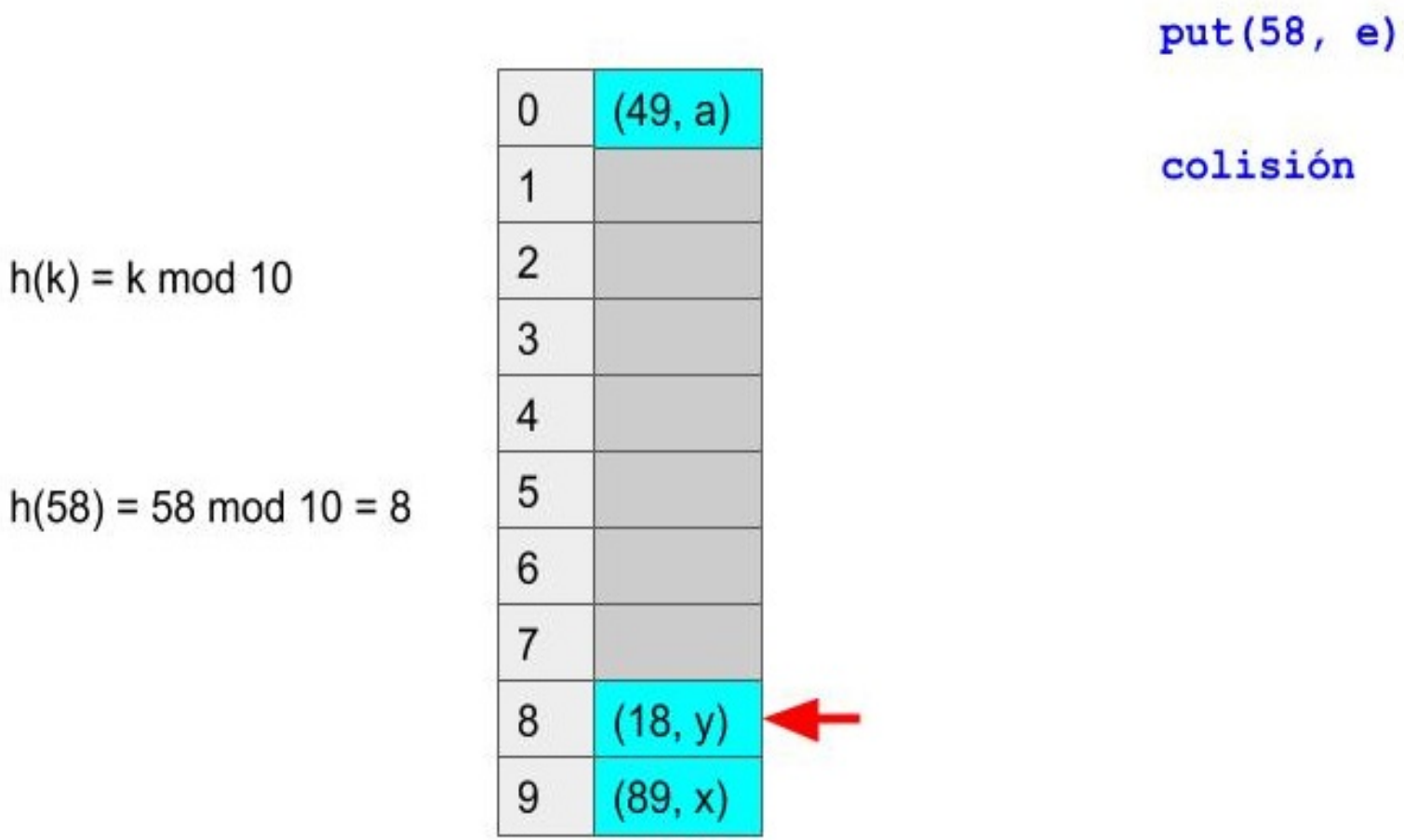
Tanteo lineal (linear probing)

Ejemplo con una tabla de tamaño 10, una función de dispersión que consiste únicamente en una función de compresión de una clave entera.



Tanteo lineal (linear probing)

Ejemplo con una tabla de tamaño 10, una función de dispersión que consiste únicamente en una función de compresión de una clave entera.



Tanteo lineal (linear probing)

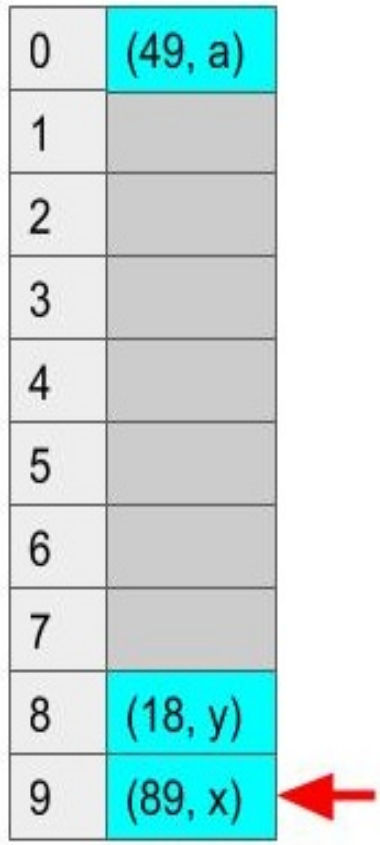
Ejemplo con una tabla de tamaño 10, una función de dispersión que consiste únicamente en una función de compresión de una clave entera.

put(58, e)

Primer intento

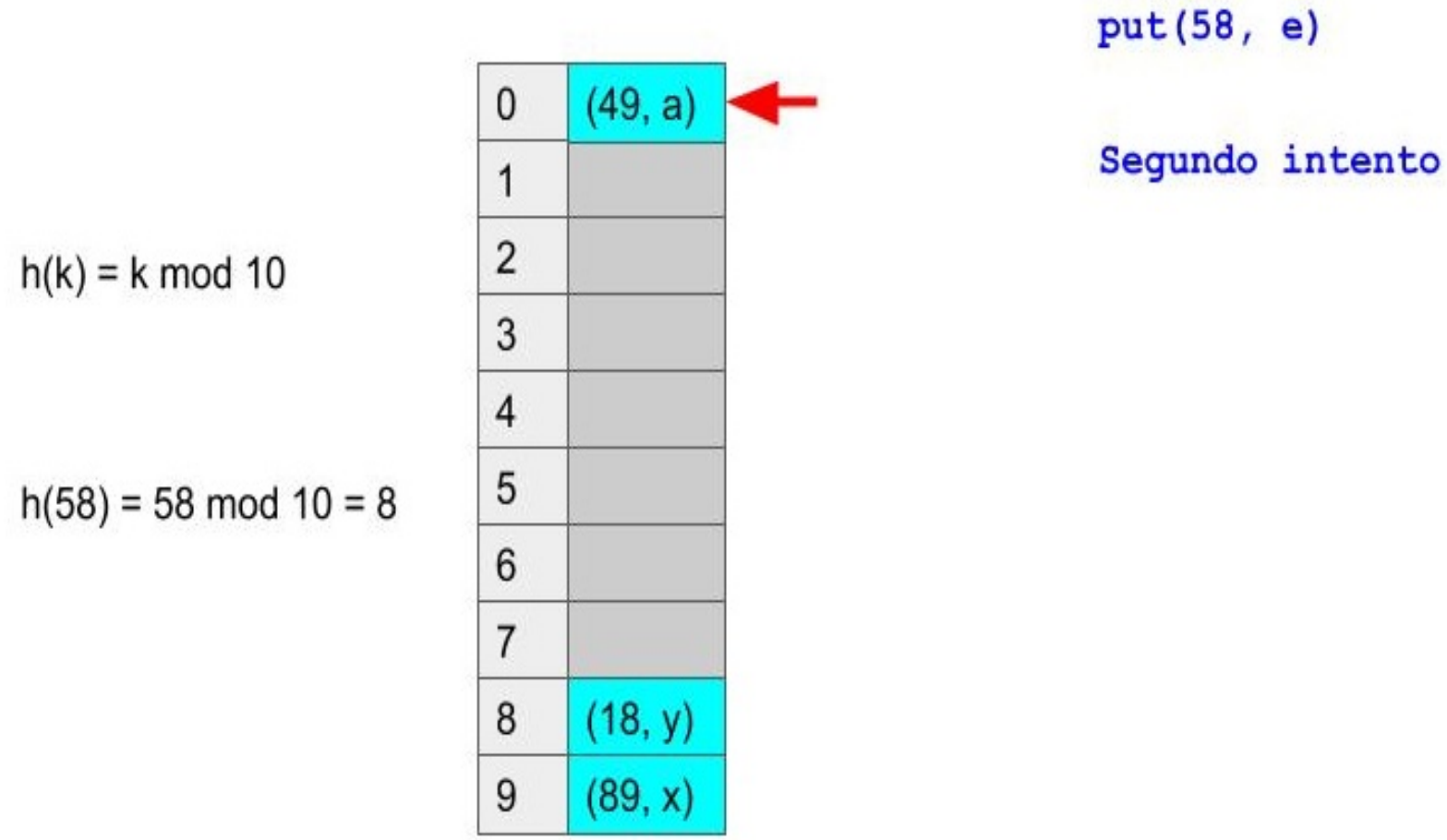
h(k) = k mod 10

h(58) = 58 mod 10 = 8



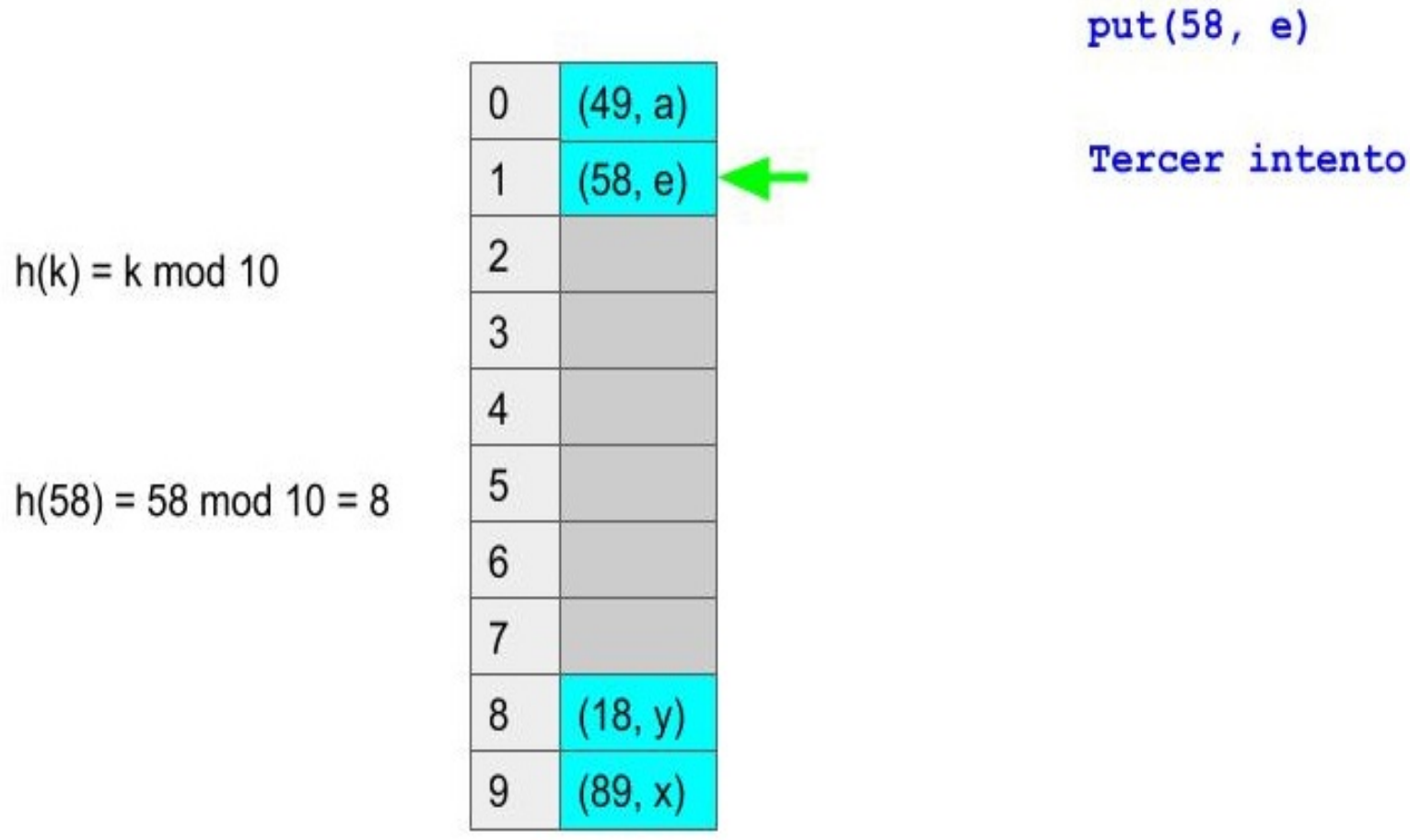
Tanteo lineal (linear probing)

Ejemplo con una tabla de tamaño 10, una función de dispersión que consiste únicamente en una función de compresión de una clave entera.



Tanteo lineal (linear probing)

Ejemplo con una tabla de tamaño 10, una función de dispersión que consiste únicamente en una función de compresión de una clave entera.



Tanteo lineal (linear probing)

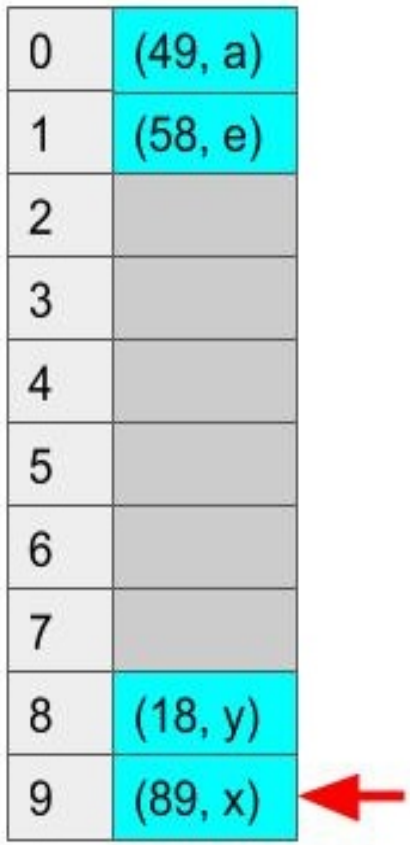
Ejemplo con una tabla de tamaño 10, una función de dispersión que consiste únicamente en una función de compresión de una clave entera.

put(69, r)

Colisión

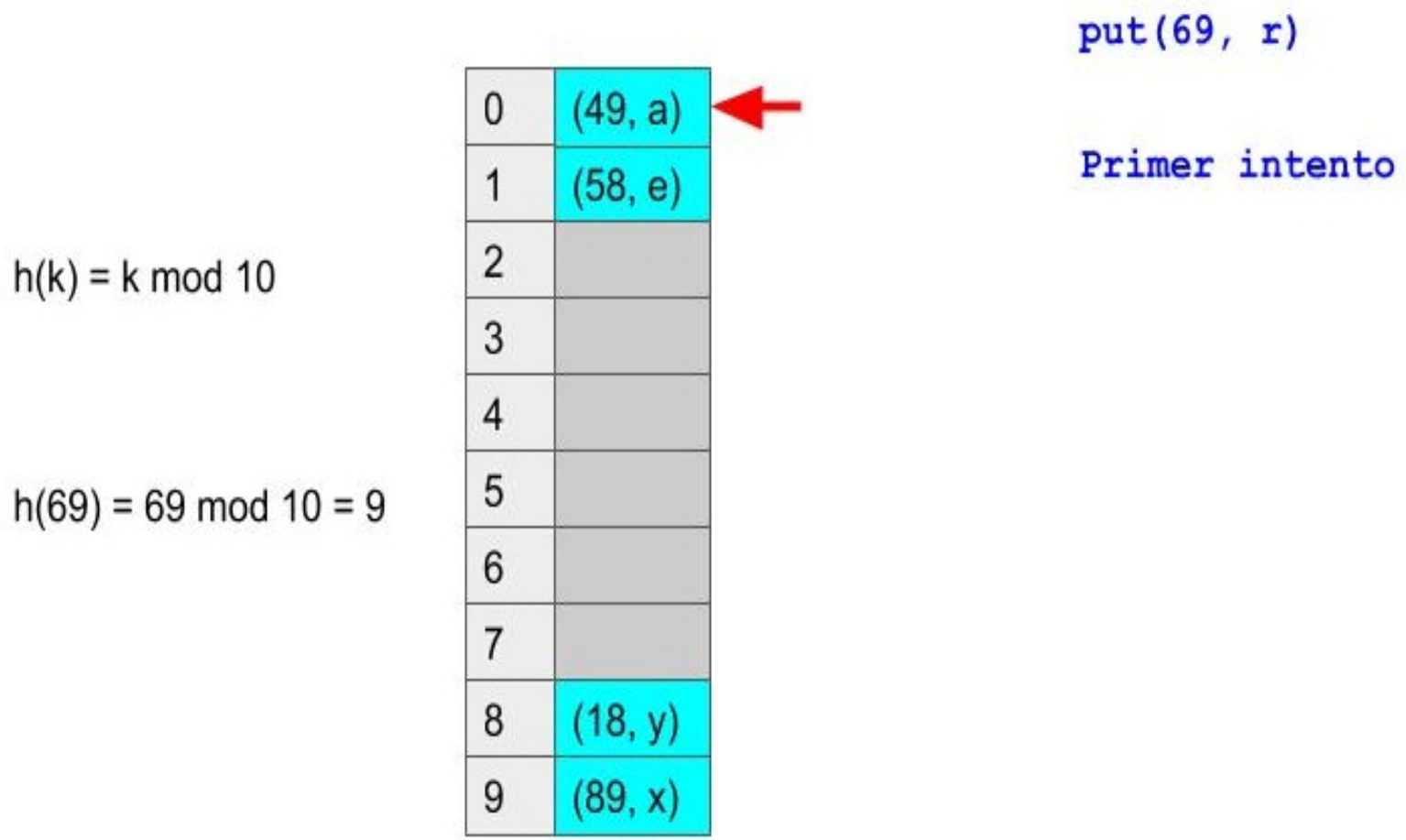
h(k) = k mod 10

h(69) = 69 mod 10 = 9



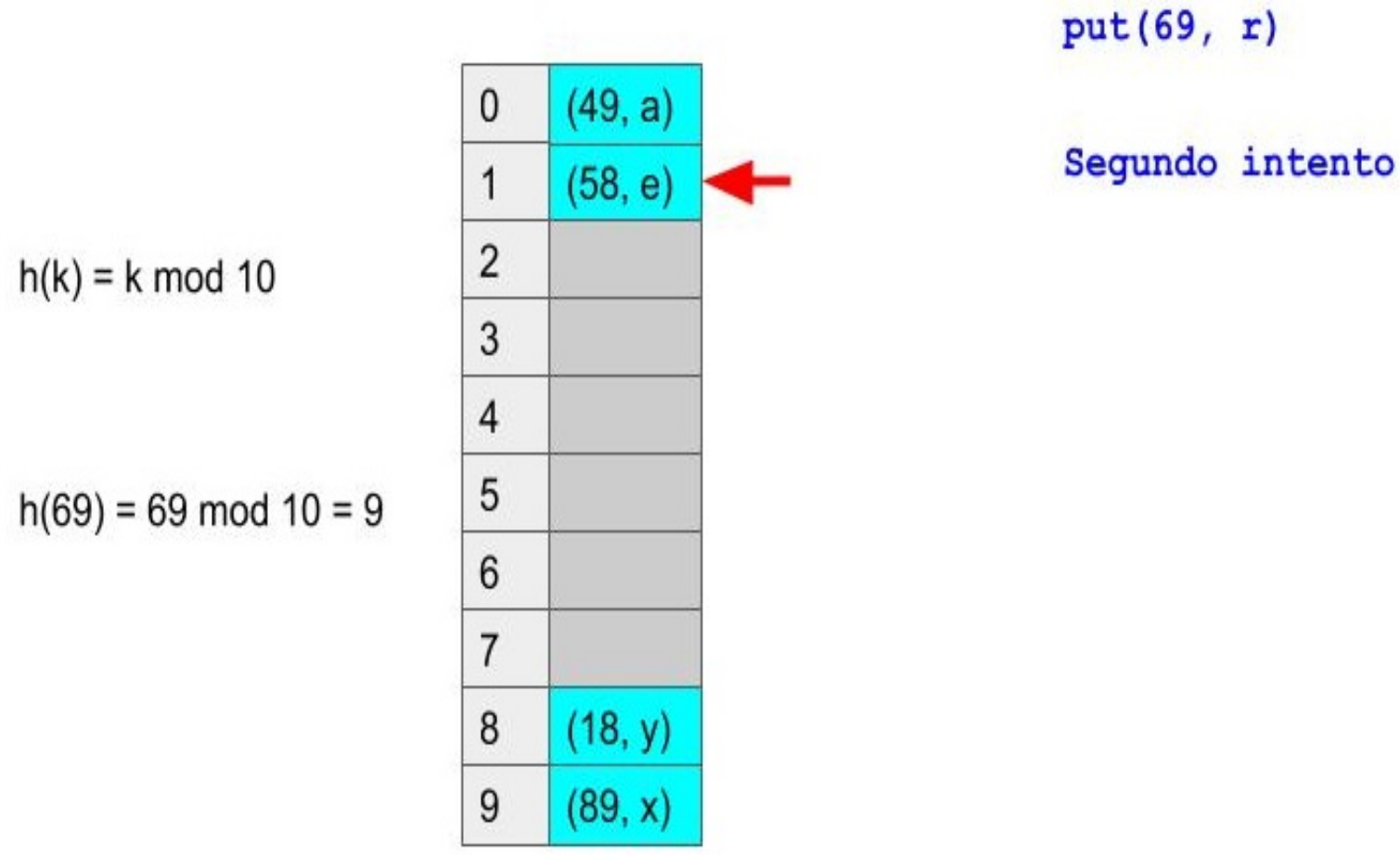
Tanteo lineal (linear probing)

Ejemplo con una tabla de tamaño 10, una función de dispersión que consiste únicamente en una función de compresión de una clave entera.



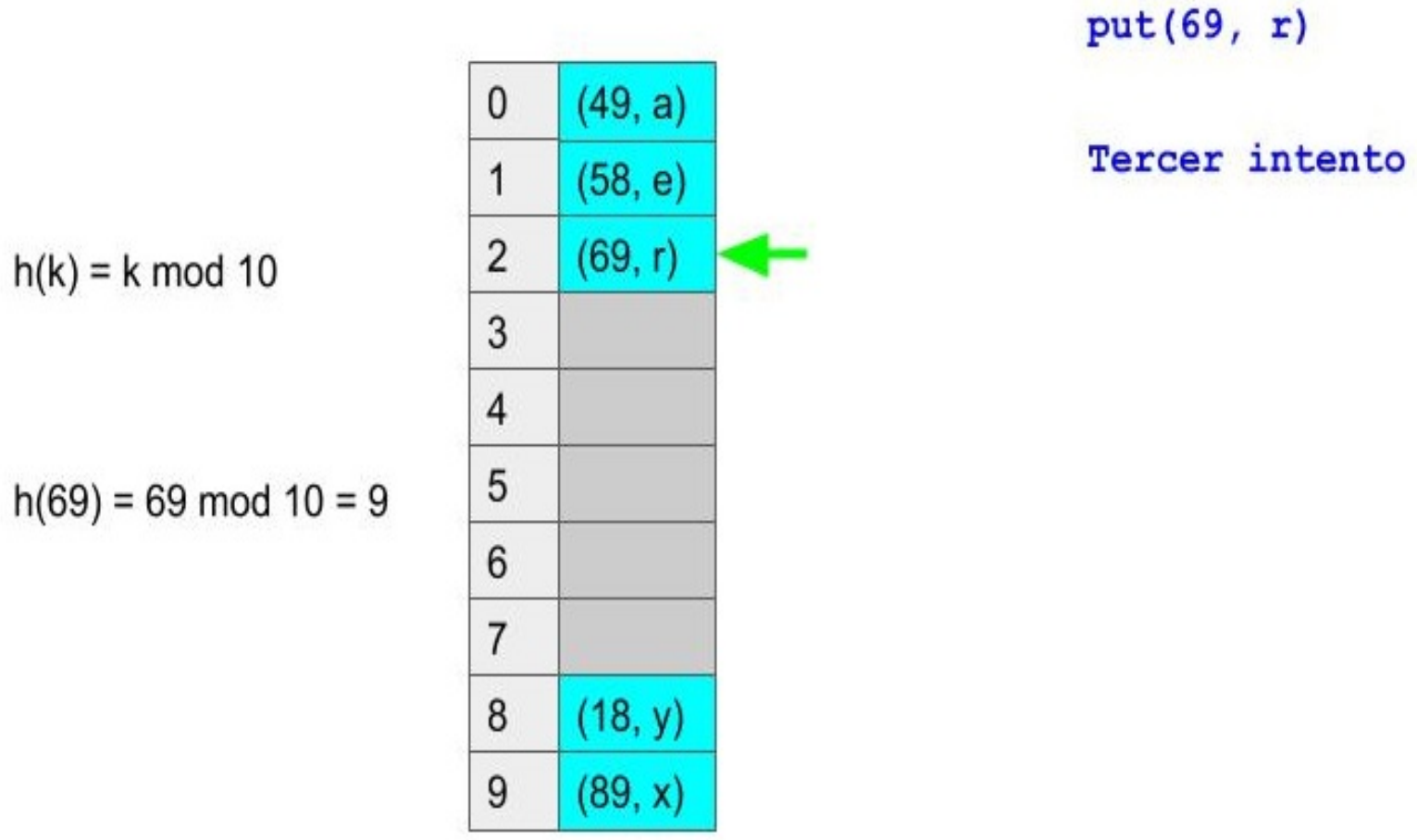
Tanteo lineal (linear probing)

Ejemplo con una tabla de tamaño 10, una función de dispersión que consiste únicamente en una función de compresión de una clave entera.



Tanteo lineal (linear probing)

Ejemplo con una tabla de tamaño 10, una función de dispersión que consiste únicamente en una función de compresión de una clave entera.

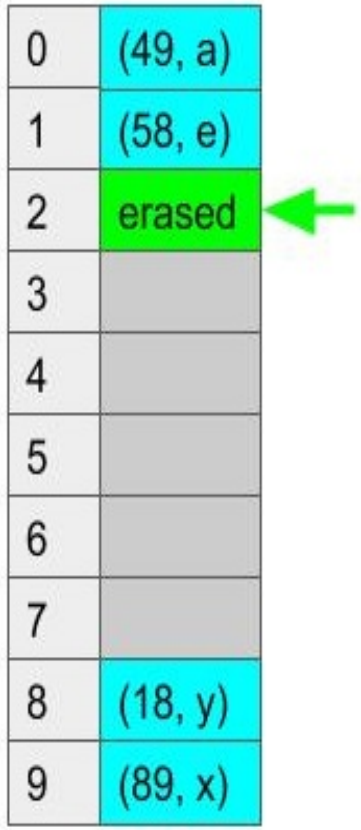


Tanteo lineal (linear probing)

Ejemplo con una tabla de tamaño 10, una función de dispersión que consiste únicamente en una función de compresión de una clave entera.

h(k) = k mod 10

h(69) = 69 mod 10 = 9



`remove(69)`

Sigue la misma secuencia: posiciones 9, 0, 1 y 2

Tanteo lineal (linear probing)

Ejemplo con una tabla de tamaño 10, una función de dispersión que consiste únicamente en una función de compresión de una clave entera.

```
put(23, m)
```

h(k) = k mod 10

h(23) = 23 mod 10 = 3

0	(49, a)
1	(58, e)
2	erased
3	(23, m)
4	
5	
6	
7	
8	(18, y)
9	(89, x)

Tanteo lineal (linear probing)

Ejemplo con una tabla de tamaño 10, una función de dispersión que consiste únicamente en una función de compresión de una clave entera.

h(k) = k mod 10

h(69) = 69 mod 10 = 9

0	(49, a)
1	(58, e)
2	erased
3	(23, m)
4	
5	
6	
7	
8	(18, y)
9	(89, x)

```
remove(69)

Sigue la secuencia:
posiciones 9, 0, 1,
2, 3 y se detiene en
4 porque el cajón es
nulo, fracasando
```

Tanteo cuadrático (quadratic probing)

- Estrategia de direccionamiento abierto que intenta situar a la entrada que colisiona aumentando el número de intento actual al cuadrado (de forma circular): si $h(k) = i$ y la localidad $A[i]$ está ocupada, el tanteo i-ésimo ocurre en la posición $s_i(k) = (h(k) + i^2)\ mod\ N$, donde $i <= N$.
- Evita el agrupamiento primario pero crea su propio tipo de agrupamiento, denominado **secundario**: el conjunto de cajones ocupados forma un patrón no uniforme, incluso suponiendo que los códigos de dispersión se distribuyen de manera uniforme.
- Cuando N es número primo y la tabla está llena a menos de la mitad de N, esta estrategia garantiza encontrar un cajón vacío, sin embargo, a partir de que la tabla se llena a la mitad, o bien, si N no es primo, podría no encontrarse cajón para insertar una entrada con la secuencia de tanteos.

Tanteo cuadrático (quadratic probing)

Ejemplo con una tabla de tamaño 10, una función de dispersión que consiste únicamente en una función de compresión de una clave entera.

Entradas = [(**89**,x), (**18**,y) ,(**49**, a), (**58**, e), (**69**, r)]

```
put(89, x)
```

h(k) = k mod 10

h(89) = 89 mod 10 = 9

0	
1	
2	
3	
4	
5	
6	
7	
8	
9	(89, x)

Tanteo cuadrático (quadratic probing)

Ejemplo con una tabla de tamaño 10, una función de dispersión que consiste únicamente en una función de compresión de una clave entera.

Entradas = [(**89**,x), (**18**,y) ,(**49**, a), (**58**, e), (**69**, r)]

```
put(18, y)
```

h(k) = k mod 10

h(18) = 18 mod 10 = 8

0	
1	
2	
3	
4	
5	
6	
7	
8	(18, y)
9	(89, x)

Tanteo cuadrático (quadratic probing)

Ejemplo con una tabla de tamaño 10, una función de dispersión que consiste únicamente en una función de compresión de una clave entera.

Entradas = [(**89**,x), (**18**,y) ,(**49**, a), (**58**, e), (**69**, r)]

h(k) = k mod 10

h(49) = 49 mod 10 = 9

0	(49, a)
1	
2	
3	
4	
5	
6	
7	
8	(18, y)
9	(89, x)

```
put(49, a)

Las posiciones de la
secuencia de tanteos
son:

(9 + 1) mod 10 = 0
```

Tanteo cuadrático (quadratic probing)

Ejemplo con una tabla de tamaño 10, una función de dispersión que consiste únicamente en una función de compresión de una clave entera.

Entradas = [(**89**,x), (**18**,y) ,(**49**, a), (**58**, e), (**69**, r)]

h(k) = k mod 10

h(58) = 58 mod 10 = 8

0	(49, a)
1	
2	(58, e)
3	
4	
5	
6	
7	
8	(18, y)
9	(89, x)

```
put(58, e)

Las posiciones de la
secuencia de tanteos
son:

(8 + 1) mod 10 = 9

(8 + 4) mod 10 = 2
```

Tanteo cuadrático (quadratic probing)

Ejemplo con una tabla de tamaño 10, una función de dispersión que consiste únicamente en una función de compresión de una clave entera.

Entradas = [(**89**,x), (**18**,y) ,(**49**, a), (**58**, e), (**69**, r)]

$h(k) = k \bmod 10$

$h(69) = 69 \bmod 10 = 9$

0	(49, a)
1	
2	(58, e)
3	(69, r)
4	
5	
6	
7	
8	(18, y)
9	(89, x)

`put(69, r)`

Las posiciones de la secuencia de tanteos son:

$(9 + 1) \bmod 10 = 0$

$(9 + 4) \bmod 10 = 3$

Dispersión doble (double hashing)

- Estrategia que no ocasiona ninguno de los agrupamientos mencionados, la cual define una segunda función de dispersión que se aplica en cuanto se presenta una colisión: si $h(k) = i$ y la localidad $A[i]$ está ocupada, se define una función $g(k)$ y el tanteo i-ésimo se lleva a cabo en la posición $s_i(k) = (h(k) + i \cdot g(k))\ mod\ N$, donde $i <= N$.

- La segunda función no puede evaluar a cero, una elección común es $g(k) = q - (k \mod q)$ para algún número primo $q < N$. N debe ser primo también.

Dispersión doble (double hashing)

Ejemplo con una tabla de tamaño 11, una función de dispersión que consiste únicamente en una función de compresión de una clave entera.

Entradas = [(**89**,x), (**18**,y) ,(**49**, a), (**58**, e), (**69**, r)]

h(k) = k mod 11

g(k) = 7 - (k mod 7)

h(89) = 89 mod 11 = 1

h(18) = 18 mod 11 = 7

h(49) = 49 mod 11 = 5

h(58) = 58 mod 11 = 3

0	
1	(89, x)
2	
3	(58, e)
4	
5	(49, a)
6	
7	(18, y)
8	
9	
10	

```
put(89, x)
put(18, y)
put(49, a)
put(58, e)
```

Dispersión doble (double hashing)

Ejemplo con una tabla de tamaño 11, una función de dispersión que consiste únicamente en una función de compresión de una clave entera.

Entradas = [(**89**,x), (**18**,y) ,(**49**, a), (**58**, e), (**69**, r)]

h(k) = k mod 11

g(k) = 5 - (k mod 5)

h(69) = 69 mod 11 = 3

g(69) = 5 - (69 mod 5) = 1

0	
1	(89, x)
2	
3	(58, e)
4	(69, r)
5	(49, a)
6	
7	(18, y)
8	
9	
10	

`put(69, r)`

Las posiciones de la secuencia de tanteos son:

`(3 + 1(1)) mod 11 = 4`

Reacomodo (rehashing)

- Si una inserción ocasiona que el factor de carga de la tabla de dispersión se pase del umbral marcado por la mitad de la capacidad, es común aumentar el tamaño de la tabla y reinsertar todas las entradas en la tabla nueva.
- No se define un nuevo código de dispersión para los objetos
- Se aplica una nueva función de compresión que tome en cuenta el nuevo tamaño de la tabla.
- Un requerimiento que funciona adecuadamente en la práctica es tomar como tamaño de la nueva tabla, un número primo que sea de aproximadamente del doble del tamaño de la tabla original.