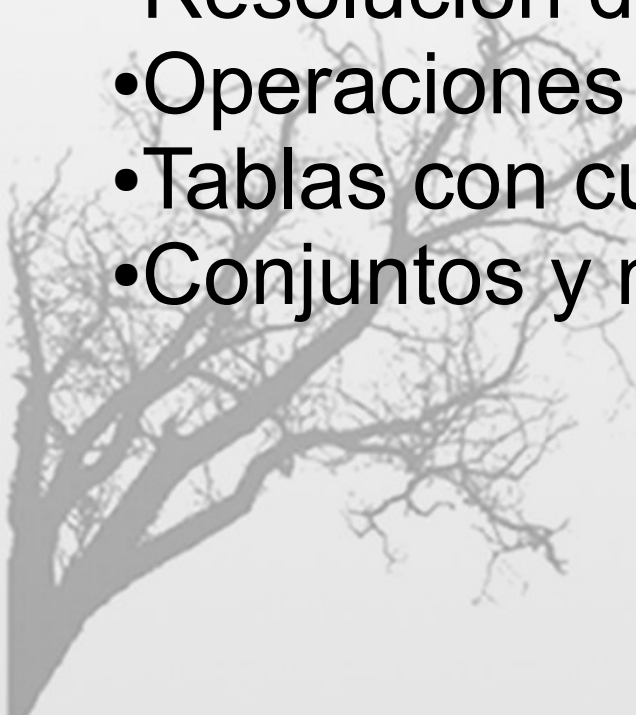


# Lección 15:

## Dispersión cerrada. Resolución de colisiones



- Motivación
- Dispersión cerrada
- Resolución de colisiones
- Operaciones en dispersión cerrada
- Tablas con cubetas
- Conjuntos y mapas dispersos



# Motivación



Un fichero disperso es una tabla de dispersión guardada en un fichero en disco. Idealmente este tipo de ficheros permite localizar un dato a partir de su clave con un único acceso.

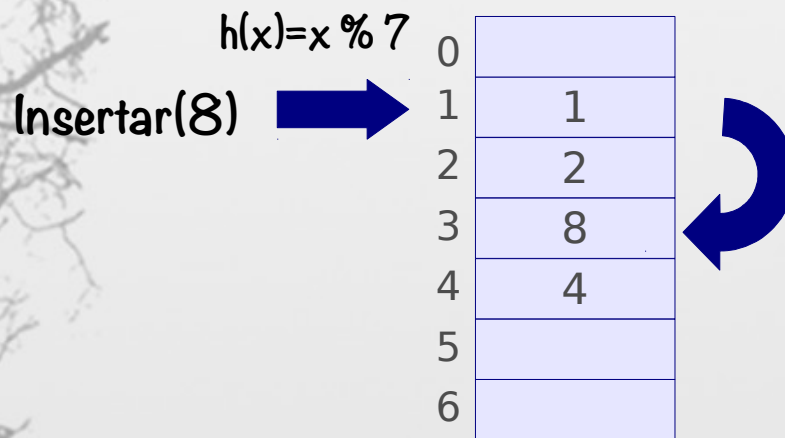
El encadenamiento separado es una técnica compleja y poco eficiente en disco, al tener que mantener múltiples listas enlazadas.

En este tipo de aplicaciones los datos deben ser guardados únicamente en la tabla de dispersión principal.

# Dispersión cerrada



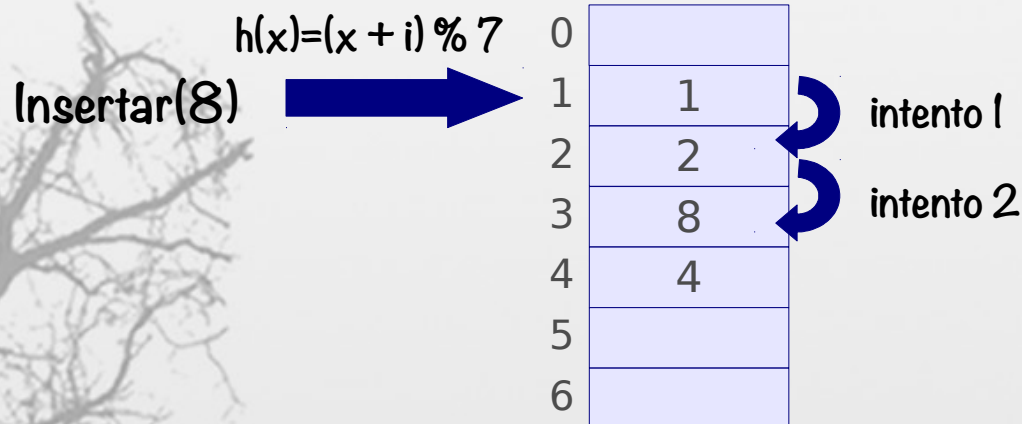
- Consiste en asignar otra posición libre de la tabla a una clave cuya posición natural está ya ocupada
- Ventajas
  - Se evitan estructuras secundarias
  - Se aprovecha mejor el espacio de la tabla



# Exploración lineal



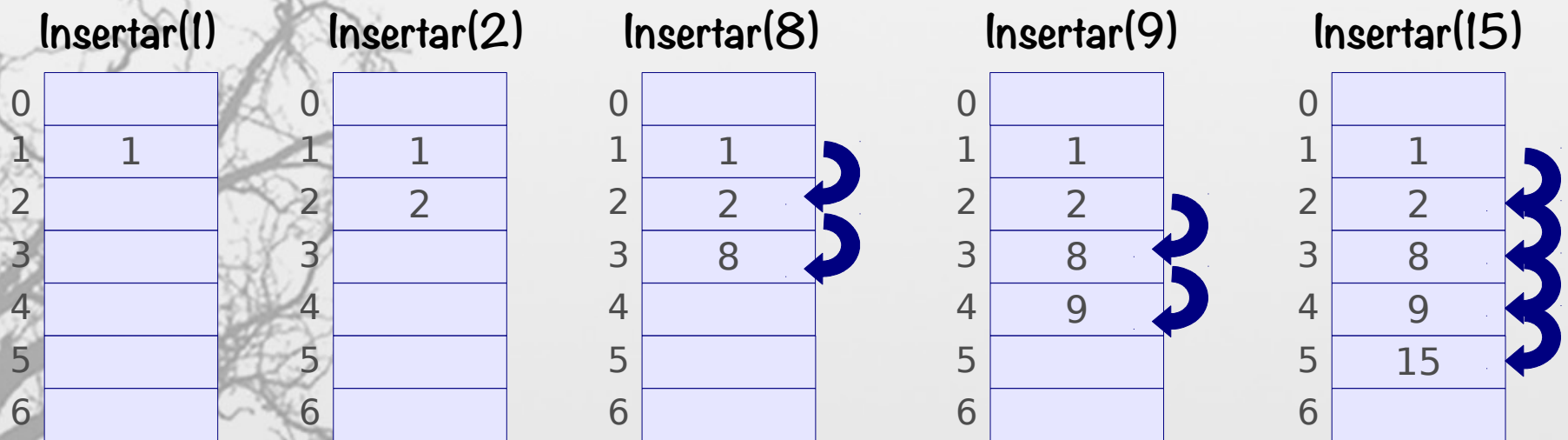
- Busca un hueco libre en posiciones consecutivas:
  - $h(x) = (x + i) \% t$ , siendo  $i$  el número de intento (0 inicialmente)



# Agrupamientos primarios



- La exploración lineal tiende a generar agrupamientos que atrapan nuevos datos y tienden a crecer
- Son los agrupamientos **primarios** → datos que van a distintas posiciones compiten por los mismos huecos durante la resolución de las colisiones



# Agrupamientos secundarios

- La exploración lineal genera para cada dato cuyo valor de  $h(x)$  coincide la misma secuencia de búsqueda de un hueco, lo que produce un segundo tipo de agrupamiento
- Son los agrupamientos **secundarios**

Insertar(1)

0	
1	1
2	
3	
4	
5	
6	

Insertar(8)

0	
1	1
2	8
3	
4	
5	
6	

Insertar(15)

0	
1	1
2	8
3	15
4	
5	
6	

Insertar(22)

0	
1	1
2	8
3	15
4	22
5	
6	

# Agrupamientos y rendimiento

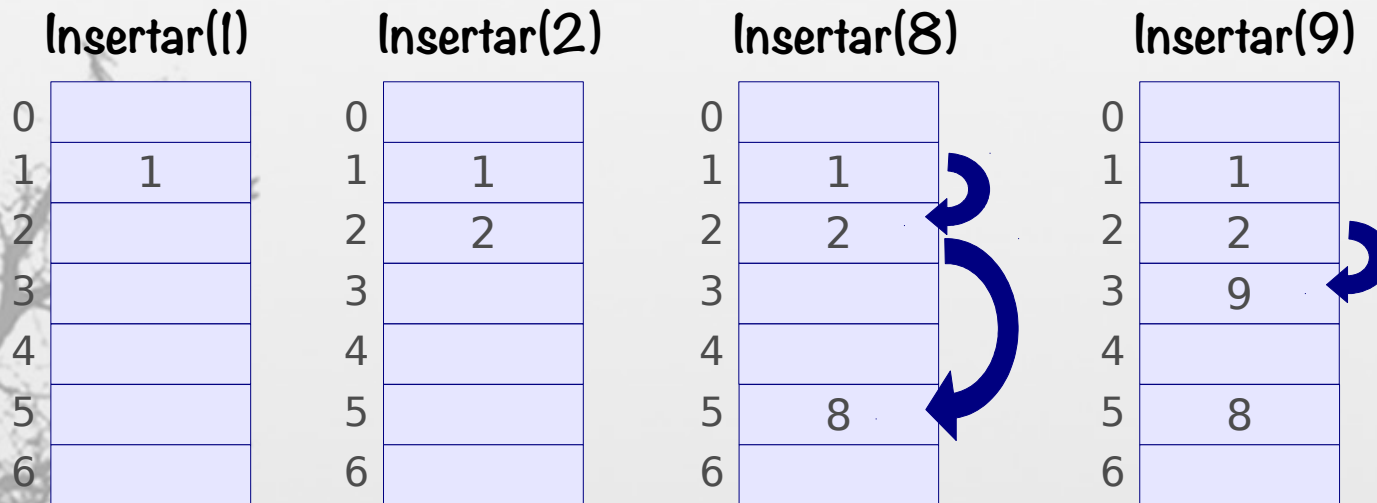


- Los dos tipos de agrupamiento tienden a alejar los datos de su posición natural
- Al crecer los agrupamientos, el rendimiento de la tabla baja notablemente
- Por ello la exploración lineal no es una buena solución

# Exploración cuadrática



- Utiliza la función:
  - $h(x) = (x + i^2) \% t$
- Evita los agrupamientos primarios, pero no los secundarios





# Dispersión doble



- La dispersión doble utiliza una segunda función de dispersión para resolver las colisiones
  - $h(x) = (h_1(x) + i \cdot h_2(x)) \% t$
- $h_2(x)$  no debe valer nunca 0. Ejemplos:
  - $h_2(x) = 1 + (x \% q)$  con  $q$  primo  $< t$
  - $h_2(x) = q - (x \% q)$  con  $q$  primo  $< t$
- Con  $h_2(x)$  bien escogida, se evitan los agrupamientos primarios y secundarios

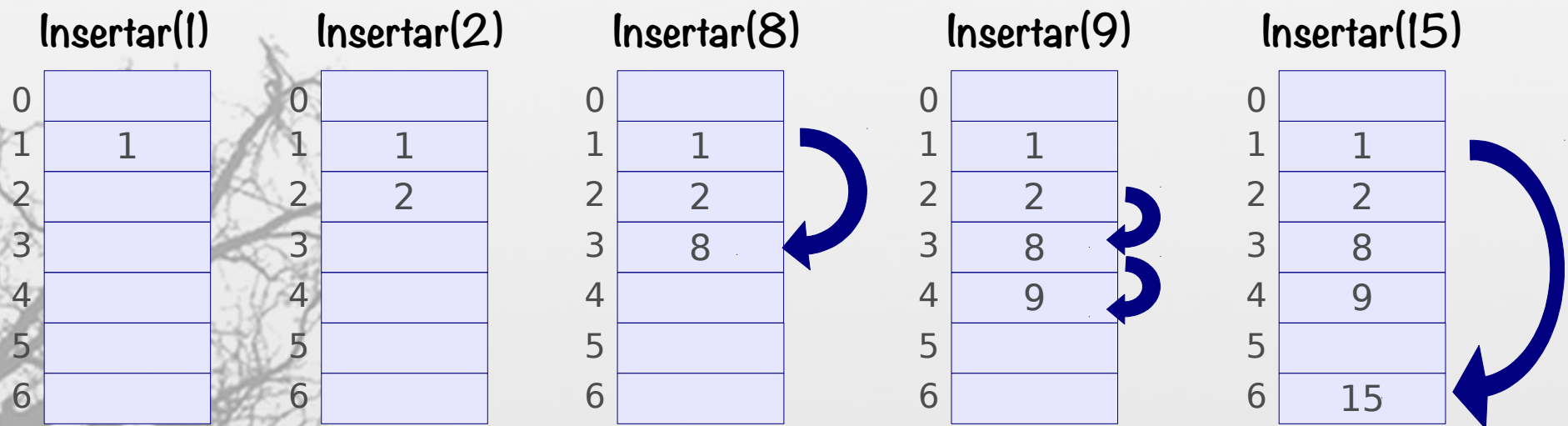
# Dispersión doble



$$h_1(x) = x \% 7$$

$$h_2(x) = x \% 5$$

$$h(x) = ((x \% 7) + i * (5 - x \% 5)) \% 7$$



# Operaciones en disp. cerrada

- El hecho de que un dato pueda no estar en su posición natural complica las búsquedas y borrados
- Al buscar hay que continuar hasta que se encuentra el dato o se llega a un hueco

$$h(x) = (x + i) \% 7$$

Buscar(8)



0	
1	1
2	2
3	8
4	9
5	
6	

Buscar(15)



0	
1	1
2	2
3	8
4	9
5	⊘
6	

# Problemas de los borrados

- Sin embargo, un borrado puede confundir la búsqueda!

$$h(x) = (x + i) \% 7$$

0	
1	1
2	2
3	8
4	9
5	
6	


Borrar(2)



0	
1	1
2	X
3	8
4	9
5	
6	

Buscar(8)



0	
1	1
2	
3	8
4	9
5	
6	



# Posiciones vacías y disponibles

- Para resolver el problema, las posiciones borradas se marcarán como **disponibles**
- Existirán casillas vacías (nunca ocupadas), disponibles (borradas) y ocupadas
- Las búsquedas sólo pararán al encontrar una casilla vacía

$$h(x) = (x + i) \% 7$$

0	
1	1
2	2
3	8
4	9
5	
6	

Borrar(2)



0	
1	1
2	X *
3	8
4	9
5	
6	

Buscar(8)

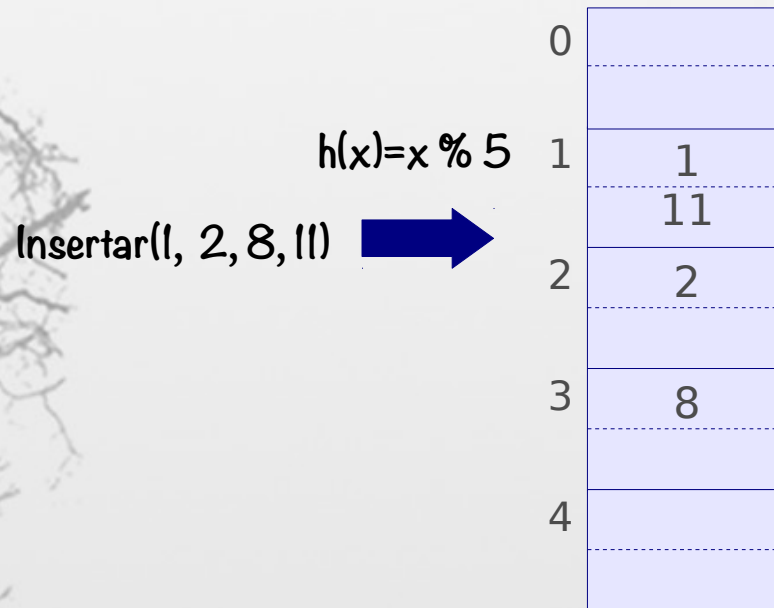


0	
1	1
2	*
3	8
4	9
5	
6	

# Tablas con cubetas



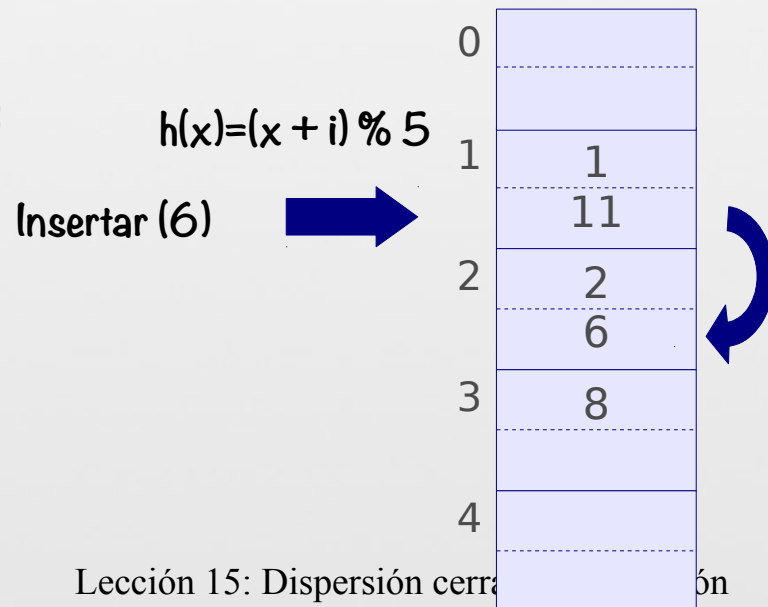
- Una tabla con **cubetas** o **compartimentos** es una tabla donde cada posición puede contener más de un dato
- Disminuye el riesgo de reasignación de datos a otras posiciones



# Colisiones en cubetas



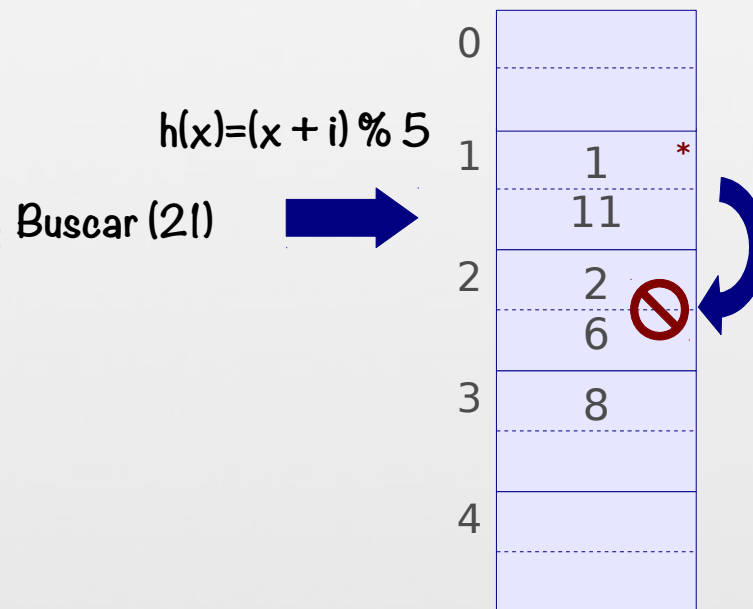
- Con una buena función de dispersión y un tamaño de tabla y cubeta adecuados, los datos deberían ser asignados a la posición que les corresponde
- Pero al final siempre es necesario disponer de una técnica de resolución de colisiones, como la exploración lineal



# Búsqueda con cubetas



- Usaremos una marca para las **cubetas desbordadas** durante la inserción
- Continuaremos la búsqueda hasta encontrar el dato o una cubeta que nunca se ha desbordado





# Rendimiento de una tabla disp.

- Una tabla de dispersión cerrada se deteriora notablemente con las inserciones y borrados
- Aparecen casillas disponibles (o cubetas desbordadas) que alargan las secuencias de búsqueda
- La dispersión cerrada funciona bien si fundamentalmente se utiliza para búsquedas
- Es mucho más sensible que la dispersión abierta a factores de carga altos

# Conjuntos y mapas dispersos

- Existe una extensión a STL (no estándar) que implementa los sets y maps (y las versiones multi) mediante dispersión abierta

GNU C++

```
#include <ext/hash_set>
using namespace __gnu_cxx;
hash_set<int> miSet;
```

```
#include <ext/hash_map>
using namespace __gnu_cxx;
hash_map<string, int> miMapa;
```

Visual C++

```
#include <hash_set>
using namespace stdext;
hash_set<int> miSet;
```

```
#include <hash_map>
using namespace stdext;
hash_map<string, int> miMapa;
```

# Clases de dispersión



- Adicionalmente puede indicarse como parámetro una **clase de dispersión** para el tipo indicado y una clase de comparación de igualdad

```
class Punto {
    int x, int y;
public:
    Punto (int x1, int y1): x(x1), y(y1) {}
    int leeX() const { return x; }
    int leeY() const { return y; }
};

class DispersionPunto {
    size_t operator()(const Punto &p) {
        return p.leeX() * p.leeY();
    }
};

class IgualPunto {
    bool operator() (const Punto &a, const Punto &b) {
        return a.leeX() == b.leeX() && a.leeY() == b.leeY();
    }
}

hash_set<Punto, DispersionPunto, IgualPunto> puntos;
```

# Consideraciones adicionales



- Puede iterarse sobre un conjunto o mapa disperso, aunque no se obtendrá un listado ordenado de los datos
- La implementación es dinámica, ampliando el tamaño de la tabla y redispersando cuando el factor de carga se acerca a 1
- Es conveniente usar la operación `resize()` para indicar desde el principio el tamaño aproximado de la tabla

# Conclusiones



- La dispersión cerrada es más sencilla y utiliza mejor el espacio de la tabla que la dispersión abierta, pero también es susceptible de sufrir mayores problemas de caída de rendimiento
- El uso de cubetas disminuye el riesgo de colisiones y está especialmente indicado para almacenamiento en memoria secundaria (ficheros)
- Existe una extensión a STL que implementa los conjuntos y mapas mediante tablas de dispersión

