Estructuras de Datos

Práctica 6. Dispersión

Sesiones de prácticas: 2

Objetivos

Aplicar la técnica de dispersión a un problema donde se requiere el acceso eficiente a un conjunto grande de datos.

Ejercicio propuesto

Volveremos nuevamente al ejercicio de los códigos postales de la práctica 4 para hacer una reimplementación usando dispersión en lugar del árbol AVL. Se diseñará una tabla de dispersión cerrada de tamaño adecuado para albergar la información de códigos postales con un número de colisiones inferior a 5 (hacer pruebas con distintos tamaños). Como técnica de resolución de colisiones se usará dispersión doble usando dos funciones de dispersión por división.

En nuestro caso no será necesario implementar soporte de borrados especiales, puesto que la información de códigos postales no se actualiza una vez cargada la estructura de datos. No obstante, cada posición de la tabla guardará además del dato un valor booleano que indica si la posición está disponible o no. La interfaz de la plantilla a implementar debe ser similar a la siguiente:

```
template<typename T>
class Dispersion<T> {
  // Estructura de datos
  ...
  public:
    /** Crear la tabla con un tamaño dado*/
    Dispersion<T>(unsigned long tam);
    /** Insertar el dato indicado. Devuelve false si el dato existe o el dato no ha podido ser insertado
    bool insertar(const long clave, const T &dato);
    /** Buscar el dato indicado. Devuelve un puntero nulo si no existe */
    T *buscar(const long clave);
};
```

La clave de tipo long que aparece en las operaciones de inserción y búsqueda se obtendrá aplicando el algoritmo *djb2* que se ha estudiado en la teoría al campo de ciudad de la estructura *StructPost*.

Ejercicio voluntario

Los valores booleanos se codifican en C++ mediante enteros con valor 0/1, así que incluir este valor por cada posición aumenta bastante el tamaño de la tabla (4 bytes por posición). Se propone eliminar este booleano y añadir un conjunto de bits en Dispersion<T> para codificar la disponibilidad de las posiciones. De esta forma reducimos el consumo de memoria a un único bit por posición.