

# Implementación de diccionarios sobre trie en C++

Algoritmos y Estructuras de Datos II

1.<sup>er</sup> cuatrimestre de 2018

# Introducción

- ▶ Vamos a implementar una interfaz de diccionario en C++.
- ▶ La representación interna consistirá en un árbol con invariante de **trie**.
- ▶ Utilizaremos memoria dinámica.

# Tries

- ▶ Árbol  $k$ -ario para alfabetos de  $k$  elementos.
- ▶ Las aristas representan las unidades sobre las que se expresan las claves (ej.: letras para strings, dígitos para números).
- ▶ Cada subárbol representa al subconjunto de las claves que tienen como prefijo las etiquetas de las aristas que llevan hasta él.
- ▶ Los nodos internos pueden o no tener significados.
- ▶ Las hojas siempre contienen un significado.

# Implementación en C++

- ▶ Vamos a implementar una clase `string_map<T>` paramétrica en un tipo `T` del que supondremos que tiene un constructor por copia y operador de asignación.
- ▶ Las claves del diccionario serán strings.
- ▶ Primero plantearemos el esquema de la clase.
- ▶ Luego la parte pública (interfaz).
- ▶ Luego la parte privada (representación y funciones auxiliares).
- ▶ Por último, la implementación de los métodos.

## Esquema de la clase string\_map<T>

```
#ifndef STRING_MAP_H_
#define STRING_MAP_H_

template <class T>
class string_map {
    public:
        /*...*/
    private:
        /*...*/
};

/* ... */
#endif //STRING_MAP_H_
```

# Interfaz

- ▶ Queremos dotar a nuestra clase de una interfaz de Diccionario.
- ▶ Vamos a simular ser programadores de la Standard Template Library y programar un subconjunto de las funciones existentes<sup>1</sup>. En particular, para el taller, nos conformamos con:
  - ▶ Crear un diccionario nuevo (vacío).
  - ▶ Crear un diccionario a partir de otro (por copia).
  - ▶ El operador de asignación.
  - ▶ Definir un significado para una clave.
  - ▶ Decidir si una clave está definida en el diccionario.
  - ▶ Obtener el significado de una clave.
  - ▶ Borrar un elemento.
  - ▶ Tamaño del diccionario.

---

<sup>1</sup><http://www.cplusplus.com/reference/map/map/>

# Interfaz

```
template <class T>
class string_map {
    public:
        string_map();
        string_map(const string_map<T>& d);
        string_map& operator=(const string_map<T>& d);
        T& operator[](const string &key);
        int count(const string &key) const;
        T& at(const string& key);
        void erase(const string& key);
        int size() const;
        bool empty() const;
    private :
        /*...*/
};
```

¿Qué hacía el operador []?

# Operador []

```
T& operator[] (const string &key);
```

- ▶ Si la clave *key* está definida en el diccionario, la función devuelve una referencia a su significado.
- ▶ Si la clave *key* no está definida, la función crea un nuevo elemento con esa clave, y lo devuelve

¿Qué sucede cuando hacemos `dicc[key] = value`?

- ▶ Se busca la clave *key* en el diccionario y, en caso de no encontrarla, se crea un valor por defecto.
- ▶ Se llama al operador de asignación del tipo de *value* para reemplazar el valor almacenado en el diccionario.



# Representación de los nodos

- ▶ Definimos una estructura `Nodo` para representar los nodos del Trie.
- ▶ La estructura estará en la parte privada de la clase `string_map`.
- ▶ La estructura va a contener un puntero a una definición `T*` y un arreglo de punteros a los nodos siguientes.
- ▶ La estructura tendrá un constructor sin parámetros.

# Representación de los nodos

```
private:
    struct Nodo{
        Nodo** siguientes;
        T* definicion;
        Nodo() {
            /*...*/
        }
    };
    /*...*/
```

- ▶ ¿Por qué siguientes es de tipo `Nodo**`? ¿Qué significa?
- ▶ ¿Por qué definicion es de tipo `T*` y no de tipo `T`?

# Representación de los nodos

```
private:
    struct Nodo{
        Nodo** siguientes;
        T* definicion;
        Nodo(){
            /*...*/
        }
    };
    Nodo* raiz;
    int size;
```

Tenemos dos variables de instancia:

- ▶ `raiz` apunta al nodo raíz del trie, o es `NULL` si el trie no tiene nodos.
- ▶ `size` contiene la cantidad de claves del `string_map`.

# Definido

- ▶ Empezamos en la raíz, si existe, si no devolver False.
- ▶ Recorremos el trie mirando cada caracter de la clave:
  - ▶ Si en siguientes del nodo actual ese caracter apunta a NULL, devolvemos False.
  - ▶ Si no, pasamos a ver el nodo al que apunta siguientes del caracter actual.
- ▶ Observación: la función `int(char)` de C++ devuelve el código ASCII de un caracter.
- ▶ ¿Que pasa con la clave que no tiene caracteres?: ""

## Definir un elemento

- ▶ Si el trie está vacío creamos un nuevo Nodo al que apunta la raíz.
- ▶ Buscamos en qué lugar del trie debe ir la nueva clave.
- ▶ Para ello vamos recorriendo el trie como en Definido.
- ▶ Cuando nos encontramos que un caracter en siguientes apunta a NULL creamos un nuevo Nodo al que apuntar.
- ▶ Al terminar de recorrer todos los caracteres de la clave llegamos al lugar donde debe ir el significado.

## Borrar un elemento

- ▶ Hay que borrar todos los nodos intermedios que ya no tengan razón de ser.
- ▶ Tres casos posibles:
  - ▶ Si la definición está en una hoja, borrar todo el camino hasta la hoja que no sea necesario para alguna otra definición.
  - ▶ Si está en un nodo intermedio, borrar solamente ese significado.
  - ▶ Si es la única definición del trie, borrar todo y dejar la raíz apuntando a NULL.

# ¡A programar!

En `string_map.h` está la declaración de la clase, su parte pública y la declaración de `Nodo`.

En `string_map.hpp` está el esqueleto para que ustedes completen la definición de los métodos de `string_map`.