Implementación de diccionarios sobre Trie en C++

Algoritmos y Estructuras de Datos II

2.^{do} cuatrimestre de 2016

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+1)-ario para alfabetos de k elementos.
- Los ejes 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 los ejes 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 `DiccString<T>` paramétrica en un tipo `T` del que sólo supondremos que tiene un constructor por copia.
- Las claves del diccionario serán strings.
- Primero plantearemos el esquema de la clase
- Luego la parte pública (interfaz)
- Luego la parte privada (representación y fcs. auxiliares)
- Por último, la implementación de los métodos

Esquema de la clase DiccString<T>

```
#ifndef DICC_STRING_H_
#define DICC_STRING_H_

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

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

Interfaz

- Queremos dotar a nuestra clase de una interfaz de Diccionario
- ¿Qué operaciones serán visibles para el usuario?

Interfaz

- Queremos dotar a nuestra clase de una interfaz de Diccionario
- ¿Qué operaciones serán visibles para el usuario?
  En particular, para el taller, nos conformamos con:
  - Crear un diccionario nuevo (vacío)
  - Crear un diccionario a partir de otro (por copia)
  - Definir un significado para una clave
  - Decidir si una clave está definida en el diccionario
  - Obtener el significado de una clave
  - Borrar un elemento
  - Devolver las claves del diccionario

Interfaz

```
template <class T>
class DiccString {
    public:
        DiccString();
        DiccString(const DiccString<T>&);
        void Definir(string clave, const T&
        bool Definido(string clave) const;
        T& Obtener(string clave) const;
        void Borrar(string clave);
        const Conj<string>& Claves() const;
    private :
        /*...*/
};
```

Representación de los nodos

- Definimos una estructura `Nodo` para representar los nodos del Trie.
- La estructura estará en la parte privada de la clase DiccString (no queremos exportarla).

Representación de los nodos

- Definimos una estructura `Nodo` para representar los nodos del Trie.
- La estructura estará en la parte privada de la clase DiccString (no queremos exportarla).
- 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;
    Conj<string> claves;
```

Tenemos dos variables de instancia:

- `raiz` apunta al nodo raíz del Trie, o es `NULL` si el Trie no tiene nodos
- `claves` contiene las claves del DiccString

Definido

Definido

- Empezamos en la raíz

Definido

- Empezamos en la raíz, si existe, si no devolver `False`

Definido

- Empezamos en la raíz, si existe, si no devolver `False`
- Recorremos el Trie mirando cada caracter de la clave:

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`

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

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.

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.

Definir un elemento

Definir un elemento

- Agregamos la clave al conjunto de claves
- 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`.
- Observación: al hacer `delete(puntero)` se libera la memoria apuntada por `puntero`, pero `puntero` sigue conteniendo el mismo valor (una posición de memoria). Si se quiere que `puntero` apunte a `NULL` hay que asignarle `NULL`.

¡A programar!

En `DiccString.hpp` está la declaración de la clase, su parte pública y la definición de `Nodo`. Completen (¡y prueben!) todos los métodos.