ICOM2020 – FINAL COMPLEMENTARIO

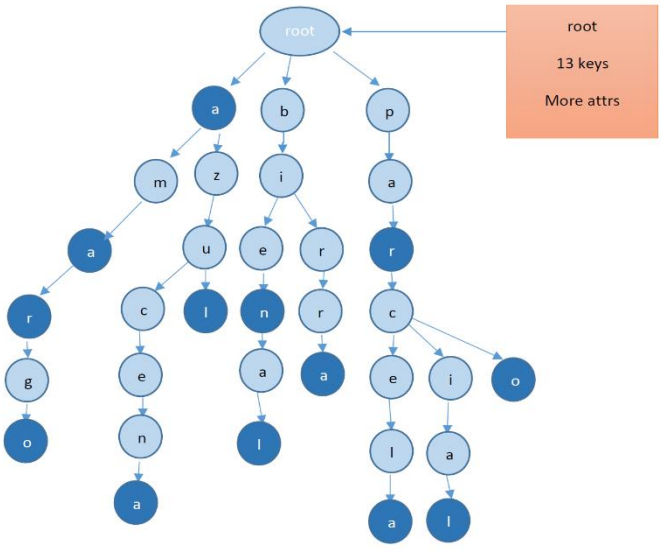
5 de febrero de 2021

Notas:

1. Uso de prácticos: **se pueden utilizar los trabajos prácticos propios realizados.**

Problema 1: Tries y Diccionarios binarios

a. Un **`Trie`** es una estructura de datos eficiente para la recuperación de información (information re**`trie`**val). Utilizando un **`trie`**, se puede buscar una clave en un tiempo proporcional a **`M`** (longitud de la clave). Cada nodo de un **`trie`** puede contener múltiples ramificaciones. Cada ramificación representa una nueva parte de la clave (un nuevo carácter en el caso de que las claves sean palabras) y en cada nodo se indica a través de un atributo si el nodo termina de definir una clave (y la cantidad de veces que la clave se agregó esta al Trie) o es solo un prefix para claves que terminan en nodos inferiores (un nodo que termina de definir una clave puede también ser prefix para otras).



En **`trie.cpp`** se encuentra una implementación parcial de este UDT. Complételo.

b. Un diccionario binario de palabras es también una estructura eficiente para recuperar información pero la almacena en un árbol binario con orden (por ejemplo todos los nodos a la izquierda de un nodo son menores al contenido de este, y los de la derecha son mayores). Junto con la palabra, cada nodo del árbol mantiene la cantidad de veces que esa palabra se insertó en el diccionario.

En **`tree.cpp`** se encuentra una implementación parcial de este UDT. Complételo.

Problema 2: **`Collectable`**'s y estadística

Suponga que define la clase abstracta **`Collectable`** que permite obtener un **`std::vector`** con todos los elementos contenidos en un contenedor:

```
struct StatWord {
    string   word;
    unsigned repeatCnt;
    bool operator<(const StatWord& e) {
        return repeatCnt < e.repeatCnt;
    }
};

class Collectable {
    public:
        virtual vector<StatWord> collect() = 0;
};
```

Modifique los UDTs del Problema 1 para que también sean "**`Collectable`**"s. (Defina nuevos tipos **`TrieCollectable`** y **`TreeCollectable`** que adhieran a la nueva interface).

En **`collectable.cpp`** se encuentra un esqueleto para implementar y probar los nuevos tipos.

Problema 3: Zombies

Se desea implementar un juego simple en el cual un jugador (H) comienza en una posición aleatoria de un mapa infestado de zombies (Z) que se mueven al azar. Para ganar debe llegar a la salida (S), también ubicada al azar en el mapa, sin toparse con ningún zombie. El código necesario se encuentra incompleto, con cada elemento faltante marcado con TODO.

Utilizando el esqueleto en **`zombies.cpp`** complete lo que haga falta para que el juego funcione correctamente.