Entrega Práctica 2 - TDA

Lukas Häring

Contenido

1 Explicación de mi TDA

Éste TDA que he realizado no sigue la estructura propuesta, pero tiene la misma funcionalidad, en primer lugar he querido utilizar la **STL** de C++, así como usando los contenedores que en mi opinión mejor les conviene y así como su eficiencia.

Como ya conocemos, cómo esté hecho el TDA no afecta a la parte visible del usuario pero voy a explicar cómo he querido implementarlo. Su estructura es sencilla, se trata de un *Tree Graph* (Véase la imagen 1), comentar que su profundidad (o *depth*) es de 3, el propio contenedor constituye el primer vértice (nodo o raíz, *depth 0*), la profundidad 2 constituye todos los nodos que forman las **centurias**, por ejemplo, si tenemos la fecha 1954, la centuria será 19 y de éste se forma un *sub-graph o sub-tree* que constituye la profundidad 2 que forman los **años** (de 0 - 99) que forman una centuria, como cada año tiene sus propios **eventos históricos**, éstos serán la última profundidad (*depth 3*).

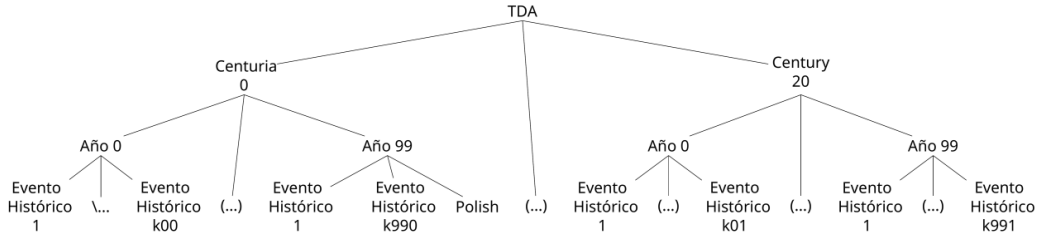

Imagen número 1
Éste grafo es para las centurias D.C y los puntos suspensivos indican que pueden haber más años o eventos históricos.

2 Eficiencia del TDA

Veamos su eficiencia en insertar elementos, voy a explicar cómo funciona, su lectura depende del número de líneas en un archivo, supongamos que el número de línes es "**n**", para cada línea (que contiene una fecha), ésta se dividirá en la *centuria* y en el *año*. La *centuria* se almacena en un contenedor de la stl, como no quiero elementos repetidos, he creado un **std::set** y un **functor** para hacer que el conjunto dependa de la centuria, ésto se hará lo mismo con los años (un functor y un set) además hay que ver si éste elemento está o no dentro del conjunto, según la **stl**, la búsqueda es **logarítmica**, pero ya veremos que no va a influir en su eficiencia, ahora cada línea se va a separar en k palabras éstas palabras y se van a almacenar en un contenedor a parte dentro de la centuria, (haciendo que sea más eficiente luego en la búsqueda) y finalmente cada *año* va a contener un vector de la stl para cada evento histórico, vemos que la cota inferior es cuando el archivo es vacío, por tanto la eficiencia será constante, por otro lado en el "peor" de los casos, éste será cuadrática según el número de elementos (No podemos encontrar realmente su eficiencia).

Veamos las búsquedas, en primer lugar, para buscar todos los eventos según una centuria, vemos que tenemos que recorrer (según el grafo), $0 - 99$ años y $1 - k$ eventos históricos, por tanto la eficiencia es constante ya que si no existe éste nodo, sale directamente y por otro lado, lineal según el número de eventos históricos; Para la búsqueda según un año, será constante si no existe dicha centuria, y lineal para acceder a todos los eventos históricos, el más complejo es buscar según los "tags" su complejidad es logarítmica para buscar un elemento (según la stl de set) en el conjunto de todos los tags que se han usado en un evento histórico en dicha centuria y suponemos que no lo hemos encontrado, por el contrario, si existe, dependerá de cuantas veces lo hemos encontrado y en cuántos nodos podemos encontrarlo, podemos asumir una eficiencia polinómica.