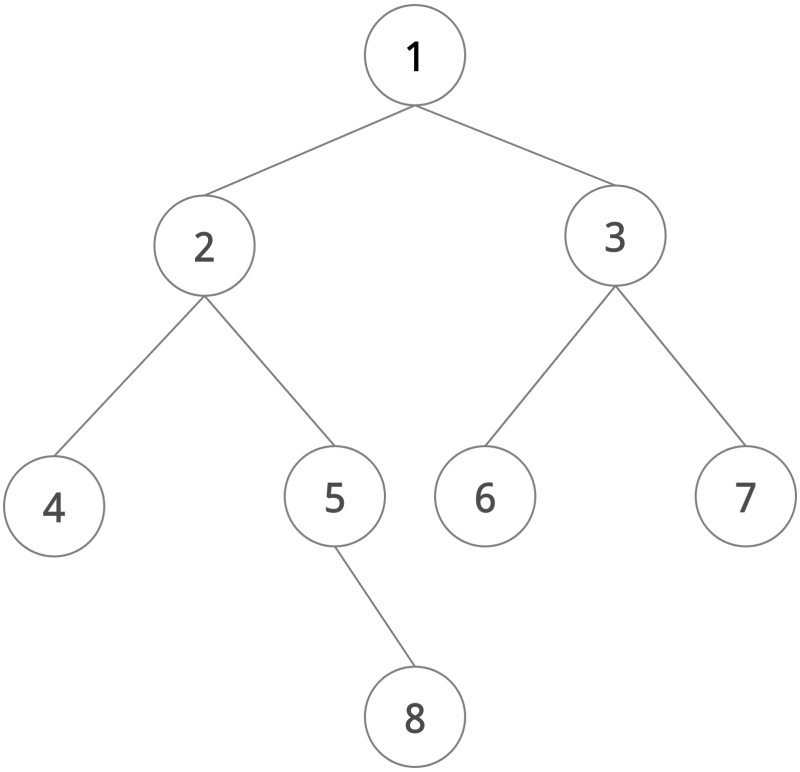
Reto 4: TDA no lineales I

Procedimiento de escritura y lectura a un árbol binario en disco.

Pablo Olivares Martínez

En clase, la forma que vimos de escritura y lectura de árbol binario en disco fue usando un recorrido de preorden con centinelas. Esto es, dado un **árbol A**, los ***nodos*** serán denotados por ***ne***, donde ***e*** es la etiqueta que le corresponde al nodo y por otro lado, los ***centinelas*** denotados por ***x***, indican que esa posición está vacía. Por ejemplo, dado A tal que:



Por los métodos de escritura utilizados por el profesor, obtendríamos que **A = n1n2n4xxn5xn8n3n6xxn7xx**. Este método, como hemos podido comprobar, es sencillo de entender además de que proporciona una manera muy sencilla de implementación del código. Sin embargo, para árboles con numerosos niveles tendríamos cadenas bastante largas donde se referencia muchas veces a posiciones nulas mediante centinelas. Por ello, yo propongo una implementación que requiere de unas pocas líneas más de código pero ahorra una gran cantidad de carácteres, lo que supone un ahorro en memoria notable para árboles grandes.

Mi idea consiste en en deshacernos de los centinelas e indicar de manera directa si un nodo tiene 1 hijo, 2 hijos o ninguno. Para ello, vamos a sustituir n en función de la familia directa del nodo. Por ello, si el nodo tiene **2 hijos** lo denotaremos por **n**; si tiene un **hijo izquierda**, ese nodo se denotará por **i**; de la misma forma, si tiene un **hijo derecha** será **d**; si no tiene hijos, es decir, es una **hoja**, lo denotaremos por **h** y si está **vacío x**. De esta manera, si tomamos de ejemplo el árbol A, este nos quedaría de forma que **A = n1n2h4d5h8n3h6h7**, lo cual es mucho más reducido pero igual de intuitivo que la escritura y lectura por centinelas en preorden.

El precio a pagar por reducir este espacio en memoria no es muy alto, pues tan solo habría que añadir unas pocas comprobaciones condiciones muy sencillas que establezcan directamente los valores tal y como corresponden en función del tipo de nodo.

La implementación quedaría de la siguiente forma:

```
//Escritura del árbol
template <typename T>
void Escribe(ostream &os, const info_nodo<T>* &n) {
    if (n != nullptr) {
        if ( n->hijoizq != nullptr && n->hijodcha != nullptr) {
            os << 'n' << n->et;
            Escribe(os, n->hijoizq);
            Escribe(os, ->hijodcha);
        }
        else if ( n->hijoizq != nullptr && n->hijodcha == nullptr) {
            os << 'i' << n->et;
            Escribe(os, n->hijoizq);
        }
        else if ( n->hijoizq == nullptr && n->hijodcha != nullptr) {
            os << 'd' << n->et;
            Escribe(os, n->hijodcha);
        }
        else if ( n->hijoizq == nullptr && n->hijodcha == nullptr) {
            os << 'h' << n->et;
        }
    }
    else if (n==nullptr)
        os << 'x';
}
```

```
//Lectura del árbol
template <typename T>
void Lee(istream &is, info_nodo<T>* &n) {
    char c;
    c = is.get();

    if (is) {
        if (c == 'x')
            n = nullptr;
        else {
            T e;
            is >> e;
            n = new info_nodo<T>(e);

            switch (c) {
                case 'n':
                    Lee(is, n->hijoizq);
                    Lee(is, n->hijodcha);
                    n->hijoizq->padre = n;
                    n->hijodcha->padre = n;
                    break;
                case 'i':
                    lee(is, n->hijoizq);
                    n->hijoizq->padre = n;
                    n->hijodcha = nullptr;
                    break;
```

```
                case 'd':
                    Lee(is, n->hijodcha);
                    n->hijodcha->padre = n;
                    n->hijoizq = nullptr;
                    break;
                case 'h':
                    n->hijoizq = nullptr;
                    n->hijodcha = nullptr;
                    break;
            }
        }
    }
}
```