

Entrega TDA no lineales I

Reto 3: TDA no lineales I

Victoria

30/11/2025

Objetivo

Diseñar procedimiento para escribir y leer en disco un árbol binario de forma que se recupere la estructura jerárquica de forma única.

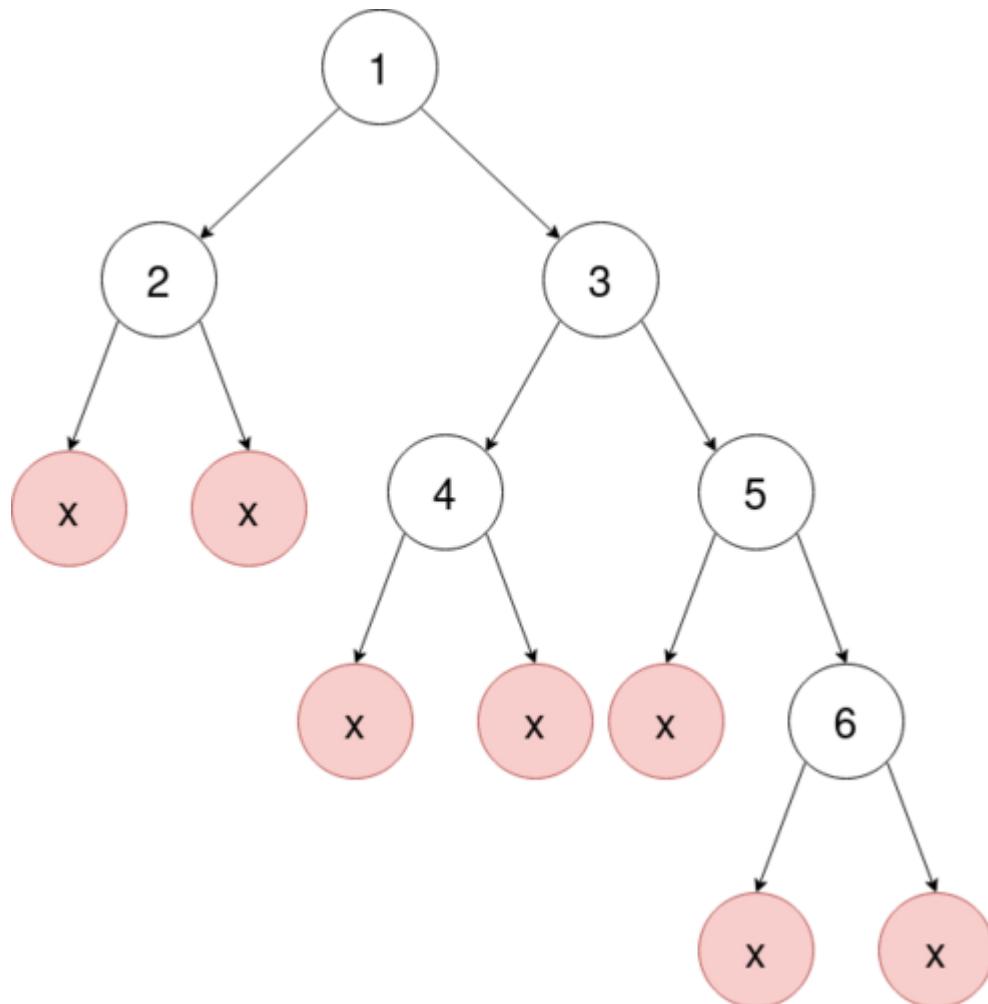
1. Escribir un árbol binario en una cadena de texto (función `serialize`)
2. Transformar una cadena de texto en un árbol binario (función `deserialize`)

Diseño

- He diseñado e implementado una variación de la representación por niveles con centinelas.
- El algoritmo agrupa los caracteres que indican nodos nulos para que ocupen menos espacio.

Por ejemplo:

El árbol:



Se representa:

- Por niveles con centinelas: 1 2 3 * * 4 5 * * * 6 * *
- Versión optimizada: 1 2 3 *2 4 5 *3 6 *2

Implementación

```
/**  
 * Método para serializar un árbol binario  
 *  
 * El método elegido es recorrido por niveles con centinelas optimizados:  
 * - Si la representación tiene varios caracteres centinela (*, *, *)  
 * - Agrupamos estos caracteres ("*, *, *" => "*3,")  
 *  
 * Dado el siguiente árbol binario:  
 *  
 *      1  
 *      /   \  
 *     2     3  
 *    /\   / \\  
 *   * *   4   5  
 *      /\   / \  
 *      * *   6  
 *          / \  
 *          * *  
 *  
 * La función serialize genera:  
 *  
 * serialize(tree) = "1,2,3,*2,4,5,*3,6,*2"  
 *  
 * En las funciones para serializar y deserializar el árbol, asumo que  
 * los parámetros tienen sentido y no son árboles nulos para simplificar  
 * el código.  
 *  
 * @file leveorder.cpp  
 * @author Victoria  
 */  
  
#include <string>  
#include <queue>  
#include <iostream>  
#include <sstream>  
using namespace std;  
  
/**  
 * Nodo de árbol binario simple  
 */  
struct TreeNode {  
    int data;
```

```

TreeNode * left;
TreeNode * right;

TreeNode(int x) : data(x), left(nullptr), right(nullptr) {}

};

/***
 * Función para serializar un árbol binario
 *
 * @param node Puntero a la raíz del árbol binario
 * @return representación en string del árbol binario
 */
string serialize(TreeNode* node) {
    string out;
    queue<TreeNode*> q; // para el recorrido por niveles se usa una cola
    q.push(node);

    // Para la optimización de centinelas es necesario
    // contar el número de nodos nulos seguidos
    int null_count = 0;
    bool prev_was_null = false;

    while (!q.empty()) {
        TreeNode* cur = q.front();
        q.pop();
        if (cur == nullptr) {
            prev_was_null = true;
            null_count++; // contar caracteres nulos consecutivos
        } else {
            if (prev_was_null) {
                // añadir centinelas optimizados
                out += "*" + to_string(null_count) + ",";
            }
            prev_was_null = false;
            null_count = 0;

            out += to_string(cur->data) + ',';
        }

        if (cur != nullptr) {
            q.push(cur->left);
            q.push(cur->right);
        }
    }

    // Añadir centinelas del final
    if (prev_was_null) {
        out += "*" + to_string(null_count) + ",";
    }

    return out;
}
*/

```

```

* @brief Convierte la representación por niveles con centinelas
* optimizados en una representación por niveles con centinelas normal.
*
* @code{.cpp}
* string str = "*10,2,3,9,*2";
* cout << decompress_bintree(str); // => *,*,*,*,*,*,*,*,*,2,3,9,*,*
* @endcode
*
* @param data Árbol binario representado por niveles con centinelas
* optimizados
* @return Árbol binario representado por niveles con centinelas
*/
string decompress_bintree(string data) {
    string out;
    for (int i = 0; i < data.size(); i++) {
        if (data[i] == '*' && data[i+1] != ',') {

            i++;
            int j = i;
            while (isdigit(data[j])) j++;

            int nulls = stoi(data.substr(i,j));

            for (int n = 0; n < nulls; n++) {
                out += "*,";

            }
            i = j;

        } else {
            out += data[i];
        }
    }

    return out;
}

/**
* @brief Reconstruye un árbol binario dada su representación
* por niveles con centinelas optimizados.
*
* Usa stringstream y getline para separar el string por las comas.
*
* @param data Representación por niveles con centinelas optimizados
* @return Puntero a la raíz del árbol binario generado
*/
TreeNode* deserialize(string data) {
    string decomp = decompress_bintree(data);
    stringstream ss(decomp);
    string str;
    getline(ss,str,',');
    queue<TreeNode*> q; // usa una cola para el recorrido por niveles

    // El primer dato nunca es null (*)
    TreeNode *root = new TreeNode(stoi(str));

```

```

q.push(root);

while (!q.empty()) {
    TreeNode* cur = q.front();
    q.pop();

    getline(ss,str,','); // hijo izquierdo de cur
    if (str == "*") {
        cur->left = nullptr;
    } else {
        TreeNode* left_node = new TreeNode(stoi(str));
        cur->left = left_node;
        q.push(cur->left);
    }

    getline(ss,str,','); // hijo derecho de cur
    if (str == "*") {
        cur->right = nullptr;
    } else {
        TreeNode* right_node = new TreeNode(stoi(str));
        cur->right = right_node;
        q.push(cur->right);
    }
}

return root;
}

/***
 * Imprime el recorrido en inorder de un árbol binario.
 * Usado en los tests.
 */
void print_inorder(TreeNode *node) {
    if (node == nullptr) return;
    print_inorder(node->left);
    cout << node->data << ", ";
    print_inorder(node->right);
}

/***
 * Imprime el recorrido en preorder de un árbol binario.
 * Usado en los tests.
 */
void print_preorder(TreeNode *node) {
    if (node == nullptr) return;
    cout << node->data << ", ";
    print_preorder(node->left);
    print_preorder(node->right);
}

void test_bintree() {
    // Construir árbol
}

```

```

TreeNode n1(1);
TreeNode n2(2);
TreeNode n3(3);
TreeNode n4(4);
TreeNode n5(5);
TreeNode n6(6);

n5.right = &n6;
n3.right = &n5;
n3.left = &n4;
n1.right = &n3;
n1.left = &n2;

// Serializar arbol
string ser = serialize(&n1);
cout << "Árbol serializado:" << endl;
cout << ser << endl;

// Deserializar y recorrer árbol
TreeNode *nuevo = deserialize(ser);
cout << "\nInorden del árbol reconstruido: " << endl;
print_inorder(nuevo);
cout << "\nPreorden del árbol reconstruido: " << endl;
print_preorder(nuevo);

cout << "\n\nÁrbol reconstruido serializado otra vez:" << endl;
cout << serialize(nuevo) << endl;
}

void test_bintree_2() {
// Construir árbol
TreeNode n1(1);
TreeNode n2(2);
TreeNode n3(3);
TreeNode n4(4);
TreeNode n5(5);
TreeNode n6(6);
TreeNode n7(7);
TreeNode n8(8);

n5.right = &n8;
n2.left = &n4;
n2.right = &n5;
n3.left = &n6;
n3.right = &n7;
n1.left = &n2;
n1.right = &n3;

// Serializar arbol
string ser = serialize(&n1);
cout << "Árbol serializado:" << endl;
cout << ser << endl;

```

```

// Deserializar y recorrer árbol
TreeNode *nuevo = deserialize(ser);
cout << "\nInorden del árbol reconstruido: " << endl;
print_inorder(nuevo);
cout << "\nPreorden del árbol reconstruido: " << endl;
print_preorder(nuevo);

cout << "\n\nÁrbol reconstruido serializado otra vez:" << endl;
cout << serialize(nuevo) << endl;
}

int main() {
    cout << "    == TEST 1 ==\n";
    test_bintree();
    cout << "\n\n    == TEST 2 ==\n";
    test_bintree_2();
}

```

Ejemplo de ejecución

El código de la función `main` imprime por pantalla este resultado:

```

== TEST 1 ==
Árbol serializado:
1,2,3,*2,4,5,*3,6,*2,

Inorden del árbol reconstruido:
2, 1, 4, 3, 5, 6,
Preorden del árbol reconstruido:
1, 2, 3, 4, 5, 6,

Árbol reconstruido serializado otra vez:
1,2,3,*2,4,5,*3,6,*2,


== TEST 2 ==
Árbol serializado:
1,2,3,4,5,6,7,*3,8,*6,

Inorden del árbol reconstruido:
4, 2, 5, 8, 1, 6, 3, 7,
Preorden del árbol reconstruido:
1, 2, 4, 5, 8, 3, 6, 7,

Árbol reconstruido serializado otra vez:
1,2,3,4,5,6,7,*3,8,*6,

```