

Bitácora de consultas al LLM

Como primera pauta se le pidió a la inteligencia artificial que tomara el rol de experto en C++, con el fin de que nos pudiera brindar mejores propuestas a la hora de comenzar a pedirle ayuda. Se le proporcionó la plantilla con la que se debía trabajar y se le explicaron algunas pautas para iniciar, la IA nos presentó algunas ideas para ejecutar el árbol de búsqueda binaria.

1. Construcción del Árbol:

- Crear el BST en el que cada nodo representa un carácter con su traducción en código Morse.

2. Traducción:

- Implementar una función para traducir un texto a código Morse.
- Crear una función para traducir el código Morse de vuelta a texto.

3. Ejemplo de Funcionamiento:

- Dar ejemplos de entrada y salida para verificar el funcionamiento.

4. Requisitos adicionales:

- Manejo adecuado de espacios entre palabras.
- Ignorar caracteres fuera del rango especificado (A-Z y 0-9).

Voy a revisar el archivo de plantilla (`BST.h`) para comprender la estructura inicial y ver cómo podemos agregar estas funcionalidades.

La IA nos explicó los contenidos del archivo que se le compartió y propuso hacerle algunos cambios a la plantilla propuesta que trabaja con los árboles de búsqueda binaria (**BST.h**).

La plantilla `BST.h` define la estructura de un Árbol Binario de Búsqueda (BST), incluyendo operaciones como inserción, eliminación, búsqueda, y métodos auxiliares para manejo de memoria y duplicados. Para adaptar esta estructura a las instrucciones de traducir entre caracteres alfanuméricos y código Morse, podemos hacer algunos ajustes y agregar funcionalidades específicas. Aquí te propongo los pasos:

1. Agregar Traducciones de Morse en el Nodo

Modifica la estructura `BinaryNode` para que cada nodo almacene un carácter y su código Morse.

```
cpp
struct BinaryNode {
    char character;
    std::string morseCode;
    BinaryNode *left;
    BinaryNode *right;

    BinaryNode(char ch, std::string morse, BinaryNode *lt, BinaryNode *rt)
        : character(ch), morseCode(morse), left(lt), right(rt) {}
};
```

2. Insertar Nodos con Caracteres y Código Morse

Creo un método específico para insertar cada letra (A-Z) y número (0-9) junto con su código Morse.

```
cpp
void insertMorse(char character, const std::string &morseCode) {
    insertMorse(character, morseCode, root);
}

void insertMorse(char character, const std::string &morseCode, BinaryNode *&t) {
    if (t == nullptr) {
        t = new BinaryNode(character, morseCode, nullptr, nullptr);
    } else if (character < t->character) {
        insertMorse(character, morseCode, t->left);
    } else if (t->character < character) {
        insertMorse(character, morseCode, t->right);
    }
}
```

3. Traducción de Texto a Morse

Implementa una función que busque cada letra en el árbol y acumule su código Morse.

```
cpp Copia codice

std::string translateToMorse(const std::string &text) {
    std::string morseTranslation;
    for (char ch : text) {
        std::string morse = findMorseCode(ch, root);
        if (!morse.empty()) {
            morseTranslation += morse + " ";
        }
    }
    return morseTranslation;
}

std::string findMorseCode(char ch, BinaryNode *t) const {
    if (t == nullptr) return "";
    if (ch < t->character) return findMorseCode(ch, t->left);
    if (ch > t->character) return findMorseCode(ch, t->right);
    return t->morseCode;
}
```

4. Traducción de Morse a Texto

Para esta función, podrías realizar una búsqueda inversa, recorriendo el árbol en búsqueda del código Morse.

Con estas modificaciones, el árbol BST almacenará caracteres con sus códigos Morse y podrá realizar traducciones en ambas direcciones. Si tienes más dudas o necesitas ayuda adicional, ¡aquí estaré!

Se le pidió que nos compartiera las letras de la A a la Z y los números del 0 al 9 con el siguiente formato **bst.insertMorse('A', "-.")**; para ahorrar un poco de tiempo y no escribir más de 30 caracteres y sus respectivos códigos morse.

```
cpp Copia codice

int main() {
    MorseBST bst;

    // Insertar caracteres A-Z con sus códigos Morse
    bst.insertMorse('A', "-.");
    bst.insertMorse('B', "-...");
    bst.insertMorse('C', "-.-.");
    bst.insertMorse('D', "-..");
    bst.insertMorse('E', ".");
    bst.insertMorse('F', "..-.");
    bst.insertMorse('G', "--.");
    bst.insertMorse('H', "...");
    bst.insertMorse('I', "..");
    bst.insertMorse('J', ".---");
    bst.insertMorse('K', "-.-");
    bst.insertMorse('L', "-...");
```

En nuestro código tuvimos un problema en el que cuando queríamos pasar de texto a morse, solo se mostraba el código de la primera palabra de prueba, es decir, al usar HELLO WORLD de prueba, únicamente imprimía los códigos morse de HELLO y al convertir de morse a texto, únicamente se imprimía la letra H de la palabra HELLO, entonces le pedimos ayuda compartiéndole nuestro código y adicionalmente le añadió la línea de código encerrada en rojo a nuestro código.



Para resolver los problemas:

1. **Texto a Morse:** El código no imprime la segunda palabra porque `std::cin >> text` solo lee hasta el primer espacio, por lo que solo se captura la primera palabra ingresada. Cambiemos `std::cin >> text` por `std::getline(std::cin, text)` para leer toda la línea incluyendo los espacios.
2. **Morse a Texto:** Solo se traduce la primera letra porque el código espera un espacio simple entre letras, pero necesita un delimitador especial como `" / "` para separar palabras. También actualizaremos el procesamiento del código Morse para manejar correctamente los espacios entre letras y palabras.

```
std::cout << "Para traducir de texto a morse, escribe 1 : "<< std::endl;
std::cout << "Para traducir de morse a texto, escribe 2 : "<< std::endl;
std::cin >> lector;
std::cin.ignore(); // Limpiar el buffer antes de getline
```

Sin embargo. Al hacer la implementación con un BST, el árbol queda desbalanceado, por lo que se procede a realiza una implementación de las funciones con un árbol de búsqueda AVL.