

My personal calendary

-Cristian Andrés Reinales Herrera

–Nicolas Fernando Botero Andramunio

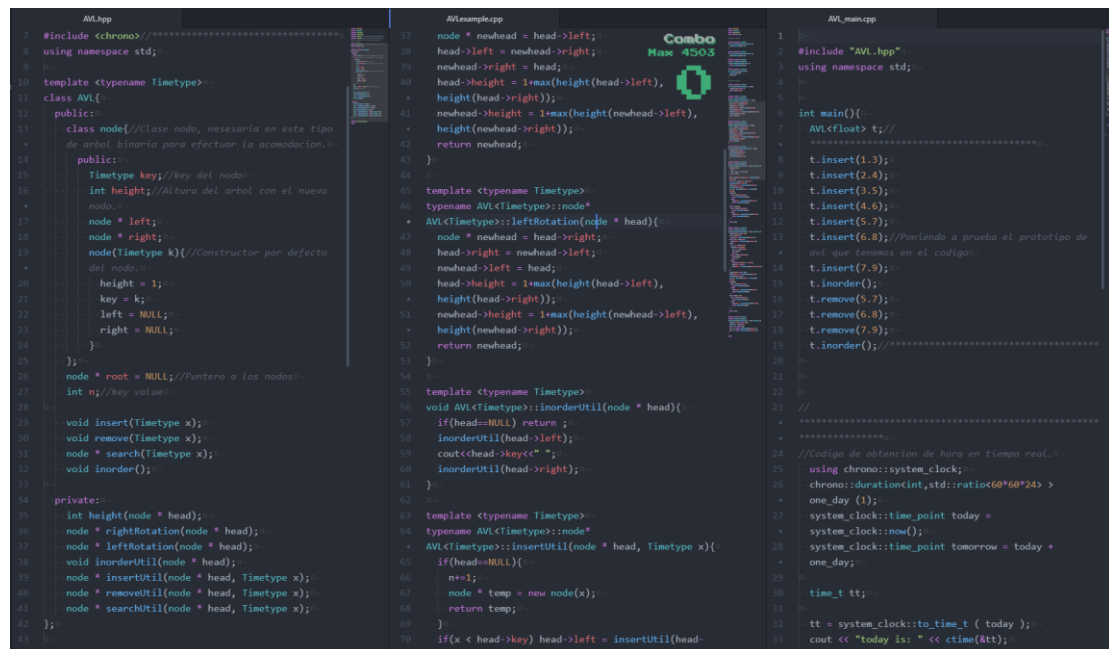
-Miguel Angel Salamanca Ortiz

Repositorio: <https://github.com/CristianReinales/Algorith-project.git>

Objetivos alcanzados:

- El primer avance que logramos en el proyecto fue definir que tipo de árbol binario usar para nuestra idea de proyecto, después de evaluar entre AVL, Red-Black y WBL Nos decidimos por usar AVL debido a que gracias a que este tipo de árbol se auto equilibra solo, sabemos que la alarma más próxima a sonar será la que este en el nodo de menor valor en el árbol, en otras palabras, aumenta la eficiencia del programa.

Una vez que nos decidimos por AVL conseguimos un prototipo de un árbol binario AVL y lo separamos en 3 archivos: hpp, cpp y main, con esto ya tenemos mas o menos una plantilla sobre la cual empezar a trabajar.



```
AVL.hpp
1 #include <chrono>
2 using namespace std;
3
4 template <typename T>
5 class AVL {
6 public:
7     class Node {
8     public:
9         T key;
10        int height;
11        Node * left;
12        Node * right;
13        Node(T key) {
14            key = key;
15            height = 1;
16            left = NULL;
17            right = NULL;
18        }
19    };
20    Node * root;
21    int n;
22
23    void insert(T x);
24    void remove(T x);
25    Node * search(T x);
26    void inorder();
27
28 private:
29     int height(Node * head);
30     Node * rightRotation(Node * head);
31     Node * leftRotation(Node * head);
32     void inorderUtil(Node * head);
33     Node * insertUtil(Node * head, T x);
34     Node * removeUtil(Node * head, T x);
35     Node * searchUtil(Node * head, T x);
36 };
37
38 AVLmain.cpp
39 #include "AVL.hpp"
40 using namespace std;
41
42 int main() {
43     AVL<int> t;
44     t.insert(1);
45     t.insert(2);
46     t.insert(3);
47     t.insert(4);
48     t.insert(5);
49     t.insert(6);
50     t.insert(7);
51     t.inorder();
52     t.remove(5);
53     t.remove(6);
54     t.remove(7);
55     t.inorder();
56 }
57
58 AVLmain.cpp
59 #include "AVL.hpp"
60 using namespace std;
61
62 int main() {
63     AVL<int> t;
64     t.insert(1);
65     t.insert(2);
66     t.insert(3);
67     t.insert(4);
68     t.insert(5);
69     t.insert(6);
70     t.insert(7);
71     t.inorder();
72     t.remove(5);
73     t.remove(6);
74     t.remove(7);
75     t.inorder();
76 }
77
78 AVLmain.cpp
79 #include "AVL.hpp"
80 using namespace std;
81
82 int main() {
83     AVL<int> t;
84     t.insert(1);
85     t.insert(2);
86     t.insert(3);
87     t.insert(4);
88     t.insert(5);
89     t.insert(6);
90     t.insert(7);
91     t.inorder();
92     t.remove(5);
93     t.remove(6);
94     t.remove(7);
95     t.inorder();
96 }
```

- Encontramos una librería estándar que nos ayuda a tener acceso a la hora y fecha del computador en tiempo real, lo cual es bastante útil, solo quedaría intentar convertir la fecha que nos retorna el código a una variable del tipo entero

```
// system_clock example
#include <iostream>
#include <ctime>
#include <ratio>
#include <chrono>
using namespace std;

int main () {
    using chrono::system_clock;
    chrono::duration<int, std::ratio<60*60*24> > one_day (1);
    system_clock::time_point today = system_clock::now();
    system_clock::time_point tomorrow = today + one_day;

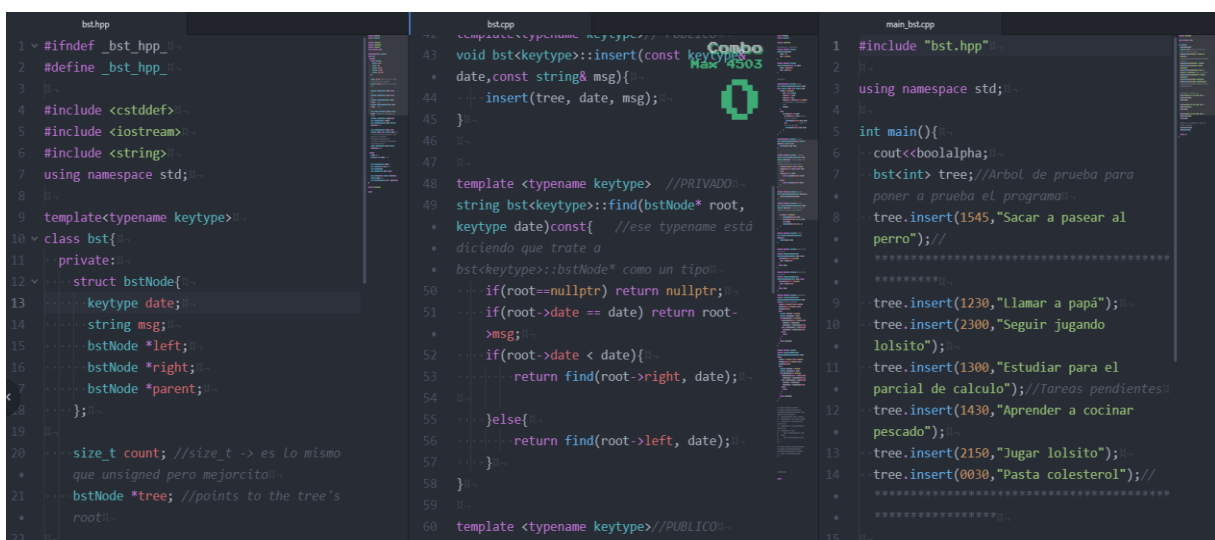
    time_t tt;

    tt = system_clock::to_time_t ( today );
    cout << "today is: " << ctime(&tt);

    tt = system_clock::to_time_t ( tomorrow );
    cout << "tomorrow will be: " << ctime(&tt);

    return 0;
}
```

- Modificamos la clase base de arboles binarios vistos en clase a una versión en la cual se crean nodos con keys=fechas y msg=mensajes, con esto tenemos la funcionabilidad de que con la hora podemos acceder a el mensaje que esta relacionado en ese nodo, el paso siguiente seria hacer una fusión del archivo AVL que modificamos y el archivo calendary_bst.



```
bst.hpp
1 #ifndef _bst_hpp_
2 #define _bst_hpp_
3
4 #include <cstdlib>
5 #include <iostream>
6 #include <string>
7 using namespace std;
8
9 template<typename keytype>
10 class bst{
11 private:
12     struct bstNode{
13         keytype date;
14         string msg;
15         bstNode *left;
16         bstNode *right;
17         bstNode *parent;
18     };
19
20     size_t count; //size_t -> es lo mismo
21     // que unsigned pero mejorito
22     bstNode *tree; //points to the tree's
23     // root
24
25 bst.cpp
26 template<typename keytype>
27 void bst<keytype>::insert(const keytype
28     * date, const string& msg){
29     insert(tree, date, msg);
30 }
31
32 template<typename keytype> //PRIVADO
33 string bst<keytype>::find(bstNode* root,
34     keytype date) const{ //ese typename está
35     // diciendo que trate a
36     // bst<keytype>::bstNode* como un tipo
37     //
38     if(root==nullptr) return nullptr;
39     if(root->date == date) return root-
40     >msg;
41     if(root->date < date){
42         return find(root->right, date);
43     }
44     else{
45         return find(root->left, date);
46     }
47 }
48
49 template<typename keytype> //PUBLICO
50 void bst<keytype>::find(bstNode* root,
51     keytype date) const{
52     if(root==nullptr) return nullptr;
53     if(root->date == date) return root-
54     >msg;
55     if(root->date < date){
56         return find(root->right, date);
57     }
58     else{
59         return find(root->left, date);
60     }
61 }
62
63 main_bst.cpp
64 #include "bst.hpp"
65
66 using namespace std;
67
68 int main(){
69     cout<<boolalpha;
70
71     bst<int> tree; //Arbol de prueba para
72     // poner a prueba el programita
73     tree.insert(1545, "Sacar a pasear al
74     // perro");
75
76     //
77     //
78     //
79     tree.insert(1230, "Llamar a papá");
80     tree.insert(2300, "Seguir jugando
81     // lolsito");
82
83     tree.insert(1300, "Estudiar para el
84     // parcial de calculo"); //Tareas pendientes
85     tree.insert(1430, "Aprender a cocinar
86     // pescado");
87     tree.insert(2150, "Jugar lolsito");
88     tree.insert(0030, "Pasta colesterol");
89
90     //
91     //
92     //
93 }
```

Algoritmos y estructuras de datos utilizadas:

1. AVL Binary search tree & Normal Binary search tree
2. Librerías: ctime, ratio y chrono.

Nuevos objetivos:

1. Implementar el concepto de nodo con fecha y mensaje a el árbol binario AVL.
2. Empezar a revisar como haremos la interfaz.
3. Aprender a trabajar con el dato tipo fecha que nos retorna ctime.

