Parcial_practico_2

June 2, 2023

1 Parcial práctico 2 - UNCode

Integrantes:

- Andrés Felipe Infante Hernández
- Manuel Camilo Rincon Blanco
- Joshua Cardona Toro
- Camilo Chitivo Cerinza

1.1 Máxima sabiduría

Dificultad:

La representación entre árboles AVL y árboles como arreglo es muy diferente, por lo que no se pudo usar funciones previas.

Solución:

Se creó una función mediante la cual se pudiera rellenar el árbol binario desde cero.

Dificultad:

Era difícil saber a qué altura estaba cada nodo basado en la función recursiva con la que se creó.

Solución:

Se añadió un atributo a los nodos para conocer su altura, y se creó una función recursiva para calcularlos cada vez.

1.1.1 Código

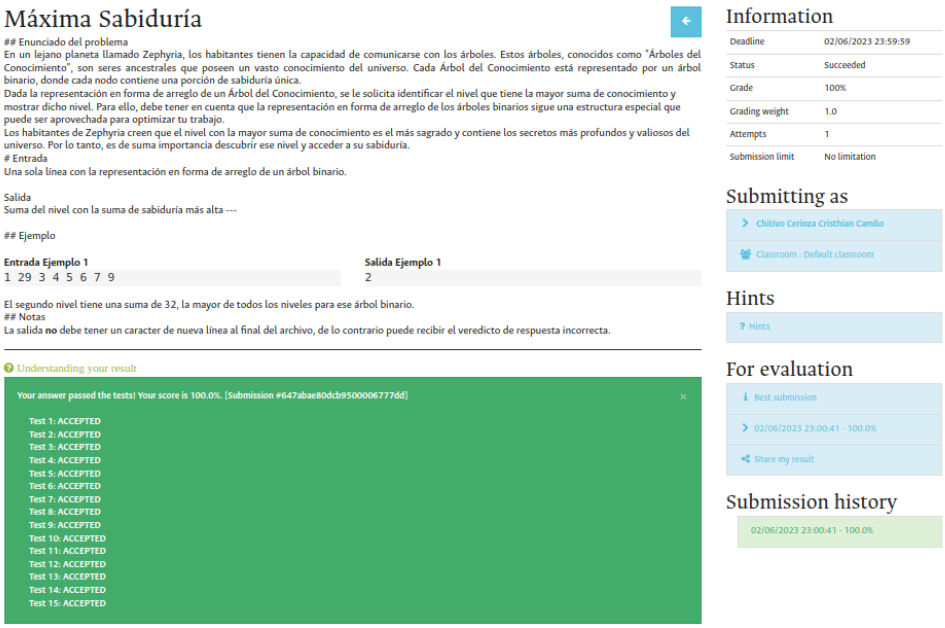
```
#include <iostream>
using namespace std;

struct treeNode{
    int numbah;
    treeNode* left;
    treeNode* right;
    int height;
};
treeNode* createHead(int numbah){
    treeNode* newNode = new treeNode;
    newNode->numbah = numbah;
```

```
    newNode->left = nullptr;
    newNode->right = nullptr;
    newNode->height = 1;
    return newNode;
}
void createNode(treeNode* head, int numbah){
        treeNode* newNode = new treeNode;
        newNode->numbah = numbah;
        newNode ->left = nullptr;
        newNode ->right = nullptr;
        newNode->height = head->height+1;
    if(head->left == nullptr){
        head->left = newNode;
    }else if(head->left->left!=nullptr and head->left->right!=nullptr){
        delete newNode;
        createNode(head->right,numbah);
    }else if(head->right == nullptr){
        head ->right = newNode;
    }else{
        createNode(head->left,numbah);
    }
}
void printInorder(struct treeNode* node,int sumarray[]){
    if (node == NULL)
        return;
    printInorder(node->left,sumarray);
    sumarray[node->height]+=node->numbah;
    printInorder(node->right,sumarray);
}

int main(){
    int numbah;
    cin >> numbah;
    treeNode* head = createHead(numbah);
    while(cin >> numbah){
     createNode(head,numbah);
    }
    int sumarray[10] = {0};
    printInorder(head,sumarray);
    int max=0;
    for(int i=0;i<10;i++){
        if(sumarray[i]>max){
            max = sumarray[i];
        }
    }
    cout << max;
    return 0;
}
```

1.1.2 Resultado



1.2 La Serpiente

Dificultad:

El principal problema en el código era presentado debido a que la función `recorrido_zigzag()` no realizaba el 'zig-zag' debidamente, en lugar de alternar entre sumar calorías y omitir el nodo, siempre sumaba calorías de cada nodo en el árbol.

Solución:

Se agregó la variable `omitir_nodo`, esta actualiza el valor de `omitir_nodo` al final de cada iteración para invertir su valor, de modo que en la siguiente iteración se realice la acción opuesta.

1.2.1 Código

```
class Serpiente_Maiz:
    def __init__(self, arbol, calorias_minimas):
        self.arbol = arbol
        self.calorias_minimas = calorias_minimas
        self.calorias_actuales = 0
        self.sobrevive = False

    def consumir_calorias(self, nodo):
        self.calorias_actuales += nodo
        if self.calorias_actuales >= self.calorias_minimas:
            self.sobrevive = True

    def recorrido_zigzag(self):
        omitir_nodo = True  # Variable para alternar entre consumir y omitir un nodo
```

```
        for nodo in self.arbol:
            if omitir_nodo:
                omitir_nodo = False
            else:
                self.consumir_calorias(nodo)
                omitir_nodo = True

            if self.sobrevive:
                break

    def verificar_supervivencia(self):
        self.recorrido_zigzag()
        if self.sobrevive:
            print("Sobrevive", end='')
        else:
            print("Muere", end='')


arbol = list(map(int, input().split()))
calorias_minimas = int(input())

serpiente = Serpiente_Maiz(arbol, calorias_minimas)
serpiente.verificar_supervivencia()
```

1.2.2 Resultado