

# 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

### Máxima Sabiduría

**## Enunciado del problema**  
En un lejano planeta llamado Zephyria, los habitantes tienen la capacidad de comunicarse con los árboles. Estos árboles, conocidos como "Árboles del Conocimiento", son seres ancestrales que poseen un vasto conocimiento del universo. Cada Árbol del Conocimiento está representado por un árbol binario, donde cada nodo contiene una porción de sabiduría única.  
Dada la representación en forma de arreglo de un Árbol del Conocimiento, se le solicita identificar el nivel que tiene la mayor suma de conocimiento y mostrar dicho nivel. Para ello, debe tener en cuenta que la representación en forma de arreglo de los árboles binarios sigue una estructura especial que puede ser aprovechada para optimizar tu trabajo.  
Los habitantes de Zephyria creen que el nivel con la mayor suma de conocimiento es el más sagrado y contiene los secretos más profundos y valiosos del universo. Por lo tanto, es de suma importancia descubrir ese nivel y acceder a su sabiduría.  
**# Entrada**  
Una sola línea con la representación en forma de arreglo de un árbol binario.

**Salida**  
Suma del nivel con la suma de sabiduría más alta ---

**## Ejemplo**

| Entrada Ejemplo 1 | Salida Ejemplo 1 |
|-------------------|------------------|
| 1 29 3 4 5 6 7 9  | 2                |

El segundo nivel tiene una suma de 32, la mayor de todos los niveles para ese árbol binario.

**## Notas**  
La salida **no** debe tener un carácter de nueva línea al final del archivo, de lo contrario puede recibir el veredicto de respuesta incorrecta.

#### Information

|                  |                     |
|------------------|---------------------|
| Deadline         | 02/06/2023 23:59:59 |
| Status           | Succeeded           |
| Grade            | 100%                |
| Grading weight   | 1.0                 |
| Attempts         | 1                   |
| Submission limit | No limitation       |

#### Submitting as

> Chitivo Cerinza Cristhian Camila

🏠 Classroom : Default classroom

#### Hints

🔍 HINTS

#### For evaluation

📄 Best submission

> 02/06/2023 23:00:41 - 100.0%

🔗 Share my result

#### Submission history

02/06/2023 23:00:41 - 100.0%

#### Understanding your result

Your answer passed the tests! Your score is 100.0%. [Submission #647abae80dc39500006777dd]

- Test 1: ACCEPTED
- Test 2: ACCEPTED
- Test 3: ACCEPTED
- Test 4: ACCEPTED
- Test 5: ACCEPTED
- Test 6: ACCEPTED
- Test 7: ACCEPTED
- Test 8: ACCEPTED
- Test 9: ACCEPTED
- Test 10: ACCEPTED
- Test 11: ACCEPTED
- Test 12: ACCEPTED
- Test 13: ACCEPTED
- Test 14: ACCEPTED
- Test 15: ACCEPTED

## 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, calorías_minimas):
        self.arbol = arbol
        self.calorías_minimas = calorías_minimas
        self.calorías_actuales = 0
        self.sobrevive = False

    def consumir_calorías(self, nodo):
        self.calorías_actuales += nodo
        if self.calorías_actuales >= self.calorías_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

### La Serpiente

#Enunciado del problema

La serpiente del maíz (*Pantherophis guttatus*) habita en América del Norte y es conocida por ser capaz de recorrer largas distancias. Para esto necesitan consumir diariamente una cantidad mínima de calorías  $c$ . La serpiente puede consumir todas las calorías que encuentre sin problema, pero si no consume la cantidad mínima muere. La serpiente va a recorrer diferentes lugares y su misión es identificar si la serpiente sobrevive o no.

El recorrido de la serpiente se realiza de la siguiente manera. En primer lugar, usted recibe la representación en forma de arreglo de un árbol binario. La serpiente recorre este árbol binario en una forma de zig-zag iniciando desde la raíz. La serpiente consume la cantidad de calorías de la raíz del árbol, luego omite un nodo y vuelve a consumir la cantidad de calorías del siguiente nodo al omitido, así hasta que llegue al final del recorrido. La idea entonces es identificar si la cantidad de calorías que logró consumir la serpiente en ese recorrido le permite sobrevivir. Todo esto se ilustra en la siguiente imagen:

[Imagen del árbol binario](https://drive.google.com/file/d/15\_uuuOKIqIjZblmTEPRZ05ZMq949bo/view?usp=share\_link)

En la imagen anterior, se arma al árbol de ejemplo, cuya representación en forma de arreglo es [1, 2, 3, 7, 6, 5, 4, 9, 8, 7, 6]. Luego, el recorrido de la serpiente inicia el zig-zag a la derecha de la raíz. Este recorrido se muestra con la línea azul. Los nodos en los que la serpiente consume calorías se marcan en verde. Para este caso, la serpiente debía consumir como mínimo 29 calorías, lo cual coincide con la suma de los nodos donde consumió calorías, por lo que la serpiente sobrevive.

## Restricciones - La cantidad mínima de calorías  $c$  está en el rango (60000, 100000) - El número de nodos del árbol binario está en el rango (20, 10000) - El valor de cada nodo está en el rango (2, 49)

## Entrada En la primera línea se recibe la representación en arreglo de un árbol binario con sus nodos separados por espacios. En la segunda línea está el número mínimo de calorías que debe consumir la serpiente.

## Salida Si la serpiente sobrevive, debe imprimir "Sobrevive". Si la serpiente no sobrevive, debe imprimir "Muere".

---

## Ejemplo

Entrada Ejemplo 1

1 2 3 7 6 5 4 9 8 7 6

29

Salida Ejemplo 1

Sobrevive

## Notas

La salida no debe tener un caracter de nueva línea al final del archivo, de lo contrario puede recibir el veredicto de respuesta incorrecta.

#### Understanding your result

Your answer passed the tests! Your score is 100.0%. [Submission #6479f19f0dc95000676cb9]

Test 1: ACCEPTED  
Test 2: ACCEPTED  
Test 3: ACCEPTED  
Test 4: ACCEPTED  
Test 5: ACCEPTED  
Test 6: ACCEPTED  
Test 7: ACCEPTED  
Test 8: ACCEPTED  
Test 9: ACCEPTED  
Test 10: ACCEPTED  
Test 11: ACCEPTED  
Test 12: ACCEPTED  
Test 13: ACCEPTED  
Test 14: ACCEPTED  
Test 15: ACCEPTED

### Information

|                  |                     |
|------------------|---------------------|
| Deadline         | 02/06/2023 23:59:59 |
| Status           | Succeeded           |
| Grade            | 100.0%              |
| Grading weight   | 1.0                 |
| Attempts         | 3                   |
| Submission limit | No limitation       |

### Submitting as

> Chitivo Cerinza Cristhian Camilo

Classroom - Default classroom

### Hints

? Hints

### For evaluation

i Best submission

> 02/06/2023 08:41:51 - 100.0%

Share my result

### Submission history

02/06/2023 08:41:51 - 100.0%

02/06/2023 07:25:05 - 0.0%

02/06/2023 07:23:39 - 0.0%