UNIVERSIDAD TECNOLÓGICA DE PANAMÁCENTRO REGIONAL DE CHIRIQUÍ
FACULTAD DE INGENIERÍA DE SISTEMAS COMPUTACIONALES

CARRERA:
Gestión y Desarrollo de Software

ACTIVIDAD No. 8

LABORATORIO No. 4

“Laboratorio 4”

ASIGNATURA: Estructura de Datos II

DOCENTE:
Profa. Nunehar Mondul

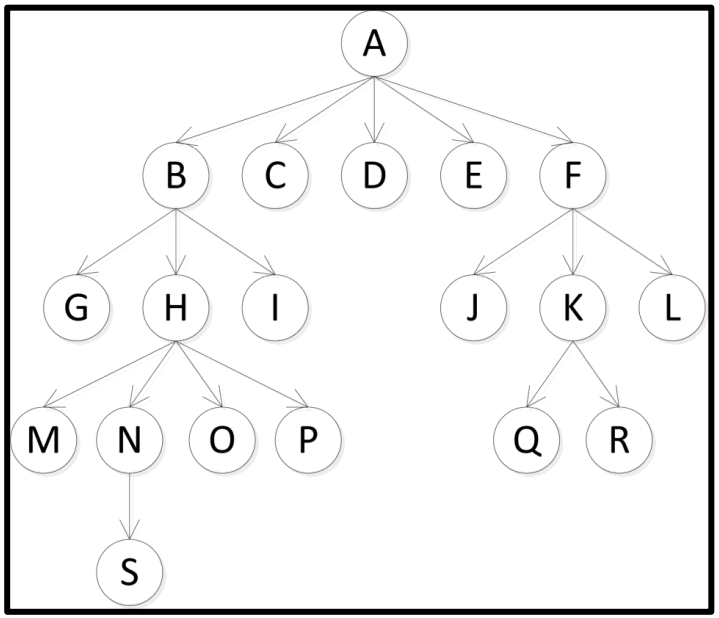
ESTUDIANTE/s:
Jorge Jiménez (4-826-874)
-
-

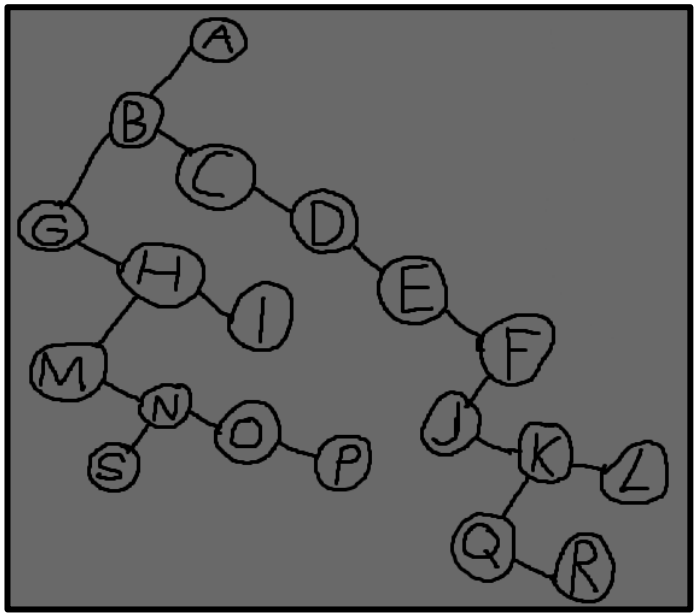
I SEMESTRE 2025

FECHA:
05/06/2025

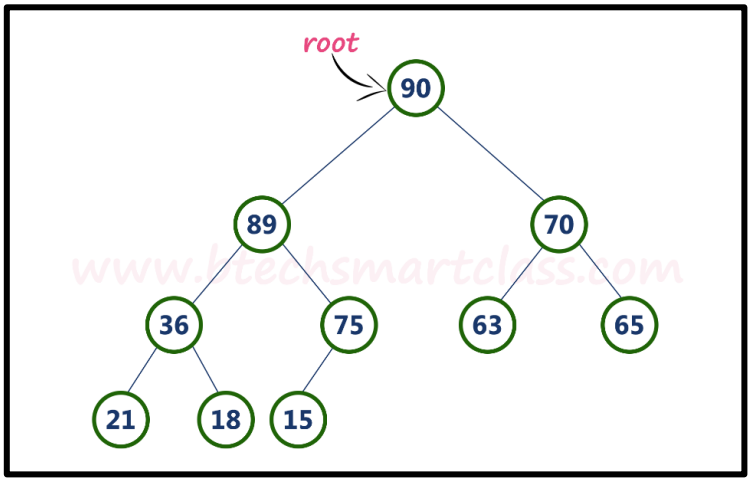
Desarrollo

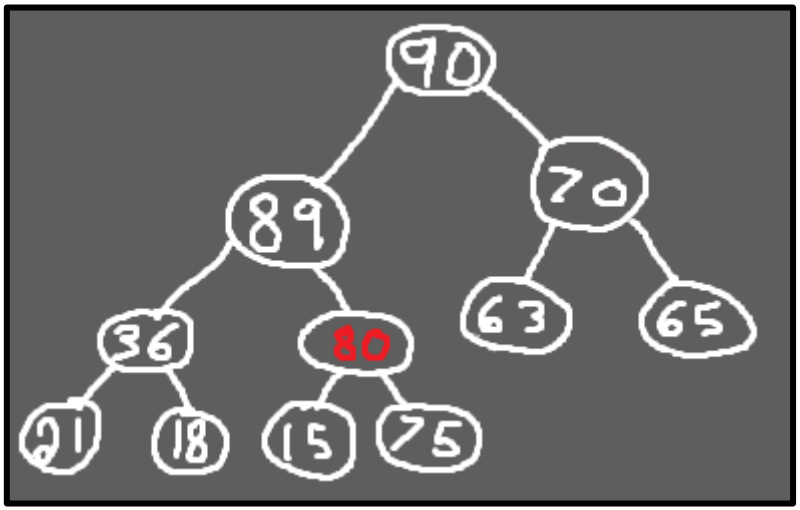
1. "Convertir el siguiente árbol general a binario."



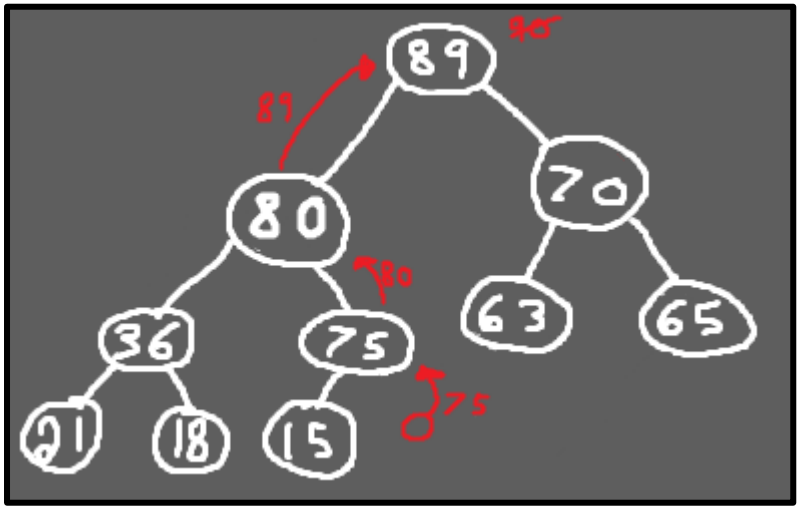


2. "Al siguiente max heap añadir nodo 80."





3. "Con el heap del 2 borrar la raíz y mostrar nuevo heap."



4. "¿Cuál es la utilidad de los heap en la programación?"
Permiten una gestión eficiente de los datos, especialmente cuando se necesita acceder rápidamente al elemento más grande (en un heap máximo) o más pequeño (en un heap mínimo).

5. "Aplicar algoritmo de Huffman a los siguientes valores 3, 7, 11, 25, 4."
Primero se ordenan: 3, 4, 7, 11, 25

Luego se combinan los dos menores: (3+4) = 7
Lista: 7, 7, 11, 25

De nuevo se combinan los dos menores: (7+7) = 14
Lista: 11, 14, 25

De nuevo se combinan los dos menores: (11+14) = 25
Lista: 25, 25

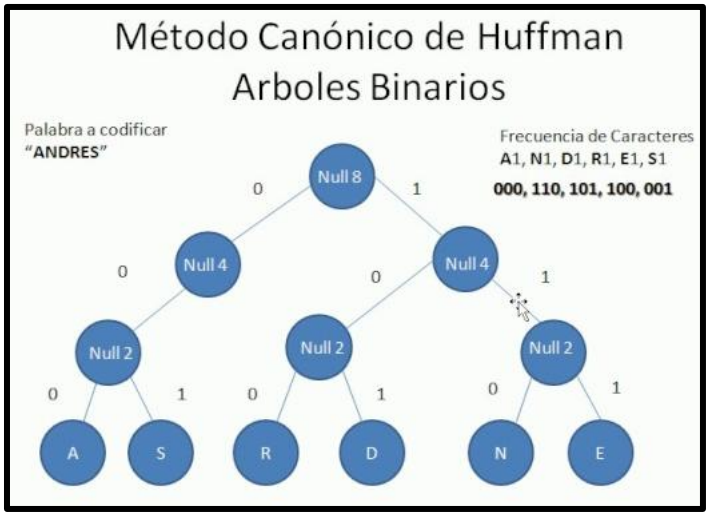
Y de ultimo se combinan los restantes: (25+25) = 50

6. "Investigar la utilidad del algoritmo de Huffman."

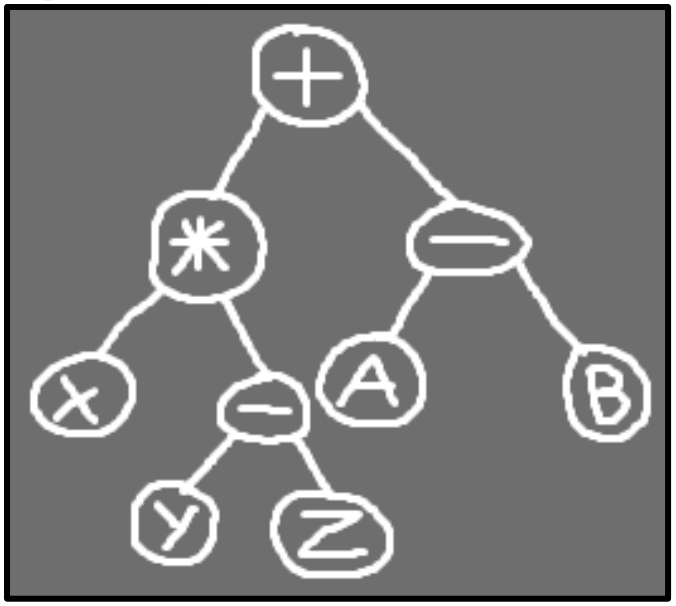
Se usa para compresión de datos sin pérdida. Asigna códigos más cortos a símbolos frecuentes y más largos a los menos frecuentes, reduciendo el tamaño total del archivo.

"¿Cómo se saca el Código huffman?"

Primero se construye un árbol de Huffman, asignando a cada carácter una frecuencia. Luego, se recorre el árbol desde la raíz hasta cada nodo hoja, asignando "0" a las ramas izquierdas y "1" a las ramas derechas. La secuencia de "0" y "1" resultante es el código Huffman para ese carácter.



7. "Hacer árbol de expresión con la siguiente operación (x*(y-z))+(a-b)."



8. ” Para el siguiente código de heap documentar que hace en cada paso y cambiarlo a max heap.”

HeapSort.java

Source | History

```java
package arbolesbin;
import java.util.Arrays;

public class HeapSort {
    public static void main(String[] args) {
        int[] arr = {5, 9, 3, 1, 8, 6, 20, 21, 19}; // añadi 3 más

        heapSort(arr);

        // Imprime el arreglo ordenado de mayor a menor (ahora si)
        System.out.println(Arrays.toString(arr));
    }

    public static void heapSort(int[] arr) {
        int n = arr.length;

        // Construye el "MAX" heap
        for (int i = n / 2 - 1; i >= 0; i--) {
            heapify(arr, n, i);
        }

        // Extrae elementos del heap uno por uno
        for (int i = n - 1; i >= 0; i--) {
            // Mueve la raíz al final
            int temp = arr[0];
            arr[0] = arr[i];
            arr[i] = temp;
```

```java
            // Llama a "heapify"
            heapify(arr, i, 0);
        }

        // Esto hace que salga de mayor a menor
        for (int i = 0; i < n / 2; i++) {
            int temp = arr[i];
            arr[i] = arr[n - 1 - i];
            arr[n - 1 - i] = temp;
        }
    }

// Reorganiza el subárbol para que cumpla la propiedad de max heap
    public static void heapify(int[] arr, int n, int i) {
        int largest = i; // Inicializa la raíz como el mayor
        int left = 2 * i + 1; // Hijo izquierdo
        int right = 2 * i + 2; // Hijo derecho

        // Si el hijo izquierdo es mayor que la raíz
        if (left < n && arr[left] > arr[largest]) {
            largest = left;
        }

        // Si el hijo derecho es mayor que el mayor actual
        if (right < n && arr[right] > arr[largest]) {
            largest = right;
```

```java
        }

        // Si la raíz no es el mayor, intercambia y sigue heapificando
        if (largest != i) {
            int swap = arr[i];
            arr[i] = arr[largest];
            arr[largest] = swap;

            // Llama recursivamente a heapify
            heapify(arr, n, largest);
        }
    }
}
```

Output - arbolesBin (run)

```
run:
[21, 20, 19, 9, 8, 6, 5, 3, 1]
BUILD SUCCESSFUL (total time: 0 seconds)
```