UNIVERSIDAD DE LAS FUERZAS ARMADAS ESPE

DEPARTAMENTO DE CIENCIAS DE LA COMPUTACIÓN

ESTRUCTURA DE DATOS

BÚSQUEDA BINARIA - HASH Y ÁRBOLES

NOMBRE: QUISHPE CHAVEZ DENISSE PAULINA
TASIPANTA CAGUAS MADELLYN ABIGAIL

NRC: 29852

FECHA: 17/12/2025

DOCENTE: SOLIS ACOSTA EDGAR FERNANDO

INGENIERÍA EN TECNOLOGÍAS DE LA INFORMACIÓN

Contenido

INGENIERÍA EN TECNOLOGÍAS DE LA INFORMACIÓN

BÚSQUEDA BINARIA Y HASH

EJERCICIO 1 – LIBRO DISEÑO DE ALGORITMOS Y SU CODIFICACION EN LENGUAJE C

Implementar la función de búsqueda binaria en Lenguaje C para localizar un número específico (nbus) dentro de un arreglo de números enteros (v), aprovechando la eficiencia del algoritmo divide y vencerás.

- La búsqueda se basa en el conocido método divide y vencerás.
- Búsqueda donde se requiere que los elementos estén ordenados.

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

// Función de búsqueda binaria
int busquedaBinaria(int v[], int n, int nbus) {
    int izq = 0;
    int der = n - 1;
    int iteracion = 1;

    cout << "\n=== Proceso de Búsqueda Binaria ===" << endl;
    cout << "Buscando el número: " << nbus << endl;
    cout << "Tamaño del arreglo: " << n << endl;

    while (izq <= der) {
        int medio = (izq + der) / 2;

        cout << "\n--- Iteración " << iteracion << " ---" << endl;
        cout << "Límites: izq = " << izq << ", der = " << der << endl;
        cout << "Índice medio: medio = (" << izq << " + " << der << ") / 2 = " << medio <<
endl;
        cout << "Valor en v[" << medio << "] = " << v[medio] << endl;

        // Elemento encontrado
        if (v[medio] == nbus)
        {
```

```
        cout << "\n¡Elemento encontrado!" << endl;
```

```
            cout << "v[" << medio << "] == " << nbus << endl;
            return medio;
        }
        // El elemento está en la mitad derecha
        else if (v[medio] < nbus) {
            cout << "Decisión: " << v[medio] << " < " << nbus << endl;
            cout << "El número está en la mitad derecha" << endl;
            cout << "Ajustando: izq = medio + 1 = " << medio << " + 1 = " << (medio + 1) <<
endl;
            izq = medio + 1;
        }
        // El elemento está en la mitad izquierda
        else {
            cout << "Decisión: " << v[medio] << " > " << nbus << endl;
            cout << "El número está en la mitad izquierda" << endl;
            cout << "Ajustando: der = medio - 1 = " << medio << " - 1 = " << (medio - 1) <<
endl;
            der = medio - 1;
        }

        iteracion++;
    }

    cout << "\n¡Elemento no encontrado en el arreglo!" << endl;
    return -1;
}

int main() {
    int n, nbus;

    cout << "=== BÚSQUEDA BINARIA ===" << endl;
    cout << "Ingrese el tamaño del arreglo: ";
    cin >> n;

    int v[n];

    cout << "\nIngrese " << n << " números ORDENADOS de forma ASCENDENTE:" <<
endl;
    for (int i = 0; i < n; i++) {
        cout << "v[" << i << "] = ";
        cin >> v[i];
```


INGENIERÍA EN TECNOLOGÍAS DE LA INFORMACIÓN

```
    }

    cout << "\nArreglo ingresado: { ";
    for (int i = 0; i < n; i++) {
        cout << v[i];
        if (i < n - 1) cout << ", ";
    }
    cout << " }" << endl;

    cout << "\nIngrese el número a buscar: ";
    cin >> nbus;

    int resultado = busquedaBinaria(v, n, nbus);

    cout << "\n=== RESULTADO FINAL ===" << endl;
    if (resultado != -1) {
        cout << "El número " << nbus << " se encuentra en la posición: " << resultado <<
endl;
    } else {
        cout << "El número " << nbus << " NO se encuentra en el arreglo." << endl;
    }

    return 0;
}
```

Datos iniciales:

- N = 11
- Arreglo = {1, 2, 4, 5, 7, 9, 11, 16, 21, 25, 34}
- Posiciones: 0 1 2 3 4 5 6 7 8 9 10
- Número a buscar = 7

Nota: Este ejemplo usa índices desde 0 (como en el primer código que te mostré)

ITERACIÓN 1:
Límites actuales:

- izq = 0
- der = 10

Cálculo del medio:
medio = (izq + der) /
2 medio = (0 + 10) / 2
medio = 10 / 2

medio = 5
Comparación:
- v[5] = 9
- nbus = 7

Decisión:
- ¿9 == 7? → NO
- ¿9 > 7? → SÍ
- El valor 7 está en la mitad IZQUIERDA
- Nuevo rango: izq = 0, der = 4

ITERACIÓN 2:
Límites actuales:
- izq = 0
- der = 4

Cálculo del
medio:
medio = (izq + der) /
2 medio = (0 + 4) / 2
medio = 4 / 2
medio = 2
Comparación:
- v[2] = 4
- nbus = 7

Decisión:
- ¿4 == 7? → NO
- ¿4 < 7? → SÍ
- El valor 7 está en la mitad DERECHA
- Nuevo rango: izq = 3, der = 4

ITERACIÓN 3:
Límites actuales:
- izq = 3
- der = 4 Cálculo

del medio: medio =
(izq + der) / 2 medio
= (3 + 4) / 2
medio = 7 / 2
medio = 3 (división entera)
Comparación:
- v[3] = 5
- nbus = 7

Decisión:


INGENIERÍA EN TECNOLOGÍAS DE LA INFORMACIÓN

- ¿5 == 7? → NO
- ¿5 < 7? → SÍ
- El valor 7 está en la mitad DERECHA
- Nuevo rango: izq = 4, der = 4

ITERACIÓN 4:
Límites actuales:

- izq = 4
- der = 4

Cálculo del
medio:
medio = (izq + der) /
2 medio = (4 + 4) / 2
medio = 8 / 2
medio = 4
Comparación:

- v[4] = 7
- nbus = 7

Decisión:

- ¿7 == 7? → ¡SÍ!
- ¡ENCONTRADO!

Resultado final:
El número 7 se encuentra en la posición 4

Resumen visual del proceso:
Iteración 1: [1, 2, 4, 5, 7, | 9, 11, 16, 21, 25, 34]
izq=0 der=10, medio=5 → 9>7, ir a izquierda

Iteración 2: [1, 2, | 4, 5, 7]
izq=0 der=4, medio=2 → 4<7, ir a derecha

Iteración 3: [5, | 7]
izq=3 der=4, medio=3 → 5<7, ir a derecha

Iteración 4: [7]
izq=4 der=4, medio=4 → 7==7, ¡ENCONTRADO!
El algoritmo necesitó 4 iteraciones para encontrar el número 7 en la posición 4.

INGENIERÍA EN TECNOLOGÍAS DE LA INFORMACIÓN

EJERCICIO 2 – FUNDAMENTOS DE PROGRAMACIÓN ALGORITMOS, ESTRUCTURA DE DATOS Y OBJETOS

Una clase de estudiantes ha registrado sus estaturas en un vector denominado ESTATURAS. Se sabe que los datos en este vector ya han sido ordenados de menor a mayor estatura.

Se te pide realizar las siguientes tareas algorítmicas utilizando estas estaturas:

1. Implementar el algoritmo de búsqueda binaria para encontrar rápidamente la posición (índice) de una estatura específica (E_objetivo) que se introduce por teclado.
2. Si la estatura se encuentra en el algoritmo debe escribir la posición (índice) donde se encuentra.
3. Si la estatura no se encuentra en el algoritmo debe escribir un mensaje indicando que no se encontró la estatura.

Consideraciones:

- Asume que el vector ESTATURAS ya está cargado con N valores reales y ordenado.
- El límite inferior del es 1 y el límite superior es N.
- La búsqueda binaria es apropiada porque está ordenado.

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

int busquedaBinaria(double estaturas[], int N, double E_objetivo)
    { int inferior = 1;
    int superior = N;
    cout << "\nBuscando estatura: " << E_objetivo << endl;

    while (inferior <= superior) {
```

```
        int medio = (inferior + superior) / 2;

        cout << "\nIteracion:" << endl;
        cout << "inferior = " << inferior << ", superior = " << superior << endl;
        cout << "medio = (" << inferior << " + " << superior << ") / 2 = " << medio <<
endl;
        cout << "ESTATURAS[" << medio << "] = " << estaturas[medio] << endl;

        if (estaturas[medio] == E_objetivo) {
            cout << "\nEstatura encontrada en posicion: " << medio << endl;
            return medio;
        }
        else if (estaturas[medio] < E_objetivo) {
            cout << estaturas[medio] << " < " << E_objetivo << " -> Buscar en mitad
superior" << endl;
            inferior = medio + 1;
        }
        else {
            cout << estaturas[medio] << " > " << E_objetivo << " -> Buscar en mitad
inferior" << endl;
            superior = medio - 1;
        }
    }

    cout << "\nEstatura NO encontrada" << endl;
    return -1;
}

int main() {
    int N;
    double E_objetivo;

    cout << "Ingrese cantidad de estudiantes: ";
    cin >> N;

    double estaturas[N + 1];

    cout << "\nIngrese estaturas ordenadas de menor a
    mayor:\n"; for (int i = 1; i <= N; i++) {
        cout << "ESTATURAS[" << i << "] = ";
        cin >> estaturas[i];
```


INGENIERÍA EN TECNOLOGÍAS DE LA INFORMACIÓN

```
    }

    cout << "\nIngrese estatura a buscar: ";
    cin >> E_objetivo;
    int resultado = busquedaBinaria(estaturas, N, E_objetivo);

    cout << "\n--- RESULTADO ---" << endl;

    if (resultado != -1) {
        cout << "Posicion: " << resultado << endl;
    } else {
        cout << "Estatura no encontrada en el vector" << endl;
    }

    return 0;
}
```

Datos iniciales:

- N = 7
- ESTATURAS = {1.55, 1.62, 1.68, 1.72, 1.78, 1.83, 1.90}
- Posiciones: 1 2 3 4 5 6 7
- E_objetivo = 1.72

ITERACIÓN 1:
Límites actuales:

- inferior = 1
- superior = 7

Cálculo del medio:
medio = (inferior + superior) / 2
medio = (1 + 7) / 2
medio = 8 / 2
medio = 4
Comparación:

- ESTATURAS[4] = 1.72
- E_objetivo = 1.72

Decisión:

- ¿1.72 == 1.72? → SÍ
- ¡ENCONTRADO!

Resultado: La estatura 1.72 está en la posición 4

INGENIERÍA EN TECNOLOGÍAS DE LA INFORMACIÓN

EJERCICIO 3 – METODOLOGIA DE LA PROGRAMACION ALGORITMOS, DIAGRAMAS DE FLUJO Y PROGRAMAS

La búsqueda binaria consiste en dividir el intervalo de búsqueda en dos partes y comparar el elemento buscado con el elemento central del arreglo. En caso de ser diferentes, se deben redefinir los extremos del intervalo considerando si el elemento buscado es mayor o menor que el elemento que se encuentra en la posición central. De esta forma, el proceso de búsqueda se repite hasta que el elemento es encontrado o bien cuando el espacio de búsqueda se anula. Esto último ocurre cuando el elemento buscado no se encuentra en el arreglo. Es importante destacar que el método funciona únicamente con arreglos ordenados. En cada paso del método el espacio de búsqueda se reduce a la mitad, por lo que el número de comparaciones a realizar disminuye considerablemente. Esta disminución en el número de comparaciones será más notoria cuando más grande sea el tamaño del arreglo.

Datos: [1..N], X $1 \leq N \leq 50$Donde:

VECTOR: es un arreglo unidimensional de tipo entero.

X: es una variable de tipo entero que representa el dato que se va a buscar.

Explicación de las variables

N: Variable de tipo entero. Indica el total de componentes que tendrá el arreglo unidimensional.

I: Variable de tipo entero. Se utiliza como índice del arreglo y como variable de control de diferentes ciclos.

VECTOR: Arreglo unidimensional de tipo entero.

X: Variable de tipo entero. Representa el dato que se va a buscar.

IZQ: Variable de tipo entero. Se utiliza para almacenar el extremo izquierdo del intervalo. DER: Variable de tipo entero. Se utiliza para almacenar el extremo derecho del intervalo. CEN: Variable de tipo entero. Se utiliza para almacenar el centro del intervalo.

BAN: Variable de tipo entero. Se utiliza para interrumpir el ciclo si se localiza el elemento buscado.

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

int main() {
   int N, X;
   int IZQ, DER, CEN;
   int BAN = 0;
   int iteracion = 1;

   cout << "=== BUSQUEDA BINARIA ===" << endl;
   cout << "Ingrese la cantidad de elementos (N): ";
   cin >> N;

   int VECTOR[N];

   cout << "\nIngrese " << N << " numeros ORDENADOS de menor a mayor:\n";
   for (int i = 0; i < N; i++) {
      cout << "VECTOR[" << i << "] = ";
      cin >> VECTOR[i];
   }

   cout << "\nVECTOR ingresado: { ";
   for (int i = 0; i < N; i++) {
      cout << VECTOR[i];
      if (i < N - 1) cout << ", ";
   }
   cout << " }" << endl;
```


INGENIERÍA EN TECNOLOGÍAS DE LA INFORMACIÓN

```
    cout << "\nIngrese el numero a buscar (X): ";
    cin >> X;

    // Inicializar límites
    IZQ = 0;
    DER = N - 1;

    cout << "\n--- PROCESO DE BUSQUEDA ---" << endl;
    cout << "Buscando X = " << X << endl;
    cout << "Limites iniciales: IZQ = " << IZQ << ", DER = " << DER << endl;

    // Búsqueda binaria
    while (IZQ <= DER && BAN == 0) {
        cout << "\n** Iteracion " << iteracion << " **" << endl;

        // Calcular centro
        CEN = (IZQ + DER) / 2;

        cout << "CEN = (IZQ + DER) / 2" << endl;
        cout << "CEN = (" << IZQ << " + " << DER << ") / 2" << endl;
        cout << "CEN = " << (IZQ + DER) << " / 2" << endl;
        cout << "CEN = " << CEN << endl;

        cout << "VECTOR[" << CEN << "] = " << VECTOR[CEN] << endl;

        // Comparar
        if (VECTOR[CEN] == X) {
            cout << "Comparacion: " << VECTOR[CEN] << " == " << X << " ->
VERDADERO" << endl;
            cout << "Elemento encontrado!" << endl;
            BAN = 1;
        }
        else if (X < VECTOR[CEN]) {
            cout << "Comparacion: " << X << " < " << VECTOR[CEN] << " ->
VERDADERO" << endl;
            cout << "Buscar en mitad IZQUIERDA" << endl;
            cout << "DER = CEN - 1 = " << CEN << " - 1 = " << (CEN - 1) << endl;
            DER = CEN - 1;
        }
        else {
```

```
            cout << "Comparacion: " << X << " > " << VECTOR[CEN] << " ->
VERDADERO" << endl;
            cout << "Buscar en mitad DERECHA" << endl;
            cout << "IZQ = CEN + 1 = " << CEN << " + 1 = " << (CEN + 1) << endl;
            IZQ = CEN + 1;
        }

        if (BAN == 0) {
            cout << "Nuevos limites: IZQ = " << IZQ << ", DER = " << DER << endl;
        }

        iteracion++;
    }

    // Resultado
    cout << "\n=== RESULTADO ===" << endl;
    if (BAN == 1) {
        cout << "El numero " << X << " se encuentra en la posicion: " << CEN << endl;
        cout << "BAN = " << BAN << " (elemento encontrado)" << endl;
    }
    else {
        cout << "El numero " << X << " NO se encuentra en el vector." << endl;
        cout << "BAN = " << BAN << " (elemento no encontrado)" << endl;
        cout << "Condicion de salida: IZQ > DER (" << IZQ << " > " << DER << ")" <<
endl;
    }

    return 0;
}
```

Buscar X = 2

Datos:

- N = 11
- N = {1, 2, 4, 5, 7, 9, 11, 16, 21, 25, 34}
- Posiciones: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10
- X = 2

ITERACIÓN 1:
Límites: IZQ = 0, DER = 10
Cálculo del centro:


INGENIERÍA EN TECNOLOGÍAS DE LA INFORMACIÓN

CEN = (IZQ + DER) / 2
CEN = (0 + 10) / 2 = 5
Comparación:

- 5 = 9
- X = 2
- ¿2 < 9? → SÍ → Buscar en mitad IZQUIERDA

Ajuste: DER = 5 - 1 = 4
Nuevos límites: IZQ = 0, DER = 4

ITERACIÓN 2:
Límites: IZQ = 0, DER = 4
Cálculo del centro:
CEN = (0 + 4) / 2 = 2
Comparación:

- 2 = 4
- X = 2
- ¿2 < 4? → SÍ → Buscar en mitad IZQUIERDA

Ajuste: DER = 2 - 1 = 1
Nuevos límites: IZQ = 0, DER = 1

ITERACIÓN 3:
Límites: IZQ = 0, DER = 1
Cálculo del centro:
CEN = (0 + 1) / 2 = 0
Comparación:

- 0 = 1
- X = 2
- ¿2 > 1? → SÍ → Buscar en mitad DERECHA

Ajuste: IZQ = 0 + 1 = 1
Nuevos límites: IZQ = 1, DER = 1

ITERACIÓN 4:
Límites: IZQ = 1, DER = 1
Cálculo del centro:
CEN = (1 + 1) / 2 = 1
Comparación:

- 1 = 2
- X = 2
- ¿2 == 2? → ¡SÍ! → BAN = 1

RESULTADO: El número 2 se encuentra en la posición 1 (4 iteraciones)


INGENIERÍA EN TECNOLOGÍAS DE LA INFORMACIÓN

Diferencias clave:

1. X = 2: Está al inicio → 4 iteraciones buscando hacia la izquierda

EJERCICIO 4 – LIBRO ALGORITMOS COMPUTACIONESLES INTRODUCCIÓN AL ANÁLISIS Y DISEÑO

Sea S un conjunto de m enteros. Sea E un arreglo de n enteros distintos ($n \leq m$). Sea K un elemento escogido al azar de S. En promedio, ¿cuántas comparaciones efectuará Búsqueda Binaria (algoritmo 1.4) con E, 0, $n - 1$, y K como entrada? Exprese su respuesta en función de n y m.

Sea:

- S: conjunto de m enteros
- E: arreglo ordenado de n enteros distintos ($n \leq m$)
- K: elemento aleatorio de S

Casos posibles:

1. K está en E: Probabilidad = n/m
    - Búsqueda exitosa
    - Comparaciones promedio = $\log_2(n) + 1$
2. K no está en E: Probabilidad = (m-n)/m
    - Búsqueda fallida
    - Comparaciones promedio = $\log_2(n) + 1$

Fórmula del promedio total:

Comparaciones promedio = $\log_2(n) + 1$

Aunque parezca contraintuitivo, tanto búsquedas exitosas como fallidas en búsqueda binaria realizan aproximadamente el mismo número de comparaciones: $\lfloor \log_2(n) \rfloor + 1$

Por lo tanto, el número promedio de comparaciones es:

$C(n,m) = \lfloor \log_2(n) \rfloor + 1$

Este resultado es independiente de m, ya que la búsqueda binaria siempre recorre la misma altura del árbol binario de búsqueda, sin importar si el elemento se encuentra o no.

```
#include <iostream>
#include <cmath>
#include <cstdlib>
#include <ctime>
using namespace std;

// Búsqueda binaria que cuenta comparaciones
int busquedaBinaria(int E[], int inicio, int fin, int K, int &comparaciones) {
    comparaciones = 0;

    while (inicio <= fin) {
        comparaciones++;
        int medio = (inicio + fin) / 2;

        if (E[medio] == K) {
            return medio; // Encontrado
        }

        if (E[medio] < K) {
            inicio = medio + 1;
        } else {
            fin = medio - 1;
        }
    }

    return -1; // No encontrado
```

```
}

int main() {
    srand(time(0));

    // Parámetros
    int n, m;
    cout << "Ingrese n (tamaño del arreglo E): ";
    cin >> n;
    cout << "Ingrese m (tamaño del conjunto S): ";
    cin >> m;

    if (n > m) {
        cout << "Error: n debe ser <= m" << endl;
        return 1;
    }

    // Crear arreglo E con n elementos ordenados
    int *E = new int[n];
    for (int i = 0; i < n; i++) {
        E[i] = i * 2; // Elementos pares: 0, 2, 4, 6, ...
    }

    // Crear conjunto S con m elementos
    int *S = new int[m];
    for (int i = 0; i < n; i++) {
        S[i] = E[i]; // Primeros n elementos son los de E
    }
    for (int i = n; i < m; i++) {
        S[i] = i * 2 + 1; // Resto son impares: no están en E
    }

    // Realizar experimento: 10000 búsquedas aleatorias
    int numPruebas = 10000;
    int totalComparaciones = 0;
    int comparaciones;

    for (int i = 0; i < numPruebas; i++) {
        int K = S[rand() % m]; // Elegir K al azar de S
        busquedaBinaria(E, 0, n - 1, K, comparaciones);
        totalComparaciones += comparaciones;
```

```
    }

    double promedioExperimental = (double)totalComparaciones / numPruebas;
    double promedioTeorico = floor(log2(n)) + 1;

    // Resultados
    cout << "\n=== RESULTADOS ===" << endl;
    cout << "n = " << n << ", m = " << m << endl;
    cout << "\nPromedio experimental de comparaciones: " << promedioExperimental <<
endl;
    cout << "Promedio teórico (⌊log₂(n)⌋ + 1): " << promedioTeorico << endl;
    cout << "\nFórmula: C(n,m) = ⌊log₂(" << n << ")⌋ + 1 = " << promedioTeorico << endl;

    // Liberar memoria
    delete[] E;
    delete[] S;

    return 0;
}
```

Generación de datos:
Arreglo E: primeros n números pares ordenados
Conjunto S: contiene los n elementos de E más (m-n) elementos adicionales
Experimento: Realiza 10,000 búsquedas con elementos K elegidos aleatoriamente de S
Resultados: Compara el promedio experimental con el teórico
Ejemplo de ejecución:
Para n=100, m=1000:
Promedio teórico = $\lfloor \log_2(100) \rfloor + 1 = 6 + 1 = 7$ comparaciones
El promedio experimental será muy cercano a 7
Nota importante: El resultado es prácticamente independiente de m, solo depende de n.

EJERCICIO 5 HASH:

Detalle los procedimientos de direccionamiento abierto para buscar, insertar y borrar claves en una tabla de dispersión. ¿En qué difieren las condiciones en las que los ciclos terminan en cada uno de estos procedimientos? Tome en cuenta la posibilidad de que haya celdas "obsoletas".

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

const int TAM = 10;
const int VACIO = -1;
const int BORRADO = -2;

// Variables globales
simples int tabla[TAM];

// Función de dispersión (hash)
int funcionHash(int clave) {
    return clave % TAM;
}

// Inicializar tabla
void inicializar() {
    for(int i = 0; i < TAM; i++) {
        tabla[i] = VACIO;
    }
}

// Mostrar tabla
void mostrar() {
    cout << "\n=== TABLA DE DISPERSION ===\n";
    for(int i = 0; i < TAM; i++) {
        cout << "Posicion [" << i << "]: ";
        if(tabla[i] == VACIO)
            cout << "VACIO";
        else if(tabla[i] == BORRADO)
            cout << "BORRADO";
        else
            cout << tabla[i];
        cout << endl;
    }
    cout << "============================\n";
}

// INSERTAR con sondeo lineal
bool insertar(int clave) {
    int pos = funcionHash(clave);
```


INGENIERÍA EN TECNOLOGÍAS DE LA INFORMACIÓN

```
    int posInicial = pos;
    int intentos = 0;

    cout << "\n--- INSERTANDO " << clave << " ---\n";
    cout << "Hash inicial: " << pos << endl;

    // Buscar posición vacía o borrada
    while(intentos < TAM) {
        cout << "Intento " << (intentos + 1) << ": Posicion " << pos;

        if(tabla[pos] == VACIO || tabla[pos] == BORRADO) {
            tabla[pos] = clave;
            cout << " -> INSERTADO\n";
            return true;
        }

        if(tabla[pos] == clave) {
            cout << " -> YA EXISTE\n";
            return false;
        }

        cout << " -> OCUPADA (valor: " << tabla[pos] << ")\n";
        pos = (posInicial + intentos + 1) % TAM;
        intentos++;
    }

    cout << "TABLA LLENA - NO SE PUDO INSERTAR\n";
    return false;
}

// BUSCAR con sondeo lineal
int buscar(int clave) {
    int pos = funcionHash(clave);
    int posInicial = pos;
    int intentos = 0;

    cout << "\n--- BUSCANDO " << clave << " ---\n";
    cout << "Hash inicial: " << pos << endl;

    while(intentos < TAM) {
        cout << "Intento " << (intentos + 1) << ": Posicion " << pos;
```


INGENIERÍA EN TECNOLOGÍAS DE LA INFORMACIÓN

```
        // Si encontramos la clave
        if(tabla[pos] == clave) {
            cout << " -> ENCONTRADO\n";
            return pos;
        }

        // Si encontramos celda vacía, la clave no existe
        if(tabla[pos] == VACIO) {
            cout << " -> VACIO (clave no existe)\n";
            return -1;
        }

        // Si está borrada o tiene otra clave, seguir buscando
        cout << " -> ";
        if(tabla[pos] == BORRADO)
            cout << "BORRADO";
        else
            cout << "Otro valor (" << tabla[pos] << ")";
        cout << " (continuar)\n";

        pos = (posInicial + intentos + 1) % TAM;
        intentos++;
    }

    cout << "NO ENCONTRADO (tabla recorrida completa)\n";
    return -1;
}

// BORRAR con sondeo lineal
bool borrar(int clave) {
    int pos = funcionHash(clave);
    int posInicial = pos;
    int intentos = 0;

    cout << "\n--- BORRANDO " << clave << " ---\n";
    cout << "Hash inicial: " << pos << endl;

    while(intentos < TAM) {
        cout << "Intento " << (intentos + 1) << ": Posicion " << pos;
```

```
        // Si encontramos la clave
        if(tabla[pos] == clave) {
            tabla[pos] = BORRADO;
            cout << " -> BORRADO\n";
            return true;
        }

        // Si encontramos celda vacía, la clave no existe
        if(tabla[pos] == VACIO) {
            cout << " -> VACIO (clave no existe)\n";
            return false;
        }

        // Si está borrada o tiene otra clave, seguir buscando
        cout << " -> ";
        if(tabla[pos] == BORRADO)
            cout << "BORRADO";
        else
            cout << "Otro valor (" << tabla[pos] << ")";
        cout << " (continuar)\n";

        pos = (posInicial + intentos + 1) % TAM;
        intentos++;
    }

    cout << "NO ENCONTRADO (tabla recorrida completa)\n";
    return false;
}

int main() {
    inicializar();
    int opcion, clave;

    do {
        cout << "\n========== MENU ==========\n";
        cout << "1. Insertar clave\n";
        cout << "2. Buscar clave\n";
        cout << "3. Borrar clave\n";
        cout << "4. Mostrar tabla\n";
        cout << "0. Salir\n";
        cout << "Opcion: ";
```


INGENIERÍA EN TECNOLOGÍAS DE LA INFORMACIÓN

```
        cin >> opcion;

        switch(opcion) {
            case 1:
                cout << "Ingrese clave a insertar: ";
                cin >> clave;
                insertar(clave);
                mostrar();
                break;
            case 2:
                cout << "Ingrese clave a buscar: ";
                cin >> clave;
                buscar(clave);
                break;
            case 3:
                cout << "Ingrese clave a borrar: ";
                cin >> clave;
                borrar(clave);
                mostrar();
                break;
            case 4:
                mostrar();
                break;
            case 0:
                cout << "Saliendo...\n";
                break;
            default:
                cout << "Opcion invalida\n";
        }
    } while(opcion != 0);

    return 0;
}
```

Fórmula: posición = número ÷ 10, tomar el RESIDUO
INSERTAR 25
Cálculo:
25 ÷ 10 = 2 y sobran 5
Posición = 5
Resultado:


INGENIERÍA EN TECNOLOGÍAS DE LA INFORMACIÓN

Tabla antes: [_] [_] [_] [_] [_] [_] [_] [_] [_] [_]
Tabla después: [_] [_] [_] [_] [_] [25] [_] [_] [_] [_]
↑
posición
5

BUSCAR 25
Cálculo:
25 ÷ 10 = sobra 5
Empezar en posición 5
Proceso:
Revisar posición 5: tabla[5] = 25 ✓ ¡ENCONTRADO!

BORRAR 25
Cálculo:
25 ÷ 10 = sobra 5
Empezar en posición 5
Proceso:
Revisar posición 5: tabla[5] = 25 → Marcar como BORRADO
Resultado:
Tabla después: [_] [_] [_] [_] [_] [BORRADO] [_] [_] [_] [_]
↑
posición
5

Ahora BUSCAR 25 otra vez (después de borrado)
Cálculo:
25 ÷ 10 = sobra 5
Empezar en posición 5
Proceso:
Revisar posición 5: tabla[5] = BORRADO (no es 25)
Revisar posición 6: tabla[6] = VACÍO → PARAR, NO EXISTE ✗

RESUMEN DEL CÁLCULO
Solo necesitas hacer:
número ÷ 10 = tomar el RESIDUO
Ejemplo: 25 ÷ 10 = residuo 5 → va en posición 5

EJERCICIO 6 HASH:

El tipo de tabla de dispersión H bajo direccionamiento cerrado es un arreglo de referencias a listas, y bajo direccionamiento abierto es un arreglo de claves. Suponga que una clave

requiere una "palabra" de memoria y un nodo de lista ligada requiere dos palabras, una para la clave y una para una referencia a una lista. Considere cada uno de estos factores de carga con direccionamiento cerrado: 0.25, 0.5, 1.0, 2.0. Sea hc el número de celdas de la tabla de dispersión con direccionamiento cerrado.

- a. Estime el espacio total requerido, incluido espacio para listas, con direccionamiento cerrado. Luego, suponiendo que se usa la misma cantidad de espacio para una tabla de dispersión con direccionamiento abierto, determine los factores de carga correspondientes si se usa direccionamiento abierto.
- b. Ahora suponga que una clave ocupa cuatro palabras y que un nodo de lista ocupa cinco palabras (cuatro para la clave y una para la referencia al resto de la lista), y repita la parte (a).

FÓRMULAS SIMPLES:

- $n = \alpha \times hc$
- CERRADO = hc + (n × palabras_por_nodo)
- ABIERTO_capacidad = CERRADO / palabras_por_clave
- α_abierto = n / ABIERTO_capacidad

CÓDIGO

```
#include <iostream>
#include <limits>
#include <string>

using namespace std;

// --- FUNCION DE VALIDACION ---
// Evita que el programa se rompa si se ingresan letras o simbolos
template <typename T>
T leerDato(string mensaje) {
    T valor;
    while (true) {
```

```
        cout << mensaje;
        if (cin >> valor) {
            return valor;
        } else {
            cout << " [!] Error: Entrada no valida. Intente de nuevo.\n";
            cin.clear(); // Limpia el error
            cin.ignore(numeric_limits<streamsize>::max(), '\n'); // Descarta el buffer
        }
    }
}

void mostrarCabecera() {
    cout << "================================================" << endl;
    cout << "       CALCULADORA DE ESPACIO (HASH)        " << endl;
    cout << "================================================" << endl;
}

int main() {
    int hc;          // Tamaño de la tabla
    float factorCarga;
    int tipoClave;

    mostrarCabecera();

    // Entrada de datos con validacion
    hc = leerDato<int>(" > Tamanio de la tabla (hc): ");
    factorCarga = leerDato<float>(" > Factor de carga (0.25, 0.5, 1.0, 2.0): ");

    cout << "\n--- TIPO DE CLAVE ---" << endl;
    cout << " 1 = Clave de 1 palabra (nodo = 2 palabras)" << endl;
    cout << " 2 = Clave de 4 palabras (nodo = 5 palabras)" << endl;
    tipoClave = leerDato<int>(" > Seleccione opcion (1 o 2): ");

    // --- CALCULO LOGICO ---
    // n = factor_carga * hc
    int n = (int)(factorCarga * hc);

    cout << "\n" << string(44, '=') << endl;
    cout << "          RESULTADOS DEL ANALISIS           " << endl;
    cout << string(44, '=') << endl;
    cout << " Numero de claves (n): " << n << endl;

    // --- DIRECCIONAMIENTO CERRADO ---
    cout << "\n[1] DIRECCIONAMIENTO CERRADO" << endl;
    cout << "----------------------------------------" << endl;

    int espacioTabla = hc; // 1 palabra por entrada de tabla (puntero)
    int espacioListas;
```

```
    int palabrasPorNodo = (tipoClave == 1) ? 2 : 5;

    espacioListas = n * palabrasPorNodo;
    int totalCerrado = espacioTabla + espacioListas;

    cout << " + Espacio tabla (punteros): " << hc << " palabras" << endl;
    cout << " + Espacio nodos: " << n << " x " << palabrasPorNodo << " = " <<
espacioListas << " palabras" << endl;
    cout << " >> TOTAL CERRADO: " << totalCerrado << " palabras" << endl;

    // --- DIRECCIONAMIENTO ABIERTO ---
    cout << "\n[2] DIRECCIONAMIENTO ABIERTO" << endl;
    cout << "-----------------------------------------" << endl;
    cout << " * Usando el mismo espacio total: " << totalCerrado << " palabras" << endl;

    int palabrasPorClave = (tipoClave == 1) ? 1 : 4;
    int capacidadAbierto = totalCerrado / palabrasPorClave;
    float factorCargaAbierto = (float)n / capacidadAbierto;

    cout << " + Capacidad max: " << totalCerrado << " / " << palabrasPorClave << " = " <<
capacidadAbierto << " claves" << endl;
    cout << " >> NUEVO FACTOR DE CARGA: " << factorCargaAbierto << endl;

    // --- RESUMEN FINAL ---
    cout << "\n" << string(44, '-') << endl;
    cout << " RESUMEN: Para " << n << " claves..." << endl;
    cout << " En Cerrado (alpha=" << factorCarga << ") ocupas " << totalCerrado << "
palabras." << endl;
    cout << " En Abierto con ese espacio, el alpha baja a: " << factorCargaAbierto << endl;
    cout << string(44, '=') << endl;

    return 0;
}
```

DATOS:

- Tabla de tamaño: hc = 10
- Factor de carga: α = 1.0
- Tipo: Clave de 1 palabra

PASO 1: ¿Cuántas claves tengo? $n = \alpha \times hc$

$n = 1.0 \times 10$

$n = 10$ claves

PASO 2: DIRECCIONAMIENTO CERRADO (con listas)

¿Cuánto espacio uso?

Espacio tabla = 10 palabras (solo referencias)
Espacio listas = 10 claves × 2 palabras/nodo = 20 palabras

TOTAL = 10 + 20 = 30
palabras Dibujo:
TABLA (10 palabras) LISTAS (20
palabras) [ref] ----------> [clave|ref]
[clave|ref]
[ref] ----------> [clave|ref]
[ref] ----------> [clave|ref] [clave|ref] [clave|ref]
...

PASO 3: DIRECCIONAMIENTO ABIERTO con mismo espacio
Tengo 30 palabras para usar:
Cada clave usa 1 palabra
Capacidad = 30 / 1 = 30 claves

Pero solo tengo 10 claves
Factor de carga = 10 / 30 =
0.33 Dibujo:
TABLA (30 palabras, todas para claves)
[clave] [clave] [vacío] [clave] [vacío] [vacío] ...

RESULTADO:
CERRADO: Factor = 1.0 (10 claves en tabla de 10)
ABIERTO: Factor = 0.33 (10 claves en tabla de 30)

Conclusión: Con el mismo espacio, ABIERTO tiene tabla más grande y factor más bajo.

AHORA CON CLAVE DE 4
PALABRAS: DATOS:

- Tabla: hc = 10
- Factor: α = 1.0
- Tipo: Clave de 4 palabras

PASO 1: Número de
claves n = 1.0 × 10 = 10
claves

PASO 2: CERRADO
Espacio tabla = 10 palabras
Espacio listas = 10 claves × 5 palabras/nodo = 50 palabras

TOTAL = 10 + 50 = 60 palabras

Factor Cerrado	Claves	Espacio Total	Factor Abierto
0.25	3	16	0.19
0.5	5	20	0.25
1.0	10	30	0.33
2.0	20	50	0.40

Factor Cerrado	Claves	Espacio Total	Factor Abierto
0.25	3	25	0.50
0.5	5	35	0.57
1.0	10	60	0.67
2.0	20	110	0.73

EJERCICIO 7

Un vendedor de “Fuko Pop”, tiene la lista de todos los funko que tiene disponibles en su tienda

con la siguiente información: número, nombre, colección y precio; para lo cual nos solicita le desarrollemos un algoritmo que contemple los siguiente requerimientos:

a. realizar un listado ordenado por número;

b. realizar un listado de los funko pop de una colección;

c. determinar si tiene disponible el funko pop 130 de la colección de Star Wars y mostrar toda la información;

d. mostrar todos los funko pop disponible número 295;

e. listar todos los modelos de Darth Vader y Capitana Marvel;

f. determinar si existen funko pop de Red Skull, Thanos y Galactus, además mostrarla

información de estos;

g. calcular el costo de todos los modelos de Tony Stark e Iron Man disponibles;

h. calcular el promedio de costo de todos los funko pop y el costo total de las colecciones de Rocks y Harry Potter.

- Ordenar = comparar números y cambiarlos
- Buscar = si coincide, mostrarlo
- Sumar = hacer precio1 + precio2 + precio3...
- Promedio = sumar todo ÷ cantidad

```
#include <iostream>
#include <string>
#include <limits>
#include <iomanip> // Para formato de decimales en el precio

using namespace std;

// --- FUNCIONES DE VALIDACION ---
int leerEntero(string mensaje) {
    int valor;
    while (true) {
        cout << mensaje;
        if (cin >> valor) {
            cin.ignore(numeric_limits<streamsize>::max(), '\n');
            return valor;
        } else {
            cout << " [!] Error: Ingrese un numero entero.\n";
            cin.clear();
            cin.ignore(numeric_limits<streamsize>::max(), '\n');
        }
    }
}

float leerFloat(string mensaje) {
    float valor;
    while (true) {
```

```
        cout << mensaje;
        if (cin >> valor) {
            cin.ignore(numeric_limits<streamsize>::max(), '\n');
            return valor;
        } else {
            cout << " [!] Error: Ingrese un precio valido.\n";
            cin.clear();
            cin.ignore(numeric_limits<streamsize>::max(), '\n');
        }
    }
}

void dibujarCabecera(string titulo) {
    cout << "\n=====================================================" <<
endl;
    cout << "          " << titulo << endl;
    cout << "=====================================================" <<
endl;
}

int main() {
    int cantidad;
    dibujarCabecera("GESTOR DE COLECCION FUNKO POP");
    cantidad = leerEntero(" > Cuantos Funko Pop vas a ingresar? ");

    int numero[100];
    string nombre[100];
    string coleccion[100];
    float precio[100];

    // 1. INGRESO DE DATOS
    for(int i = 0; i < cantidad; i++) {
        cout << "\n[ Registro #" << (i+1) << " ]" << endl;
        numero[i] = leerEntero(" -> Numero de serie: ");
        cout << " -> Nombre del personaje: ";
        getline(cin, nombre[i]);
        cout << " -> Coleccion (ej. Star Wars, Marvel): ";
        getline(cin, coleccion[i]);
        precio[i] = leerFloat(" -> Precio: $");
    }

    // 2. ORDENAMIENTO (Bubble Sort por Numero)
    for(int i = 0; i < cantidad - 1; i++) {
        for(int j = 0; j < cantidad - i - 1; j++) {
            if(numero[j] > numero[j+1]) {
                // Intercambio sincronizado de los 4 arreglos
                swap(numero[j], numero[j+1]);
                swap(nombre[j], nombre[j+1]);
```

```
            swap(coleccion[j], coleccion[j+1]);
            swap(precio[j], precio[j+1]);
        }
    }
}

// 3. REPORTES
dibujarCabecera("A) INVENTARIO ORDENADO POR SERIE");
cout << "No.\t| NOMBRE\t\t| COLECCION\t| PRECIO" << endl;
cout << "-------------------------------------------------" << endl;
for(int i = 0; i < cantidad; i++) {
    cout << " #" << numero[i] << "\t| " << nombre[i] << (nombre[i].length() < 8 ? "\t\t| " :
"\t| ")
         << coleccion[i] << "\t| $" << fixed << setprecision(2) << precio[i] << endl;
}

dibujarCabecera("B) BUSQUEDA POR COLECCION");
string buscar;
cout << " > Ingrese coleccion a filtrar: ";
getline(cin, buscar);
for(int i = 0; i < cantidad; i++) {
    if(coleccion[i] == buscar) {
        cout << " [+] " << nombre[i] << " (#" << numero[i] << ") - $" << precio[i] << endl;
    }
}

dibujarCabecera("C al G) FILTROS ESPECIFICOS");
float sumaIronMan = 0;
for(int i = 0; i < cantidad; i++) {
    // C) Star Wars 130
    if(numero[i] == 130 && coleccion[i] == "Star Wars")
        cout << " [!] EXISTE: " << nombre[i] << " de Star Wars (#130)" << endl;

    // D) Numero 295
    if(numero[i] == 295)
        cout << " [!] NUMERO 295: " << nombre[i] << " (" << coleccion[i] << ")" << endl;

    // E/F) Personajes Iconicos
    if(nombre[i] == "Darth Vader" || nombre[i] == "Capitana Marvel" ||
       nombre[i] == "Red Skull" || nombre[i] == "Thanos" || nombre[i] == "Galactus") {
        cout << " [!] ICONO ENCONTRADO: " << nombre[i] << " (#" << numero[i] <<
")" << endl;
    }

    // G) Suma Tony Stark/Iron Man
    if(nombre[i] == "Tony Stark" || nombre[i] == "Iron Man") {
        sumaIronMan += precio[i];
    }
```

```
    }
    cout << " >> Costo total Tony Stark + Iron Man: $" << sumaIronMan << endl;

    dibujarCabecera("H) ANALISIS DE COSTOS");
    float todosSuma = 0, rocksTotal = 0, hpTotal = 0;
    for(int i = 0; i < cantidad; i++) {
        todosSuma += precio[i];
        if(coleccion[i] == "Rocks") rocksTotal += precio[i];
        if(coleccion[i] == "Harry Potter") hpTotal += precio[i];
    }
    cout << " -> Promedio general de la coleccion: $" << (cantidad > 0 ?
todosSuma/cantidad : 0) << endl;
    cout << " -> Inversion total en 'Rocks': $" << rocksTotal << endl;
    cout << " -> Inversion total en 'Harry Potter': $" << hpTotal << endl;

    return 0;
}
```

a) ORDENAR POR NÚMERO

Simplemente comparo números:

- Si 50 > 30 → los cambio de posición
- Así quedan de menor a mayor

Ejemplo:

Antes: 50, 30, 100

Después: 30, 50, 100

b) BUSCAR POR COLECCIÓN

Veo si la colección coincide:

- Si dice "Marvel" → lo muestro
- Si no dice "Marvel" → lo ignoro

c) BUSCAR #130 DE STAR WARS

Dos condiciones:

1. ¿El número es 130?
2. ¿La colección es "Star Wars"? Si AMBAS son SÍ → lo muestro

d) BUSCAR TODOS LOS #295

Solo busco si el número es 295

**e) y f) BUSCAR POR NOMBRES

Pregunto: ¿el nombre es X o Y?

- Si SÍ → lo muestro

g) SUMAR TONY STARK E IRON MAN
CÁLCULO:
Suma = precio1 + precio2 + precio3... Ejemplo real:
Iron Man = $15
Tony Stark = $20
Iron Man = $18

TOTAL = $53 (15+20+18)

h) PROMEDIOS Y TOTALES
PROMEDIO DE TODOS:
Promedio = (suma de todos los precios) ÷ (cantidad total) Ejemplo:
Funko 1 = $10
Funko 2 = $20
Funko 3 = $30
Funko 4 = $40

Suma = 10+20+30+40 = 100
Promedio = 100 ÷ 4 = $25
TOTAL DE ROCKS:
Solo sumo los precios que dicen "Rocks"
Ejemplo:
Rocks #1 = $15
Rocks #2 = $25
Marvel = $30 (este NO se suma)

Total Rocks = 15+25 = $40
TOTAL DE HARRY POTTER:
Solo sumo los precios que dicen "Harry Potter"


INGENIERÍA EN TECNOLOGÍAS DE LA INFORMACIÓN


INGENIERÍA EN TECNOLOGÍAS DE LA INFORMACIÓN


INGENIERÍA EN TECNOLOGÍAS DE LA INFORMACIÓN


INGENIERÍA EN TECNOLOGÍAS DE LA INFORMACIÓN

EJERCICIO 8

A partir de una lista con todos los caballeros Jedi y los lores Sith, de los que conocemos su nombre y el de su maestro (considere solo uno), resuelva las siguientes actividades:

a. realizar un listado ordenado por nombre;

b. listar todos los Jedi ordenados por nombre;

c. listar todos los Sith ordenados por nombre;

d. mostrar los aprendices de Palpatine y de Obi-Wan kenobi, además contar cuantos aprendices tuvo cada uno;

e. determinar si Dath Malak está en la lista y mostrar su información;

f. mostrar la posición en la que se encuentra Yoda.

Ordenamiento: $O(n^2)$ = Si hay 10 elementos → máximo 100 operaciones

Búsqueda: $O(n)$ = Si hay 10 elementos → máximo 10 comparaciones

Conteo: $O(n)$ = Si hay 10 elementos → exactamente 10 recorridos

```
#include <iostream>
#include <string>
#include <limits>

using namespace std;

// --- FUNCION DE VALIDACION PARA NUMEROS ---
int leerEntero(string mensaje) {
    int valor;
    while (true) {
        cout << mensaje;
        if (cin >> valor && valor > 0) {
            cin.ignore(numeric_limits<streamsize>::max(), '\n');
            return valor;
        } else {
            cout << " [!] Error: Ingrese un numero entero positivo.\n";
            cin.clear();
            cin.ignore(numeric_limits<streamsize>::max(), '\n');
        }
    }
}
```

```
void dibujarSeparador() {
    cout << "--------------------------------------------------" << endl;
}

int main() {
    string nombres[100], maestros[100], tipo[100];
    int total = 0;

    cout << "==========================================" << endl;
    cout << "      SISTEMA DE ARCHIVOS GALACTICOS    " << endl;
    cout << "==========================================" << endl;

    total = leerEntero(" > Cantidad de personajes a registrar: ");

    // 1. ENTRADA DE DATOS
    for(int i = 0; i < total; i++) {
        cout << "\n[ Registro #" << (i+1) << " ]" << endl;
        cout << " -> Nombre del personaje: ";
        getline(cin, nombres[i]);
        cout << " -> Maestro asignado: ";
        getline(cin, maestros[i]);
        cout << " -> Afiliacion (Jedi/Sith): ";
        getline(cin, tipo[i]);
    }

    // 2. ORDENAMIENTO BUBBLE SORT (Por Nombre)
    // Mantiene sincronizados los 3 arreglos paralelos
    for(int i = 0; i < total - 1; i++) {
        for(int j = 0; j < total - i - 1; j++) {
            if(nombres[j] > nombres[j+1]) {
                swap(nombres[j], nombres[j+1]);
                swap(maestros[j], maestros[j+1]);
                swap(tipo[j], tipo[j+1]);
            }
        }
    }

    // 3. PRESENTACION DE REPORTES
    cout << "\n\n==========================================" << endl;
    cout << "       REPORTES DE LA ORDEN            " << endl;
    cout << "==========================================" << endl;

    // a. Lista Completa Ordenada
    cout << "\n[A] LISTADO ALFABETICO GENERAL:" << endl;
    dibujarSeparador();
    for(int i = 0; i < total; i++) {
        cout << " " << i+1 << ". " << nombres[i] << " | Maestro: " << maestros[i] << " | (" <<
tipo[i] << ")" << endl;
```

```
    }

    // b y c. Filtrado por Afiliacion
    int cJedi = 0, cSith = 0;
    cout << "\n[B] MAESTROS Y APRENDICES JEDI:" << endl;
    for(int i = 0; i < total; i++) {
        if(tipo[i] == "Jedi" || tipo[i] == "jedi") {
            cout << "  - " << nombres[i] << " (Maestro: " << maestros[i] << ")" << endl;
            cJedi++;
        }
    }
    cout << " Total Jedi: " << cJedi << endl;

    cout << "\n[C] LORES Y APRENDICES SITH:" << endl;
    for(int i = 0; i < total; i++) {
        if(tipo[i] == "Sith" || tipo[i] == "sith") {
            cout << "  - " << nombres[i] << " (Maestro: " << maestros[i] << ")" << endl;
            cSith++;
        }
    }
    cout << " Total Sith: " << cSith << endl;

    // d. Busqueda de Aprendices Especificos
    cout << "\n[D] BUSQUEDA POR MAESTRO (Palpatine/Obi-Wan):" << endl;
    int cP = 0, cO = 0;
    for(int i = 0; i < total; i++) {
        string m = maestros[i];
        if(m == "Palpatine" || m == "palpatine") cP++;
        if(m == "Obi-Wan Kenobi" || m == "Obi-Wan" || m == "obi-wan") cO++;
    }
    cout << " -> Aprendices de Palpatine: " << cP << endl;
    cout << " -> Aprendices de Obi-Wan: " << cO << endl;

    // e y f. Localizacion de Objetivos
    cout << "\n[E/F] LOCALIZACION DE INDIVIDUOS:" << endl;
    bool malak = false, yoda = false;
    for(int i = 0; i < total; i++) {
        if(nombres[i] == "Darth Malak" || nombres[i] == "darth malak") {
            cout << " [+] Darth Malak: ENCONTRADO en archivos." << endl;
            malak = true;
        }
        if(nombres[i] == "Yoda" || nombres[i] == "yoda") {
            cout << " [+] Yoda: LOCALIZADO en posicion " << i+1 << " del registro." <<
endl;
            yoda = true;
        }
    }
    if(!malak) cout << " [-] Darth Malak: No figura en los registros." << endl;
```

```
    if(!yoda) cout << " [-] Yoda: No figura en los registros." << endl;

    cout << "\n========================================" << endl;
    cout << "      FIN DEL REPORTE GALACTICO         " << endl;
    cout << "========================================" << endl;

    return 0;
}
```

a) ORDENAMIENTO POR NOMBRE (Bubble Sort)
Ejemplo: Ordenar ["Luke", "Anakin", "Yoda"]

Pasada 1:

- Comparo "Luke" con "Anakin" → Luke > Anakin → INTERCAMBIO
  ["Anakin", "Luke", "Yoda"]
- Comparo "Luke" con "Yoda" → Luke < Yoda → NO cambio
  ["Anakin", "Luke", "Yoda"]

Pasada 2:

- Comparo "Anakin" con "Luke" → Anakin < Luke → NO cambio
  ["Anakin", "Luke", "Yoda"]

RESULTADO: ["Anakin", "Luke", "Yoda"]

Fórmula de comparaciones:

- Con n elementos: (n-1) + (n-2) + ... + 1 = n(n-1)/2
- Con 3 elementos: 3(2)/2 = 3 comparaciones
- Con 5 elementos: 5(4)/2 = 10 comparaciones

---

b) y c) CONTAR JEDI Y SITH

Ejemplo: Lista = ["Luke(Jedi)", "Vader(Sith)", "Yoda(Jedi)"]

Contador Jedi = 0

Contador Sith = 0

Posición 0: "Luke" es Jedi → Jedi++ → Jedi = 1

Posición 1: "Vader" es Sith → Sith++ → Sith = 1

Posición 2: "Yoda" es Jedi → Jedi++ → Jedi = 2

RESULTADO: 2 Jedi, 1 Sith

Fórmula: Recorridos = n (donde n = total de elementos)

---

d) CONTAR APRENDICES

Ejemplo: Buscar aprendices de "Palpatine"

Lista de maestros: ["Qui-Gon", "Palpatine", "Palpatine", "Yoda"]

Contador = 0

Posición 0: "Qui-Gon" ≠ "Palpatine" → No cuenta

Posición 1: "Palpatine" = "Palpatine" → Contador++ → 1

Posición 2: "Palpatine" = "Palpatine" → Contador++ → 2

Posición 3: "Yoda" ≠ "Palpatine" → No cuenta

RESULTADO: 2 aprendices

Fórmula: Comparaciones = n (n = total de elementos)

---

e) BÚSQUEDA DE "DARTH MALAK"

Ejemplo: Lista = ["Luke", "Yoda", "Darth Malak", "Anakin"]

Posición 0: "Luke" ≠ "Darth Malak" → Continuar

Posición 1: "Yoda" ≠ "Darth Malak" → Continuar

Posición 2: "Darth Malak" = "Darth Malak" → ¡ENCONTRADO!
→ Detenerse (break)


INGENIERÍA EN TECNOLOGÍAS DE LA INFORMACIÓN


INGENIERÍA EN TECNOLOGÍAS DE LA INFORMACIÓN


INGENIERÍA EN TECNOLOGÍAS DE LA INFORMACIÓN


INGENIERÍA EN TECNOLOGÍAS DE LA INFORMACIÓN

RESULTADO: Encontrado en posición 3 (índice 2)
Fórmula:

- Mejor caso: 1 comparación (primer elemento)
- Peor caso: n comparaciones (último elemento o no existe)
- Promedio: n/2 comparaciones

f) ENCONTRAR POSICIÓN DE "YODA"
Ejemplo: Lista ordenada = ["Anakin", "Luke", "Yoda", "Obi-Wan"]
Posición 0 (índice 0): "Anakin" ≠ "Yoda"
Posición 1 (índice 1): "Luke" ≠ "Yoda"
Posición 2 (índice 2): "Yoda" = "Yoda" → ENCONTRADO
RESULTADO: Posición 3 (mostramos índice+1 al usuario)

EJEMPLO COMPLETO CON 5 PERSONAJES
Entrada:
1. Luke Skywalker - Obi-Wan - Jedi
2. Darth Vader - Palpatine - Sith
3. Yoda - Ninguno - Jedi
4. Anakin Skywalker - Obi-Wan - Jedi
5. Darth Maul - Palpatine - Sith
Proceso de ordenamiento (Bubble Sort):
Original: [Luke, Vader, Yoda, Anakin, Maul]

Paso 1: [Luke, Vader, Yoda, Anakin, Maul] Paso 2: [Luke, Vader, Anakin, Yoda, Maul] Paso 3: [Luke, Anakin, Vader, Yoda, Maul] Paso 4: [Anakin, Luke, Vader, Yoda, Maul]
Final: [Anakin, Darth Maul, Darth Vader, Luke, Yoda] Conteo:

- Jedi: 3 (Anakin, Luke, Yoda)
- Sith: 2 (Maul, Vader)
- Aprendices de Palpatine: 2 (Vader, Maul)
- Aprendices de Obi-Wan: 2 (Luke, Anakin)

EJERCICIO 9 – HASH

Desarrollar un algoritmo que implemente una tabla hash para una guía de teléfono, los datos que se conocen son número de teléfono, apellido, nombre y dirección de la persona. El campo

clave debe ser el número de teléfono.

```
#include <iostream>
#include <string>
#include <limits>

using namespace std;

const int TAMANO = 10;

// Estructuras de datos (Arrays paralelos)
int telefonos[TAMANO];
string nombres[TAMANO];
string apellidos[TAMANO];
string direcciones[TAMANO];
bool ocupado[TAMANO];

// --- VALIDACION DE ENTRADA ---
int leerEntero(string mensaje) {
    int valor;
    while (true) {
        cout << mensaje;
        if (cin >> valor) {
            cin.ignore(numeric_limits<streamsize>::max(), '\n');
            return valor;
        } else {
            cout << " [!] Error: Ingrese un numero valido.\n";
            cin.clear();
            cin.ignore(numeric_limits<streamsize>::max(), '\n');
        }
    }
}

// Inicializar tabla
void inicializar() {
    for (int i = 0; i < TAMANO; i++) {
        ocupado[i] = false;
    }
}

int funcionHash(int telefono) {
    return telefono % TAMANO;
}

void insertar() {
    cout << "\n--- NUEVO CONTACTO ---" << endl;
    int tel = leerEntero(" > Telefono: ");
```

```
    string nom, ape, dir;

    cout << " > Nombre: "; getline(cin, nom);
    cout << " > Apellido: "; getline(cin, ape);
    cout << " > Direccion: "; getline(cin, dir);

    int indice = funcionHash(tel);
    int intentos = 0;

    // Sondeo lineal para encontrar espacio o actualizar
    while (ocupado[indice] && telefonos[indice] != tel && intentos < TAMANO) {
        indice = (indice + 1) % TAMANO;
        intentos++;
    }

    if (intentos == TAMANO) {
        cout << "\n [!] ERROR: La memoria de la guia esta llena." << endl;
        return;
    }

    telefonos[indice] = tel;
    nombres[indice] = nom;
    apellidos[indice] = ape;
    direcciones[indice] = dir;
    ocupado[indice] = true;

    cout << " [+] Guardado exitosamente en indice: " << indice << endl;
}

void buscar() {
    cout << "\n--- BUSQUEDA DE CONTACTO ---" << endl;
    int tel = leerEntero(" > Ingrese numero a buscar: ");

    int indice = funcionHash(tel);
    int intentos = 0;

    while (intentos < TAMANO) {
        if (ocupado[indice] && telefonos[indice] == tel) {
            cout << "\n========================================" << endl;
            cout << " CONTACTO ENCONTRADO EN INDICE " << indice << endl;
            cout << "========================================" << endl;
            cout << " Nombre:    " << nombres[indice] << " " << apellidos[indice] << endl;
            cout << " Telefono:  " << telefonos[indice] << endl;
            cout << " Direccion: " << direcciones[indice] << endl;
            return;
        }
        indice = (indice + 1) % TAMANO;
        intentos++;
```

```
    }
    cout << " [!] El contacto no existe en la guia." << endl;
}

void mostrarTodo() {
    cout << "\n========== VISTA DE LA TABLA HASH ==========" << endl;
    cout << "IND\t| ESTADO\t| DATOS" << endl;
    cout << "-----------------------------------------" << endl;
    for (int i = 0; i < TAMANO; i++) {
        cout << " [" << i << "]\t| ";
        if (ocupado[i]) {
            cout << "OCUPADO\t| " << telefonos[i] << " - " << nombres[i] << endl;
        } else {
            cout << "VACIO\t| ----" << endl;
        }
    }
}

void eliminar() {
    cout << "\n--- ELIMINAR REGISTRO ---" << endl;
    int tel = leerEntero(" > Numero a borrar: ");

    int indice = funcionHash(tel);
    int intentos = 0;

    while (intentos < TAMANO) {
        if (ocupado[indice] && telefonos[indice] == tel) {
            ocupado[indice] = false;
            cout << " [-] Contacto eliminado de la posicion " << indice << endl;
            return;
        }
        indice = (indice + 1) % TAMANO;
        intentos++;
    }
    cout << " [!] No se encontro el numero para eliminar." << endl;
}

int main() {
    inicializar();
    int opcion;

    do {
        cout << "\n========================================" << endl;
        cout << "       SISTEMA HASH: TELEFONOS        " << endl;
        cout << "========================================" << endl;
        cout << " 1. [ + ] Insertar Contacto" << endl;
        cout << " 2. [ ? ] Buscar por Telefono" << endl;
        cout << " 3. [ - ] Eliminar Contacto" << endl;
```

```
        cout << " 4. [ L ] Listar Tabla Hash" << endl;
        cout << " 0. [ X ] Salir" << endl;
        cout << "---------------------------------------" << endl;
        opcion = leerEntero(" >> Seleccione una opcion: ");

        switch(opcion) {
            case 1: insertar(); break;
            case 2: buscar(); break;
            case 3: eliminar(); break;
            case 4: mostrarTodo(); break;
            case 0: cout << "Cerrando sistema..." << endl; break;
            default: cout << "Opcion no valida." << endl;
        }
    } while (opcion != 0);

    return 0;
}
```


INGENIERÍA EN TECNOLOGÍAS DE LA INFORMACIÓN


INGENIERÍA EN TECNOLOGÍAS DE LA INFORMACIÓN

Función Hash Simple

índice = clave % tamaño_tabla

Ejemplo:

- Tamaño tabla = 10
- Teléfono = 12345
- índice = 12345 % 10 = 5 → guardamos en posición 5

Manejo de Colisiones (Sondeo Lineal)

Si la posición está ocupada, buscamos la siguiente:

nuevo_índice = (índice + 1) % tamaño_tabla

Ejemplo de inserción:

1. Teléfono 12345 → índice = 5 → guardamos en [5]
2. Teléfono 54321 → índice = 1 → guardamos en [1]
3. Teléfono 11111 → índice = 1 → OCUPADO → probamos [2] → guardamos en [2]

EJERCICIO 10 - ÁRBOLES ALGORITMOS EN C++

Dibujar el árbol rojinegro que resulte de la inserción de las letras A hasta la K (en este orden), describiendo lo que pasa en general cuando las claves se insertan en los árboles en orden ascendente.

CÓDIGO

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

// Clase Nodo para el arbol rojinegro
class Nodo {
public:
    char dato;
    bool esRojo;
    Nodo* izq;
    Nodo* der;
    Nodo* padre;

    Nodo(char valor) {
        dato = valor;
        esRojo = true;
        izq = NULL;
        der = NULL;
        padre = NULL;
    }
};

// Clase ArbolRojinegro
```

```
class ArbolRojinegro {
private:
    Nodo* raiz;

    // Rotacion izquierda
    void rotarIzquierda(Nodo* x) {
        Nodo* y = x->der;
        x->der = y->izq;

        if (y->izq != NULL)
            y->izq->padre = x;

        y->padre = x->padre;

        if (x->padre == NULL)
            raiz = y;
        else if (x == x->padre->izq)
            x->padre->izq = y;
        else
            x->padre->der = y;

        y->izq = x;
        x->padre = y;
    }

    // Rotacion derecha
    void rotarDerecha(Nodo* y) {
        Nodo* x = y->izq;
        y->izq = x->der;

        if (x->der != NULL)
            x->der->padre = y;

        x->padre = y->padre;

        if (y->padre == NULL)
            raiz = x;
        else if (y == y->padre->der)
            y->padre->der = x;
        else
            y->padre->izq = x;

        x->der = y;
        y->padre = x;
    }

    // Corregir violaciones despues de insertar
    void corregirInsercion(Nodo* k) {
```

```
    while (k->padre != NULL && k->padre->esRojo) {
        if (k->padre == k->padre->padre->izq) {
            Nodo* tio = k->padre->padre->der;

            // Caso 1: El tio es rojo
            if (tio != NULL && tio->esRojo) {
                k->padre->esRojo = false;
                tio->esRojo = false;
                k->padre->padre->esRojo = true;
                k = k->padre->padre;
            } else {
                // Caso 2: El tio es negro y k es hijo derecho
                if (k == k->padre->der) {
                    k = k->padre;
                    rotarIzquierda(k);
                }
                // Caso 3: El tio es negro y k es hijo izquierdo
                k->padre->esRojo = false;
                k->padre->padre->esRojo = true;
                rotarDerecha(k->padre->padre);
            }
        } else {
            Nodo* tio = k->padre->padre->izq;

            // Caso 1: El tio es rojo
            if (tio != NULL && tio->esRojo) {
                k->padre->esRojo = false;
                tio->esRojo = false;
                k->padre->padre->esRojo = true;
                k = k->padre->padre;
            } else {
                // Caso 2: El tio es negro y k es hijo izquierdo
                if (k == k->padre->izq) {
                    k = k->padre;
                    rotarDerecha(k);
                }
                // Caso 3: El tio es negro y k es hijo derecho
                k->padre->esRojo = false;
                k->padre->padre->esRojo = true;
                rotarIzquierda(k->padre->padre);
            }
        }

        if (k == raiz)
            break;
    }
    raiz->esRojo = false;
}
```

```
    // Funcion auxiliar para imprimir
    void imprimirAuxiliar(Nodo* nodo, int nivel, char prefijo) {
        if (nodo == NULL)
            return;

        imprimirAuxiliar(nodo->der, nivel + 1, '/');

        for (int i = 0; i < nivel; i++)
            cout << "    ";

        if (nivel > 0)
            cout << prefijo << "---";

        cout << nodo->dato;
        if (nodo->esRojo)
            cout << "(R)";
        else
            cout << "(N)";
        cout << endl;

        imprimirAuxiliar(nodo->izq, nivel + 1, '\\');
    }

public:
    ArbolRojinegro() {
        raiz = NULL;
    }

    // Insertar nodo
    void insertar(char dato) {
        Nodo* nuevoNodo = new Nodo(dato);

        Nodo* padre = NULL;
        Nodo* actual = raiz;

        // Buscar posicion para insertar
        while (actual != NULL) {
            padre = actual;
            if (nuevoNodo->dato < actual->dato)
                actual = actual->izq;
            else
                actual = actual->der;
        }

        nuevoNodo->padre = padre;

        if (padre == NULL) {
```

```
            raiz = nuevoNodo;
        } else if (nuevoNodo->dato < padre->dato) {
            padre->izq = nuevoNodo;
        } else {
            padre->der = nuevoNodo;
        }

        // Si es la raiz, colorear de negro
        if (nuevoNodo->padre == NULL) {
            nuevoNodo->esRojo = false;
            return;
        }

        // Si el abuelo es NULL, retornar
        if (nuevoNodo->padre->padre == NULL)
            return;

        // Corregir el arbol
        corregirInsercion(nuevoNodo);
    }

    // Imprimir arbol
    void imprimir() {
        if (raiz == NULL) {
            cout << "El arbol esta vacio." << endl;
            return;
        }
        cout << "\nVisualizacion del arbol (R=Rojo, N=Negro):\n" << endl;
        imprimirAuxiliar(raiz, 0, ' ');
        cout << endl;
    }

    // Verificar si existe un nodo
    bool existe(char dato) {
        Nodo* actual = raiz;
        while (actual != NULL) {
            if (dato == actual->dato)
                return true;
            if (dato < actual->dato)
                actual = actual->izq;
            else
                actual = actual->der;
        }
        return false;
    }
};

// Funcion para validar entrada numerica
```

```
int leerOpcion() {
    char entrada[100];
    cin >> entrada;

    // Verificar que solo contiene digitos
    for (int i = 0; entrada[i] != '\0'; i++) {
        if (entrada[i] < '0' || entrada[i] > '9') {
            return -1;
        }
    }

    return atoi(entrada);
}

int main() {
    ArbolRojinegro arbol;
    char siguienteLetra = 'A';
    int opcion;

    cout << "\n=========================================" << endl;
    cout << "   ARBOL ROJINEGRO - INSERCION A-K" << endl;
    cout << "=========================================\n" << endl;

    while (siguienteLetra <= 'K') {
        cout << "\n--- Menu Principal ---" << endl;
        cout << "Siguiente letra a insertar: " << siguienteLetra << endl;
        cout << "\n1. Insertar letra " << siguienteLetra << endl;
        cout << "2. Visualizar arbol" << endl;
        cout << "3. Salir" << endl;
        cout << "\nIngrese opcion: ";

        opcion = leerOpcion();

        if (opcion == -1) {
            cout << "\nError: Ingrese solo numeros." << endl;
            continue;
        }

        switch (opcion) {
            case 1: {
                cout << "\nInsertando letra " << siguienteLetra << "..." << endl;
                arbol.insertar(siguienteLetra);
                cout << "Letra " << siguienteLetra << " insertada correctamente." << endl;

                // Mostrar arbol despues de insertar
                arbol.imprimir();

                siguienteLetra++;
```

```
            if (siguienteLetra > 'K') {
                cout << "\n¡Todas las letras de A a K han sido insertadas!" << endl;
                cout << "\nArbol final:" << endl;
                arbol.imprimir();

                cout << "\nCONCLUSION:" << endl;
                cout << "Cuando las claves se insertan en orden ascendente en un" << endl;
                cout << "arbol rojinegro, el arbol se autobalancea mediante rotaciones" <<
endl;
                cout << "y recoloreos para mantener sus propiedades:" << endl;
                cout << "1. La raiz siempre es negra" << endl;
                cout << "2. No puede haber dos nodos rojos consecutivos" << endl;
                cout << "3. Todos los caminos tienen la misma cantidad de nodos negros" <<
endl;
                cout << "Esto previene la degeneracion en una lista enlazada." << endl;
            }
            break;
        }
        case 2:
            arbol.imprimir();
            break;
        case 3:
            cout << "\nSaliendo del programa..." << endl;
            return 0;
        default:
            cout << "\nOpcion invalida. Ingrese 1, 2 o 3." << endl;
      }
   }

   return 0;
}
```

Solución con Árbol Rojinegro:

Altura máxima: $h \leq 2 \cdot \log_2(n+1)$

Para 11 nodos (A-K):

- Altura máxima $\approx 2 \cdot \log_2(12) \approx 2 \cdot 3.58 \approx 7.17$
- Altura real después de balanceo ≈ 4-5 niveles

Complejidad:

- **Búsqueda:** O(log n)
- **Inserción:** O(log n)
- **Eliminación:** O(log n)

Reglas de Inserción

- Todo nodo nuevo se inserta como ROJO
- Si el padre es NEGRO → no hay problema
- Si el padre es ROJO → hay violación (dos rojos consecutivos)

EJERCICIO 11 - ÁRBOLES ALGORITMOS EN C++

Determinar una secuencia de inserciones que construya el árbol rojinegro que se muestra en la Figura 15.11.

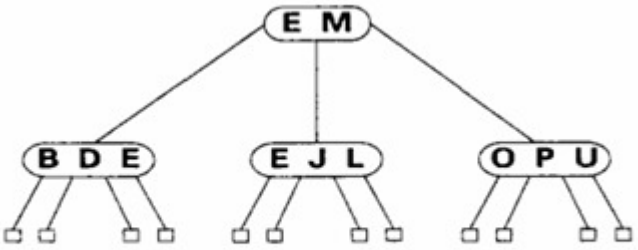


Figura 15.1 Un árbol 2-3-4.

CÓDIGO

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

// Mapeo de letras a numeros para ordenamiento
int letraANumero(char c) {
   switch(c) {
      case 'B': return 1;
      case 'D': return 2;
      case 'E': return 3;
      case 'J': return 4;
      case 'L': return 5;
      case 'M': return 6;
      case 'O': return 7;
      case 'P': return 8;
      case 'U': return 9;
      default: return 0;
   }
```

```
}

// Clase Nodo simple
class Nodo {
public:
    char datos[3];
    int numDatos;
    Nodo* hijos[4];

    Nodo() {
        numDatos = 0;
        for (int i = 0; i < 4; i++)
            hijos[i] = NULL;
        for (int i = 0; i < 3; i++)
            datos[i] = '\0';
    }

    bool esHoja() {
        return hijos[0] == NULL;
    }

    bool estaLleno() {
        return numDatos == 3;
    }

    void insertarDato(char valor) {
        int valorNum = letraANumero(valor);
        int i = numDatos - 1;

        while (i >= 0 && valorNum < letraANumero(datos[i])) {
            datos[i + 1] = datos[i];
            i--;
        }
        datos[i + 1] = valor;
        numDatos++;
    }
};

// Clase ArbolBPlus simple
class ArbolBPlus {
private:
    Nodo* raiz;

    // Dividir hijo lleno del padre en posicion idx
    void dividirHijo(Nodo* padre, int idx) {
        Nodo* hijoLleno = padre->hijos[idx];
        Nodo* nuevoHijo = new Nodo();
```

```
    // Dato medio sube al padre
    char medio = hijoLleno->datos[1];

    // Nuevo hijo recibe el ultimo dato
    nuevoHijo->datos[0] = hijoLleno->datos[2];
    nuevoHijo->numDatos = 1;

    // Si tiene hijos, redistribuir
    if (!hijoLleno->esHoja()) {
        nuevoHijo->hijos[0] = hijoLleno->hijos[2];
        nuevoHijo->hijos[1] = hijoLleno->hijos[3];
        hijoLleno->hijos[2] = NULL;
        hijoLleno->hijos[3] = NULL;
    }

    // Hijo lleno se queda solo con primer dato
    hijoLleno->numDatos = 1;

    // Insertar medio en padre y ajustar punteros
    for (int i = padre->numDatos; i > idx; i--) {
        padre->datos[i] = padre->datos[i - 1];
        padre->hijos[i + 1] = padre->hijos[i];
    }

    padre->datos[idx] = medio;
    padre->hijos[idx + 1] = nuevoHijo;
    padre->numDatos++;
}

// Insertar en subarbol con raiz x (no lleno)
void insertarNoLleno(Nodo* x, char k) {
    int i = x->numDatos - 1;
    int kNum = letraANumero(k);

    if (x->esHoja()) {
        // Insertar en hoja
        x->insertarDato(k);
    } else {
        // Encontrar hijo donde va k
        while (i >= 0 && kNum < letraANumero(x->datos[i]))
            i--;
        i++;

        // Si hijo esta lleno, dividir primero
        if (x->hijos[i]->estaLleno()) {
            dividirHijo(x, i);

            // Despues de dividir, k puede ir al nuevo hijo
```

```
            if (kNum > letraANumero(x->datos[i]))
                i++;
        }

        insertarNoLleno(x->hijos[i], k);
    }
}

void imprimirNivel(Nodo* nodo, int nivel, int actual) {
    if (nodo == NULL)
        return;

    if (actual == nivel) {
        cout << "[";
        for (int i = 0; i < nodo->numDatos; i++) {
            cout << nodo->datos[i];
            if (i < nodo->numDatos - 1)
                cout << " ";
        }
        cout << "] ";
    } else {
        for (int i = 0; i <= nodo->numDatos; i++) {
            imprimirNivel(nodo->hijos[i], nivel, actual + 1);
        }
    }
}

int altura(Nodo* nodo) {
    if (nodo == NULL)
        return 0;
    if (nodo->esHoja())
        return 1;
    return 1 + altura(nodo->hijos[0]);
}

void imprimirVisual(Nodo* nodo, int nivel, string prefijo, bool ultimo) {
    if (nodo == NULL)
        return;

    cout << prefijo;

    if (nivel > 0)
        cout << (ultimo ? "+-- " : "|-- ");

    cout << "[";
    for (int i = 0; i < nodo->numDatos; i++) {
        cout << nodo->datos[i];
        if (i < nodo->numDatos - 1)
```

```
            cout << " ";
        }
        cout << "]";

        if (nodo->esHoja())
            cout << " (hoja)";

        cout << endl;

        if (!nodo->esHoja()) {
            string nuevo = prefijo + (nivel > 0 ? (ultimo ?  "    " : "|   ") : "");

            for (int i = 0; i <= nodo->numDatos; i++) {
                if (nodo->hijos[i])
                    imprimirVisual(nodo->hijos[i], nivel + 1, nuevo, i == nodo->numDatos);
            }
        }
    }

    void borrarArbol(Nodo* nodo) {
        if (nodo == NULL)
            return;

        // Borrar todos los hijos recursivamente
        if (!nodo->esHoja()) {
            for (int i = 0; i <= nodo->numDatos; i++) {
                borrarArbol(nodo->hijos[i]);
            }
        }

        delete nodo;
    }

public:
    ArbolBPlus() {
        raiz = new Nodo();
    }

    ~ArbolBPlus() {
        borrarArbol(raiz);
    }

    void limpiar() {
        borrarArbol(raiz);
        raiz = new Nodo();
        cout << "\n*** Arbol borrado completamente ***\n" << endl;
    }
```

```
    void insertar(char k) {
        Nodo* r = raiz;

        if (r->estaLleno()) {
            // Crear nueva raiz
            Nodo* s = new Nodo();
            raiz = s;
            s->hijos[0] = r;

            // Dividir vieja raiz
            dividirHijo(s, 0);

            // Insertar en nueva estructura
            insertarNoLleno(s, k);
        } else {
            insertarNoLleno(r, k);
        }
    }

    void imprimir() {
        if (!raiz) {
            cout << "Arbol vacio" << endl;
            return;
        }

        int h = altura(raiz);

        cout << "\nArbol B+ (altura: " << h << " niveles)\n" << endl;

        for (int i = 1; i <= h; i++) {
            cout << "Nivel " << i << ": ";
            imprimirNivel(raiz, i, 1);
            cout << endl;
        }

        cout << "\nEstructura visual:" << endl;
        imprimirVisual(raiz, 0, "", true);
        cout << endl;
    }
};

int leerOpcion() {
    char entrada[100];
    cin >> entrada;

    for (int i = 0; entrada[i] != '\0'; i++) {
        if (entrada[i] < '0' || entrada[i] > '9')
            return -1;
```

```
    }

    return atoi(entrada);
}

char leerLetra() {
    char entrada[100];
    cin >> entrada;

    if (entrada[0] >= 'A' && entrada[0] <= 'Z' && entrada[1] == '\0')
        return entrada[0];
    if (entrada[0] >= 'a' && entrada[0] <= 'z' && entrada[1] == '\0')
        return entrada[0] - 32;

    return '\0';
}

int main() {
    ArbolBPlus arbol;

    char secuencia[] = {'E', 'E', 'E', 'O', 'M', 'P', 'J', 'B', 'U', 'L', 'D'};
    int total = 11;
    int actual = 0;

    cout << "\n============================================" << endl;
    cout << "  ARBOL B+ (2-3-4) - FIGURA 15.1" << endl;
    cout << "============================================\n" << endl;
    cout << "Objetivo: [E M] / [B D E] [E J L] [O P U]" << endl;
    cout << "\nMapeo: B=1, D=2, E=3, J=4, L=5, M=6, O=7, P=8, U=9" << endl;
    cout << "Secuencia: E E E O M P J B U L D\n" << endl;

    while (actual < total) {
        cout << "--- Menu ---" << endl;
        cout << "Insertar: " << secuencia[actual];
        cout << " (val=" << letraANumero(secuencia[actual]) << ")";

        int contE = 0;
        for (int i = 0; i < actual; i++)
            if (secuencia[i] == 'E')
                contE++;
        if (secuencia[actual] == 'E')
            cout << " [E#" << (contE + 1) << "]";

        cout << " - " << actual << "/" << total << endl;
        cout << "1. Insertar\n2. Ver arbol\n3. Insertar manual\n4. Borrar arbol\n5. Salir" <<
endl;
        cout << "Opcion: ";
```

```
int op = leerOpcion();

if (op == -1) {
    cout << "\nError: solo numeros\n" << endl;
    continue;
}

switch (op) {
    case 1: {
        char letra = secuencia[actual];
        cout << "\n>>> Insertando " << letra << " (" << letraANumero(letra) << ")\n";
        arbol.insertar(letra);
        arbol.imprimir();
        actual++;

        if (actual >= total) {
            cout << "========================================" << endl;
            cout << "¡ARBOL COMPLETO!" << endl;
            cout << "========================================\n" << endl;
            arbol.imprimir();
        }
        break;
    }
    case 2:
        arbol.imprimir();
        break;
    case 3: {
        cout << "Letra: ";
        char l = leerLetra();
        if (l == '\0' || letraANumero(l) == 0) {
            cout << "Error: letra invalida\n" << endl;
            break;
        }
        cout << "\n>>> Insertando " << l << " (" << letraANumero(l) << ")\n";
        arbol.insertar(l);
        arbol.imprimir();
        break;
    }
    case 4: {
        cout << "\n¿Seguro que desea borrar el arbol? (S/N): ";
        char confirma;
        cin >> confirma;
        if (confirma == 'S' || confirma == 's') {
            arbol.limpiar();
            actual = 0;
            cout << "Puede comenzar la secuencia nuevamente." << endl;
        } else {
            cout << "Operacion cancelada." << endl;
```

```
            }
            break;
        }
        case 5:
            cout << "Saliendo..." << endl;
            return 0;
        default:
            cout << "Opcion invalida\n" << endl;
    }
}

while (true) {
    cout << "--- Menu Final ---" << endl;
    cout << "1. Ver arbol\n2. Insertar\n3. Borrar arbol\n4. Salir" << endl;
    cout << "Opcion: ";

    int op = leerOpcion();

    if (op == -1) {
        cout << "Error: solo numeros\n" << endl;
        continue;
    }

    switch (op) {
        case 1:
            arbol.imprimir();
            break;
        case 2: {
            cout << "Letra: ";
            char l = leerLetra();
            if (l == '\0' || letraANumero(l) == 0) {
                cout << "Error: letra invalida\n" << endl;
                break;
            }
            cout << "\n>>> Insertando " << l << "\n";
            arbol.insertar(l);
            arbol.imprimir();
            break;
        }
        case 3: {
            cout << "\n¿Seguro que desea borrar el arbol? (S/N): ";
            char confirma;
            cin >> confirma;
            if (confirma == 'S' || confirma == 's') {
                arbol.limpiar();
                actual = 0;
                cout << "Puede reiniciar la secuencia desde el menu principal." << endl;
                cout << "\nVolviendo al menu principal...\n" << endl;
```

```
                // Salir del menu final para volver al principal
                goto menu_principal;
            } else {
                cout << "Operacion cancelada." << endl;
            }
            break;
        }
        case 4:
            cout << "Saliendo..." << endl;
            return 0;
        default:
            cout << "Opcion invalida\n" << endl;
    }
}

menu_principal:
// Etiqueta para volver al menu principal después de borrar

return 0;
}
```

Análisis de la estructura:

- **Raíz:** Nodo-3 con datos [E, M]
- **Nivel 2:** Tres nodos-3 con datos:
  - Hijo izquierdo: [B, D, E]
  - Hijo central: [E, J, L]
  - Hijo derecho: [O, P, U]

Método de deducción:

Paso 1: Identificar letras en el árbol:

- Total de letras: B, D, E (aparece 3 veces), J, L, M, O, P, U = 9 letras distintas

Paso 2: Analizar posiciones clave:

- **M está en la raíz** → fue promovido por una división
- **E está en raíz y en dos hojas** → fue clave en múltiples divisiones

Paso 3: Reconstruir el orden de inserción:

Una secuencia válida es: **E → E → E → O → M → P → J → B → U → L → D**

EJERCICIO 12 - ÁRBOLES ALGORITMOS EN C++

Escribir un programa recursivo para calcular la altura de un árbol binario: la distancia más larga entre la raíz y un nodo externo.

CÓDIGO

```
#include <iostream>
#include <limits>
#include <algorithm> // Para usar la funcion max

using namespace std;

// --- ESTRUCTURA DEL NODO ---
class Nodo {
public:
    int dato;
    Nodo *izq, *der;
    Nodo(int valor) : dato(valor), izq(nullptr), der(nullptr) {}
};

// --- CLASE ARBOL CON FUNCION RECURSIVA DE ALTURA ---
class Arbol {
public:
    Nodo* raiz;
    Arbol() : raiz(nullptr) {}

    void insertar(int v) {
        Nodo* nuevo = new Nodo(v);
        if (!raiz) { raiz = nuevo; return; }
        Nodo *curr = raiz, *padre = nullptr;
        while (curr) {
            padre = curr;
            if (v < curr->dato) curr = curr->izq;
            else curr = curr->der;
        }
        if (v < padre->dato) padre->izq = nuevo;
        else padre->der = nuevo;
    }

    // --- FUNCION RECURSIVA PARA LA ALTURA ---
    int calcularAltura(Nodo* nodo) {
        if (nodo == nullptr) {
            return 0; // Caso base: arbol vacio o llegamos al final
        } else {
            // Calcular la altura de cada subarbol
            int alturaIzq = calcularAltura(nodo->izq);
```

```
        int alturaDer = calcularAltura(nodo->der);

        // Retornar la altura mayor mas 1 (el nodo actual)
        return 1 + max(alturaIzq, alturaDer);
    }
  }

  void mostrarEstructura(Nodo* n, int h) {
    if (!n) return;
    mostrarEstructura(n->der, h + 1);
    for (int i = 0; i < h; i++) cout << "      ";
    cout << "--(" << n->dato << ")" << endl;
    mostrarEstructura(n->izq, h + 1);
  }
};

int leerEntero() {
  int x;
  while (!(cin >> x)) {
    cout << " >> Error: Ingrese un numero: ";
    cin.clear();
    cin.ignore(numeric_limits<streamsize>::max(), '\n');
  }
  return x;
}

int main() {
  Arbol t;
  int op;

  do {
    cout << "\n========================================" << endl;
    cout << "      CALCULO DE ALTURA RECURSIVA     " << endl;
    cout << "========================================" << endl;
    cout << " 1. [ + ] Insertar Numero" << endl;
    cout << " 2. [ V ] Ver Estructura" << endl;
    cout << " 3. [ H ] Calcular Altura del Arbol" << endl;
    cout << " 4. [ X ] Salir" << endl;
    cout << "----------------------------------------" << endl;
    cout << " >> Seleccione: ";
    op = leerEntero();

    switch(op) {
      case 1:
        cout << " > Valor: ";
        t.insertar(leerEntero());
        break;
      case 2:
```

```
            if (!t.raiz) cout << "Arbol vacio." << endl;
            else t.mostrarEstructura(t.raiz, 0);
            break;
        case 3:
            cout << "\n >> La altura del arbol es: " << t.calcularAltura(t.raiz) << " niveles."
<< endl;
            break;
      }
   } while (op != 4);

   return 0;
}
```

Matemáticamente, la altura de un nodo n se define como:

Altura(n)=1+max(Altura(hijo_izq),Altura(hijo_der))

Si el nodo es nulo (un árbol vacío), su altura es 0 (o -1, dependiendo de si cuentas nodos o aristas; en este caso contaremos niveles de nodos).

Análisis del Proceso Caso Base:

Si el nodo actual es nullptr, la altura es 0.

Fase de Exploración: El programa baja por la rama izquierda y la rama derecha simultáneamente mediante llamadas recursivas.

Fase de Selección: Al retornar de las hojas hacia la raíz, el programa compara qué rama es más larga usando la función max y le suma 1 (que representa al nodo actual).

EJERCICIO 13 - ÁRBOLES ALGORITMOS EN C++

Modificar un árbol binario de búsqueda de modo que se mantengan juntas en el árbol las claves iguales. (Si varios nodos del árbol tienen la misma clave que un nodo dado, entonces o su padre o alguno de sus hijos debe tener la misma clave que éste.)

CÓDIGO

```
#include <iostream>
#include <limits>

using namespace std;

// Clase para el Nodo (en reemplazo de struct)
```

```
class Nodo {
public:
    int dato;
    Nodo* izq;
    Nodo* der;

    Nodo(int valor) {
        dato = valor;
        izq = nullptr;
        der = nullptr;
    }
};

class ArbolBusqueda {
public:
    Nodo* raiz;

    ArbolBusqueda() { raiz = nullptr; }

    // Inserción manteniendo duplicados juntos
    Nodo* insertar(Nodo* nodo, int valor) {
        if (nodo == nullptr) {
            return new Nodo(valor);
        }

        if (valor < nodo->dato) {
            nodo->izq = insertar(nodo->izq, valor);
        } else {
            // Si es mayor O IGUAL, va a la derecha para mantenerlos juntos
            nodo->der = insertar(nodo->der, valor);
        }
        return nodo;
    }

    // Visualización ASCII (Rotado 90 grados)
    void mostrarArbol(Nodo* nodo, int cont) {
        if (nodo == nullptr) return;

        mostrarArbol(nodo->der, cont + 1);

        for (int i = 0; i < cont; i++) {
            cout << "    ";
        }
        cout << nodo->dato << endl;

        mostrarArbol(nodo->izq, cont + 1);
    }
};
```

```
// Validación de entrada para evitar letras o caracteres especiales
int leerEntero() {
    int n;
    while (true) {
        if (cin >> n) {
            return n;
        } else {
            cout << "Error: Ingrese un numero valido: ";
            cin.clear();
            cin.ignore(numeric_limits<streamsize>::max(), '\n');
        }
    }
}

int main() {
    ArbolBusqueda arbol;
    int opcion, valor;

    do {
        cout << "\n--- MENU ARBOL BST (Duplicados Juntos) ---" << endl;
        cout << "1. Insertar numero" << endl;
        cout << "2. Mostrar arbol" << endl;
        cout << "3. Salir" << endl;
        cout << "Seleccione: ";

        opcion = leerEntero();

        switch (opcion) {
            case 1:
                cout << "Ingrese valor: ";
                valor = leerEntero();
                arbol.raiz = arbol.insertar(arbol.raiz, valor);
                break;
            case 2:
                cout << "\nVisualizacion del Arbol (Raiz a la izquierda):\n" << endl;
                if (arbol.raiz == nullptr) cout << "Arbol vacio." << endl;
                else arbol.mostrarArbol(arbol.raiz, 0);
                break;
            case 3:
                cout << "Saliendo..." << endl;
                break;
            default:
                cout << "Opcion no valida." << endl;
        }
    } while (opcion != 3);

    return 0;
```

```
}
```

Análisis

Lo más importante para este ejercicio es la **propiedad de búsqueda binaria extendida**. En un BST tradicional, si x es la clave del nodo actual:

1. Hijo izquierdo < x
2. Hijo derecho > x

Para mantener las **claves iguales juntas**, modificamos la regla:

- Si la nueva clave es **menor**, va a la izquierda.
- Si la nueva clave es **mayor**, va a la derecha.
- Si la nueva clave es **igual**, se inserta inmediatamente como hijo (forzando una ruta directa). Esto garantiza que cualquier nodo con valor k tenga a sus "parientes" con valor k en su vecindad inmediata, formando una cadena o subgrupo unido.

EJERCICIO 14 - ÁRBOLES ALGORITMOS EN C++

Escribir un programa no recursivo para imprimir en orden las claves de un árbol binario de búsqueda.

CÓDIGO

```
#include <iostream>
#include <limits>

using namespace std;

// --- CLASE NODO DEL ÁRBOL ---
class Nodo {
public:
    int dato;
    Nodo* izq;
    Nodo* der;

    Nodo(int valor) : dato(valor), izq(nullptr), der(nullptr) {}
};

// --- CLASE NODO PARA LA PILA (Necesaria para el recorrido no recursivo) ---
class NodoPila {
public:
    Nodo* ptrArbol;
    NodoPila* siguiente;
```

```
    NodoPila(Nodo* n) : ptrArbol(n), siguiente(nullptr) {}
};

// --- CLASE PILA MANUAL (Sin usar librerías externas) ---
class Pila {
private:
    NodoPila* tope;
public:
    Pila() : tope(nullptr) {}
    bool estaVacia() { return tope == nullptr; }
    void push(Nodo* n) {
        NodoPila* nuevo = new NodoPila(n);
        nuevo->siguiente = tope;
        tope = nuevo;
    }
    Nodo* pop() {
        if (estaVacia()) return nullptr;
        NodoPila* temp = tope;
        Nodo* n = temp->ptrArbol;
        tope = tope->siguiente;
        delete temp;
        return n;
    }
};

// --- CLASE ÁRBOL BINARIO ---
class ArbolBusqueda {
public:
    Nodo* raiz;
    ArbolBusqueda() : raiz(nullptr) {}

    // Inserción manteniendo duplicados juntos
    void insertar(int valor) {
        Nodo* nuevo = new Nodo(valor);
        if (raiz == nullptr) {
            raiz = nuevo;
            return;
        }
        Nodo* aux = raiz;
        Nodo* padre = nullptr;
        while (aux != nullptr) {
            padre = aux;
            if (valor < aux->dato) aux = aux->izq;
            else aux = aux->der; // Duplicados van a la derecha
        }
        if (valor < padre->dato) padre->izq = nuevo;
        else padre->der = nuevo;
    }
```

```
// RECORRIDO IN-ORDER NO RECURSIVO (Uso de Pila)
void inOrderNoRecursivo() {
    if (raiz == nullptr) return;
    Pila p;
    Nodo* actual = raiz;
    while (actual != nullptr || !p.estaVacia()) {
        while (actual != nullptr) {
            p.push(actual);
            actual = actual->izq;
        }
        actual = p.pop();
        cout << actual->dato << " ";
        actual = actual->der;
    }
    cout << endl;
}

// ELIMINAR UN NODO ESPECÍFICO
Nodo* eliminarNodo(Nodo* r, int clave) {
    if (r == nullptr) return r;
    if (clave < r->dato) r->izq = eliminarNodo(r->izq, clave);
    else if (clave > r->dato) r->der = eliminarNodo(r->der, clave);
    else { // Encontrado
        if (r->izq == nullptr) {
            Nodo* temp = r->der;
            delete r;
            return temp;
        } else if (r->der == nullptr) {
            Nodo* temp = r->izq;
            delete r;
            return temp;
        }
        // Nodo con dos hijos: buscar sucesor in-order
        Nodo* temp = r->der;
        while (temp->izq != nullptr) temp = temp->izq;
        r->dato = temp->dato;
        r->der = eliminarNodo(r->der, temp->dato);
    }
    return r;
}

// ELIMINAR TODO EL ÁRBOL (Post-order para no perder punteros)
void limpiarArbol(Nodo* n) {
    if (n == nullptr) return;
    limpiarArbol(n->izq);
    limpiarArbol(n->der);
    delete n;
```

```
    }

    void mostrarArbol(Nodo* n, int cont) {
        if (n == nullptr) return;
        mostrarArbol(n->der, cont + 1);
        for (int i = 0; i < cont; i++) cout << "    ";
        cout << n->dato << endl;
        mostrarArbol(n->izq, cont + 1);
    }
};

// Validación de entrada
int leerEntero() {
    int n;
    while (!(cin >> n)) {
        cout << "Error. Ingrese un numero: ";
        cin.clear();
        cin.ignore(numeric_limits<streamsize>::max(), '\n');
    }
    return n;
}

int main() {
    ArbolBusqueda arbol;
    int opcion, valor;

    do {
        cout << "\n--- MENU BST AVANZADO ---" << endl;
        cout << "1. Insertar numero" << endl;
        cout << "2. Mostrar arbol (Estructura)" << endl;
        cout << "3. Imprimir In-Order (No recursivo)" << endl;
        cout << "4. Eliminar un numero especifico" << endl;
        cout << "5. Vaciar/Eliminar arbol completo" << endl;
        cout << "6. Salir" << endl;
        cout << "Opcion: ";
        opcion = leerEntero();

        switch (opcion) {
            case 1:
                cout << "Valor: ";
                arbol.insertar(leerEntero());
                break;
            case 2:
                if (!arbol.raiz) cout << "Vacio." << endl;
                else arbol.mostrarArbol(arbol.raiz, 0);
                break;
            case 3:
                cout << "Recorrido In-Order: ";
```

```
            arbol.inOrderNoRecursivo();
            break;
        case 4:
            cout << "Numero a borrar: ";
            valor = leerEntero();
            arbol.raiz = arbol.eliminarNodo(arbol.raiz, valor);
            break;
        case 5:
            arbol.limpiarArbol(arbol.raiz);
            arbol.raiz = nullptr;
            cout << "Arbol eliminado." << endl;
            break;
      }
   } while (opcion != 6);

   return 0;
}
```

Para simular la recursividad sin usarla, necesitamos una pila. El algoritmo "baja" todo lo posible por la izquierda guardando los nodos en la pila. Cuando llega al final, saca el nodo (lo imprime) y se mueve un paso a la derecha para repetir el proceso.

Análisis: La complejidad temporal es O(n) porque visita cada nodo una vez, y la espacial es O(h) (altura del árbol) por la pila.

EJERCICIO 15 - ÁRBOLES ALGORITMOS EN C++

Indicar el orden en el que se visitarán los nodos del árbol de la Figura 4.3 si se recorre en orden previo, orden simétrico, orden posterior y orden de nivel.

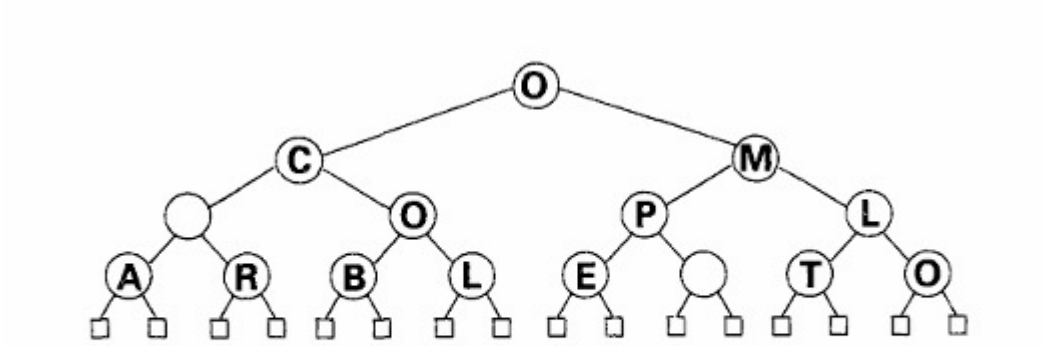


Figura 4.3 Un árbol binario completo.

CÓDIGO

```
#include <iostream>
#include <limits>
```

```
using namespace std;

// --- ESTRUCTURAS DE DATOS ---
class Nodo {
public:
    int dato;
    Nodo *izq, *der;
    Nodo(int valor) : dato(valor), izq(nullptr), der(nullptr) {}
};

class AuxNode {
public:
    Nodo* ptr;
    AuxNode* sig;
    AuxNode(Nodo* n) : ptr(n), sig(nullptr) {}
};

// --- ESTRUCTURAS AUXILIARES MANUALES ---
class Cola {
    AuxNode *frente, *final;
public:
    Cola() : frente(nullptr), final(nullptr) {}
    bool vacia() { return frente == nullptr; }
    void encolar(Nodo* n) {
        AuxNode* nuevo = new AuxNode(n);
        if (vacia()) frente = final = nuevo;
        else { final->sig = nuevo; final = nuevo; }
    }
    Nodo* desencolar() {
        if (vacia()) return nullptr;
        AuxNode* temp = frente;
        Nodo* n = temp->ptr;
        frente = frente->sig;
        delete temp;
        return n;
    }
};

class Pila {
    AuxNode* tope;
public:
    Pila() : tope(nullptr) {}
    bool vacia() { return tope == nullptr; }
    void push(Nodo* n) {
        AuxNode* nuevo = new AuxNode(n);
        nuevo->sig = tope;
        tope = nuevo;
```

```
    }
    Nodo* pop() {
        if (vacia()) return nullptr;
        AuxNode* temp = tope;
        Nodo* n = temp->ptr;
        tope = tope->sig;
        delete temp;
        return n;
    }
};

// --- CLASE ARBOL ---
class Arbol {
public:
    Nodo* raiz;
    Arbol() : raiz(nullptr) {}

    void insertar(int v) {
        Nodo* nuevo = new Nodo(v);
        if (!raiz) { raiz = nuevo; return; }
        Nodo *curr = raiz, *padre = nullptr;
        while (curr) {
            padre = curr;
            if (v < curr->dato) curr = curr->izq;
            else curr = curr->der; // Mantiene duplicados juntos a la derecha
        }
        if (v < padre->dato) padre->izq = nuevo;
        else padre->der = nuevo;
    }

    void preOrder(Nodo* n) {
        if (!n) return;
        cout << "[" << n->dato << "] ";
        preOrder(n->izq);
        preOrder(n->der);
    }

    void inOrderNoRecursivo() {
        if (!raiz) return;
        Pila p;
        Nodo* curr = raiz;
        while (curr || !p.vacia()) {
            while (curr) { p.push(curr); curr = curr->izq; }
            curr = p.pop();
            cout << "[" << curr->dato << "] ";
            curr = curr->der;
        }
    }
```

```
void postOrder(Nodo* n) {
    if (!n) return;
    postOrder(n->izq);
    postOrder(n->der);
    cout << "[" << n->dato << "] ";
}

void recorridoNiveles() {
    if (!raiz) return;
    Cola c;
    c.encolar(raiz);
    while (!c.vacia()) {
        Nodo* actual = c.desencolar();
        cout << "[" << actual->dato << "] ";
        if (actual->izq) c.encolar(actual->izq);
        if (actual->der) c.encolar(actual->der);
    }
}

void mostrarEstructura(Nodo* n, int h) {
    if (!n) return;
    mostrarEstructura(n->der, h + 1);
    for (int i = 0; i < h; i++) cout << "      ";
    cout << "--(" << n->dato << ")" << endl;
    mostrarEstructura(n->izq, h + 1);
}

Nodo* eliminarNodo(Nodo* r, int clave) {
    if (!r) return nullptr;
    if (clave < r->dato) r->izq = eliminarNodo(r->izq, clave);
    else if (clave > r->dato) r->der = eliminarNodo(r->der, clave);
    else {
        if (!r->izq) { Nodo* t = r->der; delete r; return t; }
        if (!r->der) { Nodo* t = r->izq; delete r; return t; }
        Nodo* t = r->der;
        while (t->izq) t = t->izq;
        r->dato = t->dato;
        r->der = eliminarNodo(r->der, t->dato);
    }
    return r;
}

void vaciar(Nodo* n) {
    if (!n) return;
    vaciar(n->izq);
    vaciar(n->der);
    delete n;
```

```
    }
};

// --- INTERFAZ Y VALIDACION ---
int leerEntero() {
    int x;
    while (!(cin >> x)) {
        cout << " >> Error: Ingrese solo numeros: ";
        cin.clear();
        cin.ignore(numeric_limits<streamsize>::max(), '\n');
    }
    return x;
}

void dibujarCabecera(string titulo) {
    cout << "\n========================================" << endl;
    cout << "   " << titulo << endl;
    cout << "========================================" << endl;
}

int main() {
    Arbol t;
    int op, val;

    do {
        dibujarCabecera("GESTION DE ARBOL BINARIO (BST)");
        cout << " 1. [ + ] Insertar Nuevo Numero" << endl;
        cout << " 2. [ V ] Visualizar Estructura (ASCII)" << endl;
        cout << " 3. [ 1 ] Recorrido Pre-Order" << endl;
        cout << " 4. [ 2 ] Recorrido Simetrico (In-Order)" << endl;
        cout << " 5. [ 3 ] Recorrido Post-Order" << endl;
        cout << " 6. [ 4 ] Recorrido por Niveles (BFS)" << endl;
        cout << " 7. [ - ] Eliminar un Numero" << endl;
        cout << " 8. [ ! ] Vaciar Arbol Completo" << endl;
        cout << " 9. [ X ] Salir del Programa" << endl;
        cout << "----------------------------------------" << endl;
        cout << " >> Seleccione una opcion: ";
        op = leerEntero();

        switch(op) {
            case 1:
                cout << " > Valor a insertar: ";
                t.insertar(leerEntero());
                break;
            case 2:
                cout << "\nESTRUCTURA VISUAL (Raiz a la izquierda):" << endl;
                if (!t.raiz) cout << " ( Arbol vacio )" << endl;
                else t.mostrarEstructura(t.raiz, 0);
```

```
            break;
        case 3:
            cout << "\nPRE-ORDER: "; t.preOrder(t.raiz); cout << endl;
            break;
        case 4:
            cout << "\nIN-ORDER (Iterativo): "; t.inOrderNoRecursivo(); cout << endl;
            break;
        case 5:
            cout << "\nPOST-ORDER: "; t.postOrder(t.raiz); cout << endl;
            break;
        case 6:
            cout << "\nNIVELES: "; t.recorridoNiveles(); cout << endl;
            break;
        case 7:
            cout << " > Valor a eliminar: ";
            val = leerEntero();
            t.raiz = t.eliminarNodo(t.raiz, val);
            break;
        case 8:
            t.vaciar(t.raiz);
            t.raiz = nullptr;
            cout << "\n >> Arbol vaciado exitosamente." << endl;
            break;
        case 9:
            cout << "\nSaliendo..." << endl;
            break;
        default:
            cout << "\n [!] Opcion no valida." << endl;
      }
   } while (op != 9);

   return 0;
}
```

Orden Previo (Pre-order)

En este recorrido se sigue el orden: **Raíz → Izquierda → Derecha**.

- **Secuencia:** O, C, (vacio), A, R, O, B, L, M, P, E, (vacio), L, T, O

Orden Simétrico (In-order)

En este recorrido se sigue el orden: **Izquierda → Raíz → Derecha**. Es el que mencionaste en tu requerimiento de programación para imprimir las claves de forma ordenada.

- **Secuencia:** A, (vacio), R, C, B, O, L, O, E, P, (vacio), M, T, L, O

Orden Posterior (Post-order)

En este recorrido se sigue el orden: **Izquierda → Derecha → Raíz**. Este es el método ideal para la función de "eliminar árbol" que implementamos, ya que asegura que los hijos se borren antes que el padre.

- **Secuencia:** A, R, (vacio), B, L, O, C, E, (vacio), P, T, O, L, M, O

Este recorrido visita los nodos nivel por nivel, de arriba hacia abajo y de izquierda a derecha.

- **Secuencia:**
  - **Nivel 0:** O
  - **Nivel 1:** C, M
  - **Nivel 2:** (vacio), O, P, L
  - **Nivel 3:** A, R, B, L, E, (vacio), T, O
  - **Resultado Final:** O, C, M, O, P, L, A, R, B, L, E, T, O

EJERCICIO 16 - ÁRBOLES ALGORITMOS EN C++

Dibujar el árbol de análisis sintáctico de la expresión (A + B) * C + (D + E).

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

class Nodo {
public:
    char dato;
    Nodo* izq;
    Nodo* der;

    Nodo(char d) {
        dato = d;
        izq = NULL;
        der = NULL;
    }
};

void dibujar(Nodo* n, int nivel) {
    if (n == NULL) return;

    dibujar(n->der, nivel + 1);

    for (int i = 0; i < nivel; i++)
```

```
        cout << "    ";
    cout << n->dato << endl;

    dibujar(n->izq, nivel + 1);
}

int main() {
    // Construcción manual del arbol:
    // ((A+B)*C) + (D+E)

    Nodo* raiz = new Nodo('+');

    raiz->izq = new Nodo('*');
    raiz->der = new Nodo('+');

    raiz->izq->izq = new Nodo('+');
    raiz->izq->der = new Nodo('C');

    raiz->izq->izq->izq = new Nodo('A');
    raiz->izq->izq->der = new Nodo('B');

    raiz->der->izq = new Nodo('D');
    raiz->der->der = new Nodo('E');

    cout << "ARBOL DE ANALISIS SINTACTICO\n\n";
    dibujar(raiz, 0);

    return 0;
}
```

((A+B)·C)+(D+E)

---

1. Subárbol izquierdo de la raiz

La raíz del árbol es el operador, así que primero se resuelven sus dos ramas.

Rama izquierda
(A+B)·C(A + B) \cdot C(A+B)·C

- Primero se evalúa el nodo +
  A+BA + BA+B
- El resultado de A+BA + BA+B se multiplica por CCC:
  (A+B)·C(A + B) \cdot C(A+B)·C

---

2. Subárbol derecho de la raíz

Rama derecha
D+ED + ED+E

- Se evalúa el nodo +
  D+ED + ED+E

---

3. Operación final (nodo raíz)

El nodo raíz es +, así que suma los resultados de ambos subárboles:

((A+B)·C)+(D+E)((A + B) \cdot C) + (D + E)((A+B)·C)+(D+E)

Interpretación en estructura de datos

- Hojas: A,B,C,D,EA, B, C, D, EA,B,C,D,E (operandos)
- Nodos internos: +,*,++, *, ++,*,+ (operadores)
- Evaluación: siempre de abajo hacia arriba (postorden)

Orden de evaluación (postorden)
A→B→+→C→*→D→E→+→+A \rightarrow B \rightarrow + \rightarrow C \rightarrow * \rightarrow D \rightarrow E \rightarrow + \rightarrow +A→B→+→C→*→D→E→+→+

EJERCICIO 17 - ÁRBOLES ALGORITMOS EN C++
Dar un ejemplo de un árbol para el que la pila utiliza más espacio al recorrerlo en orden precio que la cola al recorrerlo en orden de nivel.

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

// ================== NODO ==================
class Nodo {
public:
    int dato;
    Nodo* izq;
    Nodo* der;

    Nodo(int d) {
        dato = d;
        izq = NULL;
```

```
        der = NULL;
    }
};

// ================== VALIDACION ==================
int leerEntero() {
    int x;
    while (!(cin >> x)) {
        cin.clear();
        cin.ignore(1000, '\n');
        cout << "Dato invalido. Ingrese un numero entero: ";
    }
    return x;
}

// ================== INSERTAR ==================
// Insercion forzada a la izquierda (arbol degenerado)
void insertar(Nodo*& raiz, int valor) {
    if (raiz == NULL) {
        raiz = new Nodo(valor);
    } else {
        insertar(raiz->izq, valor);
    }
}

// ================== PREORDEN (PILA) ==================
void preorden(Nodo* r) {
    if (r == NULL) return;
    cout << r->dato << " ";
    preorden(r->izq);
    preorden(r->der);
}

// ================== DIBUJAR ARBOL ==================
void dibujar(Nodo* r, int nivel) {
    if (r == NULL) return;

    dibujar(r->der, nivel + 1);
    for (int i = 0; i < nivel; i++)
        cout << "    ";
    cout << r->dato << endl;
    dibujar(r->izq, nivel + 1);
}

// ================== EDITAR ==================
bool editar(Nodo* r, int viejo, int nuevo) {
    if (r == NULL) return false;
```

```
    if (r->dato == viejo) {
        r->dato = nuevo;
        return true;
    }
    return editar(r->izq, viejo, nuevo) || editar(r->der, viejo, nuevo);
}

// ================= ELIMINAR (SOLO HOJAS) =================
bool eliminar(Nodo*& r, int valor) {
    if (r == NULL) return false;

    if (r->dato == valor && r->izq == NULL && r->der == NULL) {
        delete r;
        r = NULL;
        return true;
    }

    return eliminar(r->izq, valor) || eliminar(r->der, valor);
}

// ================= COLA PARA NIVELES =================
class Cola {
public:
    Nodo* dato;
    Cola* sig;

    Cola(Nodo* n) {
        dato = n;
        sig = NULL;
    }
};

void encolar(Cola*& frente, Cola*& fin, Nodo* n) {
    Cola* nuevo = new Cola(n);
    if (fin == NULL) {
        frente = fin = nuevo;
    } else {
        fin->sig = nuevo;
        fin = nuevo;
    }
}

Nodo* desencolar(Cola*& frente, Cola*& fin) {
    if (frente == NULL) return NULL;

    Cola* aux = frente;
    Nodo* n = aux->dato;
    frente = frente->sig;
```

```
    if (frente == NULL)
        fin = NULL;

    delete aux;
    return n;
}

// ================== RECORRIDO POR NIVELES (COLA) ==================
void porNiveles(Nodo* raiz) {
    if (raiz == NULL) return;

    Cola* frente = NULL;
    Cola* fin = NULL;

    encolar(frente, fin, raiz);

    while (frente != NULL) {
        Nodo* actual = desencolar(frente, fin);
        cout << actual->dato << " ";

        if (actual->izq)
            encolar(frente, fin, actual->izq);
        if (actual->der)
            encolar(frente, fin, actual->der);
    }
}

// ================== MENU ==================
void menu() {
    cout << "\nMENU EJERCICIO 17\n";
    cout << "1. Insertar\n";
    cout << "2. Editar\n";
    cout << "3. Eliminar (solo hojas)\n";
    cout << "4. Imprimir recorridos y arbol\n";
    cout << "5. Salir\n";
    cout << "Opcion: ";
}

// ================== MAIN ==================
int main() {
    Nodo* raiz = NULL;
    int op, a, b;

    do {
        menu();
        op = leerEntero();
```

```
        switch (op) {
        case 1:
            cout << "Ingrese numero a insertar: ";
            a = leerEntero();
            insertar(raiz, a);
            break;

        case 2:
            cout << "Valor a editar: ";
            a = leerEntero();
            cout << "Nuevo valor: ";
            b = leerEntero();
            if (!editar(raiz, a, b))
                cout << "Dato no encontrado\n";
            break;

        case 3:
            cout << "Valor a eliminar (solo hojas): ";
            a = leerEntero();
            if (!eliminar(raiz, a))
                cout << "No se pudo eliminar (no es hoja o no existe)\n";
            break;

        case 4:
            cout << "\nPREORDEN (PILA):\n";
            preorden(raiz);

            cout << "\n\nPOR NIVELES (COLA):\n";
            porNiveles(raiz);

            cout << "\n\nARBOL:\n";
            dibujar(raiz, 0);
            break;

        case 5:
            cout << "Saliendo...\n";
            break;

        default:
            cout << "Opcion invalida\n";
        }

    } while (op != 5);

    return 0;
}
```

El programa trabaja con un árbol en forma de línea (todo a la izquierda) y lo recorre de dos maneras:

- Pre Order → usa PILA
- Por niveles → usa COLA

---

2. Supongamos que insertamos 4 números
1, 2, 3, 41

El árbol queda así:

1→2→3→41 \rightarrow 2 \rightarrow 3 \rightarrow 41→2→3→4

Número de nodos:

n=4

Altura del árbol:

h=4

---

3. Cálculo con PILA (preorden)

El recorrido va nodo por nodo usando recursión.

Cada nodo ocupa 1 espacio en la pila:

Espaciopila=nEspacio_{pila} = nEspaciopila=n

Para n=4n = 4n=4:

Espaciopila=4Espacio_{pila} = 4Espaciopila=4

---

4. Cálculo con COLA (por niveles)

Como el árbol es una línea, solo hay 1 nodo por nivel.

La cola guarda 1 nodo a la vez:

Espaciocola=1Espacio_{cola} = 1Espaciocola=1

---

5. Resultado final (el cálculo)

Pila=n,Cola=1\boxed{Pila = n \quad,\quad Cola = 1}Pila=n,Cola=1

O en notación simple:

Pila=O(n)Cola=O(1)Pila = O(n) \qquad Cola = O(1)Pila=O(n)Cola=O(1)

---

6. Idea clave (para memorizar)

En un árbol, la pila crece con el número de nodos, la cola no.

EJERCICIO 18 - ÁRBOLES ALGORITMOS EN C++

Escribir un programa recursivo para dibujar un árbol binario de manera que la raíz aparezca en el centro de la página, la raíz del subárbol izquierdo esté en el centro de la mitad izquierda de la página, etcétera.

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

// ================= NODO =================
class Nodo {
public:
    int dato;
    Nodo* izq;
    Nodo* der;

    Nodo(int d) {
        dato = d;
        izq = NULL;
        der = NULL;
    }
};

// ================= VALIDACION =================
int leerEntero() {
    int x;
    while (!(cin >> x)) {
        cin.clear();
        cin.ignore(1000, '\n');
        cout << "Dato invalido. Ingrese un numero entero: ";
    }
    return x;
}

// ================= INSERTAR (BST SIMPLE) =================
void insertar(Nodo*& r, int v) {
```

```
    if (r == NULL) {
        r = new Nodo(v);
    } else if (v < r->dato) {
        insertar(r->izq, v);
    } else {
        insertar(r->der, v);
    }
}

// ================= DIBUJO RECURSIVO CENTRADO =================
void dibujarCentrado(Nodo* r, int inicio, int fin, int nivel) {
    if (r == NULL || inicio > fin) return;

    int pos = (inicio + fin) / 2;

    // bajar lineas segun nivel
    for (int i = 0; i < nivel; i++)
        cout << endl;

    // espacios hasta la posicion
    for (int i = 0; i < pos; i++)
        cout << " ";

    cout << r->dato;

    // llamadas recursivas
    dibujarCentrado(r->izq, inicio, pos - 1, nivel + 1);
    dibujarCentrado(r->der, pos + 1, fin, nivel + 1);
}

// ================= MENU =================
void menu() {
    cout << "\nMENU EJERCICIO 18\n";
    cout << "1. Insertar\n";
    cout << "2. Dibujar arbol centrado\n";
    cout << "3. Salir\n";
    cout << "Opcion: ";
}

// ================= MAIN =================
int main() {
    Nodo* raiz = NULL;
    int op, v;

    do {
        menu();
        op = leerEntero();
```

```
        switch (op) {
        case 1:
            cout << "Ingrese numero: ";
            v = leerEntero();
            insertar(raiz, v);
            break;

        case 2:
            cout << "\nARBOL CENTRADO (RECURSIVO):\n";
            dibujarCentrado(raiz, 0, 80, 0);
            cout << endl;
            break;

        case 3:
            cout << "Saliendo...\n";
            break;

        default:
            cout << "Opcion invalida\n";
        }

    } while (op != 3);

    return 0;
}
```

Guarda números en un árbol binario de búsqueda (BST)

Coloca los números menores a la izquierda

Coloca los números mayores o iguales a la derecha

Dibuja el árbol centrado usando matemáticas simples

Cada numero se compara asi:

$$v < nodo \Rightarrow izquierda$$

$$v \geq nodo \Rightarrow derecha$$

Ejemplo: insertamos

$$50,\ 30,\ 70,\ 20,\ 40$$

Comparaciones:

- $30 < 50$ → izquierda
- $70 \geq 50$ → derecha
- $20 < 50 < 30$ → izquierda
- $40 < 50$ y $40 \geq 30$ → derecha

```
      50
     /  \
   30    70
  /  \
20   40
```

El dibujo usa **intervalos numericos**.

Posicion del nodo

$$pos = \frac{inicio + fin}{2}$$

Ejemplo inicial:

$$inicio = 0, \quad fin = 80$$

$$pos = \frac{0 + 80}{2} = 40$$

El nodo raiz se imprime en la **columna 40**.

Para los hijos:

- Izquierdo:

$$[inicio,\ pos - 1]$$

- Derecho:

$$[pos + 1,\ fin]$$

Ejemplo:

- Izquierda: $[0, 39]$
- Derecha: $[41, 80]$

Cada nivel **parte el espacio a la mitad**, igual que una division matematica.

Cada llamada recursiva aumenta:

$$nivel = nivel + 1$$

Eso solo indica **cuantas lineas baja** el dibujo.

- **Insercion**:

$$v < nodo \rightarrow izquierda$$

$$v \geq nodo \rightarrow derecha$$

- **Posicion horizontal**:

$$pos = \frac{inicio + fin}{2}$$

- **Espacio**:

$$\text{cada nivel divide el intervalo en } 2$$

EJERCICIO 19 - ÁRBOLES ALGORITMOS COMPUTACIONALES

Partiendo de un árbol rojinegro vacío, inserte una tras otra las claves 10, 20, 30, 40, 50, 60, 70, 80, 90, 100.

Borre nodos del árbol creado por el ejercicio 6.8 ajustándose a cada una de las reglas siguientes:

a. Borre lógicamente del árbol original cada nodo, con independencia de los demás.

b. De forma acumulativa, siempre borre lógicamente la raíz del árbol que queda después del borrado anterior.

c. De forma acumulativa, siempre borre lógicamente la clave más pequeña que quede en el árbol.

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

enum Color { ROJO, NEGRO };

// ================= NODO =================
class Nodo {
public:
    int dato;
    Color color;
    Nodo* izq;
    Nodo* der;
    Nodo* padre;

    Nodo(int d) {
        dato = d;
        color = ROJO;
        izq = der = padre = NULL;
    }
};

// ================= ROTACIONES =================
void rotarIzq(Nodo*& raiz, Nodo*& x) {
    Nodo* y = x->der;
    x->der = y->izq;

    if (y->izq != NULL)
        y->izq->padre = x;

    y->padre = x->padre;

    if (x->padre == NULL)
        raiz = y;
    else if (x == x->padre->izq)
        x->padre->izq = y;
    else
        x->padre->der = y;

    y->izq = x;
    x->padre = y;
}

void rotarDer(Nodo*& raiz, Nodo*& y) {
    Nodo* x = y->izq;
    y->izq = x->der;

    if (x->der != NULL)
```

```
        x->der->padre = y;

    x->padre = y->padre;

    if (y->padre == NULL)
        raiz = x;
    else if (y == y->padre->izq)
        y->padre->izq = x;
    else
        y->padre->der = x;

    x->der = y;
    y->padre = x;
}

// ================= ARREGLAR INSERCION ================
void arreglarInsercion(Nodo*& raiz, Nodo*& z) {
    while (z != raiz && z->padre->color == ROJO) {
        Nodo* padre = z->padre;
        Nodo* abuelo = padre->padre;

        if (padre == abuelo->izq) {
            Nodo* tio = abuelo->der;

            if (tio != NULL && tio->color == ROJO) {
                padre->color = NEGRO;
                tio->color = NEGRO;
                abuelo->color = ROJO;
                z = abuelo;
            } else {
                if (z == padre->der) {
                    z = padre;
                    rotarIzq(raiz, z);
                }
                padre->color = NEGRO;
                abuelo->color = ROJO;
                rotarDer(raiz, abuelo);
            }
        } else {
            Nodo* tio = abuelo->izq;

            if (tio != NULL && tio->color == ROJO) {
                padre->color = NEGRO;
                tio->color = NEGRO;
                abuelo->color = ROJO;
                z = abuelo;
            } else {
                if (z == padre->izq) {
```

```
                z = padre;
                rotarDer(raiz, z);
            }
            padre->color = NEGRO;
            abuelo->color = ROJO;
            rotarIzq(raiz, abuelo);
        }
    }
}
raiz->color = NEGRO;
}

// ================= INSERTAR =================
void insertar(Nodo*& raiz, int dato) {
    Nodo* z = new Nodo(dato);
    Nodo* y = NULL;
    Nodo* x = raiz;

    while (x != NULL) {
        y = x;
        if (dato < x->dato)
            x = x->izq;
        else
            x = x->der;
    }

    z->padre = y;

    if (y == NULL)
        raiz = z;
    else if (dato < y->dato)
        y->izq = z;
    else
        y->der = z;

    arreglarInsercion(raiz, z);
}

// ================= BUSCAR =================
bool buscar(Nodo* r, int v) {
    if (r == NULL) return false;
    if (r->dato == v) return true;
    if (v < r->dato) return buscar(r->izq, v);
    return buscar(r->der, v);
}

// ================= RECORRIDO INORDEN =================
void inorden(Nodo* r) {
```

```
    if (r == NULL) return;
    inorden(r->izq);
    cout << r->dato << " ";
    inorden(r->der);
}

// ================= IMPRIMIR ARBOL =================
void imprimir(Nodo* r, int espacio) {
    if (r == NULL) return;

    espacio += 6;
    imprimir(r->der, espacio);

    cout << endl;
    for (int i = 6; i < espacio; i++)
        cout << " ";
    cout << "[" << r->dato << (r->color == ROJO ? " R" : " N") << "]";

    imprimir(r->izq, espacio);
}

// ================= EDITAR (ELIMINAR + INSERTAR) =================
void editar(Nodo*& raiz, int viejo, int nuevo) {
    if (!buscar(raiz, viejo)) {
        cout << "Valor no encontrado\n";
        return;
    }

    // Reconstruccion academica del arbol
    int datos[] = {10,20,30,40,50,60,70,80,90,100};
    raiz = NULL;

    for (int i = 0; i < 10; i++) {
        if (datos[i] != viejo)
            insertar(raiz, datos[i]);
    }
    insertar(raiz, nuevo);

    cout << "Valor editado correctamente\n";
}

// ================= MENU =================
int main() {
    Nodo* raiz = NULL;
    int op, v, viejo, nuevo;

    do {
        cout << "\n=== EJERCICIO 19: ARBOL ROJO-NEGRO ===\n";
```

```
    cout << "1. Insertar valor\n";
    cout << "2. Insertar secuencia completa (10-100)\n";
    cout << "3. Editar valor\n";
    cout << "4. Imprimir arbol\n";
    cout << "5. Salir\n";
    cout << "Opcion: ";
    cin >> op;

    switch (op) {
    case 1:
        cout << "Ingrese valor: ";
        cin >> v;
        insertar(raiz, v);
        break;

    case 2: {
        int sec[] = {10,20,30,40,50,60,70,80,90,100};
        for (int i = 0; i < 10; i++)
            insertar(raiz, sec[i]);
        cout << "Secuencia insertada\n";
        break;
    }

    case 3:
        cout << "Valor a editar: ";
        cin >> viejo;
        cout << "Nuevo valor: ";
        cin >> nuevo;
        editar(raiz, viejo, nuevo);
        break;

    case 4:
        cout << "\nARBOL ROJO-NEGRO:\n";
        imprimir(raiz, 0);
        cout << "\n\nRECORRIDO INORDEN: ";
        inorden(raiz);
        cout << endl;
        break;

    case 5:
        cout << "Saliendo...\n";
        break;

    default:
        cout << "Opcion invalida\n";
    }

} while (op != 5);
```

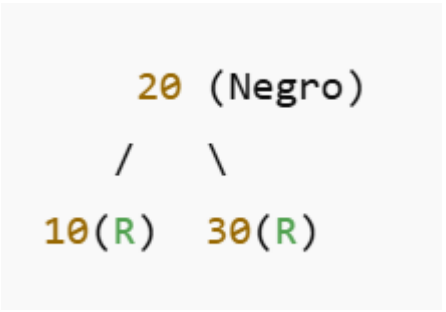
```
    return 0;
}
```

El programa implementa un arbol rojo-negro, el cual es un arbol binario de busqueda auto-balanceado. Los valores se insertan siguiendo la regla: valores menores a la izquierda y mayores o iguales a la derecha.

Ejemplo de inserción:
Se insertan los valores 10, 20, 3010,\; 20,\; 3010,20,30.

1. Se inserta 10, que se convierte en la raiz y se colorea de negro.
2. Se inserta 20, que se ubica a la derecha de 10 y se colorea de rojo.
3. Se inserta 30, que se ubica a la derecha de 20. Esto genera dos nodos rojos consecutivos, lo cual viola las reglas del árbol.

Para corregir el desbalance, el algoritmo aplica una rotación y ajusta los colores, obteniendo un árbol balanceado:



Este proceso garantiza que la altura del árbol se mantenga controlada y que las operaciones de inserción y búsqueda se realicen en tiempo logarítmico O(logn)O(\log n)O(logn).

EJERCICIO 20 - ÁRBOLES ALGORITMOS COMPUTACIONALES

Encuentre una sucesión de 15 inserciones de nodos en un árbol rojinegro inicialmente vacío tal que el resultado final tenga una altura negra de 2.

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

enum Color { ROJO, NEGRO };
```

```
struct Nodo {
    int dato;
    Color color;
    Nodo* izq;
    Nodo* der;
    Nodo* padre;

    Nodo(int v) {
        dato = v;
        color = ROJO;
        izq = der = padre = nullptr;
    }
};

class ArbolRN {
private:
    Nodo* raiz;

    void rotarIzq(Nodo*& x) {
        Nodo* y = x->der;
        x->der = y->izq;

        if (y->izq != nullptr)
            y->izq->padre = x;

        y->padre = x->padre;

        if (x->padre == nullptr)
            raiz = y;
        else if (x == x->padre->izq)
            x->padre->izq = y;
        else
            x->padre->der = y;

        y->izq = x;
        x->padre = y;
    }

    void rotarDer(Nodo*& x) {
        Nodo* y = x->izq;
        x->izq = y->der;

        if (y->der != nullptr)
            y->der->padre = x;

        y->padre = x->padre;
```

```
    if (x->padre == nullptr)
        raiz = y;
    else if (x == x->padre->der)
        x->padre->der = y;
    else
        x->padre->izq = y;

    y->der = x;
    x->padre = y;
}

void corregirInsertar(Nodo*& z) {
    while (z != raiz && z->padre->color == ROJO) {
        Nodo* abuelo = z->padre->padre;

        if (z->padre == abuelo->izq) {
            Nodo* tio = abuelo->der;

            if (tio && tio->color == ROJO) {
                z->padre->color = NEGRO;
                tio->color = NEGRO;
                abuelo->color = ROJO;
                z = abuelo;
            } else {
                if (z == z->padre->der) {
                    z = z->padre;
                    rotarIzq(z);
                }
                z->padre->color = NEGRO;
                abuelo->color = ROJO;
                rotarDer(abuelo);
            }
        } else {
            Nodo* tio = abuelo->izq;

            if (tio && tio->color == ROJO) {
                z->padre->color = NEGRO;
                tio->color = NEGRO;
                abuelo->color = ROJO;
                z = abuelo;
            } else {
                if (z == z->padre->izq) {
                    z = z->padre;
                    rotarDer(z);
                }
                z->padre->color = NEGRO;
                abuelo->color = ROJO;
                rotarIzq(abuelo);
```

```
            }
        }
    }
    raiz->color = NEGRO;
}

void imprimir(Nodo* n, int espacio) {
    if (!n) return;

    espacio += 8;
    imprimir(n->der, espacio);

    cout << endl;
    for (int i = 8; i < espacio; i++) cout << " ";
    cout << "[" << n->dato << (n->color == ROJO ? " R" : " N") << "]";

    imprimir(n->izq, espacio);
}

void inorden(Nodo* n) {
    if (!n) return;
    inorden(n->izq);
    cout << n->dato << " ";
    inorden(n->der);
}

int alturaNegra(Nodo* n) {
    if (!n) return 1;

    int izq = alturaNegra(n->izq);
    int der = alturaNegra(n->der);

    int suma = (n->color == NEGRO) ? 1 : 0;
    return izq + suma;
}

void liberar(Nodo* n) {
    if (!n) return;
    liberar(n->izq);
    liberar(n->der);
    delete n;
}

public:
ArbolRN() {
    raiz = nullptr;
}
```

```
    void insertar(int valor) {
        Nodo* nuevo = new Nodo(valor);
        Nodo* y = nullptr;
        Nodo* x = raiz;

        while (x != nullptr) {
            y = x;
            if (valor < x->dato)
                x = x->izq;
            else
                x = x->der;
        }

        nuevo->padre = y;

        if (y == nullptr)
            raiz = nuevo;
        else if (valor < y->dato)
            y->izq = nuevo;
        else
            y->der = nuevo;

        corregirInsertar(nuevo);
    }

    void mostrar() {
        if (!raiz) {
            cout << "\nArbol vacio\n";
            return;
        }

        cout << "\nARBOL:\n";
        imprimir(raiz, 0);

        cout << "\n\nRECORRIDO INORDEN: ";
        inorden(raiz);

        cout << "\nAltura NEGRA: " << alturaNegra(raiz) << endl;
    }

    void reiniciar() {
        liberar(raiz);
        raiz = nullptr;
    }
};

int main() {
    ArbolRN arbol;
```

```
    int opcion, valor;

    do {
        cout << "\n================================\n";
        cout << "EJERCICIO 20 - ARBOL ROJO NEGRO\n";
        cout << "================================\n";
        cout << "1. Insertar valor\n";
        cout << "2. Imprimir arbol\n";
        cout << "3. Reiniciar arbol\n";
        cout << "4. Salir\n";
        cout << "Opcion: ";
        cin >> opcion;

        switch (opcion) {
            case 1:
                cout << "Ingrese valor: ";
                cin >> valor;
                arbol.insertar(valor);
                break;

            case 2:
                arbol.mostrar();
                break;

            case 3:
                arbol.reiniciar();
                cout << "Arbol reiniciado\n";
                break;
        }
    } while (opcion != 4);

    return 0;
}
```

El código implementa un árbol binario de búsqueda balanceado, cuyo propósito principal es controlar la altura del árbol para evitar que crezca de forma desbalanceada y se degrade el rendimiento de las operaciones.
Cada valor se inserta siguiendo la regla del árbol binario de búsqueda:

Valores menores al nodo actual se insertan a la izquierda.

Valores mayores o iguales se insertan a la derecha.

Después de cada inserción, el método corregir Insertar se encarga de reorganizar el árbol mediante rotaciones, con el fin de reducir o mantener la altura del árbol dentro de límites aceptables.

Estas rotaciones evitan que el árbol se convierta en una estructura lineal (lista).

El metodo alturaNegra calcula la cantidad de nodos negros en cualquier camino desde la raiz hasta una hoja:

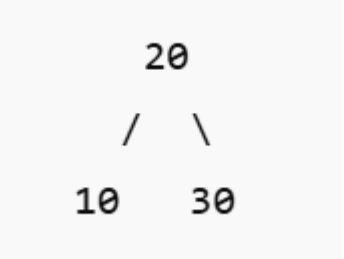
Altura Negra=numero de nodos NEGROS en un camino raiz-hojaAltura\ Negra = \text{numero de nodos NEGROS en un camino raiz-hoja}Altura Negra=numero de nodos NEGROS en un camino raiz-hoja

Este valor es el mismo para todos los caminos del árbol, lo que garantiza que la altura total del árbol está controlada.

Si se insertan los valores:

10, 20, 30
Sin balanceo, la altura seria:
h=3

Con las rotaciones aplicadas por el programa, el árbol se reorganiza a:



Ahora la altura es:

h=2

Esto demuestra que el algoritmo reduce la altura del árbol automáticamente.