Nombre: Isabela Zambrano **NRC:** 29852

Fecha: 02/02/2025

ENUNCIADO 1:

B. Libros

Cuando Valera tiene tiempo libre, va a la biblioteca a leer. Hoy tiene t minutos libres para leer. Por eso, Valera llevó n libros a la biblioteca y, para cada libro, calculó el tiempo que le tomaría leerlo. Numeremos los libros con números enteros del 1 al n. Valera necesita a_i minutos para leer el i-ésimo libro.

Valera decidió elegir un libro arbitrario con el número i y leer los libros uno por uno, empezando por este. En otras palabras, primero leerá el libro número i, luego el libro número i + 1, luego el libro número i + 2, y así sucesivamente. Continúa el proceso hasta que se le acaba el tiempo libre o termina de leer el libro n. Valera lee cada libro hasta el final; es decir, no empieza a leer el libro si no tiene suficiente tiempo libre para terminarlo.

Imprima el número máximo de libros que Valera puede leer.

ENTRADA

La primera línea contiene dos enteros n y t ($1 \leq n \leq 10^5$; $1 \leq t \leq 10^9$): el número de libros y el número de minutos libres que tiene Valera. La segunda línea contiene una secuencia de n enteros a_1, a_2, ..., an ($1 \leq a_i \leq 10^4$), donde el número ai indica el número de minutos que el niño necesita para leer el i-ésimo libro.

SALIDA

Imprima un solo entero: el número máximo de libros que Valera puede leer.

Ejemplo

Entrada
6 7 3 1 1 3 1 1
Salida
5

Entrada
4 5 3 1 2 1
Salida
3

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

// ------------------------------------------------------
// BUSQUEDA BINARIA
// ------------------------------------------------------
int busquedaBinaria(long long* pref, int n, int inicio, long long t) {
    int lo = inicio;
    int hi = n;
    int ans = inicio - 1;

    while (lo <= hi) {
        int mid = lo + (hi - lo) / 2;

        long long suma = *(pref + mid) - *(pref + (inicio - 1));

        if (suma <= t) {
            ans = mid;
            lo = mid + 1;
        } else {
            hi = mid - 1;
        }
    }

    return ans;
}

int main() {

    int n;
    long long t;

    cout << "Ingrese n y t: ";
    cin >> n >> t;

    long long* a = static_cast<long long*>(malloc(n * sizeof(long long)));

    cout << "Ingrese los tiempos de lectura de los " << n << " libros:\n";
    for (int i = 0; i < n; i++) {
        cin >> *(a + i);
    }

    long long* pref = static_cast<long long*>(malloc((n + 1) * sizeof(long long)));
    *(pref + 0) = 0;

    for (int i = 1; i <= n; i++) {
        *(pref + i) = *(pref + i - 1) + *(a + (i - 1));
    }
```

```
    int best = 0;

    for (int i = 1; i <= n; i++) {
        int j = busquedaBinaria(pref, n, i, t);
        int len = j - i + 1;
        if (len > best) best = len;
    }

    cout << "\nMaximo numero de libros que puede leer: " << best << endl;

    free(a);
    free(pref);

    return 0;
}
```

ENUNCIADO 2:

A. Cortar la cinta

Polycarpus tiene una cinta de longitud n. Quiere cortarla de forma que se cumplan las dos condiciones siguientes:

- Después de cortarla, cada trozo de cinta debe tener una longitud a, b o c.
- Después de cortarla, el número de trozos de cinta debe ser el máximo.

Ayuda a Policarpo a encontrar el número de trozos de cinta después del corte requerido.

ENTRADA

La primera línea contiene cuatro enteros separados por espacios: n, a, b y c ($1 \leq n, a, b, c \leq 4000$): la longitud de la cinta original y las longitudes aceptables de los trozos de cinta después del corte, respectivamente. Los números a, b y c pueden coincidir.

SALIDA

Imprima un solo número: el máximo número posible de piezas de cinta. Se garantiza que exista al menos un corte de cinta correcto.

Entrada
5 5 3 2
Salida
2

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

bool posible(int n, int a, int b, int c, int k) {

    for (int x = 0; x <= k; x++) {

        long long rhs = 1LL * n - 1LL * k * c - 1LL * x * (a - c);
        long long denom = (b - c);

        long long y;

        if (denom == 0) {

            if (rhs == 0) {
                y = 0;
            } else {
                continue;
            }
        } else {
            if (rhs % denom != 0) continue;
            y = rhs / denom;
        }

        long long z = k - x - y;

        if (y >= 0 && z >= 0) {
            return true;
        }
    }

    return false;
}

// ----------------------------------------------------------
// BÚSQUEDA BINARIA PARA MAXIMIZAR k
// ----------------------------------------------------------
int main() {

    int n, a, b, c;
    cout << "Ingrese n, a, b, c:\n";
    cin >> n >> a >> b >> c;

    int lo = 0;
    int hi = n;
    int best = 0;

    while (lo <= hi) {
        int mid = lo + (hi - lo) / 2;
```

```
        if (posible(n, a, b, c, mid)) {
            best = mid;
            lo = mid + 1;
        } else {
            hi = mid - 1;
        }
    }

    cout << "\nMaximo numero de piezas: " << best << endl;
    return 0;
}
```

Problema D. Distanciamiento Social

El granjero Juan está preocupado por la salud de sus vacas después de un brote de la altamente contagiosa enfermedad bovina COWVID-19.

Para limitar la transmisión de la enfermedad, las NN vacas del granjero Juan ($2 \leq N \leq 10^5$) han decidido practicar "distanciamiento social" y dispersarse a lo largo de la granja. La granja tiene forma de una línea numérica 1D, con M intervalos mutuamente disjuntos ($1 \leq M \leq 10^5$) en los que hay pasto para pastar. Las vacas quieren ubicarse en puntos enteros distintos, cada uno cubierto de pasto, para maximizar el valor de D, donde D representa la distancia entre el par más cercano de vacas. Ayuda a las vacas a determinar el mayor valor posible de D.

ENTRADA
La primera línea contiene N y M. Las siguientes M líneas describen un intervalo en términos de dos enteros a y b, donde $0 \leq a \leq b \leq 10^{18}$Ningún par de intervalos se superpone o se toca en sus extremos. Una vaca parada en el extremo de un intervalo cuenta como parada sobre pasto.

SALIDA
Imprime el mayor valor posible de D tal que todos los pares de vacas estén separados por al menos D unidades. Se garantiza que existe una solución con D>0.

Entrada
5 3 0 2 4 7 9 9
Salida
2

Una forma de lograr D=2 es colocar vacas en las posiciones 0, 2, 4, 6 y 9.

PUNTUACIÓN:

- Casos de prueba 2–3 satisfacen $b \leq 10^5$
- Casos de prueba 4–10 no tienen restricciones adicionales.

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

int inputNum(string Mensaje, bool negativos){
   if(negativos){
      cout << Mensaje;
      int input = 0;
      cin >> input;
      return input;
   }
   bool validInput = false;
   int input = 0;
   do{
      cout << Mensaje;
      cin >> input;
      if(input <= 0){
         cout << "Entrada Invalida, debe ser mayor a cero\n";
      }
      else {
         validInput = true;
      }
   } while(!validInput);
   return input;
}

int* inputIntVector(int size, string msg, bool negatives){
   int* vector = new int[size];
   for(int i = 0; i < size; i++){
      *(vector+i) = inputNum(msg, negatives);
   }
   return vector;
}

long long* inputLongVector(int size, string msg){
   long long* vector = new long long[size];
   for(int i = 0; i < size; i++){
      cout << msg;
      cin >> *(vector+i);
   }
   return vector;
}

// BUBBLE SORT para L y R juntos
void bubbleSortIntervals(long long* L, long long* R, int size){
   for (int i = 0; i < size - 1; i++) {
```

```
        for (int j = 0; j < size - i - 1; j++) {
            if (*(L + j) > *(L + (j + 1))) {

                long long tmpL = *(L + j);
                *(L + j) = *(L + (j + 1));
                *(L + (j + 1)) = tmpL;

                long long tmpR = *(R + j);
                *(R + j) = *(R + (j + 1));
                *(R + (j + 1)) = tmpR;
            }
        }
    }
}

// ----------------------
// FUNCIÓN CAN(D)
// ----------------------
// Verifica si podemos poner N vacas a distancia >= D
bool canPlace(long long* L, long long* R, int M, long long N, long long D){
    long long cows = 0;
    long long pos = -900000000000000000LL; // -∞ manual

    for (int i = 0; i < M; i++) {

        long long a = *(L + i);
        long long b = *(R + i);

        // Elegir el primer lugar válido dentro del intervalo
        if (pos + D <= a)
            pos = a;
        else
            pos = pos + D;

        // Colocar vacas dentro del intervalo mientras sea posible
        while (pos <= b) {
            cows++;
            if (cows >= N) return true;
            pos = pos + D;
        }
    }
    return false;
}

// ----------------------
// EJERCICIO DISTANCING
// ----------------------
int ejercicioDistancing(){
    long long N = inputNum("Ingrese cuantas vacas habran: ", false);
    int M = inputNum("Ingrese cuantos intervalos existen: ", false);
```

```
    cout << "\n--- Intervalos ---\n";
    long long* L = new long long[M];
    long long* R = new long long[M];

    for (int i = 0; i < M; i++){
        cout << "Ingrese inicio del intervalo " << i+1 << ": ";
        cin >> *(L + i);
        cout << "Ingrese fin del intervalo " << i+1 << ": ";
        cin >> *(R + i);
    }

    // Ordenar intervalos por L
    bubbleSortIntervals(L, R, M);

    long long low = 1;
    long long high = 1000000000000000000LL;
    long long ans = 1;

    // BÚSQUEDA BINARIA REAL SOBRE D
    while(low <= high){
        long long mid = (low + high) / 2;

        if (canPlace(L, R, M, N, mid)){
            ans = mid;       // funciona → intento más grande
            low = mid + 1;
        }
        else {
            high = mid - 1;  // no funciona → achico
        }
    }

    cout << "La maxima distancia D es: " << ans << "\n";

    delete[] L;
    delete[] R;
    return 0;
}

// ----------------------
// MAIN
// ----------------------
int main(){

    cout << "\n--- EJERCICIO DISTANCING (PROBLEMA D) ---\n";
    ejercicioDistancing();

    return 0;
}
```

EJERCICIO HASH

Se requiere desarrollar un sistema eficiente para almacenar y gestionar información de personas, específicamente sus nombres y números de cédula. El sistema debe permitir la rápida inserción y búsqueda de personas, evitando duplicados y validando la información ingresada. Implementar una tabla Hash para los datos ingresados.

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

const int TAM_TABLA = 20; // Tamaño de la tabla hash

// ============================================
// ESTRUCTURAS DE DATOS
// ============================================
struct Persona {
    string nombre;
    string cedula;
    Persona* siguiente;
    bool ocupado;

    Persona() : nombre(""), cedula(""), siguiente(nullptr), ocupado(false) {}

    Persona(string n, string c) : nombre(n), cedula(c), siguiente(nullptr), ocupado(true)
{}
};

struct TablaHash {
    Persona** tabla; // Puntero doble usando malloc
    int totalPersonas;
    int totalColisiones;

    // Constructor usando malloc
    TablaHash() : totalPersonas(0), totalColisiones(0) {
        // Reservar memoria con malloc (equivalente a Persona* tabla[TAM_TABLA])
        tabla = (Persona**)malloc(TAM_TABLA * sizeof(Persona*));

        for(int i = 0; i < TAM_TABLA; i++) {
            Persona* nuevaPersona = new Persona();
            *(tabla + i) = nuevaPersona;
        }
    }

    // Destructor
    ~TablaHash() {
        limpiar();
        // Liberar memoria malloc
```

```
        for(int i = 0; i < TAM_TABLA; i++) {
            delete *(tabla + i);
        }
        free(tabla);
    }

    // Limpiar toda la tabla
    void limpiar() {
        for(int i = 0; i < TAM_TABLA; i++) {
            Persona* actual = *(tabla + i);

            // Limpiar lista enlazada
            Persona* temp = actual->siguiente;
            while(temp != nullptr) {
                Persona* siguiente = temp->siguiente;
                delete temp;
                temp = siguiente;
            }

            // Reiniciar nodo principal
            actual->nombre = "";
            actual->cedula = "";
            actual->siguiente = nullptr;
            actual->ocupado = false;
        }
        totalPersonas = 0;
        totalColisiones = 0;
    }
};

// ============================================
// VARIABLES GLOBALES
// ============================================
TablaHash* tablaPersonas = new TablaHash();

// ============================================
// FUNCIONES AUXILIARES
// ============================================

// Función para obtener un carácter de un string sin []
char obtenerCaracter(const string& str, int indice) {
    if(indice < 0 || indice >= str.length()) return '\0';
    return str.c_str()[indice];
}

// Extraer subcadena sin substr()
string extraerSubcadena(const string& str, int inicio, int longitud) {
    string resultado = "";
    int fin = inicio + longitud;
    if(fin > str.length()) fin = str.length();
```

```
    for(int i = inicio; i < fin; i++) {
        resultado += obtenerCaracter(str, i);
    }
    return resultado;
}

// Convertir string a número sin stoi()
int stringANumero(const string& str) {
    int resultado = 0;
    for(int i = 0; i < str.length(); i++) {
        char c = obtenerCaracter(str, i);
        if(c >= '0' && c <= '9') {
            resultado = resultado * 10 + (c - '0');
        }
    }
    return resultado;
}

// ============================================
// FUNCIONES DE VALIDACIÓN
// ============================================

bool validarNombre(const string& nombre) {
    if(nombre.empty() || nombre.length() < 2) {
        cout << "Error: El nombre debe tener al menos 2 caracteres.\n";
        return false;
    }

    for(int i = 0; i < nombre.length(); i++) {
        char c = obtenerCaracter(nombre, i);
        if(!isalpha(c) && c != ' ' && c != '.') {
            cout << "Error: El nombre solo puede contener letras, espacios y puntos.\n";
            return false;
        }
    }
    return true;
}

bool validarCedulaEcuatoriana(const string& cedula) {
    if(cedula.length() != 10) {
        cout << "Error: La cédula debe tener 10 dígitos.\n";
        return false;
    }

    // Verificar que todos sean dígitos
    for(int i = 0; i < 10; i++) {
        char c = obtenerCaracter(cedula, i);
        if(!(c >= '0' && c <= '9')) {
            cout << "Error: La cédula solo puede contener números.\n";
```

```
            return false;
        }
    }

    // Validar provincia (01-24, 30)
    string provincia = extraerSubcadena(cedula, 0, 2);
    int prov = stringANumero(provincia);
    if(prov < 1 || (prov > 24 && prov != 30)) {
        cout << "Error: Los primeros dos dígitos no corresponden a una provincia válida
(01-24, 30).\n";
        return false;
    }

    // Validar tercer dígito (0-6)
    int tercerDigito = obtenerCaracter(cedula, 2) - '0';
    if(tercerDigito > 6) {
        cout << "Error: El tercer dígito debe estar entre 0 y 6.\n";
        return false;
    }

    // Algoritmo de validación
    int suma = 0;
    int coeficientes[9] = {2, 1, 2, 1, 2, 1, 2, 1, 2};

    for(int i = 0; i < 9; i++) {
        int digito = obtenerCaracter(cedula, i) - '0';
        int producto = digito * *(coeficientes + i);

        if(producto >= 10) {
            producto -= 9;
        }
        suma += producto;
    }

    int decenaSuperior = ((suma / 10) + 1) * 10;
    int digitoVerificador = decenaSuperior - suma;

    if(digitoVerificador == 10) digitoVerificador = 0;

    int ultimoDigito = obtenerCaracter(cedula, 9) - '0';

    if(digitoVerificador != ultimoDigito) {
        cout << "Error: Cédula inválida según el dígito verificador.\n";
        return false;
    }

    return true;
}

bool validarCedulaGeneral(const string& cedula) {
```

```
    if(cedula.empty()) {
        cout << "Error: La cédula no puede estar vacía.\n";
        return false;
    }

    for(int i = 0; i < cedula.length(); i++) {
        char c = obtenerCaracter(cedula, i);
        if(!(c >= '0' && c <= '9')) {
            cout << "Error: La cédula debe contener solo números.\n";
            return false;
        }
    }

    if(cedula.length() < 5 || cedula.length() > 15) {
        cout << "Error: La cédula debe tener entre 5 y 15 dígitos.\n";
        return false;
    }

    return true;
}

// ============================================
// FUNCIONES HASH
// ============================================

int hashCedula(const string& cedula) {
    long long suma = 0;
    for(int i = 0; i < cedula.length(); i++) {
        char c = obtenerCaracter(cedula, i);
        suma = (suma * 31 + c) % TAM_TABLA;
    }
    return suma % TAM_TABLA;
}

int hashNombre(const string& nombre) {
    int suma = 0;
    for(int i = 0; i < nombre.length(); i++) {
        char c = obtenerCaracter(nombre, i);
        suma = (suma * 17 + tolower(c)) % TAM_TABLA;
    }
    return suma % TAM_TABLA;
}

// ============================================
// OPERACIONES CON LA TABLA HASH
// ============================================

bool existeCedula(const string& cedula) {
    int pos = hashCedula(cedula);
    Persona* posicion = *(tablaPersonas->tabla + pos);
```

```
    // Buscar en la posición principal
    if(posicion->ocupado && posicion->cedula == cedula) {
        return true;
    }

    // Buscar en la lista enlazada
    Persona* actual = posicion->siguiente;
    while(actual != nullptr) {
        if(actual->ocupado && actual->cedula == cedula) {
            return true;
        }
        actual = actual->siguiente;
    }

    return false;
}

bool existeNombre(const string& nombre) {
    for(int i = 0; i < TAM_TABLA; i++) {
        Persona* posicion = *(tablaPersonas->tabla + i);

        if(posicion->ocupado && posicion->nombre == nombre) {
            return true;
        }

        Persona* actual = posicion->siguiente;
        while(actual != nullptr) {
            if(actual->ocupado && actual->nombre == nombre) {
                return true;
            }
            actual = actual->siguiente;
        }
    }
    return false;
}

bool insertarPersona(const string& nombre, const string& cedula) {
    // Validar datos
    if(!validarNombre(nombre)) {
        return false;
    }

    cout << "\nTipo de cédula:\n";
    cout << "1. Ecuatoriana (10 dígitos con validación completa)\n";
    cout << "2. Otro país (validación básica)\n";
    cout << "Seleccione opción: ";

    int opcion;
    cin >> opcion;
```

```
cin.ignore();

bool cedulaValida = false;
if(opcion == 1) {
   cedulaValida = validarCedulaEcuatoriana(cedula);
} else {
   cedulaValida = validarCedulaGeneral(cedula);
}

if(!cedulaValida) {
   return false;
}

// Verificar si la cédula ya existe
if(existeCedula(cedula)) {
   cout << "Error: La cédula " << cedula << " ya está registrada.\n";
   return false;
}

// Calcular hash
int pos = hashCedula(cedula);
int hashNom = hashNombre(nombre);

// Crear nueva persona
Persona* nuevaPersona = new Persona(nombre, cedula);

// Obtener posición en la tabla
Persona* posicion = *(tablaPersonas->tabla + pos);

// Insertar en la tabla
if(!posicion->ocupado) {
   // Caso: posición vacía
   posicion->nombre = nombre;
   posicion->cedula = cedula;
   posicion->ocupado = true;
   delete nuevaPersona;

   cout << "\n✓ Persona insertada exitosamente.\n";
   cout << "  Posición en tabla: " << pos << "\n";
   cout << "  Hash de nombre: " << hashNom << "\n";
   cout << "  Hash de cédula: " << pos << "\n";
} else {
   // Caso: colisión - agregar a lista enlazada
   Persona* actual = posicion;
   while(actual->siguiente != nullptr) {
      actual = actual->siguiente;
   }
   actual->siguiente = nuevaPersona;
   tablaPersonas->totalColisiones++;
```

```
        cout << "\n✓ Persona insertada exitosamente (con colisión).\n";
        cout << "  Posición base: " << pos << "\n";
        cout << "  Hash de nombre: " << hashNom << "\n";
        cout << "  Hash de cédula: " << pos << "\n";
        cout << "  Se resolvió con lista enlazada.\n";
    }

    tablaPersonas->totalPersonas++;
    return true;
}

void buscarPorCedula(const string& cedula) {
    int pos = hashCedula(cedula);
    int intentos = 0;

    cout << "\n=== BUSCANDO CÉDULA: " << cedula << " ===\n";
    cout << "Hash calculado: " << pos << "\n";

    // Obtener posición
    Persona* posicion = *(tablaPersonas->tabla + pos);

    // Buscar en posición principal
    if(posicion->ocupado && posicion->cedula == cedula) {
        cout << "\n✓ PERSONA ENCONTRADA EN POSICIÓN PRINCIPAL " << pos
<< "\n";
        cout << "  Nombre: " << posicion->nombre << "\n";
        cout << "  Cédula: " << posicion->cedula << "\n";
        cout << "  Hash de nombre: " << hashNombre(posicion->nombre) << "\n";
        return;
    }

    // Buscar en lista enlazada
    Persona* actual = posicion->siguiente;
    int posEnLista = 0;
    while(actual != nullptr) {
        intentos++;
        if(actual->ocupado && actual->cedula == cedula) {
            cout << "\n✓ PERSONA ENCONTRADA EN LISTA ENLAZADA\n";
            cout << "  Posición base: " << pos << "\n";
            cout << "  Posición en lista: " << posEnLista << "\n";
            cout << "  Intentos de búsqueda: " << intentos << "\n";
            cout << "  Nombre: " << actual->nombre << "\n";
            cout << "  Cédula: " << actual->cedula << "\n";
            cout << "  Hash de nombre: " << hashNombre(actual->nombre) << "\n";
            return;
        }
        actual = actual->siguiente;
        posEnLista++;
    }
```

```
    cout << "\n✗ PERSONA NO ENCONTRADA\n";
    cout << "  La cédula " << cedula << " no existe en la tabla.\n";
}

void buscarPorNombre(const string& nombre) {
    cout << "\n=== BUSCANDO NOMBRE: " << nombre << " ===\n";
    int hashNom = hashNombre(nombre);
    int encontrados = 0;

    cout << "Hash calculado del nombre: " << hashNom << "\n\n";

    for(int i = 0; i < TAM_TABLA; i++) {
        Persona* posicion = *(tablaPersonas->tabla + i);

        if(posicion->ocupado && posicion->nombre == nombre) {
            encontrados++;
            cout << "\n  Persona " << encontrados << ":\n";
            cout << "    Nombre: " << posicion->nombre << "\n";
            cout << "    Cédula: " << posicion->cedula << "\n";
            cout << "    Hash de cédula: " << hashCedula(posicion->cedula) << "\n";
            cout << "    Posición en tabla: " << i << "\n";
        }

        Persona* actual = posicion->siguiente;
        while(actual != nullptr) {
            if(actual->ocupado && actual->nombre == nombre) {
                encontrados++;
                cout << "\n  Persona " << encontrados << " (en lista enlazada):\n";
                cout << "    Nombre: " << actual->nombre << "\n";
                cout << "    Cédula: " << actual->cedula << "\n";
                cout << "    Hash de cédula: " << hashCedula(actual->cedula) << "\n";
                cout << "    Posición base: " << i << "\n";
            }
            actual = actual->siguiente;
        }
    }

    if(encontrados == 0) {
        cout << "✗ No se encontraron personas con el nombre '" << nombre << "'\n";
    } else {
        cout << "\n✓ Total encontrados: " << encontrados << " persona(s)\n";
    }
}

// ============================================
// VISUALIZACIÓN DE LA TABLA HASH
// ============================================

void mostrarTablaHash() {
    cout << "\n" << string(70, '=') << "\n";
```

```
    cout << "                    TABLA HASH COMPLETA\n";
    cout << string(70, '=') << "\n\n";

    cout << "Tamaño de tabla: " << TAM_TABLA << " posiciones\n";
    cout << "Total personas: " << tablaPersonas->totalPersonas << "\n";
    cout << "Total colisiones: " << tablaPersonas->totalColisiones << "\n";
    double factorCarga = (double)tablaPersonas->totalPersonas / TAM_TABLA;
    cout << "Factor de carga: " << factorCarga << "\n\n";

    for(int i = 0; i < TAM_TABLA; i++) {
        Persona* posicion = *(tablaPersonas->tabla + i);

        cout << "[" << i << "] ";

        if(posicion->ocupado) {
            // Mostrar persona principal
            cout << "* " << posicion->nombre
                 << " | Ced: " << posicion->cedula
                 << " | H(ced):" << hashCedula(posicion->cedula)
                 << " | H(nom):" << hashNombre(posicion->nombre);

            // Mostrar lista enlazada si existe
            Persona* actual = posicion->siguiente;
            int elementosEnLista = 0;
            while(actual != nullptr) {
                if(elementosEnLista == 0) cout << " \n     |";
                else cout << "     |";

                cout << "--> " << actual->nombre
                     << " | Ced: " << actual->cedula
                     << " | H(ced):" << hashCedula(actual->cedula);

                actual = actual->siguiente;
                elementosEnLista++;

                if(actual != nullptr) cout << "\n";
            }

            if(elementosEnLista > 0) {
                cout << " \n     |-> Total en lista: " << elementosEnLista << " persona(s)";
            }
        } else {
            cout << "[VACIO]";
        }
        cout << "\n" << string(50, '-') << "\n";
    }
}

void mostrarEstadisticas() {
    cout << "\n" << string(60, '=') << "\n";
```

```
    cout << "                ESTADÍSTICAS DETALLADAS\n";
    cout << string(60, '=') << "\n";

    int posicionesOcupadas = 0;
    int maxElementosEnLista = 0;
    int totalListasNoVacias = 0;

    // Array temporal para distribución
    int* distribucion = (int*)malloc(TAM_TABLA * sizeof(int));

    for(int i = 0; i < TAM_TABLA; i++) {
        int elementosEnPosicion = 0;
        Persona* posicion = *(tablaPersonas->tabla + i);

        if(posicion->ocupado) {
            elementosEnPosicion++;
            posicionesOcupadas++;

            Persona* actual = posicion->siguiente;
            while(actual != nullptr) {
                elementosEnPosicion++;
                actual = actual->siguiente;
            }
        }

        *(distribucion + i) = elementosEnPosicion;

        if(elementosEnPosicion > 0) {
            totalListasNoVacias++;
        }

        if(elementosEnPosicion > maxElementosEnLista) {
            maxElementosEnLista = elementosEnPosicion;
        }
    }

    cout << "\nDISTRIBUCION DE ELEMENTOS:\n";
    for(int i = 0; i < TAM_TABLA; i++) {
        cout << "  Posicion " << i << ": " << *(distribucion + i) << " persona(s) | ";
        for(int j = 0; j < *(distribucion + i); j++) cout << "#";
        cout << "\n";
    }

    cout << "\nESTADISTICAS:\n";
    cout << "  * Posiciones ocupadas: " << posicionesOcupadas << "/" << TAM_TABLA
<< "\n";
    cout << "  * Posiciones vacias: " << (TAM_TABLA - posicionesOcupadas) << "\n";
    cout << "  * Total personas: " << tablaPersonas->totalPersonas << "\n";
    cout << "  * Total colisiones: " << tablaPersonas->totalColisiones << "\n";
```

```
    cout << "  * Maximo en una posicion: " << maxElementosEnLista << " persona(s)\n";
    double factorCarga = (double)tablaPersonas->totalPersonas / TAM_TABLA;
    cout << "  * Factor de carga: " << factorCarga << "\n";
    cout << "  * Eficiencia de busqueda promedio: "
         << (tablaPersonas->totalColisiones > 0 ? "O(n)" : "O(1)") << "\n";

    // Liberar memoria del array temporal
    free(distribucion);
}

// ============================================
// FUNCIONES DE PRUEBA
// ============================================

void generarDatosPrueba() {
    struct PersonaPrueba {
        string nombre;
        string cedula;
    };

    // Array de prueba usando malloc
    PersonaPrueba* datosPrueba = (PersonaPrueba*)malloc(10 * sizeof(PersonaPrueba));

    // Inicializar datos de prueba
    *(datosPrueba + 0) = {"Juan Perez", "1710034567"};
    *(datosPrueba + 1) = {"Maria Garcia", "0912345678"};
    *(datosPrueba + 2) = {"Carlos Lopez", "1712345678"};
    *(datosPrueba + 3) = {"Ana Martinez", "0912345698"};
    *(datosPrueba + 4) = {"Luis Rodriguez", "1712345698"};
    *(datosPrueba + 5) = {"Sofia Gonzalez", "0912345687"};
    *(datosPrueba + 6) = {"Pedro Sanchez", "1712345687"};
    *(datosPrueba + 7) = {"Laura Fernandez", "0912345667"};
    *(datosPrueba + 8) = {"Diego Ramirez", "1712345667"};
    *(datosPrueba + 9) = {"Elena Torres", "0912345657"};

    cout << "\n=== GENERANDO DATOS DE PRUEBA ===\n";

    for(int i = 0; i < 10; i++) {
        cout << "\nInsertando: " << (*(datosPrueba + i)).nombre
             << " - " << (*(datosPrueba + i)).cedula;
        insertarPersona((*(datosPrueba + i)).nombre, (*(datosPrueba + i)).cedula);
    }

    cout << "\n✓ 10 personas de prueba insertadas correctamente.\n";

    // Liberar memoria
    free(datosPrueba);
}
```

```
// ============================================
// FUNCIÓN PRINCIPAL
// ============================================
int main() {
    int opcion;
    string nombre, cedula;

                                                    cout                                    <<
"============================================================\n";
    cout << "     SISTEMA DE GESTION DE PERSONAS CON TABLA HASH   \n";
    cout << "         (Implementado con malloc sin arrays)       \n";
                                                    cout                                    <<
"============================================================\n\n";

    do {
        cout << "\n" << string(50, '=') << "\n";
        cout << "                    MENU PRINCIPAL\n";
        cout << string(50, '=') << "\n";
        cout << "1. Insertar nueva persona\n";
        cout << "2. Buscar persona por cedula\n";
        cout << "3. Buscar personas por nombre\n";
        cout << "4. Mostrar tabla hash completa\n";
        cout << "5. Mostrar estadisticas detalladas\n";
        cout << "6. Generar datos de prueba (10 personas)\n";
        cout << "7. Limpiar toda la tabla\n";
        cout << "8. Validar una cedula ecuatoriana\n";
        cout << "9. Salir del programa\n";
        cout << "\nSeleccione una opcion: ";
        cin >> opcion;
        cin.ignore();

        switch(opcion) {
            case 1:
                cout << "\n=== INSERTAR NUEVA PERSONA ===\n";
                cout << "Ingrese el nombre completo: ";
                getline(cin, nombre);
                cout << "Ingrese la cedula: ";
                getline(cin, cedula);

                if(insertarPersona(nombre, cedula)) {
                    cout << "\n✓ Operacion completada exitosamente.\n";
                } else {
                    cout << "\n✗ No se pudo insertar la persona.\n";
                }
                break;

            case 2:
                cout << "\n=== BUSCAR POR CEDULA ===\n";
                cout << "Ingrese la cedula a buscar: ";
```

```
            getline(cin, cedula);
            buscarPorCedula(cedula);
            break;

        case 3:
            cout << "\n=== BUSCAR POR NOMBRE ===\n";
            cout << "Ingrese el nombre a buscar: ";
            getline(cin, nombre);
            buscarPorNombre(nombre);
            break;

        case 4:
            mostrarTablaHash();
            break;

        case 5:
            mostrarEstadisticas();
            break;

        case 6:
            generarDatosPrueba();
            break;

        case 7:
            cout << "\n¿Está seguro de limpiar toda la tabla? (s/n): ";
            char confirmar;
            cin >> confirmar;
            if(tolower(confirmar) == 's') {
                tablaPersonas->limpiar();
                cout << "✓ Tabla limpiada correctamente.\n";
            }
            break;

        case 8:
            cout << "\n=== VALIDAR CEDULA ECUATORIANA ===\n";
            cout << "Ingrese la cedula a validar: ";
            getline(cin, cedula);
            if(validarCedulaEcuatoriana(cedula)) {
                cout << "\n✓ Cedula ecuatoriana valida.\n";
            } else {
                cout << "\n✗ Cedula invalida.\n";
            }
            break;

        case 9:
            cout << "\n¡Gracias por usar el sistema! Hasta pronto.\n";
            break;

        default:
            cout << "\n✗ Opcion invalida. Intente nuevamente.\n";
```

```
        }

        if(opcion != 9) {
            cout << "\nPresione Enter para continuar...";
            cin.get();
        }

    } while(opcion != 9);

    // Liberar memoria
    delete tablaPersonas;

    return 0;
}
```