Resolución de Tarea 9 - Algoritmos Probabilisticos y Aproximados (Fecha: 8 de Diciembre de 2025)

Universidad Simón Bolívar
Departamento de Computación y Tecnología de la Información
CI5651 - Diseño de Algoritmos I
Septiembre - Diciembre 2025
Estudiante: Junior Miguel Lara Torres (17-10303)

Tarea 9 (9 puntos)

Indice

Pregunta 1

Por resolver

Pregunta 2

Se solicita diseñar e implementar un algoritmo probabilístico de Monte Carlo para verificar si una matriz B es la inversa de una matriz A (es decir, $A \times B = I$, donde I es la matriz identidad), con una probabilidad de error acotada por ϵ.

Utilizaremos el **Algoritmo de Freivalds** adaptado. El test se basa en el principio de que, si $A \times B \neq I$, es muy probable que una multiplicación matricial-vectorial aleatoria revele la desigualdad.

- Verificar $A \times B = I$ es equivalente a verificar si la matriz de diferencia $D = A \times B - I$ es la matriz nula.
- En lugar de calcular D (lo cual es $O(n^3)$), seleccionamos un vector aleatorio X de tamaño $1 \times n$ (donde $X_i \in \{0, 1\}$). Comprobamos si $X \cdot D = \mathbf{0}$, o lo que es lo mismo: $X \cdot A \cdot B = X \cdot I$. Dado que $X \cdot I = X$, la prueba se reduce a:
- $$X \cdot (A \times B) = X$$
- La multiplicación $X \cdot A$ (vector fila por matriz) toma $O(n^2)$. Luego, $(X \cdot A) \cdot B$ (vector fila por matriz) toma otros $O(n^2)$. El test se ejecuta en $\mathbf{O(n^2)}$ por iteración.
- Si $A \times B \neq I$, la probabilidad de que la prueba falle (produzca **X** en la salida) es a lo sumo $1/2$. Para asegurar que la probabilidad de error sea menor que ϵ, debemos repetir el test k veces, donde k se define como $k = \lceil \log_2(1/\epsilon) \rceil$.

Así, el algoritmo implementado (CheckInversion) verifica la relación $A \times B = I$ utilizando la prueba de Freivalds repetida k veces. Sabiendo que multiplicando el costo por iteración por el número de iteraciones la complejidad total queda como:

$$O(n^2 \log(1/\epsilon))$$

Implementación en C++

```cpp
typedef long long MatElement;
typedef vector<vector<MatElement>> Matrix;
typedef vector<MatElement> Vector;

// Generador de números aleatorios
mt19937_64 rng(chrono::steady_clock::now().time_since_epoch().count());

/**
 * Genera un vector aleatorio X (1 x N) con elementos {0, 1}.
 */
Vector GenerateRandomVector(int N)
{
    uniform_int_distribution<MatElement> dist(0, 1);
    Vector X(N);
    for (int i = 0; i < N; ++i)
        X[i] = dist(rng);
    return X;
}

/**
 * Calcula el producto de un vector fila X por una matriz A: R = X * A.
 * R y X son vectores 1D de tamaño N.
 * Complejidad: O(N^2).
 */
Vector MatrixVectorProduct(const Vector &X, const Matrix &A, int N) {
    Vector R(N, 0);
    // Multiplicación X (1xN) * A (NxN) = R (1xN)
    for (int j = 0; j < N; ++j) { // columna de A / elemento de R
        for (int i = 0; i < N; ++i)
            R[j] += X[i] * A[i][j]; // fila de X / fila de A
    }
    return R;
}

/**
 * Algoritmo de Monte Carlo para verificar si B = A^-1 (A * B = I)
 * Retorna true si probablemente es la inversa, false si es definitivamente compuesta.
 */
bool CheckInversion(const Matrix &A, const Matrix &B, int N, double epsilon) {
    if (N == 0)
        return true;

    // Calcular K: número de iteraciones necesarias para error < epsilon
    // K = ceil(log2(1/epsilon))
    int K = max(1, (int)ceil(log2(1.0 / epsilon)));
    cout << "\n--- Ejecucion del Algoritmo de Monte Carlo ---" << endl;
    cout << "Dimension de la matriz N: " << N << endl;
    cout << "Probabilidad de error maximo (epsilon): " << scientific << setprecision(2)
    cout << "Iteraciones requeridas (K): " << K << endl;
    cout << "----------------------------------------" << endl;

    for (int k = 1; k <= K; ++k) {
        // 1. Generar vector aleatorio X
        Vector X = GenerateRandomVector(N);

        // 2. Calcular R_AB = (X * A) * B
        Vector R_A = MatrixVectorProduct(X, A, N);
        Vector R_AB = MatrixVectorProduct(R_A, B, N);

```

```
        // 3. Comparar R_AB con X (deberían ser iguales si A*B = I)
        bool match = true;
        for (int i = 0; i < N; ++i) {
            if (R_AB[i] != X[i]) {
                match = false;
                break;
            }
        }

        cout << "Iteracion " << k << "/" << K << ": Resultado: " << (match ? "COINCIDE (Probabl

        if (!match) {
            cout << "\nCONCLUSION: B NO es la inversa de A (testigo encontrado)." << en
            return false;
        }
    }

    cout << "\nCONCLUSION: B es probablemente la inversa de A." << endl;
    return true;
}
```