Laboratorio Nro. 2: Fuerza Bruta (Brute force o Exhaustive search)

Alejandro Arroyave Bedoya
Universidad Eafit
Medellín, Colombia
aarroyaveb@eafit.edu.co

Luis Javier Palacio Mesa
Universidad Eafit
Medellín, Colombia
ljpalaciom@eafit.edu.co

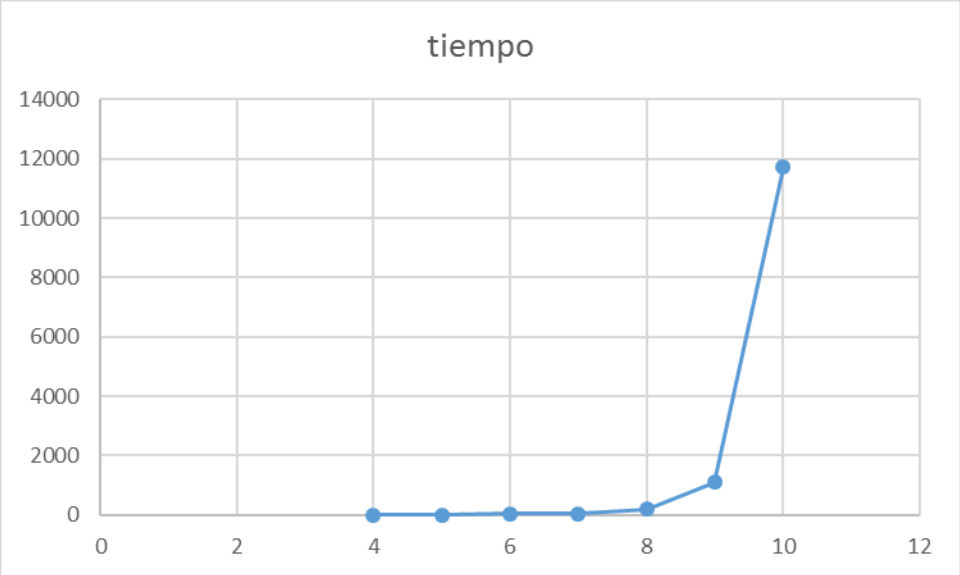
Santiago Castrillón Galvis
Universidad Eafit
Medellín, Colombia
scastrillg@eafit.edu.co

3) Simulacro de preguntas de sustentación de Proyectos

1. **Otra técnica muy utilizada para resolver este tipo de problemas es el backtracking, la cual se logra solucionar escogiendo un camino y devolviendose cuando se encuentra con un callejón sin salida, así sucesivamente hasta que encuentra un camino, lo suma a un contador o lo imprime, y sigue aplicando la técnica hasta recorrer el ultimo camino sin salida.**
2.

N° de reinas	Tiempo (en milisegundos)
4	3
5	0
6	29
7	49
8	192
9	1096
10	11718
11	Más de 50 minutos
12	Más de 50 minutos
13	Más de 50 minutos
14	Más de 50 minutos
15	Más de 50 minutos
16	Más de 50 minutos
17	Más de 50 minutos
18	Más de 50 minutos
19	Más de 50 minutos
20	Más de 50 minutos
21	Más de 50 minutos
22	Más de 50 minutos
23	Más de 50 minutos
24	Más de 50 minutos
25	Más de 50 minutos

26	Más de 50 minutos
27	Más de 50 minutos
28	Más de 50 minutos
29	Más de 50 minutos
30	Más de 50 minutos
31	Más de 50 minutos
32	Más de 50 minutos



3. **Basados en el algoritmo de "backtracking" que daba solución a las N reinas, agregamos una matriz de booleanos en la que los valores verdaderos simbolizan la existencia de casillas malas. Así pues, en la misma lectura llenamos esta matriz, y en el método recursivo "nReinas(int[] tablero, int fila)" comprobamos si la casilla es buena y podemos hacer un llamado recursivo, o de lo contrario, dejamos que siga el ciclo.**
4. **Basados en el algoritmo de "backtracking" que daba solución a las N reinas, agregamos una matriz de booleanos en la que los valores verdaderos simbolizan la existencia de casillas malas. Así pues, en la misma lectura llenamos esta matriz, y en el método recursivo "nReinas(int[] tablero, int fila)" comprobamos si la casilla es buena y podemos hacer un llamado recursivo, o de lo contrario, dejamos que siga el ciclo.**
5.

```
public class Lab2 {

    static boolean[][] esMala;
    static int cantidad;
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        int n;
        int caso = 1;
        String linea;
        n = sc.nextInt();
```

```
        while (n != 0) { // m
            esMala = new boolean[n][n];
            for (int i = 0; i < n; i++) {//m*n
                linea = sc.next();
                for (int j = 0; j < linea.length(); j++) {// m*n^2
                    if (linea.charAt(j) == '*') {
                        esMala[i][j] = true;
                    }
                }
            }
            System.out.println("Case " + caso + ": " + nReinas(n)); m * n!
            n = sc.nextInt();
            caso++;
            cantidad = 0;
        }
        System.out.println();
    }

    private static void nReinas(int[] tablero, int fila) {
        if (fila == tablero.length) { C1
            cantidad++; C2
            return; C3
        }
        for (int col = 0; col < tablero.length; col++) { C4n
            if (!esMala[fila][col]) {C5n
                if (puedoPonerReina(tablero, fila, col)) {C6n*fila
                    nReinas(tablero, fila + 1); nT(n -1)
                }
            }
        }

    }

    public static int nReinas(int n) {
        nReinas(new int[n], 0); n!
        return cantidad;
    }

    public static boolean puedoPonerReina(int[] tablero, int fila, int col) {
        tablero[fila] = col;
        for (int j = fila - 1; j >= 0; j--) {fila-1 veces
            if (tablero[j] == col || Math.abs(tablero[fila] - tablero[j]) == fila - j) {
                return false;
            }
        }
        return true;
    }
}
```

Calculo de complejidad del método nReinas(tablero,fila):

T(n) = C1 +C2 + C3 + C4n + C5n + Cn*fila + nT(n-1)
T(n) = C1 +C2 + C3 + C4n + C5n + Cn*fila + Cn!
T(n) es O(C1 +C2 + C3 + C4n + C5n + Cn*fila + Cn!)
T(n) es O(Cn!)
T(n) es O(n!)

Calculo de complejidad del método main
T(n) = m + m*n + m*n^2 + m*n!
T(n) es O(m*n!)

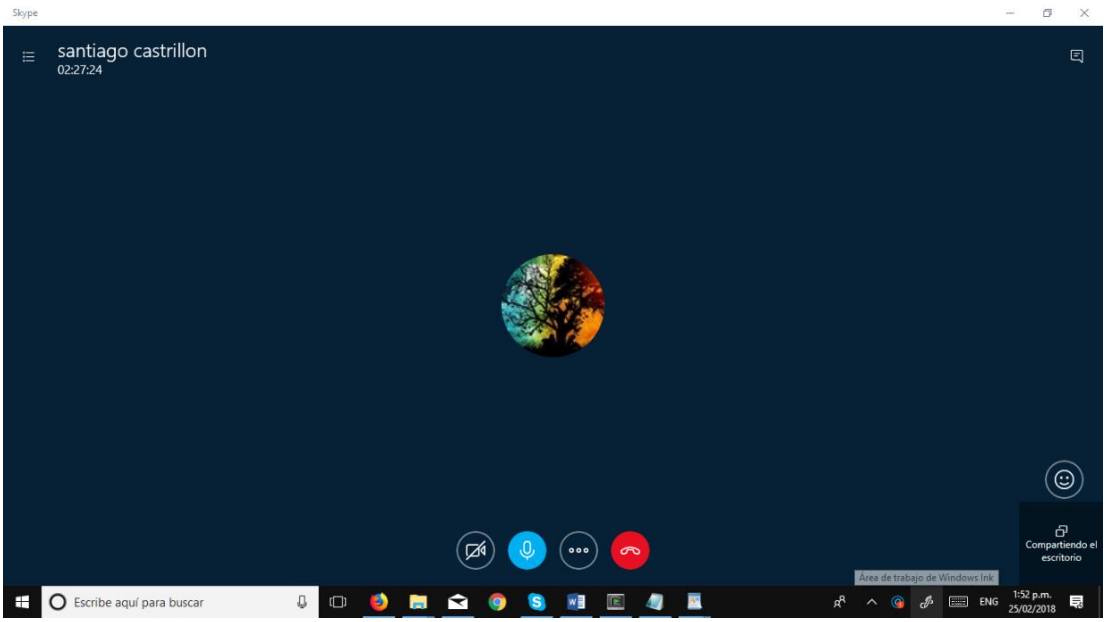
6. ***La variable m se refiere al número de casos de prueba, es decir, el número de tableros distintos que se analizan con el programa hasta que se ingrese el 0.***
La variable n se refiere al numero de filas que tiene el tablero que se analiza en el momento.
La variable fila es el iterador que denota la fila que se analiza en el momento.

4) Simulacro de Parcial

1. A)
Maximo < actual
O(n^2)
2. Ordenar(arr , k+1);
O(n!)
3. *i – (j + 1)*
return n;
*O(n*m)*

6) Trabajo en Equipo y Progreso Gradual (Opcional)

a)



DOCENTE MAURICIO TORO BERMÚDEZ
Teléfono: (+57) (4) 261 95 00 Ext. 9473. Oficina: 19 - 627
Correo: mtorobe@eafit.edu.co