Laboratorio Nro. 2: Fuerza Bruta

Isabela Muriel Roldán
Universidad Eafit
Medellín, Colombia
imurielr@eafit.edu.co

Mateo Florez Restrepo
Universidad Eafit
Medellín, Colombia
mflorezr@eafit.edu.co

3) Simulacro de preguntas de sustentación de Proyectos

1. Existen varias técnicas para darle solución al algoritmo de las nreinas, la mas usada es el de la fuerza bruta, sin embargo, hay otras mas eficientes, por ejemplo el algoritmo backtracking con depth-first search (DFS) este algoritmo tiene mejoras en cuanto al método de permutación a comparación de la fuerza bruta, este construye un árbol de búsqueda por cada fila del tablero, haciendo más fácil la detección y eliminación de aquellos lugares en el tablero donde no pueden situarse las reinas, deshaciéndose de una vez de las posiciones que no dan solución del problema. Otro método podría ser la reparación iterativa, generalmente empieza con todas las reinas en el tablero y una reina por cada columna, continuo a esto, cuenta el numero de ataques y utiliza la heurística de "ataques mínimos" para determinar cómo mejorar la ubicación de las reinas, aunque no siempre se asegura una solución al problema. También se puede usar la programación con restricciones para este problema.
2.

Valor de N	Tiempo de ejecución
4	0
5	0
6	0
7	0
8	1
9	2
10	30
11	64
12	315
13	1889
14	11244
15	80838
16	562237
17	4696319
18	Mas de 50min
19	Mas de 50min

20	**Mas de 50min**
21	**Mas de 50min**
22	**Mas de 50min**
23	**Mas de 50min**
24	**Mas de 50min**
25	**Mas de 50min**
26	**Mas de 50min**
27	**Mas de 50min**
28	**Mas de 50min**
29	**Mas de 50min**
30	**Mas de 50min**
31	**Mas de 50min**
32	**Mas de 50min**
N	**O( )**

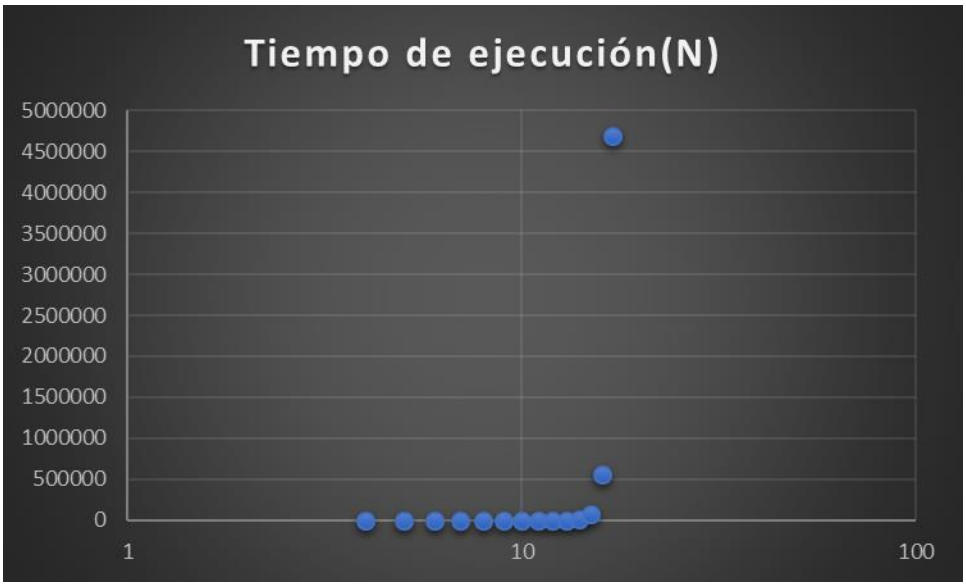

Figura 1: Grafica de los datos de la tabla N vs Tiempo de ejecución

3. Para la implementación del problema de las N reinas se verifica las posiciones en las que se atacan las reinas y se crea una matriz en la que se verifica si cierta posición del tablero es restringida (*) o no (.). En caso de que se pueda poner reina sin que ataque y la posición del tablero es un cuadro bueno se posiciona la reina, al finalizar el recorrido del tablero se aumenta el contador de jugadas posibles, esto se repite hasta que se ingrese un '0' y finalmente las respuestas, que se fueron añadidas a una lista, se imprimen dando el respectivo caso.

4. **Implementación:** El usuario ingresa un número n de reinas que se desea posicionar, y a partir de esto se crea un tablero de tamaño nxn, luego se ingresan n líneas representando el tablero, estas pueden tener cuadros buenos, que se representan con '.' y cuadros malos que se representan con '*'. Se repite este

proceso hasta que usuario ingrese un 0 representando que el programa termina, y finalmente se imprimen las respuestas obtenidas para cada caso.

5. Complejidad del algoritmo 2.1

```
private static boolean puedoPonerReina(int r, int c, int[] tablero){
      for(int i=0; i<r; ++i){                                          //C1*n
         if(tablero[i]==c||(i-r)==(tablero[i]-c)||(i-r)==(c-tablero[i])){   //C2*n
            return false;                                              //C3*n
         }
      }
      return true;                                                     //C4
   }
```

T(n)= (C1+C2+C3)*n + C4
T(n)= O(C"*n+C)
T(n)= O(C"*n)
T(n)= O(n)

```
public static void nReinas(){
      Scanner sc = new Scanner(System.in);                             //C1
      ArrayList<Integer> respuestas = new ArrayList<Integer>();        //C2
      int n;                                                           //C4
      cont = 0;                                                        //C5
      while((n = sc.nextInt()) != 0){                                  //C6*m
         boolean[][] canPlace = new boolean[n][n];                     //C7*m
         int [] tablero = new int[n];                                  //C8*m
         for(int i=0; i<n; ++i){                                       //C9* n*m
            String line = sc.next();                                   //C10*n*m
            for(int j=0;j<n;++j){                                      //C11*n^2*m
               if(line.charAt(j)=='*'){                                //C12*n^2*m
                  canPlace[i][j]=false;                                //  C13*n^2*m
               }
               else{
                  canPlace[i][j]=(true);                               //C14*n^2*m
               }
            }
         }
         respuestas.add(nReinas(0,n,tablero,canPlace));                //O(n^2)+O(n)*n
         cont = 0;                                                     //C15
      }
      for(int k = 1; k <= respuestas.size(); ++k){                     //C16*n

         System.out.println("Case "+k+": "+respuestas.get(k-1));       //C17*n
      }
   }
```

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

T(n)=
C1+C2+C3+C4+C5+C15+(C6+C7+C8)*m+(C9+C10)*n*m+(C11+C12+C13+C14)*n^2*m+(C15+C16)*n
T(n)= O(C+C*m+C*n*m+C*n^2*m+C*n)
T(n)= O(C*n^2*m)
T(n)= O(n^2*m)

```
private static int nReinas(int r, int n, int[] tablero, boolean[][] canPlace){
      for(int c=0; c<n; ++c){                                          //C1*n
         if(puedoPonerReina(r,c,tablero) && canPlace[c][r] == true){   //(C2+O(n))*n
            tablero[r] = c;                                           //C3*n
            if(r==n-1){                                               //C4*n
               ++cont;                                                //C5*n
            }
            else{
               nReinas(r+1,n,tablero,canPlace);                       //(C6+T(n-1))*n
            }
         }
      }
      return cont;                                                    //C7
   }
```

T(n)= C7+(C1+C2+C3+C4+C5+C6+O(n)+T(n-1)) *n

En el peor de los casos
T(n)= (C+T(n-1))*n+O(n)*n
T(n)= (C'+C*n)*n+O(n)*n => Ecuación de recurrencia de T(n)= C+T(n-1)=C+C*n
T(n)= O(C*n+C*n^2)+O(n)*n
T(n)= O(C*n^2)+O(n)*n
T(n)= O(n^2)+O(n)*n

6. **n y m en la complejidad**: En los cálculos de la complejidad anterior, estas variables se refieren a la cantidad de veces que el algoritmo tiene que ejecutar algún paso o acción al momento de compilarse para llegar al resultado esperado. En los ejercicios anteriores usamos en la gran mayoría solo la variable n ya que en los códigos solo se trabajaba con un solo tipo de entrada igual para cada caso. La variable m la usaríamos si hubiera más de una entrada para poder diferenciar el alcance de la complejidad de los códigos como sucede en uno de los algoritmos del 2.1 que se ejecuta según dos tipos de entrada m y n.

4) Simulacro de Parcial

1. *a) actual >maximo*
   b) O(n^2)
2. *a) arr, k+1*
   b) O(n!)
3. *1. i-m*
   2. n
   3. O(n.m)

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