Instituto Politécnico Nacional
Escuela Superior de Computo

Alumno:

- Monroy Ramírez Oscar G.

Grupo: *2CM23*

Unidad de Aprendizaje: *Algoritmos y Estructuras de Datos*

Evidencia: *Reporte Reinas que no Atacan (Backtracking)*

Docente: *De Luna Caballero Roberto*

Fecha*: 30 de noviembre de 2020*

Contenido

Introducción

Una computadora es una maquina que maneja información. El estudio de la ciencia de la computación incluye saber la forma en que se organiza la información en una computadora, como puede manejarse y la manera en que puede utilizarse esa información. Por tanto, es de suma importancia para un estudiante de la computación, entender los conceptos de organización y manejo de la información para continuar con el estudio de nuestra diciplina.

Si la ciencia de la computación es fundamental el estudio de la información, la primera pregunta que surge es: ¿qué es la información? Por desgracia, aunque ese concepto es la piedra angular de la diciplina, la interrogante anterior no puede contestar con precisión. En que sentido, el concepto de la información en esta diciplina es similar a los conceptos de punto, recta y plano para la geometría: son un grupo de términos indefinidos con los que se pueden hacer proposiciones, pero

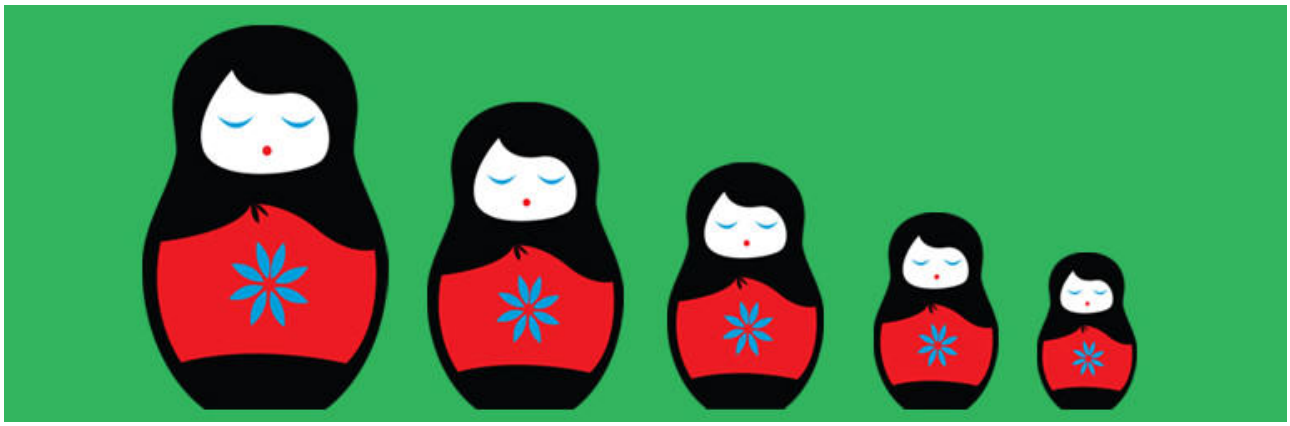
no pueden definirse con conceptos más elementales.

En la geometría es posible hablar de la longitud de una recta, a pesar de que el concepto de recta es en si mismo indefinido. La longitud de una recta es una medida cuantitativa. De manera similar, en la ciencia de la computación podemos hablar de cantidades de información. La unidad básica de información es el ***bit*** cuyo valor confirma una de dos posibilidades excluyentes. Por ejemplo, un conmutador puede estar en una de dos posiciones pero no ambas al mismo tiempo pero el hecho es que al estar en una posición representa un bit.

Pues asi es la parte estructurada de la computacion, Que nos permite encontrar diversas formas métodos, con los cual es podemos avanzar en diversas partes yasea para usarlas como herramientas que funcionen como grandes precursores en todas las operaciones básicas que realizamos a la hora de realizar diferentes operaciones en nuestros propios programas, siendo así la estructura de datos una manera de construir un mapa para no perderse a la hora de El manejo de esa intensa y extensa información, métodos que veremos a lo largo del desarrollo de este trabajo.

Marco Teórico

Recursividad



Es una técnica utilizada en programación que nos permite que un bloque de instrucciones se ejecute un cierto número de veces (el que nosotros determinemos). A veces es algo complicado de entender, pero no os preocupéis. Cuando veamos los ejemplos estará clarísimo. En Java, como en otros muchos lenguajes, los métodos pueden llamarse a sí mismos. Gracias a esto, podemos utilizar a nuestro favor la recursividad en lugar de la iteración para resolver determinados tipos de problemas. **[7]:**

Ejemplo sencillo

Vamos a ver un pequeño ejemplo que no hace absolutamente nada. Es un método cuyo único objetivo es llamarse a sí mismo: **[7]:**

```
void cuentaRegresiva () {
      cuentaRegresiva();
}
```

Si ejecutáis esto, os va a dar un error en la pila (mítico *StackOverflow Error*, Biblia de los programadores).

Como podemos ver, se ha llamado al método **cuentaRegresiva** porque vamos a mostrar por pantalla la cuenta atrás de un número que nosotros pasemos como parámetro a la función. Por ejemplo, para hacer la cuenta atrás de 10 sin recursividad, haríamos: **[7]:**

```
for( int i = 10; i >= 0; i--) {
	System.out.println(i);
}
```

Ahora, para hacerlo de manera recursiva, tendríamos que pasar como parámetro un número. Además, tras imprimir ese número, llamaremos a la misma función con el número actual restando uno: **[7]:**

```
void cuentaRegresiva(int numero) {
	System.out.println(numero);
	cuentaRegresiva(numero - 1);
}
```

Es lo que os he comentado arriba. Llamamos a la función con un 10. Imprimimos el 10 y llamamos a la función con un 9. Imprimimos el 9 y llamamos a la función con un 8. Así hasta el fin de los días. Digo hasta el fin de los días porque os va a saltar error si ejecutáis esto así directamente: **[7]:**

```
public class Recursividad {

    static void cuentaRegresiva(int numero) {
        System.out.println(numero);
        cuentaRegresiva(numero - 1);
    }

    public static void main(String[] args) {
        cuentaRegresiva(10);
    }

}
```

Problema: llamada infinita. Para ello, lo que tenemos que hacer es que cuando el número sea 0, deje de llamar a la función. Para eso, metemos una estructura condicional de toda la vida. Gracias al condicional, dejará de ejecutarse a partir de 0: **[7]:**

```
void cuentaRegresiva(int numero) {
	System.out.println(numero);
	if(numero > 0) {
		cuentaRegresiva(numero - 1);
	}
}
```

Factorial

Calcular el factorial de un número con recursividad es el típico ejemplo para explicar este método de programación. Recordad que el factorial de un número es multiplicar dicho número por todos sus anteriores hasta llegar a 1. Se representa con una exclamación. **[7]:**

Por ejemplo:

5! = 5*4*3*2*1 = 120 El recorrer los números hacia atrás ya lo tenemos hecho. Ahora lo que queda es multiplicar ese número por su anterior, y así sucesivamente:

```
int factorial(int n) {
	if (n == 0) {
		return 1;
	} else {
		return n * factorial(n - 1);
	}
```

Backtracking

En su forma básica, la idea de backtracking se asemeja a un recorrido en profundidad dentro de un grafo dirigido. El grafo en cuestión suele ser un árbol, o por lo menos no contiene ciclos. Sea cual sea su estructura, existe sólo implícitamente. El objetivo del recorrido es encontrar soluciones para algún problema. Esto se consigue construyendo soluciones parciales a medida que progresa el recorrido; estas soluciones parciales limitan las regiones en las que se puede encontrar una solución completa. El recorrido tiene éxito si, procediendo de esta forma, se puede definir por completo una solución. En este caso el algoritmo puede bien detenerse (si lo único que se necesita es una solución del problema) o bien seguir buscando soluciones alternativas (si deseamos examinarlas todas). Por otra parte, el recorrido no tiene éxito si en alguna etapa la solución parcial construida hasta el momento no se puede completar. En tal caso, el recorrido vuelve atrás exactamente igual que en un recorrido en profundidad, eliminando

sobre la marcha los elementos que se hubieran añadido en cada fase. Cuando vuelve a un nodo que tiene uno o más vecinos sin explorar, prosigue el recorrido de una solución. **[8]**

```
                        Algoritmo de Backtracking
                        proc Backtracking (↕X[1 . . . i ]:
TSolución, ↑ok: B)
                        variables L: ListaComponentes
                        inicio
                            si EsSolución (X) entonces ok   CIERTO
                            en otro caso
                                 ok   FALSO
                                 L=Candidatos (X)
                                 mientras ¬ok ^ ¬Vacía (L) hacer
                                 X[i + 1]   Cabeza (L); L   Resto
(L)
                                 Backtracking (X, ok)
                                 finmientras
                            finsi
                        fin
```

Podemos visualizar el funcionamiento de una técnica de backtracking como la exploración en profundidad de un grafo. **[8]**

Cada vértice del grafo es un posible estado de la solución del problema. Cada arco del grafo representa la transición entre dos estados de la solución (i.e., la toma de una decisión). **[8]**

Típicamente el tamaño de este grafo será inmenso, por lo que no existirá de manera explícita. En cada momento sólo tenemos en una estructura los nodos que van desde el estado inicial al estado actual. Si cada secuencia de decisiones distinta da lugar a un estado diferente, el grafo es un árbol (el árbol de estados). **[8]**

Enfoque

Los problemas que deben satisfacer un determinado tipo de restricciones son problemas completos, donde el orden de los elementos de la solución no importa. Estos problemas consisten en un conjunto (o lista) de variables a la que a cada una se le debe asignar un valor sujeto a las restricciones del problema. La técnica va creando todas las posibles combinaciones de elementos para obtener una solución. Su principal virtud es que en la mayoría de las implementaciones se puede evitar combinaciones, estableciendo funciones de acotación (o poda) reduciendo el tiempo de ejecución. **[8]**

Vuelta atrás está muy relacionado con la Búsqueda combinatoria.

Diseño e implementación

Esencialmente, la idea es encontrar la mejor combinación posible en un momento determinado, por eso, se dice que este tipo de algoritmo es una Búsqueda en profundidad. Durante la búsqueda, si se encuentra una alternativa incorrecta, la búsqueda retrocede hasta el paso anterior y toma la siguiente alternativa. Cuando se han terminado las posibilidades, se vuelve a la elección anterior y se toma la siguiente opción (hijo [si nos referimos a un árbol]). Si no hay más alternativas la búsqueda falla. De esta manera, se crea un árbol implícito, en el que cada nodo es un estado de la solución (solución parcial en el caso de nodos interiores o solución total en el caso de los nodos hoja). **[8]**

Normalmente, se suele implementar este tipo de algoritmos como un procedimiento recursivo. Así, en cada llamada al procedimiento se toma una variable y se le asignan todos los valores posibles, llamando a su vez al procedimiento para cada uno de los nuevos estados. La diferencia con la Búsqueda en profundidad es que se suelen diseñar funciones de cota, de forma que no se generen algunos estados si no van a conducir a ninguna solución, o a una solución peor de la que ya se tiene. De esta forma se ahorra espacio en memoria y tiempo de ejecución. **[8]**

Heurísticas

Algunas heurísticas son comúnmente usadas para acelerar el proceso. Como las variables se pueden procesar en cualquier orden, generalmente es más eficiente intentar ser lo más restrictivo posible con las primeras (esto es, las primeras con menores valores posibles). Este proceso poda el árbol de búsqueda antes de que se tome la decisión y se llame a la subrutina recursiva. **[8]**

Cuando se elige qué valor se va a asignar, muchas implementaciones hacen un examen hacia delante (FC, Forward Checking), para ver qué valor restringirá el menor número posible de valores, de forma que se anticipa en a) preservar una posible solución y b) hace que la solución encontrada no tenga restricciones destacadas. **[8]**

Algunas implementaciones muy sofisticadas usan una función de cotas, que examina si es posible encontrar una solución a partir de una solución parcial. Además, se comprueba si la solución parcial que falla puede incrementar significativamente la eficiencia del algoritmo. Por el uso de estas funciones de cota, se debe ser muy minucioso en su implementación de forma que sean poco costosas computacionalmente hablando, ya que lo más normal es que se ejecuten en para cada nodo o paso del algoritmo. Cabe destacar, que las cotas eficaces se crean de forma parecida a las funciones Heurísticas, esto es, relajando las restricciones para conseguir mayor eficiencia. **[8]**

Con el objetivo de mantener la solución actual con coste mínimo, los algoritmos vuelta atrás mantienen el coste de la mejor solución en una variable que va variando con cada nueva mejor solución encontrada. Así, si una solución es peor que la que se acaba de encontrar, el algoritmo no actualizará la solución. De esta

forma, devolverá siempre la mejor solución que haya encontrado. **[8]**

Ejemplos de aplicación de backtracking

SATISFABILITY

Inicialmente A contiene la expresión booleana que constituye el problema.

Elegir subproblema de A, p.ejemplo : (x+y+z)(x'+y)(y'+z)(z'+x)(x'+y'+z').

Elegir una cláusula con mínimo número de literales.

Elegir una variable x, y, z,... dentro de la cláusula y crear 2 subproblemas reemplazando x=V y x=F.

En el caso x=V

```
Omitir las cláusulas donde aparece x.
Omitir x' en las cláusulas que aparece x'.
```

En el caso x=F

```
Omitir las cláusulas donde aparece x'.
Omitir x en las cláusulas que aparece x.
```

Test

Si no quedan cláusulas. STOP. (solución encontrada).

Si hay una cláusula vacía. DROP.

En otro caso añadir a A

Nota: Observemos que si encontramos a A vacío entonces la expresión booleana no puede ser satisfecha.

HAMILTON CYCLE (VIAJANTE DE COMERCIO)

En este caso los subproblemas S son caminos que parten de a y llegan a b a través de un sucesión de nodos T. (b es el mismo a lo largo de todo el algoritmo).

Inicialmente A contiene solamente el camino (a, vacío, b).

Elegimos un subproblema S cualquiera de A (y lo borramos de A) y añadimos ramas (c, a) del grafo (las c's son las adyacentes de a) . Estos caminos extendidos son los hijos. Ahora cada c juega el rol de a.

Examinamos c/ hijo:

Test:

1)Si G-T forma un camino hamiltoniano STOP (solución hallada)

2)Si G-T tiene un nodo de grado uno (excepto a y b) o si G-T-{a, b} es disconexo entonces DROP este subproblema .

3) Si 1) y 2) fallan add subproblema en A.

EXACT COVER

Dado un conjunto finito U y una familia se subconjuntos {Tj} de U definimos una matriz A donde cada fila se corresponde con un elemento ui de U y cada columna de A con un subconjunto Tj . Ponemos aij=1 si ui∈U pertenece a Tj y aij=0 en caso contrario. Interpretamos que xj=1 significa que elegimos Tj y 0 en caso contrario. **[8]**

Se trata de averiguar si es factible Ax=1 donde A y x son binarias y las componentes de 1 son unos. **[8]**

S0= un vector de ceros (raíz del árbol)

Cada nodo S del árbol es una sucesión x cuyas primeras k componentes le han sido asignados un 1 o un 0 y el resto de componentes son ceros. Reemplazamos S por 2 subproblemas Si (i=1,2) poniendo xk+1 =1 y xk+1=0 respectivamente. **[8]**

Test

if Ax=1 STOP

if Ax>1 DROP Si

if Ax<1 add Si to A

Ejemplos de problemas comunes resueltos usando Vuelta Atrás

Problema de las N Reinas

Disponemos de un tablero de ajedrez de tamaño NxN, y se trata de colocar en él N reinas de manera que no se amenacen según las normas del ajedrez.

```
proc NReinas (↕[1 . . . i ]: TSolución, ↓N: N, ↑ok: B)
variables j : N
inicio
    si i=N entonces ok=CIERTO
    en otro caso
        ok=FALSO
        j=1
        mientras ¬ok ^ (j≤N) hacer
            si EsFactible (R, j) entonces
                R[i + 1]= j
            NReinas (R, N, ok)
```

```
            finsi
            j=j+1
        finmientras
    finsi
fin
func EsFactible (↓R[1 . . . i ]: TSolución, ↓j : N): B
variables factible: B
inicio
    factible=CIERTO
    k=1
        mientras factible ^ (k≤i) hacer
            si (j=R[k])\/(i+1-k= |j-R[k]|) entonces
            factible=FALSO
            finsi
            k=k+1
        finmientras
    devolver factible
 fin
```

Problema de la mochila 0,1

Dados n elementos e1,e2,...,en con pesos p1,p2,...,pn y beneficios b1,b2,...,bn, y dada una mochila capaz de albergar hasta un máximo de peso M (capacidad de la mochila), queremos encontrar cuáles de los n elementos hemos de introducir en la mochila de forma que la suma de los beneficios de los elementos escogidos sea máxima, sujeto a la restricción de que tales elementos no pueden superar la capacidad de la mochila.

Problema del laberinto

Se tiene una matriz bidimensional de nxn casillas para representar un laberinto cuadrado. Cada casilla está marcada como visitada o no visitada. Se debe ir desde la casilla (1,1) a la (n, n) haciendo movimientos horizontales y verticales.

Backtracking para la enumeración

El problema de la enumeración consiste en encontrar todas las soluciones del problema, es por ello que tendremos que recorrer el árbol de estados al completo.

```
Algoritmo de Backtracking para la enumeración:
proc Bactracking Enum(↕X[1 . . . i ]: TSolución, ↑num: N)
```

```
variables L: ListaComponentes
inicio
      si EsSolución (X) entonces num   num+1
           EscribeSolución (X)
      en otro caso
           L   Candidatos (X)
           mientras ¬Vacía (L) hacer
                X[i + 1]   Cabeza (L); L   Resto (L)
                BacktrackingEnum (X, num)
           finmientras
      finsi
fin
```

Ejemplo del Programa de las Reinas que no Atacan

```
#include <graphics.h>
#include <stdio.h>
#include <windows.h>
#include <stdlib.h>

#define SIZE 10
#define num 0
void Imprimir(int (*matriz)[SIZE],int n);
void ColocarReina(int (*matriz)[SIZE],int c,int f);
void QuitarReina(int (*matriz)[SIZE],int c,int f);
int ValidaAtaque(int (*matriz)[SIZE],int c,int f,int tam);
void Reinas(int (*matriz)[SIZE],int c,int f,int tam);
int size(int (*matriz)[SIZE],int tam);
//Contador para realizar la recursividad
int numerodereinas=num;

//Funcion principal
int main()
{
     int n=8;
```

```
 printf("Inserte el numero de reinas que desea colocar:");
 scanf("%d",&n);
 int matriz[SIZE][SIZE];
 int i,j,tam;
 tam=n;
 //Se llena la matriz de ceros
 for(i=0;i<n;i++)
 {
      for(j=0;j<n;j++)
      {
           matriz[i][j]=0;
      }
 }

 //Se llama a la recursividad

 initwindow(1080,700);
 setviewport(0,0,1080,700,1);
 setbkcolor(RGB(0, 0, 0));
 cleardevice();
 settextstyle(3,HORIZ_DIR,4);

 setfillstyle(SOLID_FILL,WHITE);
 setcolor(WHITE);
rectangle(50,100,100,150);
floodfill(75,125,WHITE);

setcolor(RED);
setfillstyle(SOLID_FILL,RED);
rectangle(100,100,150,150);
floodfill(125,125,RED);
```

```
setfillstyle(SOLID_FILL,WHITE);
setcolor(WHITE);
rectangle(150,100,200,150);
floodfill(175,125,WHITE);

setcolor(RED);
setfillstyle(SOLID_FILL,RED);
rectangle(200,100,250,150);
floodfill(225,125,RED);

setfillstyle(SOLID_FILL,WHITE);
setcolor(WHITE);
rectangle(250,100,300,150);
floodfill(275,125,WHITE);

setcolor(RED);
setfillstyle(SOLID_FILL,RED);
rectangle(300,100,350,150);
floodfill(325,125,RED);

setfillstyle(SOLID_FILL,WHITE);
setcolor(WHITE);
rectangle(350,100,400,150);
floodfill(375,125,WHITE);

setcolor(RED);
setfillstyle(SOLID_FILL,RED);
rectangle(400,100,450,150);
floodfill(425,125,RED);

setcolor(RED);
setfillstyle(SOLID_FILL,RED);
rectangle(50,150,100,200);
```

```
floodfill(75,175,RED);

setcolor(WHITE);
setfillstyle(SOLID_FILL,WHITE);
rectangle(100,150,150,200);
floodfill(125,175,WHITE);

setcolor(RED);
setfillstyle(SOLID_FILL,RED);
rectangle(150,150,200,200);
floodfill(175,175,RED);

setcolor(WHITE);
setfillstyle(SOLID_FILL,WHITE);
rectangle(200,150,250,200);
floodfill(225,175,WHITE);

setcolor(RED);
setfillstyle(SOLID_FILL,RED);
rectangle(250,150,300,200);
floodfill(275,175,RED);

setcolor(WHITE);
setfillstyle(SOLID_FILL,WHITE);
rectangle(300,150,350,200);
floodfill(325,175,WHITE);

setcolor(RED);
setfillstyle(SOLID_FILL,RED);
rectangle(350,150,400,200);
floodfill(375,175,RED);

setcolor(WHITE);
```

```
setfillstyle(SOLID_FILL,WHITE);
rectangle(400,150,450,200);
floodfill(425,175,WHITE);



setfillstyle(SOLID_FILL,WHITE);
 setcolor(WHITE);
rectangle(50,200,100,250);
floodfill(75,225,WHITE);

setcolor(RED);
setfillstyle(SOLID_FILL,RED);
rectangle(100,200,150,250);
floodfill(125,225,RED);

setfillstyle(SOLID_FILL,WHITE);
setcolor(WHITE);
rectangle(150,200,200,250);
floodfill(175,225,WHITE);

setcolor(RED);
setfillstyle(SOLID_FILL,RED);
rectangle(200,200,250,250);
floodfill(225,225,RED);

setfillstyle(SOLID_FILL,WHITE);
setcolor(WHITE);
rectangle(250,200,300,250);
floodfill(275,225,WHITE);

setcolor(RED);
setfillstyle(SOLID_FILL,RED);
```

```
rectangle(300,200,350,250);
floodfill(325,225,RED);

setfillstyle(SOLID_FILL,WHITE);
setcolor(WHITE);
rectangle(350,200,400,250);
floodfill(375,225,WHITE);

setcolor(RED);
setfillstyle(SOLID_FILL,RED);
rectangle(400,200,450,250);
floodfill(425,225,RED);



setcolor(RED);
setfillstyle(SOLID_FILL,RED);
rectangle(50,250,100,300);
floodfill(75,275,RED);

setcolor(WHITE);
setfillstyle(SOLID_FILL,WHITE);
rectangle(100,250,150,300);
floodfill(125,275,WHITE);

setcolor(RED);
setfillstyle(SOLID_FILL,RED);
rectangle(150,250,200,300);
floodfill(175,275,RED);

setcolor(WHITE);
setfillstyle(SOLID_FILL,WHITE);
rectangle(200,250,250,300);
floodfill(225,275,WHITE);
```

```
setcolor(RED);
setfillstyle(SOLID_FILL,RED);
rectangle(250,250,300,300);
floodfill(275,275,RED);

setcolor(WHITE);
setfillstyle(SOLID_FILL,WHITE);
rectangle(300,250,350,300);
floodfill(325,275,WHITE);

setcolor(RED);
setfillstyle(SOLID_FILL,RED);
rectangle(350,250,400,300);
floodfill(375,275,RED);

setcolor(WHITE);
setfillstyle(SOLID_FILL,WHITE);
rectangle(400,250,450,300);
floodfill(425,275,WHITE);



setfillstyle(SOLID_FILL,WHITE);
 setcolor(WHITE);
rectangle(50,300,100,350);
floodfill(75,325,WHITE);

setcolor(RED);
setfillstyle(SOLID_FILL,RED);
rectangle(100,300,150,350);
floodfill(125,325,RED);
```

```
setfillstyle(SOLID_FILL,WHITE);
setcolor(WHITE);
rectangle(150,300,200,350);
floodfill(175,325,WHITE);

setcolor(RED);
setfillstyle(SOLID_FILL,RED);
rectangle(200,300,250,350);
floodfill(225,325,RED);

setfillstyle(SOLID_FILL,WHITE);
setcolor(WHITE);
rectangle(250,300,300,350);
floodfill(275,325,WHITE);

setcolor(RED);
setfillstyle(SOLID_FILL,RED);
rectangle(300,300,350,350);
floodfill(325,325,RED);

setfillstyle(SOLID_FILL,WHITE);
setcolor(WHITE);
rectangle(350,300,400,350);
floodfill(375,325,WHITE);

setcolor(RED);
setfillstyle(SOLID_FILL,RED);
rectangle(400,300,450,350);
floodfill(425,325,RED);


setcolor(RED);
setfillstyle(SOLID_FILL,RED);
```

```
rectangle(50,350,400,400);
floodfill(75,375,RED);

setcolor(WHITE);
setfillstyle(SOLID_FILL,WHITE);
rectangle(100,350,150,400);
floodfill(125,375,WHITE);

setcolor(RED);
setfillstyle(SOLID_FILL,RED);
rectangle(150,350,200,400);
floodfill(175,375,RED);

setcolor(WHITE);
setfillstyle(SOLID_FILL,WHITE);
rectangle(200,350,250,400);
floodfill(225,375,WHITE);

setcolor(RED);
setfillstyle(SOLID_FILL,RED);
rectangle(250,350,300,400);
floodfill(275,375,RED);

setcolor(WHITE);
setfillstyle(SOLID_FILL,WHITE);
rectangle(300,350,350,400);
floodfill(325,375,WHITE);

setcolor(RED);
setfillstyle(SOLID_FILL,RED);
rectangle(350,350,400,400);
floodfill(375,375,RED);
```

```
setcolor(WHITE);
setfillstyle(SOLID_FILL,WHITE);
rectangle(400,350,450,400);
floodfill(425,375,WHITE);



setfillstyle(SOLID_FILL,WHITE);
 setcolor(WHITE);
rectangle(50,400,100,450);
floodfill(75,425,WHITE);

setcolor(RED);
setfillstyle(SOLID_FILL,RED);
rectangle(100,400,150,450);
floodfill(125,425,RED);

setfillstyle(SOLID_FILL,WHITE);
setcolor(WHITE);
rectangle(150,400,200,450);
floodfill(175,425,WHITE);

setcolor(RED);
setfillstyle(SOLID_FILL,RED);
rectangle(200,400,250,450);
floodfill(225,425,RED);

setfillstyle(SOLID_FILL,WHITE);
setcolor(WHITE);
rectangle(250,400,300,450);
floodfill(275,425,WHITE);

setcolor(RED);
```

```
setfillstyle(SOLID_FILL,RED);
rectangle(300,400,350,450);
floodfill(325,425,RED);

setfillstyle(SOLID_FILL,WHITE);
setcolor(WHITE);
rectangle(350,400,400,450);
floodfill(375,425,WHITE);

setcolor(RED);
setfillstyle(SOLID_FILL,RED);
rectangle(400,400,450,450);
floodfill(425,425,RED);



setcolor(RED);
setfillstyle(SOLID_FILL,RED);
rectangle(50,450,100,500);
floodfill(75,475,RED);

setcolor(WHITE);
setfillstyle(SOLID_FILL,WHITE);
rectangle(100,450,150,500);
floodfill(125,475,WHITE);

setcolor(RED);
setfillstyle(SOLID_FILL,RED);
rectangle(150,450,200,500);
floodfill(175,475,RED);

setcolor(WHITE);
setfillstyle(SOLID_FILL,WHITE);
rectangle(200,450,250,500);
```

```
    floodfill(225,475,WHITE);

    setcolor(RED);
    setfillstyle(SOLID_FILL,RED);
    rectangle(250,450,300,500);
    floodfill(275,475,RED);

    setcolor(WHITE);
    setfillstyle(SOLID_FILL,WHITE);
    rectangle(300,450,350,500);
    floodfill(325,475,WHITE);

    setcolor(RED);
    setfillstyle(SOLID_FILL,RED);
    rectangle(350,450,400,500);
    floodfill(375,475,RED);

    setcolor(WHITE);
    setfillstyle(SOLID_FILL,WHITE);
    rectangle(400,450,450,500);
    floodfill(425,475,WHITE);
     Reinas(matriz,0,0,tam);

     closegraph();
     return 0;
}
//Se utiliza para saber el numero de reinas que hay en la matriz
int size(int (*matriz)[SIZE],int tam)
{
     int i,j;
     int contador=0;
     for(i=0;i<tam;i++)
     {
```

```
            for(j=0;j<tam;j++)
            {
                  if((matriz[i][j])==1)
                  {
                        contador++;
                  }
            }
      }
      return contador;
}

//Funcion Recursiva que recibe la matriz y la columna donde se va
a comenzar
void Reinas(int (*matriz)[SIZE],int c,int f,int tam)
{

      //Se coloca la reina en la posicion dada
      ColocarReina(matriz,c,f);
      numerodereinas=size(matriz,tam);
      //Imprime el tablero
      Imprimir(matriz,tam);

      system("cls");

      printf("\n");
      //Si ya no hay reinas por colocar se finaliza la recursividad
      if(tam==numerodereinas && (ValidaAtaque(matriz,c,f,tam))==0)
      {
            printf("El resultado final es:\n");
            Imprimir(matriz,tam);
            Sleep(9999);
            exit(0);
      }
```

```
     //Si es atacada entonces llama a la recursividad del espacio
que esta a la derecha
     if((ValidaAtaque(matriz,c,f,tam))==0 && c<=tam-1 && f<=tam-1)
     {


          Reinas(matriz,0,f+1,tam);


          //Si regresa es por que no pudo colocar todas la reinas
entonces se debe de quitar con la que se esta probando
          QuitarReina(matriz,c,f);
          //Despues lo intenta pero cambiando a la siguiente fila
          Reinas(matriz,c+1,f,tam);


          return;
     }
     //Si es que esta siendo atacada revisa colocando la reina en
otra posicion de la misma columna
     else if((ValidaAtaque(matriz,c,f,tam))==1 && c<tam-1 &&
f<tam-1)
     {
          QuitarReina(matriz,c,f);
          //Despues de quitar a la reina llama a la recursividad
otra vez.
          Reinas(matriz,c+1,f,tam);
          return;
     }
     //Si esta en algun borde de la matriz revisa si puede seguir
aumentando la fila
     else if(c==tam-1 || f==tam-1)
     {
          if((ValidaAtaque(matriz,c,f,tam))==1)
          {
               QuitarReina(matriz,c,f);
               if((c+1)<=(tam-1))
               {
```

```
                    Reinas(matriz,c+1,f,tam);
               }

               return;
          }

     }

}
//Funcion que revisa todos los posibles casos en que puede ser atacada una reina por el lado izquierdo
int ValidaAtaque(int (*matriz)[SIZE],int c,int f,int tam)
{
     int i=0,j=0;
     for(i=c-1;i>=0;i--)
     {
          if(matriz[i][f]==1)
               return 1;
     }
     for(i=f-1;i>=0;i--)
     {
          if(matriz[c][i]==1)
               return 1;
     }
     i=1;
     while(c-i>=0 && f-i>=0)
     {
          if(matriz[c-i][f-i]==1)
          {
               return 1;
          }
          i++;
     }
```

```
     i=1;
     while(c+i<=tam-1 && f-i>=0)
     {
          if(matriz[c+i][f-i]==1)
          {
               return 1;
          }
          i++;
     }
     return 0;
}
//Funcion que coloca una reina en la posicion que recibe
void ColocarReina(int (*matriz)[SIZE],int c, int f)
{
     matriz[c][f]=1;
     return;
}
//Funcion que elimina una reina en la posicion que recibe
void QuitarReina(int (*matriz)[SIZE],int c, int f)
{
     matriz[c][f]=0;
     int i=0,j=0,x=50,y=100;
     i=c;
     j=f;
     //llenar el cuadro
     if((c==0&&f==0)||(c==0&&f==2)||(c==0&&f==4)||(c==0&&f==6)){
          setcolor(WHITE);
          setcolor(WHITE);
         setfillstyle(SOLID_FILL,WHITE);
          line
(x+5+(50*i+1),y+45+(50*j+1),x+45+(50*i+1),y+45+(50*j+1));
          line
(x+45+(50*i+1),y+45+(50*j+1),x+45+(50*i+1),y+30+(50*j+1));
          line
```

```
(x+45+(50*i+1),y+30+(50*j+1),x+40+(50*i+1),y+20+(50*j+1));
          line
(x+40+(50*i+1),y+20+(50*j+1),x+35+(50*i+1),y+30+(50*j+1));
         line
(x+35+(50*i+1),y+30+(50*j+1),x+25+(50*i+1),y+5+(50*j+1));
         line
(x+25+(50*i+1),y+5+(50*j+1),x+15+(50*i+1),y+30+(50*j+1));
          line
(x+15+(50*i+1),y+30+(50*j+1),x+10+(50*i+1),y+20+(50*j+1));
          line
(x+10+(50*i+1),y+20+(50*j+1),x+5+(50*i+1),y+30+(50*j+1));
          line
(x+5+(50*i+1),y+30+(50*j+1),x+5+(50*i+1),y+45+(50*j+1));
          floodfill((x+25+(50*i+1)),(y+25+(50*j+1)),WHITE);


     }
     else
if((c==2&&f==0)||(c==2&&f==2)||(c==2&&f==4)||(c==2&&f==6)){
          setcolor(WHITE);
          setcolor(WHITE);
         setfillstyle(SOLID_FILL,WHITE);
         line
(x+5+(50*i+1),y+45+(50*j+1),x+45+(50*i+1),y+45+(50*j+1));
          line
(x+45+(50*i+1),y+45+(50*j+1),x+45+(50*i+1),y+30+(50*j+1));
          line
(x+45+(50*i+1),y+30+(50*j+1),x+40+(50*i+1),y+20+(50*j+1));
          line
(x+40+(50*i+1),y+20+(50*j+1),x+35+(50*i+1),y+30+(50*j+1));
         line
(x+35+(50*i+1),y+30+(50*j+1),x+25+(50*i+1),y+5+(50*j+1));
         line
(x+25+(50*i+1),y+5+(50*j+1),x+15+(50*i+1),y+30+(50*j+1));
          line
(x+15+(50*i+1),y+30+(50*j+1),x+10+(50*i+1),y+20+(50*j+1));
          line
(x+10+(50*i+1),y+20+(50*j+1),x+5+(50*i+1),y+30+(50*j+1));
          line
```

```
(x+5+(50*i+1),y+30+(50*j+1),x+5+(50*i+1),y+45+(50*j+1));
          floodfill((x+25+(50*i+1)),(y+25+(50*j+1)),WHITE);


     }
     else
if((c==4&&f==0)||(c==4&&f==2)||(c==4&&f==4)||(c==4&&f==6)){
          setcolor(WHITE);
          setcolor(WHITE);
         setfillstyle(SOLID_FILL,WHITE);
         line
(x+5+(50*i+1),y+45+(50*j+1),x+45+(50*i+1),y+45+(50*j+1));
          line
(x+45+(50*i+1),y+45+(50*j+1),x+45+(50*i+1),y+30+(50*j+1));
          line
(x+45+(50*i+1),y+30+(50*j+1),x+40+(50*i+1),y+20+(50*j+1));
          line
(x+40+(50*i+1),y+20+(50*j+1),x+35+(50*i+1),y+30+(50*j+1));
         line
(x+35+(50*i+1),y+30+(50*j+1),x+25+(50*i+1),y+5+(50*j+1));
         line
(x+25+(50*i+1),y+5+(50*j+1),x+15+(50*i+1),y+30+(50*j+1));
          line
(x+15+(50*i+1),y+30+(50*j+1),x+10+(50*i+1),y+20+(50*j+1));
          line
(x+10+(50*i+1),y+20+(50*j+1),x+5+(50*i+1),y+30+(50*j+1));
          line
(x+5+(50*i+1),y+30+(50*j+1),x+5+(50*i+1),y+45+(50*j+1));
          floodfill((x+25+(50*i+1)),(y+25+(50*j+1)),WHITE);


     }
     else
if((c==6&&f==0)||(c==6&&f==2)||(c==6&&f==4)||(c==6&&f==6)){
          setcolor(WHITE);
          setcolor(WHITE);
         setfillstyle(SOLID_FILL,WHITE);
         line
(x+5+(50*i+1),y+45+(50*j+1),x+45+(50*i+1),y+45+(50*j+1));
```

```
          line
(x+45+(50*i+1),y+45+(50*j+1),x+45+(50*i+1),y+30+(50*j+1));
          line
(x+45+(50*i+1),y+30+(50*j+1),x+40+(50*i+1),y+20+(50*j+1));
          line
(x+40+(50*i+1),y+20+(50*j+1),x+35+(50*i+1),y+30+(50*j+1));
         line
(x+35+(50*i+1),y+30+(50*j+1),x+25+(50*i+1),y+5+(50*j+1));
         line
(x+25+(50*i+1),y+5+(50*j+1),x+15+(50*i+1),y+30+(50*j+1));
          line
(x+15+(50*i+1),y+30+(50*j+1),x+10+(50*i+1),y+20+(50*j+1));
          line
(x+10+(50*i+1),y+20+(50*j+1),x+5+(50*i+1),y+30+(50*j+1));
          line
(x+5+(50*i+1),y+30+(50*j+1),x+5+(50*i+1),y+45+(50*j+1));
          floodfill((x+25+(50*i+1)),(y+25+(50*j+1)),WHITE);


     }
     else if
((c==1&&f==3)||(c==1&&f==5)||(c==1&&f==7)||(c==1&&f==1)){
          setcolor(WHITE);
          setcolor(WHITE);
         setfillstyle(SOLID_FILL,WHITE);
         line
(x+5+(50*i+1),y+45+(50*j+1),x+45+(50*i+1),y+45+(50*j+1));
          line
(x+45+(50*i+1),y+45+(50*j+1),x+45+(50*i+1),y+30+(50*j+1));
          line
(x+45+(50*i+1),y+30+(50*j+1),x+40+(50*i+1),y+20+(50*j+1));
          line
(x+40+(50*i+1),y+20+(50*j+1),x+35+(50*i+1),y+30+(50*j+1));
         line
(x+35+(50*i+1),y+30+(50*j+1),x+25+(50*i+1),y+5+(50*j+1));
         line
(x+25+(50*i+1),y+5+(50*j+1),x+15+(50*i+1),y+30+(50*j+1));
          line
(x+15+(50*i+1),y+30+(50*j+1),x+10+(50*i+1),y+20+(50*j+1));
```

```
          line
(x+10+(50*i+1),y+20+(50*j+1),x+5+(50*i+1),y+30+(50*j+1));
          line
(x+5+(50*i+1),y+30+(50*j+1),x+5+(50*i+1),y+45+(50*j+1));
          floodfill((x+25+(50*i+1)),(y+25+(50*j+1)),WHITE);
     }
     else if
((c==3&&f==3)||(c==3&&f==5)||(c==3&&f==7)||(c==3&&f==1)){
          setcolor(WHITE);
          setcolor(WHITE);
         setfillstyle(SOLID_FILL,WHITE);
         line
(x+5+(50*i+1),y+45+(50*j+1),x+45+(50*i+1),y+45+(50*j+1));
          line
(x+45+(50*i+1),y+45+(50*j+1),x+45+(50*i+1),y+30+(50*j+1));
          line
(x+45+(50*i+1),y+30+(50*j+1),x+40+(50*i+1),y+20+(50*j+1));
          line
(x+40+(50*i+1),y+20+(50*j+1),x+35+(50*i+1),y+30+(50*j+1));
         line
(x+35+(50*i+1),y+30+(50*j+1),x+25+(50*i+1),y+5+(50*j+1));
         line
(x+25+(50*i+1),y+5+(50*j+1),x+15+(50*i+1),y+30+(50*j+1));
          line
(x+15+(50*i+1),y+30+(50*j+1),x+10+(50*i+1),y+20+(50*j+1));
          line
(x+10+(50*i+1),y+20+(50*j+1),x+5+(50*i+1),y+30+(50*j+1));
          line
(x+5+(50*i+1),y+30+(50*j+1),x+5+(50*i+1),y+45+(50*j+1));
          floodfill((x+25+(50*i+1)),(y+25+(50*j+1)),WHITE);
     }
     else if
((c==5&&f==3)||(c==5&&f==5)||(c==5&&f==7)||(c==5&&f==1)){
          setcolor(WHITE);
          setcolor(WHITE);
         setfillstyle(SOLID_FILL,WHITE);
         line
```

```
(x+5+(50*i+1),y+45+(50*j+1),x+45+(50*i+1),y+45+(50*j+1));
          line
(x+45+(50*i+1),y+45+(50*j+1),x+45+(50*i+1),y+30+(50*j+1));
          line
(x+45+(50*i+1),y+30+(50*j+1),x+40+(50*i+1),y+20+(50*j+1));
          line
(x+40+(50*i+1),y+20+(50*j+1),x+35+(50*i+1),y+30+(50*j+1));
         line
(x+35+(50*i+1),y+30+(50*j+1),x+25+(50*i+1),y+5+(50*j+1));
         line
(x+25+(50*i+1),y+5+(50*j+1),x+15+(50*i+1),y+30+(50*j+1));
          line
(x+15+(50*i+1),y+30+(50*j+1),x+10+(50*i+1),y+20+(50*j+1));
          line
(x+10+(50*i+1),y+20+(50*j+1),x+5+(50*i+1),y+30+(50*j+1));
          line
(x+5+(50*i+1),y+30+(50*j+1),x+5+(50*i+1),y+45+(50*j+1));
          floodfill((x+25+(50*i+1)),(y+25+(50*j+1)),WHITE);
     }
     else if
((c==7&&f==3)||(c==7&&f==5)||(c==7&&f==7)||(c==7&&f==1)){
          setcolor(WHITE);
          setcolor(WHITE);
         setfillstyle(SOLID_FILL,WHITE);
         line
(x+5+(50*i+1),y+45+(50*j+1),x+45+(50*i+1),y+45+(50*j+1));
          line
(x+45+(50*i+1),y+45+(50*j+1),x+45+(50*i+1),y+30+(50*j+1));
          line
(x+45+(50*i+1),y+30+(50*j+1),x+40+(50*i+1),y+20+(50*j+1));
          line
(x+40+(50*i+1),y+20+(50*j+1),x+35+(50*i+1),y+30+(50*j+1));
         line
(x+35+(50*i+1),y+30+(50*j+1),x+25+(50*i+1),y+5+(50*j+1));
         line
(x+25+(50*i+1),y+5+(50*j+1),x+15+(50*i+1),y+30+(50*j+1));
          line
(x+15+(50*i+1),y+30+(50*j+1),x+10+(50*i+1),y+20+(50*j+1));
```

```
          line
(x+10+(50*i+1),y+20+(50*j+1),x+5+(50*i+1),y+30+(50*j+1));
          line
(x+5+(50*i+1),y+30+(50*j+1),x+5+(50*i+1),y+45+(50*j+1));
          floodfill((x+25+(50*i+1)),(y+25+(50*j+1)),WHITE);
     }
     else{
          setcolor(RED);
          setcolor(RED);
         setfillstyle(SOLID_FILL,RED);
         line
(x+5+(50*i+1),y+45+(50*j+1),x+45+(50*i+1),y+45+(50*j+1));
          line
(x+45+(50*i+1),y+45+(50*j+1),x+45+(50*i+1),y+30+(50*j+1));
          line
(x+45+(50*i+1),y+30+(50*j+1),x+40+(50*i+1),y+20+(50*j+1));
          line
(x+40+(50*i+1),y+20+(50*j+1),x+35+(50*i+1),y+30+(50*j+1));
         line
(x+35+(50*i+1),y+30+(50*j+1),x+25+(50*i+1),y+5+(50*j+1));
         line
(x+25+(50*i+1),y+5+(50*j+1),x+15+(50*i+1),y+30+(50*j+1));
          line
(x+15+(50*i+1),y+30+(50*j+1),x+10+(50*i+1),y+20+(50*j+1));
          line
(x+10+(50*i+1),y+20+(50*j+1),x+5+(50*i+1),y+30+(50*j+1));
          line
(x+5+(50*i+1),y+30+(50*j+1),x+5+(50*i+1),y+45+(50*j+1));
          floodfill((x+25+(50*i+1)),(y+25+(50*j+1)),RED);
     }
}

//Funcion que imprime el tablero con las reinas
void Imprimir(int (*matriz)[SIZE],int n)
{
```

```
     int i,j,o=1,p=1,x=50,y=100;
     for(i=0;i<n;i++,o++)
     {
          for(j=0;j<n;j++,p++)
          {
               printf("%d",matriz[i][j]);
               if(matriz[i][j]==1){



                         setcolor(BLUE);
                         outtextxy(95,50,"Reinas que no comen");
                         setcolor(BLUE);
                    setfillstyle(SOLID_FILL,BLUE);
                    line
(x+5+(50*i+1),y+45+(50*j+1),x+45+(50*i+1),y+45+(50*j+1));
                    line
(x+45+(50*i+1),y+45+(50*j+1),x+45+(50*i+1),y+30+(50*j+1));
                    line
(x+45+(50*i+1),y+30+(50*j+1),x+40+(50*i+1),y+20+(50*j+1));
                    line
(x+40+(50*i+1),y+20+(50*j+1),x+35+(50*i+1),y+30+(50*j+1));
                    line
(x+35+(50*i+1),y+30+(50*j+1),x+25+(50*i+1),y+5+(50*j+1));
                    line
(x+25+(50*i+1),y+5+(50*j+1),x+15+(50*i+1),y+30+(50*j+1));
                    line
(x+15+(50*i+1),y+30+(50*j+1),x+10+(50*i+1),y+20+(50*j+1));
                    line
(x+10+(50*i+1),y+20+(50*j+1),x+5+(50*i+1),y+30+(50*j+1));
                    line
(x+5+(50*i+1),y+30+(50*j+1),x+5+(50*i+1),y+45+(50*j+1));

     floodfill((x+25+(50*i+1)),(y+25+(50*j+1)),BLUE);
                         }
               }
```

```
        printf("\n");
    }
    Sleep(10);
    return;
    }
```

Conclusión.

Es de esta forma que podemos concluir que todos los que nos podemos encontrar en los diversos programas que realizamos, tienen diversas formas de resolverse sin perder de vista que cada una de estas formas tienen diversas desventajas y ventajas en su haber, parte de esto es el gran trabajo del programador descubrir cuáles son y de qué manera utilizarlas en su beneficio o como herramienta, para hacer de su trabajo más fácil sencillo y eficiente, pues el código por sí mismo es solo código, es la lógica del programador la que logra que todo forme parte de un ciclo en el cual se realizan diversas partes y tareas específicas.

Referencias Bibliográficas:

[1]: Algorítmica y Programación. (2017). Algoritmo. Retrieved November 3, 2020, from Google.com website: https://sites.google.com/site/portafoliocarlosmacallums/unidad-i/algoritmo

[2]: Delgado Arturo. (2017). Características de un algoritmo. Retrieved November 3, 2020, from Google.com website: https://sites.google.com/site/portafoliocarlosmacallums/unidad-i/caracteristicasdeunalgoritmo

[3]: Guillermo, J. (2018, June 23). ¿Qué es la complejidad algorítmica y con qué se come? Retrieved November 4, 2020, from Medium website: https://medium.com/@joseguillermo_/qu%C3%A9-es-la-complejidad-algor%C3%ADtmica-y-con-qu%C3%A9-se-come-2638e7fd9e8c

[4]: José, M. (2020). Representación de un algoritmo | Algoritmos y pseudocódigo. Retrieved November 3, 2020, from Intef.es website: http://formacion.intef.es/pluginfile.php/87713/mod_imscp/content/13/representacin_de_un_al

goritmo.html#:~:text=El%20Pseudoc%C3%B3digo%20es%20sin%20duda,de%20los%20lenguajes%20de%20programaci%C3%B3n.

[5]: Programación II. (2015). Tipo de Dato Abstracto. Retrieved November 3, 2020, from Google.com website: https://sites.google.com/site/programacioniiuno/temario/unidad-2---tipo-abstracto-de-dato/tipo-de-dato-abstracto

[6]: Funciones Hash (teoría y ejemplo)*: "Programación en C: Metodología, algoritmos y estructura de datos"*

[7]: Geeky Theory, & Geeky Theory. (2020). ¿Qué es la recursividad? Retrieved November 24, 2020, from Geeky Theory website: https://geekytheory.com/que-es-la-recursividad

[8]: Vuelta atrás (backtracking) - EcuRed. (2020). Retrieved November 24, 2020, from Ecured.cu website: https://www.ecured.cu/Vuelta_atr%C3%A1s_(backtracking)