**Informe del Algoritmo Buscaminas**

Profesor:

Morello, Hernan Javier

Orosco, Ricardo Fabian

Wehbe, Ricardo Abraham

Grupo 8:

Sztafij, Bianca Lucila – LU: 1102702

Pieczocha, Juan Pablo – LU: 1103622

Mancuello, Augusto Ariel – LU: 1104732

Buenos Aires, 12 de Noviembre de 2019 -

**Tabla de Contenidos**

[Introducción 1](#_Toc24716295)

[Descripción del Problema 1](#_Toc24716296)

[Estrategia de Resolución 1](#_Toc24716297)

[Pseudocódigo del Algoritmo de Resolución del Problema 1](#_Toc24716298)

[Análisis de Complejidad Temporal 1](#_Toc24716299)

[Conclusiones 1](#_Toc24716300)

[Bibliografía 1](#_Toc24716301)

## Introducción

*Este trabajo fue realizado por Augusto Ariel Mancuello, Juan Pablo Pieczocha* y *Bianca Lucila Sztafij. Para la resolución del problema se codificó un algoritmo, en el lenguaje de programación Java, capaz de resolver el problema recorriendo todas las soluciones posibles en caso de ser necesario y devolver la que sería óptima.*

## Descripción del Problema

*El problema consiste en buscar a partir de un tablero del clásico juego de buscaminas la solución más corta del mismo simulando una partida. El tablero lo recibiríamos como un archivo de texto y el programa deberá extraer la información para crear una matriz con los datos correspondientes. Una vez armada dicha matriz, el programa proseguiría con la resolución de la partida y devolverá una lista con la menor cantidad de posiciones posibles para ganar el juego.*

## Estrategia de Resolución

La estrategia de resolución utilizada fue la de backtracking realizando una búsqueda en profundidad ya que se busca simular la partida desde la perspectiva del jugador. Además, se emplearon algunas condiciones de poda para disminuir la complejidad algorítmica lo máximo posible sin desperdiciar preciados recursos como lo son el rendimiento y el tiempo. Algunas de estas condiciones son:

* Encontrar una bomba
* Encontrar una casilla ya descubierta
* Encontrar una solución
* Descubrir una posible solución más larga que la anterior

El algoritmo recorrerá casillero por casillero buscando la mejor combinación de estos posibles empleando distintos métodos para ganar la partida haciendo uso de la menor cantidad de jugadas posibles.

## Pseudocódigo del Algoritmo de Resolución del Problema

ALGORITMO BUSCAMINAS

Entrada: *rutaArchivo*: String

Salida: *MejorSolucion*: Lista<Casilla>

Archivo *info*  AbrirArchivo(*rutaArchivo*)

String *PrimerLinea*  LeerLinea(*info*)

Vector *datos*  Separar(*PrimerLinea,* “*,*”)

*filas datos[0]*

*columnas datos[1]*

*cantBombas datos[2]*

Matriz<Casilla> *m* InicializarMatriz(*filas, columnas*)

Lista<Casilla> *pendientes* InicializarLista()

Lista<Casilla> *opciones* InicializarLista()

*i 0*

mientras (*i < filas*)

String *línea*  LeerLinea(*info*)

Lista<String> *num* Separar(*linea,* “ ”)

*j 0*

mientras (*j < columnas*)

Carácter *a*  *num[j]*

Casilla *pos* InicializarCasilla(*i, j, a*)

*m[i][j] pos*

Agregar(*pendientes, pos*)

Agregar(*opciones, pos*)

*j j + 1*

fin mientras

*i i + 1*

fin mientras

CerrarArchivo(*info*)

Lista<Casilla> *MejorSolucion* InicializarLista()

Conjunto<Casilla> *bombas* InicializarConjunto()

mientras ( Longitud(*m*))

mientras ( Longitud())

Lista<Casilla> *solucionParcial*  InicializarLista()

si (Longitud(*MejorSolucion*) Y NO Contiene(*bombas*,*m[x][y]*))

*MejorSolucion* Jugar(*m, m[x][y], pendientes, cantBombas, bombas, solucionParcial, MejorSolucion*)

fin si

fin mientras

fin mientras

devolver *MejorSolucion*

ALGORITMO JUGAR

Entrada: *m*: Matriz<Casilla>, *c*: Casilla, *pendientes*: Lista<Casilla>, *cantBombas*: Entero, *bombas:* Conjunto<Casilla>, *clicks*: Lista<Casilla>, *solucionFinal*: Lista<Casilla>

Salida: *solucionFinal*: Lista<Casilla>

si (Numero(*c*) = ‘X’)

Agregar(*bombas, c*)

sino

Lista<Casilla> *revelados* InicializarLista()

Agregar(*clicks, c*)

Revelar(*m, c, revelados, pendientes*)

si (Longitud(*pendientes*) = *cantBombas*)

*solucionFinal* InicializarLista(*clicks*)

sino

mientras ( Longitud(*m*))

si ( NO ( Longitud() + 1 < Longitud() Y Longitud() 0))

mientras ( Longitud())

si (NO (Longitud(*clicks*) + 1 Longitud(*solucionFinal*) Y Longitud(*solucionFinal*) ) Y NO Contiene(*bombas, m[i][j]*) Y Contiene(*pendientes*,*m[i][y]*))

*solucionFinal* Jugar(*m, m[i][j], pendientes, cantBombas, bombas, solucionParcial, solucionFinal*)

fin si

fin mientras

fin si

fin mientras

fin si

Ocultar(*m, revelados, pendientes*)

Sacar(*clicks*, *c*)

fin si

devolver *solucionFinal*

ALGORITMO REVELAR

Entrada: *m*: Matriz<Casilla>, *c*: Casilla, *revelados*: Lista<Casilla>, *pendientes*: Lista<Casilla>

Agregar(*revelados, c*)

Sacar(p*endientes*, *c*)

si (Numero(*c*) = ‘0’)

*x -1*

mientras (*x <= 1*)

*i*  Fila(*c*) + *x*

si (*i > 0* Y *i* < Longitud(*m*))

*y*  -1

mientras (y *<= 1*)

*j*  Columna(*c*) + *y*

si (*j > 0* Y *j* < Longitud(*m[0]*))

Casilla *x m[i][j]*

si (Contiene(*pendientes, x*))

Revelar(*m, x, revelados, pendientes*)

fin si

fin si

*y y + 1*

fin mientras

fin si

*x x + 1*

fin mientras

fin si

ALGORITMO OCULTAR

Entrada: *m*: Matriz<Casilla>, *revelados*: Lista<Casilla>, *pendientes*: Lista<Casilla>

mientras (Longitud(*revelados*) *> 0*)

Casilla *c revelados[0]*

Agregar(*pendientes, c*)

fin mientras

## Análisis de Complejidad Temporal

Para el algoritmo Revelar conocemos que es una función recursiva ya que si al obtener una casilla con un valor de 0, se revelarán las adyacentes. La función es llamada a medida que vamos verificando casilla por casilla hasta 8 veces (Los ciclos hacen 9 verificaciones, pero al preguntar si la casilla, que acabamos de sacar del vector pendientes, pertenece al vector pendientes nos dirá que no) en el peor de los casos. Entonces conocemos:

* El tipo de recurrencia: Sustracción
* Las variables:

Entonces el costo de el algoritmo Revelar es de donde es la cantidad de casillas adyacentes no reveladas cuando la casilla recibida como parámetro equivale a 0.

Para el algoritmo Ocultar recibimos una lista con los vectores que fueron revelados previamente por la función Revelar. El algoritmo recorre de forma iterativa la lista recibida agregando las posiciones a la lista de pendientes y removiendo dichas posiciones de la lista de revelados. El costo de esta función es donde es la cantidad de elementos de la lista de casillas reveladas.

El algoritmojugar recibirá como parámetro una lista con la cantidad de posiciones pendientes, a la que se le sacará un elemento por lo menos (A través de la función Revelar) antes de que se vuelva a realizar el llamado recursivo. Al realizar dicho llamado, la cantidad de opciones que tiene la lista va a ser de tamaño . La función se volverá a llamar a sí misma constantemente hasta que o bien, encuentre una solución o una bomba, o bien el valor de llegue a igualarse con la cantidad de bombas que el algoritmo recibió como parámetro. Entonces el costo de la función Jugar es de donde es la cantidad de casillas en el tablero y el factorial se debe a que se están realizando todas las combinaciones posibles.

Por último, el algoritmo Buscaminas posee dos ciclos que iteran la misma cantidad de veces como casillas tenemos y además existe otro ciclo que llamará a la función jugar otras veces. Por lo que la función posee un costo de donde es la cantidad de casillas del buscaminas y el se debe a que se llama al algoritmo Jugar.

## Conclusiones

La solución óptima se da cuando se hace click sobre casillas no reveladas cuyo valor sea 0 y sobre aquellas que no poseen un 0 como casilla adyacente o vecina.

El problema de este algoritmo es que al usar backtracking, ya que buscamos simular el juego desde la perspectiva del jugador, vuelve al algoritmo demasiado complejo ya que para terminar su ejecución debe buscar todas las soluciones posibles. Algunas condiciones de poda fueron agregadas para acortar considerablemente la ejecución del programa y no perder tiempo buscando soluciones no óptimas, ya que el tiempo que demanda para tableros de un tamaño considerable puede hacer que el programa exija de un largo período de tiempo para la resolución de este.

# Bibliografía

* [xve](https://www.lawebdelprogramador.com/codigo/usuario.php?id=147685). (29 de Octubre de 2019). “Código de Java - Leer y escribir en un archivo de texto”. La Web del programador. <https://www.lawebdelprogramador.com/codigo/Java/3842-Leer-y-escribir-en-un-archivo-de-texto.html>
* Daniel Shiffman. (30 de Octubre de 2019). “Minesweeper - Coding Challenge #71 · The Coding Train”. The Coding Train. <https://thecodingtrain.com/CodingChallenges/071-minesweeper.html>
* Ricardo Moya. (31 de Octubre de 2019). “ArrayList en Java, con ejemplos – Jarroba”. Jarroba. <https://jarroba.com/arraylist-en-java-ejemplos/>