 TECNOLÓGICO DE COSTA RICA

Escuela de Computación

***Análisis de Algoritmos***

**I Proyecto Programado**

Eduardo Binns

Dilana Sancho Vargas

Sebastián Rojas Vargas

Sede San Carlos

de Mayo, 2020

Tabla de contenido

[Introducción 2](#_Toc39186666)

[Análisis del problema 3](#_Toc39186667)

[Solución del problema 5](#_Toc39186668)

[Análisis de Resultados 9](#_Toc39186669)

[Medición empírica 10](#_Toc39186670)

[Determinar el factor de crecimiento 12](#_Toc39186671)

[Medición analítica 16](#_Toc39186672)

[Medición Gráfica 18](#_Toc39186673)

[Conclusiones 19](#_Toc39186674)

[Recomendaciones 20](#_Toc39186675)

[Bibliografía 21](#_Toc39186676)

# Introducción

El presente proyecto programado pretende realizar una introducción al análisis de algoritmos basados en estrategias de búsqueda…..

# Análisis del problema

Para comprender mejor el problema se va a separar de modo, que se analiza parte por parte la construcción de cómo se puede llegar a una posible solución.

El primero de los factores que se analiza es la forma en que se organiza la matriz, según las reglas del sudoku, el número no se puede repetir en toda la fila y en toda la columna, tampoco en el mismo recuadro, por eso se pensó que la mejor solución para el problema es dividir los números por filas.

La matriz tiene que tener una cantidad de números, cada vez menor para probar la forma en la que se comporta el algoritmo. La matriz está compuesta con filas y columnas de números enteros.

Segundo problema que analizar es cómo funciona el algoritmo de fuerza bruta

El algoritmo de fuerza bruta o búsqueda exhaustiva requiere probar todas las posibles combinaciones, y para esto todas las posibles combinaciones van del número 1 al 9.

Fuerza bruta debe probar con cada una de las posibilidades a modo que cumpla con las reglas del tablero:

Regla 1: que el numero que se prueba no esté en la fila.

Regla 2: que el número que se prueba no esté en la columna.

Regla 3: que el número que se prueba no esté en el cuadrante.

La solución se encuentra hasta que el algoritmo de fuerza bruta encuentre una solución idéntica a la original.

Tercer problema para analizar es cómo funciona la programación lineal

El Algoritmo de Programación Lineal a diferencia del de fuerza bruta busca crea una lista de las posibilidades para cada celda, quitando la necesidad de probar todas las combinaciones a la hora de resolver el sudoku.

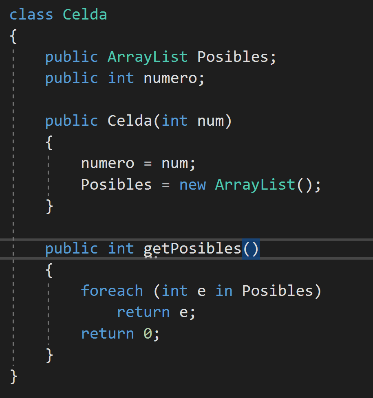
Para Algoritmo de Programación Lineal es importante ver que el tablero del sudoku necesita crear en cada celda un número entero que representa el valor posible y una lista de posibilidades.

La solución debe cumplir con las mismas reglas del algoritmo de fuerza bruta. Y cuando se coloque un numero de la lista de posibilidades quitarlo de la lista de posibilidades del cuadrante.

# Solución del problema

Mediante el uso de la programación orientada a objetos se implementó una clase llamada “Tablero” en la que se guardó por defecto un arreglo de enteros, este contiene todos los números de un tablero de sudoku de 9x9 resuelto y acomodados por filas.

Se creo un método constructor para hacer que el tablero de sudoku cree una solución temporal en la que tenga solo la cantidad de números que se piden para el ejercicio.

Se implemento una clase **Celda** para manejar la matriz del sudoku para el algoritmo de programación lineal. Se creo una matriz de Celdas.

Para el algoritmo de Fuerza bruta que tenia menos complejidad se logro con una matriz de números enteros.

Existen dos clases tablero con una matriz de enteros y otra de celdas.

**Función que verifica que el número este en el recuadro**

Esta es una función muy complicada.

|  |
| --- |
| /// <param name="numero"></param> el numero que esta probando  /// <param name="filaActual"></param> fila en la que se encuentra en toda la matriz  /// <param name="columnaActual"></param> columna en la que se encuentra en toda la matriz  /// <param name="x"></param> tablero de juego  /// <returns></returns>  public static bool auxVerificarCuadro(int numero, int filaActual, int columnaActual, int[,] x){  int filaCuadro = filaActual - filaActual % 3;  int columCuadro = columnaActual - columnaActual % 3;  // 'filaCuadro + 3' para llegar al numero superior al final,  para evitar que se salga de rango  int fila1 = (filaActual/3)\*3;  int columna1 = (columnaActual/3)\*3;  // para colocarme en la posicion inicial del recuadro  for (int fila = fila1; fila < filaCuadro + 3; fila++ )  {  for(int columna = columna1; columna < columCuadro + 3; columna++)  {  if (x[fila, columna] == numero)  {  return true;  }  }  return false;  }  return true;  } |

**Funciones que verifican la fila y la columna**

Recorren las filas y columnas del tablero buscando en cada posición si el número (parámetro “i”) que está probando se puede poner en la celda. Si lo encuentra retorna un valor **true** indicando que sí encontró el valor, y que por eso no puede poner el número en la celda. Solo al retornar una **false** coloca el número en la celda.

|  |
| --- |
| /// <summary>  Funcion que recorre las filas de la matriz  Pregunta si el numero que esta probando se encuentra en la columna  /// </summary>  /// <param name="i"></param>  /// <param name="f"></param>  /// <param name="c"></param>  /// <param name="x"></param>  /// <returns></returns>  public static bool auxVerificarColumna(int i, int c, int[,] x)  {  for (int vf = 0; vf < 9; ++vf)  {  if (x[vf, c] == i)  {  return true;  }  }  return false;  } |
| /// <summary>  Función que recorre las columnas de la matriz  Pregunta si el número que está probando se encuentra en la fila  /// </summary>  /// <param name="i"></param>  /// <param name="f"></param> fila  /// <param name="c"></param> columna  /// <param name="x"></param> arreglo tablero de juego  /// <returns></returns>  public static bool auxVerificarFila(int i, int f, int [,] x)  {  for (int vc = 0; vc < 9; ++vc)  {  if (x[f, vc] == i)  {  return true;  }  }  return false;  } |

Algoritmo de fuerza bruta

El algoritmo comienza recorriendo las filas y las columnas cuando encuentra una casilla con un 0 el equivalente a vacío comienza a probar las posibilidades, pregunta si el numero cumple las reglas y coloca el numero si las cumple. Pregunta si el algoritmo ya encontró una solución, sino pone el campo vacío otra vez.

Continúa recorriendo la matriz hace proceso nuevamente, pero con el siguiente campo para ver si la solución de ese espacio, si lo es va a llegar otra vez a la pregunta que lo manda al siguiente campo y si es una solución correcta se llena el tablero y retorna true, de lo contrario si no es solución el número que está probando se devuelve a la llamada anterior colocando un cero hasta llegar para probar con otra posibilidad que si sea solución.

|  |
| --- |
| public static Boolean algoritmo\_FuerzaBruta(int[,] tablero)  {  // recorre las filas  for (int f = 0; f < 9; f++)  { //recorre las columnas  for (int c = 0; c < 9; c++)  { //pregunta si la casilla esta vacia  if (tablero[f, c] == 0)  { //prueba las posibles soluciones  for (int i = 1; i < 10; i++)  {  //Pregunta si se cumplen estas reglas  if (auxVerificarFila(i, f, tablero) == false &&  auxVerificarColumnaFila(i, c, tablero) == false  && auxVerificarCuadro(i, f, c, tablero) == false)  {  tablero[f, c] = i;  if (algoritmo\_FuerzaBruta(tablero))  {  return true;  }  else  {  tablero[f, c] = 0;  }  }  }  return false;  }  }  }  return true;//para finalizar  } |

# Análisis de Resultados

|  |  |  |
| --- | --- | --- |
| Requerimiento | Estado | Observaciones |
| Lógica | | |
| Algoritmo fuerza bruta | Listo |  |
| Algoritmo lineal | Listo |  |
| Selección del algoritmo | Listo |  |
| Selección de la cantidad de números del arreglo FB | Listo |  |
| Mediciones Analíticas FB | Listo |  |
| Mediciones empíricas FB | Listo |  |
| Gráficos FB | Listo |  |
| Selección de la cantidad de números del arreglo Lineal | Listo |  |
| Mediciones Analíticas Lineal | Listo |  |
| Mediciones empíricas Lineal | Listo |  |
| Gráficos Lineal | Listo |  |

## Medición empírica

* 1. Nombre del algoritmo \_Recorrido en profundidad de un arbol\_\_#1: \_\_\_\_\_\_\_\_\_

| Operaciones | Cantidad de datos de entrada por defecto | | | | | | |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |
| Asignaciones |  |  |  |  |  |  |
| Comparaciones |  |  |  |  |  |  |
| Cantidad de líneas ejecutadas |  |  |  |  |  |  |
| Tiempo de ejecución |  |  |  |  |  |  |
| Cantidad de líneas del código |  | | | | | | |

* 1. Nombre del algoritmo \_\_\_Recorrido en anchura de un arbol\_\_\_#2: \_\_\_\_\_\_\_\_\_

| Operaciones | Cantidad de datos de entrada por defecto | | | | | | |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |
| Asignaciones |  |  |  |  |  |  |
| Comparaciones |  |  |  |  |  |  |
| Cantidad de líneas ejecutadas |  |  |  |  |  |  |
| Tiempo de ejecución |  |  |  |  |  |  |
| Cantidad de líneas del código |  | | | | | | |

* 1. Nombre del algoritmo \_\_\_Recorrido en anchura de un grafo\_\_\_#3: \_\_\_\_\_\_\_\_\_

| Operaciones | Cantidad de datos de entrada por defecto | | | | | | |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |
| Asignaciones |  |  |  |  |  |  |
| Comparaciones |  |  |  |  |  |  |
| Cantidad de líneas ejecutadas |  |  |  |  |  |  |
| Tiempo de ejecución |  |  |  |  |  |  |
| Cantidad de líneas del código |  | | | | | | |

* 1. Nombre del algoritmo \_\_\_Recorrido en anchura de un grafo\_\_\_#4: \_\_\_\_\_\_\_\_\_

| Operaciones | Cantidad de datos de entrada por defecto | | | | | | |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |
| Asignaciones |  |  |  |  |  |  |
| Comparaciones |  |  |  |  |  |  |
| Cantidad de líneas ejecutadas |  |  |  |  |  |  |
| Tiempo de ejecución |  |  |  |  |  |  |
| Cantidad de líneas del código |  | | | | | | |

## Determinar el factor de crecimiento

* 1. Nombre del algoritmo \_ Recorrido en profundidad de un árbol \_\_ #1: \_\_\_\_\_\_\_\_\_\_

| Talla | | Factor talla | Factor Asig | Factor Comp | Factor Cantidad de líneas ejecutadas | Factor Tiempo de ejecución |
| --- | --- | --- | --- | --- | --- | --- |
| De -- a -- |  |  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
| De -- a -- |  |  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
| De -- a -- |  |  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
| De -- a --- |  |  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
| De -- a --- |  |  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
| De --- a ---- |  |  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |

|  |  |  |
| --- | --- | --- |
| Clasificación del comportamiento de las **asignaciones** |  |  |
| Clasificación del comportamiento de las **comparaciones** |  |
| Clasificación del comportamiento de las **líneas ejecutadas** |  |
| Clasificación del comportamiento en el **tiempo de ejecución** |  |

* 1. Nombre del algoritmo \_ Recorrido en anchura de un árbol \_\_ #2: \_\_\_\_\_\_\_\_\_\_\_\_\_

| Talla | | Factor talla | Factor Asig | Factor Comp | Factor Cantidad de líneas ejecutadas | Factor Tiempo de ejecución |
| --- | --- | --- | --- | --- | --- | --- |
| De -- a -- |  |  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
| De -- a -- |  |  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
| De -- a -- |  |  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
| De -- a --- |  |  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
| De -- a --- |  |  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
| De --- a ---- |  |  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |

|  |  |  |
| --- | --- | --- |
| Clasificación del comportamiento de las **asignaciones** |  |  |
| Clasificación del comportamiento de las **comparaciones** |  |
| Clasificación del comportamiento de las **líneas ejecutadas** |  |
| Clasificación del comportamiento en el **tiempo de ejecución** |  |

* 1. Nombre del algoritmo \_ Recorrido en profundidad de un grafo \_\_ #3: \_\_\_\_\_\_\_\_\_\_

| Talla | | Factor talla | Factor Asig | Factor Comp | Factor Cantidad de líneas ejecutadas | Factor Tiempo de ejecución |
| --- | --- | --- | --- | --- | --- | --- |
| De -- a -- |  |  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
| De -- a -- |  |  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
| De -- a -- |  |  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
| De -- a --- |  |  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
| De -- a --- |  |  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
| De --- a ---- |  |  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |

|  |  |  |
| --- | --- | --- |
| Clasificación del comportamiento de las **asignaciones** |  |  |
| Clasificación del comportamiento de las **comparaciones** |  |
| Clasificación del comportamiento de las **líneas ejecutadas** |  |
| Clasificación del comportamiento en el **tiempo de ejecución** |  |

* 1. Nombre del algoritmo \_ Recorrido en anchura de un grafo \_\_ #4: \_\_\_\_\_\_\_\_\_\_\_\_\_

| Talla | | Factor talla | Factor Asig | Factor Comp | Factor Cantidad de líneas ejecutadas | Factor Tiempo de ejecución |
| --- | --- | --- | --- | --- | --- | --- |
| De -- a -- |  |  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
| De -- a -- |  |  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
| De -- a -- |  |  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
| De -- a --- |  |  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
| De -- a --- |  |  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
| De --- a ---- |  |  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |

|  |  |  |
| --- | --- | --- |
| Clasificación del comportamiento de las **asignaciones** |  |  |
| Clasificación del comportamiento de las **comparaciones** |  |
| Clasificación del comportamiento de las **líneas ejecutadas** |  |
| Clasificación del comportamiento en el **tiempo de ejecución** |  |

## Medición analítica

* 1. Nombre del algoritmo \_ Recorrido en anchura de un árbol \_\_ #1: \_\_\_\_\_\_\_\_\_\_\_\_\_

|  |  |
| --- | --- |
| Código fuente  Solo se analiza el código del método de resolver el rompecabezas. | Medición de líneas ejecutadas en el peor de los casos  (línea por línea) |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
| Total (la suma de todos los pasos) |  |
| **Clasificación en notación O Grande** |  |

* 1. Nombre del algoritmo \_ Recorrido en anchura de un árbol \_\_ #2: \_\_\_\_\_\_\_\_\_\_\_\_\_

|  |  |
| --- | --- |
| Código fuente  Solo se analiza el código del método de resolver el rompecabezas. | Medición de líneas ejecutadas en el peor de los casos  (línea por línea) |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
| Total (la suma de todos los pasos) |  |
| **Clasificación en notación O Grande** |  |

* 1. Nombre del algoritmo \_ Recorrido en anchura de un grafo \_\_ #3: \_\_\_\_\_\_\_\_\_\_\_\_\_

|  |  |
| --- | --- |
| Código fuente  Solo se analiza el código del método de resolver el rompecabezas. | Medición de líneas ejecutadas en el peor de los casos  (línea por línea) |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
| Total (la suma de todos los pasos) |  |
| **Clasificación en notación O Grande** |  |

* 1. Nombre del algoritmo \_ Recorrido en anchura de un grafo \_\_ #4: \_\_\_\_\_\_\_\_\_\_\_\_\_

|  |  |
| --- | --- |
| Código fuente  Solo se analiza el código del método de resolver el rompecabezas. | Medición de líneas ejecutadas en el peor de los casos  (línea por línea) |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
| Total (la suma de todos los pasos) |  |
| **Clasificación en notación O Grande** |  |

## Medición Gráfica

Las graficas indican un comportamiento mas constante y un menor crecimiento con el algoritmo lineal.

# Conclusiones

Al analizar los datos de entrada y la cantidad de asignaciones y comparaciones que se usaron para resolver el rompecabezas del sudoku mediante el algoritmo de fuerza bruta, podemos determinar que con tamaño de datos de entrada 80 se resuelve considerablemente rápido, y en el caso del tamaño de datos de entrada 50 el sudoku esta acomodado de tal manera que en una de las equinas inferiores toda la columna está en cero esto hace que el la cantidad de comparaciones y asignaciones aumente considerablemente, por eso podemos concluir que el orden y la posición de los números vacíos afecta la resolución del sudoku mediante el algoritmo de fuerza bruta.

Con los datos de entrada en el algoritmo lineal sucedió algo similar al algoritmo de fuerza bruta al estar acomodados todos los vacíos en una posición provoco un crecimiento en las asignaciones y comparaciones.

Se puede concluir que el algoritmo lineal es mucho más rápido en tiempo de ejecución que el algoritmo de fuerza bruta, solo cuando la cantidad de datos de entrada fue mínima se aumentó el tiempo.

# Recomendaciones

* Evitar en lo más posible repetición de código mediante el uso modularidad. Implementando la programación orientada a objetos, de esa manera se evita la duplicación de variables.

# Bibliografía