PROYECTO FINAL FUNDAMENTOS DE ANALISIS Y DISEÑO DE ALGORITMOS

SU-DOMINO-KU

JOHAN SEBASTIAN MARULANDA ALMANZA
1556060-3743

CRISTHIAN RENDON SANCHEZ
1556246-3743

MANUEL ALEJANDRO VICTORIA
1556231-3743

PROGRAMA DE INGENIERÍA DE SISTEMAS
DEPARTAMENTO DE INGENIERÍA DE SISTEMAS,
FACULTAD DE INGENIERÍA

UNIVERSIDAD DEL VALLE
TULUÁ, Junio 19 de 2017

PROYECTO FINAL FUNDAMENTOS DE ANALISIS Y DISEÑO DE ALGORITMOS

SU-DOMINO-KU

JOHAN SEBASTIAN MARULANDA ALMANZA
1556060-3743

CRISTHIAN RENDON SANCHEZ
1556246-3743

MANUEL ALEJANDRO VICTORIA
1556231-3743

Profesor: CARLOS ANDRES DELGADO SAAVEDRA
Fundamentos de Análisis y Diseño de Algoritmos

PROGRAMA DE INGENIERÍA DE SISTEMAS
DEPARTAMENTO DE INGENIERÍA DE SISTEMAS,
FACULTAD DE INGENIERÍA
UNIVERSIDAD DEL VALLE
TULUÁ, Junio 19 de 2017

INTRODUCCION

A la hora de programar existen diferentes técnicas que nos permiten abordar un problema con mayor o menor dificultad; no existe una solución perfecta que permita resolver de una forma óptima todos los problemas que puedan ser planteados e incluso existen técnicas que permiten resolver las tareas más difíciles pero en el momento de implementarlas a un problema sin un grado dificultad alto, simplemente no son útiles y solo incrementan la complejidad de la solución sin aportar ninguna mejora a lo que una solución menos elaborada y más barata lograría.

En el mundo de la programación existen dos estrategias que son ampliamente utilizadas en las situaciones que un problema exige en su solución la solución de su problemas repetidos; estas técnicas son conocidas como programación dinámica (que se encarga de abordar el problema con una serie de pasos los cuales se compone de: caracterizar la solución del problema, definir la recursividad y la estructura de datos, además de calcular su solución de forma bottom-up. El seguir esta serie de pasos permite encontrar la mejor solución más óptima) y programación voraz (que se encarga de tomar la mejor decisión posible en un instante, es decir en cada situación que se enfrente el algoritmo y deba tomar decisión siempre tomara la opción que le dé un mayor beneficio, es decir entrega una solución que está basada en la toma de las mejores decisiones, pero que no garantiza que se la mejor decisión al final de la iteración)

A continuación se presenta el análisis de un problema conocido como sudominoku que tiene como objetivo llenar un tablero de un sudoku con fichas de dominó aplicando ambas técnicas descritas anteriormente.

OBJETIVOS

OBJETIVOS GENERALES:

- Desarrollar un algoritmo que solucione el ejercicio conocido como sudominoku

OBJETIVOS ESPECIFICOS:

- Caracterizar la solución del problema del sudominoku por medio de programación dinámica
- Definir las estructuras la recursividad que permita la resolución del problema
- Calcular el valor de una solución de forma bottom-up
- Construir una solución óptima a partir de la información obtenida
- Construir un algoritmo que resuelva el sudominoku usando la programación dinámica
- Construir un algoritmo que resuelva el sudominoku usando la programación voraz

JUSTIFICACION

La resolución de juegos de mesa atraes de la programación es un planteamiento que se realiza para mejor la capacidad de abstracción que posee el estudiante, es decir el modelar el problema del sudominoku por medio de programación ayuda a la capacidad de abstraer elementos de la vida real y aplicarlos computacionalmente.

El sudominoku es un juego derivado del sudoku, cuya diferencia radica en que en vez de colocar números separados en casillas independientes, la manera en que se soluciona o se basa su sistemas de llenado, no es por medio de números separados si no de un par de números concatenados (en forma de ficha de domino) que ocupan respectivamente cada uno la casilla correspondiente; pero bajo ninguna circunstancia los números de la ficha pueden ser separados el reto es encontrar una configuración valida (o la más cercana a esta) que represente la solución atraves de la programación dinámica o voraz

ANTECEDENTES

En el siglo XVIII, el suizo, Leonhard Euler, creó un sistema de probabilidades para representar una serie de número sin repetir. Debido a esto se le considera el inventor del sudoku

Reglas:

El sudoku se presenta normalmente como una tabla de 9× 9, compuesta por subtablas de 3 × 3 denominadas "sectores"

Algunas celdas ya contienen números, conocidos como "números dados" (El objetivo es rellenar las celdas vacías, con un número en cada una de ellas, de tal forma que cada columna, fila y región contenga los números 1–9 solo una vez.

Además, cada número de la solución aparece solo una vez en cada una de las tres "direcciones

Como dato curioso la suma de todas las filas y todas las columnas da 405 y se suma individualmente cada fila o columna tiene como resultado 45, lo que nos recuerda a un cuadrado mágico de durero

SOLUCION DINAMICA:

a. Caracterización de la estructura optima:

Para la solución óptima de los sub problemas para esto. Básicamente nos planteamos una estrategia:

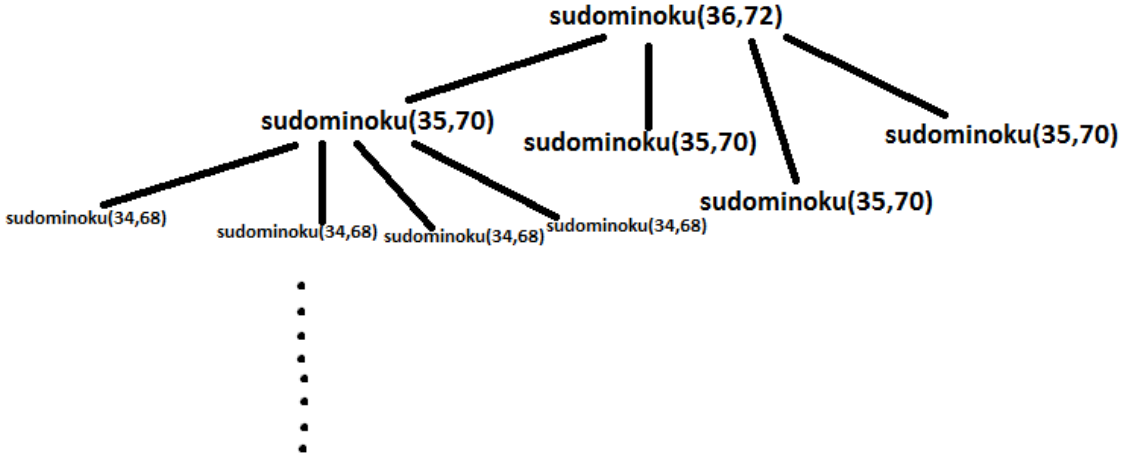
- La primera estrategia es responder la pregunta ¿Qué ficha puede ocupar esta posición? Para ello, tenemos las 3 reglas que se deben cumplir. Además de eso tenemos 4 posibles ángulos en los que se puede posicionar mi ficha y tenemos que validar el cumplimiento de las reglas para el ángulo dado

Las restricciones que tenemos son las que nos plantean inicialmente:

- Cada fila tiene que contener todos los del 1 al 9.
- Cada columna tiene que contener todos los dígitos del 1 al 9.
- Cada región de 3 x 3 tiene que contener todos los dígitos del 1 al 9.

Para la solución de una tabla podemos generar una enorme cantidad de Su-domino-kus distintos, pero matemáticamente equivalentes.

La estructura de nuestra solución quedaría algo como:

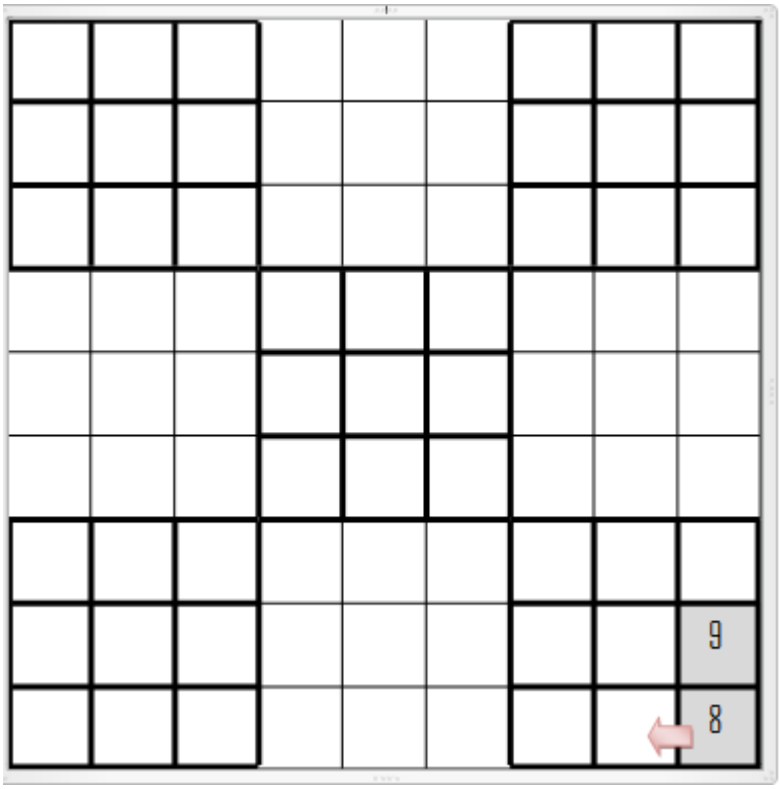


NOTA: La estructura de memorización seria la matriz en donde guardamos los valores de la tabla.

Siendo sudominoku (numeroFichas, numeroCasillas). En donde inicialmente iniciamos con (36,72) respectivamente (restando las 9 celdas que ya están usadas en su estado inicial). En el árbol cada una de las cuatro sub ramas que subdividen el árbol son las posibles rotaciones que tiene un árbol (0º, 90º, 180º, 270º).

Nuestro árbol para cuando llegamos a los casos triviales. Que es cuando tenemos sudominoku(1,2). En otras palabras una ficha y dos casillas disponibles o en otras palabras sudominoku(0,0) que es cuando ya llenamos la tabla.

El sudominoku consta de dos estructuras de almacenamiento; la primera es la matriz propia del sudoku con la variante que en vez de albergar un numero independiente en cada una de sus casillas disponibles, esta estructura va almacenar una ficha de domino la cual representa un par de números del 1 al 9 cada uno sus extremos os, esta matriz contra con un numero aleatorio de posiciones ocupadas para que luego el algoritmo se encargue de rellenar de forma bottom-up los espacios en blanco restantes



La segunda es una bolsa de fichas, la cual contiene 36 fichas que pueden ser usadas en 4 posiciones (0°, 90° 180°,270°), esta “bolsa” es una estructura de tipo tabla hash donde la clave va hacer (El mínimo de la ficha restándole el valor de uno) y se realiza la búsqueda dentro de esta con la operación (máximo número de la ficha – el mínimo número de la ficha -1) la cual da como resultado la ficha

b. Definición recursiva del valor de una solución optima

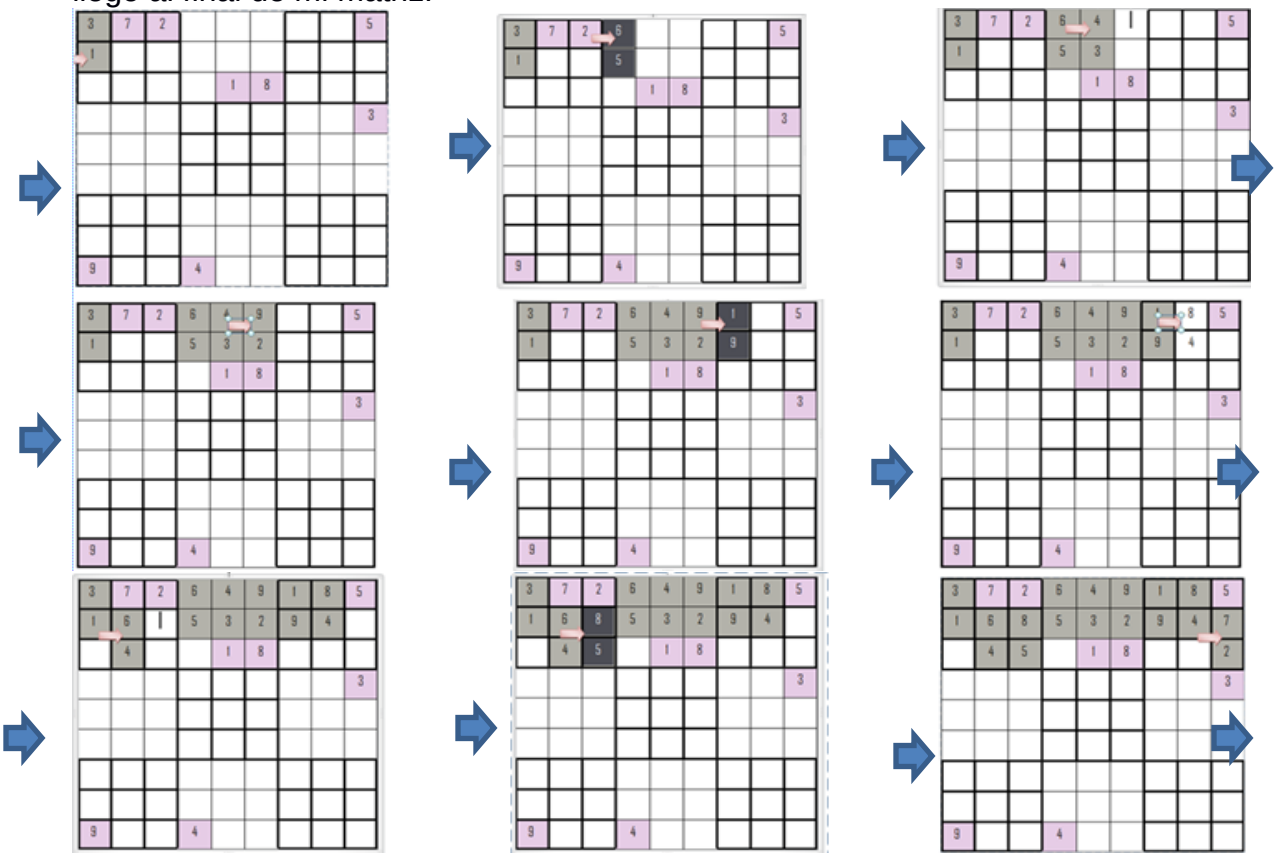
La definición recursiva de una solución óptima son los datos que almacenamos a la hora de llenar nuestra matriz. Como se puede ver en la siguiente tabla

$$\begin{pmatrix} F \\ B \\ M_{ij} \end{pmatrix} \begin{cases} si\ F == 0 \wedge B == 0 => matrix \\ si\ F \neq 0 \wedge B \neq 0 \\ \begin{cases} si\ M_i \notin ficha[i] \vee ficha[j] \wedge M_j \notin ficha[i] \vee ficha[j] \begin{cases} si\ ficha[i] \vee ficha[j] \notin SUBM_{ij} \\ => ficha \\ sudominoku\ (F-1,\ B-1,\ M_{ij}) \end{cases} \\ si\ M_i \notin ficha[i+1] \vee ficha[j+1] \wedge M_j \notin ficha[i] \vee ficha[j] \begin{cases} si\ ficha[i+1] \vee ficha[j+1] \notin SUBM_{ij} \\ => ficha \\ sudominoku\ (F,\ B,\ M_{ij}) \end{cases} \end{cases} \end{cases}$$

La función anterior representa la recursividad donde "F" representa las fichas, "B" representa los espacios en blanco y M_{ij} que es la matriz que contiene el sudominoku , la función define que si no quedan fichas ni espacios disponibles se debe retornar la matriz que contiene el sudominoku de lo contrario debe preguntar si el par de números que posee la ficha de domino se encuentran en alguna otra posición dentro de la sección de la matriz que está evaluando, si la ficha no coincide en ninguna de las factibles posiciones de la sección que se en matriz en ninguna de las posiciones que puede tomar (0°,90°,180°,270°) se pasara a evaluar con otra ficha llamando la función recursivamente

c. Calculo de una solución óptima de manera Bottom-up

Para el cálculo de la solución óptima de manera bottom-up seria ir resolviendo la matriz de fila en fila y cuando llega a 9 aumenta la columna en 1 para qué pase a la siguiente fila y cuando llega a fila 9 y columna 9 se llegó al final de mi matriz.



1) Inicia en la posición 0,0 busca la primera ficha que cree que funcione para esa posición.

2) Avanza por las filas hasta encontrar la siguiente posición vacía se encuentra que no cumple con la regla de no repetir en la misma fila y la ficha es rotada para comprobar si es un movimiento valido.

3) Sucede lo mismo que en la iteración anterior y la ficha elegida es girada para ver si es posible que encaje.

4) Avanza por la fila hasta la siguiente posición vacía y sucede lo mismo que en las iteraciones 2 y 3.

5) Avanza a la siguiente posición y coloca la ficha sin modificación alguna.

6) Siguiente posición en la fila la ficha seleccionada no cumple con una de las reglas y debe ser girada para encajar.

7) Llego al final de la primera columna aumenta en uno columnas y reinicia filas a la posición inicial, encuentra la posición ocupada por ende aumenta uno a fila, busca y selecciona una ficha, la comienza a rotar para tomar la decisión de si encaja o no.

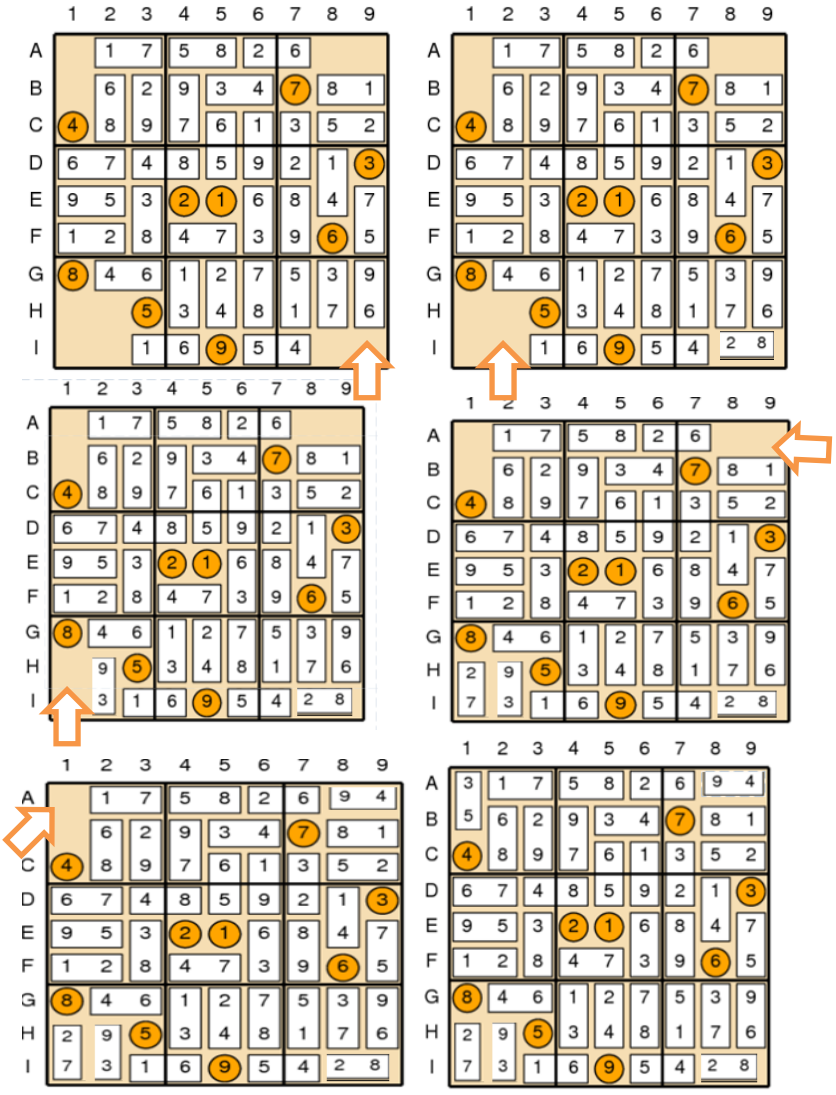
8) Siguiente posición selecciona y comienzo a rotar.

9) Avanza hasta la siguiente posición libre.

10)
   A) Sigue iterando hasta quedarse sin fichas, repitiéndose en su mayoría los casos anteriores.
   B) Sigue iterando hasta llegar una solución.
   C) Sigue iterando hasta llegar a una solución no valida.

11) Se puede observar que en la siguiente posición que va a evaluar dejara un espacio en blanco debido a que necesita un numero 9 para completar el cuadrante, pero se encuentra que el 9 está en la última casilla de la fila, entonces el algoritmo no puede llenar esta casilla y saltara a la siguiente posición disponible; este comportamiento se observa a lo largo de toda la iteración dejando numerosos espacios en blanco

d. Construcción de una solución óptima a partir de la información calculada



e. Análisis de la complejidad

El problema es de clase NP-hard lo que significa que es de forma no polinomial, (pero puede ser tratado como polinomial por medio de reducción a H); la solución que intentamos implementar con programación dinámica es de complejidad $O(n^5)$ debido a que se debe recorrer una matriz lo que tiene un costo de $O(n^2)$ y la implementación que usamos sobre la fichas las cuales deben recorrer esta matriz al tiempo también tiene un costo de $O(n^2)$; encapsulando todo esto dentro de un while cuya condición de parada es si encontró una solución óptima, de lo contrario cambia el orden de ejecución y da prioridad a las posiciones vacías, este while realiza n iteraciones; subiendo la complejidad hasta $O(n^5)$

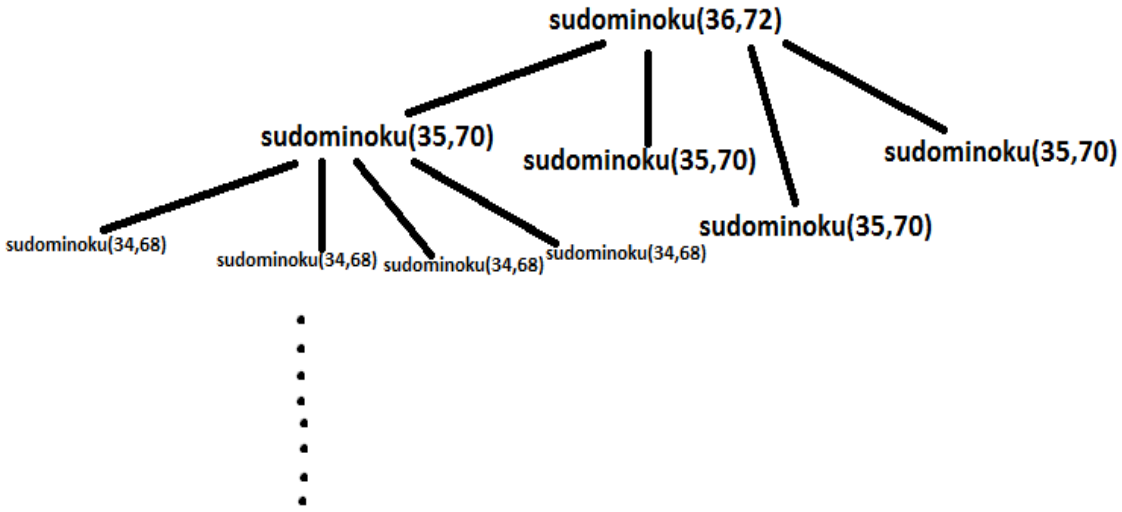
SOLUCION VORAZ

Planteamiento de la solución voraz y su estructura:

Propiedad de escogencia voraz

Inicialmente se cuenta con un tablero de sudoku con 9 posiciones predeterminadas, 72 posiciones por encontrar y 36 fichas de dominó que se usaran en dichas posiciones , en donde el árbol del problema nos va a llevar al caso trivial en el cual solo tenemos una ficha y dos espacios disponibles, si tomamos la primera ficha que ocupa este lugar sin violar ninguna de las 3 reglas (que el número no se encuentre en el sector, que el número no se encuentre en la fila y por último que el número no se encuentre en su columna), él lo colocara independientemente de si lleva a la resolución del sudominoku o no.

Subestructura óptima:



Nos encontraremos con el problema descrito anteriormente (poner la ficha en las posiciones disponibles sin violar ninguna regla) en el mejor de las 72 veces, es decir nos encontramos con el dilema de los problemas repetidos que nos llevaran a alguna de estas tres soluciones:

- El mejor caso es aquel donde se colocan las 72 fichas correctamente y se da una solución válida al sudominoku
- El caso promedio corresponde en donde se colocan las 72 fichas de manera que no den una solución óptima al sudominoku
- El peor caso es aquel en donde se usa el menor número de fichas posibles porque el algoritmo no encuentra como seguir sin violar ninguna de las reglas

Complejidad del algoritmo

El problema es de clase NP-hard lo que significa que es de tipo no-polinomial, (pero puede ser tratado como polinomial por medio de reducción a H); la solución que intentamos implementar con programación voraz es de complejidad $O(n^4)$ debido a que se debe recorrer una matriz lo que tiene un costo de $O(n^2)$ y la implementación que usamos sobre la fichas debe recorrer esta matriz al tiempo también este recorrido tiene un costo de $O(n^2)$; por ende aumenta la complejidad hasta $O(n^4)$

CONCLUSIONES

El uso de las técnicas de programación dinámica y de programación voraz no son las adecuadas para tratar problemas de este tipo, debido a que los caminos que recorre el algoritmo dentro del árbol de posibilidades lo llevan a lugares que dejan inconcluso el sudominoku; es decir este tipo de estrategia no es óptima para resolver el problema porque no permite ni tiene una herramienta que reconozca que el camino que está siguiendo no es el que lleva a la solución.

Una estrategia frente al problema de encontrar soluciones no validas es que se puedan deshacer los pasos que permitieron encontrar el estado inconcluso dentro del árbol de soluciones, la técnica descripta anteriormente hacer parte de las soluciones para búsqueda no informada, el método posee el nombre de “backtraking “ y permite solucionar los problemas de no encontrar un lugar dentro del tablero para asignar una ficha ya que simplemente volvería a un estado anterior intentando tomar otra ruta valida, llegando al extremo de volver al estado del tablero inicial para cambiar la primera decisión y así encontrar las otras posibilidades

BIBLIOGRAFIA

1) http://www.lcc.uma.es/~av/Libro/CAP5.pdf

2) http://www.diclib.com/NP-hard/show/es/es_wiki_10/18860#.WU08eGg1_IU

3) https://msdn.microsoft.com/es-es/library/aa480686.aspx