Universidad del Caribe

Materia:

Técnicas Algorítmicas

Carrera:

Ingeniería en Datos e Inteligencia Organizacional

Nombre:

Ricardo Monrreal Pool

Matrícula:

230300797

Fecha:

25 de noviembre de 2024

Ejercicio Final

Parcial 3

Desarrolla un algoritmo eficiente para resolver un Sudoku utilizando una de las técnicas algorítmicas avanzadas: programación dinámica, divide y vencerás, o algoritmos voraces. Elige la técnica que consideres más adecuada y justifica tu selección en base a la estructura del problema y la complejidad computacional esperada.

Justificación de la técnica seleccionada

Técnica: *Programación Dinámica*

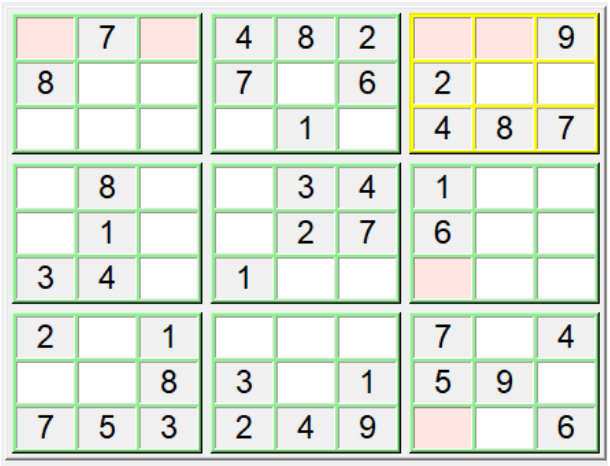
Método: *Memoización*

Esté método, la memoización es bastante útil, eficiente y adaptable a diferentes tableros de sudoku para su realización, esté método ofrece una visión general de todo el sudoku, viéndolo de una manera integral, iterando por encima de las 81 cuadrículas posibles para encontrar candidatos viables gracias a la optimización del uso de la memoización con heurísticas de decisiones de acuerdo con *n* cantidad de posibles valores que pueden entrar en una cuadrícula interna.

Siendo un caso reiterativo, en donde gracias a las heurísticas, se reduce a tan solo *O(1),* gracias a que se reducen los candidatos al mejor con menor posibilidades, sin embargo, la complejidad general de todo el método es de:

$$O(9^k), k \leq 81$$

Siendo el peor caso *k* igual a 81, que es cuando se ocupa un caso de fuerza bruta para rellenar todo un sudoku sin valores iniciales.



Esté método es eficiente y es bastante adaptable a todos los diferentes casos, ya que sin importar cuantas cuadrículas y valores iniciales haya, siempre evaluará todo y empezará en la mejor posición posible evitando cálculos innecesarios, buscando siempre el "camino" óptimo evaluando todo el sudoku, evaluando todas las opciones y siempre haciendo las iteraciones más sencillas.

Análisis de la complejidad computacional

La complejidad computacional del algoritmo utilizando la técnica de programación dinámica con el método de memoización en los casos de recursión es:

$$\boldsymbol{O}\left(\mathbf{9}^{\boldsymbol{k}}\right), \mathbf{0} \geq \ \boldsymbol{k} \leq \mathbf{81}$$

Siendo *k* el valor posibles iteraciones por celdas vacías, volviendo exponencial los 9 posibles valores dentro de todo el sudoku, tomando *k* valores de entre 0 a 81 valores, siendo 81 el peor caso con celdas vacías en todo el sudoku. Está técnica toma todo el sudoku debido a que de acuerdo a las celdas vacías y las posibilidades que tenga una celda, pero regresará a cierto *n* de pasos para lograr solucionar de acuerdo a la heurística de selección de celdas.

Para el análisis de la complejidad computacional de **SELECCIONAR** cada celda, es decir, concluir en saber cuál es el candidato óptimo para empezar a probar los posibles valores que pueden entrar, utilizando una heurística de restricción de candidatos, minimizando la profundidad de búsqueda. Es necesario entender que para cada cuadrícula hay que tomar en cuenta las 8 posibles restricciones de la fila, columna y cuadrícula en la que se encuentre el candidato, en donde mayormente, 8 posibles

restricciones es el mejor caso, debido a que la heurística buscaría el candidato con una sola posibilidad, es decir, en donde su cuadrícula, su columna y fila esté completa.

$$\boldsymbol{O(8) + O(8) + O(8) = O(24)}$$
$$\boldsymbol{O(24) = O(n)}$$

Al final convirtiéndose la complejidad de encontrar al candidato de acuerdo con la heurística de restricción, se vuelve *O(n)*.

Nuestra complejidad computacional completa de toda la ejecución del código sería la siguiente:

$$\boldsymbol{O\left(9^k\right) * O(n) = O(n * 9^k)}$$

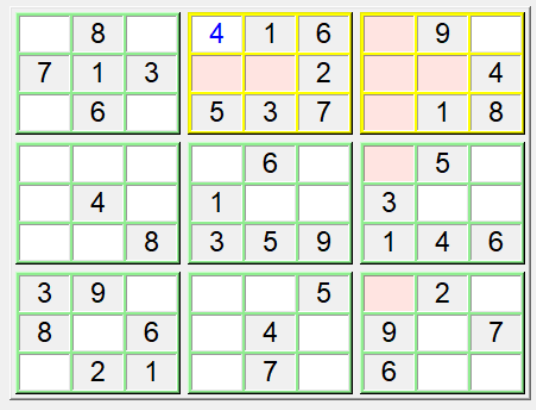
Resultados obtenidos

La técnica de programación dinámica por medio del método de memoización tuvo un tiempo promedio de 20.2 minutos de acuerdo con 5 casos en un nivel difícil de ejecución (20 valores insertados).

El algoritmo logró resolver de forma correcta el sudoku en diferentes ejecuciones, además que toma una interfaz gráfica bastante atractiva logrando captar las iteraciones sobre cada cuadrícula para encontrar los valores ya puestos, para luego empezar en el candidato con menor número de posibilidades, para ir generando un árbol de búsqueda cerrado e irlo mejorando con cada paso que da.

	8			1	6		9	
7	1	3			2			4
	6		5	3	7		1	8
				6			5	
	4		1			3		
		8	3	5	9	1	4	6
3	9				5		2	
8		6		4		9		7
	2	1		7		6		

En los siguientes pasos se logra ver como va buscando cada mejor candidato para empezar a iterar desde ahí gracias a que puede solo poner un valor ahí, siendo el candidato perfecto.



Al encontrar algún error, el código retrocede tantos *n* pasos sean necesarios cambiando las probabilidades ya antes generadas, evitando aquellas que ya usó.

Además, como último agregado a la interfaz, se agregó una consola de eventos que muestra en qué cuadrícula se trabaja, en qué celda se trabaja y con qué posibilidades.

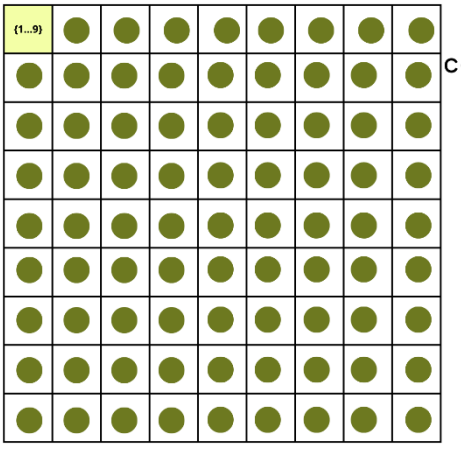
```
Analizando celda [1,3] en subcuadrícula [1,1]
Posibilidades: [5]
→ Probando 5 en [1,3]

  Analizando celda [1,7] en subcuadrícula [1,3]
  Posibilidades: [6]
  Posibilidades: [6]
```

Técnicas no elegidas:

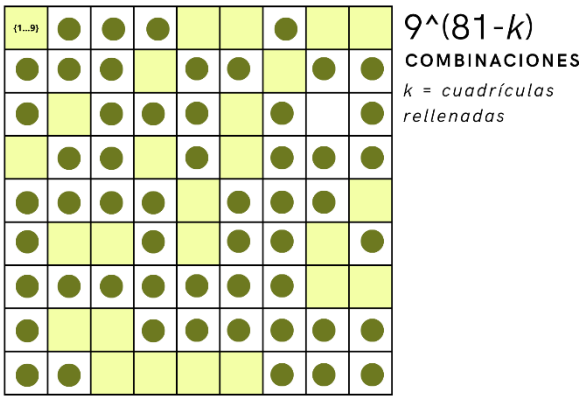
- Algoritmos voraces:

La técnica de algoritmos voraces no fue escogida debido a su extensa magnitud en cuanto a iteraciones se refiere. Algoritmos como Prim, Kruskal

o Dijkstra usan métodos de árboles binarios, conectando nodos y aristas entre sí para poder tomar la vía más optima hasta el resultado esperado. La aplicación de la técnica en este ejercicio del sudoku fue hecha a similitud, siendo que el nodo inicial, en este caso, la cuadrícula (1,1) de todo el sudoku de 9x9, toma conexiones entre todas las demás cuadrículas, es decir, 81 posibles, de esas 81 posibles existen 9 combinaciones 81 posibilidades, a lo que hace decir, en estadística, 9 elevado a 81 combinaciones, el cual sería un valor aproximado de 10^80 (cerca de: 22209113248609177041058454814754205646612382186249365825622521664322868379 84 posibilidades), este enorme valor es astronómico para la resolución del sudoku, debido a su gran cantidad de iteraciones y posibilidades.



Este gráfico representa mejor como es que intentan los algoritmos voraces resolver un sudoku de 9x9. La cuadrícula (1,1) genera conexiones con todas las 80 cuadrículas restantes, cada cuadrícula es un nodo y las aristas y sus pesos es el valor posible entre 1 a 9. Cuando las iteraciones comienzan, tan solo la primera iteración genera 9^81 combinaciones, intentando buscar que valor que pueda entrar, probando una y otra vez cada valor posible dentro de las 80 cuadrículas restantes, esto lo vuelve ineficiente, los códigos que se implementan con algoritmos voraces no acaban jamás de ejecutarse debido a

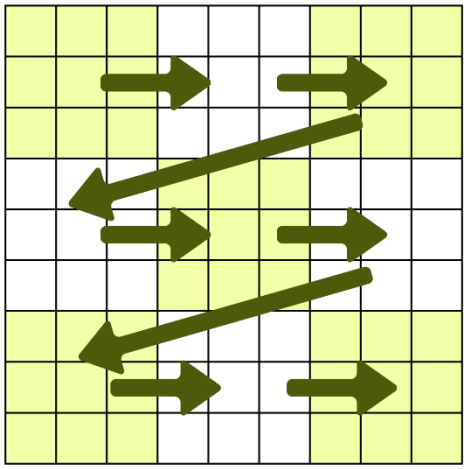


su extensión y casi infinito número de iteraciones para resolver tan solo una cuadrícula.

Aun así, si en cierto caso el sudoku tuviese valores ya rellenados, la cantidad de iteraciones por nodo y arista del sudoku sería inmenso, siendo así ineficiente de nuevo.

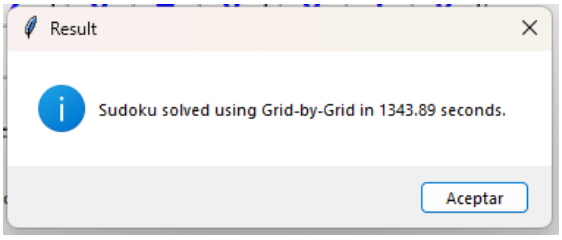
- Divide y vencerás:

  La técnica de divide y vencerás fue propuesta bajo la estrategia de dividir toda la cuadrícula de 9x9 de un sudoku original a subcuadrículas de 3x3, haciendo un total de 9 subcuadrículas que serán resueltas individualmente, avanzando en cada cuadrícula hasta que pueda ser resuelto por completo todo el sudoku.



  Está técnica ocupando divide y vencerás puede parecer eficiente desde un punto de vista teórico, sin embargo se enfrenta a problemas de recursividad e iteraciones en largas longitudes ya que la evaluación se hace por subcuadrícula 3x3 en una cuadrícula interna de la subcuadrícula, avanzando en una complejidad O(1) por cada fila de 9 elementos y columna de 9 elementos con 9 posibilidades (1 – 9), convirtiéndose en O(1), avanzando iteradamente, pero en ciertos casos se vuelve exponencial en debido al retroceso que incluye cuando se encuentra con un error, un valor no óptimo para esa cuadrícula interna, retrocede un paso, dos pasos o los pasos que sean necesarios hasta que se vuelva correcto, incluso regresando hasta el inicio.

La complejidad computacional de esté algoritmo conlleva un proceso de resolución de subcuadrículas que se convierte en exponencial cuando no encuentra la solución óptima de una subcuadrícula, siendo uno de los peores casos una complejidad de $O(9^{81})$, volviéndolo muy lento ya que no logra verificar cuál es el mejor caso posible, únicamente itera por todos los valores posibles (del 1 al 9), hasta que logre solucionarlo completamente, esto genera búsquedas de tiempo extensas.



La resolución promedio (en minutos) de está técnica de divide y vencerás, de 4 casos con un nivel de dificulta difícil (20 valores insertados dentro del sudoku inicial) fue de: **24.4 minutos.**

					8	7	5	
7					9			2
		2		5				
		6				5		
				2				6
2	5	1			6		9	8
				9	4	2		1
4	1			6	2			
	2		1	7			4	

El método divide y vencerás, por medio de cuadrícula por cuadrícula no es totalmente eficiente debido a su recursión anidada en 3 niveles, posibilidad de fila, columna y subcuadrícula interna, necesitando regresar tantos *n* pasos sean necesarios para resolver el sudoku.