Kodolányi János Egyetem

SUDOKU MEGOLDÁSI MÓDSZEREK

Megoldási módszerek programozása különböző fejlesztői platformokon.

Konzulens:

Dr. Pitlik László

Késztette:

Kovács Bálint, Kovács János, Mayerhöffer Orsolya, Csötönyi Viktória

Üzemmérnök-informatikus alapképzési szak

LIN30 - Programozási alapelvek és módszertanok

Budapest

2024.

Kodolányi János Egyetem

SUDOKU SOLVING METHODS

Programming solution methods on various development platforms.

Supervisor:

Dr. László Pitlik

Created by:

Bálint Kovács, János Kovács, Orsolya Mayerhöffer, Viktória Csötönyi

Computer Science Operational Engineering Bachelor’s Degree

LIN30 - Programming principles and methodologies

Budapest

2024.

**Tartalom**

[1. Tartalom 1](#_Toc164672765)

[2. Előszó 3](#_Toc164672766)

[3. Preface 4](#_Toc164672767)

[1. Az alkalmazott módszerek bemutatása 5](#_Toc164672768)

[I. Megoldás *go* programnyelven írt algoritmus segítségével (Kovács János - IUO13K) 5](#_Toc164672769)

[A futtatókörnyezet paraméterei és alkalmazott szoftverek 5](#_Toc164672770)

[A módszer és előzmények leírása 5](#_Toc164672771)

[GO kód elkészítése 5](#_Toc164672772)

[A program felépítése 7](#_Toc164672773)

[A kód tesztelése 9](#_Toc164672774)

[II. Megoldás VBA-ban írt algoritmus segítségével (Kovács Bálint – OE64UF) 11](#_Toc164672775)

[A futtatókörnyezet paraméterei és alkalmazott szoftverek 11](#_Toc164672776)

[A megfelelő módszer keresése 11](#_Toc164672777)

[A VBA-kód elkészítése 12](#_Toc164672778)

[A VBA-kód felépítése és működése 14](#_Toc164672779)

[A kód tesztelése 15](#_Toc164672780)

[Az Excel tábla kialakítása és a felhasználói élmény optimalizálása 15](#_Toc164672781)

[III. Megoldás MS Excel Solver beépülő modul segítségével (Mayerhöffer Orsolya – IGJ96Q) 20](#_Toc164672782)

[Bevezetés 20](#_Toc164672783)

[1. Fejlesztési környezet 20](#_Toc164672784)

[1.1 Hardver 20](#_Toc164672785)

[1.2 Szoftver 20](#_Toc164672786)

[2. Első 4x4 megoldási módszer 20](#_Toc164672787)

[2.1 Google keresés 20](#_Toc164672788)

[2.2 ChatGPT 20](#_Toc164672789)

[2.3 Értekezletről készült felvétel 22](#_Toc164672790)

[2.4 Brainstorming osztálytársakkal 23](#_Toc164672791)

[3. Második megoldási módszer 24](#_Toc164672792)

[3.1 MIAU 24](#_Toc164672793)

[4. Harmadik megoldási módszer 26](#_Toc164672794)

[5. 9x9 megoldási módszer 28](#_Toc164672795)

[5.1 ChatGPT 29](#_Toc164672796)

[6. Konklúzió 29](#_Toc164672797)

[IV. Sudoku Excel Solver megoldás 4x4 és 6x6 táblázat alapján (Csötönyi Viktória - J908Q3) 30](#_Toc164672798)

[1. Bevezetés 30](#_Toc164672799)

[2. Célkitűzés 30](#_Toc164672800)

[3. Excel Solver 31](#_Toc164672801)

[3.1 Excel solver 4x4 táblázat 31](#_Toc164672802)

[3.2 Sudoku Solver 6x6 táblázat 32](#_Toc164672803)

[3.3. Eredmény 6x6 sudokunál 32](#_Toc164672804)

[4. Összegzés 34](#_Toc164672805)

[2. Az alkalmazott módszerek összehasonlítása OAM elemzés segítségével 35](#_Toc164672806)

[Objektumok: 35](#_Toc164672807)

[Attribútumok: 35](#_Toc164672808)

[3. Tantárgyi kapcsolatok 36](#_Toc164672809)

[LIN029 - Operációs rendszerek 36](#_Toc164672810)

[LIN026 - Matematikai alapok 36](#_Toc164672811)

[LSK001 - Kultúra, sport, munkahelyi jóllét 1. 36](#_Toc164672812)

[LKN490 - Európai civilizáció és identitás 36](#_Toc164672813)

[LIN028 - Hálózatok és számítógép architektúrák 36](#_Toc164672814)

[LIN027 - Adatszerkezetek és algoritmusok 36](#_Toc164672815)

[LNE178 - A jog szerepe a modern társadalmakban 37](#_Toc164672816)

[4. Összegzés 38](#_Toc164672817)

[5. Summary 38](#_Toc164672818)

[6. Rövidítésjegyzék, hivatkozásjegyzék, irodalomjegyzék 39](#_Toc164672819)

[6.1 Csötönyi Viktória\_J908Q3: 39](#_Toc164672820)

[7. Mellékletek 40](#_Toc164672821)

# Előszó

**A Sudokuról röviden:** A hagyományos sudoku egy logikai rejtvény, amely egy 9×9-es rácsból áll, tovább osztva 3×3-as négyzetre. A cél az, hogy minden sorban, oszlopban és 3×3-as négyzetben az 1-től 9-ig terjedő számok mindegyike egyszer szerepeljen, ügyelve arra, hogy ne ismétlődjenek. A játék kezdetén néhány szám előre ki van töltve, ezek segítenek a többi szám helyének megtalálásában. A sudoku megoldása logikai gondolkodást és türelmet igényel, mivel nem szükséges matematikai tudás hozzá. Ez a játék kiváló agyi edzést nyújt, fejleszti a problémamegoldó készséget és a figyelmet.

<https://hu.wikipedia.org/wiki/Sz%C3%BAdoku>

A dokumentum további részében a különböző Sudoku megoldási módszerek megszületéséhez vezető utakat járjuk körbe, majd elemezzük és egymással összehasonlítjuk az elkészült megoldórendszerek sikerességi rátáját és teljesítményét.

A részletes összehasonlítás egy külön OAM Excel-táblában kapott helyet.

# Preface

**A brief overview of Sudoku:** Traditional Sudoku is a logic-based puzzle that consists of a 9×9 grid, further divided into 3×3 squares. The objective is to fill every row, column, and 3×3 square with the numbers 1 through 9, ensuring that each number appears exactly once, without any repetition. At the start of the game, some numbers are already filled in, which helps in determining the placement of the remaining numbers. Solving sudoku requires logical thinking and patience, as no mathematical knowledge is necessary. This game provides excellent mental exercise, enhancing problem-solving skills and attention.

<https://hu.wikipedia.org/wiki/Sz%C3%BAdoku>

The remainder of the document explores the various methods of solving Sudoku, tracing the development of these techniques, then analyzing and comparing the success rates and performance of the completed solving systems.

The detailed comparison is placed in a separate OAM Excel sheet.

# Az alkalmazott módszerek bemutatása

## Megoldás *go* programnyelven írt algoritmus segítségével (Kovács János - IUO13K)

(v20240417\_2248)

### A futtatókörnyezet paraméterei és alkalmazott szoftverek

* Macbook pro M1 ARM, 16GB
* OS: Mac OS 14.4
* Office: Google sheet és MS Excel 16.83
* Github
* VS Code 1.87.2, aktuális go pluginokkal

### A módszer és előzmények leírása

Régóta szemeztem a go nyelvvel és most úgy éreztem ez jó alkalom lesz megismerni a szintaktikáját.

Az backtrack rekurzív algoritmus a korábbi excel solveres munkálataim során már megismertem, így azt használva fejlesztettem a megoldásomat. Volt egyfajta evolúció, ezek a git-ben megtalálhatóak.

GPT 4.0-át főleg a még nem ismert go nyelvi elemek keresése közben és a dokumentálás közben használtam.

### GO kód elkészítése

A kód az alábbi helyen található:

<https://github.com/kovacs213janos/sudokugo>







### A program felépítése

A program a klasszikus backtracking algoritmust implementálja Go nyelven. Az algoritmus próbálkozik minden lehetséges szám elhelyezésével a tábla minden üres cellájába, miközben biztosítja, hogy a megoldás érvényes maradjon. A program felhasználóbarát, kéri a felhasználó bevitelét, és visszajelzést ad a folyamat során, így nem csak egy hatékony Sudoku solver, hanem interaktív is. Az algoritmus iteratív megközelítése és a memória használat kiírása segít megérteni az algoritmus futtatásának komplexitását és erőforrás-igényét. A program hasznos lehet a backtracking algoritmusok tanulmányozására és az algoritmikus gondolkodásmód fejlesztésére. A kód strukturált és jól kommentezett, ami elősegíti a megértését és a további fejlesztés lehetőségét.

A program a következő elemekből áll:

#### Állandók és Globális Változók:

* Size: A Sudoku táblázat méretét definiáló állandó, ebben az esetben 9.
* Iteraciok: Globális változó, amely nyomon követi a rekurziós hívások számát a Sudoku tábla megoldása során.

#### Sudoku Táblázat Struktúra:

* type Sudoku [Size][Size]int: Egy 9x9-es tömböt definiál, amely egy Sudoku táblát képvisel.

#### Validációs Függvény (Validalas):

* Ellenőrzi, hogy egy adott számot elhelyezhetünk-e biztonságosan egy adott sorban és oszlopban, valamint a megfelelő 3x3-as részrácson belül.

#### Solver Függvény (Solver):

* Megpróbálja megoldani a Sudoku táblát a backtracking módszerrel.
* Végigiterál a tábla összes celláján, és ha üres cellát talál (értéke 0), megpróbálja beírni a lehető összes számot 1-től 9-ig.
* Ha egy szám helyesnek bizonyul, rekurzívan hívja önmagát a következő üres cellára lépve.
* Sikertelen próbálkozás esetén visszaállítja a cellát üresre (backtracking).
* Ha az összes cella ki van töltve, a tábla megoldottnak tekinthető.

#### Kiíratás Függvény (kiiratasTabla):

* Kiírja a Sudoku tábla aktuális állapotát.

#### Szám Bekérő Függvény (szambekeres):

* Bekéri a felhasználótól azt a számot, amely megadja, hány cellát kell üresen hagyni a táblán.

#### Random Törlés Függvény (randomRemove):

* Véletlenszerűen eltávolítja a megadott számú cellát a táblából, hogy feladványt hozzon létre.

#### Main Függvény (main):

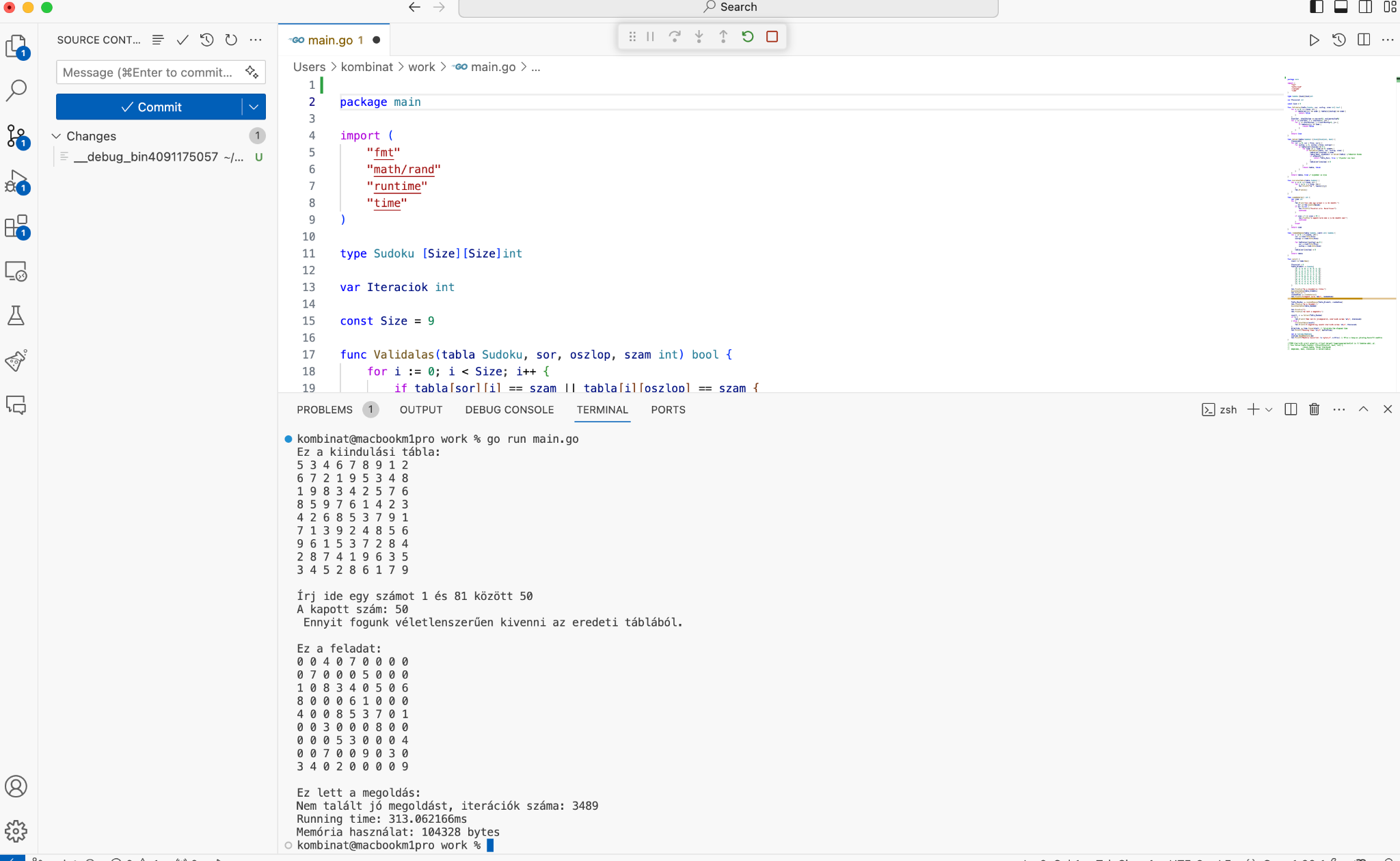
* A program belépési pontja.
* Inicializálja a táblát egy előre megadott megoldással.
* Meghívja a szambekeres függvényt a felhasználói interakcióhoz.
* Létrehoz egy feladványt a randomRemove függvény segítségével.
* Megoldja a feladványt a Solver függvény hívásával.
* Kiírja az eredményt és statisztikai információkat: a megoldáshoz szükséges iterációk számát és a program futásának idejét.
* Lekérdezi a memória használatot és kiírja azt.

#### Időmérés és Memóriahasználat:

* A main függvény méri a futási időt és lekérdezi a memóriahasználatot a runtime csomag segítségével.

### A kód tesztelése

Ez fejlesztés közben és később manuálisan történt, a közben talált hibák azonnal javításra kerültek.



## Megoldás VBA-ban írt algoritmus segítségével (Kovács Bálint – OE64UF)

(v20240417\_2248)

### A futtatókörnyezet paraméterei és alkalmazott szoftverek

* CPU: Intel Core i5-11600K @ 3,90 GHz
* RAM: 16 GB DDR4
* Háttértár: Samsung 970 EVO Plus 500GB NVMe
* OS: Windows 11 Pro, build: 22631.3296
* Office: Microsoft Office Professional Plus 2021

### A megfelelő módszer keresése

A feladat egy sudoku-megoldó rendszer elkészítése volt, Excel Solver segítségével. Mivel az Excel Solvert nem sikerült megfelelően paraméterezni a 9×9-es sudoku feladványok hatékony megoldásához, a ChatGPT 4 (továbbiakban: **GPT**) az alábbi megoldási módszereket javasolta lehetséges alternatívaként:

#### Brute force algoritmus

A brute force vagy teljes keresési algoritmus egyszerűen végigpróbálja az összes lehetséges kombinációt a rács kitöltésére, anélkül, hogy logikát vagy optimalizálási stratégiát alkalmazna. Ez az algoritmus rendkívül erőforrás-igényes.

A Sudoku valós brute force megoldása, amely egy 9×9-es grid minden cellájára és minden lehetséges számára kiterjed, a gyakorlatban nem megvalósítható a rendkívül nagy számú kombináció miatt.

A legrosszabb esetben a következő számú kombinációt kellene kipróbálni:

A Sudoku minden cellájába 1-től 9-ig terjedő számokat helyezhetünk, tehát 9 lehetőség van minden cellára. Mivel 81 cella van (9 sor × 9 oszlop), így a teljes kombinációk száma: 981, pontos számmal kifejezve:

*196627050475552913618075908526912116283103450944214766927315415537966391196809*

A tiszta brute force megközelítés a Sudoku megoldására nem praktikus. A backtracking algoritmus sokkal hatékonyabb, mivel csak az érvényes kombinációkat próbálja ki, figyelembe véve a Sudoku szabályait, ezáltal jelentősen csökkenti a szükséges próbálkozások számát.

[*https://en.wikipedia.org/wiki/Brute-force\_search*](https://en.wikipedia.org/wiki/Brute-force_search)

#### CSP algoritmus

A Constraint Satisfaction Problem (CSP) algoritmusok, mint a backtracking algoritmus kiterjesztése, olyan technikákat alkalmaznak, mint a változók értékkészletének szűkítése (domain reduction), az előretekintés (forward checking), és/vagy a konzisztencia algoritmusok (mint az AC-3 algoritmus). Ezek az algoritmusok a probléma korlátjainak kihasználásával próbálják csökkenteni a keresési teret és hatékonyabban találnak megoldást, ezért kiválóan alkalmasak a Sudokuhoz hasonló logikai rejtvények megoldására.

A (CSP) algoritmus teljes körű implementálása egy 9×9-es Sudoku esetében a VBA környezetben nagyon összetett feladat, ami magában foglalja az értékkészletek kezelését, korlátok dinamikus alkalmazását és a keresési stratégiák (pl. backtracking) implementálását.

[*https://en.wikipedia.org/wiki/Constraint\_satisfaction\_problem*](https://en.wikipedia.org/wiki/Constraint_satisfaction_problem)

*A CSP módszer a szerző által szubjektíven megítélt szépsége miatt ugyan jelen írásban említésre került, azonban összetettsége okán kidolgozása nem valósult meg. A CSP módszertől lényegesen egyszerűbben implementálható VBA alatt egy klasszikus backtracking algoritmus, amely esetében a kódolási idő / kód teljesítmény arány az aktuális feladat megoldásához optimálisabb.*

#### Backtracking algoritmus

Ez az algoritmus egy klasszikus rekurzív megközelítés, amely próbálkozik minden lehetséges értékkel a rács üres celláiban, és visszalép (backtrack), ha az aktuális kitöltés nem vezet megoldáshoz. A backtracking algoritmus nagyon hatékony a Sudoku-féle problémák megoldására, mivel képes végigiterálni az összes lehetséges konfigurációt anélkül, hogy felesleges kombinációkat vizsgálna.

[*https://en.wikipedia.org/wiki/Backtracking*](https://en.wikipedia.org/wiki/Backtracking)

***A továbbiakban a backtracking algoritmussal foglalkozunk.***

### A VBA-kód elkészítése

A VBA-kód a GPT segítségével került megírásra.

A következő prompt megadása után, a GPT legenerálja a szükséges kód alapját:

„Tudnál írni egy Backtracking Algoritmust a sudoku 9x9-hez Excel VBA-ban? Az adatokat a C12:K20-ból vegye, a megoldási grid a Q12:Y20. A VBA kódban a változók suffixe tartalmazza a változó típust rövidítve, a változó nevétől egy "\_"-al elválsztva (pl.: \_str, \_int, \_bool). A functionok prefixe legyen "fun".”

A generált kód felülvizsgálata során csak minimális módosításokra van szükség, a kód alapvetően működőképes.

A kód egy újonnan létrehozott modulba (.bas) kerül beillesztésre.

**A kód:**

Function funIsSafe(grid As Variant, row As Integer, col As Integer, num As Integer) As Boolean

Dim idx\_int As Integer

Dim jdx\_int As Integer

Dim boxStartRow\_int As Integer

Dim boxStartCol\_int As Integer

'Sor ellenőrzés

For idx\_int = 1 To 9

If grid(row, idx\_int) = num Then

funIsSafe = False

Exit Function

End If

Next idx\_int

'Oszlop ellenőrzés

For idx\_int = 1 To 9

If grid(idx\_int, col) = num Then

funIsSafe = False

Exit Function

End If

Next idx\_int

'3x3 grid ellenőrzése

boxStartRow\_int = row - ((row - 1) Mod 3)

boxStartCol\_int = col - ((col - 1) Mod 3)

For idx\_int = 0 To 2

For jdx\_int = 0 To 2

If grid(boxStartRow\_int + idx\_int, boxStartCol\_int + jdx\_int) = num Then

funIsSafe = False

Exit Function

End If

Next jdx\_int

Next idx\_int

funIsSafe = True

End Function

Sub SolveSudoku()

Dim grid As Variant

Dim i As Integer, j As Integer

Dim ws As Worksheet

Set ws = ThisWorkbook.Sheets("9x9 VBA")

ReDim grid(1 To 9, 1 To 9) As Integer

' Beolvasás tömbbe Q12:Y20

For i = 1 To 9

For j = 1 To 9

grid(i, j) = ws.Cells(11 + i, 2 + j).Value

Next j

Next i

' Megoldás

If funSolveSudoku(grid) Then

For i = 1 To 9

For j = 1 To 9

ws.Cells(11 + i, 16 + j).Value = grid(i, j)

Next j

Next i

MsgBox "A feladvány megoldása sikeres!", vbInformation

Else

MsgBox "A feladványt nem sikerült megoldani.", vbCritical

End If

End Sub

Function funSolveSudoku(grid As Variant) As Boolean

Dim row\_int As Integer

Dim col\_int As Integer

Dim num\_int As Integer

Dim isEmpty\_bool As Boolean

isEmpty\_bool = True

'Első üres cella keresése

For row\_int = 1 To 9

For col\_int = 1 To 9

If grid(row\_int, col\_int) = 0 Then

isEmpty\_bool = False

GoTo NextStep

End If

Next col\_int

Next row\_int

NextStep:

If isEmpty\_bool Then

funSolveSudoku = True

Exit Function

End If

'Összes variáció kipróbálása

For num\_int = 1 To 9

If funIsSafe(grid, row\_int, col\_int, num\_int) Then

grid(row\_int, col\_int) = num\_int

If funSolveSudoku(grid) Then

funSolveSudoku = True

Exit Function

End If

grid(row\_int, col\_int) = 0 ' Visszaállítás

End If

Next num\_int

funSolveSudoku = False

End Function

### A VBA-kód felépítése és működése

A kód két fő részből áll: a **funIsSafe** functionből, ami ellenőrzi, hogy egy adott szám érvényes-e egy adott pozícióban, és a **SolveSudoku** subból, ami maga a megoldó algoritmus. Lépésről lépésre haladva a kód működése a következő:

#### „funIsSafe” funcion

Sor ellenőrzése: A függvény végigiterál az adott sor összes elemén (*row*, fix oszlop változik) és ellenőrzi, hogy a *num* szám szerepel-e már ebben a sorban. Ha igen, *False* értékkel tér vissza, jelezve, hogy az adott szám nem helyezhető el biztonságosan.

Oszlop ellenőrzése: Hasonlóan az előzőhöz, itt az adott oszlop (col, fix sor változik) minden elemét ellenőrzi, hogy tartalmazza-e a *num* számot. Ha igen, a függvény *False*-sal tér vissza.

3x3 grid ellenőrzése: Kiszámítja a kis 3×3 doboz bal felső sarok cellájának (boxStartRow\_int és boxStartCol\_int) koordinátáit, amelybe a vizsgált cella esik. Ezután ellenőrzi, hogy a *num* szám szerepel-e ebben a 3×3 dobozban. Ha igen, *False* értékkel tér vissza.

Ha egyik ellenőrzés során sem talált problémát, akkor *True* értékkel tér vissza, jelezve, hogy a szám biztonságosan elhelyezhető az adott pozícióban.

#### „SolveSudoku” sub

Munkalap beállítása és rács inicializálása: Kiválasztja az aktuális munkalapot ("9x9 VBA") és létrehoz egy 9×9-es tömböt a Sudoku tábla tárolására.

Tábla beolvasása: Beolvassa a Sudoku táblát a munkalapról a grid tömbbe. A feladvány tábla a C12:K20 tartományban található a munkalapon.

Megoldás keresése: Meghívja a funSolveSudoku függvényt, amely rekurzívan próbálja meg kitölteni a táblát.

Eredmény kiírása: Ha a funSolveSudoku igazzal tér vissza (sikerült megoldani a feladványt), akkor a megoldást kiírja a munkalap Q12:Y20 tartományába, és információs üzenet jelenik meg. Ha hamissal tér vissza (nem sikerült megoldani), egy kritikus hibaüzenet jelenik meg.

#### „funSolveSudoku” function

Első üres cella keresése: Végigiterál a táblán, és keres egy üres cellát (aminek az értéke 0).

Ha nincs üres cella: Ha minden cella kitöltött (nincs több üres hely), akkor sikeresen megoldotta a Sudoku-t, True-val tér vissza.

Szám próbálgatása: Próbálja beilleszteni az 1-től 9-ig terjedő minden számot az üres cellába, amit az előző lépésben talált. Minden egyes szám esetén ellenőrzi a *funIsSafe* függvénnyel, hogy az adott szám érvényes-e.

Rekurzió: Ha egy számot sikeresen be lehet helyezni (azaz a *funIsSafe* igazat ad vissza), akkor rekurzívan meghívja önmagát a frissített táblával, hogy próbálja megoldani a következő cellát.

Backtracking: Ha egy adott szám beillesztése után nem sikerül megoldani a feladványt (a rekurzív hívás hamissal tér vissza), akkor visszaállítja az üres cella értékét 0-ra (visszavonja az előző lépést) és kipróbál egy másik számot.

A megoldás keresése addig folytatódik, amíg vagy meg nem találja a teljes megoldást, vagy bebizonyosodik, hogy a tábla nem oldható meg a jelenlegi állapotából kiindulva.

### A kód tesztelése

A kód tesztelése az Interneten, Google kereső segítségével fellelhető Sudoku feladványokkal történt.

A feladványok teszteléséhez a C12:K20 tartományban szükséges megadni a rendelkezésre álló számokat, majd megnyomni a „MEGOLDÁS!”- gombot.

A tesztelés során nem sikerült olyan feladványt találni, amelyet ne tudott volna megoldani az algoritmus.

Átlagos futásidő: 1-3 perc.

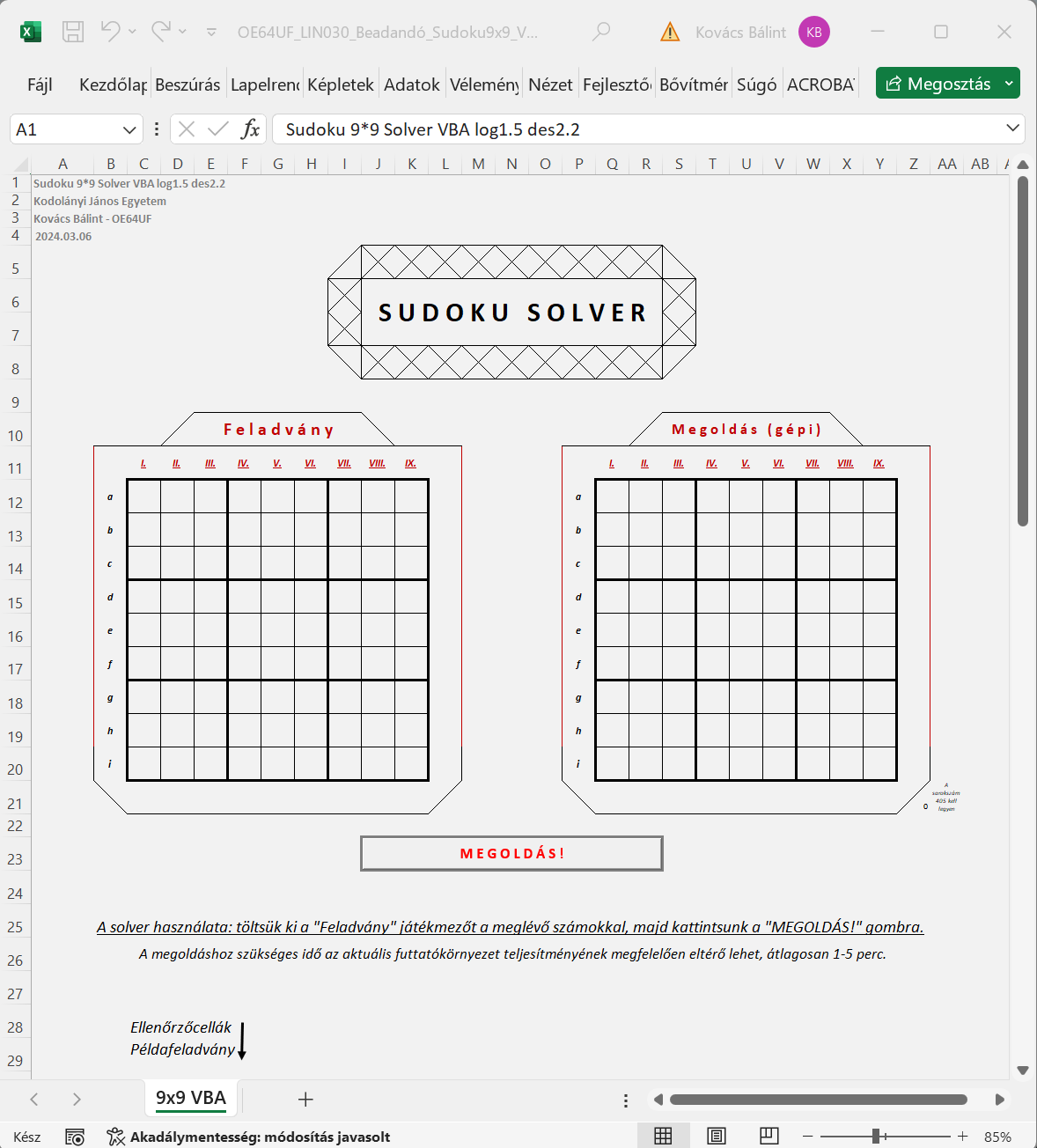
[*https://www.google.com/search?q=sudoku+with+answers*](https://www.google.com/search?q=sudoku+with+answers)

### Az Excel tábla kialakítása és a felhasználói élmény optimalizálása

Az elkészült algoritmus megfelelő minőségű keretbe (GUI-ba) foglalása sok esetben legalább olyan fontos (ha nem fontosabb), mint magának az algoritmusnak a kidolgozása.

A külalak az alábbi szempontok figyelembevételével került kialakításra:

* Felhasználóközpontúság: a tervezés során mindig a felhasználó igényeit és visszajelzéseit kell szem előtt tartani. A táblázatnak könnyen navigálhatónak és érthetőnek kell lennie minden típusú felhasználó számára.
* Konzisztencia: a felületelemek, mint a gombok, feliratok, keretek stílusa legyen konzisztens az egész táblázaton belül, hogy a felhasználók gyorsan megtanulhassák és hatékonyan használhassák azt.
* Egyszerűség: kerülni kell a felesleges bonyolultságot. A táblázat megjelenített részének csak a szükséges elemeket kell tartalmaznia, hogy a felhasználó könnyen elérje a kívánt funkciókat anélkül, hogy zavarba jönne.
* Adaptivitás: A táblázatnak alkalmazkodnia kell különböző képernyőméretekhez és felbontásokhoz, hogy a különböző eszközökön is megfelelően jelenjen meg.
* Színek és betűtípusok: A színek és betűtípusok megfelelő választása javíthatja a felhasználói élményt, segíthet a fontos elemek kiemelésében és javíthatja az olvashatóságot.



*Képernyőkép az elkészült felhasználói felületről*

#### Az Excel ablakának méretezése

A táblázat megnyitásakor célszerű a sallangoktól, segédcelláktól mentes nézettel indítani, megkönnyítve a felhasználónak a megfelelő területre való fókuszálást.

A méretezést dinamikusan is elvégezhetjük az aktuális képernyőfelbontás függvényében, azonban jelen esetben egy statikus, 600×660 pixeles méretre kényszerítjük az Excel ablakát. A felbontás megválasztása során evidenciának vesszük, hogy a programot nem fogják 800×600-as felbontástól kisebb értéken futtatni.

A méretezéshez szükséges VBA-kód:

Dim OpenWidth\_int As Integer

Dim OpenHeight\_int As Integer

Private Sub Workbook\_Open()

OpenWidth\_int = Application.Width

OpenHeight\_int = Application.Height

With Application

.WindowState = xlNormal

.Width = 600

.Height = 660

End With

End Sub

Az Excel bezárását követően vissza kell állítani az eredeti ablakméretet, különben az átméretezés a később megnyitandó Excel táblázatokra is kihat:

Private Sub Workbook\_BeforeClose(Cancel As Boolean)

With Application

.Width = OpenWidth\_int

.Height = OpenHeight\_int

End With

End Sub

#### A menüszalag bezárása

Esetünkben a menüszalag bezárásra kerül a táblázat megnyitásakor, ugyanis a felhasználónak nem kell használni azt a Sudoku Solver futtatása közben. Kilépéskor a menüszalagot újra meg kell nyitni, különben a menüszalag bezárása a később megnyitandó Excel táblázatokra is kihat:

Dim IsRibbonShowedBefore\_bool As Boolean

Private Sub Workbook\_Open()

If Application.CommandBars("Ribbon").Height >= 150 Then

IsRibbonShowedBefore\_bool = True

Application.CommandBars.ExecuteMso ("MinimizeRibbon")

Else

IsRibbonShowedBefore\_bool = False

End If

End Sub

Private Sub Workbook\_BeforeClose(Cancel As Boolean)

If IsRibbonShowedBefore\_bool = True Then

Application.CommandBars.ExecuteMso ("MinimizeRibbon")

End If

End Sub

#### A fókusz megadása

A megnyíló munkafüzetben kiválasztásra kerül a Sudoku Solvert tartalmazó munkalap, a nagyítás 85%, az aktív cella az „A1”:

Private Sub Workbook\_Open()

Sheets("9x9 VBA").Activate

ActiveWindow.Zoom = 85

With ThisWorkbook.Sheets("9x9 VBA")

.Activate

.Range("A1").Select

End With

End Sub

#### Az ellenőrző- és segédcellák

A „9x9 VBA” munkalap A35:X68 tartományában találhatóak az ellenőrző és segédcellák, amelyek segítségével követhetjük a program helyes működését.

#### Az „A1”-cella tartalma

Az „A1”-cellában egy kezdetleges verziókövetés kapott helyet. a „logX.Y” jelölés az algoritmus és az adat ki- és bemenet működési logikáját jelzi, ahol az „X” az algoritmus változásával növekszik, az „Y” a ki- és beviteli mezők pozíciójának, működésének változását jelzi. A „desX.Y” értékének módosításával a felhasználói felület változását kell lekövetni.

## Megoldás MS Excel Solver beépülő modul segítségével (Mayerhöffer Orsolya – IGJ96Q)

# Bevezetés

A dokumentum célja megismertetni a MS Excel Solver beépülő modul segítségével elkészített 4x4 es sudoku játék megoldási módszerét valamint ugyanezen módszerrel bemutatni 9x9-es játékon való próbálkozást is.

A 4x4 sudoku játék elsősorban gyerekeknek ajánlják és azoknak akik még csak ismerkednek a játékkal. Persze akkor is kiváló választás lehet ha épp csak egész kis időnk van, de szeretnénk játszani egyet. A játék szabályai hasonlóak az eredetihez. A lényeg, hogy minden sorban, oszlopban és 2x2-es mezőben 1-4-ig csak egyszer szerepelhetnek a számok. Egyszer viszont szerepelniük kell.

# Fejlesztési környezet

## Hardver

Dell Latitude 5530

RAM: 16 GB

Storage: 250 GB SSD

## Szoftver

MS Windows 11 Enterprise

Microsoft® Excel® for Microsoft 365 MSO

(Version 2402 Build 16.0.17328.20124) 64-bit

# Első 4x4 megoldási módszer

## Google keresés

Keresnem kellett egy online felületet, ahol néhány játék után megértettem a játék célját és szabályait.

## ChatGPT

Nem volt ötletem, hogy hogyan kezdjem el, így az alábbi kérdéssel fordultam a ChatGPT-hez:

A screenshot of a computer

Description automatically generated

* + - 1. ábra

A 3. pont alapján indítottam egy Google keresést, hogy megtudjam, hogyan telepíthető a bővítmény, pár kattintással használatba is vettem.

Létrehoztam a 4x4-es táblázatot, viszont a fenti válasz ennél tovább nem volt segítségemre, ezért más kérdésekkel próbálkoztam, hogy megkapjam a pontos válaszokat, de nem sikerült.

## Értekezletről készült felvétel

Többször megnéztem az óráról készült felvételből azt a részt, ahol a Solver-ről volt szó, ezután fogalmaztam meg a játék szabályait – ekkor még csak írásban -, mint megadandó feltételek:

* minden sor összege 10
* minden oszlop összege 10
* minden rács összege 10
* minden szám egyszer szerepelhet soronként, oszloponként és rácsonként
* 1<= n <=4
* minden szám egész szám legyen

SUM képlettel elvégeztem a számolásokat és beírtam a fenti feltételeket a Solver-be (lásd: sudoku\_v1.0.xlsx), lefutattam és az alábbi üzenetet adta vissza:

A screenshot of a computer

Description automatically generated

* + - 1. ábra

Minden feltétel teljesült, kivéve, hogy egy szám csak egyszer szerepelhet minden sorban, oszlopban és rácsban, az előre beírt számokat nem vette figyelembe. Ennél a pontnál ismét elakadtam.

## Brainstorming osztálytársakkal

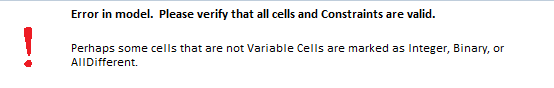
Hosszú keresés, youtube video és Solver működéséről szóló cikkek olvasása után felfedeztem, hogy a Solver-ben elérhető egy kimondottan olyan megkötés, mely biztosítja, hogy minden nézetben egyzsre forduljon elő egy szám a megadott tartományból:

A screenshot of a computer

Description automatically generated

3.ábra

Tehát beírtam egyenként minden sort, oszlopot és rácsot, mint „call reference” és „AllDifferent”-re állítottam be, valamint azt, hogy az összegük 10 kell, hogy legyen, a cél pedig a sorok, oszlopok, rácsok összege, azaz 120, így az alábbi hibaüzenetet kaptam.



4. ábra

Ekkor írtam osztálytársamnak, Ott Alexnek, aki mint kiderült, hasnoló megoldáson dolgozott 9x9-es táblán és így derült ki számomra, hogy a fenti megkötés csak akkor fut le, ha a a tábla üres, nincs előre beírt szám, így azokat kitörölve a Solver ki tudta tölteni megfelelően a táblát. (sudoku\_v1.1.xlsx)

Ahhoz, hogy csökkentsem a futási időt úgy, hogy a hiba lehetősége ne legyen több, a nézetek összegit külön mezőben számoltam, így csak ezen tartományok beírására volt szükséges a Solver-ben.

A screenshot of a graph

Description automatically generated

1. ábra (sudoku\_v1.2.xlsx)

# Második megoldási módszer

Szerettem volna egy olyan megoldást is, ami képes kiegészíteni a már meglévő számokat a szabályok szerint.

## MIAU

Ez volt a meghatározó segítség a végső megoldásomhoz, mivel már korábban elkezdtem azon gondolkozni, ha segédtáblázatokon megszámolom „COUNTIF” képlettel, hogy nézetenként hányszor szerepelnek 1 és 4 között a számok, akkor azt valahogy át lehet írni a megfelelő megkötéssé és így biztosan különbözni fognak minden nézetben a számok.

A screenshot of a computer

Description automatically generated

6. ábra

Nem akartam IF-eket használni, túl bonyolultnak találtam.

Tovább kerestem, így találtam meg egy, a [MIAU felületen](https://miau.my-x.hu/miau2009/index.php3) publikált megoldást, ami a szórás segítségével határozza meg a megkötéseket:



Így a változó mezőket egy külön táblázatba vezettem:

A green square with black text

Description automatically generated

7. ábra

A három nézetre készült segédtáblázatokra különböző módokon szórást számoltam (6. ábra), miszerint az átlagtól való eltérésük abban az esetben egyezik az átlaggal, ha mindenhol egyszer szerepel egy szám. Minden más esetben nem nulla. Tehát ezzel a logikával van megkövetelve, hogy a segédtáblázat (6. ábra) értékei mindenhol 1-es legyen 0 cél értékre minimalizálva, mivel az soha nem lesz teljesen 0, viszont egész számként várt eredmény esetében ez lényegtelen. Ezt pedig további néhány megkötés biztosítja az eddig használtak közül, mint pl. minden nézet összege 10 és kisebb vagy egyenlő, mint 4 és egyenlő vagy nagyobb, mint 1.

Bár meg volt a megoldás, mégis sokáig eltartott, mire azt helyesen be tudtam írni a Solver-be, végül a sikert a különböző nézetek szórás eredményeinek összeadása hozta meg, amit egy külön mezőben számoltam két féle módon is:

A screenshot of a calculator

Description automatically generated

8. ábra (sudoku\_v2.0.xlsx)

A screenshot of a computer

Description automatically generated

9. ábra (sudoku\_v2.0.xlsx)

Nehéséget okozott rájönnöm arra, hogy valójában a cél érték soha nem lesz pontosan nulla, így azt csak minimalizálni lehet a helyes megoldás eléréséhez.

# Harmadik megoldási módszer

Itt a célom az, hogy a Solver ne csak előre beírt számok, ahogy a 8. és 9. ábrán a piros színű számok mutatják, egy feladat esetén találjon megoldást, hanem úgy is, ha bármilyen megoldható feladat alapján adom meg számokat. Így bővítettem a változó cellák tartományát pont annyi cellára, mint amennyi a játék is, azaz 4x4-re és a játék celláit hivatkoztam a változó cellákra, majd a játék táblába beírtam számokat, úgy, hoyg az megoldható legyen, valamint a Solver megkötései közt módsítottam azt a tartományt a változó cellák szerint, ahova egész számot várok és így is sikerült megoldásra jutnom:

A green and white sheet of paper with green text

Description automatically generated

10. ábra (sudoku\_v2.1)

A hátrány az, hogy minden új játék előtt újra kell hivatkozni a játék mezőit a változó mezőkre, majd kézzel beírni a feladvány számait a játék táblázatába.

# 9x9 megoldási módszer

A negyedik pontban leírt módszer alapján 9x9-es táblára is elkészítettem a rendszert (11. ábra), viszont sikertelen megoldás született.

A screenshot of a computer

Description automatically generated

11. ábra

Ezután megpróbáltam további megkötésként megadni a változó mezőkbe várt számok összegét hozzáadva a nem haszált változó mezőkbe írt egyesek összegével, viszont így sem tudott megfelelő megoldás születni.

A screenshot of a computer

Description automatically generated

12. ábra

## ChatGPT

Kíváncsi voltam, hogy 9x9-es táblára van-e megoldása a ChatGPT-nek Excel Solver-rel:

A screenshot of a computer

Description automatically generated

# Konklúzió

A bemutatott módszerek által jól látható, hogy az Excel Solver beépülő modul nem a legjobb, csak kis térben, 4x4-es táblán működik biztosan, nehezebb, 9x9-es tábla feladatot már nem tud megoldani megfelelően.

## Sudoku Excel Solver megoldás 4x4 és 6x6 táblázat alapján (Csötönyi Viktória - J908Q3)

### Bevezetés

Sudoku népszerű logikai játék, amely a játékost gondolkodásra sarkall. A feladványokat megtalálhatjuk az újságok lapjain, a rejtvénykönyvekben vagy manapság különböző mobil applikációkban szintén.

Elsődlegesen segítséget kértem a ChatGpt-től, hogy magyarázza el a szabályokat (ChatGPT3.5).

Az alábbira kértem: ,,Kérlek magyarázd el a sudoku szabályait!”

Ezt a leírást kaptam:

,,A Sudoku egy logikai játék, amely egy 9x9-es rácsra épül, amelyet további 9 kisebb 3x3-as rácsra osztanak fel. A játék célja, hogy minden sorban, oszlopban és 3x3-as blokkban szerepeljenek a számok 1-től 9-ig úgy, hogy egyetlen szám sem ismétlődik meg ugyanabban a sorban, oszlopban vagy blokkban.

A Sudoku szabályai a következők:

1. A rácsot 9x9-es négyzetre osztják, amelyeket tovább osztanak 3x3-as blokkokra.
2. A rácsnak 9 sorban és 9 oszlopban kell lennie.
3. A rács néhány cellája már kitöltött számmal kezdődhet. Ezeket a számokat "adatoknak" nevezik.
4. A játékosok feladata a rács többi celláját kitölteni a szabályoknak megfelelően.
5. A játékosnak úgy kell kitöltenie a rácsot, hogy minden sorban, oszlopban és 3x3-as blokkban mindegyik szám (1-től 9-ig) csak egyszer szerepelhet.
6. A játék megoldása mindig egyértelmű és egyedülálló. Ez azt jelenti, hogy csak egy helyes megoldás létezik a játékban.

A Sudoku egy nagyon népszerű logikai játék, amelynek sok változata létezik, de az alapvető szabályok általában változatlanok maradnak. A játék megoldása logikai gondolkodást és problémamegoldó képességet igényel.”

A szabályok ismerete után először 4x4 táblázatot hoztam létre, majd a sikeres megoldás után áttértem a 6x6 táblázatra.

### Célkitűzés

Sudoku megoldása Excel Solver segítségével, először 4x4, majd 6x6 sudoku táblázattal.

### Excel Solver

### 3.1 Excel solver 4x4 táblázat

Sablon 4x4-es sudokut elkészítettem az Excelben (Web1.). Ezt követően a Solver működésének néztem utána, hogy megértsem a program alapját. A Miau oldalon megtaláltam a ,,SUDOKU-algoritmusok összevetése, avagy SOLVER vs. egyedi kód” című publikációt, amelynek segítségével megértettem a Solver alapjait (Pitlik L., Pitlik M., web2).

A kijelölés könnyebbsége miatt az üres cellákat összesítettem egy másik táblázatban, ezt a táblázatot neveztem el Solver változó cellának. 4x4 táblázatnál két rész táblázatot hoztam létre, amelynek a feltétele, hogy minden egyes sorban, illetve oszlopba az adott szám egyszer szerepeljen. Ezt a már fentebb említett „Darabha” függvénnyel határoztam meg és a táblázatok végén a rész táblázatok szórását határoztam meg „szórás” függvénnyel. Azonban a célérték meghatározásnál kértem segítséget Kovács János csoporttársunktól, mivel az általam meghatározott célérték (az oszlopok és a sorok maximális összege összesen (40)) nem működött. Azt a megoldást ajánlotta, hogy a két táblázat szórás összegét adjam meg a célértéknek, amelynek a célja az, hogy a sorok és oszlopoknak a szórásainak az összege 0 legyen. Ezáltal értettem meg teljesen a célértéket és ezekkel a feltételekkel a Solver helyesen oldotta meg a feladványt.

Solver feltételek:

* A solver változó cella =>1,
* A solver változó cella =<4,
* A solver változó cella = egész,
* Az oszlopok összege =10
* A sorok összege = 10
* A blokkok összege = 10

### 3.2 Sudoku Solver 6x6 táblázat

Mivel kisebb részben a Solver helyesen megtudta oldani a feladatot, ezért létrehoztam 6x6 táblázatot, amelyet 3x2 blokkokra osztottam. (Azonban ezt a táblázatot egy másik Excelben csináltam, ugyanis, ha 4x4 táblázattal szerepelt egyben külön fülön a Solver futásánál lefagyott a program). Internetről, pedig egy sablon sudokut választottam alap megoldásra (Web 3.). Ezt a táblázatot először kézzel kitöltöttem a Solver megoldás helyességének ellenőrzése miatt.

Továbbá egy külön Solver változó táblázatot hoztam létre. Illetve az oszlopok és sorok elemét itt is „Darabha” függvénnyel számoltam és ennél a táblázatnál már a 3x2 blokkhoz is létrehoztam egy ilyen táblázatot, ahol a bizonyos számok csak egyszer szerepelhetnek, az adott blokkban.

Szintén a szórások összegét adtam meg a célértéknek.

Az alábbi feltételeket adtam meg a solverben:

* A solver változó cella =>1,
* A solver változó cella =<6,
* A solver változó cella = egész,
* Az oszlopok összege =21
* A sorok összege = 21
* A blokkok összege = 21

### 3.3. Eredmény 6x6 sudokunál

Solver a megadott feltételeket teljesítette, azonban vannak olyan megoldások, amelyeket a sudoku szabályainak nem felelnek meg. Egy blokkban több azonos szám is szerepel, mégis az általam megadott feltételek teljesülnek. Többszöri próbálkozásra viszont a sudoku szabályainak megfelelő helyes megfejtést is megadta. Ebből kifolyólag a megadott feltételekkel több megoldás is van.

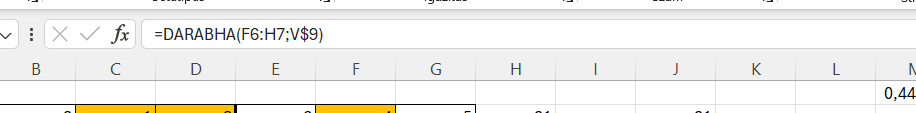
Emiatt megadtam plusz feltételnek, hogy a sor, oszlop és blokkok szórásának 0-ával kell egyenlőnek lennie, viszont a Solver hibát jelzett ki (1. ábra).

A képen szöveg, képernyőkép, szoftver, szám látható

Automatikusan generált leírás

1.ábra

Ekkor néztem át újból a hibajelzéseket és vettem észre azt, hogy képletben a $ jelek hiányoznak, így automatikusan elcsúszott a meghatározottnak vélt blokkom (2. ábra).



2.ábra

A hibát javítottam és újból a plusz feltételekkel lefuttattam a solvert, amely már csak egy helyes megoldást adott (3.ábra).

A képen szöveg, képernyőkép, sor, szám látható

Automatikusan generált leírás

3. ábra

Ennél a pontnál gondolkodtam el, hogy hogyan lehetne a feltételeket lecsökkenteni és egyszerűbben megadni, hogy sudoku szabályainak megfeleljenek. Ekkor kitöröltem ,,a sorok, oszlopok és blokkok legyenek egyenlőek 0-ával” utasítást és helyette megadtam, hogy az ,,oszolopok, sorok és blokkok szórásaik legyenek egyenlőek 0-ával” (4.ábra.). Solver megoldásig 2. sorig jutott el, utána lefagyott vagy nem végezte el helyesen a műveleteket.

A képen szöveg, képernyőkép, képernyő, szoftver látható

Automatikusan generált leírás

4.ábra

### Összegzés

Az Excel Solver megadott feltételekkel és a helyes Excel képlet használatával megoldotta megfelelően a 4x4 sudoku táblázatot. Azonban 6x6 táblázatnál lehetett megfigyelni, hogy egy adott feltételeknek több helyes megoldása is van, amely működött a 4x4-nél nem biztos, hogy megfelelően működik a 6x6-nál. OAM alapján 10-ből egyszer sikerült a helyes megoldást megjeleníteni, azonban kétszer hozott ki ugyanolyan megoldást,amely sudoku szabályainak nem megfelelő.

A leszűkített 6x6 verzió(2) viszont nem működött jól, mivel a szoftver többször lefagyott és a megoldásnál csak az első két sort oldotta meg helyesen.

# Az alkalmazott módszerek összehasonlítása OAM elemzés segítségével

Több különböző Sudoku solver (pl. Go program és Excel VBA) futtatási eredményeit hasonítjuk össze. A sikeres OAM elemzéshez először definiálni kell az objektumokat és attribútumokat, majd ezek alapján elemzést végezni.

## Objektumok:

Go program

Excel VBA

Excel Solver

## Attribútumok:

Algoritmus típusa (backtrack, Solver)

Megoldási terület mérete („4×4”; „6×6”; „9×9")

Feladat mérete (különböző nehézségű Sudoku feladványok)

Futási idő sec-ben

Memória használat byte-ban (az erőforrás-igény)

Sikerült-e megoldani (megoldás sikeressége)

**Az összehasonlítás részletei és eredménye a mellékelt Excel-táblázatban érhető el („Sudoku solver OAM excel.xlsx”).**

# Tantárgyi kapcsolatok

## LIN029 - Operációs rendszerek

Az operációs rendszerek kezelik az erőforrásokat (memória, processzoridő), amelyek létfontosságúak a Sudoku solver futtatásához. A folyamatok és szálak kezelése, amit az operációs rendszerek tantárgy tárgyal, közvetlenül befolyásolja az Excel programok, így a Sudoku solver teljesítményét és stabilitását. Emellett az operációs rendszerek hibakezelési mechanizmusai alapvetőek a Sudoku megoldóprogramok zökkenőmentes működéséhez. Röviden, az "Operációs rendszerek" elvei nélkülözhetetlenek a hatékony és megbízható Sudoku megoldó programok fejlesztéséhez Excel környezetben.

## LIN026 - Matematikai alapok

Az Excelben készült Sudoku megoldóprogram és a "Matematikai alapok" tantárgy közötti kapcsolat abban rejlik, hogy a megoldó algoritmusok tervezésekor és implementálásakor alapvető matematikai logikát és algoritmusokat, mint például kombinatorikát, logikai műveleteket és esetlegesen gráfelméletet használunk. A tantárgy által nyújtott matematikai gondolkodásmód és problémamegoldó készségek közvetlenül segítik a fejlesztőt a Sudoku rejtvények hatékony megoldási stratégiáinak kialakításában.

## LSK001 - Kultúra, sport, munkahelyi jóllét 1.

A Sudoku megfejtése javítja a logikai gondolkodást és a koncentrációt, amelyek hozzájárulnak az általános mentális egészséghez és jólléthez, ezzel összhangban állva a tantárgy által képviselt értékekkel. Ez a kapcsolat kiemeli, hogy a szellemi kihívást jelentő tevékenységek, mint a Sudoku megoldása, pozitív hatással lehetnek a munkahelyi és személyes jóllétre.

## LKN490 - Európai civilizáció és identitás

A Sudoku, mint logikai játék, és annak megoldására szolgáló programok fejlesztése tükrözheti az európai kultúrában gyökerező értékeket, mint a tudományos gondolkodás, az innováció és az intellektuális kihívások iránti elkötelezettség. Ez a kapcsolat bemutatja, hogyan hozhatnak össze különböző területeket a kulturális hagyományok és a modern technológiai megoldások.

## LIN028 - Hálózatok és számítógép architektúrák

Az Excelben készült Sudoku megoldóprogram és a tantárgy közötti kapcsolat abban rejlik, hogy a program fejlesztése és futtatása során kihasználja a számítógép architektúráját, mint a processzor teljesítményét és a memória kezelését.

## LIN027 - Adatszerkezetek és algoritmusok

A program alapját képező logikai algoritmusok, mint a backtracking és a megoldási stratégiák, közvetlenül kapcsolódnak az adatszerkezetek és algoritmusok tanulmányozásához, bemutatva, hogy ezek a fogalmak hogyan alkalmazhatók valós problémák megoldására.

## LNE178 - A jog szerepe a modern társadalmakban

A program fejlesztése során felmerülő jogi kérdések, mint a szoftver licenszelése, a felhasználási feltételek meghatározása és a szellemi alkotások jogi védelme, mind fontos elemek, amelyek bemutatják a jog szerepét a technológia és innováció világában.

# Összegzés

A választás a Sudoku megoldó program kialakításakor a különböző programnyelvek között függ a felhasználó vagy fejlesztő preferenciáitól, a rendelkezésre álló időtől, és attól, hogy mennyire bonyolult a megoldandó Sudoku rejtvény. Míg az Excel Solver egy gyors és egyszerű megoldást kínál kevésbé összetett feladványok esetén minimalizált fejlesztési erőfeszítéssel, addig a backtrack algoritmus lehetővé teszi mélyrehatóbb testreszabást és általában jobb teljesítményt nyújt bonyolultabb rejtvényeknél, cserébe a nagyobb fejlesztési idő és a szükséges programozási ismeretek miatt.

* Go program: Általánosságban kevesebb erőforrást használ, és gyorsabban fut, de vannak esetek, amikor nem sikerül megoldania a feladványt.
* Excel VBA: Nagyobb erőforrásigényű, és lassabb futási idejű, de magas sikerességi rátát mutat.

# Summary

The choice between different programming languages when designing a Sudoku solver program depends on the user or developer's preferences, the available time, and the complexity of the Sudoku puzzle to be solved. While Excel Solver offers a quick and simple solution for less complex puzzles with minimized development effort, the backtrack algorithm allows for more in-depth customization and generally provides better performance for more complex puzzles, in exchange for greater development time and the need for programming knowledge.

* Go program: Generally uses fewer resources and runs faster, but there are cases where it fails to solve the puzzle.
* Excel VBA: More resource-intensive and has a slower runtime, but shows a high success rate.

# Rövidítésjegyzék, hivatkozásjegyzék, irodalomjegyzék

## Csötönyi Viktória\_J908Q3:

Források:

Web1.:<https://letsdopuzzles.com/dl/4x4-sudoku-puzzles-set-3>

Web 2..:https://miau.my-x.hu/miau2009/index.php3?x=e0&string=sudoku

Pitlik László és Pitlik Marcell, „SUDOKU-algoritmusok összevetése, avagy SOLVER vs. egyedi kód” (2019)

Web 2.:<https://letsdopuzzles.com/dl/6x6-sudoku-puzzles-1>

# Mellékletek