Eötvös Loránd Tudományegyetem
Informatikai Kar
Komputeralgebra Tanszék

Sudoku megoldó algoritmusok

Témavezető:
Dr. Iványi Antal
Egyetemi tanár
Komputeralgebra Tanszék

Készítette:
Lakatos Zoltán
Programtervező Informatikus szak,
nappali tagozat

Budapest, 2010

Tartalom

I. Bevezetés

1. A Sudoku szabályai

Adott egy 9×9-es négyzetrács − amely 3×3-as kis négyzetekre van bontva − mezőiben az X = {0, 1, 2, …, 9} ábécé egy-egy betűje szerepel. A feladat az, hogy a nullákat úgy helyettesítsük az ábécé többi betűjével, hogy végül minden házban (azaz minden sorban, minden oszlopban, és minden 3×3-as kis négyzetben) az ábécé összes betűje pontosan egyszer szerepeljen. A rejtvény általánosítása: egy N×N-es kis négyzetekből álló, $N^2 \times N^2$-es négyzetrács mezői az X = {0, 1, …, N^2} ábécé elemeivel vannak kitöltve. A feladat a nullák helyettesítése az X ábécé többi betűjével úgy, hogy végül minden házban az ábécé minden betűje pontosan egyszer forduljon elő.

2. A program célja

A sudoku algoritmusokkal kapcsolatos, az utóbbi tíz évben gyorsan növekvő szakirodalomból Iványi Antal (*A Rózsa program algoritmusai* **[3-Iványi2009]**) és J. F. Crook (A Pencil-and-Paper Algorithm for Solving Sudoku Puzzles **[7-Crook]**) műveit fogom felhasználni.

Célom olyan algoritmusok implementálása, amelyekkel a sajtóban megjelenő rejtvények nagy többsége megoldható. Munkám során fel fogom használni a Balázs Viktor nagyprogramja **[9-Balázs2007]** által biztosított keretrendszert (amely C# nyelven íródott és Microsoft .NET Framework 2.0 környezetben működik), és ezt fogom kibővíteni újabb algoritmusokkal.

Szakdolgozatom célja:

- A fentebb említett dokumentumokban szereplő algoritmusok megvalósítása C# nyelven: Rejtett és tiszta *n*-es algoritmusok, tégla, Nishio, Crook algoritmusa.
- Keretrendszer továbbfejlesztése, ami magába foglalja például a felhasználónak vizuális segítség nyújtását, algoritmusok kikapcsolására lehetőség biztosítását és megoldónapló továbbfejlesztését.

II. Felhasználói dokumentáció

1. Hardver-szoftver követelmények

A program futtatásához Microsoft .NET keretrendszer szükséges.

2. Telepítés

A program a Sudoku.exe fájlt elindítva közvetlenül futtatható, telepítést nem igényel.

3. Program részei

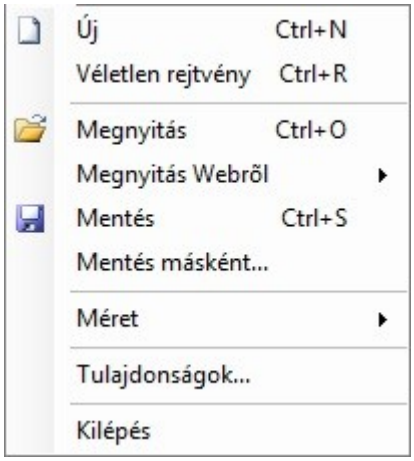
A program működéséhez a következő fájlok szükségesek:

- Sudoku.exe
- SudokuControls.dll
- SudokuTypes.dll
- sudokuhelp.chm

4. Program használata

4.1 Menük

4.1.1 Fájlmenü:



1. ábra: Fájlmenü

Új: Létrehoz egy üres sudokut (billentyűkombináció: CTRL + N).

Véletlen rejtvény: Véletlen sudoku rejtvényt generál, alkalmazkodva az aktuális mérethez. (billentyűkombináció: CTRL + R).

Megnyitás: Megnyitás ablak előhozása (billentyűkombináció: CTRL + O).

Megnyitás Webről: Az interneten számos oldal naponta új sudoku feladványokat tesz közzé, ebben a menüpontban a kiválasztott oldal napi feladványát lehet betölteni a programba.

2. ábra: Megnyitás Webről

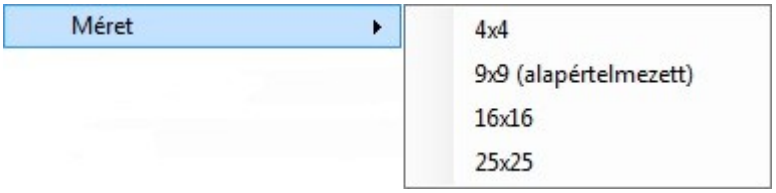
Az aktuális változat a http://www.dailysudoku.co.uk oldal napi feladványát tudja betölteni.

Mentés: Az aktuális rejtvényt menti az adott kitöltöttségi állapotban. Ha létező fájlt szerkesztettünk, akkor a fájlnevet és a formátumot nem változtatja, egyébként a fájlnév és a formátum szabadon állítható. Részletek a „Támogatott fájlformátumok” pontban (billentyűkombináció: CTRL + S).

Mentés másként: Az aktuális rejtvényt menti az adott kitöltöttségi állapotban, a fájlnév és a formátum szabadon állítható.

Megjegyzés: Mentés során az aktuális megoldási napló nem kerül elmentésre, azaz a visszaolvasott rejtvényben szereplő értékeket adatként fogja értelmezni!

Méret: A kinyíló ablakmenüben kiválasztható a tábla mérete: 4×4, 9×9, 16×16, 25×25.

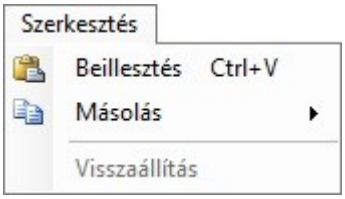


3. ábra: Táblaméret állítása

Tulajdonságok: Az SDK formátumú fájlokban különféle leíró adatok szerepelhetnek, ezek megjelenítését, szerkesztését lehet itt elvégezni.

Kilépés: A program bezárása.

4.1.2 Szerkesztés menü:



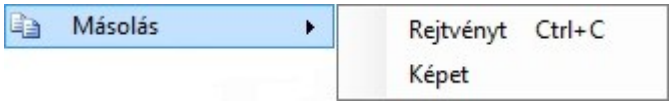
4. ábra: Szerkesztés menü

Beillesztés: A vágólapon található rejtvényt olvassa be (billentyűkombináció: CTRL + V).

Az ismert formátumok 9×9-es sorfolytonos leírásúak. Példa:

0098002101000000400600000055000020060000030000040800970005000006007200000920015000..98..21.1.....4..6.....55....2..6.....3.....4.8..97...5.....6..72.....92..15..__98__21_+1_____4_+_6_____5+5____2__6+_____3___+__4_8__97+___5_____+6__72____+_92__15__

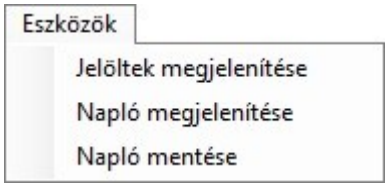
Másolás: A vágólapra illeszti a rejtvényt, illetve a képet (fekete-fehérben). Rejtvény vágólapra illesztésének billentyűkombinációja: CTRL + C. A sorfolytonos leírása (példa):
0098002101000000400600000055000020060000030000040800970005000006007200000920015 00



5. ábra: Vágólapra másolás

Visszaállítás: Visszaállítja a rejtvényt a megoldás előtti állapotába.

4.1.3 Eszközök menü:



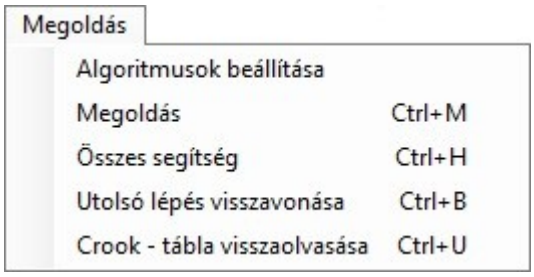
6. ábra: Eszközök menü

Jelöltek megjelenítése: Ki- és bekapcsolható a mezőkben megengedett jelölt megjelenítése.

Napló megjelenítése: Ki- és bekapcsolható az algoritmusok működését követő napló megjelenítése.

Napló mentése: Napló mentése TXT fájlba.

4.1.4 Megoldás menü:



7. ábra: Megoldás menü

Algoritmusok beállítása: Lehetőséget biztosít a program által használt algoritmusok ki-, illetve bekapcsolására.

Megoldás: A Rózsa algoritmus alapján megoldja a rejtvényt (billentyűkombináció: CTRL + M).

Összes segítség: Megmutatja az aktuális állásnál a program által talált összes mintát (billentyűkombináció: CTRL + H).

Utolsó lépés visszavonása: A rejtvényben történt utolsó változtatást visszavonja, és törli az ahhoz tartozó naplóbejegyzést (billentyűkombináció: CTRL + B).

Crook – tábla visszaolvasása: Amennyiben visszalépéses technika alkalmazása során hibát találunk, visszaolvassa az utolsó olyan táblát, ahonnan a megoldást folytatva még jó megoldáshoz juthatunk (billentyűkombináció: CTRL + U).

4.1.5 Súgó menü:

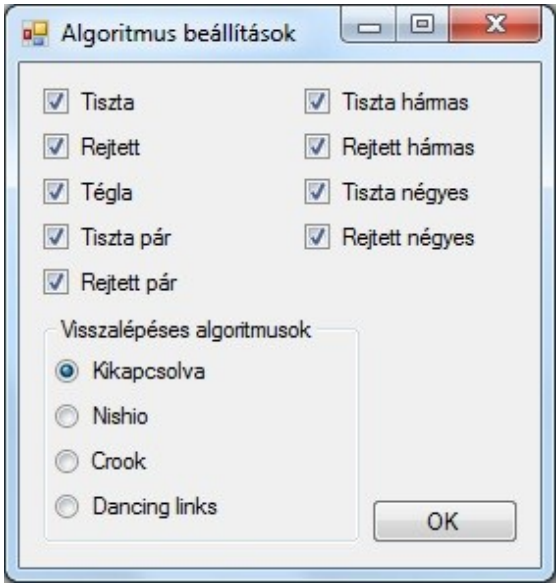
8. ábra: Sugó menü

Tartalom: Elindítja a súgót.

Névjegy: A program névjegye.

4.2 Ablakok

4.2.1 Algoritmusok beállítása:

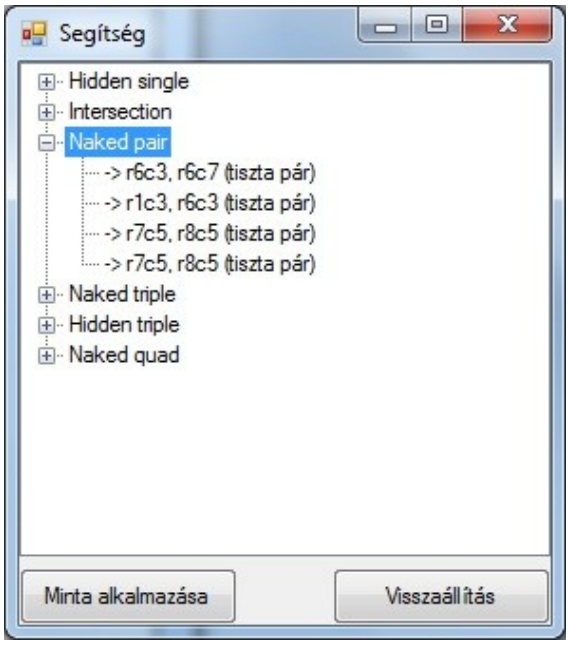


9. ábra: Algoritmusok beállítása

Itt tudjuk megadni, hogy a program melyik algoritmusokat használhassa a megoldáshoz. A visszalépéses algoritmusok közül mindig legfeljebb egy lehet aktív, míg az „egyszerű" algoritmusok közül tetszőlegesen válogathatunk.

Megjegyzés: A Dancing links algoritmushoz a program nem készít naplót!

4.2.2 Segítségek ablak:



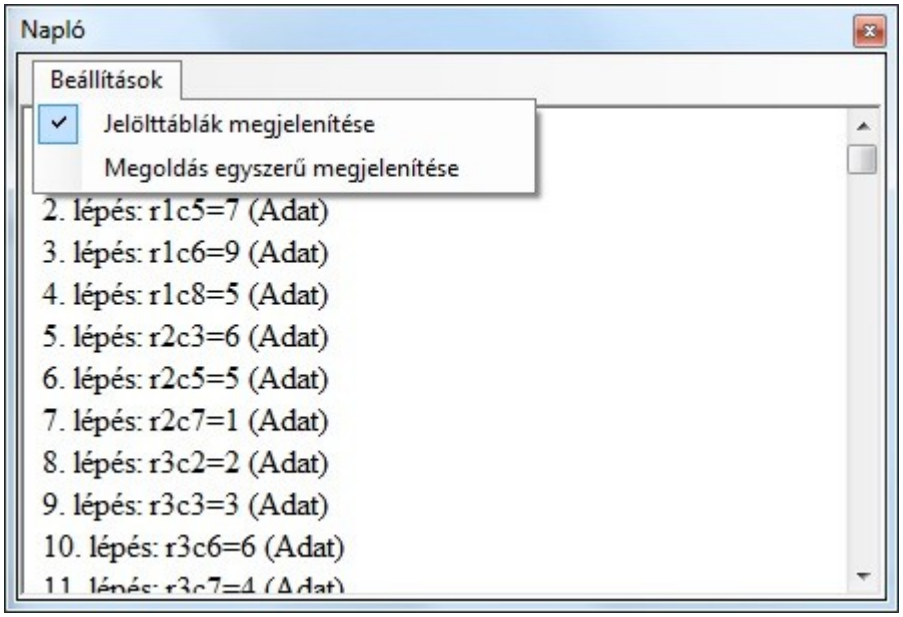
10. ábra: Segítség ablak

Megjeleníti a bekapcsolt algoritmusok aktuális állásnál megtalált összes mintáját algoritmus szerint csoportosítva, valamint a kiválasztott mintát kirajzolja a rejtvény tábláján.
Lehetőséget biztosít az alábbi két művelet elvégzésére:

Minta alkalmazása: A fent kiválasztott minta változtatásait végrehajtja a táblán.

Visszaállítás: Az utolsóként alkalmazott minta változtatásait visszavonja.

4.4.3 Napló ablak:



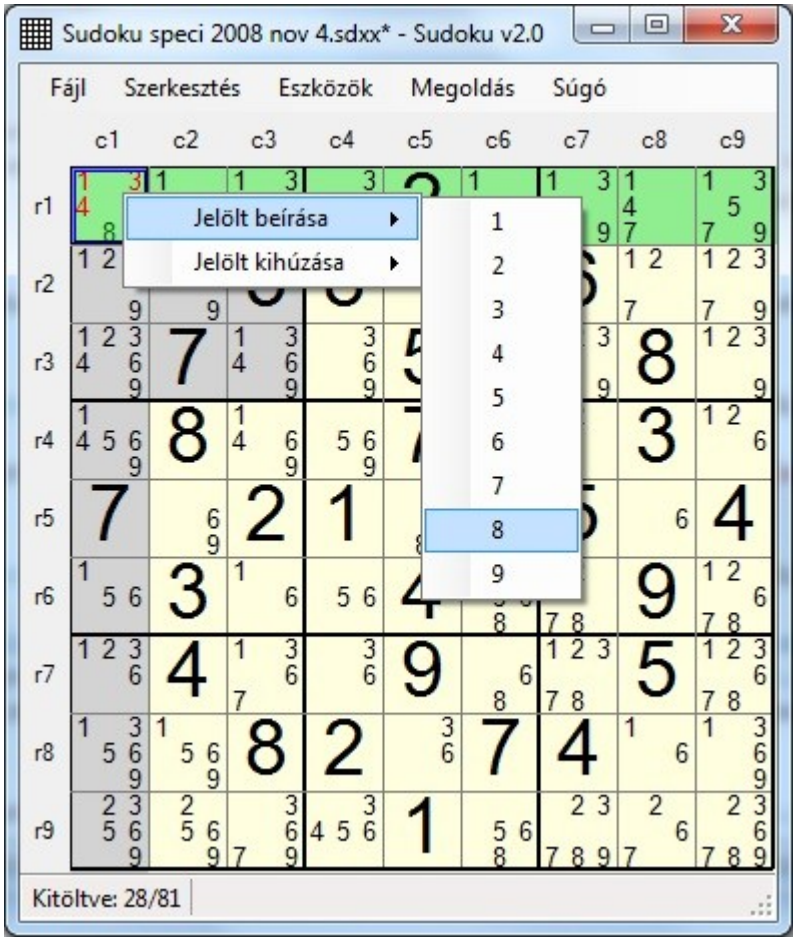
11. ábra: Napló ablak

A megoldási napló megjelenítésére szolgáló ablak. A beállítás menüben néhány, a megjelenítéssel kapcsolatos lehetőséget tudunk megadni:

Jelölttáblák megjelenítése: Bekapcsolásakor mindig, amikor egy algoritmus munkához lát, megjeleníti az aktuális jelölttáblát.

Megoldás egyszerű megjelenítése: A rejtvény megoldását egy tömörebb, egyszerűbb formában jeleníti meg. Nagy rejtvények esetében átláthatóbb, mert egyébként a rejtvényben egy sorban lévő elemek nem férnének el a napló egy sorában.

4.2.3 Főablak:



12. ábra: Főablak

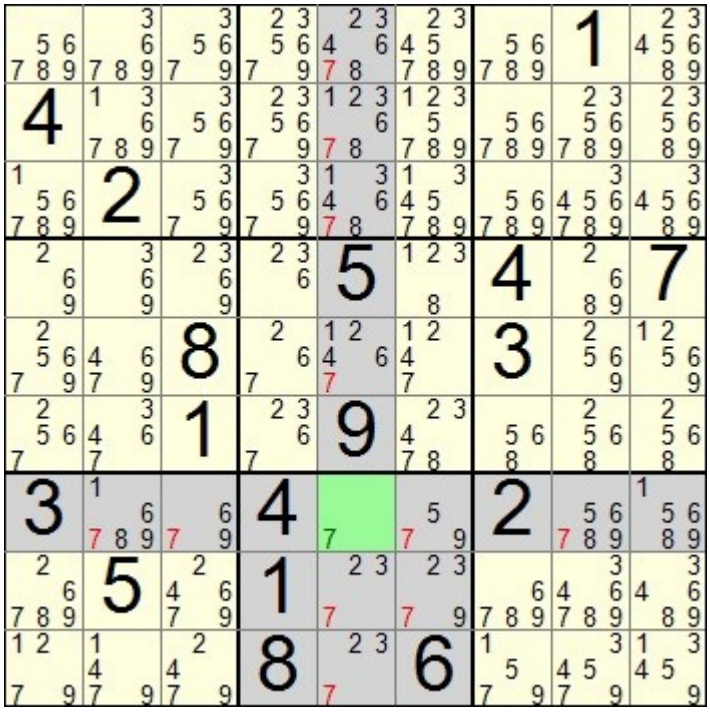
Megjeleníti a sudoku tábláját, és lehetőséget ad annak módosítására. Ezt a felhasználó két módon teheti meg:

Egérrel: Jobb gombbal egy mezőre kattintva lenyíló menü jelenik meg, ahonnan kiválaszthatjuk, hogy a kijelölt cellának melyik jelöltjét szeretnénk kihúzni vagy beírni.

Billentyűzettel: A kurzort nyilakkal tudjuk mozgatni, majd a megfelelő billentyű lenyomásával írhatunk be, illetve húzhatunk ki jelöltet. Ezen műveletek között a tabulátor billentyű segítségével lehet választani. Ha az egész cellát kijelölő zöld keret jelenik meg a képernyőn, akkor adatokat tudunk beírni, ha pedig egy, csak jelölteket kijelölő piros keret, akkor jelöltet húzhatunk ki.

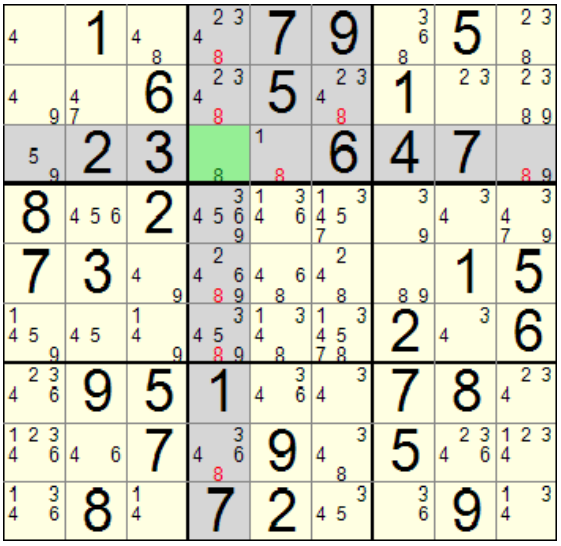
5. Felhasználó grafikus támogatása

A grafikus segítségnyújtás feladata, hogy megkönnyítse az egyes megoldási technikák megértését. Igénybevételéhez a segítségek ablakban ki kell jelölnünk egy mintát, ami azonnal kirajzolásra kerül a sudoku tábláján. Zölddel jelöljük azokat a cellákat, illetve jelölteket, amelyek meghatározzák a mintát, vagyis a kihúzásokat lehetővé teszik. Szürke háttérrel rendelkező cellák általában a jelölt kihúzásokban érintettek, míg a konkrétan kihúzható jelölteket piros színnel jelöljük.



6. Algoritmusok ismertetése

Tiszta cella (naked single):

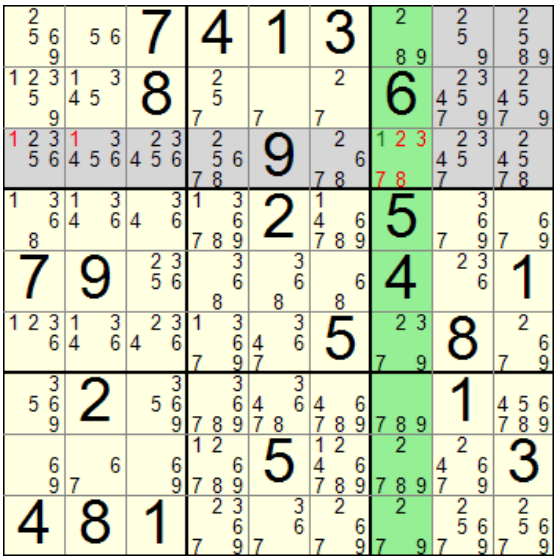


1. algoritmus: Tiszta cella

Olyan jelöltet keresünk, amely egyedüliként szerepel cellájában, vagyis az összes többi jelölt már megtalálható szomszédjaként.

A bal oldalon látható példában a zölddel színezett cellában csak a 8-as van megengedve, ezért oda azt beírhatjuk. Az ezt a cellát tartalmazó házakból pedig a 8-as jelölteket töröljük.

Rejtett cella (hidden single):



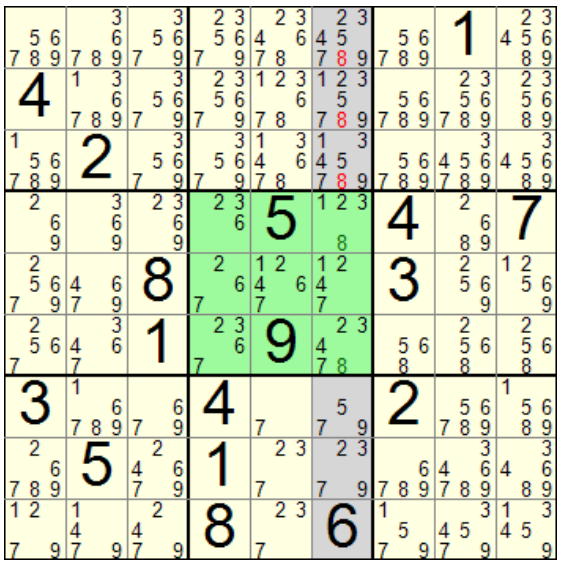
2. algoritmus: Rejtett cella az oszlopban

Def.: Háznak nevezzük a rejtvénynek egy sorát, oszlopát vagy blokkját.

Olyan házat keresünk, amiben van olyan jelölt, amely csak egy cellában fordulhat elő. Ekkor ki tudjuk használni, hogy a sudoku minden házában minden értéknek szerepelnie kell, itt pedig ez az érték csak egy adott cellában fordulhat elő, tehát ide beírhatjuk azt. Aszerint, hogy a rejtett cellát milyen házban találtuk, meg azt mondjuk, hogy a cella rejtett a sorban, oszlopban vagy blokkban.

Az ábrán, a 6. oszlopban az 1-es rejtett, mert csak a 3. sorban szerepel, ezért oda be tudjuk írni, a megfelelő házakból pedig kizárhatjuk.

Tégla (intersection):



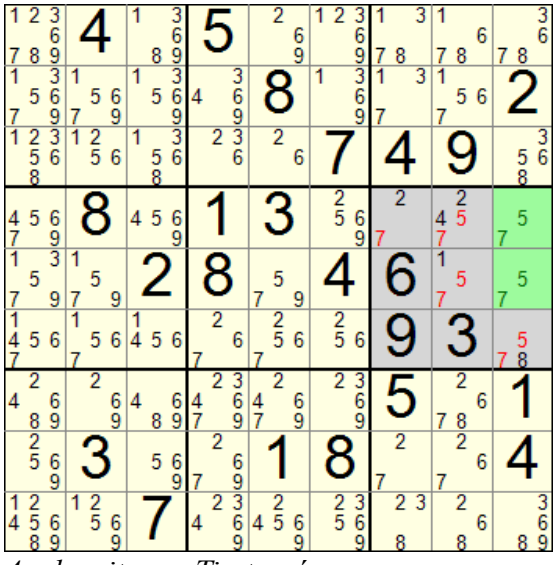
3. algoritmus: Tégla

Def.: Téglának nevezzük az olyan cella N-eseket, amelyek két ház – oszlop és blokk vagy sor és blokk – metszeteként jönnek létre.

Ha egy jelölt egy háznak csak egy téglájában fordul elő, akkor a tégla-metszetet alkotó másik házban is ott kell lennie, azaz ebben a házban minden olyan pozíción illegális, amely a téglán kívül van. Egy tégla akkor hasznos, ha legalább egy illegális jelöltet találunk a segítségével.

Az ábrán, a zölddel színezett blokk 3. téglájában biztosan lesz egy 8-as adat, mert a blokkban máshol nem lehet. Ebből tudjuk, hogy a 6. oszlopban csak a 2. téglában lehet 8-as, ezért ebből az oszlopból a pirossal színezett 8-as jelöltek törölhetők.

Tiszta pár (naked pair):



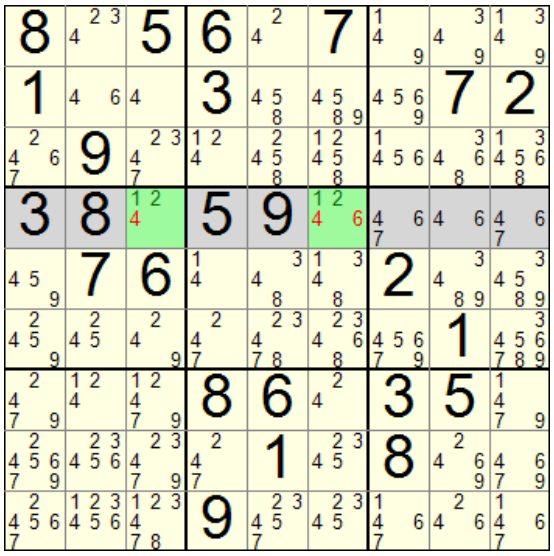
4. algoritmus: Tiszta pár

Olyan cellapárokat keresünk egy házon belül, amelyek csak ugyanazt a két jelöltet tartalmazzák. Ekkor tudjuk, hogy a megoldásban majd ezekben a cellákban a jelöltpár minden tagja fog szerepelni, igaz azt nem, hogy pontosan melyik hol. Eredményképpen pedig ezt a két értéket a ház többi cellájából törölhetjük.

Az ábrán, a zöld színű cellákban csak 5-ös és 7-es jelöltek szerepelnek, ezért itt majd a megoldásban is egyikük lesz. Ebből következik, hogy a blokkban ezen cellákon

kívül nem fordulhat elő egyik sem, tehát a szürke cellákból törölhetjük őket.

Rejtett pár (hidden pair):

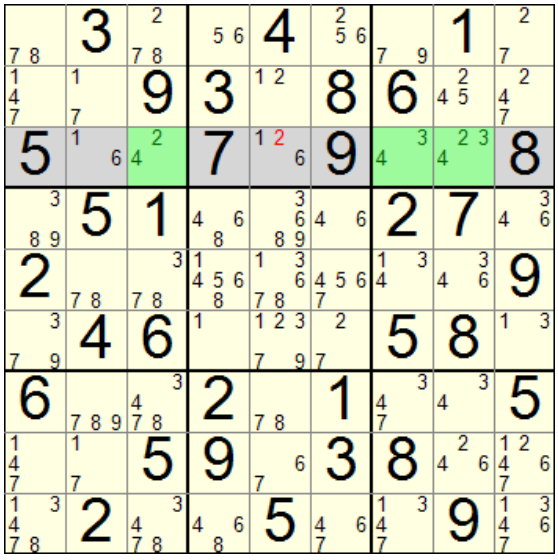


5. algoritmus: Rejtett pár

Olyan cellapárokat keresünk, amelyekben van két olyan jelölt, amelyek a házban egyetlen másik cellában sem szerepelnek. Mivel mindkét jelölt benne kell, hogy legyen a házban, a két cella egyikében lesznek majd, vagyis a cellapárból törölhetjük az összes egyéb jelöltet.

A 4. sorban az 1-es és 2-es jelöltek csak a zölddel színezett cellákban fordulnak elő, ezért majd a megoldásban is itt fognak szerepelni, vagyis ezekből a cellákból a többi – pirossal jelzett – jelölt törölhető.

Tiszta hármas (naked triple):

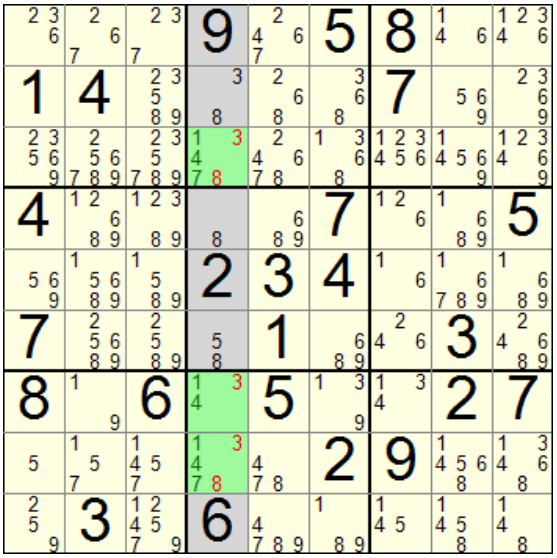


6. algoritmus: Tiszta hármas

A tiszta pár algoritmussal analóg módon most cella hármasokat keresünk, ahol vesszük a jelölthalmazok unióját. Ha három elemű halmazt kapunk, akkor tudjuk, hogy ebben a három cellában az említett unió halmaz egy-egy eleme lesz beírva, tehát a ház többi cellájából törölhetjük őket.

A zöld színű cellákban csak 2-es, 3-as és 4-es jelöltek szerepelnek, ezért itt majd a megoldásban is egyikük lesz. Ebből következik, hogy a sorban ezen cellákon kívül nem fordulhatnak elő, tehát a szürke cellákból törölhetjük ezeket a jelölteket.

Rejtett hármas (hidden triple):

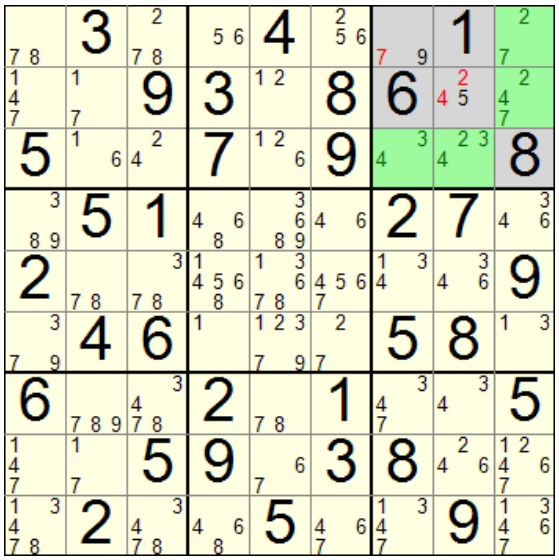


7. algoritmus: Rejtett hármas

Olyan jelölt hármasokat keresünk, amelyek elemei egy ház tetszőleges három cellájában fordulnak elő. Ha ilyet találtunk, a cella hármasból törölni tudjuk az összes többi jelöltet, mert a megoldásban majd a jelölt hármas egyik eleme fog szerepelni.

A 4. oszlopban az 1-es, 4-es és 7-es jelöltek csak a zölddel színezett cellákban fordulnak elő, ezért majd a megoldásban is itt fognak szerepelni, vagyis ezekből a cellákból a többi – pirossal jelzett – jelölt törölhető.

Tiszta négyes (naked quad):

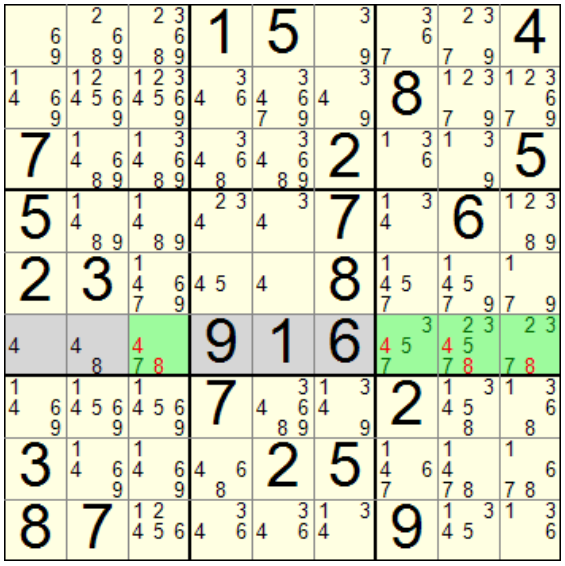


8. algoritmus: Tiszta négyes

Cella négyeseket keresünk, ahol vesszük a jelölthalmazok unióját. Ha négy elemű halmazt kapunk, akkor tudjuk, hogy ebben a négy cellában az említett unió halmaz egy-egy eleme lesz beírva, tehát a ház többi cellájából törölhetjük őket.

A zöld színű cellákban csak 2-es, 3-as, 4-es és 7-es jelöltek szerepelnek, ezért itt majd a megoldásban is egyikük lesz. Ebből következik, hogy ebben a blokkban ezen cellákon kívül nem fordulhatnak elő, tehát a szürke cellákból törölhetjük ezeket a jelölteket.

Rejtett négyes (hidden quad):



9. algoritmus: Rejtett négyes

Olyan jelölt négyeseket keresünk, amelyek elemei egy ház tetszőleges négy cellájában fordulnak elő. Ha ilyet találtunk, a cella négyesből törölni tudjuk az összes többi jelöltet, mert a megoldásban majd a jelölt négyes egyik eleme fog szerepelni.

A 6. oszlopban a 2-es, 3-as, 5-ös és 7-es jelöltek csak a zölddel színezett cellákban fordulnak elő, ezért majd a megoldásban is itt fognak szerepelni, vagyis ezekből a cellákból a többi – pirossal jelzett – jelölt törölhető.

Nishio:

Teszteli a jelölteket, azaz elmenti a táblát, és beír egy értéket, majd leellenőrzi, hogy a korábban bemutatott algoritmusokkal meg tudja-e oldani, ha nem visszaolvassa a tippelés előtti jelölt táblát. Ha a megoldás azért volt sikertelen, mert ellentmondásra jutott, akkor ez mellett törli az utolsóként megtippelt értéket a jelöltek közül. Az algoritmus akkor áll le, ha sikeresen megoldotta a feladványt, vagy az összes jelölt végigpróbálása után sem sikerült.

Crook:

A Nishio-hoz hasonlóan visszalépéses algoritmusról van szó, azonban míg a Nishio csak egy mélységig megy el, a Crook tetszőleges mélységű visszalépést megenged. A gyakorlatban ez annyit jelent, hogyha a hagyományos algoritmusok úgy akadtak el, hogy nincs ellentmondás a táblában, nem lépünk vissza, hanem egy újabb jelölt beírásával mélyebbre lépünk a keresőfában. Az algoritmus, mint minden visszalépéses keresés, minden megoldható tábla esetében sikeresen terminál, azaz talál egy megoldást.

7. Naplókészítés

A program fontos funkciója a megoldási napló készítés. Célja, hogy a megoldás teljes folyamatát nyomon tudjuk kísérni. A napló szerkezete:

- Az induló adatok feltüntetése.
- Megoldás folyamata: Minden algoritmus indulásakor naplóba kerül az aktuális jelölttábla – beállítások menüpontba kikapcsolható -, illetve az algoritmus neve. Továbbiakban az algoritmus minden sikeres találatához egy bejegyzést készítünk, ami tartalmazza a változtatást, illetve az ennek során keletkezett új mintákat.
- Ha találtunk megoldást, akkor ennek megjelenítése.
- Rejtvény típusának meghatározása.

A program naplózza a felhasználó lépéseit, és kezelni tudja a lépések visszavonását is, azaz a semmissé tett lépésekhez tartozó bejegyzést törli. Az így eredményül kapott napló, ha a lépések jó sorrendben lettek elvégezve, azonos az automatikus megoldás során generált naplóval, tehát tudjuk megoldásunk Rózsa algoritmus szerinti helyességét ellenőrizni.

Rejtvény típusok:

Egy sudoku típusát az alábbi alakban szoktuk megadni:

<rejtvény típus> := <rejtvény mérete> <rejtvény megoldhatósága> <megoldás nehézsége> - <adatok száma>

<rejtvény mérete>	- A blokkok mérete.
<rejtvény megoldhatósága>	- Értéke „A” megoldható feladvány esetében, „U” ellentmondásos feladvány esetében
<megoldás nehézsége>	- A megoldás során felhasznált legbonyolultabb algoritmus sorszáma, megoldhatatlan rejtvény esetében értéke 0 lesz.
<adatok száma>	- A feladványban szerepló kezdőadatok száma.

Például **3A2-32** egy 3×3-as, TISZTA és REJTETT algoritmusok segítségével megoldható, kezdetben 32 adatot tartalmazó rejtvényt jelent.

Megjegyzés: Léteznek több megoldással rendelkező rejtvények is, ezeket **[3-Iványi2009]** alapján „S”-el kellene jelölni, viszont a program nem tudja őket felismerni, így az ő besorolásuk is „A” lesz.

8. Program korlátozásai

A program az alábbi megkülönböztetéseket teszi az egyes rejtvény méretek között:

- Rejtvény tulajdonságainak állítása csak a 9×9-es méretben lehetséges, ott is csak .sdk formátum esetén.
- Rejtvény vágólapra illesztése sorfolytonos formában csak 9×9-es méretben lehetséges.
- Véletlen rejtvény generálása csak 4×4, 9×9, 16×16-os esetben lehetséges.
- A program nem naplózza, ha a felhasználó egyenként, nem minta alapján zár ki jelölteket.
- Visszalépés esetén korábbi táblát csak akkor olvashatunk vissza, ha ellentmondást kaptunk a megoldás során.

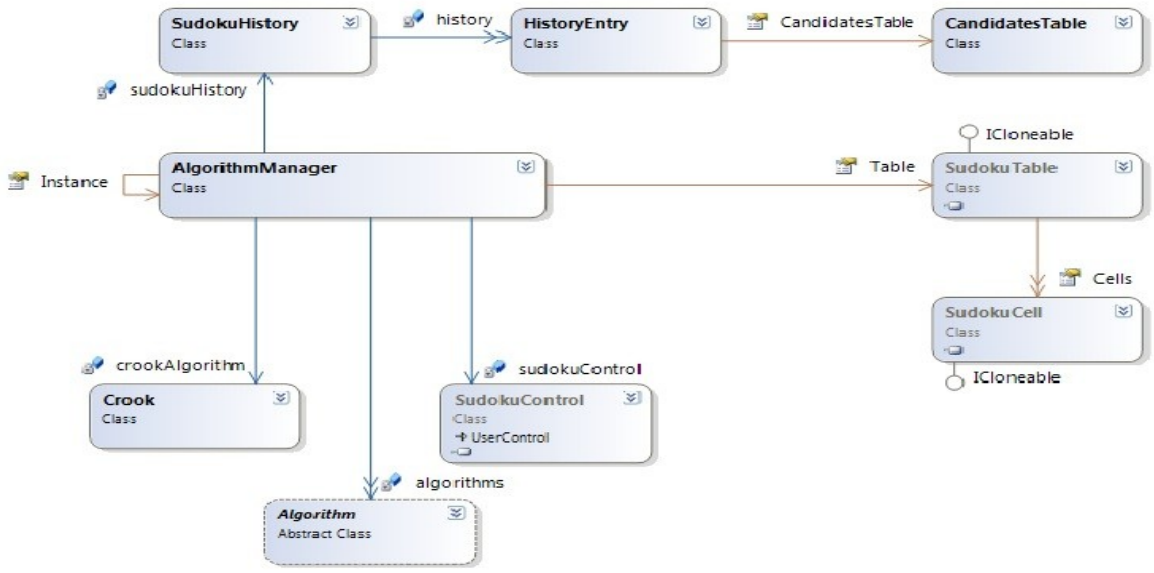
III. Fejlesztői dokumentáció

1. Elnevezések

Cella:	A rácsot alkotó kis négyzet.
Jelölt:	Egy mezőben lehetséges érték.
N:	Blokk oldalszélessége.
M:	M=N*N, azaz a rejtvény oldalának mérete, a felhasznált ábécé számossága.
Blokk:	N×N-es négyzet.
Ház:	M mezőből álló sor, oszlop, blokk közös neve.
Szomszéd:	Egy mező egy másiknak szomszédja, ha vele azonos házban van.
Minta:	Egy algoritmus találata.
Piros jelölt:	Egy minta által kihúzásra kijelölt jelölt.
Zöld jelölt:	Olyan jelölt, amit az algoritmus felhasznált egy minta megtalálásához.

2. A program felépítése

A program egy központi, `AlgorithmManager` nevű osztály köré épül. Ez egy singleton osztály, melynek feladata, hogy összehangolja a program különböző részeinek működését. Tartalmazza a munkatáblát, amelyet ily módon kényelmesen el tudunk érni a programból bárhonnan. Ez azért fontos, mert az összes algoritmus közösen fogja használni. További feladatai, hogy az algoritmusok közül kijelöli, melyek és milyen sorrendben fognak futni. Amikor módosítjuk a táblát, naplózza azt, és gondoskodik arról, hogy a módosítás megjelenjen a képernyőn is.



3. Osztályok leírása

3.1 Adatszerkezetek:

3.1.1 SudokuCell: (SudokuCell.cs)

`Icloneable` interfészt megvalósító, cella jelöltjeit tartalmazó adatszerkezet. A jelölteket egy `bool[] candidate` tömbben tárolja, hogy egy jelölt meglétének lekérdezése gyorsan történhessen. Továbbá tárolja a jelölteknek és magának a mezőnek, színére vonatkozó adatokat.

Attribútumai:

`public int BackGroundColor`	A mező hátterének színe.
`public int CandidatesNumber`	Az legális jelöltek száma.
`public int M`	A mező ábécéjének számossága.
`public int Data`	A mezőbe írt adat.
`public int Foundby`	A beírt adatot visszalépés milyen mélységében találtuk.

Függvényei:

`public void AddCandidate(int c)`	A `c` jelöltet hozzáadja a mezőhöz.
`public void RemoveCandidate(int c)`	A `c` jelöltet törli a mezőből.
`public bool IsCandidate(int c)`	Jelzi, hogy `c` jelöl-e a mezőben.
`public bool IsData()`	Jelzi, hogy van-e adat a mezőben.
`public void ResetCell()`	Visszaállítja a mezőt a kezdeti állapotba.
`public int GetColor(int candidate)`	Visszaadja a mező `candidate` értékű jelöltjének színét.
`public void SetColor(int candidate, int color)`	Beállítja a mező `candidate` értékű jelöltjének színét `color`-ra.
`public static bool IsLegalCharacter(char ch, int m)`	Jelzi, hogy `ch` érvényes jelölt-e, ha a mező mérete m.
`public static bool CompareByCandidates(SudokuCell cell1, SudokuCell cell2)`	Jelzi, hogy `cell2` a jelölteket tekintve megegyezik-e a `cell1`-el.
`public static bool Subset(SudokuCell cell1, SudokuCell cell2)`	Jelzi, hogy `cell2` a jelölteket tekintve részhalmaza-e a `cell1`-nek.
`public static bool Disjunct(SudokuCell cell1, SudokuCell cell2)`	Jelzi, hogy a két mező diszjunkt-e jelöltjeiket tekintve.
`public static string ConvertCellDataToString(int c, int m)`	Visszatér a rejtvény méretétől függően a `c`-nek megfelelő `string`-gel, azaz 9×9-es esetig magával a `c` -vel, felette a neki megfelelő betűvel.
`public static int ConvertCharToCellData(char ch)`	Egy karaktert alakít át a tárolási formának megfelelő `integer`-ré.

3.1.2 SudokuTable: (SudokuTable.cs)

`Icloneable` interfészt megvalósító, M*M-es `SudokuCell` tömböt tartalmazó adatszerkezet. A jelölttömbön felül nyilvántartja a meglévő adatok számát, továbbá egyéb leíró adatokat, melyek jellemzik a rejtvényt.

Attribútumai:

`public int DataInTable`	A táblába beírt adatok száma.
`public string[] Properties`	A rejtvény tulajdonságai (szerző, forrás, …).
`public string Filename`	A fájl neve, amiből a rejtvényt kiolvastuk.

`public SudokuCell[,] Cells`	A rejtvény celláit tartalmazó tömb.
`public int N`	A rejtvény egy blokkjának mérete.
`public int M`	A rejtvény mérete (M=N*N).

Függvényei:

`public void setCellData(int i, int j, int c)`	Beállítja `(i,j)` mezőben az adatot.
`public int getCellData(int i, int j)`	Visszaadja `(i,j)` mezőben található adatot.
`public bool RemoveCandidate(int i, int j, int c)`	Kitörli `(i,j)` mezőből a `c` jelöltet, és jelzi, hogy a törlés érvényes volt-e.
`public void HideShownPattern()`	Eltünteti az aktuálisan táblára kirajzolt mintát.
`public bool DeleteCandidateFromRow(int x, int y, int candidate)`	Kitörli a `candidate` jelöltet az `(x,y)` sorának mezőiből.
`public bool DeleteCandidateFromColumn(int x, int y, int candidate)`	Kitörli a `candidate` jelöltet az `(x,y)` oszlopának mezőiből.
`public bool DeleteCandidateFromBlock(int x, int y, int candidate)`	Kitörli a `candidate` jelöltet az `(x,y)` blokkjának mezőiből.
`public bool DeleteCandidateFromThreeHouse(int x, int y, int candidate)`	Kitörli a `candidate` jelöltet az `(x,y)` szomszédaiból.
`public void AddCandidateToRow(int x, int y, int candidate)`	Hozzáadja a `candidate` jelöltet az `(x,y)` sorának minden mezőjéhez.
`public void AddCandidateToColumn(int x, int y, int candidate)`	Hozzáadja a `candidate` jelöltet az `(x,y)` oszlopának minden mezőjéhez.
`public void AddCandidateToBlock(int x, int y, int candidate)`	Hozzáadja a `candidate` jelöltet az `(x,y)` blokkjának minden mezőjéhez.
`public void AddCandidateToThreeHouse(int x, int y, int candidate)`	Hozzáadja a `candidate` jelöltet az `(x,y)` szomszédaihoz.
`public bool ValidString(string str)`	Ellenőrzi, hogy az adott `string` érvényes.
`public bool LoadFromString(string str)`	Feltölti a táblát az adott `string`-ben található adatokkal.
`public string SaveToString()`	Előállítja az adott táblának adatait sorfolytonos leírásban.
`public string CandidateTable()`	Előállítja az adott táblának a jelölttábláját.
`public bool LoadFromSDXFile(string`	Feltölti a táblát a megadott elérésű .sdx fájl adataival.

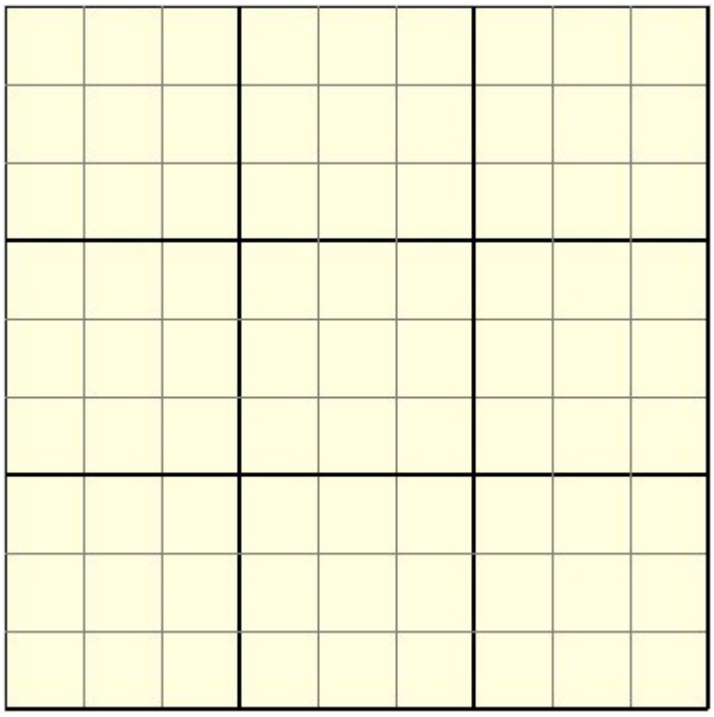
file)	
public bool LoadFromSDXXFile(string file)	Feltölti a táblát a megadott elérésű .sdxx fájl adataival.
public bool LoadFromSDKFile(string file)	Feltölti a táblát a megadott elérésű .sdk fájl adataival.
public bool SaveToSDKFile(string file)	Elmenti a táblát a megadott elérésű fájlba .sdk formátumban.
public bool SaveToSDXFile(string file)	Elmenti a táblát a megadott elérésű fájlba .sdx formátumban.
public bool SaveToSDXXFile(string file)	Elmenti a táblát a megadott elérésű fájlba .sdxx formátumban.

3.2 Vezérlők

3.2.1 SudokuControl: (SudokuControl.cs, SudokuControl.Designer.cs)

Egy `SudokuTable` típusú objektum szerkesztését teszi lehetővé. A mérete szabadon állítható.

Grafikus megjelenés:



Attribútumai:

`public IntPair SelectedCell`	A vezérlőn kijelölt cella.
`public SelectionRectangleSize SelectionSize`	A kijelölés mérete (egész cella vagy a cella egy jelöltje).
`public FontFamily Bigfontfamily`	A beírt adatok betűtípusa.
`public FontFamily Littlefontfamily`	A jelöltek betűtípusa.
`public bool Pencilmarks`	A táblán jelenítsük-e meg a jelölteket?
`public int M`	A rejtvény egy blokkjának mérete.
`public int N`	A rejtvény mérete (M=N*N).

Függvényei:

`public void UpdateTable(SudokuTable t)`	Új `SudokuTable` típusú objektumot fogad, melyet megjelenít.
`public void Reset(int size)`	Alapállapotra hozza a vezérlőt, a megadott méretben. (N értéket vár)
`public void SelectCell(int i, int j)`	Egy mezőt kijelölt.
`public void UnSelectCell()`	Leveszi egy mező kijelölését.
`public void HighlightReset()`	Kiüríti a kiemelt mezők listáját.
`public void Highlight(int c, Color color)`	Kiemeli a c -t jelöltként tartalmazó mezőket hátterük színének megváltoztatásával.
`protected void OnTableChanged(object o, SudokuTableEventArgs e)`	Kiváltja a `TableChanged` eseményt.
`protected void OnWarningEvent(object o, string message)`	Kiváltja a `WarningEvent` eseményt.
`public void CopyImageToClipboard()`	A tábla adott megjelenését fekete-fehér képként a vágólapra másolja.
`public void DrawSudokuTable(Graphics g, Color color)`	Megrajzolja a vezérlőt.
`protected override void OnPaint(PaintEventArgs e)`	Meghívja a `DrawSudokuTable` függvényt.
`protected override void OnKeyPress(KeyPressEventArgs e)`	A billentyűparancsokat dolgozza fel.
`protected override bool ProcessDialogKey(Keys keyData)`	A nyilak, illetve a `Delete` gomb lenyomását kezeli.
`protected override void OnMouseDown(MouseEventArgs e)`	Az egér kattintását kezeli.

3.3 Form-ok

3.3.1 **MainForm**: (MainForm.cs, MainForm.Designer.cs)

Vezérlő elemek:

Név	Típus	Funkció
saveFileDialog	SaveFileDialog	Rejtvény mentése
mainmenu	MenuStrip	Menüsor
statusStrip	StatusStrip	Jelöltek mutatása

Metódusok:

`void MyInit(int n)`	Inicializálja a form-ot n-es táblaméretthez.
`void WriteCandidate(object sender, MouseEventArgs e)`	Jelölt beírása a táblába.
`void RemoveCandidate(object sender, MouseEventArgs e)`	Jelölt kihúzása cellából.
`private void MainForm_Resize(object sender, EventArgs e)`	Form újraméretezése.
`private void sudokuControl_TableChanged(object o, SudokuTableEventArgs e)`	Felhasználó megváltoztatta a táblát.
`void report_FormClosed(object sender, FormClosedEventArgs e)`	Bezáródott a napló ablaka.
`private void pasteToolStripMenuItem_Click(object sender, EventArgs e)`	Vágólapról beillesztés.
`private void copytoclipboardastextToolStripMenuItem_Click(object sender, EventArgs e)`	Vágólapra másolás 0..9-es rejtvényként
`private void copytoclipboardaspictureToolStripMenuItem_Click(object sender, EventArgs e)`	Vágólapra másolás fekete-fehér képként.
`private void restarttableToolStripMenuItem_Click(object sender, EventArgs e)`	Rejtvény újrakezdése.
`private void pencilmarksToolStripMenuItem_Click(object sender, EventArgs e)`	Jelöltek mutatása.
`private void`	Napló mutatása.

`reportToolStripMenuItem_Click(object sender, EventArgs e)`	
`private void összesSegítségToolStripMenuItem_Click(ojbect sender, EventArgs e)`	Segítségek mutatása.
`private void savereportMentéseToolStripMenuItem_Click(object sender, EventArgs e)`	Napló mentése.
`private void newToolStripMenuItem_Click(object sender, EventArgs e)`	Új rejtvény.
`private void openToolStripMenuItem_Click(object sender, EventArgs e)`	Rejtvény beolvasása.
`private void openwww_dailysudoku_MenuItem_Click(object sender, EventArgs e)`	Rejtvény beolvasása internetről.
`private void generatorMenuItem_Click(object sender, EventArgs e)`	Véletlen rejtvény előállítása.
`private void saveToolStripMenuItem_Click(object sender, EventArgs e)`	Rejtvény elmentése.
`private void saveasToolStripMenuItem_Click(object sender, EventArgs e)`	Rejtvény elmentése másként.
`private void propertiesToolStripMenuItem_Click(object sender, EventArgs e)`	Rejtvény tulajdonságainak megtekintése.
`private void n2ToolStripMenuItem_Click(object sender, EventArgs e)`	Rejtvény méretének 2×2-re állítása.
`private void n3ToolStripMenuItem_Click(object sender, EventArgs e)`	Rejtvény méretének 3×3-ra állítása.
`private void n4ToolStripMenuItem_Click(object sender, EventArgs e)`	Rejtvény méretének 4×4-re állítása.
`private void n5ToolStripMenuItem_Click(object sender, EventArgs e)`	Rejtvény méretének 5×5-re állítása.
`private void exitToolStripMenuItem_Click(object sender, EventArgs e)`	Kilépés.
`private void algoritmusokBeállításaToolStripMenu`	Algoritmusok beállítása.

`Item_Click(object sender, EventArgs e)`	
`private void solveToolStripMenuItem_Click(object sender, EventArgs e)`	Megoldás.
`private void utolsóLépésVisszavonásaToolStripMenuItem_Click_1(object sender, EventArgs e)`	Utolsó változtatás visszavonása.
`private void crookTáblaVisszaolvasásaToolStripMenuItem_Click(object sender, EventArgs e)`	A Crook algoritmus utolsó érvényes táblájának visszaolvasása.
`private void aboutToolStripMenuItem_Click(object sender, EventArgs e)`	Névjegy.
`private void helpcontentToolStripMenuItem1_Click(object sender, EventArgs e)`	Segítség a program használatához.

3.3.2 **Report**: (Report.cs, Report.Designer.cs)

A napló megjelenítésére és formázására szolgáló ablak.

Vezérlő elemek:

Név	Típus	Funkció
`menuStrip`	`MenuStrip`	Naplózás beállításai.
`richTextBox`	`RichTextBox`	Napló megjelenítése.

Metódusok:

`private void jelölttáblákMutatásaToolStripMenuItem_Click(object sender, EventArgs e)`	A naplóban jelenjenek-e meg a jelölttáblák, amikor új algoritmus kerül sorra?
`private void megoldásEgyszerűToolStripMenuItem_Click(object sender, EventArgs e)`	A megoldás egyszerűsítve jelenjen-e meg a naplóban?

3.3.3 **Hints**: (Hints.cs, Hints.Designer.cs)

Vezérlő elemek:

Név	Típus	Funkció
tvHints	TreeView	Segítségek megjelenítése.
btnApply	Button	Kijelölt minta alkalmazása.
btnUndo	Button	Utolsó minta visszavonása.

Metódusok:

`private void Hints_Load(object sender, EventArgs e)`	Ablak láthatóvá tételekor minták frissítése.
`public void PopulateHints()`	Minták betöltése.
`private void twHints_AfterSelect(object sender, TreeViewEventArgs e)`	Minta kiválasztása.
`private void btnApply_Click(object sender, EventArgs e)`	Minta alkalmazása.
`private void Hints_Resize(object sender, EventArgs e)`	Ablak átméretezése.
`private void btnUndo_Click(object sender, EventArgs e)`	Utolsó lépés visszavonása.

3.3.4 **AlgorithmSettings**: (AlgorithmSetting.cs, AlgorithmSettings.Designer.cs)

A program által használható algoritmusok megadása. Az „egyszerű” algoritmusokat jelölőnégyzetek segítségével tudjuk ki-be kapcsolni, a visszalépésesek közül pedig választó gombokkal lehet egyet kiválasztani.

Vezérlő elemek:

Név	Típus	Kezdeti érték
chbNakedSingle	CheckBox	“Tiszta”
chbHiddenSingle	CheckBox	“Rejtett”
chbIntersection	CheckBox	“Tégla”
chbNakedPairs	CheckBox	“Tiszta pár”
chbHiddenPair	CheckBox	“Rejtett pár”

chbNakedTriple	CheckBox	“Tiszta hármas”
chbHiddenTriple	CheckBox	“Rejtett hármas”
chbNakedQuad	CheckBox	“Tiszta négyes”
chbHiddenQuad	CheckBox	“Rejtett négyes”
groupBox	GroupBox	“Visszalépéses algoritmusok”
rbOff	RadioButton	“Kikapcsolva”
rbNishio	RadioButton	“Nishio”
rbCrook	RadioButton	“Crook”
rbDancingLinks	RadioButton	“Dancing links”
btnOK	Button	”OK”

3.3.5 SetProperties: (SetProperties.cs, SetProperties.Designer.cs)

Az aktuális rejtvényhez tudunk megjegyzéseket megadni, amelyek .sdk formátumban történő elmentéskor a táblával együtt a fájlba íródnak. Megadható a rejtvény leírása, szerzője, a létrehozásának dátuma, forrása, nehézsége, változata, URL forrása és további tetszőleges megjegyzés.

Vezérlő elemek:

Név	Típus	Kezdeti érték
label1	Label	„Leírás”
label2	Label	„Szerző”
label3	Label	„Létrehozás dátuma”
label4	Label	„Forrás”
label5	Label	„Nehézség”
label6	Label	„Megjegyzés”
label7	Label	„Változat”
label8	Label	„Forrás URL”
desctb	TextBox	
authb	TextBox	
createtb	TextBox	
sourcetb	TextBox	
difftb	TextBox	

commtb	TextBox	
vartb	TextBox	
scurltb	TextBox	
okbtn	Button	„OK”
cancelbtn	Button	„Mégse”

3.3.6 GeneratorForm: (GeneratorForm.cs, GeneratorForm.Designer.cs)

A SudokuGenerator osztály egy példányának állapotát mutatja, melyet külön szálon futtat egy BackgroundWorker objektum segítségével. A SudokuGenerator először megpróbál egy helyesen kitöltött táblát előállítani, majd ha ez sikerül egyszerűsít, az egyes adatok egyenkénti elhagyásával.

Vezérlő elemek:

Név	Típus	Kezdeti érték
label1	Label	„Kipróbált tábla:”
label2	Label	„0/0”
label3	Label	„Aktuális elemszám:”
label4	Label	„0”
progressBar1	ProgressBar	0
button1	Button	„Leállítás”

3.3.7 About: (about.cs, about.designer.cs)

A készítők névjegye.

3.4 Naplózás

A program az automatikus rejtvényoldás mellett a felhasználó által beírt adatokat is naplózza. Mivel a háttérben folyamatosan frissíti a megtalálható minták listáját, ha beírtunk egy értéket, akkor ellenőrizni tudjuk, hogy milyen algoritmus mintái közt szerepel az adott lépés. A TISZTA és REJTETT algoritmusok esetében ezt könnyű megtenni, csupán végig kell nézni, hogy ezen algoritmusok találatai közt van-e olyan, ami pont ezt a

jelöltet írja a táblába. Ha van ilyen minta, akkor a felhasználó változtatása ekvivalens ennek a mintának alkalmazásával, ezért a naplóban úgy tüntethetjük fel, mintha utóbbi történt volna. Ha nem találunk ekvivalens mintát, akkor két lehetőség állhat fenn, nevezetesen, hogy a tábla adatokkal való feltöltése van folyamatban, vagy éppen tippelést alkalmazunk. Ennek eldöntése nem minden esetben lehetséges, ami azt eredményezi, hogy a napló és a táblán megjelenő számok színezése elromolhat. Ha egy beolvasott, vagy generált táblát kezdünk el megoldani, akkor ez a probléma nem lép fel, mert azt feltételezzük, hogy csak a beolvasott értékek lesznek adatok.

Például kezdjük egy üres táblába beírni az adatokat, majd ha ezzel megvoltunk, azonnal oldjuk is meg a rejtvényt. Honnan fogja tudni az algoritmus, hogy mikor vagyunk készen az adatok feltöltésével? Melyik volt az első olyan beírt érték, amelynek a naplóban már nem adatként kell szerepelnie? A programban ez úgy történik, hogy a megoldás kezdetének vesszük az első olyan módosítást, amelyet már a többi beírt adat alapján ki tudtunk volna következtetni. Ez azonban nem ad mindig pontos eredményt. Előfordulhat, hogy könnyű rejtvény esetében, ahol sok az adat, vannak olyanok, amelyek kikövetkeztethetők lennének a többiből. Ekkor az első ilyen beírásakor a program úgy fogja venni, hogy elkezdtük megoldást, ezután viszont, ha olyan értéket írunk, ami nem REJTETT vagy TISZTA cella, Crook-ként kerül be a naplóba. Hasonlóan, egy nehéz rejtvény esetében, ha első lépésben nem találunk semmilyen alkalmazható algoritmust, már rögtön tippelnünk kell, viszont erről a program azt fogja hinni, hogy még egy adatot írtunk be a táblába.

Egy másik gyengéje a naplókészítésnek, hogy ha a felhasználó jelölteket zár ki cellákból, akkor azok nem kerülnek naplózásra. Ennek oka, hogy a módosítások általában nem feleltethetők meg egy bejegyzéssel, hiszen egy minta többnyire több jelöltet is kizár. Megoldás az lenne, ha kizárások egy halmazából állítanánk elő naplóbejegyzést. Ehhez meg kellene állapítani, hogy mely kizárások tartoznak össze. A probléma az, hogy előfordulhat, hogy egy kizárás több bejegyzéshez is tartozhat, illetve hogy semmi sem garantálja, hogy az egy mintához tartozó kizárásokat a felhasználó egymás után lépi meg. Ilyen körülmények közt az összetartozások megállapítása nagyon nehéz feladat lenne.

3.4.1 SudokuHistory: (SudokuHistory.cs)

Tartalmazza a rejtvény megoldásának menetét. A `GetReport` függvénnyel tudjuk valós időben generálni a napló szövegét .rtf formátumban. A táblán történő változtatásokat, egy `HistoryEntry` típusú objektummal reprezentáljuk, amelyeket egy listában tárolunk. A program általában megengedi az utolsó lépés visszavonását (undo), ami abból áll, hogy töröljük a listából az utolsó elemet, és visszaállítjuk annak hatásait. Bizonyos esetekben azonban veszélyes lehet ez a művelet (lásd Crook algoritmus **[7-Crook]**).

Attribútumai:

`public HistoryEntry LastEntry`	Visszaadja a történet utolsó bejegyzését.
`public bool IsEmpty`	Üres-e a történet?
`public bool UndoIsAllowed`	Adott állapotban az utolsó lépés visszavonása engedélyezve van-e?
`public bool ShowCandidatesTable`	Új algoritmus szóhoz jutásánál a naplóban legyen-e benne a jelölttábla?
`public bool ShowSimpleResult`	A rejtvény megoldása a naplóban egyszerűsített formában szerepeljen-e?

Függvényei:

`public void ClearHistory()`	Kiüríti a történetet.
`public void ResetHistory()`	Visszaállítja a történetet arra a pontra, amikor a megoldást elkezdtük.
`public void AddEntry(HistoryEntry newEntry, bool tryGuess)`	Új bejegyzést ad hozzá a történethez. A `tryGuess` paraméterrel tudjuk megadni, hogy a `SudokuHistory`-nek ki kell-e találnia, hogy a bejegyzést milyen algoritmus alapján hoztuk létre (akkor, ha a felhasználó írt be jelöltet segítség nélkül).
`public void RemoveLastEntry()`	Eltávolítja a történetből az utolsó bejegyzést.
`public string GetReport(int tableSize)`	Generálja és visszaadja a megoldási naplót.

3.4.2 HistoryEntry: (HistoryEntry.cs)

Egy bejegyzés a történetben.

Attribútumai:

`public CandidateTable CandidatesTable`	A rejtvény jelölttáblája, ha egy újonnan sorra került algoritmus találta meg, egyébként `null`.
`public int Step`	A bejegyzés sorszáma.
`public string Text`	A bejegyzés szövege
`public int Depth`	A bejegyzések mélysége
`public int AlgorithmNumber`	A megtaláló algoritmus sorszáma

Függvényei:

`public void GuessType(int LastAlgorithmIndex, int EntryCount)`	Megpróbálja kitalálni melyik algoritmus alapján lett létrehozva.
`public void Undo()`	Visszavonja a bejegyzést létrehozó minta változtatásait.

3.4.3 **CandidatesTable**: (CandidatesTable.cs)

Tárolja a jelölttáblát

Függvényei:

`public string getCandidateTable()`	Visszaadja a jelölttábla `string` reprezentációját.
`public string getSimpleTable()`	Visszaadja a kitöltött tábla egyszerűsített `string` reprezentációját

3.5 Program működésének összehangolása

3.5.1 **AlgorithmManager**: (AlgorithmManager.cs)

Singleton osztály, irányítja és összehangolja a program különböző részeinek működését.

Attribútumai:

`public AdvancedAlgorithms AdvancedAlgorithm`	Az aktuálisan használt visszalépéses algoritmus.
`public int SolveDepth`	Milyen mélységben járunk a megoldás adott pillanatában?
`public SolveStates SolveState`	A megoldás állapota (lehetséges értékei: `RozsaSolve`, `UserSolve`, `GettingHints`, `WritingData`).

`public SudokuTable Table`	A program ezt a táblát használja munkatáblának, azaz minden változtatás ezen történik.
`public List<List<Pattern>> Findings`	Az éppen megtalálható minták.
`public SudokuControl SudokuControl`	A sudoku vezérlő.
`public bool ShowCandidateTablesInReport`	A naplóban jelenjenek-e meg a jelölttáblák új algoritmus sorrakerülésekor?
`public bool ShowSimpleResult`	A naplóban a megoldás tábla egyszerűsítve jelenjen-e meg?

Függvényei:

`public bool GetAlgorithmState(int algindex)`	Lekérdezi, hogy az `algindex`-edik algoritmus be van-e kapcsolva.
`public void SetAlgorithmState(int algindex, bool state)`	Az `algindex`-edik algoritmust tudjuk vele ki-, illetve bekapcsolni.
`public void SetTableSize(int size)`	Módosított táblaméretet beállítása.
`public void RestartTable()`	A tábla megoldás elkezdését megelőző állapotba visszaállítása.
`public void StartNewPuzzle()`	Új, üres rejtvény kezdése.
`public void StartNewPuzzle(SudokuTable t)`	Új rejtvény kezdése, melynek kezdőtáblája `t`.
`public bool openwww(string url)`	Rejtvény megnyitása internetről.
`public void Solve()`	A rejtvény Rózsa algoritmus szerinti megoldása.
`public void ProcessQueue(int queueIndex)`	A `queueIndex`-edik algorimtus jelölt sorának feldolgozása.
`public bool ApplyPattern(Pattern p)`	A `p` minta alkalmazása és naplózása, ahol `alg` a mintát megtaláló algoritmus.
`public bool ApplyPattern(int alg, int ind)`	Az `alg`-adik algoritmus sorrendben `ind`-edik megtalált mintájának alkalmazása és naplózása.
`private void ResetAlgorithms()`	Algoritmusok visszaállítása kezdeti állapotukba.
`public void LoadCrookTable()`	Visszalépéses algoritmus hibával elakadása esetén visszaolvassa az utolsó olyan tippelés előtti táblát, ahonnan még van remény megoldásra.
`public void GetHints()`	Az összes fellelhető minta megkeresése.
`public void AddPattern(int algIndex, Pattern pattern)`	Hozzáadja `pattern` mintát az `algIndex`-edik algoritmus eddigi találataihoz.
`public void UndoLastPattern()`	Utolsóként alkalmazott minta változtatásainak visszaállítása, és az utolsó bejegyzés törlése a naplóból.
`public void RemoveCandidate(int i, int j, int c)`	Az `(i,j)` pozíción lévő `c`-értékű jelölt törlése.

`public void WriteCandidate(int i, int j, int c)`	Az `(i,j)` pozíción lévő `c`-értékű jelölt beírása a táblába.
`public string TestCell(IntPair redCell, IntPair greenCell, int redCand, int greenCand)`	Az `redCell` pozíción lévő `redCand`-értékű jelölt kihúzásakor keletkező minták megkeresése. A napló bejegyés szövegével tér vissza, vagy üres `string`-gel amennyiben csak később, a jelöltsor feldolgozásánál teszteli a jelöltet. A `greenCell`, `greenCand` változókkal azonosítjuk a mintát, ami a kihúzásokat eredményezi, szerepe a ismétlődések elkerülése.
`public void RemoveShownPattern()`	Eltünteti az aktuálisan kirajzolt mintát.
`public void ChangeShownPattern(int alg, int ind)`	Kirajzolja az `alg`-adik algoritmus által talált `ind`-edik mintát.

4. Fájlformátumok

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

SDK: 9×9-es tábla leírására alkalmas, a számok az adatok, a „.” az üres mező. Az SDK formátum tartalmazhat megjegyzéseket az adott feladványról **[9-Balázs2007]**.

```
#ASudoku Program
```

```
#B2007.06.05
#L3
29.7....5
4......6.
.....534.
......6..
5..4.8...
8.437....
9.52....1
72.......
......92.
```

SDX: 9×9-es tábla leírására alkalmas, a kapcsos zárójelben álló számok a jelöltek, a zárójelen kívüli számok az adatok **[9-Balázs2007]**.

```
2 9 {1368} 7 {13468} {1346} {18} {18} 5
4 {13578} {1378} {189} {12389} {1239} {1278} 6 {2789}
{16} {1678} {1678} {1689} {12689} 5 3 4 {2789}
{13} {137} {12379} {159} {1259} {129} 6 {135789} {234789}
5 {1367} {123679} 4 {1269} 8 {127} {1379} {2379}
8 {16} 4 3 7 {1269} {125} {159} {29}
9 {3468} 5 2 {3468} {3467} {478} {378} 1
7 2 {1368} {15689} {1345689} {13469} {458} {358} {3468}
{136} {13468} {1368} {1568} {134568} {13467} 9 2 {34678}
```

SDXX: $N^2 \times N^2$-es tábla leírására alkalmas, a kapcsos zárójelben álló számok a jelöltek vesszővel elválasztva, a zárójelen kívüli számok az adatok. A vessző lényege, hogy 9×9-es tábla felett egyértelmű legyen a jelöltek listája **[9-Balázs2007]**.

```
2 9 {1,3,6,8} 7 {1,3,4,6,8} {1,3,4,6} {1,8} {1,8} 5
4 {1,3,5,7,8} {1,3,7,8} {1,8,9} {1,2,3,8,9} {1,2,3,9} {1,2,7,8} 6 {2,7,8,9}
{1,6} {1,6,7,8} {1,6,7,8} {1,6,8,9} {1,2,6,8,9} 5 3 4 {2,7,8,9}
{1,3} {1,3,7} {1,2,3,7,9} {1,5,9} {1,2,5,9} {1,2,9} 6 {1,3,5,7,8,9} {2,3,4,7,8,9}
5 {1,3,6,7} {1,2,3,6,7,9} 4 {1,2,6,9} 8 {1,2,7} {1,3,7,9} {2,3,7,9}
8 {1,6} 4 3 7 {1,2,6,9} {1,2,5} {1,5,9} {2,9}
9 {3,4,6,8} 5 2 {3,4,6,8} {3,4,6,7} {4,7,8} {3,7,8} 1
7 2 {1,3,6,8} {1,5,6,8,9} {1,3,4,5,6,8,9} {1,3,4,6,9} {4,5,8} {3,5,8} {3,4,6,8}
```

```
{1,3,6} {1,3,4,6,8} {1,3,6,8} {1,5,6,8} {1,3,4,5,6,8} {1,3,4,6,7} 9 2 {3,4,6,7,8}
```

5. Algoritmusok

5.1 Bevezetés

A programban a Rózsa algoritmusait használjuk **[3-Iványi2009]**, kiegészítve Crook algoritmusával. Csoportosítsuk őket az alábbi módon:

- Nem visszalépéses algoritmusok csoportja. Ide tartozik a Tiszta cella, Rejtett cella, Tégla, Tiszta pár, Rejtett pár, Tiszta hármas, Rejtett hármas, Tiszta négyes és Rejtett négyes. Ezen algoritmusok egy `Algorithm` ősosztályból származnak.
- Visszalépéses algoritmusok csoportja. Ide tartoznak a Nishio, Crook és Dancing Links algoritmusok.

Minden algoritmusnak van sorszáma, amellyel azonosítani tudjuk a naplóbejegyzések létrehozóját. A sorszámokat az alábbi táblázat tartalmazza:

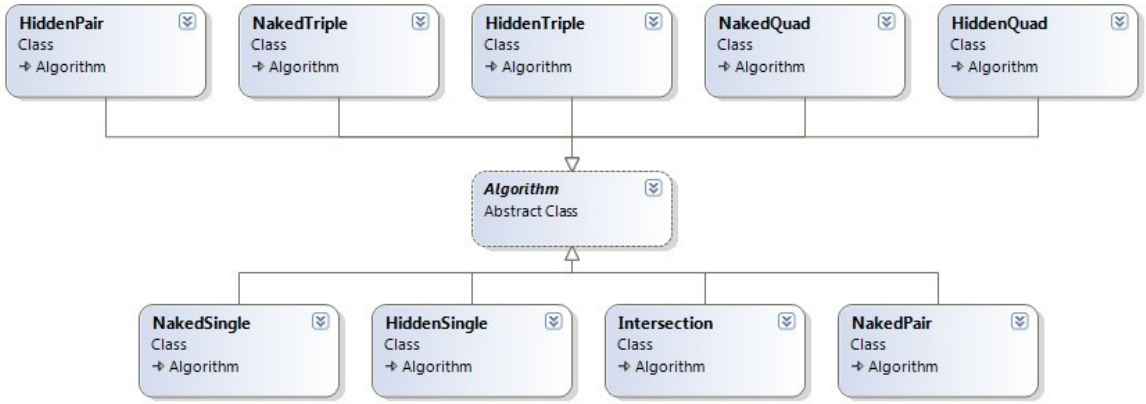
Algoritmus	Sorszám
-2	Közönséges jelölt törlés a táblázatból
-1	Adat
0	Tiszta cella
1	Rejtett cella
2	Tégla
3	Tiszta pár
4	Rejtett pár
5	Tiszta hármas
6	Rejtett hármas
7	Tiszta négyes
8	Rejtett négyes
9	Nishio

10	Crook

Megjegyzés: A Dancing links algoritmus speciális abból a szempontból, hogy lépéseit nem naplózzuk, ezért bejegyzéseket sem kell hozzá készíteni, és sorszáma sem lesz.

5.1.1 Algorithm: (Algorithm.cs)

A nem visszalépéses algoritmusokhoz tartozó absztrakt ősosztály. Leszármazottai egy tömbbe (`algorithms[]`) vannak rendezve. Megoldáskor, illetve segítségek keresésekor ezt a tömböt járjuk be, és minden algoritmusnak meghívjuk a `solve` metódusát.



Attribútumai:

`public string Name`	Az algoritmus neve.
`public bool Enabled`	Be van-e kapcsolva az algoritmus?

Függvényei:

`public abstract void solve(SudokuTable t)`	`t` tábla megoldása.
`public abstract string TestCell(SudokuTable t, IntPair redCell, IntPair greenCell, int redCand, int greenCand)`	`t` táblán `redCell` → `redCand` kihúzásakor keletkező mintákat keresi meg. `greenCell` → `greenCand` pedig a minta alkalmazása során a táblába beírt adat, ha volt ilyen.

A `solve` metódus feladata, hogy módszeresen bejárja az egész táblát, amennyiben sikeres találatra bukkan létrehoz egy speciális mintát – `Pattern` osztály leszármazottját –, és feldolgozza azt. Három dolgot tehet:

- Hozzáadja az `AlgorithmManager` találataihoz – beteszi a `Findings` adatszerkezetbe.
- Azonnal feldolgozza – ha a Rózsa algoritmus módszeres keresése van folyamatban.
- Eldobja – ha jelöltet írunk a táblába, automatikusan meghívódnak a megfelelő `TestCell` metódusok, amelyek adott esetben találhatnak mintát, viszont ha csak a táblát töltjük fel adatokkal, akkor ezekkel nem kell semmit tenni.

```
//A megtalált minta feldolgozása.
if (AlgorithmManager.Instance.SolveState == SolveStates.RozsaSolve)
{
      AlgorithmManager.Instance.ApplyPattern(p);
}
else if (AlgorithmManager.Instance.SolveState !=
SolveStates.WritingData)
{
      AlgorithmManager.Instance.AddPattern(p);
}
```

5.1.2 **Pattern**: (Pattern.cs)

Egy absztrakt osztály, amiből minden algoritmushoz saját gyerek osztályt származtatunk. Általában egy algoritmus sikeres találatát reprezentálja, de hibához is készíthetünk egy `Pattern` objektumot. Segítségével egy találatot később is felhasználhatunk, amennyiben azt eltároljuk az `AlgorithmManager` `AddPattern` metódusával. A tárolásra szolgáló adatszerkezet egy tömb, amelyben minden altípusú mintának egy saját listát tárolunk el. Így a „Segítség” ablakban bármikor meg tudjuk tekinteni az aktuális találatokat. Ha kiválasztunk innen egy mintát, akkor azt a `PaintPattern` metódussal kirajzoljuk a táblára. Kirajzolás során a megfelelő cellák `BackgroundColor` attribútumát változtatva a cellák háttérszínét állítjuk, a

`SetColor(int candidate, int color)` metódussal pedig az egyes jelöltek színét változtathatjuk. Az egyes színeknek az alábbi táblázatban szereplő értékek felelnek meg.

Szín	Érték
Black	0
DarkGreen	1
Red	2
LightYellow	3
LightGray	4
PaleGreen	5

Attribútumai:

`public int AlgorithmIndex`	A mintát létrehozó algoritmus sorszáma.
`public int CandidateValue`	A minta alkalmazása során táblába beírt adat, ha nincs ilyen, akkor értéke 0.
`public IntPair CandidatePosition`	A minta alkalmazása során kitöltött cella, ha nincs ilyen, akkor értéke `null`.
`public string Text`	A segítségek ablakban a mintát reprezentáló szöveg.

Függvényei:

`public bool ApplyPattern(SudokuTable t, out HistoryEntry historyEntry)`	Minta alkalmazása a `t` táblán.
`protected abstract string getReportEntry(SudokuTable t)`	Visszaadja a naplóban a mintát leíró `string`-et.
`public abstract void PaintPattern(SudokuTable t)`	Kirajzolja a mintát `t` táblára.
`public bool Equals(Pattern p)`	Összehasonlítja a mintát `p`-vel.
`public bool IsValid()`	Megnézi, hogy a minta változtatásait időközben egy másik minta elvégezte-e.

5.1.3 Rózsa algoritmus:

A Rózsa algoritmus egy sorrendet definiál a speciális algoritmusok közt. Először mindig a legegyszerűbb, legkisebb műveletigénnyel rendelkező algoritmusok kapják meg a vezérlést – az `algorithms[]` tömbben legkisebb indexű elemek –, a bonyolultabb technikákat pedig csak akkor alkalmazzuk, ha egyébként nem tudtuk megoldani a táblát.

Amikor egy algoritmus először kezd el mintákat keresni, módszeresen bejárja az egész táblát, és a találatokat azonnal alkalmazza. Eközben jelölteket fog kizárni a táblából. Előfordulhat, hogy ennek következtében olyan minta fog keletkezni, amit első bejárás során nem talál meg. Nem volna túl hatékony emiatt az egész táblát újból bejárni, sokkal jobb megoldás lehet, ha később csak a tábla azon részeit ellenőrizzük, ahol a kihúzások miatt keletkezhetett minta. Erre szolgál a `TestCell` metódus, amelynek pontosan meg tudjuk adni, hol kell mintát keresni, a találatokat a `findings` adatszerkezet megfelelő listájába helyezzük. A módszeres bejárás után elég tehát ezt a jelölt sort vizsgálni – itt listával van reprezentálva. Ez abból áll, hogy kivesszük az első elemet, alkalmazzuk, és a közben esetleg keletkező új mintákat a lista végére tesszük, addig ismételve amíg a lista ki nem ürül.

A nagyobb hatékonyság érdekében minden jelölt kihúzásakor ellenőrizzük, hogy keletkezett-e új REJTETT vagy TISZTA cella. Ha igen, az aktuálisan futó algoritmus befejezésekor rögtön visszaugrunk a tömb elejére, nem bajlódunk a bonyolultabb technikákkal, amelyektől legfeljebb néhány jelölt kizárását remélhetjük.

A Rózsa algoritmus akkor ér véget, ha kitöltöttük a táblát, ellentmondásos táblához jutottunk, vagy úgy értünk végig az algoritmusokon, hogy nem találtunk semmit. Abban az esetben, amikor úgy jutottunk hibához, hogy nem alkalmaztunk tippelést, kilépünk, mert a feladvány eleve hibás. Ha már tippeltünk, vagy kevés az információ a megoldáshoz, utasítjuk az éppen bekapcsolt visszalépéses algoritmust (Nishio vagy Crook esetében), hogy módosítsa a táblát – írjon be új jelöltet, vagy olvassa vissza a rossz tipp előtti táblát, és írjon be egy jelöltet. Miután ez megtörtént, visszaadjuk a vezérlést a Rózsa algoritmusainak, hogy próbálkozzanak újból.

```
Do
{
```

```
      foundPattern = true;
      while (!error && foundPattern && table.DataInTable != table.M *
table.M)
      {
            for (int i = 0; i < algorithms.Length && !error; i++)
            {
                  //Az egyszerű algoritmusok segítségével próbáljuk
megoldani a rejtvényt...
            }
      }

      //Ha az egyszeru algoritmusok sikerrel jartak, kilepunk a
ciklusbol
      if (table.DataInTable != table.M * table.M)
      {
            switch (advancedAlgorithm)
            {
                  case AdvancedAlgorithms.Nishio:
                        //Tipp kérése a Nishio algoritmustól...
                        break;
                  case AdvancedAlgorithms.Crook:
                        //Tipp kérése a Crook algoritmustól...
                        break;
                  case AdvancedAlgorithms.DancingLinks:
                        //Megoldás Dancing links algoritmussal...
                        break;
                  case AdvancedAlgorithms.Off:
                        //kilépünk
                        solved = true;
                        break;
            }
      }
      else
      {
            solved = true;
      }
} while (!solved);
```

5.1.4 Hibák felismerése:

A program kétféle hibát képes felismerni:

- Amikor egy cellából az összes jelöltet kizártuk, `NakedErrorPattern` fog keletkezni, amit a TISZTA algoritmus fog megtalálni, ha olyan cellára hívtuk meg, amelyben a jelöltek száma nulla.

- Ha egy házban az egyik jelölt nem szerepel adatként, és az összes cellából ki lett zárva, `HiddenErrorPattern` fog keletkezni, amit a rejtett cella algoritmus érzékel,

ha olyan házra hívtuk meg, ahol valamely érték nem fordul elő sem adatként, sem jelöltként.

5.2 Nem visszalépéses algoritmusok

5.2.1 Tiszta Cella: (NakedSingle.cs)

Módszeres bejárásnál két ciklus segítségével minden cellára sorfolytonosan meghívjuk a `TestCell` metódust, ami ha a cellában megengedett jelöltek száma egyenlő eggyel, létrehoz egy `NakedSinglePattern` mintát.

```
if ((t.Cells[redCell.i, redCell.j].CandidatesNumber == 1) && !
t.Cells[redCell.i, redCell.j].IsData()) //ha csak egy jelölt van
megengedve
{
      //megkeressük, melyik az a jelölt
      for (int k = 1; k <= t.M; ++k)
      {
            if (t.Cells[redCell.i, redCell.j].IsCandidate(k)) redCand =
k;
      }

      //létrehozzuk a mintát...
}
```

5.2.2 Rejtett cella: (HiddenSingle.cs)

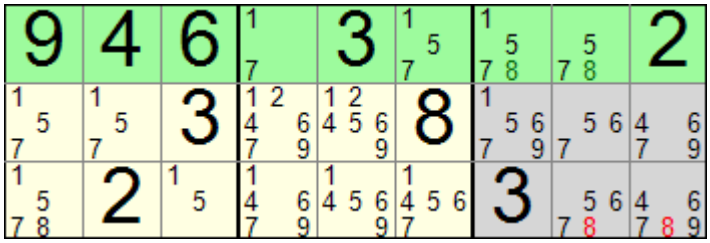
Módszeres bejárásnál ciklussal végigmegyünk először a sorokon, majd oszlopokon és blokkokon – továbbiakban mindig ezt a bejárásmódot használjuk, amennyiben nincs külön kiemelve, hogy nem. Minden ház minden jelöltjére megnézzük, hogy egyedüliként fordulnak-e elő házaikban, ha igen és most találtuk meg először, akkor feldolgozzuk a mintát. Azt, hogy egy jelöltet, illetve cellát rejtett cellaként megtaláltunk-e, egy `patterns[,]` nevű tömbben tároljuk, itt igaz érték szerepel a megfelelő indexű helyen, ha ott a táblában már találtunk egy rejtett cellát. Ha nem jegyeznénk a találatokat, előfordulhatna, hogy mivel egy cella több házban is benne van, ugyanazt a mintát többször is megtaláljuk.

```
bool found = false;
for (int j = 0; (j < t.M) && (sum < 2); j++)
{
      if (t.Cells[i, j].Data == c)
      {
            found = true;
      }
      if (t.Cells[i, j].IsCandidate(c) && !t.Cells[i, j].IsData())
      {
            sum++;
            col = j;
      }
}

//Ha ilyen jelölt nincsen beírva adatként a házba, és jelöltként nem
fordul elő, akkor hibás a tábla...
if (sum == 0 && !found)
{

}

//Bejárás esetén nem szabad figyelembe venni, hogy egyszer már
megtaláltuk, mert egyébként a jelölt kihúzásakor keletkező mintákat csak
a jelöltsor feldolgozásakor dolgoznánk fel.
if (sum == 1 && (patterns[i, col] == false ||
AlgorithmManager.Instance.SolveState == SolveStates.RozsaSolve))
{
      //Létrehozzuk a mintát...

      patterns[i, col] = true; //
}
```

5.2.3 **Tégla**: (Intersection.cs)

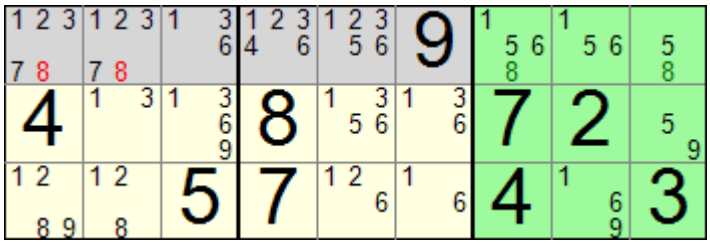
Az algoritmusnak tudnia kell, hogy a téglákban, az adott jelöltből hány van. Ezt két tömbben tartjuk nyilván, külön a sorbeli (`row_bricks[M,M,N]`) és külön az oszlopbeli (`col_bricks[M,M,N]`) téglákra, ahol az első index a jelölt értéke, a másik kettő pedig a tégla helyét határozza meg a táblában. Futtatás során ezeket a tömböket járjuk be, ha egy téglában azt találjuk, hogy egy adott jelöltből legalább kettő van, segédfüggvényekkel tovább vizsgáljuk:

```
private void TestRowBrick(int i, int j, int c)
```



Ciklussal végigmegyünk a vizsgált sor tégláin. Ha úgy találjuk, hogy csak az egyikben fordul elő `c` jelölt, akkor blokkból húzunk ki.

```
private void TestRowBoxBrick(int i, int j, int c)
```



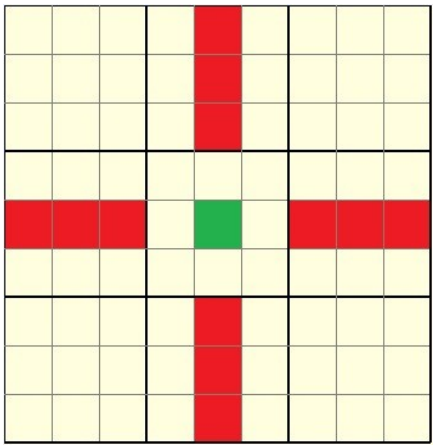
Ciklussal végigmegyünk a vizsgált blokk tégláin. Ha úgy találjuk, hogy csak az egyikben fordul elő `c` jelölt, akkor sorból húzunk ki.

Oszlopokra teljesen azonos módon járunk el.

A `testCell` metódus:

Jelölt kihúzásánál frissítsük a `row_bricks` és `col_bricks` tömböket, és szokás szerint újonnan keletkezett mintát keresünk.

Ha egy jelöltet kihúzunk, 8 helyen keletkezhet új tégla (például, ha a zöld cellából húzunk, az ábrán jelölt piros téglákban két módon mindegyikben), elég tehát ezeket ellenőrizni a fenti segédfüggvényekkel.



5.2.4 Tiszta cellacsoport: (NakedPair.cs, NakedTriple.cs, NakedQuad.cs)

Bejárjuk a táblát a szokásos módon, és minden házban sorban megnézzük, van-e TISZTA *n*-es minta, ehhez a szóba jöhető cellákat kigyűjtjük egy listába. Szóba jövő celláin olyan cellát értünk, ahol a jelöltek számossága egy TISZTA *n*-es algoritmus esetén legalább 2, legfeljebb *n*. A szóba jövő cellák közül az összes lehetséges módon kiválasztunk *n*-et, és megvizsgáljuk, hogy a jelölthalmazaik uniója hány elemű. Ha pontosan *n*, akkor TISZTA cella *n*-est találtunk, azaz a ház többi cellájából az olyan elemeket törölhetjük, amelyek szerepelnek a cellacsoportban.

```
//A szóbajöhető cellák feldolgozása tiszta pár esetén.
for (int p1 = 0; p1 < cells.Count; ++p1)
{
      for (int p2 = p1 + 1; p2 < cells.Count; ++p2)
      {
            int sum = 0; //A jelölthalmazok uniójának számossága.
            bool[] test = new bool[t.M + 1]; //A mintában résztvevő
jelöltek.

            //Vesszük jelölthalmazok unióját.
            for (int i = 1; i <= t.M; ++i)
            {
                  if (t.Cells[cells[p1].i, cells[p1].j].IsCandidate(i)
|| t.Cells[cells[p2].i, cells[p2].j].IsCandidate(i))
                  {
                        test[i] = true;
                        ++sum;
```

```
                    }
                }

                if (sum == 2) //Ha két elemű mintát találtunk
                {
                        CreatePatternBlock(test, cells[p1].i, cells[p1].j,
cells[p2].i, cells[p2].j);
                }
        }
}
```

Megjegyzés: Ennek a megoldásnak előnye, hogy a szóba jövő jelöltek kigyűjtését leszámítva azonosan működik minden háztípusra.

5.2.5 Rejtett cellacsoport: (HiddenPair.cs, HiddenTriple.cs, HiddenQuad.cs)

Bejárjuk a táblát a szokásos módon, és minden házban sorban megnézzük, van-e rejtett *n*-es minta. Ehhez először kigyűjtjük egy tömbbe a jelöltek házon belüli előfordulásait, valamint, hogy hány van belőlük. Továbbiakban azokat a jelölteket vizsgáljuk, amelyeknek egy rejtett cella *n*-es algoritmus esetében legalább 2, legfeljebb *n* előfordulásuk van az adott házban. Az összes lehetséges módon kiválasztunk ezen jelöltek közül *n*-et és megnézzük, hány cellában jelennek meg. Ha pontosan *n* darabban, akkor mintát találtunk, tehát, ha tudunk kihúzni jelöltet segítségével, akkor feldolgozzuk.

```
//Megfelelő számú előfordulás van-e?
if (candidate_numbers[c1] >= 2 && candidate_numbers[c1] <= 3 &&
candidate_numbers[c2] >= 2 && candidate_numbers[c2] <= 3 &&
candidate_numbers[c3] >= 2 && candidate_numbers[c3] <= 3)
{
      //Az előfordulások celláinak kigyűjtése
      IntPair[] cells = new IntPair[3];
      int count = 0;
      for (int k = 0; k < t.M; ++k)
      {
            if (candidate_positions[c1, k] == true ||
candidate_positions[c2, k] == true || candidate_positions[c3, k] ==
true)
            {
                  if (count < 3)
                  {
                        cells[count] = new IntPair(k, col);
                  }
                  ++count;
            }
```

```
        }

        //Ha három cellában vannak
        if (count == 3)
        {
                //Vannak-e kizárható jelöltek?
                int coun = 0;
                for (int n = 1; n <= t.M; ++n)
                {
                        if (t.Cells[cells[0].i, cells[0].j].IsCandidate(n) ||
t.Cells[cells[1].i, cells[1].j].IsCandidate(n) || t.Cells[cells[2].i,
cells[2].j].IsCandidate(n))
                        {
                                ++coun;
                        }
                }

                if (coun > 3)
                {
                        //Ha igen, minta feldolgozása...
                }
        }
}
```

5.3 Visszalépéses algoritmusok

5.3.1 **Nishio**: (Nishio.cs)

A Nishio algoritmus akkor lép működésbe, ha a fentebb leírt algoritmusok nem tudták megoldani a táblát:

```
he = nishio.Guess();
if (he == null) return;
sudokuHistory.AddEntry(he, false);
```

A `Guess()` metódus módosítja a munkatáblát és visszatér az ehhez tartozó napló bejegyzéssel. Ha a bejegyzés `null`-al egyenlő, akkor a Nishio elakadt, egyébként hozzáadjuk a naplóhoz, majd az új információval megpróbáljuk megoldani a táblát.

A `Guess` metódus:

A metódus első hívásakor beállítjuk az algoritmus kezdő értékeit: Az aktuális tábláról csináljunk egy másolatot (`backup`), a jelöltjeit pedig pakoljuk bele egy sorba – a

cellákat jelöltszám szerint növekvő sorrendben dolgozzuk fel, ettől azt várjuk, hogy hatékonyabb lesz algoritmus. A későbbiekben ha a metódus meghívásra kerül, visszaállítjuk a táblát a sikertelen tippelés előtti állapotba, és a sorból kivesszük a következő elemet, a táblába írjuk, valamint elkészítjük hozzá a napló bejegyzést.

5.3.2 **Crook**: (Crook.cs)

A J. F. Crook által leírt algoritmus **[7-Crook]** megvalósítása. Nagyon hasonlít a Nishio-ra, különbség, hogy most tetszőleges mélységű visszalépést megengedünk. Itt egy `Guess(error)` metódust kell meghívni tipp lekérdezéséhez, ahol `error` egy logikai változó, értéke igaz, ha ellentmondással akadtunk el a megoldásban, hamis egyébként.

Crook algoritmus esetében a soron következő tippeket egy fában tároljuk, illetve mivel egy mélységi keresést valósítunk meg, elég egy utat nyilván tartani. Minden mélységben a lehetséges lépéseket (tippeket) egy, a Nishio-éhoz hasonló sorban tároljuk, szerencsére azonban itt nem kell az egész tábla összes jelöltjét végigpróbálnunk, elég ha kiválasztunk egy tetszőleges – legkisebb elemszámú – cellát, és az ebben lévő jelölteket rakjuk a sorba. A különböző szintekhez tartozó jelöltsorokat pedig egy verembe pakoljuk. Hasonlóan a szintekhez tartozó elmentett táblákat is egy veremben tároljuk.

Ha a `Guess` metódust úgy hívjuk meg, hogy error értéke hamis, akkor eggyel mélyebbre lépünk a keresőfában, vagyis egy új szintet hozunk létre. Ha error értéke igaz, és a legfelső szinten lévő sorban míg van jelölt, akkor visszaolvassuk a verem tetején lévő táblát, és beleírjuk a legfelső jelölt sor egy elemét. Ha error értéke igaz és kiürült a verem tetején lévő jelölt sor, az azt jelenti, hogy zsákutcába jutottunk a kereséssel. Ekkor a vermek tetejéről addig dobáljuk el az elemeket, míg egy olyan jelölt sorhoz jutunk, amelyben még van teszteletlen jelölt. Ekkor visszaolvassuk a szinthez tartozó táblát és beírunk egy jelöltet.

A Crook algoritmus azonban nem csak akkor aktív, amikor a Rózsa sikertelensége miatt meghívjuk. Ha a felhasználó önállóan oldja a rejtvényt, és olyan adatot ír a cellába, ami nem következik sem a TISZTA sem pedig REJTETT algoritmusból, akkor azt úgy

vesszük, mintha a Crook algoritmus alapján dolgozott volna. Ilyen jelölt beírása megváltoztatja az algoritmus állapotát is, ugyanis a Crook keresőfájában mindig nyilván tartjuk, hogy éppen hol tartunk. A nyomkövetés előnye, hogy bármikor, ilyenkor is értelmes naplót tudunk generálni. Ennek persze előfeltétele az is, hogy a felhasználó módszeresen haladjon, mélységi bejárással, egyébként a készített napló is ezt az össze-vissza tippelgetést fogja átláthatatlan formában visszaadni.

A naplózás hátránya, hogy bizonyos esetekben a táblán történt változtatások visszavonását csak nagyon bonyolultan lehetne megvalósítani. Ha az utolsó naplóbejegyzést a Crook algoritmus készítette, akkor visszaállításkor a Crook vermeit is vissza kellene állítani, valamint egy teljesen más munkatáblát kellene visszatölteni a `AlgorithmManager` osztályba, amit ráadásul nem is tároltunk. Ezért minden olyan esetben, amikor ez problémát jelenthet, a visszavonás művelet le van tiltva.

```
//ha nem volt hiba, akkor egyel mélyebb szintre lépünk
if (!error) newLevel();

//ha hiba és nulla mélységben vagyunk, akkor a rejtvény nem megoldható
if (error && guessOptions.Count == 0) return null;

//ha az adott szinten az összes lehetőséget végigpróbáltuk akkor feljebb
kell lépni
while (guessOptions.Peek().Count == 0)
{
      guessOptions.Pop();
      backupTables.Pop();
      --depth;
}

//a munkatábla visszaállítása
AlgorithmManager.Instance.Table = backupTables.Peek().Clone() as
SudokuTable;

//a tippelés
GuessOption currentTipp = guessOptions.Peek()[0];
guessOptions.Peek().RemoveAt(0);

//a naplóbejegyzés elkészítése...
```

5.3.3 Dancing Links [9-Balázs2007]: (DancingLinks.cs)

Donald E. Knuth által leírt fedési problémát megoldó Dancing links algoritmus **[4-Knuth2000]** megvalósítása. A sudoku feladványt először átalakítjuk az **[2-Exact2007]**

oldalon található módszer szerint fedési problémára.

Felhasznált adatszerkezet: négy irányba láncolt lista.

Egy csúcs felépítése:

```
class Node
{
	public Node left;
	public Node right;
	public Node up;
	public Node down;
	public Header head;
	public int row;
	public Node() : this(-1) { }
	public Node(int row)
	{
		left = this;
		right = this;
		up = this;
		down = this;
		this.row = row;
		head = null;
	}
}
```

Ezekből a csúcsokból épül fel a hiányos mátrix, melyben az oszlopok nyilvántartják, hogy hány

elemük van. Erre egy, a Node osztályból származtatott Header osztály szolgál.

```
class Header : Node
{
	public int size;
	public int name;
	public Header(int name)
	{
		this.name = name;
		this.size = 0;
		this.head = this;
	}
}
```

Az algoritmus lépései:

BuildTable(): A hiányos mátrix keretének felépítése: oszlopokra, illetve sorokra mutató Header-ök és Node-ok.

SetStartingGrid(): A sudoku tábla adatai szerint feltölti a hiányos mátrixot (a jelölteket nem

veszi figyelembe) **[2-Exact2007]**.

A sudoku négy egyszerű szabálya szerint alakítjuk át a rejtvényt fedési problémára:

- Minden mező egy adatot tartalmazhat.
- Minden sorban minden szám csak egyszer fordulhat elő.
- Minden oszlopban minden szám csak egyszer fordulhat el.
- Minden blokkban minden szám csak egyszer fordulhat elő.

Minden szabályhoz egy M*M*M sorból (sorok*oszlopok*lehetséges értékek) és M*M oszlopból (mezők száma) álló mátrix feleltethető meg. A négy szabály így egy nagy M^3*(4*M^2)-es mátrixot határoz meg.

Egy 9×9-es sudokuban ez az átalakítás egy 9^3*(4*9^2)-es mátrixot jelent. Egy ilyen mátrix megtalálható ezen a helyen:

http://www.stolaf.edu/people/hansonr/sudoku/exactcovermatrix.htm

Solve(): Megoldja a feladatot, és közben számolja, hogy hány megoldást talált. Ha egynél több van, akkor leáll.

WriteBackToTable(): Ha legalább egy találat van, visszaírja az adatokat a sudoku táblába.

6. Rejtvények generálása

SudokuGenerator [9-Balázs2007]: (SudokuGenerator.cs)

A Dancing Links algoritmust felhasználva egyértelműen megoldható rejtvényt generál. Ehhez véletlenszerűen hozzávesz egy elemet a táblához az AddNewRandomData() függvénnyel, és ellenőriz a Dancing Links algoritmussal:

- Ha a tábla egyértelműen megoldható akkor megpróbálja redukálni a Reduce() függvény

segítségével.

- Ha a tábla ellentmondásos, eltávolít belőle egy elemet.
- Ha a táblának több megoldása van, akkor hozzávesz egy újabb elemet és újra ellenőriz.

Az algoritmus befejeződik, ha:

- Egyértelműen megoldható rejtvényt talál,
- vagy elér egy kritikus próbálkozásszámot,
- vagy a felhasználó leállítja.

Függvények:

`Reduce`():

Végigmegy a tábla elemein. Az elemeket egyesével elhagyja, miközben az így kialakult táblát ellenőrzi:

- Ha a tábla továbbra is egyértelműen megoldható, akkor veszi a következő elemet.
- Egyébként visszarakja a táblába a választott elemet, és csak ezután megy tovább a következő elemre.

`AddNewRandomData`():

Hozzávesz egy elemet a táblához, és a szomszédaiból kihúzza az adott jelöltet. Ha eközben valamelyik szomszédban a jelöltek száma 0-ra ér, akkor törli az elemet a választott mezőből, és a szomszédokban visszaállítja a jelölteket, majd új elemet választ.

Megjegyzés:

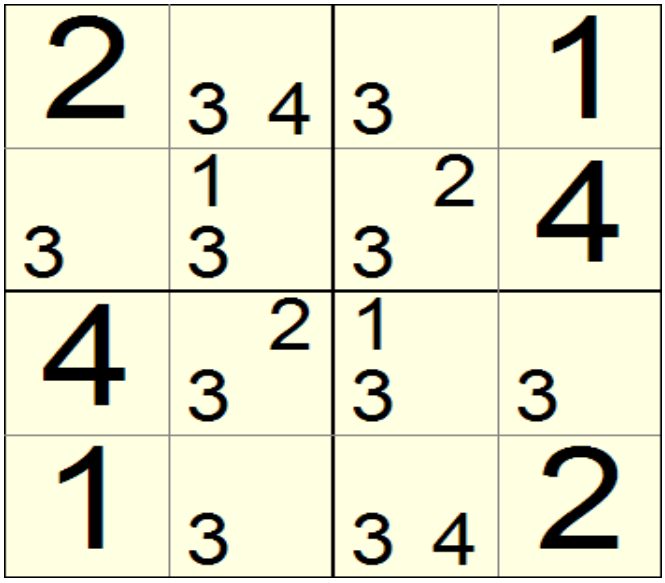
Bár az algoritmus univerzális, a program mégsem támogatja a 25×25-ös rejtvények megoldását, mivel ezzel a módszerrel nagyon sokáig tartana.

7. Tesztelés

A program tesztelését az előállított naplók helyességének ellenőrzésével, illetve más programok, például SudokuSolver **[10-Novák2010]** naplóinak összehasonlításával végeztem.

7.1 Naplókészítés

4×4-es tábla [www.menneske.no 3114. feladvány]:



Automatikus rejtvénymegoldáskor generált napló:

A0 algoritmus (adat):
1. lépés: r1c1=2 (adat)
2. lépés: r1c4=1 (adat)
3. lépés: r2c4=4 (adat)
4. lépés: r3c1=4 (adat)
5. lépés: r4c1=1 (adat)
6. lépés: r4c4=2 (adat)

	c1	c2	c3	c4
r1	2	34	3	1
r2	3	13	23	4
r3	4	23	13	3
r4	1	3	34	2

A1 algoritmus (Tiszta):
7. lépés: r1c3=3 (tiszta)
-> r1c2=4 (tiszta)
-> r2c3=2 (tiszta)
-> r3c3=1 (tiszta)

Felhasználó lépéseinek naplózása:

A0 algoritmus (adat):
1. lépés: r1c1=2 (adat)
2. lépés: r1c4=1 (adat)
3. lépés: r2c4=4 (adat)
4. lépés: r3c1=4 (adat)
5. lépés: r4c1=1 (adat)
6. lépés: r4c4=2 (adat)

	c1	c2	c3	c4
r1	2	34	3	1
r2	3	13	23	4
r3	4	23	13	3
r4	1	3	34	2

A1 algoritmus (Tiszta):
7. lépés: r1c3=3 (tiszta)
-> r2c3=2 (tiszta)
-> r3c3=1 (tiszta)
-> r4c3=4 (tiszta)
-> r1c2=4 (tiszta)

-> r4c3=4 (tiszta)
8. lépés: r2c1=3 (tiszta)
-> r2c2=1 (tiszta)
9. lépés: r2c2=1 (tiszta)
10. lépés: r2c3=2 (tiszta)
11. lépés: r3c3=1 (tiszta)
12. lépés: r3c4=3 (tiszta)
-> r3c2=2 (tiszta)
13. lépés: r4c2=3 (tiszta)
14. lépés: r4c3=4 (tiszta)
15. lépés: r1c2=4 (tiszta)
16. lépés: r3c2=2 (tiszta)

A megoldás:

	c1	c2	c3	c4
r1	2	4	3	1
r2	3	1	2	4
r3	4	2	1	3
r4	1	3	4	2

A rejtvény típusa: 2A1-6

8. lépés: r2c1=3 (tiszta)
-> r2c2=1 (tiszta)
9. lépés: r2c2=1 (tiszta)
10. lépés: r2c3=2 (tiszta)
11. lépés: r3c3=1 (tiszta)
12. lépés: r3c4=3 (tiszta)
-> r3c2=2 (tiszta)
13. lépés: r4c2=3 (tiszta)
14. lépés: r4c3=4 (tiszta)
15. lépés: r1c2=4 (tiszta)
16. lépés: r3c2=2 (tiszta)

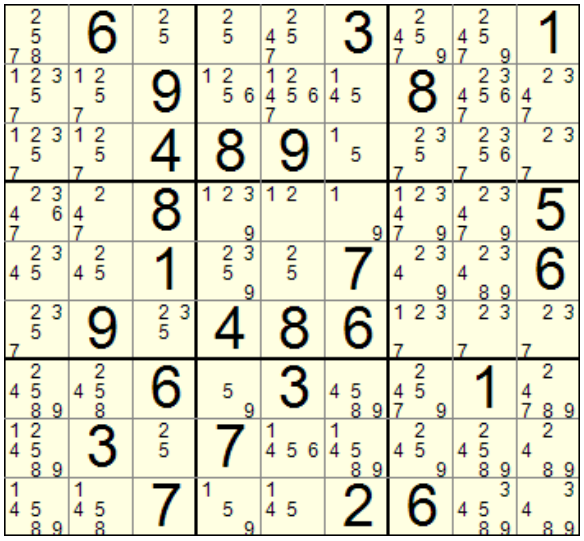
A megoldás:

	c1	c2	c3	c4
r1	2	4	3	1
r2	3	1	2	4
r3	4	2	1	3
r4	1	3	4	2

A rejtvény típusa: 2A1-6

9×9-es tábla [www.menneske.no 5673929. feladvány]:

A0 algoritmus (adat):
1. lépés: r1c8=1 (adat)
2. lépés: r2c4=5 (adat)
3. lépés: r2c6=3 (adat)
4. lépés: r2c7=9 (adat)
5. lépés: r2c8=4 (adat)
6. lépés: r2c9=8 (adat)
7. lépés: r3c4=8 (adat)
8. lépés: r3c9=3 (adat)
9. lépés: r4c7=3 (adat)
10. lépés: r5c1=2 (adat)
11. lépés: r5c2=9 (adat)
12. lépés: r5c4=3 (adat)
13. lépés: r5c6=4 (adat)
14. lépés: r6c2=6 (adat)
15. lépés: r6c5=1 (adat)
16. lépés: r7c2=5 (adat)
17. lépés: r7c7=1 (adat)
18. lépés: r8c2=4 (adat)
19. lépés: r8c3=9 (adat)
20. lépés: r8c4=2 (adat)
21. lépés: r8c5=8 (adat)
22. lépés: r8c9=5 (adat)
23. lépés: r9c3=8 (adat)



24. lépés: r9c8=7 (adat)

A1 algoritmus (Tiszta):
25. lépés: r8c7=6 (tiszta)
-> r8c8=3 (tiszta)
26. lépés: r8c8=3 (tiszta)

A2 algoritmus (Rejtett):
27. lépés: r7c8=8 (rejtett a sorban)
-> r5c7=8 (rejtett a sorban)
28. lépés: r9c4=1 (rejtett az oszlopban)
-> r8c1=1 (rejtett a blokkban)
-> r8c6=7 (tiszta)
-> r8c1=1 (rejtett a sorban)
-> r3c6=1 (rejtett az oszlopban)
29. lépés: r3c6=1 (rejtett az oszlopban)
30. lépés: r8c1=1 (rejtett a blokkban)
-> r8c6=7 (rejtett a sorban)
31. lépés: r5c7=8 (rejtett a sorban)
32. lépés: r8c6=7 (rejtett a sorban)

A3 algoritmus (Tégla):
33-1. allépés: c8 oszlopban csak a c8t2 téglában van a(z) 9 megengedve, ezért a blokk többi cellájából a(z) 9 jelölt kihúzható:
r4c9!=9; r6c9!=9

A6 algoritmus (Tiszta hármas):
33-2. allépés: c1 oszlopban r2c1, r7c1, r9c1 cellákban csak 3 6 7 jelöltek fordulnak elõ, ezért az oszlop többi cellájából kihúzhatók:
r1c1!=3; r1c1!=6; r1c1!=7; r3c1!=6; r3c1!=7; r4c1!=7; r6c1!=3; r6c1!=7
-> r6c3=3 (rejtett a sorban)

A2 algoritmus (Rejtett):
33. lépés: r6c3=3 (rejtett a sorban)
-> r1c2=3 (rejtett a sorban)
34. lépés: r1c2=3 (rejtett a sorban)
-> r1c1=8 (rejtett a sorban)
-> r4c2=8 (rejtett az oszlopban)
-> r9c2=2 (tiszta)
35. lépés: r1c1=8 (rejtett a sorban)
-> r3c1=9 (rejtett az oszlopban)
-> r6c6=8 (rejtett a sorban)
36. lépés: r4c2=8 (rejtett az oszlopban)
-> r2c2=1 (rejtett az oszlopban)
37. lépés: r3c1=9 (rejtett az oszlopban)
38. lépés: r6c6=8 (rejtett a sorban)
39. lépés: r2c2=1 (rejtett az oszlopban)
-> r3c2=7 (rejtett az oszlopban)
40. lépés: r3c2=7 (rejtett az oszlopban)
-> r9c2=2 (rejtett az oszlopban)
-> r2c1=6 (tiszta)
-> r7c1=7 (rejtett az oszlopban)
-> r2c5=7 (rejtett a sorban)

41. lépés: r9c2=2 (rejtett az oszlopban)
-> r9c7=4 (tiszta)
-> r7c9=2 (rejtett a blokkban)
42. lépés: r7c1=7 (rejtett az oszlopban)
-> r7c5=3 (rejtett a sorban)
-> r9c1=3 (rejtett az oszlopban)
-> r7c3=6 (tiszta)
43. lépés: r2c5=7 (rejtett a sorban)
-> r2c3=2 (rejtett a sorban)
44. lépés: r7c9=2 (rejtett a blokkban)
-> r9c9=9 (rejtett az oszlopban)
45. lépés: r7c5=3 (rejtett a sorban)
-> r7c4=4 (rejtett a sorban)
46. lépés: r9c1=3 (rejtett az oszlopban)
-> r2c1=6 (rejtett az oszlopban)
-> r7c3=6 (rejtett a blokkban)
47. lépés: r2c3=2 (rejtett a sorban)
48. lépés: r9c9=9 (rejtett az oszlopban)
-> r9c7=4 (rejtett a blokkban)
49. lépés: r7c4=4 (rejtett a sorban)
-> r7c6=9 (rejtett a sorban)
50. lépés: r2c1=6 (rejtett az oszlopban)
51. lépés: r7c3=6 (rejtett a blokkban)
-> r7c6=9 (tiszta)
52. lépés: r9c7=4 (rejtett a blokkban)
53. lépés: r7c6=9 (rejtett a sorban)

A3 algoritmus (Tégla):
54-1. allépés: r4 sorban csak a r4t2 téglában van a(z) 2 megengedve, ezért a blokk többi cellájából a(z) 2 jelölt kihúzható:
r4c8!=2
54-2. allépés: b1 blokkban csak a c3t1 téglában van a(z) 4 megengedve, ezért az oszlop többi cellájából a(z) 4 jelölt kihúzható:
r4c3!=4
54-3. allépés: b1 blokkban csak a c3t1 téglában van a(z) 5 megengedve, ezért az oszlop többi cellájából a(z) 5 jelölt kihúzható:
r4c3!=5; r5c3!=5

A4 algoritmus (Tiszta pár):
54-4. allépés: r5 sorban r5c5, r5c8 cellák tiszta cellapárt alkotnak 5, 6 jelöltekkel, ezért a sor többi cellájából kihúzhatók:
r5c9!=6
54-5. allépés: c5 oszlopban r5c5, r9c5 cellák tiszta cellapárt alkotnak 5, 6 jelöltekkel, ezért az oszlop többi cellájából kihúzhatók:
r1c5!=6; r3c5!=6; r4c5!=5; r4c5!=6
-> r3c8=6 (rejtett a sorban)

A2 algoritmus (Rejtett):
54. lépés: r3c8=6 (rejtett a sorban)
-> r6c8=2 (rejtett az oszlopban)
-> r5c8=5 (tiszta)
-> r5c5=6 (rejtett a sorban)
-> r4c9=6 (rejtett a blokkban)

-> r1c9=7 (tiszta)
55. lépés: r6c8=2 (rejtett az oszlopban)
-> r6c4=9 (rejtett a sorban)
-> r4c8=9 (rejtett az oszlopban)
56. lépés: r5c5=6 (rejtett a sorban)
-> r5c8=5 (rejtett a sorban)
-> r9c5=5 (rejtett az oszlopban)
-> r4c6=5 (rejtett a blokkban)
-> r9c5=5 (tiszta)
-> r9c6=6 (rejtett a sorban)
-> r1c4=6 (rejtett az oszlopban)
57. lépés: r4c9=6 (rejtett a blokkban)
-> r4c3=1 (rejtett a sorban)
-> r5c9=1 (rejtett az oszlopban)
-> r4c1=4 (rejtett a sorban)
-> r6c9=4 (rejtett az oszlopban)
58. lépés: r6c4=9 (rejtett a sorban)
-> r4c4=7 (rejtett az oszlopban)
-> r1c4=6 (tiszta)
-> r1c5=9 (rejtett a sorban)
-> r4c4=7 (tiszta)
-> r4c5=2 (tiszta)
59. lépés: r4c8=9 (rejtett az oszlopban)
60. lépés: r5c8=5 (rejtett a sorban)
-> r6c7=7 (tiszta)
-> r6c1=5 (rejtett a sorban)
61. lépés: r9c5=5 (rejtett az oszlopban)
-> r9c6=6 (tiszta)
62. lépés: r4c6=5 (rejtett a blokkban)
-> r4c5=2 (rejtett a sorban)
-> r1c6=2 (rejtett az oszlopban)
-> r4c1=4 (tiszta)
63. lépés: r9c6=6 (rejtett a sorban)
-> r1c6=2 (tiszta)
64. lépés: r1c4=6 (rejtett az oszlopban)
65. lépés: r4c3=1 (rejtett a sorban)
-> r5c3=7 (rejtett az oszlopban)
-> r5c3=7 (tiszta)
66. lépés: r5c9=1 (rejtett az oszlopban)
67. lépés: r4c1=4 (rejtett a sorban)
-> r6c1=5 (tiszta)
68. lépés: r6c9=4 (rejtett az oszlopban)
-> r6c7=7 (rejtett a sorban)
-> r1c9=7 (rejtett az oszlopban)
69. lépés: r4c4=7 (rejtett az oszlopban)
70. lépés: r1c5=9 (rejtett a sorban)
-> r1c3=4 (rejtett a sorban)
-> r3c5=4 (rejtett az oszlopban)
71. lépés: r6c1=5 (rejtett a sorban)
72. lépés: r4c5=2 (rejtett a sorban)
-> r3c5=4 (tiszta)
-> r3c7=2 (rejtett a sorban)
73. lépés: r1c6=2 (rejtett az oszlopban)

74. lépés: r5c3=7 (rejtett az oszlopban)
75. lépés: r6c7=7 (rejtett a sorban)
-> r1c7=5 (tiszta)
76. lépés: r1c9=7 (rejtett az oszlopban)
77. lépés: r1c3=4 (rejtett a sorban)
-> r1c7=5 (rejtett a sorban)
-> r3c3=5 (rejtett az oszlopban)
-> r3c3=5 (tiszta)
78. lépés: r3c5=4 (rejtett az oszlopban)
79. lépés: r3c7=2 (rejtett a sorban)
80. lépés: r1c7=5 (rejtett a sorban)
81. lépés: r3c3=5 (rejtett az oszlopban)

A megoldás:

834692517
612573948
975841263
481725396
297364851
563918724
756439182
149287635
328156479

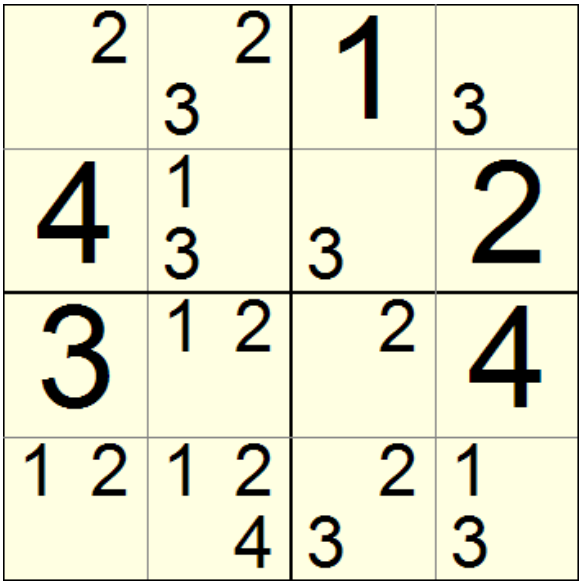
A rejtvény típusa: 3A6-24

Hibás tábla:

A0 algoritmus (adat):
1. lépés: r1c3=1 (Adat)
2. lépés: r2c1=4 (Adat)
3. lépés: r2c4=2 (Adat)
4. lépés: r3c1=3 (Adat)
5. lépés: r3c4=4 (Adat)

	c1	c2	c3	c4
r1	2	23	1	3
r2	4	13	3	2
r3	3	12	2	4
r4	12	124	23	13

A1 algoritmus (Tiszta):
6. lépés: r1c1=2 (tiszta)
-> r1c2=3 (tiszta)
-> r4c1=1 (tiszta)
8. lépés: r1c2=3 (tiszta)
-> Hiba: r1c4 <-- 3 kihúzásakor
-> r2c2=1 (tiszta)
-> r2c3=3 (rejtett a sorban)



A rejtvény típusa: 2U0-0

7.2 A program összehasonlítása a SudokuSolver-rel

Az összehasonlítás Gordon F. Royle **[11-Royle]** ausztrál matematikus adatbázisán történt, amely mintegy 50 000 minimális – 17 adatot tartalmazó –, egyértelműen megoldható 9×9-es rejtvényt tartalmaz. A teszt során azt vizsgáltam, hogy a rejtvények típusát, a két program egyformának határozza-e meg – ennek rejtett pár algoritmusig van értelme, ugyanis a SudokuSolver az X2 algoritmust előbbre veszi, mint a tiszta és rejtett hármast így az A5 algoritmus felett nincs értelme az összehasonlításnak. Az eredmények:

	Sudoku v2.0	**SudokuSolver**
Tiszta cella:	0 db	0 db
Rejtett cella:	21905 db	21905 db
Tégla:	15468 db	15468 db
Tiszta pár:	2383 db	2383 db
Rejtett pár:	1832 db	1832 db
Tiszta hármas:	34 db	...
Rejtett hármas:	20 db	
Tiszta négyes:	3 db	
Rejtett négyes:	0 db	

8. Fejlesztési lehetőségek

- A program kiegészítése újabb algoritmusokkal: manapság több, mint 60 technika ismert Sudoku rejtvények megoldására. Ebből a programom 11-et ismer.
- Naplókészítés fejlesztése: a kihúzott jelöltek alapján is fel tudja ismerni az alkalmazott mintát.
- Rejtvény készítés fejlesztése: adott nehézségű feladványokat tudjon készíteni.

IV. Összegzés

Munkám eredményeként létrejött egy olyan sudoku megoldó program, ami az összes megoldható 4×4-es, 9×9-es, 16×16-os és 25×25-ös méretű feladványnak meg tudja találni egy megoldását, és segítséget nyújt a lépések megértéséhez. Az eredményhez vezető táblamódosításokat naplózza, így pontosan nyomon lehet követni a megoldás folyamatát, sőt ha a felhasználó maga oldja meg a rejtvényt, akkor is elkészül a napló, amit könnyedén össze lehet vetni a program által készítettel. Az alkalmazott lépések, illetve az megoldási technikák megértését azzal segíti, hogy adott állásnál fellelhető bármely mintát ki tudjuk rajzolni a táblára.

A szakdolgozatban bemutatott kutatást az ELTE TÁMOP-4.2.1.B-09-1-KMR-2010-0003 kódú pályázatának támogatásával végeztem.

V. Irodalomjegyzék

[1-Albert2004] Albert István: A .NET Framework és programozása, Szak Kiadó Kft., 2004, [868], ISBN 963 9131 62 8.

[2-Exact2009] Exact cover (Sudoku átalakítása fedési problémára). http://en.wikipedia.org/wiki/Exact_cover/ Letöltve: 2010. június 5.

[3-Iványi2009] Iványi Antal: A Rózsa program algoritmusai, 2009. http://compalg.inf.elte.hu/~tony/Oktatas/Rozsa/ Letöltve: 2010. június 5.

[4-Knuth2000] Donald Ervin Knuth: Dancing links, *Millenial Perspectives in Compute Science*, 2000, [187−214]. http://www-cs-faculty.stanford.edu/~knuth/preprints.html Letöltve: 2010. június 5.

[5-SudoCue] Ruud van der Werl: Sudoku file formats (.sdk, .sdx). http://www.sudocue.net/ Letöltve: 2010. június 5.

[6-Techniques] Ruud van der Werl: Solving techniques. http://www.sudopedia.org Letöltve: 2010. június 5.

[7-Crook] J. F. Crook: A Pencil-and-Paper Algorithm for Solving Sudoku Puzzles, *Notices of the American Mathematical Society*, 56. évfolyam 4. szám, 2009, [460−468].

[8-TheLogicofSudoku] Andrew C Stuart: The Logic of Sudoku, MM Publishing Limited, 2007, [244], ISBN 978 0 9554841 0 0.

[9-Balázs2007] Balázs Viktor: Nagyprogram ELTE IK, 2007.

[10-Novák2010] Novák Balázs: SudokuSolver, Diplomamunka ELTE IK, 2010.

[11-Royle] Gordon F. Royle: Minimum sudoku. http://units.maths.uwa.edu.au/~gordon/sudokumin.php Letöltve: 2010. június 5.