Nonogrami

Abramušić, Ivan

Undergraduate thesis / Završni rad

2023

Degree Grantor / Ustanova koja je dodijelila akademski / stručni stupanj: **Josip Juraj Strossmayer University of Osijek, Faculty of Electrical Engineering, Computer Science and Information Technology Osijek / Sveučilište Josipa Jurja Strossmayera u Osijeku, Fakultet elektrotehnike, računarstva i informacijskih tehnologija Osijek**

Permanent link / Trajna poveznica: https://urn.nsk.hr/urn:nbn:hr:200:372525

Rights / Prava: In copyright / Zaštićeno autorskim pravom.

Download date / Datum preuzimanja: **2024-05-17**

Repository / Repozitorij:

Faculty of Electrical Engineering, Computer Science and Information Technology Osijek

SVEUČILIŠTE JOSIPA JURJA STROSSMAYERA U OSIJEKU
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA OSIJEK

Sveučilišni studij

NONOGRAMI

Završni rad

Ivan Abramušić

Osijek, 2023.

FERIT
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA OSIJEK

Obrazac Z1P - Obrazac za ocjenu završnog rada na preddiplomskom sveučilišnom studiju

Osijek, 01.09.2023.

Odboru za završne i diplomske ispite

Prijedlog ocjene završnog rada na preddiplomskom sveučilišnom studiju

Ime i prezime Pristupnika:	Ivan Abramušić
Studij, smjer:	Programsko inženjerstvo
Mat. br. Pristupnika, godina upisa:	R4456, 27.07.2020.
OIB Pristupnika:	30337103582
Mentor:	izv. prof. dr. sc. Alfonzo Baumgartner
Sumentor:	,
Sumentor iz tvrtke:	
Naslov završnog rada:	Nonogrami
Znanstvena grana rada:	**Programsko inženjerstvo (zn. polje računarstvo)**
Zadatak završnog rad:	[Rezervirano: Ivan Abramušić] Napraviti program koji će omogućiti riješavanje nonograma. Opisati što je nonogram i na koji se način može riješiti. Dati odgovarajuće algoritme. Omogućiti korisniku unos svih potrebnih podataka ili da se ti podaci učitaju iz datoteke, te nakon toga ispisati ili iscrtati dobiveni nonogram, po mogućnosti spremiti rezultat u odgovarajuću tekstualnu datotekku.
Prijedlog ocjene završnog rada:	Izvrstan (5)
Kratko obrazloženje ocjene prema Kriterijima za ocjenjivanje završnih i diplomskih radova:	Primjena znanja stečenih na fakultetu: 3 bod/boda Postignuti rezultati u odnosu na složenost zadatka: 3 bod/boda Jasnoća pismenog izražavanja: 3 bod/boda Razina samostalnosti: 3 razina
Datum prijedloga ocjene od strane mentora:	01.09.2023.
Datum potvrde ocjene od strane Odbora:	08.09.2023.

Potvrda mentora o predaji konačne verzije rada:	Mentor elektronički potpisao predaju konačne verzije.
	Datum: 12.09.2023.

FERIT
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA **OSIJEK**

IZJAVA O ORIGINALNOSTI RADA

Osijek, 12.09.2023.

Ime i prezime studenta:	Ivan Abramušić
Studij:	Programsko inženjerstvo
Mat. br. studenta, godina upisa:	R4456, 27.07.2020.
Turnitin podudaranje [%]:	5

Ovom izjavom izjavljujem da je rad pod nazivom**: Nonogrami**

izrađen pod vodstvom mentora izv. prof. dr. sc. Alfonzo Baumgartner

i sumentora ,

moj vlastiti rad i prema mom najboljem znanju ne sadrži prethodno objavljene ili neobjavljene pisane materijale drugih osoba, osim onih koji su izričito priznati navođenjem literature i drugih izvora informacija.
Izjavljujem da je intelektualni sadržaj navedenog rada proizvod mog vlastitog rada, osim u onom dijelu za koji mi je bila potrebna pomoć mentora, sumentora i drugih osoba, a što je izričito navedeno u radu.

Potpis studenta:

SADRŽAJ

1. UVOD 1

1.1. Zadatak završnog rada 2

2. APLIKACIJE VEZANE UZ NONOGRAME 3

3. KORIŠTENI ALATI ZA IZRADU ALGORITMA 4

3.1. *Microsoft Visual Studio 2022* 4

3.2. Programski jezik C# 4

4. RJEŠAVANJE NONOGRAMA 5

4.1. Primjer rješavanja nonograma 5

5. ALGORITAM RJEŠAVANJA NONOGRAMA 10

5.1. Klasa *Program* 10

5.2. Klasa *NonogramSolver* 12

5.2.1. Generiranje kombinacija 15

5.2.2. Popunjavanje nonograma 19

5.2.3. Ispis nonograma 24

5.2.4. Spremanje nonograma 24

6. PRIKAZ RJEŠENJA NONOGRAMA 25

7. ZAKLJUČAK 28

LITERATURA 29

SAŽETAK 30

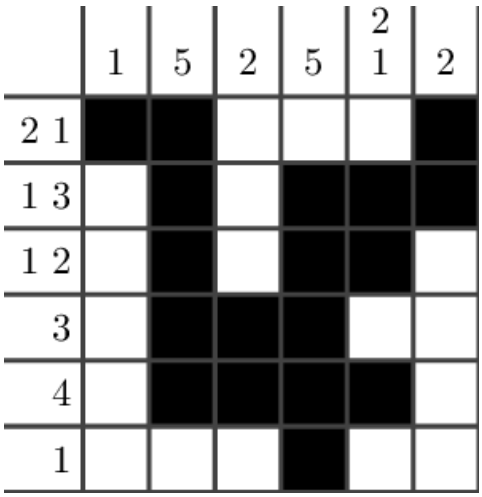
ABSTRACT 31

1. UVOD

Nonogrami su logičke zagonetke koje su predstavljene mrežom određenih dimenzija i brojevima pored svakog retka i iznad svakog stupca. Najčešće se rješavaju crno-bijeli nonogrami, iako mogu biti i višebojni. Potrebno je popuniti sve retke i stupce koristeći vrijednosti brojeva pored njih. Na primjer, ako se pored retka nalaze brojevi „1 2 3“, to bi značilo da je u tom retku ispunjeno jedno polje, zatim razmak od minimalno jednog polja, zatim 2 za redom ispunjena polja, zatim ponovno razmak, te na kraju 3 ispunjena polja. Također, prije prvog i poslije zadnjeg popunjenog polja mogu biti razmaci.

Nonogrami su nastali 1998. godine kada je Non Ishidi, Japanskoj grafičkoj urednici, sinula ideja za zagonetkom koja se bazira na popunjavanju određenih polja unutar mreže predstavljene stupcima i redcima. Te iste godine je objavila svoje prve tri zagonetke pod imenom „Prozorske umjetničke zagonetke (eng. „Window Art Puzzles“). U Ujedinjenom Kraljevstvu, 1990. godine James Dalgety počinje objavljivati te zagonetke jednom tjedno pod imenom „Nonogram“ [1].

U poglavlju broj dva opisane su neke od aplikacija vezane za nonograme, a u poglavlju broj tri su opisani alati koji su se koristili u izradi ovog završnog rada. Od poglavlja četiri do šest je opisan način rješavanja nonograma, odnosno algoritam (poglavlje četiri), programski kod i njegovo objašnjenje (poglavlje 5), te na kraju je dan i primjer rješenog lakšeg i težeg nonograma (poglavlje šest).



Sl. 1.1. Primjer jednog riješenog jednostavnog nonograma

1.1. Zadatak završnog rada

Napraviti program koji će omogućiti riješavanje nonograma. Opisati što je nonogram i na koji se način može riješiti. Dati odgovarajuće algoritme. Omogućiti korisniku unos svih potrebnih podataka ili da se ti podaci učitaju iz datoteke, te nakon toga ispisati ili iscrtati dobiveni nonogram, po mogućnosti spremiti rezultat u odgovarajuću tekstualnu datoteku.

2. APLIKACIJE VEZANE UZ NONOGRAME

Na internetu postoji mnoštvo aplikacija koje su vezane uz nonograme. Većina aplikacija se temelji na principu igre, gdje aplikacija svaki dan ima različit nonogram za rješavanje. Također, korisnik može sam odabrati veličinu i težinu nonograma kojeg želi riješiti, te isto tako mjeriti vrijeme i uspoređivati sa drugim korisnicima. Riješenja nonograma koje ove aplikacije daju za rješavanje su najčešće neke slike poput cvijeta, ptice, kuće ili slično kako bi korisniku rješavanje bilo zanimljivije. Neke od takvih aplikacija su:

- Nonogram.com – picture cross,
- Nonogram – Fun Logic Puzzle,
- Nonograms Katana,
- Nonogram Color – logic puzzle,
- Meow Tower: Nonogram Picross.

Ove aplikacije moraju imati neki algoritam pomoću kojeg rješavaju nonograme u pozadini kako bi na kraju mogle potvrditi da je korisnik dobro riješio nonogram.

Neke web stranice se koriste za rješavanje nonograma. Neke od njih korisniku omogućuju unos podataka red po red, stupac po stupac, te onda iscrtaju nonogram, dok neke prikažu nonogram dimenzija koje smo mi odabrali te podatke pored redaka i supaca koje možemo mjenjati. Pritiskom na tipku nonogram se iscrtava. Neke od takvih stranica su:

- Nonogram Solver online | Dimitriy Fedoriaka's Website,
- Nonogram Solver by Tan Li Hau,
- Free nonogram, griddler, picross puzzle solver (Griddler.co.uk),
- Nonogram Solver (landofcrispy.com).

Također, na internetskim repozitorijima postoje algoritmi za rješavanje nonograma otvorenog koda u raznim programskim jezicima.

3. KORIŠTENI ALATI ZA IZRADU ALGORITMA

Algoritam je pisan u integriranom razvojnom okruženju *Microsoft Virtual Studio 2022*, a kod je pisan u programskom jeziku C#.

3.1. *Microsoft Visual Studio 2022*

Microsoft Visual Studio je integrirano razvojno okruženje napravljeno od strane *Microsofta*. To je okruženje koje služi za razvoj programa, uređivanje programa te uklanjanje grešaka. Koristi se primarno za razvoj računalnih programa, ali se može koristiti i za razvoj mobilnih aplikacija, web-stranica te web-aplikacija. Podržava mnoštvo programskih jezika poput C, C++, C#, JavaScript i dr [2].

Sl. 3.1. *Microsoft Visual Studio* logo

3.2. Programski jezik C#

C# je programski jezik kojeg je dizajnirao Anders Hejlsberg u Microsoftu 2000. godine. On pripada skupini objektno orijentiranih programskih jezika kod kojih je glavno načelo prilagoditi se problemu, a ne problem prilagoditi kodu. Da bi programski jezik bio objektno orijentiran, mora imati tri glavne značajke: nasljeđivanje, enkapsulacija i polimorfizam. C# pripada obitelji C jezika te je jako sličan programskom jeziku C i C++ [3].

4. RJEŠAVANJE NONOGRAMA

Postupak rješavanja nonograma je jako jednostavan. On se sastoji od tri koraka koja kroz više iteracija ponavljamo nad redcima i stupcima kako bi došli do konačnog rješenja. Manji nonogrami će imati manje dok će veći, sa više redaka i stupaca, imati više iteracija.

Tri koraka od kojih se sastoji postupak rješavanja nonograma su:

1. Pronaći sve moguće kombinacije kako se može popuniti dani redak (stupac)
2. Usporediti dani redak (stupac) sa svim mogućim kombinacijama kako bi se izbacile one nemoguće
3. Popuniti polja danog retka (stupca) koja su sigurno popunjena u svim mogućim kombinacijama

Popunjavanjem barem jednog polja unutar retka (stupca) smanjujemo broj kombinacija kojima je moguće popuniti taj redak (stupac), ali i njemu odgovarajući stupac (redak). Ponavljanjem ovih koraka dolazi se do konačnog rješenja nonograma [4].

4.1. Primjer rješavanja nonograma

U ovom dijelu ću pokazati rješavanje manjeg nonograma pomoću navedenih koraka i pravila.

	3	2	2	1 1	4
4					
3 1					
1 2					
1					
1					

Sl. 4.1. Zadani nonogram

Zadani nonogram ima pet redaka i pet stupaca. Prvi korak je pronaći sve moguće kombinacije za svaki redak i stupac. Krenuti ćemo od redaka. Pored prvog retka se nalazi broj četiri, što bi značilo da u tome retku imamo četiri zaredom popunjena polja. Za ovaj redak

imamo moguće samo dvije kombinacije, ili su popunjena prva četiri polja, a peto polje je prazno, ili je prvo polje prazno a zatim popunjena sljedeća četiri retka.

Sl. 4.2. Kombinacije za prvi redak nonograma

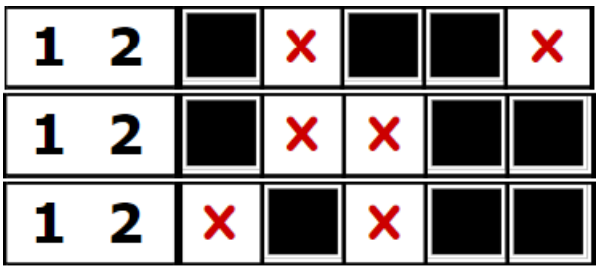
Iz ovoga možemo zaključiti da su središnja tri polja popunjena u svim kombinacijama za taj redak. To znači da ta tri polja možemo sigurno popuniti, a četvrto polje za sada ne znamo.

Pored drugog retka se nalaze brojevi tri i jedan. To znači da su u tome retku popunjena tri polja, zatim razmak, te zatim popunjeno još jedno polje. Za taj redak imamo samo jednu moguću kombinaciju.

Sl. 4.3. Kombinacije za drugi redak nonograma

Ovaj redak ćemo cijeli popuniti, a polja koja sigurno nisu popunjena unutar redaka (stupaca) ćemo označiti sa crvenim iksom.

Treći redak nonograma ima brojeve jedan i dva te se sastoji od tri moguće kombinacije.

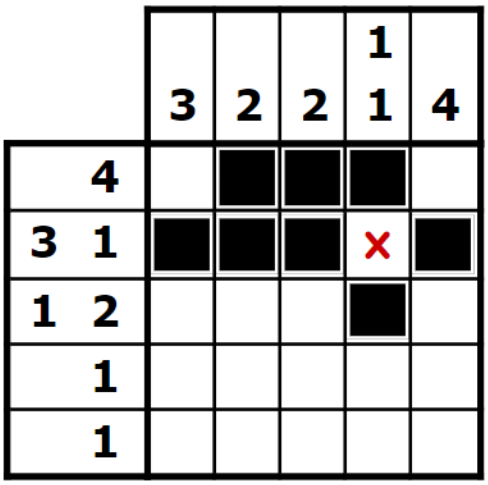


Sl. 4.4. Kombinacije za treći redak nonograma

Vidimo kako je samo četvrto polje sigurno popunjeno u svim kombinacijama te ćemo ga zbog toga popuniti, a za ostala polja nismo sigurni.

Za četvrti i peti redak ima poppet kombinacija jer se pored redaka nalazi broj jedan, odnosno samo je jedno od pet polja popunjeno. Zbog toga ne možemo popuniti niti jedno polje.

Nakon što smo prošli kroz sve retke, naš nonogram sada izgleda ovako.

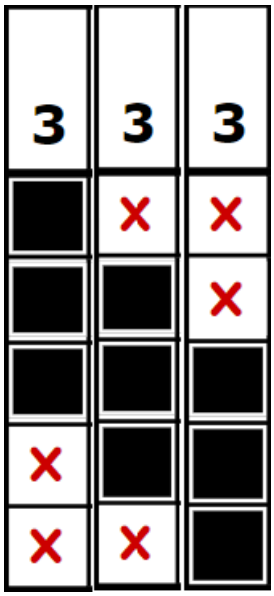


Sl. 4.5. Izgled nonograma nakon prvog prolaska kroz retke

Vidimo kako smo prolaskom kroz retke i popunjavanjem nekih polja olakšali popunjavanje stupaca jer smo smanjili broj mogućih kombinacija.

Nakon što smo prošli kroz sve retke, prolazimo na isti način i kroz sve stupce ali još uspoređujemo moguće kombinacije tog stupca sa istim stupcem u našem nonogramu kako bi smo izbacili nemoguće kombinacije.

Za prvi stupac pored kojeg je broj tri imamo tri moguće kombinacije.



Sl. 4.6. Kombinacije za prvi stupac

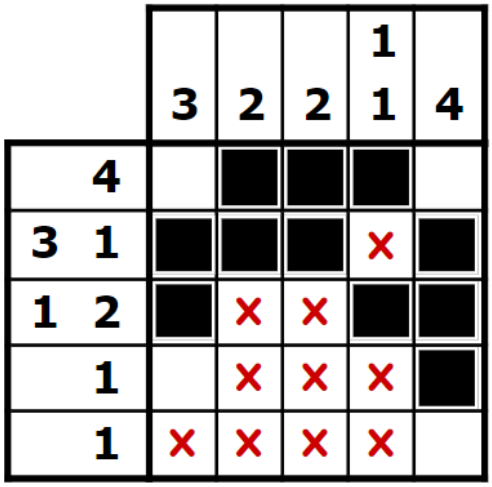
Ove moguće kombinacije uspoređujemo sa tim stupcem u našem nonogramu. Uspoređujući, primjetimo kako treća kombinacija nije moguća da se desi u našem nonogramui, jer je u nonogramu drugo polje prvog stupca sigurno popunjeno preko redaka. Zbog toga treću kombinaciju izbacujemo te promatramo samo prve dvije. Iz njih zaključujemo da su drugo i treće

polje sigurno popunjeni te ih popunjavamo. Također zaključujemo da zadnje peto polje nije popunjeno niti u jednoj od ove dvije kombinacije, te zbog toga možemo u to polje upisati „x“.

Za stupce broj dva i tri ne moramo ništa raditi jer se pored tih stupaca nalazi broj dva, a već su nam dva polja unutar popunjena, odnosno ti stupci su rješeni. Samo trebamo ostala prazna polja popuniti sa „x“. To možemo također učiniti i za stupac broj četiri jer se pored njega nalaze brojevi jedan i jedan, a dva su nam polja već popunjena u tom stupcu.

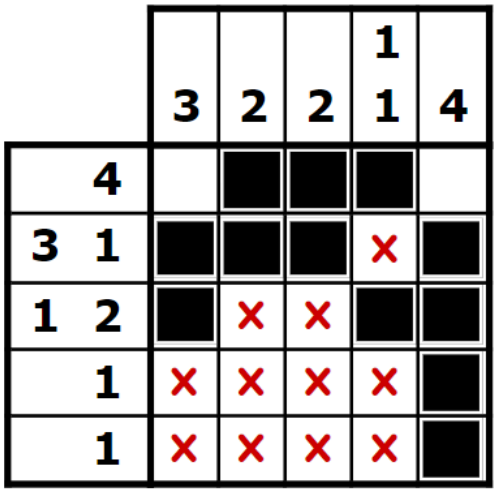
Za stupac broj pet ne možemo eliminirati niti jednu kombinaciju, te ga zatim popunjavamo kao što smo popunili prvi redak.

Nakon prve iteracije kroz retke i kroz stupce, naš nonogram izgleda ovako.

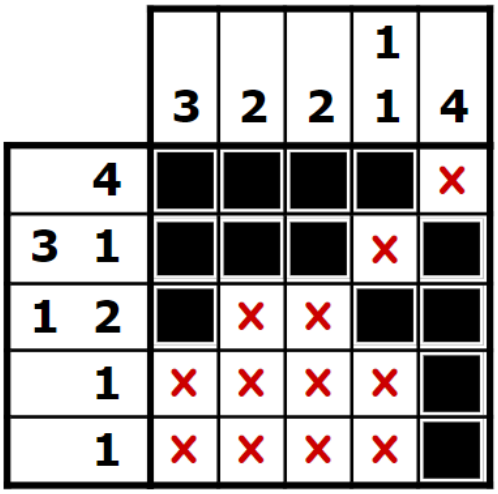


Sl. 4.7. Izgled nonograma nakon prvog prolaska kroz retke i stupce

Naš nonogram je skoro riješen. Još jednim prolaskom kroz redke i stupce, odbacivajući nemoguće kombinacije i popunjavanjem sigurnih polja, naš nonogram može biti riješen.



Sl. 4.8. Izgled nonograma nakon drugog prolaska kroz retke



Sl. 4.9. Rješen nonogram nakon drugog prolaska kroz stupce

Koristeći pravila i prolazeći više puta kroz korake, uspjeli smo rješiti ovaj jednostavni nonogram. Za njega su nam trebala dva prolaska kroz retke i stupce. Na ovaj način možemo rješiti i mnogo veće nonograme, samo će nam za njih trebati mnogo više vremena i prolazaka.

5. ALGORITAM RJEŠAVANJA NONOGRAMA

U ovom poglavlju je objašnjen algoritam rješavanja nonograma. Algoritam je podjeljen na dvije klase: *Program* i *NonogramSolver.*

5.1. Klasa *Program*

Klasa „*Program*“ sadrži glavnu, javnu, statičku, void metodu *Main.* To je metoda koja može biti pozvana bez objekta, kompajler je može izvršiti s bilo kojeg mjesta, te je metoda koju mora sadržavati svaki program napisan u programskom jeziku C#. Od nje kreće izvođenje svakog programa.

Metoda *Main* sadrži šest varijabli:

- *string userInput,*
- *string rawColumns,*
- *string rawRows,*
- *string textfileName,*
- *int [][] colums,*
- *int [][] rows.*

Program od korisnika traži unos znaka preko konzole. Ako korisnik želi učitati podatke (brojeve pored redaka i stupaca) iz tekstualne datoteke unosi slovo „d“ ili „w“ ako ih želi sam ručno unijeti preko konzole. Korisnikov unos se sprema u varijablu „*userInput*“.

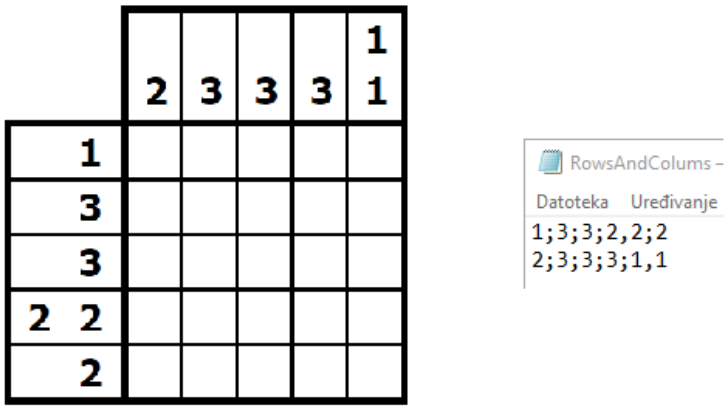
```
using System;
using System.Collections.Generic;
using static System.Linq.Enumerable;

namespace Nonogram_Solver
{
    class Program
    {
        public static void Main()
        {
            string userInput;
            string rawColumns = "";
            string rawRows = "";
            Console.WriteLine("NONOGRAM SOLVER");
            Console.WriteLine("----------------");
            Console.WriteLine();
            Console.Write("Ako zelite ucitati podatke iz tekstualne datoteke napisite 'd', a ako zelite unjeti podatke napisite 'w': ");
            Console.WriteLine();

            do
            {
                userInput = Console.ReadLine();
                if(userInput == "" || userInput != "d" && userInput != "w")
                {
                    Console.WriteLine("Unesite ispravno slovo!");
                }
            }
            while (userInput != "d" && userInput != "w");
```

Sl. 5.1. Korisnikov unos znaka u varijablu „*userInput*“

Podatci unutar datoteke, kao i kad ih korisnik sam unosi, se predaju kao niz znakova i brojeva tako da se sa znakom „ , “ odvajaju brojevi, a sa znakom „ ; “ odvajaju redci (stupci). Redci se u datoteci upisuju u prvi red datoteke, a stupci u drugi.



Sl. 5.2. Nonogram i njegovi podatci u tekstualnoj datoteci

Ukoliko se korisnik odluči za učitavanje podataka iz tekstualne datoteke, potrebno je upisati ime iz koje će se učitati podatci. Ukoliko se odluči da ih sam unosi preko konzole, unosi ih na isti način kao što se nalaze i u datoteci. Podatci redaka se učitavaju u varijablu „*rawRows*“ a podatci stupaca u varijablu „*rawColumns*“.

```
if(userInput == "d")
{
    Console.Write("Unesite ime tekstualne datoteke: ");
    string textfileName = Console.ReadLine();
    rawRows = File.ReadLines(textfileName+".txt").First();
    rawColumns = File.ReadLines(textfileName+".txt").Last();

}
else if(userInput == "w")
{
    Console.WriteLine("Upisi redove. (redove odvajati sa ';', a brojeve sa ',')");
    rawRows = Console.ReadLine();
    Console.WriteLine("Upisi stupce. (stupce odvajati sa ';', a brojeve sa ',')");
    rawColumns = Console.ReadLine();

}
```

Sl. 5.3. Dio koda za unos podataka preko tekstualne datoteke ili konzole

Unešeni podatci redaka i stupaca se zatim cijepe na mjestima gdje se u nalazi znak „ ; “, a zatim i gdje se nalazi znak „ , “. To se radi kako bi smo izlučili svaki broj pojedinačno te sve brojeve spremili u takozvano „*jagged*“ polje. Za razliku od običnog dvodimenzionalnog polja,

kod kojeg su svi redci i stupci jednakih dimenzija, „*jagged*“ polje može sadržavati redke nejednakih dimenzija. Zbog toga je ono izvrsno za spremanje podataka nongrama jer pored svakog redka (stupca) se može nalaziti različita količina brojeva, od jednog pa na više. Podatci redaka se spremaju u varijablu „*rows*“, a podatci stupaca u varijablu „*columns*“ [5].

```
int[][] columns = rawColumns.Split(';').Select(x => x.Split(',').Select(y => int.Parse(y)).ToArray()).ToArray();
int[][] rows = rawRows.Split(';').Select(x => x.Split(',').Select(y => int.Parse(y)).ToArray()).ToArray();
```

Sl. 5.4. Dio koda za spremanje podataka nonograma u „jagged“ polja

Nakon što smo sredili podatke nonograma, kreiramo objekt druge klase „*NonogramSolver*“ te mu predajemo te iste podatke. Na poslijetku, pozivamo tri glavne funkcije objekta klase „*NonogramSolver*“ te sa time rješavamo, crtamo i spremamo zadani nonogram.

```
NonogramSolver solver = new NonogramSolver(columns, rows);

solver.Solve();
solver.Paint();
solver.Save();
```

Sl. 5.5. Stvaranje objekta klase „*NonogramSolver*“ i pozivanje njegovih glavnih metoda

5.2. Klasa *NonogramSolver*

Klasa „*NonogramSolver*“ je zadužena za rješavanje, ispis i spremanje rješenja nonograma u tekstualnu datoteku. Sadrži tri atributa:

- *int [][] cols,*
- *int [][] rows,*
- *int [,] board.*

Atributima „*cols*“ i „*rows*“ se vrijednost pridružuje kroz konstruktor, dok se atribut „*board*“, koji predstavalja polje nonograma, popunjava vrijednošću „0“ pomoću metode „*Fill2DArrayWithValue*“ koja se poziva unutar konstruktora.

```
internal class NonogramSolver
{
    private int [][] cols;
    private int [][] rows;
    private int [,] board;



    public NonogramSolver(int[][] cols, int[][] rows)
    {
        this.rows = rows;
        this.cols = cols;
        board = new int[rows.GetLength(0), cols.GetLength(0)];
        Fill2DArrayWithValue(board, 0);

    }
```

Sl. 5.6. Konstruktor klase „*NonogramSolver*“

```
private void Fill2DArrayWithValue(int[,] array, int value)
{
    for (int i = 0; i < array.GetLength(0); i++)
    {
        for (int j = 0; j < array.GetLength(1); j++)
        {
            array[i, j] = value;
        }
    }
}
```

Sl. 5.7. Metoda „*Fill2DArrayWithValue*“ za popunjavanje 2D polja sa predanom vrijednosti

Glavna metoda pomoću koje se rješava zadani nonogram je metoda „*Solve*“. Unutar nje se prolazi kroz tri koraka od kojih se sastoji postupak rješavanja nonograma. Prije samog rješavanja nonograma se provjerava da li je zadani nonogram uopće rješiv. Za tu svrhu koristimo metodu „*IsSolvable*“ koju pozivamo na početku metode „*Solve*“. Ona radi na način da usporedi zbroj brojeva pored redaka i zbroj brojeva pored stupaca. Ako su zbrojevi različiti, znači da nismo dobro unijeli podatke te da nonogram nije rješiv.

```
private void IsSolvable()
{
    int sumOfRows = 0;
    int sumOfCols = 0;

    foreach (int[] innerArray in rows)
    {
        foreach (int element in innerArray)
        {
            sumOfRows += element;
        }
    }

    foreach (int[] innerArray in cols)
    {
        foreach (int element in innerArray)
        {
            sumOfCols += element;
        }
    }

    if (sumOfRows != sumOfCols)
    {
        Console.WriteLine("Nonogram is not solvable!");
        Environment.Exit(0);
    }

}
```

Sl. 5.8. Metoda za provjeru rješivosti nonograma

Nakon provjere, kreće se u rješavanje nonograma. Rješavanje nonograma se odvija u koracima koji su navedeni u poglavlju 3. koristeći tri glavne pomoćne metode. Prolazeći pomoću petlji, za svaki redak (stupac) se računaju sve moguće kombinacije. Svaka od kombinacija se uspoređuje sa pravim popunjenim retkom (stupcem) kako bi se izbacile nemoguće kombinacije te se zatim ona polja koja su sigurna popunjavaju.

```
public void Solve()
{
    IsSolvable();

    do
    {
        for (int i = 0; i < rows.GetLength(0); i++)
        {
            int[,] combinations = CalculateCombinations(rows[i], cols.GetLength(0));
            FillRowOnBoard(combinations, i);
        }
        for (int i = 0; i < cols.GetLength(0); i++)
        {
            int[,] combinations = CalculateCombinations(cols[i], rows.GetLength(0));
            FillColumnOnBoard(combinations, i);
        }

    } while (IsSolved());
}
```

Sl. 5.9. Metoda „*Solve*“ za rješavanje nonograma

Ovaj postupak se ponavlja sve dok nonogram nije rješen. To provjeravamo pomoću *bool* metode „*IsSolved*“ koja provjerava da li se u atributu „*board*“, koji predstavlja polje nonograma, na bilo kojem mjestu nalazi broj „0“. Ako se nalazi, metoda vraća „true“ te govori kako nonogram nije riješen. U suprotnom, ako se nigdje ne nalazi broj „0“, tada smo popunili sva polja nonograma, on je riješen te metoda vraća „false“.

```
private bool IsSolved()
{
    for (int i = 0; i < board.GetLength(0); i++)
    {
        for (int j = 0; j < board.GetLength(1); j++)
        {
            if(board[i, j] == 0)
            {
                return true;
            }

        }
    }
    return false;
}
```

Sl. 5.10. Metoda „*IsSolved*“

5.2.1. Generiranje kombinacija

Prvi korak pri rješavanju nonograma je generirati sve moguće kombinacije za svaki redak i stupac. Za to koristimo prvu glavnu pomoćnu metodu „*CalculateCombinations*“. Ova metoda kao ulaz prima podatke pored jednog retka ili stupca te njegovu duljinu, a za izlaz dobijamo polje gdje svaki redak predstavlja jednu kombinaciju na način da broj „1“ predstavlja popunjeno polje, a broj „-1“ nepopunjeno.

```
1 1 -1 -1 -1
-1 1 1 -1 -1
-1 -1 1 1 -1
-1 -1 -1 1 1
```

Sl. 5.11. Izgled svih kombinacija retka pored kojeg se nalazi broj dva

Unutar metode „*CalculateCombinations*“ prvo računamo broj kombinacija kako bi znali veličinu polja u koje ćemo spremiti sve kombinacije. Broj kombinacija računa metoda „*CalculateNumberOfCombinations*“ koristeći binomni koeficijent [6]. Metoda za ulaz prima dva

broja, broj „k“ koji predstavlja koliko se brojeva nalazi pored retka (stupca), tj. broj grupa i broj „n“ koji predstavlja broj grupa plus praznine, ne uključujući prazninu od jednog polja koja se mora naći između dvije grupe. Za primjer ćemo uzeti redak pored kojeg se nalaze brojevi tri i tri, a redak je duljine deset. „K“ će u ovom primjeru biti dva, jer imamo dva broja pored retka. Računanje broja „n“ je malo teže. Praznine ćemo izračunati tako što ćemo od duljine retka koja iznosi deset oduzeti broj polja koja moramo popuniti a to je šest i praznine koje se nalaze između grupa a to je jedan. Tako da u ovom primjeru imamo tri praznine, odnosno broj „n“ će biti pet.Binomni koeficijent gdje je broj „n“ pet, a broj „k“ dva iznosi deset. To znači da ovaj redak ima deset različitih kombinacija [7].

```
long k = array.Count();
long blanks = length - array.Sum() - (array.Count() - 1);
long n = k + blanks;

long numberOfCombinations = CalculateNumberOfCombinations(n, k);
```

Sl. 5.12. Računanje vrijednosti „n“ i „k“

```
private long CalculateNumberOfCombinations(long n, long k)
{
    long numerator = Factorial(n);
    long denominator = Factorial(k) * Factorial(n-k);
    return numerator / denominator;
}

private long Factorial(long number)
{
    if (number <= 1)
        return 1;
    else
        return number * Factorial(number - 1);
}
```

Sl. 5.13. Funkcija za računanje broja kombinacija

Nakon što smo izračunali broj kombinacija, potrebno je dobiti sve kombinacije. Pri tome će nam pomoći brojevi „n“ i „k“ koje smo prethodno izračunali. Zamislimo kako imamo „n“ različitih brojeva u skupu i želimo uzimati po „k“ brojeva bez ponavljanja. Upravo to radimo pomoću metode „*GetCombinationsWithElements*“ kojoj predajemo listu različitih brojeva počevši od nula do broja „n“ i broj „k“, a ona nam vraća sve moguće kombinacije od po „k“ brojeva. Za prethodni primjer gdje je broj „n“ iznosio pet a „k“ dva bi imali 10 različitih kombinacija i to: (0,1), (0,2), (0,3), (0,4), (1,2), (1,3), (1,4), (2,3), (2,4), (3,4) [8].

```
int[,] combinationsWithElements = new int[numberOfCombinations, k];
int[,] combinations = new int[numberOfCombinations, length];

List<int> elements = new List<int>();
for(int i = 0; i < n; i++)
{
    elements.Add(i);
}

var result = GetCombinationsWithElements(elements, k);
int x = 0, y = 0;
foreach (var perm in result)
{
    foreach (var c in perm)
    {
        combinationsWithElements[x,y] = c;
        y++;
    }
    y = 0;
    x++;
}
x = 0;
y = 0;
```

Sl. 5.14. Poziv metode „*GetCombinationsWithElements*“ i spremanje rezultata u polje

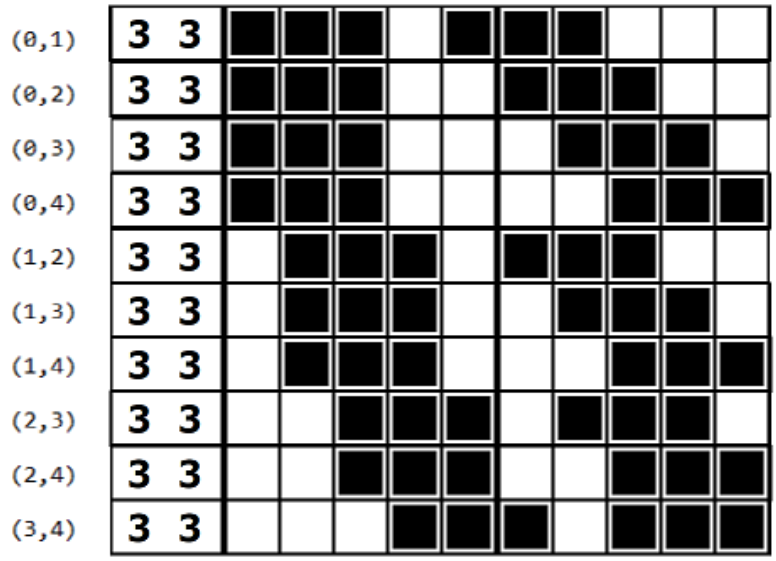
```
private IEnumerable<IEnumerable<T>> GetCombinationsWithElements <T>(IEnumerable<T> items, long count)
{
    int i = 0;
    foreach(var item in items)
    {
        if(count == 1)
        {
            yield return new T[] { item };
        }
        else
        {
            foreach (var result in GetCombinationsWithElements(items.Skip(i + 1), count - 1))
                yield return new T[] { item }.Concat(result);
        }
        ++i;
    }
}
```

Sl. 5.15. Funkcija za računanje svih kombinacija

Sada kada imamo sve kombinacije jednog retka (stupca), potrebno ih je transformirati kako bi dobili kombinacije kakve su prikazane na početku 4.2.1. poglavlja. Metodom „*TransformCombinations*“ koja prima polje prethodno izračunatih kombinacija, redak (stupac) za koji računamo kombinacije te duljinu samog retka, dobijamo upravo tako prikazane kombinacije. Njih ćemo zatim proslijediti drugoj funkciji koja će se pobrinuti za popunjavanje retka (stupca) nonograma.

Metoda „*TransformCombinations*“ prvo stvara dvodimenzionalno polje u koje ćemo unositi kombinacije te ga popunjava sa brojem „-1“ metodom „*Fill2DArrayWithValue*“. Zatim

se svaka kombinacija uzima pojedinačno te pretvara na način da prvi broj unutar kombinacije označava indeks od kojeg se kreće popunjavanje prve grupe, a razlika između susjedna dva broja označava koliko se praznih mjesta nalazi između te dvije grupe. Obojano polje će biti označeno sa „1“ a neobojano sa „-1“. Pri popunjavanju polja kombinacija sa „1“ nam pomaže metoda „*Fill*“.



Sl. 5.16. Kombinacija sa njenim pripadajućim popunjenim retkom

```
1 1 1 -1 1 1 1 -1 -1 -1
1 1 1 -1 -1 1 1 1 -1 -1
1 1 1 -1 -1 -1 1 1 1 -1
1 1 1 -1 -1 -1 -1 1 1 1
-1 1 1 1 -1 1 1 1 -1 -1
-1 1 1 1 -1 -1 1 1 1 -1
-1 1 1 1 -1 -1 -1 1 1 1
-1 -1 1 1 1 -1 1 1 1 -1
-1 -1 1 1 1 -1 -1 1 1 1
-1 -1 -1 1 1 1 -1 1 1 1
```

Sl. 5.17. Polje svih kombinacija za redak duljine deset sa podatcima tri i tri

Nakon što se sve kombinacije transformiraju, novo polje kombinacija se šalje drugoj funkciji dalje za popunjavanje nonograma.

```
private int[,] TransformCombinations(int[,] combinationsWithElements, int[] rowOrCol, int length)
{
    int[,] combinations = new int[combinationsWithElements.GetLength(0), length];
    Fill2DArrayWithValue(combinations, -1);

    for(int i = 0; i < combinationsWithElements.GetLength(0); i++)
    {
        int index = combinationsWithElements[i, 0];
        for (int j = 0; j < combinationsWithElements.GetLength(1); j++)
        {
            index = Fill(combinations, rowOrCol[j], index, i);
            if (j + 1 < combinationsWithElements.GetLength(1))
            {
                index = index + combinationsWithElements[i, j + 1] - combinationsWithElements[i, j];
            }
        }
    }

    return combinations;
}
```

Sl. 5.18. Metoda za transformiranje kombinacija

```
private int Fill(int[,] combinations, int element, int index, int i)
{

    for(int j = 0; j< element; j++ )
    {
        combinations[i, index] = 1;
        index++;
    }

    return index;
}
```

Sl. 5.19. Metoda „*Fill*“ za popunjavanje polja kombinacijama

5.2.2. Popunjavanje nonograma

Za popunjavanje nonograma se koriste dvije glavne pomoćne metode, „*FillRowOnBoard*“ koja popunjava redak nonograma i „*FillColumnOnBoard*“ koja popunjava stupac nonograma. Objema metodama se predaju prethodno izračunate kombinacije za redak (stupac) i indeks retka (stupca) kako bi funkcija znala koji redak (stupac) popunjavamo. U narednom tekstu ću koristiti slike metode „*FillRowOnBoard*“ a na kraju ću prikazati i cijelu metodu „*FillColumnOnBoard*“.

Ove metode se sastoje od tri dijela. Prvo provjeravamo da li neko polje retka (stupca) popunjeno. To radimo kako bi smo tijekom prve iteracije preskočili drugi korak, odnosno usporedbu retka (stupca) nonograma sa svim njegovim mogućim kombinacijama jer u prvom proslasku kroz retke i stupce, nonogram je prazan te je svaka od kombinacija moguća. To činimo na jednostavan način provjeravajući da li se u retku (stupcu) atributa „*board*“ nalazi broj različit

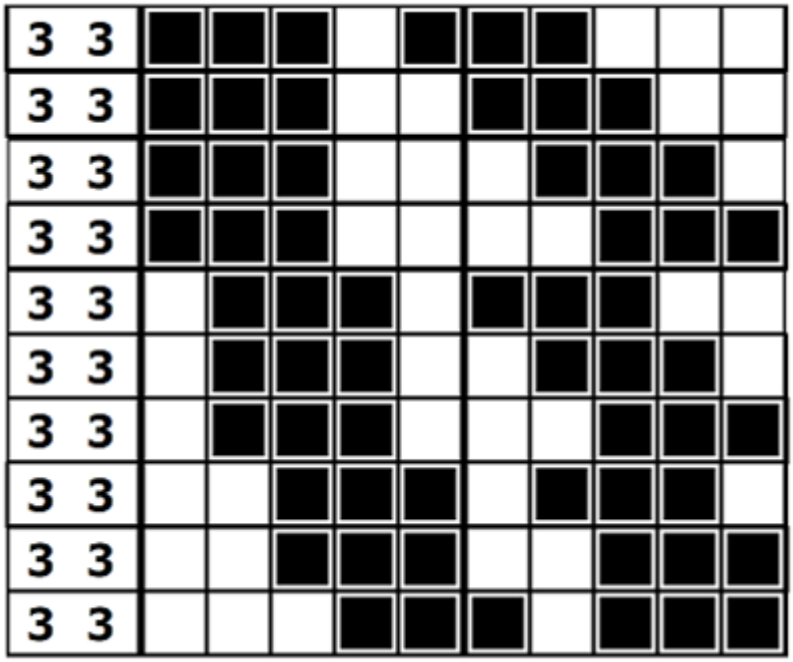
od nule. Ako se nalazi, atribut „*flag*“ postavljamo na „1“ te to znači da ovo nije prva iteracija te se izračunate kombinacije trebaju uspoređivati s tim retkom (stupcem).

```
for (int i = 0; i<board.GetLength(1); i++)
{
    if (board[row,i] == 0)
    {
        continue;
    }
    else
    {
        flag = 1;
    }
}
```

Sl. 5.20. Provjera da li se radi o prvoj iteraciji prolaska

Drugi korak je usporedba retka (stupca) sa svim mogućim kombinacijama kako bi se izbacile one nemoguće. Redak (stupac) nonograma se uspoređuje sa svakom od kombinacija na način da se uspoređuju polja na istim indeksima. Ako su polja različita, znači da ta kombinacija nije moguća te ju je potrebno izbaciti. Za primjer ćemo uzeti redak kao u prethodnim poglavljima.

Sl. 5.21. Redak polja u nonogramu



Sl. 5.22. Sve moguće kombinacije retka na slici 4.20.

Vidimo kako je redak u nonogramu popunjen na četvrtom polju. Uspoređujući taj redak sa svim njegovim kombinacijama možemo zaključiti kako prve četiri kombinacije nisu moguće jer kod njih četvrto polje nije popunjeno. Zbog toga ćemo te kombinacije izbaciti na način da ćemo ih samo popuniti s „0“ na svakom indeksu. Usput ćemo još zabilježiti koliko smo kombinacija izbacili u atribut „*combRemoved*“. Također, ako je polje u retku (stupcu) nonograma popunjeno s „0“, njega preskačemo i to polje ne uspoređujemo.

```
if (flag == 1)
{
    for (int i = 0; i < combinations.GetLength(0); i++)
    {
        for (int j = 0; j < combinations.GetLength(1); j++)
        {
            if (board[row, j] == 0)
            {
                continue;
            }
            else if (combinations[i, j] != board[row, j])
            {
                FillRowOf2DArrayWithValue(combinations, 0, i);
                combRemoved++;
                break;
            }
        }
    }
}
```

Sl. 5.23. Usporedba retka sa svim njegovim mogućim kombinacijama

```
private void FillRowOf2DArrayWithValue(int[,] array, int value, int row)
{
    for(int i = 0; i < array.GetLength(1); i++)
    {
        array[row,i] = value;
    }
}
```

Sl. 5.24. Metoda za popunjavanje retka 2D polja sa vrijednosti

Treći ujedno i zadnji korak je popuniti polja retka (stupca) nonograma koja su popunjena u svim kombinacijama. Kako bi smo dobili polja koja su popunjena u svim kombinacijama, potrebno je samo zbrojiti sva polja kombinacija po stupcima. Uradivši to, dobijamo jednodimenzionalno polje sa različitim brojevima. Budući da su kombinacije prikazane sa brojevima „1“ i „-1“, njihovim zbrajanjem možemo utvrditi koja mjesta sigurno treba popuniti sa „1“ a koja sa „-1“. Ako se u jednodimenzionalnom polju na nekom indeksu nađe broj jednak

broju kombinacija, znamo da je to polje popunjeno u svakoj kombinaciji te ga možemo popuniti sa „1“ u nonogramu. U suprotnom, ako se nađe broj jednak minusu broja kombinacija, to polje možemo popuniti sa „-1“ u nonogramu.

Na slici 4.22., izbacivanjem prve četiri kombinacije, i zbrajanjem preostalih šest, dobijamo jednodimenzionalno polje sa vrijednostima: -6, 0, 4, 6, 0, -2, 0, 6, 4, 0. Kako smo zbrojili šest kombinacija, znamo da polje gdje se nalazi „6“ ili „-6“ je polje koje možemo sigurno popuniti sa „1“ ili „-1“ u nonogramu. Za zbrajanje kombinacija koristimo metodu „*GetSumOfCombinations*“, te odmah nakon poziva te metode slijedi popunjavanje polja nonograma.

```
sumOfCombinations = new int[combinations.GetLength(1)];

sumOfCombinations = GetSumOfCombinations(combinations);
numberOfCombinations = combinations.GetLength(0) - combRemoved;


for(int i = 0; i < sumOfCombinations.GetLength(0); i++)
{
    if (sumOfCombinations[i] == numberOfCombinations)
    {
        board[row, i] = 1;
    }
    else if(sumOfCombinations[i] == -(numberOfCombinations))
    {
        board[row, i] = -1;
    }
}
```

Sl. 5.25. Popunjavanje polja nonograma

```
private int[] GetSumOfCombinations(int[,] array)
{
    int[] sumOfCombinations = new int[array.GetLength(1)];
    FillArrayWithValue(sumOfCombinations,0);
    for(int i = 0; i < array.GetLength(1); i++)
    {
        for(int j = 0; j < array.GetLength(0); j++)
        {
            sumOfCombinations[i] = sumOfCombinations[i] + array[j, i];
        }
    }

    return sumOfCombinations;
}
```

Sl. 5.26. Metoda za računanje zbroja kombinacija

Ovaj isti postupak se ponavlja i za sve stupce pomoću metode „*FillColumnOnBoard*“.

```
private void FillColumnOnBoard(int[,] combinations, int col)
{
    int flag = 0;
    int combRemoved = 0;
    int numberOfCombinations;
    int[] sumOfCombinations;

    for (int i = 0; i < board.GetLength(0); i++)
    {
        if (board[i, col] == 0)
        {
            continue;
        }
        else
        {
            flag = 1;
        }
    }

    if (flag == 1)
    {
        for (int i = 0; i < combinations.GetLength(0); i++)
        {
            for (int j = 0; j < combinations.GetLength(1); j++)
            {
                if (board[j, col] == 0)
                {
                    continue;
                }
                else if (combinations[i, j] != board[j, col])
                {
                    FillRowOf2DArrayWithValue(combinations, 0, i);
                    combRemoved++;
                    break;
                }
            }
        }

    }

    sumOfCombinations = new int[combinations.GetLength(1)];

    sumOfCombinations = GetSumOfCombinations(combinations);
    numberOfCombinations = combinations.GetLength(0) - combRemoved;

    for (int i = 0; i < sumOfCombinations.GetLength(0); i++)
    {
        if (sumOfCombinations[i] == numberOfCombinations)
        {
            board[i, col] = 1;
        }
        else if (sumOfCombinations[i] == -(numberOfCombinations))
        {
            board[i, col] = -1;
        }
    }

}
```

Sl. 5.27. Metoda za popunjavanje stupaca nonograma

Nakon više iteracija i poziva metoda dolazi se do rješenja cijelog nonograma.

5.2.3. Ispis nonograma

Ispis nonograma obavlja metoda „*Paint*“ koja prolazi kroz polje nonograma te na mjestu gdje je „1“ ona ispiše znak „#“, a gdje je „-1“ ona ispiše znak „.“.

```
public void Paint()
{
    for (int i = 0; i < board.GetLength(0); i++)
    {
        for (int j = 0; j < board.GetLength(1); j++)
        {
            if(board[i, j] == 1)
            {
                Console.Write("# ");
            }
            else if(board[i, j] == -1)
            {
                Console.Write(". ");
            }
        }
        Console.WriteLine();
    }
}
```

Sl. 5.28. Metoda za prikaz rješenja nonograma

5.2.4. Spremanje nonograma

Spremanje polja nonograma obavlja metoda „*Save*“ u tekstualnu datoteku „*Output.txt*“.

```
public void Save()
{
    using (var sw = new StreamWriter("Output.txt"))
    {
        for(int i = 0; i < board.GetLength(0); i++)
        {
            for (int j = 0; j < board.GetLength(1); j++)
            {
                if (board[i, j] == 1)
                {
                    sw.Write("# ");
                }
                else if (board[i, j] == -1)
                {
                    sw.Write(". ");
                }
            }
            sw.Write("\n");
        }
        sw.Flush();
        sw.Close();
    }
}
```

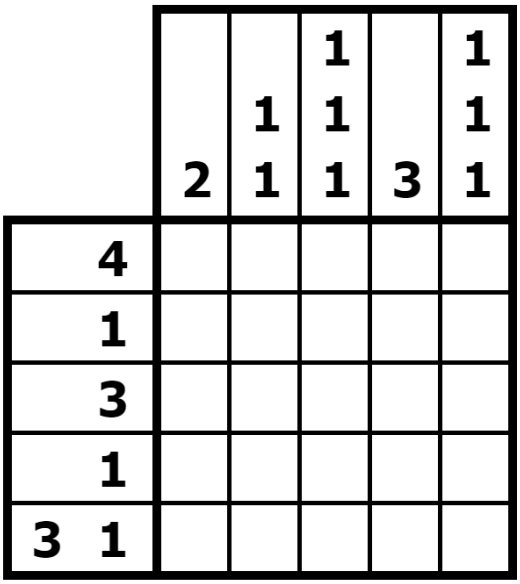
Sl. 5.29. Metoda za spremanje rješenja nonograma u datoteku

6. PRIKAZ RJEŠENJA NONOGRAMA

U ovom poglavlju ću prikazati ulaz i rješenja dva nonograma, jednog jednostavnijeg i jednog kompliciranijeg.

Rješenje jednostavnog nonograma

Za jednostavni nonogram ćemo uzeti nonogram dimenzija pet puta pet.



Sl. 6.1. Jednostavni nonogram

Ulaz i izlaz za ovaj nonogram će izgledati ovako.

```
NONOGRAM SOLVER
---------------

Ako zelite ucitati podatke iz tekstualne datoteke napisite 'd', a ako zelite unjeti podatke napisite 'w':
w
Upisi redove. (redove odvajati sa ';', a brojeve sa ',')
4;1;3;1;3,1
Upisi stupce. (stupce odvajati sa ';', a brojeve sa ',')
2;1,1;1,1,1;3;1,1,1


. # # # #
. . . # .
. . # # #
# . . . .
# # # . #
```

Sl. 6.2. Ulaz i rješenje jednostavnog nonograma

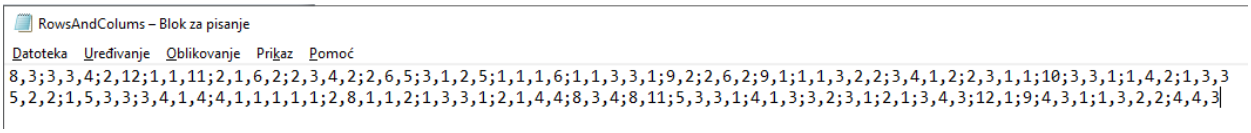
Rješenje kompliciranijeg nonograma

Za kompliciraniji nonogram ćemo uzeti nonogram dimenzija dvadeset puta dvadeset.

	5 2 2	1 5 3 3	3 4 1 4	4 1 1 1 1	2 8 1 1 2	1 3 3 1	2 1 4 4	8 3 4	8 11	5 3 3 1	4 1 3	3 2	3 1	2 1	3 4 3	12 1	9	4 3 1	1 3 2 2	4 4 3
8 3																				
3 3 4																				
2 12																				
1 1 11																				
2 1 6 2																				
2 3 4 2																				
2 6 5																				
3 1 2 5																				
1 1 1 6																				
1 1 3 3 1																				
9 2																				
2 6 2																				
9 1																				
1 1 3 2 2																				
3 4 1 2																				
2 3 1 1																				
10																				
3 3 1																				
1 4 2																				
1 3 3																				

Sl. 6.3. Kompliciraniji nonogram

Ulaz ovog nonograma ćemo upisati u tekstualnu datoteku „*RowsAndColums*“ a izlaz će izgledati ovako.



Sl. 6.4. Ulaz nonograma napisan u tekstualnoj datoteci

Sl. 6.5. Rješenje kompliciranijeg nonograma

7. ZAKLJUČAK

Cilj ovog rada bio je izraditi algoritam i napisati program u programskom jeziku C# koji će uspješno rješavati nonograme te ih spremati u tekstualnu datoteku. Zadatak je uspješno izvršen te je dobiven program koji rješava nonograme na način da korisnik putem konzole upiše podatke nonograma ili ih učita iz datoteke, a program rješi, iscrta i spremi rješenje nonograma.

Prednosti ovog programa su što je jako jednostavan, prati se samo par koraka koji se ponavljaju iznova i iznova te se dolazi do rješenja u jako kratkom vremenu.

Nedostatci ovog programa su što bi kod velikih nonograma moglo doći do nemogućnosti rješavanja zbog izračuna broja kombinacija. Za računanje broja kombinacija se koristi binomni koeficijent i njegova matematička formula gdje se računa faktorijel broja „n“. Kako broj „n“ u većim nonogramima može biti veliki, njegov faktorijel bi mogao biti preveliki za spremanje u atribut.

Također, ovaj program je napisan kako bi se uštedilo na memoriji, a smanjila brzina izvođenja jer program svakom iteracijom ponovo računa sve kombinacije za svaki redak i stupac. Umjesto toga, ako bi htjeli postići veću brzinu, mogli bi izračunati samo jednom sve kombinacije te ih zatim spremiti u različita polja, ali bi tako zauzeli previše memorije za spremanje svih kombinacija svakog retka i stupca.

LITERATURA

[1] The History and Origin of Nonogram game, 2014-2018, dostupno na: https://puzzlygame.com/pages/nonogram_history/ [29.6.2023.]

[2] Visual Studio 2022, dostupno na: https://visualstudio.microsoft.com [30.6.2023.]

[3] A tour of the C# language, dostupno na: https://learn.microsoft.com/en-us/dotnet/csharp/tour-of-csharp/ [30.6.2023.]

[4] Chugunnyy K.A. (KyberPrizrak), Learn to solve Japanese crosswords, 2009-2023, dostupno na: https://www.nonograms.org/instructions [20.8.2023.]

[5] Jagged Arrays (C# Programming Guide), dostupno na: https://learn.microsoft.com/en-us/dotnet/csharp/programming-guide/arrays/jagged-arrays [22.8.2023.]

[6] Leksikografski zavod Miroslav Krleža, 2021, Binomni koeficijent, dostupno na: https://enciklopedija.hr/natuknica.aspx?ID=70950 [23.8.2023.]

[7] Hennie de Harder, Solving Nonograms with 120 Lines of Code, 1.7.2022., dostupno na: https://towardsdatascience.com/solving-nonograms-with-120-lines-of-code-a7c6e0f627e4 [23.8.2023.]

[8] MathIsFun, Combinations and Permutations, 2021, dostupno na: https://www.mathsisfun.com/combinatorics/combinations-permutations.html [23.8.2023.]

SAŽETAK

Nonogrami su logičke zagonetke čiji je cilj popuniti polje koristeći podatke (brojeve) pored redaka i stupaca. Zadatak rada je bio razviti algoritam i napisati program u programskom jeziku C# koji će moći rješiti nonograme. Unos podataka nonograma omogućen je korisniku preko konzole ili učitavanjem iz tekstualne datoteke. Rješenje nonograma se ispisuje na ekran te sprema u tekstualnu datoteku.

Ključne riječi: nonogram, algoritam, programski jezik C#, Japanese crosswords

ABSTRACT

Algorithm for logic puzzle nonogram in programming language C#

Nonograms are logic puzzles whose goal is to fill in a field using data (numbers) next to the rows and columns. The task of the work was to develop an algorithm and write a program in the programming language C# that will be able to solve nonograms. Nonogram data entry is enabled by the user through the console or by loading from a text file. The nonogram solution is printed on the screen and saved in a text file.

Keywords: nonogram, algorithm, programming language C#, Japanese crosswords