Brzi algoritam za rješavanje sudoku zagonetke u programskom jeziku C

Knežević, Tomislav

Master's thesis / Diplomski rad

2020

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:200:893622

Rights / Prava: In copyright / Zaštićeno autorskim pravom.

Download date / Datum preuzimanja: **2025-03-21**

Repository / Repozitorij:

Faculty of Electrical Engineering, Computer Science and Information Technology Osijek

SVEUČILIŠTE JOSIPA JURJA STOSSMAYERA U OSIJEKU

FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA

Sveučilišni diplomski studij

BRZI ALGORITAM ZA RJEŠAVANJE SUDOKU ZAGONETKE U PROGRAMSKOM JEZIKU C++

Diplomski rad

Tomislav Knežević

Osijek, 2020.

FERIT
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA OSIJEK

Obrazac D1: Obrazac za imenovanje Povjerenstva za diplomski ispit

Osijek, 28.09.2020.

Odboru za završne i diplomske ispite

Imenovanje Povjerenstva za diplomski ispit

Ime i prezime studenta:	Tomislav Knežević
Studij, smjer:	Diplomski sveučilišni studij Računarstvo
Mat. br. studenta, godina upisa:	D-914R, 21.10.2019.
OIB studenta:	31853590562
Mentor:	Doc.dr.sc. Tomislav Rudec
Sumentor:	Izv. prof. dr. sc. Alfonzo Baumgartner
Sumentor iz tvrtke:	
Predsjednik Povjerenstva:	Doc.dr.sc. Anita Katić
Član Povjerenstva 1:	Doc.dr.sc. Tomislav Rudec
Član Povjerenstva 2:	Doc. dr. sc. Goran Rozing
Naslov diplomskog rada:	Brzi algoritam za rješavanje sudoku zagonetke u programskom jeziku C
Znanstvena grana rada:	**Programsko inženjerstvo (zn. polje računarstvo)**
Zadatak diplomskog rada:	Tema je zauzeta - Tomislav Knežević
Prijedlog ocjene pismenog dijela ispita (diplomskog rada):	Dobar (3)
Kratko obrazloženje ocjene prema Kriterijima za ocjenjivanje završnih i diplomskih radova:	Primjena znanja stečenih na fakultetu: 2 bod/boda Postignuti rezultati u odnosu na složenost zadatka: 3 bod/boda Jasnoća pismenog izražavanja: 1 bod/boda Razina samostalnosti: 1 razina
Datum prijedloga ocjene mentora:	28.09.2020.

Potpis mentora za predaju konačne verzije rada u Studentsku službu pri završetku studija:	Potpis:
	Datum:

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

IZJAVA O ORIGINALNOSTI RADA

Osijek, 11.10.2020.

Ime i prezime studenta:	Tomislav Knežević
Studij:	Diplomski sveučilišni studij Računarstvo
Mat. br. studenta, godina upisa:	D-914R, 21.10.2019.
Turnitin podudaranje [%]:	26

Ovom izjavom izjavljujem da je rad pod nazivom**: Brzi algoritam za rješavanje sudoku zagonetke u programskom jeziku C**

izrađen pod vodstvom mentora Doc.dr.sc. Tomislav Rudec

i sumentora Izv. prof. dr. sc. Alfonzo Baumgartner

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 diplomskog rada 1

2. SUDOKU LOGIČKA ZAGONETKA 2

2.1. Pregled područja teme 2

2.2. Uvod u sudoku 3

2.3. Latinski kvadrat 4

2.4. Povijest sudoku zagonetke 5

3. PROGRAMSKI JEZIK C++ 6

3.1. Povijest programskog jezika C++ 6

3.2. Osnovna svojstva programskog jezika C++ 8

3.3 Microsoft Visual Studio okruženje 10

4. PROGRAMSKO RJEŠENJE ALGORITMA ZA SUDOKU 13

4.1. Osnovne ideje na kojima se zasniva algoritam 13

4.1.1 Algoritam strategije povlačenja (Backtracking algoritam) 13

4.1.2. Optimizirani backtracking algoritam 14

4.2. Programski kod i analiza 16

4.2.1. Backtracking algoritam 16

4.2.2. Optimizacija algoritma 19

4.3 Rad programskog koda i rješenja 23

ZAKLJUČAK 27

SAŽETAK 28

SUMMARY 29

ŽIVOTOPIS 30

LITERATURA 31

1. UVOD

Sudoku je matematička zagonetka čije se rješavanje temelji na logici. Klasična sudoku zagonetka je veliko kvadratno polje podijeljeno na 81 manjih polja te devet potkvadrata od 9 polja. Cilj zagonetke je popuniti brojeve od 1 do 9 tako da se u svakom od redaka, stupaca i manjih potkvadrata broj pojavljuje točno jednom.

U drugom odjeljku diplomskog rada je opisana sudoku logička zagonetka, vrste sudoku zagonetki, pravila rješavanja zagonetke i njena povijest. U trećem dijelu je ukratko opisan C++ programski jezik i rad u microsoft visual studio okruženju. U četvrtkom dijelu je objašnjen princip rada brzog algoritma te je napravljena programska realizacija algoritma za rješavanje sudoku zagonetke. Naposlijetku je prikazana vremenska usporedba rješavanja algoritma strategije povlačenja (*backtracking algoritam*) i brzog algoritma koji je realiziran u ovom diplomskom radu.

1.1. Zadatak diplomskog rada

Zadatak diplomskog rada je napisati brzi algoritam u programskom jeziku C++ za rješavanje sudoku logičke zagonetke te usporediti vremena rješavanja tog brzog algoritma sa vremenom rješavanja algoritma strategije povlačenja (*backtracking algoritma)*.

2. SUDOKU LOGIČKA ZAGONETKA

2.1. Pregled područja teme

2013. godine dokazan je osnovni teorem o sudoku zagonetki, a to je da je sudoku NP-potpun [1]. Specijalno, problem rješavanja sudokua je NP-težak problem što znači da za njega ne postoji polinomijalno brzi algoritam.

Tokom godina, razvijeno je puno različitih algoritama za rješavanje ne samo klasične sudoku zagonetke nego i različitih vrsta zagonetki.

Najpoznatiji algoritam za rješavanje sudoku zagonetke je algoritam strategije povlačenja. Algoritam isprobava sva moguća rješenja za svako polje te ako se nađe greška u određenom polju vraća se nazad i ispravlja grešku. Ovaj algoritam je najsporiji od ostalih algoritama jer prolazi kroz sva moguća polja u zagonetci.

Postoje Stohastički algoritmi potrage / metode optimizacije (genetski algoritmi) u rješavanju zagonetke. Prema literaturi [2] vidimo da postoji puno stohastičkih optimizacija za rješavanja sudoku zagonetki i sve što su sudoku zagonetke složenije stohastička optimizacija puno brže rješava zadani problem od backtracking algoritma.

Također postoje algoritmi sa dodatnim strategijama u svrhu bržeg rješavanja kao što su: Hidden Singles, Hidden Pairs, Box/Line Reduction, XY-Chain. [3]

2.2. Uvod u sudoku

Sudoku je matematička logička zagonetka u obliku kvadratnog polja. Klasična sudoku zagonetka je kvadratno polje podijeljeno na 81 manjih polja te devet potkvadrata koji su podijeljeni na 9 polja. Položaj nekih brojeva u polju je unaprijed zadan, a težina zagonetke ovisi o broju i položaju zadanih brojeva. Cilj zagonetke je popunjavanje polja brojevima od 1 do 9 tako da se u svakom od redaka, stupaca i manjih potkvadrata broj pojavljuje točno jednom. Slika 2.2.1. prikazuje sudoku polje i njegovo rješenje. [4]

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

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

Slika 2.2.1. Sudoku zagonetka i njegovo rješenje

Osim klasične zagonetke postoji i više vrsta sudoku zagonetki.

- Mini sudoku – zagonetka dimenzija 6x6. Za razliku od klasične zagonetke, ovdje se koriste brojevi od 1 do 6.
- Killer sudoku – kombinacija obične zagonetke i kakura.
- Slovni sudoku – klasična zagonetka koja koristi slova umjesto brojeva.
- Hipersudoku – izgleda kao klasična zagonetka, ali ima istaknute četiri 3x3 slagalice. Broj se u svakoj od slagalica smije pojaviti točno jednom.
- Dijagonalni sudoku – brojevi se popunjavaju kao i u običnoj zagonetci, ali također moraju biti poredani od 1 do 9 dijagonalno.

Riješena sudoku zagonetke je ustvari primjer posebnog latinskog kvadrata uključujući dodatno pravilo za popunjavanje brojeva u potkvadratima.

2.3. Latinski kvadrat

Latinski kvadrat je *n* x *n* tablica s *n* različitih simbola tako da se svaki simbol pojavljuje točno jednom u svakom retku i stupcu. Zapisuje se u obliku matrice. Prvi se njima sustavno bavio Leonhard Euler. Smatrao ih je novom vrstom „magičnih kvadrata“.

Latinski kvadrat je standardan ili reduciran ako su u prvom retku i stupcu simboli poredani prirodnim redoslijedom (npr. 1, 2, 3, … ili a, b, c, …). Može se svesti na standardni oblik zamjenom određenog broja redaka i stupaca.

Što je *n* veći, tako se i broj latinskih kvadrata povećava.

n	standardni latinski kvadrati reda *n*	svi latinski kvadrati reda *n*
1	1	1
2	1	2
3	1	12
4	4	576
5	56	161 280
6	9 408	812 851 200
7	16 942 080	614 794 119 904 000
8	535281401856	108 776 032 459 082 956 800
9	377 597 570 964 258 816	5 524 751 496 156 892 842 531 225 600
10	7 580 721 483 160 132 811 489 280	9 982 437 658 213 039 871 725 064 756 920 320 000

Slika 2.3.1. Dimenzije latinskih kvadrata

2.4. Povijest sudoku zagonetke

Sudoku zagonetku je dizajnirao arhitekt Howard Grans koja je prvi put izdana u američkom časopisu Dell Magazines pod nazivom „Number place“. U Travnju 1984. godine, zagonetka se pojavljuje u časopisu Nikoli u Japanu pod nazivom „*Sūji wa dokushin ni kagiru”* što u prijevodu znači znamenke moraju biti jednoznačne. Kasnije Maki Kaji daje moderno ime zagonetke sudoku. Zagonetka je postala vrlo popularna u Japanu te ju otkriva Novozelanđanin Wayne Gould. Napisao je računalni program koji generira sudoku zagonetke i objavio u londonskom časopisu The Times 2004. godine. Ubrzo nakon toga je zavladala sudoku groznica cijelom Engleskom. Sudoku je napokon postao popularan 2005. godine u SAD-u te je postao redovno obilježje u mnogim novinama i časopisima diljem svijeta.

Prvo natjecanje u sudoku zagonetci se održalo 1.7.2005.

Slika 2.4.1. Prvo natjecanje u Sudoku

3. PROGRAMSKI JEZIK C++

3.1. Povijest programskog jezika C++

Na početku 70-tih godina se pojavio programski jezik C, koji je direktna preteča današnjeg jezika C++. To je bio prvi jezik opće namjene te je postigao neviđen uspjeh. Projektiran je tako da omogući procesorima računala izravan pristup i rad s poznatim objektima (bitovi, bajtovi, riječi, adrese i sl.). Zbog toga, kao i zbog blokovske strukture, postao je vrlo popularan za programiranje operacijskih sustava pa je i cijeli UNIX prepisan iz asemblerskog jezika u jezik C (1973. godine). Danas se jezik C koristi u svim područjima programiranja (obrada teksta, simulacija, baze podataka, procesno upravljanje i dr.).

Danski informatičar Bjarne Stroustrup 1979. godine je započeo rad na prethodniku C++ programskog jezika pod nazivom „C sa klasama“. Prije toga je radio na svom doktoratu u Computing Laboratory of Cambridge te je istraživao distribuirane sustave, granu računalne znanosti u kojoj se proučavaju modeli obrade podataka na više jedinica istodobno. Pri tome je koristio jezik Simula, koji je posjedovao neka važna svojstva koja su ga činila prikladnim za taj posao. Programski jezik Simula je posjedovao pojam klase – strukture koja objedinjava podatke i operacije nad podacima. Korištenje klasa je omogućilo da se koncepti problema koji se rješava izraze direktno pomoću jezičnih konstrukcija. Naoko idealan u teoriji, jezik Simula nije dobar u praksi: prevođenje je bilo iznimno dugotrajno, a kod se izvodio izuzetno sporo. Kada se 1979. godine zaposlio u Bell Labs u Murray Hillu, započeo je rad na onome što će kasnije postati C++. Na osnovu svog iskustva stečenog prilikom rada na doktoratu pokušao je stvoriti univerzalni jezik koji će udovoljiti današnjim zahtjevima. Pri tome je uzeo dobra svojstva niza jezika: Simula, Clu, Algol68 i Ada, a kao osnovu za sintaksu je uzeo C programski jezik, koji je već tada bio vrlo popularan i koji je uostalom bio stvoren u Bellovim laboratorijima.[5]

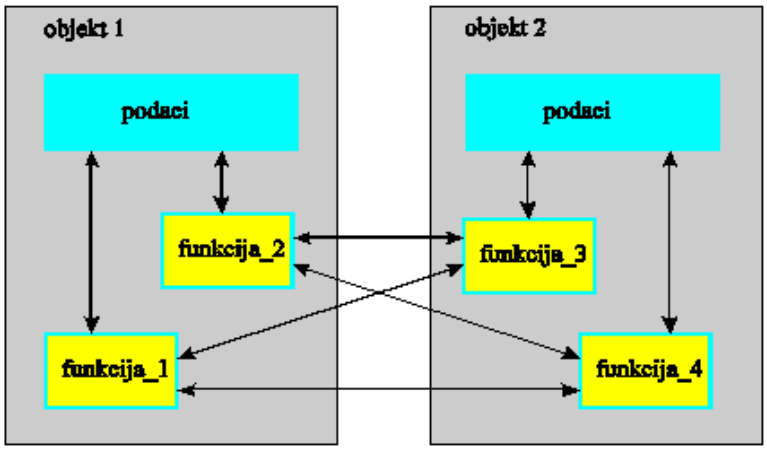
American National Standards Institute (ANSI) predložio je 1989. godine poboljšanu inačicu jezika C pod nazivom ANSI C (definiraju se tipovi funkcija, uvodi se tip *void* i dr.). Razvijen je i jezik C++ namijenjen objektnom programiranju. Jezik C++ predstavlja proširenje jezika C i koristi njegovu sintaksu uz dodavanje novih svojstava koja omogućuju objektno orijentirano programiranje (OOP). OOP je pristup organizaciji programa. Koncentracija nije usmjerena na detalje programiranja

(kodiranja), već na cijelu strukturu programa. OOP nije primjenljiv za sve probleme i programe, a naročito dolazi do izražaja u velikim programima s velikim brojem funkcija.

3.2. Osnovna svojstva programskog jezika C++

Jezik C++ je objektno orijentirani programski jezik te ga čine četiri važna svojstva:

- Enkapsulacija
- Nasljeđivanje
- Skrivanje podataka
- Polimofizam



Slika 3.2.1. Organizacija objektno orijentiranog programa

Da bi se mogao izvršiti program, funkcija *main* mora biti prisutna u svakom programu. Iz *main* funkcije se pozivaju ostale funkcije. Na početku svakog C++ programa uobičajeno je pisati pretprocesorske naredbe kojima se pozivaju biblioteke funkcija koje se rabe u programu. Opći oblik naredbe je *#include* <*ime datoteke*> i obavezno se pišu prije *main* funkcije. Da bismo mogli koristiti funkcije iz standardne biblioteke (naredbe zadužene za ulaz i izlaz podataka), program započinjemo pretprocesorskom naredbom *#include*<*iostream*>.[6]

Svaki program mora imati funkciju *main*, pretprocesorske naredbe, deklaracija funkcija i klasa koje se koriste u programu i deklaracija globalnih varijabli i konstanti. Početak funkcije se označava vitičastim zagradama '{' za početak i '}' za kraj funkcije. Na kraju naredbe se piše znak ';'.[7]

Skup znakova u programskom jeziku C++:

- Mala i velika slova abecede: a-z A-Z
- Brojevi: 0-9

- Posebni znakovi: ! ( ) „ # ' ; : < = > [ ] \ | ¸

Tipovi podataka u programskom jeziku C++:

- int
- short int
- long int
- float
- double
- long double
- char
- wchar_t
- bool
- wchar_t

Name	Description	Size	Range
char	*Char*acter or small *integer*.	1byte	signed: -128 to 127 unsigned: 0 to 255
short int(short)	Short *integer*.	2bytes	signed: -32768 to 32767 unsigned: 0 to 65535
int	*integer*.	4bytes	signed: -2147483648 to 2147483647 unsigned: 0 to 4294967295
long int (long)	Long *integer*.	4bytes	signed: -2147483648 to 2147483647 unsigned: 0 to 4294967295
bool	Boolean value. It can take one of two values: true or false.	1byte	true or false
float	*Float*ing po*int* number.	4bytes	+/- 3.4e +/- 38 (~7 digits)
double	Double precision *float*ing po*int* number.	8bytes	+/- 1.7e +/- 308 (~15 digits)
long double	Long double precision *float*ing po*int* number.	8bytes	+/- 1.7e +/- 308 (~15 digits)
wchar_t	Wide *char*acter.	2 *or* 4 bytes	1 wide *char*acter

Tablica 3.2.1. Tipovi podataka u C++

```
and          decltype
and_eq       default
             delete
alignas      double
             dynamic_cast
alignof      else
asm          enum
auto
bitand       explicit
bitor        export
bool
break        extern
case         false
catch        float
char         for
char16_t     friend
char32_t     goto
class        if
compl        inline
const        int
constexpr    long
const_cast   mutable
continue     namespace
```

```
new
noexcept
not
not_eq

nullptr
operator
or
or_eq
private
protected
public
register
reinterpret_cast
return
short
signed
sizeof
static
static_assert
static_cast
struct
```

```
switch
template
this
thread_local
throw
true
try
typedef
typeid
typename
union
unsigned
using
virtual
void
volatile
wchar_t
while
xor
xor_eq
```

Slika 3.2.2. Popis ključnih riječi u C++

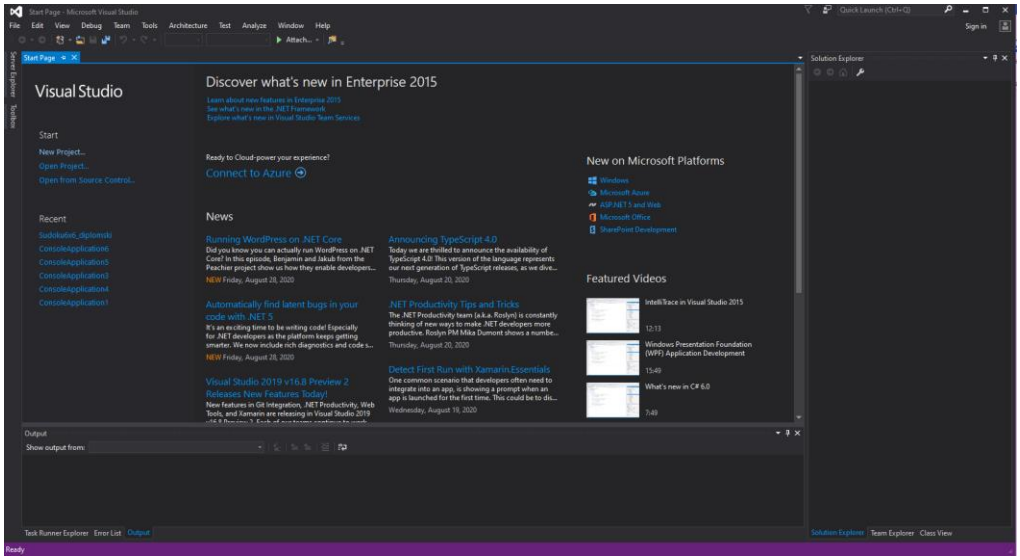
3.3 Microsoft Visual Studio okruženje

Microsoft Visual Studio je integrirano razvojno okruženje kojeg je razvila tvrtka Microsoft. Koristi se za razvoj računalnih programa, web stranica, web aplikacija, web usluga i mobilnih aplikacija. Visual Studio koristi Microsoftove platforme za razvoj softvera kao što su Windows API, Windows Forms, Windows Presentation Foundation, Windows Store i Microsoft Silverlight. Visual Studio podržava 36 različitih programskih jezika i omogućuje programeru uređivanje koda i program za ispravljanje pogrešaka u gotovo bilo kojem programskom jeziku pod uvjetom da postoji usluga specifična za taj jezik.

Ugrađeni jezici :

- C
- C++
- Visual Basic.NET
- C#
- JavaScript
- TypeScript

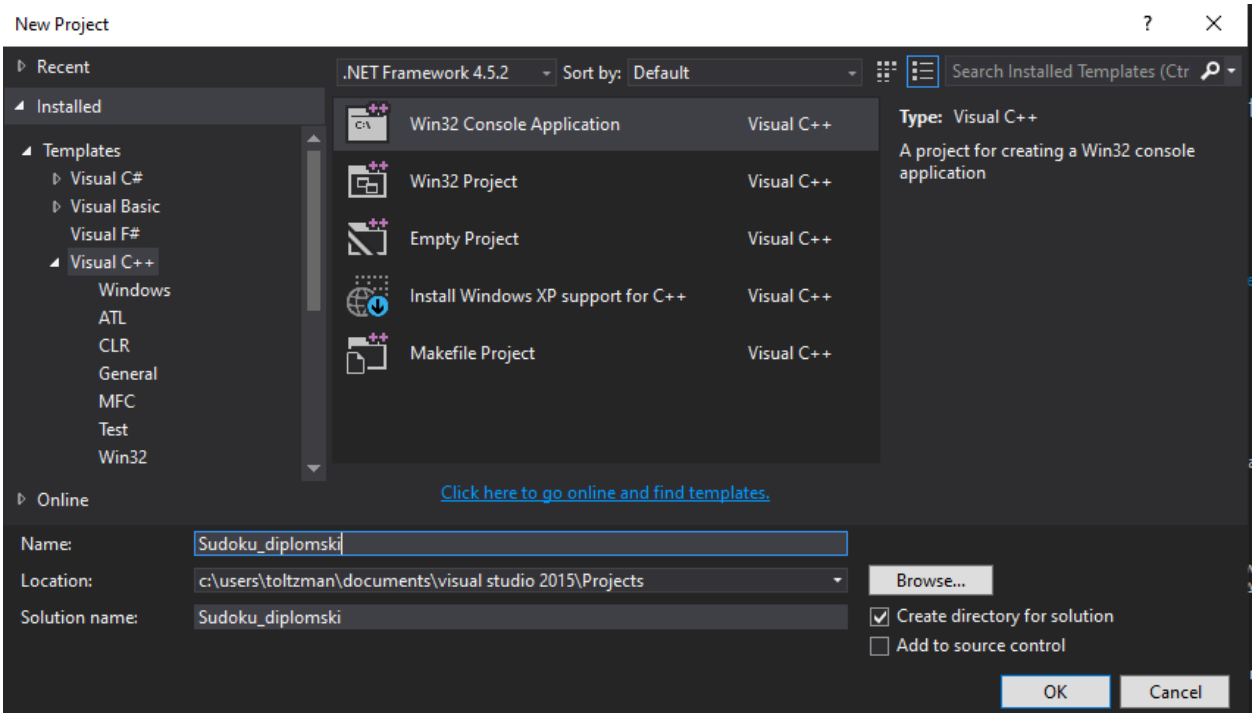
- XML
- HTML
- CSS



Slika 3.3.1. Visual Studio 2015.

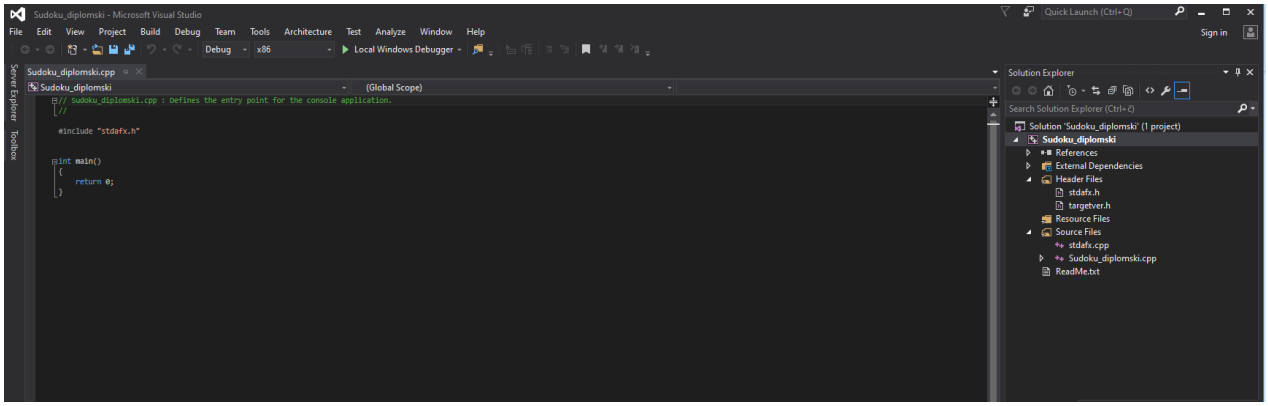
Najosnovnije izdanje Visual Studija je Community izdanje koje je besplatno i dostupno svima.

Da bi se napisao i izvršio C++ program u Visual Studiju je potrebno stvoriti novi projekt koji sadrži odgovarajuće datoteke sa izvornim kodom programa. Iz glavnog izbornika pod stavkom *File/New* se izabire kartica *Project* te *Visual C++* i zatim *Win32 Console Application.*



Slika 3.3.2. Prozor za stvaranje novog projekta

Nakon stvaranja novog projekta otvara se novi prozor u kojem se piše kod.



Slika 3.3.3. Prozor u kojemu se piše kod

Nakon napisanog programa, kod se prevodi iz jezičnog koda u strojni kod (compile) te izvodi zadanu funkciju.

4. PROGRAMSKO RJEŠENJE ALGORITMA ZA SUDOKU

4.1. Osnovne ideje na kojima se zasniva algoritam

Ideja zadatka je napisati brz algoritam i realizirati program za rješavanje sudoku zagonetke. Klasični sudoku je matematička logička zagonetka u obliku kvadratnog polja koje je podijeljeno na 81 manjih polja i 9 potkvadrata dimenzija 3x3. Kako bismo napisali algoritam za rješavanje zagonetke, moramo poznavati njena pravila rješavanja. Koriste se znamenke od 1 do 9 te se ne smije pojaviti ista znamenka u istom redu, stupcu i pod mreži. Backtracking algoritam ili algoritam strategije povlačenja je jedan od najpoznatijih algoritama za rješavanje sudoku zagonetke no vrlo je spor pri izvođenju. U ovom diplomskom radu će se optimizirati backtracking algoritam na način da ne prolazi kroz sve mogućnosti u zadanom sudoku polju te optimizacijom ubrzati njegovo izvođenje.

4.1.1 Algoritam strategije povlačenja (Backtracking algoritam)

Backtracking ili algoritam strategije povlačenja uključuje isprobavanje mogućih elemenata u jednome polju u potrazi za rješenjem, te zatim prelazak na iduće polje i tako redom do samoga kraja. Pokušava pogoditi rezultat u svakome polju isprobavajući redom sve opcije, a ako se tijekom igre ustanovi da neki element nije valjan, vraća se natrag te ga ispravlja.

Prednosti :

- Rješenje je zagarantirano (dok je slagalica valjana, ako se u nekom trenutku uspostavi da nije, odbacuje sva prethodna valjano riješena polja).
- Vrijeme rješavanja ne ovisi o težini sudoku zagonetke.
- Jednostavna implementacija.

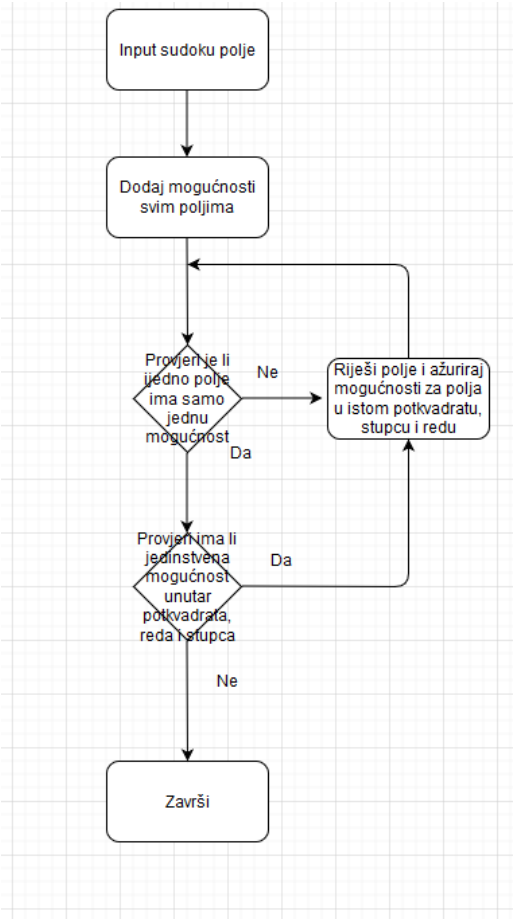
Nedostatak :

- U odnosu na ostale algoritme treba puno više vremena da dođe do rješenja.

4.1.2. Optimizirani backtracking algoritam

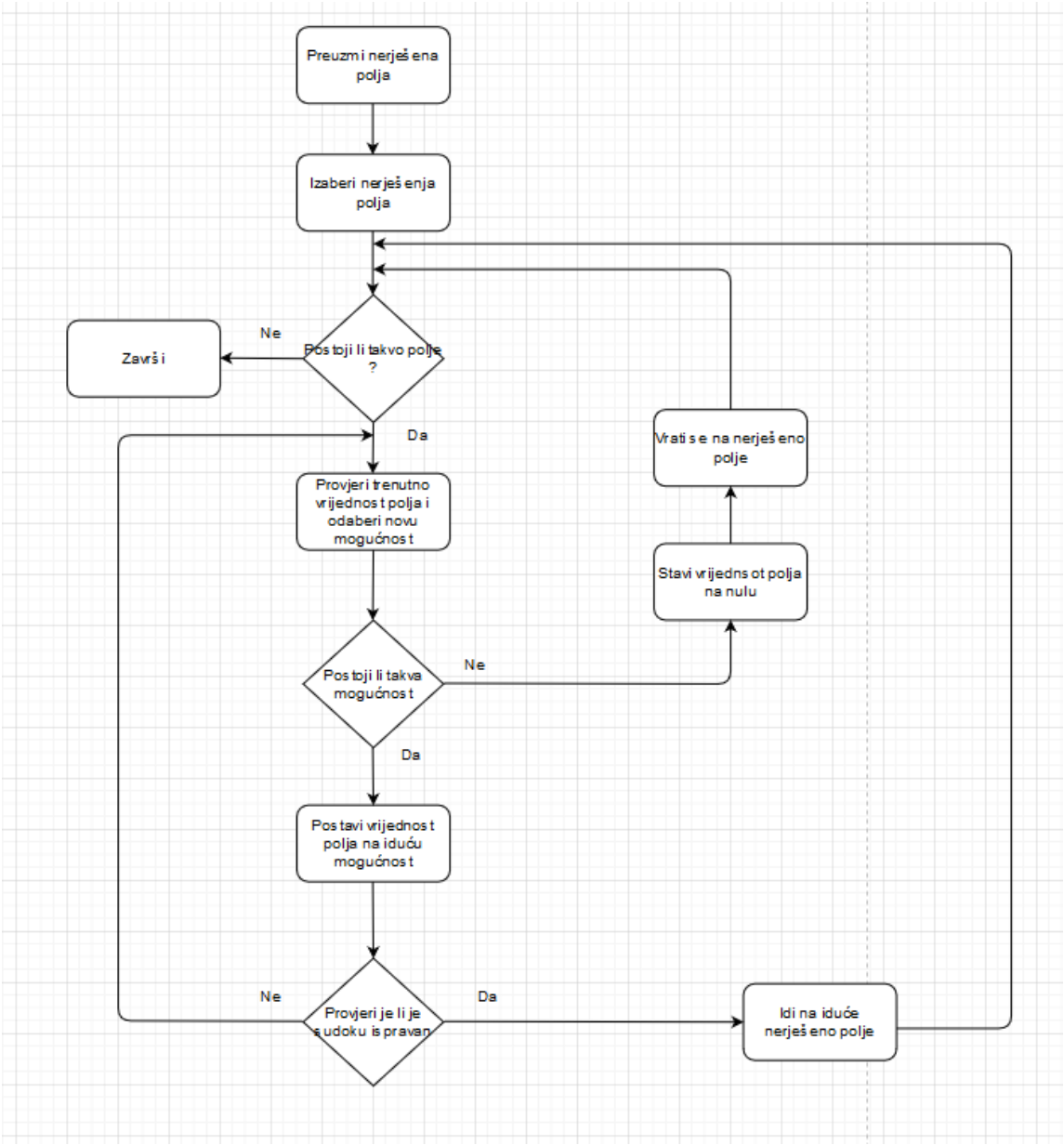
Kao što je navedeno pod 4.1.1., backtracking algoritam je vrlo spor pri izvođenju. Kako bi se algoritam ubrzao potrebno je dodati dodatnu logiku za rješavanje problema. Ideja je dodati novu logiku u algoritmu tako da se prije backtrackinga popune polja gdje jedan od brojeva sigurno može ići. Tom implementacijom se skraćuje broj mogućnosti rješenja za backtracking algoritam te se puno brže izvodi. Primijenjena su dva pravila pri rješavanju:

- Pronađi polje gdje se može samo jedan broj staviti – izabere se polje i gledaju se sve mogućnosti eliminiranjem brojeva koji su već zadani u pod mreži, redu i stupcu.
- Pronađi pod mrežu, red, stupac u kojem se broj može staviti u samo jedno polje – traži se potkvadrat, stupac ili red i provjerava može li se jedan od brojeva koji nedostaje u njemu smjestiti samo na jedno mjesto.



Slika 4.2.1. Flow dijagram prvog dijela algoritma

Nakon ispitanih i riješenih ćelija u sudoku zagonetki potrebno je dodati backtracking algoritam kako bi se sudoku zagonetka uspješno riješila.



Slika 4.2.2. Flow dijagram algoritma sa backtrackingom

4.2. Programski kod i analiza

Programski kod započinje definiranjem biblioteka (iostream, ctime, vector i stdafx.h) za korištenje ugrađenih funkcija unutar Visual studio 2015 okruženja.

4.2.1. Backtracking algoritam

Definirano je 2D polje tipa int dimenzija 9x9 koje predstavlja sudoku zagonetku. Unose se znamenke od 1 do 9, a znamenka 0 predstavlja neriješeno polje u sudoku slagalici.

Funkcija 'presentColumn' prima varijable 'j'(stupac) i 'num'(broj) te provjerava kroz sve stupce ako je znamenka prisutna u stupcu. Ako se znamenka nalazi unutar stupca funkcija vraća vrijednost '1' (true), inače vraća '0' (false).

```
bool presentColumn(int j, int num) {

    for (int i = 0; i < 9; i++) {

        if (sudoku[i][j] == num) {
            return true;
        }
    }
    return false;
}
```

Kod 4.2.1.1. Funkcija 'presentColumn'

Funkcija 'presentRow' prima varijable 'i'(red) i 'num' te provjerava kroz sve redove ako je znamenka prisutna u redu. Ako se znamenka nalazi unutar reda funkcija vraća vrijednost '1', inače vraća '0'.

```
bool presentRow(int i, int num) {
    for (int j = 0; j < 9; j++) {
        if (sudoku[i][j] == num) {
            return true;
        }
    }
    return false;
}
```

Kod 4.2.1.2. Funkcija 'presentRow'

Funkcija 'presentBox' prima varijable 'i_start', 'j_start' i 'num' te provjerava ako je znamenka prisutna unutar pod mreže.

```
bool presentBox(int start_i, int start_j, int num) {
    for (int i = 0; i < 3; i++) {
        for (int j = 0; j < 3; j++) {
            if (sudoku[i + start_i][j + start_j] == num) {
                return true;
            }
        }
    }
    return false;
}
```

Kod 4.2.1.3. Funkcija 'presentBox'

Zatim je napisana funkcija void funkcija 'printSudoku' koja ne vraća nikakvu vrijednost nego ispisuje sudoku tablicu. Prolazi kroz sve članove sudoku polja te ih ispisuje na ekran.

```
void printSudoku() {
    for (int i = 0; i < 9; i++) {
        for (int j = 0; j < 9; j++) {
            cout<< " " << sudoku[i][j];
        }
        cout << endl;
    }
    cout << "\n" << endl;
}
```

Kod 4.2.1.4. Funkcija 'printSudoku'

Funkcija 'emptySpace' provjerava nalazi li se 0, odnosno prazno mjesto kroz sudoku mrežu. Prolazi kroz sudoku polje te provjerava za svaki član je li je jednak 0. Ako je član u polju 0, tada funkcija vraća vrijednost '1', ako je prisutna znamenka (1-9) tada funkcija vraća vrijednost '0'.

```
bool emptySpace(int &i, int &j) {
    for (i = 0; i < 9; i++) {
        for (j = 0; j < 9; j++) {
            if (sudoku[i][j] == 0) {
                return true;
            }
        }
    }
    return false;
}
```

Kod 4.2.1.5. Funkcija 'emptySpace'

Slijedeća funkcija 'validPlace' prima varijable 'i', 'j', i 'num'. Provjerava nalazi li se ista znamenka u redu, stupcu ili pod mreži.

Funkcija 'solveSudoku' je funkcija u kojoj je implementiran backtracking algoritam. Prvo za zadani red i stupac provjerava nalazi li se prazno mjesto. Za svako prazno mjesto stavlja broj od 1 do 9 te provjerava je li se ta znamenka pojavljuje u istom redu, stupcu ili slagalici, odnosno nalazi li se na pravom mjestu. Ako se ne pojavljuje, dodjeljuje znamenku tom retku i stupcu. Nakon toga rekurzijom, odnosno funkcija poziva samu sebe te radi isto za ostale članove sudoku polja. Nakon što su svi članovi ispunjeni uz odgovarajuće uvjete, funkcija vraća vrijednost '1' odnosno ima rješenje ili vraća '0' odnosno nema rješenja.

```
bool solveSudoku() {
    int i, j;
    if (!emptySpace(i, j)) {
        return true;
    }

    for (int num = 1; num <= 9; num++) {
        if (validPlace(i, j, num)) {
            sudoku[i][j] = num;
            if (solveSudoku()) {
                return true;
            }
            sudoku[i][j] = false;
        }
    }
    return 0;
}
```

Kod 4.2.1.6. Funkcija 'solveSudoku'

4.2.2. Optimizacija algoritma

Definirana je klasa 'Cell' koja prima red, stupac i vrijednost polja. Unutar klase se inicijaliziraju početne mogućnosti za polje.

```
class Cell {
public:
    int row;
    int column;
    int box;
    int val;
    std::vector<int> possibilities;

    Cell(int row_number, int column_number, int nValue) {
        row = row_number;
        column = column_number;

        box = ((int)column / 3) + (((int)row / 3) * 3);

        val = nValue;

        if (nValue != 0) {
            possibilities.clear();
        }
        else {
            possibilities.clear();
            for (int i = 0; i < 9; i++) {
                possibilities.push_back(i + 1);
            }
        }

    }
};
```

Kod 4.2.2.1. Klasa 'Cell'

Zatim se radi nova klasa 'Solver' koja radi na principu iz poglavlja 4.1.2. u ovome zadatku.

```
class Solver {
public:
    std::vector <Cell> cells;

    Solver(vector<vector<int> >board) {
        for (int i = 0; i < board.size(); i++) {
            for (int j = 0; j < board.at(i).size(); j++) {
                cells.push_back(Cell(i, j, board.at(i).at(j)));
            }
        }

    }
```

Kod 4.2.2.2. Klasa 'Solver'

Unutar klase 'Solver' imamo funkcije za dohvaćanje polja iz redova, stupaca i potkvadrata

```
vector <Cell> getAllCellsFromBox(int boxNumber) {
    std::vector <Cell> boxCells;
    for (Cell cell : cells) {
        if (cell.box == boxNumber)
            boxCells.push_back(cell);
    }

    return boxCells;
}

vector <Cell> getAllCellsFromColumn(int columnNumber) {
    std::vector <Cell> columnCells;
    for (Cell cell : cells) {
        if (cell.column == columnNumber)
            columnCells.push_back(cell);
    }

    return columnCells;
}

vector <Cell> getAllCellsFromRow(int rowNumber) {
    std::vector <Cell> rowCells;
    for (Cell cell : cells) {
        if (cell.row == rowNumber)
            rowCells.push_back(cell);
    }

    return rowCells;
}
```

Kod 4.2.2.3. dohvaćanje polja iz redova, stupaca i potkvadrata

Funkcija 'getConflictingCells' dohvaća sva polja unutar pod mreže, stupca i reda za zadanu ćeliju.

```
vector <Cell> getConflictingCells(Cell _cell) {
    std::vector <Cell> conflictingCells;
    for (Cell cell : cells) {
        if (cell.row == _cell.row || cell.column == _cell.column || cell.box == _cell.box)
            conflictingCells.push_back(cell);
    }

    return conflictingCells;
}
```

Kod 4.2.2.4. 'getConflictingCells'

Funkcija 'removeCellValueFromOthersPossibilities' briše nevažeće mogućnosti iz polja.

```
void removeCellValueFromOthersPossibilities(Cell _cell) {
    for (Cell &cell : cells) {
        if (cell.row == _cell.row || cell.column == _cell.column || cell.box == _cell.box) {
            std::vector<int>::iterator position = std::find(cell.possibilities.begin(), cell.possibilities.end(), _cell.val);
            if (position != cell.possibilities.end()) {
                cell.possibilities.erase(position);
            }
        }
    }

}
```

Kod 4.2.2.5. 'removeCellValueFromOthersPossibilities'

Funkcije 'findAndSolveCellWithOnePossibility' pronalazi i rješava polje u kojem je jedna mogućnost za rješenje te funkcija 'ifUniquePossibilityWithinCells' provjerava ako postoji jedinstvena mogućnost rješenja u grupi polja.

```
bool findAndSolveCellWithOnePossibility() {
    for (Cell &cell : cells) {
        if (cell.possibilities.size() == 1) {
            solveCell(cell, cell.possibilities.at(0));
            return true;
        }
    }
    return false;
}

bool ifUniquePossibilityWithinCells(std::vector<Cell> cellGroup) {
    std::vector<int> counts(10, 0);
    for (Cell cell : cellGroup) {
        for (int possibility : cell.possibilities) {
            counts.at(possibility) += 1;
        }
    }

    std::vector<int>::iterator position = std::find(counts.begin(), counts.end(), 1);
    if (position != counts.end()) {

        for (Cell &cell : cellGroup) {
            int uniqueValue = position - counts.begin();
            std::vector<int>::iterator possPosition = std::find(cell.possibilities.begin(), cell.possibilities.end(), uniqueValue);
            if (possPosition != cell.possibilities.end()) {
                //potrebna for petlja za referencu
                for (Cell &_cell : cells) {
                    if (_cell.column == cell.column && _cell.row == cell.row) {
                        solveCell(_cell, uniqueValue);
                        return true;
                    }
                }

            }
        }
    }
    return false;
```

Kod 4.2.2.6 ‘findAndSolveCellWithOnePossibility’, ‘ifUniquePossibilityWithinCells’

Poslije funkcije ‘ifUniquePossibilityWithinCells’ je definirana funkcija 'findAndSolveUniquePossibilities’ koja traži jedinstveno rješenje za sve pod mreže, redove i stupce.

Na kraju klase 'Solver' je definirana funkcija 'run' koja pokreće algoritam.

```
    bool findAndSolveUniquePossibilities() {
        for (int i = 0; i <= 8; i++) {
            if (ifUniquePossibilityWithinCells(getAllCellsFromBox(i)))
                return true;
            if (ifUniquePossibilityWithinCells(getAllCellsFromColumn(i)))
                return true;
            if (ifUniquePossibilityWithinCells(getAllCellsFromRow(i)))
                return true;
        }

        return false;
    }

    void run() {
        while (true) {
            if (!findAndSolveCellWithOnePossibility()) {
                if (!findAndSolveUniquePossibilities()) {
                    break;
                }
            }

        }
    }
};
```

Kod 4.2.2.7. 'findAndSolveUniquePossibilities’, 'run'

Nakon što je sve definirano u klasi Solver pokreće se u mainu. Klasa solver dodaje za svako neispunjeno polje mogućnosti od 1 do 9. Za neispunjeno polje gledaju se vrijednosti polja (ispunjena

polja) iz istog reda, stupca i potkvadrata te oduzima tu vrijednost s mogućnostima tog neispunjenog polja. Nakon što algoritam prođe kroz sva neispunjena polja, ona polja kojima je ostala jedinstvena mogućnost se ispunjavaju. Ona polja koja nisu ispunjena se predaju backtracking algoritmu.

```
{9, 0, 7, 6, 0, 0, 2, 0, 0},
{0, 8, 0, 2, 0, 7, 0, 9, 6},
{6, 0, 2, 0, 0, 0, 5, 0, 7},
{0, 7, 0, 0, 6, 0, 0, 0, 0},
{0, 0, 0, 9, 0, 1, 0, 6, 0},
{0, 6, 0, 0, 2, 0, 0, 4, 0},
{0, 0, 5, 0, 0, 0, 6, 0, 3},
{0, 9, 0, 4, 0, 6, 0, 7, 0},
{0, 0, 6, 0, 0, 0, 0, 0, 0}
```

Slika 4.2.2.1. Riješena polja prije bactrackinga za Sudoku 2 (poglavlje 4.3.)

U main programu je dodana i vremenska funkcija koja prati vremensko rješavanje oba algoritma.

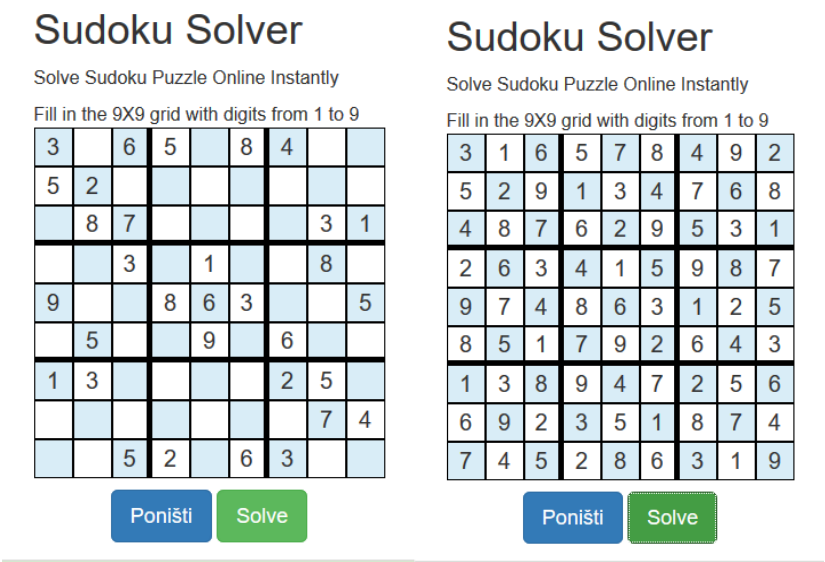
4.3 Rad programskog koda i rješenja

Program je testiran na dvije sudoku zagonetke. Kao rezultat na programu se ispisuje riješena sudoku zagonetka i vremena koja su bila potrebna algoritmima za rješavanje. Za provjeru ispravnosti programa koristi se web aplikacija za rješavanje sudoku zagonetke na stranici: https://anysudokusolver.com/ [8] .

Konfiguracija na kojoj se testira program:

- CPU – Intel i5 7400 (quad core) 3.5 GHz
- RAM memorija – 32 GB DDR4 HyperX (2666 MHz)
- GPU – Nvidia Geforce GTX 1060 6 GB VRAM

Sudoku zagonetka 1



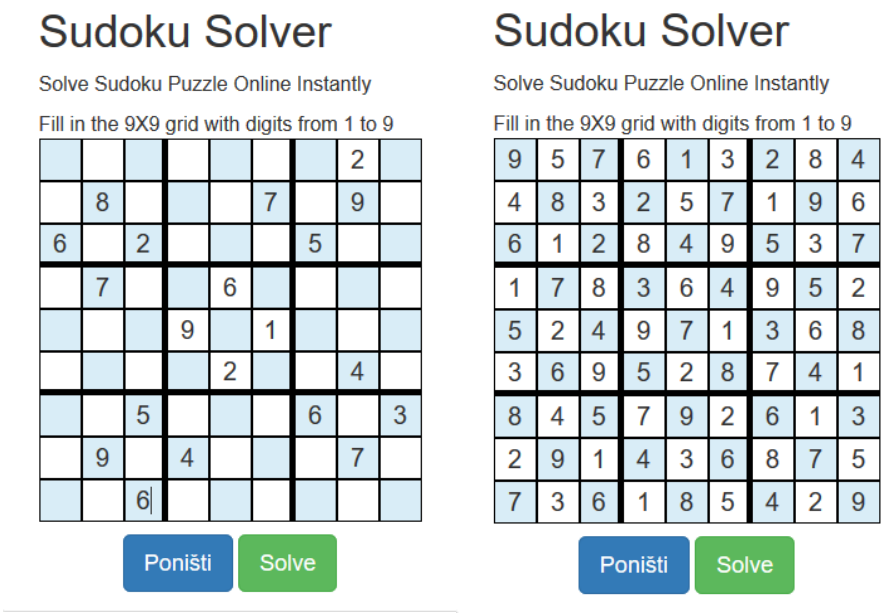
Slika 4.3.1. Sudoku 1 i njegovo rješenje online

```
        Program za rjesavanje sudoku zagonetke

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



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


Vrijeme koje je potrebno optimiziranom algoritmu: 6083 microseconds
Vrijeme potrebno pomocu backtrackinga: 23207 microseconds
Press any key to continue . . .
```

Slika 4.3.2. Sudoku 1 riješen preko programa

Iz slike 4.3.2 vidimo da su oba algoritma rješila zadanu zagonetku. Također se vidi razliku u vremenu rješavanja između dva algoritma. Optimiziranom algoritmu je potrebno 6083 mikrosekunde dok pomoću backtrackinga je potrebno 23207 mikrosekunde.

Sudoku zagonetka 2



Slika 4.3.3. Sudoku 2 i njegovo rješenje online

```
        Program za rjesavanje sudoku zagonetke

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




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


Vrijeme koje je potrebno optimiziranom algoritmu: 35007 microseconds
Vrijeme potrebno pomocu backtrackinga: 98642 microseconds
Press any key to continue . . .
```

Slika 4.3.4 Sudoku 2 riješen preko programa

Iz slike 4.3.4 vidimo da su oba algoritma riješila zadanu zagonetku. Optimiziranom algoritmu je bilo potrebno 35007 mikrosekundi dok je backtracking algoritmu trebalo 97950 mikrosekundi.

Za dane rezultate vidimo da je optimizacija backtracking algoritma uspješno provedena.

ZAKLJUČAK

U ovom radu je predložen brzi algoritam i izrađen program u programskom jeziku C++ za rješavanje sudoku logičke zagonetke.

U prvom dijelu rada opisana je sudoku logička zagonetka, njena pravila rješavanja i povijest sudoku zagonetke. Prikazane su različite vrste sudoku zagonetki i objašnjen je latinski kvadrat. Nadalje je ukratko opisan C++ programski jezik, sintaksa i njegova povijest. Pojašnjen je rad u Microsoft Visual Studiju 2015 okruženju. Zatim je implementiran Backtracking algoritam i njegova optimizacija kako bi smo pratili vrijeme rješavanja sudoku zagonetke. Iz rezultata vidimo kako uz optimizaciju postižemo puno bolji rezultat rješavanja zagonetke. Koristeći backtrack algoritam za veće sudoku zagonetke bi mogao trajati i danima ovisno o težini zadane zagonetke. Zato se rade optimizacije algoritama kako bi se vremensko rješenje smanjilo.

SAŽETAK

Cilj ovog rada je napisati algoritam u programskom jeziku C++ za rješavanje sudoku logičke zagonetke. Prvo je definirana sudoku logička zagonetka i njene različite vrste. Objašnjena su pravila rješavanja zagonetke i napravljen sažetak o povijesti sudoku zagonetci. Nakon toga, opisana je struktura i sintaksa programskog jezika C++ te pojašnjen rad u Microsoft Visual Studio okruženju. Napravljen je kratak sadržaj o povijesti programskog jezika C++. Objašnjene su ideje za optimiziranje backtracking algoritma uz dane flow dijagrame. Na kraju je opisan programski dio rada te rad optimiziranog algoritma uz prikazane rezultate riješenih sudoku zagonetki.

Ključne riječi : sudoku, zagonetka, backtracking, optimizacija programski jezik C++, Visual Studio, algoritam, program

SUMMARY

The main goal of this paper is to write algorithm in programming language C++ for solving sudoku logic puzzle 6x6. Firstly, sudoku logic puzzle is defined and different types of sudoku logic puzzles were introduced. The rules of solving sudoku are explained and a summary of the history of sudoku puzzle is made. Furthermore, the structure and syntax of the C++ programming language are described and the work in the Microsoft Visual Studio is explained. A brief summary on the history of the C++ programming language was made. Ideas for optimizing the backtracking algorithm with a flowchart diagram are explained. In the end, the program realization and the work of the optimized algorithm were described, along with the results of solved sudoku puzzles.

Key words : sudoku, puzzle, backtracking, optimization, programming language C++, Visual Studio, algorithm, computer program

ŽIVOTOPIS

Tomislav Knežević rođen je 2. prosinca 1994. Godine u gradu Nova Gradiška, Hrvatska. U Novoj Gradiški je pohađao Osnovnu školu Ljudevita Gaja, nakon koje je upisao Gimnaziju Nova Gradiška, općeg smjera. 2013. godine upisuje se na Sveučilište Josipa Jurja Strossmayera u Osijeku, fakultet elektrotehnike, računarstva i informacijskih tehnologija kao redovan student. 2017. godine završava preddiplomski studij računarstva te upisuje diplomski studij procesnog računarstva.

Potpis: _____

LITERATURA

[1] Takayuki Yato, Takahiro SETA. Complexity and Completeness of Finding Another Solution and Its Application to Puzzles. Imai.is.su-tokyo.ac.jp Tokyo, 2013. dostupno na http://www-imai.is.s.u-tokyo.ac.jp/~yato/data2/SIGAL87-2.pdf [datum zadnje posjete stranici: 28.09.2020.]

[2] Stohastic optimization approaches for solving sudoku, Meir Perez, Tshilidzi, dostupno na https://arxiv.org/ftp/arxiv/papers/0805/0805.0697.pdf [datum zadnje posjete stranici: 27.09.2020.]

[3] https://en.wikipedia.org/wiki/Sudoku_solving_algorithms [datum zadnje posjete stranici: 27.09.2020.]

[4] https://hr.wikipedia.org/wiki/Sudoku [datum zadnje posjete stranici: 27.09.2020.]

[5] Demisificirani C++, Boris Motik, Julijan Šribar, Zagreb, 2014., dostupno na https://element.hr/proizvod/demistificirani-c-4/ [datum zadnje posjete stranici: 27.09.2020.]

[6] https://hr.wikipedia.org/wiki/C%2B%2B [datum zadnje posjete stranici: 27.09.2020.]

[7] http://docbook.rasip.fer.hr/ddb/res/1/3.html#3.1 [datum zadnje posjete stranici: 27.09.2020.]

[8] https://anysudokusolver.com/ [datum zadnje posjete stranici: 27.09.2020.]