UNIVERZITET U NOVOM SADU
FAKULTET TEHNIČKIH NAUKA

UNIVERZITET U NOVOM SADU
FAKULTET TEHNIČKIH NAUKA
NOVI SAD
Departman za računarstvo i automatiku
Odsek za računarsku tehniku i računarske komunikacije

ISPITNI RAD

Kandidat: Lazar Nagulov
Broj indeksa: SV61/2022

Predmet: Objektno orijentisano programiranje 2
Tema rada: Sudoku

Mentor rada: dr Miodrag Đukić

Novi Sad, decembar, 2023.

Sadržaj

Spisak slika

Spisak tabela

1 Uvod

1.1 Sudoku

Sudoku je logička zagonetka najčešće u obliku 9×9 tabele (matrice). U prazna polja tabele se upisuju cifre, tako da se svaka broj mora pojaviti tačno jednom u svakom redu, svakoj koloni i svakoj 3×3 podmatrici (bloku).

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

Slika 1: Primer sudoku zagonetke

Zagonetka ne mora da ima jedno rešenje. Primer rešenje zagonetke sa slike 1:

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

Slika 2: Primer rešenja sudoku zagonetke

1.2 Zadatak

Realizovati konzolnu aplikaciju u C++-u (C++17) koja omogućava rešavanje i generisanje sudoku zagonetki. Korisnik unosi dateteke kroz argumente komandne linije. Primer pokretanja programa:

`./sudoku` *input.txt* *output.txt*

Argumenti:

1. *input.txt* - Datoteka iz koje se čita zagonetka.
2. *output.txt* - Datoteka u koju se upisuje zagonetka.

Svaka datoteka sadrži `jednu` zagonetku, svaki red predstavlja jedan red u tabeli, i svako polje je odvojeno razmakom. Ukoliko datoteke nisu navedene ili ne postoje, korisniku se ispisuje način korišćenja programa.

Nakon uspešnog pokretanja programa, prikazuje se početni meni koji nudi opcije:

- `Generate new Sudoku puzzle` - Generisanja nove zagonetke.
- `Load Sudoku puzzle from file` - Učitavanja zagonetke.
- `Exit` - Izlaska iz igre.

Generisana zagonetka se upisuje u *output.txt* datoteku. Zagotka se učitava iz *input.txt*. Nakon generisanja ili učitavanja zagonetke, korisnik može da:

- `Import solution` - Učita rešenje.
- `Solve` - Dopusti programu da reši zagonetku.
- `Exit` - Izlađe iz igre.

Na konzolnoj aplikaciji, posle rešenja koje je generisao program ili učitao iz datoteke, prikazuju se statistički podaci igre, uključujući broj dobro postavljenih polja, broj grešaka i brojač odigranih igara i spisak svih pronađenih grešaka. Nakon završetka igre, korisnik ima opciju da odabere ponovno igranje, što pokreće novu iteraciju igre.

2 Analiza problema

Glavni problem zadatka je mogućnost rešavanja zagonetke, jer nam je algoritam za rešavanju zagonetke neophodan za generisanje iste. Takođe je potrebna provera validnosti unete zagonetke. Prvo ćemo razmotriti probleme rešavanja zagonetke, pa onda problem validacije.

Najčešći način koji se koristi u rešavanju zagonetke je isprobavajući svaka moguća rešenja. Broj mogućnosti da se prazna polja popune eksponencijalno zavisi od broja praznih polja, pa je neophodno da razmotrimo mogućnosti optimizacije algoritma. Sam algoritam je rekurzivan, što može da dovede do prelivanja steka (eng. Stack overflow). Znači da moramo da umanjivo broj mogućih rekurzivnih poziva. Pozitivna strana je sigurno pronalaženje rešenja za svaku zagonetku (ukoliko ono postoji) i što u osnovi ne koristimo dodatne strukture podataka izuzev same tabele. Nažalost, za uvođenje bilo kakvih optimizacija neophodno je dodatno zauzeće memorije.

Za potrebe validiranja postavlja se pitanje „Šta se ubraja u greške?". Na primer, ako je broj u tabeli duplikat u koloni i u redu, koliko se tačno nalazi grešaka u tabeli? U našem slučaju, ukoliko naiđemo na broj koji je duplikat, on se uklanja iz tabele (sem ako nije bio u originalnoj zagonetci, onda se uklanja prethodni broj sa istom vrednošću) i računa kao jedna greška.

Takođe moramo razmotriti mogućnosti generisanje pseudoslučajnih brojeva za generisanje tabele. Primećujemo da korišćenjem obične random funkcije često generiše iste, ili veoma slične tabele.

3 Koncept rešenja

3.1 Algoritmi za rešavanje zagonetke

3.1.1 Obrnuta pretraga

Najčešće korišćen algoritam za rešavanje sudoku zagonetke je obrnuta pretraga (eng. Backtracking). Ovo je algoritam grube sile (eng. Brute force) koji isprobava sve moguće kombinacije. Dakle, potrebno je da se prođe kroz svako polje u tabeli. Ukoliko je polje prazno, upisujemo cifru koja u trenutnoj tabeli ispunjava sva pravila. Nakon upisivanje cifre, rekurzivno pozivamo funkciju - pokušavamo da pronađe rešenje sa novom tabelom. Ukoliko rešenje nije pronađeno, vraćamo se nazad i upisujemo drugu cifru.

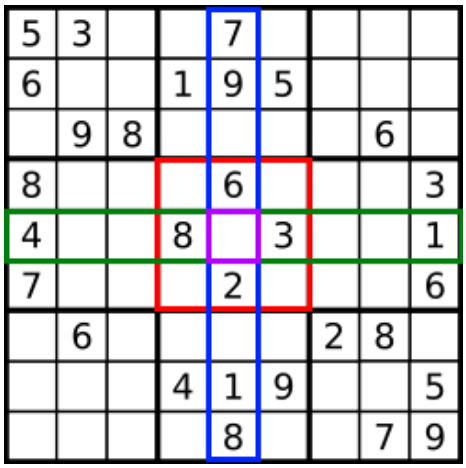
Da bi izučavanje vremenske složenosti imalo smisla, pretpostavimo da je zagonetke matrica dimenzije $n \times n$. Vremenska složenost ovog algoritma je $\mathcal{O}(n^m)$, gde je m broj polja koja trebaju da se popune.

Minimalan broj polja koja moraju biti popunjena za 9×9 zagonetku je 17, dakle, u najgorem slučaju se provera 9^{64} mogućnosti!

3.1.2 Optimizacija obrnute pretrage

Način na koji možemo optimizovati algoritam obrnute pretrage je da ubrzamo proveru da li se broj može postaviti na zadatoj poziciji. To ćemo postići tako što ćemo pamtiti koji broj se našao u redu, koloni i bloku. Za to ćemo koristiti `std::bitset` iz zaglavlja `bitset` gde, ako se za $i \in [0, 8]$ na i-toj poziciji nalazi 1, znači da se cifra $i + 1$ nalazi u redu, koloni ili bloku.

Pre samog ulaska u rekurzivnu funkciju obrnute pretrage, moramo proći kroz tabelu i zapisati svaki broj koji se nalazi u tabeli u nizove bitova. Potrebna su 3 niza bitova - za red, kolonu i blok. Za proveru da li je cifru moguće upisati




```
Polje (4,4)
RowSet:    010001101
ColSet:    111100011
BlockSet:  010100110
```

Slika 3: Primer konverzije polja u binaran broj

koristimo bitnu operaciju ili (eng. bitwise or):

```
std::bitset<> contain = rows[row] | cols[col] | blocks[block];
```

Novi niz bitova `contain` ima 0 na i-toj poziciji ako je moguće postaviti cifru $i+1$ na poziciju '(row, col)'. Na slici 3 vidimo da se u redu nalaze brojevi 1,3,4,8 (010001101), koloni 7,9,6,2,1,8 (111100011) i u bloku 6,3,2,8 (010100110). Iz ovoga zaključujemo da je u polje (4,4) moguće upisati broj 5 (111101111).

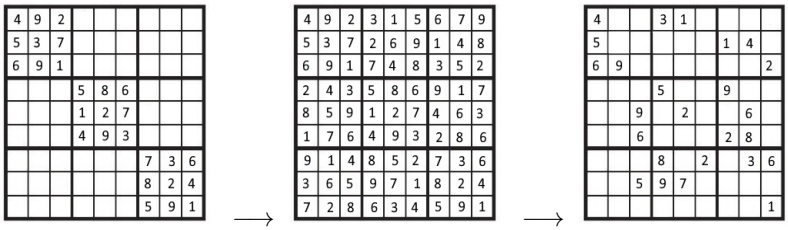
Vremenska složenost je i dalje $\mathcal{O}(n^m)$, gde je m broj polja koja trebaju da se popune, stim da je su sve provere da li se broj može upisati u polje svedene na $\mathcal{O}(m)$ za razliku od prethodnog algoritma koji ima složenost $\mathcal{O}(nm)$.

U Tabeli 1. vidimo da se prosečno vreme rešavanja duplo brže. Sam test je sproveden na 1000 tabela koja sadrže 25 popunjenih polja.

	Maksimum	Medijana	Srednja vrednost
Gruba sila	6.20147s	0.001222s	0.04691s
Optimizacija	2.14567s	0.00072s	0.02301s

Tabela 1: Optimizacija obrnute pretrage

3.2 Algoritmi za generisanje zagonetke



Slika 4: Primer generisanje zagonetke

3.2.1 Koraci u generisanju zagonetke

Generisanje zagonetke se može opisati u dva koraka:

1. Popuniti čitavu tabelu slučajnim vrednostima [1, `BOARD_SIZE`].
2. Nasumično obrisati brojeve iz tabele.

Primetimo da se blokovi na glavnoj dijagonali mogu zasebno popuniti jer ne utiču jedan na drugog. Dakle, prvi korak u generisanju se svodi na popunjavanja glavne dijagonale i rešavanja tako generisane zagonetke. Zatim brišemo brojeve iz tabele. Koliko brojava trebamo obrisati zavisi od težine zagonetke koju želimo generisati (Tabela 2).

Težina	Broj popunjenih polja
Lako	32 - 38
Srednje	26 - 31
Teško	22 - 25
Veoma teško	17 - 21

Tabela 2: Težina zagonetke u odnosu na broj popunjenih polja

3.2.2 Generisanje pseudoslučajnih brojeva

Za generisanje pseudoslučajnih brojeva koristimo Mersen Tviser algoritam (eng. Mersenne Twister algorithm). Implementacija samog algoritma se nalazi u zaglavlju `random` (`std::mt19937`). Algoritam ima ogroman period ($2^{19937} - 1$), što znači da se niz brojeva neće ponavljati. Seme (eng. seed) se nasumično određujemo koristeći `std::random_device`, a za samo generisanje brojeva u određenom intervalu koristimo `std::uniform_int_distribution`.

Ovo nam omogućava generisanje nasumičnih brojeva visokog kvaliteta.

3.3 Algoritmi za proveru validnosti zagonetke i za brojanje grešaka

Provera validnosti unete zagonetke od strane korisnika je jednostavno - proverimo da li u svakom redu, koloni ili bloku postoje duplikati. U slučaju da korisnik unosi rešenje, moramo dodatno da vodimo računa o promeni nepraznih vrednosti iz početne zagonetke.

Svakim pronalaskom pogrešnog unetog broja, izbacujemo isti iz tabele. Takođe vodimo računa da ne izbacimo broj koji je bio prisutan u početnoj tabeli i da isti brojevi nisu izmenjeni u rešenju. Pošto se kroz matricu kreće sa leva na desno, postoje dve mogućnosti:

1. Naišli smo na broj koji je bio u prethodnoj tabeli i nije validan. Znamo da smo pre njega našli drugi broj koji nije validan zbog njega. Brišemo drugi broj.

2. Naišli smo na broj koji nije bio u prethodnoj tabeli i nije validan, brišmo ga.

Dakle, pored informacije da li se broj pojavio u redu, koloni ili bloku, moramo da znamo i gde se tačno pojavio. Mana ovoga algoritma je favorizovanje brojeva na koje se prvo naiđe.

4 Opis rešenja

4.1 Modul glavnog programa (Main)

4.1.1 Funkcija main

```
int main(int argc, char** argv);
```

Glavna funckija programa. Proverava validnost datoteka prosleđenih preko argumenata komandne linije. Kreira objekat tipa Sudoku i prosleđuje mu imena datoteka.

4.1.2 Funkcija za pomoć prilikom pokretanja (Usage)

```
void Usage();
```

Ispisuje način pokretanja programa u slučaju da potrebni parametri nisu validni.

4.2 Modul Sudoku (Sudoku)

Glavni modul koji povezuje sve funkcionlanosti iz modula tabele u jednu celinu.

4.2.1 Članovi

`const std::string inputFile`	Naziv ulazne datoteke
`const std::string outputFile`	Naziv izlazne datoteke
`int currentRound;`	Trenutna runda
`int correctCount;`	Broj tačnih cifara
`int wrongCount;`	Broj pogrešnih cifara
`Board board;`	Sudoku tabela

4.2.2 Enumeracije

```
enum class Difficulty;
```

Određuje težinu Sudoku zagonetke na osnovu broja izbrisanih polja.
Vrednost: `EASY, MEDIUM, HARD, VERY_HARD`

4.2.3 Konstruktor

```
Sudoku(const std::string& inFile, const std::string& outFile);
```

Glavni parametrizovani konstruktor.
Parametri:

- (`std::string&`) inFile - ime datoteke iz koje će se čitati zagonetka ili njeno rešenje.
- (`std::string&`) outFile - ime datoteke u koju će se upisivati rešenje ili novo generisana zagonetka.

4.2.4 Funkcija članica za pokretanje (Run)

`void Run();`

Pokreće aplikaciju i kreira početni meni za korisnika. Nudi korisniku mogućnost da generiše ili učita zagonetku iz datoteke. Nakon generisanja ili učitavanja zagonetke, korisnik može da izabere način rešavanja.

4.2.5 Funcija članica za unos rešenja (SolvingOptions)

`void SolvingOptions();`

Nudi korisniku mogućnost da učita rešenje ili da dopusti programu da sam reši zagonetku. Poziva se iz `Run()` funkcije nakon generisanja ili učitavanja zagonetke.

4.2.6 Funkcija članica za rešavanje zagonetke (Solve)

`bool Solve();`

Rešava zagonetku koristeći algoritam obrnute pretrage. Rešenje upisuje u *output.txt* datoteku prosleđenu preko argumentana komandne linije.
Povratna vrednost:

- (`bool`) True ako je zagonetka uspešno rešena.

4.2.7 Funckija članica za proveru rešenja (CheckSolution)

`void CheckSolution();`

Učitava zagonetku iz *input.txt* datoteke prosleđene preko argumentana komandne linije i proverava koliko ima grešaka. Ispisuje sve greške koje pronađe, njihov broj i broj tačno upisanih brojeva.

4.2.8 Funkcija članica za gerenisanje zagonetke (Generate)

void Generate(Difficulty difficulty);

Generiše Sudoku zagonetku sa zadatom težinom. Upisuje generisanu tabelu u *output.txt* datoteku prosleđenu preko argumentana komandne linije.
Parametri:

- (`Sudoku::Difficulty`) difficulty - enumeracija koja označava težinu zagonetke.

4.2.9 Operator ispisa

`std::ostream& operator<<(std::ostream& out, const Sudoku& s);`

Ispisuje statistiku trenutne sudoke partije na standarni izlaz.

4.3 Modul Tabela (Board)

Modul koji sadrži pomoćne funkcije potrebne za menjanje trenutne tabele ili provere njene validnosti.

4.3.1 Konstante

`int BOARD_SIZE = 9;`	Veličina tabele.
`int BLOCK_SIZE = 3;`	Veličina bloka.
`int EMPTY = 0;`	Oznaka za prazno polje.
`char EMPTY_CHAR = '_';`	Oznaka za prazno polje prilikom ispisa.

4.3.2 Članovi

`int board[BOARD_SIZE * BOARD_SIZE];`	Niz koji predstavlja tabelu.

4.3.3 Alijas BitArray

```
using BitArray =
std::array<std::bitset<Board::BOARD_SIZE>, Board::BOARD_SIZE>;
```

Niz od `Board::BOARD_SIZE` setova bitova dužine `Board::BOARD_SIZE`.

4.3.4 Alijas PairArray

```
using PairArray =
std::array<std::pair<int, int>, BOARD_SIZE>
```

Niz od `Board::BOARD_SIZE` pozicija na tabeli.

4.3.5 Funckija članica za proveru validnosti tabele (IsValid)

```
bool IsValid() const;
```

Proverava da li trenutna tabela ispunjava sva pravila sudoka.
Povratna vrednost:

- (`bool`) True ako je trenutna tabela validna.

4.3.6 Funkcija članica za pronalaženje grešaka (CountErrors)

```
int CountErrors(const Board& original) const;
```

Prebrojava i ispisuje sve greške u tabeli.
Parametri:

- (`const Board&`) original - Originalna tabela pre rešavanja.

Povratna vrednost:

- (`int`) Broj grešaka u tabeli.

4.3.7 Funkcija članica za dobavljanje broja u tabeli (At)

```
int& At(int row, int col);
const int& At(int row, int col) const;
```

Dobavlja broj na polju '(row, col)'. Proverava granice. Baca `std::out_of_range` u slučaju da su uneti brojevi nevalidni.
Parametri:

- (`int`) row - Red u kome se polje nalazi.
- (`int`) col - Kolona u kome se polje nalazi.

Povratna vrednost:

- (`int&`) Referenca na broj u tabeli.

4.3.8 Statična funkcija članica za dobavljanja blocka (GetBlock)

`static constexpr int GetBlock(int row, int col);`
Vraća blok u kome se nalazi polje '(row, col)'.
Parametri:

- (`int`) row - Red u tabeli.
- (`int`) col - Kolona u tabeli.

Povratna vrednost:

- (`int`) Broj bloka u kome se nalazi polje (row, col)

4.3.9 Funkcija članica za generisanje brojeva na glavnoj dijagonali (GenerateDiagonal)

`void GenerateDiagonal();`
Generiše nasumično brojeve na glavnoj dijagonali.

4.3.10 Funcija članica za generisanje ostalih brojeva (GenerateOther)

`bool GenerateOther(int row, int col);`
Rekurzivno generiše nasumično brojeve koji se ne nalaze na glavnoj dijagonali.
Parametri:

- (`int`) row - Početan red (uglavnom 0).
- (`int`) col - Početna kolona (uglavnom 0).

Povratna vrednost:

- (`bool`) Zaustavlja rekurzivno generisanje tabele.

4.3.11 Funkcija članica za brisanje brojeva iz tabele (RemoveNumber)

`void RemoveNumber(int count);`
Nasumično briše *count* brojeva iz tabele.
Parametri:

- (`int`) count - broj brojeva koliko se briše iz tabele.

4.3.12 Funkcija članica za brojanje praznih polja (CountEmpty)

`int CountEmpty() const;`
Broji prazna polja u tabeli.
Povratna vrednost:

- (`int`) Broj praznih polja u tabeli.

4.3.13 Funkcija članica koja implementira algoritam obrnute pretrage (Backtrack)

`bool Backtrack(BitArray& rSet, BitArray& cSet, BitArray& bSet);`

Funkcija implementira algoritam obrnute pretrage. Rekurzivno popunjava tabelu. Poziva se iz funkcije `Solve()` nakon generisanja pomoćnih nizova bitova.
Parametri:

- (`BitArray`) rSet - Pomoćni niz koji prati pojavljivanje brojeva u redovima.
- (`BitArray`) cSet - Pomoćni niz koji prati pojavljivanje brojeva u kolonama.
- (`BitArray`) bSet - Pomoćni niz koji prati pojavljivanje brojeva u blokovima.

Povratna vrednost:

- (`bool`) Zaustavlja rekurziju kada pronađe rešenje.

4.3.14 Funkcija članica za pronalaženje prvog praznog polja (FindEmpty)

`bool FindEmpty(int& row, int& col) const;`

Pronalazi prvo prazno polje od pozicije '(row, col)'. Prazno polje se nalazi u *row* i *col* promenljivi nakon završetka funkcije.
Parametri:

- (`int&`) row - referenca na početan red koji se pretražuje.
- (`int&`) col - referenca na početnu kolonu koja se pretražuje.

Povratna vrednost:

- True ako postoji prazno polje.

4.3.15 Funckija članica za brisanje tabele (Clear)

`void Clear();`

Postavlja sva polja u tabeli na 0.

4.3.16 Funkcija za proveru da li je broj duplikat (IsDuplicate)

`bool IsDuplicate(int row, int col, PairArray& buff);`

Parametri:

- (`int`) row - Red u kome se polje nalazi.
- (`int`) col - Kolona u kome se polje nalazi.
- (`PairArray`) buff - Niz koji čuva pozicije prethodno pronađenih brojeva.

Povratna vrednost:

- (`bool`) True ako je broj duplikat.

4.3.17 Funkcija članica za proveru poteza (IsPossibleMove)

`bool IsPossibleMove(int row, int col, int number) const;`
Proverava da li postavka datog broja na datu poziciju je validan potez.
Parametri:

- (`int`) row - red u tabeli
- (`int`) col - kolona u tabeli
- (`int`) number - broj koji se pokušava staviti

Povratna vrednost:

- (`bool`) - true ako je moguće postaviti broj, false ako nije.

4.3.18 Funkcija članica za popunjavanje blokova (FillBlock)

`void FillBlock(int row, int col);`
Rekurzivno generiše nasumično brojeve u bloku.
Parametri:

- (`int`) row - početni red (gornje levo polje u bloku).
- (`int`) col - početna kolona (gornje levo polje u bloku).

4.3.19 Operatori upisa i ispisa

`std::istream& operator>>(std::istream& in, Board& b);`
Operator za čitanje tabele iz datoteke.
`std::ostream& operator<<(std::ostream& out, const Board& b);`
Operator za ispisivanje tabele na standarni izlaz - konzolu.
`std::ofstream& operator<<(std::ofstream& out, const Board& b);`
Operator za upisivanje tabele u datoteku.

4.3.20 Operator dobavljanja

`int& operator()(int row, int col);`
`const int& operator()(int row, int col) const;`
Operator za dobavljanje broj na poziciji '(row, col)'.

5 Testiranje

5.1 Način testiranja

Implementacija se testira pomoću modula `Test`. Napravljeni su posebni testni skupovi (skupovi testnih slučajeva) za svaki modul, a svaki testni skup sadrži testove (testne slučajeve, test cases) raznih segmenata tog modula - funkcija i struktura, kao i testiranje njihovih međusobnih odnosa.

Testovi se pokreću komandom `./sudoku -test [test_folder]`.
Opcionalni argument:

- *test_folder* - Folder u kome se nalaze testove. Podrazumeva vrednost je *../../Testovi*.

5.2 Funkcija za pokretanje

```
void TestRun(const string& path);
```

Pokreće sve testove.
Parametri:

- (`std::string&`) path - Put ka folderu koji sadrži testove.

5.3 Pomoćna funkcija za poređenje tabela

```
int CheckErrors(const std::string& org, const std::string& cmp);
```

Poredi originalnu i rešenu zagonetku kako bi prebrojao greške.
Parametri:

- (`std::string&`) org - Ime originalne datoteka gde se nalazi zagonetka.
- (`std::string&`) cmp - Ime datoteke gde se nalazi rešenje prethodne zagonetke.

Povratna vrednost:

- (`int`) Broj pronađenih grešaka.

5.4 Testni skupovi

Skup testova za verifikaciju tabele.

- *invalid_row.txt*
- *invalid_column.txt*
- *invalid_block.txt*
- *valid.txt*

Skup testova za proveru korisnikovog rešenja:

- *original_changed.txt* i *error_changed.txt*
- *original_column.txt* i *error_column.txt*
- *original_row.txt* i *error_row.txt*
- *original_block.txt* i *error_block.txt*
- *original_all_*[*i*]*.txt* i *error_all_*[*i*]*.txt*, $i \in [1, 3]$

5.5 Testiranje validnosti korisnikove zagonetke (IsValid)

5.5.1 Validna zagonetka

Test učitava validnu zagonetku.

5.5.2 Nevalidna kolona

Test učitava zagonetku sa tačno jednim duplikatom u koloni.

5.5.3 Nevalidan red

Test učitava zagonetku sa tačno jednim duplikatom u redu.

5.5.4 Nevalidan blok

Test učitava zagonetku sa tačno jedinim duplikato u bloku.

5.6 Testiranje validnosti korisnikovog rešenja (CountErrors)

5.6.1 Izmena nepraznog početnog polja

Test učitava početnu tabelu i rešenu tabelu sa izmenjenim nepraznim poljima.

5.6.2 Nevalidne kolone

Test učitava početnu tabelu i rešenu tabelu sa nevalidnim kolonama.

5.6.3 Nevalidni redovi

Test učitava početnu tabelu i rešenu tabelu sa nevalidnim redovima.

5.6.4 Nevalidni blokovi

Test učitava početnu tabelu i rešenu tabelu sa nevalidnim blokovima.

5.6.5 Sve izmene

Test učitava početnu tabelu i rešenu tabelu koja sadrži duplikat u redu, duplikat u koloni, duplikat u bloku i izmenjeno neprazno polje.

5.7 Testiranje generisanja i rešavanja tabela (Generate i Solve)

Test generiše 100 tabela i pokušava da ih reši.

6 Uočeni problemi i ograničenja

Prilikom testiranja, uočeni su sledeći problemi i ograničenja. Navedeni su i moguća rešenja.

1. Algoritam za rešavanje može da uzima (ali veoma retko) mnogo vremena za minimalno popunjene tabele (17 popunjenih polja).

2. Generisanje tabele ne garantuje jedinstvenost rešenja zagonetke.

3. Brojenje grešaka favorizuje brojeve na koje prvo naiđe (uvek će se brisati duplikat koji ima veću poziciju).

Rešenje problema jedan je da uvek generišemo tabelu koja ima više od 17 popunjenih polja. Kroz testiranje primećeno je da se tako otklanja ovaj nedostatak, odnosno da se mnogo ređe pojavljuje. Takođe je moguće optimizovati brzinu rešavanja koristeći probalističke algoritme poput Knutovog algoritma iks (eng. Knuth's Algorithm X). Primetimo da navedeni algoritam takođe koristi algoritam obrnute pretrage i rekurziju.

Rešenje problema dva je provera jedinstvenosti rešenja nakon obrisanog svakoga broja, što dodatno usporava samo generisanje tabele.

Rešenje problema tri je redefinisanje greške ili uvođenje nasumičnosti.

7 Zaključak

Napravljen je i verifikovan jedan koncept za realizaciju datog problema. Optimizovano je traženje i generisanje rešenje i date su ideje za dodatne optimizacije. Iako je vrednost veličine tabele 9, moguće je generisati i rešavati i veće tabele, izmenom konstanti `BOARD_SIZE` i `BLOCK_SIZE`, stim da važi $\texttt{BLOCK\_SIZE} \in \mathbb{N} \wedge \texttt{BLOCK\_SIZE} = \sqrt{\texttt{BOARD\_SIZE}}$, odnosno veličina tabele mora biti potpun kvadrat da bi zagonetka imala smisla.

Rezultati merenja iz Tabele 1 pokazuju znatno ubrzanje, naručito za tabele koje imaju više praznih polja. Ovo je manje primetno za tabele koje imaju manji broj praznih polja, jer je veća verovatnoća da uobičajna pretraga brzo pronađe duplikate. Sve mogućnosti za dodatne optimizacije zahtevaju dodatno zauzimanje memorije.

Cilj nam je bio da prikažemo jednostavno rešenje i primer kako se se određena optimizacija može implementirati.