Brzi algoritam za rješavanje logičke zagonetke Neboderi u programskom jeziku C++

Paradžik, Ivan

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

Sveučilišni studij

BRZI ALGORITAM ZA RJEŠAVANJE LOGIČKE ZAGONETKE NEBODERI U PROGRAMSKOM JEZIKU C

Završni rad

Ivan Paradžik

Osijek, 2020.

SADRŽAJ

1. UVOD 1

1.1. Zadatak završnog rada 1

2. LOGIČKA ZAGONETKA NEBODERI 2

2.1. Osnovne informacije 2

2.2. Pravila 2

2.3. Osnovne tehnike popunjavanja 3

2.4. Tehnika eliminacije 4

2.5. Rješavanje primjera dimenzija 5x5 4

3. PROGRAMSKI JEZIK C 7

3.1. Tipovi i operatori 7

3.2. Kontrola toka programa 9

3.3. Funkcije 12

3.4. Polja 12

3.5. Pokazivači 13

3.6. Rekurzija 14

3.7. Unatražni algoritam 14

4. PROGRAMSKO RJEŠENJE 15

4.1. Zamisao rješenja 15

4.2. Postupak rješavanja 15

4.3. Detaljan pregled funkcija 15

4.4. Tok programa 20

5. ULAZ I IZLAZ PROGRAMA 22

6. ZAKLJUČAK 26

SAŽETAK 27

ABSTRACT 28

ŽIVOTOPIS 29

LITERATURA 30

1. UVOD

Logička zagonetka Neboderi oblika je kvadratne matrice NxN. Cilj ove logičke zagonetke je popuniti cijelu matricu brojevima takvima da odgovaraju početno zadanim uvjetima. U svim retcima i stupcima moraju biti različiti brojevi. Raspon brojeva koji se unose u Neboder dimenzija NxN je od 1 do N. Problematika ovog završnog rada je osmisliti algoritam koji će za predani Neboder pronaći rješenje.

Prvo poglavlje sadrži opis zadatka ovog završnog rada.

U drugom poglavlju dane su opće informacije o logičkoj zagonetki Neboderi. Objašnjena su pravila, osnovne tehnike rješavanje te tehnika eliminacije koja je najpotrebnija pri rješavanju. Prikazan je i primjer s postepenim koracima rješavanja.

U trećem poglavlju tema je programski jezik C. On je korišten za izradu brzog algoritma za rješavanje logičke zagonetke Neboderi. Na početku je dan kratak povijesni pregled razvoja jezika. Zatim su objašnjeni tipovi podataka i operatori kao i sve korištene i potrebne značajke jezika.

U četvrtom poglavlju prikazana je ideja rješenja algoritma kao i postupak, a ujedno i njegova implementacija.

U petom poglavlju prikazani su ulaz i izlaz programa te riješeni primjeri nebodera različitih dimenzija i težina rješavanja.

Za uspješno osmišljen algoritam potrebno je povezati znanja o zagonetki sa znanjima C programskog jezika i doći do rješenja.

1.1. Zadatak završnog rada

Zadatak završnog rada je napisati algoritam u C programskom jeziku za brzo rješavanje logičke zagonetke Neboderi. Program mora korisniku ponuditi izbor dimenzija logičke zagonetke te njen unos. Nakon toga program treba riješiti uneseni neboder i ispisati rješenje.

2. LOGIČKA ZAGONETKA NEBODERI

2.1. Osnovne informacije

Logička zagonetka Neboderi kreirana je u Japanu, no nije poznato vrijeme njenog nastanka. Na scenu ostalih zagonetki predstavljena je na prvom Svjetskom prvenstvu zagonetki održanom 1992. godine u New Yorku. Logička zagonetka Neboderi izgledom podsjeća na sudoku i ostale zagonetke kvadratnog matričnog dizajna. Sastoji se od kvadratne matrice dimenzija NxN (npr. 4x4, 5x5, 6x6). Ovisno o zadanim uvjetima i broju početno smještenih nebodera u zagonetki, težina rješavanja nebodera ima tri razine: laka, srednja i teška (engl. easy, medium i hard) . [1]

2.2. Pravila

Cilj logičke zagonetke Neboderi je popuniti cijelu matricu dimenzija NxN s brojevima od 1 do N. Križaljka predstavlja dio grada, „blok“ u kojemu se nalaze jedan do drugog neboderi. Svaki broj unesen u matricu predstavlja broj katova ili visinu jednog nebodera. Dva su pravila kod rješavanja logičke zagonetke Neboderi: [1]

1. U svako polje križaljke upisuje se jedan broj koji predstavlja visinu nebodera koji se nalazi na tom mjestu. U svakom retku moraju se pojaviti svi brojevi od 1 do N. Također, i u svakom stupcu moraju se pojaviti svi brojevi od 1 do N. Svaka se visina nebodera, od 1 do N, drugim riječima, u retku i stupcu pojavljuje točno jednom, kao u logičkoj zagonetki Sudoku.
2. Brojevi koji se nalaze uz pojedini redak ili stupac izvan križaljke govore koliko slučajni prolaznik, ako je na tom mjestu i pogleda prema križaljci, vidi nebodera. (Od višeg nebodera ne vide se niži iza njega, a ako je niži ispred višeg vidjet će se oba).

		2		
1	**3**	**2**	**1**	
	2	**1**	**3**	
	1	**3**	**2**	
	3			

Slika 2.1. Riješeni primjer logičke zagonetke Neboderi dimenzija 3x3

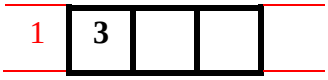
Drugo pravilo je lako shvatljivo na primjeru Slike 2.1. gdje crveno obojani broj (uvjet) označava količinu vidljivih nebodera tog stupca ili retka sa strane na kojoj se uvjet nalazi. Na Slici 2.1. postoje tri uvjeta:

1. Uvjet s brojem jedan:
   U prvom retku matrice treba se vidjeti samo jedan neboder kada se gleda s lijeve strane tog retka. Iz rješenja zagonetke vidljivo je da neboder visine 3 blokira ostale nebodere koji su niži (visine 2 i 1) te je samo on vidljiv.
2. Uvjet s brojem dva:
   U drugom stupcu gledanom s gornje strane, neboder visine 2 blokira pogled na neboder visine 1 pa se onda vidi neboder visine 3. Ukupno se vide dva nebodera te je uvjet zadovoljen.
3. Uvjet s brojem tri:
   U prvom stupcu gledanom s donje strane vidljiva su tri nebodera. Prvi neboder je visine 1 i svaki sljedeći je za jedan kat viši. Ukupno se vide tri nebodera te je zadovoljen uvjet.

2.3. Osnovne tehnike popunjavanja

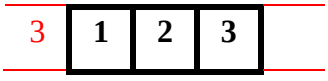
Osnovne tehnike primjenjive su na sve dimenzije nebodera, a prikazane su na primjeru nebodera 3x3:

1. Uvjet s jedinicom: ukoliko je broj 1 zadan za uvjet, do njega se stavlja neboder visine N kao što je vidljivo na slici 2.2.



Slika 2.2. Primjer retka zagonetke Neboderi s uvjetom 1

2. Uvjet s N: ukoliko je broj N zadan kao uvjet, odmah se popunjava cijeli redak/stupac kao što je prikazano na slici 2.3.

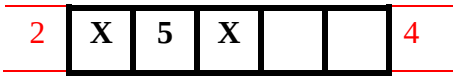


Slika 2.3. Primjer retka zagonetke Neboderi s uvjetom N

3. Ukoliko je određen smještaj jedne visine nebodera N-1 puta, preostali neboder smješta se na preostalo mjesto.

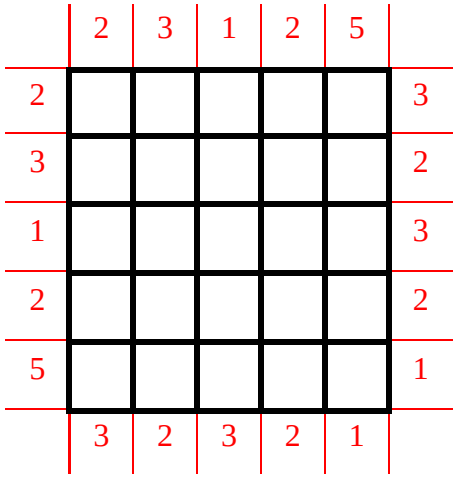
2.4. Tehnika eliminacije

Tijekom rješavanja različitih primjera nebodera najčešće se koristi metoda eliminacije potencijalnih mjesta za smještaj nebodera. Kod uvjeta nekog retka ili stupca potrebno je odrediti sva mjesta gdje je moguće smjestiti neboder neke visine. Na slici 2.4. vidljiva je mogućnost smještanja nebodera visine 4 kod uvjeta dva s lijeve strane i uvjeta četiri s desne strane nekog retka. Znak X označava mogući položaj nebodera visine četiri. Ukoliko se u stupcu gdje se nalazi X već nalazi broj 4, preostali X je mjesto na koje se smješta neboder visine 4.



Slika 2.4. Primjer retka zagonetke Neboderi sa smještenim neboderom visine 5

2.5. Rješavanje primjera dimenzija 5x5



Slika 2.5. Primjer zagonetke Neboderi dimenzija 5x5

Na početku rješavanja nebodera (Slika 2.5.) uočava se uvjet s brojem pet u petom retku i petom stupcu. Odmah se primjenjuje osnovna metoda rješavanja pa slijedi popunjavanje brojeva u tom retku i tom stupcu. U trećem retku i trećem stupcu nalazi se uvjet jednog vidljivog nebodera pa je

odmah stavljen neboder visine N (u ovom slučaju 5) do tih uvjeta. Slika 2.6. prikazuje unos brojeva na primjeru sa Slike 2.5.

	2	3	1	2	5	
2			5		1	3
3					2	2
1	5				3	3
2					4	2
5	1	2	3	4	5	1
	3	2	3	2	1	

Slika 2.6. Zagonetka popunjena primjenom osnovnih metoda rješavanja

U drugom retku broj 5 može se staviti na dva slobodna mjesta. Prvo mjesto je drugi stupac tog retka. Vidljivo je da to mjesto ne odgovara smještaju nebodera visine pet jer lijevi uvjet tog retka zahtijeva tri vidljiva nebodera. Pronađena su četiri od pet nebodera visine 5 katova. Zadnji neboder visine 5 smješten je na preostalo mjesto. Brojevi su unešeni na Slici 2.7.

	2	3	1	2	5	
2			5		1	3
3				5	2	2
1	5				3	3
2		5			4	2
5	1	2	3	4	5	1
	3	2	3	2	1	

Slika 2.7. Popunjavanje preostalih mjesta za smještaj nebodera visine 5

Gleda se gdje se može smjestiti broj 2 u prvom retku. Prvi stupac tog retka ne odgovara jer onda bi uvjet s lijeve strane bio prekršen te bi bila vidljiva tri nebodera s lijeva. Drugi stupac tog retka ne odgovara jer se neboder visine 2 već nalazi u tome stupcu. Na taj način su eliminirana dva od tri moguća mjesta za smještaj nebodera visine 2 kata te je preostalo mjesto rješenje. Nakon toga u

četvrtom stupcu nedostaju dva nebodera. Oni su visine 1 i 3 kata. Neboder visine 3 smješten je eliminiranjem trećeg retka, a na preostalo mjesto smješten je neboder visine 1 što prikazuje Slika 2.8.

	2	3	1	2	5	
2			5	2	1	3
3				5	2	2
1	5			1	3	3
2		5		3	4	2
5	1	2	3	4	5	1
	3	2	3	2	1	

Slika 2.8. Popunjavanje četvrtog stupca preostalim neboderima

Treći i četvrti redak popunjeni su pomoću eliminacije mogućih mjesta. U drugi stupac upisan je preostali neboder visine 4 te su nakon toga prvi redak i prvi stupac popunjeni pomoću metode eliminacije i zagonetka je riješena. Konačno rješenje zadanog nebodera 5x5 prikazano je na Slici 2.9.

	2	3	1	2	5	
2	4	3	5	2	1	3
3	3	1	4	5	2	2
1	5	4	2	1	3	3
2	2	5	1	3	4	2
5	1	2	3	4	5	1
	3	2	3	2	1	

Slika 2.9. Konačno rješenje zagonetke

3. PROGRAMSKI JEZIK C

Programski jezik C smatran je jednim od najznačajnijih programskih jezika na tržištu računalne industrije. Nastao je 1978. godine, a njegov tvorac je Dennis Ritchie. Prvotna namjena C programskog jezika bila je stvaranje sistemskih programa i razvoj UNIX OS-a. Programski jezik C dosta se razvijao i bivao neformalno i formalno standardiziran. Prva važnija verzija imenovana je K&R C, a ime dolazi od engleskih kratica prezimena dvaju autora najpoznatijeg C priručnika The C Programming Language Briana Kernighana i Dennisa Ritchia. Prvo izdanje te vrlo sažeto i precizno pisane knjige iz 1978. godine ujedno je standardiziralo jezik u 70-ima. Drugo izdanje iz 1988. godine opisuje ANSI C, standard kojeg je 1983. godine definirao Američki nacionalni institut za standardizaciju, a koji je i danas najbolje podržan. Krajem 2011. usvojen je ISO/IEC 9899:2011 standard, poznat kao C11, za koji su kompajleri još u razvoju. [2]

Primjena C programskog jezika danas obuhvaća mnoge segmente računalne industrije. Programeri, budući da je C jezik opće namjene, imaju slobodu programiranja raznih stvari: rješavanja zadataka, pisanja drivera, stvaranje operacijskih sustava, pisanje teksta procesora, kreiranje igara. Ipak zbog modernijih jezika, nastoji se njegovo korištenje ograničiti na ona područja u kojima je C programski jezik najbolji izbor, a ta su područja: sistemski programi na strani poslužitelja, programi prevoditelji i jezgre operativnih sustava. Potrebe za najvećom mogućom brzinom izvođenja, efikasnom kontrolom resursa i izravnom kontrolom hardvera razlog su primarnog korištenja C programskog jezika u te svrhe. [2]

3.1. Tipovi i operatori

Osnovni oblici podataka s kojima se radi u programu su varijable i konstante. Deklariranjem se stvara popis varijabli programa, određuje im se tip te eventualno njihova početna vrijednost. Operatori upravljaju varijablama. Pomoću izraza kombiniranjem varijabli i konstanti proizvode se nove vrijednosti. Tip nekog objekta određuje njegov skup vrijednosti i operacije koje se nad njim mogu izvršavati. [2]

Postoje ograničenja kod imenovanja varijabli i konstanti. Ime se može sastojati od slova i znamenki, ali prvi znak mora biti slovo. Podvlaka ("_") se uvažava kao slovo. To je korisno jer povećava čitljivost kod varijabli s dugim imenima. Ipak, ne treba imena varijabli započeti podcrtom jer rutine iz standardne biblioteke često koriste takva imena. Velika i mala slova se

razlikuju pa a i A nisu isti znak. Praksa je u C-u da se mala slova rabe za imena varijabli, a velika slova za korisnički definirane konstante. [2]

Osnovni tipovi podataka u C-u [2]:

- *char* - jedan znak iz skupa znakova, vrijednost unutar skupa [-128, 127]
- *int* - cjelobrojna vrijednost, vrijednost unutar skupa [-32768, 32767]
- *float* - realni broj jednostruke točnosti, vrijednost unutar skupa [1.2E-38, 3.4E+38]
- *double* - realni broj dvostruke točnosti, vrijednost unutar skupa [1.2E-38, 1.7E+308]

Osnovni tipovi podataka se po potrebi kombiniraju s kvantifikatorima *short*, *long*, *signed* i *unsigned*. Pomoću njih mijenja se raspon vrijednosti osnovnih tipova podataka u C-u.

Nekada je nužno tijekom pisanja programa koristiti vrijednosti koje se ne mijenjaju tijekom rada programa. U tu svrhu se koriste konstante. C poznaje cjelobrojne, realne i znakovne konstante. Konstante se mogu definirati na dva načina. Prvi način ih uvodi u program pomoću pretprocesorske naredbe *#define*. Drugi način je pomoću ključne riječi *const*. [3]

Slijedi općeniti izraz za definiranje konstante pomoću pretprocesorske naredbe *#define* i jedan primjer za konstantu imena DULJINA vrijednosti 10 (Kod 1.).

#define ime_konstante vrijednost

#define DULJINA 10

Kod 1. Opći oblik definiranja konstanti pomoću pretprocesorske naredbe *#define* i primjer

Slijedi općeniti izraz i jedan primjer za definiranje konstante pomoću ključne riječi *const*. Primjer konstante imena VISINA je podatkovnog tipa *int* vrijednosti 120 (Kod 2.).

const tip_podatka ime_konstante = vrijednost;

const int VISINA = 120;

Kod 2. Opći oblik definiranja konstanti pomoću ključne riječi *const* i primjer

U programskom jeziku C koriste se različite vrste operatora. Binarni aritmetički operatori su +, -, *, / i modul % (ostatak pri cjelobrojnom dijeljenju). Operator % se ne može koristiti nad tipovima *float* i *double*. Bitan je prioritet operatora pa tako operatori * i % imaju viši prioritet od operatora + i – koji imaju jednake prioritete. Relacijski operatori su >, >=, <=. Svi relacijski operatori imaju jednake prioritete, a prioritet relacijskih je niži od prioriteta aritmetičkih operatora. Logički operatori su &&, || i !, a predstavljaju logički i, ili i ne. C jezik podržava i operatore inkrementiranja i dekrementiranja. Oni su ++ i --, odnosno uvećavanje i umanjivanje. [2]

Također se koriste i operatori dodjeljivanja vrijednosti +, +=, -=, *=, /=, %=. Upotrebljavaju se i preostali operatori: *sizeof()* (vraća veličinu varijable u memoriji), & (vraća adresu varijable u memoriji), * (označava pokazivač) i ternarni operator *?* : (koristi se u uvjetnom izrazu). [2]

3.2. Kontrola toka programa

Budući da se naredbe u programu izvršavaju sekvencijalno, potreban je mehanizam mijenjanja toka programa. Kontrola toka programa omogućuje da se sukladno potrebama programa mijenja poredak izvršavanja naredbi. Upravo to je omogućeno pomoću naredbi koje se nazivaju naredbe kontrole toka programa. Takve naredbe su *if*, *if-else* i *switch*. Pomoću if strukture može se preskočiti određeni blok naredbi ukoliko određeni uvjet nije zadovoljen. Kod 3. prikazuje opći oblik *if* selektivne strukture. [4]

```
if(uvjet){
        naredba;
        naredba;
        ...
}
```

Kod 3. Opći oblik *if* selektivne funkcije

Pomoću *if-else* selektivne strukture možemo izvršiti prvi blok naredbi ukoliko je određeni uvjet zadovoljen ili pak izvršiti drugi blok naredbi ukoliko taj uvjet nije zadovoljen. Moguće je ugnijezditi više *if-else* naredbi. Kod 4. prikazuje opći oblik *if-else* selektivne strukture. [4]

```
if (uvjet) {
        naredba;
        naredba;
        ...
}
else {
        naredba;
        ...         }
```

Kod 4. Opći oblik *if-else* selektivne strukture

Switch kontrolna struktura predstavlja grananje koje provjerava odgovara li uvjet nekoj od konstanti cjelobrojnih vrijednosti i na osnovu toga se izvršava određena naredba. Pomoću naredbe *break* se izlazi iz *switch* strukture. Kod 5. prikazuje opći oblik *switch* naredbe. [4]

```
switch (izraz) {
        case vrijednost1 :naredba; break;
        case vrijednost2: naredba; break;
        ...
        default: naredba; break;
}
```

Kod 5. Opći oblik *switch* naredbe

Za kontrolu toka osim naredbi *if, if-else i switch* koriste se i petlje. One su *for, while* i *do-while.* Kao i naredbe kontrole toka, petlje se mogu ugnježđivati. *For* petlja se obično koristi onda kada korisnik zna koliki broj iteracija mu je potreban za određeni zadatak. Kod 6. prikazuje opći oblik *for* petlje. [5]

```
for (pocetno_stanje; uvjet; inkrement)
{
        naredba;
        naredba;
        ...
}
```

Kod 6. Opći oblik *for* petlje

Kod *while* petlje prvo se provjerava uvjet pa se tek onda izvršava blok naredbi sve dok je uvjet zadovoljen. Kod 7. prikazuje opći oblik *while* petlje.

```
while (uvjet)
{
        naredba;
        naredba;
        ...
}
```

Kod 7. Opći oblik *while* petlje

Do-while petlja se razlikuje od *while* petlje po tome što se prvo izvrši blok naredbi pa tek onda provjerava uvjet i ponavlja izvođenje se sve dok je on zadovoljen. Tok programa unutar petlji se može mijenjati pomoću naredbi *break* i *continue*. Pomoću *break* naredbe izlazi se iz aktivne petlje, a pomoću *continue* naredbe se preskaču naredbe koje slijede te se prelazi na novu iteraciju unutar petlje. [5]

3.3. Funkcije

Funkciju se definira kao programsku cjelinu koja nad određenim podacima vrši niz naredbi i vraća rezultat izvršavanja svom pozivnom programu. Pomoću funkcija složene programske zadaće se razdvajaju na manje cjeline. Time se postiže veća jasnoća koda i olakšavaju se izmjene. Svaka funkcija treba biti osmišljena da obavlja jednu dobro definiranu zadaću. Kod 8. prikazuje opći oblik funkcije:

```
tip_podatka ime_funkcije(tip_1 arg_1)
{ tijelo funkcije }
```

Kod 8. Opći oblik funkcije

Tip podatka koji je vraćen nakon izvršenja funkcije je tip_podatka. Funkcija imena ime_funkcije prima argument arg_1 koji je tipa podatka tip_1. Funkcija vraća rezultat pomoću naredbe *return* koja se nalazi na kraju tijela funkcije. Funkcije se stvaraju izvan *main()* dijela programa te se pozivaju po potrebi. Svaka funkcija ne mora nužno vraćati nekakav podatak pa je takva funkcija tipa *void*. Funkcija može primati više argumenata koji se odvajaju zarezom. Argumenti mogu biti prosljeđeni na dva načina. Prvi način je prosljeđivanje po vrijednosti. Na taj način sve operacije izvršene nad tim argumentima neće utjecati na njihove vrijednosti izvan funkcije. Drugi način je prosljeđivanje argumenata po referenci. Kad se proslijedi argument po referenci, funkcija dobiva njegovu adresu u memoriji. Sve operacije nad tim argumentom će utjecati na njegovu stvarnu vrijednost izvan funkcije. [5]

3.4. Polja

Polje je niz veličine n, a sastoji se od varijabli istog tipa podatka. Elementi polja su indeksirani cjelobrojnim vrijednostima, a pristupa im se pomoću indeksa koji su u rasponu od 0 do n – 1. Ako je vektor ime polja od 10 elemenata nekog tipa, elementi tog polja su vektor[0], vektor[1], vektor[2],..., vektor[9]. Početna vrijednost indeksa polja je uvijek 0. Kod 9. prikazuje deklaraciju polja. [5]

```
tip ime[dimenzija];
```

Kod 9. Opći oblik definiranja polja

Tip podatka je tip, a ime je ime polja te je dimenzija veličina polja. [5]

3.5. Pokazivači

Svaka varijabla ima svoju adresu u memoriji u koju je pohranjena vrijednost te varijable. Pokazivač je varijabla čija vrijednost je adresa druge varijable. Kao i svaka druga varijabla, pokazivač se definira. Kod 10. prikazuje opći oblik definicije pokazivača. [6]

```
tip *ime;
```

Kod 10. Opći oblik definiranja pokazivača

Vrijednost svih pokazivača, neovisno o tipu podatka, je heksadecimalni broj koji predstavlja adresu u memoriji. Jedina razlika između pokazivača različitih tipova podataka je tip vrijednosti na čiju adresu pokazuju. Adresa neke varijable pokazivaču predaje se pomoću operatora &. U nastavku je dan primjer definiranja pokazivača p tipa *float* i varijable var tipa *float* čija vrijednost je broj 14.01. Nakon toga je adresa varijable var predana pokazivaču p pomoću operatora & (Kod 11.). [6]

```
float *p;

float var = 14.01;

p = &var;
```

Kod 11. Primjer definiranja pokazivača i dodjele vrijednosti

Ukoliko se preko pokazivača želi pristupiti vrijednosti koja se nalazi na adresi na koju je pokazivač usmjeren to je moguće preko operatora *. U nastavku je varijabli var1 predana vrijednost koja se nalazi na adresi pokazivača p (Kod 12.). [6]

```
float var1;

var1 = *p;
```

Kod 12. Primjer dodjeljivanja vrijednosti pokazivača varijabli

Pokazivače je moguće koristiti za definiranje polja pa će takvo polje biti polje pokazivača. Moguće je definirati pokazivače koji pokazuju na druge pokazivače. Funkcije kao argumente mogu primati pokazivače, a isto tako kao povratni tip mogu imati pokazivač. [6]

3.6. Rekurzija

Proces u kojem funkcija poziva samu sebe direktno ili indirektno naziva se rekurzija, a takva funkcija je rekurzivna funkcija. Svaka rekurzivna funkcija ima svoj osnovni slučaj koji kad nastupi prekida se rekurzija. Cilj rekurzije je složeni problem rastaviti na više jednostavnih dijelova uz jedan ili više osnovnih slučajeva. Direktna rekurzivna funkcija poziva samu sebe, a indirektna poziva samu sebe preko neke druge funkcije. Kod 12. prikazuje primjer direktne rekurzivne funkcije koja računa faktorijel danog broja n. [7]

```
int fact(int n)
{
        if (n < = 1)     //osnovni slučaj
                return 1;
        else
                return n*fact(n-1);
}
```

Kod 13. Primjer rekurzije za izračun faktorijela broja n

3.7. Unatražni algoritam

Unatražni (engl. backtracking) algoritam je tehnika rješavanja problema na rekurzivan način. Do rješenja se dolazi inkrementiranjem, korak po korak, sve dok se ne dođe do rješenja problema koje zadovoljava jasno definirana ograničenja i uvjete. Unatražni algoritam će određeni zadatak inkrementalno popunjavati rješenjima koja zadovoljavaju uvjet sve dok ne dođe do trenutka kada više ni jedno rješenje ne odgovara sljedećem dijelu zadatka. Nakon toga će se vraćati i izmjenjivati ili brisati prošla rješenja sve dok ne dođe do prave kombinacije nakon koje može opet nastaviti rješavati zadatak. Ukoliko je problem rješiv unatražnim algoritmom, unatražni će algoritam proći kroz sve moguće kombinacije te će pronaći rješenje. [8]

4. PROGRAMSKO RJEŠENJE

4.1. Zamisao rješenja

Za prikaz glavne križaljke i zadanih brojeva oko nje korištena je struktura matrice (dvodimenzionalno polje). Ona sadržava sve uvjete i mjesta za smještaj nebodera pa je tako neboder dimenzija 4x4 u programskom obliku matrica dimenzija 6x6. Sukladno tome neboder dimenzija 5x5 je prikazan kao matrica 7x7 te neboder 6x6 kao matrica 8x8. Prvi i zadnji red, a tako i prvi i zadnji stupac su mjesta u kojima se nalaze uvjeti.

Budući da postoje razne dimenzije nebodera, a ujedno i neboderi različitih težina s početno zadanim neboderima, prikladan način za njihovo rješavanje je unatražnim algoritmom. Algoritam je zamišljen tako da za svako prazno mjesto gdje se treba smjestiti neboder provjerava zadovoljava li taj neboder uvjete tog retka i stupca. Algoritam će pokušavati unositi kombinacije svih brojeva sve dok ne dođe do konačnog rješenja i matrica bude popunjena.

4.2. Postupak rješavanja

Algoritam će se kretati unutrašnjim dijelom matrice, točnije dijelom za smještaj nebodera. Krenut će od prvog mjesta u matrici i kretati se pomoću inkrementa sve do prelaska u novi redak. Za svako prazno mjesto u matrici provjeravaju se uvjeti za smještaj nebodera svih visina, od 1 do N (dimenzija zagonetke).

Za prazno mjesto u matrici dva uvjeta provjeravaju nalazi li se neboder određene visine u tom retku ili stupcu. Nakon što u retku ostane jedno prazno mjesto, algoritam će koristiti treći uvjet. Treći uvjet provjerava odgovara li popunjeni redak pravilu koje određuje vidljivi broj nebodera s lijeve i desne strane toga retka. Nakon određenog broja iteracija matrica će imati stupac s jednim praznim mjestom. Za taj stupac se koristi četvrti uvjet koji provjerava odgovara li taj stupac postavljenim pravilima vidljivih nebodera s gornje i donje strane. Budući da je algoritam unatražan, pomoću navedenih uvjeta će se kretati po matrici sve dok ju ne popuni i tada završava s izvođenjem.

4.3. Detaljan pregled funkcija

Za svaku vrstu nebodera definirana je konstanta koja je korištena u njihovim pripadnim funkcijama:

- 4x4 - *#define K 6*
- 5x5 - *#define N 7*
- 6x6 - *#define M 8*

Svaka vrsta nebodera ima svojih 9 pripadnih funkcija. Razlikuju se samo po korištenoj konstanti, a funkcionalnost im je jednaka. Ukupno je 27 takvih funkcija. Preostale su dvije funkcije. Prva je *int ignorirajKut(int n, int i, int j).* Ona je korištena za lakše rukovanje četiri rubna dijela matrice budući da se ti rubovi ne koriste u logičkoj pizzlu neboderi. Preostala funkcija je glavna *main()* funkcija. Konačno, program se sastoji od 29 funkcija.

Budući da su funkcije za sve tri vrste nebodera jednake logike, sve one će biti objašnjene preko nebodera dimenzija 5x5, odnosno konstante N. Funkcije će biti objašnjene redoslijedom kojim se pojavljuju u funkciji *int rjesi5x5(int matrica[N][N])* (Slika 4.1.).

```
int rjesi5x5(int matrica[N][N]) {

    int red = 1;
    int stup = 1;

    if (!pronadiMjesto5x5(matrica, &red, &stup)){
        return 1;
    }

    for (int broj = 1; broj <= 5; broj++ ) {
        if (sigurnoUmetnuti5x5(matrica, red, stup, broj)) {
            matrica[red][stup] = broj;

            if (rjesi5x5(matrica)) {
                return 1;
            }
            matrica[red][stup] = 0;
        }
    }
    return 0;
}
```

Slika 4.1. Unatražna funkcija

Ova funkcija je glavna funkcija algoritma. Funkcija *rjesi5x5* direktno ili indirektno poziva sve ostale funkcije. Njen argument je *matrica[N][N]* koja predstavlja neboder. Početni cjelobrojni inkrementi red i stup su jednaki 1 jer tu je prvo mjesto unutrašnje matrice za smještanje brojeva.

Sljedeća pozvana funkcija je *int pronadiMjesto5x5(int matrica[N][N], int *red, int *stup)* (Slika 4.2.).

```
int pronadiMjesto5x5(int matrica[N][N], int *red, int *stup) {
    for (*red = 1; *red < N - 1; (*red)++) {
        for (*stup = 1; *stup < N - 1; (*stup)++) {
            if (matrica[*red][*stup] == 0) {
                return 1;
            }
        }
    }
    return 0;
}
```

Slika 4.2. Funkcija za pronalazak praznog mjesta i inkrementiranje

Njena uloga unutar funkcije *rjesi5x5* je pronalazak sljedećeg praznog mjesta u matrici. Argumenti su joj *matrica[N][N]* i dva *int* pokazivača red i stup. Ukoliko je pronađeno prazno mjesta, funkcija vraća 1, a u suprotnome 0. Pokazivači služe kao inkrementi za funkciju *rjesi5x5*. Tijekom potrage pomoću dvije *for* petlje i jednog *if* uvjeta inkrementi se usmjere na sljedeće prazno mjesto u matrici. Ova funkcija služi isto tako kao i opći uvjet algoritma koji će završiti rekurzivne pozive.

Povratkom u *rjesi5x5* funkciju, nakon pronalaska praznog mjesta u *matrica[N][N]* za brojeve od 1 do 5, u *for* petlji provjerava se mogući smještaj u matricu. Funkcija koja se poziva unutar *for* petlje je *int sigurnoUmetnuti5x5(int matrica[N][N], int red, int stup, int broj)* (Slika 4.3).

```
int sigurnoUmetnuti5x5(int matrica[N][N], int red, int stup, int broj) {
    return !odgovaraRedak5x5(matrica, red, broj)
           && !odgovaraStupac5x5(matrica, stup, broj);
}
```

Slika 4.3. Funkcija za sigurno umetanje broja

Pomoću funkcije *sigurnoUmetnuti5x5* za svaki *int* broj provjerava se smije li on biti smješten na prazno mjesto. Funkcija *sigurnoUmetnuti5x5* provjeru obavlja pomoću poziva funkcija *int odgovaraRedak5x5(int matrica[N][N], int red, int broj)* (Slika 4.4) i *int odgovaraStupac5x5(int matrica[N][N], int red, int broj).* Ukoliko je argument broj moguće smjestiti na prazno mjesto, funkcija vraća 1.

```
int odgovaraRedak5x5(int matrica[N][N], int red, int broj) {
    if (slobodnoMjestoRedak5x5(matrica, red)) {
        for (int stup = 1; stup < N - 1; stup++) {
            if (matrica[red][stup] == broj) {
                return 1;
            }
        }

        int redak[N];
        for (int j = 0; j < N; j++) {
            if (matrica[red][j] == 0 && (j != 0) && (j != N - 1)) {
                redak[j] = broj;
                continue;
            }
            redak[j] = matrica[red][j];
        }
        if (provjeriUvjet5x5(redak)) {
            return 0;
        }
        return 1;
    }

    for (int stup = 1; stup < N - 1; stup++) {
        if (matrica[red][stup] == broj) {
            return 1;
        }
    }
    return 0;
}
```

Slika 4.4. Funkcija koja provjerava zadovoljava li broj uvjete za smještaj u redak

Funkcija *odgovaraRedak5x5* prima matricu, trenutni redak red inkrementa i broj za koji provjerava uvjete.

U *if* bloku poziva se funkcija *int slobodnoMjestoRedak5x5(int matrica[N][N], int red)*. Ona provjerava nalazi li se u danom retku samo jedno slobodno mjesto. Ako ima više slobodnih mjesta, preskače se *if* blok. Ukoliko je preskočen *if* blok, na kraju funkcije se pomoću *for* petlje provjerava nalazi li se broj već u retku, a ako se nalazi izlaz iz funkcije je 1.

U slučaju da se izvršava *if* blok, pomoću *for* petlje provjerava se za argument *broj* nalazi li se u retku. Ako se nalazi broj u retku, izlazi se iz funkcije s 1. Sljedeći blok naredbi u *if* dijelu deklarira *int redak[N]* koji služi za spremanje trenutnog retka matrice kao jednodimenzionalno polje. U *redak[N]* sprema se trenutni redak na način da se prepišu svi brojevi, ali se umjesto 0 upisuje broj. Popunjeni *redak[N]* predaje se funkciji *int provjeriUvjet5x5(int matrica[N])* (Slika 4.5). Ta funkcija za predano jednodimenzionalno polje provjerava uvjete sa strane. Uvjeti se nalaze na

mjestima [0] i [N-1]. Ukoliko predani redak zadovoljava uvjete, izlaz if funkcije *odgovaraRedak5x5* je 0.

```
int provjeriUvjet5x5(int redak[N]) {
    int A = redak[0];
    int B = redak[N - 1];
    int brA = 1, brB = 1;
    int j;

    if (A == 0 && B == 0) {
        return 1;
    }

    if (A == 0 && B != 0) {
        j = N - 2;
        for (int i = N - 2; i > 1; i--) {
            if (redak[i - 1] > redak[j]) {
                j = i - 1;
                brB++;
            }
        }

        if (B == brB)
            return 1;
        return 0;
    }

    if (A != 0 && B == 0) {
        int j = 1;
        for (int i = 1; i < N - 2; i++) {
            if (redak[j] < redak[i + 1]) {
                brA++;
                j = i + 1;
            }
        }

        if (A == brA)
            return 1;
        return 0;
    }
    j = 1;
    for (int i = 1; i < N - 2; i++) {
        if (redak[j] < redak[i + 1]) {
            brA++;
            j = i + 1;
        }
    }

    j = N - 2;
    for (int i = N - 2; i > 1; i--) {
        if (redak[i - 1] > redak[j]) {
            j = i - 1;
            brB++;
        }
    }

    if (A == brA && B == brB)
        return 1;
    return 0;
}
```

Slika 4.5. Funkcija za provjeru uvjeta retka

Funkciji *provjeriUvjet5x5* predan je redak koji je označen kao *redak[N]*. Na početku funkcije se spremaju uvjeti retka u dvije *int* varijable, *A* i *B*. Definirani su brojači *int brA* i *int brB*. Uvjeti su brojevi od 1 do 5, a ukoliko nema zadanog uvjeta to mjesto označeno je znamenkom 0. Zbog toga su potrebna četiri slučaja uvjeta.

Prvi *if* blok označava slučaj kad oba uvjeta nisu zadana pa su sa obje strane brojevi 0. Kod tog slučaja se odmah izlazi iz funkcije budući da sve kombinacije brojeva zadovoljavaju uvjete.

Drugi slučaj je kada lijevi uvjet ne postoji pa je jednak broju 0, a uvjet s desna je zadan brojem od 1 do 5. Pomoću petlje i brojača *brB* u petlji provjerava se odgovara li trenutni popunjeni redak

matrica[N] zadanim uvjetima pa se shodno tome izlazi iz funkcije s 1 ili 0. Treći slučaj *if* bloka je isti kao i prethodni, samo što je sada uvjet s lijeve strane zadan, a s desne nije.

Ukoliko su 3 *if* uvjeta preskočena, onda je prisutan četvrti slučaj, a kod njega su oba uvjeta zadana. Pomoću brojača *brA* i *brB* i dvije petlje provjerava se zadovoljava li *matrica[N]* zadane uvjete.

Funkcija *odgovaraStupac5x5* ima istu logiku kao i funkcija *odgovaraStupac5x5* jedino što je prilagođena za stupac. Sadrži funkciju *int slobodnoMjestoRedak5x5(int matrica[N][N], int red).* Nakon povratka u funkciju *sigurnoUmetnuti5x5* funkciji *rjesi5x5* daje informaciju ukoliko se broj od 1 do 5 može umetnuti na prazno mjesto.

4.4. Tok programa

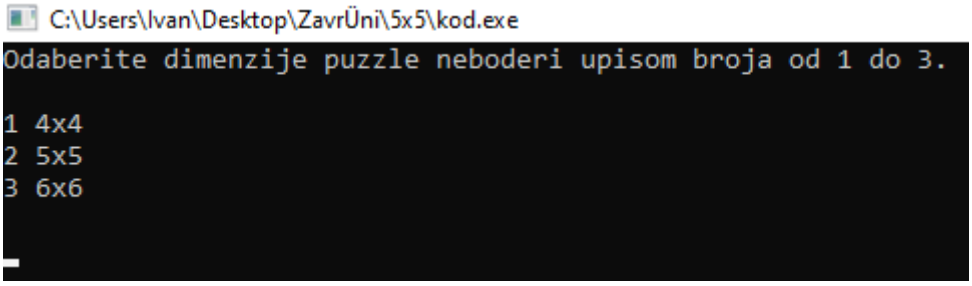
Kao i pregled funkcija, tok programa dan je na primjeru nebodera dimenzija 5x5. Tok programa sastoji se od rekurzivnih poziva funkcije *rijesi5x5*. U prvom pozivu funkcije *rjesi5x5* za predanu matricu traži se slobodno mjesto (znamenka 0) u unutrašnjoj križaljci. Tijekom prolaska kroz križaljku, inkrementi red i stupac se mijenjaju na trenutnu poziciju. Nakon pronalaska slobodnog mjesta, pokušavaju se unijeti brojevi od 1 do 5. Uvjeti za umetanje su različiti ovisno o broju popunjenih mjesta retka ili stupca.

Ukoliko je popunjeno manje od 4 mjesta nekog stupca, jedina provjera prije smještaja broja na prazno mjesto je ukoliko je dani broj sadržan u pripadnom retku ili stupcu slobodnog mjesta. Prvi broj od 1 do 5 koji nije sadržan u pripadnom retku i stupcu bit će umetnut na prazno mjesto. Provjera zadanih uvjeta nekog retka ili stupca vrši se tek kada je ostalo samo jedno nepopunjeno mjesto u retku ili stupcu. Obavlja se na način da se provjere postavljeni uvjeti retka ili stupca nakon što bi preostala znamenka bila umetnuta.

Nakon umetanja nekog broja u matricu, funkcija *rjesi5x5* prelazi u sljedeći rekurzivni poziv. U sljedećem rekurzivnom pozivu nakon pronalaska slobodnog mjesta, pokušava se umetanje brojeva od 1 do 5. Dubina rekurzije povećava se sve dok je u novom pozivu uspješno umetnut broj od 1 do 5 na prazno mjesto. Kada neki rekurzivni poziv ne smjesti broj od 1 do 5 na prazno mjesto, izvrši se povratak u prethodnu razinu rekurzije. Nakon povratka unazad, prethodno postavljeni broj vraća se na znamenku 0. Pokušavaju se umetnuti ostali brojevi te ovisno o uspješnosti odlučuje se odlazak u sljedeću ili prethodnu dubinu rekurzije.

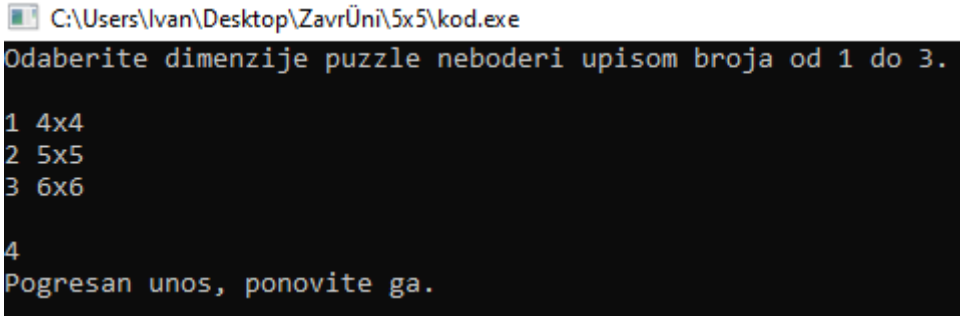
Uspješno riješena zadana matrica će u zadnjem rekurzivnom pozivu vratiti 1 zbog toga što nema preostalih slobodnih mjesta. Za matricu koja nema rješenje funkcija *rjesi5x5* će nakon povratka iz svih rekurzivnih poziva vratiti znamenku 0.

5. ULAZ I IZLAZ PROGRAMA



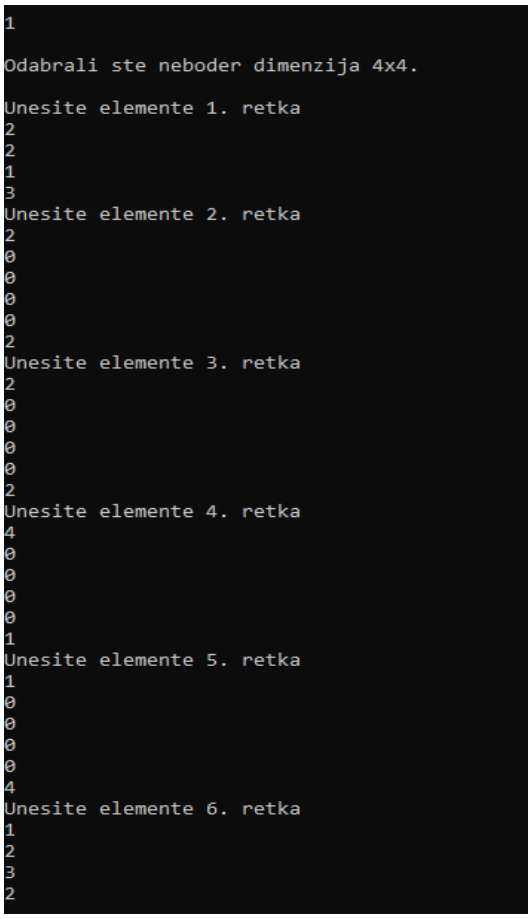
Slika 5.1. Početni prikaz programa

Na početku korisnik bira dimenzije zagonetke (Slika 5.1). Potrebno je unjeti broj od 1 do 3. Ukoliko nije unesen ispravan broj, ponavlja se unos. Za primjer krivog ulaza i izlaza u nastavku je unesen broj 4 (Slika 5.2.).



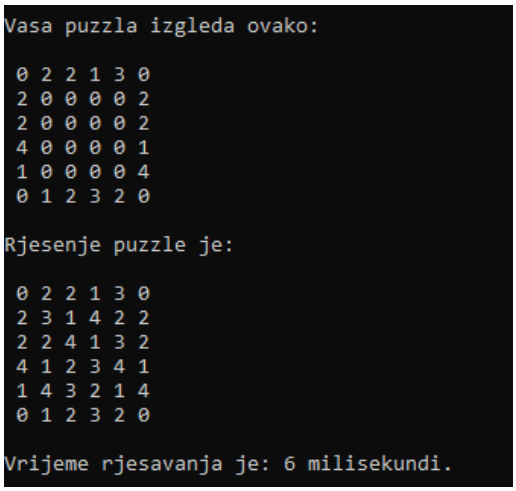
Slika 5.2. Krivi unos broja za odabir dimenzija nebodera

Ukoliko je pravilno unešen broj za odabir dimenzija nebodera, upisuju se podaci za svaki redak nebodera (Slika 5.3.).



Slika 5.3. Primjer unosa podataka o neboderu dimenzija 4x4

Nakon što je korisnik unio podatke o neboderu, ispisani su njegov izgled i rješenje. Ukoliko je uneseni neboder riješen, prikazano je i vrijeme izvođenja algoritma za rješavanje (Slika 5.4.).



Slika 5.4. Izlaz programa rješenje unešene zagonetke dimenzija 4x4 (težina easy)

Neki neboderi imaju više rješenja. Algoritam će uvijek pronaći ono rješenje do kojega je prvo došao. Ukoliko je unesen neboder koji nema rješenje, izlaz iz programa je sljedeći (Slika 5.5.).

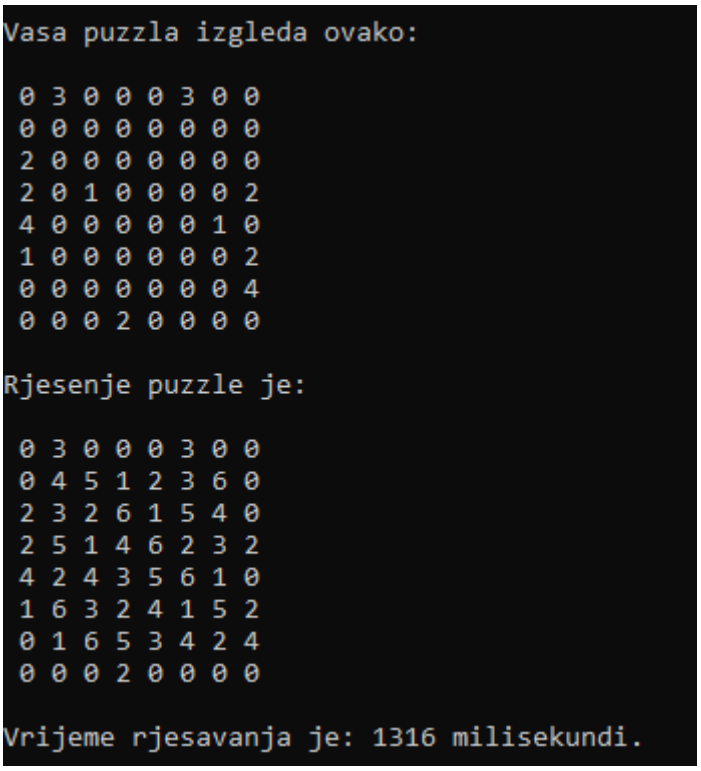
```
Vasa puzzla izgleda ovako:

 0 4 4 4 4 0
 2 0 0 0 0 0
 0 0 0 0 0 0
 0 0 0 0 0 0
 0 0 0 0 0 0
 0 0 0 0 0 0

Rjesenje puzzle je:

Nema rjesenja

Vrijeme rjesavanja je: 0 milisekundi.
```

Slika 5.5. Izlaz programa za neboder koji nema rješenje

Brzine rješavanja nebodera istih dimenzija, a različitih težina su približno jednake. Slika 5.6. prikazuje izlaz iz programa prilikom rješavanja nebodera dimenzija 5x5, a Slika 5.7. prilikom rješavanja nebodera dimenzija 6x6. Oba primjera su težina hard.

```
Vasa puzzla izgleda ovako:

 0 0 3 0 0 0 0
 0 0 0 2 0 0 0
 0 0 0 0 0 0 4
 2 0 0 0 0 0 0
 0 0 0 0 0 0 4
 3 0 0 0 0 0 0
 0 0 0 0 3 0 0

Rjesenje puzzle je:

 0 0 3 0 0 0 0
 0 3 1 2 4 5 0
 0 5 4 3 1 2 4
 2 4 2 1 5 3 0
 0 2 5 4 3 1 4
 3 1 3 5 2 4 0
 0 0 0 0 3 0 0

Vrijeme rjesavanja je: 8 milisekundi.
```

Slika 5.6. Izlaz programa rješenje unešene zagonetke dimenzija 5x5 (težina hard)



Slika 5.7. Izlaz programa rješenje unešene zagonetke dimenzija 6x6 (težina hard)

Gledajući Sliku 5.4. i Sliku 5.5. vidljivo je da su vremena rješavanja nebodera dimenzija 4x4 i 5x5 približno jednaka. Razlika je minimalna. Nasuprot tome, brzine rješavanja nebodera dimenzija 6x6, ovisno o primjeru, mogu biti od minimalno jedne sekunde pa do više sekundi. Primjer na Slici 5.7. rješavan je 1.316 sekundi.

6. ZAKLJUČAK

Cilj završnog rada bio je osmisliti i implementirati algoritam za rješavanje logičke zagonetke Neboderi. Algoritam je trebao rješavati nebodere dimenzija 4x4 i 5x5. Osmišljeni algoritam trebalo je implementirati u C programskom jeziku. Algoritam je na brz način trebao doći do rješenja.

Izgledom logička zagonetka Neboderi podsjeća na sudoku. U svakom retku moraju se pojaviti svi brojevi od 1 do N. Također, i u svakom stupcu moraju se pojaviti svi brojevi od 1 do N. Brojevi uz pojedini redak ili stupac izvan križaljke govore koliko nebodera se vidi ukoliko se s te strane gleda križaljka. (Viši neboder blokira pogled na niže, a ako je niži neboder ispred višeg vidjet će se oba). Nakon definiranja tehnika rješavanja, primijenjene su na primjeru. Najvažnija je tehnika eliminacije.

Dennis Ritchie tvorac je programskog jezika C. Priložena su sva znanja i funkcionalnosti programskog jezika C korištena pri izradi unatražnog algoritma. Najvažnije svojstvo jezika C za izradu unatražnog algoritma je svojstvo funkcije da poziva samu sebe, rekurzija.

Stvoreni algoritam je unatražne prirode. To svojstvo mu omogućuje siguran dolazak do rješenja zagonetke ukoliko ono postoji. Algoritam se kreće unutrašnjim dijelom zagonetke primjenjujući definirana pravila za smještaj nebodera. Ciljana funkcionalnost algoritma proširena je pa tako algoritam rješava i nebodere dimenzija 6x6. Algoritam uspješno rješava sve težine nebodera dimenzija 4x4, 5x5 i 6x6.

Testiranjem algoritma zaključeno je da su brzine rješavanja nebodera dimenzija 4x4 i 5x5 manje od 20 milisekundi. Brzine rješavanja nebodera 6x6 su znatno sporije i iznose više od 1 sekunde. Moguće su dorade algoritma za rješavanje nebodera dimenzija 6x6. Cilj završnoga rada je uspješno ostvaren.

SAŽETAK

Naslov: Brzi algoritam za rješavanje logičke zagonetke Neboderi u programskom jeziku C

Cilj ovog rada bio je napraviti algoritam u C programskom jeziku. Alogoritam je trebao riješiti jednu korisnički unesenu logičku zagonetku Neboderi. Takva zagonetka izgledom podsjeća na sudoku, a oblika je kvadratne matrice. Cilj ove zagonetke je popuniti matricu NxN brojevima od 1 do N na način da se u istom stupcu i istom retku ne ponavljaju isti brojevi. Tehnika eliminacije najkorištenija je metoda pri rješavanju zagonetke, a temelji se na elimininiranju potencijalnih mjesta za smještaj nebodera. Algoritam za rješavanje navedene zagonetke izrađen je u programskom jeziku C korištenjem unatražnog algoritma. Osmišljen je na način da riješava nebodere dimenzija 4x4, 5x5 i 6x6 po svim razinama težine: easy, medium i hard. Na ulazu u program korisnik odabire dimenziju nebodera čije će podatke nakon odabira dimenzije unijeti u program, a na izlazu programa prikazan je korisnički unesen neboder i njegovo konačno i točno rješenje.

Ključne riječi: C, polje, rekurzija, unatražni algoritam, zagonetka neboderi

ABSTRACT

Title: Fast solving algorithm for logic puzzle Skyscrapers in C programming langauge

The goal of this project was to make a program in C programming langauge. The program had to solve one user entered logic puzzle Skyscrapers. This kind of puzzle reminds of sudoku and its shape is of a square matrix. The purpose of this puzzle is to fill in a NxN matrix with numbers from 1 to N in a way that numbers in each column and row don't repeat themselves. The elimination technique is the most used technique when solving the puzzle and is based on eliminating the potential places where skyscraper should be placed in. The algorithm for solving this puzzle is programmed in C programming language and works as a backtracking algorithm. It is designed to solve skyscrapers puzzles of 4x4, 5x5 and 6x6 at all difficulty levels: easy, medium and hard. At program input user chooses the dimension of a skyscrapers puzzle and inputs its data afterwards. The program outputs the look of the puzzle and its final and correct solution.

Keywords: array, backtracking algorithm, C, recursion, skyscrapers puzzle

ŽIVOTOPIS

Ivan Paradžik rođen je 23. siječnja 1999. godine u Đakovu. Završio je Osnovnu školu Josipa Kozarca Semeljci. Godine 2017. završio je Gimnaziju A. G. Matoš Đakovo, smjer opća gimnazija te je iste godine upisao Fakultet elektrotehnike, računarstva i informacijskih tehnologija Osijek. Student je 3. godine preddiplomskog sveučilišnog studija računarstva.

LITERATURA

[1] The Art of Puzzles

https://www.gmpuzzles.com/blog/skyscrapers-rules-and-info/ [14.5.2020.]

[2] D. M. Ritchie, B. W. Kernighan: C Programming Language, Prentice Hall, SAD, 1998.

[3] Tutorialspoint: C – Constants and literals

https://www.tutorialspoint.com/cprogramming/c_constants.htm [16.5.2020.]

[4] N. Pavin: Kontrola toka programa (I): if i if-else kontrolne strukture, ternarni operator ?:, switch kontrolna struktura,

http://www.phy.pmf.unizg.hr/~npavin/nastava/racunarstvo/06_07/4.html [16.5.2020.]

[5] M. Jurak: Programski jezik C,

https://web.math.pmf.unizg.hr/~singer/Prog_Add/c.pdf [16.5.2020.]

[6] Tutorialspoint: C – Pointers

https://www.tutorialspoint.com/cprogramming/c_pointers.htm [17.5.2020.]

[7] GeeksforGeeks: Recursion

https://www.geeksforgeeks.org/recursion/ [18.5.2020.]

[8] GeeksforGeeks - Backtracking | Introduction

https://www.geeksforgeeks.org/backtracking-introduction/ [18.5.2020.]