Rješavanje nekih klasičnih kombinatornih problema u programskom jeziku C++

Cvjetković, Robert

Undergraduate thesis / Završni rad

2015

Degree Grantor / Ustanova koja je dodijelila akademski / stručni stupanj: **University of Rijeka, Faculty of Humanities and Social Sciences / Sveučilište u Rijeci, Filozofski fakultet u Rijeci**

Permanent link / Trajna poveznica: https://urn.nsk.hr/urn:nbn:hr:186:229225

Rights / Prava: In copyright / Zaštićeno autorskim pravom.

Download date / Datum preuzimanja: **2024-04-27**

Repository / Repozitorij:

Repository of the University of Rijeka, Faculty of Humanities and Social Sciences - FHSSRI Repository

zir.nsk.hr

UNIRI DIGITALNA KNJIŽNICA

dabar
DIGITALNI AKADEMSKI ARHIVI I REPOZITORIJI

SVEUČILIŠTE U RIJECI
FILOZOFSKI FAKULTET U RIJECI
ODSJEK ZA POLITEHNIKU

Rješavanje nekih klasičnih kombinatornih problema u programskom jeziku C++

Završni rad

Mentor završnog rada: doc. dr. sc. Marko Maliković

Student: Robert Cvjetković

Rijeka, 2015.

SVEUČILIŠTE U RIJECI
Filozofski fakultet
Odsjek za politehniku

Rijeka, Sveučilišna avenija 4

Povjerenstvo za završne i diplomske ispite

U Rijeci, 12. Lipnja, 2015.

ZADATAK ZAVRŠNOG RADA

(na sveučilišnom preddiplomskom studiju politehnike)

Pristupnik: Robert Cvjetković

Zadatak: Rješavanje nekih klasičnih kombinatornih problema u programskom jeziku C++

Rješenjem zadatka potrebno je obuhvatiti sljedeće:

1. Opis odabranih kombinatornih problema.
2. Opis algoritama za rješavanje problema.
3. Dijagrame toka programa.
4. Opis osnovnih svojstava programskog jezika C++.
5. Detaljan opis tipova podataka, izvedenih oblika podataka, naredbi i drugih elemenata iz programskog jezika C++ koji se koriste u rješenjima odabranih problema.
6. Opis rješenja koja su dobivena iz napisanih programa.
7. Cjelokupan kôd u programskom jeziku C++.

U završnom se radu obvezno treba pridržavati **Pravilnika o diplomskom radu i Uputa za izradu završnog rada sveučilišnog dodiplomskog studija**.

Zadatak uručen pristupniku: 12. lipnja 2015. godine

Rok predaje završnog rada: _____

Datum predaje završnog rada: _____

Zadatak zadao:

Doc. dr. sc. Marko Maliković

FILOZOFSKI FAKULTET U RIJECI
Odsjek za politehniku

U Rijeci, 12. lipnja 2015. godine

ZADATAK ZA ZAVRŠNI RAD

(na sveučilišnom preddiplomskom studiju politehnike)

Pristupnik: **Robert Cvjetković**

Naslov završnog rada: **Rješavanje nekih klasičnih kombinatornih problema u programskom jeziku C++**

Kratak opis zadatka: Opisati odabrane kombinatorne probleme i napisati algoritme i dijagrame toka za njihovo rješavanje. Opisati osnovna svojstva programskog jezika C++, a detaljno opisati tipove podataka, izvedene oblike podataka, naredbe i druge elemente iz programskog jezika C++ koji se koriste u rješenjima odabranih problema. Opisati rješenja koja su dobivena iz napisanih programa. Priložiti cjelokupan kôd u programskom jeziku C++.

Zadatak uručen pristupniku: **12. lipnja 2015. godine**

Ovjera prihvaćanja završnog rada od strane mentora: _____

Završni rad predan: _____

Datum obrane završnog rada: _____

Članovi ispitnog povjerenstva: 1. predsjednik - _____

2. mentor - _____

3. član - _____

Konačna ocjena: _____

Mentor

_____

Doc. dr. sc. Marko Maliković

Sažetak

Programiranje je pisanje uputa računalu kako da izvrši neku radnju. Programiranje uključuje korake poput analize problema, crtanja dijagrama toka, kreiranja algoritama, pisanja završnog programskog kôda u određenom programskom jeziku. Za potrebe ovog rada su odabrani i opisani neki klasični kombinatorni problemi, izvedeni su svi gore navedeni koraci, a završni programski kôd je pisan u programskom jeziku C++. Na početku rada su ukratko opisana osnovna svojstva programskog jezika C++, a detaljnije su opisani koncepti jezika C++ koji se koriste za potrebe ovog rada. Na kraju je prikazan cjelokupni programski kôd za rješavanje navedenih problema.

Ključne riječi:

- Kombinatorika
- Algoritam
- Programski kôd
- Dijagram toka

Kazalo

1. Uvod

Kombinatorni problemi su problemi koji spadaju u granu matematike koja se zove kombinatorika. Ona se bavi izmještanjem objekata zadanog skupa u izvjesne konfiguracije [1]. Kombinatorika istražuje metode koje daju odgovor na pitanja kao što su "Koliko ima objekata s danim svojstvom?" ili "Na koliko se načina može dogoditi izvjesni događaj?".

Jedan od poznatijih kombinatornih problema je postavljanje n kraljica na n x n ploču bez da se dvije kraljice međusobno napadaju te prikaz koliko mogućih rješenja postoji pošto je broj razmještaja, na primjer za 8 kraljica na 8 x 8 ploči, jednak 1. $78463*10^{14}$ kombinacija, a broj rješenja je 92 [4].

Kombinatorika ima veliku primjenu u matematičkoj optimizaciji, kompjuterskoj znanosti, dinamici i statistici.

Kombinatorika je doživjela nagli razvoj posljednjih 30 - 40 godina razvojem tehnologije zbog koje čovjek više nije morao ručno prolaziti sve moguće kombinacije kako bi našao odgovarajuće rješenje što bi uzelo puno vremena, nego bi sve bilo rješavano tehnologijom. Još je stroj "Victory" dešifrirao kodne riječi nacista u drugom svjetskom ratu. Naravno i dalje bi čovjek morao pronaći algoritme za rješavanje problema ali bi taj proces puno brže tekao. Već smo spomenuli da je broj kombinacija za problem 8 kraljicana 8 x 8 ploči jednak 1. $78463*10^{14}$, a broj rješenja 92. Ako bi čovjek prolazio kroz svaku kombinaciju trebalo bi jako mnogo vremena dok bi se našla sva rješenja, dok se kompjuterom ta rješenja nađu u nekoliko sekundi. Čak i da čovjek zna algoritme i pomoću njih eliminira neke slučajeve i dalje bi mu trebalo jako mnogo vremena da riješi takve probleme.

Kombinatorika ima veliku primjenu u današnjem životu jer se njome pokušavaju definirati algoritmi za rješavanje problema ili, ako takvi algoritmi već postoje, onda za smanjenje vremena za rješavanje tih problema. Jedna od velikih primjena kombinatorike u kompjuterskoj znanosti je dizajniranje efikasnih i pouzdanih metoda za komprimiranje i slanje podataka (npr. zip datoteke, mp3 i jpeg formati itd). Dosta veliku primjenu kombinatorika ima i pri radu procesora. Na primjer, kako što više bitova poslati na što manje moguće načina i s što manje logičkih uvjeta jer se na taj način smanjuje vrijeme rada i opterećenje procesora što dovodi do ekonomičnijeg rješavanja problema u kraćem vremenu.

1. 1 C++

1. 1. 1 Općenito o C++ programskom jeziku

C++ je programski jezik opće namjene koji sadrži svojstva imperativnog, objektno-orijentiranog i generičnog programiranja te još sadrži niži nivo mogućnosti upravljanja memorijom [2].
Stvorio ga je Bjarne Stroustrup 1979. godine nakon što je za temu doktorata obrađivao programiranje i vidio je da je C programski jezik dosta ograničen ali ima dobar dizajn te je nakon nekoliko godina odlučio stvoriti novi programski jezik kojeg je nazvao C++ koji će sadržavati sve naredbe C-a ali i koncepte drugih programskih jezika. Ideje za C++ je izvlačio iz programa kao što su ALGOL 68, Ada, CLU and ML jer imaju dobru softversku primjenu ali se pisanje jezika presporo izvodi, a kao osnovu jezika je uzeo C [3].
C++ se često vidi kao "napredna verzija C programskog jezika" pošto C++ sadrži objektno-orijentirano programiranje, korištenje predložaka (*template),* veliku biblioteku klasa i druge naredbe koje se automatski podrazumijevaju dok se u C-u i dalje neke stvari moraju stalno pisati.
C programski jezik i dalje ima veliku primjenu u modernom softverskom inženjeringu ali zbog njegove ograničenosti se C++ mnogo više koristi.
C++ je dizajniran sa sklonošću prema sistemskom programiranju (npr. za ugrađene sustave kao što su matična ploča ili operativne sustave koji upravljaju s ulazno/izlaznim zahtjevima od strane softvera te ih prevodi u instrukcije koje prenose podatke u centralnu upravljačku jedinicu (CPU) i ostale elektroničke komponente kompjutera), s naglaskom na perfomanse, učinkovitost i fleksibilnost u dizajniranju.
C++ je dosta koristan u drugim kontekstima kao što aplikacije za desktop, servere (npr. e-komercijalnost, web pretraživanje ili SQL server), aplikacije čije su performanse od kritične važnosti (telefonske sklopke ili svemirske sonde) i softveri za zabavu (igre itd). Ima veliku podržanost i primjenu na mnogim platformama i mnoge tvrtke poput FSF, LLVM, Microsoft i Intel ga čine dostupnim javnosti za korištenje [4].

1. 1. 2 Osnovne naredbe i tipovi podataka korišteni za rješavanje problema u ovom radu

Kako bi razumjeli kako su algoritmi za rješavanje problema u ovom radu primjenjeni u C++ programskom kôdu, ukratko ćemo objasniti osnovne naredbe (naredbe će biti označene plavom bojom).

Osnovne naredbe:

; - operator koji označava kraj naredbe
// - operator za komentare koje program ignorira i služe programerima za opis značenja pojedinih naredbi ili dijelova kôda
- pretprocesorska naredba
++ - uvećaj varijablu za jedan
+= - kraći zapis za var=var+broj
>= - veće ili jednako
<= - manje ili jednako
= - pridruživanje vrijednosti
== - provjera jednakosti između dvije varijable
&& - logički uvjet "i"
|| - logički uvjet "ili"
! - operator negacije

!= - provjera različitosti između varijabli
cin - unos vrijednosti s tipkovnice, operator >> uz cin označava input - sintaksa: cin>>varijabla;
cout - ispis vrijednosti na ekran, operator << uz cout označava output
‘ ’ - dva jednostruka navodnika označavaju jedan znak
“ ” - dva dvostruka navodnika označava znakovni niz
{} - unutar vitičastih zagrada se izvršava blok naredbi koji su odijeljeni od ostatka programa
[] - označava raspon polja
return - vraća vrijednost broja ili neke varijable funkciji. Često označava kraj programa
break - označava izlaz iz petlje ili uvjeta
true - ako je neki uvjet točan tj. jednak 1
false - ako je neki uvjet netočan tj. jednak 0
const - označava konstantnu vrijednost varijable tokom izvođenja programa

Tipovi podataka => označavaju raspon brojeva za neku varijablu ili određuju ako varijabla ne sadrži brojevnu vrijednost nego znakovnu:

int - (integer) brojevni tip cijelih brojeva u rasponu $od\ -32768\ do\ 32767$
float - brojevni tip podatka realnih brojeva u rasponu $od\ -3,4*10^{38}\ do\ -1,17*10^{-38}\ i\ -1,17*10^{-38}\ do\ 3,4*10^{38}$
double - brojevni tip podatka realnih brojeva u rasponu $od\ -1,79*10^{308}\ do\ -2{,}22*10^{-308}\ i\ 2{,}22*10^{-308}\ do\ 1,7*10^{308}$
char - (character) tip podatka koji označava jedan znak
char* - tip podatka koji označava znakovni niz
string - klasa koja služi za lakše upravljanje znakovnim nizovima

Grananje IF tokom => Naredba if omogućuje uvjetno grananje toka programa ovisno o tome da li je ili nije zadovoljen uvjet naveden u zagradi iza ključne riječi if

```
if(uvjet1){
//blok naredbi koji se izvršava ukoliko je prvi uvjet točan
}
else if(uvjet2){
//blok naredbi koji se izvršava ukoliko prvi uvjet nije točan
}
else{
//blok naredbi koji se izvršavaju ukoliko nijedan uvjet iznad nije točan
}
```

Strukture su naredbe koje omogućavaju jednoj varijabli da sadrži više tipova podataka npr. varijabla student koja sadrži ime studenta i njegovo godište.

```
struct imeStrukture{
tip1 imeVarijable;
tip2 ImeVarijable;
};
```

U glavnom dijelu programa se koristi tako da se ime strukture stavi kao tip ispred imena varijable koja bi sadržavala tip1 podatak i tip2 podatak.

Primjer:
```
struct student{
string ime;
int godiste;
};
int main(){ -  glavni dio programa
student osoba;    -  Varijabla osoba sadrži ime godiste studenta
osoba. ime;    -  Poziva se tako da se stavi točka između varijable i podatka strukture
osoba. godiste;
```

Petlje su naredbe koje služe da se određeni dio programa ponavlja sve dok je određeni uvjet jednak istini.

```
for(početna vrijednost brojača;  uvjet brojača ili krajnja vrijednost indeksa;  brojevni izraz brojača){
//blok naredbi
}
while(uvjet){  - petlja će se ponavljati dok uvjet vrijedi ali se neće izvršiti ako na početku uvjet ne vrijedi
//blok naredbi
}

do
{
//blok naredbi
}while(uvjet);  - petlja će se barem jednom izvršiti iako uvjet na početku ne vrijedi
```

Polja ili **nizovi** su varijable koje u sebi sadrže više podataka istog tipa koji se dohvaćaju preko istog simboličkog imena i indeksa koji prikazuje redni broj elementa [3].

Polja mogu biti:

- jednodimenzionalna polja - kod njih se članovi polja dohvaćaju preko jednog indeksa. Članovi polja su složeni u linearnom slijedu, a indeks svakog člana označava njegovu udaljenost od početnog člana.
- višedimenzionalna polja - kod njih se članovi polja dohvaćaju s dva ili više indeksa. Dvodimenzionalna polja imaju najveću primjenu pošto predstavljaju matrice, tablice itd. Prvi indeks se često interpretira kao redak a drugi kao stupac.

Jednodimenzionalno: tip imePolja[maksimalnaVeličinaPolja]
Dvodimenzionalno: tip imePolja[maksimalnaVeličinaRedova][maksimalnaVeličinaStupaca]

Funkcije su cjeline koje se koriste za skupove naredbi koji se koriste više puta. Umjesto da se skup naredbi svaki put ponovno piše, ponavljani postupak se piše u obliku funkcije te se funkcija pozove svaki put kada je taj skup naredbi potreban.

Dobar primjer korištenja funkcije je računanje faktorijela u formuli $\binom{p}{r} = \frac{p!}{r!(p-r)!}$. Za izračunavanje faktorijela nekog broja je svaki put potrebno izračunati vrijednost varijable u koju će se spremati umnošci brojeva od 1 do tog broja i to svaki put u novoj petlji. To je besmisleno jer se iste naredbe trebaju utipkavati više puta i velika je mogućnost da dođi do pogreške. Zato je uputno kreirati funkciju faktorijel koja će izračunavati faktorijel nekog broja svaki put kada ju se pozove [3].

tip imeFunkcije(tip argument){definicija funkcije}

int main(){ - glavni dio programa

imeFunkcije(argument); - poziv funkcije u glavnom dijelu programa

2. Problem "Osam kraljica"

2. 1 Opis problema

Problem Osam kraljica je problem postavljanja osam kraljica na n × n šahovskoj ploči, tako da ne postoje dvije kraljice koje se međusobno napadaju. Dakle, dvije kraljice se ne smiju nalaziti u istom retku, stupcu ili dijagonali. Osam kraljica je problem koji se može riješiti na bilo kojoj dimenziji šahovske ploče osim za n = 2 i n = 3.
Za bilo koji n, na n x n ploču je moguće maksimalno postaviti *n* kraljica.

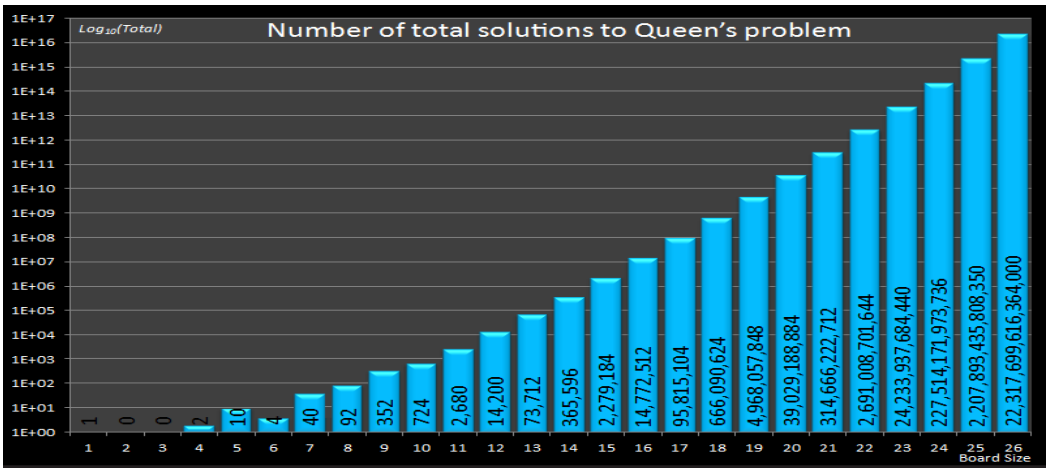
Prikaz kompleksnosti problema:

- Problem može biti vrlo računski skup zato što je kraljice moguće postaviti npr. za 8 x 8 ploču na 1. 78463e+014 načina dok postoje samo 92 rješenja [6].

n x n	Broj različitih kombinacija postavljanja kraljica na n x n ploču
n = 4	16! / 12! = 43680
n = 5	25! / 20! = 6. 3756e+006
n = 6	36! / 30! = 1. 40241e+009
n = 7	49! / 42! = 4. 32939e+011
n = 8	64! / 56! = 1. 78463e+014
n = 9	81! / 72! = 9. 4671e+016
n = 10	100! / 90! = 6. 28157e+019
n = 11	121! / 110! = 5. 09638e+022
n = 12	144! / 132! = 4. 96338e+025
n = 13	169! / 156! = 5. 71414e+028

Slika 2.1 Broj različitih kombinacija n kraljica na n x n ploči

Moguće je koristiti kratice koje smanjuju broj iteracija. Na primjer, u slučaju n = 8 primjenom jednostavnog pravila da dvije kraljice ne mogu biti na istom stupcu ili redku smanjuje se broj mogućnosti na samo 16,777,216 mogućih kombinacija. Generiranje permutacija dodatno smanjuje mogućnosti na samo 40,320 (odnosno 8!) [6].



Slika 2.2 Broj rješenja za različite dimenzije ploče

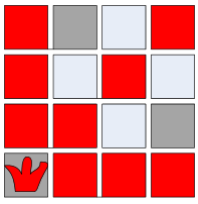
2. 2 Rješavanje problema

2. 2. 1 Unatražno rješavanje (backtracking solution)

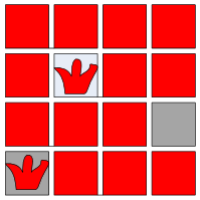
Unatražno rješavanje je takvo u kojemu se pretpostavi da se dvije kraljice ne mogu nalaziti u istom retku i istom stupcu što smanjuje broj iteracija (npr. za n = 8 broj iteracija je 8! = 40,320).
Slijedi prikaz unatražnog rješavanja problema na 4 x 4 ploči gdje je moguće postaviti najviše 4 kraljice. Kreće se od A1 (donjeg lijevog) polja prema A4 i onda na B1 prema B4 itd.

Rješenje:

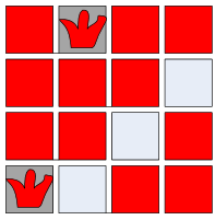
- krećemo od prvog stupca gdje kraljicu stavljamo na prvo polje



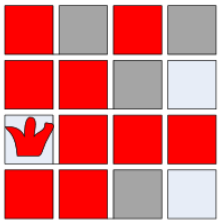
- zatim prelazimo u novi stupac i stavljamo kraljicu na prvo slobodno polje



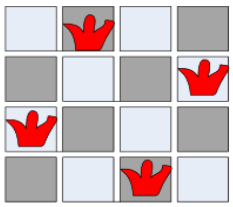
- u sljedećem stupcu ne postoji nijedno slobodno polje i zbog toga se vraćamo za jedan stupac i pomičemo kraljicu na iduće slobodno polje



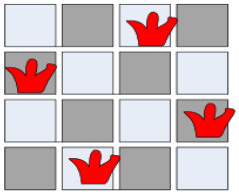
- postavljanjem kraljice na slobodno polje u idućem stupcu vidimo da u zadnjem stupcu neće biti slobodnog polja i vraćamo se za jedan stupac i pomičemo kraljicu u tom stupcu za jedno polje iznad. Ako je kraljica bila na kraju tog stupca znači da bi pomicanjem za poljem iznad izašla s ploče što znači da mičemo kraljicu iz tog stupca i opet se vraćamo za jedan stupac i pomičemo kraljicu u tom stupcu za polje iznad što je u ovom slučaju prvi stupac



- zatim ponavljamo cijeli postupak čime dolazimo do prvog rješenja



- nakon toga pomičemo kraljicu sa zadnjeg stupca i vraćamo se stupac unazad i pomičemo kraljicu na slobodno polje u tom stupcu. Ako dođemo do kraja stupca, vraćamo se za jedan stupac i pomičemo kraljicu na slobodno polje i prelazimo u idući stupac.



- postupak se ponavlja sve dok na prvom stupcu kraljica ne izađe s ploče što znači da smo isprobali sve kombinacije

2. 2. 2 Rješenje problema u C++ programskom jeziku

Jedan od načina rješavanja problema u C++ programskom jeziku je da dvodimenzionalnu ploču predočimo jednodimenzionalnim poljima (ne matricom) u kojem će indeks svakog polja `[i]` predstavljati broj stupca dok će brojevna vrijednost polja (`dame[i]`) predstavljati red u kojem se nalazi kraljica [5].
Prvo definiramo i inicijaliziramo polje od n članova (`dame[n]`) od koje je svako jednako 0 (znači da u tom stupcu nema kraljice). Kada se kraljica postavi na neko polje u tom stupcu, taj član će biti jednak redu u kojem se nalazi kraljica tog stupca.

```
int *dame = new int[n];

for (int i = 0;   i < n;   i++){
      dame[i] = 0;
}
```

Nakon toga stavljamo kraljicu na prvo polje prvog stupca (*q* = broj trenutnog stupca u kojem se postavlja kraljica ujedno i broj postavljenih kraljica na ploči).

```
int q = 0;
```

Zatim program ulazi u do-while petlju u kojoj će ponavljati kombinacije sve do krajnjeg uvjeta odnosno do izlaska kraljice s ploče u prvom stupcu.

```
            if (q < 0){ //krajnji uvjet je kad izadje iz prvog stupca
            cout << "GOTOVO!" << endl;
            break;
      }
```

Zatim program prelazi u idući stupac (na iduće polje) i provjerava da li se kraljica može postaviti u taj red tj. program poziva funkciju *safe()* koja provjerava da li je brojevna vrijednost tog polja jednaka brojevnoj vrijednosti polja prije.
Ono što smanjuje kompleksnost programa je to što funkcija provjerava samo lijevu stranu ploče pošto se podrazumijeva da je desna strana ploče prazna.

```
bool safe(int dame[],  int n,  int col){
      int c = 1;    //sluzi za oduzimanje elemenata niza jer da pise dame[i]-- onda bi
stvarno umanjivalo elemente dok ovako provjerava ako je kraljica na col-c polju na udaru
      for (int i = col;   i>0;   i--,  c++){
            if (dame[col] == dame[i - 1] || dame[col] == dame[i - 1] + c || dame[col]
== dame[i - 1] - c){ //prvi uvjet provjerava ako su kraljice u istom redu, drugi ako je
kraljica na dijagonali gore lijevo, treci ako je na dijagonali dolje lijevo
            return false;
            }
      }
      return true;
}
```

Funkcija vraća vrijednost *false* ako kraljica ne može ići na promatrano polje ili vrijednost *true* ako može. Ako funkcija vrati vrijednost *true* onda program postavlja kraljicu na to polje i prelazi na idući stupac.

```
            if (safe(dame,  n,  q)){
            q++;
```

Ako funkcija vrati vrijednost *false* onda program pomiče kraljicu na idući red i ponavlja petlju ispočetka.

```
                        else{
                                dame[q]++;
```

Nakon toga provjerava se da li je kraljica izašla s ploče i ako je izašla onda se resetira taj stupac (dame[q]=0), vraća se na stupac iza i pomiče kraljicu na idući red.

```
        if (dame[q] >= n){ //ako je na kraju reda onda ide na stupac iza
                dame[q] = 0;   //u tom stupcu nikako ne moze ici kraljica te se resetira
                        q--;   //vraca se na stupac prije
                        dame[q]++;   //na stupcu prije se uvecava za jedan
```

Nakon toga se opet provjerava da li će kraljica izaći s ploče te ponavlja prijašnji postupak.

```
                        if (dame[q] >= n){ //u slucaju ako je u stupcu prije kraljica
bila na vrhu
                                dame[q] = 0;
                                q--;
                                dame[q]++;
                        }
```

Ako je broj stupaca jednak n (broju mogućih kraljica n x n ploči) onda se ispisuju brojevne vrijednosti polja, a pomoću funkcije prikaz(dame, n) prikazuje se dvodimenzionalna ploča u kojoj nule prikazuju slobodna polja, a jedinice predstavljaju kraljice.
Ispisan će biti redni broj kombinacije, prikaz ploče pomoću nula i jedinica i ispod svake ploče će za svaki stupac biti ispisan redak u kojem se nalazi kraljica.

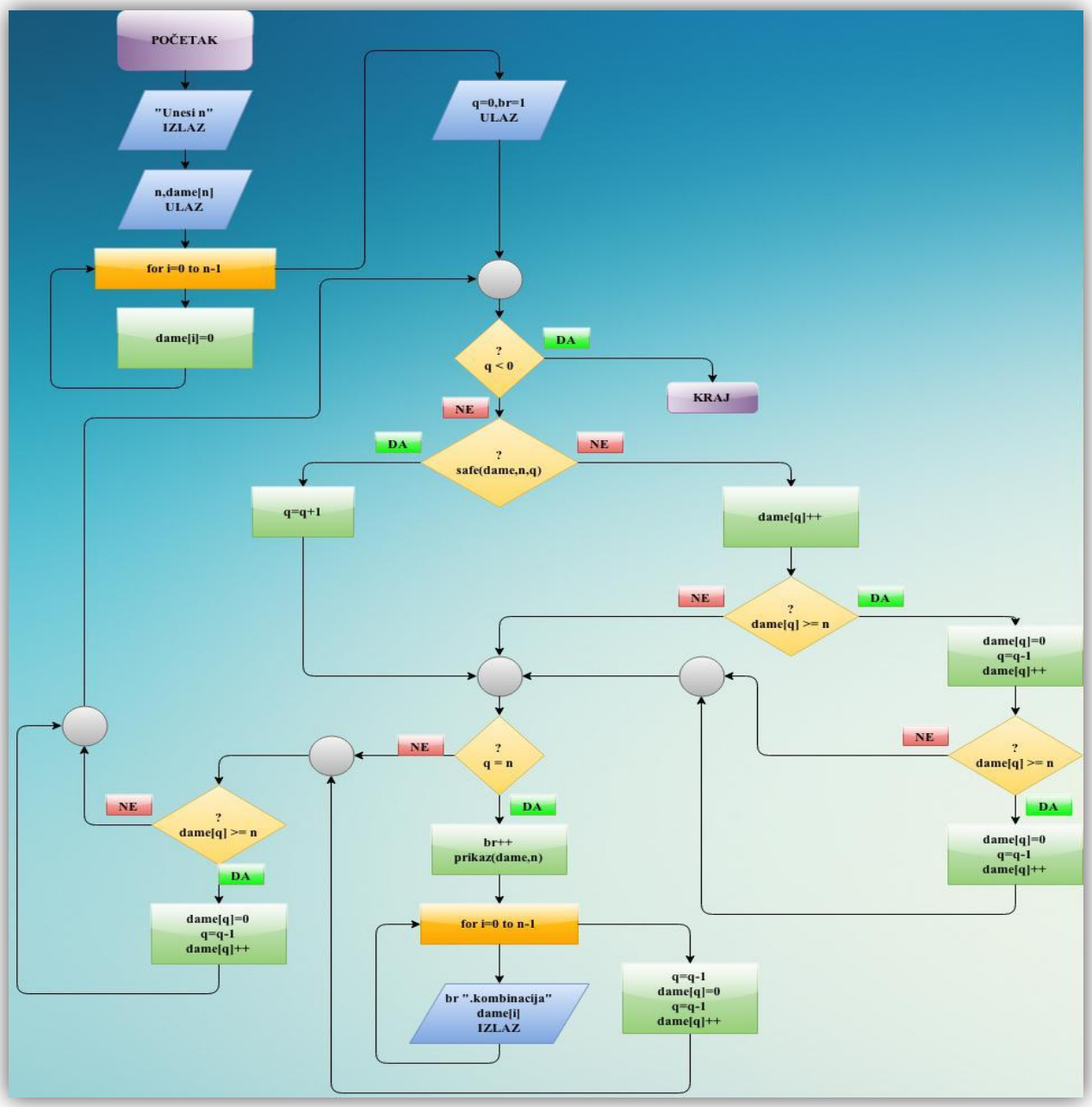
```
if (q == n){ //provjerava ako je broj stupaca jednak broju mogucih kraljica
                        cout << br++ << ".  kombinacija" << endl << endl;
                        prikaz(dame,  n);
                        for (int i = 0;   i < n;   i++){
                                cout << dame[i] + 1 << " ";
                        }
                        cout << endl << endl;
```

Zatim program briše kraljicu sa zadnjeg stupca i vraća se na stupac prije i pomiče kraljicu na idući slobodan red gdje opet provjerava da li će kraljica izaći s ploče.

```
        q--;   //onda se vraca na zadnji stupac posto je izasao s ploce
                        dame[q] = 0;   //zadnji stupac se resetira
                        q--;   //zatim se vraca stupac unazad
                        dame[q]++;   //i probava iducu moguci kombinaciju
                }
                if (dame[q] >= n){ //ako bilo u kojem trenutku predje broj redova
                        dame[q] = 0;   //onda resetira taj stupac na nulu
                        q--;   //vraca se na stupac iza
                        dame[q]++;   //i stavlja kraljicu na polje iznad
```

Ako kraljica izađe sa ploče resetirat će taj stupac i vratiti se na stupac prije i opet ponoviti petlju sve dok kraljica u prvom stupcu ne izađe s ploče. Nakon toga se izlazi iz petlje.

2. 2. 3 Prikaz rješenja pomoću dijagrama toka



Slika 2.3 Dijagram toka za "Osam kraljica"

3. "Round-Robin" algoritam

3. 1 Opis "Round-Robin" sustava

U teoriji, round robin turnir je najpravedniji način da se utvrdi prvak između poznatog i fiksnog broja sudionika. Svaki sudionik, igrač ili momčad ima jednake šanse protiv svih ostalih sudionika. Element sreće je smanjen jer jedna loša partija ne smanjuje šanse za pobjedu za razliku od "nokaut" sistema u turnirima gdje jedna loša partija znači ispadanje s turnira. Sveukupno postignuće sudionika je preciznije pošto je rezultat izveden tijekom dužeg perioda natjecanja protiv drugih suparnika. Na taj način se također može odlučiti koja je ekipa ili natjecatelj kvalitativno najlošiji. Npr. u timskom sportu gdje se primjenjuje round-robin sustav, šampioni lige se proglašavaju najboljim timom za razliku od eliminacijskog sustava turnira gdje jedna loša partija prekida sve.
Mana round-robin turnira je vrijeme potrebno da se završi. Broj rundi za n sudionika gdje je n paran je n-1 rundi, a kad je n neparan broj sudionika onda je n rundi. Npr. za 16 sudionika u eliminacijskom sustavu je potrebno 4 runde (15 partija) da se turnir završi dok je kod round-robin sustava potrebno 15 rundi (120 partija) da se sve završi i nema finalne partije koja sve odlučuje.

3. 1. 1 Raspoređivanje parova

Ako je n broj sudionika, onda je $\frac{n}{2}(n-1)$ partija. Ako je n paran onda je $n-1$ rundi i u svakoj rundi je moguće odigrati najviše $\frac{n}{2}$ partija u isto vrijeme. Ako je n neparan onda je n rundi i u svakoj je moguće najviše $\frac{n-1}{2}$ partija i jedan od sudionika se odmara u tekućoj rundi [7].
Standardni algoritam za round-robin sustav je da se svakom sudioniku dodijeli broj i spari se s nekim drugim u prvoj rundi.

Runda 1:

1	2	3	4
8	7	6	5

Tada treba fiksirati jedan broj u prvom ili zadnjem stupcu (u ovom slučaju **1**) i rotirati ostale u smjeru kazaljke na satu.

Runda 2:

1	8	2	3
7	6	5	4

Runda 3:

1	7	8	2
6	5	4	3

Runda 4:

1	6	7	8
5	4	3	2

Runda 5:

1	5	6	7
4	3	2	8

Runda 6:

1	4	5	6
3	2	8	7

Runda 7:

1	3	4	5
2	8	7	6

Ako je neparan broj natjecatelja onda jedan broj može biti lažan te kad je neki drugi broj s njim uparen, tada taj natjecatelj odmara tu rundu.
Prednost kod ovog algoritma je što jedan red može predstavljati domaću/gostujuću ekipu, crni/bijeli u šahu itd. Pošto je jedan broj fiksan onda bi ta podjela trebala ići naizmjenično svaku rundu.

U našem programu je korištena i biti će opisana alternativna verzija round-robin sustava koji se zove Bergerov sustav koji je nazvan prema austrijskom šahovskom majstoru John Bergeru. Bergerov sustav se najviše koristi u planiranju turnira [7].
Bergerov sustav se razlikuje od standardnog round-robin sustava po tome što u prvi red moraju biti postavljeni brojevi od 1 do $n/2$ ($\frac{n+1}{2}$ ako je n neparan), a u drugi red s lijeva nadesno od n do $\frac{n}{2}$+1 i n mora biti fiksni.

Runda 1:

1	2	3	4
8	7	6	5

Nakon toga se drugi brojevi rotiraju za $\frac{n}{2}$ poziciju u smjeru kazaljke na satu. Tako je bolje jer ovaj sustav naizmjenično daje podjelu domaći/gosti, bijeli/crni itd. i lakše se generira manualno.
Aritmetički se svi brojevi osim fiksnog zbroje s $\frac{n}{2}$ i ako rezultat bilo kojeg od njih bude veći od $(n-1)$ onda se rezultat oduzima s $(n-1)$.

Runda 2:

5	6	7	1
8	4	3	2

Runda 3:

2	3	4	5
8	1	7	6

Runda 4:

6	7	1	2
8	5	4	3

Runda 5:

3	4	5	6
8	2	1	7

Runda 6:

7	1	2	3
8	6	5	4

Runda 7:

4	5	6	7
8	3	2	1

3. 1. 2 Prikaz "Round-Robin" sustava u C++ programskom jeziku

U programskom jeziku C++ prvo stvorimo strukturu clan koja ce sadržavati ime i broj sudionika.

```
struct clan{

	string ime;
	int broj;
};
```

Zatim deklariramo varijable koje će nam pomoći pri grananju programa:

```
string ime_izbor;   //varijabla koja predstavlja korisnikovu odluku za upisom imena
bool imena = false;   //incijalizirana je po default-u na false
bool paran = true;   //pretpostavlja da je broj natjecatelja paran
int runde;
int n;        //broj natjecatelja
```

Nakon toga upisujemo broj sudionika `n` u do-while petlji koja će se ponavljati sve dok je broj sudionika manji ili jednak 1.

```
	do{
			cout << "Unesi broj sudionika" << endl;
			cin >> n;
			if (n < 2){
				cout << "Mora biti vise od jednog" << endl;
			}
		} while (n < 2);
```

Nakon što `n` varijabla ima vrijednost, provjeravamo da li je `n` paran ili neparan i na temelju toga varijabla `paran` prima vrijednost true ili false ovisno o rezultatu.

```
	if (n % 2 == 1){
			paran = false;
			n += 1;   //ako je n neparan onda ga povecava za 1 kako bi bio paran
		}
```

Zatim deklariramo tri niza i alociramo memoriju za svaki niz posebno. `igrac1` će predstavljati prvu polovicu sudionika (prvi red), `igrac2` drugu polovicu (drugi red), a `tmp_ime` koji ćemo popuniti prvom i drugom polovicom služit će nam za zamjenu imena.

```
clan *igrac1 = new clan[maks];   //prva polovica niza od 1 do n/2
clan *igrac2 = new clan[maks];   //druga polovica od n/2+1 do n
clan *tmp_ime = new clan[maks];   //pomocni niz koji ce sadrzavati imena igraca te ce se
nizovi koji sadrze podatke igraca usporedjivati s ovim pomocnim kako bi dobili prava
imena
                        //jer inace ce zamijeniti brojeve dok ce imena ostati ista
```

Korisnik odlučuje da li želi upisivati imena i program postavlja broj rundi na $(n-1)$.

```
	cout << "Zelite li upisivati imena sudionika?" << endl;
		cin >> ime_izbor;
		if (ime_izbor == "da" || ime_izbor == "Da" || ime_izbor == "dA" ||
ime_izbor == "d" || ime_izbor == "D" || ime_izbor == "a" || ime_izbor == "A"){
```

```
                    imena = true;

                    cout << "Unesi imena igraca" << endl;
             }

runde = n - 1;
```

Inicijaliziramo prvu polovicu sudionika `igrac1`:

```
for (int i = 1;   i <= n / 2;   i++){  //petlja koja ide od 1 do n/2

       igrac1[i]. broj = i;    //prva polovica niza

       tmp_ime[i]. broj = igrac1[i]. broj;   //popunjavamo prvu polovicu pomoćnog niza

              if (imena == true){ //ako korisnik zeli upisivati imena onda se pomocni niz
       popunjuje nakon što korisnik unese ime
                           cout << i << ".  igrac:  ";
                           cin >> igrac1[i]. ime;   //unos imena
                           tmp_ime[i]. ime = igrac1[i]. ime;
                    }
```

Inicijaliziramo drugu polovicu sudionika `igrac2`:

```
for (int j = (n / 2) + 1;   j <= n;   j++){ //petlja koja će ići od n/2+1 do n

                    igrac2[j]. broj = j;
                    tmp_ime[j]. broj = igrac2[j]. broj;

                    if (paran == false && j == n){ //ako je n neparan i ako je brojac j
jednak n onda je jedan sudionik lažan, a to je n+1 sudionik kojeg nema

                           igrac2[j]. ime == " ";   //n+1 sudionik neće imati ime
                           tmp_ime[j]. ime = igrac2[j]. ime;
                    }
                    else{
                           if (imena == true){ //upisivanje imena drugoj polovici niza
                                  cout << j << ".  igrac:  ";   //druga polovica niza
                                  cin >> igrac2[j]. ime;
                                  tmp_ime[j]. ime = igrac2[j]. ime;
                           }
                    }

             }
```

Zatim slijede dvije petlje, prva petlja koja će odbrojavati runde:

```
for (int runda = 1;   runda <= runde;   runda++){
                    cout << "Runda " << runda << ": " << endl;
```

I druga petlja koja će kombinirati parove u svakoj rundi:

```
for (int i = 1,  j = n;   i <= n / 2;   i++,  j--)
```

Brojač `i` predstavlja prvu polovicu niza `igrac1` dok brojač `j` predstavlja drugu polovicu niza `igrac2`. Svaki put kad uđe u petlju, `i` će se povećati za 1 jer poprima vrijednosti od 1 do $n/2$, a `j` će se smanjiti za 1 jer poprima vrijednosti od n do $n/2 + 1$ (taj uvjet nije potreban pošto će se brojač i povečavati $n/2$ puta kao i brojač `j`). Ovdje su zapravo dvije for petlje stavljene u jednu.

Nakon toga u petljama slijedi prvi uvjet koji provjerava da li je `n` neparan i brojač `j` jednak `n` kako bi ispisao koji igrač odmara tu rundu pošto u svakoj rundi jedan sudionik nema para.

```
if (paran == false && j == n){ //ako je n neparan, svaki put kad neki igrac zaigra s n+1 onda ispisuje da odmara

	cout << setw(5) << igrac1[i]. broj << ".  " << igrac1[i]. ime << " se odmara" << endl;

                        }
```

Inače ispisuje parove:

```
	else{
	cout << setw(5) << igrac1[i]. broj << ". " << igrac1[i]. ime << " vs " << igrac2[j]. broj << ".  " << igrac2[j]. ime << endl;

                        }
```

Onda zbraja svaki broj prve polovice niza (gornji red kojeg predstavlja varijabla `igrac1`) s $\frac{n}{2}$ nakon čega provjerava ako je rezultat veći od $(n-1)$ i ako je veći onda taj rezultat oduzima s $(n-1)$.

```
	igrac1[i]. broj += (n / 2);

	if (igrac1[i]. broj > n - 1){ //provjerava ako je veći od (n-1)

	igrac1[i]. broj -= (n - 1);   //ako je veći onda ga oduzima s (n-1)
                        }
```

Nakon toga slijedi petlja koja ide po pomoćnom nizu i uspoređuje brojeve pomoćnog niza s novim brojevima prve polovice originalnog niza koji su rezultat nakon zbrajanja s $\frac{n}{2}$ te dodjeljuje ime polja pomoćnog niza polju originalnog niza ako imaju isti broj.

```
for (int k = 1;    k <= n;    k++){ //petlja za usporedjivanje rednog broja igraca radi zamjenjivanja imena

	if (igrac1[i]. broj == tmp_ime[k]. broj){

		igrac1[i]. ime = tmp_ime[k]. ime;    //dodijeljuje ime sudionika pomocnog niza, sudioniku originalnog niza prve polovice (gornji red)
                                }
                        }
```

Nakon te petlje program dolazi do uvjeta koji provjerava da li brojač `j` ima istu vrijednost kao i fiksni broj sudionika (*n*). Ako ima istu vrijednost onda taj redni broj sudionika ne zbraja s $\frac{n}{2}$ i preskače ga.

```
if (j != n){ //n mora biti fiksiran te ako dodje na n, njega ne zbraja s n/2
```

Ako je brojač `j` različit od *n* onda zbraja broj druge polovice (donji red odnosno varijabla `igrac2`) s $\frac{n}{2}$ i provjerava da li je rezultat veći od $(n-1)$ i ako je veći onda ga oduzima s $(n-1)$.

```
igrac2[j]. broj += (n / 2);

if (igrac2[j]. broj > n - 1){

igrac2[j]. broj -= (n - 1);
}
```

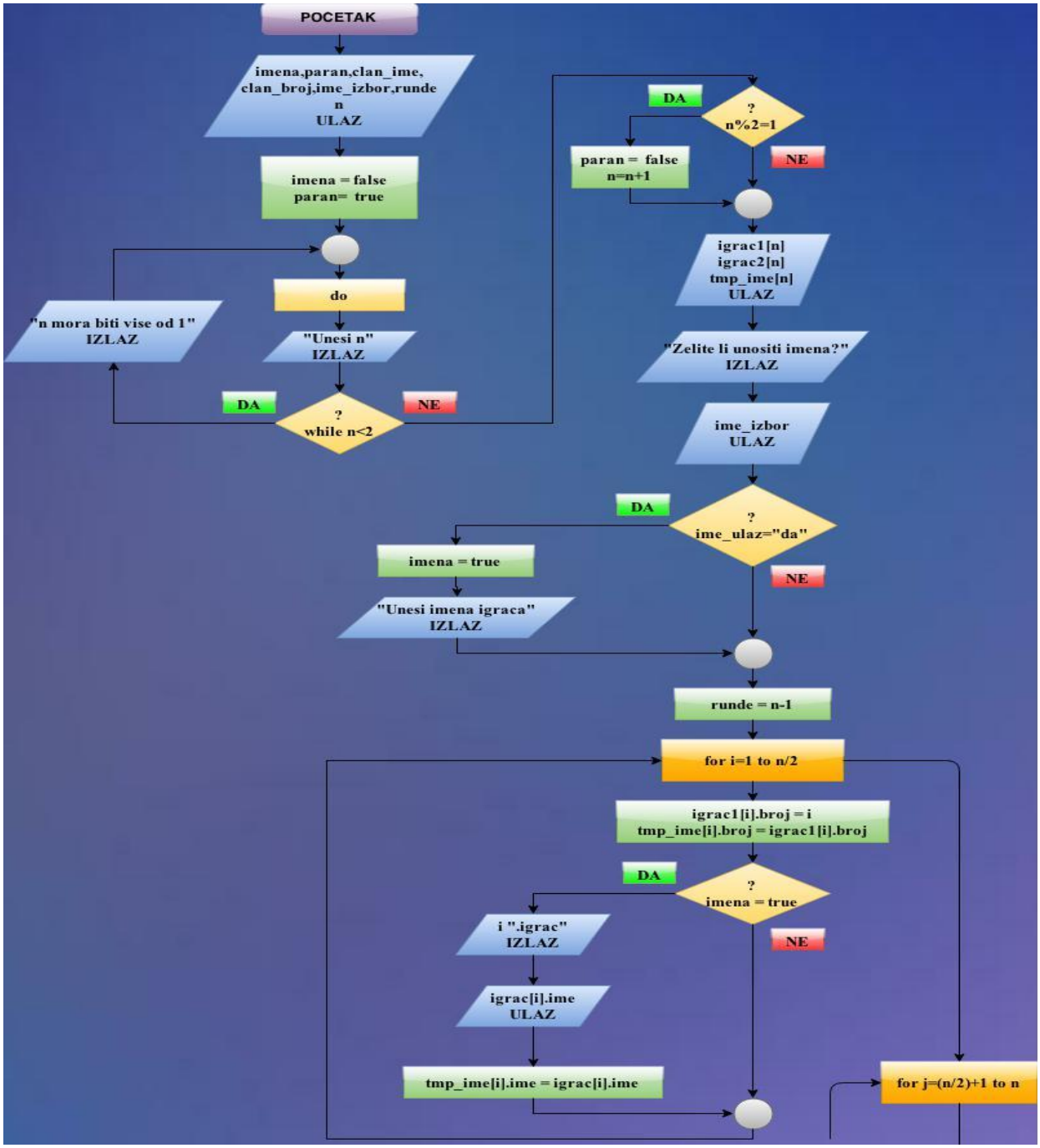
Nakon toga program ulazi u petlju koja kao i kod prve polovice niza ide po pomoćnom nizu i traži jednake brojeve i kad nađe sudionike koji imaju iste brojeve, daje vrijednost imena pomoćnog niza sudioniku originalnog niza druge polovice.

```
for (int k = 1;   k <= n;   k++){

        if (igrac2[j]. broj == tmp_ime[k]. broj){

                igrac2[j]. ime = tmp_ime[k]. ime;     //dodijeljuje ime sudionika
        pomocnog niza, sudioniku originalnog niza druge polovice (donji red)
                                        }
                                }
```

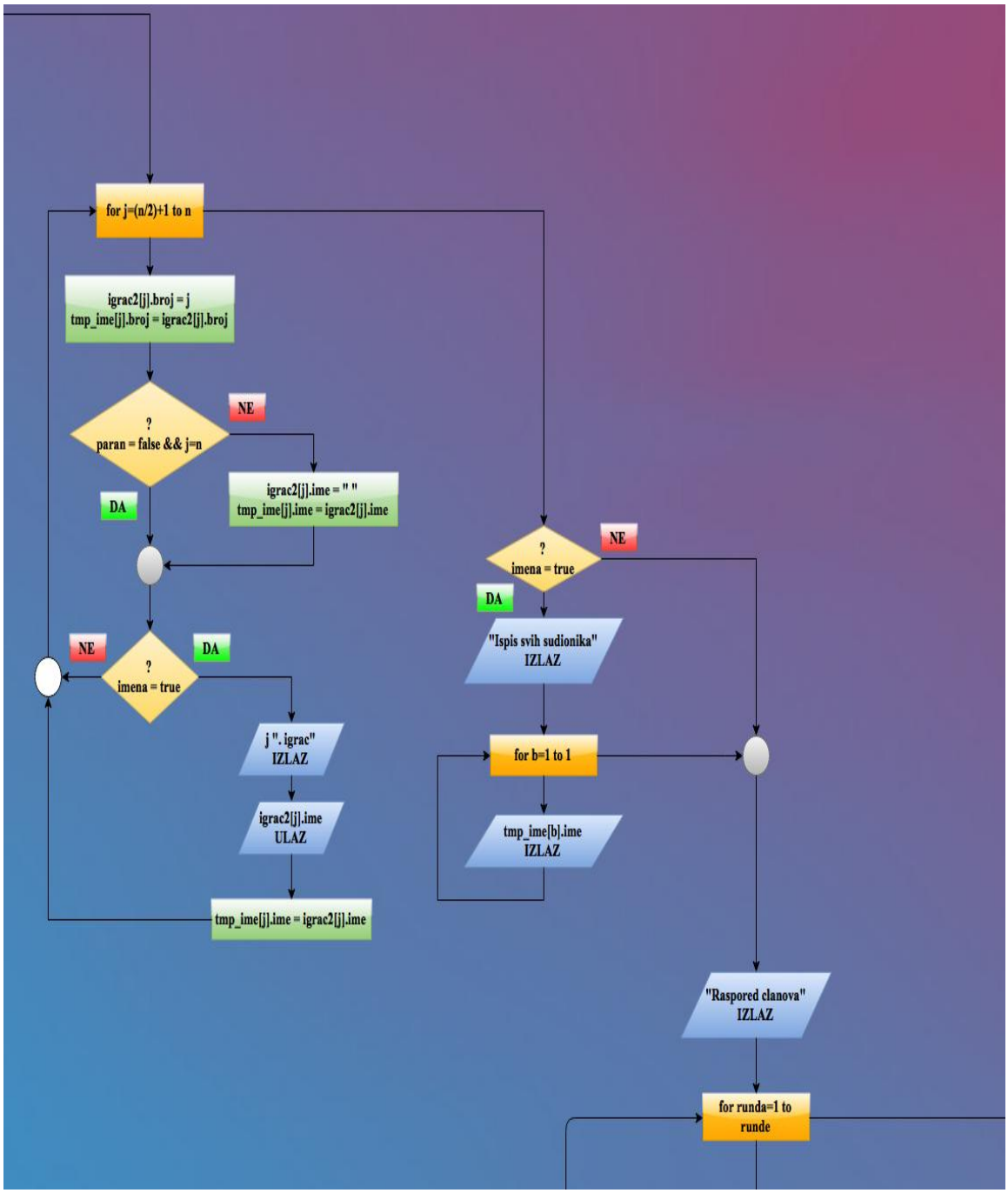
Nakon što je prošao i ispisao sve kombinacije, dealocira memorijski prostor koji su zauzimala polja `igrac1`, `igrac2` *i* `tmp_ime`.

```
delete[]igrac1;
delete[]igrac2;
delete[]tmp_ime;
```

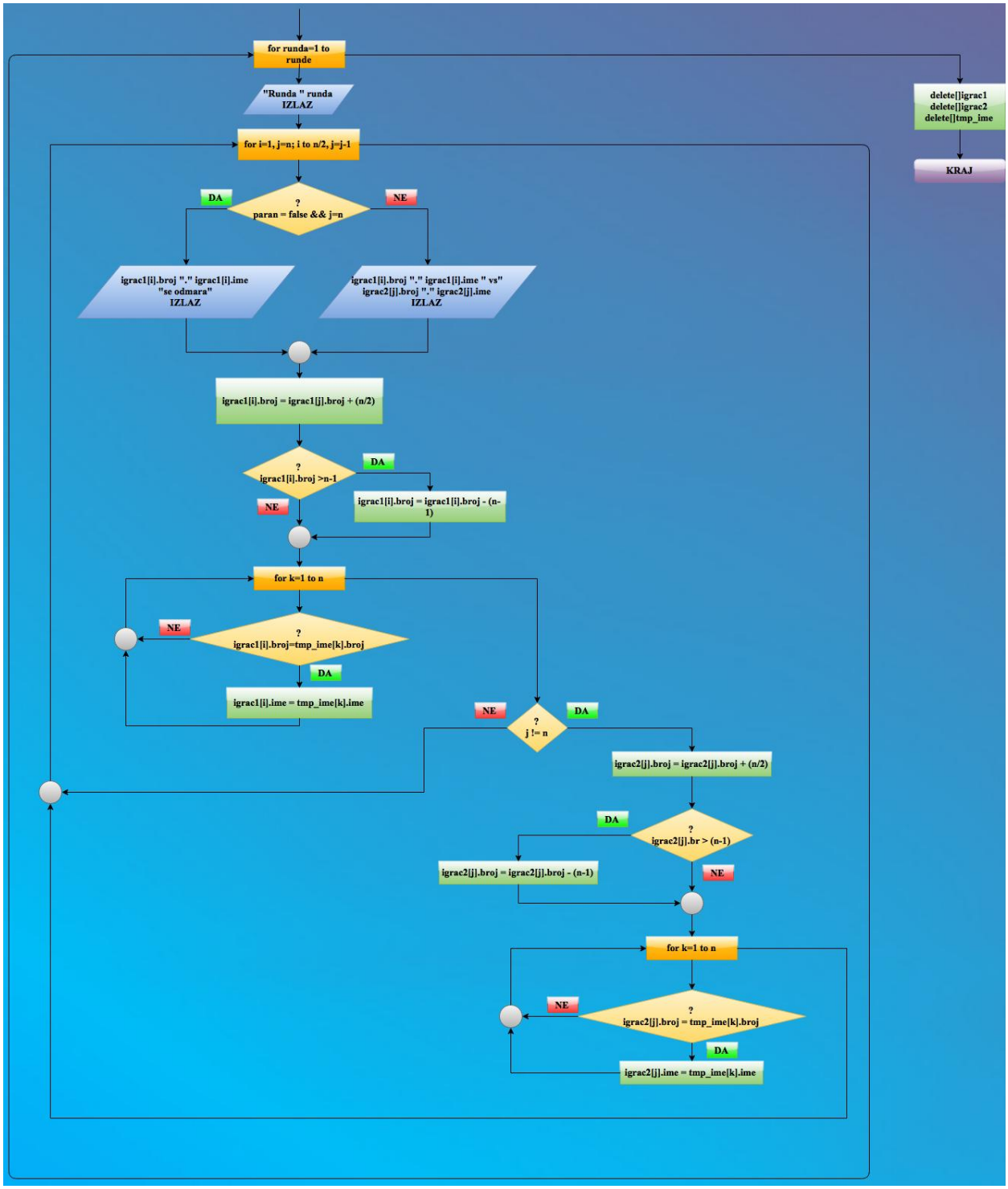
3. 1. 3 Prikaz dijagrama toka “Round-Robin” sustava u C++



Slika 3.1 Prvi dio programa, od početka do unosa imena i dodjeljivanja brojeva igračima prve polovice



Slika 3.2 Drugi dio koji prikazuje dodjeljivanje imena i brojeva drugoj polovici i ispisuje imena svih sudionika

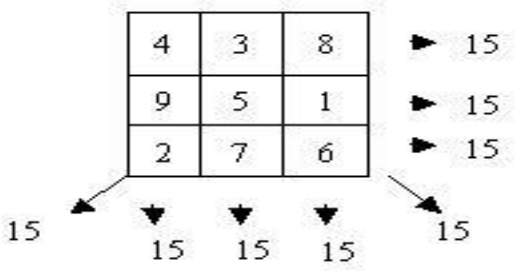


Slika 3.3 Treći dio programa koji prikazuje primjenu round-robin algoritma na Bergerov način i dealokaciju memorije

4. Magični kvadrati

4. 1 Opis magičnih kvadrata

Magični kvadrat je raspored cijelih brojeva za neki n od 1 do n^2 u $n\,x\,n$ matrici koja ima jednak broj redova i stupaca te se u svakom kvadratu nalazi po jedan broj te su ti brojevi raspoređeni tako da zbroj brojeva u u svakom retku, stupcu te glavnoj i sporednoj dijagonali bude jednak.



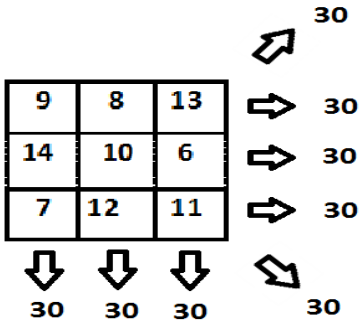
Prikaz magičnog kvadrata za n = 3 gdje je zbroj svakog retka, stupca i dijagonala jednak 15

Broj kojem moraju biti jednaki zbroj redova, stupaca i dijagonala se naziva **magična suma** i dobiva se formulom $M = [n(n^2 + 1)]/2$.
Broj *n* se naziva **red** ili **dimenzija**.
Magični kvadrat 1. reda je samo jedan broj, 2. reda ne postoji, a za redove 3, 4, 5, 6, 7, 8 je magična suma jednaka 15, 34, 65, 111, 175 i 260.

n	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
M	1	/	15	34	65	111	175	260	369	505	671	870	1105	1379	1695	2056	2925

Prikaz magičnih suma za prvih n = 17 magičnih kvadrata

Svaki magični kvadrat može biti rotiran pa se tako može kreirati $(n^2 - 1)$ različitih kvadrata.
Brojevi u magičnim kvadratima ne moraju biti nužno poredani od 1 do n^2 gdje je inkrement jednak 1. Mogu biti poredani tako da je inkrement jednak 2, 3, 4, 5 itd. Npr.



Slika 4.1 Prikaz magičnog kvadrata gdje je inkrement jednak 5, a magična suma jednaka 30

Aritmetička progresija kvadrata ne mora početi od 1 (npr. za 3. red 1, 2, 3, 4, 5, 6, 7, 8, 9). Može početi od bilo koliko, samo inkrement mora biti konstantan.

4. 2 Rješenja magičnih kvadrata

Rješenja za magične kvadrate se dijele na tri slučaja:

1. Za *n* koji je neparan
2. Za *n* koji je duplo paran tj. da je *n* djeljiv s 2 i sa 4 npr. 4, 8, 12, 16 itd.
3. Za *n* koji jednostruko paran tj. djeljiv je samo s 2 npr. 6, 10, 14, 18 itd.

U našem C++ programu nakon što se unese *n*, prvo se magični kvadrat popunjava s nulama (bit će objašnjeno kasnije) i onda se poziva funkcija ovisno o svojstvima *n*–tog reda te se upisuju brojevi u dvodimenzionalni niz tj. matricu. U nastavku će biti opisana svaka od gornjih slučajeva, prvo metoda pa zatim implementacija u C++ programskom jeziku.
Na kraju će program ispisati popunjeni magični kvadrat i ispod njega sumu svakog retka, stupca, glavne i sporedne dijagonale.

4. 2. 1 Rješenje za *n* koji je neparan

Rješenje za magične kvadrate neparnog reda se rješava Siamezovom metodom koju je predstavio francuski matematičar Simon de la Loubère u 17. stoljeću.
Siamezova metoda glasi tako da se 1 (ili prvi broj u bilo kojoj aritmetičkoj progresiji) stavlja u srednje polje prvog reda i nakon toga se kreće **gore i desno** (↗) **dijagonalno i** upisuje se sljedeći broj u nizu. Ako se izađe iz kvadrata, onda se nastavlja sa suprotne strane npr. Ako se izađe iznad kvadrata, onda se nastavlja u zadnjem redu ili izađe s desne strane kvadrata, onda nastavlja s lijeve strane.
Ako se dođe na polje koje je ispunjeno, onda se na polju od kuda se krenulo pomiče **vertikalno dolje** (↓) na prvo slobodno polje, upisuje broj i kreće dijagonalno kao prije [9].
Prikaz ispunjavanja kvadrata za 3. red:

1. korak		
	1	

2. korak		
	1	
		2

3. korak		
	1	
3		
		2

3. korak		
	1	
3		
4		2

5. korak		
	1	
3	5	
4		2

6. korak		
	1	6
3	5	
4		2

7. korak		
	1	6
3	5	7
4		2

8. korak		
8	1	6
3	5	7
4		2

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

4. 2. 2 Prikaz rješenja za *n* koji je neparan u C++ programskom jeziku

Nakon što se unese *n* koji neparan, poziva se funkcija `mag_kv_neparni(int square[][maks], int n)` koja za argument prima dvodimenzionalni niz i dimenziju kvadrata. Zatim ta funkcija zove još jednu funkciju `siamese(square, n, br)` u kojoj se primjenjuje Siamezova metoda rješavanja.

```
void siamese(int square[][maks], int n, int br = 1, int r1 = 0, int s1 = 0)
```

Funkcija *siamese* prima za argument:

- `int square[][maks]` -> dvodimenzionalni niz (matrica) u koju se pohranjuju brojevi
- `int n` -> dimenzija ili red magičnog kvadrata
- `int br = 1` -> predstavlja aritmetički red te u slučaju ako se ne unese neka druga vrijednost u pozivu funkcije, aritmetički red će uvijek biti jedan te će uvijek upisivati brojeve od 1 do n^2 s konstantnim razmakom od 1 između dva broja
- `int r1 = 0` -> broj redaka koji je inicijaliziran na nulu te se povećava do *n*
- `int s1 = 0` -> broj stupaca koji je inicijaliziran na nulu te se povećava do *n*

Pošto Siamezova metoda započinje upisivanjem broja jedan u sredinu prvog reda, prvo definiramo srednji stupac:

```
int s = (n / 2) + s1;
```

Nakon toga definiramo `aritmRed` varijablu koja predstavlja aritmetički red.

```
int aritmRed = br;
```

Zatim u while petlji koja provjerava da li je upisani broj manji od $n^2 + tmp$ i ako je taj uvjet točan onda upisujemo brojeve do $n^2 + tmp$

```
while (br < (n*n) + aritmRed)
```

Slijedi prvi if uvjet koji provjerava da li je polje prazno tako što poziva funkciju `prazno(int square[][maks], int r, int s)` koja za argument prima matricu `square[][maks]` trenutni red `r` *i* trenutni stupac `s` te vraća vrijednost true ako je polje prazno i upisuje broj ako je prazno polje ili vrijednost false ako nije prazno.

```
if (prazno(square, r, s)){
                    square[r][s] = br;   //ako je polje prazno, upisuje broj na to polje
```

Program zatim smanjuje broj reda za jedan i uvećava broj stupaca za jedan tj. pomiče se gore i desno.

```
br += 1;   //zatim broj koji se upisuje uvećava se za 1
                    r--;   //red se smanjuje za jedan tj. ide za jedno polje gore
                    if (r < r1){ //ako izadje iznad malog kvadrata
```

```
                        r = (r1 + n) - 1;    //onda se provjera nastavlja u zadnjem
retku
                }
                s++;    //povećava stupac za jedan tj. pomice se udesno za jedno
polje
                if (s >= s1 + n){ //ako izadje s desne strane malog kvadrata
                        s = s1;   //onda se provjera nastavlja u lijevom stupcu
                }
```

Ako je prvi uvjet netočan tj. polje nije prazno onda se pomiče vertikalno dolje na trenutnom polju i provjerava ako je i to polje prazno i tako sve dok ne nađe prazno polje.

```
else{
                s--;    //broj stupca se smanjuje za jedan tj. pomice se vertikalno
dolje

                if (s < s1){ //ako izadje s lijeve strane maloga kvadrata
                        s = (s1 + n) - 1;   //onda prelazi u stupac s druge strane
                }

                r += 2;    //red se povećava za dva umjesto za jedan jer se prije
ovog smanjio za jedan jer se prvo provjeravalo ako je gore-desno polje prazno

                if (r >= (r1 + n)){ //ako izadje ispod malog kvadrata
                        if (r == (r1 + n)){ //u slucaju ako izadje za jedno polje van,
onda se vraca na prvi red
                                r = r1;
                        }
                        else{ //ako je za dva polja vani onda je u drugom redu
                                r = r1 + 1;   //onda prelazi u red s druge strane
                        }
                }
```

Nakon toga se vraća na provjeru uvjeta while petlje ako je upisani broj manji od $n^2 + tmp$.

4. 2. 3 Rješenje za *n* koji je duplo paran

Parni brojevi se dijele na dvije skupine, oni koji su djeljivi sa 4 i zovu se “duplo parni” (double even) i oni koji nisu djeljivi s 4 i zovu se “jednostruko parni” (single even). Zbog toga su algoritmi za rješavanje magičnog kvadrata različiti za ove dvije skupine.
Za “duplo parne” kvadrate za *n* koji je jednak 4, cijela se matrica popuni nulama osim glavne i sporedne dijagonale koja se popuni jedinicama. Takva matrica se zove “matrica istine” [10].

1	0	0	**1**
0	**1**	**1**	0
0	**1**	**1**	0
1	0	0	**1**

Zatim se kreće od prvog polja prvog retka i stupca i upisuju se brojevi od 1 (svejedno otkuda se počinje samo da je razlika između dva susjedna člana kroz cijeli niz konstantna) do kraja matrice u polja koja su označena s jedinicom.

1	0	0	**4**
0	**6**	**7**	0
0	**10**	**11**	0
13	0	0	**16**

Nakon što smo došli do kraja, krećemo od prvog polja koje je jednako nuli u prvom retku i upisujemo neiskorištene brojeve od $(n^2 - 1)$ (ili od najvećeg neiskorištenog broja) do nule u polja koja su jednaka nuli.

1	**15**	**14**	4
12	6	7	**9**
8	10	11	**5**
13	**3**	**2**	16

Za *n* koji je duplo paran i veći od 4 podijelimo magični kvadrat u 4 x 4 mala kvadrata.

1	63	62	4	5	59	58	8
56	10	11	53	52	14	15	49
48	18	19	45	44	22	23	41
25	39	38	28	29	35	34	32
33	31	30	36	37	27	26	40
24	42	43	21	20	46	47	17
16	50	51	13	12	54	55	9
57	7	6	60	61	3	2	64

Slika 4.2 Dijeljenje 8 x 8 kvadrata na četiri 4x4 mala kvadrata

Zatim na isti način po dijagonalama malih kvadrata upisujemo jedinice te nakon toga upisujemo brojeve od 1 (ovisno o aritmetičkom redu) do n^2 u polja gdje su jedinice. Nakon što smo stigli do posljednjeg polja, od prvog polja jednakog nuli u prvom redu upisujemo brojeve od $n^2 - 1$ do nule.

1	0	0	**1**	**1**	0	0	**1**
0	**1**	**1**	0	0	**1**	**1**	0
0	**1**	**1**	0	0	**1**	**1**	0
1	0	0	**1**	**1**	0	0	**1**
1	0	0	**1**	**1**	0	0	**1**
0	**1**	**1**	0	0	**1**	**1**	0
0	**1**	**1**	0	0	**1**	**1**	0
1	0	0	**1**	**1**	0	0	**1**

Slika 4.3 Upis jedinica u male kvadrate

1	0	0	**4**	**5**	0	0	**8**
0	**10**	**11**	0	0	**14**	**15**	0
0	**18**	**19**	0	0	**22**	**23**	0
25	0	0	**28**	**29**	0	0	**32**
33	0	0	**36**	**37**	0	0	**40**
0	**42**	**43**	0	0	**46**	**47**	0
0	**50**	**51**	0	0	**54**	**55**	0
57	0	0	**60**	**61**	0	0	**64**

Slika 4.4 Upis brojeva u polja gdje su jedinice

1	**63**	**62**	4	5	**59**	**58**	8
56	10	11	**53**	**52**	14	15	**49**
48	18	19	**45**	**44**	22	23	**41**
25	**39**	**38**	28	29	**35**	**34**	32
33	**31**	**30**	36	37	**27**	**26**	40
24	42	43	**21**	**20**	46	47	**17**
16	50	51	**13**	**12**	54	55	**9**
57	**7**	**6**	60	61	**3**	**2**	64

Slika 4.5 Upis brojeva u polja jednaka nuli

4. 2. 4 Prikaz rješenja za duplo parni *n* u C++ programskom jeziku

Nakon što se unese *n* koji je djeljiv s 2 i sa 4 poziva se funkcija `mag_kv_dpl_parni(int square[][maks], int n)` koja za argumente prima dvodimenzionalni niz (matricu) `square[][maks]` i dimenziju `n` magičnog kvadrata i oba argumenta su tipa integer (cjelobrojni tipovi).
Nakon što se pozove funkcija, u njoj se prvo pozivaju dvije for petlje, prva koja ide po redovima i druga koja ide po stupcima. Pošto se magični kvadrat dijeli na male 4 x 4 kvadrate onda se petlje pomiču za 4 reda i 4 stupca. Definirane su pomoćne varijable koje nakon što petlja dođe na početak svakog malog kvadrata, idu po glavnoj i sporednoj dijagonali malog kvadrata i upisuju se jedinice.

```
for (int i = 0;   i < n;   i += 4){ //predstavlja redove 4x4 kvadrata

            int gl_dij = 0,   sp_dij = gl_dij + 3;    //gl_dij je prvi element glavne
dijagonale dok je sp_dij prvi element sporedne dijagonale

            for (int j = 0;   j < n;   j += 4,   gl_dij = j,   sp_dij = j + 3){ //j se
uvecava za cetiri jer skace na pocetak svakog MALOG kvadrata u odredjenom redu,

                        //' gl_dij = j' jer se pocinje od nultog elementa tj. pocetka
malog kvadrata ili pocetka glavne dijagonale malog kvadrata, dok je sp_dij na kraju prvog
reda malog kvadrata tj.  na pocetku sporedne dijagonale

                    int red_mk = i;   //redovi malog kvadrata.  Ići će u svaki red malog
kvadrata
```

```
                    int sq = 0;    //sq (square) sluzi kako bi prosao cetiri reda i
cetiri stupca i svaki put kad predje na novi mali kvadrat se resetira kako bi se cijeli
mali kvadrat popunio

                    while (sq < 4){ //while petlja ce se vrtiti sve dok nisu popunjene
glavna i sporedna dijagonala malog kvadrata
                           square[red_mk][gl_dij] = 1;   //posto je svaki mali kvadrat na
drugoj poziciji, treba paziti u kojem je redu i stupcu
                           square[red_mk][sp_dij] = 1;
                           gl_dij++,  sp_dij--,  red_mk++;
                           sq++;
                    }
```

Nakon što su upisane jedinice po glavnim i sporednim dijagonala malih kvadrata, slijede dvije for petlje koje idu po svakom polju magičnog kvadrata u kojima se nalazi if uvjet koji poziva funkciju `prazno(square, i, j)` koja vraća true ako je polje jednako 1 ili false ako je polje jednako nula i ako je true upisuje trenutnu vrijednost varijable `br` koja se uvećava svakim prolaskom kroz petlju.

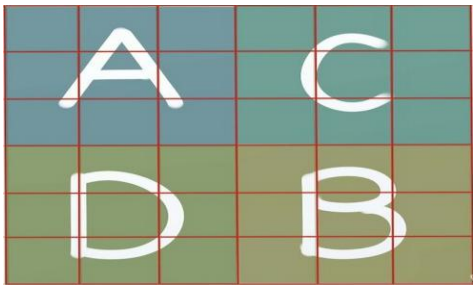
```
int br = 1;   //predstavlja upisani broj. Inicijalizirana je na jedan jer od tuda pocinje
upis te ce se njena vrijednost upisivati u polja
      for (int i = 0;   i < n;   i++){ //prva for petlja koja ide po redovima
            for (int j = 0;   j < n;   j++){ //druga for petlja koja ide po stupcima
                  if (!prazno(square,   i,   j)){ //provjerava ako je polje jednako
jedinici tj.  ako je popunjeno
                           square[i][j] = br;   //upis trenutne vrijednost varijable br
                  }
                  br++;
            }
      }
```

Na kraju imamo opet dvije for petlje u kojima se provjerava koja su polja jednaka nuli pomoću if uvjeta i `prazno()` funkcije i upisuje trenutna vrijednost varijable `br` ako je polje jednako nuli.

```
for (int i = 0;   i < n;   i++){
            for (int j = 0;   j < n;   j++){
                  br--;
                  if (prazno(square,  i,  j)){ //provjerava ako je polje jednako nuli
tj.  ako nije popunjeno
                           square[i][j] = br;
                  }
            }
      }
}
```

4. 2. 5 Rješenje za *n* koji je jednostruko paran

Da se podsjetimo, jednostruko parni brojevi su parni brojevi koji su djeljivi s 2 ali nisu s 4, a to su: 6, 10, 14, 18 tj. $(n * 4) + 2$.
Rješenje za dimenziju *n* koji jednostruko paran je malo složenije od neparnog i duplo parnog n-a.
Prvo magični kvadrat dijelimo na četiri manja kvadrata čije su dimenzije $(n/2) * (n/2)$ [9].
Primjer rješavanja će biti prikazan za $6 * 6$ magični kvadrat.



*Slika 4.6 Prikaz 6*6 magičnog kvadrata koji je podijeljen na četiri manja 3*3 kvadrata. Slova A, B, C i D predstavljaju redoslijed popunjavanja kvadrata*

Zatim svaki kvadrat ispunjavamo Sijamezovom metodom (vidi poglavlje **4. 2. 1**) ovim rasponom brojeva:

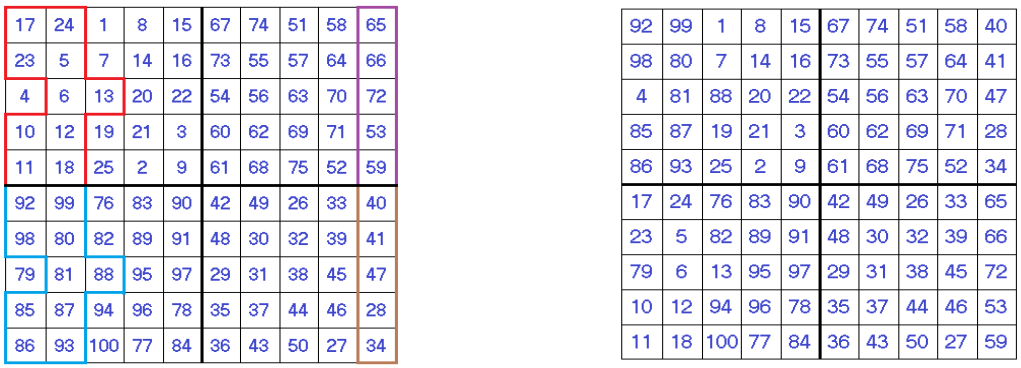
- u **A** manji kvadrat idu brojevi od 1 do $n^2/4$
- u **B** manji kvadrati idu brojevi od $(n^2/4) + 1$ do $n^2/2$
- u **C** manji kvadrat idu brojevi od $\frac{n^2}{2} + 1$ do $3 * (n^2)/4$
- u **D** manji kvadrat idu brojevi od $(3 * (n^2)/4) + 1$ do n^2

Nakon što su brojevi upisani onda se zamjenjuju brojevi za prvih **k** stupaca koji iznosi $(\frac{n-2}{4})$ osim srednjih elemenata prvog stupca prvog i trećeg kvadrata.

8	1	6	26	19	24
3	5	7	21	23	25
4	9	2	22	27	20
35	28	33	17	10	15
30	32	34	12	14	16
31	36	29	13	18	11

Slika 4.7 Prikaz zamjene prvih k stupaca za magični kvadrat 6. reda

Kao što je prikazano na slici, srednji element prvog stupca prvog kvadrata je “3” dok je srednji element prvog stupca trećeg kvadrata broj “30”.
Zbog toga što se preskaču srednji elementi prvog stupca, centralni brojevi prvog (5) i trećeg malog kvadrata (32) se zamjenjuju.
Isto tako se zamjenjuju svi elementi zadnjih **k-1** stupaca (24, 25, 20 se zamjenjuju s 15, 16, 11). Za n=10 to je samo zadnji stupac kao što je prikazano na slici.



17	24	1	8	15	67	74	51	58	65
23	5	7	14	16	73	55	57	64	66
4	6	13	20	22	54	56	63	70	72
10	12	19	21	3	60	62	69	71	53
11	18	25	2	9	61	68	75	52	59
92	99	76	83	90	42	49	26	33	40
98	80	82	89	91	48	30	32	39	41
79	81	88	95	97	29	31	38	45	47
85	87	94	96	78	35	37	44	46	28
86	93	100	77	84	36	43	50	27	34

92	99	1	8	15	67	74	51	58	40
98	80	7	14	16	73	55	57	64	41
4	81	88	20	22	54	56	63	70	47
85	87	19	21	3	60	62	69	71	28
86	93	25	2	9	61	68	75	52	34
17	24	76	83	90	42	49	26	33	65
23	5	82	89	91	48	30	32	39	66
79	6	13	95	97	29	31	38	45	72
10	12	94	96	78	35	37	44	46	53
11	18	100	77	84	36	43	50	27	59

*Slika 4.8 Zamjena prvih **k** stupaca preskačući srednje elemente prvog stupca (*4 i 79*) i zamjena centralnih elemenata prvog i trećeg kvarata (*13 i 88*).*
*Prikazana je i zamjena elemenata zadnjih **k-1** stupaca za drugi mali kvadrat koji su označeni s ljubičastom bojom i za četvrti mali kvadrat koji su označeni narančastom bojom*

4. 2. 6 Prikaz rješenja za jednostruko parni *n* u C++ programskom jeziku

Nakon što se unese *n* koji je jednostruko paran (6, 10, 14, 18 itd.), poziva se funkcija pod imenom `mag_kv_singl_parni(int square[][maks], int n)` koja za argumente prima dvodimenzionalni niz (matricu) `square[][maks]` i red kvadrata *n*.
Zatim se ulaskom u funkciju deklarira varijabla `malo` koja se odmah inicijalizira na vrijednost $\frac{n}{2}$. Ona predstavlja dimenzije malih kvadrata koje su $\frac{n}{2} * \frac{n}{2}$ dimenzija. Nakon nje se kreira varijabla `br` čija će se vrijednost preslikavati u polja malih kvadrata te će za svaki kvadrat imati svoj određeni raspon brojeva.
Na slici 4.6 je prikazan redoslijed popunjavanja kvadrata slovima A, B, C, D te:

- slovo A predstavlja prvi mali kvadrat
- slovo B predstavlja četvrti mali kvadrat
- slovo C predstavlja drugi mali kvadrat
- slovo D predstavlja treći mali kvadrat

U programu se za svaki mali kvadrat poziva funkcija `siamese(square, mali, br)` koja se koristila za rješavanje magičnih kvadrata neparnih dimenzija pomoću Sijamezove metode. Ta funkcija prima pet argumenata od kojih su prva tri uvijek ista:

1. Argument `square` koji predstavlja dvodimenzionalni niz tj. matricu magičnog kvadrata
2. Argument `mali` predstavlja dimenziju kvadrata $(n/2)$
3. Argument `br` predstavlja raspon brojeva koji se preslikavaju u polja malih kvadrata
4. Argument koji predstavlja prvi red malog kvadrata
5. Argument koji predstavlja prvi stupac malog kvadrata koji zajedno s četvrtim argumentom daju poziciju prvog broja malog kvadrata (gornji lijevi broj)

Zatim se varijabla `br` inicijalizira na 1 i poziva se funkcija za A kvadrat tj. za prvi kvadrat koji se popunjava brojevima od 1 do $n^2/4$.

```
br = 1;

siamese(square,  mali,  br);
```

Ova funkcija prosljeđuje samo tri argumenta jer su prvi red i stupac A kvadrata na nultoj poziciji, a oni su inicijalizirani automatski u deklaraciji funkcije na nulu (`r1,s1`).

```
void siamese(int square[][maks],  int n,  int br = 1,  int r1 = 0,  int s1 = 0)
```

Onda se poziva druga funkcija za B odnosno za četvrti kvadrat koji se popunjava brojevima od $(n^2/4) + 1$ do $n^2/2$.

```
br = ((n*n) / 4) + 1;
siamese(square,  mali,  br,  mali,  mali);
```

U toj funkciji se za prvi red i prvi stupac B kvadrata proslijedila varijabla `mali` jer se taj kvadrat nalazi na donjoj desnoj četvrtini kvadrata te se prvi broj tog kvadrata nalazi na $square[(n\text{^}2)/2][\,(n\text{^}2)/2]$ poziciji.

Za C odnosno drugi kvadrat se opet poziva funkcija kojoj se za četvrti argument (prvi red C kvadrata) prosljeđuje 0 dok se za stupac tog kvadrata vrijednost varijable `mali`. Prije toga se varijabla `br` inicijalizira na $(n^2/2) + 1$ jer se u taj kvadrat upisuju brojevi od te vrijednosti do $3 * (n^2)/4$.

```
br = ((n*n) / 2) + 1;   //od (n^2)/2+1 do 3(n^2)/4
siamese(square,   mali,   br,   0,   mali);    //prvi broj se nalazi na gornjoj desnoj
četvrtini
```

Za treći mali kvadrat D se poziva funkcija kojoj se za četvrti argument (prvi red D kvadrata) prosljeđuje vrijednost varijable `mali` dok se za stupac tog kvadrata automatski prosljeđuje nula jer se nalazi na donjoj lijevoj četvrtini. Varijabla `br` se inicijalizira na $(3 * (n^2)/4)) + 1$ jer se u D kvadrat upisuju brojevi od $(3 * (n^2)/4)) + 1$ do n^2.

```
br = 3 * n*n / 4 + 1;   //(3(n^2)/4)+1 do n^2
siamese(square,  mali,  br,  mali);
```

Nakon što program popuni cijeli magični kvadrat pozivom funkcija za A, B, C, D kvadrate, kreira varijablu `k` koja se inicijalizira na vrijednost $(\frac{n-2}{4}) - 1$. U C++ programskom jeziku prvi član polja je na nultoj poziciji, drugi član na prvoj, treći na drugoj itd. te se zbog toga varijabla `k` umanjuje za jedan.

Kako je rečeno u poglavlju **4. 2. 5**, nakon što se popuni magični kvadrat, zamjenjuju se brojevi prvih `k` stupaca prvog i trećeg odnosno A i D kvadrata.
Zbog toga se kreiraju dvije `for` petlje gdje će prva petlja ići po stupcima dok će druga petlja ići po redovima.

```
for (int stupac = 0;   stupac <= k;   stupac++){

            for (int r1 = 0,  r2 = mali;   r1 < mali && r2 < n;   r1++,  r2++){ //r1
predstavlja redove A kvadrata dok r2 predstavlja redove D kvadrata
```

U drugoj for petlji se poziva funkcija `zamjena1()` u kojoj se prosljeđuju vrijednosti dviju varijabli koje zamjenjuju vrijednosti unutar funkcije. U nastavku je prikazana deklaracija i definicija te funkcije.

```
void zamjena1(int &a,  int &b){ //ispred imena varijabli se stavlja operator refenciranja cime se prosljedjuju originalne vrijednosti varijable, a ne kopije jer bez tog operatora, vrijednosti varijabli se ne bi zamijenile
	int tmp = a;
	a = b;
	b = tmp;
}
```

Poziv funkcije izgleda ovako:

```
zamjena1(square[r1][stupac], square[r2][stupac]);	//zamjena brojeva prvog i treceg kvadrata
```

Prvi argument je broj koji se nalazi u redu A kvadrata (`r1`), a drugi argument je broj koji se nalazi u redu D kvadrata (`r2`). Pošto se oba broja nalaze u istom stupcu, druga dimenzija niza je ista.
Zatim bi se trebali zamijeniti srednji element A kvadrata prvog stupca sa srednjim elementom D kvadrata istog stupca. Zapravo se u rješavanju ti brojevi uopće ne zamjenjuju ali da se ne komplicira s dodatnim uvjetima koji će provjeravati da li je petlja na tim poljima pa da ih tada preskače, lakše je pozvati funkciju koja će ih vratiti na staro mjesto.

Nakon toga slijedi if-else uvjet gdje se testira da li je `n` jednak 6.

```
if (n == 6){ //ako je n jednak 6 onda je k jednak nuli ali kod nizova nula predstavlja prvi element npr.  prvi red ili prvi stupac
		zamjena1(square[k + 1][k],  square[k + mali + 1][k]);	//zamjenjuju se element srednjeg reda prvog i treceg malog kvadrata prvog stupca tj. vracaju se na pocetnu poziciju
		zamjena1(square[k + 1][k + 1],  square[k + mali + 1][k + 1]);	//element srednjeg reda malog kvadrata koji se nalazi na glavnoj dijagonali, zamjenjuje se s elementom na sporednoj dijagonali istog reda treceg (ispod) kvadrata

	}
	else{
		zamjena1(square[k + 1][k - 1],	square[k + mali + 1][k - 1]); //zamjenjuju se element srednjeg reda prvog i treceg malog kvadrata prvog stupca jer su se zamijenili svi elementi prvih k stupaca
		zamjena1(square[k + 1][k + 1],  square[k + mali + 1][k + 1]);	//element srednjeg reda malog kvadrata koji se nalazi na glavnoj dijagonali,  se zamjenjuje s elementom na sporednoj dijagonali istog reda treceg (ispod) kvadrata
	}
```

Ovdje je potreban if-else uvjet jer ako je `n` jednak 6, onda je `k` prema formuli $n = 4k + 2$ nula te bi se pri vraćanju srednjih elemenata A i D kvadrata prvog stupca dogodila greška jer bi se zamijenili brojevi drugog stupca jer se kroz funkciju `zamjena1()` za argument `square[][]` za drugu dimenziju prosljeđuje $k - 1$ dok bi za $n = 6$ gdje je k jednak nula, funkcija trebala zamijeniti elemente nultog stupca, a ne -1 stupca.

Na kraju su kreirane još dvije petlje koje služe za zamjenu zadnjih $k - 1$ stupaca.

```
//zamjena zadnjih k-1 stupaca
	for (int stupac = n - 1;   stupac >= n - k;   stupac--){
		for (int r1 = 0,  r2 = mali;   r1 < mali && r2 < n;   r1++,  r2++){
			zamjena1(square[r1][stupac],  square[r2][stupac]);
		}

	}
```

Sve je isto kao i u prošle dvije petlje samo što je varijabla stupac u prvoj petlji jednaka $n-1$ pošto se zamjenjuju elementi posljednjih stupaca. Kreće od posljednjeg stupca i sve do $n-k$ stupca.
Druga petlja i dalje služi za prolazak po svakom redu i funkcija `zamjena1()` u kojoj se zamjenjuju elementi B i C kvadrata.

4. 3. Objašnjenje ispisa magičnih kvadrata u C++ programu

Kao što je spomenuto u poglavlju **4. 2**, program će prvo ispisati poruku kojom se od korisnika traži da unese *n* i zatim će ispisati koliko je elemenata magičnih kvadrata za tu dimenziju i magičnu sumu koja se dobiva formulom $= [n(n^2+1)]/2$.

Nakon toga se dvodimenzionalni niz `square[maks][maks]` inicijalizira tako da je svaki element u matrici jednak nuli jer je za neparni i jednostruko parni *n* potrebno imati inicijalizirano polje jer se svaki element izjednačava s varijablom `br`.
Zatim se kroz if-else grananje odabire funkcija ovisno o zadovoljenom uvjetu.

```
if (n % 2 != 0){ //ako je n neparan

		mag_kv_neparni(square,  n);
	}
	else if (n % 2 == 0 && n % 4 == 0){ //ako je n djeljiv s 2 i sa 4 tj.  ako je
duplo parni

		mag_kv_dpl_parni(square,  n);
	}
	else {

		mag_kv_singl_parni(square,    n);      //ako  je  djeljiv  samo  s  dva  tj.
jednostruko parni
	}
```

Pozivanjem jedne od prethodnih funkcija, niz `square[][]` se inicijalizirao te je postao magični kvadrat. Na kraju je ostalo još samo ispisati magični kvadrat odnosno matricu te zbrojeve redova, stupaca i dijagonala kako bi se uvjerili da je ta matrica zapravo magični kvadrat. Za tu akciju je korištena funkcija `ispis_suma(square,  n)` koja za argumente prima niz `square[][]` i dimenziju matrice *n*.

U funkciji se deklariraju lokalne varijable tj. varijable koje postoje samo u toj funkciji i izvan te funkcije se ne mogu koristiti.
Prvo se deklarira varijabla *a* cijelobrojnog tipa int i inicijalizira se na nulu. Ona će služiti za zbrajanje elemenata glavne dijagonale. Svaki put kad se petlja ponovi ona će se uvećati za jedan te će ona biti postavljena kao prva dimenzija nizu `square[][]` i svaki put kad se uveća, dohvaća se element u sljedećem redu.

Slijedeća varijabla koja će imati istu funkciju kao i varijabla *a* samo što će služiti za zbrajanje elemenata sporedne dijagonale i svakim prolaskom kroz petlju će se umanjivati za jedan, a to je varijabla *c* koja je isto cijelobrojnog tipa int i inicijalizira se na vrijednost $n-1$.
Sljedeće varijable služe za pohranu rezultata.

```
int suma_r = 0,  suma_s = 0,  suma_gl = 0,  suma_sp = 0;   //s lijeva na desno -> suma retka, suma stupca, suma glavne dijagonale i suma sporedne dijagonale
```

Pozivaju se dvije petlje kroz koje se ispisuje svaki element matrice.

```
for (int i = 0;   i < n;   i++){
              for (int j = 0;   j < n;   j++){

                     cout << setw(5) << square[i][j];
                     izlaz<< setw(8) << square[i][j];
              }
              cout << endl << endl;
              izlaz<< endl << endl;
       }
```

Dalje je prikazana petlja u kojoj se pristupa svakom element te se zbrojevi određenih elemenata dodaju odgovarajućim varijablama (`suma_r, suma_s, suma_gl, suma_sp`).

```
for (int i = 0;   i < n;   i++,  a++,  c--){ //u for petlju se smiju dodavati i druge varijable koje ce se ponasati na definirani nacin svakim novim prolaskom kroz petlju

              suma_r = 0;   //suma retka se svakim novim prolaskom kroz petlju resetira
              suma_s = 0;   //suma stupca se svakim novim prolaskom kroz petlju resetira

              for (int j = 0;   j < n;   j++){

                     suma_r += square[i][j];   //dohvacaju se svi elementi jednog stupca
                     suma_s += square[j][i];   //dohvacaju se svi elementi jednog reda

              }
              suma_gl += square[a][i];   //svakim prolazom kroz petlju se prelazi u novi red i novi stupac krenuvsi s lijeva na desno i s vrha prema dnu matrice
              suma_sp += square[c][i];   //svakim prolazom kroz petlju se prelazi u novi red i novi stupac krenuvsi s desna na lijevo i s vrha prema dnu matrice


              cout << "Suma " << i + 1 << ".  retka je " << suma_r << endl;
              cout << "Suma " << i + 1 << ".  stupca je " << suma_s << endl;
              izlaz<< "Suma " << i + 1 << ".  retka je " << suma_r << endl;
              izlaz<< "Suma " << i + 1 << ".  stupca je " << suma_s << endl;

       }
```

Na kraju funkcije je postavljen ispis rezultata dijagonala.

```
cout << "Suma glavne dijagonale je " << suma_gl << endl;
       cout << "Suma sporedne dijagonale je " << suma_sp << endl;
       izlaz << "Suma glavne dijagonale je " << suma_gl << endl;
       izlaz<< "Suma sporedne dijagonale je " << suma_sp << endl;
```

5. Programski kôdovi

5. 1 Programski kôd za problem “Osam dama”

```
#include <iostream>
#include <iomanip>
#include <stdlib. h>
using namespace std;
const int maks = 1000;
void prikaz(int dame[],  int n){

	int(*ploca)[maks] = new int[n][maks];
	for (int i = 0;   i<n;   i++){
		for (int j = 0;   j < n;   j++){

			if (j == dame[i]){
				ploca[i][j] = 1;
			}
			else{
				ploca[i][j] = 0;
			}
			cout << ploca[i][j] << " ";
		}
		cout << endl;
	}
	cout << endl;
	delete[]ploca;
}

bool safe(int dame[],  int n,  int col){
	int c = 1;    //sluzi za oduzimanje elemenata niza jer da pise dame[i]-- onda bi stvarno umanjivalo elemente dok ovako provjerava ako je kraljica na col-c polju na udaru

	for (int i = col;   i>0;   i--,  c++){

		if (dame[col] == dame[i - 1] || dame[col] == dame[i - 1] + c || dame[col] == dame[i - 1] - c){ //prvi uvjet provjerava ako su kraljice u istom redu, drugi ako je kraljica na dijagonali gore lijevo, treci ako je na dijagonali dolje lijevo

			return false;

		}

	}

	return true;
}

int main(){
	//metoda u kojoj ide iz reda u red i upisuje te ako ne moze upisati u jedan red, vraca se u red iznad i onda tamo za jedan stupac se pomice i upisuje itd. .
	int n;
	cout << "Unesi dimenzije n*n ploce" << endl;
	cin >> n;
```

```
	int *dame = new int[n];

	for (int i = 0;   i < n;   i++){
		dame[i] = 0;
	}

	int q = 0;
	int br = 1;   // broji kombinacije
	do{

		if (q < 0){ //krajnji uvjet je kad izadje iz prvog stupca
			cout << "GOTOVO!" << endl;
			break;
		}
		if (safe(dame,   n,   q)){ //provjrava ako kraljica moze ici u odredjenom stupcu na odredjeno polje

			q++;   //ako moze onda prelazi na iduci stupac

		}
		else{
			dame[q]++;   //ako ne moze onda prelazi na iduci red
			if (dame[q] >= n){ //ako je na kraju reda onda ide na stupac iza
				dame[q] = 0;   //u tom stupcu ne moze ici dama te se resetira
				q--;   //vraca se na stupac prije
				dame[q]++;   //na stupcu prije se uvecava za jedan
				if (dame[q] >= n){ //u slucaju ako je u stupcu prije, kraljica bila na vrhu
					dame[q] = 0;
					q--;
					dame[q]++;
				}
			}
		}

		if (q == n){ //ako je prošao sve stupce i onda ispisuje kombinacije
			cout << br++ << ".  kombinacija" << endl << endl;
			prikaz(dame,  n);
			for (int i = 0;   i < n;   i++){
				cout << dame[i] + 1 << " ";
			}
			cout << endl << endl;

			q--;   //onda se vraca na zadnji stupac posto je izasao s ploce
			dame[q] = 0;   //zadnji stupac se resetira
			q--;   //zatim se vraca stupac unazad
			dame[q]++;   //i probava iducu moguci kombinaciju
		}
		if (dame[q] >= n){ //ako bilo u kojem trenutku predje broj redova
			dame[q] = 0;   //onda resetira taj stupac na nulu
			q--;   //vraca se na stupac iza
			dame[q]++;   //i stavlja kraljicu na polje iznad

		}
	} while (1);   //vrti petlju do krajnjeg uvjeta
	system("pause");
	return 0;
}
```

5. 2 Programski kôd za “Round-robin” algoritam

```
#include <iostream>
#include<iomanip>
#include <string>
#include<fstream>
#include <stdlib. h>
using namespace std;

const int maks = 10000;

struct clan{

	string ime;
	int broj;
};

int main(){

	ofstream izlaz;
	izlaz. open("Clanovi_parovi. txt");
	string ime_izbor;
	bool imena = false;    //provjerava ako korisnik zeli za svakog clana upisivati imena
	bool paran = true;
	int runde;

	int n;



		do{
			cout << "Unesi broj sudionika" << endl;
			cin >> n;
			if (n < 2){
				cout << "Mora biti vise od jednog" << endl;
			}
		} while (n < 2);
		if (n % 2 == 1){
			paran = false;
			n += 1;
		}

		clan *igrac1 = new clan[maks];   //prva polovica niza
		clan *igrac2 = new clan[maks];   //druga polovica
		clan *tmp_ime = new clan[maks];    //pomocni niz koji ce sadrzavati imena igraca te ce se nizovi koji sadrze podatke igraca usporedjivati s ovim pomocnim kako bi dobili prava imena
                                                                                //jer    inace    ce zamijeniti brojeve dok ce imena ostati ista

		cout << "Zelite li upisivati imena sudionika?" << endl;
		cin >> ime_izbor;
```

```
		if (ime_izbor == "da" || ime_izbor == "Da" || ime_izbor == "dA" ||
ime_izbor == "d" || ime_izbor == "D" || ime_izbor == "a" || ime_izbor == "A"){

			imena = true;

			cout << "Unesi imena igraca" << endl;
		}

		runde = n - 1;

		//incijaliziranje prve polovice niza
		for (int i = 1;   i <= n / 2;   i++){

			igrac1[i]. broj = i;    //prva polovica niza

			tmp_ime[i]. broj = igrac1[i]. broj;

			if (imena == true){
				cout << i << ".  igrac:  ";
				cin >> igrac1[i]. ime;
				tmp_ime[i]. ime = igrac1[i]. ime;
			}

		}

		//incijaliziranje druge polovice niza
		for (int j = (n / 2) + 1;   j <= n;   j++){

			igrac2[j]. broj = j;
			tmp_ime[j]. broj = igrac2[j]. broj;

			if (paran == false && j == n){

				igrac2[j]. ime == " ";
				tmp_ime[j]. ime = igrac2[j]. ime;
			}
			else{
				if (imena == true){
					cout << j << ".  igrac:  ";   //druga polovica niza
					cin >> igrac2[j]. ime;
					tmp_ime[j]. ime = igrac2[j]. ime;
				}
			}

		}
		if (imena == true){
			cout << "Ispis svih sudionika" << endl;
			for (int b = 1;   b <= n;   b++){
				cout <<setw(b+4)<< tmp_ime[b]. ime << endl;
			}
		}
		cout << "Raspored clanova: " << endl;
		for (int runda = 1;   runda <= runde;   runda++){
			cout << "Runda " << runda << ": " << endl;
			izlaz<< "Runda " << runda << ": " << endl;
			for (int i = 1,  j = n;   i <= n / 2;   i++,  j--){
				if (paran == false && j == n){ //ako je n neparan, svaki put
kad neki igrac zaigra s n+1 onda ispisuje da odmara
```

```
                                    cout << setw(5) << igrac1[i]. broj << ". " <<
igrac1[i]. ime << " se odmara" << endl;
                                    izlaz<< setw(5) << igrac1[i]. broj << ". " <<
igrac1[i]. ime << " se odmara" << endl;
                              }
                              else{
                                    cout << setw(5) << igrac1[i]. broj << ". " <<
igrac1[i]. ime << " vs " << igrac2[j]. broj << ". " << igrac2[j]. ime << endl;
                                    izlaz << setw(5) << igrac1[i]. broj << ". " <<
igrac1[i]. ime << " vs " << igrac2[j]. broj << ". " << igrac2[j]. ime << endl;
                              }

                              igrac1[i]. broj += (n / 2);
                              if (igrac1[i]. broj > n - 1){
                                    igrac1[i]. broj -= (n - 1);
                              }

                              for (int k = 1;   k <= n;   k++){ //petlja za usporedjivanje
rednog broja igraca radi zamjenjivanja imena

                                    if (igrac1[i]. broj == tmp_ime[k]. broj){
                                          igrac1[i]. ime = tmp_ime[k]. ime;
                                    }

                              }
                              if (j != n){ //n mora biti fiksiran te ako dodje na n, njega
ne zbraja s n/2

                                    igrac2[j]. broj += (n / 2);

                                    if (igrac2[j]. broj > n - 1){

                                          igrac2[j]. broj -= (n - 1);
                                    }
                                    for (int k = 1;   k <= n;   k++){

                                          if (igrac2[j]. broj == tmp_ime[k]. broj){
                                                igrac2[j]. ime = tmp_ime[k]. ime;
                                          }
                                    }
                              }

                        }
                  }
                  delete[]igrac1;
                  delete[]igrac2;
                  delete[]tmp_ime;

                  izlaz. close();


      system("pause");
      return 0;
}
```

5. 3 Programski kôd za magične kvadrate

```
#include <iostream>
#include <string>
#include <iomanip>
#include <fstream>
#include <stdlib.h>

using namespace std;

const int maks = 10000;

void zamjena1(int &a, int &b){ //ispred imena varijabli se stavlja operator refenciranja
cime se prosljedjuju originalne vrijednosti varijable,
	//a ne kopije jer bez tog operatora,vrijednosti se ne bi zamijenile
	int tmp = a;
	a = b;
	b = tmp;
}
bool prazno(int square[][maks], int r, int s){ //funkcija koja vraca true ako je polje
prazno ili false ako je puno

	if (!square[r][s]){

		return true;
	}
	return false;
}
void ispis_suma(int square[][maks], int n){

	ofstream izlaz;
	izlaz.open("Magic.txt"); //otvaranje textualne datoteke u koju ce se ispisavati
ispis matrice square[][]
	cout << "Broj elemenata magicnog kvadrata je " << n*n << endl << "Magicna suma je
" << ((n*n + 1) / 2.)*n << endl << endl;
	izlaz << "Broj elemenata magicnog kvadrata je " << n*n << endl << "Magicna suma je
" << ((n*n + 1) / 2.)*n << endl << endl;

	int a = 0;
	int c = n - 1;
	int suma_r = 0, suma_s = 0, suma_gl = 0, suma_sp = 0; //s lijeva na desno -> suma
retka,suma stupca,suma glavne dijagonale,suma sporedne dijagonale

	for (int i = 0; i < n; i++){
		for (int j = 0; j < n; j++){

			cout << setw(5) << square[i][j];
			izlaz << setw(8) << square[i][j];
		}
		cout << endl << endl;
		izlaz << endl << endl;
	}

	for (int i = 0; i < n; i++, a++, c--){ //u for petlju se smiju dodavati i druge
varijable koje ce se ponasati na definirani nacin svakim novim prolaskom kroz petlju
```

```
			suma_r = 0; //suma retka se svakim novim prolaskom kroz petlju resetira
			suma_s = 0; //suma stupca se svakim novim prolaskom kroz petlju resetira

			for (int j = 0; j < n; j++){

				suma_r += square[i][j]; //dohvacaju se svi elementi jednog stupca
				suma_s += square[j][i]; //dohvacaju se svi elementi jednog reda

			}
			suma_gl += square[a][i]; //svakim prolazom kroz petlju se prelazi u novi
red i novi stupac krenuvsi s lijeva na desno i s vrha prema dnu matrice
			suma_sp += square[c][i]; //svakim prolazom kroz petlju se prelazi u novi
red i novi stupac krenuvsi s desna na lijevo i s vrha prema dnu matrice


			cout << "Suma " << i + 1 << ". retka je " << suma_r << endl;
			cout << "Suma " << i + 1 << ". stupca je " << suma_s << endl;
			izlaz << "Suma " << i + 1 << ". retka je " << suma_r << endl;
			izlaz << "Suma " << i + 1 << ". stupca je " << suma_s << endl;

		}

		cout << "Suma glavne dijagonale je " << suma_gl << endl;
		cout << "Suma sporedne dijagonale je " << suma_sp << endl;
		izlaz << "Suma glavne dijagonale je " << suma_gl << endl;
		izlaz << "Suma sporedne dijagonale je " << suma_sp << endl;

}
void siamese(int square[][maks], int n, int br = 1, int r1 = 0, int s1 = 0){
		int r = r1; //r predstavlja mali magicni kvadrat u slucaju ako je n jednostruko
parni
		int tmp = br;  //tmp predstavlja aritemticki red
		int s = (n / 2) + s1; //treba poceti na polovici prvog reda malog kvadrata

		while (br < (n*n) + tmp){ //

			if (prazno(square, r, s)){
				square[r][s] = br; //ako je polje prazno,upisuje broj na to polje

				br += 1; //zatim se broj uvecava za 1
				r--; //red se smanjuje za jedan tj ide za jedno polje gore
				if (r < r1){ //ako izadje iznad malog kvadrata
					r = (r1 + n) - 1; //onda se provjera nastavlja u zadnjem retku
				}
				s++; //povecava stupac za jedan tj. pomice se udesno za jedno polje
				if (s >= s1 + n){ //ako izadje desno od malog kvadrata
					s = s1; //onda se provjera nastavlja u lijevom stupcu
				}

			}
			else{
				s--; //broj stupca se smanjuje za jedan tj.pomice se vertikalno
dolje

				if (s < s1){ //ako izadje lijevo od maloga kvadrata
					s = (s1 + n) - 1; //onda prelazi u stupac s druge strane
				}
```

```
                        r += 2; //red se povecava
                        if (r >= (r1 + n)){ //ako izadje ispod malog kvadrata
                                if (r == (r1 + n)){ //u slucaju ako izadje za jedno polje
van,onda se vraca na prvi red
                                        r = r1;
                                }
                                else{ //ako je za dva polja vani onda je u drugom redu
                                        r = r1 + 1; //onda prelazi u red s druge strane
                                }
                        }
                }
        }

}
void mag_kv_neparni(int square[][maks], int n){

        int br = 1; //magicni kvadrati ne moraju biti ispunjeni s brojevima od 1,2,3...n
nego mogu biti ispunjenji i brojevima izmedju kojih je razlika konstantna npr. 5,10,15..
ili 4,8,12,16
        siamese(square, n, br);
}
void mag_kv_dpl_parni(int square[][maks], int n){


        for (int i = 0; i < n; i += 4){ //predstavlja redove 4x4 kvadrata

                int gl_dij = 0, sp_dij = gl_dij + 3; //a je prvi element glavne dijagonale
dok je b prvi element sporedne dijagonale

                for (int j = 0; j < n; j += 4, gl_dij = j, sp_dij = j + 3){ //j se uvecava
za cetiri jer skace na pocetak svakog MALOG kvadrata u odredjenom redu,
                        //'a=j' jer se pocinje od nultog elementa tj. pocetka malog kvadrata
ili pocetka glavne dijagonale malog kvadrata,dok je b na kraju prvog reda malog kvadrata
tj. na pocetku sporedne dijagonale

                        int red_mk = i; //redovi malog kvadrata. Ici ce u svaki red malog
kvadrata

                        int sq = 0; //sq (square) sluzi kako bi prosao cetiri reda i cetiri
stupca i svaki put kad predje na novi mali kvadrat se resetira kako bi se cijeli mali
kvadrat popunio

                        while (sq < 4){ //while petlja ce se vrtiti sve dok nisu popunjene
glavna i sporedna dijagonala malog kvadrata
                                square[red_mk][gl_dij] = 1; //posto je svaki mali kvadrat na
drugoj poziciji,treba paziti u kojem je redu i stupcu
                                square[red_mk][sp_dij] = 1;
                                gl_dij++, sp_dij--, red_mk++;
                                sq++;
                        }
                }
        }


        /*Princip je uglavnom da ide po petlji i trazi "puna" polja tj. ako je polje
jednako jedinici onda ga popunjava i tako do kraja matrice*/
        int br = 1; //predstavlja upisani broj.Inicijalizirana je na jedan je rod tuda
pocinje upis
```

```
	for (int i = 0; i < n; i++){ //prva for petlja koja ide po redovima

		for (int j = 0; j < n; j++){ //druga for petlja koja ide po stupcima

			if (!prazno(square, i, j)){ //provjerava ako je polje jednako
jedinici tj. ako je popunjeno
				square[i][j] = br;
			}
			br++;
		}
	}
	/*Ovdje krece opet od pocetka matrice sa najvecim neiskoristenim brojem te trazi
prazna polja i u njih upisuje brojeve sve do nule*/
	for (int i = 0; i < n; i++){
		for (int j = 0; j < n; j++){
			br--;
			if (prazno(square, i, j)){ //provjerava ako je polje jednako nuli
tj. ako nije popunjeno
				square[i][j] = br;
			}

		}
	}
}
void mag_kv_singl_parni(int square[][maks], int n){

	int mali = n / 2; //mali kvadrati.Velika se matrica rastavlja na cetiri (n/2) *
(n/2) kvadrata
	int br; //varijabla cija ce se vrijednost preslikavati u polja malih kvadrata

	// prvi kvadrat

	br = 1; //od 1 do n^2/4

	siamese(square, mali, br);

	//cetvrti kvadrat
	br = ((n*n) / 4) + 1; // od n^2/+1 do (n^2)/2
	siamese(square, mali, br, mali, mali);

	//drugi kvadrat

	br = ((n*n) / 2) + 1; //od (n^2)/2+1 do 3(n^2)/4
	siamese(square, mali, br, 0, mali);

	//treci kvadrat

	br = 3 * n*n / 4 + 1; //(3(n^2)/4)+1 do n^2
	siamese(square, mali, br, mali);


	int k = ((n - 2) / 4 - 1); //n=4k+2 -->k ce predstavljati broj prvih i zadnjih
stupaca kojima ce se elementi zamijeniti

	//zamjena elemenata prvih k stupaca prvog i treceg kvadrata
	for (int stupac = 0; stupac <= k; stupac++){
```

```
			for (int r1 = 0, r2 = mali; r1 < mali && r2 < n; r1++, r2++){ //r1
predstavlja redove A kvadrata dok r2 predstavlja redove D kvadrata

				zamjena1(square[r1][stupac], square[r2][stupac]); //zamjena brojeva
prvog i treceg kvadrata

			}
		}

		if (n == 6){ //ako je n jednak 6 onda je k jednak nuli ali kod nizova nula
predstavlja prvi element npr. prvi red ili prvi stupac
			zamjena1(square[k + 1][k], square[k + mali + 1][k]); //zamjenjuju se
element srednjeg reda prvog i treceg malog kvadrata prvog stupca tj.vracaju se na pocetnu
poziciju
			zamjena1(square[k + 1][k + 1], square[k + mali + 1][k + 1]); //element
srednjeg reda malog kvadrata koji se nalazi na glavnoj dijagonali, se zamjenjuje s
elementom na sporednoj dijagonali istog reda treceg (ispod) kvadrata

		}
		else{
			zamjena1(square[k + 1][k - 1], square[k + mali + 1][k - 1]); //zamjenjuju
se element srednjeg reda prvog i treceg malog kvadrata prvog stupca jer su se zamijenili
svi elementi prvih k stupaca
			zamjena1(square[k + 1][k + 1], square[k + mali + 1][k + 1]); //element
srednjeg reda malog kvadrata koji se nalazi na glavnoj dijagonali, se zamjenjuje s
elementom na sporednoj dijagonali istog reda treceg (ispod) kvadrata
		}

		//zamjena zadnjih k-1 stupaca
		for (int stupac = n - 1; stupac >= n - k; stupac--){
			for (int r1 = 0, r2 = mali; r1 < mali && r2 < n; r1++, r2++){
				zamjena1(square[r1][stupac], square[r2][stupac]);
			}
		}
}
void magic_squares(){
		int n;

		cout << "Unesi n poredak magicnih kvadrata" << endl;
		cin >> n;

		int(*square)[maks] = new int[n][maks];

		for (int i = 0; i < n; i++){
			for (int j = 0; j < n; j++){
				square[i][j] = 0;
			}

		}
		cout << endl << endl;

		if (n % 2 != 0){ //ako je n neparan

			mag_kv_neparni(square, n);
		}
		else if (n % 2 == 0 && n % 4 == 0){ //ako je n djeljiv s 2 i sa 4 tj. ako je duplo
parni
```

```
			mag_kv_dpl_parni(square, n);
		}
		else {

			mag_kv_singl_parni(square, n); //ako je djeljiv samo s dva tj. jednostruko
parni
		}
		ispis_suma(square, n); //funkcija u kojoj se ispisuje magicni kvadrat te zbroj
svakog reda, stupca i dijagonala
}

int main(){

		magic_squares();

		system("pause");
		return 0;
}
```

5. Zaključak

U ovom radu je prikazano koliko je zapravo čovjek ograničen bez kompjutera ali i koliko je kompjuter ograničen bez čovjeka. Za prvi primjer je prikazano da je broj mogućih kombinacija kraljica na 8 x 8 ploči nekoliko milijardi, a postoje samo 92 rješenja. Čovjek je ograničen bez kompjutera zato jer bi čovjeku trebalo nekoliko godina dok bi našao sve kombinacije dok kompjuter u roku od nekoliko sekundi pronalazi sve moguće kombinacije za bilo koju dimenziju (ako je $n < 20$). Isto tako je kompjuter ograničen jer bez čovjeka koji će kreirati i upisati algoritam, nikada ne bi bila pronađena rješenja navedenih problema.

Kao što je spomenuto u uvodu, kombinatorika je doživjela nagli rast tek u posljednjih 30-40 godina kad je i tehnologija procvjetala. Ako bi se testirao bilo koji program koji rješava neki kombinatorni problem na kompjuteru prije 30 godina onda bi se taj program izvršavao nekoliko sati za razliku od današnjih kompjutera koji takve probleme rješavaju u roku nekoliko sekundi. Obični kućni kompjuteri više nisu dovoljno jaki za rješavanje nekih problema kao što je na primjer simulacija svemira u svrhu shvaćanja tamne materije i tamne energije gdje treba simulirati ponašanje nekoliko milijuna galaksija od kojih svaka ima jedinstvena svojstva zajedno sa svojstvima milijardi zvijezda od kojih se galaksije sastoje. To je veliko opterećenje za kompjuter i takve simulacije se izvršavaju sati i danima. Zbog toga su proizvedeni kvantni kompjuteri koji tako kompleksne probleme prevode mnogo brže i uz to se primjenjuju kvalitetniji algoritmi kojima se nastoji umanjiti vrijeme računanja i povećati efikasnost i ekonomičnost simulacije.

6. Literatura

[1] Wikipedija, Kombinatorika, https://hr.wikipedia.org/wiki/Kombinatorika, pregledano: 25. Lipanj 2015.

[2] Element d. o. o. *Demistificirani C++* (Četvrto izdanje 2014.). Boris Motik i Julijan Šribar, 1997.

[3] Wikipedia, C++, https://en.wikipedia.org/wiki/C%2B%2B, pregledano: 10.Lipanj 2015.

[4] Nicholas A. Solter i Scott J. Kleper *Proffesional C++*, http: //edc. tversu. ru/elib/inf/0146. pdf, pregledano: 20. Lipanj 2015.

[5] Igor's Website - Articles - Recursion tutorial, N-queens problem, http: //www. igorsevo. com/Article. aspx?article=Recursion+tutorial%2C+N-queens+problem, pregledano: 24. Lipanj 2015.

[6] Data Genetics, Blog *Eight queens problem,* http: //www. datagenetics. com/blog/august42012/, pregledano: 15. Lipanj 2015.

[7] Wikipedia, Round – robin tournament, http: //en. wikipedia. org/wiki/Round-robin_tournament, pregledano: 25. Lipanj 2015.

[8] 1728 Software Systems, Magic squares, http: //www. 1728. org/magicsq1. htm, pregledano: 30. Lipanj 2015.

[9] wikiHow – How to do anything, How to solve a Magic Square, http: //www. wikihow. com/Solve-a-Magic-Square, pregledano: 01. Srpanj 2015.

[10] *On the Construction of Doubly Even Order Magic Squares*, Grasha Jacob, Dr. A. Murugan, http://arxiv.org/ftp/arxiv/papers/1402/1402.3273.pdf, pregledano: 02. Srpanj 2015.