Određivanje kvadrata u programskom jeziku C

Poznić, Luka

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

Sveučilišni prijediplomski studij Računarstvo

ODREĐIVANJE KVADRATA U PROGRAMSKOM JEZIKU C

Završni rad

Luka Poznić

Osijek, 2024.

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

Obrazac Z1P: Obrazac za ocjenu završnog rada na sveučilišnom prijediplomskom studiju

Ocjena završnog rada na sveučilišnom prijediplomskom studiju

Ime i prezime pristupnika:	Luka Poznić
Studij, smjer:	Sveučilišni prijediplomski studij Računarstvo
Mat. br. pristupnika, god.	4577, 28.10.2021.
JMBAG:	0152207788
Mentor:	doc. dr. sc. Vinko Petričević
Sumentor:	prof. dr. sc. Tomislav Keser
Sumentor iz tvrtke:	
Naslov završnog rada:	Određivanje kvadrata u programskom jeziku C
Znanstvena grana završnog rada:	**Procesno računarstvo (zn. polje računarstvo)**
Zadatak završnog rada:	U brojnim matematičkim/kriptografskim problemima javlja se potreba testiranja puno brojeva, jesu li su oni kvadrati nekih drugih brojeva? U ovome radu ćemo opisati nekoliko brzih algoritama, implementiranih u C-u, koji daju odgovor na prethodo pitanje u slučaju da promatramo rješenja u skupu cijelih brojeva i u skupu ostataka modulo P (a takvi skupvi često imaju primjenu u kriptoanalizi i sličnim problemima).

Datum prijedloga ocjene završnog rada od strane mentora:	25.09.2024.
Prijedlog ocjene završnog rada od strane mentora:	Izvrstan (5)
Datum potvrde ocjene završnog rada od strane Odbora:	27.09.2024.
Ocjena završnog rada nakon obrane:	Izvrstan (5)
Datum potvrde mentora o predaji konačne verzije završnog rada čime je pristupnik završio sveučilišni prijediplomski studij:	29.09.2024.

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

IZJAVA O IZVORNOSTI RADA

Osijek, 29.09.2024.

Ime i prezime Pristupnika:	Luka Poznić
Studij:	Sveučilišni prijediplomski studij Računarstvo
Mat. br. Pristupnika, godina upisa:	4577, 28.10.2021.
Turnitin podudaranje [%]:	12

Ovom izjavom izjavljujem da je rad pod nazivom**: Određivanje kvadrata u programskom jeziku C**

izrađen pod vodstvom mentora doc. dr. sc. Vinko Petričević

i sumentora prof. dr. sc. Tomislav Keser

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

Potpis pristupnika:

SADRŽAJ

1. UVOD

U ovom završnom radu izrađen je C program koji se koristi za provjeru da li su dani cijeli brojevi kvadrati drugih cijelih brojeva. Proces je realiziran pomoću uređivača koda Microsoft Virtual Studio Code, verzije 2019, na sustavu Windows 10.

Konfiguracija računala je:

- Procesor: Intel Core i5 4690K @ 3.50GHz
- RAM: 16,0GB Dual-Channel DDR3 @ 798MHz
- Matična ploča: ASUSTeK COMPUTER INC. H81-PLUS (SOCKET 1150)
- Grafička kartica: NVIDIA GeForce GTX 1050 Ti

Na danoj konfiguraciji unutar programa su postavljene dvije funkcije, jedna koja koristi modulo P, te jedna koja redom provjerava redom sve brojeve. Osim tih programa su postavljene i štoperice za vremenska mjerenja trajanja pojedinačnih funkcija sa danim brojevima. Kvadrate računamo Newtovom metodom aproksimacije rješenja jednadžbi. Najveći problem pri ovakvim zadatcima je pri računanju kvadrata sa visokim brojevnim redom (veći od 10^{10}) zbog nepreciznosti i limitacijama tipova podataka.

Nakon uvodnog poglavlja će zadatak ovog završnog rada biti pojašnjen, dok poglavlje 2 opisuje programski jezik C. Detaljno pojašnjenje programa se nalazi u 3. poglavlju. Kroz riješenje danog zadatka prolazimo u poglavlju 4, te će zaključak ponuditi osvrt na ciljeve zadatka, te dobivene rezultate.

1.1. Zadatak završnog rada

U ovome radu su opisani dva algoritma, koje možemo koristiti pri pojavi raznih kripografskih ili matematičkih problema gdje moramo testirati veliku grupaciju brojeva, te da li su oni kvadrati drugih brojeva, velikom brzinom. Algoritmi su implementirani u C programskom jeziku, koji daju odgovor na prethodo pitanje u slučajevima da se promatraju rješenja u skupu cijelih brojeva i u skupu ostataka modulo P.

2. PROGRAMSKI JEZIK C

U ovom poglavlju biti će opisan programski jezik C, koji je programski jezik opće namjene. Autor ovog jezika je Denis Ritchie koji ga je razvio u istraživačkom centru Bell Laboratoriesa u New Jersey-u ranih 70-ih godina 20. stoljeća. Značajan doprinos nastanku ovog programskog jezika su dali i Ken Thompson koji je autor programskog jezika B, te Martin Richards koji je autor programskog jezika BCPL.

Razvijen je u namjeri da rješava probleme koji su rješavani u ASSEMBLER-u, ali da si omoguće i prednosti jezika više razine. Da su uspjeli u svojoj namjeri, pokazuje to da se u ovaj programski jezik mogu uključivati naredbe pisane u ASSEMBLER-u, što ga čini vrlo pogodnim za pisanje sistemskih programa. Također se smatra i jednim od najvažnijih programa komercijalne računalne industrije jer je jedini programski jezik prilagođen za sve računalne platforme.

Tijekom 1980-ih je Bjarne Stroustrup sa drugim istraživačima u Bell Labs-u proširio C programski jezik dodavajući mnoge mogućnosti objekto orijentiranog programiranja, nazivaju novi programski jezik C++. Također je C++ naslijedio i mnoge probleme C programskog jezika.

2.1. Struktura programa u C-u

C programski jezik omogućava strukturirano programiranje, gdje rastavljamo probleme u niz manjih povezanih cijelina. Svaki C program je sastavljen od jedne ili više funkcija, a glavna i jedina obavezna funkcija u ovom programskom jeziku je main(). Početak i kraj funkcije se označava sa vitičastim zagradama, a nakon naredbi se mora koristiti ; za kraj trenutne i početak iduće naredbe.

2.1.1. Osnovni skup znakova u C programiranju

Osnovni skup znakova u programskom jeziku C su:

1. Velika i mala slova engleske abecede A-Z i a-z,
2. Numerički znakovi – dekadske znamenke 0-9,
3. Posebni znakovi u koje spadaju matematički simboli, razmak i mnogi drugi.

Pojavom standarda C11 sada je moguće umetnuti i posebne znakove (UTF8).

2.1.2. Ključne riječi i identifikatori

Programski jezik C ima samo 32 ključne riječi u svom začeću, te sa kasnijim standardima se dodaje nekolicina novih sa novim standardima.

Ključne riječi se koriste za definiciju varijabli, razne operacije nad varijablama i ne mogu se koristiti kao identifikatori.

Identifikatori su imena koje se dodijeljuju varijablama, funkcijama i sl. Sastoje se od alfanumeričkih znakova (brojki i slova). Razlikuju se ako u prvih 32 znaka postoji razlika među znakovima.

2.1.3. Pretprocesorske naredbe

Na vrhu svakog C programa se uobičajeno pišu pretprocesorske naredbe koje pozivaju biblioteke funkcija koje se rabe u programu. Mogu se zapisati u obliku: #naredba parametri. Najčešće upotrebljavana pretprocesorska naredba je: #include <ime datoteke>, koja se obavezno piše prije početka funkcije main(). Prevođenjem programa se na lokaciji zapisa #include kopira sadržaj navedene datoteke ili biblioteke, pa su sadržaji datoteka spremni za koristiti.

2.1.4. Tipovi podataka

U programskom jeziku možemo koristiti pet različitih tipova podataka. Ti različiti tipovi podataka su: znakovni, cjelobrojni, realni, realni dvostruke preciznosti i tip podataka koji ne sadrži vrijednost. Ključne riječi koje ih određuju su char, int, float, double i void. Tip podataka koji ne sadrži vrijednosti se često koristi kod provjere tipova podataka, za definiranje funkcija bez argumenata i funkcije koja ne vraća vrijednost.

Efikasan program ovisi o biranju idealnog tipa podatka za program na kojem radimo, te koji podatak se predaje varijabli. Dozvoljene su sve vrste pretvaranja tipova podataka, a u izrazima su dovoljne različite kombinacije tipova.

Osim osnovnih tipova podataka postoje i složeniji tipovi podataka poput polja, struktura, polja struktura, unija i polja bitova. Postoji i mogućnost definiranja vlastitih tipova podataka.

Redni broj	*Tip podataka*	*Vrsta*	*Predznak*	*Duljina (broj bajtova)*	*Opseg*
1	int	short	signed	2	-32768 do 32767
			unsigned	2	0 do 65.535
		Int	signed	4	-2,147,483,647 do 2,147,483,647
			unsigned	4	0 do 4,294,967,295
		long	signed	4	–2.147.483.648 do 2.147.483.647
				8	-9,223,372,036,854,775,807 do 9,223,372,036,854,775,807
			unsigned	4	0 do 4.294.967.295
				8	0 do 18,446,744,073,709,551,615
		long long	signed	8	–9.223.372.036.854.775.808 do 9.223.372.036.854.775.807
			unsigned	8	0 do 18.446.744.073.709.551.615
2	float			4	-3.4·10−38 do 3.4·1038
3	double			8	-1.7·10−308 do 1.7·10308
4	char	long	signed	1	-128 do 127
			unsigned	1	0 do 255
5	bool			1	true (1) ili false (0)
6	void			0	bez vrijednosti

Tablica 2.1 **Tipovi podataka**

Ovisno o operacijskim sustavima se mjenja duljina tipa podataka long int, u slučaju da je 64-bitni linux sustav je veličine 8 bajtova, te različitih opsega ovisno o tome da li je signed ili unsigned. [1]

Da bi se tipovi podataka ispisali potrebna nam je funkcija printf() koja je oblika:

Linija ***Kod***

```
1:      printf("Hello, World!");
```

Funkcija. 2.1. **Primjer printf() funkcije**

A za ispis željenog tipa podatka se treba upisati oznaku formata za ispis:

	Format	*Značenje*
1	%c	oznaka formata za ispis jednog znaka
2	%d	ispis cijelog broja
3	%f	ispis realnog broja
4	%lf	ispis realnog broja tipa double
5	%s	ispis niza znakova
6	%e	ispis realnog broja u E-notaciji

Tablica 2.2. **Oznake formata**

Ispis se uređuje unutar navodnika, izvan navodnika se navode sve varijable čije vrijednosti se trebaju ispisati. Oznake formata trebaju odgovarati redoslijedu danih varijabli. Pogrešno zadan format upisa će rezultirati pogrešnim ispisom.

3. METODA RAČUNANJA KVADRATA

Od programa se traži da što efikasnije i brže sazna da li je dani cijeli broj kvadrat drugog cijelog broja. Da bi se pobrinuli za ovo moramo implementirati nekoliko optimizacija da eliminiramo što više brojeva koji nisu u mogućnosti ispuniti kriterij. Za implementaciju i izradu optimiziranog algoritma smo iskoristili skriptu A. Dujella, *Teorija brojeva u kriptografiji*, 2003. [1]

3.1. Algoritam za računanje korijena

Da bi pronašli cijeli dio kvadratnog korijena danog broja koristimo Newtonovu metodu, koja će nam dati približno računanje korijena jednadžbe. Riješenja jednadžbe *f(x)=0* se računaju po formuli:

$$x_{k+1} = x_k - \frac{f(x_k)}{f'(x_k)}$$

Jednadžba 3.1 **Newtonova metoda**

Složenost danog algoritma je $O(ln^3 n)$.

Linija ***Kod***

```
1:      long long integerSquareRoot(long long n) {
2:          long long x = n;
3:          long long y = (x + n / x) / 2;
4:          while (y < x) {
5:              x = y;
6:              y = (x + n / x) / 2;
7:          }
8:          return x;
9:      }
```

Funkcija 4.1. **Izračunavanje cijelobrojnog dijela kvadratnog korijena**

Mi trebamo preformulirati jednadžbu u $f(x) = x^2 - n$. Cilj ove funkcije je pronaći kvadratni korijen broja n. Algoritam započinje sa aproksimacijom te koristi iterativnu formulu. Dana formula je u kodu implementirana i preformulirana pod nazivom funkcije *integerSquareRoot*.

3.2. Generiranje pripadnih tablica

Pripadne tablice generiramo u nadi eliminirati što više prirodnih brojeva koji nisu kvadrati. Treba iskoristiti činjenicu da ako je n potpun kvadrat, n je kvadratni ostatak modulo m za svaki m koji je relativno prost s n. Što znači da ako je n kvadratni neostatak, n sigurno nije kvadrat. Dakle izrađuje se pripadna tablica izradom nekoliko konkretnih modula koji se unaprijed izračunaju u pripadnom prstenu. Kombinacija modula u skripti je 64, 63, 65 i 11.

$$\frac{12}{64} \cdot \frac{16}{63} \cdot \frac{21}{65} \cdot \frac{6}{11} = \frac{6}{715} < 0.01$$

Jednadžba 3.2 **Kombinacija modula**

Redoslijed testiranja modula je od omjera broja kvadrata sa kombinacijom, pa redoslijedom provjeravamo: 64,63,65 i 11. Ako je učitan broj kvadratni ostatak za sve četiri module, to je velika vjerojatnost da je broj savršen kvadrat. Međutim, nije zajamčeno jer postoje i drugi brojevi koji zadovoljavaju uvijet. Ova funkcija je ključna jer eliminira mnoge brojeve koji nisu savršeni kvadrati velikom brzinom. Kasnije se priziva, te je ključna u provjeri savršenih kvadrata.

Funkcija u programskom kodu:

Linija ***Kod***

```
1:      void fillQuadraticResidues() {
2:          for (int i = 0; i < 64; i++) q64[(i * i) % 64] = 1;
3:          for (int i = 0; i < 63; i++) q63[(i * i) % 63] = 1;
4:          for (int i = 0; i < 65; i++) q65[(i * i) % 65] = 1;
5:          for (int i = 0; i < 11; i++) q11[(i * i) % 11] = 1;
6:      }
```

Funkcija 4.2. **Popunjavanje pripadne tablice**

3.2.1. Primjer

Ako pozovemo funkciju s n = 25:

1. Nulta iteracija: x = 25, y = (25 + 25 / 25) / 2 = 13
2. U prvoj iteraciji: x = 13, y = (13 + 25 / 13) / 2 ≈ 7
3. U drugoj iteraciji: x = 7, y = (7 + 25 / 7) / 2 ≈ 5
4. U trećoj iteraciji: x = 5, y = (5 + 25 / 5) / 2 = 5

Petlja se zaustavlja jer više nema poboljšanja, pa funkcija vraća 5, što je kvadratni korijen od 25.

3.3. Detekcija kvadrata

Za detektiranje savršenog kvadrata su implementirane dvije funkcije unutar programa, jedna optimizira kod pomoću kvadratnih ostataka. Druga funkcija je implementirana pomoću funkcije *sqrt()* iz *math.h* biblioteke.

3.3.1. Sqrt metoda provjere

Funkcija *notModuloP* provjerava da li je učitani broj n savršen kvadrat. Koristi standardni matematički pristup izračunavanja tako da kvadrira *sqrt(n)* i uspoređuje ga sa n. Prva provjera prije izračuna je da li je broj n veći od 0. Nakon izračunavanja kvadratnog korijena broja n, pretvara ga u cijeli broj, te provjerava da li je kvadrat tog broja jednak originalnom broju. Ako je jednak onda je savršeni kvadrat.

Linija **Kod**

```
1:      int notModuloP(long long n) {
2:          if (n < 0) {
3:              return 0;
4:          }
5:
6:          double sqrt_n = sqrt(n);
7:          long long int_sqrt_n = (long long)sqrt_n;
8:
9:          return (int_sqrt_n * int_sqrt_n == n);
10:     }
```

Funkcija 3.3. **Brute force funkcija**

3.3.2. Primjer sqrt metode

1. Poziv funkcije s n = 16:

   sqrt_n = sqrt(16) = 4.0

   int_sqrt_n = (long long) 4.0 = 4

   Provjera: 4 * 4 == 16 → vraća 1 (broj 16 je savršeni kvadrat).

2. Poziv funkcije s n = 20:

   sqrt_n = sqrt(20) ≈ 4.472

   int_sqrt_n = (long long)4.472 = 4

   Provjera: 4 * 4 == 20 → vraća 0 (broj 20 nije savršeni kvadrat).

3.3.3. Metoda kvadratnih ostataka

Većina funkcija koje su nam potrebne za upotrebu ove metode su implementirana u poglavljima iznad. Potrebna je jedino implementacija provjera pomoću tablica kvadratnih ostataka. Prvo, provjerava se je li n negativan. Ako jest, funkcija vraća 0. Zatim se provjerava je

li kvadratni ostatak n modula 64 u tablici q_{64}. Ako nije, funkcija vraća 0. Isti postupak se ponavlja za module 45045, 65 i 11. Modul 45045 koristimo jer je zajednički višekratnik brojeva 63, 65 i 11. Koristi se jer savršeni kvadrati imaju specifične obrasce prilikom dijeljenja s 63, 65 i 11. Nizovi q63, q65 i q11 sadrže informacije o tome koji ostaci pri djeljenju sa 63,65 i 11 mogu biti savršeni kvadrati. Ako broj n ne zadovoljava jedan od danih uvijeta, funkcija vraća uvijet jer broj sigurno nije savršen kvadrat. Na kraju ako prođe sve provjere izračun kvadratnog korijena je zadnji korak.

Značajno ubrzava proces korištenjem modularnih operacija. Većina brojeva bude eliminirana prije izračuna kvadratnog korijena. Ako prođe i sve modularne provjere, izračuna se kvadratni korijen cijelog broja, te se vrši konačna provjera da li je n savršen kvadrat.

Linija	**Kod**
1:	`int isPerfectSquare(long long n) {`
2:	`    if (n < 0) return 0;`
3:	`    long long t = n % 64;`
4:	`    if (q64[t] == 0) return 0;`
5:	
6:	`    long long r = n % 45045;`
7:	`    if (q63[r % 63] == 0) return 0;`
8:	`    if (q65[r % 65] == 0) return 0;`
9:	`    if (q11[r % 11] == 0) return 0;`
10:	
11:	`    long long sqrt_n = integerSquareRoot(n);`
12:	`    return (sqrt_n * sqrt_n == n);`
13:	`}`

Funkcija 3.4. **Provjera savršenog kvadrata**

3.4. Main() funkcija

Unutar main() funkcije se pozivaju funkcije za popunjavanje tablica kvadratnih ostataka, te se inicijaliziraju početni i krajnji brojevi (*startNumber* i *endNumber*) koji će biti skup nad kojim će program odraditi i mjeriti vrijeme trajanja računice. Vrijeme se mjeri za sqrt metodu i metodu modulo P koji i ispisuju sve kvadrate koje su izračunali. Pomoću clock() funkcije zadajemo

početno i krajnje vrijeme. Osim što ispisujemo brojeve koji su kvadrati, ispisuje se i ukupno trajanje pojedinačne funkcije, te broj savršenih kvadrata koji su pronađeni za pojedinačne funkcije da bi na kraju mogli usporediti rezultate.

Linija **Kod**

```
1:      int main() {
2:          fillQuadraticResidues();
3:
4:          long long startNumber = 1000000;
5:          long long endNumber = 10000000000;
6:
7:
8:          int countModulo = 0;
9:          int countRegular = 0;
10:
11:         startModulo = clock();
12:         for (long long number = startNumber; number <= endNumber; number++) {
13:             if (isPerfectSquare(number)) {
                    printf("Broj %lld je kvadrat nekog broja. (Modulo P)\n",
14:     number);
15:                 countModulo++;
16:             }
17:         }
18:         endModulo = clock();
19:         moduloTimer = ((double)(endModulo - startModulo)) / CLOCKS_PER_SEC;
20:
21:         printf("Kraj modulo P metode\n");
22:
23:         startNormal = clock();
24:         for (long long number = startNumber; number <= endNumber; number++) {
25:             if (notModuloP(number)) {
                    printf("Broj %lld je kvadrat nekog broja. (Normalna
26:     metoda)\n", number);
27:                 countRegular++;
28:             }
29:         }
30:         endNormal = clock();
31:         regularTimer = ((double)(endNormal - startNormal)) / CLOCKS_PER_SEC;
32:
33:         printf("Kraj obične metode\n");
34
35:         printf("Modulo P metoda je trajala %f sekundi.\n", moduloTimer);
36:         printf("Obicna metoda je trajala %f sekundi.\n", regularTimer);
37:
            printf("Modulo P metoda je pronasla %d kvadratnih brojeva.\n",
38:     countModulo);
            printf("Obicna metoda je pronasla %d kvadratnih brojeva.\n",
39:     countRegular);
```

```
40:
41:         return 0;
42:     }
```

Funkcija 3.5. **Main() funkcija**

4. REZULTATI PROGRAMA

4.1. Mjerenja

Odrađena su dva mjerenja, prvo za običnu metodu, te drugo za metodu modulo p. Mjerenje vremena se odvija za isti skup brojeva koji se sastoji od raspona brojeva između 1000000 i 10000000000. Obje metode redom prolaze kroz skup, te izbacuju u konzolu koji brojevi su kvadrati. Na kraju računanja za obje funkcije se ispiše konačno ukupno vrijeme trajanja, u sekundama, operacija prve i druge funkcije na usporedbu (*regularTimer* i *moduloTimer*). Uz vrijeme se ispisuje koliko je ukupno bilo kvadrata tijekom računanja (*countRegular* i *countModulo*).

4.2. Rezultati mjerenja

Za skup brojeva koji je pripremljen u main() funkciji program i ispisuje brojeve koji su kvadrati, te uz to kojom metodom su pronađeni.

Linija **Kod**

```
1:      Broj 55056400 je kvadrat nekog broja. (Modulo P)
2:      Broj 55071241 je kvadrat nekog broja. (Modulo P)
3:      Broj 55086084 je kvadrat nekog broja. (Modulo P)
4:      Broj 55100929 je kvadrat nekog broja. (Modulo P)
5:      Broj 55115776 je kvadrat nekog broja. (Modulo P)
6:      Broj 55130625 je kvadrat nekog broja. (Modulo P)
7:      Broj 55145476 je kvadrat nekog broja. (Modulo P)
8:      Broj 55160329 je kvadrat nekog broja. (Modulo P)
9:      Broj 55175184 je kvadrat nekog broja. (Modulo P)
10:     Broj 55190041 je kvadrat nekog broja. (Modulo P)
```

Linija **Kod**

```
1:      Broj 214388164 je kvadrat nekog broja. (Normalna metoda)
2:      Broj 214417449 je kvadrat nekog broja. (Normalna metoda)
3:      Broj 214446736 je kvadrat nekog broja. (Normalna metoda)
4:      Broj 214476025 je kvadrat nekog broja. (Normalna metoda)
```

```
5:      Broj 214505316 je kvadrat nekog broja. (Normalna metoda)
6:      Broj 214534609 je kvadrat nekog broja. (Normalna metoda)
7:      Broj 214563904 je kvadrat nekog broja. (Normalna metoda)
8:      Broj 214593201 je kvadrat nekog broja. (Normalna metoda)
9:      Broj 214622500 je kvadrat nekog broja. (Normalna metoda)
10:     Broj 214651801 je kvadrat nekog broja. (Normalna metoda)
```

Linija **Kod**

```
1:      Obicna metoda je trajala 118.261000 sekundi.
2:      Modulo P metoda je trajala 153.229000 sekundi.
3:      Obicna metoda je pronasla 99001 kvadratnih brojeva.
4:      Modulo P metoda je pronasla 99001 kvadratnih brojeva.
```

Rezultati 4.1. **Prvo mjerenje**

Linija **Kod**

```
1:      Obicna metoda je trajala 125.594000 sekundi.
2:      Modulo P metoda je trajala 154.030000 sekundi.
3:      Obicna metoda je pronasla 99001 kvadratnih brojeva.
4:      Modulo P metoda je pronasla 99001 kvadratnih brojeva.
```

Rezultati 4.2. **Drugo mjerenje**

Linija **Kod**

```
1:      Modulo P metoda je trajala 153.500000 sekundi.
2:      Obicna metoda je trajala 122.943000 sekundi.
3:      Modulo P metoda je pronasla 99001 kvadratnih brojeva.
4:      Obicna metoda je pronasla 99001 kvadratnih brojeva.
```

Rezultati 4.3. **Treće mjerenje**

Linija **Kod**

```
1:      Kraj modulo P metode
2:      Kraj obične metode
3:      Modulo P metoda je trajala 166.663000 sekundi.
4:      Obicna metoda je trajala 117.249000 sekundi.
5:      Modulo P metoda je pronasla 99001 kvadratnih brojeva.
6:      Obicna metoda je pronasla 99001 kvadratnih brojeva.
```

Rezultati 4.4. **Četvrto mjerenje – bez ispisa**

Rezultati koje smo dobili prikazuju da su obje metode dobile točne rezultate sa količinom pronađenih kvadratnih brojeva u danom skupu sa 99001 kvadratnim brojem.

Dok vremenski se ukazuje primjetna razlika između metode koja koristi sqrt() funkciju i modulo p metodu.

Srednje vrijednosti:

1. Za sva četiri mjerenja srednje metode se dobije srednju vrijednost 121.01175 sekundi.

2. Za sva četiri mjerenja modulo p metode se dobije srednja vrijednost 156.8555 sekundi.

Što ukazuje da je obična metoda brža za 35.85475 sekundi, ako izračunamo srednju razliku u vremenu između metoda.

5. ZAKLJUČAK

Na temelju mjerenja i analize koda, obična metoda se pokazala bržom. U ovom konkretnom slučaju se iskazalo da dodatni proračuni i provjere unutar modulo P metode su poništili prednost optimizacije te funkcije.

Razlog za neočekivani rezultat može biti veličina skupa podataka koje smo učitali može biti značajnija u odnosu na jednostavniju metodu ili upravljanje i pristup tablicama kvadratnih ostataka može uvesti dodatni overhead koji poništava prednosti metode.

Rezultati naglašavaju važnost eksperimentalnog vrednovanja raznih algoritama u svrhu odabira optimalnog pristupa trenutnog problema.

LITERATURA

[1]“C Language Reference“, dostupno na: https://learn.microsoft.com/en-us/cpp/c-language/c-language-reference?view=msvc-170, [29. srpnja, 2024.]

[2] „Teorija brojeva u kriptografiji“, Andrej Dujella, dostupno na: https://web.math.pmf.unizg.hr/~duje/tbkript/tbkriptlink.pdf, [5. kolovoza, 2024.]

SAŽETAK

Naslov: Određivanje kvadrata u programskom jeziku C.

U ovom završnom radu istražujemo i implementiramo dva algoritma za provjeru je li dani cijeli broj kvadrat nekog drugog cijelog broja, koristeći programski jezik C. Prvi algoritam koristi modulo operacije za brzo eliminiranje brojeva koji ne mogu biti kvadrati, dok drugi algoritam primjenjuje Newtonovu metodu aproksimacije za precizno izračunavanje kvadratnog korijena. Razvijeni C programi su implementirani u razvojnom okruženju Microsoft Visual Studio Code na sustavu Windows 10. Programi uključuju dvije funkcije za provjeru kvadrata brojeva, zajedno s mjerenjem vremena izvršavanja svake funkcije. Kvadrati brojeva provjeravaju se Newtonovom metodom aproksimacije, koja je prilagođena za rad s velikim brojevima, unatoč problemima s preciznošću i ograničenjima tipova podataka u C-u. Rad također pokriva osnovne aspekte programskog jezika C, uključujući njegovu povijest, strukturu programa, ključne riječi, identifikatore, pretprocesorske naredbe i tipove podataka. Na kraju, rad pruža osvrt na ostvarene ciljeve, analizira dobivene rezultate i optimizaciju algoritama za provjeru kvadrata brojeva.

Ključne riječi: C programski jezik, algoritmi, kvadrat broja, modulo operacija, Newtonova metoda, Visual Studio Code

ABSTRACT

Title: Determining Squares in the C Programming Language

In this thesis, we explore and implement two algorithms to check whether a given integer is the square of another integer, using the C programming language. The first algorithm uses modulo operations to quickly eliminate numbers that cannot be squares, while the second algorithm applies Newton's method of approximation for accurately calculating the square root. The

developed C programs are implemented in the Microsoft Visual Studio Code development environment on a Windows 10 system. The programs include two functions to check square numbers, along with measuring the execution time of each function. Square numbers are checked using Newton's method of approximation, which is adapted to work with large numbers despite issues with precision and data type limitations in C. The paper also covers the basic aspects of the C programming language, including its history, program structure, keywords, identifiers, preprocessor directives, and data types. Finally, the paper provides a review of the achieved objectives, analyzes the obtained results, and optimization of algorithms for checking square numbers.

Keywords: C programming language, algorithms, square number, modulo operation, Newton's method, Visual Studio Code

PRILOG A. Program

```
#include <stdio.h>
#include <time.h>
#include <math.h>  // Dodano za sqrt funkciju

// Funkcija za izračunavanje cijelobrojnog dijela kvadratnog korijena
long long integerSquareRoot(long long n) {
    long long x = n;
    long long y = (x + n / x) / 2;
    while (y < x) {
        x = y;
        y = (x + n / x) / 2;
    }
    return x;
}

// Tablice kvadratnih ostataka za module 64, 63, 65 i 11
int q64[64] = { 0 };
int q63[63] = { 0 };
int q65[65] = { 0 };
int q11[11] = { 0 };

clock_t startModulo, endModulo, startNormal, endNormal;
double moduloTimer, regularTimer;
```

```
// Popunjavanje tablica kvadratnih ostataka
void fillQuadraticResidues() {
    for (int i = 0; i < 64; i++) q64[(i * i) % 64] = 1;
    for (int i = 0; i < 63; i++) q63[(i * i) % 63] = 1;
    for (int i = 0; i < 65; i++) q65[(i * i) % 65] = 1;
    for (int i = 0; i < 11; i++) q11[(i * i) % 11] = 1;
}

int isPerfectSquare(long long n) {
    if (n < 0) return 0;
    long long t = n % 64;
    if (q64[t] == 0) return 0;

    long long r = n % 45045;
    if (q63[r % 63] == 0) return 0;
    if (q65[r % 65] == 0) return 0;
    if (q11[r % 11] == 0) return 0;

    long long sqrt_n = integerSquareRoot(n);
    return (sqrt_n * sqrt_n == n);
}

// Funkcija koja koristi sqrt za provjeru je li broj kvadrat
int notModuloP(long long n) {
    if (n < 0) {
        return 0;  // Negativni brojevi nemaju kvadratne korijene
    }

    double sqrt_n = sqrt(n);  // Izračun kvadratnog korijena
    long long int_sqrt_n = (long long)sqrt_n;  // Pretvaranje u cijeli
broj

    return (int_sqrt_n * int_sqrt_n == n);
}

int main() {
```

```
    fillQuadraticResidues();     //  Popunjavanje  tablica  kvadratnih
ostataka

    long long startNumber = 1000000;
    long long endNumber = 10000000000;

    // Broji koliko je kvadrata pronađeno
    int countModulo = 0;
    int countRegular = 0;

    startModulo = clock();  // Vremensko mjerenje za Modulo P metodu
    for (long long number = startNumber; number <= endNumber; number++)
{
        if (isPerfectSquare(number)) {
            printf("Broj %lld je kvadrat nekog broja. (Modulo P)\n",
number);  // Ispis svakog kvadrata
            countModulo++;
        }
    }
    endModulo = clock();
    moduloTimer = ((double)(endModulo - startModulo)) / CLOCKS_PER_SEC;

    printf("Kraj modulo P metode\n"); //oznaka za kraj modulo P metode

    startNormal = clock();  // Vremensko mjerenje za običnu metodu
    for (long long number = startNumber; number <= endNumber; number++)
{
        if (notModuloP(number)) {
            printf("Broj  %lld  je  kvadrat  nekog  broja.  (Normalna
metoda)\n", number);  // Ispis svakog kvadrata
            countRegular++;
        }
    }
    endNormal = clock();
    regularTimer = ((double)(endNormal - startNormal)) / CLOCKS_PER_SEC;

    printf("Kraj obične metode\n");
```

```
    printf("Modulo P metoda je trajala %f sekundi.\n", moduloTimer);
    printf("Obicna metoda je trajala %f sekundi.\n", regularTimer);

    printf("Modulo P metoda je pronasla %d kvadratnih brojeva.\n",
countModulo);
    printf("Obicna metoda je pronasla %d kvadratnih brojeva.\n",
countRegular);

    return 0;
}
```

_____

Potpis autora