UNIVERZA V LJUBLJANI
FAKULTETA ZA RAČUNALNIŠTVO IN INFORMATIKO
FAKULTETA ZA MATEMATIKO IN FIZIKO

Jure Medvešek

# IMPLEMENTACIJA METODE INDEX CALCULUS ZA RAČUNANJE DISKRETNEGA LOGARITMA NAD PRAŠTEVILSKIMI OBSEGI

Seminarska naloga pri predmetu
kriptografija in teorija kodiranja 2

Mentor: prof, dr. Aleksandar Jurišić

Ljubljana, 2012

# 1 Uvod

Dandanes se vse več informacij prenaša preko interneta, ki je nezaščiten kanal in nam zato ne nudi tajnosti podatkov. Če tajnost želimo, lahko s pomočjo kriptografskih metod zakodiramo želene podatke in s tem napadalcu onemogočimo, da bi jih lahko prebral, četudi bi jih prestregel.

Kriptosistemi na osnovi računske nedosegljivosti diskretnega logaritma so eden pomembnejših delov kriptografije z javnimi ključi. Skozi nalogo bomo spoznali napad nanj v grupi $(\mathbb{Z}_p^*, *)$, zaradi česar se danes vse več uporablja kriptosisteme, ki temeljijo na eliptičnih krivuljah, saj zanje ne poznamo učinkovitih metod za izračun diskretnega logaritma.

Predstavili bomo nekaj učinkovitih metod za izračun diskretnega logaritma v splošnih grupah in implementirali knjižico, ki izkorišča ranljivost družine $(\mathbb{Z}_p^*, *)$ na metode tipa index calculus.

# 2 Problem diskretnega logaritma

Preden definiramo diskretni logaritem podajmo še nekaj definicij z algebraičnih struktur.
Naj bo $(G, \circ)$ končna ciklična grupa z enoto $e$ in elementom $g \in G$. Nad njo definiramo naslednje pojme:
(1) $|G|$ : število elementov v grupi
(2) $\exists e \in G : \forall t \in G : t \circ e = t$ : enota
(3) $\forall t : t \circ t^{-1} = e$ : inverz
(4) $g^m = g \circ g \circ \ldots \circ g = g^{m-1} \circ g ; g^0 = e$ : potenca
(5) $\mathrm{ord}(g) = \min\{m ; m \in \mathbb{N} \wedge g^m = e\}$ : red
(6) $PG = \{g ; \mathrm{ord}(g) = |G|\}$ : množica vseh možnih generatorjev za grupo $G$

Iz definicije direktno sledi:
(7) $\forall g \in PG : g^{\mathrm{ord}(g)} = e$

Sedaj definirajmo preslikavo $\log_g x : G \to \mathbb{Z}_{\mathrm{ord}(g)}$, za katero velja:
(8) $\log_g x = y ; g^y = x$

V primeru, ko je $g$ generator množice $G$ ( $m = \mathrm{ord}(g)$ ), je funkcija $\log_g x$ bijektivna in inverzna s potenciranjem $g^x$. To je ena izmed lastnosti, ki je skupna diskretnemu logaritmu in logaritmu v realnih številih. Nekatere izmed ostalih skupnih lastnosti so:
(9) $\log_g g = 1$
(10) $\log_g(x \circ y) \equiv \log_g x + \log_g y \pmod m$
(11) $\log_g x^y \equiv y * \log_g x \pmod m$

Modul pri enačbah (10, 11) sledi iz enačbe (7).

## 2.1 Nekaj primerov grup

### Grupa $(\mathbb{Z}_p, +)$ :

$G=\{0,1,...,p-1\}$ ; $g\circ h:=g+h \pmod p$

$|G|=p$ ; $e=0$ ; $g^m:=g*m \pmod p$ ;

$\forall g: g\neq 0 \Rightarrow \text{ord}(g)=|G|=p$ ; $PG=\{1,...,p-1\}$ ;

Diskretni logaritem za poljuben element $x$ učinkovito izračunamo, saj zanj poznamo ugodno faktorizacijo:

$\log_g x \equiv \log_g(1\circ 1 ... \circ 1) \equiv \log_g 1^x \equiv x*\log_g 1 \pmod p$

$\log_g 1 \equiv g^{-1} \pmod p$ : kjer $g^{-1}$ predstavlja inverz za množenje, ki ga preprosto dobimo z razširjenim Evklidovim algoritmom, odvisen je samo od generatorja grupe in ne od števila, katerega logaritem iščemo ( $\log_g 1$ si zapomnimo kot edini element baze).

### Grupa $(\mathbb{Z}_p^*, *)$ :

$G=\{1,...,p-1\}$ ; $g\circ h:=g*h \pmod p$

$|G|=p-1$ ; $e=1$ ; $g^m:=g^m \pmod p$

$|PG|=\varphi(p-1)$ : $\varphi(x)$ je Eulerjevo funkcija oziroma število naravnih števil, ki so manjša in hkrati tuja $x$ .

V tej grupi je računanje diskretnega logaritma precej težji problem kot v $(\mathbb{Z}_p, +)$ , saj trdimo, da je faktorizacija števil na zmnožek praštevilskih faktorjev računsko nedosegljiva (za splošen $p$ reda $O(\sqrt{p})$ ). Hkrati so praštevilski faktorji kar vsa praštevila, ki so manjša od $o$ . Pri $(\mathbb{Z}_p, +)$ za vsako število preprosto najdemo faktorizacijo, kjer je njen edini faktor 1.

# 3 Diffie-Hellmanov protokol za izmenjavo ključev

Zgodovinsko prvi objavljen protokol, ki uporablja kriptografijo z javnim ključem je bil objavljen leta 1976.

Javni podatki:

1. Izberemo $p$ za grupo $(\mathbb{Z}_p^*, *)$
2. Izberemo $g\in G;\text{ord}(g)=p-1$

Izmenjava ključev:

1. Uporabnik $U$ izbere naključen $a_U$, kjer $a_U\in\mathbb{Z}_{p-1}$. Nato izračuna $b_U=g^{a_U} mod\ p$ in pošlje potenco $b_U$ prejemniku $V$.
2. Uporabnik $V$ izbere naključen $a_V$, kjer $a_V\in\mathbb{Z}_{p-1}$. Nato izračuna $b_V=g^{a_V} mod\ p$ in pošlje potenco $b_V$ prejemniku $U$.
3. Uporabnik $U$ izračuna $K=b_V{}^{a_U}$, uporabnik $V$ izračuna $K=b_U{}^{a_V}$.
4. Od tu naprej pri šifriranju oba uporabljata ključ $K$.

Varnost kriptosistema temelji na tem, da če poznamo samo $g^{a_U}$ in $g^{a_V}$ je računsko nedosegljivo izraziti $g^{a_U a_V}$.

Eden izmen načinov kako izračunamo $g^{a_U a_V}$ je:

1. $a_V = \log_g(g^{a_V})$
2. $g^{a_U a_V} = (g^{a_U})^{a_V}$

Od tod sledi, da v primeru obstoja algoritma, ki učinkovito izračuna diskretni logaritem, predlagani protokol ne omogoča varnosti. Če bi tak algoritem obstajal, bi lahko napadalec, ki bi samo prisluškoval, lahko dešifriral vse prenesene podatke tako od $U$ kot $V$.

# 4 ElGamalov kriptosistem

Leta 1985 je ElGamal predstavil kriptosistem z javnimi ključi, ki temelji na računski nedosegljivosti diskretnega logaritma znotraj $(\mathbb{Z}_p^*, *)$.

Ključ:

$\kappa = \{(p, g, a, \beta); \beta = g^a\}$, kjer je g generator grupe $(\mathbb{Z}_p^*, *)$. Pri tem je trojica $(p, g, \beta)$ javni ključ, $a$ pa zasebni ključ.

Šifriranje in dešifriranje:

Za ključ $k = (p, g, a, \beta)$ in naključno število $r \in \mathbb{Z}_{p-1}$ definiramo

1. $y_1 = g^r \bmod p;\ y_2 = x\beta^r \bmod p;\ x \in \mathbb{Z}_p^*$
2. $e_k(x, r) = (y_1, y_2)$
3. $d_k(y_1, y_2) = y_2(y_1^a)^{-1}$

Varnost temelji na tem, da zasebni ključ $a$ pozna samo prejemnik. Prejemnik nekje javno objavi ostali del ključa $(p, g, \beta)$ in s tem omogoči vsem, ki mu želijo poslati sporočilo da uporabijo te informacije pri kodiranju besedila, ki mu ga želijo poslati.

Pasivni napadalec med pošiljanjem vidi samo sporočilo $(y_1, y_2)$, ki pa ga brez zasebnega ključa $a$ ne more dekodirati.

Podobno kot pri Diffie-Hellmanov protokolu, lahko tu s pomočjo diskretnega logaritma izračunamo zasebni ključ:

$a = \log_g(\beta)$.

# 5 Algoritmi za računanje diskretnega logaritma

Algoritme delimo na take, ki delujejo na splošnih grupah in tiste, ki uporabljajo posebne lastnosti nekaterih grup. Pri vseh algoritmih bomo z $g$ označili generator grupe, in z $m$ njegov red.

## *5.1 Splošni algoritmi*

Naivni algoritem za diskretni logaritem $y = \log_g x$, je kar zaporedno računanje potenc $g, g^2, g^3 \ldots$ dokler ne najdemo take, kjer velja $x = g^y$. Algoritem porabi $O(m)$ časa in $O(1)$ prostora.

Podobno lahko vse vrednosti $g, g^2, g^3 \ldots$ izračunamo v naprej in si jih uredimo v pare $(g^i, i)$, ki jih shranimo v zgoščevalno tabelo. Ko želimo izračunati diskretni logaritem najdemo polje z vrednostjo ključa $g^i$ in vrnemo vrednost $i$. Čas, ki ga potrebujemo je reda $O(1)$. Problem te metode sta prostorska zahtevnost in čas pred-procesiranja, ki sta reda $O(m)$.

### Shanksov in Pollardov Rho algoritem

Shanksov algoritem je znan tudi pod imenom metoda majhnega in velikega koraka (v angleščini ima kratico BSGS). Oba algoritma temeljita na iskanju ciklov, kjer ima Shanksov prostorsko in časovno zahtevnost reda $O(\sqrt{m})$. Pollardov Rho algoritem pa pričakovano časovno zahtevnost reda $O(\sqrt{m})$, prostorsko pa $O(1)$. Zaradi tega je največkrat uporabljen algoritem za računanje diskretnega logaritma v splošnih abelovih grupah.

### Pohlig-Hellmanov algoritem

Algoritem deluje v splošnih grupah, a se zanaša na dejstvo da se v nekaterih primerih red grupe $m$ lahko faktorizira na majhne faktorje $m = \prod_{i=1}^{k} p_i^{c_i}$. Najprej izračunamo diskretne logaritme po modulih faktorjev in jih nato sestavimo skupaj s pomočjo kitajskega izreka o ostankih.

Če faktorizacijo $m = \prod_{i=1}^{k} p_i^{c_i}$ poznamo (ali jo lahko učinkovito izračunamo), s poljubnim algoritmom za iskanje diskretnega logaritma poiščemo rešitve $y_{ik} = \log_g x \pmod{p_i^k}; k \in \{1, \ldots, c_i\}$ za vse potence vseh faktorjev. Nato v času $O(\sum c_i) < O(\log m)$ (kar lahko upoštevamo kar kot konstanto) sestavimo rešitev po modulu $m$.

Če kot metodo za reševanje podproblemov vzamemo Pollardov Rho algoritem imamo časovno zahtevnost reda $O(c_i * \sqrt{p_i})$, kjer je $p_i$ največji prafaktor $m$. Prostorska zahtevnost je v tem primeru reda $O(1)$ [2].

Napadom, ki temeljijo na tem algoritmom se izognemo, če izberemo tako grupo, kjer je red grupe oblike $m = 2 * q$, kjer je $q$ praštevilo.

## 6 Metode tipa Index calculus

Motivacija v algoritmih tipa index calculus je ta, da če poznamo diskretne logaritme nekega majhnega števila neodvisnih elementov (kasneje prafaktorjev), lahko s pomočjo njih določimo logaritme skoraj vseh elementov v grupi. Zanašamo se na lastnost nekaterih grup, kjer lahko večino njenih elementov izrazimo samo s pomočjo majhnega števila prafaktorjev.

Primera grup v prid tej trditvi sta:

$(\mathbb{Z}_p, +)$ : Edini prafaktor je 1. Faktorizacija je časovnega in prostorskega reda $O(1)$.

$(\mathbb{Z}_p^*, *)$ : Prafaktorji so vsa praštevila, ki so manjša od p, kar je skoraj reda $O(p)$, a hkrati velja, da je polovico elementov v grupi možno faktorizirati na prafaktorje manjše od $\sqrt{p}$.

Primer grup kjer osnovna trditev ne velja:

Eliptične krivulje: Učinkovite algoritme za faktorizacijo točk na eliptični krivulji so našli za zelo malo posebnih poddružin. Zato je tu metoda index calculus v splošnem neuporabna.

V metodi bomo s pomočjo linearne kombinacije elementov iz faktorske baze, ki vsebuje samo zgoraj omenjeno majhno podmnožico prafaktorjev, izrazili diskretne logaritme kateregakoli elementa v grupi.

Vsaka metoda tipa index calculus se deli na 3 podprobleme:

1. Generiranje gladkih relacij, ki uporabljajo samo faktorje iz faktorske baze.
2. Reševanje zgoraj dobljenih linearnih enačb in izračun diskretnih logaritmov za vse elemente v faktorski bazi.
3. Izračun diskretnega logaritma poljubnega števila v grupi s pomočjo logaritmov elementov v faktorski bazi.

Pomembno je izpostaviti, da sta prvi in drugi podproblem popolnoma neodvisna od števila, katerega logaritem iščemo, in ju lahko zato izračunamo čim poznamo grupo. Hkrati velja, da sta fazi neodvisni tudi od generatorja $g'$, saj lahko dobljen logaritem za bazo $g$ učinkovito prenesemo v bazo $g'$.

$$\log_{g'} a = \log_g a * \log_g (g')^{-1}$$

## 6.1 Generiranje baznih enačb

Računamo diskretni logaritem znotraj grupe $(G, \circ)$ z močjo $m$, generatorjem $g$ in faktorsko bazo $B = \{p_1, p_2, \ldots, p_{|B|}\}$.

Za naključne potence $x_j$ izračunamo naslednje relacije (uporabimo le tiste, ki imajo vse faktorje $p_i$ znotraj baze):

$$g^{x_j} = {p_1}^{d_{1j}} \circ {p_2}^{d_{2j}} \circ \ldots \circ {p_B}^{d_{Bj}}$$

Obe strani enačbe logaritmiramo:

$$\log_g (g^{x_j}) \equiv \log_g ({p_1}^{d_{1j}} \circ {p_2}^{d_{2j}} \circ \ldots \circ {p_B}^{d_{Bj}}) \pmod m$$

Uporabimo lastnosti logaritma, da dobimo:

$$x_j \equiv \sum_{i \in (1..B)} d_{ij} \log_g p_i \pmod m$$

## 6.2 Reševanje linearnih enačb

Po izreku iz linearne algebre vemo, da za rešitev sistema z $n$ neznankami potrebujemo $n$ neodvisnih enačb. Torej, če v prvi fazi generiramo $n$ neodvisnih enačb lahko s pomočjo Gaussove eliminacije izračunamo vrednosti vseh logaritmov elementov iz baze.

Eliminacija znotraj $\mathbb{Z}_m$ je enaka tisti znotraj $\mathbb{Z}$. Paziti moramo le na to, da $\mathbb{Z}_m$ nima operacije deljenja, in zato množimo z inverzom za množenje. Hkrati imajo inverz za množenje le tista števila $t$, ki so tuja $m$.

V primeru, ko želimo deliti in nimamo inveza za $t$, imamo 3 možnosti:

1. Uporabimo Pohlig-Hellmanov algoritem in za reševanje podproblemov izkoristimo kar metodo index calculus.
2. Najdemo ustrezno linearno kombinacijo enačb, v kateri je izračunan faktor $t'$ tuj $m$. Upoštevamo lastnost:

$m=\prod_{i=1}^{k} p_i^{c_i}$ $(p_i|u)\wedge(p_i\nmid v)\rightarrow \gcd(u,v)\leq \min(u,v)$

Torej lahko poljuben par vrstic, ki zadošča zgornjemu pogoju zreduciramo do $t'<min(u,v)$. Redukcijo izvajamo s pomočjo razširjenega Evklidovega algoritma in tudi nad več pari zaporedoma, tako da z veliko verjetnostjo dobimo $t'$, ki je tuj $m$.
Če taka linearna kombinacija enačb ne obstaja, hkrati velja, da ne moremo rešiti vsaj enega podproblema v Pohlig-Hellmanovem algoritmu (vseh tistih kjer velja $\gcd(t', p_i)\neq 1$).

3. Delimo s faktorjem $t=\gcd(t_i, m)$ in nato upoštevamo, da od tu naprej vse računamo po modulu $m/t$, kjer moramo upoštevati, da ob prenosu nazaj na osnovo $m$ dobimo $t_i$ enakovrednih rešitev, izmed katerih moramo najti pravilo.

### 6.3 Izračun diskretnega logaritma

Iščemo $y=\log_g a$ znotraj grupe $(G,\circ)$.

Po analogiji z faze 1 generiramo enačbe, dokler ne najdemo take, ki jo faktoriziramo znotraj baze:

$a\circ g^x = {p_1}^{d_1}\circ {p_2}^{d_2}\circ ...\circ {p_B}^{d_B}$

Obe strani enačbe logaritmiramo:

$\log_g(a\circ g^x)\equiv \log_g({p_1}^{d_1}\circ {p_2}^{d_2}\circ ...\circ {p_B}^{d_B}) \pmod m$

Uporabimo lastnosti logaritma in s pomočjo logaritmov baznih elementov izračunamo rešitev:

$\log_g(a)\equiv \sum_{i\in(1..B)} d_i \log_g p_i - x \pmod m$

# 7 Index calculus v grupi $(\mathbb{Z}_p^*, *)$

Za učinkovito implementacijo metode index calculus znotraj $(\mathbb{Z}_p^*, *)$ potrebujemo učinkovito metodo za generiranje enačb in faktoriziranje števil s pomočjo faktorske baze.

### 7.1 Faktorizacija naravni števil na produkt prafaktorjev

Poznanih algoritmov za praštevilsko faktorizacijo je veliko. Nekaj izmed njih je zbrano v viru [4].

Najpreprostejši in pri majhnih bazah najhitrejši je kar algoritem poskusnega deljenja. Pri njem delimo število $x$ z vsemi elementi $q\in\{(p\in B)\wedge(p|x)\}$ na poljubno potenco.

Na koncu imamo 2 primera:

1. $x=1$ : faktorizacija je množica parov $F=\{(q,c); q^c|x\}$
2. $x\neq 1$ : števila $x$ se na da faktorizirati znotraj faktorske baze

Njegova časovna zahtevnost je velikostnega reda $O(|B|)$.

Za večje baze se uporabi metode po principu številskega sita, ki imajo pričakovan čas izvajanja reda $O(e^{(1.92+O(1))(\ln(n))^{1/3}(\ln(\ln(n)))^{2/3})})$, kjer je $n$ število, ki ga želimo faktorizirati [1, str. 200].

## 7.2 Vplivi na izbiro baze

Baza mora biti majhna, saj moramo biti sposobni rešiti sistem enačb v 2. fazi metode index calculus, ki ima časovno zahtevnost reda $O(|B|^3)$. Manjša baza nam tudi zmanjša čas faktorizacije in porabo pomnilnika.

Izbrati moramo zadosti veliko, da s čim večjo verjetnostjo zgeneriramo števila, ki so gladka znotraj naše baze.

## 7.3 Implementacija

Pri vseh funkcijah bomo upoštevali, da so vsi parametri manjši od $p$, ki ima $N$ bitov.

Iz definicije problema sledi, da moramo implementirati sledeče metode:

- pri generiranju enačb potrebujemo učinkovite metode za množenje, potenciranje in faktorizacijo znotraj grupe $(\mathbb{Z}_p^*, *)$
- v drugi fazi vse operacije potekajo znotraj $(\mathbb{Z}_m, +)$ in so enake ne glede na izbiro grupe v kateri računamo diskretne logaritme
  - potrebujemo seštevanje, odštevanje, množenje, inverz za množenje, razširjen Evklidov algoritem, iskanje največjega skupnega delitelja...
  - operacije nad vrsticami v matriki, ki jih potrebujemo pri Gaussovi eliminaciji

### Množenje v $(\mathbb{Z}_p^*, *)$

Definicija: $*(a, b) = a * b \bmod p$

Algoritem:

- VHOD: $a, b \in 2^N$
- IZHOD: $c \in 2^N$

1. $c = 0$
2. **while** $a > 0$
   $c = (c + (a \textbf{ mod } 2) * b) \textbf{ mod } p$
   $a = a \div 2;\ b = (2 * b) \textbf{ mod } p$
3. **return** $c$

Predlagani algoritem pazi, da noben vmesni rezultat ne preseže vrednosti $2 * p$, in uporablja idejo z Montgomeryjevega množenja [3]. Po $O$ notaciji, je enako hiter, saj na trenutnih arhitekturah bitna rotacija in odštevanje porabita enako časa.

Poleg tega upoštevamo da redukcijo po modulu v tem primeru definiramo kot $(x < p)\,?\,x : x - p$, ki porabi čas preverjanja in odštevanja.

Množenje z $a \bmod 2$ pa se prevede v operacijo $a_0 * b = (a_0 = 1)\,?\,b : 0$, ki porabi konstanten čas.

### Potenciranje v $(\mathbb{Z}_p^*, *)$

Za potenciranje uporabimo metodo kvadriraj in zmnoži. Hkrati se izkaže , da pri generiranju enačb lahko zaporedne kandidate dobimo samo z množenjem in zaradi tega potenciranja ne potrebujemo.

Operacije prevedemo na množenje, ker ne more obstajati definicija, kjer bi bilo potenciranje po $O$ notaciji hitrejše kot množenje. Velja enaka analogija kot pri hitrosti množenja in kvadriranja.

### Faktorizacija elementov iz $(\mathbb{Z}_p^*, *)$ znotraj baze

Pravzaprav iščemo kar faktorizacijo znotraj $\mathbb{N}$, kjer so vsa števila, ki jih bomo morali faktorizirati manjša od $p$. Tu uporabimo poljuben algoritem za faktorizacijo naravnih števil. Ker je baza majhna, se dobro odnese že algoritem poskusnega deljenja.

### Operacije v $(\mathbb{Z}_m, +)$

Za odštevanjem in seštevanjem izvedemo še modularno redukcijo. Pri množenju uporabimo metodo z $(\mathbb{Z}_p^*, *)$. Vsi ostali algoritmi temeljijo na razširjenem Evklidovem algoritmu, ki je lahko implementiran, kar po definiciji z srednješolskih knjig za matematiko in že ima časovni red primerljiv trenutno najboljšim poznanim algoritmom.

### Reševanje linearnega sistema enačb s koeficienti znotraj $(\mathbb{Z}_m, +)$

Linearni sistem enačb, ki ga rešujemo pri drugi fazi metode index calculus ima zelo malo neničelnih elementov, zato za shranjevanje enačb raje uporabimo kar seznam, v katerem si zapomnimo le neničelne elemente. Zaradi takšne predstavitve se trudimo, da vse vrednosti koeficientov čim prej izračunamo in s tem čim bolj zmanjšamo porabo pomnilnika. Z enakega razloga pri eliminaciji vrstic, kot pivotno vrstico vedno izberem tako vrstico, ki ima najmanj neničelnih elementov.

Kot je že v poglavju [6.2] omenjeno, je edina stvar, ki se razlikuje od klasičnega reševanja sistema linearnih enačb ta, da se nam lahko zgodi, da nimamo elementa, ki bi ga pri metodi Gaussove eliminacije lahko uporabili kot pivot. V tem primeru uporabimo eno izmed treh naštetih metod [6.2].

## *7.4 Programski jezik in podrobnosti moje implementacije*

Za implementacijo sem uporabil programski jezik C# (2.0), ker se pred izvajanjem prevede in se zato izvaja hitro, a hkrati ima podporo za samodejno upravljanje s pomnilnikom.

Hkrati omogoča tudi preoblaganje operatorjev, zaradi česar sem lahko elemente znotraj $(\mathbb{Z}_p^*, *)$ in $(\mathbb{Z}_m, +)$, opisal kar kot razreda, med katerimi lahko operacije izvajamo z operatorji (+, -, *, ...) in je zato njihova uporaba v kodi precej bolj pregledna

Za omejitev moje knjižnice na grupe za množenje, kjer je $p < 2^{63}$, sem se odločil, ker ima prevajalnik ukaze samo do števil takega reda. Novejše različice programskega jezika C# (4.0 -) imajo podprt tudi tip BigInteger, ki omogoča operacije nad poljubno velikimi celimi števili.

# 8 Rezultati

Za dokaz pravilnosti knjižnice zadošča njen izračun diskretnega logaritma dveh velikih zaporednih števil.

Testna grupa je bila $(\mathbb{Z}_p^*, *)$; $p = 4611686018427394499$; $g = 7213$, kjer velja $p = 2*q+1$ in je $q$ praštevilo. Število $g$ je poljuben generator grupe.

Izračunali smo diskretna logaritma števil

$\log_g(1582712748901606033) = 168601842739449$ in

$\log_g(1582712748901606034) = 599969798411339363$

## 8.1 Čas izračuna pri različnih velikostih baze

Tabela prikazuje izmerjen čas izvajanja določene faze v primeru različnih velikosti baz.

| #baznih elementov | Čas generiranja enačb [s] | Čas reševanja sistema enačb [s] | Čas iskanja logaritma [s] | Skupen čas [s] |
|---|---|---|---|---|
| 8000 | 1060 | 160 | 0 | 1260 |
| 2000 | 300 | 30 | 0 | 330 |
| 1000 | 225 | 20 | 0 | 245 |
| 500 | 200 | 5 | 0 | 205 |
| 400 | 210 | 2 | 0 | 212 |

Meritve so bile izvedene na računalniku z *Intel Core 2 Duo* procesorjem s taktom 2.4 GHz. Čeprav je procesor 64 biten, prevedena koda uporablja 32 bitne strojne ukaze, ker imam 32 bitni operacijski sistem.

## 8.2 Časovna in prostorska analiza moje implementacije

Za število operacij štejem število množenj krat število poskusnih deljenj pri algoritmu poskusnega deljenja. To poenostavitev sem uporabil, ker imata pri splošnih parametrih obe operaciji asimptotično enak čas izvajanja.

V tabeli so prikazani izmerjeni rezultati za prostorsko in časovno zahtevnost algoritmov, kjer pri metodi index calculus upoštevam samo tretjo fazo, pri optimalno izbrani bazi.

| | Index calculus | Shanks | Pollard Rho |
|---|---|---|---|
| Velikost baze | $5404 = O(\sqrt[5]{p})$ | $O(\sqrt{p})$ | $O(1)$ |
| #množenj | $O(\sqrt[8.48]{p})$ | $O(\sqrt{p})$ | $O(\sqrt{p})$ |
| #operacij | $O(\sqrt[3.15]{p})$ | $O(\sqrt{p})$ | $O(\sqrt{p})$ |

Pri metodi index calculus lahko obravnavamo tudi čas vseh treh faz skupaj, kjer je $E(x)$ pričakovano število generiranih enačb, preden najdemo tako z nam ugodno faktorizacijo. V našem primeru smo $E(x)$ izračunali kot kvocient med številom vseh generiranih enačb in številom enačb za katere smo našli faktorizacijo znotraj faktorske baze.

1. faza $T_1 = O(|B|^2 * E(x))$
2. faza $T_2 = O(|B|^3)$
3. faza $T_3 = O(|B| * E(x))$

Skupen čas metode je $T = T_1 + T_2 + T_3$.

| #baznih elementov | $E(x)$ | Skupen čas T |
|---:|---:|---:|
| $5404 = \sqrt[5]{p}$ | 158,73 | $\sqrt[1.66]{p}$ |
| $2010 = \sqrt[5.65]{p}$ | 481,74 | $\sqrt[1.86]{p}$ |
| $1290 = \sqrt[6]{p}$ | 893,98 | $\sqrt[1.95]{p}$ |
| $463 = \sqrt[7]{p}$ | 5122,95 | $\sqrt[2.06]{p}$ |
| $215 = \sqrt[8]{p}$ | 31118,30 | $\sqrt[2.04]{p}$ |

Rezultati meritev se ujemajo z izmerjenimi časi izvajanja pri različnih velikostih baze.

### *8.3 Primerjava z ostalimi poznanimi algoritmi*

Trenutno največji praštevilski obseg v katerem so izračunali diskretni logaritem z algoritmom tipa index calculus ima za osnovo praštevilo z 560 biti [6]. Taka števila so povsem nedosegljiva moji implementaciji iz naslednjih razlogov:

1. Enačbe generiram povsem naključno in imam pri majhnih bazah zaradi tega majhno verjetnost, da se bo enačbo dalo faktorizirati znotraj faktorske baze. Obstajajo algoritmi, ki generirajo enačbe, ki jih lahko faktoriziramo z večjo verjetnostjo kot naključne enačbe.
2. Faktorizacijo izvajam s poskusnim deljenjem, ki je asimptotično počasnejše od metod temelječih na številskem situ.
3. Sistem enačb rešujem z Gaussovo eliminacijo, ki skoraj ne upošteva, da je večino faktorjev v sistemu enakih 0.

# 9 Zaključek in ideje za nadaljnje delo

Trenutna implementacija, če upoštevamo vse tri faze skupaj, deluje le za odtenek hitreje kot Pollardova Rho metoda, a hkrati pokaže, da so lahko metode tipa index calculus uporabne že pri tako majhnih številih.

Za iskanje diskretnih logaritmov večjih praštevil, pa nujno potrebujemo knjižnico za operacije nad velikimi števili, ki so napisane v zbirniku, katerih pa C# vse do različice 4.0 na žalost ni imel. Hkrati tudi ne moremo napisati odsekov kode v zbirnem jeziku (lahko vključimo knjižnice v zbirniku, a klic zunanje knjižnice porabi več časa, kot izvajanje danega odseka).

V viru [6] so v bazi imeli logaritme vseh praštevil, ki so manjša od $2^{32}$. S tem so potrdili, da se lahko omejimo na definicijo funkcije deljenja, kjer je delitelj manjši od $2^{32}$, ki pa na 64 bitni arhitekturi deluje s časovnim redom $O(\log x)$. Splošne funkcije deljenja (tudi ta, ki jo trenutno uporabljam) pa imajo časovno zahtevnost $O(\log^2 x)$.

Kot opombo lahko omenim, da za izračun vseh praštevil do $2^{32}$, porabimo manj kot 12 ur, že s funkcijo, ki je implementirana v naši knjižnici.

Ob uporabi zbirniških ukazov na Intelovi 64 bitni arhitekturi [7] lahko napišemo tudi metode za množenje z redukcijo med dvema faktorjema s 64 biti, ki traja konstanten čas (nekaj urinih period).

Uporabimo strojna ukaza MUL in DIV in upoštevamo naslednje dve lastnosti, ki sta nujno potrebni:

1. $a < p \wedge b < p \rightarrow a * b < p^2$
2. $\frac{a*b}{p} < p$

Trenutno implementacijo, bi z danima idejama na 64 bitni arhitekturi lahko hitro priredili do te mere, da bi bila zmožna izračunati tudi logaritme do reda $2^{128}$, kar pa se trenutno tudi zdi zgornja meja ob uporabi faktorizacije z metodo poskusnega deljenja. Za kakršno koli opazno optimizacijo, bi morali uporabiti generiranje enačb in faktorizacijo z funkcijami, ki temeljijo na številskem situ.

## Viri in literatura


1. D. Stinson, *Cryptography: Theory and Practice,* CRC Press, 2006
2. Jason S. Howell, *The index calculus Algorithm for Discrete Logarithms*, magistrsko delo, Clemson University, 1998
3. A. Menezes, P. Van Oorschot, in S. Vanstone, *Handbook of Applied Cryptography*, CRC Press, 1996
4. P. Bogataj, *Faktorizacija naravnih števil*, diploma, Univerza v Ljubljani, 2011
5. P. Nose, *Učinkovita aritmetika na eliptičnih krivuljah nad praštevilskimi obsegi,* diploma, Univerza v Ljubljani, 2008
6. T.Kleinjung, *Discrete logarithms in GF(p) --- 160 digits*, University of Bonn, 2007
7. Intel Corporation*, Intel® 64 and IA-32 Architectures Software Developer's Manual,* 2012