»Hitro« odkrivanje potenc praštevil

Seminarska naloga pri predmetu Kriptografija in računalniška varnost

Mentor : Aleksandar Jurišić
Avtor : Hubert Benedik

FRI,
Ljubljana,24.5.2000

1.UVOD

Seminarska naloga ni neposredno s področja kriptografije ali računalniške varnosti, temveč spada bolj na področje (algoritmične) teorije števil oz. ožje transcendentne teorije števil, seveda pa obstaja možnost njene uporabe tudi na omenjenem področju. Lep primer uporabe hitrega odkrivanja potenc praštevil bi bila na primer kriptoanaliza z znanim tajnopisom, kjer vemo, da je bil za enkripcijo uporabljen nek »potenčni« algoritem (npr.RSA). Z uporabo algoritma za razbijanje popolnih potenc nam namreč lahko v tem primeru uspe poiskati čistopis in s pomočjo tega nadalje tudi razbiti kriptosistem (t.j. poiskati vrednost privatnega ključa).
V uvodu bom najprej opisal posamezne dele seminarske naloge, takoj zatem pa razjasnil oz. definiral nekaj osnovnih pojmov iz transcendentne teorije števil, ki so bistveni za razumevanje vsebine naloge, kot so :

- transcendentna števila
- popolne potence
- algoritmi za odkrivanje, razbijanje (dekompozicijo) in razvrščanje (klasifikacijo) popolnih potenc (iskanje takšnega algoritma je problem, ki sem si ga zastavil in tudi tema te seminarske naloge)

V jedru naloge bom predstavil Loxtonov izrek o linearnih formah in Bernsteinovo aplikacijo tega izreka :

- za postavitev meje števila popolnih potenc v nekem kratkem intervalu (Loxton)
- za analizo funkcije *F(n)* - funkcijo, ki opredeljuje časovne zahtevnosti algoritma za odkrivanje popolnih potenc

temu pa bo sledil opis reševanja problema, s katerim sem se ukvarjal :

- poizkus uporabe Loxtonovega izreka o linearnih formah
- naiven pristop k reševanju problema (logaritmiranje in razvoj v Taylorjevo vrsto - prednosti/pomanjkljivosti, poenostavitev problema s približkom ovrednotenja logaritmov)
- Bernsteinova aplikacija Loxtonovega izreka in časovna zahtevnost tega algoritma (osnutek algoritma za rešitev problema)

Torej najprej nekaj besed o transcendentnih številih. Beseda transcendenten v našem besednjaku ni ravno pogosta, pomeni pa nekaj nenavadnega, abstraktnega, lahko tudi metafizičnega, nadnaravnega, kar nekako že daje slutiti, da gre za neka posebna števila. Matematična analiza pa realna števila, kot vemo deli na racionalna - to so vsa cela in vsa ulomljena števila in iracionalna števila - slednja lahko izrazimo z neperiodičnim neskončnim decimalnim ulomkom. Če si iracionalna števila ogledamo podrobneje, spadajo mednje ne celi realni koreni polinomov s celimi koeficienti; taka števila imenujemo algebraične

iracionalnosti (preprost primer algebraičnih iracionalnosti so koreni binomskih enačb $x^n-a=0$ ali števila oblike $\sqrt[n]{a}$, če niso racionalna); iracionalna števila, ki niso algebraične iracionalnosti , imenujemo **transcendentna števila**; k njim spadajo π, e, desetiški logaritmi celih števil (razen oblike 10^n), večina vrednosti trigonometričnih funkcij kota, ki je enak celemu številu stopinj...
Neko naravno število $n>1$ je **popolna potenca**, če obstajajo taka naravna števila x in $k>1$, za katera velja $n = x^k$. Pri tem hitro lahko vidimo, da velja $k \leq \log_2 n$ in da je najmanjši k praštevilo.
V zvezi s popolnimi potencami ločimo nekako tri vrste algoritmov. Ti so : algoritmi za odkrivanje, razbijanje (dekompozicijo) in razvrščanje (klasifikacijo) popolnih potenc.

Algoritem za odkrivanje popolnih potenc je algoritem, ki za podano naravno število $n>1$ ugotovi, ali n je popolna potenca ali ne. **Algoritem za razbijanje popolnih potenc** mora storiti nekaj več: če je n popolna potenca, poišče x in $k>1$, tako da velja $n = x^k$. **Algoritem za klasifikacijo popolnih potenc** pa stori vse, kar se od teh algoritmov pričakuje : t.j. zapiše n v obliki x^k, kjer je k največji.

Motivacija za implementacijo tovrstnih algoritmov pa je v tem, da preden poizkušamo razbiti število na prafaktorje, bi se morali prepričati da n ni popolna potenca ali vsaj potenca praštevila, četudi metoda za preverjanje potence ni najhitrejša.

2. TRANSCENDENTNE METODE

Števila $x_0,\ldots,x_n$ so **multiplikativno odvisna**, če obstajajo naravna števila $a_0,\ldots,a_n$, ki niso vsa enaka nič, tako da velja $x_0^{a_0} \cdots x_n^{a_n} = 1$.

Pa si sedaj oglejmo Loxtonov izrek o linearnih formah. Zato najprej definirajmo še **višino** neničelnega racionalnega števila α, kot $H(\alpha)$=max $\{|i|, |j|\}$, pri čemer privzamemo da je $\alpha=i/j$ že okrajšan ulomek. Višina 0 je 0.
Loxtonov izrek je posplošitev Bakerjevega izreka, ki trdi da neničelna linearna forma logaritmov ne more biti poljubno blizu 0 oz. kateremukoli algebrajskemu številu, na več neodvisnih linearnih formah v isti množici logaritmov. Če njegov izrek zapišemo za d=1, dobimo naslednjo pomožno trditev.

Trditev: *Izberi nek* $c \geq 1, n \geq 1$. *Naj bodo* $\alpha_1,\ldots,\ \alpha_n$ *multiplikativno neodvisna pozitivna racionalna števila. Naj bo* $\begin{bmatrix} \beta_{11} & \beta_{12} & \cdots & \beta_{1n} \\ \beta_{21} & \beta_{22} & \cdots & \beta_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ \beta_{c1} & \beta_{c2} & \cdots & \beta_{cn} \end{bmatrix}$ *matrika racionalnih števil ranga C.*

Določi $A_j \geq 4$ in $B \geq 4$ tako, da je $H(\alpha_j) \leq A_j$ in $H(\beta_{ij}) \leq B$. Zapiši $\Omega = (\log A_1) \ldots (\log A_n)$ ter

$$\begin{bmatrix} \Lambda_1 \\ \Lambda_2 \\ \vdots \\ \Lambda_c \end{bmatrix} = \begin{bmatrix} \beta_{11} & \beta_{12} & \cdots & \beta_{1n} \\ \beta_{21} & \beta_{22} & \cdots & \beta_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ \beta_{c1} & \beta_{c2} & \cdots & \beta_{cn} \end{bmatrix} \cdot \begin{bmatrix} \log \alpha_1 \\ \log \alpha_2 \\ \vdots \\ \log \alpha_n \end{bmatrix}.$$

Potem obstaja nek i, tako da je $|\Lambda_i| > \exp(-(16n)^{200n} (\Omega \log \Omega)^{1/C} \log B\Omega)$*; torej neka kombinacija neodvisnih (neničelnih) linearnih form v isti množici logaritmov ne more biti poljubno blizu 0.*

Bernstein je v svojem delu uporabil Loxtonov izrek za dosego naslednjih dveh rezultatov:

- postavil je mejo za število popolnih potenc v nekem kratkem intervalu,
- napravil je analizo funkcije *F(n)* - funkcije, ki opredeljuje časovno zahtevnost algoritma za odkrivanje popolnih potenc.

Pa si poglejmo podrobneje vsako od aplikacij posebej, saj sta oba rezultata pomembna za reševanje našega problema.

2.1 POTENCE V KRATKIH INTERVALIH

Motivacija za posebno obravnavo potenc v kratkih intervalih je v tem, da se na podlagi moči množice popolnih potenc znamo odločiti, kdaj iskati potence ročno in kdaj algoritemsko. Bernstein trdi, da nek kratek interval $[a,b]$ ne more vsebovati mnogo popolnih potenc. Pri tem v resnici šteje število takih eksponentov k, da je neka k-ta potenca ravno na intervalu $[a,b]$, trditev pa nasloni na dve pomožni trditvi, od katerih prva pravi, da lahko obstaja le malo »velikih« eksponentov k, druga pa podaja zgornjo mejo za število eksponentov k, kjer je k praštevilo.

Obe pomožni trditvi navajam tu brez dokaza:

Lema 1: *Izberimo nek interval $[a,b]$, kjer velja $1 < b/e < a < b$. Določi celo število $C \geq 1$ in $K \geq 4$ tako, da velja* $K \geq (16C)^{200C} \frac{\log b}{-\log\log(b/a)} (\log b)^{1/C} ((C+1)\log\log b)^2$ *in*

$$K \geq (16C)^{200C} \frac{\log b}{-\log\log(b/a)} (\log 6C^C + (2C+1)\log\log b)^2.$$

Naj bo S množica vseh celoštevilčnih parov (x,k), kjer je $x^k \in [a,b]$, $x \geq 4$, $k \geq K$. Potem velja: $|S| \leq C + \log_2 C! + C \log_2 \log_2 b$. ▫

Lema 2: *Izberi $n \geq \exp\exp 1000$. Postavi $u = \log\log 2n$. Določi v tako, da bo $1 \leq v \leq \log_5 n$. Naj bo S množica takih praštevil k, da je k-ta potenca nekega števila na intervalu $[n - n^{1-1/v}, n + n^{1-1/v}]$. Potem je* $|S| < 3vu^3 \exp(30\sqrt{u \log 2.56u})$. ▫

Dokaza obeh pomožnih trditev sta precej obsežna in ju lahko najlažje dokažemo z uporabo Loxtonovega izreka o linearnih formah logaritmov, zato ju opuščam. Namesto njiju se raje seznanimo z algoritmom za odkrivanje popolnih potenc in njegovo časovno zahtevnostjo.

2.2 ANALIZA FUNKCIJE *F(N)*

Zapisal bom algoritem X za preverjanje, če je neko število popolna potenca in ga opisal. Za funkcijo $F(n)$, ki bo postavljala zgornjo mejo časovne zahtevnosti tega algoritma. Taka ocena nam bo namreč povedala, kako potraten je algoritem X glede na velikost problema.
Poglejmo si najprej, kako preveriti za podani n, če je popolna potenca. To storimo tako, da preverimo za vsa praštevila $p \le \log_2 n$, če je n p-ta potenca.

Algoritem *X*: *Za podani $n \ge 2$ in zapis n-ja kot popolne potence, če je to mogoče: Najprej postavi $f = \lfloor \log_2 2n \rfloor$*
1. *Izračunaj $y \leftarrow exp(\ln(n)/(3+\lceil f/2 \rceil))$*
2. *Za vsako praštevilo $p < f$:*
3. *Uporabi algoritem K za (n,p,y); naj bo x rezultat*
4. *Če je $x > 0$ izpiši (x,p) in končaj*
5. *Izpiši (n,1)*

Pri tem smo za preverjanje, ali je n p-ta potenca uporabili algoritem K, y pa nam služi kot vnaprej izračunani približek za n^{-1}. Ideja delovanja algoritma K je v tem, da izračuna realni približek r za $n^{1/k}$, recimo $|n^{1/k} - r| < 1/4$. Potem pa, če je r znotraj 1/4 okoli celega števila x, preveri s pomočjo algoritma C, če je $x^k = n$. Algoritem C bi to lahko napravil enostavno tako, da bi potenciral x, vendar ker bi bilo to prezamudno, to napravi tako, da izračuna le prvih nekaj števil potence in takoj ko pride do različnega števila vrne predznak razlike $x^k - n$. Algoritem K nam izpiše $n^{1/k}$, če n je k-ta potenca in 0, če to ni.

Trditev: *Algoritem X deluje pravilno, t.j. zapiše n kot popolno potenco, če je to mogoče.*
Dokaz: Prepričajmo se sedaj v pravilnost delovanja algoritma X tako, da postavimo invarianto algoritma $y(1-2^{-3-\lceil f/p \rceil}) < n < y(1+2^{-3-\lceil f/p \rceil})$. Prvi korak algoritma invarianti zadošča, algoritem K v 3. koraku nam jo ohranja, saj ne spreminja parametrov y,f,p,n in končno, če se algoritem X v 4. koraku ustavi, potem je $x^p = n$. ▫

Čas, ki ga potrebuje za klasifikacijo popolnih potenc algoritem v [2] je $\log^3 n$; v povprečju, pri razumnih predpostavkah pa potrebuje čas $\log^2 n / \log^2 \log n$. Izvajalni čas algoritma X je precej boljši, saj je v osnovi linearen v $\log n$ z uporabo hitrega množenja. Dokaz uporablja transcendentno teorijo števil in ga bom podal v nadaljevanju, sedaj pa še nekaj o algoritmu X samem. Algoritem ni nov in je bil že zapisan v npr. [3, poglavje 2.4], toda bistvo algoritma je v podrobnostih: brez dobre metode za izračun $n^{1/k}$ (t.j. $exp(\ln(n)/(3+\lceil f/2 \rceil))$) in za preverjanje ali je $x^k = n$, algoritem X sploh ni zanimiv. Avtorji [3] pravijo, da se da prihraniti čas z dodajanjem vrste testov k algoritmu X. Nekatere variante tega algoritma so tudi opustili [4, stran 38] (»*To je očitno popolnoma neučinkovito*«) in [2].
Za označevanje praštevil $p < f$ je Bernstein uporabil Eratostenovo sito, najboljši vrstni red preverjanja v algoritmu X pa je po njegovem mnenju odvisen od porazdelitve vhoda; npr. če bo vhodni vir zelo verjetno ustvarjal 37-te potence, potem naj se izvaja najprej s $p=37$.
S tem sem zaključil opis algoritma X in se lahko osredotočim na časovno zahtevnost algoritma za klasifikacijo popolnih potenc. Definirajmo sedaj funkcijo

$$F(n) = \sum_{2 \le p \le \log_2 n} (\log_2 p) \cdot \max\{1, \log_2 n - d(n, (round(n^{1/p})^p))\}$$, za $n \ge 2$, kjer je p praštevilo

in round(t) pomeni celo število znotraj intervala 1/2 okrog t.

S seštevanjem časovnih zahtevnosti algoritmov K in C lahko izračunamo časovno zahtevnost algoritma X, ki jo zapišimo v obliki naslednje pomožne trditve.

Lema 3: *Postavi* $f=\lfloor \log_2 2n \rfloor$
Za $n\geq 2$ *algoritem* X *porabi največ* $(8F(n)+6f\lfloor \log_2 16f \rfloor^3)\,O(2f+\lfloor \log_2 128f \rfloor)$ *M-časa.* ▫

Preveriti se da, da algoritem ne porabi dosti ne-M-časa, zato algoritem X zahteva v bistvu linearen čas v $F(n)+\log n$, ob zagotovilu, da uporablja algoritem za hitro množenje. Ker pa je $F(n)$ v bistvu linearen v $\log n$, je celotna časovna zahtevnost algoritma X v bistvu linearna v $\log n$. Da je to res, nam pove naslednja pomožna trditev, ki postavlja zgornjo mejo za funkcijo $F(n)$.

Lema4: *Določi* $n\geq exp\ exp\ 1000$. *Postavi* $u=log\ log\ 2n$.
Potem je $F(n) < (\lg n \cdot \lg\lg n)(1+3u^3 \cdot \exp(30\sqrt{u \log 2.56u})\lg(4\lg n))$. ▫

Iz časovne zahtevnosti algoritma X pa zlahka izračunamo časovno zahtevnost algoritma za klasifikacijo popolnih potenc (PPC). Pa si oglejmo PPC algoritem, ki je rekurziven algoritem in nato poiščimo še njegovo časovno zahtevnost z ozirom na zahtevnost algoritma X.

PPC algoritem:Podan imamo $n\geq 2$, izpiše (x,k), če velja (1) $x^k=n$ in (2) x ni popolna potenca :
1. Uporabi algoritem X nad n; naj bo (x, p) rezultat
2. Če je $(x,p)=(n,1)$, izpiši $(n,1)$ in končaj.
3. $(2\leq x<n)$ Uporabi algoritem PPC nad x; naj bo (c,k) rezultat.
4. Izpiši (c,kp).

Če označimo zgornjo mejo časa, ki ga porabi algoritem X za $n\geq 2$ s $T(n)$, lahko kot že rečeno vzamemo za $T(n)\in \log_2 n$, ob uporabi hitrega množenja. Definirajmo $U(n)=\max\{T(m)/\log_2 m : m\leq n\}\log_2 n$ za $n\geq 2$. Sedaj je $T(m)/\log_2 m \in (\log_2 m)^{o(1)}$ za $m>2$, tako da je $U(n)/\log_2 n \in (\log_2 n)^{o(1)}$ za $n>3$. Odtod sledi $U(n) \in (\log_2 n)^{o(1)+1}$. Sedaj moramo samo še dokazati, da je časovna zahtevnost PPC algoritma kvečjemu $2U(n)$ in smo s tem pokazali, da obstaja algoritem za klasifikacijo popolnih potenc, ki potrebuje kvečjemu čas $(\log_2 n)^{1+o(1)}$.

Korak 1 zahteva kvečjemu čas $T(n)$. Če n je popolna potenca, potem se PPC algoritem kliče rekurzivno; po indukciji to zahteva kvečjemu $2U(x)$ časa. Celotna časovna zahtevnost algoritma je torej kvečjemu $T(n)+2U(x) \leq U(n)+pU(x) \leq 2U(n)$. Pri tem smo uporabili dejstvo, da je $pU(x) \leq U(n)$ (če je $n=x^p$), kar sledi iz dejstva da je $U(n)/\log_2 n$ nepadajoča funkcija n-ja, tako da je $pU(x) / \log_2 x \leq pU(n) / \log_2 n = pU(n) / p\log_2 x = U(n) / \log_2 x$.

3. PROBLEM IN NJEGOVO REŠEVANJE

V tem razdelku bom skušal podati opis reševanja naslednjega problema:

»Za dana naravna števila a,b,c,i,j,k v polinomskem času odloči, če je $a^i b^j < c^k$.«

Ali se za rešitev tega problema lahko uporabi Bernsteinovo aplikacijo Loxtonovega izreka o linearnih formah pri logaritmih ?
Za reševanje problema sem preizkusil naslednje pristope:

1.poizkus uporabe Loxtonovega izreka o linearnih formah
2.naiven pristop k reševanju problema (logaritmiranje in razvoj v Taylorjevo vrsto - prednosti/pomanjkljivosti, poenostavitev problema s približkom ovrednotenja logaritmov)
3.Bernsteinova aplikacija Loxtonovega izreka in časovna zahtevnost tega algoritma (pisanje algoritma za rešitev problema)

Pa si sedaj poglejmo vsakega od njih pobliže:

3.1. Loxtonov izrek

Opraviti imamo z neenačbo $a^i b^j < c^k$, katere obe strani najprej logaritmiramo, tako dobimo :

$\log a^i b^j < \log c^k$, ker je log nepadajoča funkcija in odtod
$\log a^i + \log b^j < \log c^k$, oz. če prenesemo vse na levo stran, ter vzamemo njeno absolutno vrednost
$|i\log a + j\log b - k\log c| > 0$

Pojdimo sedaj po Loxtonovem izreku za linearno formo pri logaritmih :
določimo c=1, n=3. Ker so a,b,c naravna števila, so tudi multiplikativno neodvisna pozitivna racionalna števila in zato lahko zapišemo $\alpha_1=a$, $\alpha_2=b$ in $\alpha_3=c$ ter zapišimo matriko ranga 1 racionalnih (pravzaprav celih) števil, tako da ustreza naši logaritemski formi :
$\begin{bmatrix} i & j & -k \end{bmatrix}$, torej velja Ω_r je r-ti eksponent iz množice eksponentov, ki nastopajo v neenačbi {i,j,-k} - pazi eksponenti na desni strani nastopajo z nasprotnim predznakom.
Določimo sedaj $A_l \geq 4$ in $B \geq 4$ tako, da bodo višine $H(\alpha_l) \leq A_l$ in $H(\Omega_{lm}) \leq B$. Če izberemo $A_1=\max\{4,a\}$, $A_2=\max\{4,b\}$, $A_3=\max\{4,c\}$ in $B=\max\{4,i,j,k\}$, vrednosti zadoščajo pogojem. Zapišimo sedaj $\Omega=(\log A_1)(\log A_2)(\log A_3)$. Potem velja, ker obstaja le ena vrstica, da je $|i\log a + j\log b - k\log c| > \exp(-(48)^{600}\Omega\log\Omega\log B\Omega)$.

Na žalost pa nam absolutna vrednost razlike pri naši odločitvi prav nič ne pomaga, saj pravzaprav potrebujemo le predznak, torej $\operatorname{sgn}\Lambda_1 = \operatorname{sgn}(i\log a + j\log b - k\log c)$, ki je povezan z našo odločitvijo o neenačbi takole: $\operatorname{sgn}\Lambda_1 = 0$ ali $\operatorname{sgn}\Lambda_1 = +1 \Rightarrow$ neenačba ne drži, ter $\operatorname{sgn}\Lambda_1 = -1 \Rightarrow$ neenačba drži. Torej se Loxtonovega izreka ne da uporabiti neposredno. Poizkusimo še kak drug pristop.

3.2. Taylorjeva vrsta

Drugi pristop, ki sem ga ubral je preprosto preverjanje vrednosti logaritmirane neenačbe $i\log a + j\log b < k\log c$, kar ustreza našemu problemu oz. odločitvi, ob upoštevanju da je log nepadajoča funkcija. Pri tem sem vrednost log izračunal z razvojem logaritemske funkcije v potenčno (t.j. Taylorjevo) vrsto, ki je konvergenčna za vsa naravna števila in se glasi $\ln x = 2\sum_{i=0}^{+\infty} \frac{(x-1)^{2i+1}}{(2i+1)\cdot(x+1)^{2i+1}}$. Seveda se izračun vrednosti vrste dogaja samo do določene meje N, ki je odvisna od natančnosti ε izračuna vrednosti potenčne vrste, tako da velja $\sum_{i=N+1}^{+\infty} \frac{(x-1)^{2i+1}}{(2i+1)\cdot(x+1)^{2i+1}} < \varepsilon$. Da je vrednost te vrste res lahko poljubno majhna, če le vzamemo dovolj velik N, se lahko prepričamo, če se prepričamo v konvergenco te vrste. Ker pa je ta vrsta ravno enaka razvoju funkcije Arth((*x*-1)/(*x*+1)) v vrsto, ki konvergira povsod, velja po definiciji konvergence, da je dovolj pozen preostanek vrste res lahko poljubno majhen. Če sedaj zapišemo algoritem, ki uporablja hitro množenje, lahko tudi za velika naravna števila $a,b,c,i,j,k < n$ hitro (v polinomskem času) odločimo želeno neenačbo; t.j. določimo predznak funkcije $i\log a + j\log b - k\log c$. Da gre res za polinomski čas vemo zato, ker moramo za izračun log funkcij 3x ovrednotiti polinom stopnje N (kjer koeficiente računamo sproti), preostanejo pa le še 3 operacije množenja in seštevanja.

Prednosti tega pristopa so v tem, da je izračun hitro opravljen, da ni zapleten in je preprost oz. zelo nazoren. Pomanjkljivosti tega pristopa pa so v tem, da nimamo eksplicitno podane meje N, do katere se nam ob podani natančnosti »splača« razvijati vrsto, pač pa se moramo zadovoljiti s tem, da ko je trenutni člen dovolj majhen, potrebujemo vedno več (ogromno) členov vsote za isti prirastek. Razvoj vseh treh vrst v našem algoritmu izvajamo do istega N.

3.3. Bernsteinova aplikacija Loxtonovega izreka

Zadnji pristop temelji na Bernsteinovi trditvi, da poljuben kratek interval $[L,U]$ ne more vsebovati mnogo popolnih potenc. Problem je še vedno ta, da moramo za dana naravna števila a,b,c,i,j,k v polinomskem času odločiti, če je $a^i b^j < c^k$. Preformulirajmo naš problem v sledečega : $l=\max(i,j)$, $d=ab$, $e = a^{i-l}b^{j-l}$, $ed^l < c^k$, kjer v *e*-ju nastopa neka potenca *a*-ja ali *b*-ja. Ker je ta odločitev težka samo v primeru, ko imamo opraviti z velikimi števili, si pomagamo še s tem, da obe strani neenačbe okrajšamo, kolikor se to le da. To napravimo tako, da obe strani neenačbe delimo z $\gcd(d,c)^l$.
Še vedno imamo opraviti z neenačbo (sicer okrajšano - beri enostavnejšo od prvotne), oblike $ed^l < c^k$. Sedaj se znebimo še *e*-ja, kar napravimo podobno kot prej pri poenostavljanju, t.j., delimo desno stran z $\gcd(e,c)$, dokler si nista števili tuji. Nato *e* delimo z *d*, *l*-ju prištejemo 1 in postopek ponavljamo, dokler *e*-ja ne odpravimo ($e=1$). Če pri deljenju dobimo ostanek, ga odštejemo pomnoženega z d^l od desne strani. Na ta način smo se dokopali do neke enačbe oblike $d^x < m$. Uporabimo PPC algoritem nad *m* in če ta uspe dobimo neenačbo oblike $d^x < f^y$. Če ne uspe smo pač prisiljeni ali izračunati d^x (vendar le dokler ne presežemo *m*-ja) ali pa poiskati neko popolno potenco, ki bi bila najbližja *m*-ju; v nasprotnem primeru je odločitev lahko takojšnja, če sta le *d* in *x* ali oba manjša ali pa oba večja od *f* in *y*. Če tudi to ne drži se odločimo za : izračun diskretnega logaritma *d* in *f* ter nato množenja (z *x* oz. *y*) in

primerjanja izračunanih rezultatov (Pozor! ne zanima nas spodnja meja logaritemske forme, temveč njen predznak). Kot vidimo si z Bernsteinovo aplikacijo Loxtonovega izreka lahko pomagamo le posredno.

Zapišimo sedaj odločitveni algoritem za dana števila a,b,c,i,j,k še po korakih :

1. *Izračunaj* $l=max(i,j)$, $d=ab$, $e = a^{i-l}b^{j-l}$
2. *Okrajšaj* ed^l *in* c^k *kolikor je to mogoče (deli* ed^l *in* c^k *z* $gcd(d,c)^l$*)*
3. *Odpravimo e (deli e in* c^k *z gcd(e,c), dokler si števili ne postaneta tuji; poizkusimo še z deljenjem e-ja z* ***d'*** *in povečevanjem potence l, delimo* c^k *s preostalim e-jem)*
4. *V prejšnjih dveh korakih pridobljeni števili označimo z* d^x *in m*
5. *Uporabi algoritem PPC nad m; dobimo (f,y)*
6. *Izračun* $\mathrm{sgn}(d^x, f^y)$ *z neposrednim primerjanjem oz. izračunom diskretnega logaritma (npr. z uporabo Shankovega algoritem, Pohlig-Hellmanovega alg. ali pa metode Index Calculus)*

Bistvo algoritma je torej v poenostaviti logaritemske forme iz treh na dva člena in seveda v uporabi Loxtonovega izreka (v 5.koraku). Skupna časovna zahtevnost odločitvenega algoritma je sestavljena iz časovne zahtevnosti Evklidovega algoritma($O((\log_2 n)^2)$), PPC algoritma ($O(\log_2 n)$) in npr. Pohlig-Hellmanovega alg. ($O(\log_2 n)$) in je torej polinomska v log n, kjer je teža problema $n=\max\{a,b,c\}$.

Za zaključek lahko zapišem, da od vseh treh pristopov k reševanju problema, uspemo le z zadnjima dvema. Od teh dveh je časovna zahtevnost prvega (Taylor) polinomska, drugega (Bernstein, Loxton) pa polinomska v logaritmu, zato sam dajem prednost slednjemu.

4. LITERATURA

1. Daniel J. Bernstein, *Detecting perfect powers in essentially linear time*, del avtorjeve doktorske disertacije na Berkeley-u
2. Eric Bach, Jonathan Sorenson, *Sieve algorithms for perfect power testing*, Algorithmica **9** (1993), 313-328
3. Arjen K.Lenstra, Hendrik W.Lenstra, Jr., Mark S. Manasse, John M. Pollard, *The number field sieve*, Lecture Notes in Mathematics 1554, Springer-Verlag, Berlin, 1993, 11-40
4. Henri Cohen, *A course in computational algebraic number theory*, Springer-Verlag, Berlin, 1993
5. Alan Baker, *The Theory of linear forms in logarithms*, in *Transcendence theory: advances and applications*, Academic press (1977), 1-27
6. John H. Loxton, *Some problems involving powers of integers*, Acta Arithmetica **46** (1986), 113-123