4 ARVUTEOORIA

Pierre de Fermat

Arvuteooria on matemaatika haru, mis tegeleb peamiselt täisarvudega, samuti ka objektidega, mis koosnevad täisarvudest. Sellised objektid on näiteks ratsionaalarvud ehk arvud, mida saab esitada kahe täisarvu jagatisena.

Kaasaegsele arvuteooriale panid aluse Pierre de Fermat (1601-1665) ja Leonhard Euler (1707-1783).

Füüsika on ajalooliselt põhinenud pigem reaalarvude ehk pideval matemaatikal, mistõttu viimane oli ka ülikoolide matemaatikakursustes esikohal.

Pärast arvutite leiutamist on arvuteooria ja selle üldistused nagu diskreetne matemaatika oma tähtsust aga suurendanud. Arvutid töötavad enamasti pigem diskreetsete objektidega ja ka kõik arvutis esinevad arvud on piisavalt madalale tasemele minnes täisarvud.

Donald Knuth ütles 1974. aastal: "Peaaegu kõik arvuteooria teoreemid on loomulikul viisil tarvilikud seoses vajadusega panna arvuteid sooritama võimalikult kiireid arvutusi".

Leonhard Euler

Kuna käesoleva raamatu läbivaks teemaks ongi kiirete arvutuste tegemine, on matemaatika, eriti arvuteooria tundmine suureks abiks. Eesti-suguses pisikeses riigis on ka hästi märgata, et need, kes saavad häid tulemusi informaatikaolümpiaadidel, on enamasti väga tugevad ka matemaatikas. Kindlasti pole see ainult juhus!

Matemaatika annab hulgaliselt kavalaid vahendeid, kuidas efektiivsemalt arvutada. Kuna arvutid suudavad teostada erinevaid tehteid ülikiiresti, siis võib tunduda, et neid teadmisi enam eriti vaja ei lähegi, kuid mitmete nippide abil on siiski võimalik programmi tööaega märgatavalt lühendada.

Vahel on ülesannete põhirõhk kusagil mujal, aga arvuteooria alased teadmised ja rakendused on mõne detaili juures abiks. Sageli on programmeerimisvõistlustel aga ka ülesandeid, mis ongi oma põhisisult matemaatikaülesanded. Siin peatükis käsitletakse programmeerimisülesannetes sagedamini ette tulevaid kontseptsioone ja meetodeid.

4.1 JAGUVUS JA JÄÄK

4.1.1 Jaguvus

Jaguvus on arvuteooria keskseks mõisteks. Öeldakse, et täisarv n **jagub** täisarvuga m, kui leidub selline täisarv k, et $n = km$. Jaguvust võib tähistada kahel moel: $n \vdots m$ (arv n jagub arvuga m) või $m|n$ (arv m jagab arvu n; arv m on arvu n jagaja). Näiteks arv 12 jagub arvuga 4, sest $3 \cdot 4 = 12$. Arv 5 aga ei jaga arvu 12, sest ei leidu sellist täisarvu k, et $k \cdot 5 = 12$.

Jaguvusel on järgmised olulised põhiomadused:

1. $a|a$ – mistahes täisarv jagub iseendaga, näiteks $12|12$.
2. $1|a$ – arv 1 jagab mistahes täisarvu, näiteks $1|31$.
3. $a|0$ – null jagub mistahes arvuga, näiteks $5|0$, sest $5 \cdot 0 = 0$
4. Kui $c|a$ ja $c|b$, siis $c|(a+b)$ ja $c|(a-b)$.
5. Kui $c|(a+b)$ ja $c|a$, siis $c|b$.
6. Kui $c|a$ ja $c \nmid b$, siis $c \nmid (a+b)$.
7. Kui $c|a$, siis $c|ka$, kus $k \in Z$.
8. Kui $a|b$ ja $b|c$, siis $a|c$.
9. Kui $a|b$ ja $c|d$, siis $ac|bd$
10. Kui $a|b$ ja $b|a$ ja $a, b \in N$, siis $a = b$.
11. Kui $b|a$ ja $b = cd$, siis $c|a$ ja $d|a$.
12. Kui $a|c$ ja $b|c$ ja ei leidu sellist arvu $k > 1$, et $k|a$ ja $k|b$ (a ja b on ühistegurita), siis $ab|c$.
13. Kui $d|a$ ja $b|d$, siis $d/b|a/b$.
14. Kui on antud n järjestikust arvu, siis jagub alati täpselt üks neist arvuga n.

Enamik neist paistavad algul triviaalsed, kuid nende abil saab hiljem põhjendada juba oluliselt huvitavamaid tulemusi.

4.1.2 Jääk

Kui meil on täisarv n ja naturaalarv $m > 0$, saab alati leida täisarvud k ja r, nii et $n = km + r$, kus $|r| < m$.

Sellisel juhul nimetatakse k **jagatiseks** (n/m), m **mooduliks** ning r **jäägiks** mooduli m järgi. Juhul, kui $r = 0$, siis $n \vdots m$. Näiteks $12 = 2 \cdot 5 + 2$, seega arvu 12 jääk mooduli 5 järgi on 2. Arvu 12 jääk mooduli 8 järgi on 4, sest $12 = 1 \cdot 8 + 4$.

Jäägi absoluutväärtus on alati väiksem mooduli absoluutväärtusest. Näiteks arvu 12 saab esitada ka kujul $12 = 1 * 5 + 7$, kuid 7 ei ole jääk mooduli 5 järgi, sest $7 > 5$.

4.1.3 Jäägi leidmine programmeerimiskeeltes

Arvust jäägi leidmine mooduli järgi on programmeerimises väga tavapärane tehe. Siin raamatus käsitletavates keeltes kasutatakse tehtemärgina %-märki. Tehe `7 % 3` annab vastuseks ühe.

Positiivsete arvudega on asi alati ühtmoodi, kuid negatiivsete arvude korral on oluline teada, kuidas käitub konkreetne programmeerimiskeel. Peamiselt on kasutuses kolm varianti:

1. Jagatis ümardatakse alati nulli suunas. Sellisel juhul jääk on sama märgiga, mis jagatav: $r = a - n\ trunc(a/n)$. Seda varianti kasutavad nii Java kui ka C++ alates versioonist 11. Varasemates C++ versioonides jäeti selles osas vabad käed ning kasutus oleneb konkreetsest keele implementatsioonist.
2. Jagatis ümardatakse alati alla. Sellisel juhul jääk on sama märgiga, mis jagaja: $r = a - n \lfloor a/n \rfloor$. Seda varianti kasutab Python.
3. Jääk on alati positiivne. Sellisel juhul positiivse jagaja korral jagatis ümardatakse alla, negatiivse jagaja korral aga üles:
$$q = \begin{cases} \lfloor a/n \rfloor, kui\ n > 0, \\ \lceil a/n \rceil, kui\ n < 0, \end{cases}$$
ja jääk avaldub kujul $r = a - |n| \lfloor a/|n| \rfloor$.

Järgnevas tabelis on toodud näited tulemustest erinevatel juhtudel:

Tehe	Jagatava järgi	Jagaja järgi	Alati positiivne
5 % 3	2	2	2
-5 % 3	-2	1	1
5 % -3	2	-1	2
-5 % -3	-2	-2	1

Pane tähele, et negatiivsest tulemusest saab vajadusel alati positiivse, liites leitud jäägile juurde mooduli ehk jagaja absoluutväärtuse. Kui peaks olema vaja positiivsest negatiivset saada, siis piisab, kui lahutada mooduli absoluutväärtus.

Kui esmapilgul tundub tegemist olevat küllaltki ebaolulise detailiga, siis algajatel programmeerijatel võib tekkida sellega seoses ootamatuid vigu. Näiteks kirjutades funktsiooni `OnPaaritu` paarsuse kontrolliks kujul:

```
bool OnPaaritu(int a) {
    return a % 2 == 1;
}
```

võib selguda, et negatiivsete täisarvude korral tagastab see funktsioon alati väärtuse `false`, kuna näiteks `-1 % 2` tulemuseks on `-1`. Kindlam on alati kirjutada:

```
bool OnPaaritu(int a) {
    return a % 2 != 0;
}
```

4.1.4 Ülesanne: loosiümbrikud

Järgmiseks üks lihtne soojendusülesanne:

Suures kastis on hulk ümbrikke. Mõned neist on tühjad ja mõned täidetud. Lapsed võtavad üks-haaval kastist ümbrikke. Iga laps võtab korraga kastist täpselt 2 ümbrikku ning juhul, kui need on mõlemad täis või mõlemad tühjad, paneb ta kasti tagasi tühja ümbriku (kui mõlemad on täis, eemaldab ta ühest sisu ning paneb tühjendatud ümbriku tagasi), kui aga üks ümbrik on tühi ja teine täis, peab ta tagasi panema täidetud ümbriku. Lõpuks jääb kasti üks ümbrik. Kas see on tühi või täis? Sisendi esimesel ja ainsal real on kaks täisarvu `m` ja `n`, vastavalt tühjade ja täidetud ümbrike arv. Väljastada `TÜHI` kui viimane ümbrik on tühi, `TÄIS`, kui viimane ümbrik on täis või `EI TEA` , kui ei ole võimalik otsustada, kas ümbrik on tühi või täis.

NÄIDE :

`2 1`

Vastus:

`TÄIS`

(Kui esimene laps võtab mõlemad tühjad ümbrikud, siis ta paneb neist ühe tagasi ja teisele lapsele jääb kasti üks tühi ja üks täis ümbrik, millest ta peab tagasi panema täis ümbriku. Kui esimene laps võtab tühja ja täis ümbriku, siis ta peab tagasi panema täidetud ümbriku ja nii jääb teisele lapsele samuti tühi ja täis ümbrik, millest ta peab tagasi panema täidetud ümbriku. Nii jääb igal juhul lõpuks kasti täidetud ümbrik).

Kuigi esmapilgul võib see ülesanne tunduda üsna keerukas, siis lähemal vaatlusel pole raske märgata, et täis ümbrikud lahkuvad kastist ainult paarikaupa – kui keegi õnnelik saab korraga kaks täidetud ümbrikku. Samuti ei teki võtmise ajal täidetud ümbrikke juurde.

Viimasel võtmisel on kolm võimalust: mõlemad ümbrikud on tühjad, mõlemad on täis või on üks ümbrik täis ja üks tühi. Kui mõlemad ümbrikud on tühjad või täis, läheb kasti tagasi tühi ümbrik ja viimane ümbrik kastis on seega tühi. Kui üks ümbrik on tühi ja teine täis, siis sel juhul läheb kasti tagasi täis ümbrik, mis jääb sinna viimaseks. Kuna täis ümbrikud lahkuvad kastist ainult paarikaupa, siis saab lõpuks alles olla üks tühi ja üks täis ümbrik ainult siis, kui täis ümbrikke oli kastis kohe alguses paaritu arv. Kehtib ka vastupidine – kui täis ümbrikke oli alguses paaritu arv, siis peab enne viimast võtmist olema kastis täpselt üks täidetud ümbrik. Seega piisab kogu ülesande lahendamiseks ainult sellest, et vaatame, kas täidetud ümbrikke oli algselt kastis paaris või paaritu arv:

```
int main()
{
    int tyhjad, tais;
    cin >> tyhjad >> tais;
    if (tais % 2 == 0) // Kui täis ümbrike on paarisarv,
        cout << "TÜHI"; // siis viimane ümbrik on tühi,
    else // muidu (kui on paaritu)
        cout << "TÄIS"; // on viimane ümbrik täis
    return 0;
}
```

Sellise ülesande lahendamiseks pole tegelikult muidugi isegi arvutit vaja, kuid võistlustel (ja ka päriselus!) esineb siiski aeg-ajalt ülesandeid, kus keerulise vormi sisse on peidetud väga lihtne lahendus. Sarnased võtted võivad sageli osutuda oluliseks ka keerulisemate ülesannete jaoks optimaalsemate algoritmite leidmisel.

4.1.5 Arvutamine moodulitega

Sageli saab tehte jääki leida tehte operandide järgi, ilma tehte vastust välja arvutamata. Selleks on kõigepealt vaja teada kaht lihtsat, kuid olulist seost:

1. $a \bmod m = a,\ kui\ 0 \leq a < m$
2. $(a \bmod m) \bmod m\ =\ a \bmod m$

Teine seos tähendab, et mitmekordsel jäägi võtmisel sama mooduli järgi jääb tulemus samaks. Kuna $a \bmod m$ tulemus on hulgas $(0, 1 \ldots m-1)$, ning kehtib esimene seos, siis pole selle kehtivuses raske veenduda. Näiteks

$$(17 \bmod 3) \bmod 3\ =\ 2 \bmod 3\ =\ 2$$

Liitmine ja lahutamine

Liitmise jaoks saab kasutada seost:

$$(a + b) \bmod m\ =\ [(a \bmod m) +\ (b \bmod m)] \bmod m.$$

Näiteks:

$$(17 + 8) \bmod 3 = [(17 \bmod 3) + (8 \bmod 3)] \bmod 3 = (2 + 2) \bmod 3 = 4 \bmod 3 = 1.$$

Lahutamine on põhimõtteliselt sarnane liitmisele:

$$(17 - 8) \bmod 3 = [(17 \bmod 3) - (8 \bmod 3)] \bmod 3 = (2 - 2) \bmod 3 = 0 \bmod 3 = 0.$$

Aga mis teha, kui $a \bmod m < b \bmod m$, nagu tehtes $(9 - 7) \bmod 3$? Sellisel juhul

$$(9 - 7) \bmod 3 = [(9 \bmod 3) - (7 \bmod 3)] \bmod 3 = (0 - 1) \bmod 3 = -1 \bmod 3 = ?$$

Nagu juba teame, sõltub tehte $-1\ mod\ 3$ vastus juba konkreetsest programmeerimiskeelest või lausa implementatsioonist: Python annab vastuseks sobivalt 2, Java aga -1. Sellisel juhul saab negatiivsest jäägist positiivse, liites vastusele juurde mooduli, antud näites siis $-1 + 3 = 2$.

Kindla peale kehtib seos:

$$(a - b)\ mod\ m = [(a\ mod\ m) - (b\ mod\ m) + m]\ mod\ m,$$

millest saame, et

$$(9 - 7)\ mod\ 3 = [(9\ mod\ 3) - (7\ mod\ 3) + 3]\ mod\ 3 = (0 - 1 + 3)\ mod\ 3 = 2\ mod\ 3 = 2.$$

Korrutamine ja astendamine

Korrutamise puhul kehtib samuti seos:

$$(ab)\ mod\ m = [(a\ mod\ m)(b\ mod\ m)]\ mod\ m$$

Näiteks:

$$(5 * 8) mod\ 3 = [(5\ mod\ 3) \cdot (8\ mod\ 3)] mod\ 3 = (2 \cdot 2) mod\ 3 = 4\ mod\ 3 = 1$$

Astendamise puhul kehtib seos:

$$a^b\ mod\ m = (a\ mod\ m)^b\ mod\ m$$

Näiteks:

$$5^4\ mod\ 3 = (5\ mod\ 3)^4\ mod\ 3 = 2^4\ mod\ 3 = 16\ mod\ 3 = 1$$

Kuna just korrutades ja eriti astendades lähevad arvud kiiresti väga suureks, siis on see koht, kus moodularvutusest on sageli abi. Küllaltki levinud on ülesanded, kus tuleb leida mingi suure arvu, näiteks 574232^{100}, viimane koht. Viimase ehk üheliste koha leidmine on tegelikult jäägi leidmine mooduli 10 järgi (kahe viimase koha leidmine on ekvivalentne jäägi leidmisega mooduli 100 järgi jne).

Jagamine

Jagamine on mõnevõrra keerulisem, liitmise ja korrutamisega analoogsed seosed jagamise puhul ei kehti. Näiteks:

$$(30 \div 2)\ mod\ 8 = 15\ mod\ 8 = 7,$$

aga

$$[(30\ mod\ 8) \div (2\ mod\ 8)]\ mod\ 8 = (6 \div 2)\ mod\ 8 = 3\ mod\ 8 = 3.$$

Jagamise puhul tuleb appi võtta **modulaarne pöördväärtus**. Arvu a modulaarne pöördväärtus mooduli m suhtes on selline arv a^{-1}, mille korral kehtib seos $a \cdot a^{-1}\ mod\ m = 1$.

Jagamisel kehtib seos:

$$(a \div b)\ mod\ m = [(a\ mod\ m) \cdot (b^{-1}\ mod\ m)]\ mod\ m,$$

kus b^{-1} on arvu b modulaarne pöördväärtus mooduli m suhtes.

Näiteks:

$$(20 \div 5)\ mod\ 7 = [(20\ mod\ 7) \cdot (3\ mod\ 7)]\ mod\ 7 = (6 \cdot 3)\ mod\ 7 = 18\ mod\ 7 = 4,$$

Kus arv 3 on arvu 5 modulaarne pöördväärtus mooduli 7 suhtes, sest $(3 \cdot 5)\ mod\ 7 = 1$.

Arvul eksisteerib modulaarne pöördväärtus parajasti siis, kui arv ja moodul on ühistegurita.

4.1.6 Ülesanne: anagrammid 2

Mõnikord on arvutusülesande vastus nii suur, et see ei mahu suurimassegi olemasolevasse täisarvutüüpi ära või ei peeta praktiliseks mitmekümnekohalist vastust küsida. Sageli küsitakse programmeerimisvõistlustel vastust mingi etteantud mooduli järgi.

Teises peatükis punkt „2.7.2 Permutatsioonid“ all oli ülesanne, kus tuli leida etteantud nimest kõik võimalikud anagrammid. Selles ülesandes jäid nimede pikkused küllaltki lühikesteks. Vaatame selle ülesande suuremat varianti, kus tähti, millest anagramme moodustatakse, on juba palju rohkem:

Ingliskeelses Scrabble'i komplektis on 100 täheklotsi: E tähega klotse on kõige rohkem - 12, A ja I tähega on 9, O-ga 8, N-i, R-i ja T-ga 6, D, L-i, S-i ja U-ga 4 klosti ja G-ga kolm klotsi. Kaks korda on B, C, F, H, M, P, V, W ja Y. J, K, Q, X ja Z esinevad ainult ühe korra. Lisaks on komplektis 2 tühja klotsi. Scrabble'i komplekt on olemas ka Eesti keele jaoks, selles on 102 tähte ning kõige rohkem on A tähega klotse – 10. Klotside jaotus on olemas ka tehiskeelte, näiteks Klingoni keele jaoks.
Väikesel Pärdil on ingliskeelne Scrabble'i komplekt. Lugeda ta veel ei oska, kuid talle meeldib neid klotse ühte ritta laduda, nii et moodustub üks n-täheline „sõna“. Pärdi õde Piret soovib aga välja arvutada, mitu erinevat n-tähelist „sõna“ on Pärdil võimalik väljavalitud klotsidest moodustada. Pärt ei vali kunagi tühje klotse.
Sisendi esimesel ja ainsal real on 26 arvu. Esimene arv on A-tähega klotside arv, teine B tähega klotside arv, jne, vastavalt inglise keele tähestikule. Viimane on seega Z-tähega klotside arv. Arvud on eraldatud tühikutega. Klotsid moodustavad alamhulga ingliskeelsest Scrabble'i komplektist.
Väljundisse kirjutada üks arv: Erinevate n-täheliste järjendite arv mooduli 10^9+7 järgi.

NÄIDE 1:

```
2 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
```

Vastus:

```
12
```

(AABC, AACB, ABAC, ABCA, ACAB, ACBA, BAAC, BACA, BCAA, CAAB, CABA, CBAA)

NÄIDE 2:

```
9 2 2 4 12 2 3 2 9 1 1 4 2 6 8 2 1 6 4 6 4 2 2 1 2 1
```

Vastus:

```
820218216
```

(Tegelik võimaluste arv on ligikaudu 9.74×10^{111})

Sel korral kõiki permutatsioone leida ei ole vaja, on vaja ainult nende arvu. Nagu juba 2. peatükis selgus, on selle jaoks olemas järgmine valem:

$$\frac{n!}{m_1!\, m_2! \ldots m_k!}$$

Nii ei ole tegelikult oluline teada, millised tähed konkreetselt kasutuses on, on vaja vaid täheklotside koguarvu ning iga erineva klotsi esinemissagedust.

Väikese n korral on tegu väga lihtsa arvutusülesandega, problemaatiliseks teeb antud ülesande see, et juba 22! on 21-kohaline ega ei mahu kuidagi suurimassegi täisarvutüüpi ära. Ülesandes aga võib murru lugejaks tulla lausa 98! (Pärdil on 98 tähega klotsi), mis on juba 154-kohaline arv.

Samas ega täpset vastust leida polegi ju vaja, piisab jäägist mooduli 10^9+7 järgi. Eelnevalt tuli välja, et korrutamine jääkidega ei ole kuigi keeruline. Isegi 100! leidmine mooduli 1 000 000 007 järgi ei ole kuigi vaevanõudev, kasutades järgnevat funktsiooni:

```
int fakt(int n, int m)
{
    int i = 1;
    long long f = 1;
    while (i < n) {
        f *= ++i;
        f %= m; // iga korrutamise järel leiame jäägi
    }
    return f;
}
```

Keeruliseks teeb asja jagamistehe, sest nagu eespool selgus, siis pole lugeja ja nimetaja jääkide leidmisest eraldi kuigi palju kasu.

Murru $\frac{n!}{m_1!m_2!\dots m_k!}$ saab teisendada täisarvude korrutiseks, kasutades omadust, et n järjestikuse täisarvu korrutis jagub alati $n!$-ga (huvi korral loe lähemalt https://math.stackexchange.com/questions/12065/the-product-of-n-consecutive-integers-is-divisible-by-n-factorial).

Näiteks kui on 10 klotsi: esimest tähte 4, teist 3, kolmandat 2 ja neljandat 1, siis erinevate järjendite arv on

$$\frac{10!}{4!\,3!\,2!\,1!} = \frac{1\cdot 2\cdot 3\cdot 4}{4!}\cdot\frac{5\cdot 6\cdot 7}{3!}\cdot\frac{8\cdot 9}{2!}\cdot\frac{10}{1!} = 1\cdot 35\cdot 36\cdot 10 = 12600.$$

Klotsid on hea esinemissageduse järgi sorteerida, nii saab paremini kontrollida, kui suureks tehete tulemused lähevad. Siinne näidislahendus annab õige vastuse ka siis, kui 12 E-tähe asemel on hoopis 12 Z-tähte, kuid sorteerimata jättes võib tekkida olukord, kus viimaseks lugejaks jääb $89\cdot 90\cdot \dots \cdot 100$, mis ei mahu enam 64-bitisesse arvutüüpi ära.

Ülesande lahendus:

```
#include <iostream>
#include <algorithm>
using namespace std;

int main()
{
    long long moodul = 1e9 + 7; // moodul, mille järgi vastuse leiame
    int klotsid[26] = {0}; // massiiv erinevate tähtede arvu jaoks
    int tahti = 0; // erinevaid kasutusesolevaid tähti
    int i, j; // tsüklimuutujad
    int algus = 1; // abumuutuja n! tükeldamisel: uue korrutise algus
    int n = 0; // klotside arv kokku
    long long vastus = 1; // ülesande vastus
    for (i = 0; i < 26; i++) {
        int s;
        cin >> s;
        // kui seda tähte ei ole, siis jätkame järgmise lugemisega
        if (s == 0) continue;
        klotsid[tahti] = s;
        tahti++; // suurendame erinevate tähtede arvu
        n += s; // liidame kogu klotside arvule selle tähega klotside arvu
    }
    sort(klotsid, klotsid + tahti); // sorteerime suuruse järgi
    for (i = tahti-1; i >= 0; i--) { // alustame sagedasemast
        int lopp = algus + klotsid[i];
        long long f = 1; // järgmise klotsid[i] arvu korrutis
        long long f2 = 1; // m[i] faktoriaal
        for (j = algus; j < lopp; j++) {
            f *= j;
        }
        for (j = 2; j <= klotsid[i]; j++) {
            f2 *= j;
        }
        long long t = (f / f2) % moodul;
        vastus *= t;
        vastus = vastus % moodul;
        algus = lopp;
    }
    cout << vastus;
    cin >> i;
    return 0;
}
```

Tasub ka tähele panna, et esimene jagatis on tegelikult alati 1, täiesti mõistlik on see arvutamata jätta. Selleks piisab, kui põhitsüklit alustada `i = tahti-2` ning enne tsüklit muuta muutuja `algus` väärtust: `algus += klotsid[tahti-1]`.

4.2 SÜT JA VÜK

Arvuteoorias tähtsateks mõisteteks, millega ka koolimatemaatikas juba üsna varakult kokku puututakse, on suurim ühistegur (SÜT) ja vähim ühiskordne(VÜK).

Naturaalarvude a ja b **suurimaks ühisteguriks** nimetatakse suurimat naturaalarvu c, nii et $c|a$ ja $c|b$. Naturaalarvude a ja b **vähimaks ühiskordseks** nimetatakse vähimat naturaalarvu c, nii et $a|c$ ja $b|c$.

Koolimatemaatikas ja ka igapäevaelus on suurimat ühistegurit vaja enamasti murdude taandamiseks ning vähimat ühiskordset erinimeliste murdude liitmisel ühise nimetaja leidmiseks.

4.2.1 Eukleidese algoritm suurima ühisteguri leidmiseks

Kreeka matemaatik Eukleides Aleksandriast märkas juba 300 aastat enne meie aega, et arvude a ja b suurim ühistegur c jagab lisaks arvudele a ja b ka nende vahet $a - b$. See on tegelikult triviaalne, sest kui $c|a$, siis jaguvuse definitsiooni järgi $a = xc$ ja analoogselt, kui $c|b$, siis $b = yc$, seega

$$a - b = xc - yc = c(x - y) \implies c|(a - b).$$

Praktikas tähendab see, et kui meil on vaja leida kahe suure arvu a ja b $(a > b)$ suurimat ühistegurit, siis saab ülesande taandada lihtsamale kujule. Nimelt on a ja b suurim ühistegur sama, mis $a - b$ ja b ühistegur. Sama võtet võib korrata korduvalt. Näiteks arvude 63 ja 49 suurima ühisteguri saab leida nii:

$$\begin{aligned} SÜT(63,49) = {} & SÜT(63-49,49) = SÜT(14,49) = SÜT(14,49-14) = SÜT(14,35) \\ = {} & SÜT(14,35-14) = SÜT(14,21) = SÜT(14,21-14) = SÜT(14,7) \\ = {} & SÜT(14-7,7) = SÜT(7,7) = 7. \end{aligned}$$

Siin on täpselt sama vahetu lahendus C++ funktsioonina:

```
int syt(int a, int b) {
    while (a != b) {
        if (a > b) {
            a -= b;
        }
        else {
            b -= a;
        }
    }
    return a;
}
```

Ülaltoodud näites torkab silma, et näiteks arvu 14 lahutatakse mitu korda järjest, õigemini nii kaua, kuni saadud vahe on ≤14. Seega võiks ju lahutada kohe sobiva 14-kordse, antud näites siis $3 \cdot 14 = 42$. Veelgi enam: kui lahutada arvust a mingi arvu b kordset, kuni vahe on väiksem kui lahutatav arv, siis tegelikult leitakse arvu a jääki mooduli b järgi. Jäägi definitsioonist:

$$a = kb + r \implies a - kb = r\,,$$

Kus r on a jääk mooduli b järgi.

Suurima ühisteguri leidmine jäägi abil annab kompaktse rekursiivse lahenduse:

```
int syt(int a, int b) {
       return (b == 0) ? a : syt(b, a % b);
}
```

4.2.2 Vähima ühiskordse leidmine

Kahe arvu vähima ühiskordse leidmisel kasutatakse omadust, et kahe arvu suurima ühisteguri ja vähima ühiskordse korrutis on võrdne nende arvude korrutisega:

$$a \cdot b = SÜT(a,b) \cdot VÜK(a,b),$$

millest

$$VÜK(a,b) = \frac{a \cdot b}{SÜT(a,b)}.$$

Seda omadust kasutatakse programmides enamasti vähima ühiskordse leidmiseks:

```
int vyk(int a, int b) {
	int temp = syt(a, b);
	return temp ? (a / temp * b) : 0;
}
```

Siin tasub tähele panna, et `a*b/syt(a,b)` asemel tagastatakse `a/syt(a,b)*b` . Sellisel moel hoitakse arvud väiksemad ja välditakse nii võimalikku ületäitumist. Kuna definitsiooni järgi $SÜT(a,b)|a$, siis võib rahulikult jagamistehte `a/syt(a,b)` sooritada enne `b`-ga korrutamist.

Suurimat ühistegurit ja vähimat ühiskordset saab leida ka rohkem kui kahel arvul. Mitme arvu suurim ühistegur on suurim selline arv, mis jagab igaüht neist arvudest, ning vähim ühiskordne on kõige väiksem selline arv, mida kõik need arvud jagavad. $SÜT(a,b,c) = SÜT(a,SÜT(b,c))$ ja samuti $VÜK(a,b,c) = VÜK(a,VÜK(b,c))$.

4.2.3 Ülesanne: hammasrattad

Järgmine ülesanne „Hammasrattad“ oli 2017. aastal Eesti informaatikaolümpiaadi lõppvoorus:

Kellassepal on tööpink, mis suudab teha `M` kuni `N` hambaga hammasrattaid. Kirjutada programm, mis leiab, mitu erinevat kahest hammasrattast koosnevat ülekannet saab selle pingi abil teha. Kahte ülekannet loeme erinevaks, kui nende ülekandearvud (esimese ratta hammaste arv jagatud teise ratta hammaste arvuga) on erinevad.
Sisendi esimesel real on kaks tühikuga eraldatud täisarvu `M` ja `N` (`1≤M≤N≤1000`), mis tähistavad minimaalset ja maksimaalset hammaste arvu hammasratastel, mida antud pingil teha saab.

Väljundi ainsale reale väljastada võimalike ülekandesuhete arv.

NÄIDE:

```
2 6
```

Vastus:

```
17
```

(Võimalikud suhted on järgmised: 2 : 6; 2 : 5; 2 : 4 = 3 : 6; 3 : 5; 2 : 3 = 4 : 6; 3 : 4; 4 : 5; 5 : 6; 2 : 2 = 3 : 3 = 4 : 4 = 5 : 5 = 6 : 6; 6 : 5; 5 : 4; 4 : 3; 3 : 2 = 6 : 4; 5 : 3; 4 : 2 = 6 : 3; 5 : 2 ja 6 : 2.)

Lihtsa topelttsükliga saame konstrueerida kõik võimalikud hammasrataste paarid. Ülesande keerukamaks kohaks on see, kuidas leida sama suhtega paare. Suhteid saab vaadelda murdudena ja nii on sama suhtega paaride tuvastamine tegelikult murru taandamise ülesanne. Taandada on aga kõige lihtsam nii, et leiame lugeja ja nimetaja suurima ühisteguri ja jagame nii lugeja kui ka nimetaja sellega läbi. Kuna võib tulla mitmeid ekvivalentseid ülekandeid, siis on kõige mugavam hoida ülekandeid `set` andmestruktuuris – seal võib iga element esineda täpselt ühe korra ning katset lisada korduvat elementi ignoreeritakse.

Siin on programm, mis antud ülesande lahendab:

```
#include <iostream>
#include <set>

using namespace std;
int syt(int a, int b) {
    while (b > 0) {
        int c = b;
        b = a % b;
        a = c;
    }
    return a;
}

int main()
{
    int m, n;
    cin >> m >> n;

    std::set<pair<int, int>> s;
    for (int a = m; a <= n; a++) { // esimese hammasratta võimalikud hambad
        for (int b = m; b <= n; b++) { // teise hammasratta võimalikud hambad
            int d = syt(a, b); // leiame a ja b suurima ühisteguri
            // ja jagame a ja b sellega, et leida ülekannet
            pair<int, int> p = make_pair(a / d, b / d);
            s.insert(p);
        }
    }
    cout << s.size();
    return 0;
}
```

4.2.4 Diofantilised võrrandid

Diophantos Aleksandriast oli vanakreeka matemaatik, kes tegeles 3. sajandil algebraliste võrrandite lahendamisega. Tema järgi nimetataksegi täisarvuliste kordajatega algebralist määramata võrrandit, millele otsitakse täisarvulisi lahendeid, **diofantiliseks võrrandiks.** Võrrandit kujul $ax + by = c$ nimetatakse **lineaarseks diofantiliseks võrrandiks**. Koolis õpitakse, et võrrandi(süsteemi) lahendamiseks peab võrrandeid olema vähemalt sama palju kui tundmatuid. Hoolimata sellest, et antud juhul on ühe võrrandi kohta kaks tundmatut, on sellisel spetsiifilisel kujul võrranditele võimalik üldlahendid leida.

Selleks on kõigepealt vaja leida a ja b suurim ühistegur. Olgu see d. Võrrandil puuduvad lahendid, kui $d \nmid c$. Kui aga $d|c$, on võrrandil lõpmatu hulk lahendeid. Lahendamiseks saab kasutada veidi täiendatud Eukleidese algoritmi suurima ühisteguri leidmiseks:

```
int dsyt(int a, int b, int &x, int &y) {
    if (b == 0) {
        x = 1;
        y = 0;
        return a;
    }

    int x1, y1, syt = dsyt(b, a % b, x1, y1);
    x = y1;
    y = x1 - (a / b) * y1;
    return syt;
}
```

See algoritm annab meile x' ja y', nii et $ax' + by' = S\ddot{U}T(a,b)$. Korrutades selle võrrandi mõlemat poolt $c/S\ddot{U}T(a,b)$-ga, saame

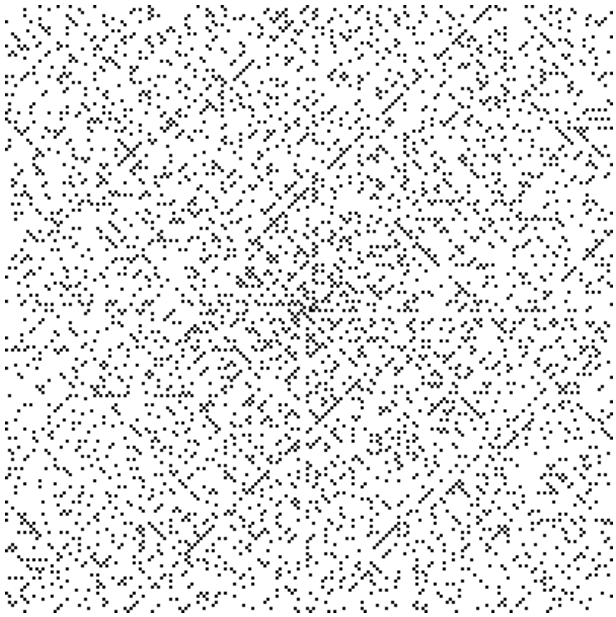
$$a\frac{cx'}{S\ddot{U}T(a,b)} + b\frac{cy'}{S\ddot{U}T(a,b)} = c,$$

kus $x_0 = \frac{cx'}{S\ddot{U}T(a,b)}$ ja $y_0 = \frac{cy'}{S\ddot{U}T(a,b)}$, on esialgse võrrandi ühed võimalikud lahendid. Üldlahend avaldub kujul $x = x_0 + n * \frac{b}{S\ddot{U}T(a,b)}$ ja $y = y_0 - n * \frac{a}{S\ddot{U}T(a,b)}$, kus $n \in Z$.

4.3 ALGARVUD

Algarvuks nimetatakse ühest suuremat naturaalarvu, mis jagub vaid arvuga 1 ja iseendaga. Arvuteoorias on algarvudel väga tähtis roll: **aritmeetika põhiteoreem** ütleb, et iga ühest suurem naturaalarv on ühesel viisil (tegurite järjekorda arvestamata) lahutatav oma algarvuliste tegurite ehk **algtegurite** korrutiseks. Näiteks $6 = 2 * 3$ ja $16 = 2 * 2 * 2 * 2$.

Algarvud on matemaatikuid huvitanud juba väga varastest aegadest. Kuidas neid leida ja ära tunda? Kui palju neid on? Kui tihedalt nad esinevad?



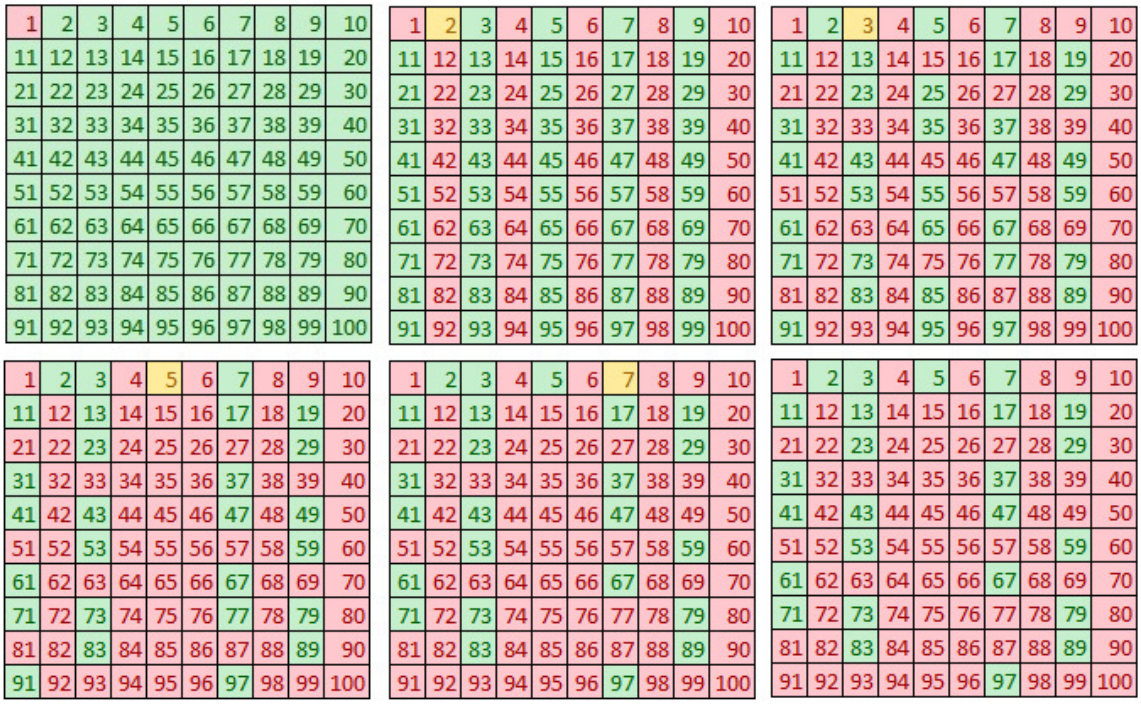
Ulami spiraal – joonis, kus keskpunktist alates on spiraalis loendatud naturaalarve ning nende seas algarvud musta punktiga ära märgitud. Pane tähele tekkivaid mustreid!

Eukleides, lisaks ühisteguri leidmise algoritmile ja paljudele muudele avastustele matemaatikas, tõestas ka, et algarve on lõputult. Algarvude loendamine aga osutus juba keerukamaks ülesandeks. Arvust n väiksemate algarvude arvu tähistatakse $\pi(n)$. Näiteks $\pi(100) = 25$, see tähendab, et 100 väiksemaid algarve on 25. Arvust x väiksemate algarvude arv on

$$\pi(x) \approx \frac{x}{\ln x}$$

4.3.1 Eratosthenese sõel

Kreeka õpetlane Eratosthenes Küreenest mõtles juba paarsada aastat enne meie aega välja algoritmi algarvude leidmiseks. Tema järgi nimetatakse vastavat meetodit Eratosthenese sõelaks, mõnikord ka lihtsalt sõelaks. Vaatame järgmiseks, kuidas sõela kasutades käsitsi algarve välja sõeluda. Kõigepealt kirjutame vaadeldava hulga arve järjest üles, näiteks sellise tabelina, nagu alljärgneval joonisel.



Alguses on kõik arvud potentsiaalsed algarvukandidaadid ja märgitud tabelis rohelisena. Ainsaks erandiks on arv 1, mille võime kohe märkida punaseks kui mitte-algarvu. Sellele situatsioonile vastab joonisel esimese rea esimene tabel. Edasi asumegi otsima neid arve, mis on mingi muu arvu kordsed ehk **kordarvud**. Selleks valime vähima vaatlemata arvu, mis on potentsiaalne algav (tabelis roheline). See arv ise on kindlasti algarv, sest ta ei jagu ühegi endast väiksema arvuga peale ühe. Esimene selline arv on 2. Nüüd on teada, et kaks on algarv, aga kahe kordsed ehk arvud, mis kahega jaguvad, ei ole algarvud. Kuna 2|2, siis ka 2|2+2 ja 2|2+2+2 jne. Märgime tabelis iga teise arvu pärast 2 punaseks kui mitte-algarvu. Sellele vastab esimese rea keskmine tabel. Parasjagu vaadeldav arv 2 on selles tabelis märgitud kollasega lihtsamaks visuaalseks eristamiseks.

Järgmiseks kordame sama protseduuri arvuga 3 (ülemise rea parempoolne tabel). Kuna arv 4 on juba punane ehk välja sõelutud kui mitte-algarv, siis sellega ei ole vaja enam midagi teha. Sõelume välja veel 5 ja 7 kordsed (alumise rea vasak ja keskmine tabel). Alumise rea parempoolses tabelis on lõppseis – rohelisega on märgitud veel ainult algarvud, teised on läbi sõela „pudenenud“ kui mingi arvu kordsed.

Aga miks pole vaja enam sõeluda arvuga 11? Tõesti, kui seda proovida, siis leiaksime arvud 22, 33, 44, 55, 66, 77, 88 ja 99, mis kõik on juba punased. Tabelis on arvud kuni 100ni ja $100 = 10 \cdot 10$. Selleks, et korrutis $a \cdot b \leq 100$, ei saa mõlemad arvud a ja b olla samaaegselt suuremad kümnest. Kui üks tegur aga on väiksem kui 10, siis oleme seda juba vaadanud ja selle kordsed punaseks märkinud. Seega piisab alati, kui vaatame potentsiaalseid tegureid kuni $\sqrt{n}$, kus n on arv, milleni algarve otsime.

Sõela puhul veelgi efektiivsem on tähelepanek, et kui me sõelume arvu i tegureid, siis võib sõelumist alustada arvust i^2, sest analoogselt eelmise põhjendusega on i-ni jõudes välja sõelutud juba kõik väiksemad i kordsed ehk arvud kujul $k \cdot i$, kus $1 < k < i$.

Lihtne sõela algoritm on järgmine:

```
int SP = 1000 // sõela pikkus
bool soel[SP]; // algarvude jada

void alg() {
    int i, j;
    for (i = 0; i < SP; i++)
        soel[i] = 1; // alguses paneme kõik algarvudeks
    soel[0] = 0; // 0 ei ole algarv
    soel[1] = 0; // 1 ei ole algarv
    for (i = 2; i * i <= SP; i++) {
        if (soel[i])
            for (j = i * i; j < SP; j += i)
                soel[j] = 0;
    }
}
```

4.3.2 Algarvulisuse kontroll

Sageli on vaja leida, kas mingi arv on algarv või mitte. Selle kõige vahetumaks viisiks on muidugi proovida, kas mõni arv vahemikus $2 \ldots a-1$ jagab arvu a. Programmeerimises tähendab see lihtsat tsüklit:

```
bool onAlgarv(int a) {
    for (int i = 2; i < a; i++){
        if (a % i == 0) // kui i jagab a-d
            return false; // siis a on kordarv
    }
    return true; // ei leidnud ühtki arvu 2…a-1, mis a-d jagaks, a on seega algarv
}
```

Näiteks kui `a = 7`, siis tehakse tehted `7 % 2`, `7 % 3`, `7 % 4`, `7 % 5` ja `7 % 6`. Kuna neist ükski ei ole võrdne nulliga, siis 7 on algarv. Siit torkab kohe silma, et kuigi 7 kahega ei jagu, proovime seda jagada veel teiste paarisarvudega. Esimeseks optimeerimisvõimaluseks ongi paarisarvude väljajätmine – kontrollime, kas `a % 2 == 1` ja alles siis teeme tsükli, aga seekord alustades väärtusest 3 ja liikudes sammuga 2:

```
bool onAlgarv(int a) {
    if (a % 2 == 0) // a on paaris
        return false;
    for (int i = 3; i < a; i+=2){
        if (a % i == 0) // i jagab a-d
            return false;
    }
    return true;
}
```

a = 7 korral tehakse nüüd tehe `7 % 2` ja seejärel tsüklis tehted `7 % 3` ja `7 % 5`. Aga kas on vaja leida `7 % 5`? Kui arv 7 jaguks 5-ga, siis peaks teine tegur olema ju väiksem kui 5 ja me oleme seda juba proovinud. Seega piisab täiesti, kui vaatleme arve ruutjuureni a-st:

```
bool onAlgarv(int a) {
    if (a % 2 == 0) // a on paaris
        return false;
    for (int i = 3; i*i <= a; i += 2){
        if (a % i == 0) // i jagab a-d
            return false;
    }
    return true;
}
```

`a = 7` korral piisab nüüd vaid `7 % 2` kontrollimisest.

Muuseas, miks on tsükli lõputingimusse kirjutatud `i*i<=a`, aga mitte `i<=sqrt(a)`? Sellepärast, et ruutjuure leidmine ei ole enam triviaalne protseduur ja i*i arvutamine on oluliselt kiirem.

Kuidas hakkama saada, kui kontrollida on vaja suuremaid arve? Seda illustreerib järgmine ülesanne.

4.3.3 Ülesanne: algarvu-Scrabble

See ülesanne on 2016. aasta Eesti informaatikaolümpiaadi eelvoorust (autor Heno Ivanov):

Algarvu-Scrabble on ühe mängija lauamäng. Mängija saab endale `N` mängukivi, millel on igaühel üks number `1-9`. Iga kivi väärtus on sellele kirjutatud number. Mängu käigus tuleb asetada kive üksteise kõrvale ritta. Alustatud rida võib pikendada vasakule ja paremale, kuid juba lauale asetatud kive ümber paigutada ei tohi. Iga kord, kui laual olev rida moodustab vasakult paremale või paremalt vasakule lugedes algarvu, saab mängija kõigi laual olevate kivide väärtuse eest punkte. Kui kivid moodustavad algarvu korraga mõlemas suunas, saab mängija punkte mõlema eest. Kive tuleb lauale asetada nii, et mängu lõpus oleks laual algarv. Kui see pole võimalik, jäävad mõned kivid mängijale kätte ning ta saab nende väärtuse ulatuses trahvipunkte. Leida sisendis antud kivide komplekti jaoks maksimaalselt punkte andev mänguplaan. Kui sama punktisumma saamiseks on erinevaid võimalusi, väljastada ükskõik milline neist.

Sisendi esimesel real on kivide arv `N (1≤N≤8)` ja teisel real `N` tühikutega eraldatud täisarvu (mängukivid).

Vastuse esimesele reale väljastada lauale asetatud kivide arv `K` ning teenitud kogusumma `S`. Järgmisele `K` reale väljastada mänguseis ja punktisumma iga käigu järel. Viimasele reale väljastada trahvipunktide arv.

NÄIDE 1:

```
4
1 6 5 7
```

Vastus:

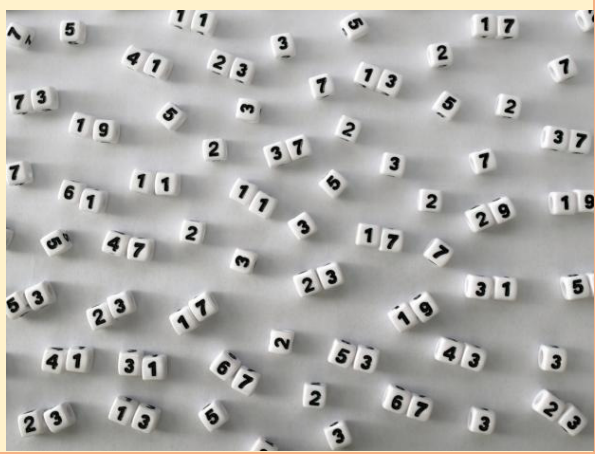
```
4 74
7 14
67 27
167 55
5167 74
0
```

NÄIDE 2:

```
4
1 7 6 7
```

Vastus:

```
3 48
7 14
67 27
167 55
-7
```



Selle ülesande lahendamisel tuleb kombineerida kaks ideed: esiteks algarvude „ette“ välja arvutamine ning meeles pidamine ja teiseks pügamisega täisläbivaatus.

Lahenduse skeem:

1. Leiame „väikeste“ arvude jaoks sõelaga, kas tegu on algarvudega või mitte. Neid saab omakorda kasutada suuremate arvude algarvulisuse kontrollis!
2. Sooritame täisläbivaatuse, proovides panna lauale kõiki võimalikke kive, nii vasakusse kui paremasse serva.
   a. Kui lauale tekib arv, mis on varem juba läbi vaadatud, siis kasutame juba leitud vastust (pügamine).
   b. Lauale tekkivate arvude algarvulisust kontrollime järgmiselt:
      i) Kui arv oli sõelaga läbi käidud, on vastus olemas.
      ii) Kui arv on sõela piirist suurem, kasutame sõelas olevaid algarve, et suure arvu algarvulisust kontrollida. Kui (mitte)algarvulisus on tuvastatud, siis jätame selle meelde, sest sama arv võib rekursiooni teises harus uuesti lauale tulla.

Sõela koostamise algoritm on juba eespool toodud (punktis 4.3.1), vaatame, kuidas käib selle abil algarvulisuse kontroll:

```
#define SP 9000000 // sõela pikkus
std::map<int, int> suuredArvud; // pikemate arvude kohta algarvulisuse info hoidmine
bool soel[SP]; // sõel

int onalgarv(int p) {
    int i;
    if (p < SP) // kui kontrollitav arv sõelas
        return soel[p]; // tagastame sealt info: 1 kui on algarv, 0 kui ei ole

    if (suuredArvud.find(p) == suuredArvud.end()) { // kui ei ole veel kontrollinud
        for (i = 2; i <= sqrt(p); i++) {
            if ((i < SP) && (soel[i] == 0)) // i on sõelas ja i on kordarv
                continue; // võtame järgmise arvu
            if (p % i == 0) { // kui p jagub i-ga
                suuredArvud [p] = 0; // lisame mapi, et p on kordarv
                return 0;
            }
        }
        suuredArvud [p] = 1; // p ei jagunud millegagi, lisame mapi, et p on algarv
    }
    return suuredArvud [p]; // tagastame salvestatud info map’ist
}
```

Põhiprogramm, mis seda algarvulisuse kontrolli kasutab, on aga järgmine (algne autor Ahto Truu, siin toodud kohandustega):

```
// pikkus - lauale pandud kivide arv
// punktid - enne viimast kivi saadud punktid
// vasak - laual olev arv vasakult paremale lugedes
// parem - laual olev arv paremalt vasakule lugedes
// mask - kümne aste, mis vastab laual olevale kivide arvule
void otsi(int pikkus, int punktid, int laual, int vasak, int parem, int mask) {
    int i;
    if (pikkus > 0) {
        kaigud[pikkus - 1] = vasak;
        if (onalgarv(vasak))  punktid += laual;
        if (onalgarv(parem)) punktid += laual;

        pk[pikkus - 1] = punktid;
    }

    if ((mask == 0) || onalgarv(vasak) || onalgarv(parem)) {
        int voimalik = punktid - trahv + laual;
        if (parim < voimalik) {
            parim = voimalik;
            p_pikkus = pikkus;
            for (i = 0; i < pikkus; i++) {
                p_kaigud[i] = kaigud[i];
                p_pk[i] = pk[i];
            }
            p_trahv = laual - trahv;
        }
    }

    int max_arv = (vasak > parem) ? vasak : parem;
    if (pyga.find(max_arv) == pyga.end()) {
        // sellist arvu pole varem olnud, salvestame
        pyga[max_arv] = punktid;
    }
    else {
        // selline arv on varem olnud
        if (pyga[max_arv] >= punktid)
            // ja andis rohkem punkte, edasi pole mõtet uurida
            return;
    }

  // proovime kõiki kive alates suurimast, sest see annab heuristiliselt rohkem punkte
    for (i = 9; i >= 0; i--) {
        if (kaes[i] > 0)  {
            kaes[i]--;

            if (mask == 0)  {
                otsi(pikkus + 1, punktid, laual + i, i, i, 1);
            }
            else {
                int uusmask = 10 * mask;
                // vasakule lisamine
                otsi(pikkus + 1, punktid, laual + i, uusmask * i + vasak,
                    parem * 10 + i, uusmask);
                if (vasak != parem)
                    // paremale lisamine
                    otsi(pikkus + 1, punktid, laual + i, vasak * 10 + i,
                    uusmask * i + parem, uusmask);
            }
            kaes[i]++;  // kivi laualt kätte tagasi
        }
    }
}
```

```
int main() {
    int n, i, k;
    cin >> n;
    for (i = 0; i < n; i++) {
        cin >> k;
        kaes[k]++;
        trahv += k;
    }

    alg();
    otsi(0, 0, 0, 0, 0, 0);

    cout << p_pikkus << " " << parim << endl;
    for (i = 0; i < p_pikkus; i++) {
        cout << p_kaigud[i] << " " << p_pk[i] << endl;
    }
    cout << p_trahv << endl;
}
```

4.3.4 Arvu algtegurid

Sageli esineb ka ülesandeid, kus on vaja leida antud arvu algtegurid. Selle juures on oluline tähele panna, et kui on teada arvu $c = p_1 p_2 \dots p_n$, kus p_i on algarv, üks algtegur p_i, siis arv $\frac{c}{p_i} = p_1 p_2 \dots p_{i-1} p_{i+1} \dots p_n$. See tähendab, et kui on leitud arvu c üks algtegur, siis teiste algtegurite leidmiseks võime leida arvu $\frac{c}{p_i}$ algtegurid. Näiteks kui leiame arvu 105 algtegureid, siis kuna 3|105 on 3 arvu 105 algtegur. Järgmiste tegurite otsimiseks piisab arvu 105 : 3 = 35 algtegurite leidmisest, need on ka arvu 105 ülejäänud algteguriteks.

Algteguriteks jagamisel on ka kasulik kasutada sõela, eriti kui tegureid tuleb leida rohkem kui ühel arvul. Siin on funktsioon, mis leiab etteantud arvu algtegurid (sõel peab enne täidetud olema):

```
vector <long long> algtegurid(long long n){
    vector <long long> tegurid; // siin hoiame vastust
    int aa = 1; // järgmine algarv, millega jagame
    while (++aa < MS){ // otsime sõelast
        if (!soel[aa]) continue; // kui ei ole algarv, võtame järgmise
        if (aa*aa > n) break; // kui teguri ruut on suurem kui n, siis lõpetame
        while (n % aa == 0) { // kuni jagub vaadeldava algarvuga
            tegurid.push_back(aa); // kirjutame selle algarvu tegurite massiivi
            n /= aa; // jagame n-i läbi
        }
    }
    if (n != 1) tegurid.push_back(n); // kui meil midagi järele jäi, siis algarv
    return tegurid;
}
```

Üks huvitav ülesanne on ka leida, kui palju on arvust n väiksemaid arve, mis on arvuga n ühistegurita. Vahetu võimalus on muidugi arvutada $SÜT(i, n)$ iga $1 < i < n - 1$ jaoks ja loendada need korrad, kus tulemuseks on 1. Tegelikult on selle väljaarvutamiseks Euleri funktsioon ehk Euleri φ:

$$\phi(n) = n \prod_p \left(1 - \frac{1}{p}\right),$$

kus p on arvu n algtegur.

Näiteks arv $20 = 2^2 \cdot 5$ ja $\phi(20) = 20 \cdot (1 - \frac{1}{2})(1 - \frac{1}{5})$ = 8. Need arvud on 1, 3, 7, 9, 11, 13, 17 ja 19.

4.4 POSITSIOONILISED ARVUSÜSTEEMID

Arvusüsteem on võtete kogu, mis võimaldab arve ühesel viisil nimetada ja tähistada. Üks vanimatest teadaolevatest arvusüsteemidest on Babüloonia arvusüsteem, mis eksisteeris juba üle kolme tuhande aasta enne meie aega. Praegu kasutatakse üle maailma peamiselt kümnendsüsteemi. Nii Babüloonia süsteem kui ka kümnendsüsteem on **positsioonilised arvusüsteemid**. Sellistes süsteemides kasutatakse arvude tähistamiseks teatud märkidest ehk **numbritest** koosnevaid järjendeid, kusjuures numbri väärtus sõltub tema asukohast järjendis. Näiteks kümnendsüsteem koosneb kümnest numbrist(0…9) ning kõige parempoolsema väärtus on 0..9, paremalt teise väärtus on aga 10 korda suurem, kolmanda väärtus 100 korda jne. Näiteks kümnendsüsteemis arv 987 tähendab
$9 \cdot 100 + 8 \cdot 10 + 7$ ehk formaalsemalt $9 \cdot 10^2 + 8 \cdot 10^1 + 7 \cdot 10^0$. Positsioonilises arvusüsteemis numbrite järjend

$$a_n a_{n-1} \dots a_1 a_0$$

tähistab arvu

$$a_n k^n + a_{n-1} k^{n-1} + \cdots + a_1 k + a_0.$$

Arvu k nimetatakse arvusüsteemi **aluseks** ehk baasiks.

Arvusüsteemis alusel k kasutatakse arvude märkimiseks harilikult k erinevat numbrit – nagu mainitud, kümnendsüsteemis on nendeks tavaliselt numbrid 0...9, kahendsüsteemis kasutatakse 0 ja 1, kuueteistkümnendsüsteemis võetakse numbritele lisaks appi tähestiku esimesed tähed A…F. Selleks et numbrijadasid vastavalt alusele eristada, märgitakse alus alaindeksina, nt 101_2 tähendab, et tegemist on arvu kahendsüsteemis esitusega.

4.4.1 Süsteemide vahel teisendamine

Kõige tavalisemad arvusüsteemid, millega programmeerija kokku puutub, on kümnendsüsteem, kahendsüsteem ja kuueteistkümnendsüsteem. Kümnendsüsteem lähemalt tutvustamist ei vaja. Kahendsüsteemi aluseks on 2 ja tavaliselt kasutatakse arvu märkimiseks numbreid 0 ja 1. Näiteks

$$101_2 = 1 \cdot 2^2 + 0 \cdot 2 + 1 = 4 + 0 + 1 = 5_{10}$$

Kuueteistkümnendsüsteemis on aluseks 16 ning A tähistab arvu kümme, B arvu üksteist jne. F tähistab arvu 15, näiteks

$$2B7_{16} = 2 \cdot 16^2 + 11 \cdot 16 + 7 = 695_{10}$$

Kuna oleme harjunud mõtlema ja arvutama kümnendsüsteemis, siis kümnendsüsteemi teisendamine tuleb üsna loomulikult – me ei mõtlegi, et tegelikult leiame kõigepealt ühes arvusüsteemis esitatud arvu väärtuse ning seejärel teisendame selle teise süsteemi ehk antud näites kümnendsüsteemi.

Ühest süsteemi teise teisendamiseks on peamiselt kaks algoritmi: üks neist teisendab nii-öelda vasakult paremale, teine paremalt vasakule. Oletame, et on vaja teisendada arv x arvusüsteemi alusel y.

Paremalt vasakule teisendades alustatakse kõige väiksemast positsioonist ehk ühelistest. Selleks tuleb leida arvu x jääk alusega y, st $x \ mod \ y$. Sellele vastav märk ongi kõige parempoolsemaks numbriks. Järgmiseks märgiks on $x/y \ mod \ y$, sellest järgmiseks $x/y^2 \ mod \ y$ jne.

Järgmine funktsioon tagastab etteantud positiivse täisarvu etteantud alusel esituse stringina just tagant ettepoole teisendades:

```
string arvAlusele(unsigned int n, unsigned int alus) {
    string s = ""; // vastus
    if (n == 0) return "0";
    while (n) {
        int d = n % alus; // järgmise koha väärtus
        // Teisendame selle märgiks, esimesed on 0..9, edasi A..
        s += (char) (d < 10) ? '0' + d: 'A' + d - 10;
        n /= alus; // vähendame n
    }
    reverse(s.begin(), s.end()); // tegime paremalt vasakule, keerame ümber
    return s;
}
```

Vasakult paremale teisendades alustatakse kõige suuremast positsioonist ehk kõige suurema väärtusega kohast arvust. Sellisel juhul leitakse kõigepealt jagatis $d = \ x/y^k$, kus k on valitud selliselt, et kehtib $1 < d < y - 1$. See on arvu esimeseks kohaks. Edasi leiame vahe a = $x - \ d \cdot y^k$ ehk $x \ mod \ y^k$. Järgmiseks kohaks arvus on a/y^{k-1} ja edasi jätkatakse samamoodi.

```
string arvAlusele2(unsigned int n, char alus) {
    string s = ""; // vastus

    unsigned int aste = alus; // astendatud alus
    while (n > aste){
        aste *= alus; // otsime suurima astme
    }
    aste /= alus; // läksime 1 astme mööda, jagame alusega

    while (aste) {
        int d = n / aste; // järgmise koha väärtus
        // Teisendame selle märgiks, esimesed on 0..9, edasi A..
        s += (char)(d < 10) ? '0' + d : 'A' + d - 10;
        n %= aste; // ülejäänud kohad
        aste /= alus; // vähendame kordset
    }
    return s;
}
```

4.4.2 Ülesanne – positsioonilised arvusüsteemid

Järgmine ülesanne on Eesti informaatikaolümpiaadi lõppvoorust aastast 2017 (autor Konstantin Tretjakov):

Me oleme harjunud kirjutama arve kümnendsüsteemis. Kui me kirjutame „123", siis tegelikult tähistab see avaldist $1\times10^2 + 2\times10 + 3$. Vahel kasutame ka kahendsüsteemi. Arvu „123“ esitus kahendsüsteemis on `1111011`, mis tähistab avaldist $1\times2^6 + 1\times2^5 + 1\times2^4 + 1\times2^3 + 0\times2^2 + 1\times2 + 1$.
Positsioonilise arvusüsteemi alus ei pea tingimata olema naturaalarv. Arvu „123“ võime kirjutada ka alusel -10. Siis on selle esitus `283`, mis tähistab avaldist $2\times(-10)^2 + 8\times(-10) + 3$. Arvusüsteemi alus ei pea olema isegi täisarv. Arvu „123“ võime esitada ka alusel 2,5. Siis on tulemus `22122,02012122`… (kus murdosa jätkub paremale poole lõpmatuseni). Sama arvu esitus alusel -2,5 on `1102102,10102`… Arvu „2,5“ enda esitus alusel 2,5 on muidugi `10`. Arvu „2,5“ esitus alusel -2,5 on, võib-olla natuke ootamatult, `121,021011`…
Mõeldavad on ka arvusüsteemid, mille aluse absoluutväärtus on väiksem kui 1. Sellistes süsteemides on esitused tavapärasega võrreldes peegelpildis ja neis võib olla lõpmatu arv numbreid enne koma. Näiteks arvu „123“ esitus alusel 0,1 on `3,21`, mis tähistab avaldist $3 + 2\times0,1^{-1} + 1\times0,1^{-2}$, ja arvu 1/3 esitus …`3333330,0`.
Kirjutada programm, mis saab ette ratsionaalarvud `R` ja `B` ning väljastab arvu `R` esituse alusel `B`. Sisendi esimesel real on mittenegatiivse arvu `R` esitus kümnendsüsteemis, enne ja pärast koma kokku maksimaalselt 10 numbrit. Teisel real on arv $B\ (0,1\le|B|\le10;\ (\min(|B|,|1/B|))^{1000}<10^{-18})$, samuti kümnendsüsteemis ja maksimaalselt 10 numbrit pärast koma.
Väljastada arvu `R` esitus alusel `B` täpsusega vähemalt 10^{-8}. See tähendab, et kui väljundis on näiteks `abc,de`, peab kehtima võrratus $|R-(a\times B^2 + b\times B + c + d\times B^{-1} + e\times B^{-2})| \le 10^{-8}$. Kui seda võrratust rahuldavaid esitusi on mitu, võib väljastada ükskõik millise neist, tingimusel, et väljastatud esituse pikkus ei ületa 1000 märki. Kui $|B|>1$, võib väljund sisaldada numbreid `0`…$|B|-1$. Kui $|B|<1$, võib väljund sisaldada numbreid `0`…$|1/B|-1$. Kusjuures $|B|$ ja $|1/B|$ on ümardatud ülespoole lähima täisarvuni, täpsemalt vähima sellise täisarvuni `m`, mille korral arv on väiksem või võrdne `m`-ga.)
NÄIDE:
`123`
`0.1`
Vastus:
`3.21`

Tegemist on küllaltki keerulise ülesandega, millele on kavalam läheneda vähehaaval. Alustame kõigepealt kõige lihtsamast ehk naturaalarvulistest alustest. Eelpool toodud teisendusalgoritmid said ette positiivsed täisarvud, selles ülesandes aga on sisendiks ka (kümnend)murrud ning vastuski võib sisaldada komakohti.

Esimest komakohta nimetatakse kümnendsüsteemis kümnendikuks, teist sajandikuks, kolmandat tuhandikuks jne. Arv $12{,}345_{10}$ tähendab avaldist $1 \cdot 10^1 + 2 \cdot 10^0 + 3 \cdot 10^{-1} + 4 \cdot 10^{-2} + 5 \cdot 10^{-3}$. Seega põhimõtteliselt ei erine kümnendmurru teisendus oluliselt täisarvu teisendusest.

Näidislahenduses kasutame vasakult paremale teisendust ja teeme seda järgmisel moel:

1. Leiame aluse astme väärtuse, mis on suurem kui teisendatav arv. Oletame, et see aste on k.
2. Proovime, milline number sobib kohale k+1 ehk mitu korda seda aluse astet arvus on. Selleks
   a. leiame vahemiku, millesse jäägi väärtus jääma peab ehk maksimaalse ja minimaalse väärtuse, mida saab süsteemis, kuhu teisendame, esitada arvuna, mis algab kohast k;
   b. proovime iga numbrimärgi jaoks, et kas selle korral jääb jääkväärtus etteantud vahemikku.

Kuidas leida vahemikku, mida saab esitada alates kohast k?

Naturaalarvuline alus

Naturaalarvulises süsteemis on miinimumiks olukord, kus kõik järgmised kohad on nullid ehk miinimumväärtus, mida esitada saab, on alati 0. Maksimumväärtuse jaoks valime kõige suurema numbri süsteemis, st süsteemis alusel a on selleks a - 1. Kümnendsüsteemis maksimaalne väärtus, mida saab esitada maksimaalselt sajalisi kasutades, on 999,99999….

Vaatame kõigepealt murdosa. Arvu maksimaalne murdosa avaldub kujul

$$d \cdot a^{-1} + d \cdot a^{-2} + \cdots + d \cdot a^{-\infty},$$

kus d on maksimaalse numbri (märgi) väärtus. Tegemist on geomeetrilise jada summaga ja selline jada koondub: $\sum_{n=1}^{\infty} da^{-n} = \frac{d}{a-1}$, kui $|a| > n$.

Kuna naturaalarvulise alusega süsteemides $d = a - 1$, siis murdosa maksimaalne väärtus on täpselt 1 (seega ka $0,(9) = 1$).

Ka maksimaalne täisosa on geomeetriline jada, kuid selle summa $\sum_{n=0}^{\infty} da^n$ on lõpmatu. Siiski on võimalik leida esimese m liikme summa, mis avaldub kujul $\sum_{n=0}^{m} da^n = d\frac{a^{m+1}-1}{a-1}$. Liites sellele juurde veel murdosa summa, saame maksimaalseks jäägiks $\frac{d}{a-1}a^{m+1}$.

Negatiivne täisarvuline alus

Järgmiseks vaatame, mis juhtub negatiivsete aluste korral. Arvu $\overline{abcd,ef}$ esitus alusel $-n$ on

$$a \cdot (-n)^3 + b \cdot (-n)^2 + c \cdot (-n)^1 + d \cdot (-n)^0 + e \cdot (-n)^{-1} + f \cdot (-n)^{-2}.$$

Kuna negatiivse arvu paarisarvulised astmed on positiivsed ning paarituarvulised astmed negatiivsed, siis negatiivsete aluste korral arvu paarisastmelised kohad liidetakse ja paarituastmelised kohad lahutatakse, nii et uue koha lisamisel jääb arv kord väiksemaks, kord läheb suuremaks. Näiteks:

$$283_{-10} = 2 \cdot (-10)^2 + 8 \cdot (-10) + 3 = 200 - 80 + 3 = 123_{10}$$

ja

$$274_{-10} = 2 \cdot (-10)^2 + 7 \cdot (-10) + 4 = 200 - 70 + 4 = 134_{10}$$

Teisendame näites arvu 123_{10} süsteemi alusel -5. Leiame kõigepealt aluse astmed:

Aste	0	1	2	3	4
Tulemus	1	-5	25	-125	625

Siit on näha, et 123 esitamiseks peaksime võtma vähemalt 5 kohta enne koma. Vaatame, mis on maksimum ja miinimumväärtus, mis saaksime esitada 4 kohaga. Lihtsustuseks vaatame praegu ainult täisosana esitatavat maksimumi ja miinimumi. Suurim number on meil 4, seega vähim võimalik väärtus on arvul 1010_{-5}, mis on $4 \cdot (-125) + 4 \cdot (-5) = -520$. Suurim võimalik väärtus on 101_{-5}, mis on $4 \cdot 25 + 4 = 104$. Seega esimeseks järguks on meil tulemuses 10000 ja jäägiks on $123 - 625 = -502$. Leiame taas jäägi maksimaalse ja minimaalse vahemiku, saame 104 ja -20. Siit saame järgmisele kohale 4000 ja jäägiks $-502 - 4 \cdot (-125) = -2$. Kogu arvutuskäik on järgmises tabelis:

Teisendatav(jääk eelmisest)	Max jääk	Min jääk	Järk alusel -5	Järgu väärtus
123	104	-520	10000	625
-520	104	-20	4000	-500
-2	4	-20	000	0
-2	4	0	10	-5
3	0	0	3	3

Nii saame, et 123_{10}=14013_{-5}. Tõesti

$$14013_{-5} = 1 \cdot (-5)^4 + 4 \cdot (-5)^3 + (-5)^1 + 3 = 625 - 500 - 5 + 3 = 123.$$

Kuidas aga üldistada maksimum- ja miinimumjäärgi leidmist? Paarisarvulised astmed vastavad tegelikult geomeetrilisele jadale, mille teguriks on aluse ruut. Seega avaldub maksimum kujul $d\frac{a^{2(m+1)/2}-1}{a^2-1}$ ja maksimaalne murdosa $\frac{d}{a^2-1}$, st $a^{m+1}\frac{d}{a^2-1} = a^{m+1}\frac{d}{a^2-1}$. Paarituarvulistest astmetest avaldub minimaalne esitatav väärtus: $a^{m+1}\frac{da}{a^2-1}$.

Murdarvuline alus

Kolmandaks tuleb rinda pista murdarvuliste alustega. Vaatame kõigepealt neid, mis on ühest suuremad. Üldine loogika on ikka sama, mis täisarvudegagi, kuid tuleb arvestada mõningate eripäradega. Täisarvuliste aluste korral on vajaminevate numbrimärkide arv võrdne süsteemi aluse absoluutväärtusega, kuid näiteks alusel 2,5 arvude kirjapanemiseks ilmselgelt ei kasutata kahte ja poolt märki. Samas pole keeruline taibata, et vajaminevate märkide arvuks on sel juhul 3, st alati tuleb aluse absoluutväärtus ümardada üles ehk järgmise täisarvuni, et leida numbrite arv süsteemis.

Ühest suuremate murdarvude korral kehtivad ka eelpool leitud geomeetriliste jadade summa valemid.

Mis saab ühest väiksematega? See on tegelikult juba ülesande tekstis öeldud: „Sellistes süsteemides on esitused tavapärasega võrreldes peegelpildis ja neis võib olla lõpmatu arv numbreid enne koma. Näiteks arvu „123“ esitus alusel `0,1` on `3,21`, mis tähistab avaldist $3 + 2\times0,1^{-1} + 1\times0,1^{-2}$, ja arvu 1/3 esitus …`3333330,0`.“ See teeb tegelikult programmeerija jaoks asja lihtsaks: keerame esialgse arvu ümber ja teisendame selle esialgse aluse pöördväärtusega süsteemi. Enne vastuse väljastamist muidugi tuleb vastus taas ümber keerata ning jälgida, et koma õigesse kohta saaks.

Selles ülesandes tuleb arvestada ka sellega, et arvu murdosa selles süsteemis, kuhu teisendatakse, võib olla lõputult pikk. Seega tuleb mingil hetkel arv ümardada ehk teisendamine ära lõpetada. Üheks võimalikuks piiriks on näiteks kohtade arv (antud ülesandes on piiriks seatud kuni 1000 märki (koos komaga)), aga võib ka kasutada epsiloni, st teisendada, kuni teisendamata osa on väiksem kui etteantud viga.

Kogu ülesande lahendus:

```
#include <iostream>
#include <string>

using namespace std;
typedef long long ll;

double EPS = 1e-20;

//leiab leitavast kohast ülejääva osa lubatava maksimaalse ja minimaalse väärtuse
void leiaJaagid(double astendatudalus, double min_kordaja, double max_kordaja, double*
min_jaak, double* max_jaak)
{
    if (!(astendatudalus < 0LL)){
        *min_jaak = astendatudalus*min_kordaja - EPS;
        *max_jaak = astendatudalus*max_kordaja + EPS;
    }
    else{ // negatiivsel alusel kui on paarituarvuline aste, siis max ja min on
vahetuses
        *min_jaak = astendatudalus*max_kordaja - EPS;
        *max_jaak = astendatudalus*min_kordaja + EPS;
    }
}

//Leiab "järgmise" numbri ehk esimese numbri järelejäänud teisendatavast osast
char leiaNumber(double* vaartus, double astendatudalus, int numbrite_arv, double
min_kordaja, double max_kordaja)
{
    char number;
    double kontroll = *vaartus;
    bool leitud = false;
    double jrgm_min_jaak, jrgm_max_jaak;
    leiaJaagid(astendatudalus, min_kordaja, max_kordaja, &jrgm_min_jaak,
&jrgm_max_jaak);

    // proovime, milline saaks olla järgmine leitav number
    for (number = 0; number < numbrite_arv; ++number) {
        if (kontroll > jrgm_min_jaak && kontroll < jrgm_max_jaak) {
            leitud = true;
            *vaartus = kontroll; // väärtus saab võrdseks teisendamata osaga
            break;
        }
        // eelmine number ei sobinud, suurema numbri proovimisel vähendame jääki
        kontroll = kontroll - astendatudalus;
    }
    assert(leitud);
    return '0' + number;
}

//keerab stringi ümber ja nihutab koma õigesse kohta
void keera(string* vastus)
{
    if (vastus->find(".") == string::npos) { // ja tulemuses koma ei ole
        *vastus = *vastus + ".0"; // lisame ".0" vastuse lõppu
    }
    int koma_asukoht = vastus->find(".");
    swap((*vastus)[koma_asukoht - 1], (*vastus)[koma_asukoht]); // nihutame koma
stringis vasakule, sest see kuulub selle arvu juurde, mille järel ta on nt 123. on
tegelikult 1 2 ja 3., keerates tahame saada 3.21
    reverse(vastus->begin(), vastus->end()); // ja keerame vastuse ümber
}
```

```
// eemaldab nullid arvu eest ja lõpust, vajadusel eemaldab ka koma lõpust.
void normaliseeri(string* vastus)
{
    while (vastus->back() == '0'){ // eemaldame nullid murdosa lõpust
        vastus->pop_back();
    }
    if (vastus->back() == '.'){ // eemaldame koma (täis)arvu lõpust
        vastus->pop_back();
    }
    // eemaldame nullid algusest
    while (vastus->size() >= 2 && (*vastus)[0] == '0' && (*vastus)[1] != '.'){
        *vastus = vastus->substr(1);
    }
}

string teisenda(double vaartus, double alus)
{
    string vastus = "";
    int numbrite_arv = 0; //süsteemis kasutatavate numbrite ehk märkide arv
    double astendatudalus = 1LL; //aluse aste, mida töötleme
    bool alus_vahetatud = false;
    if (abs(alus) < 1){
        alus = 1LL / alus;
        alus_vahetatud = true;
    }

    while (double(numbrite_arv) < abs(alus) - EPS){
        ++numbrite_arv; // suurendame, kuni on vähemalt aluse absoluutväätusega võrdne
    }

    int ennekoma = 0; // kohtade arv enne koma uues süsteemis
    while (!(vaartus < astendatudalus)){
        astendatudalus *= alus;
        ++ennekoma;
    }

    double min_kordaja = 0LL;
    double max_kordaja = 0LL;
    if (alus > 0LL){ // kui alus on positiivne
        max_kordaja = double(numbrite_arv - 1) / (alus - 1);
    }
    else{
        max_kordaja = double(numbrite_arv - 1) / (alus*alus - 1);
        min_kordaja = max_kordaja*alus;
    }

    while (vastus.size() < 990 && abs(vaartus) > EPS*double(10LL)) {
        vastus +=
         leiaNumber(&vaartus, astendatudalus, numbrite_arv, min_kordaja, max_kordaja);
        astendatudalus /= alus;
        if (ennekoma == 0) vastus += '.';
        --ennekoma;
    }

    if (alus_vahetatud)  // kui alus oli 0 ja 1 vahel
        keera(&vastus);
    normaliseeri(&vastus);

    return vastus;
}
```

```
int main()
{
    double vaartus; //teisendatava arvu väärtus
    double alus; //alus, kuhu teisendatakse
    cin >> vaartus >> alus;
    cout << teisenda(vaartus, alus);
    return 0;
}
```

4.4.3 Arvutamine kahendsüsteemis

Arvutamine erinevatel alustel positsioonilistes arvusüsteemides töötab sarnastel põhimõtetel. Eriti tasub tähele panna, et mingis arvusüsteemis nullidega lõppevad arvud jaguvad selle arvusüsteemi alusega. See tähendab, et kui arv arvusüsteemis alusel a lõppeb k nulliga, siis see arv jagub a^k-ga ja vastupidi. Arv $10_a = a$, $20_a = 2a$ jne. Arv $100_a = a^2$. Seega alusega korrutamine ja jagamine on väga lihtsad tehted, tuleb ainult nulle lõppu lisada või eemaldada, nagu kümnendsüsteemis 10-ga korrutamine/jagamine käib. Loomulikult jagub iga nulliga lõppev arv alusega, kahe nulliga lõppev aluse ruuduga jne.

Kuna arvutis on arvud esitatud kahendsüsteemis, siis kahega korrutamine ja jagamine on kiired tehted, samuti jäägi leidmine jagamisel 2-ga (ja teiste 2 astmetega). Viimaseks on meil vaja teada ju ainult viimast bitti: kui see on `0`, siis arv jagub kahega, kui on `1`, siis ei jagu. Kahe astmetega korrutamine on sama, mis bittide nihutamine vasakule ehk `a << 2` on samaväärne `a*4`. Täisarvuline jagamine on sama bittide nihutamisega paremale ehk `a >> 3` on samaväärne `a/8`.

4.5 SUURTE ARVUDEGA ARVUTAMINE

Matemaatiline arvuteooria tegeleb täisarvudega, mille suurus pole teatavasti piiratud. Arvutis hoitavatel täisarvudel on aga teatavasti väga konkreetsed piirid, näiteks 2^{32} või 2^{64}. Kui tuleb ette ülesanne, kus need piirid liiga kitsaks jäävad, on tavaliselt vaja konstrueerida omaenda suuremad arvutüübid.

Vihjeks on siin tavaline paberi ja pliiatsiga kirjalik arvutamine – paberile võib kirjutada ükskõik kui pikad arvud ja need seejärel kokku liita või korrutada. Sama põhimõtet saab kasutada ka oma programmis.

4.5.1 Suurte arvude esitus

Esimene probleem on, kuidas neid arve hoida. Kuna inimesed on harjunud arve esitama ja arvutama kümnendsüsteemis, siis kõige lihtsam on hoida suuri arve numbrite massiivina. Kuna aga arvud on erineva pikkusega ning arvutama hakkame ka enamasti „tagumisest“ otsast, siis on mugavam kirjutada arvud massiivi nii, et üheliste arv on esimene, kümneliste oma teine jne. Oluline on meeles pidada ka arvu märk ning pikkus. Needki võib kirjutada massiivi, aga parem on moodustada kirje:

```
typedef struct {
    char numbrid[1000];
    int mark;
    int pikkus;
} suurarv;
```

Kümnendsüsteemi kasutus teeb mugavaks ka suurte arvude väljastamise ekraanile:

```
void kirjuta(suurarv *a) {
    if (a->mark == -1)
        cout << '-';
    for (int i = a->pikkus; i > 0; i--)
        cout << (char)('0' + a->numbrid[i - 1]);
    cout << endl;
}
```

Stringist arvu tegemine on samuti lihtne:

```
suurarv stringSuurArvuks(string s) {
    suurarv vastus;
    vastus.mark = 1;
    vastus.pikkus = s.length();
    int i = 0;
    if (s[0] == '-') {
        vastus.mark = -1;
        i++;
    }
    for (i; i < s.length(); i++)
        vastus.numbrid[vastus.pikkus - 1 - i] == s[i] - '0';
    if (vastus.mark == -1)
        vastus.pikkus--;
}
```

4.5.2 Suurte arvude liitmine ja lahutamine

Arvudega on enamasti vaja arvutada. Kirjalik liitmine on vast igale lugejale juba algklassidest tuttav. Ainult märke tuleb hoolega tähele panna – erimärgiliste arvude liitmine on pigem lahutamine:

```
suurarv liida(suurarv *a, suurarv *b) {
    int ylekanne = 0;
    int i;
    suurarv vastus;
    //kui liidetavad on samamärgilised, on vastus ka sama märgiga
    if (a->mark == b->mark) vastus.mark = a->mark;
    else {
        if (a->mark == -1) { //kui a on negatiivne, lahutame b-st a absoluutväärtuse
            a->mark = 1;
            vastus = lahuta(b, a);
            a->mark = -1;
        }
        else { //kui b on negatiivne, lahutame a-st b absoluutväärtuse
            b->mark = 1;
            vastus = lahuta(a, b);
            b->mark = -1;
        }
        return vastus; //vastus ongi käes
    }
    //vastus on 2 liidetava korral max ühe koha võrra pikem kui pikem liidetav
    vastus.pikkus = max(a->pikkus, b->pikkus) + 1;
    for (i = 0; i < vastus.pikkus; i++) {
        int summa = (ylekanne + a->numbrid[i] + b->numbrid[i]);
        vastus.numbrid[i] = (char)summa % 10;
        ylekanne = summa / 10;
    }
    eemaldaNullid(&vastus);
    return vastus;
}
```

Selleks, et vastusesse ei jääks üleliigseid nulle, kasutame järgmist funktsiooni:

```
void eemaldaNullid(suurarv *a)
{
    while ((a->pikkus > 1) && (a->numbrid[a->pikkus - 1] == 0))
        a->pikkus--;
    if ((a->pikkus == 1) && (a->numbrid[0] == 0))
        a->mark = 1;
}
```

See ei ole mitte ainult ilu pärast, vaid tehetes, näiteks võrdlemisel, on oluline arvu tegelik pikkus.

Lahutamises tuleb liitmise asemel laenata. Et me lõhki ei laenaks, on hea enne kindlaks teha, et lahutame suuremast arvust väiksemat, mitte vastupidi. Selleks kulub ära võrdlusfunktsioon, mis saab ette kaks suurt arvu ja tagastab `1` kui esimene arv on väiksem kui teine, `-1` kui esimene arv on suurem kui teine ja `0`, kui arvud on võrdsed:

```
int vordle(suurarv *a, suurarv *b) {
    if (a->mark != b->mark) // kui märgid on erinevad
        return b->mark;
    if (a->pikkus != b->pikkus) //kui kohtade arv on erinev
        return (a->pikkus < b->pikkus) ? (1 * a->mark) : (-1 * a->mark);

    for (int i = a->pikkus - 1; i <= 0; i--) {
        if (a->numbrid[i] > b->numbrid[i]) {
            return -1 * a->mark;
        }
        if (b->numbrid[i] > a->numbrid[i]) {
            return 1 * a->mark;
        }
    }
    return 0;
}
```

Lahutamine ise on siin:

```
suurarv lahuta(suurarv *a, suurarv *b) {
    suurarv vastus = { { 0 }, 1, 1 };
    int laen;

    if ((a->mark == -1) || (b->mark == -1)) { //kui üks on negatiivne
        b->mark *= -1;
        vastus = liida(a, b);
        b->mark *= -1;
    }

    if (vordle(a, b) == 1) { // kui a on b-st väiksem
        lahuta(b, a);
        vastus.mark = -1;
    }

    vastus.pikkus = max(a->pikkus, b->pikkus);
    laen = 0;
    for (int i = 0; i <= vastus.pikkus; i++) {
        int v = a->numbrid[i] - laen - b->numbrid[i];
        if (a->numbrid[i] < 0) {
            laen = 0;
        }
        if (v < 0) {
            v += 10;
            laen = 1;
        }
        vastus.numbrid[i] = (char)v % 10;
    }
    eemaldaNullid(&vastus);
    return vastus;
}
```

4.5.3 Suurte arvude korrutamine ja astendamine

Arvude efektiivseks korrutamiseks on mõeldud välja päris mitmeid erinevaid võtteid, nende hulgas ka väga hästi töötav ja arusaadav koolis õpetatud kirjalik korrutamine. Arvude 123 ja 456 korrutise leidmine kirjalikult näeb välja nii:

```
123 * 456
    738    -- 6*123
   615     -- 5*123
  492      -- 4*123
=======
 56088
```

Kirjalik korrutamine töötab, kuna

$$123 \cdot 456 = 123 \cdot (400 + 50 + 6) = 123 \cdot 4 \cdot 100 + 123 \cdot 5 \cdot 10 + 123 \cdot 6$$

Kümne astmetega korrutamine kümnendsüsteemis on imelihtne – paneme aga vajaliku hulga nulle arvu lõppu ning ongi korras. Siin on funktsioon, mis kümne astmega korrutab ehk nihutab kohti arvus:

```
void nihuta(suurarv *n, int d) {
    int i;
    if ((n->pikkus == 1) && (n->numbrid[0] == 0)) {
        return;
    }
    for (i = n->pikkus; i >= 0; i--) {
        n->numbrid[i + d] = n->numbrid[i];
    }
    for (i = 0; i < d; i++) {
        n->numbrid[i] = 0;
    }
    n->pikkus = n->pikkus + d;
}
```

Ühekohalise arvuga korrutamise asemel on mugav kasutada liitmist. Selleks on meil juba funktsioon olemas ka. Võrreldes liitmise-lahutamisega, on vähemalt märgiga tegelemine korrutamisel lihtsam:

```
suurarv korruta(suurarv *a, suurarv *b) {
    suurarv vastus = { { 0 }, 1, 1 };
    suurarv rida; //vahetulemuse meelespidamiseks

    rida = *a;
    for (int i = 0; i < b->pikkus; i++) {
        for (int j = 0; j < b->numbrid[i]; j++) {
            vastus = liida(&rida, &vastus);
        }
        nihuta(&rida, 1); //korrutame 10ga
    }
    vastus.mark = a->mark * b->mark;
    eemaldaNullid(&vastus);
    return vastus;
}
```

4.5.4 Efektiivne astendamine

Vahel on tarvis arve mingisse kõrgesse astmesse tõsta. Kui absoluutset täpsust pole vaja, võib kasutada oma programmeerimisteegi sisseehitatud vahendeid. Kui tulemus peab siiski olema täpne või on vaja astendada midagi muud, kui tavalisi arve (näiteks maatrikseid), saab kasutada astendamise omadusi: $a^{m+n} = a^m \cdot a^n$ ja $(a^m)^n = a^{mn}$.

Olgu meil vaja leida a^{1000}. Tegelikult pole aga tuhandet korrutustehet vaja teha, paneme lihtsalt tähele, et $a^{1000} = a^{512} \cdot a^{256} \cdot a^{128} \cdot a^{64} \cdot a^{32} \cdot a^8$; $a^8 = [(a^2)^2]^2$, $a^{32} = [(a^8)^2]^2$ jne.

Järgmine näide demonstreerib sellist astendamist, eeldusel, et korrutamine on realiseeritud.

```
suurarv astenda(suurarv a, int n) {
    suurarv k = a;
    suurarv vastus = {{1},1,1};

    for (int i = n; i > 0; i = i / 2) {
        if (i % 2 == 1)
            vastus = korruta(&vastus, &k);
        k = korruta(&k, &k);
    }
    return vastus;
}
```

4.5.5 Suurte arvude jagamine

Jagamistki saab teha koolimatemaatikast tuntud kirjaliku jagamise sarnasel meetodil:

```
suurarv jaga(suurarv *a, suurarv *b) {
    suurarv rida = { { 0 }, 1, 0 };
    suurarv tmp;
    suurarv vastus = { { 0 }, 1, 1 };
    int amark, bmark;
    int i;
    vastus.mark = a->mark * b->mark;
    amark = a->mark;
    bmark = b->mark;
    a->mark = 1;
    b->mark = 1;
    vastus.pikkus = a->pikkus;
    for (i = a->pikkus - 1; i >= 0; i--) {
        nihuta(&rida, 1);
        rida.numbrid[0] = a->numbrid[i];
        vastus.numbrid[i] = 0;
        while (vordle(&rida, b) != 1) {
            vastus.numbrid[i]++;
            tmp = lahuta(&rida, b);
            rida = tmp;
        }
    }
    eemaldaNullid(&vastus);
    a->mark = amark;
    b->mark = bmark;

    return vastus;
}
```

4.5.6 Ülesanne – anagrammid 3

Suurte arvudega arvutamise näitlikustamiseks vaatame veel üht varianti anagrammide ülesandest. Kõik eelnev on jäänud samaks, kuid moodulit pole enam vaja võtta ning piirid on kõrgemad: kasutatavaid tähti on kuni tuhat ja korduvate tähtede arv pole piiratud.

Kasutame lahenduses omaloodud arvutüüpi `suurarv`:

```
int leiaAnagrammideArv()
{
    int n;
    cin >> n;
    int m[n];
    int i, j = 1, a;
    suurarv total;
    total.mark = 1;
    total.pikkus = 1;
    total.numbrid[0] = 1;
    for (i = 0; i < n; i++) {
        cin >> m[i];
    }
    sort(m, m + n);
    reverse(m, m + n);
    for (i = 0; i < n; i++) {
        int k = j + m[i];
        suurarv f; // kasutame omaloodud suure arvu tüüpi
        f.mark = 1;
        f.pikkus = 1;
        f.numbrid[0] = 1;
        suurarv f2 = faktoriaal(m[i]);
        for (a = j; a < k; a++) {
            suurarv b;
            int_suuravuks(a, &b);
            f = korruta(&f, &b);
        }
        suurarv t = jaga(&f, &f2);
        total = korruta(&total, &t);
        j = k;
    }
    kirjuta(&total);
}
```

4.5.7 Suured arvud erinevates programmeerimiskeeltes

Javas on suurte täisarvudega arvutamiseks standardteegis olemas klass `BigInteger`, mis töötab sarnaselt eelpool illustreeritud `suurarv` klassiga. Tuleb siiski meeles pidada, et just nagu klassi `suurarv` meetodid, on ka `BigInteger`'i meetodid palju keerulisemad ja aeglasemad kui

operatsioonid tavaliste täisarvutüüpidega, seega pole päris igaks juhuks mõtet neid kasutada. Siin on näide Anagrammi ülesande lahendamisest Javas:

```
import java.math.BigInteger;
import java.util.Arrays;
import java.util.Scanner;

public class Main {

    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        int n;
        n = scanner.nextInt();
        int[] m = new int[n];
        int i, j = 1, a;
        BigInteger total = BigInteger.valueOf(1);
        for (i = 0; i < n; i++) {
            m[i] = scanner.nextInt();
        }
        Arrays.sort(m);
        for (i = 0; i < n; i++) {
            int k = j + m[n - i - 1];
            BigInteger f = BigInteger.valueOf(1);
            BigInteger f2 = BigInteger.valueOf(1);
            for (a = 2; a <= m[n - i - 1]; a++) {
                f2 = f2.multiply(BigInteger.valueOf(a));
            }
            for (a = j; a < k; a++) {
                f = f.multiply(BigInteger.valueOf(a));
            }
            BigInteger t = f.divide(f2);
            total = total.multiply(t);
            j = k;
        }
        System.out.println(total);
    }
}
```

Pythonis on mõnes mõttes asi veelgi mugavam: seal kasutatakse seda tüüpi/klassi, mis parasjagu vaja. Aga siingi tekib probleem, et tavapärasest pikkusest välja minnes muutub arvutamine aeglasemaks, nii et võimalusel on tark neid piire jälgida.

4.5.8 Logaritm ja eksponent suurte arvudega arvutamisel

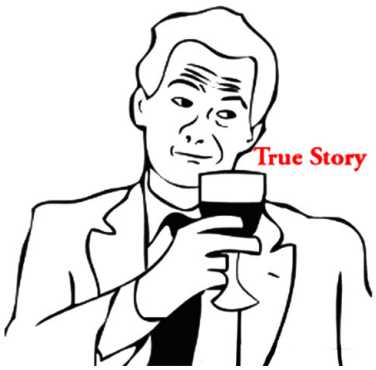
Tavaelu ülesannetes pole vastusest mooduli võtmine enamasti siiski lubatud ja suurte täisarvude realiseerimine (või valmisteegi kasutamine) võib olla liiga aeglane. Kui absoluutset täpsust ei ole vaja, aga on siiski tarvis väga suurte arvudega korrutada ja jagada, saab kasutada ka logaritmi ja eksponenti.

Logaritmi omadustest on teada, et

- $a^{\log_a b} = b$,
- $\log(a \cdot b) = \log a + \log b$,
- $\log(a/b) = \log a - \log b$.

Siit järeldub, et näiteks $ab/c = e^{(ln(a)+ln(b)-ln(c))}$ – oleme korrutamise ja jagamise asendanud logaritmi, liitmise-lahutamise ning eksponendiga.

Illustratsiooniks järgmine päriselu näide: kord tuli mul lahendada probleem mänguteooria vallast, täpsemalt seoses internetipokkeriga.



Vastane võib teatud situatsioonis teha kas käigu A või käigu B. See, kui tihti ta teeb ühe või teise käigu, annab olulise vihje tema strateegia kohta. Kui vastane kasutab strateegiat x_1, on käigu A tegemise tõenäosus p_1, kui strateegiat x_2, siis p_2 jne.

Olgu meil N vaatlust, mille jooksul vastane tegi k korda käiku A. Naiivne meetod oleks eeldada, et käigu A tegemise tõenäosus ongi k/N, kuid kui vaatluste arv on ebapiisav, annab see kergesti eksitava tulemuse.

Selle asemel proovisin hinnata, mis on tõenäosus, et vastane kasutab strateegiat x_1, x_2 jne.

Kui vastane kasutab strateegiat x, millele vastab käigu A tegemise tõenäosus p, siis N käiguga tulemuse k saamise tõenäosus avaldub binoomvalemist:

$$P = p^k \cdot (1-p)^{N-k} \cdot C(N,k)$$

Tõenäosuste p_1, p_2 jne läbimängimine annab erinevad tulemused P_1, P_2 jne. Nende tõenäosuste omavaheline võrdlemine annab omakorda võimaluse hea vastustrateegia koostamiseks. Konkreetsed mänguteoreetilised kaalutlused jäävad praegusest teemast väljapoole, aga raskus tekkis lihtsalt ülaltoodud arvutuse sooritamisel. Kui N ja k on piisavalt suured arvud (tuhandetes), siis hoolimata sellest, et lõppvastus on reaalarv nulli ja ühe vahel, ei mahu vahetulemused ühegi tavalise arvutüübi sisse.

Appi tuleb logaritm ja selle omadused:

$$P = e^{ln(P)}$$

ja

$$ln(P) = \ln\left(p^k \cdot (1-p)^{N-k} \cdot C(N,k)\right) = ln(p^k \cdot (1-p)^{N-k} \cdot (N!/k!/(N-k)!) =$$

$$= ln(p^k) + ln((1-p)^{N-k}) + ln(N!) - ln(k!) - ln((N-k)!) =$$

$$k \cdot ln(p) + (N-k) \cdot ln(1-p) + ln(N!) - ln(k!) - ln((N-k)!)$$

Faktoriaali logaritmi leidmiseks saab kasutada Stirlingi valemit:

$$ln(n!) \approx ln(n) \cdot n - n + ln(\pi \cdot 2 \cdot n) / 2.$$

koguvalem on seega:

$$P = e^{k \cdot ln(p) + (N-k) \cdot ln(1-p) + S(N) - S(k) - S(N-k)},$$

kus

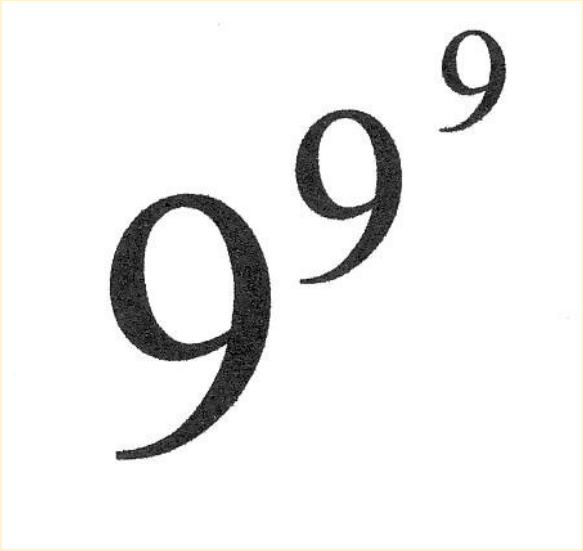
$$S(x) = ln(x) \cdot x - x + ln(\pi \cdot 2 \cdot x) / 2$$

Kõik vahetulemused on mõistliku suurusega reaalarvud ja ka lõppvastuse täpsus on väga hea. Kokkuvõttes oli kood võrreldes eraldi arvutüübiklassi loomisega kümneid kordi lühem ja töötas tuhandeid kordi kiiremini.

4.6 KONTROLLÜLESANDED

4.6.1 Suur arv

Väikeste arvude puhul on kerge kontrollida, kas nad jaguvad mõne teise arvuga. Suuremate arvude jaoks õpetatakse koolis mitmesuguseid jaguvuse tunnuseid, mis võimaldavad neid kontrolle lihtsamalt sooritada. Arvuti jaoks on need piirid samuti olemas, aga nad on oluliselt kõrgemad.
Sisendi ainsal real on antud suur arv `M` ($1 \le M \le 10^{1000}$).
Väljundisse kirjutada 12-kohaline kahendarv, mille iga koht vastab sellele, kas arv `M` jagub vastava arvuga (1-12) või ei.



NÄIDE 1:
`0`
Vastus:
`111111111111`
NÄIDE 2:
`379749833583241`
Vastus:
`100000000010`
NÄIDE 3:
`3909821048582988049`
Vastus:
`100000100000`
NÄIDE 4:
`10`
Vastus:
`110010000100`

4.6.2 Faktoriaali lõpp

`N!` tähistab arvu `N` faktoriaali. Järgnevas tabelis on toodud mõnede arvude faktoriaalid:

N	1	2	3	5	10	20
N!	1	2	6	120	3628800	2432902008176640000

Nagu näha, lõpevad neljast suuremate arvude faktoriaalid nulli(de)ga. Antud juhul on meil aga vaja teada, mis on arvu faktoriaali viimane nullist erinev number.
Sisendis on antud täpselt üks naturaalarv `N` (`1≤N≤10000`). Leida `N!` viimane nullist erinev number ja kirjutada see väljundisse.

NÄIDE 1:
2
Vastus:
2
NÄIDE 2:
26
Vastus:
4
NÄIDE 3:
9999
Vastus:
8

4.6.3 Goldbachi hüpotees

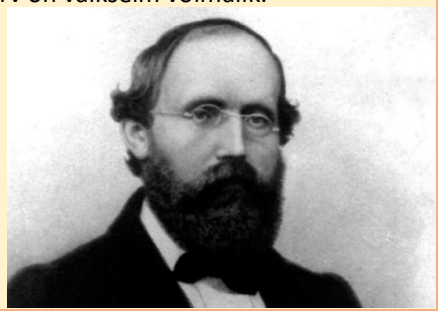
1742. aastal kirjutas saksa amatöörmatemaatik Christian Goldbach oma aja (ja võib-olla ka kogu ajaloo) kuulsaimale matemaatikule Leonhard Eulerile kirja, kus kirjeldas huvitavat avastust, mida tänapäeval tuntakse Goldbachi hüpoteesi nime all.
Kaasaegses sõnastuses kõlab see järgmiselt: iga paarisarv alates neljast on väljendatav kahe algarvu summana.
Näiteks kehtib 8=3+5, 20=3+17, 42=19+23 jne.
Hoolimata paljude matemaatikute tööst vahepealsete sajandite jooksul pole Goldbachi hüpoteesi kunagi tõestatud, kuid üsna suurte arvudeni ($4x10^{18}$) on näidatud, et hüpotees kehtib.
Praeguses ülesandes tuleb sul kirjutada programm, mis kontrollib Goldbachi hüpoteesi mingi hulga arvude jaoks.
Sisendi esimesel real on arv `N` (`1≤N≤1000`) – kontrollitavate arvude arv.
Järgmisel `N` real on igaühel üks paarisarv vahemikus `4` kuni `1 000 000`.
Väljundisse kirjutada `N` rida: igaühel neist kaks algarvu, mille summa on vastav arv sisendist. Kui võimalikke vastuseid on mitu, siis väljastada see, kus esimene arv on väikseim võimalik.

NÄIDE:

```
3
8
20
42
```

Vastus:

```
3 5
3 17
5 37
```



4.6.4 Klaaskuulid

Hulk klaaskuule tuleb mahutada kastidesse. Kaste on kahesuguseid: suurusega n_1 ja maksumusega c_1 ning suurusega n_2 ja maksumusega c_2. Kõik kastid peavad täpselt täis saama ja samas peab nende kogumaksumus olema minimaalne.
Sisendi esimesel real on klaaskuulide arv `N`. Teisel real on arvud c_1 ja n_1 ning kolmandal real on arvud c_2 ja n_2. Kõik arvud on positiivsed täisarvud maksimaalse suurusega $2*10^9$.
Väljundisse kirjutada kui palju mõlemat sorti kaste vaja läheb. Kui lahendit ei ole, kirjutada väljundisse `-1`.

NÄIDE 1:

```
43
1 3
2 4
```

Vastus:

```
13 1
```

NÄIDE 2:

```
40
5 9
5 12
```

Vastus:

```
-1
```

4.6.5 VÜK-võimsus

Igal naturaalarvude paaril on unikaalne vähim ühiskordne (VÜK). Samas võib üks ja seesama arv olla VÜKiks mitmele erinevale arvupaarile.
Näiteks 12=VÜK(1,12)=VÜK(2,12)=VÜK(3,4) jne. Nimetame selliste erinevate paaride arvu esialgse arvu VÜK-võimsuseks.
Sisendis on antud on täisarv `N (1≤N≤2*10^9)`. Leida tema VÜK-võimsus ja kirjutada see arv väljundisse.

NÄIDE 1:
2
Vastus:
2

NÄIDE 2:
12
Vastus:
8

NÄIDE 3:
24
Vastus:
11

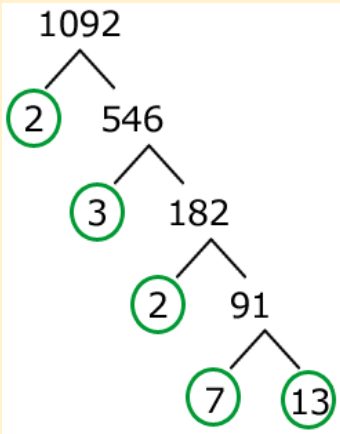
NÄIDE 4:
101101291
Vastus:
5

4.6.6 Suurim algtegur

Aritmeetika põhiteoreem ütleb, et iga naturaalarvu saab täpselt ühel viisil jagada algteguriteks. Sellest järeldub muuhulgas, et igal ühest suuremal naturaalarvul on olemas ka suurim algtegur.
Sisendis on antud täisarv `N (1≤N≤10^14)`. Leida selle suurim algtegur ja kirjutada väljundisse.

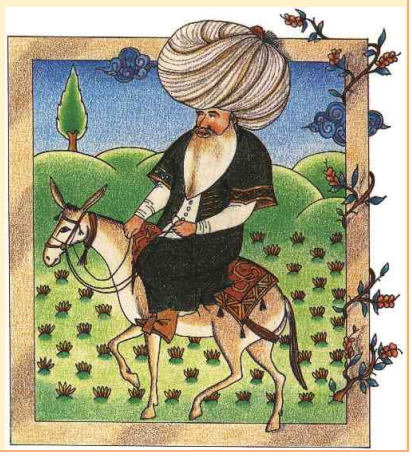
NÄIDE 1:
6
Vastus:
3

NÄIDE 2:
100
Vastus:
5



4.6.7 Vahetusraha

Hodža Nasreddin kaupleb turul aedviljaga. Kui aga keegi talt midagi ostab, siis teeb ta nägu, et tal pole õiget vahetusraha. Täpsemalt kasutab ta sellist reeglit: kui ostja annab talle A dirhemit, siis küsib ta ostjalt lisaks veel B dirhemit, nii et VÜK(A, B)=C, väites, et siis saab ta suurema raha tagasi anda.
Aita ostjal Hodža Nasreddini vastu saada ja leida tal arv B.
Sisendis on kaks naturaalarvu: A ja C ($1 \le A, C \le 10^7$). Väljundisse kirjutada arv B. Et ostja võimalikult vähe riskiks, peab B olema vähim võimalik, mille puhul võrdus kehtib. Kui sellist arvu B ei leidu, väljastada -1.



NÄIDE 1:
`2 6`
Vastus:
`3`
NÄIDE 2:
`32 1760`
Vastus:
`55`
NÄIDE 3:
`7 16`
Vastus:
`-1`

4.6.8 Taandumatud murrud

Murdu `m/n` nimetatakse lihtmurruks, kui $0 \le m < n$ ja taandumatuks, kui `SÜT(m, n)=1`. Sisendis on antud on naturaalarv `N` ($1 \le n \le 10^9$). Leida ja väljastada kui palju on taandumatuid lihtmurde, mille nimetajaks on `N`.

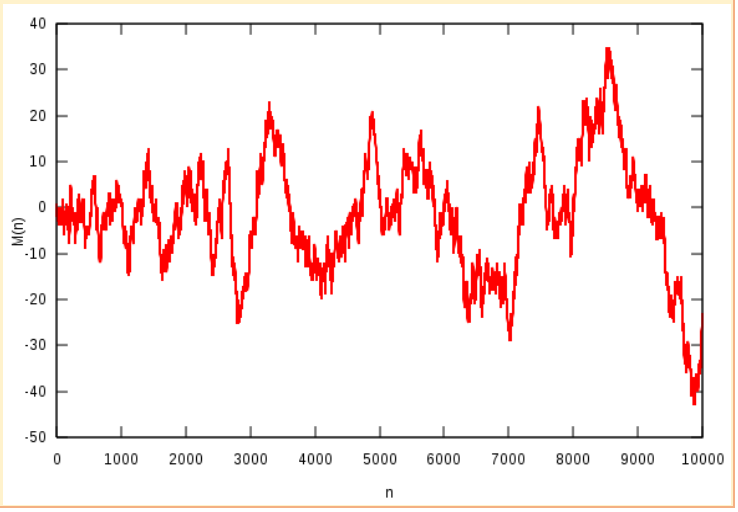
NÄIDE 1:
`12`
Vastus:
`4` (1/12, 5/12, 7/12, 11/12)

NÄIDE 2:
`123456`
Vastus:
`41088`
NÄIDE 3:
`7654321`
Vastus:
`7251444`

4.6.9 Mertensi funktsioon

Mertensi funktsioon on defineeritud kui $M(n) = \sum_{k=1}^{n} \mu(k)$, kus $\mu(k)$ on Möbiuse funktsioon. Möbiuse funktsioon on omakorda defineeritud järgmiselt:
$\mu(n)$ on
0, kui arv n jagub mingi algarvu ruuduga,
1, kui n ei jagu ühegi algarvu ruuduga ning tal on paarisarv algtegureid,
-1, kui n ei jagu ühegi algarvu ruuduga ning tal on paaritu arv algtegureid.
Mertensi funktsioonil on mitmesuguseid huvitavaid omadusi, näiteks see, et funktsiooni väärtus ostsilleerib positiivsete ja negatiivsete väärtuste vahel, kusjuures võnked kasvavad aja jooksul (vt joonist).
Sisendis on antud arv `N` (`1≤N≤10⁶`). Väljundisse kirjutada arv M(n).

NÄIDE 1:
`20`
Vastus:
`-3`
NÄIDE 2:
`15`
Vastus:
`-1`
NÄIDE 3:
`73`
Vastus:
`-4`
NÄIDE 4:
`144`
Vastus:
`-1`



4.6.10 Mõrvamüsteerium

Sherlock Rebane jälitab kurjategijat, kes on endast maha jätnud rea kahendkoodis vihjeid. On aga teada, et õiged vihjed on ainult need, kus kahendkoodile vastav arv jagub arvuga 131071.
Sisendi ainsal real on ühtedest ja nullidest koosnev string pikkusega 1…1000. Kui sellele vastav arv jagub arvuga 131071, kirjutada väljundisse `1`, vastasel korral `0`.

NÄIDE 1:
`101111111111111111101`
Vastus:
`1`
NÄIDE 2:
`11111100111111111100111`
Vastus:
`0`

4.7 VIITED LISAMATERJALIDELE

Kaasasolevas failis VP_lisad.zip, peatükk4 kaustas on abistavad failid käesoleva peatüki materjalidega põhjalikumaks tutvumiseks:

Failid	Kirjeldus
Loos.cpp, Loos.java, Loos.py	Loosiümbrike ülesanne (paarsus)
Anagrammid2.cpp, Anagrammid2.java, Anagrammid2.py	Anagrammid 2 (moodul)
Hammasrattad.cpp, Hammasrattad.java, Hammasrattad.py	Hammasrataste ülesanne (SÜT)
Arvuteooria.cpp, Arvuteooria.java, Arvuteooria.py	SÜT, VÜK jvm funktsioonid.
Algarv.cpp, Algarv.java, Algarv.py	Algarvu Scrabble'i ülesanne (algarvud)
Algtegurid.cpp, Algtegurid.java, Algtegurid.py	Algteguriteks lahutamine
Teisendus.cpp, Teisendus.java, Teisendus.py	Ühest arvusüsteemist teise teisendamine kahel moel
Arvusysteemid.cpp, Arvusysteemid.java, Arvusysteemid.py	Positsiooniliste arvusüsteemide ülesanne
Anagrammid3.cpp, Anagrammid3.java, Anagrammid3.py	Anagrammide ülesanne täisvastusega (Suurte arvudega arvutamine)