Teoria numerelor

Capitolul 20

- **Algoritmul extins al lui Euclid**
- **Aritmetică modulară**
- **Ridicarea la putere**
- **Rezumat**
- **Implementări sugerate**
- **Probleme propuse**
- **Soluțiile problemelor**

În cadrul acestui capitol vom prezenta câteva detalii referitoare la teoria numerelor. Pentru început vom prezenta cunoscutul algoritm al lui Euclid, în varianta sa extinsă.

În continuare, vom descrie modul în care se realizează operațiile modulo, iar în final vom arăta modul în care poate fi efectuată eficient operația de ridicare la putere.

20.1. Algoritmul extins al lui Euclid

Algoritmul lui Euclid este, cu siguranță, unul dintre cei mai cunoscuți şi mai des utilizați algoritmi. El poate fi utilizat pentru a determina cel mai mare divizor comun a două numere naturale.

20.1.1. Algoritmul lui Euclid

Paşii algoritmului sunt foarte simpli; unul dintre numere este considerat a fi deîmpărțitul, iar celălalt împărțitorul. La fiecare pas, se va calcula restul împărțirii întregi a deîmpărțitului la împărțitor. În cazul în care acesta este 0, se consideră că împărțitorul este cel mai mare divizor comun al numerelor. În caz contrar, împărțitorul devine deîmpărțit, restul devine împărțitor şi se trece la pasul următor.

Până la urmă restul va deveni 0 (deoarece scade la fiecare pas), deci după un număr finit de paşi vom determina cel mai mare divizor comun.

20.1.2. Implementarea iterativă

Algoritmul poate fi implementat folosind o simplă structură repetitivă, corpul său conținând operațiile care trebuie efectuate la fiecare pas.

Versiunea iterativă a algoritmului este prezentată în continuare:

```
Algoritm Euclid(m,n):
                                    { m, n – numerele al căror cmmdc este căutat }
  repetă
    r ← rest[m/n]
    m ← n
    n ← r
  cât timp r > 0
  returnează m
sfârşit algoritm
```

20.1.3. Implementarea recursivă

O a doua variantă de implementare a algoritmului lui Euclid este utilizarea unei funcții recursive. Această variantă este prezentată în cele ce urmează:

```
Algoritm Euclid(m,n):
  dacă n = 0 atunci
    returnează m
  altfel
    returnează Euclid(n,rest[m/n])
  sfârşit dacă
sfârşit algoritm
```

Practic, în această variantă, înlocuirea deîmpărțitului cu împărțitorul se realizează prin autoapelul recursiv.

20.1.4. Algoritmul extins

Forma extinsă a algoritmului lui Euclid este utilizată pentru a determina, pe lângă cel mai mare divizor comun, anumite informații care se pot dovedi utile.

Mai exact, acest algoritm poate fi folosit pentru a identifica două valori x şi y astfel încât să avem $d = m \cdot x + n \cdot y$, unde d este cel mai mare divizor comun al numerelor m şi n.

După cum veți vedea în momentul în care vom prezenta algoritmul, acesta nu este mai lent decât cel clasic. Aşadar, aceste informații suplimentare se determină, practic, fără a consuma timp suplimentar.

20.1.5. Implementarea algoritmului extins

După cum se poate vedea, algoritmul extins al lui Euclid va returna trei valori şi anume d, x şi y, pe baza a doi parametri m şi n.

Modul de calcul al acestor valori este următorul:

```
Algoritm EuclidExtins(m,n):
  dacă n = 0 atunci
    returnează (a,1,0)
  altfel
    (d',x',y') ← EuclidExtins(n,rest[m/n])
    (d,x,y) ← (d',y',[m/n]·y')
    returnează (d,x,y)
  sfârşit dacă
sfârşit algoritm
```

20.2. Aritmetică modulară

În cadrul acestei secţiuni vom introduce aritmetica modulară. Aceasta este foarte asemănătoare cu aritmetica numerelor naturale, singura diferenţă fiind faptul că rezultatele tuturor operaţiilor trebuie să fie numere cuprinse între 0 şi o valoare dată.

Practic, se efectuează operaţiile obişnuite şi rezultatele sunt înlocuite cu restul împărţirii lor la un număr *n*. Aritmetica modulară mai poartă şi denumirea de aritmetică modulo *n*.

20.2.1. Adunare, scădere şi înmulţire

Având în vedere cele amintite anterior, este foarte uşor să observăm că, în aritmetica modulară, calculele se efectuează pe baza următoarelor formule:

```
a + b (mod n) = rest[(a + b) / n]
a · b (mod n) = rest[(a · b) / n].
```

De asemenea, se observă foarte uşor că, în aritmetica modulară, scăderea unei valori *k* este echivalentă cu adunarea valorii *n* – *k*. Aceasta se datorează faptului că avem:

```
(n - k) + k (mod n) = rest[(n - k + k) / n] = rest[n / n] = 0.
```

20.2.2. Proprietăţi

Pe baza modului de efectuare a operaţiilor în aritmetica modulară se observă foarte uşor că următoarele relaţii sunt respectate întotdeauna:

```
(a + b) (mod n) = ((a (mod n)) + ((b mod n)) (mod n)
(a · b) (mod n) = ((a (mod n)) · ((b mod n)) (mod n).
```

Cu alte cuvinte, se poate spune că restul împărţirii la o valoare *n* a sumei a două numere naturale este egal cu restul împărţirii la *n* a sumei resturilor împărţirii la *n* ale celor două numere.

Similar, restul împărţirii la o valoare *n* a produsului a două numere naturale este egal cu restul împărţirii la *n* a produsului resturilor împărţirii la *n* ale celor două numere.

20.2.3. Inversul

În cazul în care valoarea n este un număr prim, pentru orice număr mai mic decât n (dar mai mare decât 0), va exista un alt număr (cuprins tot între 1 şi $n - 1$), astfel încât produsul celor două numere să fie 1. Acest al doilea număr este numit inversul primului număr.

Proprietatea nu este respectată în cazul în care valoarea n nu este număr prim. De exemplu, pentru $n = 4$, nu există nici un invers al valorii $k = 2$.

20.3. Ridicarea la putere

Există multe variante prin care se pot efectua operațiile de ridicare la putere. În cadrul acestei secțiuni vom prezenta abordarea clasică, ineficientă, precum şi o metodă mult mai rapidă care permite realizarea acestei operații.

20.3.1. Înmulțiri succesive

Practic, pentru a ridica un număr x la puterea n, va trebui să efectuăm un număr de $n - 1$ înmulțiri. De exemplu, am putea utiliza formula recursivă $x^n = x^{n-1} \cdot x$.

Un algoritm simplu care utilizează această formulă este:

```
Algoritm Ridicare_la_putere(x,n):
  dacă n = 0 atunci
    returnează 1
  altfel
    returnează x · Ridicare_la_putere(x,n-1)
  sfârşit dacă
sfârşit algoritm
```

Evident, există şi variante iterative dar, toate acestea au ordinul de complexitate $O(n)$.

20.3.2. Algoritmul rapid

O variantă mult mai rapidă de determinare a puterii unui număr se bazează tot pe o formulă recursivă şi anume:

$$x^n = \begin{cases} 1 & n = 0 \\ x^{[n/2]} \cdot x^{[n/2]} & n \text{ par} \\ x \cdot x^{[n/2]} \cdot x^{[n/2]} & n \text{ impar} \end{cases}$$

Este foarte uşor de observat că formula este corectă şi pe baza acesteia poate fi implementat foarte uşor următorul algoritm:

```
Algoritm Ridicare_rapidă_la_putere(x,n):
  dacă n = 0 atunci
    returnează 1
  altfel
    dacă k este par atunci
      returnează Ridicare_rapidă_la_putere(x,[n/2])·
                 Ridicare_rapidă_la_putere(x,[n/2])
    altfel
      returnează Ridicare_rapidă_la_putere(x,[n/2])·
                 Ridicare_rapidă_la_putere(x,[n/2])·x
    sfârşit dacă
  sfârşit dacă
sfârşit algoritm
```

20.3.3. Ridicare la putere în aritmetica modulară

Este foarte uşor să adaptăm algoritmul prezentat anterior pentru a determina restul împărțirii întregi a unei valori de forma x^k la o valoare n.

Tot ce trebuie să facem este să calculăm un rest la final. Totuşi, dacă utilizăm proprietatea potrivit căreia pentru înmulțirea modulară avem:

```
(a · b) (mod n) = ((a (mod n)) · ((b mod n)) (mod n),
```

nu vom ajunge niciodată să lucrăm cu numere foarte mari. Aşadar, o simplă transformare a algoritmului poate duce la eliminarea operațiilor costisitoare cu numere mari.

Prezentăm acum o variantă a algoritmului de ridicare la putere în aritmetica modulară:

```
Algoritm Ridicare_la_putere_modulo_n(x,k,n):
  dacă k = 0 atunci
    returnează 1
  altfel
    dacă k este par atunci
      returnează rest[(Ridicare_la_putere(x,[k/2],n)·
                       Ridicare_la_putere(x,[k/2],n)) / n]
    altfel
      returnează rest[
                    (rest[(Ridicare_la_putere(x,[k/2],n)·
                           Ridicare_la_putere(x,[k/2],n)) / n])
                    · x / n
                  ]
    sfârşit dacă
  sfârşit dacă
sfârşit algoritm
```

Pentru o mai mare claritate, putem rescrie secvența corespunzătoare unei puteri impare astfel:

```
a ← Ridicare_la_putere_modulo_n (x,[k/2],n)
b ← rest[(a·a)/n]
returnează rest[(b·x)/n]
```

Un avantaj principal al acestei abordări constă în faptul că nu se mai realizează două autoapeluri recursive care furnizează aceeaşi valoare. Aşadar, un algoritm eficient poate fi scris astfel:

```
Algoritm Ridicare_la_putere_modulo_n_eficient(x,k,n):
  dacă k = 0 atunci
    returnează 1
  altfel
    dacă k este par atunci
      a ← Ridicare_la_putere_modulo_n_eficient (x,[k/2],n)
      returnează rest[(a·a)/n]
    altfel
      a ← Ridicare_la_putere_modulo_n_eficient (x,[k/2],n)
      b ← rest[(a·a)/n]
      returnează rest[(b·x)/n]
    sfârşit dacă
  sfârşit dacă
sfârşit algoritm
```

Evident, vom putea rescrie şi algoritmul pentru aritmetica clasică astfel:

```
Agoritm Ridicare_la_putere_eficient(x,n):
  dacă n = 0 atunci
    returnează 1
  altfel
    dacă n este par atunci
      a ← Ridicare_la_putere_eficient(x,[n/2])
      returnează a·a
    altfel
      a ← Ridicare_la_putere_eficient(x,[k/2],n)
      returnează a·a·x
    sfârşit dacă
  sfârşit dacă
sfârşit algoritm
```

20.4. Rezumat

În cadrul acestui capitol am realizat o scurtă introducere în teoria numerelor. Pentru început am descris algoritmul extins al lui Euclid şi am prezentat modul în care poate fi implementat acesta.

În continuare am realizat o scurtă prezentare a aritmeticii modulare şi am descris modul în care trebuie efectuate operaţiile, precum şi câteva proprietăţi utile.

În final am descris mai multe modalităţi prin care se realizează rapid ridicarea la putere, atât pentru aritmetica modulară, cât şi pentru cea clasică.

20.5. Implementări sugerate

Pentru a vă însuşi noţiunile prezentate în cadrul acestui capitol vă sugerăm să realizaţi implementări pentru:

1. algoritmul extins al lui Euclid;
2. determinarea restului împărţirii întregi la o valoare *n* a sumei elementelor unui şir de numere;
3. determinarea restului împărţirii întregi la o valoare *n* a produsului elementelor unui şir de numere;
4. efectuarea de operaţii de ridicare la putere prin diverse metode; de asemenea, este indicat să comparaţi timpii de execuţie ai algoritmilor.

20.6. Probleme propuse

În continuare vom prezenta enunţurile câtorva probleme pe care vi le propunem spre rezolvare. Toate aceste probleme pot fi rezolvate folosind informaţiile prezentate în cadrul acestui capitol. Cunoştinţele suplimentare necesare sunt minime.

20.6.1. Pluto

Descrierea problemei

Pluto are în faţă *M* bilețele roşii şi *N* bilețele galbene. El doreşte să formeze mai multe grupuri de bilețele, astfel încât toate grupurile să conţină acelaşi număr de bilețele, toate bilețelele dintr-un grup să aibă aceeaşi culoare şi numărul de bilețele dintr-un grup să fie cât mai mare posibil.

După ce a reuşit să facă împărţeala, *Pluto* a descoperit într-o ascunzătoare o mulţime de alte bilețele roşii şi galbene. Imediat a început să le împartă în grămezi de *M* bilețele roşii sau *N* bilețele galbene. Fiecare grămadă conţine bilețele de aceeaşi culoare.

Numărul total al grămezilor formate este foarte mare, aşa că *Pluto* îşi imaginează că acum poate face orice. A observat acum grupurile de bilețele pe care le-a construit anterior şi i-a venit o idee năstruşnică. El doreşte să aleagă un număr *X* de grămezi cu

bilețele roşii şi un număr *Y* de grămezi cu bilețele galbene, astfel încât diferența dintre numărul total de bilețele roşii şi numărul total de bilețele galbene să fie egală cu numărul de bilețele dintr-unul din grupurile construite la început. Nu s-a gândit dacă să aleagă mai multe bilețele roşii sau mai multe bilețele galbene. El doreşte doar ca diferența să fie egală cu numărul de bilețele dintr-un grup, indiferent dacă are mai multe bilețele galbene sau mai multe bilețele roşii.

Date de intrare
Fişierul de intrare **PLUTO.IN** conține o singură linie pe care se vor afla numerele *M* şi *N*, separate printr-un spațiu.

Date de ieşire
Fişierul de ieşire **PLUTO.OUT** va conține o singură linie pe care se vor afla trei numere întregi, separate prin câte un spațiu. Primul dintre ele reprezintă numărul de bilețele din fiecare dintre grupurile construite inițial, al doilea reprezintă numărul de grămezi cu bilețele roşii, iar al treilea reprezintă numărul de grămezi cu bilețele galbene.

Restricții şi precizări
- $1 \le M, N \le 1000000000$;
- există posibilitatea ca Pluto să nu aleagă nici o grămadă galbenă sau nici o grămadă roşie.

Exemple

PLUTO.IN	PLUTO.OUT
4 6	2 1 1

PLUTO.IN	PLUTO.OUT
10 5	5 0 1

PLUTO.IN	PLUTO.OUT
24 63	3 8 3

Timp de execuție: 1 secundă/test

20.6.2. Hoții

Descrierea problemei
Regula ghildei hoților este simplă. La începutul săptămânii, fiecare hoț îşi ascunde toate proprietățile. În timpul săptămânii, fiecare hoț trebuie să predea ghildei toți galbenii pe care reuşeşte să îi obțină în momentul în care are asupra sa *N* galbeni. Numă-

rul de galbeni pe care îi deține la sfârşitul săptămânii vor rămâne în proprietatea sa şi îi va ascunde pentru a începe o nouă săptămână de la 0.

Din ghildă fac parte K hoți şi, pentru fiecare dintre aceştia se cunoaşte numărul jafurilor pe care le-a efectuat, precum şi valoarea fiecărui jaf.

În nici un moment un hoț nu poate avea asupra sa decât cel mult $N - 1$ galbeni (decât imediat după un jaf) deoarece imediat ce are mai mult de N galbeni, trebuie să îi predea ghildei. Mai mult, dacă după ce predă N galbeni, numărul de galbeni va fi din nou mai mare sau egal cu N, el va preda din nou N galbeni şi "depunerile" vor continua până în momentul în care hoțul va avea mai puțin de N galbeni.

Va trebui să determinați, pentru fiecare hoț în parte, numărul de galbeni pe care acesta îi va avea asupra sa la sfârşitul săptămânii.

Date de intrare

Prima linie a fişierului de intrare **HOTI.IN** conține numărul K al hoților şi numărul N, separate printr-un spațiu. Primul număr de pe fiecare dintre următoarele K linii reprezintă numărul J_i al jafurilor efectuate de un hoț în timpul săptămânii. Linia va mai conține J_i numere, reprezentând "valorile" jafurilor. Numerele de pe o linie vor fi separate prin spații.

Date de ieşire

Fişierul de ieşire **HOTI.OUT** va conține K linii; fiecare dintre acestea va conține un număr întreg, reprezentând numărul de galbeni rămaşi în posesia unui hoț la sfârşitul săptămânii.

Restricții şi precizări

- $1 \le N \le 1000000000$;
- $1 \le K \le 1000$;
- $0 \le J_i \le 1000$;
- valoarea unui jaf este un număr întreg cuprins între 1 şi 1000000000;
- ordinea în care sunt prezentate sumele în fişierul de ieşire trebuie să respecte ordinea hoților din fişierul de intrare.

Exemplu

HOTI.IN
```
3 50
4 1 2 3 4
5 10 20 30 40 50
3 874 9735 835
```

HOTI.OUT
```
10
0
44
```

Timp de execuție: 1 secundă/test

20.6.3. Viruşi

Descrierea problemei
Un virus se află într-un lanț format din K celule. După un timp acesta se înmulțește dând naştere la alți N viruşi. Aceştia vor ocupa următoarele celule din lanț. În momentul în care toate celulele vor conține un virus, ei vor "locui" câte doi în celulă, apoi câte trei şi aşa mai departe. În orice moment numărul viruşilor din oricare două celule va diferi prin cel mult 1.

Aşadar, la a doua generație vom avea $N + 1$ viruşi. După un timp, fiecare va da naştere la alți N viruşi care vor ocupa celulele în acelaşi mod. După a doua generație vom avea $(N + 1)^2 = N^2 + 2 \cdot N + 1$ viruşi.

Va trebui să determinați numărul celulelor aglomerate după cea de-a G-a generație. O celulă este aglomerată dacă există cel puțin o altă celulă care conține mai puțini viruşi (datorită regulii de ocupare a celulelor, o astfel de celulă va conține cu exact un virus mai puțin)

Date de intrare
Prima linie a fişierului de intrare **VIRUSI.IN** conține trei numere naturale, separate prin câte un spațiu, reprezentând numărul N al viruşilor care apar după înmulțirea unui virus, numărul K al celulelor şi numărul G al generațiilor.

Date de ieşire
Fişierul de ieşire **VIRUSI.OUT** va conține o singură linie pe care se va afla numărul celulelor aglomerate după cea de-a G-a generație.

Restricții şi precizări
- $1 \le N \le 10000$;
- $1 \le K \le 1000000000$;
- $1 \le G \le 2000000000$.

Exemple

VIRUSI.IN	VIRUSI.OUT
1 2 3	0

VIRUSI.IN	VIRUSI.OUT
3 3 3	1

VIRUSI.IN	VIRUSI.OUT
534 65 2	30

Timp de execuție: 1 secundă/test

20.7. Soluțiile problemelor

Vom prezenta acum soluțiile problemelor propuse în cadrul secțiunii precedente. Pentru fiecare dintre acestea va fi descrisă doar metoda de rezolvare. Datorită faptului că toate problemele reprezintă simple "transformări" ale elementelor teoretice prezentate în cadrul acestui capitol nu vom mai efectua şi o analiză a complexității algoritmilor.

20.7.1. Pluto

Problema se reduce la a determina cel mai mare divizor comun a două numere M şi N şi apoi a determina o pereche de numere X şi Y, astfel încât $|M \cdot X - N \cdot Y| =$ cmmdc(M, N). Aşadar, va trebui doar să aplicăm algoritmul extins al lui *Euclid* şi să afişăm valorile absolute ale numerelor returnate de acest algoritm.

20.7.2. Hoții

Problema se reduce la efectuarea unor însumări modulo N. Folosind adunările modulare vom determina foarte simplu numărul de galbeni rămaşi în posesia fiecăruia dintre hoți.

20.7.3. Viruşi

Se observă foarte uşor că, după cea de-a G-a generație, numărul total al viruşilor este $(N + 1)^G$. Aşadar, problema se reduce la determinarea valorii $(N + 1)^G$ modulo K. Pentru aceasta vom folosi un algoritm de ridicare la putere în aritmetica modulară.