Teoria numerelor în probleme de algoritmică

În cadrul acestui material ne propunem să abordăm 2 noțiuni foarte importante, des întâlnite și utilizate în probleme de programare competitivă. Ne dorim să le prezentăm atât din punct de vedere al aplicabilității acestora în diverse contexte, cât și din punct de vedere matematic, studiind raționamentul din spatele conceptelor.

1.Ridicare la putere în timp logaritmic

a. Descriere generală

Acest procedeu pornește de la o problemă simplă : fiind date două numere naturale, x și n, se cere a se calcula x^n.

O primă abordare ar fi construirea unei soluții lineare, calculând gradual valoarea cerută, prin înmulțiri repetate cu x.

```
int r = 1;
for(int i = 1; i ≤ n; i++)
{
    r = r * x;
}
return r;
```

Această soluție este realizată în n iterații, având o complexitate $O(n)$.

Pentru valori mici ale lui n, această soluție este acceptabilă, însă pentru valori mari, timpul de execuție va fi substanțial mai mare. În plus, dacă și x tinde să ia valori mari atunci ne vom găsi în situația de overflow. Numărul generat o să depășească limita de stocare a tipului de date int. În acest caz, nu putem reține valoarea exactă generată, fiind necesară o reducere modulară a acesteia, însă aceste situații sunt, de obicei, specificate în enunțul problemei (A se vedea https://infoarena.ro/problema/lgput).

Ne propunem să găsim o soluție mai rapidă și plecăm de la o observație simplă:

$$x^n = \begin{cases} x(x^2)^{\frac{n-1}{2}}, & n \equiv 1 \ (mod\ 2) \\ (x^2)^{\frac{n}{2}}, & n \equiv 0 \ (mod\ 2) \end{cases}$$

Prin urmare, ne folosim de reprezentarea binară a numărului x pentru a vedea ce puteri sunt calculate. Exemplificăm cum este calculat x^{13} :

$13 = 1101_2$

Reprezentarea binară a lui 13 folosește 4 biți, prin urmare algoritmul având 4 iterații:

- Inițial : $r \leftarrow 1$ $(r = x^0)$
- Pasul 1 : $r \leftarrow r^2$ $(r = x^0)$, $bit\ 3 = 1 => r \leftarrow r \cdot x$ $(r = x^1)$
- Pasul 2 : $r \leftarrow r^2$ $(r = x^2)$, $bit\ 2 = 1 => r \leftarrow r \cdot x$ $(r = x^3)$
- Pasul 3 : $r \leftarrow r^2$ $(r = x^6)$, $bit\ 1 = 0 => nu\ facem\ nimic$ $(r = x^6)$
- Pasul 4 : $r \leftarrow r^2$ $(r = x^{12})$, $bit\ 0 = 1 => r \leftarrow r \cdot x$ $(r = x^{13})$

În general, dacă avem reprezentarea binară $n = (b_k b_{k-1} \dots b_0)_2$, definim secvența $r_{k+1}, r_k \dots r_0$, caracterizată de $r_{k+1} = 1$ și de relația de recurență $r_i = r_{i+1}^2 \cdot x^{b_i}, i = k \dots 0$.

Rezultatul final va fi reprezentat de r_0.

O astfel de implementare va avea complexitatea $O(log_2 n)$.

b. Implementare

Acest algoritm poate fi implementat atât recursiv, cât și iterativ.

Varianta recursivă:

```
float exp_log_rec(float x, int n){
    if(n < 0) return exp_log_rec(1.0 / x, -n);
    if(n == 0) return 1;
    if(n % 2 == 0) return exp_log_rec(x*x, n/2);
    if(n % 2 == 1) return x * exp_log_rec(x*x, n/2);
}
```

Varianta iterativă:

```
float exp_log(float x, int n){
    if(n < 0){
        x = 1.0 / x;
        n = (-1) * n;
    }

    if(n == 0) return 1;

    float p = 1;
    while(n > 0){
        if(n%2){
           p = p * x;
        }
        p = p * p;
        n = n /2;
    }
    return p;
}
```

c. Observație + exemplu

Acest algoritm nu se limitează la înmulțirea de numere naturale, el putând fi folosit pentru generarea de puteri în orice semigrup. Spre exemplu, putem genera puteri de polinoame sau matrici, dar pentru o implementare a acestora este necesar ca noi să ne scriem o funcție care ne furnizează produsul a 2 polinoame/matrice. Apelul acestei funcții va înlocui înmulțirea clasică.

A se vedea: https://infoarena.ro/problema/kfib

2.Inversul modular

Pentru a putea discuta despre inversul modular, ar fi bine să începem cu o problemă clasică din geometria combinatorică.

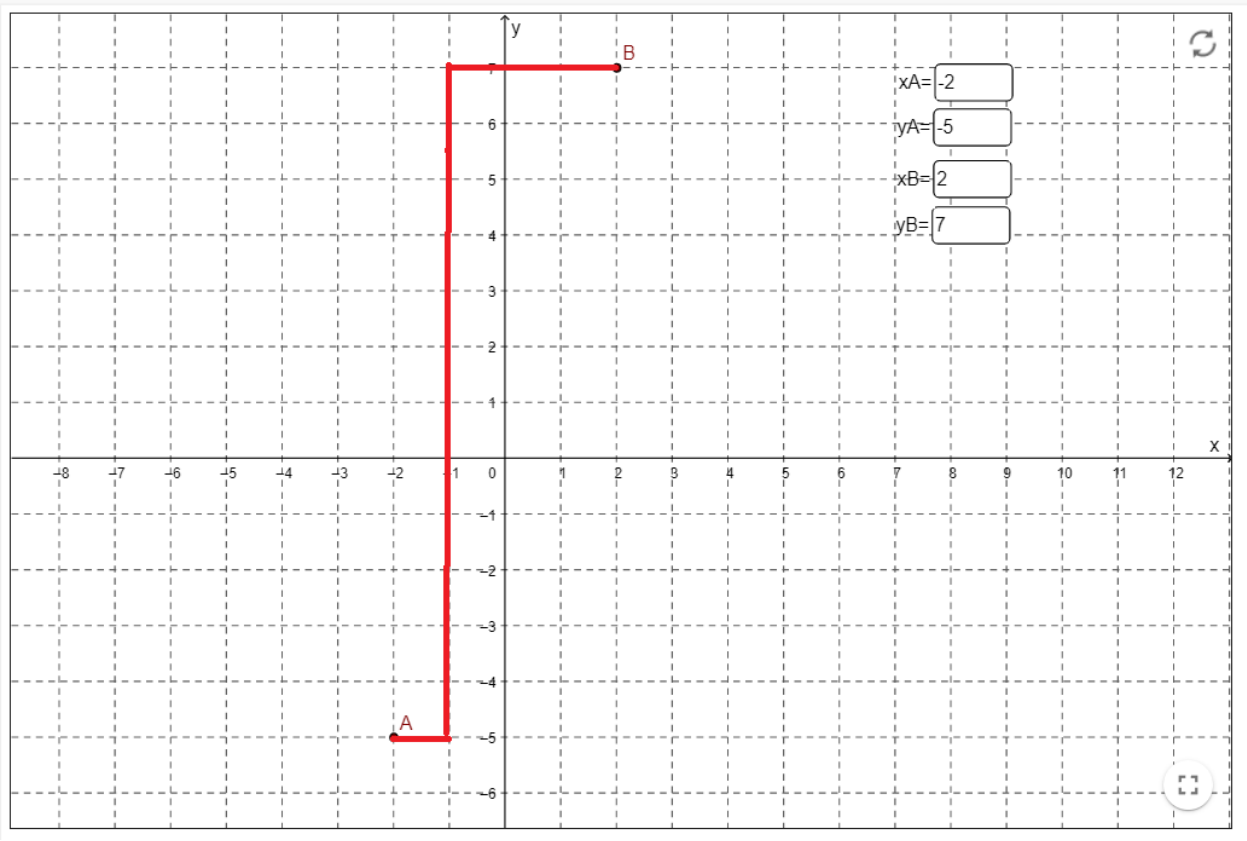
Enunț: Considerăm o rețea laticeală standard, generată de vectorii $(1,0)$ și $(0,1)$. Fie punctele $A(0,0)$ și $B(m,n)$. Se cere să se calculeze numărul drumurilor laticeale între A și B.

Definiție: Numim o rețea laticeală generată de versorii i și j mulțimea punctelor din plan generată de combinațiile lineare dintre i și j, împreună cu sistemul de coordonate asociat.

$$\pi = \{x \cdot i + y \cdot j \mid (x,y) \in \mathbb{Z}^2\}$$

Definiție: Un drum laticeal între două puncte este un drum de lungime minimă care trece doar prin segmente unitare și merge doar în sensul pozitiv al axelor de coordonate.

Exemplu de drum laticeal într-o latice:



Fiecare astfel de drum, având lungime minimă, va fi compus din exact $m+n$ segmente, dintre care m orizontale. Prin urmare, trebuie să numărăm modurile în care putem alege m segmente orizontale din cele $m+n$.

Intuim că răspunsul căutat este $C_{m+n}^{m} = \binom{m+n}{m}$. Mai mult, datorită simetriei,

$$C_{m+n}^{m} = \binom{m+n}{m} = \binom{m+n}{n} = C_{m+n}^{n}$$

Totuși, pentru valori foarte mari ale lui m și n, pentru a se evita simularea greoaie a operațiilor aritmetice, se cere deseori afișarea restului împărțirii rezultatului la o anume valoare.

Prin urmare, ne propunem să calculăm $\binom{m+n}{m}\ \%\ M$ unde M este dat.

Exemplu: Studiem cazul $m = 5, n = 11, M = 19$

$$\binom{m+n}{n} = \binom{16}{5} = \frac{16!}{5! \cdot 11!} = \frac{6 \cdot 7 \cdot \ldots \cdot 16}{11!} = 4368 \equiv 17 (mod\ 19)$$

Simple computații ne arată că $6 \cdot 7 \cdot \ldots \cdot 16 \equiv 11\ (mod\ 19)$ și $11! \equiv 4\ (mod\ 19)$

Dacă este să vedem care sunt operațiile modulare, avem adunarea, scăderea și înmulțirea. Operația de împărțire nu este definită. În acest context a fost introdusă noțiunea de *invers modular*. În loc să împărțim un număr la altul, îl vom înmulți cu inversul său modular.

Definiție: Fie $a \in \mathbb{Z}$. Numim ***inversul modular al lui a* în *raport cu M*** acel număr x pentru care este adevărată relația:

$$ax \equiv 1\ (mod\ M)$$

Teoremă: Dacă $\gcd(a, M) = 1$ atunci x există și este unic.

Observație: x este tot timpul determinat în mulțimea $\{1, 2, \ldots, M-1\}$

Demonstrație: Afirmația de mai sus se poate lega de un context mai larg. Considerăm inelul claselor de resturi $(\mathbb{Z}_M, +, \cdot)$. Inelul este comutativ și, în plus, orice element din acesta care este prim cu M este inversabil, iar inversul acestuia este unic determinat.

Teoremă: Dacă $\gcd(a, M) \neq 1$ atunci x nu există.

Demonstrație: Fără a intra în detalii mult prea tehnice, este suficient să spunem că, în acest caz, inelul $(\mathbb{Z}_M, +, \cdot)$ poate avea divizori ai lui 0.

Mai departe ne propunem să calculăm inversul modular. Propunem spre studiu 2 algoritmi care fac acest lucru.

a. Algoritm bazat pe Teorema lui Euler

Pentru a putea detalia modul de funcționare al acestui algoritm, este nevoie să introducem câteva noțiuni suplimentare.

Descriere generală

Definiție: Numim ***Funcția lui Euler*** / ***Indicatorul lui Euler*** funcția

$$\varphi : \mathbb{N} \to \mathbb{N}$$

$$\varphi(n) = \begin{cases} 1, & n = 0 \\ numărul\ de\ numere\ naturale \leq n\ și\ prime\ cu\ n, & n \geq 0 \end{cases}$$

Exemple: $\varphi(0) = 1, \varphi(1) = 1, \varphi(2) = 1, \varphi(3) = 2, \ldots$

Teoremă: Dacă avem descompunerea în factori primi:

$$n = {p_1}^{\alpha_1} \cdot {p_2}^{\alpha_2} \cdot \ldots \cdot {p_k}^{\alpha_k}$$

atunci

$$\varphi(n) = \prod_{i=1}^{k} {p_i}^{\alpha_i - 1} \cdot (p_i - 1)$$

Demonstrația depășește scopul acestui curs 😊

Observație: $n\ este\ prim => \varphi(n) = n - 1$

Teorema lui Euler: $a^{\varphi(n)} \equiv 1\ (mod\ n)\ \forall\ a, n \in \mathbb{N},\ gcd(a, n) = 1$

Revenind la problema noastră obținem:

$$a^{\varphi(M)} \equiv 1\ (mod\ M)\ ,$$

echivalent cu

$$a^{\varphi(M)-1} \cdot a \equiv 1\ (mod\ M)$$

Prin urmare, inversul modular căutat de noi este $x = a^{\varphi(M)-1}$.

Implementare

Implementarea acestui algoritm are loc în doi pași:

- Calcularea funcției $\varphi(M)$

Oferim o implementare de complexitate $O(\sqrt{M})$. Există soluții și mai eficiente, dar ne este suficientă aceasta 😊

```
int phi(int n) {
    int result = n;
    for (int i = 2; i * i <= n; i++) {
        if (n % i == 0) {
            while (n % i == 0)
                n /= i;
            result -= result / i;
        }
    }
    if (n > 1)
        result -= result / n;
    return result;
}
```

- Calcularea $a^{\varphi(M)-1}$

Pentru acest lucru ne vom folosi de ridicarea la putere în timp logaritmic, prezentată mai sus 😊. Pentru acest lucru vom avea complexitatea $O(log_2 M)$

Complexitatea totală va fi $O(\sqrt{M} + log_2 M)$.

Exemplu:

a = 4, M = 19

$\varphi(M) = 18$

$x = 4^{17} \equiv 5 \ (mod\ 19)$

$\binom{16}{5} \equiv 11 \cdot 5 = 55 \ (mod\ 19) = 17 \ (mod\ 19)$

b. Algoritmul lui Euclid extins

Descriere generală

Și în acest caz introducem niște rezultate auxiliare.

Lema lui Bézout:

$$\forall\, a, b \in \mathbb{Z}^*, dacă\, \gcd(a, b) = d$$

atunci

$$\exists\, x, y \in \mathbb{Z}\ astfel\ încât\ ax + by = d$$

Pentru datele problemei noastre alegem: $b = M$

Avem că $\gcd(a, M) = 1$

$$=> ax + My = 1$$

$$=> -My = ax - 1$$

$$=> M \mid ax - 1$$

$$=> ax + My = 1$$

$$=> ax \equiv 1\ (mod\ M)$$

Prin urmare, acest x generat de lemă este chiar numărul pe care îl căutăm noi. Pentru a găsi numărul nostru, ne folosim de algoritmul lui Euclid extins.

Algoritmul acesta este daor o prelungire a celui normal, folosind 4 șiruri care sunt actualizate la fiecare iterație:

$$(r_n)_{n\geq 0}, (q_n)_{n\geq 2}, (x_n)_{n\geq 0}, (y_n)_{n\geq 0}$$

Avem inițializările:

$$r_0 = a, r_1 = M$$

$$x_0 = 1, x_1 = 0$$

$$y_0 = 0, r_1 = 1$$

și recurențele

$$q_k = r_{k-2} \,/\, r_{k-1}$$

$$r_{k+1} = r_{k-1} - q_k \cdot r_k$$

$$x_{k+1} = x_{k-1} - q_k \cdot x_k$$

$$y_{k+1} = y_{k-1} - q_k \cdot y_k$$

Algoritmul se oprește pentru $r_{k+1} = 0$.

La oprire, avem

$$r_k = \gcd(a, M)$$

$$x_k = x$$

$$y_k = y$$

Demonstrația se face prin inducție.

Aplicăm algoritmul pentru exemplul nostru:

i	q_i	r_i	x_i	y_i
0	-	4	1	0
1	-	19	0	1
2	0	4	1	0
3	4	3	-4	1
4	1	1	5	-1
5	-	0	-	-

Am obținut că $x = 5$ este inversul nostru modular.

Observație: La terminarea algoritmului, putem obține un număr negativ. Acest lucru trebuie corectat, prin adunări succesive cu M la rezultatul obținut, până când acesta devine pozitiv.

Implementare

Avem 2 variante de implementare, una iterativă și una recursivă.

Varianta iterativă

```
int y0 = 0, y1 = 1;
int aux = M;
while (a != 0)
{
        r = M % a;
        c = M / a;
        M = a;
        a = r;
```

```
        y = y0 - c * y1;
        y0 = y1;
        y1 = y;
}
while (y0 < 0)
{
        y0 += aux;
}
cout << y0;
```

Varianta recursivă

```
void euclid_extins (int x, int y, int a, int b)
{
        if(!b)
{
        x = 1;
        y = 0;
}
else
{
        euclid_extins (x, y, b, a % b);
        int aux = x;
        x = y;
        y = aux - y * (a / b);
}
}

int inv, ins;
euclid_extins(inv, ins, a, M);
if (inv <= 0)
        inv = M + inv % M;
cout << inv;
```

Complexitatea algoritmului necesită o discuție mai amplă

- Cazul cel mai defavorabil
  - $a = F_n, b = F_{n+1}$
  - $O(5 \cdot \lg(\min(a, b)))$
- Complexitatea medie : $O(log_2(\min(a, b)))$

Probleme propuse ca exercițiu

https://infoarena.ro/problema/inversmodular
https://infoarena.ro/problema/euclid3

https://infoarena.ro/problema/sandokan
https://infoarena.ro/problema/nmult
https://infoarena.ro/problema/functii
https://infoarena.ro/problema/jap2

3. Bibliografie

- Centrul InfO(1)
- https://infoarena.ro/
- https://ro.wikipedia.org/
- https://www.geeksforgeeks.org/