

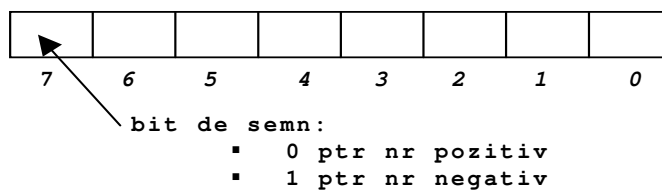
Operații pe biți

1. Reprezentarea internă a numerelor întregi

Reprezentarea în memorie a numerelor întregi se face printr-o secvență de cifre de **0** și **1**. Această secvență poate avea o lungime de **8**, **16** sau **32** de biți.

Forma de memorare a întregilor se numește *cod complementar*. În funcție de lungimea reprezentării se stabilește domeniul valorilor care pot fi stocate.

Modulul de reprezentare (în cod complementar) va fi prezentat în cele ce urmează, folosind reprezentarea pe o lungime de **8** biți, valabilă pentru tipul `shortint` în Pascal, respectiv `char` în C/C++.



Se observă că numerotarea pozițiilor se face de la dreapta la stânga (de la **0** la **7**), poziția **7** fiind bit de semn (**0** pentru numerele pozitive și **1** pentru numerele negative).

Rezultă că doar **7** biți (pozițiile **0–6**) se folosesc pentru reprezentarea valorii absolute a numărului.

Numerele întregi pozitive se convertesc în baza **2**, și se face completarea cu cifre **0** ne semnificative, până la completarea celor **7** biți.

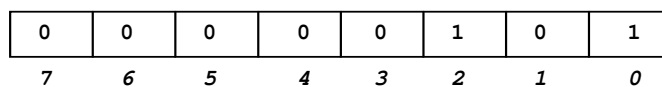
Exemplu:

Să determinăm forma de reprezentare a numărului întreg **5**.

$$5_{10} = 101_2$$

Vor fi necesare **4** cifre de **0** ne semnificative, pentru completarea primelor **7** biți, iar poziția **7** (bitul **8**) va fi **0** deoarece numărul este pozitiv.

Deci reprezentarea internă este următoarea:



Nu în același mod se face reprezentarea numerelor întregi negative. Pentru aceasta este necesară efectuarea următorilor pași:

- 1) determinarea reprezentării interne a numărului ce reprezintă valoarea absolută a numărului inițial. Acesta are bitul de semn egal cu **0**.
- 2) se calculează complementul față de **1** a reprezentării obținute la pasul anterior (bitul **1** devine **0**, iar bitul **0** devine **1**)
- 3) se adună **1** (adunarea se face în baza **2**) la valoarea obținută.

Exemplu:

Pentru determinarea reprezentării numărului **-5** se procedează în felul următor:

Reprezentarea valorii absolute a numărului **-5** este **0000101**. Complementul față de **1** este: **11111010**

Numărul obținut după adunarea cu 1 este :

11111011

Deci reprezentarea valorii întregi **-5** pe **8** biți este:

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 1 | 1 | 1 | 1 | 1 | 0 | 1 | 1 |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

După cum observăm bitul de semn este **1** ceea ce ne indică faptul că avem de a face cu un număr negativ.

Putem trage concluzia că numerele întregi care se poate reprezenta pe **8** biți sunt cuprinse între **1000000₂** și **0111111₂**, adică **-128₁₀**, **127₁₀**.

Așa cum am spus și la începutul cursului, numerele întregi se pot reprezenta în cod complementar, având la dispoziție **16**, **32** sau chiar și **64** de biți. Mecanismul este același, însă valorile numerelor cresc.

2. Operatori la nivel de bit

Operatorii pe biți se pot aplica datelor ce fac parte din tipurile întregi. Operațiile se efectuează asupra biților din reprezentarea internă a numerelor.

Operatorul de negație

not (Pascal) respectiv **~** (C/C++)

Este un operator unar care întoarce numărul întreg a cărui reprezentare internă se obține din reprezentarea internă a numărului inițial, prin complementarea față de **1** a fiecărui bit(**1→0** și **0→1**).

Exemplu:

$$\text{not } 5 == -6 \quad | \quad \sim 5 == -6$$

Reprezentarea lui **5** este **00000101**. Complementând acest număr obținem reprezentarea **11111010** care corespunde numărului întreg **-6**. Să verificăm:

Numărul **6** se reprezintă intern astfel:

6₂ = 00000110, complementând obținem **11111001** și adunând **1** (în baza **2**) se obține în final **11111010** adică exact reprezentarea internă returnată a numărului returnat de operația **not 5** respectiv **~5**.

Operatorul de conjuncție

and respectiv **&**

Este un operator binar care returnează numărul întreg a cărui reprezentare internă se obține prin conjuncția biților care apar în reprezentarea internă a operanzilor. Conjuncția se face cu toate perechile de biți situați pe aceeași poziție.

Exemplu:

$$5 \text{ and } 3 = 1 \quad | \quad 5 \ \& \ 3 == 1$$

Să verificăm:

Reprezentarea internă a lui **5** este **00000101**, iar a lui **3** este **00000011**.

| | |
|---|---|
| 0000101 and 0000011 <hr style="width: 100%;"/> 0000001 | 0000101 & 0000011 <hr style="width: 100%;"/> 0000001 |
|---|---|

Această reprezentare este dată de numărul întreg **1**.

Operatorul de disjuncție

or respectiv **|**

Este un operator binar care returnează numărul întreg a cărui reprezentare internă se obține prin disjuncția biților care apar în reprezentarea internă a operandilor. Disjuncția se face între biții situați pe aceeași poziție.

Exemplu:

| | |
|-------------------|-------------------|
| 15 or 3=15 | 15 3==15 |
|-------------------|-------------------|

Să verificăm:

Reprezentarea internă a lui **15** este **00001111** iar a lui **3** este **00000011**.

| | |
|---|--|
| 00001111 or 00000011 <hr style="width: 100%;"/> 00001111 | 00001111 00000011 <hr style="width: 100%;"/> 00001111 |
|---|--|

Această reprezentare este dată de numărul întreg **15**.

Operatorul “sau exclusiv”

xor respectiv **^**

Este un operator binar care returnează numărul întreg a cărui reprezentare internă se obține prin operația or exclusiv asupra biților care apar în reprezentarea internă a operandilor. Operația se face între biții situați pe aceeași poziție.

Exemplu:

| | |
|--------------------|------------------|
| 15 xor 3=12 | 15 ^ 3=12 |
|--------------------|------------------|

Să verificăm:

Reprezentarea internă a lui **15** este **00001111** iar a lui **3** este **00000011**.

| | |
|--|--|
| 00001111 xor 00000011 <hr style="width: 100%;"/> 00001100 | 00001111 ^ 00000011 <hr style="width: 100%;"/> 00001100 |
|--|--|

Această reprezentare este dată de numărul întreg **12**.

Operatorul shift left

shl respectiv **<<**

Este un operator binar care returnează numărul întreg a cărui reprezentare este obținută din reprezentarea internă a primului operand prin deplasare la stânga cu un număr de biți egal cu al doilea operand.

Exemplu:

$$4 \text{ shl } 2=16 \quad | \quad 4 \ll 2==16$$

Să verificăm:

Reprezentarea internă a lui 4 este 00000100. Prin deplasare la stânga cu doi biți se obține 00010000, care este reprezentarea internă a lui 16.

Acest operator poate fi folosit pentru calculul numerelor întregi de forma 2ⁿ prin efectuarea operațiilor de forma 1 shl n. (1<<n)

Operatorul shift right

shr respectiv >>

Este un operator binar care returnează numărul întreg a cărui reprezentare este obținută din reprezentarea internă a primului operand prin deplasare la dreapta cu un număr de biți egal cu al doilea operand.

Prin deplasarea la dreapta, primii biți din reprezentarea internă a numărului (pozițiile 1,2, ș.a.m.d.) se pierd iar ultimii se completează cu zero.

Exemplu:

$$14 \text{ shr } 2=3 \quad | \quad 14 \gg 2==3$$

Să verificăm:

Reprezentara internă a numărului 14 este 000001110. Prin deplasare la dreapta cu doi biți a reprezentării lui 14 se obține 00000011, care este reprezentarea internă a lui 3.

Operația n shr 1 (n>>1) este echivalentă cu împărțirea întregă la 2.

3. Operații la nivel de biți

Transformarea unui bit în 1

Pornim de la valoarea întregă X=50. Reprezentarea acestuia pe 8 biți este 00110010. Presupunem că dorim setarea bitului 2 la valoarea 1. Pentru aceasta vom folosi o mască logică în care doar bitul 2 este 1 restul biților fiind 0, adică M=00000100. Valoarea lui M este 1 shl 2 (2²). Operația de disjuncție SAU aplicată asupra lui X și a lui M, conduce la obținerea rezultatului dorit.

| | | | | | | |
|-----|-------------------|----|--|-----|-------------------|--|
| X | 00110010 | OR | | X | 00110010 | |
| M | 00000100 | | | M | 00000100 | |
| Rez | 00110 <u>1</u> 10 | | | Rez | 00110 <u>1</u> 10 | |

Generalizând, dacă se dorește ca valorii X, să i se seteze la valoarea 1, bitul B (0≤B≤7), atunci masca logică este 1 shl B.

$$X \text{ or } (1 \text{ shl } B) \quad | \quad X | (1 \ll B)$$

Transformarea unui bit în 0

Să luăm ca exemplu X= 109, pentru a vedea cum se setează un bit la valoarea 0. Reprezentarea internă a lui este 01101101. Se cere să se seteze bitul 5 la valoarea 0. De data aceasta masca va conține toți biții de 1, excepție bitul 5. Aspura lui X și M vom aplica ȘI logic.

```

X   01101101 AND
M   11011111
-----
Rez 01001101

```

```

X   01101101 &
M   11011111
-----
Rez 01001101

```

Presupunem că dorim să setăm la **0** valoarea bitului **B** ($0 \leq B \leq 7$). Pentru determinarea valorii măștii **M**, plecăm de la valoarea **255** din care scădem **1** shl **B**.

X and (255 - (1 shl B))

X & (255 - (1 << B))

Masca mai poate fi obținută și aplicând sau exclusiv în locul scăderii, **255 xor (1 shl B)**. Rescriem rezultatul sub forma:

X and (255 xor (1 shl B))

X & (255 ^ (1 << B))

Testarea valorii unui bit

Plecăm de la valoarea **X=47**. Reprezentarea internă a lui este **00101111**. Presupunem că dorim să cunoaștem valoarea bitului **3** și bitului **6**. Vom folosi măștile $M_1=00001000$ și $M_2=01000000$. Vom aplica de fiecare dată **ȘI** logic între **X** și cele două măști:

```

X   00101111 AND
M1  00001000
-----
Rez 00001000

```

```

X   00101111 &
M1  00001000
-----
Rez 00001000

```

Respectiv

```

X   00101111 AND
M2  01000000
-----
Rez 00000000

```

```

X   00101111 &
M2  01000000
-----
Rez 00000000

```

Generalizând, testarea se va realiza prin :

X and (1 shl B) <> 0

X & (1 << B) != 0

Testarea valorilor ultimilor biți

Pornim de la valoarea întreagă **X=50**. Reprezentarea acestuia pe **8** biți este **00110010**. Presupunem că dorim să cunoaștem restul la împărțirea întreagă a lui **X** la **8**, adică **X mod 8** respectiv **x%8**. Valoarea ultimilor **3** biți din reprezentarea internă a lui **X**, reprezintă tocmai acest rest. Pentru aceasta vom folosi o mască în care ultimii trei biți sunt **1** restul **0**. Aceasta mască are valoarea **7**, adică **00000111**. Vom aplica operația **ȘI** logic.

```

X   00110010 AND
M   00000111
-----
Rez 00000010

```

```

X   00110010 &
M   00000111
-----
Rez 00000010

```

Pe caz general, dacă dorim să cunoaștem valoarea ultimilor **B** biți (care este egal cu restul împărțirii lui **X** la 2^B) vom exprima astfel:

X and (1 shl B - 1)

X & (1 << B - 1)

Aplicații

Problema 1.

Realizați un subprogram care determină numărul de cifre de **1** din reprezentarea binară a unui număr natural nenul **n** mai mic ca **2000000000**.

Soluția 1:

Numărul îl vom reține într-o variabilă **longint** respectiv **long** și vom parcurge secvențial biții lui.

```
function nr( n:longint):byte;
var nm,i:byte;
begin
  nm:=0;
  for i:=0 to 31 do
    if n and (1 shl i) <>0 then
      inc(nm);
  nr:=nm;
end;
```

```
int nr (long n){
  int nm=0;
  for ( int i=0; i<32; i++)
    if (n & (1 << i)) nm++;
  return nm;
}
```

Soluția 2:

O altă soluție, mai rapidă este folosirea operației **SI** logic între valorile lui **n** și a lui **n-1**.

Această operație anulează cel mai nesemnificativ bit de **1** a lui **n**. Să luăm ca exemplu valoarea lui **n**:

```
n          = (110011101000100)2
n-1        = (110011101000011)2
n and (n-1) = (110011101000000)2
```

```
n          = (110011101000100)2
n-1        = (110011101000011)2
n & (n-1)  = (110011101000000)2
```

De aici și ideea algoritmului următor:

```
function nr( n:longint):byte;
var nm:byte;
begin
  nm:=0;
  repeat
    n:=n and(n-1); inc(nm);
  until n = 0;
  nr:=nm;
end;
```

```
int nr (long n){
  int nm=0;
  do {
    n &= n-1;
    nm++;}
  while (n);
  return nm;
}
```

Rezultatul este mai bun la cea de a doua metodă deoarece execută un număr de pași egali cu numărul de cifre de **1** din reprezentare.

Problema 2.

Realizați un subprogram care verifică dacă un număr natural nenul **n** este o putere a lui **2**.

Soluție:

Dacă **n** este o putere a lui **2**, va avea o singură cifră de **1** în reprezentarea binară. Dacă ne bazăm pe observația de la problema anterioară, obținem:

```
function pow2( n:longint):boolean;
begin
  pow2 := n and(n-1) = 0;
end;
```

```
long pow2 (long n)
{
  return (n & (n-1)==0);
}
```

Problema 3.

Realizați un subprogram care identifică cea mai mare putere a lui **2** care îl divide pe **n**, număr natural nenul.

Soluția 1:

Problema se reduce la determinarea celui mai nesemnificativ bit de **1** din reprezentarea binară a lui **n**. Prima soluție caută poziția, pornind cu bitul **0**.

```
function pow(n:longint):longint;
var i:integer;
begin
  i:=0;
  while n and (1 shl i)=0 do inc(i);
  pow := 1 shl i;
end;
```

```
long pow (long n){
  int i=0;
  while (!(n & (1<<i))) i++;
  return 1 << i ;
}
```

Soluția 2

Se bazează pe observația că **n and (n-1)** are ca rezultat numărul din care lipsește cel mai nesemnificativ bit de **0**. Atunci aplicând **XOR** între valoarea inițială a lui **n** și ce a lui **n and (n-1)** vom obține valoarea cerută.

Exemplu:

```
n = (101000100)2
n and (n-1) = (101000000)2
n xor (n and (n-1)) = (000000100)2
```

```
n = (101000100)2
n & (n-1) = (101000000)2
n ^ ( n & (n-1)) = (000000100)2
```

```
function pow( n:longint):longint;
begin
  pow := n xor (n and (n-1));
end;
```

```
long pow (long n){
  return n^(n & (n-1)) ;
}
```

Problema 4.

Realizați un subprogram care identifică numărul de ordine al celui mai semnificativ bit de **1** al lui **n** număr natural nenul.

Soluție

Să luăm ca exemplu următorul șir de operații:

```
n=(10000000)2
n=n or (n shr 1)
n=(11000000)2
n=n or (n shr 2)
n=(11110000)2
n=n or (n shr 4)
n=(11111111)2
```

```
n=(10000000)2
n=n | (n >> 1)
n=(11000000)2
n=n | (n >> 2)
n=(11110000)2
n=n | (n >> 4)
n=(11111111)2
```

Se observă că aplicând o secvență asemănătoare de instrucțiuni putem transforma un număr **n** în alt număr în care numărul de biți de **1** este egal cu **1 + indexul celui mai semnificativ bit de 1**.

```

function index( n:longint):byte;
begin
  n := n or (n shr 1);
  n := n or (n shr 2);
  n := n or (n shr 4);
  n := n or (n shr 8);
  n := n or (n shr 16);
  index:= nr(n)-1;
end;

```

```

int index(long n){
  n = n | (n >> 1);
  n = n | (n >> 2);
  n = n | (n >> 4);
  n = n | (n >> 8);
  n = n | (n >> 16);
  return nr(n) - 1;
}

```

Problema 5

Gigel trebuie să cumpere n medicamente, numerotate de la 1 la n . Doctorul i-a dat m rețete de două tipuri, codificate cu numerele 1, 2 astfel:

- 1 - rețetă necompensată, adică prețul medicamentelor de pe rețetă se achită integral de către cumpărător;
- 2 - rețetă compensată **50%**, adică prețul medicamentelor înscrise pe rețetă se înjumătățește.

Se știe că pe rețete nu există un alt medicament decât cele numeroatete de la 1 la n și o rețetă nu conține două medicamente identice.

Dacă o rețetă este folosită atunci se vor cumpăra toate medicamentele înscrise pe ea.

Cerință

Scrieți un program care să determine suma minimă de bani necesară pentru a cumpăra exact câte unul din fiecare dintre cele n medicamente, folosindu-se de rețetele avute la dispoziție.

Date de intrare

Fișierul de intrare `reteta.in` are următorul format :

- pe prima linie sunt scrise numerele naturale n și m ;
- pe următoarele m linii sunt descrise cele m rețete, câte o rețetă pe o linie. Linia care descrie o rețetă conține tipul rețetei (1 necompensată sau 2 compensată), urmat de un număr natural q reprezentând numărul de medicamente de pe rețetă, apoi de q numere distincte din mulțimea $\{1, 2, \dots, n\}$ reprezentând medicamentele înscrise pe acea rețetă;
- pe ultima linie a fișierului de intrare sunt scrise n numere naturale separate prin câte un spațiu, reprezentând în ordinea de la 1 la n , prețul medicamentelor.

Toate numerele de pe aceeași linie sunt separate prin câte un spațiu.

Date de ieșire

Fișierul de ieșire `reteta.out` va conține o singură linie pe care va fi scris un număr real cu o singură zecimală, reprezentând suma minimă determinată.

Restricții

$$1 \leq N \leq 20$$

$$1 \leq M \leq 15$$

$$1 \leq \text{preț} \text{ oricărui medicament} \leq 200$$

Pentru datele de test există întotdeauna soluție.

Exemplu

| reteta.in | reteta.out | Explicație |
|---|------------|---|
| 4 5 2 1 3 2 2 2 3 1 1 1 1 3 4 1 2 1 1 3 8 20 2 16 | 45.0 | Soluția s-a obținut prin folosirea primei și celei de a patra rețete. O altă soluție, dar de cost mai mare, s-ar fi obținut dacă se folosea rețeta a patra și cea de a cincea. |

Timp maxim de execuție/test: 1 secundă

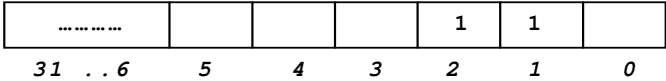
Soluție

Problema admite o soluție de complexitate exponențială $O(2^m)$. Se generează fiecare submulțime a mulțimii rețetelor. Se identifică acele submulțimi pentru care medicamentele înscrise pe rețete includ toate cele n medicamente, fără să repete vreunul.

Se va păstra ca soluție submulțimea de rețete care are cost total minim. În continuare vă este prezentată o rezolvare folosind lucru pe biți.

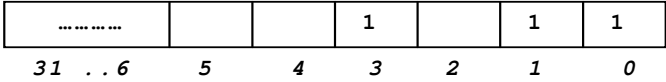
O rețetă se va codifica cu ajutorul unui întreg (**longint-long**). Fiecare bit x ($0 \leq x \leq 19$) va codifica prezenta-absența medicamentului $x+1$ de pe rețetă. Cum există m rețete, vectorul L va codifica prin elementul $L[i]$ rețeta $i+1$, $0 \leq i \leq m-1$.

Plecând de la exemplu din enunț, a doua rețetă ce conține medicamentele **2** și **3** se va codifica prin elementul $L[1]$ și va avea valoarea **6**.



În același mod, pentru generarea submulțimilor de rețete ce vor fi folosite, ne vom folosi de vectorii caracteristici asociați acestora. Pentru aceasta vom utiliza reprezentarea binară a tuturor numerelor de la 0 la 2^m-1 .

Dacă variabila s are valoarea **11** atunci ea cuprinde prima, a doua și a patra rețetă. Medicamentele pe care acestea le conțin sunt codificate în $L[0]$, $L[1]$ și $L[3]$.



Variabila u va reprezenta o mască logică care indică medicamentele prinse pe rețetele ce aparțin submulțimii curente.

```

const INF=2000000000;
var L,T,C:array[0..20]of longint;
    P:array[0..20]of real;
    n,m,i,j,k,nr_med,x,s,u:longint;
    ok:boolean;
    sum,min:real;
    f:text;
begin
    assign( f, 'reteta.in');reset(f);
    read(f,n,m);
    for i:=0 to m-1 do begin
        read(f, T[i], nr_med );
        for j:=1 to nr_med do begin
            read(f, x);
            L[i]:=L[i] or (1 shl(x-1));
        end;
    end;
    for i:=0 to n-1 do read(f, C[i]);
    close ( f );

```

```

#include <stdio.h>
#include <string.h>

long L[20],T[20],C[20],n,m,i,j;
long k,nr_med,x,s,u,corect;
double P[20],sum,min;

int main (){

    freopen ( "reteta.in" , "r" , stdin );
    scanf ( "%d %d" , &n , &m );
    for ( i=0 ; i<m ; i++ ) {
        scanf ( "%d %d" , &T[i] , &nr_med );
        for ( j=0 ; j<nr_med ; j++ ) {
            scanf ( "%d" , &x );
            L[i] |= 1<<(x-1);
        }
    }
    for (i=0;i<n;i++) scanf ("%d",&C[i]);
    fclose ( stdin );

```

```

for i:=0 to m-1 do begin
  sum := 0;
  for j:=0 to n-1 do
    sum:=sum+((L[i] shr j) and 1)*C[j];
  P[i]:=sum/T[i];
end;

min:=INF;
for s:=0 to 1 shl m -1 do begin
  sum:=0; u := 0; ok:=true; i:=0;
  while (i<m) and ok do begin
    if (s shr i) and 1 = 1 then
      if (u and L[i]<>0 then
        ok:=false
      else begin
        sum:=sum+P[i];
        u:=u or L[i];
      end;
      inc(i);
    end;
    if ok and (sum<min) and
      (u=( 1 shl n)-1) then min:=sum;
  end;

assign(f,'reteta.out'); rewrite(f);
if (INF-min>0.1)then
  writeln (f, min:0:1)
else
  writeln(f,'imposibil');
close ( f );
end.

```

```

for ( i=0 ; i<m ; i++ ) {
  sum = 0;
  for ( j=0 ; j<n ; j++ )
    sum += ((L[i]>>j)&1) * C[j];
  P[i]=sum/T[i];
}
int pp=0;

min=2000000000;
for (s=0 ; s<=(1<<m)-1 ; s++)
{
  sum=0; u = 0; corect=1;
  for (i=0 ; i<m && corect ; i++ )
    if ((s>>i)&1)
      if (u&L[i]) corect=0;
    else
    {
      sum += P[i];
      u |= L[i];
    }
}

if (corect && sum<min && u==(1<<n)-1)
  min=sum;

freopen ( "reteta.out","w",stdout);
if (min!=2000000000)
  printf ( "%.11f\n" , min );
else printf ( "imposibil\n" );
fclose ( stdout );
return 0;
}

```

Problema 6

Presupunem că avem n numere prime notate a_1, a_2, \dots, a_n sortate strict crescător. Formăm un șir strict crescător b ale cărui elemente sunt toți multiplii acestor n numere prime astfel încât, multipli comuni apar o singură dată. Presupunem că numerotarea pozițiilor elementelor din șirul b începe tot cu 1.

Cerință

Scrieți un program care citește din fișierul de intrare valoarea lui n și apoi cele n elemente ale șirului a , determină elementul de pe poziția m din șirul b și afișează în fișierul de ieșire valoarea acestuia.

Date de intrare

Fișierul de intrare **numar.in** conține

- pe prima linie două numere naturale separate printr-un spațiu care reprezintă primul valoarea lui n și al doilea valoarea lui m ;
- pe a doua linie n numere naturale prime separate prin câte un spațiu care reprezintă valorile elementelor șirului a . Aceste valori sunt dispuse în ordine strict crescătoare iar ultima dintre ele este mai mică decât un milion.

Date de ieșire

Fișierul de ieșire **numar.out** va conține pe prima linie o singură valoare care reprezintă termenul de pe poziția m din șirul b .

Restricții și precizări

- $a_n < 1000000$

Exemple:

| numar.in | numar.out | Explicații |
|--------------------------|-----------|---|
| 3 10 2 3 5 | 14 | Șirul b e format din valorile: 2, 3, 4, 5, 6, 8, 9, 10, 12, 14, 15, 16, 18, 20, 21, 22... Pe poziția 10 se află numărul 14 |
| 4 20 7 23 37 131 | 98 | |
| 3 11111 977 1009 1031 | 3726237 | |

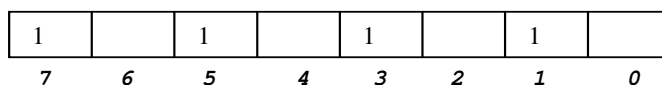
Timp maxim de executare/test: 1 secundă.

Soluție

Soluția următoare nu reprezintă soluția oficială a problemei. Însă, datorită optimizării memoriei reușește să obțină un punctaj foarte bun.

Fiecare multiplu este marcat prin intermediul unui bit. În loc să ne folosim de un vector de booleene, sau 0-1 ca să însemnăm un număr ca fiind multiplu al unei valori din cele citite, ne vom folosi de un vector de **byte-unsigned char**, în care vor fi setați fiecare din cei 8 biți ai fiecărei valori.

Astfel, se va marca fiecare multiplu al valorilor citite. Să luăm ca exemplu valoarea 2. Trebuie marcate valorile multiplu de 2, deci 2, 4, 6, 8, 10, 12 ș.a.m.d. În primul element al vectorului v , vor marca primii 4 multipli, deci $v[0]=170$



Procedeeul continuă până sunt marcați toți multipli numerelor citite. Dacă dorim să setăm cu 1 bitul corespunzător valorii x , atunci va trebui ca în cadrul elementului $v[(x-1)\text{div } 8]$ să marcăm bitul $(x-1)\text{mod } 8$.

După procesul de marcare, urmează regăsirea multiplului cu numărul de ordine m , care va fi identificat prin traversarea secvențială a biților elementelor vectorului v .

```

program numar;
var f,g:text;
    a:array[0..200] of longint;
    v:array[0..60000] of byte;
    x,i,j,nr,n,y,m:longint;
begin
assign(f,'numar.in'); reset(f);
assign(g,'numar.out'); rewrite(g);
readln(f,n,m);
for i:=1 to n do read(f,a[i]);

for i:=1 to n do begin
x:=0; j:=1;
while (j<m)and(x<400000) do begin
x:=x+a[i];
v[(x-1) shr 3]:=
v[(x-1) shr 3]or(1 shl((x-1)and 7));
inc(j);
end;
end;

nr:=0; i:=0;
while nr<m do begin
for j:=0 to 7 do begin
x:=1 shl j;
if (v[i]shr j) and 1=1 then
begin
inc(nr);
if nr=m then y:=i*8 + j + 1;
end;
end;
inc(i);
end;
writeln(g,y);
close(g); close(f);
end.

```

```

#include <stdio.h>

int a[201];
unsigned char v[60001];
long x,i,j,nr,n,y,m;

int main(){
freopen("numar.in","r",stdin);
freopen("numar.out","w",stdout);
scanf("%ld%ld",&n,&m);
for (i=1; i<=n; i++)
scanf("%ld",&a[i]);

for (i=1; i<=n; i++) {
x:=0; j:=1;
while ((j<m)&&(x<400000)) {
x += a[i];
v[(x-1) >> 3]=
v[(x-1) >> 3] | (1 << ((x-1) & 7));
j++;
}
}

nr:=0; i:=0;
while (nr<m) {
for (j=0;j<=7;j++) {
if (((v[i] >> j) & 1)==1) {
nr++;
if (nr==m) y=i*8 + j + 1;
}
}
i++;
}
printf("%ld", y);
fclose(stdout); fclose(stdin);
return 0;
}

```