Skrypt

19 lutego 2022

Spis treści

1 Notatki

1.1 Teoria liczb

1.1.1 Algorytm Euklidesa

Dany jest następujący problem:

WEJŚCIE:	liczby całkowite dodatnie a i b
WYJŚCIE:	największy wspólny dzielnik liczb a i b - największa taka liczba całkowita c, że $c\|a$ oraz $c\|b$

Można go rozwiązać *algorytmem Euklidesa*:

```
1.  NWD( a, b )
2.     if a < b
3.        SWAP( a, b )
4.     while b > 0
5.        a := a − b
6.        if a < b
7.           SWAP( a, b )
8.     return a
```

Aby udowodnić poprawność algorytmu, potrzebne będą następujące obserwacje:

Lemat 1.1 *Jeżeli $d|a$ oraz $d|b$, to $d|a-b$.*

Dowód: Jeżeli $d|a$, to $\exists_{k\in\mathbb{Z}}\ a = dk$. Analogicznie $\exists_{l\in\mathbb{Z}}\ b = dl$. Wówczas $a - b = d(k-l)$, gdzie $k - l \in \mathbb{Z}$, zatem $d|a-b$. □

Lemat 1.2 *Jeżeli $d|a-b$ oraz $d|b$, to $d|a$.*

Dowód: Jeżeli $d|a-b$, to $\exists_{k\in\mathbb{Z}}\ a - b = dk$. Analogicznie $\exists_{l\in\mathbb{Z}}\ b = dl$. Wówczas $a - b + b = a = d(k+l)$, gdzie $k + l \in \mathbb{Z}$, zatem $d|a$. □

Poprawność algorytmu można udowodnić korzystając z niezmienników pętli 4. - 7. :
1) $a \geq b$
2) $NWD(x,y)|a$ oraz $NWD(x,y)|b$
3) $NWD(a,b)|x$ oraz $NWD(a,b)|y$
gdzie x i y są początkowymi wartościami a oraz b przekazanymi jako argumenty funkcji NWD. Niezmiennik 1) jest zapewniony przez instrukcje 2. - 3. oraz 6. - 7. (jeżeli porządek a i b się zmieni, to wartości zmiennych zostaną zamienione), niezmiennik 2) jest konsekwencją lematu 1.1 i niezmiennik 3) jest konsekwencją lematu 1.2 (zakładając nie wprost, że $NWD(a,b)|x$ i $NWD(a,b)|y$ oraz $r = NWD(a-b,b)$ nie dzieli x (lub y) otrzymuje się $r|a$ (z lematu 1.2) co w połączeniu z $r|b$ daje $r|NWD(a,b)$, a ponieważ $NWD(a,b)|x$, to również $r|x$ - sprzeczność). Zatem po zakończeniu pętli 4. - 7. zachodzi $NWD(a,b) = NWD(a,0) = a$, czyli (na mocy niezmiennika 3)) $a|NWD(x,y)$, czyli $a \leq NWD(x,y)$ oraz (na mocy niezmiennika 2)) $NWD(x,y)|a$, czyli $a \geq NWD(x,y)$, czyli ostatecznie $a = NWD(x,y)$.

Łatwo pokazać, że algorytm działa w czasie $O(x+y)$ - z każdą iteracją pętli zmienna a zmniejsza się o co najmniej 1, więc po $x+y$ iteracjach suma $a+b$ będzie równa co najwyżej $x+y-(x+y) = 0$, więc algorytm na pewno skończy działanie szybciej. Można też pokazać, że w pesymistycznym przypadku faktycznie złożoność będzie liniowa (na przykład dla $x = 1$

pętla wykona dokładnie y iteracji).

Algorytm można w łatwy sposób bardzo przyspieszyć - jeżeli zamiast odejmować, będzie się brało resztę z dzielenia (jeżeli od większej liczby odejmuje się mniejszą tak długo, aż ta pierwsza stanie się w końcu mniejsza od drugiej, to takie działanie odpowiada braniu reszty z dzielenia). Taki algorytm działa w czasie logarytmicznym $O(\ln(max\{a,b\}))$ - jeżeli $a \geq b$, to wówczas $\exists_{k\in\mathbb{Z},r\in\{0,1,2,\ldots,b-1\}} a = kb + r$ i wówczas $2r < r + b \leq a$, czyli $a\%b = r < \frac{a}{2}$, czyli po dwóch iteracjach zmienne a i b przyjmą wartości $a\%b < \frac{a}{2}$ oraz $b\%(a\%b)$ - czyli po $2\log_2(a)$ iteracjach zmienna a będzie miała wartość mniejszą niż $\frac{a}{2^{\log_2(a)}} = \frac{a}{a} = 1$, a to jest niemożliwe (bo $a > b \geq 0$), więc algorytm na pewno skończy działanie szybciej. W szybszej wersji algorytmu ponadto nie trzeba w pętli sprawdzać, czy $a < b$, bo $a\%b$ zawsze jest mniejsze od b - zatem można od razu zamienić zmienne wartościami:

```
1.  NWD( a, b )
2.      if a < b
3.          SWAP( a, b )
4.      while b > 0
5.          a := a%b
6.          SWAP( a, b )
7.      return a
```

1.1.2 Rozszerzony algorytm Euklidesa

Algorytm Euklidesa można wykorzystać do wyznaczenia takich liczb całkowitych x i y, że $ax + by = NWD(a,b)$ $(a,b \in \mathbb{Z})$. Kluczowa jest tutaj obserwacja: jeżeli $k = ax_1 + by_1$ oraz $l = ax_2 + by_2$, gdzie $x_1, x_2, y_1, y_2 \in \mathbb{Z}$, to $t = k - l = ax_3 + by_3$, gdzie $x_3 = x_1 - x_2$ oraz $y_3 = y_1 - y_2$, czyli $x_3, y_3 \in \mathbb{Z}$. Podobnie dla dzielenia modulo: $r = k\%l = ax_4 + by_4$, gdzie $x_4 = x_1 - x_2\lfloor\frac{k}{l}\rfloor$ i $y_4 = y_1 - y_2\lfloor\frac{k}{l}\rfloor$, czyli $x_4, y_4 \in \mathbb{Z}$. Wystarczy zatem zmodyfikować algorytm tak, by cały czas pamiętał obie pary liczb (początkowo są to pary $(1,0)$ oraz $(0,1)$).

```
1.  RNWD( a, b )
2.      x1 := 1
3.      y1 := 0
4.      x2 := 0
5.      y2 := 1
6.      if a < b
7.          SWAP( a, b )
8.          SWAP( x1, x2 )
9.          SWAP( y1, y2 )
10.     while b > 0
11.         x1 := x1 − x2 × (a/b)
12.         y1 := y1 − y2 × (a/b)
13.         a := a%b
14.         SWAP( a, b )
15.         SWAP( x1, x2 )
16.         SWAP( y1, y2 )
17.     return (x1, y1)
```

1.1.3 Najmniejsza wspólna wielokrotność

Najmniejszą wspólną wielokrotność dwóch liczb a i b można obliczyć korzystając z faktu, że $NWW(a,b) = \frac{ab}{NWD(a,b)}$. Aby to udowodnić, można skorzystać z poniższych obserwacji:

Twierdzenie 1.3 *Rozkład liczby naturalnej na czynniki pierwsze jest zadany jednoznacznie.*

Dowód: Niech $\mathcal{A}$ będzie zbiorem liczb naturalnych o kilku różnych rozkładach na czynniki pierwsze. Niech $a = min(\mathcal{A})$. Niech ${q_1}^{k_1}{q_2}^{k_2}{q_3}^{k_3}\dots{q_n}^{k_n}$ oraz ${p_1}^{l_1}{p_2}^{l_2}{p_3}^{l_3}\dots{p_m}^{l_m}$ będą różnymi rozkładami liczby a na czynniki pierwsze ($q_i \in \mathbb{P}$ oraz $p_j \in \mathbb{P}$ dla $1 \leq i \leq n$ oraz $1 \leq j \leq m$, gdzie $\mathbb{P}$ jest zbiorem liczb pierwszych). Wówczas dla dowolnych $i \in \{1,2,3,\dots,n\}$ oraz $j \in \{1,2,3,\dots,m\}$ zachodzi $q_i \neq p_j$ - w przeciwnym wypadku istniałyby takie u i r, że $q_u = p_r$, i wówczas liczba a/q_u miałaby dwa różne rozkłady na czynniki pierwsze i byłaby mniejsza od a, co byłoby sprzeczne z tym, że a jest minimalne. Ponieważ q_1 oraz p_1 są różnymi liczbami pierwszymi, to $NWD(q_1,p_1) = 1$, więc istnieją takie niezerowe liczby całkowite x oraz y, że $xq_1 + yp_1 = 1$ (można je wyznaczyć rozszerzonym algorytmem Euklidesa). Zatem $x{q_1}^{k_1}{q_2}^{k_2}q_3\dots{q_n}^{k_3} + yp_1{q_1}^{k_1-1}{q_2}^{k_2}{q_3}^{k_3}\dots{q_n}^{k_n} = {q_1}^{k_1-1}{q_2}^{k_2}{q_3}^{k_3}\dots{q_n}^{k_n}$ (mnoży się obustronnie przez ${q_1}^{k_1-1}{q_2}^{k_2}{q_3}^{k_3}\dots{q_n}^{k_n}$), czyli $xa + yp_1{q_1}^{k_1-1}{q_2}^{k_2}{q_3}^{k_3}\dots{q_n}^{k_n} = {q_1}^{k_1-1}{q_2}^{k_2}{q_3}^{k_3}\dots{q_n}^{k_n}$. Ponieważ $p_1|a$, to p_1 dzieli lewą stronę równości, więc dzieli także prawą stronę równości. Wówczas liczba $b = {q_1}^{k_1-1}{q_2}^{k_2}{q_3}^{k_3}\dots{q_n}^{k_n} = a/q_1$ jest podzielna przez p_1, więc ma rozkład na czynniki pierwsze zawierający liczbę p_1 (można ten rozkład wyznaczyć domnażając p_1 do rozkładu liczby b/p_1). Zachodzi $q_i \neq p_1$ dla $i \in \{1,2,3,\dots,n\}$, więc b ma dwa różne rozkłady na czynniki pierwsze i $b < a$, co przeczy minimalności a. Skoro istnienie w zbiorze $\mathcal{A}$ elementu minimalnego prowadzi do sprzeczności, to $\mathcal{A}$ musi być puste. Zatem wszystkie liczby naturalne mają jednoznaczny rozkład na czynniki pierwsze. □

Niech dane będzie oznaczenie: $n = (a_1,a_2,a_3,\dots)_p \Leftrightarrow n = {p_1}^{a_1}{p_2}^{a_2}{p_3}^{a_3}\cdots = \prod_{i=1}^{\infty} {p_i}^{a_i}$, gdzie $p_1,p_2,p_3,\dots$ są kolejnymi liczbami pierwszymi. Innymi słowy, $(a_1,a_2,a_3,\dots)_p$ to reprezentacja rozkładu liczby na czynniki pierwsze (i jest ona jednoznaczna). Przykładowo: $24 = (3,1,0,0,\dots)_p$, bo $24 = 2^3 \times 3$, oraz $49 = (0,0,0,2,0,0,\dots)_p$, bo $49 = 7^2$.

Lemat 1.4 *Jeżeli* $a = (\alpha_1,\alpha_2,\alpha_3,\dots)_p$ *oraz* $b = (\beta_1,\beta_2,\beta_3,\dots)_p$, *to:*

$$ab = (\alpha_1+\beta_1, \alpha_2+\beta_2, \alpha_3+\beta_3, \dots)_p$$

Dowód: Równość wynika z definicji oznaczenia $(\dots)_p$.

Lemat 1.5 *Jeżeli* $a = (\alpha_1,\alpha_2,\alpha_3,\dots)_p$ *oraz* $b = (\beta_1,\beta_2,\beta_3,\dots)_p$ *i* $a|b$, *to:*

$$\forall_{i\in\mathbb{N}_+}\alpha_i \leq \beta_i$$

Dowód: Skoro $a|b$ oraz a i b są dodatnie, to istnieje dodatnia liczba całkowita c taka, że $ac = b$. Niech $c = (\gamma_1,\gamma_2,\gamma_3,\dots)_p$. Wówczas $ac = (\alpha_1+\gamma_1, \alpha_2+\gamma_2, \alpha_3+\gamma_3,\dots)_p = (\beta_1,\beta_2,\beta_3,\dots)_p$ oraz $\forall_{i\in\mathbb{N}_+}\gamma_i \geq 0$, więc $\forall_{i\in\mathbb{N}_+}\alpha_i + \gamma_i \geq \alpha_i$, czyli $\forall_{i\in\mathbb{N}_+}\beta_i \geq \alpha_i$. □

Lemat 1.6 *Jeżeli* $a = (\alpha_1,\alpha_2,\alpha_3,\dots)_p$ *oraz* $b = (\beta_1,\beta_2,\beta_3,\dots)_p$ *i* $\forall_{i\in\mathbb{N}_+}\alpha_i \leq \beta_i$, *to* $a|b$.

Dowód: Niech $c = (\gamma_1,\gamma_2,\gamma_3,\dots)_p = (\beta_1-\alpha_1, \beta_2-\alpha_2, \beta_3-\alpha_3,\dots)_p$. Wówczas $\forall_{i\in\mathbb{N}_+}\gamma_i \geq 0$, czyli $c \in \mathbb{N}_+$, oraz $ca = (\beta_1-\alpha_1+\alpha_1, \beta_2-\alpha_2+\alpha_2, \beta_3-\alpha_3+\alpha_3,\dots)_p = (\beta_1,\beta_2,\beta_3,\dots)_p = b$, więc $a|b$. □

Lemat 1.7 *Jeżeli* $a = (\alpha_1, \alpha_2, \alpha_3, \dots)_p$, $b = (\beta_1, \beta_2, \beta_3, \dots)_p$, $c = NWW(a,b) = (\gamma_1, \gamma_2, \gamma_3, \dots)_p$, *to:*

$$\forall_{i \in \mathbb{N}_+} \gamma_i = max\{\alpha_i, \beta_i\}$$

Dowód: Zachodzi $a|c$ oraz $b|c$, więc $\forall_{i \in \mathbb{N}_+} \gamma_i \geq \alpha_i$ oraz $\forall_{i \in \mathbb{N}_+} \gamma_i \geq \beta_i$, czyli $\forall_{i \in \mathbb{N}_+} \gamma_i \geq max\{\alpha_i, \beta_i\}$, czyli $c \geq d = (max\{\alpha_1, \beta_1\}, max\{\alpha_2, \beta_2\}, max\{\alpha_3, \beta_3\}, \dots)_p$. Jednocześnie $\forall_{i \in \mathbb{N}_+} max\{\alpha_i, \beta_i\} \geq \alpha_i$ i $\forall_{i \in \mathbb{N}_+} max\{\alpha_i, \beta_i\} \geq \beta_i$, więc $a|d$ oraz $b|d$. Zatem d jest wielokrotnością a i b oraz jest najmniejsze możliwe (bo $NWW(a,b) \geq d$), zatem jest najmniejszą wspólną wielokrotnością. □

Lemat 1.8 *Jeżeli* $a = (\alpha_1, \alpha_2, \alpha_3, \dots)_p$, $b = (\beta_1, \beta_2, \beta_3, \dots)_p$, $c = NWD(a,b) = (\gamma_1, \gamma_2, \gamma_3, \dots)_p$, *to:*

$$\forall_{i \in \mathbb{N}_+} \gamma_i = min\{\alpha_i, \beta_i\}$$

Dowód: Zachodzi $c|a$ oraz $c|b$, więc $\forall_{i \in \mathbb{N}_+} \gamma_i \leq \alpha_i$ oraz $\forall_{i \in \mathbb{N}_+} \gamma_i \leq \beta_i$, czyli $\forall_{i \in \mathbb{N}_+} \gamma_i \leq min\{\alpha_i, \beta_i\}$, czyli $c \leq d = (min\{\alpha_1, \beta_1\}, min\{\alpha_2, \beta_2\}, min\{\alpha_3, \beta_3\}, \dots)_p$. Jednocześnie $\forall_{i \in \mathbb{N}_+} min\{\alpha_i, \beta_i\} \leq \alpha_i$ i $\forall_{i \in \mathbb{N}_+} min\{\alpha_i, \beta_i\} \leq \beta_i$, więc $d|a$ oraz $d|b$. Zatem d jest dzielnikiem a i b oraz jest największe możliwe (bo $NWD(a,b) \leq d$), zatem jest największym wspólnym dzielnikiem. □

Lemat 1.9 *Dla dowolnych* $a, b \in \mathbb{N}_+$ *zachodzi* $NWW(a,b) = \frac{ab}{NWD(a,b)}$.

Dowód: Wystarczy pokazać, że $NWW(a,b) \times NWD(a,b) = ab$. Zachodzi:

$$NWW(a,b) = (max\{\alpha_1, \beta_1\}, max\{\alpha_2, \beta_2\}, max\{\alpha_3, \beta_3\}, \dots)_p$$

$$NWD(a,b) = (min\{\alpha_1, \beta_1\}, min\{\alpha_2, \beta_2\}, min\{\alpha_3, \beta_3\}, \dots)_p$$

$$ab = (\alpha_1 + \beta_1, \alpha_2 + \beta_2, \alpha_3 + \beta_3, \dots)_p$$

czyli

$$NWW(a,b) \times NWD(a,b) = (min\{\alpha_1, \beta_1\} + max\{\alpha_1, \beta_1\}, \dots)_p$$

Dla dowolnych liczb rzeczywistych r i q zachodzi $r + q = min\{r,q\} + max\{r,q\}$, więc:

$$(min\{\alpha_1, \beta_1\} + max\{\alpha_1, \beta_1\}, \dots)_p = (\alpha_1 + \beta_1, \dots)_p = ab$$

a to kończy dowód. □

Aby obliczyć najmniejszą wspólną wielokrotność (największy wspólny dzielnik) więcej niż dwóch liczb, można skorzystać z poniższych obserwacji:

Lemat 1.10 *Dla* $a_i = (\alpha_{i,1}, \alpha_{i,2}, \alpha_{i,3}, \dots)_p$, *gdzie* $i \in \{1, 2, 3, \dots n\}$, *zachodzi:*

$$NWW(\{a_i\}_{i \in \{1,2,3,\dots n\}}) = (max\{\alpha_{i,1}\}_{i \in \{1,2,3,\dots n\}}, max\{\alpha_{i,2}\}_{i \in \{1,2,3,\dots n\}}, max\{\alpha_{i,3}\}_{i \in \{1,2,3,\dots n\}}, \dots)_p$$

$$NWD(\{a_i\}_{i \in \{1,2,3,\dots n\}}) = (min\{\alpha_{i,1}\}_{i \in \{1,2,3,\dots n\}}, min\{\alpha_{i,2}\}_{i \in \{1,2,3,\dots n\}}, min\{\alpha_{i,3}\}_{i \in \{1,2,3,\dots n\}}, \dots)_p$$

Dowód: Dowód jest analogiczny, jak w lematach 1.7 oraz 1.8.

Lemat 1.11 *Dla* $a_i \in \mathbb{N}_+$ *zachodzi:*

$$NWW(\{a_i\}_{i \in \{1,2,3,\dots n+1\}}) = NWW(NWW(\{a_i\}_{i \in \{1,2,3,\dots n\}}), a_{n+1})$$

Dowód: Z lematu 1.10:

$$NWW(\{a_i\}_{i\in\{1,2,3,\dots n+1\}}) = (max\{\alpha_{i,1}\}_{i\in\{1,2,3,\dots n+1\}}, \dots)_p$$

$$NWW(NWW(\{a_i\}_{i\in\{1,2,3,\dots n\}}), a_{n+1}) = (max\{max\{\alpha_{i,1}\}_{i\in\{1,2,3,\dots n\}}, \alpha_{n+1,1}\}, \dots)_p$$

Zachodzi $max\{max\{\alpha_{i,j}\}_{i\in\{1,2,3,\dots n\}}, \alpha_{n+1,j}\} = max\{\alpha_{i,j}\}_{i\in\{1,2,3,\dots n+1\}}$ (dla $j \in \mathbb{N}_+$), stąd prawe strony obu równości są sobie równe. □

Lemat 1.12 *Dla $a_i \in \mathbb{N}_+$ zachodzi:*

$$NWD(\{a_i\}_{i\in\{1,2,3,\dots n+1\}}) = NWD(NWD(\{a_i\}_{i\in\{1,2,3,\dots n\}}), a_{n+1})$$

Dowód: Dowód jest analogiczny jak w lemacie 1.11.

1.1.4 Znajdowanie dzielników liczby całkowitej w $O(\sqrt{n})$

Naiwny sposób znajdowania dzielników liczby $n \geq 2$ polegałby na sprawdzeniu wszystkich liczb ze zbioru $\{1, 2, 3, ..., n\}$ w czasie $O(n)$. Można to przyspieszyć dwukrotnie, ograniczając się tylko do zbioru $\{1, 2, 3, ..., \lfloor \frac{n}{2} \rfloor\}$. Optymalniejsze rozwiązanie korzysta z faktu, że dla dowolnej liczby całkowitej k jeżeli jest ona dzielnikiem n, to $k \leq \sqrt{n}$ lub $\frac{n}{k} \leq \sqrt{n}$. Gdyby było inaczej (czyli $k > \sqrt{n}$ i $\frac{n}{k} > \sqrt{n}$), to zaszłoby $k \times \frac{n}{k} > \sqrt{n} \times \sqrt{n}$, czyli $n > n$, co jest nieprawdą.

```
1.  DZIELNIKI( n )
2.      i := 1
3.      while i × i ≤ n
4.          if n%i = 0
5.              WYPISZ( i )
6.              if i × i ≠ n
7.                  WYPISZ( n/i )
8.          i := i + 1
```

1.1.5 Sprawdzanie pierwszości liczby

Algorytm jest analogiczny do poprzedniego - wystarczy zwrócić fałsz, gdy znajdzie się jakiś dzielnik liczby różny od 1 i jej samej (algorytm działa dla liczb większych od 1).

```
1.  PIERWSZA( n )
2.      i := 2
3.      while i × i ≤ n
4.          if n%i = 0
5.              return false
6.          i := i + 1
7.      return true
```

1.1.6 Wyznaczanie cyfr liczby

Ostatnia cyfra w systemie dziesiętnym liczby dodatniej n jest równa reszcie z dzielenia jej przez 10. Kolejne cyfry można wyznaczyć w ten sam sposób, postępując tak samo dla wartości $\frac{n-n\%10}{10}$.

```
1.  CYFRY( n )
2.      while n > 0
3.          WYPISZ( n%10 )
4.          n := n/10
```

Można zmodyfikować algorytm tak, aby wyznaczał cyfry danej liczby w reprezentacji o dowolnej podstawie p, zamieniając 10 na p. Złożoność obliczeniowa procedury jest równa ilości cyfr w liczbie, czyli $O(\ln(n))$.

1.1.7 Odwracanie kolejności cyfr w liczbie

Można wykorzystać poprzedni algorytm, *doklejając* w każdej iteracji pętli do wyniku bierzącą cyfrę (czyli wykonując operację $r := r \times 10 + c$, gdzie c jest wartością bierzącej cyfry).

```
1.  ODWROC( n )
2.      r := 0
3.      while n > 0
4.          r := r × 10 + n%10
5.          n := n/10
6.      return r
```

1.1.8 Sito Eratostenesa

Sprawdzanie pierwszości pojedynczej liczby naturalnej n metodą 1.1.5 zajmuje czas $O(\sqrt{n})$. Zastosowanie tej metody dla kolejnych liczb naturalnych $1, 2, 3, \ldots, n$ miałoby kosz czasowy $O(n\sqrt{n})$. Można to zrobić szybciej stosując sito Eratostenesa. Metoda jest następująca: przetwarza się wszystkie liczby od 1 do n i jeżeli natrafiona liczba nie została wcześniej oznaczona, to jest ona liczbą pierwszą - wówczas oznacza się wszystkie wielokrotności danej liczby mniejsze lub równe n. Ilość wykonanych operacji jest rzędu $n + \sum_{p\in\mathbb{P}} \lfloor \frac{n}{p} \rfloor$. Prawdziwe jest twierdzenie:

Twierdzenie 1.13 *Dla liczb naturalnych n zachodzi:* $\sum_{p\in\mathbb{P}} \lfloor \frac{n}{p} \rfloor = O(n \ln(\ln(n)))$.

Zatem algorytm ma złożoność obliczeniową $O(n \ln(\ln(n)))$.

```
1.  SITO( n )
2.      for i := 2 to n
3.          t[i] := false
4.      for i := 2 to n
5.          if t[i] = false
6.              WYPISZ(i)
7.              for j := 2 to n/i
8.                  t[i × j] := true
```

Algorytm można zmodyfikować tak, by działał w czasie $O(n)$ (każda liczba jest oznaczana maksymalnie jeden raz - jeżeli $a > 1$ i $a = (\alpha_1, \alpha_2, \alpha_3, \ldots)_p$, to jeżeli dla pewnego $j \in \mathbb{N}_+$ zachodzi $\alpha_j > 0$ oraz $\forall_{i<j}\alpha_i = 0$, to liczba a jest oznaczana w b-tej iteracji głównej pętli, gdzie $b = (\alpha_1, \alpha_2, \alpha_3, \ldots, \alpha_j - 1, \ldots)_p$).

```
1.  SITO( n )
2.      for i := 2 to n
3.          t[i] := false
4.      P := ∅
5.      for i := 2 to n
6.          if t[i] = false
7.              WYPISZ(i)
8.              DODAJ(P,i)
9.          for j in P
10.             t[i × j] := true
11.             if i%j = 0
12.                 break
```

1.1.9 Szybkie potęgowanie

Naiwna metoda obliczenia $(a^n)\%m$ polegająca na wykonaniu n mnożeń działa w czasie liniowym. Można ją przyspieszyć zauważając, że $a^n = (a^{\lfloor \frac{n}{2} \rfloor})^2 \times a^{n\%2}$. Wówczas problem można w czasie stałym sprowadzić do problemu dla wykładnika co najmniej dwa razy mniejszego. Zatem liczba wywołań rekurencyjnych poniższej procedury będzie nie większa niż $\log_2 n+1$ - złożoność obliczeniowa metody wyniesie $O(\ln(n))$.

```
1.  POT( a, n, m )
2.      if n = 0
3.          return 1
4.      if n = 1
5.          return a%m
6.      if n%2 = 0
7.          return POT(a, n/2, m)^2%m
8.      return (POT(a, n/2, m)^2 × a)%m
```

1.2 Algorytmy grafowe

1.2.1 Przeszukiwanie w głąb (DFS)

Algorytm przeszukiwania w głąb działa następująco: dla każdego sąsiada wierzchołka v, jeżeli nie został on odwiedzony, algorytm wykonuje się rekurencyjnie. Nazwa bierze się stąd, że algorytm przegląda wierzchołki "do przodu", zawracając dopiero, gdy wszyscy sąsiedzi danego wierzchołka zostali już odwiedzeni. Złożoność czasowa algorytmu to $O(n+m)$, gdzie $n = |V(G)|$ oraz $m = |E(G)|$ - każdy wierzchołek jest przetwarzany maksymalnie raz, oraz każda krawędź jest przetwarzana maksymalnie dwa razy.

```
1.  DFS(G,v)
2.      if vis[v] = false
3.          vis[v]:= true
4.          for (v, u) in E(G)
5.              DFS(G,u)
```

1.2.2 Przeszukiwanie wszerz (BFS)

Algorytm przeszukiwania wszerz odwiedza wierzchołki w kolejności od tych najbliższych do tych najdalszych - najpierw odwiedza sąsiadów, później sąsiadów sąsiadów itd. Innymi słowy, w i-tej iteracji odwiedzane są wierzchołki oddalone od wierzchołka początkowego o dokładnie

$i-1$ krawędzi. Złożoność czasowa algorytmu to $O(n+m)$, podobnie jak dla przeszukiwania w głąb.

```
1.  BFS( G, v )
2.      S1 := ∅
2.      S2 := ∅
2.      ADD(S1,v)
3.      vis[v]:= true
2.      while S1 ≠ ∅
4.          q := TOP(S1)
4.          POP(S1)
4.          for (q, u) in E(G)
5.              if vis[u]= false
3.                  vis[u]:= true
5.                  ADD(S2,u)
5.          if S1 = ∅
5.              SWAP(S1,S2)
```

2 Rozwiązania zadań

2.1 Lista 8.

2.1.1 Doskonałość

Można dla każdej liczby x z osobna w czasie $O(\sqrt{x})$ wyznaczyć sumę jej dzielników (korzystając np. z 1.1.4). Złożoność obliczeniowa: $O(n\sqrt{max\{x_i\}_{1\leq i\leq n}})$.

```
1.  SUMA( x )
2.     i := 1
3.     s := 0
4.     while i × i ≤ x
5.        if x%i = 0
6.           s := s + i
7.           if i × i ≠ x
8.              s := s + x/i
9.        i := i + 1
10.    return s − x
11. MAIN()
12.    WCZYTAJ( n )
13.    for i := 1 to n
14.       WCZYTAJ( x )
15.       r := SUMA(x)
16.       if r = x
17.          WYPISZ( 'OK' )
18.       else if r < x
19.          WYPISZ( 'NIEDOSTATECZNA' )
20.       else
21.          WYPISZ( 'WIELKA' )
```

2.1.2 Liczby zaprzyjaźnione

Dla każdej liczby naturalnej $x \in [a, b]$ można wyznaczyć potencjalną kandydatkę na liczbę zaprzyjaźnioną z nią - ta liczba jest równa $y = \text{SUMA}(x)$. Następnie należy zweryfikować, czy kandydatka rzeczywiście spełnia wymagane warunki: czy $y \in [a, b]$, czy $\text{SUMA}(y) = x$ oraz czy $y \neq x$. Tak postępuje się dla wszystkich $x \in [a, b]$ oraz dodatkowo sprawdza się, czy $x < y$, żeby zapobiec dwukrotnemu wypisaniu każdej pary. Złożoność obliczeniowa: $O((b-a)\sqrt{b})$.

```
1.  FUNC( x, b )
2.     y := SUMA(x)
3.     if y > x and y ≤ b and SUMA(y) = x
4.        WYPISZ( x, y )
5.        return true
6.     return false
7.  MAIN()
8.     WCZYTAJ( a, b )
9.     w := false
10.    for i := a to b
11.       w := w or FUNC(i, b)
12.    if w = false
13.       WYPISZ( 'BRAK' )
```

2.1.3 Odwróć i dodaj

Do odwrócenia kolejności cyfr liczby można wykorzystać 1.1.7. Wówczas wystarczy do danej liczby dodawać jej *odwrócenie* tak długo, aż otrzyma się palindrom (liczba jest palindromem, gdy jest równa swojemu *odwróceniu*). Złożoność obliczeniowa zależy od odpowiedzi na każde zapytanie (dla jednego zapytania o liczbę x złożoność obliczeniowa jego obsłużenia wynosi $O(k \times \ln(x) + k^2)$, gdzie k jest odpowiedzią na to zapytanie).

```
1.  FUNC( x )
2.      i := 0
3.      o := ODWROC(x)
4.      while o ≠ x
5.          x := x + o
6.          o := ODWROC(x)
7.          i := i + 1
8.      WYPISZ(i, x)
9.  MAIN()
10.     WCZYTAJ(n)
11.     for i := 1 to n
12.         WCZYTAJ(x)
13.         FUNC(x)
```

2.1.4 Liczby B-pierwsze

Aby sprawdzić, czy dana liczba jest pierwsza, można skorzystać z 1.1.5. Żeby natomiast wyznaczyć sumę cyfr, można użyć algorytmu analogicznego do 1.1.6.

```
1.  SUMACYFR( n, p )
2.      s := 0
3.      while n > 0
4.              s := s + (n%p)
5.              n := n/p
6.      return s
7.  SBP( n )
8.      return PIERWSZA(SUMACYFR(n,10)) and PIERWSZA(SUMACYFR(n,2))
        and PIERWSZA(n)
9.  MAIN()
10.     WCZYTAJ(a,b)
11.     c := 0
12.     for i := a to b
13.         if SBP(i) = true
14.             c := c + 1
15.     WYPISZ(c)
```

2.1.5 Suma ułamków

Zachodzi $\frac{a}{b} + \frac{c}{d} = \frac{ad+bc}{bd}$, więc jedyne co trzeba zrobić, to obliczyć wartości licznikia i mianownika oraz podzielić je przez ich największy wspólny dzielnik (skrócić ułamek).

```
1.  MAIN()
2.      WCZYTAJ(a,b,c,d)
3.      l := a × d + c × b
4.      m := b × d
5.      n := NWD(l, m)
6.      l := l/n
7.      m := m/n
8.      WYPISZ(l,'/',m)
```

2.2 Lista 9.

2.2.1 Drzewo Sterna-Brocota

Należy obliczać wartości ułamków po kolei na kolejnych poziomach drzewa: mając zapisane wartości ułamków w tablicy o długości n (dokładnie to będą dwie tablice z licznikami i mianownikami) można utworzyć nową tablicę o długości $2n-1$, gdzie na pozycjach parzystych $2i$ będą wartości ułamków z pozycji i w pierwszej tablicy, a na pozycjach nieparzystych będą wartości *mediantów* sąsiednich ułamków. Na początku tablica ma długość 2 i zawiera ułamki $\frac{0}{1}$ oraz $\frac{1}{0}$.
Można też użyć funkcji rekurencyjnej (wypisuje wszystkie ułamki pomiędzy $\frac{a}{b}$ oraz $\frac{c}{d}$ do głębokości n):

```
1.  POMIĘDZY(a,b,c,d,n)
2.      if n > 0
3.          POMIĘDZY(a,b,a + c,b + d,n − 1)
4.          WYPISZ(a + c,'/',b + d)
5.          POMIĘDZY(a + c,b + d,c,d,n − 1)
6.  MAIN()
7.      WCZYTAJ(n)
8.      WYPISZ(0,'/',1)
8.      POMIĘDZY(0,1,1,0,n)
10.     WYPISZ(1,'/',0)
```

2.2.2 Legiony

Wystarczy obliczyć wartość $(r^k \times b^m)\%1000000009$ (korzystając np. z 1.1.9).

2.2.3 Liczby parzystocyfrowe

Kluczowa jest obserwacja, że dowolna liczba parzystocyfrowa podzielona przez 2 jest równa liczbie, której cyfry należą do zbioru $\{0,1,2,3,4\}$. Jednocześnie każdą liczbę, której cyfry są mniejsze od 5, można pomnożyć przez 2 i otrzymać liczbę parzystocyfrową. Zatem n-ta liczba parzystocyfrowa jest równa dwukrotności n-tej liczby o cyfrach mniejszych niż 5. Natomiast n-ta liczba o cyfrach mniejszych niż 5 jest po prostu reprezentacją liczby n w systemie o podstawie 5.

```
1. REPR(n,p)
2.     r := 0
3.     while n > 0
4.         r := r × 10 + n%p
5.         n := n/p
6.     return ODWROC(r)
7. MAIN()
8.     WCZYTAJ(n)
9.     WYPISZ(REPR(n, 5) × 2)
```

Ze względu na ograniczenia liczby na wejściu, konieczne jest użycie własnej implementacji arytmetyki dla długich liczb.

2.2.4 Struś pędziwiatr

Wystarczy obliczyć $NWW(\{x_i + 1\}_{1 \leq i \leq n}) - 1$ (korzystając np. z 1.11).

3 Oznaczenia

$\mathbb{N}$	zbiór liczb naturalnych z zerem
$\mathbb{N}_+$	zbiór liczb naturalnych bez zera
$\mathbb{P}$	zbiór liczb pierwszych
$\mathbb{Z}$	zbiór liczb całkowitych
$\mathbb{R}$	zbiór liczb rzeczywistych
$a\%b$	reszta z dzielenia a przez b, gdzie $a, b \in \mathbb{N}$ i $b \neq 0$
$\lfloor a \rfloor$	największa liczba całkowita mniejsza lub równa a ($a \in \mathbb{R}$)
$\lceil a \rceil$	najmniejsza liczba całkowita większa lub równa a ($a \in \mathbb{R}$)
$E(G)$	zbiór krawędzi grafu G
$V(G)$	zbiór wierzchołków grafu G
$\lvert\mathcal{A}\rvert$	ilość elementów (moc) zbioru $\mathcal{A}$