

## Wstęp do informatyki

## Lista 5

Uwagi:

- Programy stanowiące rozwiązania zadań na tej liście powinny być napisane w języku C lub Python i poprzedzone prezentacją **idei rozwiązania** (najlepiej przy pomocy **pseudokodu**). Należy również przeanalizować **złożoność** czasową i pamięciową. Staraj się, aby złożoność Twojego rozwiązania była jak najmniejsza!
- W rozwiązaniach zadań nie należy korzystać z funkcji/narzędzi wspomagających ten proces, dostępnych w wykorzystywanym języku programowania i/lub jego bibliotekach, które nie były stosowane na wykładzie (w szczególności, korzystamy tylko z operatorów arytmetycznych  $+$   $-$   $*$   $/$   $\%$ , w przypadku Pythona również  $//$ ).

- [1] Dla ustalonej liczby naturalnej  $k$  wyznacz wartości wykładnika  $b$  z przedziału  $[2^k, 2^{k+1}]$ , dla których algorytm szybkiego potęgowania podany na wykładzie wykona:

- najwięcej mnożeń
- najmniej mnożeń

W odpowiedziach do punktów a. i b. podaj wartość  $b$ , liczbę wykonanych mnożeń i potęgi liczby  $a$ , które będą domnażane do wyniku (czyli zmiennej `rez`). Analizę przeprowadź dla obu implementacji algorytmu (rekurencyjnej i nierekurencyjnej).

*Wskazówka:* najpierw możesz rozwiązać zadanie dla konkretnej wartości  $k$  (np.  $k=10$ ), a potem uogólnić poczynione obserwacje.

- [2] Silniową reprezentacją liczby  $n$  nazywamy ciąg  $s_k, s_{k-1} \dots s_2, s_1$  taki, że

- $n = 1! \cdot s_1 + 2! \cdot s_2 + \dots + k! \cdot s_k$
- $s_i \leq i$  dla każdego  $i \in \{1, 2, \dots, k\}$
- $s_k > 0$

Wiadomo, że każda liczba naturalna ma dokładnie jedną reprezentację silniową. Napisz funkcję, która dla zadanej liczby  $n$ , wypisuje jej reprezentację silniową i działa w czasie  $O(\log n)$ .

*Przykład.* Silniowa reprezentacja liczby 100 jest równa 4, 0, 2, 0 gdyż  $100 = 1! \cdot 0 + 2! \cdot 2 + 3! \cdot 0 + 4! \cdot 4$

- [1] Wiadomo, że zachodzą tożsamości:

$$\text{nwd}(2n, 2m) = 2 \cdot \text{nwd}(n, m)$$

$$\text{nwd}(2n, m) = \text{nwd}(n, m) \text{ dla nieparzystej liczby } m > 0.$$

Następująca funkcja wyznacza największy wspólny dzielnik liczb  $n$  i  $m$ , wykorzystując powyższe tożsamości.

```
int gcd(int n, int m)
{
    int ilenp;
    if (!m) return n;
    if (n < m) return gcd(m, n);
    ilenp = n%2 + m%2;
    if (ilenp == 2) return gcd(n-m, m);
    if (!ilenp) return 2*gcd(n/2, m/2);
    if (n%2 == 0) return gcd(n/2, m);
    else return gcd(n, m/2);
}
```

```
def gcd(n,m):
    if (m==0): return n
    if (n<m): return gcd(m,n)
    ilenp = n%2 + m%2
    if (ilenp==2): return gcd(n-m,m)
    if (ilenp==0): return 2*gcd(n/2,m/2)
    if (n%2==0): return gcd(n/2,m)
    else: return gcd(n,m/2)
```

- a. Pokaż, że funkcja gcd działa w czasie  $O(\log n + \log m)$ .
  - b. Napisz nierekurencyjną funkcję wyznaczającą największy wspólny dzielnik z dwóch liczb w taki sposób, w jaki realizuje to funkcja gcd.
4. [1] Ciąg  $G$  zdefiniowany jest w następujący sposób
- $$G_0 = G_1 = G_2 = 1$$
- $$G_n = G_{n-1} + G_{n-2} + G_{n-3} \text{ dla } n > 2.$$
- Napisz funkcję, która dla liczby naturalnej  $n$  wyznacza wartość  $G_n$  w czasie  $O(n)$  i pamięci  $O(1)$ .
5. Niech funkcja  $T$  określona na liczbach naturalnych będzie zadana następującym wzorem:
- $$T(n, 0) = n \text{ dla } n \geq 0$$
- $$T(0, m) = m \text{ dla } m \geq 0$$
- $$T(n, m) = T(n-1, m) + 2T(n, m-1) \text{ dla } n > 0 \text{ i } m > 0$$
- a. [1] Napisz rekurencyjną funkcję  $fTrec(\text{int } n, \text{int } m)$  obliczającą wartość funkcji  $T$  dla argumentów  $n$  i  $m$ . Narysuj drzewo wywołań dla  $fTrec(3, 4)$  i podaj wartość  $T(3, 4)$ .
  - b. [1] Napisz nierekurencyjną funkcję  $fTiter(\text{int } n, \text{int } m)$  obliczającą wartość funkcji  $T(n, m)$  w czasie  $O(n \cdot m)$  i pamięci  $O(n+m)$ .
6. [1] Napisz funkcję  $fibonacci(k, r)$ , która zwraca wartość  $F_k \bmod r$ .  
*Uwaga:* W obliczeniach wykonywanych przez Twoją funkcję nie powinny występować liczby większe od dwukrotności maksimum z liczb  $r$  i  $k$ .
7. Napisz funkcję, która dla podanych liczb naturalnych  $n, m$  wyznacza najmniejszą liczbę naturalną  $k$  taką, że  $n^k \geq m$ .
- Wersja łatwiejsza [1 pkt]: złożoność czasowa algorytmu może wynieść  $O(k)$ .
- Wersja trudniejsza [2 pkt]: złożoność czasowa algorytmu powinna być  $O(\log k)$ .
- Wskazówka.* Najpierw wyznacz najmniejsze  $i$  takie, że  $n^{2^i} \geq m$ .

**Zadania dodatkowe, nieobowiązkowe (nie wliczają się do puli punktów do zdobycia na ćwiczeniach, punktacja została podana tylko jako informacja o trudności zadań wg wykładowcy)**

1. [1] Napisz funkcję  $fibonacci(k, r)$ , która zwraca wartość  $p$  taką, że  $F_p$  jest najmniejszą liczbą Fibonacciego, która przy dzieleniu przez  $k$  daje resztę  $r$ . Czy potrafisz oszacować złożoność czasową swojego rozwiązania?  
*Przykład*  
 Wartość zwracana dla  $fibonacci(5, 4)$  to 8, ponieważ najmniejszą liczbą Fibonacciego dającą resztę 4 przy dzieleniu przez 5 jest  $F_8$ .  
*Uwaga:* W obliczeniach wykonywanych przez Twoją funkcję nie powinny występować liczby większe od dwukrotności maksimum z liczb  $p$  i  $k$ .
2. [1] Napisz funkcję, która dla liczby naturalnej  $n$  wyznacza wartość  $n$ -tej liczby Fibonacciego  $F_n$  w czasie  $O(n)$  i pamięci  $O(1)$ .
3. [3] Napisz funkcję  $fibonacci(n)$ , która wyznacza  $F_n$  w czasie  $O(\log n)$ .

Uwaga do zad. 8 i 9: wartości  $F_n$  już dla niewielkich  $n$  przekraczają zakres typów `int` i `long` w języku C; implementując swoje rozwiązania możesz wyznaczać np. resztę z dzielenia przez 100 liczby  $F_n$ .

4. Napisz nierekurencyjną funkcję `fTiter(int n, int m)` obliczającą wartość funkcji  $T(n, m)$  z zadania 5 w czasie  $O(n \cdot m)$  i pamięci  $O(\min(n, m))$ .