**Metoda programowania „dziel i zwyciężaj” – część II**

Przy analizowaniu metody programowania „dziel i zwyciężaj” warto zadać pytanie, czy można przyspieszyć działanie np. wyszukiwania binarnego, które i tak już jest szybkie. Odpowiedź jest twierdząca i z życia wzięta. Gdybyśmy bowiem przeszukiwali np. słownik polsko-angielski, aby znaleźć angielski odpowiednik słowa ***bateria***, to raczej nie otwieralibyśmy słownika w połowie (zakładamy milcząco, iż częstotliwość występowania liter i wyrazów w każdym języku jest podobna, co *nota bene* nie jest prawdą), lecz bliżej początku. W algorytmach binarnych nie uwzględniamy tej okoliczności, dzieląc przedziały na połowy, aż do osiągnięcia satysfakcjonującego wyniku, co obrazuje wzór (dla ułatwienia zajmiemy się algorytmem wyszukującym liczbę $y$ w uporządkowanym ciągu liczbowym)

$$srodek = \frac{lewy + prawy}{2}$$

Wzór ten można zapisać inaczej jako

$$srodek = lewy + \frac{1}{2}(prawy - lewy)$$

Ten ostatni wzór można poprawić, a tym samym poprawić złożoność obliczeniową algorytmu, jeśli uwzględnimy wartość poszukiwanej liczby $y$ oraz wartości elementów $a_{lewy}$ oraz $a_{prawy}$ w przeszukiwanym ciągu, korzystając ze znanej w matematyce metody interpolacyjnej

$$s = lewy + \frac{y - a_{lewy}}{a_{prawy} - a_{lewy}}(prawy - lewy)$$

Tak zmodyfikowany algorytm nazywany jest **algorytmem umieszczania interpolacyjnego**, którego specyfikację i listę kroków zamieszczono poniżej.

**Wejście:**
Uporządkowany ciąg liczb w tablicy $T[k \ldots l]$, gdzie $k \leq l$.

**Wyjście:**
Miejsce dla liczby $y$ w ciągu $T[k \ldots l]$, czyli taka wartość $s$, że $a_s = y$, jeśli $y$ równe jest jakiemuś elementowi przeszukiwanego ciągu albo $s$ jest największą liczbą taką, że $a_s \leq y$.

**K1**.

$$lewy \leftarrow k; prawy \leftarrow l;\ \text{(przypisanie początkowych końców przedziału przeszukiwań)}$$

**K2**.

$$s \leftarrow min\left\{lewy + \left[\frac{y - a_{lewy}}{a_{prawy} - a_{lewy}}(prawy - lewy)\right], prawy\right\}$$

**K3**.
Jeśli $a_s \leq y$, to $lewy \leftarrow s$. W przeciwnym wypadku $prawy \leftarrow s - 1$.

**K4**.
Jeśli $lewy = prawy$ albo $a_{lewy} = y$, to zakończ algorytm. W przeciwnym wypadku powtórz **K2**.

Teoretycznie można dowieść, że złożoność zaprezentowanego algorytmu (pod kątem przeciętnej liczby porównań) wynosi jedynie $O(\log_2 \log_2 n)$. Dla przykładu, jeśli przyjmiemy $n = 65536$, to $O(\log_2 n) = 16$, natomiast $O(\log_2 \log_2 n) = 4$. Jeśli jednak uwzględnimy dodatkowe operacje arytmetyczne, które występują wyłącznie w algorytmie umieszczania interpolacyjnego, to różnica między nim a algorytmami binarnymi nie jest już taka duża.

**Z1. (tylko dla ambitnych)**
W języku C# (Java, Python lub C++) napisz program, który przy dużej liczbie danych wejściowych porównuje algorytm binarny oraz algorytm umieszczania interpolacyjnego pod kątem złożoności obliczeniowej.

**Z2.**
W pierwszym semestrze na przedmiocie „Algorytmy i struktury danych" poznaliśmy zastosowanie metody „dziel i zwyciężaj" do wyszukiwania przybliżonej wartości miejsca zerowego (miejsc zerowych) funkcji w zadanym przedziale argumentów. Zastosowany w tym celu **algorytm bisekcji** (połowienia) działał poprawnie, jeśli funkcja $f(x)$ była ciągła w tym przedziale oraz istniały w nim co najmniej dwa takie punkty $a$ i $b$, w których wartość funkcji ma przeciwne znaki, czyli $f(a) \cdot f(b) < 0$.

Dana jest funkcja $y = x^3 - 12x^2 + 11x - 1$, o której wiemy jedynie tyle, że może mieć jedno, dwa lub trzy miejsca zerowe. Nie znamy natomiast przedziału, w którym miejsca zerowe mogą wystąpić. Zaproponuj oraz zaimplementuj w języku C# (Java, Python lub C++) algorytm, który oblicza miejsce zerowe tej funkcji z dokładnością zadaną przez użytkownika w sposób iteracyjny lub rekurencyjny (do wyboru).

**Z3.**
Napisz w języku C# (C++, Python, Java) program, który wyświetla liczbę arabską w zapisie rzymskim w zakresie od 1 do 3999 oraz odwrotnie (wybór należy do użytkownika). Liczby rzymskie w formie takiej, jaką znamy obecnie, używają siedmiu znaków: I = 1, V = 5, X = 10, L = 50, C = 100, D = 500 i M = 1000. System wykorzystuje dodawanie i odejmowanie w celu tworzenia symboli liczbowych. Symbole od 1 do 10 są takie: I, II, III, IV, V, VI, VII, VIII, IX i X. Rzymianie nie używali symbolu zera i w celu jego reprezentacji pisali słowo *nulla*. W tym systemie największe symbole znajdują się po lewej stronie, a najmniej znaczące – po prawej. Przykładowo, liczbą rzymską reprezentującą rok 2023 jest MMXXIII.

**Z4.**
Silniowy system pozycyjny to pozycyjny zapis liczb naturalnych, w którym mnożniki dla kolejnych pozycji definiowane są przez silnie kolejnych liczb naturalnych

$$(x)_! = (x_n x_{n-1} x_{n-2} \cdots x_2 x_1) = x_n \cdot n! + x_{n-1} \cdot (n-1)! + \cdots + x_2 \cdot 2! + x_1 \cdot 1!$$

W systemie silniowym współczynnik $x_i$, który odpowiada mnożnikowi $i!$, spełnia zależność $0 \leq x_i \leq i$.

Zapis każdej liczby w silniowym systemie pozycyjnym jest jednoznaczny, tj. każdą liczbę naturalną można zapisać dokładnie i tylko w jeden sposób, przykładowo

$$(1220)_! = 1 \cdot 4! + 2 \cdot 3! + 2 \cdot 2! + 0 \cdot 1! = 40$$

Zamiana zapisu liczby $x$ w systemie dziesiętnym na zapis liczby w systemie silniowym może przebiegać wg następującego algorytmu: szukamy największej liczby $k$, której silnia nie przekracza $x$. Pierwsza jej cyfra to wynik dzielenia całkowitego $x$ przez $k!$. Kolejne cyfry zapisu silniowego (zaczynając od cyfr najbardziej znaczących) otrzymujemy przez wyznaczanie wyników dzielenia liczby $x$ przez $(k-1)_!$, $(k-2)_!$,⋯, $2!$, $1!$. Po wyznaczeniu cyfry $x_i$, odpowiadającej współczynnikowi $i!$, zmniejszamy wartość $x$ o liczbę odpowiadającą cyfrze $x_i$, czyli $x_i \cdot i$. Oznacza to, że $x$ przyjmuje wartość $x \bmod k!$.

Przykład zamiany liczby dziesiętnej 1548 na zapis silniowy

| $x$ | $k$ | $x \operatorname{div} k!$ | $x \bmod k!$ |
|---|---|---|---|
| 1548 | 6 | **2** | 108 |
| 108 | 5 | **0** | 108 |
| 108 | 4 | **4** | 12 |
| 12 | 3 | **2** | 0 |
| 0 | 2 | **0** | 0 |
| 0 | 1 | **0** | 0 |

Zatem $(1548)_{10} = (204200)_!$.

Na podstawie powyższych rozważań napisz w języku C# (C++, Python, Java) program, który zamienia dziesiętną liczbę naturalną na odpowiadającą jej liczbę w systemie silniowym oraz odwrotnie (wybór należy do użytkownika).

Termin wykonania zadań domowych: **14 kwietnia 2023 r. (do północy)**. Zasady nazewnictwa i przysyłania zadań bez zmian.