Ćwiczenia 1

1.1 Mamy daną n-elementową listę liczb całkowitych $a[1..n]$. Chcemy znaleźć

$$s^\star = \max\left(\{0\} \cup \left\{\sum_{k=i}^{j} a[k] : 1 \leqslant i \leqslant j \leqslant n\right\}\right).$$

(a) Rozważmy algorytm obliczania $s^\star$ wprost z definicji

Algorytm A

$s^\star := 0;$
for $i \in [1..n]$ **do**
  for $j \in [i..n]$ **do**
    $s := 0;$
    for $k \in [i..j]$ **do**
      $s := s + a[k];$ ▷ operacja dominująca
    $s^\star := \max(s^\star, s);$

– ile **dokładnie** operacji dominujących wykona algorytm A?

(b) Spójrzmy na proste usprawnienie Algorytmu A

Algorytm B

$s^\star := 0;$
for $i \in [1..n]$ **do**
  $s := 0;$
  for $j \in [i..n]$ **do**
    $s := s + a[k];$ ▷ operacja dominująca
    $s^\star := \max(s^\star, s);$

– na czym polega usprawnienie?
– ile **dokładnie** operacji dominujących wykona algorytm B?

(c) Na koniec przeanalizujemy szybki algorytm obliczania $s^\star$

Algorytm C

$s^\star := 0$, $p := 0;$
for $i \in [1..n]$ **do**
  $p := p + a[i];$ ▷ operacja dominująca
  $s^\star := \max(s^\star, p);$
  if $p < 0$ **then**
    $p := 0;$

– udowodnij poprawność Algorytmu C przez podanie stosownego niezmiennika pętli „for”,
– dla każdego algorytmu policz, jak długo będzie trwało jego wykonywanie dla $n = 1000^2$ przy założeniu, że w jednej sekundzie jest wykonywanych 1000^3 dodawań (operacji dominujących).

1.2 Liczby Fibonacciego definiujemy następująco:

$$F_n = \begin{cases} n, & \text{dla } n \in \{0, 1\} \\ F_{n-1} + F_{n-2}, & \text{dla } n > 1. \end{cases}$$

(a) Oblicz, ile dodawań jest wykonywanych przy liczeniu F_n rekurencyjnie.

(b) Zaprojektuj algorytm obliczania liczby F_n wykonujący $O(\log n)$ operacji arytmetycznych, z wykorzystaniem wzorów rekurencyjnych (dla $n > 1$)

$$\begin{aligned} F_{2n-1} &= F_n^2 + F_{n-1}^2 \\ F_{2n} &= F_n^2 + 2F_nF_{n-1}. \end{aligned}$$

(c) Zaproponuj algorytm obliczania liczby F_n wykonujący $O(\log n)$ operacji arytmetycznych, z wykorzystaniem wzoru rekurencyjnego (dla $n > 1$):

$$\begin{bmatrix} F_n \\ F_{n-1} \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} F_{n-1} \\ F_{n-2} \end{bmatrix}.$$

1.3 Zaproponuj iteracyjny algorytm, który w czasie $O(\log n)$ oblicza $\lfloor \sqrt{n} \rfloor$ dla danej nieujemnej liczby całkowitej n. Udowodnij poprawność swojego algorytmu.

1.4 Rozważmy następujący algorytm:

Algorytm ?

Założenie: $x \leqslant 100$ — liczba całkowita

$y := x$, $z := 1$;
while $(y \leqslant 100)$ or $(z \neq 1)$ **do**
  if $y \leqslant 100$ **then**
    $y := y + 11$;
    $z := z + 1$;
  else
    $y := y - 10$;
    $z := z - 1$;

Udowodnij, że Algorytm ? ma własność stopu.

1.7 Niech n będzie dodatnią liczbą całkowitą i niech S będzie multizbiorem n liczb całkowitych dodatnich. W tym zadaniu za koszt sumowania dwóch liczb przyjmujemy wartość ich sumy. Rozważmy następujący algorytm:

Algorytm

Suma(S) ::
$z := 0$;
while $|S| \neq 1$ **do**
  $(x, y) :=$ Para(S);
  $S := S \setminus \{x, y\}$;
  $S := S \cup \{x + y\}$;
return $z \in S$;

(a) W wyniku wywołania funkcji Para otrzymujemy parę elementów z S. Zaimplementuj funkcję Para w taki sposób, żeby koszt obliczania z był jak najmniejszy. Udowodnij poprawność swojego rozwiązania.

(b) Załóżmy teraz, że S jest ciągiem, a funkcja Para zwraca i usuwa dwa sąsiednie elementy w ciągu oraz wstawia w ich miejsce ich sumę. Zaprojektuj wydajny algorytm, który wyznaczy strategię pobierania par z S tak, żeby w wyniku koszt obliczania z był jak najmniejszy.