02. Przepływ sterowania - zadania

2.1 Napisz własną implementację funkcji `sign`()

2.2 Napisz własną implementację funkcji `abs`()

2.3 Napisz własną implementację funkcji `all`()

2.4 Napisz własną implementację funkcji `any`()

2.5 Napisz własną implementację funkcji `cumsum`().

2.6 Napisz funkcję `dziesietnaNaBinarna`(), która jako parametr przyjmuje wektor liczb całkowitych `x`. Funkcja powinna zwracać wektor napisów, gdzie i-ty napis to binarny zapis i-tego elementu z `x`.

Algorytm na zamianę liczby z systemu dziesiętnego (dowolnego) na binarny:

Wprowadźmy zmienną `liczba`, równą liczbie, którą chcemy zamienić na system binarny. Dopóki `liczba` nie jest równa zero, to wykonuj dwa polecenia: zapisz resztę z dzielenia, a następnie podziel `liczbę` przez 2 i zapisz wynik w zmiennej `liczba`.

Liczbę cyfr w systemie dwójkowym możemy z góry oszacować poprzez log_2(`liczba`) (logarytm o podstawie 2). Dla jakiej liczby algorytm nie zadziała? (przypadek ten należy obsłużyć “ręcznie”)

2.7 Napisz funkcję `dziesietnaNaBinarnaUlamek`(), która jako parametr przyjmuje wektor `x` liczb z zakresu `(0,1)`. Funkcja powinna zwracać wektor napisów, gdzie i-ty napis to binarny zapis (ułamek dwójkowy) i-tego elementu z `x`.

Algorytm na zamianę ułamka z systemu dziesiętnego (dowolnego) na binarny:

Wprowadźmy zmienną `liczba`, równą ułamkowi, którą chcemy zamienić na system binarny.
Dopóki `liczba` nie jest równa zero, to wykonuj dwa polecenia:

1. zapisz część całkowitą (na lewo od przecinka) ze zmiennej `liczba`, jeśli ta część była równa 1, to odejmij 1 od `liczby`
2. pomnóż `liczbę` przez 2 i zapisz wynik w zmiennej `liczba`.

Załóż z góry, ile cyfr chcesz otrzymać maksymalnie, aby zabezpieczyć się przed rozwinięciami nieskończonymi. Sprawdź wyniki dla 0.25, 0.5, ale też 0.1 czy ⅛.

2.8 Napisz własną implementację funkcji `rle`().

2.9 Porównaj czas działania swoich funkcji z zadań 2.1 i 2.2 z tymi dostępnymi w R. Wykorzystaj funkcję wbudowaną `system.time` lub funkcję `microbenchmark` z pakietu `microbenchmark`. Sprawdź w dokumentacji jak użyć tych funkcji.

2.10 Napisz funkcję która dla danego wektora liczbowego (należy przerwać działanie funkcji, jeśli zostanie podany inny) zwróci jedno ze słów:

- malejący - jeśli każdy kolejny element wektora jest mniejszy od poprzedniego
- stały - jeśli wszystkie są równe
- rosnący - jeśli każdy kolejny element wektora jest większy od poprzedniego
- nieokreślony - jeśli nie zachodzi żadna z powyższych sytuacji

Wskazówka 1 - da się to zrobić bez użycia pętli.
Wskazówka 2 - pomocne mogą być funkcje `diff` oraz `all`

2.11 Napisz funkcję `zlicz`, która dla danej wartości całkowitej `k > 0` i wektora całkowitego `x` o `n` elementach ze zbioru `{1, 2, ... , k}` (jeśli taki nie jest podany, należy przerwać wykonywanie funkcji), zwróci wektor o długości `k`, w którym i-ty element jest równy liczbie wystąpień wartości `i` w `x`, dla `i = 1, ... , k`.
`Przykład:`
`zlicz(c(1,4,1,2,1), 5) == c(3, 1, 0, 1, 0)`

2.12 Za Wikipedią: Sortowanie przez zliczanie (ang. counting sort) – metoda sortowania danych, która polega na sprawdzeniu ile wystąpień kluczy mniejszych od danego występuje w sortowanej tablicy. Algorytm zakłada, że klucze elementów należą do skończonego zbioru (np. są to liczby całkowite z przedziału 0..100), co ogranicza możliwości jego zastosowania.
Korzystając z funkcji napisanej w poprzednim zadaniu (zlicz) użyj jej aby wypisać posortowany wektor powyższą metodą. Skoro wiadomo, że były trzy 1, jedna 2, zero 3 itd to oznacza, że wystarczy wypisać wektor postaci: `c(1, 1, 1, 2, 4)`.
Utrudnienie - funkcja nie musi przyjmować wartości k, a powinna ją sama wyliczać na podstawie wektora wejściowego.

2.13 Napisz funkcję `dodawanie`(), która przyjmuje następujące argumenty:

1. wektor liczbowy `a`,
2. wektor liczbowy `b`,
3. wektor liczbowy `base`, jedna wartość domyślnie równa 10

Wektory `a` i `b` mają równą długość.

Wektory `a` i `b` reprezentują dwie liczby zapisane w systemie o podstawie `base`.
Wektory te mogą mieć dowolną liczbę zer nieznaczących. Funkcja powinna dodać je pisemnie, a następnie zwrócić wynik.
W wyniku nie powinniśmy mieć żadnych nieznaczących zer.
Zakładamy, że `a` i `b` mają równą długość.

Jeśli wektory `a` i `b` mają długość równą `n`, to ich suma zmieści się na `n+1` cyfrach.

Podpowiedź: najpierw rozwiąż zadanie dla systemu dziesiętnego, a potem zobacz, gdzie należy wprowadzić zmiany w kodzie, aby go uogólnić na dowolną bazę.

Przykłady

```
a <- c(9,9,9); b <- c(9,9,9)
dodawanie(a,b)
## [1] 1 9 9 8

a <- c(9,1,9); b <- c(9,8,9)
dodawanie(a,b)
## [1] 1908

a <- c(7,8,9); b <- c(2,1,9)
dodawanie(a,b)
## [1] 1 0 0 8

a <- c(1,2,3); b <- c(2,1,4)
dodawanie(a,b)
## [1] 3 3 7

a <- c(0,0,0); b <- c(0,0,0)
dodawanie(a,b)
```

[1] 0

2.14 Napisz funkcję `sredniaRuchoma()`, która dla danego wektora numerycznego x o n elementach oraz nieparzystej liczby naturalnej k wyznaczy k-średnią ruchomą, k<n, tj. zwróci wektor w= $(w_1, \ldots, w_{n-k+1})$, dla którego $w_i = \sum_{j=1}^{k} x_{i+j-1}/k$. Zadbaj o wydajność i wykorzystanie wyników z poprzedniej iteracji.

2.15 Napisz funkcję `potegowanie()`, która przyjmuje jako argumenty dwie liczby, pierwszą rzeczywistą x i drugą całowitą n. Niech funkcja zwraca wartość x^n. Czy umiesz zrobić to wydajnie? Oczywiście bez używania operatora potęgowania.

2.16 Napisz funkcję `szachownica()`, która wypisuje na konsolę szachownicę o zadanych rozmiarach przekazywanych jako argumenty, czyli ciąg typu:

```
*#*#*#*#*#
#*#*#*#*#*
*#*#*#*#*#
#*#*#*#*#*
*#*#*#*#*#
```

2.17 Napisz funkcję `powiekszonaSzachownica()`, która przyjmuje 3 parametry: n i m (rozmiary szachownicy), ale także k, czyli jej powiększenie. Dla k=3, n=10 i m=3 otrzymujemy:

```
******############******###
******############******###
******############******###
######******############***
######******############***
######******############***
******############******###
******############******###
******############******###
```

2.18 Zaimplementuj funkcję, `is.prime()`, która dla danego wektora liczb naturalnych x zwraca wektor logiczny, którego i-ty element odpowiada na pytanie, czy i-ty element x jest liczbą pierwszą czy nie.
Tzw. formuła Eulera $n^2 + n + 41$ generuje 40 różnych liczb pierwszych dla n całkowitych

od 0 do 39. Możesz użyć tych liczb do testów.