Algorytmy i Struktury Danych
Kolokwium I (29. III 2021)

Format rozwiązań

Rozwiązanie każdego zadania musi się składać z opisu algorytmu (wraz z uzasadnieniem poprawności i oszacowaniem złożoności obliczeniowej) oraz jego implementacji. Zarówno opis algorytmu jak i implementacja powinny się znajdować w tym samym pliku Pythona (rozszerzenie `.py`). Opis powinien być na początku pliku w formie komentarza (w pierwszej linii w komentarzu powinno być imię i nazwisko studenta). Opis nie musi być długi—wystarczy kilka zdań, jasno opisujących ideę algorytmu. Implementacja musi być zgodna z szablonem kodu źródłowego dostarczonym wraz z zadaniem. Niedopuszczalne jest w szczególności:

1. zmienianie nazwy funkcji implementującej algorytm, listy jej argumentów, lub nazwy pliku z rozwiązaniem,
2. modyfikowanie testów dostarczonych wraz z szablonem,
3. wypisywanie na ekranie jakichkolwiek napisów innych niż wypisywane przez dostarczony kod (ew. napisy dodane na potrzeby diagnozowania błędów należy usunąć przed wysłaniem zadania),
4. korzystanie z wbudowanych algorytmów sortowania i zaawansowanych struktur danych (np. słowników czy zbiorów).

Dopuszczalne jest natomiast:

1. korzystanie z następujących elementarnych struktur danych: krotka, lista, kolejka `collections.deque`,
2. korzystanie ze struktur danych dostarczonych razem z zadaniem (jeśli takie są).

Wszystkie inne algorytmy lub struktury danych wymagają implementacji przez studenta. Dopuszczalne jest oczywiście implementowanie dodatkowych funkcji pomocniczych w pliku z szablonem rozwiązania. Na potrzebny analizy złożoności algorytmów można zakładać, że funkcja `Partition` z algorytmu `QuickSort` dzieli tablicę na dwie równe części (nawet jeśli dostarczona implementacja nie ma tej własności).

Zadania niezgodne z powyższymi ograniczeniami otrzymają ocenę 0 pkt. Rozwiązania w innych formatach (np. `.PDF`, `.DOC`, `.PNG`, `.JPG`) z definicji nie będą sprawdzane i otrzymają ocenę 0 punktów, nawet jeśli będą poprawne.

Proszę pamiętać, że rozwiązania trochę wolniejsze niż oczekiwane, ale poprawne, mają szanse na otrzymanie 1 punktu. Rozwiązania szybkie ale błędnie otrzymają 0 punktów. Proszę mierzyć siły na zamiary!

Testowanie rozwiązań

Żeby przetestować rozwiązania zadań należy wykonać:

```
python3 zad1.py

python3 zad2.py

python3 zad3.py
```

[2pkt.] Zadanie 1.

Szablon rozwiązania: `zad1.py`

Dana jest dwuwymiarowa tablica T o rozmiarach $N \times N$ wypełniona liczbami naturalnymi (liczby są parami różne). Proszę zaimplementować funkcję `Median(T)`, która przekształca tablicę T, tak aby elementy leżące pod główną przekątną nie były większe od elementów na głównej przekątnej, a elementy leżące nad główną przekątną nie były mniejsze od elementów na głównej przekątnej. Algorytm powinien być jak najszybszy oraz używać jak najmniej pamięci ponad tę, która potrzebna jest na przechowywanie danych wejściowych (choć algorytm nie musi działać w miejscu). Proszę podać złożoność czasową i pamięciową zaproponowanego algorytmu.

Przykład. Dla tablicy:

```
T = [ [ 2, 3, 5],
      [ 7,11,13],
      [17,19,23] ]
```

wynikiem jest, między innymi tablica:

```
T = [ [13,19,23],
      [ 3, 7,17],
      [ 5, 2,11] ]
```

[2pkt.] Zadanie 2.

Szablon rozwiązania: `zad2.py`

Węzły jednokierunkowej listy odsyłaczowej reprezentowane są w postaci:

```
class Node:
    def __init__(self):
        self.val  = None  # przechowywana liczba rzeczywista
        self.next = None  # odsyłacz do nastepnego elementu
```

Niech p będzie wskaźnikiem na niepustą listę odsyłaczową zawierającą parami różne liczby rzeczywiste $a_1, a_2, \ldots, a_n$ (lista nie ma wartownika). Mówimy, że lista jest k-chaotyczna jeśli dla każdego elementu zachodzi, że po posortowaniu listy znalazłby się na pozycji różniącej się od bieżącej o najwyżej k. Tak więc 0-chaotyczna lista jest posortowana, przykładem 1-chaotycznej listy jest $1, 0, 3, 2, 4, 6, 5$, a $(n-1)$-chaotyczna lista długości n może zawierać liczby w dowolnej kolejności. Proszę zaimplementować funkcję `SortH(p,k)`, która sortuje k-chaotyczną listę wskazywaną przez p. Funkcja powinna zwrócić wskazanie na posortowaną listę. Algorytm powinien być jak najszybszy oraz używać jak najmniej pamięci (w sensie asymptotycznym, mierzonym względem długości n listy oraz parametru k). Proszę oszacować jego złożoność czasową dla $k = \Theta(1)$, $k = \Theta(\log n)$ oraz $k = \Theta(n)$.

[2pkt.] Zadanie 3.

Szablon rozwiązania: `zad3.py`

Mamy daną N elementową tablicę T liczb rzeczywistych, w której liczby zostały wygenerowane z pewnego rozkładu losowego. Ten rozkład mamy zadany jako k przedziałów $[a_1, b_1], [a_2, b_2], \ldots, [a_k, b_k]$ takich, że i-ty przedział jest wybierany z prawdopodobieństwem c_i, a liczba z przedziału jest wybierana zgodnie z rozkładem jednostajnym. Przedziały mogą na siebie nachodzić, liczby a_i, b_i są liczbami naturalnymi ze zbioru $\{1, \ldots, N\}$. Proszę zaimplementować funkcję `SortTab(T,P)` sortująca podaną tablicę. Pierwszy argument to tablica do posortowania a drugi to opis przedziałów w postaci:

`P = [`(a_1, b_1, c_1)`, `(a_2, b_2, c_2)`, ..., `(a_k, b_k, c_k)`]`.

Na przykład dla wejścia:

```
P = [(1,5, 0.75) , (4,8, 0.25)]
T = [6.1, 1.2, 1.5, 3.5, 4.5, 2.5, 3.9, 7.8]
```

po wywołaniu `SortTab(T,P)` tablica `T` powinna być postaci:

```
T = [1.2, 1.5, 2.5, 3.5, 3.9, 4.5, 6.1, 7.8]
```

Algorytm powinien być możliwie jak najszybszy. Proszę podać złożoność czasową i pamięciową zaproponowanego algorytmu.