PPPD - Lab. 07

Zadanie punktowane, lab 07, grupa B, 2022/2023, autor: Piotr Wolszakiewicz

---

Uwaga: w rozwiązaniu zadania nie można używać append, slice, sort, sorted, ani indeksowania ujemnego. Listy tworzymy od razu w potrzebnym rozmiarze (w żaden sposób nie dodajemy nowych elementów, nie łączymy list).

Temat: Listy i złożoności

Treść zadania

Zadanie będzie polegało na zaimplementowaniu funkcji zdefiniowanych poniżej i odpowiednim przetestowaniu ich działania w głównej funkcji programu `main`. Funkcje można implementować niezależnie.

- *mix_with_inversion(arr)*
  Funkcja, która dla posortowanej rosnąco listy liczb całkowitych `arr` pozmienia kolejność w niej tak, aby pierwszy element był największy, drugi najmniejszy, trzeci drugi największy, czwarty drugi najmniejszy itd.
  Funkcja modyfikuje oryginalną listę i nic nie zwraca.
  np.

  ```
  arr = [1,2,3,4,5,6]
  Output: [6,1,5,2,4,3]
  ```

  Optymalna złożoność obliczeniowa O(n)
  Wymagana złożoność pamięciowa O(n)
  `2pkt`

- *find_first_letter_occurrence(arr, occurrence_count)*
  Funkcja przyjmuje jako parametr listę znaków `a-z` oraz parametr `occurrence_count` będący liczbą dodatnią typu `int`. Zwraca pierwszy element z tej listy, który wystąpi w niej `occurrence_count` razy. Jeśli nie ma takiego elementu zwraca `None`.
  Patrz pomocnicze funkcje: `number_to_letter` i `letter_to_number`
  np.

  ```
  arr=['m', 'i', 'n', 'i', 'm', 'i', 'm']
  occurrence_count = 3
  output = 'i'
  bo 'i' wystąpiło w liście 3 razy jako pierwsze
  ```

  Optymalna złożoność obliczeniowa O(n)
  Optymalna złożoność pamięciowa O(1)
  `2pkt`

- *get_dominators_from_right(arr)*
  Funkcja przyjmuje jako argument listę liczb całkowitych nieujemnych `arr`. Zwraca listę krotek(wartość dominatora, index i sumę elementów po jego prawej stronie) z informacjami o elementach listy, które są dominatorami. Element w tablicy jest dominatorem, jeśli wszystkie elementy po jego prawej stronie sumują się do wartości mniejszej od niego.

Funkcja nie może korzystać z dodatkowych list oprócz listy zwracanej, która ma być stworzona od razu w odpowiednim rozmiarze. Zwaracana lista powinna być posortowana względem indeksów dominatorów. np.

```
arr=[1, 9, 0, 4, 1, 2]
Output=[(9,1,7),(4,3,3)]
bo 0+4+1+2 < 9 oraz 1+2 < 4
```

Optymalna złożoność obliczeniowa O(n)
2pkt

- *join_for_max_number(arr)*
  Funkcja przyjmuje jako argument listę liczb całkowitych nieujemnych. Funkcja ma poukładać liczby w liście w taki sposób, aby liczba stworzona poprzez połączenie ich była największa. Patrz pomocnicza funkcja `get_digit_by_index`.

  Liczby w tablicy mogą się składać z 1-2 cyfr, a cyfry w liczbie się powtarzają

  ```
  np. [11, 2, 8, 99, 22, 0]
  Output: [99, 8, 2, 22, 11, 0] lub [99, 8, 22, 2, 11, 0]
  bo 998222110 jest największą możliwą liczbą
  ```

  Optymalna złożoność obliczeniowa O(nˆ2), z założeniem że wszystkie operatory arytmetyczne działają w czasie stałym.
  Optymalna złożoność pamięciowa O(1) 2pkt

Napisz program, który będzie implementował powyższe funkcje i testował ich działanie w głównej funkcji main. Patrz przykładowe testy z przykładowego wywołania.

Pomocnicze funkcje

```python
def number_to_letter(number):
    """Zamienia liczby z zakresu 0-25 na litery a-z"""
    return chr(number + ord('a'))

def letter_to_number(letter):
    """Zamienia litery z zakresu a-z na liczby 0-25"""
    return ord(letter) - ord('a')

def get_digit_by_index(number, index_from_left, invalid_result=-1):
    """
    Funkcja zwraca cyfrę z liczby number znajdującą się pod indeksem index_from_left
    (patrząc od lewej) bądź wartość invalid_result jeśli indeks jest nieprawidłowy
    """
    if index_from_left < 0:
        return invalid_result
    if number == 0 and index_from_left > 0:
        return invalid_result
    if number == 0:
        return 0

    digits_count = int(math.log10(number)) + 1
    if index_from_left + 1 > digits_count:
        return invalid_result

    return number // 10**(digits_count-1-index_from_left) % 10
```

```python
# Przykłady użycia
print(number_to_letter(0))     # a
print(number_to_letter(25))    # z

print(letter_to_number('a'))         # 0
print(letter_to_number('z'))         # 25

print(get_digit_by_index(321, 0))    # 3
print(get_digit_by_index(321, 1))    # 2
print(get_digit_by_index(321, 2))    # 1
print(get_digit_by_index(321, 3))    # -1
```

Punktacja

Za poszczególne elementy można uzyskać następującą liczbę punktów:

- poprawne zaimplementowanie funkcji `main` z testowaniem zaimplementowanych funkcji - `2pkt`
- `mix_with_inversion(arr)`- dla wymaganej złożoności `2pkt`
- `find_first_letter_occurrence(arr, occurrence_count)` - dla optymalnej złożoności `2pkt`
- `get_dominators_from_right(arr)` - dla optymalnej złożoności `2pkt`
- `join_for_max_number(arr)` - dla optymalnej złożoności `2pkt`

Uwaga

- Jeśli rozwiązanie nie spełnia wymagań `złożoności obliczeniowej` lub `pamięciowej` maksymalna ilość punktów wynosi `1/2` oryginalnej.
- Jeśli rozwiązanie nie spełnia postawionych wymagań (korzysta z zabronionych funkcji), zadanie jest oceniane na `0 punktów`.
- Jeśli program się nie kompiluje (interpretuje), ocena jest zmniejszana o połowę.
- Jeśli kod programu jest niskiej jakości (nieestetycznie formatowanie, mylące nazwy zmiennych itp.), ocena jest zmniejszana o `2p`.

Przykłady interakcji użytkownika z programem

Przykład wywołania:

```
-------------mix_array_with_inversion-------------
1. mix_array_with_inversion([1, 2, 3, 4, 5, 6])
Output: [6, 1, 5, 2, 4, 3]
2. mix_array_with_inversion([1, 2, 3, 4, 5, 6, 7, 8, 9])
Output: [9, 1, 8, 2, 7, 3, 6, 4, 5]
-------------find_first_letter_occurrence-------------
1. find_first_letter_occurrence(['z', 's', 'a', 'z', 't', 'd', 'z', 'a', 'u', 'a', 'a', 't', 'a',
'z', 'z', 'c', 'q', 'c', 'x', 's', 'z', 'e', 'a', 'q', 'a', 'k', 'a', 'd', 'o', 'a'], 4)
Output: a
2. find_first_letter_occurrence(['a', 'b', 'c'], 2)
Output: None
-------------get_dominators_from_right-------------
1. get_dominators_from_right([1, 9, 0, 4, 1, 2])
Output: [(9, 1, 7), (4, 3, 3)]
2. get_dominators_from_right([1, 2, 3, 4, 5, 6])
Output: []
-------------join_for_max_number-------------
1. join_for_max_number([11, 2, 8, 99, 22, 0])
```

```
Output: [99, 8, 2, 22, 11, 0]
-------------join_for_max_number* (dla dowolnych liczb)-------------
2. join_for_max_number([15, 5, 9, 51, 83, 4, 837, 0])
Output: [9, 83, 837, 5, 51, 4, 15, 0]
```