Algorytmy i Struktury Danych
Egzamin (3. VII 2020)

Format rozwiązań

Rozwiązanie każdego zadania musi składać się 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). Implementacja musi być zgodna z szablonem kodu źródłowego dostarczonym wraz z zadaniem. Niedopuszczalne jest w szczególności:

1. zmiana nazwy funkcji implementującej algorytm lub listy jej argumentów,
2. modyfikacja 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).

Dopuszczalne jest natomiast:

1. korzystanie z następujących elementarnych struktur danych: krotka, lista, kolejka `collections.deque`,
2. korzystanie z wbudowanych algorytmów sortowania (poza zadaniem 3),
3. korzystanie ze struktur danych dostarczonych razem z zadaniem.

Wszystkie inne algorytmy lub struktury danych (w tym słowniki) wymagają implementacji. Dopuszczalne jest oczywiście implementowanie dodatkowych funkcji pomocniczych w pliku z szablonem rozwiązania. Jeśli ktoś zaimplementuje standardowe drzewo BST, to może w analizie zakładać, że złożoność operacji na nim jest rzędu $O(\log n)$.

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 za to poprawne, mają szanse na otrzymanie 1 punktu. Rozwiązania próbujące osiągnąć jak najlepszą złożoność, ale zaimplementowane błędnie otrzymają 0 punktów. Proszę mierzyć siły na zamiary!

Testowanie rozwiązań

Żeby przetestować rozwiązanie danego zadania należy wykonać odpowiednio:

```
python3 zad1.py

python3 zad2.py

python3 zad3.py
```

[2pkt.] Zadanie 1.

Szablon rozwiązania: `zad1.py`

Pewna kraina składa się z wysp pomiędzy którymi istnieją połączenia lotnicze, promowe oraz mosty. Pomiędzy dwoma wyspami istnieje co najwyżej jeden rodzaj połączenia. Koszt przelotu z wyspy na wyspę wynosi 8฿, koszt przeprawy promowej wynosi 5฿, za przejście mostem trzeba wnieść opłatę 1฿. Poszukujemy trasy z wyspy `A` na wyspę `B`, która na kolejnych wyspach zmienia środek transportu na inny oraz minimalizuje koszt podróży.

Dana jest tablica `G`, określająca koszt połączeń pomiędzy wyspami. Wartość 0 w macierzy oznacza brak bezpośredniego połączenia. Proszę zaimplementować funkcję `islands( G, A, B )` zwracającą minimalny koszt podróży z wyspy `A` na wyspę `B`. Jeżeli trasa spełniająca warunki zadania nie istnieje, funkcja powinna zwrócić wartość `None`.

Przykład Dla tablicy

```
G1 = [ [0,5,1,8,0,0,0 ],
       [5,0,0,1,0,8,0 ],
       [1,0,0,8,0,0,8 ],
       [8,1,8,0,5,0,1 ],
       [0,0,0,5,0,1,0 ],
       [0,8,0,0,1,0,5 ],
       [0,0,8,1,0,5,0 ] ]
```

funkcja `islands(G1, 5, 2)` powinna zwrócić wartość `13`.

[2pkt.] Zadanie 2.

Szablon rozwiązania: `zad2.py`

Asystent znanego profesora otrzymał polecenie wyliczenia sumy pewnego ciągu liczb (liczby mogą być zarówno dodatnie jak i ujemne):

$$n_1 + n_2 + ... + n_k$$

Aby zminimalizować błędy zaokrągleń asystent postanowił wykonać powyższe dodawania w takiej kolejności, by największy co do wartości bezwzględnej wynik tymczasowy (wynik każdej operacji dodawania; wartość końcowej sumy również traktujemy jak wynik tymczasowy) był możliwie jak najmniejszy. Aby ułatwić sobie zadanie, asystent nie zmienia kolejności liczb w sumie a jedynie wybiera kolejność dodawań.

Napisz funkcję `opt_sum`, która przyjmuje tablicę liczb $n_1, n_2, \ldots, n_k$ (w kolejności w jakiej występują w sumie; zakładamy, że tablica zwiera co najmniej dwie liczby) i zwraca największą wartość bezwzględną wyniku tymczasowego w optymalnej kolejności dodawań. Na przykład dla tablicy wejściowej:

$$[1, -5, 2]$$

funkcja powinna zwrócić wartość 3, co odpowiada dodaniu -5 i 2 a następnie dodaniu 1 do wyniku.

Uzasadnij poprawność zaproponowanego rozwiązania i oszacuj jego złożoność obliczeniową. Nagłówek funkcji `opt_sum` powinien mieć postać:

```
def opt_sum(tab):
  ...
```

[2pkt.] Zadanie 3.

Szablon rozwiązania: `zad2.py`

Pewien eksperyment fizyczny daje w wyniku liczby rzeczywiste postaci a^x, gdzie a to pewna stała większa od 1 ($a > 1$) zaś x to liczby rzeczywiste rozłożone równomiernie na przedziale $[0, 1]$. Napisz funkcję `fast_sort`, która przyjmuje tablicę liczb z wynikami eksperymentu oraz stałą a i zwraca tablicę z wynikami eksperymentu posortowanymi rosnąco. Funkcja powinna działać możliwie jak najszybciej. Uzasadnij poprawność zaproponowanego rozwiązania i oszacuj jego złożoność obliczeniową. Nagłówek funkcji `fast_sort` powinien mieć postać:

```
def fast_sort(tab, a):
    ...
```