Algorytmy i Struktury Danych
Kolokwium Zaliczeniowe 2 (1.IX 2021)

Format rozwiązań

Rozwiązanie każdego zadania musi się składać z **krótkiego** 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 zaawansowanych struktur danych (np. słowników czy zbiorów).

Dopuszczalne jest natomiast:

1. korzystanie z następujących elementarnych struktur danych: krotka, lista, kolejka `collections.deque`, kolejka priorytetowa (`queue.PriorityQueue`),
2. korzystanie ze struktur danych dostarczonych razem z zadaniem (jeśli takie są).
3. korzystanie z wbudowanych funkcji sortujących (można założyć, że mają złożoność $O(n \log n)$).

Wszystkie inne algorytmy lub struktury danych wymagają implementacji przez studenta. Dopuszczalne jest oczywiście implementowanie dodatkowych funkcji pomocniczych w pliku z szablonem rozwiązania.

Zadania niezgodne z powyższymi ograniczeniami otrzymają ocenę 0 punktów. 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 błędne otrzymają 0 punktów.

Testowanie rozwiązań

Żeby przetestować rozwiązania zadań należy wykonać:

```
python3 zad1.py

python3 zad2.py

python3 zad3.py
```

[2pkt.] Zadanie 1.

Szablon rozwiązania:	`zad1.py`
Pierwszy próg złożoności:	$O(N \log N)$
Drugi próg złożoności:	$O(N)$

Dany jest zbiór N prostokątów o bokach równoległych do osi układu współrzędnych. Proszę zaimplementować funkcję:

```
def rect( D ):
  ...
```

która wskaże, który prostokąt należy usunąć tak, żeby przecięcie pozostałych miało jak największe pole. Każdy prostokąt opisuje czwórka liczb całkowitych (x_1, y_1, x_2, y_2) określających współrzędne lewego dolnego i prawego górnego rogu prostokąta. Funkcja otrzymuje listę takich czwórek i powinna zwrócić najmniejszy numer prostokąta, który należy usunąć.

Funkcja powinna być możliwie jak najszybsza. Proszę oszacować złożoność czasową i pamięciową użytego algorytmu.

Przykład. Dla listy:

```
D = [(2,3,10,6),(3,1,8,8),(5,4,9,7)]
```

prawidłowym wynikiem jest liczba 2.

[2pkt.] Zadanie 2.

Szablon rozwiązania:	`zad2.py`
Pierwszy próg złożoności:	$O(N + M^2)$
Drugi próg złożoności:	$O(N + M)$

Na osi liczbowej znajduje się N punktów większych od $M = 10^K$. Z punktu A można przeskoczyć na punkt B wtedy i tylko wtedy gdy $A\%10^K == B//10^K$. Proszę zaimplementować funkcję:

```
def order( L,K ):
  ...
```

porządkującą punkty, tak aby możliwe było przejście od najwcześniejszego punktu w tym porządku, kolejno przez wszystkie punkty, do ostatniego. Funkcja otrzymuje listę wartości określającą położenie punktów na osi liczbowej i powinna zwrócić listę punktów w kolejności ich odwiedzania. Jeżeli uporządkowanie punktów nie jest możliwe, funkcja powinna zwrócić $None$.

Funkcja powinna być możliwie jak najszybsza. Proszę oszacować złożoność czasową i pamięciową użytego algorytmu.

Przykład. Dla danych:

```
L = [56,15,31,43,54,35,12,23], K = 1
```

przykładowym, prawidłowym wynikiem jest lista:

```
L = [12,23,31,15,54,43,35,56]
```

[2pkt.] Zadanie 3.

Szablon rozwiązania:	`zad3.py`
Pierwszy próg złożoności:	$O(\|V\|^3)$
Drugi próg złożoności:	$O(\|E\| \log \|V\|)$

Dany jest nieskierowany graf $G = (V, E)$ oraz dwa wierzchołki, s i t. Proszę zaimplementować funkcję:

```
def paths( G,s,t ):
  ...
```

która zwraca liczbę krawędzi e takich, że e występuje na pewnej najkrótszej ścieżce z s do t. Graf dany jest jako lista list sąsiedztwa w postaci $[(v_0, w_0), (v_1, w_1), ...]$, gdzie: v_i to numer wierzchołka, w_i to waga krawędzi prowadzącej do wierzchołka v_i. Wagi krawędzi są dodatnie.

Funkcja powinna być możliwie jak najszybsza. Proszę oszacować złożoność czasową i pamięciową użytego algorytmu.

Przykład. Dla listy sąsiedztwa postaci:

```
G = [ [(1,2),(2,4)],                # sąsiedzi i wagi wierzchołka nr 0
      [(0,2),(3,11),(4,3)],         # sąsiedzi i wagi wierzchołka nr 1
      [(0,4),(3,13)],               # itd.
      [(1,11),(2,13),(5,17),(6,1)],
      [(1,3),(5,5)],
      [(3,17),(4,5),(7,7)],
      [(3,1),(7,3)],
      [(5,7),(6,3)] ],
s = 0, t = 7
```

funkcja powinna zwrócić wartość 7. Krawędzie 0-1, 1-4, 4-5, 5-7, 1-3, 3-6, 6-7.