Algorytmy i Struktury Danych
Kolokwium Zaliczeniowe I (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. 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).

Dopuszczalne jest natomiast:

1. korzystanie z następujących elementarnych struktur danych: krotka, lista, kolejka `collections.deque`,
2. korzystanie z wbudowanych algorytmów sortowania,
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 rozwiązania zadań należy wykonać:

```
python3 zad1.py

python3 zad2.py

python3 zad3.py
```

[2pkt.] Zadanie 1.

Szablon rozwiązania: `zad1.py`

Dana jest tablica dwuwymiarowa `G`, przedstawiająca macierz sąsiedztwa skierowanego grafu ważonego, który odpowiada mapie drogowej (wagi przedstawiają odległości, liczba `-1` oznacza brak krawędzi). W niektórych wierzchołkach są stacje paliw, podana jest ich lista `P`. Pełnego baku wystarczy na przejechanie odległości `d`. Wjeżdżając na stację samochód zawsze jest tankowany do pełna. Proszę zaimplemntować funkcję `jak_dojade(G, P, d, a, b)`, która szuka najkrótszej możliwej trasy od wierzchołka `a` do wierzchołka `b`, jeśli taka istnieje, i zwraca listę kolejnych odwiedzanych na trasie wierzchołków (zakładamy, że w `a` też jest stacja paliw; samochód może przejechać najwyżej odległość `d` bez tankowania).

Zaproponowana funkcja powinna być możliwe jak najszybsza. Uzasadnij jej poprawność i oszacuj złożoność obliczeniową.

Przykład Dla tablic

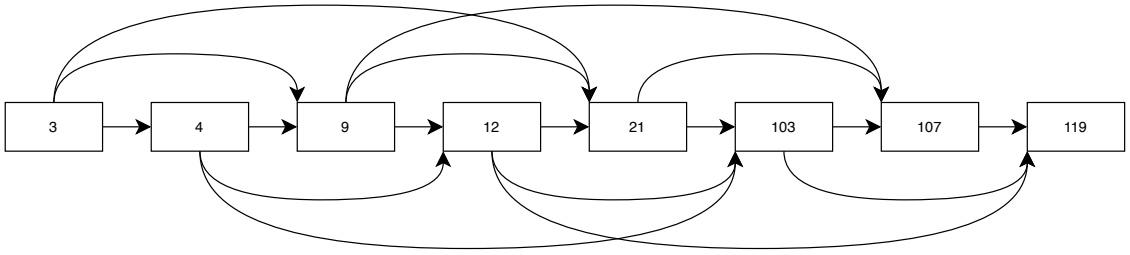
```
G = [[-1, 6,-1, 5, 2],
     [-1,-1, 1, 2,-1],
     [-1,-1,-1,-1,-1],
     [-1,-1, 4,-1,-1],
     [-1,-1, 8,-1,-1]]
P = [0,1,3]
```

funkcja `jak_dojade(G, P, 5, 0, 2)` powinna zwrócić `[0,3,2]`. Dla tych samych tablic funkcja `jak_dojade(G, P, 6, 0, 2)` powinna zwrócić `[0,1,2]`, natomiast `jak_dojade(G, P, 3, 0, 2)` powinna zwrócić `None`.

[2pkt.] Zadanie 2.

Szablon rozwiązania: `zad2.py`

W szybkiej liscie odsyłaczowej i-ty element posiada referencje (odsyłacze) do elementów: $i+2^0$, $i+2^1$, $i+2^2$, ... (lista odsyłaczy z i-tego elementu kończy się na ostatnim elemencie o numerze postaci $i+2^k$, który występuje w liście). Lista ta przechowuje liczby całkowite w kolejności niemalejącej. Przykładową szybką listę przedstawia poniższy rysunek:



Napisz funkcję `fast_list_prepend`:

```
def fast_list_prepend(L, a):
  ...
```

która przyjmuje referencję na pierwszy węzeł szybkiej listy (`L`) oraz liczbę całkowitą (`a`) mniejszą od wszystkich liczb w przekazanej liście i wstawia tę liczbę na początek szybkiej listy (jako nowy węzeł). W wyniku dodania nowego elementu powinna powstać prawidłowa szybka lista. W szczególności każdy węzeł powinien mieć poprawne odsyłacze do innych węzłów. Funkcja powinna zwrócić referencję na nowy pierwszy węzeł szybkiej listy.

Zaproponowana funkcja powinna być możliwe jak najszybsza. Uzasadnij jej poprawność i oszacuj złożoność obliczeniową. Węzły szybkiej listy reprezentowane są w postaci:

```
class FastListNode:
    def __init__(self, a):
        self.a = a     # przechowywana liczba całkowita
        self.next = [] # lista odsyłaczy do innych elementów; początkowo pusta

    def __str__(self): # zwraca zawartość węzła w postaci napisu
        res = 'a: ' + str(self.a) + '\t' + 'next keys: '
        res += str([n.a for n in self.next])
        return res
```

[2pkt.] Zadanie 3.

Szablon rozwiązania: `zad3.py`

Dana jest tablica `A` zawierająca `n = len(A)` liczb naturalnych. Dodatkowo wiadomo, że `A` w sumie zawiera `k` różnych liczb (należy założyć, że `k` jest dużo mniejsze niż `n`). Proszę zaimplementować funkcję `longest_incomplete(A, k)`, która zwraca długość najdłuższego spójnego ciągu elementów z tablicy `A`, w którym nie występuje wszystkie `k` liczb. (Można założyć, że podana wartość `k` jest zawsze prawidłowa.)

Przykład Dla tablicy

```
A = [1,100, 5, 100, 1, 5, 1, 5]
```

wartością wywołania `longest_incomplete(A, 3)` powinno być `4` (ciąg `1, 5, 1, 5` z końca tablicy).