Algorytmy i Struktury Danych
Kolokwium 1 (23. IV 2020)

Format rozwiązań

Rozwiązania zadań opisowych powinny być wysłane jako pojedyncze pliki tekstowe (rozszerzenie `.txt`). Rozwiązania zadań implementacyjnych powinny być wysłane jako pliki źródłowe Pythona (rozszerzenie `.py`). Krótkie wyjaśnienia i oszacowanie złożoności w zadaniach implementacyjnych powinny być w formie komentarza Pythonowego. W pierwszej linijce każdego rozwiązania należy podać swoje imię i nazwisko. Rozwiązania w innych formatach (np. `.PDF`, `.DOC`, `.PNG`, `.JPG`) z definicji nie będą sprawdzane i otrzymają ocenę 0 pkt. nawet jeśli będą poprawne. W szczególnośći niedopuszczalne jest rozwiązanie zadania na kartce i wysłanie jej zdjęcia.

Python

Kod w Pythonie powienien korzystać tylko z elementarnych fragmentów języka. Wolno korzystać z wewnętrznych funkcji sortujących. Wszystkie użyte struktury danych należy zaimplementować samodzielnie (poza listami Pythonowymi traktowanymi jako tablice; np. `A = [0]*n` jest dopuszczalne).

Zadania

[**2pkt.**] **Zadanie 1.** Proszę zaimplementować funkcję `heavy_path(T)`, która na wejściu otrzymuje drzewo `T` z ważonymi krawędziami (wagi to liczby całkowite—mogą być zarówno dodatnie, ujemne, jak i o wartości zero) i zwraca długość (wagę) najdłuższej ścieżki prostej w tym drzewie. Drzewo reprezentowane jest za pomocą obiektów typu `Node`:

```
class Node:
  def __init__( self ):
    self.children = 0   # liczba dzieci węzła
    self.child = []     # lista par (dziecko, waga krawędzi)
    ...                 # wolno dopisać własne pola
```

Poniższy kod tworzy drzewo z korzeniem `A`, który ma dwoje dzieci, węzły `B` i `C`, do których prowadzą krawędzie o wagach 5 i −1:

```
A = Node()
B = Node()
C = Node()
A.children = 2
A.child = [ (B,5), (C,-1) ]
```

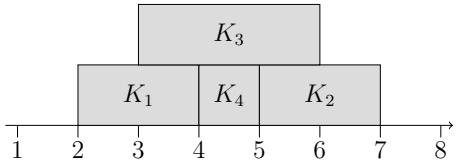
Rozwiązaniem dla drzewa `A` jest 5 (osiągnięte przez ścieżkę `A-B`; ścieżka `B-A-C` ma wagę $5 - 1 = 4$. Proszę skrótowo wyjaśnić ideę algorytmu oraz oszacować jego złożoność czasową.

[2pkt.] Zadanie 2. Algocja leży na wielkiej pustyni i składa się z miast oraz oaz połączonych drogami. Każde miasto jest otoczone murem i ma tylko dwie bramy—północną i południową. Z każdej bramy prowadzi dokładnie jedna droga do jednej oazy (ale do danej oazy może dochodzić dowolnie wiele dróg; oazy mogą też być połączone drogami między sobą). Prawo Algocji wymaga, że jeśli ktoś wjechał do miasta jedną bramą, to musi go opuścić drugą. Szach Algocji postanowił wysłać gońca, który w każdym mieście kraju odczyta zakaz formułowania zadań "o szachownicy" (obraza majestatu). Szach chce, żeby goniec odwiedził każde miasto dokładnie raz (ale nie ma ograniczeń na to ile razy odwiedzi każdą z oaz). Goniec wyjeżdża ze stolicji Algocji, miasta x, i po odwiedzeniu wszystkich miast ma do niej wrócić. Proszę przedstawić (bez implementacji) algorytm, który stwierdza czy odpowiednia trasa gońca istnieje. Proszę uzasadnić poprawność algorytmu oraz oszacować jego złożoność czasową.

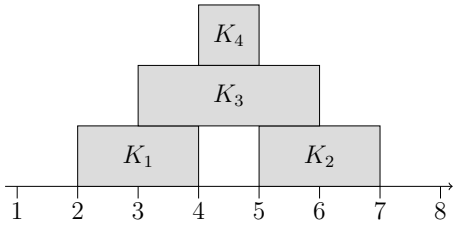
[2pkt.] Zadanie 3. Dany jest zbiór klocków $\mathcal{K} = \{K_1, \ldots, K_n\}$. Każdy klocek K_i opisany jest jako jednostronnie domknięty przedział $(a_i, b_i]$, gdzie $a_i, b_i \in \mathbb{N}$, i ma wysokość 1 (należy założyć, że żadne dwa klocki nie zaczynają się w tym samym punkcie, czyli wartości a_i są parami różne). Klocki są zrzucane na oś liczbową w pewnej kolejności. Gdy tylko spadający klocek dotyka innego klocka (powierzchnią poziomą), to jest do niego trwale doczepiany i przestaje spadać. Kolejność spadania klocków jest *poprawna* jeśli każdy klocek albo w całości ląduje na osi liczbowej, albo w całości ląduje na innych klockach. Rozważmy przykładowy zbiór klocków $\mathcal{K} = \{K_1, K_2, K_3, K_4\}$, gdzie:

$$K_1 = (2, 4], \quad K_2 = (5, 7], \quad K_3 = (3, 6], \quad K_4 = (4, 5].$$

Kolejność K_1, K_4, K_2, K_3 jest poprawna (choć są też inne poprawne kolejności) podczas gdy kolejność K_1, K_2, K_3, K_4 poprawna nie jest (K_3 nie leży w całości na innych klockach):



(a) Klocki po tym, jak spadały w poprawnej kolejności K_1, K_4, K_2, K_3.



(b) Klocki po tym, jak spadały w niepoprawnej kolejności K_1, K_2, K_3, K_4.

Proszę podać algorytm (bez implementacji), który sprawdza czy dla danego zbioru klocków istnieje poprawna kolejność spadania. Proszę uzasadnić poprawność algorytmu oraz oszacować jego złożoność. Proszę także odpowiedzieć na następujące pytanie:

> Czy jeśli początki klocków nie muszą być parami różne to algorytm dalej działa poprawnie? Jeśli tak, proszę to uzasadnić. Jeśli nie, to proszę podać kontrprzykład.