Algorytmy i Struktury Danych
Kolokwium III (14.VI 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`

Carol musi przewieźć pewne niebezpieczne substancje z laboratorium x do laboratorium y, podczas gdy Max musi zrobić to samo, ale w przeciwną stronę. Problem polega na tym, że jeśli substancje te znajdą się zbyt blisko siebie, to nastąpi reakcja w wyniku której absolutnie nic się nie stanie (ale szefowie Carol i Max nie chcą do tego dopuścić, by nie okazało się, że ich praca nie jest nikomu potrzebna). Zaproponuj, uzasadnij i zaimplementuj algorytm planujący jednocześnie trasy Carol i Maxa tak, by odległość między nimi zawsze wynosiła co najmniej d. Mapa połączeń dana jest jako graf nieskierowany, w którym każda krawędź ma dodatnią wagę (x i y to wierzchołki w tym grafie). W jednostce czasu Carol i Max pokonują dokładnie jedną krawędź. Jeśli trzeba, dowolne z nich może się w danym kroku zatrzymać (wówczas pozostaje w tym samym wierzchołku). Carol i Max nie mogą równocześnie poruszać się tą samą krawędzią (w przeciwnych kierunkach).

Rozwiązanie należy zaimplementować w postaci funkcji:

```
def keep_distance(M, x, y, d):
   ...
```

która przyjmuje numery wierzchołków `x` oraz `y`, minimalną odległość `d` i graf reprezentowany przez kwadratową, symetryczną macierz sąsiedztwa `M`. Wartość `M[i][j] == M[j][i]` to długość krawędzi między wierzchołkami `i` oraz `j`, przy czym `M[i][j] == 0` oznacza brak krawędzi między wierzchołkami. W macierzy nie ma wartości ujemnych. Funkcja powinna zwrócić listę krotek postaci:

```
[(x, y), (u1, v1), (u2, v2), ..., (uk, vk), (y, x)]
```

reprezentującą ścieżki Carol i Max. W powyższej liście element `(ui, vi)` oznacza, że Carol znajduje się w wierzchołku `ui`, zaś Max w wierzchołku `vi`. Można założyć, że rozwiązanie istnieje.

Przykład. Dla argumentów:

```
M = [
      [0, 1, 1, 0],
      [1, 0, 0, 1],
      [1, 0, 0, 1],
      [0, 1, 1, 0],
    ]
x = 0
y = 3
d = 2
```

wynikiem jest na przykład lista: `[(0, 3), (1, 2), (3, 0) ]`

Podpowiedź. Proszę rozważyć nowy graf, być może z dużo większą liczbą wierzchołków niż graf wejściowy.

[2pkt.] Zadanie 2.

Szablon rozwiązania: `zad2.py`

Dane jest drzewo BST zbudowane z węzłów

```
class BNode:
  def __init__( self, value ):
    self.left = None
    self.right = None
    self.parent = None
    self.value = value
```

Klucze w tym drzewie znajdują się w polach `value` i są liczbami całkowitymi. Mogą zatem mieć wartości zarówno dodatnie, jak i ujemne. Proszę napisać funkcję, która zwraca wartość będącą minimalną możliwą sumą kluczy zbioru wierzchołków oddzielających wszystkie liście od korzenia w taki sposób, że na każdej ścieżce od korzenia do liścia znajduje się dokładnie jeden wierzchołek z tego zbioru. Zakładamy że korzeń danego drzewa nie jest bezpośrednio połączony z żadnym liściem (ścieżka od korzenia do każdego liścia prowadzi przez co najmniej jeden dodatkowy węzeł). Jako liść jest rozumiany węzeł `W` typu `BNode` such that `W.left = W.right = None`.

Rozwiązanie należy zaimplementować w postaci funkcji:

```
def cutthetree(T):
   ...
```

która przyjmuje korzeń danego drzewa BST i zwraca wartość rozwiązania. Nie wolno zmieniać definicji `class BNode`.

Przykład. Dla drzewa BST, utworzonego przez dodawanie do pustego drzewa kolejno elementów z kluczami 10, 3, 15, 11, 17, -1, -5, 0 wynikiem jest 14 (usuwamy węzły o wartośćiach -1 oraz 15).

[2pkt.] Zadanie 3.

Szablon rozwiązania: `zad3.py`

Dany jest ważony graf nieskierowany reprezentowany przez macierz T o rozmiarach $n \times n$ (dla każdych i, j zachodzi $T[i][j] = T[j][i]$; wartość $T[i][j] > 0$ oznacza, że istnieje krawędź między wierzchołkiem i a wierzchołkiem j z wagą $T[i][j]$). Dana jest także liczba rzeczywista d. Każdy wierzchołek w G ma jeden z kolorów: zielony lub niebieski. Zaproponuj algorytm, który wyznacza największą liczbę naturalną ℓ, taką że w grafie istnieje ℓ par wierzchołków $(p, q) \in V \times V$ spełniających warunki:

1. q jest zielony, zaś p jest niebieski,
2. odległość między p i q (liczona jako suma wag krawędzi najkrótszej ścieżki) jest nie mniejsza niż d,
3. każdy wierzchołek występuje w co najwyżej jednej parze.

Rozwiązanie należy zaimplementować w postaci funkcji:

```
def BlueAndGreen(T, K, D):
    ...
```

która przyjmuje:

T: graf reprezentowany przez kwadratową macierz sąsiedztwa, gdzie wartość 0 oznacza brak krawędzi, a liczba większa od 0 przedstawia odległość pomiędzy wierzchołkami,

K: listę przedstawiającą kolory wierzchołków,

D: odległość o której mowa w warunku 2 opisu zadania.

Funkcja powinna zwrócić liczbę ℓ omawianą w treści zadania.

Przykład. Dla argumentów:

```
T = [
      [0, 1, 1, 0, 1],
      [1, 0, 0, 1, 0],
      [1, 0, 0, 0, 1],
      [0, 1, 0, 0, 1],
      [1, 0, 1, 1, 0],
    ]
K = ['B', 'B', 'G', 'G', 'B']
D = 2
```

wynikiem jest wartość 2.

Maksymalny przepływ. Do treści zadania dostarczony jest plik zad3EK.py, w którym zaimplementowany jest algorytm Edmondsa-Karpa obliczający maksymalny przepływ w grafie, w następującej postaci:

```
edmonds_karp(graph, source, sink)
```

który przyjmuje:

graph: graf reprezentowany przez kwadratową macierz sąsiedztwa, gdzie wartość oznacza pojemność (ang. capacity) danej krawędzi,

source: numer wierzchołka-źródła,

sink: numer wierzchołka-ujścia.