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):
    ...
```

Algorytmy i Struktury Danych
Egzamin Poprawkowy (1. IX 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. Nie dopuszczalne 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,
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 poprawne, mają szanse na otrzymanie 1 punktu. Rozwiązania szybsze, ale błędne 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`

Żab Zbigniew skacze po osi liczbowej. Ma się dostać z zera do $n-1$, skacząc wyłącznie w kierunku większych liczb. Skok z liczby i do liczby j ($j > i$) kosztuje Zbigniewa $j-i$ jednostek energii, a jego energia nigdy nie może spaść poniżej zera. Na początku Zbigniew ma 0 jednostek energii, ale na szczęście na niektórych liczbach—także na zerze—leżą przekąski o określonej wartości energetycznej (wartość przekąki dodaje się do aktualnej energii Zbigniewa).

Proszę zaimplementować funkcję `zbigniew(A)`, która otrzymuje na wejściu tablicę `A` długości `len(A) = n`, gdzie każde pole zawiera wartość energetyczną przekąski leżącej na odpowiedniej liczbie. Funkcja powinna zwrócić minimalną liczbę skoków potrzebną, żeby Zbigniew dotarł z zera do `n-1` lub -1 jeśli nie jest to możliwe.

Podpowiedź. Warto rozważyć funkcję $f(i, y)$ zwracającą minimalną liczbę skoków potrzebną by dotrzeć do liczby i mając w zapasie dokładnie y jednostek energii.

Przykład. Dla tablicy `A = [2,2,1,0,0,0]` wynikiem jest 3 (Zbigniew skacze z 0 na 1, z 1 na 2 i z 2 na 5, kończąc z zerową energią). Dla tablicy `A = [4,5,2,4,1,2,1,0]` wynikiem jest 2 (Zbigniew skacze z 0 na 3 i z 3 na 7, kończąc z jedną jednostką energii).

[2pkt.] Zadanie 2.

Szablon rozwiązania: `zad2.py`

W pewnym państwie, w którym znajduje się N miast, postanowiono połączyć wszystkie miasta siecią autostrad, tak aby możliwe było dotarcie autostradą do każdego miasta. Ponieważ kontynent, na którym leży państwo jest płaski położenie każdego z miast opisują dwie liczby x, y, a odległość w linii prostej pomiędzy miastami liczona w kilometrach wyraża się wzorem $len = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$. Z uwagi na oszczędności materiałów autostrada łączy dwa miasta w linii prostej.

Ponieważ zbliżają się wybory prezydenta, wszystkie autostrady zaczęto budować równocześnie i jako cel postanowiono zminimalizować czas pomiędzy otwarciem pierwszej i ostatniej autostrady. Czas budowy autostrady wyrażony w dniach wynosi $\lceil len \rceil$ (sufit z długości autostrady wyrażonej w km).

Proszę zaimplementować funkcję `highway(A)`, która dla danych położeń miast wyznacza minimalną liczbę dni dzielącą otwarcie pierwszej i ostatniej autostrady.

Przykład Dla tablicy `A =[(10,10),(15,25),(20,20),(30,40)]` wynikiem jest 7 (Autostrady pomiędzy miastami 0-1, 0-2, 1-3).

[2pkt.] Zadanie 3.

Szablon rozwiązania: `zad3.py`

Dany jest zbiór N zadań, gdzie niektóre zadania muszą być wykonane przed innymi zadaniami. Wzajemne kolejności zadań opisuje dwuwymiarowa tablica $T[N][N]$. Jeżeli $T[a][b] = 1$ to wykonanie zadania a musi poprzedzać wykonanie zadania b. W przypadku gdy $T[a][b] = 2$ zadanie b musi być wykonane wcześniej, a gdy $T[a][b] = 0$ kolejność zadań a i b jest obojętna. Proszę zaimplementować funkcję `tasks(T)`, która dla danej tablicy T, zwraca tablicę z kolejnymi numerami zadań do wykonania.

Przykład Dla tablicy `T = [ [0,2,1,1], [1,0,1,1], [2,2,0,1], [2,2,2,0] ]` wynikiem jest tablica `[1,0,2,3]`.

Algorytmy i Struktury Danych
Egzamin Poprawkowy 2 (15. IX 2020)

Imię i nazwisko: _____ A

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. Nie dopuszczalne 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 poprawne, mają szanse na otrzymanie 1 punktu. Rozwiązania szybsze, ale błędne 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`

Każdy nieskierowany, spójny i acykliczny graf $G = (V, E)$ możemy traktować jako drzewo. Korzeniem tego drzewa może być dowolny wierzchołek $v \in V$. Napisz funkcję `best_root(L)`, która przyjmuje nieskierowany, spójny i acykliczny graf G (reprezentowany w postaci listy sąsiedztwa) i wybiera taki jego wierzchołek, by wysokość zakorzenionego w tym wierzchołku drzewa była możliwie najmniejsza. Jeśli kilka wierzchołków spełnia warunki zadania, funkcja może zwrócić dowolny z nich. Wysokość drzewa definiujemy jako liczbę krawędzi od korzenia do najdalszego liścia. Uzasadnij poprawność zaproponowanego algorytmu i oszacuj jego złożoność obliczeniową.

Funkcja `best_root(L)` powinna zwrócić numer wierzchołka wybranego jako korzeń. Wierzchołki numerujemy od 0. Argumentem `best_root(L)` jest lista postaci:

$$L = [l_0, l_1, \ldots, l_{n-1}],$$

gdzie l_i to lista zawierająca numery wierzchołków będących sąsiadami $i-$tego wierzchołka. Można przyjąć (bez weryfikacji), że lista opisuje graf spełniający warunki zadania. W szczególności, graf jest spójny, acykliczny, oraz jeśli $a \in l_b$ to $b \in l_a$ (graf jest nieskierowany). Nagłówek funkcji powinien mieć postać:

```
def best_root(L):
  ...
```

Przykład. Dla listy sąsiedztwa postaci:

```
L = [ [ 2 ],
      [ 2 ],
      [ 0, 1, 3],
      [ 2, 4 ],
      [ 3, 5, 6 ],
      [ 4 ],
      [ 4 ] ]
```

funkcja powinna zwrócić wartość 3.

[2pkt.] Zadanie 2.

Szablon rozwiązania: `zad2.py`

Dany jest ciąg klocków (a_1, b_1), ... (a_n, b_n). Każdy klocek zaczyna się na pozycji a_i i ciągnie się do pozycji b_i. Klocki mogą spadać w kolejności takiej jak w ciągu. Proszę zaimplementować funkcję `tower(A)`, która wybiera możliwie najdłuższy podciąg klocków taki, że spadając tworzą wieżę i żaden klocek nie wystaje poza którykolwiek z wcześniejszych klocków. Do funkcji przekazujemy tablicę `A` zawierającą pozycje klocków a_i,b_i. Funkcja powinna zwrócić maksymalną wysokość wieży jaką można uzyskać w klocków w tablicy `A`.

Przykład Dla tablicy `A = [(1,4),(0,5),(1,5),(2,6),(2,4)]` wynikiem jest 3, natomiast dla tablicy `A = [(10,15),(8,14),(1,6),(3,10),(8,11),(6,15)]` wynikiem jest 2.

[2pkt.] Zadanie 3.

Szablon rozwiązania: `zad3.py`

Dana jest struktura realizująca listę jednokierunkową:

```
class Node:
  def __init__( self, val ):
    self.next = None
    self.val = val
```

Proszę napisać funkcję, która mając na wejściu ciąg tak zrealizowanych posortowanych list scala je w jedną posortowaną listę (składającą się z tych samych elementów).

Przykład Dla tablicy `[[0,1,2,4,5],[0,10,20],[5,15,25]]` - po przekształceniu jej elementów z Python'owskich list na listy jednokierunkowe - wynikiem powinna być lista jednokierunkowa, która po przekształceniu jej na listę Python'owską przyjmie postać `[0,0,1,2,4,5,5,10,15,20,25]`.