Algorytmy i Struktury Danych
Egzamin 1 (30.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 $O(n \log n)$). **Z wbudowanych funkcji sortowania nie wolno korzystać w zadaniu 1!**

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`
W tym zadaniu nie wolno korzystać z wbudowanych funkcji sortowania!

Mówimy, że tablica `T` ma współczynnik nieuporządkowania równy k (jest k-Chaotyczna), jeśli spełnione są łącznie dwa warunki:

1. tablicę można posortować niemalejąco przenosząc każdy element `A[i]` o co najwyżej k pozycji (po posortowaniu znajduje się on na pozycji różniącej się od i co najwyżej o k),
2. tablicy nie da się posortować niemalejąco przenosząc każdy element o mniej niż k pozycji.

Proszę zaproponować i zaimplementować algorytm, który otrzymuje na wejściu tablicę liczb rzeczywistych `T` i zwraca jej współczynnik nieuporządkowania. Algorytm powinien być jak najszybszy oraz używać jak najmniej pamięci. Proszę uzasadnić jego poprawność i oszacować złożoność obliczeniową. Algorytm należy zaimplementować jako funkcję:

```
def chaos_index( T ):
   ...
```

przyjmującą tablicę `T` i zwracającą liczbę całkowitą będącą wyznaczonym współczynnikiem nieuporządkowania.

Przykład. Dla tablicy:

```
T = [0, 2, 1.1, 2]
```

prawidłowym wynikiem jest $k = 1$.

[2pkt.] Zadanie 2.

Szablon rozwiązania: `zad2.py`

Robot porusza się po dwuwymiarowym labiryncie i ma dotrzeć z pocycji $A = (x_a, y_a)$ na pozycję $B = (x_b, y_b)$. Robot może wykonać następujące ruchy:

1. ruch do przodu na kolejne pole,
2. obrót o 90 stopni zgodnie z ruchem wskazówek zegara,
3. obrót o 90 stopni przeciwnie do ruchów wskazówek zegara.

Obrót zajmuje robotowi 45 sekund. W trakcie ruchu do przodu robot się rozpędza i pokonanie pierwszego pola zajmuje 60 sekund, pokonanie drugiego 40 sekund, a kolejnych po 30 sekund na pole. Wykonanie obrotu zatrzymuje robota i następujące po nim ruchy do przodu ponownie go rozpędzają. Proszę zaimplementować funkcję:

```
def robot( L, A, B):
  ...
```

która oblicza ile minimalnie sekund robot potrzebuje na dotarcie z punktu A do punktu B (lub zwraca `None` jeśli jest to niemożliwe).

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

Labirynt. Labirynt reprezentowany jest przez tablicę w wierszy, z których każdy jest napisem składającym się z k kolumn. Pusty znak oznacza pole po którym robot może się poruszać, a znak `'X'` oznacza ścianę labiryntu. Labirynt zawsze otoczony jest ścianami i nie da się opuścić planszy.

Pozycja robota. Początkowo robot znajduje się na pozycji $A = (x_a, y_a)$ i jest obrócony w prawo (tj. znajduje się w wierszu y_a i kolumnie x_a, skierowany w stronę rosnących numerów kolumn).

Przykład. Rozważmy labirynt składający się z 5 wierszy i 10 kolumn:

```
      # 0123456789
L = [ "XXXXXXXXXX", # 0
      "X X      X", # 1
      "X XXXXXX X", # 2
      "X        X", # 3
      "XXXXXXXXXX", # 4
```

Robot ma przejść z punktu $A = (1, 1)$ do punktu $B = (8, 3)$ (czyli z wiersza nr jeden i kolumny nr jeden do wiersza nr trzy i kolumny nr osiem). Rozwiązanie wymaga następujących kroków:

1. obrót zgodnie z ruchem wskazówek zegara (45s.);
2. ruch do przodu (60s.) i drugi ruch do przodu (40s.) (robot znajuje się na polu $(1, 3)$);
3. obrót przeciwny do ruchu wskazówek zegara (45s.);
4. ruch do przodu (60s.) i kolejny (40s.) (robot znajuje się na pozycji $(3, 3)$
5. pięć kolejnych ruchów do przodu (30s. każdy).

Cały przejazd trwa 440 sekund.

[2pkt.] Zadanie 3.

Szablon rozwiązania: `zad3.py`

Dany jest zbiór przedziałów $A = \{(a_0, b_0), \ldots, (a_{n-1}, b_{n-1})\}$. Proszę zaimplementować funkcję:

```
def kintersect( A, k ):
   ...
```

która wyznacza k przedziałów, których przecięcie jest jak najdłuższym przedziałem. Zbiór A jest reprezentowany jako lista par. Końce przedziałów to liczby całkowite. Można założyć, że $k \geq 1$ oraz k jest mniejsze lub równe łącznej liczbie przedziałów w A. Funkcja powinna zwracać listę numerów przedziałów, które należą do rozwiązania.

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

Przykład: Rozważmy listę przedziałóœ:

```
A = [(0,4),(1,10),(6,7), (2,8)]
```

Dla $k = 3$ wynikiem powinno być `[0,1,3]` (lub dowolna permutacja tej listy), co daje przedziały o przecięciu $[2, 4]$, o długości $4 - 2 = 2$.