Algorytmy i Struktury Danych
Zadanie offline 7 (16.V.2022)

Format rozwiązań

Rozwiązanie zadania musi się składać z **krótkiego** opisu algorytmu (wraz z uzasadnieniem poprawności) 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. korzystanie z zaawansowanych struktur danych (np. słowników czy zbiorów),
2. zmienianie nazwy funkcji implementującej algorytm, listy jej argumentów, lub nazwy pliku z rozwiązaniem,
3. modyfikowanie testów dostarczonych wraz z szablonem,
4. 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`, kolejka priorytetowa (`queue.PriorityQueue` lub `heapq`),
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.

Testowanie rozwiązań

Żeby przetestować rozwiązanie zadania należy wykonać polecenie: `python3 zad7.py`

Zadanie offline 7.

Szablon rozwiązania: `zad7.py`

Dane jest N miast, każde miasto jest otoczone murem, w którym znajdują się dwie bramy: północna i południowa. Jeżeli przyjedziemy do miasta jedną z bram musimy wyjechać z niego tą drugą. Wyjeżdżając każdą z bram można dojechać bezpośrednio do jednego lub więcej innych miast. Proszę zaproponować i zaimplementować algorytm który sprawdza czy wyruszając z jednego z miast można odwiedzić wszystkie miasta dokładnie jeden raz i powrócić do miasta, z którego się wyruszyło. Algorytm powinien być jak najszybszy i używać jak najmniej pamięci. Proszę skrótowo uzasadnić jego poprawność i oszacować złożoność obliczeniową.

Algorytm należy zaimplementować jako funkcję:

```
def droga(G):
   ...
```

która przyjmuje sieć połączeń `G` pomiędzy miastami i zwraca listę numerów odwiedzanych miast albo `None` jeśli takiej drogi nie ma. Sieć połączeń jest dana w postaci listy, która dla każdego miasta zawiera dwie listy: miast dostępnych z bramy północnej oraz miast dostępnych z bramy południowej. Miasta numerowane są od 0.

Przykład. Dla argumentów:

```
G = [ ([1],[2,3,4]),
      ([0],[2,5,6]),
      ([1,5,6],[0,3,4]),
      ([0,2,4],[5,7,8]),
      ([0,2,3],[6,7,9]),
      ([1,2,6],[3,7,8]),
      ([1,2,5],[4,7,9]),
      ([4,6,9],[3,5,8]),
      ([3,5,7],[9]),
      ([4,6,7],[8]) ]
```

wynikiem jest np. lista: ([0, 1, 5, 7, 9, 8, 3, 2, 6, 4])