algograf

Projekt 2: Patrol stolicy

W ramach projektu należy napisać program w języku Python rozwiązujący poniższe zadanie. Następnie program należy wysłać do oceny poprzez następujący kurs na platformie UPEL:

- nazwa: Algorytmy grafowe 2021/2022 - projekty
- hasło: galgo2122

Projekt zostanie oceniony w następujący sposób:

- osoby z najlepszymi programami (5 najszybszych) otrzymają +1.5 do oceny końcowej
- osoby z dobrymi programami otrzymują +1.0 do oceny końcowej
- osoby z programami niezbyt dobrymi (rozwiązującymi mało testów) otrzymują +0.5 do oceny końcowej
- osoby, które nie nadeślą programu lub ich program nie będzie działał otrzymują +0.0 do oceny końcowej

Termin nadsyłania rozwiązań: **14 lutego 2022, 23:59**

Treść zadania

Mimo militarnych sukcesów w zbrojnej kampanii przeciw Qubicji, kłopoty Króla Bajtycjana nie dobiegły końca - przyparty do muru przeciwnik zaczął tunelować bezpośrednio do stolicy niewielkie oddziały sabotażystów. By powstrzymać chaos, monarcha postanowił zamówić u Gildii Automoatyków eksperymentalne maszyny strażnicze, które będą patrolować newralgiczne punkty Bitogrodu. Niestety, mają one pewne ograniczenia - nie są w stanie zawrócić w miejscu (Bitogród jest gęsto zabudowany, a ulice wąskie) ani wykonywać złożonych rozkazów, więc trasa patrolu pojedynczej maszyny musi być cykliczna. Co więcej, w wyniku kosztownego do naprawienia błędu projektowego, strażnicy rozpoznają się nawzajem jako wrogie oddziały, należy więc uniknąć sytuacji, w której dwie maszyny mogłyby spotkać się w tym samym miejscu - żadne miejsce nie powinno leżeć na trasie patrolu więcej niż jednej maszyny.

Twoim zadaniem jest zaplanowanie tras patroli tak, by pokrywały one wszystkie ważne punkty miasta przy zachowaniu powyższych warunków. Ilość wykorzystanych maszyn jest dowolna, ale muszą być one w ruchu - Twój poprzednik, który zaproponował, by przy każdym ważnym miejscu postawić jednego nieruchomego strażnika, został wtrącony do lochu za nieroztropne zarządzanie zasobami królestwa.

Dane wejściowe

Do zaimplementowanej funkcji przekazane zostaną następujące argumenty:

- *N* - ilość miejsc które mają być patrolowane (miejsca numerowane od *1* do *N*)
- lista par opisujących ścieżki pomiędzy tymi miejscami, po których mogą przemieszczać się maszyny strażnicze

Przykładowe wywołanie może wyglądać następująco:

```
solve(6, [
    (1,2), (1,3), (2,3), (2,4), (3,5), (4,5), (4,6), (5,6)
])
```

Zaimplementowana funkcja powinna zwracać listę tras patrolu. Każda trasa patrolu to lista numerów miejsc, które kolejno odwiedzać będzie używająca tej trasy maszyna strażnicza. Przykładowe zwracane wartości dla powyższego wywołania:

```
[[1,2,3], [4,5,6]]     # OK
[[3,2,1], [5,4,6]]     # OK
[[1,2,4,6,5,3]]        # OK
[[1,2,3,4,5,6]]        # NIE OK - nie da się przejść z 3 do 4
[[1,2,3,5,4,6]]        # NIE OK - nie da się przejść z 6 z powrotem do 1
[[1,2], [3,5], [4,6]]  # NIE OK - nie można zawracać w miejscu
[[1,2,3], [4,6,5,3,2]] # NIE OK - 2 i 3 są patrolowane więcej niż raz
[[1], [2,3,5,6,4]]     # NIE OK - marnotrawstwo zasobów => loch
```

Jeśli rozwiązanie nie istnieje, funkcja powinna zwracać listę pustą.

Instrukcja

Infrastruktura do projektu dostępna jest w formie archiwum z plikami źródłowymi w języku Python (link na dole). Szkielet rozwiązania znajduje się w pliku *example.py* - importuje on funkcję `runtests` z modułu `data` i uruchamia ją, podając swoją funkcję rozwiązującą jako argument. Przesyłane rozwiązania powinny mieć postać analogicznego pliku. Przetestować rozwiązanie można uruchamiając ów plik, np.

```
python3 example.py
```

Na wyjście standardowe wypisane zostaną informacje o rezultatach poszczególnych testów, a także podsumowanie z ilością testów zakończonych sukcesem i przybliżonym łącznym czasie obliczeń.

Warunki techniczne

- Program powinien być napisany w języku Python i działać z wersją 3.8.10
- Program nie może wykorzystywać zewnętrznych bibliotek (biblioteka standardowa jest dopuszczalna)

Pliki

- project2.zip