Języki Skryptowe

dokumentacja projektu „Hotele”

Grzegorz Koperwas

8 grudnia 2021

Część I

Opis programu

W Bajtocji jest n miast połączonych zaledwie n − 1 drogami. Każda z dróg łączy bezpośrednio dwa miasta. Wszystkie drogi mają taką samą długość i są dwukierunkowe. Wiadomo, że z każdego miasta da się dojechać do każdego innego dokładnie jedną trasą, złożoną z jednej lub większej liczby dróg. Inaczej mówiąc, sieć dróg tworzy drzewo.

Król Bajtocji, Bajtazar, chce wybudować trzy luksusowe hotele, które będą gościć turystów z całego świata.

Król chciałby, aby hotele znajdowały się w różnych miastach i były położone w tych samych odległościach od siebie.

Pomóż królowi i napisz program, który obliczy, na ile sposobów można wybudować takie trzy hotele w Baj- tocji.

Instrukcja obsługi

Należy wykonać plik `./run.sh`, wygeneruje on automatycznie zestawy danych testowych, wykona plik `./projekt.py` dla nich, oraz wygeneruje skryptem `./raport.py` plik `./raport.html`.

Dodatkowe informacje

Wymagania:

1. Biblioteka Jinja2
2. Python3 (sprawdzanie działanie na wersji 3.9.7, wcześniejsze mogą nie działać)

Część II

Opis działania

Dane jest drzewo n węzłów.

W pierwszej fazie, dla każdego węzła jest twożony jest słownik zwracający dystans do dannego węzła (miasta), dla dannego węzła[1].

W drugiej fazie, dla każdego miasta, twożony jest słownik zwracający listę wszystkich miast, których odległość jest równa, dla danej odległości.

Następnie, dla każdej odległości[2], jest zliczana liczba miast spełniających warunek zadania[3].

Algorytmy

Generacja słownika Miasto → odległość

Zrealizowany jest głównie w metodzie `get_connections` w klasie `Miasto`.

Każde „Miasto" jest węzłem w drzewie, zawiera ono domyślnie pusty słownik `city2distace`[4], flagę trybu szybkiego, puste polę na bramę trybu szybkiego oraz listę połączeń z innymi miastami.

Tryb szybki jest uruchamiany jeżeli napotkamy na sytuację, gdzie występuje *dokładnie jeden* węzeł łączący dane miasto (brama) z resztą drzewa. Wtedy zamiast „chodzić" po drzewie, możemy przepisać słownik z *bramy trybu szybkiego*.

Algorytm wyznaczania ilości hoteli.

Każde miasto posiada unikalne id. Zatem w celu wyeliminowania powtórzeń tych samych konfiguracji hoteli, każde miasto wyznacza możliwe konfiguracje hoteli, gdzie jego „partnerzy" posiadają większe id.

Implementacja

Opis, zasada i działanie programu ze względu na podział na pliki, nastepnie funkcje programu wraz ze szczegółowym opisem działania (np.: formie pseudokodu, czy odniesienia do równania)

Testy

Tutaj powinna pojawić się analiza uzyskanych wyników oraz wykresy/pomiary.

[1] W dalszej części pracy będe stosował te wyrażenia wymiennie

[2] kluczy słownika

[3] W wynniku optymalizacji powtóżenia są eliminowane

[4] Zwany równierz słownikiem Miasto → odległość

Eksperymenty

Sekcję używamy gdy porównywaliśmy dwa lub więcej algorytmów, albo wykonywaliśmy jakies pomiery.

Warto dodać jakies wykresy jako obraz, albo tabele z wynikami.

Wszyskie wyniki powinny być opisane/poddane komentarzowi i poddane analizie statystycznej.

Pełen kod aplikacji

```
funkcja GetConnections(miasto, force_slow):
    if miasto jest w trybie szybkim i nie ma ustawionej flagi force_slow then
        /* W trybie szybkim dopisujemy nowe miasta z ,,bramy''          */
        for miasto w city2distace w bramie do
            if miasto nie jest w city2distace then
                city2distace[ miasto ] = dystans do bramy + dystans z bramy do
                 miasta
            end
        end
    end
    else if dict city2distance jest pusty then
        for połączenie w miasto do
            city2distace[ miasto z połączenia ] = 1
        end
        /* Sprawdź czy możemy wejść w tryb szybki                       */
        if miasto ma jednego sąsiada then
            wejdź w tryb szybki i ustaw bramę na znalezione miasto
        end
    end
    else
        tmp = dict();
        for miasto w city2distace do
            if miasto jest najbardziej oddalonym miastem then
                for dla sąsiadów miasta do
                    if sąsiad nie jest w city2distace then
                        tmp[ sąsiad ] = dystans do miasta + 1
                    end
                end
            end
        end
        if w tmp jest tylko jedno miasto then
            wejdź w tryb szybki
        end
        dopisz elementy z tmp do city2distance
    end
    return czy jestem w trybie szybkim
```

Algorithm 1: Metoda pomocnicza do obliczania dystansu

```
funkcja main(miasta):
    skończoneMiasta = [] while len(skończoneMiasta) != len(miasta) do
        for miasto w miasta do
            if miasto w skończoneMiasta then
                continue
            end
            else if miasto ma odległość do wszystkich innych miast then
                dodaj miasto do skończoneMiasta
            end
            else
                GetConnections(miasto) if miasto jest w trybie szybkim then
                    dodaj miasto do skończoneMiasta
                end
            end
        end
    end
    fin = [] /* Dokończ miasta w trybie szybkim                    */
    while len(fin) != len(skończoneMiasta) do
        for miasto w skończoneMiasta do
            if miasto w fin then
                continue
            end
            else if Miasto ma odległość do wszystkich innych miast then
                dodaj miasto do fin
            end
            else
                GetConnections(miasto)
            end
        end
        if nie było zmian w fin then
            /* wymuś tryb powolny w miastach                       */
            for miasto w skończoneMiasta do
                if miasto w fin then
                    continue
                end
                else
                    GetConnections(miasto, true)
                end
            end
        end
    end
    return fin
```

Algorithm 2: Obliczanie dystansu dla wszystkich miast

```
Input: miasto
Output: res
distance2city = {} /* odwrócenie słownika city2distace                    */
for miasto w city2distace do
    if id miasta > id miasta rozważanego then
        dodaj miasto do distance2city
    end
end
for miasta w distance2city do
    if conajmniej 2 miasta dla danego dystansu then
        for miasto poza pierwszym dla danego dystansu do
            for miasta po rozważanym miastem do
                if Dystans między miastami jest równy then
                    dodaj hotele do res
                end
            end
        end
    end
end
```

Algorithm 3: Wyznaczanie ilości hoteli w jednym mieście