Sprawozdanie

**Zespół:**

- Maciej Dmowski

- Jakub Strawa 300266

Projekt został napisany w języku Python 3.8

**Temat: Ekstremalnie tania linia lotnicza**

Jak wiadomo gdy samolot pasażerski nie znajduje się w powietrzu, to przynosi straty. Z problemem strat zaś poradzić sobie muszą pracownicy ekstremalnie taniej linii lotniczej, znajdującej się na skraju bankructwa i dysponującej jednym samolotem. Dział sprzedaży i marketingu zbiera zamówienia od biur podróży, zaś dział optymalizacji stara się wskazać te, które należy przyjąć tak - aby czas postoju samolotu był jak najmniejszy.

Dla uproszczenia przyjmujemy, iż zamówienia określać będą jedynie moment wylotu i powrotu samolotu na lotnisko (przyjmujemy tylko zamówienia, w których zamawiający zapewnia także „ładunek” na natychmiastowy lot powrotny), zaś czas obsługi naziemnej samolotu jest pomijalny. A zatem - dana jest lista par (w,p) reprezentujących zamówienia zebrane przez dział sprzedaży, gdzie w - czas wylotu samolotu, p - czas przylotu samolotu; w,p < 30000, długość listy jest nie większa niż 10000. Zaproponuj algorytm, który wskaże które zamówienia należy przyjąć, tak aby zmaksymalizować czas pobytu samolotu w powietrzu.

**Opis problemu**

Posiadamy maksymalnie 10000 lotów opisanych czasami wylotu i przylotu. Celem naszego algorytmu jest uzyskanie jak największej zajętości czasowej - czasu pobytu samolotu w powietrzu dla przedziału 0 – 30000 liczonego w minutach. Uzyskamy dzięki temu plan lotów na prawie 21 dni zakładając, że 1 jednostka czasowa to 1 minuta.

**Założenia**

- 1 jednostka czasu trwania lotu dla uproszczenia nazywana jest minutą.

- Czasy wylotu oraz przylotu generowane są z dokładnością do minuty (przedział 0-30000), a planowanie wykonujemy dla 30000 minut, ponieważ najpóźniejszy możliwy powrót jest po 30000 minutach od początku okresu planowania.

- Każda minuta poza lot lotniskiem przynosi taki sam zysk bez względu czy lot trwa 10 czy 10000 minut.

**Struktury danych wejściowych**

Danymi wejściowymi jest lista par liczb naturalnych w pliku .txt, gdzie druga liczba jest większa od pierwszej o co najmniej 1, a przedział utworzony przez te liczby należy do przedziału [0,30000].

**Przykładowe dane wejściowe (np. plik.txt):**

432,533

64,112

34,234

543,2240

234,653

3,68

0,94

2345,21532

34,67

443,765

2354,5843

723,2341

45,7643

**Generacja danych:**

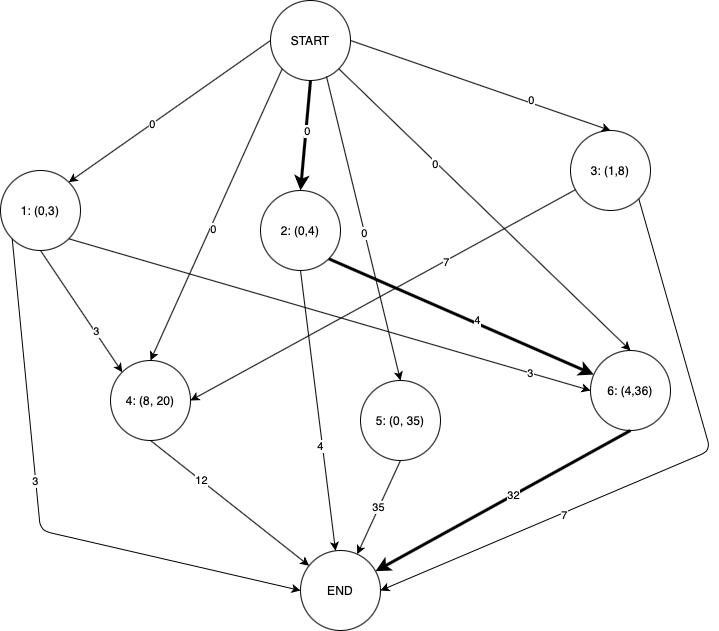
Generujemy maksymalnie 10 000 lotów sprawdzając tylko czy godzina wylotu jest wcześniejsza od godziny przylotu.

W zależności od potrzeb zmieniane będą parametry generacji, np. generacja lotów o długości 30000 (maksymalnej) lub 1 (minimalnej), w celu sprawdzenia przypadków skrajnych.

**Opis rozwiązania**

**Opis algorytmu:**

Na początku sortujemy dane według rosnących godzin wylotów. Następnie, generujemy graf skierowany ważony. Wierzchołek startowy: 0, połączony jest ze wszystkimi lotami. Każdy lot połączony jest z każdym możliwym następnym lotem - przylot lotu poprzedniego <= wylot lotu następnego. Krawędzie wychodzące z wierzchołka mają wagi odpowiadające długości lotu tego wierzchołka. Ostatni wierzchołek jest punktem końcowym porządkowania - minuta: 30.000, do którego wchodzą krawędzie z wierzchołków każdego z lotów. Następnie wyszukujemy ścieżkę o największym koszcie w grafie, dzięki czemu otrzymujemy największy sumaryczny czas w powietrzu. Przykładowy graf wraz ze ścieżką możemy zobaczyć na rysunku 1.



Rys.1 przedstawienie generowanego grafu

**Opis plików źródłowych:**

- *main.py* - główny plik programu, analizuje podane flagi i uruchamia algorytm z odpowiednimi parametrami podanymi przez użytkownika

- *generator.py* - plik odpowiada za generacje danych oraz wczytywanie ich z pliku tekstowego (w przypadku trybu 1)

- *graph.py* - plik w którym realizowana jest struktura grafu

- *vertex.py* - plik w którym realizowana jest struktura wierzchołka grafu

- *edge.py* - plik w którym realizowana jest struktura krawędzi grafu

**Opis struktur oraz funkcji:**

*Klasa Vertex* — reprezentuje pojedynczy wierzchołek grafu — lot o określonym czasie rozpoczęcia i zakończenia.

Zawiera zbiory wierzchołków:

- poprzedników — wierzchołki, od których krawędzie prowadzą do wierzchołka

- następników — wierzchołki, do których prowadzą krawędzie zaczynające się w tym wierzchołku

W celu wyliczenia najdłuższej ścieżki w grafie dodatkowo zawiera:

- wartość logiczną czy wierzchołek został odwiedzony — wykorzystywane w sortowaniu topologicznym

- koszt najdłuższej ścieżki kończącej się w tym wierzchołku

- poprzedni wierzchołek z najdłuższej ścieżki

*Klasa Graph* - przechowuje wszystkie wierzchołki stanowiące graf, tworzy je na podstawie otrzymanej listy lotów oraz tworzy między nimi krawędzie.

*Metoda topological\_sort* — dokonuje sortowania topologicznego wierzchołków w grafie metodą DFS.

*Metoda max\_path* — zwraca najdłuższą ścieżkę w grafie (największa suma długości krawędzi).

Przechodząc po tablicy wierzchołków posortowanej topologicznie, zapisywana jest najdłuższa ścieżka kończąca się w danym wierzchołku. Najdłuższa osiągnięta ścieżka jest zwracana jako wynik działania funkcji w postaci tablicy zawierającej łączny czas w powietrzu oraz loty, które należy wybrać.

**Ocena złożoności algorytmu**

Ze wstępnych obliczeń wynikało, że generowanie grafu ma złożoność czasową n^2, a przeszukiwanie: liniową lub V+E, więc łącznie złożoność czasowa i tak wyniesie n^2.

Z przeprowadzonych analiz wynika, że teoretyczna, obliczona przez nas złożoność praktycznie nie odbiera od tej faktycznej, zmierzonej.

W celu analizy złożoności algorytmu przeprowadziliśmy kilka uruchomień programu w trybie 3, gdzie generowaliśmy 20 instancji problemu w przedziale 1000,8000 z wartością skoku - 500. Dodatkowo wykorzystaliśmy współczynnik zgodności oceny teoretycznej z pomiarem q(n), podstawiając za medianę n wartość środkową średnich z iteracji danego problemu lub przyjmując wartość n/2 + 1, ponieważ obliczenie średniej arytmetycznej z dwóch sąsiednich wartości w znaczący sposób zaburzyłoby miarodajność współczynnika q(n). Poniżej prezentujemy wykresy uzyskanych danych.

**Przeprowadzone testy:**

W celu przeprowadzenia rzetelnej analizy, testowaliśmy dane zupełnie losowe, ale także skrajne przypadki: loty o długości 1 i 30000. Podsumowanie łącznego czasu działania programu, czyli: generowania danych, tworzenia grafu, sortowania topologicznego oraz znalezienia ścieżki, znajduje się na rysunku 2. Program został uruchomiony trzykrotnie z takimi samymi parametrami - 16 problemów do rozwiązania po 20 iteracji (10 dla danych minimalnych i maksymalnych), zaczynając od 500, krok równy 500, ale za każdym razem generował loty:

-losowej długości – kolor pomarańczowy

-minimalne o długości 1 – kolor niebieski

-maksymalne o długości 30000 – kolor szary

Rys.2 Wykres przedstawia złożoności w przypadkach: najlepszym, najgorszym i średnim.

Na rysunku 3 możemy zobaczyć sposób zachowania się współczynnika q(n) dla 3 uruchomień programu z rysunku 2. Dane te potwierdzają, że nasze przypuszczenia o złożoności n^2 są w gruncie rzeczy prawidłowe. Szczegółowe informacje o czasach działania poszczególnych operacji znajdują się w plikach:

- table\_testing\_random.txt – tabela z wynikiem rozwiązywania losowych lotów

- table\_testing\_30000.txt – tabela z wynikiem rozwiązywania maksymalnych lotów

- table\_testing\_1.txt – tabela z wynikiem rozwiązywania minimalnych lotów

Fragment tabeli table\_testing\_random.txt zamieszczamy poniżej:

Obraz zawierający tekst, komputer

Opis wygenerowany automatycznie

Zdj.1 Fragment tabeli dla losowych długości lotów

Rys.3 Wykres przedstawia rozrzut wartości współczynnika q(n).

Kolejnym testem było uruchomienie 40 iteracji problemu n=1000. Na rysunku 4 możemy zauważyć, że rozrzut wartości q(n) w znaczącej większości przypadków wynosi około 5%. Szczegółowe dane o tym uruchomieniu znajdują się w pliku: table\_testing\_40\_1000.txt.

Na rysunkach 5-7 możemy zobaczyć jak wygląda udział poszczególnych operacji w całkowitym czasie rozwiązywania problemów. Dodawanie krawędzi zajmuje najwięcej czasu i rośnie kwadratowo. W ten sam sposób rośnie czas szukania ścieżki, który jest drugim najdłuższym zadaniem, a czas sortowania topologicznego – trzecie najdłuższe zadanie, w sposób |V| + |E|. Pozostałe operacje rosną liniowo, dlatego czas ich działania dla n > 2000 staje się nieistotny, a nawet pomijalny.

Rys.4 Wykres przedstawia rozrzut wartości współczynnika q(n) dla 40 iteracji n=1000.

Rys.5 Wykres przedstawia udział czasowy wszystkich operacji w czasie trwania programu.

Rys.6 Wykres przedstawia udział czasowy operacji w czasie trwania programu.

Rys.7 Wykres przedstawia udział czasowy operacji w czasie trwania programu.

**Dane wynikowe**

Dla trybu 1(-m 1) do pliku zapisywane są następujące dane:

- łączny czas samolotu w powietrzu

- łączny czas wszystkich obliczeń dla danego problemu

- wyodrębnione czasy poszczególnych operacji: generacja danych, dodawanie wierzchołków, dodawanie krawędzi, sortowanie, szukanie ścieżki

- kolejność oraz czasy wylotów i przylotów wybranych lotów

Dla trybu 2(-m 2) do pliku zapisywane są następujące dane:

- łączny czas samolotu w powietrzu

- łączny czas wszystkich obliczeń dla danego problemu

- wyodrębnione czasy poszczególnych operacji: generacja danych, dodawanie wierzchołków, dodawanie krawędzi, sortowanie, szukanie ścieżki

- kolejność oraz czasy wylotów i przylotów wybranych lotów

Dodatkowo do pliku generated\_data.txt zapisywane są wygenerowane loty.

Dla trybu 3(-m 3) do pliku zapisywane są następujące dane:

- tabela do pliku table.txt posiadająca wszystkie informacje o iteracjach i instancjach problemów(wszystkie informacje z trybu 1)

- porównanie z założoną złożonością obliczeniową

**Sposoby uruchomienia programu**

0. Do poprawnego działania program wymaga ściągnięcia biblioteki Beautiful Table:

*pip install beautifultable*

1. Pobranie danych wejściowych z pliku i wypisanie rozwiązania do pliku:

*python main.py –m 1 -in data.txt -out result.txt*

2. Wygenerowanie instancji problemu o rozmiarze n i wypisanie rozwiązanie do pliku:

*python main.py -m 2 -n 1000 -out result.txt*

3. Przeprowadzenie całego procesu testowania z pomiarem czasu dla rosnącego o s(tep) n i porównaniem ze złożonością teoretyczną:

*python main.py -m 3 -n 1000 -s 500 -k 10 -r 5*

Pomiar czasu dla k problemów o wielkościach n, n+step, n+2\*step itd. Dla każdej wielkości losowanych jest r instancji problemu.

4. Uzyskanie prostej pomocy co do flag:

*python main.py -h*

Uwaga - Kolejność wpisania flag jest istotna.