Sprawozdanie

Zespół:
- Maciej Dmowski 300202
- 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].

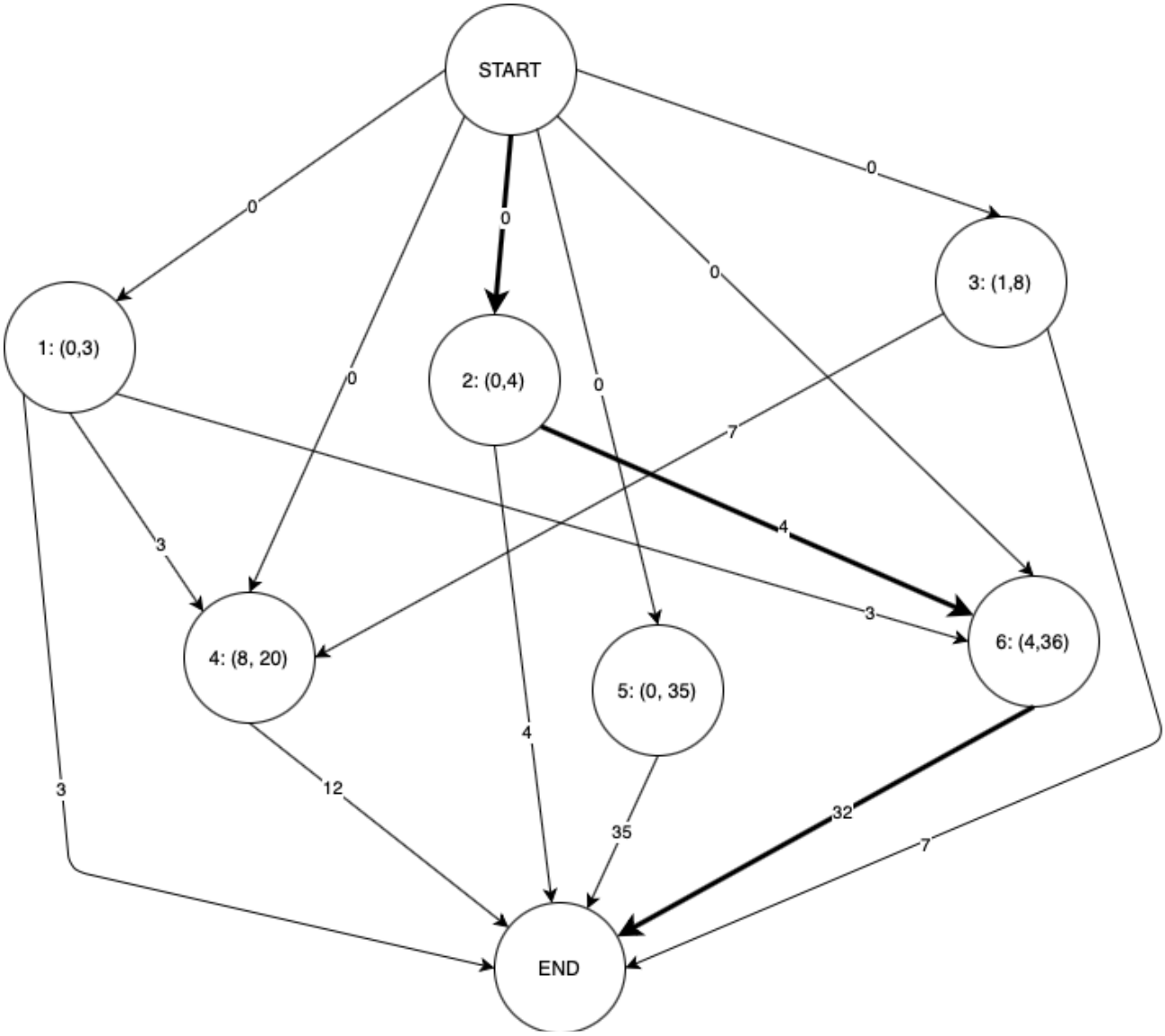
Przykładowe dane wejściowe (np. plik.txt):
432,533
64,112
34,234
543,2240

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).

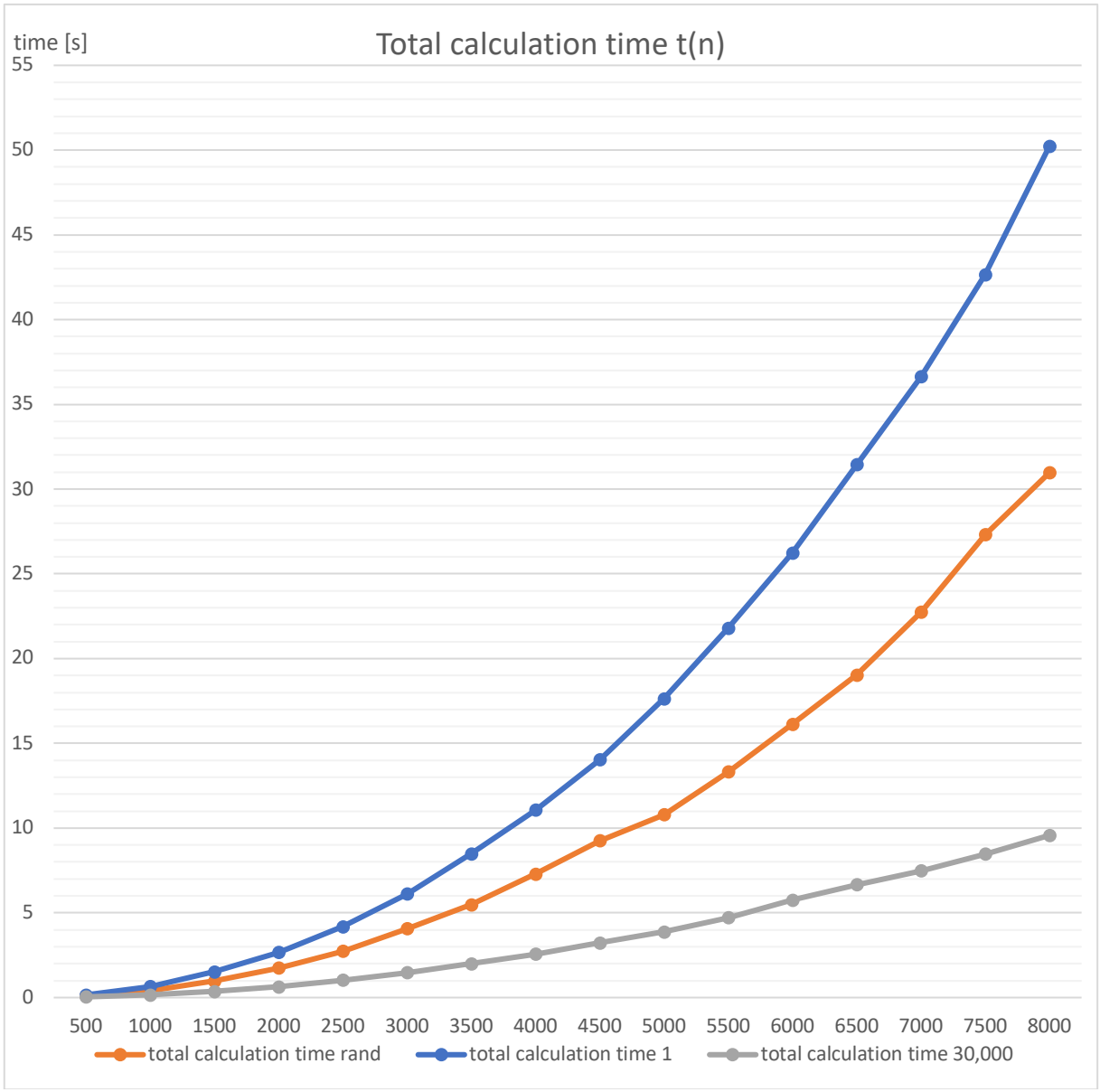
Przechodząc po tablicy wierzchołków posortowanej topologicznie, zapisywana jest najdłuższa ścieżka kończąca się w danym wierzchołku. Najdłuższa osiągnięta ścieżka jest zwracana jako wynik działania funkcji w postaci tablicy zawierającej łączny czas w powietrzu oraz loty, które należy wybrać.

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



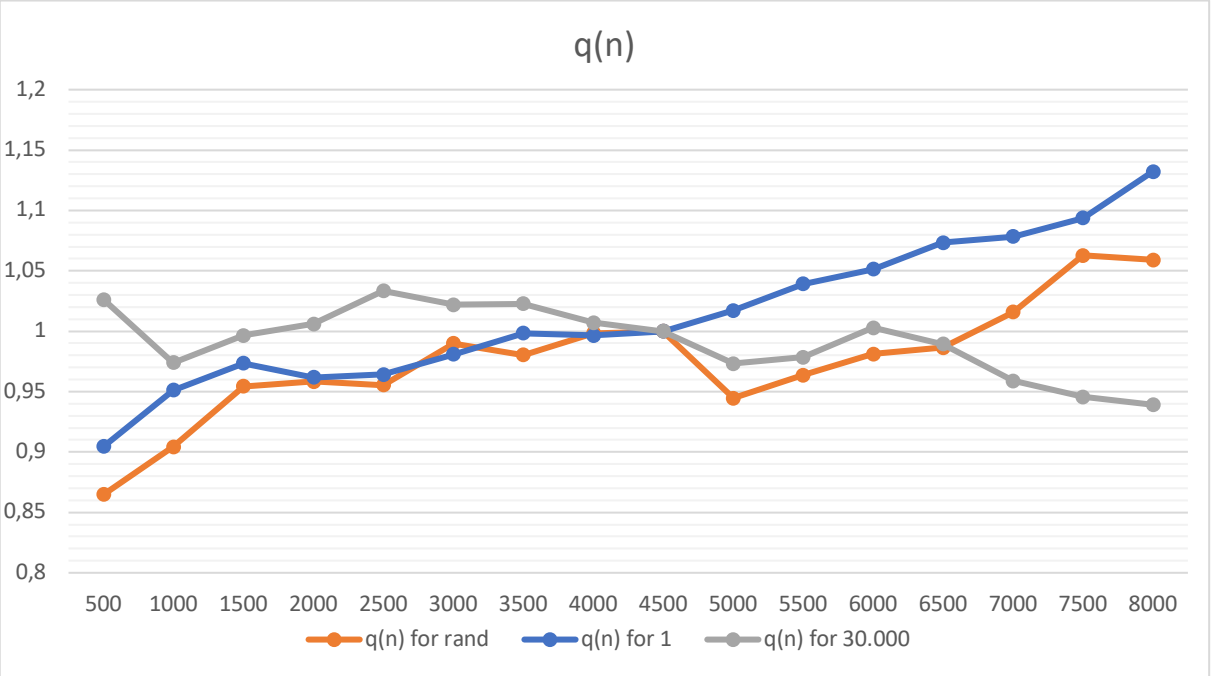
Rys.2 Wykres przedstawia złożoności w przypadkach: najlepszym, najgorszym i średnim.

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:

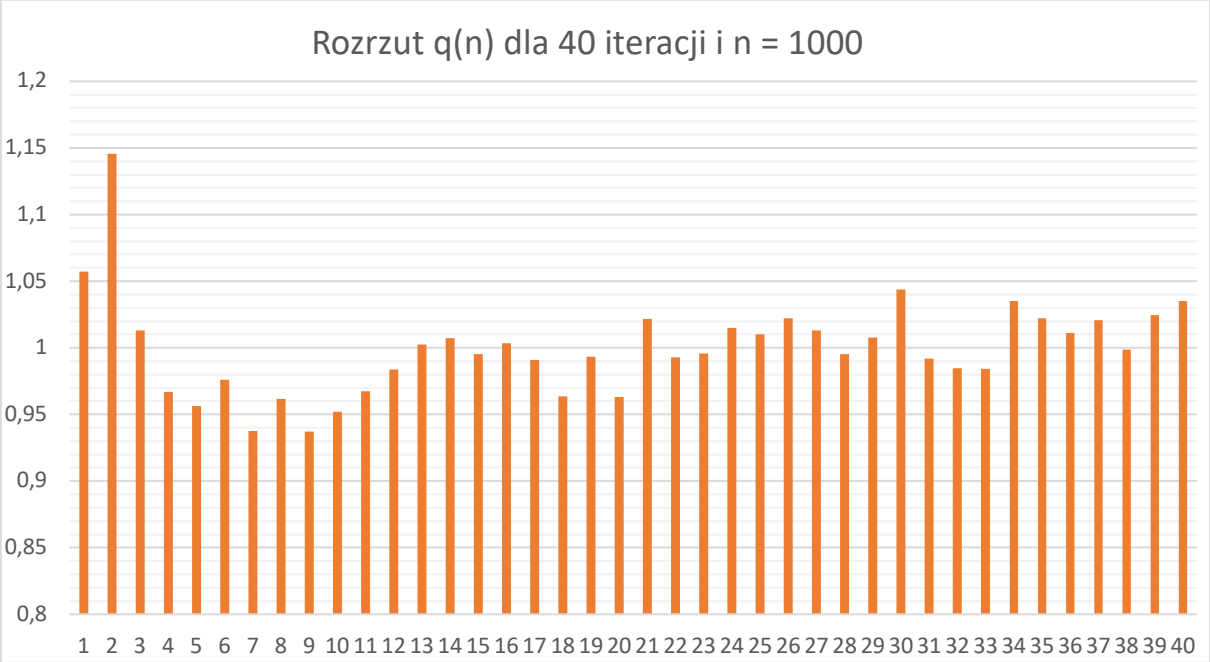
Problem size	Instance	Tot. airtime	Tot. calculation time	Data generation time	Add vertices time	Add edges time	Topological sort time	Path finding time	q(n)
500	1	29830	0.097533873	0.000936343	0.001337526	0.079759635	0.004839231	0.010582839	0.8538269781
500	2	29828	0.099871429	0.00092249	0.000728794	0.080124358	0.004593158	0.01341352	0.8742903137
500	3	29652	0.106578999	0.0009142	0.000707172	0.086132063	0.005380607	0.013352216	0.9330094443
500	4	29639	0.098384545	0.000898638	0.000600503	0.076782351	0.006078563	0.013923086	0.8612738956
500	5	29602	0.093606686	0.000896572	0.000559529	0.076578027	0.004923619	0.010569593	0.8194477609
500	6	29848	0.098377624	0.000971581	0.000687461	0.078219347	0.005074976	0.013324368	0.8612133081
500	7	29817	0.101743282	0.001266336	0.000791063	0.081137979	0.005160899	0.013271757	0.8906768115
500	8	29907	0.096456237	0.001130191	0.000809868	0.078064474	0.005175616	0.011141583	0.8443931818
500	9	29868	0.10142982	0.000878831	0.000577651	0.082713461	0.004791607	0.012394086	0.8879327154
500	10	29734	0.097543129	0.000958542	0.000672917	0.077403377	0.005947778	0.012461362	0.8539080066
500	11	29663	0.096579894	0.000965437	0.000705078	0.077935115	0.004865725	0.012028676	0.8454756948
500	12	29828	0.101259846	0.00088619	0.000541193	0.08098809	0.005329252	0.013436839	0.8864447361
500	13	29751	0.094553659	0.000890831	0.000544037	0.075165298	0.006143543	0.011717733	0.8277377126
500	14	29736	0.103453108	0.000926065	0.000707423	0.08536351	0.004995405	0.011374463	0.9056448992
500	15	29777	0.106530583	0.000880546	0.000891228	0.085609188	0.005416431	0.013612105	0.9325856029
500	16	29683	0.096285042	0.000979822	0.000695753	0.07727605	0.004762949	0.01249109	0.8428945137
500	17	29949	0.092835113	0.001001342	0.000538794	0.074499591	0.005732995	0.010967363	0.8126932886
500	18	29696	0.101518996	0.001083675	0.000756298	0.081419308	0.005257563	0.012867305	0.8887133763
500	19	29696	0.095052278	0.000884335	0.000536992	0.078160966	0.004762198	0.010631116	0.8321027023
500	20	29815	0.096070832	0.00087698	0.00073317	0.078531113	0.004853735	0.010997706	0.8410192854
500	average	29765.95	0.0987832487	0.0009574473	0.0007061225	0.079593165	0.0052042925	0.0122279403	0.8647642114
1000	1	29846	0.399038426	0.001934208	0.001243182	0.325900934	0.022155781	0.047707896	0.8733114018
1000	2	29885	0.415797023	0.001768647	0.001122905	0.34167097	0.022933636	0.048187198	0.9099882552

Zdj.1 Fragment tabeli dla losowych długości lotów

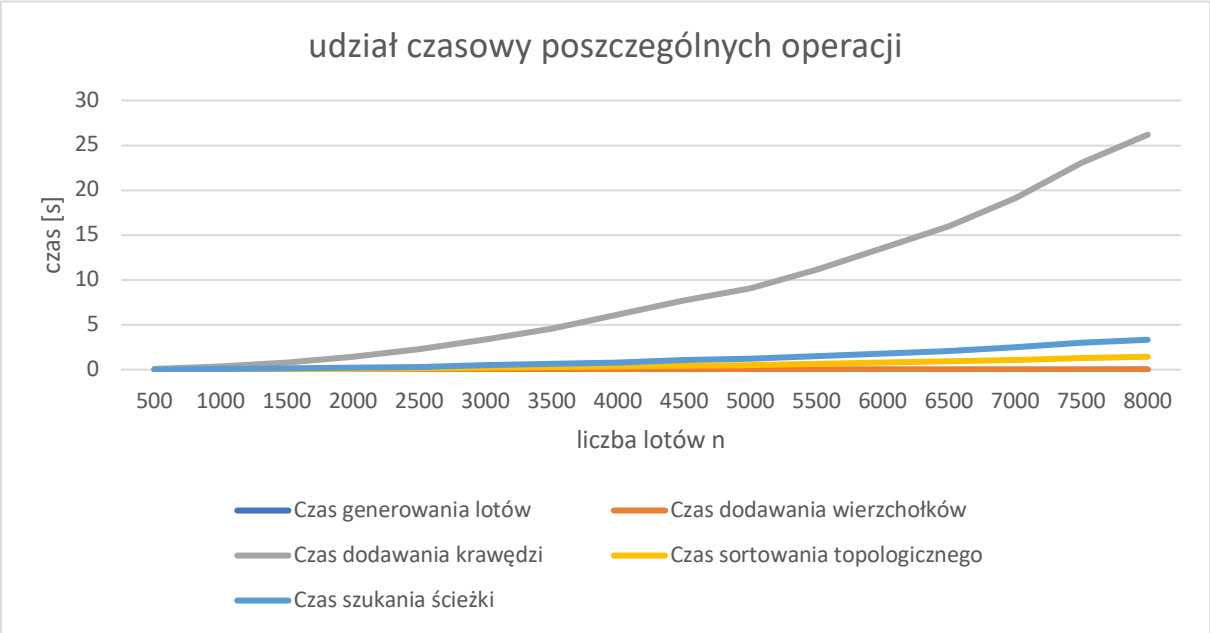


Rys.3 Wykres przedstawia rozrzut wartości współczynnika q(n).

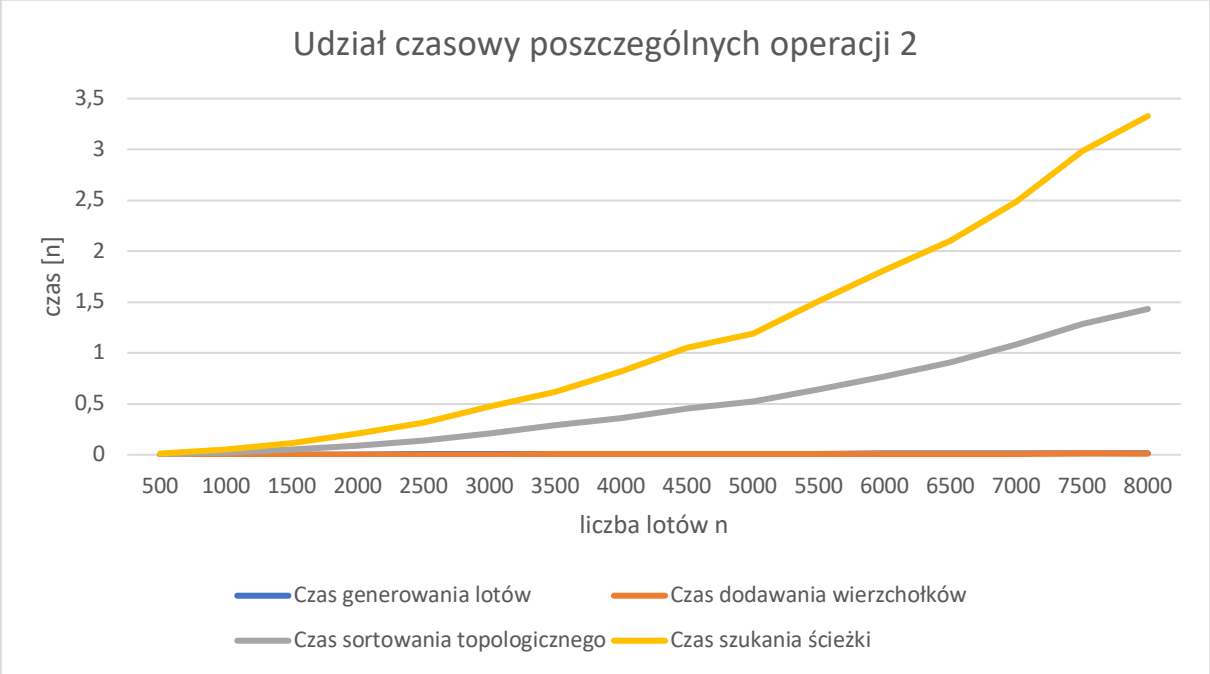
Kolejnym testem było uruchomienie 40 iteracji problemu n=1000. Na rysunku 4 możemy zauważyć, że rozrzut wartości q(n) w znaczącej większości przypadków wynosi około 5%. Szczegółowe dane o tym uruchomieniu znajdują się w pliku: table_testing_40_1000.txt.
Na rysunkach 5-7 możemy zobaczyć jak wygląda udział 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.



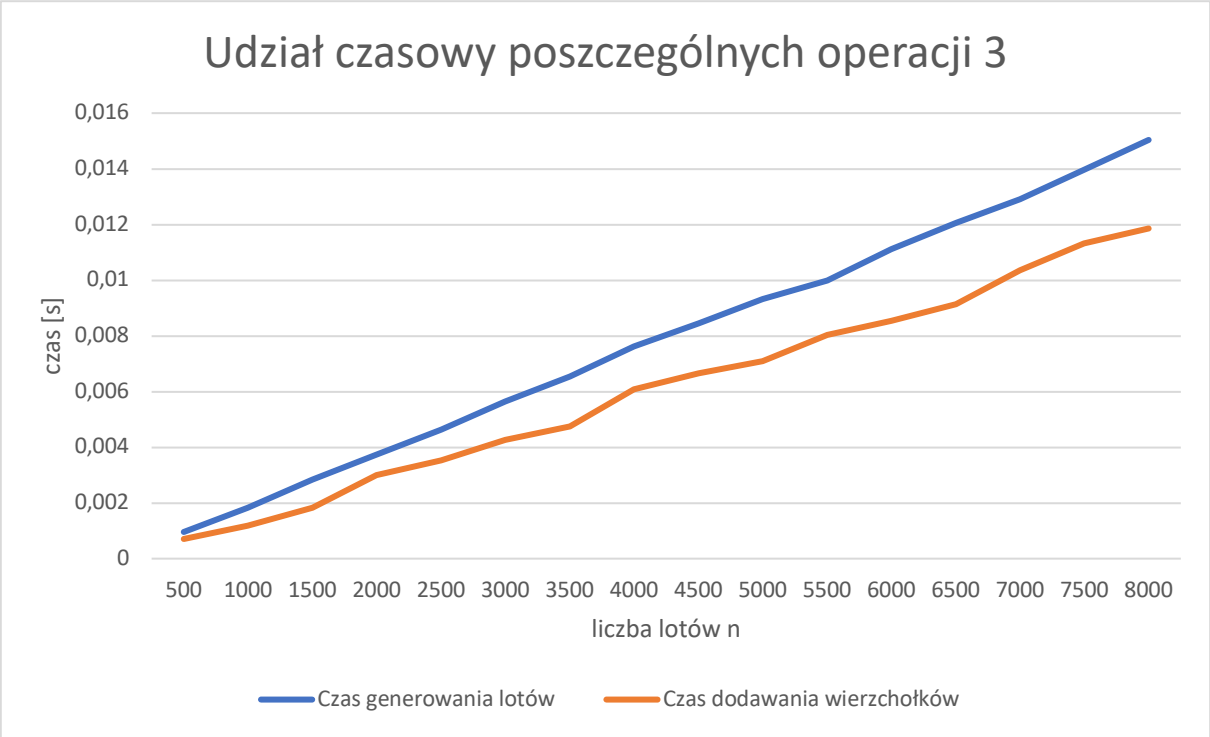
Rys.4 Wykres przedstawia rozrzut wartości współczynnika q(n) dla 40 iteracji n=1000.



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.

Wnioski

Obliczona przez nas złożoność obliczeniowa jest zgodna z zaobserwowaną po licznych testach. Zauważyliśmy, że największym problemem w naszym projekcie jest generowanie grafu. Uważamy, że znaczącym usprawnieniem byłoby pojedyncze wygenerowanie grafu i ewentualne późniejsze wprowadzanie poprawek i zmian wraz z upływem czasu. W diametralny sposób zmniejszyłoby to czas działania programu w kolejnych uruchomieniach, ponieważ zaobserwowaliśmy, że dla n>2000 znaczną większość czasu działania programu

zajmuje utworzenie grafu. Pozwoliłoby to na operowaniu w sensownym czasie na grafach o wielkości rzędu kilkuset tysięcy wierzchołków, ponieważ utworzenie grafu dla 10 000 lotów zajmuje około 45 sekund, zatem dla 100 000 lotów wygenerowanie grafu zajęłoby około 75 minut, a następnie wyszukanie najdłuższej ścieżki około 8 minut, co jest znaczącym usprawnieniem.

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.