Wojciech Sitek

Projekt Analiza Algorytmów

Dokumentacja

Tytuł i treść problemu

Przepustowość

> *Dany jest graf opisujący pewną sieć wodociągową, w którym wierzchołki są oznaczone jako zawory, źródła lub punkty odbioru. Każdy zawór może być ustawiony na przepustowość w zakresie [0,1], każde źródło ma określoną wydajność, a odcinek stałą przepustowość w zakresie [0,1].*
>
> *Opracować algorytm, który dla danych nastaw zaworów określi ilość wody docierającej do wszystkich punktów odbioru.*

Interpretacja problemu

W opisie problemu występują trzy rodzaje obiektów:

- Źródło – obiekt, który dostarcza wodę, o parametrze *wydajność*
- Zawór – obiekt, który znajduje się na końcach odcinków, o parametrze *przepustowość* w zakresie [0,1]
- Odcinek – obiekt o parametrze *przepustowość* w zakresie [0,1].
- Punkt odbioru – obiekt o parametrze *przepływ*, który jest do znalezienia przez program.

Wszystkie te obiekty są częścią grafu przepływowego skierowanego od wierzchołka s do wierzchołka t. Od wierzchołka s krawędzie skierowane są do źródeł i mają przepustowości równe wydajności źródeł, do których prowadzą. Następnie odcinki odwzorowywane są w krawędzie o przepustowości równej przepustowości odcinków. Zawory odwzorowywane są w wierzchołki, które także posiadają przepustowość równą przepustowości zaworów. Punkty odbioru są odwzorowywane w wierzchołki o nieskończonej przepustowości. Następnie wszystkie punkty odbioru są łączone krawędziami z wierzchołkiem t. Przepływy krawędzi to szukane wartości programu.

Algorytmy

Do rozwiązania problemu zostanie użyty algorytm największego przepływu z algorytmem Edmondsa-Karpa. Znajduje on ścieżkę powiększającą w grafie, a następnie aktualizuje przepływ w grafie wzdłuż tej ścieżki. Algorytm kończy działanie, gdy nie istnieje już żadna ścieżka powiększająca w grafie.

Zostanie on zmodyfikowany faktem, iż w niniejszym projekcie wierzchołki także posiadają przepustowość, co zostanie uwzględnione w implementacji algorytmu.

Ścieżka powiększająca będzie poszukiwana algorytmem BFS (przeszukiwania wszerz).

```
załaduj_dane_wejściowe_do _struktur()
while(istnieje_ścieżka_powiększająca) {
        znajdź_ścieżkę_powiększającą()
        aktualizuj_przepływ_i_przepustowość_wzdłuż_ścieżki()
}
```

wypisz_przepływ_punktów_odbioru()

Wartość zwracana

W przypadku gdyby nie było możliwe przepłynięcie wody do żadnego punktu odbioru, program wypisze, że dostarczono 0j wody do każdego punktu odbioru.

W przypadku gdy istnieje rozwiązanie, algorytm zwraca listę punktów odbioru wraz z ilością wody, która do niego dopłynęła.

Złożoność algorytmu

- Załadowanie danych wejściowych do struktur będzie miało złożoność P(E) = O(|E|)
- Ścieżka powiększająca będzie znajdowana za pomocą algorytmu BFS o złożoności O(|E| + |V|).
- Aktualizacja przepływu i przepustowości w najgorszym przypadku będzie miała złożoność O(|E|), gdy ścieżka będzie zajmowała cały graf.
- Maksymalna ilość powtórzeń pętli algorytmu ma złożoność O(|V| * |E|)[1].

W związku z tym cały algorytm będzie miał złożoność O((E+E+V)*V*E) = **O($E^2V + EV^2$)**. W związku z tym zależnie od tego czy graf będzie rzadki czy gęsty, dominował będzie albo składnik wierzchołków, albo krawędzi.

Interfejs użytkownika

Część algorytmiczna projektu zostanie napisana w języku C++. Dane wejściowe będą wczytywane ze standardowego wejścia lub z pliku. Format danych wejściowych jest przedstawiony poniżej.

```
[source count]
[valve count]
[receiver count]
for each source:
     [src performance] list of: [adjacent valve/rcvr index] [edge capacity]
for each valve:
     [vlv performance] list of: [adjacent valve/rcvr index] [edge capacity]
```

W pierwszym wierszu znajduje się liczba źródeł, są to wierzchołki o numerach rozpoczynających się od 1. W drugim wierszu znajduje się numer wierzchołka pierwszego punktu odbioru i liczba punktów odbioru.

Następnie w kolejnych liniach wejścia programu znajdują się charakterystyki kolejnych wierzchołków, rozpoczynając od wierzchołka 1. Znajduje się tam kolejno: przepustowość wierzchołka (tylko dla zaworów), lista par {numer wierzchołka z którym dany wierzchołek tworzy krawędź; przepustowość tej krawędzi}. Punkty odbioru nie są charakteryzowane przez linie wejścia programu.

Struktury danych

Projekt jest napisany w języku C++ z użyciem jednowątkowego programowania obiektowego oraz inteligentnych wskaźników.

Do komunikacji z funkcją main używane są 2 klasy:

---

[1] Dowód na stronie: https://brilliant.org/wiki/edmonds-karp-algorithm/

- **Solver** – służy do:
    - Ładowania danych z pliku (bądź strumienia wejściowego) – metoda **void load_data(istream &)**
    - Rozwiązywania grafu, który jest jego polem - metoda **double solve()**, zwraca czas działania z biblioteki std::chrono
- **Generator** – służy do:
    - Generowania grafu (**Graph**) na podstawie parametrów **void generateGraph()**
    - Przechowywania grafu wynikowego
    - Wypisania grafu (operator wypisania)

Ponadto, w projekcie znajdują się następujące klasy:

- **Graph**
    - Przechowuje strukturę danych std::vector<std::shared_ptr<Vertice>> vertices
        - W tym wektorze wierzchołków są przechowywane wierzchołek s, źródła, zawory, punkty odbioru i wierzchołek t w takiej kolejności
    - Przechowuje wektor wartości przepływów dla każdego odbiorcy
    - Zarządza tymi strukturami danych
- **Vertice**
    - Przechowuje strukturę danych std::unordered_map<int, std::shared_ptr<Edge>> edges
        - W tej mapie znajdują się wszystkie sąsiadujące wierzchołki o numerze większym niż numer danego wierzchołka
    - Przechowuje przepustowość wierzchołka – w przypadku zaworów są one niezerowe, w przypadku pozostałych jest to maksymalna wartość double (czyli tzw. Vertice::infinity())
    - Przechowuje kolor oraz numer wierzchołka-rodzica - wykorzystywane w BFS i generatorze przy szukaniu ścieżek powiększających, żeby uniknąć powtórzenia wierzchołka, łatwo zaktualizować przepustowości oraz przywrócić kolor domyślny (biały)
- **Edge**
    - Reprezentuje krawędź (odcinek) między wierzchołkami. Znajduje się w hash mapie w klasie Vertice. Nie zawiera numerów żadnego wierzchołka, jej położenie w grafie determinuje położenie w strukturach danych (Graph::vertices oraz Vertice:: edges)
    - Zawiera przepustowość oraz wskaźnik na odwrotną krawędź (służy do aktualizowania przepustowości w algorytmie Edmondsa-Karpa)

Wizualizacja danych

Po wykonaniu obliczeń i otrzymaniu wyników przez algorytm, do pliku tekstowego zostają zapisane parametry czasowe wykonania programu. Następnie zostanie wykonany skrypt w języku **Python**, w którym zostanie zaimplementowana wizualizacja danych za pomocą m.in. biblioteki **numpy, matplotlib.pyplot** oraz **pandas** w postaci wykresów dowodzących złożoności algorytmu.

Dane w pliku tekstowym zostają zapisane w formacie:
Ilość zaworów;ilość odcinków;czas rozwiązywania
Przykładowy fragment pliku:
100;268;152090
110;303;203680

120;350;271294

Generator danych testowych

W języku **C++** zostanie przygotowany także generator losowych danych testowych z parametrami liczby źródeł, zaworów, odcinków, punktów odbioru i opcjonalnymi parametrami średniej długości ścieżki powiększającej oraz odchylenia standardowego długości ścieżki powiększającej.

Algorytm

1. Graf przepływowy zawierał będzie na początku
   - niepołączone węzły s i t
   - połączone z s wierzchołki źródeł
   - połączone z t wierzchołki punktów odbioru
2. Do grafu zostaną dodane wierzchołki zaworów w ilości podanej w parametrach
3. Zostanie dodany licznik wierzchołków spójnych z podstawową wersją grafu tak, aby na koniec uspójnić graf
4. W pętli będą dodawane losowo ścieżki powiększające do grafu i zliczać dodane do spójnego grafu wierzchołki i krawędzie. Początkowo ścieżki powiększające będą dodawane w taki sposób, aby dodać jak najwięcej krawędzi, czyli będą tworzone z wierzchołków niepołączonych.

Wyniki projektu

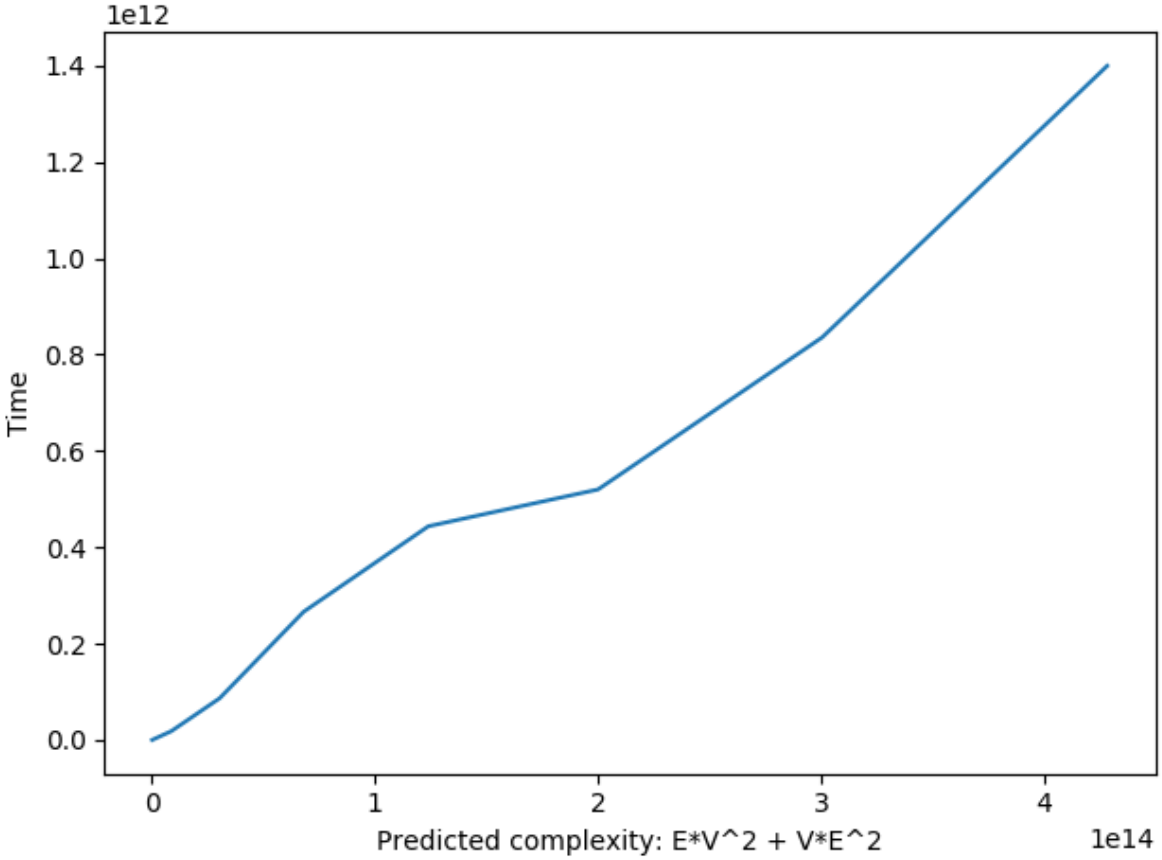
Testowanie poprawności solvera

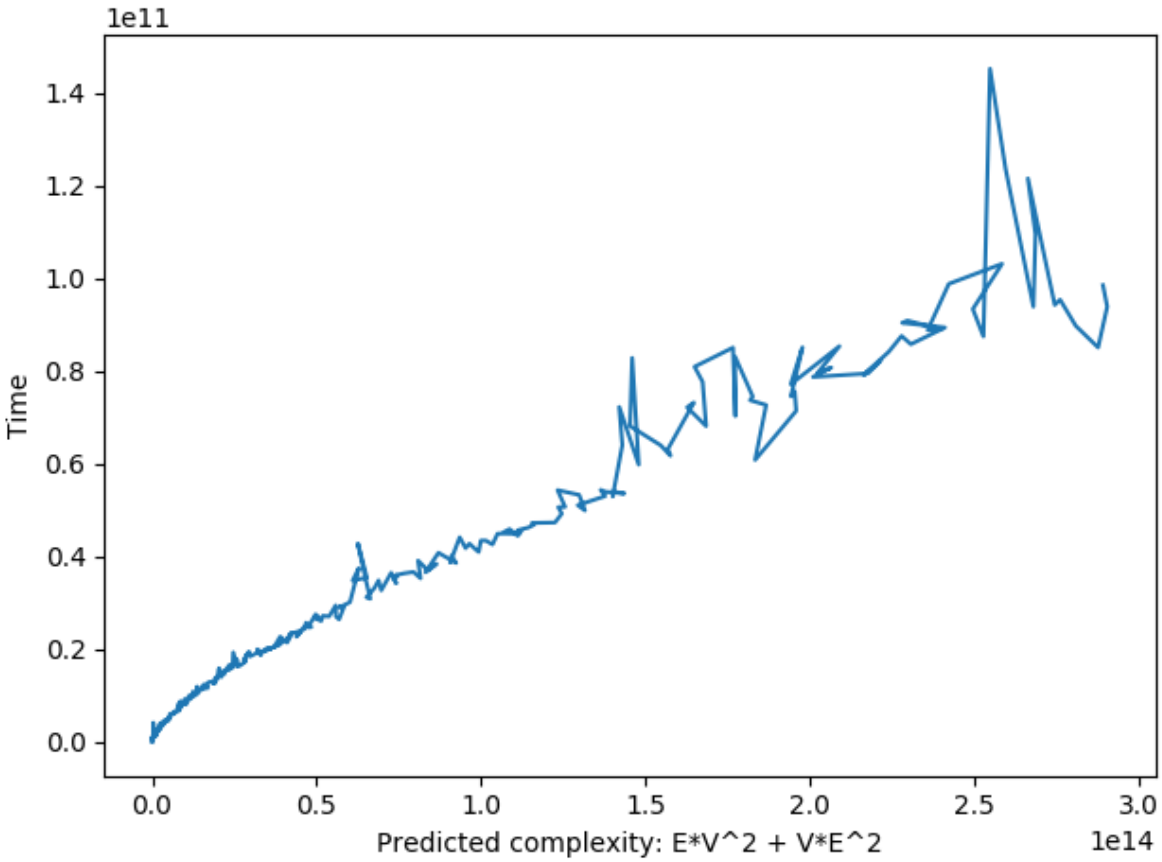
Poprawność solvera została sprawdzona za pomocą prostych przypadków, dla których wyniki zostały obliczone ręcznie. Ponadto, została ona sprawdzona za pomocą generatora (wykonanie ./generate), gdzie generowano graf wraz z rozwiązaniem, następnie rozwiązywano go i sprawdzano, czy wyniki różnią się o więcej niż 10%.

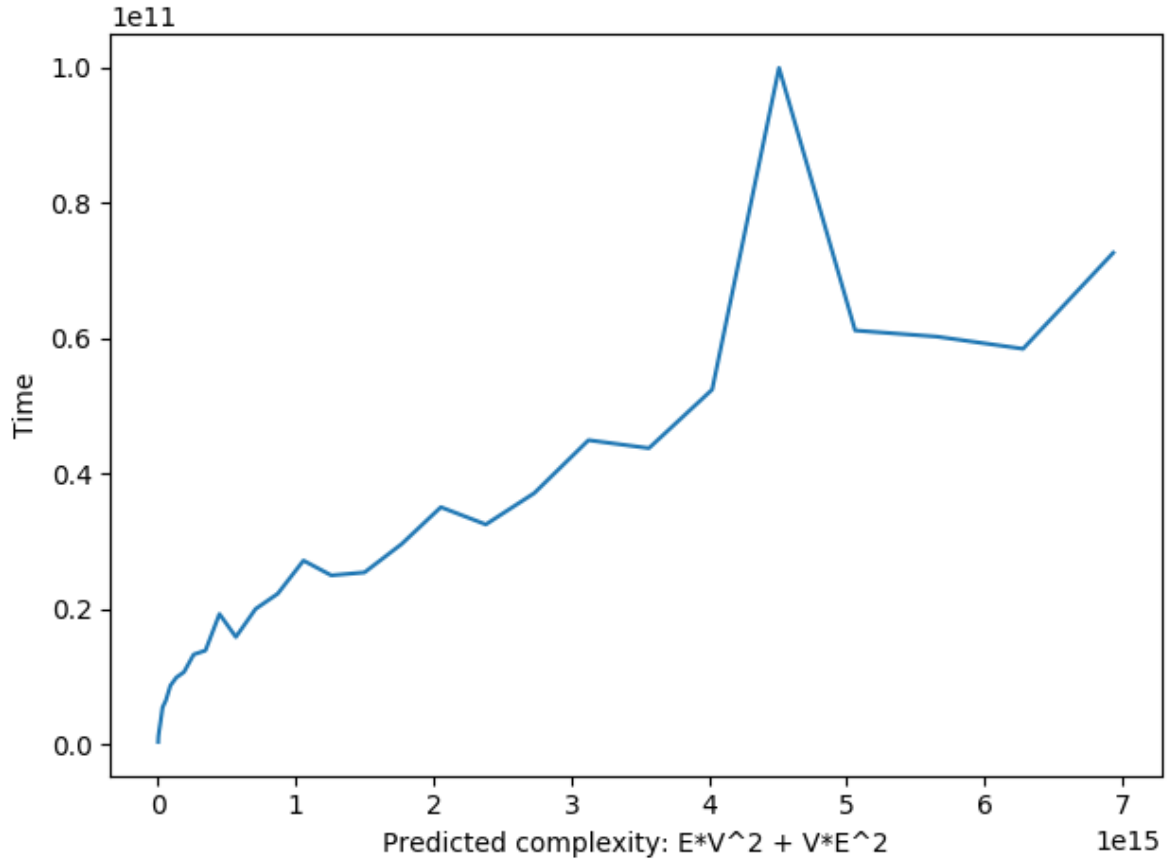
Sprawdzenia dokonano dla ponad 400 próbek, rozpoczynając od ilości zaworów 100 aż do 4170. Dla wszystkich próbek sprawdzenie zakończyło się sukcesem.

Testowanie złożoności

W tym samym uruchomieniu (./generated) sprawdzano czasy wykonania rozwiązywania grafu, które w nanosekundach zapisywano do pliku tekstowego. Czasy wykonania sprawdzano także dla profilerów zaworów oraz odcinków (./valve_profiler i ./Edge_profiler). Następnie za pomocą skryptu **plot.py** w języku Python, narysowano wykresy oczekiwanej złożoności (e*e*v + e*v*v) od faktycznego czasu. Wykresy znajdują się poniżej:


1e12
1.4
1.2
1.0
0.8
0.6
0.4
0.2
0.0
Time
0
1
2
3
4
Predicted complexity: E*V^2 + V*E^2
1e14


1e11
1.4
1.2
1.0
0.8
0.6
0.4
0.2
0.0
Time
0.0
0.5
1.0
1.5
2.0
2.5
3.0
Predicted complexity: E*V^2 + V*E^2
1e14



Wnioski złożoności

Wszystkie wykresy ukazują jednoznacznie, że zależność przewidywanej złożoności od rzeczywistego czasu wykonania jest liniowa. Oznacza to, że złożoność została przewidziana poprawnie i wynosi $O(E*V^2+E^2*V)$.

Ilość zaworów	**Ilość krawędzi**	**Współczynnik q(n)**
10	33	836.276387
20	62	162.710845
30	90	189.427052
40	111	34.872341
50	142	29.051578
60	169	24.269635
70	193	30.600701
80	219	17.692449
90	242	9.851843
100	268	10.667095
110	303	13.579513
120	350	8.289440
130	406	10.269706
140	457	15.354774
150	507	8.478910
160	560	14.290303
170	615	7.346440
180	686	7.243315
190	755	6.029785
200	867	7.096104
210	922	5.087306

220	983	5.399756
230	1072	6.745094
240	1164	5.816640
250	1303	4.377543
260	1359	4.313605
270	1411	4.500278
280	1541	4.592619
290	1560	3.078905
300	1738	3.844035
310	1804	3.200365
320	1899	2.822346
330	2063	2.555941
340	2095	2.954454
350	2296	3.049688
360	2213	2.688857
370	2391	2.660301
380	2622	2.420989
390	2672	2.282649
400	2855	-2.301458
410	2982	1.888858
420	3084	-1.843234
430	3193	-1.784782
440	3294	-1.435550
450	3561	-0.777115
460	3674	-0.476913
470	3790	-0.721402
480	4073	-0.489795
490	4183	0.511037
500	4310	0.456764
510	4511	-0.397850
520	4719	-0.132285
530	4823	-0.091206
540	5130	-2.841940
550	5105	-0.386404
560	5310	-9.722900
570	5552	4.633729
580	5808	-1.323960
590	5915	-1.543239
600	6335	-0.557325
610	6407	0.304020
620	6546	0.775015
630	6678	1.282348
640	6681	1.015073
650	7044	3.550729
660	7318	0.915898
670	7665	-0.779550
680	7862	0.955345
690	7875	-0.636825
700	8181	2.013704
710	8661	-1.896811
720	8756	0.172399
730	8924	-0.407934
740	9172	-1.154488
750	9362	7.977133
760	9781	0.513384
770	9734	-0.364463

780	9929	1.248330
790	10143	-2.517118
800	10604	0.464569
810	10661	-0.810170
820	10852	0.313063
830	11292	0.741864
840	11396	0.438014
850	11837	-3.369876
860	12116	-0.835109
870	12386	7.887625
880	12796	-1.093810
890	12974	-0.639794
900	13345	1.063377
910	13185	-0.486208
920	13316	-0.068058
930	13765	0.639367
940	13965	0.011819
950	14933	0.025924
960	14631	-0.236701
970	15240	-0.216982
980	15595	0.371096
990	15908	2.746206
1000	16486	-2.660248
1010	16166	0.785539
1020	16496	0.471589
1030	17418	-0.763396
1040	17514	-2.257351
1050	17399	-0.342452
1060	17781	-0.884057
1070	18261	-0.441615
1080	18585	3.670950
1090	18801	1.795063
1100	19646	20.230006
1110	20174	-0.613712
1120	20050	0.138076
1130	20470	-3.097701
1140	20601	0.224930
1150	21478	0.728364
1160	21775	0.337787
1170	21481	-0.969256
1180	21739	1
1190	23050	-1.297057
1200	21955	-18.488997
1210	23407	-2.449733
1220	22981	1.986647
1230	23372	-0.853272
1240	25305	0.691277
1250	24356	0.620388
1260	25330	-0.774590
1270	25061	-0.897458
1280	26305	-0.685715
1290	25915	-0.292794
1300	27473	0.741061
1310	26616	-1.526335
1320	27629	2.918887
1330	28140	-0.840676

1340	27937	0.341863
1350	28559	1.257203
1360	29590	0.684533
1370	29322	0.511211
1380	29190	-1.338154
1390	30215	-12.474872
1400	30599	0.900158
1410	31182	2.277801
1420	31455	-0.506707
1430	31928	-8.838742
1440	32532	-1.505128
1450	32398	0.675113
1460	33189	0.259789
1470	33876	1.042131
1480	34189	-0.218879
1490	34604	2.143792
1500	35196	-3.307937
1510	36062	2.556594
1520	35245	-1.522177
1530	36036	-0.512555
1540	36343	0.525022
1550	37524	-1.468270
1560	38317	1.696165
1570	39287	-0.652414
1580	38713	-0.456004
1590	38729	-0.404731
1600	40328	0.349003
1610	40546	0.649927
1620	41889	-1.267614
1630	40401	-1.100047
1640	41336	0.700108
1650	43313	-0.940706
1660	42856	-0.935234
1670	43251	2.326919
1680	44657	1.002325
1690	45364	2.051560
1700	44742	4.197478
1710	45852	67.925149
1720	45081	0.521545
1730	46404	0.970201
1740	46887	0.632149
1750	46626	0.897869
1760	47939	-0.224183
1770	48149	-1.049618
1780	48708	-1.965858
1790	50222	1.047460
1800	49958	0.345199
1810	51561	1.294769
1820	51465	0.453885
1830	50944	-0.822779
1840	52749	0.855023
1850	51240	1.387897
1860	52940	-4.403283
1870	53827	-1.134030
1880	54856	-0.273530
1890	56287	-0.105611

1900	56159	-0.216822
1910	55900	-1.757067
1920	57766	2.160545
1930	57018	0.875403
1940	58586	-0.721046
1950	59834	1.586927
1960	59645	-1.460789
1970	59798	0.695469
1980	60943	0.078204
1990	61339	-0.182388
2000	63244	1.630260
2010	62394	0.198438
2020	64714	1.108815
2030	64081	0.228305
2040	63867	7.661200
2050	66284	2.147353
2060	65773	-0.464456
2070	66973	0.799064
2080	67575	-1.023071
2090	67781	-6.071418
2100	68438	0.941532
2110	68730	14.910154
2120	70478	3.047659
2130	70177	-1.133698
2140	70589	1.015362
2150	72073	0.264090
2160	72450	0.969513
2170	70920	0.670117
2180	73736	1.079702
2190	72770	1.698732
2200	74672	-1.069609
2210	73855	-0.708871
2220	75613	-1.245374
2230	77926	-4.666968
2240	76954	-0.829196
2250	76469	-0.435211
2260	78973	1.127205
2270	78298	-0.421713
2280	80490	-1.544089
2290	81519	1.054582
2300	81949	-1.265328
2310	83512	-0.505563
2320	81010	-0.548153
2330	82625	-0.575563
2340	83619	1.689572