Obliczeniowe badanie oczekiwanego zachowania dzielników pierwszych liczb naturalnych

Maciej Dębski

9 stycznia 2016

Podstawy teoretyczne

Jak wiadomo z podstaw teorii liczb, każdą liczbę naturalną $n \in \mathbb{N}$ można przedstawić jako

$$n = p_1 \cdot p_2 \cdot p_3 \cdot \ldots \cdot p_k, \quad p_i \in \mathbb{P}.$$

Będziemy tutaj rozpatrywać podobne przedstawienie, tylko uporządkowane według rozmiaru p_i i uzupełnione jedynkami do stałej długości K:

$$n = a_1 \cdot a_2 \cdot \ldots \cdot a_K, \quad a_i \in \mathbb{P} \cup \{1\}, a_1 \geqslant a_2 \geqslant \ldots \geqslant a_K.$$

Takie przedstawienie zawsze istnieje dla odpowiednio dużego K, np. dla $K > \log_2(n)$.

Będę badał średni rozmiar dzielników a_i w porównaniu z n. W szczególności ciekawe wyniki daje badanie ich logarytmów o podstawie n. Możemy zapisać

$$n = n^{\gamma_1} \cdot \ldots n^{\gamma_K}, \quad \gamma_i = \log_n a_i.$$

Oczywiście, po zlogarytmowaniu stronami otrzymujemy

$$1 = \gamma_1 + \ldots + \gamma_K. \tag{1}$$

Zdefiniujmy teraz badane średnie. Niech $\gamma_i(n), i \in \{1, \ldots, K\}$ oznacza odpowiednią wartość z rozkładu n, jak powyżej. Dla ustalonego N, przyjmijmy

$$\bar{\gamma}_i = \frac{1}{N} \sum_{n=1}^{N} \gamma_i(n).$$

Obliczenia wskazują, że dla każdego i, ciąg $\bar{\gamma}_i$ jest zbieżny dla $N \to \infty$. Ponadto, suma tych ciągów wydaje się szybko zbiegać do 1, co jest odpowiednikiem równości (1) dla średnych.

Metoda obliczeń

Obliczenia przeprowadziłem dla $N = 10^9$ i $K = 30$, na pojedynczym komputerze z procesorem i7-3537U (2.00GHz) i 10GB pamięci RAM. Program napisałem w C++, a następnie mocno zoptymalizowałem i sprofilowałem - początkowe wersje działały kilka rzędów wielkości wolniej. Ostateczne obliczenia trwały poniżej 10 minut. Kod programu oraz dane wynikowe jest dostępny w repozytorium projektu: https://github.com/mdebski/atl

Algorytm

Jednym z głównych problemów przy pisaniu programu było kontrolowanie zużycia pamięci. Tablica liczb (uint32_t) wielkości $N = 10^9$ zajmuje $4GB$ pamięci, i dwie takie tablice już się nie mieściły, przy standardowym zużyciu pamięci przez system. Stąd na przykład niemożliwe było zapisanie pełnego rozkładu na czynniki pierwsze wszystkich rozpatrywanych liczb.

Problem podzieliłem na dwie części. W pierwszej za pomocą sita znajdowałem pewien dzielnik pierwszy dla każdej z liczb. Sprytna implementacja sita pozwoliła na osiągnięcie gwarancji że jest to najmniejszy dzielnik pierwszy, bez ograniczania prędkości sita. Następnie łatwo obliczyć największy dzielnik pierwszy - przechodząc dynamicznie tablicę od najmniejszych liczb, możemy obliczyć:

$$MaxPrimeDivisor[n] = max(MinPrimeDivisor[n], MaxPrimeDivisor[n/MinPrimeDivisor[n]]).$$

W następnym etapie korzystając z obliczonej tablicy największych dzielników obliczałem $\bar{\gamma}_1, \ldots \bar{\gamma}_K$. Warto zauważyć, że wszystkie dzielniki pierwsze n, wraz z krotnościami, uporządkowane od największego, to:

$$p_1 = MaxPrimeDivisor[n]$$

$$p_2 = MaxPrimeDivisor[n/p_1]$$

$$p_3 = MaxPrimeDivisor[n/p_2]$$

$$\ldots$$

W ten sposób dla każdej liczby obliczałem wszystkie dzielniki i odpowiednie wyrażenia $\log(p_i)/\log(n)$ i dodawałem je do akumulatorów.

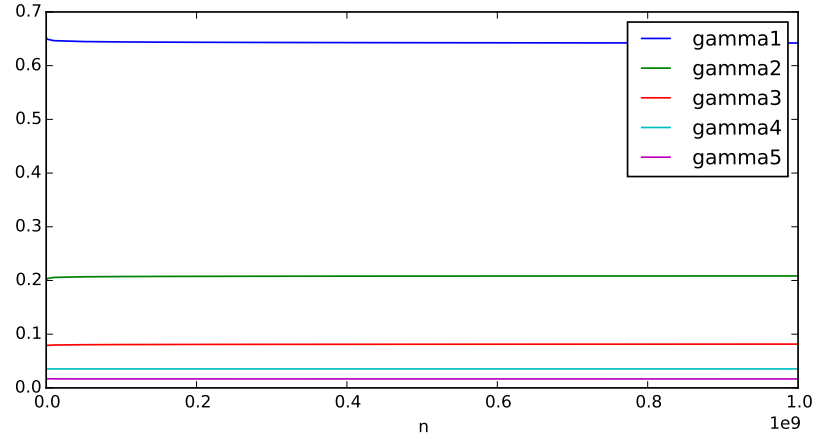
Optymalizacje

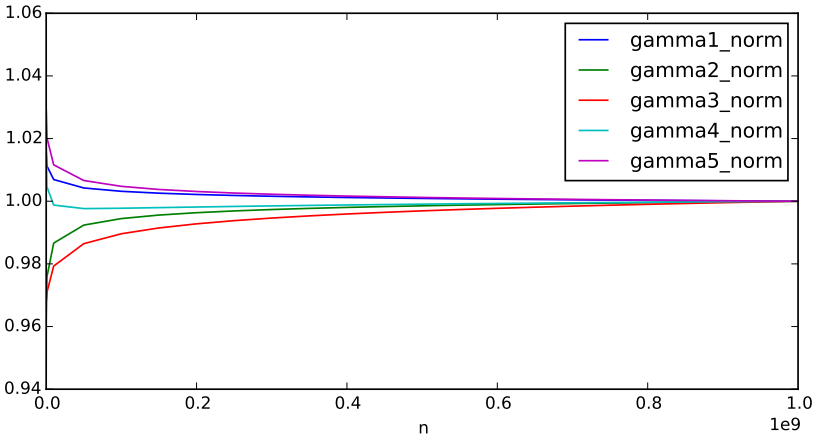
W pierwszej części zoptymalizowałem zużycie pamięci rezygnując z pamiętania całej tabeli dla sita - zamiast tego, trzymałem tylko dzielniki dla liczb względnie pierwszych z $15 = gcd(2, 3, 5)$, a dla pozostałych obliczałem je na bieżąco - dzieląc przez $2, 3, 5$. Pozwoliło to na zużycie $15/phi(15) \simeq 4$ razy mniej pamięci. Dla przyspieszenia poprzez zwiększenie lokalności odwołań do pamięci użyłem sita segmentowanego, według http://primesieve.org/segmented_sieve.html.

W drugiej części istotną część kosztu stanowiło obliczanie logarytmów. Z tego powodu wpierw stablicowałem wartości logarytmów liczb pierwszych mniejszych od 10^9. Jako że jest ich tylko około $5 \cdot 10^7$ nie stanowiło to dużego narzutu pamięciowego. Do ich przechowywania istotnie lepiej zadziałała mapa boost::container::flat_map niż std::map. Ponadto, zmodyfikowałem wewnętrzną pętle by nie obliczała $\log(n)$ dla każdego p_i, a tylko jednokrotnie. Oczekiwałem że kompilator sam to zrobi w ramach optymalizacji wspólnych podwyrażeń, ale się przeliczyłem.

Wyniki

Szacowanie $\bar{\gamma}_i$





Zależności pomiędzy $\bar{\gamma}_i$