Projekt 2 – liczby pierwsze

Hubert Nowak & Wiktor Szczeszek

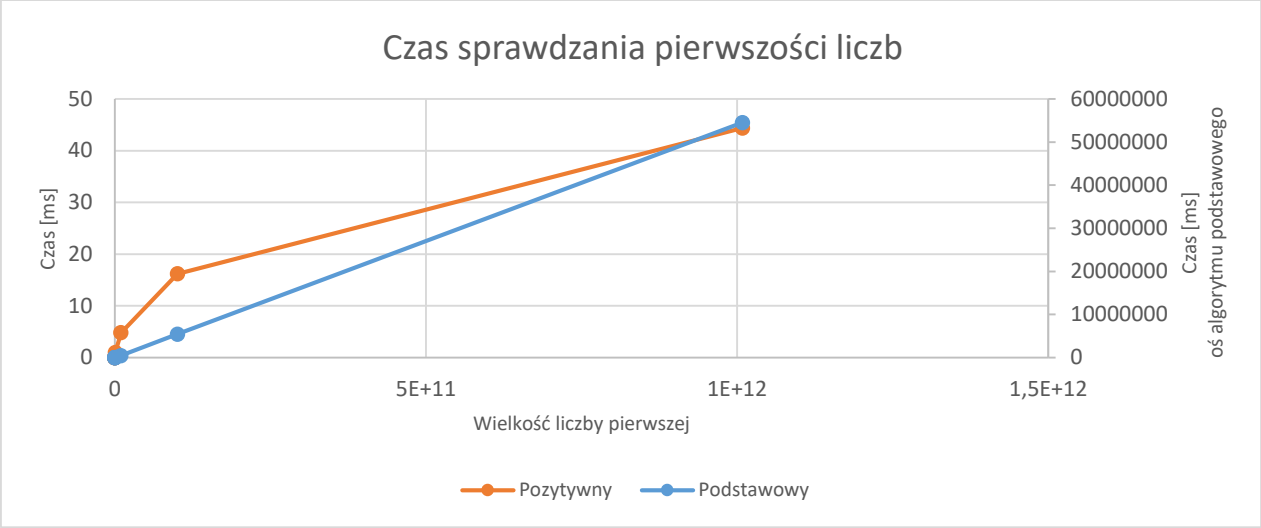
Projekt ten skupia się na analizie algorytmów sprawdzających liczby pierwsze. W ramach projektu podany został podstawowy algorytm w notacji O(n), a na jego podstawie mieliśmy za zadanie zaprojektować wydajniejsze odpowiedniki. Każdy z badanych algorytmów poddanych został pomiarom czasowym lub pomiarom tick'ów procesora oraz instrumentacji, czyli zliczaniu punktów krytycznych. Do tego przedsięwzięcia wykorzystaliśmy IDE Microsoft Visual Studio Community 2019 w wersji 16.5.3, .NET framework 4.8.03761, język programistyczny C# oraz serwis hostingowy GitHub. Sprzęt dokonujący pomiarów: (AMD Athlon II X4 651, NVIDIA GeForce GTS 450, 8 GB RAM), (Intel Core i5-6600K, NVIDIA GeForce GTX 1050 Ti, 16 GB RAM).

Istota liczby pierwszej

Liczba pierwsza jest to liczba naturalna, która dzieli się bez reszty tylko przez jeden oraz przez samą siebie. W odniesieniu do programowania liczby pierwsze maja wiele zastosowań głównie w kryptografii. Dlatego to w jaki sposób są one generowane lub sprawdzane wpływa na wydajność wielu programów. Istnieje szeroka gama algorytmów liczb pierwszych, a niektóre z nich poddane zostaną badaniu w tym projekcie zarówno jak i stworzone przez nas inne ich warianty.

Sprawdzanie pierwszości liczby

Na sprawdzenie pierwszości liczby jest wiele sposobów, natomiast najbardziej podstawowym z nich jest sprawdzanie podzielności. W języku komputerowym takie działanie wykonuje operacja modulo, która jest bardzo wymagająca dla procesora w stosunku do innych działań arytmetycznych. W zagadnieniach projektu została podana lista liczb pierwszych: { 100913, 1009139, 10091401, 100914061, 1009140611, 10091406133, 100914061337, 1009140613399 }, który jest podstawą badania wydajności naszych algorytmów.



WYKRES 1.1 – PORÓWNANIE CZASÓW SPRAWDZANIA PIERWSZOŚCI LICZB DLA ALGORYTMÓW PODSTAWOWEGO I POZYTYWNEGO.

Algorytm podstawowy został podany w projekcie jako baza do kolejnych badań. Jest on złożoności liniowej a jego czas działania jest bardzo długi, bez względu na optymalizację opierającą się na omijaniu liczb parzystych. Na wykresie powyżej zobrazowany on został osią pomocniczą, aby zachować proporcje.

Funkcja regresji liniowej

Ze względu na bardzo długi czas sprawdzenia ostatniej z podanej liczb, postanowiliśmy ten czas zaprognozować. Wykorzystaliśmy do tego funkcję regresji liniowej, czyli funkcję, która na podstawie zestawu danych X i Y oblicza wzór funkcji liniowej.

TIME	
a	b
0,00005405782826739	-22845,3
n = a*p + b	
p	n
1009140613399	54529105

TICKS	
a	b
0,12762893697974	-14006578,78
n = a*p + b	
p	n
1009140613399	128781537172

INSTR	
a	b
0,25	-0,422124385833740
n = a*p + b	
p	n
1009140613399	252285153352

LEGENDA:	współczynnik a	współczynnik b	wsp. wyznaczania	błąd oceny Y	błąd względny Y

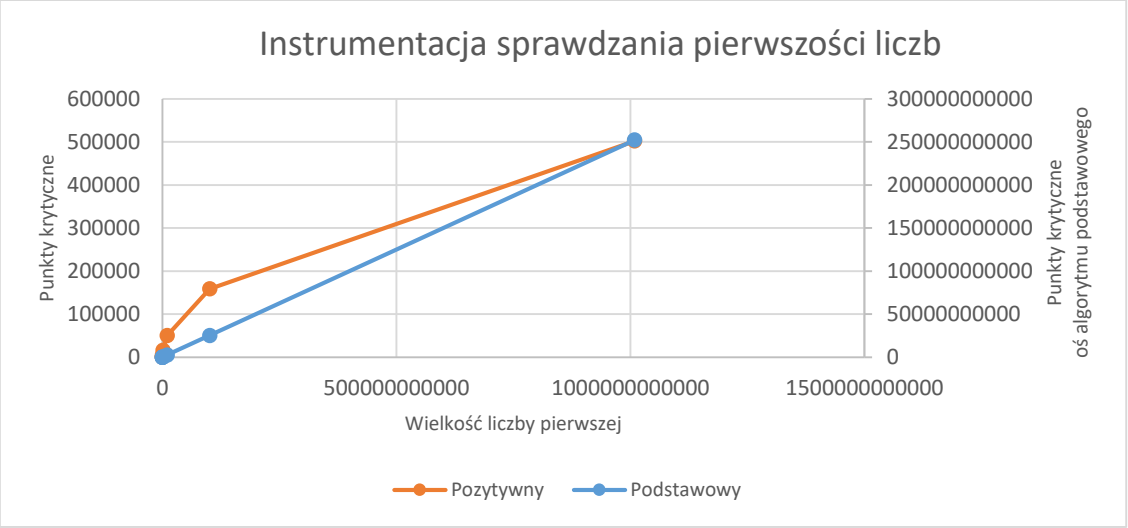
TIME		
wynik funkcji LINEST() z danych liczb 1-7		
0,000054057828267388 3	-22845,30975	
0,000000462	17721,38	
0,999634474	42596,89	0,7829%
13673,89919	5	
24811223240381	9072475559	
wynik funkcji LINEST() z danych liczb 1-8		
0,000054057828267388 3	-22845,30975	
0,0000000417	14936,32	
0,999996438	38885,46	0,0713%
1684256,172	6	
2546728825445660	9072475559	

TICKS		
wynik funkcji LINEST() z danych liczb 1-7		
0,1276289369800800	-14006578,78	
0,000291	11162597,49	
0,99997397	26831546	0,2086%
192104,66	5,00	
1383022647382510 00000	3599659211006810,00	
wynik funkcji LINEST() z danych liczb 1-8		
0,1276289369797420	-14006578,78	
0,0000262	9408305,36	
0,99999975	24493738	0,0190%
23662121,66	6	
1419592902852990000 0000	3599659211005820	

INSTR		
wynik funkcji LINEST() z danych liczb 1-7		
0,25	-0,4	
0,00	0,11	
1	0,256436	0,0000000010%
8069615637262890000000	5,00	
5306534659696390000000	0,33	
wynik funkcji LINEST() z danych liczb 1-8		
0,25	-0,4	
0,00	0,09	
1	0,234649	0,0000000001%
9892586262620200000000	6,00	
5446851471283350000000	0,33	

RYSUNEK 1.1 - TABELE WYNIKÓW OBLICZANIA FUNKCJI REGRESJI

Do tego celu użyliśmy funkcji LINEST (funkcja REGLINP w polskiej wersji) w programie MS Excel. Jak widzimy na rysunku otrzymaliśmy współczynniki, na których podstawie obliczamy prognozowany czas szukania. Sposób ten cechuje się bardzo małym błędem oszacowania, który zmniejsza się tym bardziej, im większe liczby chcemy prognozować.

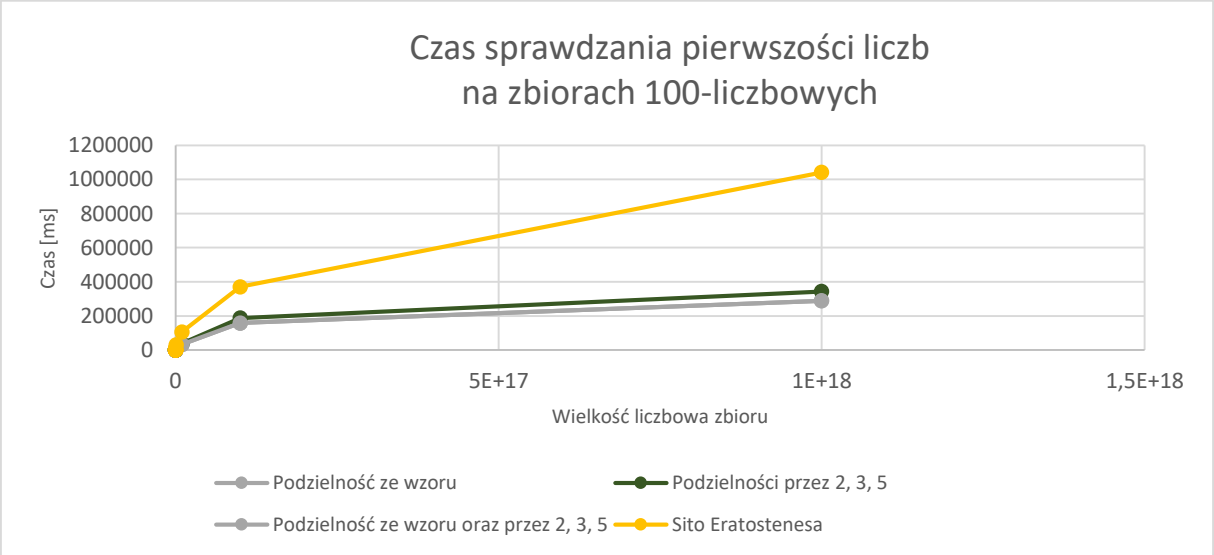


WYKRES 1.2 – PORÓWNANIE INSTRUMENTACJI SPRAWDZANIA PIERWSZOŚCI LICZB DLA ALGORYTMÓW PODSTAWOWEGO I POZYTYWNEGO.

Algorytm pozytywny posiada drobną różnicę względem algorytmu podstawowego, a mianowicie jego zakres poszukiwań dzielników liczby jest równy pierwiastkowi sprawdzanej liczby, co diametralnie skraca czas działania. Instrumentacja również odzwierciedla te zależność mniejszą liczbą operacji krytycznych.

W poszukiwaniu wydajności

Test wydajności wyłącznie na liczbach pierwszych jest bardzo skuteczny, jednak w celu zbadania również uniwersalności algorytmów warto je przebadać na zbiorach liczbowych. Podczas takiego testu program musi się również wykazać szybkością eliminowania liczb złożonych, co przyspiesza jego selekcję. W naszym doświadczeniu każdy z algorytmów ma za zadanie przeanalizować zbiory stu kolejnych liczb o wielkości od 1 do 10^{18}, a pomiary czasu wykonywane są w pięciu próbach.



WYKRES 1.3 – PORÓWNANIE CZASÓW SPRAWDZANIA PIERWSZOŚCI LICZB NA ZBIORACH 100-LICZBOWYCH DLA ALGORYTMÓW POSŁUGUJĄCYCH SIĘ WARIANCJAMI ZASAD PODZIELNOŚCI ORAZ SITA ERATOSTENESA.

Algorytm podzielności ze wzoru oparty jest o sprawdzanie podzielności liczb sąsiednich dla wielokrotności liczby 6, można to zobrazować wzorem: P = 6 * k ±1. Oczywiście nie każda liczba wygenerowana tym wzorem jest liczbą pierwszą, natomiast przynajmniej jedna ze sąsiadujących nią jest.

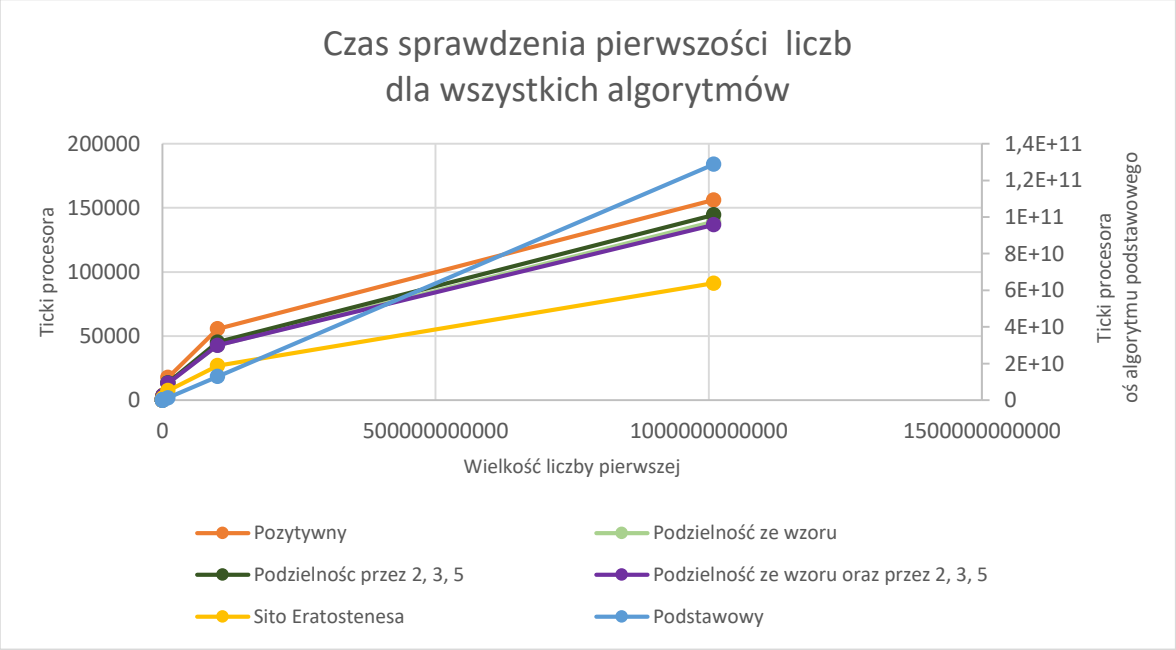
Algorytm podzielności przez 2, 3, 5 dotyczy zasad podzielności liczb. W tym przypadku sprawdza, czy liczba jest podzielna przez 2 (dla liczby, która na miejscu dziesiętnym dzieli się przez 2 bez reszty), 3 (dla liczby, która po zsumowaniu liczb systemu dziesiątkowego dzieli się przez 3 bez reszty), 5 (dla liczby, która na miejscu dziesiętnym dzieli się przez 5 bez reszty).

Algorytm podzielności ze wzoru oraz przez 2, 3, 5 to hybryda obu wcześniej wymiennych sposobów sprawdzania pierwszości liczby. Łączy ze sobą wzór podzielności oraz zasady podzielności przez 2, 3 i 5.

Algorytm Sita Eratostenesa to dobrze znany sposób na liczby pierwsze. Tworzy on tablicę oraz zapełnia liczbami od 2 do pierwiastka z szukanej liczby, a następnie wykreśla z niej iteracyjnie wielokrotności kolejnych liczb (pierwszych), jednocześnie sprawdzając ich podzielność.

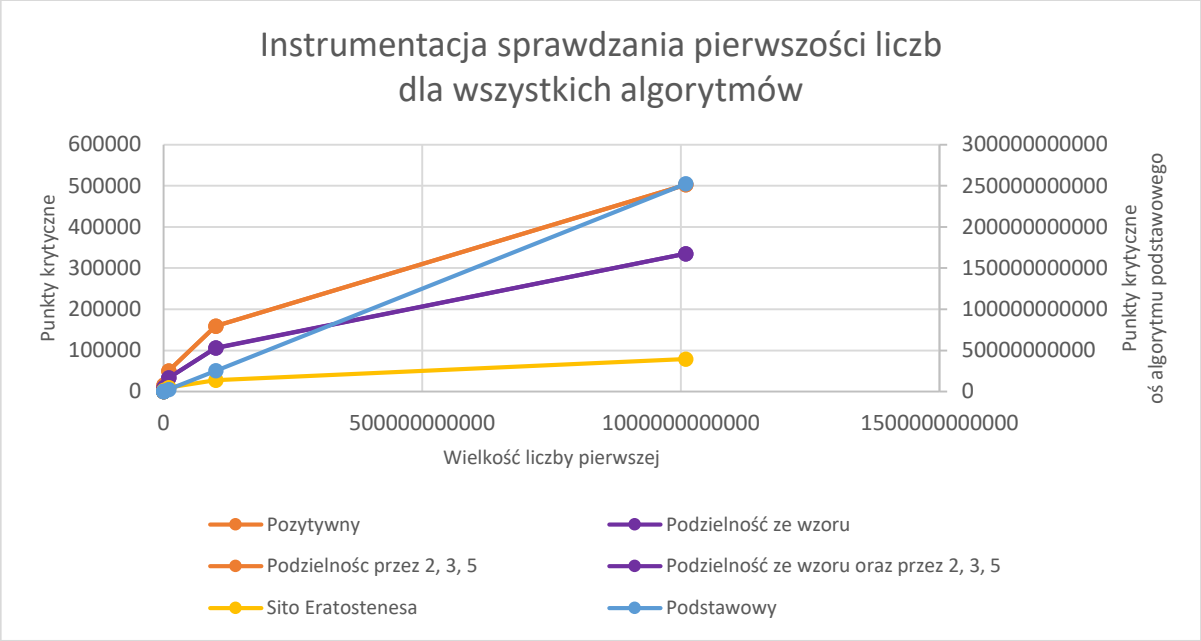
Ostateczne zestawienie

Powracając do naszej listy liczb podanej w projekcie i zestawiając z nim wszystkie opisane algorytmy, prezentują się one następująco:



WYKRES 1.4 – PORÓWNANIE CZASÓW SPRAWDZANIA PIERWSZOŚCI LICZB DLA ALGORYTMÓW PODSTAWOWEGO I POZYTYWNEGO, WARIANTÓW PODZIELNOŚCI ORAZ SITA ERATOSTENESA.

Na tle pozostałych najszybszy okazuje się być algorytm sita Eratostenesa. Przewaga jest znacząca, natomiast we wcześniejszym badaniu (wykres 1.3) sytuacja była odwrotna. Powodem tego było działanie na zbiorach liczb, dla których za każdym razem algorytm generował nowe tablice zwiększając liczbę operacji. Widać również, że pozytywny algorytm wzbił się na górną część wykresu co wskazuje na dość powolne działanie, pomimo dużego wzrostu wydajności względem podstawowego algorytmu (wykres 1.1).



WYKRES 1.4 – PORÓWNANIE INSTRUMENTACJI SPRAWDZANIA PIERWSZOŚCI LICZB DLA ALGORYTMÓW PODSTAWOWEGO I POZYTYWNEGO, WARIANTÓW PODZIELNOŚCI ORAZ SITA ERATOSTENESA.

Niektóre wartości punktów krytycznych pokrywają się ze sobą, algorytmy te mają po prostu bardzo podobne działanie oraz wydajność. Natomiast zależność ze wcześniejszego wykresu (wykres 1.4) się sprawdza i sito Eratostenesa jest najmniej obciążające dla procesora w przypadku zliczania operacji modulo jako operacje krytyczne.

Podsumowanie

Algorytmy przedstawione w tym projekcie to tylko odsetek możliwości w dziedzinie liczb pierwszych. Każdy algorytm ma swoje zastosowanie, co udowodniło sito Eratostenesa. Działało one znakomicie w sprawdzeniu pierwszości liczb w liście liczb pierwszych podanych w projekcie, natomiast przeciwnie dla zbiorów liczbowych składających się również z liczb złożonych. Oczywistym faktem jest bezużyteczność algorytmu podstawowego – podanego w projekcie, natomiast jego drobna modyfikacja skróciła działanie i była podstawą dla następnych rozwiązań.

GitHub: https://github.com/wiQ1999/School-algorithms/tree/master/Projekt2_Liczby_Pierwsze/Projekt2_Liczby_Pierwsze