Sito Eratostenesa w języku C

Ogólnie o sicie Eratostenesa

Algorytm, opracowany przez Eratostenesa, służy do wyszukiwania liczb pierwszych (liczb mających dokładnie dwa dzielniki). Najbardziej naturalna metoda określania, czy liczba jest pierwsza, polega na sprawdzeniu jej podzielności przez wszystkie liczby naturalne mniejsze od niej. Eratostenes zauważył, że można ograniczyć ilość dzielników. Doszedł on do wniosku, że mając daną liczbę naturalną N, w celu wyznaczenia wszystkich liczby pierwszych mniejszych od N, należy znaleźć największą liczbę pierwszą która spełni warunek $p^2<=N$ (gdzie p – liczba pierwsza), a wszystkie większe od niej sprawdzić pod względem podzielności przez liczby pierwsze mniejsze lub równe p.

Przykładowo dla N=100:
p=7 (bo $7^2=49$, a $11^2=121$)
Liczby pierwsze mniejsze od p:
$p_1=2$
$p_2=3$
$p_3=5$

Sprawdzamy liczby od 8 do 100. Jeśli dzielenie daje resztę inną niż 0, dla każdej z powyższych liczb pierwszych, użytych w postaci dzielników, liczba jest pierwsza. Dodatkowo ze zbioru otrzymanych liczb należy odrzucić 1, która ma tylko jeden dzielnik.
Łatwo zauważyć, że największa z liczb pierwszych, uzyskanych w wyniku tych działań, spełnia warunek $p_{najwyższa}^2<=N^2$. Oznacza to, że dysponując wynikami wcześniejszych obliczeń, możemy znaleźć nieskończenie wiele liczb pierwszych.

Adaptacja algorytmu do kodu źródłowego

Zapiszmy powyższy przykład w postaci kodu źródłowego w języku C. Kod prezentuje jedno z najprostszych możliwych rozwiązań.

```
--------------ŹRÓDŁO 1--------------
   #include<stdio.h>
   main()
   {
   int i;
   int liczp[99];       //tablica dla odnajdywanych liczb pierwszych
   int illiczb,nilliczb;//zmienne zliczające ilość liczb na tablicy liczp[]
   int liczba;//zmienna przechowująca sprawdzaną liczbę

   liczp[1]=2;//wpisujemy na początkowe pozycje tablicy liczby pierwsze
   liczp[2]=3;//dla N=100
   liczp[3]=5;
   liczp[4]=7;
   illiczb=4;//wpisujemy do zmiennej aktualną ilość liczb
   nilliczb=illiczb;

   for(liczba=8; liczba<=100; liczba++){

   if((liczba%liczp[1]!=0) && (liczba%liczp[2]!=0) && (liczba%liczp[3]!=0)
   && (liczba%liczp[4]!=0)){
   nilliczb++;
   liczp[nilliczb]=liczba;  //jeśli sprawdzana liczba nie jest podzielna
                            //przez wprowadzone początkowo do tablicy
                            //liczby pierwsze to zapisujemy ja do tablicy
                            //i zwiększamy zmienną nilliczb; zmienna
                            //illiczb (liczba dzielników) nie zmienia się
```

```
    }
    }

    for(i=1; i<=nilliczb; i++){
    printf("%i) %i\n",i,liczp[i]);  //po skończeniu przesiewu wyświetlamy
                                    //zawartość tablicy
    }

    getchar();
    }
```

Optymalizacja kodu – wstęp do przeszukiwania kolejnych zakresów

Zmodyfikujmy powyższy kod tak, aby mógł służyć do sprawdzania liczb większych od 100 na podstawie wyników wcześniejszych zakresów.
Dla N=100 mamy tylko cztery dzielniki, więc możemy pozwolić sobie na wypisanie każdego warunku oddzielnie. Ilość dzielników wzrasta jednak po sprawdzeniu każdego kolejnego zakresu. Musimy więc zmodyfikować ten fragment. Dzięki zmiennym `illiczb` i `nilliczb` wiemy ile jest liczb na tablicy. Możemy zastosować pętle for dla dzielenia przez kolejne liczby pierwsze. Spowoduje to jednak, że wykonywane będzie dzielenie zawsze tyle razy ile liczb mamy na tablicy, pomimo tego, że sprawdzana liczba okaże się podzielna przez którąś z nich. Jeśli dzielenie przez 3 lub 5 da resztę 0 to dalsze dzielenie nie jest uzasadnione. Zastosujmy więc pętlę while. Dodajmy zmienną `flaga` która będzie regulowała działanie pętli. Określmy, że pętla będzie wykonywana gdy `flaga!=0`.

Wartość 0 należy w tej sytuacji przypisać zmiennej `flaga`, gdy:
- w wyniku dzielenia otrzymamy resztę 0;
- wykonamy dzielenie tyle razy, ile liczb mieliśmy na tablicy przed rozpoczęciem przesiewu;

Aby umożliwić sprawdzanie drugiego z powyższych warunków wprowadźmy zmienną j, która będzie zliczała wykonania pętli. Sprawdzana liczba będzie pierwsza, jeśli wykonamy pętle tyle razy ile mamy na tablicy liczb i nie otrzymamy reszty 0 w ostatnim dzieleniu. Wprowadźmy zmienną `flagapoz`, której zadaniem będzie zliczanie powtórzeń pętli, w których osiągnęliśmy resztę inną niż 0.

Dodajmy jeszcze funkcję `pauza`, która zatrzyma działanie programu po wyświetleniu każdych 23 linii wyników i umożliwi wyświetlenia kolejnych wyników lub wyjście z programu.

```
--------------ŹRÓDŁO 2 --------------
#include<stdio.h>
#include<stdlib.h>
#include<conio.h>

long int liczp[9999999];      //zwiększamy rozmiar i zakres tablicy
long int illiczb,nilliczb;
long int liczba;
long int minz, maxz;
//dodajemy zmienne minz i maxz do obsługi pętli "for" w funkcji "szukaj"

short int ilwierszy;//dodajemy zmienną "ilwierszy" dla funkcji "pauza"

void init(){

liczp[1]=2;
liczp[2]=3;
liczp[3]=5;
liczp[4]=7;
illiczb=4;
```

```
nilliczb=illiczb;
printf("%i\n%i\n%i\n%i\n",liczp[1],liczp[2],liczp[3],liczp[4]);
ilwierszy=4;
//wpisujemy do tablicy początkowe liczby pierwsze dla N=100
// i wypisujemy je na ekranie
}
void pauza()
{
char znak;
if(ilwierszy==23){
printf("[Enter] - następna strona wyników; [k] - koniec programu");
if((znak=getchar())!='k'){
ilwierszy=0;
}
else{
exit(0);
}
}
}

void szukaj()
{
short int flaga;
long int flagapoz, j;

illiczb=nilliczb;

for(liczba=minz; liczba<=maxz; liczba+=2)
{
flaga=1;
flagapoz=0;
j=0;

while(flaga!=0)
{
j++;
if(j>illiczb || liczba%liczp[j]==0){
flaga=0;
}
else{
flagapoz++;
}
}

if(flagapoz==illiczb){
nilliczb++;
liczp[nilliczb]=liczba;
printf("%i\n",liczba);  //jeśli liczba jest pierwsza to ją wypisujemy
ilwierszy++;            //zwiększamy zmienną przechowującą ilość
                        //wykorzystanychwierszy na ekranie
pauza();                //i odwołujemy się do funkcji “pauza”
}
}
}
main()
{
init(); minz=9; maxz=100;
szukaj();
minz=101;maxz=10000;
szukaj();
minz=10001;maxz=100000000;
szukaj();

getchar();
}
```

Jeszcze wyższe zakresy

Część liczb ze zbioru od 100000001 do 10^{16} wykracza poza zakresy zmiennych w których są przechowywane sprawdzane wartości. Komendy zostaną wykonane $10^8 * 10^8$, czyli 10^{16} razy. Dla przeszukania zakresu liczb do 10^{32} potrzebne będzie dodanie kolejnych pętli.

Problem wykonywania obliczeń na sprawdzanej liczbie wymaga bardziej złożonego rozwiązania. Wykorzystajmy metodę, używaną w bankowości do sprawdzania poprawności numerów rachunków bankowych. Postępując zgodnie z nią, modulo z danej liczby możemy obliczyć, dzieląc sprawdzaną liczbę na części. Gdy to zrobimy obliczamy modulo z pierwszej części, a wynik dopisujemy na początku następnego fragmentu liczby. Obliczamy modulo z otrzymanej liczby i ponownie wynik umieszczamy na początku kolejnej części liczby.

Przykład:

Szukamy wyniku następującego działania: 21357648 % 7
Podzielmy liczbę na części:
21 – 35 – 76 – 48
Obliczmy modulo z pierwszego fragmentu liczby:
21 % 7 = 0
Dopiszmy wynik na początku drugiego fragmentu i obliczmy modulo:
035 % 7 = 0 (liczymy jak 35 % 7)
Powtórzmy powyższe czynności aż do uzyskania ostatecznego wyniku:
076 % 7 = 6
648 % 7 = 4

Otrzymaliśmy ostatecznie 4, więc reszta z dzielenia 21357648 przez 7 wynosi 4.

Tą metodę można wykorzystać do potrzeb programu odpowiednio go modyfikując. Problemem, oprócz formy zapisu liczb, jest też miejsce na ich zapis. Ilość liczb pierwszych jakie zostaną odnalezione z zakresu do 10^{16} jest tak duża, że należy zastanowić się nad ich zapisem poza programem (na przykład w zewnętrznym pliku).
Największa odkryta dotychczas liczba pierwsza składa się z ponad 7 milionów cyfr (poprzednia zawierała ponad 6 milionów cyfr, a wcześniejsza ponad 4 miliony). Dla pierwszej osoby (lub zespołu), która odnajdzie liczbę pierwszą dłuższą niż 10 milionów cyfr, przewidziana jest nagroda 100000 dolarów (nagroda dla osoby, która jako pierwsza odnalazła liczbę pierwszą mającą więcej niż milion cyfr, wynosiła 50000 dolarów).

Obecnie największa liczba pierwsza składa się z **22 338 618 cyfr** i jest większa o ponad 5 mln cyfr od poprzedniej. Jest tzw. liczbą Mersena - czyli ma postać:

$$2^{74207281} - 1$$