SITO LICZB PIERWSZYCH Z WYKORZYSTANIEM CIĄGÓW LICZBOWYCH O PRZYROŚCIE CYKLICZNYM.

MAREK MATUSIAK I PAWEŁ RZECHONEK

STRESZCZENIE. Tematem pracy jest przedstawienie algorytmu oraz ogólnej metody generowania liczb pierwszych z wykorzystaniem ciagów liczbowych o cyklicznym charaktererze wartości przyrostów wyrazów ciągu, do zadanej wartości N. Efektywność implementowanego w języku Java algorytmu jest co najmniej trzykrotnie większa od dotychczas znanych algorytmów generujących liczby pierwsze. Przedstawiony algrytm jest jedną z możliwych wersji realizujacą, przedstawioną metodę generowania i nie musi być optymalny.

1. WPROWADZENIE.

Przewaga przedstawionego algorytmu nad innymi dotychczas stosowanymi algorytmami, polega na tym, że wygenerowany wstępny zbiór liczb dla wartości pomocniczej k o następujących własnościach $P_k!<n<P_{(k+1)}!$ jest zbiorem liczb niepodzielnych przez wszystkie liczby pierwsze do wartości k .

Dodatkowo jednokrotne usunięcie liczb podzielnych przez daną liczbę pierwszą z mniejszego zbioru (cyklu), usuwa wszystkie mogące się pojawić liczby o większej wartości podzielne przez dana liczbę pierwszą.

Nie występują również testy podzielności liczby. Liczby są usuwane ze zbioru na podstawie iloczynu odpowiednich wcześniej wygenerowanych liczb.

Algorytm pracuje w dwóch etapach. W pierwszym etapie dla zadanej liczby maksymalnej N , generowany jest zbiór liczb niepodzielnych przez kolejne liczby pierwsze do wartości k .

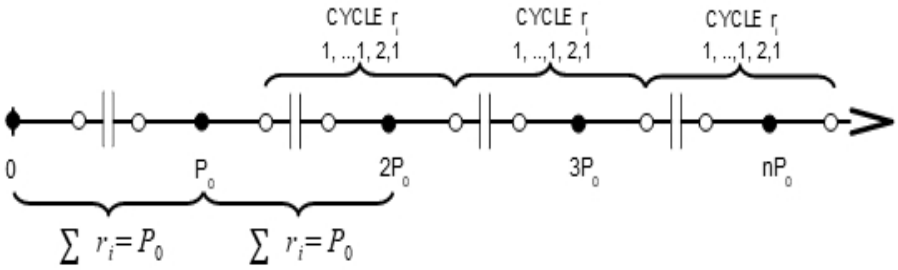
Drugi etap polega na eliminacji z wygenerowanego zbioru liczb złożonych uzyskanych z kombinacji iloczynów liczb zawartych w zbiorze, aż do wartości całkowitej $\sqrt{N}$.

Algorytm ma złożoność co najwyżej liniową. $T(n)=O(f(n))$

2. CZĘŚĆ TEORETYCZNA.

2.1. Generowanie ciągu liczb niepodzielnych przez pojedynczą liczbę P_0 .

Przyjmijmy liczbę naturalną $P_0>1$. Aby wygenerować wszystkie liczby niepodzielne (pierwsze), względem P_0 , należy wytworzyć ciąg liczbowy o następujących cyklicznych własnościach przyrostów r_i .



1. illustration: Generating for one number.

$a_0=0,\ a_1=a_0+1,\ a_2=a_1+1\ \ldots\ a_{(P_0-1)}+2+a_{P_0}+1+\ldots+a_{(nP_0-n)}+2\ldots$

Ciąg $a_{(i+1)}=a_i+r_i$ ma następujące wartości przyrostów r_i dla ciągu.

$r_0=1,\ r_1=1,\ r_2=1,\ \ldots\ r_{(P_0-1)}=2,\ r_{P_0}=1,\ +\ldots+r_{nP_o-n}=2,\ \ldots$

Ciąg liczb pierwszych względem P_0 będzie miał cykliczne przyrosty r_i o sumie przyrostów w cyklu, równym P_0 .

$$\sum_{i=0}^{(P_0-1)} r_i=P_0$$

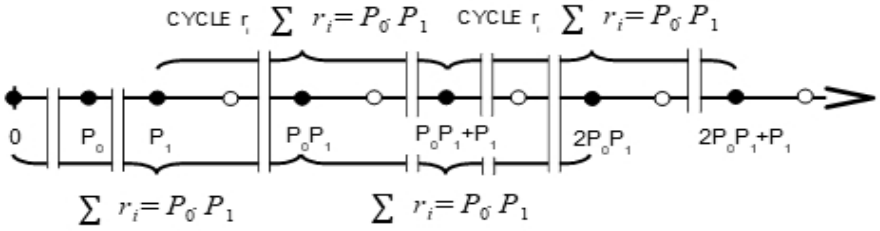
$$r_k=\left\{\begin{array}{c} P_0\,|a_{k+1}\rightarrow r_k=r_k+r_{k+1} \\ n>k\rightarrow r_n=r_{n+1} \end{array}\right\}$$

Zasada ogólna generowania takiego ciągu dla pojedynczej liczby P_0 , przydatna w konstruowaniu algorytmu, może być przedstawiona w następujący sposób:
Dla ciągu liczb naturalnych o stałym przyroście równym 1 należy wartość przyrostu dla wyrazu ciągu podzielnego przez P_0 dodać do przyrostu wyrazu poprzedzającego. Wyraz ciągu podzielny przez P_0 przestaje istnieć.

Wygenerowane liczby zachowają własność niepodzielności przez P_0 w dowolnie wielkim zbiorze liczb.

2.2. Generowanie ciągu liczb niepodzielnych dla większej ilości liczb względnie pierwszych.

Przyjmijmy dowolną liczbę P_1 pierwszą względem P_0 oraz $P_1>P_0>1$. Aby wygenerować wszystkie liczby pierwsze, względem P_0 lub P_1 . Należy wytworzyć ciąg liczbowy o następujących cyklicznych własnościach przyrostów r_i .



2. illustration: Generating for two numbers.

Przyrosty wyrazów ciągu mają charakter cykliczny. Ciąg liczb niepodzielnych

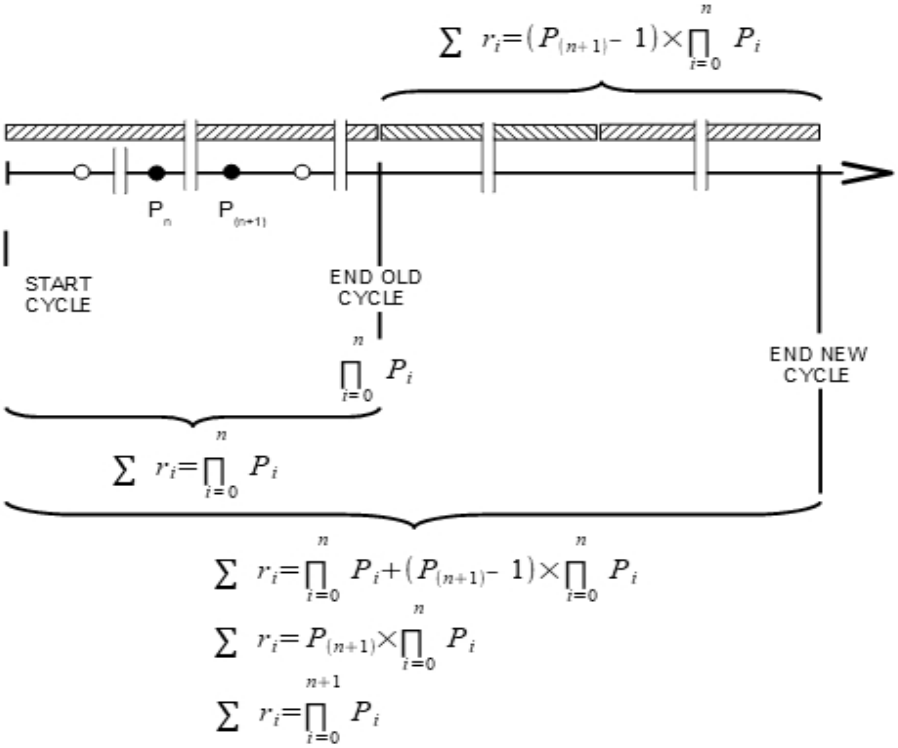
(pierwszych) względem P_0 lub P_1 będzie miał cykliczne przyrosty r_i o sumie przyrostów ciagu w cyklu, równym iloczynowi P_0 i P_1 . Wygenerowane liczby zachowają własność niepodzielności przez P_0 lub P_1 w dowolnie wielkim zbiorze liczb.

$$r_k=\begin{Bmatrix} P_0 \mid a_{k+1} \vee P_1 \mid a_{k+1} \rightarrow r_k = r_k + r_{k+1} \\ n > k \rightarrow r_n = r_{n+1} \end{Bmatrix}$$

Dla liczb podzielnych przez P_0 lub P_1 należy wartość przyrostu dla wyrazu podzielnego przezlu P_0 lub P_2 dodać do przyrostu wyrazu poprzedzającego. Wyraz ciagu podzielny przezlu P_0 lub P_1 przestaje istnieć.

Rozumowanie takie można przeprowadzić dla większej ilości liczb względnie pierwszych. Uogólnienie rozumowania można przeprowadzić dla pojęcia znanego cyklu oraz dodatkowej liczby względnie pierwszej.

2.3. Generowanie ciągu liczb niepodzielnych dla znanego cyklu przyrostów dla n liczb względnie pierwszych o początku cyklu przyrostów w 0 i znanego przebiegu przyrostów do wartości ciągu $\prod_{i=0}^{n} P_i$ oraz dodatkowej liczby względnie pierwszej $P_{(n+1)}$ większej od P_n .



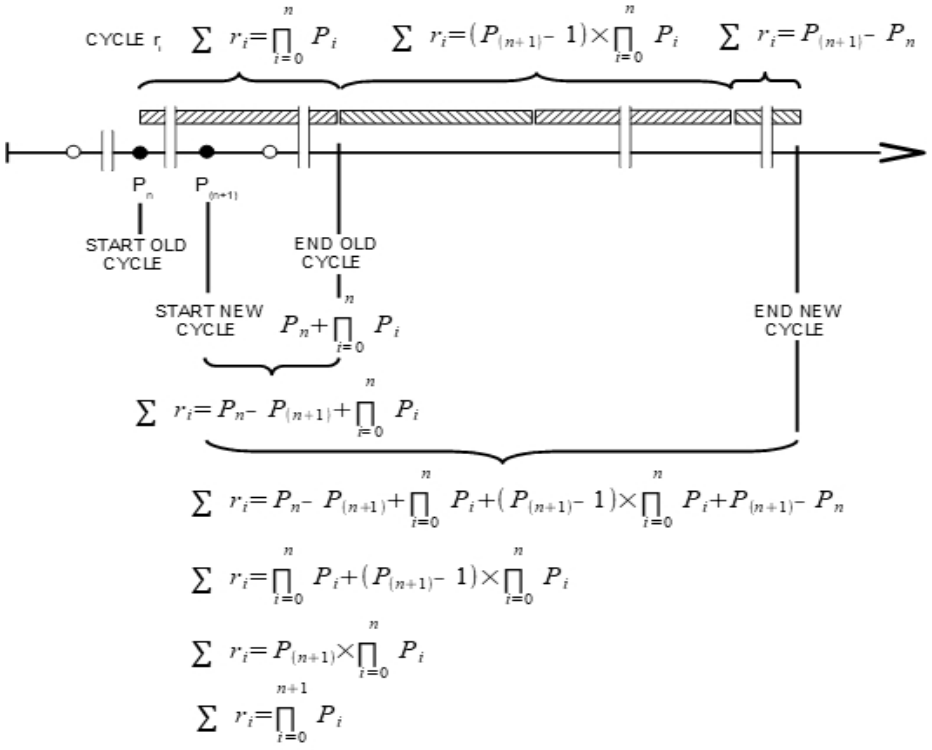
3. illustration: Generating for a number and known cycle, start at 0.

Aby wytworzyć nowy cykl przyrostów z początkiem cyklu w 0 dla nowej liczby $P_{(n+1)}$ należy, poprzedni cykl przyrostów powtórzyć $P_{(n+1)}-1$ razy.
Otrzymany nowy cykl przyrostów będzie miał sumę przyrostów równy iloczynowi $\sum r_i=\prod_{i=0}^{n+1} P_i$. Z nowo otrzymanego cyklu przyrostów należy usunąć wszystkie wyrazy ciągu podzielne przez $P_{(n+1)}$ sumując wartość ich przyrostu do wyrazu poprzedzającego.

2.4. Generowanie ciągu liczb niepodzielnych dla znanego cyklu przyrostów dla n liczb względnie pierwszych o początku cyklu przyrostów w P_n i znanego przebiegu przyrostów do wartości ciągu $P_n+\prod_{i=0}^{n} P_i$ oraz dodatkowej liczby względnie pierwszej $P_{(n+1)}$ większej od P_n .

Dla każdego ciagu o cyklicznych przyrostach r_i , do pełnego przedstawienia przebiegu ciągu wystarczające jest podanie wartości dowolnego pierwszego wyrazu cyklu oraz wartości przyrostów ciągu w pełnym cyklu (analogicznie do właściwości funkcji okresowych $f\left(x+T\right)=f\left(x\right)$ gdzie T jest okresem funkcji).

Taka właściwość ciągów cyklicznych jest istotna przy generowaniu liczb pierwszych ponieważ wartości przyrostów w ciągu od 0 do P_n powinny różnić się od wartości wygenerowanych przyrostów cyklicznych (chcemy by wyrazy od P_0 do P_{n+1} występowały w zbiorze liczb), generowanie nowego cyklu przyrostów należy więc rozpocząć od znanego cyklu przyrostów z zakresu od P_n do wartości $P_n+\prod_{i=0}^{n} P_i$ tak aby zachować warunek pełnego cyklu. Początkowy wyraz cyklu przyjmie wartośc P_n a długość cyklu przyrostów. $\sum r_i=\prod_{i=0}^{n} P_i$



4. illustration: Generating for a number and known cycle, start at P_{n+1}.

Aby wytworzyć nowy cykl przyrostów z początkiem cyklu dla nowej liczby $P_{(n+1)}$ należy, poprzedni cykl przyrostów powtórzyć $P_{(n+1)}-1$ razy oraz dodatkowo ciąg przyrostów zawarty pomiędzy P_n i $P_{(n+1)}$. Otrzymany nowy cykl przyrostów będzie miał sumę przyrostów równy iloczynowi $\sum r_i=\prod_{i=0}^{n+1} P_i$. Z nowo otrzymanego cyklu przyrostów należy usunąć wszystkie wyrazy ciągu podzielne przez $P_{(n+1)}$ sumując wartość ich przyrostu do wyrazu poprzedzającego.

2.5. Generowanie liczb pierwszych.

Ponieważ wszystkie liczby pierwsze są wzajemnie pierwsze stąd, należy je generować zakładając, że dla wygenerowanej ostatniej liczby pierwszej następna wygenerowana liczba względnie pierwsza będzie liczbą pierwszą. Liczba taka jest niepodzielna przez wszystkie wygenerowane liczby mniejsze od tej liczby co wynika z właściwości przedstawionego procesu generowania.

Generowanie liczb należy rozpocząć od najmniejszej liczby pierwszej czyli liczby 2 dla której przyrost r_i odpowiada przyrostom liczb naturalnych czyli 1 .

Ze względu na ograniczenia fizyczne ciąg liczb niepodzielnych zostanie ograniczony do wartości maksymalnej. Taki ciąg ograniczony stanie się zbiorem liczb niepodzielnych.
Ponieważ przyrost długości cyklu w zależności od ilości liczb przyrasta w funkcji

$P_n!+P_n$, po przekroczeniu końca generowanego cyklu wartości maksymalnej zbioru, pozostaje usunięcie ze zbioru wszystkich kombinacji iloczynów liczb, wygenerowanych do zbioru, większych od P_n oraz mniejszych od wartości maksymalnej zbioru wygenerowanych liczb.

Ze względu na to że $P_n * P_n$ jest najmniejszą możliwą liczbą usuwaną ze zbioru (wygenerowane P_n zostanie zachowane w zbiorze) Proces selekcji zbioru powinien się odbywać do czasu gdy spełniony jest warunek $P_n^2 \leqslant P_{max}$.

2.6. Algorytm generujący liczby pierwsze.

Algorytm generujący liczby pierwsze może zostać wytworzony w dwojaki sposób.

- ✔ Algorytm oparty na tablicach liczbowych. W takim przypadku pomijane są liczby podzielne zachowane jednak są wartości przyrostów dla danych liczb. W takim przypadku występują dwie tablice liczbowe lub jedna dwuwymiarowa. Jedna z tablic (kolumn) zawiera wartość liczby a druga tablica (kolumna) zawiera wartość przyrostu.

- ✔ Algorytm oparty na tablicach logicznych prawda/fałsz, gdzie fałsz oznacza liczbę złożoną a prawda liczbę pierwszą. W takim wypadku pomijane są wartości przyrostów a podzielne liczby przyjmują wartość logiczną fałsz. Adres komórki odpowiada wartości liczby.

Poniższa propozycja algorytmu generującego liczby pierwsze jest wykonana z zastosowaniem tablic logicznych. W tym przypadku pojęcie wyrazów ciagu oraz przyrostów wyrazów nie jest wykorzystywane. Zachowana jes logika postępowania. Kolejnym wyrazom ciągu oraz przyrostom odpowiadają cykle wartości $\{true \,/\, false\}$, gdzie adres komórki tablicy jest odpowiednikie liczby.

2.7. Proces generowania.

Aby wygenerować nowy cykl względem poprzedniego znanego cyklu należy poprzedni cykl dodać $p_{(n+1)}-1$ razy oraz dodatkowo cykl pomiędzy $p_{(n+1)}-p_n$. W tym celu wykorzystana jest właściwość warunku cykliczności $p_n = p_n - l$ gdzie l to długość cyklu. Długość cyklu można obliczyć jako końcowy adres wygenerowanego cyklu minus początkowy adres cyklu $l = e - p_n + 1$, gdzie e adres końcowy cyklu a p_n adres początkowy cyklu. Koniec nowego cyklu znajdzie się na pozycji $e_1 = l \cdot p_n + p_{(n+1)} - 1$ a początek w p_n .

2.8. Proces czyszczenia.

Z wygenerowanego nowego cyklu należy usunąć wszystkie liczby podzielne przez p_n . Ponieważ w cyklu nie występuja liczby podzielne przez $p_i \quad i \in \langle 2,\, n-1 \rangle$, do usunięcia pozostana jedynie iloczyny p_n oraz liczb wygenerowanych w cyklu pierwotnym. Są to jedyne liczby złożone podzielne przez p_n w nowym cyklu.

3. ALGORYTM MR-SIEVE - TABLICA BOOLEANOWA

Wejście:

n - Liczba określająca wartość maksymalną przedziału generowania liczb pierwszych, $n \in N \wedge n > 1$.

Wyjście:

$s[i]$ - (TRUE) Kolejne liczby pierwsze w przedziale od 2 do n .

Zmienne pomocnicze:

$s[i]$ - tablica wartości logicznych. $s[i] \in \{true / false\}\ dla\ \ i = 0, 1, 2, \ldots, n.$

i - Przebiega przez kolejne indeksy elementów $s[i]$.

p - Aktywna liczba pierwsza (początek cyklu).

e - Koniec cyklu.

Metody:

$nextPrime(p,\ s[i])$

Metoda zwraca wartość następnej pozycji i dla której $s[i] = true$ dla $i > p$.

Metoda:

$nextPrime(p,\ s[i])$

$i \leftarrow p$

do

$i \leftarrow i + 1$

while $!\,s[i]$

return i

$cleaner(p,\ e,\ s[i])$

Metoda nadaje wartość $false$ wszystkim iloczynom liczby p oraz liczby i dla której $s[i] = true$ z zakresu od e do p .

Metoda:

$cleaner(p,\ e,\ s[i])$

for $i \leftarrow e$ **to** $i \geq p$ $i--$

if $s[i]$

$s[i * p] = false$

return $s[i]$

Metoda:

$generator(p, e, n, s[i])$

Proces wypełniania tablicy $s[i]$ wartościami $\{true / false\}$ aż do wartości $e_1 = l \cdot p + nextPrime(p) - 1$ lub n. W pętli dla wartości i z zakresu $r \leqslant i \leqslant e_1$, $s[i] = s[i-l]$.

Zmienne pomocnicze generator:

l - Wielkość cyklu. $l = e - p + 1$

r - Początek odkładania cyklu. $r = e + 1$

e_1 - Koniec nowego cyklu. $e_1 = l \cdot p + nextPrime(p, s[i]) - 1$

$generator(p, e, n, s[i])$

$l \leftarrow e - p + 1$

$r \leftarrow e + 1$

$e_1 = l \cdot p + nextPrime(p, s[i]) - 1$

if $e_1 \geq n$

$e_1 \leftarrow n$

for $i \leftarrow r$ **to** $i \leq e_1$

if $s[i]$

$s[i] = s[i-l]$

return $s[i]$

Program:

$Prime(n)$

$n \leftarrow$ Wczytanie wartości maksymalnej.

$s[n+1] \leftarrow$ Zadeklarowanie tablicy logicznej $s[i]$ $(0, 'n)$.

$s[2] = s[3] = s[5] = true \leftarrow$ Nadanie wartości początkowych wyrazom tablicy.

$p \leftarrow 2,\ e \leftarrow 2$ Nadanie wartości p i e

do Pętla wykonywana do wartości $p * e \leqslant n$

$generator(p,\ e,\ n,\ s[i])$	Generowanie.
$cleaner(p,\ e,\ s[i])$	Czyszczenie
$e = p*l + nextPrime(p,\ s[i]) - 1 \leftarrow$ $p = nextPrime(p,\ s[i]) \leftarrow$	Nadanie nowych wartości e i p

while $p*e \leqslant n$

$generator(p,\ e,\ n,\ s[i])$	Dodatkowy obieg generatora

do Pętla wykonywana do wartości $p*p \leqslant n$

$cleaner(p,\ \frac{n}{p},\ s[i])$	Czyszczenie
$p = nextPrime(p,\ s[i]) \leftarrow$	Nadanie nowej wartości p .

while $p*p \leqslant n$

return $s[i]$

3.1. Złożonosc obliczeniowa.

Wartośc pomocnicza:
Niech k jest liczbą o następujących własnościach $P_k! < n < P_{(k+1)}!$.
Generowanie:
Ilość odczytów n .
Ilosc zapisów n .
Ilość odejmowania $n + 2\mathrm{k}$.
Ilość porównań $< n$
Ilość dodawania $n + k$
Ilość mnożenia k
Ilość NextNumber k
Czyszczenie :
Ilość odczytów $n - \frac{n}{\ln(n)}$
Ilosc zapisów. $n - \frac{n}{\ln(n)}$

NextNumber:
Ilość odczytów $\frac{\sqrt{n}}{\ln(\sqrt{n})}$.

Całościowa ilość operacji $5 \cdot n + 5 \cdot k + 2 \cdot (n - \frac{n}{\ln(n)}) + \frac{\sqrt{n}}{\ln\sqrt{n}}$

Ponieważ $\lim \frac{k}{n} \to 0$ oraz $\frac{n}{\ln(n)} < n$

$$T(n) = O(f(n))$$

3.2. Przykładowa realizacja algorytmu w języku C++.

```
//=======================================================
// MRsieve - prime generator boolean
// Data: 10.2017
// (C) 2017 mgr inz. Marek Matusiak, mgr Pawel Rzechonek
//=======================================================
#include <iostream>
#include <ctime>
#include<fstream>
#include <utility>
#include <algorithm>

using namespace std;

// ----------------------chooser---------------------
int nextPrime(int p, bool s[])
{
    int i = p;
    do
    {
        i++;
    }
    while (!s[i]);
    return i;
}
// ------------------- end chooser --------------------
// -----------------------generator----------------------------
void generator(int p, int e, int n, bool s[])
{
    int l = (e - p + 1);
    int start = e + 1;
    int end = (p) * l + nextPrime(p,s) - 1;
    if (end >= n || end < 0)
    {
        end = n;
    }
    for (int i = start; i <= end; i++)
    {
        s[i] = s[i - l];
    }
}
// -------------------end generator-----------------------
unsigned int e, p, n;
// ---------------------cleaning---------------------------
void cleaning(int p, int e, bool s[])
{

    for (int i = e; i >= p; i--)
    {

        if (s[i])
        {
```

```
            s[i * p] = false;
        }
    }
}

// ----------------end cleaning-----------------
int  main()
{
    bool *s;
    cout << "Enter the maximum number: ";
    cin >> n;
    clock_t start = clock();
    s= new bool[(max(n + 1u, 6u))];

    e = 2;
    p = 2;
    s[2] = true;
    s[3] = true;
    s[5] = true;
    do
    {

        generator(p, e, n, s);
        cleaning(p, e, s);
        e = p * (e - p + 1) + nextPrime(p, s) - 1;
        p = nextPrime(p, s);

    }
    while ((p * e <= n && p * e > 0));

    generator(p, e, n, s);

    do
    {
        cleaning(p, n/p, s);
        p = nextPrime(p, s);
    }
    while (p * p <= n);
    float timeA =(float)(clock() - start)/1000;
    cout << ( "Action time  = " );
    cout << timeA << " s";
    //---------------------------- RECODR TO FILE -------------------------
    ofstream record ("prime.txt");
    int k=0;
    for (int i=0; i<= n; i++)
    {
        if(s[i])
        {
            k+=1;
            record << i <<" ";
            if (k%25==0)
            {
                record << "\n";
            }
        }

    }
    record.close();

    delete [] s;
    return 0;
}
```