Курсова работа по САА

13.Алгоритми за генериране на прости числа. Решето на Ератостен.

Алгоритмите за генериране на прости числа са част от „Computational number theory“. Има много алгоритми за намирането или генерирането на прости числа и всичките са част от „Computational number theory“, то ги обединява. Примери за такива алгоритми са – Решето на Ератостен, решето на Сундарам и решето на Аткин. Тези алгоритми използват така наречените методи - решетата на колелото (wheel sieves).

Нормалната версия на решето на Ератостен се смята за едно от най-лесните алгоритми за имплементиране, но не е най-бързия алгоритъм понеже числата се увеличават лавинообразно и е със сложност O(n log log n). А решето на Аткин има специализирана версия която използва част от метода на Ератостен и част от метода на колелото и може да изчислява задачи с O(*N/*log log *N*), и фактически е по ефективен от нормалната версия на Ератостен. Алгоритъма на Сундарам е почти същия както алгоритъма на Ератостен, но с голяма разлика, че при този метод се изпълнява същото условие, но с повече операции и е изключително бавен и неефективен от гледна точка да бързината на програмата и е със сложност O(n log n). Трябва да се вметне, че съществува подобрена версия на алгоритъма на Ератостен който е най-бързия от всичките и е със сложност O(n). Като заключение може да се каже, че решето на Аткин се доближава до подобрената версия на Ератостен и е по-бърз от обикновената версия, но не го надминава за сега.

Простите числа се разглеждат в математиката още от дълбока древност и са свързани с много интересни задачи далеч извън нейните предели.В информатиката те намират приложение в криптирането, архивирането и т.н.

Дефиниция на просто число :

* Еднопросто число се нарича просто, ако няма други делители освен 1 и себе си, като числото ч не се счита за просто. Ако не е просто, то се нарича съставно.

Доказано е от Евклид, че простите числа са безброй много.

Едно от приложенията на простите числа е за криптиране на данни, които се предават през „Несигурна“ мрежа (финансови транзакции в Интернет).Някои алгоритми за кодиране на предаваната информация използват произведение на големи прости числа.За да бъде „разбит“ такъв канал за предаване на информация, трябва да бъдат известни самите прости числа, а не само тяхното произведение. Ако разгледаме числото 55, веднага можем да се досетим че се разлага като – произведение на 5 и 11. За числото 4853 трудно бихме могли да възтановим на ум простите му делители (211 и 23), но бихме могли да съставим програма която ги намира. Ако се опитаме да възстановим произведението на 100 цифрени прости числа с програма , по който и да е алгоритъм , ще отнеме неприемливо дълго време.

Когато се разглежда редицата на простите числа , възникват някои въпроси :

1. Колко прости числа има в даден интервал [a,b]
2. Каква част от безкрайността представляват простите числа ?
3. Съществува ли формула за намиране на n-тото поред просто число ?
4. Защо намирането на точната формула за π(x) ще даде отговор и на трите въпроса за простите числа.

Отговора на четвъртия въпрос, е че :

* Ако имаме алгоритъм или формула чрез който да можем да изразим числото „π“ което е със стойност 3.14159… (безкрайност след запетаята) и по някакъв начин изразяването на π го свържем с получаването на поредното просто число, тоест начин по който изразяваме безкрайността , а безкрайността на простите числа започва от -> 2,3,5,7,11,13,17 ...+ ∞, то така ние ще намерим всички прости числа в безкрайността, следователно винаги ще успеем да намерим простите числа които са в дадена граница например простите числа в интервала [5 , 37]. Така ние отговорихме на 1-ви въпрос.

Формулата за безкрайност изразена чрез уравнение относно „π“ ще ни даде отговор и на 2-рия въпрос понеже ако сметнем че в интервала [2, ∞] има ∞-ен брой прости числа то като заклучение можем да кажем че броя на всички числа делен на броя на всички прости числа в интервала [0 , ∞], ще даде каква част от безкрайността ще са простите числа.

Алгоритъма относно π(x) ако съществуваше то той ще може да ни даде което си поискаме просто число от ∞ по тази причина ако съществува формула за намиране на n-тото поред просто число например (числото ) 5 то тази формула ще бъде еквивалентна на формулата π(x)=5 (или което и да било друго просто число). Примера за 5 се явява частен случай който ще се съдържа в общия (тоест общия случай ще може да намери частния, но по различна формула ), но не и обратното.

За да се измисли алгоритъм за генериране на прости числа може да се подходи чрез генериране на число в интервала [0, ∞] и след това числото да бъде проверявано дали е просто чрез алгоритъма на Уилсън .

#include<stdio.h>

#include<stdlib.h>

#include<math.h>

int\* numberGenerator()

{

int \*array = (int\*)malloc(10000 \* sizeof(int));

for (int i = 1;i<10000; i++)

{

if (isPrime(i))

{

printf("The number %d is PRIME ! \n", i);

array[i] = i;

}

else

{

printf("The number %d is NOT PRIME ! \n", i);

}

}

return array;

}

int isPrime(unsigned number)

{

unsigned checker = 2;

if (number == 2)

{

return 1;

}

while (checker <= sqrt(number))

{

if (number%checker == 0)

{

return 0;

}

checker++;

}

return 1;

}

int main()

{

int \*array;

array = numberGenerator();

}

Идеята на Ератостен за търсене на прости числа в интервал (250 преди новата ера).Един очевиден подход за решаване на задачата е да се проверява дали всяко естествено число по-малко от n-тото е просто. Така се извършват n проверки, като за всяко число k ще бъдат най-много на брой проверки за делимост.

При условие, че разполагаме с достатъчно памет, можем да приложим по-бърз метод за намиране на простите числа в интервал, наречен „решето на Ератостен“.Методът носи името на своя създател – Ератостен от Сирен – първият, предположил точно диаметъра на Земята.Известен е още с това, че дълги години е работил в Александрийската библиотека.

Както подсказва името, „решетото“ е метод за програмиране, при който се изключват всички елементи от крайното множество, които не ни интересуват.

Примера е – Записваме числата от 2 до n в редицата: 2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20...n

Намираме първото незачертано и немаркирано число- това е 2.Маркираме го, след което задраскваме всяко второ число след него :

(2),3,~~4,~~5,~~6,~~7,~~8,~~9,~~10,~~11,~~12,~~13,~~14,~~15,~~16,~~17,~~18,~~19,~~20~~...n

По-нататък, отново намираме първото незачертано и немаркирано число: това е числото 3.Маркираме го и задраскваме всички числа в редицата, кратни на 3:

(2),(3),~~4,~~5,~~6,~~7,~~8,9,10,~~11,~~12,~~13,~~14,15,16,~~17,~~18,~~19,~~20~~...n

След това, на ред е числото 5 – маркираме го и задраскваме всяко 5-то :

(2),(3),~~4,~~(5),~~6,~~7,~~8,9,10,~~11,~~12,~~13,~~14,15,16,~~17,~~18,~~19,~~20~~...n

Така всички съставни числа се „отсяват“ и винаги сме сигурни, че минималното незадраскано и немаркирано i е просто. Процесът продължава, докато не остане нито едно незадраскано или немаркирано число – тогава всички маркирани числа са прости, а всички задраскани – съставни.

#include<stdio.h>

#include<stdlib.h>

#include<math.h>

#define MAX 10000

void EratostenAlgorithm(unsigned numberN,char array[])

{

unsigned counter, i = 2;

while (i <= numberN)

{

if (array[i] == 0)

{

printf("%5u", i);

counter = i\*i;

while (counter <= numberN) // на всяка интерация слагаме 1 на мястото на следващото съставно число по индекс

{

array[counter] = 1;

counter += i;

}

}

i++;

}

printf("\n");

}

int main()

{

const unsigned maxNumber = MAX;

char array[MAX+10000];

for (unsigned i = 0; i < maxNumber; i++)

{

array[i] = 0;

}

EratostenAlgorithm(maxNumber,array);

return 0;

}

Обаче съществува и още по-ефективен алгоритъм за търсене на всички прости числа в интервал. При него не е необходима памет (масив array[MAX+10000]) с големина на интервала, както не е необходимо и на всяка стъпка да се обхожда целия интервал.

При модификация ще се започне като в празен списък последователно ще добавяме числа. Например, поставяйки в списъка първото просто число 2 можем да намерим всички прости числа в интервала [3,22] – такова е 3 и го добавяме в списъка. По-нататък , разполагайки вече с простите числа до 3 , можем да намерим всички прости числа в интервала [4,32], това са 5 и 7, които също добавяме в списъка. Продължаваме този процес, докато в array[] се добавят достатъчно прости числа, за да покрият проверката дали всяко число от интервала [2, n] е просто.

#include<stdio.h>

#include<stdlib.h>

#include<math.h>

#define MAX 10000

const unsigned n = 500;

unsigned primes[MAX], pN = 0;

char isPrime(unsigned n)

{

unsigned i = 0;

while (i < pN && primes[i] \* primes[i] <= n)

{

if (n%primes[i] == 0)

{

return 0;

}

i++;

}

return 1;

}

void findPrimes(unsigned n)

{

unsigned i = 2;

while (i < n)

{

if (isPrime(i))

{

primes[pN] = i;

pN++;

printf("%5u", i);

}

i++;

}

}

int main()

{

findPrimes(n);

printf("\n");

return 0;

}

В случаите, когато търсим първите n прости числа, методът на решетото на Ератостен дава много добри резултати.

Интересен факт за простите числа е че част от тях са Мерсенови числа.

Дефиниция за Мерсеново число : Едно просто число се нарича Мерсеново просто ,ако може да се представи във вида 2P-1, където P е просто.

Мерсеновите числа намират редица приложения : за търсене на съвършени числа, при търсенето на много големи прости числа и др.

Все по-често срещано явлени е за сериозни научни или емпирични резултати, да се обявяват големи парични награди. Така например , съществуват общи проекти в Интернет за търсене на следващото голямо просто число (а то с голяма вероятност ще се окаже Мерсеново просто). Подобни проекти са финансирани от известни изследователски организации и участването в такъв проект прилича по-скоро на лотария : късметлията , имал щастието да „уцели“ 38-мото Мерсеново число , е спечелил 50 000 долара. Наградите за следващите такива числа започват от 250 000 долара.

Очевиден алгоритъм за търсене на Мерсенови числа е следният : последователно за всяко просто число p=2,3,5,7,11... проверяваме дали M=2P-1 е просто. Този алгоритъм обаче е крайно неефективен и на практика не дава задоволителни резултати , приложен за големи числа.

Ключът за намирането на такива големи числа е теоремата на Лукас от 1870 година, модифицирана по-късно от Лемер , като за нейно приложение е необходима и бърза програма за умножаване на числа.

Алгоритъм на Лукас-Лемер :

#include<iostream>

#include<math.h>

using namespace std;

\_\_int64 isPrime(int number)

{

int counter1 = 0;

unsigned temp = 2;

if (number == 2)

{

return number;

}

while (temp <= sqrt(number))

{

if (number%temp == 0)

{

return 0;

}

temp++;

}

return number;

}

int main()

{

\_\_int64 array[100];

\_\_int64 arrayPrime[100];

array[0] = 4;

int counter = 0;

for (int i = 1; i < 100; i++)

{

array[i] = (array[i - 1] \* array[i - 1]) - 2;

if (i > 1)

{

if (isPrime(i))

{

arrayPrime[counter++] = isPrime(i);

}

}

}

for (int i = 0; i < 100; i++)

{

if (arrayPrime[i] % 2 != 0)

{

if ((array[arrayPrime[i] - 2] % ((\_\_int64)pow((double)2, (double)arrayPrime[i])-1) == 0))

{

cout << "The number : " << ((\_\_int64)pow((double)2, (double)arrayPrime[i])-1) << " is PRIME !" << endl;

}

}

}

return 0;

}