**Student :** Ionasco Gheorghe

**Group :** FAF-193

Chișinău 2021

Simon Plouffe explained in [[1](https://bellard.org/pi/pi_n2/pi_n2.html#plo96)] a new algorithm to compute the n'th digit of and some other mathematical constants in any base with very little memory. Its running time is . We present here an improvement of this algorithm whose running time is while its memory requirements stay , which makes it practical to compute the millionth digit of for example.

**Application**

Given



we can use the result of section [2](https://bellard.org/pi/pi_n2/pi_n2.html#res2) because if **p** is a prime number, we notice that



where is the multiplicity of **p** in **n**. It comes from the relation



Hence, if we want the n'th digit of in base B, we may use the following algorithm:

* where is a small integer to ensure we have the precision needed ; .
* For each prime number **a** with **2 < a < 2N**, do:
  + ; .
  + ; ; ; .
  + for **k** in do:
    - ; ; .
    - if do: .
  + ; .
* If we suppose that , then, if we neglect rounding errors, . The number **q** of correct digits depends on .

The running time is because there are prime numbers between **2** and **2n** . The memory requirements are, as expected, in .

**Program C++**

#include <stdlib.h>

#include <stdio.h>

#include <math.h>

#include <time.h>

#ifdef HAS\_LONG\_LONG

#define mul\_mod(a,b,m) (( (long long) (a) \* (long long) (b) ) % (m))

#else

#define mul\_mod(a,b,m) fmod( (double) a \* (double) b, m)

#endif

int inv\_mod(int x, int y)

{

int q, u, v, a, c, t;

u = x;

v = y;

c = 1;

a = 0;

do {

q = v / u;

t = c;

c = a - q \* c;

a = t;

t = u;

u = v - q \* u;

v = t;

} while (u != 0);

a = a % y;

if (a < 0)

a = y + a;

return a;

}

int pow\_mod(int a, int b, int m)

{

int r, aa;

r = 1;

aa = a;

while (1) {

if (b & 1)

r = mul\_mod(r, aa, m);

b = b >> 1;

if (b == 0)

break;

aa = mul\_mod(aa, aa, m);

}

return r;

}

int is\_prime(int n)

{

int r, i;

if ((n % 2) == 0)

return 0;

r = (int) (sqrt(n));

for (i = 3; i <= r; i += 2)

if ((n % i) == 0)

return 0;

return 1;

}

int next\_prime(int n)

{

do {

n++;

} while (!is\_prime(n));

return n;

}

int main(int argc, char \*argv[])

{

clock\_t tStart = clock();

int av, a, vmax, N, n, num, den, k, kq, kq2, t, v, s, i;

double sum;

if (argc < 2 || (n = atoi(argv[1])) <= 0) {

printf("This program computes the n'th decimal digit of \\pi\n"

"usage: pi n , where n is the digit you want\n");

exit(1);

}

N = (int) ((n + 20) \* log(10) / log(2));

sum = 0;

for (a = 3; a <= (2 \* N); a = next\_prime(a)) {

vmax = (int) (log(2 \* N) / log(a));

av = 1;

for (i = 0; i < vmax; i++)

av = av \* a;

s = 0;

num = 1;

den = 1;

v = 0;

kq = 1;

kq2 = 1;

for (k = 1; k <= N; k++) {

t = k;

if (kq >= a) {

do {

t = t / a;

v--;

} while ((t % a) == 0);

kq = 0;

}

kq++;

num = mul\_mod(num, t, av);

t = (2 \* k - 1);

if (kq2 >= a) {

if (kq2 == a) {

do {

t = t / a;

v++;

} while ((t % a) == 0);

}

kq2 -= a;

}

den = mul\_mod(den, t, av);

kq2 += 2;

if (v > 0) {

t = inv\_mod(den, av);

t = mul\_mod(t, num, av);

t = mul\_mod(t, k, av);

for (i = v; i < vmax; i++)

t = mul\_mod(t, a, av);

s += t;

if (s >= av)

s -= av;

}

}

t = pow\_mod(10, n - 1, av);

s = mul\_mod(s, t, av);

sum = fmod(sum + (double) s / (double) av, 1.0);

}

printf("Decimal digits of pi at position %d: %09d\n", n,

(int) (sum \* 1e9));

printf("Time taken: %.9fs\n", (double)(clock() - tStart)/CLOCKS\_PER\_SEC);

return 0;

}

# Results:

# 

**Conclusion**

We have presented an algorithm to compute the n'th digit in any base B of whose running time is . It has the same running time as other classical methods for computing (e.g. arctangent formulas), but it uses little memory, it is very simple and does not need high precision computations. It is still slower than the BBP algorithm [[2](https://bellard.org/pi/pi_n2/pi_n2.html#bbp95)], but it works in any base. As described in [[1](https://bellard.org/pi/pi_n2/pi_n2.html#plo96)], the same algorithm may be used to compute other numbers such as , , , and .