Министерство образования и науки Российской Федерации Федеральное агентство по образованию

Федеральное государственное бюджетное образовательное учреждение высшего образования

«Вятский государственный университет»

Факультет автоматики и вычислительной техники

Кафедра электронных вычислительных машин

**Применение С-кода в языке программирования Python**

Лабораторная работа №9

по курсу «Разработка программных систем»

Выполнил студент группы ИВТб-31 /Категов А. Д./ Проверил преподаватель /Чистяков Г.А./

Киров 2024

1. Цель

Целью работы является получение навыков повышения производительности приложений на Python за счет реализации вычислительно сложных операций в модулях на языке С/C++.

1. Задание

Для выполнения лабораторной работы необходимо решить следующие задачи:

* По результатам лабораторной работы №1 выбрать один из наиболее требовательных методов
* Реализовать данный метод на языке С/C++
* Оформить реализацию в виде подключаемого модуля
* Провести сравнения реализаций метода на Python и на C/C++ для набора подготовленных данных.

1. Листинг программы

Листинг программной реализации приведен в приложении А.

1. Результаты тестов

Для сравнения реализаций был выбран метод факторизации чисел. В качестве входных данных использовались массивы чисел.

Результаты приведены в таблице 1.

Таблица 1 – Результаты тестирования

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| № | Объем входных данных | Python, c | С++, с | Ускорение |
| 1 | 10 | 0,023 | 0,001 | 23,00 |
| 2 | 50 | 6,123 | 0,332 | 18,44 |
| 3 | 100 | 11,927 | 0,658 | 18,13 |
| 4 | 150 | 32,598 | 1,750 | 18,63 |
| 5 | 200 | 39,498 | 2,162 | 18,27 |

**Вывод**: в ходе выполнения лабораторной работы были изучены способы повышения производительности программ на Python за счет реализации вычислительно сложных операций в модулях на языке С/C++. В процессе работы был разработан подключаемый модуль на языке C++ выполняющий факторизацию целых чисел. Выполнено сравнение производительности вычислений алгоритма, реализованного на python и алгоритма в подключаемом C++ модуле. По результатам время выполнения алгоритма, реализованного на C++ значительно ниже по сравнению с реализацией на Python. На основании всего сказанного выше можно сделать вывод о том, что цель достигнута, необходимые знания получены, задание выполнено верно, что доказывается полученными результатами.

Приложение А

(обязательное)

Листинг программы

**main.py**

from time import time

from fast\_factorize import primeFactorize

from prime import \*

def speed\_test(size):

    print(f'Starting a {size} test:')

    data = generate\_primes(size)

    start = time()

    for v in data:

        prime\_factorize(v)

    duration = time() - start

    print(f"[Python]: {time() - start}")

    start = time()

    for v in data:

        primeFactorize(v)

    duration = time() - start

    print(f"[C++]: {time() - start}")

def main():

    speed\_test(10)

    speed\_test(50)

    speed\_test(100)

    speed\_test(150)

    speed\_test(200)

if \_\_name\_\_ == "\_\_main\_\_":

    main()

**prime.py**

import random

MAX\_PRIME = 999999937

def is\_prime(n): # optimized trial division with 6k optimization

    if n <= 1: # numbers less than or equal to 1 are not prime

        return False

    if n <= 3:                   # 2 and 3 are prime

        return True

    if n % 2 == 0 or n % 3 == 0: # exclude even numbers and numbers divisible by 3

        return False

    if n > MAX\_PRIME:

        raise Exception('obtained a number exceeding the MAX\_PRIME')

    i = 5 # check the divisors from 5 to sqrt(n) in increments of 6

    while i \* i <= n:

        if n % i == 0 or n % (i + 2) == 0:

            return False

        i += 6

    return True

def prime\_factorize(num, k=2):

    factors = []

    while num > 1:

        if is\_prime(num):

            factors.append(num)

            break

        elif num % k == 0:

            factors.append(k)

            num //= k

        else:

            k += 1

    return factors

get\_random\_number = lambda: random.randint(2, MAX\_PRIME - 1)

def generate\_primes(size): # generating an array of numbers

    primes = []

    while len(primes) < size:

        prime = get\_random\_number()

        primes.append(prime)

    return primes

**fast\_factorize.cpp**

#include "pch.h"

#include <Windows.h>

#include <pybind11/pybind11.h>

#include <pybind11/stl.h>

#include <vector>

const long long MAX\_PRIME = 999999937;

using namespace std;

namespace py = pybind11;

bool isPrime(long long n) { //optimized trial division with 6k optimization

if (n <= 1) { //numbers less than or equal to 1 are not prime

return false;

}

if (n <= 3) { //2 and 3 are prime

return true;

}

if (n % 2 == 0 || n % 3 == 0) { //exclude even numbers and numbers divisible by 3

return false;

}

if (n > MAX\_PRIME) {

throw std::runtime\_error("obtained a number exceeding the MAX\_PRIME");

}

long long i = 5; //check the divisors from 5 to sqrt(n) in increments of 6

while (i \* i <= n) {

if (n % i == 0 || n % (i + 2) == 0) {

return false;

}

i += 6;

}

return true;

}

vector<long long> primeFactorize(long long num) {

vector<long long> factors;

long long k = 2;

while (num > 1) {

if (isPrime(num)) {

factors.push\_back(num);

break;

}

else if (num % k == 0) {

factors.push\_back(k);

num /= k;

}

else {

k += 1;

}

}

return factors;

}

PYBIND11\_MODULE(fast\_factorize, m) {

m.def("primeFactorize", &primeFactorize, "Factorize function");

#ifdef VERSION\_INFO

m.attr("\_\_version\_\_") = VERSION\_INFO;

#else

m.attr("\_\_version\_\_") = "dev";

#endif

}