Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение высшего образования
«Московский государственный технический университет имени Н.Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ «Информатика и системы управления»
КАФЕДРА «Программное обеспечение ЭВМ и информационные технологии»

Отчет по по рубежному контролю по курсу "Анализ алгоритмов"

Тема Эффективный алгоритм нахождения N первых простых чисел
Студент Топорков Павел
Группа ИУ7-53Б
Дисциплина Анализ алгоритмов

Преподаватели: _____ Строганов Ю.В., Волкова Л.Л.
подпись, дата Фамилия, И.О.

Оценка _____

Москва — 2021 г.

Оглавление

Введение

В современной теории алгоритмов существует ряд задач, получение точного ответа в которых возможно только полным перебором всевозможных вариантов множества решений (т.н. NP-полные задачи). Однако зачастую в подобных задачах оптимизации допустимо получение ответа, приближенного к идеальному. В этих целях используются алгоритмы, работающие за приемлемое полиномиальное время, такие как генетические и муравьиные алгоритмы.

Муравьиный алгоритм — один из эффективных полиномиальных алгоритмов для нахождения приближенных решений задачи коммивояжёра, а также решения аналогичных задач поиска маршрутов на графах.

В ходе лабораторной работы предстоит:

- рассмотреть муравьиный алгоритм и алгоритм полного перебора в задаче коммивояжера;
- реализовать алгоритмы;
- сравнить временную эффективность алгоритмов.

1 Аналитическая часть

В данном разделе будут описаны алгоритмы поиска первых N простых чисел

1.1 Простой алгоритм

Простой перебор всех чисел с проверкой на простоту числа, пока не найдем первые нужные N

1.2 Улучшенный алгоритм

Во-первых, проверять четные числа не имеет смысла, ведь 2 — единственное простое четное число. То же самое с делителями — на четные можно не проверять. В-третьих, в качестве делителей достаточно брать числа не превышающие половину проверяемого числа.

1.3 Улучшенный алгоритм с использование решета Эратосфена

Наиболее идейно простым алгоритмом решения задачи коммивояжера [1] является полный перебор решений с выбором кратчайшего из полученных путей. Очевидным недостатком данного алгоритма является необходимость перебора значительного числа комбинаций, которое с ростом числа городов быстро выходит за рамки вычислительных мощностей современных компьютеров. Трудоёмкость алгоритма полного перебора – $O(n!)$.

2 Технологическая часть

В данном разделе приведены требования к программному обеспечению, средства реализации и листинги кода.

2.1 Средства реализации

Для реализации был выбран язык c++, так как он хорошо подходит для измерения скорости и в нем есть все необходимые инструменты.

2.2 Листинг кода

В листингах 2.1 – 2.3 приведены реализации алгоритмов.

Листинг 2.1 – Простой алгоритм

```
int main() {
    const int AIM = 100000;
    vector<int> prime_nums;
    prime_nums.push_back(2);
    for (int num = 3; prime_nums.size() < AIM; num += 1) {
        bool isprime = true;
        for (int i = 3; i < num; i += 1) {
            if (num % i == 0) {
                isprime = false;
                break;
            }
        }
        if (isprime)
            prime_nums.push_back(num);
    }
}
```

Листинг 2.2 – Улчушенный алгоритм

```
int main() {
    const int AIM = 100000;
    vector<int> prime_nums;
    prime_nums.push_back(2);
```

```
    for (int num = 3; prime_nums.size() < AIM; num += 2) {
        bool isprime = true;
        for (int i = 3; i <= num / 2; i += 2) {
            if (num % i == 0) {
                isprime = false;
                break;
            }
        }
        if (isprime)
            prime_nums.push_back(num);
    }
}
```

Листинг 2.3 – Улчушенный алгоритм

```
int main() {
    const int AIM = 1000000;

    int startSize = AIM;
    int addSize = AIM;
    vector<bool> nums(startSize);
    vector<int> primeNums(AIM);

    int foundPrimes = 0;
    for (int i = 2; i < startSize; i++)
        nums[i] = true;

    bool addition = false;
    int adder = 0;
    while (true) {
        if (addition) {
            nums.resize(nums.size() + addSize, true);

            for (int i = 0; i < foundPrimes; i++) {
                int cur_num = primeNums[i];
                if ((addSize + ((nums.size() - addSize) %
                   cur_num)) < cur_num)
                    continue;
                for (int j = ((nums.size() - addSize) / cur_num
                   ) * cur_num; j < nums.size(); j += cur_num)
                    nums[j] = false;
            }
```

```
26          }
27          else
28              addition = true;
29
30          int iter;
31          if (foundPrimes == 0)
32              iter = 2;
33          else
34              iter = primeNums[foundPrimes - 1] + 2;
35
36          for ( ; iter < nums.size(); iter++) {
37              if (nums[iter]) {
38                  primeNums[foundPrimes] = iter;
39                  foundPrimes++;
40                  if (foundPrimes == AIM)
41                      break;
42                  for (int j = iter + iter; j < nums.size(); j +=
                        iter)
43                      nums[j] = false;
44              }
45              else
46                  continue;
47          }
48          if (foundPrimes == AIM)
49              break;
50      }
51  }
```

3 Иследовательская часть

Результат 1 алгоритма

10 000 — 0,589 с.

100 000 — 72,662 с.

1 000 000 — 426,208 с.

Результат 2 алгоритма

10 000 — 0,356 с.

100 000 — 46,436 с.

1 000 000 — 238,354 с.

Результат 3 алгоритма

10 000 — 0,001 с.

100 000 — 0,021 с.

1 000 000 — 0,262 с.

Заключение

В ходе лабораторной работы были изучены и реализованы алгоритмы решения задачи нахождения простых чисел.

Были получены результаты работы.

Литература

[1] Perl problems - Commis Voyageur [Электронный ресурс]. Режим доступа: http://mech.math.msu.su/~shvetz/54/inf/perl-problems/chCommisVoyageur.xhtml (дата обращения: 12.12.2020).