Московский авиационный институт
(национальный исследовательский университет)

Факультет информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

Лабораторная работа №8 по курсу «Дискретный анализ»

Студент: В. С. Епанешников
Преподаватель: А. А. Кухтичев
Группа: М8О-206Б
Дата:
Оценка:
Подпись:

Москва, 2021

Лабораторная работа №8

Задача: Разработать жадный алгоритм решения задачи, определяемой своим вариантом. Доказать его корректность, оценить скорость и объём затрачиваемой оперативной памяти.

Реализовать программу на языке C или C++, соответсвующую построенному алгоритму. Формат входных и выходных данных описан в варианте задания.

На первой строке заданы два числа, N и p > 1, определяющие набор монет некоторой страны с номиналами p0, p1, ..., p(N-1). Нужно определить наименьшее количество монет, которое можно использовать для того, чтобы разменять заданную на второй строчке сумму денег M не больше $2^{32} - 1$ и распечатать для каждого i-го номинала на i-ой строчке количество участвующих в размене монет. Кроме того, нужно обосновать почему жадный выбор неприменим в общем случае (когда номиналы могут быть любыми) и предложить алгоритм, работающий при любых входных данных.

Формат входных данных

На первой строке заданы два числа N и p, на второй строке сумма денег M.

Формат результата

Для каждого i-го номинала на i-ой строчке вывести количество участвующих в размене монет.

1 Описание

Жадные алгоритмы - алгоритмы, предполагающие принятие локально оптимальных решений на каждом этапе, допуская, что конечное решение также окажется оптимальным. В общем случае жадные алгоритмы могут не находить глобального оптимума, однако в некоторых задачах позволяют это сделать.

Для моей задачи алгоритм выглядит так:

Вычитаем из суммы денег M наибольший номинал пока это возможно (пока число положительное). Таким образом, мы получаем оптимальное решение на каждом этапе, ведь чем больше номинал, тем меньше остаётся на размен.

2 Исходный код

```cpp
#include <iostream>
#include <vector>
#include <algorithm>

int main() {
  long long N, p, M;
  std::cin >> N >> p >> M;
  std::vector<long long> nominals;
  std::vector<long long> result(N, 0);

  for (int i = N - 1; i >= 0; --i) {
    long long currCoin = pow(p, i);
    if (M < currCoin) {
      continue;
    }

    result[i] = M / currCoin;
    M -= result[i] * currCoin;
    if (M == 0) {
      break;
    }
  }

  for (int i = 0; i < N; ++i) {
    std::cout << result[i] << "\n";
  }

  return 0;
}
```

3 Консоль

```
MacBook:exercise_08 vladislove$ ./a.out
3 5
71
1
4
2
MacBook:exercise_08 vladislove$ ./a.out
5 3
577
1
0
1
0
7
```

4 Тест производительности

Для тестов я использвал утилиту gnuplot для построения графиков зависимости времени работы программы от количества чисел. Так же для сравнения использовал библиотеку chrono для замера времени.

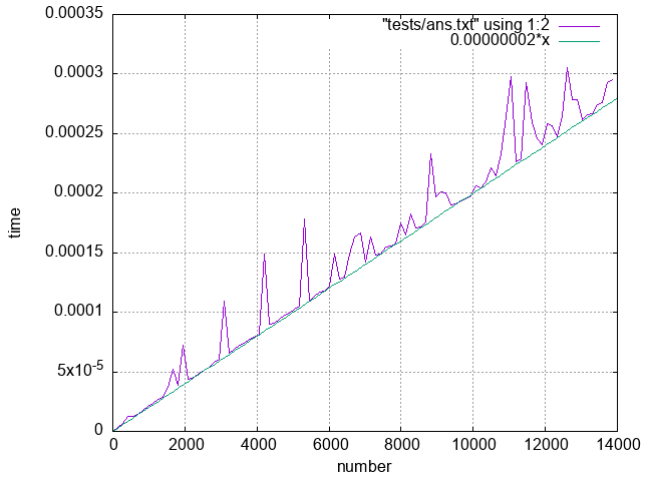

Рис. 1: Графики работы жадного алгоритма и сопоставление с линейной функцией

Как видно, сложность алгоритма линейная.

5 Выводы

В ходе восьмой лабораторной работы я познакомился с жадными алгоритмами. В процессе выполнения задания особых проблем не было.

Список литературы

[1] *Сайт с подробной документацией библиотек C++*
URL: https://en.cppreference.com/

[2] *Gnuplot и с чем его едят*
URL: https://habr.com/ru/company/ruvds/blog/517450/

[3] *ЖАДНЫЙ ПОДХОД ПРОТИВ ДИНАМИЧЕСКОГО ПРОГРАММИРОВАНИЯ*
URL: http://espressocode.top/greedy-approach-vs-dynamic-programming/