Московский авиационный институт
(национальный исследовательский университет)

Факультет компьютерных наук и прикладной математики

Кафедра вычислительной математики и программирования

Лабораторная работа №8 по курсу «Дискретный анализ»

Студент: О. А. Мезенин
Преподаватель: А. А. Кухтичев
Группа: М8О-306Б-21
Дата:
Оценка:
Подпись:

Москва, 2023

Лабораторная работа №8

Задача: На первой строке заданы два числа, N и $p > 1$, определяющие набор монет некоторой страны с номиналами $p^0, p^1, ..., p^{N-1}$. Нужно определить наименьшее количество монет, которое можно использовать для того, чтобы разменять заданную на второй строчке сумму денег $M \leq 2^{32} - 1$ 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.

1 Описание

Воспользуемся жадным алгоритмом. Пусть P – список из отсортированных номиналов. Тогда будем брать сначала максимальное количество монет самого большого номинала $P[i]$, затем максимальное количество монет номинала $P[i-1]$ и так далее.

Жадный алгоритм работает тогда, когда каждый следующий номинал делится на предыдущий [1]. Приведём пример, когда жадный алгоритм не работает. Предположим, у нас номинал $2, 9, 10$, а сумма денег 18. Следуя жадному алгоритму, мы выберем одну монету номинала 10 и четыре монеты номинала 2, хотя оптимальный ответ – две монеты номинала 9. Для любых входных данных можно использовать алгоритм, использующий динамическое программирование. Пусть $dp[i]$ – минимальное количество монет, из которых можно составить сумму i, используя доступный номинал. Изначально $dp[i] = 1$, если $i \in P$, иначе $dp[i] = inf$. Тогда $dp[i] = min(dp[i], dp[i - P[1]] + 1, dp[i - P[2] + 1, ..., dp[i - P[m - 1]] + 1)$. Ответом будет $dp[m]$.

2 Исходный код

Функция `main` считывает входные данные, вызывает функцию `coinExchange` и выводит ответ. Логика функции `coinExchange` очень простая: начинаем идти с самого высокого номинала и в ответ `answer[i]` записываем $m/nominal[i]$, где m – оставшееся сумма денег.

```
#include <iostream>
#include <vector>
#include <cmath>

std::vector<uint32_t> coinExchange(std::vector<uint64_t>& nominal, uint32_t m) {
    std::vector<uint32_t> answer(nominal.size());
    for (size_t i = nominal.size() - 1; m > 0; --i) {
        answer[i] = m / nominal[i];
        m -= answer[i] * nominal[i];
    }
    return answer;
}


int main() {
    std::ios_base::sync_with_stdio(false);
    std::cin.tie(nullptr);
    uint64_t n, p;
    uint32_t m;
    std::cin >> n >> p >> m;
    std::vector<uint64_t> nominal(n);
    for (size_t i = 0; i < n; ++i) {
        nominal[i] = std::pow(p, i);
    }
    std::vector<uint32_t> nominalNumbers = coinExchange(nominal, m);
    for (auto nom: nominalNumbers) {
        std::cout << nom << '\n';
    }
    return 0;
}
```

3 Консоль

```
aprold@SAI:~/Documents/GitHub/MAI-DA/lab8$ make lab8
g++ -std=c++2a -pedantic -Wall -Wextra -Werror main.cpp coin_exchange.cpp -o
lab8.out
aprold@SAI:~/Documents/GitHub/MAI-DA/lab8$ ./lab8.out
3 5
71
1
4
2
aprold@SAI:~/Documents/GitHub/MAI-DA/lab8$ ./lab8.out
4 65
1073741824
64
4
55
3909
```

4 Тест производительности

Тесты производительности представляют из себя следующее: «жадный» алгоритм будет сравниваться с алгоритмом, основанном на методе динамического программирования. Будет два теста:

1. $n = 5, p = 3, m = 2^{13}$
2. $n = 5, p = 5, m = 2^{28}$

```
aprold@SAI:~/Documents/GitHub/MAI-DA/lab8$ make benchmark
g++ -std=c++2a -pedantic -Wall -Wextra -Werror benchmark.cpp coin_exchange.cpp
-o benchmark.out
aprold@SAI:~/Documents/GitHub/MAI-DA/lab8$ ./benchmark.out <benchmark_tests/01.t
greedy time: 1us
DP time: 1362us
aprold@SAI:~/Documents/GitHub/MAI-DA/lab8$ ./benchmark.out <benchmark_tests/02.t
greedy time: 2us
DP time: 16586025us
```

«Жадный» алгоритм работает за $O(n)$, а «ДП» за $O(m * n)$. «Жадный» алгоритм оказался быстрее в обоих случаях.

5 Выводы

Выполнив восьмую лабораторную работу по курсу «Дискретный анализ», узнал про жадный алгоритм и о задачах, где он применяется.
Если говорить о задаче «Размен монет», то жадный алгоритм применим не всегда, зато работает очень быстро ($O(n)$, где n – количество номинала). В общем случае в этой задаче можно использовать метод динамического программирования, но его время работы составит $O(n*m)$, где m – количество денег, которое нужно составить.

Список литературы

[1] *Жадный алгоритм*
URL: `https://algorithmica.org/tg/greedy` (дата обращения: 05.11.2023).