Московский авиационный институт (национальный исследовательский университет)

Институт №8 «Компьютерные науки и прикладная математика»

Кафедра 806 «Вычислительная математика и программирование»

Лабораторная работа по курсу "Дискретный анализ" №8

Студент: Клименко В. М.

Преподаватель: Макаров Н. К.

Группа: М8О-203Б-22

Дата: ___________

Оценка: ___________

Подпись: ___________

Содержание

Постановка задачи

Нужно определить наименьшее количество монет, которое можно использовать для того, чтобы разменять заданную сумму денег.

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 ввода

На первой строке заданы два числа, N и $p > 1$, определяющие набор монет некоторой страны с номиналами p_0, p_1, ..., p_{N-1}. На второй строке сумма денег $M \leq 2^{32} - 1$.

Формат вывода

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

Алгоритм решения

Жадный алгоритм

Пока сумма денег не равна нулю, уменьшаем ее на номинал наибольшей возможной монеты, при этом сохраняем количество взятых монет определенного номинала. Временная сложность – $O(M)$, пространственная сложность – $O(N)$.

К сожалению этот алгоритм не работает если в размене участвуют монеты любого номинала. При данных входных данных о номиналах монет, сумма $p_0 + ... + p_j + 1 = p_{j+1}$, значит при значениях заданной суммы больших p_{j+1} нужно всегда брать монеты номиналом p_{j+1}

Динамическое программирование

Будем решать задачу снизу-вверх.

Заведем массив длиной $M + 1$, в котором i-ый элемент – это пара состоящая из количества монет для размена суммы i и индекса последней взятой монеты. Тогда, на i-ом шаге рассчитываем значение $i + p_j$ для $\forall j \in 0..N$, проверяем меньше ли оно по количеству монет чем записанное, если да – сохраняем новое количество монет и индекс выбранной монеты j.

Тогда, в последнем элементе массива будет лежать пара значений – минимальное количество монет, нужное для размена начальной суммы, во втором – индекс последней взятой монеты. С помощью индекса монеты мы можем посчитать сколько всего монет определенного номинала мы использовали для размена данной суммы.

Временная сложность – $O(NM)$, пространственная сложность – $O(M)$.

Исходный код

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



template <typename T>
T power(const T base, const T exponent) {
      T result = 1;
      for (T _ = 1; _ < exponent; ++_) {
          result *= base;
      }
      return result;
}



std::vector<uint32_t> dynamic(const std::vector<uint64_t> &coins, const
uint32_t amount) {
      const size_t coinsSize = coins.size();

      // vector of pairs: count of coins and the last added coin for current
amount
      std::vector<std::pair<uint32_t, uint64_t>> dp(amount + 1,
std::pair<uint32_t, uint64_t>(amount, 0));
      dp[0].first = 0;
```

```
        for (uint32_t currentAmount = 0; currentAmount <= amount;
++currentAmount) {
            for (size_t currentCoinIndex = 0; currentCoinIndex < coinsSize;
++currentCoinIndex) {
                const uint64_t currentCoin = coins[currentCoinIndex];

                if (currentAmount + currentCoin > amount) continue;

                if (dp[currentAmount].first + 1 < dp[currentAmount +
currentCoin].first) {
                    dp[currentAmount + currentCoin].first =
dp[currentAmount].first + 1;
                    dp[currentAmount + currentCoin].second =
currentCoinIndex;
                }
            }
        }

        std::vector<uint32_t> result(coinsSize, 0);

        size_t currentAmount = amount;
        while (currentAmount != 0) {
        const size_t currentCoinIndex = dp[currentAmount].second;
        const uint64_t currentCoin = coins[currentCoinIndex];
        result[currentCoinIndex]++;
        currentAmount -= currentCoin;
    }

    return result;
}


std::vector<uint32_t> greedy(const std::vector<uint64_t> &coins, uint32_t
amount) {
    const size_t coinsSize = coins.size();
    size_t currentCoinIndex = coinsSize - 1;

    std::vector<uint32_t> result(coinsSize, 0);

    while (amount != 0) {
        uint64_t currentCoin = coins[currentCoinIndex];
        while (amount < currentCoin) {
            currentCoin = coins[--currentCoinIndex];
        }
```

```
            result[currentCoinIndex] = amount / currentCoin;
            amount = amount % currentCoin;
        }

        return result;
}



int main() {
        uint64_t exponent, base;
        uint32_t amount;

        std::cin >> exponent >> base >> amount;

        // populate coins
        std::vector<std::uint64_t> coins(exponent);
        for (size_t currentExponent = 1; currentExponent <= exponent;
++currentExponent) {
            coins[currentExponent - 1] = power(base, currentExponent);
        }

        // std::vector<uint32_t> result = dynamic(coins, amount);
        std::vector<uint32_t> result = greedy(coins, amount);

        for (uint32_t count : result) {
            std::cout << count << '\n';
        }

        return 0;
}
```

Тесты

Входные данные:

```
12 13
2817172897
```

Выходные данные:

```
6
1
```

```
9
0
6
8
11
5
3
0
0
0
```

Входные данные:

```
3 13
923849234
```

Выходные данные:

```
9
6
5466563
```

Входные данные:

```
1 1
255
```

Выходные данные:

```
255
```

Вывод

В ходе лабораторной работы я научился решать задачи при помощи жадных алгоритмов и доказывать, почему их можно/нельзя использовать при разных условиях задачи.