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: М. А. Волков
Преподаватель: А. А. Кухтичев
Группа: М8О-207Б-19
Дата: 12 мая 2021 г.
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 $p_0, p_1, \ldots, p_{N-1}$. 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, нжуно обосновать почему жадрный выбор неприменим в общем случае (когда номиналы могут быть любыми) и предложить алгоритм, работающий при любых входных данных.

1 Описание

Посмотрев на условие задачи, можно понять, что нужно всего лишь данное на второй строчке число разложить на сумму степеней одного основания. Очевидно, что это возможно сделать для любого целого числа и любого натурального основания.

Данный алгоритм демонстрирует идею жадного алгоритма: мы не смотрим все возможные способы размена монет, а выбираем каждый раз монету с наибольшим номиналом. Очевидно, что таким образом мы получим наименьшее количество монет.

Предоставленный выше алгоритм не работает для общего случая, когда номинал монеты может быть любым, потому что сразу ломается теорема о разложении любого числа на сумму степеней одинакого натурального основания. Вот пример:

Предположим у меня есть монеты 4, 6 и 8 номиналов. Я никогда не смогу выдать этими монетами сумму в 7 условных единиц валюты.

Чтобы решить данную задачу, или хотя бы узнать возможно ли разложить заданное число на заданные монеты, нужно прибегнуть к динамическому программированию. Суть алгоритма заключается в следующем:

Имеется массив, условно называнный dp. В i-ой ячейке данного массива будет хранится число - минимальное количество монет, нежное для размена суммы i.

Начальное значение:

$dp[0] = 0$ - потому что сумму, равную 0 можно выдать 0-ми монет. Все остальные значения выберем равные inf.

Динамический переход:

$dp[i] = min\{dp[i-c_j]|$по всем j от 0 до N$\}$ Ответ далее собирается путем запоминания пути, откуда был достигнут минимум.

Посчитаем сложность жадного алгоритма: нам нужно подготовить массив с предподстчитанными значениями степеней основания, для удобного вычисления. Это делается за $O(N)$. Далее нужно посчитать количество монет. Это делается одним проходом справа налево ранее подготовленного массива и деления данного нам числа на число элементов массива. Это $O(log_p(M))$.

Итоговая сложность: $O(N + log_p(M)) = O(N)$.

Использования доп памяти: $O(N)$.

Посчитаем сложность алгоритма для общей задачи: нам нужно будет пройтись по массиву длины M и при этом посчитать его для каждого номинала N. Итоговая сложность: $O(N * M)$.

Использование доп памяти: $O(M)$.

2 Исходный код

Код достаточно простой. Единственная сложность была подгадать момент переполнения числа. Благо ограничение памяти мне позволяет взять наиболее возможное значение типа – long long.

Для бесконечно больших чисел, уже придется прибегнуть к длинной арифметике.

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

int main(){
  long long N,p;
  std::cin >> N >> p;
  long long M;
  std::cin >> M;
  std::vector<unsigned long long> pows(N,1);
  long long count = (int) N-1;
  for (int i = 1; i < N; ++i) {
    pows[i] = pows[i-1] * p;

    if(pows[i] > M){
      count = i;
      break;
    }
  }

  std::vector<unsigned long long> ans(N);

  while(M > 0){
    ans[count] = M/pows[count];
    M %= (int) pows[count];
    count--;
  }

  for (auto& item : ans){
    std::cout << item << std::endl;
  }
  return 0;
}
```

3 Тест производительности

В тесте производительности сравним жадный алгоритм с ДП, предложенный в описании.

Тесты:

1. $M = 2^{18} - 1; p = 2; N = 18$
2. $M = 2^{24} - 1; p = 2; N = 32$

ДП:

```
1. DP time 0.32 ms
2. DP time 2.97 ms
```

Жадный алгоритм:

```
1. Greevy time 0.0 ms
2. Greevy time 0.0 ms
```

Как видно жадный алгоритм дает мгновенный ответ, когда как ДП еще думает. Ранее 2-ой тест должен быть при $M = 2^{32} - 1$, алгоритм с ДП выполнялся очень долго.

4 Выводы

В результате выполнения лабораторной работы я изучил основные алгортмы, использующие идею жадных алгоритмов, составил и отладил программу для своего варианта задания.

В отличие от динамического программирования жадные алгоритмы предполагают, что задача имеет оптимальное решение, которое строится из оптимальных решений для подзадач с заранее определённым выбором, а не перебором всех вариантов перехода. Такой подход уменьшает временные и пространственные ресурсы, нужные для решения задачи.

К большому сожелению данный метод программирования применим к очень узкому спектру задач и по большей части они все достаточно очевидные.

Список литературы

[1] Томас Х. Кормен, Чарльз И. Лейзерсон, Рональд Л. Ривест, Клиффорд Штайн. *Алгоритмы: построение и анализ, 2-е издание.* — Издательский дом «Вильямс», 2007. Перевод с английского: И. В. Красиков, Н. А. Орехова, В. Н. Романов. — 1296 с. (ISBN 5-8459-0857-4 (рус.))