Лабораторная работа № 8 по курсу дискретного анализа: «Жадные алгоритмы»

Выполнил студент группы М8О-307Б-21 МАИ Дубровин Дмитрий

Условие:

Вариант: 1 -> Размен монет

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 $M \leqslant 2^{32} - 1$ 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.

Пример

Ввод	Вывод
3 5	1
71	4
	2

Метод решения

Вот как работает этот алгоритм:

1. Считываются параметры nomin и money, которые определяют структуру номиналов монет.
2. Считывается сумма amount, которую необходимо разменять.
3. Создается вектор kol_money для хранения количества монет каждого номинала.
4. Запускается цикл while, который выполняется до тех пор, пока amount не станет равной 0.
5. Внутри цикла проверяется, можно ли вычесть максимальную степень money, чтобы уменьшить amount. Если это возможно, то вычитается максимальная степень money, и увеличивается соответствующий элемент kol_money.
6. Если нельзя вычесть максимальную степень money без превышения amount, уменьшается значение nomin, чтобы перейти к следующему номиналу монет.
7. Цикл продолжает выполняться, пока amount не станет равной 0.
8. Затем программа выводит количество монет каждого номинала на экран.

Этот код пытается разменять сумму amount на монеты, используя наибольшие номиналы в соответствии с заданными параметрами nomin и money. (Жадный алгоритм) Но этот подход работает в случае, если номиналы монет образуют "жадный" набор, то есть можно разменять любую сумму, комбинируя их. Однако, если номиналы не образуют жадный набор или есть другие ограничения, то этот алгоритм может не давать оптимального результата.

Для примера можно привести такой набор номиналов: 2 и 5, а «собрать» из них необходимо 8, при таком наборе входных данных жадный алгоритм не сможет выдать правильный ответ. При таких входных данных можно использовать динамическое программирование, вот общий алгоритм решения этой задачи с использованием динамического программирования:

1. Создайте массив dp, где dp[i] будет хранить минимальное количество монет для размена суммы i.
2. Инициализируйте dp[0] как 0, так как минимальное количество монет для размена 0 всегда равно 0.
3. Для каждой суммы от 1 до amount, вычислите минимальное количество монет, необходимых для размена этой суммы, используя имеющиеся номиналы монет. Для этого пройдитесь по всем номиналам монет и выберите тот, который минимизирует количество монет для текущей суммы.
4. Заполните массив dp последовательно для всех сумм от 1 до amount.
5. После завершения заполнения массива dp, вы сможете найти минимальное количество монет для размена суммы amount в dp[amount].
6. Для восстановления набора монет, используемых для размена суммы amount, начните с dp[amount] и двигайтесь назад, выбирая номиналы монет, которые минимизируют количество монет для текущей суммы.

Тест производительности

Сначала проанализирую сложность работы исходной программы:

1. Считывание nomin и money из стандартного ввода имеет временную сложность O(1).
2. Считывание amount из стандартного ввода также имеет временную сложность O(1).
3. Создание вектора kol_money размером nomin имеет временную сложность O(nomin).

   Основной цикл выполняется до тех пор, пока amount не станет равным 0. В каждой итерации происходит следующее:

4. Возведение в степень money^(nomin - 1) с использованием pow, что имеет временную сложность O(log(money)) в среднем.
5. Вычитание этой степени из amount.
6. Инкрементация соответствующего элемента kol_money.
7. После завершения основного цикла, выполняется цикл для вывода результатов, который имеет временную сложность O(nomin).

Исходя из вышеперечисленного:

Итак, для маленьких и фиксированных значений nomin и money, алгоритм имеет временную сложность O(1). Для больших значений money, алгоритм имеет временную сложность O(log(money)).

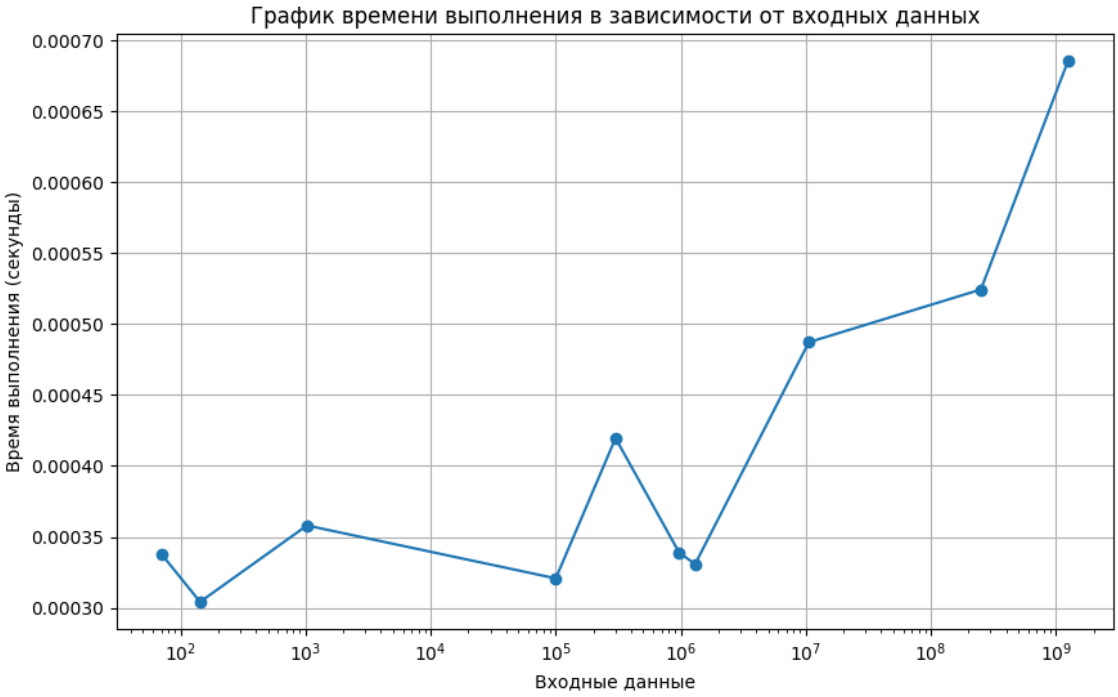

Рисунок 1 - График времени работы программы в зависимости от входных данных

На графике представлена примерная логарифмическая сложность, так как алгоритм работы также зависит от исходного номинала и его степеней, поэтому есть небольшой разброс в точках.

Вывод

Выполнив лабораторную работу №8 по курсу «Дискретный анализ», я изучил жадные алгоритмы и вспомнил C++. Я убедился, что жадные алгоритмы это довольно интересный способ решения задач, его также легко понять. Однако стоит заметить, что не при всех входных данных можно применить данный подход. Что, в конечном итоге, приводит нас к тому, что нужно внимательно проверять входные данные для использования такого метода решения.