Алгоритмы. Жадные алгоритмы.

Информатика
10-11 классы

23 января 2012 г.

Жадные алгоритмы

- Жадный (greedy) алгоритм — пошаговый алгоритм, основанный на *жадном принципе* выбора следующего шага.
- На новом шаге мы выбираем самый “толстый” вариант.
- Формально: для подготовки глобально оптимального решения на каждом шаге выбираем локально оптимальный вариант.

Задача о размене монет

В одном государстве имеются монеты достоинством 1, 2, 5, 10, 20, 50 и 100 тугриков. Вася каждый месяц получает зарплату N. Помогите начальнику Васи выдать сумму N наименьшим количеством монет.

Общий случай

В одном государстве имеются монеты достоинством $a_1 = 1 < a_2 < \cdots < a_k$ тугриков. Вася каждый месяц получает зарплату N. Помогите начальнику Васи выдать сумму N наименьшим количеством монет.

Алгоритм

1. Будем последовательно выдавать Васе по одной монете вплоть до того момента, когда будет выдана вся зарплата (итеративность).
2. *Принцип жадного выбора* в данном алгоритме заключается в следующем: на каждом шаге выберем из всех монет те, которые ещё можно выдать (достоинство монеты меньше оставшейся суммы) и выдадим Васе максимальную из них.
3. “Выдадим ... максимальную” — и есть “жадность” алгоритма.
4. Процесс выдачи можно организовать как циклом (динамическое программирование), так и рекурсией.

Рекурсия

Рекурсивный формализованный алгоритм

1. Зададим монеты в виде массива.
2. Создадим рекурсивную функцию, которая выдаёт одну монету. На вход она будет принимать набор монет и оставшуюся сумму. Возвращать — достоинство монеты, которую надо выдать.
3. База рекурсии: если оставшаяся сумма совпадает с достоинством одной из монет, возвращаем эту монету.
4. Рекуррентная формула: находим максимальную монету, которую можем выдать Васе. Выдаём её Васе и запускаем вызываем функцию с уменьшенным значением суммы.

Реализация рекурсии

Listing 1: Монеты рекурсией

```
def give_coin(coins, n)
  if coins.include?(n)
    puts n
  else
    max = coins.find_all{|elem| elem < n}.max
    puts max
    give_coin(coins, n-max)
  end
end

coins = [1,2,5,10,20,50,100]
n = 398
give_coin(coin, n)
```

Динамическое программирование

Алгоритм с циклом

1. Разобьём задачу на две составляющие: нахождение жадного выбора и цикл выдачи монет.
2. Цикл продолжается до тех пор, пока Васе есть что выдавать.
3. Жадный выбор: находим максимальную монету, которую можем выдать Васе.
4. В специальной переменной сохраняем значение остатка невыданных Васе средств.

Реализация циклом

Listing 2: Монеты циклом

```
coins = [1,2,5,10,20,50,100]
n = 48

while (n != 0)
  max = coins.find_all{|elem| elem<=n}.max
  puts max
  n = n - max
end
```

Задание

- Решить задачу в общем случае для любого количества и достоинства монет.
- (*) Предложенное решение не оптимально: если, например, надо выдать 453 тугрика, то мы можем сразу выдать 400, а не выдавать 4 раза по 100 тугриков, удлинняя в 4 раза алгоритм. Написать программу, реализующую улучшенный алгоритм, который сразу выдаёт максимально возможное количество монет наивысшего достоинства.

Задача о коммивояжёре

Рис.: Источник: http://ru.wikipedia.org

Задача о коммивояжёре

- Коммивояжёру для продажи своего барахла нужно посетить несколько городов. Требуется составить кратчайший из возможных путей. Предполагается, что все города соединены прямыми дорогами каждый с каждым.

Концепт решения - жадный выбор

- Сама задача имеет множество вариантов решений — от простых до более сложных.
- Одно из возможных решений — жадный алгоритм.
- Принципов жадного выбора может быть несколько.
- Возьмём самый простой — из каждого города будем ехать в самый ближайший по списку из тех, в которых мы ещё не были.

Концепт решения - хранение данных

- Во многих задачах стоит вопрос, как хранить данные?
- Вопрос 1: как хранить местоположение городов?
- Простейший вариант — в виде двумерного массива городов. Ключи — номера городов, значения — две координаты (широта и долгота в оригинале, но мы решаем в плоской задаче, поэтому x и y).
- Вопрос 2: надо как-то хранить города, в которых мы уже были.
- Для этого можно завести булевский одномерный массив длиной в количество городов. Если мы побывали в i-ом городе, то делаем i-ый элемент этого массива равным **истине**. По умолчанию выставляем все элементы равными **лжи**.

Алгоритм

1. Напишем функцию, вычисляющую по теореме Пифагора расстояние между двумя городами. На вход — координаты городов (в виде двух массивов), на выходе — вещественное число.
2. Допустим, мы начинаем в точке (0,0). Так как нам надо пройтись на всем N городам, зададим цикл от 0 до $N-1$ (по количеству шагов).
3. Перед этим заведём две переменные, показывающие наше текущее местоположение.
4. Вычислим ближайший к нашему текущему местоположению город. Для этого пройдёмся по всем городам, в которых мы ещё не были (циклом, не рассматриваем такие города, у которых булевское значение соответствующего элемента равно истине).

Алгоритм

5. Вычисление ближайшего города похоже на вычисление минимального элемента массива, только с дополнительным условием в виде истинности элемента булевского массива. Расстояние между городами вычисляем с помощью заданной функции.
6. Вычислив ближайший город, выведем его на экран, как элемент маршрута. Заменим переменные, в которых мы храним текущие координаты.
7. В булевском массиве напротив соответствующего города поставим флажок ИСТИНА.
8. Повторяем операцию вплоть до последнего города. Заканчиваем вместе с первым циклом.

Задание

- Написать программу для решения задачи о коммивояжёре по предложенному алгоритму.
- Придумать такое расположение городов, когда предложенный алгоритм работает очень некорректно (не более 6 городов).
- (*) Решить задачу в пространственном случае, когда для каждого города хранятся широта и долгота, а расстояния вычисляются на сфере, а не в плоскости.

References

- Все презентации доступны на http://school.smirik.ru!
- Вопросы, предложения, д/з: smirik@gmail.com