Куракин Павел Вячеславович
Институт проблем управления им. В. А. Трапезникова РАН

Бросаем монетку и ждем чуда

Решение ряда тривиальных задач может
привести к решению нетривиальной задачи.

Д. В. Беклемишев.
«Курс аналитической геометрии и линейной алгебры»

— Это потому, что здесь одной храбрости мало, — сказала Кнопочка.
— Я в какой-то сказке читала, что надо совершить три хороших поступка подряд.
Тогда перед тобой появится волшебник и даст тебе все, что ты у него попросишь.

Н. Носов. «Незнайка в Солнечном городе»

Введение

Вы любите сложные и непонятные задачи? Я — когда как. Когда у меня получается решить такую задачу, то я ее люблю. А когда не получается — не люблю. (И не люблю тех умников, которые ее решили).

Есть разные способы решать сложные и непонятные задачи. Есть такие умные люди — их зовут математиками, которые подходят к сложной задаче вот таким способом: давайте посмотрим на математическую структуру задачи. Может, эта структура уже давно известна и на этот счет уже есть какая-нибудь готовая теорема, тогда и думать не о чем.

Возможно — и, даже, скорее всего, задача-то сложная! — такой готовой структуры еще не существует. Ну, тогда придется попотеть. Тем не менее, мы все равно будем искать именно математическую структуру задачи. Мы, математики, всегда так делаем. И когда мы ее найдем, нам будем ясно, как запрограммировать эту задачу на компьютере.

Я не могу критиковать этот подход. Это умный подход, правильный подход. Кто хорошо владеет этим методом — тому можно только молча завидовать.

Но это не единственный возможный подход. Не все такие умные, как эти математики, а решать сложные задачи все равно надо. И такая возможность есть! Этот альтернативный подход состоит в том, чтобы маленькими шагами пытаться решить задачу хотя бы в каком-то, ограниченном смысле. Пусть это будет решение настолько плохое, что его вообще нельзя признать как решение. Не беда — мы сделаем следующий шаг, настолько смешной и маленький, насколько возможно, но это будет шаг вперед! И потом мы сделаем еще много таких шагов.

Таким способом можно добраться до полноценного решения, и это настоящее чудо.

Задача

Вот вам такая задача. Представим, что мы бросаем монетку, и смотрим — что выпало, орел или решка. Бросаем, скажем, 10000 раз. И запоминаем или записываем, что у нас выпало. Это называется случайный ряд.

При этом у нас есть некоторая готовая последовательность орлов и решек — а лучше сказать, нулей и единиц. Скажем, 001110011010101. Это неизменная последовательность. Мы ее один раз записали и уже не меняем. Давайте назовем ее «последовательность А» или «образец».

А нашу монетку мы тоже не забываем — мы ее бросаем и бросаем. И записываем. Давайте назовем эту запись «последовательность Б». Это нарастающая по времени последовательность. Бросаем и смотрим на нашу последовательность А (она фиксированная): а не случилось ли так, что в последовательности Б **целиком** встретилась последовательность А? Например вот так:

А: **001110011010101**
Б : 11101010100011**001110011010101**01110101001...

Задача состоит в том, что надо посчитать число раз, сколько такое совпадение произошло. Вот и все. Давайте сформулируем это кратко и, как говорят математики, формально:

Посчитать число вхождений фиксированной бинарной последовательности А длины n в случайную последовательность Б длины m, m > n.

Казалось бы — а что тут сложного? Но есть одна хитрость, дополнительное ограничение: по мере бросания монеты нам запрещается запоминать выпавшие значения. Говоря языком программиста, нам нельзя заводить какой-то массив, в который мы записываем выпавшие нолики и единички. Вообще, мы можем создавать всякие массивы, но нам запрещено запоминать в этих массивах выпавшие значения монетки. И вот именно это ограничение делает задачу сложной.

Пробуем построить хоть какое-нибудь решение

Я даже представить не мог, как это можно не сохранять выпадающие монеты. Поэтому решил: да плевал я на ваши запреты — и я найду самое простое решение, какое только можно. И буду использовать массив! Я создам дополнительный массив, который назову `window` (окно).

Через это окно я буду смотреть на последовательность Б, и смотреть: совпадает ли содержимое моего окна с содержимым последовательности А. При этом я буду шаг за шагом продвигать свое окно вдоль последовательности Б. Если содержимое окна совпало с содержимым последовательности А, я буду увеличивать счетчик совпадений на 1 (Рис. 1).

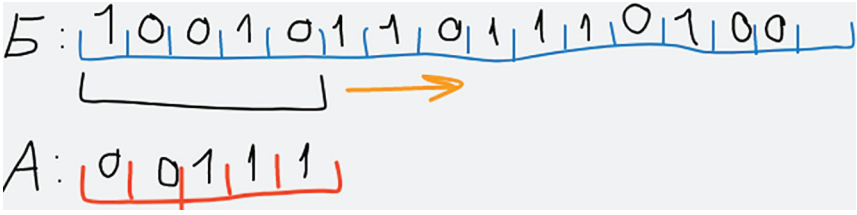

Рис. 1

Давайте оформим этот предварительный алгоритм на языке Python. Для начала поймем, что нам понадобится генератор случайных чисел, значит, в программу надо включить соответствующий модуль:

```
import random
```

Теперь для последовательности А (образца) заведем массив (в Python это список) и посчитаем его длину (pattern - образец):

```
pattern = [0, 1, 1, 1, 0, 0, 1, 1, 0, 1]
lp = len(pattern)
```

Теперь заведем переменную для «окна», в котором мы будем сохранять значения подбрасываемой монетки (мы же решили, что плевали мы на все запреты и ограничения):

```
window = []
```

Заведем счетчик совпадений (coincide по-английски - совпадать):

```
coincidences = 0
```

Все готово, открываем цикл (cnt от слова count - считать):

```
for cnt in range(10000) :
```

Бросаем монетку (coin):

```
 # 0 or 1
  coin = random.choice([0, 1])
```

Добавляем в «окно» выпавшее значение монеты:

```
  window.append(coin)
```

Если длина окна превысит длину образца (последовательность А), мы уберем из «окна» ту монету, которая находится в самом начале. Так мы поддерживаем фиксированный размер «окна» и обеспечиваем «движение» окна вдоль последовательности Б:

```
if len(window) > lp :
  window.pop(0)
```

Кто-то может знать, что такой специфический массив называется «стек». Это очень важное в информатике понятие, рекомендую почитать хорошую статью на эту тему [3]. Если содержимое окна – стека совпало с содержимым образца, увеличиваем счетчик совпадений на единицу:

```
if window == pattern :
  coincidences += 1
```

Готово! Можете не сомневаться — считает абсолютно правильно.

Пытаемся избавиться от окна

Мы сделали важный шаг: нашли хоть какое-то решение, даже ценой того, что мы не выполнили наложенное ограничение. Ограничение было в том, чтобы **не запоминать** выпавшие значения монетки. Хорошо. Но давайте теперь думать: а как же нам обойтись без этого окна?

Давайте попробуем *думать* так, как будто у нас есть окно, но на самом деле не будем использовать этот список. Но хоть какую-то переменную мы все равно должны использовать, верно? Что это может быть за переменная?

Как я предложил, давайте *воображать,* будто у нас по-прежнему есть наше «окно», в котором мы определенным образом сохраняем выпавшие значения монеток. С самого начала у нас окно пустое. Вот мы один бросили монетку — добавили в окно. Второй раз бросили монетку — положили в окно. А разве нам так уж нужно хранить обе эти монетки?

Давайте вернемся на шаг назад. Вот мы бросили монетку первый раз. Она совпадает с первой монеткой в образце (последовательность А). Допустим, не совпадает. Ну и ладно — не будем ее класть в окно. Бросили монетку второй раз. Совпадает с первым значением образца? Да — да, сравниваем по-прежнему **с первым** значением в образце! Опять не совпадает. Бросили монетку в третий раз. Совпадает? Ну надо же — совпало! Отлично, «кладем» монетку в окно. Точнее — запоминаем в массиве «окна».

Так, идем дальше. Бросаем монетку. С чем мы теперь должны сравнивать? Ага, у нас уже есть одна монетка в «окне», она совпала с первым значением в образце. Значит, теперь не надо сравнивать выпавшую монетку с первым значением, а нужно теперь сравнивать **со вторым**!

Хорошо. Снова бросили монетку. Совпала она со вторым значением в образце? Ты погляди — совпала! Значит, (мысленно) положили ее в окно. Снова бросаем монетку. С чем ее сравнивать? Правильно, теперь — **с третьим** элементом образца. И вот оказалось, что очередная подброшенная монета не совпала с третьим элементом образца. Что нам делать?

Мы вздохнем (это обязательно) и забудем про близкое счастье и про третий элемент. Мы будем бросать монетку дальше, но начнем сравнивать **снова с первым элементом** образца. И так будем поступать каждый раз, когда мы встретим несовпадение очередного броска монетки с тем элементом образца, **номер**

которого мы держим в уме, и увеличиваем его при удаче. Когда мы доберемся до последнего элемента в образце — это значит, что мы насчитали еще одно совпадение мысленного окна с образцом.

Смотрите, у нас уже появилась важная новаторская идея: достаточно помнить вот этот **номер элемента образца**, с которым будем сравнивать очередной бросок монеты.

Исправленная версия алгоритма #1

Мы хитрые и кое-что подозреваем, поэтому давайте будем вести подсчет искомого числа совпадений двумя способами: «старым добрым» при помощи «запрещенного» окна и нашим новым способом — мало ли что!

```
pattern = [0, 1, 1, 1, 0, 0, 1, 1, 0, 1]
lp = len(pattern)
window = []

coincidences1 = 0
coincidences2 = 0
```

Вот она — наша единственная переменная, которая как-то ведет учет обработанной информации по броскам монет (ptr от английского слова pointer — указатель, вы мы отслеживаем именно указатель на текущую позицию в образце):

```
ptr = 0
```

```
    Запускаем цикл:

# -------- 1-я итерация алгоритма ---------
for cnt in range(10000) :

  # 0 or 1
  coin = random.choice([0, 1])

  # подсчет 1-м способом
  window.append(coin)

  if len(window) > lp :
    window.pop(0)
  if window == pattern :
    coincidences1 += 1

  # подсчет 2-м способом

  if coin == pattern[ptr] :
    ptr += 1
  else :
    ptr = 0
```

```
  if ptr == lp :
    ptr = 0
    coincidences2 += 1
```

Выводим на печать число совпадений, сначала по первому (железобетонному, но тупому) методу, потом по второму:

```
print(coincidences1, coincidences2)
```

Запускаем программу! И смотрим, что выходит на печать:

```
>> 7 4
```

Хмм?.. А если еще раз?

```
>> 8 5
```

Мда... Еще раз...

```
>> 11 7
```

Что происходит? У нас систематический «недолет»: новый метод (почему-то) дает меньшее число совпадений образца с серией бросков, чем есть на самом деле (ведь мы знаем, что первый метод абсолютно точен). Почему?

Представим, что у нас серия бросков (последовательность Б) и образец (последовательность А) — как на Рис. 2. Если мы действуем нашим новым методом, мы отследим совпадение «окна» (оно продолжает существовать у нас в уме) в позициях #1 и #2. Это так, потому что пока у нас единички и нолики все совпадают и совпадают с образцом, мы не начинаем нового отсчета, не обнуляем переменную **ptr**. И следующее совпадение (воображаемого!) окна с образцом может произойти только без наложения с предыдущим положением окна. А в реальности возможно некое промежуточное положение «окна», совпадающее с образцом (#3), и наш первоначальный метод эту возможность не упустит.

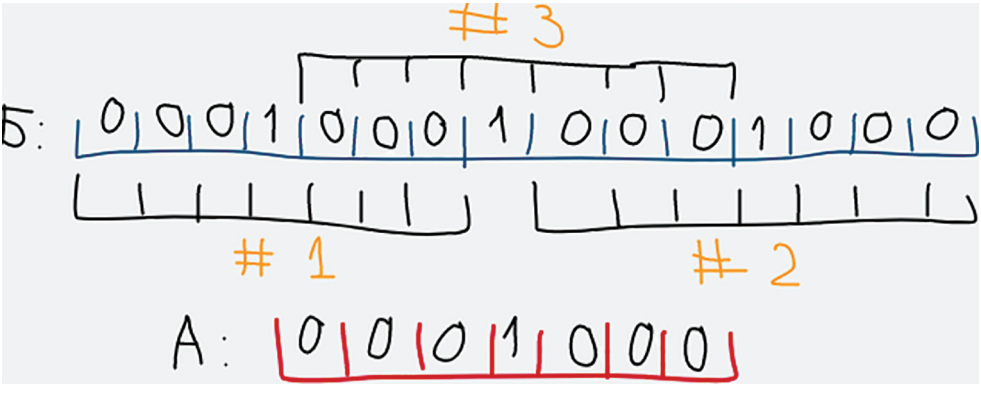

Рис. 2

Исправленная версия алгоритма #2

Такое ощущение, что обнаруженную утечку совпадений несложно исправить. Давайте будем поддерживать целый массив указателей на позицию в образце, а не один **ptr**. Длина этого массива указателей равна длине образца.

Фактически это означает, что мы держим в голове такой же массив «окон», последовательно смещенных друг относительно друга на 1 ячейку, и эти окна синхронно «едут» вдоль последовательности Б (Рис. 3). Тогда уж точно мы не упустим такие наложенные друг на друга окна!

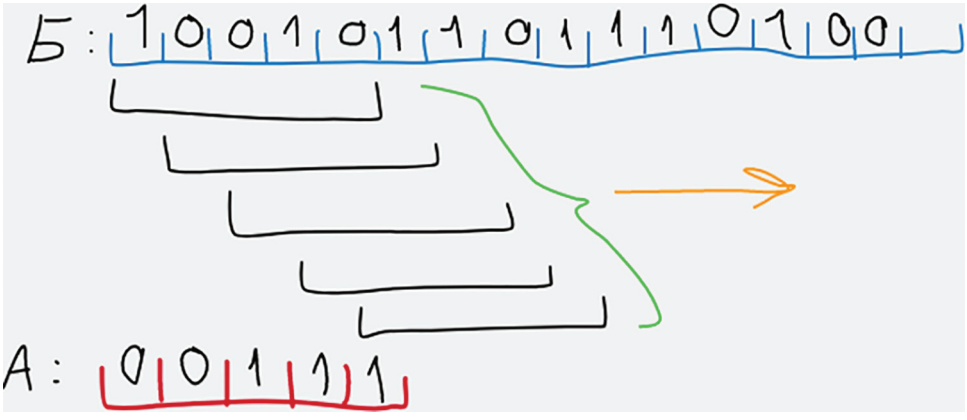

Рис. 3

Стратегия ясна — пишем программу. Начало нам уже известно, тут нет новостей: сначала ведем подсчет самым первым методом с явным созданием массива «окна»:

```
# -------- 2-я итерация алгоритма ---------
pattern = [0, 1, 1, 1, 0, 0, 1, 1, 0, 1]
lp = len(pattern)
window = []

coincidences1 = 0
coincidences2 = 0

for cnt in range(10000) :

  # 0 or 1
  coin = random.choice([0, 1])

  # подсчет 1-м способом
  window.append(coin)

  if len(window) > lp :
    window.pop(0)
```

```
  if window == pattern :
    coincidences1 += 1

…
```

Создаем массив (список) указателей длиной `lp` **перед** запуском цикла:

```
…
pointers = [0 for _ in range(lp)]
```

Теперь ведем подсчет новым (исправленным) способом. При каждом броске монеты перебираем все указатели из массива и для каждого из них придерживаемся той же тактики, как прежде для одного указателя. То есть, если монетка совпала (с ячейкой образца по этому указателю!), увеличиваем этот указатель; если совпадения прекратились — увы — сбрасываем этот указатель в 0.

```
for cnt in range(10000) :

  # подсчет 1-м способом

      …

  # подсчет 2-м способом
  for ptr_id in range(lp) :
```

Очень важный момент: наши «окна» сдвинуты друг от друга на одну ячейку. Если первое (по-человечьи именно первое, но не забываем, что по-питонски это нулевое!) «окно» начинает движение от самого начала бросания монеты (последовательности Б), то каждое следующее окно начинает «движение» на шаг позже.

```
    if ptr_id > cnt :
      break
```

После этой проверки поступаем с каждым счетчиком точно так же, как и раньше с единственным счетчиком:

```
    ptr = pointers[ptr_id]

    if coin == pattern[ptr] :
      pointers[ptr_id] += 1
    else :
      pointers[ptr_id] = 0

    if pointers[ptr_id] == lp :
      coincidences2 += 1
      pointers[ptr_id] = 0

print(coincidences1, coincidences2)
```

Смотрим, что у нас выходит на печать:

```
>> 6 10
>> 4 30
>> 9 50
```

Ух ты — недолета теперь точно нет! Зато, какой перелет!.. Что же мы в этот раз делаем не так?

Исправленная версия алгоритма #3

Самое время произнести вслух то, о чем многие до сих пор догадывались (включая автора), но стеснялись произнести вслух. Мы используем воображаемые «окна» как вспомогательный образ для построения алгоритма, и это правильно, но между движением «окон» и изменением соответствующих им указателей нет простой однозначной связи. «Окно» ползет по последовательности Б **равномерно** (с постоянной скоростью), а указатели то растут, то сбрасываются в ноль. До сих пор я старался тщательно избегать того, чтобы честно признать это. И не пытался погрузить читателя в размышления о выявлении более точного и аккуратного соответствия. Потому что все работало и без этого, и это хорошо!

Давайте постараемся рассуждать максимально просто. Настолько просто, насколько это возможно. Представьте, что одно окно наехало на то место, где недавно было другое окно. Если содержимое окна **в этот момент** совпадает с образцом (последовательность А), это означает, что мы **дважды** посчитаем «включение» образца в серию бросков. Как бы нам исключить из счета такие случаи?

На языке указателей (а не окон) это означает, что какой-то указатель сравнялся по значению с каким-то другим указателем. Остановитесь на минутку, задумайтесь, что это означает, прочувствуйте это: в этом случае получается, что мы «закидываем» выпавшую монетку сразу в два (а то и больше) разных окна, а этого делать не следует.

Так давайте исключим эти ситуации! Мы добавим функцию, которая будет увеличивать произвольный указатель, **только** если указанного совпадения с другими указателями не происходит. Если эта ситуация произошла, соответствующий указатель надо сбросить в нуль. В качестве аргументов функция принимает массив указателей и **номер** указателя в этом массиве:

```
def incrementPtr(ptrs, ptrId) :

  test_val = ptrs[ptrId] + 1

  valid = True
  for pid in range(ptrId):
    if _ptrs[pid] == test_val :
      valid = False
      break

  if valid :
      ptrs[ptrId] = test_val
```

```
  else :
      _ptrs[_ptrId] = 0
```

А теперь повторим предыдущий вариант алгоритма с той оговоркой, что мы будем не механически увеличивать каждый счетчик, а посредством вызова функции **incrementPtr()**:

```
# -------- 3-я итерация алгоритма ---------
…

for cnt in range(10000) :

  # 0 or 1
  coin = random.choice([0, 1])
  #coin = 1
  #coin = cnt%2

  # подсчет 1-м способом
  window.append(coin)
  if len(window) > lp :
    window.pop(0)

  if window == pattern :
    coincidences1 += 1

  # подсчет 2-м способом
  for ptr_id in range(len(pointers)) :

    if ptr_id > cnt :
      break

    ptr = pointers[ptr_id]

    if coin == pattern[ptr] :
      incrementPtr(pointers, ptr_id)
    else :
      pointers[ptr_id] = 0

    if pointers[ptr_id] == lp :
      coincidences2 += 1
      pointers[ptr_id] = 0

print(coincidences1, coincidences2)
```

Проверяем:

```
>> 10 10
>> 14 14
>> 8 8
```

Без комментариев.

Подведение итогов

Для начала отмечу главное: мы вводим дополнительные переменные — целый массив — но мы честно **никак не запоминаем** значения выброшенной монетки. Это принципиально! Мы герои, мы выполнили ограничение задачи. Но надо еще кое-что обсудить.

Нам была дана задача с ограничением, и мы поначалу не представляли, с какого боку к ней подойти. И мы сделали следующее: мы наплевали на ограничение и сделали так, как могли. Криво, косо – но сделали. А следующими шагами мы попробовали, глядя на свое «неправильное», но все же решение, как-то исправить его, внести какие добавления или изменения, **компенсирующие** «запрещенные» или просто неправильные последствия. Причем проделали это несколько раз.

Вы удивитесь, но такой способ смотреть на сложные и вроде бы безнадежные задачи, во-первых, работает гораздо чаще, чем можно ожидать на первый взгляд. Во-вторых — что еще более удивительно — такой способ применим далеко не только в задачах информатики, а вообще везде в технике. Об этом лично я узнал из одной книги выдающегося советского инженера и автора Теории Решения Изобретательских Задач Г.С. Альтшуллера [1]. Я всем рекомендую эту и любые другие книги Генриха Сауловича.

Саму задачу я взял в книжке [2], которую тоже настоятельно рекомендую всем программирующим для души, экзамена и работы. Решения задачи в книжке нет, это мое собственное решение.

И последнее: полученное решение, скорей всего, не единственное.

Литература

1.*Г.С. Альтшуллер.* Алгоритм изобретения. — 2-е изд., испр. и доп. — М.: Моск. рабочий, 1973. — 296 с.

2.*Арсак Ж.* Программирование игр и головоломок: Пер. с франц.— М.: Наука. Гл. ред. физ.-мат. лит., 1990. — 224 с.

3.Что такое стек. "Код" — журнал Яндекс — практикума. https://thecode.media/stack/

Юмор Юмор Юмор Юмор Юмор Юмор

Идет экзамен типа тест с вопросами, на которые надо отвечать «да» или «нет». Один из студентов подбрасывает монетку и записывает результаты. Препод думает: «Ну, этот первым закончит». Экзамен закончился, все студенты ушли, а этот все сидит и монетку подбрасывает. Препод подходит к нему и спрашивает:

— Ну что. Ответил на вопросы?

— Да.

— А чего тогда делаешь?

— Проверяю.