Угадай число

1 Правила игры

Представьте, что вы играете с вашим другом в игру. Друг загадал целое число от 1 до n и записал его на листочке. Вы можете пытаться угадывать число, а ваш друг будет говорить, верна ли ваша догадка.

Конечно, в такой игре при хорошем раскладе вы можете угадать число с первой попытки. Но в худшем случае вам нужно будет перебрать все возможные n вариантов (то есть назвать каждое из чисел), и только последнее окажется верной догадкой. Других разумных стратегий в этой игре нет, ведь когда мы получаем информацию, что наша догадка неверна, это всегда исключает только одно число из возможных ответов. Получается, чтобы гарантированно угадать число из диапазона от 1 до n, нужно предварительно исключить $n - 1$ вариант.

Посмотрим на другую вариацию этой игры. Пусть ваш друг все еще загадывает число от 1 до n, но теперь в ответ на вашу догадку x он сообщает один из трех возможных вариантов ответа: загаданное число больше x, загаданное число меньше x или же вы угадали число. Пусть $n = 20$. Давайте посмотрим, какую информацию мы можем получить, если назовем, например, число 3.

Если окажется, что загаданное число больше 3, то мы можем убрать из множества потенциальных ответов не только тройку, но и двойку и единицу. Получается, что за один запрос в случае ответа "больше" мы исключим три варианта. Если окажется, что загаданное число меньше трех, мы уберем из рассмотрения сразу восемнадцать чисел: ответом могут быть только числа один и два. Получается, назвав число $x = 3$, мы в худшем случае уберем из рассмотрения 3 числа (если загаданное число окажется больше). Если мы хотим минимизировать количество ходов, за которое мы **гарантированно** угадаем число, то нам нужно рассматривать именно худший случай – то есть представить, что нам постоянно не везет.

2 Оптимальный алгоритм решения

Давайте будем выбирать x таким образом, чтобы даже в худшем случае убирать как можно больше вариантов. Будем выбирать x ровно посередине отрезка из текущих

возможных значений. Например, в начале у нас возможны значения от 1 до 20, давайте выберем $x = 10$. Если нам скажут, что загаданное число больше, мы отбросим десять вариантов, а если меньше — одиннадцать вариантов.

Давайте промоделируем нашу игру с такой стратегией до конца, предполагая, что нам всегда "не везет", то есть остается наибольший из отрезков возможных значений. Как мы уже обсудили, на первом этапе выберем $x = 10$. Нам отвечают, что загаданное число больше, поэтому убираем из рассмотрения все числа от 1 до 10. Выберем середину отрезка оставшихся чисел – число 15. В худшем случае нам снова сообщают, что загаданное число больше. Мы выбрасываем из рассмотрения числа от 11 до 15, остаются числа от 16 до 20. Теперь назовем $x = 18$. Оба случая одинаковые по количеству отбрасываемых значений – давайте для разнообразия загаданное число окажется меньше. Из рассмотрения уходят числа $18, 19, 20$, остаются 16 и 17. Спросим про одно из них, в нашем случае число 16, и если мы не угадали, спросим про оставшееся число. Таким образом, для $n = 20$ мы гарантировано отгадаем число за пять запросов.

3 Код решения

Давайте попробуем переложить эту игру на программу. Ваша программа будет играть за вас, то есть угадывать число, а вы будете играть за вашего друга: загадывать число и отвечать на вопросы программы. Первое, что вам потребуется, это ввести число n, чтобы обозначить границы рассматриваемых значений чисел. Теперь давайте заведем переменные для поддержания границ подходящих под ответ значений. Будем в переменной `left` хранить левую границу рассматриваемых значений, а в переменной `right` – правую границу рассматриваемых значений. Изначально положим левую границу равной `1`, а правую границу равной `n`:

```
n = int(input())
left = 1
right = n
```

Понятно, что теперь нам потребуется цикл, который будет выполнять запросы. Мы можем делать это, пока не угадаем число (то есть пока не получим соответствующий ответ на запрос), или же пока в диапазоне допустимых значений есть хотя бы одно число, то есть пока `left <= right`. Давайте в этой задаче воспользуемся тем, что загаданное число точно есть и нам в какой-то момент скажут, что мы угадали. Будем бесконечно задавать вопросы, а как только получим утвердительный ответ, завершим программу.

Мы решили, что сделаем бесконечный цикл. Теперь на каждой итерации нам нужно выбрать в качестве `x` середину отрезка. Прибавим к левой границе половину длины отрезка. Если мы немного математически упростим запись, то увидим, что середина отрезка — это сумма его границ, делённая на два:

$$\texttt{x} = \texttt{left} + \frac{\texttt{right} - \texttt{left}}{2} = \frac{\texttt{left} + \texttt{right}}{2}$$

Теперь, когда мы определили значение `x`, сделаем соответствующий запрос. Для этого воспользуемся функцией `ask`, которую напишем позднее. Эта функция будет возвращать, например, минус единицу, если загаданное число меньше `x`, единицу, если загаданное число больше `x`, и завершать программу, если мы угадали число. Если загаданное число меньше `x`, то нам нужно сдвинуть правую границу диапазона, ведь числа `x`, `x+1` и так далее больше не подходят. Тогда положим `right = x-1`. А если загаданное число больше `x`, это означает, что `x`, `x-1` и так далее до единицы не подходят – тогда положим `left = x+1`. Тогда, наш поиск будет выглядить так:

```
n = int(input())
left = 1
right = n
while True:
    x = (left + right) // 2
    if ask(x) == -1:   # загаданное число меньше x
        right = x - 1
    else:              # загаданное число больше x
        left = x + 1
```

Осталось написать функцию `ask`. Она будет довольно простой. Сначала, мы выведем на экран число `x`, которое приняли в качестве аргумента, затем считаем строку (которую вводите вы, как загадывающий число). В строке будет символ `'<'` (меньше), `'>'` (больше) или `'='` (равно). Если это символ `'='`, то завершим программу, а иначе вернем минус единицу или единицу — в зависимости от знака:

```
import sys

def ask(x):
    print(x)
    ans = input()
    if ans == '=':
        sys.exit(0)
    return -1 if ans == '<' else 1
```

Для этой игры мы воспользовались методом **двоичного поиска** (еще его называют бинарным, от английского binary search).

4 Интерактивные задачи

Теперь давайте попробуем модифицировать наш код, чтобы можно было решить другую задачу: ваша программа должна будет все так же совершать догадки о загаданном числе, а вот вашу роль теперь будет играть программа жюри. Жюри загадает число, и их программа будет общаться с вашей, отвечать на ее запросы. Такие задачи, когда ваша программа общается с программой жюри, называются **интерактивными**.

На самом деле, если программа жюри будет вместо вас подавать результат сравнения на

вход вашей программе, то почти ничего менять и не придется. Нужно только учесть одну особенность. Когда вы просите программу вывести на экран какую-то информацию, на самом деле это не происходит моментально. Дело в том, что в момент вывода ваша программа останавливается операционной системой и вывод передается куда требуется (в нашем случае на экран или программе жюри). Такая операция довольно требует довольльно большое количество времени, поэтому Python старается оптимизировать их количество. Python накапливает информацию, которую нужно вывести, в некоторый буфер, и периодически его сбрасывает – то есть выводит накопленные данные. А нам нужно выводить число сразу, так как что пока мы не выведем, мы не получим ответ от программы жюри. Для этого сбросим буфер, передав параметр `flush=True` в функцию `print`:

```
import sys

def ask(x):
    print(x, flush=True)
    ans = input()
    if ans == '=':
        sys.exit(0)
    return -1 if ans == '<' else 1
```

Теперь наша программа стала примером правильного решения интерактивной задачи. Замечание на будущее: в интерактивных задачах вердикты тестирующих систем могут быть различными, и в случае, например, неправильного ответа может происходить разное странное поведение программы жюри, что может привести, например, к ошибке исполнения или превышению ограничения на время выполнения. В дальнейшем мы еще поупражняемся с двоичным поиском на примере классических, не интерактивных задач.