Помехоустойчивость

Для ращения данной задачи можно использовать строки. Наша задача заключается в выборе сообщения с наименьшим количеством помех. Мы начинаем сравнивать наше сообщение s, с полученными сообщениями и ответом будет являться сообщение, которое имеет минимальное отклонение.

Городок H

Это классическая задача на теорию графов тут нужно обратить внимание на то что нам не сказано про направление дорог, поэтому мы считаем, что они направлены в обе стороны. После этого вспоминаем, что количество ребер графа равно половине суммы степеней его вершин. И наша задача сводится к подсчету количества «1» и деления результата на два.

На старт! Внимание!

Не смотря на запутанное на первый взгляд условие, наша задача сводится к поиску строк внутри строки.

По условию задачи мы должны найти количество вхождений строки b (и её циклических сдвигов) в строку a.

Обратимся к примеру, показанному в задаче:

StrA = ‘abcabc’

StrB = ‘abc’

Ответ: 4

Строка B имеет три вариации циклического сдвига: abc, bca, cab. Строка abc встречается два раза, строки bca и cab по одному. Таким образом вся задача сводится к генерации всех циклических сдвигов строки и поиску их в строке A.

Код Да Винчи

Для решения данной задачи необходимо перевести введенное число во все системы счисления от 2 до 36 и проверить его на симметричность.

Инвентаризация

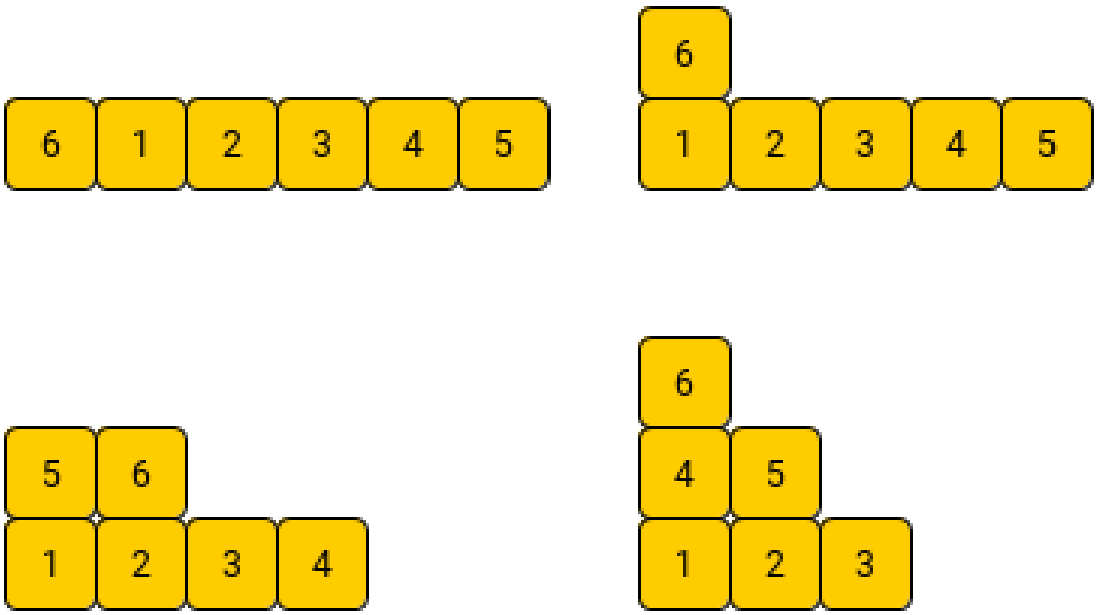
Для решения данной задачи используем метод динамического программирования. Пусть d[i] - количество чисел, состоящих ровно из i цифр и оканчивающихся на цифру, отличную от нуля. Аналогично, обозначим через d[0] количество i-значных чисел, оканчивающихся на нуль. При этом будем рассматривать только те числа, которые не содержат в своей записи двух нулей подряд. Так же не будем рассматривать числа с лидирующими нулями,

даже в том случае, когда i=1 (так как у нас n>1). Тогда очевидно, что d[1]=k-1 и d[0]=1. Теперь мы можем выразить d[i] через d[i-1] и d[i-2] следующим образом: d[i] = (d[i - 1] + d[i - 2]) * (k - 1) Используя данные рекуррентные соотношения, мы можем линейно, пробегая по i от 2 до n, вычислить d[n] и d0[n]. Ответом на задачу будет значение d[n]+d0[n]. Вышеописанный алгоритм решения представим в следующей форме:

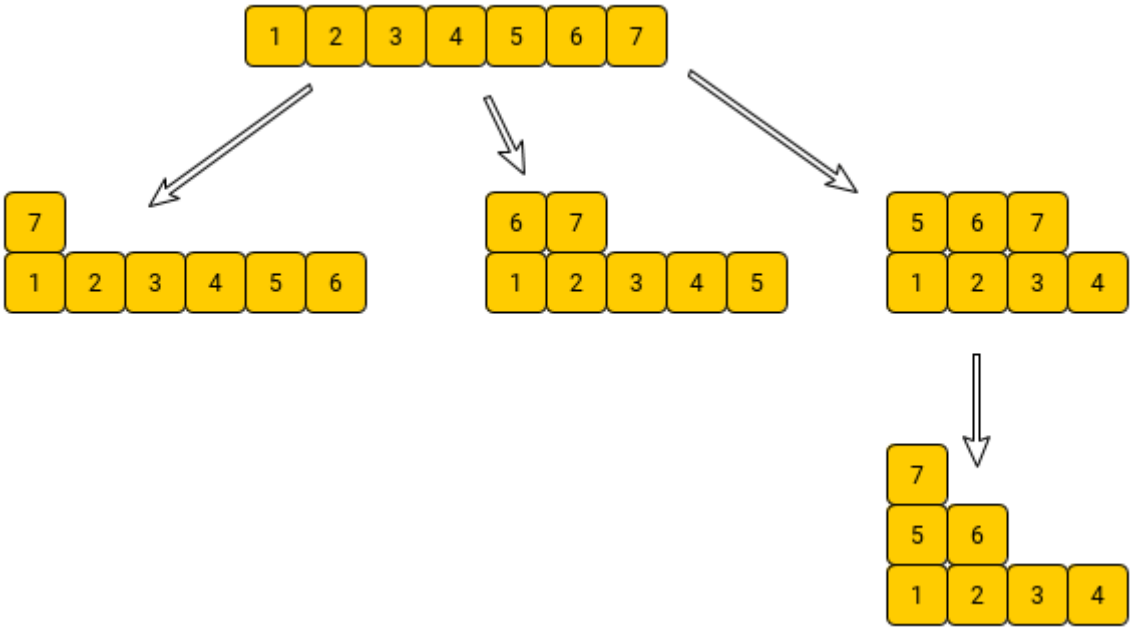
```
def main(n, k):
    d = [0] * 200
    d[0] = 1
    d[1] = k - 1
    for i in range(2, n + 1):
        d[i] = (d[i - 1] + d[i - 2]) * (k - 1)
    print(d[n])


if __name__ == '__main__':
    N, K = map(int, input().split())
    main(N, K)
```

Укрытия

Для решения задачи нужно для начала понять, что из себя представляет лесенка. Лесенка — это набор кубиков где каждый следующий ряд меньше прошлого. Для лесенки из шести кубиков возможны четыре варианта расположения:



Для семи кубиков уже пять:



Если у нас есть 7 кубиков, положенных в ряд, то число кубиков, которое мы можем положить на следующий ряд может быть рано 1,2,3. Положить 4 кубика наверх уже нельзя потому что тогда следующий ряд будет больше прошлого. После чего мы будем иметь ряд из 1,2,3 кубиков к которым можно применить такое разложения.

```
def get_count(prev_level, n):
    if n == 0:
        return 1
    count = 0
    for level in range(1, prev_level):
        if (n - level) < 0:
            break
        count += get_count(level, n - level)
    return count


if __name__ == '__main__':
    n = int(input())
    print(get_count(n + 1, n))
```

Функция get_count возвращает единицу (лесенку удалось сгенерировать) если количество кубиков равно нулю — т.е. мы каким-то корректным образом строили лесенку и у нас кончились кубики. Для остальных случаев мы запускаем рекурсивную функцию с различными наборами данных, уменьшая количество кубиков на текущем уровне и увеличивая на следующем.