Задача A. Поиск подстроки

Задачу можно решить префикс-функцией. Строим строку $S + \# + T$, на ней считаем префикс функцию и потом выводим индексы всех ячеек, длина префикса в которых равна длине S

```
def pref_func(s):
    n = len(s)
    p = [0 for i in range(n)]
    for i in range(1, n):
        j = p[i-1]
        while j > 0 and s[i] != s[j]:
            j = p[j-1]
        if s[i] == s[j]:
            j += 1
        p[i] = j
    return p

def find_pattern_occurrences(t, s):
    re = s + '#' + t
    p = pref_func(re)
    return [i-2*len(s) for i in range(len(s)+1, len(re)) if p[i] == len(s)]
```

Задача B. Задачечка на строчечки

Самое сложное в этой задаче - это прочитать условие. Когда условие прочитано, то легко понять, что задача решается алгоритмом Ахо-Корасик. Делаем дерево поиска по строкам запросов

```
t = input()
n = int(input())
answer = [[] for i in range(n)]
counter = [0]*n
bohr = [Node()]
bohr_size = 1
for i in range(n):
    s_f = input()
    add(s_f, i)
```

Затем находим все вхождения строк в данный текст

```
def find(s):
    u = 0
    for i in range(len(s)):
        u = get_go(u, s[i])
        temp = bohr[u]
        while temp != bohr[0]:
            for j in temp.pattern_num:
                answer[j].append(i-temp.len_word+2)
                counter[j] += 1
            temp = bohr[get_suflink(temp.number)]
find(t)
for i in range(n):
    print(counter[i], end=" ")
    print(*answer[i])
```

Задача C. Лотерея

Сложим все билеты в бор, посчитаем для каждой вершины число терминалов в поддереве. Будем обходить его в глубину, поддерживая текущий размер выплаты. в конце выбираем минимальный.

Фиксируя очередную i-ю цифру мы знаем, сколько нам надо будет заплатить сверх уже набранной суммы: всем терминалам в поддереве, которое мы выбрали, надо доплатить $A_i - A_{i-1}$,

а количество терминалов мы знаем. При этом, как только мы находим, что есть вариант поставить цифру, которую никто не написал в i-м разряде, то дальше можно не перебирать, потому что оптимальная дополнительная выплата будет нулевая.

```
def get_num(seq, cur_index, node):
    global best_seq
    global best_price
    if seq in used or node is None or cur_index >= m:
        return seq

    if len(node.next) < k:
        for i in range(k):
            if str(i) not in node.next:
                final_seq = seq + (str(i),)
                if final_seq not in used and len(final_seq) <= m:
                    final_price = check_price(root, ''.join(final_seq))
                    if final_price < best_price:
                        best_seq = final_seq
                        best_price = final_price
                    if final_price == 0:
                        return final_seq
                used.add(seq)
                return seq
    for i in range(k):
        final_seq = get_num(seq + (str(i),), cur_index + 1, node.next[str(i)])
        if final_seq not in used:
            final_price = check_price(root, ''.join(final_seq))
            if final_price < best_price:
                best_seq = final_seq
                best_price = final_price
            if final_price == 0:
                return final_seq
        used.add(final_seq)
    used.add(seq)
    return seq


def check_price(root, T):
    count = 0
    node = root
    price = 0
    for i in range(len(T)):
        if T[i] in node.next:
            node = node.next[T[i]]
            count = node.count
        else:
            break
        price -= count * prices[i]
        price += count * prices[i+1]
    return price
```

Задача D. Вирусы

Задача также решается используя ахо-корасик. На построенном дереве запрещенных строк, мы хотим найти цикл, который не содержит терминальных вершин, используя который сможем постро-

ить нашу бесконечную строку. Цикл в графе будем искать с помощью обхода в глубину, закрашивая посещенные вершины в два цвета: при входе в вершину красим ее в серый, при выходе в черный. Если мы попали в серую вершину при обходе, то в графе, есть цикл, и мы можем построить бесконечную строку, иначе нет.

```
def dfs(v: Node):
   v.c = 1
   for u in v.go:
       if v.go[u].terminal:
           continue
       if v.go[u].c == 0:
           if dfs(v.go[u]):
               return True
       elif v.go[u].c == 1:
           return True
   v.c = 2
   return False

n = int(input().rstrip())
p = Trie()
for i in range(n):
   p.add(input().strip())
bfs(p.root)
s = ''
if dfs(p.root):
   print('TAK')
else:
   print('NIE')
```