Ахо-Корасик

15/04/2023

В прошлой серии: ДКА

- Конечный автомат - это ориентированный граф $G = (V, E)$, где на каждом ребре записана буква, принадлежащая алфавиту Σ
- Есть одна стартовая вершина и выделенные терминальные
- Автомат "принимает" строку, если ей можно пройти от старта до терминала по соответствующим буквам

Задача

- Найти вхождения ключевых слов $s_1, s_2, \ldots, s_n$ в текст T.

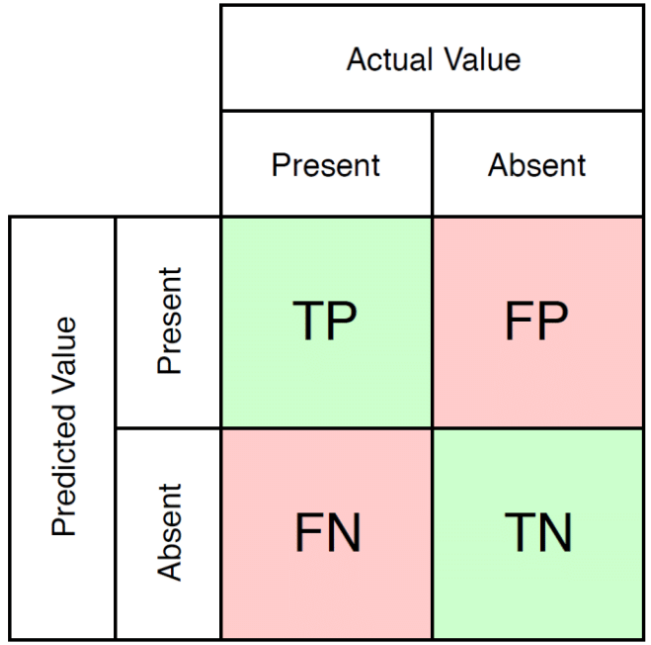
Применение

- Классификация текстов, спам-фильтр

Оффтоп: как сделать спам-фильтр

- Составляем словарь
- **Находим вхождения s_i в T**
- С помощью формулы Байеса можно оценить вероятность того, что T спам, на исторических данных.

$$isSpam(T) = P_{spam}(T) \geq p_{threshold}$$



Подзадача 1: много строк

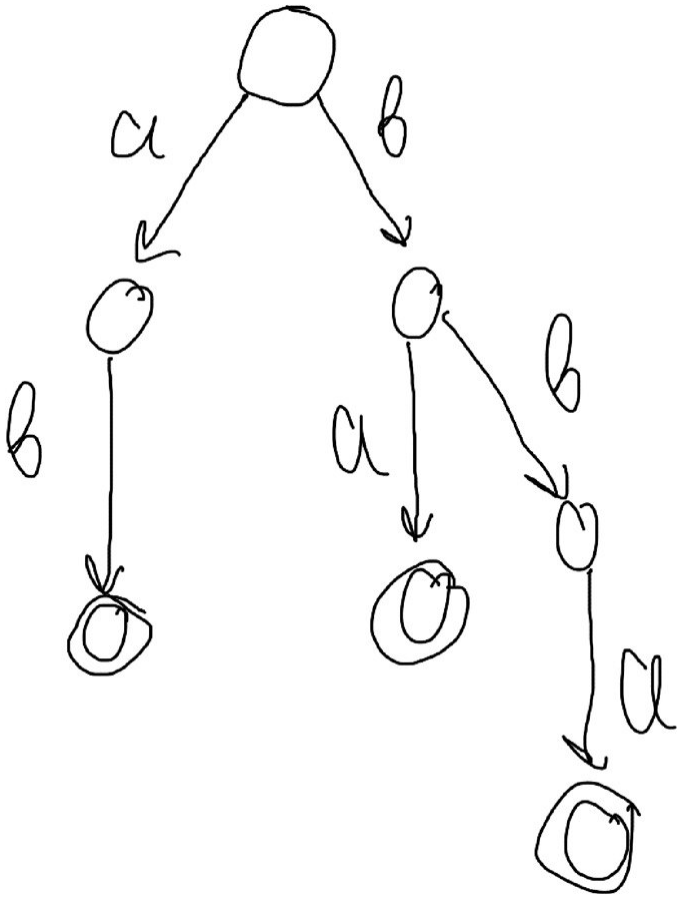
- Хотим проверить, что T является словом из словаря ($T = s_i$)
- Можно сделать хеш-таблицу
- Хочется детерменированное время работы

Бор

Построим ДКА, который будет выглядеть как дерево, в которое мы явно сложили строки.

Добавление строки в такое дерево работает за $O(|s|)$, проверка T на вхождение за $O(|T|)$.

Бор



Реализация

```
class Node:
    def __init__():
        self.next = {}
        self.terminal = False

def add(root, s):
    node = root
    for c in s:
        if c not in node.next:
            node.next[c] = Node()
        node = node.next[c]
    node.terminal = True
```

Переход $T = s_i \to s_i \in T$

Мы научились проверять, что T соответствет какому-то слову за $O(|T| + \sum_i |s_i|)$.

Но для того, чтобы находить вхождение слова, нужно будет искать в боре сначала `T[0:]`, потом `T[1:]`, и так далее. Получится асимптотика $O(|T| \max |s_i| + \sum_i |s_i|)$, которая нас не устраивает.

Вопрос: Почему можно не ставить правую границу для подстроки?

Подзадача 2: поиск s в T

Чтобы упростить проблему с прошлого слайда, пока что будем считать, что работаем с одной строкой.

Постановка: Дана строка s, хотим найти все ее вхождения в T за $O(|s| + |T|)$.

Префикс-функция, КМП

Сначала введем префикс-функцию. π_i - самая большая такая длина (не равная $i+1$), что подстрока кончающаяся в i, и совпадающая с префиксом, имеет длину π_i.

Иначе говоря,

$$s[0:\pi_i] = s[i-\pi_i+1:i+1]$$

Тогда для строки $s' = s\#T$ все индексы с $\pi_i = |s|$ соответствуют вхождению s в T.

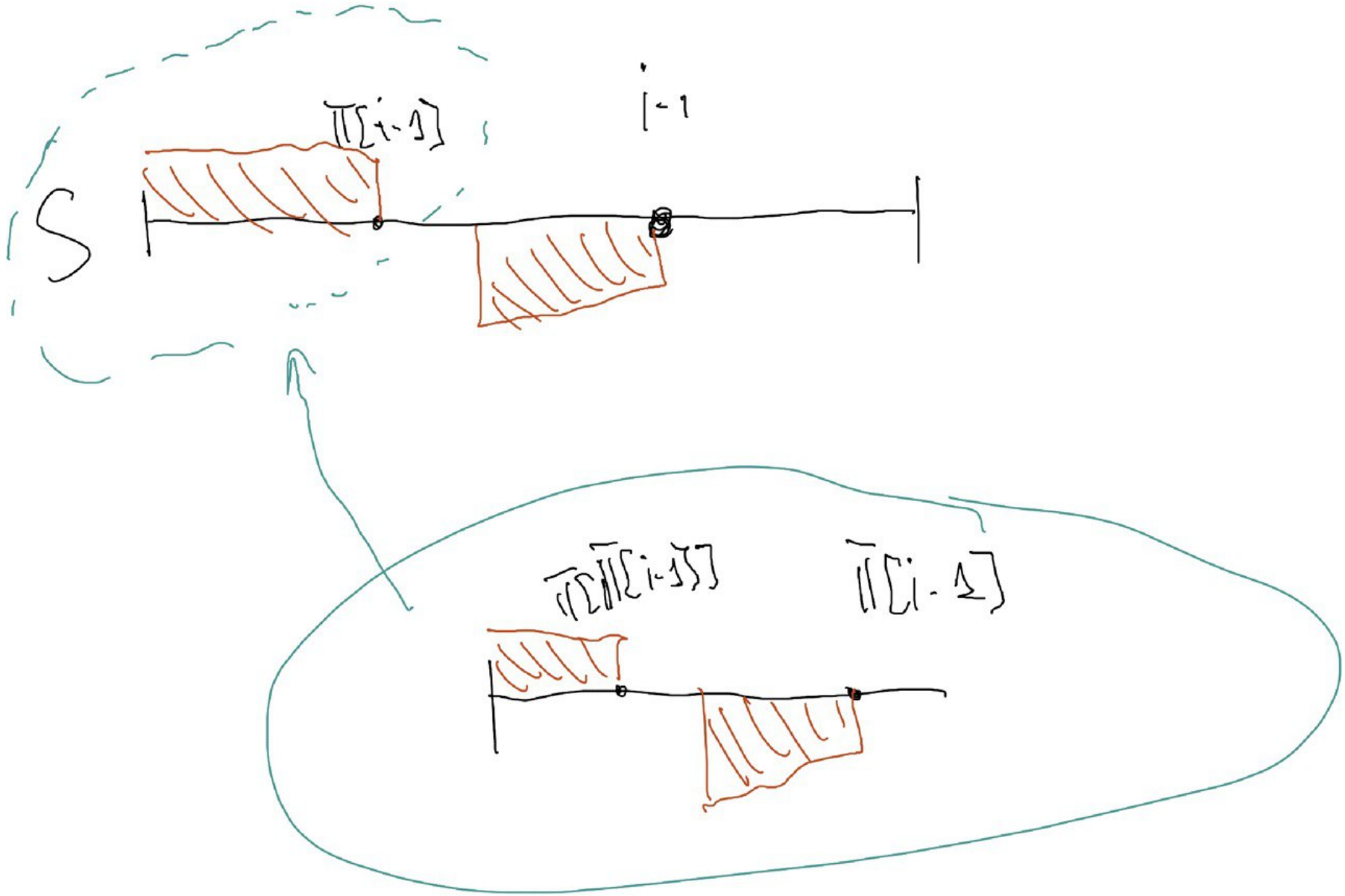
Подсчет префикс-функции

Пусть мы посчитали все $\pi_k, k < i$

Тогда мы знаем ответ для π_{i-1}, можем попробовать его продолжить.

```
if s[i] == s[pi[i-1]]:
    pi[i] = pi[i-1] + 1
```

Но что делать, если равенства не случилось?



Реализация

```
pi = [0] * n
for i in range(len(s)):
    k = pi[i-1]
    while True:
        if s[i] == s[k]:
            pi[i] = k + 1
            break
        if k == 0:
            break
        k = pi[k-1]
```

Асимптотика $O(|s|)$ (в случае КМП $O(|s| + |T|)$) амортизированно, потому что увеличений k не больше чем $|s|$, а каждая итерация $while$ уменьшает k.

Суффиксная ссылка

В случае префикс-функции, выражение $(\pi_i - 1)$ является суффиксной ссылкой для i. Оно показывает самый большой суффикс, который совпадает с префиксом s, не равный самому себе.

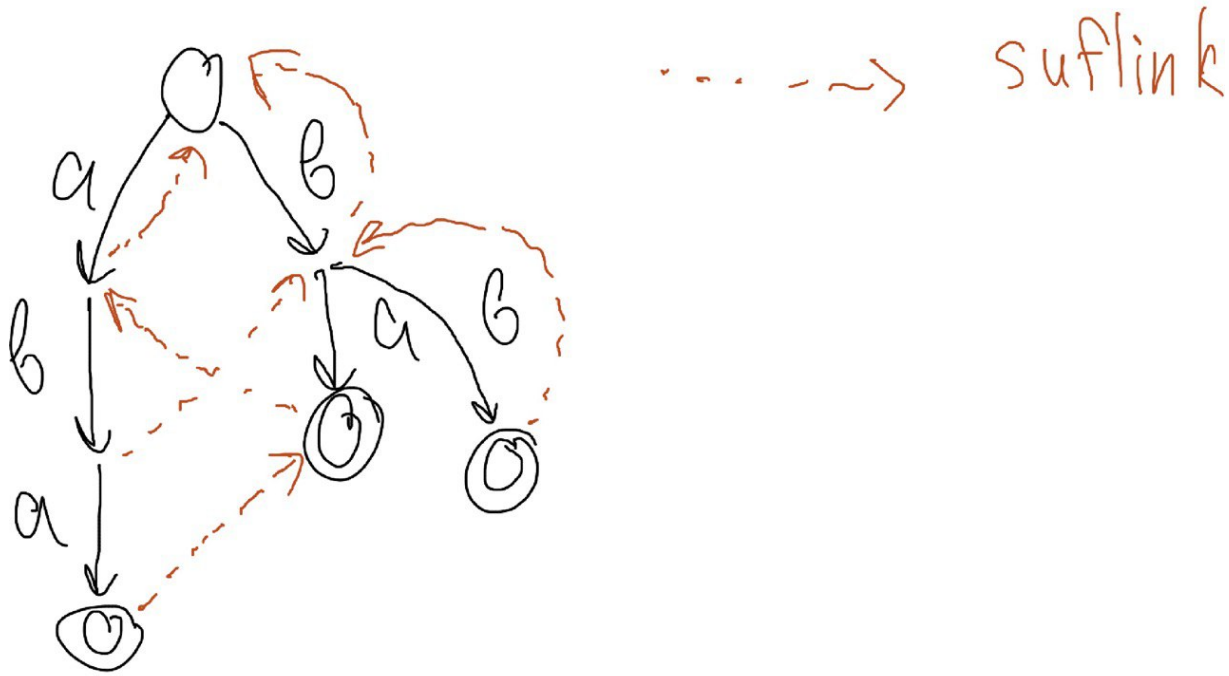
Ахо-Корасик

Суффиксные ссылки + Автомат + Бор!

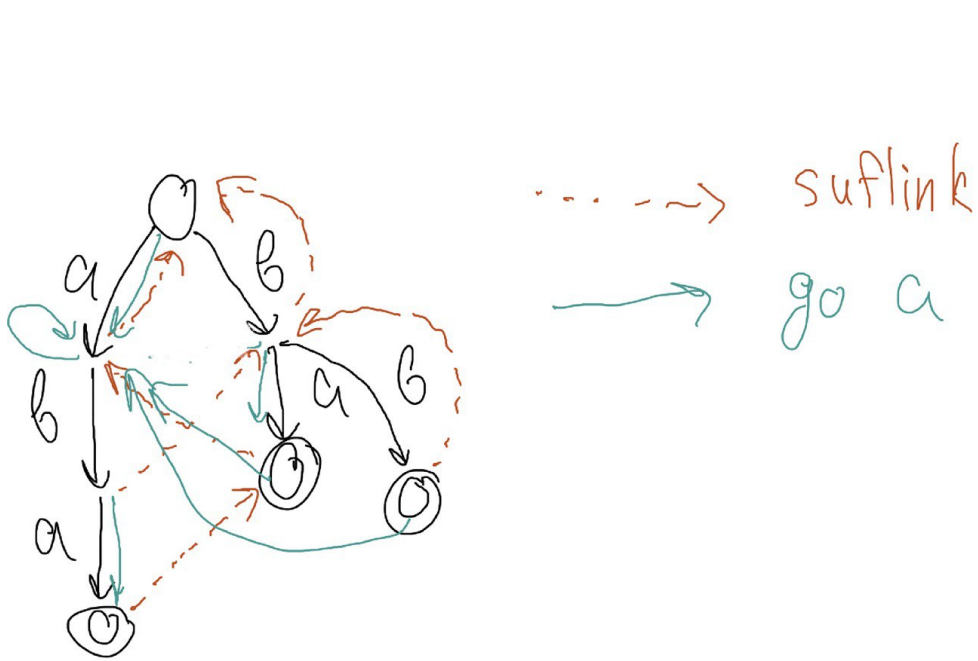
Алгоритм

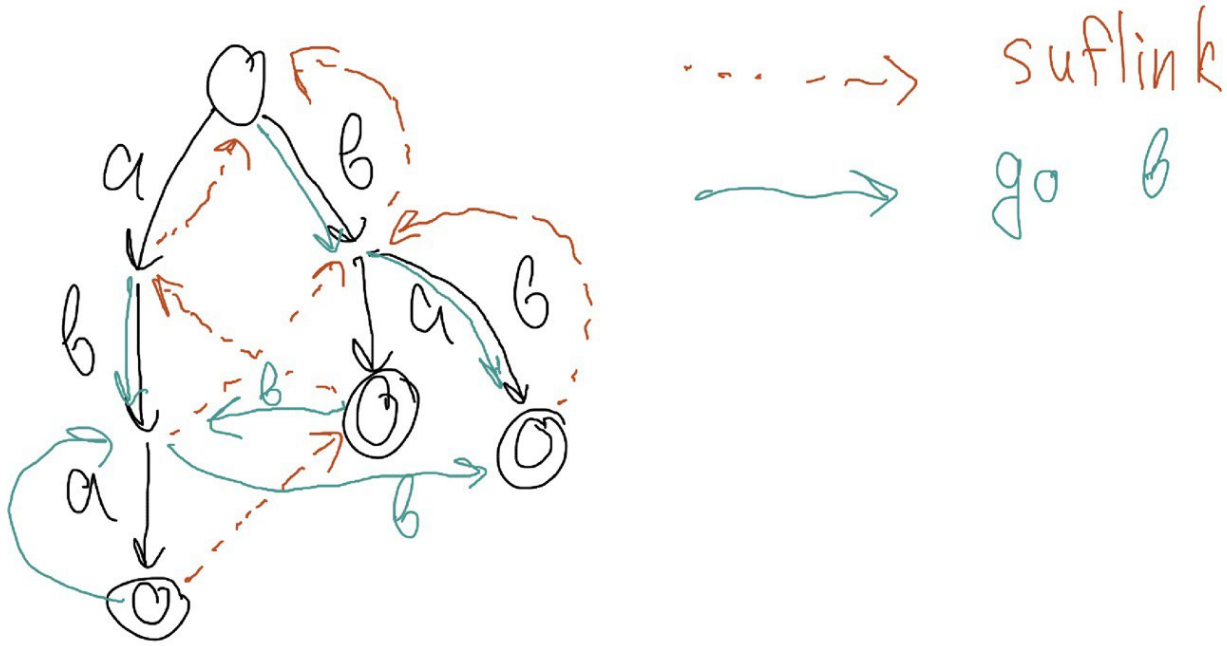
1. Сложим все слова из словаря в бор.
2. Определим $suflink_v = u$, если s_u наибольший суффикс s_v из встречающихся в боре.
3. Построим автомат: $go_{v,c}$ будет соответствовать самой глубокой вершине в боре, которая соответствует суффиксу строки $s_v + c$

Суфссылки



Автомат





Использование

Будем последовательно "идти" текстом T по автомату.

Из каждой $node$ можем подниматься по $suflink$ наверх, чтобы найти терминальные вхождения:

```
node = root
for i in range(len(T)):
    node = go[node][T[i]]
    v = node
    while v != root:
        if v.terminal:
            print(f'T[{i-v.depth + 1}:{i+1}]')
        v = v.suflink
```

Пересчет ссылок и автомата

Как посчитать $go_{v,c}$? Если есть ребро $v \to u$ с символом c, то $g_{v,c} = u$. Иначе можно посмотреть $suflink_v$, который является самым большим суффиксом в дереве, и посчитать через него:

$$go_{v,c} = go_{suflink_v,c}$$

Как посчитать $suflink_v$? Можно посмотреть на $suflink_{parent_v}$. Если из него есть переход по символу на ребре $parent_v \to v$, то переход по этому ребру будет вести в искомую суфссылку. Иначе нужно подниматься выше.

Можно заметить, что если сохранить символ на ребре $parent_v \to v$ как pc, то:

$$suflink_v = go_{suflink_{parent_v},pc}$$

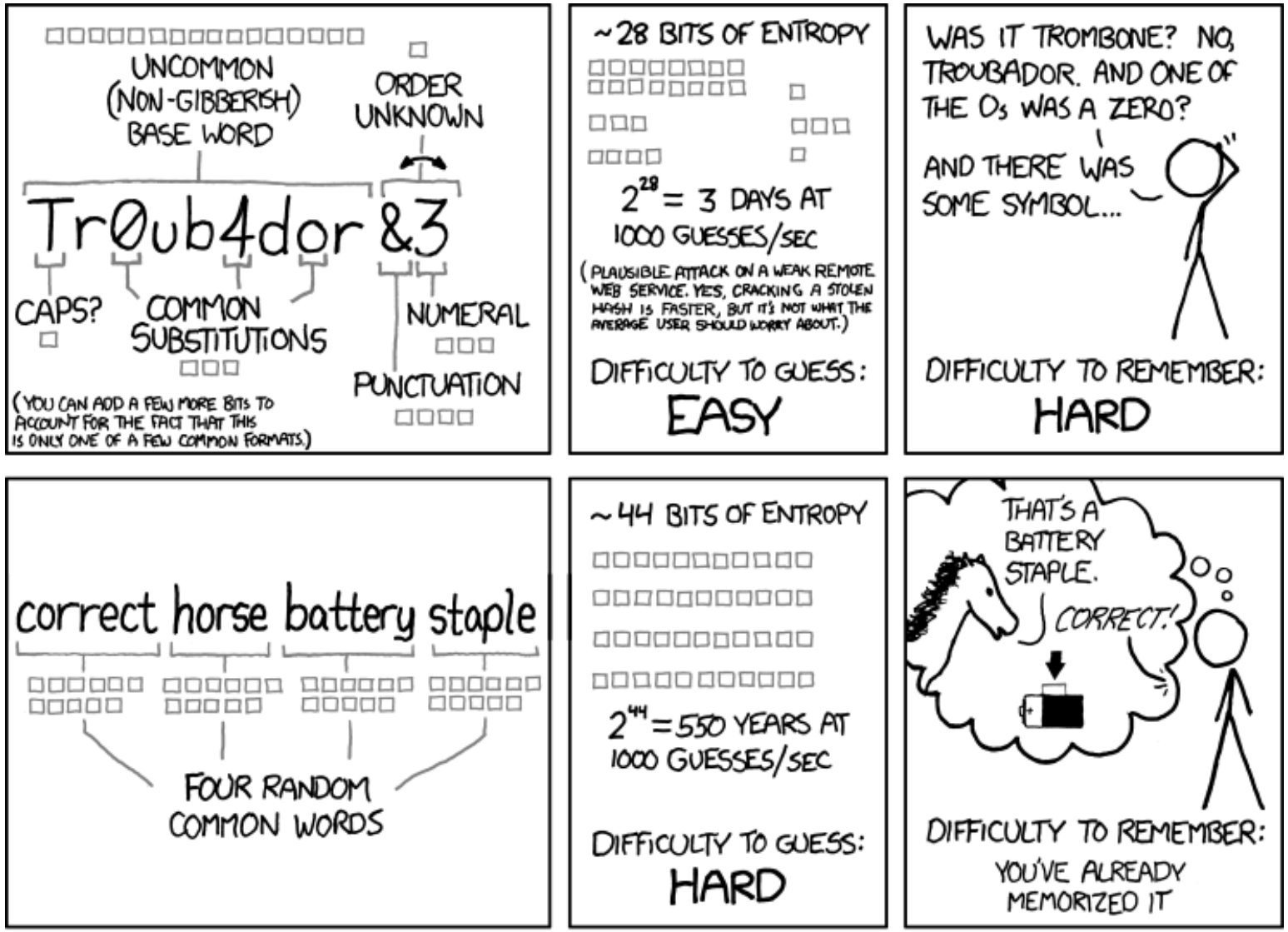
Реализация

```
@dataclass
class Node:
    pc: string, parent: Node, go: Dict[string, Node], suflink: Node

def get_suflink(v):
    if v.suflink is not None:
        return v.suflink
    elif v == root:
        v.suflink = root
    elif v.parent = root: # why do we need this?
        v.suflink = root
    else:
        v.suflink = get_go(get_suflink(v.parent), v.pc)
    return v.suflink

def get_go(v, c):
    if c in v.go:
        return v.go[c]
    elif v == root:
        v.go[c] = root
    else:
        v.go[c] = get_go(get_suflink(v), c)
    return v.go[c]
```

Бонус: сжатые суффссылки

```
def get_compressed_link(v):
    if v.compressed_suflink is not None:
        return v.compressed_suflink
    elif v == root:
        v.compressed_suflink = v
    elif v.suflink.terminal:
        v.compressed_suflink = get_suflink(v)
    else:
        v.compressed_suflink = get_compressed_link(get_suflink(v))
    return v.compressed_suflink
```

Мем


UNCOMMON (NON-GIBBERISH) BASE WORD
ORDER UNKNOWN
Tr0ub4dor&3
CAPS?
COMMON SUBSTITUTIONS
NUMERAL
PUNCTUATION
(YOU CAN ADD A FEW MORE BITS TO ACCOUNT FOR THE FACT THAT THIS IS ONLY ONE OF A FEW COMMON FORMATS.)
~28 BITS OF ENTROPY
2^28 = 3 DAYS AT 1000 GUESSES/SEC
(PLAUSIBLE ATTACK ON A WEAK REMOTE WEB SERVICE. YES, CRACKING A STOLEN HASH IS FASTER, BUT IT'S NOT WHAT THE AVERAGE USER SHOULD WORRY ABOUT.)
DIFFICULTY TO GUESS:
EASY
WAS IT TROMBONE? NO, TROUBADOR. AND ONE OF THE Os WAS A ZERO?
AND THERE WAS SOME SYMBOL...
DIFFICULTY TO REMEMBER:
HARD
correct horse battery staple
FOUR RANDOM COMMON WORDS
~44 BITS OF ENTROPY
2^44 = 550 YEARS AT 1000 GUESSES/SEC
DIFFICULTY TO GUESS:
HARD
THAT'S A BATTERY STAPLE.
CORRECT!
DIFFICULTY TO REMEMBER:
YOU'VE ALREADY MEMORIZED IT
THROUGH 20 YEARS OF EFFORT, WE'VE SUCCESSFULLY TRAINED EVERYONE TO USE PASSWORDS THAT ARE HARD FOR HUMANS TO REMEMBER, BUT EASY FOR COMPUTERS TO GUESS.

Для дальнейшего изучения

- Спам-фильтры
- bfs-версия Ахо-корасик + реализация
- Z-функция
- Сжатый бор