

$S: \underline{a} \underline{b} a c \underline{a} \underline{b} a b$

предикс P_i 0 0 1 0 1 2 3 2

предикс: $\underline{a} \underline{b} a$

суффикс: a, ba

Предикс
суффикс

Пр. срдчкодл - одна max суфф. для каждого предикса ког. слов.
с предиксом \underline{a} все строки?

Сложность: $O(n \times n \times n) = O(n^3)$

для каждого
позиции поиск

внутри одної
позиції перебирає
все суффікси
в середньому $n/2$

сравниваєм суффікс
и предикс

Оптимизация 1:

$\boxed{a} \boxed{b} a c \boxed{a} \boxed{b} a b$
0 0 1 0 1 2 3 2
↑

Когда стоим тут, то значение предикс срдчкодл
может увеличиваться не больше чем на 1
по сравнению предыдущим шагом.
т.к. добавили 1 новую букву.
Можем проверить, т.к. $S[P[i-1]] = S[i]$
когда можно искать +1 элемент.

Но скорее всего так будет редко случаться
и сложность все равно $O(n^3)$

Оптимизация 2

0 1 2 3 4 5
 $\underline{a} \underline{b} a \underline{b} a b$
0 0 1 2 3

6 7 8 9 10 11
 $a \underline{b} a b a$
..... 5 1

1. $i=10$, хотим узнать $P_{i+1} = P_1$

$$S[i+\beta] \neq S[p_i]$$

2. Ищем самое короткое префикс:

^{непр.} \underline{ababa} $k=5$ $S[5] \neq S[i+1]$

\underline{abq} $k=3$ $S[3] \neq S[i+1]$

\underline{q} $k=1$ $S[1] \neq S[i+1]$

$\hookrightarrow k=0$ $S[0] = S[i+1]$ $P = k+1 = 1$

$$P_0 = 0$$

$$P_{i+1} = P_i + 1, \text{ если } S[i+1] = S[p_i]$$

иначе

$$k=p_i$$

если $S[x] = S[i+1]$: $P_{i+1} = k+1$
иначе $k=p_{k-1}$

Сложность:

Это каждого действия мы делаем либо +1

либо какое-то кон-во действий не больше, чем

узнать насколько в префиксе различий. В примере 5.

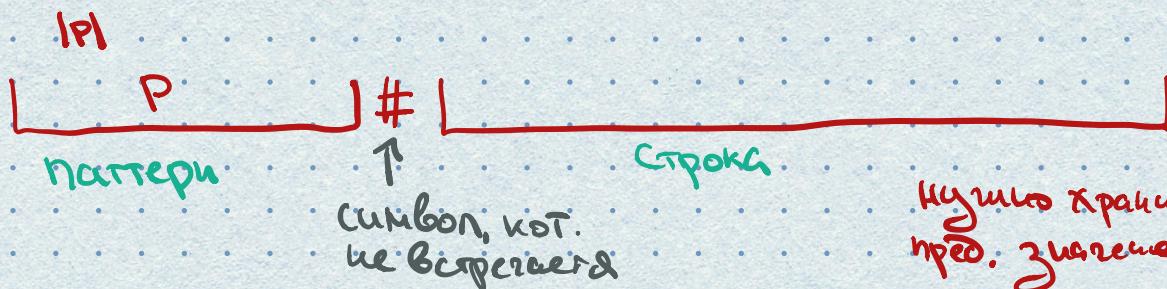
Максимальное значение префикса различий - $n-1$

и на каждом шаге мы совершают какое-то

кон-во действий ≤ разнице в стеках.

$O(n)$

Использование:



Нужно хранить только пред. значение префикса различий. Такие мы уже будем работать с паттерном.

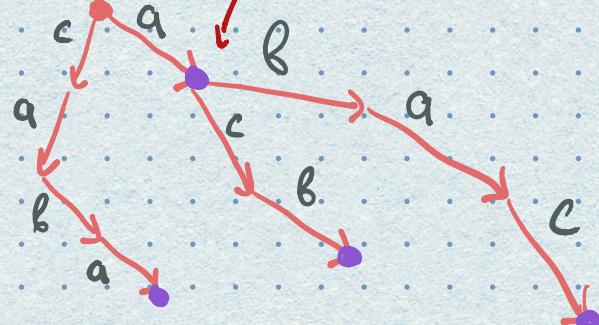
Aho - Корасик

Алгоритм для поиска вхождений множества строк в текст.

Словарь слов:

a
abac
acb
cabca

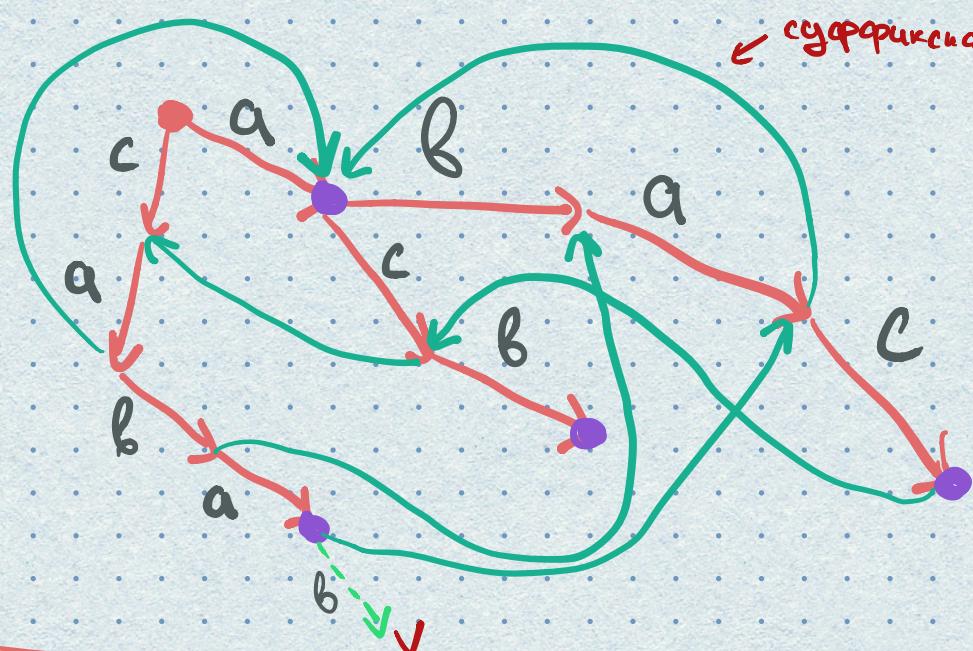
Бор: TRIE
терм. вершина



Если хотим проверить, есть ли в множестве слова, идем по дереву. И если путь заканчивается в терм. вершине, то слово есть.

+ Погрешает мало памяти

суперфиксная ссылка.



* где нет стрелок - они идут в корень

суперфиксная ссылка для вершины - это суперфикс которой идет вверх.

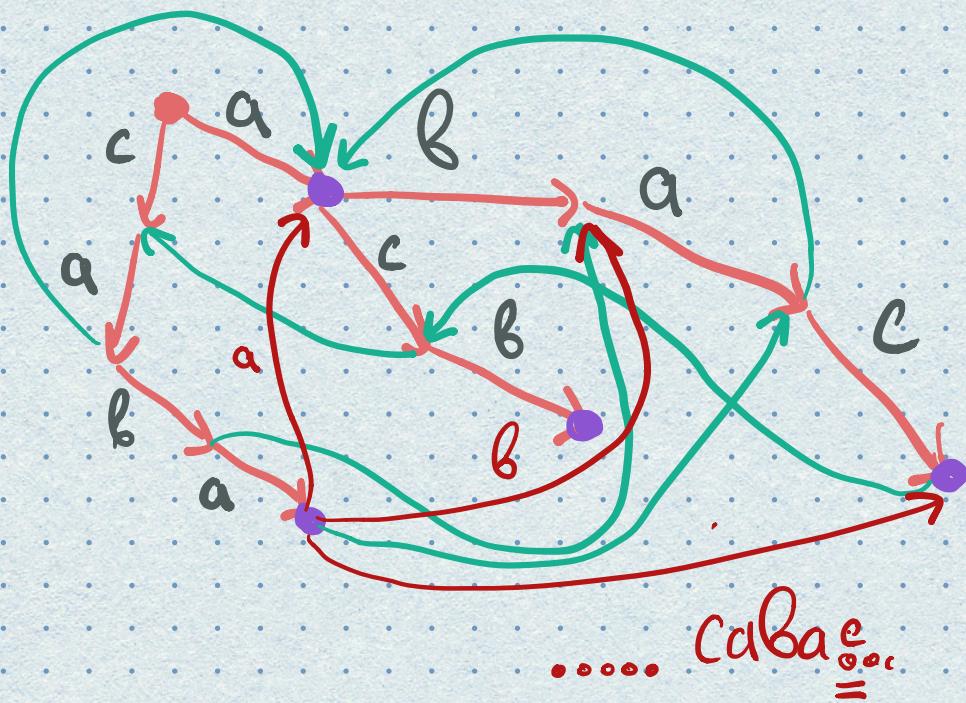
Построение:

Хотим для S найти супр. ссылку.

1. Смотрим для предка: она идет в $a \cdot b$
2. Пытаемся продолжить, но после $a \cdot b$ нет ребра b
3. Переходим дальше по ссылке, идет в a .
4. У a уже есть предок b поэтому ссылка из $c \cdot a \cdot b$ будет вести в $a \cdot b$.
- 5.* Если бы было $c \cdot b \cdot a$, то на пред. шаге мы бы не начали продолжение с поиска b в корне. а из корня уже ехал ребро a . и ссылка должна была идти в a .

Сложноса.

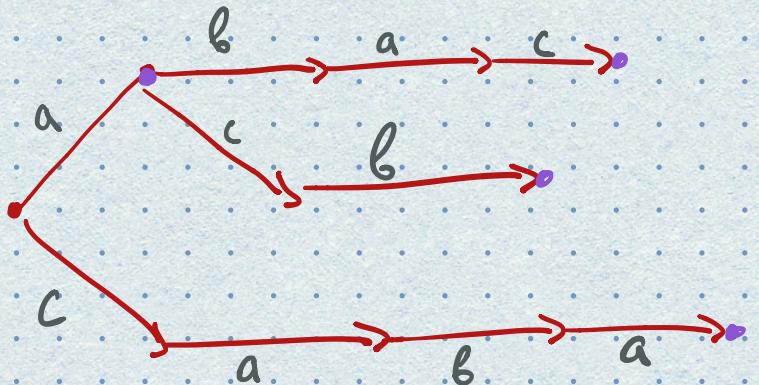
1. Поиск супрор. ссылки для 1 слова не больше двух слов.



..... Слово

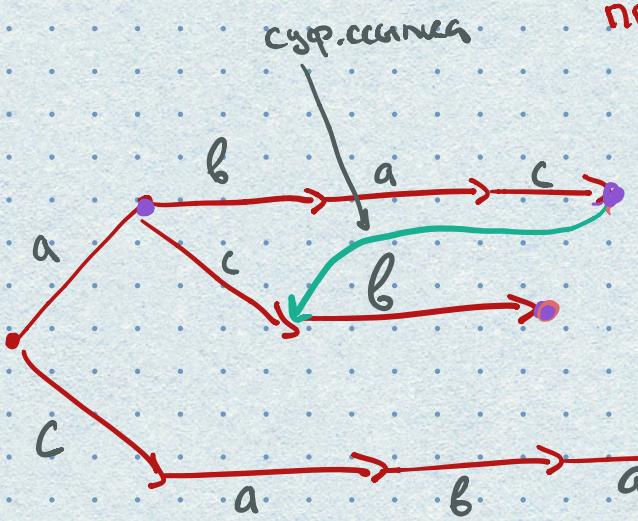
сохраняет красные, чтобы
данные были еще искать переходы.

Axo-Kорасик



a
авас
асб
сава

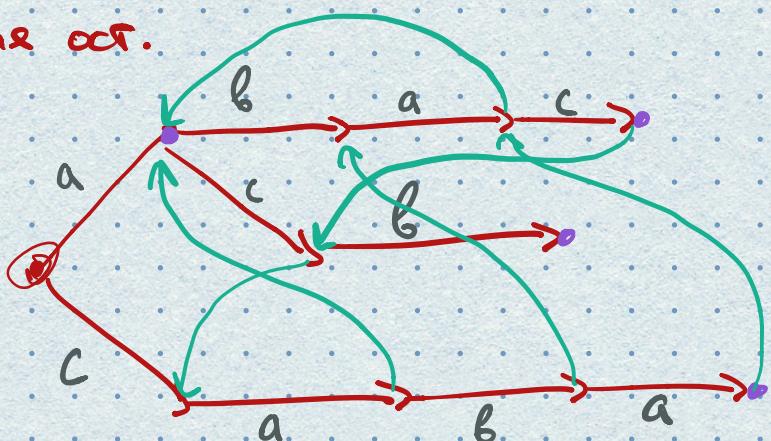
Супр. ссылки:



пример текст
авасв
для авас наименовн.
далее прости не помен.
не хотим начинать перебор
с васв т.к. помен
лишние шаги убрать:
перейдем по суп.ссылке
и дальше по ребру в
и у нас сава сменила
асв всего за
1 действие.

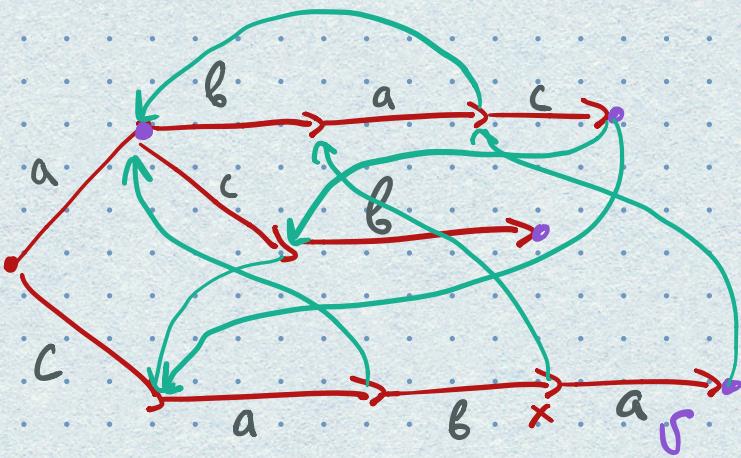
Какие супр. у авас? вас ас с, найдем их бтвк.

Аналогично для ост.



Внедрение:

1. Хотим использовать рефлекс:



$$v = \text{suf}(\text{parent}(v))$$

$v = \text{suf}(k)$ новое суффиксное дерево.

$$\text{suf}(v) = \text{trie}(k, c)$$

Хотим найти суфф. дер аба

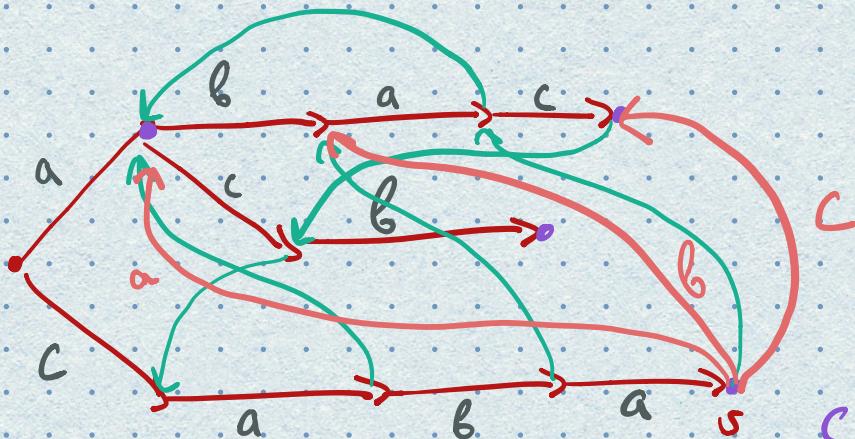
1. смотрю суфф дер аба. — он есть в a
2. пытаюсь суффикса по c — нет.
3. суфф. ac .

Не будем менять:

acb — скажи беда в копене.

$\text{go}(v, c)$ — автоматизирован

тако $\text{trie}(s, c)$ тако $\text{trie}(\text{suf}(v), c)$, тако
 $\text{trie}(\text{suf}(\text{suf}(v)), c)$



сабав
сабаа

пришло сабаа

дальнее пытаемся перейти по суфф.

из ава не можем. идем в а из а не можем.
идем в корень. из него можем.

сабав

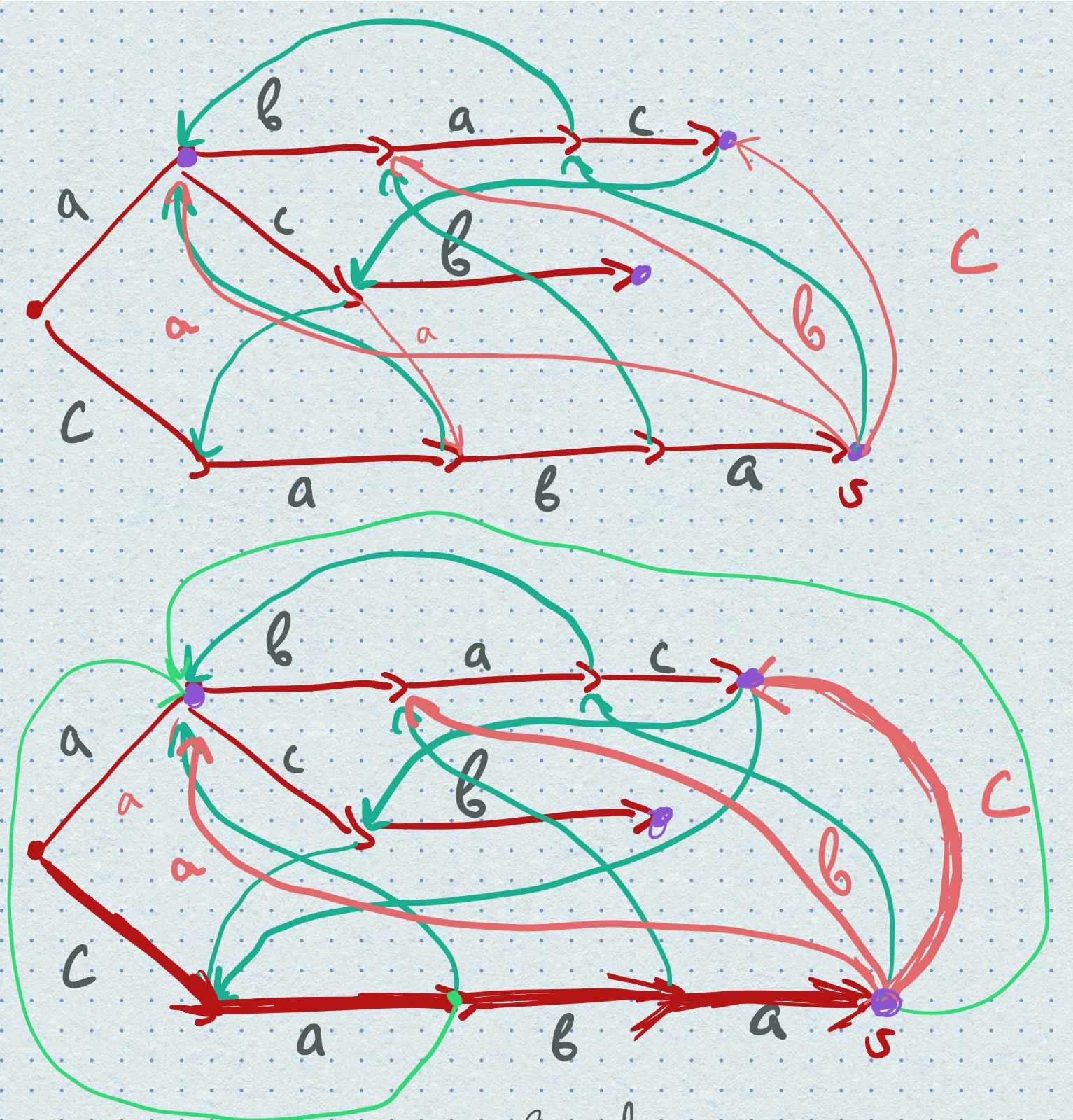
```

find_suf(v, parent_c): // из BFS
    k = v.parent->suf
    while k != root and !k->trie[parent_c]:
        k = k->suf
    if k->trie[parent_c]:
        k = k->trie[parent_c]
    v.suf = k

build_automaton(v, c): // BFS или DFS
    k = &v
    while k != root and !k->trie[c]:
        k = k->suf
    if k->trie[c]:
        k = k->trie[c]
    v.go[c] = k

```

- $\text{go}(v, c)$ — лучшее совпадение для суффикса $\text{str}(v) + c$
- $\text{suf}(v)$ — лучшее совпадение для суффикса $\text{str}(v)$
- $\text{suf}(v) = \text{go}(\text{suf}(\text{parent}(v)), \text{parent}_c)$
- $\text{go}(v, c) = \text{trie}(v, c)$ или $\text{go}(\text{suf}(v), c)$



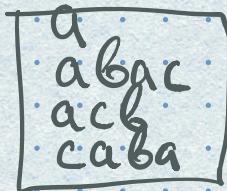
cabac

cab bac

Нашли: cab, abac, но не нашли a

сава
авас

а а



Сложность:

Построив все переходы: $O(\sum |P_i| \times A)$

суфф. и терм. ссылки: $O(\sum |P_i|)$ длина шаблона

Матчинг текста: $O(|T| + |Result|)$

т.к. кон-бо переходов по генерализованной ссылке

размер алфавита

Строим автомат:

- $O(\sum |P_i|)$ на один BFS для суффиксных и терминальных ссылок
- $O(\sum |P_i| \times |A|)$ для автоматных переходов, но можно лениво

Принимаем текст:

- $O(|T|)$, если бы не терминальные ссылки
- Крайний случай: $T = \text{аа...а}, P = \{\text{а, аа, ааа, ...}\}$
- Получается асимптотика $O(|T| + |Result|)$