Алгоритм Бойера-Мура

Алгоритм (R.S. Boyer, J.S. Moore, 1977)

Алгоритм Бойера-Мура - эвристический алгоритм, являющийся на практике одним из наиболее эффективных алгоритмов поиска подстроки P в строке S.

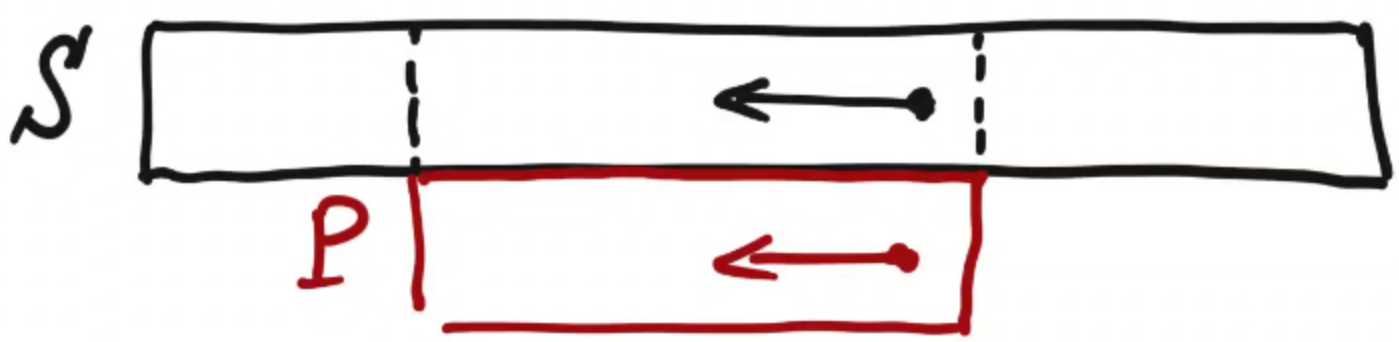
Сложность алгоритма не фиксирована, в зависимости от входных данных может варьироваться от $|S|/|P|$ до $|S| \cdot |P|$. Но на реальных и случайных данных (то есть если нарочно не подбирать плохие строки) в среднем работает быстрее рассмотренных в курсе линейных алгоритмов.

На сегодняшний день существует множество модификаций для различных постановок и ограничений, что обуславливает широкую популярность и применимость.

Идея алгоритма

Будем действовать наивной стратегией - "прикладывать" образец P к различным позициям в строке S и сравнивать элементы последовательно.

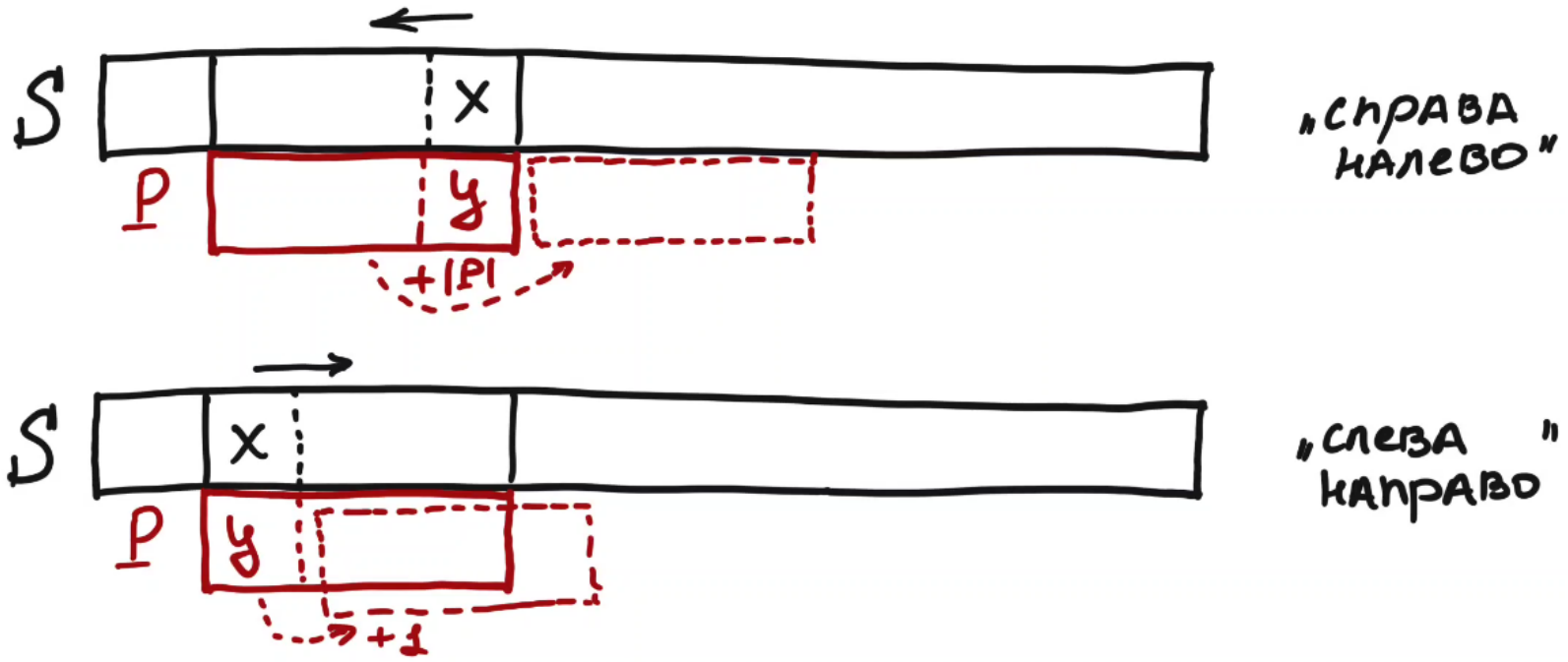
Но! Будем идти не слева направо, а справа налево.



Идея алгоритма

Будем идти не слева направо, а справа налево.

Чем это поможет? Допустим, произошло несоответствие на одном из первых символов. Причем этот символ отсутствует в P. Тогда можем сдвинуть P целиком на P шагов вперед.

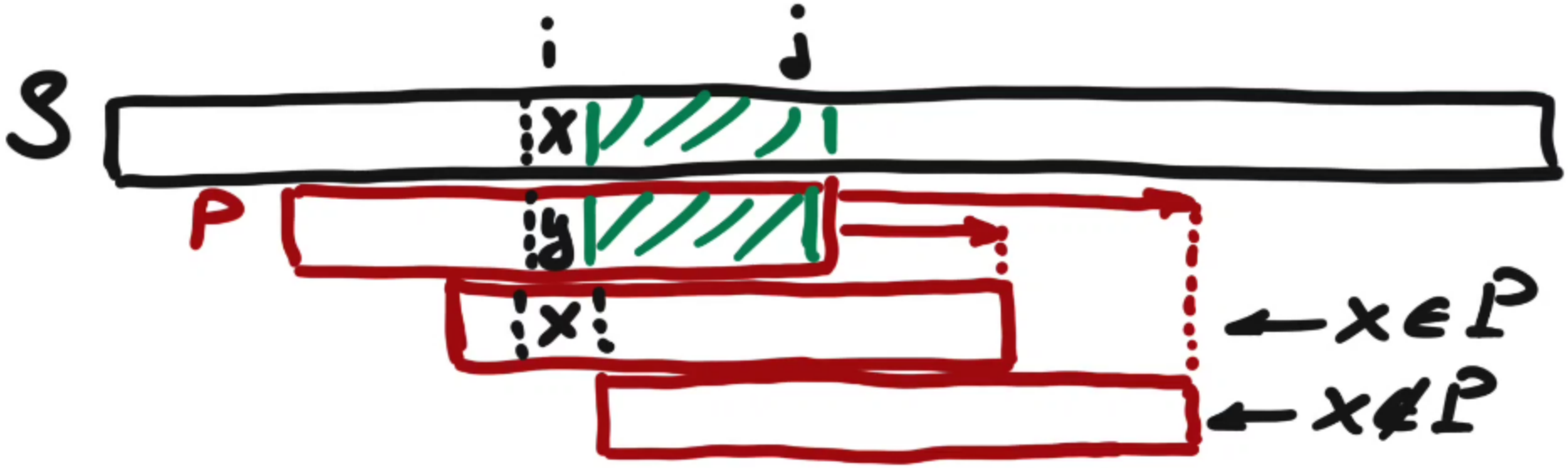


Видно, что сложность алгоритма $\Omega(|S|/|P|)$

Эвристика 1: сдвиг плохого символа

Пусть часть строки $S[i+1 \ldots j]$ совпала с суффиксом строки P,
а $S[i] = x \neq P[|P| - 1 - (j - i)] = y$ (расхождение в следующем символе).

Тогда строку P можно сдвинуть вправо так, чтобы самое правое вхождение x в P совпало с элементом $S[i]$. Если $x \notin P$, то сдвигаем начало P до позиции $i + 1$.



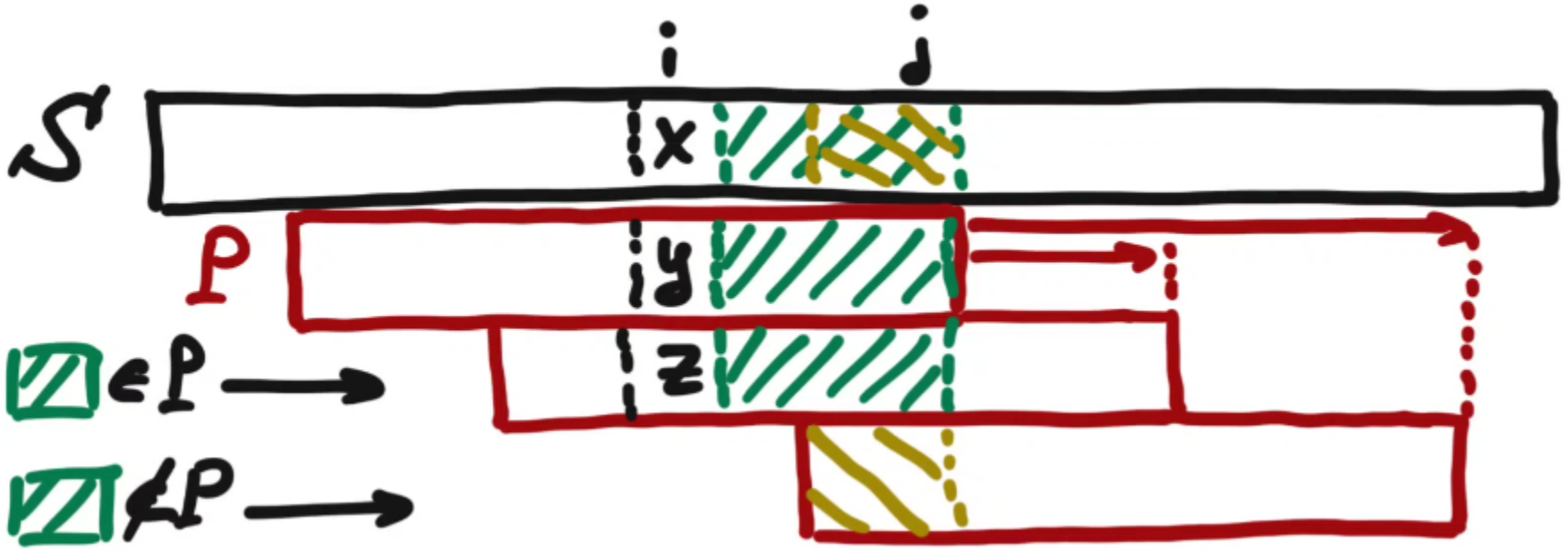
Эвристика 1: сдвиг плохого символа

```
def CalculateBadSymbol(P):  # подсчет последней позиции для каждой буквы
    bad_symbol = {letter : |P| for letter in alphabet}
    for i in range(|P| - 1):  # последний символ никогда не даст сдвиг
        bad_symbol[P[i]] = i
    return bad_symbol
```

```
def BoyerMoore(S, P):
    bad_symbol = CalculateBadSymbol(P)
    j = |P| - 1  # позиция начала поиска (справа налево)
    while j < |S|:
        i = j  # текущая позиция в S
        k = |P| - 1 # текущая позиция в P
        while P[k] == S[i]:
            if k == 0:
                yield i  # нашли полное вхождение
                break
            i -= 1; k -= 1
        j += max(k - bad_symbol[S[i]], 1)  # отрицательные сдвиги ни к чему
```

Эвристика 2: сдвиг хорошего суффикса

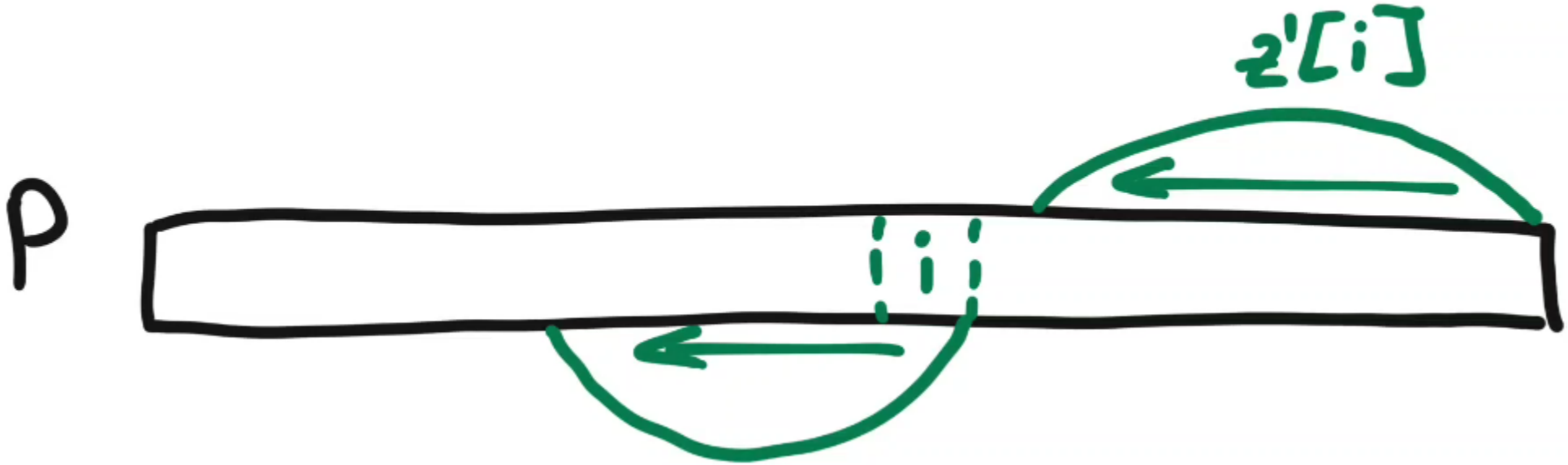
Можно действовать другой стратегией: знаем, что суффикс P совпал с частью строки S, но на очередном символе случился провал. Давайте найдем подстроку в P, которая совпадает с этим суффиксом и сдвинем P до нее вправо.



Эвристика 2: сдвиг хорошего суффикса

То есть для каждого суффикса P необходимо найти его самое правое вхождение (не совпадающее с ним).

Это можно сделать эффективно за линию с помощью предпосчета, например, z-функции для перевернутой строки P (детали обсуждать не будем).



Итог

```
def BoyerMoore(S, P):
    bad_symbol = CalculateBadSymbol(P)
    good_suffix = CalculateGoodSuffix(P)
    j = |P| - 1  # позиция начала поиска (справа налево)
    while j < |S|:
        i = j  # текущая позиция в S
        k = |P| - 1 # текущая позиция в P
        while P[k] == S[i]:
            if k == 0:
                yield i  # нашли полное вхождение
                break
            i -= 1; k -= 1
        j += max(k - bad_symbol[S[i]],
                 k - good_suffix[k])
```