Алгоритмы поиска подстроки в строке

1. «Наивный» алгоритм

о	б	а		о	б	о	б	р	а	л	и		о	б	о	и		б	о	б	р	а
о	б	о	и																			

Число сравнений символов:

3 + 1 + 1 + 1 + 4 + 1 + 3 + 1 + 1 + 1 + 1 + 1 + 1 + 4 = 24

```
public static int simpleSearch(String where, String what) {
    int n = where.length();
    int m = what.length();
    extLoop:    // Внешний цикл поиска в исходной строке
    for (int i = 0; i <= n-m; i++) {
        // Внутренний цикл сравнения:
        for (int j = 0; j < m; j++) {
            if (where.charAt(i+j) != what.charAt(j))
                continue extLoop;
        }
        return i;
    }
    return -1;
}
```

Худший случай: `simpleSearch("aaaaaaaaaaaaaaaaaaaaaaaaaaaaab", "aaaaaaab");`

2. Алгоритм Рабина – Карпа

2	3	2	3	3	2	4	3	2	3	1	5	3	3	2	4	2	3	3	2	2	5	1
3	2	4	2																			

Функция: $\sum_{i=1}^{m} s_i$ = 11

Число сравнений символов:

0 + 3 + 0 + 0 + 0 + 1 + 0 + 0 + 1 + 0 + 0 + 0 + 0 + 4 = 9

Значения функции на подстроках:

10 11 10 12 12 11 12 9 11 12 12 13 12 11

```
public static int RabinKarp(String where, String what) {
    int n = where.length();   // Длина строки, в которой происходит поиск
    int m = what.length();    // Длина подстроки
    long h = 1;               // Вычисляемый числовой показатель вытесняемой буквы
    for (int k = 1; k <= m-1; k++) {
        h <<= 8;
        h %= q;
    }

    long p = 0;     // Числовой показатель подстроки - вычисляется один раз
    long t = 0;     // Изменяемый числовой показатель участка исходной строки
    for (int k = 0; k < m; k++) {
        p = ((p << 8) | (byte) what.charAt(k)) % q;
        t = ((t << 8) | (byte) where.charAt(k)) % q;
    }

    // Внешний цикл по исходной строке
    extLoop:
    for (int i = 0; i <= n-m; i++) {
        if (p == t) {
            // Показатели строк совпали; проверяем, не холостое ли это срабатывание
            for (int j = 0; j < m; j++) {
                if (where.charAt(i+j) != what.charAt(j)) {
                    // символы не совпали - продолжаем поиск
                    continue extLoop;
                }
            }
            // подстрока найдена!
            return i;
        } else if (i < n-m) {
            // сдвиг по исходной строке
            t = (((t - h * (byte) where.charAt(i)) << 8) | (byte) where.charAt(i+m)) % q;
        }
    }
    return -1;
}
```

3. Алгоритм Кнута – Морриса – Пратта

о	б	о	и
0	0	1	0

о	б	а		о	б	о	б	р	а	л	и		о	б	о	и		б	о	б	р	а
о	б	о	и																			

а	б	р	а	к	а	д	а	б	р	а
0	0	0	1	0	1	0	1	2	3	4

```
public static int KnuthMorrisPratt(String where, String what) {
    int n = where.length();   // Длина строки, в которой происходит поиск
    int m = what.length();    // Длина подстроки

    // Формирование таблицы сдвигов
    int[] table = new int[m];
    table[0] = 0;
    int shift = 0;
    for (int q = 1; q < m; q++) {
        while (shift > 0 && what.charAt(shift) != what.charAt(q)) {
            shift = table[shift-1];
        }
        if (what.charAt(shift) == what.charAt(q)) shift++;
        table[q] = shift;
    }

    // Поиск с использованием таблицы сдвигов
    shift = 0;
    for (int i = 0; i < n; i++) {
        while (shift > 0 && what.charAt(shift) != where.charAt(i)) {
            shift = table[shift-1];
        }
        if (what.charAt(shift) == where.charAt(i)) shift++;
        if (shift == m) return i-m+1;   // подстрока найдена
    }
    return -1;    // подстрока не найдена
}
```

4. Алгоритм Бойера - Мура

	а	б	и	л	о	р
4	4	2	4	4	1	4

о	б	а		о	д	о	б	р	и	л	и		о	б	о	и		б	о	б	р	а
о	б	о	и																			

Число сравнений символов:

1 + 1 + 2 + 1 + 1 + 4 = 10

```
private static final int shLen = 256;
private static int hash(char c) { return c & 0xFF; }

public static int BoyerMoore(String where, String what) {
    int n = where.length();    // Длина исходной строки
    int m = what.length();     // Длина образца

    // Формирование массива сдвигов
    int[] shifts = new int[shLen];
    // Для символов, отсутствующих в образце, сдвиг равен длине образца
    for (int i = 0; i < shLen; i++) {
        shifts[i] = m;
    }
    // Для символов из образца сдвиг равен расстоянию от
    // последнего вхождения символа в образец до конца образца
    for (int i = 0; i < m-1; i++) {
        shifts[hash(what.charAt(i))] = m-i-1;
    }

    // Поиск с использованием таблицы сдвигов
    for (int i = 0; i <= n-m; ) {
        // Сравнение начинается с конца образца
        for (int j = m-1; j>=0; j--) {
            if (where.charAt(i+j) == what.charAt(j)) {
                if (j == 0) return i;
            } else {
                break;
            }
        }
        // Сдвиг производится в соответствии с кодом последнего из сравниваемых символов
        i += shifts[hash(where.charAt(i+m-1))];
    }
    return -1;
}
```