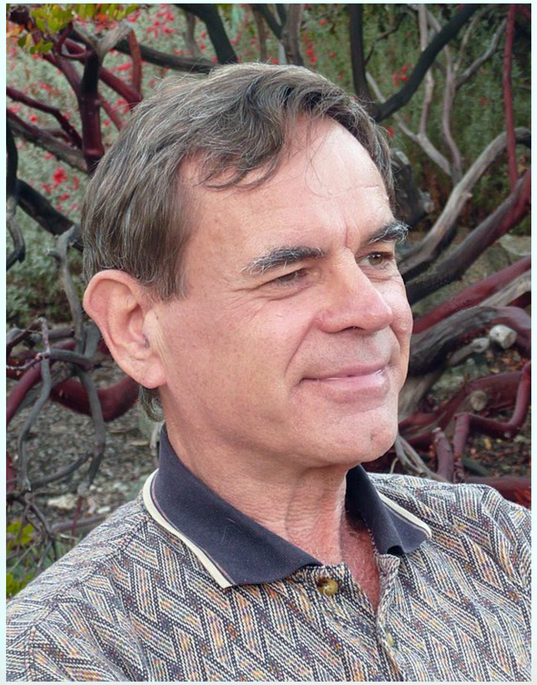
Алгоритм поиска подстроки в строке: Алгоритм Кнута — Морриса — Пратта

Акименко Матвей, студент 24.Б44-мм.

Историческая справка

Кнут, Дональд Эрвин



Пратт, Вон Рональд

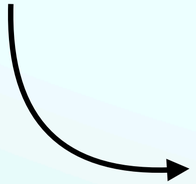
Моррис, Джеймс Хирам

Матиясевич, Юрий Владимирович

Механизм работы КМП

P — подстрока (pattern), которую мы ищем в T (text)

Анализ P — построение массива π (префикс-функции)



Линейный проход по T, базируясь на сдвигах π

Определение префикс-функции

Префикс-функция $\pi[i]$ для позиции i в строке S (или подстроке) представляет собой длину наибольшего собственного префикса, который является суффиксом для подстроки $S[0:i]$.

Технически выполняется за $O(m)$, m — длина S.

Построение префикс-функции

Пример для P = "ABABC":

$P[0:1]$ = "A"	$\pi[0] = 0;$
$P[0:2]$ = "AB"	$\pi[1] = 0;$
$P[0:3]$ = "ABA"	$\pi[2] = 1;$
$P[0:4]$ = "ABAB"	$\pi[3] = 2;$
$P[0:3]$ = "ABABC"	$\pi[4] = 0;$

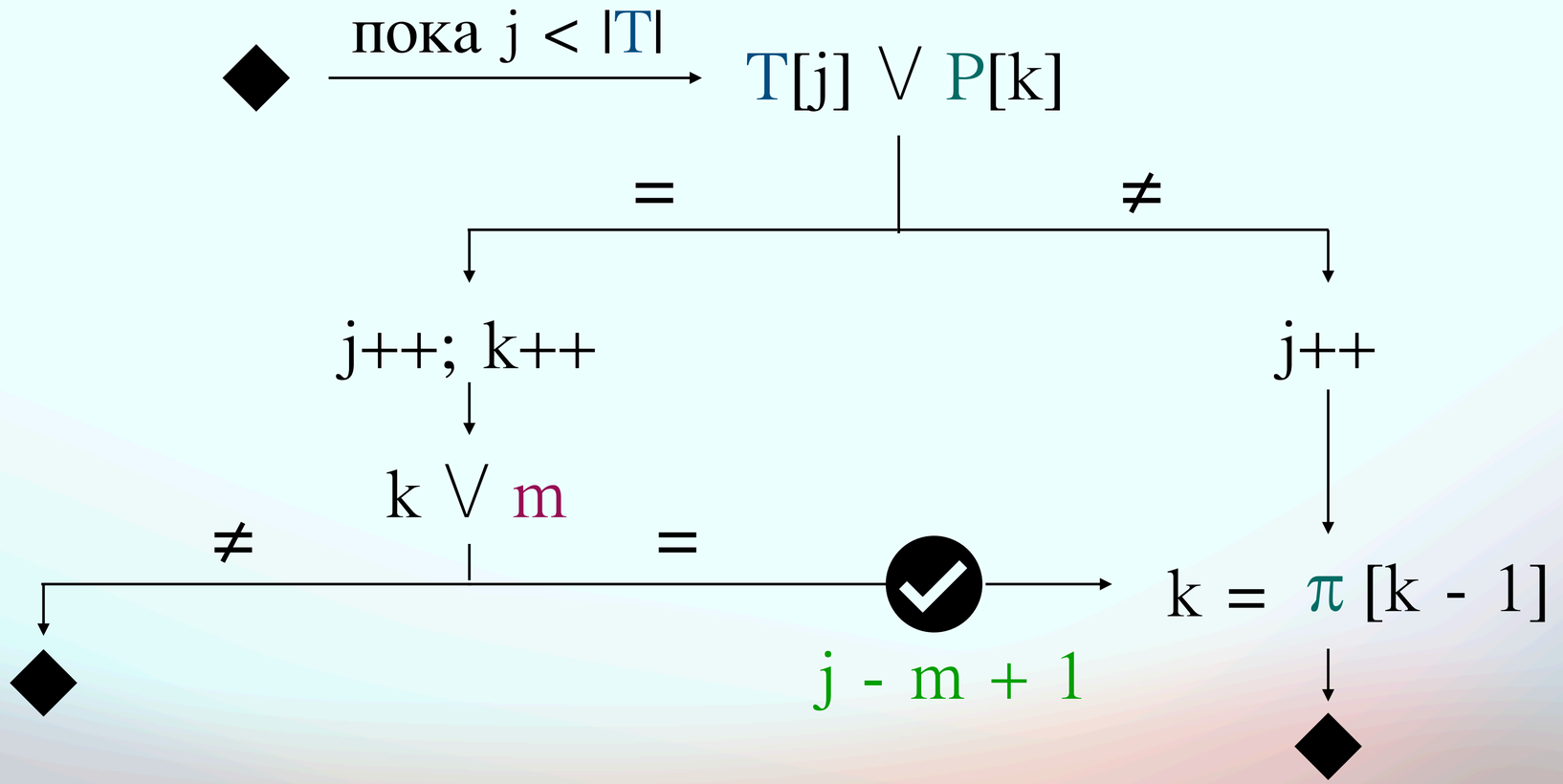
Вариант реализации префикс-функции

```
void computePrefixFunction(const char *pattern, int *prefix) {
    int m = strlen(pattern);
    prefix[0] = 0;
    int k = 0;

    for (int i = 1; i < m; i++) {
        // Если символы не совпадают, сокращаем длину префикса
        while (k > 0 && pattern[k] != pattern[i]) {
            k = prefix[k - 1];
        }

        // Если символы совпадают, увеличиваем длину префикса
        if (pattern[k] == pattern[i]) {
            k++;
        }

        prefix[i] = k;
    }
}
```

Механизм поиска КМП

$j = 0;\ k = 0$

$m = |P| = |\pi|$



Вариант реализации КМП

```c
void KMP(const char *text, const char *pattern) {
    int n = strlen(text);
    int m = strlen(pattern);
    int prefix[m];

    computePrefixFunction(pattern, prefix);
    int k = 0;  // Количество совпавших символов
    for (int i = 0; i < n; i++) {
        // Если символы не совпадают, уменьшаем k с помощью префикс-функции
        while (k > 0 && pattern[k] != text[i]) {
            k = prefix[k - 1];
        }

        // Если символы совпадают, увеличиваем k
        if (pattern[k] == text[i]) {
            k++;
        }

        // Если нашли совпадение шаблона в тексте
        if (k == m) {
            printf("Pattern found at index %d\n", i - m + 1);
            k = prefix[k - 1]; // Продолжаем поиск
        }
    }
}
```

Преимущества

-Сложность: $O(m + n)$, — когда наивный метод: $O(m * n)$

-Избегает избыточных сравнений

Вспомогательные материалы

Публикация Матиясевича (1971)

Публикация Кнута, Морриса и Пратта (1977)

Видео с очень подробным и наглядным объяснением