Алгоритм Shift-And

- Относится к получисленным алгоритмам (как и алг. Карпа-Рабина)
- Аналогично основная цель – не минимизация числа шагов, а эффективность вычислений на каждом шаге
- Вместо посимвольных сравнений – логические операции над числами и сдвиги
- Числа – битовые карты (1 – совпадение подстрок, 0 – различие)
- Приобрел популярность в 90-е годы благодаря работам Р. Бейза-Йетса и Г. Гоннета (Shift-Or – инвертированный смысл 0 и 1)
- Чрезвычайно эффективен, линейное время
- Недостаток – небольшие образцы
- Распространяется на задачи поиска приближенных вхождений

Описание подхода

- Поиск всех вхождений образца P длины m в текст T длины n
- Рассматриваются все префиксы образца $P[0..j]$, $j = 0, 1, \ldots, m-1$
- Булева матрица M размера n x m: строка i матрицы соответствует i-ой позиции текста T, а столбец j – j-ой позиции образца P
- $M[i, j] = 1$ тогда и только тогда, когда префикс $P[0..j]$ входит в текст, заканчиваясь в его позиции i

Пример

P = abra, T = abracadabra

	a	b	r	a
a	1	0	0	0
b	0	1	0	0
r	0	0	1	0
a	1	0	0	1
c	0	0	0	0
a	1	0	0	0
d	0	0	0	0
a	1	0	0	0
b	0	1	0	0
r	0	0	1	0
a	1	0	0	1

Роль матрицы *M*

- Показывает вхождения образца: символ-метка строки матрицы, завершающейся единицей, совпадает с концом вхождения *P* в *T*
- Точнее: $M[i, m\text{-}1] = 1 \Leftrightarrow P$ совпадает с подстрокой текста $T[i\text{-}m\text{+}1..i]$
- Более того, матрица *M* отображает также и все вхождения префиксов образца *P* в текст *T*
- Эффективный алгоритм ее вычисления?

Схема вычисления матрицы *M*

- Последовательно по строкам: для нахождения строки i обращаемся только к строке i-1
- Для $M[i, j] = 1$ (префикс $P[0..j]$ совпал до позиции текста i) необходимо и достаточно выполнения следующих условий:
  a) $M[i\text{-}1, j\text{-}1] = 1$ (совпадение предыдущего префикса до предыдущей позиции)
  b) $P[j] = T[i]$ (совпадение текущего – последнего символа)
- *Замечание*. При $j = 0$ условие *a*) автоматически выполнено для $\forall\ i$ (совпадение пустого префикса), а при $i = 0$, $j > 0$ – не выполнено (совпадение от несуществующей позиции текста)

Вычисление матрицы *M* – условие а)

- Если *m* невелико (≤ 32 или ≤ 64), то каждую строку матрицы можно представить целым числом (битовой картой)
- Цель – вычислять такие строки не поэлементно, а целиком – с помощью логических операций над целыми числами
- По условию *a*) для вычисления в позиции *j* строки матрицы $M[i]$ необходимо использовать позицию j-1 строки $M[i\text{-}1]$
- Решение: сдвинуть $M[i\text{-}1]$ на одну позицию вправо, чтобы результат ($R[i] = M[i\text{-}1] >> 1$) применить к вычислению всей текущей строки $M[i]$
- После сдвига записать 1 в начальную (левую) позицию (см. *Замечание*): $R[i]$ |= ($1 << m\text{-}1$)
- Для вычисления $M[0]$ полагаем $R[0] = 1 << m\text{-}1$

Вычисление матрицы *M* – условие b)

- Для реализации условия *b*) для ∀ символа $x \in A$ подготовить битовую карту $B[x]$ длины *m*, характеризующую присутствие символа в образце
- В $B[x]$ единицы расположены на позициях вхождения *x* в *P*
- Для предыдущего примера массив карт выглядит так:

	a	b	r	a
B[a]	1	0	0	1
B[b]	0	1	0	0
B[r]	0	0	1	0

- Для остальных символов алфавита – нулевые карты
- При большом алфавите (Unicode) возможны проблемы хранения массива вхождений

Формулы для матрицы M

- С учетом изложенного выше, строки матрицы M могут быть вычислены следующим образом (& – соединяет 2 условия):

  $M[0] = 1 << m\text{-}1 \ \& \ B[T[0]]$

  $M[i] = (M[i\text{-}1] >> 1 \mid 1 << m\text{-}1\ ) \ \& \ B[T[i]],\ i = 1, \ldots, n\text{-}1$

- Заметим, что вариация этого алгоритма Shift-Or проще, т.к. не требует записи 1 в старший разряд чисел.
- Далее приводится псевдокод алгоритма Shift-And
- Сложность линейна по n + длина алфавита
- При длинном образце метод также применим, но для строки матрицы используются несколько чисел. Поэтому теоретически алгоритм Shift-And не относится к линейным.

Алгоритм Shift-And - инициализация

```
Shift-And (P, T)
{
  m = strlen (P);  n = strlen (T);
  chBeg = '0'; chEnd = 'z';  // Алфавит: от цифр до букв латиницы
  nA = chEnd - chBeg + 1; // Длина алфавита
  // Подготовка массива вхождений
  for (k = 0; k < nA; ++k) B[k] = 0;
  for (j = 0; j < m; ++j) B[P[j] - chBeg] |= 1 << (m - 1 - j);
  uHigh = 1 << (m-1); // Константа для установки 1 в старший разряд
  M = 0;
```

Алгоритм Shift-And – основные вычисления

```
// Вычисление «строк матрицы» и фиксация вхождений
for (i = 0; i < n; ++i)
{
  M = (M >> 1 | uHigh) & B[T[i] - chBeg];
  if  (M & 1)
  { // Найдено вхождение
    printf ("Вхождение с позиции %d\n", i - m+1);
  }
}
}
```

Задача о неточных совпадениях

- Метод Shift-And может быть распространен на задачу поиска неточных вхождений образца в текст
- На тех же принципах основана известная утилита поиска в текстовых файлах aGrep, реализованная на множестве платформ
- Под неточностью подразумевается допущение небольшого числа отклонений – несовпадений, вставок или пропусков символов.
- Для упрощения задачи рассмотрим лишь разрешенные несовпадения

Описание подхода

- Поиск всех вхождений образца *P* длины *m* в текст *T* длины *n*
- При этом допускается несовпадение до $k < m$ символов
- Рассматриваются все префиксы образца *P*[0..*j*], $j = 0, 1, \ldots, m-1$
- Булева матрица M^k размера $n \times m$: строка *i* матрицы соответствует *i*-ой позиции текста *T*, а столбец *j* – *j*-ой позиции образца P
- $M^k[i, j] = 1$: префикс *P*[0..*j*] входит в текст, заканчиваясь в его позиции *i*, с возможным несовпадением до *k* символов
- Очевидно, что при $k = 0$ получается матрица $M^0 = M$, определенная и использованная ранее

Схема вычисления матрицы M^k

- Для нахождения матрицы M^k последовательно и построчно вычисляются элементы матриц M^l, $l = 0, 1, \ldots, k$
- Для M^l [*i*, *j*] = 1 (префикс *P*[0..*j*] совпал до позиции текста *i* с возможным несовпадением до *k* символов) необходимо и достаточно выполнения одного из двух условий:

  1) (аналогичное) *P*[0..*j*-1] совпадает с текстом до (*i*-1)-ой позиции, с не более чем *l* отклонениями (M^l [*i*-1, *j*-1] = 1), и следующая пара символов совпадает (*P*[*j*] = *T*[*i*]);
  2) (дополнительное) *P*[0..*j*-1] с текстом до (*i*-1)-ой позиции, с не более чем *l*-1 отклонениями (M^{l-1} [*i*-1, *j*-1] = 1), и в этом случае проверка последних символов не требуется

Формулы для матрицы M^k

- В результате получаем следующие рекуррентные формулы для вычисления строк матриц M^l (массив *B* был определен ранее):

  $M^0[0] = 1 << m\text{-}1 \ \& \ B[T[0]]$; $M^l[0] = 1 << m\text{-}1$, $l = 1, \ldots, k$;

  $M^0[i] = (M^0\ [i\text{-}1] >> 1 \mid 1 << m\text{-}1\ ) \ \& \ B[T[i]]$, $i = 1, \ldots, n\text{-}1$;

  $M^l[i] = M^l\ [i\text{-}1] >> 1 \ \& \ B[T[i]] \mid M^{l-1}\ [i\text{-}1] >> 1 \mid 1 << m\text{-}1$,

  $l = 1, \ldots, k$; $i = 1, \ldots, n\text{-}1$.

- Конечная цель вычислений заключается в просмотре всех строк матрицы M^k и проверке их младшего разряда
- Далее приводится псевдокод алгоритма Shift-And-Fz
- Сложность оценивается как $\Theta(nk$ + длина алфавита)

Алгоритм Shift-And-Fz - инициализация

```
Shift-And-Fz (P, T, k)
{
   m = strlen (P);  n = strlen (T);
   // Алфавит: например, от цифр до букв латиницы
   chBeg = '0'; chEnd = 'z';  nA = chEnd - chBeg + 1; // Длина алфавита
    // Подготовка массива вхождений символов алфавита
   for (k = 0; k < nA; ++k) B[k] = 0;
   for (j = 0; j < m; ++j) B[P[j] - chBeg] |= 1 << (m-j-1);
   uHigh = 1 << (m-1); // Константа для установки 1 в старший разряд
   // Инициализация строк (M1 можно заменить парой переменных)
   for (l = 0; l <= k; ++l) { M[l] = 0; M1[l] = 0; }
```

Алгоритм Shift-And-Fz – основные вычисления

```
for (i = 0; i < n; ++i)
  { // Вычисление «строк матриц» и фиксация вхождений
    for (l = 0; l <= k; ++l)
    {
      M1[l] = M[l]; // Запомнить (i-1)-ю строку
      M[l] = (M[l] >> 1 | uHigh) & B[T[i] - chBeg];
      if  (l) M[l]  |= (M1[l-1] >> 1 | uHigh); // Используем (i-1)-ю строку
      if  (l == k && M[l] & 1)
      { // Найдено вхождение
        printf ("Вхождение с позиции %d\n", i - m+1);
      }
}}}
```