# ВЫЧИСЛИТЕЛЬНЫЕ МЕТОДЫ В ДИСКРЕТНОЙ МАТЕМАТИКЕ

DOI 10.17223/20710410/24/11

УДК 519.7

## ОДНОВРЕМЕННЫЙ ПОИСК НЕСКОЛЬКИХ ДВОИЧНЫХ ШАБЛОНОВ В ПОТОКЕ С ПОМОЩЬЮ КОНЕЧНОГО АВТОМАТА

И. В. Панкратов

*г. Томск, Россия*

**E-mail:** ivan.pankratov2010@yandex.ru

Рассматривается задача поиска булевых векторов в потоке данных. Предлагается метод построения конечного автомата, который ищет одновременно несколько векторов, совершая только две простые операции на каждый бит или группу битов, например байт данных. При этом с увеличением количества искомых шаблонов объём требуемой памяти растёт медленнее, чем суммарная длина шаблонов, а трудоёмкость не изменяется совсем. Приводятся оценки размеров таблиц переходов и выходов автомата. Рассматриваются известные подходы к решению этой задачи. Есть возможность обобщить алгоритм построения поискового автомата на поиск не полностью определённых булевых векторов, однако в этом случае объём требуемой памяти может превышать найденную в данной работе оценку.

**Ключевые слова:** *поиск битовых последовательностей, синхропосылка, поиск подстроки, КМП-поиск, алгоритм Ахо — Корасик.*

### Введение

Рассматриваемую задачу можно сформулировать так: имеем двоичную последовательность (поток данных) и набор из $n$ булевых векторов (*шаблонов*) $V_1$, $V_2$, …, $V_n$ длин $k_1$, $k_2$, …, $k_n$ соответственно. Необходимо найти вхождения всех шаблонов в последовательность.

Поиск заданных слов в потоке данных применяется в различных анализаторах трафика, для нахождения синхропосылки в шифртексте и в системах беспроводной цифровой связи.

### 1. Известные подходы к задаче

Рассмотрим стандартные подходы к этой задаче. Оценивать их эффективность будем в предположении, что значения битов в потоке равновероятны и независимы друг от друга.

Можно «прикладывать» шаблоны, начиная с каждого бита потока, и побитно сравнивать их с содержимым потока. Оценим количество сравнений для одного шаблона длиной $k$ битов. Первый бит шаблона нужно сравнивать с битом потока каждый раз, второй — только если первый бит совпал, то есть в среднем в половине случаев, и так далее. Всего потребуется $1 + \frac{1}{2} + \frac{1}{4} + \ldots + \frac{1}{2^{k-1}} = 2 - \frac{1}{2^{k-1}} \approx 2$ сравнения на каждый бит потока. Если всего шаблонов $n$, то на каждый бит потока в среднем потребуется $2n$ сравнений.

Другой подход основан на *регистре сдвига*, в который последовательно подаются биты из потока. После подачи каждого бита содержимое регистра сравнивается со всеми шаблонами. Тут потребуется ровно $n$ сравнений, но уже не битовых, а целочисленных, если содержимое регистра помещается в элементарный тип данных. С точки зрения реализации тут есть преимущество: сравнивать значение регистра с шаблоном проще и быстрее, чем значения отдельных битов. С другой стороны, если длина регистра отлична от длины шаблона, нужно обнулять старшие биты регистра перед сравнением, что занимает какое-то время, особенно если искомые шаблоны имеют различные длины. Если же регистр не помещается в элементарный тип данных, эта схема затрудняется ещё сильней.

## 2. Поиск шаблона с помощью конечного автомата

В данной работе предлагается строить конечный автомат [1], на вход которого подаются биты или байты потока, а на выходе получается информация о найденных в потоке шаблонах. На каждый бит или байт потока автомат совершает две простейших операции: изменение состояния по таблице переходов и определение выхода по таблице выходов. При этом автомат осуществляет одновременный поиск произвольного количества шаблонов любой длины, в том числе различной.

К недостаткам такого подхода относится необходимость хранить таблицы переходов и выходов автомата, которые могут оказаться довольно объёмными. Кроме того, при использовании автомата сложно реализовать «мягкий» поиск, допускающий некоторое количество ошибок в потоке.

### 2.1. Поисковый автомат с битовым входом

Сначала определим *автомат с битовым входом*, или *битовый автомат*.

Входной алфавит автомата — $\{0, 1\}$.

Выходной алфавит должен позволять выдавать информацию о позициях и номерах найденных шаблонов, однако для битового автомата достаточно указать лишь номера найденных шаблонов, поскольку за один такт работы автомата каждый шаблон может быть найден лишь один раз и может заканчиваться только в позиции последнего обработанного бита. Следовательно, выходной алфавит можно задать, например, множеством булевых векторов $\{0, 1\}^n$, где каждому биту сопоставлен взаимно однозначно некоторый шаблон из заданного набора, и бит принимает значение 1, если в текущей позиции закончился этот шаблон.

Определим множество состояний. Если автомат ищет один шаблон, то всё, что ему нужно знать, — это информацию о позициях, в которых этот шаблон мог начаться. Если длина шаблона $k$ битов, а на вход автомата подали $i$-й бит, где $i \geqslant k$, то нас уже не интересует, мог ли шаблон начаться в $(i - k)$-м бите или ранее, так как если шаблон начался в $(i - k)$-м бите, то он закончился в $(i - 1)$-м, и автомат уже выдал информацию о его нахождении. Таким образом, нужно знать, могло ли вхождение шаблона начаться в позициях с номерами $i - k + 1, \ldots, i - 1$.

Вместо запоминания позиций возможного начала шаблона в потоке можно хранить длины уже найденных префиксов шаблона. Соответствие между длинами префиксов и позициями такое: шаблон мог начаться в позиции $(i - l)$, если и только если найден его префикс длины $l$ после обработки $(i - 1)$-го бита. Нас интересуют только длины префиксов от 1 до $(k-1)$. Состояние такого автомата можно задать булевым вектором длины $(k - 1)$. Состояние автомата для поиска нескольких шаблонов можно задать набором аналогичных векторов.

Однако такой способ задания состояний избыточен, поскольку некоторые значения векторов бессмысленны. Например, для шаблона 101 невозможно найти одновременно префиксы длины 1 и 2 бита, следовательно, векторы, имеющие единицы в соответствующих позициях, задают невозможные состояния. Возможные состояния автомата можно закодировать целыми числами.

### 2.2. Построение битового автомата

Опишем алгоритм построения битового автомата.

Строится автомат по индукции. Сначала вводится нулевое состояние, соответствующее ситуации, когда не найден ни один префикс ни одного шаблона. Это состояние естественно для начала входной последовательности. Затем алгоритм поочерёдно обрабатывает все состояния условной подачей на вход автомата 0 и 1, попутно сохраняя новые полученные состояния и строя таблицы переходов и выходов автомата. Обработав таким образом все состояния автомата, получим таблицы переходов и выходов, а также таблицу соответствия номеров состояний их описаниям: состоянию $s$ сопоставляется набор $T_s = (T_{s,1}, T_{s,2}, \ldots, T_{s,n})$, где $T_{s,j}$ содержит множество длин найденных префиксов шаблона номер $j$.

Обозначим $l$-й бит вектора $V$ через $V[l]$, биты нумеруем с нуля.

Выходной алфавит автомата — множество булевых векторов длины $n$. В выходном векторе $F$ бит $F[j-1]$ соответствует $j$-му шаблону.

**Алгоритм построения битового поискового автомата**

**Вход:** набор булевых векторов (шаблонов) $V_1$, $V_2$, …, $V_n$, их длины $k_1$, $k_2$, …, $k_n$
**Выход:** функции переходов $\psi$ и выходов $\varphi$ поискового автомата

1. Запоминаем начальное состояние $T_0 := (\varnothing, \varnothing, \ldots \varnothing)$, $s := 0$.
2. Обрабатываем состояние $s$ условной подачей нуля и единицы. Подаваемый бит обозначим $b$. Сначала зададим $b := 0$.
3. Строим новое состояние $T$, в которое автомат должен перейти после подачи бита $b$ в состоянии $T_s$, и соответствующий выходной символ — вектор $F$; сначала полагаем $F := 00\ldots0 = 0^n$.
   **Для каждого** $j$ от 1 до $n$:
   3.1. Зададим $A := T_{s,j} \cup \{0\}$, $B_j := \varnothing$.
   3.2. **Для всех** $l \in A$ **если** $b = V_j[l]$, **то** $B_j := B_j \cup \{l+1\}$.
   Теперь множество $B_j$ содержит длины всех найденных префиксов вектора $V_j$ после подачи бита $b$ в состоянии $T_s$.
   3.3. **Если** $k_j \in B_j$,**то** вектор $V_j$ найден; полагаем $B_j := B_j \backslash \{l+1\}$, $F[j-1] := 1$.
4. Получили набор $T := (B_1, B_2, \ldots, B_n)$, описывающий следующее состояние, и выходной символ автомата — вектор $F$. Ищем набор $T$ среди имеющихся наборов $T_0, \ldots, T_s$.
   **Если** нашли, что $T = T_h$, **то** присваиваем $s' := h$;
   **если** такого состояния ещё нет, **то** добавляем его в таблицу в новую ячейку. Пусть номер этой ячейки $s'$. Тогда состояние $T_{s'} = T$.
5. Запоминаем $\psi(s, b) := s'$, $\varphi(s, b) := F$.
6. **Если** $b = 0$, **то** $b := 1$ и переход на шаг 3.
7. $s := s + 1$. **Если** в таблице есть состояние $T_s$, **то** переход на шаг 2.
8. Ответ: функции $\psi$ и $\varphi$.

### 2.3. Пример поискового битового автомата

Приведём пример битового автомата для поиска шаблонов 0111 и 1101 (таблица). Построенный автомат имеет семь состояний, в двух из которых возможен ненулевой выходной символ. Два состояния автомата (0 и 2) являются несущественными, то есть в них автомат может находиться только в начале обработки последовательности.

**Таблица переходов и выходов автомата**

| Состояния | | Функция $\psi$ | | Функция $\varphi$ | |
|---|---|---|---|---|---|
| | | 0 | 1 | 0 | 1 |
| $T_0 = (\varnothing, \varnothing)$ | 0 | 1 | 2 | 00 | 00 |
| $T_1 = (\{1\}, \varnothing)$ | 1 | 1 | 3 | 00 | 00 |
| $T_2 = (\varnothing, \{1\})$ | 2 | 1 | 4 | 00 | 00 |
| $T_3 = (\{2\}, \{1\})$ | 3 | 1 | 5 | 00 | 00 |
| $T_4 = (\varnothing, \{1, 2\})$ | 4 | 6 | 4 | 00 | 00 |
| $T_5 = (\{3\}, \{1, 2\})$ | 5 | 6 | 4 | 00 | 10 |
| $T_6 = (\{1\}, \{3\})$ | 6 | 1 | 3 | 00 | 01 |

### 2.4. Построение байтового автомата

Имея автомат, принимающий на вход биты, можно построить автомат, принимающий на вход сразу пачки битов, например байты или полубайты. В любом случае будем называть такой автомат *байтовым*, а его входные векторы — байтами. Множество состояний у него будет таким же, входной алфавит $\{0, 1\}^q$, где $q$ — число битов в байте; выходной алфавит будет чуть сложнее, поскольку в байтовом автомате возможно нахождение сразу нескольких вхождений одного шаблона в различных позициях. От выходного алфавита требуется способность передавать списки найденных шаблонов вместе с позициями. В программной реализации автор использовал целочисленный выходной алфавит и дополнительную таблицу соответствия номера выхода спискам найденных шаблонов.

При наличии битового автомата таблицы переходов и выходов байтового автомата строятся просто: по очереди обрабатываются все состояния автомата и все возможные значения байта. Для каждой пары входного байта и состояния анализ происходит так: битовый автомат устанавливается в соответствующее состояние и на его вход побитно подаётся входной байт. Все выходные сигналы автомата собираются вместе и запоминаются с учётом того, при подаче какого бита был получен выходной сигнал. Новое состояние байтового автомата должно соответствовать состоянию, в которое перешёл битовый автомат после подачи всех битов байта.

## 3. Затраты по памяти

Одним из ограничений применения поисковых автоматов является необходимость хранения таблиц переходов и выходов, которые могут занимать значительные объёмы памяти, поэтому их стоит оценить.

### 3.1. Оценка количества состояний поискового автомата

Состояние автомата соответствует набору длин найденных префиксов искомых шаблонов. Пусть в некотором состоянии $s$ самый длинный найденный префикс имеет длину $l$, и это префикс шаблона $V_j$. Тогда последние $l$ битов, обработанных автоматом, совпадают с префиксом шаблона $V_j$ и, следовательно, присутствие префиксов длины не больше $l$ определено однозначно. Следовательно, любое состояние полностью определяется парой $(j, l)$, соответствующей самому длинному найденному префиксу.

Если имеем всего $n$ векторов с длинами $k_1$, $k_2$, …, $k_n$, то возможно всего $(k_1 - 1)$ пар вида $(1, l)$, $(k_2 - 1)$ пар вида $(2, l)$ и т. д. Помимо описанных, есть ещё нулевое состояние. Получается, что всего возможно не больше $\sum_{j=1}^{n} k_j - n + 1$ состояний.

Автомат, ищущий один шаблон длины $k$, всегда имеет ровно $k$ состояний, то есть оценка достижима. Отметим, что в примере из п. 2.3 эта оценка также достигается.

### 3.2. Оценка объёма таблиц переходов и выходов

В программной реализации таблицы переходов и выходов объединены в одну таблицу, в ячейках которой вместе содержатся номер следующего состояния и индекс выхода. Сами выходы организованы в виде списков пар (номер шаблона, отступ позиции), описывающих найденные автоматом шаблоны, и хранятся в линейном массиве, называемом *массивом выходных сигналов*. Объём этого массива сильно зависит от вида шаблонов.

Минимальный объём массива выходных сигналов байтового автомата можно оценить числом $nq$, поскольку в любом бите байта может начаться любой шаблон. Однако среди выходов автомата могут встречаться не только единичные найденные шаблоны, но и списки нескольких найденных шаблонов, что может существенно увеличить объём массива. Практика показывает, что при достаточно большой длине шаблонов (как минимум, превышающей длину байта) этого обычно не происходит.

Для байтового автомата, полученного из битового автомата в примере из п. 2.3, объём массива выходных сигналов оказался очень значительным, что объясняется возможностью нахождения большого количества различных комбинаций искомых шаблонов после обработки очередного байта. Однако использование байтового автомата для поиска столь коротких шаблонов в любом случае не представляется разумным.

Оценим объём таблицы переходов и выходов. Число строк этой таблицы равно количеству состояний автомата, число столбцов — мощности входного алфавита. Для битового автомата оно равно 2, для байтового — $2^q$.

В системах с восьмибитными байтами получаем 256 столбцов. Если состояние автомата и индекс выхода помещаются в машинное слово, то объём одной ячейки объединённой таблицы переходов и выходов занимает 4 байта, строка байтового автомата — 1024 байта. При большом количестве состояний объём таблиц может оказаться слишком большим для внутреннего высокоскоростного кэша процессора, особенно если речь идёт о специализированных процессорах сетевого оборудования или портативных устройств цифровой передачи данных. В этом случае стоит использовать автомат, получающий на вход половинки байтов, и обрабатывать каждый байт за два этапа. Это снизит скорость обработки приблизительно в 2 раза и немного усложнит реализацию, но уменьшит объём таблицы в 16 раз. Кроме того, уменьшится массив выходных сигналов.

## 4. Возможные обобщения

Описанную методику несложно обобщить на случай поиска неполностью определённых булевых векторов, заданных троичными векторами. Однако в этом случае оценка количества состояний будет отличаться, и, вероятно, в большую сторону. Другое возможное обобщение — это использование отличного от $\{0, 1\}$ алфавита для последовательности и искомых шаблонов.

## 5. Связь с известными алгоритмами

Описанный подход имеет много общего с алгоритмом поиска подстроки в строке Кнута — Морриса — Пратта (КМП) [2]. Оба метода предполагают предварительные вычисления, а затем требуют одну простую операцию на каждый символ входной последовательности.

Фактически битовый автомат для поиска одного шаблона реализует КМП-алгоритм: каждому состоянию поискового автомата можно сопоставить число — максимальную длину найденного префикса шаблона. Это число совпадёт с длиной найденного префикса шаблона в алгоритме КМП; назовём его номером состояния. При совпадении очередного бита шаблона с битом потока в алгоритме КМП длина найденного префикса увеличивается на единицу, как и номер состояния поискового автомата. При несовпадении автомат откатится назад в состояние с номером, соответствующим значению префикс-функции алгоритма КМП.

Однако есть и существенные отличия:

1) Поисковый автомат способен искать сразу несколько шаблонов. Для поиска $n$ шаблонов можно использовать $n$ автоматов, ищущих по одному шаблону, однако, как видно из оценки количества состояний, суммарный объём таблиц этих автоматов неизбежно будет превышать объём автомата, ищущего сразу все $n$ шаблонов. Кроме того, набору из $n$ автоматов на каждый входной символ (бит, полубайт или байт) необходимо производить в $n$ раз больше действий, чем одному автомату.
2) Поисковые автоматы способны принимать символы последовательности не только по одному, но и пачками.

Существует также сходство описываемых в данной работе поисковых автоматов с алгоритмом поиска подстроки в строке Ахо — Корасик [3], который тоже позволяет искать сразу набор шаблонов. Алфавит в этом алгоритме не обязательно двоичный, может быть произвольным. В алгоритме Ахо — Корасик используется конечный автомат, состояния которого образуют дерево. Узлы дерева соответствуют префиксам искомых шаблонов, как и в поисковых автоматах. Функционирование автомата Ахо — Корасик происходит аналогично функционированию описываемых в данной работе автоматов.

В [3] при подаче на вход автомата символа, который не может продолжить префикс, соответствующий текущему узлу, происходит дополнительный переход по так называемой *суффиксной ссылке*, что немного замедляет работу автомата. Вместо таблиц переходов и выходов хранится дерево состояний автомата. Количества состояний автомата Ахо — Корасик и поискового автомата, очевидно, совпадают. Для каждого состояния необходимо хранить всю информацию о соответствующем узле дерева состояний, которая включает набор ссылок на потомков и суффиксную ссылку. В случае двоичного алфавита суммарно будет не более двух ссылок. Дополнительно требуется хранить так называемую *словарную суффиксную ссылку* и признак, является ли узел *словарным* и какому шаблону он соответствует.

Однако вместо этого можно закодировать узлы дерева — они же состояния автомата — и составить таблицы переходов и выходов, получив таким образом тот же самый поисковый автомат, который предлагается в данной работе. Требуемый объём памяти и вычислительная сложность будут совпадать с поисковым автоматом.

В данной работе предлагается прямой алгоритм построения таблиц переходов и выходов без использования дерева состояний, а также автомат, принимающий на вход

сразу пачки битов или байты и позволяющий искать неполностью определённые булевы векторы.

## Заключение

Предлагаемый поисковый автомат позволяет с очень высокой эффективностью одновременно искать в потоке данных несколько булевых векторов (шаблонов) некоторых длин. При этом с увеличением количества шаблонов объём требуемой памяти растёт медленнее, чем их суммарная длина, а трудоёмкость не изменяется совсем.

Для увеличения скорости обработки потока данных можно подавать биты потока на вход автомата пачками или байтами. При подаче битов пачками по $q$ битов скорость обработки возрастёт в $q$ раз, а объём требуемой памяти — в $2^q$ раз.

Есть возможность обобщить алгоритм построения поискового автомата на поиск неполностью определённых булевых векторов, однако в этом случае объём требуемой памяти может превышать найденную оценку.

## ЛИТЕРАТУРА

1. *Агибалов Г. П., Оранов А. М.* Лекции по теории конечных автоматов. Томск: Изд-во Том. ун-та, 1984. 185 с.
2. *Knuth D. E., Morris J. H. Jr., and Pratt V. R.* Fast pattern matching in strings // SIAM J. Comput. 1977. No. 6(2). P. 323–350.
3. *Aho A. V. and Corasick M. J.* Efficient string matching: An aid to bibliographic search // Commun. ACM. 1975. No. 18(6). P. 333–340.