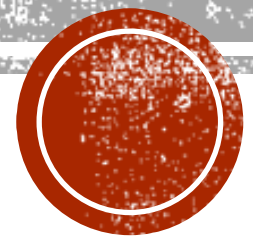
ПОИСК ПОДСТРОК
ЧАСТЬ 2



Префикс функция
ДКА

ВЫЧИСЛЕНИЕ И АНАЛИЗ СЛОЖНОСТИ

```
BuildP(T)

i = 0, j = 0
while i < n:
    if T[i] = P[j]:
        pi[i+1] = j + 1
        i++, j++
    else:
        if j > 0:
            j = pi[j]
        else:
            pi[i+1] = 0
             i++
```

Анализ переменных цикла while:

1. Либо увеличились i и j на единицу
2. Либо уменьшилось j минимум на единицу
3. Либо увеличилось i на единицу

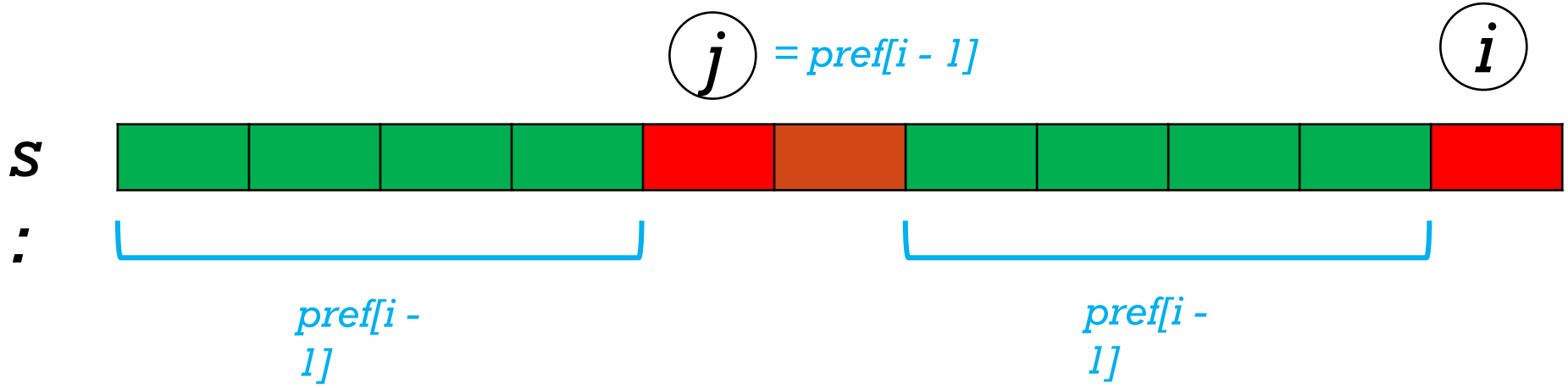
Анализ цикла while на следующем слайде

- Асимптотика: O(n)

ХОТИМ ИЗБАВИТЬСЯ ОТ ЯВНОГО СРАВНЕНИЯ СТРОК

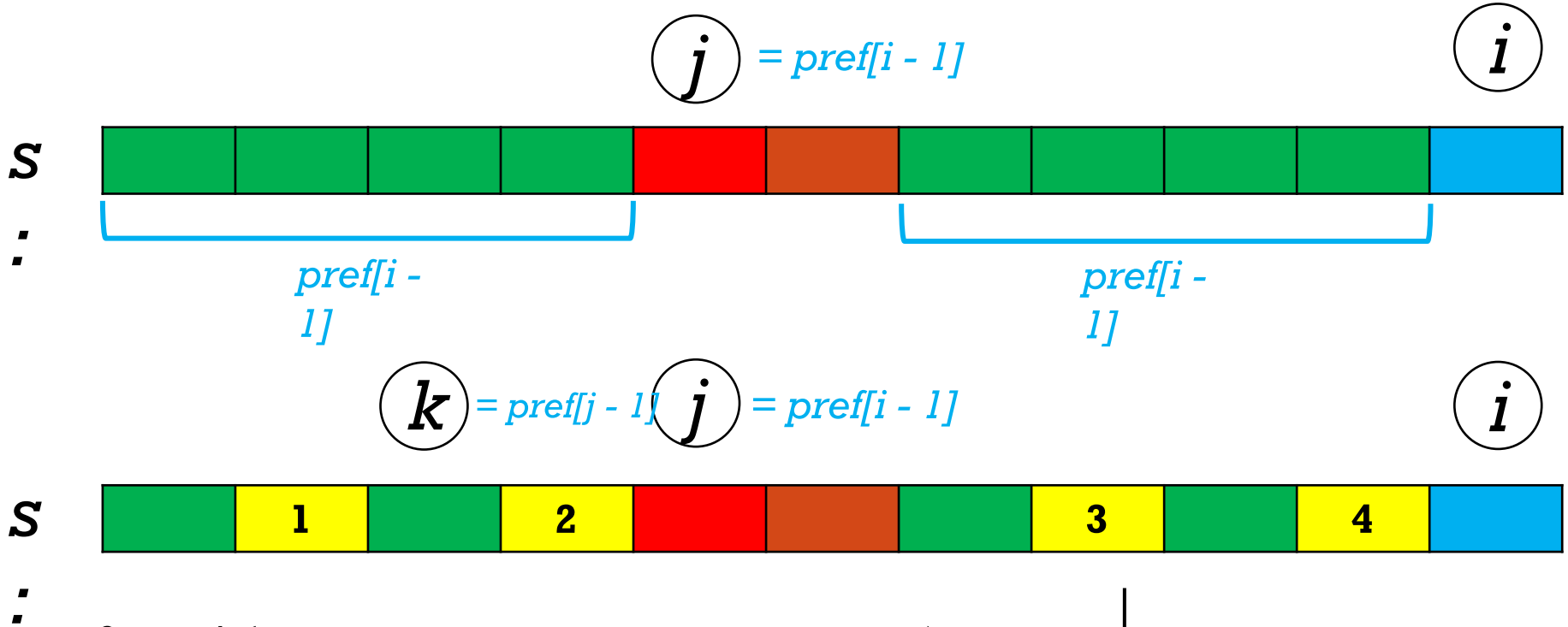
Пользуемся методом ДП и попробуем пересчитать *pref[i]* через предыдущие значения

- Если *s[i] == s[pref[i - 1]]*, то *pref[i] = pref[i - 1] + 1*



- Пусть *s[i] != s[pref[i - 1]]*

Тогда необходимо найти префикс наибольшей длины k такой, что для него выполняется префикс-свойство и *s[k] == s[i]*, либо сказать, что такого нет, и поставить *pref[i] = 0*



2 == 4 (как части одинаковых подстрок)

1 == 2 (потому что является ее преф-функцией)

=> 1 == 4

```
string s;
vector<int> pref(n, 0);
for (int pos = 1 ... n)
{
    int k = pref[pos - 1];
    while (s[pos] != s[k] and k > 0)
    {
        k = pref[k - 1];
    }
    if (s[pos] == s[k])
    {
        pref[pos] = k + 1;
    }
}
```

// перебираем индекс в префикс-функции за O(n)

// сколько-то раз уменьшаемся

// возможно 1 раз прибавляем

ЗА СКОЛЬКО ЭТО РАБОТАЕТ?

```
string s;
vector<int> pref(n, 0);
for (int pos = 1 ... n)
{
    int k = pref[pos - 1];
    while (s[pos] != s[k] and k > 0)
    {
        k = pref[k - 1];
    }
    if (s[pos] == s[k])
    {
        pref[pos] = k + 1;
    }
}
```

За **O(n)**

Рассмотрим другую задачу:

- изначально x = 0
- n действий вида x + 1 - <что-то > 0>
- в каждый момент времени x >= 0

Вопрос: сколько раз мы сможем что-то вычесть?
Ответ: <= n раз

Теперь заметим, что x пересчитывается так же, как значение префикс функции в нашем алгоритме

ВЫЧИСЛЕНИЕ И АНАЛИЗ СЛОЖНОСТИ

- В цикле **while** у нас нет ситуации, при которой бы не изменилось значение **i** или **j**. На каждой итерации цикла у нас как минимум меняется значение одного из итераторов
- Так как **i** не уменьшается, то максимальное количество раз, которое мы можем увеличить **i** равняется **n**
- Из этого следует, что затормозить цикл может только **j**
- Максимально **j** может уменьшаться на длину подстроки. Поэтому самый худший случай – уменьшение **j** на **1**
- Цикл while либо увеличивается **i** на **1**, либо оставляет неизменным столько раз, сколько было до этого шага совпадений, где за количество совпадений и отвечает **j**
- Пусть на какой-то итерации было **k** совпадений, тогда **i** увеличилось на **k**, **j** стало равно **k**
- Так как мы рассматриваем худший случай, то после этой итерации у нас не будет совпадений по префиксу и суффиксу **k** итераций

ВЫЧИСЛЕНИЕ И АНАЛИЗ СЛОЖНОСТИ

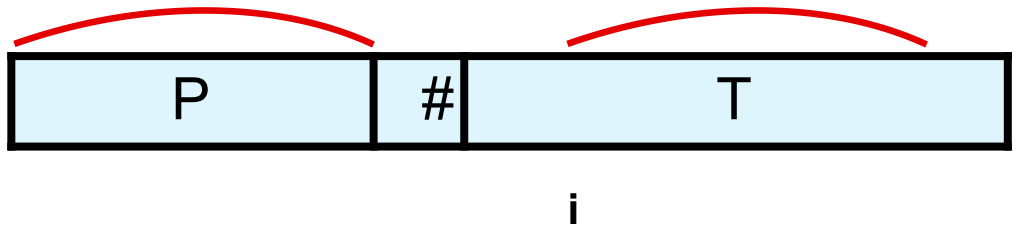
- Тогда **j** уменьшиться на **k**, а **i** останется неизменным
- В результате рассматрения данной подстроки за **k** итераций **i** увеличилось на **k**, затем за итераций **j**
- уменьшилось на **k** и стало равно **0**
- Тогда на данную подстроку ушло **2k** итераций
- Так как **i** у нас ограничено **n**, то максимум **i** увеличиться на **n**, а так как в худшем случае **i** остается неизменным
- столько раз, сколько до этого было совпадений, то **j** уменьшится на это же количество, то есть на длину
- рассматриваемой подстроки => в сумме **j** будет уменьшаться **n** раз
- А значит всего итераций в худшем случае будет $\leq$ **2n** и следовательно сложность данного цикла **O(2n) = O(n)**

АЛГОРИТМ
КНУТА - МОРРИСА - ПРАТТА!

ПОИСК ПОДСТРОКИ В СТРОКЕ

Ищем шаблон P в тексте T

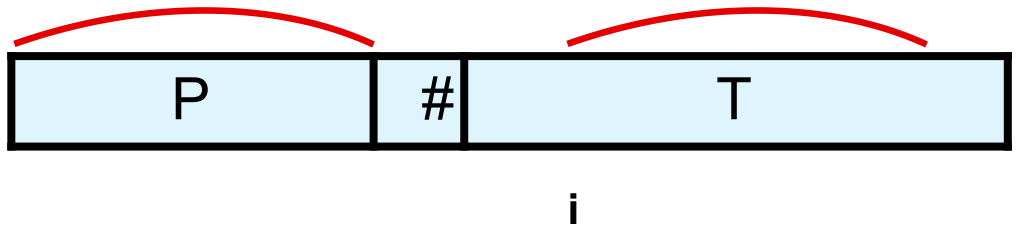
- **префикс**-функция хорошо ищет префиксы в строке
- Вычислим **префикс**-функцию для **P#T**



ПОИСК ПОДСТРОКИ В СТРОКЕ

Ищем шаблон P в тексте T

- **префикс**-функция хорошо ищет префиксы в строке
- Вычислим **префикс**-функцию для **P#T**



(-) сложность по времени O(P+T)

(-) сложность по доп памяти O(P+T)

ОПТИМИЗАЦИЯ ДЛЯ
КНУТА - МОРРИСА - ПРАТТА!

АЛГОРИТМ КНУТА-МОРРИСА-ПРАТТА

```
KMP_Matcher(T, P):

BuildP(P)
i = 0, j = 0
n = T.length, m = P.length
while i < n && j < m:
    if T[i] = P[j]:
        i++, j++
    else:
        if j > 0:
            j = pi[j]
        else:
            i++
if j == m:
    return i - m
else:
    return -1
```

Предподсчет префикс-функции для паттерна
Работает за длину паттерна: O(m)

Нахождение паттерна в тексте
Работает за O(n)

Итоговая асимптотика:
O(n+m)

АЛГОРИТМ КМП, РЕАЛТАЙМ

Иногда текст очень большой, и не хочется тратить |T| памяти. Вспомним, как считали префикс-функцию:

- Поддерживаем k — текущую максимальную длину префикса, совпадающего с суффиксом
- На i-м шаге уменьшаем k по формуле k := pi[k−1], пока не окажется S[k] = S[i]
- Если не повезло, pi[i] = 0. Иначе pi[i] = ++k

- Нам лишь важно отловить i, для которыхpi[i] = |P|
- Достаточно хранить pi лишь для i=0..|P|−1

АЛГОРИТМ КМП, РЕАЛТАЙМ

P	a	b	a	c	a
pi	0	0	1	0	1

T

k = 0

Ответ
:

АЛГОРИТМ КМП, РЕАЛТАЙМ

P	a	b	a	c	a
pi	0	0	1	0	1

T	a

T[0] = P[0]

k = 0 → 1

Ответ:

АЛГОРИТМ КМП, РЕАЛТАЙМ

P	a	b	a	c	a
pi	0	0	1	0	1

T	a	b

T[1] = P[1]

k = 1 → 2

Ответ:

АЛГОРИТМ КМП, РЕАЛТАЙМ

P	a	b	a	c	a
pi	0	0	1	0	1

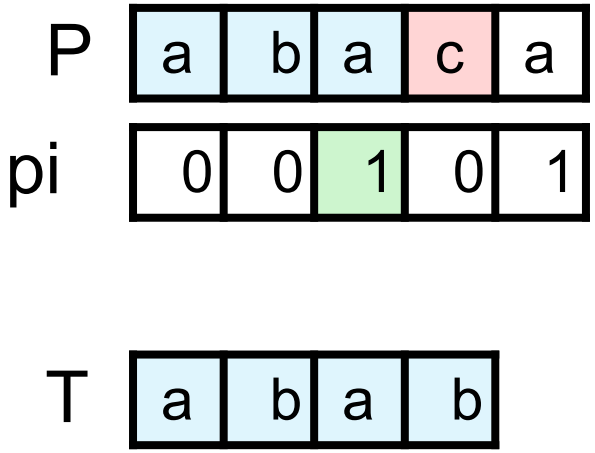
T	a	b	a

T[2] = P[2]

k = 2 → 3

Ответ:

АЛГОРИТМ КМП, РЕАЛТАЙМ

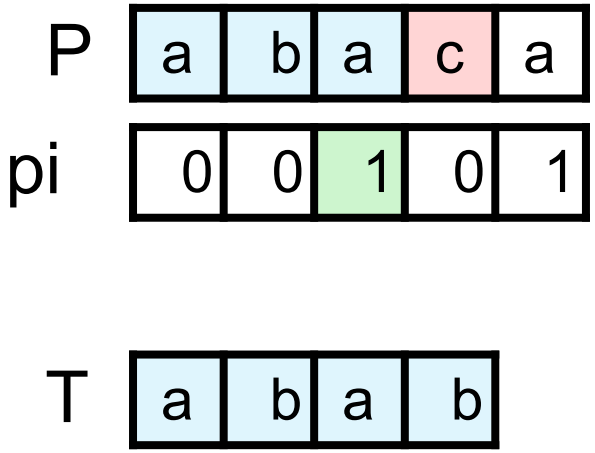


T[3] != P[3]

k = 3 → pi[k−1] = 1

Ответ:

АЛГОРИТМ КМП, РЕАЛТАЙМ



T[3] != P[3]

k = 3 → pi[k−1] = 1

Ответ:

АЛГОРИТМ КМП, РЕАЛТАЙМ

P	a	b	a	c	a
pi	0	0	1	0	1

T	a	b	a	b

T[3] = P[1]

k = 1 → 2

Ответ:

АЛГОРИТМ КМП, РЕАЛТАЙМ

P	a	b	a	c	a
pi	0	0	1	0	1

T	a	b	a	b	a

T[4] = P[2]

k = 2 → 3

Ответ:

АЛГОРИТМ КМП, РЕАЛТАЙМ

P	a	b	a	c	a
pi	0	0	1	0	1

T	a	b	a	b	a	c

T[5] = P[3]

k = 3 → 4

Ответ:

АЛГОРИТМ КМП, РЕАЛТАЙМ

P	a	b	a	c	a
pi	0	0	1	0	1

T	a	b	a	b	a	c	a

T[6] = P[4]

k = 4 → 5 = |P|

Ответ: 6−|P|+1 = 2

АЛГОРИТМ КМП, РЕАЛТАЙМ

P	a	b	a	c	a
pi	0	0	1	0	1

T	a	b	a	b	a	c	a

T[6] = P[4]

k = 4 → 5 = |P|

Ответ: 6−|P|+1 = 2

АЛГОРИТМ КМП, РЕАЛТАЙМ

P	a	b	a	c	a
pi	0	0	1	0	1

T	a	b	a	b	a	c	a

T[6] = P[4]

k = 4 → 5 = |P|

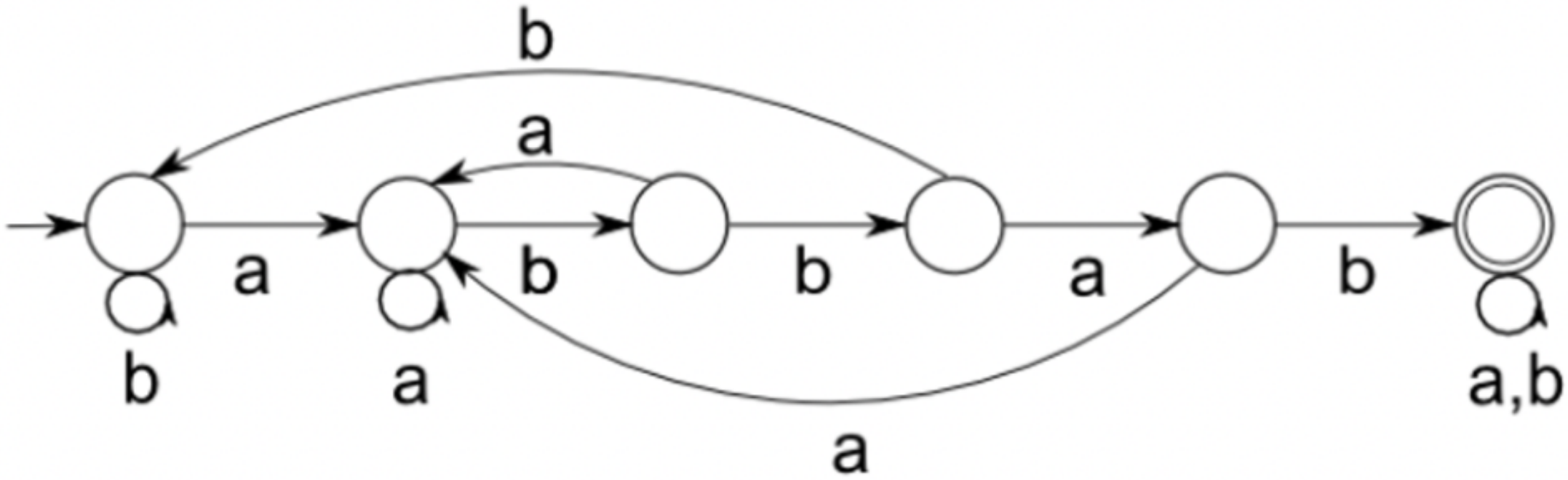
Ответ: 6−|P|+1 = 2

Итого:

- **O(|P|)** памяти, столько же времени на предподсчёт
- **O(|T|)** времени на весь текст
- Может быть **O(|P|)** времени на обработку одного символа

ДКА

ПРИМЕР



ДКА: ОПРЕДЕЛЕНИЕ

- Конечный автомат **M:**
- **Q** - конечное множество состояний
- **q0** ∈ **Q** - начальное состояние
- **T** ⊆ **Q** - множество допускающих состояний
- **Σ** - конечный входной алфавит
- **δ** - функция переходов автомата: **Q** x **Σ** –> **Q**

ДКА: ОПРЕДЕЛЕНИЕ

- Конечный автомат **M:**
- **Q** - конечное множество состояний
- **q0** ∈ **Q** - начальное состояние
- **T** ⊆ **Q** - множество допускающих состояний
- **Σ** - конечный входной алфавит
- **δ** - функция переходов автомата: **Q** x **Σ** –> **Q**

(производит переход из одного состояния в другое засчет перехода по соответствующему символу из **Σ**)

ДКА: ОПРЕДЕЛЕНИЕ

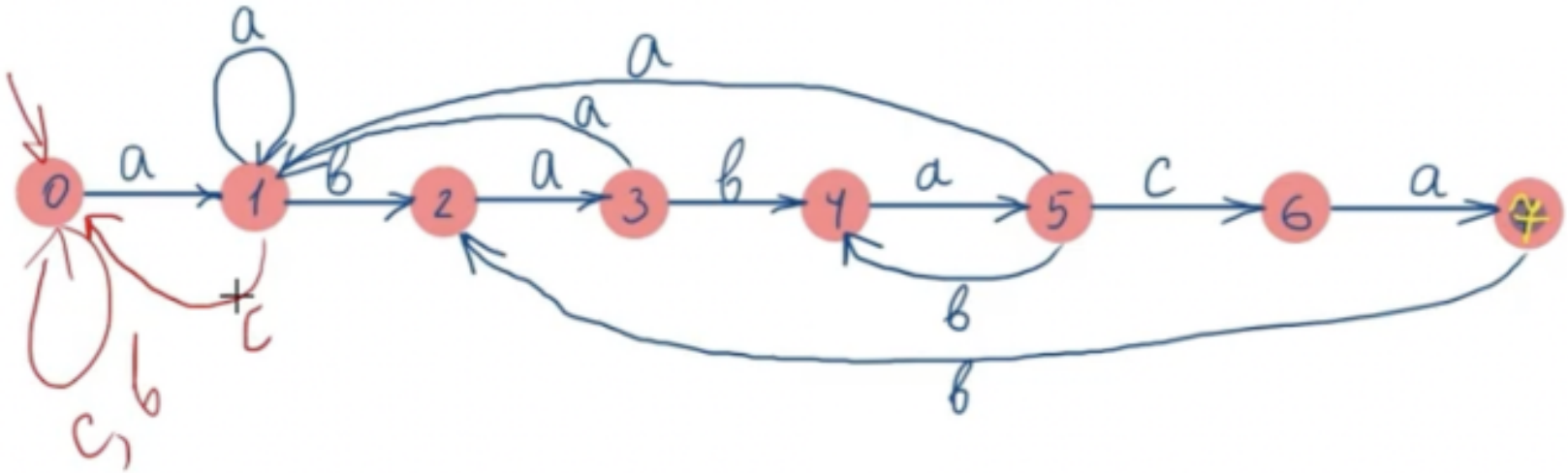
- Конечный автомат **M:**
- **Q** - конечное множество состояний
- **q0** ∈ **Q** - начальное состояние
- **T** ⊆ **Q** - множество допускающих состояний
- **Σ** - конечный входной алфавит
- **δ** - функция переходов автомата: **Q** x **Σ** –> **Q**

(производит переход из одного состояния в другое засчет перехода по соответствующему символу из **Σ**)

- **Как работает:** начинает в **q0** и считывает символы по одному и при попадании в **qi** из **T** - нашли подстроку!

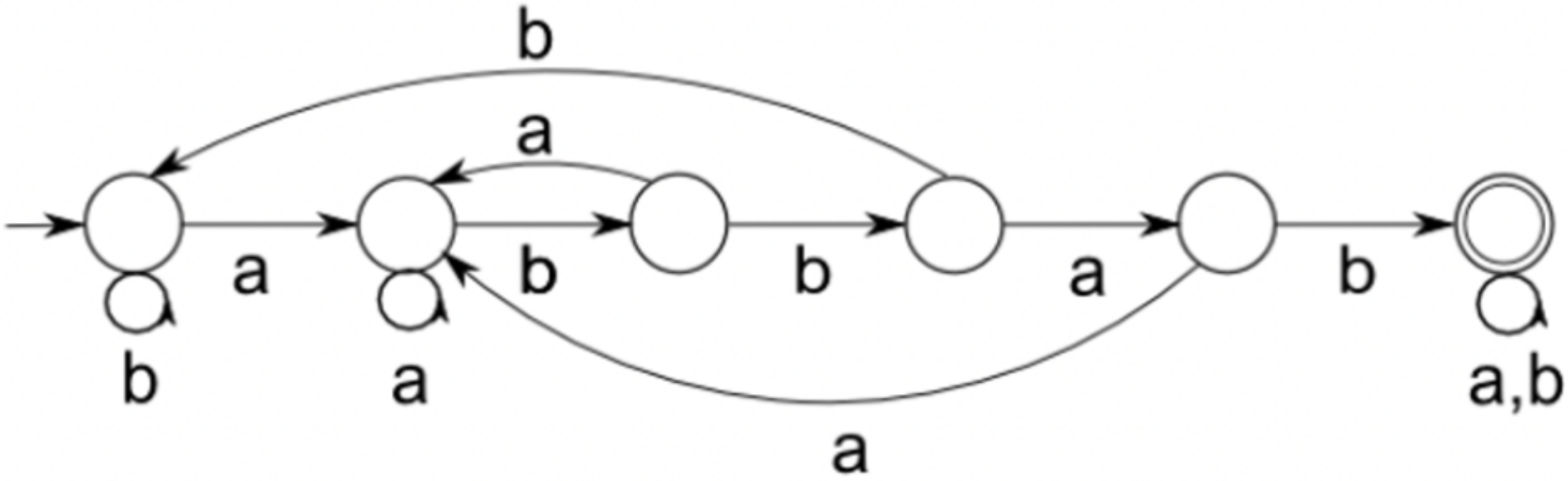
КОНЕЧНЫЙ АВТОМАТ: ОПРЕДЕЛЕНИЕ

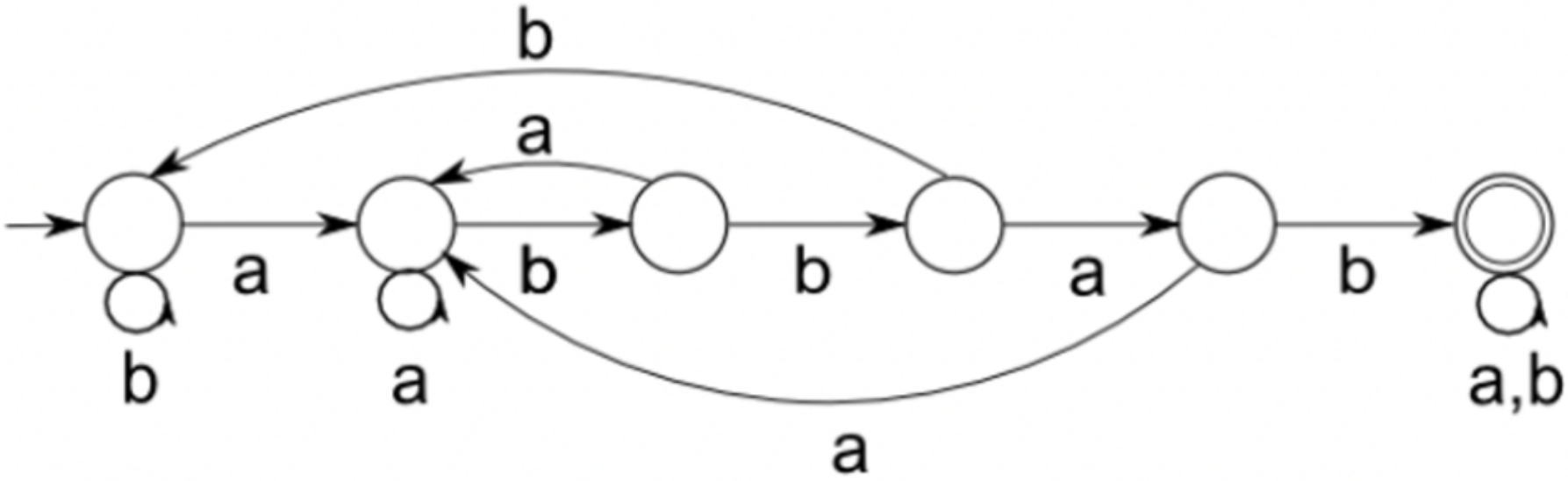
- **ϕ** - функция конечного состояния **Σ*—> Q:** **ϕ(*ω*)** состояние, в котором оказывается автомат **M** после сканирования ***ω*, *ω*** - слово
- **M** принимает ***ω*,** если **ϕ(*ω*)** ∈ **T**

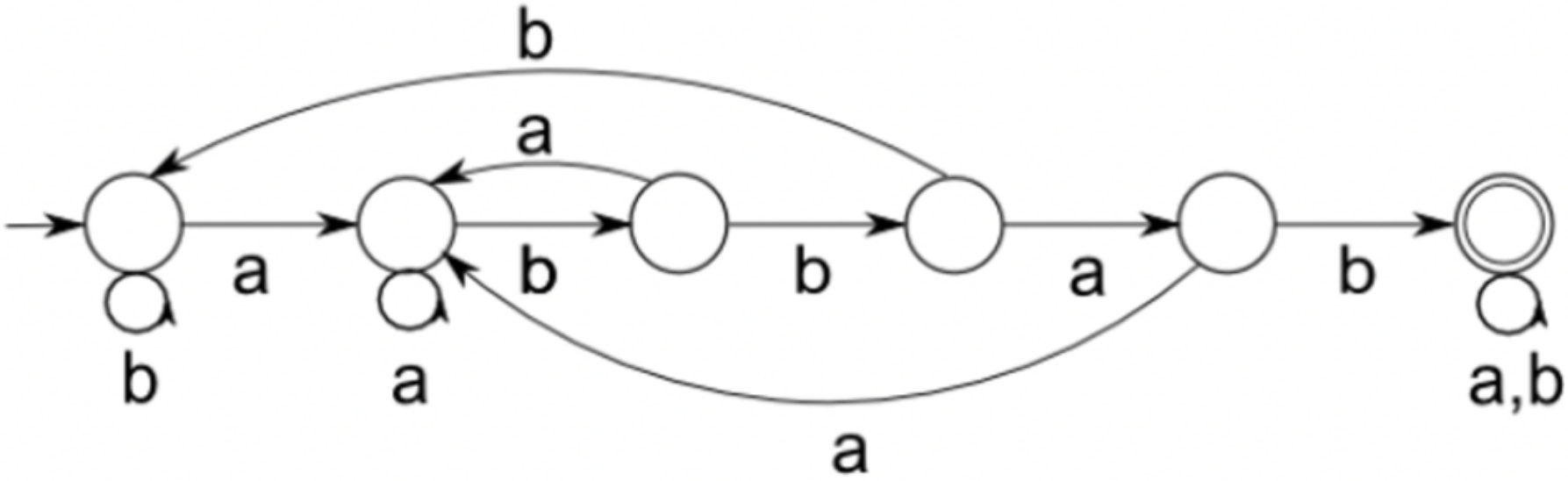


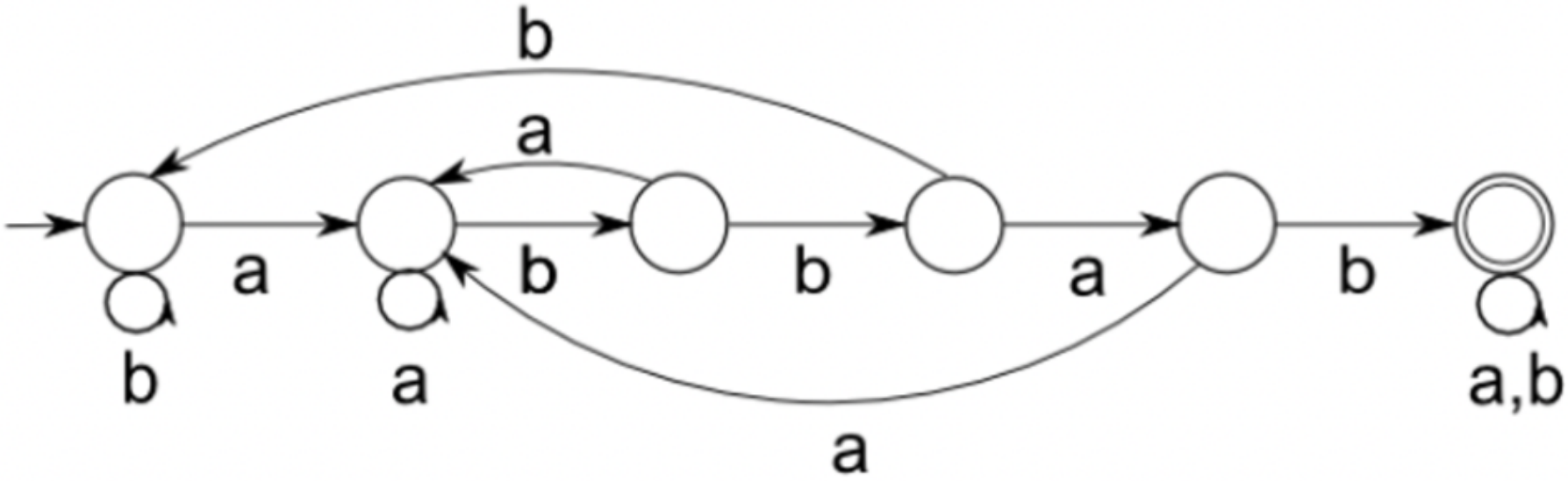
*из всех вершин есть возможность перейти по ВСЕМ символам алфавита. Здесь для уменьшения объема графа не показаны переходы по каждому неучтенному симоволу в начальное состояние

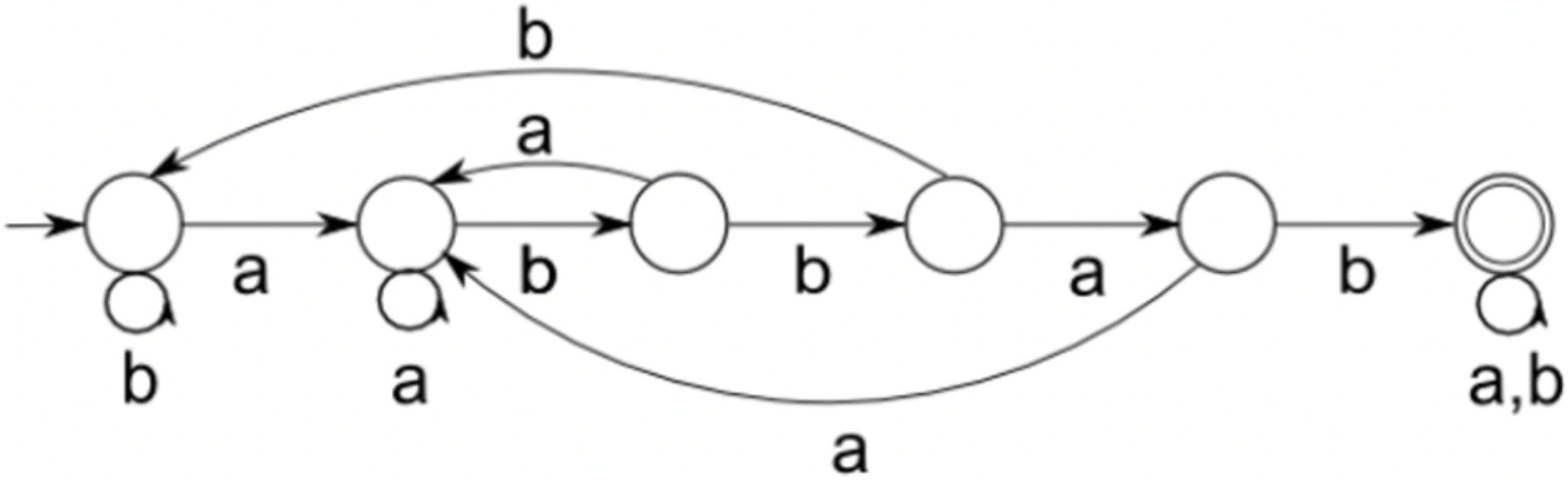
КАК ХРАНИМ **ДКА** ?

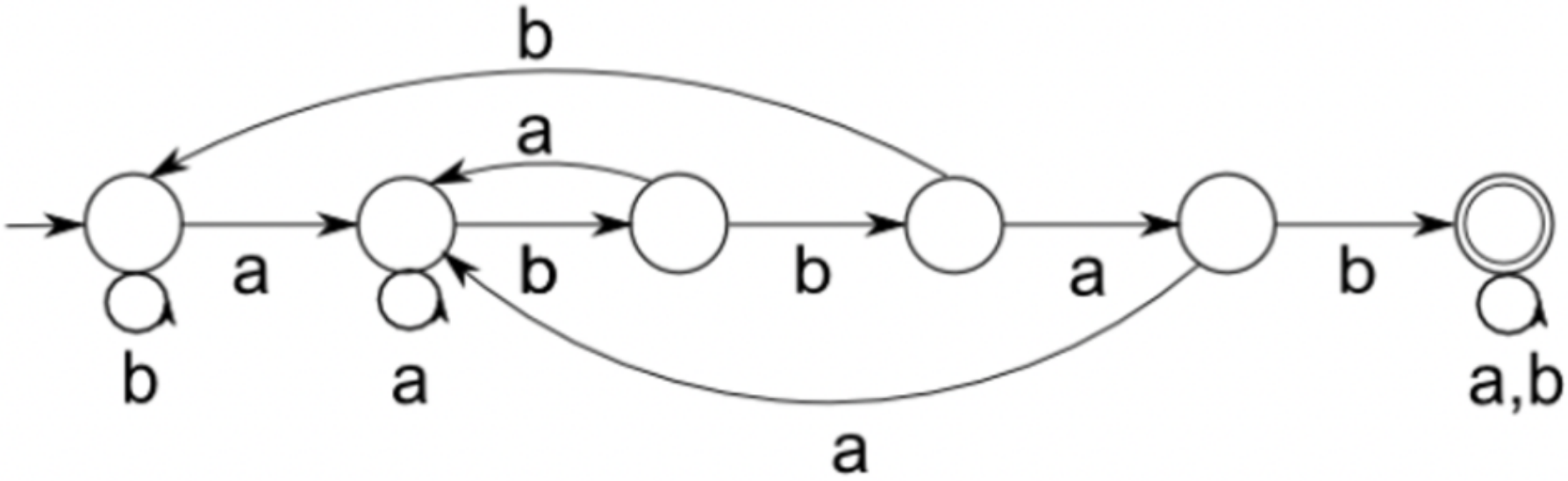

b
a
a
b
b
a
b
b
a
a,b
a

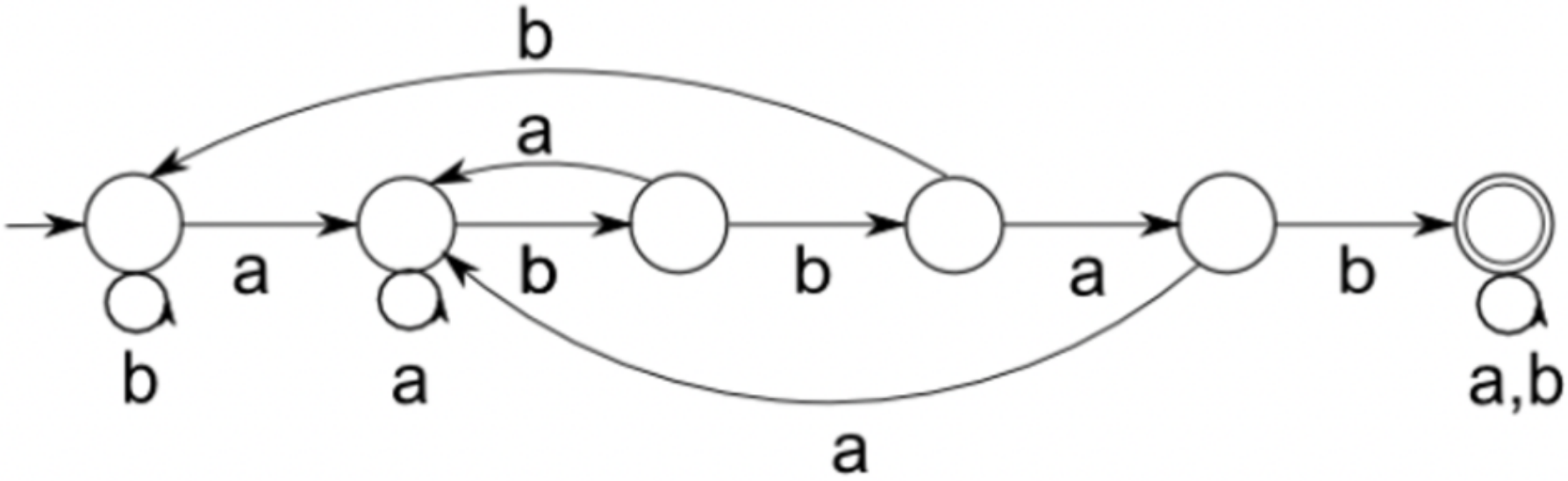

b
a
a
b
b
a
b
b
a
a,b
a

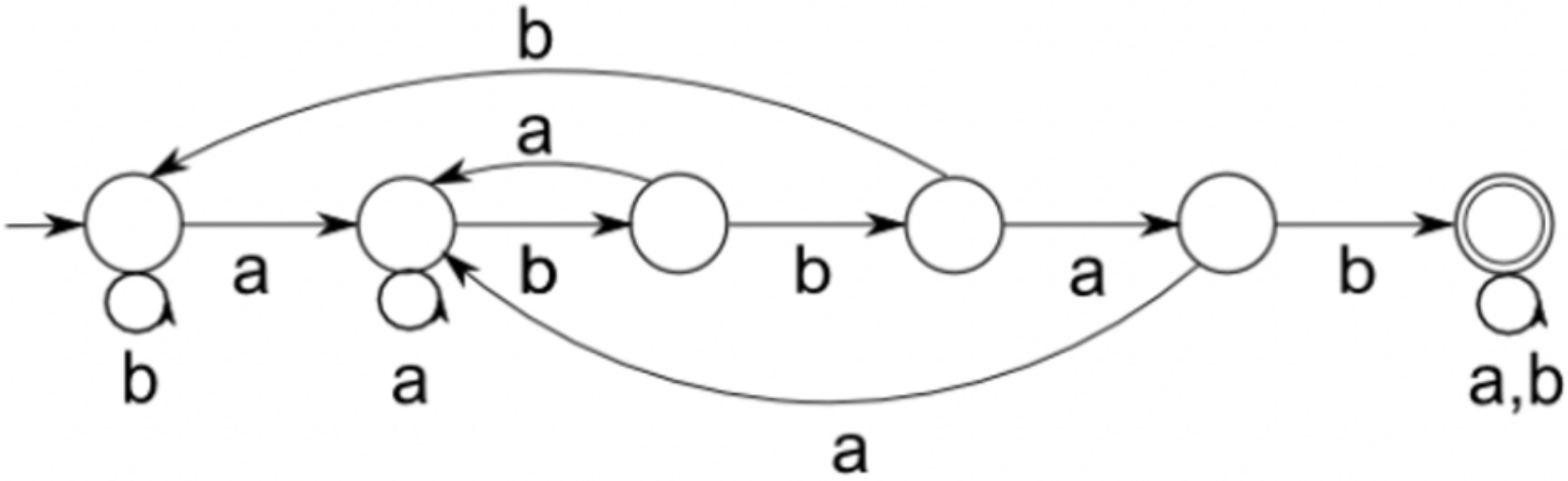

b
a
a
b
b
a
b
b
a
a,b
a

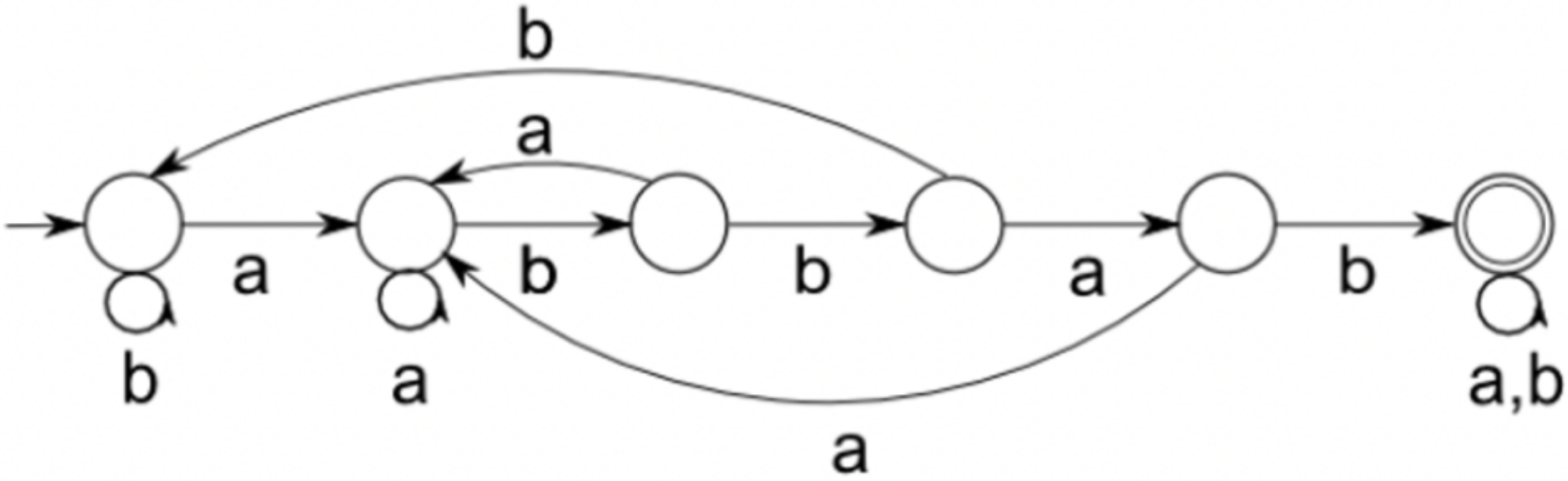

b
a
a
b
b
a
b
b
a
a,b
a

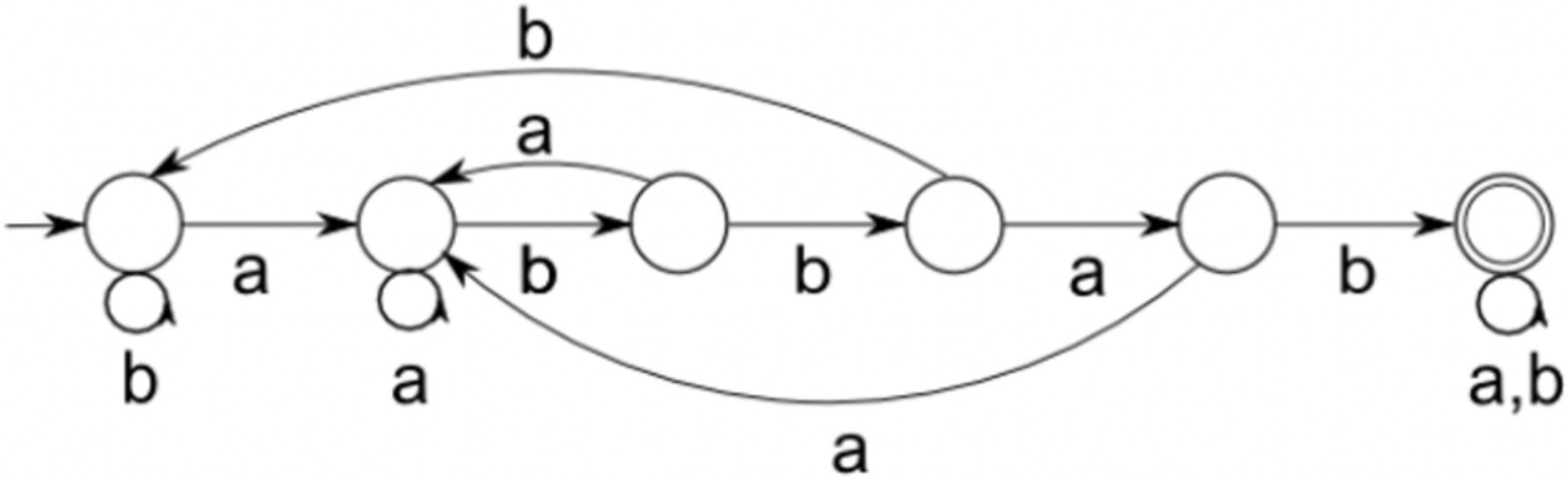

b
a
a
b
b
a
b
b
a
a,b
a

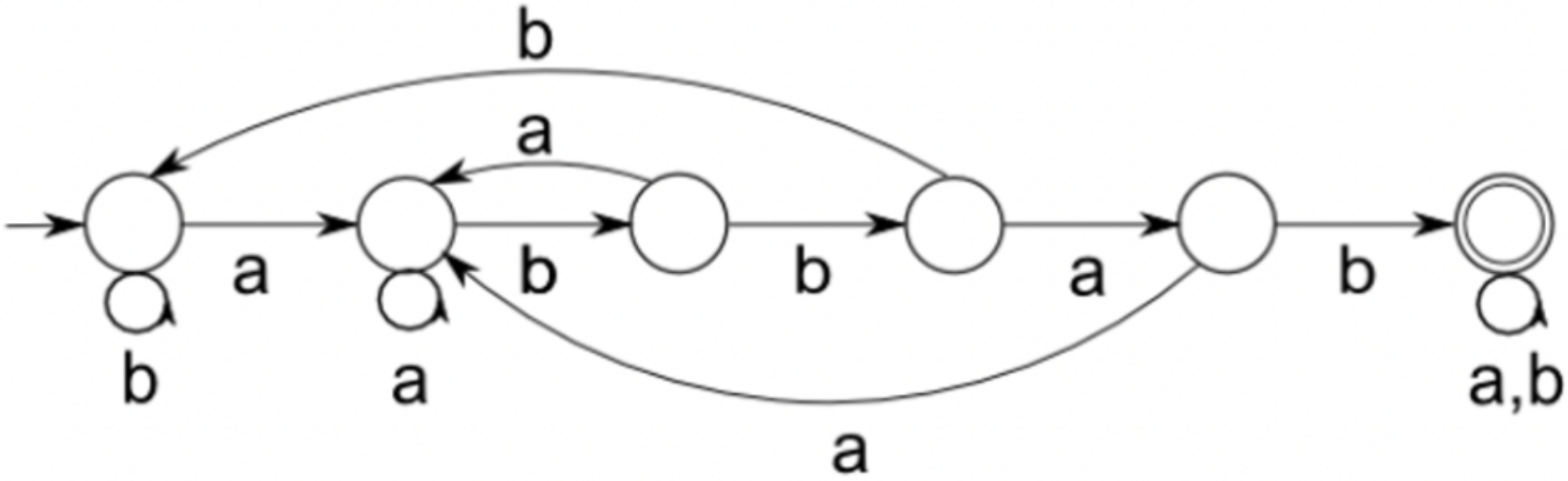

b
a
a
b
b
a
b
b
a
a,b
a

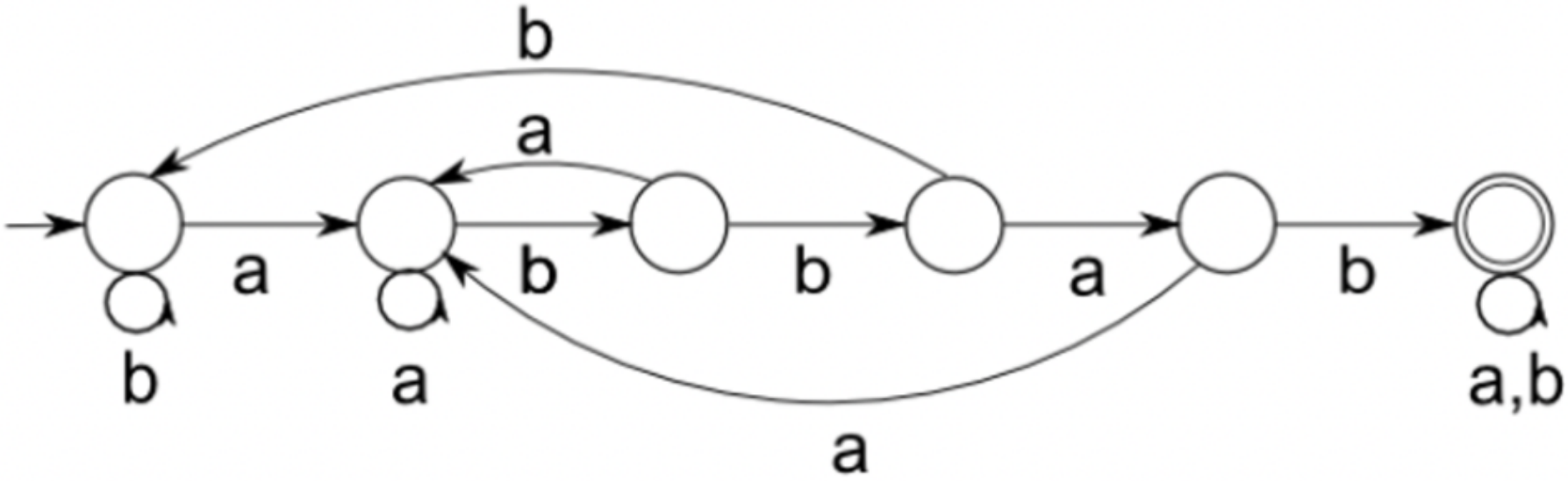

b
a
a
b
b
a
b
b
a
a,b
a

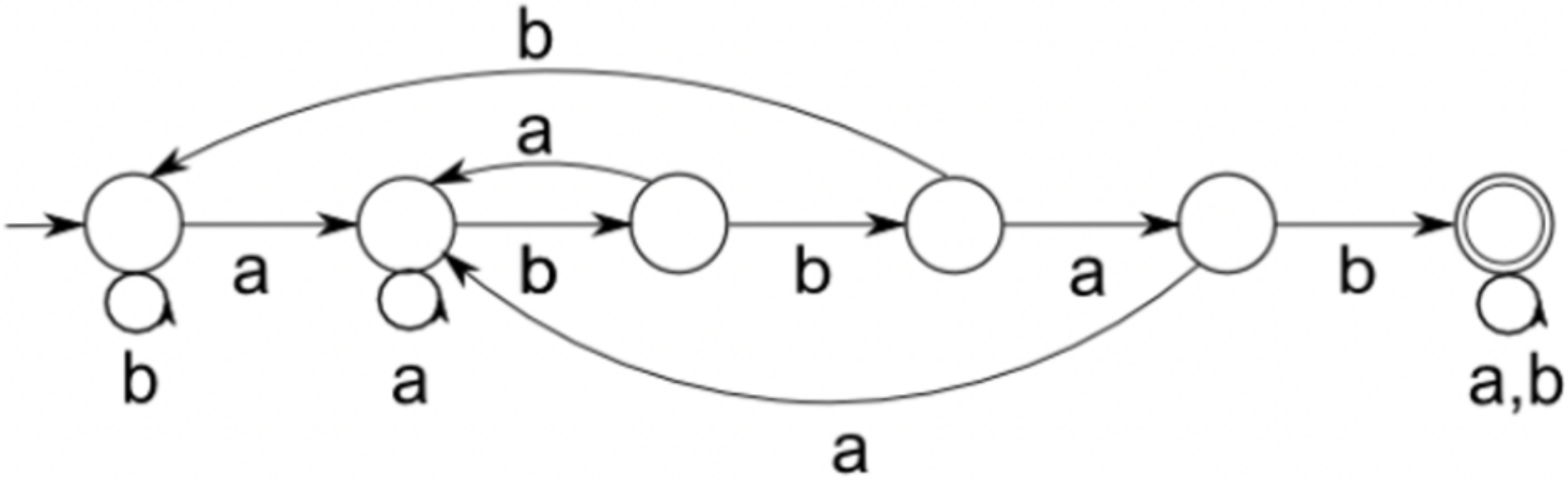

b
a
a
b
b
a
b
b
a
a,b
a

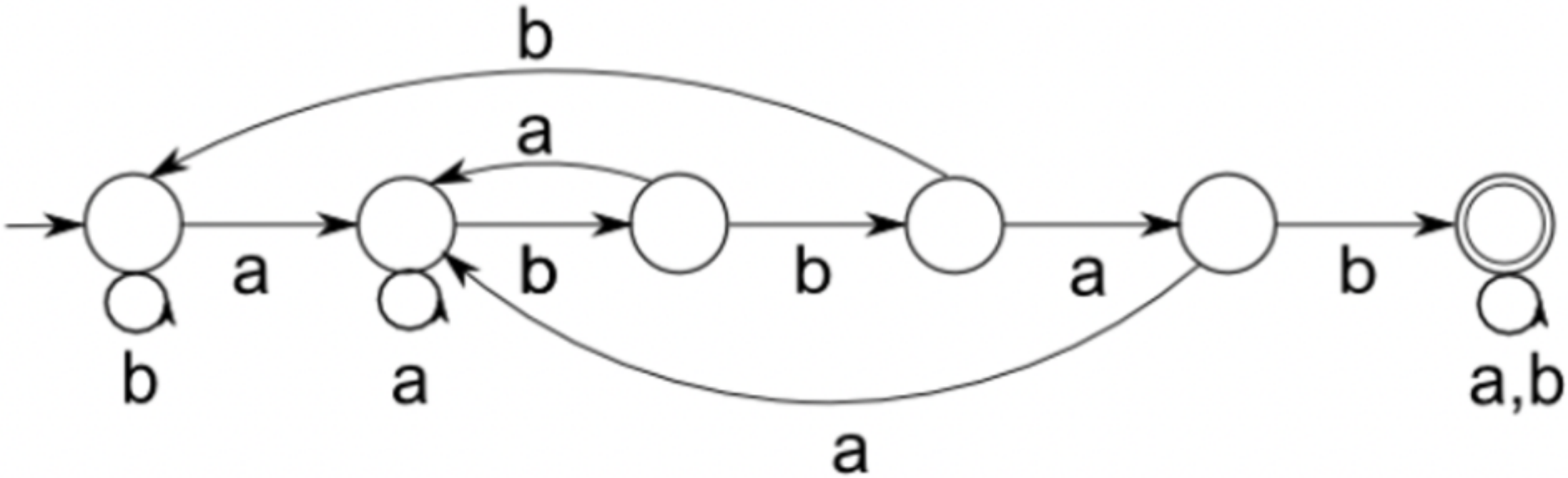

b
a
a
b
b
a
b
b
a
a,b
a

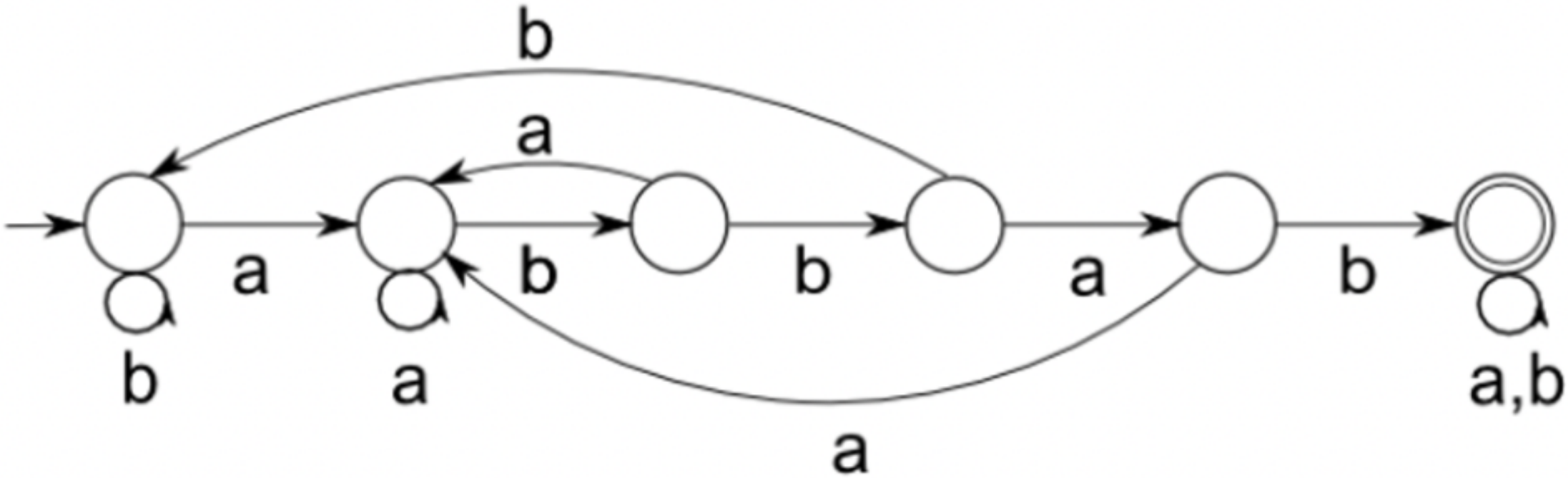

b
a
a
b
b
a
b
b
a
a,b
a

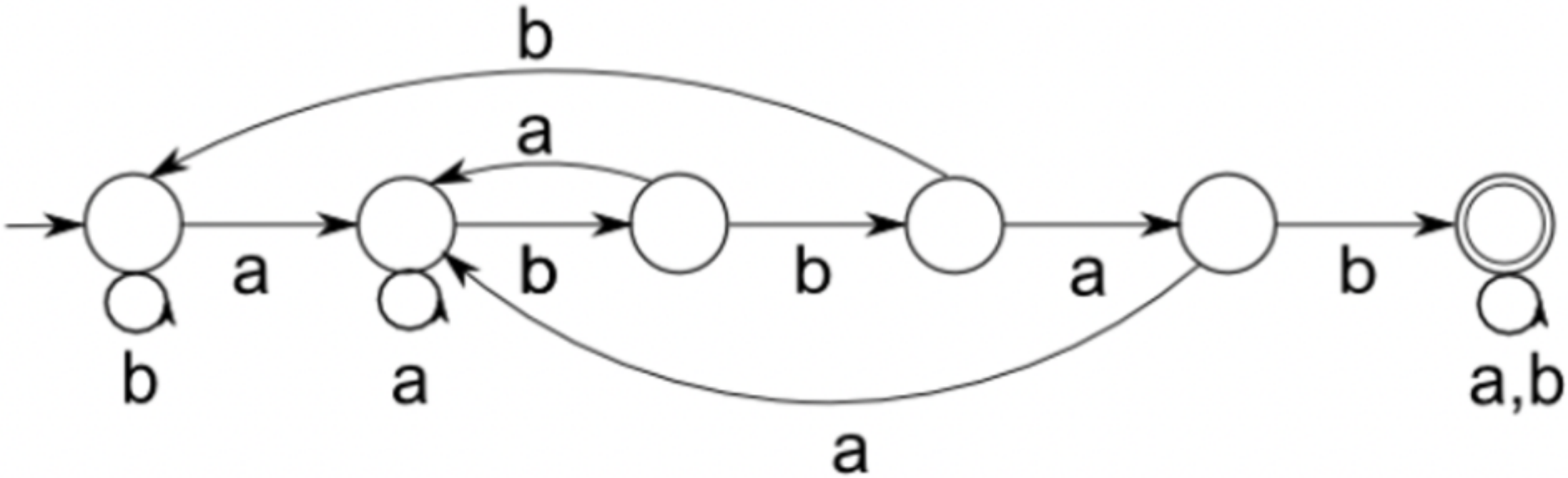

b
a
a
b
b
a
b
b
a
a,b
a

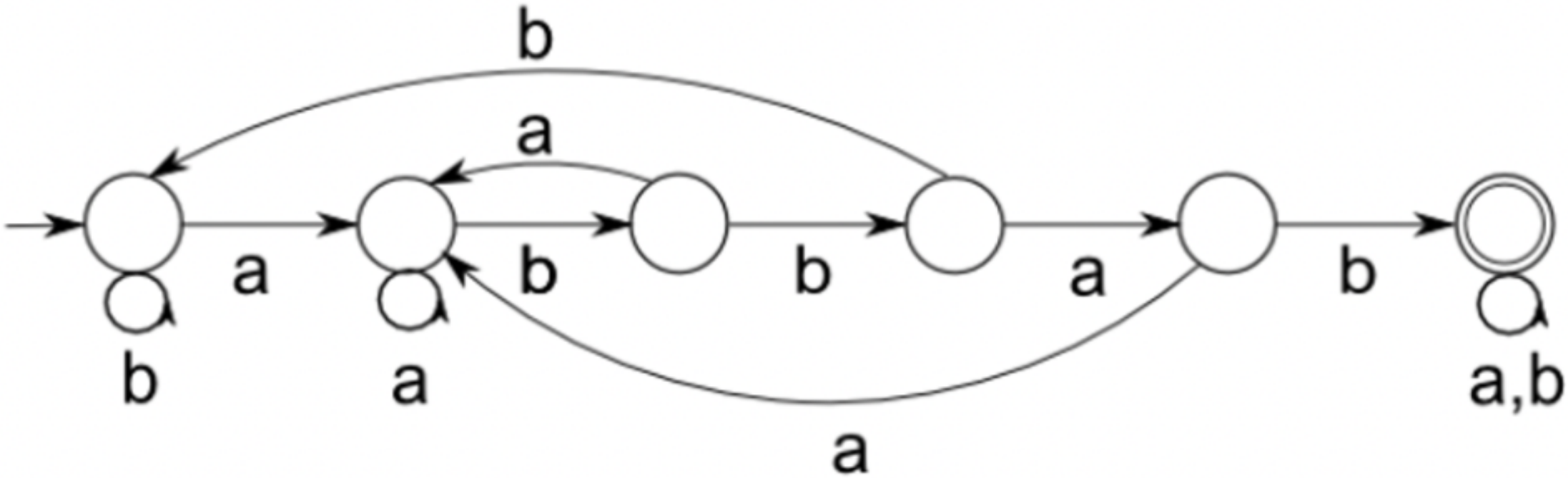

b
a
a
b
b
a
b
b
a
a,b
a

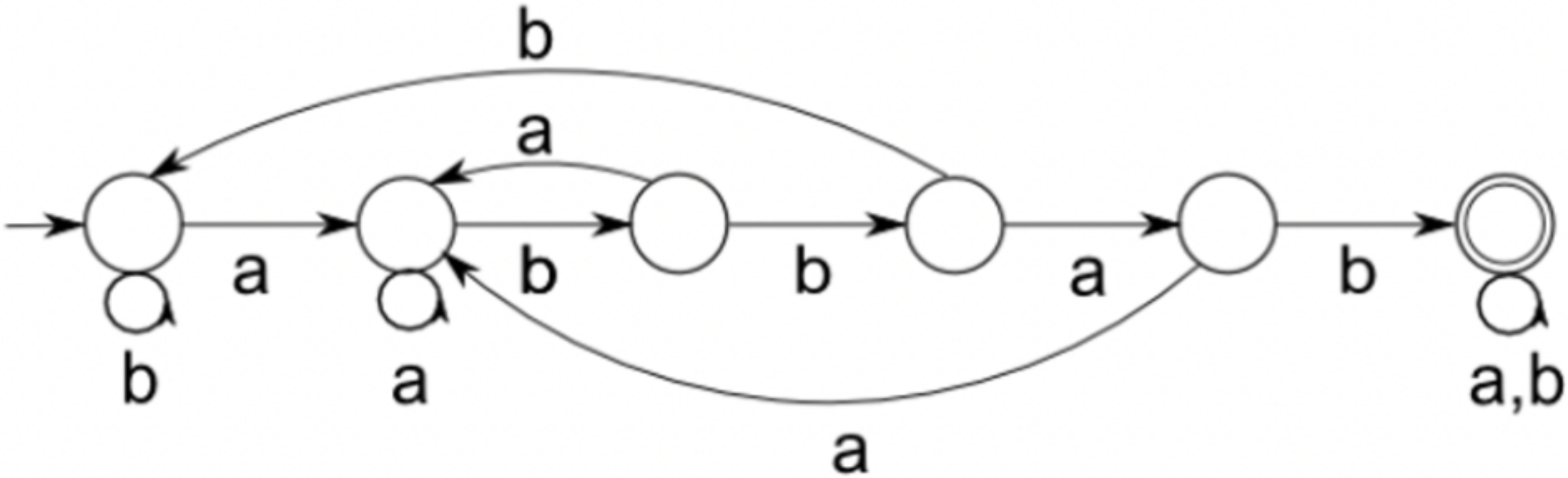

b
a
a
b
b
a
b
b
a
a,b
a

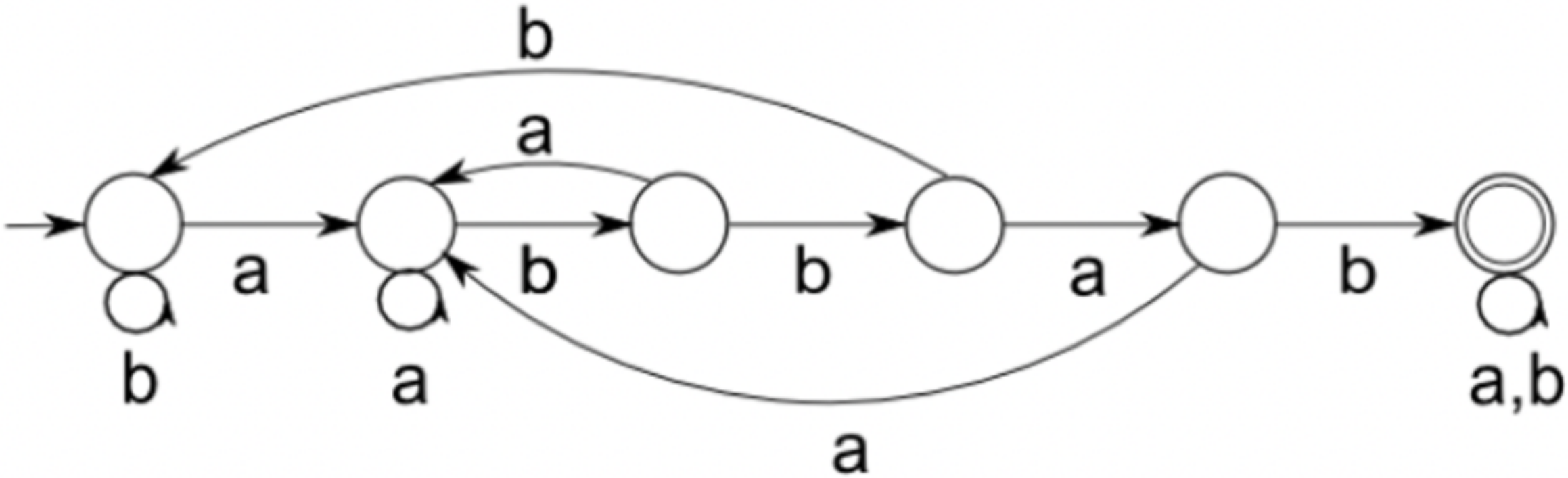

b
a
a
b
b
a
b
b
a
a
a,b


b
a
a
b
b
a
b
b
a
a
a,b


b
a
a
b
b
a
b
b
a
a
a,b


b
a
a
b
b
a
b
b
a
a
a,b


b
a
a
b
b
a
b
b
a
a
a,b


b
a
a
b
b
a
b
b
a
a
a,b

ПОСТРОЕНИЕ ДКА
ПО ПАТТЕРНУ

ПРИМЕР СОЗДАНИЯ АВТОМАТА ДЛЯ СБОРКИ-ОБРАЗЦА:

- P = **ababaca**
- Префикс функция π =

	a	b	a	b	a	c	a
-	0	0	1	2	3	0	1

ПРИМЕР СОЗДАНИЯ АВТОМАТА ДЛЯ СБОРКИ-ОБРАЗЦА:

- P = **ababaca**
- Префикс функция π =

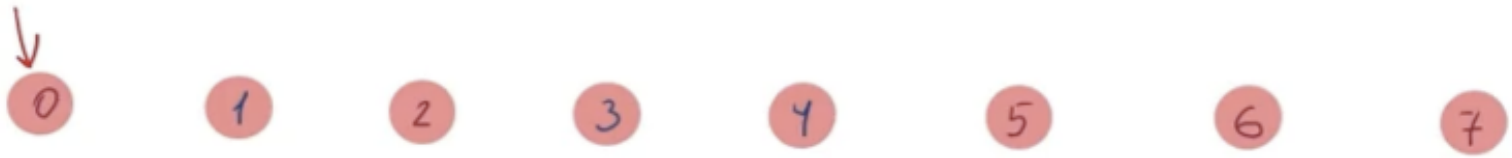
	a	b	a	b	a	c	a
-	0	0	1	2	3	0	1
0	1	2	3	4	5	6	7

ПРИМЕР СОЗДАНИЯ АВТОМАТА ДЛЯ СБОРКИ-ОБРАЗЦА:

- **P = ababaca**
- **Префикс функция π =**

	a	b	a	b	a	c	a
-	0	0	1	2	3	0	1
0	1	2	3	4	5	6	7

- Первоначальное представление автомата:

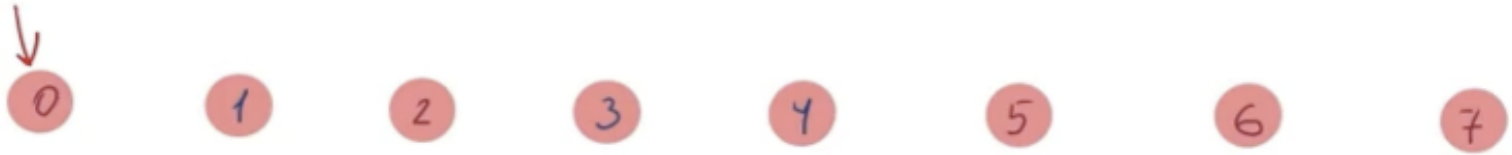


ПРИМЕР СОЗДАНИЯ АВТОМАТА ДЛЯ СБОРКИ-ОБРАЗЦА:

- **P = ababaca**
- **Префикс функция π =**

	a	b	a	b	a	c	a
-	0	0	1	2	3	0	1
0	1	2	3	4	5	6	7

- Первоначальное представление автомата:

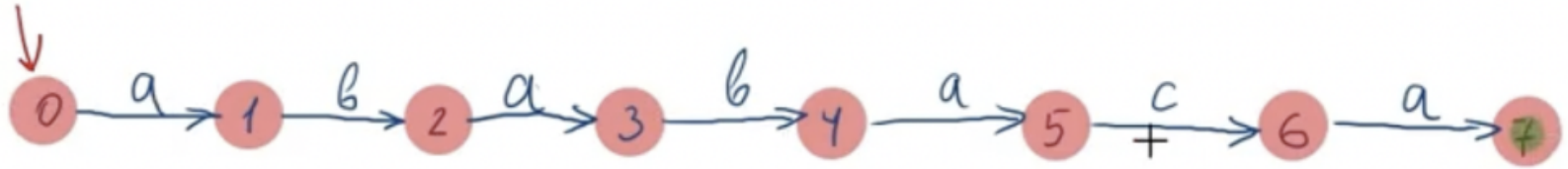


- Теперь необходимо подсчитать функции переходов

ПОДСЧЕТ ФУНКЦИИ ПЕРЕХОДОВ

- Перед подсчетом функции переходов для каждого состояния, заметим, что у нас уже существует идеальный сценарий, который заключается в следующем:

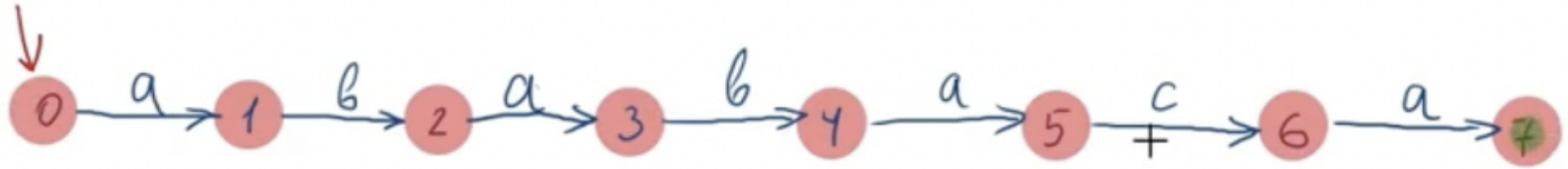
1. Видим **a** - идем в состояние **1**
2. Видим **b** - идем в состояние **2**
3. Видим **a** - идем в состояние **3**
4. Видим **b** - идем в состояние **4**
5. Видим **a** - идем в состояние **5**
6. Видим **c** - идем в состояние **6**
7. Видим **a** - идем в состояние **7**



ПОДСЧЕТ ФУНКЦИИ ПЕРЕХОДОВ

- Перед подсчетом функции переходов для каждого состояния, заметим, что у нас уже существует идеальный сценарий, который заключается в следующем:

1. Видим **a** - идем в состояние **1**
2. Видим **b** - идем в состояние **2**
3. Видим **a** - идем в состояние **3**
4. Видим **b** - идем в состояние **4**
5. Видим **a** - идем в состояние **5**
6. Видим **c** - идем в состояние **6**
7. Видим **a** - идем в состояние **7**

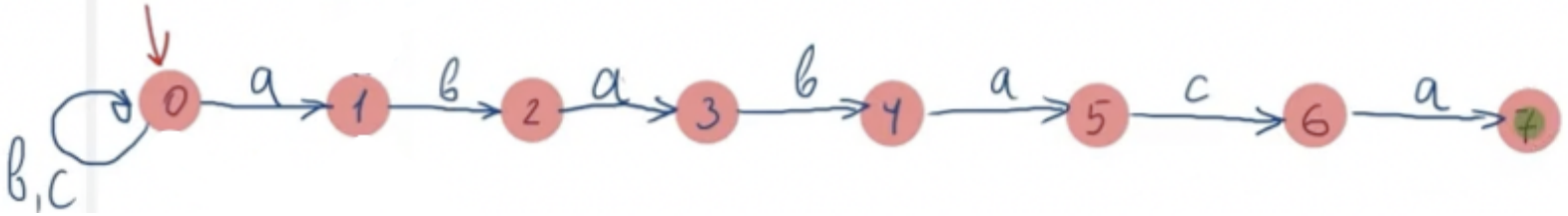


ПОДСЧЕТ ФУНКЦИИ ПЕРЕХОДОВ

- Предположим, что случилась ситуация, когда мы не увидели символ, позволяющий перейти в следующее состояния, тогда потребуется какой-то другой переход
- Данный переход рассчитывается с помощью префикс-функции, которая была посчитана ранее
- Для расчета требуется просмотреть значение префикс-функции под индексом, который соответствует состоянию, из которого должен произойти переход
- В ходе данного перехода автомат может перейти в любое состояние, предшествующего тому, из которого происходил данный переход
- Значение префикс-функции позволяет в подобных ситуациях не возвращаться каждый раз в нулевое состояния, а переходить в состояние, несущее уже некоторую информацию о части паттерна, который

ПОДСЧЕТ ФУНКЦИИ ПЕРЕХОДОВ: 0 СОСТОЯНИЕ

- Первоначально нам известно, что из **0** состояния мы можем попасть в **1** состояние пройдя по символу **a**
- Пройдя по символам **b** или **c** автомат останется в начальной(нулевом) состоянии
- Данную информацию мы берем за базу, благодаря которой сможем подсчитать функции переходов для остальных состоян

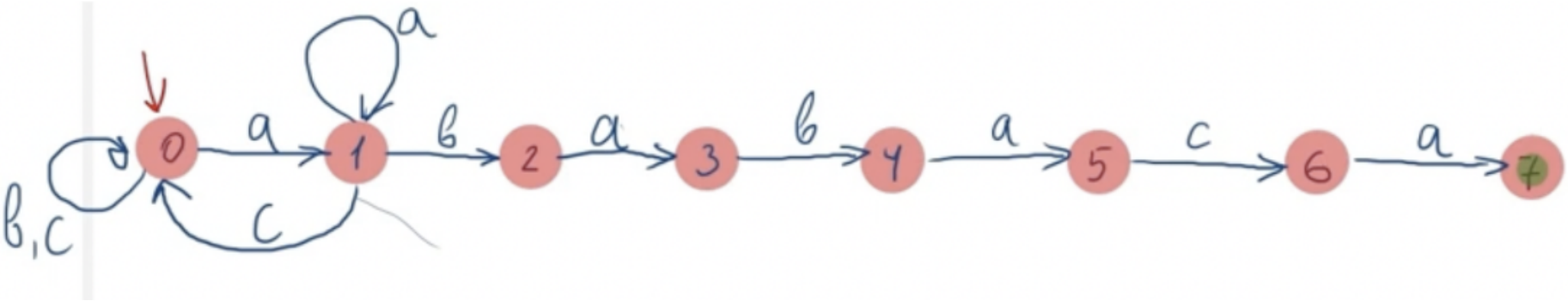


$\delta(0,a) = 1$, $\delta(0,b) = 0$ и т.д. $\delta(0,c) = 0$ ...

ПОДСЧЕТ ФУНКЦИИ ПЕРЕХОДОВ: 1 СОСТОЯНИЕ

- Переход из состояния **1** в состояние **2** по символу **b** нам известен:
- Требуется подсчитать функции переходов для символа **a** и **c**:

$\delta(1, b) = 2$



ПОДСЧЕТ ФУНКЦИИ ПЕРЕХОДОВ: 1 СОСТОЯНИЕ

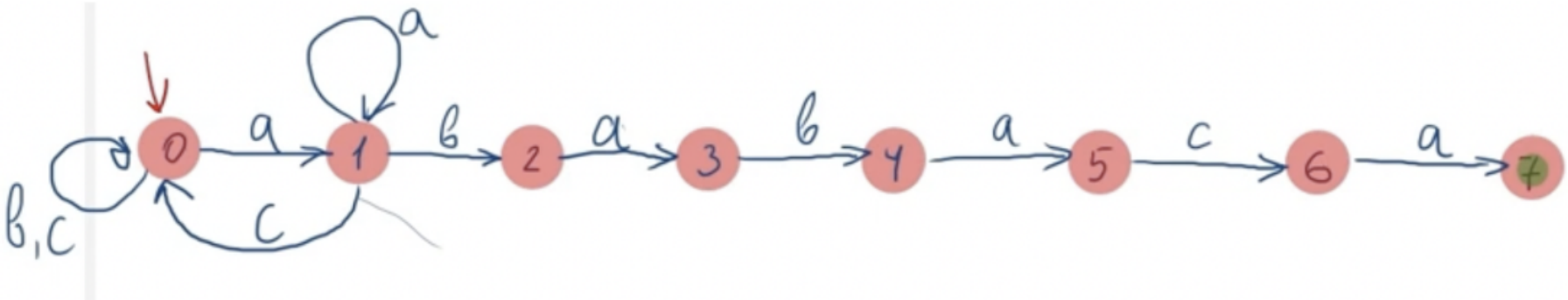
- Переход из состояния **1** в состояние **2** по символу **b** нам известен:

$$\delta(1, b) = 2$$

- Требуется подсчитать функции переходов для символа **a** и **c**:

$$\delta(1, a) = \delta(\pi_q(1), a) = \delta(0, a) = 1$$

$$\delta(1, c) = \delta(\pi_q(1), c) = \delta(0, c) = 0$$



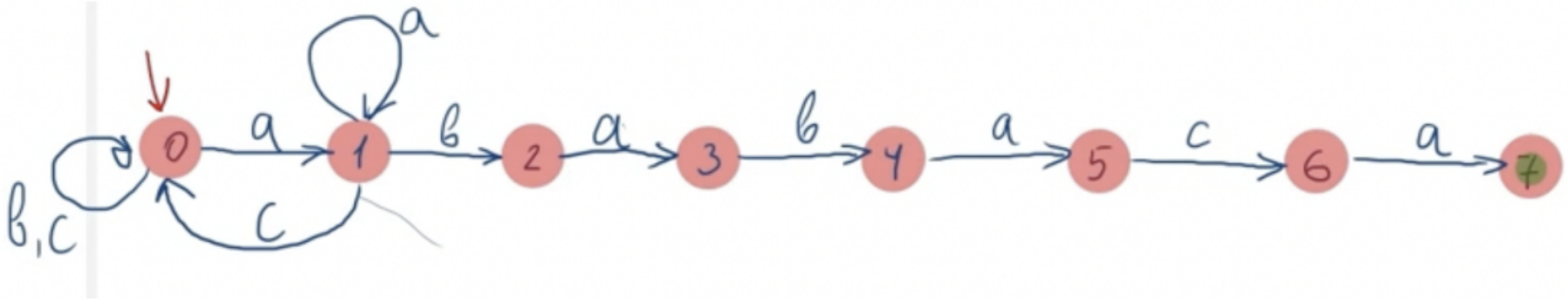
КАК ВЫЧИСЛЯЕМ ПЕРЕХОДЫ ДЛЯ 1 СОСТОЯНИЯ:

- **P = ababaca**
- **Префикс функция π =**

	a	b	a	b	a	c	a
-	0	0	1	2	3	0	1
0	1	2	3	4	5	6	7

$\delta(1, a) = \delta(\pi_q(1), a) = \delta(0, a) = 1$

$\delta(1, c) = \delta(\pi_q(1), c) = \delta(0, c) = 0$



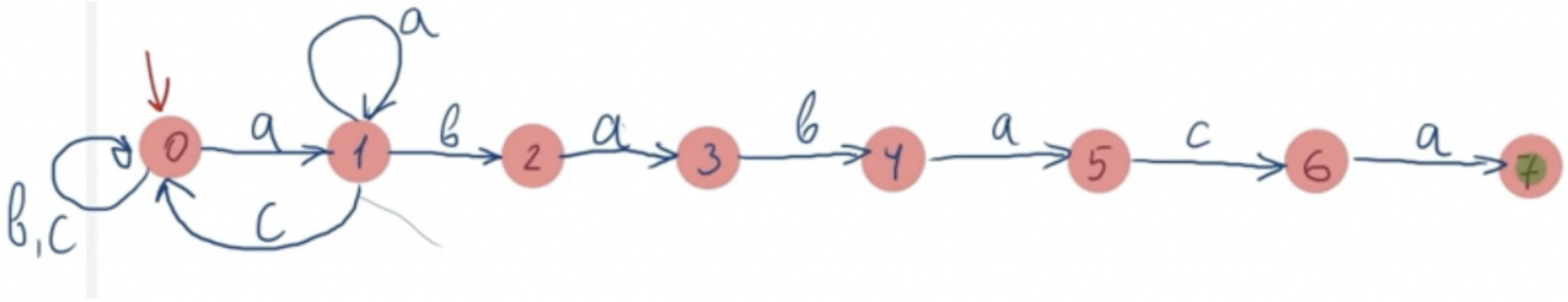
КАК ВЫЧИСЛЯЕМ ПЕРЕХОДЫ ДЛЯ 1 СОСТОЯНИЯ:

- **P = ababaca**
- **Префикс функция π =**

	a	b	a	b	a	c	a
-	0	0	1	2	3	0	1
0	1	2	3	4	5	6	7

$\delta(1, a) = \delta(\pi_q(1), a) = \delta(0, a) = 1$

$\delta(1, c) = \delta(\pi_q(1), c) = \delta(0, c) = 0$



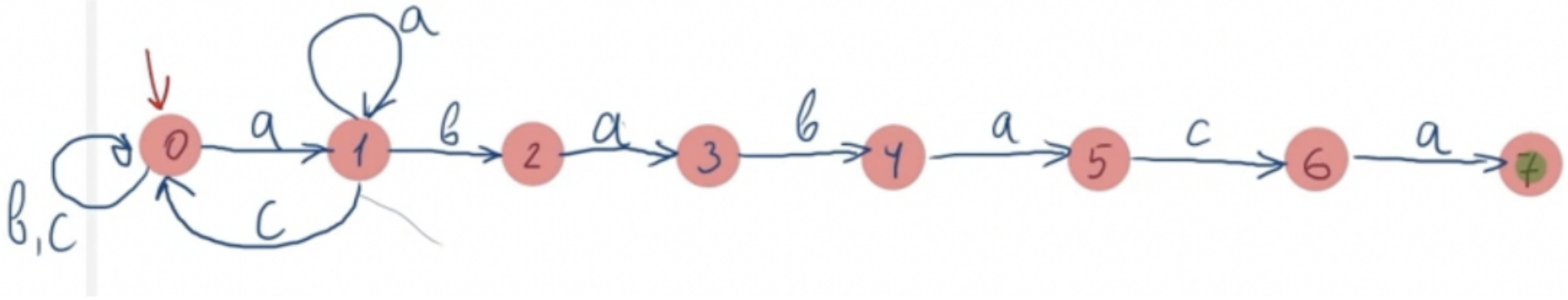
КАК ВЫЧИСЛЯЕМ ПЕРЕХОДЫ ДЛЯ 1 СОСТОЯНИЯ:

- P = **ababaca**
- Префикс функция π =

	a	b	a	b	a	c	a
-	0	0	1	2	3	0	1
0	1	2	3	4	5	6	7

$\delta(1, a) = \delta(\pi_q(1), a) = \delta(0, a) = 1$

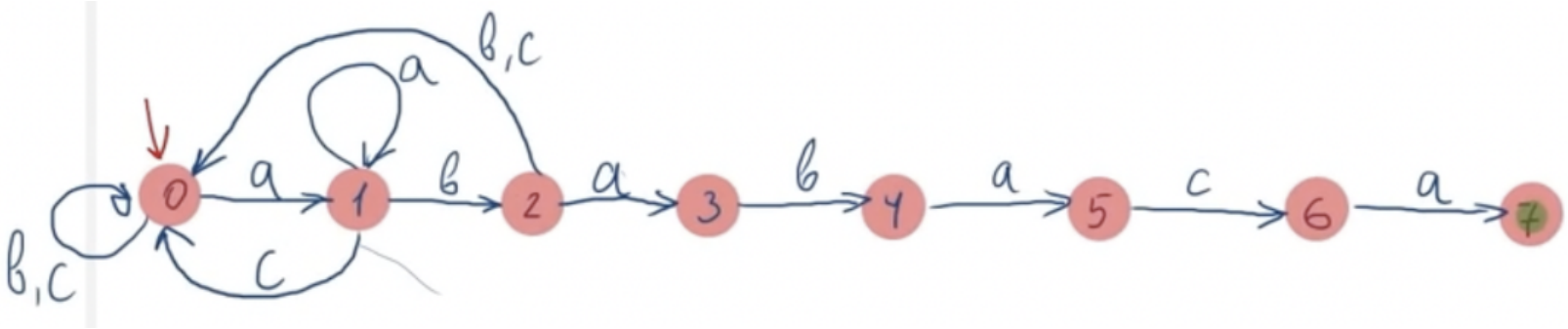
$\delta(1, c) = \delta(\pi_q(1), c) = \delta(0, c) = 0$



ПОДСЧЕТ ФУНКЦИИ ПЕРЕХОДОВ: **2 СОСТОЯНИЕ**

- Переход из состояния **2** в состояние **3** по символу **a** нам известен:
- Требуется подсчитать функции переходс для символа **b** и **c**:

$$\delta(2,a) = 3$$



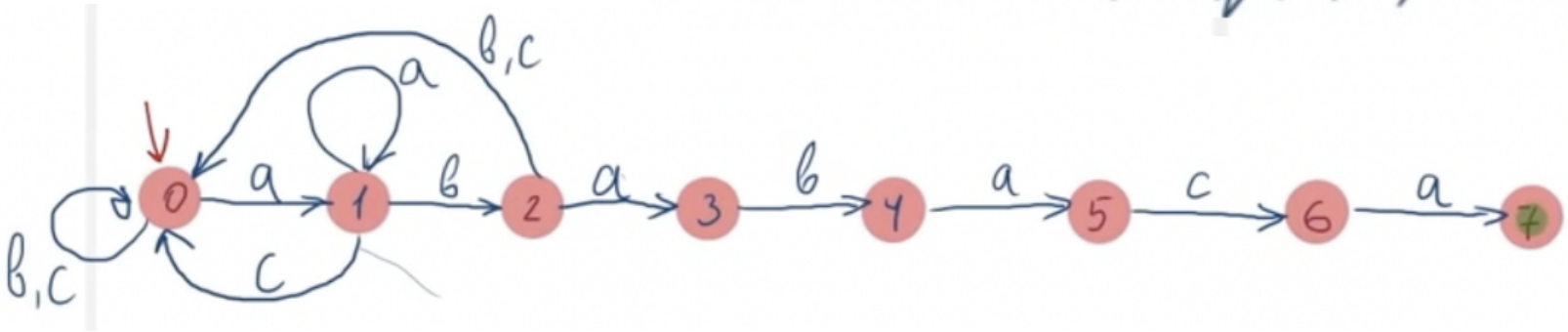
ПОДСЧЕТ ФУНКЦИИ ПЕРЕХОДОВ: **2 СОСТОЯНИЕ**

- Переход из состояния **2** в состояние **3** по символу **a** нам известен:
- Требуется подсчитать функции переходс для символа **b** и **c**:

$$\delta(2,a) = 3$$

$$\delta(2,b) = (\pi_q(2), b) = \delta(0,b) = 0$$

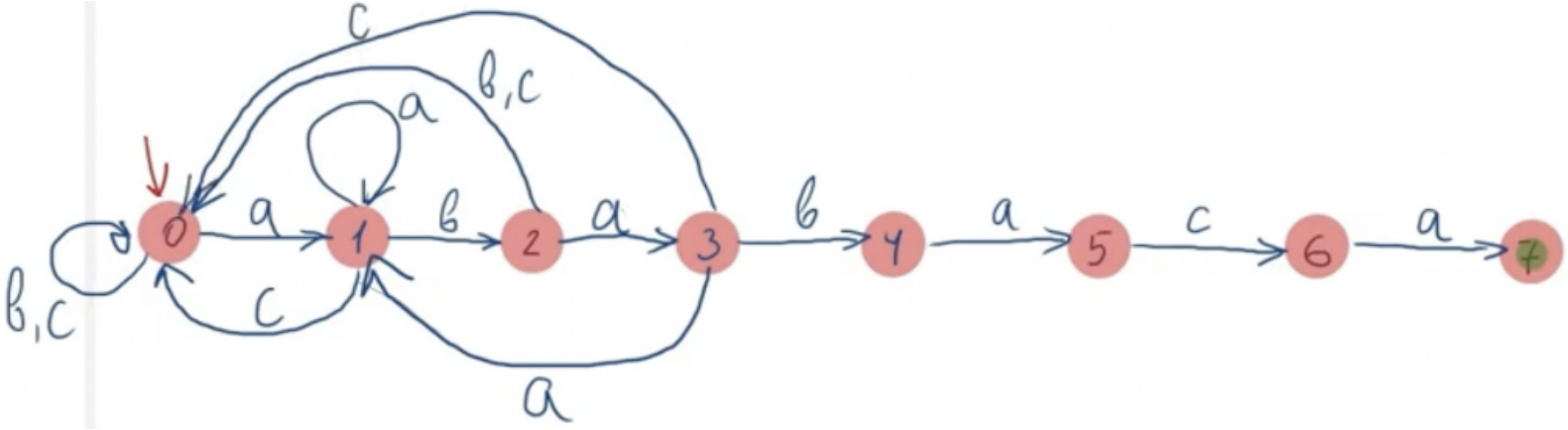
$$\delta(2,c) = (\pi_q(2), c) = \delta(0,c) = 0$$



ПОДСЧЕТ ФУНКЦИИ ПЕРЕХОДОВ: **3 СОСТОЯНИЕ**

- Переход из состояния **3** в состояние **4** по символу **b** нам известен:
- Требуется подсчитать функции переходов для символа **b** и **c**:

$$\delta(3, b) = 4$$



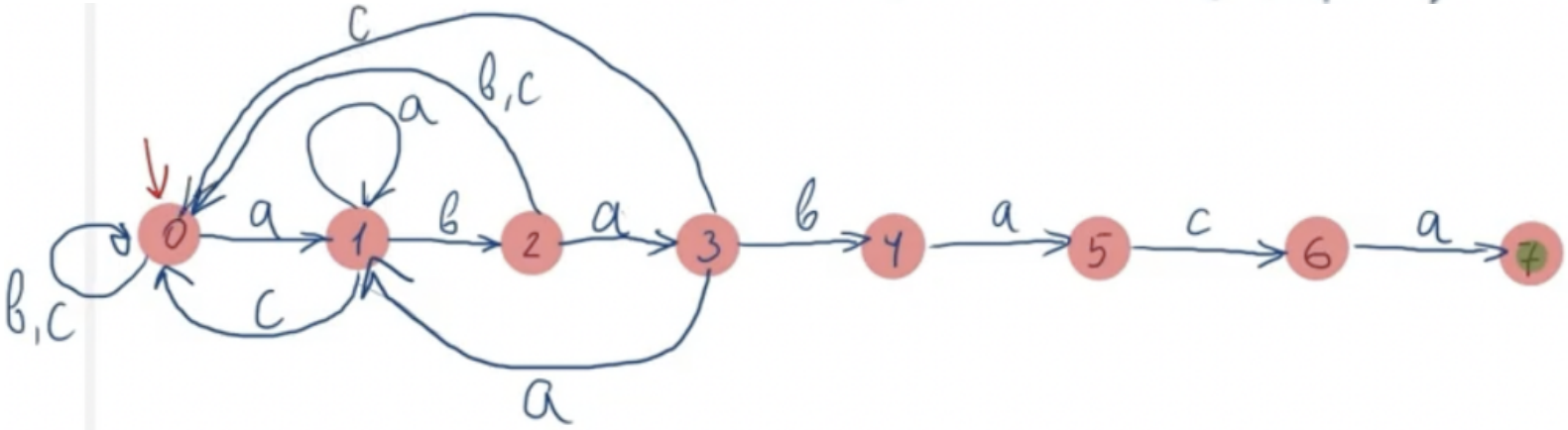
ПОДСЧЕТ ФУНКЦИИ ПЕРЕХОДОВ: 3 СОСТОЯНИЕ

- Переход из состояния **3** в состояние **4** по символу **b** нам известен:
- Требуется подсчитать функции переходов для символа **b** и **c**:

$$\delta(3, b) = 4$$

$$\delta(3, a) = \delta(1, a) = 1$$

$$\delta(3, c) = \delta(1, c) = 0$$



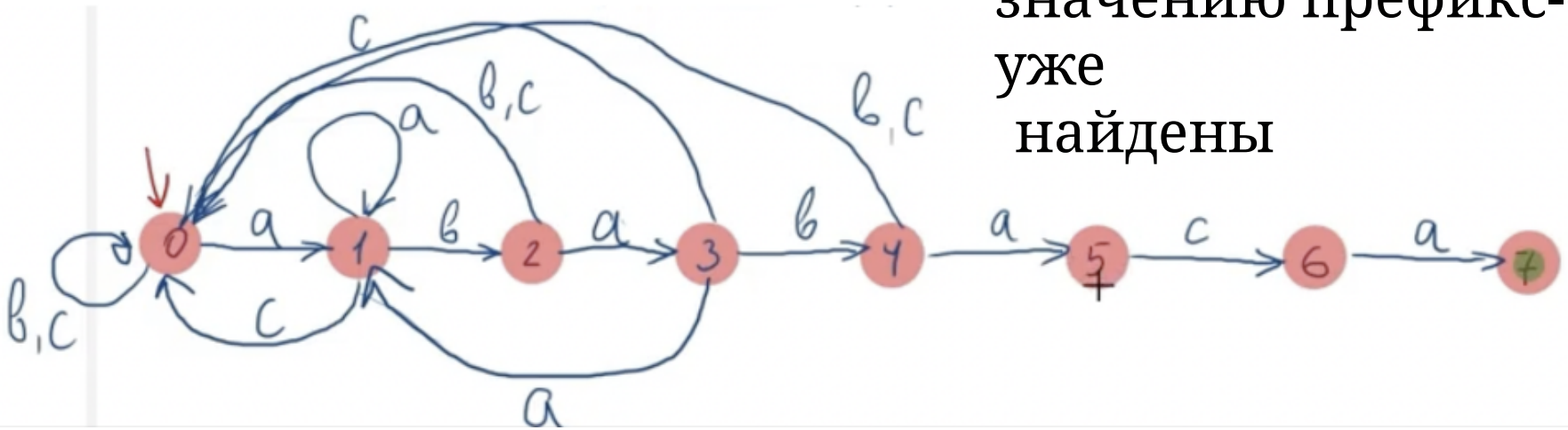
После первого знака равенства было сразу подставлено значение префикс-функции под индексом 3

ПОДСЧЕТ ФУНКЦИИ ПЕРЕХОДОВ: 4 СОСТОЯНИЕ

- Переход из состояния **4** в состояние **5** по символу **a** нам известен: **δ(4, a) = 5**
- Требуется подсчитать функции переходов для символа **b** и **c**:
- • **δ(4,b)=**
- • **δ(4,c)=**

Аналогичные действия:

1. Берем значение префикс-функции под индексом текущего состояния
2. Все переходы для состояния равного значению префикс-функции, взятой выше уже найдены

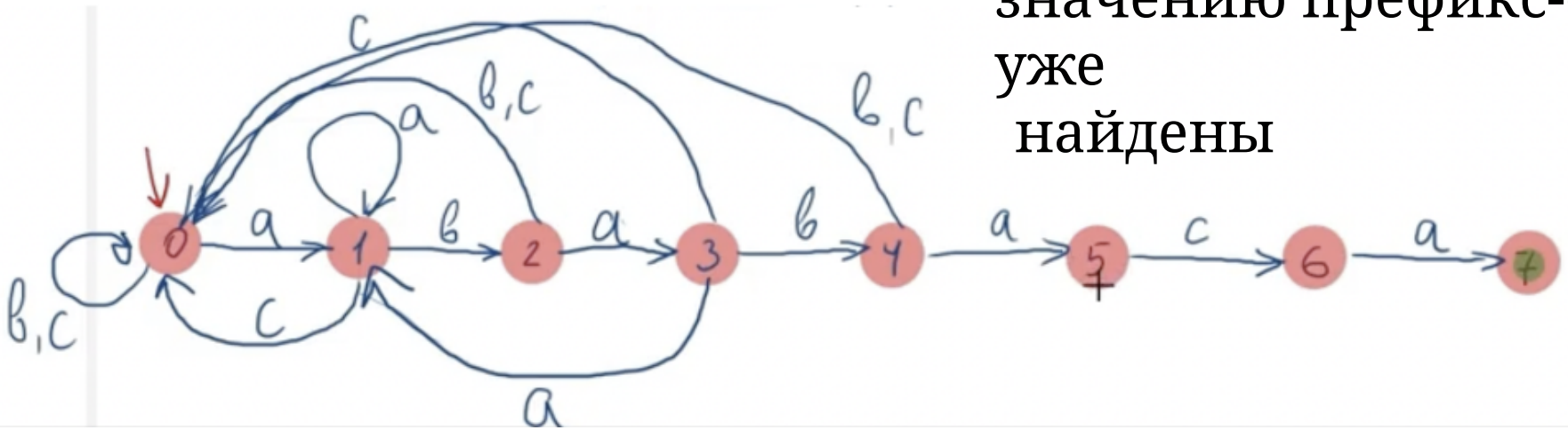


ПОДСЧЕТ ФУНКЦИИ ПЕРЕХОДОВ: **4 СОСТОЯНИЕ**

- Переход из состояния **4** в состояние **5** по символу **a** нам известен: **δ(4, a) = 5**
- Требуется подсчитать функции переходов для символа **b** и **c**:
- **• δ(4,b)=δ(2,b)=0**
  • δ(4,c)=δ(2,c)=0

Аналогичные действия:

1. Берем значение префикс-функции под индексом текущего состояния
2. Все переходы для состояния равного значению префикс-функции, взятой выше уже найдены

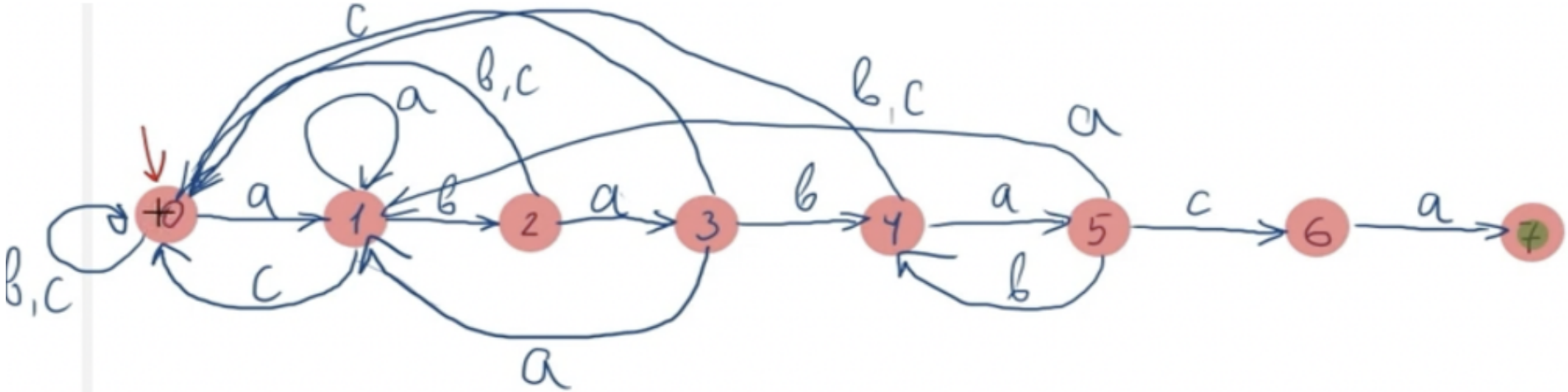


ПОДСЧЕТ ФУНКЦИИ ПЕРЕХОДОВ: 5 СОСТОЯНИЕ

- Переход из состояния **5** в состояние **6** по символу **c** нам известен: **δ(5, c) = 6**
- Требуется подсчитать функции переходов для символа **a** и **b**:
- **δ(5,a)=**
- **δ(5,b)=**

Аналогичные действия:

1. Берем значение префикс-функции под индексом текуще состояния
2. Все переходы для состояния равного значению префикс

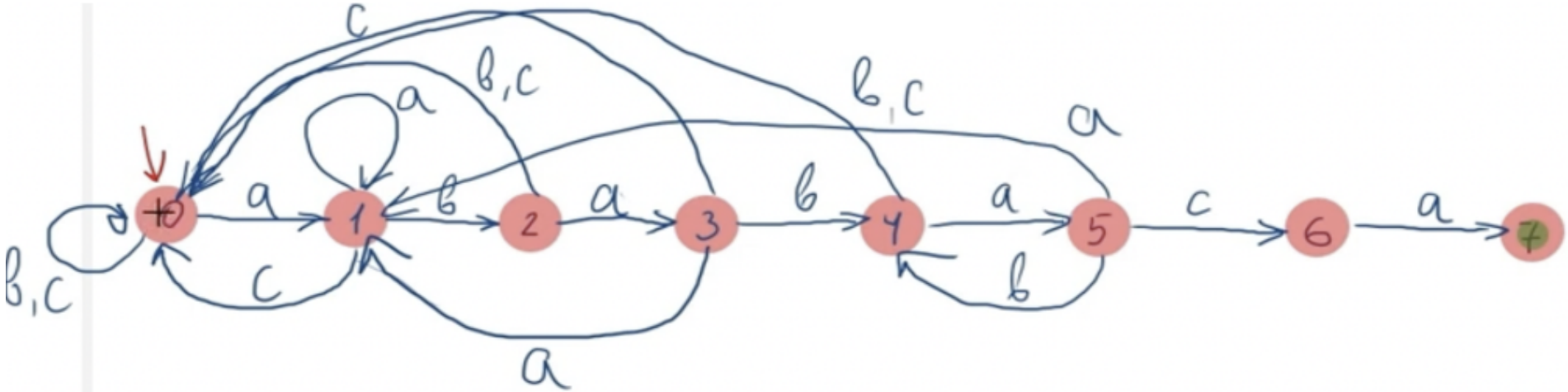


ПОДСЧЕТ ФУНКЦИИ ПЕРЕХОДОВ: 5 СОСТОЯНИЕ

- Переход из состояния **5** в состояние **6** по символу **c** нам известен: **δ(5, c) = 6**
- Требуется подсчитать функции переходов для символа **a** и **b**:
- **δ(5,a)=δ(3,a)=1**
- **δ(5,b)=δ(3,b)=4**

Аналогичные действия:

1. Берем значение префикс-функции под индексом текуще состояния
2. Все переходы для состояния равного значению префикс

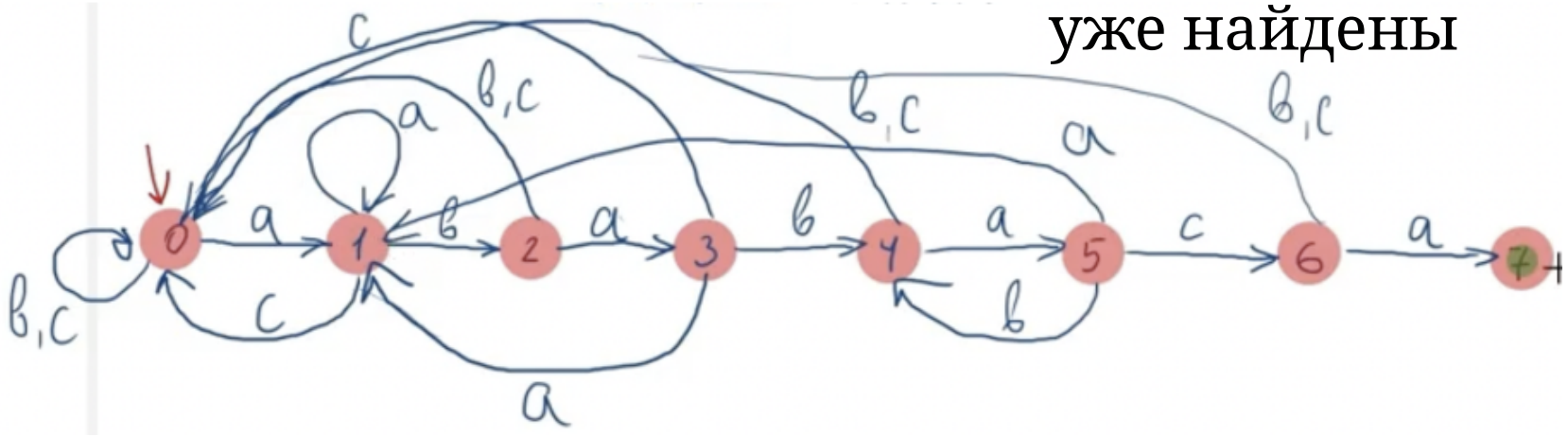


ПОДСЧЕТ ФУНКЦИИ ПЕРЕХОДОВ: 6 СОСТОЯНИЕ

- Переход из состояния **6** в состояние 7 по символу **a** нам известен: **δ(6, a)** = 7
- Требуется подсчитать функции переходов для символа **b** и **c**: •
- **δ(6,b)**=
- **δ(6,c)**=

Аналогичные действия:

1. Берем значение префикс-функции под индексом текущего состояния
2. Все переходы для состояния равного значению префикс-функции, взятой выше, уже найдены

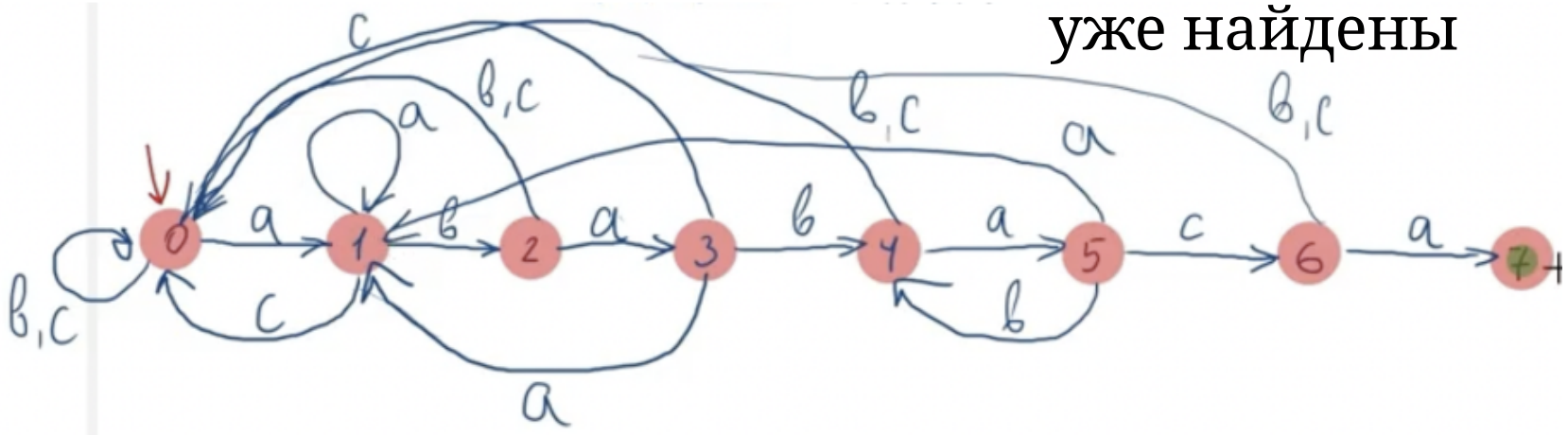


ПОДСЧЕТ ФУНКЦИИ ПЕРЕХОДОВ: 6 СОСТОЯНИЕ

- Переход из состояния **6** в состояние 7 по символу **a** нам известен: **δ(6, a)** = 7
- Требуется подсчитать функции переходов для символа **b** и **c**: •
- **δ(6,b)=δ(0,b)=0**
- **δ(6,c)=δ(0,c)=0**

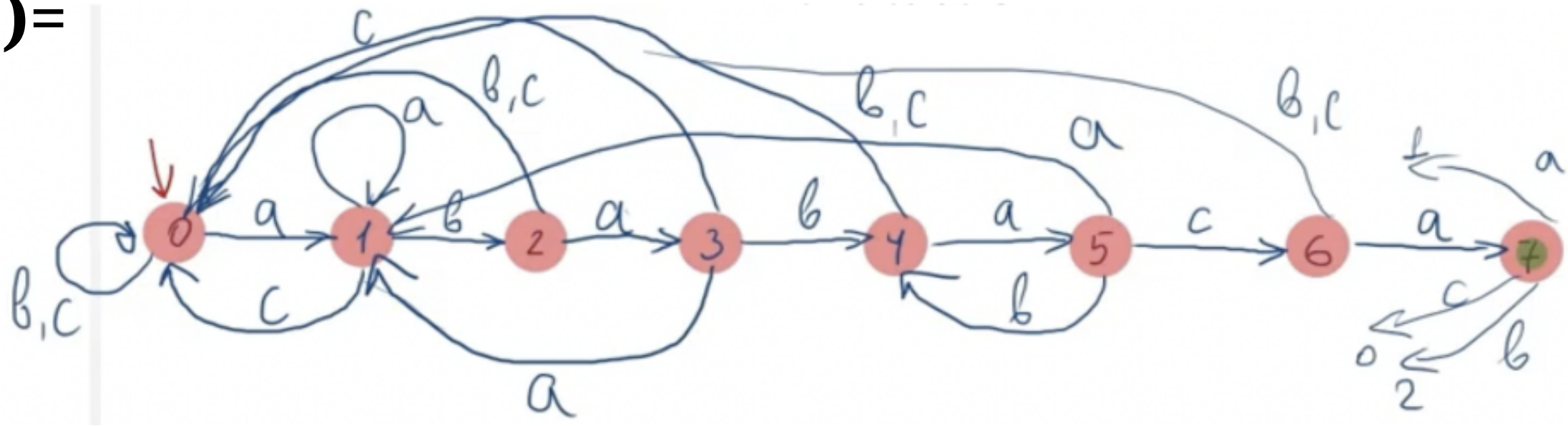
Аналогичные действия:

1. Берем значение префикс-функции под индексом текущего состояния
2. Все переходы для состояния равного значению префикс-функции, взятой выше, уже найдены



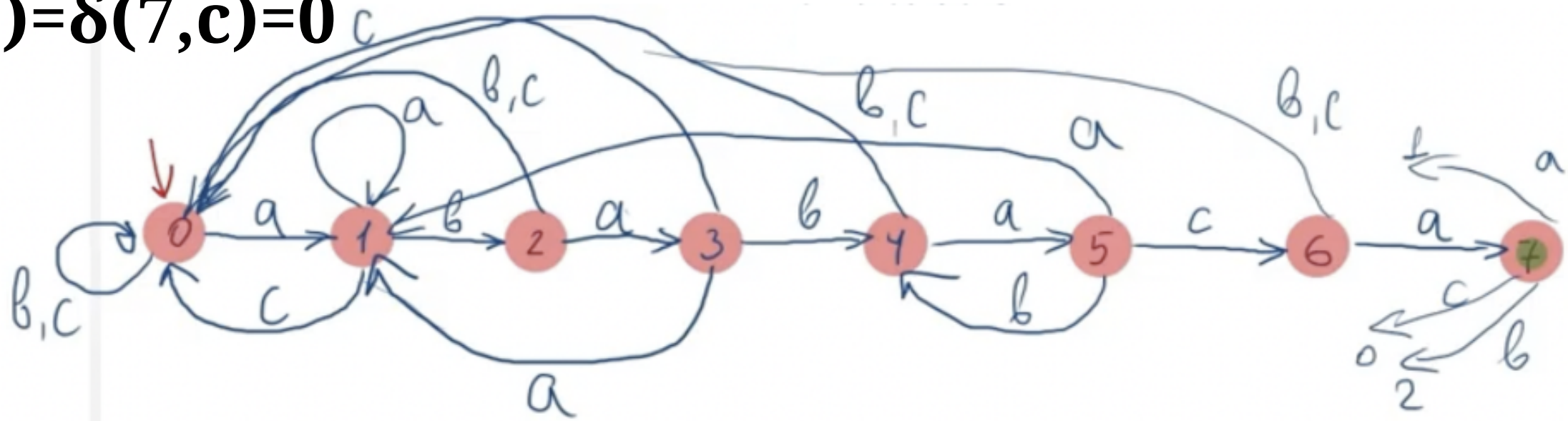
ПОДСЧЕТ ФУНКЦИЙ ПЕРЕХОДОВ: 7 СОСТОЯНИЕ

- 7 состояние - допускающее состояние
- Переходы такие же, как и у **1** состояния
- **δ(7,a)=**
- **δ(7,b)=**
- **δ(7,c)=**



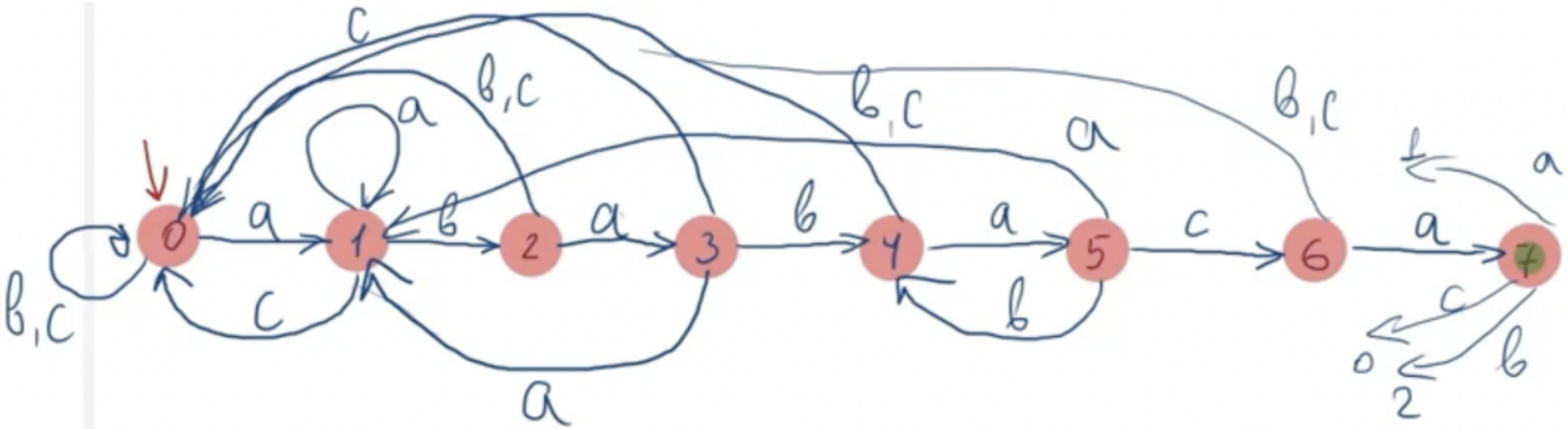
ПОДСЧЕТ ФУНКЦИЙ ПЕРЕХОДОВ: 7 СОСТОЯНИЕ

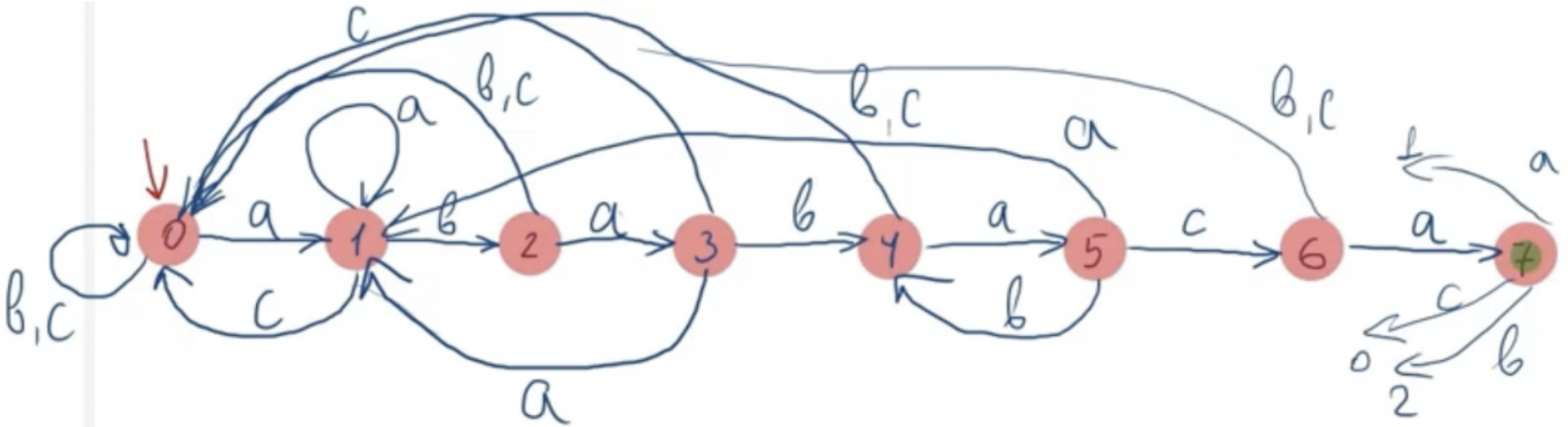
- 7 состояние - допускающее состояние
- Переходы такие же, как и у **1** состояния
- **δ(7,a)=δ(1,a)=1**
- **δ(7,b)=δ(1,b)=2**
- **δ(7,c)=δ(7,c)=0**



ИТОГОВЫЙ РЕЗУЛЬТАТ АВТОМАТА ДЛЯ СБОРКИ-ОБРАЗЦА

- **P = abababaca**




c
b,c
a
b,c
b,c
a
b,c
a
a
0
1
2
3
4
5
6
7
b,c
c
a
b
a
b
c
a
c
b
0
2

АСИМПТОТИКА

- Так как мы работаем с паттерном, то нам необходимо просмотреть каждый его символ: это занимает **O(m)**, **m** – длина паттерна

АСИМПТОТИКА

- Так как мы работаем с паттерном, то нам необходимо просмотреть каждый его символ: это занимает **O(m)**, **m** – длина паттерна
- Для каждого **i-го** символа паттерна **(i=0…m-1)** мы считаем функцию переходов

АСИМПТОТИКА

- Так как мы работаем с паттерном, то нам необходимо просмотреть каждый его символ: это занимает **O(m)**, **m** – длина паттерна
- Для каждого **i-го** символа паттерна **(i=0…m-1)** мы считаем функцию переходов
- Количество подсчетов для очередного символа зависит от размерности алфавита, так как для каждого состояния нам необходимо подсчитать функцию переходов по каждому символу

АСИМПТОТИКА

- Так как мы работаем с паттерном, то нам необходимо просмотреть каждый его символ: это занимает **O(m)**, **m** – длина паттерна
- Для каждого **i-го** символа паттерна **(i=0…m-1)** мы считаем функцию переходов
- Количество подсчетов для очередного символа зависит от размерности алфавита, так как для каждого состояния нам необходимо подсчитать функцию переходов по каждому символу
- **В итоге:** просмотрели все символы паттерна и для каждого подсчитали функции переходов

АСИМПТОТИКА

- Так как мы работаем с паттерном, то нам необходимо просмотреть каждый его символ: это занимает **O(m)**, **m** – длина паттерна
- Для каждого **i-го** символа паттерна **(i=0…m-1)** мы считаем функцию переходов
- Количество подсчетов для очередного символа зависит от размерности алфавита, так как для каждого состояния нам необходимо подсчитать функцию переходов по каждому символу
- В итоге: просмотрели все символы паттерна и для каждого подсчитали функции переходов
- **Получаем O(m ∑)**

ПОИСК ПОДСТРОКИ С ПОМОЩЬЮ АВТОМАТА

FINITE-AUTOMATON-MATCHER(T, δ, m)

```
1  n = T.length
2  q = 0
3  for i = 1 to n
4      q = δ(q, T[i])
5      if q == m
6          print "Образец найден со сдвигом" i − m
```

Функция перехода, которая возвращает номер состояния, в которое можно перейти по данному символу

ПОИСК ПОДСТРОКИ С ПОМОЩЬЮ АВТОМАТА

```
FINITE-AUTOMATON-MATCHER(T, δ, m)
1  n = T.length
2  q = 0
3  for i = 1 to n
4      q = δ(q, T[i])
5      if q == m
6          print "Образец найден со сдвигом" i − m
```

Функция перехода, которая возвращает номер состояния, в которое можно перейти по данному символу

Время работы:

- Просмотр всех символов текста – **O(n)**
- Подсчет функций переходов - **O(m ∑)**
- В итоге имеем:**O(n+m∑)**

ПОИСК ПОДСТРОКИ С ПОМОЩЬЮ АВТОМАТА

```
FINITE-AUTOMATON-MATCHER(T, δ, m)
1  n = T.length
2  q = 0
3  for i = 1 to n
4      q = δ(q, T[i])
5      if q == m
6          print "Образец найден со сдвигом" i − m
```

Функция перехода, которая возвращает номер состояния, в которое можно перейти по данному символу

Время работы:

- Просмотр всех символов текста – **O(n)**
- Подсчет функций переходов - **O(m ∑)**

Доп память:

- Префикс функция и состояния **O(2m)**
- ДКА - **O(∑ m)**

БОР

БОР

- **Другие названия:** префиксное дерево, trie (от re**trie**val)
- **Задача:** сохранить (отсортированное) множество строк
- **Идея:** автомат-дерево
- В вершине признак **терминальности** и **переходы**

БОР

- **Другие названия:** префиксное дерево, trie (от re**trie**val)
- **Задача:** сохранить (отсортированное) множество строк
- **Идея:** автомат-дерево
- В вершине признак **терминальности** и **переходы**