Алгоритми та складність

І семестр

Лекція 9

Скінченні автомати для пошуку підрядків

- Ряд алгоритмів пошуку підрядків в ході роботи здійснюють побудову скінченного автомата.
- Автомат, побудований для зразка P, сканує рядок тексту, шукаючи всі входження шаблону.
- Автомат перевіряє кожен символ тексту рівно один раз, витрачаючи на це фіксований час.
- Це дає час пошуку $\Theta(n)$, де n – розмір тексту.
- Однак для великих алфавітів час побудови автомата може виявитися значним.

Скінченні автомати

- Скінченний автомат $M=(Q, q_0, A, \Sigma, \delta)$, де
  - Q – скінченна множина станів,
  - $q_0 \in Q$ – початковий стан,
  - $A \subseteq Q$ – множина допустимих станів,
  - Σ – скінченний вхідний алфавіт,
  - δ – функція переходів вигляду $Q \times \Sigma \to Q$.
- Автомат знаходиться в стані q та зчитує символ *a*: перехід зі стану q в стан $\delta(q, a)$.
- Якщо стан $q \in A$, то автомат M *сприймає* (*допускає*) зчитаний на цей момент рядок.
- Не сприйняті вхідні дані називають *відхиленими*.

Скінченні автомати

- *Функція кінцевого стану* φ: $\Sigma^*\to Q$ повертає стан $\varphi(w)$, в який автомат переходить після зчитування рядка w.
- Отже, автомат M сприймає рядок w $\Leftrightarrow$ $\varphi(w)\in A$.
- Функція φ визначається рекурентним співвідношенням на основі функції переходів:

$$\varphi(\varepsilon) = q_0,$$

$$\varphi(wa) = \delta(\varphi(w), a) \text{ для } w\in\Sigma^*,\ a\in\Sigma.$$

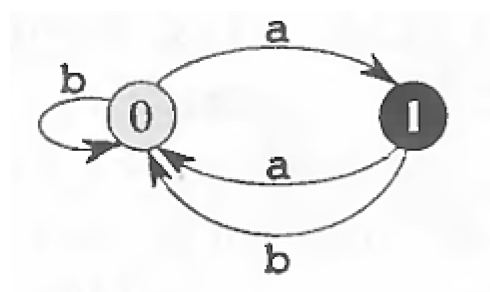
Скінченні автомати

- Приклад скінченного автомата з двома станами Q={0,1}, початковим станом q_0 та вхідним алфавітом $\Sigma=\{a,b\}$:

Вхід

Стан	a	b
0	1	0
1	0	0

Табличне представлення функції переходів δ

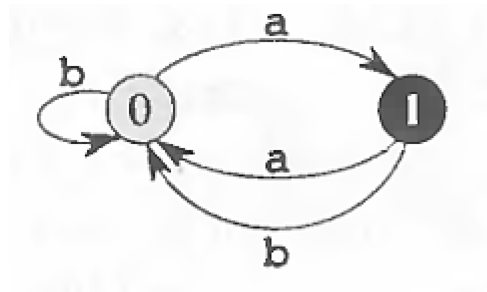


Діаграма станів

Скінченні автомати

Вхід

Стан	a	b
0	1	0
1	0	0



- Темна мітка стану 1 – допустимий (кінцевий) стан.
- Переходи представлені орієнтованими ребрами.
- Наприклад, ребро зі стану 1 в стан 0 з міткою *b* означає $\delta(1, b) = 0$.
- Автомат сприймає рядки, що закінчуються непарною кількістю символів *a*.
- Послідовність станів для рядка *abaaa* (включно з початковим): <0,1,0,1,0,1> – допустимий рядок.
- Для *abbaa*: <0,1,0,0,1,0> – відхилений рядок.

Автомати пошуку підрядків

- Для кожного шаблону на етапі попередньої обробки будується свій автомат пошуку.
- Визначимо допоміжну *суфіксну функцію* $\sigma:\Sigma^*\to\{0,1,\ldots,m\}$ для слова x:

$$\sigma(x) = \max\{k: P_k \sqsupset x\}.$$

- $\sigma(x)$ повертає довжину максимального префікса зразка P, що є суфіксом рядка x.
- Порожній рядок $P_0=\varepsilon$ є суфіксом будь-якого рядка.
- Для зразка P довжиною m: $\sigma(x) = m \Leftrightarrow P \sqsupset x$.
- З означення випливає: якщо $x \sqsupset y$, то $\sigma(x) \leq \sigma(y)$.
- Наприклад, для зразка P=*ab*: $\sigma(\varepsilon) = 0$, $\sigma(ccaca) = 1$ та $\sigma(ccab) = 2$.

Автомати пошуку підрядків

- Визначимо автомат пошуку підрядків для зразка P[1..m].
- Множина станів Q = {0, 1, …, m}, початковим станом є 0, єдиним допустимим станом є m.
- Функція переходів δ визначається для довільних стану q та символу *a*:

$$\delta(q, a) = \sigma(P_q a).$$

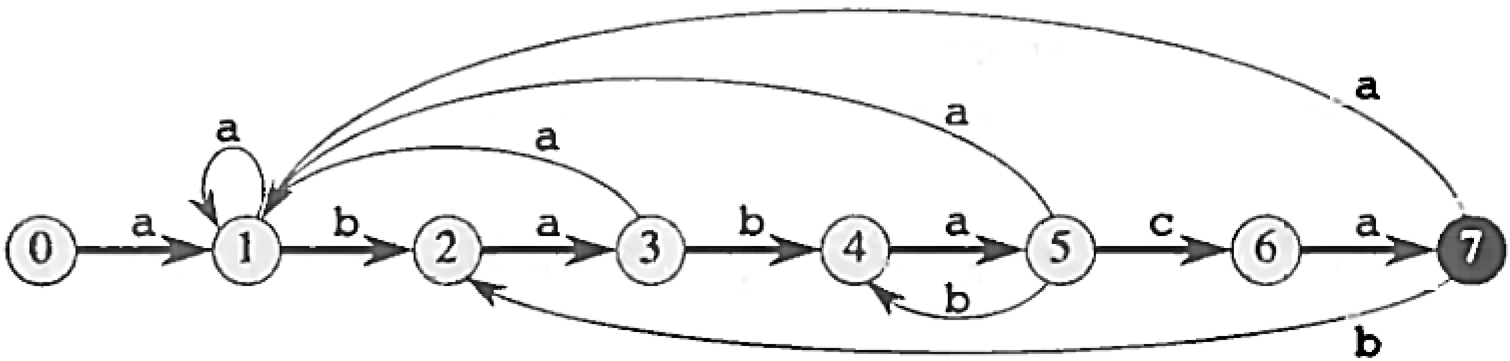
- Ми хочемо відслідкувати найдовший префікс шаблону P, який досі відповідав рядку T.

Автомати пошуку підрядків

- Щоб підрядок, що закінчується в T[i], відповідав префіксу P_j зразка, треба, щоб P_j був суфіксом T_i.
- Якщо $q = \varphi(T_i)$ (після читання T_i автомат в стані q), то функція переходів має повертати значення q таке, що $P_q \sqsupset T_i$ та $q = \sigma(T_i)$ (при q = m співпадає весь шаблон).
- Оскільки $\varphi(T_i)$ та $\sigma(T_i)$ дорівнюють q, автомат підтримує інваріант $\varphi(T_i) = \sigma(T_i)$.
- Якщо автомат перебуває в стані q та зчитується наступний символ T[i+1] = *a*, потрібно, щоб перехід вів до стану $\sigma(T_i a)$. Можна показати, що $\sigma(T_i a) = \sigma(P_q a)$.
- За умови *a* = P[q+1] символ *a* продовжує відповідати шаблону: перехід $\delta(q, a) = q + 1$.
- У випадку $a \neq$ P[q+1] символ *a* вже не відповідає зразку: пошук коротшого префіксу P, що є суфіксом T_i.

Автомати пошуку підрядків

- Приклад автомата, що сприймає всі рядки, які закінчуються на *ababaca*:



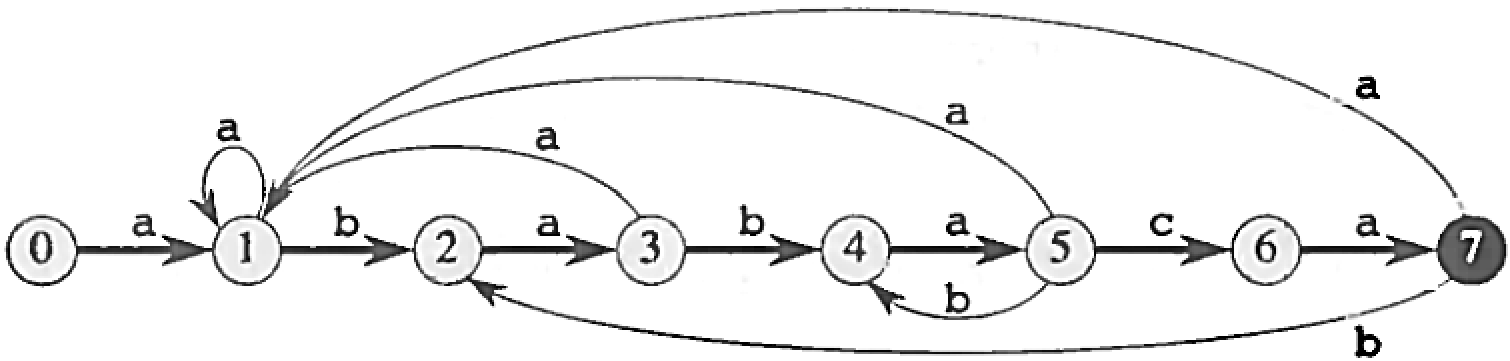
Початковий стан 0, єдиний кінцевий – 7.

Орієнтовані ребра представляють значення δ(i, *a*) = j переходів зі стану i в стан j по символу *a*.

Жирні ребра відповідають співпадінням шаблону та вхідних символів.

Автомати пошуку підрядків

- Приклад автомата, що сприймає всі рядки, які закінчуються на *ababaca*:

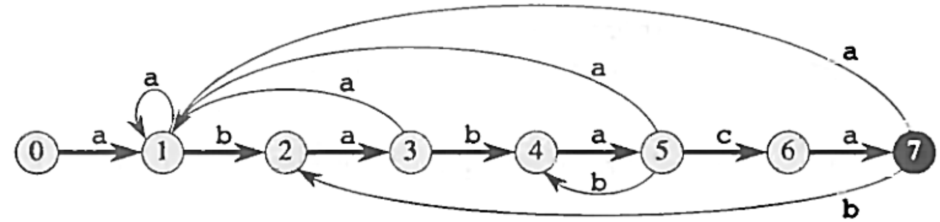


Всі направлені вліво ребра (виняток – дуги зі стану 7) відповідають неспівпадінням.

Деякі дуги, що відповідають за неспівпадіння, не зображені: якщо перехід по символу явно не вказаний, вважається, що автомат має перейти в початковий стан.

Автомати пошуку підрядків

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



i	—	1	2	3	4	5	6	7	8	9	10	11
$T[i]$	—	a	b	a	b	a	b	a	c	a	b	a
$\phi(T_i)$	0	1	2	3	4	5	4	5	6	7	2	3

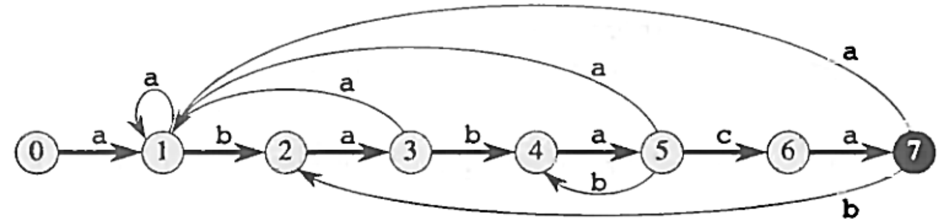
У функції переходів сірим виділені елементи, що відповідають співпадінням зразка з текстом.

Наведено приклад пошуку в тексті Т = *abababacaba*.

При співпадінні визначається позиція останнього символа входження зразка в текст.

Автомати пошуку підрядків

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



i	—	1	2	3	4	5	6	7	8	9	10	11
$T[i]$	—	a	b	a	b	a	b	a	c	a	b	a
$\phi(T_i)$	0	1	2	3	4	5	4	5	6	7	2	3

При неспівпадінні: перехід $\delta(5, b) = 4$.

Якщо автомат зчитує *b* в стані q = 5, то $P_q b$ = *ababab*. При цьому найбільшим префіксом P = *ababaca*, що одночасно є суфіксом *ababab,* є P_4 = *abab*.

Автомати пошуку підрядків

- Промоделювати роботу автомата, представленого функцією переходів δ при пошуку зразка P[1..m] у вхідному тексті T[1..n] може наступний алгоритм:

FINITE-AUTOMATON-MATCHER(T, δ, m)

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

- Час роботи пошуку Θ(n).
- Однак є ще передобробка, що обчислить δ.

Автомати пошуку підрядків

- Обчислення функції переходів δ для заданого шаблону P[1..m].

COMPUTE-TRANSITION-FUNCTION(P, Σ)

$m = P.length$
for $q = 0$ **to** m
    for каждого символа $a \in \Sigma$
        $k = \min(m+1, q+2)$
        repeat
            $k = k - 1$
        until $P_k \sqsupset P_q a$
        $\delta(q, a) = k$
return δ

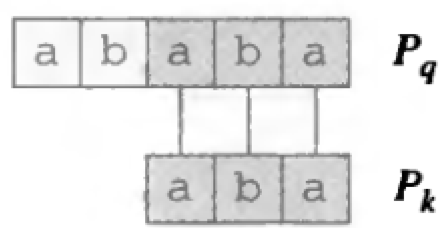
- Функція обчислюється безпосередньо за означенням.

Автомати пошуку підрядків

- Відбувається повний перебір по всіх станах q і символах *a*.
- Пошук чергового значення функції $\delta(q, a)$ – такого найбільшого k, що $P_k \sqsupset P_q a$, – починається з максимально можливого значення min(m,q+1) та зменшується в найгіршому випадку до 0.
- Час роботи $O(m^3|\Sigma|)$:
  – зовнішні цикли $m|\Sigma|$,
  – цикл **repeat** (m+1) ітерація максимум,
  – перевірка $P_k \sqsupset P_q a$ до m символів.
- Існують алгоритми, що обчислюють δ за $O(m|\Sigma|)$.

Алгоритм КПМ (нагадування)

- Алгоритм використовує принцип скінченного автомата.
- Рух шаблону по тексту та звірка зі зразком відбуваються зліва направо.
- Робота функції переходів імітується через префікс-функцію $\pi: \{1,2,\dots,m\} \to \{0,1,\dots,m-1\}$:
$$\pi[q] = \max\{k: k < q \text{ та } P_k \sqsupset P_q\}.$$
- $\pi[q]$ є довжиною найбільшого префікса зразка, який є істинним суфіксом рядка P_q:



Алгоритм КПМ (нагадування)

- Для довільних стану q = 0,1,…,m та символу $a \in \Sigma$ значення π[q] містить інформацію, необхідну для обчислення δ(q, *a*), при цьому не залежачи від *a*.
- Масив π містить лише m елементів, на відміну від масива δ, де їх $\Theta(m|\Sigma|)$.
- Тому час попередньої обробки зменшується в $|\Sigma|$ разів і складає $\Theta(m)$.
- Час пошуку залишається $\Theta(n)$.

Алгоритм КПМ (нагадування)

q	1	2	3	4	5	6
P[q]	*a*	*b*	*a*	*c*	*a*	*b*
π[q]	0	0	1	0	1	2

Задача множинного пошуку підрядків

- Нехай маємо скінченну множину рядків-зразків $P=\{P_1,\ldots,P_k\}$ («словник») та текст $T[1..m]$.
- Необхідно знайти всі входження шаблонів в текст.
- Нехай n – сума довжин всіх зразків у словнику.
- Задача може бути розв'язана за час

$$O(|P_1| + m + \ldots + |P_k| + m) = O(n + km),$$

якщо k разів застосувати звичайний алгоритм пошуку підрядка з лінійним часом.

- Але є спеціальні швидші алгоритми.
- Класичний підхід – алгоритм Ахо-Корасік (Aho-Corasick), розв'яже задачу за $O(n + m + z)$, де z – кількість входжень зразків.

Алгоритм Ахо-Корасік

- Алгоритм будує скінченний автомат на основі префіксного дерева.
- Примітне використання: база вірусів в антивірусах
- *Префіксне дерево* (trie) для множини шаблонів P є деревом з коренем K:
  - кожне ребро помічене власним символом-міткою,
  - всякі два ребра, що виходять з однієї вершини, мають різні мітки.

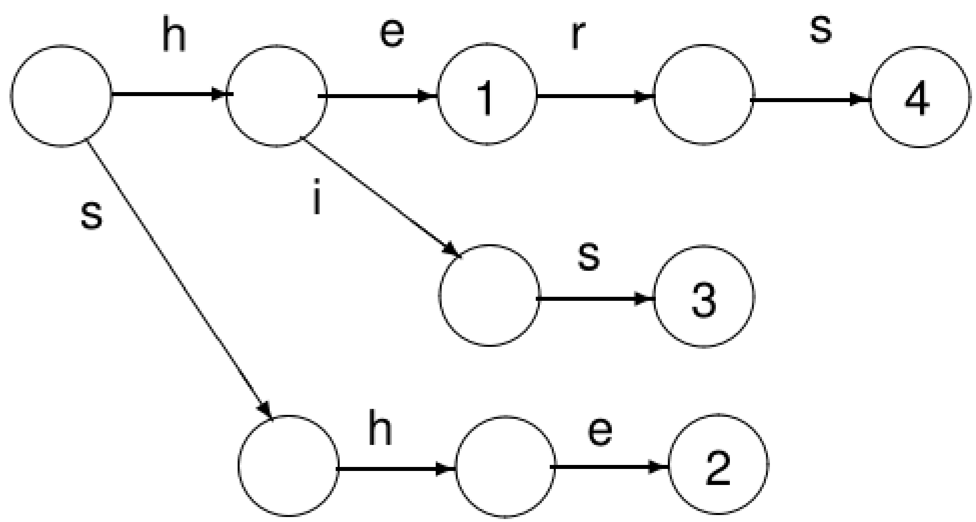
Позначимо L(v) – мітку вершини v – як конкатенацію міток ребер, які утворюють шлях з кореня до v:

- для кожного шаблону $P_i \in P$ є вершина v: $L(v)=P_i$.
- мітка кожної вершини-листа є зразком з P.

Алгоритм Ахо-Корасік

- Префіксне дерево для множини

$$P = \{he, she, his, hers\}$$



- Такий спосіб дозволяє ефективно зберігати словник рядків.

Алгоритм Ахо-Корасік

Побудова префіксного дерева

- Починаємо з однієї вершини – кореня.
- Додаємо шаблони один за одним.
  - Рух від кореня по ребрам, послідовно поміченими символами з P_i, поки можливо.
  - Якщо відсутнє ребро, відповідне черговому символу з P_i, створюємо нові ребра та вершини для решти символів P_i.
  - Якщо P_i закінчується у вершині v, зберігаємо в ній ідентифікатор P_i (наприклад, індекс i).
- Побудова відбувається за $O(|P_1|+\ldots+|P_k|) = O(n)$.

Алгоритм Ахо-Корасік

Пошук рядка S в префіксному дереві

- Починаючи від кореня, рух по ребрам, поміченим символами слова-зразка поки можливо.
- Якщо з останнім символом приходимо до вузла з ідентифікатором, слово належить до словника.
- Якщо в певний момент помічене потрібним символом ребро відсутнє – слова у словнику немає.

- Такий пошук займе час O(|S|).
- Префіксне дерево також дозволяє ефективно шукати в ньому слова.
- Для множинного пошуку за лінійний час треба перетворити префіксне дерево на автомат.

Алгоритм Ахо-Корасік

Побудова автомата

- *Стани*: вузли дерева.
- *Початковий стан*: корінь, 0.

Дії автомата визначаються трьома функціями, визначеними для всіх станів.

- Функція **goto** g(q,a) вказує, в який стан переходити з поточного стану q при перегляді символу a.
- Якщо ребро (q, v) помічене символом a, то g(q, a) = v;
- g(0, a) = 0 для всіх символів a, якими не помічене жодне ребро, що виходить з кореня. Тобто автомат залишається в корені, поки проглядаються символи, які не співпадають.
- При всіх інших аргументах g(q, a) = $\varnothing$.

Алгоритм Ахо-Корасік

Побудова автомата (далі)

- Функція **неуспіху** f(q) при $q \neq 0$ вказує, в який стан переходити при перегляді невідповідного символу.
- Розглянемо мітку вершини q і знайдемо найдовший власний суфікс w цієї мітки такий, що з нього починається деякий шаблон з множини L(w). Тоді f(q) вказує на вершину, мітка якої - цей суфікс.
- f(q) завжди визначена, бо L(0) = ε – префікс будь-якого зразка.
- Функція **виходів** out(q) повертає множину шаблонів, розпізнаних при переході в стан q.

Алгоритм Ахо-Корасік

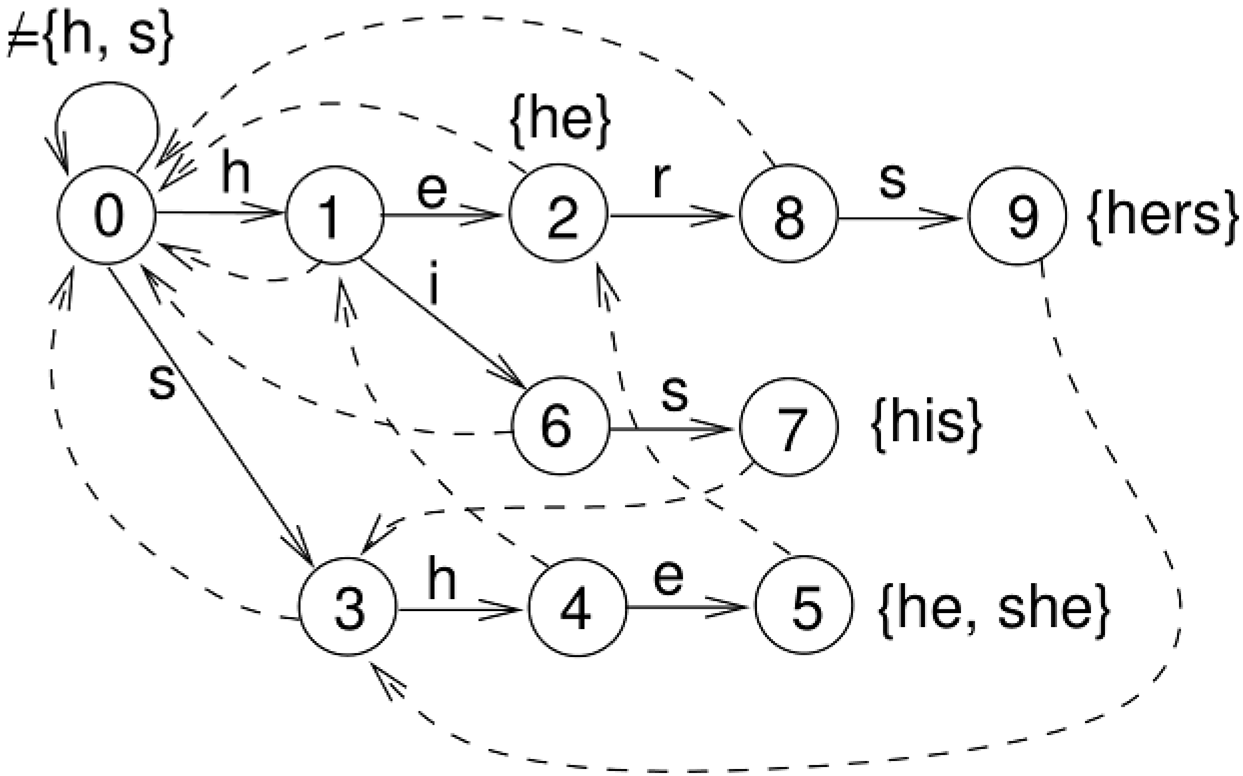
Робочий цикл автомата виглядатиме так.

- Нехай q – поточний стан автомата, a – поточний символ вхідного тексту T.

1. Якщо $g(q,a) = q_1$, автомат робить *goto-перехід*. Він переходить в стан q_1, поточним стає наступний вхідний символ T. Крім того, якщо $\text{out}(q_1) \neq \varnothing$, автомат видає множину $\text{out}(q_1)$ разом з позицією поточного вхідного символу. Завершення поточного робочого циклу.
2. Якщо $g(q,a) = \varnothing$, автомат використовує функцію неуспіху f та робить *перехід неуспіху*. У випадку $f(q) = q_1$ повтор дії зі станом q_1 та символом a як поточними.

Алгоритм Ахо-Корасік

Приклад автомата для P = {*he*, *she*, *his*, *hers*}



Пунктиром позначені суфіксні посилання – переходи при неуспіху (значення функції f).

Явно не вказані переходи, що ведуть до кореня.

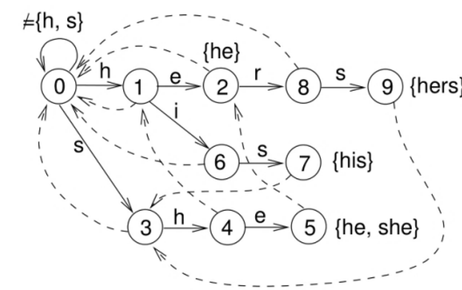
Алгоритм Ахо-Корасік

Алгоритм пошуку множини зразків у тексті T[1..m] через автомат:

$q := 0$; // initial state (root)
for $i := 1$ **to** m **do**
    while $g(q, T[i]) = \emptyset$ **do**
        $q := f(q)$; // follow a fail
    $q := g(q, T[i])$; // follow a goto
    if $\mathrm{out}(q) \neq \emptyset$ **then print** i, $\mathrm{out}(q)$;
endfor;

- Проведемо пошук зразків в тексті “*ushers*”.

Алгоритм Ахо-Корасік



u s h e r s
0 0 3 4 5 8 9
2

Розглянемо робочий цикл, коли M перебуває в стані 4 та поточний вхідний символ *e*.

- g(4, *e*) = 5 => перехід в стан 5, взяття наступного вхідного символу та видає out(5), показуючи знайдені слова “*she*” та “*he*”, що закінчуються в позиції 4 вхідного тексту.

В стані 5 вхідний символ *r*, автомат робить два переходи по станах в цьому робочому циклі.

- g(5, *r*) = $\varnothing$ => перехід в стан 2 = f(5);
- g(2, *r*) = 8 => перехід в стан 8, взяття наступного вхідного символу; вихідні дані не видаються.

Алгоритм Ахо-Корасік

Оцінимо ефективність пошуку.

- При перегляді кожного символу здійснюється один перехід по функції goto і, можливо, кілька переходів по функції неуспіху.
- Після кожного goto ми або залишаємося в корені, або переходимо в черговий стан, глибина якого на 1 більше глибини попереднього => глибина стану збільшується не більше m разів.

Алгоритм Ахо-Корасік

- Кожен перехід по функції неуспіху наближає нас до кореня (довжина мітки вершини завідомо зменшується) => кількість таких переходів не перевищить m.
- z появ шаблонів можна відслідкувати за сумарний час O(z), якщо про кожен з них повідомляти за константний час (наприклад, зберігаючи його номер і позицію в тексті).

Загалом пошук займе час O(m+z) при довжині тексту m та кількості входжень z.

Алгоритм Ахо-Корасік

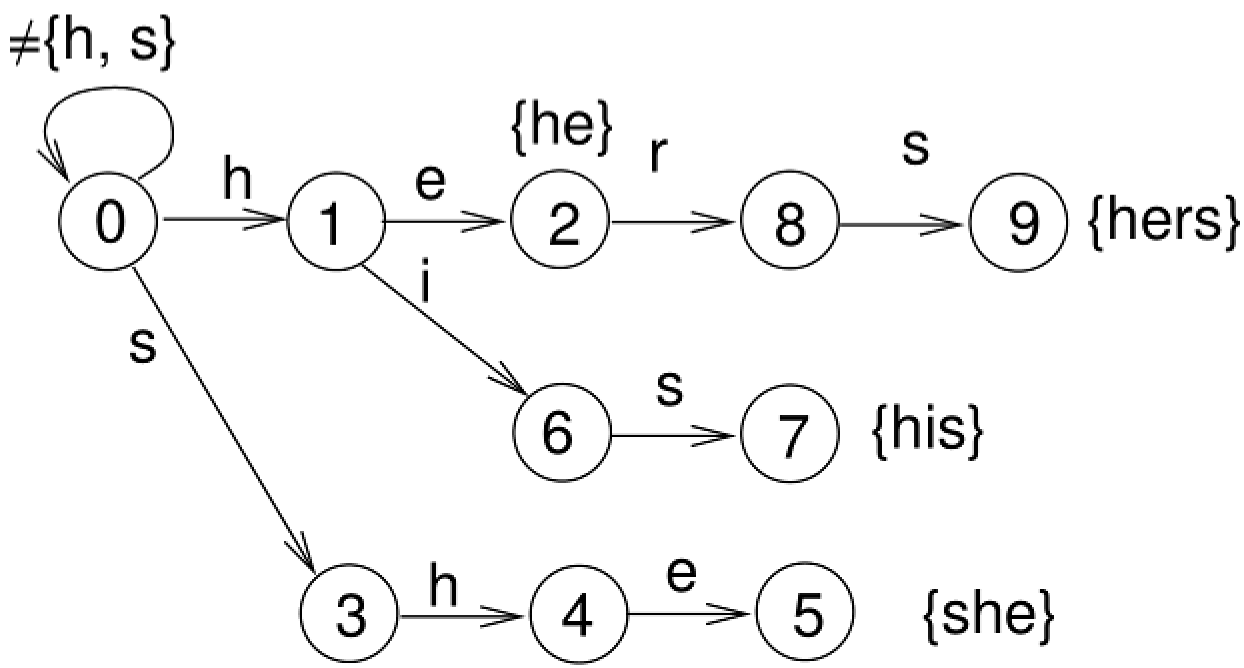
Побудова автомату відбувається в 2 етапи.

Етап I

- Будуємо префіксне дерево для словника P.
- При додаванні кожного слова P_i з P, вершині *v* з міткою P_i зіставимо out(*v*): = {P_i}.
- Завершимо побудову функції goto, додавши неіснуючі переходи з кореня: g(0, *a*) = 0 для всіх символів *a*, що не помічають жодного вихідного ребра з кореня. Це також можна зробити неявно.
- Якщо алфавіт фіксований, етап I займе O(n) часу.

Алгоритм Ахо-Корасік

Етап I – результат для P = {*he*, *she*, *his*, *hers*}



Алгоритм Ахо-Корасік

Етап II

$Q :=$ emptyQueue();

for $a \in \Sigma$ **do**

  if $q(0, a) = q \neq 0$ **then**

    $f(q) := 0$; enqueue(q, Q);

while not isEmpty(Q) **do**

  r:= dequeue(Q);

  for $a \in \Sigma$ **do**

    if $g(r, a) = u \neq \emptyset$ **then**

      enqueue(u, Q); $v := f(r)$;

      while $g(v, a) = \emptyset$ **do** $v := f(v)$; // (*)

      $f(u) := g(v, a)$;

      out(u) := out$(u) \cup$ out$(f(u))$;

Алгоритм Ахо-Корасік

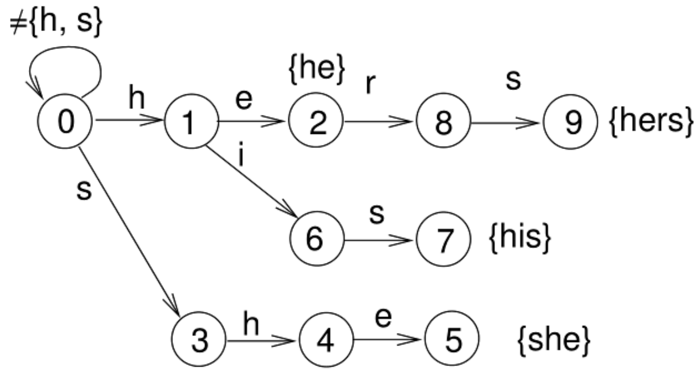
- Функція невдачі і вихідна функція обчислюються для всіх вершин в порядку обходу в ширину, тому коли ми працюємо з вершиною, всі вершини, що знаходяться ближче за неї до кореня (зокрема ті, що мають коротші мітки за поточну), вже оброблені.
- Розглянемо вузли r та u = g(r,a) (r батько u), і L(u)=L(r)a. Треба, щоб f(u) вказувала на ту вершину, мітка якої є найдовшим суфіксом L(u), що є також початком деякого шаблону з множини P. Знаходимо її шляхом проглядання вершин, мітки яких є все коротшими суфіксами L(r), поки не знайдеться вершина v, для якої g(v,a) визначена (рядок (*)); присвоїмо це значення f(u).
- Як v, так і g(v, a) можуть бути коренем.

Алгоритм Ахо-Корасік

- Потрібно також оновити значення out(*u*). Як тільки визначається значення f(*u*), зливаємо множини виходів out(*u*) та out(f(*u*)).
- Це робиться тому, що всі шаблони, які розпізнаються при переході в стан f(*u*) (і тільки вони) є власними суфіксами L(*u*) і повинні бути відслідковані при переході в стан *u*.

- Етап II теж може бути виконаний за час O(n).

Алгоритм Ахо-Корасік



i	*output* (i)
2	{he}
5	{she, he}
7	{his}
9	{hers}

i	1	2	3	4	5	6	7	8	9
$f(i)$	0	0	0	1	2	0	3	0	3

Зокрема, після визначення f(5) = 2, ми зливаємо множину виходів стану 2 ({*he*}) з множиною виходів стану 5, отримавши нову множину {*he*, *she*}.

Пошук шаблонів з масками

- Нехай φ – *маска*, що позначає будь-який одиничний символ.
- Наприклад, шаблон abφφcφ зустрічається на позиціях 2 і 7 рядка xabvccababcax (x**ab**vc**c**ababcax, xabvcc**ab**ab**c**ax)
- Якщо кількість φ обмежена зверху константою, то шаблони з масками можуть бути виявлені за лінійний час алгоритмом Ахо-Корасік. Для цього треба виявити безмаскові шматки шаблону Q.

Пошук шаблонів з масками

- Нехай $\{Q_1,...,Q_k\}$ - набір підрядків Q, розділених масками, і нехай $p_1, ..., p_k$ – їх кінцеві позиції в Q.
- Будуємо автомат Ахо-Корасік для множини Q.
- Вводиться масив C, де C[i] – кількість зустрівшихся в тексті безмаскових підрядків шаблону, що входить до тексту починаючи з позиції i.
- Тоді поява підрядка Q_j в тексті на позиції i означатиме можливу появу шаблону на позиції $i– p_j + 1$.

Пошук шаблонів з масками

- Ініціалізуємо лічильник входжень:

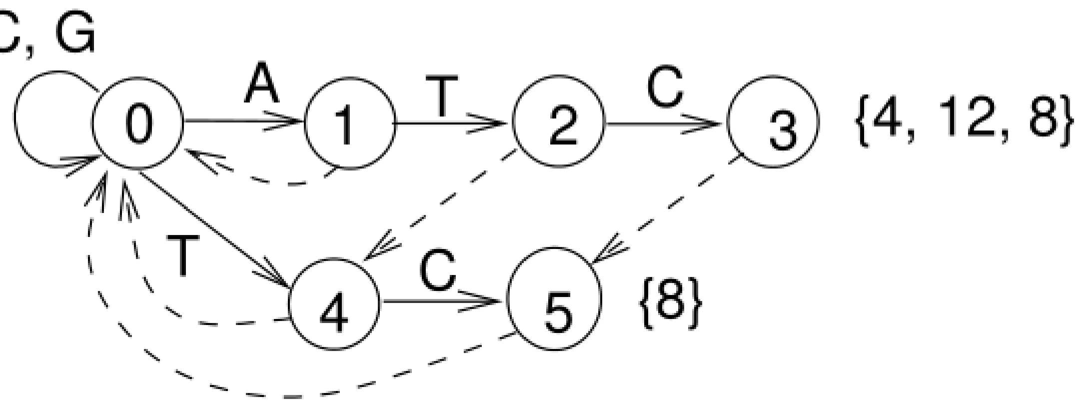
  for i: = 1 to |T| do C[i]: = 0;

- Використовуючи автомат, шукаємо підрядки Q: коли знаходимо Q_j в тексті T, що закінчується на позиції $i \geq p_j$, збільшуємо на одиницю значення $C[i - p_j + 1]$.
- Кожне i, для якого C[i] = k, є стартовою позицією появи шаблону Q.

Пошук шаблонів з масками

Приклад. Дано зразок φATCφφTCφATC.

- Q = {ATC,TC,ATC} з $p_1 = 4$, $p_2 = 8$, $p_3 = 12$.



Пошук:

```
i: 12345678901234...
T: ACGATCTCTCGATC...
```

C[1] = C[7] = C[11] = 1 та C[3] = 3 => входження.

Пошук шаблонів з масками

Оцінимо час роботи алгоритму

- Нехай $|Q| = n$, $|T| = m$.
- Передобробка: $O(m + n)$.
- Пошук: $O(m + z)$, де z - кількість входжень.
- Кожна поява підрядка Q збільшує значення однієї комірки C на 1, і значення кожної комірки може збільшуватися до k разів, тому кількість входжень $\leq km$ (або $O(m)$, якщо k обмежена константою.)
- Отже, якщо кількість φ обмежена константою, пошук шаблонів з масками виконується за час $O(|Q|+|T|)$.