Алгоритми та складність

I семестр

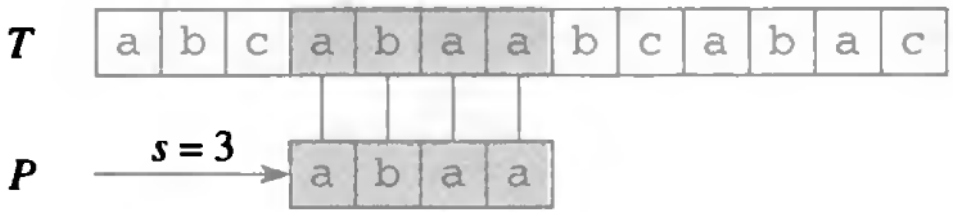
Лекція 8

Пошук підрядків

- Задача пошуку підрядка в рядку (string-matching problem) зустрічається в найрізноманітніших областях, починаючи з текстових редакторів та закінчуючи пошуковими системами в інтернеті і пошуком зразків в молекулах ДНК.
- Потрібно в тексті (символьний рядок) знайти підрядок, що відповідає рядку-шаблону.
- Тобто, вказати індекс першого зліва символу тексту, з якого починається входження шаблона (або позиції всіх таких входжень).
- Часом шаблону пошуку дають назву needle («голка»), а тексту – haystack («копиця сіна»).

Пошук підрядків

- Нехай *текст* задано масивом T[1..n] довжиною n, а *зразок* (*шаблон*) – масивом P[1..m] довжиною m ($m \le n$).
- Елементи рядків символів P і T належать до скінченного алфавіту Σ.
- Зразок P зустрічається в тексті T зі *зміщенням* s (тобто починаючи з позиції (s+1)), якщо $0 \le s \le n-m$ та T[s+1..s+m] = P[1..m] (або T[s+j] = P[j], $1 \le j \le m$).



- Якщо P зустрічається в T зі зміщенням s, то s – *допустиме* (*коректне*) зміщення, інакше зміщення s є *недопустимим* (*некоректним*).
- Задача пошуку підрядка є задачею пошуку всіх допустимих зміщень для шаблона P в тексті T.

Пошук підрядків

- Нехай розглядаємо рядки скінченної довжини.
- Σ^* – множина всіх скінченних рядків, утворених з символів алфавіту Σ.
- $\varepsilon \in \Sigma^*$ – *порожній рядок*, довжини 0 (тобто $|\varepsilon|=0$).
- xy – конкатенація рядків x та y (довжина $|x|+|y|$).
- $w \sqsubset x$ – рядок w є *префіксом* рядка x, якщо $x=wy$ для деякого $y \in \Sigma^*$. При цьому $|w| \leq |x|$.
- $w \sqsupset x$ – рядок w є *суфіксом* рядка x, якщо $x=yw$ для деякого $y \in \Sigma^*$. Аналогічно $|w| \leq |x|$.

Наприклад, $ab \sqsubset abcca$ та $cca \sqsupset abcca$.

- Порожній рядок ε завжди буде і префіксом, і суфіксом будь-якого рядка.

Пошук підрядків

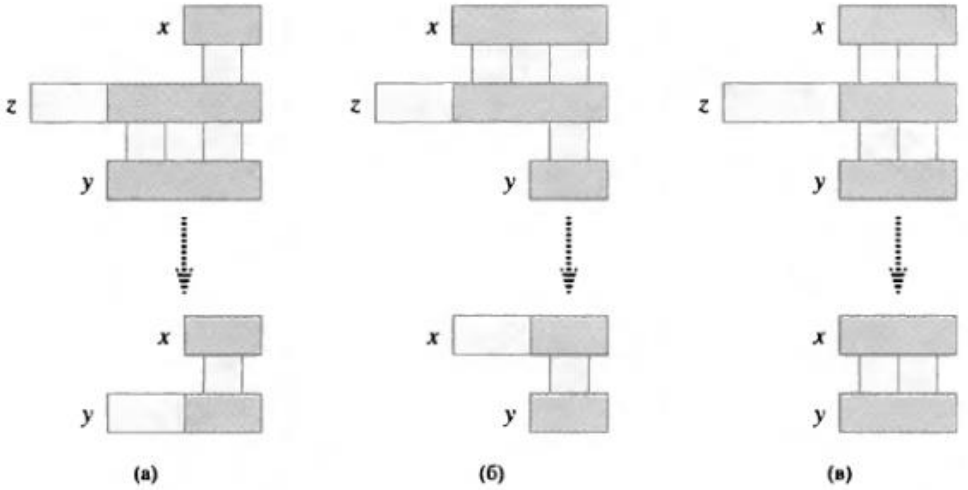
- Для довільних рядків x та y, для будь-якого символа a виконується: $x \sqsupset y \Leftrightarrow xa \sqsupset ya$.
- Відношення $\sqsubset$ та $\sqsupset$ – транзитивні.

<u>Лема про суфікси, що перекриваються</u>.
Нехай x, y, z – такі рядки, що $x \sqsupset z$ та $y \sqsupset z$. Тоді:

а) якщо $|x| \leq |y|$, то $x \sqsupset y$;

б) якщо $|x| \geq |y|$, то $y \sqsupset x$;

в) якщо $|x| = |y|$, то $x = y$.

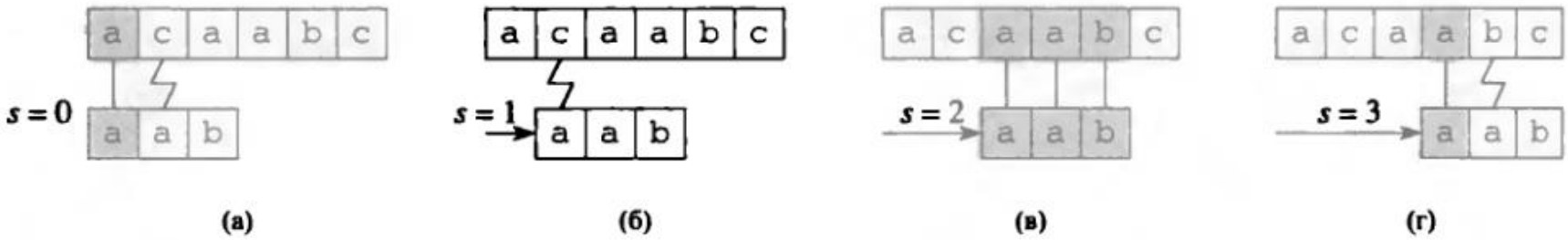


Пошук підрядків

- Розглянемо найпростіший алгоритм (груба сила).
- Вирівняємо шаблон з початком тексту і станемо зліва направо посимвольно їх порівнювати.
- Якщо всі m пар символів рівні, то входження знайдене, інакше зміщуємо зразок на один символ вправо і починаємо порівняння спочатку.
- Остання позиція зміщення, яку є сенс перевірити – (n – m + 1) при індексації масивів з одиниці.

```
NAIVE-STRING-MATCHER(T, P)
1  n = T.length
2  m = P.length
3  for s = 0 to n − m
4       if P[1..m] == T[s + 1..s + m]
5            print "Образец найден со сдвигом" s
```

Пошук підрядків



Приклад роботи алгоритму

- Рядок 4 перевіряє коректність поточного зміщення і неявно містить цикл попарної перевірки позицій.
- Цей тест виконується за час Θ(t+1), де t – кількість символів, що співпали (форма запису підкреслює, що навіть у разі неспівпадіння першого символу на перевірку витратився певний час).

Пошук підрядків

- В найгіршому випадку алгоритм виконає всі m порівнянь для кожного з (n–m+1) зміщень зразка.
- Тому загальна оцінка алгоритму O((n–m+1)m).
- Гірші випадки виникатимуть якраз при багатократних входженнях зразка.
- Приклад найгіршого випадку: нехай текст має вигляд “a^n” (n символів a), а зразок – “a^m”. Якщо $m = \lfloor n/2 \rfloor$, час роботи для найгіршого випадку стає $\Theta(n^2)$.

Пошук підрядків

- Однак загалом для пошуку слова в природомовному тексті найгірша ситуація не є типовою.
- Середня оцінка для випадкових текстів складе $\Theta(n+m) = \Theta(n)$. Типова кількість операцій $2n$.
- В алгоритмі ніяк не враховується інформація про текст, отримана для одного значення s, при розгляді наступних зміщень.
- Однак це може бути корисним. Наприклад, якщо для зразка P=aaab знайдене допустиме зміщення s=0, то зміщення 1, 2 і 3 гарантовано недопустимі.

Пошук підрядків

Існує багато алгоритмів пошуку підрядка, і при виборі слід врахувати ряд факторів.

- Чи потрібна оптимізація взагалі. Наївний алгоритм в середньому працює непогано, не потребує передобробки, і саме він реалізований в стандарнтих бібліотеках.
- Чи висока ймовірність появи поганих даних. Тоді потрібно вибирати алгоритми з кращими оцінками для найгіршого випадку.
- Чи природа тексту (мови) не вступає у протиріччя з евристикою прискорення пошуку в середньому.
- Архітектура системи іноді дозволяє швидко порівнювати дві ділянки в оперативній пам’яті, можна це використати.

Пошук підрядків

- Розмір алфавіту. Багато алгоритмів мають евристики, пов’язані з символом, що не співпав. Для великих алфавітів відповідні таблиці символів займатимуть забагато пам’яті, для малих – евристика не буде ефективною.
- Пошук прискориться, якщо є можливість проіндексувати текст.
- Чи одночасно треба шукати декілька рядків? Чи потрібен наближений пошук (fuzzy string search)? Деякі алгоритми дозволяють таке робити після нескладних модифікацій.

Алгоритм Хорспула (Horspool)

- Для пришвидшення пошуку в багатьох алгоритмах застосовується ідея покращення вхідних даних: робиться передобробка шаблону, отримана інформація зберігається в таблиці і потім використовується в процесі пошуку.
- Загальна стратегія алгоритму Хорспула: зразок рухається по тексту зліва направо, але порівняння проводиться *справа наліво*.
- В разі виявлення неспівпадіння це дозволяє просунути шаблон одразу на декілька позицій.
- Особливість алгоритму полягає у способі обчислення зсуву зразка.
- Серед подібних алгоритмів цей – найпростіший.

Алгоритм Хорспула

- Нехай в тексті шукається підрядок BARBER:

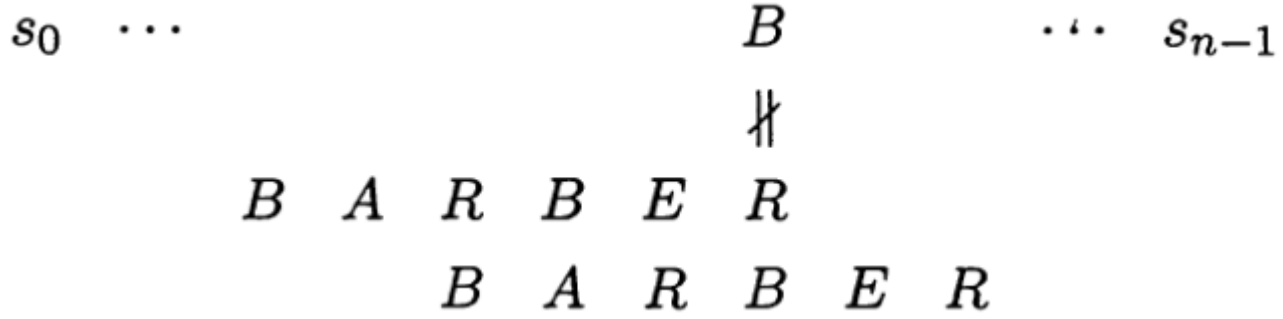
$$s_0 \quad \cdots \qquad\qquad\qquad\qquad c \quad \cdots \quad s_{n-1}$$
$$B \quad A \quad R \quad B \quad E \quad R$$

- Нехай символ тексту с лежить напроти останнього символу зразка (порівнюємо з кінця) і входження слова зараз немає. Тоді можливі чотири ситуації.
- **Випадок 1**. Якщо зразок не містить символ с (для нашого прикладу це S), шаблон можна зсунути на всю його довжину.

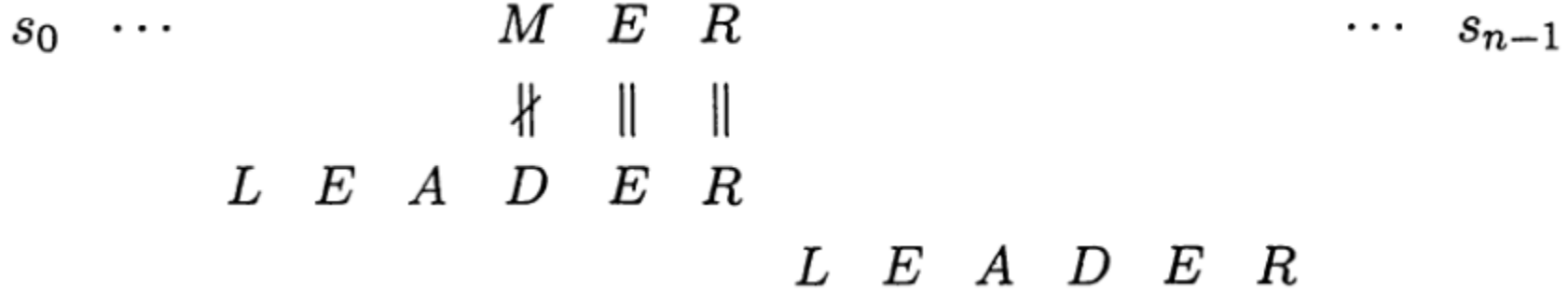
$$s_0 \quad \cdots \qquad\qquad S \qquad\qquad\qquad \cdots \quad s_{n-1}$$
$$\qquad\qquad\qquad\qquad \nparallel$$
$$B \quad A \quad R \quad B \quad E \quad R$$
$$\qquad\qquad\qquad\qquad B \quad A \quad R \quad B \quad E \quad R$$

Алгоритм Хорспула

- **Випадок 2**. Якщо зразок містить символ с, але він не останній (у нас це B), шаблон слід зсунути так, щоб напроти с було найправіше входження цього символу в шаблон.

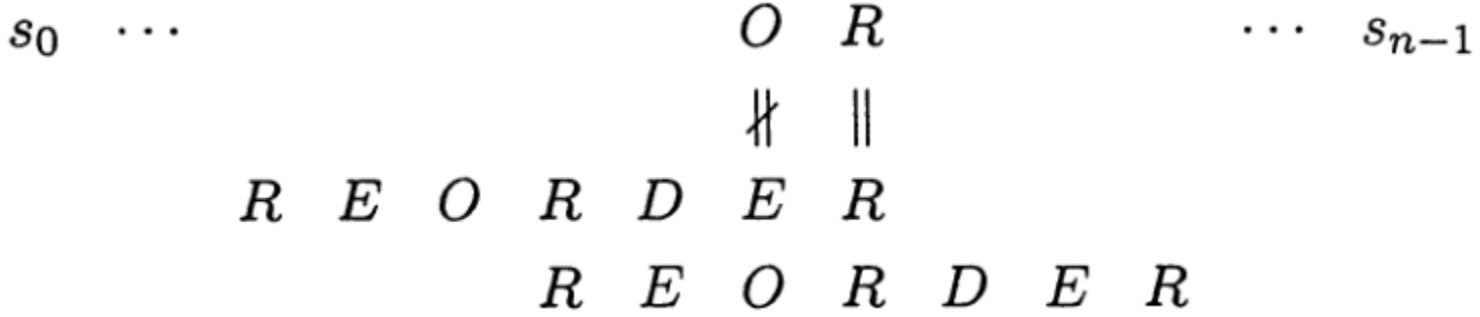


- **Випадок 3**. Якщо с – останній символ зразка і більше в шаблоні він не зустрічається, зсуваємо як у випадку 1 на всю довжину.



Алгоритм Хорспула

- **Випадок 4**. Якщо c – останній символ зразка і серед інших (m–1) символів шаблону є інші його входження, зсуваємо як у випадку 2: входження символу c, найправіше серед решти інших входжень c у зразку, має бути напроти цього символу в тексті.



Алгоритм Хорспула

- Можна помітити, що порівняння символів справа наліво може привести до більших зсувів, ніж в наївному алгоритмі.
- Однак перевага буде втрачена, якщо при кожній перевірці переглядати всі символи зразка.
- Пропонується зберігати в спеціальній таблиці попередньо обчислені величини зсувів для кожного з можливих символів тексту.
- Для кожного $c \in \Sigma$ визначається

$$t(c) = \begin{cases} \text{довжина зразка } m, \text{ якщо } c \text{ немає серед перших } (m-1) \text{ символів шаблону;} \\ \text{відстань від найправішого символа } c \text{ серед перших } (m-1) \text{ символів шаблону до останнього символа зразка, інакше.} \end{cases}$$

- Наприклад, для шаблону BARBER всі елементи таблиці будуть дорівнювати 6, окрім t(E)=1, t(B)=2, t(R)=3, t(A)=4.

Алгоритм Хорспула

АЛГОРИТМ $ShiftTable\,(P\,[0..m-1])$

// **Входные данные:** Образец $P[0..m-1]$ и алфавит символов
// **Выходные данные:** Таблица $Table[0..size-1]$,
// индексированная символами алфавита
// и заполненная величинами сдвигов
Инициализация всех элементов $Table$ значениями m
for $j \leftarrow 0$ **to** $m-2$ **do**
  $Table[P[j]] \leftarrow m-1-j$
return $Table$

- Спочатку всі значення в таблиці ініціалізуються довжиною зразка m.
- Потім шаблон переглядається зліва направо до передостаннього символа, і відповідний елемент таблиці заповнюється відстанню цього символу до правого кінця зразка.
- Якщо шаблон містить повтори символів, то за рахунок проходу зліва направо в кінці отримуємо шукані відстані від найправіших входжень.
- Ця таблиця також буде використана в алгоритмі Боєра-Мура.

Алгоритм Хорспула

- Загальна схема алгоритму Хорспула

Крок 1. Для заданого шаблону довжини m та відомого алфавіту будується таблиця зсувів.

Крок 2. Вирівнюємо початок зразка з початком тексту.

Крок 3. Поки не знайдеться шуканий підрядок чи шаблон не досягне кінця тексту, повторюємо: починаючи з кінця, порівнюємо відповідні символи зразка і тексту, поки не встановимо співпадіння всіх m символів (успішне завершення пошуку) або знайдеться пара різних символів; в цьому випадку зсуваємо шаблон на t(c) символів вправо, де c – символ тексту напроти останнього символу зразка.

Алгоритм Хорспула

АЛГОРИТМ $HorspoolMatching\,(P\,[0..m-1]\,,T\,[0..n-1])$

// Реализация алгоритма Хорспула поиска подстрок
// **Входные данные:** Образец $P[0..m-1]$ и текст $T[0..n-1]$
// **Выходные данные:** Индекс левого конца первой найденной
// подстроки или -1, если искомой подстроки
// в тексте нет
$ShiftTable(P[0..m-1])$ // Генерация таблицы сдвигов $Table$
$i \leftarrow m-1$ // Позиция правого конца образца
while $i \leqslant n-1$ **do**
  $k \leftarrow 0$ // Количество совпадающих символов
  while $k \leqslant m-1$ **and** $P[m-1-k] = T[i-k]$ **do**
    $k \leftarrow k+1$
  if $k = m$
    return $i-m+1$
  else
    $i \leftarrow i + Table[T[i]]$
return -1

Алгоритм Хорспула

Приклад. Проілюструємо пошук зразка BARBER в тексті з латинських літер і пробілів (позначені для зручності підкресленнями).

Таблиця зсувів:

Символ c	A	B	C	D	E	F	...	R	...	Z	_
Зсув $t(c)$	4	2	6	6	1	6	6	3	6	6	6

Вигляд реального пошуку в конкретному тексті:

J I M _ S A W _ M E _ I N _ A _ B A R B E R S H O P
B A R B E R
B A R B E R
B A R B E R
B A R B E R
B A R B E R

Алгоритм Хорспула

- Оцінка складності алгоритму для найгіршого випадку Θ(nm).
- Для випадкових текстів час роботи Θ(n).
- Передобробка виконується за час Θ(m+|Σ|).
- Хоча оцінки мають той самий порядок, що і в наївному алгоритмі, середня ефективність алгоритму краща: типова кількість операцій близька до 2n/|Σ|.
- Алгоритм Хорспула є спрощеною версією алгоритма Боєра-Мура, але при цьому працює навіть краще його на випадкових текстах (тим краще, чим більше різних символів у тексті). Однак на поганих даних регресує дещо частіше.

Алгоритм Райти (Raita)

- Також належить до сімейства алгоритмів типу Боєра-Мура, є покращенням алгоритму Хорспула для англійських текстів.
- Його відрізняє особливий спосіб порівняння символів шаблону і тексту.
- Назвемо *вікном* ту частину тексту, яка в поточний момент звіряється зі зразком.

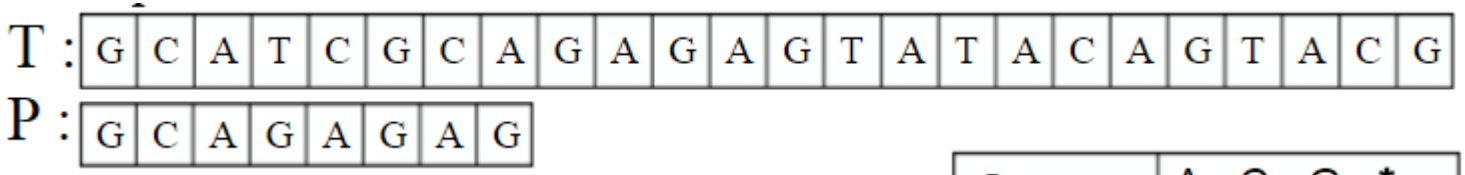
1. Порівнюються останній символ зразка та найправіший символ вікна.
2. Порівнюються перший символ зразка та найлівіший символ вікна.
3. Порівнюється середній символ зразка та вікна.
4. Якщо попередні порівняння успішні, відбувається посимвольна звірка справа наліво з передостаннього до другого символу.
5. В разі виникнення неспівпадіння на будь-якій стадії, відбувається зсув шаблону за Хорспулом.

Алгоритм Райти

- Оцінки роботи алгоритму співпадають з оцінками алгоритму Хорспула.
- В середньому кількість операцій буде < n.
- Спосіб порівняння дозволяє ефективніше шукати зокрема слова з суфіксами –ion чи –ed, які широко розповсюджені в англійській мові. Зрозуміло, при звичайному порівнянні справа наліво прийдеться повністю проходити по таким суфіксам.
- Швидкодія зростатиме зі збільшенням довжини зразка, оскільки буде зменшуватися вклад трьох порівнянь символів.
- Спадання ефективності відбувається зі зменшенням алфавіту, але на практиці це може бути помітним лише для зовсім малих алфавітів.

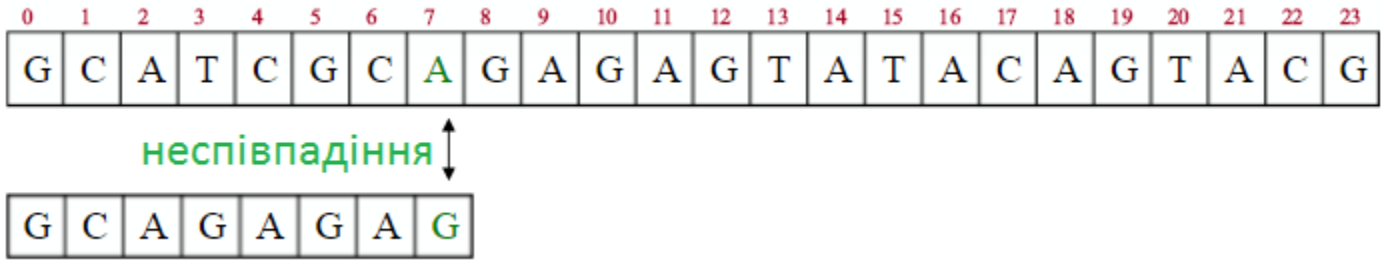
Алгоритм Райти

- Приклад.

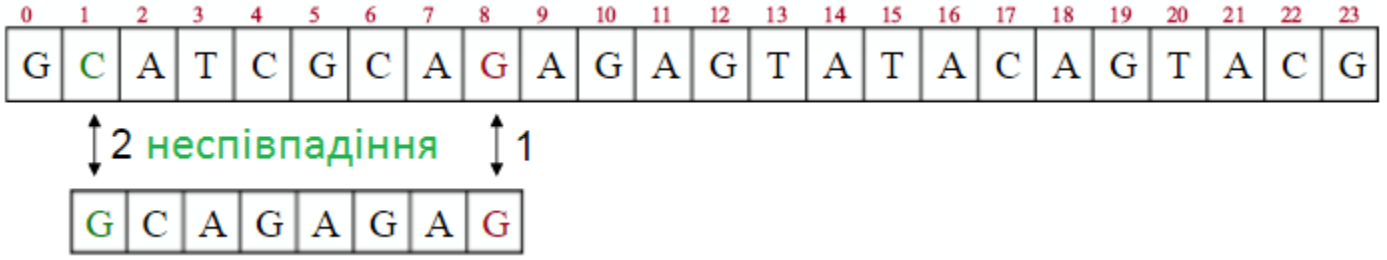


Символ	A	C	G	*
Зсув	1	6	2	8

(1)



(2)

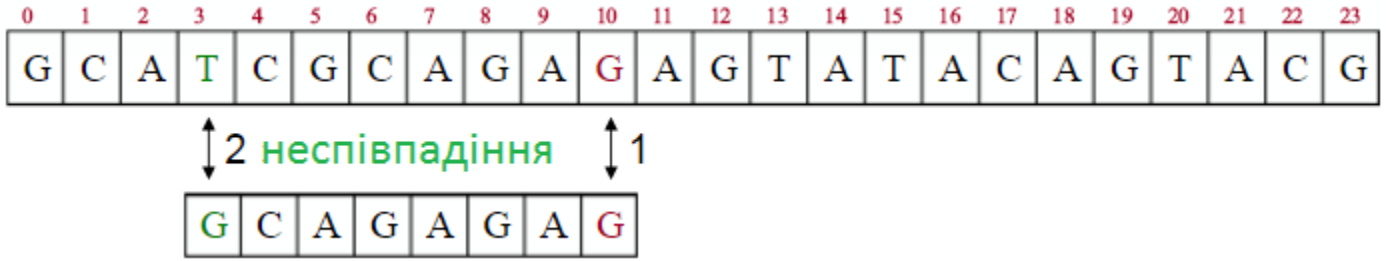


Алгоритм Райти

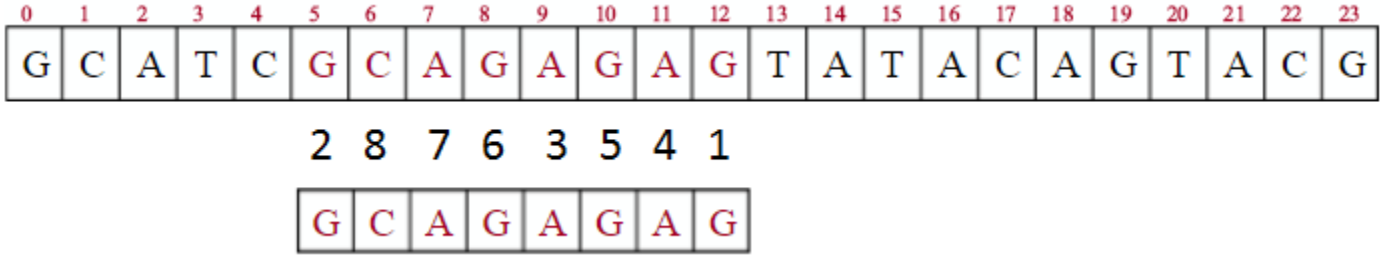
- Приклад (далі).

Символ	A	C	G	*
Зсув	1	6	2	8

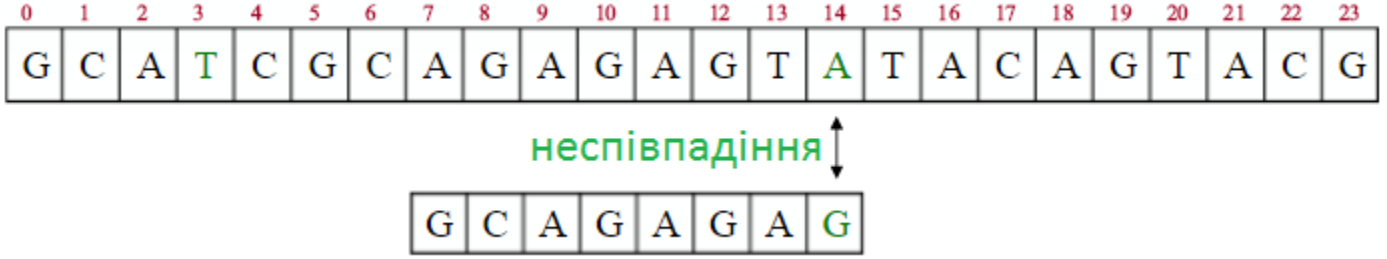
(3)



(4)



(5)

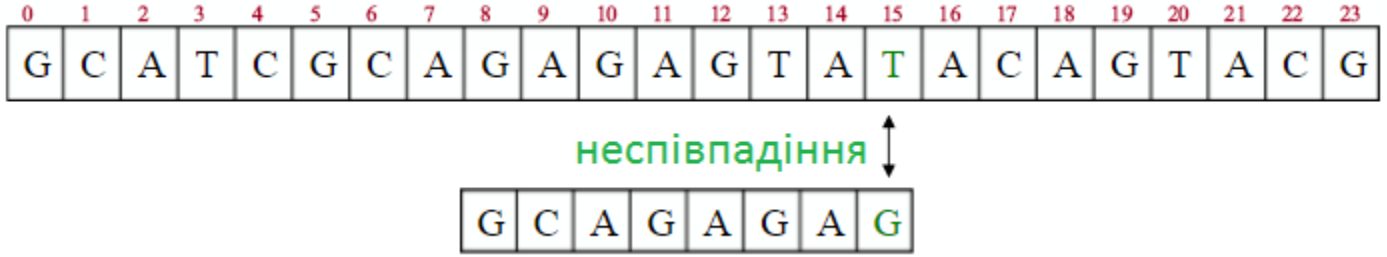


Алгоритм Райти

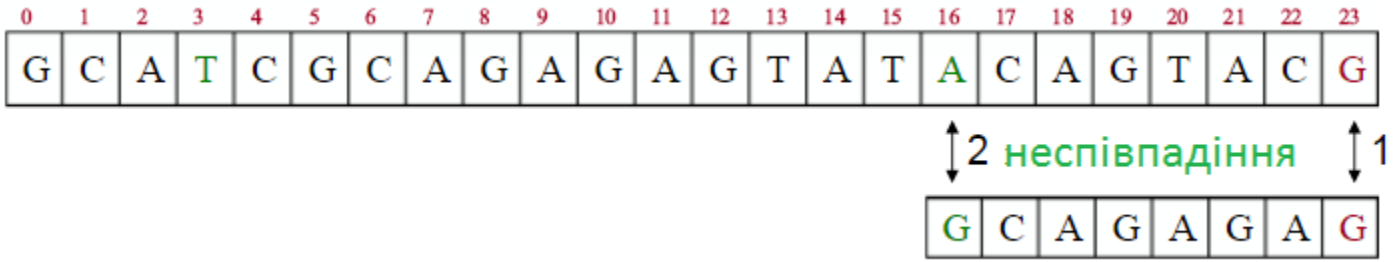
- Приклад (завершення).

Символ	A	C	G	*
Зсув	1	6	2	8

(6)



(7)

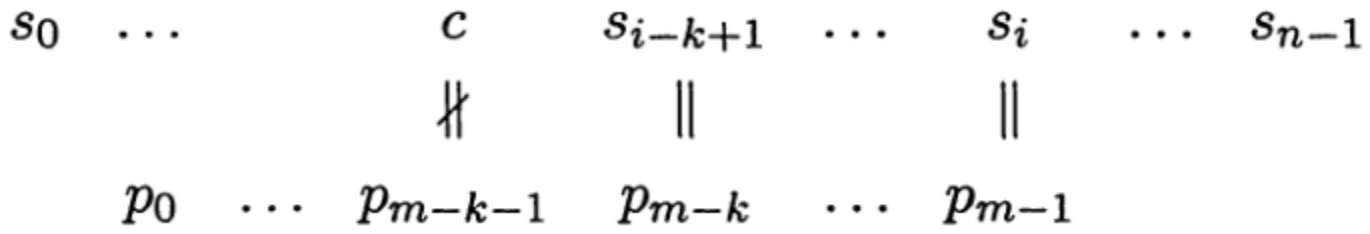


Алгоритм Боєра-Мура (Boyer-Moore)

- Алгоритм вважається стандартом при пошуку підрядка і є найефективнішим з алгоритмів загального призначення.
- В найкращому випадку працює за час $\Omega(n/m)$.
- Зазвичай час роботи сублінійний.
- Найгірший випадок: O(nm), якщо є співпадіння та O(n+m), якщо збігів немає.
- При оптимізації з використанням правила Галіла (the Galil rule) найгірший час роботи O(n+m).
- Порівняння зі зразком відбувається справа наліво.
- При обчисленні зсуву шаблону алгоритм враховує інформацію про частину, що співпала.

Алгоритм Боєра-Мура

- Якщо перше порівняння символів неуспішне, шаблон зсувається вправо на t(c) (за Хорспулом).
- Нехай є k співпадінь символів зразка з текстом (0<k<m) та символ c – перший, що відрізняється:

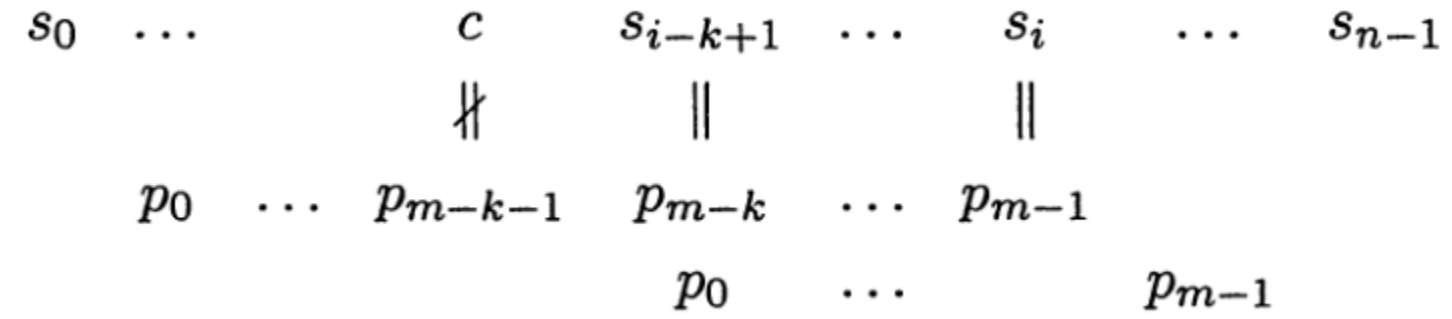


- Величина зсуву вибирається з урахуванням двох величин:
  - *зсув неспівпадаючого символу* d_1 (bad-symbol shift);
  - *зсув співпадаючого суфікса* d_2 (good-suffix shift).

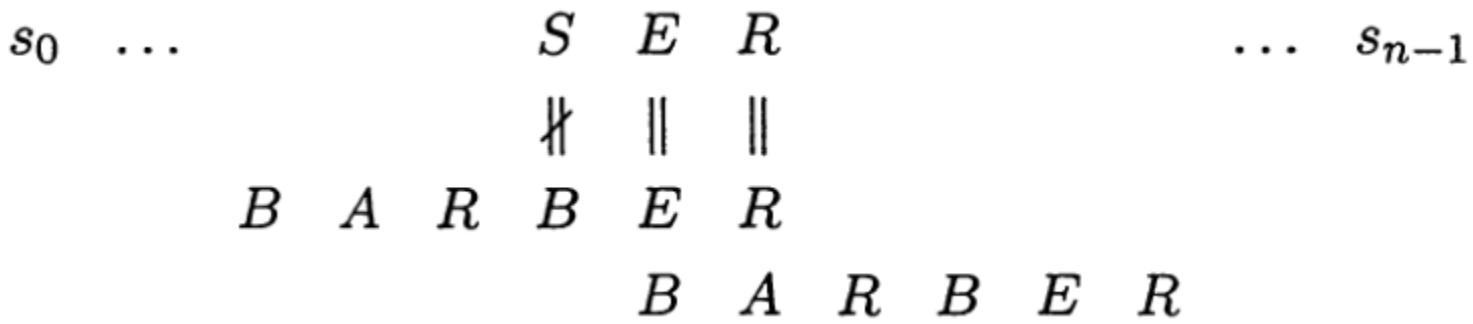
Алгоритм Боєра-Мура

Зсув неспівпадаючого символу.

- Якщо шаблон не містить символу c, зсуваємо його вправо на (t(c) – k) символів, щоб c вийшов за межі зразка:



- Наприклад, тут зсув буде на t(S)–2=6–2=4 позиції:



Алгоритм Боєра-Мура

Зсув неспівпадаючого символу.

- Зсув обчислиться так само, якщо шаблон містить символ c та при цьому t(c) – k > 0.
- Наприклад, тут зсув буде на t(A)–2=4–2=2 позиції:

- У випадку t(c) – k ≤ 0 просто зсуваємо шаблон на одну позицію вправо.
- Загалом, перший варіант зсуву d_1 визначається так: $d_1 = \max\{t(c) - k, 1\}$.

Алгоритм Боєра-Мура

Зсув співпадаючого суфікса.

- Позначимо як *suff*(k) суфікс зразка довжиною k (це та частина, що співпала).
- Припустимо, в шаблоні зустрічається ще одна послідовність *suff*(k) та їй передує символ, відмінний від c (інакше маємо повтор ситуації).
- Тоді зразок можна зсунути на відповідну відстань до найправішої такої послідовності.
- Наприклад, для шаблону ABCBAB:

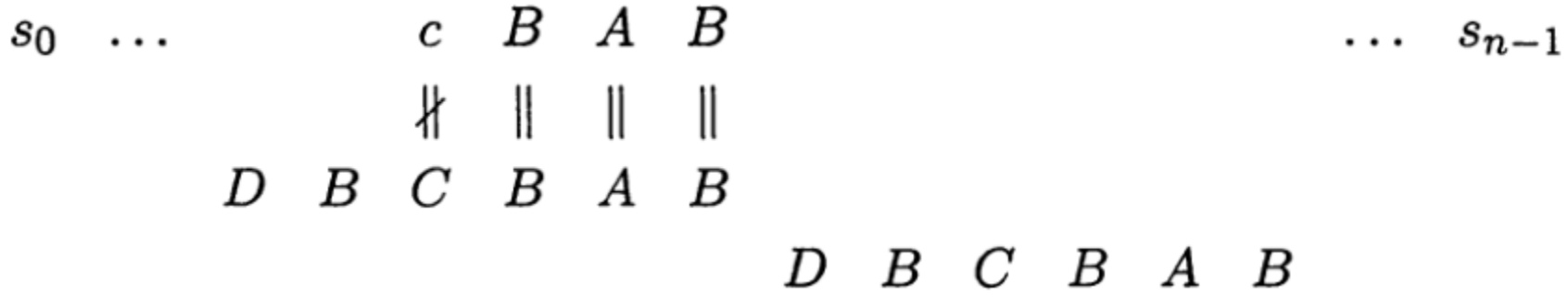
при k = 1 значення зсуву d_2 буде 2 ($ABC\overline{B}A\underline{B}$),

при k = 2 значення зсуву d_2 буде 4 ($\overline{AB}CB\underline{AB}$).

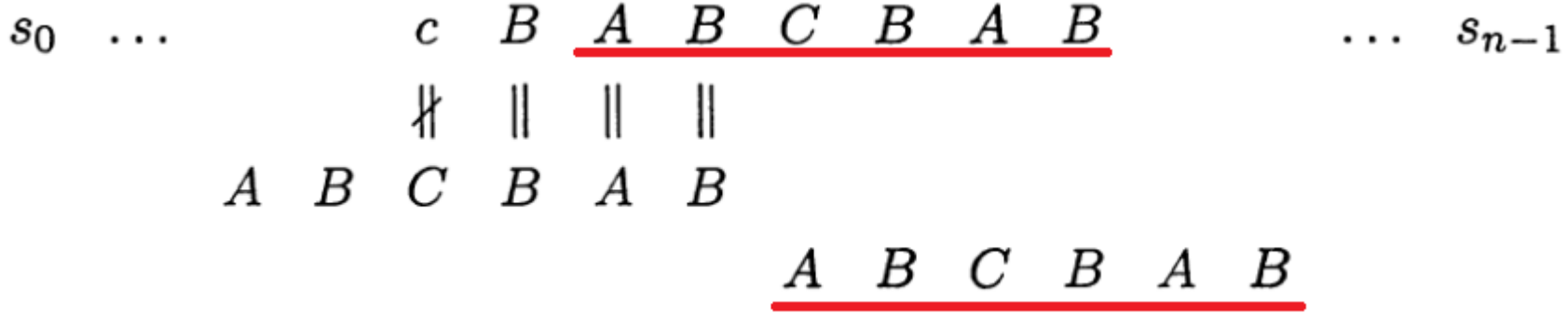
Алгоритм Боєра-Мура

Зсув співпадаючого суфікса.

- А якщо в зразку подібних входжень *suff*(k) немає?
- Спробуємо зсунути шаблон на всю довжину m:



- Але це не завжди дасть коректний результат!



Алгоритм Боєра-Мура

Зсув співпадаючого суфікса.

- Можна помітити, що в останньому зразку ABCBAB є префікс AB, що співпадає з суфіксом.
- Тому в шаблоні слід спочатку шукати найбільший префікс довжиною p<k, який співпадає з суфіксом аналогічної довжини p.
- Якщо такий префікс є, значенням d_2 буде відстань між префіксом і суфіксом, інакше $d_2 = m$.
- Для прикладу розглянемо таблицю співпадаючих суфіксів для зразка ABCBAB:

k	Зразок	d_2
1	ABCBAB	2
2	ABCBAB	4
3	ABCBAB	4
4	ABCBAB	4
5	ABCBAB	4

Алгоритм Боєра-Мура

- Загальна схема алгоритму Боєра-Мура

Крок 1. Для заданого шаблону довжини m та відомого алфавіту будується таблиця зсувів неспівпадаючих символів.

Крок 2. Для заданого шаблону будується таблиця зсувів співпадаючих суфіксів.

Крок 3. Вирівнюємо початок зразка з початком тексту.

Крок 4. Поки не знайдеться шуканий підрядок чи шаблон не досягне кінця тексту, повторюємо: починаючи з кінця, порівнюємо відповідні символи зразка і тексту, поки не встановимо співпадіння всіх m символів (успішне завершення пошуку) або знайдеться пара різних символів після $k \geq 0$ символів, що співпали.

Алгоритм Боєра-Мура

Крок 4 (кінець). В останньому випадку знаходимо d_1 з використанням значення t(c) з таблиці зсувів неспівпадаючих символів, де c – символ тексту, що не співпав. Якщо k > 0, також вибираємо відповідне d_2 з таблиці співпадаючих суфіксів. Визначаємо зсув:

$$d = \begin{cases} d_1 = \max\{t(c) - k, 1\}, & \text{при } k = 0, \\ \max\{d_1, d_2\}, & \text{при } k > 0. \end{cases}$$

- Зсув на максимальне з двох значень коректний, бо результат по кожному з випадків гарантує, що менші зсуви не приведуть до отримання шуканого підрядка.

Алгоритм Боєра-Мура

Приклад. Пошук зразка BAOBAB в тексті з латинських літер і пробілів.

- Таблиця зсувів неспівпадаючих символів:

c	A	B	C	D	...	O	...	Z	_
$t(c)$	1	2	6	6	6	3	6	6	6

- Таблиця зсувів співпадаючих суфіксів:

k	Зразок	d_2	k	Зразок	d_2
1	$BAO\overline{B}A\underline{B}$	2	4	$\overline{B}A\underline{OBAB}$	5
2	$\overline{B}AOB\underline{AB}$	5	5	$\overline{B}\underline{AOBAB}$	5
3	$\overline{B}AO\underline{BAB}$	5			

- Перебіг пошуку:

$B\ E\ S\ S\ \_\ K\ N\ E\ W\ \_\ A\ B\ O\ U\ T\ \_\ B\ A\ O\ B\ A\ B\ S$

$B\ A\ O\ B\ A\ B$

$d_1 = t(K) - 0 = 6$

$B\ A\ O\ B\ A\ B$

$d_1 = t(\_) - 2 = 4$

$d_2 = 5$

$d = \max\{4, 5\} = 5$

$B\ A\ O\ B\ A\ B$

$d_1 = t(\_) - 1 = 5$

$d_2 = 2$

$d = \max\{5, 2\} = 5$

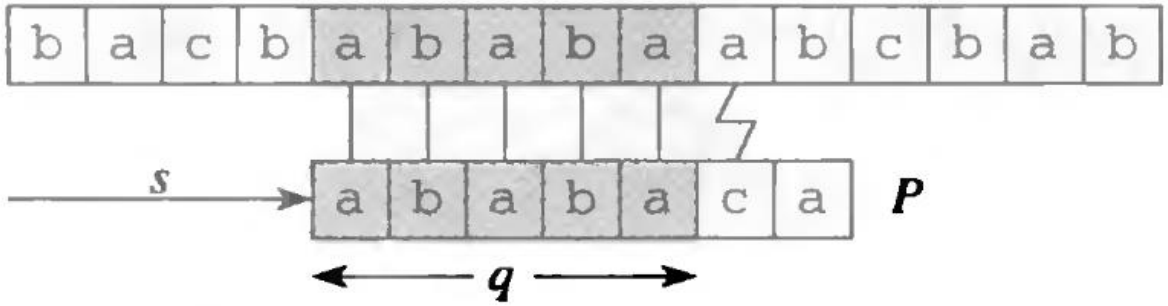
$B\ A\ O\ B\ A\ B$

Алгоритм Кнута-Морріса-Пратта

- Алгоритм КМП (Knuth-Morris-Pratt).
- Один з найвідоміших алгоритмів пошуку підрядка.
- Має лінійну оцінку в найгіршому випадку (що компенсується не настільки високою ефективністю в середньому).
- На практиці метод використовується рідко, але є основою алгоритма Ахо-Корасік, що дозволяє знаходити входження зразка з заданого набору.
- Належить до алгоритмів, що проводять порівняння зліва направо.

Алгоритм Кнута-Морріса-Пратта

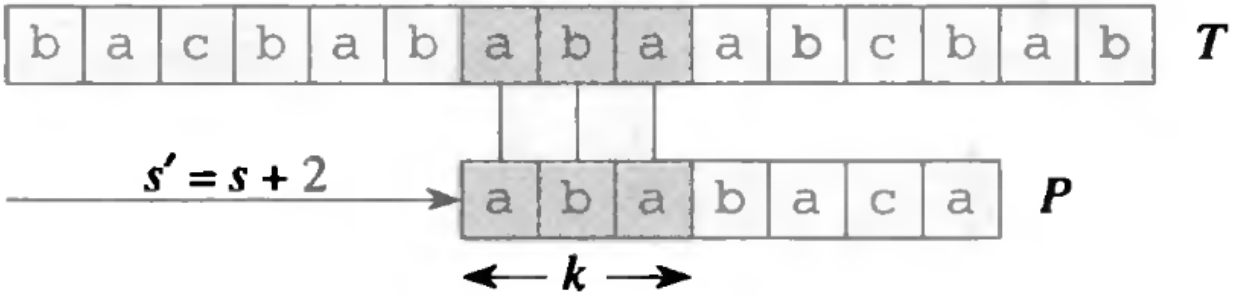
- Як і в попередніх непримітивних алгоритмах, КМП використовує інформацію про символи, що співпали, для обчислення зсуву.
- Нехай ми знаходимося в процесі перевірки зразка abababca при зміщенні s, при цьому q=5 символів співпали, а шостий відрізняється:



- Інформація про довжину ділянки збігу дозволяє дізнатися, які символи містить відповідна частина тексту, і визначити недопустимість певних зсувів.

Алгоритм Кнута-Морріса-Пратта

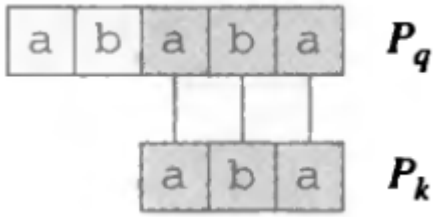
- Наприклад, тут зсув (s+1) був явно недопустимий:



- Як при цьому визначити допустимість зсуву (s+2)?
- Якщо символи P[1..q] шаблону відповідають символам T[s+1..s+q] тексту, то треба знайти найменше зміщення s'>s таке, що для деякого k<q P[1..k] = T[s'+1..s'+k], де s'+ k = s + q.
- Тобто, знаючи, що $P_q \sqsupset T_{s+q}$ необхідно знайти найдовший істинний префікс P_k рядка P_q, який одночасно буде суфіксом T_{s+q}.

Алгоритм Кнута-Морріса-Пратта

- Нове зміщення отримується як s’ = s + (q – k).
- В найкращому випадку k = 0 і s’ = s + q, але в будь-якому разі вже не потрібно порівнювати перші k символів зразка, бо вони гарантовано співпадають.
- Інформацію про зсув можна обчислити порівнюючи шаблон сам з собою:



- Шукається максимальне k<q, для якого $P_k \sqsupset P_q$.
- Зручніше зберігати для кожного q кількість k символів, що співпадають при новому зміщенні s’.

Алгоритм Кнута-Морріса-Пратта

- Для заданого шаблону P[1..q] визначимо *префікс-функцію* $\pi: \{1,2,\ldots,m\} \to \{0,1,\ldots,m-1\}$ так:
$$\pi[q] = \max\{k: k < q \text{ та } P_k \sqsupset P_q\}.$$
- Тобто, $\pi[q]$ є довжиною найбільшого префікса зразка, який є істинним суфіксом рядка P_q.
- Наприклад, префікс-функція для шаблону ababaca: [0, 0, 1, 2, 3, 0, 1]:

“a”, “ab” не містять нетривіального префікса = суфіксу

“aba”: префікс довжини 1 співпадає з суфіксом

“abab” : префікс довжини 2 співпадає з суфіксом

“ababa” : префікс довжини 3 співпадає з суфіксом

“ababac” не містить нетрив. префікса = суфіксу

“ababaca” : префікс довжини 1 співпадає з суфіксом

Алгоритм Кнута-Морріса-Пратта

Схема алгоритму обчислення префікс-функції.

- Значення $\pi[i]$ обчислюємо від i=1 до i=m–1 (при цьому $\pi[0] = 0$).
- Для обчислення $\pi[i]$ вводиться j – довжина поточного зразка, який розглядається. Спочатку $j = \pi[i-1]$.
- Тестуємо зразок довжини j, порівнюючи символи s[j] та s[i]. Якщо вони співпадають: $\pi[i] = j + 1$ та перехід до наступної ітерації (i + 1). Інакше: зменшуємо j: $j = \pi[i-1]$ та продовжуємо тестування.
- Якщо дійшли до j = 0 не знайшовши співпадінь: присвоєння $\pi[i] = 0$ та перехід на ітерацію (i + 1).

Алгоритм Кнута-Морріса-Пратта

```
KMP_MATCHER(T, P)
 1  n ← length[T]
 2  m ← length[P]
 3  π ← COMPUTE_PREFIX_FUNCTION(P)
 4  q ← 0                                  ▷ Число совпавших символов
 5  for i ← 1 to n                         ▷ Сканирование текста слева направо
 6        do while q > 0 и P[q + 1] ≠ T[i]
 7              do q ← π[q]                ▷ Следующий символ не совпадает
 8           if P[q + 1] = T[i]
 9              then q ← q + 1             ▷ Следующий символ совпадает
10           if q = m                      ▷ Совпали ли все символы образца P?
11              then print "Образец обнаружен при сдвиге" i − m
12                   q ← π[q]              ▷ Поиск следующего совпадения
```