Міністерство освіти і науки України
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «КИЄВО-МОГИЛЯНСЬКА АКАДЕМІЯ»
Факультет інформатики
Кафедра мережних технологій факультету інформатики

Кваліфікаційна робота

освітній ступінь – бакалавр

на тему: «**Чисельні характеристики побудови узагальнених FM-index для алфавітів різних розмірів**»

Виконав: студент 4-го року навчання,

Спеціальності

121 Інженерія програмного забезпечення

Фомін Володимир Олегович

Керівник: Зважій Д. В.

асистент

Рецензент _____
(прізвище та ініціали)

Кваліфікаційна робота захищена
з оцінкою _____
Секретар ЕК_____
“_____” _____ 2024 р.

Міністерство освіти і науки України

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «КИЄВО-МОГИЛЯНСЬКА АКАДЕМІЯ»

Факультет інформатики

Кафедра мережних технологій факультету інформатики

ЗАТВЕРДЖУЮ

Зав. кафедри мережних технологій,

_____

“___” _____ 2024 р.

ЗАВДАННЯ

ДЛЯ КВАЛІФІКАЦІЙНОЇ РОБОТИ СТУДЕНТУ

Фоміну Володимиру Олеговичу

1. Тема роботи: Чисельні характеристики побудови узагальнених FM-index для алфавітів різних розмірів керівник роботи Зважій Дмитро Володимирович

   затверджені наказом вищого навчального закладу від «__»_____20__року №____
2. Строк подання студентом роботи _____
3. План роботи _____

4. **Календарний план виконання роботи**:

№	Назва етапу	Термін виконання	Примітка
1.	Отримання теми курсової роботи	16.10.2023	
2.	Пошук тематичної літератури	17.11.2023	
3.	Ознайомлення з літературою	24.12.2023	
4.	Створення плану роботи	26.12.2023	
5.	Написання теоретичної частини роботи	25.03.2024	
6.	Розробка застосунку для дослідження	14.04.2024	
7.	Описання практичної частини роботи	16.04.2024	
8.	Проведення дослідження	18.04.2024	
9.	Аналіз дослідження та висновки	18.04.2024	
10.	Подання першої версії записки науковому керівнику	19.04.2024	
11.	Перегляд змісту роботи керівником	14.05.2024	
12.	Внесення змін до курсової роботи відповідно до зауважень наукового керівника	15.05.2024	

Студент Фомін В.О.

Керівник Зважій Д.В.

“____” _____

ЗМІСТ

АНОТАЦІЯ

Дана робота розкриває деталі алгоритму побудови *узагальненого FM-index*, який дозволяє ефективно працювати з колекцією текстових рядків. В рамках роботи також досліджено різні чинники, які впливають на чисельні характеристики побудови *узагальненого FM-index*, зокрема вплив на обсяг пам'яті та швидкість пошуку підрядків. Особливу увагу було приділено аналізу впливу розміру алфавіту. Дослідження та вимірювання проведені у браузері на основі власній імплементації *узагальненого FM-index* з використанням мови програмування JavaScript.

ВСТУП

Обробка та аналіз великих обсягів текстових даних є ключовим завданням у багатьох галузях інформаційних технологій, таких як біоінформатика, комп'ютерна лінгвістика, обробка природної мови, геноміка та інформаційний пошук. Через постійне збільшення кількості текстової інформації, яку потрібно обробити, постає потреба у швидших та більш ефективних алгоритмах для обробки та зберігання цих даних.

Одним з потужних інструментів для роботи з текстовою інформацією є *FM-index*. *FM-index* є важливою структурою даних, яка використовується для ефективного пошуку підрядків у великих текстових даних. Завдяки своїй здатності до одночасного стиснення даних та виконання швидких пошукових запитів, *FM-index* широко застосовується в біоінформатиці та інформаційному пошуку. Зазвичай *FM-index* використовується для обробки одного великого рядка тексту з алфавітом невеликого розміру, наприклад, ДНК.

Водночас у сфері розробки вебзастосунків дедалі більш поширеною стає робота з великими алфавітами та колекцією невеликих рядків, яка зберігається у текстовому файлі. Це призводить до необхідності використання *узагальненого FM-index*, який дозволяє працювати з багатьма рядками одночасно.

Отже, метою цієї роботи є ретельне дослідження та аналіз чисельних характеристик побудови *узагальненого FM-index* залежно від розміру алфавіту. Окрім цього, велика популярність вебзастосунків вимагає врахування особливостей середовища веббраузера, що має обмеження на розмір даних, з якими можна працювати, наприклад алфавіт. Таким чином, буде також досліджено оптимальний розмір алфавіту для ефективної роботи з *узагальненим FM-index* у веббраузері.

1 ТЕОРЕТИЧНИЙ МАТЕРІАЛ

Для кращого розуміння предметної області, дослідження та проблематики, детально розглянемо основні терміни, принципи та позначення, які використовуються для побудови *узагальнених FM-index*.

1.1. Текстові рядки та операції над ними

Рядок S — послідовність символів типу $\{c_0, c_1,..., c_n\}$. Символи рядка є елементами множини певного алфавіту $\Sigma = \{1, 2,..., \sigma\}$. Довжину рядка S будемо позначати як $|S|$. Будемо вважати рядок масивом і звертатись до його елементів (символів) як $S[i]$, де $0 \leq i < |S|$.

Підрядок рядка S — рядок вигляду $S[i, j]$, який є підмножиною рядка S починаючи з символу на позиції i та закінчуючи символом на позиції j, де $0 \leq i \leq j < |S|$.

Префікс рядка S — підрядок вигляду $S[0, i]$, де $0 \leq i < |S|$.

Суфікс рядка S — підрядок вигляду $S[i, |S|-1]$, де $0 \leq i < |S|$.

Конкатенація рядків S_1 *та* S_2 — створення одного рядка через послідовне об'єднання рядків S_1 та S_2. Будемо позначати як S_1S_2 або $S_1 \cdot S_2$.

Колекція рядків C — множина рядків $S_1, S_2, ..., S_n$ яку можна представити у вигляді рядка $S = S_1S_2...S_n$, де кожен рядок S_i має певну довжину l_i, а загальна довжина колекції дорівнює $|C| = |S| = l$.

Лексикографічне впорядкування — розташування рядків згідно з позицією їх символів у алфавіті. Наприклад, якщо S_1 та S_2 це рядки довжиною l_1 і l_2 відповідно, тоді $S_1 < S_2$ за виконання однієї з наступних умов:

- $S_1[0, i - 1] = S_2[0, i - 1]$ та $S_1[i] < S_2[i]$, де $0 < i \leq min(l_1, l_2)$.
- $l_1 < l_2$ та $S_1[0, l_1 - 1] = S_2[0, l_1 - 1]$.

1.2. FM-index

Індекс підрядка — структура даних для пошуку підрядка в тексті або наборі текстів за сублінійний час. Це означає, що пошук входжень зразка *S* у тексті *T* довжиною *n* або наборі текстів $D = \{T^1, T^2, ..., T^m\}$ загальною довжиною n відбуватиметься швидше за *O(n)* (перевірка кожного символу тексту).

FM-index — стиснутий індекс підрядка, який будується за допомогою *перетворення Берроуза-Вілера (англ. BWT)* та інших допоміжних структур даних, таких як *таблиця входжень* та *частковий суфіксний масив*. Авторами *FM-index* є Паоло Феррагіно та Джованні Манзіні, які вперше представили його у статті “Opportunistic Data Structures with Applications” 2000 року. Автори описують *FM-index* як опортуністичну структуру даних, оскільки він дозволяє одночасно стискати вхідний текст та виконувати швидкий пошук підрядка. Назва розшифровується як повнотекстовий індекс у хвилинному просторі (англ. **F**ull-text index in **M**inute space), хоча прізвища авторів починаються на ті ж самі літери (**F**erragina та **M**anzini). Дослідження показали, що *FM-index* є оптимальним вибором для задач, де необхідне швидке та ефективне виконання пошукових запитів у великих текстових даних. Порівняно з більш традиційними структурами даних, такі як суфіксні дерева та суфіксні масиви, *FM-index* у побудованому вигляді використовує значно менше пам'яті завдяки стисненню даних. Ця властивість робить *FM-index* дуже гарним рішенням для роботи з великими обсягами тексту, особливо де є критичні обмеження ресурсів пам'яті, наприклад веббраузер. Крім того, *FM-index* забезпечує високу швидкість виконання пошукових запитів, що робить його дуже практичним для розробки комерційних застосунків.

Разом з тим, *FM-index* має наступні недоліки, деякі з яких можна виправити залежно від способу реалізації:

- Для побудови необхідно провести попередній аналіз тексту, що може зайняти багато часу та ресурсів. Це є особливо критичним у випадку текстів, які динамічно змінюються.

- Може бути менш ефективним для деяких типів запитів, наприклад, пошук за виключенням.
- Чутливість до алфавітів великих розмірів.

Для зручності та зрозумілості варто позначати кінець рядка термінальним символом. Зазвичай, у ролі термінального символу використовують «$», але це не є єдиним варіантом. Головне, щоб термінальний символ більше ніде не зустрічався у тексті. Термінальний символ має найменшу лексикографічну оцінку в алфавіті.

Узагальнений FM-index — варіант *FM-index* для збереження та пошуку підрядка у колекції рядків. Основною перевагою є те, що час на пошук підрядка не залежить від кількості рядків в колекції, що робить *узагальнений FM-index* чудовою структурою для пошуку у словнику.

2 АЛГОРИТМ ПОБУДОВИ FM-INDEX

Побудова *FM-index* засновується на *перетворенні Берроуза-Вілера (BWT)* для вхідного тексту, пошуку *LF-відображення* та застосуванні допоміжних структур даних.

2.1 Перетворення Берроуза-Вілера

Перетворення Берроуза-Вілера (англ. Burrows-Wheeler transform, BWT) — метод перестановки символів стрічки *S* у результаті якого отримується стрічка *BWT(S)* з якої можливо отримати початкову послідовність символів.

Алгоритм отримання стрічки *BWT(S)* складається з наступних кроків:

1. Створити *n* ротацій рядка *S*, де *n* – довжина рядка *S*.
2. Створити матрицю *BWM* розташувавши кожну ротацію одна під одною.
3. Лексикографічно впорядкувати *BWM*.
4. Скласти новий рядок за допомогою конкатенації останніх символів кожної ротації починаючи згори.

Перетворення відповідає новому рядку утвореному в останньому кроці.

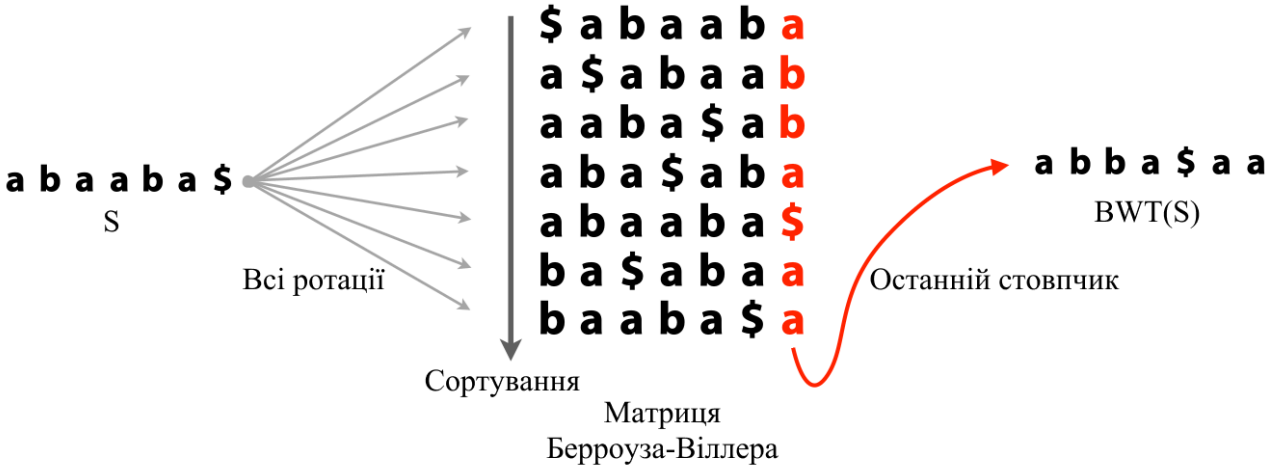


Рисунок 2.1 - Перетворення Берроуза-Вілера для рядка "abaaba$"

2.2 LF-відображення

LF-відображення (англ. – LF-mapping, last-to-first-mapping) — операція, яка використовується для оберненого перетворення *BWT* та пошуку в *FM-index*. Операція ґрунтується на тому, що у матриці перетворення Берроуза-Вілера (*BWM*) відносний порядок входжень символів *c* в першому стовпчику *F* зберігається в останньому стовпчику *L*.

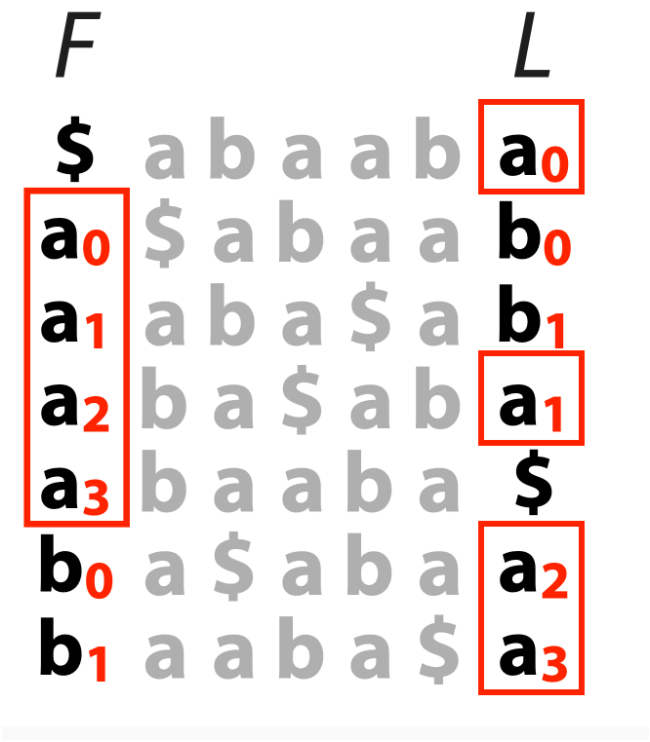


Рисунок 2.2 - Відносний порядок входжень символів у стовпчиках F та L

LF-відображення застосовується наступним чином:

1. Беремо символ зі стовпчика *F*. Спочатку, це завжди термінальний символ («$») який знаходиться нагорі. Додаємо цей символ в кінець рядка.
2. Беремо символ зі стовпчика *L*, який знаходиться на тій самій позиції що й символ, який ми взяли зі стовпчика *F* у попередньому кроці. Якщо це термінальний символ, припиняємо процес перетворення.
3. Знаходимо у стовпчику *F* символ, який ми взяли у попередньому кроці. Додаємо цей символ в кінець рядка. Далі продовжуємо виконання переходом до кроку 2 допоки ми не зустрінемо термінальний символ.

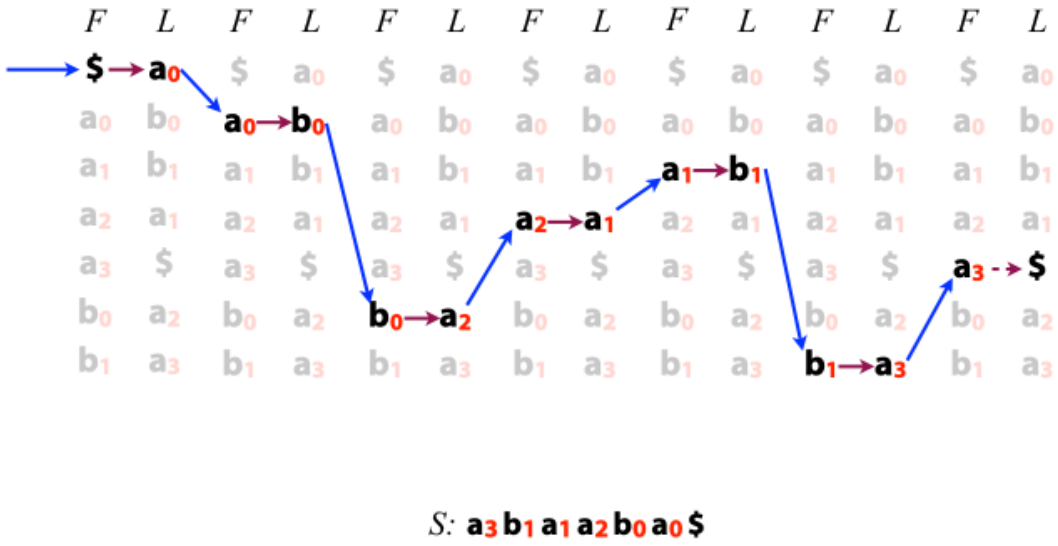


Рисунок 2.3 - Обернене перетворення рядка "abaaba$"

2.3 Обробка запитів

Для обробки пошукових запитів використовується *зворотний пошук (англ. – reverse search)*. Якщо ми шукаємо рядок $S = c_0c_1...c_n$, то пошук розпочнеться з останнього символу c_n:

1. Шукаємо всі входження c_n у стовпчику F та залишаємо лише ті, де на відповідній позиції у стовпчику L стоїть символ c_{n-1}.
2. Повертаємося до стовпчика F і шукаємо входження символу $cn-1$ і так само залишаємо ті, де на відповідній позиції у стовпчику L стоїть символ c_{n-2}.
3. Продовжуємо процес пошуку для решти символів рядка $\{c_{n-2},..., c_0\}$. Процес зупиняється при досягненні символу c_0 або коли відсутні рядки, які задовольняють умову. Якщо шуканий підрядок зустрічається декілька разів у тексті, в результаті пошуку буде вказано всі позиції, де знайдено підрядок.

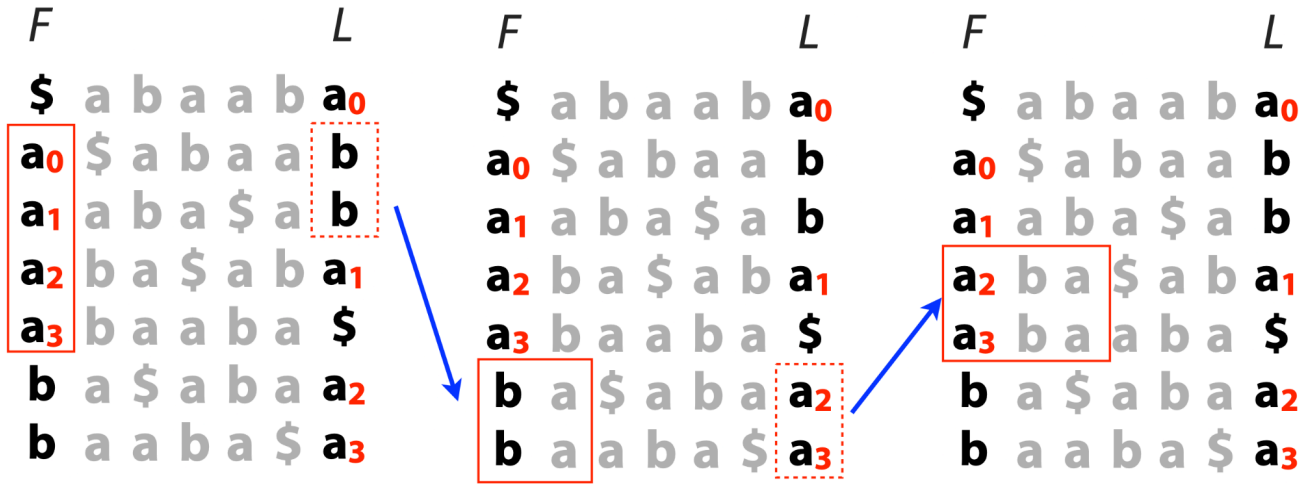


Рисунок 2.4 - Зворотний пошук підрядка "aba"

Постає дві проблеми:

1) Для застосування *LF-відображення* необхідно мати можливість визначати ранги символів у стовпчику *L*. Очевидним рішенням є почергове сканування символів, але воно не підходить, оскільки в найгіршому випадку його виконання буде вимагати *O(n)* часу, де *n* – довжина стовпчика *L*, що не є ефективним.
2) Для пошуку підрядка потрібно мати можливість визначити позиції його входжень. Це можна вирішити за допомогою зберігання *суфіксного масиву*, але тоді втрачається стиснення даних, що є основною перевагою *FM-index*.

Для розв'язання цих проблем, використаємо дві допоміжні структури даних.

2.4 Допоміжні структури даних для побудови індексу

Таблиця входження (англ. Occurrence Table) — допоміжна структура даних, де зберігається кількість попередніх входжень кожного символу алфавіту, включно із термінальним символом, в останньому стовпчику *L* матриці *перетворення Берроуза-Вілера*. Використовується для пришвидшення пошуку підрядка та визначення рангів символів у *FM-index*. Ранг символу *c* можна визначити зменшивши на 1 попередні входження цього символу на поточній позиції.

L	a	b
a	1	0
b	1	1
b	1	2
a	2	2
$	2	2
a	3	2
a	4	2

Рисунок 2.5 - Таблиця входження для рядка "abaaba$"

Тоді пошук символів у стовпчику *L* та визначення рангу конкретного символу не буде займати багато часу, проте буде займати багато пам’яті — *O(m*|Σ|)*, де *m* довжина стовпчика *L*.

Цей процес можна оптимізувати зберігаючи лише кожне *k*-те входження — назвемо такі входження контрольними точками. Тоді, щоб дізнатись ранг символу, достатньо взяти найближчу верхню контрольну точку до заданого символу та додати кількість входжень символу у проміжку між цією точкою та заданим символом. Чим більше значення *k*, тим більше часу потрібно на пошук. Тому у кожному конкретному випадку потрібно шукати прийнятний баланс між довжиною проміжків та економією пам’яті.

Для визначення позицій входжень шуканого підрядка можна застосувати схожий підхід, але зберігати часткові дані вже про *суфіксний масив*. *Суфіксний масив (англ. Suffix Array)* — масив, де зберігається інформація про всі суфікси певного рядка *S* відсортовані у лексикографічному порядку. Для кожного суфіксу буде зберігатись його відступ від початку рядка *S*. Наприклад, у *суфіксному масиві* рядка *“banana$”* для суфікса *“nana$”* буде записано значення 2, оскільки для

отримання цього суфікса потрібно посунутись на два символи праворуч від початку рядка.

Використання *суфіксного масиву* для визначення позицій входжень підрядка працює через однаковий порядок суфіксів у матриці перетворення *BWM* та суфіксному масиві.

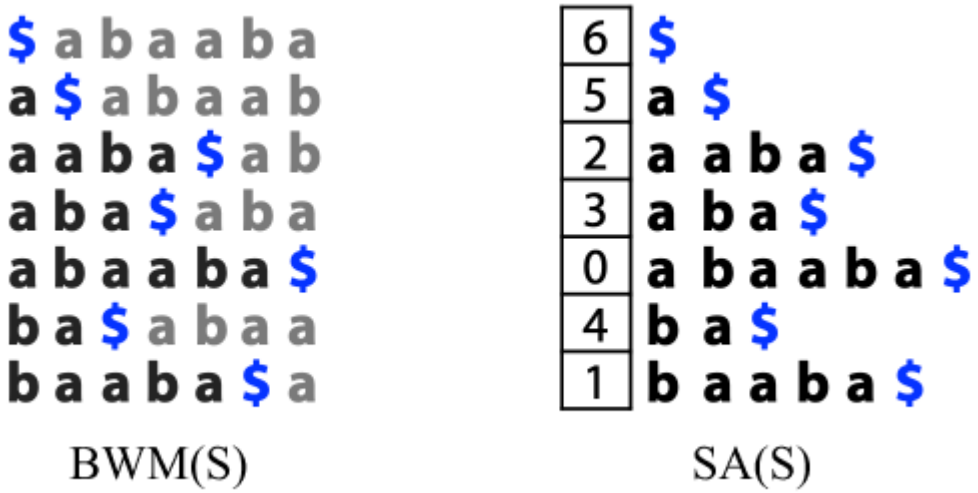


Рисунок 2.6 - Схожість BWM та суфіксного масиву

Будемо позначати рядок, що утворився внаслідок перетворення рядка *S* як *BWT(S)*, а суфіксний масив рядка *S* будемо позначати як *SA(S)*. Тоді:

$$\text{BWT(S)[i]} = \begin{cases} \text{S}[SA(\text{S})[i] - 1], SA(\text{S})[i] > 0 \\ \$, SA(\text{S})[i] = 0 \end{cases} \quad (2.1)$$

Для того, щоб знайти значення для пропущеного символу зі стовпчика *F*, робимо наступне:

1. Беремо відповідний символ зі стовпчика *L* та шукаємо його *LF-відображення* у *F*. Повторюємо цей процес допоки не буде знайдено збережений індекс.
2. До знайденого індексу додаємо кількість витрачених кроків пошуку, бо шукаючи *LF-відображення* ми фактично зміщали суфікс на 1 символ вліво, ближче до початку рядка.

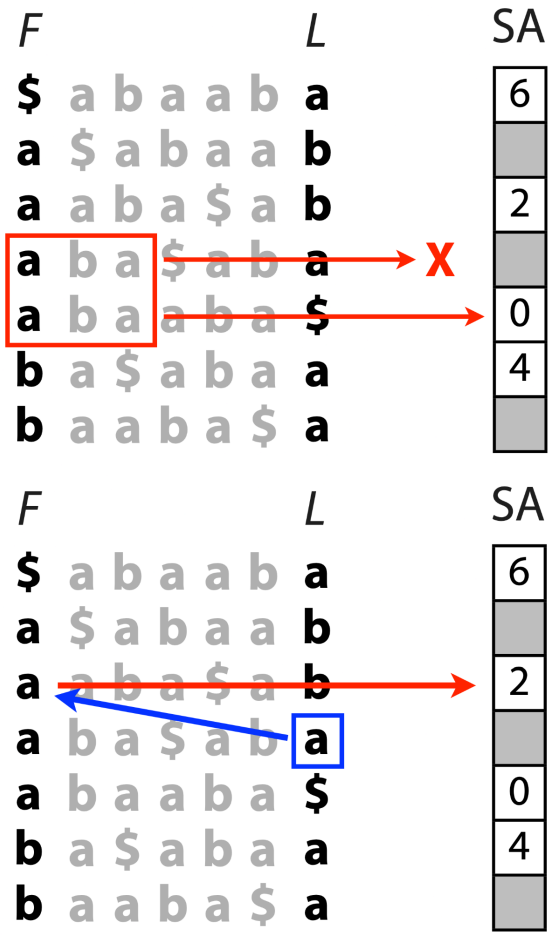


Рисунок 2.7 - Визначення позиції через частково збережений суфіксний масив

3 ПОБУДОВА УЗАГАЛЬНЕНОГО FM-INDEX

Узагальнений FM-index передбачає роботу з колекцією рядків, що вимагає певні модифікації алгоритму побудови звичайного *FM-index*.

Для використання *перетворення Берроуза-Вілера* для колекції рядків необхідно спочатку об'єднати всі рядки в один за допомогою конкатенації. Кожен рядок має бути відокремлений спеціальним термінальним символом (наприклад “$”). Після застосування *перетворення Берроуза-Вілера* до цього з'єднаного рядка, ми отримаємо новий рядок, який можна обробляти за допомогою *LF-відображення*.

Крім того, виникає нова обов’язкова умова для *Таблиці входжень*. Якщо раніше ми могли не зберігати інформацію про термінальний символ, то для *узагальненого FM-index* це стає обов'язковою складовою, щоб можна було дати явну вказівку на те, що шуканий підрядок має бути суфіксом.

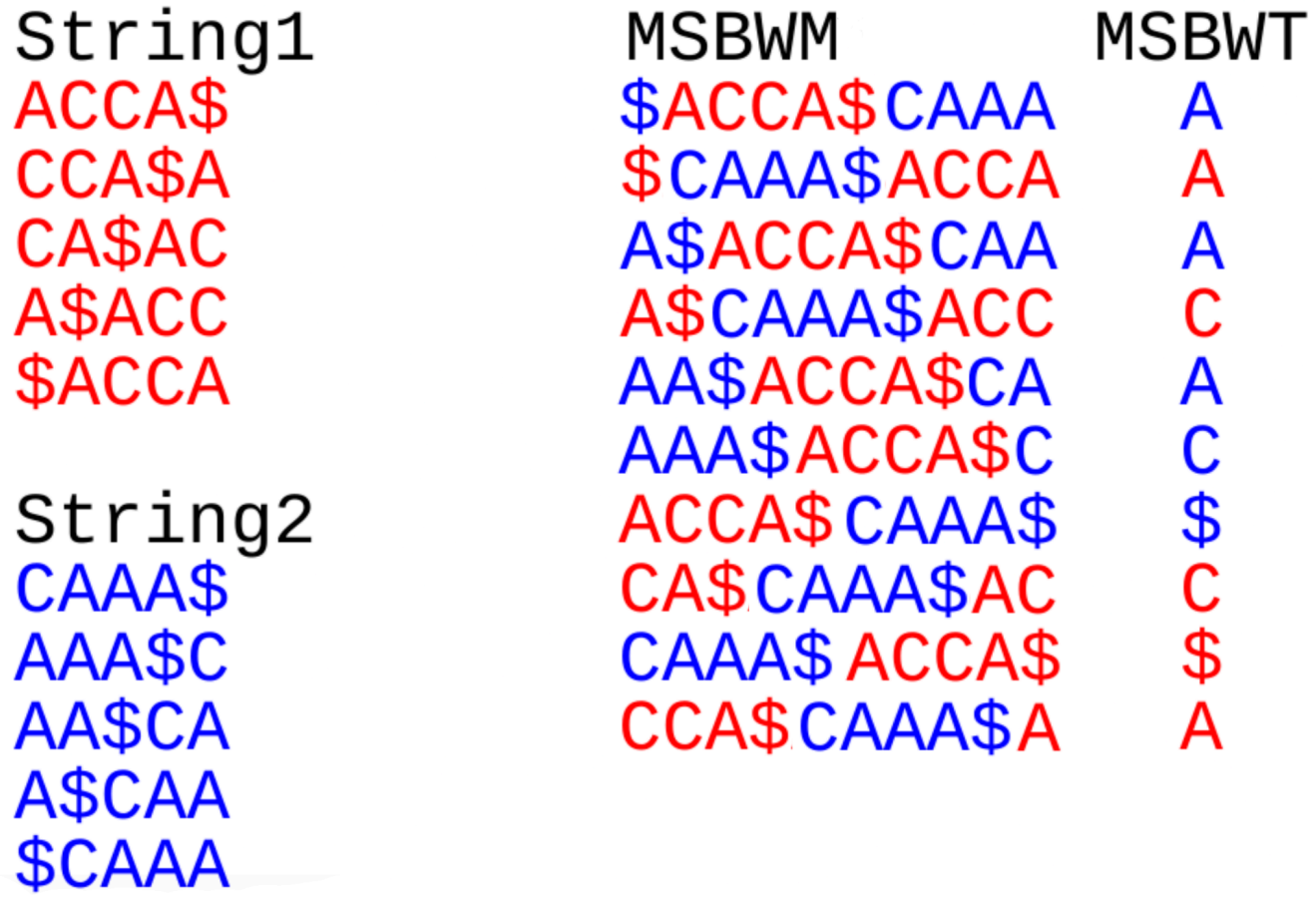


Рисунок 3.1 - Перетворення Берроуза-Вілера для колекції рядків

4 ЧИННИКИ ЯКІ ВПЛИВАЮТЬ НА ЧИСЕЛЬНІ ХАРАКТЕРИСТИКИ ПОБУДОВИ

Стиснення досягається за допомогою зберігання неповних вхідних даних. Для реалізації *FM-index* нам необхідно зберігати лише значення стовпчиків F та L, *таблиці входжень* і *часткового суфіксного масиву*. Стовпчик F можна зберігати у вигляді масиву чисел F_c так, що i-му символу алфавіту відповідатиме кількість символів у рядку лексикографічно менших за нього. Таке представлення можна використовувати через те, що символи у стовпчику F лексикографічно впорядковані. Наприклад, для алфавіту $\{\$, a, b, n, s\}$, $F_c = [0, 2, 8, 9, 13]$ та $L =$ *“asnnbnnaaaaa”*.

Можемо зробити наступну оцінку чисельних характеристик побудови *FM-index*:

- Масив розміром $|\Sigma|$ для збереження F.
- Масив розміром m для збереження L, де m — довжина L та дорівнює довжині вхідного тексту.
- Масив розміром $m \cdot |\Sigma| \cdot 1/k_1$ для збереження *таблиці входження*, де k_1 — довжина проміжку між контрольними точками;
- Масив розміром $m \cdot 1/k_2$ для збереження значень SA, де k_2 — довжина проміжку між контрольними точками.
- Пошук підрядка має складність $O(p)+O(1)$, де p — довжина шуканого рядка.

Таким чином, можна виділити наступні чинники, які впливають на чисельні характеристики побудови *узагальненого FM-index*:

- Розмір алфавіту.
- Обсяг вхідного тексту.
- Довжина шуканого підрядка.

5 РЕЗУЛЬТАТИ

Вимірювання чисельних характеристик побудови *узагальненого FM-index* проводилось на пристрої Apple MacBook Air 13” Late 2020 з процесором Apple M1 та оперативною пам'яттю 16 Гб у середовищі браузера Brave. При вимірюванні враховувались наступні параметри:

- Розмір вхідних текстових даних S_o.
- Розмір алфавіту Σ, який використовується у вхідних текстових даних.
- Час T для побудови *узагальненого FM-index*.
- Розмір S для збереження *узагальненого FM-index*.

№	Назва	S_o	$\|\Sigma\|$	T	S	Опис
1	Eng	88 Мб	26	122,197 с	2719,84 Мб	Вісім мільйонів рядків довжиною від 5 до 15 символів англійського алфавіту
2	Eng2	88 Мб	8	117,27 с	1331,69 Мб	Вісім мільйонів рядків довжиною від 5 до 15 символів скороченого англійського алфавіту
3	Eng3	88 Мб	17	111,552 с	2001,22 Мб	Вісім мільйонів рядків довжиною від 5 до 15 символів скороченого англійського алфавіту
4	Eng4	11 Мб	17	11,146 с	264,55 Мб	Один мільйон рядків довжиною від 5 до 15 символів скороченого

						англійського алфавіту
5	Ukr	839 Кб	34	269,619 с	17,82 Мб	Сорок тисяч рядків довжиною від 5 до 15 символів українського алфавіту
6	Kor	928 Кб	19	106,315 с	10,65 Мб	Тридцять тисяч рядків довжиною від 5 до 15 символів корейського алфавіту

Таким чином, можна побачити, що *узагальнений FM-index* у побудованому вигляді вимагає в 12-30 разів більше пам’яті ніж вхідні дані. Також помітно простежується залежність між розміром алфавіту та розміром побудованої структури.

Проте не можна простежити, щоб розмір алфавіту впливав на час побудови самої структури. Натомість помітно присутність залежності часу побудови *узагальненого FM-index* від типу символів, які присутні в алфавіті. Таким чином використання українських та корейських символів в алфавіті значно знизило швидкість побудови структури, хоча не мало впливу на співвідношення між розміром вхідних даних та розміром кінцевої структури. Сповільнення у побудові структури при використанні символів українських чи корейських символів в першу чергу спостерігається під час порівняння рядків між собою для сортування проміжних структур даних.

Побудова для англійського алфавіту виконується за відносно невелику кількість часу. Для найбільшого файлу розміром 88 мегабайтів, *узагальнений FM-index* був побудований за 122 секунди, а для найменшого файлу розміром 11 мегабайтів, час побудови зайняв 11 секунд. Для файлів більших розмірів продуктивність програми перестає бути прийнятною та в деяких випадках навіть

можливою. Для текстових даних, де використовуються символи, які не належать до латинської абетки, прийнятними є файли розміром до 1 Мб.

ВИСНОВКИ

В ході роботи було досліджено та проаналізовано алгоритм побудови стандартного *FM-index* та виявлено, що ця структура даних є виразним прикладом високоякісного та ефективного інструменту для роботи з текстами. Спочатку, було проведено детальний огляд кожного кроку цього алгоритму та оцінено його ефективність. Побудова *FM-index* складається із застосування *перетворення Берроуза-Вілера* до вхідного тексту та використанні допоміжних структур даних. Також було запропоновано деякі оптимізації для економії ресурсів при побудові. Основними способами оптимізації є здійснення *перетворення Берроуза-Вілера* за допомогою *суфіксного масиву*, який є однією із допоміжних структур даних потрібних для роботи з *FM-index*. Окрім цього, оптимізації використання ресурсів було досягнуто за допомогою часткового зберігання цього *суфіксного масиву* та *таблиці входжень*, шляхом збереження кожного *k*-го входження, що дозволяє у кожному окремому випадку реалізації знайти компроміс між швидкістю роботи та використанням пам’яті.

Також було проведено огляд необхідних модифікацій алгоритму для побудови узагальненої версії цієї структури. Варто зазначити, що побудова узагальненої версії не вимагає значних змін у початковому алгоритмі.

Після ретельного огляду алгоритму побудови, застосування оптимізацій, та модифікації алгоритму для роботи з колекцією рядків, було проведено теоретичний аналіз чинників, які впливають на чисельні характеристики побудови *узагальненого FM-index* та роботи з ним. На основі цього аналізу, було зроблено висновок, що основними чинниками є розмір алфавіту та вхідних даних. Розмір шуканого підрядка впливає на швидкість роботи зі структурою.

Нарешті, було імплементовано запропонований алгоритм побудови *узагальненого FM-index* на мові JavaScript та проведено вимірювання чисельних характеристик його побудови на пристрої Apple MacBook Air 13” Late 2020 з процесором Apple M1 та оперативною пам'яттю 16 Гб. Це вимірювання

продемонструвало практичний вплив чинників, які були виявлені в ході теоретичного аналізу. Так більший розмір алфавіту збільшує розмір кінцевої структури, а більший розмір вхідних даних збільшує як розмір структури, так і час її побудови. Розмір побудованого *узагальненого FM-index* зайняв у 12-30 разів більше пам’яті порівняно з розміром вхідних даних. Час побудови *узагальненого FM-index* зайняв небагато часу для словників з латинськими символами. Для колекції восьми мільйонів рядків довжиною від п’яти до п’ятнадцяти символів, побудова зайняла 122 секунди, а для колекції одного мільйона рядків схожої довжини, *узагальнений FM-index* було побудовано протягом 11 секунд. Для словників, де використовуються символи нелатинського алфавіту, швидкість побудови *узагальненого FM-index* була значно більш повільною. Так для словника, де міститься сорок тисяч рядків довжиною від 5 до 15 символів українського алфавіту, побудова *узагальненого FM-index* зайняла 270 секунд. Сповільнення спостерігається під час порівняння між собою рядків, які складаються з нелатинських символів, особливо при побудові суфіксного масиву.

Оскільки дослідження проводилось у середовищі браузера Brave, було також визначено максимальний розмір вхідних даних, які можна обробити у цьому середовищі. Він становить приблизно 88 мегабайтів.

СПИСОК ЛІТЕРАТУРИ

1. Pang Ko, Srinivas Aluru. Space efficient linear time construction of suffix arrays. 2004. URL: https://doi.org/10.1016/j.jda.2004.08.002.
2. Burrows-Wheeler Transform and FM Index Ben Langmead https://www.cs.jhu.edu/~langmea/resources/lecture_notes/bwt_and_fm_index.pdf.
3. Burrows-Wheeler Transform & FM Index Ben Langmead https://www.cs.jhu.edu/~langmea/resources/lecture_notes/10_bwt_and_fm_index_v2.pdf.
4. FM-index for Dummies | SpringerLink. https://link.springer.com/chapter/10.1007/978-3-319-58274-0_16.
5. An optimized FM-index library for nucleotide and amino acid search .... https://almob.biomedcentral.com/articles/10.1186/s13015-021-00204-6.
6. Opportunistic Data Structures with Applications - uniupo.it. https://people.unipmn.it/manzini/papers/focs00draft.pdf.
7. Opportunistic data structures with applications. https://users.cs.utah.edu/~pandey/courses/cs6968/spring23/papers/fmindex.pdf.
8. Opportunistic data structures with applications - uniupo.it. https://people.unipmn.it/manzini/papers/focs00.html.
9. Comp 555 - BioAlgorithms - Spring 2020 https://www.csbio.unc.edu/mcmillan/Comp555S20/Lecture11.pdf.
10. Comp 555 - BioAlgorithms - Spring 2020 https://www.csbio.unc.edu/mcmillan/Comp555S20/Lecture12.pdf.
11. Multi-String BWTs https://www.csbio.unc.edu/mcmillan/Comp555S18/Lecture14.pdf.
12. String Algorithms and Data Structures CS 199-225 Brad Solomon March 28, 2022 FM Index

https://courses.engr.illinois.edu/cs225/fa2022/assets/honors/slides/cs199sp22-9-FMI-slides.pdf.

13. A SIMPLE ALPHABET-INDEPENDENT FM-INDEX ∗ https://www.researchgate.net/publication/220180334_A_Simple_Alphabet-independent_Fm-index.
14. «Методи оптимізації використання пам’яті для узагальнених суфіксних масивів» — Барабуха Марія Максимівна https://ekmair.ukma.edu.ua/handle/123456789/28681.

ДОДАТКИ

Додаток А

Код для побудови суфіксного масиву:

```
function suffixArray(s) {
    const satups = [...Array(s.length).keys()].sort((a, b) => s.slice(a).localeCompare(s.slice(b)));
    return satups;
}
```

Додаток Б

Код для перетворення Берроуза-Вілера із застосуванням суфіксного масиву:

```
function bwtFromSa(t, sa) {
    const bw = [];
    if (!sa) {
        sa = suffixArray(t);
    }
    for (const si of sa) {
        if (si === 0) {
            bw.push('$');
        } else {
            bw.push(t[si - 1]);
        }
    }
    return bw.join('');
}
```

Додаток В

Код для створення контрольний точок для узагальненого FM-index:

```
class FmCheckpoints {
    constructor(bw, cpIval = 4) {
        this.cps = {};
        this.cpIval = cpIval;
        const tally = {};

        for (const c of bw) {
            if (!(c in tally)) {
                tally[c] = 0;
                this.cps[c] = [];
            }
        }

        for (let i = 0; i < bw.length; i++) {
            const c = bw[i];
            tally[c] += 1;
            if (i % cpIval === 0) {
                for (const c in tally) {
                    this.cps[c].push(tally[c]);
                }
            }
        }
    }

    rank(bw, c, row) {
        if (row < 0 || !(c in this.cps)) {
            return 0;
        }

        let i = row, nocc = 0;

        while (i % this.cpIval !== 0) {
            if (bw[i] === c) {
                nocc += 1;
            }
            i -= 1;
        }

        return this.cps[c][Math.floor(i / this.cpIval)] + nocc;
    }
}
```

Додаток Г

Код для побудови узагальненого FM-index:

```
class FmIndex {

    static downsampleSuffixArray(sa, n = 4) {
        const ssa = {};
        for (let i = 0; i < sa.length; i++) {
            if (sa[i] % n === 0) {
                ssa[i] = sa[i];
            }
        }
        return ssa;
    }

    constructor(t, cpIval = 4, ssaIval = 4) {
        const sa = suffixArray(t);
        this.bwt = bwtFromSa(t, sa);
        this.ssa = FmIndex.downsampleSuffixArray(sa, ssaIval);
        this.slen = this.bwt.length;
        this.cps = new FmCheckpoints(this.bwt, cpIval);
        const tots = {};
        for (const c of this.bwt) {
            tots[c] = (tots[c] || 0) + 1;
        }
        this.first = {};
        let totc = 0;
        for (const [c, count] of Object.entries(tots).sort()) {
            this.first[c] = totc;
            totc += count;
        }
    }

    count(c) {
        if (!(c in this.first)) {
            let cc;
            for (cc of Object.keys(this.first).sort()) {
                if (c < cc) return this.first[cc];
            }
            return this.first[cc];
        } else {
            return this.first[c] || 0;
        }
    }

    range(p) {
        let l = 0, r = this.slen - 1;
        for (let i = p.length - 1; i >= 0; i--) {
            const offset = this.count(p[i]);
            l = offset + this.cps.rank(this.bwt, p[i], l - 1);
            r = offset + this.cps.rank(this.bwt, p[i], r) - 1;
            if (r < l) break;
        }
        return [l, r + 1];
    }

    resolve(row) {
        const stepLeft = (row) => {
            const c = this.bwt[row];
            return this.cps.rank(this.bwt, c, row - 1) + (this.count(c) || 0);
        };
        let nsteps = 0;
        while (!(row in this.ssa)) {
            row = stepLeft(row);
            nsteps++;
        }
        return this.ssa[row] + nsteps;
    }

    hasSubstring(p) {
        const [l, r] = this.range(p);
        return r > l;
    }

    occurrences(p) {
        const [l, r] = this.range(p);
        const occurences = [];
        for (let x = l; x < r; x++) {
            occurences.push(this.resolve(x));
        }
        return occurences;
    }
}
```

Додаток Ґ

Код для вимірювання часу побудови узагальненого FM-index:

```
async function readText(event) {
    const file = event.target.files.item(0);
    if(!file)
        return;
    const text = await file.text();
    const startTime = performance.now();

    const fm = new FmIndex(text);

    const endTime = performance.now();
    output.innerText = `FMIndex built in ${endTime - startTime} milliseconds`;
}
```

Додаток Д

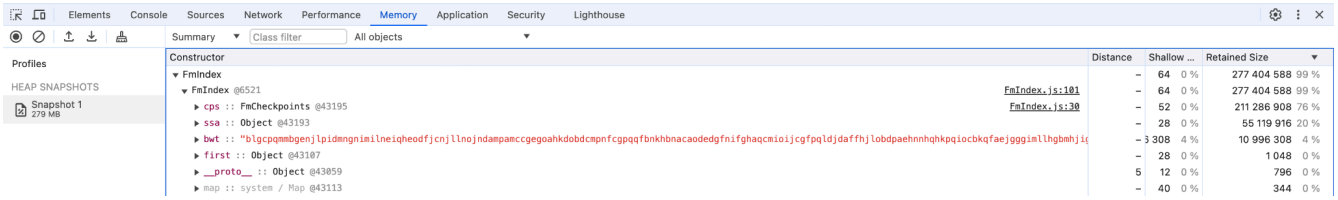
Код для створення словників:

```
const fs = require('fs');

function generateRandomString(minLength, maxLength, alphabet) {
    const characters = alphabet;
    const length = Math.floor(Math.random() * (maxLength - minLength + 1)) + minLength;
    let result = '';
    for (let i = 0; i < length; i++) {
        result += characters.charAt(Math.floor(Math.random() * characters.length));
    }
    return result;
}

function generateFile(filename, nStrings, alphabet) {
    let randString = '';
    for(let i = 0; i < nStrings; i++)
        randString += generateRandomString(5, 15, alphabet) + '$';

    fs.writeFile(filename, randString, err => {
        if (err) {
            console.error(err);
        } else {
            console.log("Success!");
        }
    });
}
```

Додаток Е

Вимірювання чисельних характеристик побудови узагальненого FM-index у браузері Brave:

Choose file eng4.txt
FMIndex built in 11146 milliseconds



Choose file eng4.txt
FMIndex built in 11146 milliseconds