Чисельні характеристики побудови узагальнених FM-index для алфавітів різних розмірів

Підготував студент ІПЗ-4
Фомін Володимир
Науковий керівник: Зважій Д.В.

Мета роботи

- Розробка ефективного алгоритму побудови узагальненого FM-index.
- Ретельне дослідження та аналіз чисельних характеристик побудови узагальненого FM-index залежно від розміру алфавіту.
- Визначення оптимального розміру алфавіту для ефективної роботи з узагальненим FM-index у веббраузері.

Хід виконання

01 Вивчення класичного рішення

02 Оптимізація алгоритму

03 Модифікація під узагальнену структуру даних

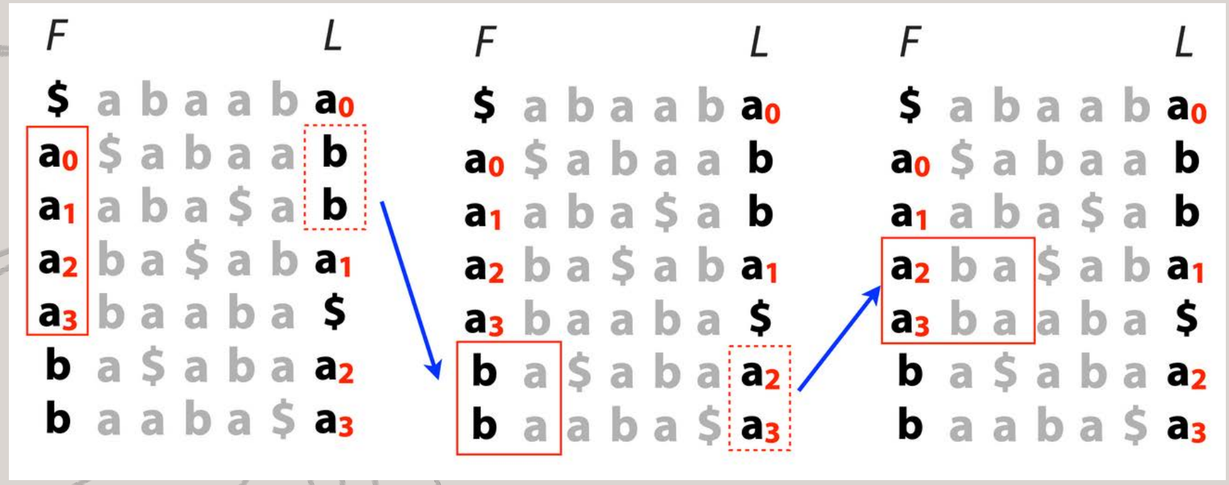
04 Аналіз чинників впливу на чисельні хар-ки

05 Визначення оптимального розміру даних

FM-index

Структура даних, яка дозволяє одночасно стискати вхідний текст та виконувати швидкий пошук підрядка. Вперше представлено Паоло Феррагіно та Джованні Манзіні у 2000 році.

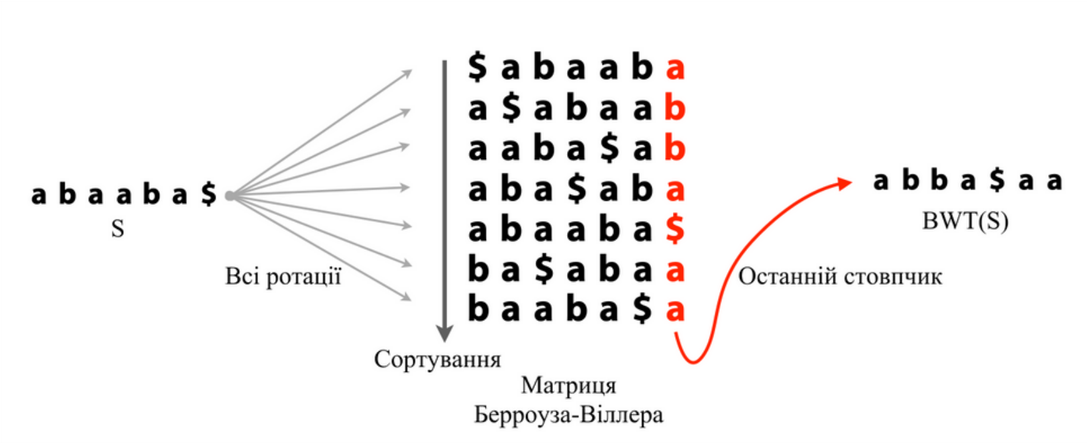
Широко використовується у біоінформатиці та інформаційному пошуку. Є дуже ефективним для обробки одного великого рядка з алфавітом невеликого розміру (ДНК).



Для побудови необхідно провести попередній аналіз тексту, що може вимагати багато ресурсів. Це є особливо критичним для тексту, який динамічно змінюються.

Менш ефективний для деяких типів запитів (пошук за виключенням).

Перетворення Берроуза-Вілера



Перет ворення Берроуза-Вілера (Burrows-Wheeler transform, BWT) — метод перестановки символів рядка у результаті якого отримується стрічка з якої можливо отримати початкову послідовність символів.

Алгоритм перетворення:

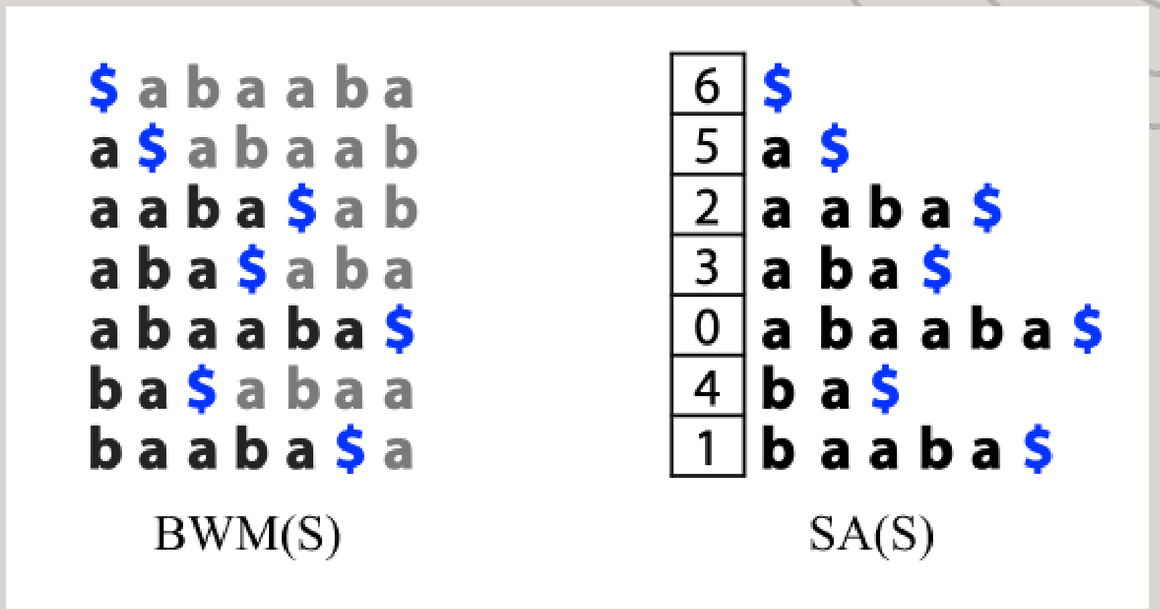
1. Створити n ротацій рядка (n – довжина рядка).
2. Створити матрицю BWM розташувавши кожну ротацію одна під одною.
3. Лексикографічно впорядкувати матрицю.
4. Скласти новий рядок за допомогою конкатенації останніх символів кожної ротації починаючи згори.

Перетворення відповідає новому рядку утвореному в останньому кроці.

Суфіксні масиви

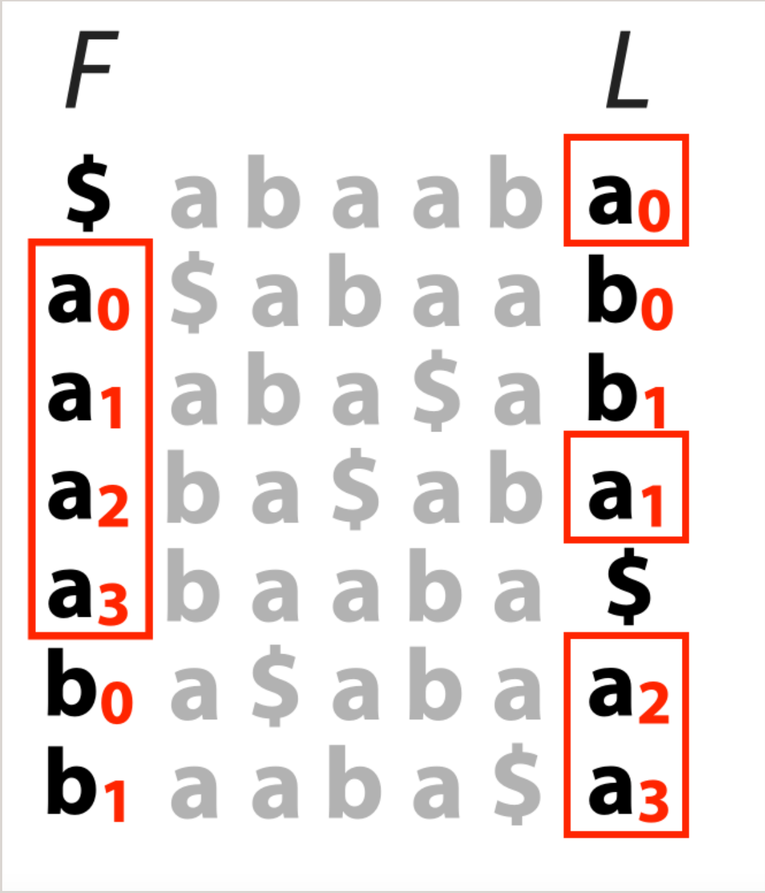
Суфіксний масив (Suffix array) — масив, де зберігається інформація про всі суфікси певного рядка відсортовані у лексикографічному порядку.

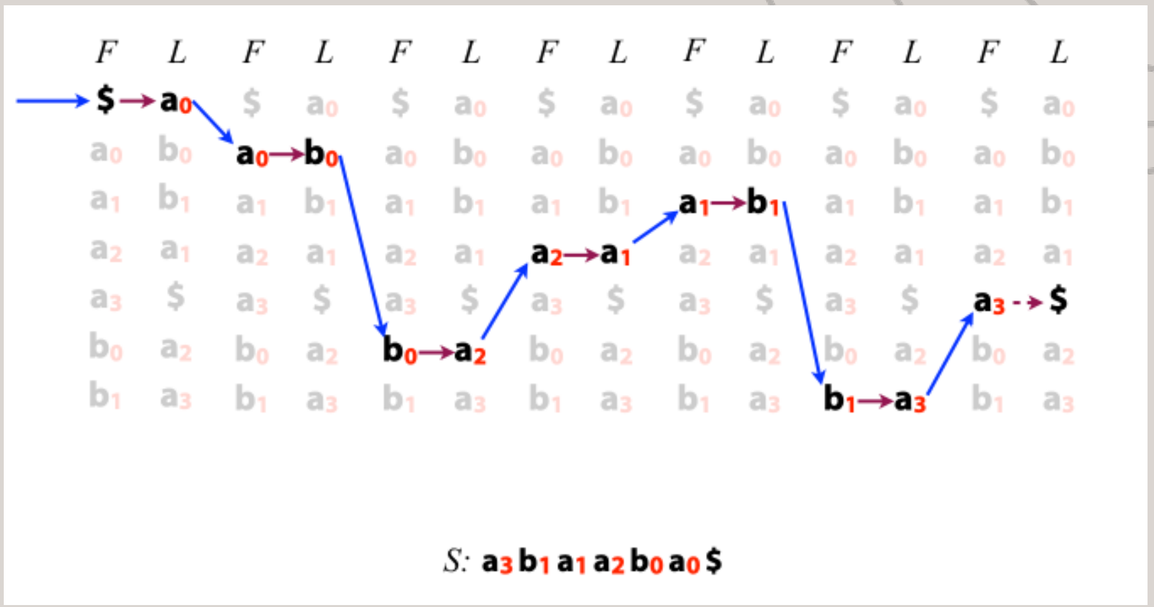
Суфіксний масив можна використати для здійснення перетворення Берроуза-Вілера завдяки їх схожим властивостям.



LF-Відображення

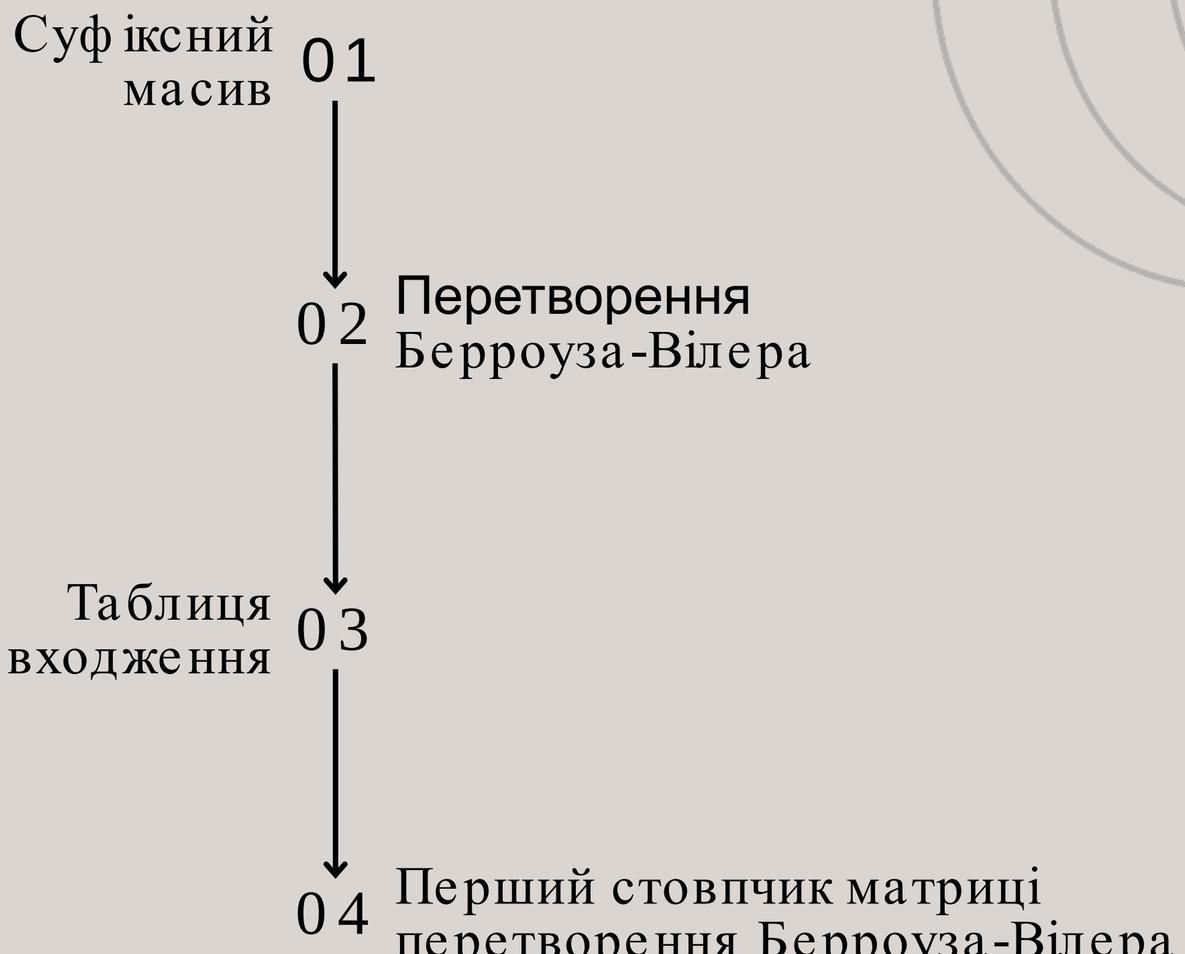
LF-Відображення (LF-Mapping) — операція, яка використовується для оберненого перетворення BWT та пошуку в FM-index.





Ґрунтується на тому, що у матриці перетворення Берроуза-Вілера (BWM) зберігається відносний порядок входжень символів в першій та останній колонках.

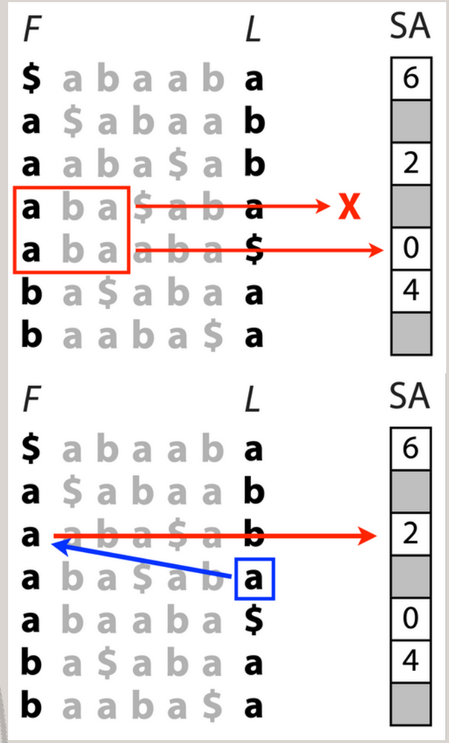
Алгоритм побудови



Оптимізації

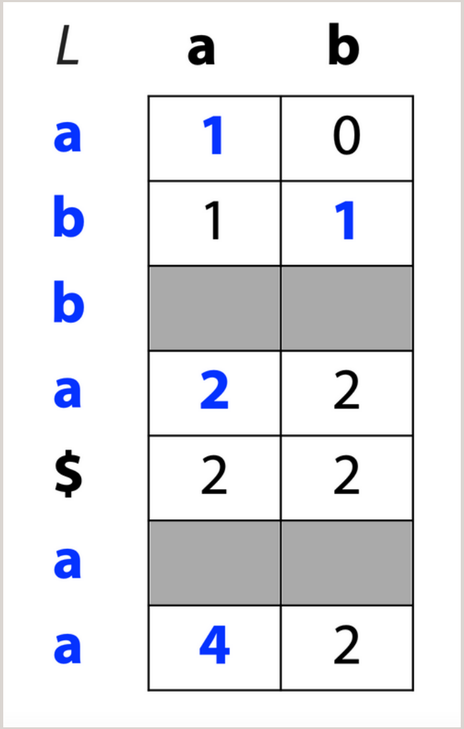
01

Суфіксний масив для перетворення Берроуза-Вілера



02

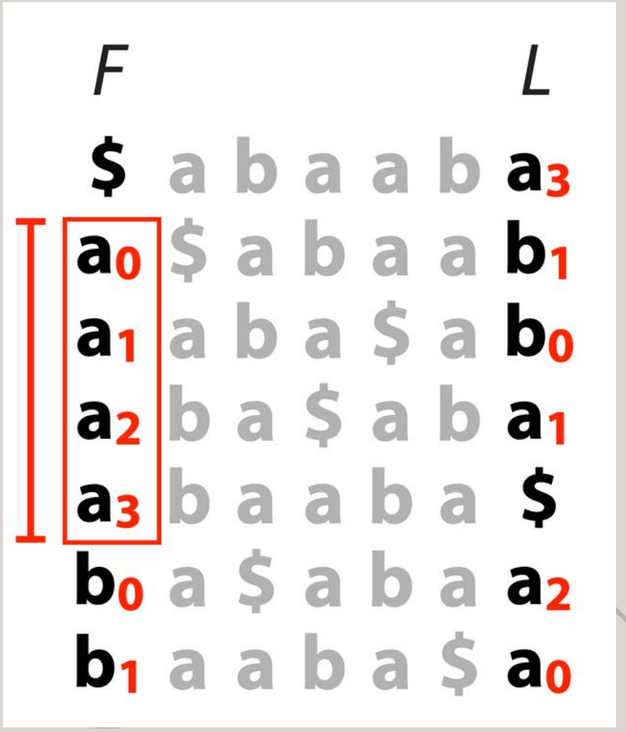
Часткове збереження суфіксного масиву та таблиці входження



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

03

Компактне збереження першого стовпчика *матриці BWM*



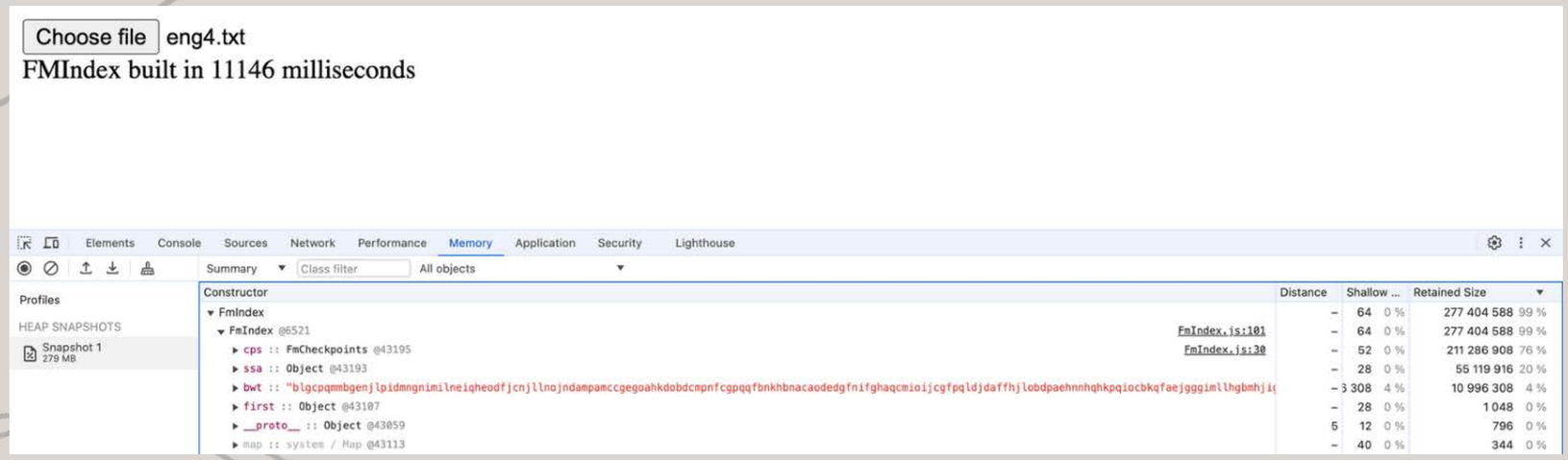
Узагальнений FM-index

Узагальнений FM-index передбачає роботу з колекцією рядків, що вимагає певні модифікації:

- Для перетворення Берроуза-Вілера, необхідно конкатенувати всі рядки колекції.
- У таблиці входження необхідно зберігати інформацію про термінальний символ.

String1	MSBWM	MSBWT
ACCA$	$ACCA$CAAA	A
CCA$A	$CAAA$ACCA	A
CA$AC	A$ACCA$CAA	A
A$ACC	A$CAAA$ACC	C
$ACCA	AA$ACCA$CA	A
	AAA$ACCA$C	C
String2	ACCA$CAAA$	$
CAAA$	CA$CAAA$AC	C
AAA$C	CAAA$ACCA$	$
AA$CA	CCA$CAAA$A	A
A$CAA		
$CAAA		

Реалізація та вимірювання




```
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;
    }
}
```

Результати

№	Назва	So	|Σ|	T	S	Опис
1	Eng	88 Мб	26	122,197 с	2719,84 Мб	8 млн рядків довжиною 5-15 символів англійського алфавіту
2	Eng2	88 Мб	8	117,27 с	1331,69 Мб	8 млн рядків довжиною 5-15 символів скороченого англійського алфавіту
3	Eng3	88 Мб	17	111,552 с	2001,22 Мб	8 млн рядків довжиною 5-15 символів скороченого англійського алфавіту
4	Eng4	11 Мб	17	11,146 с	264,55 Мб	1 млн рядків довжиною 5-15 символів скороченого англійського алфавіту
5	Ukr	839 Кб	34	269,619 с	17,82 Мб	40 тис. рядків довжиною 5-15 символів українського алфавіту
6	Kor	928 Кб	19	106,315 с	10,65 Мб	30 тис. рядків довжиною 5-15 символів корейського алфавіту

Висновки

01

Дослідження

Вивчення особливостей структури даних, класичного способу побудови та можливих варіантів оптимізації.

02

Реалізація

Написання алгоритму побудови на JavaScript, модифікація під роботу з колекцією рядків, подальша оптимізація.

03

Вимірювання

Тестування роботи на різних даних, визначення оптимальних розмірів даних, встановлення максимальної межі розміру.