Національний технічний університет України «Київський Політехнічний Інститут імені Ігоря Сікорського»

Комп’ютерний практикум №1
Алгоритми факторизації

Виконав:
студент групи ФІ-94
Маринін Іван Павло Ігорович

Перевірив:
Якимчук Олексій Петрович

Київ – 2022

Зміст

1. Мета комп’ютерного практикуму.

Практичне ознайомлення з різними методами факторизації чисел, реалізація цих методів і їх порівняння. Виділення переваг, недоліків та особливостей застосування алгоритмів факторизації. Застосування комбінації алгоритмів факторизації для пошуку канонічного розкладу заданого числа.

2. Постановка задачі та варіант завдання.

Факторизація цілого числа — це пошук нетривіального дільника (одного, не обов’язково простого) заданого числа. Якщо велике складене число розкладено на добуток менших цілих чисел, кожне з яких є простим числом, то такий розклад називається канонічним розкладом складеного числа. Таким чином, завданням задачі пошуку канонічного розкладу натурального числа є пошук всіх його простих дільників. Доведено, що задача пошуку канонічного розкладу числа та задача факторизації є поліноміально еквівалентними за часовою складністю. Ці задачі вважаються складними, тобто на сьогодні не існує ефективного алгоритму їх розв’язання в загальному випадку. Це й обумовлює використання цих задач в будові багатьох криптографічних примітивів, наприклад, криптосистеми RSA.

Однак існує багато алгоритмів факторизації, що мають експоненційну або субекспоненційну оцінку складності, або, наприклад є ефективними для деяких часткових випадків. В цьому комп’ютерному практикумі ми на практиці ознайомимося з деякими з них, а саме з методом пробних ділень, ρ-методом Полларда, алгоритмом Брілхарта-Моррісона (CFRAC) та алгоритмом Померанця (квадратичного сита), та спробуємо поєднати ці алгоритми для пошуку канонічного розкладу великого складеного числа.

Для розв’язання задачі пошуку канонічного розкладу числа необхідними є тести на простоту для перевірки чи є частка від ділення числа на простий дільник простим числом, чи необхідно продовжити розклад.

3. Хід роботи, опис труднощів, що виникали під час виконання завдання, та шляхи їх подолання.

3.1 Хід роботи:

1. Написано програми, що реалізовують такі алгоритми:

(а) тест на простоту Соловея-Штрассена;

(б) метод пробних ділень;

(в) ρ-метод Полларда;

(г) метод Брілхарта-Моррісона.

2. Створено та реалізовано алгоритм для пошуку канонічного розкладу числа, що використовує (всі) вищезгадані алгоритми:

(а) Вхід: n. Перевірити чи вхідне число n є простим. Якщо число є простим, то додати n до результату і завершити роботу, якщо число є складеним, перейти до наступного кроку.

(б) Вхід: n. Застосувати метод пробних ділень для пошуку дільників вхідного числа. Пробні ділення робити числами, що не перевищують 47. Якщо дільник a знайдено, записати його у вихідний результат і повернутися до кроку 2a з вхідним значенням n/a. Якщо дільник не знайдено перейти до кроку 2г.

(в) Вхід: n. Перевірити чи вхідне число є простим. Якщо число є простим, то додати n до результату і завершити роботу, якщо число є складеним, перейти до наступного кроку.

(г) Вхід: n. Застосувати ρ-метод Полларда. Якщо дільник a знайдено, записати його у вихідний результат і повернутися на один крок назад з числом n/a. Якщо дільник не знайдено перейти до кроку 2е.

(д) Вхід: n. Перевірити чи вхідне число є простим. Якщо число є простим, то додати n до результату і завершити роботу, якщо число є складеним, перейти до наступного кроку.

(е) Вхід: n. Застосувати метод Брілхарта-Моррісона. Якщо дільник a знайдено, записати його у вихідний результат і повернутися на один крок назад з числом n/a. Якщо дільник не знайдено завершити роботу з повідомленням "я не можу знайти канонічний розклад числа :(".

3. Застосувати алгоритм, створений на попередньому кроці, для пошуку канонічного розкладу числа відповідного варіанту.

3.2 Опис труднощів та шляхи їх подолання:

Алгоритм пробних ділень, агоритм Соловея-Штрасена та ρ-метод Полларда не викликали значних труднощів у програмній реалізації, чого не можна сказати про алгоритм Брілхарта-Моррісона. Останній виконувався протягом 2 місяців з перервами, але часто по цілими днями присвяченими йому і тільки йому. Цей єдиний алгоритм забрав у мене більше часу, ніж будь-який предмет в цілому цього семестру і який, врешті-решт, таки не працює ідеально на числах, які надані у комп’ютерному практикумі для факторизації (детальніше у наступному розділі). Подолав я цей камінь спотикання лише насиллям над власною мораллю та впертою працею (‘Ніхто так не дивується робочому коду як його автор.’).

4. Результати дослідження, зокрема результат канонічного розкладу числа відповідного варіанту.

4.1 Результати алгоритмів факторизації:

Число	ρ-метод Полларда	метод Брілхарта-Моррісона
9172639163	99971	NaN
8627969789	86341	NaN
8937716743	89387	NaN
278874899	2789	2789
99400891	9973	9967
116381389	11719	NaN
4252083239	65213	65213
6633776623	81457	NaN
227349247	98377	2311
3568572617	36083	36083

Таблиця 1

4.2 Канонічні розклади чисел:

Варіант	Число	Канонічний розклад
1	49347803087	[11, 397, 10009, 1129]
2	40314824977	[7, 547, 1049, 10037]
3	56872139357	[11, 1217, 10091, 421]
4	25169050345	[5, 10061, 491, 1019]
5	17807879769	[3, 1033, 569, 10099]
6	37007068943	[13, 269, 1051, 10069]
7	36478753357	[17, 211, 1009, 10079]
8	44729396957	[7, 487, 1307, 10039]
9	62728529929	[13, 1229, 389, 10093]
10	20057299527	[3, 577, 10067, 1151]

Таблиця 2

5. Результати продуктивності роботи програмної реалізації алгоритмів.

5.1 Час роботи алгоритмів факторизації (год:хв:сек.):

Число	ρ-метод Полларда	метод Брілхарта-Моррісона
9172639163	0:00:00	NaN
8627969789	0:00:00	NaN
8937716743	0:00:00	NaN
278874899	0:00:00	0:01:09.225412
99400891	0:00:00	0:00:53.555187
116381389	0:00:00	NaN
4252083239	0:00:00	0:05:10.674594
6633776623	0:00:00	NaN
227349247	0:00:00	0:09:05.255394
3568572617	0:00:00	0:42:17.399391

Таблиця 3

5.2 Час роботи канонічного розкладу чисел (год:хв:сек.):

Варіант	Число	Канонічний розклад
1	49347803087	0:00:00
2	40314824977	0:00:00.015629
3	56872139357	0:00:00
4	25169050345	0:00:00.015627
5	17807879769	0:00:00.031253
6	37007068943	0:00:00.015629
7	36478753357	0:00:00.015629
8	44729396957	0:00:00.031252
9	62728529929	0:00:00.015626
10	20057299527	0:00:00.031254

Таблиця 4

5.3 Час роботи перевірки числа на простоту (алгоритм Соловея-Штрассена) (год:хв:сек.):

Варіант	Число	Соловей-Штрассен
1	49347803087	0:00:00
2	40314824977	0:00:00
3	56872139357	0:00:00
4	25169050345	0:00:00
5	17807879769	0:00:00
6	37007068943	0:00:00
7	36478753357	0:00:00
8	44729396957	0:00:00
9	62728529929	0:00:00
10	20057299527	0:00:00
	524287	0:00:00.015624

Таблиця 5

6. Аналіз отриманих результатів.

6.1 Пошук нетривіального дільника:

У таблиці 1 в алгоритмі Брілхарта-Моррісона бачимо значення NaN – реалізація алгоритму не змогла виконати факторизацію відповідно числа із помилкою: RecursionError: maximum recursion depth exceeded while calling a Python Object. Це пояснюється наступним чином: для розв'язання СЛР я використовував перебір у вигляді рекурсивної функції, яка проходилася по датафрейму(по рядках – гладкі числа, а по стовпчиках – факторна база) спершу зліва направо, а потім зверху вниз і викликала сама ж себе у випадку знаходження значень, які давали би остачу 1 при діленні на 2, а в протилежному випадку викликала функцію знаходження чисел X та Y, які необхідні на фінальному кроці для факторизації (у разі незадовільного результату X = +- Y mod n фукнція 'забувала' поточний рядок/стовпчик і поверталася до попереднього стовпчика/рядка). Хоча попередньо і було проведено фільтрацію датафрейму, в якому видалялися стовпчики, що були лише заповнені 0 та тих, в яких містилось лише одне число і то те, що дає остачу 1 при діленні на 2, проте отримані датафрейми все одно були занадто великі для отримання результатів алгоритму, що був прописаний. Наприклад, для факторизації числа 9172639163 (варіант №1) остаточний датафрейм мав розмір 76 рядків на 23 стовпчики (10кб - .xlsx), а нефільтрований – 76 на 593 (110кб - .xlsx)
На відміну від метода Брілхарта-Моррісона ρ-метод Полларда не мав жодних проблем із результативністю. Усі значення були отримані швидко і якісно.

6.2 Час роботи:

У наслідок об'ємів даних, що опрацьовувалися, та методу 'в лоб' створення датафреймів та їх редагування (при якому видавало попередження: 'PerformanceWarning: DataFrame is highly fragmented. This is usually the result of calling `frame.insert` many times, which has poor performance.') алгоритм Брілхарта-Моррісона має дуже поганий час виконання.

Щодо ρ-методу Полларда, то час його виконання незрівняно менший, ніж у попереднього. Навіть функції datetime.now(), яка показує 6 знаків після коми для секунди, не вдалося заміряти час роботи алгоритму для чисел, наведених у практикумі. Шляхом випадкового натискання на клавіші цифр було отримано число 521064401567922879406069432539095585333583453753499728342798427984728937347498646749749243229848390805645835218221. Застосувавши ρ-метод Полларда отримаємо нетривіальний дільник 19 за час роботи: 0:00:00. Проте, якщо число таки просте, то буде і відповідний час виконання роботи. Наприклад, я взяв із джерел відкритого доступу декілька простих чисел: 1) 524287, 2) 2147483647, 3) 999999000001; та отримав наступні результати заміру часу роботи: 1) 0:00:00.015627, 2) 0:00:00.109386, 3) 0:00:03.295258. В усіх випадках алгоритм повернув значення 1.

Щодо канонічного розкладу числа, то час роботи є доволі таки хорошим. Припускаю, що алгоритм жодного разу не дійшов до методу Брілхарта-Моррісона☺.

Алгоритм Соловея-Штрассена має аналогічну ρ-методу Полларда ситуацію: час його робити мені не вдалося заміряти. Проте, варто пам'ятати, що тест Соловея-Штрассена є одностороннім.

7. Висновки.

У рамках комп’ютерного практикуму було розглянуто тест перевірки на простоту (Соловея-Штрассена) та 3 алгоритми факторизації чисел (метод пробних ділень, ρ-метод Полларда та алгоритм Брілхарта-Моррісона). Для кожного алгоритму була розроблена власна програмна реалізація мовою програмування Python у середовищі розробки PyCharm. Проведено факторизацію ряду чисел та заміряння часу роботи кожного алгоритму для кожного числа. Також було розроблено алгоритм канонічного розкладу числа, який містив у собі усі вищезгадані алгоритми та проведено заміряння часу його роботи.