Виконану мною роботу можна розділити на дві частини: теорія про паліндроми та програмна реалізація алгоритму знаходження найдовшої спільної підпослідовності(звертаю на це слово увагу) за завдяки цьому побудова паліндромів; теорія про алгоритми знаходження найдовшого спільного рядка(знов акцент уваги) за допомогою суфіксних дерев та алгоритми оптимізації їх побудови(зокрема алгоритм Укконена та Мак-Крейга). Спочатку було припущення, що будувати паліндроми можна завдяки алгоритму знаходження найдовшого спільного рядка, але виявилося, що цей алгоритм хоча і дає рядок-паліндром, кількість символів в результаті є зовсім невеликою, оскільки рядок – це безперервна послідовність символів, тобто рядок є окремим випадком послідовності, а значить майже завжди має меншу кількість символів, що завжди призводить до неправильної роботи алгоритмів додавання та видалення. Тому було вирішено використовувати алгоритм пошуку найбільшої спільної підпослідовності.

Перша частина – програма має реалізацію алгоритму знаходження найдовшої спільної підпослідовності (відтепер буду називати її **LCS**) заданого та віддзеркаленого від нього рядків. Ця підпослідовність сама по собі є паліндромом. Тому від цього можна відштовхуватися при створенні паліндромів на основі цього рядка. Сам алгоритм пошуку цієї підпослідовності я піддивився на цьому сайті: <https://www.programmingalgorithms.com/algorithm/longest-common-subsequence/>. Якщо коротко пояснювати, то цей алгоритм можна розділити надвоє: в першій частині ми посимвольно порівнюємо початковий та віддзеркалений рядки, і записуємо у деяку таблицю(2d масив) цифри починаючи з нуля, які змінюються при співпадінні двох однакових символів. А в другій частині ми, вже знаючи кількість букв у LCS, можемо створити масив, у який завдяки тому ж посимвольному порівнянню добавимо усі символи LCS до масиву, який потім перетворюємо на строку. Після знаходження LCS, можна перейти до створення паліндромів за допомогою алгоритмів додавання, видалення та заміни символів.

**Припущення на оптимізацію: чи можливо за допомогою 2d масиву визначити одразу індекси символів, що входять до LCS, тим самим пропустити другий цикл посимвольного порівняння?**

Алгоритм видалення є найпростішим, оскільки він просто бере строку LCS та видає її як результат. Це можливо, оскільки LCS сама по собі є паліндромом, тим більше, найбільшим можливим паліндромом, який є у початковому рядку. Далі по складності – створення паліндрому за допомогою заміни. У ньому все просто: виконується посимвольне порівняння та якщо знаходиться розбіжність – символ, який має найбільший індекс замінюється на символ, що має менший індекс. Тепер найскладніше – додавання. В ньому є декілька проблем: по-перше, додавання має відбуватися відносно середини паліндрому, тобто відносно середини LCS. Знайти індекс цієї середини можна тільки за допомогою посимвольного перебору. Звідси випливає проблема, що якщо у слові є декілька однакових букв, одна з котрих є серединою LCS, може статися неправильне знаходження цього індексу у початковому рядку і тому паліндром утвориться, але неправильно. По-друге, треба знайти індекси усіх неспарених символів цього рядка, а потім змінити їх так, щоб нулем був індекс середини LCS, неспарені символи, що йдуть перед ним, мали від’ємні індекси, а ті, що йдуть після нього – додатні. Тоді буде простіше додавати символи з іншої сторони відносно середини LCS. У проекті я зробив невеличку магію: спочатку можна повернути початковий рядок так, щоб найбільша кількість неспарених елементів була до середини LCS – завдяки цьому треба буде у цю частину ліву частину вставити меншу кількість неспарених символів, які знаходяться у правій частині. Після цього усі неспарені і спарені символи будуть знаходитися у лівій частині, тому праву частину можна просто видалити, а на її місце віддзеркалити ліву частину. Це має, хоча і ненабагато, але зменшити та спростити алгоритм.

Тепер перейдемо до порівняння роботи цих трьох алгоритмів. Якщо б ми робили чесно, наприклад, на бумазі, нам знадобилося б і для додавання, і для видалення додати або видалити однакову кількість символів, а саме кількість неспарених елементів рядка, тобто довжина рядка мінус довжина LCS. А вже алгоритм заміни потребує майже завжди більше замін, чим додавання. Він може бути найкращим вибором лише в одному випадку – коли неспарені елементи знаходяться симетрично відносно середини цього рядка.

Тепер можна перейти до суфіксних дерев. Суфіксні дерева – це спосіб зображення строки, який відображає усі суфікси цього рядка. Суфікс – це підрядок, початком якого може бути будь-який символ початкового рядка, окрім останнього, бо останній символ завжди є кінцем суфікса. І завдяки тому, що суфіксне дерево містить у собі будь-який підрядок якогось рядка, можна порівняти два суфіксних дерева двох різних рядків та визначити найбільший спільний підрядок. Зазвичай для позначення кінця рядку до нього додають символ, якого, за припущенням, нема в алфавіті цього рядка, тобто цього символу немає у цьому рядку. Частіш за все це символ $. Суфіксне дерево складається з декількох елементів:

1. Корінь – вершина, від якої йдуть усі дуги.
2. Внутрішня вершина – вершина, у яку веде 1 дуга, та з якої виходить 2 або більше дуг. Вона завжди означає, що до цього моменту суфікси мають однакові початки, але після цього моменту їх закінчення відрізняються. Їх буде завжди n (n дорівнює кількості символів цього рядка)
3. Кінцева вершина – вершина, у яку йде 1 дуга, та яка містить індекс початку того чи іншого суфіксу. Їх буде завжди n + 1 (+1 оскільки додається символ $, якого нема в алфавіті рядка)
4. Дуга – лінія, що об’єднує усі вершини та корінь. Дуги, які виходять з однієї вершини, ніколи не мають однаковий початковий символ.
5. Мітка дуги – містить підрядок суфікса, із них і складаються суфікси.