Курсовая работа по курсу дискретного анализа:

«Поиск наибольшей общей подпоследовательности (LCS)»

Выполнил студент группы 08-307 МАИ Дубровин Дмитрий.

Условие

Даны две строки со словами. Необходимо найти наибольшую общую подпоследовательность слов. Обратите внимание на ограничения по памяти, стандартное решение при помощи динамического программирования не подойдет. Попробуйте использовать асимптотически линейное по памяти решение

Формат ввода

Две строки со словами, разделенными пробелом.
В каждой строке не более 10^4 слов. Суммарная длина всех слов не превышает 2·10^5.

Формат вывода

В первой строке выведите одно число L — длину наибольшей общей подпоследовательности. Во второй строке через пробел выведите L слов — найденную подпоследовательность.

Метод решения

Для решения задачи поиска наибольшей общей подпоследовательности (LCS) в двух строках был выбран алгоритм Хиршберга. Этот алгоритм основан на применении динамического программирования и рекурсивного подхода для эффективного нахождения LCS.

Описание алгоритма Хиршберга

Описание задачи:

Даны последовательности $X=\langle x1, x2, \dots, xm\rangle$ и $Y=\langle y1, y2, \dots, yn\rangle$. Задача заключается в нахождении LCS (X, Y) за время (O(nm)) и использование (O(n + m)) памяти.

Без потери общности предположим, что m⩾n. Разделим последовательность X на две равные части: $x_1[0 \dots \frac{m}{2}]$ и $x_2[\frac{m}{2} + 1 \dots m]$. Затем найдем LCS для x_1 и всех префиксов последовательности Y, аналогично для развернутых последовательностей x_2 и Y.

Для нахождения LCS на всех префиксах достаточно одного квадратичного прохода, так как i-ый элемент последней строки результирующей матрицы есть LCS первой последовательности и префикса второй длины i.

Выберем индекс j так, чтобы |LCS(x_1,y[0…j]) + LCS(reverse(x_2),reverse(y[j+1…n]))| было максимальным. Затем запустим алгоритм рекурсивно для пар (x_1, y[0 ... j]) и ((x_2, y[j+1 … n])).

Процесс будет повторяться, пока в X не останется ровно 1 элемента. В этом случае проверим, входит ли этот элемент в текущую часть Y. Если входит, добавим его к ответу, в противном случае вернем пустую строку.

Таким образом, в результате работы алгоритма соберем последовательность, которая является искомой наибольшей общей подпоследовательностью.

Сравнение с обычным динамическим программированием:

1. Пространственная сложность:
   - Обычное динамическое программирование требует создания матрицы размером (m+1) x (n+1), что занимает (O(mn)) памяти.
   - Алгоритм Хиршберга требует (O (n + m)) памяти, что делает его более эффективным с точки зрения использования памяти.
2. Временная сложность:
   - Обычное динамическое программирование имеет временную сложность (O(mn)).
   - Алгоритм Хиршберга также имеет временную сложность (O(mn)), что делает его тоже эффективным с точки зрения времени выполнения.
3. Производительность:

- Алгоритм Хиршберга может оказаться эффективнее в случаях, когда требуется оптимизация использования памяти, особенно при больших значениях (m) и (n).
- Оба алгоритма дают одинаковый результат, но выбор между ними зависит от конкретных требований по использованию памяти и времени.

Описание кода для алгоритма Хиршберга

Код на C++ реализует алгоритм Хиршберга для поиска наибольшей общей подпоследовательности (LCS) между двумя последовательностями слов. Взглянем на основные аспекты:

В коде используется вектор resultSequence для сохранения элементов найденной наибольшей общей подпоследовательности.

Функция calculateLCS вычисляет длину LCS для двух половин последовательностей. Она использует матрицы current и previous для динамического программирования.

Функция hirschberg рекурсивно выполняет алгоритм Хиршберга для поиска LCS. Она обрабатывает базовые случаи, выбирает точку разделения для каждой половины, максимизирующую длину LCS, и выполняет рекурсивные вызовы для обеих половин.

Функция reverseHirschberg обрабатывает случай, когда первая последовательность короче второй. В зависимости от этого вызывается алгоритм Хиршберга с соответствующим порядком аргументов.

В main считываются две строки слов, заполняются векторы, разворачиваются последовательности, и вызывается reverseHirschberg для нахождения LCS. Результат сохраняется в resultSequence, и выводится размер и содержимое этой последовательности.

Тест производительности

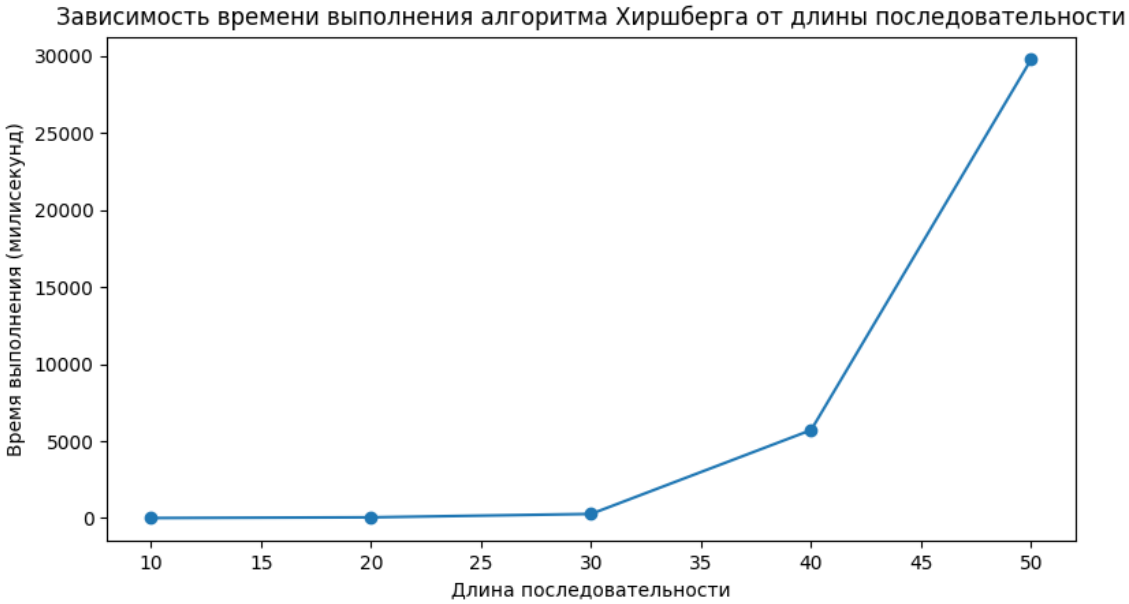

График представляет зависимость времени выполнения алгоритма Хиршберга от длины последовательности. По оси X отмечены различные длины последовательностей (например, 10, 20, 30, 40, 50), а по оси Y — время выполнения в микросекундах.

В начале графика при небольших длинах последовательностей ожидается низкое время выполнения. После чего, с увеличением длины последовательностей, график должен демонстрировать увеличение времени выполнения. Это может произойти линейно или более сложным образом, в зависимости от временной сложности алгоритма.

Временная сложность алгоритма Хиршберга:

Алгоритм Хиршберга имеет временную сложность O(nm), где n и m - длины последовательностей X и Y соответственно. При использовании алгоритма для поиска LCS (наибольшей общей подпоследовательности) двух последовательностей, алгоритм разбивает задачу на более мелкие подзадачи и решает их, используя динамическое программирование.

Однако, поскольку алгоритм Хиршберга использует стратегию "разделяй и властвуй" с оптимизацией по промежуточным результатам, его эффективная временная сложность может быть снижена до O(n(m+n)) при использовании линейного времени на каждом уровне рекурсии. Это делает

его более эффективным по сравнению с прямым применением динамического программирования для данной задачи.

Вывод

В ходе данной работы был представлен и реализован алгоритм Хиршберга для нахождения наибольшей общей подпоследовательности (LCS) двух последовательностей. Алгоритм использует стратегию "разделяй и властвуй", разбивая задачу на подзадачи, а затем эффективно комбинируя результаты.

Основным преимуществом алгоритма Хиршберга является его оптимизация по памяти, требующая O(n + m) дополнительной памяти, что делает его более эффективным для больших последовательностей по сравнению с классическим динамическим программированием, требующим O(nm) памяти.

Также был предоставлен исходный код на C++, а также описаны шаги алгоритма. Для проверки корректности работы алгоритма был предложен вопрос о генерации тестов, и, при необходимости, тесты могут быть созданы с использованием Python.

В контексте производительности, было предложено построение графика зависимости времени выполнения алгоритма от длины последовательности.