Самоподстраивающееся динамическое программирование на примере задачи о поиске наибольшей общей подпоследовательности

Шовкопляс Г.Ф., студент 2-го курса магистратуры ф-та ИТиП Университета ИТМО, grigory.96@gmail.com

Научный руководитель: Буздалов М. В., к.т.н., доц. ф-та ИТиП Университета ИТМО

Аннотация

В ходе данной работы предполагается рассмотреть пересчет динамического программирования при изменении входных данных, использующий предыдущие значения, и сравнить с полным пересчетом динамического программирования.

Введение

Динамическое программирование — метод решения задачи путём её разбиения на несколько подзадач меньшего размера, рекуррентно связанных между собой. Класс задач решаемых данным методом довольно обширен и включает в себя задачи на такие популярные темы, как поиск кратчайших путей в графах и задачи на работу со строками. Однако, в случае изменения начальных данных (возможно, незначительного), задачу приходится решать заново, что есть ресурсозатратно. Не исключено, что есть возможность упростить решение задачи при изменении начальных данных.

Задача о поиске наибольшей общей подпоследовательности

Подпоследовательность данной последовательности — это последовательность, получаемая из исходной удалением нуля или более элементов.

Последовательность Z является общей подпоследовательностью последовательностей X и Y, если Z является подпоследовательностью как X, так и Y.

Рассмотрим стандартное решение поставленной задачи с использованием метода динамического программирования. В двумерном массиве $lcs_{i,j}$ будем хранить длину наибольшей общей подпоследовательности двух последовательностей, составленных из i первых элементов первой последовательности и j первых элементов второй последовательности, соответственно. Формула подсчета динамического программирования для входных последовательностей X и Y:

$$lcs_{i,j} = \begin{cases} 0, & i = 0 \text{ or } j = 0 \\ lcs_{i-1,j-1} + 1, & A_i = B_j \\ \max(lcs_{i-1,j}, lcs_{i,j-1}), & A_i \neq B_j \end{cases}$$

Пересчет при изменении входных данных

Рассмотрим изменение некоторого элемента с индексом k второй последовательности.

Простая оптимизация

Заметим, что после данного изменения значения $lcs_{i,j}$ для j меньших k не изменится. В силу этого можно начать пересчет значений начиная с j равного k.

Алгоритм с использованием обхода в ширину

Будем поддерживать очередь значений, которые потенциально могли измениться. Изначально это $lcs_{i,k}$ для любого i.

Если значение $lcs_{i,j}$ изменилось, добавим ее «соседей» в очередь. Здесь под «соседями» подразумеваются значения, которые зависят от значения $lcs_{i,j}$, а именно: $lcs_{i+1,j}$, $lcs_{i,j+1}$ и $lcs_{i+1,j+1}$. Заметим, что нужна не очередь, а дек, так как значения в столбцах нужно пересчитывать в порядке увеличения номера столбца.

Проблема этого алгоритма в том, что он использует $O(n^2)$ дополнительной памяти при размере входных данных $O(n)$, а также имеет не самую приятную константу в асимптотике времени.

Тем не менее на рис. 1 приведен пример, показывающий что таких состояний обычно не очень много.

Оптимизация предыдущего алгоритма

Применим следующие оптимизации, чтобы избавиться от деков и двумерного хранения массива посещенных состояний.

		1	1	1	1	4	3	4	1	2	4	4	2	3	2	2	4	3	1	4	1
	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1
1	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	2	2	2
3	0	1	1	1	1	1	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2
1	0	1	2	2	2	2	2	2	3	3	3	3	3	3	3	3	3	3	3	3	3
4	0	1	2	2	2	3	3	3	3	3	4	4	4	4	4	4	4	4	4	4	4
4	0	1	2	2	2	3	3	4	4	4	4	5	5	5	5	5	5	5	5	5	5
4	0	1	2	2	2	3	3	4	4	4	5	5	5	5	5	5	6	6	6	6	6
4	0	1	2	2	2	3	3	4	4	4	5	6	6	6	6	6	6	6	6	7	7
2	0	1	2	2	2	3	3	4	4	5	5	6	7	7	7	7	7	7	7	7	7
1	0	1	2	3	3	3	3	4	5	5	5	6	7	7	7	7	7	7	8	8	8
4	0	1	2	3	3	4	4	4	5	5	6	6	7	7	7	7	8	8	8	9	9
1	0	1	2	3	4	4	4	4	5	5	6	6	7	7	7	7	8	8	9	9	10
4	0	1	2	3	4	5	5	5	5	5	6	7	7	7	7	7	8	8	9	10	10
4	0	1	2	3	4	5	5	6	6	6	6	7	7	7	7	7	8	8	9	10	10
1	0	1	2	3	4	5	5	6	7	7	7	7	7	7	7	7	8	8	9	10	11
4	0	1	2	3	4	5	5	6	7	7	8	8	8	8	8	8	8	8	9	10	11
3	0	1	2	3	4	5	6	6	7	7	8	8	8	9	9	9	9	9	9	10	11
2	0	1	2	3	4	5	6	6	7	8	8	8	9	9	10	10	10	10	10	10	11
1	0	1	2	3	4	5	6	6	7	8	8	8	9	9	10	10	10	10	11	11	11
3	0	1	2	3	4	5	6	6	7	8	8	8	9	10	10	10	10	11	11	11	11

Рис. 1: Значения НОП двух последовательностей в виде таблицы. Зеленым обозначен измененный элемент второй последовательности, красным пересчитанные значения.

- Будем идти по столбцам, последовательно запоминая потенциальные значения следующего столбца, которые могли измениться.
- Заметим, что из каждого состояния в начало дека может поместиться не более одного состояния.
- Вместо помещения в начало дека будем заменять значение в списке.

Графики

В этом разделе приведены графики зависимости числа состояний, необходимых для пересчета от диапазона значений последовательностей. Последовательности выбраны, не умаляя общности, одинаковой длины.

Для каждого диапазона значений было проведено по десять итераций пересчета при случайном изменении.

По графикам на рис. 2 и рис. 3 можно заметить, что изменяемых состояний в среднем сильно меньше, чем теоритически могло измениться. Также видно, что при увеличении диапазона значений, число изменяемых состояний уменьшается.

Рис. 2: График зависимости числа состояний, необходимых для пересчета от диапазона значений последовательностей для двух последовательностей длины 100.

Рис. 3: График зависимости числа состояний, необходимых для пересчета от диапазона значений последовательностей для двух последовательностей длины 1000.

По графику на рис. 4 видно, что даже при неаккуратной реализации второго метода он в среднем работает быстрее полного пересчета.

Заключение

В данной работе предложен алгоритм пересчета значений динамического программирования для задачи НОП.

Результаты располагают к продолжению работы над оптимизацией данного алгоритма, а также рассмотрению подобных алгоритмов для других задач динамического программирования.

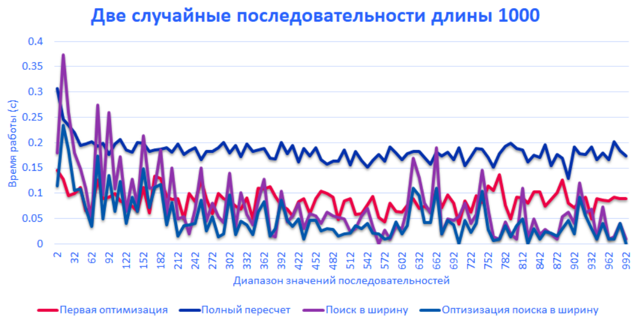

Рис. 4: График зависимости времени исполнения от диапазона значений последовательностей для двух последовательностей длины 1000.

Литература

[1] Томас Х. Кормен, Чарльз И. Лейзерсон, Рональд Л. Ривест, Клиффорд Штайн Алгоритмы: построение и анализ — 2-е изд. — М.: «Вильямс», 2007. — с. 459.