Динамическое программирование: наибольшая возрастающая подпоследовательность

Александр Куликов

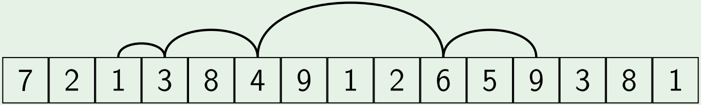
Онлайн-курс «Алгоритмы: теория и практика. Методы»
http://stepic.org/217

Наибольшая возрастающая подпоследовательность

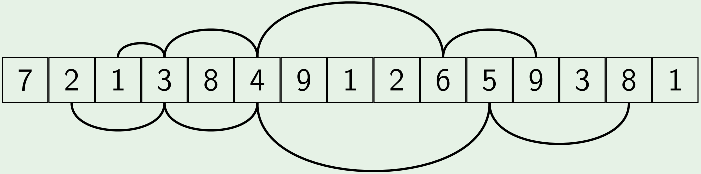
Вход: последовательность $A[1 \dots n] = [a_1, a_2, \dots, a_n]$.

Выход: наибольшая возрастающая подпоследовательность (НВП), то есть $a_{i_1}, a_{i_2}, \dots, a_{i_k}$ такие что $i_1 < i_2 < \dots < i_k$, $a_{i_1} < a_{i_2} < \cdots < a_{i_k}$ и k максимально.

Пример

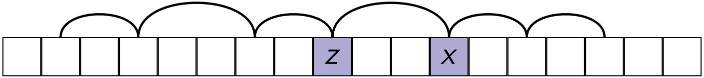


Пример



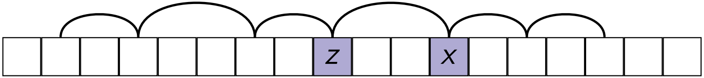
Подзадачи

- Рассмотрим НВП и два её соседних элемента:



Подзадачи

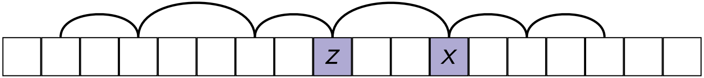
- Рассмотрим НВП и два её соседних элемента:



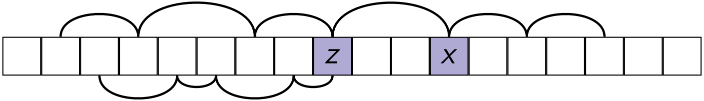
- Ясно, что $z < x$.

Подзадачи

- Рассмотрим НВП и два её соседних элемента:

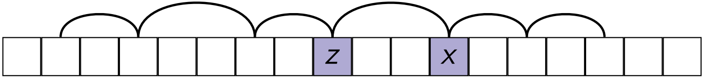


- Ясно, что $z < x$.
- Более того, префикс НВП, заканчивающийся в z, должен быть оптимальным, поскольку в противном случае НВП не была бы оптимальной:

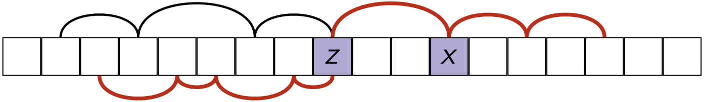


Подзадачи

- Рассмотрим НВП и два её соседних элемента:

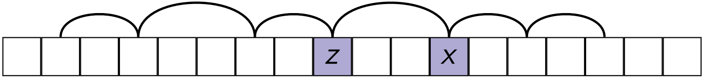


- Ясно, что $z < x$.
- Более того, префикс НВП, заканчивающийся в z, должен быть оптимальным, поскольку в противном случае НВП не была бы оптимальной:

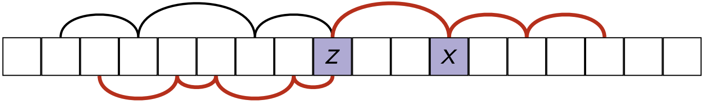


Подзадачи

- Рассмотрим НВП и два её соседних элемента:



- Ясно, что $z < x$.
- Более того, префикс НВП, заканчивающийся в z, должен быть оптимальным, поскольку в противном случае НВП не была бы оптимальной:



- Оптимальность для подзадач методом «вырезать и вставить».

Подзадачи и рекуррентное соотношение

- Пусть $D[i]$ — длина НВП, заканчивающейся в $A[i]$.

Подзадачи и рекуррентное соотношение

- Пусть $D[i]$ — длина НВП, заканчивающейся в $A[i]$.
- Тогда

$$D[i] = 1 + \max\{D[j] \colon j < i \text{ и } A[j] < A[i]\}\,.$$

Подзадачи и рекуррентное соотношение

- Пусть $D[i]$ — длина НВП, заканчивающейся в $A[i]$.
- Тогда

$$D[i] = 1 + \max\{D[j] \colon j < i \text{ и } A[j] < A[i]\}\,.$$

- Таким образом, вместо решения исходной задачи мы будем решать множество подзадач того же типа.

Функция $\textsc{LISBottomUp}(A[1 \ldots n])$

```
создать массив D[1...n]
для i от 1 до n:
    D[i] ← 1
    для j от 1 до i − 1:
        если A[j] < A[i] и D[j] + 1 > D[i]:
            D[i] ← D[j] + 1
ans ← 0
для i от 1 до n:
    ans ← max(ans, D[i])
вернуть ans
```

Время работы

Время работы алгоритма квадратично, поскольку он перебирает все пары (i, j), где $1 \leq j < i \leq n$. Количество таких пар равно

$$\binom{n}{2} = \frac{n(n-1)}{2} = \Theta(n^2)\,.$$

Восстановление решение

- Пока что мы научились искать длину оптимальной возрастающей подпоследовательности. Теперь научимся находить и саму подпоследовательность.
- Чтобы восстановить её, для каждой задачи будем поддерживать не только оптимальную длину, но и некоторую дополнительную информацию, показывающую, на какой именно подзадаче реализовалась оптимальная длина.

Функция $\texttt{LISBottomUp2}(A[1\dots n])$

создать массивы $D[1\dots n]$ и $\texttt{prev}[1\dots n]$
для i от 1 до n:
    $D[i] \leftarrow 1$, $\texttt{prev}[i] \leftarrow -1$
    для j от 1 до $i-1$:
        если $A[j] < A[i]$ и $D[j]+1 > D[i]$:
            $D[i] \leftarrow D[j]+1$, $\texttt{prev}[i] \leftarrow j$

$ans \leftarrow 0$
для i от 1 до n:
    $ans = \max(ans, D[i])$
вернуть ans

Пример

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
A	7	2	1	3	8	4	9	1	2	6	5	9	3	8	1
D	1	1	1	2	3	3	4	1	2	4	4	5	3	5	1
`prev`	-1	-1	-1	2	4	4	5	-1	3	6	6	10	9	10	-1

Пример

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
A	7	2	1	3	8	4	9	1	2	6	5	9	3	8	1
D	1	1	1	2	3	3	4	1	2	4	4	5	3	5	1
`prev`	-1	-1	-1	2	4	4	5	-1	3	6	6	10	9	10	-1

Пример

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
A	7	2	1	3	8	4	9	1	2	6	5	9	3	8	1
D	1	1	1	2	3	3	4	1	2	4	4	5	3	5	1
`prev`	-1	-1	-1	2	4	4	5	-1	3	6	6	10	9	10	-1

Пример

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
A	7	2	1	3	8	4	9	1	2	6	5	9	3	8	1
D	1	1	1	2	3	3	4	1	2	4	4	5	3	5	1
`prev`	-1	-1	-1	2	4	4	5	-1	3	6	6	10	9	10	-1

Пример

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
A	7	2	1	3	8	4	9	1	2	6	5	9	3	8	1
D	1	1	1	2	3	3	4	1	2	4	4	5	3	5	1
`prev`	-1	-1	-1	2	4	4	5	-1	3	6	6	10	9	10	-1

Пример

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
A	7	2	1	3	8	4	9	1	2	6	5	9	3	8	1
D	1	1	1	2	3	3	4	1	2	4	4	5	3	5	1
`prev`	-1	-1	-1	2	4	4	5	-1	3	6	6	10	9	10	-1

Пример

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
A	7	2	1	3	8	4	9	1	2	6	5	9	3	8	1
D	1	1	1	2	3	3	4	1	2	4	4	5	3	5	1
`prev`	-1	-1	-1	2	4	4	5	-1	3	6	6	10	9	10	-1

Пример

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
A	7	2	1	3	8	4	9	1	2	6	5	9	3	8	1
D	1	1	1	2	3	3	4	1	2	4	4	5	3	5	1
`prev`	-1	-1	-1	2	4	4	5	-1	3	6	6	10	9	10	-1

Пример

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
A	7	2	1	3	8	4	9	1	2	6	5	9	3	8	1
D	1	1	1	2	3	3	4	1	2	4	4	5	3	5	1
`prev`	-1	-1	-1	2	4	4	5	-1	3	6	6	10	9	10	-1

Пример

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
A	7	2	1	3	8	4	9	1	2	6	5	9	3	8	1
D	1	1	1	2	3	3	4	1	2	4	4	5	3	5	1
`prev`	-1	-1	-1	2	4	4	5	-1	3	6	6	10	9	10	-1

Пример

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
A	7	2	1	3	8	4	9	1	2	6	5	9	3	8	1
D	1	1	1	2	3	3	4	1	2	4	4	5	3	5	1
`prev`	-1	-1	-1	2	4	4	5	-1	3	6	6	10	9	10	-1

Восстановление ответа

создать массив $L[1 \ldots ans]$ {индексы НВП}

Восстановление ответа

создать массив $L[1 \ldots ans]$ {индексы НВП}
$k \leftarrow 1$
для i от 2 до n:
  если $D[i] > D[k]$:
    $k \leftarrow i$

Восстановление ответа

```
создать массив L[1 ... ans]    {индексы НВП}
k ← 1
для i от 2 до n:
   если D[i] > D[k]:
      k ← i
j ← ans
пока k > 0:
   L[j] ← k
   j ← j − 1
   k ← prev[k]
```

Восстановление ответа

```
создать массив L[1...ans]    {индексы НВП}
k ← 1
для i от 2 до n:
   если D[i] > D[k]:
      k ← i
j ← ans
пока k > 0:
   L[j] ← k
   j ← j − 1
   k ← prev[k]
```

Время работы: $O(n)$.

Восстановление без `prev`

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
A	7	2	1	3	8	4	9	1	2	6	5	9	3	8	1
D	1	1	1	2	3	3	4	1	2	4	4	5	3	5	1

Восстановление без `prev`

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
A	7	2	1	3	8	4	9	1	2	6	5	9	3	8	1
D	1	1	1	2	3	3	4	1	2	4	4	5	3	5	1

Восстановление без `prev`

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
A	7	2	1	3	8	4	9	1	2	6	5	9	3	8	1
D	1	1	1	2	3	3	4	1	2	4	4	5	3	5	1

Восстановление без `prev`

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
A	7	2	1	3	8	4	9	1	2	6	5	9	3	8	1
D	1	1	1	2	3	3	4	1	2	4	4	5	3	5	1

Восстановление без `prev`

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
A	7	2	1	3	8	4	9	1	2	6	5	9	3	8	1
D	1	1	1	2	3	3	4	1	2	4	4	5	3	5	1

Факт

Существует алгоритм, находящий наибольшую возрастающую подпоследовательность за время $O(n \log n)$.

Заключение

- Оптимальность для подзадач: любой префикс НВП является НВП, заканчивающейся в данном элементе.
- Подзадачи: длина НВП, заканчивающейся в i-м элементе исходной последовательности.
- Решаем подзадачи снизу вверх, от $i = 1$ до $i = n$.
- Решение может быть восстановлено как с использованием дополнительной информации, так и просто из ответов для всех подзадач.