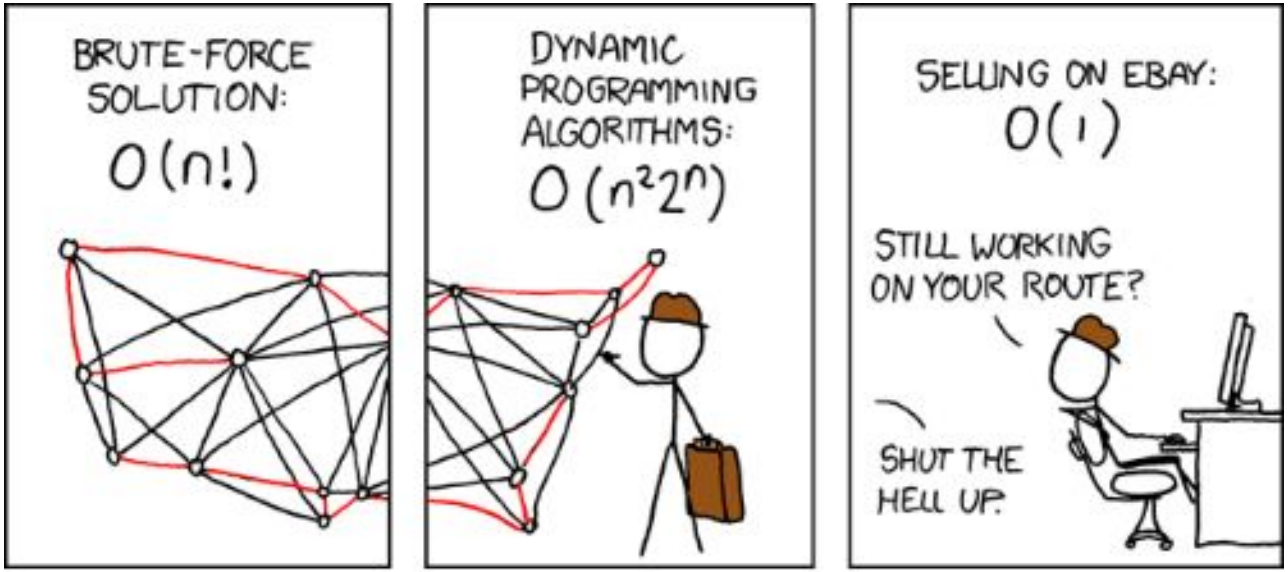
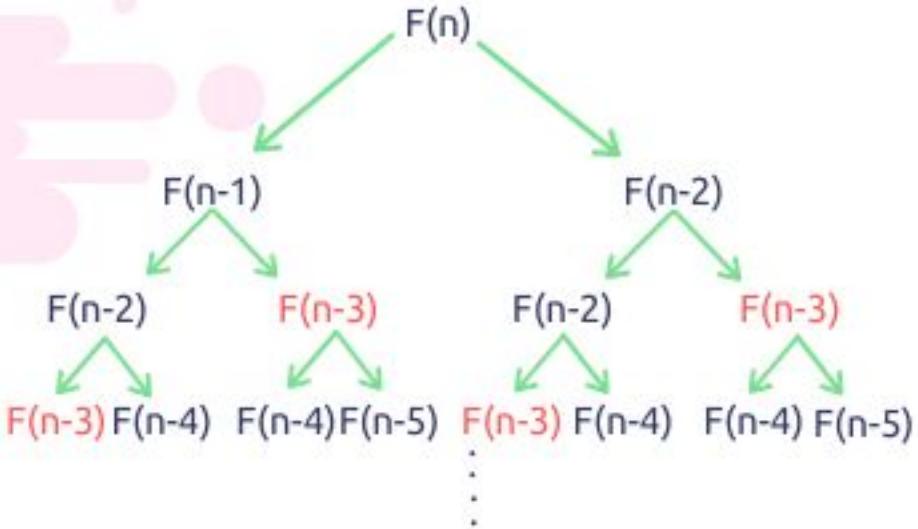
Дивизион D. День 1
Динамическое программирование

Лектор: Дмитрий Руденко



The Traveling Salesman...

Algorithms



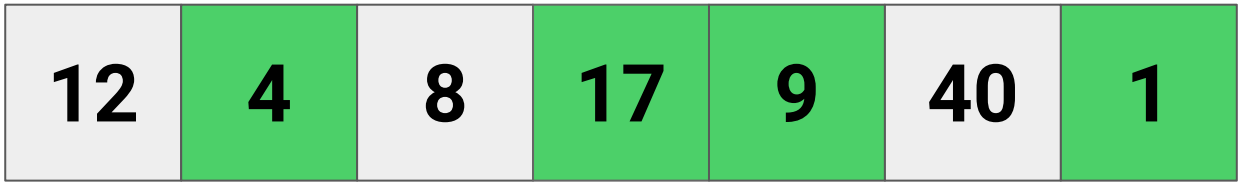
Fibonacci Recursion and Dynamic Programming

Наибольшая возрастающая подпоследовательность

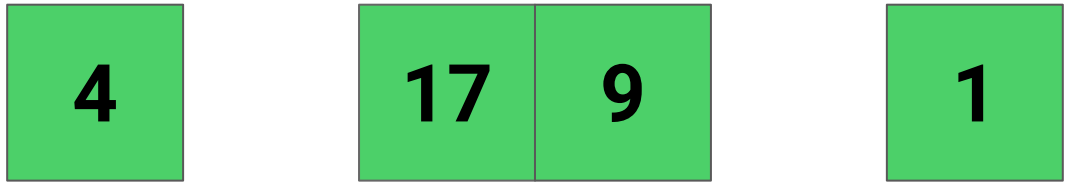
Наибольшая возрастающая подпоследовательность

12	4	8	17	9	40	1

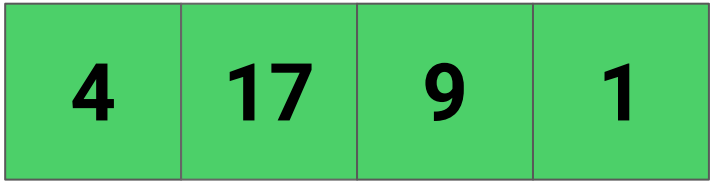
Наибольшая возрастающая подпоследовательность



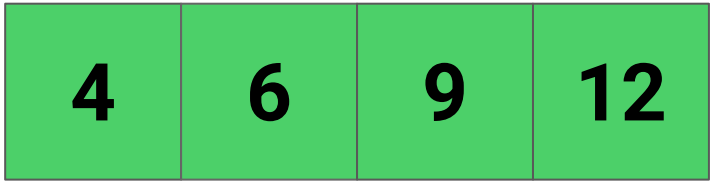
Наибольшая возрастающая подпоследовательность



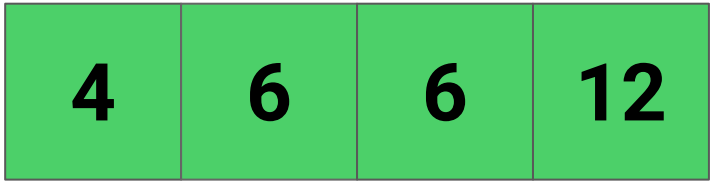
Наибольшая возрастающая подпоследовательность



Наибольшая возрастающая подпоследовательность



Наибольшая возрастающая подпоследовательность



Наибольшая возрастающая подпоследовательность

Формальное определение

Пусть дан массив a размера n, наибольшая возрастающая подпоследовательность (НВП) для него это такая подпоследовательность $a_{i1} < a_{i2} < \ldots < a_{ik}$, что:

- $i_1 < i_2 < \ldots < i_k$
- $0 <= i < n$
- k – максимально возможное

Наибольшая возрастающая подпоследовательность

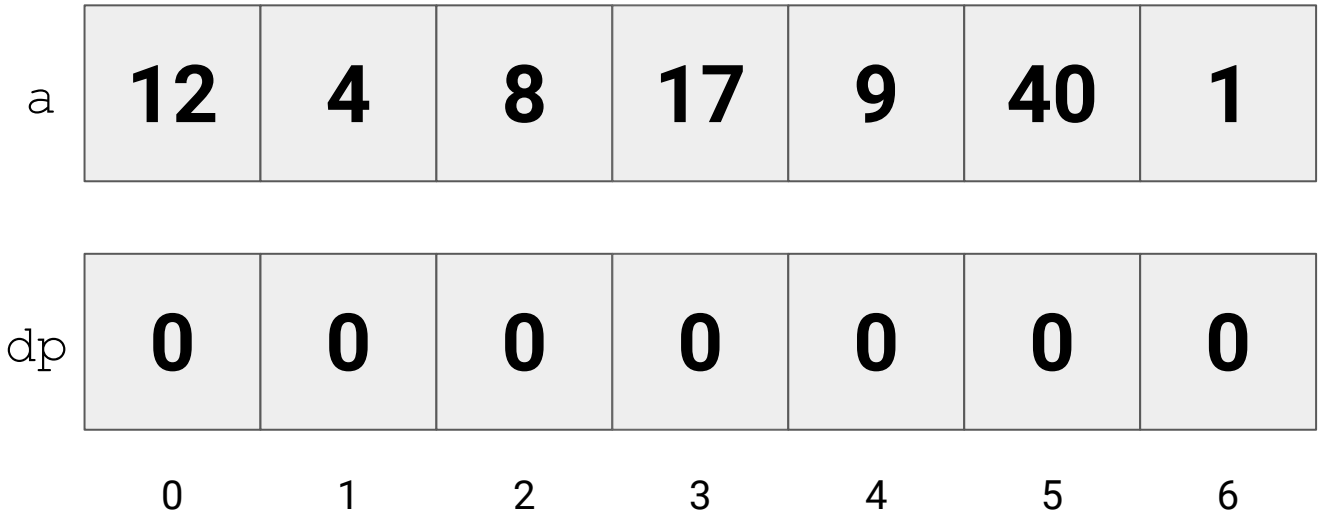
Наивное решение

1) Давайте переберем все подпоследовательности за $O(2^n)$
2) Проверим каждую подпоследовательность на условие “возрастающей последовательности” за $O(n)$
3) Среди всех найденных выберем максимальную
4) Итоговая сложность алгоритма $O(2^n * n)$

Наибольшая возрастающая подпоследовательность

Решение с помощью ДП

Введем массив `dp`
В `dp[i]` будем хранить длину НВП, которая оканчивается элементом на позиции `i`



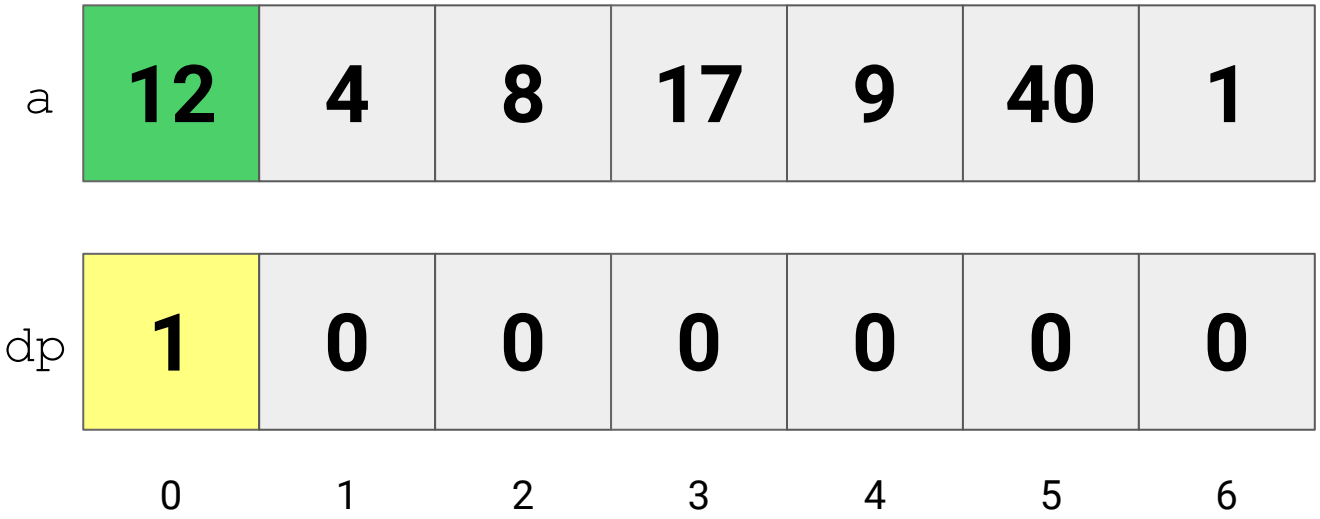
Наибольшая возрастающая подпоследовательность

Решение с помощью ДП

1) Рассматриваем элементы слева направо
2) Пусть текущий элемент `a[i]`
3) Если итоговая последовательность имеет длину `1`, то `dp[i] = 1`
4) Если больше `1`, то перед `a[i]` существует такой элемент `a[j]`, что `j < i`
5) Так как мы уже знаем ответы для всех `a[j]`, то давайте найдем найдем среди всех `a[j]`, тот который имеет наибольшую подпоследовательность и добавим к ней элемент `a[i]`
6) Длина получившейся последовательности будет равняться `dp[j] + 1`

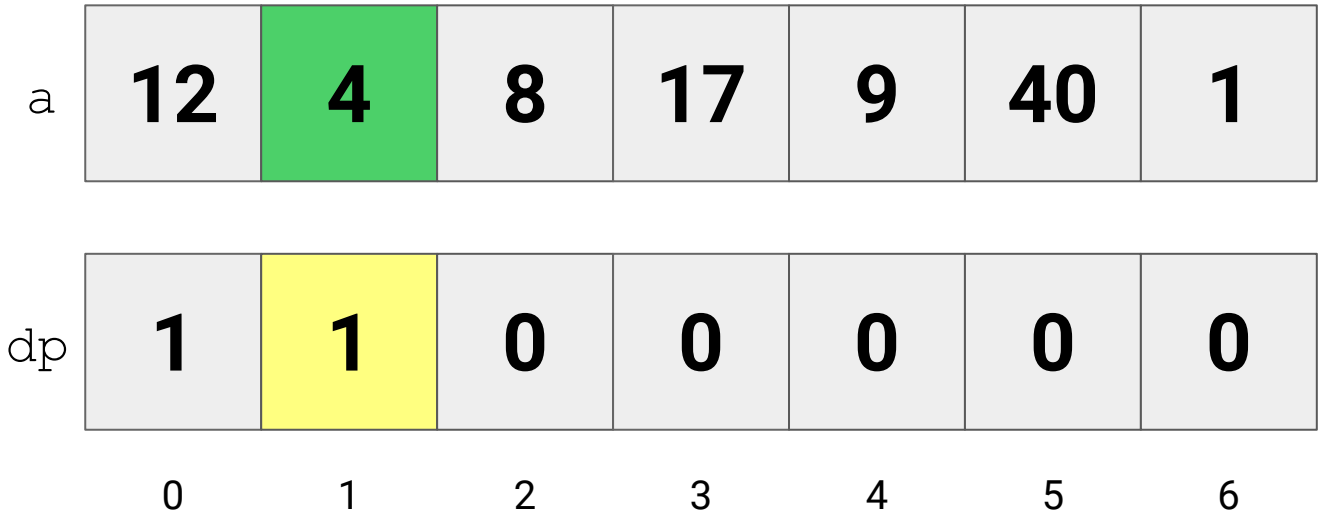
Наибольшая возрастающая подпоследовательность

Заполнение динамики



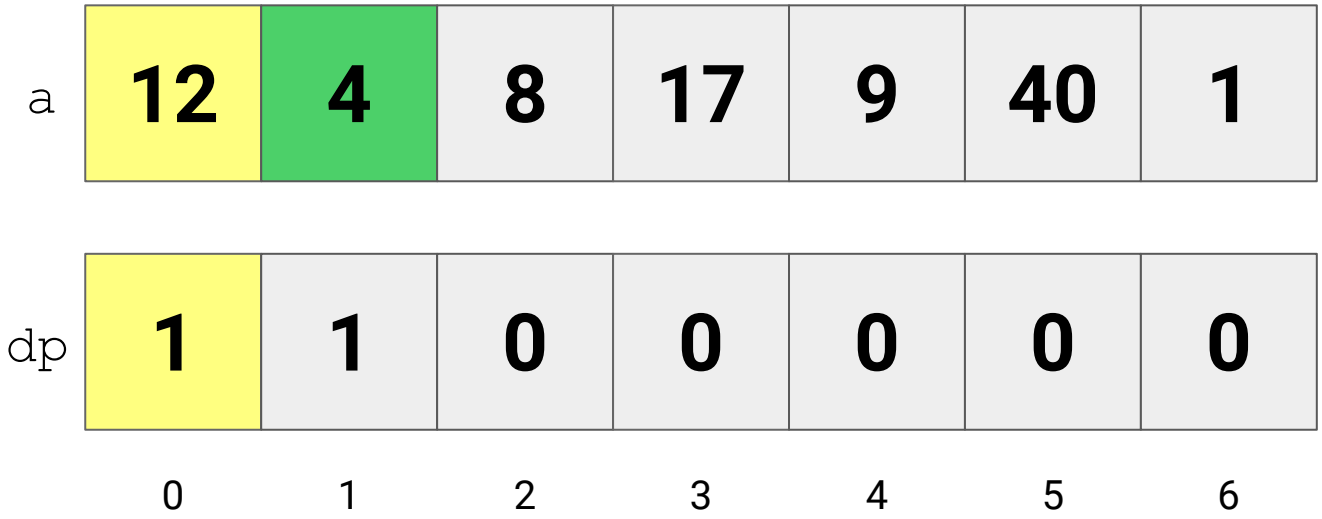
Наибольшая возрастающая подпоследовательность

Заполнение динамики



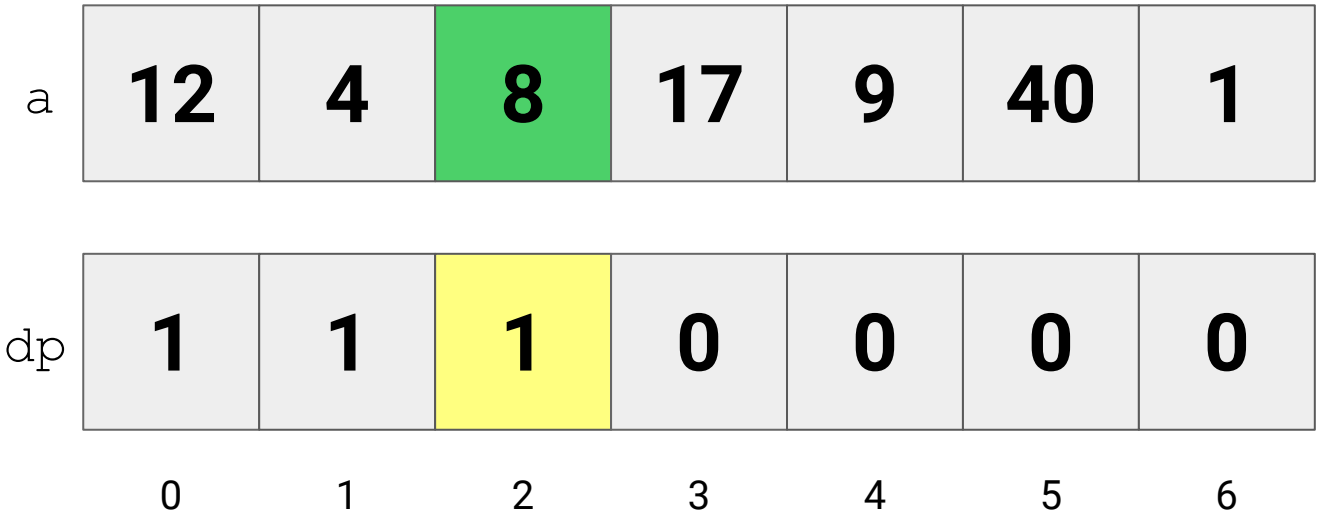
Наибольшая возрастающая подпоследовательность

Заполнение динамики



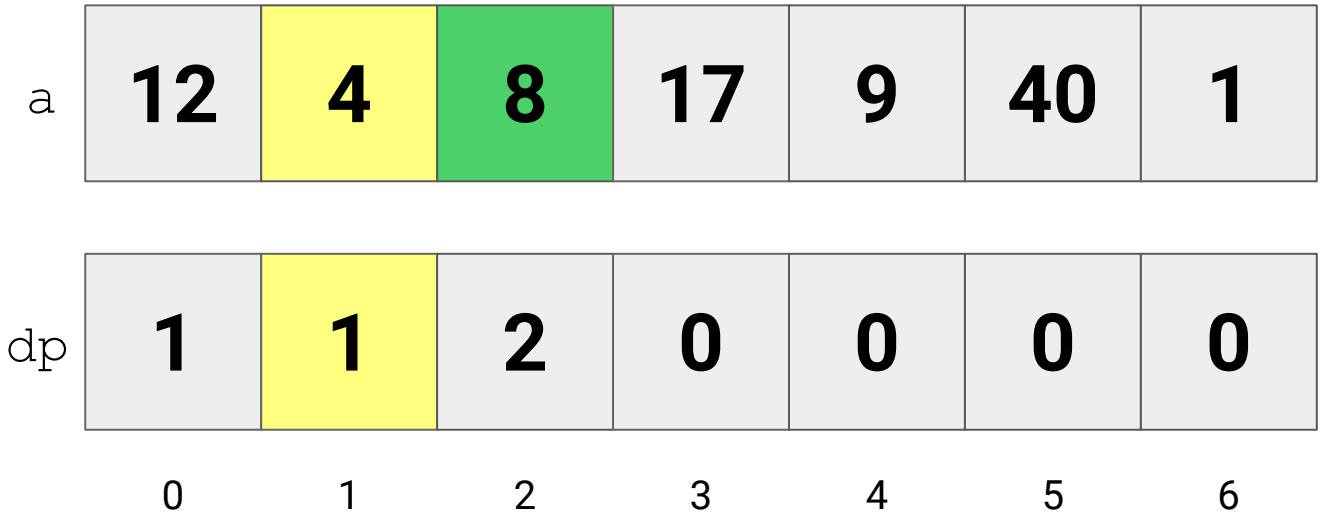
Наибольшая возрастающая подпоследовательность

Заполнение динамики



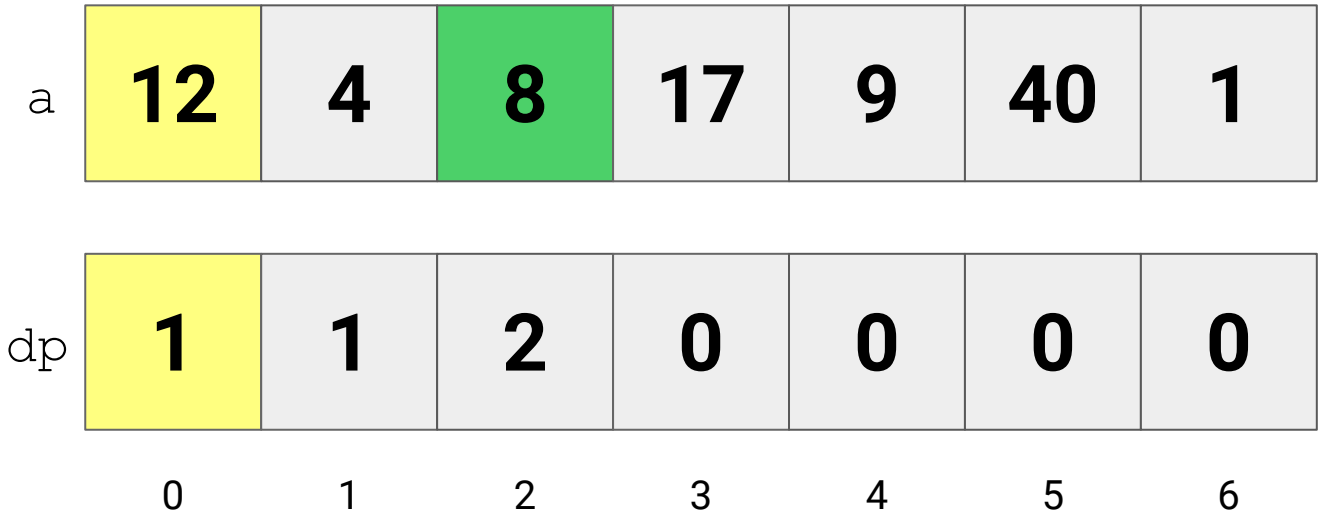
Наибольшая возрастающая подпоследовательность

Заполнение динамики



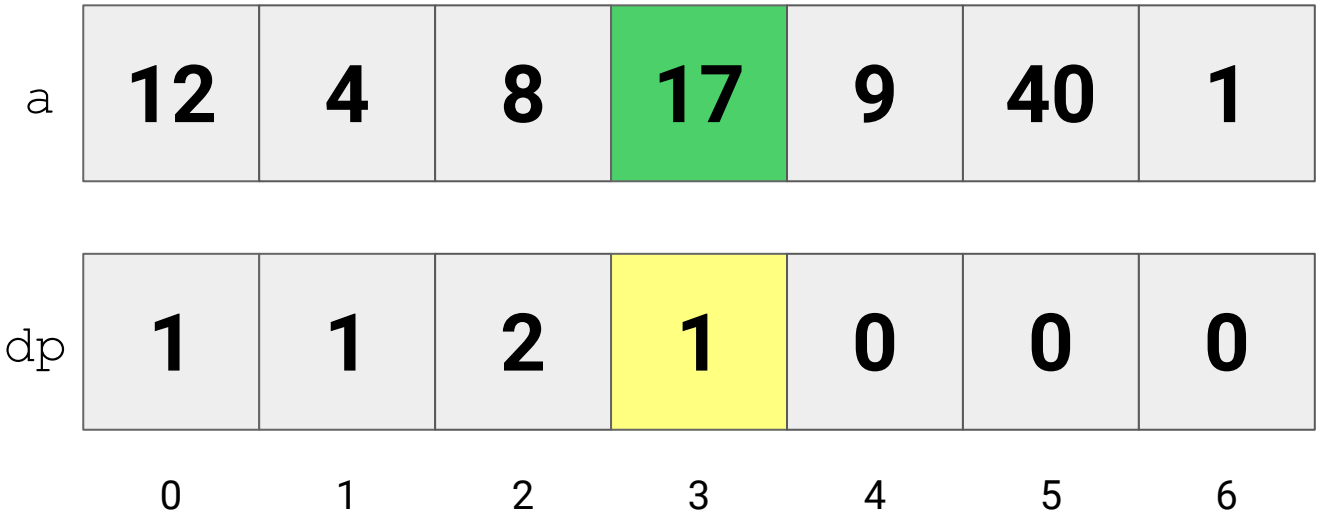
Наибольшая возрастающая подпоследовательность

Заполнение динамики



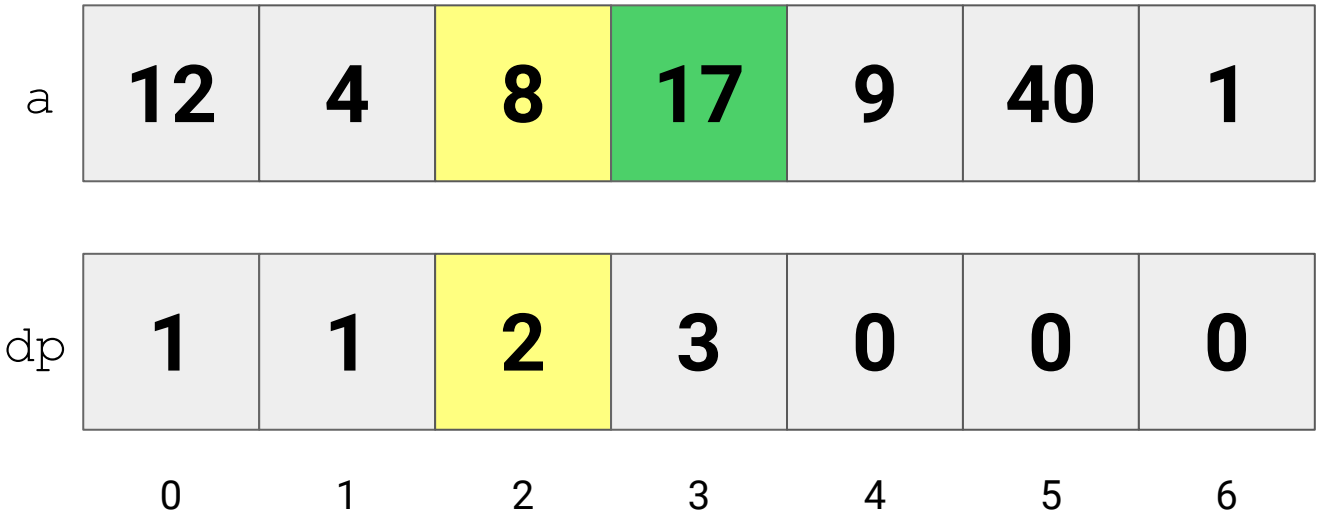
Наибольшая возрастающая подпоследовательность

Заполнение динамики



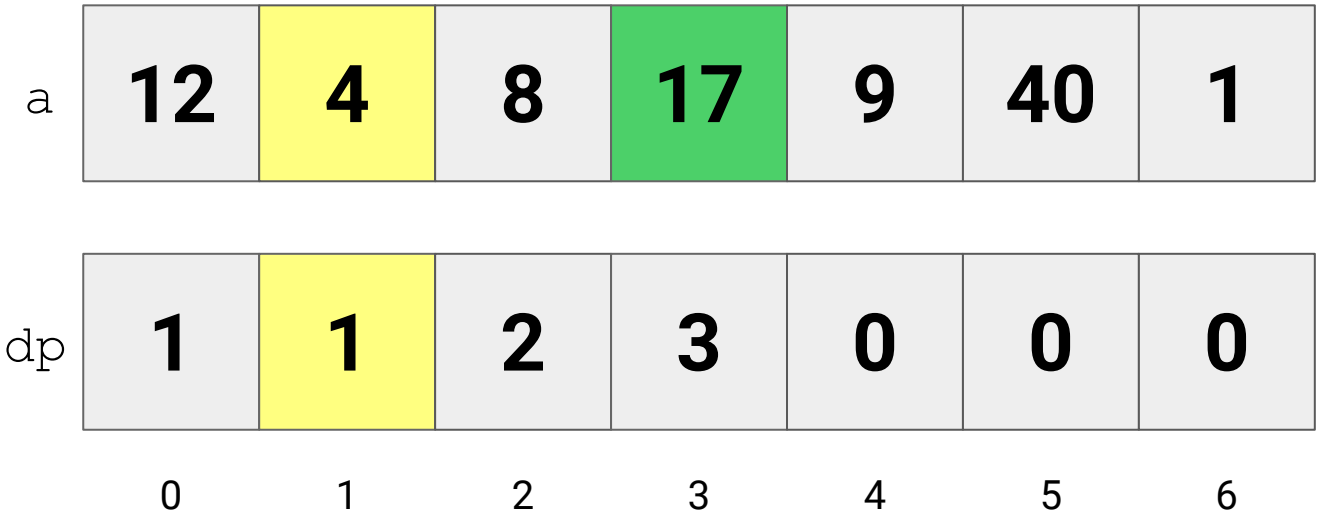
Наибольшая возрастающая подпоследовательность

Заполнение динамики



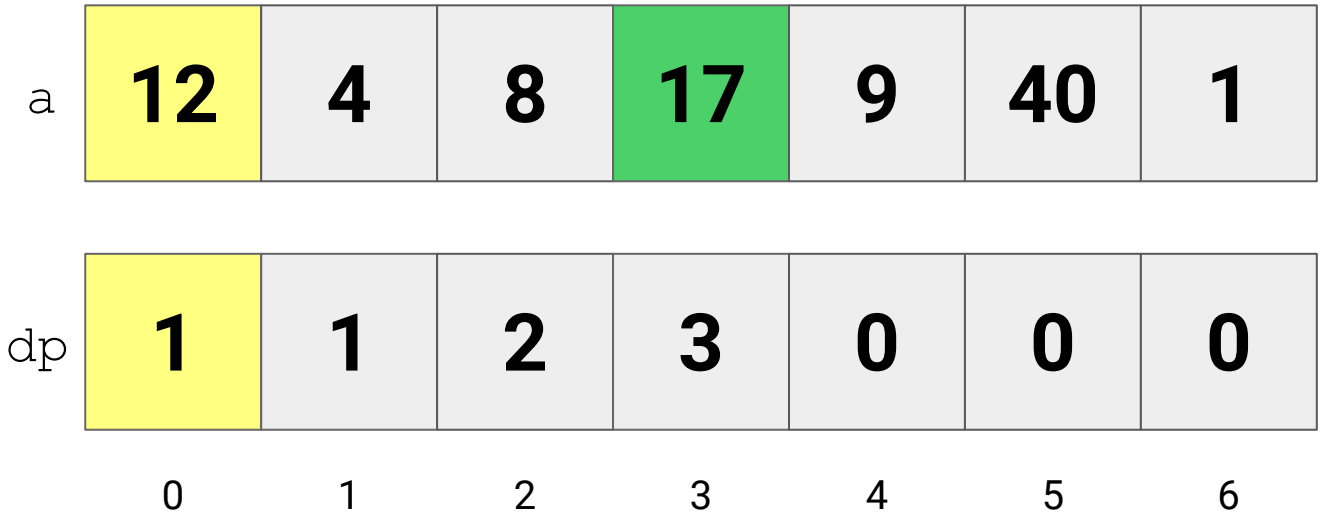
Наибольшая возрастающая подпоследовательность

Заполнение динамики



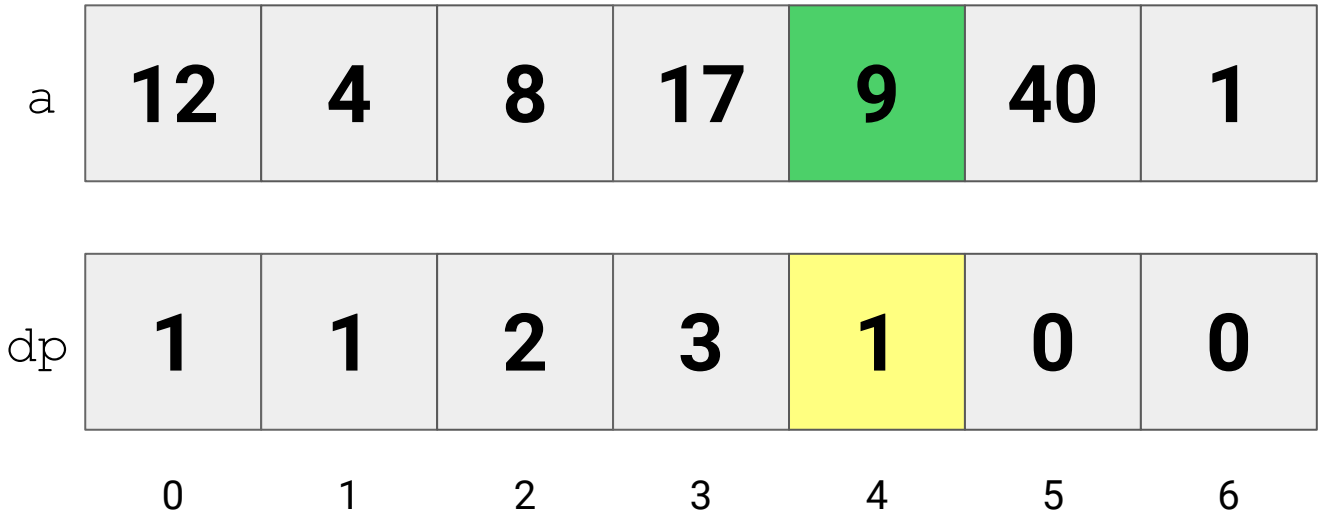
Наибольшая возрастающая подпоследовательность

Заполнение динамики



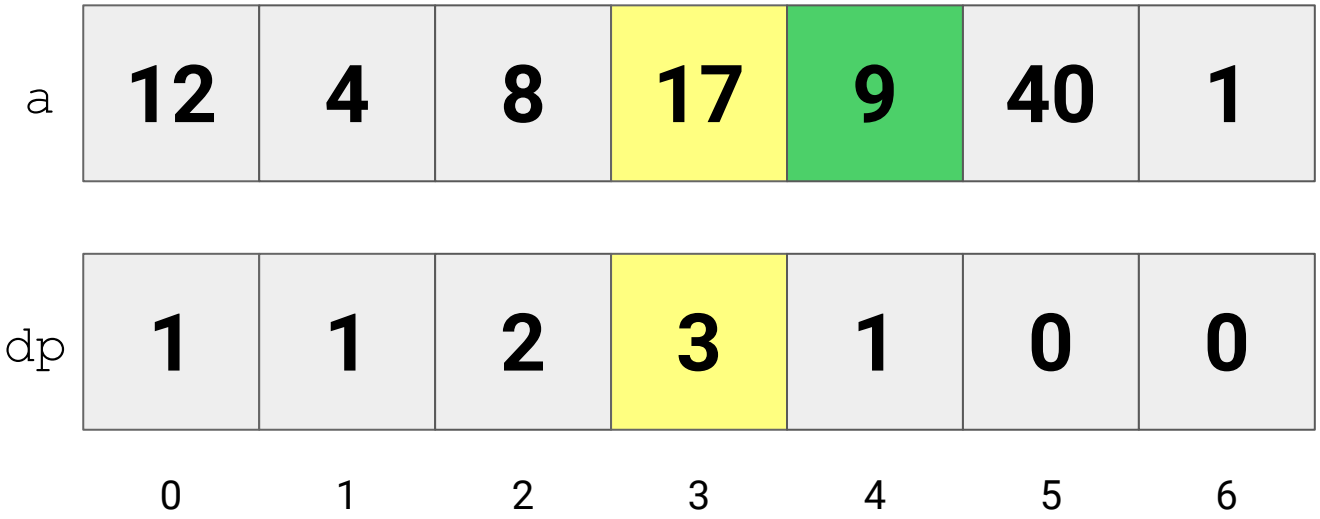
Наибольшая возрастающая подпоследовательность

Заполнение динамики



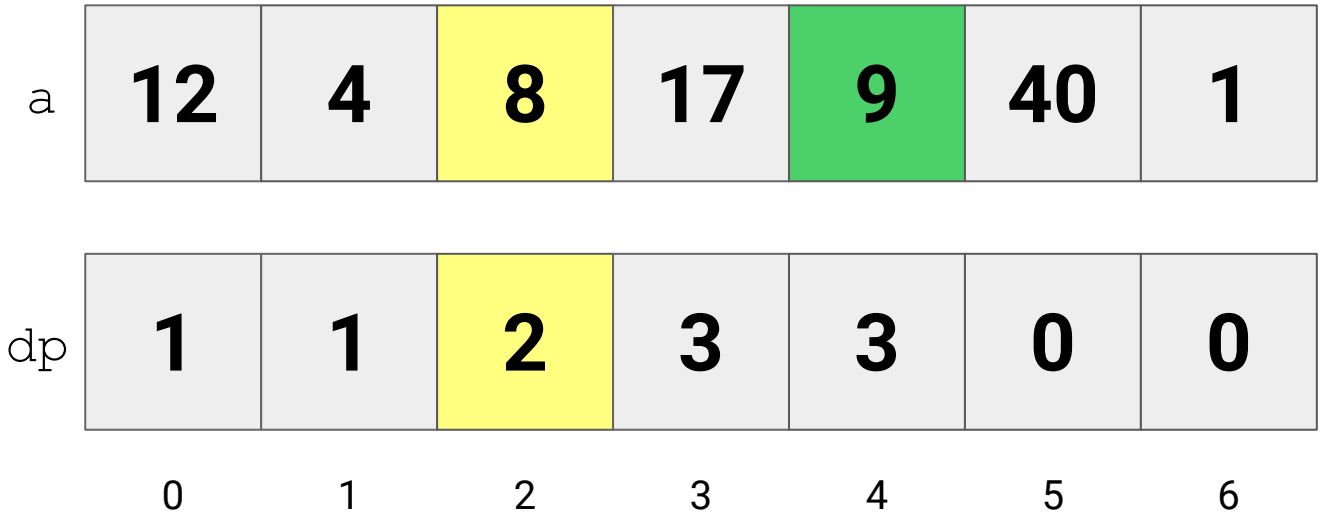
Наибольшая возрастающая подпоследовательность

Заполнение динамики



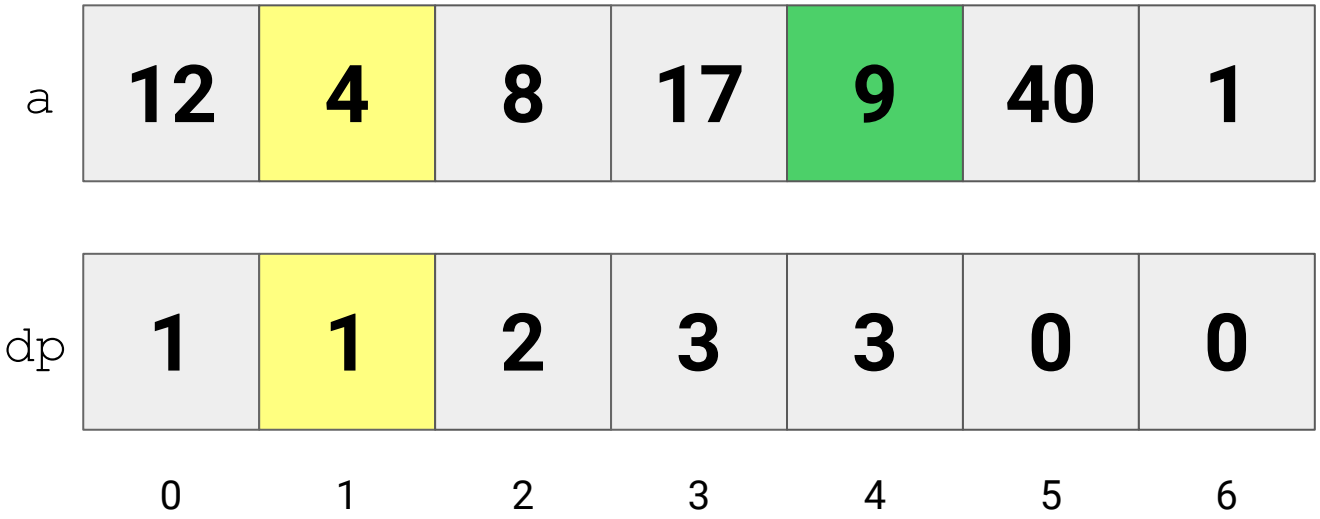
Наибольшая возрастающая подпоследовательность

Заполнение динамики



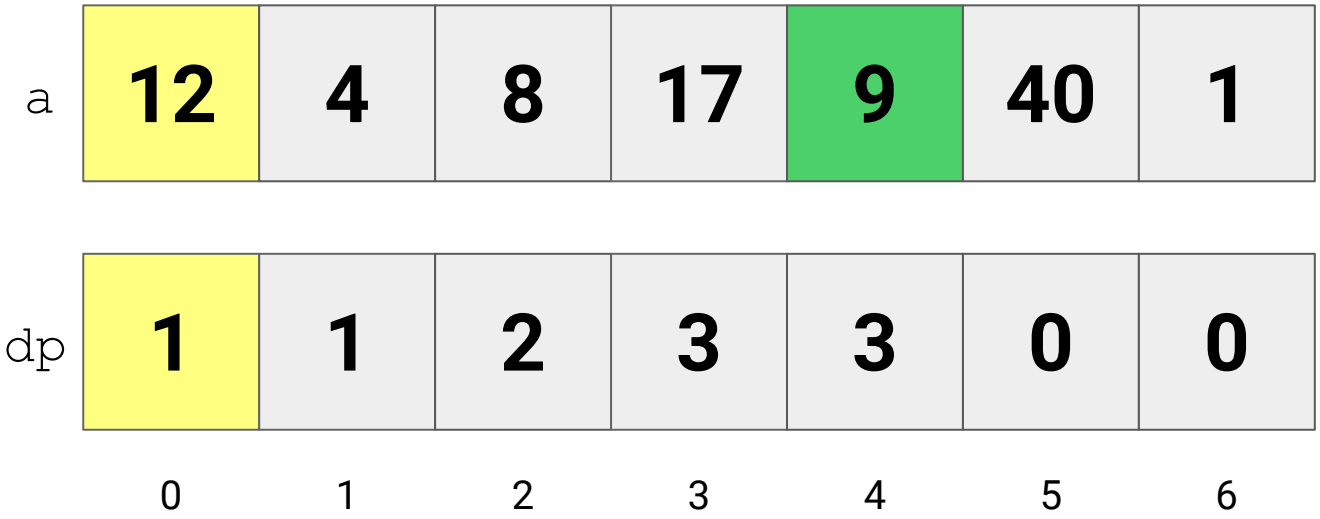
Наибольшая возрастающая подпоследовательность

Заполнение динамики



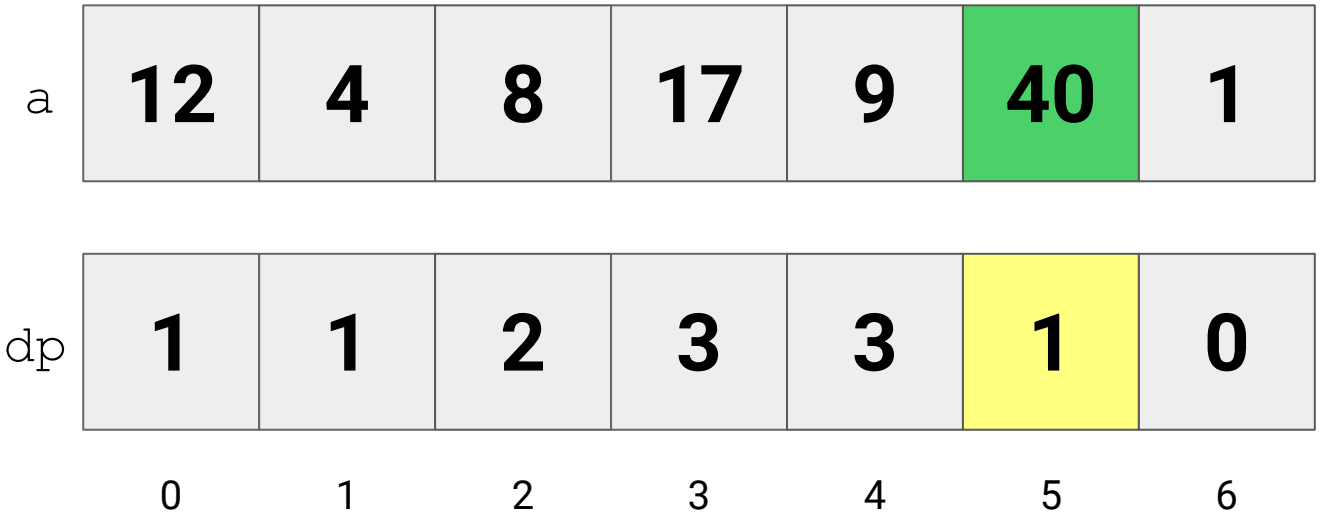
Наибольшая возрастающая подпоследовательность

Заполнение динамики



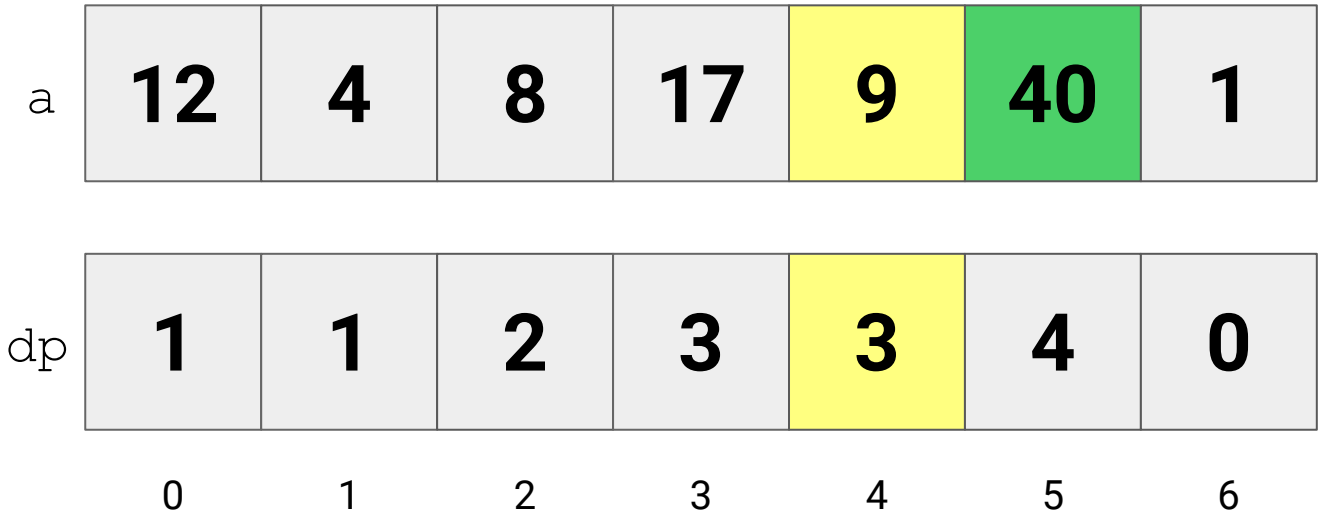
Наибольшая возрастающая подпоследовательность

Заполнение динамики



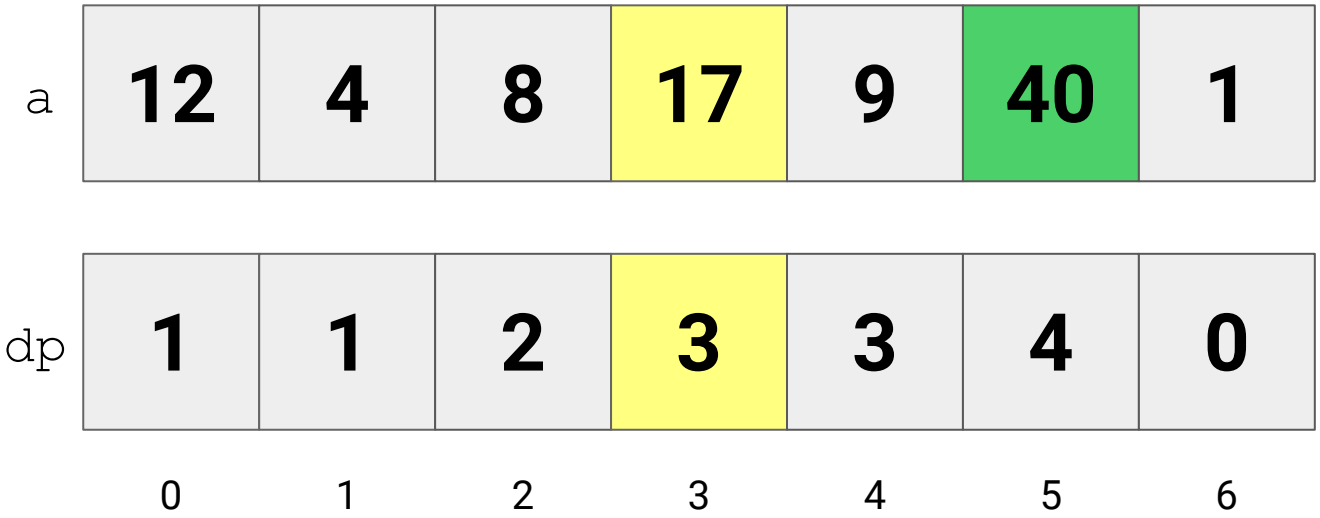
Наибольшая возрастающая подпоследовательность

Заполнение динамики



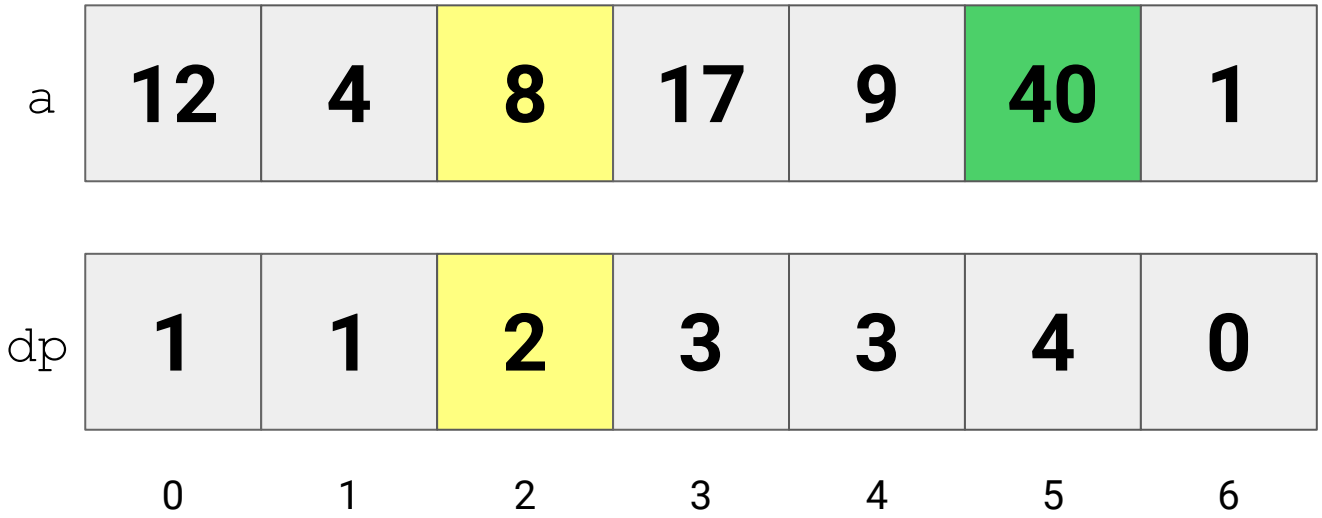
Наибольшая возрастающая подпоследовательность

Заполнение динамики



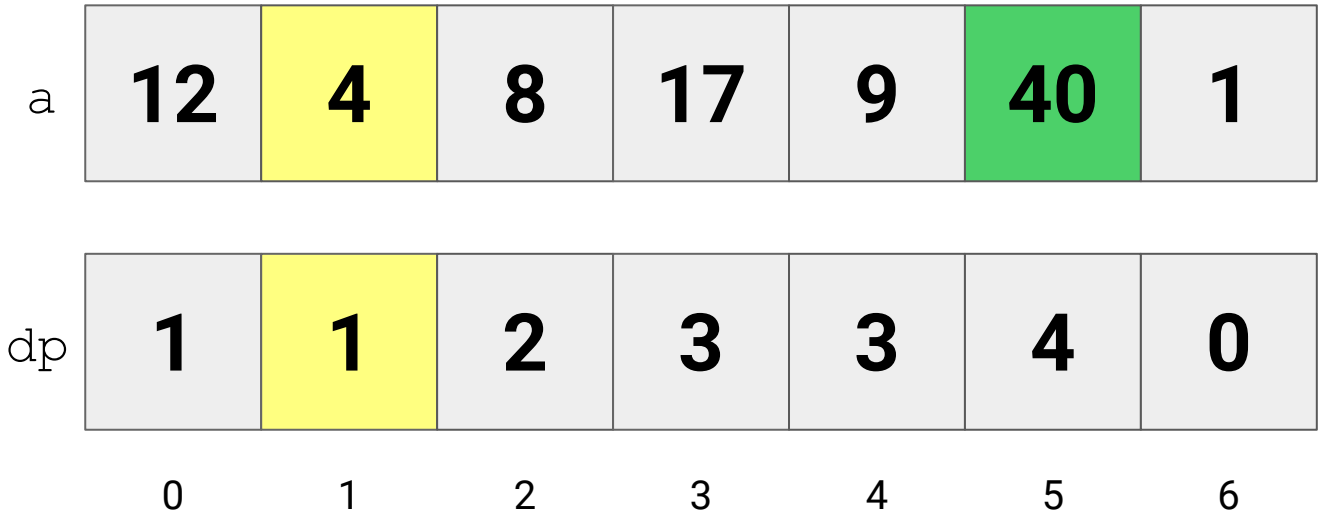
Наибольшая возрастающая подпоследовательность

Заполнение динамики



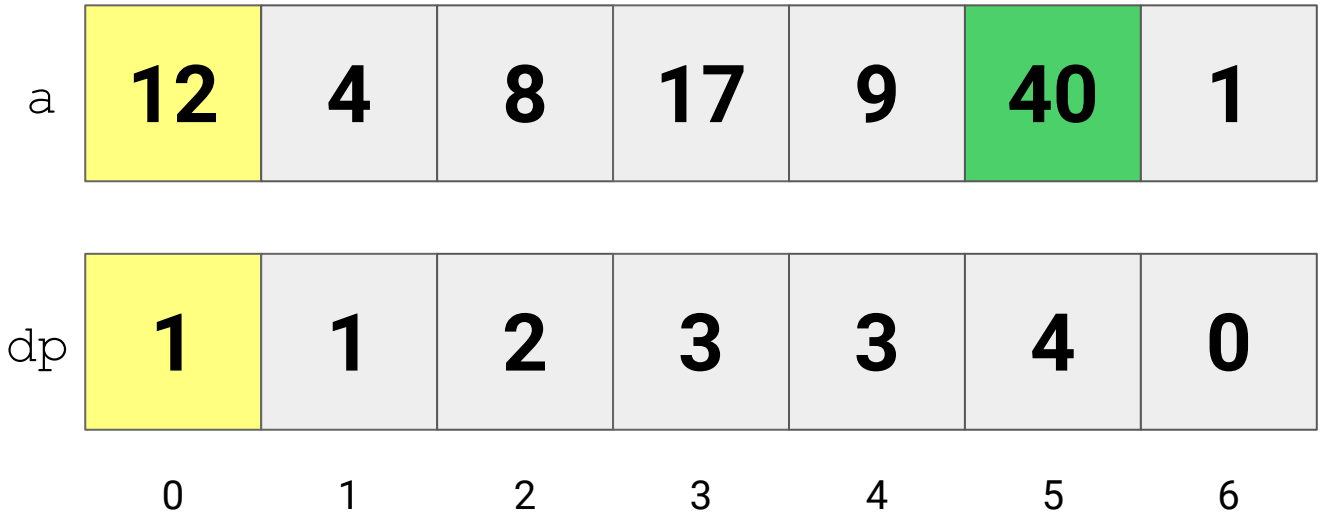
Наибольшая возрастающая подпоследовательность

Заполнение динамики



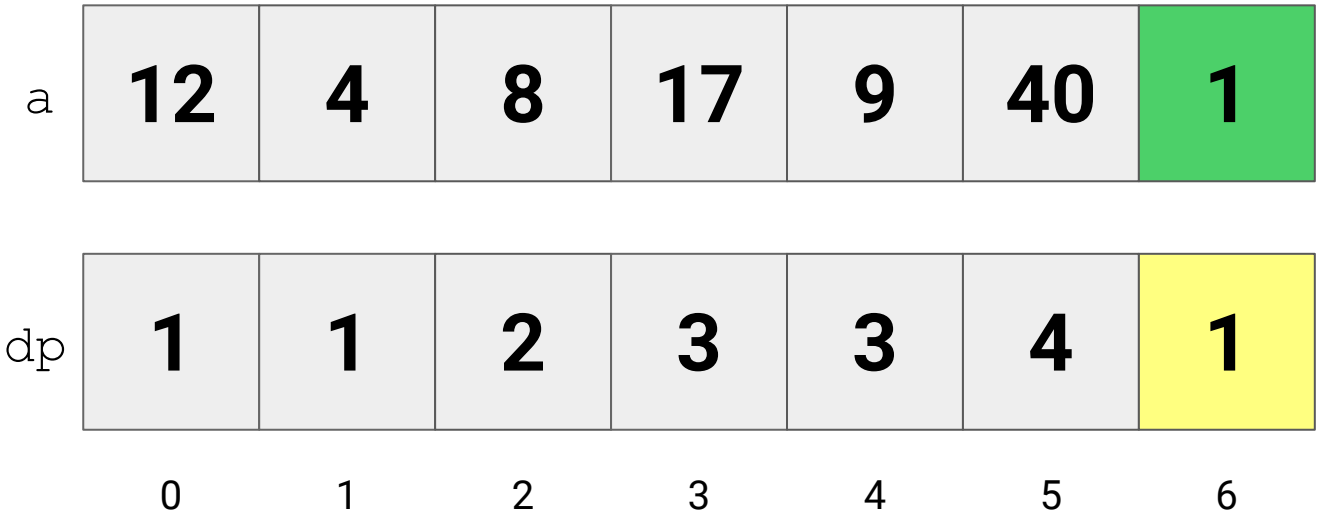
Наибольшая возрастающая подпоследовательность

Заполнение динамики



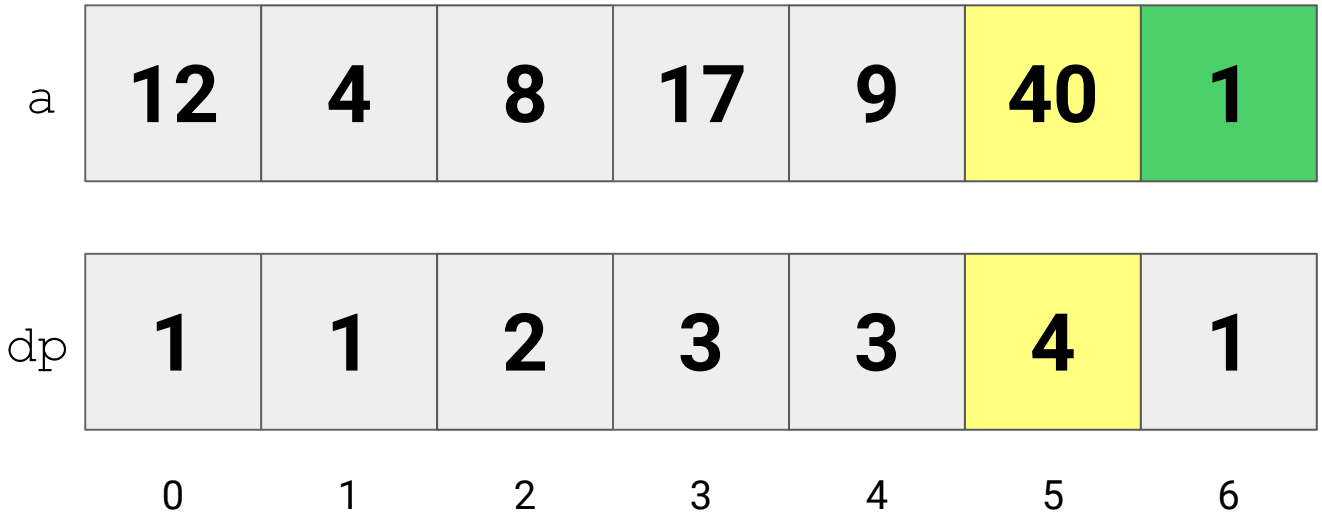
Наибольшая возрастающая подпоследовательность

Заполнение динамики



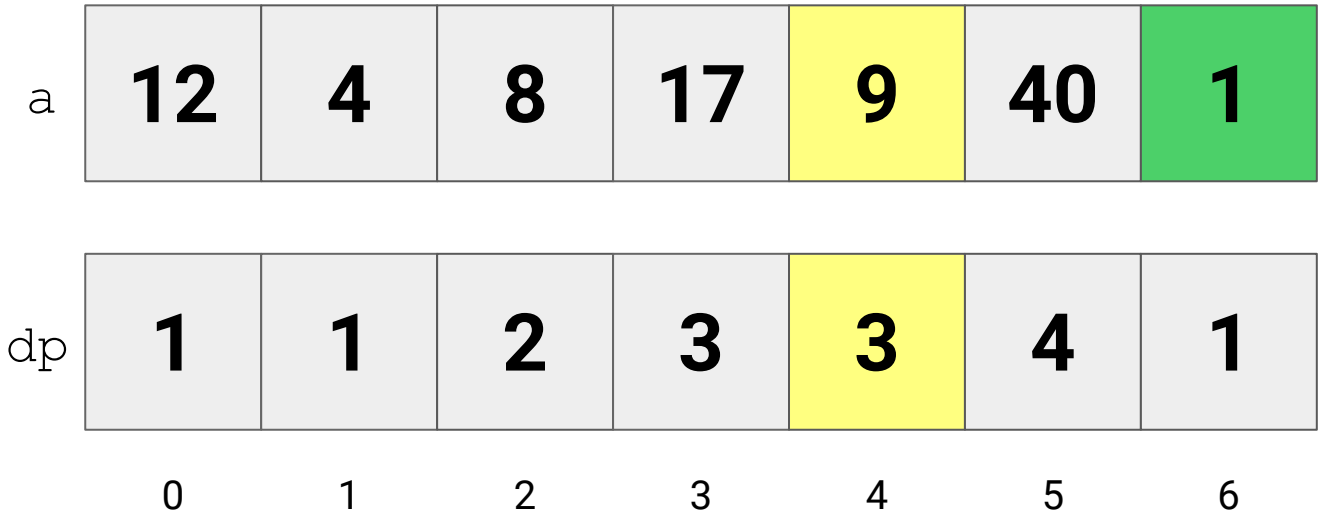
Наибольшая возрастающая подпоследовательность

Заполнение динамики



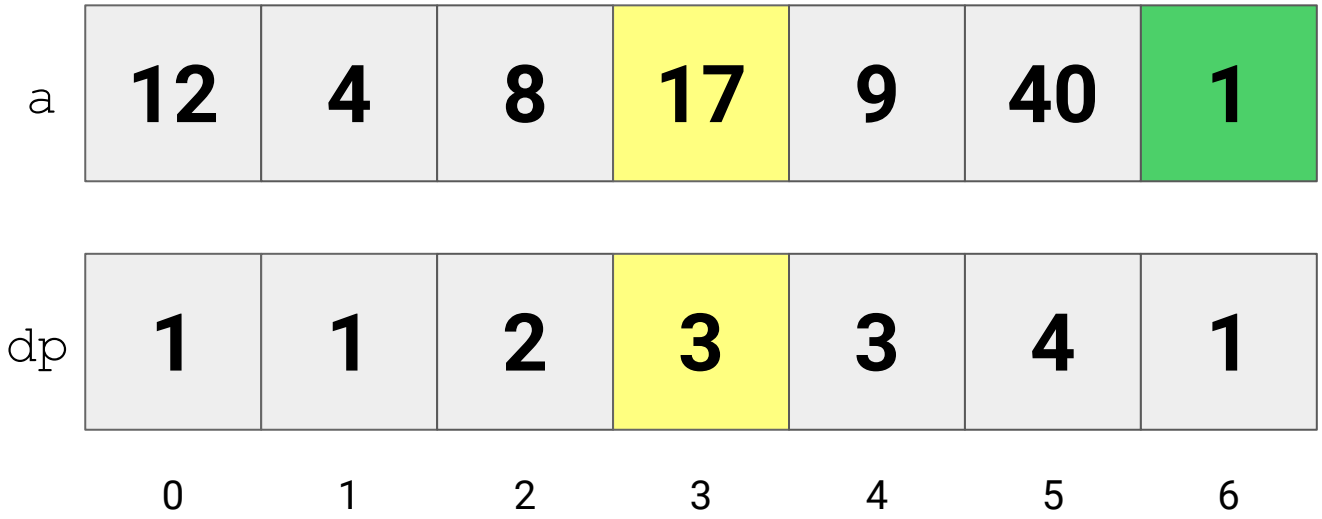
Наибольшая возрастающая подпоследовательность

Заполнение динамики



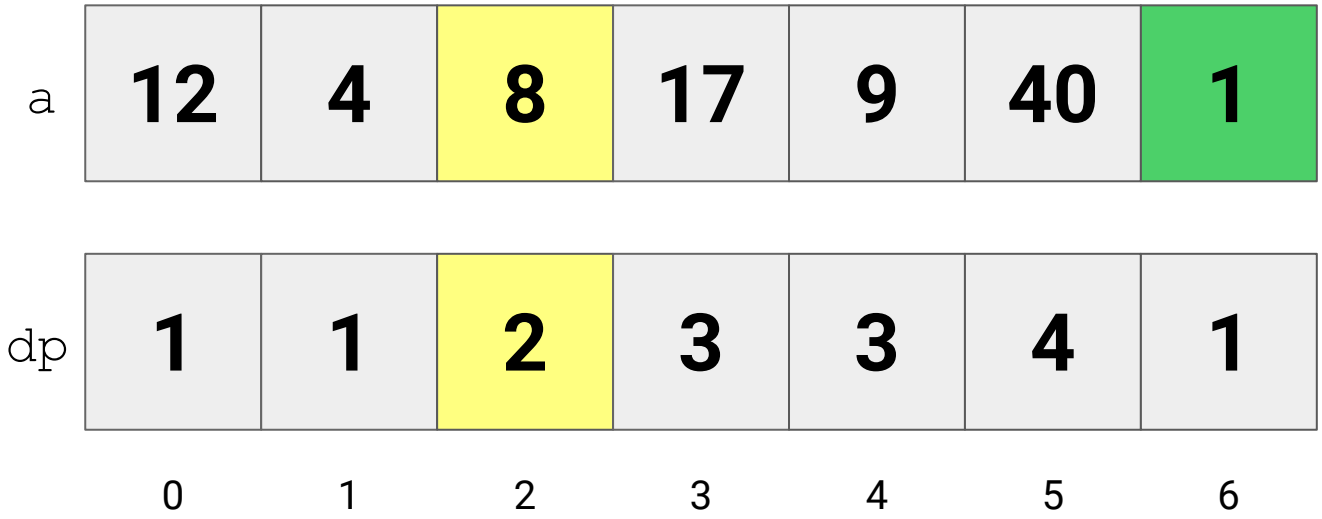
Наибольшая возрастающая подпоследовательность

Заполнение динамики



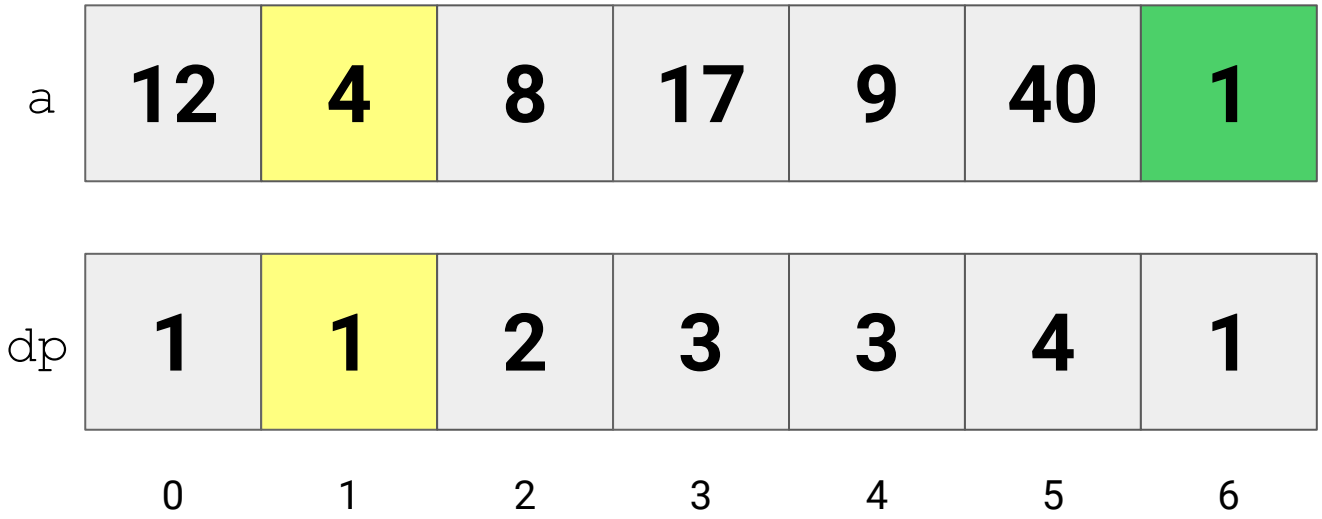
Наибольшая возрастающая подпоследовательность

Заполнение динамики



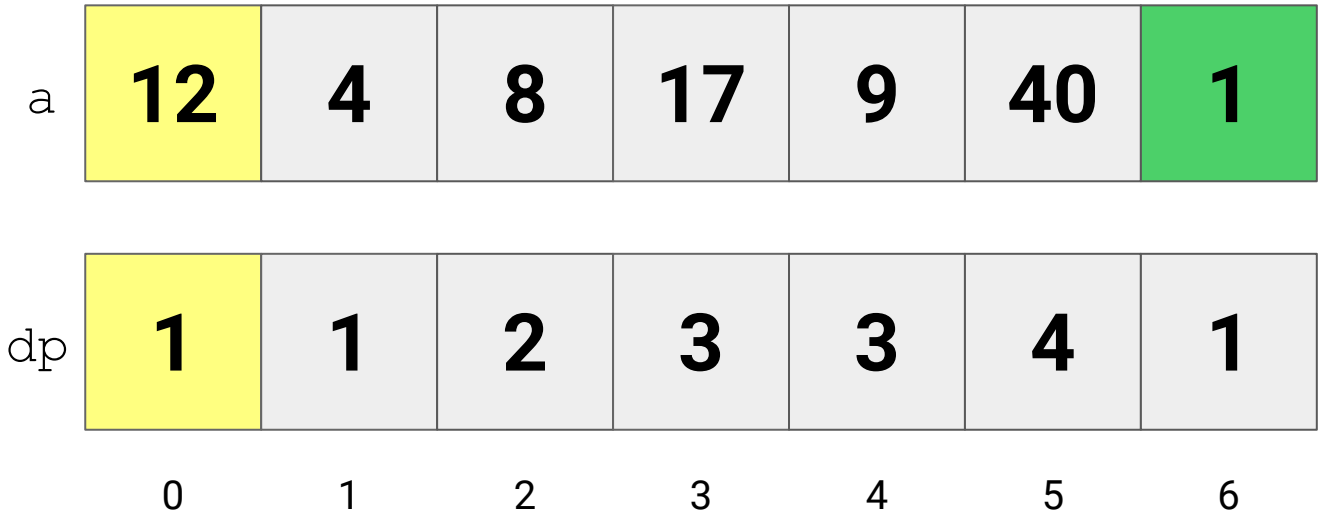
Наибольшая возрастающая подпоследовательность

Заполнение динамики



Наибольшая возрастающая подпоследовательность

Заполнение динамики



Наибольшая возрастающая подпоследовательность

Решение с помощью ДП

1) Рассматриваем элементы слева направо
2) Пусть текущий элемент `a[i]`
3) Если итоговая последовательность имеет длину `1`, то `dp[i] = 1`
4) Если больше `1`, то перед `a[i]` существует такой элемент `a[j]`, что `j < i`
5) Так как мы уже знаем ответы для всех `a[j]`, то давайте найдем найдем среди всех `a[j] < a[i]`, тот который имеет наибольшую подпоследовательность и добавим к ней элемент `a[i]`
6) Длина получившейся последовательности будет равняться `dp[j] + 1`

Наибольшая возрастающая подпоследовательность

```
int longest_increase_sequence(const vector<int>& a) {
    vector<int> dp(a.size(), 0);
    int answer = 0;

    for (size_t i = 0; i < a.size(); ++i) {
        dp[i] = 1;

        for (size_t j = 0; j < i; ++j) {
            if (a[i] > a[j]) {
                dp[i] = max(dp[i], dp[j] + 1);
            }
        }

        answer = max(answer, dp[i]);
    }

    return answer;
}
```

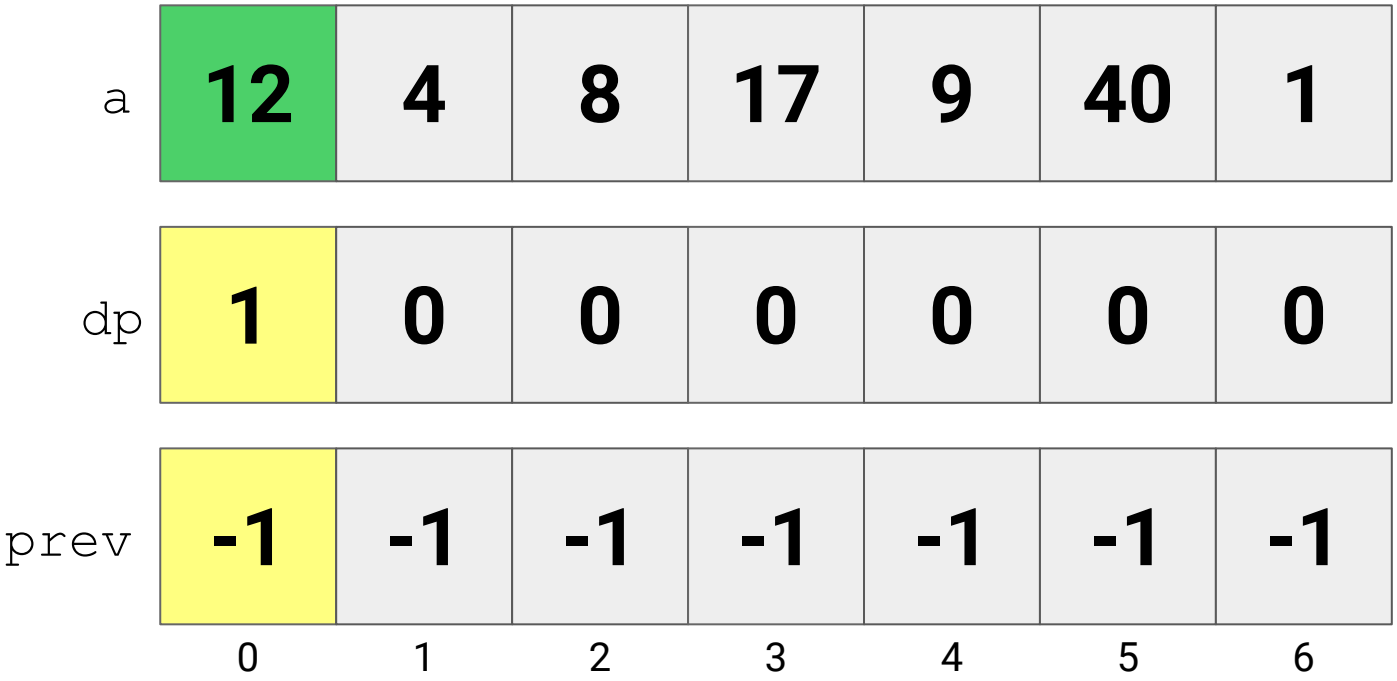
Наибольшая возрастающая подпоследовательность

Восстановление ответа

1) Мы умеем находить длину последовательности, но не саму последовательность
2) Давайте научимся восстанавливать последовательность которая заканчивается элементом с индексом
3) Заведем массив предков `prev`, где `prev[i]` это индекс предыдущего элемента в результирующей последовательности или `-1`, если текущий элемент первый в последовательности
4) Будем “прыгать” по массиву `prev`, пока не наткнемся на `-1`

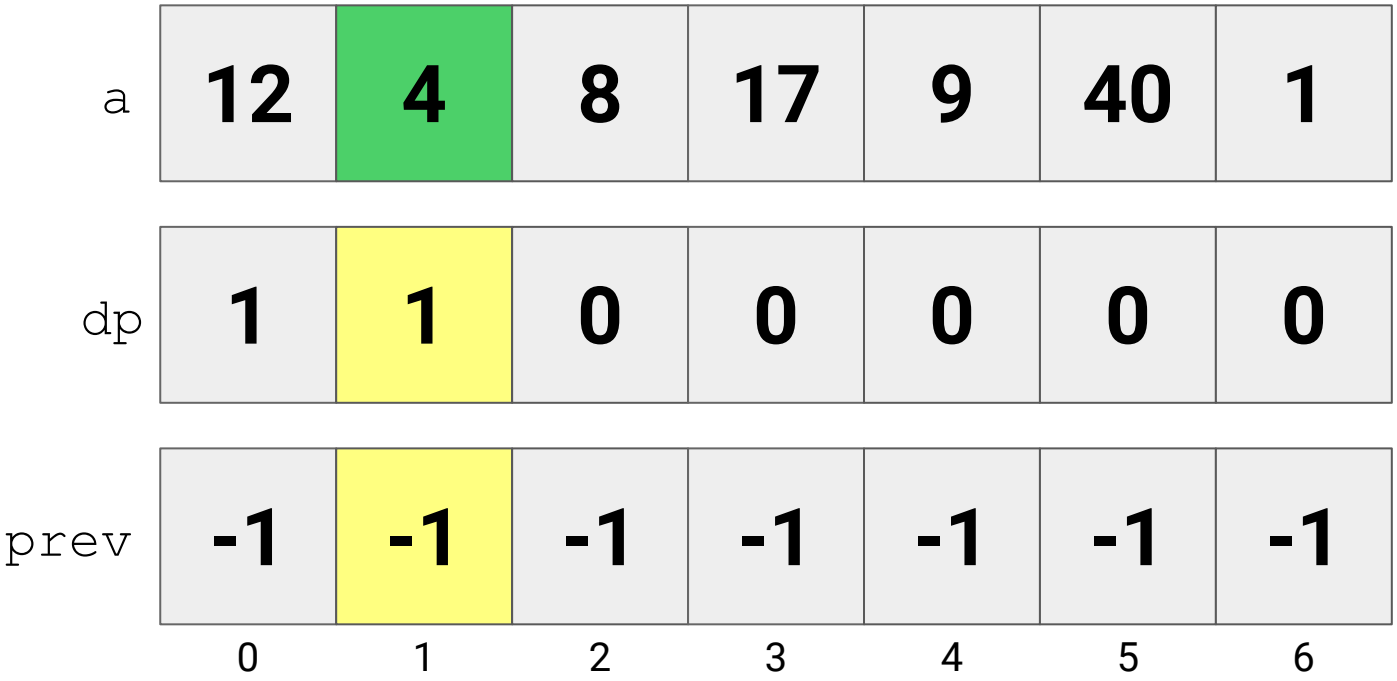
Наибольшая возрастающая подпоследовательность

Заполнение массива предков



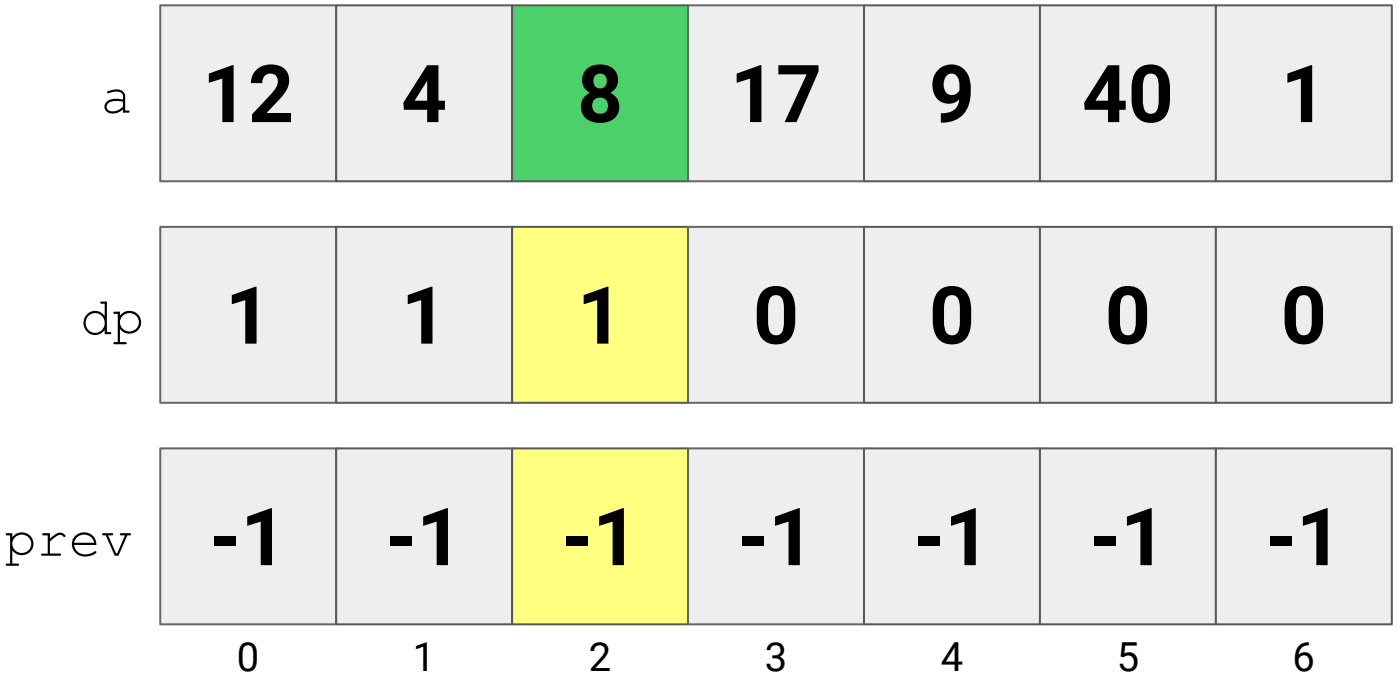
Наибольшая возрастающая подпоследовательность

Заполнение массива предков



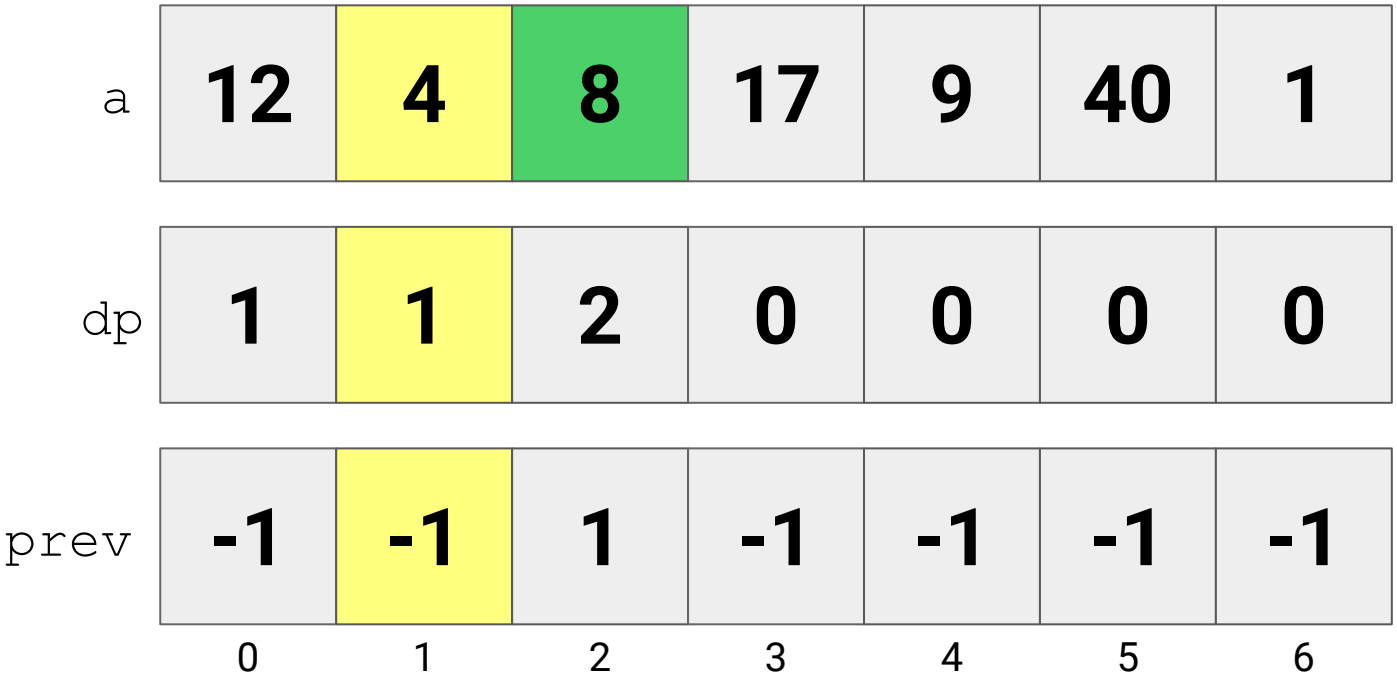
Наибольшая возрастающая подпоследовательность

Заполнение массива предков



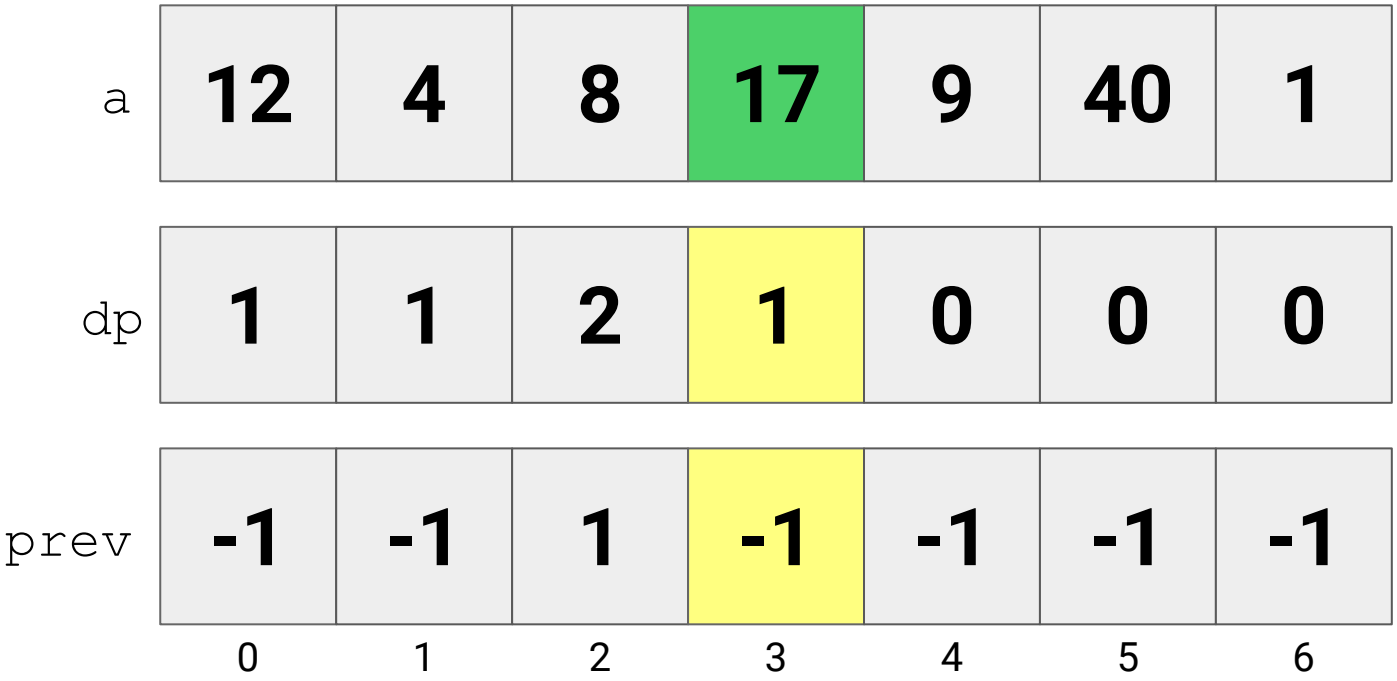
Наибольшая возрастающая подпоследовательность

Заполнение массива предков



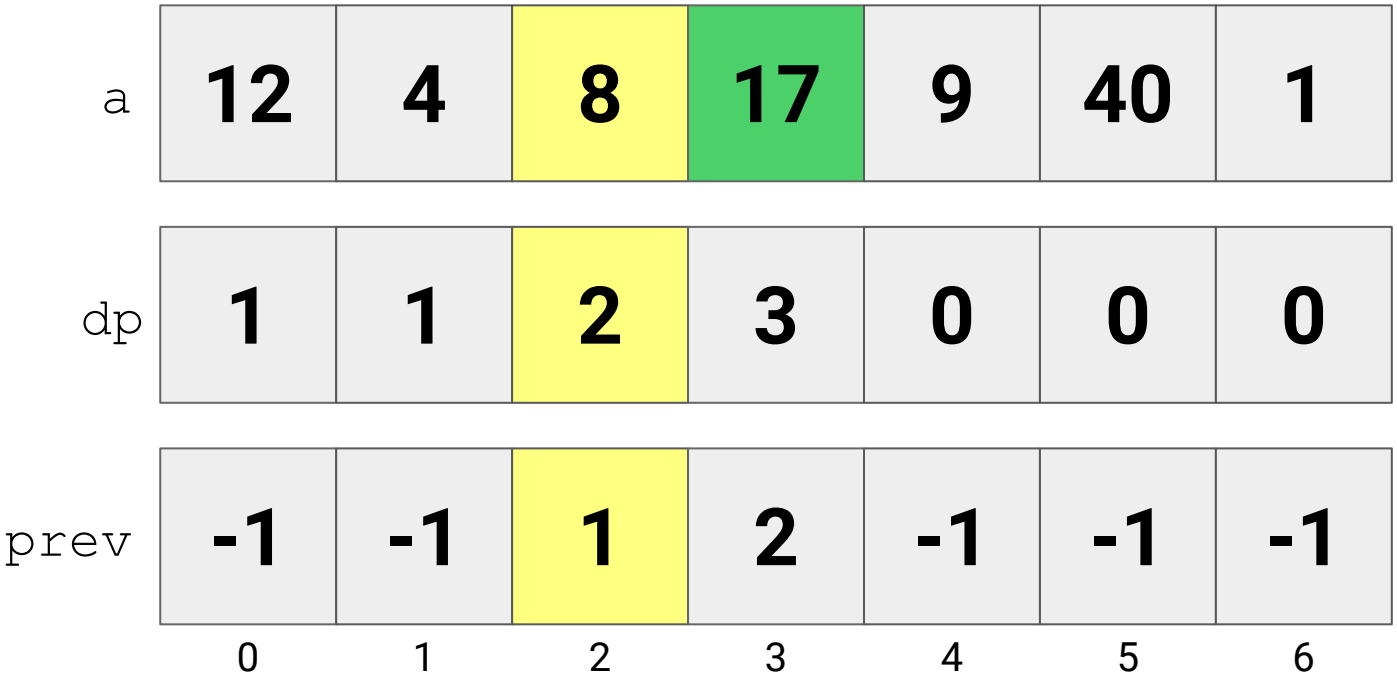
Наибольшая возрастающая подпоследовательность

Заполнение массива предков



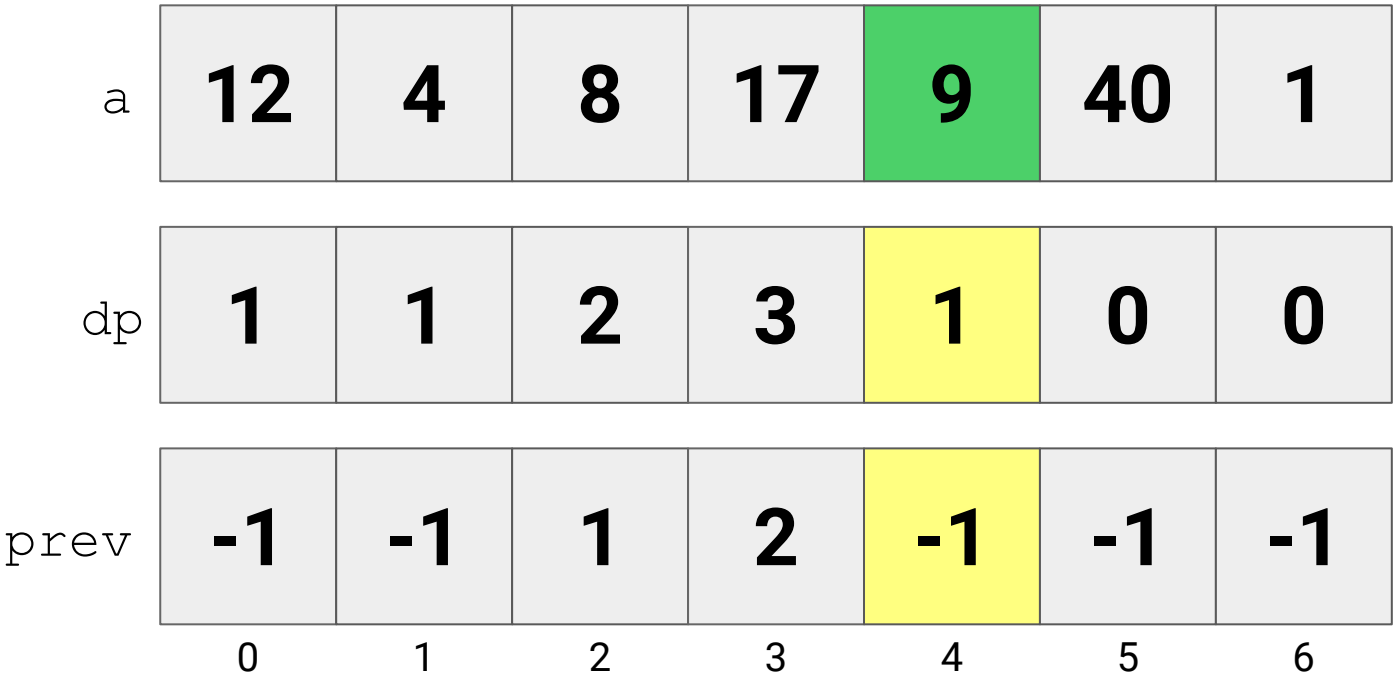
Наибольшая возрастающая подпоследовательность

Заполнение массива предков



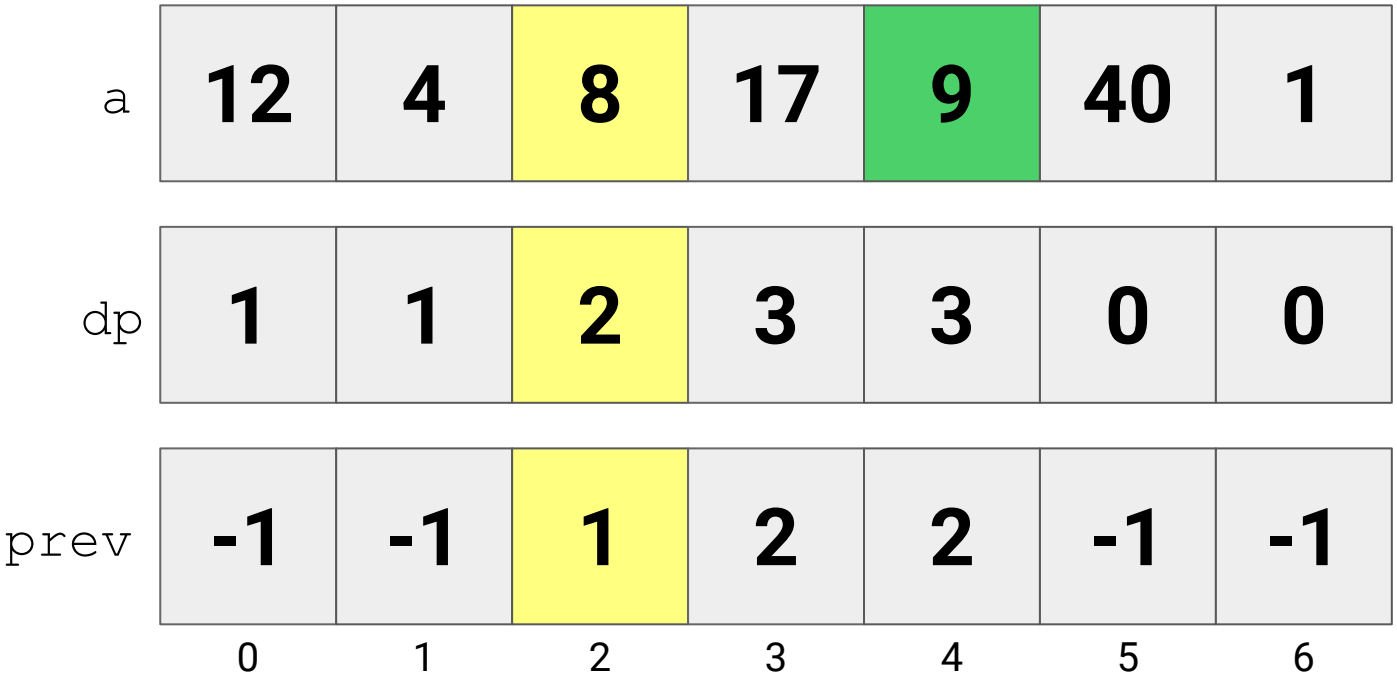
Наибольшая возрастающая подпоследовательность

Заполнение массива предков



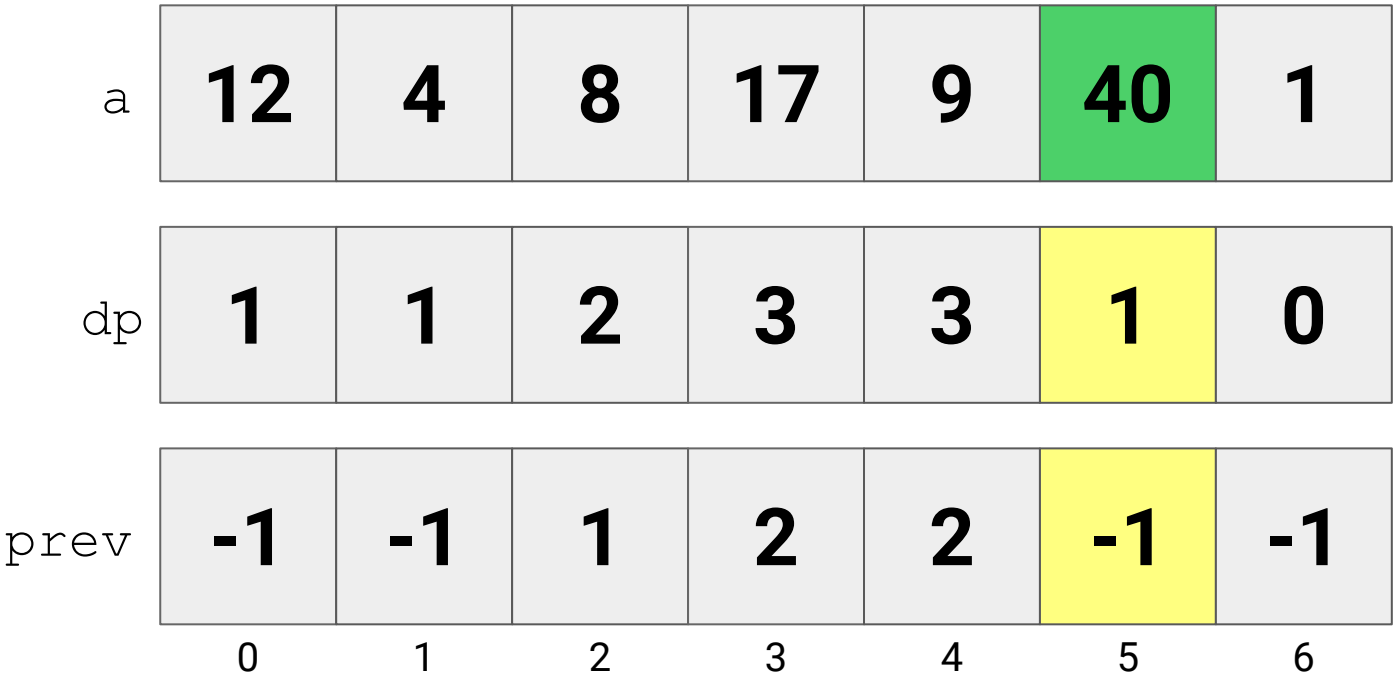
Наибольшая возрастающая подпоследовательность

Заполнение массива предков



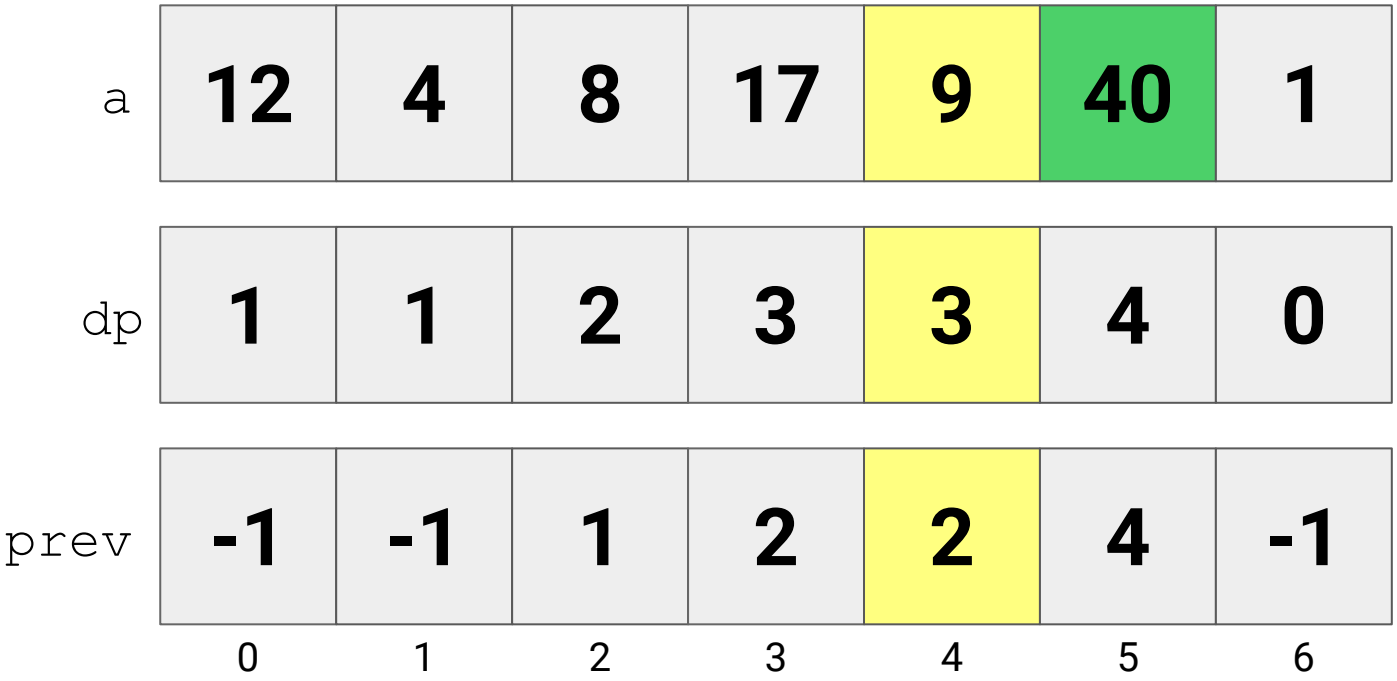
Наибольшая возрастающая подпоследовательность

Заполнение массива предков



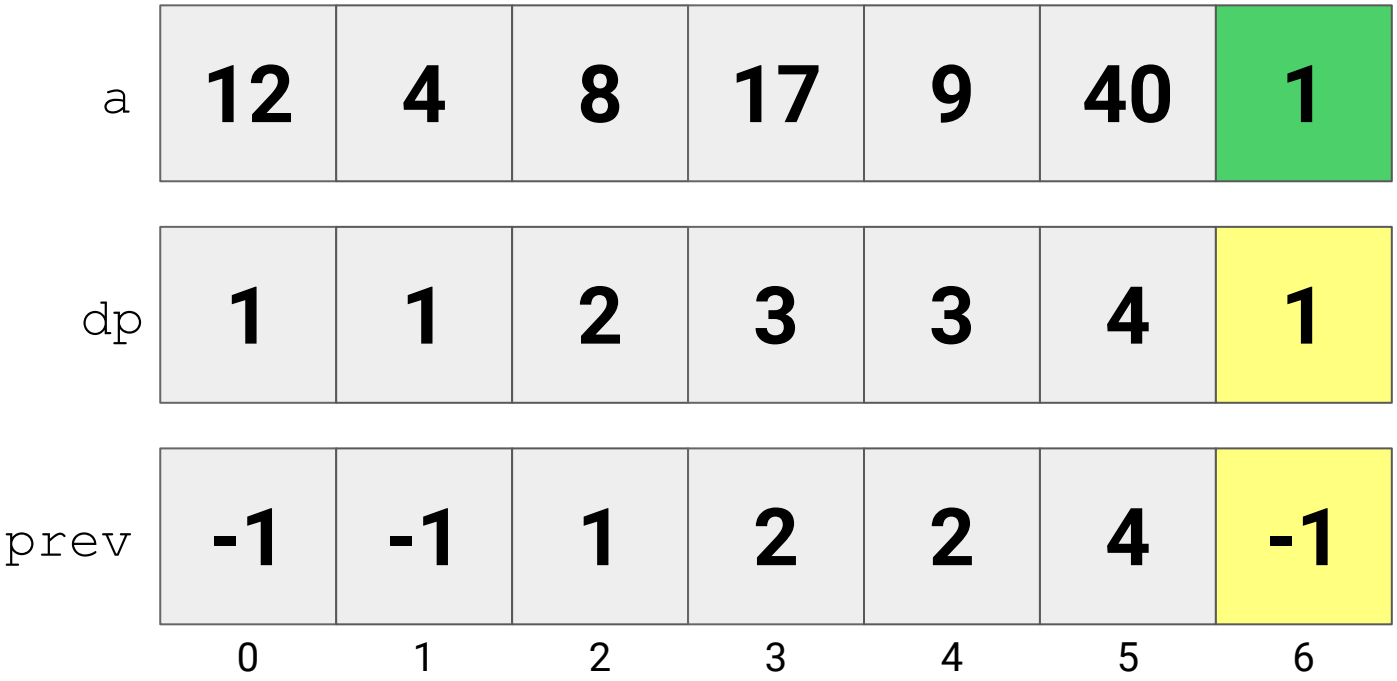
Наибольшая возрастающая подпоследовательность

Заполнение массива предков



Наибольшая возрастающая подпоследовательность

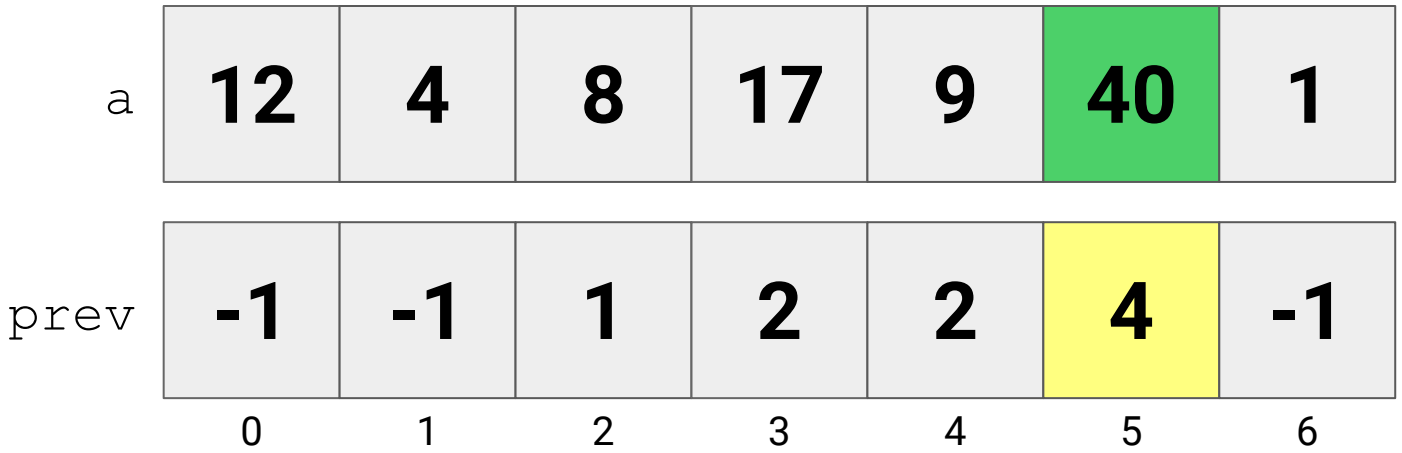
Заполнение массива предков



Наибольшая возрастающая подпоследовательность

Восстановление ответ

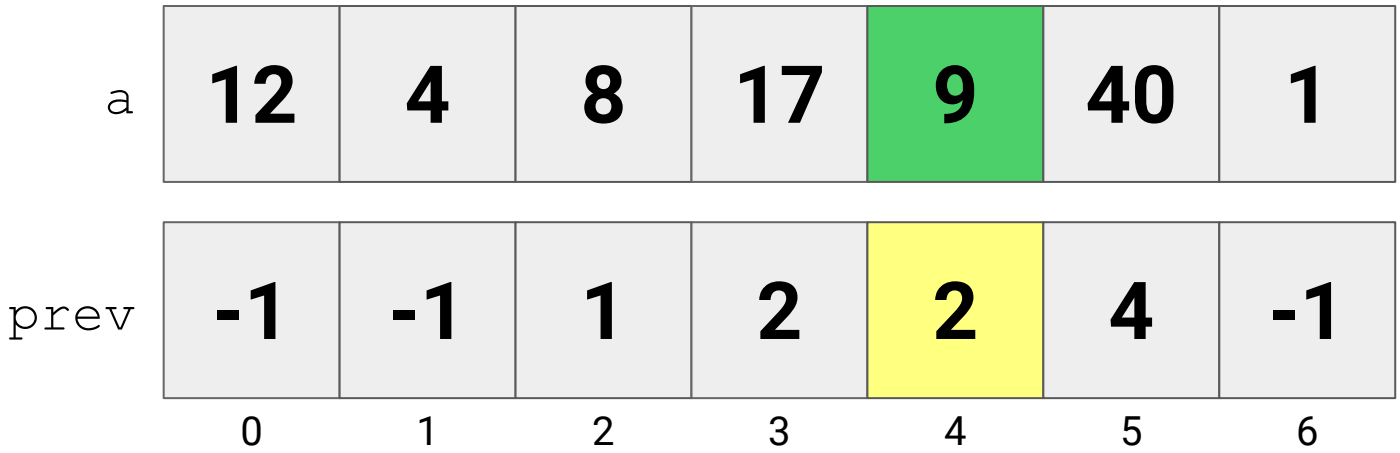
Ответ: 40



Наибольшая возрастающая подпоследовательность

Восстановление ответ

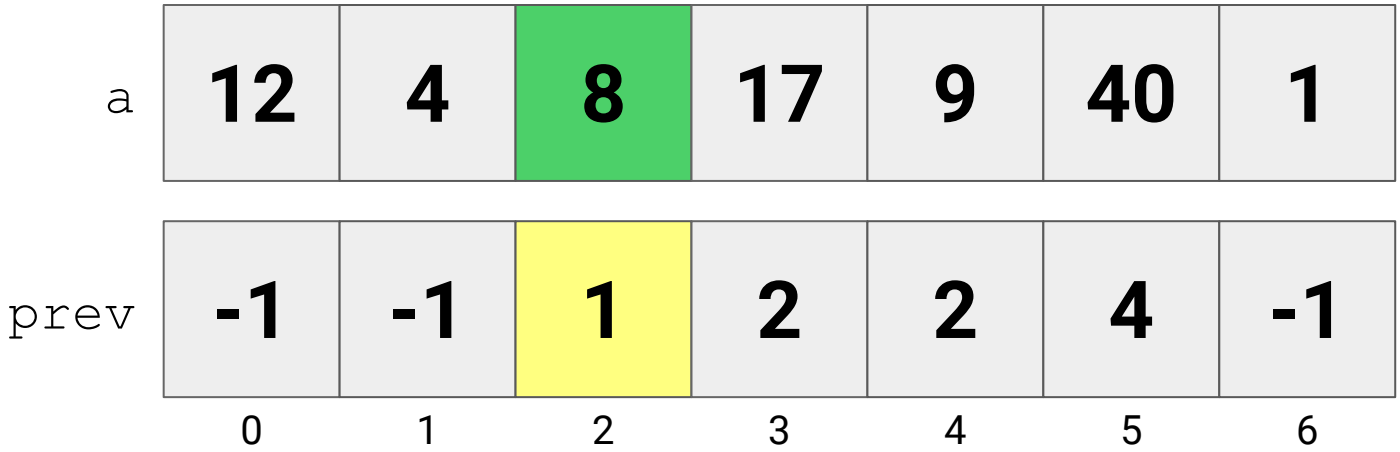
Ответ: 40 9



Наибольшая возрастающая подпоследовательность

Восстановление ответ

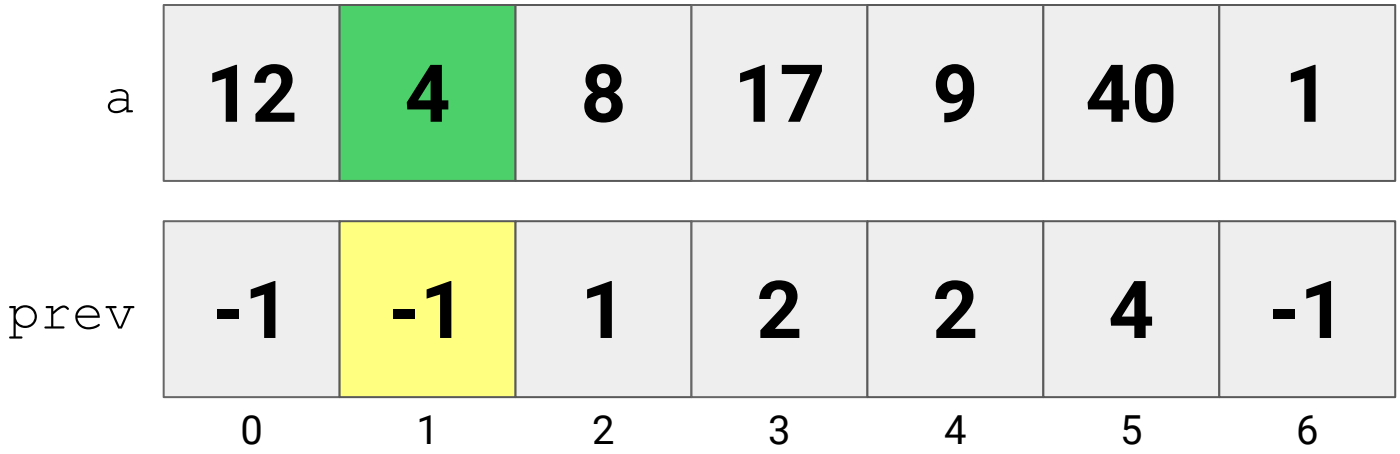
Ответ: 40 9 8



Наибольшая возрастающая подпоследовательность

Восстановление ответ

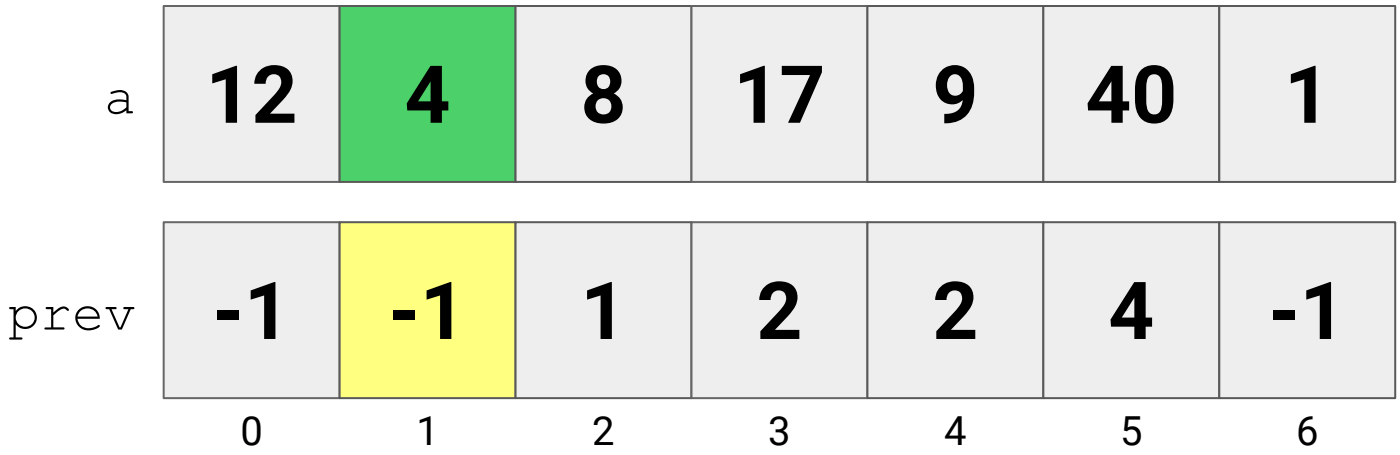
Ответ: 40 9 8 4



Наибольшая возрастающая подпоследовательность

Восстановление ответ

Ответ: 40 9 8 4 (4 8 9 40)



Наибольшая возрастающая подпоследовательность

```
vector<int> longest_increase_sequence(const vector<int>& a) {
    vector<int> dp(a.size(), 0);
    vector<int> prev(a.size(), -1);
    vector<int> answer;

    for (size_t i = 0; i < a.size(); ++i) {
        dp[i] = 1;

        for (size_t j = 0; j < i; ++j) {
            if (a[i] > a[j]) {
                if (dp[j] + 1 > dp[i]) {
                    dp[i] = dp[j] + 1;
                    prev[i] = j;
                }
            }
        }
    }
...
```

Наибольшая возрастающая подпоследовательность

```
vector<int> longest_increase_sequence(const vector<int>& a) {
    vector<int> dp(a.size(), 0);
    vector<int> prev(a.size(), -1);
    vector<int> answer;

    for (size_t i = 0; i < a.size(); ++i) {
        dp[i] = 1;

        for (size_t j = 0; j < i; ++j) {
            if (a[i] > a[j]) {
                if (dp[j] + 1 > dp[i]) {
                    dp[i] = dp[j] + 1;
                    prev[i] = j;
                }
            }
        }
    }
...
```

Наибольшая возрастающая подпоследовательность

```
...
    int last_idx = 0;
    for (size_t i = 1; i < a.size(); ++i) {
        if (dp[i] > dp[last_idx]) {
            last_idx = i;
        }
    }

    do {
        answer.push_back(a[last_idx]);
        last_idx = prev[last_idx];
    } while (last_idx != -1);

    reverse(answer.begin(), answer.end());

    return answer;
}
```

Наибольшая возрастающая подпоследовательность

Итоговая сложность

1) Заполнение массива dp - $O(n^2)$
2) Восстановление ответа - $O(n)$

Время вопросов и ответов

Наибольшая общая подпоследовательность

Наибольшая общая подпоследовательность

Определение

Общая подпоследовательность двух последовательностей X и Y - это такая последовательность Z, которая одновременно является подпоследовательностью X и Y

Например для последовательностей:

X = ABBCBACCBAC

Y = BCABABBCA

некоторая общая подпоследовательность:

Z = BCBACA

Наибольшая общая подпоследовательность

Наивное решение

1) Найдем все подпоследовательности обеих последовательностей за $O(2^n)$ для каждой
2) Сравним их и найдем общие за $O(n)$
3) Выберем наибольшую
4) Итоговая сложность $O(n * 2^{2n})$

Наибольшая общая подпоследовательность

Префикс на отрезке

Введем понятия `i-го` **префикса последовательности** `X` **размера** `n`**,**
$X_i = (x_0, x_2, \ldots, x_{i-1})$**, где** $0 <= i <= n$

Например:

$X = (A, B, C, B, D, A)$**, тогда**

$X_4 = (A, B, C, B)$

$X_0 = ()$

Наибольшая общая подпоследовательность

Теорема

Пусть имеются последовательности $X = (x_1, x_2, \ldots, x_m)$ **и** $Y = (y_1, y_2, \ldots, y_n)$, **а** $Z = (z_1, z_2, \ldots, z_k)$ **— их** LCS^*

Если $x_m = y_n$, **то** $z_k = x_m = y_n$ **и** Z_{k-1} **—** $LCS(X_{m-1}, Y_{n-1})$
Если $x_m \neq y_n$, **то из** $z_k \neq x_m$ **следует, что** Z **—** $LCS(X_{m-1}, Y)$
Если $x_m \neq y_n$, **то из** $z_k \neq y_n$ **следует, что** Z **—** $LCS(X, Y_{n-1})$

LCS* - Longest Common Subsequence

Наибольшая общая подпоследовательность

Рекуррентная формула

$$
\begin{aligned}
&LCS(i, j) = \\
&\quad 0, \; i == 0 \text{ or } j == 0 \\
&\quad LCS(i - 1, j - 1), \; x_i == y_j \\
&\quad \max(LCS(i - 1, j), LCS(i, j - 1)), \quad x_i \; != y_j
\end{aligned}
$$

Наибольшая общая подпоследовательность

Иллюстрация

X = (A, B, C, B, D, A, B)
Y = (B, D, C, A, B, A)

	j	0	1	2	3	4	5	6
i		y_j	B	D	C	A	B	A
0	x_i	0	0	0	0	0	0	0
1	A	0	0	0	0	1	1	1
2	B	0	1	1	1	1	2	2
3	C	0	1	1	2	2	2	2
4	B	0	1	1	2	2	3	3
5	D	0	1	2	2	2	3	3
6	A	0	1	2	2	3	3	4
7	B	0	1	2	2	3	4	4

Наибольшая общая подпоследовательность

Иллюстрация

X = (A, B, C, B, D, A, B)
Y = (B, D, C, A, B, A)

	j	0	1	2	3	4	5	6
i		y_j	B	D	C	A	B	A
0	x_i	0	0	0	0	0	0	0
1	A	0	0	0	0	1	1	1
2	B	0	1	1	1	1	2	2
3	C	0	1	1	2	2	2	2
4	B	0	1	1	2	2	3	3
5	D	0	1	2	2	2	3	3
6	A	0	1	2	2	3	3	4
7	B	0	1	2	2	3	4	4

Наибольшая общая подпоследовательность

Иллюстрация

X = (A, B, C, B, D, A, B)
Y = (B, D, C, A, B, A)

	j	0	1	2	3	4	5	6
i		y_j	B	D	C	A	B	A
0	x_i	0	0	0	0	0	0	0
1	A	0	0	0	0	1	1	1
2	B	0	1	1	1	1	2	2
3	C	0	1	1	2	2	2	2
4	B	0	1	1	2	2	3	3
5	D	0	1	2	2	2	3	3
6	A	0	1	2	2	3	3	4
7	B	0	1	2	2	3	4	4

Наибольшая общая подпоследовательность

Иллюстрация

X = (A, B, C, B, D, A, B)
Y = (B, D, C, A, B, A)

	j	0	1	2	3	4	5	6
i		y_j	B	D	C	A	B	A
0	x_i	0	0	0	0	0	0	0
1	A	0	0	0	0	1	1	1
2	B	0	1	1	1	1	2	2
3	C	0	1	1	2	2	2	2
4	B	0	1	1	2	2	3	3
5	D	0	1	2	2	2	3	3
6	A	0	1	2	2	3	3	4
7	B	0	1	2	2	3	4	4

Наибольшая общая подпоследовательность

Иллюстрация

X = (A, B, C, B, D, A, B)
Y = (B, D, C, A, B, A)

	j	0	1	2	3	4	5	6
i		y_j	B	D	C	A	B	A
0	x_i	0	0	0	0	0	0	0
1	A	0	0	0	0	1	1	1
2	B	0	1	1	1	1	2	2
3	C	0	1	1	2	2	2	2
4	B	0	1	1	2	2	3	3
5	D	0	1	2	2	2	3	3
6	A	0	1	2	2	3	3	4
7	B	0	1	2	2	3	4	4

Наибольшая общая подпоследовательность

Иллюстрация

X = (A, B, C, B, D, A, B)
Y = (B, D, C, A, B, A)

	j	0	1	2	3	4	5	6
i		y_j	B	D	C	A	B	A
0	x_i	0	0	0	0	0	0	0
1	A	0	0	0	0	1	1	1
2	B	0	1	1	1	1	2	2
3	C	0	1	1	2	2	2	2
4	B	0	1	1	2	2	3	3
5	D	0	1	2	2	2	3	3
6	A	0	1	2	2	3	3	4
7	B	0	1	2	2	3	4	4

Наибольшая общая подпоследовательность

Иллюстрация

X = (A, B, C, B, D, A, B)
Y = (B, D, C, A, B, A)

	j	0	1	2	3	4	5	6
i		y_j	B	D	C	A	B	A
0	x_i	0	0	0	0	0	0	0
1	A	0	0	0	0	1	1	1
2	B	0	1	1	1	1	2	2
3	C	0	1	1	2	2	2	2
4	B	0	1	1	2	2	3	3
5	D	0	1	2	2	2	3	3
6	A	0	1	2	2	3	3	4
7	B	0	1	2	2	3	4	4

Наибольшая общая подпоследовательность

Иллюстрация

X = (A, B, C, B, D, A, B)
Y = (B, D, C, A, B, A)

	j	0	1	2	3	4	5	6
i		y_j	B	D	C	A	B	A
0	x_i	0	0	0	0	0	0	0
1	A	0	0	0	0	1	1	1
2	B	0	1	1	1	1	2	2
3	C	0	1	1	2	2	2	2
4	B	0	1	1	2	2	3	3
5	D	0	1	2	2	2	3	3
6	A	0	1	2	2	3	3	4
7	B	0	1	2	2	3	4	4

Наибольшая общая подпоследовательность

Иллюстрация

X = (A, B, C, B, D, A, B)
Y = (B, D, C, A, B, A)

	j	0	1	2	3	4	5	6
i		y_j	B	D	C	A	B	A
0	x_i	0	0	0	0	0	0	0
1	A	0	0	0	0	1	1	1
2	B	0	1	1	1	1	2	2
3	C	0	1	1	2	2	2	2
4	B	0	1	1	2	2	3	3
5	D	0	1	2	2	2	3	3
6	A	0	1	2	2	3	3	4
7	B	0	1	2	2	3	4	4

Наибольшая общая подпоследовательность

Иллюстрация

X = (A, B, C, B, D, A, B)
Y = (B, D, C, A, B, A)

	j	0	1	2	3	4	5	6
i		y_j	B	D	C	A	B	A
0	x_i	0	0	0	0	0	0	0
1	A	0	0	0	0	1	1	1
2	B	0	1	1	1	1	2	2
3	C	0	1	1	2	2	2	2
4	B	0	1	1	2	2	3	3
5	D	0	1	2	2	2	3	3
6	A	0	1	2	2	3	3	4
7	B	0	1	2	2	3	4	4

Наибольшая общая подпоследовательность

Иллюстрация

X = (A, B, C, B, D, A, B)
Y = (B, D, C, A, B, A)

Ответ

Z = (B, C, B, A)

	j	0	1	2	3	4	5	6
i		y_j	B	D	C	A	B	A
0	x_i	0	0	0	0	0	0	0
1	A	0	0	0	0	1	1	1
2	B	0	1	1	1	1	2	2
3	C	0	1	1	2	2	2	2
4	B	0	1	1	2	2	3	3
5	D	0	1	2	2	2	3	3
6	A	0	1	2	2	3	3	4
7	B	0	1	2	2	3	4	4

Наибольшая общая подпоследовательность

```
vector<int> longest_common_subsequence(const vector<int>& a,
                                       const vector<int>& b) {
    vector<vector<int>> dp(a.size() + 1,vector<int>(b.size() + 1, 0));
    vector<pair<int, int>> prev(a.size() + 1, vector<pair<int, int>>(b.size() + 1);
    vector<int> answer;
...
```

Наибольшая общая подпоследовательность

```
vector<int> longest_common_subsequence(const vector<int>& a,
                                       const vector<int>& b) {
    vector<vector<int>> dp(a.size() + 1,vector<int>(b.size() + 1, 0));
    vector<pair<int, int>> prev(a.size() + 1, vector<pair<int, int>>(b.size() + 1);
    vector<int> answer;
...
```

Наибольшая общая подпоследовательность

```
...
    for (size_t i = 1; i <= a.size(); ++i) {
        for (size_t j = 1; j <= b.size(); ++j) {
            if (a[i] == b[j]) {
                dp[i][j] = dp[i - 1][j - 1] + 1, prev[i][j] = { i - 1, j - 1 };
            } else if (dp[i - 1][j] >= dp[i][j - 1]) {
                dp[i][j] = dp[i - 1][j], prev[i][j] = { i - 1, j };
            } else {
                dp[i][j] = dp[i][j - 1], prev[i][j] = { i, j - 1 };
            }
        }
    }
...
```

Наибольшая общая подпоследовательность

```
...
    for (size_t i = 1; i <= a.size(); ++i) {
        for (size_t j = 1; j <= b.size(); ++j) {
            if (a[i] == b[j]) {
                dp[i][j] = dp[i - 1][j - 1] + 1, prev[i][j] = { i - 1, j - 1 };
            } else if (dp[i - 1][j] >= dp[i][j - 1]) {
                dp[i][j] = dp[i - 1][j], prev[i][j] = { i - 1, j };
            } else {
                dp[i][j] = dp[i][j - 1], prev[i][j] = { i, j - 1 };
            }
        }
    }
...
```

Наибольшая общая подпоследовательность

```
...
    for (int i = a.size(), j = b.size(); i > 0 || j > 0;) {
        if (prev[i][j] == { i - 1, j - 1 }) {
            answer.push_back(a[i]);
        }
        i = prev[i][j].first, j = prev[i][j].second;
    }

    reverse(answer.begin(), answer.end());

    return answer;
}
```

Наибольшая общая подпоследовательность

Итоговая сложность алгоритма

1) Пусть `n` и `m` - размеры двух последовательностей
2) Заполнение таблицы - `O(nm)`
3) Восстановление ответа - `O(n + m)`

Время вопросов и ответов