БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

Методы разработки алгоритмов

Метод «разделяй и властвуй»

Динамическое программирование

Метод «разделяй и властвуй»

«Разделяй и властвуй»

1. ***«Разделение»***

Задача разбивается на **<u>независимые подзадачи</u>**, т.е. подзадачи не пересекаются (*две задачи назовем независимыми, если они не имеют общих подзадач*).

2. ***«Покорение»***

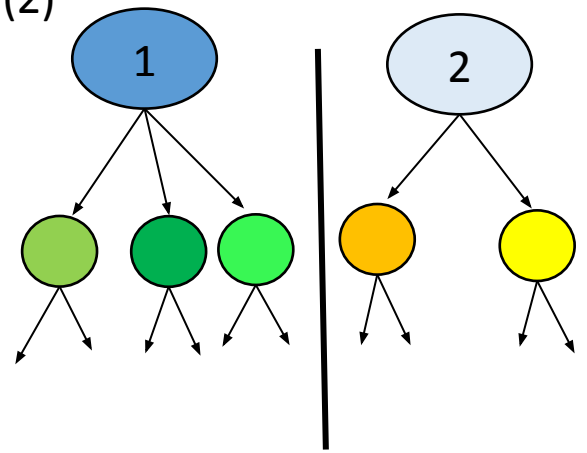
Каждая подзадача решается отдельно (рекурсивным методом).
Когда объем возникающих подзадач достаточно мал, то подзадачи решаются непосредственно.

3. ***«Комбинирование»***

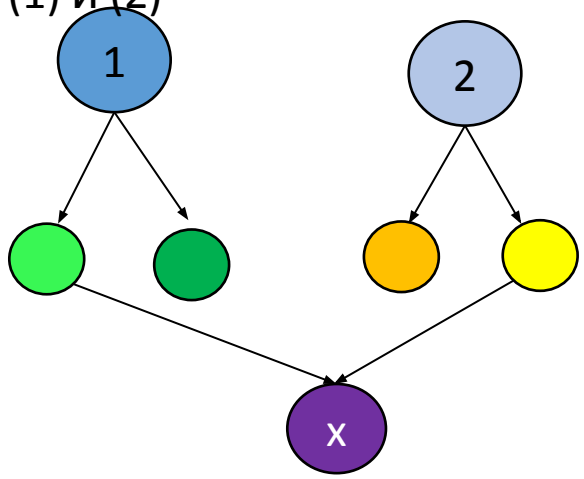
Из отдельных решений подзадач строится решение исходной задачи.

$$\begin{cases} T(n) = c_1, n \le c, \\ T(n) = D(n) + aT\left(\frac{n}{b}\right) + F(n), n > c, \end{cases}$$

независимые задачи (1) и (2)



зависимые задачи (1) и (2)



Принцип балансировки

При разбиении задачи на подзадачи полезен ***принцип балансировки***, который предполагает, что задача разбивается на подзадачи приблизительно равных размерностей, т. е. идет поддержание равновесия.

Обычно такая стратегия приводит к разделению исходной задачи пополам и обработке каждой из его частей тем же способом до тех пор, пока части не станут настолько малыми, что их можно будет обрабатывать непосредственно. Часто такой процесс приводит к логарифмическому множителю в формуле, описывающей трудоемкость алгоритма.

Таким образом, в основе техники рассматриваемого метода лежит процедура разделения. Если разделение удается произвести без слишком больших затрат, то может быть построен эффективный алгоритм.

Поиск максимального и минимального элементов

«Разделение» (балансировка) Разделим массив на две части (предположим, что $n=2^k$).

«Покорение» В каждой из частей этим же алгоритмом найдём локальные $\max_1$, $\min_1$, $\max_2$, $\min_2$.

«Комбинирование» Полагаем max=*наибольший* ($\max_1$, $\max_2$), min=*наименьший* ($\min_1$, $\min_2$).

Если в рассматриваемой области меньше 2-х элементов, то деление не выполняем, а за 1 сравнение определим максимальный и минимальный элемент области.

$$\begin{cases} T(n) = C + 2 \cdot T\left(\frac{n}{2}\right) + 2, n = 2^k, k \geq 2 \\ T(2) = 1 \end{cases}$$

Решение на основе принципа «разделяй и властвуй»

$$\frac{3}{2}n - 2$$

Последовательный поиск

$$2n - 3$$

Сортировка массива слиянием

«Разделение» (балансировка)

1. Делим последовательность элементов на две части (границы *l* и *r* включаем; если сортируемая последовательность состояла из *n* элементов, то первая часть может содержать первых элементов, а вторая часть – оставшиеся; порядок следования элементов в каждой из полученных частей совпадает с их порядком следования в исходной последовательности). Если в последовательности только один элемент, то деление не выполняем.

«Покорение»

2. Сортируем отдельно каждую из полученных частей этим же алгоритмом.

«Комбинирование»

3. Производим слияние отсортированных частей последовательности так, чтобы сохранилась упорядоченность.

```
def MergeSort (l,r):
  if l ≠ r:
          k = (l + r) // 2
          MergeSort (l,k)
          MergeSort (k+1,r)
          MergeList (l,k,r)
```

$$\begin{cases} C(n) = T_1 + 2 \cdot C\left(\frac{n}{2}\right) + n_2 \cdot , k = 2^k, \quad \geq 1 \\ C(1) = {}_3 \end{cases}$$

$$T\left(n\right) = \mathrm{O}\left(n \log n\right)$$

Быстрая сортировка массива Ч. Хоара

«Разделение»
(балансировка)

1. Выбираем в качестве сепаратора **x** медиану рассматриваемой области (за линейное от числа элементов время). Относительно сепаратора **x** делим массив на три части:
   1) в первой части - элементы, которые меньше или вравны **x**;
   2) во второй части - элемент **x**;
   3) в третьей части – элементы, которые больше или равны **x**.

«Покорение»

2. Сортируем отдельно I и III части этим же алгоритмом. Если в некоторой менее одного элемента, то ничего не делаем.

«Комбинирование»

3. Происходит слияние отсортированных сегментов в один путем присоединения сегментов.

```
def QuickSort(l, r):
        if l < r:
            Partition
            QuickSort(l, j)
            QuickSort(i, r)
```

$$\begin{cases} C(n) = T \cdot + 2n\left(\frac{n}{2}\right), k \geq 2^k, \geq 2 \\ C(1) = 2 \end{cases}$$

$$T(n) = O(n \cdot \log n)$$

Динамическое программирование

Динамическое программирование

Динамическое программирование применяется к задачам, в которых нужно что-то подсчитать или к оптимизационным задачам.

Например, в задаче требуется определить число различных способов подняться по ступенькам при заданном способе подъема, или вычислить число способов размещения k единиц в строке длины n, или вычислить F_n-ое число Фибоначчи и т.п.

Задачи оптимизации. В таких задачах существует много решений, каждому из которых поставлено в соответствие некоторое значение. Необходимо найти среди всех возможных решений одно с оптимальным (наибольшим или наименьшим) значением. Например, во взвешенном графе между заданной парой вершин существует несколько маршрутов, каждый маршрут характеризуется своей длиной, и нам необходимо найти маршрут кратчайшей длины.

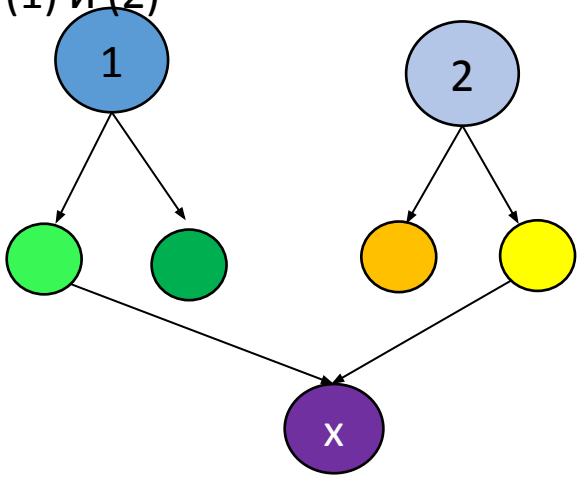
Стратегия метода динамического программирования – попытка свести рассматриваемую задачу к более простым задачам.

> *Задача может быть проще из-за того, что опущены некоторые ограничения. Она может быть проще из-за того, что некоторые ограничения добавлены. Однако, как бы ни была изменена задача, если это изменение приводит к решению более простой задачи, то, возможно, удастся, опираясь на эту более простую, решить и исходную.*

Так как возникающие подзадачи являются зависимыми, то данная техника находит своё применение, когда все нужные значения оптимальных решений подзадач помещаются в память. Вычисление идет от малых подзадач к большим, и ответы запоминаются в таблице, что позволяет исключить повторное решение задачи.

Метод динамического программирования часто в литературе называют **табличным**, а одна из клеток таблицы и даёт значение оптимального решения исходной задачи.

зависимые задачи
(1) и (2)



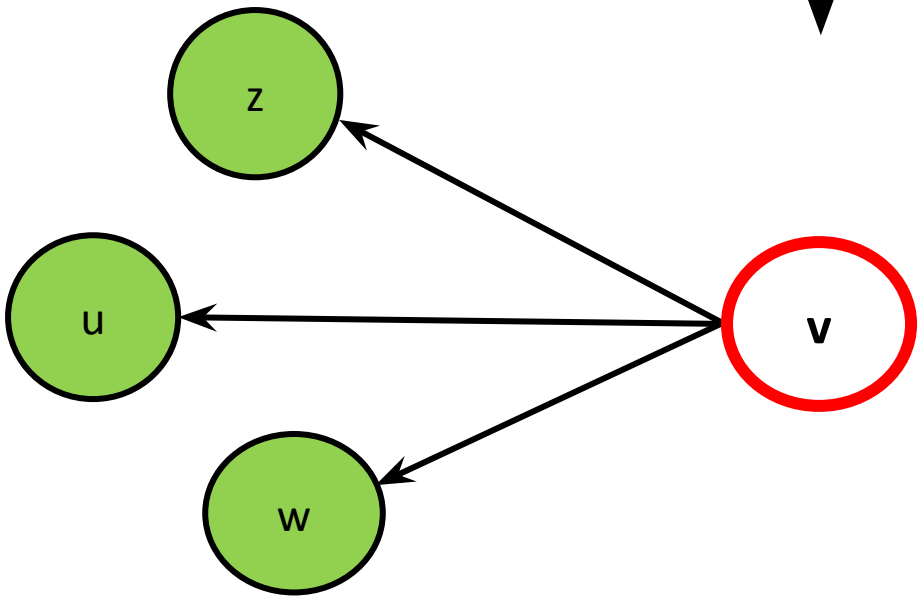
Этапы динамического программирования

1) Задача погружается в семейство вспомогательных подзадач той же природы. Возникающие подзадачи могут являться зависимыми и должны удовлетворять следующим двум требованиям:
   - подзадача должна быть более простой по отношению к исходной задаче;
   - оптимальное решение исходной задачи определяется через оптимальные решения подзадач (в этом случае говорят, что задача обладает свойством *оптимальной подструктуры*, и это один из аргументов в пользу применения для ее решения метода «динамического программирования»).
2) Каждая вспомогательная подзадача решается (рекурсивно) только один раз. Значения оптимальных решений возникающих подзадач запоминаются в таблице, что позволяет не решать повторно ранее решенные подзадачи.
3) Для исходной задачи строится возвратное соотношение, связывающее значение оптимального решения исходной задачи со значениями оптимальных решений вспомогательных подзадач (т. е. методом восходящего анализа от простого к сложному вычисляем значение оптимального решения исходной задачи).
4) Данный этап выполняется в том случае, когда требуется помимо значения оптимального решения получить и само это решение. Часто для этого требуется некоторая вспомогательная информация, полученная на предыдущих этапах метода.

ДП «назад»

ранее решённые подзадачи **z, u, w**

решаем задачу **v**

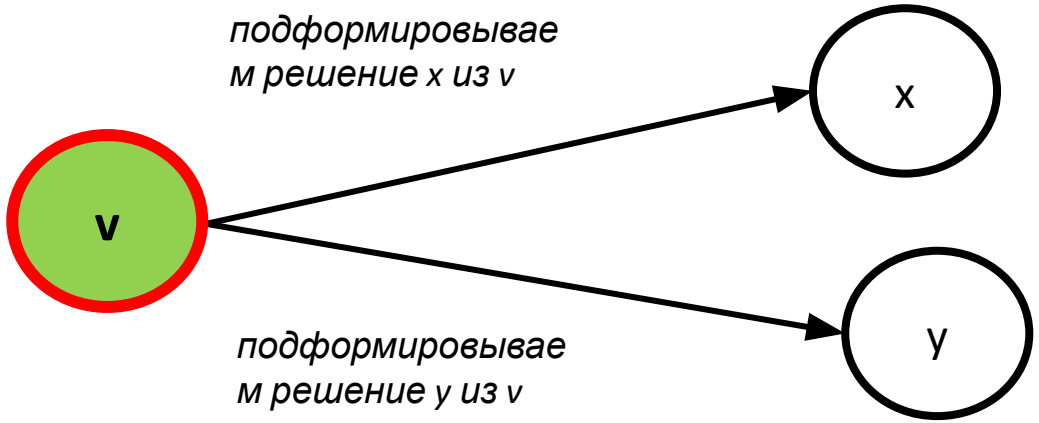


$$f(v) = \mathbf{G}\,(f(z), f(u), f(w))$$

ДП «вперед»

задача **v** уже решена

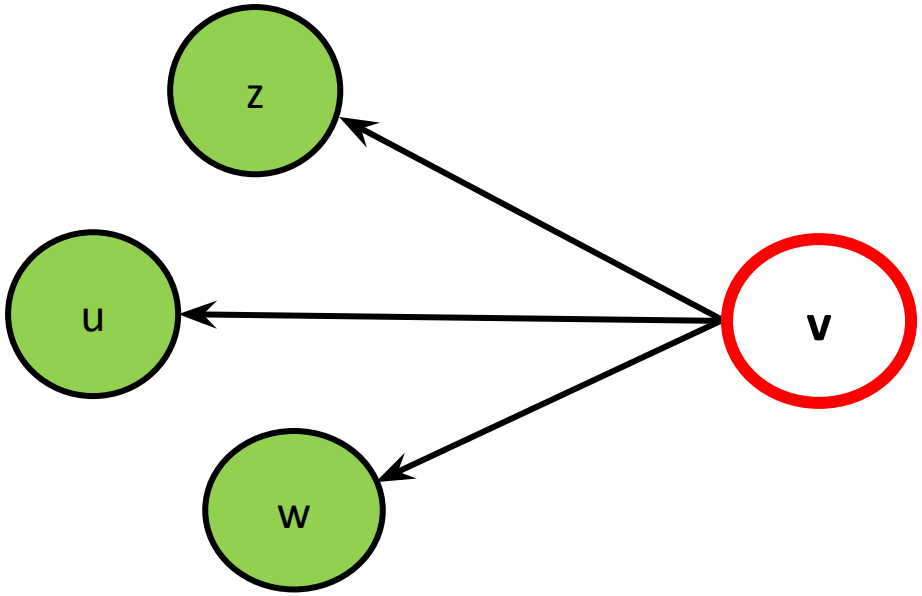
НЕ решённые подзадачи x и y (обе зависимые от v)



$$f(x) = G(\, f(x), f(v)\,)$$

$$f(y) = G(\, f(y), f(v)\,)$$

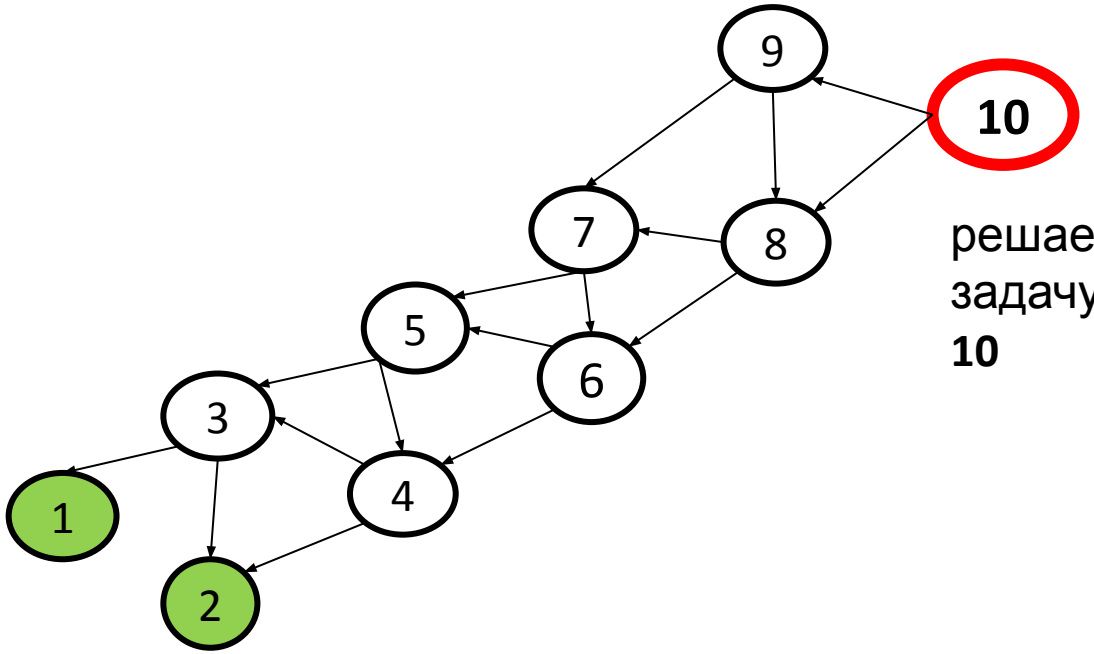
ДП «назад»



ДП «назад» («ленивое»)

не решена

решена



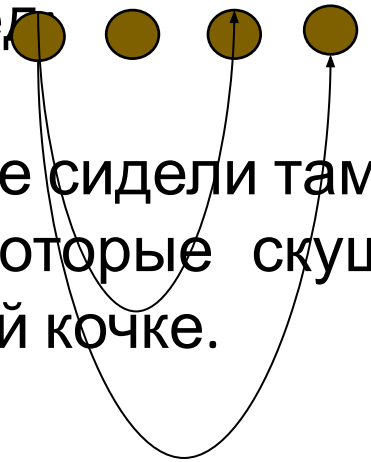
решаем задачу **10**

база ДП

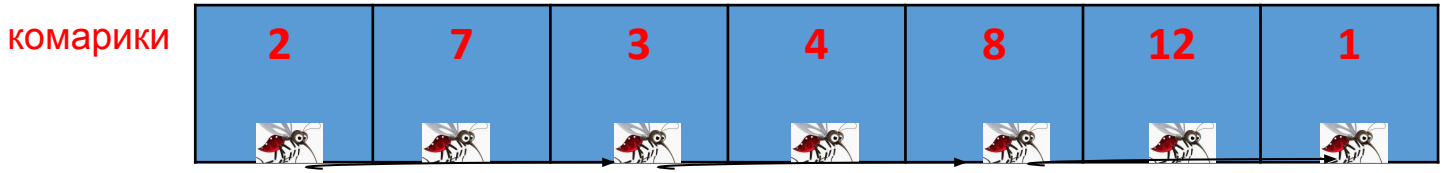
ФПМИ БГУ

Задача 1.
Лягушка

Заданы n кочек.
Лягушка сидит на первой кочке.
На каждой кочке сидят комарики, известно их число.
За один прыжок лягушка может прыгнуть на 2 или 3 кочки вперёд.

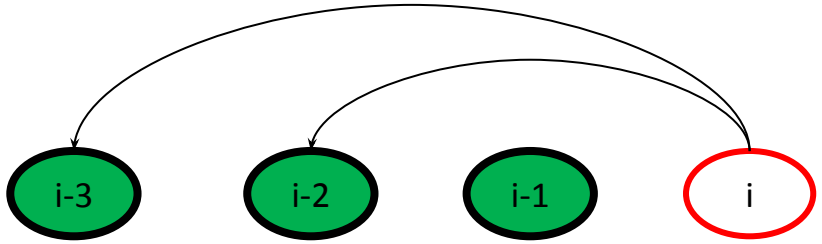


Оказавшись на кочке, лягушка скушает всех комариков, которые сидели там.
Необходимо определить максимальное число комариков, которые скушает лягушка, которой обязательно надо приземлиться на последней кочке.



Ответ:
14

ДП назад (одномерное):



Обозначения:

$F[i]$ – максимальное число комариков, которые скушает лягушка, приземлившись на кочке с номером i;
$array[i]$ – число комариков на кочке с номером i.

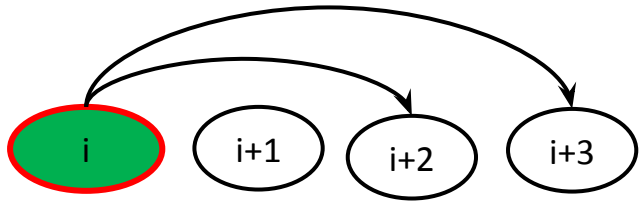
$$\begin{cases} F[1] = array[1] \\ F[2] = -\infty \\ F[i] = \max\{F[i-2], array[i-1]\} + array[i], \quad i = \overline{3, n} \end{cases}$$

$$\begin{cases} F[1] = array[1] \\ F[2] = -\infty \\ F[i] = max(F[i-2], array[i]) + array[i], \quad i = \overline{3, n} \end{cases}$$

i	1	2	3	4	5	6	7
array	2	7	3	4	8	12	1
F	2	$-\infty$	5	6	13	18	14

Ответ:
14

ДП вперёд (одномерное):

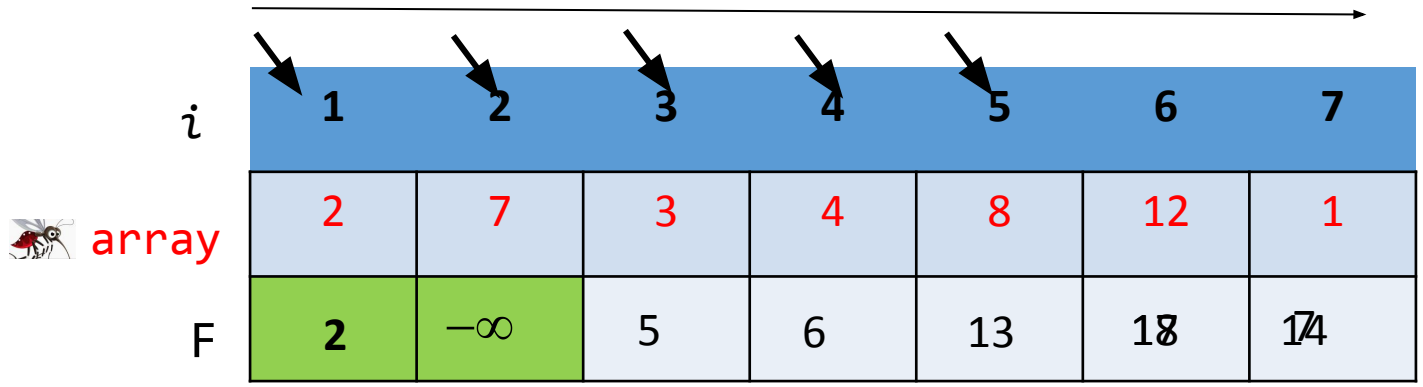


Обозначения:

$F[i]$ – максимальное число комариков, которые скушает лягушка, приземлившись на кочке с номером i

$$\begin{cases} F[1] = array[1] \\ F[2] = -\infty \\ \\ i = \overline{1,n}: \\ F[i+2] = \max\{F[i+2], array[i] + [i+2]\}, \quad i+2 \le n \\ F[i+3] = \max\{F[i+3], array[i] + [i+3]\}, \quad i+3 \le n \end{cases}$$

$$\begin{cases} F[1] = array[1] \\ F[2] = -\infty \\ \\ i = \overline{1,n}: \\ F[i+2] = \max\{F[i+2], array[i] + [i+2 \le n]\} \\ F[i+3] = \max\{F[i+3], array[i] + [i+3 \le n]\} \end{cases}$$



Ответ: 14

Время работы алгоритма, основанного на методе ДП: $O(n)$

Время работы алгоритма для задачи «Лягушка», основанного на методе ДП:

$$O(n)$$

Полный перебор всех вариантов описывается n-м числом Фибоначчи:

$$\Omega(F_n),$$

где F_n – n-ое число Фибоначчи $F_n = \frac{\Phi^n - \hat{\Phi}^n}{\sqrt{5}}$, где $\Phi = \frac{1+\sqrt{5}}{2}$ и $\hat{\Phi} = \frac{1-\sqrt{5}}{2}$

Задача 2. Задача расстановки единиц

Задана строка длины *n*.
Имеется *k* единиц (*k*≤*n*).
Необходимо определить количество способов, для того, чтобы расставить *k* единиц в строке длины *n*.

1									n
	1	1		1	…	1			1

Количество способов можно посчитать комбинаторно:

$$C_n^k = \frac{n!}{k! \cdot (n-k)!}$$

Однако при больших значениях n и k итоговое значение уже может не помещаться в целочисленные типы данных.

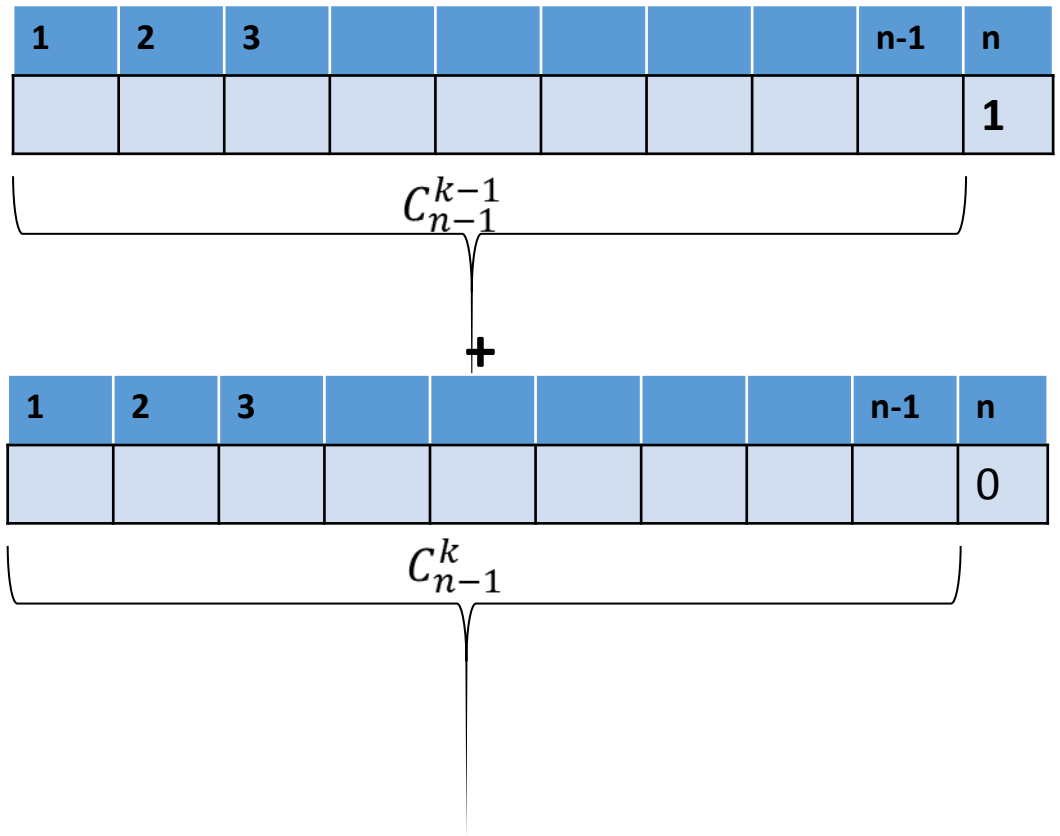
Например, при подсчете числа сочетаний через факториал при

n = 100, k = 1

произойдет переполнение, но в тоже время при вычислении с помощью метода ДП не возникнет проблем, так как итоговое значение равно всего лишь

C_n^k

$$C_n^k = C_{n-1}^{k-1} + C_{n-1}^k$$



ДП назад (двумерное):

Обозначим

F [*i*, *j*] - количество способов, для того, чтобы в строке длины i расставить j единиц.

1	2	3						*i*-1	*i*
	1					1		1	

j единиц

$$\begin{cases} F[i,i] = 1, i = \overline{0,n} \\ F[i,0] = 1, i = \overline{1,n} \\ F[i,j] = F[i-1,j-1] + F[i-1,j], \quad i = \overline{1,n}, \quad j = \overline{1,i-1} \end{cases}$$

длина строки

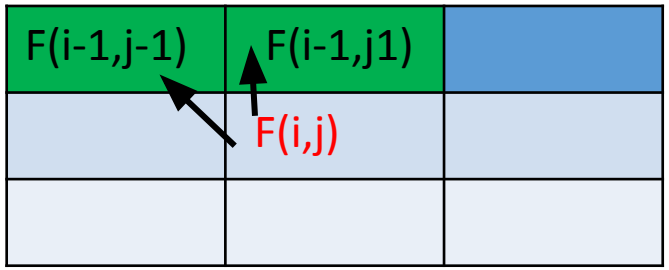
↓	0	1	2	3	…
0	**1**				
1	**1**	**1**			
2	**1**		**1**		
3	**1**			**1**	
…	**1**				**1**

число единиц

F(i-1,j-1)	F(i-1,j)	
	F(i,j)	

ДП назад (двумерное):

$$\begin{cases} F[i,i] = 1, i = \overline{0,n} \\ F[i,0] = 1, i = \overline{1,n} \\ F[i,j] = F[i-1,j-1] + F[i-1,j], \quad i = \overline{1,n}, \quad j = \overline{1,i-1} \end{cases}$$



	0	1	2	3	4
0	**1**				
1	**1**	**1**			
2	**1**	2	**1**		
3	**1**	3	3	**1**	
4	**1**	4	6	4	**1**

Время работы алгоритма:

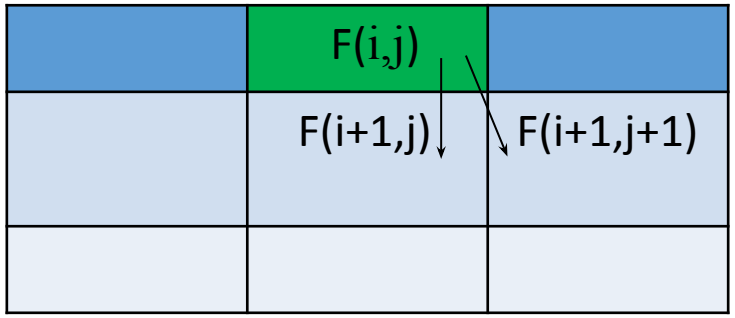
$$O(n \cdot k) = O(n^2)$$

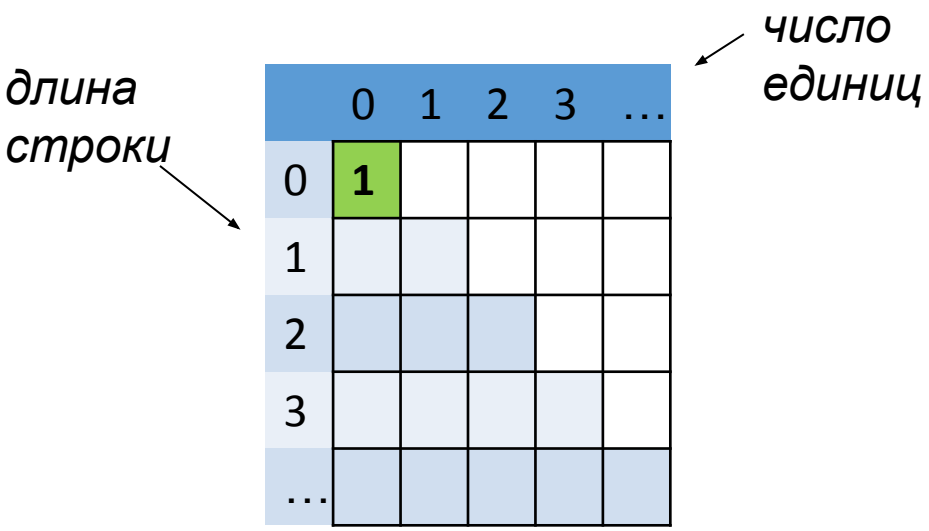
ДП вперёд (двумерное):

$$C_n^k = C_{n-1}^{k-1} + C_{n-1}^k$$

Обозначим через `F[i,j]` - количество способов, для того, чтобы расставить *j* единиц в строке длины *i*.

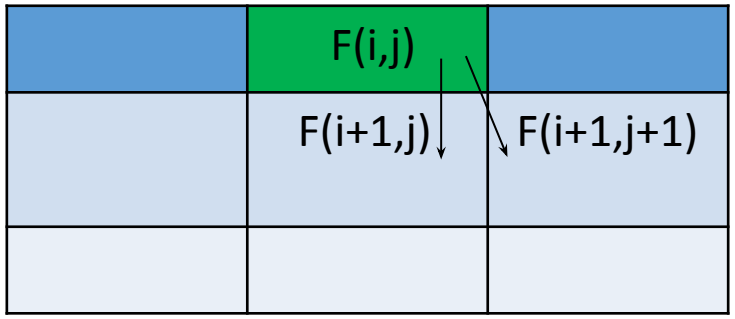
$$\begin{cases} F[0,0] = 1 \\ F[i,j] = 0, i = \overline{1,n}, \quad j = \overline{0,i} \\ F[i+1,j] = F[i+1,j] + F[i,j], i = \overline{1,n-1}, \quad j = \overline{0,i} \\ F[i+1,j+1] = F[i+1,j+1] + F[i,j], i = \overline{1,n-1}, \quad j = \overline{0,i} \end{cases}$$





ДП вперёд (двумерное):

$$\begin{cases} F[0,0]=1 \\ F[i,j]=0, i=\overline{1,n}, \quad j=\overline{0,i} \\ F[i+1,j]=F[i+1,j]+F[i,j], i=\overline{1,n-1}, \quad j=\overline{0,i} \\ F[i+1,j+1]=F[i+1,j+1]+F[i,j], i=\overline{1,n-1}, \quad j=\overline{0,i} \end{cases}$$



	0	1	2	3	4
0	1				
1	1	1			
2	1	2	1		
3	1	3	3	1	
4	1	4	6	4	1

Время работы алгоритма:

$$O(n \cdot k) = O(n^2)$$

Время работы алгоритма «Расстановка единиц», основанного на методе динамического программирования:

$$\mathrm{O}\left(k \cdot n\right) = \mathrm{O}\left(n^2\right)$$

Количество способов можно посчитать и комбинаторно, но при больших значениях *n* и *k* промежуточные результаты вычислений могут уже не помещается в целочисленные типы данных.

На практике, когда результат является достаточно большим числом, в задаче предлагается найти ответ по модулю (% p).

эквивалентные формы записи:

$a \% p = y$

$a \bmod p = y$

Свойства модульной арифметики:

$$(A + B) \% p = ((A \% p) + (B \% p)) \% p$$

$$(A - B) \% p = \big((A \% p) - (B \% p) + p\big) \% p$$

$$(A \cdot B) \% p = ((A \% p) \cdot (B \% p)) \% p$$

Если два числа сравнимы по модулю p, $a \bmod p = b \bmod p$, то это записывается: $a \equiv b (\bmod p)$ т.е.

Малая теорема Ферма

Если p – простое число, *a* – целое число, которое не делится на *p*, $a^{p-1} \equiv 1 (\bmod p)$

Следствие из малой теоремы Ферма (a – целое, p – простое число):

$$a^{p-2} \equiv \frac{1}{a} (\bmod p), \quad \text{другими словами: } \frac{\mathbf{1}}{\boldsymbol{a}} \% \boldsymbol{p} = \boldsymbol{a}^{\boldsymbol{p-2}} \% \boldsymbol{p}$$

Задача 3.
Оптимального перемножения группы матриц

Заданы *s* матриц: $A = A_1 \cdot A_2 \cdot \ldots \cdot A_s$

$$A_i\,[n_i \times m_i], m_i = n_{i+1}, \forall i = \overline{1, s-1}$$

Определить, какое минимальное число операций умножения требуется для перемножения *s* матриц, причем перемножать можно любые две рядом стоящие матрицы.

Порядок перемножения всех *s* матриц неоднозначен. Чтобы устранить неоднозначность, нужно расставить скобки. Порядок расстановки скобок однозначно определит последовательность перемножаемых матриц.

Поскольку матричное произведение ассоциативно, то результат не зависит от расстановки скобок, но порядок перемножения может существенно повлиять на время работы алгоритма.

Сведения из математики:
при перемножении двух матриц:
B [n × k] * C [k × m]
получим матрицу **D [n × m]**
Выполнив **n·k·m** операций умножения.

$$A = A_1 \cdot A_2 \cdot A_3$$

$A_1 - [20 \times 5]$

$A_2 - [5 \times 10]$

$A_3 - [10 \times 4]$

?

1 порядок:

$$A=(A_1)\cdot(A_2 \cdot A_3)$$

$$[20 \times 5][5 \times 4]$$

Число операций умножения:

$$(5 \cdot 10 \cdot 4)+(20 \cdot 5 \cdot 4)=600$$

2 порядок:

$$A=(A_1 \cdot A_2)\cdot(A_3)$$

$$[20 \times 10][10 \times 4]$$

Число операций умножения:

$$(20 \cdot 5 \cdot 10)+(20 \cdot 10 \cdot 4)=1\ 800$$

Числа Каталана – это последовательность чисел, названная в честь бельгийского математика Эжен Шарля Каталана.

C_n - обозначение n-ого числа Каталана.

1. Количество способов расстановки скобок в произведении из (n+1) множителя.
2. Количество двоичных корневых деревьев с n листьями, у которых из каждого внутреннего узла выходит ровно 2 узла.
3. Количество правильных скобочных последовательностей длины 2n.
4. Количество триангуляций выпуклого (n+2)–угольника (разбиение на треугольники непересекающимися диагоналями).

	0	1	2	3	4	5	6	7	8
C_n	**1**	**1**	**2**	**5**	**14**	**42**	**132**	**429**	**1430**

Эжен Шарль Каталан

фр. *Eugène-Charles Catalan*

Эжен Шарль Каталан

Дата рождения	30 мая 1814
Место рождения	Брюгге
Дата смерти	14 февраля 1894 (79 лет)
Место смерти	Льеж
Страна	Бельгия Франция
Научная сфера	математика
Место работы	Санкт-Петербургская академия наук Льежский университет
Альма-матер	Политехническая школа Льежский университет
Научный руководитель	Жозеф Лиувилль

	0	1	2	3	4	5	6	7	8
C_n	1	1	2	5	14	42	132	429	1430

рекуррентная формула для C_n

$$C_n = \sum_{k=0}^{n-1} C_k \cdot C_{n-1-k} = C_0 \cdot C_{n-1} + C_1 \cdot C_{n-2} + C_2 \cdot C_{n-3} + \cdots + C_{n-1} \cdot C_0$$

$$C_4 = \binom{C_0}{\times}{C_3} + \binom{C_1}{\times}{C_2} + \binom{C_2}{\times}{C_1} + \binom{C_3}{\times}{C_0}$$

$$\begin{array}{cccc} 1 & 1 & 2 & 5 \\ + \times & \times & \times & \times \\ 5 & 2 & 1 & 1 \end{array}$$

$$5 + 2 + 2 + 5 = 14$$

$$C_n = C_{n-1} \cdot \frac{2 \cdot (2n-1)}{(n+1)}$$

аналитическая формулы для C_n

$$C_n = \frac{(2n)!}{n!\,(n+1)!}$$

$$C_n \approx \frac{4^n}{n^{\frac{3}{2}} \cdot \sqrt{\pi}}$$

Числа Каталана в треугольнике Паскаля

0							1					
1						1		1				
2					1		2		1			
3				1		3		3		1		
4			1		**4**		**6**		4		1	
6		1		5		10		10		5		1
7	1		6		15		20		15		6		1

Если в треугольнике Паскаля в строке n слева направо пронумеровать числа (нумерация с 0), то m-е число есть биномиальный коэффициент: (число способов выбрать m элементов из n)

$$C_n^m = \binom{n}{m} = \frac{n!}{m!(n-m)!}$$

Если в чётных строках *i* от серединной линии отнять соседний элемент,

то получиться C_n число Каталана.

	0	**1**	**2**	**3**	**4**	**5**
C	1	1	2	5	14	42

n

ФПМИ БГУ

Задача

$$A = A_1 \cdot A_2 \cdot \ldots \cdot A_s$$

$$A_i\,[n_i \times m_i], m_i = n_{i+1}, \forall i = \overline{1, s-1}$$

Количество различных способов задать однозначно порядок перемножения матриц – C_{s-1} число Каталана, т.е. экспоненциальная функция.

$$C_{s-1} \approx \frac{4^{s-1}}{n^{\frac{3}{2}} \cdot \sqrt{\pi}}$$

Метод динамического программирования позволит решить задачу за время :

$$O\left(s^3\right)$$

Задача

$$A = A_1 \cdot A_2 \cdot \ldots \cdot A_s$$

$$A_i\,[n_i \times m_i], m_i = n_{i+1}, \forall i = \overline{1, s-1}$$

Подзадача

Обозначим через $F[i, j]$ минимальное число операций умножения, чтобы перемножить матрицы с номерами от ***i*** до ***j*** включительно:

$$A_i \cdot A_{i+1} \cdot \ldots \cdot A_{k+1} \cdot A_k \cdot A_{k+1} \cdot \ldots \cdot A_{j-1} \cdot A_j$$

Ответ: $F[1, s]$

Предположим, что мы решили подзадачу оптимального перемножения группы матриц $A_i \cdot A_{i+1} \cdot \ldots \cdot A_j$

На последнем этапе, когда формировалась результирующая матрица $A[n_i \times m_j]$, должны были перемножаться две матрицы. Рассмотрим все возможные варианты того, как эти две матрицы могли быть получены.

Фиксируем некоторое значение k: $i \leq k < j$

$$(\boldsymbol{A_i \cdot A_{i+1} \cdot \ldots \cdot A_{k+1} \cdot A_k}) \cdot (\boldsymbol{A_{k+1} \cdot \ldots \cdot A_{j-1} \cdot A_j})$$
$$A[n_i \times m_k] \quad \cdot \quad A[n_{k+1} \times m_j]$$

Так как у нас оптимизационная задача, то перемножать матрицы

$$A_i \cdot A_{i+1} \ldots \cdot A_k$$
$$A_{k+1} \cdot A_{k+2} \ldots \cdot A_j$$

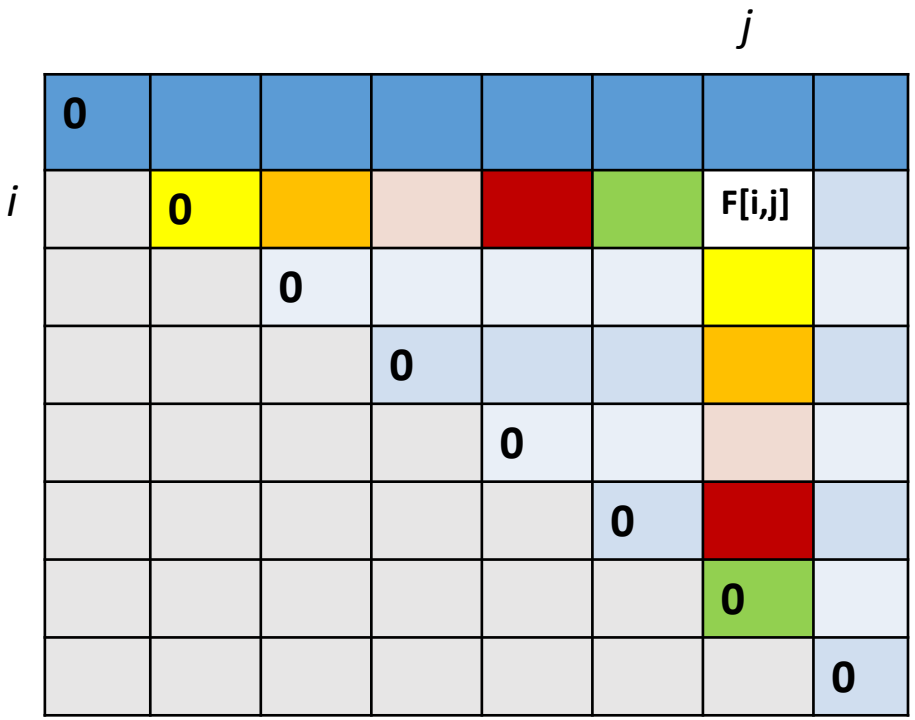
надо за минимально возможно число операций умножения.

$$(A_i \cdot \ldots \cdot A_k) \cdot (A_{k+1} \cdot \ldots \cdot A_j)$$
$$A[n_i \times m_k] \quad \cdot \quad A[n_{k+1} \times m_j]$$

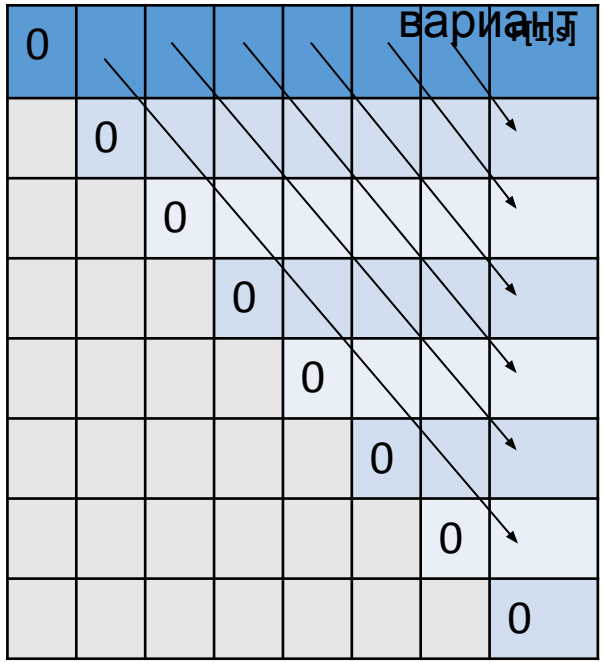
Справедливо следующее рекуррентное соотношение:

$$\begin{cases} F[i,i] = 0, \quad i = \overline{1,s} \\ F[i,i+1] = n_i \cdot m_i \cdot m_{i+1}, \quad i = \overline{1,s-1} \\ F[i,j] = \min_{i \le k < j} \{F[i,k] + F[k+1,j] + n_i \cdot m_k \cdot m_j\}, \quad i = \overline{1,s}, \quad j > i \end{cases}$$

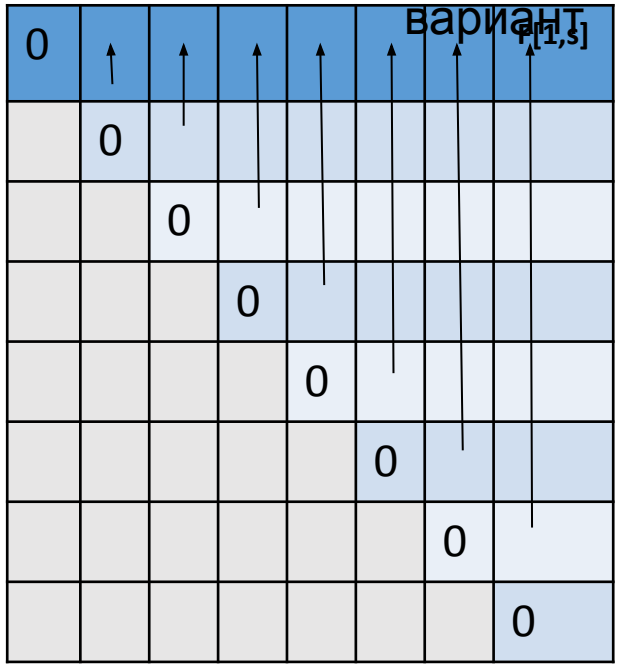
$$\begin{cases} F[i,i] = 0, \quad i = \overline{1,s} \\ F[i,i+1] = n_i \cdot m_i \cdot m_{i+1}, \quad i = \overline{1,s-1} \\ F[i,j] = \min_{i \le k < j} \{F[i,k] + F[k+1,j] + n_i \cdot m_k \cdot m_j\}, \quad i = \overline{1,s}, \quad j > i \end{cases}$$



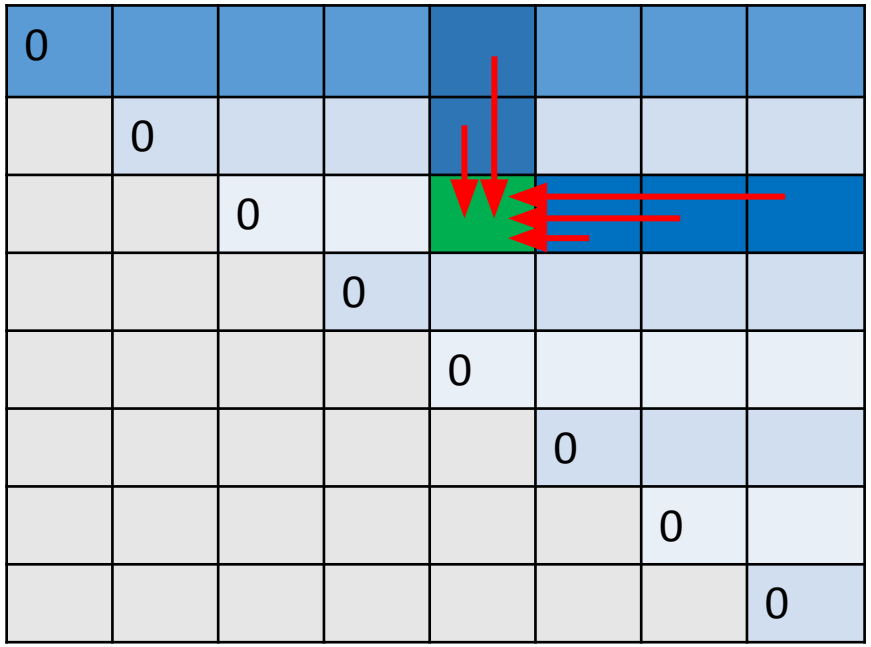
1 вариант



2 вариант



Зависимые
подзадачи



```python
def matrix_chain_multiplication_order(dim):
    n=len(dim)
    m = [[0 for i in range(n)] for j in range(n)]
    for l in range(2, n):
        for i in range(1, n-l+1):
            j = i+l-1
            m[i][j] = float('Inf')
            for k in range(i, j):
                cost = m[i][k] + m[k+1][j] + dim[i-1]*dim[k]*dim[j]
                if cost < m[i][j]:
                    m[i][j] = cost

    return m[1][n-1]
```

Время работы алгоритма оптимального перемножения группы матриц, основанного на методе динамического программирования:

- вычислить `s(s+1)/2` элементов таблицы;
- каждый элемент таблицы вычисляется ровно один раз за не более, чем s арифметических операций.

$$O\left(s^3\right)$$

Приме
р

$$A = A_1 \cdot A_2 \cdot A_3 \cdot A_4$$

$$A_1 - [20 \times 5]$$

$$A_2 - [5 \times 35]$$

$$A_3 - [35 \times 4]$$

$$A_4 - [4 \times 25]$$

$A_1 - [20 \times 5]$
$A_2 - [5 \times 35]$
$A_3 - [35 \times 4]$
$A_4 - [4 \times 25]$

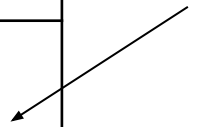
	1	2	3	4
1	**0**	$(A_1 \cdot A_2)$ **3 500**		
2		**0**	$(A_2 \cdot A_3)$ **700**	
3			**0**	$(A_3 \cdot A_4)$ **3 500**
4				**0**

$A_1 - [20 \times 5]$
$A_2 - [5 \times 35]$
$A_3 - [35 \times 4]$
$A_4 - [4 \times 25]$

	1	2	3	4
1	**0**	$(A_1 \cdot A_2)$ **3 500**	$(A_1 \cdot A_2 \cdot A_3)$ **1 100** k=1 $(A_1) \cdot (A_2 \cdot A_3) = 1\,100$ $(A_1 \cdot A_2) \cdot (A_3) = 6\,300$	
2		**0**	$(A_2 \cdot A_3)$ **700**	
3			**0**	$(A_3 \cdot A_4)$ **3 500**
4				**0**

$A_1 - [20 \times 5]$
$A_2 - [5 \times 35]$
$A_3 - [35 \times 4]$
$A_4 - [4 \times 25]$

	1	2	3	4
1	**0**	$(A_1 \cdot A_2)$ **3 500**	$(A_1 \cdot A_2 \cdot A_3)$ **1 100** k=1	
2		**0**	$(A_2 \cdot A_3)$ **700**	$(A_2 \cdot A_3 \cdot A_4)$ **1 200** k=3 $(A_2) \cdot (A_3 \cdot A_4) = 7\,875$ $(A_2 \cdot A_3) \cdot (A_4) = 1\,200$
3			**0**	$(A_3 \cdot A_4)$ **3 500**
4				**0**



$A_1 - [20 \times 5]$
$A_2 - [5 \times 35]$
$A_3 - [35 \times 4]$
$A_4 - [4 \times 25]$

	1	2	3	4
1	**0**	$(A_1 \cdot A_2)$ **3 500**	$(A_1 \cdot A_2 \cdot A_3)$ **1 100** k=1	$(A_1 \cdot A_2 \cdot A_3 \cdot A_4)$ **3 100** k=3 $(A_1) \cdot (A_2 \cdot A_3 \cdot A_4)$=3 700 $(A_1 \cdot A_2) \cdot (A_3 \cdot A_4)$=24 500 $(A_1 \cdot A_2 \cdot A_3) \cdot (A_4)$=3 100
2		**0**	$(A_2 \cdot A_3)$ **700**	$(A_2 \cdot A_3 \cdot A_4)$ **1 200** k=3
3			**0**	$(A_3 \cdot A_4)$ **3 500**
4				**0**

$$A = A_1 \cdot A_2 \cdot A_3 \cdot A_4$$
$$A_1 - [20 \times 5]$$
$$A_2 - [5 \times 35]$$
$$A_3 - [35 \times 4]$$
$$A_4 - [4 \times 25]$$

Отве

Т: 3 100 операций умножения

Порядок перемножения:

$(A_1 \cdot A_2 \cdot A_3) \cdot A_4$

$(A_1 \cdot (A_2 \cdot A_3)) \cdot A_4$

	1	2	3	4
1	**0**	$(A_1 \cdot A_2)$ **3 500**	$(A_1 \cdot A_2 \cdot A_3)$ **1 100** k=1	$(A_1 \cdot A_2 \cdot A_3 \cdot A_4)$ **3 100** k=3
2		**0**	$(A_2 \cdot A_3)$ **700**	$(A_2 \cdot A_3 \cdot A_4)$ **1 200** k=3
3			**0**	$(A_3 \cdot A_4)$ **3 500**
4				**0**

Задача 4.
Наибольшая общая подпоследовательность (НОП)

англ. longest common subsequence (**LCS)**

Для конечной последовательности

$$Z = x_1, x_2, \dots, x_k$$

некоторую её **подпоследовательность** можно получить, если удалить из Z некоторое (возможно пустое) множество элементов.

Менять элементы последовательности местами нельзя.

Удаляемые элементы не обязательно идут подряд.

Пример.

$Z = x_1, x_2, x_3, x_4, x_5$

$x_1, \cancel{x_2}, x_3, x_4, \cancel{x_5} = x_1, x_3, x_4$ - подпоследовательность для Z

Крайние случаи:

- ✔ самая короткая - пустая подпоследовательность (удалены все элементы последовательности Z);
- ✔ самая длинная - совпадает с самой последовательностью (удалено нулевое множество элементов).

Заданы две конечные последовательности

$$X = x_1, x_2, \ldots, x_n$$

$$Y = y_1, y_2, \ldots, y_m.$$

Необходимо найти **наибольшую общую подпоследовательность** двух последовательностей X и *Y*, т.е. общую их подпоследовательность, имеющую максимальную длину (англ. longest common subsequence).

Пример 1.

X=Ф,Б,П,Г,М,У,И

Y=А,Ф,И,П,С,Д,М,И,Б,Г,У

LCS (X,Y) = Ф,П,М,И

|LCS (X,Y)| = 4

Пример 2.

X=0,2,0,9,1,9,6,7,22

Y=2,5,0,1,1,9,5,5,22

LCS (X,Y)=2,0,1,9,22

|LCS (X,Y)| = 5

В общем случае задача может иметь неоднозначное решение.

Задачу LCS можно решить полным перебором

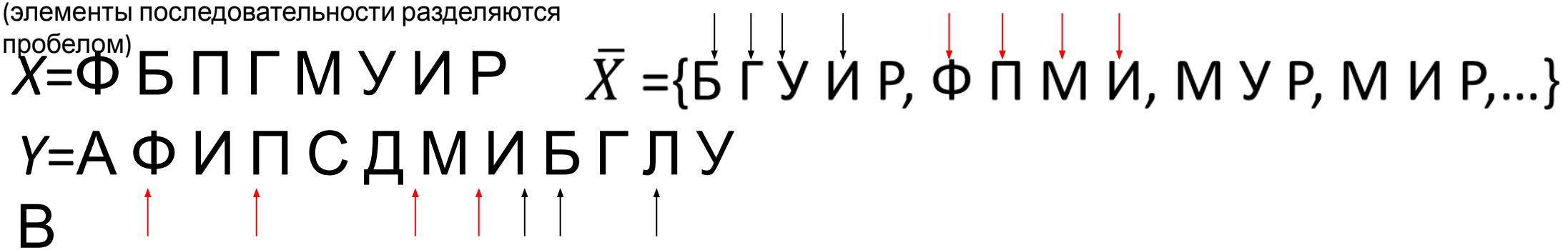
1. Построим множество $\bar{X}$, в которое добавим все возможные подпоследовательности последовательности $X = x_1, x_2, \dots, x_n$.

   Сколько подпоследовательностей будет сгенерировано? $|\bar{X}| = 2^n$

2. Для каждой последовательности из множества $\bar{X}$ проверяем, является ли она подпоследовательностью для $Y = y_1, y_2, \dots, y_m$? Если является, то среди таких последовательностей выберем ту, у которой наибольшая длина.

Время работы алгоритма LCS (X,Y), основанного на полном переборе: $\boldsymbol{\Omega(2^n \cdot m)}$

(элементы последовательности разделяются пробелом)



Задачу LCS можно решить динамическим программированием

Обозначим через F[i,j] длину наибольшей общей подпоследовательности для двух префиксов:

$$X_i = x_1, x_2, \dots, x_i$$

$$Y_j = y_1, y_2, \dots, y_j.$$

Рассмотрим последние элементы префиксов X_i и Y_j.

Возможны два случая:

1) $x[i] = y[j]$;
2) $x[i] \neq y[j]$.

F

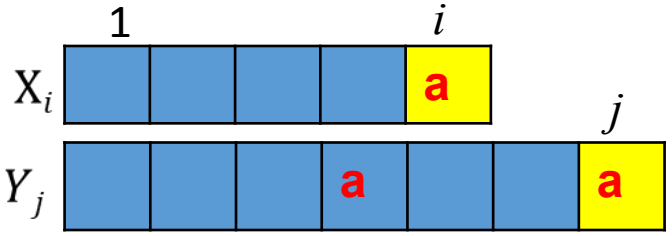
	0	1 y_1	2 y_2	3 y_3	…	m y_m
0	0	0	0	0	0	0
1 x_1	0					
2 x_2	0					
3 x_3	0					
…	0					
N x_n	0					?

Граничные условия (один из префиксов пустой):

$$f[0, j] = 0, j = 0, \dots, m$$
$$f[i, 0] = 0, i = 0, \dots, n$$

Случай **1**.

Предположим, что $x[i] = y[j]$



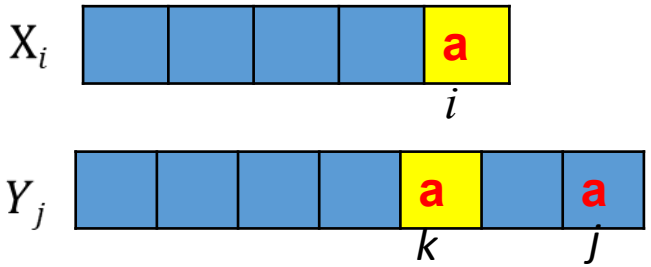
Утверждение.

Если $x[i] = y[j]$, то $НОП(X_i, Y_j)$ обязательно будет содержать самым последним элементом $x[i] = y[j] = a$

От противного.

1) Предположим, что последний элемент $НОП(X_i, Y_j)$ отличен от $x[i] = y[j] = a$ (т.е. общая подпоследовательность завершилась элементом, который стоит в префиксах не последним). Тогда к $НОП(X_i, Y_j)$ в конец добавим $x[i] = y[j] = a$, получая при этом $ОП(X_i, Y_j)$ большей на 1 длины, чем построенная ранее $НОП(X_i, Y_j)$. Противоречие.

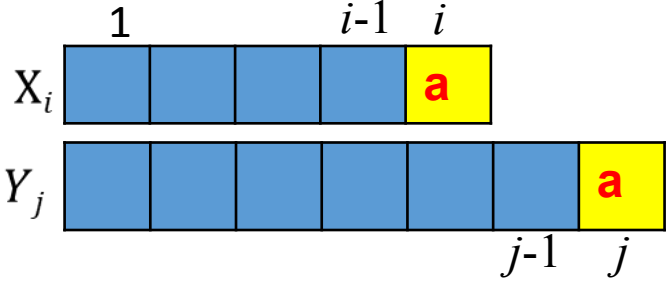
2) Рассмотрим случай, когда последний элемент $НОП(X_i, Y_j)$ равен $x[i] = a$, но $\neq y[j]$,
т. е. из Y_j элемент $y[j]$ был вычеркнут (случай $y[j] = a$, но $\neq x[i]$ рассматривается аналогично).



Тогда в последовательности Y_j есть еще один элемент $y[k] = a$, который стоит в Y_j раньше, т. е. $k < j$ и после него никто более не вошел в $НОП(X_i, Y_j)$. Поэтому $НОП(X_i, Y_j)$ не изменится, если мы вычеркнем из последовательности Y_j элемент $y[k]$, а добавим вместо него $y[j]$.
Доказательство завершено.

Случай **1**.

Предположим, что $x[i] = y[j]$



Утверждение.

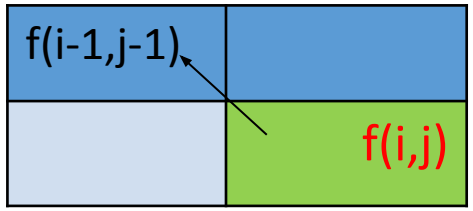
Если $x[i] = y[j]$**, то** $НОП(X_i, Y_j)$ **обязательно будет содержать самым последним элементом** $x[i] = y[j] = a$

Поэтому в Случае 1 можно пока забыть про элементы $x[i]$ и $y[j]$, решить задачу НОП (X_{i-1}, Y_{j-1}), а затем в конец построенной НОП (X_{i-1}, Y_{j-1}) приписать элемент $x[i]$ $(= y[j])$.

Рекуррентное соотношение:

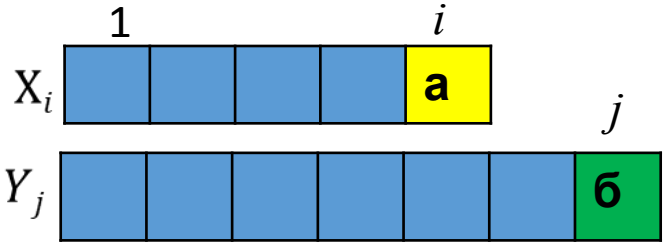
если $x[i] = y[j]$, то

$$f[i,j] = f[i-1, j-1] + 1$$



Случай **2**.

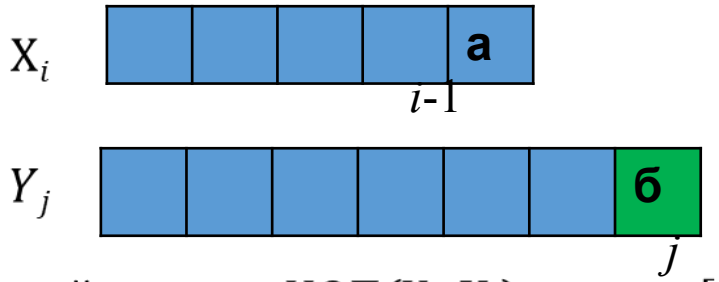
Предположим, что $x[i] \neq y[j]$



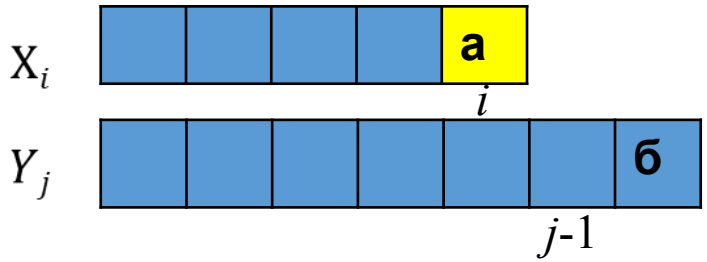
Утверждение.

Если $x[i] \neq y[j]$, то $НОП(X_i, Y_j)$ точно не сможет завершиться на один из этих элементов.

1) Предположим, что последний элемент $НОП(X_i, Y_j)$ равен $y[j]$. Тогда элемент $x[i]$ не войдет в $НОП(X_i, Y_j)$, поэтому его можно вычеркнуть из X_i и $НОП(X_i, Y_j) = НОП(X_{i-1}, Y_j)$.



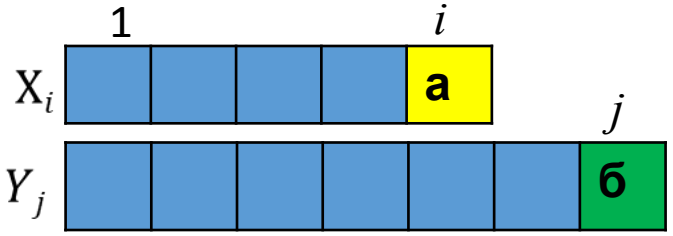
2) Предположим, что последний элемент $НОП(X_i, Y_j)$ равен $x[i]$. Тогда элемент $y[j]$ не войдет в $НОП(X_i, Y_j)$, поэтому его можно вычеркнуть из Y_j и $НОП(X_i, Y_j) = НОП(X_i, Y_{j-1})$.



3) Случай, когда в $НОП(X_i, Y_j)$ не входит ни один из элементов $x[i], y[j]$ будет рассмотрен через шаг в подзадачах $НОП(X_{i-1}, Y_j)$ и $НОП(X_i, Y_{j-1})$

Случай **2**.

Предположим, что $x[i] \neq y[j]$



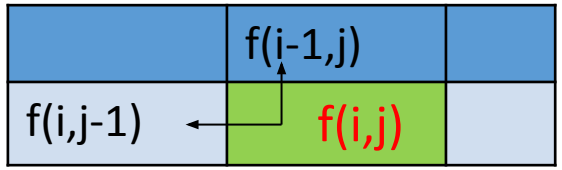
Утверждение.

Если $x[i] \neq y[j]$, то $НОП(X_i, Y_j)$ точно не сможет завершиться на один из этих элементов.

Получаем для Случая 2 следующее рекуррентное соотношение:

если $x[i] \neq y[j]$, то

$$f[i,j] = max\{f[i-1,j], f[i,j-1]\},$$



Объединяя оба случая, получаем следующее рекуррентное соотношение:

$$f[0,j] = 0, j = 0, \dots, m$$
$$f[i,0] = 0, i = 0, \dots, n$$

$$i = 1, \dots, n$$
$$j = 1, \dots, m$$

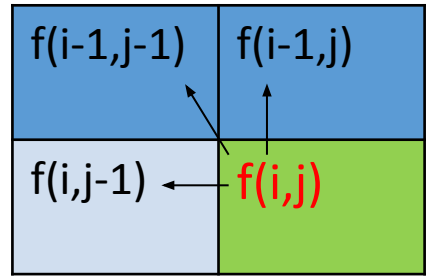
$$f[i,j] = \begin{cases} f[i-1, j-1] + 1, & if \quad x[i] = y[j] \\ max\{f[i-1,j], f[i,j-1]\}, & if \quad x[i] \neq y[j] \end{cases}$$

$f[n,m]$ – решение задачи

		0	1 м	2 а	3 м	4 и	5 т	m=6 а
0		0	0	0	0	0	0	0
1	м	0	1	1	1	1	1	1
2	о	0	1	1	1	1	1	1
3	т	0	1	1	1	1	2	2
4	м	0	1	1	2	2	2	2
n=5	а	0	1	2	2	2	2	3

ДП
назад

$$f[i,j] = \begin{cases} f[i-1,j-1]+1, & if \quad x[i] = y[j] \\ max\{f[i-1,j], f[i,j-1]\}, & if \quad x[i] \neq y[j] \end{cases}$$



1-й способ

F

	0	1	2	3	…	m
0	0	0	0	0	0	0
1	0	→	→	→	→	→
2	0	→	→	→	→	→
3	0	→	→	→	→	→
…	0	→	→	→	→	→
n	0	→	→	→	→	?

2-й способ

	0	1	2	3	…	m
0	0	0	0	0	0	0
1	0	↓	↓	↓	↓	↓
2	0	↓	↓	↓	↓	↓
3	0	↓	↓	↓	↓	↓
…	0	↓	↓	↓	↓	↓
n	0	↓	↓	↓	↓	?

если не нужно восстанавливать саму подпоследовательность, то можно в памяти хранить только предыдущую строку матрицы и текущую (предыдущий столбец и текущий)

Время работы алгоритма, основанного на ДП: $O(n \cdot m)$

Восстановление наибольшей общей подпоследовательности – «обратный ход»:

стартуем из $f[n, m]$;

движение осуществляем до тех пор, пока не придем в нулевую строку или нулевой столбец (либо пока не придем к нулевому элементу матрицы);

движение осуществляем по следующему правилу:

если $x[i] = y[j]$, то добавляем $x[i]$ (или $y[j]$) к ответу и переходим к элементу $f[i-1, j-1]$;

если $x[i] \neq y[j]$, то переходим к любому из элементов $f[i-1, j]$ и $f[i, j-1]$, который равен $f[i, j]$;

Для получения $НОП(X, Y)$ нужно перевернуть полученный ответ.

		0	1 м	2 а	3 м	4 и	5 т	m=6 а
0		0	0	0	0	0	0	0
1	м	0	1	1	1	1	1	1
2	о	0	1	1	1	1	1	1
3	т	0	1	1	1	1	2	2
4	м	0	1	1	2	2	2	2
n=5	а	0	1	2	2	2	2	3

$НОП(м\ а\ м\ и\ т\ а, м\ о\ т\ м\ а) =$

м т а

(в примере при неоднозначности движение шло вверх)

ФПМИ БГУ

Задача 5.
Наибольшая подпоследовательность-палиндром

Задана строка длины n.
Необходимо вычеркнуть минимальное число элементов так, чтобы получился палиндром
(палиндром - *строка, которая одинаково читается слева направо и справа налево*).

Для строки: a s d f s l a

Наибольшая подпоследовательность-палиндром: a s d s a

Неявное решение задачи

Предположим, что у нас задана строка $X = x_1, x_2, \dots, x_n$.

Перевернем эту строку: $\overline{X} = x_n, x_{n-1}, \dots, x_1$

Наибольшая подпоследовательность-палиндром = $НОП(X_n, \overline{X}_n)$.

X=a s d f s l a

		a	l	s	f	d	s	a
	0	0	0	0	0	0	0	0
a	0	1	1	1	1	1	1	1
s	0	1	1	2	2	2	2	2
d	0	1	1	2	2	3	3	3
f	0	1	1	2	3	3	3	3
s	0	1	1	2	3	3	4	4
l	0	1	2	2	3	3	4	4
a	0	1	2	2	3	3	4	5

Время работы $\Theta(n^2)$.

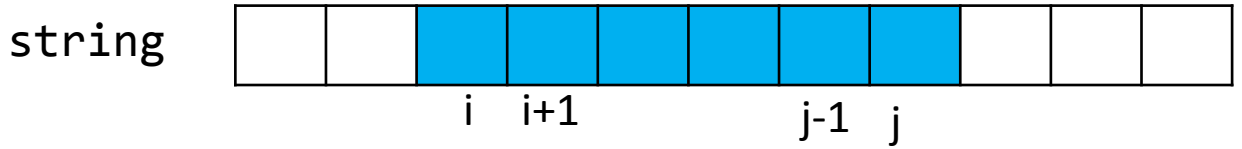
Требуемая память $\Theta(n^2)$.

Наибольший палиндром

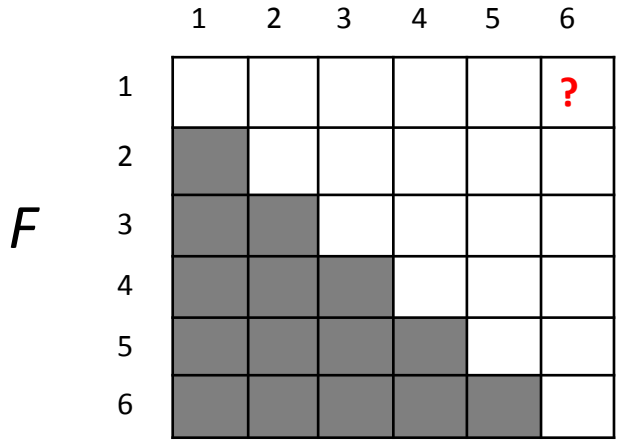
a s d s a

Явное решение задачи

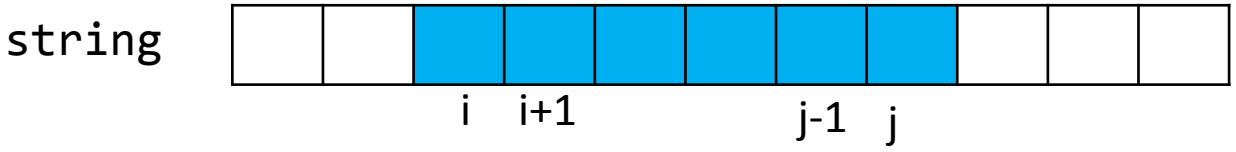
Обозначим через $F[i,j]$ длину наибольшего палиндрома, который можно получить, если мы рассматриваем элементы строки от индекса i до j включительно.



Ответ:



n=6



Строки длины 1

$$F[i,i]=1, i=\overline{1,n}$$

Строки длины 2

$$i=\overline{1,n-1}$$

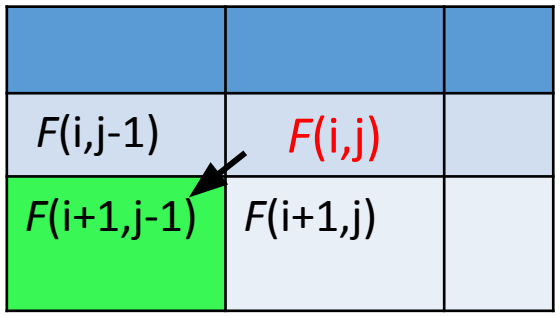
$$F[i,i+1]=\begin{cases}2, if & string[i]=string[i+1]\\ 1, if & string[i]\neq string[i+1]\end{cases}$$

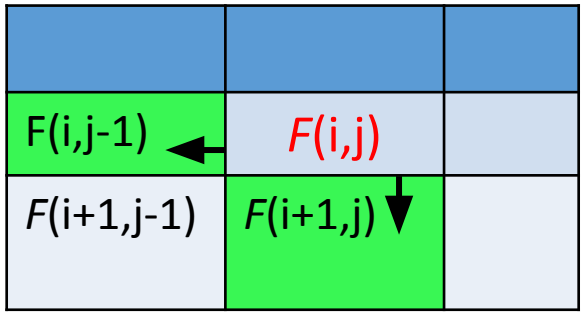
1	1(2)				?
	1	1(2)			
		1	1(2)		
			1	1(2)	
				1	1(2)
					1

Строки длины >2

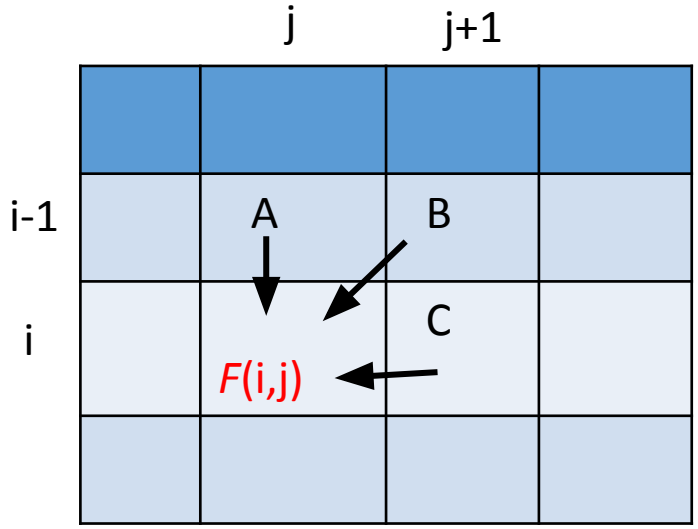
$$i=\overline{1,n-2},\quad j>i$$

$$F[i,j]=\begin{cases}F[i+1,j-1]+2, & if \quad string[i]=string[j]\\ \max\{F[i,k]+F[k+1,j]\}, & if \quad string[i]\neq string[j]\end{cases}$$





Задачи A, B и C являются зависимыми, так как они требуют для своего решения знать длину наибольшего палиндрома для строки string от индекса *i* до *j* включительно.



Приме
р

a s d f s l a

	a	s	d	f	s	l	a
a	**1**	**1**					
s		**1**	**1**				
d			**1**	**1**			
f				**1**	**1**		
s					**1**	**1**	
l						**1**	**1**
a							**1**

$$f[i,i] = 1, i = \overline{1,n}$$

$$f[i,i+1] = \begin{cases} 2, & if \quad string[i] = string[i+1] \\ 1, & if \quad string[i] \neq string[i+1] \end{cases} \quad i = \overline{1,n-1}$$

a s d f s l a

$$f[i,j] = \begin{cases} f[i+1, j-1] + 2, & if \quad string[i] = string[j] \\ max\{f[i+1,j], f[i,j-1]\}, & if \quad string[i] \neq string[j] \end{cases}$$

$$i = \overline{1, n-2}, \quad j > i$$

	a	s	d	f	s	l	a
a	**1**	**1**	**1**				
s		**1**	**1**	**1**			
d			**1**	**1**	**1**		
f				**1**	**1**	**1**	
s					**1**	**1**	**1**
l						**1**	**1**
a							**1**

a s d f s l a

$$f[i,j] = \begin{cases} f[i+1, j-1] + 2, & if \quad string[i] = string[j] \\ max\{f[i+1,j], f[i,j-1]\}, & if \quad string[i] \neq string[j] \end{cases}$$

$$i = \overline{1, n-2}, \quad j > i$$

	a	s	d	f	s	l	a
a	**1**	**1**	**1**	**1**			
s		**1**	**1**	**1**	**3**		
d			**1**	**1**	**1**	**1**	
f				**1**	**1**	**1**	**1**
s					**1**	**1**	**1**
l						**1**	**1**
a							**1**

a s d f s l a

$$f[i,j] = \begin{cases} f[i+1, j-1] + 2, & if \quad string[i] = string[j] \\ max\{f[i+1,j], f[i,j-1]\}, & if \quad string[i] \neq string[j] \end{cases}$$

$$i = \overline{1, n-2}, \quad j > i$$

	a	s	d	f	s	l	a
a	**1**	**1**	**1**	**1**	**3**		
s		**1**	**1**	**1**	**3**	**3**	
d			**1**	**1**	**1**	**1**	**1**
f				**1**	**1**	**1**	**1**
s					**1**	**1**	**1**
l						**1**	**1**
a							**1**

a s d f s l a

$$f[i,j] = \begin{cases} f[i+1, j-1] + 2, & if \quad string[i] = string[j] \\ max\{f[i+1, j], f[i, j-1]\}, & if \quad string[i] \neq string[j] \end{cases}$$

$$i = \overline{1, n-2}, \quad j > i$$

	a	s	d	f	s	l	a
a	**1**	**1**	**1**	**1**	**3**	**3**	
s		**1**	**1**	**1**	**3**	**3**	**3**
d			**1**	**1**	**1**	**1**	**1**
f				**1**	**1**	**1**	**1**
s					**1**	**1**	**1**
l						**1**	**1**
a							**1**

a s d f s l a

$$f[i,j] = \begin{cases} f[i+1, j-1] + 2, & if \quad string[i] = string[j] \\ max\{f[i+1, j], f[i, j-1]\}, & if \quad string[i] \neq string[j] \end{cases}$$

$$i = \overline{1, n-2}, \quad j > i$$

	a	s	d	f	s	l	a
a	**1**	**1**	**1**	**1**	**3**	**3**	**5**
s		**1**	**1**	**1**	**3**	**3**	**3**
d			**1**	**1**	**1**	**1**	**1**
f				**1**	**1**	**1**	**1**
s					**1**	**1**	**1**
l						**1**	**1**
a							**1**

Время работы алгоритма : $O(n^2)$

Восстановление палиндрома

a s d f s l a

	a	s	d	f	s	l	a
a	1	1	1	1	3	3	5
s		1	1	1	3	3	3
d			1	1	1	1	1
f				1	1	1	1
s					1	1	1
l						1	1
a							1

a

a s

a s f

---

a s f s a

Задача 6.
Наибольшая возрастающая подпоследовательность

Longest increasing subsequence, LIS

Задана конечная последовательность чисел (могут быть повторения)

$$X = x_1, x_2, \dots, x_n .$$

Необходимо найти **наибольшую возрастающую подпоследовательность** (или наибольшую строго возрастающую).

X= 0, 2, 9, 1, 9, 6, 7, 22

НВП(X)= 0, 2, 6, 7, 22

Неявное решение задачи (двумерное ДП)

Предположим, что у нас задана последовательность чисел $X = x_1, x_2, \dots, x_n$.

Отсортируем последовательность X по не убыванию, получим последовательность $Y: y_1 \le y_2 \le \dots \le y_n$

(если требуется, чтобы подпоследовательность строго возрастала, то удалим из Y повторяющиеся элементы).

Сделать это можно за время $O(n \cdot log\, n)$.

Тогда $НВП(\boldsymbol{X}) = НОП(\boldsymbol{X}, \boldsymbol{Y})$.

X= 0, 2, 9, 1, 9, 6, 7, 22

Y= 0, 1, 2, 6, 7, 9, 9, 22

		0	1	2	6	7	9	9	22
	0	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
0	**0**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**
2	**0**	**1**	**1**	**2**	**2**	**2**	**2**	**2**	**2**
9	**0**	**1**	**1**	**2**	**2**	**2**	**3**	**3**	**3**
1	**0**	**1**	**2**	**2**	**2**	**2**	**3**	**3**	**3**
9	**0**	**1**	**2**	**2**	**2**	**2**	**3**	**4**	**4**
6	**0**	**1**	**2**	**2**	**3**	**3**	**3**	**4**	**4**
7	**0**	**1**	**2**	**2**	**3**	**4**	**4**	**4**	**4**
22	**0**	**1**	**2**	**2**	**3**	**4**	**4**	**4**	**5**

НВП (0, 2, 9, 1, 9, 6, 7, 22)=(22, 9, 9, 2,0)

Время работы $\Theta(n^2)$.

Требуемая память $\Theta(n^2)$.

Явное решение задачи (одномерное ДП)

Обозначим через *F*[*i*] длину максимальной возрастающей подпоследовательности, которая обязательно заканчивается в элементе x_i.

Рассмотрим некоторую возрастающую подпоследовательность, которая заканчивается элементе x_i. В этой последовательности элемент x_i является последним и остальные элементы этой последовательности не больше, чем он.

Непосредственно перед x_i могут оказаться те из элементов x_1, …, x_i, которые не больше, чем x_i. Для того, чтобы получить последовательность наибольшей длины, заканчивающуюся в x_i нужно, чтобы перед x_i стоял тот элемент $x_j \leq x_i$ (*j*=1, …, *i*) для которого значение $f(x_j)$ наибольшее.

Тогда получаем следующее рекуррентное соотношение:

$$f(i) = \begin{cases} 1, i = 1, \dots, n \\ \max\limits_{\substack{j=1,\dots,i-1 \\ x_j \leq x_i}} f(j) + 1, \ i = 2, \dots, n \end{cases}$$

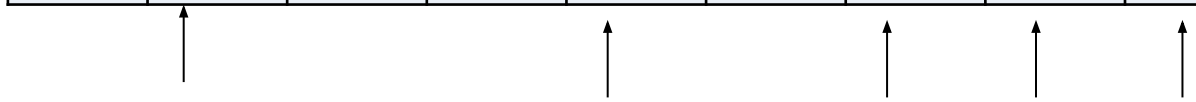
Ответ:

$$\text{НВП}(X_n) = \max_{j=1,\dots,n} f(j)$$

Построить НВП для последовательности:

X= 0, 2, 9, 1, 9, 6, 7, 22, 1

i	1	2	3	4	5	6	7	8	9
X	0	2	9	1	9	6	7	22	1
F	**1**	**2**	**3**	**2**	**4**	**3**	**4**	**5**	**4**



Время работы $\Theta(n^2)$.

Требуемая память $\Theta(n)$.

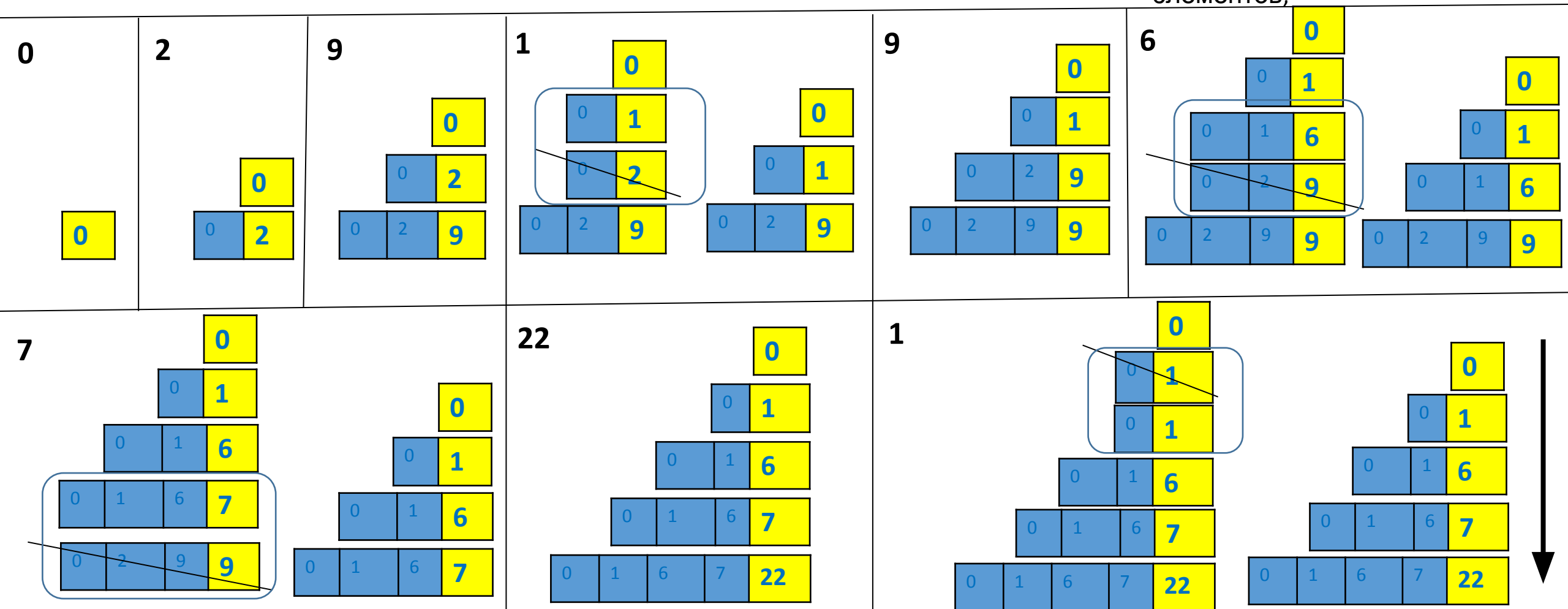
Как восстановить саму НВП (X)?

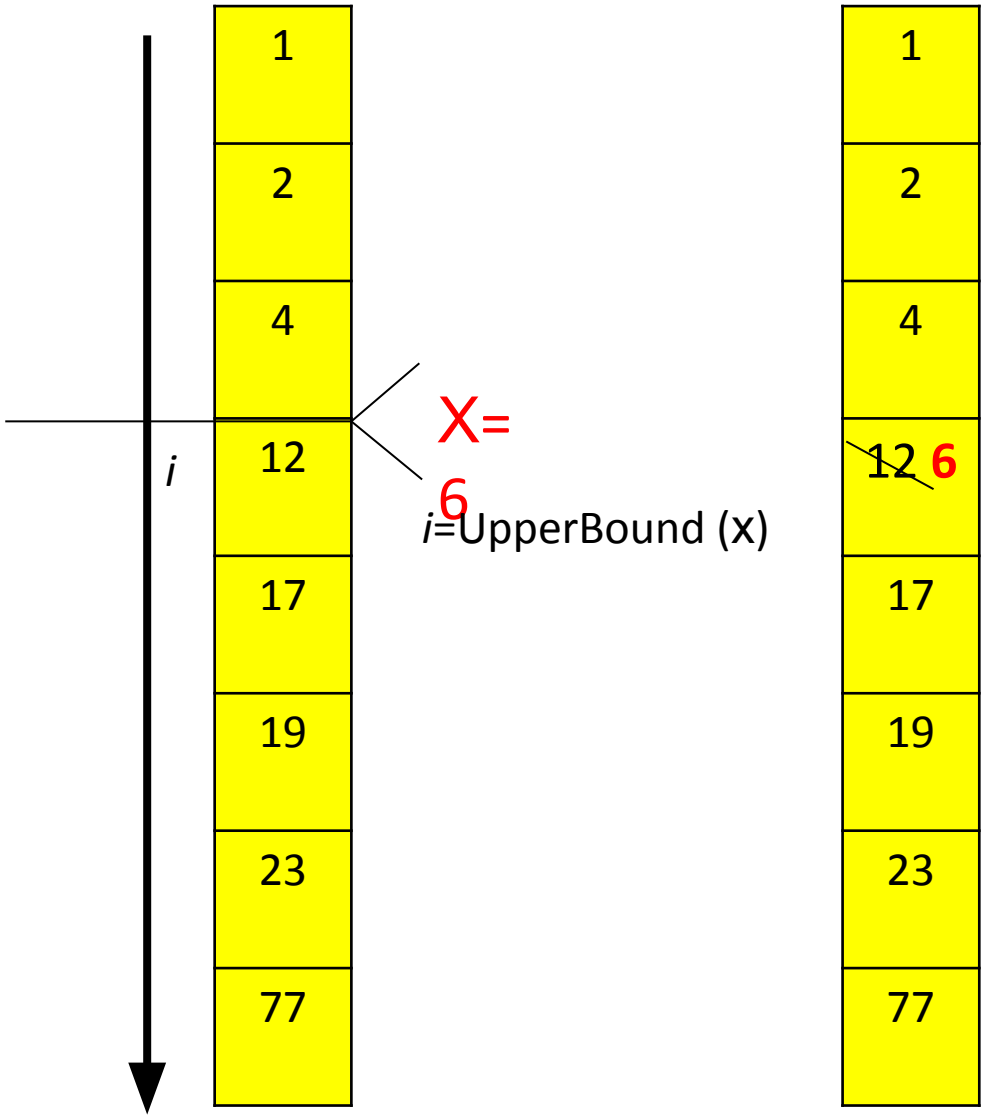
НВП(0, 2, 9, 1, 9, 6, 7, 22, 1)=(0,1,6,7,22)

Можно ли решить эту задачу быстрее, например, за $O(n \cdot \log n)$?

i	1	2	3	4	5	6	7	8	9
X	**0**	**2**	**9**	**1**	**9**	**6**	**7**	**22**	**1**
F	1	2	3	2	4	3	4	5	4

среди подпоследовательностей одинаковой длины оставляем одну – самую **перспективную из них**: ту, к которой можно подсоединить больше элементов;





Время работы алгоритма построения НВП(X):

$$O(n \cdot \log n).$$

Задача 7.
Преобразование строк

вычисление расстояния Левенштейна
(редакционное расстояние)

Заданы две строки, как сравнить их, чтобы определить насколько они похожи?

$$X = x_1, x_2, \dots, x_n$$

$$Y = y_1, y_2, \dots, y_m.$$

Приложения:

- ✔ для исправления ошибок при наборе слова;
- ✔ для сравнения текстовых файлов («символы» – строки файла; «строки» - файлы);
- ✔ в биоинформатике при сравнении аминокислот.

Если строки имеют одинаковую длину,

$$X = x_1, x_2, \dots, x_n$$
$$Y = y_1, y_2, \dots, y_n.$$

то для их сравнения можно использовать следующую метрику: **расстояние Хэмминга**

число позиций, в которых соответствующие символы двух строк одинаковой длины отличаются.

X		л	г	о	р	и	т	м	и	к	а
Y		л	г	о	р	р	т	м	и	к	а

$$d(X, Y) = 2$$

Если расстояние Хэмминга равно 1, то говорят, что строки являются «соседними».

Ричард Уэсли Хэмминг
Richard Wesley Hamming
1915-1998
США, Чикаго

Если строки имеют разную длину:

$$X = x_1, x_2, \dots, x_n$$
$$Y = y_1, y_2, \dots, y_m$$

то для их сравнения можно использовать следующую метрику: расстояние Левенштейна (др. словами редакционное расстояние).

Расстояние Левенштейна для двух строк равно минимальному числу односимвольных «редакторских правок»:

1) замена символа в строке X (**R** -*replace*) ;
2) удаление символа из строки X (**D** - *delete*) ;
3) вставка символа в строку X (**I** - insert) .

в результате которых строка X преобразуется с Y.

Владимир Иосифович Левенштейн
1935-2017
СССР (Россия), Москва
доктор физ.-мат.наук

R −*replace*
D − *delete*
I − insert

X=лгорритмиккк

Y=алгоритмика

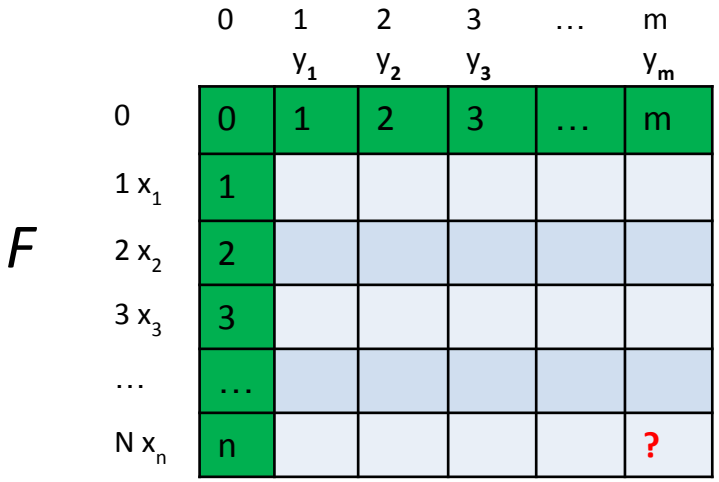
X		л	г	о	р	р	и	т	м	и	к	к	к
Y	а	л	г	о	р		и	т	м	и	к		а
	I					D						D	R

d(X,Y)=4

Задачу можно решить динамическим программированием

Обозначим через $F[i,j]$
расстояние Левенштейна для двух префиксов:

$$X_i = x_1, x_2, \dots, x_i$$
$$Y_j = y_1, y_2, \dots, y_j.$$



Граничные условия (один из префиксов пустой):

$$f[0,j] = j, j = 0, \dots, m \ (\mathbf{I})$$
$$f[i,0] = i, i = 0, \dots, n \quad (\mathbf{D})$$

$X_i = x_1, x_2, \dots, x_{i-1}, x_i$

$Y_j = y_1, y_2, \dots, y_{j-1}, y_j$

предположим, что x_i **был удален**;
тогда, чтобы получить Y_j нужно преобразовать X_{i-1} в
а потом удалить x_i

предположим, что y_j **был добавлен**;
тогда нужно сначала преобразовать X_i в Y_{j-1},
потом добавить y_j

предположим, что x_i не удаляли, а y_j не добавляли;
случай 1: если $x_i = y_j$, то
т. к. x_i не удаляли, а на его месте надо получить y_j,
то преобразуем оптимально X_{i-1} в Y_{j-1}, а
элемент $y_j = x_i$ уже стоит на своем месте;
случай 2:
если $x_i \neq y_j$, то
т. к. x_i не удаляли, а y_j не добавляли, то
единственный способ - преобразовать оптимально
X_{i-1} в Y_{j-1} и потом заменить x_i на y_j

$$f[i,j] = \min\{\ f[i-1,j]+1,\quad f[i,j-1]+1,\quad f[i-1,j-1]+\delta(x[i],y[j])\ \}$$

Предположим, что одиночные операции вставки, удаления и замены имеют разную стоимость:

$p(\boldsymbol{D})$ - стоимость удаления одного символа
$p(\boldsymbol{I})$ - стоимость вставки одного символа
$p(\boldsymbol{R})$ - стоимость замены одного символа на другой

$X_i = x_1, x_2, \dots, x_{i-1}, x_i$

$Y_j = y_1, y_2, \dots, y_{j-1}, y_j$

$p(\boldsymbol{D})$ - стоимость удаления одного символа

$p(\boldsymbol{I})$ - стоимость вставки одного символа

$p(\boldsymbol{R})$ - стоимость замены одного символа на другой

предположим, что x_i **был удален**;
тогда, чтобы получить Y_j нужно преобразовать X_{i-1} в Y_j, а потом удалить x_i со штрафом $p(\boldsymbol{D})$

предположим, что y_j **был добавлен**;
тогда нужно преобразовать X_i в Y_{j-1}, потом добавить y_j штрафом $p(\boldsymbol{I})$

предположим, что одиночные операции удаления x_i или добавления y_j не выполнялись;

случай 1: если $x_i = y_j$, то

т. к. x_i не удаляли, а на его месте надо получить y_j, то преобразуем оптимально X_{i-1} в Y_{j-1}, а элемент y_j(= x_i) уже стоит на своем месте;

случай 2: если $x_i \neq y_j$, то

нужно преобразовать оптимально X_{i-1} в Y_{j-1}, а потом заменить x_i на y_j, выбирая наилучший из двух способов:

(1)непосредственная замена x_i на y_j со штрафом $p(\boldsymbol{R})$;

(2) две операции: удаление x_i за $p(\boldsymbol{D})$ и добавление y_j за $p(\boldsymbol{I})$;

$$f[i,j] = min\{\, f[i-1,j] + p(\boldsymbol{D}),\ f[i,j-1] + p(\boldsymbol{I}),\ f[i-1,j-1] + \delta(x[i], y[j]) \cdot \min(p(\boldsymbol{R}), p(\boldsymbol{D}) + p(\boldsymbol{I}))\},$$

$$\delta(x[i], y[j]) = \begin{cases} 1, \text{если } x[i] \neq y[j] \\ 0, \text{если } x[i] = y[j] \end{cases}$$

Проанализируем случай 2, в котором рассматривались два способа замены x_i на y_j , и покажем, что справедлива более упрощённая формула:

$$f[i,j] = min\{ f[i-1,j] + p(D), f[i,j-1] + p(I), f[i-1,j-1] + \delta(x[i], y[j]) \cdot p(R)\}, \delta(x[i], y[j]) = \begin{cases} 1, \text{если } x[i] \neq y[j] \\ 0, \text{если } x[i] = y[j] \end{cases}$$

Действительно, из приведенной выше формулы следуют два неравенства:

$$f[i,j] \leq f[i-1,j] + p(\boldsymbol{D}), \qquad (1)$$

$$f[i,j] \leq f[i,j-1] + p(\boldsymbol{I}). \qquad (2)$$

Из (1) и (2) следует, что всегда будет выполняться неравенство:

$$f[i,j] \leq [\text{н} - \text{во } 1] \leq f[i-1,j] + p(\boldsymbol{D}) \leq [\text{н} - \text{во } 2] \leq (f[i-1,j-1] + p(\boldsymbol{I})) + p(\boldsymbol{D}) = f[i-1,j-1] + p(\boldsymbol{I}) + p(\boldsymbol{D})$$

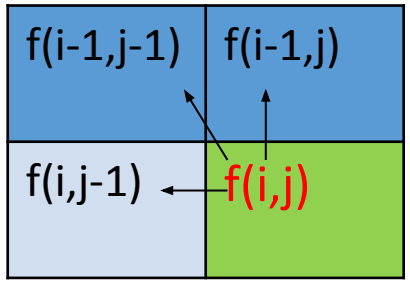
Следовательно, случай замены x_i на y_j через две операции: удаление x_i за $p(\boldsymbol{D})$ и добавление y_j за $p(\boldsymbol{I})$ можно не рассматривать отдельно, так как он уже будет учтён.

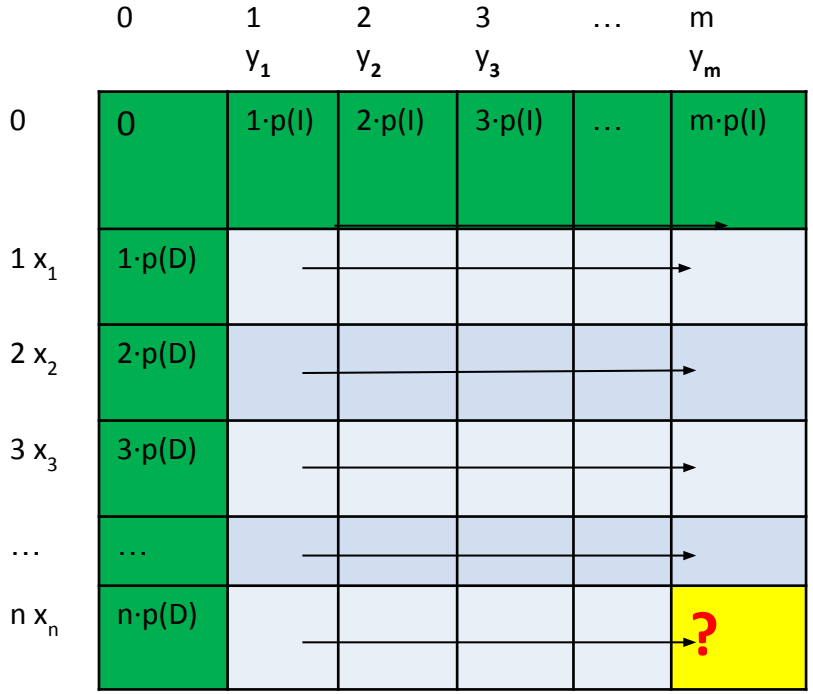
Обоснование корректности упрощенной формулы Кощенко Владиславом, 2022 год

ФПМИ БГУ

$$f[i,j] = min\{\, f[i-1,j] + p(\boldsymbol{D}), f[i,j-1] + p(\boldsymbol{I}), f[i-1,j-1] + \delta(\mathrm{x}[i], y[j]) \cdot p(\boldsymbol{R})\},$$

$$\delta(\mathrm{x}[i], y[j]) = \begin{cases} 1, \text{если } \mathrm{x}[i] \neq y[j] \\ 0, \text{если } \mathrm{x}[i] = y[j] \end{cases}$$





Время работы алгоритма: $\Theta(n \cdot m)$

Требуемая память: $\Theta(n \cdot m)$

ФПМИ БГУ

Приме

р X=лгорритмиккк

Y=алгоритмика

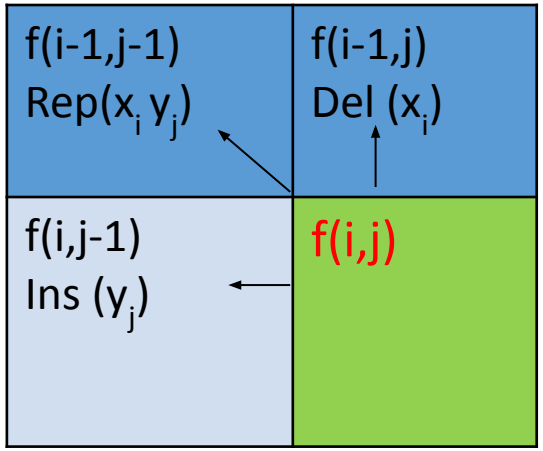
$p(\boldsymbol{D})$ =1

$p(\boldsymbol{I})$ =1

$p(\boldsymbol{R})$ =1

Рекуррентное соотношение:

$$f[i,j] = \min\begin{Bmatrix} f[i-1,j]+1 \\ f[i,j-1] \ +1 \\ f[i-1,j-1]+\delta(\mathrm{x}[i],y[j]) \end{Bmatrix}$$



		а	л	г	о	р	и	т	м	и	к	а
	0	1	2	3	4	5	6	7	8	9	10	11
л	1	1	1	2	3	4	5	6	7	8	9	10
г	2											
о	3											
р	4											
р	5											
и	6											
т	7											
м	8											
и	9											
к	10											
к	11											
к	12											

X=лгорритмиккк
Y=алгоритмика

$$f[i,j] = \min \begin{cases} f[i-1,j]+1 \\ f[i,j-1]+1 \\ f[i-1,j-1]+\delta(\mathrm{x}[i],y[j]) \end{cases}$$

f(i-1,j-1) Rep(x_i y_j)	f(i-1,j) Del (x_i)
f(i,j-1) Ins (y_j)	f(i,j)

		а	л	г	о	р	и	т	м	и	к	а
	0	1	2	3	4	5	6	7	8	9	10	11
л	1	1	1	2	3	4	5	6	7	8	9	10
г	2	2	2	1	2	3	4	5	6	7	8	9
о	3											
р	4											
р	5											
и	6											
т	7											
м	8											
и	9											
к	10											
к	11											
к	12											

X=лгорритмиккк
Y=алгоритмика

$$f[i,j] = \min \begin{cases} f[i-1,j]+1 \\ f[i,j-1]+1 \\ f[i-1,j-1]+\delta(\mathrm{x}[i],y[j]) \end{cases}$$

f(i-1,j-1) Rep(x_i y_j)	f(i-1,j) Del (x_i)
f(i,j-1) Ins (y_j)	f(i,j)

		а	л	г	о	р	и	т	м	и	к	а
	0	1	2	3	4	5	6	7	8	9	10	11
л	1	1	1	2	3	4	5	6	7	8	9	10
г	2	2	2	1	2	3	4	5	6	7	8	9
о	3	3	3	2	1	2	3	4	5	6	7	8
р	4											
р	5											
и	6											
т	7											
м	8											
и	9											
к	10											
к	11											
к	12											

X=лгорритмиккк
Y=алгоритмика

$$f[i,j] = \min \begin{Bmatrix} f[i-1,j]+1 \\ f[i,j-1]+1 \\ f[i-1,j-1]+\delta(\mathrm{x}[i],y[j]) \end{Bmatrix}$$

f(i-1,j-1) Rep(x_i y_j)	f(i-1,j) Del (x_i)
f(i,j-1) Ins (y_j)	f(i,j)

ФПМИ БГУ

		а	л	г	о	р	и	т	м	и	к	а
	0	1	2	3	4	5	6	7	8	9	10	11
л	1	1	1	2	3	4	5	6	7	8	9	10
г	2	2	2	1	2	3	4	5	6	7	8	9
о	3	3	3	2	1	2	3	4	5	6	7	8
р	4	4	4	3	2	1	2	3	4	5	6	7
р	5											
и	6											
т	7											
м	8											
и	9											
к	10											
к	11											
к	12											

X=лгорритмиккк
Y=алгоритмика

$$f[i,j] = \min \begin{Bmatrix} f[i-1,j]+1 \\ f[i,j-1]+1 \\ f[i-1,j-1]+\delta(\mathrm{x}[i],y[j]) \end{Bmatrix}$$

f(i-1,j-1) Rep(x_i y_j)	f(i-1,j) Del (x_i)
f(i,j-1) Ins (y_j)	f(i,j)

		а	л	г	о	р	и	т	м	и	к	а
	0	1	2	3	4	5	6	7	8	9	10	11
л	1	1	1	2	3	4	5	6	7	8	9	10
г	2	2	2	1	2	3	4	5	6	7	8	9
о	3	3	3	2	1	2	3	4	5	6	7	8
р	4	4	4	3	2	1	2	3	4	5	6	7
р	5	5	5	4	3	2	2	3	4	5	6	7
и	6											
т	7											
м	8											
и	9											
к	10											
к	11											
к	12											

X=лгорритмиккк
Y=алгоритмика

$$f[i,j] = \min \begin{Bmatrix} f[i-1,j]+1 \\ f[i,j-1]+1 \\ f[i-1,j-1]+\delta(\mathrm{x}[i],y[i]) \end{Bmatrix}$$

f(i-1,j-1) Rep(x_i y_j)	f(i-1,j) Del (x_i)
f(i,j-1) Ins (y_j)	f(i,j)

		а	л	г	о	р	и	т	м	и	к	а
	0	1	2	3	4	5	6	7	8	9	10	11
л	1	1	1	2	3	4	5	6	7	8	9	10
г	2	2	2	1	2	3	4	5	6	7	8	9
о	3	3	3	2	1	2	3	4	5	6	7	8
р	4	4	4	3	2	1	2	3	4	5	6	7
р	5	5	5	4	3	2	2	3	4	5	6	7
и	6	6	6	5	4	3	2	3	4	4	5	6
т	7											
м	8											
и	9											
к	10											
к	11											
к	12											

X=лгорритмиккк
Y=алгоритмика

$$f[i,j] = \min \begin{Bmatrix} f[i-1,j]+1 \\ f[i,j-1]+1 \\ f[i-1,j-1]+\delta(\mathrm{x}[i],y[i]) \end{Bmatrix}$$

f(i-1,j-1) Rep(x_i y_j)	f(i-1,j) Del (x_i)
f(i,j-1) Ins (y_j)	f(i,j)

		а	л	г	о	р	и	т	м	и	к	а
	0	1	2	3	4	5	6	7	8	9	10	11
л	1	1	1	2	3	4	5	6	7	8	9	10
г	2	2	2	1	2	3	4	5	6	7	8	9
о	3	3	3	2	1	2	3	4	5	6	7	8
р	4	4	4	3	2	1	2	3	4	5	6	7
р	5	5	5	4	3	2	2	3	4	5	6	7
и	6	6	6	5	4	3	2	3	4	4	5	6
т	7	7	7	6	5	4	3	2	3	4	5	6
м	8											
и	9											
к	10											
к	11											
к	12											

X=лгорритмиккк
Y=алгоритмика

$$f[i,j] = \min\begin{Bmatrix} f[i-1,j]+1 \\ f[i,j-1]+1 \\ f[i-1,j-1]+\delta(\mathrm{x}[i],y[j]) \end{Bmatrix}$$

f(i-1,j-1) Rep(x_i y_j)	f(i-1,j) Del (x_i)
f(i,j-1) Ins (y_j)	f(i,j)

		а	л	г	о	р	и	т	м	и	к	а
	0	1	2	3	4	5	6	7	8	9	10	11
л	1	1	1	2	3	4	5	6	7	8	9	10
г	2	2	2	1	2	3	4	5	6	7	8	9
о	3	3	3	2	1	2	3	4	5	6	7	8
р	4	4	4	3	2	1	2	3	4	5	6	7
р	5	5	5	4	3	2	2	3	4	5	6	7
и	6	6	6	5	4	3	2	3	4	4	5	6
т	7	7	7	6	5	4	3	2	3	4	5	6
м	8	8	8	7	6	5	4	3	2	3	4	5
и	9											
к	10											
к	11											
к	12											

X=лгорритмиккк
Y=алгоритмика

$$f[i,j] = \min \begin{Bmatrix} f[i-1,j]+1 \\ f[i,j-1]+1 \\ f[i-1,j-1]+\delta(\mathrm{x}[i],y[j]) \end{Bmatrix}$$

f(i-1,j-1) Rep(x_i y_j)	f(i-1,j) Del (x_i)
f(i,j-1) Ins (y_j)	f(i,j)

		а	л	г	о	р	и	т	м	и	к	а
	0	1	2	3	4	5	6	7	8	9	10	11
л	1	1	1	2	3	4	5	6	7	8	9	10
г	2	2	2	1	2	3	4	5	6	7	8	9
о	3	3	3	2	1	2	3	4	5	6	7	8
р	4	4	4	3	2	1	2	3	4	5	6	7
р	5	5	5	4	3	2	2	3	4	5	6	7
и	6	6	6	5	4	3	2	3	4	4	5	6
т	7	7	7	6	5	4	3	2	3	4	5	6
м	8	8	8	7	6	5	4	3	2	3	4	5
и	9	9	9	8	7	6	5	4	4	2	3	4
к	10											
к	11											
к	12											

X=лгорритмиккк
Y=алгоритмика

$$f[i,j] = \min \begin{Bmatrix} f[i-1,j]+1 \\ f[i,j-1]+1 \\ f[i-1,j-1]+\delta(\mathrm{x}[i],y[j]) \end{Bmatrix}$$

f(i-1,j-1) Rep(x_i y_j)	f(i-1,j) Del (x_i)
f(i,j-1) Ins (y_j)	f(i,j)

		а	л	г	о	р	и	т	м	и	к	а
	0	1	2	3	4	5	6	7	8	9	10	11
л	1	1	1	2	3	4	5	6	7	8	9	10
г	2	2	2	1	2	3	4	5	6	7	8	9
о	3	3	3	2	1	2	3	4	5	6	7	8
р	4	4	4	3	2	1	2	3	4	5	6	7
р	5	5	5	4	3	2	2	3	4	5	6	7
и	6	6	6	5	4	3	2	3	4	4	5	6
т	7	7	7	6	5	4	3	2	3	4	5	6
м	8	8	8	7	6	5	4	3	2	3	4	5
и	9	9	9	8	7	6	5	4	4	2	3	4
к	10	10	10	9	8	7	6	5	5	3	2	3
к	11											
к	12											

X=лгорритмиккк
Y=алгоритмика

$$f[i,j] = \min \begin{Bmatrix} f[i-1,j]+1 \\ f[i,j-1]+1 \\ f[i-1,j-1]+\delta(\mathrm{x}[i],y[j]) \end{Bmatrix}$$

f(i-1,j-1) Rep(x_i y_j)	f(i-1,j) Del (x_i)
f(i,j-1) Ins (y_j)	f(i,j)

ФПМИ БГУ

		а	л	г	о	р	и	т	м	и	к	а
	0	1	2	3	4	5	6	7	8	9	10	11
л	1	1	1	2	3	4	5	6	7	8	9	10
г	2	2	2	1	2	3	4	5	6	7	8	9
о	3	3	3	2	1	2	3	4	5	6	7	8
р	4	4	4	3	2	1	2	3	4	5	6	7
р	5	5	5	4	3	2	2	3	4	5	6	7
и	6	6	6	5	4	3	2	3	4	4	5	6
т	7	7	7	6	5	4	3	2	3	4	5	6
м	8	8	8	7	6	5	4	3	2	3	4	5
и	9	9	9	8	7	6	5	4	4	2	3	4
к	10	10	10	9	8	7	6	5	5	3	2	3
к	11	11	11	10	9	8	7	6	6	4	3	3
к	12											

X=лгорритмиккк
Y=алгоритмика

$$f[i,j] = \min \begin{Bmatrix} f[i-1,j]+1 \\ f[i,j-1]+1 \\ f[i-1,j-1]+\delta(\mathrm{x}[i],y[i]) \end{Bmatrix}$$

f(i-1,j-1) Rep(x_i y_j)	f(i-1,j) Del (x_i)
f(i,j-1) Ins (y_j)	f(i,j)

		а	л	г	о	р	и	т	м	и	к	а
	0	1	2	3	4	5	6	7	8	9	10	11
л	1	1	1	2	3	4	5	6	7	8	9	10
г	2	2	2	1	2	3	4	5	6	7	8	9
о	3	3	3	2	1	2	3	4	5	6	7	8
р	4	4	4	3	2	1	2	3	4	5	6	7
р	5	5	5	4	3	2	2	3	4	5	6	7
и	6	6	6	5	4	3	2	3	4	4	5	6
т	7	7	7	6	5	4	3	2	3	4	5	6
м	8	8	8	7	6	5	4	3	2	3	4	5
и	9	9	9	8	7	6	5	4	4	2	3	4
к	10	10	10	9	8	7	6	5	5	3	2	3
к	11	11	11	10	9	8	7	6	6	4	3	3
к	12	12	12	11	10	9	8	7	7	6	4	4

X=лгорритмиккк
Y=алгоритмика

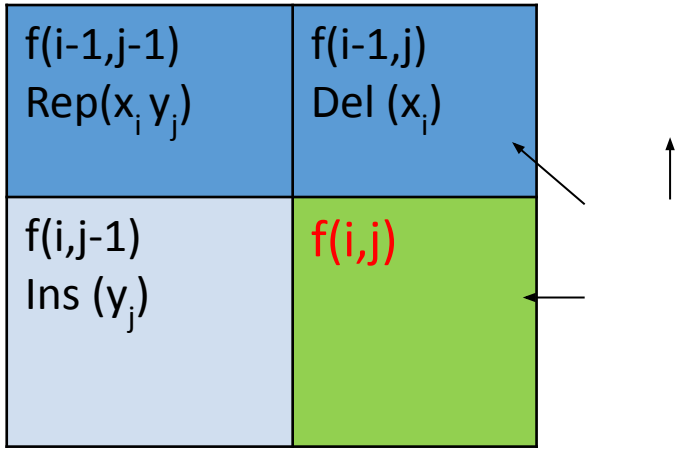
$$f[i,j] = \min \begin{Bmatrix} f[i-1,j]+1 \\ f[i,j-1]+1 \\ f[i-1,j-1]+\delta(\mathrm{x}[i],y[j]) \end{Bmatrix}$$

f(i-1,j-1) Rep(x_i y_j)	f(i-1,j) Del (x_i)
f(i,j-1) Ins (y_j)	f(i,j)

Как восстановить редакторские правки?

		а	л	г	о	р	и	т	м	и	к	а
	0	1	2	3	4	5	6	7	8	9	10	11
л	1	1	1	2	3	4	5	6	7	8	9	10
г	2	2	2	1	2	3	4	5	6	7	8	9
о	3	3	3	2	1	2	3	4	5	6	7	8
р	4	4	4	3	2	1	2	3	4	5	6	7
р	5	5	5	4	3	2	2	3	4	5	6	7
и	6	6	6	5	4	3	2	3	4	4	5	6
т	7	7	7	6	5	4	3	2	3	4	5	6
м	8	8	8	7	6	5	4	3	2	3	4	5
и	9	9	9	8	7	6	5	4	4	2	3	4
к	10	10	10	9	8	7	6	5	5	3	2	3
к	11	11	11	10	9	8	7	6	6	4	3	3
к	12	12	12	11	10	9	8	7	7	6	4	4

X=лгорритмиккк
Y=алгоритмика



Задания

БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

Выполнить в системе Insight Runner следующие общие задачи.

Тема. Динамическое программирование

0.1 Порядок перемножения матриц

0.2 Единицы - число сочетаний из n по k

0.3. Единицы

6. Строго возрастающая без разрывов последовательнсть

20. Палиндром

25. Преобразование строк

69. Кувшинки

Спасибо за внимание!