БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

Методы разработки эффективных алгоритмов

Метод «разделяй и властвуй»

Динамическое программирование

©ДМА ФПМИ Соболевская Е.П., 2021 год

Метод «разделяй и властвуй»

«Разделяй и властвуй»

1. ***«Разделение»***

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

2. ***«Покорение»***

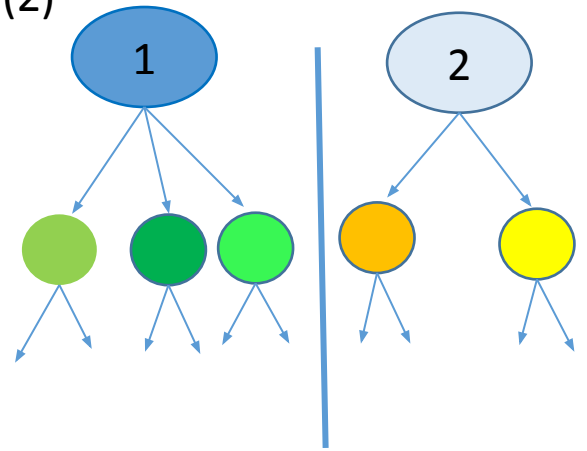
Каждая подзадача решается отдельно (рекурсивным методом). Когда объем возникающих подзадач достаточно мал, то подзадачи решаются непосредственно.

3. ***«Комбинирование»***

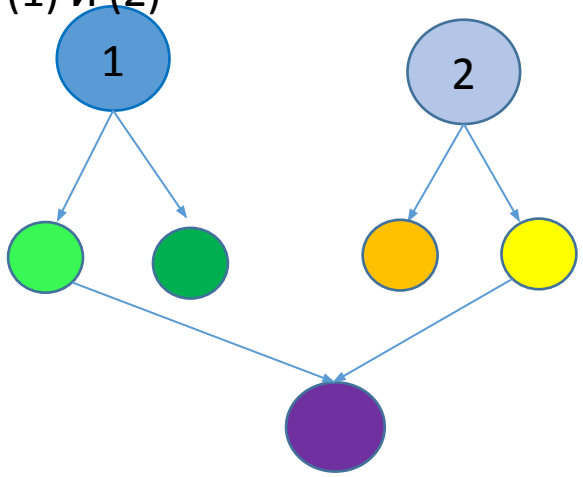
Из отдельных решений подзадач строится решение исходной задачи.

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

независимые задачи (1) и (2)



зависимые задачи (1) и (2)



«Разделяй и властвуй». 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

«Разделяй и властвуй»

«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.

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

Решение на основе принципа «разделяй и властвуй»

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

Последовательный поиск

$$2n - 3$$

ФПМИ БГУ

Сортировка массива слиянием

«Разделяй и властвуй»

«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)
```

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

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

Быстрая сортировка массива Ч. Хоара

«Разделяй и властвуй»

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

«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.

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

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

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 Фиббоначи 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.

Динамическое программирование

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.*

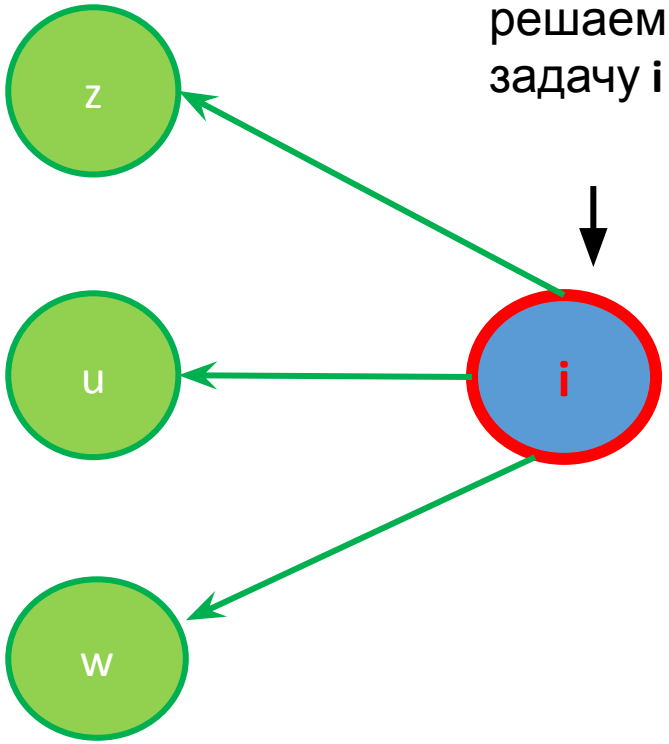
Так как возникающие подзадачи являются зависимыми, то данная техника находит своё применение, когда все нужные значения оптимальных решений подзадач помещаются в память. Вычисление идет от малых подзадач к большим, и ответы запоминаются в таблице, что позволяет исключить повторное решение задачи.

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

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»



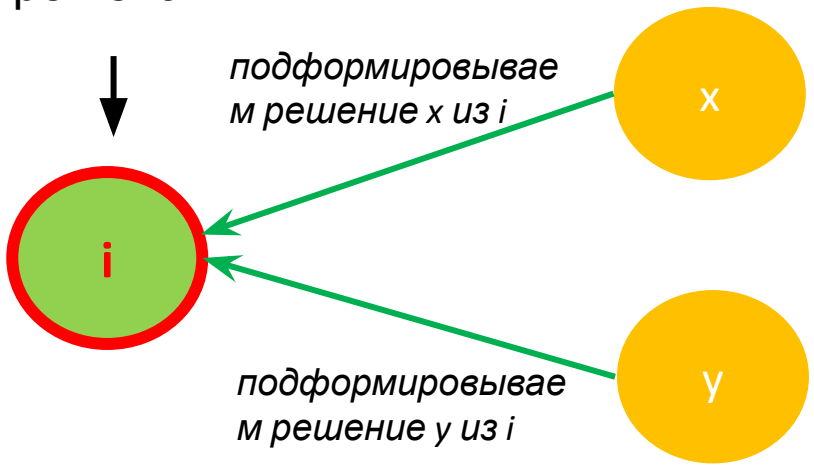
решаем задачу **i**

решённые подзадачи z, u, w

$$f(i)=G(f(z)+f(u)+f(w))$$

ДП «вперед»

задача **i** уже решена



подформировывае м решение x из i

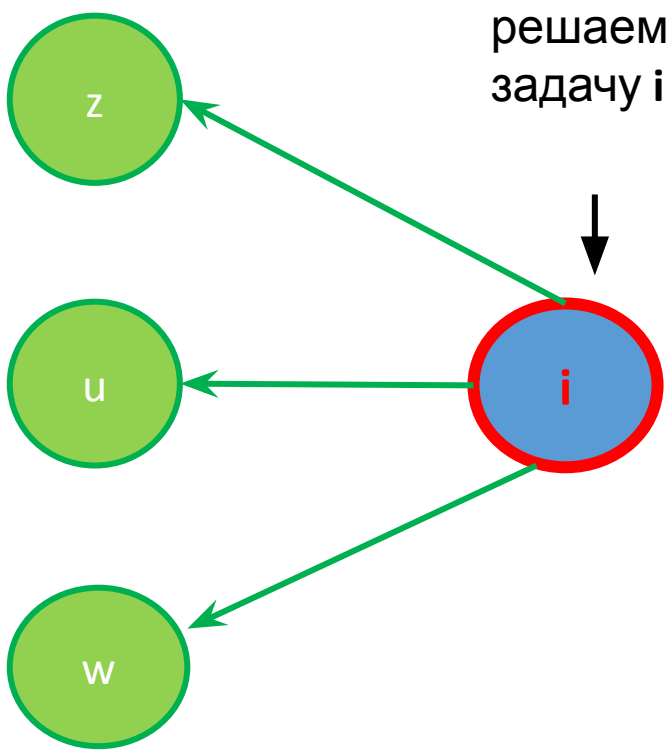
подформировывае м решение y из i

еще НЕ решённые подзадачи x и y (зависимые)

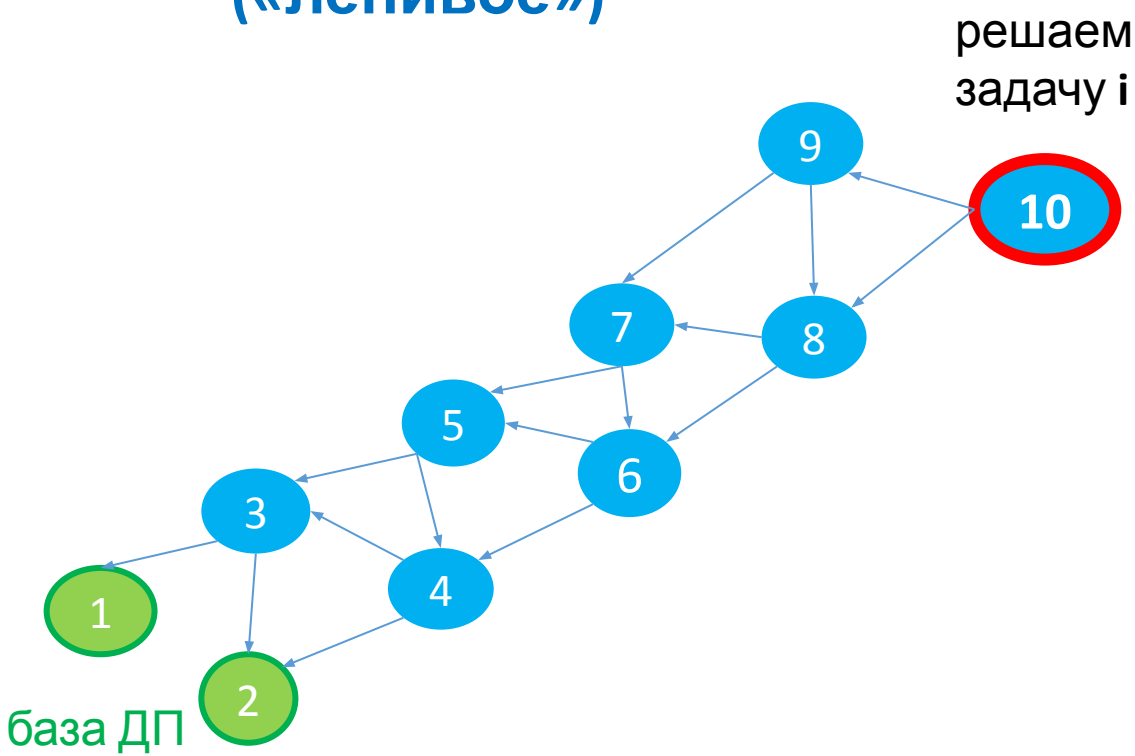
$$f(x)=G(f(x),f(i))$$

$$f(y)=G(f(y),f(i))$$

ДП «назад»



ДП «назад» («ленивое»)

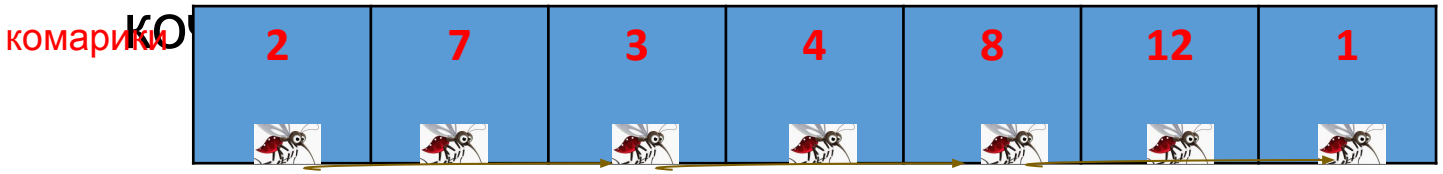


Задача 1. Лягушка

Заданы n кочек. Лягушка сидит на первой кочке. На каждой кочке сидят комарики, известно их число.

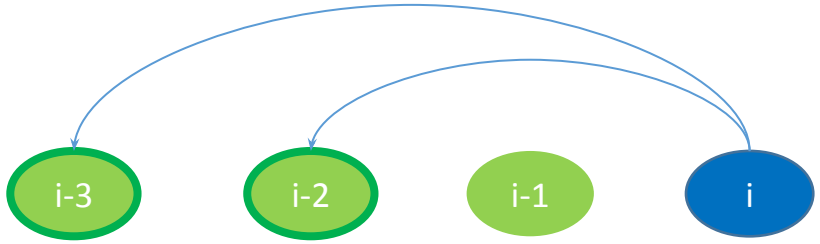
За один прыжок лягушка может прыгнуть на 2 или 3 кочки вперёд. Оказавшись на кочке, лягушка скушает всех комариков, которые сидели там.

Необходимо определить максимальное число комариков, которые скушает лягушка, которой обязательно надо приземлиться на последней ко



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.

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

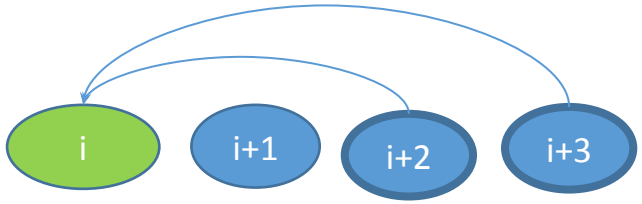
ДП назад (одномерное):

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

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):



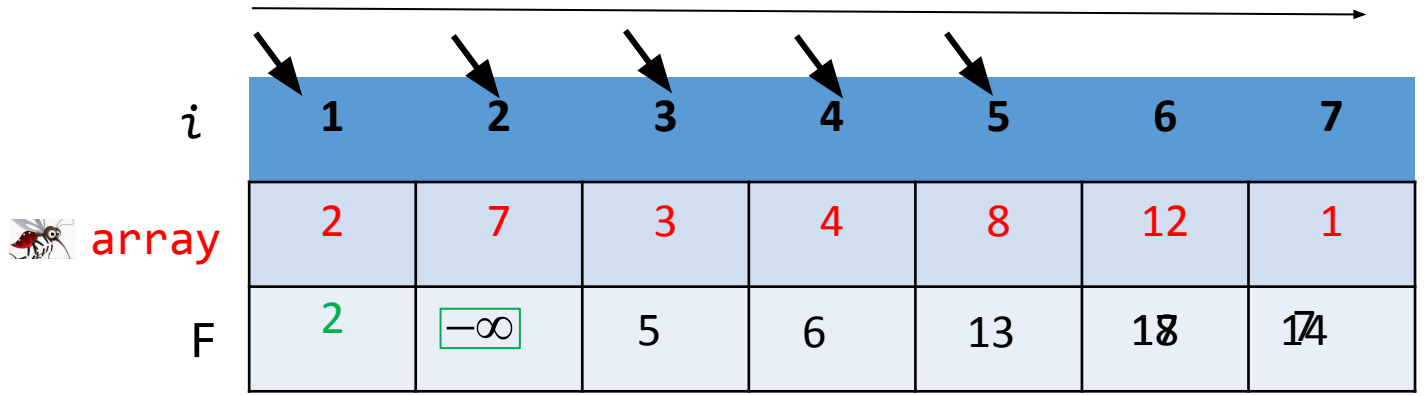
Обозначения:

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

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

ДП вперёд (одномерное):

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



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. Необходимо определить количество способов, для того, чтобы расставить *j* единиц в строке длины *n (j≤n)*.

1									n
	1	1		1	…	1			1

Количество способов можно посчитать комбинаторно:

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

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 100.

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:

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

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

Малая теорема Ферма

Если p – простое число, *a* – целое число, которое не делится на *p*, то

$$a^{p-1} \equiv 1 \pmod p$$

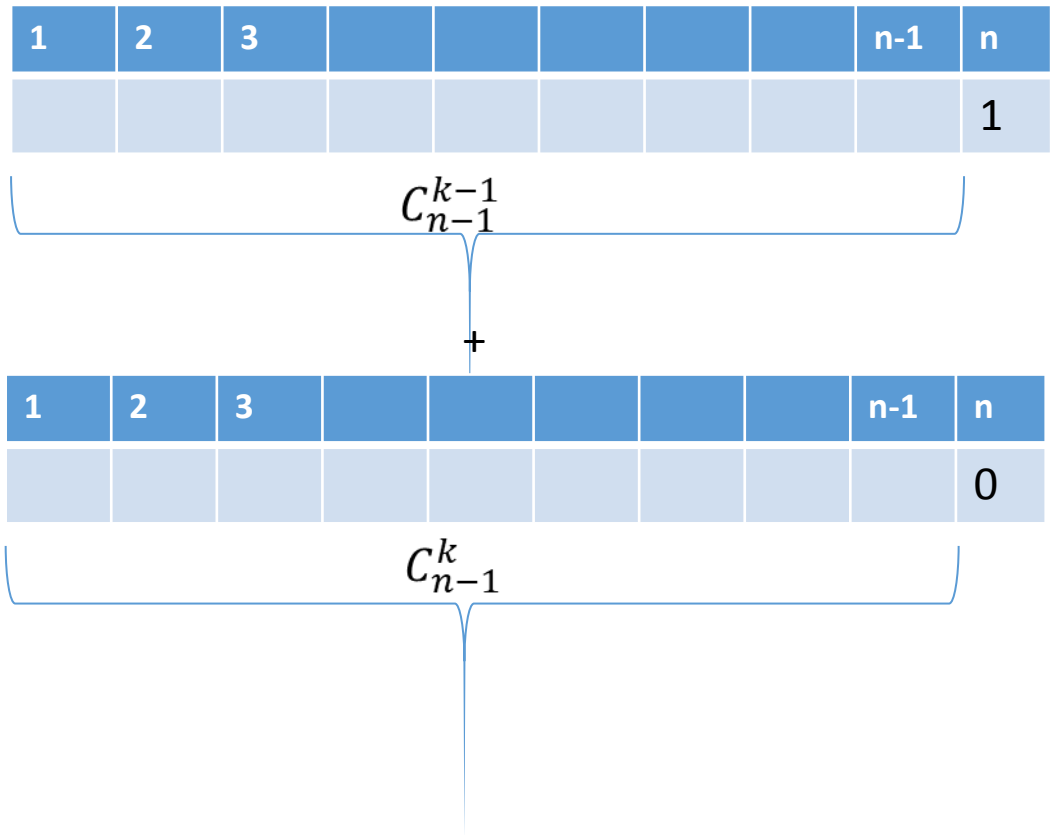
Следствие из малой теоремы Ферма (a – целое, p – простое число):

$$a^{p-2} \equiv \frac{1}{a} \pmod p$$

другими словами: $\frac{1}{a} \,\%\, p = a^{p-2} \,\%\, p$

$$C_n^k = ???$$

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

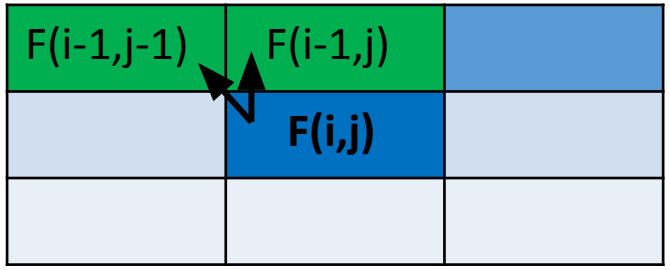


ДП назад (двумерное):

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

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

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



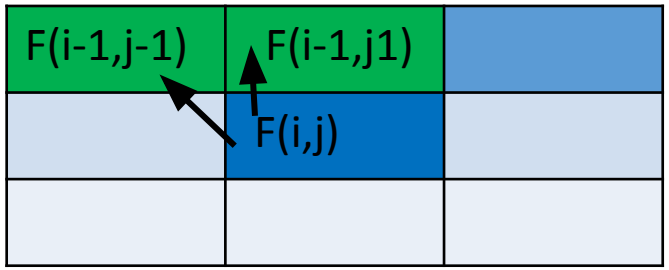
длина строки

число единиц

	0	1	2	3	…
0	1				
1	1	1			
2	1		1		
3	1			1	
…	1				1

ДП назад (двумерное):

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



	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:

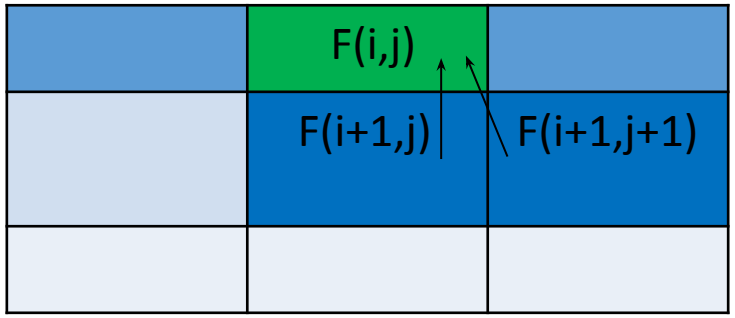
$$O(n \cdot k) = O(n^2)$$

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*.

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



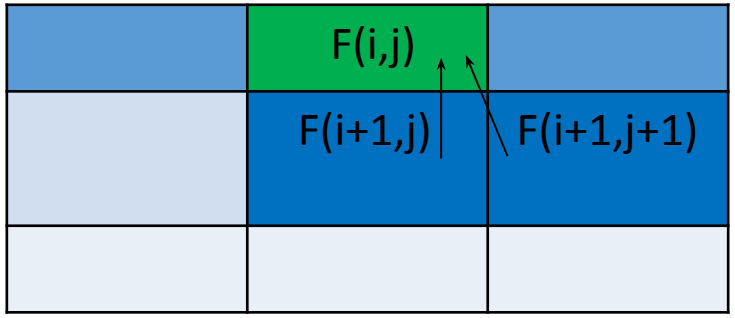
длина строки

число единиц

	0	1	2	3	…
0	1				
1					
2					
3					
…					

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:

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

Задача расстановки единиц

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

Время работы алгоритма, основанного на методе динамического программирования:

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

Количество способов можно посчитать и комбинаторно, но при больших значениях *n* и *k* итоговое значение может уже не помещается 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 100.

Задача 3. Оптимального перемножения группы матриц

Заданы s матриц:

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

$$A_i - n_i \times m_i$$

$$m_i = n_{i+1}, \forall i = \overline{1, s-1}$$

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 и аналитическая формулы для C

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

$$C_4 = \begin{pmatrix} C_0 \\ \cdot \\ C_3 \end{pmatrix} + \begin{pmatrix} C_1 \\ \cdot \\ C_2 \end{pmatrix} + \begin{pmatrix} C_2 \\ \cdot \\ C_1 \end{pmatrix} + \begin{pmatrix} C_3 \\ \cdot \\ C_0 \end{pmatrix}$$

$1 \quad 1 \quad 2 \quad 5$

$5 \quad 2 \quad 1 \quad 1$

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

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

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

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

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 $C_{i/2}$ число Каталана.

	0	1	2	3	4	5
C_n	1	1	2	5	14	42

Задача оптимального перемножения группы матриц

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

$$A_i - n_i \times m_i$$

$$m_i = n_{i+1}, \forall i = \overline{1, s-1}$$

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

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

Метод динамического программирования позволит решить задачу за время :

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

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

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

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

$$A_i - n_i \times m_i$$

$$m_i = n_{i+1}, \forall i = \overline{1, s-1}$$

На последнем этапе, когда формируется 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: $i \le k < j$

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

$$A[n_i \times m_k] \quad \cdot \quad A[n_{k+1} \times m_j]$$

Так как у нас оптимизационная задача, то перемножать матрицы
надо за минимально возможно число операций умножения.

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

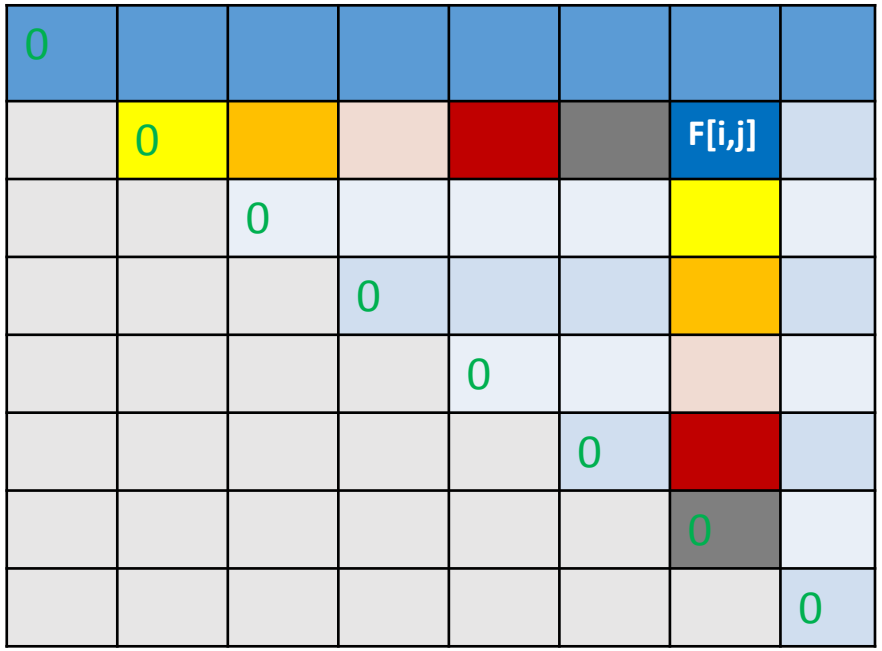
$$A_{k+1} \cdot A_{k+2} \ldots \cdot A_j$$

Справедливо следующее рекуррентное соотношение:

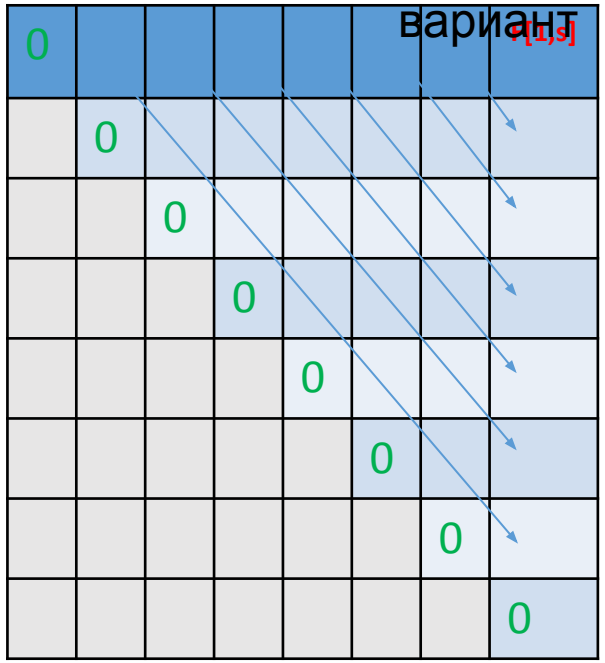
$$\begin{cases} F[i,i] = 0, \quad i = \overline{1,s} \\ F[i,i+1] = n_i \cdot m_i \cdot m_{i+1}, \quad i = \overline{1,s-1} \\ F[i,j] = \min_{i \le k < j} \{F[i,k] + F[k+1,j] + n_i \cdot m_k \cdot m_j\}, \quad i = \overline{1,s}, \quad j > i \end{cases}$$

$$(A_i \cdot \ldots \cdot A_k) \cdot (A_{k+1} \cdot \ldots \cdot A_j)$$

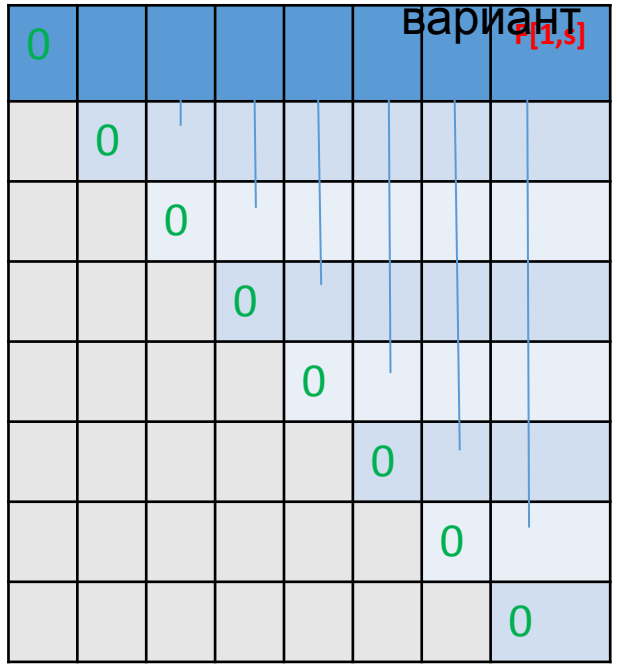
$$A[n_i \times m_k] \cdot A[n_{k+1} \times m_j]$$



1 вариант

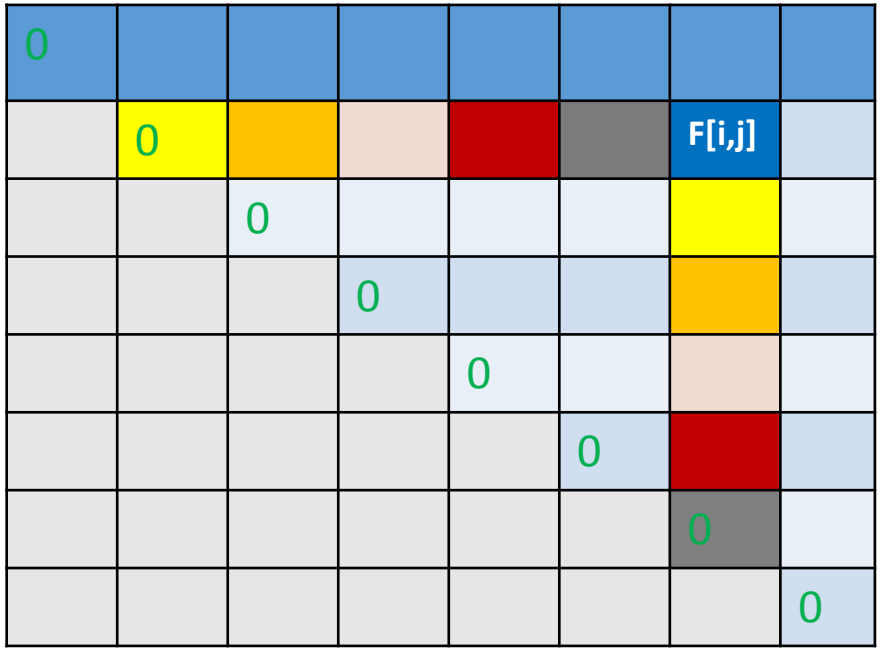


2 вариант

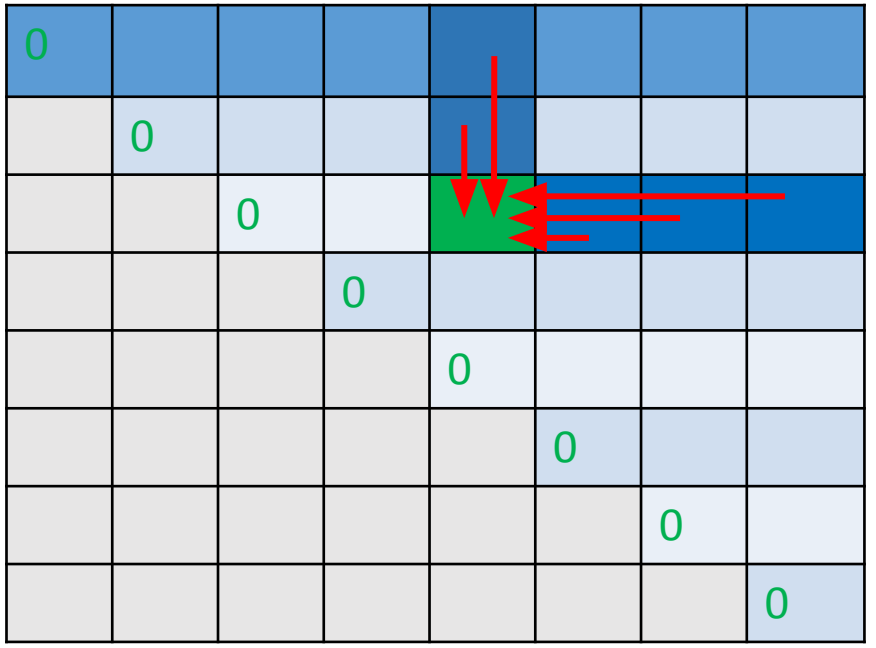


ФПМИ БГУ

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



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

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

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

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

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

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

	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	$(A_1 \cdot A_2 \cdot A_3 \cdot A_4)$ **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 $(A_2) \cdot (A_3 \cdot A_4)$ = 7 875 $(A_2 \cdot A_3) \cdot (A_4)$ = 1 200
3			0	$(A_3 \cdot A_4)$ **3 500**
4				0

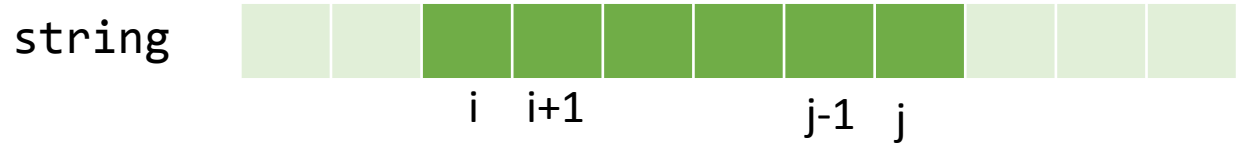
Ответ: 3 100 операций умножения $(A_1 \cdot (A_2 \cdot A_3)) \cdot A_4$

Задача 4. Максимальный палиндром

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

Для строки: a s d f s l a
Максимальный палиндром: a s d s a

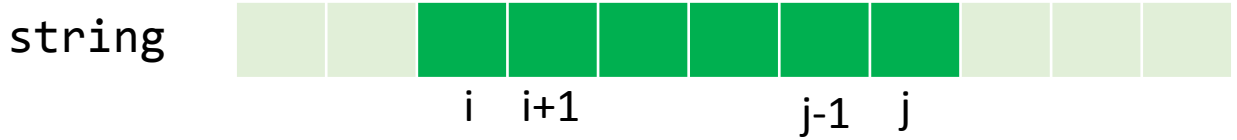
Обозначим через $F[i,j]$ длину максимального палиндрома, который можно получить, если мы рассматриваем элементы строки от индекса i до j включительно.



n=6

F

	1	2	3	4	5	6
1						?
2						
3						
4						
5						
6						



Строки длины 1

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

Строки длины 2

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

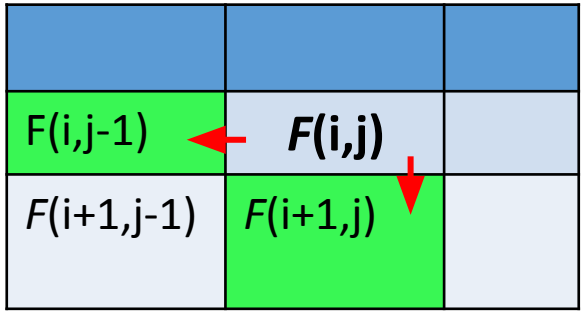
$$F[i,i+1]=\begin{cases}2, if & string[i]=string[i+1]\\ 1, if & string[i]\neq string[i+1]\end{cases}$$

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

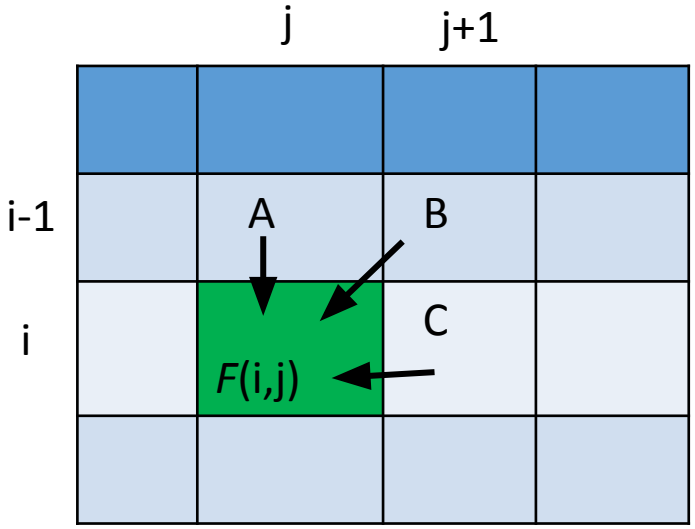
Строки длины >2

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

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



Задачи A, B и C являются зависимыми, так как они требуют для своего решения знать длину максимального 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	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 s f s a

Время работы алгоритма $O(n^2)$:

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

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

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

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

Задания

БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

Выполнить в системе Insight Runner следующие общие задачи:

Тема. Динамическое программирование

0.1 Оптимальное перемножение группы матриц (двухмерное ДП)
0.2 Единицы - число сочетаний из n по k(одномерное ДП, модульная арифметика)
0.3. Единицы (большие ограничения, только для желающих)
20. Палиндром (двухмерное ДП, строки)
69. Кувшинки (простейшее одномерное ДП)

ЗАДАЧА 5.
БИНАРНЫЕ ПОСЛЕДОВАТЕЛЬНОСТИ

©ФПМИ, БГУ, Доскоч Роман, 2020 год

ЗАДАЧА

Пусть X и Y — две бинарных последовательности длины N и M соответственно, состоящие из нулей и единиц. Сами последовательности X и Y можно рассматривать как запись в двоичной форме некоторых двух положительных целых чисел.

Необходимо найти максимальное число Z, двоичную запись которого можно получить как из x, так и из y вычёркиванием цифр.

1 ЧАСТЬ (НАХОЖДЕНИЕ ДЛИНЫ LCS)

Сначала перевернем строки.

$$\begin{cases} f[i][j] = f[i-1][j-1] + 1, \\ X[i] = Y[j] \\ f[i][j] = \max(f[i-1][j], f[i][j-1]), \\ X[i] \neq Y[j] \end{cases}$$

Дп назад

Сложность O(|Y|*|X|)

Память O(|Y|*|X|)

\	1	0	1	1	0	1	0	1
1	1	1	1	1	1	1	1	1
0	1	2	2	2	2	2	2	2
0	1	2	2	2	3	3	3	3
0	1	2	2	2	3	4	4	4
1	1	2	3	3	3	4	5	5
1	1	2	3	3	4	4	5	6
0	1	2	3	4	4	4	5	6
1	1	2	3	4	5	5	5	6

2 ЧАСТЬ (ВОССТАНОВЛЕНИЕ ПУТИ)

i, j текущие индексы
i1,j1 индексы первых единиц
i0,j0 индексы первых нулей

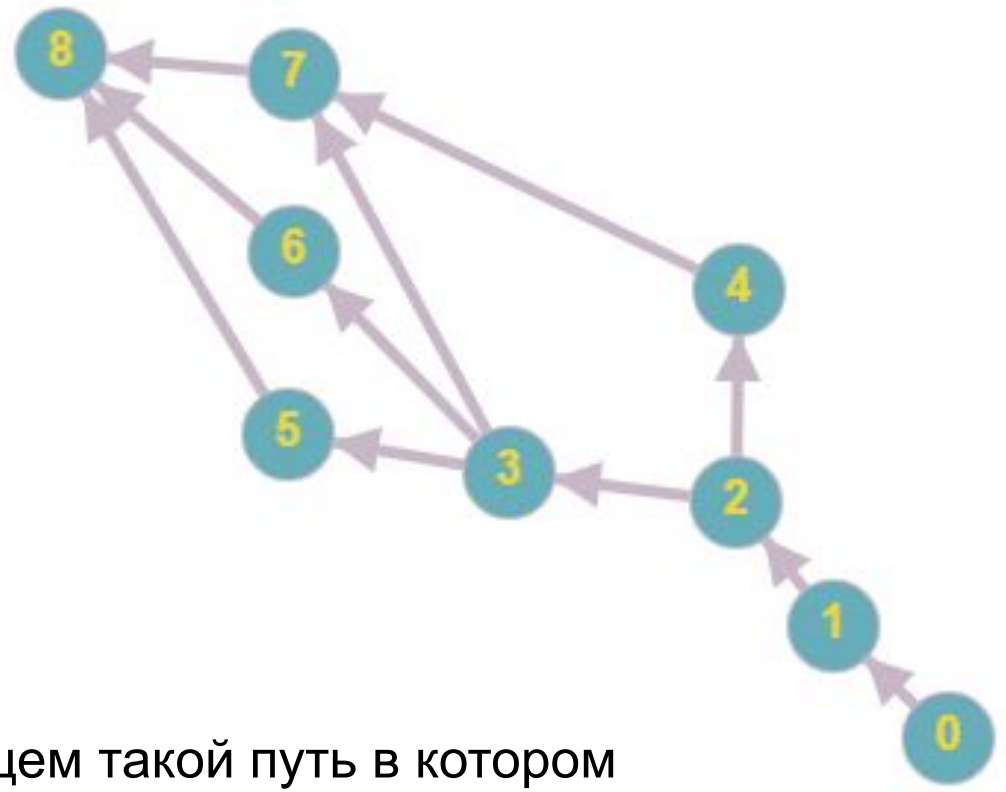
$$\begin{cases} ans \mathrel{+}= \text{"1"}, X[i1] = Y[j1] = \text{"1"}\ and\ f[i][j] = f[i1][j1] + 1 \\ \Rightarrow i = i1, j = j1; \\ \text{---------- --} \\ ans \mathrel{+}= \text{"0"}, \text{иначе} \\ \Rightarrow i = i0, j = j0. \end{cases}$$

X = 10101101
Y = 10110001
ans = 101101

\	1	0	1	1	0	1	0	1
1	1	1	1	1	1	1	1	1
0	1	2	2	2	2	2	2	2
0	1	2	2	2	3	3	3	3
0	1	2	2	2	3	4	4	4
1	1	2	3	3	3	4	5	5
1	1	2	3	3	4	4	5	6
0	1	2	3	4	4	4	5	6
1	1	2	3	4	5	5	5	6

2 СПОСОБ ВОССТАНОВЛЕНИЯ ПУТИ

Поиск в ширину



Ищем такой путь в котором “быстрее”
встречаются первые 1
Верный путь будет:
012358

\	1	0	1	1	0	1	0	1
1	1\	1<	1\	1\	1<	1\	1<	1\
0	1^	2\	2<	2<	2\	2<	2\	2<
0	1^	2\	2*	2*	3\	3<	3\	3<
0	1^	2\	2*	2*	3\	4*	4\	4<
1	1\	2^	3\	3\	3*	4\	5*	5\
1	1\	2^	3\	3\	4*	4\	5^	6\
0	1^	2\	3^	4*	4\	4*	5\	6^
1	1\	2^	3\	4\	5*	5\	5*	6\

Спасибо за внимание!

©ДМА ФПМИ Соболевская Е.П., 2021 год