

# ОСНОВЫ ПРОГРАММИРОВАНИЯ

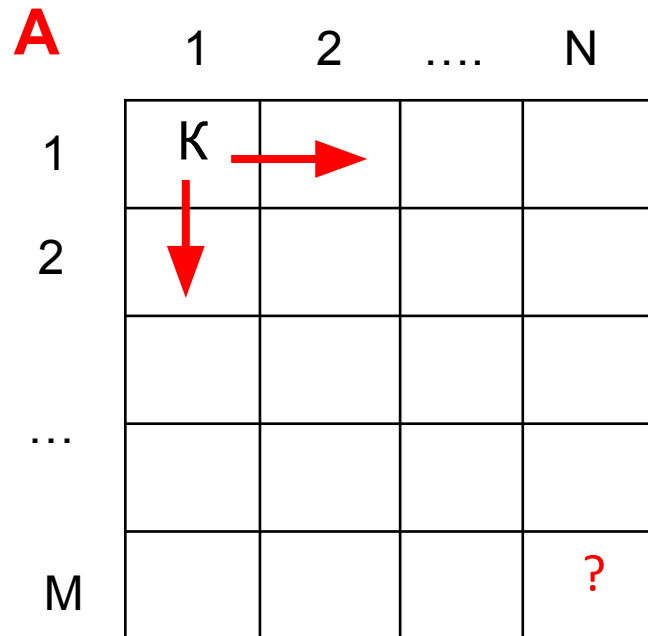
## Лекция 20

### **Динамическое программирование**

#### Часть 2

## Пример 6. Сколько путей («Задача о хромом короле»)

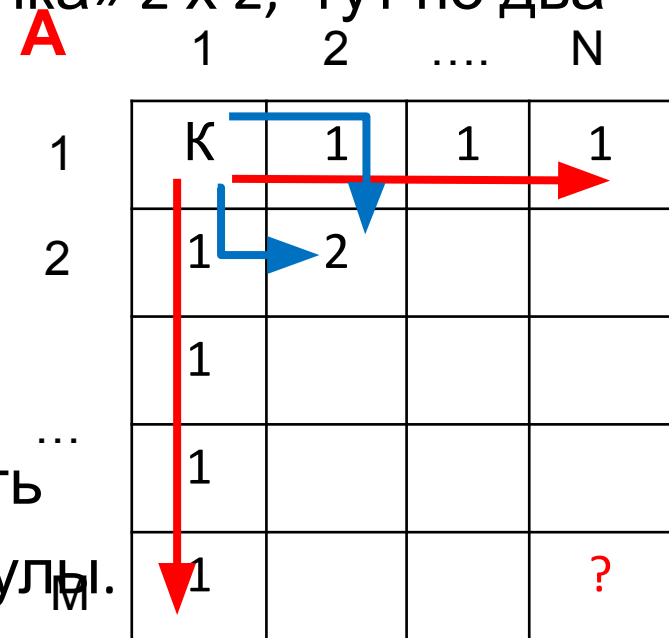
Дана матрица  $A$  размером  $M \times N$ . Король стоит в клетке (1, 1). Ему нужно попасть в некоторую клетку по диагонали. Король может за один шаг перейти на клетку вправо или на клетку вниз. Сколько вариантов путей у короля?



## Пример 6. Сколько путей («Задача о хромом короле»)

- Начнем с маленьких табличек. Если у нас «табличка» в одну строку, то количество способов пройти по клеткам = 1, то есть король может только двигаться все время направо.
- То же самое, если у нас «табличка» представляет из себя один столбик - тоже один способ.
- Чуть больше способов, если «табличка»  $2 \times 2$ , тут по два способа:
  - либо вправо и вниз
  - либо вниз и вправо.

Таким образом у нас есть какие-то начальные значения, можно приступить к анализу для получения общие формулы.



## Пример 6. Сколько путей («Задача о хромом короле»)

Очевидно, что заполнение нашей матрицы – симметрично относительно диагонали, т.к. в одном случае мы рассматриваем горизонтально расположенные прямоугольники, а в другом – вертикально (т.е. решение для клетки  $A[2,3] = A[3,2]$ ).

Как можно попасть в клетку  $A[i,j]$ ? Очевидно, что способы разбиваются на две группы: или из  $i-1$  столбца или с  $j-1$  строки. Следовательно **количество способов попасть в  $A[i,j]$**  клетку равно сумме способов попасть в соседнюю сверху и соседнюю слева клетки.

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

<b>A</b>	1	2	....	N
1	K	1	1	1
2	1	2	3	4
...	1	3	6	10
...	1	4	10	
M	1	5	15	?

<b>A</b>	1	2	....	N
1	K	1	1	1
2	1	2	3	4
...	1	3	6	
...	1	4		
M	1	5		?

## Пример 6. Сколько путей («Задача о хромом короле»)

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

Видно, что формула работает, если  $i$  и  $j$  больше 1. Ну а если  $i$  и  $j$  равны 1, то эти частные случаи у нас уже были (это одномерная таблица: строка или столбец). Таким образом  $A[1,j] = 1$  и  $A[i,1] = 1$ . Заполнять будем по строкам.

Заполнив всю таблицу мы получим ответ задачи – он будет лежать в последней клетке.

Замечание: если в задаче просто спрашивается количество способов, то мы можем себе позволить не хранить весь двухмерный массив в памяти. Когда мы вычисляем какую-то строчку у нас вычисления идут по строкам. Мы основываемся только на тех клеточках этой строки, которые мы уже посчитали на левых и на тех клеточках предыдущей строки которые над нами. По этому, для того чтобы посчитать решение, если речь идет не о **лучших** путях, а об их **количестве**, достаточно хранить только последнюю строку которую мы вычислили целиком и держать какой-то временный массив, где мы будем считать новую строку. Потом, переприсвоить и т.д. Тогда памяти нам надо не под массив  $M$  умножить на  $N$ , а  $N$  умножить на 2. Так что, здесь можно поэкономить, вот такое примечание по поводу оптимизации по памяти.

# Пример 7. Максимальная сумма в таблице

Дана матрица  $A$  размером  $M \times N$ . В каждой клетке – целые положительные числа. Королю нужно пройти по столбцам матрицы и набрать максимальную сумму.

Двигаться нужно по следующим правилам:

- Начинать нужно с первого столбца, с верхнего элемента.
- Передвигаться в следующий столбец: либо в рядом стоящую клетку справа, либо в соседнюю вниз.
- Закончить движение в последнем столбце.

Для решения будем использовать дополнительную матрицу  $B$  такого же размера. В каждой клетке этой матрицы будет храниться максимальная сумма, которую можно набрать, двигаясь по исходной матрице и придя в эту клетку.

	1	2	....	N	
A	K	7	5	2	9
1	4	3	1	5	6
2	8	10	16	1	3
...					
M	4	6	5	4	7

# Пример 7. Максимальная сумма в таблице

Создадим дополнительную матрицу  $M$  такого же размера. В каждой клетке которой будем записывать **максимальную сумму, которую мы накопим дойдя до неё**.

Значения для  $M[0, j] = A[0, j] + A[0, j-1]$ , а для  $M[i, 0] = A[i, 0] + A[i-1, 0]$ . Манёвра тут не много, а из прошлой задачи мы знаем про «маленькие одномерные таблички».

Не трудно увидеть и общий случай:

$$M[i, j] = A[i, j] + \max \begin{cases} M[i, j-1] \\ M[i-1, j] \end{cases}$$

Заполнив таблицу, ответ получим в последней клетке  $M[M, N]$ .

	1	2	....	N	
A	K	7	5	2	9
1	4	3	1	5	6
2	8	10	16	1	3
...					
M	4	6	5	4	7

**M**

0	7	12	14	23
4	10	13	19	29
12	22	38	39	42
16	28	43	47	<b>54</b>

**M**

0	7	7 + 5 = 12	12 + 2 = 14	14 + 9 = 23
4	7 + 3 = 10	12 + 1 = 13	14 + 5 = 19	23 + 6 = 29
4 + 8 = 12	12 + 10 = 22	22 + 16 = 38	38 + 1 = 39	39 + 3 = 42
12 + 4 = 16	22 + 6 = 28	38 + 5 = 43	43 + 4 = 47	47 + 7 = <b>54</b>

# Пример 7. Максимальная сумма в таблице

Очевиден стал и маршрут нашего короля. **Обратный ход:** начинаем от клетки с ответом и на каждом следующем ходе выбираем клетку по вертикали или горизонтали с самой большой суммой:

**М**

0	7	12	14	23
4	10	13	19	29
12	22	38	39	42
16	28	43	47	<b>54</b>

Значит исходный путь короля, по матрице А, такой:

Можете проверить все другие пути – сумма полученная по проходу через выделенные клетки самая большая для данной таблицы.

	1	2	....	N	
A	K	7	5	2	9
1	4	3	1	5	6
2	8	10	16	1	3
...					
M	4	6	5	4	7



## Пример 8. Максимальная сумма в таблице 2

Дана матрица  $A$  размером  $M \times N$ . Нужно пройти по столбцам матрицы и набрать максимальную сумму.

Двигаться нужно по следующим правилам:

- Начинать нужно с первого столбца.
- Передвигаться в следующий столбец: либо в рядом стоящую клетку, либо в соседнюю по диагонали вверх, либо в соседнюю по диагонали вниз.
- Закончить движение в последнем столбце.

Для решения будем использовать дополнительную матрицу **B** такого же размера. В каждой клетке этой матрицы будет храниться максимальная сумма, которую можно набрать, двигаясь по исходной матрице и придя в эту клетку.

## Пример 8. Продолжение

1. По условию задачи мы начинаем из любой клетки ПЕРВОГО столбца!
  2. Можем перейти на клетку в соседнем столбце, но **первые** и **последние** строки отличаются (меньше путей перехода)
  3. Когда мы записываем значение в матрицу **B**, мы выбираем **максимальное** из возможных, которое получается если прийти в эту клетку разными путями из предыдущего столбца. То есть, мы заполняем матрицу **B** – ПО СТОЛБЦАМ!
- Справедливы следующие отношения:**

$$B[i, 0] = A[i, 0], \text{ при } 0 \leq i < M.$$

$$B[0, j] = A[0, j] + \max( B[0, j - 1], B[1, j - 1] ), \text{ при } 1 \leq j < N.$$

$$B[M - 1, j] = A[M - 1, j] + \max( B[M - 1, j - 1], B[M - 2, j - 1] ), \text{ при } 1 \leq j < N.$$

$$B[i, j] = A[i, j] + \max( B[i - 1, j - 1], B[i, j - 1], B[i + 1, j - 1] ), \text{ при } 1 \leq i < M - 1, 1 \leq j < N.$$

<b>A</b>	0	1	...	N
1	2	6	10	3
2	4	12	1	9
...	1	4	9	8
...	8	7	5	6
M	3	5	9	2

## Пример 8. Продолжение

Справедливы следующие отношения:

$$B[i, 0] = A[i, 0], \text{ при } 0 \leq i < M. \quad B[0, j] = A[0, j] + \max\{B[0, j-1], B[1, j-1]\}, \text{ при } 1 \leq j < N.$$

$$B[M-1, j] = A[M-1, j] + \max\{B[M-1, j-1], B[M-2, j-1]\}, \text{ при } 1 \leq j < N.$$

$$(B[i-1, j-1])$$

<b>A</b>	0	1	...	N
0	2	6	10	3
1	4	12	1	9
...	1	4	9	8
...	8	7	5	6
M	3	5	9	2

<b>B</b>	0	1	...	N
0	2	10	26	29
1	4	16	17	35
...	1	12	25	33
...	8	15	20	31
M	3	13	24	26

## Пример 8. Продолжение

Теперь подумаем, где лежит ответ.

По условию задачи, нам нужно найти путь по столбцам и набрать максимальную сумму. В максимальной сумме – максимальные слагаемые (логично?! ;-). Значит ищем в **последнем** столбце максимальное число – это и будет та ячейка в которой мы **должны закончить** путь.

Потом, из этой клетки, ищем соседнюю клетку, в которую мы можем перейти по правилам из условий задачи, с максимальным значением в ней. И так до первого столбца.

В нашем случае **обратный ход** по матрице **B** такой:

**A**

	0	1	...	N
0	2	6	10	3
1	4	12	1	9
...	1	4	9	8
...	8	7	5	6
M	3	5	9	2

А по матрице **A** мы должны были пройти так (**начав** с A[1,0]):

**B**

	0	1	...	N
0	2	10	26	29
1	4	16	17	35
...	1	12	25	33
...	8	15	20	31
M	3	13	24	26

## Пример 9. Задача "Divisibility" 1999-2000 ACM NEERC

Рассмотрим произвольную последовательность целых чисел. Можно поставить знаки операций « + » или « - » между целыми в данной последовательности, получая при этом различные арифметические выражения, которые при их вычислении имеют различные значения. Давайте, например, возьмем следующую последовательность: 17, 5, -21, 15. Из нее можно получить восемь различных выражений:

$$\begin{aligned}17 + 5 + (-21) + 15 &= 16 & 17 - 5 + (-21) + 15 &= 6 \\17 + 5 + (-21) - 15 &= -14 & 17 + 5 - (-21) + 15 &= 58 \\17 + 5 - (-21) - 15 &= 28 & 17 - 5 + (-21) - 15 &= -24 \\17 - 5 - (-21) + 15 &= 48 & 17 - 5 - (-21) - 15 &= 18\end{aligned}$$

Назовем последовательность **делимой** на  $K$ , если можно так расставить операции + или - между целыми в последовательности, что значение полученного выражения делилось бы нацело на  $K$ . В приведенном выше примере последовательность делима на 7 ( $17+5+-21-15=-14$ ), но не делима на 5.

Напишите программу, которая определяет делимость последовательности целых чисел.

## Пример 9. Задача "Divisibility" 1999-2000 ACM NEERC

Определение делимости последовательности целых чисел:

- 1) Если число  $N$  делится на некоторое число  $K$ , то остаток от деления  $N$  на  $K$  равен 0.
- 2) Остаток от деления суммы чисел на некоторое число равен остатку от деления суммы остатков от деления каждого числа на это число.

$$(a + b) \bmod c == (a \bmod c + b \bmod c) \bmod c$$

Вспомогательная таблица строится по строкам, где каждая строка соответствует рассмотренной части последовательности (от начала до  $i$ -го элемента), а столбцы соответствуют возможным остаткам от деления нацело на  $K$ .

Каждый элемент таблицы заполняется на основе строки предыдущих элементов.

Для этого используются два правила:

1. Если для 1-го элемента последовательности можно получить остаток  $j$  при делении нацело на  $K$ , то клетка таблицы с индексом  $[1, j]$  заполняется единицей.
2. Для каждой последующей строки таблицы мы заполняем значения клеток на основе предыдущей строки.

Для этого берём остаток от деления на  $K$  текущего,  $i$ -го, элемента нашей последовательности (обозначим его  $a[i]$ ), и рассматриваем два случая :

- Если предыдущая строка имела значение **1** в клетке  $[(i-1), j]$ , то мы можем получить новый остаток  $j'$ , если **добавим**  $a[i]$  к  $j$  из предыдущей строки:  $j' = j + a[i]$  и ставим **1** в ячейку  $[i, j']$ . Если получилось  $j' > (K-1)$ , то из  $j'$  вычитаем  $K$ :  $j' = j' - K$ , а потом записываем **1**.

- Если предыдущая строка имела значение **1** в клетке  $[(i-1), j]$ , то мы можем получить ещё один новый остаток  $j'$ , если **вычтем**  $a[i]$  из  $j$  из предыдущей строки:  $j' = j - a[i]$  и ставим **1** в ячейку  $[i, j']$ . Если получилось  $j' < 0$ , то из  $j'$  прибавляем  $K$ :  $j' = j' + K$ , а потом записываем **1**.

Используя этот алгоритм, мы можем построить всю таблицу и определить, является ли исходная последовательность делимой на  $K$ .

# Решение

Проверим наш пример:

- 1)  $17 \bmod 7 = 3$ . В ячейку  $[0, 3]$  записываем 1.
- 2)  $5 \bmod 7 = 5$ . Смотрим предыдущую строку. 1 в ячейке  $[0, 3]$ , значит прошлый остаток от деления  $j=3$ , и у нас два варианта для расчёта:
  - 1)  $J' = 3 + 5 = 8$ . Так как  $J' > 7$ , то  $J' = J' - 7 = 1$ . В ячейку  $[1, 1]$  записываем 1
  - 2)  $J' = 3 - 5 = -2$ . Так как  $J' < 0$ , то  $J' = J' + 7 = 5$ . В ячейку  $[1, 5]$  записываем 1
- 3)  $-21 \bmod 7 = 0$ . Смотрим предыдущую строку. 1 в двух ячейках:  $[1, 1]$  и  $[1, 5]$  значит у нас **два** прошлых остатка от деления  $j=1$  и  $j=5$ . **На каждый из них** у нас **по два варианта для расчёта**. Но, так как один из операндов 0, то и сложение и вычитание не отличаются и не меняют остатков, мы только переносим их на строку ниже - ставим по одной 1 в ячейки  $[2, 1]$  и  $[2, 5]$ .
- 4)  $15 \bmod 7 = 1$ . Смотрим предыдущую строку. 1 в двух ячейках:  $[2, 1]$  и  $[2, 5]$  значит у нас снова **два прошлых остатка** от деления  $j=1$  и  $j=5$ . **На каждый из них** у нас **по два варианта** для расчёта:
  - 3)  $J' = 1 + 1 = 2$ . В ячейку  $[3, 2]$  записываем 1
  - 4)  $J' = 1 - 1 = 0$ . В ячейку  $[3, 0]$  записываем 1
  - 5)  $J' = 5 + 1 = 6$ . В ячейку  $[3, 6]$  записываем 1
  - 6)  $J' = 5 - 1 = 4$ . В ячейку  $[3, 4]$  записываем 1

Так как, в ячейке  $[3, 0]$  стоит 1, значит существует такая последовательность из наших чисел, которую можно собрать при помощи знаков  $+$  и  $-$  и которая будет делиться на 7 (

		0	1	2	3	4	5	6
0	$17 \bmod 7 = 3$				1			
1	$5 \bmod 7 = 5$		1				1	
2	$-21 \bmod 7 = 0$		1				1	
3	$15 \bmod 7 = 1$	1		1		1		1

# Решение

Проверим наш пример на кратность 5:

- 1)  $17 \bmod 5 = 2$ . В ячейку  $[0, 2]$  записываем 1.
- 2)  $5 \bmod 5 = 0$ . Смотрим предыдущую строку. 1 в ячейке  $[0, 2]$ , значит прошлый остаток от деления  $j=2$ , и у нас два варианта для расчёта. Но, так как один из операндов 0, то и сложение и вычитание не отличаются и не меняют остатков, мы только переносим их на строку ниже - ставим 1 в ячейку  $[1, 2]$ .
- 3)  $-21 \bmod 5 = -1$ . Смотрим предыдущую строку. 1 в ячейке  $[1, 2]$ , значит прошлый остаток от деления  $j=2$ , и у нас два варианта для расчёта:
  - 1)  $J' = 2 + (-1) = 1$ . В ячейку  $[2, 1]$  записываем 1
  - 2)  $J' = 2 - (-1) = 3$ . В ячейку  $[2, 3]$  записываем 1
- 4)  $15 \bmod 5 = 0$ . Смотрим предыдущую строку. У нас две 1: в ячейках  $[2, 1]$  и  $[2, 3]$ , значит у нас два прошлых остатка от деления  $j=1$  и  $j=1$ , и у нас по два варианта для расчёта каждого из них! Но, так как один из операндов 0, то и сложение и вычитание не отличаются и не меняют остатков, мы только переносим их на строку ниже - ставим 1 в ячейки  $[3, 1]$  и  $[3, 3]$ .

Так как, в ячейке  $[3, 0]$  нет 1, значит нет такой последовательности из наших чисел 17, 5, -21 и 15, которую можно собрать при помощи знаков + и - и которая будет делиться на 5 без остатка.

		0	1	2	3	4
0	$17 \bmod 5 = 2$			1		
1	$5 \bmod 5 = 0$			1		
2	$-21 \bmod 5 = -1$		1		1	
3	$15 \bmod 5 = 0$		1		1	



## Пример 10. Задача "Gangsters"

- $N$  гангстеров идут в ресторан.  $i$ -ый гангстер заходит в  $T_i$ -е время и имеет при себе  $P_i$  денег. Дверь ресторана имеет  $k+1$  стадий открытия, выраженных в целых числах от 0 до  $k$ . Состояние открытия может измениться на 1 в единицу времени, т.е. либо открыться на 1, либо закрыться на 1, либо остаться прежним. В начальный момент состояние двери закрытое = 0.
- $i$ -тый гангстер может войти в ресторан, если дверь открыта специально для него, т.е. состояние двери совпадает с шириной его плеч  $S_i$ . Если в момент времени, когда гангстер подошел к ресторану, состояние открытия двери не совпадает с шириной его плеч, то он уходит и никогда не возвращается. Ресторан работает в интервале времени  $[0, T]$ .
- Цель: собрать в ресторане гангстеров с максимальным количеством денег.

Другие применения данной задачи:

«О максимизации прибыли от продажи товаров». Она использует динамическое программирование для нахождения максимальной прибыли, которую можно получить, продав товары в определенные моменты времени.

«О расписании такси». Дан набор заявок на такси, каждая из которых описывается временем начала  $t$ , временем окончания  $s$  и платой  $p$ . Необходимо определить максимальную сумму, которую может заработать таксист, выполнив некоторый набор заявок. При этом время начала выполнения заявки не может быть раньше времени ее начала  $t$  и не может быть позже времени ее окончания  $s$ .

# Гангстеры , продолжение

Первая строка входного файла содержит значения  $N$ ,  $K$  и  $T$ , разделенные пробелами ( $1 \leq N \leq 100$ ,  $1 \leq K \leq 100$ ,  $0 \leq T \leq 30000$ );

вторая строка содержит моменты времени, в которые гангстеры подходят к ресторану  $T_1, T_2, \dots, T_N$ , разделенные пробелами ( $0 \leq T_i \leq T$  для  $i = 1, 2, \dots, N$ );

в третьей строке записаны суммы денег каждого гангстера  $P_1, P_2, \dots, P_N$ , разделенные пробелами ( $0 \leq P_i \leq 300$ , для  $i = 1, 2, \dots, N$ );

четвертая строка содержит значения ширины плеч каждого гангстера, разделенные пробелами ( $0 \leq S_i \leq K$  для  $i = 1, 2, \dots, N$ ). Все значения целые.

**Выходные данные:** В выходной файл выдать одно целое число — максимальное значение достатка всех гангстеров, собранных в ресторане. Если ни один гангстер не может попасть в ресторан, выдать 0.

## Пример 1

Вход:            Выход:

4 10 20            26

10 16 8 16

10 11 15 1

10 7 1 8

## Пример 2

Вход:            Выход:

2 17 100            0

5 0

50 33

6 1

# Гангстеры , продолжение

Для решения этой задачи методом динамического программирования, мы можем использовать подход "восходящей динамики".

Для начала, мы создадим таблицу размером  $(N+1) \times (k+1)$ , где каждый элемент  $(i, j)$  представляет собой максимальную сумму денег, которую можно собрать, если первые  $i$  гангстеров зашли в ресторан, а дверь открыта на  $j$  единиц от начального состояния.

Мы начнем заполнять эту таблицу снизу вверх, начиная с случая, когда ни один гангстер не входит в ресторан. В этом случае, все элементы таблицы будут равны 0.

Затем, мы будем постепенно добавлять гангстеров в таблицу. Для каждого гангстера  $i$  мы проверяем, может ли он войти в ресторан, если дверь открыта на  $j$  единиц от начального состояния. Если да, то мы можем выбрать, включить его или нет в решение. Если мы выберем его, то мы должны добавить его деньги к максимальной сумме денег, которую мы можем собрать при двери в состоянии  $j - S_i$ , когда  $i$ -й гангстер зашел в ресторан. Если мы не выберем его, то мы должны просто перейти к следующему гангстеру.

Мы будем заполнять таблицу слева направо, т.е. для каждого  $i$  и  $j$  мы будем использовать значения, которые мы уже посчитали для  $i-1$  и всех  $j$ . На каждом шаге мы будем обновлять таблицу, используя следующую формулу:

$$dp[i][j] = \max(dp[i-1][j], dp[i-1][j - S_i] + P_i), \text{ если гангстер } i \text{ может войти в ресторан}$$

# Гангстеры , продолжение

Затем, чтобы получить максимальную сумму денег, которую мы можем собрать, когда все гангстеры уже добавлены в решение, мы просто выбираем максимальный элемент в последней строке таблицы.

1. Сначала отсортируем гангстеров по времени входа в ресторан, так как мы должны убедиться, что дверь открыта в нужный момент времени для каждого гангстера.
2. Создадим таблицу размером  $(N+1) \times (k+1)$ , где  $N$  - количество гангстеров,  $k$  - количество возможных состояний открытия двери (от 0 до  $k$  включительно). В ячейке  $(i,j)$  будем хранить максимальную сумму денег, которую можно собрать, если у нас есть первые  $i$  гангстеров и дверь открыта на  $j$ .
3. Начнем заполнять таблицу снизу вверх, начиная с первого гангстера и состояния двери 0. Для каждого гангстера  $i$  и состояния двери  $j$  (где  $j \geq S_i$ ) вычислим значение в соответствующей ячейке  $(i,j)$  следующим образом:
  1. Если гангстер не может войти в данный момент времени (т.е.  $j \neq S_i$ ), то значение в ячейке  $(i,j)$  равно значению в ячейке  $(i-1,j)$ .
  2. Если гангстер может войти в данный момент времени (т.е.  $j = S_i$ ), то значение в ячейке  $(i,j)$  равно максимуму из двух значений:
    1. Значение в ячейке  $(i-1,j)$ , если гангстер не заходит в данный момент времени.
    2. Сумма денег гангстера  $P_i$  и значения в ячейке  $(i-1, j-S_i)$ , если гангстер заходит в данный момент времени.
4. Заполнение таблицы заканчивается, когда мы вычисляем значение в ячейке  $(N,k)$ .

Искомая максимальная сумма денег будет находиться в ячейке  $(N,k)$ .

# Гангстеры , продолжение

Чтобы найти, какие гангстеры были выбраны, начинаем с ячейки  $(N,k)$  и движемся вверх по таблице, пока не дойдем до ячейки  $(1,0)$ . Если значение в ячейке  $(i,j)$  равно значению в ячейке  $(i-1,j)$ , то гангстер  $i$  не был выбран. Если значение в ячейке  $(i,j)$  равно сумме денег гангстера  $P_i$  и значения в ячейке  $(i-1,j-S_i)$ , то гангстер  $i$  был выбран, а дверь была открыта в момент времени  $j-S_i$ .

# Пример

t = 1 2 3 4 5 6 - времена прихода гангстеров

S = 1 2 3 4 5 1 - ширина плеч

P = 1 1 1 1 1 100 - деньги

↓ L - состояние двери

4					4	5	5
3				3	3	4	5
2			2	2	3	3	4
1		1	1	2	2	3	103
0	0	0	1	1	2	2	3
0	0	1	2	3	4	5	6

← Время прихода гангстеров

$$m_{i,j} = \max \{ [m_{i-1,j-1}, m_{i-1,j}, m_{i-1,j+1}] + f_i \}$$

где

$$f_i = \begin{cases} p_i, & \text{если } L = s_i \\ 0, & \text{иначе} \end{cases}$$

# Гангстеры , вариант 2

1. Отсортируйте гангстеров по времени их прихода  $T_i$ .
2. Создайте двумерный массив  $dp$  размером  $(N+1) \times (k+1)$ , где  $dp[i][j]$  будет представлять максимальное количество денег, которое можно получить, если рассмотреть только первых  $i$  гангстеров и дверь находится в состоянии  $j$ .
3. Инициализируйте  $dp[0][0] = 0$ , что означает, что если не пришел ни один гангстер и дверь закрыта, то заработать нельзя.
4. Заполните массив  $dp$  построчно, начиная со второй строки ( $i = 2$ ).
5. Для каждого гангстера  $i$  и для каждого состояния двери  $j$  переберите все предыдущие состояния двери  $k$  ( $k \leq j$ ), при которых дверь открыта для гангстера  $i$ . Для этого проверьте, что  $j - k = S_i$ . Если это условие выполняется, то гангстер  $i$  может войти в ресторан в состоянии двери  $j$  и его деньги будут добавлены к максимальному количеству денег, которое можно получить в состоянии двери  $k$ . Таким образом, обновите значение  $dp[i][j] = dp[i-1][k] + P_i$ .
6. Если гангстер  $i$  не может войти в ресторан при состоянии двери  $j$ , то значение  $dp[i][j]$  будет равно значению  $dp[i-1][j]$ .
7. Максимальное количество денег, которое можно получить, будет равно максимальному значению  $dp[N][j]$  для всех  $j$ .

Этот алгоритм будет работать за время  $O(N * K^2)$ , что является

# Задача о преобразовании строк. Алгоритм Ахо

Пусть даны две строки  $S_1$  и  $S_2$ . Необходимо за минимальное число **допустимых** операций преобразовать строку  $S_1$  в строку  $S_2$ . Допустимой операцией являются следующие операции удаления символа из строки и вставки символа в строку:

$\text{DEL}(S, i)$  – удалить  $i$ -ый элемент строки  $S$ ;

$\text{INS}(S, i, c)$  – вставить символ  $c$  после  $i$ -го элемента строки  $S$ .

Минимальное количество операций =

редакторское расстояние =

расстояние Левенштейна

В общем случае алгоритм, который будет рассмотрен, носит имя Вагнера — Фишера



Пусть  $M(i, j)$  – минимальное количество операций, которые требуются, чтобы преобразовать **начальные  $i$  символов** строки  $S_1$  в **начальные  $j$  символов** строки  $S_2$ :  $S_1[0..i] \rightarrow S_2[0..j]$ .

*Начальные значения*

$S_1[0..0]$  и  $S_2[0..0]$  – пустые строки.

Заметим, что для преобразования пустой строки в строку длины  $j$  требуется  $j$  операций вставки, т.е.  $M(0, j) = j$ .

Аналогично, для преобразования строки длины  $i$  в пустую строку требуется  $i$  операций удаления, т.е.  $M(i, 0) = i$ .

Если мы решили подзадачу с параметрами  $i-1$  и  $j-1$ . Это означает, что из строки  $S_1[0..i-1]$  построена строка  $S_2[0..j-1]$  за минимальное число допустимых операций  $M(i-1, j-1)$ .

Рассмотрим два случая:

- 1) Пусть  $S_1[i] = S_2[j]$ . В этом случае для получения строки  $S_2[0..j]$  из строки  $S_1[0..i]$  не требуется никаких дополнительных операций, достаточно тех, что были выполнены для получения строки  $S_1[0 .. i-1]$  из  $S_2[0..j-1]$ . Следовательно,  $M(i, j) = M(i - 1, j - 1)$ .
- 2) Пусть теперь  $S_1[i] \neq S_2[j]$ . Возможны два способа получения строки  $S_2[0..j]$ :

1. Пусть из строки  $S_1[0 .. i - 1]$  построена строка  $S_2[0 .. j]$  за минимальное количество операций  $M(i - 1, j)$ . Тогда для получения строки  $S_2[0..j]$  из строки  $S_1[0 .. i]$  **требуется удалить  $i$ -ый символ из строки  $S_1$ .**

$$M(i, j) = M(i - 1, j) + 1$$

2. Пусть из строки  $S_1[0..i]$  построена строка  $S_2[0..j-1]$  за минимальное количество операций  $M(i, j-1)$ . Тогда для получения строки  $S_2[0..j]$  из строки  $S_1[0..i]$  **потребуется одна операция вставки  $i$ -го символа строки  $S_1$  после символа  $S_2[j-1]$ .**

$$M(i, j) = M(i, j - 1) + 1$$

Понятно, что из этих двух возможностей нам нужно выбрать лучшую.

Таким образом, получаем следующие рекуррентные соотношения:

$$\bullet M(0, j) = j;$$

$$M(i, 0) = i;$$

$$M(i, j) = \begin{cases} \min \begin{pmatrix} M(i-1, j-1) \\ M(i-1, j) + 1 \\ M(i, j-1) + 1 \end{pmatrix}, & \text{если } S_1[i] = S_2[j]; \\ \min \begin{pmatrix} M(i-1, j) \\ M(i, j-1) \end{pmatrix} + 1, & \text{если } S_1[i] \neq S_2[j]; \end{cases}$$

Решением задачи будет значение  $M(m, n)$ , где  $m$  — длина

# Пример

$$S_1 = "abc", S_2 = "aabddc"$$

Построим таблицу  $M$ , нумерация строк и столбцов которой начинается с нуля и элементами которой будут числа, равные значениям функции, описанной выше. В данном случае таблица состоит из семи строк и четырёх столбцов.

<b><i>c</i></b>	6	5	4	<b>3</b>
<b><i>d</i></b>	5	4	3	4
<b><i>d</i></b>	4	3	2	3
<b><i>b</i></b>	3	2	1	2
<b><i>a</i></b>	2	1	2	3
<b><i>a</i></b>	1	0	1	2
$S_2$	0/0	1	2	3
$j/i$	$S_1$	<b><i>a</i></b>	<b><i>b</i></b>	<b><i>c</i></b>

В позиции  $M[i, j]$  будет находиться число, соответствующее минимальному количеству операций, преобразовывающих начальные  $i$  символов строки  $S_1$  в начальные  $j$  символов строки  $S_2$ . Например,  $M[1, 3] = 2$ , означает, что из строки «а» можно получить строку «aab», используя две допустимых операции. Решением задачи, является число, полученное в правом верхнем углу таблицы (у нас – это 3).

Значит за три допустимых операции можно преобразовать строку  $S_1$  в  $S_2$ .

## Обратный ход

Для определения операций нужно встать на последний символ строки  $S_1$  и начать движение по таблице от правого верхнего угла  $M[m, n]$ . В примере движение начнется с ячейки  $M[3,6]$ .

Находясь в ячейке  $M[i, j]$ , будем рассматривать три рядом расположенные ячейки:  $M[i-1, j-1]$ ,  $M[i, j-1]$  и  $M[i-1, j]$ . Всегда будем смещаться в ту из них, в которой значение **наименьшее**.

- Если такой ячейкой будет  **$M[i-1, j-1]$** , то **будем сдвигаться по диагонали влево-вниз** и тем самым будем перемещаться по строке  $S_1$  на один символ влево, т.е. сделаем текущим в строке символ, находящийся на  $i-1$  позиции.
- Если такой ячейкой будет  **$M[i-1, j]$** , то **будем сдвигаться влево** и при движении **будем удалять  $i$ -ый символ в строке**, перемещаясь на один символ влево по ней.
- Если такой ячейкой будет  **$M[i, j-1]$** , то будем сдвигаться вниз и при движении **будем вставлять после  $i$ -го**

## Последовательность действий для рассматриваемого примера

Изначально текущим в строке  $S_1$  является последний символ – символ  $c$ , в таблице эта ситуация соответствует ячейке  $M[3, 6]$ . Так как  $M[2, 5]$  имеет минимальное значение из трёх, окружающих эту ячейку, то осуществляем переход в эту ячейку  $M[2,5]$  и текущим в  $S_1$  становится предпоследний символ –  $b$ .

Далее, так как  $M[2, 4] = \min (M[2, 4], M[1, 4] \text{ и } M[1, 5])$ , передвигаемся в ячейку  $M[2, 4]$ . При этом вставим после текущего символа  $b$  символ  $S_2[5] = d$  ( $j=5$ ). Продолжая этот процесс вставим символ  $S_2[4] = d$ , затем в строке  $S_1$  сделаем текущим символ  $a$ , после чего вставим в строку  $S_1$  символ  $a$ . Процесс продолжается до тех пор, пока не достигнем ячейки  $M[0,0]$ .

Для нашего примера последовательность операций будет следующая:  $\text{INS}(S_1, 2, 'd'), \text{INS}(S_1, 2, 'd'), \text{INS}(S_1, 1, 'a')$ .

$abc \rightarrow abc \rightarrow abdc \rightarrow abddc \rightarrow abddc \rightarrow aabddc$

## Наш пример

$S_1 = "abc", S_2 = "aabddc"$

Изначально текущим в строке  $S_1$  является последний символ – символ **c**, в таблице эта ситуация соответствует ячейке  $M[3, 6]$ . Так как  $M[2, 5]$  имеет минимальное значение из трёх, окружающих эту ячейку, то осуществляем переход в эту ячейку  $M[2,5]$  и текущим в  $S_1$  становится предпоследний символ – **b**.

<b>c</b>	6	5	4	<b>3</b>
<b>d</b>	5	4	3	4
<b>d</b>	4	3	2	3
<b>b</b>	3	2	1	2
<b>a</b>	2	1	2	3
<b>a</b>	1	0	1	2
	0/0	1	2	3
		<b>a</b>	<b>b</b>	<b>c</b>

Далее, так как  $M[2, 4] = \min(M[2, 5], M[1, 4], M[1, 5])$ , передвигаемся в ячейку  $M[2, 4]$ . При этом вставим после текущего символа **b** символ  $S_2[5] = \mathbf{d}$  ( $j=5$ ). Продолжая этот процесс вставим символ  $S_2[4] = \mathbf{d}$ , затем в строке  $S_1$  сделаем текущим символ **a**, после чего вставим в строку  $S_1$  символ **a**. Процесс продолжается до тех пор, пока не достигнем ячейки  $M[0,0]$ .

Для нашего примера последовательность операций будет следующая:

$\text{INS}(S_1, 2, 'd'), \text{INS}(S_1, 2, 'd'), \text{INS}(S_1, 1, 'a')$ .

$abc \rightarrow abc \rightarrow abdc \rightarrow abddc \rightarrow abddc \rightarrow aabddc$

# Пример

$S_1 = "cabac", S_2 = "babddc"$

$$M(0, j) = j;$$

$$M(i, 0) = i;$$

$$\underline{S_1[i] = S_2[j]}:$$

$$M(i, j) = \min( M(i-1, j-1), \\ M(i-1, j) + 1, \\ M(i, j-1) + 1 )$$

$$\underline{S_1[i] \neq S_2[j]}:$$

$$M(i, j) = \min( M(i-1, j) + 1, \\ M(i, j-1) + 1 ),$$

<b>c</b>	6	5	6	5	4
<b>d</b>	5	6	5	4	5
<b>d</b>	4	5	4	3	4
<b>b</b>	3	4	3	2	3
<b>a</b>	2	3	2	3	4
<b>b</b>	1	2	3	2	3
	0	1	2	3	4
		<b>c</b>	<b>a</b>	<b>b</b>	<b>c</b>

Ins(3, 'd');

Ins(3, 'd');



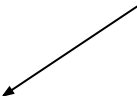
Del( 1 );

Ins(0, 'b');



Отметим, что решений в данной задаче может быть несколько.

Движение по таблице представлено ниже.

	вниз по $i$ -му столбцу из $j$ -ой строки в $j-1$ -ю	$\text{INS}(S_1, i, S_2[j])$	вставка после $i$ -й позиции в $S_1$ символа $S_2[j]$
	движение влево по $j$ -й строке из $i$ -го столбца в $i-1$ -й	$\text{DEL}(S_1, i)$	удаление $i$ -го символа в $S_1$ и передвижение на $i-1$ -ю позицию
	движение по диагонали влево вниз		перемещение текущей позиции в $S_1$ на один символ влево

# Задача о телефонном номере

Если вы обратили внимание, то клавиатура многих телефонов выглядит как показано →

Использование изображенных на клавишах букв позволяет представить номер телефона в виде легко запоминающихся слов. Многие фирмы пользуются этим и стараются подобрать себе номер телефона так, чтобы он содержал как можно больше букв из имени фирмы.

1	2 ABC	3 DEF
4 GHI	5 JKL	6 MN
7 PRS	8 TUV	9 WXY
	0 OQZ	

Напишите программу, которая преобразует исходный цифровой номер телефона в соответствующую последовательность букв и цифр, содержащую как можно больше символов из названия фирмы. При этом буквы из названия фирмы должны быть указаны в полученном номере в той же последовательности, в которой они встречаются в названии фирмы. Например, если фирма называется *IBM*, а исходный номер телефона — **246**, то замена его на ***BIM*** не допустима, тогда как замена его на ***2IM*** или ***B4M*** является правильной.

$S_1 = \text{"IBM"}, S_2 = \text{"246"}$ . При этом, если в  $S_1$  встречаются буквы, которые соответствуют цифрам номера телефона в нужном порядке, то они останутся без изменения.

## ***Рекомендации по решению:***

При решении задачи можно использовать только что рассмотренный алгоритм Ахо для преобразования строк.

Например, для приведённого примера про IBM, задайте строки  $S_1$  и  $S_2$  :  $S_1 = \text{"IBM"}$ ,  $S_2 = \text{"246"}$ . Обратите внимание, что операция INS будет вставлять цифры. При этом, если в встречаются буквы, которые соответствуют цифрам номера телефона в нужном порядке, то они останутся без изменений.

# Решение задачи о телефоне

1	2 ABC	3 DEF
4 GHI	5 JKL	6 MN
7 PRS	8 TUV	9 WXY
	0 OQZ	

6 : M N	3	2	3	2
4 : G Y I	2	1	2	3
2 : A B C	1	2	1	2
	0	1	2	3
		I	B	M

2    B    M  
I    B    M

↑  
↑

[illegible]

$S_1$	ffadac
$S_2$	dcbea

Оно равно \_\_\_\_\_

## Пример 10. Задача о расстановке скобок

Рассмотрим вычисление произведения  $n$  матриц

$$M = M_1 \times M_2 \times \dots \times M_n. \quad (1)$$

Порядок, в котором матрицы перемножаются, может существенно сказаться на общем числе операций, требуемых для вычисления матрицы  $M$ , независимо от алгоритма, применяемого для умножения матриц.

Умножение матрицы размера  $[p \times q]$  на матрицу размера  $[q \times r]$  требует  $pqr$  операций.

## Пример 10. Задача о расстановке скобок

В чём суть представьте что у вас есть  $n$  матриц и они у вас записаны в каком то произведении  $A_1 \times A_2 \times A_3$  и так далее до  $A_n$ . При этом, понятно что перемножать можно не любые две матрицы, а нужно чтобы матрицы имели правильный размер, например:

матрица  $A_i$  имеет размер  $p_i \times q_i$ ,

матрица  $A_{i+1}$  имеет размер  $p_{i+1} \times q_{i+1}$ , то можно перемножать только тогда когда  $q_i$  равно  $p_{i+1}$

Будем считать что это условие всегда выполнено. Тем не менее, поскольку размеры матриц все разные, самом деле, от того в каком порядке мы будем их перемножать, результат **не будет зависеть, потому что нас умножение матрицы ассоциативным**, и всё всегда будет хорошо, а вот сложность умножения будет зависеть очень сильно, почему потому что, если у нас есть матрицы  $A$  и  $B$ .  $A$  размером  $p \times q$ ,  $B$  размером  $q \times r$ . Сложность их умножения =  $p \times q \times r$ .

## Пример 10. Задача о расстановке скобок

• Тогда матрица  $C$  (произведение матриц  $A$  и  $B$ ), т.е.  $C=A*B$ . Это значит  $C_{i,j} = \sum_{k=1}^q a_{ik} * b_{kj}$ . Здесь у нас примерно  $2*q$  операций. Таких элементов  $C_{i,j}$  у нас  $p*r$  штук. Итого сложность умножения выходит порядка  $p*q*r$ . Отсюда мы можем сделать вывод, что сложность зависит от того в каком порядке мы перемножаем матрицы.

Например:  $C=A_1*A_2*A_3$ .

$A_1$  имеет размер  $[2 \times 50]$ ,  $A_2$  имеет размер  $[50 \times 2]$ ,  $A_3$  имеет размер  $[2 \times 50]$ .

- Если умножать  $(A_1*A_2)*A_3 \Rightarrow$  размер матрицы в скобках получится  $2 \times 2$ , а сложность умножения  $\Rightarrow 2*50*2$ . Потом умножаем результирующую матрицу на  $A_3$  это прибавит сложность  $2*2*50$ . Итого:  $200 + 200 = 400$  действий.
- Если умножать  $A_1*(A_2*A_3) \Rightarrow$  размер матрицы в скобках получится  $50 \times 50$ , а сложность умножения  $\Rightarrow 50*2*50$ . Потом умножаем результирующую матрицу на  $A_1$  это прибавит сложность  $2*50*50$ . Итого:  $5000 + 5000 = 10000$  действий.

В 25 раз больше! А чем больше перемножаем матриц и чем больше их размеры – тем больше получится эта разница.



## Пример 10. Задача о расстановке скобок

- Часто, в практических задачах, вот даже те затраты, которые по времени, которые тратятся на то чтобы определить оптимальный порядок, они полностью компенсируются выгодой, которую мы получим при умножении, поэтому надо выбрать наилучший способ перемножения, то есть такое, что бы сложность была наименьшая, но если бы мы решали полным перебором, то надо было бы прибрать возможные расстановки скобок в выражении  $A_1A_2\dots A_n$ . Там была бы  $n-1$  пара скобок. В дискретной математике это изучается, но это очень большое число (у него даже имя есть – число Каталана), что-то типа  $\frac{4^n}{n^2}$ , т.е. порядок сильно экспоненциальный. Следовательно полный перебор – крайне не эффективен!

Вернёмся к динамическому программированию и будем решать задачу оптимальной расстановки скобок для произведения от  $A_i * \dots * A_j$ . Пусть  $\text{compl}(i,j)$  – оптимальное количество действий для такого произведения. Тогда ответом будет искомое число.

## Пример 10. Задача о расстановке скобок

Это как  $\text{compl}(1, n)$  как раз это будет целая последовательность, ну и надо научиться считать  $\text{compl}(i, j)$ .

Давайте заметим кое-что. Как вообще выглядит расстановка скобок. Она выглядит так, что у нас есть какие-то члены от  $A_i$  и  $A_k$  какого-нибудь. Там тоже стоят какие-то скобки  $(A_i * \dots * A_k)$ , но главное, что вот выражение  $A_{ij}$ , самое внешне умножение, стоит между элементами  $A_k$  и  $A_{k+1}$ . Т.е.  $(A_i * \dots * A_k) (A_{k+1} * \dots * A_j)$ . Т.е. здесь стоит тоже внешнее умножение, а тут тоже какие-то расстановки скобок есть. Понятно, что если у нас оптимальная расстановка скобок будет у всего большого выражения, то и у под выражений тоже расстановки скобок - минимальны. Т.е. у первой группы скобок  $\text{compl}(i, k)$ , а у второй группы скобок -  $\text{compl}(k+1, j)$ . Тогда, чтобы найти  $\text{compl}(i, j)$  (а ищем по всем  $k$ , от  $i$  (если первая скобка состоит из одного элемента), до  $j-1$  (если вторая скобка состоит из одного элемента)). Т.е.  $i \leq k \leq j-1$ .

$\text{compl}(i, j) = \min(\text{compl}(i, k) + \text{compl}(k+1, j)) + \text{затраты на вычисление } 1\text{-й и } 2\text{-й скобки (это } p_i * p_{k+1} * p_{j+1})$

## Пример

Рассмотрим произведение матриц:

$$M = M_1 \times M_2 \times M_3 \times M_4$$
$$[10 \times 20] \quad [20 \times 50] \quad [50 \times 1] \quad [1 \times 100]$$

Если вычислять матрицу  $M$  в порядке:  $M_1 \times (M_2 \times (M_3 \times M_4))$ ,  
то

потребуется 125 000 операций.

$$(50 \times 1 \times 100) \rightarrow [50 \times 100], \quad 5000;$$

$$(20 \times 50 \times 100) \rightarrow [20 \times 100], \quad 100000;$$

$$(10 \times 20 \times 100) \rightarrow [10 \times 100], \quad 20000.$$

Вычисление же в порядке:  $(M_1 \times (M_2 \times M_3)) \times M_4$  требует лишь  
2 200 операций.

$$(20 \times 50 \times 1) \rightarrow [20 \times 1], \quad 1000;$$

$$(10 \times 20 \times 1) \rightarrow [10 \times 1], \quad 200;$$

$$(10 \times 1 \times 100) \rightarrow [10 \times 100], \quad 1000.$$

Перебор с целью минимизировать число операций имеет экспоненциальную сложность.

На первом этапе определим за какое минимальное количество операций можно получить матрицу  $M$  из равенства (1).

Будем считать подзадачами вычисление минимального количества операций при перемножении меньшего, чем  $n$ , количества матриц. В качестве параметров рассматриваемой задачи возьмем индексы  $i$  и  $j$  ( $1 \leq i \leq j \leq n$ ), обозначающие номера первой и последней матриц в цепочке

$$M_i \times M_{i+1} \times \dots \times M_j.$$

Сначала решим подзадачи, когда  $j = i+1$ , т.е. когда перемножаются

две рядом стоящие матрицы.

Решения – количество затраченных операций, запишем в ячейке таблицы  $T$  с номерами  $(i, j)$ .

$T_{ij}$  – число, равное количеству операций при умножении цепочки матриц  $M_i \times \dots \times M_j$ , где  $1 \leq i \leq j \leq n$ .

Обозначим через  $t_{ij}$  минимальную сложность вычисления цепочки матриц  $M_i \times M_{i+1} \times \dots \times M_j$ , где  $1 \leq i \leq j \leq n$ .  
Ее можно получить следующим образом:

$$t_{ij} = \begin{cases} 0, & \text{если } i = j \\ \min_{i \leq k < j} (t_{ik} + t_{k+1,j} + r_{ikj}), & \text{если } j > i \end{cases}$$

Здесь  $t_{ik}$  — минимальная сложность вычисления цепочки

$$M' = M_i \times M_{i+1} \times \dots \times M_k,$$

а  $t_{k+1,j}$  — минимальная сложность вычисления цепочки

$$M'' = M_{k+1} \times M_{k+2} \times \dots \times M_j.$$

Третье слагаемое  $r_{ikj}$  равно сложности умножения  $M'$  на  $M''$ .  
Утверждается, что  $t_{ij}$  ( $j > i$ ) — наименьшая из сумм этих трех членов по всем возможным значениям  $k$ , лежащим между  $i$  и  $j$  - 1.

Для примера из четырех матриц в таблице будут определены

следующие элементы  $T$ :  $t_{1,2}$ ,  $t_{2,3}$  и  $t_{3,4}$ .

0	10000		
	0	1000	
		0	5000
			0

$M_1 \times M_2 \times M_3 \times M_4$   
 $[10 \times 20] \quad [20 \times 50] \quad [50 \times 1] \quad [1 \times 100]$

$$t_{ij} = \begin{cases} 0, & \text{если } i = j \\ \min_{i \leq k < j} (t_{ik} + t_{k+1,j} + r_{ikj}), & \text{если } j > i \end{cases}$$

Далее перейдем к решению подзадач с параметрами  $j = i + 2$ .

Рассмотрим, например, цепочку матриц  $M_1 \times M_2 \times M_3$ .

Решением этой подзадачи будет минимальное количество операций,

выбранное из двух возможных порядков перемножения матриц:  $(M_1 \times M_2) \times M_3$  и  $M_1 \times (M_2 \times M_3)$ . При этом для выражений в скобках ответы уже записаны в таблице  $T$ . Результат запишем в ячейку  $T_{1,3}$ .

Затем перейдем к решению подзадач с параметрами  $j = i + 3$  и т.д.

Итак,  $t_{ij}$  вычисляются в порядке возрастания разностей нижних индексов. Процесс начинается с вычисления  $t_{ij}$  для всех  $i$ , затем  $t_{i,i+1}$  для всех  $i$ , потом  $t_{i,i+2}$  и т. д. При этом  $t_{ik}$  и  $t_{k+1,j}$  будут уже вычислены, когда мы приступим к вычислению  $t_{ij}$ .  
Оценка сложности данного алгоритма  $O(n^3)$ .

В результате работы алгоритма для примера из четырех матриц будет построена следующая таблица  $T$ :

Порядок, в котором можно произвести эти умножения, легко определить, приписав каждой клетке то значение  $k$ , на котором достигается минимум.

0	10000	1200	2200
	0	1000	3000
		0	5000
			0

# Алгоритм

```
for (i=0; i<n; i++)    mi,i = 0;
for (l=1; l<n; l++)
    for (i=0; i<n; i++) {
        j = i + l;
        for (k = 0; k < j; k++)
            mij = min(mi,k + mk+1,j + ri-1*rk* rj)
    }
```

$r_{i-1}$  – количество строк в  $M'$

$r_k$  – количество столбцов в  $M'$

$r_j$  – количество столбцов в  $M''$



# Упражнение

Задана строка, состоящая из вещественных чисел,  
разделенных  
арифметическими операциями.

Требуется расставить в строке скобки таким образом,  
чтобы  
значение полученного выражения было максимальным.