Занятие №9. Двумерные массивы

Известно, что в таблице, заполненной нулями и единицами, **i-й** столбец состоит целиком из единиц, а **i-я** строка из нулей (на их пересечении может быть что угодно).

Как найти **i**, сделав всего лишь N сравнений? (Ответ смотрите в Подсказке 9.1)

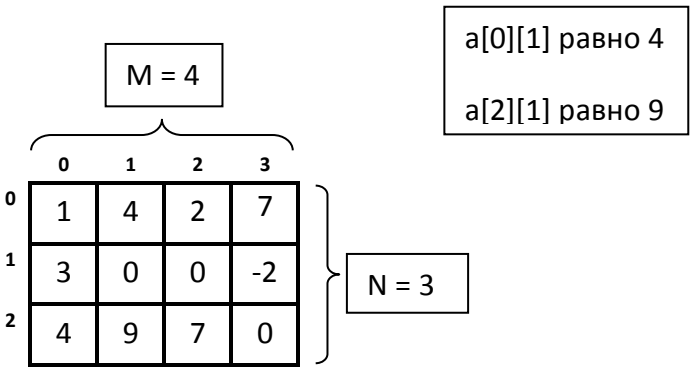
Двумерные массивы - это массивы. Только… двумерные.

Они предназначены для хранения и обработки таблиц. Обычно прямоугольных (на математическом языке их называют «*матрицами*»). Но мы покажем, что в Java можно легко создать и использовать «неровные», например, треугольные массивы. Это бывает удобно, например, для хранения графов в виде списков смежности, как мы увидим на одном из следующих занятий.

При работе с двумерными массивами у ячеек две координаты. Конечно, это больше вопрос восприятия, но обычно принято указывать сначала номер строки (мы привыкли к обозначению «y»), а потом уже номер столбца («x»). Количество строк обычно обозначают буквой N, а столбцов – M.

Двумерные массивы в Java – объекты. Поэтому создавать их следует при помощи команды new:

```
int a[3][4] = new int[3][4];
```



Работа с двумерными массивами очень напоминает работу с одномерными. Для обработки часто используют вложенные циклы (цикл по сторокам, а в нем цикл по столбцам).

Рассмотрим типичные задачи, которые обычно возникают при работе с двумерными массивами.

Создание и «стандартное» чтение

Обычно сначала задается размер массива, потом сами элементы.

```
int N = 0, M = 0;
N = in.nextInt();
M = in.nextInt();
// Массив в Java – это объект, поэтому его
// следует создавать командой new
int[][] a = new int[N][M];
for (int y = 0; y < N; y++)
{
     for (int x = 0; x < M; x++)
     {
          ________________
     }
}
```

Обратите внимание: внешний цикл идет по строкам, а внутренний по элементам строк.

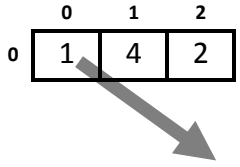
Вывод массива в виде таблицы

```
for (int y = 0; y < N; y++)
{
     for (int x = 0; x < M; x++)
     {
          out.print (a[y][x] + " "); // разделяем элементы
пробелами
     }
     ____________________
}
```

Сумма всех элементов

```
int s = 0;
for (int y = 0; y < N; y++)
{
     for (int x = 0; x < M; x++)
     {
          s += a[y][x];
     }
}
```

Сумма элементов главной диагонали



1	3	0	0
2	4	9	7

Если массив квадратный – размером NxN.

Часто приходится видеть у начинающих такой код «по образцу»:

```
int s = 0;
for (int y = 0; y < N; y++)
{
     for (int x = 0; x < M; x++)
     {
          if (x == y) s += a[y][x];
     }
}
```

Но в использовании вложенных циклов при решении этой задачи нет необходимости:

```
int s = 0;
for (int i = 0; i < M; i++)
{
     s += a[_][_];
}
```

Неровные массивы

Часто бывает, что прямоугольная таблица – это нерациональный раход в памяти. Скажем, мы знаем, что первая строка должна и будет содержать 7 элементов, вторая всего 2, а третья 10. Делать массив на 30 элементов расточительно. На Java можно легко для данной задачи создать массив ровно на 19 элементов и обрабатывать его почти стандартным способом.

Дело в том, что двумерный массив это – массив массивов. А так как массивы в Java - это объекты – никаких проблем:

```
int a[3][] = new int[3][]; // в нашем массиве будет 3 строки
a[0] = new int [7];// a[0], a[1] и a[2]
a[1] = new int[2]; // станут соответствующими
a[2] = new int[10] // массивами желаемого размера.
```

Для общего случая это можно сделать циклом.

Создадим, скажем, треугольный массив на N строк. В первой строки один элемент, во второй два и т.д.

```
int a[N][] = new int[N][]; // вторые скобки оставляем пустыми!
for (int i = 0; i < N; i++)
{
     a[i] = new int[i];
}
```

Свойство **length** позволяет обрабатывать эту структуру двумя вложенными циклами. Например, выведем все элементы:

```
for (int y = 0; y < a.length; y++) //a.length – количество строк
{
     for (int x = 0; x < a[y].length; x++)
     // a[y].length – количество элементов в строках
     {
          out.print (a[y][x] + " ");
     }
     out.println();
}
```

Задача «Треугольник Паскаля» решается даже при помощи одномерных массивов. Но для того, чтобы попрактиковаться, рекомендуется ее решать именно неровным – треугольным массивом.

Задачи

Практически во всех задачах этого занятия большой объем входных данных. Поэтому лучше использовать для ввода-вывода файлы. Подробно об этом написано в тексте Занятия 5 на стр. Ошибка! Закладка не определена. и Справочнике на стр. Ошибка! Закладка не определена..

Задача A. Обороноспособность

Король Квадратландии прекрасно знает, сколько солдат патрулируют каждый квадрат своей квадратной страны.

Теперь он желает оценить обороноспособность Квадратландии. Хорошо известно, что она зависит от количества солдат, которые патрулируют границу.

Он поручает эту непростую задачу вам, своему придворному счетоводу.

Посчитайте общее количество солдат, патрулирующих пограничные квадраты.

Формат входных данных

На первой строке величина Квадратландии - число квадратов по одной стороне. (1 < N < 100)

Дальше следуют N строк по N чисел в каждой - количество солдат, которые охраняют каждый квадрат.

Выходные данные

Одно число – обороноспособность страны.

Примеры

Ввод	**Вывод**
1 3	3
3 1 2 3 3 4 7 0 2 3	21
5 1 1 1 1 1 3 4 7 7 1 0 2 3 1 1 1 1 1 1 1 1 1 1 1 1	17

Задача B. Координаты соседей

Для клетки с координатами (*x*, *y*) в таблице размером *M* × *N* выведите координаты ее соседей. Соседними называются клетки, имеющие общую сторону.

(**Координаты** элементов таблицы **нумеруются с единицы**)

Формат входного файла

Даны натуральные числа *M*, *N*, *x*, *y* ($1 \le x \le M \le 10^9$, $1 \le y \le N \le 10^9$).

Формат выходного файла

В выходной файл выведите пары координат соседей этой клетки в произвольном порядке.

Пример

Ввод	**Вывод**
3 3 2 2	2 1 1 2 2 3 3 2

Задача C. Симметричная ли матрица?

Проверьте, является ли двумерный массив симметричным относительно главной диагонали. Главная диагональ — та, которая идёт из левого верхнего угла двумерного массива в правый нижний.

Формат входных данных

Программа получает на вход число `n ≤ 100`, являющееся числом строк и столбцов в массиве. Далее во входном потоке идет `n` строк по `n` чисел, являющихся элементами массива.

Формат выходных данных

Программа должна выводить слово `yes` для симметричного массива и слово `no` для несимметричного.

Пример

Ввод	Вывод
3 0 1 2 1 2 3 2 3 4	yes
3 0 1 2 1 8 4 2 4 9	yes
3 0 1 2 1 8 3 2 4 9	no
3 0 1 2 4 3 1 5 4 0	no
5 0 1 2 3 4 1 5 9 8 7 2 9 6 0 0 3 8 0 0 0 4 7 0 0 0	yes
5 0 1 2 3 4 1 5 9 8 7 2 9 6 0 0 3 8 0 0 0 4 7 0 5 0	no

Задача D. Заполнение змейкой

Даны числа **n** и **m**. Создайте массив **A[n][m]** и заполните его змейкой (см. пример).

Формат входных данных

Программа получает на вход два числа **n** и **m**.

Формат выходных данных

Программа должна вывести полученный массив, отводя на вывод каждого числа ровно 3 символа.

Пример

Ввод	**Вывод**
4 6	0 1 2 3 4 5 11 10 9 8 7 6 12 13 14 15 16 17 23 22 21 20 19 18

Задача E. Заполнение диагоналями

Даны числа **n** и **m**. Создайте массив **A[n][m]** и заполните его, как показано на примере.

Формат входных данных

Программа получает на вход два числа **n** и **m**.

Формат выходных данных

Программа должна вывести полученный массив, отводя на вывод каждого числа ровно 3 символа.

Пример

Ввод	**Вывод**
4 6	0 1 3 6 10 14 2 4 7 11 15 18 5 8 12 16 19 21 9 13 17 20 22 23

Задача F. Заполнение спиралью

Дано число **n**. Создайте массив **A[2*n+1][2*n+1]** и заполните его по спирали, начиная с числа 0 в центральной клетке **A[n+1][n+1]**. Спираль выходит вверх, далее закручивается против часовой стрелки.

Формат входных данных

Программа получает на вход одно число **n**.

Формат выходных данных

Программа должна вывести полученный массив, отводя на вывод каждого числа ровно 3 символа.

Пример

Ввод	**Вывод**
2	12 11 10 9 24 13 2 1 8 23 14 3 0 7 22 15 4 5 6 21 16 17 18 19 20

Пожалуйста, не надо спрашивать, «Что за два?!». Прочитайте условие еще раз внимательно!

Задача G. Сапер

Источник: Заочный тур Всероссийской открытой олимпиады 2004 года

Мальчику Васе очень нравится известная игра "Сапер" ("Minesweeper").

В "Сапер" играет один человек. Игра идет на клетчатом поле (далее будем называть его картой) *N*x*M* (*N* строк, *M* столбцов). В *K* клетках поля стоят мины, в остальных клетках записано либо число от 1 до 8 — количество мин в соседних клетках, либо ничего не написано, если в соседних клетках мин нет. Клетки являются соседними, если они имеют хотя бы одну общую точку, в одной клетке не может стоять более одной мины. Изначально все клетки поля закрыты. Игрок за один ход может открыть какую-нибудь клетку. Если в открытой им клетке оказывается мина — он проигрывает, иначе игроку показывается число, которое стоит в этой клетке, и игра продолжается. Цель игры — открыть все клетки, в которых нет мин.

У Васи на компьютере есть эта игра, но ему кажется, что все карты, которые в ней есть, некрасивые и неинтересные. Поэтому он решил нарисовать свои. Однако фантазия у него богатая, а времени мало, и он хочет успеть нарисовать как можно больше карт. Поэтому он просто выбирает *N*, *M* и *K* и расставляет мины на поле, после чего все остальные клетки могут быть однозначно определены. Однако на определение остальных клеток он не хочет тратить свое драгоценное время. Помогите ему!

По заданным *N*, *M*, *K* и координатам мин восстановите полную карту.

Формат входных данных

В первой строке входного файла содержатся числа N, M и K ($1 \le N \le 200$, $1 \le M \le 200$, $0 \le K \le N \cdot M$). Далее идут K строк, в каждой из которых содержится по два числа, задающих координаты мин. Первое число в каждой строке задает номер строки клетки, где находится мина, второе число — номер столбца. Левая верхняя клетка поля имеет координаты (1,1), правая нижняя — координаты (N,M).

Формат выходных данных

Выходной файл должен содержать N строк по M символов — соответствующие строки карты. j-й символ i-й строки должен содержать символ ‘*‘ (звездочка) если в клетке (i,j) стоит мина, цифру от 1 до 8, если в этой клетке стоит соответствующее число, либо ‘.‘ (точка), если клетка (i,j) пустая.

Пример

Входные данные	Выходные данные
10 9 23 1 8 2 3 3 2 3 3 4 3 5 7 6 7 7 1 7 2 7 3 7 4 7 5 7 6 7 7 7 8 8 1 8 3 8 5 8 7 9 3 9 5 9 6 9 7	.111..1*1 13*2..111 1**3..... 13*2.111. .111.2*2. 233335*41 ********1 *6*7*8*41 13*4***2. .1122321.

Задача H. Робот К-79

Источник: Заочный тур Всероссийской открытой олимпиады 2004 года

Петя написал программу движения робота К-79. Программа состоит из следующих команд:

S — сделать шаг вперед

L — повернуться на 90° влево

R — повернуться на 90° вправо

Напишите программу, которая по заданной программе для робота определит, сколько шагов он сделает прежде, чем впервые вернется на то место, на котором уже побывал до этого, либо установит, что этого не произойдет.

Формат входных данных

Во входном файле записана одна строка из заглавных латинских букв `S`, `L`, `R`, описывающая программу для робота. Общее число команд в программе не превышает 200, при этом команд `S` — не более 50.

Формат выходных данных

В выходной файл выведите, сколько шагов будет сделано (то есть выполнено команд `S`) прежде, чем робот впервые окажется в том месте, через которое он уже проходил. Если такого не произойдет, выведите в выходной файл число `-1`.

Примеры

Входные данные	**Выходные данные**
`SSLSLSLSSRSRS`	`5`
`LSSSS`	`-1`

Задача I. Квадрат

Источник: Личная московская олимпиада 2003-2004 учебного года

Требуется в каждую клетку квадратной таблицы размером *N*x*N* поставить ноль или единицу так, чтобы в любом квадрате размера *K*x*K* было ровно *S* единиц.

Формат входных данных

Во входном файле записаны три числа — *N*, *K*, *S* ($1 \le N \le 100$, $1 \le K \le N$, $0 \le S \le K^2$).

Формат выходных данных

В выходной файл выведите заполненную таблицу. Числа в строке должны разделяться пробелом, каждая строка таблицы должна быть выведена на отдельной строке файла. Если решений несколько, выведите любое из них.

Примеры

Входные данные	**Выходные данные**

`3 2 1`	`0 0 0`
	`0 1 0`
	`0 0 0`

4 2 2	1 0 0 1 0 1 1 0 1 0 0 1 0 1 1 0

Задача J. Треугольник Паскаля

Треугольник Паскаля строится следующим образом. Первая строка состоит из одной единицы. Каждая следующая содержит на одно число больше, чем предыдущая. Первое и последнее из этих чисел равны 1, а все остальные вычисляются как сумма числа, стоящего в предыдущей строке над ним, и числа, стоящего в предыдущей строке слева от него.

По введенному $n \leq 30$ выведите n первых строк треугольника Паскаля.

Ввод	**Вывод**
5	1 1 1 1 2 1 1 3 3 1 1 4 6 4 1

Подсказки и решения. 9 Занятие.

9.1

Будем «двигаться» по таблице следующим образом.

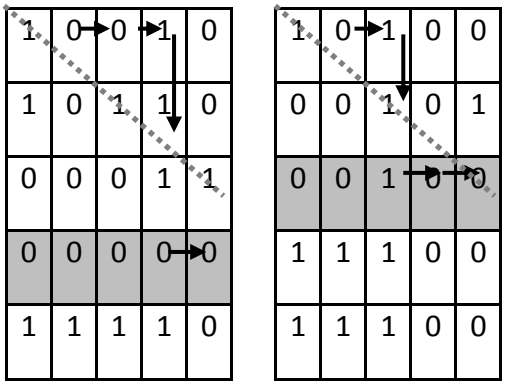
Начнем с клетки (1, 0).

Если в клетке 0 – двигаемся вправо.

Если 1 – спускаемся вниз на диагональ и снова идем вправо.

И так делаем, пока не достигнем края.

Пример 1 *Пример 2*



Ответом будет номер строки, в которой мы остановились.

Почему это работает?

Удивительно, что этот алгоритм имеет непосредственное отношение к… сортировке массива!

Представим себе содержимое таблицы результатами сравнений N камней между собой. Тогда i-й столбец и i-я строка соответствуют самому тяжелому камню. Но ведь максимум можно найти за N-1 сравнение! Начнем с i равного 0. Если в клетке с координатами (0, 1) стоит 1 – проверяем клетку (0, 2), если же там стоит 0 – меняем i на 1 и дальше проверяем (1, 2)… и так далее.

Заметим, что если таблица не симметрична относительно главной диагонали, в результате каждой проверки один из камней все равно отбрасывается.

9.2

Потому, что, скажем, пара `a[2][3]` и `a[3][2]`, проверяется первый раз, когда `i` = 2, `j` = 3, а второй – когда `i` = 3, `j` = 2. Исправить это можно так. Во внутреннем цикле «запускать» `j` со значения `i + 1`, как, например, при сортировке массива методом максимума:

```
for (int j = i+1; j < N; j++)
```

Разбор. Занятие 9

9A. Обороноспособность

Требуемую сумму можно посчитать, например, четырьмя циклами (первая, последняя строка, первый, последний столбец). Вот один из них:

```
for (int x = 0; x < N; x++)
{
     s += a[N-1][x];
}
```

А потом не забыть вычесть четыре угла – мы же их посчитали дважды. Можно и сразу считать без углов.

А можно и просто пройтись по всей матрице, увеличивая s, если проходим граничный квадрат:

// как обычно, в двух вложенных циклах

```
if (x == 0 || y == 0 || x == M - 1 || y == N - 1)
{
     s += a[y][x];
}
```

Это, конечно, совсем неэффективно, но очень просто. Ограничения в условии вполне позволяют так делать.

9B. Координаты соседей

Конечно, массив в решении этой задачи заводить не нужно. Да и невозможно (в худшем случае получится миллиард в квадрате ячеек, спросите у Google, как называется это число и попробуйте перевести его в… терабайты!). Но решение научит вас хорошо ориентироваться на координатной плоскости, что очень важно при работе с двумерными массивами.

Единственная сложность в этой задаче – прописать граничные условия. Ведь у крайних клеток не четыре соседа, а три или два (для угловых)!

Прописывается это примерно так

```
if (x + 1 <= M) out.println(x + 1 + " " + y) // правый сосед
```

Обратите особое внимание на условие. Нестрогое равенство здесь не типично для программ на Java. Но по условию этой задачи координаты нумеруются с единицы.

9C. Симметричная ли матрица?

Поймем, что такое «симметричные» элементы. У элементов квадратной матрицы, симметричных относительно главной диагонали, координаты «наоборот». Элементу с координатами i, j соответствует элемент с координатами j, i. Соответственно пройдемся по всей матрице стандартным способом и проверим, для всех ли a[i][j] == a[j][i]. Если для каких-то элементов это не выполнится, «поднимем флажок»: логической переменной **simm**, которая изначально истинна, то есть матрица считается пока симметричной, придадим значение false.

```
boolean simm = true;
// матрица считается симметричной, пока не установлено обратное
for (int i = 0; i < N; i++)
     for (int j = 0; j < N; j++)
     {
          if (a[i][j] != a[j][i])
          {
               simm = false; // оказалась несимметричной
          }
     }
}
```

и после цикла выводим **yes** или **no** в зависимости от значения переменной **simm**.

Заметим, что стоит сразу выходить из циклов - «хуже уже не будет». Это сделать просто, каждое из условий циклов дополним проверкой. Например,

```
for (int j = 0; j < N && simm; j++)
```

Еще один недостаток нашего кода в том, что каждая пара проверяется дважды. Почему, как это исправить? (См. Подсказку 9.2)

9D. Заполнение змейкой

Стоит заметить, что змейка меняет свое направление следующим образом: если номер строки четное число, то заполнение происходит от 0 столбца до m если номер строки нечетное число, то заполнение происходит от m до 0.

Заведем переменную k, которую будем постоянно увеличивать. А внутренний цикл будем совершать либо слева-направо, если i четно, либо наоборот. Нагляднее всего, хотя и довольно громоздко, это записать при помощи двух циклов:

```
int k = 0;
for (int i = 0; i < N; i++)
{
     if(i % 2 == 0)
     {
          for (int j = 0; j < M; j++) // направо
          {
```

```
                a[i][j] = k;
                k++;
           }
      }
      else
      {
           for (int j = M - 1; j >= 0; j--) // налево
           {
                a[i][j] = k;
                k++;
           }
      }

}
```

Попробуйте обойтись без двух внутренних циклов!

9E. Заполнение диагоналями

Это задача сложнее, чем предыдущая. Иногда действительно бывает необходимо заполнять массив именно таким образом. Мы встретимся с такими случаями, например, при обсуждении задач динамического программирования в одном из следующих модулей.

Заполнение можно делать по-разному. Например, во внутреннем цикле по-диагонали до края:

```
int k = 0;
for (int i = 0; i < N; i++)
     x = i;
     for (int y = 0; y < M && x >= 0;/*шаг - в теле цикла*/)
     {
          x--; y++; // движемся по диагонали
          a[y][x] = k;
          k++;
     }
}
```

Так будет заполнена верхняя-левая треугольная часть таблицы. Для заполнения нижней-правой можно сделать аналогичные вложенные циклы, сдвигая `y` во внешнем цикле.

В условии этой и следующей задачи определен строгий формат вывода – по три символа на число. Это можно сделать, например, при помощи условного оператора, выводя допонительный пробел для однозначных чисел:

```
if (x !=0 && a[y][x] < 10) out.print(" ");
out.print(a[y][x]+ " ");
```

9F. Заполнение спиралью

Это самая сложная задача из мини-цикла на заполнение. Вам предлагается решить ее полностью самостоятельно.

Скажем лишь, что один замечательный учитель программирования ставил своим ученикам за эту задачу 6 – N пятерок, где N – это количество циклов в программе (фрагменте заполнения). О том, как получить шесть пятерок, вы узнаете в следующем модуле!

Задачи G, H и I с олимпиад. Приведем их разборы с некоторыми сокращениями по книге «Московские олимпиады по информатике» и с переводом программных текстов на язык Java.

9G. Сапер

Для того чтобы решить эту задачу, мы воспользуемся двумерным массивом «`a`», представляющим игровое поле. Изначально мы заполним его нулями, а затем для каждой мины будем увеличивать на 1 числа во всех соседних с ней клетках. В саму же клетку, где стоит мина, мы будем записывать какое-нибудь большое число («бесконечность»). Вот как может выглядеть соответствующий фрагмент программы:

```
i = in.nextInt(); // читаем из файла координаты
j = in.nextInt(); // очередной мины
a[i-1][j-1]++; a[i-1][j]++; a[i-1][j+1]++;
a[i][j-1]++; a[i][j+1]++;
a[i+1][j-1]++; a[i+1][j]++; a[i+1][j+1]++;
a[i][j]= 100;
```

Таким образом, когда мы обработаем все мины из входного файла, мы получим массив, в котором числа, не превосходящие 8, будут означать количество мин по соседству с данной клеткой, а числа от 100 и больше — клетки, в которых стоят мины. Останется лишь распечатать полученную схему:

```
for (int i = 1; i <= N; i++)
{
     for (int j = 1; j <= N; j++)
     {
          if (a[i][j] > 8) out.print('*')
          else if (a[i][j] == 0) out.print('.')
               else out.print(a[i][j]);
          out.println();// переводим строку
     }
}
```

Осталось обсудить несколько тонкостей.

1. Мы никак отдельно не обрабатываем случай, когда мина стоит на границе поля и у нее меньше восьми соседних клеток. Чтобы упростить программу, мы воспользуемся

стандартным трюком: увеличим поле со всех сторон на 1 ряд, т.е. будем использовать массив, индексы которого изменяются от 0 до M + 1 и от 0 до N + 1 соответственно. *(В Java и так индексы нумеруются с нуля, поэтому нужно только заводить массив на одну строку и один столбец больше.)* При этом в выходной файл мы выводим только нужную нам часть массива.

2. Код, который мы написали для увеличения чисел в клетках, не выглядит изящным. Можно заменить его, например, на такой:

```
for (dy = -1; dy <= 1; dy++)
{
     for (dx = -1; dx <= 1; dx++)
     {
          a[i + dy] [j + dx]++;
     }
     a[i][j] = 100;
}
```

Может показаться, что выбор из этих двух вариантов есть лишь вопрос вкуса, но на самом деле первый вариант таит в себе большую опасность сделать опечатку, которую непросто будет найти.

Автор разбора — В. М. Гуровиц

9H. Робот К-79

Представим себе, что робот ходит по большой шахматной доске, каждым ходом перемещаясь на одну клетку влево, вправо, вперед или назад. Мы будем хранить в двумерном массиве путь робота и после каждого очередного хода проверять, не попал ли робот в клетку, в которой он уже был раньше. Чтобы отслеживать передвижения робота, нам нужно в каждый момент времени знать, в какую сторону он смотрит. Обозначим направления цифрами: 0 — направо, 1 —вперед, 2 — налево, 3 — назад. Тогда, если мы храним в переменной dir текущее направление движения, то поворот налево будет соответствовать прибавлению к dir единицы по модулю 4:

```
dir = (dir + 1) % 4; // 0->1, 1->2, 2->3, 3->0
```

Поворот направо, аналогично, будет соответствовать вычитанию единицы, или что то же прибавлению 3, по модулю 4:

```
dir = (dir + 3) % 4;
```

Заведем два массива:

```
int[] dx = new int[]{1, 0, -1, 0};
int[] dy = new int[]{0, -1, 0, 1};
```

Их смысл в следующем: если **`dir`** — текущее направление движения робота, то за один ход его координата x изменяется на **`dx[dir]`**, а координата y на **`dy[dir]`**.

После этого несложно написать требуемую программу.

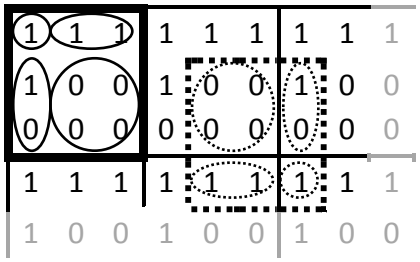
Автор разбора — В. М. Гуровиц

9I. Квадрат

Придумать решение этой задачи не просто.

Раскроем секрет: достаточно сгенерировать любой квадрат K x K, содержащий ровно S единиц, а затем просто заполнить такими квадратами квадрат N x N.

Поясним идею картинкой:



В квадратах-копиях – на рисунке только один такой, справа от оригинала, - сумма та же. Но и в любых других квадратах тоже, так как каждый из них будет состоять из частей «оригинала». Действительно, когда мы «сдвигаем» квадрат, то «теряем» одну сторону, но «приобретаем» ее с другого конца. Квадрат, выделенный на рисунке пунктиром, состоит из тех же частей, что и оригинал, выделенный жирной границей.

Таким образом, алгоритм будет состоять из двух шагов.

Шаг первый. Необходимо получить любой квадрат размером K x K, в котором будет S единиц. Сделаем это каким-либо простым способом. Например, сначала заполним квадрат нулями. Затем будем проходить его по строкам и ставить единицы до тех пор, пока не поставим столько, сколько нам нужно. Если за полный проход по квадрату нам так и не удалось поставить S единиц, то это значит, что задача не имеет решения. Такой случай возможен только при $S > K^2$, а это противоречит условию.

Шаг второй. Распространим полученный квадрат размером K x K на искомый квадрат N x N.

9J. Треугольник Паскаля

Эта задача расположена последней не потому, что она сложная, а потому что ее следует попробовать сделать «неровными» массивами.

Создаем массив массивов

```
int a[][] = new int[N + 1][];
```

Создаем первую строку из одного элемента и закладываем в нее 1.

```
a[0] = new int[1];
a[0][0] = 1;
```

А дальше во внутреннем цикле по строкам делаем так. Создаем очередную строку, заполняем первый и последний элементы единицами, а «средние» по условию, по формуле `a[i][j] = a[i - 1][j -1] + a[i - 1][j]`

Получается примерно такой код:

```
for (int i = 1; i <= N; i++)
{
     a[0] = new int[i + 1];
     a[i][0] = 1;
     a[i][i - 1] = 1;
     for (int j = 1; j < i - 1; j++)
     {
          a[i][j] = a[i - 1][j -1] + a[i - 1][j];
     }
}
```

В конце выводим массив стандартным способом, как описано в занятии.

Заметим, что Java-программисты, имеющие в своем распоряжении мощный BigInteger могут решать эту задачу со значительно большими ограничениями на **N**.

Занятие 9.Справочник

Создание и «стандартное» чтение двумерного массива

Обычно сначала задается размер массива, потом сами элементы.

```
int N = 0, M = 0;
N = in.nextInt();
M = in.nextInt();
// Массив в Java – это объект, поэтому его
// следует создавать командой new
int[][] a = new int[N][M];
for (int y = 0; y < N; y++)
{
     for (int x = 0; x < M; x++)
     {
          a[y][x] = in.nextInt();
     }
}
```

Обратите внимание: внешний цикл идет по строкам, а внутренний по элементам строк.

Вывод двумерного массива в виде таблицы

```
for (int y = 0; y < N; y++)
{
     for (int x = 0; x < M; x++)
     {
          // разделяем элементы пробелами
          out.print (a[y][x] + " ");
     }
     out.print ();// переводим на новую строку
}
```

Сумма всех элементов двумерного массива

```
int s = 0;
for (int y = 0; y < N; y++)
{
     for (int x = 0; x < M; x++)
     {
          s += a[y][x];
     }
}
```

Сумма элементов главной диагонали двумерного массива

Если массив квадратный – размером NxN

```
int s = 0;
for (int i = 0; i < N; i++)
{
     s += a[i][i];
}
```

Неровные двумерные массивы

Двумерный массив это - массив массивов. А массивы в Java - это объекты. Создадим, скажем, треугольный массив на N строк. В первой строки один элемент, во второй два и т.д.

```
int a[N][] = new int[N][]; // вторые скобки оставляем пустыми!
for (int i = 0; i < N; i++)
{
     a[i] = new int[i];
}
```

Свойство **length** позволяет обрабатывать эту структуру двумя вложенными циклами. Например, выведем все элементы:

```
for (int y = 0; y < a.length; y++) // a.length – количество
строк
{
     for (int x = 0; x < a[y].length; x++) // a[y].length –
                                           // количество
     {                                     // элементов в строках
          out.print (a[y][x] + " ");
     }
     out.println();
}
```