ЗАДАНИЕ ПО ИНФОРМАТИКЕ
ВАРИАНТ 73111 для 11 класса

Для заданий 1, 2, 4, 5 требуется разработать алгоритмы на языке блок-схем, псевдокоде или естественном языке

1. Для проверки, является ли большое целое простым, может использоваться вероятностный тест Леманна. Пусть p ≥ 5 – проверяемое нечётное число. Тогда:
   - случайно выбираем a: $2 \le a \le p - 2$;
   - вычисляем $r = a^{(p-1)/2} \{\mathrm{mod}\ p\}$;
   - если $r \neq 1$ и $r \neq p-1$, то p – составное.

   В тесте Леманна эти проверки выполняются для t случайно выбираемых a.
   Написать алгоритм проверки вводимого числа на простоту по тесту Леманна.
   Примечание: $x = y \{\mathrm{mod}\ n\}$, если существует целое k, для которого $x = y + k \cdot n$.

Решение. В цикле t раз выбираем число a в заданном диапазоне. Для каждого a вычисляем $r = a^{(p-1)/2} \{\mathrm{mod}\ p\}$. Для того чтобы при возведении в степень не получилось слишком большого числа (выходящего за пределы чисел, представимых в компьютере), можно применять операцию вычисления остатка от деления после каждого умножения на a. Если выполняется условие $r \neq 1$ и $r \neq p-1$, значит, число составное, и проверку можно прекращать.

Выполняется следующее равенство: $(x \cdot y) \bmod p = ((x \bmod p) \cdot (y \bmod p)) \bmod p$

Поскольку в нашем случае $a < p$, то $a \bmod p = a$. Значит, на первом шаге цикла мы имеем результат операции $a^n \bmod p$ (для $n = 1$). Тогда $a^{n+1} \bmod p = ((a^n \bmod p) \cdot (a \bmod p)) \bmod p$. $a^n \bmod p$ мы уже имеем, и $a \bmod p = a$. Остаётся только выполнить умножение и взятие остатка от деления.

```
алг ТестЛеманна()
нач
  цел p, a, t, r, i
  лог prime

  ввод p, t
  если p <= 4 или p mod 2 = 0 или t <= 0 то
    вывод "Некорректные исходные данные"
  иначе

    prime = истина
    i = 1
    пока i <= t и prime
    нц
      генерация a в диапазоне от 2 до p - 2
      r = a
      для j от 2 до (p – 1) div 2
      нц
        r = (r * a) mod p
      кц
      если r <> 1 и r <> p - 1 то
        prime = ложь
      всё
    кц

    если prime то
      вывод "Число простое"
    иначе
      вывод "Число составное"
    всё
  всё

кон
```

2. В пансионате для спортивного досуга детей оборудована специальная площадка с большим числом крупных клеток L, выложенных в дорожки одинаковой длины. По дорожкам (от начала до конца) с клетки на клетку любят прыгать отдыхающие дети. На каждой клетке нарисован вес – натуральное число. Выигрывает тот ребенок, который при прыжках набрал минимальный суммарный вес. В игре принимали участие $M<<L$ *(меньше на порядки)* детей, прыгающих за один ход на 1, 3, 4 или 5 клеток. Предложите наиболее оптимальный способ обработки и хранения информации для моделирования ситуации (например, для определения победителя). Примечание: прыгать на 2 клетки нельзя.

Решение. Для хранения данных будем использовать матрицу из $L \times L$ элементов, в элементах которой записан вес каждой клетки. Результаты детей запишем в виде массива из $L \times M$ элементов, причём элемент равен 1, если ребёнок прыгал на данную клетку, и 0 – в противном случае. Также надо записать номер дорожки, по которой он прыгал. Чтобы получить итоговый результат, надо сложить веса клеток, на которые прыгал каждый ребёнок. Для проверки соблюдения правил будем вычислять разницу между индексами элементов в каждой строке второй матрицы, содержащих значение 1. Для этого положим сначала номер предыдущей использованной клетки 0, затем при нахождении в массиве значения 1 вычисляем разницу. Если она не равна 1, 3, 4 или 5, значит, правила была нарушены. Иначе прибавляем к общему результату вес текущей клетки и запоминаем номер текущей клетки как номер предыдущей.

```
алг СпортивныйДосуг()
нач
  цел l, m, weights[l, l], tracks[m], jumps[l, m], results[m], i, j, prev
  лог right

  ввод l, m
  если l <= 0 или m <= 0 то
    вывод "Некорректные исходные данные"
  иначе

    для i от 1 до l
    нц
      для j от 1 до l
      нц
        ввод weights[i, j]
      кц
    кц

    для i от 1 до m
    нц
      ввод tracks[i]
    кц

    для i от 1 до l
    нц
      для j от 1 до m
      нц
        ввод jumps[i, j]
      кц
    кц

    для i от 1 до m
    нц
      results[i] = 0
      right = истина
      prev = 0
      j = 1
      пока j <= l и right
      нц
        если weights[tracks[i], j] = 1 то
          если j – prev = 1 или j – prev = 3 или j – prev = 4 то
            results[i] = results[i] + weights[tracks[i], j]
            prev = j
          иначе
            right = ложь
          всё
        всё
```

```
      кц

      если right то
        results[i] = -1
      всё
    кц

    для i от 1 до m
    нц
      если results[i] = -1 то
        вывод "Ребёнок нарушил правила"
      иначе
        вывод results[i]
      всё
    кц

  всё
кон
```

3. Одиннадцатиклассник Иван любит играть с калькулятором. Он часто сначала вычисляет функцию y=sin(cos(sin(cos(… sin(x)))), а затем к результату применяет обратную функцию arcsin(arccos(arcsin(… arcsin(y))). Выполнив эти действия одинаковое число раз, Иван получил в результате некоторое число. Будет ли оно исходным? Объясните, почему?

Решение. Не будет. Возникающая в процессе вычислений погрешность, обусловленная округлением вещественных чисел из-за ограниченного количества знаков в числе при представлении в ЭВМ, изменит результат.

4. Не используя дополнительный массив или простые методы сортировок, найти в матрице значения трех первых минимальных элементов.

Решение. Найдём минимальный элемент матрицы. Проверим, сколько таких элементов существует в матрице. Если их 3 и больше, значит, задача решена. Если нет, ещё раз найдём минимальный элемент из тех элементов, которые больше первого минимума. В случае необходимости найдём также третий минимальный элемент из тех элементов, которые больше первых двух минимумов.

Приведённый алгоритм может быть использован для поиска любого количества минимумов.

```
алг ТриМинимума()
нач
  цел m, n, x[m, n], min[m * n], k, km, max, i, j

  ввод m, n
  если m <= 0 или n <= 0 то
    вывод "Некорректные исходные данные"
  иначе

    для i от 1 до m
    нц
      для j от 1 до n
      нц
        ввод x[i, j]
      кц
    кц

    // Находим минимальное значение в матрице, количество таких значений и максимальное значение в матрице
    min[1] = x[1, 1]
    km = 1
    max = x[1, 1]
    для i от 1 до m
    нц
      для j от 1 до n
      нц
        если x[i, j] < min[1] то
          min[1] = x[i, j]
          km = 1
        иначе если x[i, j] = min[1] то
          km = km + 1
```

```
        иначе если x[i, j] > max то
          max = x[i, j]
        всё
        всё
        всё
      кц
    кц

    // Дублируем минимальное значение столько раз, сколько оно встречается в матрице
    для i от 2 до km
    нц
      min[i] = min[1]
    кц
    k = km

    пока k < 3
    нц
      // Ищем минимальное значение среди значений, которые больше последнего найденного минимума
      // Поскольку последний найденный минимум не меньше предыдущих, элемент матрицы, который больше
      // последнего минимума, будет больше всех найденных минимумов
      // Чтобы не искать первое значение, которое больше последнего найденного минимума, используем
      // для инициализации максимальное значение в матрице, что не повлияет на результат
      min[k + 1] = max
      km = 1
      для i от 1 до m
      нц
        для j от 1 до n
        нц
          если x[i, j] > min[k] и x[i, j] < min[k + 1] то
            min[k + 1] = x[i, j]
            km = 1
          иначе если x[i, j] = min[k + 1] то
            km = km + 1
          всё
          всё
        кц
      кц
      если min[k + 1] = max то
        km = km - 1
      всё

      // Дублируем минимальное значение
      для i от 2 до km
      нц
        min[k + i] = min[k + 1]
      кц
      k = k + km
    кц

    для i от 1 до 3
    нц
      вывод i, "-й минимум равен ", min[i]
    кц

  всё
кон
```

Однако, если подходить формально, получается, что в этом алгоритме всё-таки используется дополнительный массив – для хранения результата. Если же мы захотим избавится и от этого массива, можно обрабатывать матрицу как одномерный массив в одном цикле, высчитывать номер строки и столбца, и найденные минимумы перекладывать в начало.

Приведённый алгоритм может быть использован для поиска любого количества минимумов.

```
алг ТриМинимума()
нач
  цел m, n, x[m, n], min, k, km, i

  ввод m, n
  если m <= 0 или n <= 0 то
    вывод "Некорректные исходные данные"
  иначе
```

```
    для i от 1 до m
    нц
      для j от 1 до n
      нц
        ввод x[i, j]
      кц
    кц

    k = 0
    пока k < 3
    нц
      // Находим минимальный элемент и его номер
      min = x[k div n + 1, k mod n + 1]
      km = k
      для i от k + 1 до m * n - 1
      нц
        если x[i div n + 1, i mod n + 1] < min то
          min = x[i div n + 1, i mod n + 1]
          km = i
        всё
      кц

      // Переставляем найденный минимальный элемент на k-ое место
      x[km div n + 1, km mod n + 1] = x[k div n + 1, k mod n + 1]
      x[k div n + 1, k mod n + 1] = min

      // Ищем элементы, равные минимальному, после минимального, считаем их и переставляем
      для i от km + 1 до m * n - 1
      нц
        если x[i div n + 1, i mod n + 1] = min то
          k = k + 1
          x[i div n + 1, i mod n + 1] = x[k div n + 1, k mod n + 1]
          x[k div n + 1, k mod n + 1] = min
        всё
      кц
      k = k + 1
    кц

    для i от 1 до 3
    нц
      вывод i, "-й минимум равен ", x[i div n + 1, i mod n + 1]
    кц

  всё
кон
```

5. Простым числом Мерсенна называется простое число E, представимое в виде $E = 2^p - 1$, где p – простое число. На листе бумаги нарисована таблица размером $M \times M$ клеток. Таблица разделена на 4 равных квадрата. Какие-то из клеток в квадратах заполнены натуральными числами. Посчитать число простых чисел Мерсенна в правом нижнем квадрате. Число M должно быть чётным.

Решение. Если M – нечётное число, значит, исходные данные заданы неверно. Иначе перебираем элементы таблицы с индексами $i = M / 2 + 1 \ldots M$ и $j = M / 2 + 1 \ldots M$. Для каждого элемента, являющегося натуральным числом, надо проверить, является ли он простым числом Мерсенна. Можно просто проверять каждое число n на простоту, и, если оно является простым, вычислять $p = \ln_2(n + 1)$ и проверять, что p является целым простым числом. Или можно найти максимальное число в правом нижнем квадрате, построить массив простых чисел в диапазоне от 2 до *max*, и проверять каждое число n из правого нижнего квадрата таблицы и число $p = \ln_2(n + 1)$ на вхождение в построенный массив.

```
алг ПростыеЧислаМерсенна1()
нач
  цел m, x[m, m], n, k
  вещ p

  ввод m, n
  если m <= 0 или m div 2 <> 0 то
    вывод "Некорректные исходные данные"
  иначе
```

```
    для i от 1 до m
    нц
      для j от 1 до m
      нц
        ввод x[i, j]
      кц
    кц

    k = 0
    для i от m div 2 + 1 до m
    нц
      для j от m div 2 + 1 до m
      нц
        если x[i, j] >= 1 и Простое(x[i, j]) то
          p = ln(x[i, j] + 1)
          n = целая_часть(p)
          если n = p и Простое(n) то
            k = k + 1
          всё
        всё
      кц
    кц

    вывод k

  всё
кон

алг Простое(арг цел N)
нач
  цел i

  если N <= 1 то
    вернуть ложь
  всё
  если N = 2 или  N = 3 или  N = 5 или  N = 7 то
    вернуть истина
  всё
  если N mod 2 = 0 то
    вернуть ложь
  всё
  если N mod 3 = 0 то
    вернуть ложь
  всё

  для i от 5 до целая_часть(sqrt(N)) шаг 6               // Рассматриваем числа, меньшие корня (!) из N
  нц
    если N mod i = 0 то
      вернуть ложь
    всё
    если N mod (i + 2) = 0 то
      вернуть ложь
    всё
  кц
  вернуть истина

кон

алг ПростыеЧислаМерсенна2()
нач
  цел m, x[m, m], n, k, max, nums[100000]
  вещ p

  ввод m, n
  если m <= 0 или m div 2 <> 0 то
    вывод "Некорректные исходные данные"
  иначе

    для i от 1 до m
    нц
      для j от 1 до m
      нц
        ввод x[i, j]
      кц
    кц

    max = x[m, m]
```

```
    для i от m div 2 + 1 до m
    нц
      для j от m div 2 + 1 до m
      нц
        если x[i, j] > max то
          max = x[i, j]
        всё
      кц
    кц

    РешетоЭратосфена(max)

    k = 0
    для i от m div 2 + 1 до m
    нц
      для j от m div 2 + 1 до m
      нц
        если x[i, j] >= 1 и nums[x[i, j]] <> 0 то
          p = ln(x[i, j] + 1)
          n = целая_часть(p)
          если n = p и nums[n] <> 0 то
            k = k + 1
          всё
        всё
      кц
    кц

    вывод k

  всё
кон

алг РешетоЭратосфена(арг цел n)
  цел i

  nums[1] = 0
  для i от 2 до n
  нц
    nums[i] = i
  кц

  i = 2
  пока i <= целая_часть(sqrt(n))
  нц
    для j от 2 * i до n шаг i
    нц
      nums[j] = 0
    кц
    выполнить
      i = i + 1
    до nums[i] <> 0
  кц
кон
```

ЗАДАНИЕ ПО ИНФОРМАТИКЕ
ВАРИАНТ 73112 для 11 класса

Для заданий 1, 2, 4, 5 требуется разработать алгоритмы на языке блок-схем, псевдокоде или естественном языке

1. Для проверки, является ли большое целое простым, может использоваться вероятностный тест Леманна. Пусть $p \geq 5$ – проверяемое нечётное число. Тогда:
   - случайно выбираем a: $2 \leq a \leq p - 2$;
   - вычисляем $r = a^{(p-1)/2} \{\mathrm{mod}\ p\}$;
   - если $r \neq 1$ и $r \neq p-1$, то p – составное.

   В тесте Леманна эти проверки выполняются для t случайно выбираемых a.
   Написать алгоритм проверки вводимого числа на простоту по тесту Леманна.
   Примечание: $x = y \{\mathrm{mod}\ n\}$, если существует целое k, для которого $x = y + k \cdot n$.

Решение. В цикле t раз выбираем число a в заданном диапазоне. Для каждого a вычисляем $r = a^{(p-1)/2} \{\mathrm{mod}\ p\}$. Для того чтобы при возведении в степень не получилось слишком большого числа (выходящего за пределы чисел, представимых в компьютере), можно применять операцию вычисления остатка от деления после каждого умножения на a. Если выполняется условие $r \neq 1$ и $r \neq p-1$, значит, число составное, и проверку можно прекращать.

Выполняется следующее равенство: $(x \cdot y) \bmod p = ((x \bmod p) \cdot (y \bmod p)) \bmod p$

Поскольку в нашем случае $a < p$, то $a \bmod p = a$. Значит, на первом шаге цикла мы имеем результат операции $a^n \bmod p$ (для $n = 1$). Тогда $a^{n+1} \bmod p = ((a^n \bmod p) \cdot (a \bmod p)) \bmod p$. $a^n \bmod p$ мы уже имеем, и $a \bmod p = a$. Остаётся только выполнить умножение и взятие остатка от деления.

```
алг ТестЛеманна()
нач
  цел p, a, t, r, i
  лог prime

  ввод p, t
  если p <= 4 или p mod 2 = 0 или t <= 0 то
    вывод "Некорректные исходные данные"
  иначе

    prime = истина
    i = 1
    пока i <= t и prime
    нц
      генерация a в диапазоне от 2 до p - 2
      r = a
      для j от 2 до (p – 1) div 2
      нц
        r = (r * a) mod p
      кц
      если r <> 1 и r <> p - 1 то
        prime = ложь
      всё
    кц

    если prime то
      вывод "Число простое"
    иначе
      вывод "Число составное"
    всё
  всё

кон
```

2. В пансионате для спортивного досуга детей оборудована специальная площадка с большим числом крупных клеток L, выложенных в дорожки одинаковой длины. По дорожкам (от начала до конца) с клетки на клетку любят прыгать отдыхающие дети.

На каждой клетке нарисован вес – натуральное число. Выигрывает тот ребенок, который при прыжках набрал минимальный суммарный вес. В игре принимали участие *M<<L (меньше на порядки)* детей, прыгающих за один ход на 1, 3, 4 или 5 клеток. Предложите наиболее оптимальный способ обработки и хранения информации для моделирования ситуации (например, для определения победителя). Примечание: прыгать на 2 клетки нельзя.

Решение. Для хранения данных будем использовать матрицу из $L \times L$ элементов, в элементах которой записан вес каждой клетки. Результаты детей запишем в виде массива из $L \times M$ элементов, причём элемент равен 1, если ребёнок прыгал на данную клетку, и 0 – в противном случае. Также надо записать номер дорожки, по которой он прыгал. Чтобы получить итоговый результат, надо сложить веса клеток, на которые прыгал каждый ребёнок. Для проверки соблюдения правил будем вычислять разницу между индексами элементов в каждой строке второй матрицы, содержащих значение 1. Для этого положим сначала номер предыдущей использованной клетки 0, затем при нахождении в массиве значения 1 вычисляем разницу. Если она не равна 1, 3, 4 или 5, значит, правила была нарушены. Иначе прибавляем к общему результату вес текущей клетки и запоминаем номер текущей клетки как номер предыдущей.

```
алг СпортивныйДосуг()
нач
  цел l, m, weights[l, l], tracks[m], jumps[l, m], results[m], i, j, prev
  лог right

  ввод l, m
  если l <= 0 или m <= 0 то
    вывод "Некорректные исходные данные"
  иначе

    для i от 1 до l
    нц
      для j от 1 до l
      нц
        ввод weights[i, j]
      кц
    кц

    для i от 1 до m
    нц
      ввод tracks[i]
    кц

    для i от 1 до l
    нц
      для j от 1 до m
      нц
        ввод jumps[i, j]
      кц
    кц

    для i от 1 до m
    нц
      results[i] = 0
      right = истина
      prev = 0
      j = 1
      пока j <= l и right
      нц
        если weights[tracks[i], j] = 1 то
          если j – prev = 1 или j – prev = 3 или j – prev = 4 то
            results[i] = results[i] + weights[tracks[i], j]
            prev = j
          иначе
            right = ложь
          всё
        всё
      кц

      если right то
        results[i] = -1
```

```
      всё
    кц

    для i от 1 до m
    нц
      если results[i] = -1 то
        вывод "Ребёнок нарушил правила"
      иначе
        вывод results[i]
      всё
    кц

  всё
кон
```

3. Одиннадцатиклассник Иван любит играть с калькулятором. Он часто сначала вычисляет функцию y=sin(cos(sin(cos(… sin(x)))), а затем к результату применяет обратную функцию arcsin(arccos(arcsin(… arcsin(y))). Выполнив эти действия одинаковое число раз, Иван получил в результате некоторое число. Будет ли оно исходным? Объясните, почему?

Решение. Не будет. Возникающая в процессе вычислений погрешность, обусловленная округлением вещественных чисел из-за ограниченного количества знаков в числе при представлении в ЭВМ, изменит результат.

4. Не используя дополнительный массив или простые методы сортировок, найти в матрице значения трех первых минимальных элементов.

Решение. Найдём минимальный элемент матрицы. Проверим, сколько таких элементов существует в матрице. Если их 3 и больше, значит, задача решена. Если нет, ещё раз найдём минимальный элемент из тех элементов, которые больше первого минимума. В случае необходимости найдём также третий минимальный элемент из тех элементов, которые больше первых двух минимумов.

Приведённый алгоритм может быть использован для поиска любого количества минимумов.

```
алг ТриМинимума()
нач
  цел m, n, x[m, n], min[m * n], k, km, max, i, j

  ввод m, n
  если m <= 0 или n <= 0 то
    вывод "Некорректные исходные данные"
  иначе

    для i от 1 до m
    нц
      для j от 1 до n
      нц
        ввод x[i, j]
      кц
    кц

    // Находим минимальное значение в матрице, количество таких значений и максимальное значение в матрице
    min[1] = x[1, 1]
    km = 1
    max = x[1, 1]
    для i от 1 до m
    нц
      для j от 1 до n
      нц
        если x[i, j] < min[1] то
          min[1] = x[i, j]
          km = 1
        иначе если x[i, j] = min[1] то
          km = km + 1
        иначе если x[i, j] > max то
          max = x[i, j]
        всё
        всё
```

```
        всё
      кц
    кц

    // Дублируем минимальное значение столько раз, сколько оно встречается в матрице
    для i от 2 до km
    нц
      min[i] = min[1]
    кц
    k = km

    пока k < 3
    нц
      // Ищем минимальное значение среди значений, которые больше последнего найденного минимума
      // Поскольку последний найденный минимум не меньше предыдущих, элемент матрицы, который больше
      // последнего минимума, будет больше всех найденных минимумов
      // Чтобы не искать первое значение, которое больше последнего найденного минимума, используем
      // для инициализации максимальное значение в матрице, что не повлияет на результат
      min[k + 1] = max
      km = 1
      для i от 1 до m
      нц
        для j от 1 до n
        нц
          если x[i, j] > min[k] и x[i, j] < min[k + 1] то
            min[k + 1] = x[i, j]
            km = 1
          иначе если x[i, j] = min[k + 1] то
            km = km + 1
          всё
          всё
        кц
      кц
      если min[k + 1] = max то
        km = km - 1
      всё

      // Дублируем минимальное значение
      для i от 2 до km
      нц
        min[k + i] = min[k + 1]
      кц
      k = k + km
    кц

    для i от 1 до 3
    нц
      вывод i, "-й минимум равен ", min[i]
    кц

  всё
кон
```

Однако, если подходить формально, получается, что в этом алгоритме всё-таки используется дополнительный массив – для хранения результата. Если же мы захотим избавится и от этого массива, можно последовательно искать три минимальных элемента так, чтобы каждый следующий был больше предыдущего.

Данный алгоритм рассчитан на поиск именно трёх минимальных элементов.

```
алг ТриМинимума()
нач
  цел m, n, x[m, n], min1, min2, min3, k, km, max, i, j

  ввод m, n
  если m <= 0 или n <= 0 то
    вывод "Некорректные исходные данные"
  иначе

    для i от 1 до m
    нц
      для j от 1 до n
      нц
        ввод x[i, j]
      кц
```

```
кц

// Находим минимальное значение в матрице, количество таких значений и максимальное значение в матрице
min1 = x[1, 1]
km = 1
max = x[1, 1]
для i от 1 до m
нц
  для j от 1 до n
  нц
    если x[i, j] < min1 то
      min1 = x[i, j]
      km = 1
    иначе если x[i, j] = min1 то
      km = km + 1
    иначе если x[i, j] > max то
      max = x[i, j]
    всё
    всё
    всё
  кц
кц

если km = 2 то
  min2 = min1
всё
если km >= 3 то
  min2 = min1
  min3 = min1
всё
k = km

если k < 2
  // Ищем минимальное значение среди значений, которые больше первого минимума
  // Чтобы не искать первое значение, которое больше последнего найденного минимума, используем
  // для инициализации максимальное значение в матрице, что не повлияет на результат
  min2 = max
  km = 1
  для i от 1 до m
  нц
    для j от 1 до n
    нц
      если x[i, j] > min1 и x[i, j] < min2 то
        min2 = x[i, j]
        km = 1
      иначе если x[i, j] = min2 то
        km = km + 1
      всё
      всё
    кц
  кц
  если min2 = max то
    km = km – 1
  всё

  если km >= 2 то
    min3 = min2
  всё
  k = k + km
всё

если k < 3
  // Ищем минимальное значение среди значений, которые больше второго минимума
  // Чтобы не искать первое значение, которое больше последнего найденного минимума, используем
  // для инициализации максимальное значение в матрице, что не повлияет на результат
  min3 = max
  для i от 1 до m
  нц
    для j от 1 до n
    нц
      если x[i, j] > min2 и x[i, j] < min3 то
        min2 = x[i, j]
      всё
    кц
  кц
всё
```

```
    вывод "Первый минимум равен ", min1
    вывод "Второй минимум равен ", min2
    вывод "Третий минимум равен ", min3

  всё
кон
```

5. Число Фибоначчи – натуральное число, удовлетворяющее следующим соотношениям: $F_0 = 1,\ F_1 = 1,\ F_n = F_{n\text{-}1} + F_{n\text{-}2},\ n \geq 2$. На листе бумаги нарисована таблица размером $M \times M$ клеток. Таблица разделена на 4 равных квадрата. Какие-то из клеток в квадратах заполнены натуральными числами. Посчитать число чисел Фибоначчи в правом нижнем квадрате. Число M должно быть чётным.

Решение. Если M – нечётное число, значит, исходные данные заданы неверно. Иначе перебираем элементы таблицы с индексами $i = M / 2 + 1 \ldots M$ и $j = M / 2 + 1 \ldots M$. Для каждого элемента, являющегося натуральным числом, надо проверить, является ли он числом Фибоначчи. Для этого найдём максимальное число в правом нижнем квадрате, построим массив чисел Фибоначчи в диапазоне от 0 до *max* и будем проверять каждое число из правого нижнего квадрата таблицы на вхождение в построенный массив.

```
алг ЧислаФибоначчи()
нач
  цел m, x[m, m], k, max, f[0..100000], n, i, j, p
  лог fib

  ввод m, n
  если m <= 0 или m div 2 <> 0 то
    вывод "Некорректные исходные данные"
  иначе

    для i от 1 до m
    нц
      для j от 1 до m
      нц
        ввод x[i, j]
      кц
    кц

    max = x[m, m]
    для i от m div 2 + 1 до m
    нц
      для j от m div 2 + 1 до m
      нц
        если x[i, j] > max то
          max = x[i, j]
        всё
      кц
    кц

    f[0] = 1
    f[1] = 1
    n = 1
    пока f[n] < max
    нц
      n = n + 1
      f[n] = f[n – 1] + f[n – 2]
    кц

    k = 0
    для i от m div 2 + 1 до m
    нц
      для j от m div 2 + 1 до m
      нц
        fib = ложь
        p = 1
        пока p <= n и не fib
        нц
          если x[i, j] = f[p] то
            fib = истина
          всё
        кц
        если fib то
          k = k + 1
```

```
        всё
      кц
    кц

    вывод k

  всё
кон
```