ЗАДАНИЕ ПО ИНФОРМАТИКЕ
ВАРИАНТ 73111 для 11 класса

Для заданий 2, 3, 4, 5 требуется разработать алгоритм на языке блок-схем, псевдокоде или естественном языке

1. Утверждения $A \to C$, $A \,\&\, B \to D$, $\neg B \to E$ истинны. Чему равны A и B, если C, D и E ложны?

Решение. Таблица истинности для логической функции «импликация» представлена ниже.

X	Y	$X \to Y$
ложь	ложь	истина
ложь	истина	истина
истина	ложь	ложь
истина	истина	истина

Из таблицы видно, что если следствие ложно, то для того, чтобы вся формула была истинной, необходимо, чтобы посылка также была ложна. Таким образом, утверждение A должно быть ложно, в этом случае $A \,\&\, B$ также будет ложно. Кроме того, ложно должно быть $\neg B$, т.е. утверждение B должно быть истинно.

2. В археологических раскопках в Крыму при строительстве трассы «Таврида» археологи нашли табличку с таким текстом: $\sqrt{19} = 4 + \cfrac{1}{2 + \cfrac{1}{1 + \cfrac{1}{3 + \cfrac{1}{1 + \cfrac{1}{2 + \cfrac{1}{8 + \ldots}}}}}}$

   Далее в формуле структура из $2, 1, 3, 1, 2, 8$ повторяется бесконечное число раз. Пожалуйста, проверьте записанное предположение – разработайте алгоритм проверки на ЭВМ
   с точностью до 0.0001 справедливости этой формулы.

Решение. Данную дробь можно вычислить по формуле

$$f_1 = 1 / (2 + 1 / (1 + 1 / (3 + 1 / (1 + 1 / (2 + 1 / (8 + f_0))))))),$$

где f_0 – отбрасываемая часть, обозначенная многоточием. Положим сначала f_0 равной 0 и вычислим f_1. Если разность по модулю между переменными f_0 и f_1 окажется меньше 0.0001, можно прекращать вычисления. Иначе положим переменную f_0 равной f_1 и снова вычислим f_1 по той же формуле. Итоговый результат равен $4 + f_1$.

```
алг Дробь
нач
  вещ f0, f1

  f1 = 0
  повторять
    f0 = f1
    f1 = 1 / (2 + 1 / (1 + 1 / (3 + 1 / (1 + 1 / (2 + 1 / (8 + f0))))))
  до abs(f0 – f1) < 0.0001

  если abs((4 + f1) – sqrt(19)) < 0.0001 то
    вывод 'Формула верна'
  иначе
    вывод 'Формула не верна'
  всё
кон
```

3. По прямоугольной области вертикальной стены размера $M \times N$ плит прыжками передвигается робот. Изначально робот находится в левом нижнем углу (Н) области. Робот может передвигаться, прыгая одним из четырёх способов, показанных на рисунке. В таблице размера $M \times N$ указано, на каких плитах области находятся ступеньки. Ступенька

всегда есть на левой нижней и правой верхней плитах. Если ступенька находится на вертикальном участке траектории движения робота, то совершать прыжок нельзя (робот разбивается). Если ступенька находится на горизонтальном участке траектории движения робота, то совершать прыжок можно. Приземление возможно только на ступеньку. При этом приземление возможно на первую встреченную ступеньку, находящуюся ниже плиты, завершающей траекторию, если на ней ступеньки нет (робот падает вниз, не разбиваясь). Если же в области под плитой, завершающей траекторию, ступенек нет, то робот разбивается. Разработайте алгоритм, отвечающий на вопрос: может ли робот добраться до указанной верхней правой плиты (К). Выход за пределы области не является возможным (робот разбивается).

Траектории движения робота

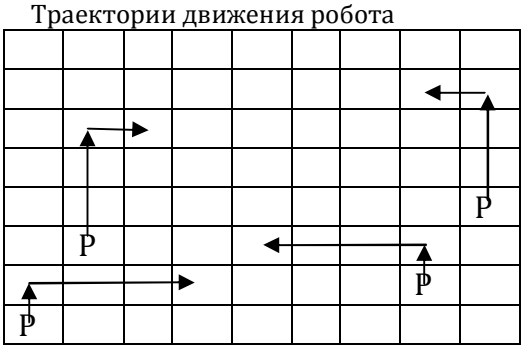
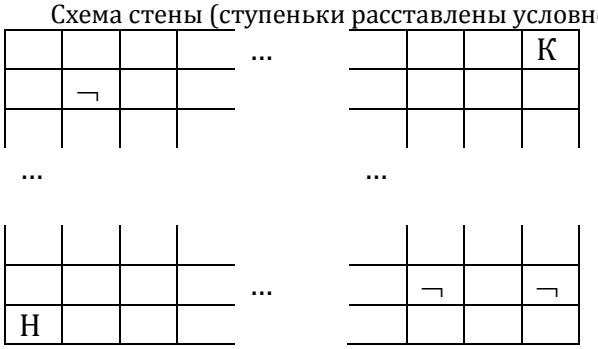


Схема стены (ступеньки расставлены условно)



Решение. Из каждой точки робот может, в принципе, совершить четыре разных прыжка. Однако, некоторые прыжки могут закончиться падением, и робот разобьётся. Поэтому на каждом шаге надо проверять, куда именно может прыгнуть робот.

Запишем в массив координаты начальной точки. Далее в цикле извлекаем (удаляем) из массива координаты первой находящейся в нём точки, взамен кладём в него координаты точек, куда робот может прыгнуть из этой точки. Таких точек может оказаться от 0 до 4. Если среди них есть конечная точка, значит, робот может туда добраться. Если же робот в принципе не может добраться до конечной точки, то в какой-то момент массив станет пустым, и можно будет прекратить поиск.

Пусть координаты начальной точки равны (1, 1), а координаты конечной – (*M*, *N*). Пусть также в таблице стоит 0, если ступеньки нет, и 1, если ступенька есть. Координаты точек, куда может попасть робот, будем хранить в массивах *x* и *y*, при этом *x*-координата соответствует номеру строки, а *y*-координата – номеру столбца. Также необходимо сохранять координаты посещённых ступенек в массивах *xp* и *yp*, чтобы робот не ходил по кругу.

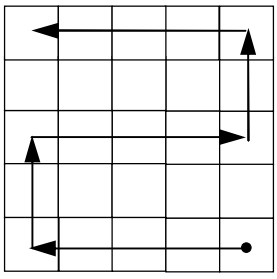
```
цел m, n
цел wall[m, n]

алг Робот
нач
  цел x[1000],  y[1000]
  цел xp[1000], yp[1000]
  цел xc, yc
  цел k, p, i, j
  лог roborCanReach

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

  k = 1
  x[1] = 1
  y[1] = 1
  p = 1
  xp[1] = 1
  yp[1] = 1
```

```
  roborCanReach = ложь
  пока k <> 0 и не roborCanReach
  нц
    xc = x[1]
    yc = y[1]
    если РоботМожетПрыгнутьПоПервойТраектории(xc, yc) и РоботНеБылВДанной Точке(xc, yc, xp, yp, p) то
      k = k + 1
      x[k] = xc
      y[k] = yc
      p = p + 1
      xp[k] = xc
      yp[k] = yc
    всё
    если x[k] = m и y[k] = n то
      roborCanReach = истина
    всё
    xc = x[1]
    yc = y[1]
    если РоботМожетПрыгнутьПоВторойТраектории(xc, yc) и РоботНеБылВДанной Точке(xc, yc, xp, yp, p) то
      k = k + 1
      x[k] = xc
      y[k] = yc
      p = p + 1
      xp[k] = xc
      yp[k] = yc
    всё
    если x[k] = m и y[k] = n то
      roborCanReach = истина
    всё
    xc = x[1]
    yc = y[1]
    если РоботМожетПрыгнутьПоТретьейТраектории(xc, yc) и РоботНеБылВДанной Точке(xc, yc, xp, yp, p) то
      k = k + 1
      x[k] = xc
      y[k] = yc
      p = p + 1
      xp[k] = xc
      yp[k] = yc
    всё
    если x[k] = m и y[k] = n то
      roborCanReach = истина
    всё
    xc = x[1]
    yc = y[1]
    если РоботМожетПрыгнутьПоЧетвёртойТраектории(xc, yc) и РоботНеБылВДанной Точке(xc, yc, xp, yp, p) то
      k = k + 1
      x[k] = xc
      y[k] = yc
      p = p + 1
      xp[k] = xc
      yp[k] = yc
    всё
    если x[k] = m и y[k] = n то
      roborCanReach = истина
    всё
    СдвинутьЭлементыМассивовВперёд(x, y, k)
  кц

  если roborCanReach то
    вывод 'Робот может достичь конечной точки'
  иначе
    вывод 'Робот не может достичь конечной точки'
  всё
кон

алг СдвинутьЭлементыМассивовВперёд(арг рез цел x[1000], алг рез цел y[1000], арг рез цел k)
нач
  цел i

  для i от 2 до k
  нц
    x[i - 1] = x[i]
    y[i - 1] = y[i]
  кц
  k = k - 1
кон
```

```
алг РоботНеБылВДанной Точке(арг цел xc, арг цел yc, арг цел xp[1000], арг цел yp[1000], арг цел p)
нач
  цел i
  лог notWas

  notWas = истина
  i = 1
  пока i <= p и notWas
  нц
    если xc = xp[i] и yc = yp[i] то
      notWas = ложь
    всё
    i = i + 1
  кц
  вернуть notWas
кон

// Вспомогательные алгоритмы не только определяют возможность прыжка,
// но и вычисляют, на какую ступеньку приземлится робот
алг РоботМожетПрыгнутьПоПервойТраектории(арг рез цел x, арг рез цел y)
нач
  цел i

  если x > m - 3 или y > n - 1 то
    вернуть ложь
  всё
  если wall[x + 1, y] = 1 или wall[x + 2, y] = 1 или wall[x + 3, y] = 1 то
    вернуть ложь
  всё
  если wall[x + 3, y + 1] = 1 то
    x = x + 3
    y = y + 1
    вернуть истина
  всё
  для i от x + 2 до 1 шаг -1
  нц
    если wall[i, y + 1] = 1 то
      x = i
      y = y + 1
      вернуть истина
    всё
  кц
  вернуть ложь
кон

алг РоботМожетПрыгнутьПоВторойТраектории(арг рез цел x, арг рез цел y)
нач
  цел i

  если x > m - 1 или y > n - 3 то
    вернуть ложь
  всё
  если wall[x + 1, y] = 1 то
    вернуть ложь
  всё
  если wall[x + 1, y + 3] = 1 то
    x = x + 1
    y = y + 3
    вернуть истина
  всё
  для i от x до 1 шаг -1
  нц
    если wall[i, y + 3] = 1 то
      x = i
      y = y + 3
      вернуть истина
    всё
  кц
  вернуть ложь
кон

алг РоботМожетПрыгнутьПоТретьейТраектории(арг рез цел x, арг рез цел y)
нач
  цел i

  если x > m - 3 или y < 2 то
    вернуть ложь
```

```
  всё
  если wall[x + 1, y] = 1 или wall[x + 2, y] = 1 или wall[x + 3, y] = 1 то
    вернуть ложь
  всё
  если wall[x + 3, y - 1] = 1 то
    x = x + 3
    y = y - 1
    вернуть истина
  всё
  для i от x + 2 до 1 шаг -1
  нц
    если wall[i, y - 1] = 1 то
      x = i
      y = y - 1
      вернуть истина
    всё
  кц
  вернуть ложь
кон

алг РоботМожетПрыгнутьПоЧетвёртойТраектории(арг рез цел x, арг рез цел y)
нач
  цел i

  если x > m – 1 или y < 4 то
    вернуть ложь
  всё
  если wall[x + 1, y] = 1 то
    вернуть ложь
  всё
  если wall[x + 1, y - 3] = 1 то
    x = x + 1
    y = y - 3
    вернуть истина
  всё
  для i от x до 1 шаг -1
  нц
    если wall[i, y - 3] = 1 то
      x = i
      y = y - 3
      вернуть истина
    всё
  кц
  вернуть ложь
кон
```

4. Хранитель леса снова решил поиграть и окутал дорогу туманом, превратив в лабиринт. На входе разместил схему лабиринта (квадратная таблица размером $N \times N$). Стрелками обозначены проходы, по которым можно идти, не опасаясь быть пойманным местными энтами. На каждом шаге при движении по лабиринту встречаются шишки с вырезанными на них натуральными числами, которые можно собирать. Путник Ткач Туманов собирал шишки и попался на шутку Хранителя леса. Помогите Ткачу Туманов пройти по лабиринту из левого нижнего угла в правый верхний и собрать шишки, на которых записаны простые числа. Число называется простым, если оно делится только на само себя и на 1.



Решение. При такой схеме лабиринта проход из левого нижнего угла в правый верхний возможен, только если количество строк N выражается формулой $4K + 1$, где K – любое число от 0. Поэтому если $N - 1$ не делится нацело на 4, то проход не возможен. Иначе K раз повторяем следующие действия: идём из по текущей строке i (i начинается с N) слева направо, затем обрабатываем ячейку с индексами $(i - 1, N)$, затем идём по строке $i - 2$ справа налево, и, наконец, обрабатываем ячейку с индексами $(i - 3, 1)$. Строка с номером 1 проходится слева

направо после завершения обработки остальных строк. При проходе проверяем числа в обрабатываемых ячеек на простоту. Если число является простым, выводим его. Можно сохранять найденные простые числа в массив, а потом вывести этот массив.

Чтобы проверить, что число *x* является простым, необходимо перебрать его возможные делители от 2 до $\sqrt{x}$ и проверить, делится ли число *x* на какой-либо из возможных делителей. Если это так, то число не является простым. При этом число 2 является простым, а число 1 не является простым.

Для сокращения перебора можно сначала проверить делимость числа *x* на 2, а потом проверить его делимость на возможные нечётные делители от 3 до $\sqrt{x}$ с шагом 2. Не забываем, что само число 2 является простым.

Можно ещё больше сократить перебор, отбрасывая числа, которые делятся на 2 и на 3. Для этого для числа *x* проверяются возможные делители *i* от 5 до $\sqrt{x}$ с шагом 6, но на каждом шаге цикла проверяется делимость числа *x* на *i* и на (*i* + 2), т.е. получается ряд 5, 7, 11, 13 и т.д. До цикла надо проверить делимость числа *x* на 2 и на 3, а также надо учесть, что сами числа 2 и 3 являются простыми.

```
алг Лабиринт
нач
  цел n
  цел maze[n, n]
  цел k, i, j, p

  ввод n
  если n <= 0 то
    вывод 'Некорректное значение'
  иначе
    если (n – 1) mod 4 <> 0 то
      вывод 'Выход из лабиринта невозможен'
    иначе
      для i от 1 до n
      нц
        для j от 1 до n
        нц
          ввод maze[i, j]
        кц
      кц

      для i от n до 2 шаг -4
      нц
        для j от 1 до n
        нц
         если Простое(maze[i, j]) то
           вывод maze[i, j]
         всё
        кц
       если Простое(maze[i - 1, n]) то
         вывод maze[i - 1, n]
       всё
        для j от n до 1 шаг -1
        нц
         если Простое(maze[i - 2, j]) то
           вывод maze[i - 2, j]
         всё
        кц
       если Простое(maze[i - 3, 1]) то
         вывод maze[i - 3, 1]
       всё
      кц
      для j от 1 до n
      нц
       если Простое(maze[1, j]) то
         вывод maze[1, j]
       всё
      кц
    всё
  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
```

5. В таблице размером $N \times 2$ записаны координаты точек на плоскости (x, y). N достаточно велико. Все точки лежат на графике некоторой функции. Таким образом, функция задана табличным способом. Разработайте алгоритм проверки того, что функция является монотонно возрастающей.

Решение. Отсортируем точки по x-координате. После этого проверим, что y-координата каждой точки, начиная со второй, не меньше y-координаты предыдущей точки.

```
алг Точки
нач
  цел n
  вещ table[n, 2]
  цел i
  лог increasing

  ввод n
  если n <= 0 то
    вывод 'Некорректное значение'
  иначе
    для i от 1 до n
    нц
      ввод table[i, 1], table[i, 2]
    кц

    QuickSort(table, 1, n)

    increasing = истина
    i = 2
    пока i <= n и increasing
    нц
      если table[i - 1, 2] > table[i, 2] то
        increasing = ложь
      всё
      i = i + 1
    кц

    если decreasing то
      вывод 'Функция является монотонно возрастающей'
    иначе
      вывод 'Функция не является монотонно возрастающей'
    всё
  всё
кон


алг QuickSort(арг рез цел x[10000, 2], арг цел n1, n2)
нач
```

```
  цел i, j,
  вещ y, k

  если n2 - n1 = 1 то
    если x[n1, 1] > x[n2, 1] то
      y = x[n1, 1]
      x[n1, 1] = x[n2, 1]
      x[n2, 1] = y
      y = x[n1, 2]
      x[n1, 2] = x[n2, 2]
      x[n2, 2] = y
    всё
  иначе
    k = x[(n1 + n2) div 2, 1]
    i = n1
    j = n2
    повторять
      пока x[i, 1] < k
      нц
        i = i + 1
      кц
      пока x[j] > k
      нц
        j = j - 1
      кц
      если i < j то
        y = x[i, 1]
        x[i, 1] = x[j, 1]
        x[j, 1] = y
        y = x[i, 2]
        x[i, 2] = x[j, 2]
        x[j, 2] = y
        i = i + 1
        j = j – 1
      иначе
        если i = j то
          i = i + 1
          j = j – 1
        всё
      всё
    до i > j
    если n1 < j то
      QuickSort(x, n1, j)
    всё
    если i < n2 то
      QuickSort(x, i, n2)
    всё
  всё
кон
```