

Задача 1. Незнайка и римско-буквенная система счисления**Критерий оценивания решений задачи 1**

Общим критерием оценивания решения любой задачи является количество тестов, верно пройденных программой, составленной участником. Сначала решение проверяется на тестах из условия. Если не пройден хотя бы один из них, то решение оценивается в 0 технических баллов. Если тесты из условия пройдены, то решение проверяется на основном наборе тестов. Высчитывается доля (в процентах) верно пройденных тестов из основного набора по отношению к общему количеству тестов в наборе. Полученное количество процентов, округлённое до целого, определяет количество технических баллов за решение.

Технические баллы за задачу 1

За решение задачи начисляется от 0 до 100 технических баллов по критерию, указанному выше.

Перевод технических баллов в оценку

По каждой задаче определяются самые удачные её решения, отправленные участником. Технические баллы за них суммируются. Полученная сумма умножается на корректировочный коэффициент $\frac{1}{6}$ и округляется до целого. Результат определяет итоговую оценку.

Условия задачи 1

Когда-то Знайка ввёл в Цветочном городе буквенную систему счисления. Это позиционная система счисления с основанием 26, в которой цифрами служат строчные латинские буквы и только они. В ней используется такая таблица цифр и их значений:

цифра	<i>a</i>	<i>b</i>	<i>c</i>	<i>d</i>	<i>e</i>	<i>f</i>	<i>g</i>	<i>h</i>	<i>i</i>	<i>j</i>	<i>k</i>	<i>l</i>	<i>m</i>	<i>n</i>	<i>o</i>	<i>p</i>	<i>q</i>	<i>r</i>	<i>s</i>	<i>t</i>	<i>u</i>	<i>v</i>	<i>w</i>	<i>x</i>	<i>y</i>	<i>z</i>
значение	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25

Приведем пример записи числа в буквенной системе счисления:

$$2026_{10} = 2 * 26^2 + 25 * 26^1 + 24 * 26^0 = czy$$

Знайка решил не останавливаться на достигнутом. Он вычитал в научном журнале, выписанном из Солнечного города, про двоично-десятичный способ записи чисел, при котором каждая цифра десятичной записи числа представляется четырёхбитным двоичным кодом её значения. Пример записи числа двоично-десятичным способом: $2026_{10} = 0010.0000.0010.0110_{bcd}$

Знайке понравилась идея смешивать разные системы счисления и получать новые способы записи чисел, и он пошел по этому пути. Знайка решил каждую цифру своей буквенной системы счисления представить как римское число, и ставить одну точку между записями соседних цифр. Так им была изобретена римско-буквенная система. В ней используется такая таблица (обратите внимание, что в качестве римских цифр используются строчные латинские буквы):

запись цифры	<i>n</i>	<i>i</i>	<i>ii</i>	<i>iii</i>	<i>iv</i>	<i>v</i>	<i>vi</i>	<i>vii</i>	<i>viii</i>	<i>ix</i>	<i>x</i>	<i>xi</i>	<i>xii</i>
значение	0	1	2	3	4	5	6	7	8	9	10	11	12
запись цифры	<i>xiii</i>	<i>xiv</i>	<i>xv</i>	<i>xvi</i>	<i>xvii</i>	<i>xviii</i>	<i>xix</i>	<i>xx</i>	<i>xxi</i>	<i>xxii</i>	<i>xxiii</i>	<i>xxiv</i>	<i>xxv</i>
значение	13	14	15	16	17	18	19	20	21	22	23	24	25

Приведем пример записи числа в римско-буквенной системе счисления:

$$2026_{10} = 2 * 26^2 + 25 * 26^1 + 24 * 26^0 = ii.xv.xxiv$$

Коротышки, привычные к изобретениям Знайки, стали пользоваться римско-буквенной системой счисления, как положено. Один лишь Незнайка, как всегда, творчески подошёл к делу. Однажды Знайка заглянул в исписанный Незнайкой листок, схватился за голову и пришёл в ужас. Он увидел, что в римско-буквенной записи чисел Незнайка нарушал правила римской системы счисления. Например, записывая в разряд значение 6, Незнайка мог написать *vi* по правилам римской системы или по своей прихоти – *iiii.x*. Знайка обратился к Незнайке за разъяснениями. Как выяснилось, Незнайке не понравилось, что в римской системе не используют больше одной *i* перед *v* или перед *x*, обозначая уменьшение, например, в *iv* и в *ix*. Незнайка разрешил себе использовать столько *i* перед *v* или перед *x*, сколько ему надо (от 0 до 5 перед *v*: *iiiiiv*, *iiiiv*, *iiiv*,

Знайка сразу понял, что модифицированная Незнайкой римско-буквенная система позволяет одно и то же число записать по-разному. Например:

Требуется написать программу, которая считает последовательность записей чисел в модифицированной Незнайкой римско-буквенной системе. В ответе программа должна выдать записи в буквенной системе счисления самого большого числа и самого малого числа в последовательности.

Формат вывода: В первой строке выводится запись в буквенной системе счисления элемента последовательности A_{max} – максимального среди чисел A_i . Во второй строке выводится запись в буквенной системе счисления элемента последовательности A_{min} – минимального среди чисел A_i . Если $A_{max} = A_{min}$, то дважды выводится запись одного и того же числа. При выводе незначащие нулевые разряды не выводятся. Например, запись нуля в буквенной системе выводится как **a**

[illegible]

В решении стоит использовать массив из 25 элементов со значениями от 0 до 25 как внутреннее представление чисел, записанных 25 разрядами рассматриваемой системы. Для удобства сравнения старшие разряды стоит хранить в начальных элементах массива. В старших разрядах могут быть незначащие нули. Программа будет в цикле считывать запись очередного числа последовательности, осуществлять перевод записи во внутреннее представление, сравнивать текущее число с текущим максимумом и текущим минимумом, обновлять текущий максимум, если считанное число больше, обновлять текущий минимум, если считанное число меньше. После окончания ввода найденные максимум и минимум будут переводиться в буквенную систему счисления и выводиться как результаты.

```

program ROMLETTERS710(input, output);
type number = array [1..25] of byte;
var K, I : word; RESMIN, RESMAX, AI : number;
function roman2Decimal(DIGIT : char): byte;
var RESULT : byte;
begin
    case DIGIT of
        'n' : begin RESULT := 0 end;
        'i' : begin RESULT := 1 end;
        'v' : begin RESULT := 5 end;
        'x' : begin RESULT := 10 end;
        else RESULT := 255;
    end;
end;

```

```

    roman2Decimal := RESULT
end;
function decimal2Letter(DIGIT : byte): char;
var RESULT : char;
begin
    case DIGIT of
        0..25 : begin RESULT := chr(ord('a') + DIGIT) end;
        else RESULT := '#';
    end;
    decimal2Letter := RESULT
end;
procedure readRoman(var N : byte);
var BUFFER : array[1..25] of byte;
    I, J : word; CH : char; D : byte;
begin
    for I := 1 to 25 do BUFFER[I] := 255;
    if not (eoln or eof) then begin
        read(CH);
        I := 1;
        D := roman2Decimal(CH);
        while (D < 26) and (I < 26) do begin
            BUFFER[I] := D;
            if not (eoln or eof) then begin
                read(CH);
                D := roman2Decimal(CH);
                I := I + 1;
            end
            else D := 255
        end;
        if (I = 1) and (BUFFER[1] > 25) then N := 255
        else begin
            I := 1;
            N := 0;
            while (I < 26) and (BUFFER[I] < 26) do begin
                J := 1;
                N := N + BUFFER[I];
                if (I > J) and (BUFFER[I] > 1) and (BUFFER[I - J] = 1) then
                    while (I > J) and (BUFFER[I - J] = 1) do begin
                        N := N - 2;
                        J := J + 1;
                    end;
                I := I + 1;
            end
        end
    end
    else N := 255
end;
procedure writeNumber(var A : number);
var I, J : word;
begin
    I := 1;
    while (I < 25) and (A[I] = 0) do I := I + 1;
    for J := I to 25 do write(decimal2Letter(A[J]))
end;
procedure readNumber(var A : number);
var CH : byte; I, J : word;

```

```

begin for I := 1 to 25 do A[I] := 0;
    readRoman(CH);
    I := 1;
    J := 0;
    while (I <= 25) and (CH < 26) do begin
        J := J + 1; A[J] := CH;
        if not (eoln or eof) then begin
            readRoman(CH);
            I := I + 1
        end else I := 26
    end;
    for I := J downto 1 do A[25 - J + I] := A[I];
    for I := 1 to 25 - J do A[I] := 0;
end;
begin readln(K);
    readNumber(RESMIN);
    RESMAX := RESMIN;
    if eoln and not(eof) then readln;
    for I := 2 to K do begin
        readNumber(AI);
        if eoln and not(eof) then readln;
        if (CompareByte(RESMAX, AI, 25) < 0) then RESMAX := AI
        else if (CompareByte(AI, RESMIN, 25) < 0) then RESMIN := AI
    end;
    writeNumber(RESMAX);
    writeln;
    writeNumber(RESMIN)
end.

```

Задача 2. Незнайка и простые бразильские числа

Критерий оценивания решений задачи 2

Общим критерием оценивания решения любой задачи является количество тестов, верно пройденных программой, составленной участником. Сначала решение проверяется на тестах из условия. Если не пройден хотя бы один из них, то решение оценивается в 0 технических баллов. Если тесты из условия пройдены, то решение проверяется на основном наборе тестов. Высчитывается доля (в процентах) верно пройденных тестов из основного набора по отношению к общему количеству тестов в наборе. Полученное количество процентов, округлённое до целого, определяет количество технических баллов за решение.

Технические баллы за задачу 2

За решение задачи начисляется от 0 до 100 технических баллов по критерию, указанному выше.

Перевод технических баллов в оценку

По каждой задаче определяются самые удачные её решения, отправленные участником. Технические баллы за них суммируются. Полученная сумма умножается на корректировочный коэффициент $\frac{1}{6}$ и округляется до целого. Результат определяет итоговую оценку.

Условия задачи 2

Незнайка бродил по интернету и наткнулся на архив задач Международной иберо-американской олимпиады. Его внимание привлекла задача о простых бразильских числах. Незнайка выяснил, что простое бразильское число – это простое число, которое в некоторой позиционной системе по основанию b , где $b > 1$, записывается тремя или более чем тремя единицами, при этом никаких других цифр кроме единиц в записи числа нет. Например, $7 = 111_2$, $13 = 111_3$, $31 = 11111_2 = 111_5$. Но оказалось, что такую запись могут иметь некоторые составные числа. Например, $21 = 111_4$, $111 = 111_{10}$. Такие числа к простым бразильским числам не относятся.

Незнайка очень увлёкся простыми бразильскими числами. Буквально про каждое встреченное им число он хотел знать, является ли оно простым бразильским числом. Помогите Незнайке и составьте для него программу.

Программа считывает десятичное натуральное ненулевое число N , а затем последовательность из N десятичных натуральных ненулевых чисел $A[i]$. Программа находит все простые бразильские числа в последовательности и выводит количество различных простых бразильских чисел в последовательности. Если простых бразильских чисел в последовательности нет, то программа выводит 0.

Формат ввода: В первой строке содержится десятичное натуральное число N : $0 < N < 1001$. Во второй строке содержится последовательность из N десятичных натуральных чисел $A[i]$: $0 < A[i] < 60001$, $i = 1, \dots, N$.

Формат вывода: Выводится десятичная запись без незначащих нулей количества различных простых бразильских чисел в последовательности $A[i]$.

Ввод примера №1:		Ввод примера №2:		Ввод примера №3:
4		1		3
2 7 21 7		111		31 13 7
Вывод примера №1:		Вывод примера №2:		Вывод примера №3:
1		0		3

Решение задачи 2

В диапазон, заданный в условии задачи, попадает лишь 70 простых бразильских чисел: 7, 13, 31, ..., 55987. Подробнее см. на странице про последовательность A085104 в OEIS (On-Line Encyclopedia of Integer Sequences): <https://oeis.org/A085104>. Для получения эффективной программы следует самостоятельно рассчитать эти числа. Массив из них будет использоваться для быстрой проверки того, является ли число простым бразильским.

В решении используем массив BP из предварительно рассчитанных простых бразильских чисел. Также используем массив C из 70 булевых значений. Массив C инициализируем ложными значениями. В цикле будем считывать очередное число $A[i]$. Бинарным поиском по массиву BP будем проверять, является ли число простым бразильским. При успешной проверке получим индекс числа в массиве BP . В элемент массива C с тем же индексом занесём истинное значение. Если

ранее там было ложное значение, то увеличим счётчик различных бразильских чисел в последовательности. Если счётчик насчитал 70 чисел, то не нужно дочитывать последовательность, так как ответ уже известен.

Код возможного решения задачи 2

```
program BRAZILPRIMES710(input, output);
const NUMBP =70;
    BP: array [1 .. NUMBP] of word = (7, 13, 31, 43, 73, 127, 157, 211,
    241, 307, 421, 463, 601, 757, 1093, 1123, 1483, 1723, 2551, 2801,
    2971, 3307, 3541, 3907, 4423, 4831, 5113, 5701, 6007, 6163, 6481,
    8011, 8191, 9901, 10303, 11131, 12211, 12433, 13807, 14281, 17293,
    19183, 19531, 20023, 20593, 21757, 22621, 22651, 23563, 24181, 26083,
    26407, 27061, 28057, 28393, 30103, 30941, 31153, 35533, 35911, 37057,
    37831, 41413, 42643, 43891, 46441, 47743, 53593, 55933, 55987);
var N, A, J : word;
    I, RESULT : byte;
    C: array [1 .. NUMBP] of boolean;
    FLAG : boolean;
function binSearch(A : word) : byte;
var L, M, H : byte;
begin
    L := 1;
    H := NUMBP;
    while L <= H do
    begin
        M := (L + H) div 2;
        if BP[M] > A then
        begin
            H := M - 1;
        end
        else if BP[M] < A then
        begin
            L := M + 1;
        end
        else
        begin
            break;
        end;
    end;
    if (M < NUMBP) AND (BP[M + 1] <= A) then binSearch := M + 1
    else if (M > 1) AND (BP[M] > A) then binSearch := M - 1
    else binSearch := M
end;

begin
    RESULT := 0;
    FLAG := false;
    for I := 1 to NUMBP do C[I] := false;
    J := 0;
    readln(N);
    repeat
        read(A);
        J := J + 1;
        I := binsearch(A);
        if A = BP[I] then
```

```
    if not C[I] then begin
        C[I] := true;
        RESULT := RESULT + 1;
        if RESULT = NUMBP then FLAG := true
    end;
until FLAG or (J >= N);
write(RESULT)
end.
```

Задача 3. Незнайка и реновация

Критерий оценивания решений задачи 3

Общим критерием оценивания решения любой задачи является количество тестов, верно пройденных программой, составленной участником. Сначала решение проверяется на тестах из условия. Если не пройден хотя бы один из них, то решение оценивается в 0 технических баллов. Если тесты из условия пройдены, то решение проверяется на основном наборе тестов. Высчитывается доля (в процентах) верно пройденных тестов из основного набора по отношению к общему количеству тестов в наборе. Полученное количество процентов, округлённое до целого, определяет количество технических баллов за решение.

Технические баллы за задачу 3

За решение задачи начисляется от 0 до 100 технических баллов по критерию, указанному выше.

Перевод технических баллов в оценку

По каждой задаче определяются самые удачные её решения, отправленные участником. Технические баллы за них суммируются. Полученная сумма умножается на корректировочный коэффициент $\frac{1}{6}$ и округляется до целого. Результат определяет итоговую оценку.

Условия задачи 3

Знайка побывал на экскурсии в Солнечном городе и ему там очень понравилось. Вернувшись домой, он решил провести реновацию Цветочного города – расширить его и полностью перестроить по примеру Солнечного города. По плану Знайки обновлённый Цветочный город стал квадратом из $N * N$ одинаковых квадратных участков. На каждом участке был выстроен один домик коротышки. Домики были перенумерованы согласно бустрофедоническому способу, названному так из-за схожести формой борозды, оставляемой при вспашке поля плугом, который тянет бык. Коротышки согласились с реновацией, предложенной Знайкой. Один лишь Незнайка решил разузнать, как на самом деле обстоят дела в Солнечном городе. Оказалось, что домики там нумеровались спиральным способом, а вовсе даже не бустрофедоническим. Домик с номером 0 (ноль) располагался в северо-западном углу (на плане это верхний левый угол). От него нумерация шла вдоль горизонтальной стороны квадрата по направлению вправо (или по ходу часовой стрелки). Достигнув правого края, нумерация продолжалась вдоль вертикальной стороны квадрата по направлению вниз. Достигнув нижнего края, она следовала вдоль горизонтальной стороны квадрата по направлению влево. Дальнейший её ход замыкал обход периметра квадрата по часовой стрелке. Потом следовал ещё один виток спирали по периметру внутреннего квадрата $(N - 2) * (N - 2)$. И так далее. Например, при $N = 6$ результат нумерации был бы таким:

0	1	2	3	4	5
19	20	21	22	23	6
18	31	32	33	24	7
17	30	35	34	25	8
16	29	28	27	26	9
15	14	13	12	11	10

Эти сведения Незнайка тут же сообщил Знайке и стал требовать соблюдения правил нумерации домиков Солнечного города. Знайка обещал рассмотреть новую реновацию – ререновацию, как он её назвал, – только в том случае, если будет надёжный способ узнать номер каждого домика по его расположению. В который раз Незнайке нужна помощь с составлением программы.

Требуется написать программу, которая считывает N – количество участков вдоль одной стороны города. Затем программа считывает I – номер вертикального ряда домиков и J – номер горизонтального ряда домиков. Рядам присвоены номера, начиная с 1, слева направо для вертикальных рядов и сверху вниз – для горизонтальных. На пересечении рядов, с номерами I и J находится домик, номер которого в спиральной нумерации следует найти. Программа находит и выводит результат – искомый номер домика. Например, если $N = 6$, как выше, и $I = 4$, а $J = 2$, то искомым номером будет 22.

0	1	2	3	4	5
19	20	21	22	23	6
18	31	32	33	24	7
17	30	35	34	25	8
16	29	28	27	26	9
15	14	13	12	11	10

Формат ввода: В первой строке вводится натуральное число N в десятичной записи: $0 < N < 1001$. Во второй строке вводится натуральное число I : $0 < I < N + 1$. В третьей строке вводится натуральное число J : $0 < J < N + 1$.

Формат вывода: Выводится десятичная запись искомого номера домика согласно спиральной нумерации. При выводе незначащие нулевые разряды не выводятся.

Ввод примера №1: | Ввод примера №2: | Ввод примера №3:

1 | 2 | 6

1 | 1 | 4

1 | 2 | 2

Вывод примера №1: | Вывод примера №2: | Вывод примера №3:

0 | 3 | 22

Решение задачи 3

Сначала определяется K – номер витка спирали, во время прохода по которому будет пронумерован домик на пересечении рядов с номерами I и J . Он равен увеличенному на 1 расстоянию от домика до ближайшего края квадрата (вертикального или горизонтального – того, что окажется ближе). В примере №3 $K = \min(I, J, (N + 1 - I), (N + 1 - J)) = 2$. Номер M , назначенный домику в начале витка, определяется как сумма первых $K - 1$ элементов последовательности: $4 * (N - 1), 4 * (N - 3), \dots, 12$ для чётных N или $4 * (N - 1), 4 * (N - 3), \dots, 8$ для нечётных N . Выведем формулу для M . При $K = 1$ $M = 0$. При больших K $M = 4 * (N - 1) - 4 * 2 * 0 + 4 * (N - 1) - 4 * 2 * 1 + \dots + 4 * (N - 1) - 4 * 2 * (K - 2) = 4 * ((N - 1) * (K - 1) - (K - 2) * (K - 1)) = 4 * (K - 1) * (N - K + 1)$. В примере №3 $M = 4 * 1 * 5 = 20$. Имеются 4 случая расположения домика:

- 1) Он находится на начальном участке витка – в горизонтальном отрезке, проходимом слева направо (в примере №3 именно этот случай, $K = J = 2$). Искомый номер находится как $M + (I - K) = 20 + 2 = 22$.
- 2) Он находится на втором участке витка – в вертикальном отрезке, проходимом сверху вниз (в таком случае $N + 1 - I = K$). Искомый номер находится как $M + (N + 1 - 2 * K) + (J - K) = M + N + J - 3 * K + 1$.
- 3) Он находится на третьем участке витка – в горизонтальном отрезке, проходимом справа налево (в таком случае $N + 1 - J = K$). Искомый номер находится как $M + 3 * (N + 1 - 2 * K) - (I - K) = M + 3 * N - I - 5 * K + 3$.
- 4) Он находится на последнем участке витка – в вертикальном отрезке, проходимом снизу вверх (в таком случае $K = I$). Искомый номер находится как $M + 4 * (N + 1 - 2 * K) - (J - K) = M + 4 * N - J - 7 * K + 4$.

Код возможного решения задачи 3

```

program SPIRAL710(input, output);
var N, I, J, K: word;
    M, RESULT: longword;
function MIN4(A1, A2, A3, A4: word): word;
begin
    if (A1 > A2) then A1 := A2;
    if (A3 > A4) then A3 := A4;
    if (A1 > A3) then MIN4 := A3 else MIN4 := A1
end;
```

```

begin
  readln(N);
  readln(I);
  read(J);
  K := MIN4(I, J, (N+1-I), (N+1-J));
  M := 4 * (K - 1) * (N - K + 1);
  if (K = J) then RESULT := M + I - K
  else if (N + 1 - I) = K then RESULT := M + N + J - 3 * K + 1
  else if (N + 1 - J) = K then RESULT := M + 3 * N - I - 5 * K + 3
  else RESULT := M + 4 * N - J - 7 * K + 4;
  write(RESULT)
end.

```

Задача 4. Очередь на Луну

Критерий оценивания решений задачи 4

Общим критерием оценивания решения любой задачи является количество тестов, верно пройденных программой, составленной участником. Сначала решение проверяется на тестах из условия. Если не пройден хотя бы один из них, то решение оценивается в 0 технических баллов. Если тесты из условия пройдены, то решение проверяется на основном наборе тестов. Высчитывается доля (в процентах) верно пройденных тестов из основного набора по отношению к общему количеству тестов в наборе. Полученное количество процентов, округлённое до целого, определяет количество технических баллов за решение.

Технические баллы за задачу 4

За решение задачи начисляется от 0 до 100 технических баллов по критерию, указанному выше.

Перевод технических баллов в оценку

По каждой задаче определяются самые удачные её решения, отправленные участником. Технические баллы за них суммируются. Полученная сумма умножается на корректировочный коэффициент $\frac{1}{6}$ и округляется до целого. Результат определяет итоговую оценку.

Условия задачи 4

Коротышки из Цветочного города готовятся отправиться в путешествие на Луну. Знайка поручил Незнайке, выстроить всех коротышек в очередь на посадку в космический корабль так, чтобы они были упорядочены по росту: первым должен стоять самый низкий, последним – самый высокий. Однако, пока Незнайка витал в облаках, все перепутались и встали в очередь, кто как хотел. Строгий Знайка решил навести порядок и наказать всех тех, кто встал в очередь не по порядку. Знайка решил выбрать самую длинную цепочку коротышек, которые уже стоят (не обязательно рядом) в правильном порядке (по неубыванию роста). Всех остальных, кто не входит в эту цепочку, он будет наказывать, заставляя учить лунные правила. Помогите Знайке узнать, каково количество коротышек, которые окажутся вне цепочки и будут наказаны.

Требуется составить программу, которая считывает натуральное число N : $0 < N < 5001$ – общее количество коротышек в очереди, а затем N натуральных чисел $H[i]$: $0 < H[i] < 1000000001$. $H[i]$ – рост коротышки, стоящего на i -ом месте в очереди. Программа находит наименьшее целое число K ($-1 < K < N$) – количество коротышек, которые не входят в цепочку из коротышек, стоящих в очереди на местах с номерами i_1, i_2, \dots, i_{N-K} , где $i_1 < i_2 < \dots < i_{N-K}$, и $H[i_1], H[i_2], \dots, H[i_{N-K}]$ является неубывающей, т. е. $H[i_j] < H[i_{j+1}]$ или $H[i_j] = H[i_{j+1}]$ для любого j ($0 < j < N - K$).

Формат ввода: В первой строке задано число N : $0 < N < 5001$. Во второй строке заданы N натуральных чисел $H[i]$: $0 < H[i] < 1000000001$.

Формат вывода: Выводится искомое число K : $-1 < K < N$.

Ввод примера №1:		Ввод примера №2:		Ввод примера №3:
6		5		5
3 29 5 5 28 6		5 8 10 4 1		1 1 2 2 2
Вывод примера №1:		Вывод примера №2:		Вывод примера №3:
2		2		0
Пояснение:		Пояснение:		Пояснение:
Цепочка: 3 5 5 6		Цепочка: 5 8 10		Все выстроились верно.
Вне цепочки: 29 28		Вне цепочки: 4 1		

Пояснения к решению задачи 4

Вместо того чтобы искать наименьшее количество наказуемых, будем искать наибольшее количество тех, кого наказывать не нужно. Тогда ответ можно получить как $ans = N - LnDS$, где $LnDS$ – это длина наибольшей невозрастающей подпоследовательности. При данных ограничениях на $N \leq 10^4$ задачу можно решить с помощью метода динамического программирования. Для этого определим «динамику» $dp[i]$ как длину наибольшей неубывающей подпоследовательности, которая заканчивается на $H[i]$ элементе. Легко видеть, что $dp[0]$ можно положить за 1. Пересчет можно выполнять в цикле по следующему соотношению $dp[i] = \max(dp[i], dp[j] + 1)$ для всех $j < i$

с условием, что $H[j] \leq H[i]$. 4. Учитывая, что наибольшая неубывающая подпоследовательность закончится на каком-то элементе, $LnDS = \max(dp[0], dp[1], \dots, dp[N - 1])$

Код возможного решения задачи 4

```
#include <bits/stdc++.h>

using namespace std;

int main(void) {
    int n;
    cin >> n;
    vector<int> arr(n);
    for (auto &i : arr) cin >> i;

    if (n == 0)
    {
        cout << 0 << endl;
        return 0;
    }

    vector<int> dp(n, 1);

    for (int i = 0; i < n; ++i) {
        for (int j = 0; j < i; ++j) {
            if (arr[j] <= arr[i] && dp[j] + 1 > dp[i]) {
                dp[i] = dp[j] + 1;
            }
        }
    }

    int len = *max_element(dp.begin(), dp.end());
    int punish = n - len;
    cout << punish << endl;

    return 0;
}
```

Задача 5. Лунная прогулка

Критерий оценивания решений задачи 5

Общим критерием оценивания решения любой задачи является количество тестов, верно пройденных программой, составленной участником. Сначала решение проверяется на тестах из условия. Если не пройден хотя бы один из них, то решение оценивается в 0 технических баллов. Если тесты из условия пройдены, то решение проверяется на основном наборе тестов. Высчитывается доля (в процентах) верно пройденных тестов из основного набора по отношению к общему количеству тестов в наборе. Полученное количество процентов, округлённое до целого, определяет количество технических баллов за решение.

Технические баллы за задачу 5

За решение задачи начисляется от 0 до 100 технических баллов по критерию, указанному выше.

Перевод технических баллов в оценку

По каждой задаче определяются самые удачные её решения, отправленные участником. Технические баллы за них суммируются. Полученная сумма умножается на корректировочный коэффициент $\frac{1}{6}$ и округляется до целого. Результат определяет итоговую оценку.

Условия задачи 5

На Луне, куда прилетел Незнайка, поверхность покрыта лунными горами. Каждая гора пронумерована начиная с 1 и имеет три характеристики: 1) x – координата долготы (восток-запад); 2) y – координата широты (север-юг); 3) h – высота горы над лунной равниной. Незнайка любит покорять вершины, но так как гор много, он придумал себе специальные ограничения, чтобы не запутаться. Он стартует с первой горы. Дальше он хочет прыгать с горы на гору, но только если выполняются такие правила:

1. Новая гора должна быть правее (восточнее) текущей: $x_{new} > x_{current}$
2. Новая гора должна быть выше на карте (севернее) текущей: $y_{new} > y_{current}$
3. Новая гора должна быть ниже той, с которой прыгает Незнайка: $h_{new} < h_{current}$

После прыжка Незнайка останавливается на новой горе и снова может прыгать дальше, если есть подходящая цель. Помогите Незнайке узнать: какое максимальное количество гор он сможет посетить за одну такую прогулку?

Формат ввода: Первая строка ввода содержит одно целое число n ($1 \leq n \leq 5000$) – количество гор. В следующих n строках содержатся по три целых числа в каждой строке: $0 \leq x_i, y_i, h_i \leq 10^9$ – координаты i горы и её высота.

Формат вывода: Выводится одно целое число – максимальное количество гор, которые Незнайка может посетить, если начнет свою прогулку с горы, характеристики которой ввели первой.

Ввод примера №1:

```
5
0 0 10
1 2 9
2 1 8
3 3 7
4 4 6
```

Вывод примера №1:

```
4
```

Ввод примера №2:

```
6
2 2 4
0 0 5
1 1 6
3 3 4
4 4 2
5 5 1
```

Вывод примера №2:

3

Ввод примера №3:

```
10
0 0 5
1 1 4
2 2 3
3 3 10
4 4 9
5 5 8
0 0 4
0 100 15
6 6 7
7 7 7
```

Вывод примера №3:

3

Пояснение к решению задачи 5

Рассматриваем горы как вершины в графе. Ребро из вершины v в вершину u есть только тогда, когда правила позволяют прыгнуть с горы v на гору u . Тогда искомая величина – это максимальная длина пути, начинающемся в вершине 1. Циклов в графе нет, так как, например при каждом прыжке координата x строго увеличивается.

Код возможного решения задачи 5

```
#include <bits/stdc++.h>
using namespace std;

struct Mountain {
    long long x, y, h;
};

int n;
vector<Mountain> a;
vector<int> dp;

int dfs(int v) {
    int &res = dp[v];
    if (res != -1) return res;

    res = 1;
    for (int to = 0; to < n; ++to) {
        if (a[to].x > a[v].x &&
            a[to].y > a[v].y &&
            a[to].h < a[v].h) {
            res = max(res, 1 + dfs(to));
        }
    }
    return res;
}

int main() {
    ios::sync_with_stdio(false);
    cin.tie(nullptr);

    cin >> n;
```

```
a.resize(n);
for (int i = 0; i < n; ++i) {
    cin >> a[i].x >> a[i].y >> a[i].h;
}

dp.assign(n, -1);

cout << dfs(0) << '\n';
return 0;
}
```

Задача 6. Морской бой

Критерий оценивания решений задачи 6

Общим критерием оценивания решения любой задачи является количество тестов, верно пройденных программой, составленной участником. Сначала решение проверяется на тестах из условия. Если не пройден хотя бы один из них, то решение оценивается в 0 технических баллов. Если тесты из условия пройдены, то решение проверяется на основном наборе тестов. Высчитывается доля (в процентах) верно пройденных тестов из основного набора по отношению к общему количеству тестов в наборе. Полученное количество процентов, округлённое до целого, определяет количество технических баллов за решение.

Технические баллы за задачу 6

За решение задачи начисляется от 0 до 100 технических баллов по критерию, указанному выше.

Перевод технических баллов в оценку

По каждой задаче определяются самые удачные её решения, отправленные участником. Технические баллы за них суммируются. Полученная сумма умножается на корректировочный коэффициент $\frac{1}{6}$ и округляется до целого. Результат определяет итоговую оценку.

Условия задачи 6

Во время одного из заседаний АО «Общество гигантских растений» Незнайка рассказал Миге про игру «Морской бой». В игре на прямоугольном поле расставляются один четырёхпалубный корабль, два трёхпалубных корабля, три двухпалубных корабля и четыре однопалубных корабля. Но, рассказывая об игре, Незнайка забыл, что корабли не могут соприкасаться, и в его версии правил корабли могут располагаться на поле как угодно. Корабли одного типа неразличимы, поэтому если поменять местами два корабля одного типа, получится та же самая расстановка.

Однажды, рассматривая тетрадку, в которой он рисовал поля для игры, Незнайка увидел, что границы между кораблями стёрлись и остался только силуэт конфигурации, то есть для каждой клетки поля известно только, был ли там корабль или нет.

Напишите программу, которая по заданному полю, на котором стёрты границы между кораблями, определит количество различных расстановок кораблей. Все клетки кораблей должны размещаться только на занятых клетках поля, и любая занятая клетка поля должна принадлежать какому-либо кораблю. Каждый корабль может размещаться на поле как горизонтально, так и вертикально.

Например, если дана конфигурация

```
.....  
.#####.  
.....
```

то существует 6 способов разместить один четырёхпалубный и два однопалубных корабля как показано ниже.

111123

111132

211113

311112

231111

321111

Формат ввода: В первой строке вводятся целые числа $N_1 : (0 \leq N_1 \leq 4)$, $N_2 : (0 \leq N_2 \leq 3)$, $N_3 : (0 \leq N_3 \leq 2)$, $N_4 : (0 \leq N_4 \leq 1)$, $R : (0 < R \leq 20)$ и $C : (0 < C \leq 20)$. N_1 – это число однопалубных кораблей, N_2 – это число двухпалубных кораблей, N_3 – это число трёхпалубных кораблей, и N_4 – это число четырёхпалубных кораблей. Затем вводятся R строк, каждая из

которых состоит из C символов, не считая конца строки. Символ # («решетка») обозначает клетку с кораблём, а символ . («точка») — пустую клетку. В задаваемой конфигурации поля могут использоваться меньше кораблей, чем в полной игре.

Формат вывода: Выводится одно целое число — количество различных конфигураций кораблей с заданным во вводе заполнением клеток поля. Корабли одного типа (например, однопалубные) различимы, поэтому перестановка местами двух однопалубных кораблей даёт новую конфигурацию.

Ввод примера №1:

2 0 0 1 3 8

.....

.#####.

.....

Вывод примера №1:

6

Код возможного решения задачи 6

```
#include <iostream>
#include <string>
#include <vector>
#include <array>

using namespace std;

int rows;
int cols;
int poscount = 0;
array<int, 4> cnt;
vector<string> table;

void solve(int kind, int curr, int curc)
{
    char ship;
    int k = 0;
    for (; k < 4; ++k) {
        if (cnt[k] > 0) break;
    }
    if (k == 4) {
        ++poscount;
        return;
    }
    ++curc;
    if (curc == cols) {
        curc = 0;
        ++curr;
    }
    if (k != kind) {
        curr = 0;
        curc = 0;
    }
    for (int r = curr; r < rows; ++r) {
        for (int c = curc; c < cols; ++c) {
            curc = 0;
            if (table[r][c] != '#') continue;
```

```

if (k == 0) {
    // 4 deck
    if (c + 3 < cols && table[r][c+1] == '#' && table[r][c+2] == '#'
        && table[r][c+3] == '#') {
        ship = '4';
        table[r][c] = ship;
        table[r][c+1] = ship;
        table[r][c+2] = ship;
        table[r][c+3] = ship;
        --cnt[k];
        solve(k, r, c);
        ++cnt[k];
        table[r][c+3] = '';
        table[r][c+2] = '';
        table[r][c+1] = '';
        table[r][c] = '';
    }
    if (r + 3 < rows && table[r+1][c] == '#' && table[r+2][c] == '#'
        && table[r+3][c] == '#') {
        ship = '4';
        table[r][c] = ship;
        table[r+1][c] = ship;
        table[r+2][c] = ship;
        table[r+3][c] = ship;
        --cnt[k];
        solve(k, r, c);
        ++cnt[k];
        table[r][c] = '';
        table[r+1][c] = '';
        table[r+2][c] = '';
        table[r+3][c] = '';
    }
} else if (k == 1) {
    // 3 deck
    if (c + 2 < cols && table[r][c+1] == '#' && table[r][c+2] == '#') {
        ship = '3';
        table[r][c] = ship;
        table[r][c+1] = ship;
        table[r][c+2] = ship;
        --cnt[k];
        solve(k, r, c);
        ++cnt[k];
        table[r][c+2] = '';
        table[r][c+1] = '';
        table[r][c] = '';
    }
    if (r + 2 < rows && table[r+1][c] == '#' && table[r+2][c] == '#') {
        ship = '3';
        table[r][c] = ship;
        table[r+1][c] = ship;
        table[r+2][c] = ship;
        --cnt[k];
        solve(k, r, c);
        ++cnt[k];
        table[r][c] = '';
    }
}

```

```

        table[r+1][c] = '#';
        table[r+2][c] = '#';
    }
} else if (k == 2) {
    // 2 deck
    if (c + 1 < cols && table[r][c+1] == '#') {
        ship = '2';
        table[r][c] = ship;
        table[r][c+1] = ship;
        --cnt[k];
        solve(k, r, c);
        ++cnt[k];
        table[r][c+1] = '#';
        table[r][c] = '#';
    }
    if (r + 1 < rows && table[r+1][c] == '#') {
        ship = '2';
        table[r][c] = ship;
        table[r+1][c] = ship;
        --cnt[k];
        solve(k, r, c);
        ++cnt[k];
        table[r][c] = '#';
        table[r+1][c] = '#';
    }
} else {
    abort();
}
}
}

int main()
{
    int n1, n2, n3, n4;

    cin >> n1 >> n2 >> n3 >> n4;
    cin >> rows >> cols;
    if (n1 < 0 || n1 > 4 || n2 < 0 || n2 > 3 || n3 < 0 || n3 > 2 || n4 < 0 || n4 > 1)
        abort();
    if (rows < 1 || rows > 20 || cols < 1 || cols > 20) abort();

    string buf;
    getline(cin, buf);

    int hashcount = 0;
    for (int i = 0; i < rows; ++i) {
        getline(cin, buf);
        if (cin.eof()) abort();
        if (int(buf.length()) != cols) abort();
        for (char c : buf) hashcount += (c == '#');
        table.push_back(buf);
    }
    getline(cin, buf);
    if (!cin.eof()) abort();
    if (hashcount != n1 + 2 * n2 + 3 * n3 + 4 * n4) abort();
}

```

```

cnt = array<int, 4>{ n4,n3,n2,0 };
solve(-1, 0, -1);

int mult = 1;
for (int i = 2; i <= n1; ++i) mult *= i;
for (int i = 2; i <= n2; ++i) mult *= i;
for (int i = 2; i <= n3; ++i) mult *= i;
for (int i = 2; i <= n4; ++i) mult *= i;

cout << (poscount * mult) << endl;
}

```