Открытая олимпиада школьников (информатика) 2024-25
Решения заданий заключительного этапа для 9 класса (типовой вариант)

1. **Кодирование информации. Системы счисления (1 балл)**
   [WOW!]

Известно, что запись числа в шестнадцатеричной системе состоит из идущих подряд последовательностей четырех цифр KXYZ, то есть выглядит как KXYZKXYZ…KXYZ$_{16}$ (K, X, Y, Z – цифры 16-ричной системы счисления). После перевода этой записи в двоичную систему счисления оказалось, что она состоит из двоичных чисел W, разделённых двумя идущими подряд нолями, то есть выглядит как W00…00W00W$_2$, причем чисел W сколько же, сколько последовательностей KXYZ в исходном числе (W также может содержать в своей записи 00). Определите количество различных вариантов последовательностей цифр KXYZ, для которых это возможно. Два варианта последовательностей будут считаться различными, если отличаются значения хотя бы одной цифры, например варианты 1234, 1134 и 1324 – это три различных варианта. В ответе укажите целое число.

Ответ: 8192

Решение:

Вспомнив, что существует упрощённый способ перевода чисел из шестнадцатеричной системы счисления в двоичную, где каждой цифре шестнадцатеричного числа ставится в соответствие 4 цифры двоичной системы, после чего у первой цифры удаляются ведущие ноли, несложно сделать вывод, что каждый набор KXYZ, кроме первого, будет соответствовать ровно 16 знакам числа W00...00W00W. У результата перевода первого набора KXYZ, в отличие от остальных, будут удалены ведущие ноли. Так как каждое число W соответствует одному набору KXYZ, можно сделать вывод, что так как перед первым числом W нет ведущих нолей и оно соответствует первому набору KXYZ, то 00, разделяющие W являются ведущими нолями результата перевода KXYZ. Первые 2 цифры будут относиться к переводу K, то есть K должно в двоичной системе иметь вид 0010 или 0011, т.е. иметь ровно 2 ведущих ноля. Для остальных цифр у нас ограничения не накладываются, то есть они могут принять любое из 16 возможных значений. Итого получим 2 * 16 * 16 * 16 = 8192.

2. **Кодирование информации. Объем информации (3 балла)**
   [Алиса и Боб]

Алиса решила написать генератор простых текстов на английском языке. Для хранения текста в памяти она решила использовать неравномерное кодирование, основываясь на частоте букв в английском алфавите. Она воспользовалась простой таблицей:

Буква	**E**	**T**	**A**	**O**	**I**	**N**	**S**	**H**	**R**	**D**	**L**	**C**	**U**
Частотность, %	12,7	9,06	8,17	7,51	6,97	6,75	6,33	6,09	5,99	4,25	4,03	2,78	2,76
Буква	**M**	**W**	**F**	**G**	**Y**	**P**	**B**	**V**	**K**	**X**	**J**	**Q**	**Z**
Частотность, %	2,41	2,36	2,23	2,02	1,97	1,93	1,49	0,98	0,77	0,15	0,15	0,1	0,05

Кодирование Алисы построено следующим образом:

1. В начале каждого кода символа находится специальный бит, обозначающий начало кода.
2. Затем идёт бит, отвечающий за то, является ли буква заглавной.
3. Кодом длиной i бит она кодирует 2^i букв с наибольшей популярностью, которым ещё не назначены более короткие коды. Например, буквы E и T получат однобитные коды, следующие 4 буквы – двухбитные коды, и так далее.

Её друг Боб справедливо заметил, что так как она генерирует простые тексты, в них используется не более 1500 различных слов (Боб также игнорирует регистр букв в слове) и предложил использовать минимальное, одинаковое для всех слов количество бит, а также для каждого слова записывать 1 бит метаинформации – начинается ли слово с заглавной буквы. Определите, при какой минимальной средней длине слова способ Боба будет эффективнее способа Алисы (для хранения всего текста потребуется строго меньше бит). Средней длиной слова будем считать количество символов, поделенное на количество слов и округлённое до ближайшего целого числа. В ответ запишите одно число – искомую среднюю длину.

Ответ: 3

Решение:

Определим, какое количество бит будет соответствовать каждой из букв. Буквы E и T получат коды длиной 1 бит.
Буквы A, O, I, N - длиной 2 бита, буквы S-M получат коды длиной 3 бита, оставшиеся буквы - длиной 4 бит.
Возьмём общее количество букв в тексте за N. Тогда каждая буква встретится $N*x_i/100$ раз, где x_i - частотность буквы i-ой буквы. Общий вес текста будет вычисляться как $N \cdot \sum_{i=0}^{25}(2 + c_i) \cdot \frac{x_i}{100}$, где c_i - длина кода i-ой буквы. Подставив все необходимые значения получим $4{,}4128 \cdot N$.
Вычислим ответ для способа Боба: в его случае ответ будет вычисляться как $\left((\log_2 1500) + 1\right) \cdot \frac{N}{A}$, где N - число букв в тексте, A - искомая средняя длина, $(\log_2 1500)$ - округленный вверх логарифм. Упросим выражение и получим $\frac{12 \cdot N}{A}$.
Составим итоговое неравенство: $4{,}4128 \cdot N > \frac{12 \cdot N}{A}$. Сократив данное выражение, получим, что $A > \frac{12}{4{,}4128}$, где правая часть выражения примерно равна 2,716, таким образом минимальное A, большее этой величины будет равно 3.

3. **Основы логики (2 балла)**
   [Под знаком вопроса]

Дано логическое уравнение с тремя логическими переменными A, B и C:

$$(A\ ?\ B)\ ?\ (B\ ?\ C)\ ?\ (A\ ?\ C) = 0.$$

В данном уравнении на месте символа ? может стоять знак операции конъюнкции или дизъюнкции. Сколько существует способов заменить ? так, чтобы уравнение имело ровно 4 решения?

Ответ: 2

Решение:

Для начала заметим, что всего возможных решений 2^5, т.е. 32 варианта. Чтобы найти ответ на данную задачу необходимо найти все варианты, при которых решений ровно 4 и доказать, что больше решений быть не может. Однако, очевидно, что хотелось бы избежать построения 32 таблиц истинности. Также выпишем все возможные варианты и будет вычёркивать их по мере рассмотрения (+ будет обозначать дизъюнкцию, * - конъюнкцию.

Для удобства рассуждений также пронумеруем знаки вопроса слева направо числами от 1 до 5.

12345	12345	12345	12345
+++++	+*+++	*++++	**+++
++++*	+*++*	*+++*	**++*
+++*+	+*+*+	*++*+	**+*+
+++**	+*+**	*++**	**+**
++*++	+**++	*+*++	***++
++*+*	+**+*	*+*+*	***+*
++**+	+***+	*+**+	****+
++***	+****	*+***	*****

Обратим внимание, что если в одной из крайних скобок (на позициях 1 или 5) будет стоять дизъюнкция и на соседней с ней позиции (т.е. 2 или 4 соответственно) будет также стоять дизъюнкция, то итоговое выражение будет иметь минимум 6 единиц вне зависимости от остальных знаков, т.е. максимум два ноля. Выражений, имеющих на позициях 1 и 2 дизъюнкции 2^3, т.е. 8. Также вычислим, что выражений, имеющих в начале две конъюнкции или конъюнкцию и дизъюнкцию (в любом порядке) и две дизъюнкции в конце будет (2 * 2 – 1) * 2 * 1 * 1 = 6 вариантов. Итого 14 вариантов, которые нам точно не подходят. Отметим красным отсечённые варианты

12345	12345	12345	12345
+++++	+*+++	*++++	**+++
++++*	+*++*	*+++*	**++*
+++*+	+*+*+	*++*+	**+*+
+++**	+*+**	*++**	**+**
++*++	+**++	*+*++	***++
++*+*	+**+*	*+*+*	***+*
++**+	+***+	*+**+	****+
++***	+****	*+***	*****

Также отсечём вариант, где дизъюнкции стоят на 2, 3 и 4 позициях (а на остальных - конъюнкции, т.к. варианты, где слева или справа две дизъюнкции подряд уже учтены) - очевидно, что в таком случае также вне зависимости от результатов в крайних скобках, результат в центральной и то, что с ним проводится только дизъюнкция, не позволит нам получить более 2 нолей.

Заметим, что если в выражении будет всего одна дизъюнкция, то она будет на позиции 1, 3 или 5 позиции, то выражение будет обращаться в ноль во всех случаях, кроме A=B=C=1. А если она будет на позиции 2 или 4, то единица будет достижима, если единице будет равна соответствующая крайняя скобка, т.е. либо дизъюнкция на позиции 2, A=B=1 и C – любое, либо дизъюнкция на позиции 4 и A=C=1, B - любое. То есть, варианты, где всего одна дизъюнкция, которых ровно 5, нам не походят. Также отметим, что вариант, где дизъюнкций нет вовсе нам тоже не подойдёт - там также ноли во всех случаях, кроме A=B=C=1.

12345	12345	12345	12345
+++++	+*+++	*++++	**+++
++++*	+*++*	*+++*	**++*
+++*+	+*+*+	*++*+	**+*+
+++**	+*+**	*++**	**+**
++*++	+**++	*+*++	***++
++*+*	+**+*	*+*+*	***+*
++**+	+***+	*+**+	****+
++***	+****	*+***	*****

На текущий момент остаётся 32-14-1-5-1=11 вариантов. Заметим, что среди них нет вариантов более чем с 3-мя дизъюнкциями, т.к. все такие варианты были отсечены как имеющие две дизъюнкции в начале или в конце.

Вариантов с ровно тремя дизъюнкциями осталось ровно 3 - на позициях 1, 3, 4, или 1, 3, 5 или 2, 3, 5. Построив таблицы истинности для этих трёх выражений выясним, что варианты 1, 3, 4 и 2, 3, 5 дают по 3 ноля, а вот вариант 1, 3, 5 даёт ровно искомые 4 ответа.

12345	12345	12345	12345

+++++	+*+++	*++++	**+++
++++*	+*++*	*+++*	**++*
+++*+	+*+*+	*++*+	**+*+
+++**	+*+**	*++**	**+**
++*++	+**++	*+*++	***++
++*+*	+**+*	*+*+*	***+*
++**+	+***+	*+**+	****+
++***	+****	*+***	*****

Во всех оставшихся вариантах по 3 конъюнкции. Заметим, что по принципу Дирихле хотя бы одна из них окажется внутри скобок. Рассмотрим вариант, где только одна из конъюнкций внутри скобок. Тогда другие две - между скобок, на позициях 2 и 4. Тогда мы получим 0 во всех случаях, когда он получается в скобках с конъюнкцией, то есть в 6 случаях из 8 вне зависимости от того, в какую скобку мы ставим конъюнкцию. То есть, мы отсекаем ещё 3 варианта. Рассмотрим вариант, где внутри скобок окажутся все 3 конъюнкции. То есть, они будут на позициях 1, 3 и 5. В этом варианте в таблице истинности будет ровно 4 ноля.

12345	12345	12345	12345
+++++	+*+++	*++++	**+++
++++*	+*++*	*+++*	**++*
+++*+	+*+*+	*++*+	**+*+
+++**	+*+**	*++**	**+**
++*++	+**++	*+*++	***++
++*+*	+**+*	*+*+*	***+*
++**+	+***+	*+**+	****+
++***	+****	*+***	*****

Остаются 4 варианта - заметим, что во всех них мы проводим дизъюнкцию результата конъюнкции двух переменных и остального выражения, которое в свою очередь состоит из конъюнкции результатов конъюнкции и дизъюнкции. То есть, все эти варианты будут иметь одинаковое число нолей в таблице истинности. Посчитав для любого из них таблицу истинности выясним, что нолей будет 5, т.е. найденный ответ не увеличится. Итого ответ 2.

4. **Алгоритмизация и программирование. Формальные исполнители (1 балл)**
   [Простые замены]

Дана последовательность нолей и единиц вида 00110011..., длиной 5000 символов. За один ход можно либо заменить 3 подряд идущих символа, начиная с текущего, на противоположные (0 на 1, 1 на 0), либо перейти к следующему символу. В случае если от некоторого символа до конца строки меньше 3 символов, замена производится для всех оставшихся до конца строки символов (например, строку 100 можно превратить в строку 111, сделав переход к следующему символу, и затем заменив 00 на 11). Для первого хода, текущим считается первый символ строки. Сколько раз понадобится совершить ход с заменой, чтобы превратить эту строку в состоящую только из единиц?

Ответ: 2502

Решение:

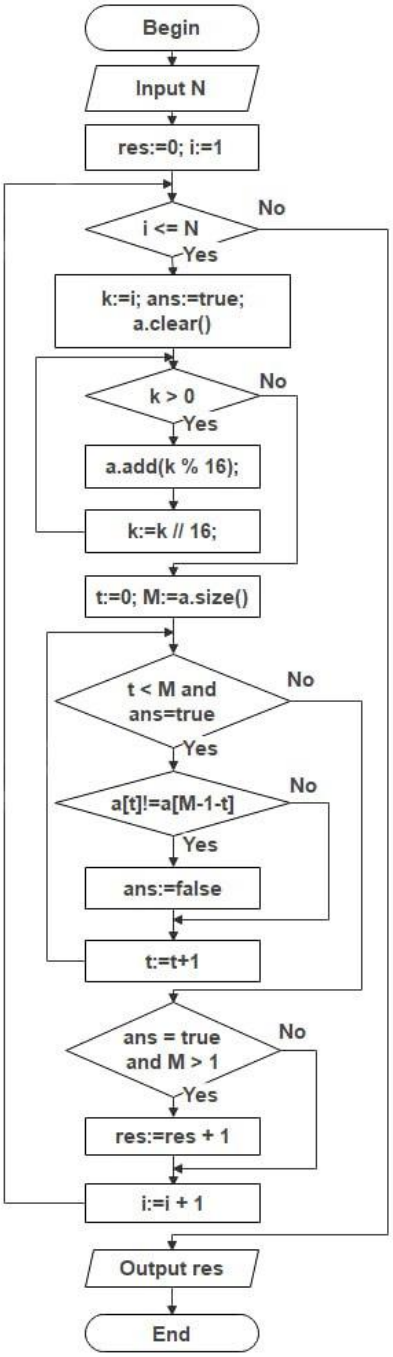
Чтобы понять ход работы алгоритма, применим его на меньшей строке - например, на строке из 20 символов. Для простоты будем выписывать результаты только искомых шагов с заменой, выделяя заменяемую подстроку жирным:

1. **001**10011001100110011
2. 11**010**011001100110011
3. 111**010**11001100110011
4. 1111**011**1001100110011
5. 11111**001**001100110011
6. 1111111**000**1100110011
7. 11111111111100110011

Заметим интересную закономерность: первые 12 символов превращаются в строку из единиц ровно за 6 шагов. Также заметим, что последующие 12 символов в точности равны первым 12 символам, а значит тоже потребуется ровно 6 шагов, и так далее. Значит, первые 5000//12 чисел буду преобразованы за (5000//12)*6=2496 шагов. Останется ещё 8 символов, то есть строка 00110011. Воспользовавшись вычислениями выше, заметим, что для 8 символов также нужно 6 шагов, т.е. всего нужно 2496+6=2502 шага.

5. **Алгоритмизация и программирование. Анализ алгоритма, заданного в виде блок-схемы (3 балла)**
   [Поиск мина и макса]

Дана блок-схема алгоритма:



Операция **%** обозначает взятие остатка от деления, операция // обозначает целочисленное деление. Операция **a.clear()** создаёт пустой список чисел, операция **a.size()** возвращает количество чисел в **a**, операция **a.add(x)** дописывает **x** в конец списка **a**. На вход было подано некоторое натуральное число **N**, после чего алгоритм вывел 278. Найдите минимальное и максимальное **N**, при которых это возможно. В ответ запишите через пробел два числа – минимальное и максимальное значения.

Ответ: 9826 10097

Решение:

Для решения задачи реализуем описанный алгоритм. Нетрудно понять, что он вычисляет количество чисел, меньших либо равных N, которые при переводе в шестнадцатеричную систему счисления будут иметь вид палиндрома длиной 2 или больше. Очевидно, что минимальное число, при котором можно получить ответ 278 — это 278-й палиндром длины 2 или более, а максимальное — это предыдущее число перед 279-м палиндромом. Нетрудно вычислить, что двузначных шестнадцатеричных палиндромов - 15, трёхзначных - 15 * 16 = 240, значит искомый палиндром - 23-й четырёхзначный. Четырёхзначных палиндромов, начинающихся с 1 – 16, значит, нам нужен 7-й, начинающийся с 2, т.е. 2662. $2662_{16}=9826_{10}$. Следующий палиндром — это 2772, т.е. искомое максимальное $N=2771_{16}=10097_{10}$. Итого ответ - 9826 10097.

6. Телекоммуникационные технологии (2 балла) [Последствия невнимательности]

Маска сети для IPv4 адресации – это 4-байтное число, которое делит IP адрес на адрес сети (первая часть) и адрес узла (вторая часть). У всех адресов одной IP-сети совпадают первые части и отличаются вторые. Для части IP адреса, соответствующей адресу сети, в маске сети содержатся двоичные единицы, а для части IP адреса, соответствующей адресу узла, в маске сети содержатся двоичные нули.

Недавно Олег узнал о масках подсети, однако, пока ему рассказывали, он был не слишком внимателен, и запомнил, что чтобы получить адрес сети, нужно применить побитовую операцию «исключающее или» к адресу узла и маске. В результате из адреса узла X и трёх различных корректных масок A, B и C Олег получил следующие “адреса сетей” – 40.102.219.169, 40.102.196.89 и 40.102.59.169. Определите адрес узла, который использовал Олег. В ответ запишите 4 числа без ведущих нолей, разделённых точками – адрес узла.

Пример записи ответа: 192.168.0.1

Ответ: 215.153.59.169

Решение:

Для решения данной задачи переведём все 3 “адреса сети” в двоичный формат:

	Первый байт	Второй байт	Третий байт	Четвёртый байт
Адрес от маски A	00101000	01100110	11011011	10101001
Адрес от маски B	00101000	01100110	11000100	01011001
Адрес от маски C	00101000	01100110	00111011	10101001
Восстановленный адрес				

Обратим внимание, что у всех адресов совпадают начала — значит, для всех совпадающих бит начала был применён XOR с одним и тем же значением, так как это начало маски - с единицей. Применив к этим битам XOR с 1, повторно получим начальный фрагмент исходного адреса

	Первый байт	Второй байт	Третий байт	Четвёртый байт
Адрес от маски A	00101000	01100110	11011011	10101001
Адрес от маски B	00101000	01100110	11000100	01011001
Адрес от маски C	00101000	01100110	00111011	10101001
Восстановленный адрес	11010111	10011001		

Повторим аналогичную операцию с совпадающими концами, но с одним отличием так как это конец маски, то нужно применять XOR с 0.

	Первый байт	Второй байт	Третий байт	Четвёртый байт
Адрес от маски A	00101000	01100110	11011011	10101001
Адрес от маски B	00101000	01100110	11000100	01011001
Адрес от маски C	00101000	01100110	00111011	10101001
Восстановленный адрес	11010111	10011001		0110

Обратим внимание на третий байт: так как первые 3 бита совпадают у масок A и B, и при этом все 3 маски различны и все маски состоят сначала из последовательности единиц, а затем из последовательности нолей, очевидно, что нули не могли начаться одновременно у масок A и B (это противоречит условию о неравенстве масок), следовательно нули начались у маски C. Используя это знание, можно из 3-го адреса полностью восстановить искомый адрес: 11010111.10011001.00111011.10101001, т.е. 215.153.59.169.

7. Технологии обработки информации в электронных таблицах, технологии сортировки и фильтрации данных (2 балла) [Есть тут кто?]

В ячейки таблицы A1, A4, B1, B2, C1, D1 поместили некоторые формулы (см. изображение).

	A	B	C	D	E
1	=COUNTIF(B2:AD30; "=0")	=A2	=A3	=IF(AND((B$1>C$1); (C$1>1)); C$1-1; IF(C$1<10; C$1+1; C$1-1))	
2	[redacted]	=MOD(MOD((B1+A2); 100); 11)			
3					
4	=IF(AND(($A2>$A3); ($A3>1)); $A3-1; IF($A3<10; $A3+1; $A3-1))				
5					

Формулу из ячейки B2 скопировали во все ячейки диапазона B2:AD30. Формулу из ячейки A4 скопировали во все ячейки диапазона A4:A30. Формулу из ячейки D1 скопировали во все ячейки диапазона D1:AD1.

В каждую из ячеек A2 и A3 либо поместили число от 1 до 10, либо оставили ячейку пустой. В результате в ячейке A1 отобразилось значение 206, число в ячейке B2 оказалось нечётным, число в ячейке B3 совпало с числом в ячейке B2. Определите для каждой из ячеек A2 и A3, поместили ли туда число (и тогда какое) или оставили её пустой. Если ячейка

осталась пустой, напишите -, в противном случае укажите число. В ответ запишите через пробел два значения – значение в ячейке A2 и значение в ячейке A3.

Пример записи ответа: 12 -

Примечание: таблица соответствия имён используемых функций.

	Google Sheets	**Excel**	**LibreOffice**
Условное вычисление	IF	ЕСЛИ	IF
Логическое И	AND	И	AND
Остаток от деления	MOD	ОСТАТ	MOD
Число ячеек, для которых верно условие	COUNTIF	СЧЁТЕСЛИ	COUNTIFS

Ответ: 8 -

Решение:

В первую очередь воссоздадим указанную таблицу. Очевидно, что число в ячейке B2 - это сумма чисел в ячейках A2 и B1, взятая по модулю 100, а затем по модулю 11. Сразу заметим, то так как мы суммируем числа, не превосходящие 10 (11, если в суммировании участвуют ячейки из диапазона B2:AD30), взятие по модулю 100 не имеет никакого смысла. Так как число в ячейке B1 равно числу в ячейке A2, то изначальная сумма будет равна удвоенному A2. Следовательно, удвоенное A2 меньше 11 - оно и останется чётным, иначе чётность изменится. Следовательно, из того, что в B2 нечётное число, следует, что в A2 число от 6 до 10 включительно. Число в ячейке B3 является суммой чисел в ячейках B2 и A3, взятой по модулю 100, а затем по модулю 11. Так как число в B3 равно числу в B2, то число в B3, то число в ячейке A3 должно быть равно 0 по модулю 11. Ни одно из чисел от 1 до 10 не подходит под это определение, значит, в ячейке A3 числа нет, т.е. ответ для этой ячейки -. Установив значение (точнее, его отсутствие) в A3, значение в A2 можно восстановить простым перебором возможных значений. Только при установлении туда значения 8, число в A1 совпадёт с искомым 206. Итого ответ - 8 -.

Задание 8 (3 балла) и Задание 9 (4 балла). Технологии программирования

В отдельном архиве опубликованы тесты, решения жюри и проверяющие программы для задач по технологиям программирования. Архив содержит 4 директории: по два варианта каждой из двух задач. В поддиректории tests каждой задачи содержатся тесты и ответы решения жюри в файлах "??" и "??.a" соответственно, где "??" -- двузначное число с, возможно, ведущим нулем. В поддиректории solutions каждой задачи содержатся решения жюри на различных языках программирования, а также примеры неверных решений (см. файлы с расширением *.desc, чтобы классифицировать решения). В корневом директории архива находится программа, использовавшаяся для проверки ответа участника (check.cpp). В поддиректории statements содержатся условия задач. Апелляции заданий по программированию должны быть основаны на предоставленных тестах.

Наименования директорий с задачами (по вариантам): задача 8 – robot-pixel-art-v1, robot-pixel-art-v2; задача 9 – robot-connection-v1, robot-connection-v2.

Покраска холста

Эта задача на реализацию не требовала никаких алгоритмических идей. В ней было необходимо просто аккуратно реализовать все требования из условия, при этом ограничения по времени позволяли выполнять каждый запрос за время $\mathcal{O}(nm)$ с итоговой асимптотикой времени решения $\mathcal{O}(nmq)$.

Для начала поймем, какие данные нам надо хранить.

1. Во-первых, надо хранить настройки. Для этого достаточно было завести матрицу `settings` размером $n \times m$, каждый элемент которой — либо какое-то особое значение, соответствующее $\varnothing$, либо реальная пара из направления и цвета (или новой клетки и цвета во втором варианте).

2. Во-вторых, ограничения. Ограничения первого типа можно хранить в словаре (unordered map, hash map) `limits` из цветов в последний введенный лимит. Ограничения второго типа достаточно хранить просто в множестве из четверток цветов `block`.

3. Чтобы быстро понимать, не нарушаем ли мы ограничение на число клеток какого-либо цвета, будем поддерживать число клеток каждого цвета на поле в словаре `count` из цветов в количество.

4. Ну и, разумеется, надо хранить само поле `field`.

И на самом деле, этого уже достаточно для полного решения. Опишем ниже процесс ответов на запросы (команды).

1. Запрос покраски. Перед обработкой заведем изначально пустое множество уже посещенных клеток. Затем в цикле `while` будем совершать покраску и перемещения. Покраску проще всего расписать следующим псевдокодом:

```
func color(i, j, c):
    if count[c] + 1 > limit[c]:
        STOP
    if block.contains(any of {
        [field[i-1][j-1], field[i-1][j], field[i][j-1], c],
        [field[i-1][j], field[i-1][j+1], c, field[i][j+1]],
        [field[i][j-1], c, field[i+1][j-1], field[i+1][j]],
        [c, field[i][j+1], field[i+1][j], field[i+1][j+1]],
    }):
        STOP

    count[field[i][j]]--
    field[i][j] = c
    count[c]++
```

   То есть проверим, не будет ли нарушено никакое ограничение, после чего выполним покраску и обновим количество клеток двух затронутых цветов на поле. Затем уже выполним перемещение и проверку, что мы не вышли за границы поля, и не пришли в клетку, которая уже есть в нашем множестве посещенных.

2. Запросы добавления или удаления ограничения на квадраты транслируются просто в добавление или удаление четверки цветов из `block`. Единственное отличие — перед удалением ограничения ничего проверять не надо, а перед добавлением надо отдельно пройти по всему полю и для каждого квадрата проверить, не совпадает ли он с добавляемым ограничением. Это просто два вложенных цикла по строкам и столбцам поля.

3. Аналогично, запрос изменения лимита по цвету состоит из проверки текущего `count` этого цвета и выставления нового значения в `limit`.

4. Ну и запросы изменения настроек просто меняют матрицу настроек без каких-либо дополнительных проверок. Направления (первый вариант) можно было хранить просто как пару из смещения по строкам и смещения по столбцам, что является стандартной практикой.

Таким образом, мы получаем полное решение, работающее за время $\mathcal{O}(nmq)$. Во втором варианте некоторые детали были реализованы по-другому, но полностью аналогично описанной выше логике.

Кластеры роботов

В этой задаче были необходимы базовые алгоритмы работы с графами, а также идеи жадных алгоритмов: понимание того, какие действия приводят к строго более положительной ситуации, и почему такими действиями можно прийти к глобальному оптимуму.

Первым действием выделим в графе компоненты связности. Каждая компонента связности — это множество вершин, попарно связанных друг с другом путями, то есть последовательностями ребер, имеющих общие концы. Выделим три типа компонент связности:

1. изолированная вершина (вершина, из которой вообще не выходят ребра), которая образует компоненту связности размера 1;

2. дерево размера больше 2; это компонента связности, в которой число ребер на 1 меньше числа вершин, поэтому между любыми двумя вершинами есть единственный путь, и удаление любого ребра ведет к тому, что компонента распадается на два меньших дерева;

3. все оставшиеся — компоненты связности, в которых есть «лишние» ребра, от удаления которых свойство связности всех ее вершин не нарушается.

Две ключевые идеи задачи:

- можно использовать «лишнее» ребро для подключения любой компоненты связности к текущей;

- можно использовать ребро в дереве между листом (вершиной с одним исходящим ребром) и его соседом, чтобы подключить любую другую компоненту, потеряв в процессе этот лист.

Заметим, что при соединении двух компонент их множества лишних ребер объединяются, а в конечном итоге либо лишние ребра останутся, и тогда вообще все роботы подключены к первому (напрямую или косвенно), либо лишних ребер не останется, и тогда компонента связности первого робота будет деревом. Это означает, что в первом случае можно спокойно расходовать лишние ребра, потому что мы все равно сможем подключить всех роботов, а во втором — их можно спокойно расходовать, потому что в итоге они все равно будут израсходованы целиком.

Остается только заметить, что в случае, когда сейчас нет доступных «лишних» ребер, мы можем либо подключить за счет листа компоненту, в которой они есть, и затем подключить все оставшиеся, в которых они есть, либо все оставшиеся компоненты связности — деревья, и тогда остается только обменять листья на оставшиеся деревья размера хотя бы 2, после чего завершить алгоритм — остальные вершины все равно подключить не получилось бы.

Таким образом, полный алгоритм выглядит так.

1. Выделим компоненты связности, это можно сделать с помощью алгоритмов обхода `dfs` или `bfs`. При обходе они также строят то, что называется остовным деревом компоненты — множество ребер, образующее дерево. Выделим отдельно дерево и все «лишние» ребра. А для дерева запомним порядок обхода и «родителя» каждой вершины.

2. Оставим отдельно компоненту связности первой вершины. Если она размера 1, то мы ничего не можем сделать, и ответ — «`1 0`». Иначе отсортируем все остальные компоненты по следующему критерию: сначала по убыванию числа «лишних» ребер, затем по убыванию размера. Тогда изолированные вершины будут в самом конце.

3. По очереди будем подключать эти компоненты в полученном порядке, пока есть возможность. Если у нас есть «лишнее ребро», воспользуемся им. Иначе, если размер компоненты, которую мы хотим подключить, больше 1, потратим лист. Для этого возьмем последнюю в порядке обхода вершину (она точно будет листом в дереве обхода) и потратим ребро из нее в предка. Образуется новая компонента размера 1, ее можно положить в конец списка компонент, но зато мы можем подключить следующую компоненту из списка.

При подключении компоненты объединим ее множество «лишних» ребер с нашим, а также припишем ее порядок обхода в конец к нашему. Действительно, если мы присоединяем новую компоненту, то дальше перед тем, как расходовать листья нашей исходной компоненты, мы можем целиком израсходовать то, что только что подключили, если в этом будет необходимость.

Процесс остановится, когда либо все компоненты будут подключены, либо не останется «лишних» ребер и неподключенных компонент размера больше 1. Весь алгоритм занимает $\mathcal{O}(n \log n + m)$ времени из-за обхода графа и сортировки.