Краткий разбор задач первого тура

Задача A. Игра в дурака

Формализуем условие задачи: имеется 4 числа, принимающие значения из $\{0, 6, 7, \ldots, 14\}$. Спрашивается, сколько среди них совпадающих чисел, отличных от нуля. Остаётся аккуратно реализовать написанное.

Задача B. Контрольная работа

Автор задачи — В. Ильин

Существует два подхода к решению: математический и программистский.

Математический. Обозначим через (x, y) НОД чисел x и y. По условию задачи, числа aN, bN, cN, dN должны делиться на 100. Если все четыре a, b, c, d делятся не некоторое число $q \neq 1$, то, во-первых, на q делится $100 = a + b + c + d$, во-вторых, $N = 100/q$ удовлетворяет условиям делимости. Если же такого q нет, то ответ на вопрос задачи не может быть меньше 100. Итак, искомое $N = 100/\text{НОД}(a, b, c, d) = 100/((a, b), (c, d))$.

Программистский. Очевидно, что $N = 100$ подходит. Необходимо лишь проверить, подходят ли $N < 100$. Будем перебирать N с 1 до 100 и проверять каждое значение N на условия делимости. Первое N, которое подошло, и есть ответ.

Задача C. Транспортные вопросы

К решению этой задачи также существует два подхода.

Программистский. Переберём количество такси, которое нужно заказать с 0 до $(n + 3)/4$ (а не до $n/4$ — это ошибка). Для каждой итерации просто вычислить количество автобусов, которое нужно заказать на оставшихся участников, и посчитать суммарные затраты. Среди всех найденных способов несложно выбрать минимум и вывести его.

Математический. В этом подходе можно было совершить ощутимо больше ошибок, чем в предыдущем методе. Он основа на следующей идее: посчитаем стоимость провоза одного участника на автобусе ($B/50$) и на такси ($T/4$). Наиболее оптимально было бы возить всех участников тем транспортом, для которого эта стоимость ниже. Однако это не работает тогда, когда количество участников не делится нацело на количество посадочных мест в оптимальном средстве передвижения. В качестве подтверждения приведём 14-й тест: `154 51 4`. Ответ на него `1 26`. Аккуратно анализируя граничные случаи, мы приходим, например, к такому решению (оно приведено без заголовков):

```
int Round(int x, int y) {
    if (x < 0) return 0;
    return x / y + (x % y != 0);
}
int main()
{
    int n, B, T;
    cin >> n >> B >> T;
    int bestTaxi = 0, bestBus = 0;
    if (B*4 > T*50) { /// один автобус дороже 12.5 такси
        bestTaxi = Round(n, 4);
        if (B + Round(n - 50, 4)*T < bestTaxi*T) /// выгоднее-таки взять один автобус
            bestBus = 1, bestTaxi = Round(n - 50, 4);
```

```
    } else {
        bestBus = n / 50;
        bestTaxi = Round(n % 50, 4);
        if (B < T*bestTaxi) /// один неполный автобус дешевле, чем несколько такси
            ++bestBus, bestTaxi = 0;
    }
    cout << bestBus << " " << bestTaxi << endl;
    return 0;
}
```

Задача D. Числовые промежутки

В этой задаче нужно было аккуратно написать так, как сказано в условии. Вот пример реализации на C++ от Серёжи Копелиовича (без заголовков):

```
struct rational
{
  int a, b;

  void read()
  {
    char s[100];
    scanf(" %[-0-9/]s", s);
    b = 1;
    sscanf(s, "%d%*c%d", &a, &b);
  }

  int down() { return (int)floor((double)a / b); }
  int up() { a += b - 1; return down(); }
};

int main()
{
  rational A, B;
  char open, close;
  scanf("%c", &open);
  A.read();
  scanf("%*c");
  B.read();
  scanf("%c", &close);

  if (open == '(') A.a++;
  if (close == ')') B.a--;

  printf("%d\n", B.down() - A.up() + 1);
  return 0;
}
```

Задача E. Делёж яблок

Эта задача на аккуратную реализацию классической идеи “поиск с конца”. Предположим,

что после $(k-1)$-го внука осталось A_k яблок. Тогда $\left[\frac{A_k}{2}\right] - a_k = 0$, при этом неизвестно, в какую сторону происходит округление. Тем не менее, мы можем точно утверждать, что $2a_k \leqslant A_k \leqslant 2a_k + 1$. В общем случае, если после $(i-1)$-го внука осталось A_i яблок, то $\left[\frac{A_i}{2}\right] - a_i = A_{i+1}$ и $2(A_{i+1} + a_i) \leqslant A_i \leqslant 2(A_{i+1} + a_i) + 1$. Применяя оценки для $A_{i+1}, \ldots, A_k$, получим:

$$2(a_i + 2(a_{i+1} + 2(\cdots + 2a_k))\ldots) \leqslant A_i \leqslant 2(a_i + 2(a_{i+1} + 2(\cdots + 2a_k + 1) + 1) + 1\ldots) + 1 \qquad (1)$$

Обозначим левую оценку через L_i. Вынося из правой оценки единицы за скобки и используя равенство $1 + 2 + \cdots + 2^{k-i} = 2^{k-i+1} - 1$, получаем $L_i \leqslant A_i \leqslant L_i + 2^{k-i+1} - 1$.

Искомые числа — это L_1 и $L_1 + 2^k - 1$. Ответы на тесты третьей группы будут переполнять все стандартные типы, поэтому на полный балл необходимо писать "длинную" арифметику. Для тех, кто ни разу не слышал этого понятия, стоит ознакомиться с ним, например, на http://informatics.mccme.ru/moodle/course/view.php?id=17.

Задача G. Студентам — бесплатно!

Автор задачи — Е. Андреева

Формализуем условие задачи: дана таблица $S \times S$, некоторые клетки которой помечены. Нужно раскрасить эти клетки в два цвета (допустим, чёрный и белый), чтобы в каждом столбце и каждой строке количество чёрных и белых клеток различалось не более чем на 1.

Построим двудольный граф: вершинами первой доли пусть будут строки таблицы, второй доли — столбцы, а рёбра графа будут соответствовать помеченным клеткам таблицы (две вершины соединим ребром, если на пересечении соответствующих им строки и столбца находится помеченная клетка). В этих условиях раскраска клеток будет означать раскраску рёбер. В каждой доле создадим по одной новой фиктивной вершине, из которых проведём по одному ребру в каждую вершину нечётной степени из противоположной доли (эти рёбра также назовём *фиктивными*). Степени обеих фиктивных вершин имеют одинаковую чётность. Если они нечётные, соединим фиктивные вершины ещё одним ребром. Таким образом, в полученном графе степени всех вершин будут чётными.

Если бы наш граф был связным, можно было бы утверждать, что в нём существует эйлеров цикл. Однако, вообще говоря, он может быть несвязным, поэтому можно утверждать лишь то, что этот граф можно разбить на рёберно непересекающиеся простые циклы (это делается аналогично поиску эйлерова цикла). В силу двудольности графа все эти циклы будут иметь чётную длину. Раскрасим каждое второе ребро каждого найденного цикла в чёрный цвет, все остальные — в белый, после чего удалим фиктивные вершины и ребра. Очевидно, что для любой вершины количество чёрных и белых рёбер, смежных с ней, различается не более чем на 1, что и требовалось.

Таким образом, искомая раскраска всегда существует.

Задача H. Магия числа 23

Автор задачи — Ф. Ивлев

Заметим, что число $\overline{KK} = K \cdot (10^L + 1)$, где L (длина числа K) — некоторое число в промежутке от N до $N+23$. Обозначим $10^L + 1$ через M. Чтобы $K \cdot M$ было квадратом, достаточно двух условий:

- Длина десятичной записи числа $\overline{K}$ равна $L - 1$.
- Степень вхождения любого простого числа в K той же четности, что и в M.

Допустим мы нашли простой делитель p, который входит в M не менее чем во второй степени. Тогда $\frac{M}{p^2} \cdot M$ является полным квадратом. Заметим, что $\frac{M}{p^2}$ может оказаться короче M более чем на один знак. Тогда мы домножим его несколько раз на 4. При умножении на M это число все равно будет давать полный квадрат, а длина увеличится до $L - 1$.

Осталось заметить, что в качестве этого простого делителя нам всегда подойдет число 11. Если L нечетно, то M делится на 11, и при этом получается число вида $9090\ldots9091$. Оно же в свою очередь делится на 11 раз в 11 раз (см. следующий абзац). Итак, раз в двадцать два значения L число M делится на 121. Соответственно, можно в заданном интервале всегда найти число вида 10^L+1, которое будет делиться на 121. Решение работает за время деления на 121 и умножения на 4. После этого можно было сделать ещё один шаг и понять, что ответ всегда представляет из себя число, выглядящее как предпериод с последующим периодом, которые можно предпосчитать.

Обоснуем делимость на 121 более строго. Для всех нечетных L

$$M = \underbrace{909090\ldots9091}_{L-2}\cdot 11$$

По признаку делимости на 11, число $9090\ldots9091$ будет опять делиться на 11, если $\frac{9(L-2)-2}{2}$ делится на 11. Итак, нами найдено число $A = \frac{M}{121}$, которое при умножении на M дает полный квадрат. Но его длина равна $L-3$ в отличие от нужного $L-1$. Умножив нужное количество раз (двух достаточно) A на 4, мы получим число, удовлетворяющее обоим необходимым условиям. Оно и будет являться ответом.

Задача I. Как стать призёром

Автор задачи — Г. Евстропов

Сперва сделаем несколько очевидных замечаний:

1. Если увеличить число подходов, которые может сделать преподаватель, не изменяя при этом множества школьников, то ответ не ухудшится.

2. Если к школьнику сделать ровно k подходов, но этого будет недостаточно для того чтобы он стал призером, то нет смысла их делать.

3. Если к школьнику сделать k подходов, и этого будет достаточно для того чтобы он стал призером, нет смысла подходить к нему еще.

4. Если к школьнику сделать k подходов, но для того, чтобы он стал призером, достаточно $(k-1)$ подхода, нужно сделать $(k-1)$ (следует из первого пункта).

5. Из предыдущих пунктов следует, что к любому школьнику имеет смысл либо делать некоторое строго определенное число подходов (для каждого школьника свое), либо не подходить вовсе.

Вопрос вычисления этого строго определенного числа рассмотрим чуть позже. Теперь задача выглядит следующим образом: для каждого школьника известно, сколько нужно сделать подходов, чтобы он стал призером, и известно, сколько всего подходов можно сделать. Это ни что иное, как простейший вариант задачи о рюкзаке: есть N предметов, у каждого свой вес, фиксирован максимальный суммарный вес, требуется взять как можно больше предметов. Так как все предметы одной ценности, интуитивно понятно, что сначала нужно брать предметы с меньшим весом. Итак, получили следующее решение:

1. Вычислим, сколько раз требуется подойти к каждому школьнику, чтобы он взял диплом олимпиады.

2. Отсортируем школьников в порядке неубывания этой величины.

3. Будем подготавливать их по очереди, пока не закончатся школьники или свободные подходы.

Теперь рассмотрим проблемы, которые могли возникнуть на каждом из шагов.

1. Достаточно очевидная формула $(K - A_i)/B_i$, округленное вверх, таит в себе несколько подвохов. Во-первых, число $K - A_i$ может оказаться отрицательным. Это означает, что школьник уже способен взять диплом, и его не требуется тренировать. Однако формула дает отрицательное число, что повлияет на дальнейшие вычисления, так что в этом случае необходимо сказать, что требуется сделать 0 подходов. Во-вторых, число B_i может оказаться равным нулю, что означает необучаемость школьника. В этом случае сразу хочется поставить число подходов равным бесконечности (то есть обучить его нельзя). Но не стоит спешить: возможно, он уже достаточно подготовлен, чтобы взять диплом. Чтобы избежать этого неприятного случая, следует сначала сравнивать числа K и A_i, и только потом проверять B_i на равенство нулю. Все вышеприведенные ошибки относятся к online-группам тестов.

2. $N = 10^6$ – число достаточно большое, чтобы требовать какой-нибудь быстрой сортировки, и достаточно маленькое, чтобы эти сортировки успевали с большим запасом.

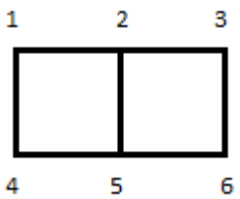
3. Теперь требуется просто пройтись по отсортированному массиву и на каждом шаге проверять, хватает ли нам свободных подходов для обучения очередного школьника. Если да, то необходимо его обучить и уменьшить число свободных подходов. Если нет, то процесс тренировки следует прекратить и вывести ответ. Основная ошибка, которую можно было сделать — "увлечься". Если ответ равен N, то некоторые программы могли попытаться подготовить еще и $N+1$, а то и более школьников. Эта ошибка также относится к онлайновым группам тестов.

Задача J. Вася и его друзья

Автор задачи — М. Пядёркин

Решение на 40 баллов. В данном случае решением может являться практически любой разумный перебор всех циклов. Рассмотрим одну из возможных реализаций этого перебора. Представим нашу табличку в виде графа: узлы будут вершинами, а границы клеток — ребрами. Занумеруем каким-либо удобным образом вершины. Теперь научимся считать количество циклов в этом графе. Для этого заметим, что в каждом цикле есть ровно одна вершина с наименьшим номером. Будем сначала перебирать эту вершину, а затем перебирать все циклы из неё, проходящие по вершинам только с большим номером (алгоритм очень напоминает перебор всех перестановок). В результате, мы посчитаем каждый цикл ровно два раза: в одну сторону и другую.

Рассмотрим пример (см. рисунок). Циклы с фиксированной вершиной 1: $1 \to 2 \to 3 \to 6 \to 5 \to 4$, $1 \to 4 \to 5 \to 6 \to 3 \to 2 \to 1$, $1 \to 2 \to 5 \to 4$, $1 \to 4 \to 5 \to 2$ Циклы с фиксированной вершиной 2: $2 \to 3 \to 6 \to 5$, $2 \to 5 \to 6 \to 3$. Ниже приведена короткая реализация Серёжи Копелиовича (без заголовков):




```
#define forn(i, n) for (int i = 0; i < (int)(n); i++)
#define mp make_pair

const int maxn = 100;

int h, w, K, cnt = 0, SX, SY, u[maxn][maxn];

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

```
void go( int x, int y, int dep )
{
  u[y][x] = 1;
  forn(k, dep ? 4 : 1)
  {
    int x1 = x + dx[k], y1 = y + dy[k];
    if (mp(x1, y1) == mp(SX, SY))
      cnt += (dep + 1 >= 4);
    else if (mp(x1, y1) > mp(SX, SY) && x1 <= w && 0 <= y1 && y1 <= h && !u[y1][x1])
      go(x1, y1, dep + 1);
  }
  u[y][x] = 0;
}

int main()
{
  scanf("%d%d%d", &h, &w, &K);
  if (h * w > 24) /// получит TL
    return 0;
  forn(y, h)
    forn(x, w)
      go(SX = x, SY = y, 0); /// количество циклов с минимальной вершиной (SX, SY)
  printf("%d\n", (cnt - 1) % K + 1);
  return 0;
}
```

Решение на 100 баллов. Данная задача решается методом динамического программирования по профилю. Если вы не знакомы с этой темой, то настоятельно рекомендуется прорешать более простые задачи на эту тему, например, на сайте http://informatics.mccme.ru в разделе "Динамическое программирование".

Профилем столбца в данной задаче будет являться информация о том, по каким горизонтальным линиям столбца шел наш цикл. К сожалению, недостаточно просто хранить информацию, брали мы эту линию или нет (подумайте, почему). Будем для каждой горизонтальной линии хранить число: 0, если мы эту линию в цикл вообще не брали, и номер связной компоненты в противном случае (компоненты будем нумеровать с 1).

Пусть A_{ip} обозначает количество "корректных слева" циклов (т.е. которые могут быть достроены до правильного цикла), которые находятся в первых i столбцах, и оканчиваются на профиль p (т.е. i-ый столбец каждого из A_{ip} циклов описывается профилем p).

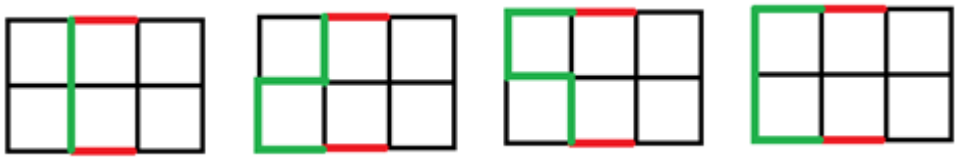


Рис. 1: Например, если $i = 2$, $m = 2$, $p = \{1, 0, 1\}$, то $A_{ip} = 4$.

Теперь сделаем несколько наблюдений. Первое состоит в следующем: пусть мы знали, на какой профиль оканчивался i-й столбец. Тогда, зная маску вертикальных линий (какие берем в цикл,

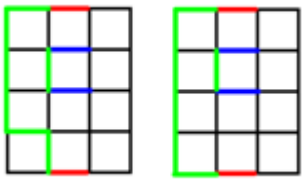
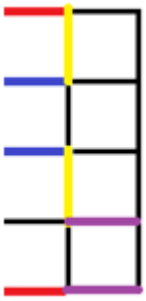


Рис. 2: Более сложный случай, $i = 2$, $m = 4$, $p = \{1, 2, 2, 0, 1\}$, $A_{ip} = 2$.

какие нет) между i-м и $(i + 1)$-м столбцами, можно однозначно восстановить профиль $(i + 1)$-го столбца.



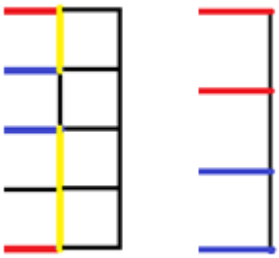
К примеру, пусть $m = 4$, $p = 1, 2, 2, 0, 1$. Если маска вертикальных линий $\{1, 0, 1, 0\}$, то получаем рисунок слева.

По заданным профилю столбца и маске мы однозначно определяем, какой будет профиль следующего столбца — $\{0, 0, 0, 1, 1\}$ (легко убедиться, что никакой другой профиль не подходит). Заметим, что некоторые маски могут быть недопустимы для некоторых профилей, например, при $p = \{1, 2, 2, 0, 1\}$ невозможна $\{1, 1, 0, 0\}$. Маска $\{0, 1, 0, 0\}$ также недопустима для этого профиля: в этом случае мы объединим две синих палочки, и, таким образом, замкнем компоненту связности, что можно делать только в том случае, если компонента всего одна.

Второе наблюдение состоит в том, что различных компонент связности, при ограничении $m \leqslant 8$ может быть не более чем 4 (от каждой компоненты должно быть хотя бы по две линии). Таким образом, количество различных профилей невелико.

Рассмотрим одно из возможных решений, основанное на данных наблюдениях. Сгенерируем все корректные профили для данной маски (переберем все возможные, откинем заведомо неподходящие — например, если есть компонента с номером 3 и нет компоненты с номером 2, или же отрезков каждого цвета не ровно по два). После этого посчитаем динамику A_{ip} следующим образом. Переберем все возможные маски, рассмотрим профиль q, который получится после присоединения этой маски к профилю p, и увеличим $A_{i+1,q}$ на A_{ip}. Заметим, что переходы из каждого профиля p (т.е. все профили q такие, что из профиля p при некоторой маске можно получить профиль q) можно насчитать заранее. Инициализация состоит в том, что для всех профилей, которые получаются при переходе из пустого профиля по какой-либо маске (т.е. для всех профилей, которые могут являться началом цикла), и для всех i нужно положить $A_{ip} = 1$.

Ответом на вопрос задачи является сумма A_{ip} по всем i по всем таким профилям, из которых можно перейти в пустой профиль по некоторой маске (это означает, что можно в данном месте закончить цикл).



Например (рисунок справа), из профиля $\{1, 2, 2, 0, 1\}$ можно перейти в пустой с маской $\{1, 0, 1, 1\}$, а из профиля $\{1, 1, 2, 2\}$ нельзя ни при какой маске.

Задача К. Отрезки на прямой возвращаются

Е. Андреева и Е. Курпилянский

Задача решается методом сканирующей прямой. Будем двигаться по оси координат слева направо. При этом у нас могут возникать события бывают двух видов: начало отрезка и конец отрезка. Обрабатывая эти события, будем поддерживать двусвязный список начавшихся и еще не

закончившихся отрезков, отсортированных по возрастанию координаты левого конца (а при их равенстве по убыванию координаты правого). Тогда если мы находим начало отрезка, достаточно добавить этот отрезок в конец этого списка. При обработке конца отрезка мы, собственно, и найдем ответ для данного отрезка. В качестве ответа можно взять отрезок, предшествующий данному в поддерживаемом нами списке (если данный отрезок находится первым в списке, то ответом для него будет 0), после этого конечно же необходимо удалить закончившийся отрезок из списка. Почему выбранный нами отрезок является правильным ответом? Он начинается раньше данного (по построению списка) и заканчивается позже, так как конец этого отрезка ещё не обработан. Также нетрудно понять, что если какой-то другой отрезок также содержит внутри себя данный, то он обязан лежать в нашем списке до данного отрезка. Поскольку выбранный отрезок является ближайшим к данному в списке, то никакой другой отрезок не находится между ними.

Оценка сложности решения - $O(N \log N)$ на сортировку событий + $O(1)$ на обработку каждого события. Так всего событий $2N$, то суммарная сложность $O(N \log N)$.

Задача L. Отрезки на прямой возвращаются–2 *Е. Андреева и Е. Курпилянский*

Эта задача также решается методом сканирующей прямой. Сначала ознакомьтесь с разбором задачи K, так как решение этой задачи получится путем модернизаций предыдущего. По-прежнему, будет два типа событий: начало и конец отрезка, при обработке событий будем поддерживать тот же самый список, что и в задаче K. Когда какой-то отрезок заканчивается, попытаемся проверить правильность ответа для данного отрезка. Если ответ участника или ответ жюри равен 0, то проверить ответ участника не представляет труда. В противном случае проверим для начала, что выбранный участником отрезок содержит данный. После этого, как было доказано, можно быть уверенными, что выбранный отрезок лежит в нашем списке, причем до рассматриваемого нами отрезка. Заметим, что все отрезки в списке, находящиеся между ответом участника и рассматриваемым, содержат в себе рассматриваемый, а также могут лежать внутри выбранного участником отрезке. Итак, осталось проверить, содержится ли один из этих отрезков внутри выбранного. Так как левый конец всех этих отрезков внутри выбранного, ответ участника правильный только в том случае, если отрезок, заканчивающийся раньше остальных, закончится после выбранного. Другими словами, минимум среди координат правых концов этих отрезков должен быть больше координаты правого конца отрезка, заявленного участником как ответ. Чтобы быстро находить минимальную координату, будем использовать структуру данных «дерево отрезков». Она позволяет изменять элементы массива и находить минимум на отрезке из этого массива за $O(\log N)$. Хотя нам необходимо искать минимум на отрезке из списка, дерево отрезков удастся использовать, так как мы никогда не добавляем отрезки в середину списка. При удалении отрезка из списка нужно в массиве для дерева отрезков записать заведомо большое число, тогда взятие минимума на отрезке массива будет эквивалентно взятию минимуму из элементов, находящихся в списке.

Оценка сложности решения - $O(N \log N)$ на сортировку событий + $O(\log N)$ на обработку каждого события. Так всего событий $2N$, то суммарная сложность $O(N \log N)$.

Краткие разборы писали: Василевский Борис, Евстропов Глеб, Ивлев Фёдор, Курпилянский Евгений, Полднев Антон, Пядёркин Михаил.