Для решения подзадачи 3 требуется быстрее отвечать на запрос. Для этого используем метод «двоичных подъемов». Посчитаем величину go[*j*][*i*], которая равна максимальному номеру станции, до которого можно доехать от станции, используя не более 2^j билетов.

Код, вычисляющий go[*j*][*i*] по go1[*i*] приведен ниже.

```
for (int i = 1; i <= sz; i++) {
    for (int j = 0; j < n; j++) {
        int u = go[i - 1][j];
        go[i][j] = go[i - 1][u];
    }
}
```

В качестве sz следует выбрать минимальную величину, такую что $2^{sz} \geq n$.

Теперь для получения ответа на запрос будем делать все уменьшающиеся «прыжки» по степеням двойки. Код, который отвечает на запрос с использованием массива go приведен ниже.

```
// перемещаемся от fr до to
if (go[sz][fr] < to) {
    cout << to << endl;
} else {
    int steps = 0;
    int curK = sz;
    while (go1[fr] < to) {
        while (go[curK][fr] >= to) {
            curK--;
        }
        steps += 1 << curK;
        fr = go[curK][fr];
    }
    cout << steps + 1 << endl;
}
```

Задача 5 «Три сына»

Заметим следующее: чтобы минимизировать сумму квадратов необходимо стараться выбрать числа *a, b* и *c* близкими к *n*/3. Формализуем это утверждение.

Докажем, сначала, вспомогательный факт про два числа. Пусть различные положительные целые $a < b$ таковы, что $a + b = m$. Тогда если сумма $a^2 + b^2$ минимальна, то $b - a \leq 2$. Действительно, пусть $b - a > 2$, тогда $a + 1 \neq b - 1$ и $(a + 1)^2 + (b - 1)^2 = a^2 + 2a + 1 + b^2 - 2b + 1 = a^2 + b^2 + 2(a - b) + 2 < a^2 + b^2 - 4 + 2 < a^2 + b^2$, следовательно взяв вместо *a* и *b*

числа $a + 1$ и $b - 1$, мы получим два различных числа с такой же суммой, но меньшей суммой квадратов.

Рассмотрим теперь три числа a, b и c, которые требуется найти в задаче. Применив предыдущее утверждение для a и b при фиксированном c, а также для b и c при фиксированном a, получим, что все числа не более чем на 4 отстоят от значения $n / 3$.

Получаем следующее решение: переберем все тройки целых положительных чисел, где каждое число, не более чем на 4 отличается от $n / 3$ и выберем среди них тройку с минимальной суммой квадратов.

Приведем пример программы на Си++, которая решает поставленную задачу.

```
int n3 = n / 3;
int delta = 4;
int minv = max(n3 - delta, 1);
int maxv = min(n3 + delta, n);
long long best = (long long) n * n;
int ba, bb, bc = 0;
for (long long a = minv; a <= maxv; a++)
    for (long long b = a + 1; b <= maxv; b++)
        for (long long c = b + 1; c <= maxv; c++)
            if (a + b + c == n && a*a + b*b + c*c < best) {
                best = a * a + b * b + c * c;
                ba = a;
                bb = b;
                bc = c;
            }
cout << ba << " " << bb << " " << bc << endl;
```

Отметим, что вместо числа 4 можно использовать и любое большее значение. Главное, чтобы оно не было *слишком* большим, чтобы решение по прежнему проходило по времени.

Частичные решения для подзадач 1, 2 и 3 позволяют получить частичные баллы в случае неполного развития этой идеи.

Для решения подзадачи 1 можно перебрать все возможные значения a, b и c и выбрать из них оптимальное. Время работы $O(n^3)$.

Для решения подзадачи 2 следует заметить, что можно перебрать лишь два значения, например, a и b, а значение с вычислить из равенства $a + b + c = n$. . Время работы $O(n^2)$.

Наконец, для решения подзадачи 3 можно, например, перебрать большее из трех значений: c, и заметить, что $a + b = n - c = m$, а для минимизации $a^2 + b^2$ следует взять $a = (m - 1) / 2$, $b = (m + 1) / 2$ при нечетном m и $a = m / 2 - 1$, $b = m / 2 + 1$ при четном m. Время работы O(n).

Задача 6 «Счет в гипершашках»

Рассмотрим три случая: все три игрока набрали равное число баллов, все три игрока набрали попарно различное число баллов, либо все два игрока из трех набрали равное число баллов, а у третьего число баллов отлично от тех двух.

Научимся находить количество вариантов счета в каждом из трех случаев, получившиеся значения необходимо сложить. Отметим, что в подзадаче 1 выполнено условие, что $k = 1$, и следовательно имеет место только первый случай, а в подзадаче 3 все числа различны, и следовательно имеет место только второй случай. Этим можно воспользоваться для написания частичных решений для этих подзадач.

Поместим входные данные в массив и отсортируем его. Одним проходом по получившемуся массиву заменим числа на пары (x_i, c_i), где c_i – количество раз, которое значение x_i встречается на карточках у Андрея, x_i во всех парах различны. Заметим, что массив пар получился отсортированным по по x_i.

```
cin >> n >> k;
vector<long long> x(n);
for (int i = 0; i < n; i++) {
    cin >> x[i];
}
sort(x.begin(), x.end());
vector<pair<long long, int>> p;
int pr = -1;
for (int i = 0; i < n; i++) {
    if (i == n - 1 || x[i] != x[i + 1]) {
        p.push_back({x[i], i - pr});
        pr = i;
    }
}

int m = p.size();
```

Андрей может показать счет, в котором все три игрока набрали равное число баллов, в случае, если у него есть хотя бы три карточки с этим числом баллов. Поэтому число таких вариантов, для которых можно показать счет, равно количеству чисел, которые встречаются хотя бы 3 раза. Ниже приведен код на Си++, который считает число таких вариантов.

```
long long ans1 = 0;
  for (int i = 0; i < m; i++) {
      if (p[i].second >= 3) {
          ans1++;
      }
  }
```

Для определения числа вариантов в остальных двух случаях требуется применить метод двух указателей. Для определения количества вариантов счета, в котором баллы всех игроков различны, сделаем следующее. Для каждого варианта максимального из баллов трех игроков найдем минимальное значение баллов другого игрока, которое можно показать и оно не нарушает условия, что баллы различаются не более чем в k раз. Пусть для максимального значения x_i соответствующее минимальное значение x_j. Тогда два других значения баллов можно выбрать среди значений x_j, x_{j+1},…, x_{j-1}, то есть существует $(i - j)(i - j - 1) / 2$ способов это сделать. Поскольку 3 различных числа можно упорядочить 6 способами, это число необходимо умножить на 6.

Просуммируем эти значения по всем i. Заметим, что при увеличении i значение j также может только увеличиваться, поэтому проход занимает O(n).

Приведем код на Си++, который подсчитывает это число вариантов.

```
long long ans2 = 0;
int j = 0;
for (int i = 0; i < m; i++) {
    while (j < i && p[j].first * k < p[i].first) {
        j++;
    }
    ans2 += 6LL * (i - j) * (i - j - 1) / 2;
}
```

Наконец, аналогичный метод применим для подсчета количества вариантов счета, когда два игрока набрали равное количество баллов, а третий игрок набрал отличное от них количество баллов.

```
long long ans3 = 0;
j = 0;
for (int i = 1; i < m; i++) {
    while (j < i && p[j].first * k < p[i].first) {
        j++;
    }
    if (p[i].second > 1) {
        ans3 += 3 * (i - j);
    }
}
j = m - 1;
for (int i = m - 2; i >= 0; i--) {
    while (j > i && p[i].first * k < p[j].first) {
        j--;
    }
    if (p[i].second > 1) {
        ans3 += 3 * (j - i);
    }
}
```

Для решения подзадачи 1 достаточно рассмотреть вариант, что у всех игроков одинаковое число баллов. Отметим, что дополнительное ограничение $1 \le x_i \le 100\,000$ позволяет обойтись без сортировки пар или сложных структур данных и запоминать количество карточек с каждым числом в массиве.

Для решения подзадачи 2 можно перебрать все тройки карточек, для каждого варианта счета в трехмерном массиве отметить, можно ли получить этот счет, и вывести количество вариантов, которое можно получить.

Для решения подзадачи 3 требуется реализовать описанное выше решение, но не требуется разбирать случай, когда два игрока набрали равное число баллов.

Задача 7 «Интересные числа»

Для решения этой задачи необходимо применить динамическое программирование.

Обозначим количество интересных чисел от L до R как $c(L, R)$. Заметим, что $c(L, R) = c(1, R) - c(1, L - 1)$, поэтому достаточно научиться считать количество интересных чисел, не превышающих заданного R.

Рассмотрим сначала решение третьей подзадачи, где $R = 10^k$. Поскольку само число 10^k интересным не является, задача сводится к подсчету количества интересных чисел, состоящих из k цифр, причем ведущие нули разрешаются (будет напрасно посчитано число 0, можно затем вычесть 1 из ответа). Используем динамическое программирование: обозначим как d[i][j] количество интересных чисел из i цифр, последняя цифра которых равна j. Тогда d[1][j] = 1 для всех j, а для $i > 1$ выполнено равенство d[i][j] = sum(d[$i - 1$][k], $k = 0..j$). Отметим, что все вычисления необходимо производить по модулю $10^9 + 7$, так что реализация работы с длинными числами не требуется.

Обратимся теперь к полному решению. Если R не является степенью 10, то не все интересные числа из k цифр подходят. Тем не менее, если рассмотреть префикс интересного числа, то если в нем есть хотя бы одна цифра строго меньшая соответствующей цифры R, то продолжение может быть любым. Иначе все цифры должны совпадать с соответствующим префиксом R и следующая цифра не должна превышать очередной цифры R.

Переберем общий префикс интересного числа и R. Пусть он равен k. Для каждого значения k запустим отдельное вычисление. Зафиксируем k. Будем использовать такое же динамическое программирование: d[i][j] будет как и раньше содержать количество интересных чисел из i цифр, в которых последняя цифра j, но теперь первые Ш цифр должны совпадать с соответствующими цифрами R, а следующая цифра должна быть строго меньше. Тогда формула для пересчета не меняется, а вот начальные значения меняются: если первые k цифр числа R идут в неубывающем порядке, то d[$k + 1$][j] = 1 для тех j, которые больше или равны k-й цифре числа R и строго меньше его (k+1)-й цифры.

Заметим, что никаких арифметических действий с числом R не производится, поэтому реализация «длинной арифметики» для решения этой задачи не требуется.

Для решения подзадачи 1 достаточно перебрать все числа из диапазона от L до R и для каждого из них проверить, является ли оно интересным.

Для решения подзадачи 2 достаточно перебирать только интересные числа, сохраняя в переборе уже поставленную часть числа и последнюю цифру, будем дописывать только те цифры, которые приводят к сохранению свойства интересности.

Задача 8. «Гармоничная последовательность»

Обозначим первые два числа гармонической последовательности как x и y. Тогда сама последовательность будет иметь вид x, y, $(y - x)$, $-x$, $-y$, $(x - y)$, x, y, ... Заметим, что последовательность имеет цикл длины шесть. Если изначальная последовательность имеет вид b_1, b_2, b_3, ..., то необходимо подобрать целые числа x и y, которые минимизируют

функцию $|x - b_1| + |y - b_2| + |(y - x) - b_3| + |-x - b_4| + |-y - b_5| + |(x - y) - b_6| + ... = |x - b_1| + |y - b_2| + |(y - x) - b_3| + |x - (-b_4)| + |y - (-b_5)| + |(y - x) - (-b_6)| + ...$ Таким образом, необходимо минимизировать сумму модулей величин, которые можно разбить на три группы. Слагаемые первой группы имеют вид $|x - a|$, второй $|y - b|$, третей $|(y - x) - c|$.

Заметим, что существует пара (x, y), которая минимизирует сумму модулей, такая, что как минимум в двух из трех групп значения хотя бы одного модуля равно нулю. Это свойство можно легко доказать от противного. Также заметим, что если зафиксировать, какие именно модули равны нулю, то можно однозначно восстановить числа x и y. Таким образом получаем решение за $O(n^3)$: зафиксируем две из трех групп, переберем, какие именно модули равны нулю, восстановим последовательность, обновим ответ.

Пусть все числа в исходной последовательности по модулю не превосходят A. Заметим, что существует правильный ответ в котором одно из чисел x и y не превосходит по модулю A, а другое $2A$. Это справедливо, так как существует модуль вида $|x - a_i|$ или $|y - b_i|$, который равен нулю, и из которого можно получить значение одной из переменных. A в худшем случае значение другой переменной можно будет определить из уравнения вида $|(y - x) - c_j| = 0$.

Таким образом, для решения подзадачи 1 можно просто перебрать значения трех элементов последовательности в пределах от –20 до 20.

Для решения подзадачи 2 можно перебрать значение x и y в пределах от –200 до 200, восстановить последовательность и выбрать лучший вариант.

Для решения подзадачи 3 можно перебрать значения x, получить выражение вида $|y - y_1| + |y - y_1| + ...$, которое необходимо минимизировать. Далее перебрать модуль, который будет равен нулю и, используя префиксные и суффиксные суммы, посчитать значение выражения.

Для решения подзадачи 4 можно перебрать, какие два модуля равны нулю и с помощью префиксных и суффиксных сумм посчитать ответ.

Для решения подзадачи 5 необходимо заметить следующий факт. Если для некоторого x найдено минимальное значение y, которое является оптимальным, то для большего x оптимальное значение y не может быть меньше найденного. Таким образом полное решение задачи выглядит следующим образом. Разобьем все модули на три группы. Отсортируем модули в порядке увеличения значения константы в них. Переберем две группы, в которых будут модули равные нулю (без ограничения общности будем считать, что это группы вида $|x - x_i|$ и $|y - y_i|$). Переберем, какой из модулей первой группы равен нулю. Будем поддерживать указатель на модуль из второй группы, при обнулении которого

получается оптимальный ответ. При изменении выбранного модуля первой группы, попытаемся использовать модуль с большей константой во второй группе (пока следующий модуль дает ответ лучше чем текущий, будем увеличивать указатель). Для того, чтобы посчитать ответ для конкретной пары (x, y) необходимо понять, какие из модулей третей группы раскроются с плюсом, а какие с минусом. Для этого можно воспользоваться двоичным поиском, а далее с помощью префиксных и суффиксных сумм посчитать значение выражения. Итого общая сложность решения равна $O(n \log n)$.

В заключение заметим, что функция, которую необходимо минимизировать, является выпуклой вниз как по x, так и по y, поэтому для решения задачи можно также воспользоваться различными численными методами, например градиентным спуском или двумерным троичным поиском. В зависимости от эффективности вычисления значения целевой функции такие решения могут набрать от 44 до 100 баллов.