,

Задача A. Гулять так гулять

Для решения задачи на полный балл достаточной идти в порядке увеличения дней, поддерживать счётчик количества нераспределённых праздников. Как только мы встретили день, рекомендованный советниками, мы увеличиваем счётчик на их количество.

В каждый из дней, если значение счетчика праздников больше нуля, мы пытаемся поставить праздник и уменьшить счетчик на один. Если мы обработали все N дней и счётчик стал равен нулю — завершаем алгоритм, иначе — продолжаем расставлять праздники, но уже не увеличивая счетчик. Сложность $O(N + SUM)$.

Задача B. Новогодний поезд

Для решения данной задачи давайте воспользуемся жадным алгоритмом. Будем поддерживать последний вагон на каждом пути в депо. При обработке очередного вагона в поезде, отправим его на путь, на котором находится наибольший вагон, меньший текущего. Если такого пути нет, то нам придётся занять новый путь. Данная жадность работает, так как очередной вагон обязан быть последним на каком-то пути, нам не выгодно увеличивать количество занятых путей, и нам выгодно дописать его в конец пути с наибольшим возможным вагоном, чтобы значения в множестве значений вагонов на концах путей были как можно меньше. Наивно это можно реализовать за $O(N^2)$ и получить 60 баллов, если воспользоваться $std :: set$ для хранения номеров последних вагонов на путях депо, то можно получить решение за $O(NlogN)$ и 100 баллов.

Задача C. Горная трасса

В начале избавимся от цикличности трассы. Рассмотрим максимальный элемент, никакую ступень не выгодно делать больше, чем его высота. Тогда выпишем массив, начиная с этой позиции, и допишем в конец массива копию максимального элемента. Тогда в формуле сложности трассы нужно будет сравнивать только соседние элементы.

Рассмотрим, как изменяется сложность трассы от добавления одного блока на позицию i. Если $a_i \geqslant a_{i-1}$ и $a_i \geqslant a_{i+1}$, то она увеличивается на 2. Если $a_i < a_{i-1}$ и $a_i \geqslant a_{i+1}$ или же $a_i \geqslant a_{i-1}$ и $a_i < a_{i+1}$, то она не изменяется. Если же $a_i < a_{i-1}$ и $a_i < a_{i+1}$, то уменьшается на два.

Это означает, что чтобы уменьшить сложность трассы, необходимо увеличить высоту таких ступеней, что слева и справа были ступени большей высоты. Тогда, если рассмотреть какой-то локальный минимум ступеней(где они все одинаковой высоты, а с краев ступени выше), то чтобы уменьшить сложность трассы на два нужно увеличить высоту всех ступеней в этом локальном минимуме на 1. Тогда, для решения задачи за $O(kN)$ можно действовать жадно: найдем все локальные минимумы, выберем среди них минимум с минимальным число ступеней в группе, увеличим их высоты на 1, и уменьшим k. У нас при этом может появиться новый локальный минимум, который тоже будет рассматриваться на следующем шаге. Такое решение набирает 30 баллов.

Для более оптимального решения нужно поддерживать все локальные минимумы в $std::set$ отсортированные по длине. Еще можно просто понять для каждого минимума, сколько раз мы можем увеличивать высоту ступеней в нём: достаточно посмотреть на высоты ступеней слева и справа от него, и взять минимум по разницам высот. Когда мы увеличим высоты ступеней настолько, что этот локальный минимум исчезнет, нужно проверить, появились ли какие-то другие локальные минимумы. Заметим, что в этот момент высота либо правого, либо левого минимума будет совпадать с высотой ступеней на бывшем локального минимуме. Мы можем использовать систему непересекающихся множеств для того, чтобы поддерживать множества ступеней на одинаковой высоте, потому что если в одной группе ступени были одинаковой высоты, наш алгоритм будет их увеличивать либо все, либо ни одной. Таким образом, используя СНМ можно быстро находить новые локальные минимумы, и добавлять их в set. Такое решение работает за $O(NlogN)$, потому что суммарно мы создадим не больше $O(N)$ локальных минимумов (для появления нового какие-то две группы одинаковых элементов должны объединиться), и оно набирает 100 баллов.

На самом деле задачу можно красиво решить и за $O(N)$. Давайте найдем все возможные горизонтальные отрезки, которые возникли бы в процессе решения с поиском локальных минимумов, если бы $K- > inf$. И будем подсчитывать количество отрезков каждой длины в массиве подсчета (различных длин горизонтальных отрезков у нас не больше, чем N.

Давайте поддерживать убывающий стек текущих левых границ «впадин». Если очередная высота a_i меньше, чем последний элемент из стека, то просто добавляем её (а на самом деле её индекс в стек), иначе мы можем выделить некие горизонтальные отрезки для замощения: пока последний элемент стека меньше нас, удаляем его, и получаем сколько-то новых отрезков от элемента после последнего в обновленном стеке до i-го столбика (как бы замощаем всё до минимума из высоты нового последнего элемента стека и a_i, а на самом деле в массиве подсчета увеличиваем значение соответствующего элемента на количество таких горизонтальных отрезков). Если в какой-то момент, мы встретили элемент равный a_i, то мы удаляем его. В конце добавим индекс элемента a_i в стек. Таким образом мы найдем для каждой длины горизонтального отрезка, сколько раз можно её использовать для уменьшения сложности трассы, и дальше можно просто жадным алгоритмом набрать по увеличению длин уменьшение трассы, пока K достаточно большое. Решение работает, потому что если в какой-то момент мы хотим взять большой отрезок, то все маленькие отрезки внутри него мы уже взяли, и там все ступени, лежащие ниже этого отрезка, будут увеличены до нужного уровня.

Код этой части:

```
st.pb(0);
for (int i = 1; i < n; i++) {
    while (!st.empty()) {
        int last = a[st.back()];
        if (last > a[i]) {
            break;
        }
        st.pop_back();
        if (last == a[i]) {
            break;
        }
        int tl = st.back() + 1;
        int tr = i - 1;
        int cnt = min(a[st.back()], a[i]) - last;
        cnts[tr - tl + 1] += cnt;
    }
    st.pb(i);
}
```

Задача D. Чимбулак

В решении на 30 баллов за $O(N^3)$ достаточно посчитать расстояния между всеми парами вершин с помощью модифицированного алгоритма Флойда, который считает еще количество кратчайших путей между вершинами. Тогда нужно будет найти максимум по всем длинам, и просуммировать количество путей с такой длиной.

В решении на 60 баллов можно воспользоваться тем фактом, что все рёбра невзвешенные и можно просто запустить BFS N раз(который так же будет считать количество кратчайших путей), получив решение за $O(N^2)$.

Связный граф из N рёбер — это дерево с дополнительным ребром. Заметим, что оно представляет из себя цикл, из вершин которого исходят другие корневые деревья. Для того, чтобы обработать все пары вершин, находящиеся в одном таком дереве, можно посчитать стандартную динамику по поддеревьям. В начале посчитаем максимальную глубину вершин в каждом поддереве, и сколько в нем вершин на такой глубине. Затем достаточно перебрать наименьшего общего предка двух вершин, перебрать в каком сыне лежит первая вершина, и пересчитать ответ.

Для обработки вершин из разных деревьев нужно уметь правильно обрабатывать цикл. Предподсчитаем для каждой вершины цикла количество самых глубоких вершин в её поддереве и их глубину. Расстояние между вершинами, которые лежат в поддеревьях i и j вершины цикла на глубинах h_i и h_j равно $h_i + h_j + min(|i-j|, len-|i-j|)$, где len - длина цикла. Тогда, можно перебрать

вершины на цикле, и рассмотреть два случая: до другого поддерева от нее ближе идти налево или направо. В обоих случаях, чтобы найти наиболее удаленную вершину в другом поддереве достаточно посчитать значение максимума на отрезке и количество максимумов. Это можно сделать с помощью дерева отрезков/разреженной таблицы или другой подобной структуры, и получить решение за $O(NlogN)$, которое уже набирало 100 баллов.

При переходе к следующей вершине на цикле мы изменим на один значение расстояний в структуре.

Нужно не забыть про случай, когда существуют два кратчайших пути: это возможно, если вершины находятся в поддеревьях, расположенных на цикле диаметрально противоположно.

Но решение можно улучшить до линейного: заметим, что отрезки, где выгодно идти налево/-направо имеют фиксированную длину(половина длины цикла), и максимум в них можно искать с помощью алгоритма поиска максимума в скользящем окне.