# Задача A. Торговый центр

*Автор задачи и разработчик: Николай Будин*

Если Кендис воспользуется $i$-м автобусом, она приедет в ТЦ спустя $t_i + l_i$ минут. Ответом является минимум из этих значений.

# Задача B. Переливание жижи

*Автор задачи и разработчик: Даниил Орешников*

Эту задачу можно было решить обычным поиском в ширину. Поначалу кажется, что возможных состояний порядка $n_1 \cdot n_2 \cdot n_3$, но эту оценку можно сильно уменьшить, заметив, что

1. уже после первого переливания всегда найдется либо пустой, либо полный бак
2. сумма объемов жижи в баках не меняется

Заметим, что в таком случае состояние баков можно описать четырьмя параметрами: номер пустого или полного бака (3 варианта), пустой этот бак или полный (2 варианта), номер еще одного бака (2 варианта) и количество жижи в нем (не более $\max(n_1, n_2, n_3)$ вариантов). Итого — потенциальных состояний системы из трех баков не более $12 \cdot \max(n_1, n_2, n_3)$, так что работающий за линейное время поиск в ширину позволит найти минимальное число переливаний для достижения требуемого состояния.

Важно обработать случай, когда $b_1$, $b_2$ и $b_3$ соответствуют исходным объемам и не достижимы после первого же переливания, а также не забыть о том, что порядок $b_i$ не важен, и соответствующих им конечных состояний несколько.

# Задача C. Поиск пирамиды

*Автор задачи: Владислав Власов, разработчики: Дмитрий Гнатюк и Даниил Орешников*

Для решения задачи воспользуемся деревом отрезков. Заметим, что нам необходимо находить максимальный по длине отрезок на всем массиве, в котором элементы сначала возрастают, а затем убывают. Для подсчета соответствующего значения в вершине дерева отрезков давайте хранить 1. наибольший такой отрезок, 2. наибольший отрезок с началом в левой границе вершины, 3. наибольшие отрезок с концом в правой границе вершины.

Для пересчета соответствующего значения в вершине через детей достаточно заметить, что ответ для нее либо полностью лежит в левом ребенке, либо полностью в правом, либо на стыке, но тогда и только тогда, когда

- либо слева в конце нет убывающей части и конец левого ребенка меньше начала правого
- либо справа в начале нет возрастающей части и конец левого ребенка больше начала правого
- либо левая половина заканчивается возрастающим отрезком, а правая — убывающим, тогда не важно в каком они отношении

Нетрудно заметить, что все эти величины можно пересчитывать за $\mathcal{O}(1)$, аккуратно разбирая случаи различного местоположения начала и конца ответа, а операция обновления в дереве отрезков будет работать за $\mathcal{O}(\log n)$, так как затронет только две вершины на нижнем уровне. Таким образом, мы получаем решение, работающее за $\mathcal{O}(\log n)$ на операцию.

Такой же асимптотики можно было добиться, храня описанные отрезки в куче (`std::set`), и проводя операции их склеивания и разделения при изменении элементов массива, однако такое решение может потребовать разбора порядка 37-и случаев.

# Задача D. Игра в Мафию

*Автор задачи и разработчик: Григорий Хлытин*

Заметим, что мафией могут быть только выжившие после всех ночей игроки, и по условию их количество равно $k - m$ ($1 \leqslant k - m \leqslant 15$). Далее будем называть их подозреваемыми. Давайте переберем всевозможные подмножества подозреваемых, для каждого такого подмножества проверим,

могли ли игроки из этого подмножества убить всех жертв. После этого выберем минимальное по размеру подмножество из всех прошедших проверку — это и будет ответ на задачу.

Заметим, что для такой проверки нам нужна лишь информация, с кем из подозреваемых контактировала каждая жертва в ночь убийства. В качестве одного из возможных решений можно было построить таблицу «подозреваемые-жертвы» размера $(k-m)\cdot m$, а затем проверить, что каждая жертва пересекалась хотя бы с одним из выбранных подозреваемых.

Такое решение работает за время $\mathcal{O}\left(2^{k-m}\cdot(k-m)\cdot m\right)$.

# Задача E. Японский кроссворд

*Автор задачи и разработчик: Николай Будин*

В задаче дано поле размера $n\times m$ и требуется в каждой строке и каждом столбце найти длины отрезков из «#». Чтобы найти длины отрезков в строке, нужно пробежаться по ее символам слева направо. Каждый раз, когда встречается символ «#», если предыдущий символ тоже равнялся «#», то нужно увеличить длину последнего отрезка на 1, а иначе нужно добавить новый отрезок длины 1.

# Задача F. Минимальная строка

*Автор задачи: Даниил Орешников, разработчик: Ильдар Загретдинов*

Ответом является лексикографически минимальная строка длины $|a|$, составленная из символов из строк $a$ и $b$. Действительно, давайте на первое место поставим минимальный из символов, представленных в строках. Если он находится в строке $b$, то поменяем первый элемент из $a$ с этим элементом. Если минимальный элемент находится в строке $a$, то поменяем его с любым элементов из $b$ и аналогично предыдущему варианту, поменяем его с первым символом из $a$. Повторяя такое действие для всех позиций первой строки, мы получим лексикографически минимальную строку.

# Задача G. Большое задание

*Автор задачи и разработчик: Николай Будин*

В задаче было дано дерево, каждая вершина которого покрашена в один из $m$ цветов. Требовалось посчитать количество связных подграфов, которые содержат вершины всех цветов.

Для решения этой задачи, можно воспользоваться формулой включений-исключений. Переберем подмножество цветов $s$ и найдем количество связных подграфов, которые не содержат вершины, не принадлежащие множеству $s$. Пусть это количество равно $f(s)$. Тогда ответом является:

$$\sum_{s=0}^{2^m-1} f(s)\cdot(-1)^{m-|s|}$$

Чтобы лучше понять, почему эта формула работает, рассмотрим несколько слагаемых. Слагаемое $f(2^m-1)$ это просто количество всех связных подграфов. Вычтем из них те, в которых нет цвета $x$, потому что они нам не подходят — для этого в сумме есть слагаемые $-f(2^m-1-2^x)$. Однако, теперь мы дважды вычли подграфы, в которых нет ни цвета $x$, ни цвета $y$ $(x\neq y)$. Прибавим их один раз — слагаемое $f(2^m-1-2^x-2^y)$. И так далее.

Теперь научимся вычислять $f(s)$. Оставим в дереве только вершины, цвет которых находится в множестве $s$. Ответом является сумма по всем компонентам связности количество различных связных подграфов в компоненте. Поэтому, осталось научиться считать количество связных подграфов в дереве. Это делается простой динамикой по поддеревьям. Значение $dp[v]$ равно количеству различных связных подграфов, лежащих в поддереве $v$ и содердащих $v$.

# Задача H. Большой батут

*Автор задачи: Степанов Семен, разработчик: Арсений Кириллов*

Так как $n\leqslant 9$, то можно перебрать все перестановки точек. Тогда можно проверить, пересекаются ли какие-нибудь рёбра и посчитать площадь многоугольника за $O(n^2)$. И того итоговая асимптотика $O(n!\cdot n^2)$.

# Задача I. Ксероксинатор

*Автор задачи: Григорий Шовкопляс, разработчик: Ильдар Загретдинов*

Давайте хранить очередь, где каждый элемент — это пара из количества элементов, добавленных в секунду $i$, и самой секунды $i$. Тогда, на каждой итерации мы добавляем в очередь пару, а затем вынимаем из начала очереди $b$ элементов, пересчитывая ответ. Ответ можно пересчитать, зная количество людей в стоящей в начале очереди группе и время их добавления в очередь.

После того, как итерации окончились, пересчитаем ответ с учетом оставшихся элементов (можно считать, что мы вытаскиваем все оставшиеся элементы на секунде $n + 1$).

# Задача J. Выходной

*Автор задачи: Даниил Орешников, разработчик: Николай Будин*

Заметим, что текущее состояние куба описывается следующими параметрами:

- Число, написанное на нижней грани.
- Число, написанное на грани, смотрящей, например, на север.
- Направление $d$.

Итого, есть $6 \cdot 4 \cdot 4$ различных состояния. Будем эмулировать процесс до тех пор, пока не придем в состояние, в котором уже бывали. Таким образом, мы нашли предпериод и цикл. Если $q_i$ попало в предпериод, то нужно вывести запомненный ответ. Иначе, нужно из $q_i$ вычесть значение $s$, которое было в тот момент, когда мы впервые пришли на цикл, затем взять остаток от деления на число, равное изменению $s$ за один цикл, и затем опять же вывести ответ.

Детали реализации можно посмотреть в авторском решении.