## *Задача 1. Черно-белая графика*

Решение этой задачи состоит в симуляции процесса, описанного в условии. Для хранения заданной логической операции можно использовать массив целочисленного типа `d[0..1,0..1]`, элемент которого `d[i,j]` хранит результат логической операции с первым аргументом *i* и вторым аргументом *j*. Таким образом, значение пиксела результата с координатами *(x, y)* равно `d[a[x,y],b[x,y]]`, где `a[1..h,1..w]` и `b[1..h,1..w]` – целочисленные матрицы, хранящие исходные изображения.

При оценивании решений участников рекомендуется учитывать частичное решение, неправильно обрабатывающее последовательность, описывающую операцию (оценивается из 56 баллов).

## *Задача 2. Газон*

Для каждой прямой, содержащей все пучки травы с равной координатой по оси абсцисс (также можно рассматривать ось ординат), найдем ее точки пересечения с окружностью действия поливальной установки (если такие точки существуют). Получив данные точки, не сложно за время *O(1)* посчитать количество пучков травы на данной прямой, одновременно и политых, и постриженных. Таким образом, время работы решения *O(r)*.

При оценивании решений участников рекомендуется учитывать следующие частичные решения:

1) Решение со временем работы $O(r^2)$, перебирающее все постриженные пучки травы и считающее количество политых среди них (решение оценивается из 55 баллов);
2) Решение, оперирующее 32-битными числами (решение оценивается из 70 баллов).
3) Решение, содержащее различные ошибки при увеличении ответа (решение оценивается из 80 баллов).

## *Задача 3. Трамвай*

Будем поддерживать два упорядоченных множества: пассажиры, которые в данный момент сидят, и пассажиры, которые стоят. Критерий упорядочения таков: пассажир *i* считается «меньше», чем пассажир *j*, если $a_i - b_i$ меньше, чем $a_j - b_j$. Иными словами, самый «большой» пассажир получает больший, чем у остальных, прирост удовлетворения, если он из стоячего положения перейдет в сидячее.

Рассмотрим все события вида «пассажир вошел» или «пассажир вышел» в том порядке, в котором они происходят.

Если пассажир вошел, то мы должны определить, будет ли он стоять или сидеть. При этом бессмысленно заставлять сидеть пассажира, если он любит стоять больше, чем сидеть. Если пассажир желает сесть и место для него еще есть, то посадим его (внесем во множество сидящих); иначе временно внесем его во множество сидящих и попросим встать самого «маленького» пассажира – тем самым мы нанесем наименьший урон суммарной удовлетворенности.

Если выходит стоявший до этого пассажир, то ничего не изменяется – мы лишь извлечем этого пассажира из множества стоящих. Если же выходит сидевший до этого пассажир, то выберем среди стоящих пассажиров того, кто больше остальных хочет сесть («наибольшего» пассажира из стоящих), и попросим его сесть. Мы не станем этого делать, если никто из стоящих пассажиров не желает садиться.

Данную логику реализует несколько различных алгоритмов, использующих структуры данных разной сложности. Достаточно эффективным является использование

сбалансированных деревьев поиска для представления упорядоченных множеств пассажиров – с этими структурами данных время работы алгоритма составит $O(N\log N)$. В эту оценку по времени включена сортировка событий по времени. Чуть более простой, но при данных ограничениях не менее эффективный алгоритм, перебирающий остановки со списками входящих и выходящих на них пассажиров, имеет асимптотику $O(N\log N+P)$.

При оценке решений участников рекомендуется учитывать следующие частичные решения:

4) Неэффективные решения, реализующие операции с упорядоченными множествами за линейное время (решение оценивается из 80 баллов).
5) Неэффективные решения, имеющие худшие асимптотические оценки, чем $O(N\cdot P)$ (решение оценивается исходя из 60 баллов).
6) Решения, оперирующие 32-битными числами для подсчета ответа (решение оценивается исходя из 60 баллов).

## *Задача 4. Треугольники*

Посчитаем количество равнобедренных треугольников с фиксированной вершиной (назовем ее *B*), противолежащей основанию. Для этого отсортируем все остальные точки по расстоянию до *B*. Пусть количество точек на расстоянии ровно *d* есть *q(d)*, тогда искомое количество будет равно сумме по всем возможным *d* величины $q(d)\cdot(q(d)-1)/2$.

Однако, мы не учли, что с помощью этого алгоритма мы посчитаем и вырожденные треугольники (то есть отрезки прямых вида *ABC*, где |*AB*| = |*BC*|). Чтобы учесть это, мы при равных расстояниях до B будем сортировать точки по полярному углу относительно *B*, а затем за один проход посчитаем, сколько среди таких точек центрально-симметричных пар, и вычтем это количество из ответа. Можно это сделать немного проще – при сравнении точек брать полярный угол по модулю развернутого угла и вычесть количество пар «равных» в этом смысле точек.

Для того, чтобы набрать полный балл, сортировка по полярному углу должна быть выполнена без вычисления значений угла. Такую сортировку можно реализовать в целых числах на основе понятия векторного произведения. В противном случае возникают проблемы с потерей точности.

Возможно также реализовать решение, которое не производит дополнительной сортировки по полярному углу, а вместо этого просматривает точки с фиксированным расстоянием до *B* на предмет центральной симметрии. Анализ дает для этого решения, с учетом геометрической специфики задачи, асимптотическую оценку $O(N^2\log N+N^{2.137})$, что с учетом ограничений все еще достаточно эффективно, а по причине меньшей константы реализации работает несколько быстрее предыдущего.

При оценке решений участников рекомендуется учитывать следующие частичные решения:

1) Неэффективные решения, такие как перебор и проверка всех возможных треугольников (решение оценивается из 40 баллов).
2) Решения с потерей точности, использующие вычисление полярного угла и сортировку по этому значению (решение оценивается из 60 баллов).

Задача 7. Числа.

Предложенная задача решается методом динамического программирования. Состояние – суффикс исходной строки. При переходе перебираем префикс текущего суффикса, пока образуемое число не превышает $C$. Особо требуется рассмотреть случай, когда число начинается с 0. Таким образом, алгоритм работает за $O(n \log C)$.

При оценивании решений участников следует учитывать переборное решение из 70 баллов.

Задача 8. Перестановки.

Предложенная задача решается методом динамического программирования. Состояние – пара из подмножества ($P$) исходного множества ($S$) и номера выделенного элемента ($j$) в нем. Для каждого состояния хранится количество существующих $k$ – перестановок из данного множества $P$, таких что первый элемент – $j$-й (в исходной нумерации). Начальное состояние – $P=\varnothing$. Подмножество $P$ удобно хранить битовыми масками. Переход динамики очевиден. Время работы есть $O(m2^m)$. После вычисления динамики требуется выполнить восстановление ответа, последовательно решая, какое число поставить на следующую позицию.

При оценивании решений участников следует учитывать следующие частичные решения:

1) переборное решение из 50 баллов.
2) решение, оперирующее 32-битными числами из 84 баллов.