1.1. Задача Stars (acm.timus).

Во-первых, необходимо понять, что тот факт, что звезды даны в порядке увеличения y (а внутри одной координаты по увеличению x) очень нам помогает: звезды, которые выше какой-либо из рассматриваемых не изменяют уровень рассматриваемой звезды. Поэтому координату y можно вообще выбросить из рассмотрения.

А по x (вкупе с тем фактом, что по x звезды упорячены тоже по возрастанию) получается достаточная простая матмодель: даны числа, нужно для каждого нового поступащего числа *K* быстро посчитать следующую статистику: выдать количество чисел не превосходящих *K*.

Можно воспользоваться AVL-деревом. Будем в каждой вершине хранить высоту вершины (для поддержки свойства баланса) и еще size – количество вершин в поддереве с корнем в этой вершине. Обновление этих статистик не предоставляет сложности: при поворотах аккуратно пересчитываем за O(1) высоту и размеры поддеревьев (ибо свойства нарушаются только в двух узлах – концы того ребра, относительно которого идет поворот).

Вставка будет работать так: если число К меньше ключа в узле, то кидаем (рекурсивно) элемент в левое поддерево, иначе (если больше **или равен**) – то в правое, причем в этом случае будем аккумулировать такую сумму: `res = res + size(node.left) + 1`. Она как раз и будет отвечать на вопрос «сколько элементов не больше К»: это и понятно, если мы от узла *node* пошли в правое дерево, то гарантированно есть `size(node.left) + 1` чисел не превышающих К.

Вывести ответ не предоставляет труда: может в массиве поддерживать распределение и после обработки всех звезд вывести этот массив.

Код в приложении к письму (вместе с этим файлом).

1.2. Доказать, что в AVL-дереве можно вывести k подряд идущих элеметов за O(k+logN).

1-й способ.

Вообще говоря, можно дополнить каждый узел прямыми указателями на следующий (и предыдущий) элемент, тогда мы можем простой найти какой-то первый элемент (за O(logN)) и за O(k) получить k следующих за ним, просто вызывая функцию SUCC, которая уже работает за O(1). Итог: **O(k+logN)**

Осталось поддерживать указатели при изменениях дерева (очевидно, они не меняются при поворотах). Ну, это просто: при вставке элемента в дерево, вызовем обычные фукнции SUCC и PRED, работающие за логарифм (соответственно, не превышающие время вставки) и обновим указатели. Аналогичное действие совершаем при удалении узла – ищем его SUCC и PRED и обновляем указатели.

2-й способ.

Можно и не хранить дополнительную инфомарцию, а переоценить время работы k вызовов обычных SUCC. Пусть, мы начали с вершины x. (на ее поиск может уйти O(logN) времени)

Этот проход посетит по крайней мере k вершин с ключами k ≥ key(x) – нужные нам элементы.

Также, в обходе могут быть вершины с ключами k ≤ key(x). Однако их число ограничено высотой дерева – это может быть например в том случае, когда начальный узел – лист, а следующий за ним элемент – корень дерева.

И также, в обходе могут встретиться элементы с ключами, $k \geq key(x)$, но которые не входят в нужное нам множество, но их число также ограничено $h = \log N$.

Осталось заметить, что каждое «ребро» в этом дереве при проходе будет просмотрено не более двух раз: один раз при спуске вниз, один раз наверх, поэтому общее время работы можно оценить как $O(3*\log N + k)$ = **$O(k+\log N)$**

1.3. Сохранить высоту вершины в AVL-дереве за O(1) бит.

Как известно из определения, в AVL-дереве высоты поддеревьев не могут отличаться более чем на 1, следовательно, у каждой конкретной вершины может быть только три ситуации, связанных с распределением ее высоты: оба дерева имеют одну высоту, левое поддерево больше правого ровно на 1, либо правое поддерево больше – будем называть это характеристикой вершины. Следовательно, можно в каждой вершине хранить не собственно число, равное высоте, а ее характеристику; так как их всего 3, достаточно двух бит: **00** – первый случай, **01** – второй, **11** – третий.

Высота вершины будет вычисляться так:

```
h(node) = (node.characteristic == 01 {левое поддерево больше}) ? h(node.left) + 1 : h(node.right) + 1
```

Ясно, что такая характеристика считается за O(h) всего дерева, то есть за $O(\log N)$, то есть за время, сравнимое со временем изменения дерева.