Теоретическое решение задачи B.

Задачу будем решать с помощью двумерного дерева отрезков (дерева отрезков, каждый узел которого будет также состоять из дерева отрезков, которые в свою очередь уже будут хранить максимальные значения). Тогда мы сможем достаточно быстро отвечать на запросы бабушки. Хранить дерево отрезков будем в виде двумерного массива $tree$ размером $4n \times 4m$.

Из условия видно, что помимо самого построения двумерного дерева отрезков, к нему будет еще 2 типа запросов – посчитать максимум на подпрямоугольнике и обновить какое-то одно значение в матрице. Каждый из этих запросов можно реализовать как запросы к стандартному одномерному дереву отрезков, но только вместо числового значения каждого отрезка мы будем записывать целое дерево отрезков (имеем фиксированное значение по одной координате и соответственно в этот момент можем работать с меньшим деревом отрезков, соответствующим этой координате).

Доказательство корректности.

Допустим у нас есть матрица $matrix$ размером $n \times m$ элементов, нужно корректно построить дерево отрезков, отвечать на запросы типа $getMax(x1,\ x2,\ y1,\ y2)$, где $x1,\ x2,\ y1,\ y2$ – координаты подпрямоуголька в матрице, на котором нужно найти максимум, а также $update(i,\ j,\ d)$, где $i,\ j$ – координаты элемента матрицы, а d – новое значение, на которое нужно заменить предыдущее значение элемента.

Тогда для того, чтобы построить двумерное дерево отрезков посчитаем и запомним максимальное значение на подпрямоугольнике $matrix[0 \dots n-1][0 \dots m-1]$, на подпрямоугольниках в два раза меньше: $matrix[0 \dots n/2][0 \dots m/2]$, $matrix[0 \dots n/2][m/2+1 \dots m-1]$, $matrix[n/2+1 \dots n-1][0 \dots m/2]$, $matrix[n/2+1 \dots n-1][m/2+1 \dots m-1]$, на подпрямоугольниках в четыре раза меньше, … и так далее. На каждом шагу количество таких подпрямоугольников будет увеличиваться в четыре раза, а их размеры уменьшаться в два раза. Делить их будем до тех пор, пока у нас не будет $n \times m$ подпрямоугольников размера 1. Таким образом, мы сможем найти максимум абсолютно на любом подпрямоугольнике (максимум на подпрямоугольнике будет равен максимуму из максимумов всех подпрямоугольников, находящихся в этом подпрямоугольнике). С построением дерева и запросом максимума разобрались, осталось разобраться с обновлением значения.

Обновлять значения элемента нужно не только для самого элемента в дереве отрезков, а также и значения максимумов для всех посчитанных ранее подпрямоугольников, в которые данный элемент входит. Обновление элемента по своей логике очень похоже на построение дерева отрезков, мы аналогичным образом делаем пересчет максимумов (но не для всех элементов, а лишь для тех, куда входит обновляемый элемент). Таким образом сможем корректно обновить максимумы, а соответственно и в дальнейшем получить правильный ответ на запрос максимума на очередном подпрямоугольнике.

Оценим время работы описанного алгоритма.

1. Построение дерева отрезков происходит за $O(N \times M)$ по времени. Данная операция будет произведена лишь один раз в самом начале выполнения программы (предподсчет).
2. Запрос на получения максимума будет выполняться за $O(\log N \times \log M)$, поскольку она сначала спускается по большому дереву и для каждой пройденной в нем вершины выполняет запрос, как у обычного (меньшего) дерева отрезков.
3. Запрос на обновление значения будет также работать за $O(\log N \times \log M)$, поскольку, как уже говорилось ранее, обновление будет затрагивать лишь те вершины дерева, которые хранят значения на подпрямоугольниках, в которые входит данный обновляемый элемент, а таких вершин будет $\log N \times \log M$.

Таким образом, все решение будет работать за $O(N \times M) + O(Q \times \log N \times \log M) = O(NM + Q \log N \log M)$, где Q – количество запросов на максимум/обновление значения матрицы.

Оценим требуемую память данного алгоритма. Требуется $O(N \times M)$ для хранения исходного массива с элементами матрицы, а также $O(16 \times N \times M)$ для хранения двумерного дерева отрезков. Таким образом всего требуется $O(17 \times N \times M)$ памяти. Ограничения на входные данные в задаче позволяют хранить практически все значения (кроме количества запросов) в int_16t и тогда данное решение укладывается в ограничения по памяти.