1 UVA Problem 12192: Grapevine

In Quadradonia, all rural properties are square, all have the same area, all are perfectly flat and all have the sides aligned to the North-South and West-East axes. Since properties are flat, the hills in Quadradonia look like a series of huge stairs' steps, with different heights. In a certain mountain, an interesting situation occurs in a rectangular area of N ×M properties. Starting from anywhere within the region, traversing it in the West to East direction, the properties have non-descending heights. Similarly, traversing that region in the North to South direction, starting from anywhere, the properties have also non-descending heights.

A large wine company in Quadradonia wants to rent some properties from that region to grow wine grapes. The company is interested in some special varieties of wine grapes, which are productive only if grown in properties whose heights are within a certain interval. That is, the company is interested in renting properties whose heights are equal to or higher than a given altitude L, and equal to or lower than a given altitude U . To make it easier for harvesting, the rented properties must form a contiguous area. And since everyone in Quadradonia likes squares, the area to be rented must have the shape of a square.

The company has not yet decided which variety of grapes it will grow, and therefore it has a list of queries involving intervals, one for each grape variety. The figure below shows an area of interest of dimensions 4×5 (in number of properties) with examples of areas the company could rent to grow grapes in heights within the intervals given in the picture.

13	21	25	33	34
16	21	33	35	35
16	33	33	45	50
23	51	66	83	93

You must write a program that, given the description of the rectangular area of interest in the mountain, and a list of queries containing height intervals, determines, for each query, the largest side, in number of properties, of a contiguous square area with heights within the specified interval.

1.1 Mathematical Formulation

Given an N x M board, where each n_i ¡ n_{i+1} and m_j ¡ m_{j+1} so that the smallest value is in index (0, 0) and the largest is in (1, 1). We will determine the maximum square area that is within the user specified bounds (L, U).

1.2 Solution

In our main functionality we will be reading in the data and storing it in a 1-D representation of a 2-D topological map. Each index will represent a square lot in this **int[N*M]**, int array of size N*M. We use regular int values to store the Lower and Upper bounds (L,U) and number of cases (Q). For each case, we will go through each row executing a **modified binary search** to search for the index of the lot which is the lowest of the in-bound lots (left-most lowerbound). This is explained in Algorithm II. Once this value is retrieved we use the fact that each n_i ¡ n_{i+1} and m_j ¡ m_{j+1} meaning that we **only need to check the diagonal** from this lowerBound to ensure that this square is a valid one. We will record the maximum size and continue checking. We will terminate either on the last row or the last possible row. i.e. if we have found 3 to be the max size, once we reach the second to last row, we cannot find a better than 3 square so we are done.

Algorithm 1 GrapeVine main

```
procedure MAIN( )
    reader ← BufferedReader
    stringBuilder ← StringBuilder
    line ← reader.READLINE( )
    while there is a new line do // get the dimentions of the board
        N, M ← line.SPLIT(bySpace)
        Ensure N, M are in bounds
        map ← int[N][M] // this is actually 1-Dimensional
        for i ∈ length(N) do
            splitLine ← reader.READLINE( ).SPLITLINE(bySpace)
            for j ∈ length(M) do
                map[i][j] ← Integer.PARSEINT(splitLine[j])
        Q ← Integer.PARSEINT(reader.readLine())
        stringBuilder ← new StringBuilder()
        while Q ≠ 0 do
            L, U ← Lower and Upper bounds
            max ← 0
            for i ∈ N && (N − i + 1)>max do
                lowerIndex ← lowerBound(map, i, (M-1), L) // see below
                if lowerIndex == 0 then
                    continue
                for j = max..<M do
                    n ← i + j
                    m ← lowerBound + j
                    if n ≥ N || m ≥ M || map[n][m]>U then
                        break
                    if j + 1>max then
                        max ← j + 1
            stringBuilder.APPEND(max)
            Q–
        stringBuilder.APPEND(-)
        line ← reader.READLINE( )
```

The point of the following algorithm is to binary search a row within the map and returns the lowest possible valid lot.

Algorithm 2 GrapeVine get lower bound

```
procedure lowerBound(int[] map, int row, int size, int L)
    lower ← 0, higher ← size, answer ← -1, mid
    while lower ≤ higher do
        mid ← lower + (higher - lower) / 2
        if map[row][mid] ≥ L then
            answer ← mid
            higher ← mid - 1
        else
            lower ← mid + 1
```

1.3 Correctness

Proposition 1.

The lowerBound method will give us the index m_j of the lot which has the lowest elevation within the bounds (U, L) within the specified row n_i.

Proof.

We begin this by stating that the length of any given row is $k + 1$ and, since this method call is independent between rows we can say that we are only looking at the elements $m_0, m_1, ..., m_k$ in row n_i. We do a binary search then, our checking statement will be to see if the midpoint is within the bounds (U,L) s.t. first time through we check $m_{k/2} \in (U, L)$. If in bounds, our new upper bound will be $m_{k/2-1}$, otherwise our new lower bound will be $m_{k/2-1}$. This process is done until we have gone through the complete search (lg(k)). □

Proposition 2.

In any square area, the top left corner lot will have the lowest elevation and the bottom right will have the largest area.

Proof.

This is more or less given in the problem description but to put it into a mathematical formulation we say that given the square area

$$\begin{bmatrix} x_{i,j} & x_{(i+1),(j+1)} & \cdots & x_{(i),(j+k)} \\ x_{(i+1),j} & x_{(i+1),(j+1)} & \cdots & x_{(i+1),(j+k)} \\ \vdots & \vdots & \ddots & \vdots \\ x_{(i+k),(j)} & x_{(i+k),(j+1)} & \cdots & x_{(i+k),(j+k)} \end{bmatrix}$$

we are gauranteed that (1) $x_{i,j}<x_{i,(j+1)}<...<x_{i,(j+k)}$ and (2) $x_{i,j}<x_{(i+1),j}<...<x_{(i+k),j}$. This translates accross rows so that in (1) if we replace i with $(i + 1)...(i + k)$ the

same relationship will hold and similarly, replacing j with $(j+1)...(j+k)$ in (2) will also hold.

$\therefore$ The lowest value will be $x_{i,j}$ and the highest will be $x_{(i+k),(j+k)}$.

□

Proposition 3.

Given the left most valid lot $x_{i,j}$ on a given row i, when we diagonally search through the topological map we will find the maximum size square possible originating from that row (not unique). i.e. look at $x_{(i+1),(j+1)}, x_{(i+2),(j+2)}, ..., x_{(i+p),(j+p)}$ where $0 \leq p \leq k = min(N, M)$.

Proof.

Given $x_{i,j}$, we know from proposition 2 that $x_{i,j}<x_{i,(j+1)}$, $x_{i,j}<x_{(i+1),j}$, and all three $< x_{(i+1),(j+1)} \implies$ as we go diagonally we are increasing in area covered s.t. the bottom right corner, $x_{(i+p),(j+p)}$ for $i \leq p \leq k$ is the biggest and $x_{i,j}$ is the smallest.Therefore the square matrix produced by this area is within the bounds (L,U) if both $x_{i,j}$ *and* $x_{(i+p),(j+p)} \in (L, U)$. Lets say the diagonal check will return that the $x_{(i+(p+1)),(j+(p+1))}$ fails $\implies$ that all lots $x_{(i+(p+2)),(j+(p+1))}, x_{(i+(p+3)),(j+(p+1))}$, ... ,$x_{(i+(p+k),(j+(p+1))}$ and $x_{(i+(p+1)),(j+(p+2))}, x_{(i+(p+1)),(j+(p+3))}, ..., x_{(i+(p+1)),(j+(p+k))}$ will fail as well as all $x_{(i+y),(j+z)}$ are also invalid $\forall$ (p+1) $< y, z \leq$ k where $k \in min(N, M)$. This means that all lots in the matrix below will be invalid.

$$\begin{bmatrix} x_{(i+(p+1)),(j+(p+1))} & x_{(i+(p+1)),(j+(p+2))} & \cdots & x_{(i+(p+1)),(j+(p+k))} \\ x_{(i+(p+2)),(j+(p+1))} & x_{(i+(p+2)),(j+(p+2))} & \cdots & x_{(i+(p+2)),(j+(p+k))} \\ \vdots & \vdots & \ddots & \vdots \\ x_{(i+(p+k)),(j+(p+1))} & x_{(i+(p+k)),(j+(p+2))} & \cdots & x_{(i+(p+k)),(j+(p+k))} \end{bmatrix}$$

Assume the valid lot matrix starting at $x_{i,(j+1)}$ as its top left corner is larger than the lot matrix we have just found. $\implies$ this lot must contain at least $x_{(i+(p+1)),(j+(p+2))}$ which would make the matrix size (p+1), one bigger than the lots previously found. However we just showed that $x_{(i+(p+1)),(j+(p+2))}$ which is a contradiction. we can replace this with any $x_{i,(j+m)}$) where $1 \leq m \leq k$ and the same contradiction will arrise. $\therefore$ Given the left most valid lot, this is the source from which we can find one of the biggest lots in any given map. □

Proposition 4.

Our algorithm will find the size of the largest square region. Note that this does not return the lot numbers, just the size $\therefore$ if there are several lots of the same largest size, the algorithm will return the size of them.

Proof.

From proposition 3, we know that we can find the largest size of valid lots for any given row. $\implies$ By saving the max size seen thusfar and stopping our search when it is impossible to get a size bigger than the current max, we will have the largest possible area at the end. □

1.4 Analysis

For the following analysis, we will say that the topological map is of size NxM and each entry can be represented using the notation $x_{i,j}$ where $i \in [0, N], j \in [0, M]$. As i and j increase, so does the value of each subsequent $x_{k,l}$ for $i, j<k, l$ respectively.

Proposition 5. *The space complexity of this algorithm is* ***O(N·M)***

Proof. This is due to the fact that all of our data can be infered from simply looking at the topological map which is a One-Dimensional array of size N·M, all other information is stored in integers so these are constant.

Giving us a space complexity of **O(N·M)**

□

Proposition 6. *The time complexity of this algorithm is* ***O(N·min(N,M))***

Proof. Let us address first the min(N,M) portion: the reasoning behind this is that as we do our diagonalization, our worst case is that we must go to the other side of the matrix i.e. $x_{i,j}$ to $x_{k,k}$ where $k = min(M, N)$ which is k comparisons. This this would be the bottom right most corner of the square matrix contained in the map (without the bounds).
Now we say for each row (**N** of them):

- **O(log(M))** Binary searching to find the leftmost lot within the bounds (L,U)
- **O(min(N,M))** Diagonally searching from the found lot t̂o the first out of bounds lot

Putting these together we get O(N·log(M) + N·min(N,M))

Giving us a time complexity of **O(N·min(N,M))**

□

1.5 An Example

See **grapeVine_example.txt** for a tracing with the input: 4 5
13 21 25 33 34
16 21 33 35 35
16 33 33 45 50
23 51 66 83 93
3
22 90
33 35
20 100