DP Speedups II: The Monge Property

Last Revised – November 16, 2006

Outline

- Totally monotone and Monge Matrices
- The SMAWK Theorem for finding row minima/maxima of totally monotone and Monge matrices quickly.
- A DP application:
  Web proxies on a line topology network.
- Proof of the SMAWK Theorem

Problem Definition

Our problem is to find the row-maxima in an $n \times m \; (n \leq m)$ matrix M:

10	12	9	10	10	6
10	13	10	11	11	8
7	10	8	9	9	7
8	11	10	11	11	10
4	8	8	9	10	11
1	5	5	6	7	9

For later use let $\pi(i)$ be column index of row maxima (In case of ties we let $\pi(i)$ be smallest such index).

This looks as if it should take $\theta(mn)$ time.

In practice, in many applications the matrix is often totally-monotone. In this case, the SMAWK algorithm due to A.Aggarwal, M.M.Klawe, S.Moran, P.Shor and R.Wilber permits us to solve the problem in $O(m+n)$ time. This will permit speeding up many dynamic programming aplications.

Monge and Monotone Matrices

10	12	9	10	10	6
10	13	10	11	11	8
7	10	8	9	9	7
8	11	10	11	11	10
4	8	8	9	10	11
1	5	5	6	7	9

A 2×2 matrix $\begin{pmatrix} a & b \\ c & d \end{pmatrix}$ is *monotone* if it is not simultaneously true that $a < b$ and $c \geq d$, i.e., row maxima can **not** be c, b.

An $n \times m$ matrix is *totally monotone* if every 2×2 submatrix is monotone.

An $n \times m$ matrix is *Monge* if $\forall i < n$, $\forall j < m$,

$$a_{i,j} + a_{i+1,j+1} \geq a_{i,j+1} + a_{i+1,j}$$

Lemma: Every Monge matrix is totally monotone.

The SMAWK Theorem

10	12	9	10	10	6
10	13	10	11	11	8
7	10	8	9	9	7
8	11	10	11	11	10
4	8	8	9	10	11
1	5	5	6	7	9

Theorem: (SMAWK)
If M is a totally monotone $n \times m$ $(n \leq m)$ matrix then its row-maxima can be found in $O(m + n)$ time.

We will now see an application of this theorem and, afterwards, a proof.

Note: The original proof of this theorem appears in the SMAWK paper. Our proof is a modified version of that scribed by by Bruce Maggs and Avrim Blum for the course notes of Computational Geometry (18.409) given in Spring 1988 by Alok Aggawaal and Joel Klein at MIT.

Note. The SMAWK Theorem implies that if M is a $n \times m\,(n \leq m)$ **Monge** matrix, i.e., $\forall i < n$, $\forall j < m$, it satisfies

$$a_{i,j} + a_{i+1,j+1} \geq a_{i,j+1} + a_{i+1,j},$$

then the row-maxima of M can be found in $O(m+n)$ time.

By taking negatives of all elements this implies that if $\forall i < n$, $\forall j < m$,

$$a_{i,j} + a_{i+1,j+1} \leq a_{i,j+1} + a_{i+1,j}$$

then the **row-minima** of M can be found in $O(m+n)$ time.

When dealing with row-minima problems, we will call matrices that satisfy the second inequality Monge as well.

The Web Proxy on a Line Problem

Consider a network as a weighted graph $G = (V, E)$, with the distance between two vertices being the shortest path between them. Let us specify one server $s \in V$ on the network as being the **source** of all information. Nodes will also have **weights**; the weight of node $v \in V$ will be w_v, the total amount of service (or frequency of requests) that v is requesting from s.

The total traffic associated with the request will then be $d(v, s)w_v$ (sometimes known as latency). Total latency of the system will be $\sum_{v \in V} d(v, s)w_v$.

In order to reduce the latency of the system we can make copies of the server's information and cache them at web-proxy nodes in the network. If S is the set of web proxies (we always assume $s \in S$) let $d(v, S) = \min\{d(v, u) : u \in S\}$. The total latency of the system is then reduced to $\sum_{v \in V} d(v, S)$.

The problem that we want to solve is, given the constraint that $|S| = m+1$, find the set of $m+1$ of web-proxies S that minimizes $\sum_{v \in V} d(v, S)$ *(recall that $s \in V$)*.

In graph theory this is known as the m-median problem and, for general graphs, is NP-hard to solve or even approximate.

For some special graph topologies, though, this problem can be solved in polynomial time. In particular, consider the directed line topology. G is a directed line with edges (v_i, v_{i-1}) for $i = 1, \ldots, n$ and original server $s = v_0$. The problem then has a simple Dynamic Programming formulation with a $\Theta(mn^2)$ solution. We will now see how to use monotone-matrix properties (the SMAWK algorithm) to reduce this down to $\Theta(mn)$.

The algorithm in these notes is from:
G. Woeginger, "Monge strikes again: optimal placement of web proxies in the internet," *Operations Research Letters,* 27(3), pp. 93-96 (2000).

The DP setup

Set $d_q = d(v_q, v_{q-1})$. Note that if $i < j$, then $d(v_i, v_j) = \sum_{q=i+1}^{j} d_q$. Set

$$\tilde{a}(i,j) = \sum_{\ell=i+1}^{j-1} w_\ell d(v_\ell, v_i) = \sum_{\ell=i+1}^{j-1} \sum_{q=i+1}^{\ell} w_\ell d_q$$

Let $F[j,k]$ denote the *minimum latency on the vertices $v_0, v_1, \ldots, v_j$ when using k proxies where the rightmost proxy is on node v_j*. Then

$$\begin{array}{lrcl} \forall j, & F[j,1] & = & \tilde{a}(0,j) \\ \forall k > 0 & F[1,k] & = & 0 \end{array}$$

and for $2 \le j \le n$ and $2 \le k \le m$,

$$\begin{aligned} F[j,k] &= \min\{F[i,k-1] + \tilde{a}(i,j) \,:\, 1 \le i \le j\} \\ OPT &= \min\{F[i,m] + \tilde{a}(i,n) \,:\, 1 \le i \le n\} \end{aligned}$$

where OPT is the final solution we are looking for.

Recall $\tilde{a}(i,j) = \sum_{\ell=i+1}^{j-1} \sum_{q=i+1}^{\ell} w_\ell d_q.$

Lemma: Set

$$D[j] = \sum_{q=0}^{j} d_q, \quad W[j] = \sum_{\ell=j}^{n} w_\ell,$$

$$X[j] = \sum_{q=0}^{j} \sum_{\ell=q}^{n} w_\ell d_q, \quad Y[j] = \sum_{\ell=j}^{n} \sum_{q=0}^{\ell} w_\ell d_q$$

Then

$$\tilde{a}(i,j) = Y[0] - Y[j] - X[i] + D[i] \cdot W[j].$$

Corollary: Using only $\Theta(n)$ preprocessing time and $\Theta(n)$ space, $\tilde{a}(i,j)$ can be calculated in $O(1)$ time.

Lemma: If $i+1 \le j$ then

$$\tilde{a}(i,j) + \tilde{a}(i+1,j+1) \le \tilde{a}(i,j+1) + \tilde{a}(i+1,j).$$

Furthermore, let M be the $n \times n$ matrix with entries

$$a_{i,j} = \begin{cases} \tilde{a}(i,j) & \text{if } i \le j \\ \infty & \text{otherwise} \end{cases}$$

Then M is Monge.

Proof: Recall that

$$\tilde{a}(i,j) = Y[0] - Y[j] - X[i] + D[i] \cdot W[j].$$

Then, if $i+1 \le j$

$$\begin{aligned} \tilde{a}(i,j) &+ \tilde{a}(i+1,j+1) - \tilde{a}(i,j+1) - \tilde{a}(i+1,j) \\ &= D[i] \cdot W[j] + D[i+1] \cdot W[j+1] \\ &\quad - D[i] \cdot W[j+1] - D[i+1] \cdot W[j] \\ &= (D[i] - D[i+1]) \cdot (W[j] - W[j+1]) \\ &= -d_{i+1} \cdot w_{j+1} < 0 \end{aligned}$$

To see that M is Monge simply note that if $i+1 > j$ then the right hand side of the corresponding inequality in $a_{i,j}$ is ∞ and is therefore obviously correct.

Lemma: Let $M = (a_{i,j})$ be an $n \times m$ Monge matrix, i.e.,

$$a_{i,j} + a_{i+1,j+1} \leq a_{i,j+1} + a_{i+1,j}.$$

Let $F_i, i = 1, \ldots, n,$ be any constants. Set

$$b_{i,j} = F_i + a(i,j)$$

Then $b_{i,j}$, viewed as a matrix, is also Monge.

Proof:

$$\begin{aligned} b_{i,j} + b_{i+1,j+1} &= F_i + a_{i,j} \quad + \quad F_{i+1} + a_{i+1,j+1} \\ &\leq F_i + a_{i,j+1} \quad + \quad F_{i+1} a_{i+1,j} \\ &= b_{i,j+1} + b_{i+1,j}. \end{aligned}$$

Our DP is

$$
\begin{array}{lrcl}
\forall j, & F[j,1] & = & \tilde{a}(0,j) \\
\forall k > 0 & F[1,k] & = & 0
\end{array}
$$

and for $2 \le j \le n$ and $2 \le k \le m$,

$$
\begin{array}{rcl}
F[j,k] & = & \min\{F[i,k-1] + \tilde{a}(i,j) \;:\; 1 \le i \le j\} \\
OPT & = & \min\{F[i,m] + \tilde{a}(i,n) \;:\; 1 \le i \le n\}
\end{array}
$$

Rewrite this as

$$
\begin{array}{lrcl}
\forall j, & F[j,1] & = & \tilde{a}(0,j) \\
\forall k > 0 & F[1,k] & = & 0
\end{array}
$$

and for $2 \le j \le n$ and $2 \le k \le m+1$,

$$
\begin{array}{rcl}
b^k_{i,j} & = & F[i,k-1] + \tilde{a}(i,j) \\
F[j,k] & = & \min\{b^k_{i,j} \;:\; 1 \le i \le j\} \\
OPT & = & F[n,m+1]
\end{array}
$$

Our DP is now

$$
\begin{array}{lrcl}
\forall j, & F[j,1] & = & \tilde{a}(0,j) \\
\forall k > 0 & F[1,k] & = & 0
\end{array}
$$

and for $2 \le j \le n$ and $2 \le k \le m+1$,

$$
\begin{array}{rcl}
b^k_{i,j} & = & F[i,k-1] + \tilde{a}(i,j) \\
F[j,k] & = & \min\{b^k_{i,j} \,:\, 1 \le i \le j\} \\
OPT & = & F[n,m+1]
\end{array}
$$

Note that this can be read as saying that we have $m-1$ matrices $M^{(2)}, M^{(3)}, \ldots, M^{(m+1)}$, where the entries of $M^{(k)}$ are $b^k_{i,j}$.

$F[j,k]$ are the row-minima of $M^{(k)}$ and OPT is the n'th row-minima of $M^{(m+1)}$.

If the $M^{(k)}$ were all Monge we could use the SMAWK algorithm to find each of their row-minima in $O(n)$ time, and all of them in $O(mn)$ time. We will now show that they are all Monge, immediately giving an $O(mn)$ time algorithm for calculating OPT.

Our DP is now

$$
\begin{array}{llcl}
\forall j, & F[j,1] & = & \tilde{a}(0,j) \\
\forall k > 0 & F[1,k] & = & 0
\end{array}
$$

and for $2 \le j \le n$ and $2 \le k \le m+1$,

$$
\begin{array}{rcl}
b_{i,j}^{k} & = & F[i,k-1] + \tilde{a}(i,j) \\
F[j,k] & = & \min\{b_{i,j}^{k} \,:\, 1 \le i \le j\} \\
OPT & = & F[n,m+1]
\end{array}
$$

We have $m-1$ matrices $M^{(2)}, M^{(3)}, \ldots, M^{(m+1)}$, where the entries of $M^{(k)}$ are $b_{i,j}^{k}$.

We have already seen that

(I) $\tilde{a}(i,j)$ is Monge.

(II) If $a(i,j)$ is Monge then, for any F_i,
$b_{i,j} = F_i + a(i,j)$ is Monge.

So, by induction, all of the $M^{(k)}$ are Monge, and we are done.

The SMAWK algorithm

To find the row maxima of an $n \times m \, (n \leq m)$ totally monotone matrix M:

Step 1: The algorithm reduces M to an $n \times n$ totally-monotone matrix M' using subroutine $REDUCE(n,m)$. The row maxima of M' will be the same as the row maxima of corresponding rows in M.
$REDUCE(n,m)$ will use $2m - n$ comparisons.

Step 2: All the odd rows of M' are then extracted to form an $n/2 \times n$ matrix M''.

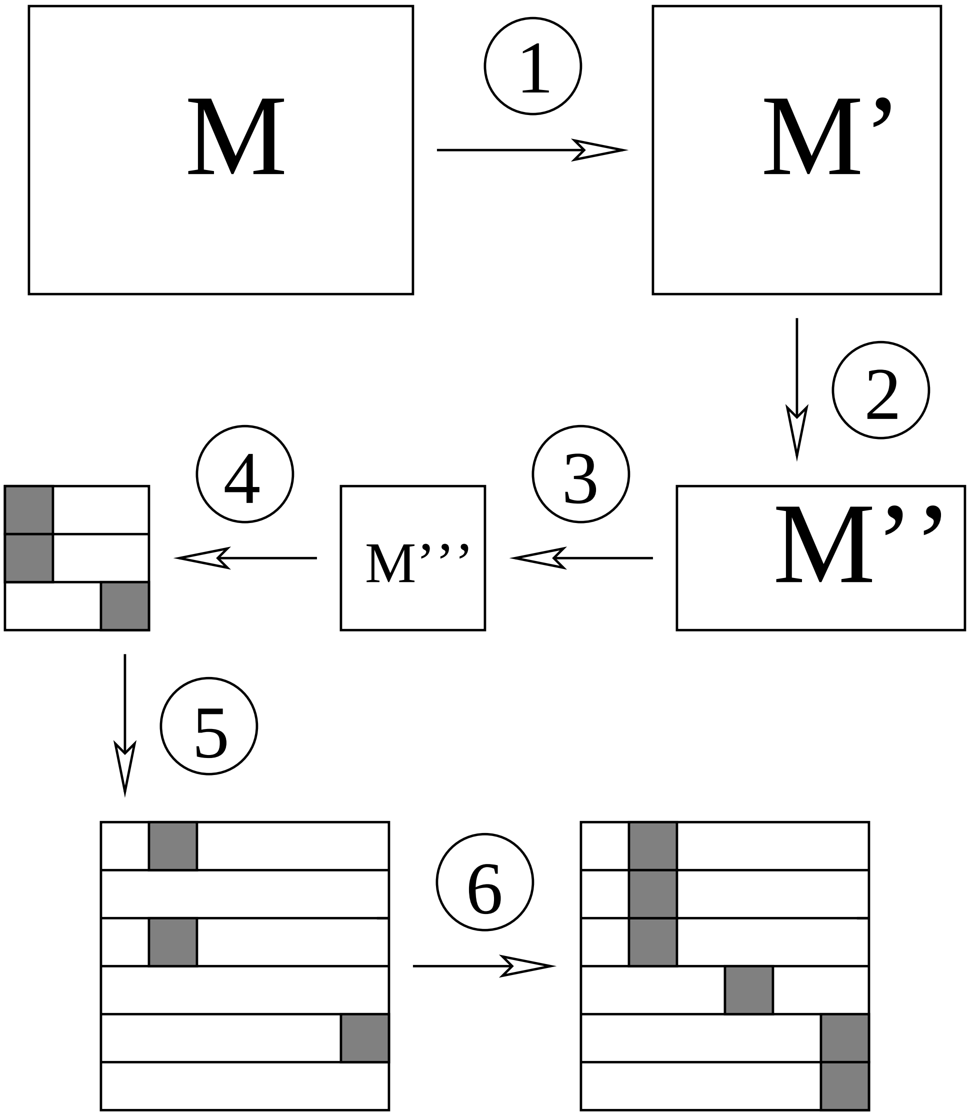
Step 3: $REDUCE(n/2, n)$ will then be applied to M'' to reduce it to an $n/2 \times n/2$ totally-monotone matrix M''', again without without changing the row maxima. This is done using $2n - n/2 = 3n/2$ comparisons.

The SMAWK algorithm: cont

Step 4: The algorithm is then recursively applied to M''' to determine the positions of the row maxima of M''' and thus of M''.

Step 5: All the rows and columns discarded in steps 2 and 3 are returned, to yield M'.

Step 6: Given all the positions of maximum of the odd rows, i.e., those of M'', the maxima of the even rows can be found in $3n/2$ comparisons.


M
1
M’
2
M’’
3
M’’’
4
5
6

Example

10	12	9	10	10	6
10	13	10	11	11	8
7	10	8	9	9	7
8	11	10	11	11	10
4	8	8	9	10	11
1	5	5	6	7	9

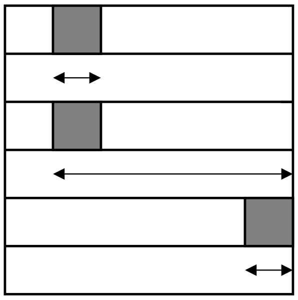
10	12	9	10	10	6
7	10	8	9	9	7
4	8	8	9	10	11

12	10	6
10	9	7
8	10	11

12	10	6
10	9	7
8	10	11

10	12	9	10	10	6
10	13	10	11	11	8
7	10	8	9	9	7
8	11	10	11	11	10
4	8	8	9	10	11
1	5	5	6	7	9

10	12	9	10	10	6
10	13	10	11	11	8
7	10	8	9	9	7
8	11	10	11	11	10
4	8	8	9	10	11
1	5	5	6	7	9



Staircase Lemma: $\pi(i) \leq \pi(i+1)$.

Proof: This follows directly from the definition of **totally monotonicity**.

This implies that, given the row maxima in the odd rows, the row maxima in the even rows can be found using *3n/2* comparisons (proving the correctness of Step 6).

Running time of SMAWK

Assume for now that $REDUCE(m, n)$ uses $\leq 2m - n$ comparisons. Let $T(m, n)$ denote the number of comparison required to find all the row maxima of an $m \times n$ totally-monotone matrix. Then,

$$\begin{aligned} T(n, n) &\leq 3n/2 + T(n/2, n/2) + 3n/2 \\ &\leq 6n \\ T(m, n) &\leq T(n, n) + 2m - n \\ &= 5n + 2m \end{aligned}$$

$\boxed{REDUCE(n, m)}$

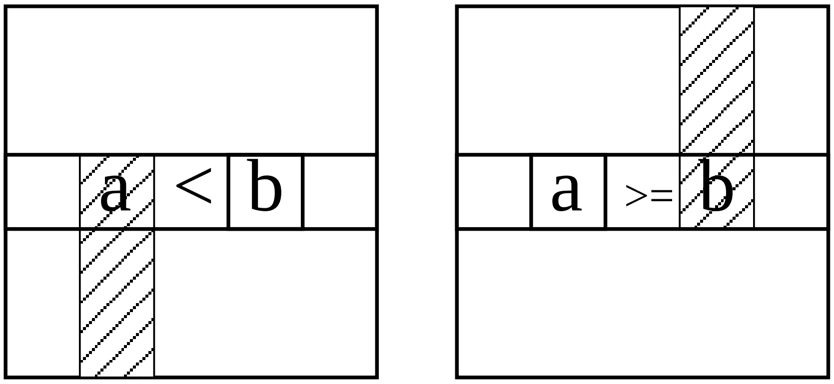
Lemma:

Let a and b be two entries in the same row of a totally-monotone matrix.

(i) If $a < b$, there will be no row-maximum below the entry of a.

(ii) If $a \geq b$, there will be no row-maximum above the entry of b.

Proof: If this is not true, then the totally-monotonicty property will be violated.



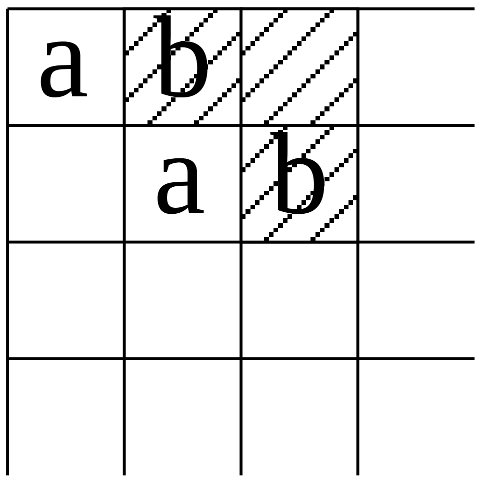
$\boxed{REDUCE(n, m)}$

Start with the first two elements a, b in the top row. Algorithm is:

If $a < b$, eliminate the entire column of a. Set the old b to be the new a and the element to its right to be the new b; repeat.

If $a \geq b$, eliminate b and all the entries above it. Set the element below the old b to be the new a and the element to its right to be the new b; go to next page.

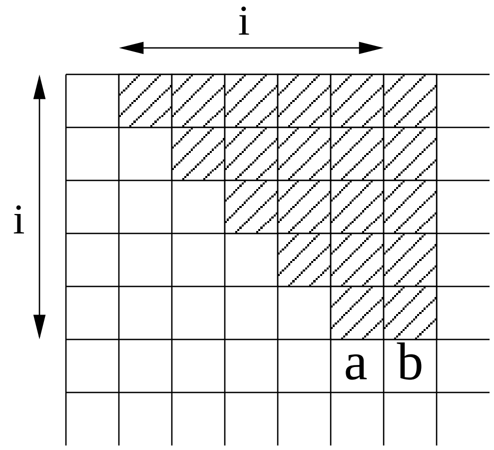
Note that, by previous slide, eliminated-items **can not** be row-maxima.



$\boxed{REDUCE(n,m)}$

In step k, assume we have an $i \times i$ staircase of eliminated entries ($i \leq n$). Compare a and b where

$$a = (i+1, i+1) \text{ and } b = (i+1, i+2).$$



There are two cases: (I) $a < b$ and (II) $a \geq b$.
We will see, that after both cases, we will have a staircase of i' eliminated entries such that
(I) a's complete column is eliminated and $i' = i - 1$ or
(IIa) b's complete column is eliminated and $i' = i$ or
(IIb) $i' = i + 1 \leq n$ and b and items above eliminated.

Since only $m - n$ columns are eliminated, (I) and (IIa) together occur exactly $m - n$ times. (IIb) can occur at most n times more than the number of columns that are eliminated. So, the total number of comparisons performed is $\leq (m-n) + (n + (m-n)) = 2m - n$.

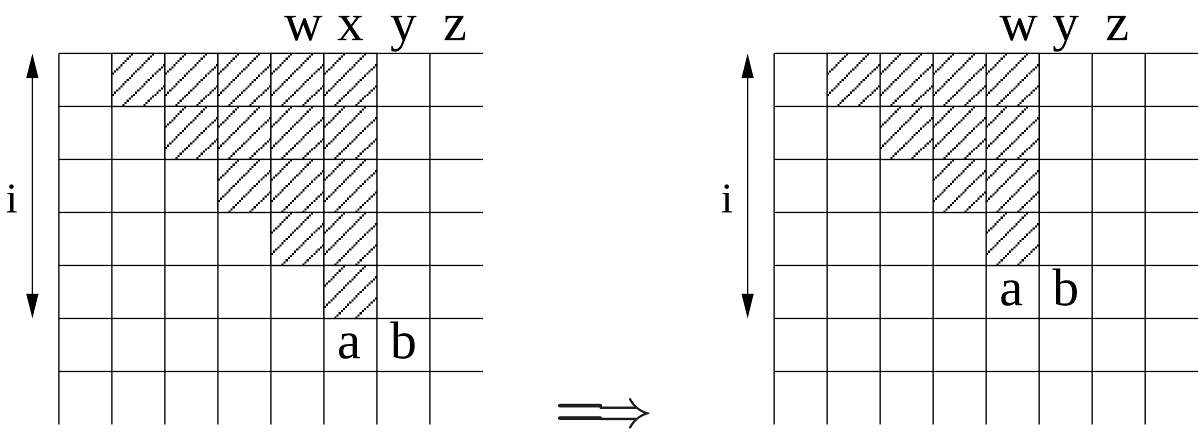
$\boxed{\text{(I) } a < b}$

In step k, assume we have an $i \times i$ staircase of eliminated entries ($i \leq k$). Compare a and b where

$$a = (i+1, i+1) \text{ and } b = (i+1, i+2).$$

There are two cases: (I) $a < b$ and (II) $a \geq b$.

(I) If $a < b$, eliminate a and all items below it. Since items above a previously eliminated, eliminate entire column $i+1$. This gives a $(i-1) \times (i-1)$ staircase of eliminated entries. New a is highest non-eliminated entry in column i.



(II) $a \geq b$

In step k, assume we have an $i \times i$ staircase of eliminated entries ($i \leq k$). Compare a and b where

$$a = (i+1, i+1) \text{ and } b = (i+1, i+2).$$

There are two cases: (I) $a < b$ and (II) $a \geq b$.

(II) If $a \geq b$, eliminate b and all the entries above it.

Either (a) $i = n$, in which case b is bottom item in column so we eliminate entire column $i+2$, set new a to be old a and the staircase remains $i \times i$,

or (b) $i < n$ and we will have increased the size of the staircase to $(i+1) \times (i+1)$ and set new a to be old b.