Solution for Pyramid Base

We start off with a few observations regarding the placement of the optimal square:

Lemma 1
It suffices to consider squares whose left side touches either the left side of the field, or the right boundary of some obstacle.
Proof
We can always shift the optimal square leftwards until it hits such a boundary, without increasing the cost of placing it.

As x and y dimensions are independent in terms of translation, we also have:
Lemma 2
It suffices to consider squares whose bottom side touches either the bottom side of the field, or the upper boundary of some obstacle.
Proof
Similar to lemma 1, except we shift the optimal square downwards instead.

This observation alone yields a $O(P^3 \log P)$ solution since we could now enumerate the possible locations of the bottom left corner of the square in $O(P^2)$ time, then for each such location try increasing the square size and keep track of the obstacles we hit. It is not difficult to optimize this approach into a $O(P^3)$ solution.

We now try to get a faster solution by turning the problem into a decision problem.
Suppose that we have a square we can afford. If we now shrink it, while keeping its lower left corner in place, we will get a smaller square we can afford. This means that if X is the size of the optimal square, then for all Y<=X there is a square of size Y we can afford, and there is no such square for any Y>X. As a result, we can binary search on the side length of the square and then solve the following problem: 'Given a field with a set of rectangle obstacles, each with a given cost, find the square of side length Y such that the sum of costs of rectangles it intersect is minimum.'

It is possible to solve this problem directly by sweeping and using some clever data structures. However, one simple observation can give us a solution that is pretty easy to implement. The observation is that all we actually need is the location of the bottom left corner of this square. Note that for each point P we can easily compute which rectangles intersect a square of size Y with the bottom left corner in P – these are precisely the same rectangles that contain the point P, if we extend them by Y downwards and by Y to the left.

In other words, to solve our decision problem, we can extend each rectangle by Y both leftwards and downwards, and then solve a simpler problem: 'Given a field with a set of rectangles, each with a given cost, find the point contained in the rectangles with minimum total weight.'

This problem can be solved by a range sweep going from left to right. As we encounter a rectangle's left x value, we increment the y range corresponding to it by its value, and when we pass a rectangle's right x value, we decrement the y range accordingly. So we need the following data structure:

Given an array of integers, support:
- Increasing/decreasing a section by a value.
- Query for the minimum value in the array.

Note that by lemma 2, it suffices to only consider y values that are right above a top edge of a rectangle, so we have O(P) of these entries in the array. There are 3 basic data structures that support these requirements:
- A plain array, where each operation takes O(P).
- A 2-level B-tree with each operation in O(sqrt(P)).
- A range tree with each operation in O(log P).

More details on the construction of a range tree can be found in the solution for Fish.
The solution with the range tree runs in O(P * log P * log min(M,N)), the last factor being the binary search for the square size. This solution was expected to receive 70 points, the one with the B-tree 55, and the one with the array 35 points.

The remaining 30 points were awarded for large cases with zero allowed cost. We will now outline one approach to solve these cases. This approach will originate from a different approach to the general problem.

Let C_{lim} be the largest cost we can afford. Define $f(x_1,x_2)$ as the maximum vertical size of a rectangle with cost at most C_{lim}

that has left edge on x_1 and right edge on x_2. Then we can prove the following about the function f:

Lemma 3
$f(x_1,x_2) >= f(x_1-1,x_2)$ and also $f(x_1,x_2) >= f(x_1,x_2+1)$.
Proof
Take any rectangle A with left edge at x_1-1 and right edge at x_2. If we throw away the leftmost column, we get an equally tall rectangle B with left edge at x_1 and right edge at x_2. Obviously, the cost of the new rectangle is at most the cost of the old rectangle. Thus whenever we can afford rectangle A, we can also afford rectangle B (and we may even be able to make rectangle B taller within our budget).

In terms of f, the goal of the problem can be rephrased as finding x_1 and x_2 such that $\min(x_2-x_l+1, f(x_1,x_2))$ is maximized. Another way of stating the same goal is that we try to maximize x_2-x_l+1 over all pairs (x_1,x_2) such that $f(x_1,x_2) >= x_2-x_l+1$.

To find this maximal value, for each x_1 we can find the largest x_2 such that $f(x_1,x_2) >= x_2-x_1+1$. We will use the notation $g(x_1)$ for the largest such x_2 to the given x_1.

Lemma 4
$g(x_1+1) >= g(x_1)$.
Proof
In words, if there is a valid square starting at x_1 and extending all the way to x_2, there is also such a square starting at x_1+1. Formally, let $x_2=g(x_1)$. From the definition of $g(x_1)$, we have $f(x_1,x_2) >= x_2-x_1+1$. From Lemma 3 it follows that $f(x_1+1,x_2) >= f(x_1,x_2)$. Combining these, we get $f(x_1+1,x_2) > x_2-(x_1+1)+1$. This means that $g(x_1+1)$ is at least x_2, which is exactly what we needed.

By Lemma 4 we know that we can calculate g(x) for all x values by looping left to right on them, and incrementing the g value for the current x. This can be visualized as a double sliding window, as we insert rectangles as the right side of the window hits them and remove them as the left side leaves them. Clearly, each rectangle is inserted once and deleted once. The problem once again reduces to a data structure one:

We need a data structure that will maintain a set of weighted intervals and support:

- Insertion.
- Deletion.
- Finding the longest interval whose total cost is at most C_{lim}.
  Note that this longest interval can be an arbitrary interval within the bounds of the field, and its cost is the sum of the costs of stored intervals it intersects.

In the general case where $C_{lim} > 0$ this strucutre is quite difficult to maintain and the host committee was not able to find a reasonable data structure that does it better than in O(P) time per operation.

When $C_{lim}=0$, the last requirement simplifies to finding the longest empty interval.
This is maintainable using a range tree, by tracking, for each segment S:

- The length of the empty segment that starts on the left end of S.
- The length of the empty segment that ends on the right end of S.
- The maximum length of an empty segment contained in S.

These values then propagate nicely up the tree and each operation takes O(log P) time. As in the previous solution, maintenance in O(sqrt(P)) time is also possible.

The cost for moving the left and right boundary during the sweep takes O(N) time if implemented naively. However, Lemma 1 along with the fact that the function f can only change when the right boundary of the sweep reaches the left boundary of some rectangle means that we can move the boundaries in jumps, and process O(P) interesting events only.

The motivation for creating this problem came from the maximum axes-parallel empty rectangle problem, which in its simplest form can be phrased as follows: Given a set of P point obstacles (having zero area, contrary to this problem) and a bounding rectangle, find the rectangle of maximum area that does not contain any of those points in its interior. Using the earlier lemmas, one could derive that all sides of the rectangle touch some of the points and obtain a $O(P^2)$ solution. Subquadratic solutions are possible, one possibility is to use repeated divide-and-conquer followed by 3-dimensional halfspace queries. None of the known solutions were suitable for the IOI, as already the O(P log P) implementations of convex hulls are far from trivial. As a final note, Agarwal and Suri gave a $O(P \log^2 P)$ solution in 1989.