# SACO 2004 Day 2 Solutions

## 1 Biggest Barn

A brute force solution would be to check each possible square to see if it contains any trees. This solution is O($N^5$) worst case and will get about 50%. It can be sped up by using binary search on the square size, searching for a square of a sepecific size at each step of the binary search.

To get 100% you need to use Dynamic Programming. Call $S_{i,j}$ the largest square we can place that fits inside the area $[(0,0)$, $(i,j)]$, i.e. the bottom-right corner of the square must be $\leq (i,j)$. It can be shown that:

$S_{i,j} = \max(S_{i-1,j}, S_{i,j-1}, S_{i-1,j-1}) + 1$

if there is no tree at location $(i,j)$, otherwise $S_{i,j} = 0$.

For $i < 0$ or $j < 0$, we set $S_{i,j} = 0$.

## 2 Prison Break

Since $N$ and the number of doorways per room is small, generating every path starting at $B$ is fast enough. We can generate every path recursively using depth first search. See `http://en.wikipedia.org/wiki/Depth_first_search` for an explanation of DFS. In our recursive method we pass the current path and the room we are currently in. Then for each adjacent room that is not already on the path we:

1. add it to the path
2. recurse with the new path and the adjacent room
3. remove it from the path

We keep track of the longest path found, and update it whenever we find a longer one. Note that in the path we need to store the door numbers.

## 3 Store Supplies

The simplest solution for this problem is to keep an array containing the quantities for each ID, and update each element of an order. In this solution, queries can be answered very quickly (in constant time, but updates can be very slow (they take time proportional to the size of the range). This solution gets 50%.

The next solution is to keep a list of orders (as an object or tuple containing the start and end of the range), and to search through all the orders for every query. Here, updates are much faster (constant time) but queries take time proportional to the number of orders. Since the maximum number of orders is much smaller than the maximum size of an order, this solution is much better. Because we have a much faster marker now than we did when this problem was set, this solution scores now 100%. However, there is a better solution (and this would be needed to get full marks in the Third Round).

The full solution is to use a data structure called an interval tree. An interval tree is a binary tree where each node represents an interval (in this case an interval of product IDs). Each node has a count and two children, representing the two halves of its interval (it doesn't matter if their lengths differ by one). The root represents the interval from 1 to 1 000 000, and the leaves represent the one-product intervals.

When Fred makes an order, you recurse into the tree. If the order covers the whole of the current interval, increment the count of the node and stop. If the order covers *none* of the current interval, stop. Otherwise, recurse into both children.

To answer queries, you also recurse into the tree, adding up the counts for each interval that covers the queried product. Thus, in this case, you either recurse on the left or right child, rather than both.

It is easy to see that queries take time proportional to the height of the tree, which is proportional to the logarithm of the width (which is about 20 here). It is a little harder to analyse the time-complexity of updates, but it turns out to be roughly the same as for queries. Thus, this solution is much faster than the previous solutions.