KLONOVI

The lengthy problem statement really says that we need to find the prime factors of each capacity, and use each of those factors in the cloning process so that the product of factors used in a machine is equal to its capacity.

Determining the order in which a prime factor is used is tricky. It is not good enough to arbitrarily choose which machines will use the factor. For example, in the second example from the problem statement, if we first use the factor 5 in machines 1 2 3 and then in machines 1 2 4, we will need two additional clone commands because we cannot use machine 5 twice in the last command.

The solution is to use a factor earlier in machines that need to use it more times.

BAGER

Because the bulldozer is limited to moving in three directions, when it moves from cell (r, c) right to cell (r, c+1), then we can assign all cells directly above (r, c+1) to village B. Similarly, when it moves down, we can assign all cells directly left of (r+1, c) to village A. By the time the bulldozer reaches the final cell, all cells will have been correctly assigned.

Let f(r, c) be the largest sum achievable if the bulldozer is currently in cell (r, c). The value can be calculated recursively as:

$$f(r,c) = \max\begin{cases} f(r,c+1) + bananas((1,c+1) \to (r-1,c+1)) \\ f(r+1,c) + apples((r+1,1) \to (r+1,c-1)) \\ f(r+1,c+1) + bananas((1,c+1) \to (r,c+1)) + apples((r+1,1) \to (r+1,c)) \end{cases}$$

A direct implementation of this recursive relation is too slow because of too many branches.

Notice that the result of the function does not depend on the path the bulldozer has taken so far. The direct implementation will make many calls with the same parameters (r, c), when the result will always be the same. Because of this, we can use dynamic programming to calculate the result, in one of two ways:

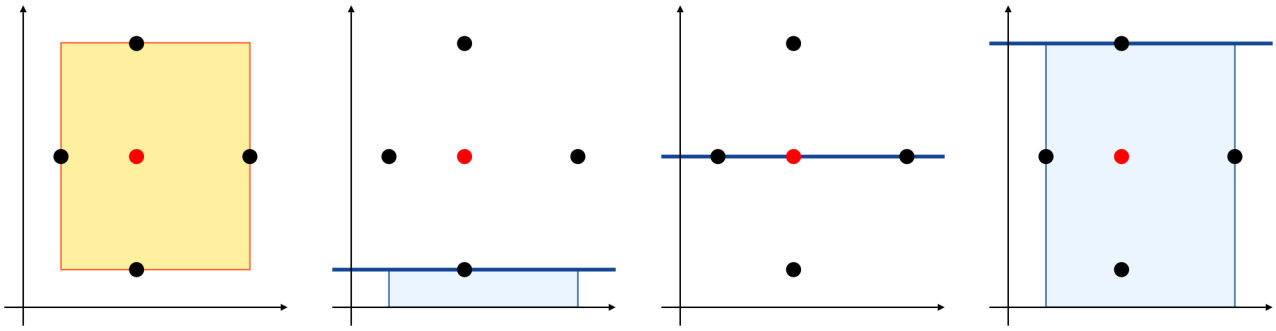
- Top-down: memoize the result for each pair of parameters (r, c) and return it on subsequent calls with the same parameters.
- Bottom-up: do not implement f(r, c) as a function, instead using a two-dimensional array and calculating in the correct order (from lower-right to upper-left) to ensure that values necessary for calculation are available.

The example source code in C++ demonstrates the first approach. Example code in Pascal demonstrates the other.

PCELICE

First, let us determine the bounding rectangles for all flowers. If we sort flowers by y-coordinate (breaking ties by x-coordinate), then for some flower its closest left and closest right neighbours are immediate neighbours in the sorted array (we need to additionally check if they have the same y-coordinates, in case the flower does not have a left or right neighbour). Similarly, sorting by x-coordinate gives us the up and down neighbours for each flower.

Imagine now moving a horizontal line from y=0 upwards. Assume we have a data structure available to answer queries of the form: "For given x_1 and x_2, how many flowers with x-coordinates between x_1 and x_2 are below our horizontal line?" This structure can be a Fenwick tree or a binary search tree. The following four images illustrate three important events for one highlighted flower:



0. The first image shows the highlighted (red) flower and its neighbours in all four directions. For this flower we need to calculate the flowers inside the orange rectangle.
1. First important event; the horizontal line hits the lower side of the orange rectangle. Subtract the number of flowers inside the blue rectangle (using our data structure).
2. Second important event; the horizontal line hits the highlighted flower. We need to insert it into the data structure, so that we can count it in rectangles of other flowers.
3. Third important event; the horizontal line hits the upper side of the orange rectangle. Add the number of flowers inside the blue rectangle (using our data structure).

The orange area is the difference between blue areas in the fourth and second image so the result obtained in the first and third steps is exactly the number of flowers in the orange rectangle.

The time complexity is O(N log N) for sorting and O(N log M) to process events.