z5242692

Chenqu Zhao

COMP9101 (T2-2020)

Homework 1 – Q4

Define a rectangular coordinate system in the map, mark the left top tree as (0,0) and the right bottom tree as (4n,4n). The horizontal direction is the X-axis and the vertical direction is the Y-axis, derivate the coordinates of the other trees following this rule. Assume that there is a slidable window in the orchard representing our potential choices of trees, which has the exact same size as n^2. The window starts from the left top n * n square.

We record numbers of apples on each tree according to its position row by row in array and store the array in 2-d array M. We compute the sums of apple at this initial state by brute force. Store this sum in array A. Store the coordinates of left top corner along with the coordinates of right bottom corner in array B. For example, the initial window is recorded as {(0,0), (n,n)}. The sum computation of apples costs $O(n^2)$.

Now we slide the window down tree by tree, instead of using brute force, we apply the prefix sum algorithm. For instance, when we slide the window down by one tree, the sum of apples now = A[last step] – sum of the first row of M[last step] + the sum of new row. Append corresponding data to array A and array B as above. This computation of sum of new row costs $O(n)$. Slide down to the bottom row of the orchard which takes 3n steps, this way the leftmost side n * 4n area is known. Reset the window to initial state, and then apply the similar operation to slide the window right to the rightmost column of the orchard, thus, the topmost 4n * n area is also known. The sliding window operations cost $O(n) \cdot 3n \cdot 2 = O(n^2)$.

Touch the unknown trees from (n+1,n+1), this tree is the right bottom corner of window {(1,1), (n+1,n+1)}. As we already know apple amounts of all trees in this window except (n+1,n+1), we can then compute its sum as well as all information we need for this window in $O(1)$. Again, append corresponding data to A and B as below. Similarly, solve the other adjacent unknown trees and windows from left top to right bottom (general direction). We solve all the unknown windows left in $O(1) \cdot 3n \cdot 3n = O(n^2)$.

Traverse array A to find the index of the largest value in A, then lookup the element of the same index in B which is the window we want. The traverse and lookup costs $3n \cdot 3n \cdot 2 = O(n^2)$.

In total, this algorithm costs $4 \cdot O(n^2) = O(n^2)$ time that is runs in $O(n^2)$.