IOI Training Camp 1, 2009 Day 2 Solutions

SACO Scientific Committee

1 Tree Tiles

1.1 50% Solution

The main solution to this problem requires DP. We can break the problem down into smaller problems of "How many tilings are there for a tree of depth d?". There is exactly one way to tile a tree of depth 0. We then iterate from depth 1 to D, working out the number of tilings for each depth.

The number of tilings for depth d with a specific tile at the root is the product of the number of tilings for all the newly placed tile's subtrees. The subtrees are the "children" of the leaves of the newly placed tile. The number of a subtree's tilings has already been calculated, since its height will always be less than d.

To get the number of tilings for a tree of depth d using any tiling, add up the number of tilings with each available tile at the root.

1.2 100% Solution

The number of possible tilings grows exponentially. Therefore, for larger values of D the problem requires big numbers.

One potential stumbling block is the requirement to output the answer modulo 10^80. An easy solution is to have one additional "digit" into which we discard all carries that would make the number exceed 10^80 and then just not output this "digit".

2 Ben's Mom

2.1 30% Solution

Generate every possible rectangle. You can do this by iterating through every pair of points (r_1, c_1) (r_2, c_2) with $r_1 \leq r_2$ and $c_1 \leq c_2$ (can be done with 4 for loops). If a rectangle does not contain a tree, you compare the area to the largest area found so far. This is an $O(R^3C^3)$.

We can speed up the valid rectangle check to $O(1)$. We do this by doing a DP where $dp(r, c)$ stores the number of trees in the rectangle $((1, 1), (r, c))$.

$$
\begin{aligned}
dp(r,0) &= 0 \\
dp(0,c) &= 0 \\
dp(r,c) &= dp(r-1,c) + dp(r,c-1) - dp(r-1,c-1) + (tree[r][c]?1:0)
\end{aligned}
$$

Then the number of trees in $((r_1,c_1),(r_2,c_2))$ is

$$ntrees(r_1,c_1,r_2,c_2) = dp[r_2][c_2] - dp[r_1-1][c_2] - dp[r_2][c_1-1] + dp[r_1-1][c_1-1]$$

You can then check if a rectangle is valid by $ntrees(r_1,c_1,r_2,c_2) == 0$. This will get you 50%.

2.2 100% DP+Stack Solution

There is an $O(R^2C)$ solution for this using a simple DP and a stack. The DP works per column, by finding the number of empty cells above a cell until the next tree. The dp is as follows:

$$
\begin{aligned}
dp(r,0) &= 0 \\
dp(r,c) &= 0 \text{ if tree[r][c]} \\
dp(r,c) &= dp(r-1,c) + 1 \text{ otherwise}
\end{aligned}
$$

You then do the following:

```
area = 0
for each row r:
  stack = {}
  stack.push((height=0, column=0))
  for each column c:
    height = dp(r, c)
    c1 = c
    while stack.top.height > height:
      c1 = stack.top.column
      stack.pop()
    if stack.top.height != height:
      stack.push((height=height, column=c1))
    for item in stack:
      a = (c - item.column + 1) * item.height
      area = max(area, a)
```

2.3 100% DP Solution

It is also possible to solve the problem in $O(RC)$ using three DP's:

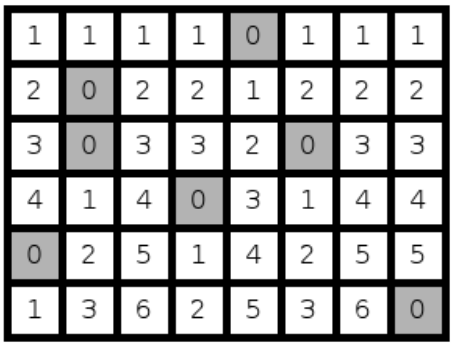

Figure 1: *The DP matrix for the sample input for the 100% DP+stack solution of largerec.*

- $h(r, c)$: if we start at (r, c) and go upwards, how many empty cells do we find before the first tree?
- $l(r, c)$: how far left can we extend a rectangle with bottom-right corner at (r, c) and height $h(r, c)$?
- $r(r, c)$: how far right can we extend a rectangle with bottom-left corner at (r, c) and height $h(r, c)$?

The three recurrence relations are:

$$\begin{aligned} h(0, c) &= 0 \\ h(r, c) &= 0 \text{ if tree[r][c]} \\ h(r, c) &= h(r-1, c) + 1 \text{ otherwise} \end{aligned}$$

$$\begin{aligned} l(r, 0) &= 0 \\ l(r, c) &= c - p \text{ if tree[r-1][c]} \\ l(r, c) &= \min(l(r-1, c), c - p) \text{ otherwise} \end{aligned}$$

$$\begin{aligned} r(r, C+1) &= 0 \\ r(r, c) &= p - c \text{ if tree[r-1][c]} \\ r(r, c) &= \min(r(r-1, c), p - c) \text{ otherwise} \end{aligned}$$

where p is the column of the previous tree as we populate l from left-right and r from right-left.

The answer is then:

$$\max_{r,c}(h(r, c) * (l(r, c) + r(r, c) - 1))$$

This works because of the observation that the largest rectangle will always touch a tree (considering the edge as being covered in trees) on all four sides. By considering all rectangles with at least top, left and right touching a tree, we cover all candidate rectangles.

3 Newspaper Delivery

3.1 100% Solution

This can be done in $O(V+E)$ time, where V is the number of intersections and E is the number of roads. The first step is to calculate the strongly connected components. This will create a new graph which is a directed acyclic graph (DAG). Each strongly connected component is represented as a vertex in the DAG. There is an edge from component C_i to C_j if there is a vertex v_i in C_i and a vertex v_j in C_j such that there is an edge $v_i \to v_j$ in the original graph.

Now for each component you work out the number of edges in the component. An edge is in a component C, if there exist two verticies v_i, v_j in C such that there is an edge $v_i \to v_j$.

We now do a DFS on the DAG to work out the maximum number of edges starting from each component. So

$$\text{dfs}(C) = \text{number of edges in } C + \max\{\ \text{dfs}(n) + 1 : n \text{ is a child vertex of } C\ \}$$

Note that for a component C, dfs(C) will always return the same value. So we can use memoization on it. Doing all this will get you 100%.