This problem is slightly different from other problems we have solved with dynamic programming, because it is not an optimization problem. But the same steps can be used to solve it.

0. **Recursive structure.**
   Every path from p_1 to p_n must go through either $p_{up[n]}$ or $p_{left[n]}$, just before reaching p_n. So the total number of paths from p_1 to p_n is equal to the number of paths from p_1 to $p_{up[n]}$ plus the number of paths from p_1 to $p_{left[n]}$.

1. **Array definition.**
   For $i = 0, 1, \ldots, n$, let $Num[i]$ store the number of distinct paths from p_1 to p_i. ($Num[0]$ is a degenerate case used to simplify the recurrence below).

2. **Recurrence relation.**
   Base cases: $Num[0] = 0$ and $Num[1] = 1$.

   General case: for $i = 2, 3, \ldots, n$, $Num[i] = Num[up[i]] + Num[left[i]]$.

   Remember that $up[i] = 0$ when there is no "down" edge into p_i and $left[i] = 0$ when there is no "right" edge into p_i; together with the value of $Num[0]$, this ensures that correct values are computed for each i, even when p_i is missing a left neighbour or an up neighbour, or both.

3. **Iterative algorithm.**
   From the recurrence above:

   $Num[0] \leftarrow 0$
   $Num[1] \leftarrow 1$
   for $i = 2, 3, \ldots, n$:
       $Num[i] \leftarrow Num[up[i]] + Num[left[i]]$
   return $Num[n]$

4. **Optimum solution.**
   Nothing to do here: previous step already returns desired value.