Canyon paths: Analysis

The core of the problem is finding a “shortest” path in the grid. However, there are two extra challenges: First, instead of adding the values of each cell visited on the path to get the total distance of the path, we take the minimum of all those values. Together with this, we want to maximize instead of minimize the resulting value. Second, we have the option to ignore a number of grid cells when calculating its length.

The intended solution works around these challenges by reversing the problem. Instead of asking “what is the highest path value possible”, we find the answer to the question “is there a path using at most K bridges whose minimal height is at least H”. It then uses binary search on H to find the required answer.

The question “is there a path using at most K bridges whose minimal height is at least H” is intended to be answered through a modified breath first search. Start a breath first search from the top edge, adding adjacent grid cells that are high enough ($\geq H$) to the work queue, and the remaining adjacent grid cells to a next queue. In the next iteration, initialize the work queue using the next queue from the previous iteration.

Doing this for $K+1$ iterations, one can calculate all grid cells that can be reached with a path of minimum height above H containing at most K bridges. If one of the cells reached is in the lowest row, there is a path spanning the canyon.

Using the above algorithm, each question “is there a path using at most K bridges whose minimal height is at least H” takes $O(RC)$ to answer. The binary search then takes $O(RC\log(H_{\max}))$.

Partial solutions

First, consider the case where $K = 0$. Here, the problem reduces to finding a shortest path with a somewhat strange metric. Dijkstra’s algorithm can be adapted to work for this.

A solution using Dijkstra’s algorithm can be extended to testcases with small values for K, using graph duplication. We construct $K+1$ copies of the graph, one for each of the possible number of bridges already used. Edges between the layers then have “infinite” height, and need to be directed. The “shortest” path in this duplicated graph then gives the required solution in $O(KRC\log(KRC))$.

Finally, a solution might do a binary search on the height of the lowest path, but instead of using depth first search to check for existence, use Dijkstra’s algorithm for that purpose. This gives a complexity of $O(RC\log(H_{\max})\log(RC))$, combined with a somewhat larger execution constant making it infeasible.

Suggested grading

Based on the above suggested solution and partial solutions, the following could be a reasonable distribution of points into subtasks:

- 20 points worth of testcases with $K = 0$. (Intended to accept Dijkstra-based solutions without graph duplication)
- 30 points worth of testcases with $R \leq 100$ and $C \leq 100$. (Intended to accept Dijkstra-based solutions with graph duplication
- 30 points worth of testcases with only $C \leq 100$. (Intended to accept solutions that fail to optimize to a linear time breath-first-search)
- 20 points worth of testcases with no restrictions.

Aditionally, for $R = 1$ the problem reduces to finding the maximum value of the grid cells, which is relatively trivial to solve. It could, depending on the total problem set, be an option to take out 5 to 10 points from the first and or second subtask to create a subtask with $R = 1$.