CCO ’20 J5/S2 – Escape Room (Editorial)

Subtasks 1 – 5:

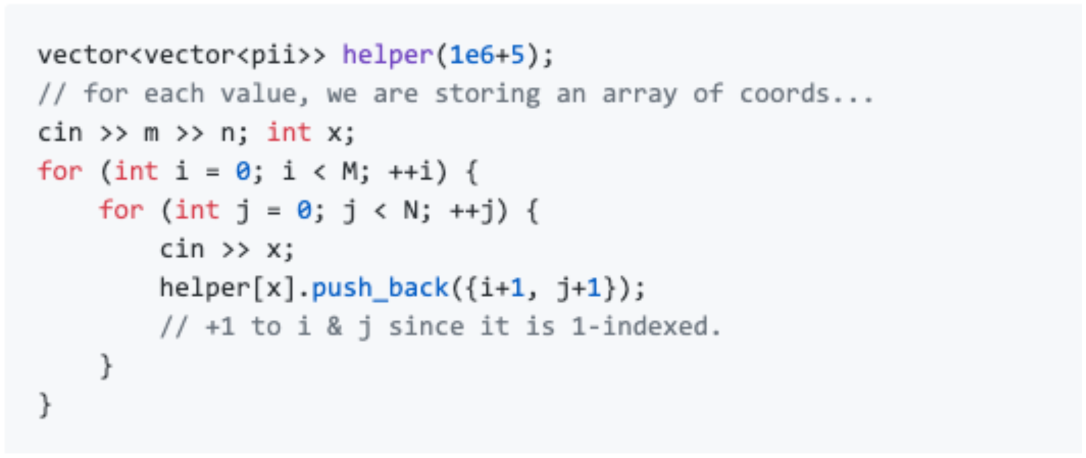
For partial points, one can utilize the Breadth First Search Algorithm and apply it to this problem. For any position (r, c), we can iterate over the array to find all cells (a, b) such that $a \cdot b = arr[r][c]$. “Breadth first search” all on all neighbours if a neighbour is ever (m, n) it means a path exists.

However, the described approach is must too slow, and TLE’s on subtask 6 and 7 yielding a result of 11/15.

Subtasks 6 – 7:

For the last two subtasks, the algorithm must be optimized. We notice that for every node, we are iterating over the whole array to find neighbours. If the same thing is done for thousands of nodes, it will result in TLE, and a lot of operations will be repeated. We can reverse this problem and start the bfs from (m, n). For any node (r, c) we will need find all values in the array such that it holds the value of $r \cdot c$.

Optimization can be done while reading input.



```
vector<vector<pii>> helper(1e6+5);
// for each value, we are storing an array of coords...
cin >> m >> n; int x;
for (int i = 0; i < M; ++i) {
    for (int j = 0; j < N; ++j) {
        cin >> x;
        helper[x].push_back({i+1, j+1});
        // +1 to i & j since it is 1-indexed.
    }
}
```