Kick Start 2018 - Round G

Analysis: Cave Escape

Cave Escape: Analysis

Small dataset

Since there are no available potions in this dataset, we cannot increase our energy points and hence we are looking for a path with the smallest sum of traps which leads us to the exit. We can consider the cells of the grid as the vertices of our graph and moves between adjacent cells as the edges, with the weight of an edge being the absolute value of the destination cell. Running Dijkstra's algorithm on this graph gives us the path with the smallest sum of traps to reach the exit in O(**NM** * log(**NM**)) time.

Large dataset

We can observe that whenever are potions that we can take without going through traps, we should always take them all before going through a trap. After that, we have two recursive choices :

- Go to the exit if it is reachable without going through a trap.
- Go through a trap cell which is reachable without going through any other trap. This choice can help us get more potions or help us reach the exit cell. After we choose to go through a trap cell, we should again take all potions that we can reach without going through an unvisited trap, and then repeat the choice above.

Let C_T be the count of traps in our test case. A naive recursive solution based on the above would be of the order of O(**NM** * C_T!).

Observe that for a chosen subset of traps, all orderings of this subset have the same energy points (after going through all the traps), the same set of reachable unvisited traps and the same reachability of the exit cell. We can precompute these values by doing a graph search (BFS / DFS) on all 2^{C_T} subsets. To find the order of visiting the traps, we can use dynamic programming along with bitmasks. For every mask of traps, we can choose to go to the next unvisited reachable trap if its cost is not greater than our current energy points, or go to the exit cell if it is reachable and take the choice that maximizes our answer.

The time complexity for precomputation is O(**NM** * 2^{C_T}) and the time complexity of finding the ordering using dynamic programming is O(2^{C_T} * C_T).

Pseudocode

```
for every subset i in $2^{C_T}$ subsets precompute :
  $E_i$ := Energy points left after visiting traps in subset i
  $R_i$ := Reachable unvisited traps after visiting traps in subset i
  $Exit_i$ := Denotes if exit cell is reachable from given subset of visited traps

def MaxEnergy(mask) {
  ret := -1

  // Go to exit if it is reachable
  if $Exit_{mask}$ = True :
    ret := $E_{mask}$

  // Go to an unvisited reachable trap
  for i in $R_{mask}$ :
    if $TrapCost_i \le E_{mask}$ :
      ret := max(ret, MaxEnergy(mask | $2^i$))
  return ret
}
```