Problem

Our Battlestarcraft Algorithmica ship is being chased through space by persistent robots called Pylons! We have just teleported to a new galaxy to try to shake them off of our tail, and we want to stay here for as long as possible so we an buy time to plan our next move... but we do not want to get caught!

This galaxy is a flat grid of R rows and C columns; the rows are numbered from 1 to R from top to bottom, and the c lumns are numbered from 1 to C from left to right. We can choose which cell to start in, and we must continue to ju p between cells until we have visited each cell in the galaxy exactly once. That is, we can never revisit a cell, includi g our starting cell.

We do not want to make it too easy for the Pylons to guess where we will go next. Each time we jump from our curr nt cell, we must choose a destination cell that does not share a row, column, or diagonal with that current cell. Let (i, j) denote the cell in the i-th row and j-th column; then a jump from a current cell (r, c) to a destination cell (r', c') is i valid if and only if any of these is true:

r = r'
c = c'
r - c = r' - c'
r + c = r' + c'
Can you help us find an order in which to visit each of the R × C cells, such that the move between any pair of conse utive cells in the sequence is valid? Or is it impossible for us to escape from the Pylons?

Input

The first line of the input gives the number of test cases, T. T test cases follow. Each consists of one line containing t o integers R and C: the numbers of rows and columns in this galaxy.

Output

For each test case, output one line containing Case #x: y, where y is a string of uppercase letters: either POSSIBLE o IMPOSSIBLE, according to whether it is possible to fulfill the conditions in the problem statement. Then, if it is po sible, output R × C more lines. The i-th of these lines represents the i-th cell you will visit (counting starting from 1), and should contain two integers ri and ci: the row and column of that cell. Note that the first of these lines represents your chosen starting cell.

Limits

Time limit: 20 seconds per test set.
Memory limit: 1GB.

Test set 1 (Visible)

T = 16.
$2 \leq R \leq 5$.
$2 \leq C \leq 5$.

Test set 2 (Hidden)

$1 \leq T \leq 100$.
$2 \leq R \leq 20$.
$2 \leq C \leq 20$.

Sample

Input

Output

2
2 2
2 5

Case #1: IMPOSSIBLE
Case #2: POSSIBLE
2 3
1 1
2 4
1 2
2 5
1 3
2 1
1 5
2 2
1 4

In Sample Case #1, no matter which starting cell we choose, we have nowhere to jump, since all of the remaining ce ls share a row, column, or diagonal with our starting cell.

In Sample Case #2, we have chosen the cell in row 2, column 3 as our starting cell. Notice that it is fine for our final ell to share a row, column, or diagonal with our starting cell. The following diagram shows the order in which the ce ls are visited:

2 4 6 10 8
7 9 1 3 5

Solution:

```
/**
 *    author:  tourist
 *    created: 31.03.2018 11:41:48
**/
#include <bits/stdc++.h>

using namespace std;

int main() {
  ios::sync_with_stdio(false);
  cin.tie(0);
  mt19937 rng(5);
  vector<int> prec[22][22];
  for (int h = 2; h <= 20; h++) {
    for (int w = 2; w <= 20; w++) {
      vector<vector<int>> g(h * w);
      for (int i = 0; i < h; i++) {
        for (int j = 0; j < w; j++) {
```

```
        for (int ii = 0; ii < h; ii++) {
          for (int jj = 0; jj < w; jj++) {
            if (i != ii && j != jj && i - j != ii - jj && i + j != ii + jj) {
              g[i * w + j].push_back(ii * w + jj);
            }
          }
        }
      }
      for (int i = 0; i < h * w; i++) {
        shuffle(g[i].begin(), g[i].end(), rng);
      }
      vector<int> cur;
      vector<int> res;
      vector<int> was(h * w, 0);
      function<void(int, int)> dfs = [&](int cnt, int v) {
        if (!res.empty()) {
          return;
        }
        cur.push_back(v);
        was[v] = 1;
        if (cnt == h * w) {
          res = cur;
        }
        for (int u : g[v]) {
          if (!was[u]) {
            dfs(cnt + 1, u);
          }
        }
        cur.pop_back();
        was[v] = 0;
      };
      for (int i = 0; i < h * w; i++) {
        dfs(1, i);
      }
      prec[h][w] = res;
//      cerr << h << " " << w << " done, res.size() = " << res.size() << '\n';
    }
  }
  int tt;
  cin >> tt;
  for (int qq = 1; qq <= tt; qq++) {
    cout << "Case #" << qq << ": ";
    int h, w;
    cin >> h >> w;
    if (prec[h][w].empty()) {
      cout << "IMPOSSIBLE" << '\n';
    } else {
      cout << "POSSIBLE" << '\n';
      for (int x : prec[h][w]) {
        cout << x / w + 1 << " " << x % w + 1 << '\n';
      }
```

```
    }
  }
  return 0;
}
```