Problem

The first two paragraphs (not counting this one) of this problem and "New Elements: Part 1" are identical. The probl ms can otherwise be solved independently; you do not need to read or solve one in order to read or solve the other.

Muriel is on the path to discovering two new elements that she has named Codium and Jamarium. She has not been ble to isolate them yet, but she wants to start investigating some important properties, like their atomic weights, by i direct means. Since Muriel is working with a single isotope of Codium and a single isotope of Jamarium, their atomi weights are strictly positive integers.

Muriel managed to create N different molecules, each of which contains one or more atoms of Codium and one or m re atoms of Jamarium, and no other elements. For each molecule, she knows how many atoms of each element are pr sent in it. The molecular weight of a molecule is the sum of the atomic weights of all the atoms it contains.

As a first step, Muriel sorted the molecules by strictly increasing molecular weight. Now she wants to find out possi le integer values for the atomic weights of both Codium and Jamarium that are consistent with the ordering. Since sh is aware there could be many consistent pairs of values, she wants one that minimizes the atomic weight of Codium. If there are multiple pairs in which Codium's atomic weight is minimum, she wants the one in which Jamarium's ato ic weight is minimum.

Input

The first line of the input gives the number of test cases, T. T test cases follow. The first line of a test case contains a single integer N, the number of molecules. Each of the next N lines describes a different molecule with two integers i and Ji that represent the number of Codium and Jamarium atoms in the i-th molecule, respectively. The molecules re given in strictly increasing order of molecular weight.

Output

For each test case, output one line containing Case #x: y, where x is the test case number (starting from 1), and y is I POSSIBLE (in uppercase) if there is no pair of integer atomic weights that would make the order of the molecules st ictly increasing in molecular weight. Otherwise, y should be two integers c j where c is the atomic weight of Codium and j is the atomic weight of Jamarium, chosen according to the rules above.

Limits

Time limit: 20 seconds per test set.
Memory limit: 1GB.
$1 \leq T \leq 100$.
$2 \leq N \leq 10$.
$(Ci, Ji) \neq (Cj, Jj)$ for all $i \neq j$. (All molecules are different.)

Test set 1 (Visible)

$1 \leq Ci \leq 100$, for all i.
$1 \leq Ji \leq 100$, for all i.

Test set 2 (Hidden)

$1 \leq Ci \leq 109$, for all i.
$1 \leq Ji \leq 109$, for all i.

Sample

Input

Output

3
3
1 1
1 2
2 1
4
1 2
2 1
4 2
2 4
3
1 2
1 3
2 3

Case #1: 2 1
Case #2: IMPOSSIBLE
Case #3: 1 1

In Sample Case #1, the difference between the last two molecules is having an extra atom of one element or the othe . Given that the one having the extra Codium is heavier overall, we conclude that Codium must be heavier than Jama ium. The values 2 and 1 for the atomic weights of Codium and Jamarium make the molecular weights $1 \times 2 + 1 \times 1$ 3, $1 \times 2 + 2 \times 1 = 4$, and $2 \times 2 + 1 \times 1 = 5$, respecting the strict ordering. Since Codium is heavier than Jamarium in his case, 2 is Codium's minimum atomic weight, and 1 is of course Jamarium's minimum atomic weight.

Let a, b, c and d be the molecular weights of the molecules in Sample Case #2, in increasing order of molecular weig t. By their atom contents, $d = 2 \times a$ and $c = 2 \times b$. It follows from $a < b$ that $d = 2 \times a < 2 \times b = c$, which means there s no pair of values for the atomic weights that would make the ordering strictly increasing.

In Sample Case #3, notice that the molecules happen to be sorted in strictly increasing order of total number of atom . Therefore, assigning both elements an atomic weight of 1 makes the atomic weights be sorted in strictly increasing rder.

Solution:

```
#include <bits/stdc++.h>
#include <ext/pb_ds/assoc_container.hpp>

using namespace std;
#define PB push_back
#define MP make_pair
#define LL long long
#define int LL
#define FOR(i,a,b) for(int i = (a); i <= (b); i++)
#define RE(i,n) FOR(i,1,n)
#define REP(i,n) FOR(i,0,(int)(n)-1)
#define R(i,n) REP(i,n)
#define VI vector<int>
#define PII pair<int,int>
```

```
#define LD long double
#define FI first
#define SE second
#define st FI
#define nd SE
#define ALL(x) (x).begin(), (x).end()
#define SZ(x) ((int)(x).size())

#define unordered_map __fast_unordered_map
template<class Key, class Value, class Hash = std::hash<Key>>
using unordered_map = __gnu_pbds::gp_hash_table<Key, Value, Hash>;

template<class C> void mini(C &a4, C b4) { a4 = min(a4, b4); }
template<class C> void maxi(C &a4, C b4) { a4 = max(a4, b4); }

template<class TH> void _dbg(const char *sdbg, TH h){ cerr<<sdbg<<'='<<h<<endl; }
template<class TH, class... TA> void _dbg(const char *sdbg, TH h, TA... a) {
  while(*sdbg!=',')cerr<<*sdbg++;
  cerr<<'='<<h<<','; _dbg(sdbg+1, a...);
}

template<class T> ostream &operator<<(ostream& os, vector<T> V) {
  os << "["; for (auto vv : V) os << vv << ","; return os << "]";
}
template<class L, class R> ostream &operator<<(ostream &os, pair<L,R> P) {
  return os << "(" << P.st << "," << P.nd << ")";
}

#ifdef LOCAL
#define debug(...) _dbg(#__VA_ARGS__, __VA_ARGS__)
#else
#define debug(...) (__VA_ARGS__)
#define cerr if(0)cout
#endif

// some code copied from EBAIT from snackdown 2019 elimination round
struct Fraction {
  int p, q;
  bool operator<(const Fraction& other) const {
    return p * other.q < q * other.p;
  }

  vector<int> GetCC() const {
    int s = p / q;
    Fraction f{q, p - s * q};
    if (f.q == 0) { return {s}; }
    vector<int> other_cc = f.GetCC();
    other_cc.insert(other_cc.begin(), s);
    return other_cc;
  }

  static Fraction FromCC(VI cc) {
```

```
    reverse(ALL(cc));
    Fraction f{0, 1};

    bool is_first = true;
    for (int x : cc) {
      if (is_first) {
        is_first = false;
      } else {
        f = Fraction{f.q, f.p};
      }

      f.p += f.q * x;
    }

    assert(f.p <= 2e18 && f.q <= 2e18);
    return f;
  }
};

Fraction Approximate(Fraction lbound, Fraction rbound) {
  vector<int> lbound_cc = lbound.GetCC();
  vector<int> rbound_cc = rbound.GetCC();
  assert(count(lbound_cc.begin() + 1, lbound_cc.end(), 0) == 0);
  assert(count(rbound_cc.begin() + 1, rbound_cc.end(), 0) == 0);
  debug(lbound.p, lbound.q, lbound_cc);
  debug(rbound.p, rbound.q, rbound_cc);

  for (auto add_l : {false, true}) {
    for (auto add_r : {false, true}) {
      vector<int> lcc = lbound_cc;
      vector<int> rcc = rbound_cc;
      if (add_l) { --lcc.back(); lcc.PB(1); }
      if (add_r) { --rcc.back(); rcc.PB(1); }
      vector<int> res_cc;
      int ptr = 0;
      while (ptr < SZ(lcc) && ptr < SZ(rcc) && lcc[ptr] == rcc[ptr]) {
        res_cc.PB(lcc[ptr]);
        ++ptr;
      }
      int min_val = 2e9;
      if (ptr < SZ(lcc)) { mini(min_val, lcc[ptr]); }
      if (ptr < SZ(rcc)) { mini(min_val, rcc[ptr]); }
      assert(min_val < 1.5e9);

      res_cc.PB(min_val + 1);
      auto best_frac = Fraction::FromCC(res_cc);
      debug(lcc, rcc, res_cc, best_frac.p, best_frac.q);

      if (lbound < best_frac && best_frac < rbound) {
        return best_frac;
      }
    }
```

```
  }
  assert(false);
}

struct Testcase {
  int test_idx_;
  Testcase(int tidx) : test_idx_(tidx) {}

  VI Solve() {
    int N;
    cin >> N;
    vector<PII> elems(N);
    for (auto &el : elems) { cin >> el.st >> el.nd; }

    Fraction lbound{0, 1};
    Fraction ubound{(int)2e9, 1};

    for (int i = 1; i < N; ++i) {
      const int c1 = elems[i - 1].st;
      const int j1 = elems[i - 1].nd;
      const int c2 = elems[i].st;
      const int j2 = elems[i].nd;

      if (c1 >= c2 && j1 >= j2) { return {}; }
      if (c1 <= c2 && j1 <= j2) { continue; }

      if (c1 < c2) {
        assert(j1 - j2 > 0);
        maxi(lbound, Fraction{j1 - j2, c2 - c1});
      } else {
        assert(j2 - j1 > 0);
        mini(ubound, Fraction{j2 - j1, c1 - c2});
      }
    }

    debug(lbound.p, lbound.q, ubound.p, ubound.q);
    if (!(lbound < ubound)) { return {}; }
    auto ans = Approximate(lbound, ubound);
    return {ans.p, ans.q};
  }

  void Run() {
    auto ans = Solve();

    cout << "Case #" << test_idx_ << ": ";
    if (ans.empty()) {
      cout << "IMPOSSIBLE\n";
    } else {
      cout << ans[0] << " " << ans[1] << "\n";
    }
  }
};
```

```
int32_t main() {
  ios_base::sync_with_stdio(0);
  cin.tie(0);
  cout << fixed << setprecision(11);
  cerr << fixed << setprecision(6);

  int T;
  cin >> T;
  for (int i = 1; i <= T; ++i) {
    Testcase(i).Run();
  }
}
```