```
#include <bits/stdc++.h>
using namespace std;
#define IOS       ios_base::sync_with_stdio(false); cin.tie (nullptr)
#define PREC      cout.precision (10); cout << fixed

#if __cplusplus > 201103L
seed_seq seq{
  (uint64_t) chrono::duration_cast<chrono::nanoseconds>
    (chrono::high_resolution_clock::now().time_since_epoch()).count(),
    (uint64_t) __builtin_ia32_rdtsc(),
    (uint64_t) (uintptr_t) make_unique<char>().get()
};
mt19937 rng(seq);
#else
auto seed = chrono::high_resolution_clock::now().time_since_epoch().count();
mt19937 rng(seed);
#endif

template <class F = int>
class Rand {
  private:
    static const F __LOW = numeric_limits<F>::min();
    static const F __HIGH = numeric_limits<F>::max();

  public:
    F gvUnifRand (const F& low = __LOW, const F& high = __HIGH) {
      assert(low <= high);
      return uniform_int_distribution <F> (low, high)(rng); //closed interval [low,
high]
    }

    F gvRand (const F& low = __LOW, const F& high = __HIGH) {
      // if (is_floating_point<F>::value) return gvUnifRand (low, high);
      // else return gvFineRand (low, high);
      return gvUnifRand(low, high);
    }
};

template <class T = int>
class KuchBhiDedo {
  public:
    static const T __LOW = numeric_limits<T>::min();
    static const T __HIGH = numeric_limits<T>::max();
    Rand <T> rndT;
    Rand <int> rndInt;

    vector <T> gvVec(int n, const T& low = __LOW, const T& high = __HIGH) {
      vector <T> v(n);
      for_each(v.begin(), v.end(), [&] (T &x) -> void { x = rndT.gvRand (low, high);
 });
      return v;
    }

    set <T> gvSet (int n, const T& low = __LOW, const T& high = __HIGH) {
      assert(high - low + 1 >= n);
      set <T> s;
      while (static_cast <int> (s.size()) != n)
        s.insert(rndT.gvRand (low, high));
      return s;
    }

    vector <T> gvVecUnique (int n, const T& low = __LOW, const T& high = __HIGH) {
      set <T> s = gvSet(n, low, high);
      vector <T> v (s.begin(), s.end());
      random_shuffle(v.begin(), v.end());
      return v;
    }

    string gvString (int n, bool lowerCaseOnly = true) {
      string s(n, 'x');
      for (char &ch : s) {
        if (lowerCaseOnly) ch = static_cast <char>('a' + rndInt.gvRand (0, 25));
        else {
          bool up = rndInt.gvRand (0, 1);
          ch = static_cast <char> ((up ? 'A' : 'a') + rndInt.gvRand (0, 25));
        }
      }
      return s;
    }

    vector <vector <T>> gvMat (int mnn, int mxn, const T& low = __LOW, const T& high
 = __HIGH) {
      int n = rndInt.gvRand (mnn, mxn), m = rndInt.gvRand (mnn, mxn);
      vector <vector <T>> mat(n, vector <T> (m));

      cout << n << ' ' << m << '\n';
      for (auto &row : mat) {
        for (T& c : row)
          cout << (c = rndT.gvRand (low, high)) << ' ';
        cout << '\n';
      }
      return mat;
    }

    vector <int> gvPerm (int n) {
      vector <int> pi(n);
      iota(pi.begin(), pi.end(), 0);
      random_shuffle(pi.begin(), pi.end());
      return pi;
    }

    void gvTree (int mnn, int mxn, const T& low = __LOW, const T& high = __HIGH) {
      int n = rndInt.gvRand (mnn, mxn);
      cout << n << '\n';
      vector <int> isomorph = gvPerm(n);

       vector <T> nodeValue = gvVec(n, low, high);
      // for (T x : nodeValue) { cout << x << ' '; } cout << '\n';  // uncomment to
have NODE VALUES

      vector <pair <int, int>> edges;
      for (int u = 1; u < n; ++u) {
        bool swp = rndInt.gvRand (0, 1);
        int pr = rndInt.gvRand (0, u - 1);
        edges.push_back(swp ? make_pair(isomorph[u], isomorph[pr]) : make_pair(isomo
rph[pr], isomorph[u])); // undirected edge
      }
      random_shuffle(edges.begin(), edges.end());
      for (auto e : edges) {
        cout << e.first + 1 << ' ' << e.second + 1 <<
          // ' ' << rndT.gvRand (low, high) <<  // uncomment to have EDGE VALUES
          "\n";
      }
    }

    void gvDag (int mnn, int mxn, const T& low = __LOW, const T& high = __HIGH) {
      int n = rndInt.gvRand (mnn, mxn);
      int m = 0;
      vector <int> isomorph = gvPerm(n);
      vector <pair <int, int>> edges;

      for (int u = 0; u < n - 1; ++u) {
        int remVer = n - 1 - u;
        int k = rndInt.gvRand (0, remVer);
        m += k;

        set <int> neigh = gvSet(k, u + 1, n - 1);
        for (int v : neigh)
          edges.push_back(make_pair(isomorph[u], isomorph[v])); // directed edge
      }
      random_shuffle(edges.begin(), edges.end());

      cout << n << ' ' << m << '\n';

       vector <T> nodeValue = gvVec(n, low, high);
      // for (T x : nodeValue) { cout << x << ' '; } cout << '\n';  // uncomment to
```

```
have NODE VALUES

      for (auto e : edges) {
        cout << e.first + 1 << ' ' << e.second + 1 <<
          // ' ' << rndT.gvRand (low, high) <<  // uncomment to have EDGE VALUES
          "\n";
      }
    }
};

template <class T = int> T giveRand (const T& low, const T& high) {
   auto seed = chrono::high_resolution_clock::now().time_since_epoch().count();
   mt19937 mt_rand(seed);
   return uniform_int_distribution<T> (low, high)(mt_rand); //closed interval [low,
high]
}

signed main() {
  IOS; PREC;
  KuchBhiDedo <int> k;
  int tc = 1;
  // cout << tc << '\n';
  while (tc--) {
    int n = (int)2e5 - 1;
    cout << n + 1 << '\n';
    while (n--) {
      cout << (int)1e9 << ' ';
      // cout << giveRand((int)1e9 - (int)1e5, (int)1e9) << ' ';
    }
    cout << (int)1 << '\n';
    // cout << '\n';
  }
  return EXIT_SUCCESS;
}
```