Ad Hoc I

By: Ethan Pronev, Andrew Qi Tang

What is Ad Hoc?

Problems that don’t have a specific intended solution

Generally few or no algorithms needed

Each ad-hoc problem is unique, and requires a specialized approach

Can range from very easy (implementation) to very hard (puzzles, strategy games)

Problem 1: Beautiful Sets of Points

Manao has invented a new mathematical term — a beautiful set of points. He calls a set of points on a plane *beautiful* if it meets the following conditions:

1. The coordinates of each point in the set are integers.
2. For any two points from the set, the distance between them is a non-integer.

Consider all points (x, y) which satisfy the inequations: $0 \le x \le n$; $0 \le y \le m$; $x + y > 0$. Choose their subset of maximum size such that it is also a beautiful set of points.

Input

The single line contains two space-separated integers n and m $(1 \le n, m \le 100)$.

Output

In the first line print a single integer — the size k of the found beautiful set. In each of the next k lines print a pair of space-separated integers — the x- and y- coordinates, respectively, of a point from the set.

If there are several optimal solutions, you may print any of them.

Source: https://codeforces.com/problemset/problem/268/C

Solution 1: Beautiful Sets of Points

Let’s assume that $n \leq m$

If the number of points used is greater than n, then at least 1 row will contain more than one point

Any points in the same row have integer distance, so this is not allowed

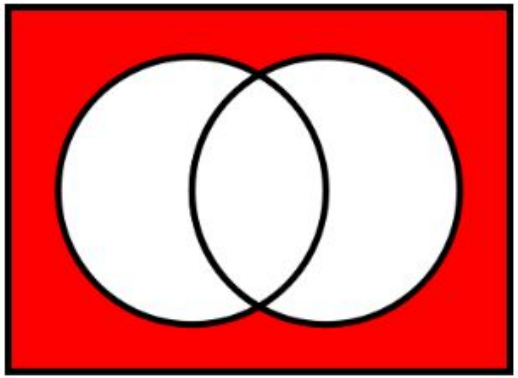
Therefore the maximum number of points is $\min(n,m)+1$ (+1 because we can place a point at $x=0$ or $y=0$)

The construction for this is placing every point along the diagonal ($x=y$)

Problem 2: NOR Operator

The only required knowledge is the NOR operator. All of its possible outputs can be stored concisely in this table.

a	b	$a \text{ NOR } b$
0	0	1
0	1	0
1	0	0
1	1	0



A Venn diagram of $A \text{ NOR } B$ from the Wikimedia Commons.

You are given a sequence A consisting of 0's and 1's. Here, the i^{th} element of A is denoted with A_i. A has length N $(2 \leq N \leq 10^6)$, and is indexed from 1 to N.

There are Q $(1 \leq Q \leq 10^5)$ queries, with each query consisting of integers x and y $(1 \leq x < y \leq N)$. For each query, output the value of $(A_x \text{ NOR } A_{x+1} \text{ NOR } \ldots \text{ NOR } A_{y-1} \text{ NOR } A_y)$ by itself on a line. Because the NOR operator is not associative, please evaluate the operations from left to right.

Source: https://dmoj.ca/problem/tle16c7p3

Solution 2: NOR Problem

a	b	a NOR b
0	0	1
0	1	0
1	0	0
1	1	0

The key point is realizing that anything NOR 1 = 0

Therefore, we can ignore any digits that come before a 1

Example:

0 NOR 1 NOR 0 NOR 1 NOR 0 NOR 0

= ~~0 NOR 1 NOR 0 NOR 1~~ NOR 0 NOR 0

= 0 NOR 0 NOR 0

Solution 2: NOR Problem

a	b	a NOR b
0	0	1
0	1	0
1	0	0
1	1	0

Using this idea, we can ignore every digit before the rightmost ‘1’ in a given range

Every digit after the rightmost ‘1’ is a 0

0 NOR 0 = 1

0 NOR 0 NOR 0 = (0 NOR 0) NOR 0 = 1 NOR 0 = 1

0 NOR 0 NOR 0 NOR 0 = (0 NOR 0 NOR 0) NOR 0 = 0

Therefore, NORing an even number of zeroes will return 1, and an odd number of zeroes will be 0

Solution 2: NOR Problem C++ code

Source: https://hastebin.com/jotapataja.cpp

```cpp
#include <bits/stdc++.h>
using namespace std;

int main()  {
  int n; cin >> n;  //inputting the size of the list
  int arr[n+1], last[n+1], idx=0; //variables/arrays used

  //idx is used to store the position of the last '1' inputted. Initally we assume the last '1' is at position 0

  //last[i] is used to store the position of the rightmost '1' in the range {1,i}

  for (int i=1; i<=n; i++)
  {
    cin >> arr[i];  //inputting each number in the list
    if (arr[i]==1) idx=i; //If the current element is a '1', then the last '1' is the current element
    last[i]=idx;  //the rightmost '1' <= current position is initialized
  }

  int q; cin >> q;  //getting the number of queries

  for (int i=1; i<=q; i++)
  {
    int l, r; cin >> l >> r;  //left and right indexes for the range
    int num=r-max(l,last[r])+1; //num is the number of 0's after the rightmost 1

    if (last[r]==l) cout << num%2 << endl;  //special case if the rightmost 1 is at the left pointer
    else cout << 1-num%2 << endl; // # of 0's % 2 determines the result of NORing them together
  }
}
```