code jam
print "hello, world!"

Practice Mode Rank: 13551 Score: 35 **asprazz** | Contest scoreboard | Sign out

Qualification Round 2017

- A. Oversized Pancake Flipper
- B. Tidy Numbers
- C. Bathroom Stalls
- D. Fashion Show

Contest Analysis

Ask a question

View my submissions

Submissions

Oversized Pancake Flipper

5pt	Correct **19628/23634 users** correct (83%)
10pt	Correct **17799/19074 users** correct (93%)

Tidy Numbers

5pt	Correct **24253/26071 users** correct (93%)
15pt	Correct **17756/22162 users** correct (80%)

Bathroom Stalls

5pt	Not attempted **13983/16043 users** correct (87%)
10pt	Not attempted **10822/13227 users** correct (82%)
15pt	Not attempted **5954/8864 users** correct (67%)

Fashion Show

10pt	Not attempted **996/2522 users** correct (39%)
25pt	Not attempted **591/843 users** correct (70%)

Top Scores

FatalEagle	100
ACMonster	100
y0105w49	100
johngs	100
HellKitsune123	100
SergeyRogulenko	100
spnautilus	100
BudAlNik	100
mjy0724	100
pwild	100

Contest Analysis

Overview | Problem A | Problem B | Problem C | Problem D

Oversized Pancake Flipper: Analysis

Let's start with two simple but key observations. First, the flips are commutative. That is, the order in which you make the flips doesn't matter: flipping at the same positions in any order always yields the same result. Second, you never need to flip at the same position more than once: since flips are commutative, you may rearrange the flips so that two flips at the same position happen consecutively, and then it is clear that they cancel each other out and you might as well not do either.

Small dataset

Applying the observations above to the Small dataset leaves only a really small number of combinations to try. Since there are **N**-**K**+1 possible flips, the number of flip subsets is $2^{\mathbf{N}-\mathbf{K}+1}$, which is at most $2^8 = 512$ for this dataset. We can just try all of these combinations, discard the ones that leave at least one pancake blank side up, and get the minimum number of flips or determine that it is impossible.

Even if you don't have the observations above, a breadth-first search of all possible pancake states, using all possible flips as transitions, is also easily fast enough.

Large dataset

Of course, the approach above will take way too long for the Large dataset, which can have up to 1000 pancakes. There is a simple greedy strategy that is not too hard to find. Notice that the left-most pancake p_1 is only affected by the left-most flip f_1. That means that the initial status of p_1 completely determines what do we need to do with f_1 if we want to have any chance of leaving all pancakes happy side up: use f_1 if and only if p_1 is initially blank side up. After deciding on f_1, we can notice that there is only one remaining flip f_2 that affects the second to the left pancake p_2. So, after we have used f_1 (or not), the current state of p_2 completely determines whether to use f_2.

Continuing with this reasoning, we can use the leftmost **N**-**K**+1 pancakes to determine all flips. After doing that, we only need to check whether the last **K**-1 pancakes are happy side up or not. If they all are, then the answer is the number of flips we used. Otherwise, the answer is IMPOSSIBLE. Notice that this reasoning also implies that at most one of the subsets of flips from our Small-only solution above would have worked.

If we carry out the above strategy literally and actually flip all of the pancakes in memory each time, the complexity is $O(\mathbf{N}^2)$. We can reduce this to $O(\mathbf{N})$ by only flipping one pancake at a time, and keeping a running count of the number of flips that we "owe" the current pancake. For instance, suppose that $\mathbf{N} = 10$ and $\mathbf{K} = 5$, and the first pancake is blank side up. We start with the first pancake, and because we must flip it, we know we must flip the first five pancakes. We increase our flip count to 1, and also make a memo to decrease the flip count by 1 once we reach the sixth pancake. We continue from left to right, and whenever a pancake is blank side up and the flip count is even, or a pancake is happy side up and the flip count is odd, we increase the flip count and make another memo. These memos can be easily stored in another array of length **N** that is checked before handling each pancake. This optimization is not necessary to solve the Large dataset, but it is a useful trick in programming contest problems (and at least one of your Code Jam engineers has used it in day-to-day software engineering work!)