DEVSHOW - Editorial

0

PROBLEM LINK:

Contest
Practice

Author: Praveen Dhinwa
Tester: Kevin Atienza and Sergey Kulik
Editorialist: Kevin Atienza

PREREQUISITES:

Greedy, ad hoc

PROBLEM:

There are n people, and each person has to vote two different people. No one can vote himself/herself. After the voting, all the highest- and lowest-voted people are kicked out of the show.

You are given the votes of the first $n-2$ people. If the last two people, friends Devu (person $n-1$) and Amit (person n), vote optimally, what is the maximum number of people among those two that can be saved? You also need to give one assignment of votes of the two that can achieve this.

QUICK EXPLANATION:

It can be shown that there are only two choices of votes per person that needs to be checked:

- Vote the highest-voted among the first $n-2$ people and your friend (Devu for Amit and vice versa), and
- Vote the highest-voted and second-highest-voted person among the first $n-2$ people.

(Note that the second choice is invalid if $n=3$, in which case there is only one choice of votes) Therefore, there are only at most $2 \times 2 = 4$ possibilities to check. Choose the one that saves the most among the two.

EXPLANATION:

Let's call the number of votes each person received as his/her **score**.

Let's consider a brute-force approach. Since each person has two votes, there are $(n-1)(n-2) = O(n^2)$ possibilities for each person on who to vote. So there are $O(n^4)$ possibilities in all, which clearly is a lot for us to check them all. Therefore, we must find ways to reduce these possibilities.

We want to save as many people among Devu and Amit as possible, which means that they must not be the lowest- or the highest-scoring people. If either Devu or Amit is currently the lowest-scoring, then we want to increase his score as much as possible, but since no one can vote oneself, then only Devu can increase Amit's vote and vice versa, and so we can only increase their scores by one. This means that if either Devu or Amit (or both) have lower scores than the other $n-2$ people, then he (or both, respectively) can't be saved.

Now, we also need to ensure that they don't have the highest scores. Thus, we want to increase the score of the highest scorer among the first $n-2$ people. It's always beneficial to increase the score of the highest scorer, because we have two votes each and we don't have anything else to do with the extra vote. However, we can only increase that score by at most 2, so if the scores of Devu and/or Amit are still larger (or equal to it) after this, then he (or both) can't be saved.

So far, here is our strategy:

- Have Devu vote for Amit and vice versa (to ensure they are not the lowest, if possible).
- Have them vote for the highest scorer among the first $n-2$ people (to ensure that they are not the highest, if possible).

Of course, if there are multiple highest scorers among the first $n-2$, then they should focus their votes on a single person (any one of the highest scorers).

So far, sounds reasonable, right? However, this strategy doesn't always work! For example, if we have the following table of votes (with $n=5$):

```
Person# | Vote1 | Vote2
--------+-------+-------
   1    |   2   |   3
   2    |   1   |   5
```

```
3   |   4   |   5
4   |   ?   |   ?
5   |   ?   |   ?
```

Then the scores are $[1, 1, 1, 1, 2]$, and according to our strategy, Devu and Amit should vote themselves and one of the highest scorers among the first 3, say person 1. With these votes, the new scores are $[3, 1, 1, 2, 3]$, which only saves Devu :(However, this is not optimal because there is a way to save both of them: Have Devu vote persons 1 and 2, and Amit vote person 1 and Devu. This way, the scores become $[3, 2, 1, 2, 2]$, and both are saved!

The above just shows that sometimes Devu or Amit should not vote their friend. In other words, their two votes must go to two of the $n-2$ people. Let's assume that this is possible, i.e. $n-2$ is at least 2, otherwise there is only one choice of voting.

Of course, it is still beneficial to vote the highest scorer among the first $n-2$, but what about the remaining vote? Clearly, we don't want to vote the lowest scorer (unless we have no choice, i.e. $n=4$) because that would risk having Devu or Amit become the lowest scorer. But other than that, any other choice doesn't matter! Any other person we vote won't give any risk for Devu or Amit to become the lowest scorer, and this person also won't exceed the score of the highest scorer because we have already voted the highest scorer twice! Therefore, we can just try voting an arbitrary person (that is not the lowest or highest scorer), instead of trying them all. To be safe, and to avoid having to handle the $n=4$ case specially, we can just check the second-highest scorer.

We have just shown above that for any person among Devu and Amit, there are only two choices of sets of votes to make that can give the best results, namely:

- Vote the highest scorer among the first $n-2$ people and your friend (Devu for Amit and vice versa), and
- Vote the highest scorer and second-highest scorer among the first $n-2$ people.

(Note that the second choice is invalid if $n=3$, in which case there is only one choice of votes). Since there are just at most $2 \times 2 = 4$ possibilities to check, and checking each possibility can be done $O(n)$ time (I trust you know how to do this part :D), then the whole algorithm takes $4 \times O(n) = O(n)$ time. Of course, you should select the set of votes that saves the most among Devu and Amit :)

Generalizations

Actually, the above strategy can be generalized. If we instead assume that each person votes exactly k people (the problem above is the special case $k=2$. Also we ensure $k < n$ so there is at least one valid way to vote), then for each of Devu and Amit there are only two choices of sets of votes that can give the best results:

- Vote the $k-1$ highest scorers among the first $n-2$ people and your friend (Devu for Amit and vice versa), and
- Vote the k highest scorers among the first $n-2$ people.

Even though there are more votes, there are still only at most $2 \times 2 = 4$ possibilities to check. Therefore, this takes $O(kn)$ time, dominated by reading the input. However, if the input is different: instead of the votes of each person, you are given the *scores* of the n people instead, then the algorithm takes $O(n)$ time.

We can generalize the problem further: by considering the case where there are more friends aside from Devu and Amit. Suppose we have c people to assign (the problem above is the special case $c=2$), then the straightforward extension of the above algorithm runs in $O(c^c n)$ time. But by analyzing a bit more, it is possible to solve this generalization in $O(n)$ time. We'll leave it to the reader to discover :)

Time Complexity:

$O(n)$

AUTHOR'S AND TESTER'S SOLUTIONS:

Setter
Tester 1
Tester 2

snck151c editorial

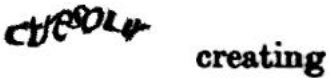
This question is marked "community wiki".

edited **9 hours ago**
admin ◆◆
10.9k●346●472●486

asked **2 days ago**
kevinsogo
1.6k●4●13●26
accept rate: 12%

Be the first one to answer this question!

[hide preview] community wiki



Type the text

Privacy & Terms

Post Your Answer