The Closed Judging Algorithm

Kjell Post
World Photographic Cup
kjell@irstafoto.se

Abstract

In competitions, where entries are ranked by score, it can be useful to increase the scores for certain entries to achieve a desired final order. (This is commonly known as "closed judging".) The algorithm shown in this paper accepts as input a sequence of entries and outputs new scores for the entries with the minimum amount of increase so that these entries take 1st, 2nd, 3rd, etc, place, without ties.

1 Introduction

To illustrate the problem, let's look at a real example. The following table shows the top six entries and their scores, in sorted order:

10322:	85.0 p
10324:	81.4 p
10328:	80.4 p
10326:	80.0 p
10323:	80.0 p
10330:	79.8 p
⋮	⋮

Let's say the closed judging decides that entry 10322 should remain in 1st place, entry 10328 should be 2nd place and 10326 should be in 3rd place:

10322:	? p
10328:	? p
10326:	? p
10324:	81.4 p
10323:	80.0 p
10330:	79.8 p
⋮	⋮

The question we address is: "What scores should we assign to the top three entries so that this order is achieved?" In our competition, we have the following requirements:

- The scores must remain the same, or increase, but never decrease.
- If scores are increased, they should be whole numbers.
- The top three scores must be unique.

  Specifically, there can not be a tie between the 3rd and 4th place.

The solution to the above problem is:

10322:	85.0 p
10328:	**83** p
10326:	**82** p
10324:	81.4 p
10323:	80.0 p
10330:	79.8 p
⋮	⋮

The scores in bold indicate that they have been changed according to the above requirements.

2 Algorithm

The algorithm operates in two steps: first "lift" the selected entries to the top, and then re-calculate their scores. Let S be the score table and $p_1, p_2, \ldots, p_n$ be the entries that are selected for the new top places. ($p_1 = 10322, p_2 = 10328, p_3 = 10326$ in the above example.)

Step 1: We begin by letting the nth entry float to the top, then the $(n-1)$th entry, etc, and finally the 1st entry.

function rise(p) // move p to beginning of S
  let i such that $S[i] = p$
  let $x \leftarrow S[i]$
  remove $S[i]$
  insert x at beginning of S

rise(p_n); **rise**(p_{n-1}); ...; **rise**(p_1);

Step 2: After the order is correct, we re-calculate the scores:

function dominate(p) // re-calculate score for p
  let i such that $S[i] = p$
  if $i \geq |S|$ **then return**
  let $s_i \leftarrow \mathit{score}(S[i])$
  let $s_{i+1} \leftarrow \mathit{score}(S[i+1])$
  if $s_i \leq s_{i+1}$ **then** $\mathit{score}(p) = \mathit{max}(s_i, \lfloor 1 + s_{i+1} \rfloor)$

dominate(p_n); **dominate**(p_{n-1}); ...; **dominate**(p_1);