Algorithms Competition Guidelines

AppHack X

April 5, 2019

The algorithms competition recognizes ingenuity in algorithm design and analysis of a module or a component in your AppHack projects. This document illustrates the expectations and deliverables for evaluating this award.

Writeup: For this award, you are to submit a pdf document that is no more than 2 pages which contains the following sections.

1. **Problem Definition:** Please define your problem clearly. You may need to abstract away unnecessary details of your project and focus on a component or module that you found an interesting solution to. Clearly state the inputs and the outputs. You may use common terminology found in the literature, for example, you can use $a_0 \dots a_{n-1}$ or $a[0] \dots a[n-1]$ to refer to an array or sequence a with n items. The input size is always referred to by the variable n. Please also indicate assumptions about the type of data or other assumptions about the inputs.

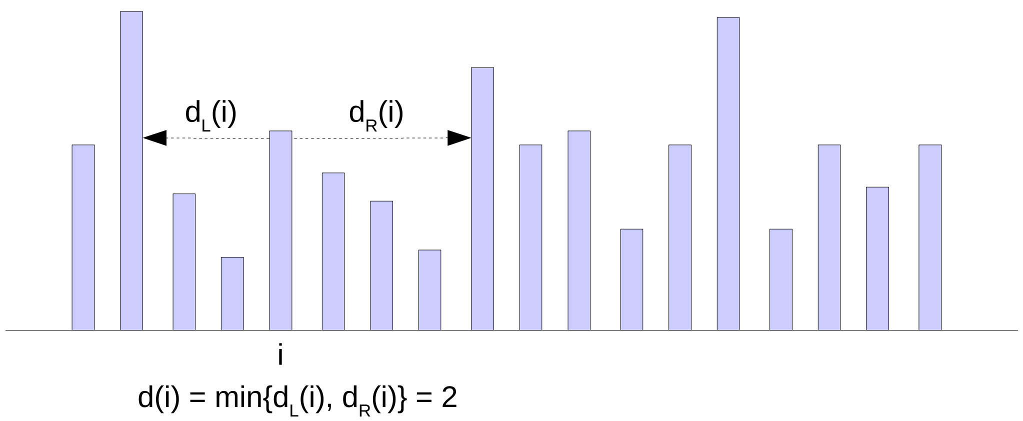
2. **Algorithm Description:** Please describe your algorithm in plain English precisely and concisely. Your description should be coherent in following a train of thought that transforms the set of inputs to the set of outputs. The individual steps of the algorithm should be clearly identifiable. Please do not include ad-hoc, random or vague descriptions of algorithms. Feel encouraged to simply list the step-by-step instructions of the algorithm, and include neat figures or visualizations to aid in your explanations. You may also include links to web-pages that you have created that provides visualizations of your algorithms.

   In explaining algorithms, it is important to abstract away unnecessary details and focus on what is important. Common algorithms and data structures can be used as black boxes without needing further elaboration. You can use standard terminology in interacting with these data structures, for example, you can say "*find* item x in a binary search tree", without requiring to elaborate on the *find* operation. Also such operations can be written in italics. If you are using advanced algorithms as black boxes, then please provide a link to web-pages or actual peer-reviewed publications that contains their descriptions. For example, you can say "please refer to the Reingold-Tilford algorithm for drawing trees in a 2 dimensional plane `https://rachel53461.wordpress.com/tag/reingold-tilford/`". Note that your contribution should be clearly distinguished from the black boxes you use.

3. **Algorithm Analysis:** Please clearly describe the running time (or/ and space) complexities of your algorithm. You need to reason in your own words why the algorithm is correct and what the running rime or space complexities are. Merely listing these as facts does not suffice.

An example is shown below.

1. **Problem Definition:** As part of our project, we are given an array that contains heights of n individuals $a_1 \dots a_n$ standing in a line. The *left-domination* $d_L(i)$ for an individual i is the number of consecutive elements leading upto i from the left that are all smaller than a_i. That is, we find the largest index j for which elements $a_{i-1}, a_{i-2}, \dots a_{j+1}$ are all smaller than a_i, and that $a_j > a_i$, $1 \le j < i \le n$. The *left-domination* of i is then $i - j$. Similarly, we find the smallest index j, $i < j \le n$, for which elements $a_{i+1}, a_{i+2}, \dots a_{j-1}$ are all smaller than a_i, and that $a_j > a_i$. The *right-domination* $d_R(i)$ for an individual i is then $j - i$. The domination radius $d(i)$ for individual i is the minimum of its left and right dominations, i.e., $d(i) = \min\{d_L(i), d_R(i)\}$. We have designed an in-place algorithm for computing the domination radius of every individual. Without loss of generality, we assume that the ends contain people at infinite height, i.e., $a_0 = a_{n+1} = \infty$.



2. **Algorithm Description:** The naive solution is to scan through the array of heights one individual at a time and compute his/her domination radius in the straightforward fashion – scan left first and then right from i until we reach an individual taller than i to compute $d_L(i)$ and $d_R(i)$ respectively. Then the domination radius can be computed as $d(i) = \min\{d_L(i), d_R(i)\}$. If the array is sorted in ascending order, then for each element i, it could take $O(n)$ time to compute $d_L(i)$. So the total running time is $O(n^2)$.

   We can do slightly better by stepping in both directions at the same time. So starting from individual i, we step in both $(i-1)$ and $(i+1)$ in the first iteration, then in both $(i-2)$ and $(i+2)$ in the second iteration, and so on until we reach any individual in either side who is taller. This reduces the running time $O(n \log n)$.

3. **Algorithm Analysis:** To analyze the running time, we first observe that no element (or individual) takes more than $n/2$ iterations to compute its domination radius because there are only n elements. For elements that take $n/4$ iterations, there could be only at most 3 of them – in positions $n/4$, $n/2$, and $3n/4$. So if the domination radius is in the range $n/4 \dots n/2$, then there won't be more than 4 elements. Extending the same argument to elements whose domination radius is in $n/8 \dots n/4$, we see that there won't be more than 8 of them. Similarly, there are at most 16 elements with domination radius in $n/16 \dots n/8$, and so on. We can summarize these in the following table.

Domination radius range	# elements with this domination radius	total work done
$n/4 \dots n/2$	4	$2n$
$n/8 \dots n/4$	8	$2n$
$n/16 \dots n/8$	16	$2n$
$\dots$	$\dots$	$\dots$

For each of the elements in the first category, we spend at most $n/2$ time to find its domination radius. Adding up over all 4 elements gives a total of $2n$ units of work for all the elements in the first category. Similarly, it takes at most $n/4$ units of time to compute the domination radius for elements in the second category, and there are 8 of them which totals to $2n$ units of work. So the total work done in every category is $2n$ units, and there are only $O(\log n)$ total categories as each category reduces the range by a factor of 2. So adding up all work done across all categories, we get a total time of $O(n \log n)$.