Good Problems

Ahmed Zaheer Dadarkar
tymefighter

July 31, 2020

1 Sum of Floor of Harmonic Progression

2 Maximum Area of a rectangle formed on a histogram array

We are given an array of heights of histogram bars $h[0..n-1]$, we must construct a rectangle with base formed by indices $\{l, .., r\}$ and height

$$ht = \min_{j \in \{l,..,r\}} h[j]$$

such that the area of the rectangle is maximum.
We solve this problem by computing for each $i \in \{0, .., n-1\}$ the maximum left and right indices of the base such that the height of the rectangle remains $h[i]$. We do this by computing the following two arrays l and r:

$$\forall i \in \{0, .., n-1\}\ l[i] = j\ such\ that\ \forall k \in \{j, .., i\}\ h[k] \geq h[i]$$

$$\forall i \in \{0, .., n-1\}\ r[i] = j\ such\ that\ \forall k \in \{i, .., j\}\ h[k] \geq h[i]$$

Then the maximum area $maxArea$ would be as follows:

$$maxArea = \max_{i \in \{0,..,n-1\}} (r[i] - l[i] + 1) * h[i]$$

We can compute l and r using the following algorithm that uses a stack. We describe the algorithm only for l, it can be run in reverse to compute r as well. When we process element indexed by i, we first remove all the elements of the stack which have value less than $h[i]$, then if the stack is empty we assign $l[i]$ zero, else we assign the index of the top element in the stack. Then, we insert the current element along with its index.
To see why this works, we need to show that the elements in the stack are actually the results of repeated application of a particular function. The function is as follows:

$$\forall i \in \{0, .., n-1\},\ \phi(i) = j\ such\ that\ \forall k \in \{j, .., i\}\ h[k] \geq h[i]$$

Now, at the start of ith iteration, the stack would contain the following (left is top, right is bottom)

$$stack = [i-1, \phi(i-1), \phi(\phi(i-1)), ...]$$

On completion of the ith iteration, the stack would become this

$$stack = [i, \phi(i), \phi(\phi(i)), ...]$$

The reason for this is that $\phi(i)$ must be inside the stack at the start of the ith iteration itself, then we remove elements until we find it. If it were not so, then our assumption that the stack was correct at the start of ith iteration would be contradicted (this can be shown but I am too lazy to do it).

3 Number of Distinct Substrings that occur atleast twice in a given string

Essentially the solution is to add all those values of the lcp array which are greater than the next value, and also add the final value in the lcp array (i.e. the $n-1$ indexed element). The lcp array is constructed using the suffix array by Kasai's Algorithm.
The reason why this is true is that when $lcp[i] = x > 0$ for some i, then we already have x strings that occur atleast twice. But we cannot simply add all the lcp values since there may by substrings that are repeated. We can see that if $lcp[i-1] > lcp[i]$, then $lcp[i-1] - lcp[i]$ substrings are unique, hence we can count them.

4 Maximum Number of Points that can be covered by a circle

You are given a set of points S in the X-Y plane. You are alse given a the radius $r > 0$ of a circle which you must construct so as to maximize the number of points from S which lie inside or on the circle.

We initialize the answer by 1 because we can construct the circle with any of the points as the center. Now, consider an optimal solution C which is a circle that covers the maximum number of points. If the number of points covered by C is 1, then we already have the answer since we initialize the answer by 1. If the answer is greater than 1, then consider the cases where the number of points that lie on the circle are less than 2. The first case is when no point lies on the circle, then we translate the circle until atleast one point lies on it (we may end up in the second case by this). The second case is when a single point lies on the circle, now we pivot the circle at this point and rotate until atleast one more point is on the circle. Now that we have shown that an optimal solution exists which has alteast two points on the circle.

We construct a solution by going over all possible pairs of points and constructing circle, and for each constructed circle we count how many points lie inside or on the circle. Then answer is the maximum over all these values.

5 All lengths to which a string can be compressed

6 Number Of Occurrances of Every Prefix

7 Small To Large

You are given a set $S = \{S_1, .., S_n\}$ consisting of n multisets each currently containing a single element. You have to perform $n - 1$ operations, in each operation you would be given i and j and you have to merge the sets S_i and S_j and replace them with $S_{min(i,j)}$ and then provide the number of distinct elements in that set. Method is that when asked to merge to sets, place the elements of the smaller set into the larger one. Now, consider any element $x \in \bigcup_{i=1}^{n} S_i$, this element is added to a new set $O(\log n)$ times, the reason being that the size of the set this element is part of becomes atleast double of the previous one. Hence, after m merges, we have

$$2^m \leq n$$

So,

$$m = O(\log n)$$

Now, each add operation has $O(\log n)$ cost, hence for each element we have a total cost of $O(\log^2 n)$, So for all elements together we have time complexity

$$O(n \log^2 n)$$

7.1 DSU using size of component

The method is that each time we are asked to merge two components, we make the node having subtree with larger size as the parent of the other Then at each such operation, we increase the size of the smaller tree by atleast a factor of 2.

Hence, since each op costs $O(\log n)$, and there are $O(\log n)$ such operations for **each** element, the total complexity is $O(n \log^2 n)$

7.2 A Tree Problem

You are given a tree rooted at 0 and a color on each node, you are then asked to compute the number of distinct colours in the subtree of each node.
The method is to find out for each node which child is heavy, by heavy we mean that it has the largest size of subtree compared to other children. Then we

merge the sets of all other children into the heavy child.
An important thing to note is that we must not create a new set for the parent, instead we must keep the reference to the heavy child's set as the parent.
For analysis, assume the color at each node is unique, meaning no other node has it. Now, pick color c at any node(initially), then c is moved at most $O(\log n)$ times since the set containing c has it's size doubled at each operation. Also, cost of each operation is $O(\log min(n, |C|))$ where C is the set of all colors, and when we consider all elements, we have the following complexity

$$O(n(\log n)(\log min(n, |C|))) = O(n \log^2 n)$$

8 Query for Number of Distinct elements in a Subarray

9 References

- https://cp-algorithms.com/string/z-function.html
- https://cp-algorithms.com/string/prefix-function.html
- http://codingwithrajarshi.blogspot.com/2018/06/small-to-large.html
- https://stackoverflow.com/questions/39787455/is-it-possible-to-query-number-of-distinct-integers-in-a-range-in-olg-n