CS6363 - Design and Analysis of Algorithms

HomeWork - 4

Name: HIMA SRI.T
UTDID: 2021624644

① Let $X[1,\dots,n]$ and $Y[1,\dots,n]$ be two arrays that are already sorted. Give an $O(\lg n)$-time algorithm to find the median of all $2n$ elements in arrays X and Y.

Sol: Input: Two sorted arrays of length n.
⇒
$X[1,\dots,n]$ $X[1] \le X[2] \le \dots X[n]$
$Y[1,\dots,n]$ $Y[1] \le Y[2] \le \dots Y[n]$.

Output: Median of all the $2n$ elements in arrays X & Y.

Usually median of a sorted array is calculated as below:

A sorted array $A[1,\dots,n]$:

Median of A:

If n is even:

$$\text{median} = (A[n/2] + A[n/2-1])/2$$

i.e mid value of the 2 middle elements.

If n is odd:

$$\text{median} = A[n/2]$$

i.e middle element

Brute-force: For brute-force we can merge 2-arrays and find the median of the single array using the above formula: But time-complexity here is $O(n)$; which is the time taken to merge the 2 arrays of size n.

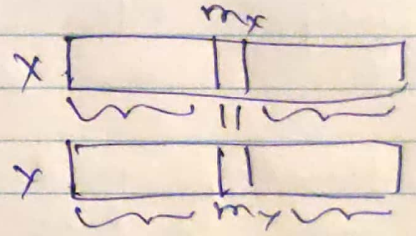
<u>Divide & Conquer approach:</u>

In this algorithm, we can find the median of each sorted array and then based on the value of those medians; divide the search space.

① Let m_X = median of X array
m_Y = median of Y array.
Calculate the median of the X and Y array.

② <u>Case 1:</u>

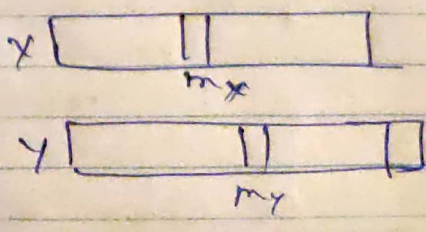
If $m_X == m_Y$

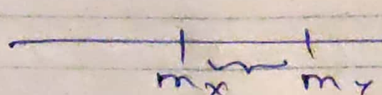


Here half ~~all~~ elements of X & Y arrays combined are less than m_X or m_Y; $||^{ly}$ the other half elements are greater than m_X. Hence, m_X or m_Y is the median of the combined array.

<u>Case 2:</u>

If $m_X < m_Y$





↳ In this case, the median lies in btw. the m_X & m_Y.

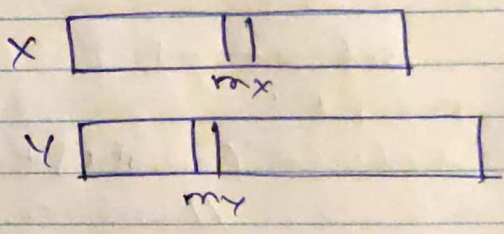
Because,
the left half of elements in array X are less than m_x. Hence we ignore them.
Similarly, the right half of elements in array Y are greater than m_y. Hence we can ignore them as well as it won't contain the median.

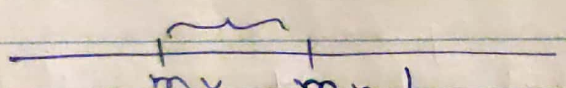
Here, in this case; the median lies in this array part

$X[n/2+1, --- n]$ and $Y[0, ---, n/2]$

<u>Case 3:</u>

If $m_x > m_y$





In this case, the median lies btw. m_x & m_y.

The right half of elements in array X are greater than m_x. Hence we can ignore them.
Similarly, the left half of elements in array Y are less than m_y. Hence we can ignore them.

Here, in this case, the median lies in the array part

$X[0, ----, n/2]$ and $Y[n/2+1, -- n]$

Pseudo Code:

Median-of-single-array (arr, n):

```
① If (n % 2 == 0) // even elements
     return (arr[n/2] + arr[n/2 - 1]) / 2;

② else      // odd elements
     return arr[n/2];
```

Median(arr1, arr2, n):

```
① If n == 0:
       return -1;
② If n == 1:
       return (arr1[0] + arr2[0]) / 2;
③ If n == 2:  // when array has 2 elements
       return (max(arr1[0], arr2[0]) + min(arr1[1], arr2[1])) / 2

   // This is based on the merge procedure with
        2 elements in each array
④ mx = Median-of-single-array(arr1, n);
⑤ my = median-of-single-array(arr2, n);
⑥ If (mx == my)
       return mx;
⑦ If (mx < my):
       If (n % 2 == 0)
         return Median(arr1 + n/2 - 1, arr2, n - n/2 + 1);
       else
         return Median (arr1 + n/2, arr2, n - n/2);

   // For considering the right half of arr1 & left
        half of arr2.
```

(8) else if (mx > my):
    if (n%2 == 0)
        return Median(arr1, arr2 + n/2 - 1, n - n/2 + 1);
    else
        return Median(arr1, arr2 + n/2, n - n/2);

// For considering the left half of arr1 & right half of arr2

Time Complexity of the above algorithm:

$$T(n) = T(n/2) + 1.$$

Since the array search space is reduced to half.

Based on master's theorem;

$$f(n) = 1 = O(n^{\log_2 1}) = O(n^0) = 1$$
$$= O(n^{\log_b a})$$

$\therefore$ Time Complexity $= O(n^{\log_b a} \lg n) = O(1 \lg n)$ (Based on MT Case 2)

$\therefore$ Time Complexity $= O(\lg n)$.

② Solve Exercise 23-1.10 on page 630 in CLRS. Given a graph G and a minimum spanning tree T, suppose that we decrease the weight of one of the edges in T. Show that T is still a minimum spanning tree for G with edge weights given by weight function w. Choose one edge $(x, y) \in T$ and a positive number k, and define the weight function w' by

$$w'(u,v) = \begin{cases} w(u,v) & \text{if } (u,v) \neq (x,y) \\ w(x,y) - k & \text{if } (u,v) = (x,y) \end{cases}$$

Show that T is a minimum spanning tree for G with edge weights given by w'.

Sol: Given :

A Graph G with vertices V.
T is a minimum spanning tree.

Let us get that
W = weight of the minimum spanning tree T.

$$w(T) = W = \sum_{(u,v) \in T} w(u,v)$$

To Prove:

T is still a minimum spanning tree for the graph G with edge weights given by w'.

Proof:

let us define

e = edge whose weight is decreased by k.

$w(e)$ = weight of edge e in given graph G

$w'(e) = w(e) - k$
= weight of edge e in the obtained graph G' after reducing weight

$w(T)$ = weight of the minimum spanning tree T

$= \sum_{(u,v) \in T} w(u,v)$.

$w'(T)$ = weight of the MST with weight func $w'(u,v)$

Assume that T is no longer a minimum spanning tree when its edge is reduced by k.

let T' is the new minimal spanning tree with weight function $w'(T')$

There are two cases which arise here:

Case i) $e \in T'$

i.e reduced edge is a part of the new MST T'

$w'(T') < w'(T)$ // since T' is the new MST.

$w'(e) + w'(T'-e) < w'(e) + w'(T-e)$

$w(e) - k + w'(T'-e) < w(e) - k + w'(T-e)$

// separting e since it is the modified edge

// Since e is only the edge which caused the change, other edges weight would not impact

$\Rightarrow w(T'-e) < w(T-e)$

$[\because w'(u,v) = w(u,v) \text{ if } (u,v) \neq (u,v)]$

Adding the original weight of e on both sides:

$$w(e) + w(T' - e) < w(T - e) + w(e)$$

$$\Rightarrow w(T') < w(T)$$

It gives us that ~~the ori~~ even with the original weights, ~~the~~ T' is the minimal spanning tree in graph G.

⇒ It is a contradiction because given that T is the MST of original graph G.

∴ Our assumption that T' is the new MST ~~of~~ is false.

Case (ii) $e \notin T'$

i.e reduced edge is not a part of the new MST.

$$w'(T') < w'(T)$$

// Since e is not a part of the tree T'

$$w(T') < \underbrace{w(T) - K}_{\text{reduced edge weight}} < w(T)$$

$$\Rightarrow w(T') < w(T)$$

Again we arrived at a stage that in the original graph G with weight function w, T' is the MST ⇒ which is not true according to the given information

⇒ Our assumption that T' is the new MST is false.

∴ In both cases, we arrived at contradiction that T' is not the new MST

⇒ T is still the minimal spanning Tree.

Hence proved that;
T is the minimal spanning tree of graph G with edge weights given by w'.

∴ T is still the minimal spanning tree even if we decrease the weight of one of edges of T

(3) Exercise 16.2-5 on page 428 in CLRS.
Describe an efficient algorithm that given a set $\{x_1, x_2, ---, x_n\}$ of points on the real line, determines the smallest set of unit-length closed intervals that contains all of the given points. Argue that your algorithm is correct.

Sol: I/P: set of points on the real line
$\{x_1, x_2, ---, x_n\}$.

O/P: smallest set of unit length closed intervals that contains all of the given points.

Eg: I/P: $\{0.5, 1.6, 1.55, 1.4, 2.5\}$
O/P: $\{[0.5, 1.5]; [1.55, 2.55]$.

Algorithm:

(i) Sort the given input points in non-decreasing order.
i.e. sorted $Z = \{z_1, z_2, z_3, ---, z_n\}$
$z_1 \leq z_2 \leq -- \leq z_n$

(ii) Do a linear search on the sorted set Z; from the beginning. Everytime we encounter a z_i, for some $i \in \{1, 2, --, n\}$, we can add the closed interval $[z_i, z_i+1]$ into the solution set S.

(iii) After adding the interval; removes all the elements $z_k \in [z_i, z_{i+1}]$ from the set.

(iv) Repeat the above process until there are no elements in the set Z.

Pseudocode:

* Sort the given points $X = x_1 \leq x_2 \leq \cdots \leq x_n$

* Let S be the set of unit length intervals.

* For each $1 \leq k \leq n$, let $I_k = [x_k, x_{k+1}]$

* Initialize $S = \phi$; // empty set in the beginning

* While ($X \neq \phi$): // i.e while there are points in set X.
  for each $x_k \in X$.
  → $S = S \cup \{I_k\}$

  → $X = X - \{x_j$ s.t $x_j \in I_k\}$.

* Return S.

~~Proof of algorith~~

Time Complexity:

The time complexity depends on the sorting the input and traversing the input in linear search

∴ Time complexity: $O(n\lg n + n) = O(n \lg n)$

Proof of algorithm:

We are using greedy approach here as we choose the best possible choice at every step

Scanned with CamScanner

Let us suppose that
S_1 is an optimal solution which contains the interval $[p, p+1]$ where x_1 is covered. i.e $p < x_1$.
As we could see that there are no elements in the given set X which are btw. x_1 & p as x_1 is the smallest element.
i.e As x_1 is leftmost points, there are no points btw. x_1 & p. i.e $[p, x_1) \notin X$.
Therefore, we can replace p with x_1 in S_1 where the interval becomes $[x_1, x_1+1]$.

$||^{ly}$, if we extend the argument to all other elements; we get that S_1 is equal to S which we constructed using greedy property
i.e $|S| = |S_1|$

This proves that the given algorithm is correct