

Jay Kalyanbhai Savani
CWID – 20009207
Homework -4

 EXERCISE | 7.5.5 

(a) One additional feature of the list-based implementation of a union-find structure is that it allows for the contents of any set in a partition to be listed in time proportional to the size of the set. Describe how this can be done.

[Feedback?](#)

Ans: The contents of any set in a partition may be listed in time proportionate to the size of the set by representing the list-based implementation using a linked list.

Take into account a list for set A that has the head node holding the reference to the beginning and end nodes of a linked list having pointers to all of A's items. A pointer to an element from that set and a pointer to the head node for A are stored at each node of a linked list for A.

The insertion operation for n elements in a linked list will take $O(n)$ time when creating a set using that method.

 EXERCISE | 7.5.9 

(a) Describe how to implement a union-find structure using extendable arrays, which each contains the elements in a single set, instead of linked lists. Show how this solution can be used to process a sequence of m union-find operations on an initial collection of n singleton sets in $O(n \log n + m)$ time.

[Feedback?](#)

Ans: When given a single set of elements, such as 1, 2, 3, ..., n , execute a UNION operation on the set by combining smaller sets into bigger sets. This reduces amortized costs since all of the components are effectively walked from 1 to n .

Consider an element x in the union-find operation sequence for m . The size of the set holding element x increases by one every time the element pointer changes.

After n unions, the set containing x will have a size of $2n$. A node's update pointer may be modified a maximum of $\log n$ times.

Consequently, it will take O to process all n nodes of the union-find data structure ($n \log n$). Amortized time is O for each union ($\log n$).

The find and make set operations in a union-find structure will require $O(1)$ time.

Union operations will require $O(n \log n + m)$ time in the worst case and for a series of m .



- (a) Consider the game of Hex, as in the previous exercise, but now with a twist. Suppose some number, k , of the cells in the game board are colored gold and if the set of stones that connect the two sides of a winning player's board are also connected to $k' \leq k$ of the gold cells, then that player gets k' bonus points. Describe an efficient way to detect when a player wins and also, at that same moment, determine how many bonus points they get. What is the running time of your method over the course of a game consisting of n moves?

[Feedback?](#)

Ans: We can determine when a player wins and how many additional points, they will get using the Union-Find data structure.

1. First, we create a singleton set for the n nodes and name every node in the $n * n$ grid from 0 to $n^2 - 1$. For instance: {1}, {2}, {3},{n}.
2. Starting from both corners and moving from left to right, respectively, place stones of the black and white colors from top to bottom.
3. Starting from two end points, link each new cell with its neighboring cell if they have the same color (black or white) using the union procedure (left-right or top- bottom).
4. Use the find operation to see whether the newly inserted cell is a member of the same color cell family. Make sure every set with a cell on the left boundary is identical to any set with a cell on the right boundary (similarly for top-bottom).
5. This is the winning move if the condition is true; else, proceed with the union-find. Conditions may also be examined from top to bottom.

From one end to the other, we complete the chain (left to right or top to bottom)
Count the amount of gold cells (k') in the final set when the chain is complete to determine the winning candidate's bonus points.

In this case, creating a set will take $O(n)$ time, but making a union of n elements would require $O(n \log n)$ time.

As a result, the runtime will be $O(n \log n)$.



EXERCISE

8.5.12



- (a) Suppose we are given a sequence S of n elements, each of which is colored red or blue. Assuming S is represented as an array, give an in-place method for ordering S so that all the blue elements are listed before all the red elements. Can you extend your approach to three colors?

[Feedback?](#)

Ans: All of the blue items are listed before all of the red elements in S using an in-place (Quick-sort) approach.

1. Take the integers "1" and "0," which scan items in the right and left directions, respectively. Increase the value for l until it reaches the red element if it is at the blue element.
2. If the value for 0 is at the red element, decrease it until it reaches the blue element.
1. 3) Swap the elements $S[l]$ and $S[0]$ when l reaches the red element and 0 reaches the blue element.
2. 4) Repeatedly use this technique until the positions of l and 0 coincide.

The aforementioned strategy may be expanded to three colors by using it twice to solve the problem.

First, by shifting one color to the left of the array and switching the other two colors to the right of the array, we can solve it for only one color. Applying the same strategy once again to the two more colors that were shifted to the right

The starting point for this will be the last position of the sorted color, where l and r will be at the array's rightmost index, and the same methodology as described above will then be used.

The time complexity of the above approach for sorting an array of n elements with two or three colors is $O(n \log(n))$. This occurs when the array is already partially sorted, such that the pivot chosen by the algorithm always divides the sub-array into two roughly equal halves in each iteration. This leads to a logarithmic growth in the number of iterations required, resulting in an $O(n \log(n))$ time complexity.



EXERCISE

8.5.22



- (a) Suppose we are given an n -element sequence S such that each element in S represents a different vote in an election, where each vote is given as an integer representing the ID of the chosen candidate. Without making any assumptions about who is running or even how many candidates there are, design an $O(n \log n)$ -time algorithm to see who wins the election S represents, assuming the candidate with the most votes wins.

[Feedback?](#)

Ans: In an election, each vote is represented by an integer, and we are given an n -element sequence called S that has this property.

An algorithm to determine the election's winner:

1. Using the quicksort technique, we first sort the items in sequence S .
2. Go down the list and note how many votes each contender has received. The candidate who obtains the most votes should use the phrase "maxVotes," maintain track of the results, and consult the other candidates on the list.
3. Replace the maximum votes with the current votes if the total votes cast for any other candidate exceeds "maxVotes."
4. after going over the list, the candidate who received the most votes would be declared the winner.

It will take $O(n \log n)$ time to sort all the components in this case. The time required to traverse every component of sequence S is $O(n)$.

The method will thus complete in $O(n \log n)$ time.

 EXERCISE | 8.5.23 

(a) Consider the voting problem from the previous exercise, but now suppose that we know the number $k < n$ of candidates running. Describe an $O(n \log k)$ -time algorithm for determining who wins the election.

[Feedback?](#)

Ans: Considering the prior exercise and algorithm

1. In order to build the technique, we employ a balanced search tree, where each tree node represents a contender's ID and keeps the number of votes the candidate has earned.
2. The initial count for votes is 0, and when we start to traverse the items in S sequence, the count of votes for a certain ID will be increased.
3. We search through every node in the tree to identify the candidate ID that has the most votes.

It will take $O(\log k)$ time to find and update the tree for k items.

It will take $O(n)$ time to traverse the n -element sequence in sequence S .

The method will thus complete in $O(n \log k)$ time.



- (a) Suppose you are given two sorted lists, A and B , of n elements each, all of which are distinct. Describe a method that runs in $O(\log n)$ time for finding the median in the set defined by the union of A and B .

[Feedback?](#)

Ans: Two unique sorted lists with n entries each, A and B , are presented to us.

How to get the median in the set that union of A and B defines:

- a. To begin, we compute the medians of lists A and B that have been sorted.
- b. Return either medA or medB if medA and medB are equal.
- c. If none of the two subarrays specified below has a median, then medA is bigger than medB .
 - i. From medB to final element of B ($B[n/2]$ to $B[n-1]$) or
 - ii. from first element of A to medA ($A[0]$ to $A[n/2]$).
- d. If medA is lower than medB , then one of the two subarrays specified below contains the median.
 - i. From the first element of list B to the medB ($B[0]$ $B[n/2]$) or
 - ii. From the middle to the final element of list A ($A[n/2]$ $A[n-1]$).
- e. Repeat the aforementioned procedure until both subarrays' sizes equal 2.
- f. The median is calculated using the following formula when the size of the lists is two: $\text{median} = (\max(A[0], B[0]) + \min(A[1], B[1])) / 2$

Runtime for the aforementioned algorithm is $O(\log n)$.

Example:

$A = \{4, 8, 14, 22, 35\}$

$B = \{5, 9, 16, 26, 36\}$

$\text{medA} = 14$

$\text{medB} = 16$

$\text{medA} < \text{medB}$ (apply step d)

$[14, 22, 35]$ and $[5, 9, 16]$

In above two medA and medB is 22 and 9 respectively

Now $\text{medA} > \text{medB}$ (apply step c)



Subarray becomes $[14, 22]$ and $[9, 16]$

Now, we use the formula to calculate median which we have given in Step f:

$\text{Median} = (\max(14, 9) + \min(22, 16)) / 2$

$= (14 + 16) / 2$

$= 15$

 EXERCISE 9.5.24 

(a) Suppose University High School (UHS) is electing its student-body president. Suppose further that everyone at UHS is a candidate and voters write down the student number of the person they are voting for, rather than checking a box. Let A be an array containing n such votes, that is, student numbers for candidates receiving votes, listed in no particular order. Your job is to determine if one of the candidates got a majority of the votes, that is, more than $n/2$ votes. Describe an $O(n)$ -time algorithm for determining if there is a student number that appears more than $n/2$ times in A .

[Feedback?](#)

Ans: We must utilize the Randomized Quick select technique based on the prune and search paradigm to determine if any student numbers occur more than $n/2$ times in the data set A .

The vote value will be placed in a specific index using the student number as the index.

Apply prune and search after tallying the votes for a certain contender.

1. Prune: Dividing an array A into three sequences by choosing a random element x from an array A with n votes.
2. Lt: substances that contain x or less.
3. Eq: substances similar to x
4. Gt: components bigger than x
5. According to this method, candidates who received less votes will be on the Lt side of x , while those who received more votes will be on the Gt side of x .
6. Search: At this point, we repeatedly use the Gt fast select command, which returns the name of the candidate who received the majority of the vote as the result.

Partitioning all the elements into the three Lt, Eq, and Gt will take prune $O(n)$ time. The second step will likewise take $O(n)$ time since it repeatedly invokes the fast choose method.

Time complexity will thus be $O(n)$ time.