CS 600 HW 4
SHUCHI PARAGBHAI MEHTA
CWID: 20009083

- 7.5.5

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.

When the list-based approach is represented using a linked list, the contents of any set inside a partition can be listed in time proportionate to the size of the set.

let's consider a list for set A, containing a head node that stores the pointers to the first and last nodes of a linked list, which has pointers to every element in set A. Each node of a linked list for A stores a pointer to an element belonging to that set and pointer to the head node for A.

Forming a set using a linked list is equal to performing an $O(n)$ insertion operation on each of the n elements in the list.

- 7.5.9

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.

As given, there are n elements in a single set, such as $\{1\}, \{2\}, \{3\}, \ldots\ldots, \{n\}$. By combining smaller sets into larger sets, perform the UNION operation on the set to reduce the amortized cost. As the walking for all the elements occur from $1\ to\ n$ in an effective way.

Let's consider an element x in the sequence of m union-find operation. Every time the size of the set containing element pointer of x changes, it gets double. It is called extendable array implementation.
The size of the set, containing x after n union will be 2n. Each node's update pointer may be changed a maximum of log n times. Consequently, it will take $O(n\ log\ n)$ to process all n nodes of the union-find data structure. Amortized time per union is $O(log\ n)$. In a union-find structure, In the worst situation, find and make-set operation will take $O(1)$ time, and a sequence of m union operations will take $O(m\ +\ n\ log\ n)$.

- 7.5.21

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?

Using Union-Find data structure, we can detect when a player wins and how many bonus points it will earn. We can determine when a player wins and how many bonus points, they will receive using the Union-Find data structure.

1. First, assign names to each node in the $n \times n$ grid, ranging from 0 to $n^2 - 1$. Create a singleton set n nodes. For instance: {1}, {2}, {3}, ……{n}. Placing stones of color black and white from top to down, starting from the respective corners and left to right.
2. Now, starting from two end points (left-right or top- bottom), if two adjacent cells share the same color (black or white), join them using the union function.
3. To check whether the added cell belongs to same color cell, use the find operation. Additionally, verify that any set that contains a cell on the left boundary is similar to any set that contains a cell on the right boundary (similarly for top-bottom). This is the winning move if this condition is true. If not, carry on with the union-finding.

Conditions can also be examined from top to bottom.
Completing the chain from one end to other (left to right or top to bottom)
Count the number of gold cells (k') in the final set after the chain has been completed; this will be the candidate who receives the most bonus points.

Time Complexity:
An operation to make a set will require $O(n)$ time. It will take $O(n \log n)$ time to perform a union for n elements. Thus, the overall time complexity is $O(n \log n)$.

- 8.5.12

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?

Implementing an in-place (Quick-sort) method to order S, so all the blue components are listed before all the red elements.
Consider the integers l and r, which respectively scan elements in the right and left directions. Increase the value for l until it gets the red element if it is at the blue element.
Decrease the value of r till it reaches the blue element, if it is at the red element.
Swap the elements $S[l]$ and $S[r]$ when l reaches the red element and r reaches the blue element. Call this method repeatedly until l and r are in the same location.

Yes, we can extend this approach to three colors. The problem can be solved for three colors by repeating the above approach twice. First, we can find a solution for a single color by shifting one color to the left of the array and switching the other two colors at the right. Applying the same method once more to the other two colors that were shifted to the right Starting point for this will be the rightmost index of the array, where L and R will be the finishing points of the sorted color. Then, use the similar approach as mentioned above.

- 8.5.22

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.

As given, we have sequence of n elements called S, where each element in S represents a different vote in an election and each vote is given as an integer representing the ID of chosen candidate.

1. Using quicksort or mergesort, as we have to order the elements in S according to the supplied sequence.
2. Go through the list and note how many votes each candidate has received. The candidate who receives the most votes declare IdMax, keeps track of the results, and check with the other candidates on the list.
3. If any other candidate receives more votes than IdMax, the maximum number of votes will be changed to current number of votes.
4. After traversing the list, the candidate with highest number of votes will win the election.

Thus, It will take $O(n\ log\ n)$ time to sort the elements, and $O(n)$ time to traverse n the elements of S. The given algorithm will therefore execute in $O(n\ log\ n)$.

- 8.5.23

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.

1. Balanced search trees can be used to implement the algorithm and store the vote counts in relation to each candidate's ID. where each node stores the value of votes received for a specific ID and each node represents an ID.
2. As we start to traverse the elements in S sequence, we increase the count of the votes for a specific ID from 0 to 1.
3. Go through each node in the tree to identify the ID that has the most votes.

It will take $O(log\ k)$ time to search and update the tree for k items. Additionally, it will take $O(n)$ time to traverse the sequence of n elements in S. Consequently, the algorithm's total running time is $O(n\ log\ k)$.

- 9.5.17

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.

Given: Two sorted list A and B of n elements each, all of which are distinct.

1. To start, find the medians for lists A and B that have been sorted and call them MedianA and MedianB respectively.
2. Return either of them if MedianA and MedianB are equal.
3. Else if MedianA is greater than MedianB, then one of the two subarrays specified below has a median:

   From first element of A to MedianA or from MedianB to last element of B meaning,
   $A[0]\ \ldots\ldots A[n/2]$ or $B[n/2]\ to\ B[n-1]$

4. Else if MedianA is smaller than MedianB, then one of the two subarrays specified below has a median:

   From MedianA to last element of list A or from first element of list B to MedianB meaning,
   $A[n/2]\ \ldots\ldots A[n-1]$ or $B[0]\ \ldots\ldots B[n/2]$

5. Recursively call above process until the size of both the subarrays becomes 2.
6. Use the following formula to calculate the median when size of both the lists equals to 2,

$$Median\ =\ (max(A[0], B[0])\ +\ min\ (A[1], B[1]))\ /\ 2$$

Time Complexity: $O(log\ n)$ for above algorithm.

Example:

$A\ =\ \{2, 11, 17, 26, 35\}$
$B\ =\ \{3, 12, 20, 30, 40\}$
$MedianA\ =\ 17$
$MedianB\ =\ 20$

If $MedianA\ <\ MedianB$ // (apply step 4)
// $([17, 26, 35]\ and\ [3, 12, 20]$

// (In above two MedianA and MedianB is 26 and 12 respectively)
Else if $MedianA\ >\ MedianB$ // (apply step 3)
// (Subarray becomes [17, 26] and [12, 20])

// (Now, checking step no. 6 from above method, as there are 2 elements left)

$$Median\ =\ (max(17, 12)\ +\ min(26, 20))\ /\ 2$$
$$=\ (17\ +\ 20)/2$$
$$=\ 18.5$$

- 9.5.24

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.

We have to utilize the Randomized Quick Select algorithm based on the prune and search paradigm to determine if any student numbers appear more than n/2 times in A.

The vote value will be stored in a specific index using the student number as the index. Apply prune and search after counting the votes for a specific candidate.

Prune:
Dividing A into three sequences by choosing a random element x from an array A with n votes.

L: elements less than x
E: elements equal x
G: elements greater than x

According to this algorithm, candidates who received fewer votes will be on the right side of x and those who received most of the votes will be on the left.

Search:
Therefore, recursively calling quick select on G will return the result for the candidate that received the majority of the vote.

Time Complexity:
This algorithm will execute in $O(n)$. Partitioning all the elements into L, E, and G will take $O(n)$ in the first phase, or prune. The second step will likewise require $O(n)$ time because it calls the quick select repeatedly.