CS2040S Semester 1 2023/2024

Data Structures and Algorithms

Tutorial+Lab 05

Midterm Quiz/First Half Review; UFDS

For Week 07

Document is last modified on: June 28, 2023

MODAL ANSWER IS FOR OUR CLASS ONLY; NOT TO BE DISTRIBUTED IN PUBLIC

1 Introduction and Objective

In the early part of the tutorial component of this session, we will properly discuss the solutions and a few common mistakes that were found during grading over recess week. Then, we will discuss more of the short one-off `https://visualgo.net/en/ufds` in this tutorial. We will do a (much) longer a lab component today (a small warm-up for PE on Week 11).

2 Tutorial 06 Questions

Midterm Quiz Review

Q1). See the midterm quiz solution draft file that has been uploaded at Canvas before this tutorial. TA will do one quick (re-)presentation of the solutions and highlight the common mistakes. TA will open a 5-10m AMA (Ask Me Anything) session about that Quiz to give closure to all.

Nothing here. See the other file.

UFDS Review

Q2). Using `https://visualgo.net/en/ufds`, quickly review the `findSet(i)`, `isSameSet(i, j)`, `unionSet(i, j)` operations of the Union-Find Disjoint Sets (UFDS) data structure.

Just do a very quick one for this easy DS to ensure that almost everyone understand this data structure. Potential concepts that probably need to be strengthened are: path

compressions, the fact that rank values may be wrong after compressions, and how to use rank values to do more efficient union (union-by-rank).

Q3). The basic UFDS data structure can be *augmented* to support extra operations. The first (and easiest) augmentation is to support `numDisjointSets()` query in $O(1)$ (instead of in $O(N)$). When we create a new instance of UFDS, we create N initially disjoint sets. Show how we can carefully track these information throughout various UFDS other operations!

`findSet(i)` and `isSameSet(i, j)` operations do not really change the number of disjoint sets (they may flatten the trees due to path compression). Only `unionSet(i, j)` operation that can *potentially* decrease the number of disjoint sets by *exactly one* if i and j were initially from two disjoint sets before the union. Thus, we can simply create one additional private counter variable, initialized to N during the construction of UFDS, and we decrease this counter whenever we have a successful `unionSet(i, j)` operation. At all times, this counter variable will contain the correct information of the current number of disjoint sets.

Q4). The second (harder, but more versatile) augmentation is to support `sizeOfSet(i)` query in $O(1)$ (instead of in $O(N)$). This query reports the size of set that currently contains item i. Think of how to do this operation quickly and especially if two previously disjoint sets were merged into one!

Again, `findSet(i)` and `isSameSet(i, j)` operations do not really change the size of any disjoint set (they may flatten the trees due to path compression). Again, only `unionSet(i, j)` operation that can *potentially* combine the size of disjoint sets of set that contains item i (let's say x elements) and set that contains item j (let's say y elements — and again, if they were initially from two disjoint sets before the union). The combined set will then have size $x+y$, but where should we store this information? It is simple, we do similar thing as with the rank information, i.e., each root of a disjoint set remembers the size of its tree. After union, the combined set will have a new root (one of the previous two roots). This new root will store that the size of its tree is now $x+y$. PS: This is part of the live demo on Thu, 30 Sep 2022. Steven has actually discussed this, but since only about 1/3 of the class attending, the rest have to review that. Note to TA: The concept of augmenting 'set data' at the parent/representative item of the set can be used for other purposes too.

Hands-on 5

TA will run the second half of this session with a few to do list:

- Review of our-own custom implementation of UFDS, see `https://github.com/stevenhalim/cpbook-code/blob/master/ch2/ourown/unionfind_ds.cpp`
- Do a sample speed run of VisuAlgo online quiz that are applicable so far, e.g., `https://visualgo.net/training?diff=Medium&n=5&tl=5&module=ufds`,

- Finally, live solve TWO chosen Kattis problem involving material from the **first half** of CS2040C (please treat this as a warm-up exercise for the upcoming Practical Exam (PE) that will be harder than this; or for preparation of the application question(s) in Midterm Quiz)

Spend some time exploring `https://github.com/stevenhalim/cpbook-code/blob/master/ch2/ourown/unionfind_ds.cpp`, try to explain that `numDisjointSets()` and `sizeOfSet(i)` have been actually implemented as part of that library template. Ask students if any of them want to clarify any portion of the library code.

Show off in 5m :), again (on another set of random questions).

Do Kattis /speedrun, this is a classic greedy problem that has one key component, sort the input first. Give this major hint to the student (sort by t_{end} first, and if ties by t_{start}) and see if many students can arrive at a simple greedy strategy.
Then do Kattis /annoyedcoworkers, here we show how to use Priority Queue to greedily simulate what is needed (always annoy the most generous friend). This time, it is harder to see the greedy strategy.

Problem Set 4

We will end the tutorial with a **short algorithmic** discussion of PS4.
As we still have Week 08 before PS4 is due, then TAs are not supposed to reveal the algorithmic ideas of the near 100+100 solutions publicly (yet).

Tutorial TA can now discuss in high level on what are required to get 100+100 points for PS4 A+B, respectively, in somewhat vague way.

For PS4A (/swaptosort), we are given a reverse sorted array and we want to sort the array into ascending order by swapping them (somewhat similar to PS2B /jobbyte). This time we are not asked to come up with the minimum number of swaps (with no restriction), but a Yes/No question on whether we can sort the array if we can only swap certain pairs of indices. To solve this problem, we need to first observe that two indices (a, b) can always be swapped if there is a chain of pairs connecting them, i.e., pair (a, b) directly, pair (a, c) - (c, b), pair (a, c) - (c, d) - (d, b), etc. However, we cannot do an $O(N^2)$ algorithm. So what data structure that we have just learned that can help us to significantly speed up the solution?

For PS4B (/kaploeb), you will need to go through Hash Table data structure first to get a fast solution. So the early hint: understand Week 07 lecture material first before doing this problem.