Design Decisions and BFS

Due date: Friday May 8, 2020 at 11:59 pm

Instructions: Submit a typed or neatly handwritten scan of your responses to the "Exercise 3" assignment on Gradescope here: https://www.gradescope.com/courses/97095.

1. Design decisions: Evaluating designs with runtimes

In this problem we're going to analyze the runtimes of various data structures under several scenarios. Being able to analyze your options in terms of runtime is one of the most important parts in being able to make design decisions.

Imagine we want to store the data from the CSE 373 Zoom's participant panel – in particular meeting attendees and their current reaction status (thumbs-up, thumbs-down, clapping, etc.). Consider the following scenarios that we want to optimize for:

- Scenario 1: ~~Kasey wants to be able to view (edit 5/3: get a collection of) all the students who are currently reacting with the thumbs-up.~~ Kasey wants to be able to obtain a LinkedList of all the students who are currently reacting with thumbs-up. Note: We're clarifying some ambiguity, but if you interpreted this previously as needing to create a linked-list (or collection) and moving these values to that data structure, we will also accept that as a correct interpretation and grade the runtime accordingly.
- Scenario 2: Zach wants to know the current reaction status of his favorite student (which is you, the reader of course!) so he can check on them in particular.
- Scenario 3: Dubs the Husky wants to make everyone happy and wants to change everyone's current reaction status to thumbs-up. (e.g. people who are thumbs-down will be forced to thumbs-up)

For each of the above scenarios, we will also consider each of the following data structures:

- `HashMap<Person, ReactionType>`
  - Example of a key-value pair: Naveena $\to$ Clapping
- `HashMap<ReactionType, LinkedList<Person> >`
  - Example of a key-value pair: Clapping $\to$ [Naveena, Xin, Sasha]
- `HashMap<ReactionType, HashMap<Person, Boolean> >`
  - Example of a key-value pair: Clapping $\to$ {Naveena $\to$ true, Xin $\to$ true, Sasha $\to$ true, Esteban $\to$ false, Alex $\to$ false, ...}. For this data structure, each `ReactionType` key is associated with an inner `HashMap` where a key represents a `Person` and the value represents whether or not that `Person` is currently using the given outer `ReactionType`. Note: each inner `HashMap` contains a mapping of every `Person` in the class to their `Boolean`.
- `MaxArrayPriorityQueue<Student>`
  - A `Student` object is prioritized by how much Zach likes them (Zach's favorite is the max of the PQ)

We use `Person` and `ReactionType` types above, but they're nothing too complicated; you can think of them as basically `String`s, or read the following definitions for more clarity.

```
public class Person {
    public String name;
}
```

```
public class ReactionType {
   public String reaction;
}
```

```
public class Student {
   public Person person;
   public ReactionType reaction;

   // how much Zach likes this Student
   public int favoriteValue;
}
```

For each of the possible scenario and data structure combinations, ~~think of the most efficient algorithm to solve the task~~ (Edit: 5/3) describe the most efficient algorithm to solve the task by explaining what your algorithm is and how it uses the given data structure (what methods are called) in 1-2 sentences. Then give a big-Theta bound for its worst-case runtime.

- Assume that any hash-related activities have well-defined hash functions and minimal collisions and that any array-based structures will not require resizing when added to.
- Assume that you can use every data structure's iterator to loop through all elements in linear time in terms of that data structure's size.
- Assume you don't have access to the student or reaction type data in a format outside of the current data structure.
- Don't worry about how long it takes to create the data structures and only analyze the runtimes of the scenarios stated.

Your big-Theta bounds will be in terms of the variables P (the number of `Person` objects) and/or R (the number of possible different reaction types) or $\Theta(1)$. Note that we do not need a variable for the number of `Student` objects because there are the same number of `Student` objects as there are `Person` objects. If you want to read more on multi-variable asymptotic analysis, check out this additional resource: https://courses.cs.washington.edu/courses/cse373/19su/files/resources/multivariableBigO.pdf

(a) `HashMap<Person, ReactionType>`

- Scenario 1 worst-case runtime: $\Theta(\_\_\_\_\_\_)$
- Scenario 2 worst-case runtime: $\Theta(\_\_\_\_\_\_)$
- Scenario 3 worst-case runtime: $\Theta(\_\_\_\_\_\_)$

(b) `HashMap<ReactionType, LinkedList<Person> >`

- Scenario 1 worst-case runtime: $\Theta(\_\_\_\_\_\_)$
- Scenario 2 worst-case runtime: $\Theta(\_\_\_\_\_\_)$
- Scenario 3 worst-case runtime: $\Theta(\_\_\_\_\_\_)$

(c) `HashMap<ReactionType, HashMap<Person, Boolean> >`

- Scenario 1 worst-case runtime: $\Theta(\_\_\_\_\_\_)$
- Scenario 2 worst-case runtime: $\Theta(\_\_\_\_\_\_)$
- Scenario 3 worst-case runtime: $\Theta(\_\_\_\_\_\_)$

(d) `MaxArrayPriorityQueue<Student>`

- Scenario 1 worst-case runtime: $\Theta(\_\_\_\_\_\_)$
- Scenario 2 worst-case runtime: $\Theta(\_\_\_\_\_\_)$
- Scenario 3 worst-case runtime: $\Theta(\_\_\_\_\_\_)$

(e) State which scenario (1, 2, or 3) you think will occur most frequently and then state the data structure type that has the best worst-case runtime for that scenario.

2. Implementing BFS and interpreting results

In this problem, you'll implement BFS with a small modification and answer some high-level questions about the outputs. In doing so, you'll get some more practice thinking about data represented as a graph as well as working with graphs in code. Additionally, you'll get to practice turning pseudocode into real code that fits your context and data types. This is a common situation in programming: you have a reference point for a similar problem and solution, but there are technical formatting issues that don't allow you to use the reference solution directly and instead require you to adapt it.

2.1. Context:

In the current global state of quarantine, the New York City (NYC) community started a trend in March in which people clap, bang pots and pans, and make noise at 7pm each day to cheer and show support for essential workers. You could imagine that the way this is coordinated is that one person starts it off by making some loud noise. Soon, nearby neighbors who can hear that noise will do their part and make noise. This process repeats and the sound wave will keep spreading. This is the process that we're going to model our graph computation after – we're going to use BFS to simulate this sound wave spreading across NYC.

For our graph, each vertex represents a building and each edge that exists between buildings A and B represents that the clapping from building A is audible in building B and vise-versa.

The graph that you'll be working with is from the following dataset: `https://data.cityofnewyork.us/City-Government/NYC-Address-Points/g6pj-hd8k` which contains close to 1 million NYC building address data points. By default, the graph you'll be working with will contain a subset of the data, specifically focusing on ZIP code 10128.

Disclaimer: The actual way edges were constructed between vertices was based on distance as an approximation of sound, though we could have tried to do something more clever like consider if the buildings were across a noisy street vs on the same side of the street. This overall setup isn't the most realistic scenario, but it can still give some satisfying answers to our hypothetical curiosities.

2.2. Instructions:

Visit the repo here: `https://gitlab.cs.washington.edu/cse373-20sp-students/exercise-3` and clone it to get started. See `https://courses.cs.washington.edu/courses/cse373/20sp/projects/cse143review/setup/#intellij` for a reminder on how to do this. The `BFS.java` file is where you'll implement your BFS and do any extra analysis on the output. The other provided source file, `LoadAddressData.java`, is where we load the data and convert it to a graph format. If you want to optionally explore the data beyond the questions asked below, this is where you can make changes to the ZIP code or more generally choose what information gets included in the graph. Note that there are also tests for your BFS implementation located in `BFSTests.java`.

As mentioned in lecture, BFS and DFS by themselves don't do anything since they're just standard ways of traversing graphs. So to get useful information out of the BFS, you'll make a modification to record how many levels away each address is from the source (i.e. the starting source of the sound wave is 0 distance, the buildings one edge away are 1 distance, the next level out is 2 distance, etc.). Your BFS implementation will return a `Map<String, Integer>` where the keys represent the building addresses and the values represent the associated distance. We recommend you use either slide 25 or slide 39 of `https://courses.cs.washington.edu/courses/cse373/20sp/lectures/14-bfs-dfs/14-bfs-dfs.pdf` as your starting point for implementing BFS.

2.3. Questions:

Answer these questions after implementing BFS. You should play around with your returned output in the main method to do some more investigation to answer the following questions. It's possible to obtain the answers to these questions with a few lines of code for each question.

Note: you do not need to turn in your code to Gradescope – just the answers to these questions.

(a) Which address(s) hears the clapping started by `212 E  95 ST, NY` last/the latest? If there are multiple addresses, then state all of them.

(b) Yes or No: are there any addresses in the 10128 Zip code that the BFS sound wave is unable to reach when we start from `212 E  95 ST, NY`?

(c) (Optional) If you have time, feel free to explore the data some more; you could look up some of the points on Google Maps (copy-paste the addresses), or explore different ZIP codes, or explore the meaning of your BFS output more. Is there anything else you learned about the data while exploring that you want to share?