Introduction to Java

CS9053

Thursday 6 PM – 8:30 PM

Prof. Dean Christakos

Assignment 7

July 19th, 2023

Due: July 26th, 2024 11:55 PM

**Part I: Least Recently Used Cache**

1. You’re going to implement an LRU cache.

A cache is a key -> value store, similar to a Map. However, LRU Caches have limits, and the least recently used element should be removed.

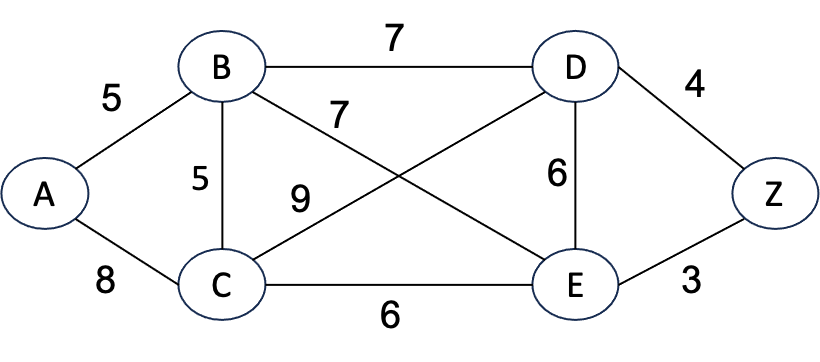
How do we keep track of the Least Recently Used element? It means whenever we access the cache, the element that we just accessed goes to the end of a queue, and when we add a new element to the cache at its current limit, we remove whatever element is at the head of the queue.

(Assume for object **O** in the cache, the “cache queue” can be removed in O(1) time by using remove(**O**) and added to the end of the queue in O(1) using offer(**O**) and removed from the head of the queue in O(1) using remove()).

Assume you have a key value of Integer. The value can be any other kind of Object.

**Part II: Shortest path in a graph**

1. Here, I am going to give you a graph where you have to find the shortest path between node A and Z. The algorithm itself will be pretty straightforward, but the challenge is that you need to come up with the right Collection classes to represent the graph and to solve the problem as quickly as possible.



You will have to represent this graph in some way. My suggestion is to create an adjacency list that stores the edge weights and pick the correct Collections to implement an adjacency list.

To find the shortest path between A and Z, you will probably want to implement Dijkstra’s algorithm using a Breadth First Search (BFS). BFS means when you discover a node, you put the node one end of a list and take it from the other, and you have to do this in O(1) for each operation. There are a couple options about what data structure to use for this, but an ArrayList is not one of them, so don’t use that. You will also need a means of keeping track of which nodes you’ve already visited.

You can find Dijkstra’s algorithm described here: <https://www.geeksforgeeks.org/dijkstras-shortest-path-algorithm-greedy-algo-7/>

You will return the shortest from A to Z and the length. Obviously, a quick survey of the graph should make it clear what the correct answer is so it should be easy to check what the answer is.

**Part II: Sets**

The reason we like Sets in Java is because they help us think about Sets in a mathematical sense and we can easily implement the functions of Sets that exist in Math—eg, Set intersections and unions. In Python, these set functions are explicit. In Java, they are not, something which I briefly forgot during lecture.

Create a class MathSet which extends HashSet. It should have three methods:

public Set intersection(Set s2): Takes a Set, s2, and returns the intersection of the Set and s2—the elements that are in both sets.

public Set union(Set s2): Takes a Set, s2, and returns the union of the Set and s2—the combination of all elements.

public Set<Pair<T,S>> cartesianProduct(Set s2)

I have provided a Pair class for this. Return the Cartesian Product of the base set, s and s2: s × s2:

A **Cartesian product** of two sets A and B, written as A×B, is the set containing **ordered** pairs from A and B. That is, if C=A×B, then each element of C is of the form (x,y) where x∈A and y∈B:

A×B={(x,y)|x∈A and y∈B}.

For example, if A={1,2,3} and B={H,T}, then

A×B={(1,H),(1,T),(2,H),(2,T),(3,H),(3,T)}

Note that here the pairs are ordered, so for example, (1,H)≠(H,1). Thus A×B is **not** the same as B×A.