Madison Smith
Chloe Lathe
CSE 373
22 February 2017

Homework 4 Write-up

1. Describe the worst-case asymptotic running times of your methods adjacentVertices and edgeCost. In your answers, use |E| for the number of edges and |V| for the number of vertices. Explain and justify your answers.

adjacentVertices-

The worst-case asymtotic running time for my adjacentVertices method is O(|V|). In my method, there's a for loop that goes through all the Vertex objects and checks that if that vertex is adjacent to the given vertex. That for loop has a cost of O(|V|). Since all the other parts of the method are O(1) (the map I used was a hashMap, so the run time for .get() is O(1) and I used an array to implement the matrix, so the run time to find a spot in the matrix is roughly O(1)) the worst case asymptotic running time of my adjacent method is O(1).

Assign and create a new HashSet- O(1)
Test if given vertex is exists- O(1)
    Throw exception- O(1)
For loop through all vertices – **O(|V|)**
    Get value in hashMap and assign value – O(1)
    Get object in matrix- O(1)
    Test if object is null- O(1)
        Create new vertex and add to collection – O(1)
Return collection – O(1)

edgeCost-

The worst case asymptotic running times for my edgeCost method is O(1). My edgeCost method is just a series of O(1) actions, and since constants do not matter when considering asymptotic running times, the worst case running time for this method O(1).

Test if parameters are null – O(1)
    Throw exception – O(1)
Create and assign new vertex – O(1)
Create and assign new vertex – O(1)
Get and assign value from hashmap – O(1)
Get and assign value from hashmap – O(1)
Test if there's an edge in a location of the hashMap – O(1)
    Find and return weight – O(1)
Return -1 – O(1)

2. Describe how your code ensures the graph abstraction is protected. If your code makes extra copies of objects for (only) this purpose, explain why. If not, explain why it is not necessary.

My code ensures the graph abstraction by copying all parameters and objects I return that are mutable (e.g. collections, vertices, edges) down to the immutable components. For example, when I copied the collections, I copied the vertices/edges in that collection. I do not have to copy strings and ints within those vertices/edges because they are declared private final. Since the getSource and getDestination vertices in an Edge object are also immutable (they are also declared as private final fields), then those vertices are immutable and I do not have to copy those vertices either.

3. Describe how you tested your code.

I tested my code by first creating a normal graph. I wrote down a graph in a notebook and made the same graph through my code. I made sure the adjacentVertices and edgeCost methods gave me the same answers as the written graph and that all the vertices and edges in my notebook showed up when I printed the vertices and edges methods. Then I tried testing weird cases. I tried passing the MyGraph class an empty collection of vertices and made sure the proper exception was thrown. I tried passing the MyGraph class an empty collection of edges and made sure there was no error. I made sure that if I passed the constructor a set of vertices with duplicates that it would ignore the duplicates. I made sure that an error was thrown if I added two edges from the same source vertex to the same destination vertex but different weights. I tried passing in parameters that did not exist in the graph and made sure the proper errors were thrown.

Then I tested my abstraction. I tried adding and removing vertices and edges from my sets after I made my graph and made sure these changes were not reflected in the graph. I tried switching pointer from one vertex/edge to another vertex/edge after making my graph and made sure these changes were not reflected in the graph. Then I tried to mess with the collections returned in the vertices, edges, and adjacentVertices methods. I tried adding and removing objects from these collections and made sure that these changes were not reflected in the graph.