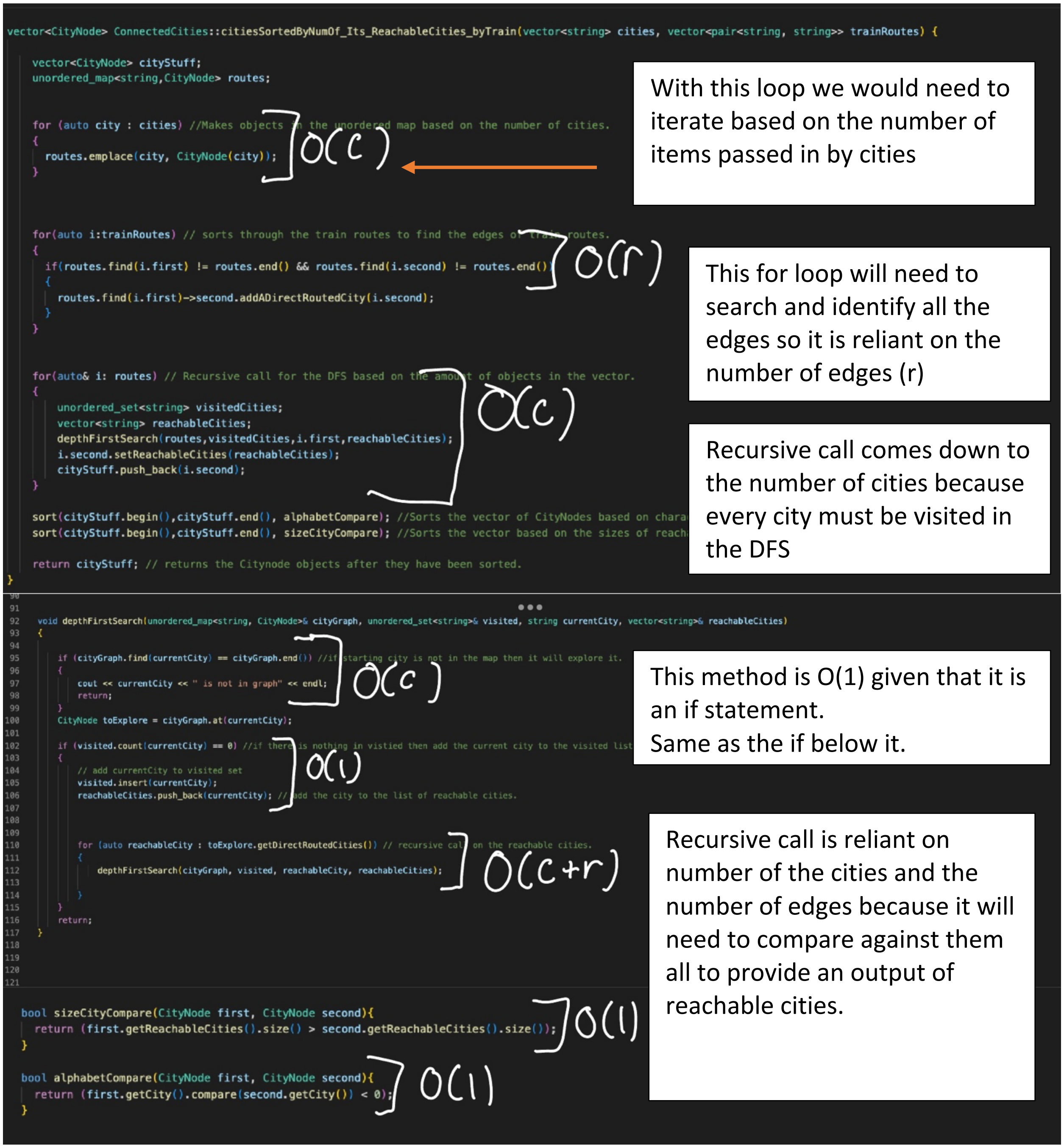
Jacob Holloway
827294826 CS 210 Data Structures 11/22/2022

Assignment Four Complexity Analysis



```
vector<CityNode> ConnectedCities::citiesSortedByNumOf_Its_ReachableCities_byTrain(vector<string> cities, vector<pair<string, string>> trainRoutes) {

    vector<CityNode> cityStuff;
    unordered_map<string,CityNode> routes;


    for (auto city : cities) //Makes objects in the unordered map based on the number of cities.
    {
      routes.emplace(city, CityNode(city));
    }


    for(auto i:trainRoutes) // sorts through the train routes to find the edges of train routes.
    {
      if(routes.find(i.first) != routes.end() && routes.find(i.second) != routes.end())
      {
        routes.find(i.first)->second.addADirectRoutedCity(i.second);
      }
    }


    for(auto& i: routes) // Recursive call for the DFS based on the amount of objects in the vector.
    {
        unordered_set<string> visitedCities;
        vector<string> reachableCities;
        depthFirstSearch(routes,visitedCities,i.first,reachableCities);
        i.second.setReachableCities(reachableCities);
        cityStuff.push_back(i.second);
    }

    sort(cityStuff.begin(),cityStuff.end(), alphabetCompare); //Sorts the vector of CityNodes based on chara
    sort(cityStuff.begin(),cityStuff.end(), sizeCityCompare); //Sorts the vector based on the sizes of reach

    return cityStuff; // returns the Citynode objects after they have been sorted.
}
```

```
void depthFirstSearch(unordered_map<string, CityNode>& cityGraph, unordered_set<string>& visited, string currentCity, vector<string>& reachableCities)
{

    if (cityGraph.find(currentCity) == cityGraph.end()) //if starting city is not in the map then it will explore it.
    {
        cout << currentCity << " is not in graph" << endl;
        return;
    }
    CityNode toExplore = cityGraph.at(currentCity);

    if (visited.count(currentCity) == 0) //if there is nothing in vistied then add the current city to the visited list
    {
        // add currentCity to visited set
        visited.insert(currentCity);
        reachableCities.push_back(currentCity); // add the city to the list of reachable cities.


        for (auto reachableCity : toExplore.getDirectRoutedCities()) // recursive call on the reachable cities.
        {
            depthFirstSearch(cityGraph, visited, reachableCity, reachableCities);

        }
    }
    return;
}
```

```
bool sizeCityCompare(CityNode first, CityNode second){
  return (first.getReachableCities().size() > second.getReachableCities().size());
}

bool alphabetCompare(CityNode first, CityNode second){
  return (first.getCity().compare(second.getCity()) < 0);
}
```

With this loop we would need to iterate based on the number of items passed in by cities

This for loop will need to search and identify all the edges so it is reliant on the number of edges (r)

Recursive call comes down to the number of cities because every city must be visited in the DFS

This method is O(1) given that it is an if statement.
Same as the if below it.

Recursive call is reliant on number of the cities and the number of edges because it will need to compare against them all to provide an output of reachable cities.

Time Complexity explanation:

For the complexity of my code I believe it is running at O(c^2 + r), with the first statement that will need to run for the amount of cities O(c) to create my unordered map. The second for loop is dependent on the number of edges which will bring it to O(r). The third call is the recursive call which calls on the DFS function. With the recursive call function it need to search the number of cities and the number of the nodes so the O notation will need to add the two together, resulting in O(C + R). The sort functions that are called both have O(nlogN) complexity but given that the overall time space complexity is C^2 + R, I believe that overtakes the complexity of nLogn.

C + C + R = O(C^2 + R) for time space complexity given the worst case scenario.