RYAN FILGAS
CS 163
KARLA FANT
03-08-2021

Program 4 Efficiency Writeup

1. How well did the data structure selected perform for the assigned application?
   This data structure is perhaps not the best to store and retrieve specific information by key value, but it excels in finding paths. This data structure is valuable for finding the list of edges itself, and could be applied to educate decision making with repetitive tasks. Each subtask could be a node, weighted by the amount of time it takes to complete that subtask. The path with the smallest sum between subtasks would be the most efficient way to complete the larger task. I could see this being really useful for machine learning, and predictive algorithms.
2. Would a different data structure work better? If so, which one and why...
   To retrieve information like a journal entry, I would use a hash table. In this case, access to data can take A LOT of traversal. Every time a vertex is checked for a visit, every time the next edge is visited, etc. I wouldn't use this structure if I needed fast access to data. A hash table works at o(1), and this gets into exponential territory potentially because we have to check if a node was visited before traversing, return back if so, change another variable etc. It's horrible for efficiency, but it's very good for organizing data in a very specific way.
3. What was efficient about your design and use of the data structure?
   As soon as I found what I was looking for in each function, I suspended execution. In the case of finding complete and connected graphs, it was stopping as soon as I found a contradiction to those definitions. For identifying a connected graph, as soon as I visited every node, I suspended execution. When identifying if a graph was complete, I suspended execution as soon as I found a vertex without all of the edges. When I found my data in the last function I returned up the stack.
4. What was not efficient?
   Finding the index of my vertex for every recursive call wasn't efficient. I don't know another way around this problem other than storing the index in the vertex, but I didn't know if we were allowed to do that. I did a lot of checking for when the program had found what it was looking for, and I do wonder if there Is a more clever solution. How I kept track of visit flags may not have been the most memory efficient in the complete function because I had a local array of int flags for every recursive call. The last inefficient thing, was not being able to add a class member to keep track of the number of entries as they were added, and not being able to add other information into the nodes. There's nothing else I can think of to do, other than to jam through the array counting entries.

RYAN FILGAS
CS 163
KARLA FANT
03-08-2021

5. What would you do differently if you had more time to solve the problem?
   If I had more time to solve the problem, I would spend some time really checking my conditions. In all cases, I wasn't able to find a clever way to use the return, and still maintain most of the functionality inside the recursive inner function. The function only returns true or false, which is expected, but normally I like to return something else. In this case for the depth first search for a connected node, I could return the path taken by reference as an edge list. This would allow for direct traversal to access that data without going through the efficiency suck of the depth first search again. I could have used less recursion and a little more iteration in some cases.