CS2040C Semester 2 2018/2019

Data Structures and Algorithms

Tutorial 09 - Graph Traversal

For Week 11

Document is last modified on: March 1, 2019

1 Introduction and Objective

Now that we have stored our graphs in one (or more – by now you should realize that you can do this) graph data structure(s), we want to run various (graph) algorithms on it.

In this tutorial, we will focus on two graph traversal algorithms: Depth-First Search (DFS) and Breadth-First Search (BFS) and concentrate on what they can do on top of just traversing the underlying graph.

We will heavily use `https://visualgo.net/en/dfsbfs` in this tutorial.

2 Tutorial 09 Questions

Review Harder Topics

Q1. Tutor will spend some time (depending on the requests) to review any remaining harder topics about graph traversal that may not be clear even after the flipped classroom on Week 09+10. In recent years, these are the usually harder topics for students, in decreasing order of difficulty:

1. `https://visualgo.net/en/dfsbfs?slide=7-1` to 7-3 (about back edge/detecting cycle in the graph; we will review this if majority still have difficulty)
2. `https://visualgo.net/en/dfsbfs?slide=7-10` to 7-11 (toposort, revisited in Q3+Q4 below)
3. `https://visualgo.net/en/dfsbfs?slide=7-6` to 7-9 (should be clearer this time, but check your understanding about the $O(V \times (V + E))$ versus just $O(V + E)$ analysis again)
4. `https://visualgo.net/en/dfsbfs?slide=8` (all other more advanced graph traversal topics that are not the main focus of CS2040/C are optional and such questions will be answered **offline**, after/outside class and will not be part of CS2040/C final assessment, especially during the shorter Special Semester 4 final assessment...)

DFS/BFS Applications

Q2. Back in Week 10, we have discussed:
`https://nus.kattis.com/problems/countingstars` (easy floodfill/finding connected components),
`https://nus.kattis.com/problems/reachableroads` (easy DFS/BFS application; finding CCs too),
`https://nus.kattis.com/problems/runningmom` (back edge/cycle detection problem),

Tutor will ask the class on which problem is currently still hard/unclear and we will spend some time clearing the remaining doubts, at least in algorithmic level. However, in the event that nobody in class asked anything, then tutor will skip this Q2). and may want to open a new question Q5). :O...

Deeper Stuffs about Topological Sort

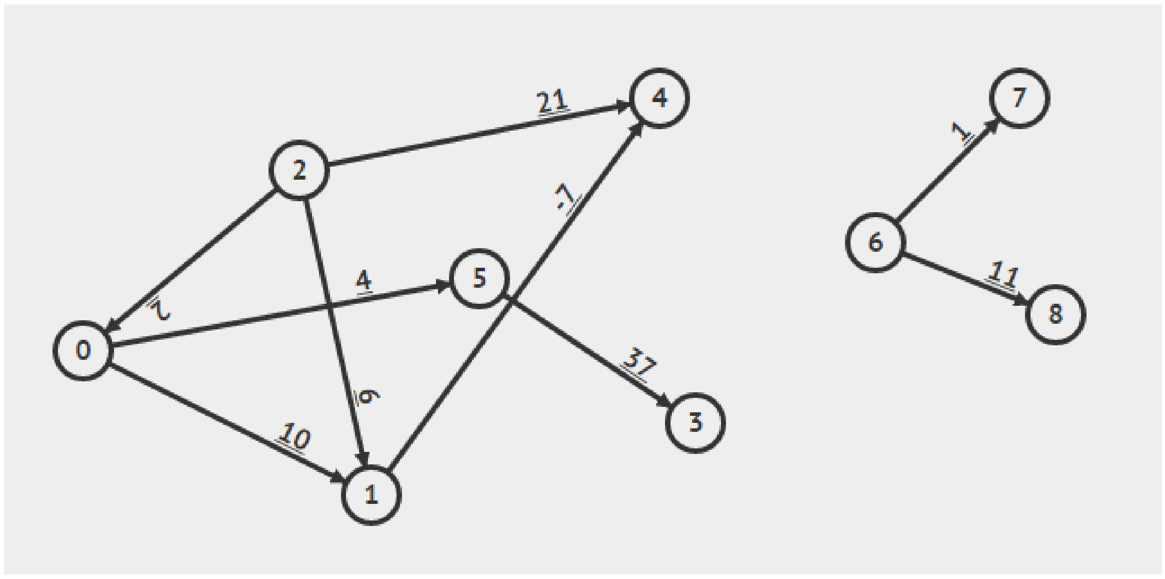

Figure 1: A Sample DAG (ignore edge weights for this question)

Q3. The modified DFS or modified BFS (Kahn's) topological sort algorithm given in class (please review `https://visualgo.net/en/dfsbfs`, 'topological sort', either the DFS or BFS version) only gives *one* valid topological ordering. How can we find **all** possible valid topological orderings for a given DAG? For example, there are **1 008** possible valid topological orderings of the DAG in Figure 1. Starting point: What kind of DAG has the smallest/largest number of possible valid topological ordering, respectively?

Q4. The modified BFS (Kahn's) topological sort algorithm is actually quite interesting (read the details at `https://en.wikipedia.org/wiki/Topological_sorting#Kahn's_algorithm`). Can we change the underlying data structure (from a queue that is used in the modified BFS @ VisuAlgo) into another data structure? What if we replace the queue with a stack (`std::stack`)? What if we replace the queue with a priority queue (`std::priority_queue` or `std::set` – hopefully you now remember that this is also a valid ADT Priority Queue implementation)? What if we replace the queue with a hash table (`std::unordered_set`)?

Graph Modeling Exercise Part 2

Q5). If time permitting (if the clock is still around 35m into the tutorial because Q2). is skipped), then the tutor will open Q5). and do 'live algorithm-level solve' for that (CS2040/C-level) problem.