IT5003 Oct-Dec 2023

Data Structures and Algorithms

# Tutorial+Lab 07
# Graph Data Structures (Fast Recap), Graph Traversal, and Shortest Paths (Part 1)

Document is last modified on: November 17, 2023

## 1 Introduction and Objective

There was no Lab 06 last week, so this Lab will feel packed...

In this penultimate lab session, we will quickly recap graph data structures, review graph traversal topics, and start discussing the last topic for this module: Single-Source Shortest Paths (SSSP) problem that has just been introduced recently (only the unweighted version first). We will start talking about the 'graph modeling' soft skill, i.e., ability to model a seemingly random (non-explicit-graph) problem into a graph problem.

We use `https://visualgo.net/en/graphds`, `https://visualgo.net/en/dfsbfs`, and `https://visualgo.net/en/sssp` (limited to BFS first) during our discussion in this session.

During the hands-on session, we will spend some time discussing one of the hardest Graph Traversal topic discussed in IT5003: topological sorting (using either DFS or BFS modification), `https://visualgo.net/en/dfsbfs?slide=7-10` to 7-11 (toposort).

## 2 Questions

### Graph Data Structures Open QnA

We will start this session by checking if anyone still has issues with the material about graph that has been presented so far (graph data structures: AM/AL/EL/new: implicit).

This part is left to the tutor but should not take too much time especially as the scope is narrower for IT5003. Due to not having Lab 6, some students may still be '1 week behind', i.e., still not fully clear with graph data structures topic. Discuss these:

1. Graph DS conversion, i.e., given an AM—AL—EL (usually because on its original form, we cannot do certain things, e.g., in AM—AL, we cannot sort the edges), convert it into an another graph DS (so we can do what we want to do, efficiently).

2. Drawing certain (special) graphs with certain constraints, e.g., Draw a Directed Acyclic Graph (DAG) with $V$ vertices and $V \times (V-1)/2$ directed edges (Just draw a line containing $V$ vertices labeled from 0 to $V-1$. Then draw directed edge from a vertex $i$ to a vertex $j$ for all pairs of $(i, j)$ where $i < j$), Draw a Bipartite Graph with $V$ vertices (assume that $V$ is even) and $V^2/4$ undirected edges (Put $\frac{V}{2}$ vertices on the left and the other $\frac{V}{2}$ vertices on the right and link them all. This is the configuration that yield the most edges).

3. The new concept of implicit graph, i.e., to not actually storing the graph connectivity information in an explicit data structure (AM/AL/EL), but just generate the edges on the fly, e.g., one example has been shown so far on grid graph: `https://nus.kattis.com/problems/amoebas` (7a), and there may be more in the future.

### Graph Traversal Open QnA

We continue with QnA on graph traversal algorithms: basic DFS/basic BFS.

These are some of the key 'troublesome concepts' to be highlighted (from historical records):

1. Reachability test, printing the traversal path, identifying/labeling/counting CCs (on standard graph or on 2D grid graph) should be OK for most iterations of IT5003,

2. Cycle check perhaps can still be problematic for some. Re-explain the need for using three DFS visitation states (UNVISITED, **EXPLORED**, and VISITED) one more time,

3. Topological sorting usually explained in a rush in IT5003 Lecture 7a, and thus will be repeated one more time in the hands-on section of this lab.

### SSSP Open QnA

During your self-study via VisuAlgo e-Lecture and in real life class discussions, you were presented with these SSSP algorithms: BFS (only for unweighted graph) — already discussed by this tutorial and Modified Dijkstra's algorithm (for non-negative weighted graph) — will be discussed next week. We skipped negative weight cases in IT5003 (moved to optional topics in the last recitation r8)

To start this review, the tutor will (re-)demonstrate the executions of BFS algorithms on a small (un)directed unweighted graph using `https://visualgo.net/en/sssp` from a certain source vertex $s$. The tutor will re-explain why BFS can still be used for this variant. The tutor may invite some students to do this live demonstration using different source vertex $s$ and/or using different graph.

For this session, just focus on BFS first.

1. BFS will only work if the graph is unweighted, or... all edges have the same **positive** constant weight $c$ that can be transformed back to edges with weight 1 by dividing all edges with that same constant weight $c$ (not many realize this, try to emphasize this idea). Note that if the graph contains negative constant weight $-c$, we may NOT be able to do this transformation (either wrong answer as $k \times -c$ (that uses more edges) is smaller than $-c$ for any positive integer $k$, or ill-defined if there is at least one negative weight cycle present). PS: If the graph contains zero constant weight, the SSSP problem becomes weird as it degenerates to reachability test, 0 for all vertices reachable from $s$ or +Inf otherwise.

## Graph Modeling Exercise, via Past Paper Discussion

There are a few graph questions in past final assessment papers. Let's discuss one of them (considering that SSSP will be asked in Final Assessment – but usually not going to be the hardest question).

Read `https://www.comp.nus.edu.sg/~stevenha/cs2040/tests/CS2010-2013-14-S1-final.pdf`, Question 4.1, Facebook Privacy Setting. Is this an unweighted SSSP problem solvable with BFS?

Two trivial cases of Facebook Privacy Setting are as follows:
1). $i == j$, we answer true; and
2). $i$ is a friend of $j$, we also answer true.

For the non trivial cases, we can do a check whether $j$ is inside the list of friends of friends of $i$. This can be done using a **depth 2 BFS check** (SSSP on unweighted graph). However, implementing this idea can be slow as $V + E$ is big.

A better way is to realize that AdjList[i] and AdjList[j] are already sorted. We can then reduce this whole problem into a set intersection problem like Kattis (/judging) discussed long ago :). We can use modified merge sort :O to answer if $i$ and $j$ has a common friend (degree 2) which means that $i$ and $j$ are connected. So this is actually a merge-sort question :O...

### Hands-on 7

TA will run the second half of this session with a few to do list:

- PS5+6 Quick Debrief,
- Do a sample speed run of VisuAlgo online quiz that are applicable so far, e.g., `https://visualgo.net/training?diff=Medium&n=5&tl=0&module=graphds,dfsbfs`.
  PS: Skip parts that are skipped for this sem's IT5003.
- Then, live solve another chosen Kattis problem involving Graph Traversal (toposort).

Do PS5+6 quick debrief according to the context of your lab group.

Do VisuAlgo Online Quiz sample run in medium setting.
VA OQ sample run should skip topics that are not included in IT5003.

SSSP will be part of VA OQ 3.

`https://nus.kattis.com/problems/builddeps`.

Kattis /builddeps is one of the simplest topological sorting problem in Kattis. We store the graph in an AL (of strings, perhaps first time seeing this for some students), then run modified DFS (to get toposort) from changed file. The only concern is Python's low default recursion limit. So as usual, we use `sys.setrecursionlimit(1000000)`.

### Problem Set 7

We will end the tutorial with high level discussion of PS7 A+B.

PS7 A (/conquest) is an interesting graph traversal problem where instead of using an implicit stack (DFS) or an explicit queue (BFS), we use... a priority queue instead. This is so that we can find the smallest-size island $u$ that is connected with the current graph traversal spanning tree rooted from island 1 (the starting point). We have to do this kind of greedy strategy in order to conquer as many islands as possible. We can show via an exchange argument that even if there are other optimal solution that does not conquer island $u$ (e.g., it conquer island $z$ first) while our greedy algorithm is trying to conquer the smallest-size island $u$ connected to island 1, exchanging $z$ with $u$ will always be possible and equally optimal.

PS7 B (/escapewallmaria) has an interesting anime/manga background (the story is very gory though). Again, the traversal is done on a 2D grid, using the familiar N/E/S/W movement pattern. This time, we start from 'S' and have to avoid all '1' (burned). We can freely step on a cell with a '0' (safe) or U/D/L/R cells that can only be entered via S/N/E/W movement, respectively. TA can use sample input/output 1/2/3 to discuss the requirements. Afterwards, this is just an unweighted SSSP problem solvable with BFS (as Eren wants to reach 't' cell as fast as he can).