CS2040C Semester 1 2018/19

Data Structures and Algorithms

Tutorial 10 - DFS/BFS applications, PS5

For Week 12 (Week Starting 05 November 2018)

Document was last modified on: October 31, 2018

1 Introduction and Objective

Now that we have stored our graphs in one (or more) graph data structure(s), we want to run various 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.

2 Tutorial 10 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 5a+5b. In recent years, these are the usually harder topics for students:

1. `https://visualgo.net/en/dfsbfs?slide=7-1` to 7-3 (about back edge/detecting cycle in the graph; we will focus on this first). To test your understanding, draw a graph such that back-edge checking will be erroneous if two states (unvisited/visited) were used instead of three states (unvisited/explored/visited).

2. `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)

For back edge detection, focus on the reason why we need to differentiate visited versus explored to detect back edge/cycle (the ternary state). The graph that will be errorneous if two states were used instead of three states is: `https://visualgo.net/en/dfsbfs?create={"vl":{"0":{"x":240,"y":120},"1":{"x":280,"y":200},"2":{"x":200,"y":180},"3":{"x":300,"y":260},"4":{"x":160,"y":20},"5":{"x":160,"y":100},"6":{"x":160,"y":260},"7":{"x":80,"y":160}},"el":{"0":{"u":1,"v":3,"w":1},"1":{"u":0,"v":2,"w":1},"2":{"u":0,"v":1,"w":1},"3":{"v":0,"u":4,"w":1},"4":{"v":2,"u":5,"w":1},"5":{"u":4,"v":5,"w":1},"6":{"u":2,"v":6,"w":1},"7":{"u":5,"v":7,"w":1},"8":{"u":7,"v":4,"w":1}}}`.

Ans: O($V + E$) to find number of connected components.
In lecture, we have mentioned that DFS/BFS is O($V + E$) time complexity. The V is the number of vertices that the algorithm visits and E is the number of edges the algorithm used. When executing DFS/BFS on one connected component (c.c.), the time complexity is O($v + e$) where v is the number of vertices in the c.c. and e is the number of edges in the connected component. This is because DFS/BFS only traverses vertices and edges that are connected to the vertex it starts from. (i.e. only the same connected component). By performing DFS/BFS only once for every connected component, the total number of vertices visited by **all** the DFS/BFS **in total** is still V. Similarily, the number of edges used by the algorithm will total up to E in total. Hence, the overall time complexity is still O($V + E$).

DFS/BFS Applications

Q2. Please look at `https://open.kattis.com/problems/countingstars` and try to solve it with either DFS or BFS.

Note to tutors: The answer is in page 137 of CP3.17b.
Show the implicit graph buried inside typical grid-like structure.
The vertices are the cells of the grid and there will be undirected edges between two adjacent cells that are both '-'.
The answer will then be the number of connected components in the graph. This can be counted using either DFS or BFS.

Q3. Please look at `https://www.comp.nus.edu.sg/~stevenha/cs2040c/tests/CS2010-2013-14-S1-final.pdf`, Question 4.1, Facebook Privacy Setting and try to solve it.

Facebook Privacy Setting: Trivial: i == j, true, i is friend of j, true. Then naive check for friends of friends is depth 2 BFS check. This can be slow if V+E is big. A better way is to realize that AdjList[i] and AdjList[j] are sorted. We can then reduce this whole problem into a set intersection problem like Kattis - cd :). We can use modified merge sort to answer if i and j has a common friend (degree 2) which means i and j are connected.

Problem Set 5

Q4. For PS5A, the given graph is in the form of a tree. Note that $V = 1000$ and $Q = 100\,000$. To help you think of a solution, answer the following questions:

1. What is the time complexity if one DFS/BFS/Bellman Ford/Dijkstra's algorithm is performed for each query? Will it run in time?
2. A Single Source Shortest Path algorithm from one vertex will compute the shortest path from one starting vertex to **only one other vertex**. True/False?
3. How many simple paths are there between two vertices of a tree?

1. If we use one DFS/BFS for each query, the time complexity will be O($Q(V + E)$). If we use Bellman Ford and Dijkstra, the time complexity will be O(QVE) and O($Q(ElogV)$) respectively. This will not run in time :O.
2. False, Single Source Shortest Path algorithm will compute shortest paths from one starting vertex to **every other vertex**.
3. There is only one simple path between two vertices of a tree. Hence, that path is also the shortest path and longest path :).

This marks the end of Tutorial 10. Tutor can continue to discuss about PS5 if time permits.