Lab07-Graph

Algorithm: Analysis and Theory (X033533), Xiaofeng Gao, Spring 2018.

∗ If there is any problem, please contact TA Mingding Liao.

*

1. A graph is showed in the Figure 1.

   Please describe the process of BFS and DFS in the graph. The process of BFS and DFS should be started at the node a. Specially, you should visualize the stack of DFS and the queue of BFS. Microsoft Visio is the optional software to visualize it and "Lab07-Stack.vsd" is the example of Visio project.

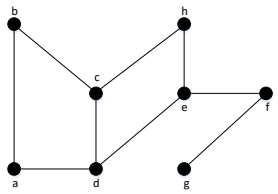


图 1: Graph Example

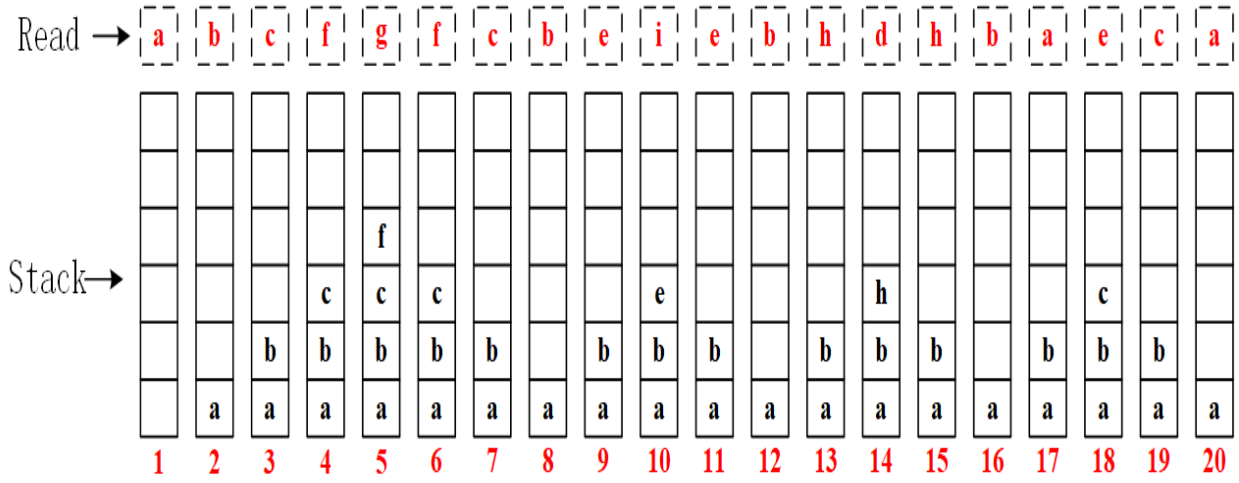
Solution: Common graph algorithms uses a breadth-first approach
Example Problem: Search all nodes for a node containing a given value
Example Problem: Find length of shortest path from node s to all other nodes
Example Problem: Find shortest path from node s to all other nodes

Breadth-first Algorithm:
Depth-first: visit all neighbors before visiting neighbors of your neighbors

Start from node s.
Visit all neighbors of node s.
Then visit all of their neighbors, if not already visited
Continue until all nodes visited

Intuition of Implementation
Keep a queue of nodes to visit
When visiting node u, put neighbors of u on the queue, if neighbor not yet visited
Queue contains all nodes that have been seen, but not yet visited
Queue can be generally thought as horizontal in structure i.e, breadth/width can be attributed to it - BFS, whereas

Stack is visualized as a vertical structure and hence has depth - DFS.
BFS explores/processes the closest vertices first and then moves outwards away from the source. Given this, you want to use a data structure that when queried gives you the oldest element, based on the order they were inserted. A queue is what you need in this case since it is first-in-first-out(FIFO). Whereas a DFS explores as far as possible along each branch first and then bracktracks. For this, a stack works better since it is LIFO(last-in-first-out)



https://www.ics.uci.edu/~eppstein/161/960215.html https://stackoverflow.com/questions/3929079/how-can-i-remember-which-data-structures-are-used-by-dfs-and-bfs

2. For a weighted graph $G = (V, E)$, let $G_w = (V, \{e \in E \mid c(e) \leq w\})$. That is, G_w is the spanning subgraph of G consisting of all edges of weight at most w. Let T be the minimal spanning tree of G and let $w \in R$. Prove that T_w and G_w has exactly the same connected components. (That is, two vertices $u, v \in V$ are connected in T_w if and only if they are connected in G_w).

Solution: For a connected undirected graph G = (V, E), a spanning tree is a tree Note that a spanning tree of a graph G is a subgraph of G that spans the graph (includes all its vertices). A graph can have many spanning trees, but all have jV j vertices and jV j 鋆 edges. In a directed graph G=(V,E), two nodes u and v are strongly connected if and only if there is a path from u to v and a path from v to u. The strongly connected relation is an equivalence relation. Its equivalence classes are the strongly connected components.
Each directed acyclic graph has at least one source node and at least one sink node.
Given an undirected and connected graph G = (V, E), a spanning tree of the graph is a tree that spans G (that is, it includes every vertex of) and is a subgraph of G (every edge in the tree belongs to G)
The cost of the spanning tree is the sum of the weights of all the edges in the tree. There can be many spanning trees. Minimum spanning tree is the spanning tree where the cost is minimum among all the spanning trees. There also can be many minimum spanning trees.

Minimum spanning tree has direct application in the design of networks. It is used in algorithms approximating the travelling salesman problem, multi-terminal minimum cut problem and minimum-cost weighted perfect matching.
If each edge has a distinct weight then there will be only one, unique minimum spanning tree. This is true in many realistic situations, such as the telecommunications company example above, where it's unlikely any two paths have exactly the same cost. This generalizes to spanning forests as well.

Proof:

Assume the contrary, that there are two different MSTs A and B. Since A and B differ despite containing the same nodes, there is at least one edge that belongs to one but not the other. Among such edges, let e1 be the one with least weight; this choice is unique because the edge

weights are all distinct. Without loss of generality, assume e1 is in A. As B is a MST, $e1 \cup \cup$ B must contain a cycle C. As a tree, A contains no cycles, therefore C must have an edge e2 that is not in A. Since e1 was chosen as the unique lowest-weight edge among those belonging to exactly one of A and B, the weight of e2 must be greater than the weight of e1. Replacing e2 with e1 in B therefore yields a spanning tree with a smaller weight. This contradicts the assumption that B is a MST. More generally, if the edge weights are not all distinct then only the (multi-)set of weights in minimum spanning trees is certain to be unique; it is the same for all minimum spanning trees
Hence, the Sufficient is easy to explain as beacuse every edge in T must in G, and the path always holds the graph.

https://cs.stackexchange.com/questions/61188/how-to-show-that-two-vertices-in-a-conn
https://www.hackerearth.com/practice/algorithms/graphs/minimum-spanning-tree/tutorial
https://en.wikipedia.org/wiki/Minimum_spanning_tree

3. In this question, you are required to implement the program to find the connected components and strongly connected components.

   (a) The structure of the **undirected** graph is given in the file "Data-Graph.txt". In the first line, there are two numbers: the size of nodes $|V|$ and edges $|E|$. The nodes in the graph are labeled by integer $1, 2, \ldots, |V|$. The following contents are given in $|E|$ lines. Each line contains two number u and v, which means there is a **undirected** edge between u and v. Please implement a program to find the maximal connected component by C/C++, Java or Python. The requirements are following:

      i. You should Describe the idea of your program and the size of the maximal connected component in this assignment;
      ii. You should Attach the **source code** called "Code-CC.*" and the **result file** called "Result-CC.txt";
      iii. The result file contains p lines where p is the size of the maximal connected component and you should provide the node label line by line in the ascending order;
      iv. Give the arbitrary one if there are more than one maximal connected component;
      v. Please do not upload other files (for example, "Data-Graph.txt").

   (b) In this subquestion, the graph given in "Data-Graph.txt" is a **directed** graph. For the line 2 to line $|E| + 1$, the two number u and v means there is a **directed** edge from u to v. Please implement a program to find the maximal strongly connected components by C/C++, Java or Python. You should attach **source code** called "Code-SCC.*" and the **result file** called "Result-SCC.txt". The other requirements of this question are same as question 3(a).

   (c) (Bonus) Visualize the result in 3(a)(b). You should visualize the meta-node and the edges between meta-node and the radius of meta-node represents the size of corresponding (strongly) connected component.

Solution: Please check the Code file for part(a) Lab07.py + result-cc.txt and for part(b) Lab07-scc.py + result-scc.txt
Code Reference:
https://www.cdn.geeksforgeeks.org/strongly-connected-components/