COSC 302: Analysis of Algorithms Lecture — Spring 2018
Prof. Darren Strash
Colgate University

Worksheet 7 — Graphs; Traversals & Applications; Minimum Spanning Trees

1 Graphs

1. Give the formal definition of a graph.

   Solution: A graph $G = (V, E)$ is an ordered pair of a set of vertices V and a set of edges $E \subseteq V \times V$.

2. A graph can be formed between any objects that have pairwise relationships. Describe how to form graphs from objects that you encounter every day. What do the vertices and edges represent?

   Solution:

   - Friendship networks; there is a vertex for each person, and an edge exists between two vertices iff their respective people are friends.
   - "Who follows whom" in the Twitter network; again a vertex for each person, and an edge (u, x) is in the graph iff u follows x.
   - Road networks; there is a vertex for each intersection, and edges for each road segment connecting intersections.

3. What is the maximum number of edges that an undirected graph on n vertices can have?

   Solution: This is the number of ways to choose 2 items from n items. That is, $\binom{n}{2} = \frac{n(n-1)}{2}$.

4. What is the running time of breadth-first search if you are given the adjacency matrix representation (and not the adjacency list) of the graph?

   Solution: When removing a vertex from the queue, we evaluate all of its neighbors. In an adjacency matrix, this takes $O(n)$ time. This is done n times, taking overall $O(n^2)$ time.

5. What is a tree and how does it differ from a rooted tree and an ordered tree?

   Solution: A graph $G = (V, E)$ is a tree iff

   - G is connected,
   - G has $|V| - 1$ edges, and
   - G has no cycles.

   Noe that, if any two of these are true, this implies the third is true.

   A rooted tree has a distinguished node called the *root*, which induces a parent/child relationship between vertices. An ordered tree is rooted, and children have a predefined order. A heap is an example of a rooted tree, a binary search tree is an example of an ordered tree.

2 Graph Traversals and Applications

6. Suppose you are in a maze and you wish to find the closest exit. Describe an efficient algorithm to solve this problem.

   Solution: Model the maze as a graph. If you have access to a map of the maze, then run BFS on the graph, ending when encountering the first exit. This is the closet exit.

7. Describe an algorithm to decide if a given graph is a tree. What is the running time of your algorithm?

   Solution: Verify two of the three properties listed above for trees:

   - Check that G is connected. Run BFS and check that it reaches all vertices in the graph.
   - Check to see if the number of edges is $|V| - 1$. This can by done by iterating over the adjacency list and counting the edges in $\Theta(n + m)$ time.
   - Check that G has no cycles. Run DFS on the input graph. If DFS yields any non-tree edges then the graph has a cycle.

8. Describe an algorithm to compute an Euler tour in $O(n + m)$ time.

   Solution: Run DFS. Every recursive call introduces a new forward edge, returning from the recursive call introduces a return edge. When encountering a non-tree edge, introduce two edges, one out and back.

9. Prove by contradiction that the component graph is a dag.

   Solution:

   Proof. Assume for sake of contradiction that G is a component graph and not a dag. We will show that G is not a component graph.

   Since G is a component graph, it is directed, and each vertex V_i corresponds to a strongly connected component, which is a set of vertices in some other graph $G' = (V', E')$.

   If G is not a dag, then it contains a cycle $V_1, V_2, V_3, \ldots, V_k, V_1$. Then $\exists (v_1, v_2) \in E'$, such that $v_1 \in V_1, v_2 \in V_2$.

   Note then that there is a path from v_1 to v_2. Note that there is also a path from v_2 to v_1, that passes through vertex sets V_2,V_3,...,V_k,V_1. Thus, $V_1 \cup V_2 \cup \cdots V_k$ is strongly connected, and V_1,V_2 are not strongly connected components. Therefore, G is not a component graph. □

3 Minimum Spanning Trees

10. Give a graph that does not have a unique minimum spanning tree.

    Solution: A cycle where every edge has weight one.

11. Describe an algorithm to determine if a graph has a unique minimum spanning tree.

    Solution: First compute a minimum spanning tree (MST) $T = (V_T, E_T)$. For each edge in E_T, remove the edge, compute a minimum spanning tree of the remaining graph, and if it has the same weight as T, then the MST is not unique; otherwise return e to the graph and continue. If no MST (computed in this way) has the same weight as T then the MST is unique.

12. Describe how to implement the Prim-Jarník algorithm to run in $O((n + m) \lg n)$ time.

    (To be discussed in lab.)