# Exercises on Graphs

Giulia Bernardini

May 2022

Besides the exercises listed here below, you can find a lot of exercises both on Cormen and on Gusfield books. I also recommend you try to solve the exercises I proposed during the lectures. You can always email me if you want to check your solutions. Some of the questions of the exam will also require theoretical notions given in the lectures (e.g., to define some objects or to describe some of the analysed algorithms). Good luck!

**Exercise 1.** List the vertices in the order in which they are visited (for the first time) in DFS for the following undirected graph, starting from vertex 0.

0-1 1-2 1-7 2-0 2-4 3-2 3-4 4-5 4-6 4-7 5-3 5-6 7-8 8-6

Assume that the graph is represented using a adjacency matrix so that you iterate through the vertices adjacent to v in increasing order.

**Exercise 2.** Consider the same graph as Exercise 1. Give a topological sort of the vertices according to the DFS resulting from Exercise 1.

**Exercise 3.** Consider the same graph as Exercise 1. List the vertices in the order in which they are visited (for the first time) in BFS, starting from vertex 0. Also give the length of the shortest path (number of edges) from vertex 0 to each other vertex.

**Exercise 4.** A *rooted DAG* is a connected directed acyclic graph such that exactly one node (called the root) has in-degree 0 and out-degree $> 0$, and the other nodes have in-degree $\geq 1$ and out-degree $\geq 0$. Given a rooted DAG with root r, the *level* of a node $u$ is given by the length of the path from $r$ to $u$. Describe an algorithm that assigns to each node its level and bound its asymptotic time complexity. (Hint: use BFS)

**Exercise 5.** Every tree is a directed, acyclic graph (DAG), but there exist DAGs that are not trees. How can we tell whether a given DAG is a tree? Devise an algorithm to test whether a given DAG is a tree. (Hint: use the colors assigned to the vertices during a visit of the graph)

**Exercise 6.** In general, an undirected graph contains one or more connected components. A connected component of a graph G is a subgraph of G that is connected and contains the largest possible number of vertices. Each vertex of G is a member of exactly one connected component of G. Describe an algorithm that assigns a different identifier to each connected component in a graph and bound its asymptotic time complexity. (Hint: use DFS)

**Exercise 7.** Given a connected, undirected graph, design an algorithm that partitions the vertices in two groups such that no edge links two vertices of the same group; or return FAIL if no such partition exists. Bound the asymptotic time complexity of your algorithm. (Hint: use values $v.distance$ computed during BFS).

**Exercise 8.** Consider the graph in Figure 1. Apply the Dijkstra Algorithm to the graph to compute a shortest-path tree rooted in a and the distance between any vertex and vertex $a$. The first three steps of the algorithm must be detailed (specify the elements in the min-priority queue and in the finalised vertices set at each step). Moreover, indicate the order in which vertices are considered during the execution of the algorithm.

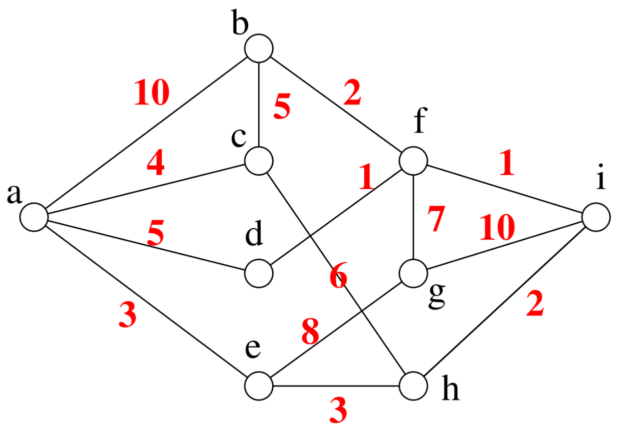

Figure 1

**Exercise 9.** Draw the suffix tree of the string $S = babau\$$, including suffix links, labelling each edge both with the explicit substring of $S$ it represents and with an interval of positions in $S$.

**Exercise 10.** Describe an algorithm that, given a text $T$, finds the longest substring of $T$ that occurs in T at least 3 times. Analyse the asymptotic worst-case time complexity of your algorithm (hint: preprocess the suffix tree of $T$ suitably)

**Exercise 11.** Describe how to list all suffixes of a string $T$ in lexicographical order, using the suffix tree of $T$. Analyse the asymptotic worst-case time complexity of your algorithm.

**Exercise 12.** Compute the array built in the preprocessing phase of the KMP algorithm for the pattern $P = xyzxwxuxyzx$.

**Exercise 13.** Describe the steps done to find the pattern $P = ip$ in the suffix array of string $S = mississippi$.

**Exercise 14.** Apply the shift-and method to the pattern $P = uxxw$ and the text $T = uxxyuxuxuxxz$.

**Exercise 15.** Compute the edit distance between strings $S = zyu$ and $T = xyzuw$, filling in the dynamic programming table. List all possible optimal alignments of the two strings.