Mathematics for Computer Science (30470023) Spring 2020

Lecture 10 — Apr 27, 2020

Instructor: Prof. Andrew C. Yao *Scribes: Yaxin Tu, Chang Liu*

1 Overview

In this lecture, we introduce some preliminary concepts in graph theory, expound briefly on the subjects of Eularian and Hamiltonian cycles and touch on the notion of reducibility.

2 Graph Theory Preliminaries

We start with some basic concepts in graph theory.

2.1 Simple Graphs

A simple graph G is specified by two sets V and E, where V is a non-empty finite set and E is a subset of $\{\{v, v'\} | v \neq v' \in V\}$. We denote the simple graph G by $G = (V, E)$.

Example 2.1

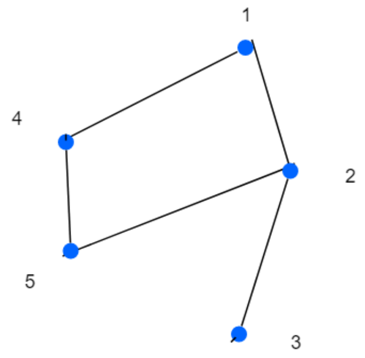

Figure 1: An example of a simple graph

Figure 1 above is a graphical representation of the simple graph G, which has 5 vertices and 5 edges. To put it more formally, $G = (V, E)$ where

$$V = \{1, 2, 3, 4, 5\}$$
$$E = \{\{1, 2\}, \{2, 3\}, \{2, 5\}, \{4, 5\}, \{1, 4\}\}$$

We also define complete graphs, which are graphs where $|E| = \binom{|V|}{2} = \frac{|V|(|V|-1)}{2}$. We denote by K_n a complete graph on n vertices. More specifically, $K_n = (V, E)$, where $|V| = n$ and $E = \{\{v, v'\}|v \neq v' \in V\}$.

In addition, G is an empty graph when the $E = \emptyset$.

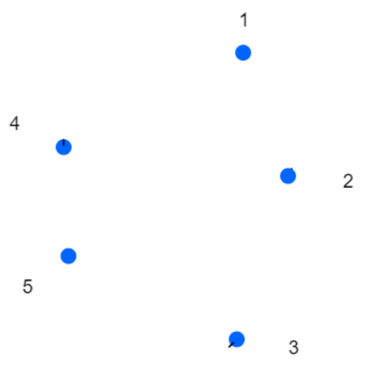

Figure 2: Empty graph illustrated

2.2 Multi-graphs

The definitions above are for simple graphs. Now we "generalize" the definition of it and define a multigraph $G = (V, E)$. While V, the vertex set, is still required to be a finite set, in a multigraph, the definition of E, the edge set is different: in a multigraph, $E = \{e_1, e_2, \cdots e_m\}$ where each e_i is of the form $\{v, v'\}$ s.t. $v, v' \in V$.

To illustrate this, we note the following:

1. Multi-graph is a generalization of simple graph
2. The difference between multi-graph and simple graph:
   - Multi-graphs allow for multiple edges, (E can be a multiset). For example, in a muli-graph, we can have $e_3 = \{v_5, v_8\}$ and $e_4 = \{v_5, v_8\}$.
   - Multi-graphs allow for self-loops, i.e. $e = \{v, v\}$.

In Figure 3 below, the multigraph $G = (V, E)$ has $V = \{1, 2, 3, 4, 5, 6\}$, $E = \{e_1, e_2 \cdots, e_{13}\}$, multi-edges $e_1 = e_2 = e_3 = e_4 = \{1, 6\}$, and self-loops e_5, e_{12}, e_{13}

2.3 Paths, Connectivity, and Degree of a Vertex

In a multi-graph $G = (V, E)$, A path from v_1 to v_l is a sequence

$$v_1 \xrightarrow{e_1} v_2 \xrightarrow{e_2} v_3 \xrightarrow{e_3} v_4 \xrightarrow{e_4} \cdots v_{l-1} \xrightarrow{e_{l-1}} v_l$$

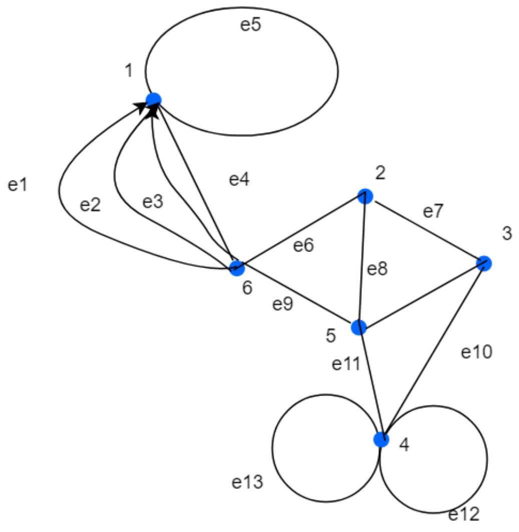

Figure 3: Multi-graph illustrated

where $e_i \in E, v_i \in V$.

A path is called a cycle if $v_1 = v_l$.

Note that both v_is and e_is may be repeated.

A multi-graph G is connected if for every pair $v, v' \in V$, there exists a path from v to v'. In Figure 4 below, G is connected but G' isn't, since there is no path from v to v'.

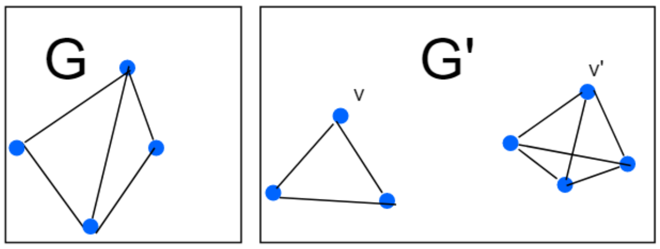

Figure 4: Connectivity illustrated

In addition, in a multi-graph $G = (V, E)$, we define the degree of of a vertex v, denoted by $\deg(v)$, is the number of edges incident to v ($\{v, v'\} \in E$). It is worthy to note that, to make some theorems more elegant, we require that a self loop be counted *twice* for the degree. In Figure 5 below, $\deg(v_1) = 3, \deg(v_4) = \deg(v_3) = 2, \deg(v_2) = 5$.

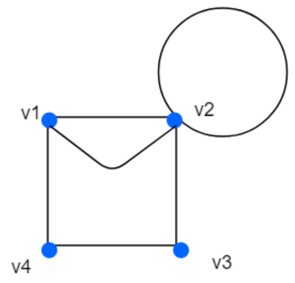

Figure 5: Degree of a vertex illustrated

3 Eulerian Cycles

3.1 The Königsberg Bridge Problem

The notion of Eulerian Cycles arose from the historically famous Königsberg Bridge Problem. Figure 6 is a cartoon illustration of the problem: There are two islands in the river Preger and seven bridges connecting the opposite banks of the revier and these two islands; is it possible to take a round tour of the city so that every bridge is passed exactly once? In 1736, a negative

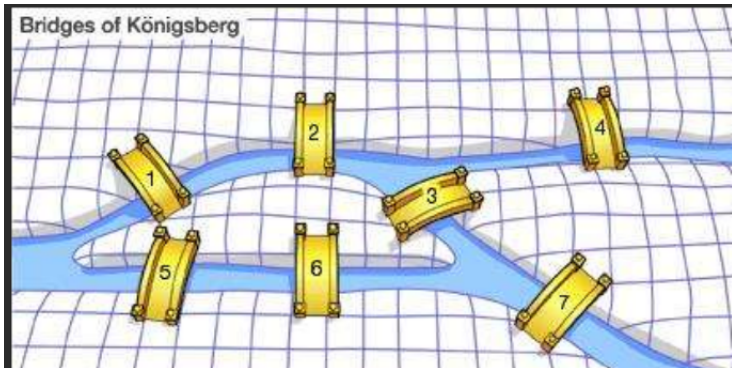

Figure 6: Königsberg Bridge Problem

answer to this question was put forward by the great mathematician Euler, in a paper that marked the nascence of graph theory. (In fact, the Königsberg Bridge Problem served as a predecessor of Topology.)

3.2 The Eulerian Cycle Problem

We first state the Eulerian Cycle Problem:

Given a multi-graph G, decide whether there exists an Eulerian Cycle, i.e. a cycle that traverses every edge exactly once.

Clearly, the Königsberg Bridge Problem is equivalent to solving the Eulerian Cycle Problem in the graph presented in Figure 7. In 1736, an Euler's Theorem specifying the necessary and sufficient conditions of the existence of an Eulerian cycle was discovered and proved by Euler himself:

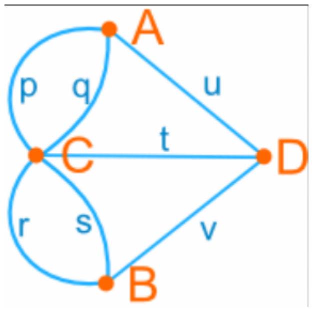

Figure 7: Königsberg Bridge Problem presented in a graph

Theorem 3.1. A multigraph $G = (V, E)$ has an Eulerian Cycle if and only if:

(1) G is connected except for isolated vertices.

(2) $\deg(v)$ is even for every $v \in V$. $\diamondsuit$

It is easy to see that the two conditions are necessary: G must be connected, and since for every vertex, traversing a cycle means "leaving" and then "coming back to" the vertex via distinct edges connected to it; v has an odd degree, there would be a contradiction (since in this case, a cycle will leave a vertex and never return or go into a vertex and stop there). Hence, to prove Euler's Theorem, we need only to verify that these two conditions are sufficient.

The idea of the proof is as follows: start at any vertex v_0, try to construct a path using new (i.e. unused) edges until you cannot continue. In this case, you can always continue unless you have come back to v_0. So you have traversed a Cycle C, and C is either Eulerian or there still exists other un-used edges.

If the latter is the case, since G is connected, there must be a vertex v_1 on C such that there are extra edges on it that are not included in C. We repeat with respect to v_1 the process delineated above, generating C' and augment C and C' by traversing $v_0 \xrightarrow{C} v_1 \xrightarrow{C'} v_1 \xrightarrow{C} v_0$. The augmentation process is illustrated in Figure 8

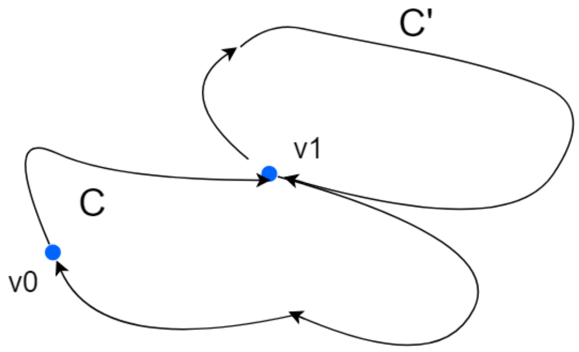

Figure 8: Augmenting cycles C and C'

We go on to state the proof of the theorem more formally.

Proof. We shall display an algorithm through which we'll eventually find an Eulerian Cycle in any multigraph $\mathrm{G} = (\mathrm{V}, \mathrm{E})$ satisfying (1) and (2). The algorithm is as follows:

Step 1 Keep the current cycle C as a variable and initiate it to be $\emptyset$. Let u be the variable denoting the starting vertex and initiate it to be an arbitrary vertex v_0. Let $\mathrm{E}' = \mathrm{E}$. Note that $\mathrm{E}' \cup C$ is a division of E.

Step 2 Construct a path $C' : u \xrightarrow{e_1} u_1 \xrightarrow{e_2} u_2 \xrightarrow{e_3} \cdots \xrightarrow{e_{l-1}} u_{l-1} \xrightarrow{e_l} u_l$ with distinct edges from E' until one cannot continue extending the path. The only case that one has to stop is that $u_l = u$. Thus C' is actually a cycle. Then update C by $C \cup C'$ and update E' by $\mathrm{E}' \backslash C'$. Still the updated C and E' forms a division of E.

Step 3 If $\mathrm{E}' \neq \emptyset$, pick a new vertex u in C that has an edge in E' incident to u. There must be such a vertex due to the connectivity of G. Then go back to step 2. If $\mathrm{E}' = \emptyset$, then terminate and output C.

We shall see directly that first the process will definitely come to an end since each time we update C and E' the number of edges in E' strictly decreases. Secondly, the output C is an Eulerian Cycle since its complement E' is an emptyset.

From simple observations we can see that $|\mathrm{E}'|$ decreases at least one at a time when we complete step 2. So the total number of steps will not surpass $2|\mathrm{E}|$. In other words, the algorithm is quite efficient.

Theorem 3.2. There exists an efficient algorithm that can decide whether G is Eulerian and construct an Eulerian Cycle. ♢

4 Representations of a Simple Graph

In this section we'll learn how computers represent a simple graph $\mathrm{G} = (\mathrm{V}, \mathrm{E})$ where $\mathrm{V} = \{v_1, v_2, \cdots, v_n\}$.

1. Adjacency Matrix

Sometimes we may represent G with a $n \times n$ matrix $\mathrm{M}_\mathrm{G} = (a_{ij})$ where $a_{ij} = \begin{cases} 1 & \{v_i, v_j\} \in \mathrm{E} \\ 0 & \{v_i, v_j\} \notin \mathrm{E} \end{cases}$. a_{ii} is always 0.

For example the adjacency matrix of the graph in Figure 9

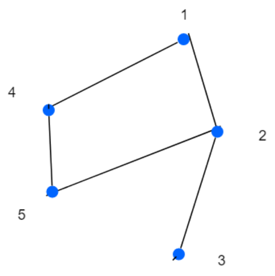

Figure 9: A graph G

should be

$$M_G = \begin{pmatrix} 0 & 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 \end{pmatrix}$$

Note that the adjacency matrix of an undirected graph is symmetric. The storage it uses will always be $O(n^2)$ which is not that efficient in some cases.

2. Adjacency List

Another method is to obtain n lists among which the ith list is $v_i \to v_{i1} \to v_{i2} \to \cdots \to v_{ik}$ where $\{v_i, v_{ij}\}$, $1 \leq j \leq k$ are all edges incident to v_i. The storage used by this method is $O(|V| + |E|)$ and it performs much better than the first one when G doesn't have too many edges.

5 Hamiltonian Cycle

In previous section we have learnt about a cycle that covers all edges exactly once. In this section we'll encounter a cycle that covers all vertices exactly once.

Definition 5.1. A *Hamiltonian Cycle* is a cycle in a simple graph that goes through all vertices exactly once. ♢

Example 5.2. The graph in figure 10 does not have a Hamiltonian Cycle.

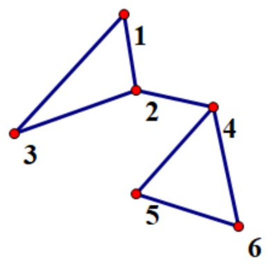

Figure 10: A graph containing no Hamiltonian Cycle

Finding a Hamiltonian Cycle in a large graph is a very difficult game for even the smartest computers in that no “efficient” algorithm is known for deciding whether a large graph has a Hamiltonian Cycle.

6 Introduction to Reducibility

We have seen that Eulerian Cycle Problem is equivalent to a rather simple problem but Hamilton Cycle problem is of no simple characteristic so far. Why are Eulerian Cycle Problem and Hamiltonian Cycle Problem so different in complexity?

In order to explain it, we introduce the concept *reducibility.* This concept generates the central idea to ”P$\neq$ NP” question.

Definition 6.1. Problem A is *(polynomial-time)-reducible* to problem B if any algorithm for solving B can be utilized to solve A. In a sense, A is easier than B. $\diamondsuit$

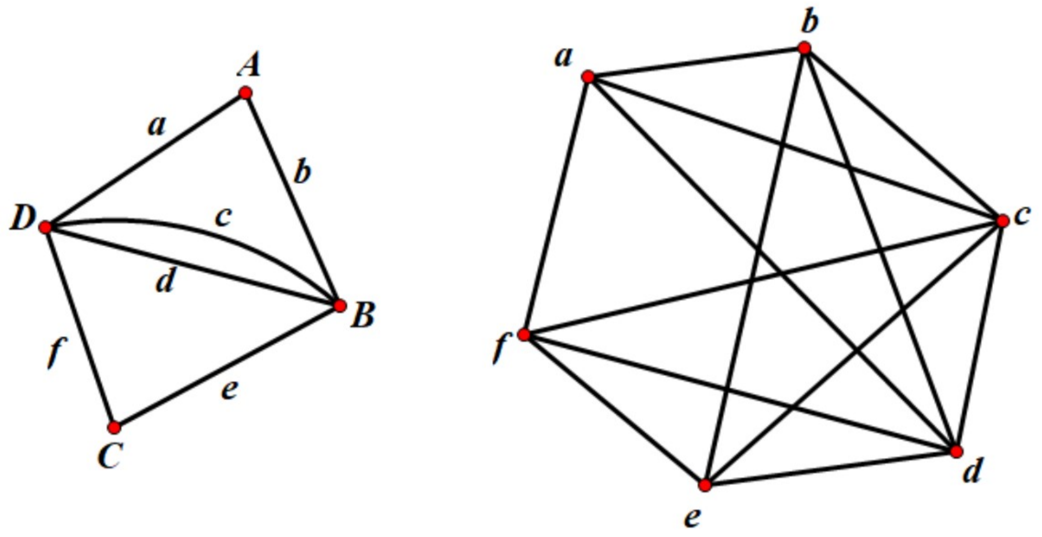

Figure 11: The Illustration of convertion

Next we’ll show Eulerian Cycle Problem is reducible to Hamiltonian Cycle Problem, and hence the former one is “easier”. The idea is creating a new graph G′ based on the original (multi)-graph G in Eulerian Cycle Problem by exchanging vertices and edges. The vertices in G′ are the edges in G and two vertices are neighbours in G′ if and only if the corresponding pair of edges in G share a common end point. The idea is illustrated in following figure and will be demonstrated completely in next class.