Lecture 15: Graph Theory (Contd)

Types of graphs

Depending on their connectivity structures, graphs can be classified according to various types. Here are some common graph classes:

- A *line graph* with nodes/vertices $v_1, v_2, \ldots, v_n$ consist of edges $\{v_1, v_2\}, \{v_2, v_3\}, \ldots, \{v_{n-1}, v_n\}$, i.e., the nodes are connected sequentially. Line graphs with n nodes have $n - 1$ edges.
- A *cycle* with $n \geq 3$ nodes has all the edges as a line graph, with the additional edge $\{v_1, v_n\}$. A cycle is like a "closed loop": one can start at any node and traverse the rest of the nodes sequentially to return to the starting point.
- A *tree* is a connected graph containing no cycles. (Here, "connected" means that the graph consists of a single component as opposed to multiple islands, and that there is a well-defined path from every node to every other node.)
- A *star graph* consists of a "central node" v_1 that is connected to the other "outer" nodes via edges $\{v_1, v_2\}, \{v_1, v_3\}, \ldots, \{v_1, v_n\}$. Star graphs with n nodes have $n - 1$ edges.
- A *wheel* with n nodes is formed by starting with a star graph (having $n - 1$ edges), and forming a cycle through the $n - 1$ "outer" nodes by adding $n - 1$ additional edges.
- A *complete* graph with n nodes, commonly denoted by K_n, is formed by connecting each of the n nodes with every other node. Complete graphs with n nodes have $\binom{n}{2}$ vertices; try proving this via a simple counting **exercise**.

Euler, Hamilton, and Traveling Saleperson

We return to the point we started when we discuss graphs. Recall the map of Konigsberg, where Euler used to live in the 18th century.

There were 7 bridges across the rivers connecting different landmasses, and a friend of Euler posed to him the following question:

> Is it possible to walk around the city such that each bridge is crossed once, and exactly once?

You can try tracing different walks across the bridges yourself. However, you'll find that a walk that does not repeat edges seems difficult to find. In fact, such a walk is impossible, and Euler proved this using graph theory.

Euler's main idea (like most great ideas) was deceptively simple:

> Draw the map as a graph.

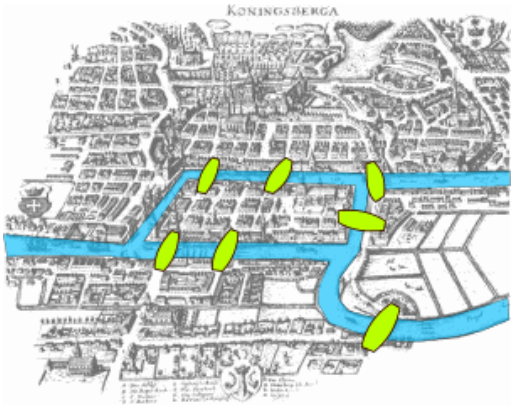

Figure 1: Bridges of Konigsberg

It does not matter where the bridges were located; all that matters is that we represent landmasses as nodes, and bridges between landmasses as edges connecting nodes. Rewriting the map thus gives us the following graph:

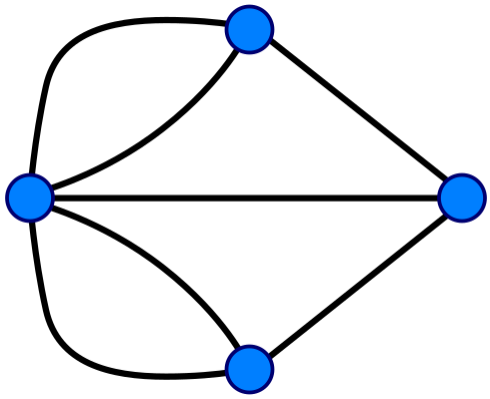

Figure 2: Graph of Konigsberg

except the initial and final landmasses, every landmass must be touching an *even* number of bridges.

In the language of graph theory, it is necessary that every node in the path (except first and last) has to have *even degree*.

However, the graph in question has all four nodes of *odd* degree. therefore, no matter what the start and end points were, such a path cannot exist. Done!

Such paths that traverse a given graph without repeating edges more than once are called *Euler* paths. The above proof can be generalized into a more general *theorem*: if any graph exhibits an Euler path, two conditions are necessary: the graph should be connected, and that the number of nodes with odd degree must be zero or two. Try writing this out into a formal proof.

In fact, the above two conditions are also *sufficient* for an Euler path to exist. This is somewhat harder

to prove, but the textbook has a full proof.

On the other hand, there is a different type of path known as a *Hamilton* path: here, the goal is to traverse a graph by visiting each *node* exactly once. In contrast with Euler paths, Hamilton paths are much harder to reason about. For example, as an **exercise**, try tracing out a Hamilton path through the graph given below.

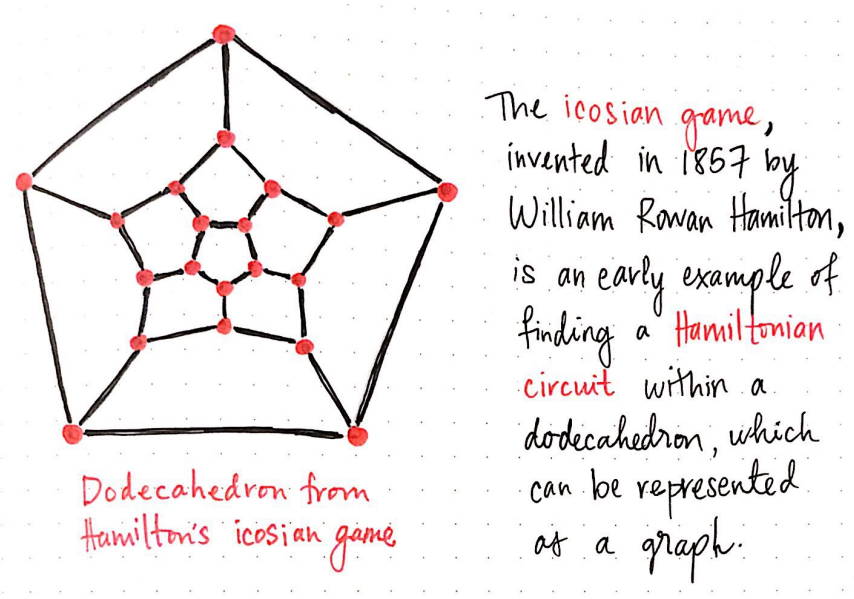

Figure 3: Source: medium.com

Applications in NLP

Graphs are integral data structures in a vast field of study called *natural language processing* (NLP). This is an area of artificial intelligence which attempts to help computers understand, model, and synthesize spoken and written language.

NLP is everywhere: it powers search engines such as Google; helps MS Word auto-correct your grammar; and lets your car audio system understand and respond to simple voice commands. Let us focus on a simple and familiar example: how does your text messaging software have the ability to auto-suggest the next word when you type out a message? (Keep in mind that the word not only has to make sense, but also produce a grammatically correct sentence. They work very well – try generating a few synthetic sentences using your phone if you haven't already.)

The idea is to store words in the form of a special *directed* graph called a "word chain automaton". Here is a picture:

In this directed graph, nodes represent words, and edges represent allowable grammatical relationships. You can start from any reasonable start state (a preposition, or a noun) and hop along edges to traverse this graph. As you do so, you always generate grammatically meaningful sentences.

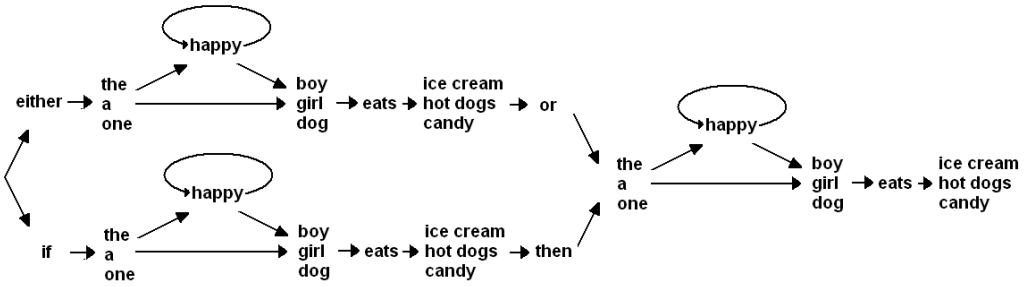

Figure 4: Word Chain Automata

In practice, the automaton is much bigger (containing thousands of words and millions of edges). But the idea is the same. This is also how your word processing program draws a squiggly red line below a sentence which doesn't make grammatical sense – as you type the sentence out, the program traverses the same graph and checks whether the sentence respects the structure of the automaton.

Now, "grammatically meaningful" isn't the same as "meaningful", and that is why it is perfectly possible to generate synthetic sentences that are grammatically correct but don't make sense. (For example, the phrase "Fish fish fish" is grammatically correct!)

Word chain automata are examples of so-called "finite state machines", which can be used to model all kinds of systems ranging from software checking programs to elevator scheduling mechanisms. Automata theory is an entire course in itself and we won't have time to delve too deep into this; just be aware that they exist.