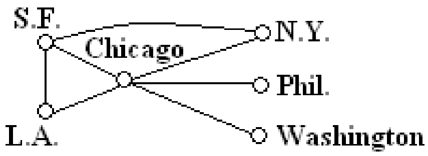
CS 70 Fall 2006 — Discrete Mathematics for CS — Papadimitriou & Vazirani — Lecture 16

Graphs

Formulating a simple, precise specification of a computational problem is often a prerequisite to writing a computer program for solving the problem. Many computational problems are best stated in terms of graphs: a directed graph $G(V,E)$ consists of a finite set of vertices V and a set of (directed) edges or arcs E. An edge is an ordered pair of vertices (v,w) and is usually indicated by drawing a line between v and w, with an arrow pointing towards w. Undirected graphs may be regarded as special kinds of directed graphs, such that $(u,v) \in E \leftrightarrow (v,u) \in E$. Thus, since the directions of the edges are unimportant, an undirected graph $G(V,E)$ consists of a finite set of vertices V, and a set of edges E, each of which is an unordered pair of vertices $\{u,v\}$. As we have defined them, graphs are allowed to have self-loops; i.e. edges of the form (u,u) that go from a vertex to itself. Sometimes it is more convenient to disallow such self-loops.

Graphs are useful for modeling a diverse number of situations. For example, the vertices of a graph might represent cities, and edges might represent highways that connect them. In this case, the edges would be undirected:



In the above picture, $V = \{\text{SF,LA,NY},\ldots\}$, and $E = \{\{\text{SF,LA}\},\{\text{SF,NY}\},\ldots\}$. Alternatively, an edge might represent a flight from one city to another, and now edges would be directed (a certain airline might have a non-stop flight from SFO to LAX, but no non-stop flight back from LAX to SFO).

Graphs can also be used to represent more abstract relationships. For example, the vertices of a graph might represent tasks, and the edges might represent precedence constraints: a directed edge from u to v says that task u must be completed before v can be started. An important problem in this context is scheduling: in what order should the tasks be scheduled so that all the precedence constraints are satisfied.

As a reminder, the Cartesian product of two sets U and V is defined as follows (more details can be found in the appendix): $U \times V = \{(u,v) : u \in U \text{ and } v \in V\}$. Thus, if we have a graph $G = (V,E)$ with V being a finite set of vertices, then $E \subseteq V \times V$. For the purposes of this lecture, we will study graphs that do not contain self loops.

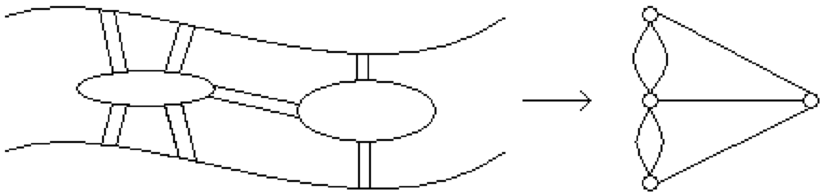
A **path** in a directed graph $G = (V,E)$ is a sequence of neighbor edges $(v_1,v_2),(v_2,v_3),\ldots,(v_{n-2},v_{n-1}),(v_{n-1},v_n)$. In this case we say that there is a path between v_1 and v_n. A path in an undirected graph is defined similarly. A path is called **simple** if it has no repeating vertices.

A graph is called **connected** if there is a path between any two distinct vertices.

If $G = (V,E)$ is an undirected graph then the **degree** of vertex $v \in V$ is the number of edges incident to v. In notation, $degree(v) = |\{u \in V : \{v,u\} \in V\}|$. A vertex v whose degree is 0 is called an **isolated vertex**, since there is no edge which connects v to the rest of the graph.

Eulerian Paths and Tours

A few centuries ago, people were trying to solve a problem that today is called The Seven Bridges of Königsberg, inspired by a real place and situation:



People wanted to know whether or not it was possible to cross all seven bridges without crossing any bridge more than once. In 1736, the brilliant mathematician Leonhard Euler proved that such a task was impossible. In doing so, Euler was hailed as the inventor of graph theory.

An **Eulerian path** is a path in a graph that uses each edge exactly once (sometimes to emphasize that Eulerian paths are not simple, i.e. they might go through a vertex more than once, they are called Eulerian walks). So a path from u to w is a sequence of edges $(u,v_1),(v_1,v_2),(v_2,v_3),\ldots,(v_k,w)$. An **Eulerian tour** is an Eulerian path whose starting point is also the ending point (i.e., $u = w$).

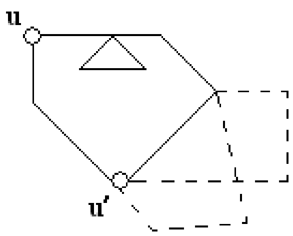
The next theorem gives necessary and sufficient conditions of a graph having an Eulerian tour.

Euler's Theorem: An undirected graph $G = (V,E)$ has an Eulerian tour if and only if the graph is connected (with possible isolated vertices) and every vertex has even degree.

Proof ($\Longrightarrow$): So we know that the graph has an Eulerian tour. This means every vertex that has an edge is in the tour, and is therefore connected with all other vertices in the tour. This proves the condition of connectedness. For each vertex in the tour except the first one, the walk leaves it just after entering. Thus, every time we use two edges (i.e., an even number of edges). Since the Eulerian tour uses each edge exactly once, every vertex has even degree. Also, the first vertex v has even degree because the walk leaves it in the beginning but returns to v at the end.

Proof ($\Longleftarrow$): Our strategy is to construct an Eulerian tour knowing that we have a connected graph $G(V,E)$ with each vertex having an even degree. But first, we make the following two observations.

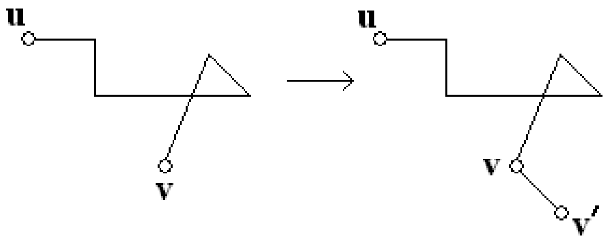
- We can start walking from some vertex u, never repeating edges until we are stuck. We claim that we will be stuck at u.
- Now, let us pick some untraversed edge $\{u',v'\}$ with one endpoint u' on the current closed walk. We can repeat the first step starting from u' and splice in the resulting closed walk.



As the figure above shows, we may have to splice in a closed walk because we cannot be certain that when we get stuck at u, we will have already traversed every edge in the graph. How can we be sure that we will get stuck at u? We will show that we can always keep going if we are not yet at u by proving the proposition

$P(n)$: a length n walk from u to v, $u \neq v$, has only two vertices of odd degree (u and v) with the rest having even degree.

- Base Case: Prove $P(1)$. This is easy though, since it is a walk that traverses one edge which connects two vertices u and v. Each of these vertices has odd degree, since there is only one edge incident to each vertex.
- Inductive Hypothesis: Assume $P(n)$ is true. That is, in a length n walk from u to v, the degree of u and v in the walk is odd, and the rest of the vertices have even degree.
- Inductive Step: Prove $P(n+1)$. By the inductive hypothesis, the walk traversing the first n edges from u to v has two vertices of odd degree with the rest being even (since we enter and then leave all such vertices). So now, if we connect the $n+1^{st}$ edge $\{v, v'\}$, to the graph, we see that the vertex v now has an even degree while v' has odd degree.



In constructing the Eulerian tour, we take for granted that two closed paths with a common vertex can be combined into one larger closed path.

de Bruijn Graphs

A **de Bruijn sequence** is a 2^n bit circular sequence such that every string of length n occurs as a contiguous substring of the sequence exactly once. For example, the following is a de Bruijn sequence in the case that $n = 3$:

1 0

0 0

1 0

1 1

Notice that there are eight substrings of length three, each of which corresponds to a binary number from 0 to 7. It turns out that such sequences can be generated from a de Bruijn graph $G = (V, E)$, where V is the set of all $n-1$ bit strings (i.e., $V = \{0,1\}^{n-1}$). Each vertex $a_1a_2...a_{n-1}$ has two outgoing edges: $(a_1a_2...a_{n-1}, a_2a_3...a_{n-1}0)$ and $(a_1a_2...a_{n-1}, a_2a_3...a_{n-1}1)$. Therefore, each vertex also has two incoming edges: $(0a_1a_2...a_{n-2}, a_1a_2...a_{n-1})$, $(1a_1a_2...a_{n-2}, a_1a_2...a_{n-1})$. For example, if we have the vertex 110_2, then the two outgoing edges would be directed toward the vertices 100_2, and 101_2. Note that these are directed edges, and self-loops are permitted.

The de Bruijn sequence is generated by an Eulerian tour in this graph. Euler's theorem above can be modified to work for directed graphs. This is not too difficult, since all we need to modify is the second condition; it must say: "for every vertex v in V, the indegree of v equals the outdegree of v". Clearly, the de Bruijn graph satisfies this condition, and therefore it has an Eulerian tour.

To actually generate the sequence, starting from any vertex, we walk along the tour T and add the corresponding bit which was shifted in from the right as we traverse each edge. Here is an example in the case that $n = 3$: