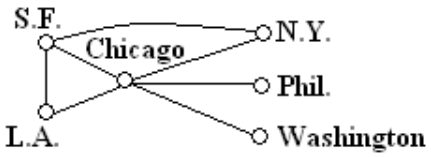
CS 70 Discrete Mathematics for CS
Spring 2008 David Wagner
Note 13

# An Introduction to 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* or *nodes*) $V$ and a set of (directed) *edges* $E$. 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, in which $(u,v) \in E$ if and only if $(v,u) \in E$. Thus in an undirected graph the directions of the edges are unimportant, so an edge of an undirected graph is an *unordered* pair of vertices $\{u,v\}$ and is indicated by a line between $u$ and $v$ with no arrow.

More formally, the edge set of a directed graph is a subset $E \subseteq V \times V$, where $V \times V$ is the *Cartesian product* of $V$ with itself.[1] As we have defined them, graphs are allowed to have *self-loops*; i.e., edges of the form $(u,u)$ or $\{u,u\}$ that go from a vertex $u$ to itself. Usually, however, graphs are assumed to have no self-loops unless otherwise stated, and we will assume this from now on.

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 the edges would be directed (since there may be a non-stop flight from SF to LA, but no non-stop flight back from LA to SF).

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?

A *path* in a directed graph $G = (V,E)$ is a sequence of neighboring 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 *from* $v_1$ *to* $v_n$. A path in an undirected graph is defined similarly. A path is called *simple* if it has no repeating vertices. A *cycle* (or *circuit*) is a path that begins and ends at the same vertex (i.e., it is a path from $v$ to $v$ for some $v$).

A graph is said to be *connected* if there is a path between any two distinct vertices.

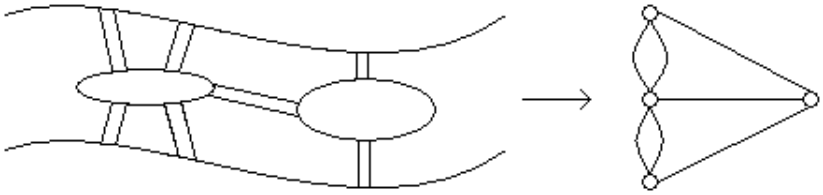
If $G = (V,E)$ is an undirected graph then the *degree* of vertex $v \in V$ is the number of edges incident to $v$. In notation, $\text{degree}(v) = |\{u \in V : \{v,u\} \in V\}|$. A vertex $v$ whose degree is 0 is called an *isolated* vertex, since

[1] Recall that the Cartesian produce of two sets $U$ and $V$ is defined as $U \times V = \{(u,v) : u \in U \text{ and } v \in V\}$.

there is no edge which connects $v$ to the rest of the graph. In a directed graph, the *in-degree* of a vertex $v$ is the number of edges from other vertices to $v$, and the *out-degree* of $v$ is the number of edges from $v$ to other vertices.

# 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. Euler did this by modeling the arrangement of bridges as a graph (or, more accurately, a *multigraph* since there can be multiple edges between the same pair of vertices) and asking whether a certain type of path could exist in the graph. For this reason, Euler is generally 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*). An *Eulerian tour* or *Eulerian cycle* is an Eulerian path whose starting point is the same as its ending point.

The next theorem gives necessary and sufficient conditions for a graph to have an Eulerian tour.

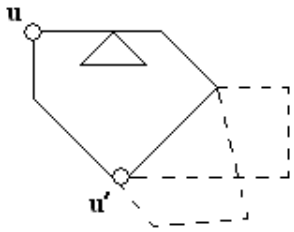
**Euler's Theorem:** An undirected graph $G = (V, E)$ has an Eulerian tour if and only if the graph is connected (except possibly for isolated vertices) and every vertex has even degree.

**Proof**: (proof of $\Longrightarrow$): Assume that the graph has an Eulerian tour. This means every vertex that has an edge adjacent to it (i.e., every non-isolated vertex) must lie on the tour, and is therefore connected with all other vertices on the tour. This proves that the graph is connected (except for isolated vertices). Next note that, for each vertex in the tour except the first (and last), the walk leaves it just after entering. Thus, every time the tour visits a vertex, it traverses exactly two edges (an even number). Since the Eulerian tour uses each edge exactly once, this implies that every vertex except the first has even degree. And the same is true of the first vertex $v$ as well because the first edge leaving $v$ can be paired up with the last edge returning to $v$. This completes the proof of this direction.

(proof of $\Longleftarrow$): Assume that $G = (V, E)$ is connected and all its vertices have even degree. We will show how to construct an Eulerian tour in $G$. We do this as follows:

- Suppose we start walking from some vertex $u$, never repeating edges, until we get stuck (because there is no unused edge out of our current vertex). We claim that we will get stuck only at $u$. This gives us a closed walk (starting and ending at $u$) that does not necessarily traverse all the edges.
- Suppose we are stuck. If there are any remaining untraversed edges, we pick one of them $\{u', v'\}$ with one endpoint $u'$ on the current closed walk (this must exist since the graph is connected). We then repeat the first step starting from $u'$ to get another closed walk (starting and ending at $u'$), and we "splice this in" to the previous walk at vertex $u'$ (see figure below). We repeat this step until all edges

are traversed.




□

Why is the second step necessary? As the figure above shows, we cannot be certain that when we get stuck at $u$, we will have traversed every edge in the graph, so we may indeed need to splice in additional closed walks. [Exercise: Convince yourself that the additional walks can be spliced in to create an Eulerian tour.]

In the first step, how can we be sure that we will get stuck at $u$? This follows from a fact we will prove below by induction: in any walk from $u$ to $v$, with $u \neq v$, the only vertices with odd degree are $u$ and $v$. This means that, in the walk in the first step, whenever we visit a vertex (other than $u$), there must always be an unused edge that we can take out of the vertex (since the vertex has even degree, and is currently the end point of our walk so only an odd number of the edges at the vertex are so far on the walk). Thus we cannot get stuck at any vertex other than $u$, so we must get stuck at $u$.

We now go back and prove the fact mentioned above.

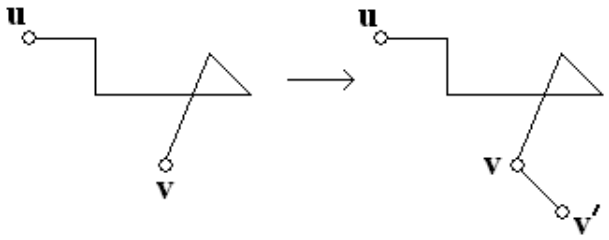
**Claim:** In any walk from $u$ to $v$, with $u \neq v$, the only vertices with odd degree are $u$ and $v$.

**Proof**: We prove by induction on $n$ the following proposition $P(n)$: in a length-$n$ walk from $u$ to $v$, $u \neq v$, the only vertices of odd degree are $u$ and $v$.

*Base case:* We prove $P(1)$. This is easy, since the walk consists of a single edge from $u$ to $v$. Clearly each of $u$ and $v$ has degree 1 in this walk.

*Inductive hypothesis:* Assume $P(n)$ is true. That is, in a length-$n$ walk from $u$ to $v$, the degrees of $u$ and $v$ in the walk are odd, and the rest of the vertices have even degree.

*Inductive step:* We prove $P(n+1)$. Suppose the length-$(n+1)$ walk starts at $u$, and that its last edge is $\{v,v'\}$, so that it ends at $v'$. By the inductive hypothesis, the walk traversing the first $n$ edges from $u$ to $v$ has two vertices ($u$ and $v$) of odd degree with the rest being even. So when we add the $(n+1)$st edge $\{v,v'\}$, the degree of $v$ becomes even and the degree of $v'$ becomes odd. This completes the proof by induction.




□

# Hamiltonian Paths and Cycles

A *Hamiltonian path* in an undirected graph $G = (V,E)$ is a path that goes through every vertex *exactly once*. A *Hamiltonian cycle* (or *Hamiltonian tour*) is a cycle that goes through every vertex exactly once. Note that,

in a graph with $n$ vertices, a Hamiltonian path consists of $n-1$ edges, and a Hamiltonian cycle consists of $n$ edges. See below for further details on this.

The remainder of this Lecture Note is not required material. It shows several (optional) applications of these concepts. You may read it for your own interest, if you like, but you are also free to skip it. It will not be required on the exams.

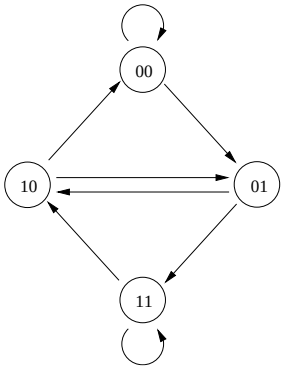
# de Bruijn Graphs

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 for the case $n = 3$:


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, then the two outgoing edges would be directed toward the vertices 100, and 101. 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: all we need to modify is the second condition, which should now say: "for every vertex $v$ in $V$, the in-degree of $v$ equals the out-degree 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 and add the corresponding bit which was shifted in from the right as we traverse each edge. Here is the de Bruijn graph for $n = 3$. Exercise: Find the Eulerian tour of this graph that generates the de Bruijn sequence given above.



# Hypercubes

Recall that the set of all $n$-bit strings is denoted by $\{0,1\}^n$. The $n$-dimensional hypercube is an undirected graph whose vertex set is $\{0,1\}^n$ (i.e., there are exactly $2^n$ vertices, each labeled with a distinct $n$-bit string), and with an edge between vertices $x$ and $y$ iff $x$ and $y$ differ in exactly one bit position. In other words, if $x = x_1x_2\ldots x_n$ and $y = y_1y_2\ldots y_n$, then there is an edge between $x$ and $y$ iff there is an $i$ such that $\forall j \neq i, x_j = y_j$ and $x_i \neq y_i$.

There is another equivalent recursive definition of the hypercube:

The $n$-dimensional hypercube consists of two copies of the $n-1$-dimensional hypercube (the 0-subcube and the 1-subcube), and with edges between corresponding vertices in the two subcubes. i.e. there is an edge between vertex $x$ in the 0-subcube (also denoted as vertex $0x$) and vertex $x$ in the 1-subcube (denoted $1x$).

**Claim:** The total number of edges in an $n$-dimensional hypercube is $n2^{n-1}$.

**Proof**: Each vertex has $n$ edges incident to it, since there are exactly $n$ bit positions that can be toggled to get an edge. Since each edge is counted twice, once from each endpoint, this yields a grand total of $n2^n/2$. □

**Alternative Proof:** By the second definition, it follows that $E(n) = 2E(n-1) + 2^{n-1}$, and $E(1) = 1$. A straightforward induction shows that $E(n) = n2^{n-1}$. □

# Gray Codes: Hamiltonian Cycles in the Hypercube

**Theorem:** For every $n \geq 2$, the $n$-dimensional hypercube has a Hamiltonian cycle.

**Proof**: By induction on $n$. In the base case, $n = 2$, the 2-dimensional hypercube, the cycle starts at 00, goes through 01, 11, and 10, and returns to 00.

Suppose now that every $(n-1)$-dimensional hypercube has a Hamiltonian cycle. Let $v \in \{0,1\}^{n-1}$ be a vertex adjacent to $0^{n-1}$ (the notation $0^{n-1}$ means a sequence of $n-1$ zeroes) in the Hamiltonian cycle in a $(n-1)$-dimensional hypercube. The following is a Hamiltonian cycle in an $n$-dimensional hypercube: have a path that goes from $0^n$ to $0v$ by passing through all vertices of the form $0x$ (this is simply a copy of the Hamiltonian path in dimension $(n-1)$, minus the edge from $v$ to $0^{n-1}$), then an edge from $0v$ to $1v$, then a path from $1v$ to $10^{n-1}$ that passes through all vertices of the form $1x$, and finally an edge from $10^{n-1}$ to $0^n$. This completes the proof of the Theorem. □

When we start from $0^n$ and we follow the Hamiltonian cycle described in the above proof, we find an ordering of all the $n$-bit binary strings such that each string in the sequence differs from the previous string in only one bit. Such an ordering is called a *Gray code* (from the name of the inventor) and it has various applications in error correction schemes and other fields. You can read about some of these in the wikipedia entry (`http://en.wikipedia.org/wiki/Gray_code`).