CMPSCI 250: Introduction to Computation

Lecture #16: Paths and Connectivity
David Mix Barrington
28 March 2013

Paths and Connectivity

- Review: Representations and Isomorphism
- Directed and Undirected Paths
- Examples of Paths and Distance
- Connectedness in Undirected Graphs
- Strong Connectedness in Directed Graphs
- Articulation Points
- Counting Paths With Matrices
- The Path-Matrix Theorem

Review: Representations and Isomorphism

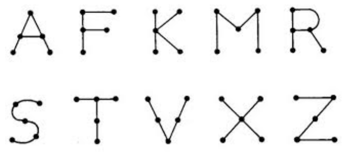
- Last time we defined a **graph** to be an object with **vertices** connected by **edges**, either **undirected** or **directed**. A **simple graph** is undirected and cannot have **parallel edges** or **loops**. Several other types of graphs relaxed these restrictions.

- We can represent an n-node graph as an **adjacency matrix**, an n by n matrix with rows and columns indexed by the vertices, where the (u,v) entry is 1 if there is an edge from u to v and is 0 otherwise. We also discussed the **adjacency list** and the **incidence matrix**.

- Two graphs G and H are **isomorphic** if there is a function (called an **isomporphism**) from the vertices of G to the vertices of H such that (1) f is a one-to-one, onto function and (2) there is an edge from u to v in G if and only if there is an edge from f(u) to f(v) in H.

iClicker Question 1: Graph Isomorphism

- Look at the ten graphs below. **One** of the following four statements is **false** -- **which one**?
- (a) Each of these graphs is isomorphic to at least one of the others.
- (b) Graphs M, S, V, and Z are all isomorphic to one another.
- (c) Graphs F, K and T are all isomorphic to one another.
- (d) Graphs A and R are the only ones that are not bipartite.



Directed and Undirected Paths

- A **path**, in either an undirected or a directed graph, is a sequence of edges where the destination of each edge is the source of the next edge. For example, a sequence of edges (u_0, u_1), (u_1, u_2),..., (u_{k-1},u_k) forms a path of k edges from vertex u_0 to vertex u_k. If {u, v} is an edge of an undirected graph, the path may use either (u, v) or (v, u).

- A path from a vertex to itself is called a **circuit**. Circuits include the **trivial path** from any vertex to itself, with no edges at all.

- A path is called **simple** (by Rosen, at least) if it never contains the same *edge* more than once. Others, including today's version of Wikipedia, use this word for paths that never revisit the same *vertex*.

- The **length** of a path is the number of *edges* in it, not the number of vertices.

Examples of Paths and Distance

- The **distance** from vertex u to vertex v is the length of the shortest path from u to v, if there is such a path. Otherwise the distance is undefined.

- We can model a game such as Rubik’s Cube as a graph with a vertex for each position and an edge for each legal move. A position is solvable if there exists a path from it to the goal position, and the distance to the goal position (if it is defined) is the smallest number of moves to solve the puzzle.

- A **collaboration graph** has vertices for people and an edge between any two people who have collaborated. An actor’s **Bacon number** is their distance to Kevin Bacon in a collaboration graph where “collaboration” means “have been in the same film”. A mathematician’s **Erdos number** is their distance to Paul Erdos where collaboration means “have co-authored a paper”. (Mine is 3.) A few people, such as actor Natalie Portman, have both a defined Erdos number and a defined Bacon number -- see the Wikipedia article on “Erdos-Bacon number” for more examples.

Connectedness in Undirected Graphs

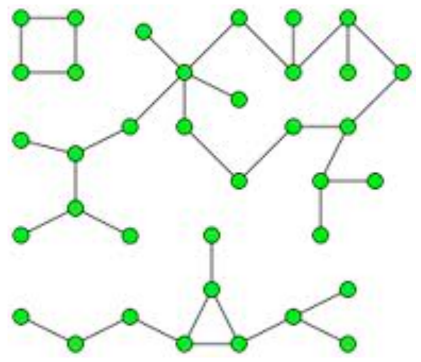
- An undirected graph is defined to be **connected** if there is a path from any vertex to any other vertex.

- In CMPSCI 187 we saw two methods to determine whether a structure is connected -- **depth-first search** and **breadth-first search**. Each method successively marks vertices that are reachable from the start vertex of the search. In both cases we put a vertex on an **open list** as soon as we find an edge to it. Later we take that vertex off the open list and check all of its edges to see whether more vertices should go on the open list. In DFS the open list is kept as a **stack**, and in BFS it is kept as a **queue**.

- A **connected component** in an undirected graph is the set of nodes that have paths to some given node. A DFS or BFS marks all the nodes in the connected component of the start node. A graph is connected if and only if all the vertices are in the same connected component.

iClicker Question #2: Connected Components

- How many connected components does the graph below have?
- (a) One
- (b) Two
- (c) Three
- (d) Four



Strong Connectedness in Directed Graphs

- In a directed graph, it is possible for there to be a path from vertex u to vertex v without there being a path from v to u. We thus have to be more careful about the definition of connectedness. We define a directed graph to be **strongly connected** if there is a path from any vertex u to any vertex v.
- We still have **strongly connected components** with this notion -- the strongly connected component of vertex u consists of all those vertices v such that there are paths *both* from u to v *and* from v to u.
- One strongly connected component may have a path to another, or a path from another, or neither, but not both.
- A DFS or BFS starting at vertex u will find all vertices reachable from u. To find strongly connected components efficiently requires a trick that you will see in CMPSCI 311.

Articulation Points

- Some connected graphs are more connected than others. Let's look at simple graphs to keep things easier. An **articulation point** is a vertex v in a connected graph G, such that G is no longer connected if we remove v (and its accompanying edges).

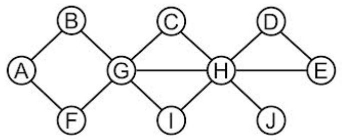
- For v to be an articulation point, there must be two other vertices u and w such that every path from u to w goes through v. (Thus u and w are in different connected components after v is removed.)

- A simple graph (with three or more vertices) is called **biconnected** if it is connected and has no articulation points. It's called **k-connected** if you can remove any k-1 points and the remaining graph is still connected.

- A multiply connected graph is clearly better for a communication network, because it can more easily tolerate node failures.

iClicker Question #3: Articulation Points

- What is the set of articulation points of the graph below?
- (a) G and H only
- (b) A, G, and H
- (c) A, G, H, and E
- (d) Every vertex except J



Counting Paths With Matrices

- We called our adjacency matrices “matrices” rather than just arrays, because treating them as matrices has a useful computational meaning. Remember that in a directed multigraph, the (u, v) entry of the adjacency matrix A tells us how many edges there are from u to v.

- Because A is an n by n matrix, we can **multiply** it by itself. In fact, we can define powers of A by a recursive definition similar to that for powers of integers. A^0 is defined to be the n by n **identity matrix I**, with 1’s on the main diagonal and 0’s everywhere else. (You should know that AI = IA = A for any A.) Then for any i, we define A^{i+1} to be A^i times A. So, for example, A^3 is A times A times A, just as x^3 is x times x times x.

- We’ll now prove that the entries of A^t have a specific meaning in the graph. The **Path-Matrix Theorem** says that the (u, v) entry of A^t is the number of paths of length t from u to v. We’ll prove this by induction on t.

iClicker Question #4: Counting Paths

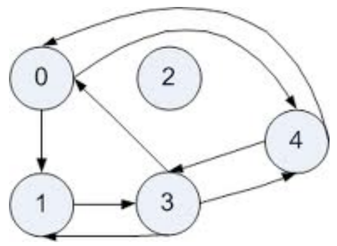
- How many three step paths are there from vertex 4 to vertex 3 in this directed graph?

- (a) two

- (b) three

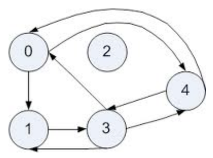
- (c) four

- (d) more than four



Counting the Paths With Matrices

- The red (4,3) entry of A^3 gives the number of three-step paths from vertex 4 to vertex 3.



$$\underset{A}{\begin{matrix}0&1&0&0&1\\0&0&0&1&0\\0&0&0&0&0\\1&1&0&0&1\\1&0&0&1&0\end{matrix}}\qquad\underset{A^2}{\begin{matrix}1&0&0&2&0\\1&1&0&0&1\\0&0&0&0&0\\1&1&0&2&1\\1&2&0&0&2\end{matrix}}\qquad\underset{A^3}{\begin{matrix}2&3&0&0&3\\1&1&0&2&1\\0&0&0&0&0\\3&3&0&2&3\\2&1&0&4&1\end{matrix}}$$

Proving the Path-Matrix Theorem

- The base case of $t = 0$ is easy. A path of length 0 can have no edges, so it must go from a vertex u to itself. There is exactly one length-0 path from u to itself, and no length-0 paths from any vertex to any *other* vertex. Thus the number of length-0 paths from u to v is just the (u, v) entry of the matrix $I = A^0$.

- Now assume that for any u and v, the (u, v) entry of A^i gives the number of length i paths from u to v. We need to show that the number of length i+1 paths from u to v is exactly the (u, v) entry of the matrix $A^{i+1} = A^i \cdot A$. By the definition of matrix multiplication, $(A^{i+1})_{u,v}$ is the sum, over all vertices w, of $(A^i)_{u,w}$ times $(A)_{w,v}$.

- But how can we have a path of length i+1 from u to v? It must follow i edges to some vertex w, then continue on a last edge to v. How many ways are there to do this? For each w, there are $(A^i)_{u,w}$ length-i paths from u to w, and $(A)_{w,v}$ edges from w to v. The total number is exactly the sum given for $(A^{i+1})_{u,v}$.

The General Path-Matrix Theorem

- In our proof of the Path-Matrix Theorem, we used only a few basic facts about “addition” and “multiplication”, such as the distributive law. What this means is that the theorem holds even if we redefine “addition” and “multiplication”, as long as our new operations follow those laws.

- We proved that the (u, v) entry of A^t is the “sum”, over all paths of length exactly t from u to v, of the “product” of the labels of the edges on the path. In the path-counting version, the label was the number of edges, so the product of the labels was the number of ways to visit a particular sequence of vertices.

- If we redefine “sum” to mean OR and “product” to mean AND, we get that $(A^t)_{u,v}$ is 1 if there is *any* path of length t from u to v, and 0 if there is no path. By combining the matrices A^i for all i less than n, we can find out whether there is a path of any length.

- In Monday’s discussion we’ll use a similar idea to find **shortest paths**.