MATHEMATICS 152, FALL 2003
METHODS OF DISCRETE MATHEMATICS
Outline #11 (Graphs)

This is based on Biggs' Chapter 15 and the attached notes, which were originally prepared for the Summer School course Computer Science S-111. Feel free to ignore the C code in the notes, which is taken from the program miles.exe (on the course Web site and distributed by email.)

Some of these presentations are quite short, but that way everyone gets to do one.

For many of the proofs requested below, the only thing that is hard is to figure out what needs to be proved! Ask yourself what could go wrong, and devise a proof that rules it out.

Transparencies made from the notes can be available if you want to use them for your presentation. Send email to bamberg@tiac.net before 8:30 AM on the day of your presentation, specifying what page you want.

The program miles.exe comes with lots of data files. If you want to run it on your Windows computer, download miles.zip from the course Web site and unzip everything into a single directory.

1. Present Biggs' definition of a graph (p. 178). Invent a graph with no more than 5 vertices and no more than 6 edges and represent it by a list of sets, a picture, and an adjacency list.

   State how this definition might prove inadequate for the following cases and suggest how it could be modified to cover them.

   - Vertices represent islands, and edges represent bridges between them. There may be two bridges between the same pair of islands.
   - Vertices represent elements of a group, and the edge joining a to b represents the element g such that $ga = b$.
   - From an existing graph, a new graph is formed by deleting a vertex that has two edges connected to it and combining those edges into a single edge.

2. Define the complete graph K_n (Biggs, p. 179, exercise 3). Draw pictorial representations for K_3 and for K_4 in which edges do not cross. Show that this cannot be done for K_5, even if the rules are relaxed to allow edges to be represented by curves instead of straight line segments.

3. Define what is meant by isomorphism of graphs (Biggs, section 15.2). Invent an example of two graphs that are isomorphic (even though it may not be apparent from their pictorial representations) and of two graphs that have equal numbers of vertices and edges but that are not isomorphic.

4. Define the valency (sometimes also called the degree) of a vertex of a graph, and prove that the number of odd vertices is even (Biggs, p. 182).

5. Define the terms "walk," "path," and "cycle" for a graph. Define "Hamiltonian cycle," "Eulerian walk," and "Eulerian cycle." State and prove a necessary condition for a connected graph to have an Eulerian walk and to have an Eulerian cycle.

6. Define a tree T as a connected graph with no cycles (Biggs, p. 185). Prove the following properties of trees:

   (a) for each pair of vertices x and y, there is a unique path in T joining x and y.

   (b) the graph obtained from T by removing an edge has two components, each a tree.

   (c) $|E| = |V| - 1$.

7. Given a connected graph whose vertices all have even valency (degree), describe an algorithm for constructing an Eulerian cycle, and prove that the algorithm always works. This establishes that the condition of no odd-degree vertices is also sufficient.

   This algorithm and proof are not done in Biggs. The secret of the proof is to use "strong induction." Use $n = 3$ as your base case. You may assume that the result is true for all integers between 3 and $n-1$ inclusive (that's what is meant by "strong induction")in proving it for case n.

   Here is the algorithm, which makes lots of use of the "axiom of choice."

   - Choose a starting vertex, and follow edges until you return to that vertex. Delete all the edges in this original cycle from the graph, and delete any vertex that no longer has any edges leading to it.
   - This may leave one or more connected graphs of lower degree. In each of these graphs, choose one vertex from the original cycle. As you traverse the original cycle, make a detour to carry out (recursively, by the same algorithm) an Eulerian cycle, using edges that are not in the original cycle, that starts and ends at that vertex. The algorithm is recursive; the proof is inductive!

   To prove the algorithm correct, you must show that

   - Each smaller graph for which you do an Eulerian cycle is connected ,has at least three vertices, and has no vertices of odd degree.
   - After you delete the original cycle, every remaining edge is in a connected graph that includes a vertex from the original cycle.

To illustrate the algorithm, choose a case where deleting the first cycle splits the original graph into two disconnected graphs.

Implementing this algorithm is programming project #4. You can download instructions from the course Web site. The folks in the Science Center suggest that you build the project in a subdirectory of /temp, but if you do this, remember to copy it to your home directory before you log out. Otherwise it will disappear!

8. Describe the algorithm for carrying out a preorder depth-first search (traversal) of a graph. (This is also discussed in Biggs, section 16.4). Illustrate it using the graph shown on page 10-8 of the notes. You may use Boston as the starting city, as shown in the notes, or you may choose a different starting city. The pages omitted from the notes, which were all about implementation details, point out that the roads from a city are arranged in order of increasing length. This fact will let you replicate the results of miles.exe on the file short.aaa.

9. Invent an inductive proof that for a finite, connected graph, a depth-first search starting from city A will eventually reach city B. Hint: choose a path joining A to B, and use induction on the length of the path.

10. Describe the algorithm for carrying out a breadth-first search (traversal) of a graph. (This is also discussed in Biggs, section 16.5). Illustrate it using the graph shown on page 10-11 of the notes. You may use Boston as the starting city, as shown in the notes, or you may choose a different starting city. Knowing that the roads from a city are arranged in order of increasing length, you can replicate the results of miles.exe on the file short.aaa.

11. Invent a proof that for a finite connected graph, a breadth-first search starting from city A will eventually reach city B. Hint: choose a path of shortest length joining A to B, and use induction on the length of the path.

12. Describe Dijkstra's algorithm for finding the shortest path between two vertices of a weighted graph, and explain precisely how it finds the shortest path from Providence to Portland in the mileage graph shown on page 10-15 of the notes. (Use these specific cities, which illustrate the key step in the algorithm.) (This is also discussed in Biggs, section 16.6).

13. Prove that Dijkstra's algorithm finds the shortest path between any two vertices in a connected weighted graph, as long as all weights are positive. Invent a simple graph (three vertices, with a negative weight on one edge, is enough) in which there exists a well-defined shortest path between any two cities, but Dijkstra's algorithm fails to find it. (Think of weight as the toll rather than the distance between cities, and this becomes outlandish but not impossible.)

14. Define "minimum spanning tree" and state and prove the minimum spanning tree property (page 10-18 of the notes). (This is the first part of the proof of Theorem 16.3 in Biggs, but it makes sense to do the proof for an arbitrary set S of vertices.)

15. Describe Prim's algorithm for constructing a minimum spanning tree. Illustrate its action on the graph shown on page 10-19 of the notes, and prove that it is correct. The proof is easy, since the minimum-spanning-tree property has already been proved. (This algorithm, for a non-weighted graph, is described at the start of Biggs, section 16.3.)

16. Describe Kruskal's algorithm for constructing a minimum spanning tree. Illustrate its action on the graph shown on page 10-21 of the notes, and prove that it is correct. The proof is easy, since the minimum-spanning-tree property has already been proved. (This is the "greedy algorithm" that Biggs describes on pp. 200-201.)

Assignments:

1. Eugene
2. Tali
3. Sam
4. Vivian
5. Meagan
6. Christine
7. Mark Z. (this may take some work!)
8. Mark K.
9. Wei
10. Brian
11. Ed
12. Amie
13. Dennis
14. Zachary
15. Greg
16. Stephanie