Reminders

1. Homework #7 (h07) is due today.
2. Homework #8 (h08) is due Wednesday, May 5, at 11:59 pm.
3. Test 2 is Thursday, May 6.
4. Check Blackboard to see if you have every grade you have submitted.

Coding Algorithms

1. Log in to your Sage/CoCalc account.
   (a) Start the Chrome browser.
   (b) Go to http://cocalc.com and sign in.
   (c) You should see an existing Project for our class. Click on that.
   (d) Click “New”, call it s12, then click “Sage Worksheet”.
   (e) For each problem number, label it in the Sage cell where the work is. So for Problem 1, the first line of the cell should be #Problem 1.
   (f) When you are finished with the worksheet, click ”make pdf”, email me the pdf (at clarson@vcu.edu, with a header that says Math 356 s12 worksheet attached).

Saving and Re-using Code

We’ve coded several graphs now, and have added code for functions of graph invariants and auxiliary functions and stored them in “graphs.sage”. I pushed my updated version to your Handouts folder. Either copy that file to your Home directory—or add the new stuff to your own “graphs.sage” file. We’ll need those functions.

2. I’ve updated the copy of “graphs.sage” in your Handouts folder to include what we’ve added in class. Copy the current version from Handouts to your Home directory.
3. Load your copy of “graphs.sage”. Run: load(‘graphs.sage’).
4. Use the is_eulerian function we coded to see which graphs in my_graphs are eulerian.
Euler Tours

An Euler tour in a graph is a closed walk that contains every edge of the graph (so vertices may be repeated, every edge will be used exactly once, and we return to the starting vertex. (Since it is a circuit we can of course began at any vertex). If it does we say the graph is Eulerian. (Note too this is a graph property: either a graph is Eulerian or it is not).

Not every connected graph has an Euler tour—the Bull graph for instance does not. How can we test if a graph has an Euler circuit? Or better, find an Euler circuit in the case that it does not?

A first observation is that if a graph has an Euler tour every degree must be even (because the number of times our trail enters a vertex must equal the number of times it leaves that vertex). So that is a necessary condition. In fact, we will prove that this (together with being connected) is also a sufficient condition. That is, we’ll prove: A graph is Eulerian if and only if it is connected and every vertex has even degree.

We proved that we can find an Eulerian circuit in a connected graph whose vertices all have even degree. An algorithmic idea is to start at any vertex, greedily find a cycle (we argued that it must return to that very same vertex), and then extend that cycle.

How? How can we extend the cycle? If the cycle doesn’t contain every graph edge, then some cycle vertex $v$ must have adjacent edges. Delete the cycle and check the degrees of the cycle vertices. Find a new cycle, add it to the existing cycle and repeat.

Here’s how we’ll break all this down (assuming our graph has an Eulerian tour):

(a) Write a function $\text{find\_cycle}(g,v)$ that takes a graph $g$ and vertex $v$ and greedily finds a cycle starting and ending with $v$.

(b) Write a function $\text{find\_remaining\_subgraph}(g,C)$ that takes the original graph $g$ and the cycle $C$ found so far, deletes the cycle edges, and returns the remaining subgraph.

(c) Write a function $\text{find\_start\_vertex}(h,C)$ that takes a (non-empty) graph $h$ (subgraph of original graph $g$) and cycle $C$ and returns a cycle vertex $v$ that has positive degree in $h$ (this will be the start vertex for extending $C$).

(d) Write a function $\text{extend\_cycle}(h,C,v)$ that takes a (non-empty) graph $h$ (subgraph of original graph $g$) and cycle $C$, a vertex $v$ of $C$ of positive degree, finds a cycle $C'$ in $h$ starting at $v$, and returns the cycle formed by gluing $C$ and $C'$ together.

(e) Write a function $\text{find\_euler\_tour}(g)$ by putting these auxiliary functions (“ingredients”) together.

Here’s the code:
def find_cycle(g,v):
    Remaining_edges = list(g.edges(labels=False))
    V = g.vertices()
    Cycle = [v]

    N = neighbors(g,v)
    u = N[0]
    Cycle.append(u)
    if (u,v) in Remaining_edges:
        Remaining_edges.remove((u,v))
    if (v,u) in Remaining_edges:
        Remaining_edges.remove((v,u))
    New_graph = Graph([V,Remaining_edges])

    while True:
        current = u
        N = neighbors(New_graph,current)
        u = N[0]
        if u == v:
            return Cycle
        Cycle.append(u)
        if (u,current) in Remaining_edges:
            Remaining_edges.remove((u,current))
        if (current,u) in Remaining_edges:
            Remaining_edges.remove((current,u))
        New_graph = Graph([V,Remaining_edges])

def find_remaining_subgraph(g,C):
    Remaining_edges = list(g.edges(labels=False))
    V = g.vertices()
    cycle_length = len(C)
    for i in range(cycle_length-2):
        u = C[i]
        v = C[i+1]
        if (u,v) in Remaining_edges:
            Remaining_edges.remove((u,v))
        if (v,u) in Remaining_edges:
            Remaining_edges.remove((v,u))

    u = C[0]
    v = C[cycle_length-1]
    if (u,v) in Remaining_edges:
        Remaining_edges.remove((u,v))
    if (v,u) in Remaining_edges:
        Remaining_edges.remove((v,u))

    New_graph = Graph([V,Remaining_edges])
    return New_graph
def extend_cycle(g,C):
    h = find_remaining_subgraph(g,C)
    u = find_start_vertex(h,C)
    C2 = find_cycle(h,u)
    start = C.index(u)
    newC = C[start:]+ C[:start] + C2
    return newC

def find_euler_tour(g):
    V = g.vertices()
    E = g.edges()
    v = V[0]
    C = find_cycle(g,v)
    while len(C)+1 < len(E):
        C = extend_cycle(g,C)
    return C

Euler Paths

A graph may not have an Euler tour, but may have a trail from some vertex \(v\) to another vertex \(w\) that traverses every edge exactly once called an Euler path. When will this happen?

5. Write a function has_euler_path(g) that tests if an input graph \(g\) has an Euler path.

Hamilton cycles

A graph has a Hamilton cycle if there is a cycle in the graph that includes every vertex (so a closed walk with no repeated edges or vertices). Such a graph is called Hamiltonian.

6. What is a test for whether a graph is Hamiltonian?