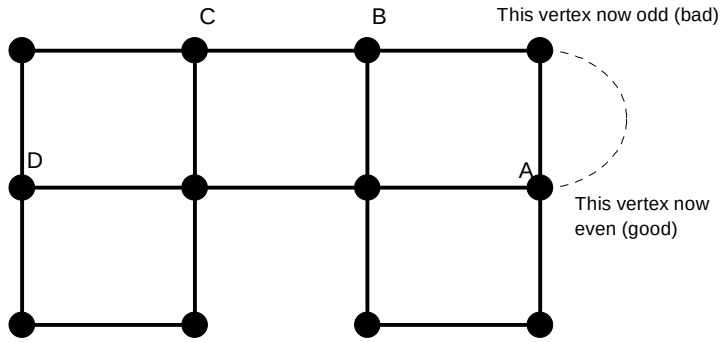
Math 203
Eulerization – Why and How

An Euler circuit, when it exists, describes the most efficient solution to any problem where tasks have to be done along the edges of a graph. Examples of such tasks abound if you try to think of them, for example mail or newspaper delivery, garbage collection, parking meter monitoring, street sweeping, even walking though a store's aisles looking for unadvertised specials (or through a gift shop looking for inspiration right before your significant other's birthday)! But many of the graphs involved don't have an Euler path, or even if it does we may have to start at the wrong vertex (if you're a mail carrier, you have to start at the post office, even if that isn't an odd vertex - you may have to end there too). However, if the streets in your neighborhood don't have an Euler circuit, is it okay with you if your garbage doesn't get collected? I (and your neighbors) hope not.

When the graph of edges we need to cover to perform an important task doesn't have the Euler path/circuit that meets our needs, what we must do is usually to cover paths not needed for the task and/or to cover parts of our path twice. In the graph, that corresponds to adding edges to represent this extra travel – the process of adding edges to get to a graph with an Euler path/circuit is called eulerizing. Any edge we have to cover a second time becomes an edge that gets duplicated in the graph – covering the edge and its duplicate really means traveling that edge twice. Any new path we take that wasn't in the graph becomes a new edge between the appropriate vertices on the old graph. (For example, in the newspaper delivery application there might be a block where nobody subscribes, so that isn't on our initial diagram, but that block can be added to the graph if it helps form an efficient route.) However, most of the time the edges already in the graph represent our options – adding a new edge between two vertices that don't already have an edge between them usually represents something like bulldozing everything between two intersections and building a new direct road connecting them. Normally, that's not feasible. So to simplify the setting in dealing with these problems, we'll limit ourselves to looking at changes in the graph that duplicate existing edges. We'll also assume for simplicity that all edges are equally costly to travel a second time (costly may be in terms of time, money, or effort), implying that the most efficient eulerization is the one that minimizes the number of edges we duplicate. (It's not that hard to tweak the process a little to fix things if either of these assumptions is false in a particular application.)

We'll focus first on problems where we'd like to have an Euler circuit, i.e., we want to cover every edge and end up at the vertex where we start. That means we need to add edges until every vertex is of even degree. What happens when we duplicate an edge? Both vertices at the ends of the edge change in degree parity (i.e. from odd to even or even to odd). If we have an odd degree vertex, and we duplicate an edge to make that vertex even, if the other end of the edge was an odd vertex also, great, we've made both

even. But if it was even, we've now made it odd, and need to duplicate another edge starting at that point to fix it. The process will continue until the edge we duplicate has an odd vertex at both ends. See the diagram below, where the dotted edge was just added to fix vertex A at right middle, but messes up the vertex at upper right. We must now add another edge starting at there. That means we're building a path starting at vertex A, and the path can end only when it gets to another odd vertex. What this says is that eulerizing requires pairing up the odd vertices, and connecting the two odd vertices in each pair by duplicating a path between them. Clearly each such path needs to be done in a way that is as efficient as possible, but also the pairing of odd vertices must be done intelligently, to keep the average distance between vertices in the same pair as small as possible. In the above diagram, the odd vertices originally are A, B, C, and D. We can pair A with B and C with D, for an average pair distance of 2, or we can pair A with D and B with C, and the average pair distance is still 2! But it would be silly to pair A with C and B with D, for an average pair distance of 3. One sure sign even before we find the average that the last pairing is a loser is that the edge from B to C could be duplicated twice by that pairing, once duplicating an optimal path from A to C and again duplicating an optimal path from B to D. If you ever find one edge duplicated twice in an eulerization, the eulerization is inefficient, because if both duplicates are eliminated, the vertices all remain of even degree and thus the graph is still eulerized without them.



METHOD FOR EULERIZING A GRAPH

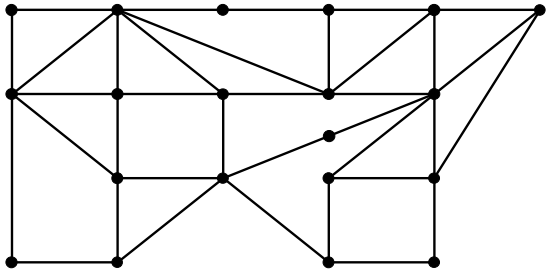
To eulerize a connected graph into a graph that has all vertices of even degree:

1) Identify all of the vertices whose degree is odd. (Recall that there must be an even number of such vertices.
2) Pair up the odd vertices, keeping the average of the distances (number of edges) between the vertices of the pairs as small as possible.
3) For each pair, duplicate all of the edges along an optimal path between those two vertices.

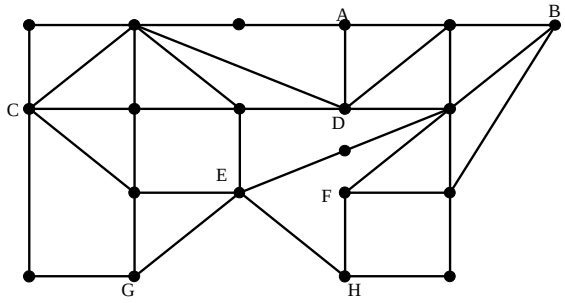
Note that if you are dealing with a graph which is a complete rectangular array of edges with at least 2 edges on each side, there is a particularly simple way to find a best-possible eulerization called the edge-walker algorithm, described in problem #50 on pages 359-360 of the text.

Once you've eulerized the graph, you can find an Euler circuit on the Eulerized graph and reinterpret that as a path on the original graph that efficiently meets your needs. Recall that if you've chosen the pairing correctly, no edge of the original graph will be duplicated more than once in the eulerized graph, so an edge in the original graph will be traveled on at most one extra time.

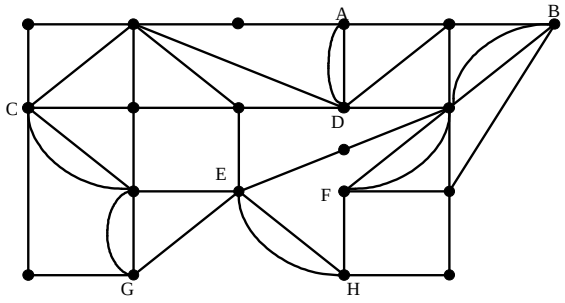
Example: Suppose we want a reasonable eulerization of the following graph:



First we must identify the odd vertices. They are the eight vertices labeled A through H in the copy of the graph below:



Note that we'll need 4 pairs, and that every odd vertex except C and B is adjacent to at least one other odd vertex (i.e., connected to it by a single edge). A and D are adjacent, as are F and H and G and E. In fact, E and H are also adjacent to each other, and thus each is adjacent to two odd vertices. With four pairs, knowing each pair will require at least one edge to be duplicated, we'd like to keep the number of edges we duplicate as close to 4 as possible. You might think we should pair A with D, E with G, and F with H to use as many of those adjacent odd vertices as possible, but if we did that, we'd have to pair B with C, and they're somewhat far apart – it requires at least 5 edges to get between them, meaning we'd need to duplicate 8 edges in all. We can try to do better than that, by checking whether we can pair up B and C with other closer odd vertices without messing up too many other odd vertices. The closest odd vertices to B are A, D, and F, which are each two edges distant, and the closest odd vertices to C are E and G, which are also 2 edges distant. If we pair up B with either A or D, we will then leave the other without a nearby odd vertex and have more edges to add, but not so if we pair up B with F. Likewise, if we pair up C with E, we are leaving G without an adjacent odd vertex to pair with, but if we pair C with G, we can still pair E with H. Thus our best pairing is A with D, B with F, C with G, and E with H. Our distances between the vertices in our pairs are 1, 2, 2, and 1, for a total of 6 edges to duplicate. In some cases, there can be a tie for the pairing that produces the best possible eulerization, but in this case we have found the only pairing requiring just 6 duplicated edges. One eulerized graph for this pairing looks like:

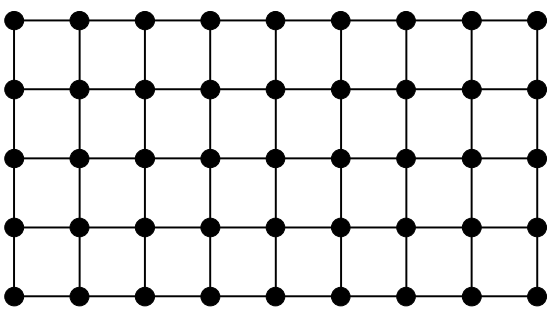


Note that with a complex diagram with many odd vertices, it may not always be obvious what the best pairing is, but it will often be obvious that certain pairings are not good. If the diagram is complex, we may ask you to find a decent pairing, but we won't expect you to prove you've found the best possible pairing.

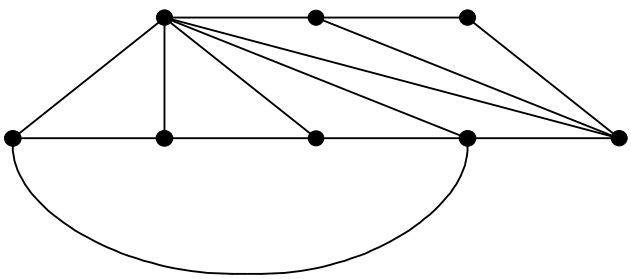
One final comment: What changes are necessary if we have a connected graph where we need to travel all of the edges but we don't want a circuit, just a path? I.e., what if we don't need or want to end up where we start? Let's be specific, suppose we wish to start at vertex A and end at a different vertex B, covering every edge at least once. The easy way to find an efficient route is to start by introducing an artificial edge E between A and B that is special in that edge E cannot itself be duplicated. Now eulerize the graph with edge E added, remembering E cannot be duplicated. The result has every vertex of even degree. Now remove edge E and *voila!*, we have a connected graph where A and B are the only two odd degree vertices. That means there is now an Euler path going from A to B on the graph, which interprets as an efficient path to cover every edge at least once on the original graph.

Problems:

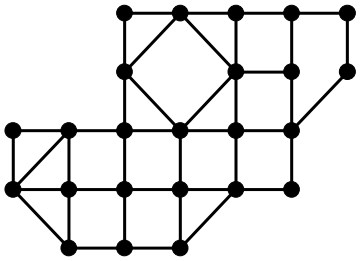
1. Find good eulerizations of each of the following graphs, so that you could efficiently cover each edge and end up where you start.



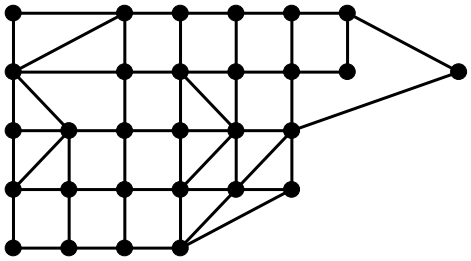
(a)



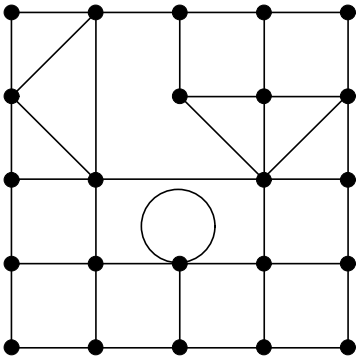
(b)



(c)



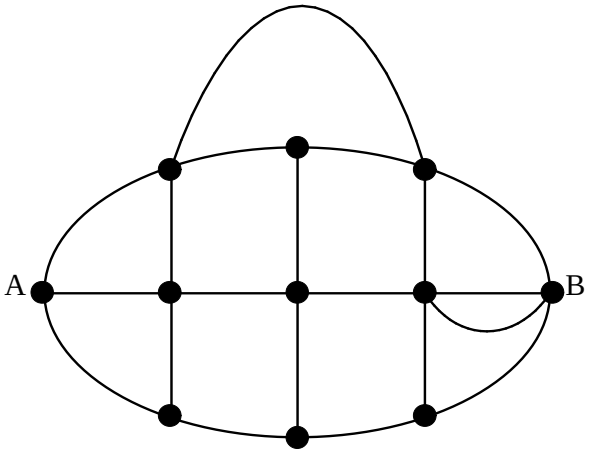
(d)



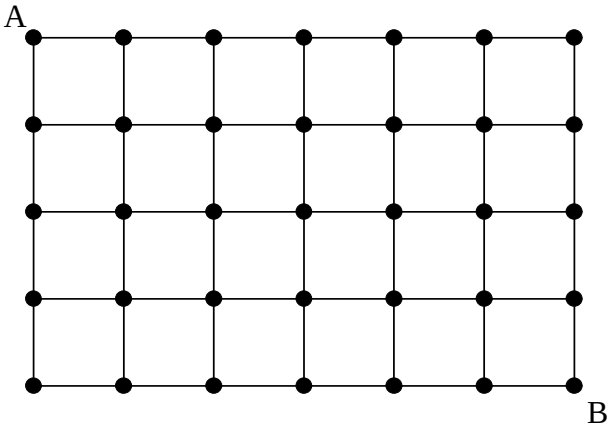
(e)

2. In 1 parts b, c, and e, find an Euler circuit on the modified graph you created.

3. Find a graph that would be useful for creating an efficient path that starts at vertex A and ends at vertex B for each of the following graphs. Then find an Euler path starting at A on the modified graph.



(a)



(b)

4. Using the eulerized graphs:
   a) In problem 1(e), indicate an efficient path on the original graph that starts and ends in the upper left corner and covers each edge at least once, by numbering the edges in the order in which they are used.
   b) In problem 3(a), indicate an efficient path on the original graph that starts at vertex A and ends at vertex B and covers each edge at least once, by numbering the edges in the order in which they are used.