1 Walking in graphs

Recall our definition of a *walk*:

Definition 1.1. Suppose that G is a graph. A sequence of edges

$$e_1, e_2, \ldots, e_i, e_{i+1}, \ldots, e_n$$

is a *walk* in G of length n if there are vertices $v_0, v_1, \ldots, v_n$ so that for all i the edge e_i connects v_{i-1} to v_i.

If we never repeat an edge in the walk then we call it a *trail*. If we never repeat a vertex in the walk (except for possibly allowing $v_0 = v_n$) then we call it a *path*. If in fact $v_0 = v_n$ then we call the walk, trail, or path *closed*. Another name for a closed path is a *cycle*.

It is often useful to decorate the edges of a graph with *weights*. That is, in addition to G we are also given a function $l\colon E(G) \to \mathbb{R}_{\geq 0}$ which assigns to every edge some non-negative real number. The quantity $l(e)$ will tell us something about the length of e, or the cost to cross e, or the time to cross e, or some other property. If we are given a walk $W = e_1 e_2 \ldots e_n$ then $l(W) = \sum_i l(e_i)$ is the total weight of W. Of course, if all weights are equal to one, then $l(W)$ is just the number of edges that W crosses.

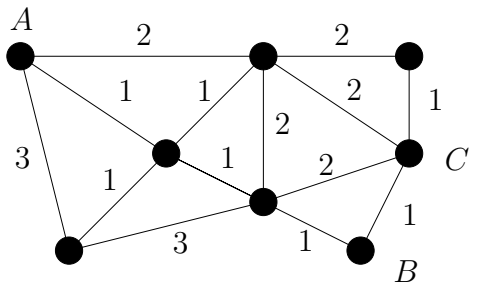

Figure 1: The shortest path from A to B has length 3 while the shortest path from A to C has length 4.

To motivate all these definitions we offer the following problems: Suppose that a graph G is given and $l\colon E(G) \to \mathbb{R}$ tells us the lengths of the edges of G.

Problem 1.2. The Chinese postperson problem: Given a vertex v find closed walk which starts and ends at v, crosses every edge at least once, and is as short as possible.

Problem 1.3. The travelling saleperson problem: Given a vertex v find closed walk which starts and ends at v, visits every vertex at least once, and is as short as possible.

Problem 1.4. The shortest path problem: Given two vertices v and w of G find the shortest path in G connecting v to G.

We will deal with each problem in turn. The first two are *NP-complete*: there is no known *algorithm* to solve the problem *efficiently*. The last problem is fairly easy to solve, with a good idea. We will briefly discuss these issues next week.

2 Algorithms

We have given several problems above but have not yet said what constitutes a solution. We surely do not want to be given graph after graph and asked over and over to solve the postperson problem, say. What is wanted is a purely mechanical method which takes *any* graph and produces the right answer. It may require insight to find the method but the method itself should be capable of being programmed into a computer, say.

To make matters concrete, consider the following:

Problem 2.1. Given two graphs G and H decide if G is isomorphic to H.

We do not want to solve the isomorphism problem over and over again. Instead we would rather think deeply any find one infallible method, or *algorithm* which a computer could carry out. To be precise, the method would receive the graphs G and H, perhaps encoded in some standard fashion, would carry out various computations, and output the desired answer which in this case is either "yes" or "no".

As we have discussed, there is such an algorithm which solves the isomorphism problem. Here it is: take the graphs G and H and check if $|V(G)| = |V(H)|$. If not then the graphs are not isomorphic. If so then list all bijections of $V(G)$ with $V(H)$. (There are $n! = n \cdot (n-1) \cdot (n-2) \ldots 3 \cdot 2 \cdot 1$ of these.) For each bijection, check if it gives an isomorphism. If it does stop and print "yes". If none of the bijections give an isomorphism print "no".

This is an algorithm, but not a very good one – it takes "exponential time" in the size of the inputs G and H. Sometimes this kind of algorithm is called a *brute force* algorithm or an *exponential search* algorithm. Unfortunately, there does not seem to be, at present, an "efficient" solution to the graph isomorphism problem.

3 Euler

We begin by considering the Chinese post-person problem (CPP, for short). Is the problem solvable? Are there graphs where the problem has a nice solution? How many solutions does the problem have?

To answer the first question we can again appeal to an exponential search algorithm. Suppose we are given a graph G. We begin by finding *some* closed walk, W, which crosses every edge at least once. (Possibly using the shortest path problem, or one of the tree-traversal techniques, to be discussed later.) This gives an upper bound, say $L = l(W)$, on the length of the closed walk which actually solves the problem. Next we build a list $\{W_i\}$ of *all* walks of length up to L. For each of these of the W_i we check if it is closed and if crosses every edge. If not we discard the path. For all of the W_i passing the test we compute $l(W_i)$ After all of this any walk on our list with shortest length is a solution to the problem.

Again, the fly in the ointment here is the size of the list $\{W_i\}$. Since this is certain to be huge the brute force approach will take far too long for any but the smallest of graphs. Notice that the brute force solution doesn't give us much insight into the problem. Let us turn instead to an important special case. For the remainder of the section we suppose that all weights on G are equal to one:

Definition 3.1. We say that a graph is *Eulerian* if there is a closed trail which visits every edge of the graph exactly once. We call such a trail an *Eulerian trail*.

We now make an important observation:

Lemma 3.2. *If G is Eulerian then every vertex of G has even degree.*

Proof. Suppose that W is a closed trail, crossing every edge of G exactly once. Fix attention on a vertex v. Suppose that W visits v exactly k times. It follows that $\deg(v) = 2k$ and we are done. □

Before proving the converse we require one new idea: *cut and paste* of closed trails. Suppose that W and W' are edge-disjoint closed trails. Suppose also that both W and W' visit the vertex v. Then form a new closed trail W'' by travelling along W until you reach v, travel along all of W', and then finish travelling along W.

Lemma 3.3. *If G is connected and every vertex of G has even degree then G is Eulerian.*

Proof. We begin by finding a closed trail W in G. To do this fix attention on W_k, any trail in G which has length $k > 0$. Suppose that W_k starts at the vertex $v = v_0$ and ends at $w = v_k$. If $w \neq v$ then we find that W_k crosses an odd number of edges incident to w. This is because W_k crosses two edges every time it vists w, except the last time, when W ends at w. As w has even degree then we form W_{k+1} with length one greater than W_k. We continue this process until we find a closed trail W in G starting and ending at $v = v_0$.

Suppose for the moment that W does not cross every edge of G. Examine the graph $G - W$. This has all degrees even. By induction we conclude that every component of $G - W$ is Eulerian. Fix attention on $G' \subset G - W$, one of these components. Let W' be a closed trail in G' which crosses every edge exactly once. Then we form W'' by taking a cut and paste sum of W and W'. Note that W'' is a closed trail in G which is longer than W. We continue this process of adding closed trails to W, via cut and paste, until we have a closed trail cross which every edge of G once. □

It is an easy modification of the proof to show that: *if G is connected and has exactly two vertices, v and w, of odd degree then there is a trail from v to w crossing every edge of G exactly once.* The book calls such a graph *semi-Eulerian*.

We also note that the proof of Lemma 3.3 essentially gives a recipe, or an *algorithm*, for finding an Eulerian trail, if such exists. Here is an explicit version of this algorithm:

Algorithm 3.4. Fix G an Eulerian graph. Fix any vertex v_0 as the start vertex. Let U_0 be the trail of length zero which starts and ends at v_0. We proceed in stages, starting at stage number zero:

- At stage k create a trail W_k by starting at the vertex v_k, travelling randomly through the graph, marking edges as you cross them, and never crossing a previously marked edge.
- After creating W_k let U_{k+1} be the cut and paste sum of W_k and U_k at the vertex v_k.
- If there is a vertex w of U_{k+1}, incident to an unmarked edge, then take $v_{k+1} = w$ and proceed to the next stage.

Note that marked edges remain marked in all later stages.

The proof that the algorithm works is straightforward. We point out only two facts:

- After every stage k the union of the unmarked edges is a graph with all degrees even. (The union of the marked edges forms the closed trail U_{k+1}.)
- After every stage, if there is any unmarked edge, then there is an unmarked edge adjacent to the trail U_{k+1}. (This is because the graph is connected.)

We end this section by noting how Eulerian graphs help to understand the Chinese postperson problem. Fix a graph G. If all degrees of G are even then Algorithm 3.4 will find an Eulerian trail. If not, then there are an even number of vertices of odd degree. Let $O(G) \subset V(G)$ be the set of vertices of odd degree. It is not hard to see that at every vertex of odd degree some incident edge will have to be crossed twice.

So here is an idea: if we could add edges to the graph, one for each pair of elements of $O(G)$, then we would have a new graph where all degrees are even, and the graph would be Eulerian. But we cannot add edges without altering G! Instead, for every pair of vertices $v, w \in O(G)$ find a path $P(v, w)$ from v to w which is as short as possible. This requires a solution to the *shortest path problem*, which will be discussed next week. Next, partition $O(G)$ into a collection of pairs $\{(v_i, w_i)\}$ which minimizes the sum $\sum_i l(P(v_i, w_i))$. This is called choosing a *matching* on $O(G)$ and will be discussed at the end of the semester.

Now attempt to use Algorithm 3.4. We will sometimes get arrive at an odd degree vertex, say v_i, where all edges incident to v_i have been marked *before* finishing the closed trail W_k. In such situations use the chosen short path $P(v_i, w_i)$ to go to w_i, without marking the edges of $P(v_i, w_i)$, and continue the trail W_k.

4 Hamilton

Definition 4.1. We say that a graph is *Hamiltonian* if there is a closed path walk which visits every vertex of the graph exactly once.