Fourteenth Lecture: 4/5

Paths and cycles. We start by introducing a bunch of notation and terminology.

Definitions 14.1. A *path* or *walk* in a graph $G = (V, E)$ is a sequence of vertices $v_1 v_2 \dots v_k$ such that $\{v_i, v_{i+1}\} \in E$ for each $i = 1, \dots, k-1$.

If $v_k = v_1$ then the path is said to be *closed* and is then called a *cycle* or *circuit*.

If either (i) $v_i \neq v_j$ for all $i \neq j$ or (ii) the path is closed but $v_i \neq v_j$ for all $1 \leq i \neq j \leq k-1$, then the path is said to be *simple*. Note that it is generally not sufficient for a path to be simple that it not go over the same edge twice - it must not even go through the same vertex twice, except in the case of the start/endpoint of a closed path.

The *length* of a path is the number of edges in it. Thus, a path $v_1 v_2 \dots v_k$ has length $k-1$.

Definitions 14.2. Let $G = (V, E)$ and $v_1, v_2 \in V$. The *(graph) distance* between v_1 and v_2 is the minimum length of a path between them. One writes $d(v_1, v_2)$. If there is no path between v_1 and v_2, i.e.: if v_1 and v_2 lie in different connected components of G, then one sets $d(v_1, v_2) := +\infty$. It is easy to see that graph distance is a metric, i.e.: it satisfies the following three properties:

(i) $d(v_1, v_2) = 0$ if and only if $v_1 = v_2$.
(ii) Symmetry: $d(v_1, v_2) = d(v_2, v_1)$.
(iii) Triangle inequality: $d(v_1, v_3) \leq d(v_1, v_2) + d(v_2, v_3)$.

The *diameter* of a graph G, denoted $\mathrm{diam}(G)$, is the maximum distance between a pair of vertices, i.e.: $\mathrm{diam}(G) = \max_{v_1, v_2 \in V} d(v_1, v_2)$. Note that $\mathrm{diam}(G) < +\infty$ if and only if G is connected.

One of the oldest problems in the subject of graph theory is the so-called *Bridges of Königsberg problem*, which asks whether it would have been possible, in 17th century Königsberg, to take a walk round the city such that one crossed every bridge exactly once - see Figure 14.1. Consideration of this problem lead to the following notions:

Definition 14.3. Let $G = (V, E)$ be a (multi)graph. A path (resp. cycle) in G is called an *Euler path (resp. cycle)* if it uses every edge exactly once.

The Königsberg problem can be abstracted and reduces to asking whether the multigraph in Figure 14.2 contains an Euler path/cycle ? This graph is small enough that it is possible to check this by brute force - there are 7 edges hence, a priori, $2^7 \cdot 7! = 645,120$ possible ways to order and direct the 7 edges. One finds that no Euler path exists. However, for larger graphs, such a brute force approach will evidently become infeasible. So we are left with a classic Decision and Search Problem: is there a simple rule for determining whether or not a graph posseses an Euler path/cycle and, if it does, is there a simple procedure for constructing one ?

What's particularly nice about Euler paths is that here we have an instance where both the decision and search problems have a simple and elegant solution. Euler himself proved the following theorem:

Theorem 14.4. *Let $G = (V, E)$ be a connected (multi)graph. Then*
(i) G possesses an Euler path between distinct vertices v and w if and only if deg(v) and deg(w) are both odd, and every other vertex has even degree. In particular, G possesess some non-closed Euler path if and only if there are exactly two vertices of odd degree.
(ii) G possesses an Euler cycle if and only if every vertex has even degree.

Note that, in Figure 14.2, the degrees of the vertices are $3, 3, 3, 5$. Hence, no Euler path exists.

Proof. It is easy to see that the conditions imposed on G are *necessary* for existence of Euler paths and cycles. An Euler path never uses the same edge twice. This means that, whenever the path enters a vertex along an edge, it must exit along a different edge. Hence, every visit to a vertex uses up two of the edges incident with that vertex. The only exceptions to this rule are (i) when we leave the starting vertex for the first time (ii) when we enter the destination vertex for the last time. Hence, if these are distinct vertices, they must both have odd degree, while if they are the same vertex (i.e.: if the path is a cycle), then its degree must be even. Every other vertex must have even degree. This proves necessity.

The proof of sufficiency is *constructive*, i.e.: it yields an algorithm for actually finding an Euler path/cycle when one exists, thus solving the search problem. First, let me describe the proof in case (ii), when every vertex has even degree. I will at the end describe what modifications need to be made to the argument in case (i), when there are two vertices of odd degree.

Formally, one can proceed by induction on the number of edges in G. If every vertex has even degree, then G must have at least two edges. If there are two edges, then the only possibility is a graph with two vertices and a pair of edges between them. Clearly this graph has an Euler cycle - just go forward along one edge and back along the other. This takes care of the base case.

Assume that Theorem 14.4(ii) holds for all graphs on at most $n \geq 2$ edges, and let G be a graph on $n + 1$ edges in which every vertex has even degree. Pick any vertex v and, starting from v, perform a "random walk" along the graph, taking care not to use the same edge twice. Because every vertex has even degree, the walk can't get stuck at any other vertex w - each time we visit w along an edge, there must be another edge to pair it off with, along which we can exit again. Hence, the only way we can get stuck is if we arrive back at v and have used up all the edges through v. If this happens, remove from G all the edges along the walk and let G' be the remaining subgraph of G. In G' it will still be the case that every vertex has even degree. Note that G' may not be connected, but even so every connected component of it must satisfy the even-degree condition. Hence, by the induction hypothesis, every component of G' possesses an Euler cycle. Finally, then, one obtains an Euler cycle in G by inserting the Euler cycles of G' as *detours* into the walk we took earlier - more precisely, perform the original walk,

but as soon as one encounters a vertex which is contained in one of the Euler cycles constructed in G', then insert that cycle before continuing with the walk. Note that the inductive nature of the argument means that there could be several levels of embedded detours (i.e.: we may end up making detours of detours of detours etc), but since G has a finite number of edges, we must ultimately get a cycle which uses every edge of G exactly once, v.s.v.

In the case when G possesses exactly two vertices of odd degree, then the only modification required to the above procedure is to make sure that one starts the initial random walk at one of the two odd-degree vertices. Then, by the same argument as before, the only place one can get stuck is at the other odd-degree vertex. If this happens, when the walk is removed from G, the remaining subgraph G' will have all its vertices of even degree. Hence G' possesses an Euler cycle, and this can be inserted as a detour in the original walk, thus yielding an Euler path in G between the two odd-degree vertices. □

The above argument immediately yields an algorithm for finding an Euler path or cycle in a graph which possesses one. It is an example of a so-called *greedy algorithm*, in that it proceeds by adding edges in a headlong forward rush (the random walk), for as long as it can until it is forced to backtrack and insert detours. Here is a quick synopsis of the algorithm. In the first case, we assume every vertex has even degree and in the second that there are exactly two vertices of odd degree.

Euler cycle finding algorithm.

STEP 1: Choose a starting vertex v at random.
STEP 2: Start walking from v. Make sure you don't use the same edge twice. Continue until you return to v and have used up every edge through it.
STEP 3: Remove from G all the edges you've walked along and let G' be the remaining subgraph. For each connected component of G', go to Step 1.
STEP 4: Finally, when there are no edges left, glue together the hierarchy of walks so as to obtain an Euler cycle in the original graph G.

Euler path finding algorithm.

STEP 1: Locate the two vertices of odd degree. Pick one of them at random.
STEP 2: Start walking from the chosen vertex. Make sure you don't use the same edge twice. Continue until you get stuck - this can only happen when you visit the other odd-degree vertex and have used up every edge through it.
STEP 3: Remove from G all the edges you've walked along and let G' be the remaining subgraph. For each connected component of G', perform the Euler cycle finding algorithm.
STEP 4: Glue together the hierarchy of walks so as to obtain an Euler path between the two odd-degree vertices in the original graph G.

Finally for today, we record an extension of Theorem 14.4 to digraphs. This is what we will need to solve the Keycode Problem next time. First, an extension of Definition 13.11 to digraphs:

Definition 14.5. Let $G = (V, E)$ be a digraph and $v \in V$. The *outdegree* of v is the number of edges exiting v, i.e.:

$$\text{outdeg}(v) = \#\{w \in V : (v, w) \in E\}. \tag{14.1}$$

The *indegree* of v is the number of edges entering v, i.e.:

$$\text{indeg}(v) = \#\{w \in V : (w, v) \in E\}. \tag{14.2}$$

Theorem 14.6. (Euler's theorem for digraphs) *Let $G = (V, E)$ be a di(multi)graph. Then*

(i) G possesses an Euler path from a vertex v to a different vertex w if and only if

(a) outdeg(v) = indeg$(v) + 1$,
(b) indeg(w) = outdeg$(w) + 1$,
(c) for all other vertices x, outdeg(x) = indeg(x).

(ii) G possesses an Euler cycle if and only if, for every every vertex x, outdeg(x) = indeg(x).

The theorem is proven in exactly the same way as Theorem 14.4 and yields exactly the same greedy path/cycle-finding algorithm. The reader is left to meditate on this him/herself.