Graph Theory Fall 2019 – EPFL – Lecture Notes.

LECTURER: FRIEDRICH EISENBRAND

October 3, 2019

Acknowledgements: These notes are heavily based on the lecture notes of the Graph Theory courses given by István Tomon, Frank de Zeeuw, Andrey Kupavskii and Dániel Korándi of the DCG group.

Lecture 1

Introduction

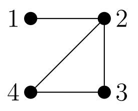
1 Definitions

Definition. *A* graph $G = (V, E)$ *consists of a finite set* V *and a set* E *of two-element subsets of* V*. The elements of* V *are called* vertices *and the elements of* E *are called* edges*.*

For instance, very formally we can introduce a graph like this:

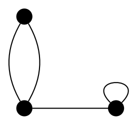
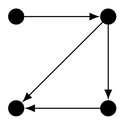
$$V = \{1, 2, 3, 4\}, \quad E = \{\{1, 2\}, \{3, 4\}, \{2, 3\}, \{2, 4\}\}.$$

In practice we more often think of a drawing like this:



Technically, this is what is called a *labelled graph*, but we often omit the labels. When we say something about an unlabelled graph like , we mean that the statement holds for any labelling of the vertices.

Here are two examples of related objects that we do not consider graphs in this course:

The first is a *multigraph*, which can have multiple edges and loops; the corresponding definition would allow the edge set and the edges to be multisets. The second is a *directed graph*, in which every edge has a direction; in the corresponding definition the edges would be ordered pairs instead of two-element subsets.

Although this course is mostly not about these variants, in some cases it will be more natural to state our results for directed or multigraphs. In any case, we will not treat *infinite graphs* in this course.

Graphs (and their above-mentioned variants) are highly applicable in- and outside mathematics because they provide a simple way of modeling many concepts involving connections between objects. For example, graphs can model social networks (vertices=people & edges=friendships), computer networks (computers & links), molecules (atoms & bonds) and many other things. The aim of this course is to study graphs in the abstract sense, and to introduce the fundamental concepts, tools, tricks and results about them.

Some notation: Given a graph G, we write $V(G)$ for the vertex set, and $E(G)$ for the edge set. For an edge $\{x, y\} \in E(G)$, we usually write xy, and we consider yx to be the same edge. If $xy \in E(G)$, then we say that $x, y \in V(G)$ are *adjacent* or *connected* or that they are *neighbors.* If $x \in e$, then we say that $x \in V(G)$ and $e \in E(G)$ are *incident.*

Definition (Subgraphs)**.** *Two graphs* G, G' *are* isomorphic *if there is a bijection* $\varphi : V(G) \to V(G')$ *such that* $xy \in E(G)$ *if and only if* $\varphi(x)\varphi(y) \in E(G')$*. A graph* H *is a* subgraph *of a graph* G*, denoted* $H \subset G$*, if there is a graph* H' *isomorphic to* H *such that* $V(H') \subset V(G)$ *and* $E(H') \subset E(G)$*.*

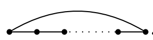
With this definition we can for instance say that is a subgraph of . As mentioned above, when we talk about graphs we often omit the labels of the vertices. A more formal way of doing this is to define an *unlabelled graph* to be an isomorphism class of labelled graphs. We will be somewhat informal about this distinction, since it rarely leads to confusion.

Definition (Degree)**.** *Fix a graph $G = (V, E)$. For $v \in V$, we write*

$$N(v) = \{w \in V : vw \in E\}$$

for the set of neighbors of v (which does not include v). Then $d(v) = |N(v)|$ is the degree *of v. We write $\delta(G)$ for the minimum degree of a vertex in G, and $\Delta(G)$ for the maximum degree.*

Definition (Examples)**.** *The following are some of the most common types of graphs.*

- Paths *are the graphs P_n of the form* *. The graph P_n has $n-1$ edges and n different vertices; we say that P_n has* length *$n-1$.*
- Cycles *are the graphs C_n of the form* *. The graph C_n has n edges and n different vertices; the* length *of C_n is defined to be n.*
- Complete graphs *(or* cliques*) are the graphs K_n on n vertices in which all vertices are adjacent. The graph K_n has $\binom{n}{2}$ edges. For instance, K_4 is* *.*
- *The* complete bipartite graphs *are the graphs $K_{s,t}$ with a partition $V(K_{s,t}) = X \cup Y$ with $|X| = s, |Y| = t$, such that every vertex of X is adjacent to every vertex of Y, and there are no edges inside X or Y. Then K_{st} has st edges. For example, $K_{2,3}$ is* *.*

The following are the most common properties of graphs that we will consider.

Definition (Bipartite)**.** *A graph G is* bipartite *if there is a partition $V(G) = X \cup Y$ such that every edge of G has one vertex in X and one in Y; we call such a partition a* bipartition*.*

Definition (Connected)**.** *A graph G is* connected *if for all $x, y \in V(G)$ there is a path in G from x to y (more formally, there is a path P_k which is a subgraph of G and whose endpoints are x and y).*
A connected component *of G is a maximal connected subgraph of G (i.e., a connected subgraph that is not contained in any larger connected subgraph). The connected components of G form a partition of $V(G)$.*

2 Basic Facts

In this section we prove some basic facts about graphs. It is a somewhat arbitrary collection of statements, but we introduce them here to get used to the terminology and to see some typical proof techniques.

Proposition 1.1. *In any graph G we have $\sum\limits_{v \in V(G)} d(v) = 2|E(G)|$.*

Proof. We count the number of pairs $(v, e) \in V(G) \times E(G)$ such that $v \in e$, in two different ways. On the one hand, a vertex v is involved in $d(v)$ such pairs, so the total number of such pairs is $\sum_{v \in V(G)} d(v)$. On the other hand, every edge is involved in two such pairs, so the number of pairs must equal $2|E(G)|$. $\square$

What we used here is a very powerful proof technique in combinatorics, called *double counting*. The lemma itself is sometimes called the "handshake lemma" because it says that at a party the number of shaken hands is twice the number of handshakes. It has useful corollaries, such as the fact that the number of odd-degree vertices in a graph must be even.

Next, we will describe a very important characterization of bipartite graphs. But first, we need two more definitions.

Definition (Walk). *A* walk *is a sequence $v_1 e_1 v_2 e_2 \dots v_k$ of (not necessarily distinct) vertices v_i and edges e_i such that $e_i = v_i v_{i+1}$. A* closed walk *is a walk with $v_1 = v_k$. The length of this walk is the number of edges, $k-1$.*

It is easy to see that paths are exactly walks with no repeating vertices, and cycles are exactly closed walks with no repeating vertices apart from $v_1 = v_k$.

Definition (Distance). *The* distance *$d(u,v)$ of two vertices $u, v \in V(G)$ is the length of the shortest path (or walk) in G from u to v. (If there is no u-v path in G then $d(u,v) = \infty$.)*

Now we are ready to prove the characterization. Note that "contains a cycle" means that the graph has a subgraph that is isomorphic to some C_n, and similarly for paths. An "odd cycle" is just a cycle whose length is odd.

Theorem 1.2. *A graph is bipartite if and only if it contains no odd cycle.*

Proof. To prove the easy direction of the statement, suppose that G is bipartite with bipartition $V(G) = X \cup Y$, and let $v_1 \cdots v_k v_1$ be a cycle in G with, say, $v_1 \in X$. We must have $v_i \in X$ for all odd i and $v_i \in Y$ for all even i. Since v_k is adjacent to v_1, it must be in Y, so k must be even and the cycle is not odd.

Now for the other direction, suppose G has no odd cycles. We may assume that G is connected. Indeed, otherwise we can apply the same argument to each connected component G_i of G to get a bipartition $X_i \cup Y_i$ of G_i. Choosing $X = \cup_i X_i$ and $Y = \cup_i Y_i$ will then give a bipartition of G.

So if v is a fixed vertex, then every other vertex $u \in V(G)$ has finite distance from v. Let

$$X = \{u : \text{distance of } v \text{ and } u \text{ is } \textit{even}\}$$
$$Y = \{u : \text{distance of } v \text{ and } u \text{ is } \textit{odd}\}.$$

Our aim is to prove that this is a bipartition of G. For this, we need to check that no two vertices in X are adjacent and no two vertices in Y are adjacent.

Suppose for contradiction that some two vertices $u_1, u_2 \in X$ are adjacent, and let e be the edge $u_1 u_2$. By construction, there are paths P_1 from v to u_1 and P_2 from u_2 to v that both have even lengths. But then joining P_1, P_2 and the edge e gives a closed walk $P_1 e P_2$ of odd length, so by Claim 1.3 below, G contains an odd cycle, as well, contradiction the assumption. (Note that $P_1 e P_2$ is not necessarily a cycle because P_1 and P_2 might intersect!)

We can do the same to show that no two vertices $u_1, u_2 \in Y$ are adjacent: here the paths P_1, P_2 will both have odd lengths, so again $P_1 e P_2$ is a closed odd walk. So $X \cup Y$ is indeed a bipartition of G. □

The proof above is a *constructive argument*, where we explicitly constructed the object we were looking for (and then proved that it satisfies the required properties). Of course, we still need to prove the following lemma to complete the proof. We use an *inductive argument*.

Claim 1.3. *Every closed walk of odd length contains an odd cycle.*

Proof. We apply induction on the length k of the walk. Since there is no closed walk of length 1, the statement is vacuously true for $k = 1$. We could use this as the base case, but it is also easy to see that the only closed walk of length 3 is the triangle (K_3), which itself is an odd cycle.

Now suppose the statement is true for every odd length $< k$, and let $W = v_1 e_1 v_2 \dots v_{k+1}$ be a closed walk of odd length k. Let j be the smallest index such that $v_i = v_j$ for some $i < j$. We have two cases. If $j - i$ is even, then deleting the $j - i$ edges $e_i, \dots, e_{j-1}$ from W yields another closed walk $W' \subseteq W$ of odd length. Applying induction on W' then gives an odd cycle in W' and hence in W.

On the other hand, if $j - i$ is odd (and it cannot be 1, so $j - i \geq 3$), then the $j - i$ edges $e_i, \dots, e_{j-1}$ form an odd cycle. Indeed, they form an odd walk without repeated vertices by the choice of v_j. This is what we were looking for. $\square$

3 Second Lecture: Teaser

The next theorem shows that if a graph has many edges, then it must contain a long path.

Theorem 1.4. *Let $k \geq 2$ be an integer and let G be a graph on n vertices with at least $(k-1)n$ edges. Then G contains a path of length k.*

The proof of this theorem is the combination of two lemmas, which are interesting on their own.

First, we consider the special case when every degree of G is large.

Lemma 1.5. *If the minimum degree of G is k, then G contains a path of length k.*

Proof. Let $v_1 \cdots v_l$ be a maximal path in G, i.e., a path that cannot be extended. Then any neighbor of v_1 must be on the path, since otherwise we could extend it. Since v_1 has at least k neighbors, the set $\{v_2, \dots, v_l\}$ must contain at least k elements. Hence $l \geq k + 1$, so the path has length at least k. $\square$

Note that in general this bound cannot be improved, because the complete graph K_{k+1} has minimum degree k, but its longest path has length k. In problem set 1, we will prove an analogous statement for cycles.

Lecture 2

Basic results. Trees.

1 More basic facts

The next theorem shows that if a graph has many edges, then it must contain a long path.

Theorem 2.6. *Let $k \geq 2$ be an integer and let G be a graph on n vertices with at least $(k-1)n$ edges. Then G contains a path of length k.*

Proof. The proof of this theorem is the combination of two lemmas, which are interesting on their own.

First, we consider the special case when every degree of G is large.

Lemma 2.7. *If the minimum degree of G is k, then G contains a path of length k.*

Proof. Let $v_1 \cdots v_l$ be a maximal path in G, i.e., a path that cannot be extended. Then any neighbor of v_1 must be on the path, since otherwise we could extend it. Since v_1 has at least k neighbors, the set $\{v_2, \dots, v_l\}$ must contain at least k elements. Hence $l \geq k+1$, so the path has length at least k. □

Note that in general this bound cannot be improved, because the complete graph K_{k+1} has minimum degree k, but its longest path has length k. In problem set 1, we will prove an analogous statement for cycles.

The next statement shows that every graph with sufficiently many edges must contain a subgraph with large minimum degree. We give an inductive proof, although it could also be proved using an algorithmic argument.

Lemma 2.8. *Let G be graph on n vertices with at least $(k-1)n$ edges. Then G contains a subgraph H with $\delta(H) \geq k$.*

Proof. We proceed by induction on n. Note that $n \leq 2(k-1)$ is impossible because such a graph has at most $n(n-1)/2 < n(k-1)$ edges. Also, for $n = 2k-1$ the only graph with $n(k-1)$ edges is $K_n = K_{2k-1}$, which has minimum degree $2k-2$, so we can take $H = G$.

Now suppose $n > 2k-1$. If $\delta(G) \geq k$, then we can take $H = G$. Otherwise, there is a vertex v of degree $d(v) \leq k-1$. Let $G' = G - v$ be the subgraph we obtain from G by deleting v and all the edges touching it. Then G' has $n-1$ vertices and at least $n(k-1)-(k-1) = (n-1)(k-1)$ edges, so by induction, there is a subgraph $H \subseteq G' \subseteq G$ such that $\delta(H) \geq k$. □

We are done, as G contains a subgraph H with minimum degree at least k by Lemma 2.8, and H contains a path of length k by Lemma 2.7. But then G contains a path of length k as well. □

The following lemma can be helpful when trying to prove certain statements for general graphs that are easier to prove for bipartite graphs. The lemma says that you don't have to remove more than half the edges of a graph to make it bipartite. The proof is an example of an *algorithmic proof*, where we prove the existence of an object by giving an algorithm that constructs such an object.

Proposition 2.9. *Any graph G contains a bipartite subgraph H with $|E(H)| \geq |E(G)|/2$.*

Proof. We prove the stronger claim that G has a bipartite subgraph H with $V(H) = V(G)$ and $d_H(v) \geq d_G(v)/2$ for all $v \in V(G)$. Starting with an arbitrary partition $V(G) = X \cup Y$ (which need not be a bipartition for G), we apply the following procedure. We refer to X and Y as "parts". For any $v \in V(G)$, we see if it has more edges to X or to Y; if it has more edges that connect it to the part it is in than it has edges to the other part, then we move it to the other part. We repeat this until there are no more vertices v that should be moved.

There are at most $|V(G)|$ consecutive steps in which no vertex is moved, since if none of the vertices can be moved, then we are done. When we move a vertex from one part to the other, we increase the number of edges between X and Y (note that a vertex may move back and forth between X and Y, but still the total number of edges between X and Y increases in every step). It follows that this procedure terminates, since there are only finitely many edges in the graph. When it has terminated, every vertex in X has at least half its edges going to Y, and similarly every vertex in Y has at least half its edges going to X. Thus the graph H with $V(H) = V(G)$ and $E(H) = \{xy \in E(G) : x \in X, y \in Y\}$ has the claimed property that $d_H(v) \geq d_G(v)/2$ for all $v \in V(G)$. □

The following is an easy but important fact. Its proof is extremal.

Proposition 2.10. *Every u-v walk W contains a u-v path.*

Proof. Let $v_1v_2\dots v_k$ be a shortest u-v walk in W (more precisely, in the graph defined by the edges of W), so $u = v_1$ and $v = v_k$. We claim that this walk is in fact a path. Indeed, if $v_i = v_j$ for some $i < j$, then $v_1v_2\dots v_iv_{j+1}\dots v_k$ is also a u-v walk, and it is shorter (has fewer edges), which is not possible. So the shortest walk has no repeated vertices, i.e., it is a path. □

This fact has the useful corollaries that we can replace paths with walks in some of our definitions:

- The distance $d(u, v)$ is equal to the length of the shortest u-v walk.
- A graph is connected if and only if every pair of vertices u, v is connected by a walk.

The latter connectivity property is sometimes easier to check. Also, it clearly implies that connectivity is an equivalence relation.

Our last basic result gives a connection between the number of edges and the number of connected components in a graph.

Proposition 2.11. *If a graph G has n vertices and k edges, then it has at least $n - k$ components.*

Proof. Let us start with the empty graph and add the edges of G to it one-by-one. At the beginning there are n vertices and no edges, so we have n components. Each added edge touches at most 2 of the components, and joins these components if they are different (an edge within a component does not affect any components). This means that adding an edge decreases the number of components by at most 1. Adding k edges therefore decreases the number of components by at most k, so after adding all k edges of G, we are left with at least $n - k$ of them. □

Corollary 2.12. *Every connected graph on n vertices has at least $n - 1$ edges.*

2 Trees

Definition. *A* tree *is a connected graph without cycles. A* forest *is a graph without cycles. In a tree or a forest, a vertex of degree one is called a* leaf.

Lemma 2.13. *Every tree with at least two vertices has a leaf.*

Proof. Take a longest path $v_0v_1\dots v_k$ in the tree (so $k \geq 1$, since the tree has at least two vertices). A neighbor of v_0 cannot be outside the path, since then the path could be extended. But if v_0 were adjacent to v_i for some $i > 1$, then $v_0v_1\cdots v_iv_0$ would be a cycle. So the only neighbor of v_0 is v_1, and v_0 is a leaf. The same argument shows that v_k is also a leaf. $\square$

Theorem 2.14. *Any tree T on n vertices has $n-1$ edges.*

Proof. We use induction on the number of vertices. If $n = 1$, then we have 0 edges. Otherwise, Proposition 2.13 gives a leaf v of T. Let $T' = T - v$ be the graph obtained by removing v and its only edge. Then T' is connected, since for any $x, y \in V(T')$ there is a path from x to y in T, and this path cannot pass through v, so it is also a path in T'. Since T has no cycles, neither does T', so T' is a tree, on $n-1$ vertices. By induction T' has $n-2$ edges, so, using $|E(T')| = |E(T)| - 1$, we see that T has $n-1$ edges. $\square$

Theorem 2.15. *A graph G is a tree if and only if for all $u, v \in V(G)$ there is a unique path from u to v.*

Proof. First suppose we have a graph G in which any two vertices are connected by a unique path. Then G is certainly connected. Moreover, if G contained a cycle $v_1\vdots v_kv_1$, then v_1v_k and $v_1v_2\cdots v_k$ would be two distinct paths between v_1 and v_k. Hence G is a tree.

Suppose G is a tree and $u, v \in V(G)$. Since G is connected, there is at least one path from u to v. Suppose there are two distinct paths P, P' from u to v. If these paths only intersect at u and v, we can immediately combine them into a cycle, but in general the paths could intersect in a complicated way, so we have to be careful. The paths P and P' could start out from u being the same; let x be the first vertex that they leave at different edges (so their next vertices are different). Let y be the first vertex of P after x that is also contained in P'. Then there is a cycle in G that goes along P from x to y, and then back along P' from y to x. This is a contradiction, so there is a unique path from u to v in G. $\square$

Lecture 3

BFS. Euler tours. Hamilton cycles.

1 Breadth-first search

Definition. *A* spanning tree *of a graph G is a subgraph $T \subseteq G$, which is a tree with $V(T) = V(G)$.*

Our aim is to show that

Theorem 3.16. *Every connected graph has a spanning tree.*

We start with the *greedy algorithm.* For this, we assume that the edges are indexed by $e_1, \ldots, e_m$.

Algorithm 1. *Unweighted Greedy*

```
Initialize the edge set $F = \varnothing$
for  $i = 1, \ldots, m$
    if $(V, F + e_i)$ does not contain a cycle
        $F = F + e_i$
```

Theorem 3.17. *Let F be constructed by the procedure in Algorithm 2. If G is connected, then (V, F) is a spanning tree.*

Proof. Clearly, (V, F) has no cycles. We only have to show that (V, F) is connected. Let $x = v_1, \ldots, v_l = y$ be a path connecting x and in y in G. And suppose that x and y are not connected in (V, F), then let $k \geq 2$ be minimal such that v_1 and v_k are not connected in (V, F). The edge $v_{k-1}v_k$ has not been inserted by the greedy algorithm. This means that there is a path from v_{k-1} to v_k in (V, F). This means that v_k can be reached from v_1 in (V, F), a contradiction. $\square$

In the *minimum weight spanning tree problem* we are given a connected graph and a weight function $w : E \to \mathbb{R}$. The goal is to find a spanning tree $T = (V, F)$ such that $w(F) \leq w(F')$ for each spanning tree (V, F') of G. This is computed by the weighted greedy algorithm.

Algorithm 2. *Weighted Greedy*

```
Sort the edges in increasing order according to weight:
$w(e_1) \leq w(e_2) \leq \cdots \leq w(e_m)$
Initialize the edge set $F = \varnothing$
for  $i = 1, \ldots, n$
    if $(V, F + e_i)$ does not contain a cycle
        $F = F + e_i$
```

Theorem 3.18. *The weighted greedy algorithm finds a minimum weight spanning tree.*

Proof. Theorem 3.18 shows that T is a tree. Let $|V| = n$ and suppose that the edges of T are $e_{\pi_1}, \ldots, e_{\pi_{n-1}}$ and that they are picked in that order. Let T_{OPT} be a minimum weight spanning tree for which the smallest k such that e_{π_k} is not an edge of T_{OPT} is maximal.

Inserting e_{π_k} into T_{OPT} results in a cycle that does not only contain edges of T. Let e be an edge on this cycle that does not belong to T. This edge must be after e_k in the ordering of the edges according to their weight, since the edges $e_{\pi_1}, \ldots, e_{\pi_{k-1}}$ also belong to T_{OPT} and since any other edge before e_{π_k} creates a cycle in $e_{\pi_1}, \ldots, e_{\pi_{k-1}}$.

This means that $w(e) \leq w(e_{\pi_k})$. Thus $T_{OPT} + e_{\pi_k} - e$ is an optimal spanning tree which contradicts the maximality of k. □

We next describe a spanning tree that is called *breadth-first-search* tree. To this end, let $s \in V$ be a starting vertex and let $V_i = \{v \in V \colon d(s, v) = i\}$ be the vertices that have distance i from s. Let k be the largest distance of a vertex from s that is connected to s. We can build a tree as follows.

```
Initialize the edge set $F = \varnothing$. **for** $i = 1, \ldots, k$
                    **for each** $v \in V_i$:
                        choose $u \in V_{i-1}$ with $uv \in E$.
                        $F = F + uv$
```

Let $V' = \cup_{i=0}^{k} V_i$. The graph $T_{BFS} = (V', F)$ is connected, since one can reach s from any vertex by following their edge "downwards". Also $|F| = |V'| - 1$ which implies that T_{BFS} is a tree. It is a *breadth-first-search tree.* The unique path from s to any other vertex in T_{BFS} is a shortest path from s to that vertex in G. It is computed by the following algorithm.

Algorithm 3 (Breadth-First-Search)**.**

```
*Initialize:* $F = \varnothing$, *queue* $Q = \{s\}$
*Label each vertex* $v \in V \setminus \{s\}$ unknown
*Label* $s$ known
**while** $Q$ *is not empty*
    $u = head(Q)$
    $dequeue(u)$
    **for each** $v \in N(u)$
        **if** $v$ *is unknown*
            *label* $v$ *as known*
            $F = F + uv$
            *enqueue* $v$
```

The algorithm computes a bfs-tree in a bottom up manner. The following are claims that are easily verified by induction.

Lemma 3.19. *For each* $i = 0, \ldots, k$, *there is a moment in time, when* Q *contains exactly* V_i, *all vertices in* $\cup_{j=0}^{i} V_j$ *are labeled as known and all other vertices are labeled as unknown. Furthermore* $(\cup_{j=0}^{i} V_j, F)$ *is a tree in which the unique path from* s *to any other vertex is a shortest path in* G.

Definition. *The* diameter *of a graph* G *is the largest distance among any pairs of vertices:* $diam(G) = \max_{x,y \in V(G)} d(x, y)$. *If* G *is disconnected, then* $diam(G) = \infty$.

BFS has a number of applications in providing fairly (or very) efficient algorithms for solving certain tasks on graphs. For example:

- *Find a shortest path from xu to v in G:* Run the algorithm with root u to get a tree T. The unique path from u to v in T (which exists by Lemma 2.15) is a shortest path.
- *Find the connected components of G:* Run the algorithm with some root r. The vertices explored by BFS are exactly the component of r. If there is an unexplored vertex r', run BFS again from r' as a root. Repeat until all vertices are visited. This actually gives a spanning forest of G.
- *Compute diam(G):* For each pair of vertices, we can find a shortest path and thus the distance. Do this for all pairs and take the largest distance.
- *Find a shortest cycle in G:* For every edge uv, find a shortest path between u and v in $G - uv$ (if it exists), then combine this path with uv to get a cycle. (This will be a shortest cycle through uv.) Compare all these cycles to find the shortest.
- *Determine if G is bipartite:* Determine the connected components of G. In every component H, select a root r, and partition the vertices into $X = \{x \in V(H) : d(r,x) \text{ is even}\}$ and $Y = \{y \in V(H) : d(r,y) \text{ is odd}\}$. Then H is bipartite if and only if X and Y have no internal edges (see the proof of Theorem 1.2), and G is bipartite if and only if every component is bipartite.

Not all of these algorithms are the most efficient, but they are already much better than brute force approaches that go over all possible answers. There are all kinds of algorithms that do these tasks faster, but in this course, we don't care too much about efficiency, and we focus on the graph-theoretical aspects (in particular, proving that the algorithms work).

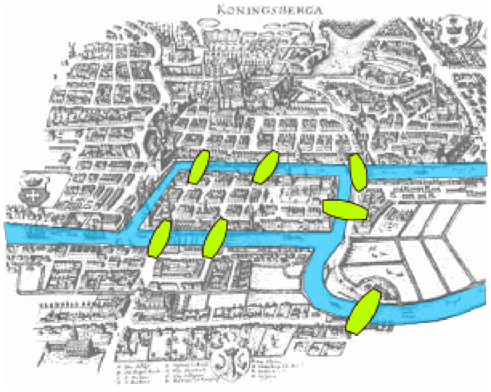
We should emphasize, though, that for finding shortest paths between two vertices, BFS is the best general algorithm. For example, *Dijkstra's algorithm*, a variant of BFS that works for graphs with positive edge weights (representing, e.g, the lengths of streets) is directly used by routing softwares.

2 Euler tours

Definition. *A* trail *is a walk with no repeated edges. A* tour *is a closed trail (i.e., one that starts and ends at the same vertex).*

Definition. *An* Euler (or Eulerian) trail *in a (multi)graph $G = (V, E)$ is a trail in G passing through every edge (exactly once). An* Euler tour *is a tour in G passing through every edge.*

This notion originates from the "seven bridges of Königsberg" problem – the oldest problem in graph theory, originally solved by Euler in 1736 – that asked if it was possible to walk through all the seven bridges of Königsberg in one go without crossing any of them twice.



This question can be turned into a graph problem asking for an Euler trail. Euler solved the problem by noticing that the existence of Euler trails is closely related to the degree parities.

Theorem 3.20. *A connected (multi)graph has an Eulerian tour if and only if each vertex has even degree.*

The proof of this theorem is based on the following simple lemma.

Lemma 3.21. *In a graph where all vertices have even degree, every maximal trail is a closed trail.*

Proof. Let T be a maximal trail. If T is not closed, then T has an odd number of edges incident to the final vertex v. However, as v has even degree, there is an edge touching v that is not contained in T. This edge can be used to extend T to a longer trail, contradicting the maximality of T. □

Proof of Theorem 3.20. To see that the condition is necessary, suppose G has an Eulerian tour C. If a vertex v was visited k times in the tour C, then each visit used 2 edges incident to v (one incoming edge and one outgoing edge). Thus, $d(v) = 2k$, which is even.

To see that the condition is sufficient, let G be a connected graph with even degrees. Let $T = e_1e_2 \dots e_\ell$ (where $e_i = (v_{i-1}, v_i)$) be a longest trail in G. Then it is maximal, of course. According to the Lemma, T is closed, i.e., $v_0 = v_\ell$. G is connected, so if T does not include all the edges of G then there is an edge e outside of T that touches it, i.e., $e = uv_i$ for some vertex v_i in T. Since T is closed, we can start walking through it at any vertex. But if we start at v_i then we can append the edge e at the end: $T' = e_{i+1} \dots e_\ell e_1 e_2 \dots e_i e$ is a trail in G which is longer than T, contradicting the fact that T is a longest trail in G. Thus, T must include all the edges of G and so it is an Eulerian tour. □

Corollary 3.22. *A connected multigraph G has an Euler trail if and only if it has either* 0 *or* 2 *vertices of odd degree.*

Proof. Suppose T is an Euler trail from vertex u to vertex v. If $u = v$ then T is an Eulerian tour and so by Theorem 3.20, it follows that all the vertices in G have even degree. If $u \neq v$ then let us add a new edge $e = uv$ to G. In this new multigraph $G \cup \{e\}$, $T \cup \{e\}$ is an Euler tour. By Theorem 3.20 we see that all the degrees in $G \cup \{e\}$ are even. This means that in the original multigraph G, the vertices u, v are the only ones that have odd degree.

Now we prove the other direction of the corollary. If G has no vertices of odd degree then by Theorem 3.20 it contains an Eulerian tour which is also an Eulerian trail. Suppose now that G has 2 vertices u, v of odd degree. Then add a new edge e to G. Now all vertices of the resulting multigraph $G \cup \{e\}$ have even degree, so, by Theorem 3.20, it has an Eulerian tour C. Removing the edge e from C gives an Eulerian trail of G from u to v. □