GRAPH THEORY

Graph Coloring

Definition 22.1 Let $G = (V, E)$ be a graph and $k$ be a positive integer. A $k$-coloring of $G$ is a function $f : V \rightarrow [1..k]$. A coloring is called proper when for any edge $u-v$ in $E$ we have $f(u) \not= f(v)$. A graph that admits a proper $k$-coloring is called $k$-colorable. The smallest $k$ such that $G$ is $k$-colorable is called the chromatic number of $G$.

It is easy to see that a graph is 1-colorable iff it is edgeless. It is also easy to see that if two graphs are isomorphic then their colorings correspond one-to-one.

An important application of graph coloring is map coloring. The famous Four Color Theorem states that any planar graph is 4-colorable. We will only be studying planar graphs a bit in this course.

Example 22.2 Let $\Delta(G)$ be the maximum degree of a node in $G$. Show that $G$ is $\Delta(G)+1$-colorable and, moreover, the chromatic number of $K_n$ is $n$ hence there exist graphs $G$ whose chromatic number is $\Delta(G) + 1$.

Solution: By induction on the number of vertices. Omitted. (The proof also suggests an algorithm.)

For $K_n$ note that nodes must have different colors. ■

Bipartite Graphs

2-colorable graphs are also called bipartite.

Warning: graphs can be 2-colored in more than one way, consider $(\{1,2,3,4\}, \{1,2,3\})$. I will commonly refer to the two colors in bipartite graphs as red and blue (rather than 1 and 2).

Example 22.3 Let $n \geq 3$. Prove that a cycle $C_n = \{[1..n],[1-2,\ldots,(n-1)-n,n-1]\}$ is bipartite iff $n$ is even.

Solution: W.l.o.g., suppose we color 1 red. Then every even vertex has to be blue and every odd vertex has to be red. Because of the edge $n-1$, this coloring is proper iff $n$ is blue. ■
**Proposition 22.4** *Every tree is bipartite.*

**Proof:** Single vertex trees are edgeless hence 1-colorable hence 2-colorable. For edgy trees we prove this it by induction on the number of vertices.

**Base case:** $n = 2$. A one edge tree. Color one of the two vertices red and the other blue.

**Induction step:** Let $k \geq 2$. Assume (IH) that any tree with $k$ vertices is bipartite.

Let $G$ have $k + 1$ vertices. By Lemma 20.12, it has a leaf, $u$. Delete $u$ and the one edge adjacent to $u$ to obtain $G_u$. By Lemma 20.13 $G_u$ is also a tree. Since it has $k$ vertices it is bipartite, hence it has a proper red/blue coloring. Let $v$ be the vertex adjacent to $u$ in $G$. W.l.o.g. assume $v$ is red in this coloring. Now we put back $u$ and its adjacent edge and we color $u$ blue to extend the coloring to all of $G$.

When we say “let $G = (V, E)$ be a bipartite graph” we will assume that we are also given a partition $V = R \cup B$, $R \cap B = \emptyset$. As before, I will usually refer to the nodes in $R$ as “red” and to those in $B$ as “blue”.

**Definition 22.5** *The complete bipartite graph* $K_{m,n}$ *has* $m$ *red nodes, n blue nodes and an edge between every red node and every blue node.*

**Example 22.6** Show that $K_{m,n} \simeq K_{n,m}$.

**Solution:** The isomorphism maps red to blue and blue to red.

**Proposition 22.7** *A graph is bipartite iff it does not contain an odd cycle.*

**Proof:** First the easy direction. If a graph is bipartite and it contains a cycle, than that cycle is also 2-colorable hence by Example 22.3 it must have an even number of vertices and edges.

In the other direction, assume that a graph $G = (V, E)$ contains no odd cycle and we want to show that it is bipartite.

Since no edges go between connected components, a graph has a proper coloring iff each of its connected components has a proper coloring. Therefore, w.l.o.g. we can assume that $G$ is connected.

Now we need to take a detour and define the concept of **distance between vertices**.

**Definition 22.8** *Let $G = (V, E)$ be a connected graph. The distance between two vertices* $u, v \in V$, notation $d(u, v)$, *is defined as the length of a shortest path from* $u$ *to* $v$. *We know that some path*

\[1\] We say “a” instead of “the” because there may be more than one path whose length is the smallest.
exists because the graph is connected. The smallest length therefore exists, by the WOP. Hence a shortest path exists.

When \( u = v \) we have \( d(u, u) = 0 \) given by the path of length 0. Moreover, \( d(u, v) = d(v, u) \) since the reversal of a path has the same length.

**Lemma 22.9 (The Triangle Inequality)**

\[
d(u, v) \leq d(u, w) + d(w, v)
\]

**Proof:** (of Lemma)
The cases when two of the three (or all three) vertices coincide is immediate. So assume \( u, v, w \) distinct. By the definition, there is a path of length \( d(u, w) \) from \( u \) to \( w \) and a path of length \( d(w, v) \) from \( w \) to \( v \). Concatenating these two paths gives us a walk (we don’t know if it is a path because the two paths may have intermediate vertices in common) of length \( d(u, w) + d(w, v) \) from \( u \) to \( v \). But a shortest path from \( u \) to \( v \) has length \( \leq \) than the length of any walk from \( u \) to \( v \). (Recall the proof “where there is a walk there is a path”.)

Another useful observation is the following.

**Lemma 22.10 (Locality of shortest paths)** Consider a shortest path \( p \) from \( u \) to \( v \) and let \( x \) and \( y \) be two vertices in this path. Then the portion \( x \cdots y \) of \( p \) is a shortest path from \( x \) to \( y \), so \( d(x, y) \leq d(u, v) \).

**Proof:** (of Lemma)
Indeed, if there exists a strictly shorter path from \( x \) to \( y \) then we can replace the portion \( x \cdots y \) of \( p \) with that shorter path and get a strictly shorter path from \( u \) to \( v \).

Back to the proof of the proposition. We have a connected graph \( G = (V, E) \) without odd cycles and we want to show that it is bipartite.

Fix and arbitrary vertex \( w_0 \). Now color the vertices of \( G \) as follows: \( w \) is colored red when \( d(w_0, w) \) is even (in particular, \( w_0 \) is colored red) and is colored blue when \( d(w_0, w) \) is odd. Since \( d(w_0, w) \) is either even or odd, but not both, every vertex is colored, and is colored with a single color, so we have defined a 2-coloring.

Now we claim that this 2-coloring is proper. Once we show this, the proof is done.

Suppose, toward a contradiction, that we have an edge \( u-v \in E \) such that both \( u \) and \( v \) are colored blue. (Later we consider the case when they are both colored red.) Our hope is to show that this implies that \( G \) has an odd cycle, hence we have a contradiction.

\(^2\)Some textbooks define distance for arbitrary graphs by allowing \( \infty \) to be the distance between vertices that are not connected.
Note that $w_0, u, v$ are pairwise distinct. Let $p_1$ be a shortest path $w_0\ldots u$ and $p_2$ be a shortest path $w_0\ldots v$. Since $u, v$ are colored blue the lengths of $p_1$ and $p_2$ are odd.

It is is true that from $p_1$, $u-v$ and (reversing) $p_2$ we get a closed walk $w_0\ldots u-v\ldots w_0$ of length $d(w_0, u) + 1 + d(w_0, v)$, which is odd. But this walk is, in general, not a cycle because $p_1$ and $p_2$ may have intermediate vertices in common and having a closed walk of odd length does not, by itself, imply that we have an odd cycle.

Let $S$ be the set of vertices that $p_1$ and $p_2$ have in common. Suppose $u \in S$. Then, by the lemma above (locality of shortest paths) we have $d(w_0, v) = d(w_0, u) + 1$, but both $d(w_0, v)$ and $d(w_0, u)$ are odd, contradiction. Hence $u \notin S$. Similarly $v \notin S$.

$S$ is not empty because $w_0 \in S$. Let $w \in S$ be “closest” to $u$, i.e., all the nodes in the portion $w\ldots u$ of $p_1$ are not in $S$. It follows that all the nodes in the portion $w\ldots v$ of $p_2$ are also not in $S$. Thus the closed walk $w\ldots u-v\ldots w$ forms a cycle, $C$. We shall prove that $C$ has odd length.

By the lemma above (locality of shortest paths) the portion $w_0\ldots w$ of $p_1$ has length $d(w_0, w)$. Similarly, the portion $w_0\ldots w$ of $p_2$ has length $d(w_0, w)$. Hence the length of $C$ is $d(w_0, u) - d(w_0, w) + 1 + d(w_0, v) - d(w_0, w) = d(w_0, u) + 1 + d(w_0, v) - 2d(w_0, w)$ which is odd.

Suppose now that we have an edge $u-v \in E$ such that $u$ and $v$ are both colored red. Then $u \neq w_0$, otherwise $d(w_0, v) = 1$ and $v$ should be blue. Similarly $v \neq w_0$.

Now that we have established that $w_0, u, v$ are pairwise distinct the proof proceeds as in the earlier case, again resulting in a cycle of length $d(w_0, u) + 1 + d(w_0, v) - 2d(w_0, w)$ which is again odd, even though now $d(w_0, u)$ and $d(w_0, v)$ are both even. 

\section*{Spanning Trees}

Definition 22.11 A \textit{spanning subgraph} of the graph $G = (V, E)$ is a subgraph whose vertex set in $V$. A \textit{spanning tree (forest)} of $G$ is a spanning subgraph that is a tree (a forest).

Proposition 22.12 Every connected graph has a spanning tree. (Hence every graph has a spanning forest.)

Proof: Consider all the connected spanning subgraphs of $G$. There is at least one such subgraph, $G$ itself. By the Well-Ordering Principle at least one of the connected spanning subgraphs must have the smallest number of edges. We prove that each edge $e$ of this spanning subgraph is a cut edge. Indeed, if not, then we can delete $e$ obtaining still a connected spanning subgraph but with strictly fewer edges. It now follows from Proposition 21.7 (see statement (iii)) that we have a spanning tree.

The proof above suggests an algorithm for constructing a spanning tree but it seems to involve computing all connected spanning subgraphs and counting their edges. Here are two alternative algorithms for constructing a spanning tree that are much better.
**Edge-Pruning Algorithm**

*Input:* A connected graph $G = (V, \{e_1, \ldots, e_m\})$ where $|V| \geq 2$.

*Output:* A spanning tree $T$ of $G$.

1. Let $T = G$.
2. For $k = 1, \ldots, m$ do:
   2a) if $e_k$ is not a cut edge in $T$
   2b) then delete $e_k$ from $T$
   2c) else leave $e_k$ in $T$
3. Output $T$

**Proposition 22.13** The Edge-Pruning Algorithm terminates and when it does it constructs a spanning tree. *(This provides an alternative proof for Proposition 22.12)*

**Proof:** The body of the for-loop in the algorithm executes $m$ times after which the algorithm terminates.

In what follows we denote by $T$ the graph that keeps changing during the execution of the algorithm and we denote by $T_o$ the final form of $T$, i.e., the graph that is outputted by the algorithm.

We prove that $T_o$ is spanning subgraph which is connected and acyclic (hence a spanning tree).

$T$ starts with all the nodes and it never drops a node so it is a spanning subgraph throughout the algorithm. In particular, $T_o$ is a spanning subgraph.

Next we show that $T_o$ is a connected graph. We do this by proving that $T$ is always a connected graph, by induction on the number $p$ of times the body of the “for-loop”, (i.e., steps (2a), (2b), and (2c)) is executed. (BTW, a predicate that is true before the body of a loop is executed and stays true afterwards is called a “loop invariant”.)

**Base case:** loop body executed $p = 0$ times. That is before the loop is executed and at that point $T = G$ and $G$ is given as connected.

**Induction step:** Let $p \in \mathbb{N}$ arbitrary. Assume (IH) that after the body of the loop is executed $p$ times $T$ is connected.

Now we execute the body of the loop one more time. Suppose, toward a contradiction, that after the $(p+1)^{st}$ execution of the loop, $T$ is not connected. By IH we know that $T$ was connected before the $(p+1)^{st}$ execution of the loop. Thus it can only happen because of of step (2b) during the the $(p+1)^{st}$ execution of the loop, i.e., a cut edge was deleted. But this is specifically forbidden by the condition (2a).

Hence $T_o$ is also connected.

**Acyclicity:** Now we wish to argue that $T_o$ is acyclic, i.e., $T$ will be acyclic at the termination of the algorithm. Suppose, toward a contradiction, that there is a cycle $C$ in $T_o$. Consider $e$, the last edge to be examined (processed) by the algorithm among the edges of $C$. Since $e$ was the last edge of $C$ examined, the other edges of $C$ must have already been examined and kept in $T$ otherwise they
would not be in \( T \) (recall that the algorithm does not add edges and examines each edge exactly once). Thus, the entire cycle \( C \) must have existed in \( T \) at the time when \( e \) was examined. Hence \( e \) was not a cut edge in \( T \) at that time (we proved in lecture that edges in a cycle cannot be cut edges) and should have been deleted. This contradicts the fact that \( e \) is still in \( T_o \). 

\[ \square \]

**Edge-Growing Algorithm**

*Input:* A connected graph \( G = (V, \{e_1, \ldots, e_m\}) \) where \( |V| \geq 2 \).

*Output:* A spanning tree \( T \) of \( G \).

1. Let \( T = (V, \emptyset) \) (the edgeless graph on \( V \)).
2. For \( k = 1, \ldots, m \) do:
   1. if adding \( e_k \) to \( T \) does *not* form a cycle with edges already in \( T \)
   2. then add \( e_k \) to \( T \)
   3. else leave \( e_k \) out of \( T \)
3. Output \( T \)

**Proposition 22.14** The Edge-Growing Algorithm terminates and when it does it constructs a spanning tree. (This provides yet another proof for Proposition 22.12.)

**Proof:** The algorithm terminates after \( m \) executions of the body of the for-loop.

Again we denote by \( T \) the graph that keeps changing during the execution of the algorithm and we denote by \( T_o \) the final form of \( T \), i.e., the graph that is outputted by the algorithm.

We prove that \( T_o \) is spanning subgraph which is is connected and acyclic (hence a spanning tree). \( T \) starts with all the nodes and it never drops a node so it is a spanning subgraph throughout the algorithm. In particular, \( T_o \) is a spanning subgraph.

Next we show that \( T_o \) is an acyclic graph. We do this by proving that \( T \) is always an acyclic graph, by induction on the number \( p \) of times the body of the “for-loop”, (i.e., steps (2a), (2b), and (2c)) is executed. (BTW, a predicate that is true before the body of a loop is executed and stays true afterwards is called a “loop invariant”.)

**Base case:** loop body executed \( p = 0 \) times. That is before the loop is executed and at that point \( T \) is an edgeless graph, clearly acyclic.

**Induction step:** Let \( p \in \mathbb{N} \) arbitrary. Assume (IH) that after the body of the loop is executed \( p \) times \( T \) is acyclic.

Now we execute the body of the loop one more time. Suppose, toward a contradiction, that after the \( (p + 1) \)st execution of the loop, \( T \) contains a cycle. This cycle was either (i) created by the addition of an edge in (2b) during the the \( (p + 1) \)st execution of the loop, or (ii) it was already in \( T \) before.

But (i) is specifically forbidden by the condition (2a) while (ii) is impossible due to the IH.
Hence $T_o$ is also acyclic.

Finally we show that $T_o$ is a connected graph. We are given that $G$ is connected, so for any two arbitrary vertices $u$ and $v$ in $V$, there exists a path $p$ between them in $G$. If all the edges in $p$ are in $T_o$ we are done.

Now consider the case when there may exist some edges along $p$ which are not added to $T$ when they are processed by the algorithm in step (2). Consider one such edge $e = v_a - v_b$. It is left out of $T$ because it would complete a cycle $C$ with edges already in $T$. Let $q \equiv v_a - v_1 - v_2 - \cdots - v_n - v_b$ be the path obtained by deleting $e$ from $C$. The edges of $q$ were in in $T$ when $e$ was processed. Since no edges are deleted from $T$ after they are added these edges remain in $T_o$. Replace $e$ in $p$ with the path $q$.

We do the same with each of the other edges in $p$ which are not added to $T$ when they are processed by the algorithm in step (2). The result is a walk between $u$ and $v$ whose edges are all in $T_o$. It follows that $T_o$ is connected.

We have shown that $T_o$ is a spanning subgraph of $G$ that is acyclic and connected so it’s a spanning tree.

What is remarkable about both algorithms is that each edge needs to be considered only once, and the order in which edges are considered does not matter. There is still the matter of testing whether edges are cut edges or whether they form cycles. It turns out that there are even better ways of finding spanning trees, using depth-first or breadth-first traversals of the graph, as you will learn in CIS 121.

**Eulerian Graphs**

**Definition 22.15** Let $G$ be a graph without isolated vertices. An Eulerian trail in $G$ is a walk that traverses each edge of $G$ exactly once. An Eulerian tour in $G$ is an Eulerian trail that is also a closed walk.

Notice that we used “trail” instead of “path” and “tour” instead of “cycle”. Indeed, an Eulerian tour need not be a cycle as it may still have repeated vertices. Similarly, an Eulerian trail, even with distinct endpoints, is not necessarily a path. Clearly, an Eulerian trail contains all the vertices hence a graph with an Eulerian trail (in particular, with an Eulerian tour) is connected. For connected graphs the absence of isolated vertices is equivalent to the existence of at least one edge (i.e., the graph is edgy). Thus we will focus on edgy connected graphs.

Next we give necessary and sufficient conditions for the existence of Eulerian trails/tours in edgy connected graphs.

---

3This will avoid some trivial cases.
Proposition 22.16 An edgy connected graph has an Eulerian trail iff exactly two of its vertices have odd degree (there others, if any, have even degree). An edgy connected graph has an Eulerian tour iff all its vertices have even degree.

Proof: Suppose an edgy graph has an Eulerian trail: $u\cdots v$. Every intermediate occurrence of a vertex in this walk adds 2 to its degree. If $v$ is the same as $u$ (i.e., this is an Eulerian tour) then the end occurrences add 2 to the degree of $u$. Overall, every vertex has even degree. If $u \neq v$ then $u$ and $v$ are the only vertices of odd degree.

Next we prove the converse, by induction on the number of edges. Specifically we prove the following:

For any $m \geq 1$, for any connected graph $G$ with $m$ edges ($m \geq 1$ hence $G$ is edgy), if $G$ has exactly two vertices of odd degree, $a$ and $b$, then $G$ has an Eulerian trail from $a$ to $b$ and if all of $G$’s vertices have even degree then $G$ has an Eulerian tour.

Base case: $m = 1$. Then $G = (\{a, b\}, \{a-b\})$, $a$ and $b$ have both degree 1 and we have a one-edge Eulerian trail from $a$ to $b$.

Induction step: Let $k \geq 1$ arbitrary. Assume (IH) that for any connected graph $G$ with $k$ edges, if $G$ has exactly two vertices of odd degree, $a$ and $b$, then $G$ has an Eulerian trail from $a$ to $b$ and if all of $G$’s vertices have even degree then $G$ has an Eulerian tour.

Now let $G = (V, E)$ be any connected graph with $k + 1$ edges.

Case 1: all vertices of $G$ have even degree.

Lemma A graph $G$ in which all vertices have even degree has no cut edges.

Proof of lemma Suppose, toward a contradiction, that $u-v$ is a cut edge in $G$. Delete $u-v$ from $G$. $u$ and $v$ are in the same connected component in $G$ but after deleting $u-v$ they must end up in different connected components, otherwise the number of connected components remains the same. Let $D$ be the connected component that contains $u$ after deletion. Looking at $D$ as a graph, $u$ is the only vertex on odd degree in $D$. This contradicts what we proved from the Handshaking Lemma: the number of vertices of odd degree is even. (Beautiful, eh?)

Back to the proof of the proposition. Delete an edge (any edge, it does not matter) $e \equiv u-v$ from $G$ to obtain a graph $G_e$. By the lemma $e$ cannot be a cut edge so $G_e$ is still connected. Notice that $G_e$ has exactly two vertices of odd degree, $u$ and $v$. Since $G_e$ has $k$ we can apply the IH and obtain an Eulerian trail $u\cdots v$ is $G_e$. Together with $e$ this forms an Eulerian tour in $G$.

Case 2: exactly two of the vertices of $G$ have odd degree.

The proof here proceeds along similar lines but it is a bit longer and we omit it. (It’s a good exercise!)