11.9 Connectivity

Definition 11.9.1. *Two vertices are connected* in a graph when there is a path that begins at one and ends at the other. By convention, every vertex is connected to itself by a path of length zero. A *graph is connected* when every pair of vertices are connected.

11.9.1 Connected Components

Being connected is usually a good property for a graph to have. For example, it could mean that it is possible to get from any node to any other node, or that it is possible to communicate between any pair of nodes, depending on the application.

But not all graphs are connected. For example, the graph where nodes represent cities and edges represent highways might be connected for North American cities, but would surely not be connected if you also included cities in Australia. The same is true for communication networks like the internet—in order to be protected from viruses that spread on the internet, some government networks are completely isolated from the internet.

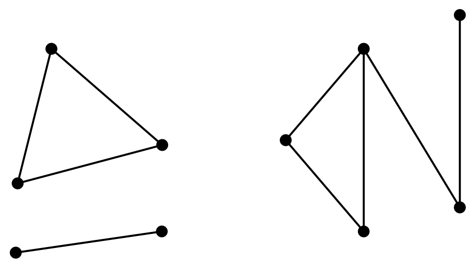

Figure 11.16 One graph with 3 connected components.

Another example is shown in Figure 11.16, which looks like a picture of three graphs, but is intended to be a picture of *one* graph. This graph consists of three pieces (subgraphs). Each piece by itself is connected, but there are no paths between vertices in different pieces. These connected pieces of a graph are called its *connected components*.

Definition 11.9.2. A *connected component* of a graph is a subgraph consisting of some vertex and every node and edge that is connected to that vertex.

So, a graph is connected iff it has exactly one connected component. At the other extreme, the empty graph on n vertices has n connected components.

11.9.2 Odd Cycles and 2-Colorability

We have already seen that determining the chromatic number of a graph is a challenging problem. There is one special case where this problem is very easy, namely, when the graph is 2-colorable.

Theorem 11.9.3. *The following graph properties are equivalent:*

1. *The graph contains an odd length cycle.*
2. *The graph is not 2-colorable.*
3. *The graph contains an odd length closed walk.*

In other words, if a graph has any one of the three properties above, then it has all of the properties.

We will show the following implications among these properties:

1. IMPLIES 2. IMPLIES 3. IMPLIES 1.

So each of these properties implies the other two, which means they all are equivalent.

1 IMPLIES 2 *Proof.* This follows from equation 11.3. ■

2 IMPLIES 3 If we prove this implication for connected graphs, then it will hold for an arbitrary graph because it will hold for each connected component. So we can assume that G is connected.

Proof. Pick an arbitrary vertex r of G. Since G is connected, for every node $u \in V(G)$, there will be a walk $\mathbf{w}_u$ starting at u and ending at r. Assign colors to vertices of G as follows:

$$\text{color}(u) = \begin{cases} \text{black}, & \text{if } |\mathbf{w}_u| \text{ is even}, \\ \text{white}, & \text{otherwise}. \end{cases}$$

Now since G is not colorable, this can't be a valid coloring. So there must be an edge between two nodes u and v with the same color. But in that case

$$\mathbf{w}_u \widehat{\ } \text{reverse}(\mathbf{w}_v) \widehat{\ } \langle v\text{—}u \rangle$$

is a closed walk starting and ending at u, and its length is

$$|\mathbf{w}_u| + |\mathbf{w}_v| + 1$$

which is odd. ■

3 IMPLIES 1 *Proof.* Since there is an odd length closed walk, the WOP implies there is an odd length closed walk $\mathbf{w}$ of minimum length. We claim $\mathbf{w}$ must be a cycle. To show this, assume to the contrary that $\mathbf{w}$ is not a cycle, so there is a repeat vertex occurrence besides the start and end. There are then two cases to consider depending on whether the additional repeat is different from, or the same as, the start vertex.

In the first case, the start vertex has an extra occurrence. That is,

$$\mathbf{w} = \mathbf{f} \,\widehat{x}\, \mathbf{r}$$

for some positive length walks $\mathbf{f}$ and $\mathbf{r}$ that begin and end at x. Since

$$|\mathbf{w}| = |\mathbf{f}| + |\mathbf{r}|$$

is odd, exactly one of $\mathbf{f}$ and $\mathbf{r}$ must have odd length, and that one will be an odd length closed walk shorter than $\mathbf{w}$, a contradiction.

In the second case,

$$\mathbf{w} = \mathbf{f}\,\widehat{y}\,\mathbf{g}\,\widehat{y}\,\mathbf{r}$$

where $\mathbf{f}$ is a walk from x to y for some $y \neq x$, and $\mathbf{r}$ is a walk from y to x, and $|\mathbf{g}| > 0$. Now $\mathbf{g}$ cannot have odd length or it would be an odd-length closed walk shorter than $\mathbf{w}$. So $\mathbf{g}$ has even length. That implies that $\mathbf{f}\,\widehat{y}\,\mathbf{r}$ must be an odd-length closed walk shorter than $\mathbf{w}$, again a contradiction.

This completes the proof of Theorem 11.9.3. ■

Theorem 11.9.3 turns out to be useful, since bipartite graphs come up fairly often in practice. We'll see examples when we talk about planar graphs in Chapter 12.

11.9.3 k-connected Graphs

If we think of a graph as modeling cables in a telephone network, or oil pipelines, or electrical power lines, then we not only want connectivity, but we want connectivity that survives component failure. So more generally, we want to define how strongly two vertices are connected. One measure of connection strength is how many links must fail before connectedness fails. In particular, two vertices are k-*edge connected* when it takes at least k "edge-failures" to disconnect them. More precisely:

Definition 11.9.4. Two vertices in a graph are k-*edge connected* when they remain connected in every subgraph obtained by deleting up to $k - 1$ edges. A graph is k-edge connected when it has more than one vertex, and pair of distinct vertices in the graph are k- connected.

Notice that according to Definition 11.9.4, if a graph is k-connected, it is also j-connected for $j \leq k$. This convenient convention implies that two vertices are connected according to definition 11.9.1 iff they are 1-edge connected according to Definition 11.9.4. From now on we'll drop the "edge" modifier and just say "k-connected."[9]

[9]There is a corresponding definition of k-vertex connectedness based on deleting vertices rather than edges. Graph theory texts usually use "k-connected" as shorthand for "k-vertex connected." But edge-connectedness will be enough for us.

For example, in the graph in figure 11.15, vertices c and e are 3-connected, b and e are 2-connected, g and e are 1 connected, and no vertices are 4-connected. The graph as a whole is only 1-connected. A complete graph, k_n, is $(n-1)$-connected. Every cycle is 2-connected.

The idea of a *cut edge* is a useful way to explain 2-connectivity.

Definition 11.9.5. If two vertices are connected in a graph G, but not connected when an edge e is removed, then e is called a *cut edge* of G.

So a graph with more than one vertex is 2-connected iff it is connected and has no cut edges. The following Lemma is another immediate consequence of the definition:

Lemma 11.9.6. *An edge is a cut edge iff it is not on a cycle.*

More generally, if two vertices are connected by k edge-disjoint paths—that is, no edge occurs in two paths—then they must be k-connected, since at least one edge will have to be removed from each of the paths before they could disconnect. A fundamental fact, whose ingenious proof we omit, is Menger's theorem which confirms that the converse is also true: if two vertices are k-connected, then there are k edge-disjoint paths connecting them. It takes some ingenuity to prove this just for the case $k = 2$.

11.9.4 The Minimum Number of Edges in a Connected Graph

The following theorem says that a graph with few edges must have many connected components.

Theorem 11.9.7. *Every graph, G, has at least $|V(G)| - |E(G)|$ connected components.*

Of course for Theorem 11.9.7 to be of any use, there must be fewer edges than vertices.

Proof. We use induction on the number, k, of edges. Let $P(k)$ be the proposition that

> every graph, G, with k edges has at least $|V(G)| - k$ connected components.

Base case ($k = 0$): In a graph with 0 edges, each vertex is itself a connected component, and so there are exactly $|V(G)| = |V(G)| - 0$ connected components. So $P(0)$ holds.

Inductive step:

Let G_e be the graph that results from removing an edge, $e \in E(G)$. So G_e has k edges, and by the induction hypothesis $P(k)$, we may assume that G_e has at least $|V(G)| - k$-connected components. Now add back the edge e to obtain the original graph G. If the endpoints of e were in the same connected component of G_e, then G has the same sets of connected vertices as G_e, so G has at least $|V(G)| - k > |V(G)| - (k + 1)$ components. Alternatively, if the endpoints of e were in different connected components of G_e, then these two components are merged into one component in G, while all other components remain unchanged, so that G has one fewer connected component than G_e. That is, G has at least $(|V(G)| - k) - 1 = |V(G)| - (k + 1)$ connected components. So in either case, G has at least $|V(G)| - (k + 1)$ components, as claimed.

This completes the inductive step and hence the entire proof by induction. ■

Corollary 11.9.8. *Every connected graph with n vertices has at least* $n - 1$ *edges.*

A couple of points about the proof of Theorem 11.9.7 are worth noticing. First, we used induction on the number of edges in the graph. This is very common in proofs involving graphs, as is induction on the number of vertices. When you're presented with a graph problem, these two approaches should be among the first you consider.

The second point is more subtle. Notice that in the inductive step, we took an arbitrary $(k+1)$-edge graph, threw out an edge so that we could apply the induction assumption, and then put the edge back. You'll see this shrink-down, grow-back process very often in the inductive steps of proofs related to graphs. This might seem like needless effort: why not start with an k-edge graph and add one more to get an $(k + 1)$-edge graph? That would work fine in this case, but opens the door to a nasty logical error called *buildup error*, illustrated in Problem 11.48.

11.10 Forests & Trees

We've already made good use of digraphs without cycles, but *simple* graphs without cycles are arguably the most important graphs in computer science.

11.10.1 Leaves, Parents & Children

Definition 11.10.1. An acyclic graph is called a *forest*. A connected acyclic graph is called a *tree*.

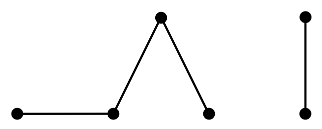

Figure 11.17 A 6-node forest consisting of 2 component trees.

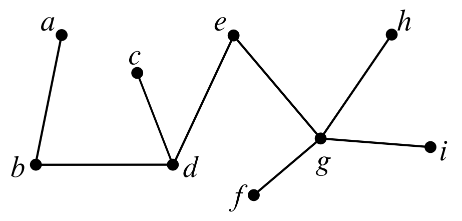

Figure 11.18 A 9-node tree with 5 leaves.

The graph shown in Figure 11.17 is a forest. Each of its connected components is by definition a tree.

One of the first things you will notice about trees is that they tend to have a lot of nodes with degree one. Such nodes are called *leaves*.

Definition 11.10.2. A degree 1 node in a forest is called a *leaf*.

The forest in Figure 11.17 has 4 leaves. The tree in Figure 11.18 has 5 leaves.

Trees are a fundamental data structure in computer science. For example, information is often stored in tree-like data structures, and the execution of many recursive programs can be modeled as the traversal of a tree. In such cases, it is often useful to arrange the nodes in levels, where the node at the top level is identified as the *root* and where every edge joins a *parent* to a *child* one level below. Figure 11.19 shows the tree of Figure 11.18 redrawn in this way. Node d is a child of node e and the parent of nodes b and c.

11.10.2 Properties

Trees have many unique properties. We have listed some of them in the following theorem.

Theorem 11.10.3. *Every tree has the following properties:*

1. *Every connected subgraph is a tree.*

2. *There is a unique path between every pair of vertices.*

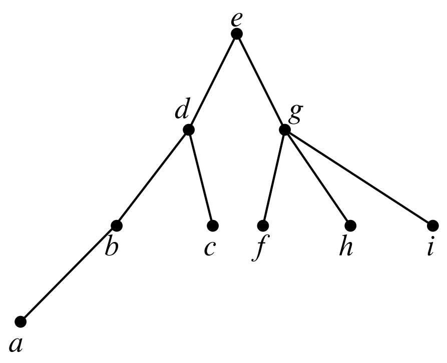

Figure 11.19 The tree from Figure 11.18 redrawn with node e as the root and the other nodes arranged in levels.

3. *Adding an edge between nonadjacent nodes in a tree creates a graph with a cycle.*
4. *Removing any edge disconnects the graph. That is, every edge is a cut edge.*
5. *If the tree has at least two vertices, then it has at least two leaves.*
6. *The number of vertices in a tree is one larger than the number of edges.*

Proof. 1. A cycle in a subgraph is also a cycle in the whole graph, so any subgraph of an acyclic graph must also be acyclic. If the subgraph is also connected, then by definition, it is a tree.

2. Since a tree is connected, there is at least one path between every pair of vertices. Suppose for the purposes of contradiction, that there are two different paths between some pair of vertices. Then there are two distinct paths $\mathbf{p} \neq \mathbf{q}$ between the same two vertices with minimum total length $|\mathbf{p}| + |\mathbf{q}|$. If these paths shared a vertex, w, other than at the start and end of the paths, then the parts of $\mathbf{p}$ and $\mathbf{q}$ from start to w, or the parts of $\mathbf{p}$ and $\mathbf{q}$ from w to the end, must be distinct paths between the same vertices with total length less than $|\mathbf{p}| + |\mathbf{q}|$, contradicting the minimality of this sum. Therefore, $\mathbf{p}$ and $\mathbf{q}$ have no vertices in common besides their endpoints, and so $\mathbf{p}\,\widehat{\ }\,\text{reverse}(\mathbf{q})$ is a cycle.

3. An additional edge $\langle u\text{—}v \rangle$ together with the unique path between u and v forms a cycle.

4. Suppose that we remove edge $\langle u\text{—}v\rangle$. Since the tree contained a unique path between u and v, that path must have been $\langle u\text{—}v\rangle$. Therefore, when that edge is removed, no path remains, and so the graph is not connected.

5. Since the tree has at least two vertices, the longest path in the tree will have different endpoints u and v. We claim u is a leaf. This follows because, since by definition of endpoint, u is incident to at most one edge on the path. Also, if u was incident to an edge not on the path, then the path could be lengthened by adding that edge, contradicting the fact that the path was as long as possible. It follows that u is incident only to a single edge, that is u is a leaf. The same hold for v.

6. We use induction on the proposition

$$P(n) ::= \text{there are } n-1 \text{ edges in any } n\text{-vertex tree.}$$

Base case ($n = 1$): $P(1)$ is true since a tree with 1 node has 0 edges and $1 - 1 = 0$.

Inductive step: Now suppose that $P(n)$ is true and consider an $(n+1)$-vertex tree, T. Let v be a leaf of the tree. You can verify that deleting a vertex of degree 1 (and its incident edge) from any connected graph leaves a connected subgraph. So by Theorem 11.10.3.1, deleting v and its incident edge gives a smaller tree, and this smaller tree has $n - 1$ edges by induction. If we re-attach the vertex, v, and its incident edge, we find that T has $n = (n+1) - 1$ edges. Hence, $P(n + 1)$ is true, and the induction proof is complete. ■

Various subsets of properties in Theorem 11.10.3 provide alternative characterizations of trees. For example,

Lemma 11.10.4. *A graph G is a tree iff G is a forest and $|V(G)| = |E(G)| + 1$.*

The proof is an easy consequence of Theorem 11.9.7.6 (Problem 11.55).

11.10.3 Spanning Trees

Trees are everywhere. In fact, every connected graph contains a subgraph that is a tree with the same vertices as the graph. This is called a *spanning tree* for the graph. For example, Figure 11.20 is a connected graph with a spanning tree highlighted.

Definition 11.10.5. Define a *spanning subgraph* of a graph, G, to be a subgraph containing all the vertices of G.

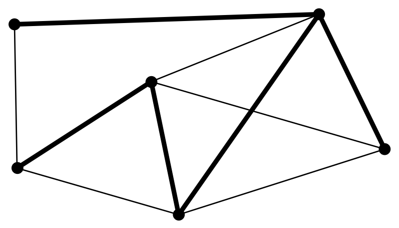

Figure 11.20 A graph where the edges of a spanning tree have been thickened.

Theorem 11.10.6. *Every connected graph contains a spanning tree.*

Proof. Suppose G is a connected graph, so the graph G itself is a connected, spanning subgraph. So by WOP, G must have a minimum-edge connected, spanning subgraph, T. We claim T is a spanning tree. Since T is a connected, spanning subgraph by definition, all we have to show is that T is acyclic.

But suppose to the contrary that T contained a cycle C. By Lemma 11.9.6, an edge e of C will not be a cut edge, so removing it would leave a connected, spanning subgraph that was smaller than T, contradicting the minimality to T. ■

11.10.4 Minimum Weight Spanning Trees

Spanning trees are interesting because they connect all the nodes of a graph using the smallest possible number of edges. For example the spanning tree for the 6-node graph shown in Figure 11.20 has 5 edges.

In many applications, there are numerical costs or weights associated with the edges of the graph. For example, suppose the nodes of a graph represent buildings and edges represent connections between them. The cost of a connection may vary a lot from one pair of buildings or towns to another. Another example is where the nodes represent cities and the weight of an edge is the distance between them: the weight of the Los Angeles/New York City edge is much higher than the weight of the NYC/Boston edge. The *weight of a graph* is simply defined to be the sum of the weights of its edges. For example, the weight of the spanning tree shown in Figure 11.21 is 19.

Definition 11.10.7. A *minimum weight spanning tree* (MST) of an edge-weighted graph G is a spanning tree of G with the smallest possible sum of edge weights.

Is the spanning tree shown in Figure 11.21(a) an MST of the weighted graph shown in Figure 11.21(b)? It actually isn't, since the tree shown in Figure 11.22 is also a spanning tree of the graph shown in Figure 11.21(b), and this spanning tree has weight 17.

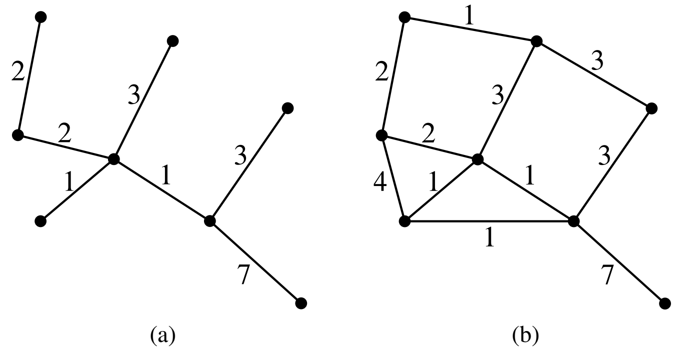

Figure 11.21 A spanning tree (a) with weight 19 for a graph (b).

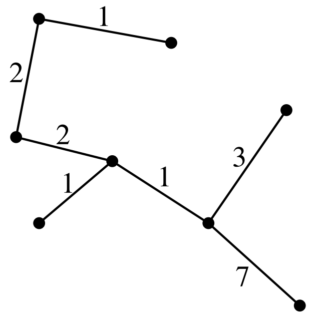

Figure 11.22 An MST with weight 17 for the graph in Figure 11.21(b).

What about the tree shown in Figure 11.22? It seems to be an MST, but how do we prove it? In general, how do we find an MST for a connected graph G? We could try enumerating all subtrees of G, but that approach would be hopeless for large graphs.

There actually are many good ways to find MST's based on a property of some subgraphs of G called *pre-MST*'s.

Definition 11.10.8. A *pre-MST* for a graph G is a spanning subgraph of G that is also a subgraph of some MST of G.

So a pre-MST will necessarily be a forest.

For example, the empty graph with the same vertices as G is guaranteed to be a pre-MST of G, and so is any actual MST of G.

If e is an edge of G and S is a spanning subgraph, we'll write $S + e$ for the spanning subgraph with edges $E(S) \cup \{e\}$.

Definition 11.10.9. If F is a pre-MST and e is a new edge, that is $e \in E(G) - E(F)$, then e *extends* F when $F + e$ is also a pre-MST.

So being a pre-MST is contrived to be an invariant under addition of extending edges, by the definition of extension.

The standard methods for finding MST's all start with the empty spanning forest and build up to an MST by adding one extending edge after another. Since the empty spanning forest is a pre-MST, and being a pre-MST is, by definition, invariant under extensions, every forest built in this way will be a pre-MST. But no spanning tree can be a subgraph of a different spanning tree. So when the pre-MST finally grows enough to become a tree, it will be an MST. By Lemma 11.10.4, this happens after exactly $|V(G)| - 1$ edge extensions.

So the problem of finding MST's reduces to the question of how to tell if an edge is an extending edge. Here's how:

Definition 11.10.10. Let F be a pre-MST, and color the vertices in each connected component of F either all black or all white. At least one component of each color is required. Call this a *solid coloring* of F. A *gray edge* of a solid coloring is an edge of G with different colored endpoints.

Any path in G from a white vertex to a black vertex obviously must include a gray edge, so for any solid coloring, there is guaranteed to be at least one gray edge. In fact, there will have to be at least as many gray edges as there are components with the same color. Here's the punchline:

Lemma 11.10.11. *An edge extends a pre-MST F if it is a minimum weight gray edge in some solid coloring of F.*

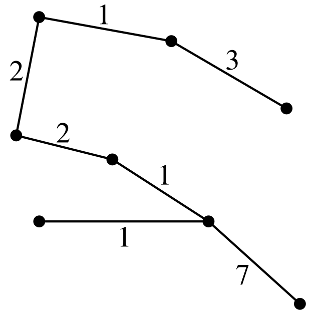

Figure 11.23 A spanning tree found by Algorithm 1.

So to extend a pre-MST, choose any solid coloring, find the gray edges, and among them choose one with minimum weight. Each of these steps is easy to do, so it is easy to keep extending and arrive at an MST. For example, here are three known algorithms that are explained by Lemma 11.10.11:

Algorithm 1. *[Prim] Grow a tree one edge at a time by adding a minimum weight edge among the edges that have exactly one endpoint in the tree.*

This is the algorithm that comes from coloring the growing tree white and all the vertices not in the tree black. Then the gray edges are the ones with exactly one endpoint in the tree.

Algorithm 2. *[Kruskal] Grow a forest one edge at a time by adding a minimum weight edge among the edges with endpoints in different connected components.*

An edge does not create a cycle iff it connects different components. The edge chosen by Kruskal's algorithm will be the minimum weight gray edge when the components it connects are assigned different colors.

For example, in the weighted graph we have been considering, we might run Algorithm 1 as follows. Start by choosing one of the weight 1 edges, since this is the smallest weight in the graph. Suppose we chose the weight 1 edge on the bottom of the triangle of weight 1 edges in our graph. This edge is incident to the same vertex as two weight 1 edges, a weight 4 edge, a weight 7 edge, and a weight 3 edge. We would then choose the incident edge of minimum weight. In this case, one of the two weight 1 edges. At this point, we cannot choose the third weight 1 edge: it won't be gray because its endpoints are both in the tree, and so are both colored white. But we can continue by choosing a weight 2 edge. We might end up with the spanning tree shown in Figure 11.23, which has weight 17, the smallest we've seen so far.

Now suppose we instead ran Algorithm 2 on our graph. We might again choose the weight 1 edge on the bottom of the triangle of weight 1 edges in our graph. Now, instead of choosing one of the weight 1 edges it touches, we might choose the weight 1 edge on the top of the graph. This edge still has minimum weight, and will be gray if we simply color its endpoints differently, so Algorithm 2 can choose it. We would then choose one of the remaining weight 1 edges. Note that neither causes us to form a cycle. Continuing the algorithm, we could end up with the same spanning tree in Figure 11.23, though this will depend on the tie breaking rules used to choose among gray edges with the same minimum weight. For example, if the weight of every edge in G is one, then all spanning trees are MST's with weight $|V(G)| - 1$, and both of these algorithms can arrive at each of these spanning trees by suitable tie-breaking.

The coloring that explains Algorithm 1 also justifies a more flexible algorithm which has Algorithm 1 as a special case:

Algorithm 3. *Grow a forest one edge at a time by picking any component and adding a minimum weight edge among the edges leaving that component.*

This algorithm allows components that are not too close to grow in parallel and independently, which is great for “distributed” computation where separate processors share the work with limited communication between processors.

These are examples of greedy approaches to optimization. Sometimes greediness works and sometimes it doesn't. The good news is that it does work to find the MST. Therefore, we can be sure that the MST for our example graph has weight 17, since it was produced by Algorithm 2. Furthermore we have a fast algorithm for finding a minimum weight spanning tree for any graph.

Ok, to wrap up this story, all that's left is the proof that minimal gray edges are extending edges. This might sound like a chore, but it just uses the same reasoning we used to be sure there would be a gray edge when you need it.

Proof. (of Lemma 11.10.11)

Let F be a pre-MST that is a subgraph of some MST M of G, and suppose e is a minimum weight gray edge under some solid coloring of F. We want to show that $F + e$ is also a pre-MST.

If e happens to be an edge of M, then $F + e$ remains a subgraph of M, and so is a pre-MST.

The other case is when e is not an edge of M. In that case, $M + e$ will be a connected, spanning subgraph. Also M has a path $\mathbf{p}$ between the different colored endpoints of e, so $M + e$ has a cycle consisting of e together with $\mathbf{p}$. Now $\mathbf{p}$ has both a black endpoint and a white one, so it must contain some gray edge $g \neq e$. The trick is to remove g from $M + e$ to obtain a subgraph $M + e - g$. Since gray edges by definition are not edges of F, the graph $M + e - g$ contains $F + e$. We claim that $M + e - g$ is an MST, which proves the claim that e extends F.

To prove this claim, note that $M + e$ is a connected, spanning subgraph, and g is on a cycle of $M + e$, so by Lemma 11.9.6, removing g won't disconnect anything. Therefore, $M + e - g$ is still a connected, spanning subgraph. Moreover, $M + e - g$ has the same number of edges as M, so Lemma 11.10.4 implies that it must be a spanning tree. Finally, since e is minimum weight among gray edges,

$$w(M + e - g) = w(M) + w(e) - w(g) \leq w(M).$$

This means that $M + e - g$ is a spanning tree whose weight is at most that of an MST, which implies that $M + e - g$ is also an MST. ■

Another interesting fact falls out of the proof of Lemma 11.10.11:

Corollary 11.10.12. *If all edges in a weighted graph have distinct weights, then the graph has a* unique *MST.*

The proof of Corollary 11.10.12 is left to Problem 11.70.