Solutions to question set for Chapter 3

1. Let T be a spanning tree in a graph G and $e \in E(G)$ an edge not in T. Prove that $T \cup \{e\}$ contains a unique cycle (called a **fundamental cycle** of G). Prove that deleting any edge from a fundamental cycle yields a spanning tree of G.

   SOLUTION: Since $T \cup \{e\}$ will have more than $|V(G)| - 1$ edges we know it cannot be a tree. Since T is connected, so must $T \cup \{e\}$ be (we cannot destroy any walks by adding edges). So $T \cup \{e\}$ fails to be a tree because it has a cycle, say, C. We must prove that C is unique.

   Suppose there is another cycle C' distinct from C in $T \cup \{e\}$. Then C' has some edge $f = uv$, say, not in C. There is a path in C', not passing e, from u to a vertex u' of C; similarly, there is a path in C' from v, not passing e, to a vertex v' of C. We know that u' and v' exist since e must lie in both C and C' (otherwise both cycles would exist in T, which is impossible), and the endpoints of e can be reached by paths in C'.

   Let $P_{u',u}$, $P_{v,v'}$ and $P_{v',u'}$ be the paths from u' to u and v to v' in C', and the path from v' to u' in C, respectively. Then $f; P_{v,v'}; P_{v',u'}; P_{u',u}$ is a cycle not passing e. But this cycle must lie in T which is impossible. So C' cannot exist and C is unique.

   Consider the spanning subgraph $H = T \cup \{e\}$ and let $f = uv$ be any edge in a the unique fundamental cycle. Since deleting f destroys the fundamental cycle and cannot create new cycles, it is sufficient to prove that deleting f cannot disconnect H. Let x and y be any two vertices of H and let W be a walk connecting them. If W does not pass f then it still exists in $H \backslash \{f\}$. Suppose W does pass f; let u be first vertex of f encountered in W. Then in $H \backslash \{f\}$ there is still a walk from x to y since we can follow W from x to u, following the fundamental cycle from u to v in the direction which avoids f, and then follow W again from v to y. Since this is true for any pair x and y of vertices $H \backslash \{f\}$ is connected and we are done.

   Notice the two arguments about walks: in the first proof we make a specific formal construction of a path; in the second proof we are much less formal: "following the fundamental cycle from u to v in the direction which avoids f" almost amounts to 'hand waving'! Both types of arguments occur frequently in graph theory texts. As long as everyone is convinced, that is the main thing.

2. Let G be a connected graph with weighted edges and let H be a connected subgraph of G. Consider the following Assertion:

   Assertion: Let T be a minimum-weight spanning tree for H and let $e = xy$ be an edge of minimum weight leaving H, i.e. with $x \in V(H)$ and $y \notin V(H)$. Let H^+ be the subgraph of G obtained by adding to H the vertex y and all edges of G joining H to y. Then $T + e$ is a minimum weight spanning tree for H^+.

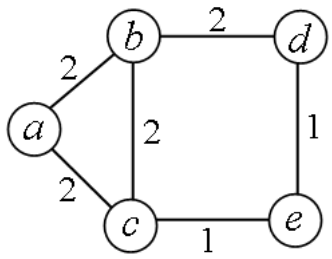
   (a) Would this assertion, if true, give a proof of correctness of the MWST algorithm?

   (b) Is the assertion true? If so, prove it; if not produce a counterexample (a graph G and subgraph H for which we can show the assertion is false).

   SOLUTION:

   (a) Yes! Because at each iteration of MWST we have a tree. Let H be the subgraph of G consisting of all vertices of T and all edges of G with these vertices as endpoints. The assertion says that T remains a minimum-weight spanning tree when we extend this subgraph by a including a new vertex. Eventually we will include a final vertex and H will be the whole of G.

(b) Unfortunately not! (Which is why Chapter 3, Section 1 has the more sophisticated proof based on extendability). A counterexample is shown on the right: if H is the subgraph on vertices a, b, c and d then a minimum-weight subtree of H must have three edges of weight 2. But when we add the vertex e we create a graph in which a minimum-weight subtree has two edges of weight 1 and two of weight 2. This is obviously not an extension of a minimum-weight subtree of H.



3. You are given a connected graph G. Consider the following three tasks:
   (a) T1: find a vertex in G of maximum degree;
   (b) T2: decide if G is simple (i.e. has no self-loops or multiple edges);
   (c) T3: decide if G has a bridge (an edge whose deletion causes a connected graph to become disconnected).

   Give a possible value for the complexity of an algorithm for each of these tasks, using big-Oh notation and giving a brief justification for your answers.

   Are your answers different if you are told that every vertex in G has degree 2 (in which case G is said to be **2-regular**))?

   SOLUTION:

   T1: to find a vertex in G of maximum degree we can look at each vertex and for each one find its degree. As seen in Week 1 Lecture 3 this gives complexity $|V||E|$. This is not necessarily the quickest possible algorithm (there may be some clever trick like the Handshaking Lemma which we have missed) but it answers the question.

   T2: to decide if G is simple we can check each edge $e \in E(G)$. If e is subset of size 1 then it is a self-loop. So it takes $|E|$ steps to check if there are self-loops. To see if e is a multiple edge we can check whether the same subset is repeated 'further down' the list of edges. So this will take $|E| - 1$ steps for the first edge, $|E| - 2$ steps for the second edge, etc. Each check will take 2 steps to compare the two elements of e. So this is a total of

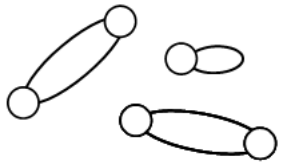
   $$2 \times (|E| - 1 + |E| - 2 + |E| - 3 + \ldots + 2 + 1) = 2 \times \frac{1}{2}|E|(|E| - 1).$$

   So overall time for checking self-loops and multiple edges is $|E| + |E|(|E| - 1) = |E|^2 = O(|E|^2)$.

   T3: decide if G has a bridge we can look at each edge e and check if $G - e$ is connected. We saw in Week 2 Lecture 2 that connectedness may be checked using the Maximal_Subtree algorithm which has complexity $O(|V||E|)$. Running this algorithm for each edge gives a total time of $O(|V||E|^2)$.

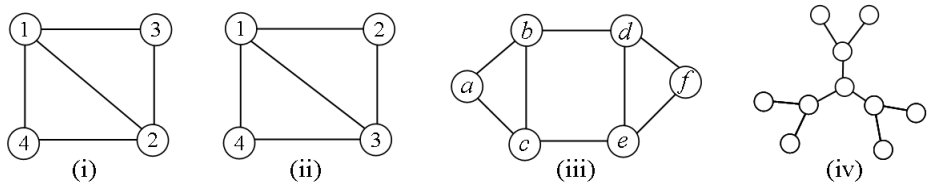
   Now we are told that G is 2-regular. Then T1 becomes trivial: we must just check in 1 step to see that V is nonempty. If so we return 2, otherwise we return 0 (or 'trivial graph', say). So this takes $O(1)$ steps.

   T2: we must still check every edge; again $|E|$ steps will find any self-loops. We observe that a multiple edge can only be a double edge, and this must be disconnected from the remainder of the graph (as must a self-loop, see the illustration on the right), since this gives degree 2 to both its vertices. However, it is not obvious that this saves us from checking for repeats all the way through E. So complexity remains $O(|E|^2)$. Perhaps you can think of something better?



   T3: we can dramatically improve things thanks to the following **Lemma:** a 2-regular graph consists of a collection of disjoint cycles. (Try proving this as an extra exercise!). So no bridge can exist and, as for T1, we have an $O(1)$ algorithm.

4. A graph $G = (V, E)$ is said to be **prime** if we can label the vertices with the integers $1, \ldots, |V|$ in such a way that every edge has coprime endpoints (greatest common divisor 1). For example, the graph (i) below has been given a non-prime labelling since an edge joins 2 and 4 which are not coprime ing.



(a) Prove that the graph at (iii) is not prime.

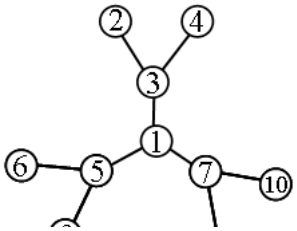
(b) Show that the tree at (iv) is prime.

(A 30-year old conjecture of Entringer and Tout asserts that every tree is prime. In 2010, Penny Haxell, Oleg Pikhurko and Anusch Taraz presented a proof that this is true for all sufficiently large trees. By 'sufficiently large' they suggest a figure of $10^{10^{100}}$ vertices!)

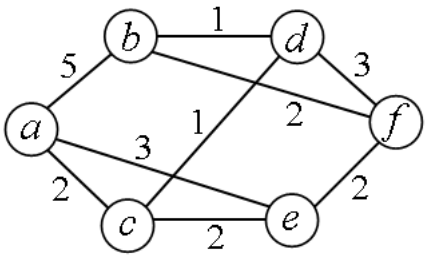
SOLUTION:

(a) The vertices must be labelled, $1, \ldots, 6$. Now 6 is not coprime to 2, 3 or 4 so 6 must label a vertex v which has three non-adjacent vertices. In a 6-vertices graph such a vertex v must have degree 2. So $v = a$ or $v = f$. But if $v = a$ then 2, 3 and 4 must label d, e and f which is impossible since 2 cannot be adjacent to 4. Similarly $v = f$ cannot be labelled 6. So no prime labelling is possible.

(b) A prime labelling is given as shown on the right. This is not unique—many such labellings are possible

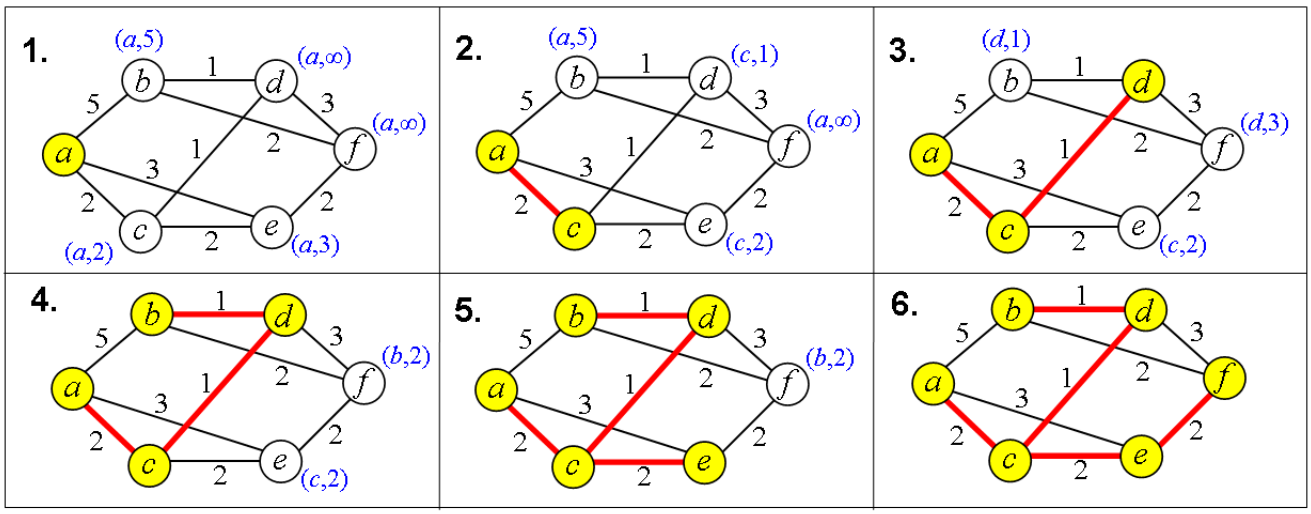


5. Run Prim's Algorithm on the graph shown on the right, starting at vertex a and showing the iterations of the algorithm
(i) diagrammatically; (ii) in a table.
Your output trees need not be identical in each case but should be clearly specified, together with their total edge weights.

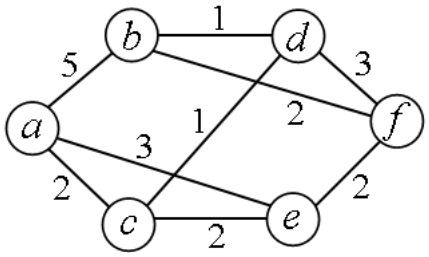


SOLUTION:

(a) A diagrammatic solution is shown below. To save time it combines the updating and extending steps of each iteration into one diagram. This solution is not unique since we have a choice at step whether to extend the tree by adding ce (as shown) or bf. Notice that no updating takes place at step 5: we do **not** replace the label $(b, 2)$ on vertex f with $(e, 2)$ because we only update labels when we find an improvement on the current label.



(b) The table solution is shown below, together with the original graph, for ease of reference.



	Vertices:	a	b	c	d	e	f	u	$V(T)$	$E(T)$
	Initialise:		(a, ∞)	(a, ∞)	(a, ∞)	(a, ∞)	(a, ∞)	a	$\{a\}$	$\emptyset$
Iteration										
1	Update labels:		$(a, 5)$	$(a, 2)$	(a, ∞)	$(a, 3)$	(a, ∞)			
	Extend tree:			✔				c	$\{a, c\}$	$\{ac\}$
2	Update labels:		$(a, 5)$		$(c, 1)$	$(c, 2)$	(a, ∞)			
	Extend tree:				✔			d	$\{a, c, d\}$	$\{ac, cd\}$
3	Update labels:		$(d, 1)$			$(c, 2)$	$(d, 3)$			
	Extend tree:		✔					b	$\{a, c, d, b\}$	$\{ac, cd, bd\}$
4	Update labels:					$(c, 2)$	$(b, 2)$			
	Extend tree:					✔		e	$\{a, c, d, b, e\}$	$\{ac, cd, bd, ce\}$
5	Update labels:						$(b, 2)$			
	Extend tree:						✔	f	$\{a, c, d, b, e, f\}$	$\{ac, cd, bd, ce, bf\}$