Module MA2C03: Discrete Mathematics
Hilary Term 2016
Sections 32 to 35: Spanning Trees

D. R. Wilkins

Contents

Preliminaries

An *undirected graph* can be thought of as consisting of a finite set V of points, referred to as the *vertices* of the graph, together with a finite set E of *edges*, where each edge joins two distinct vertices of the graph.

We now proceed to formulate the definition of an undirected graph in somewhat more formal language.

Let V be a set. We denote by V_2 the set consisting of all subsets of V with exactly two elements. Thus, for any set V,

$$V_2 = \{A \in \mathcal{P}V : \#(A) = 2\},$$

where $\mathcal{P}V$ denotes the power set of V (i.e., the set consisting of all subsets of V), and $\#(A)$ denotes the number of elements in a subset A of V.

Definition An *undirected graph* (V, E) consists of a finite set V together with a subset E of V_2 (where V_2 is the set consisting of all subsets of V with exactly two elements). The elements of V are the *vertices* of the graph; the elements of E are the *edges* of the graph.

Definition A graph is said to be *trivial* if it consists of a single vertex.

We may denote a graph by a single letter such as G. Writing $G = (V, E)$ indicates that V is the set of vertices and E is the set of edges of some graph G.

Definition If v is a vertex of some graph, if e is an edge of the graph, and if $e = v\,v'$ for some vertex v' of the graph, then the vertex v is said to be *incident* to the edge e, and the edge e is said to be *incident* to the vertex v.

Definition Two distinct vertices v and v' of a graph (V, E) are said to be *adjacent* if and only if $v\,v' \in E$.

Definition Let (V, E) and (V', E') be graphs. The graph (V', E') is said to be a *subgraph* of (V, E) if and only if $V' \subset V$ and $E' \subset E$ (i.e., if and only if the vertices and edges of (V', E') are all vertices and edges of (V, E)).

Definition Let (V, E) be a graph. The *degree* $\deg v$ of a vertex v of this graph is defined to be the number of edges of the graph that are incident to v (i.e., the number of edges of the graph which have v as one of their endpoints).

Definition A vertex of a graph of degree 0 is said to be an *isolated* vertex.

Definition A vertex of a graph of degree 1 is said to be an *pendent* vertex.

Definition Let (V, E) be a graph. A *walk* $v_0\, v_1\, v_2\, \ldots\, v_n$ of length n in the graph from a vertex a to a vertex b is determined by a finite sequence $v_0, v_1, v_2, \ldots, v_n$ of vertices of the graph such that $v_0 = a$, $v_n = b$ and $v_{i-1}\, v_i$ is an edge of the graph for $i = 1, 2, \ldots, n$.

A walk $v_0\, v_1\, v_2\, \ldots\, v_n$ in a graph is said to *traverse* the edges $v_{i-1}v_i$ for $i = 1, 2, \ldots, n$ and to *pass through* the vertices $v_0, v_1, \ldots, v_n$.

Each vertex v in a graph determines a walk of length of length zero in the graph, consisting of the single vertex v; such a walk is said to be *trivial*.

Definition Let (V, E) be a graph. A *trail* $v_0\, v_1\, v_2\, \ldots\, v_n$ of length n in the graph from a vertex a to a vertex b is a walk of length n from a to b with the property that the edges $v_{i-1}v_i$ are distinct for $i = 1, 2, \ldots, n$.

A trail in a graph is thus a walk in the graph which traverses edges of the graph at most once.

Definition Let (V, E) be a graph. A *path* $v_0\, v_1\, v_2\, \ldots\, v_n$ of length n in the graph from a vertex a to a vertex b is a walk of length n from a to b with the property that the vertices $v_0, v_1, \ldots, v_n$ are distinct.

A path in a graph is thus a walk in the graph which passes through vertices of the graph at most once.

Definition An undirected graph is said to be *connected* if, given any two vertices u and v of the graph, there exists a path in the graph from u to v.

Definition Let (V, E) be a graph. A *walk* $v_0\, v_1\, v_2\, \ldots\, v_n$ in the graph is said to be *closed* if $v_0 = v_n$.

Thus a walk in a graph is closed if and only if it starts and ends at the same vertex.

Definition Let (V, E) be a graph. A *circuit* in the graph is a non-trivial closed trail in the graph.

Definition A circuit $v_0\, v_1\, v_2\, \ldots\, v_{n-1}\, v_0$ in a graph is said to be *simple* if the vertices $v_0, v_1, v_2, \ldots\, v_{n-1}$ are distinct.

Theorem 29.1 *If a graph has no isolated or pendant vertices then it contains at least one simple circuit.*

Theorem 29.2 *Let u and v be vertices of a graph, where $u \neq v$. Suppose that there exist at least two distinct paths in the graph from u to v. Then the graph contains at least one simple circuit.*

32 Forests and Trees

Definition A graph is said to be *acyclic* if it contains no circuits.

Definition A *forest* is an acyclic graph.

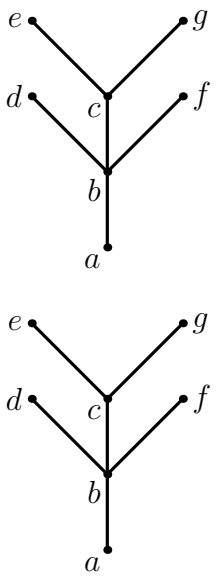
Definition A *tree* is a connected forest.

Note that the components of any forest are trees.

Example The graph (V, E), where

$$\begin{aligned} V &= \{a,\, b,\, c,\, d,\, e,\, f,\, g\}, \\ E &= \{a\,b,\, b\,c,\, b\,d,\, c\,e,\, b\,f,\, c\,g\}, \end{aligned}$$

is a tree.



The vertices a, d, e, f and g are pendent vertices (i.e., each of these vertices is incident to exactly one edge of the graph, and is therefore of degree one.) The tree has 7 vertices and 6 edges.

Theorem 32.1 *Every forest contains at least one isolated or pendent vertex.*

Proof If a graph has no isolated or pendent vertices, then it contains a circuit (Theorem 29.1). But a forest contains no circuits. Therefore must have at least one isolated or pendent vertex. ∎

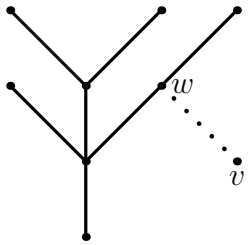
Theorem 32.2 *A non-trivial tree contains at least one pendent vertex.*

Proof A non-trivial graph has more than one vertex. If a non-trivial graph has an isolated vertex then there does not exist any path or walk from that vertex to any other vertex of the graph, and therefore the graph is not connected. But a tree is by definition connected. Therefore a non-trivial tree cannot have any isolated vertex. However a tree is a forest, and therefore contains at least one vertex that is either an isolated vertex or a pendent vertex (Theorem 32.1). Such a vertex must then be a pendent vertex. ∎

Theorem 32.3 *Let (V, E) be a tree. Then $\#(E) = \#(V) - 1$, where $\#(V)$ and $\#(E)$ denote respectively the number of vertices and the number of edges of the tree.*

Proof We can prove the result by induction on the number $\#(V)$ of vertices of the tree. The result is clearly true when the tree is trivial, since it then consists of one vertex and no edges.

Suppose that every tree with m vertices has $m - 1$ edges. Let (V, E) be a tree with $m + 1$ vertices. At least one of these vertices is a pendent vertex (Theorem 32.2). Let v be a pendent vertex, let w be the vertex that is adjacent to v, let $V' = V \setminus \{v\}$, and let $E' = E \setminus \{v\,w\}$. Then (V', E') is a subgraph of (V, E), and this subgraph has m vertices. (This subgraph is obtained from the original graph by deleting the vertex v and the edge $v\,w$ from that graph.) We claim that this subgraph (V', E') is in fact a tree.



First we show that (V', E') is connected. Now, given any two vertices in V', there exists a path in (V, E) from one vertex to the other. This path could not pass through the vertex v, since otherwise the path would have to pass through w twice (going out to v and then returning from v), which is impossible since a path by definition has no repeated vertices. Therefore this path is in fact a path in (V', E'). We conclude that (V', E') is connected.

Now the tree (V, E) does not contain any circuits. It follows immediately that the connected subgraph (V', E') does not contain any circuits, and is thus a tree. It has m vertices.

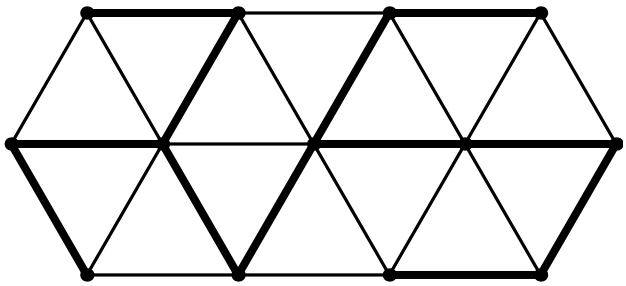
The induction hypothesis now ensures that the tree (V', E') has $m - 1$ edges, and therefore the tree (V, E) has m edges. The required result therefore follows by the Principle of Mathematical Induction. ∎

Theorem 32.4 *Given two distinct vertices of a tree, there exists a unique path in the tree from the first vertex to the the second.*

Proof Let u and v be distinct vertices of the tree. There must exist at least one path in the tree from u to v, since any tree is connected. Were there to exist more than one, then it would follow from Theorem 29.2 that there would exist at least one circuit in the tree, which is impossible, since that a tree cannot contain any circuits. Therefore there must exist exactly one path in the tree from u to v. ∎

33 Spanning Trees

Definition A *spanning tree* in a graph (V, E) is a subgraph of the graph (V, E) that is a tree which includes every vertex of the graph (V, E).



Theorem 33.1 *Every connected graph contains a spanning tree*

Proof Let (V, E) be a connected graph. The collection consisting of all the connected subgraphs of (V, E) with the same vertices as (V, E) is non-empty, since it includes the graph (V, E) itself. Choose a subgraph (V, E') in this collection such that the number $\#(E')$ of edges in this subgraph is less than or equal to the number of edges of any other subgraph in the collection. We claim that (V, E') is the required spanning tree. Clearly (V, E') is connected and has the same vertices as V. It only remains to show that (V, E') does not contain any circuits.

Suppose that (V, E') were to contain a circuit. Let $v\,w$ be an edge traversed by some circuit in (V, E'), and let $E'' = E' \setminus \{v\,w\}$. There would then exist a walk from v to w whose edges belong to E''. (Such a walk could

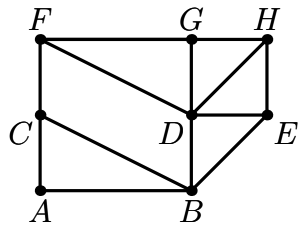
consist of the remaining edges of the circuit traversing the edge $v\,w$.) Moreover every vertex in V could be joined to v by a walk whose edges belong to E', and could therefore be joined either to v or to w by a walk whose edges belong to E''. It would then follow that every vertex of V could be joined to v by a walk whose edges belong to E'', and therefore the graph (V, E'') would be a connected subgraph of (V, E) with the same vertices as (V, E) and with fewer edges than (V, E'), which is impossible. We conclude therefore that the subgraph (V, E') of (V, E) cannot contain any circuits and is therefore the required spanning tree. ∎

Corollary 33.2 *Let (V, E) be a connected graph with $\#(V)$ vertices and $\#(E)$ edges. Suppose that $\#(E) = \#(V) - 1$. Then the graph (V, E) is a tree.*

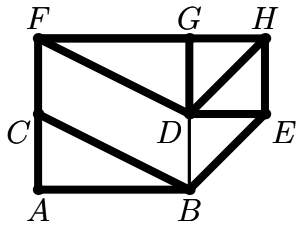
Proof A connected graph (V, E) contains a spanning tree, by Theorem 33.1. This spanning tree must have $\#(V) - 1$ edges, by Theorem 32.3. But the spanning tree then has the same number of edges as the original graph (V, E), and must therefore be the same as this graph. It follows that the graph (V, E) must be a tree, since it is a spanning tree of itself. ∎

The proof of Theorem 33.1 corresponds to an algorithm for finding a spanning tree for a connected graph. The algorithm proceeds as follows. We start with a subgraph consisting of all the vertices and vertices of the original graph. If that subgraph contains a circuit, then we can remove one of the edges of that circuit. The resultant subgraph will still be a connected subgraph of the original graph that includes all the vertices of the original graph. We can then iteratively break remaining circuits in the subgraph, one by one, so that, at each stage of the algorithm, we have a current subgraph that is connected and includes all the vertices of the original graph. We proceed in this fashion until the current subgraph has no more circuits to break. The subgraph will then be the required spanning tree.

Example We find a spanning tree for the connected graph with vertices A, B, C, D, E, F, G, H, and edges $A\,B$, $A\,C$, $B\,C$, $B\,D$, $B\,E$, $C\,F$, $D\,E$, $D\,F$, $D\,G$, $D\,H$, $E\,H$, $F\,G$ and $F\,H$. This graph is pictured below.

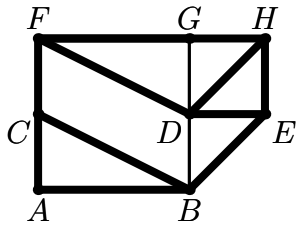


Starting with the current subgraph equal to the given graph, we note that the subgraph has a circuit $B\,C\,F\,D\,B$. We may therefore remove one of the edges of this circuit. Let us therefore remove the edge $B\,D$ from the subgraph. The resultant subgraph is then represented by the thick edges of the diagram below:—

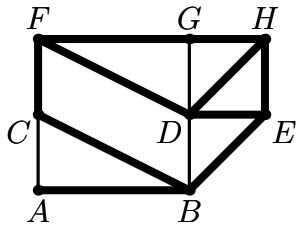


This is the current subgraph for the second removal.

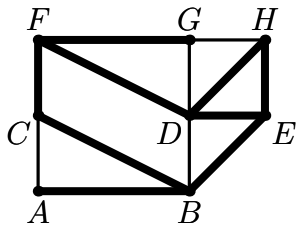
We then break the circuit $D\,F\,G\,D$ of the current subgraph by removing the edge $D\,G$. The resulting subgraph is then the current subgraph for the third removal, and is pictured below.



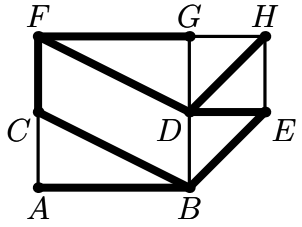
We then break the circuit $A\,B\,C\,A$ of the current subgraph by removing the edge $A\,C$. The resulting subgraph is then the current subgraph for the fourth removal, and is pictured below.



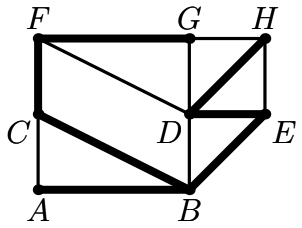
We then break the circuit $D\,F\,G\,H\,D$ of the current subgraph by removing the edge $G\,H$. The resulting subgraph is then the current subgraph for the fifth removal, and is pictured below.



We then break the circuit $B\,C\,F\,D\,H\,E\,B$ of the current subgraph by removing the edge $E\,H$. The resulting subgraph is then the current subgraph for the sixth removal, and is pictured below.

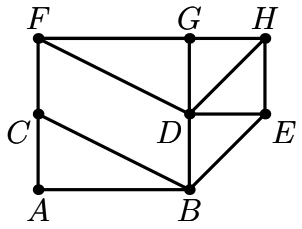


Finally break the circuit $B\,C\,F\,D\,E\,B$ of the current subgraph by removing the edge $D\,F$. The resulting subgraph has no circuits, but is connected and includes all the vertices of the given graph. It is thus a spanning tree for the given graph. This spanning tree is then the subgraph with edges $A\,B$, $B\,C$, $B\,E$, $C\,F$, $D\,E$, $D\,H$, $F\,G$ represented by the thick edges of the following diagram:—

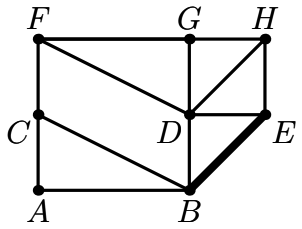


There is an alternative algorithm for finding spanning trees of connected graphs. The procedure is to start with current subgraph of the given graph consisting of just a single vertex. We then add edges one by one, together with any extra vertices incident on those added edges, so as to ensure that, at each stage, the current subgraph is acyclic. When we reach the stage that no further edges can be added to the subgraph without introducing a circuit then the subgraph must be connected and must include all the vertices of the given graph. The subgraph at the final stage must therefore be a spanning tree of the given graph. We illustrate this algorithm by showing how to apply it to find a spanning tree of the graph considered in the previous example.

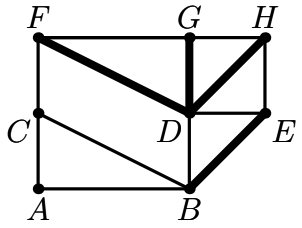
Example We seek a spanning tree of the graph with vertices A, B, C, D, E, F, G, H, and edges $A\,B$, $A\,C$, $B\,C$, $B\,D$, $B\,E$, $C\,F$, $D\,E$, $D\,F$, $D\,G$, $D\,H$, $E\,H$, $F\,G$ and $F\,H$. This graph is pictured below.



We first add the edge $B\,E$ to obtain the acyclic graph pictured below.

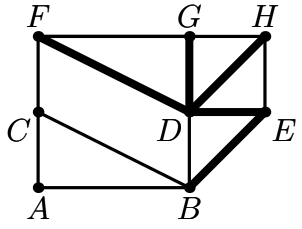


Let us then successively add the edges $D\,F$, $D\,G$ and $D\,H$ (which we can do) to build up the acyclic subgraph, so as to obtain the subgraph pictured below.



It would not then be possible to proceed by adding any of the edges $F\,G$ or $G\,H$ to the acyclic subgraph at the following stage.

We can, for example, add the edge $D\,E$. Adding this edge joins the two components of the acyclic subgraph so as to obtain the acyclic subgraph pictured below.



It would not then be possible to proceed by adding any of the edges $B\,D$, $E\,H$, $F\,G$ or $G\,H$ to the acyclic subgraph at the following stage.

The possibilities for the remaining two steps can then be enumerated as follows:—

(i) add $A\,B$ and then $A\,C$;

(ii) add $A\,B$ and then $B\,C$;

(iii) add $A\,B$ and then $C\,F$;

(iv) add $A\,C$ and then $A\,B$;

(v) add $A\,C$ and then $C\,B$;

(vi) add $A\,C$ and then $C\,F$;

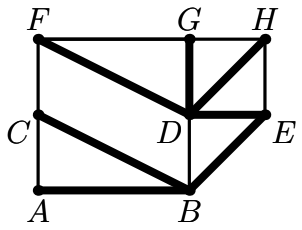
(vii) add $B\,C$ and then $A\,B$;

(viii) add $B\,C$ and then $A\,C$;

(ix) add $C\,F$ and then $A\,B$;

(x) add $C\,F$ and then $A\,C$.

For example, opting for possibility (vii) results in the subgraph with vertices A, B, C, D, E, F, G, H and edges $A\,B$, $B\,C$, $B\,E$, $D\,E$, $D\,F$, $D\,G$ and $D\,H$. This subgraph is pictured below.



We now consider the reasons why adding edges to an acyclic subgraph of a given connected graph so as to ensure that the resulting graph remains acyclic is guaranteed to arrive at a spanning tree for the given connected graph.

Suppose that some connected graph is given and that an acyclic subgraph of the given graph has been constructed. Suppose first that the acyclic subgraph does not contain all the vertices of the given graph. Let **v** be a vertex of the given connected graph that does not belong to the acyclic

subgraph. Then, if any edge incident on the vertex **v** is added to the acyclic subgraph, the resulting subgraph will be acyclic, because the addition of an edge incident on the vertex **v** cannot result in the formation of a cycle in the resulting larger subgraph.

Next suppose that we have constructed an acyclic subgraph that contains all the vertices of the given connected graph. If this acyclic subgraph is connected then it is a spanning tree. Otherwise there will exist a walk in the given connected graph from a vertex in one connected component of the acyclic subgraph to a vertex in some other connected component. This walk must contain at least one edge whose endpoints are in distinct connected components of the acyclic subgraph. If this edge is added to the acyclic subgraph the resultant subgraph will be acyclic. (The addition of the edge with reduce the number of connected components of the acyclic subgraph by one.)

These observations ensure that if we are given a connected graph, if we have constructed an acyclic subgraph, and if it is impossible to add an edge to that acyclic subgraph so as to ensure that the resulting subgraph is also acyclic, then the acyclic subgraph that has been constructed is a spanning tree for the given connected graph.

This methodology leads to an alternative proof of Theorem 33.1, which asserts that any connected graph has a spanning tree. Indeed any connected graph must contain an acyclic subgraph, where the number of edges in that acyclic subgraph is greater than or equal to the number of edges in any other acyclic subgraph of the given connected graph. It will not then be possible to add an edge to the acyclic subgraph so as to obtain a larger acyclic subgraph. It follows that the acylic subgraph with the maximum possible number of edges must be a spanning tree of the given connected graph.

34 Kruskal's Algorithm

Definition Let (V, E) be an undirected graph whose set of vertices is V and whose set of edges is E. A *cost* function $c\colon E \to \mathbb{R}$ on the set E of edges of the graph is a function that assigns to each edge e of the graph a real number $c(e)$.

Let $c\colon E \to \mathbb{R}$ be a cost function on the set E of edges of a graph (V, E). Given any subset S of E, we define

$$c(S) = \sum_{e \in S} c(e).$$

(Thus $c(S)$ is the sum of the costs of all the edges of the graph that belong to the subset E.)

Let (V, E) be a connected graph. We recall that a subgraph of (V, E) is said to be a *spanning tree* if it is a tree that includes all the vertices of the given connected graph. Thus a subgraph of that given graph is a spanning tree if and only if it is a connected acyclic subgraph of the given graph that includes all the vertices of the given graph.

Definition Let (V, E) be a connected graph on which is defined a cost function $c: E \to \mathbb{R}$ that assigns a cost $c(e)$ to each edge e of the graph. A spanning tree (V_M, E_M) is said to be *minimal* (with respect to this cost function) if $c(E_M) \leq c(E_T)$ for all spanning trees (V_T, E_T) of the given graph.

We discuss *Kruskal's Algorithm* for finding minimal spanning trees. Let (V, E) be a graph, and let $c: E \to \mathbb{R}$ be a cost function defined on the set E of edges of the graph (V, E). We start with a subgraph that initially consists of all the vertices of the graph (with no edges). List the edges of the graph in a *queue* so that the the edges is non-decreasing in the queue. (Thus if e and e' are edges of the graph, and if $c(e) < c(e')$ then e precedes e' in the queue.)

Take edges successively from the front of the queue, and determine whether or not addition of that edge to the current subgraph will create a cycle. If such a cycle would be created then discard it; otherwise add it to the subgraph. Continue till the queue is emptied.

The algorithm described always yields a minimal spanning tree for the given graph.

We now justify this assertion.

Let the edges in the queue that are added to the subgraph as that subgraph is built up be ordered in sequence as $e_1, e_2, e_3, \ldots, e_m$. The spanning tree ultimately constructed then consists of the vertices of the original graph, together with the edges $e_1, e_2, \ldots, e_m$. Moreover $c(e_i) \leq c(e_j)$ whenever $i < j$. Also if e' is an edge of the original graph, if $c(e') \leq c(e_k)$ for some integer j betweeen 2 and m, and if the subgraph consisting of the $e_1, e_2, \ldots, e_{m-1}$ and e' together with the endpoints of those edges would contain a cycle, then the edge e' is disarded by the Kruskal algorithm, and is thus not included in the spanning tree constructed.

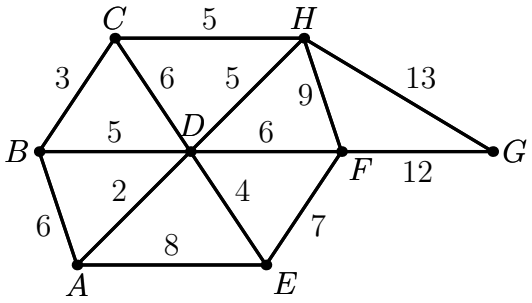
The algorithm described always yields a minimal spanning tree for the given graph.

We now justify this assertion.

Let the edges in the queue that are added to the subgraph as that subgraph is built up be ordered in sequence as $e_1, e_2, e_3, \ldots, e_m$. The spanning

tree ultimately constructed then consists of the vertices of the original graph, together with the edges $e_1, e_2, \ldots, e_m$. Moreover $c(e_i) \leq c(e_j)$ whenever $i < j$. Also if e' is an edge of the original graph, if $c(e') \leq c(e_k)$ for some integer j betweeen 2 and m, and if the subgraph consisting of the $e_1, e_2, \ldots, e_{m-1}$ and e' together with the endpoints of those edges would contain a cycle, then the edge e' is disarded by the Kruskal algorithm, and is thus not included in the spanning tree constructed.

Example We apply Kruskal's Algorithm to find a minimal spanning tree for the following graph:—



(Costs are specified next to the relevant edge.)

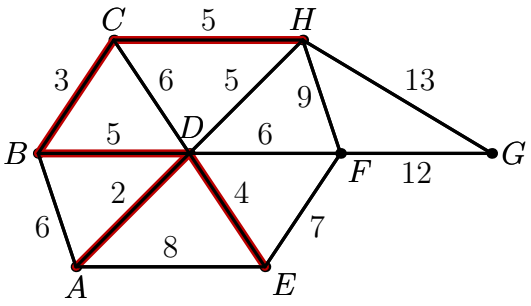
We order the edges so that the associated costs are non-decreasing. The edges are listed in order with their associated costs as follows:—

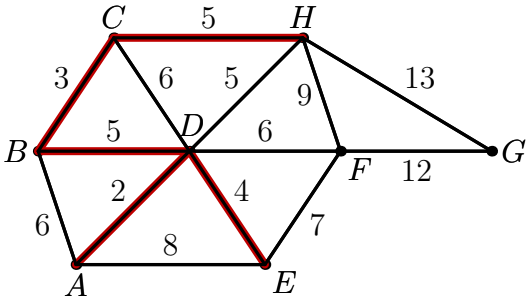
AD	BC	DE	BD	CH	DH	AB
2	3	4	5	5	5	6

CD	DF	EF	AE	FH	FG	GH
6	6	7	8	9	12	13

We build up an acyclic subgraph of the given connected graph as follows. We start with an acyclic subgraph consisting of the vertices of the original graph. We then treat the above list of edges as a queue, taking edges in turn from the head of the queue, and add them to the subgraph if and only if doing so does not create any circuits in the subgraph.

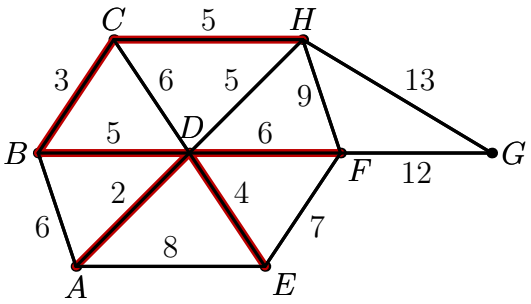
In the first five iterations we add the edges AD, BC, DE, BD and CH. No circuit is created at any stage, and the resultant acyclic subgraph is represented by the vertices and thick edges in the following diagram:—



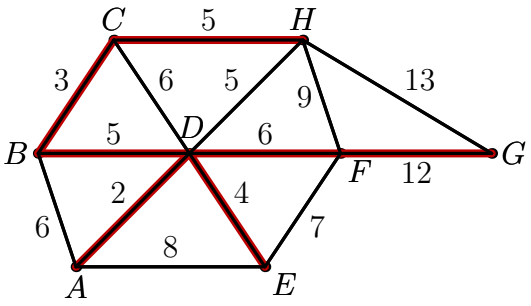


Adding the edge DH would create a circuit $DBCHD$; therefore the edge DH is discarded. Adding the edge AB would create a circuit $ABDA$; therefore the edge AB is discarded. Adding the edge CD would create a circuit $DBCD$; therefore the edge CD is discarded.

The next edge in the queue is DF. We can add this edge to the acyclic subgraph. However the edges EF, AE and FH must be discarded, since adding any of those edges to the subgraph would create a circuit. The current acyclic subgraph is now as follows:—



We add the edge FG to the subgraph, as this is the next in the queue that may be added without creating a circuit in subgraph. We cannot then add GH. Thus the minimal spanning tree generated by Kruskal's Algorithm is as follows:—



The minimal spanning tree generated by Kruskal's Algorithm thus consists of the vertices A, B, C, D, E, F G and H of the given connected graph, together with the edges e_1, e_2, e_3, e_4, e_5, e_6 and e_7, where $e_1, \ldots, e_7$ denote the edges of the minimal spanning tree listed in the order in which they were added to the acyclic graph, so that

$$e_1 = AD, \quad e_2 = BC, \quad e_3 = DE, \quad e_4 = BD,$$

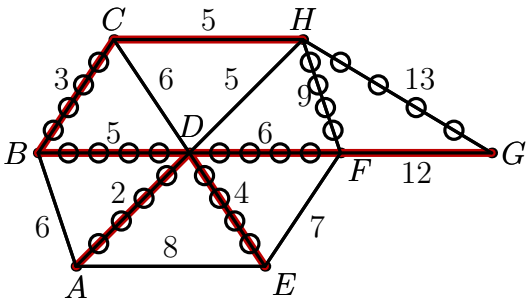
$$e_5 = CH, \quad e_6 = DF, \quad e_7 = FG.$$

We refer to the spanning tree generated by Kruskal's Algorithm as the *Kruskal tree*.

However we have not yet shown that the Kruskal tree has minimal cost.

We claim that, given any integer k satisfying $0 < k \leq 7$, and given any spanning tree that includes edges e_i of the Kruskal tree whenever $i < k$ but does not include the edge e_k, this spanning tree can be modified to yield a spanning tree that includes edges e_i of the Kruskal tree for $i \leq k$, where the cost of the modified tree does not exceed that of the given spanning tree.

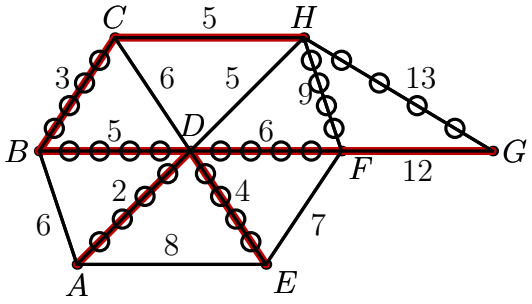
We consider the example below. The spanning tree T represented by the edges with circles includes edges e_1, e_2, e_3 and e_4 of the Kruskal tree (i.e., the edges AD, BC, DE and BD), but does not include the edge e_5 where $e_5 = CH$:–



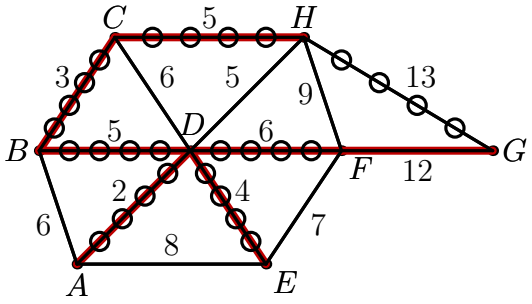
Adding any edge to a spanning tree creates a circuit. In particular, if we add the edge CH to the spanning tree T, then the resultant subgraph of the

original connected graph must contain a circuit. In the present example the circuit created is $CHFDBC$. Now not all edges of that circuit can belong to the Kruskal tree, because trees cannot contain circuits. Therefore at least one edge of the circuit $CHFDBC$ does not belong to the Kruskal tree.

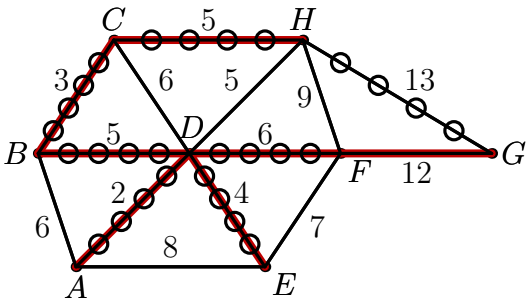
In the example under consideration, the edge FH does not belong to the Kruskal tree. Now if the cost of FH were less than that of CH then the Kruskal algorithm would have required the edge FH to be added to the Kruskal tree before CH. It follows that the cost of the edge FH cannot be less than CH. Indeed FH has cost 9, whereas CH has cost 5.



It follows that we can modify the spanning tree T to obtain a new spanning tree T', where T' includes the edges e_1, e_2, e_3, e_4 and e_5 of the Kruskal tree, and where the cost of T' does not exceed that of T. The resultant tree T' consists of the edges with circles in the following figure:—



The spanning tree T' includes edges e_i of the Kruskal spanning tree for $i < 7$ but does not include the edge e_7, where $e_7 = FG$. Now addition of the edge e_7 to the spanning tree creates a circuit $FGHCBDF$ in the resultant subgraph. This circuit cannot consist entirely of edges of the Kruskal tree. Therefore at least one edge of the circuit does not belong to the Kruskal tree. That edge is GH.



The cost of the edge GH cannot exceed that of the edge e_7, because otherwise GH would have been added to the Kruskal tree before FG was considered in the application of Kruskal's Algorithm. Therefore replacement of GH by FG results in a spanning tree T'' whose cost does not exceed that of T'. This spanning tree T'' then includes all the edges of the Kruskal tree, and must therefore be identical to the Kruskal tree. It follows that the cost of the Kruskal tree cannot exceed that of the tree T', and therefore cannot exceed that of the tree T.

We claim that the procedure just described can be applied to demonstrate that the cost of the Kruskal tree is less than or equal to the cost of any spanning tree for the given connected graph.

We now analyze Kruskal's Algorithm in order to show that if that algorithm is applied to a given connected graph, then the spanning tree generated by that algorithm minimizes cost.

First we recall the specification of the algorithm.

We are given a connected graph. Let V denote the set of vertices of the given graph, and let E denote the set of edges of the given graph. There is a cost function $c\colon E \to \mathbb{R}$ defined on the set of edges of the given graph. The cost of any spanning tree is defined to be the sum of the costs of the edges of that spanning tree. The objective is to find a spanning tree of the connected graph whose cost is less than or equal to that of every other spanning tree of the connected graph.

To implement Kruskal's Algorithm we order the edges of the given connected graph in a finite sequence, or *queue*, so that, given any pair of edges e and e' for which $c(e) < c(e')$, the edge e precedes the edge e' in the queue.

We start with a acyclic subgraph of the given connected graph consisting of all the vertices of the given graph. We build up an acyclic subgraph through the addition of edges. This initial subgraph has no edges. We then take edges in order from the the front of the queue. Having taken an edge from the front of the queue, we determine whether or not addition of that edge to the current acyclic subgraph would create a circuit in the resultant

graph. If a circuit would be created, then we discard the edge. Otherwise we add the edge to the acyclic subgraph so as to create a larger acyclic subgraph. We continue until the queue has been exhausted.

Proposition 34.1 *Let a connected graph be given, together with a cost function defined on its set of edges, and let Kruskal's Algorithm be applied to determine a spanning tree K. Let T be a spanning tree in the connected graph that does not coincide with the spanning tree K resulting from application of Kruskal's Algorithm. Then there exists a spanning tree T', where the cost of T' does not exceed that of T, such that T' includes more edges of K than does T.*

Proof Let us refer to the spanning tree of the given connected graph constructed using Kruskal's Algorithm as the *Kruskal spanning tree*.

Let the given connected graph have $m+1$ vertices. Then any spanning tree for this graph has m edges and $m+1$ vertices (see Theorem 32.3). Moreover any connected subgraph of the given connected graph with m edges and $m+1$ vertices is a spanning tree for that graph (see Corollary 33.2).

Let the edges of the Kruskal spanning tree be denoted by $e_1, e_2, \ldots, e_m$, where the order of these edges is the order established by the queue constructed in applying the algorithm.

Now let T be an spanning tree for the given connected graph that is distinct from the Kruskal spanning tree. Both spanning trees have the same number of edges. It follows that the Kruskal spanning tree must have at least one edge that does not belong to the spanning tree T. Therefore there exists some integer k satisfying $0 \leq k \leq m$ such that e_i is an edge of T whenever $i < k$ but e_k is not an edge of T.

Because T is connected, there is a path in T between any two vertices of T. It follows that if the edge e_k is added to T, then the resultant graph contains a circuit. This circuit must include the edge e_k. But it cannot be contained within the Kruskal spanning tree, because the Kruskal spanning tree is acyclic. Therefore the circuit must include at least one edge e' that does not belong to the Kruskal spanning tree.

Now there cannot exist any circuit in the subgraph of the given connected graph consisting of the vertices of that graph, the edge e' and the edges e_i for $i < k$, because that subgraph is contained in the spanning tree T. Had it been the case that $c(e') < c(e_k)$, then e' would have preceded e_k in the queue in the application of Kruskal's algorithm, and it would accordingly have been added to the Kruskal spanning tree. Because e' was not added to the Kruskal spanning true, it must be the case that $c(e') \geq c(e_k)$.

Suppose we modify the spanning tree T by adding the edge e_k and then removing the edge e' to obtain a subgraph T' of the given connected graph.

Once the edge e_k is added, the resultant graph contains a circuit which includes the edge e'. The removal of e' then breaks the circuit, leaving behind a connected graph T' with m edges and $m+1$ vertices. This connected graph T' must be a spanning tree of the given connected graph. Its cost cannot exceed that of the graph T, because $c(e') \geq c(e_k)$. Moreover T' includes edges the e_i of the Kruskal spanning tree corresponding to all positive integers i satisfying $i \leq k$. Moreover the spanning tree T' contains more edges of the Kruskal spanning tree than does the spanning tree T, because an edge e' that does not belong to the Kruskal spanning tree has been replaced by an edge e_k that does. The result follows. ∎

Theorem 34.2 *Let a connected graph be given, together with a cost function defined on its set of edges, and let Kruskal's Algorithm be applied to determine a spanning tree K. Then the cost of the spanning tree generated by Kruskal's Algorithm is less than or equal to that of every other spanning tree for the given connected graph.*

Proof The number of possible spanning trees of the connected graph is finite. There is therefore a well-defined real number that is the minimum of the costs of all spanning trees for the given connected graph. Moreover there exists a spanning tree T with this minimum cost that has the maximum possible number of edges in common with the spanning tree K generated by Kruskal's Algorithm. But then T must coincide with K.

Indeed if it were the case that T did not coincide with K, then Proposition 34.1 would guarantee the existence of a spanning tree T', where the cost of T' does not exceed that of T, such that T' includes more edges of K than does T. The cost of T' would then also be the minimum of the costs of all spanning trees for the given connected graph. But the existence of such a spanning tree T' would contradict the choice of T as the spanning tree of minimum cost with the maximum possible number of edges in common with K. We conclude therefore that T must coincide with K, and therefore the cost of K is less than or equal to every other spanning tree. The result follows. ∎

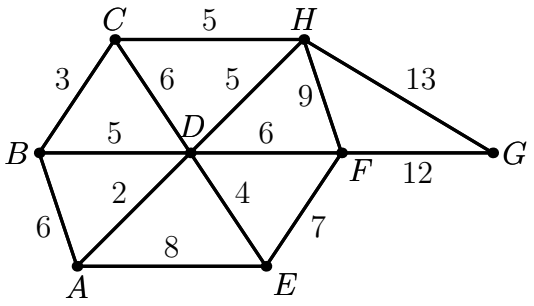
35 Prim's Algorithm

There is an alternative algorithm for constructing a spanning tree of minimal cost for a connected graph. This algorithm is known as *Prim's* Algorithm. It was first discovered by Vojtěch Jarník and published by him in 1930. It was subsequently rediscovered and published by Robert Prim and published by him in 1957. It was again rediscovered by Edsger Dijkstra in 1959.

A connected graph is given. A cost function is defined on the edges of the graph. To apply Prim's Algorithm, one first orders the edges of the graph so that if e and e' are edges of the graph, and if their costs $c(e)$ and $c(e')$ satisfy $c(e) < c(e')$, then e precedes e' in the ordering.

A vertex of the graph is chosen. Each successive iteration of the algorithm we have a subgraph that is a tree. We then identify the first edge in the chosen ordering which has one vertex included in the current subgraph and the other vertex not included in that subgraph. We then add that edge to the current subgraph, together with the endpoint of that edge that is not in the current subgraph. The resultant subgraph of the given connected graph will then be a tree. We continue this process until we can proceed no further. At that point all vertices of the given connected graph will be in the subgraph, and therefore the subgraph at that iteration will be a spanning tree. It can be shown that this spanning tree minimises cost amongst all spanning trees of the given connected graph.

Example We apply Prim's Algorithm to find a minimal spanning tree for the following graph:—



(Costs are specified next to the relevant edge.)

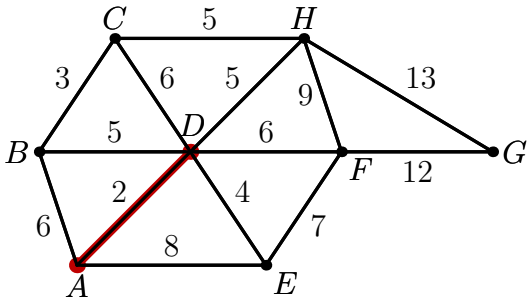
We order the edges so that the associated costs are non-decreasing. The edges are listed in order with their associated costs as follows:—

AD	BC	DE	BD	CH	DH	AB
2	3	4	5	5	5	6
CD	DF	EF	AE	FH	FG	GH
6	6	7	8	9	12	13

We build up an acyclic subgraph of the given connected graph as follows. We start with a subgraph consisting of a single vertex. We then build up in the graph a succession of subgraphs that are trees (i.e., connected acyclic graphs). At each iteration we add to the current subtree the first edge in the

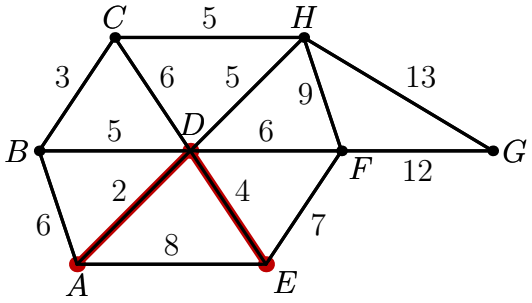
above list that joins a vertex in that subtree to a vertex not in that subtree. We continue till we can go no further. We will then have constructed a spanning tree for the given connected graph.

We start with the vertex A. In the first iteration we add the edge AD, obtaining the tree in the connected graph indicated by the thick edge in the following diagram:—

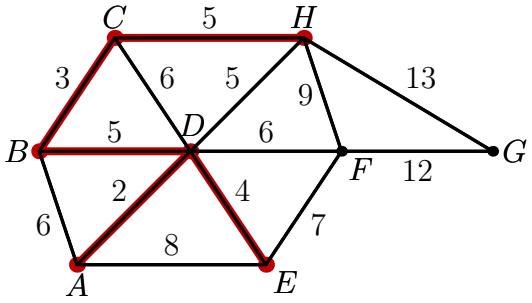


It is not then possible, applying Prim's Algorithm, to add the edge BC, because adding this edge would result in an acyclic subgraph. The first edge in the list that we can add is the edge DE.

We add the edge DE to obtain the tree in the given connected graph represented by the thick edges in the following diagram:—

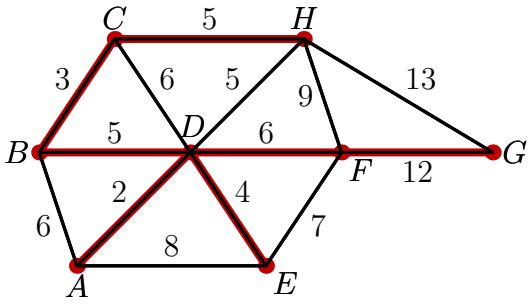


In the next step we add the edge BD of cost 5. We can then add the edge BC of cost 3 and the edge CH of cost 5 to obtain the following tree:—



The edges of the tree obtained after applying the first five steps of Prim's algorithm are the same as those obtained after applying the first five steps of Kruskal's Algorithm.

At the final two iterations we successively add the edges DF and FG so as to obtain a minimal spanning tree that in this example is identical to that generated by Kruskal's Algorithm.



The minimal spanning tree generated by Prim's Algorithm thus consists of the vertices A, B, C, D, E, F G and H of the given connected graph, together with the edges e_1, e_2, e_3, e_4, e_5, e_6 and e_7, where $e_1, \ldots, e_7$ denote the edges of the minimal spanning tree listed in the order in which they were added to the acyclic graph, so that

$$e_1 = AD, \quad e_2 = DE, \quad e_3 = BD, \quad e_4 = BC,$$

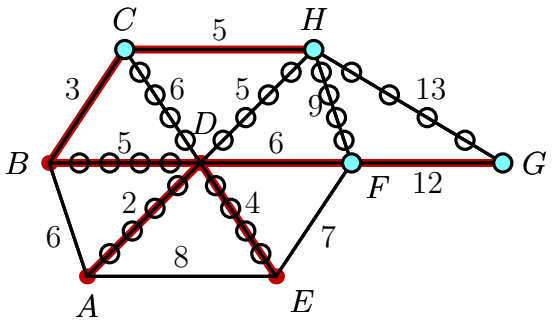
$$e_5 = CH, \quad e_6 = DF, \quad e_7 = FG.$$

We refer to the spanning tree generated by Prim's Algorithm as the *Prim spanning tree*.

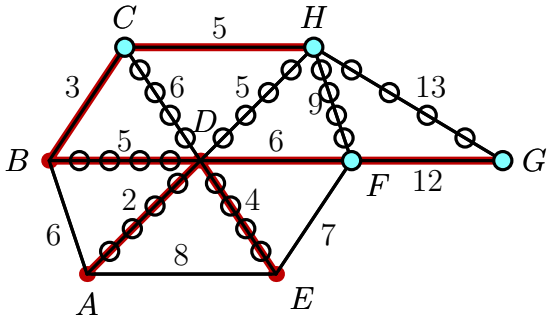
We now consider an example to show how successive modifications can convert an arbitrary spanning tree for the connected graph into the Prim spanning tree, whilst ensuring that the cost of each successive modified spanning tree does not exceed that of the spanning tree from which it is derived.

Now after the jth iteration of Prim's Algorithm results in a subgraph of the given connected graph that is a tree with edges $e_1, e_2, \ldots, e_j$. This tree has $j + 1$ vertices. We refer to those vertices as the *visited vertices* after the jth iteration. The remaining vertices of the given connected graph are then referred to as the *unvisited vertices* after the jth iteration.

We start with the spanning tree T, where T consists of all vertices of the original graph together with the edges AD, DE, BD, CD, DH, FH and GH. It is represented by the edges with circles in the following diagram:—



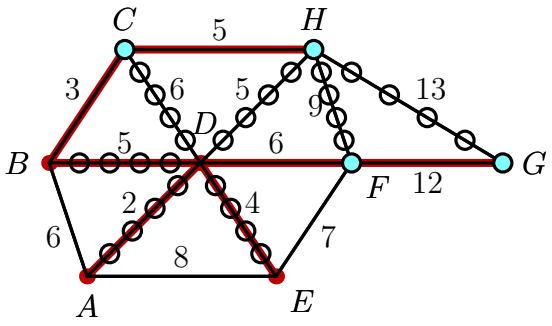
The first three edges added to the Prim spanning tree are also included in the spanning tree just specified. These edges are AD, DE and BD, which are the edges e_1, e_2 and e_3 respectively are the first three edges added to the Prim spanning tree. Now the edge BC is the fourth edge e_4 added to the Prim spanning tree. Addition of this edge to the tree T creates a circuit $BCDB$.



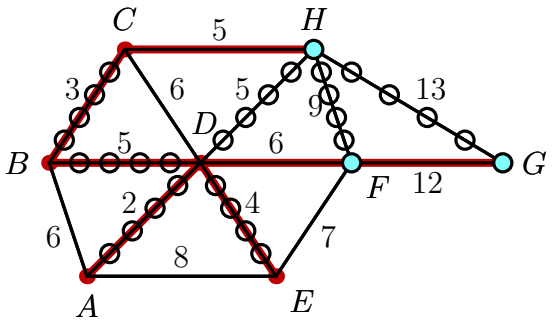
Let the vertices A, B, D, E incident on the edges AD, DE, BD be referred to as the *visited vertices* after the first three iterations of Prim's Algorithm (with the given ordering of edges). Let the remaining vertices C, F, G, H be referred to as the *unvisited vertices* after the first three iterations.

('Visited vertices' are indicated by solid dark red disks, and 'unvisited vertices' by light cyan disks bordered in black on the following diagrams.)

The circuit $BCDB$ cannot be contained within the Prim spanning tree, because a spanning tree has no circuits. The edge BC joins the visited vertex B to the unvisited vertex C. Some other edge of the circuit must also join a visited vertex to an unvisited vertex. At this stage of the example under discussion that edge is the edge CD.

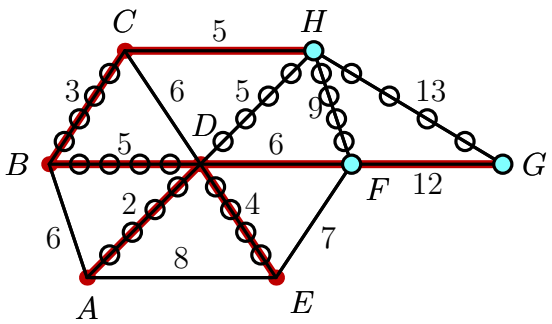


Had the cost of CD been less than that of BC then Prim's Algorithm would have added CD to the Prim spanning tree in place of BC. This did not happen. Therefore the cost of BC does not exceed that of CD, and indeed the costs of BC and CD are 3 and 6 respectively. We now modify the tree T, replacing CD by BC, to obtain the tree T' indicated by circles in the following diagram:—

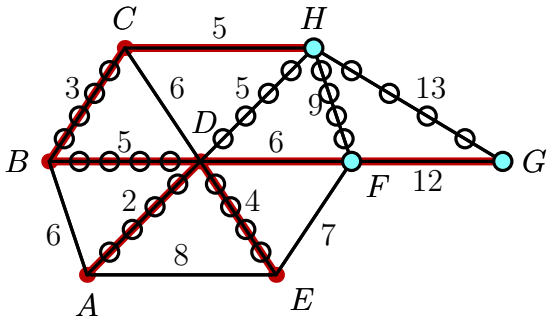


The modified spanning tree T' includes the edges e_1, e_2, e_3 and e_4 of the Prim spanning tree, and its cost does not exceed that of the tree T.

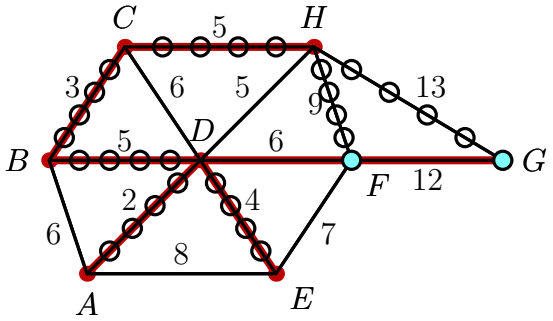
Now the fifth edge e_5 added to the Prim spanning tree is the edge CH. Addition of this edge to the tree T' creates a circuit $CHDBC$ in the resultant subgraph.



Now the visited vertices after the fourth iteration of the Kruskal algorithm are A, B, C, D and E, and the unvisited vertices are F, G and H. The edge CH joins a visited vertex to an unvisited vertex. At least one other edge of the circuit $CHDBC$ must also join a visited vertex to an unvisited vertex. The edge DH does so.

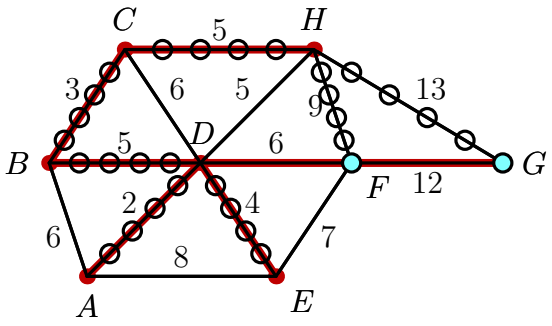


Had the cost of DH been less than that of CH then Prim's Algorithm would have added DH to the Prim spanning tree in place of CH. This did not happen. Therefore the cost of CH does not exceed that of DH, and indeed the costs of CH and DH are 5 and 5 respectively. We now modify the tree T', replacing DH by CH, to obtain the tree T'' indicated by circles in the following diagram:—

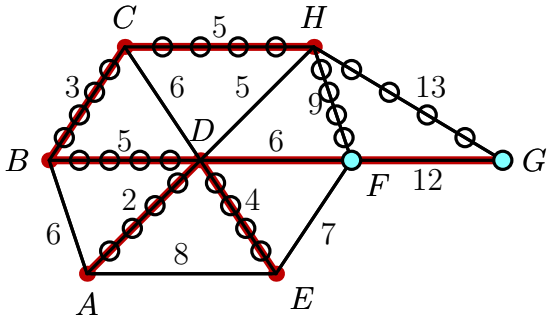


The modified spanning tree T'' includes the edges e_1, e_2, e_3, e_4 and e_5 of the Prim spanning tree, and its cost does not exceed that of the tree T'. It follows that the cost of T'' does not exceed that of T.

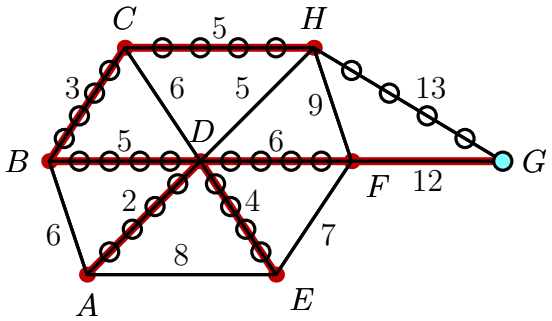
Now the sixth edge e_6 added to the Prim spanning tree is the edge DF. Addition of this edge to the tree T'' creates a circuit $DFHCBD$ in the resultant subgraph.



Now the visited vertices after the fifth iteration of the Kruskal algorithm are A, B, C, D, E and H, and the unvisited vertices are F and G. The edge DF joins a visited vertex to an unvisited vertex. At least one other edge of the circuit $DFHCBD$ must also join a visited vertex to an unvisited vertex. The edge FH does so.

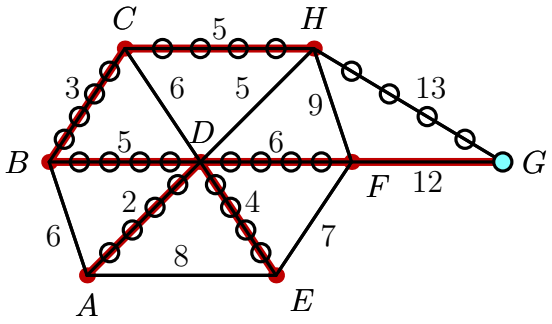


Had the cost of FH been less than that of DF then Prim's Algorithm would have added FH to the Prim spanning tree in place of DF. This did not happen. Therefore the cost of DF does not exceed that of FH, and indeed the costs of DF and FH are 6 and 9 respectively. We now modify the tree T'', replacing FH by DF, to obtain the tree T''' indicated by circles in the following diagram:—

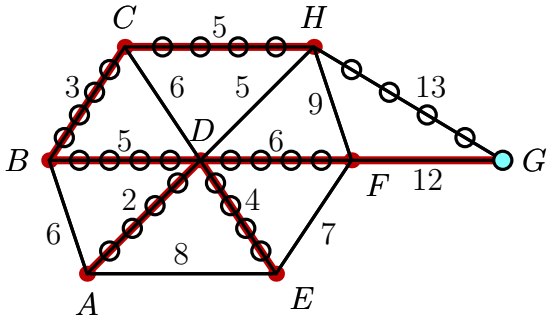


The modified spanning tree T''' includes the edges e_1, e_2, e_3, e_4, e_5 and e_6 of the Prim spanning tree, and its cost does not exceed that of the tree T''. It follows that the cost of T''' does not exceed that of T.

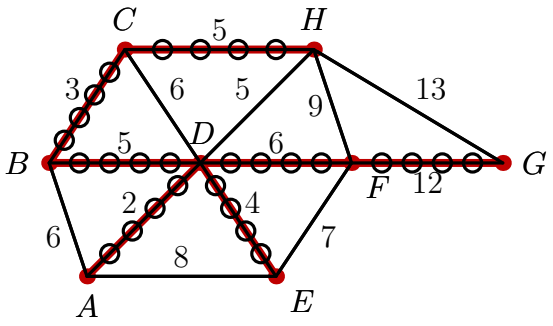
Now the seventh edge e_7 added to the Prim spanning tree is the edge FG. Addition of this edge to the tree T''' creates a circuit $FGHCBDF$ in the resultant subgraph.



Now the visited vertices after the sixth iteration of the Kruskal algorithm are A, B, C, D, E, F and H, and sole unvisited vertex is G. The edge FG joins a visited vertex to an unvisited vertex. At least one other edge of the circuit $FGHCBDF$ must also join a visited vertex to an unvisited vertex. The edge GH does so.



Had the cost of GH been less than that of FG then Prim's Algorithm would have added GH to the Prim spanning tree in place of FG. This did not happen. Therefore the cost of FG does not exceed that of GH, and indeed the costs of FG and GH are 12 and 13 respectively. We now modify the tree T''', replacing GH by DF, to obtain the tree T'''' indicated by circles in the following diagram:—



The modified spanning tree T'''' includes all seven edges of the Prim spanning tree, and its cost does not exceed that of the tree T'''. It follows that the cost of T'''' does not exceed that of T. But T'''' actually coincides with the Prim spanning tree. Thus the cost of the Prim spanning tree P does not exceed that of the tree T.

Given any spanning tree in the given connected graph, an analogous procedure can be followed to modify it in stages, without increasing cost at any stage, so that the ultimate modification yields the Prim spanning tree itself. The following proposition establishes this fact.

Proposition 35.1 *Let a connected graph be given, together with a cost function defined on its set of edges, and let Prim's Algorithm be applied to determine a spanning tree P. Let $e_1, e_2, \ldots, e_m$ denote the edges of the spanning tree P generated by Prim's Algorithm, listed in the order in which they are added to that spanning tree. Let T be a spanning tree in the connected graph that does not coincide with the spanning tree P resulting from application of Prim's algorithm, and let k be the smallest positive integer for which T does not include the edge e_k of P. Then there exists a spanning tree T', where the cost of T' does not exceed that of T, such that T' includes the edges $e_1, e_2, \ldots, e_k$ of P.*

Proof Let us refer to the spanning tree P generated by the application of Prim's algorithm as the *Prim spanning tree*. This spanning tree has m edges, and therefore the given connected graph has $m + 1$ vertices (see Theorem 32.3). Moreover any connected subgraph of the given connected graph with m edges and $m+1$ vertices is a spanning tree for that graph (see Corollary 33.2).

The spanning tree T is a connected subgraph of the given connected graph containing all the vertices of that given connected graph. Therefore there is a path in T between any two vertices of T. It follows that if any edge is added to T then the resultant graph will contain a circuit.

Let the vertices of the given connected graph that are incident on the edges $e_1, e_2, \ldots, e_{k-1}$ be referred to as *visited vertices*, and let the remaining vertices of the graph be referred to as *unvisited vertices*. (The visited vertices are those that have been 'visited' once the first $k-1$ edges have been added to the Prim spanning tree.)

Prim's Algorithm then ensures that the edge e_k joins a visited vertex to an unvisited vertex. Moreover the cost of the edge e_k is less than or equal to the cost of any other edge of the given connected graph that joins a visited to an unvisited vertex.

Now suppose that we add the edge e_k to T to obtain a subgraph of the given connected graph which we denote by $T + e_k$. This graph $T + e_k$ has a circuit. This circuit includes the edge e_k and therefore there are both visited and unvisited vertices included in the circuit. The circuit must therefore include at least one other edge e' besides the edge e_k that joins a visited vertex to an unvisited vertex.

Now the edge e_k added to the Prim spanning tree at the kth stage must minimize cost amongst all edges of the given connected graph that join a visited vertex to an unvisited vertex. Therefore the costs $c(e_k)$ and $c(e')$ of the edges e_k and e' respectively satisfy $c(e_k) \leq c(e')$.

Let T' be the subgraph of the given connected graph obtained by removing the edge e' from $T + e_k$. Then T' is connected, because the edge e' is included in a circuit within the graph $T + e_k$. Also T' has m edges and $m+1$ vertices. It is therefore a spanning tree. The cost of T' is less than or equal to that of T. And the spanning tree T' contains the edges $e_1, e_2, \ldots, e_k$. The result follows. ∎

Theorem 35.2 *Let a connected graph be given, together with a cost function defined on its set of edges, and let Prim's Algorithm be applied to determine a spanning tree. Then the cost of the spanning tree generated by Prim's Algorithm is less than or equal to that of every other spanning tree for the given connected graph.*

Proof Let $e_1, e_2, \ldots, e_m$ be the edges of the spanning tree P generated by Prim's Algorithm, listed in the order in which they are added to that spanning tree through the application of that algorithm. Because the number of spanning trees of the given connected graph is finite, there is a well-defined real number that is the minimum cost of any spanning tree of the given graph.

There then exists a spanning tree T with minimal cost which maximizes the number k for which T contains edges $e_1, e_2, \ldots, e_{k-1}$ of the spanning tree P generated by Prim's Algorithm. It then follows from Proposition 35.1,

together with the maximality of k, that the spanning tree T must include all the edges of the Prim spanning tree P and must therefore coincide with the Prim spanning tree. Therefore the Prim spanning tree has minimal cost, as required. ∎

Remark Let a connected graph be given, together with a cost function on the vertices of the graph. Suppose that no two edges of this graph have the same cost. Then there is only one ordering of the edges of that graph consistent with the requirement that whenever e and e' are edges of the graph whose costs $c(e)$ and $c(e')$ satisfy $c(e) < c(e')$ then $e < e'$. We can apply Prim's Algorithm to construct a minimal spanning tree. We refer to this minimal spanning tree as the *Prim spanning tree*.

Let T be a spanning tree that is distinct from the Prim spanning tree. If we apply to T the procedure used in the proof of Proposition 35.1 to construct a modified spanning tree T', then, in this situation where no two edges of the given connected graph have the same cost, an appropriate edge of T is replaced in T' by an edge of the Prim spanning tree whose cost is strictly lower, and therefore the cost of the modified spanning tree T' is strictly less than that of the tree T. It follows that if T does not coincide with the Prim spanning tree then T cannot itself be a minimal spanning tree.

We conclude from this that if no two edges of the given connected graph have the same cost then the minimal spanning tree of that graph is uniquely determined.