MA2C03 Mathematics
School of Mathematics, Trinity College
Hilary Term 2016
Lecture 39 (February 3, 2016)

David R. Wilkins

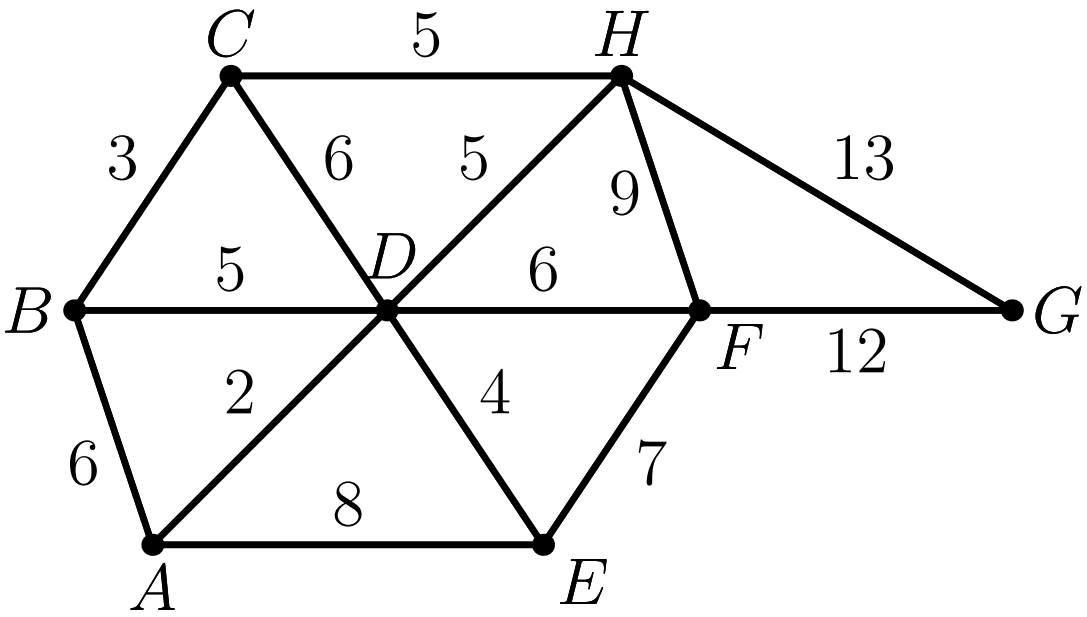
35. Prim's Algorithm

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:—



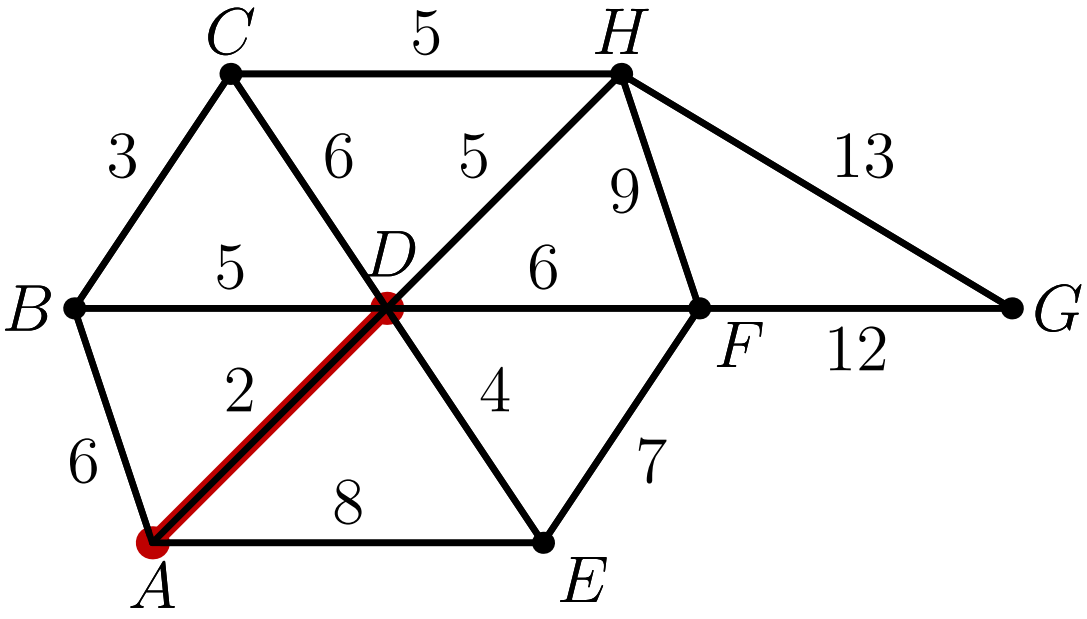
(Costs are specified next to the relevant edge.)

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

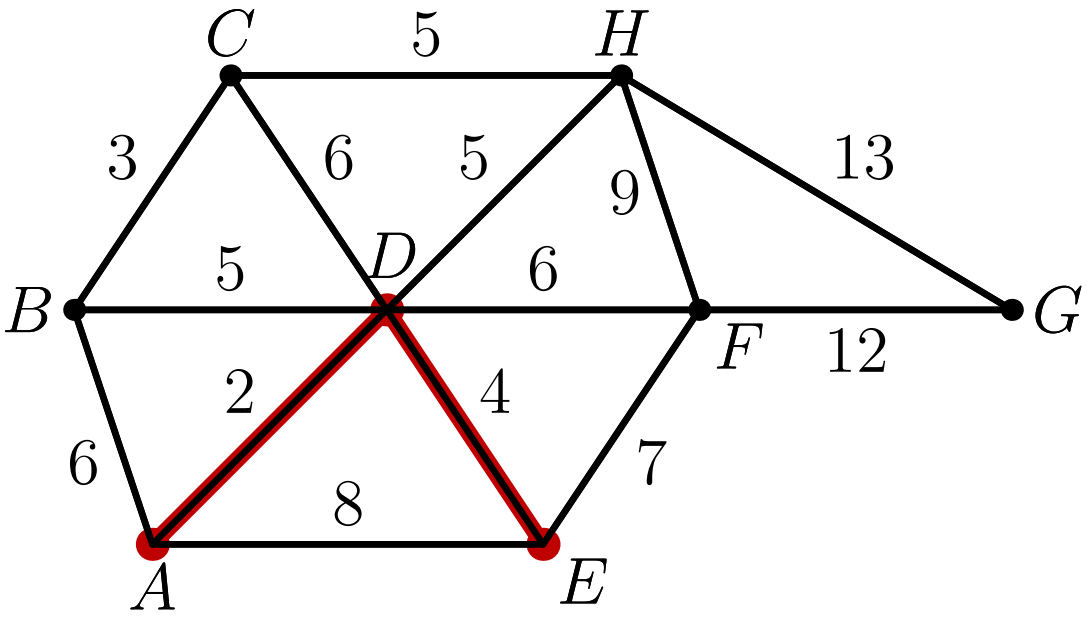
We build up an acyclic subgraph of the given connected graph as follows. We start with a subgraph consisting of a single vertex. We then build up in the graph a succession of subgraphs that are trees (i.e., connected acyclic graphs). At each iteration we add to the current subtree the first edge in the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

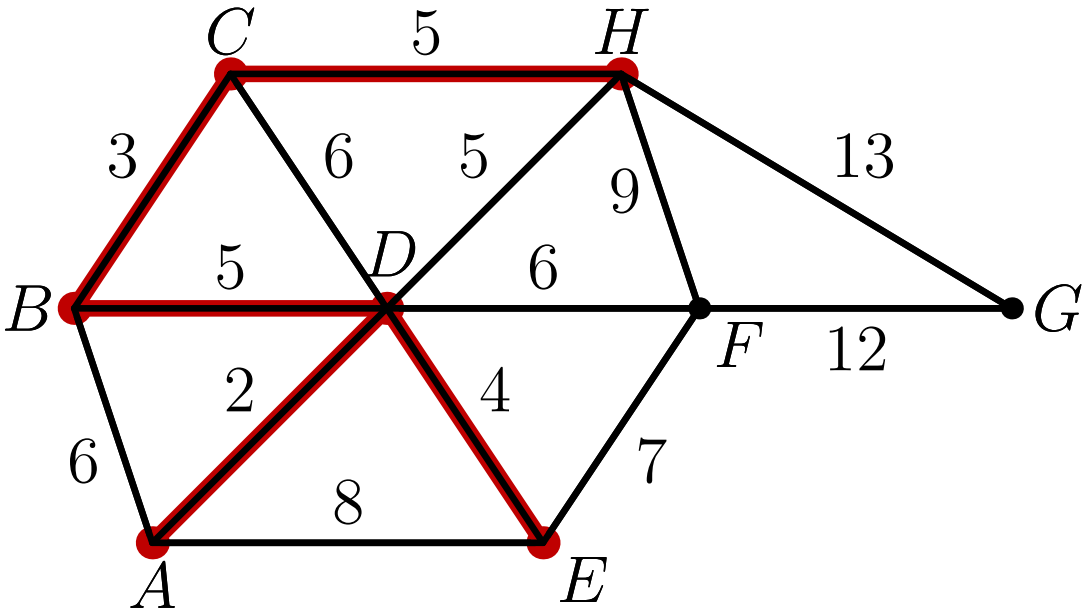


It is not then possible, applying Prim's Algorithm, to add the edge BC, because adding this edge would result in an acyclic subgraph. The first edge in the list that we can add is the edge DE.

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

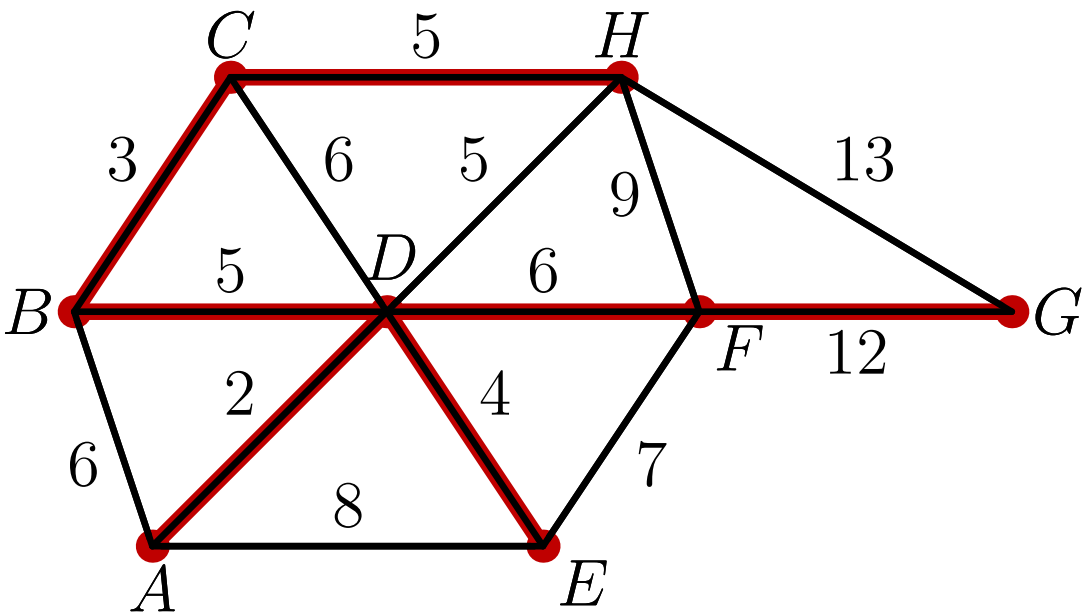


In the next step we add the edge BD of cost 5. We can then add the edge BC of cost 3 and the edge CH of cost 5 to obtain the following tree:—



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

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

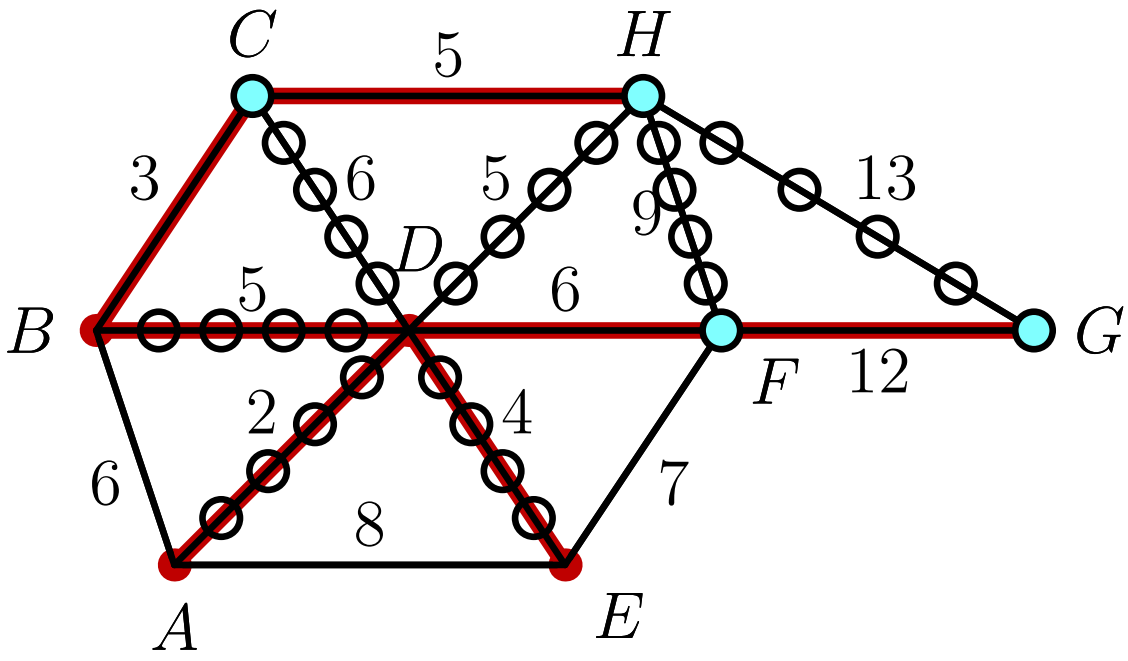
$$e_5 = CH, \quad e_6 = DF, \quad e_7 = FG.$$

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

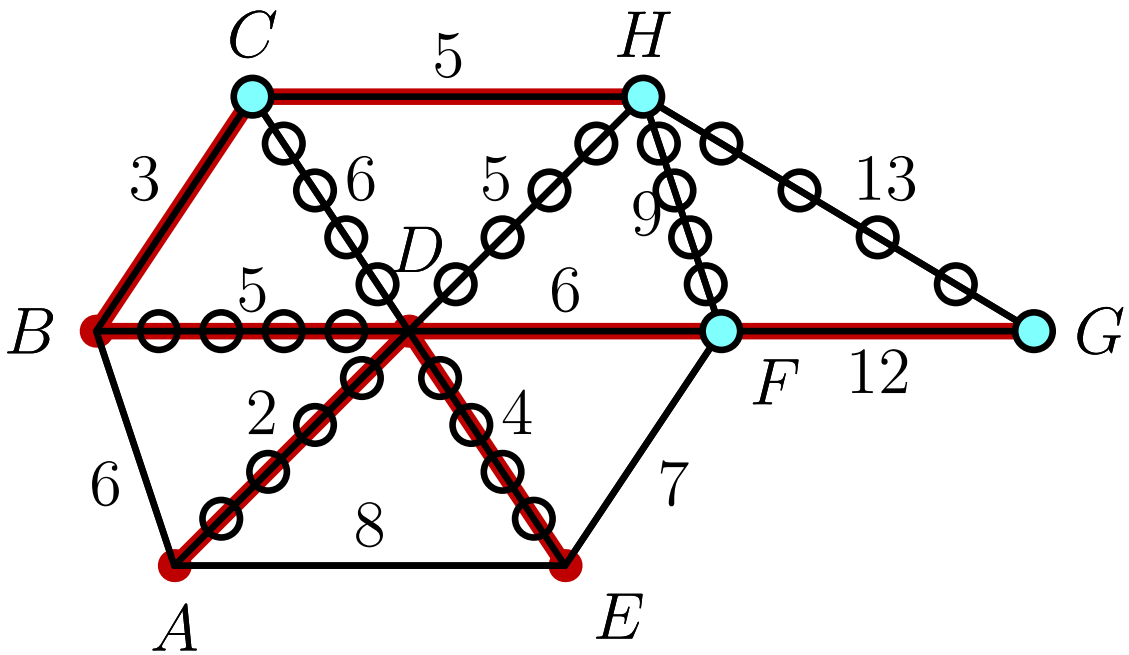
We now consider an example to show how successive modifications can convert an arbitrary spanning tree for the connected graph into the Prim spanning tree, whilst ensuring that the cost of each successive modified spanning tree does not exceed that of the spanning tree from which it is derived.

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:—



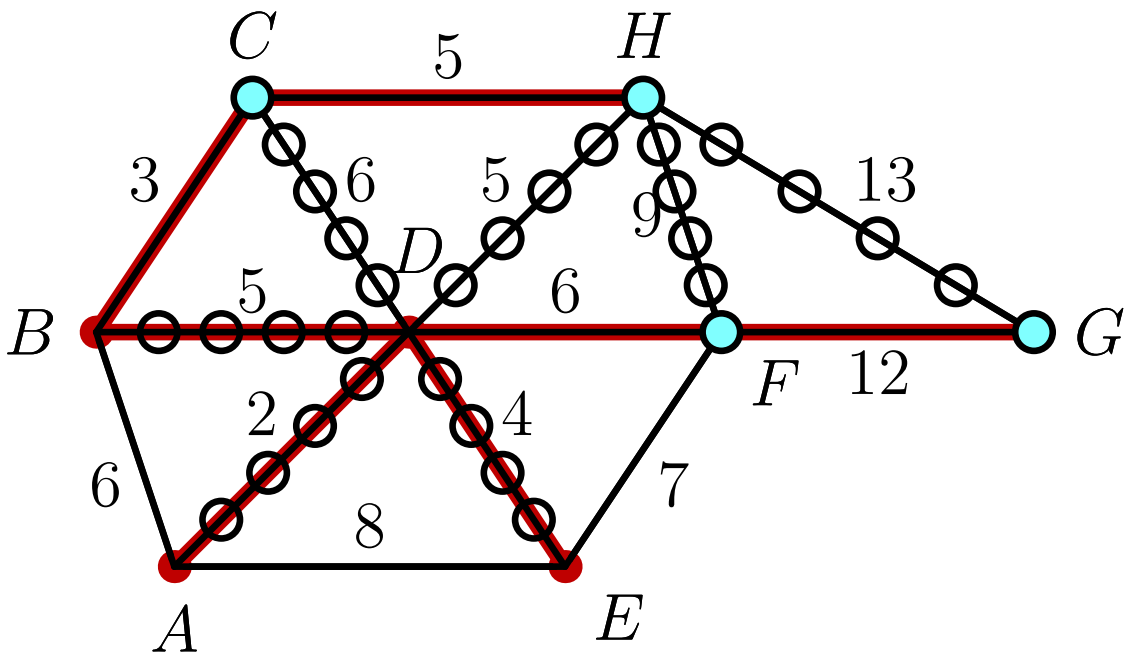
The first three edges added to the Prim spanning tree are also included in the spanning tree just specified. These edges are AD, DE and BD, which are the edges e_1, e_2 and e_3 respectively are the first three edges added to the Prim spanning tree. Now the edge BC is the fourth edge e_4 added to the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



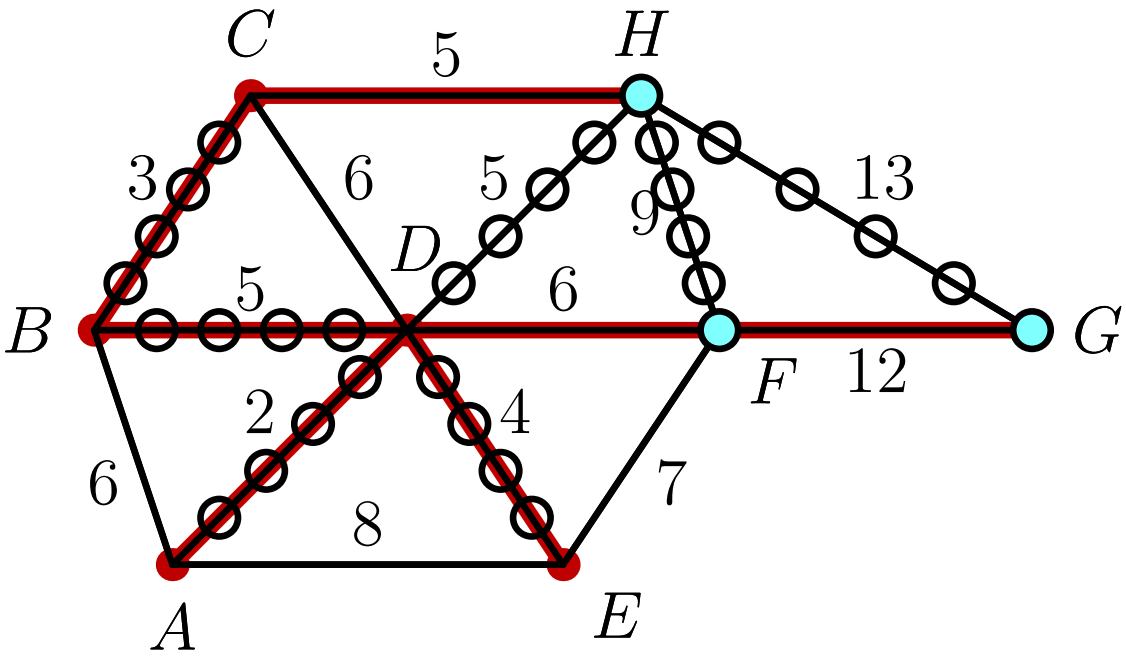
Let the vertices A, B, D, E incident on the edges AD, DE, BD be referred to as the *visited vertices* after the first three iterations of Prim's Algorithm (with the given ordering of edges). Let the remaining vertices C, F, G, H be referred to as the *unvisited vertices* after the first three iterations.

('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.

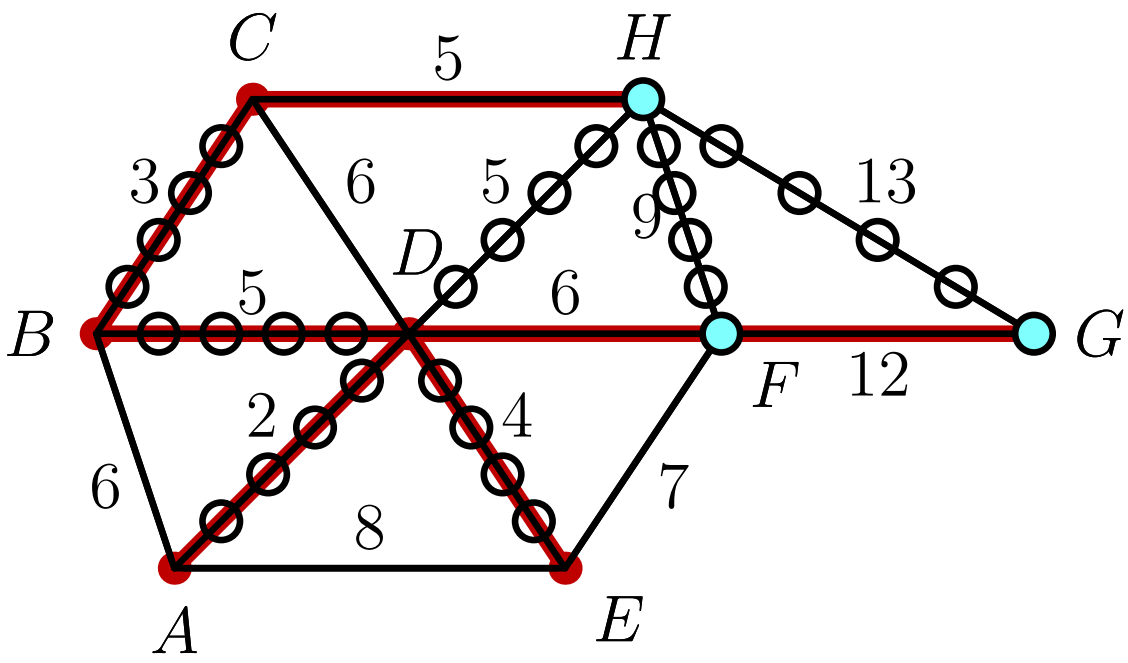


Had the cost of CD been less than that of BC then Prim's Algorithm would have added CD to the Prim spanning tree in place of BC. This did not happen. Therefore the cost of BC does not exceed that of CD, and indeed the costs of BC and CD are 3 and 6 respectively. We now modify the tree T, replacing CD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

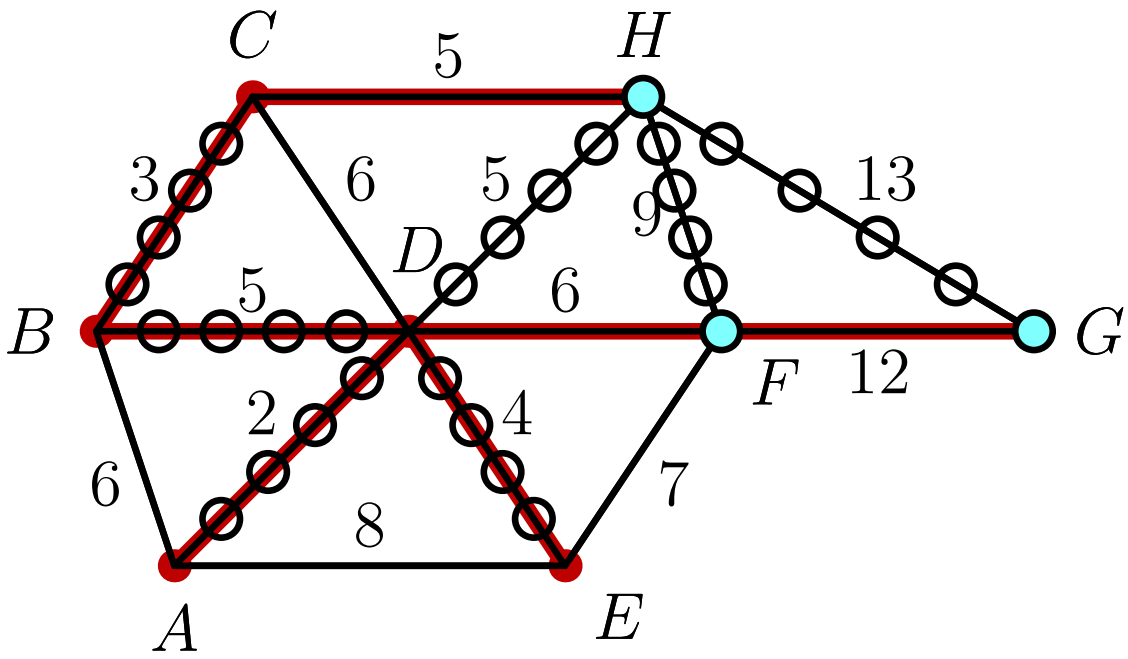


The modified spanning tree T' includes the edges e_1, e_2, e_3 and e_4 of the Prim spanning tree, and its cost does not exceed that of the tree T.

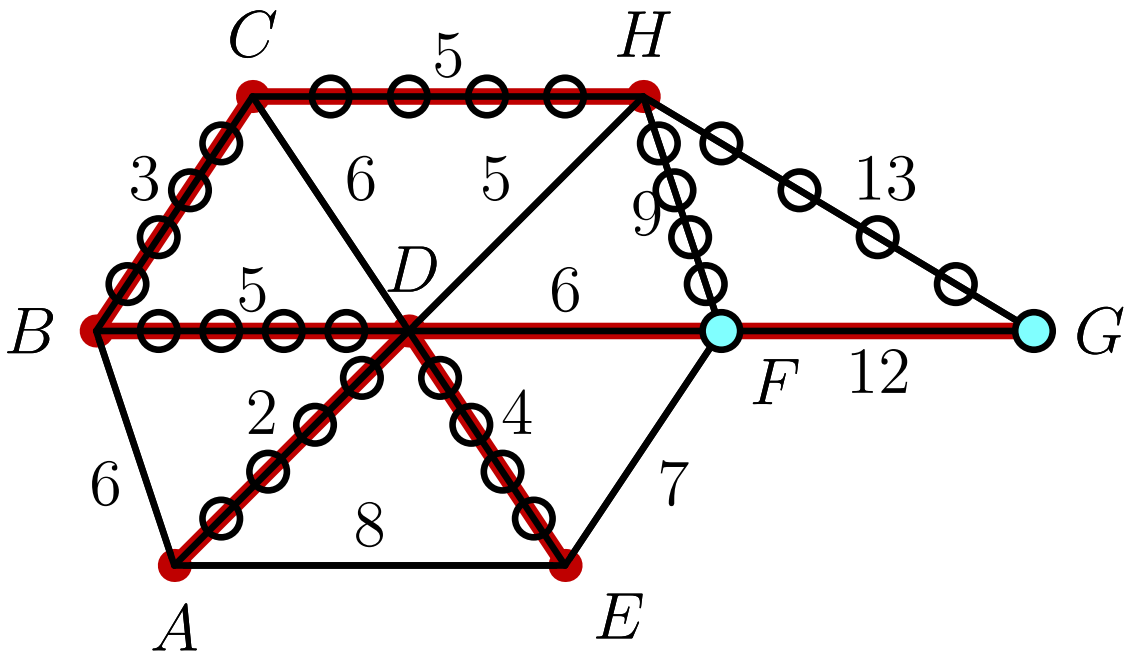
Now the fifth edge e_5 added to the Prim spanning tree is the edge CH. Addition of this edge to the tree T' creates a circuit $CHDBC$ in the resultant subgraph.



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.

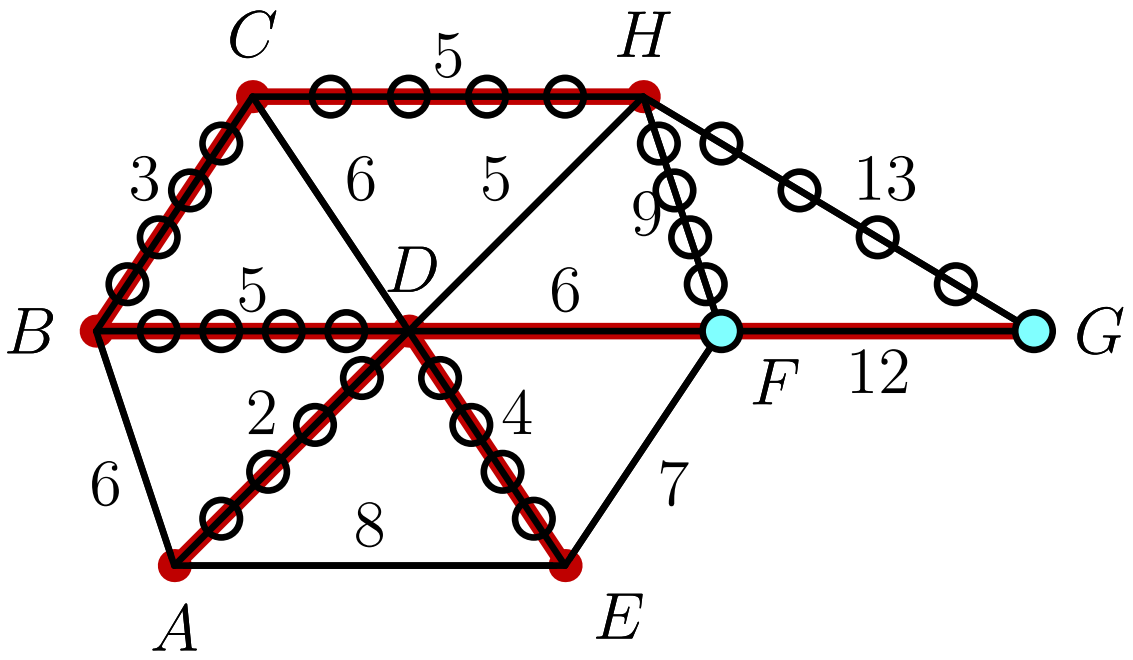


Had the cost of DH been less than that of CH then Prim's Algorithm would have added DH to the Prim spanning tree in place of CH. This did not happen. Therefore the cost of CH does not exceed that of DH, and indeed the costs of CH and DH are 5 and 5 respectively. We now modify the tree T', replacing DH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

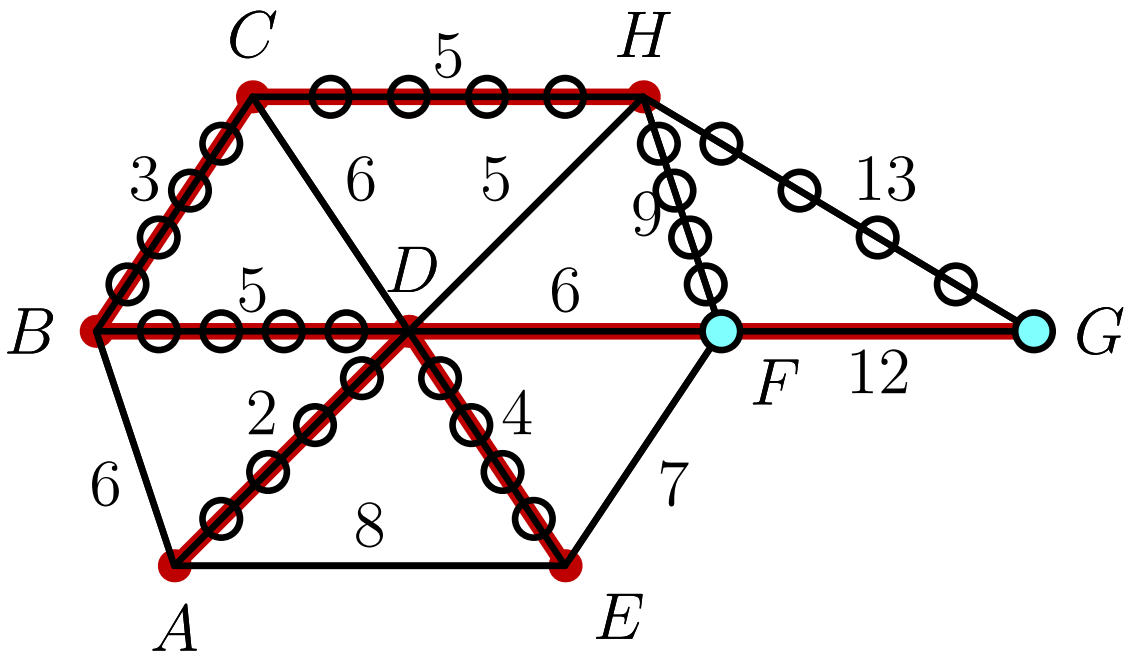


The modified spanning tree T'' includes the edges e_1, e_2, e_3, e_4 and e_5 of the Prim spanning tree, and its cost does not exceed that of the tree T'. It follows that the cost of T'' does not exceed that of T.

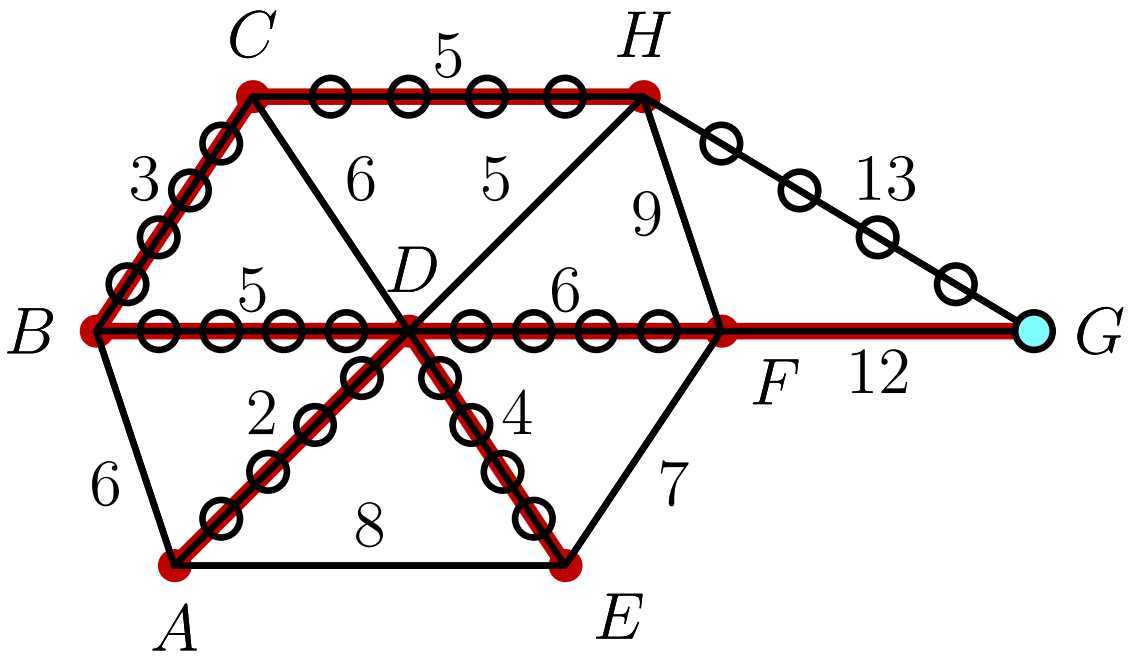
Now the sixth edge e_6 added to the Prim spanning tree is the edge DF. Addition of this edge to the tree T'' creates a circuit $DFHCBD$ in the resultant subgraph.



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.

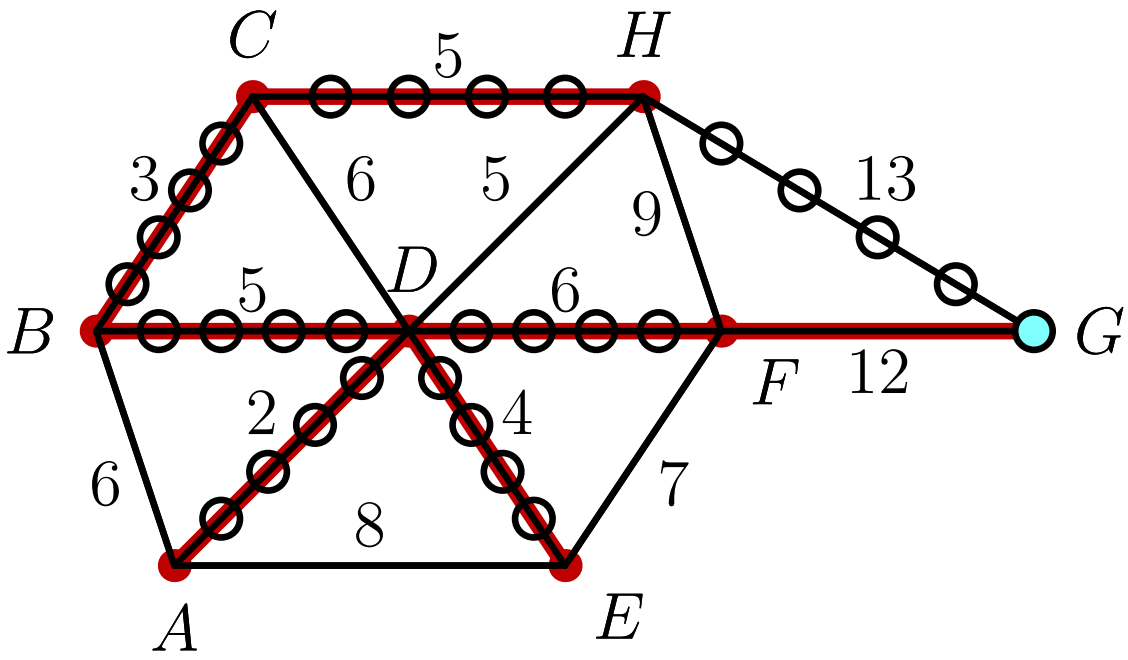


Had the cost of FH been less than that of DF then Prim's Algorithm would have added FH to the Prim spanning tree in place of DF. This did not happen. Therefore the cost of DF does not exceed that of FH, and indeed the costs of DF and FH are 6 and 9 respectively. We now modify the tree T'', replacing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

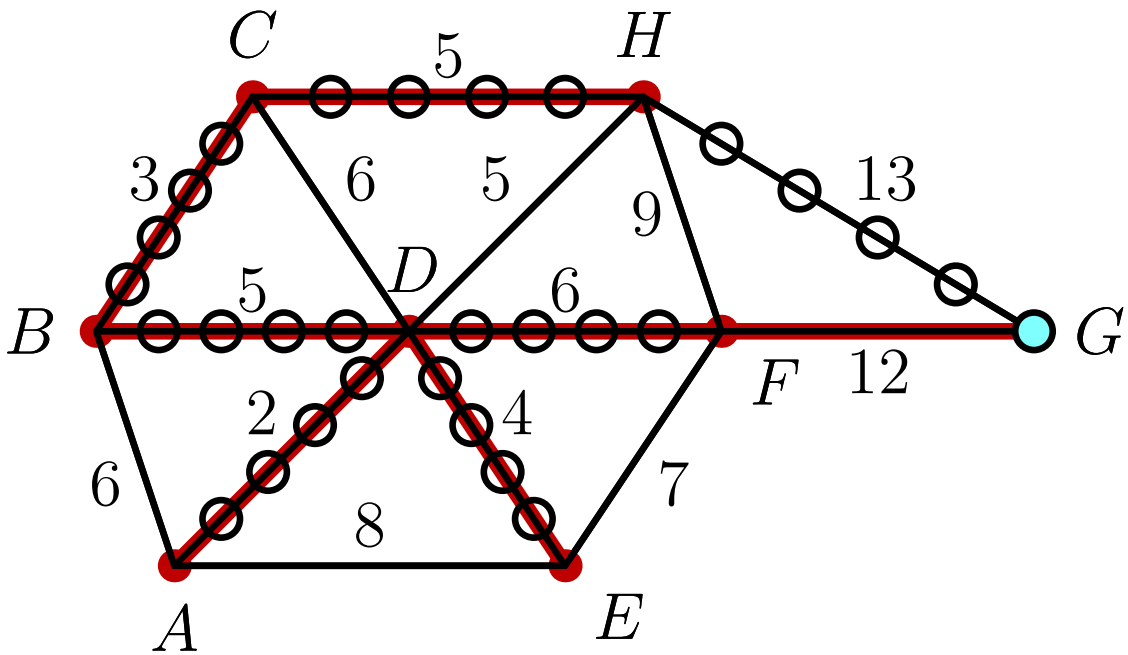


The modified spanning tree T''' includes the edges e_1, e_2, e_3, e_4, e_5 and e_6 of the Prim spanning tree, and its cost does not exceed that of the tree T''. It follows that the cost of T''' does not exceed that of T.

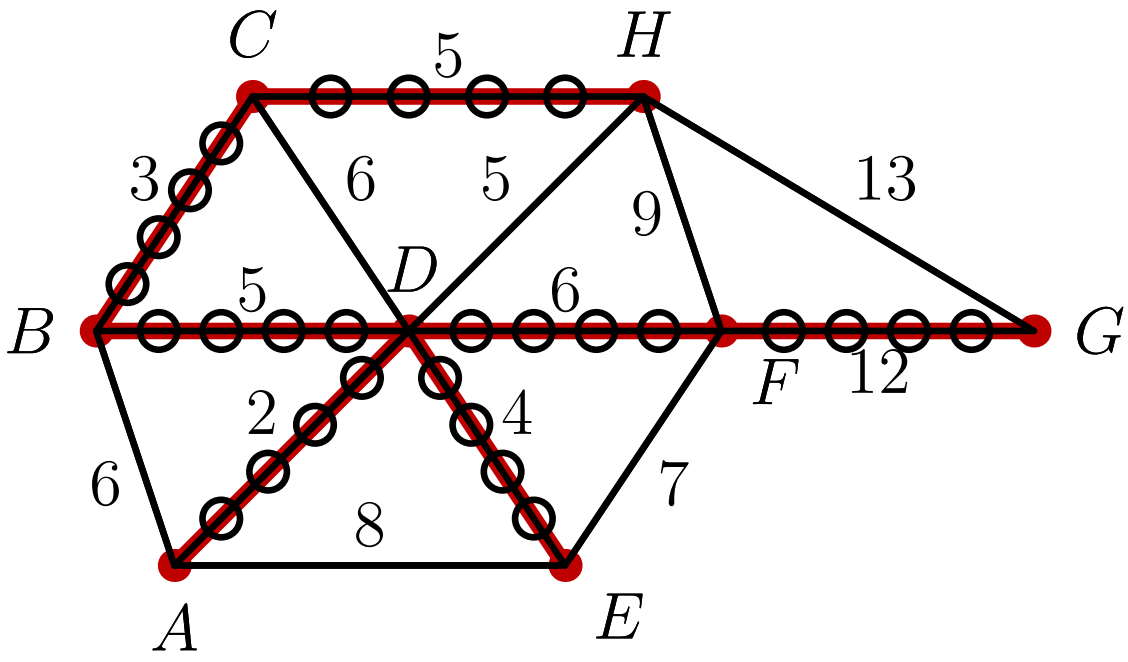
Now the seventh edge e_7 added to the Prim spanning tree is the edge FG. Addition of this edge to the tree T''' creates a circuit $FGHCBDF$ in the resultant subgraph.



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 33.1).

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 35.1

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. 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.