Universität Augsburg

A New Correctness Proof for Prim's Algorithm

P. Höfner, B. Möller

Report 2019-02 June 2019

Institut für Informatik
D-86135 Augsburg

A New Correctness Proof for Prim's Algorithm

Peter Höfner[1,2] and Bernhard Möller[3]

[1] Data61, CSIRO, Australia
[2] Computer Science and Engineering, University of New South Wales, Australia
[3] Institut für Informatik, Universität Augsburg, Germany

Abstract. We present a new correctness proof for Prim's algorithm. The standard proof establishes the invariant that each iteration constructs a subtree of some minimal spanning tree, and heavily relies on the existence of a spanning tree of the overall graph, as well as an 'edge exchange' property, which includes reasoning about graph cycles. We establish a stronger property showing that the algorithm builds a minimal spanning tree in each step, w.r.t. the vertices already covered. As a consequence, the proof neither uses the existence of a minimal spanning tree of the entire graph, nor the classical exchange property.

1 Introduction

Prim's algorithm is a greedy algorithm that calculates a minimal spanning tree of a given weighted undirected graph. It is one of the most prominent algorithms in computer science and usually part of the curriculum of undergraduate students. The algorithm was developed by Jarník [8], and later independently rediscovered by Prim [9] and Dijkstra [4]. The correctness of this greedy algorithm is usually proved by maintaining the invariant that each iteration constructs a subgraph of some minimal spanning tree (e.g. [3]). To the best of our knowledge all correctness proofs for Prim's algorithm are built on two facts:

(a) the existence of a minimal spanning tree of the overall graph, and the invariant stating that the constructed tree is a subtree of some minimal spanning tree;
(b) an 'edge exchange' law, stating that an edge in a minimal spanning tree can be replaced by an edge of the same weight if the result is again a spanning tree. This includes reasoning about graph cycles.

Figure 1 illustrates the situation. Although the algorithm – described in detail in Sect. 3 – has only constructed the subtree T_i of an overall spanning tree, the reasoning involves the entire tree, as depicted. Moreover, the existing proofs involve reasoning about cycles that involve the edges being exchanged – in the figure one may replace the bold edge by the dotted one.

In this short paper we develop a different proof. It shows that the algorithm in each step constructs a minimal spanning tree for the vertices already covered – a property which seems to be natural and intuitive, but we are not aware of such a proof in the literature. One reason for this may be that the property is not

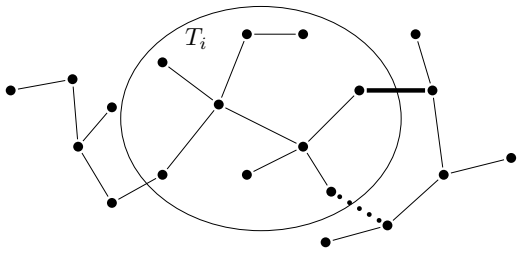


Fig. 1: 'Classical' reasoning within the correctness proof.

an invariant, but nevertheless suffices to prove correctness of Prim's algorithm. Our property and its proof have been developed to facilitate (semi-)automated verification of Prim's algorithm using an algebraic style of reasoning. In particular, we want to avoid reasoning about cycles and their transformations by edge exchange, which tends to be quite cumbersome. Although our property follows from the standard invariant, we did not want to use that path of reasoning: it would have meant that we would have to prove the standard invariant first – with exactly the difficulties we want to avoid. Rather we want a direct proof 'in one go', as simple as possible.

2 Prerequisites and Notation

In this section we recapitulate the basic terminology and recall the basic definitions needed for Prim's algorithm and our correctness proof. For its formulation we first define weighted undirected graphs.

Definition 2.1

1. An *undirected graph* is a pair $G = (V, E)$ comprising a set V of vertices and a set E of edges, which are 2-element subsets of V, i.e. self-loops are excluded. In this paper we assume V (and hence E) to be finite. To ease readability, we denote a single edge $\{u, v\} \in \mathcal{P}(V)$ by uv or vu.
2. For a vertex $v \in V$ of a graph $G = (V, E)$ we define the *degree* of v as $deg_E(v) =_{df} |\{e \in E : v \in e\}|$.
3. An *end vertex* of G is a vertex of degree 1, and a *thorn* is an edge $e \in E$ one of whose two vertices is an end vertex.
4. To add *weights* to edges we assume a function $\mathsf{w} : E \to W$, where W forms a cancellative, commutative monoid $(S, +, 0)$ of weights. Cancellative means $x + z = y + z \Rightarrow x = y$, for all $x, y, z \in S$. Classical examples for a such monoids are $\mathbb{N}_{\geq 0}$ and $\mathbb{R}_{\geq 0}$ under standard addition $+$.
5. Addition induces a preorder $\leq$ on weights by setting $x \leq y \Leftrightarrow_{df} \exists\ z : x + z = y$; we assume this preorder to be a linear order. In case $x \leq y$ for some $x, y \in W$, we denote the unique z satisfying $x + z = y$ by $y - x$. The definition implies that $+$ is cancellative also w.r.t. inequations, i.e.,

$$x + z \leq y + z \Rightarrow x \leq y\ . \tag{1}$$

6. A *weighted undirected graph* is a triple $G = (V, E, \mathsf{w})$, where (V, E) is an undirected graph and $\mathsf{w} : E \to W$ is a weight function into some set of weights. For a subset $E' \subseteq E$ we define the *weight of* E' as $\mathsf{w}(E') =_{df} \sum_{e \in E'} \mathsf{w}(e)$.

Based on this definition we now introduce further basic concepts for Prim's algorithm, such as subgraph, path and minimal spanning tree.

Definition 2.2 Assume a (weighted) graph $G = (V, E, \mathsf{w})$.

1. A *subgraph* of G is a graph $G' = (V', E', \mathsf{w}')$ with $V' \subseteq V$, $E' \subseteq E$ and $\mathsf{w}' = \mathsf{w}|_{E'}$. When G is understood, we denote G' by (V', E') and omit w'. Occasionally we will restrict a graph to a subset of vertices. This means, for a given set $V' \subseteq V$ of vertices we define the *restriction subgraph* $G|_{V'} =_{df} (V', \{e \in E \mid e \subseteq V'\})$ and the *subtraction subgraph* $G - V' =_{df} G|_{V \setminus V'}$.
2. A *path* in a graph $G = (V, E)$ is a repetition-free sequence $P = v_0, \ldots, v_n$ of vertices $v_i \in V$ such that $\forall\, i < n : v_i v_{i+1} \in E$. If $n > 0$ then $v_0 v_1$ is called the *starting edge* of P. For an edge $e \in E$ we write $e \in P$ if there is an $i < n$ such that $v_i v_{i+1}$ is a subsequence of P and $e = v_i v_{i+1}$. As in formal language theory, concatenation of paths is denoted by juxtaposition and a singleton path is identified with its only vertex. Hence, e.g., xP means the path starting with vertex x and continuing with the vertices of P.
3. A graph $G = (V, E)$ is *connected* if there is a path between each pair of vertices.
4. G is a *tree* if there is exactly one path between each pair of vertices.
5. As usual, a *subtree* T of G is a subgraph of G that is also a tree.
6. A *spanning tree (ST)* for $G = (V, E)$ is a tree $T = (V, F)$ for some subset $F \subseteq E$.
7. The weight of a graph $G = (V, E, \mathsf{w})$ is $\mathsf{w}(G) =_{df} \mathsf{w}(E)$.
8. A *minimal spanning tree (MST)* for a weighted graph $G = (V, E, \mathsf{w})$ is a spanning tree T of G with minimal weight $\mathsf{w}(T)$, i.e., for all spanning trees T' of G, $\mathsf{w}(T) \leq \mathsf{w}(T')$. For a subgraph (V', E') of G we will frequently abbreviate "MST for (V', E')" to "MST for V'" when E' does not matter.

3 The Algorithm of Jarník, Prim, and Dijkstra

As mentioned in the introduction Jarník, Prim, and Dijkstra independently developed the same algorithm for constructing a minimal spanning tree [8,9,4]. The input of this greedy algorithm is a connected, weighted undirected graph $G = (V, E, \mathsf{w})$. Informally, the algorithm first picks an arbitrary vertex $s \in V$ and defines the subtree $(\{s\}, \emptyset)$, which is a minimal spanning tree for $G|_{\{s\}}$. Until an MST for G is found, the algorithm chooses a weight-minimal edge among those edges $e \in E$ that connect the (already constructed) tree to vertices not yet in the tree. If all vertices in V have been connected then a minimal spanning tree has been constructed.

More formally the algorithm can be described as follows:[4]

input: $G = (V, E, \mathsf{w})$
choose $s \in V$;
$W := \{s\}$;
$F := \emptyset$;
while $|W| < |V|$ **do**
  choose a minimum weight edge uv such that $u \in W$ and $v \notin W$;
  $W := W \cup \{v\}$;
  $F := F \cup \{uv\}$;
od
return: (W, F)

To distinguish the constructed sets of vertices and edges, we denote the values of the variables W and F at the start of the kth iteration by W_k and F_k, respectively $(0 \leq k < |V|)$. The tree (W_k, F_k) is denoted by T_k. It is clear that all T_k are trees.

4 Auxiliary Properties

Before proving the correctness of Prim's algorithm we show two simple properties about MSTs, which we will use later on. These properties are well known in graph theory (e.g. [2]) and only included for completeness.

Lemma 4.1 For a graph $G = (V, E)$ let $T = (W, F)$ be an MST for $G|_W$ with some $W \subseteq V$ and v an end vertex of T. Then $T' =_{df} T - \{v\}$ is an MST for $W' =_{df} W \backslash \{v\}$.

Proof. Let e be the unique thorn of T that contains v. Consider an ST $S' = (W', F')$ for W' and set $S = (W, F' \cup \{e\})$. Since, by construction, $v \notin W'$ and F' is a tree, so is S, which means that S is an ST for W. Since T is an MST, we know

$$\mathsf{w}(T') + \mathsf{w}(e) = \mathsf{w}(T) \leq \mathsf{w}(S) = \mathsf{w}(S') + \mathsf{w}(e) \ ,$$

and cancellativity shows $\mathsf{w}(T') \leq \mathsf{w}(S')$. □

Note that minimality of the weight of the removed thorn e is not needed.

Corollary 4.2 If $T = (W, F)$ is an MST for W and $T' = (W', F')$ is a subtree of T then T' is an MST for W'.

Proof. T' results from T by successive removal of thorns. Hence the claim follows by repeated application of Lm. 4.1. □

5 Correctness of the Minimum Spanning Tree Algorithm

We show the following property of Prim's algorithm on a starting graph G:

Theorem 5.1 T_k is an MST for $G|_{W_k}$, where T_k and W_k are the sets of vertices and edges constructed in the kth iteration of the algorithm.

[4] It is not the aim of this paper to present an efficient implementation; of course one would introduce a counter for $|W|$, etc.

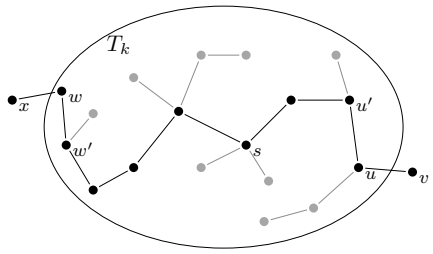


Fig. 2: Illustration of Lemma 5.3

Corollary 5.2 $T_{|V|-1}$ is an MST for V.

The proof of Thm. 5.1 is by induction on k. For the induction step we need the following lemma which looks at paths constructed by Prim's algorithm. To give an informal explanation, consider the path P from w to u in Fig. 2. The following lemma says if there is at least one outgoing edge on both sides of the path, here wx and uv, then the weight of the heavier of these edges is larger or equal to all weights in the path. This property is not obvious as (a) it compares weights to entire paths (something the algorithm does not consider), and (b) it does not hold for paths with only one outgoing edge. A more detailed discussion can be found in Sect. 6.

To formulate the lemma we denote for a tree $T = (V, E)$, with $u, v \in V$, the unique path from u to v within T by $P_T(u, v)$. Note that for $u \neq v$ we have $P_T(u, v) \neq P_T(v, u)$: in fact $P_T(v, u)$ is the reverse of $P_T(u, v)$. Still, for all $e \in E$ we have $e \in P_T(v, u)$ iff $e \in P_T(u, v)$. To save indices we write $P'(u, v)$ and $P_k(u, v)$ instead of $P_{T'}(u, v)$ and $P_{T_k}(u, v)$, etc.

Lemma 5.3 Let $T_k = (W_k, F_k)$ be a tree constructed by Prim's algorithm. Let uv and wx be two different edges leading out of T_k, i.e. $u, w \in W_k$ and $v, x \notin W_k$ and either $u \neq w$ or $v \neq x$. We further assume $\mathsf{w}(wx) \leq \mathsf{w}(uv)$. Then $\mathsf{w}(e) \leq \mathsf{w}(uv)$ for all edges $e \in P_k(w, u)$.

Proof. We use induction on T_k.
Base: The case $k = 0$ is trivial, since $T_0 = (\{s\}, \emptyset)$ is a tree containing only one path $P_0(s, s)$ without any edges.
Step: Suppose that Prim has constructed T_{k+1} by adding the edge h to T_k. Let the edges uv and wx be as in the claim for k+1. Moreover, let $P_{k+1}(w, u)$ be the unique path in T_{k+1} from w to u and let f, g be the first and last edges of $P_{k+1}(w, u)$. Then there are vertices w', u' with $f = ww'$, $g = u'u$ and $P_{k+1}(w, u) = wP_k(w', u')u$. Note that f and g coincide when $P_{k+1}(w, u)$ has only one edge.

We distinguish three cases.

1. Neither f nor g coincides with the edge h added by Prim's algorithm. Then the claim holds immediately by the induction hypothesis as $P_{k+1}(w, u) = P_k(w, u)$ and $v, x \notin W_{k+1}$ implies $v, x \notin W_k$.

2. $f = h$. This means $\mathsf{w}(f) \leq \mathsf{w}(uv)$. By the induction hypothesis we have $\mathsf{w}(e) \leq \mathsf{w}(uv)$ for all edges $e \in P_k(w', u)$ and therefore
$$\mathsf{w}(e) \leq \mathsf{w}(uv) \text{ for all edges } e \in wP_k(w', u) = P_{k+1}(w, u) \ . \tag{2}$$
If $\mathsf{w}(wx) \leq \mathsf{w}(uv)$ the lemma's claim is identical to (2); if $\mathsf{w}(uv) \leq \mathsf{w}(wx)$ that claim trivially follows from (2) and transitivity of $\leq$.
3. $g = h$. The reasoning is completely symmetric to that in the previous case.

□

We are now ready to prove the main property of the algorithm.

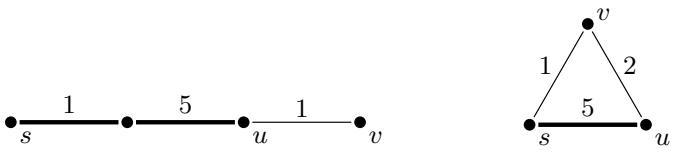
Proof (Thm. 5.1). We use induction on T_k.
Base: The induction base $k = 0$ is trivial, since $T_0 = (\{s\}, \emptyset)$ is a tree containing only one vertex (and no edges).
Step: For $k > 0$ we distinguish three cases. By the induction hypothesis, T_k is an MST for W_k. Consider an arbitrary MST $T' = (W_{k+1}, F')$ and the new edge uv chosen by Prim's algorithm. To show $\mathsf{w}(T_{k+1}) \leq \mathsf{w}(T')$ we distinguish three cases.

1. $uv \in F'$ is a thorn of T'.
By Lm. 4.1 $T' - \{v\}$ is an MST of W_k and hence $\mathsf{w}(T' - \{v\}) = \mathsf{w}(T_k)$. This implies
$$\mathsf{w}(T_{k+1}) = \mathsf{w}(T_k) + \mathsf{w}(uv) = \mathsf{w}(T' - \{v\}) + \mathsf{w}(uv) = \mathsf{w}(T') \ ,$$
and hence T_{k+1}, too, is an MST for W_{k+1}.
2. $uv \in F'$, and $deg_{F'}(v) = n > 1$.
This means that there is a vertex w with $u \neq w$, $u, w \in W_k$, $v \notin W_k$, and $uv, wv \in F'$. By the choice of uv in Prim's algorithm $\mathsf{w}(uv) \leq \mathsf{w}(wv)$.
Since T_k is an ST for W_k, it has the path $P =_{df} P_k(u, w)$, which by construction does not contain wv. In P there must be at least one edge e that is not in T'; otherwise T' would contain P and all its subpaths and thus, for every edge xy of P, have the two different paths $vP_k(u, x)$ and $vP_k(w, x)$ from v to x, contradicting treeness of T'.
Now pick an edge e in P but not in T'. Since $\mathsf{w}(uv) \leq \mathsf{w}(wv)$, we have, by Lemma 5.3 $\mathsf{w}(e) \leq \mathsf{w}(wv)$. Then $T'' =_{df} (W_{k+1}, F'')$ with $F'' =_{df} (F'\backslash\{wv\}) \cup \{pq\}$ is an ST of $G|_{W_{k+1}}$ with $\mathsf{w}(T'') \leq \mathsf{w}(T')$. Hence $\mathsf{w}(T'') = \mathsf{w}(T')$ and T'' is also an MST. Moreover, $deg_{F''}(v) = n - 1$.
Iterative application of this removal technique eventually leads to an MST $\hat{T} = (W_{k+1}, \hat{F})$ with $deg_{\hat{F}}(v) = 1$.
Since $\hat{T}$ satisfies Case 1, we obtain $\mathsf{w}(T_{k+1}) \leq \mathsf{w}(\hat{T})(= \mathsf{w}(T'))$.
3. $uv \notin F'$. Since T' is an MST for W_{k+1}, there must be a vertex w with an edge $wv \in F'$ such that $T'' = (W_{k+1}, (F'\backslash\{wv\}) \cup \{uv\})$ is an ST for W_{k+1}. By construction, wv is an edge leading out of W_k and hence, by the choice in Prim's algorithm, $\mathsf{w}(uv) \leq \mathsf{w}(wv)$. Therefore, T'' is an MST for W_{k+1}. Using T'' instead of T' we are back to Cases 1 or 2. □

This concludes the proof of Thm. 5.1.

6 Discussion

Although the Proof of Thm. 5.1 seems to be long, it is a simple case distinction, with Lm. 5.3 at its heart. In this section we have a closer look at the shape of this lemma as well as a crucial assumption for it.



(a) Lemma 5.3 fails (b) No invariant

Fig. 3: Illustrations concerning Lemma 5.3

First, it is crucial that the paths $P_k(u, w)$ under consideration by Lm. 5.3 have two outgoing edges. If there is only one, an analogous property does not hold as shown by the graph in Fig. 3(a). The path $P_2(s, u)$ is built in the second step of Prim's algorithm, when s is chosen as starting point. However, the weight of uv does not dominate $P_2(s, u)$.

The second interesting point is the proof method used for Lm. 5.3. It was quite non-trivial to realise that an induction over the construction of Prim graphs yields a simple proof. In fact, we first developed a proof by contradiction, which was far more intricate and lengthy. We even believe that our inductive proof is simpler than any proof found in the literature.

Last, it is important to know that the property proven by Thm. 5.1 is stronger than the classical invariant in the sense that it establishes that in every step of the algorithm a minimal spanning tree is created. However, in itself it is not an invariant in the classical sense, but an *always-true* property [10]. To illustrate this fact consider the graph in Fig. 3(b). The edge sv is an MST with regard to the set of vertices $\{s, v\}$. Applying one iteration of Prim's algorithm to this MST adds the edge sv. The graph $(\{s, u, v\}, \{su, sv\})$ is obviously not an MST to for the entire graph. The problem with an invariant approach is that it must cope with the extension of *arbitary* graphs, whereas the MST consisting of su would have never been constructed by Prim's algorithm. However, Thm. 5.1 can easily be turned into an invariant by adding a reachability premise, more precisely the requirement that the spanning tree must have been created by Prim's algorithm.[5] A different problem of always-true properties is that they are not preserved under parallel composition in a concurrent setting; this, however, is not a issue for Prim's algorithm.

[5] Such a transformation works for all always-true properties.

7 Conclusion and Outlook

In this paper we have presented a new correctness proof for Prim's algorithm. For a given graph $T = (G, E, \mathsf{w})$ it determines a minimal spanning tree. The proof is different from the existing ones in that it does not speak about some MST for the overall graph T, but only about the subtrees $T|_{W_k}$ that arise in the course of the algorithm's iteration.

Another difference to existing proofs is that we directly show an up to now implicit property (see Thm. 5.1), namely that all subtrees T_k are MSTs for their vertex sets. Although this may seem intuitively clear, we have pinned down that assertion by strict mathematical argument.

We hope that this property provides additional insight *why* the algorithm does what it is supposed to do. We further hope that similar arguments can be found for other greedy algorithms.

At first glance our proof may seem longer than proofs found in the literature. This is not because our proof is more complicated; the reason is that we wanted to present all details; e.g. Sect. 4 follows from standard graph theory and could be classified as 'folklore'. All proofs we found in the literature (e.g. [3]) would become equally long when all details were added.

Our proof follows 'classical' verification techniques for while-programs in that the correctness of the algorithm immediately follows from the main property in combination with the termination condition $|W| \geq |V|$ of the while loop (Cor. 5.2).

One aim with deriving this new proof was to pave the way for easier (semi-) automated verification, notably using purely algebraic techniques (e.g. [1,5,6,7]). The particular advantages of algebraic proofs are that they allow purely (in)equational reasoning (which is less error-prone than using full first or second order logic) and are easy to automate. Since our treatment avoids mentioning cycles, no cumbersome algebraic formalisation and analysis of these is necessary. The details will be the topic of a subsequent paper.

Acknowledgment We are grateful to Roland Glück and Rob van Glabbeek for helpful remarks and suggestions.

References

1. Berghammer, R., von Karger, B., Wolf, A.: Relation-algebraic derivation of spanning tree algorithms. In: Jeuring, J. (ed.) Mathematics of Program Construction (MPC'98). Lecture Notes in Computer Science, vol. 1422, pp. 23–43. Springer (1998)
2. Chartrand, G., Zhang, P.: A First Course in Graph Theory. Dover (2012)
3. Cormen, T.H., Leiserson, C., Rivest, R.L., Stein, C.: Introduction to Algorithms, 3rd Edition. MIT Press (2009), `http://mitpress.mit.edu/books/introduction-algorithms`
4. Dijkstra, E.W.: A note on two problems in connexion with graphs. Numerische Mathematik 1(1), 269–271 (1959)

5. Guttmann, W.: Relation-algebraic verification of prim's minimum spanning tree algorithm. In: Sampaio, A., Wang, F. (eds.) Theoretical Aspects of Computing (ICTAC'16). Lecture Notes in Computer Science, vol. 9965, pp. 51–68 (2016)
6. Guttmann, W.: An algebraic framework for minimum spanning tree problems. Theoretical Computer Science (2018), (in press)
7. Höfner, P., Möller, B.: Dijkstra, Floyd and Warshall meet Kleene. Formal Aspects of Computing 24(4-6), 459–476 (2012)
8. Jarník, V.: O jistém problému minimálním. (Z dopisu panu O. Borůvkovi) [On a certain problem of minimization]. Práce moravské přírodovědecké společnosti 6(4), 57–63 (1930), http://hdl.handle.net/10338.dmlcz/500726
9. Prim, R.C.: Shortest connection networks and some generalizations. Bell System Technical Journal 36(6), 1389–1401 (1957)
10. Van Gasteren, A., Tel, G.: Comments on "on the proof of a distributed algorithm": Always-true is not an invariant. Information Processing Letters 35(6), 277–279 (1990)