

Dartmouth College

Dartmouth Digital Commons

Computer Science Technical Reports

Computer Science

3-31-1992

A Visualization System for Correctness Proofs of Graph Algorithms

Peter A. Gloor

Massachusetts Institute of Technology

Donald B. Johnson

Dartmouth College

Fillia Makedon

Dartmouth College

Panagiotis Metaxas

Dartmouth College

Follow this and additional works at: https://digitalcommons.dartmouth.edu/cs_tr



Part of the [Computer Sciences Commons](#)

Dartmouth Digital Commons Citation

Gloor, Peter A.; Johnson, Donald B.; Makedon, Fillia; and Metaxas, Panagiotis, "A Visualization System for Correctness Proofs of Graph Algorithms" (1992). Computer Science Technical Report PCS-TR92-180. https://digitalcommons.dartmouth.edu/cs_tr/68

This Technical Report is brought to you for free and open access by the Computer Science at Dartmouth Digital Commons. It has been accepted for inclusion in Computer Science Technical Reports by an authorized administrator of Dartmouth Digital Commons. For more information, please contact dartmouthdigitalcommons@groups.dartmouth.edu.

**A VISUALIZATION SYSTEM FOR CORRECTNESS
PROOFS OF GRAPH ALGORITHMS**

**P.A. GLOOR
D.B. JOHNSON
F. MAKEDON
P. METAXAS**

Technical Report PCS-TR92-180

A Visualization System for Correctness Proofs of Graph Algorithms

P.A. Gloor¹, D.B. Johnson², F. Makedon², P. Metaxas²

March 31, 1992

Abstract

In this paper we describe a system for visualizing correctness proofs of graph algorithms. The system has been demonstrated for a greedy algorithm, Prim's algorithm for finding a minimum spanning tree of an undirected, weighted graph. We believe that our system is particularly appropriate for greedy algorithms, though much of what we discuss can guide visualization of proofs in other contexts. While an example is not a proof, our system provides concrete examples to illustrate the operation of the algorithm. These examples can be referred to by the user interactively and alternatively with the visualization of the proof where the general case is portrayed abstractly.

¹MIT Laboratory for Computer Science. email: gloor@lcs.mit.edu

²Mathematics and Computer Science, Dartmouth College. email:
{djohnson,makedon,takis}@dartmouth.edu

1. INTRODUCTION

We address the problem of visualizing proofs of graph algorithms. In particular we look at correctness proofs of certain greedy graph algorithms. To the best of our knowledge, no previous work on visualizing proofs appears in the literature, despite the considerable body of research in algorithm and scientific visualization, and the significance of understanding an algorithm's proof of correctness.

Visualizing the proof of an algorithm is different from visualizing the algorithm itself, as is done by various systems [Bae69], [Bro85], [Bro88], [Dui86], [Hel89], [Sta90]. An algorithm visualization system is composed of an integrated set of multimedia tools (such as graphics, animation, text, video, code, etc.) which simulate how the algorithm works using abstractions of the data, operations, and semantics of the program behind the algorithm. Algorithm visualization may involve program views that have no direct correspondence to the program's data or execution. Visualization of a proof is an algorithm visualization system which goes one step further: it can be composed of an algorithm visualization system plus a proof-dialog which extracts in a temporal fashion and with visual means the correctness of the algorithm. The main problem of all these systems is how to simplify the production of the algorithm visualization. In this paper we provide an easy to produce framework that defines three primary components for such a system: the definition of the abstract operations necessary for a proof of correctness visualization, the design of operations for the proof process or dialog and the correspondence between the proof visualization and the algorithm visualization.

The correctness proofs we consider are at the level that such proofs are customarily done in a classroom and thus should not be confused with *program verification*. The latter usually refers to a formal proof that examines the actual code implementing some algorithm in a step-like fashion and shows that the program actually achieves the goal. Our endeavor is to illustrate *why* the algorithm works; an interactive system that deals with the algorithm at a high level and with the fundamental ideas underlying the algorithm.

There is yet another class of proofs that we will not address in this paper, though our work may have application to this class. This class includes the complexity proofs, i.e. proofs of the worst-case running time claimed by some algorithm.

The lack of previous work in this area is partly due to the fact that proofs often involve hypothetical and abstract objects created for the purpose of the proof. Such objects are difficult to visualize. We make use of the fact that there is a question-answer process that goes on when one reads or is told a proof. We call this process the *proof-dialog*. During the proof-dialog, the reader/listener seeks answers from the text or the speaker on several questions in order to become persuaded of the proof's correctness. These questions mainly attempt to find counter-examples in the arguments presented. It is this proof process that we try to simulate with visualization.

The proof-dialog operates on two levels, those of concrete and abstract objects. Our database consists, correspondingly, of two kinds of objects. Concrete graph objects such as vertices, edges, adjacency matrices and linked lists that represent the primitives and the data structures implementing them, and abstract objects that, for example, are hypothetical or of indefinite size or structure. Concrete objects are shown explicitly. Abstract objects must of necessity have a symbolic representation.

2. SYSTEM OVERVIEW

We describe a system capable of visualizing correctness proofs for graph algorithms. In this section we present the components of such a system and in section 4 we go through a detailed example that will clarify our description.

We have done our prototype implementation in HyperCard on a Macintosh. The proof visualization system is composed of:

1. abstract graph objects, for example graphs, subgraphs and paths that are hypothetical, of indefinite size or structure.
2. concrete graph objects, such as vertices and edges of a given graph.
3. descriptions of theorems to be proven and invariants examined for validity during the proof.
4. dynamic help mode based on hypertext that provides updatable information about the objects appearing on the screen.
5. the proof-dialog description.

In the description below we will distinguish between the *user* and the *programmer*. The first term refers to the person that uses the system in order to get a better understanding of the proof, while the second term refers to the programmer who will use the system to compose a new proof visualization.

Though our system could be stand-alone, we suggest that it be used in conjunction with the following parts: (a) a proof orientation (introduction) describing the problem and the algorithm through pseudocode that explains the important steps, and (b) an integrated animation of the algorithm. The term "integrated" refers to an animation that operates on the same pseudocode and uses the same concrete objects as the proof system. Users are urged to examine and understand the animation first, so they have some intuition of *how* the algorithm works. As we mentioned, the aim of the proof system is to show *why* the proof works.

A typical session is composed of the following steps: In the beginning the user goes through the introduction and the interactive visualization of the algorithm. When the users are familiar with the algorithm and the *pseudocode* that implements the algorithm, they are told the *theorem* along with a set of *invariants* that should be satisfied during the run of the algorithms. Special attention must be paid in choosing these three components, in order to satisfy the following objective: When presented to the user it must be clear that, if the invariants are satisfied during the run of the algorithm, then the theorem is true.

The user is then guided through the proof step-by-step. The proof visualization is controlled by the proof-dialog in the following form: A sequence of statements is presented to the user and the user can either accept them or ask for more explanation of any of these statements. In the latter case, the system gives further explanation. Moreover, users can at any time jump back into the algorithm animation and examine more examples to improve the understanding of the algorithm, before resuming the proof session. The session ends when a complete set of statements has been presented and has been understood by the user. Of course, users can go through the proof-dialog as many times as they wish. They also have the option of stepping backwards to take a closer look at some concepts presented earlier.

The programmer who uses the system has to provide the following things:

1. The introduction part, in which the definition of the problem, its significance and other similar information appears.
2. An animation of the algorithm that solves the problem. (As our prototype system has been implemented in HyperCard, the animation ideally should be implemented in HyperCard, too. But with enough technical knowledge it is possible to integrate animation in any format into our system using digitized video.)
3. The proof-dialog in terms of a text file having a special format.
4. The animation of the basis and inductive proof steps.
6. A help file that contains information about the objects.

3. EDUCATIONAL ASPECTS

To present complex concepts, we are using a three-step process called *master teaching* that has already been applied successfully in other educational multimedia projects (Dartmouth Interactive Media Lab). The three steps are defined as follows:

1. Master Teaching:
In the first step a teacher who is a master in teaching as well as a master in the methods he teaches, presents the subject to the students. Ideally, this is done interactively by the human teacher. Unfortunately, master teachers are a scarce resource, and it is desirable to replace or augment the master teacher. In the simplest case this is done by a text provided by the master teacher, but with today's multimedia capabilities this can be done much more effectively including video segments showing the master teacher teaching.
2. Experiential Learning:
The second step includes practicing the concepts taught by the master teacher. "Learning by doing" is still one of the most effective ways to understand new ideas. The learning effect can be improved if the new ideas are presented, and can be practiced, in different ways.
3. Reflective Learning:
The third step provides for an in depth understanding of the taught and practiced concepts. By combining his own experiences with the methods taught by the master teacher, the student builds his own mental model of the concepts.

This three-step-learning process can easily be adapted for the multimedia enhanced teaching of computer science algorithms. It is particularly well suited for teaching proofs of theorems. Fig. 1 contains the overview map of an animated learning environment for Prim's algorithm for finding a Minimum Cost Spanning Tree (MST) [Pri57].

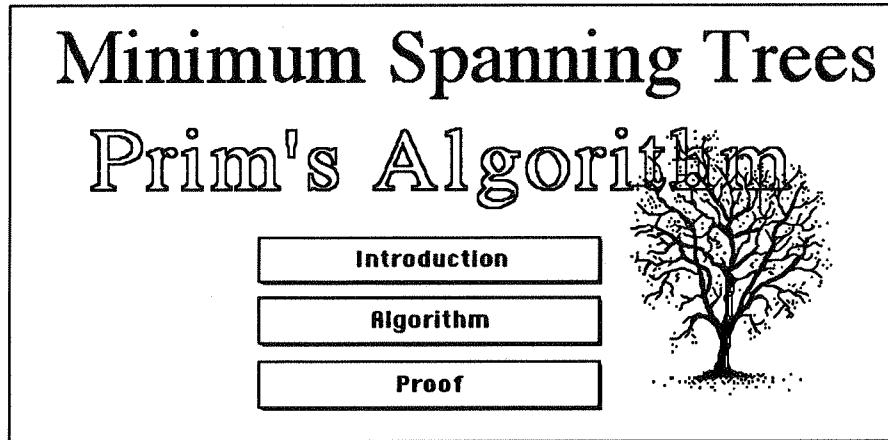


fig. 1 overview map of Prim's algorithm/theorem

Fig. 1 shows the three main parts of our learning environment. These correspond directly to the three steps of master teaching. The "introduction" part explains the basic concepts as they would be explained by the master teacher. The "algorithm" part contains an interactive animation environment corresponding to the "experiential learning" chapter in the master teaching method. Fig. 2 shows a particular instance of the animation of Prim's algorithm.

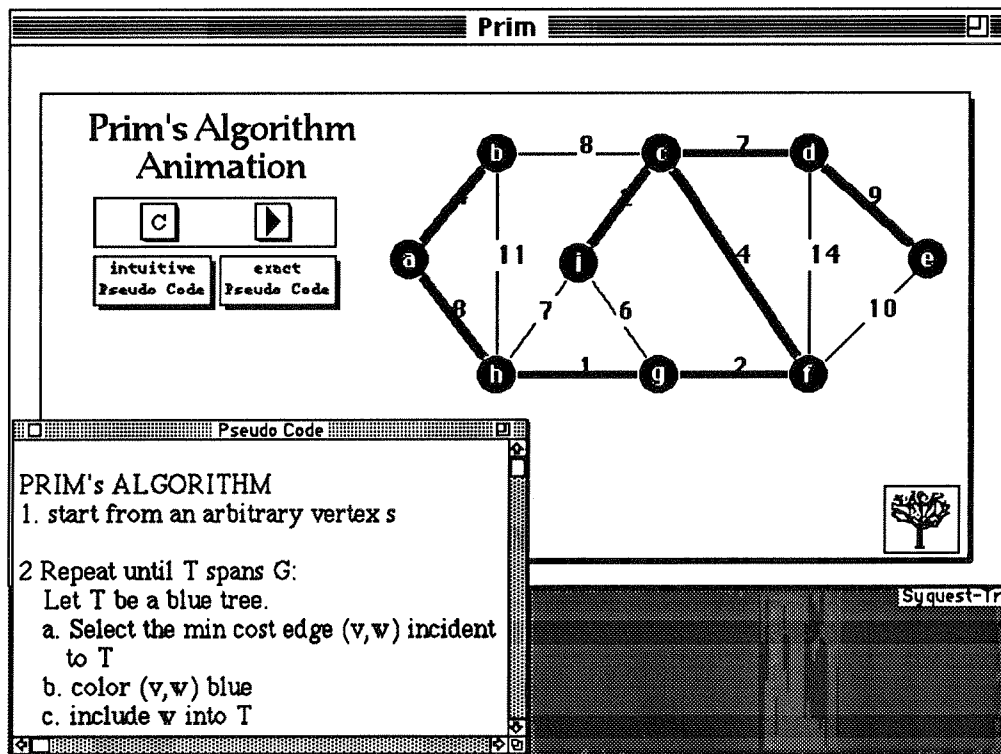


fig. 2 Animation of Prim's Algorithm

The interactive environment in Fig. 2 allows the student to experiment with Prim's algorithm, while the animation is running synchronously with different pseudo code representations. This visual-with-code representation of the algorithm allows an intuitive and, ideally, thorough understanding of the algorithm which, in turn, leads to a better understanding of how the proof works. The presentation of the proof corresponds to the third "reflective learning" step in the master teaching method. In this step the student is expected to understand the proof based on an understanding of the algorithm gained in the previous two steps.

4. CORRECTNESS PROOF FOR PRIM'S MINIMUM SPANNING TREE ALGORITHM.

In this section we go over an example that will better illustrate our approach. The algorithm we want to visualize is the well-known Prim's MST algorithm [Pri57] (also known as the *snowball* algorithm). This algorithm computes, for any given weighted undirected graph G , a spanning tree whose sum of weighted edges is minimum. There are several ways to go about proving this algorithm; we choose one that we find more appropriate for visualization. We adopt an approach similar to the one described by Tarjan [Tar83].

Let us define a blue tree $T=(V_T, E_T)$ on V_T nodes and E_T edges to be an MST on V_T . The objective of the algorithm is to divide up the set of edges E of the graph into two sets; a set of *blue edges* that are contained in an MST, and a set of *red edges* that are not part of any MST. A *cut* is a set of edges that disconnects the given graph G .

Prim's algorithm can be viewed as computing an MST by applying the following rule repeatedly:

Blue rule: *Select a cut where each edge has exactly one endpoint in the blue tree. Among the edges of the cut, select one with minimum cost and color it blue.*

Now, the algorithm can be stated as follows:

Prim's algorithm: Given a connected graph $G=(V,E)$ on $|V|=n$ vertices and $|E|=m$ edges, find a MST $T' = (V, E_{T'})$ by starting from some arbitrary vertex s and repeating the following inductive step $n-1$ times:

Let $T=(V_T, E_T)$ be a blue tree containing s . Select the minimum-cost uncolored edge (v,w) incident to T , and color it blue. Augment T by edge (v,w) .

In light of the previous discussion, the appropriate invariant is the following:

Invariant: In the beginning of each step, (V_T, E_T) is a minimum spanning tree on the subgraph of the given graph G induced by V_T .

It is now straightforward to see that if we show that the invariant holds in the beginning of every step of the algorithm then we have proven the following:

Theorem: Prim's algorithms correctly finds a MST T' of G .

The proof-dialog proceeds by presenting a sequence of statements that will eventually lead to a complete proof of the theorem.

The user can challenge the validity of these statements at any point and ask for additional explanation.

The proof-dialog proceeds as follows. After presenting the theorem, the step invariant, and the pseudocode, the system presents the first statement:

Statement 1. Examine the theorem and the invariant. If we show that the invariant is satisfied at every step, then the theorem is true.

At this point, the program prompts for the user's opinion on this statement. The user can either agree on the validity of this statement by clicking on the "next proof step" button appearing on the screen or ask for more explanation by clicking on the "explain this step" button. In the second case the dialog provides the following explanation:

If the invariant is satisfied on every step, then it is satisfied on the last step as well, i.e. in the step that T' spans the vertices of G . Since T' is a blue tree, i.e. a MST of the whole graph, then the theorem holds true.

When the user agrees on the validity of the statement, the proof-dialog presents the second statement:

Statement 2. We will prove the theorem by induction on the number of steps required. First, the invariant holds when $T = \{s\}$.

The sequence of statements and explanations has been specified by the programmer in the proof-dialog document. In our example the proof-dialog document is as follows:

1. Examine the theorem and the invariant. (If we show that the invariant is satisfied in every step of the algorithm, then the theorem is true)
 - 1.1 If the invariant is satisfied in every step of the algorithm, then it is satisfied in the last step as well. Since T' is a blue tree, i.e., an MST of the whole graph, the theorem holds true.
2. We will prove the theorem by induction on the number of steps required.
3. The invariant holds true for $T=\{s\}$.
 - 3.1 The MST of a graph containing one vertex s and no edges is just that vertex.
4. Assumption hypothesis:
assume the invariant holds at the k -th step and let's assume that the invariant is violated for the first time in the $(k+1)$ st step with the incorporation of edge (v,w) into the blue tree.
5. There are two possible cases:
 - 5a. There exists a MST T' on $V_T \cup \{w\}$ with exactly one edge (x,w) incident on w .
 - 5b. Every MST on $V_T \cup \{w\}$ has two (or more) edges incident on w .
6. (explanation of 5a) By the invariant at step k , $\text{cost}(T' - \{w\}) = \text{cost}(T)$, and by the algorithm, $\text{cost}(v,w) \leq \text{cost}(x,w)$, since x is in V_T . This implies $\text{cost}(T \cup (v,w)) \leq \text{cost}(T')$, a contradiction.

7. (explanation of 5b) Let the two edges incident on w be (x,w) and (y,w) , where x and y are in V_T . Now consider the subgraph $H = T \cup (x,w) \cup (y,w)$. H will have exactly one cycle C , and C will contain edges (x,w) and (y,w) .

8. (explanation of 5b) If no edge in C is more costly than one of (x,w) and (y,w) , then one of (x,w) and (y,w) can be deleted from H to give an MST with only one edge incident on w . But, by assumption of case 5b, this is impossible.

9. (explanation of 5b) So, without loss of generality, assume that x was admitted to the blue tree before y , and let (u,y) be in C . Since both (x,w) and (y,w) are less costly than (u,y) , edges (x,w) and (y,w) would have to have been brought into the blue tree before edge (u,y) was a least costly cut edge. But this did not happen, so the assumption of 5b is false.

10. Since neither case 5a nor 5b can happen, $T \cup \{v,w\}$ is a MST.

Basically, our proof visualization system demands two input sets:

- an animation of the graph:
This animation can be used as a repository for the concrete and abstract proof objects. In addition the animation also contains an educationally suitable pseudo code description of the algorithm that is employed in the theorem.
- a text file containing the basis for the proof-dialog.
This text file contains:
 - theorem
 - set of invariants
 - set of expandable statements

Obviously the set of expandable statements represents the core of our animation system. That means, that our system still is based on a textual description of the text. But the usefulness of the text alone gets vastly expanded by adding hypertext links on a micro level [Glo91]. The hypertext links allow direct access to any points where the teacher thinks that a student might have difficulties in understanding, but they do not interrupt the flow of the proof because the student is not forced to follow the links.

For a sequence of screen dumps showing how the proof visualization works, see the Appendix.

5. CONCLUSIONS

We have presented a system for visualization of correctness proofs for greedy graph algorithms. There is considerable research activity in the field of algorithm animation, but so far nobody has ever tried to combine hypertext concepts with algorithm animation to visualize a correctness proof of an algorithm. The following points resume our experience:

- The method for animating proofs is basically just a guided walkthrough of slides to be shown one after the other.
- Emphasis is on the hypertext aspect of the system because access to additional information can be given at the location where it is actually needed.

- The animations for the proof are extremely simple, based on an animation of the algorithm to be presented before the proof.
- Color is very useful for emphasizing the actual/important point of a slide.

Obviously the above listed points can be used to visualize any proof, not only induction based correctness proofs of greedy algorithms. Graph algorithms are particularly well suited for animation, therefore the integration of the algorithm animation into the visualization of the correctness proof was relatively straightforward. This is the first time such a system is introduced, but we hope that it will stimulate research in the area and influence the generation of more systems for proof visualization.

REFERENCES

- [Bae69] R. M. Baecker. Picture driven animation. In *Spring-Joint Computer Conference*, vol. 34, pages 273-288. AFIPS Press, 1969.
- [Bro85] M. H. Brown and R. Sedgewick. Techniques for algorithm animation. *IEEE Software*, 2(1):28-39, January 1985.
- [Bro88] M. H. Brown. Exploring Algorithms using BALSAs-II. *IEEE Computer*, 21(5):14-36, May 1988.
- [Dui86] R. A. Duisberg. Animated graphical interfaces using temporal constraints. In proceedings of the *ACM SIGCHI '86 Conf. on Human Factors in Computing Systems*, pages 131-136, Boston, Ma, April 1986.
- [Glo91] P. A. Gloor. Presenting Hypermedia Concepts using Hypermedia Techniques. In *Hypertext/Hypermedia 91*, Graz, Österreich, May 27-28, 1991, Springer Informatikfachberichte 276, Heidelberg, 1991.
- [Hel89] E. Helttula, A. Hyrskykari and K. Raiha. Graphical specification of algorithm animations with Alladin. In Proc. of the *22nd Hawaii Intern. Conf. on System Sciences*, pp. 892-901, HI, Jan. 1989.
- [Pri57] R.C. Prim. Shortest connection networks and some generalizations. *Tech. Journal*, Bell Labs, 36:1389-1401, 1957.
- [Sta90] J. T. Stasko. Simplifying Algorithm Animation with TANGO. In *1990 IEEE Workshop on Visual Languages*, October, 1990, Skokie, IL.
- [Tar83] R.E. Tarjan. *Data structures and network algorithms*. Soc. for Industrial and Applied Math., Philadelphia, PA 19103, 1983.

APPENDIX

The following screen shots describe on Prim's algorithm how our proof visualization system works.

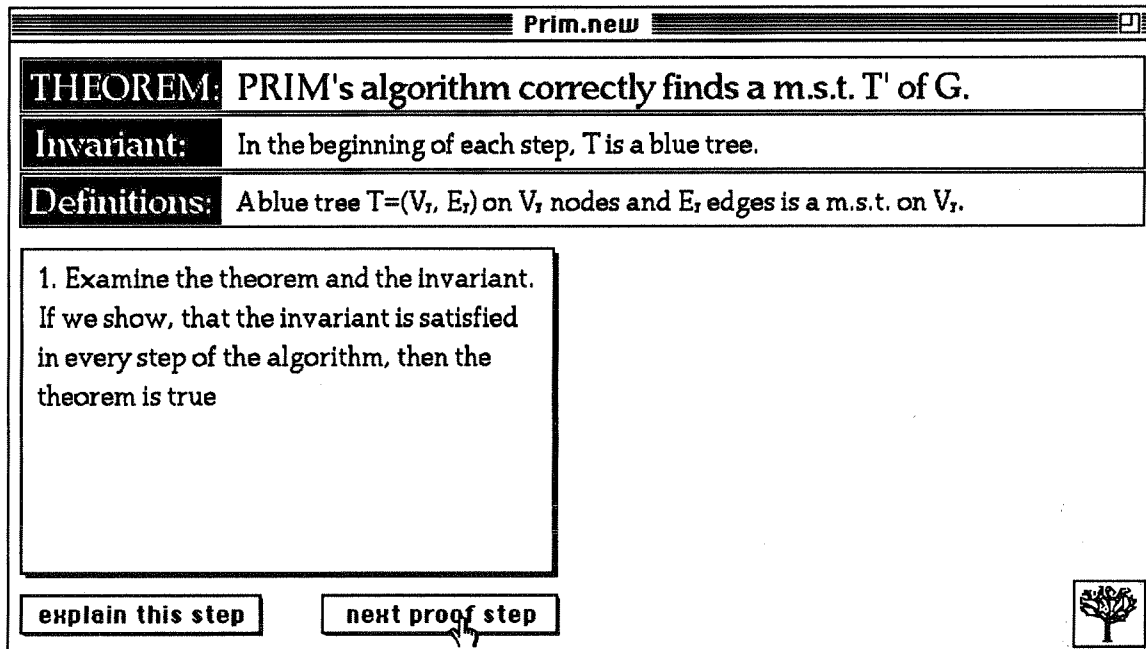


fig. 3. first step in proof of Prim's algorithm


Prim.new

THEOREM: PRIM's algorithm correctly finds a m.s.t. T' of G .

Invariant: In the beginning of each step, T is a blue tree.

Definitions: A blue tree $T=(V_T, E_T)$ on V_T nodes and E_T edges is a m.s.t. on V_T .

3. The invariant holds true for (s)



number of vertices in graph = 1

explain this step

next proof step




fig. 4 step 3 in proof of Prim's algorithm

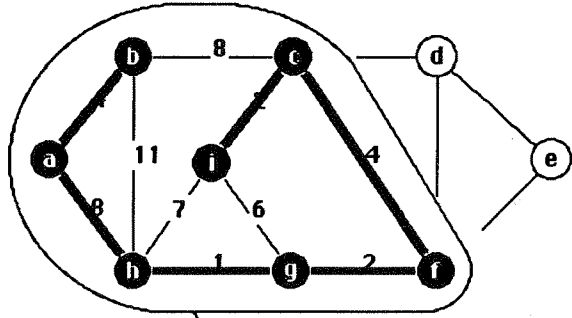
Prim.new

THEOREM: PRIM's algorithm correctly finds a m.s.t. T' of G .

Invariant: In the beginning of each step, T is a blue tree.

Definitions: A blue tree $T=(V_T, E_T)$ on V_T nodes and E_T edges is a m.s.t. on V_T .

4. Assumption hypothesis:
assume the invariant holds at the k -th step and let's assume that the invariant is violated for the first time in the $(k+1)$ st step with the incorporation of edge (v,w) into the blue tree.



T is m.s.t.

next proof step




fig. 5 step 4 in proof of Prim's algorithm

Prim.new

THEOREM: PRIM's algorithm correctly finds a m.s.t. T' of G .

Invariant: In the beginning of each step, T is a blue tree.

Definitions: A blue tree $T=(V_T, E_T)$ on V_T nodes and E_T edges is a m.s.t. on V_T .

5. Now we have to distinguish between two different cases:

a. There exists a MST T' on $V_T \cup \{w\}$ with exactly one edge (x, w) incident on w .

b. Every MST on $V_T \cup \{w\}$ has two (or more) edges incident on w .

next proof step

show definition of cut

fig. 6 step 5 in proof of Prim's algorithm

Prim.new

Definition: A *cut* $(S, V-S)$ of an undirected graph $G=(V,E)$ is a partition of V

fig. 7 definition of a "cut"

Prim.new

THEOREM: PRIM's algorithm correctly finds a m.s.t. T' of G .

Invariant: There exists a MST T' on $V_T \cup \{w\}$ with exactly one edge (x,w) incident on w .

Definition: T is a m.s.t. on V_T .

6. (explanation of 5a))
 By the invariant at step k ,
 $\text{cost}(T' - \{w\}) = \text{cost}(T)$, and by the algorithm,
show algorithm
 $\text{cost}(v,w) \leq \text{cost}(x,w)$, since x is in V_T .
show g, v'
 $\Rightarrow \text{cost}(T \cup (v,w)) \leq \text{cost}(T')$, a
 \Rightarrow contradiction.

next proof step

fig. 8 step 6 in proof of Prim's algorithm

Prim.new

THEOREM: PRIM's algorithm correctly finds a m.s.t. T' of G .

Invariant: Every MST on $V_T \cup \{w\}$ has two (or more) edges incident on w .

Definition: T is a m.s.t. on V_T .

7. (explanation of 5b))
 Let the two edges incident on w be (x,w)
 and (y,w) , where x and y are in V_T . Now
 consider the subgraph
 $H = T \cup (x,w) \cup (y,w)$.
 H will have exactly one cycle C , and C will
 contain edges (x,w) and (y,w) .

next proof step

fig. 9 step 7 in proof of Prim's algorithm

Prim.new

THEOREM: PRIM's algorithm correctly finds a m.s.t. T' of G .

Invariant: In the beginning of each step, T is a blue tree.

Definitions: A blue tree $T=(V, E_T)$ on V , nodes and E_T edges is a m.s.t. on V .

9. **5b** Without loss of generality, assume that x was admitted to the blue tree before y , and let (u,y) be in C .

show a 'u' Since both (x,w) and (y,w) are less costly than (u,y) , edges (x,w) and (y,w) would have to have been brought into the blue tree before edge (u,y) was a least costly cut edge. => contradiction.

next proof step

fig. 10 step 9 in proof of Prim's algorithm

Prim.new

THEOREM: PRIM's algorithm correctly finds a m.s.t. T' of G .

Invariant: In the beginning of each step, T is a blue tree.

Definitions: A blue tree $T=(V, E_T)$ on V , nodes and E_T edges is a m.s.t. on V .

10. Since neither case **5a** nor **5b** can happen, $T \cup \{v,w\}$ is a MST.

repeat proof

fig. 11 step 10 in proof of Prim's algorithm