Connected Graphs and Spanning Trees

GAINA, Daniel

Japan Advanced Institute of Science and Technology

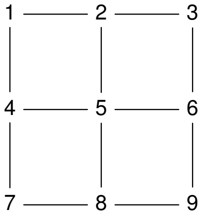
January 9, 2014

Describing the problem I

$G = (V, E)$ - graph

1. V - set of vertices
2. E - (multi)set of edges

Example:



$V = \{1, \ldots, 9\}$
$E = \{< 1, 2 >; < 1, 4 >; < 2, 3 >; < 2, 5 >; < 3, 6 >; < 4, 5 >; < 4, 7 >; < 5, 6 >; < 5, 8 >;$
$< 6, 9 >; < 7, 8 >; < 8, 9 >\}$

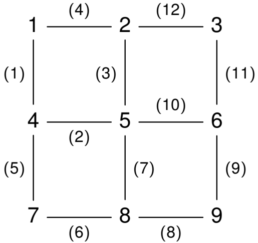
Describing the problem II

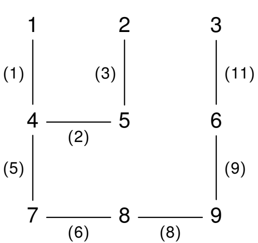
$G = (V, E)$ connected;

$T = (V, E')$ spanning tree of G when $\begin{cases} T \text{ is a tree} \\ E' \subseteq E \end{cases}$

Theorem

Every connected graph has a spanning tree.





Towards formalization

$$(\forall G : \texttt{Graph})\texttt{connected}(G) \Rightarrow (\exists G' : \texttt{Graph}) \subseteq G.\texttt{tree}(G')$$

We need a witness, $\texttt{mktree} : \texttt{Graph} \to \texttt{Graph}$

$$(\forall G : \texttt{Graph})\texttt{connected}(G) \Rightarrow \texttt{connected}(\texttt{mktree}(G)) \wedge \texttt{nocycle}(\texttt{mktree}(G))$$

To do :

1. data representations for mathematical objects ($\texttt{Graph}$)
2. define $\begin{cases} \texttt{connected} : \texttt{Graph} \to \texttt{Bool} \\ \texttt{nocycle} : \texttt{Graph} \to \texttt{Bool} \\ \texttt{mktree} : \texttt{Graph} \to \texttt{Graph} \end{cases}$

Spanning forests I

In the attempt of proving the desired properties we realized is much easier to prove a more general result:

Every graph has a spanning forest!

Definition

A **spanning forest** of a graph is a subgraph that consists of a set of spanning trees, one for each maximal connected component of the initial graph.

$\texttt{mktree} : \texttt{Graph} \to \texttt{Graph}, \texttt{mktree}(G) = \text{spanning forest of } G$

Spanning forests II

Example

```
G=({1,2,3,4,5,6,7,8}, <1,2>;<1,6>;<1,5>;<3,4>;<3,7>;<4,7>;<5,6>)
```

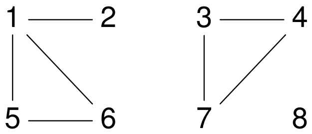

```
mktree(G)=({1,2,3,4,5,6,7,8},<1,2>;<1,6>;<1,5>;<3,4>;<3,7>)
```

Remark

The value is relative to the order chosen for the edges.

Functions on graphs

$G = (\{1, 2, 3, 4, 5, 6, 7, 8\}, < 1, 2 >; < 1, 6 >; < 1, 5 >; < 3, 4 >; < 3, 7 >; < 4, 7 >; < 5, 6 >)$



- $\begin{cases} \texttt{mcc} : \texttt{Vertex Graph} \to \texttt{VtxSet} \\ \texttt{mcc}(A, G) = \text{max. connected comp. of } A \text{ in G} \\ \texttt{mcc}(6, G) = \{1, 2, 5, 6\} \end{cases}$
- $\begin{cases} \texttt{nomcc} : \texttt{Graph} \to \texttt{Nat} \\ \texttt{ncc}(G) = \text{ no. of max. connected components} \\ \texttt{ncc}(G) = 3 \end{cases}$

Properties to prove

Assuming $\left(\begin{array}{l}\texttt{mcc} : \texttt{Vertex Graph} \to \texttt{VtxSet} \\ \texttt{nomcc} : \texttt{Graph} \to \texttt{Nat} \\ \texttt{nocycle} : \texttt{Graph} \to \texttt{Bool} \\ \texttt{mktree} : \texttt{Graph} \to \texttt{Graph}\end{array}\right)$ we need to prove

1. $(\forall G : \texttt{Graph}, A : \texttt{Vertex})\texttt{mcc}(A, \texttt{mktree}(G)) = \texttt{mcc}(A, G)$
2. $(\forall G : \texttt{Graph})\texttt{nomcc}(\texttt{mktree}(G)) = \texttt{nomcc}(G)$
3. $(\forall G : \texttt{Graph})\texttt{nocycle}(\texttt{mktree}(G)) = \texttt{true}$

Remark

If $\texttt{nomcc}(G) = 1$ then
$\texttt{connected}(G) \Rightarrow \texttt{connected}(\texttt{mktree}(G)) \wedge \texttt{nocycle}(\texttt{mktree}(G))$

Set I

```
(fmod VERTEX is ***> fth VERTEX
 pr BOOL . pr INT .
 op no :  -> Nat .
 sort Vertex .
 op _~_ :  Vertex Vertex -> Bool [comm] .
 vars I J : Vertex .
  eq   I ~ I = true .
  ceq  I = J if I ~ J [nonexec].
endfm)

(fmod VTXSET is
 pr VERTEX .
 sort VtxSet .
 subsorts Vertex < VtxSet .
 op empty :  -> VtxSet [ctor].
 op _U_ :  Vertex VtxSet -> VtxSet [ctor assoc comm].
 op _U_ :  VtxSet VtxSet -> VtxSet [assoc comm].
 vars A B : VtxSet .  vars I J : Vertex .
  eq (A U A) = A .
```

Set II

```
op _in_ :  Vertex VtxSet -> Bool .
 eq    I in empty = false .
 ceq   I in (J U A) = true     if I = J .
 ceq   I in (J U A) = I in A   if (I ~ J) = false .
*** ************************************************
op card :  VtxSet -> Nat .
 eq   card(empty) = 0 .
 eq   card(I U A) = 1 + card(A).
*** ************************************************
op _<_ :  VtxSet VtxSet -> Bool .
 eq    empty < B = true .
 ceq   I U A < B = A < B    if I in B .
 ceq   I U A < B = false    if I in B = false .
*** ************************************************
op _<>_ :  VtxSet VtxSet -> Bool [comm].
 eq    A <> A = true .
 ceq   A <> B = true     if A < B and B < A .
 ceq   A <> B = false    if not A < B .
 ceq   A <> B = false    if not B < A .
endfm)
```

GRAPH I

```
(fmod GRAPH is pr VTXSET .
 sorts Edge Graph .
 op <_`,_> :  Vertex Vertex -> Edge [ctor].
 op nil :  -> Graph [ctor].
 op _;_ :  Edge Graph -> Graph [ctor].
*** *****************************************
vars A B C : Vertex .  var G : Graph .
*** *****************************************
*** mcc(A,G)= max.  connected component of A in G ***
*** *****************************************
op mcc :  Vertex Graph -> VtxSet .
 eq   mcc(A,nil) = A .

 ceq  mcc(A,< B,C > ; G)= mcc(B,G) U mcc(C,G) if
      mcc(A,G) = mcc(B,G)                          [metadata "CA-1"].
 ceq  mcc(A,< B,C > ; G)= mcc(B,G) U mcc(C,G) if
      mcc(A,G) = mcc(C,G)                          [metadata "CA-2"].
 ceq  mcc(A,< B,C > ; G) = mcc(A,G) if
      mcc(A,G) <> mcc(B,G) = false ∧
      mcc(A,G) <> mcc(C,G) = false                 [metadata "CA-3"].
```

GRAPH II

```
op nocycle :  Graph -> Bool .
 eq   nocycle(nil) = true .
 ceq  nocycle(< A,B > ; G)= false if
      mcc(A,G) = mcc(B,G)                  [metadata "CA-1"].
 ceq  nocycle(< A,B > ; G)= nocycle(G) if
      mcc(A,G) <> mcc(B,G) = false         [metadata "CA-2"].
*** *****************************************************
*** nomcc(G) = number of max.  connected comp.  of G ***
*** *****************************************************
op nomcc :  Graph -> Int .
 eq   nomcc(nil) = no .

 ceq  nomcc(< A,B > ; G) = nomcc(G) if
      mcc(A,G) = mcc(B,G)                   [metadata "CA-1"].
 ceq  nomcc(< A,B > ; G) = nomcc(G) - 1 if
      mcc(A,G) <> mcc(B,G) = false          [metadata "CA-2"].
```

GRAPH III

```
*** ************************************************
*** mktree(G) returns the spanning forest of G
*** ************************************************
op mktree :  Graph -> Graph .
 eq    mktree(nil) = nil .

 ceq   mktree(< A,B > ; G)= mktree(G) if
       mcc(A,G) = mcc(B,G)                            [metadata "CA-1"].
 ceq   mktree(< A,B > ; G)= < A,B > ; mktree(G)if
       mcc(A,G) <> mcc(B,G)= false                    [metadata "CA-2"].
endfm)
```

Remark

- `Edge` and `Graph` are constrained.
- Models consist of interpretations of terms formed with constructors and elements of sort `Vertex`.

Properties to be proved

Theorem

$\texttt{mktree}(G)$ *is a spanning forest of G.*

Proof.

1. $(\forall G, A)\texttt{mcc}(A, \texttt{mktree}(G)) = \texttt{mcc}(A, G)$
2. $(\forall G)\texttt{nomcc}(\texttt{mktree}(G)) = \texttt{nomcc}(G)$
3. $(\forall G)\texttt{nocycle}(\texttt{mktree}(G)) = \texttt{true}$

$\square$

First Lemma

Lemma

$(\forall G, A)\mathtt{mcc}(A, \mathtt{mktree}(G)) = \mathtt{mcc}(A, G)$

Proof by induction on the structure of G

- IB $(\forall A)\mathtt{mcc}(A, \mathtt{mktree}(\mathtt{nil})) = \mathtt{mcc}(A, \mathtt{nil})$
- IS $(\forall A_1)\mathtt{mcc}(A_1, \mathtt{mktree}(\mathtt{g})) = \mathtt{mcc}(A_1, \mathtt{g}) \Rightarrow$
  $(\forall B, C)(\forall A_2)\mathtt{mcc}(A_2, \mathtt{mktree}(< B, C >; \mathtt{g})) = \mathtt{mcc}(A_2, < B, C >; \mathtt{g})$

For the induction base

```
(goal GRAPH |- eq mcc(A:Vertex,mktree(nil))= mcc(A:Vertex,nil);)
(apply TC RD .)
```

For the induction step

```
(fmod TH1 is --> fth TH1
 pr GRAPH .
 op g :  -> Graph .
 var A : Vertex .
 eq [IH]: mcc(A,mktree(g)) = mcc(A,g) .
endfm)

 (goal TH1 |-   eq mcc(A:Vertex,mktree(< B:Vertex, C:Vertex > ; g))=
                mcc(A:Vertex,< B:Vertex, C:Vertex > ; g) ;)
 (apply TC CA CA-1 RD .)
```

Second Lemma

Lemma

$(\forall G)\mathtt{nomcc}(\mathtt{mktree}(G)) = \mathtt{nomcc}(G)$

Proof by induction on the structure of `G`.

- IB `nomcc(mktree(nil))=nomcc(nil)`
- IS $\mathtt{nomcc}(\mathtt{mktree}(\mathtt{g})) = \mathtt{nomcc}(\mathtt{g}) \Rightarrow$
  $(\forall B, C)\mathtt{nomcc}(\mathtt{mktree}(< B, C >; \mathtt{g})) = \mathtt{nomcc}(< B, C >; \mathtt{g})$

For the induction base

```
(goal GRAPH |- eq nomcc(mktree(nil)) = nomcc(nil) ;)
(apply RD .)
```

For the induction step

```
(fmod TH2 is --> fth TH2
 pr GRAPH .
 op g :  -> Graph .
 var A : Vertex .  var G : Graph .
  eq [TH1]:  mcc(A,mktree(G)) = mcc(A,G) .
  eq [IH] :  nomcc(mktree(g)) = nomcc(g) .
endfm)

 (goal TH2 |-  eq nomcc(mktree(< A:Vertex,B:Vertex > ; g)) =
               nomcc(< A:Vertex,B:Vertex > ; g) ;)
 (apply TC CA RD .)
```

Third Lemma

Lemma

$(\forall G)\texttt{nocycle}(\texttt{mktree}(G)) = \texttt{true}$

Proof by induction on the structure of `G`.

IB `nocycle(mktree(nil))=true`

IS $\texttt{nocycle}(\texttt{mktree}(\texttt{g})) = \texttt{true} \Rightarrow$
$(\forall B, C)\texttt{nocycle}(\texttt{mktree}(< B, C >; \texttt{g})) = \texttt{true}$

For the induction base

```
(goal GRAPH |- eq nocycle(mktree(nil)) = true ;)
(apply RD .)
```

For the induction step

```
(fmod TH3 is --> fth TH3
 pr GRAPH .
 op g :  -> Graph .
 var A : Vertex .  var G : Graph .
 eq [TH1]:  mcc(A,mktree(G)) = mcc(A,G) .
 eq [TH2]:  nomcc(mktree(G)) = nomcc(G) .
 eq [IH]: nocycle(mktree(g)) = true .
endfm)
```

```
(goal TH3 |- eq nocycle(mktree(<A:Vertex,B:Vertex>; g))= true ;)
(apply TC CA RD .)
```

Conclusions

- we have proved a more general property (e.g every graph has a spanning forest) in order to achieve our goal;
- the data structure `VERTEX` for the set of vertices is very general and can be instantiated with natural numbers;
- this case study requires a more general induction scheme;
- most of the proof is automated with CITP;