Julius-Maximilians-
UNIVERSITÄT
WÜRZBURG

Approximation Algorithms for the Maximum Leaf Spanning Tree Problem on Acyclic Digraphs

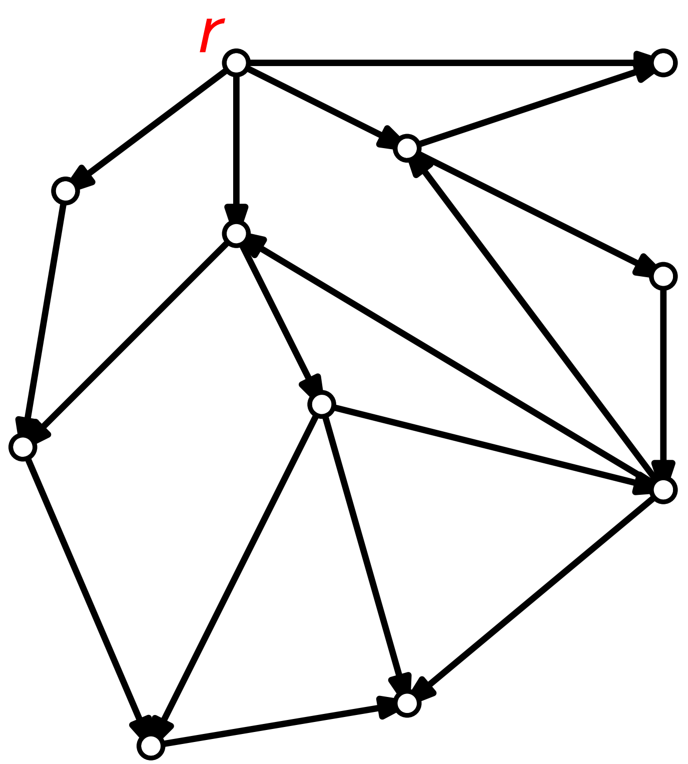
Nadine Schwartges · Joachim Spoerhase · Alexander Wolff

WAOA '11

Lehrstuhl für Informatik I
Universität Würzburg, Germany

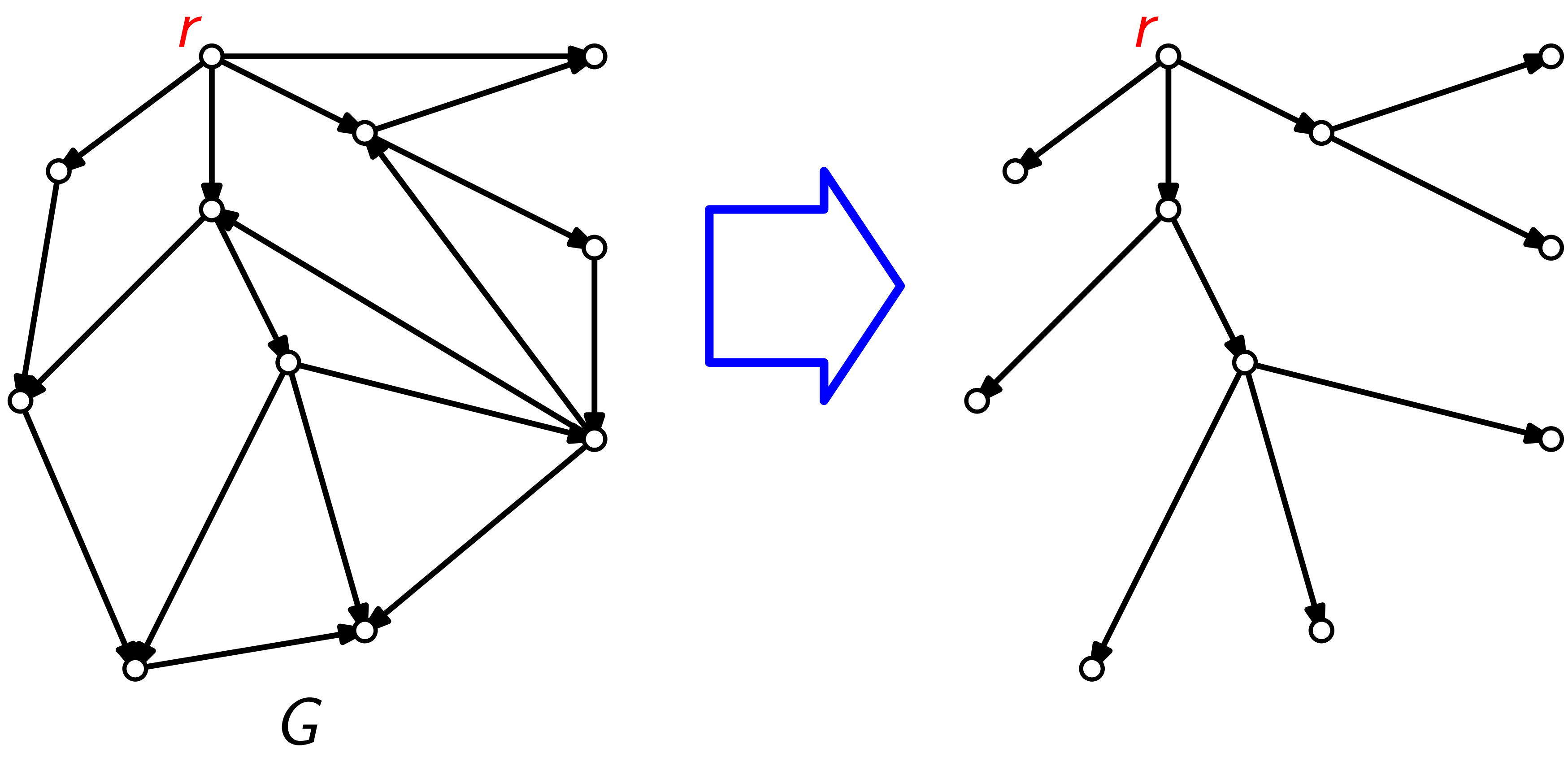
Problem Definition

Given a **digraph** G with **root** r,
find an r-rooted **spanning tree**
with the **maximum number of leaves**.



Problem Definition

Given a **digraph** G with **root** r,
find an r-rooted **spanning tree**
with the **maximum number of leaves**.



Previous Results

Undirected graphs

- classical NP-hard problem listed by Garey & Johnson ['74]

Previous Results

Undirected graphs

- classical NP-hard problem listed by Garey & Johnson ['74]
- 3-approximation by local search [Lu and Ravi '90]
  $\rightsquigarrow O(n^5)$ running time

Previous Results

Undirected graphs

- classical NP-hard problem listed by Garey & Johnson ['74]
- 3-approximation by local search [Lu and Ravi '90]
  $\rightsquigarrow O(n^5)$ running time
- 3-approximation in (almost) linear time by expansion-based approach [Lu and Ravi '98]

Previous Results

Undirected graphs

- classical NP-hard problem listed by Garey & Johnson ['74]
- 3-approximation by local search [Lu and Ravi '90]
  $\rightsquigarrow$ $O(n^5)$ running time
- 3-approximation in (almost) linear time by expansion-based approach [Lu and Ravi '98]
- linear-time **2-approximation** by expansion [Solis-Oba '98]

Previous Results

Undirected graphs

- classical NP-hard problem listed by Garey & Johnson ['74]
- 3-approximation by local search [Lu and Ravi '90]
  $\rightsquigarrow O(n^5)$ running time
- 3-approximation in (almost) linear time by expansion-based approach [Lu and Ravi '98]
- linear-time **2-approximation** by expansion [Solis-Oba '98]

Digraphs

- $\sqrt{\mathrm{OPT}}$-approximation [Drescher and Vetta, '10]
- **92-approximation** [Daligault and Thomassé, 10]

Previous Results

Undirected graphs

- classical NP-hard problem listed by Garey & Johnson ['74]
- 3-approximation by local search [Lu and Ravi '90]
  $\rightsquigarrow O(n^5)$ running time
- 3-approximation in (almost) linear time by expansion-based approach [Lu and Ravi '98]
- linear-time **2-approximation** by expansion [Solis-Oba '98]

Digraphs

- $\sqrt{\mathrm{OPT}}$-approximation [Drescher and Vetta, '10]
- **92-approximation** [Daligault and Thomassé, 10]
- What about special classes of digraphs?

Previous Results

Undirected graphs

- classical NP-hard problem listed by Garey & Johnson ['74]
- 3-approximation by local search [Lu and Ravi '90]
  $\rightsquigarrow$ $O(n^5)$ running time
- 3-approximation in (almost) linear time by expansion-based approach [Lu and Ravi '98]
- linear-time **2-approximation** by expansion [Solis-Oba '98]

Digraphs

- $\sqrt{\mathrm{OPT}}$-approximation [Drescher and Vetta, '10]
- **92-approximation** [Daligault and Thomassé, 10]
- What about special classes of digraphs? $\rightsquigarrow$ **DAGs**

Our Results for DAGs

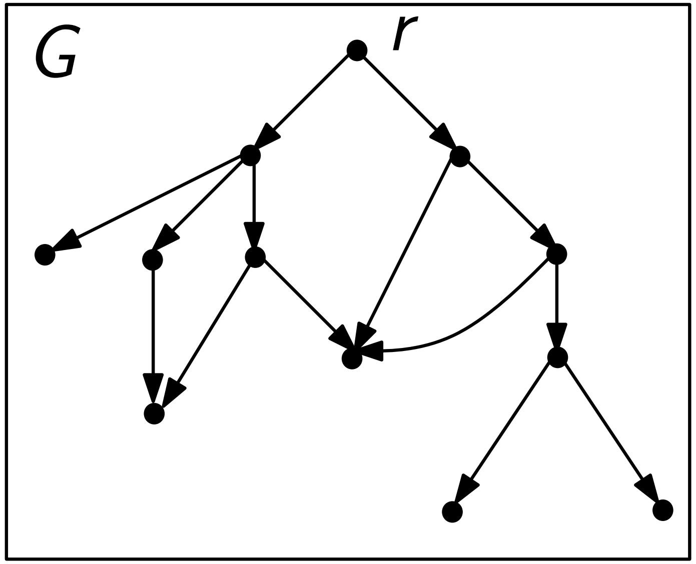
- MaxSNP-hard, i.e., no PTAS
- linear-time 4-approximation algorithm
- linear-time 2-approximation algorithm (expansion-based)

Our Results for DAGs

- MaxSNP-hard, i.e., no PTAS
- linear-time 4-approximation algorithm
- linear-time 2-approximation algorithm (expansion-based)

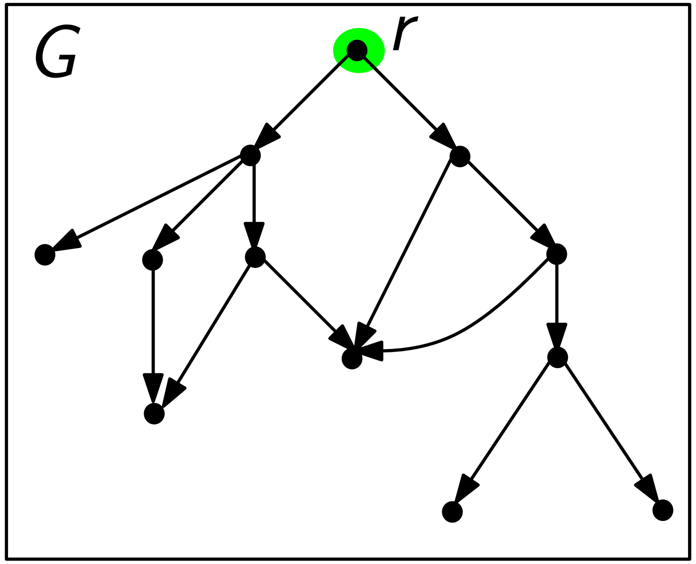
Algorithm

Input: acyclic digraph G with root r
Output: spanning tree T
mark r
$F \leftarrow \texttt{expand}(G)$
$T \leftarrow \texttt{connect}(G, F)$
return T



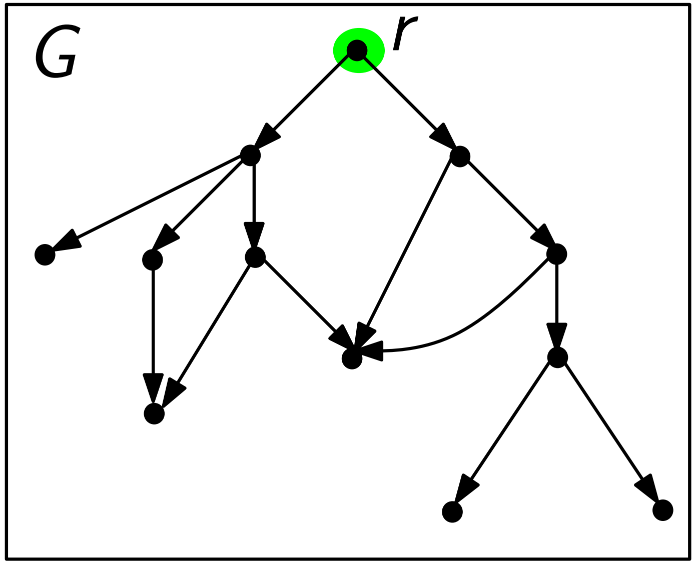
Algorithm

Input: acyclic digraph G with root r
Output: spanning tree T
mark r
$F \leftarrow \texttt{expand}(G)$
$T \leftarrow \texttt{connect}(G, F)$
return T



Algorithm

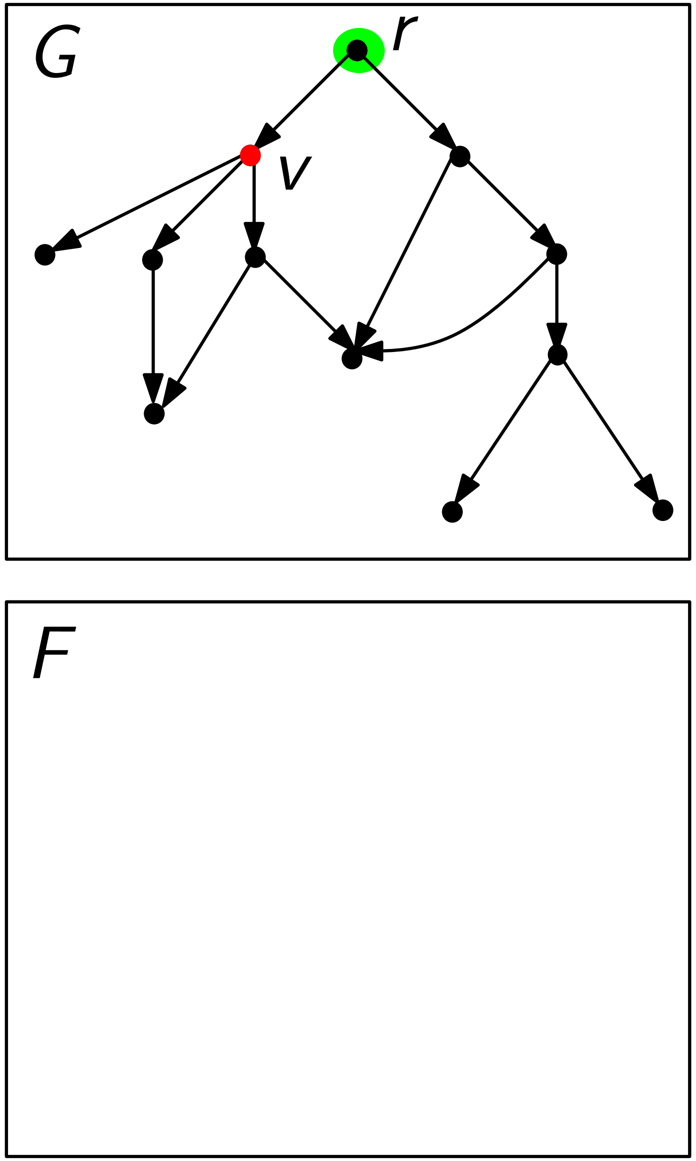
Input: acyclic digraph G with root r
Output: spanning tree T
mark r
$F \leftarrow \texttt{expand}(G)$
$T \leftarrow \texttt{connect}(G, F)$
return T



Procedure `expand`

$F \leftarrow \emptyset$

foreach node v in G **do**



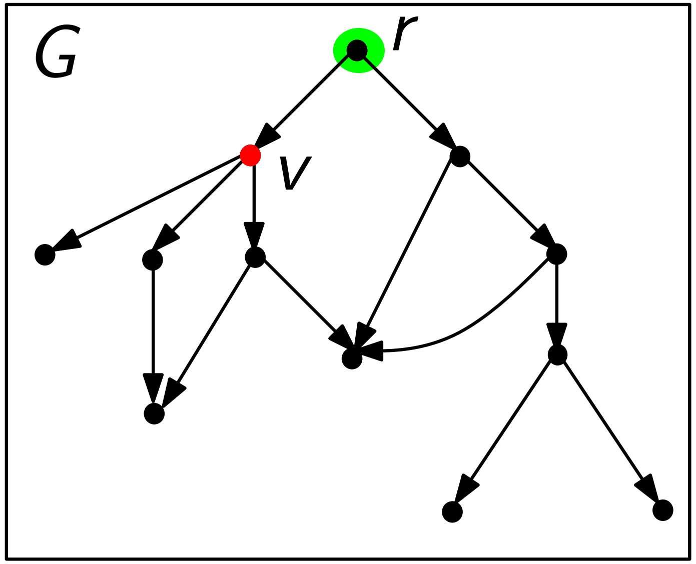
Procedure `expand`

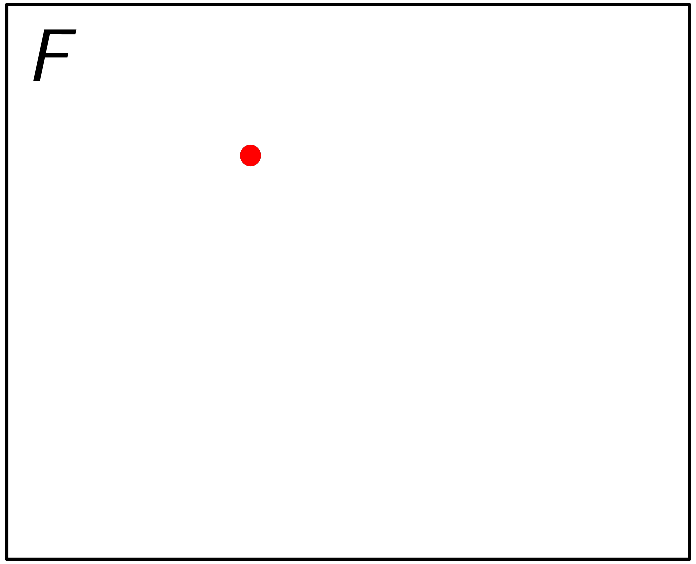
$F \leftarrow \emptyset$

foreach node v in G **do**

  if $v \notin F$ **then**

    $F \leftarrow F + v$





Procedure `expand`

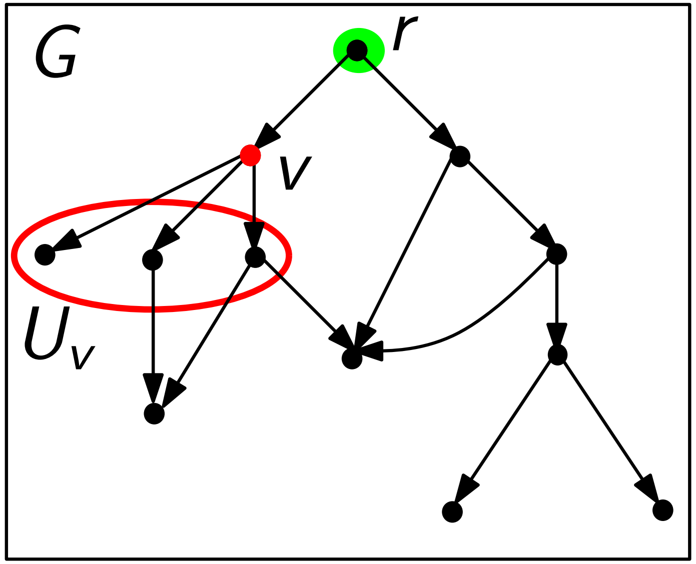
$F \leftarrow \emptyset$

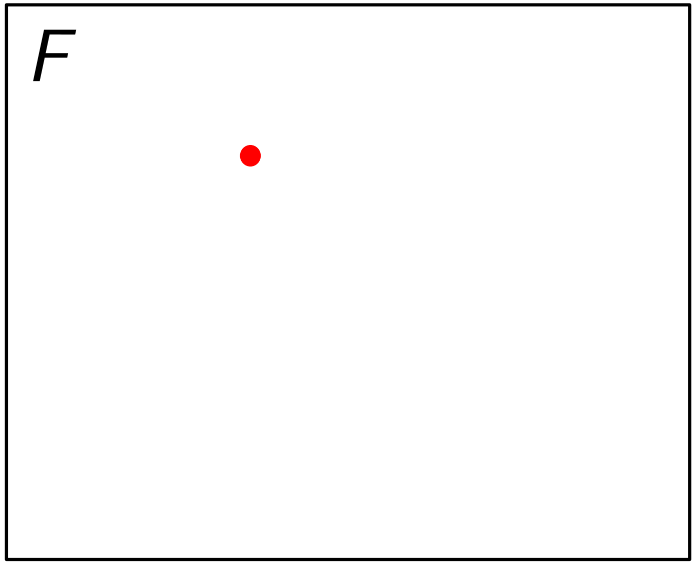
foreach node v in G **do**

  if $v \notin F$ **then**

    $F \leftarrow F + v$

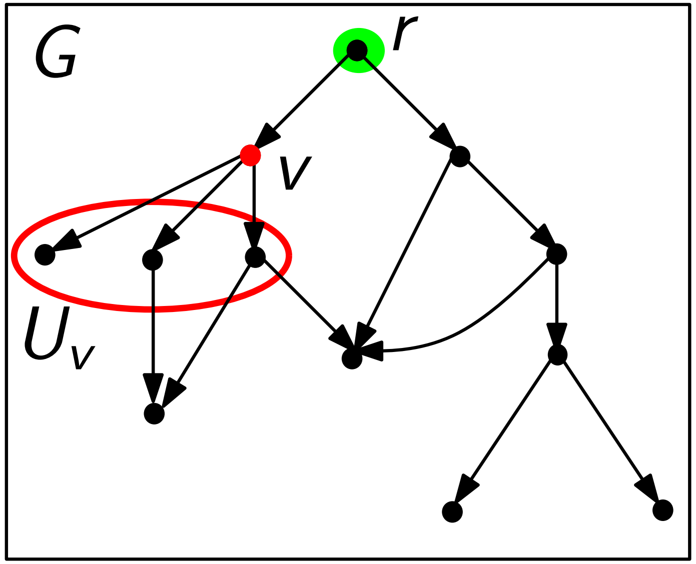
  $U_v \leftarrow$ unmarked children of v

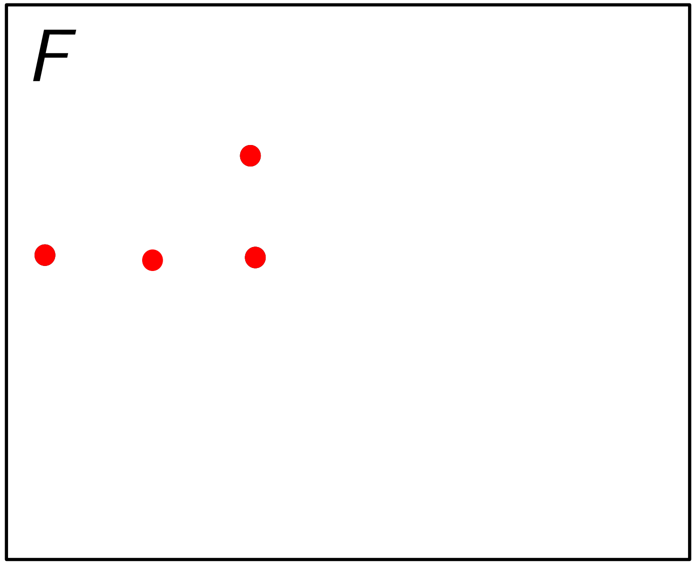




Procedure expand

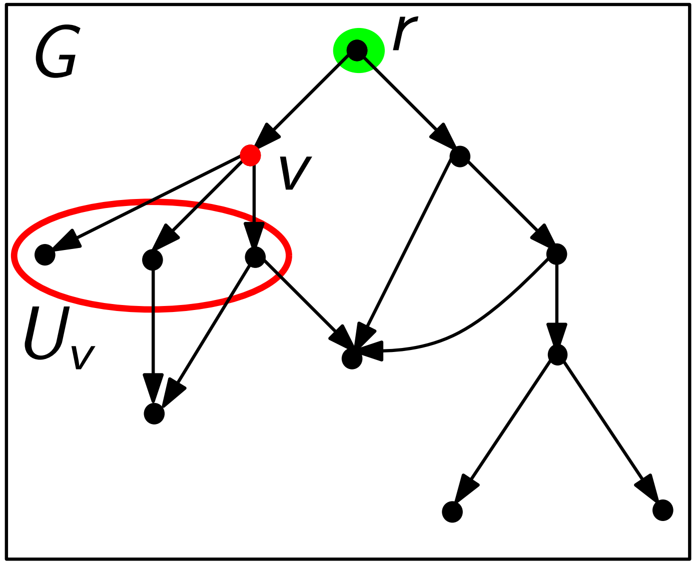
$F \leftarrow \emptyset$
foreach node v in G **do**
    if $v \notin F$ **then**
        $F \leftarrow F + v$
    $U_v \leftarrow$ unmarked children of v
    if $|U_v| \geq 2$ **then**
        $F \leftarrow F + U_v$

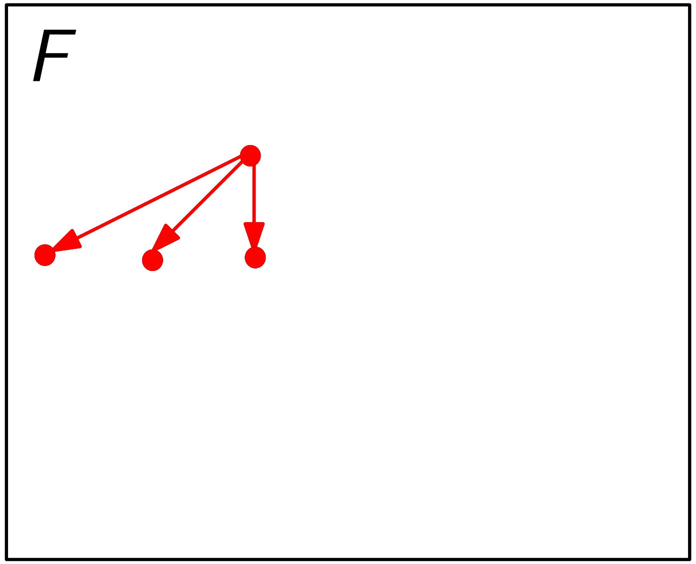




Procedure expand

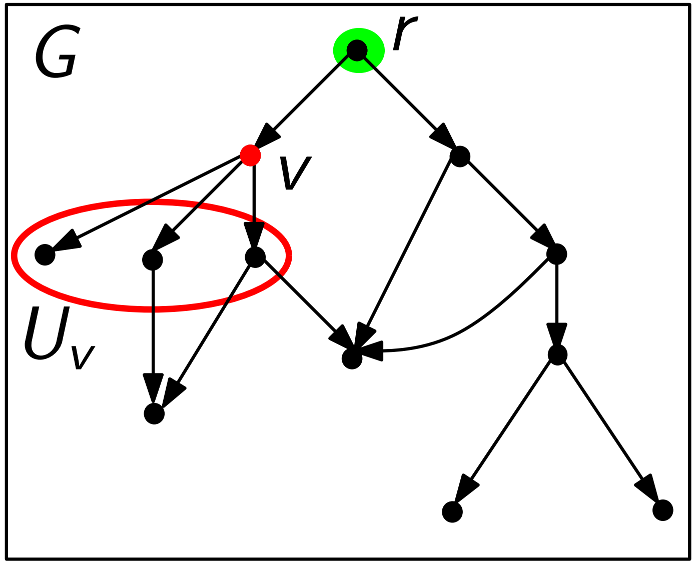
$F \leftarrow \emptyset$
foreach node v in G **do**
  if $v \notin F$ **then**
    $F \leftarrow F + v$
  $U_v \leftarrow$ unmarked children of v
  if $|U_v| \geq 2$ **then**
    $F \leftarrow F + U_v$
    foreach $u \in U_v$ **do**
      $F \leftarrow F + (v, u)$

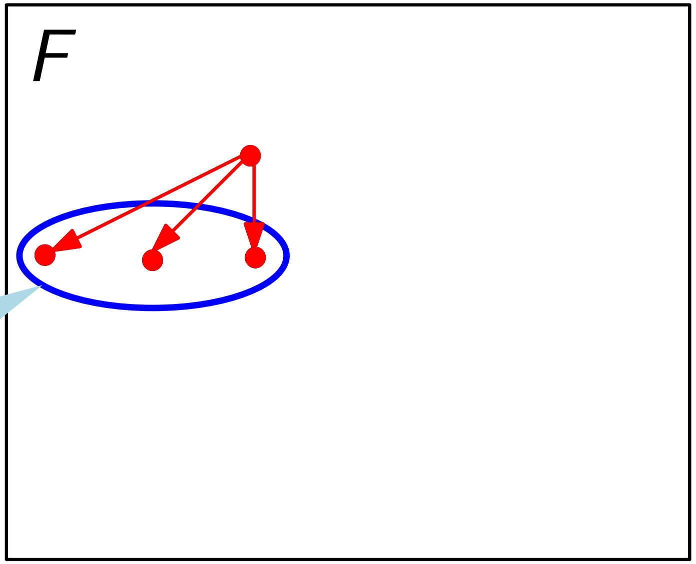




Procedure expand

$F \leftarrow \emptyset$
foreach node v in G **do**
  if $v \notin F$ **then**
    $F \leftarrow F + v$
  $U_v \leftarrow$ unmarked children of v
  if $|U_v| \geq 2$ **then**
    $F \leftarrow F + U_v$
    foreach $u \in U_v$ **do**
      $F \leftarrow F + (v, u)$

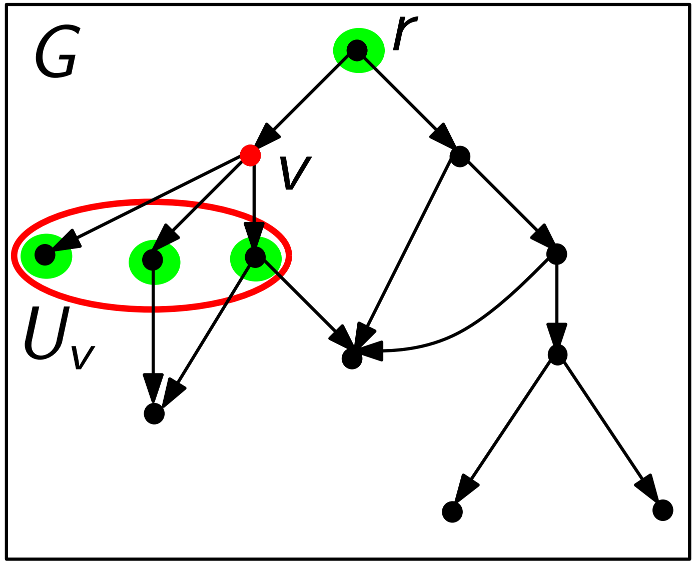


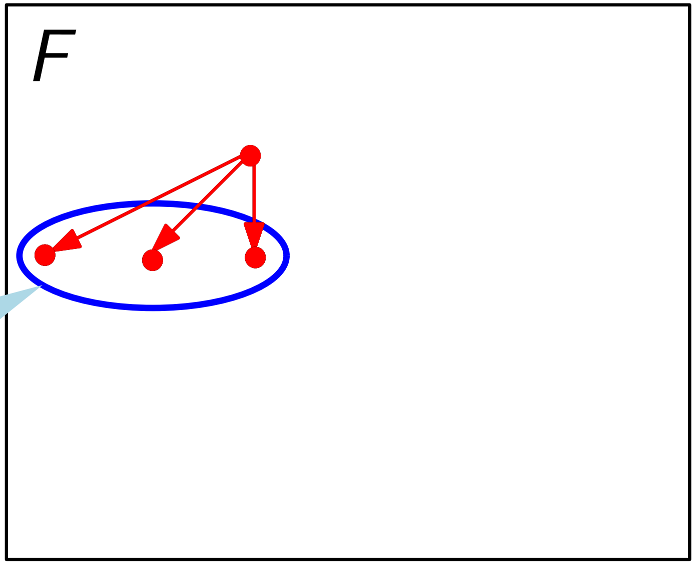


every $u \in U_v$ has exactly one incoming edge in F

Procedure expand

$F \leftarrow \emptyset$
foreach node v in G **do**
  if $v \notin F$ **then**
    $F \leftarrow F + v$
  $U_v \leftarrow$ unmarked children of v
  if $|U_v| \geq 2$ **then**
    $F \leftarrow F + U_v$
    foreach $u \in U_v$ **do**
      $F \leftarrow F + (v, u)$
      mark u





every $u \in U_v$ has exactly one incoming edge in F

Procedure expand

$F \leftarrow \emptyset$

foreach node v in G **do**

  if $v \notin F$ **then**

    $F \leftarrow F + v$

  $U_v \leftarrow$ unmarked children of v

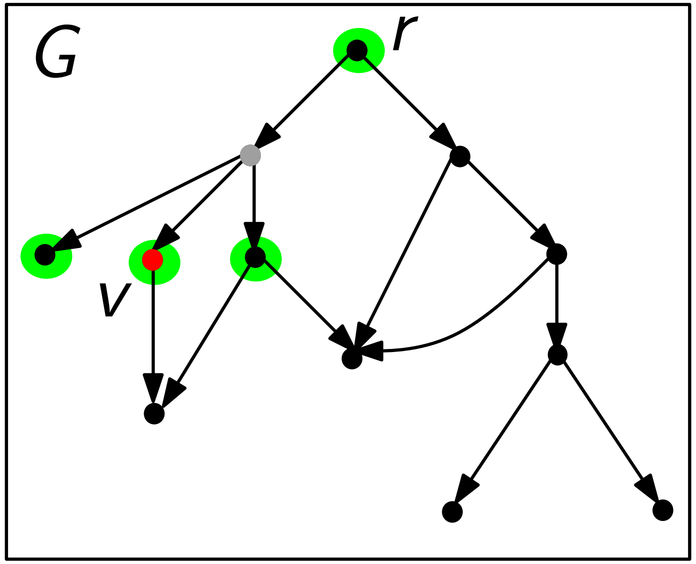
  if $|U_v| \geq 2$ **then**

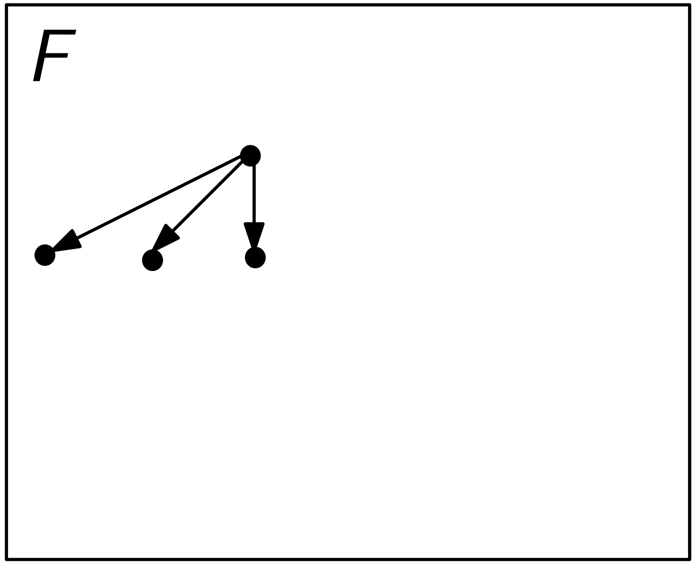
    $F \leftarrow F + U_v$

    foreach $u \in U_v$ **do**

      $F \leftarrow F + (v, u)$

      mark u



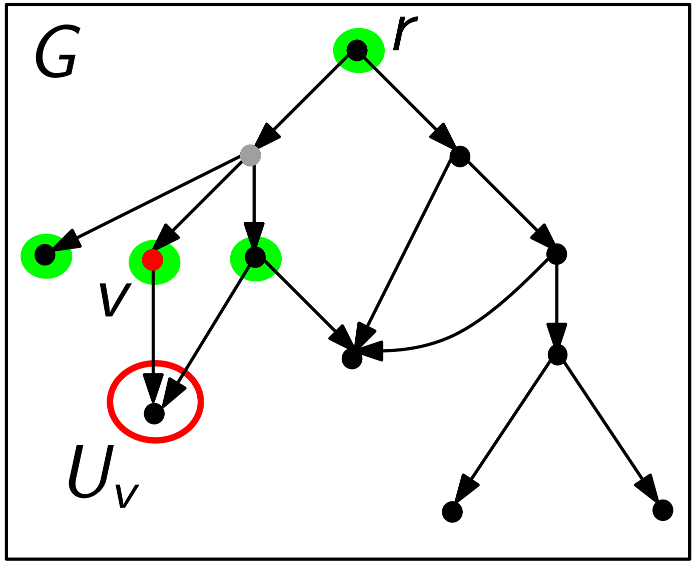


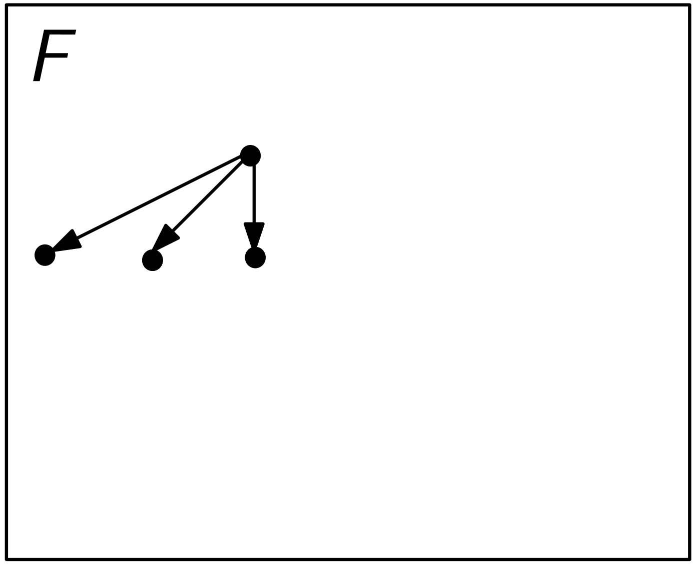
Procedure $\texttt{expand}$

$F \leftarrow \emptyset$

foreach node v in G **do**

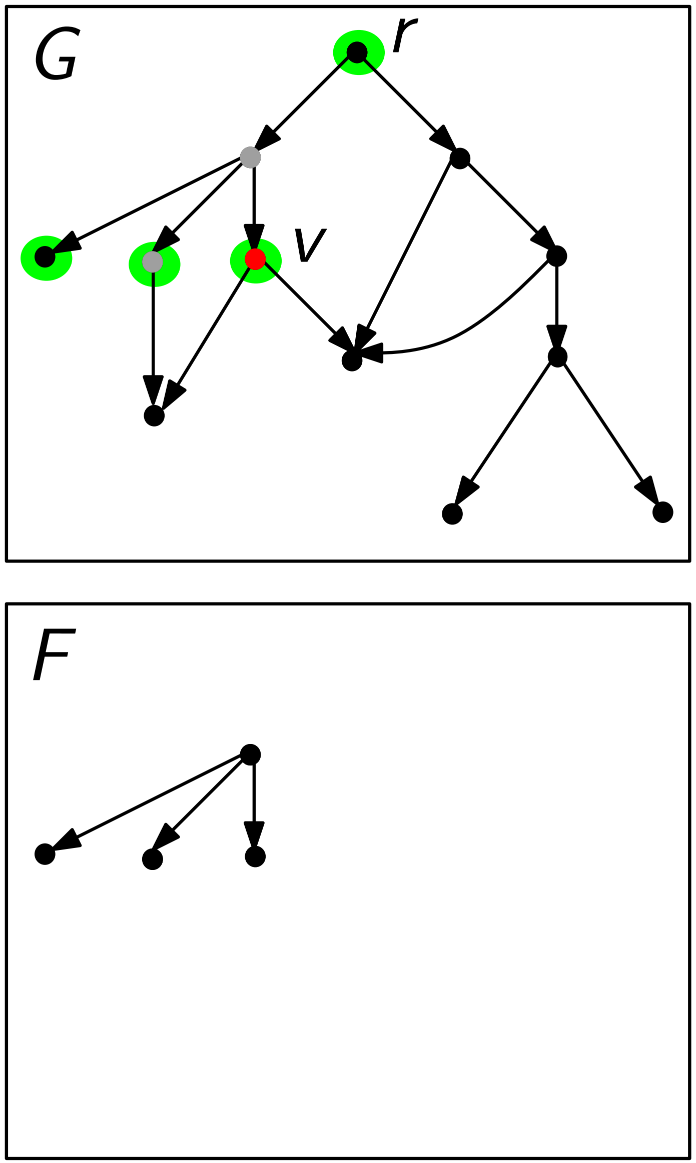
- **if** $v \notin F$ **then**
  - $F \leftarrow F + v$
- $U_v \leftarrow$ unmarked children of v
- **if** $|U_v| \geq 2$ **then**
  - $F \leftarrow F + U_v$
  - **foreach** $u \in U_v$ **do**
    - $F \leftarrow F + (v, u)$
    - mark u





Procedure $\texttt{expand}$

$F \leftarrow \emptyset$
foreach node v in G **do**
    if $v \notin F$ **then**
        $F \leftarrow F + v$
    $U_v \leftarrow$ unmarked children of v
    if $|U_v| \geq 2$ **then**
        $F \leftarrow F + U_v$
        foreach $u \in U_v$ **do**
            $F \leftarrow F + (v, u)$
            mark u



Procedure expand

$F \leftarrow \emptyset$

foreach node v in G **do**

  if $v \notin F$ **then**

    $F \leftarrow F + v$

  $U_v \leftarrow$ unmarked children of v

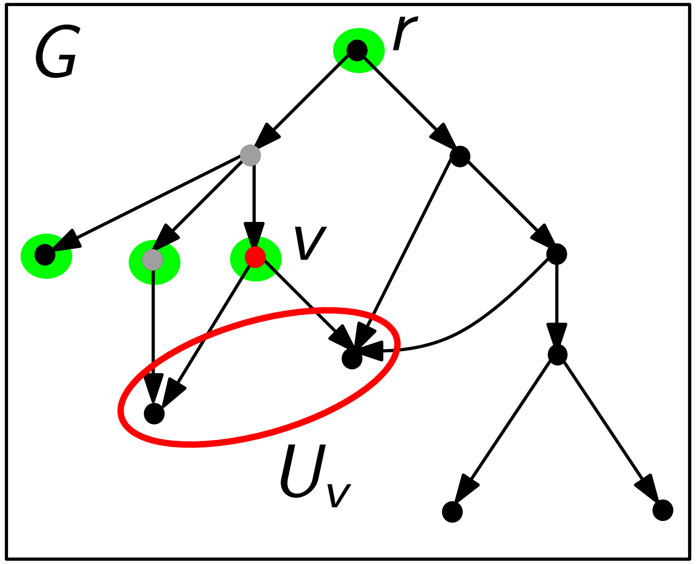
  if $|U_v| \geq 2$ **then**

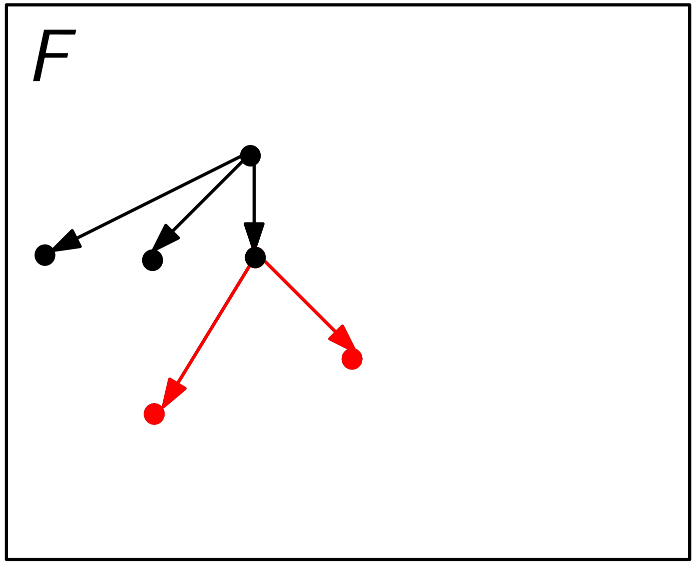
    $F \leftarrow F + U_v$

    foreach $u \in U_v$ **do**

      $F \leftarrow F + (v, u)$

      mark u





Procedure expand

$F \leftarrow \emptyset$

foreach node v in G **do**

    if $v \notin F$ **then**

        $F \leftarrow F + v$

    $U_v \leftarrow$ unmarked children of v

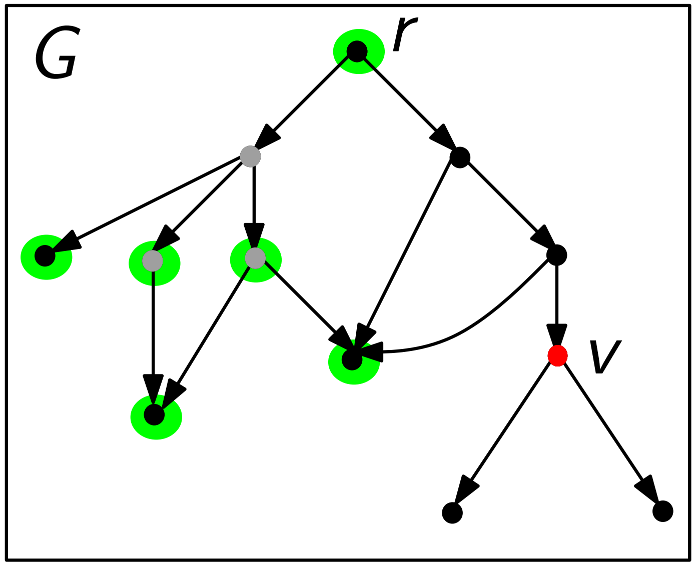
    if $|U_v| \geq 2$ **then**

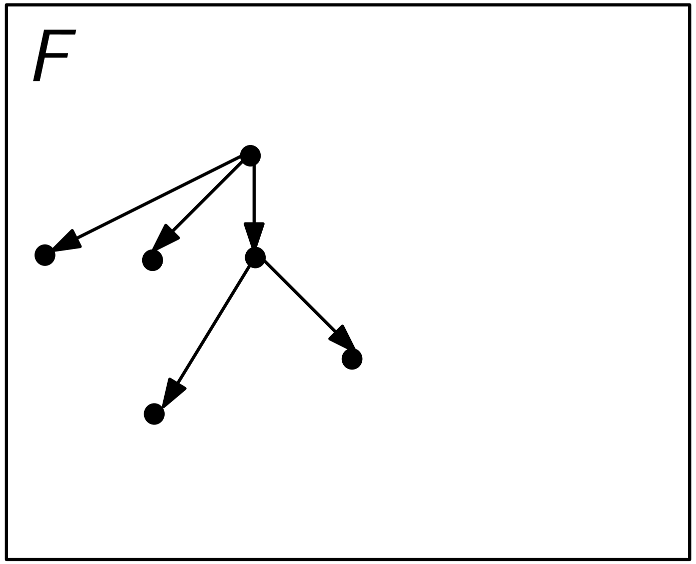
        $F \leftarrow F + U_v$

        foreach $u \in U_v$ **do**

            $F \leftarrow F + (v, u)$

            mark u





Procedure $\texttt{expand}$

$F \leftarrow \emptyset$

foreach node v in G **do**

  if $v \notin F$ **then**

    $F \leftarrow F + v$

  $U_v \leftarrow$ unmarked children of v

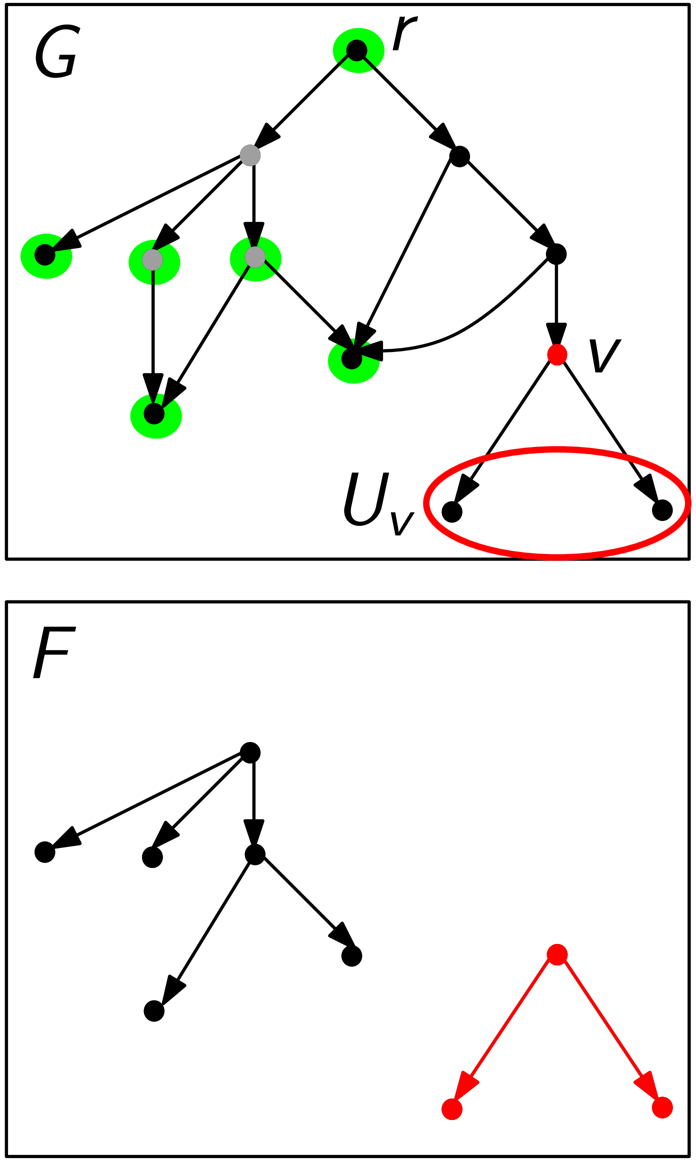
  if $|U_v| \geq 2$ **then**

    $F \leftarrow F + U_v$

    foreach $u \in U_v$ **do**

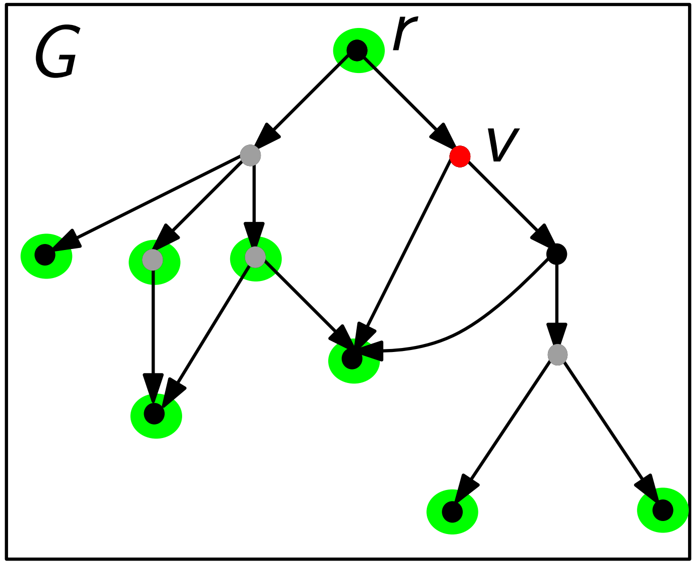
      $F \leftarrow F + (v, u)$

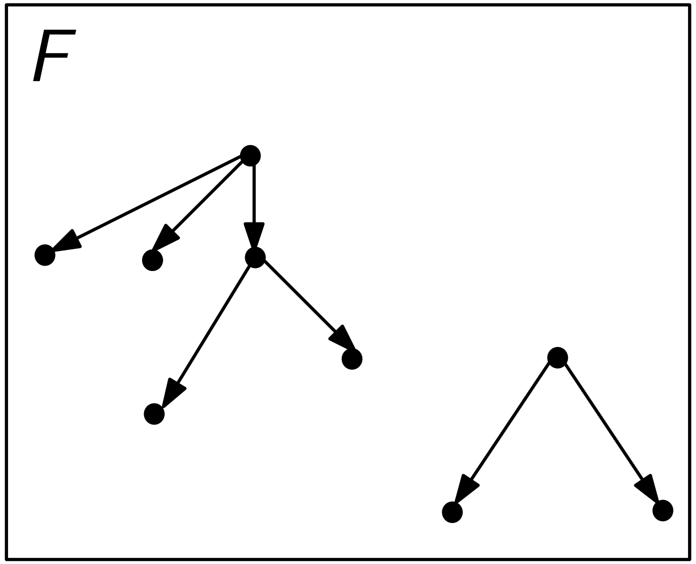
      mark u



Procedure expand

$F \leftarrow \emptyset$
foreach node v in G **do**
  if $v \notin F$ **then**
    $F \leftarrow F + v$
  $U_v \leftarrow$ unmarked children of v
  if $|U_v| \geq 2$ **then**
    $F \leftarrow F + U_v$
    foreach $u \in U_v$ **do**
      $F \leftarrow F + (v, u)$
      mark u





Procedure expand

$F \leftarrow \emptyset$

foreach node v in G **do**

    if $v \notin F$ **then**

        $F \leftarrow F + v$

    $U_v \leftarrow$ unmarked children of v

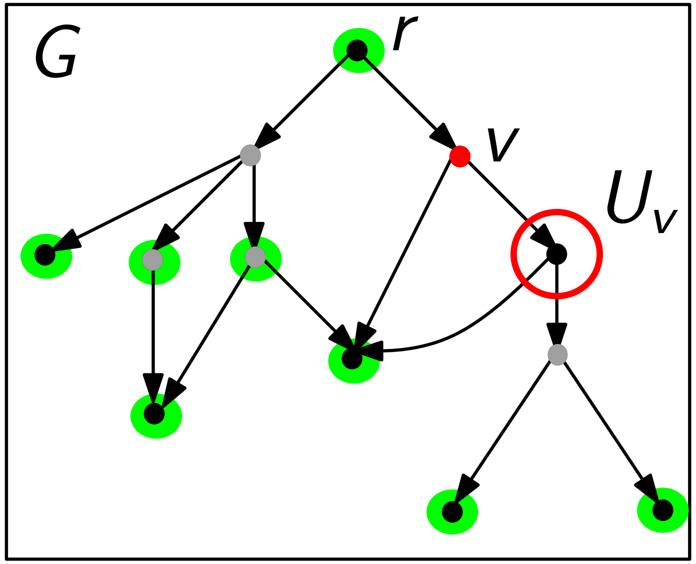
    if $|U_v| \geq 2$ **then**

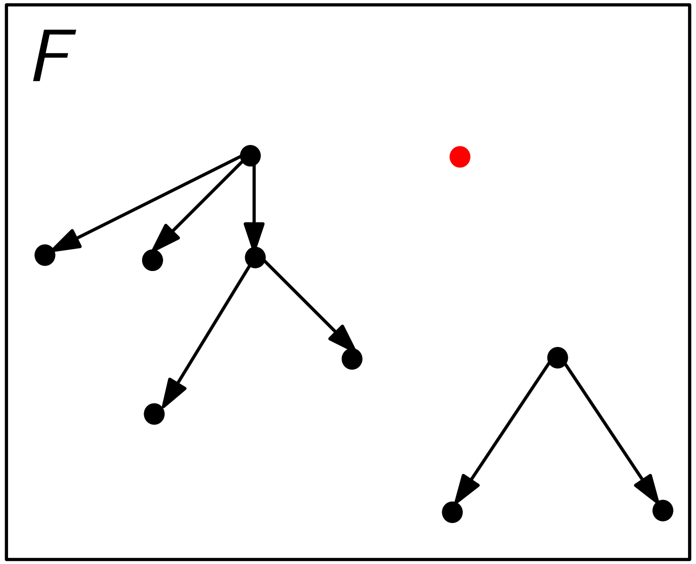
        $F \leftarrow F + U_v$

        foreach $u \in U_v$ **do**

            $F \leftarrow F + (v, u)$

            mark u





Procedure expand

$F \leftarrow \emptyset$

foreach node v in G **do**

  if $v \notin F$ **then**

    $F \leftarrow F + v$

  $U_v \leftarrow$ unmarked children of v

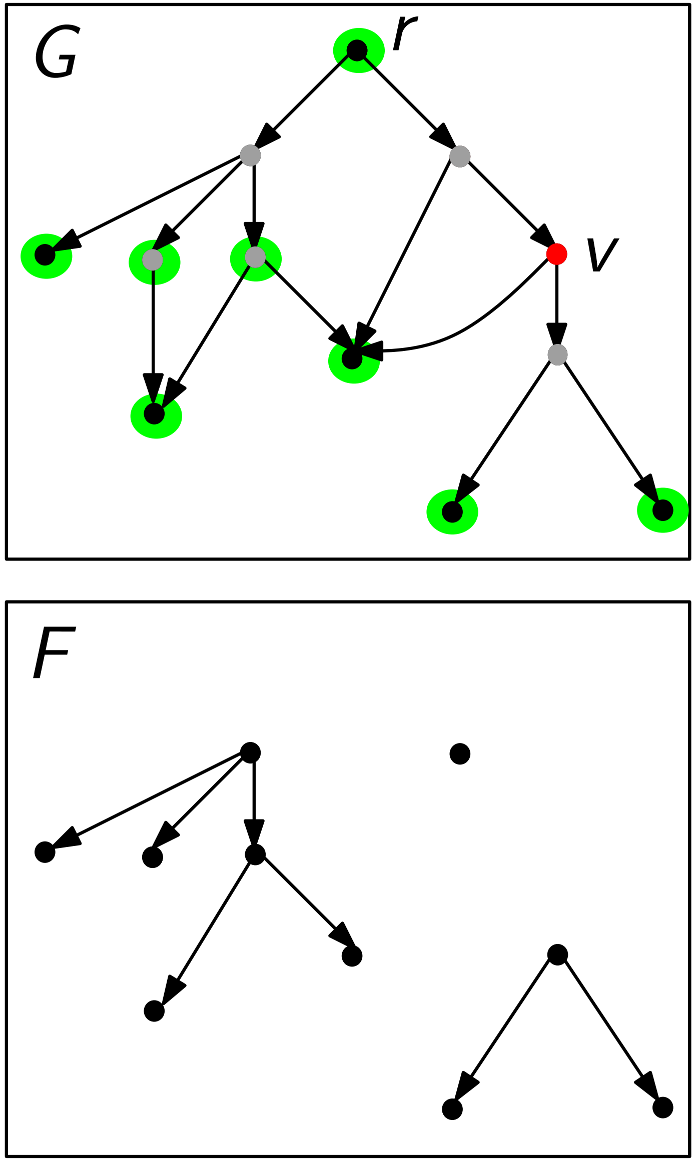
  if $|U_v| \geq 2$ **then**

    $F \leftarrow F + U_v$

    foreach $u \in U_v$ **do**

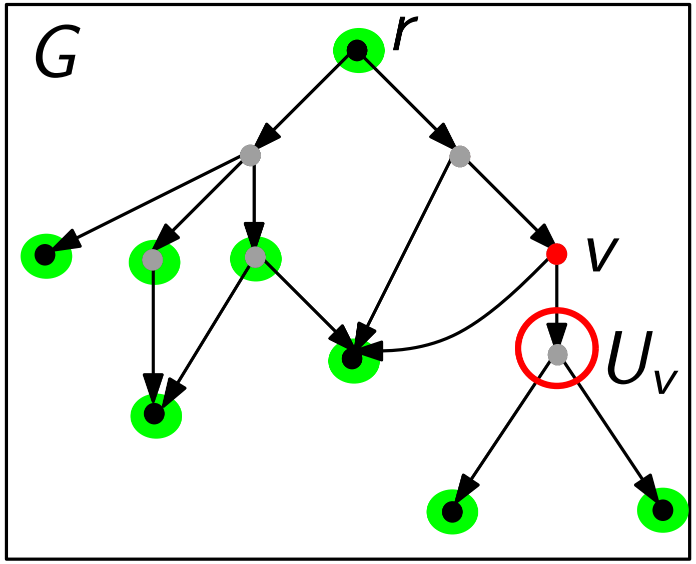
      $F \leftarrow F + (v, u)$

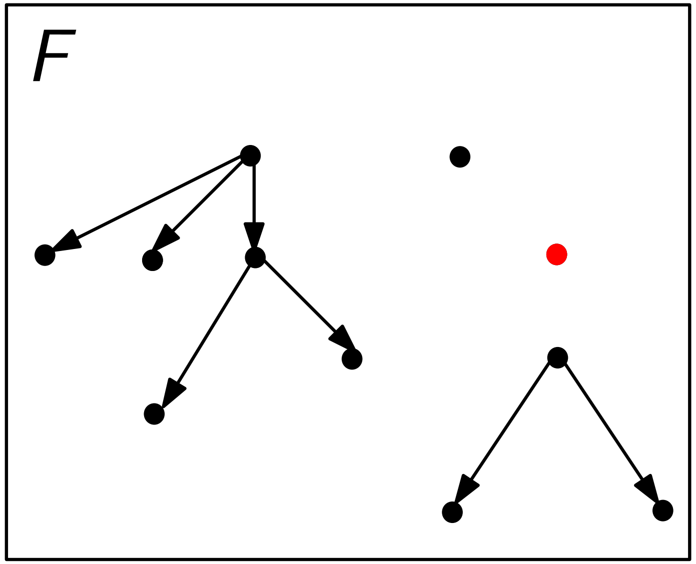
      mark u



Procedure expand

$F \leftarrow \emptyset$
foreach node v in G **do**
  if $v \notin F$ **then**
    $F \leftarrow F + v$
  $U_v \leftarrow$ unmarked children of v
  if $|U_v| \geq 2$ **then**
    $F \leftarrow F + U_v$
    foreach $u \in U_v$ **do**
      $F \leftarrow F + (v, u)$
      mark u





Procedure expand

$F \leftarrow \emptyset$

foreach node v in G **do**

  if $v \notin F$ **then**

    $F \leftarrow F + v$

  $U_v \leftarrow$ unmarked children of v

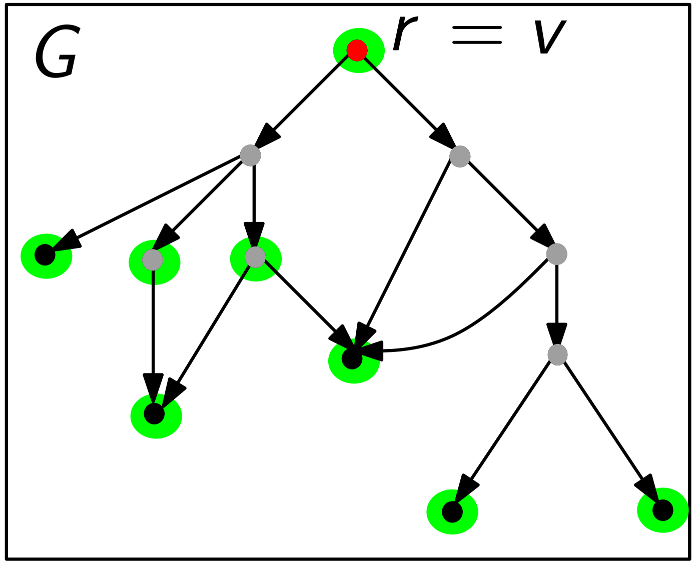
  if $|U_v| \geq 2$ **then**

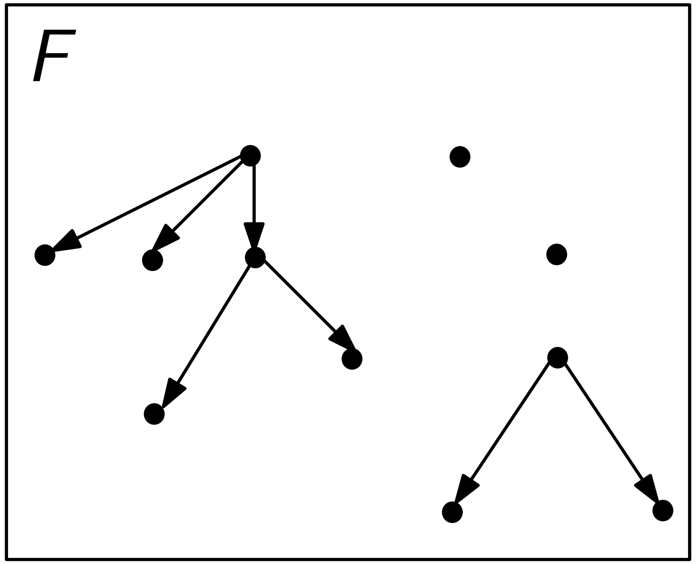
    $F \leftarrow F + U_v$

    foreach $u \in U_v$ **do**

      $F \leftarrow F + (v, u)$

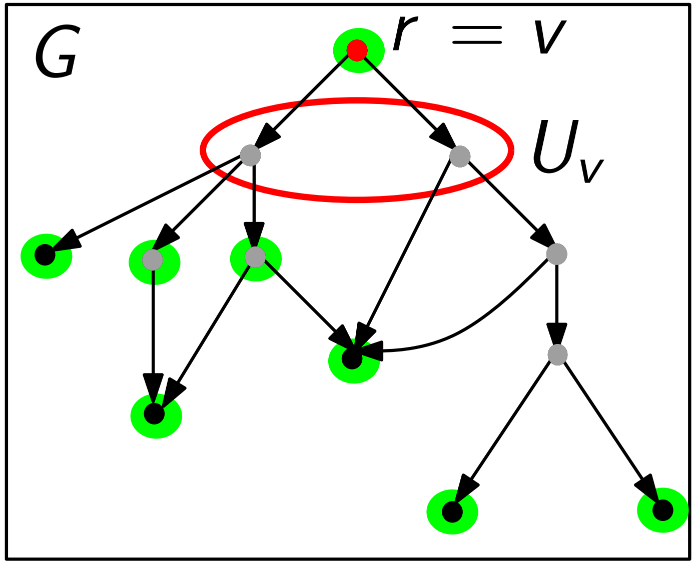
      mark u

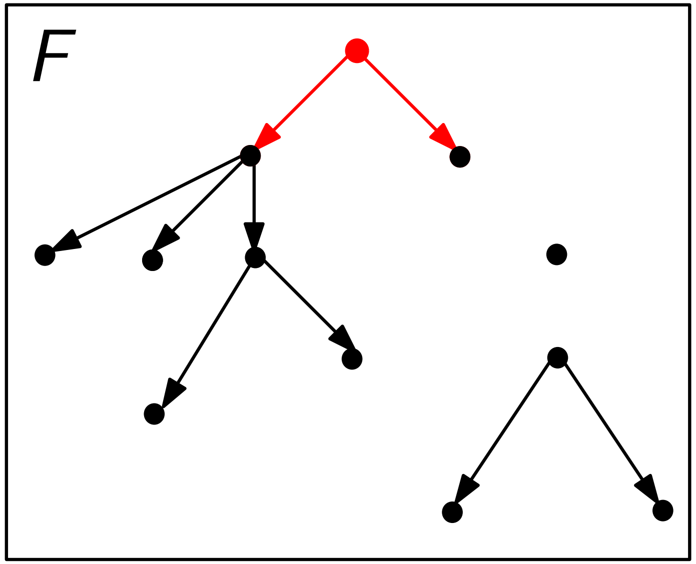




Procedure expand

$F \leftarrow \emptyset$
foreach node v in G **do**
    if $v \notin F$ **then**
        $F \leftarrow F + v$
    $U_v \leftarrow$ unmarked children of v
    if $|U_v| \geq 2$ **then**
        $F \leftarrow F + U_v$
        foreach $u \in U_v$ **do**
            $F \leftarrow F + (v, u)$
            mark u





Procedure expand

$F \leftarrow \emptyset$

foreach node v in G **do**

  if $v \notin F$ **then**

    $F \leftarrow F + v$

  $U_v \leftarrow$ unmarked children of v

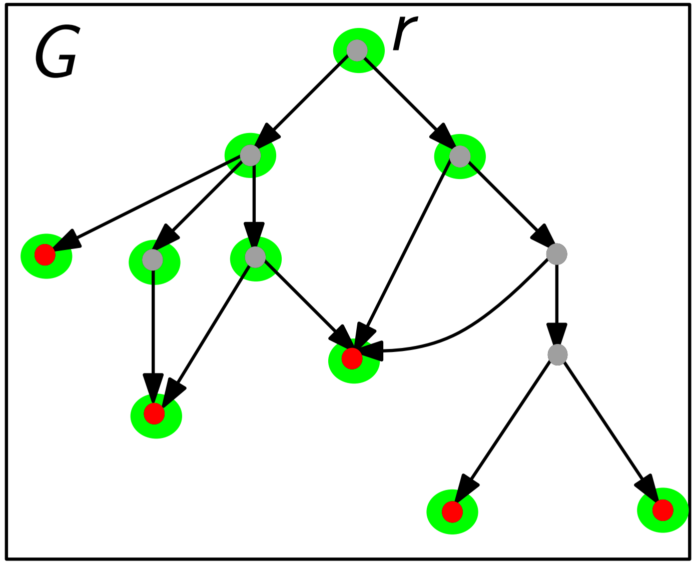
  if $|U_v| \geq 2$ **then**

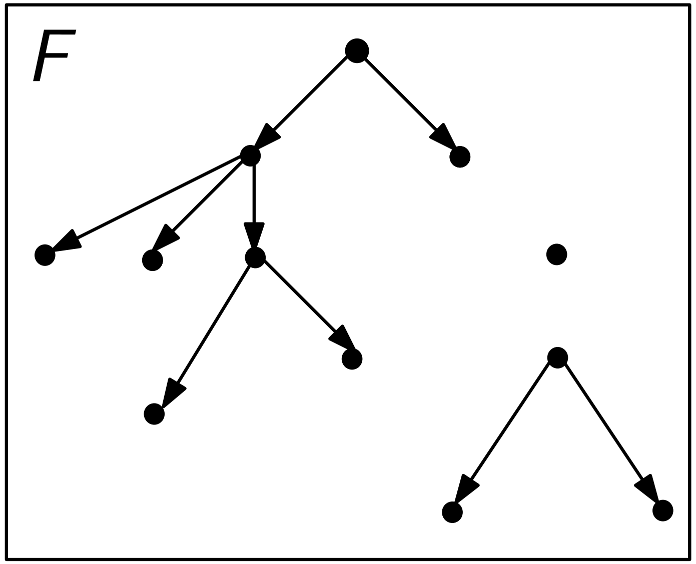
    $F \leftarrow F + U_v$

    foreach $u \in U_v$ **do**

      $F \leftarrow F + (v, u)$

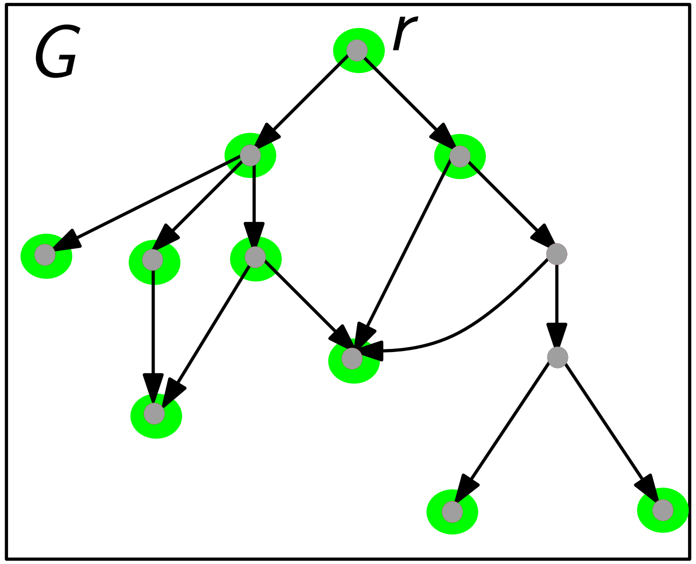
      mark u

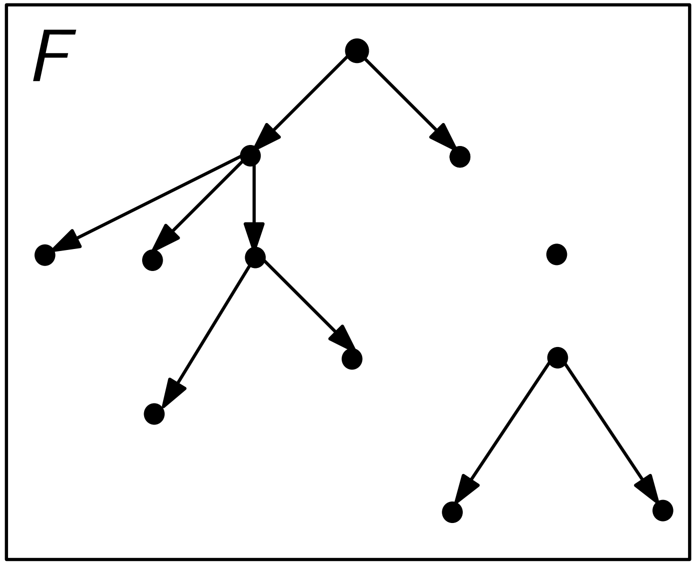




Algorithm

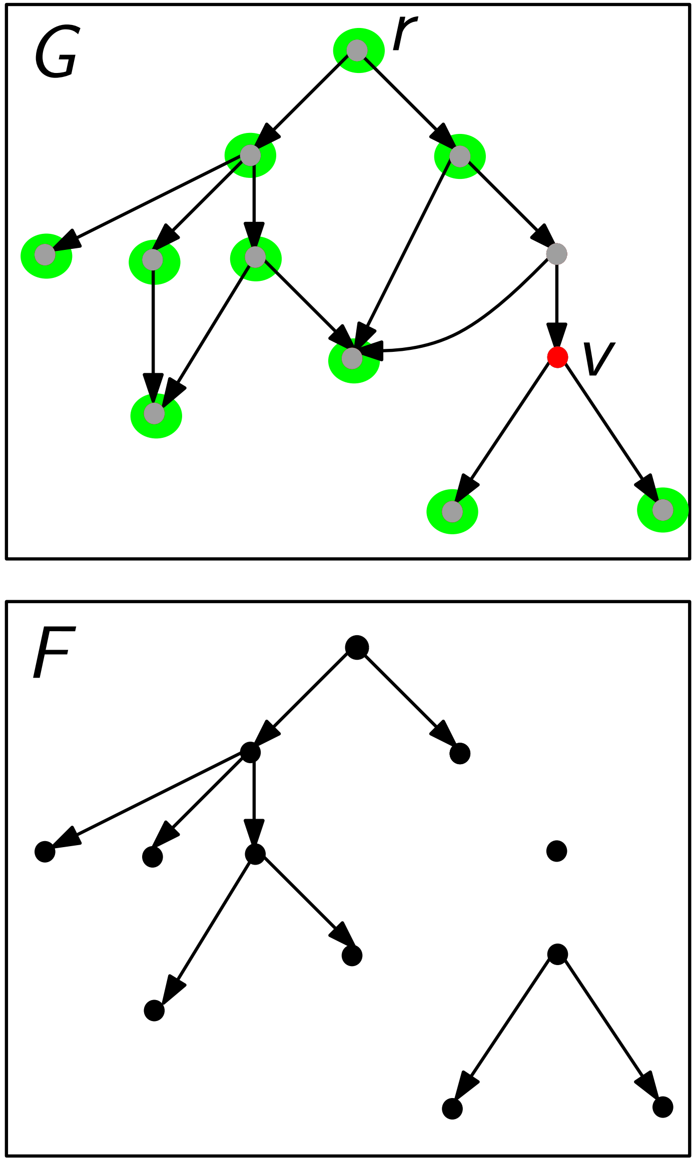
Input: acyclic digraph G with root r
Output: spanning tree T
mark r
$F \leftarrow \texttt{expand}(G)$
$T \leftarrow \texttt{connect}(G, F)$
return T





Procedure `connect`

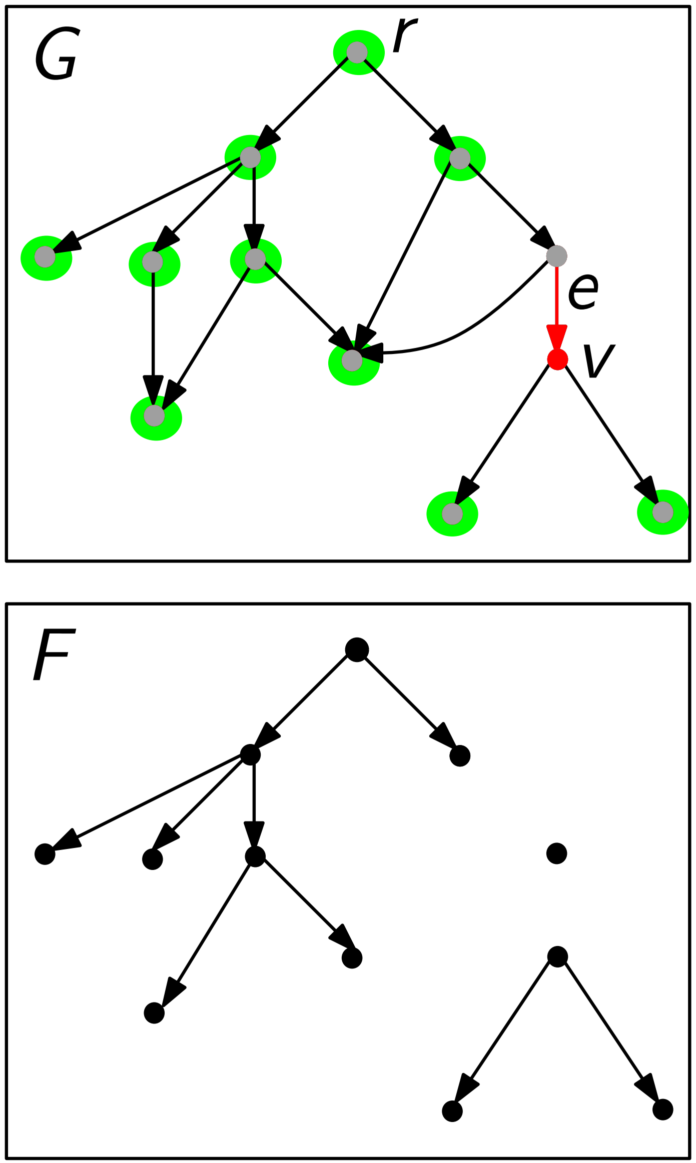
foreach unmarked node v **do**



Procedure `connect`

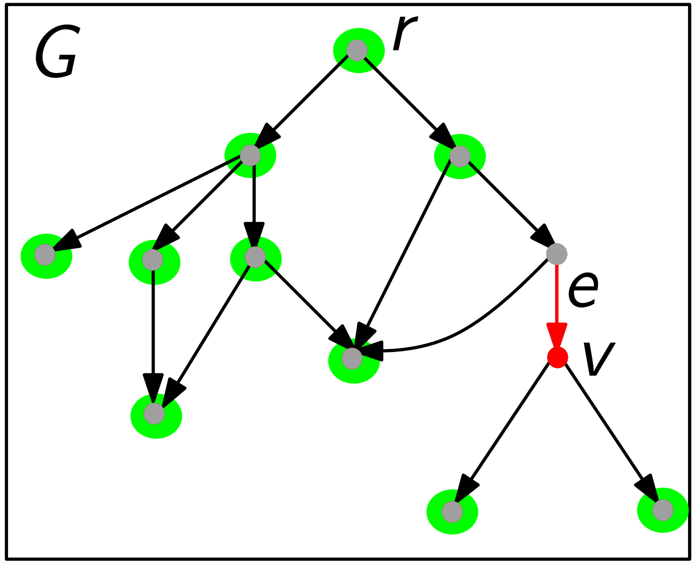
foreach unmarked node v **do**

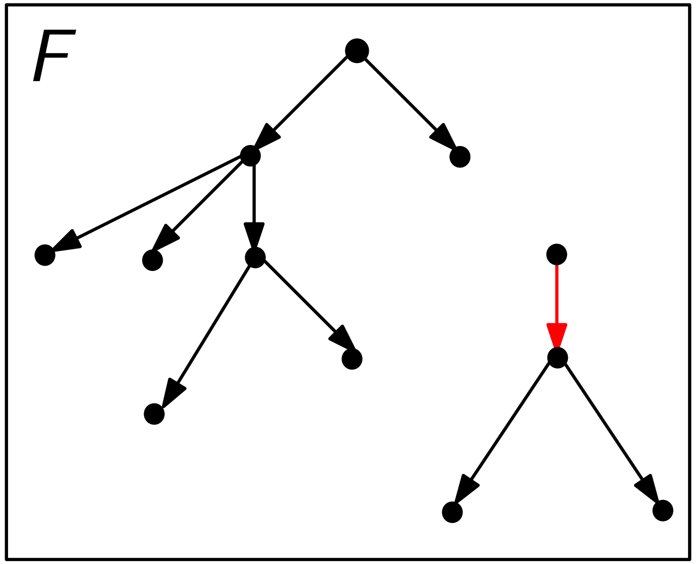
  choose arbitrary incoming edge e of v in G



Procedure `connect`

foreach unmarked node v **do**
- choose arbitrary incoming edge e of v in G
- $F \leftarrow F + e$





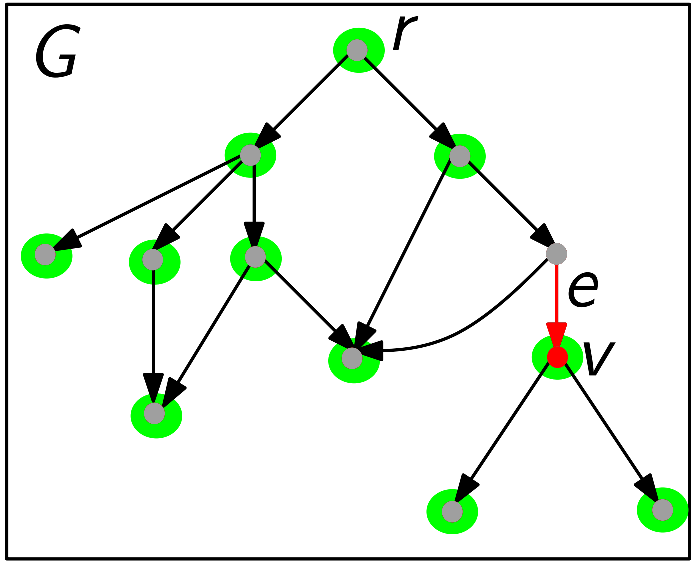
Procedure connect

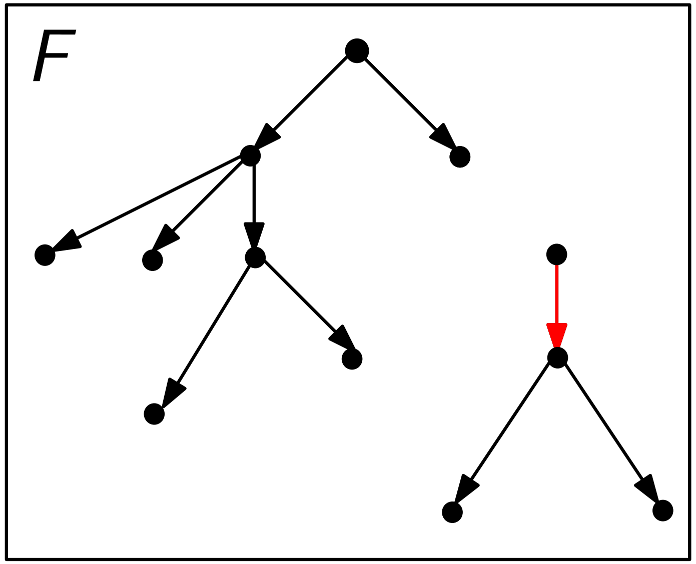
foreach unmarked node v **do**

  choose arbitrary incoming edge e of v in G

  $F \leftarrow F + e$

  mark v

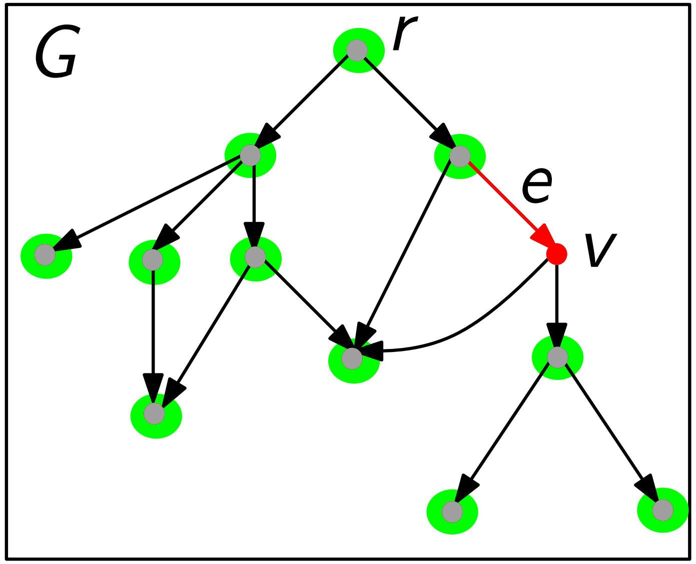


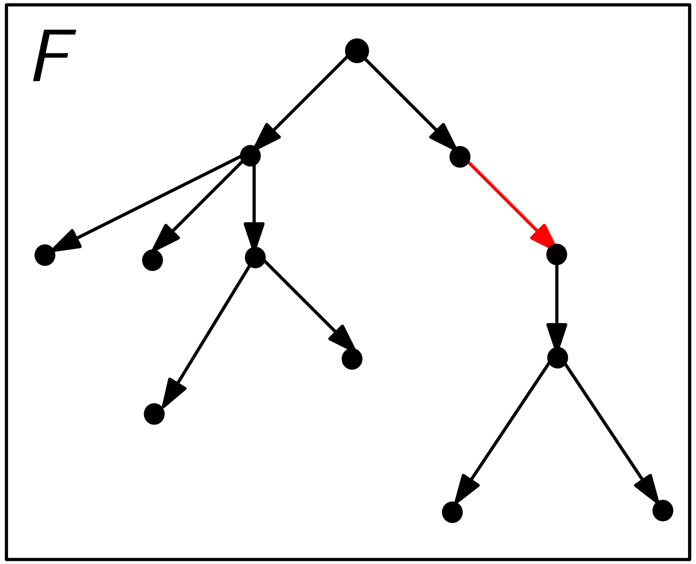


Procedure `connect`

foreach unmarked node v **do**

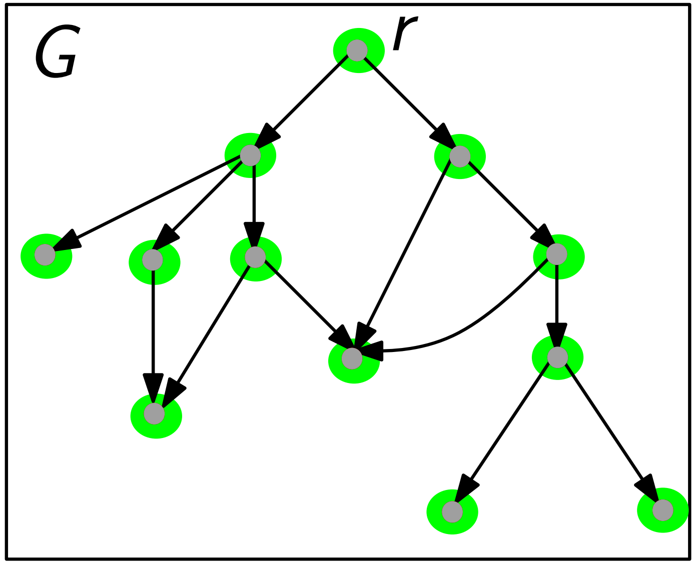
- choose arbitrary incoming edge e of v in G
- $F \leftarrow F + e$
- mark v

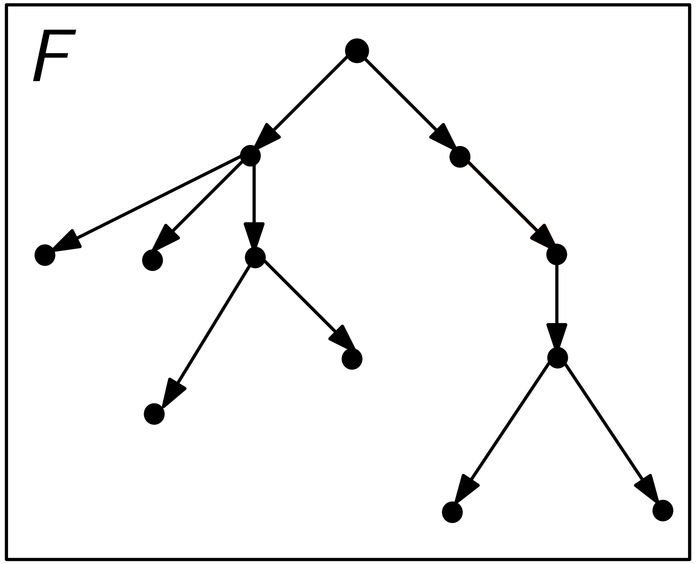




Procedure `connect`

foreach unmarked node v **do**
- choose arbitrary incoming edge e of v in G
- $F \leftarrow F + e$
- mark v





Correctness

Lemma. *Given an acyclic digraph G, the algorithm computes a spanning tree of G.*

Correctness

Lemma. *Given an acyclic digraph G, the algorithm computes a spanning tree of G.*

Observations.

- If a node is marked,
  - it has exactly one incoming edge in F (except r)

Correctness

Lemma. *Given an acyclic digraph G, the algorithm computes a spanning tree of G.*

Observations.

- If a node is marked,
  - – it has exactly one incoming edge in F (except r) and
  - – no further incoming edges are added.

Correctness

Lemma. *Given an acyclic digraph G, the algorithm computes a spanning tree of G.*

Observations.

- If a node is marked,
  - it has exactly one incoming edge in F (except r) and
  - no further incoming edges are added.
- At the end of the algorithm every node is marked.

Correctness

Lemma. *Given an acyclic digraph G, the algorithm computes a spanning tree of G.*

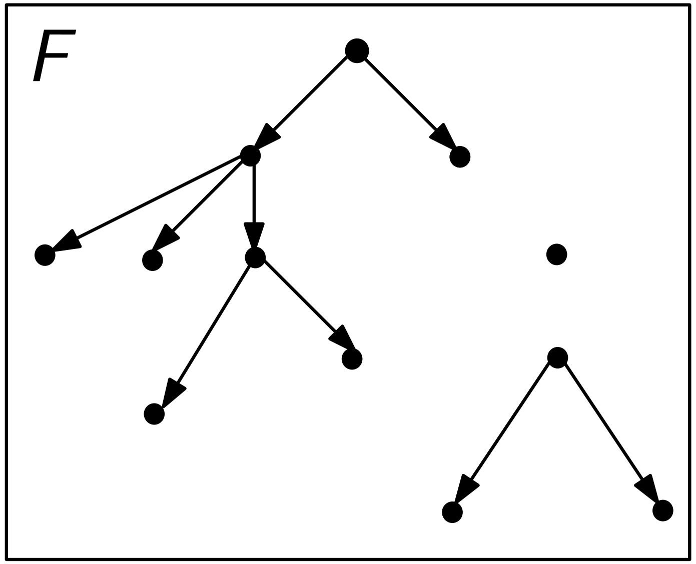
Observations.

- If a node is marked,
  - it has exactly one incoming edge in F (except r) and
  - no further incoming edges are added.
- At the end of the algorithm every node is marked.

$\Rightarrow$ spanning tree

Analysis

Observation. Procedure `expand` yields forest F.

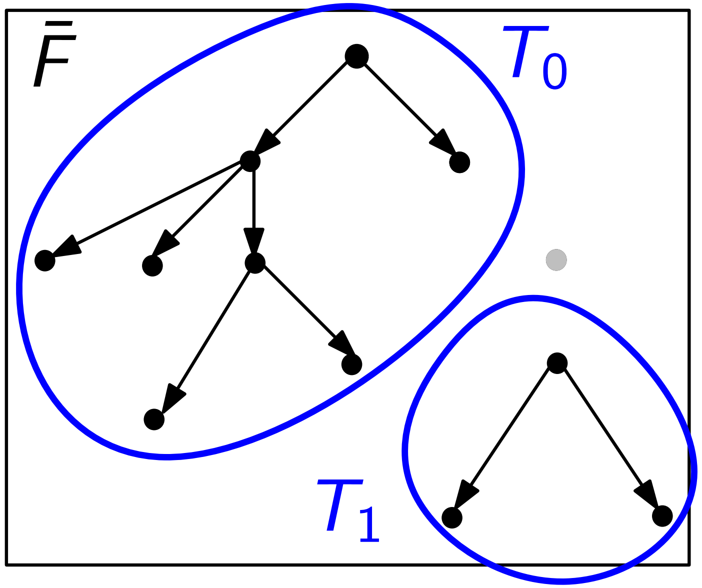


Analysis

Observation. Procedure `expand` yields forest F.

Def. Let

- $\bar{F} = F - \{\text{isolated nodes}\}$
  $= \{T_0, \ldots, T_k\}$,

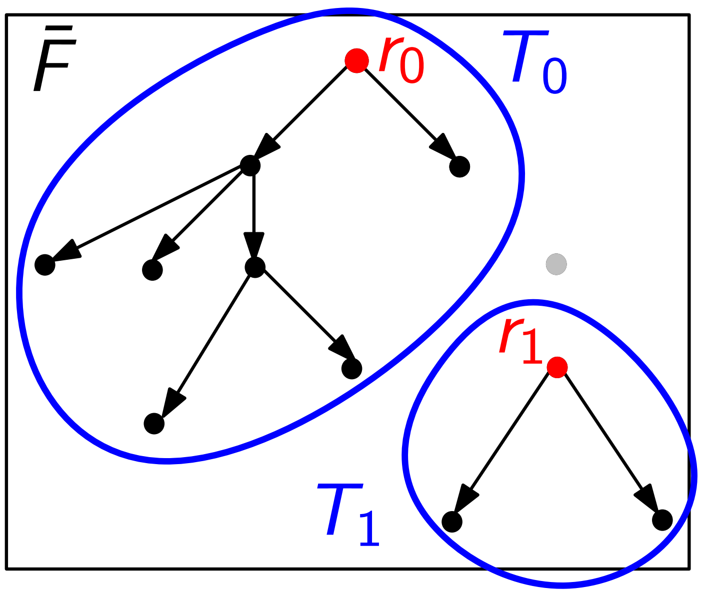


Analysis

Observation. Procedure `expand` yields forest F.

Def. Let

- $\bar{F} = F - \{\text{isolated nodes}\}$
  $= \{T_0, \ldots, T_k\}$,
- $r_i \leftarrow$ root of T_i

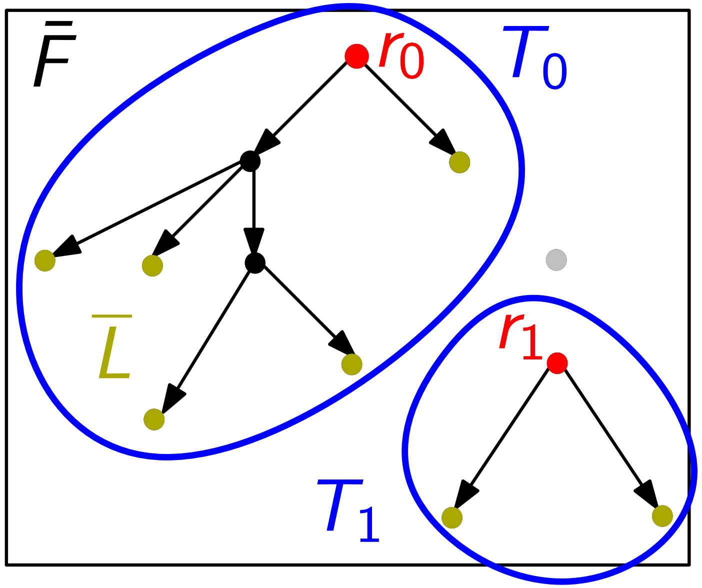


Analysis

Observation. Procedure `expand` yields forest F.

Def. Let

- $\bar{F} = F - \{\text{isolated nodes}\}$
  $= \{T_0, \dots, T_k\}$,
- $r_i \leftarrow$ root of T_i and
- $\bar{L} \leftarrow$ set of leaves in $\bar{F}$.



Analysis

Lemma 1. *For* $i = 0, \ldots, k$, *any subtree* $T_i \in \bar{F}$ *has at least* $(|V(T_i)| + 1)/2$ *leaves.*

Analysis

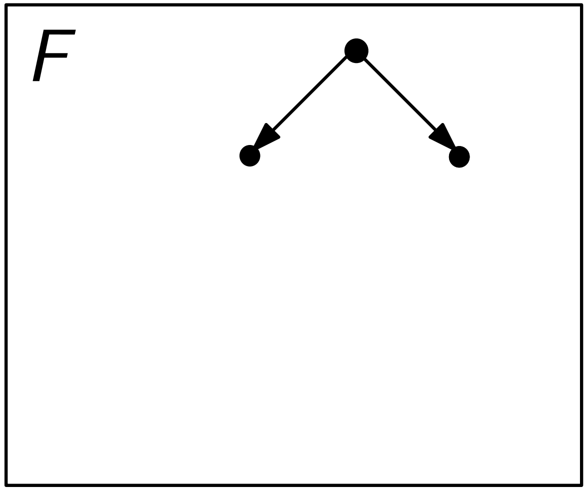
Lemma 1. *For* $i = 0, \ldots, k$*, any subtree* $T_i \in \bar{F}$ *has at least* $(|V(T_i)| + 1)/2$ *leaves.*

Expand F in appropriate order: top-down.

Analysis

Lemma 1. *For $i = 0, \ldots, k$, any subtree $T_i \in \bar{F}$ has at least $(|V(T_i)| + 1)/2$ leaves.*

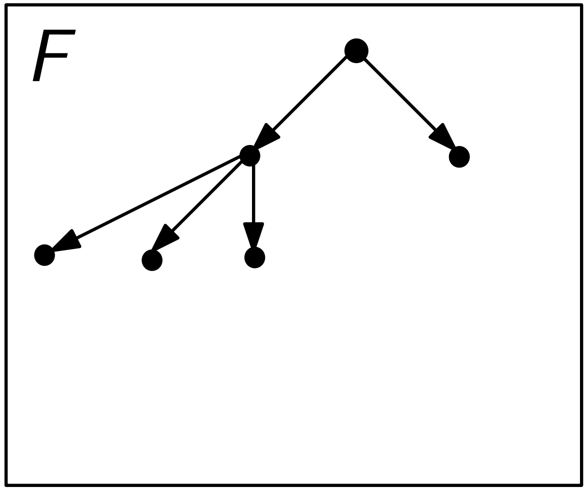
Expand F in appropriate order: top-down.



Analysis

Lemma 1. *For $i = 0, \ldots, k$, any subtree $T_i \in \bar{F}$ has at least $(|V(T_i)| + 1)/2$ leaves.*

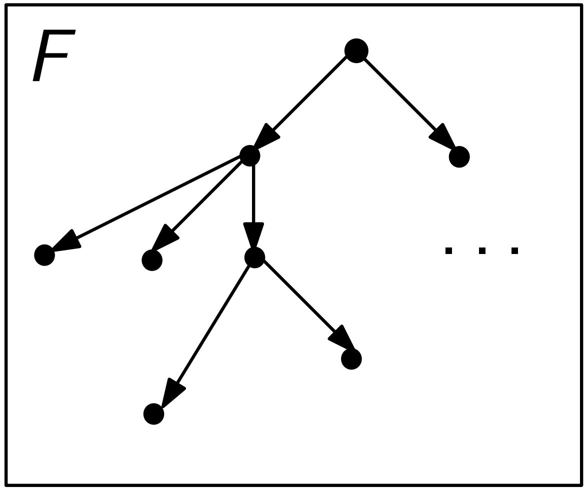
Expand F in appropriate order: top-down.



Analysis

Lemma 1. *For* $i = 0, \ldots, k$*, any subtree* $T_i \in \bar{F}$ *has at least* $(|V(T_i)| + 1)/2$ *leaves.*

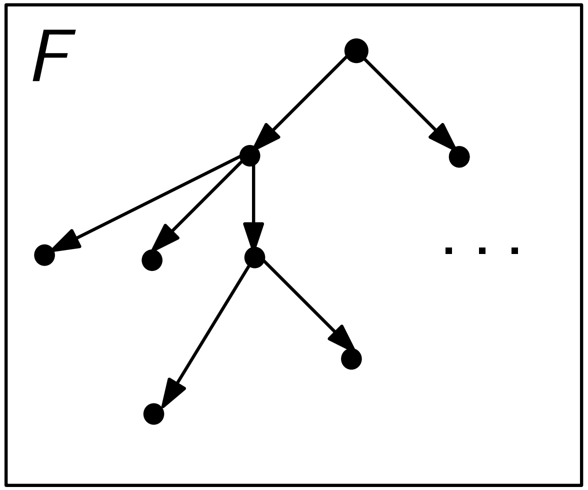
Expand F in appropriate order: top-down.



Analysis

Lemma 1. *For* $i = 0, \ldots, k$, *any subtree* $T_i \in \bar{F}$ *has at least* $(|V(T_i)| + 1)/2$ *leaves.*

Expand F in appropriate order: top-down.

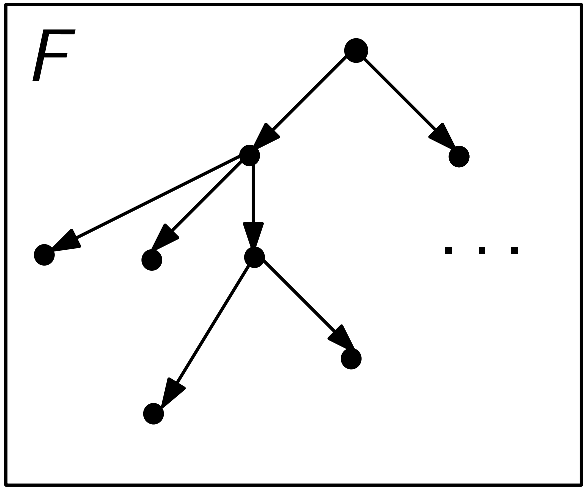


Every expansion

- creates at least two leaves and
- destroys exactly one leaf.

Analysis

Lemma 1. *For $i = 0, \ldots, k$, any subtree $T_i \in \bar{F}$ has at least $(|V(T_i)| + 1)/2$ leaves.*



Expand F in appropriate order: top-down.

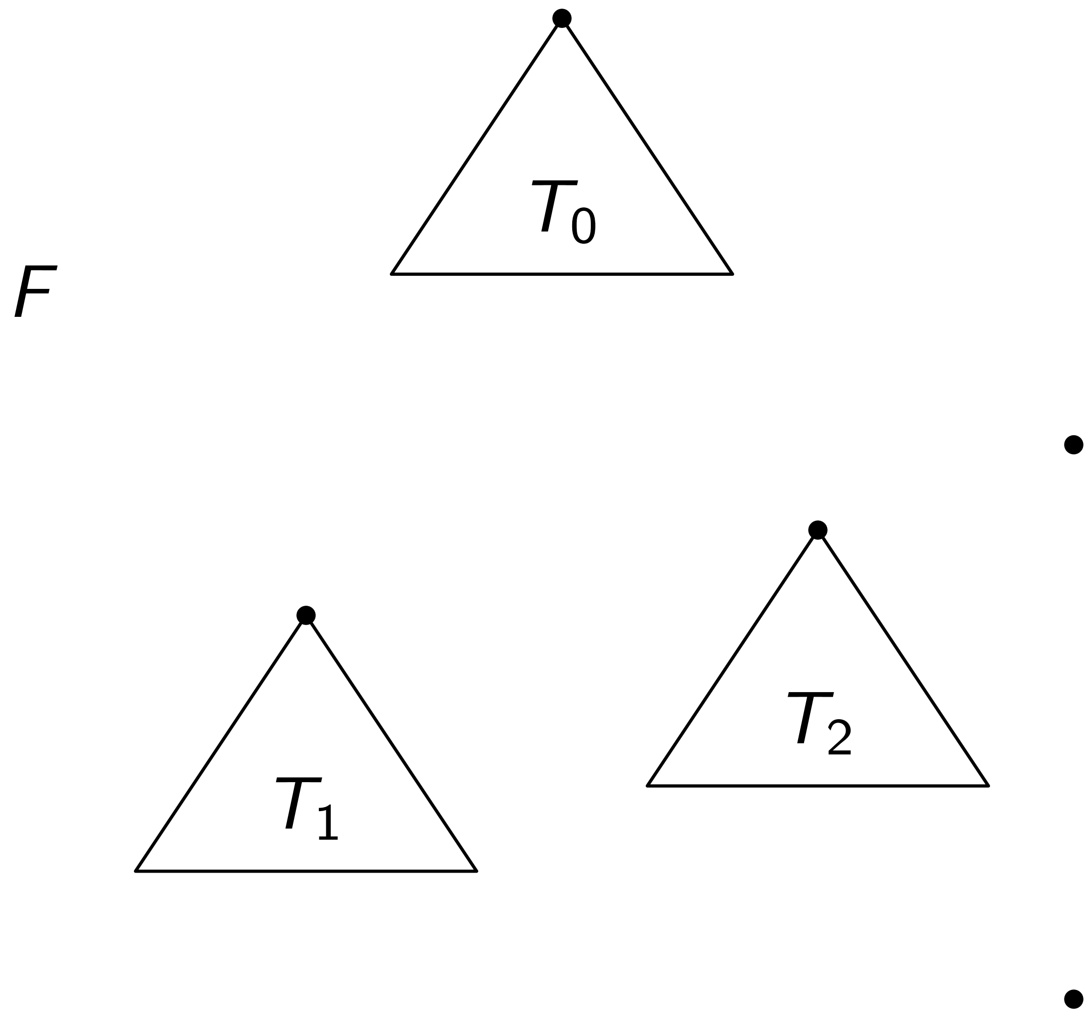
Every expansion

- creates at least two leaves and
- destroys exactly one leaf.

$\Rightarrow$ at least $|V(T_i)|/2$ leaves for each $T_i \in \bar{F}$

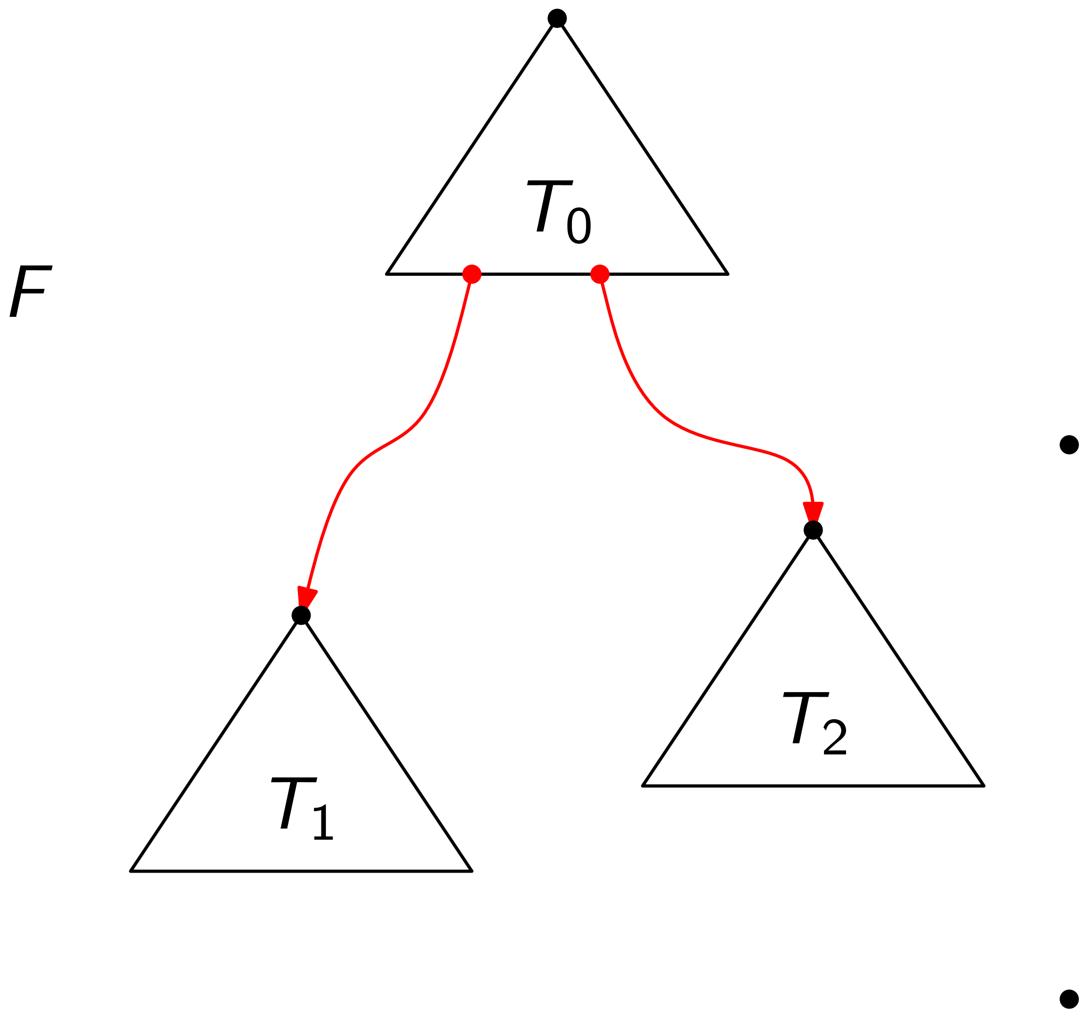
Analysis

Lemma 2. *Procedure* `connect` *creates a tree with at least* $|\bar{L}| - k$ *leaves.*



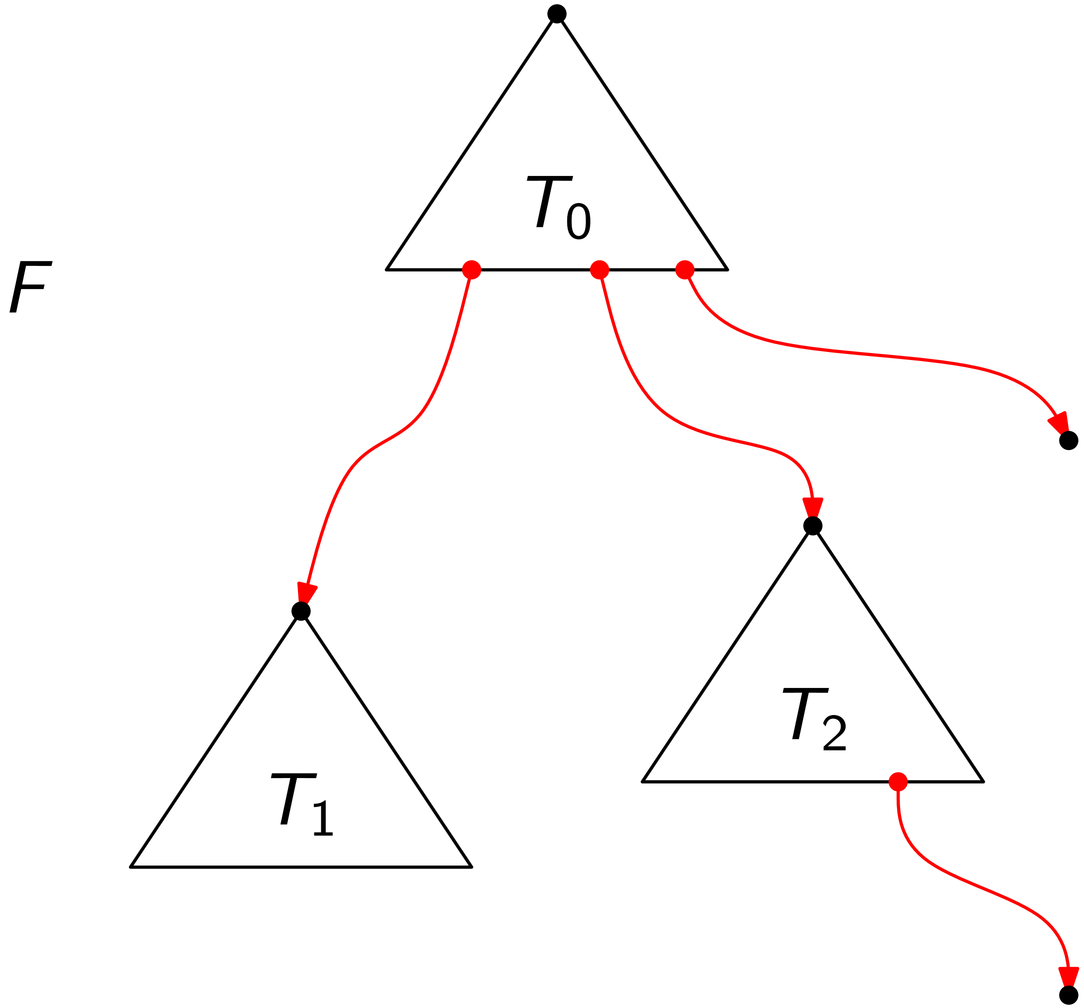
Analysis

Lemma 2. *Procedure* `connect` *creates a tree with at least* $|\bar{L}| - k$ *leaves.*



Analysis

Lemma 2. *Procedure* `connect` *creates a tree with at least* $|\bar{L}| - k$ *leaves.*



Analysis

Lemma 3. $OPT \leq |V(\bar{F})| - k$.

Analysis

Lemma 3. $OPT \leq |V(\bar{F})| - k.$

Let T^* be an optimum spanning tree.

Analysis

Lemma 3. $OPT \leq |V(\bar{F})| - k$.

Let T^* be an optimum spanning tree.

For

- each root r_i and
- each leaf ℓ of T^* outside of $V(\bar{F})$

identify a unique internal node in T^*.

Analysis

Lemma 3. $OPT \leq |V(\bar{F})| - k$.

Let T^* be an optimum spanning tree.

For

- each root r_i and
- each leaf ℓ of T^* outside of $V(\bar{F})$

identify a unique internal node in T^*.

$\Rightarrow$ exclusion of k nodes of $V(\bar{F})$

Analysis

Lemma 3. $OPT \leq |V(\bar{F})| - k$.

Let T^* be an optimum spanning tree.

For

- each root r_i and
- each leaf ℓ of T^* outside of $V(\bar{F})$

identify a unique internal node in T^*.

show the uniqueness

$\Rightarrow$ exclusion of k nodes of $V(\bar{F})$

Analysis

Take the first node in $V(\bar{F})$ on the way from r_i or ℓ to r.

Claim: This node is unique.

Analysis

Take the first node in $V(\bar{F})$ on the way from r_i or ℓ to r.
Claim: This node is unique.

Assume to the contrary that u is the unique node of v and w.
case 1: $u, v, w \in$ path

case 2: $u, v, w \notin$ path

,

Analysis

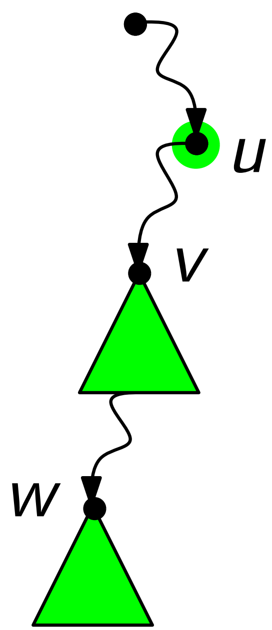
Take the first node in $V(\bar{F})$ on the way from r_i or ℓ to r.

Claim: This node is unique.

Assume to the contrary that u is the unique node of v and w.

case 1: $u, v, w \in$ path

case 2: $u, v, w \notin$ path



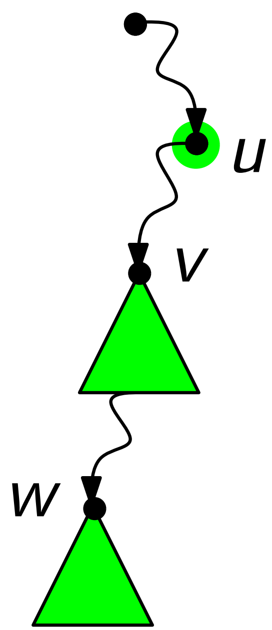
,

Analysis

Take the first node in $V(\bar{F})$ on the way from r_i or ℓ to r.
Claim: This node is unique.

Assume to the contrary that u is the unique node of v and w.
case 1: $u, v, w \in$ path

case 2: $u, v, w \notin$ path



$\lightning$ u is first node $\in V(\bar{F})$ for w

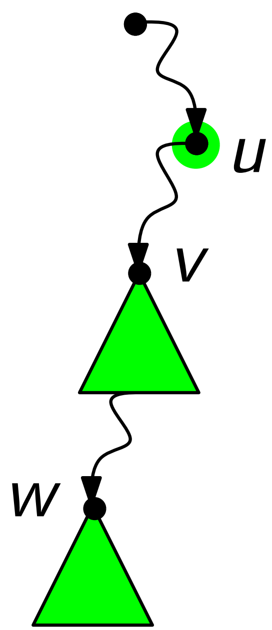
Analysis

Take the first node in $V(\bar{F})$ on the way from r_i or ℓ to r.

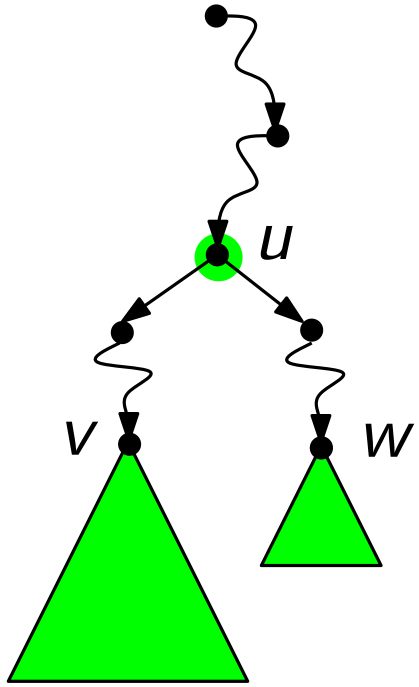
Claim: This node is unique.

Assume to the contrary that u is the unique node of v and w.

case 1: $u, v, w \in$ path



case 2: $u, v, w \notin$ path



$\lightning$ u is first node $\in V(\bar{F})$ for w

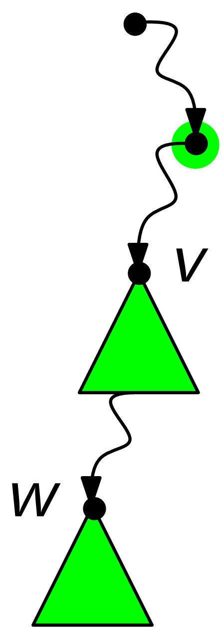
Analysis

Take the first node in $V(\bar{F})$ on the way from r_i or ℓ to r.

Claim: This node is unique.

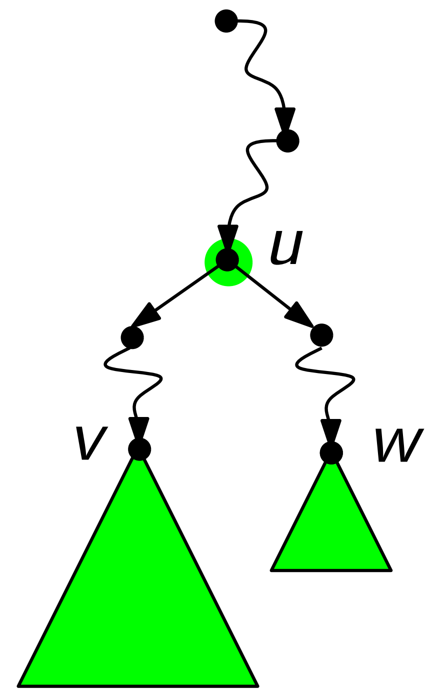
Assume to the contrary that u is the unique node of v and w.

case 1: $u, v, w \in$ path



$\lightning$ u is first node $\in V(\bar{F})$ for w

case 2: $u, v, w \notin$ path



$\lightning$ procedure `expand`

Analysis

Putting things together...

Lemma 1. *For $i = 0, \ldots, k$, any subtree $T_i \in \bar{F}$ has at least $(|V(T_i)| + 1)/2$ leaves.*

Lemma 2. *Procedure* `connect` *creates a tree with at least $|\bar{L}| - k$ leaves.*

Lemma 3. $OPT \leq |V(\bar{F})| - k$.

Analysis

Putting things together...

Lemma 1. *For $i = 0, \ldots, k$, any subtree $T_i \in \bar{F}$ has at least $(|V(T_i)| + 1)/2$ leaves.*

Lemma 2. *Procedure* `connect` *creates a tree with at least $|\bar{L}| - k$ leaves.*

$$\Rightarrow \geq |\bar{L}| - k \geq \sum_{i=0}^{k} \frac{|V(T_i)| + 1}{2} - k = \frac{|V(\bar{F})| - k}{2}$$

Lemma 3. $OPT \leq |V(\bar{F})| - k$.

Analysis

Putting things together...

Lemma 1. *For* $i = 0, \ldots, k$, *any subtree* $T_i \in \bar{F}$ *has at least* $(|V(T_i)| + 1)/2$ *leaves.*

Lemma 2. *Procedure* `connect` *creates a tree with at least* $|\bar{L}| - k$ *leaves.*

$$\Rightarrow \geq |\bar{L}| - k \geq \sum_{i=0}^{k} \frac{|V(T_i)| + 1}{2} - k = \frac{|V(\bar{F})| - k}{2}$$

Lemma 3. $OPT \leq |V(\bar{F})| - k$.

$\Rightarrow$ ratio 2

Summary and Open Questions

Summary.

- MaxSNP-hardness for DAGs

Summary and Open Questions

Summary.

- MaxSNP-hardness for DAGs
- linear-time 4- and 2-approximation algorithms for DAGs improving upon 92-approximation for general digraphs

Summary and Open Questions

Summary.

- MaxSNP-hardness for DAGs
- linear-time 4- and 2-approximation algorithms for DAGs improving upon 92-approximation for general digraphs

Open questions

- expansion-approach extendable to general digraphs?

Summary and Open Questions

Summary.

- MaxSNP-hardness for DAGs
- linear-time 4- and 2-approximation algorithms for DAGs improving upon 92-approximation for general digraphs

Open questions

- expansion-approach extendable to general digraphs?
- better results (DAGs or general) by multi-stage expansion?

Summary and Open Questions

Summary.

- MaxSNP-hardness for DAGs
- linear-time 4- and 2-approximation algorithms for DAGs improving upon 92-approximation for general digraphs

Open questions

- expansion-approach extendable to general digraphs?
- better results (DAGs or general) by multi-stage expansion?

Thank you!