Graph Algorithms 1

breadth first search (bfs) and depth first search (dfs)

definitions

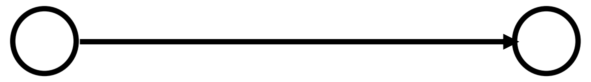
graph $G = (V, E)$

V: set of nodes/vertices

E : set of edges

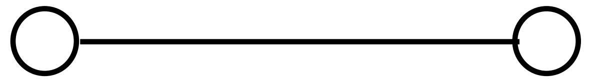
directed

$(u, v) \in E$ $\quad u, v \in V$



undirected

$\{u, v\} \in E$ $\quad u, v \in V$



3.4.1 Directed Graphs

In graph theory a *directed graph* G is specified by

- a set $G.V$ of *nodes*. Here we consider only finite graphs, thus $G.V$ is finite.
- a set $G.E \subseteq G.V \times G.V$ of *edges*. Edges $(u,v) \in G.E$ are depicted as arrows from node u to node v as shown in Fig. 1. For $(u,v) \in G.E$ one says that v is a *successor* of u and that u is a *predecessor* of v.

Fig. 1. Drawing an edge (u,v) from u to v

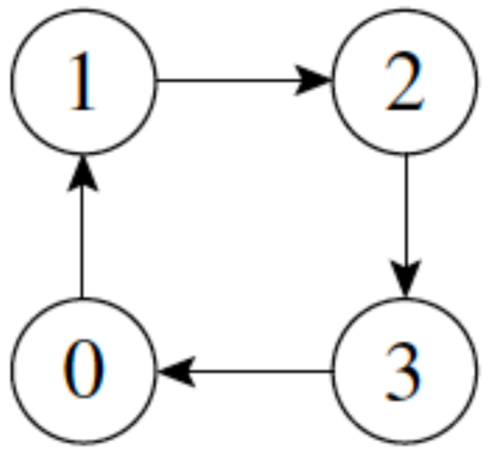


Fig. 2. Graph G

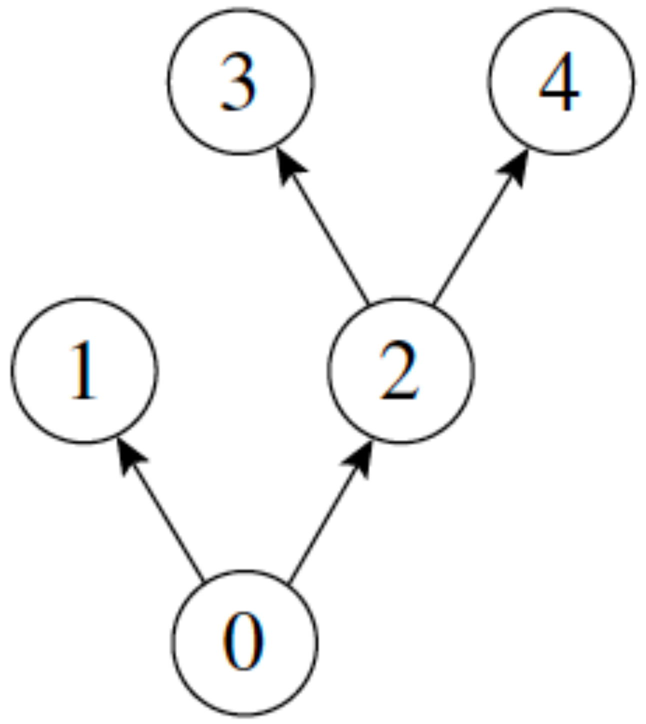


Fig. 3. Graph G'

If it is clear which graph G is meant, one abbreviates

$$\begin{aligned} V &= G.V, \\ E &= G.E. \end{aligned}$$

The graph G in Fig. 2 is formally described by

$$\begin{aligned} G.V &= \{0,1,2,3\}, \\ G.E &= \{(0,1),(1,2),(2,3),(3,0)\}. \end{aligned}$$

The graph G' in Fig. 3 is formally described by

$$\begin{aligned} G'.V &= \{0,1,2,3,4\}, \\ G'.E &= \{(0,1),(0,2),(2,3),(2,4)\}. \end{aligned}$$

definitions

graph $G = (V, E)$

V: set of nodes/vertices
E : set of edges

path $p = (v_0, \ldots, v_n)$ $\quad v_i \in V$

from v_0 to v_n of length n

cycle if $v_0 = v_n$

directed

$(u, v) \in E \quad u, v \in V$

$(v_i, v_{i+1}) \in E$

undirected

$\{u, v\} \in E \quad u, v \in V$

$\{v_i, v_{i+1}\} \in E$

data structures for representation of graphs

graph $G = (V, E)$

V: set of nodes/vertices

E : set of edges

path $p = (v_0, \ldots, v_\ell)$ $v_i \in V$

from v_0 to v_ℓ of length ℓ

cycle if $v_0 = v_n$

- adjacency matrix M

rename $V = [1{:}n]$

directed

$(u, v) \in E$ $u, v \in V$

$(v_i, v_{i+1}) \in E$

$M(i, j) = ((i, j) \in E ? 1 : 0)$

undirected

$\{u, v\} \in E$ $u, v \in V$

$\{v_i, v_{i+1}\} \in E$

$M(i, j) = (\{i, j\} \in E ? 1 : 0)$

symmetric

data structures for representation of graphs

directed

undirected

graph $G = (V, E)$

$(u, v) \in E \quad u, v \in V$

$\{u, v\} \in E \quad u, v \in V$

V: set of nodes/vertices

E : set of edges

$(v_i, v_{i+1}) \in E$

path $p = (v_0, \ldots, v_\ell)$ $\quad v_i \in V$

$\{v_i, v_{i+1}\} \in E$

from v_0 to v_ℓ of length ℓ

cycle if $v_0 = v_n$

- adjacency matrix M

$M(i, j) = ((i, j) =\in E ? 1 : 0)$

rename $V = [1 : n]$

$M(i, j) = (\{i, j\} =\in E ? 1 : 0)$

symmetric

- adjacency lists

$(u, v) \in E$

for each node u linked list $L(u)$ with nodes v with:

$\{u, v\} \in E$

array a with: $a[u]$ points to head of $L(u)$

data structures for representation of graphs

directed

$(u, v) \in E$ $\quad u, v \in V$

$(v_i, v_{i+1}) \in E$

$M(i, j) = ((i, j) =\in E ? 1 : 0)$

graph $G = (V, E)$

V: set of nodes/vertices
E : set of edges

path $p = (v_0, \dots, v_\ell)$ $\quad v_i \in V$

from v_0 to v_ℓ of length ℓ

cycle if $v_0 = v_n$

- adjacency matrix M

rename $V = [1 : n]$

space $O(\#V^2)$

undirected

$\{u, v\} \in E$ $\quad u, v \in V$

$\{v_i, v_{i+1}\} \in E$

$M(i, j) = (\{i, j\} =\in E ? 1 : 0)$

symmetric

data structures for representation of graphs

directed

$(u, v) \in E \quad u, v \in V$

undirected

$\{u, v\} \in E \quad u, v \in V$

graph $G = (V, E)$

V: set of nodes/vertices
E : set of edges

- adjacency lists

symmetric

$(u, v) \in E$

for each node u linked list $L(u)$ with nodes v with:

$\{u, v\} \in E$

array a with: $a[u]$ points to head of $L(u)$

space $O(\#V + \#E)$

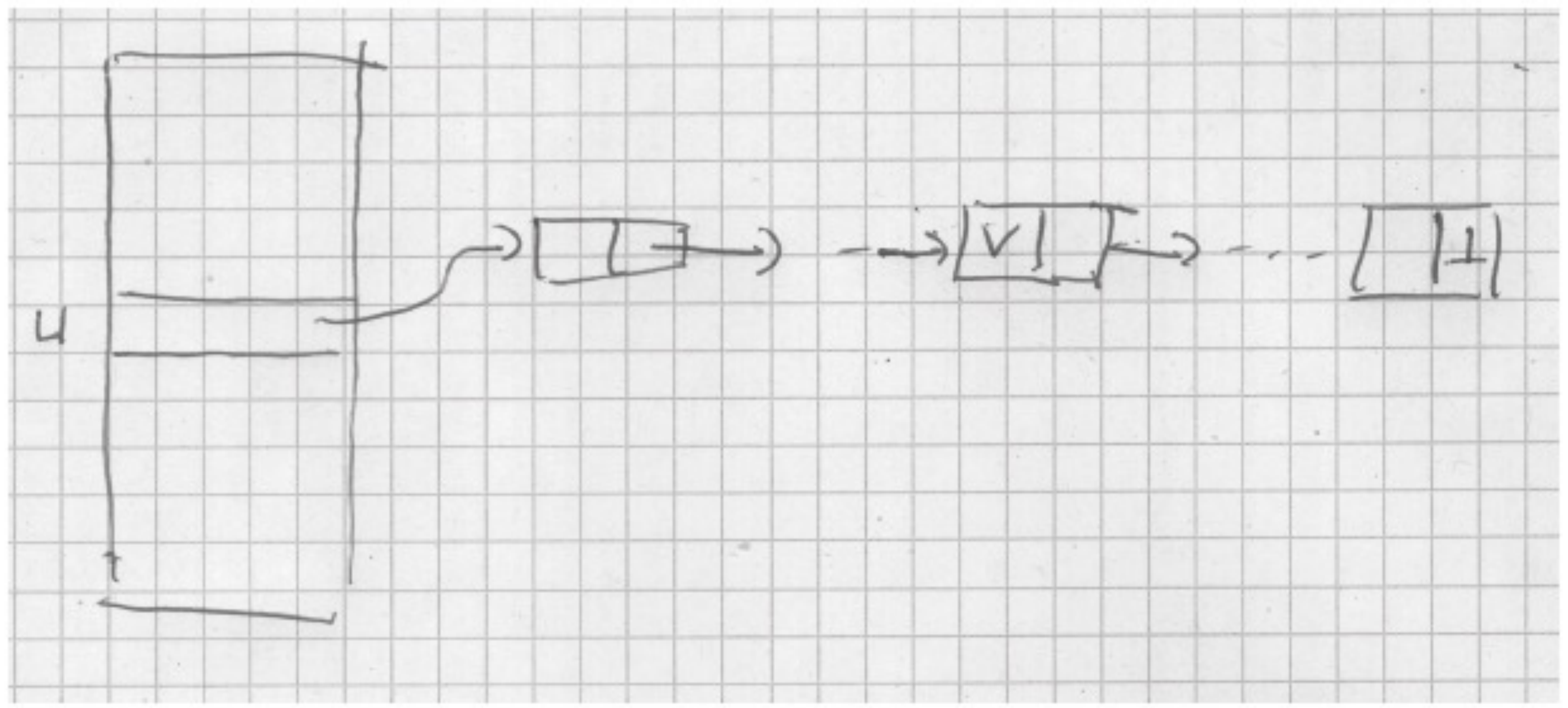
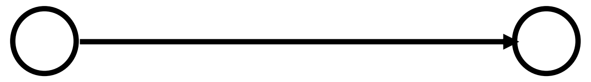


Figure 1: For each node $u \in V$ array element $a[u]$ points to the head of the adjacency list $L[u]$ of the nodes v with $(u, v) \in E$

reachability

directed

$(u, v) \in E$ $u, v \in V$



$(v_i, v_{i+1}) \in E$

graph $G = (V, E)$

V: set of nodes/vertices
E : set of edges

path $p = (v_0, \dots, v_n)$ $v_i \in V$

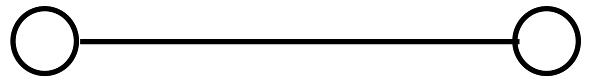
from v_0 to v_n of length n

cycle if $v_0 = v_n$

def: node v reachable from node u if there is a path prom u to v

undirected

$\{u, v\} \in E$ $u, v \in V$



$\{v_i, v_{i+1}\} \in E$

reachability

graph $G = (V, E)$

V: set of nodes/vertices
E : set of edges

path $p = (v_0, \dots, v_n)$ $\quad v_i \in V$

from v_0 to v_n of length n

cycle if $v_0 = v_n$

def $r(u, v)$: node v reachable from node u if there is a path prom u to v

equivalence classes

directed

$(u, v) \in E \quad u, v \in V$

$(v_i, v_{i+1}) \in E$

$r(u, v) \vee r(v, u)$
equivalence relation

strongly connected components

undirected

$\{u, v\} \in E \quad u, v \in V$

$\{v_i, v_{i+1}\} \in E$

$r(u, v)$
equivalence relation

connected components

searching the nodes reachable from $s \in V$

breadth first search (bfs)

$S_0 = \{s\}$ start set

searching the nodes reachable from $s \in V$

breadth first search (bfs)

$i = 0;$

$S_0 = \{s\}$ start set

/*suppose sets $S_0, \ldots S_i$ found*/

/*next round*/

$R_i = V \setminus (S_0 \cup \cdots \cup S_i)$ /*nodes not visited before round i*/

$i = i + 1;$

$S_{i+1} = \{v \in R_i | \exists u \in S_i . (u, v) \in E\}$

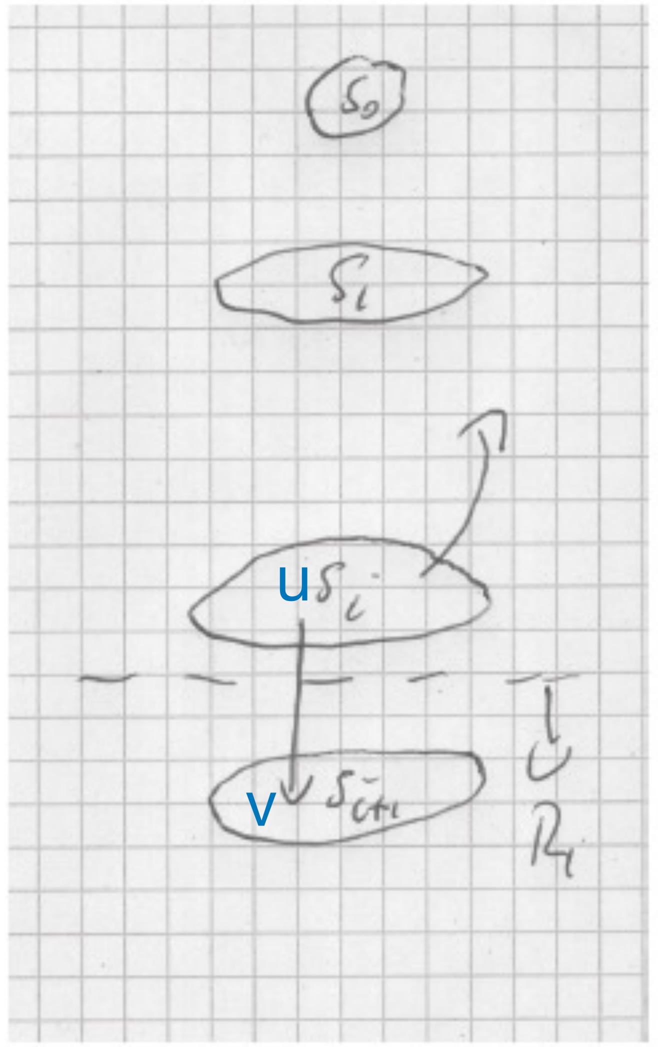


Figure 2: In round $i+1$ of breadth first search one collects the nodes v which are reachable from an node $u \in S_i$ by and edge (u, v) and which have not been visited before ($v \in R_i$) into the set S_{i+1}.

searching the nodes reachable from $s \in V$

breadth first search (bfs)

$i = 0;$

$S_0 = \{s\}$ start set

while $S_i \neq \emptyset$ {

/*suppose sets $S_0, \ldots S_i$ found*/

/*next round*/

$R_i = V \setminus (S_0 \cup \cdots \cup S_i)$ /*nodes not visited before round i*/

$i = i + 1;$

$S_{i+1} = \{v \in R_i | \exists u \in S_i. (u, v) \in E\}$ }

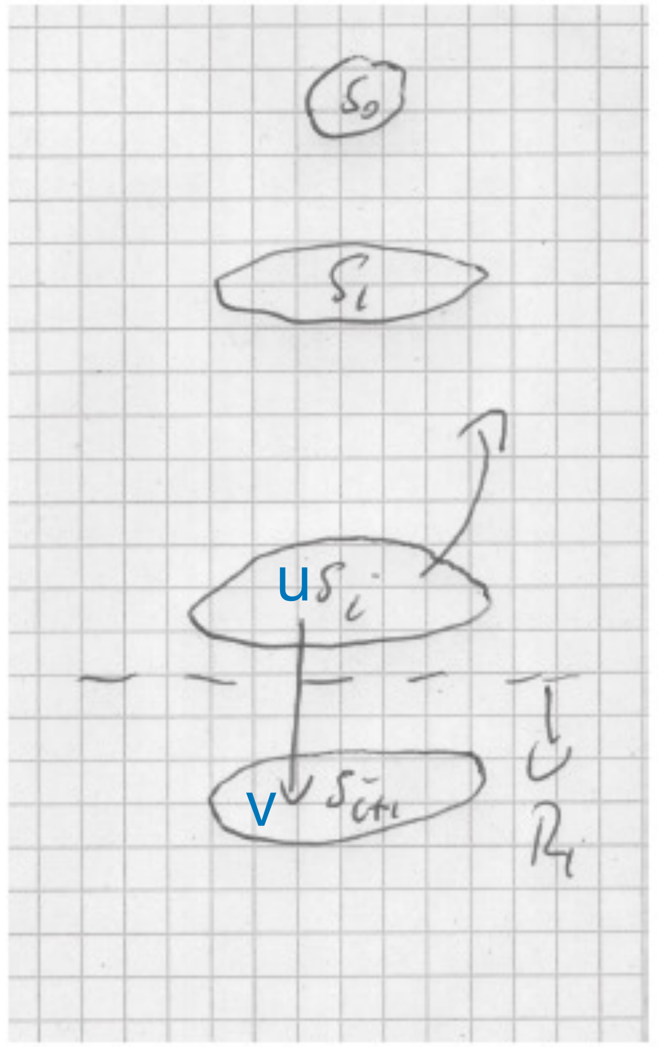


Figure 2: In round $i+1$ of breadth first search one collects the nodes v which are reachable from an node $u \in S_i$ by and edge (u, v) and which have not been visited before ($v \in R_i$) into the set S_{i+1}.

searching the nodes reachable from $s \in V$

$i = 0;$

$S_0 = \{s\}$ start set

while $S_i \neq \emptyset$ {

/*suppose sets $S_0, \ldots S_i$ found*/

/*next round*/

$R_i = V \setminus (S_0 \cup \cdots \cup S_i)$ /*nodes not visited before round i*/

$i = i + 1;$

$S_{i+1} = \{v \in R_i | \exists u \in S_i . (u, v) \in E\}$

distance

$d(s, v)$ =length of a shortest path from s to v

Lemma 1.

$$S_i = \{v \in V \mid d(s, v) = i\}$$

Proof. Induction on i. Trivial for $i = 0$.

searching the nodes reachable from $s \in V$

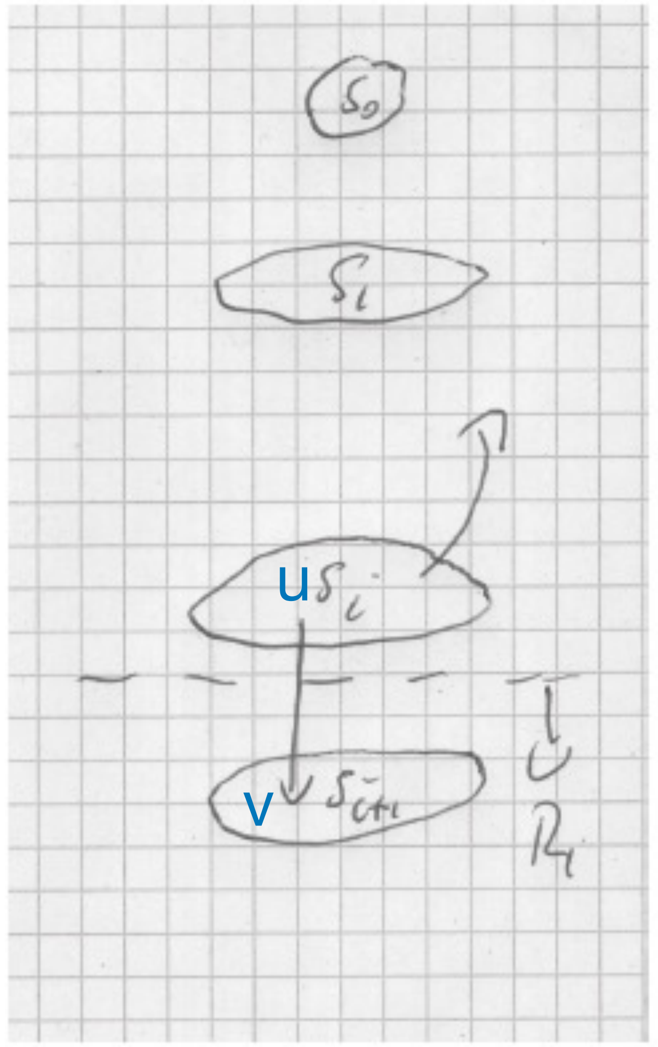


Figure 2: In round $i+1$ of breadth first search one collects the nodes v which are reachable from an node $u \in S_i$ by and edge (u,v) and which have not been visited before ($v \in R_i$) into the set S_{i+1}.

distance

$d(s,v)$ =length of a shortest path from s to v

Lemma 1.

$$S_i = \{v \in V \mid d(s,v) = i\}$$

Proof. Induction on i. Trivial for $i = 0$.

$i \to i+1$.

- $\subseteq$:

  Let $v \in S_{i+1}$.

$$\to (u,v) \in E \text{ for some } u \in S_i$$
$$d(s,u) = i \quad \text{(induction hypothesis)}$$
$$\to d(s,v) \le d(s,u) + 1 = i + 1$$

searching the nodes reachable from $s \in V$

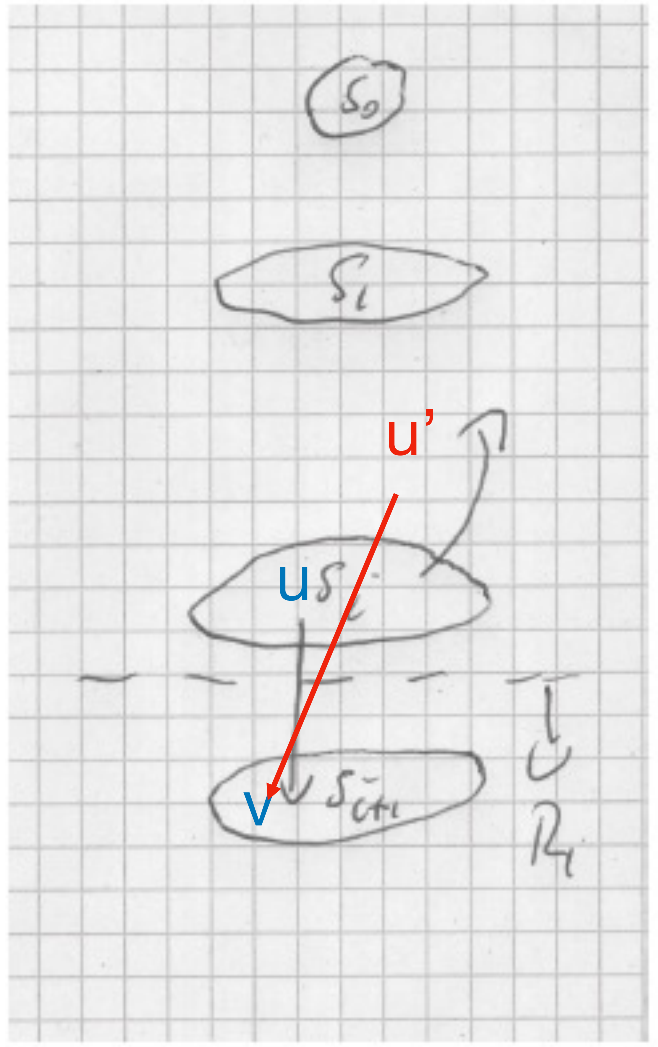


Figure 2: In round $i+1$ of breadth first search one collects the nodes v which are reachable from an node $u \in S_i$ by and edge (u,v) and which have not been visited before ($v \in R_i$) into the set S_{i+1}.

distance

$d(s,v)$ =length of a shortest path from s to v

Lemma 1.

$$S_i = \{v \in V \mid d(s,v) = i\}$$

Proof. Induction on i. Trivial for $i = 0$.

$i \to i+1$.

- $\subseteq$:

Let $v \in S_{i+1}$.

$$\to (u,v) \in E \text{ for some } u \in S_i$$
$$d(s,u) = i \quad \text{(induction hypothesis)}$$
$$\to d(s,v) \le d(s,u) + 1 = i + 1$$

Assume $d(s,v) = j \le i$

$$\to \exists u' : d(s,u') = j-1 \wedge (u',v) \in E$$
$$u' \in S_{j-1} \quad \text{(induction hypothesis)}$$
$$\to v \in S_j \ , \ j \le i \quad \text{would have been found earlier}$$

searching the nodes reachable from $s \in V$

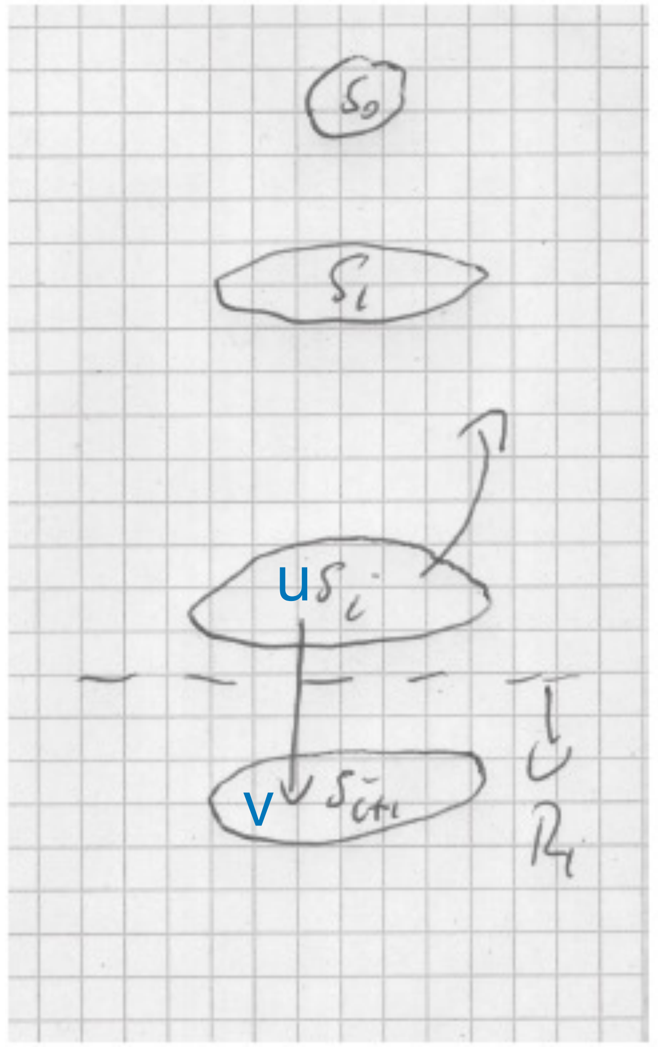


Figure 2: In round $i+1$ of breadth first search one collects the nodes v which are reachable from an node $u \in S_i$ by and edge (u,v) and which have not been visited before ($v \in R_i$) into the set S_{i+1}.

distance

$d(s,v)$ =length of a shortest path from s to v

Lemma 1.

$$S_i = \{v \in V \mid d(s,v) = i\}$$

Proof. Induction on i. Trivial for $i = 0$.

$i \to i+1$.

- $\subseteq$:

Let $v \in S_{i+1}$.

$$\to (u,v) \in E \text{ for some } u \in S_i$$

$$d(s,u) = i \quad \text{(induction hypothesis)}$$

$$\to d(s,v) \le d(s,u) + 1 = i + 1$$

Assume $d(s,v) = j \le i$

$$\to \exists u' : d(s,u') = j - 1 \wedge (u',v) \in E$$

$$u' \in S_{j-1} \quad \text{(induction hypothesis)}$$

$$\to v \in S_j \,,\ j \le i \quad \text{would have been found earlier}$$

- $\supseteq$: let $d(s,v) = i+1$

$$\exists u.\ d(s,u) = i \wedge (u,v) \in E$$

$$\to u \in S_i \quad \text{(induction hypothesis)}$$

$$\to v \in S_{i+1}$$

searching the nodes reachable from $s \in V$

v start node

$dfs(v)$:

depth first search (dfs)

here: for undirected graph

$\lambda: V \to \{new, \overset{old}{\cancel{visited}}\}$ labelling of nodes

store in array $b[n]$

initially $\lambda(v) = new$ for all v

searching the nodes reachable from $s \in V$

$dfs(v)$:

$\{\lambda(v) = new\}$

$\lambda(v) = old;$

for all $w \in L[v]$

{ if $\lambda(w) = new$ {dfs(w} }

v start node

precondition

depth first search (dfs)

here: for undirected graph

$\lambda: V \rightarrow \{new, \overset{old}{\cancel{visited}}\}$ labelling of nodes

store in array $b[n]$

initially $\lambda(v) = new$ for all v

searching the nodes reachable from $s \in V$

$dfs(v)$: v start node

$\{\lambda(v) = new\}$ precondition

$\lambda(v) = old$;

for all $w \in L[v]$

{ if $\lambda(w) = new$ {dfs(w)} }

depth first search (dfs)

here: for undirected graph

$\lambda : V \to \{new, \cancel{visited}\}$ old labelling of nodes

store in array $b[n]$

initially $\lambda(v) = new$ for all v

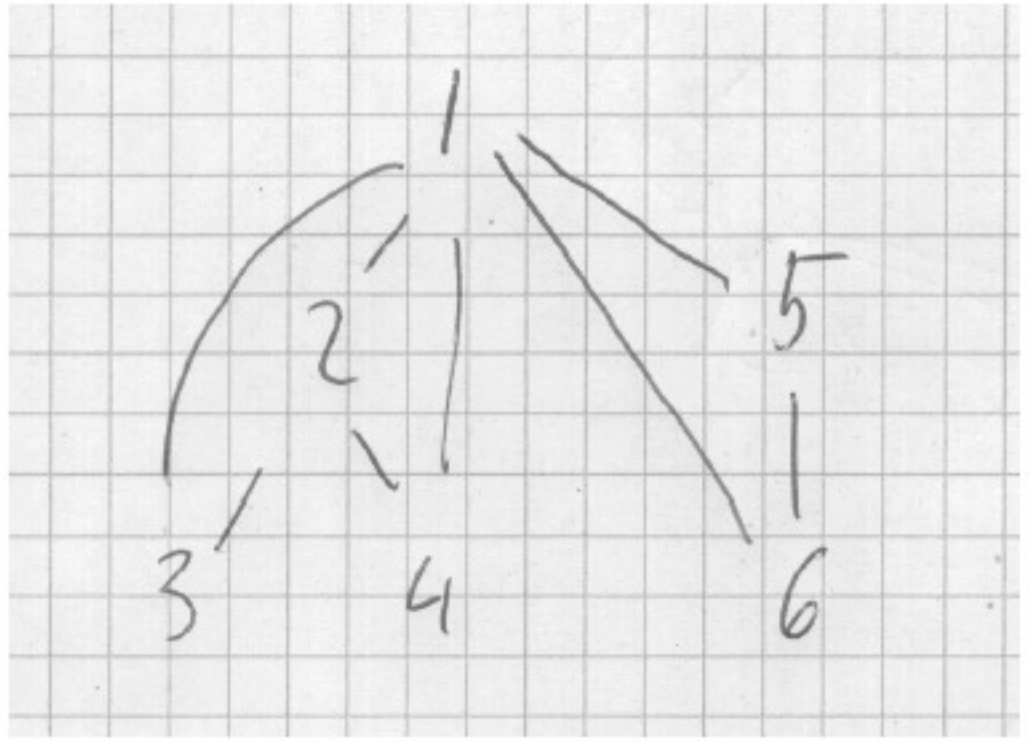


Figure 3: with edges $\{1,2\},\{1,3\},\{1,4\},\{1,5\},\{1,6\},\{2,3\},\{2,4\},\{5,6\}$ depth first search visits nodes in the order $1,2,3,4,5,6$.

spanning trees

here: undirected graphs

undirected graph $G = (V, E)$ is called *connected* if r(u,v) (reachable) for all u,v, in V

an undirected graph is called a *tree*, if it is connected and cycle free

a *spanning tree* of $G = (V, E)$ is a tree (V, T) with $T \subseteq E$

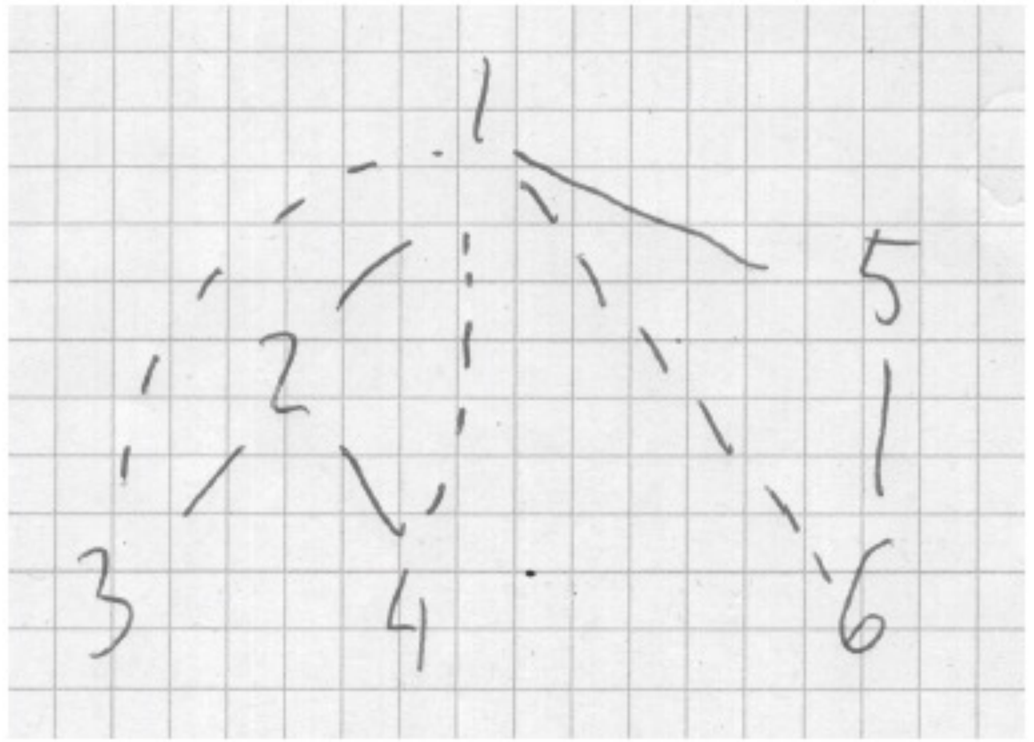


Figure 4: the edges drawn with solid lines form a spanning tree.

constructing a spanning tree (U, T) for the connected component of $s \in V$

initially $U = \{s\}, T = \emptyset$

dfs(v): v start node

$\{\lambda(v) = new\}$ precondition

$\lambda(v) = old;$

for all $w \in L[v]$

{ if $\lambda(w) = new$ $\{U = U \cup \{w\}; T = T \cup \{v, w\}; dfs(w)$ }

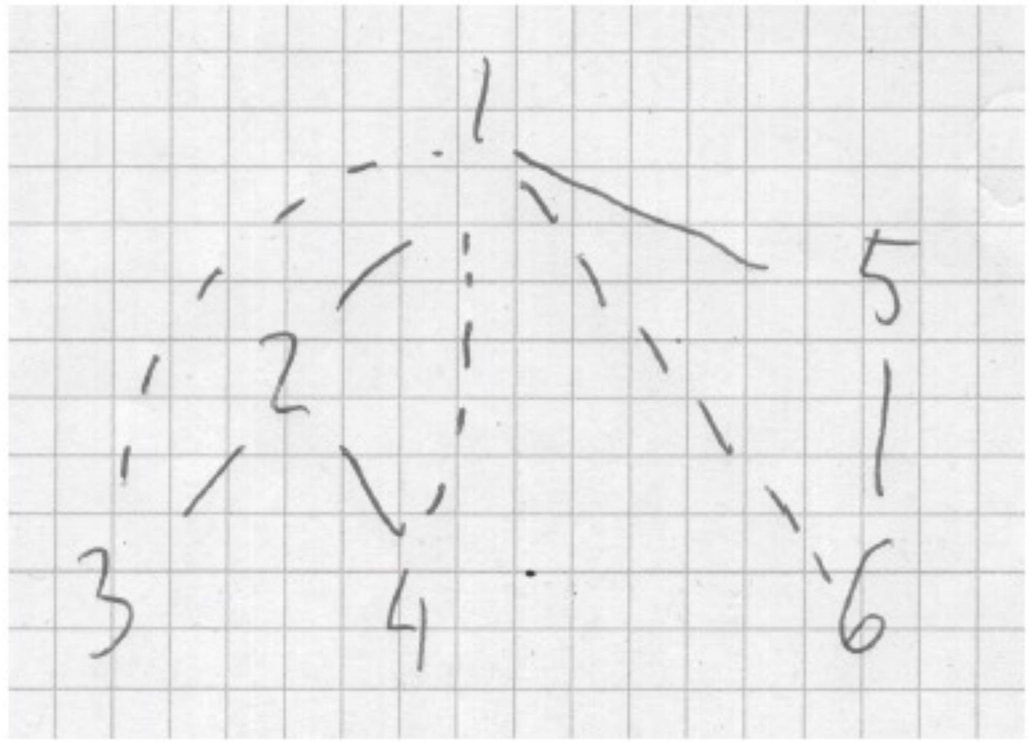


Figure 4: the edges drawn with solid lines form a spanning tree.

back edges and cross edges:

initially $U = \{s\}, T = \emptyset$

dfs(v): v start node

$\{\lambda(v) = new\}$ precondition

$\lambda(v) = old;$

for all $w \in L[v]$

{ if $\lambda(w) = new$ $\{U = U \cup \{w\}; T = T \cup \{v, w\}; dfs(w)$ }

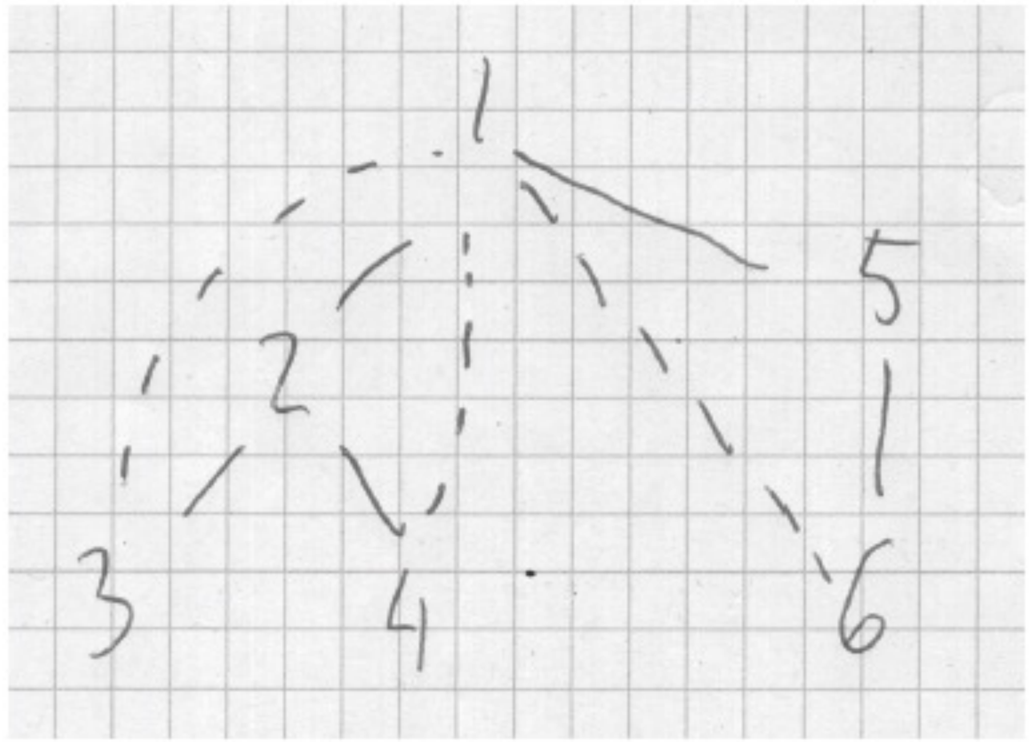


Figure 4: the edges drawn with solid lines form a spanning tree.

back edges and cross edges:

undirected graph $G = (V, E)$ is called *connected* if r(u,v) (reachable) for all u,v, in V

an undirected graph is called a *tree*, if it is connected and cycle free

a *spanning tree* of $G = (V, E)$ is a tree (V, T) with $T \subseteq E$

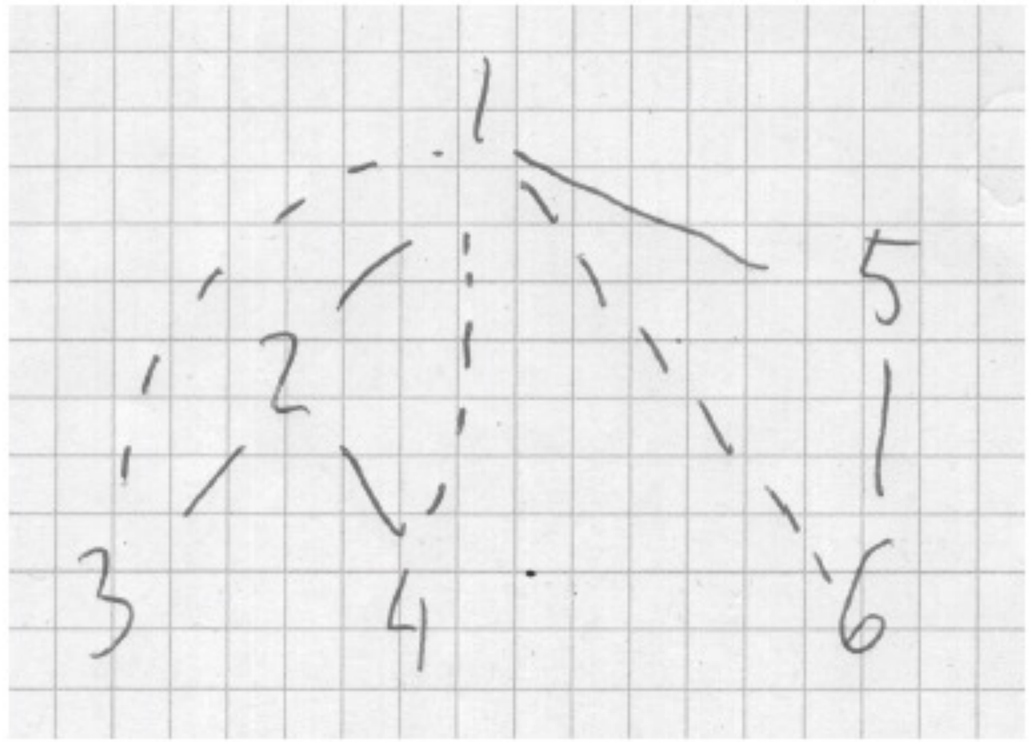


Figure 4: the edges drawn with solid lines form a spanning tree.

$$E = T \,\dot\cup\, B \,\dot\cup\, C$$

- T: tree edges
- B: back edges
- C: cross edges

$\{u, v\} \in E \setminus T$ is a *back edge* if there is a path in T from the root s touching both u and v

back edges and cross edges:

- T: tree edges
- B: back edges
- C: cross edges

undirected graph $G = (V, E)$ is called *connected* if r(u,v) (reachable) for all u,v, in V

an undirected graph is called a *tree*, if it is connected and cycle free

a *spanning tree* of $G = (V, E)$ is a tree (V, T) with $T \subseteq E$

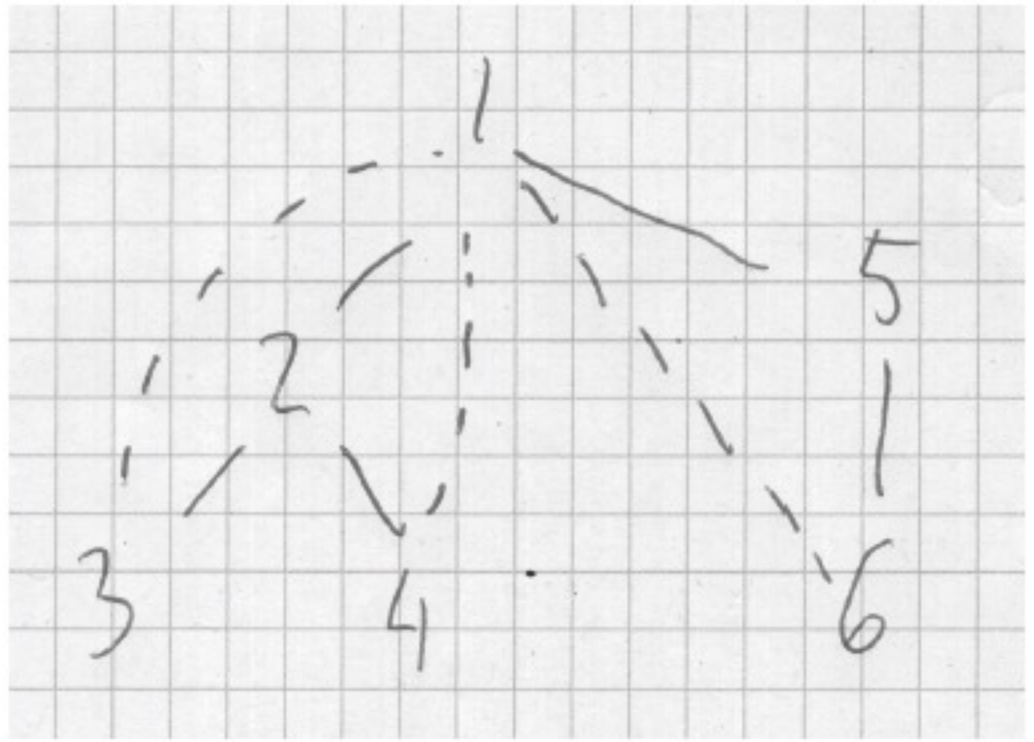


Figure 4: the edges drawn with solid lines form a spanning tree.

dotted lines are back edges

$$E = T \dot{\cup} B \dot{\cup} C$$

$\{u, v\} \in E \setminus T$ is a *back edge*
if there is a path in T from the root s touching both u and v

back edges and cross edges:

undirected graph $G = (V, E)$ is called *connected* if r(u,v) (reachable) for all u,v, in V

an undirected graph is called a *tree*, if it is connected and cycle free

a *spanning tree* of $G = (V, E)$ is a tree (V, T) with $T \subseteq E$

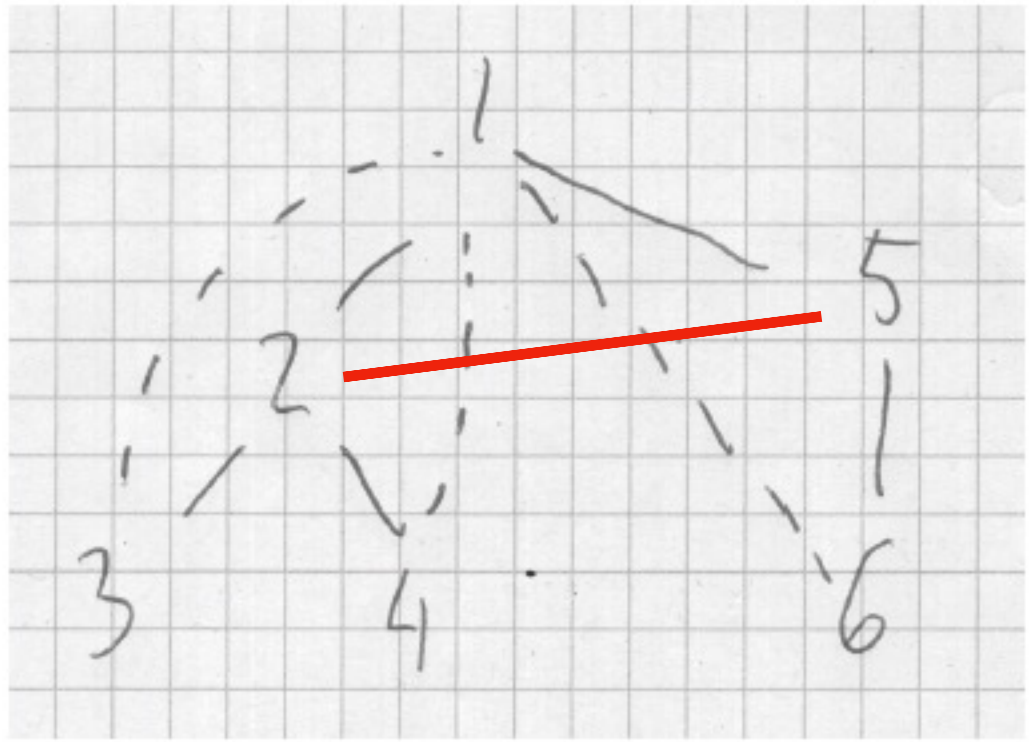


Figure 4: the edges drawn with solid lines form a spanning tree.

dotted lines are back edges

red line (if present) would be a cross edge

$$E = T \dot{\cup} B \dot{\cup} C$$

- T: tree edges
- B: back edges
- C: cross edges

$\{u, v\} \in E \setminus T$ is a *back edge* if there is a path in T from the root s touching both u and v

back edges and cross edges:

undirected graph $G = (V, E)$ is called *connected* if r(u,v) (reachable) for all u,v, in V

an undirected graph is called a *tree*, if it is connected and cycle free

a *spanning tree* of $G = (V, E)$ is a tree (V, T) with $T \subseteq E$

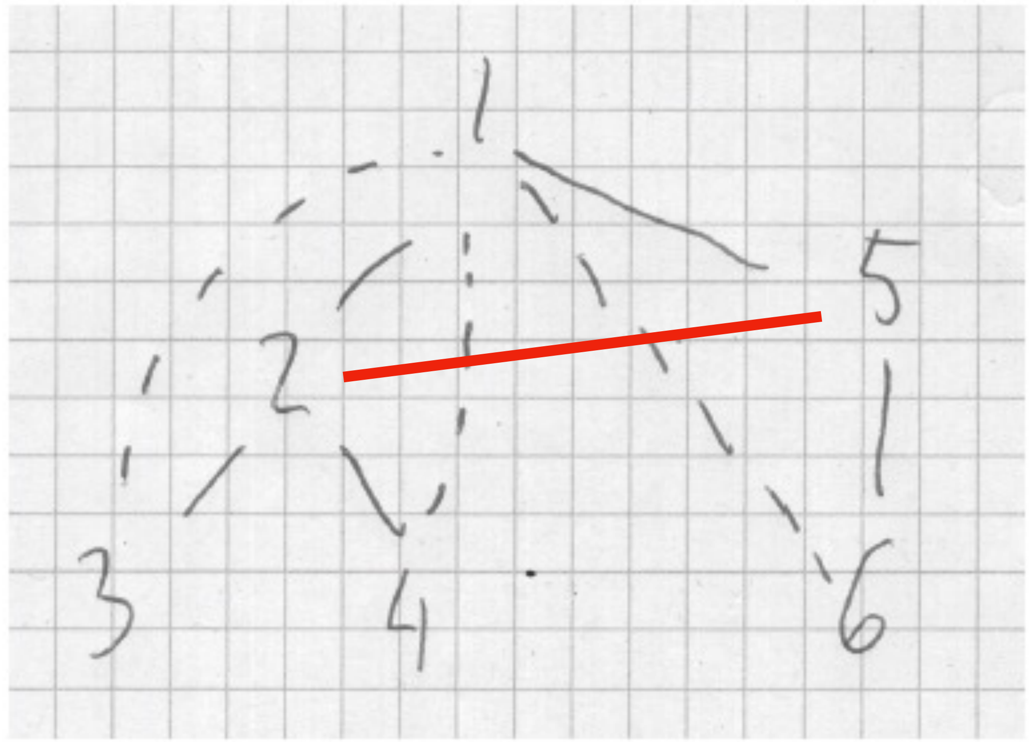


Figure 4: the edges drawn with solid lines form a spanning tree.

dotted lines are back edges

red line (if present) would be a cross edge

$$E = T \,\dot\cup\, B \,\dot\cup\, C$$

- T: tree edges
- B: back edges
- C: cross edges

$\{u, v\} \in E \setminus T$ is a *back edge* if there is a path in T from the root s touching both u and v

$\{u,v\}$ cross if u not neighbor of v in T.

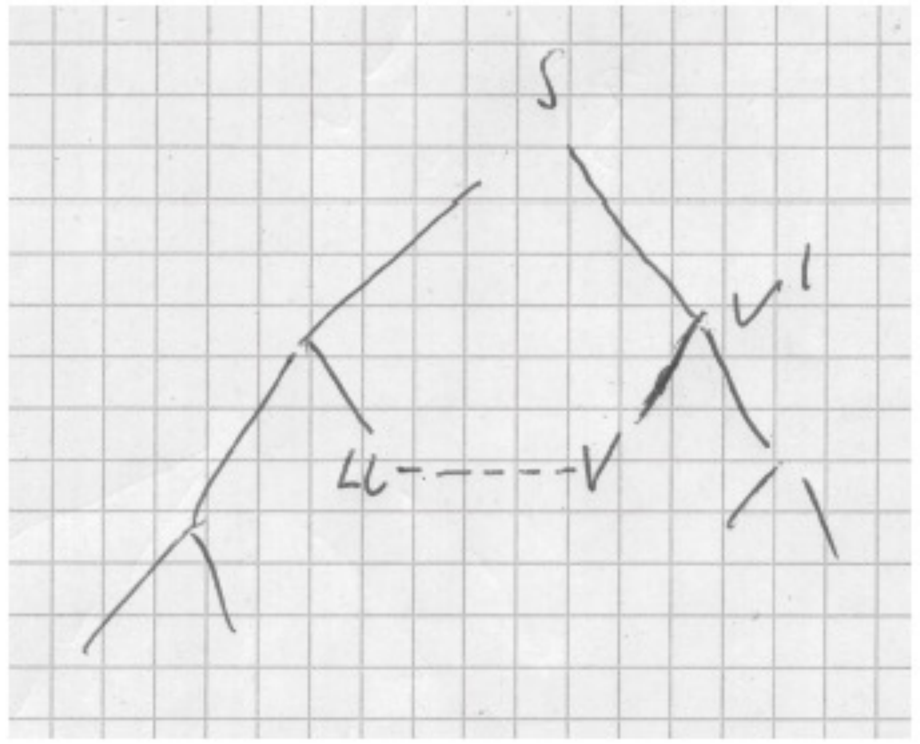
Lemma 2. *If a spanning tree is constructed by depth first search, then there are no cross edges.*

back edges and cross edges:

undirected graph $G = (V, E)$ is called *connected* if r(u,v) (reachable) for all u,v, in V

an undirected graph is called a *tree*, if it is connected and cycle free

a *spanning tree* of $G = (V, E)$ is a tree (V, T) with $T \subseteq E$



$$E = T \dot{\cup} B \dot{\cup} C$$

- T: tree edges
- B: back edges
- C: cross edges

$\{u, v\} \in E \setminus T$ is a *back edge* if there is a path in T from the root s touching both u and v

Lemma 2. *If a spanning tree is constructed by depth first search, then there are no cross edges.*

Proof.

- Assume $\{u, v\}$ is cross edge. Without loss of generality u is found before v.
- then edge $\{u, v\}$ is included in T and v is labelled *old* already during recursive call of $dfs(u)$

back edges and cross edges:

- T: tree edges
- B: back edges
- C: cross edges

$$E = T \,\dot{\cup}\, B \,\dot{\cup}\, C$$

undirected graph $G = (V, E)$ is called *connected* if r(u,v) (reachable) for all u,v, in V

$\{u, v\} \in E \setminus T$ is a *back edge*
if there is a path in T from the root s touching both u and v

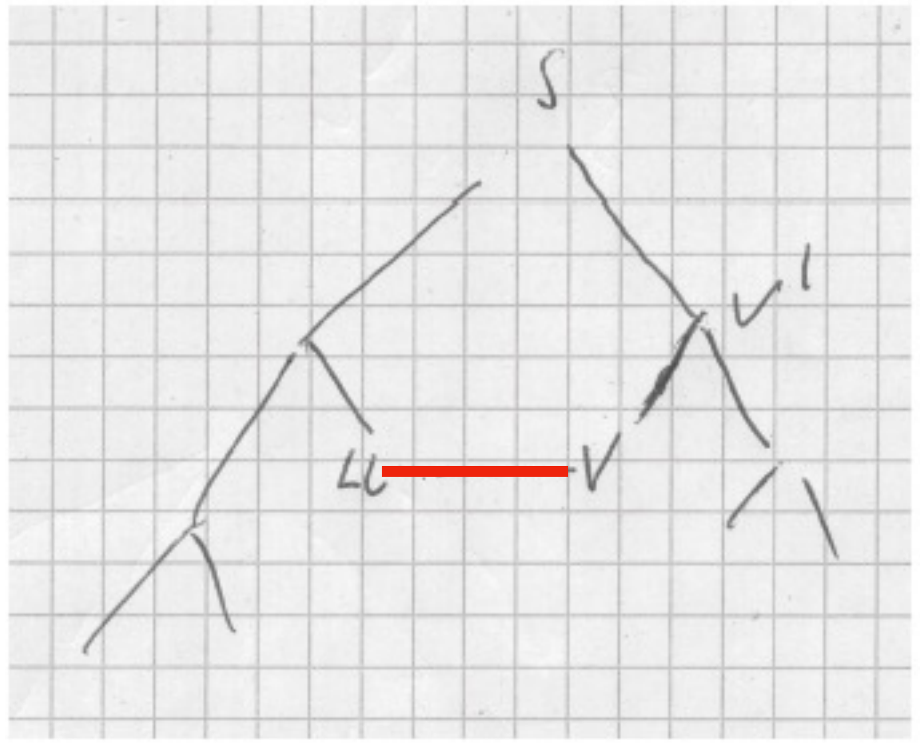
an undirected graph is called a *tree*,
if it is connected and cycle free

a *spanning tree* of $G = (V, E)$ is
a tree (V, T) with $T \subseteq E$

Lemma 2. *If a spanning tree is constructed by depth first search, then there are no cross edges.*

Proof.

- Assume $\{u, v\}$ is cross edge. Without loss of generality u is found before v.
- then edge $\{u, v\}$ is included in T and v is labelled *old* already during recursive call of $dfs(u)$



back edges and cross edges:

- T: tree edges
- B: back edges
- C: cross edges

$$E = T \,\dot{\cup}\, B \,\dot{\cup}\, C$$

undirected graph $G = (V, E)$ is called *connected* if r(u,v) (reachable) for all u,v, in V

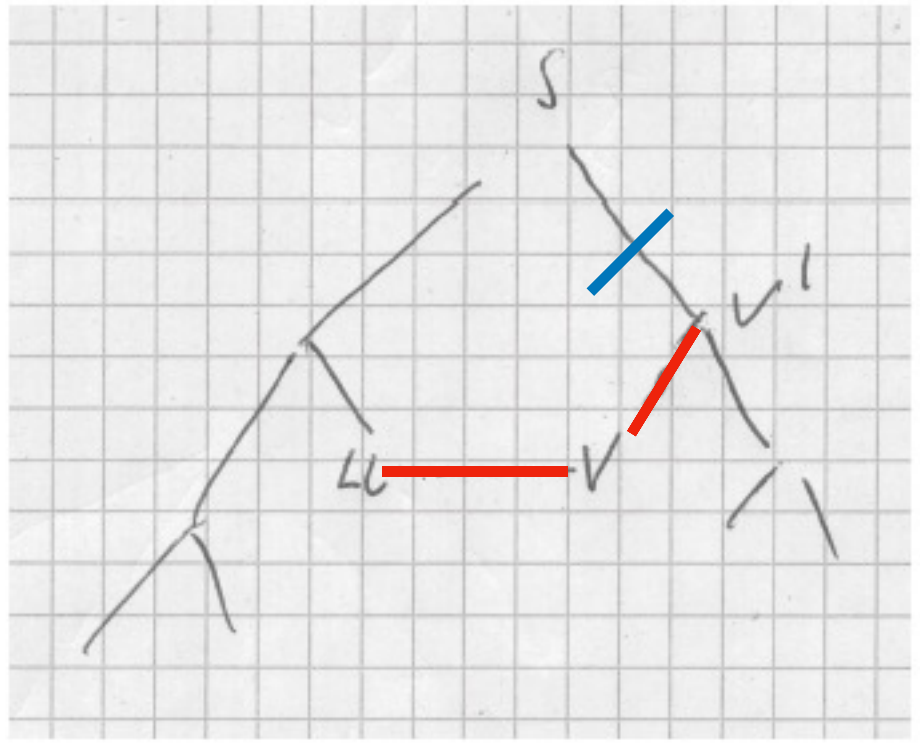
$\{u, v\} \in E \setminus T$ is a *back edge*
if there is a path in T from the root s touching both u and v

an undirected graph is called a *tree*,
if it is connected and cycle free

a *spanning tree* of $G = (V, E)$ is
a tree (V, T) with $T \subseteq E$

Lemma 2. *If a spanning tree is constructed by depth first search, then there are no cross edges.*

Proof.



- Assume $\{u, v\}$ is cross edge. Without loss of generality u is found before v.
- then edge $\{u, v\}$ is included in T and v is labelled *old* already during recursive call of $dfs(u)$