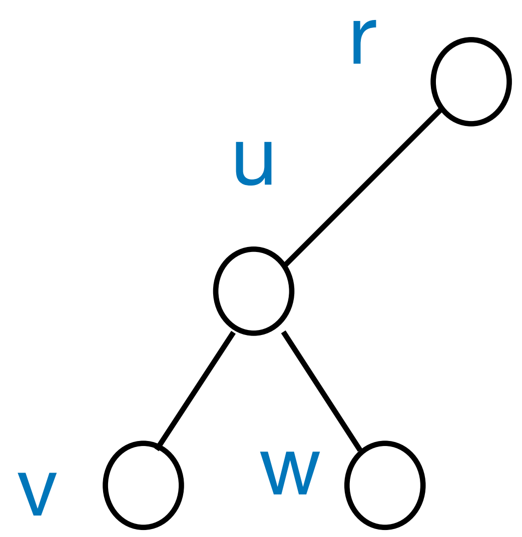
binary search trees

binary search trees

binary search trees

binary search trees (BST's): definition and notations

- interior nodes have 1 or 2 sons
- set elements $k \in S$ are stored in *key* component of all nodes

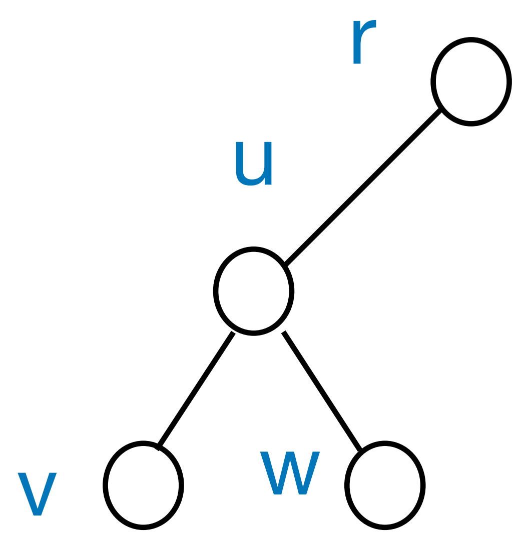


nodes u are structs with components

- p: parent
- l,r: sons
  - n.l= null: left son not present
  - u.r = null: right son not present
  - u.p = null: root
- key: for elements $s \in S$
- max: maximal key stored in T(u)

binary search trees (BST's): definition and notations

- interior nodes have 1 or 2 sons
- set elements $x \in S$ are stored in *key* component of all nodes

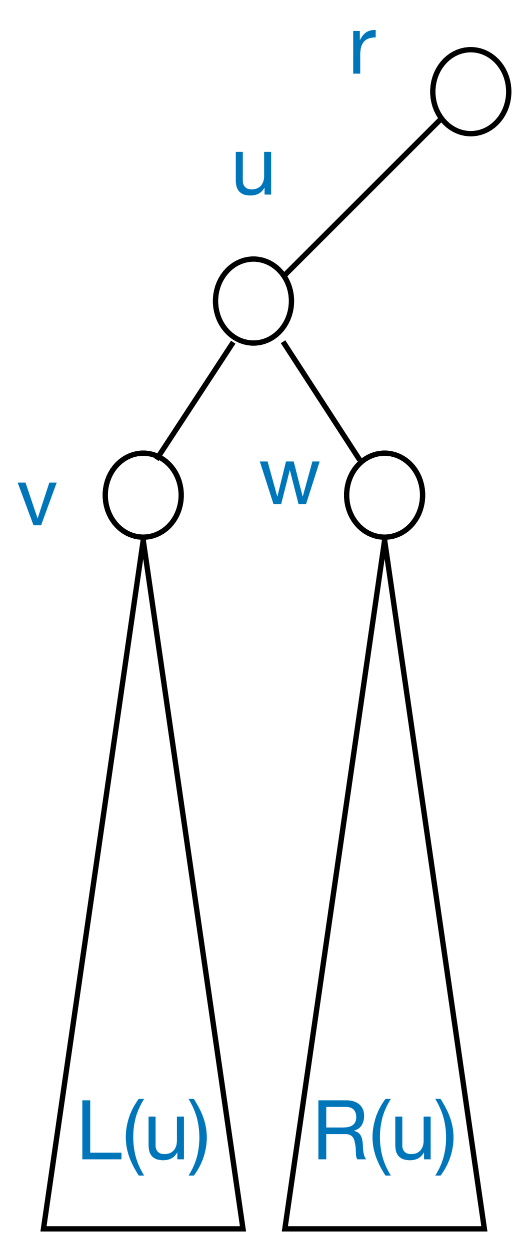


$l(u) = v \;\; r(u) = w \;\; p(u) = r$

notation (Java):

$$
\begin{array}{rcll}
l(u) & = & u.l & \text{left son} \\
r(u) & = & u.r & \text{right son} \\
p(u) & = & u.p & \text{parent} \\
L(u) & = & T(l(u)) & \text{subtree rooted in left son of } u \\
R(u) & = & T(r(u)) & \text{subtree rooted in right son of } u \\
T(u) & : & \text{subtree of T with root u} & \\
h(u) & = & \text{heigt of } u & \\
h(T) & = & h(root) &
\end{array}
$$

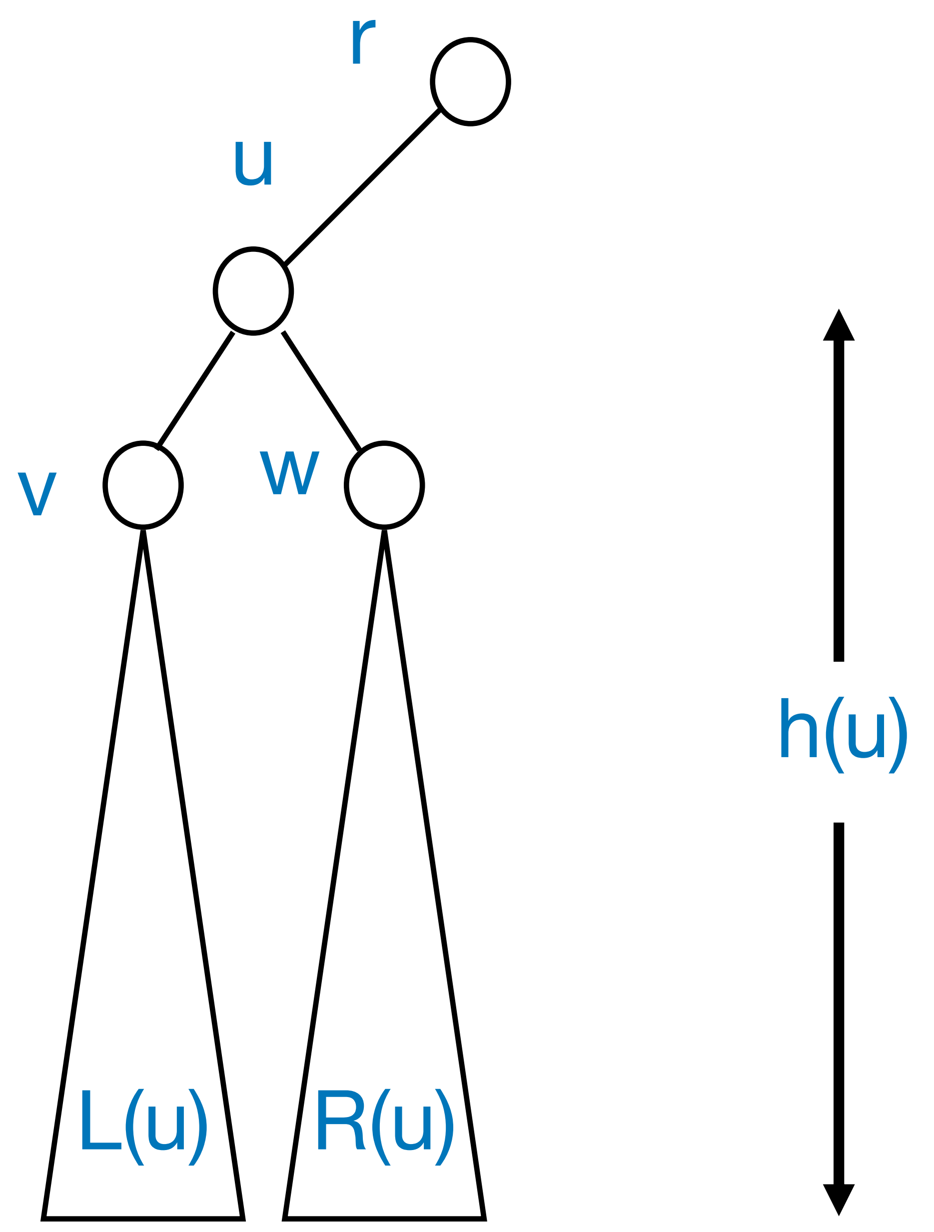
binary search trees (BST's): definition and notations



notation (Java):

$$
\begin{array}{lcll}
l(u) & = & u.l & \text{left son} \\
r(u) & = & u.r & \text{right son} \\
p(u) & = & u.p & \text{parent} \\
L(u) & = & T(l(u)) & \text{subtree rooted in left son of } u \\
R(u) & = & T(r(u)) & \text{subtree rooted in right son of } u \\
T(u) & : & \multicolumn{2}{l}{\text{subtree of T with root u}} \\
h(u) & = & \multicolumn{2}{l}{\text{heigt of } u} \\
h(T) & = & h(root) &
\end{array}
$$

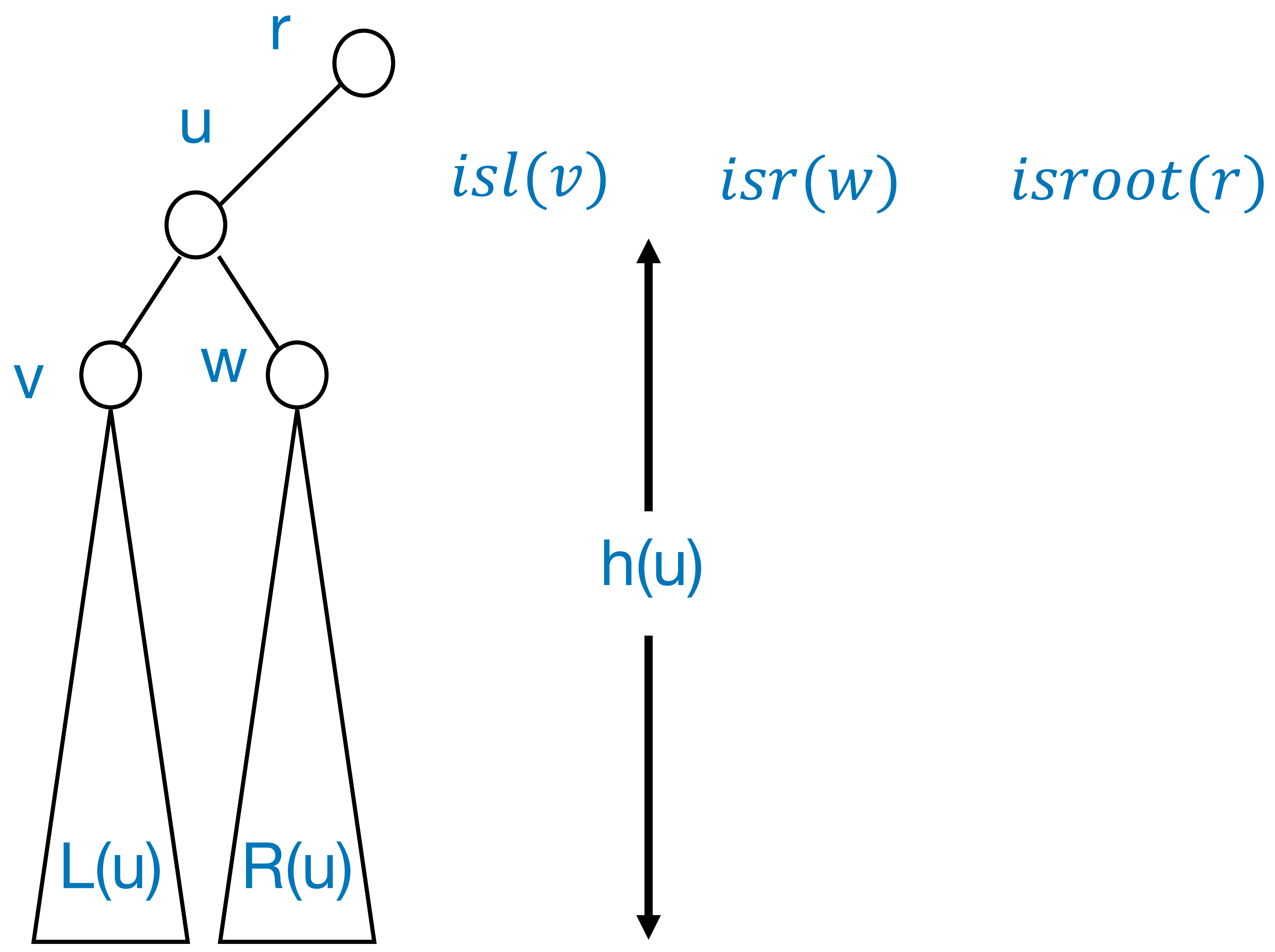
binary search trees (BST's): definition and notations



notation (Java):

$$
\begin{array}{lcll}
l(u) & = & u.l & \text{left son} \\
r(u) & = & u.r & \text{right son} \\
p(u) & = & u.p & \text{parent} \\
L(u) & = & T(l(u)) & \text{subtree rooted in left son of } u \\
R(u) & = & T(r(u)) & \text{subtree rooted in right son of } u \\
T(u) & : & \text{subtree of T with root u} & \\
h(u) & = & \text{heigt of } u & \\
h(T) & = & h(root) &
\end{array}
$$

binary search trees (BST's): definition and notations



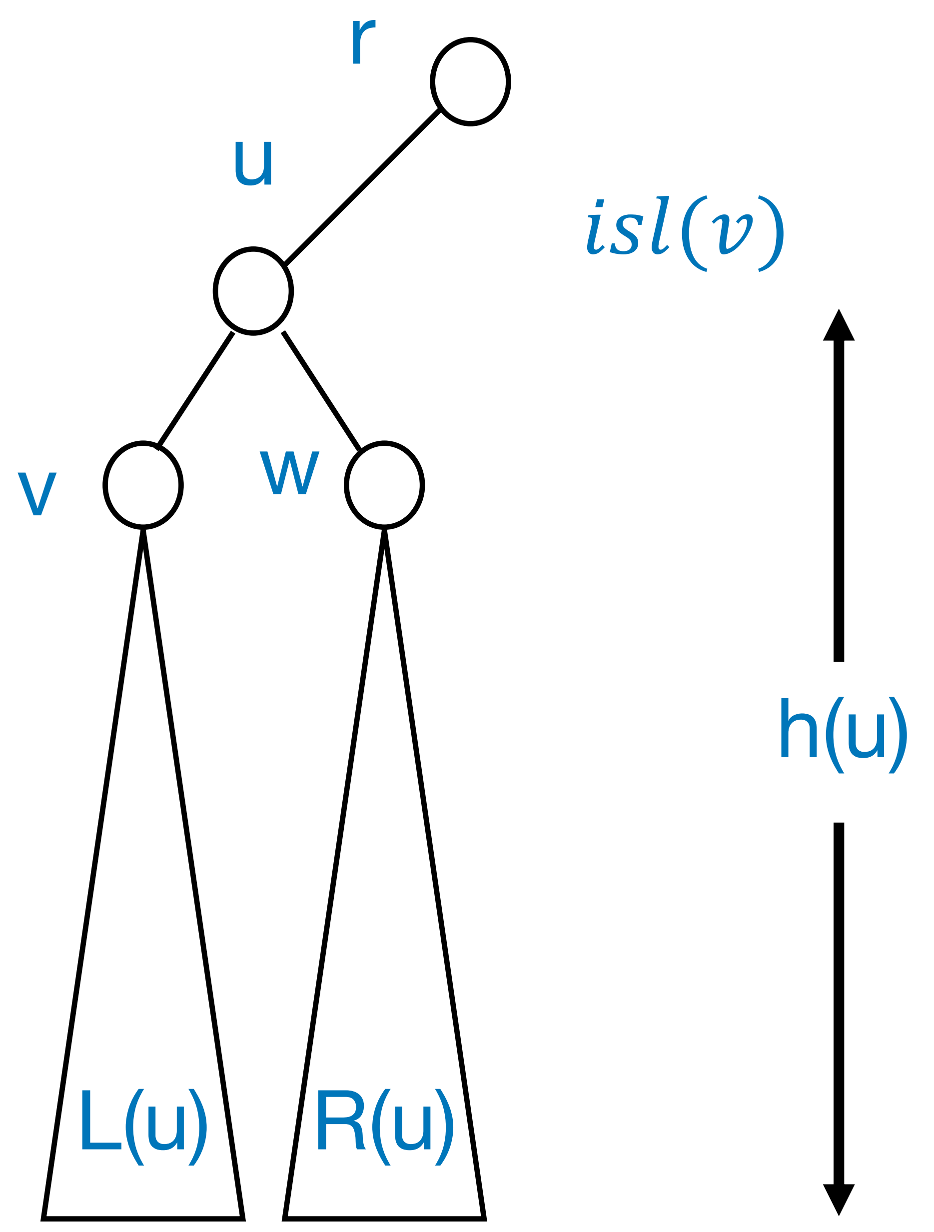
notation (Java):

$$
\begin{aligned}
l(u) &= u.l \quad \text{left son}\\
r(u) &= u.r \quad \text{right son}\\
p(u) &= u.p \quad \text{parent}\\
L(u) &= T(l(u)) \quad \text{subtree rooted in left son of } u\\
R(u) &= T(r(u)) \quad \text{subtree rooted in right son of } u\\
T(u) &: \text{subtree of T with root u}\\
h(u) &= \text{heigt of } u\\
h(T) &= h(root)
\end{aligned}
$$

predicates:

$$
\begin{aligned}
isl(u) &\equiv y = l(p(y)) \quad \text{'is left son'}\\
isr(u) &\equiv y = r(p(y)) \quad \text{'is right son'}\\
isroot(u) &\equiv u.p = null \quad \text{'is root'}\\
isleaf(u) &\equiv u.l = null \wedge u.r = null \quad \text{'is leaf'}
\end{aligned}
$$

binary search trees (BST's): definition and notations



$isl(v)$ $isr(w)$ $isroot(r)$

notation (Java):

$$
\begin{array}{lcl}
l(u) & = & u.l \quad \text{left son} \\
r(u) & = & u.r \quad \text{right son} \\
p(u) & = & u.p \quad \text{parent} \\
L(u) & = & T(l(u)) \quad \text{subtree rooted in left son of } u \\
R(u) & = & T(r(u)) \quad \text{subtree rooted in right son of } u \\
T(u) & : & \text{subtree of T with root u} \\
h(u) & = & \text{heigt of } u \\
h(T) & = & h(root)
\end{array}
$$

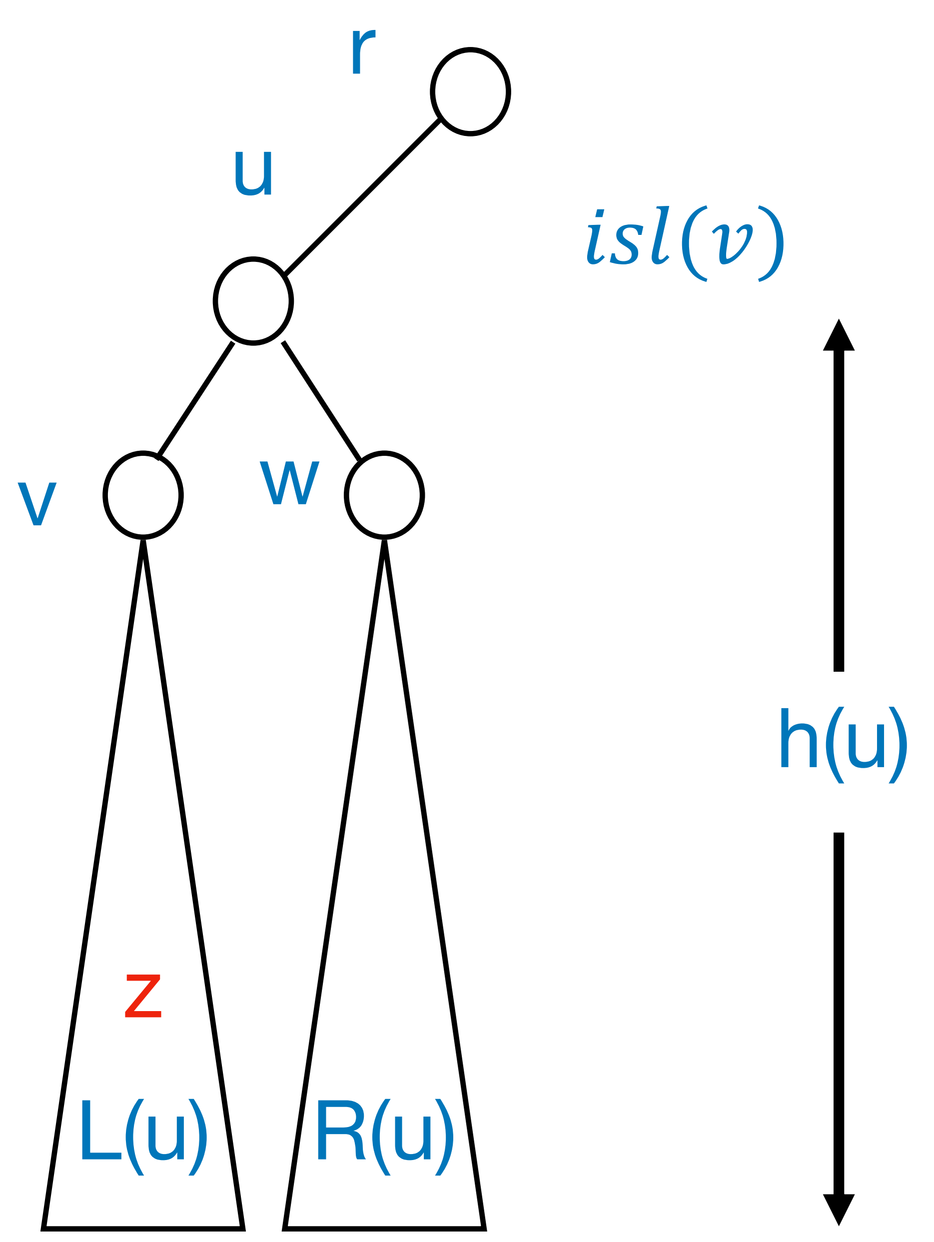
predicates:

$$
\begin{array}{lcl}
isl(u) & \equiv & y = l(p(y)) \quad \text{'is left son'} \\
isr(u) & \equiv & y = r(p(y)) \quad \text{'is right son'} \\
isroot(u) & \equiv & u.p = null \quad \text{'is root'} \\
isleaf(u) & \equiv & u.l = null \wedge u.r = null \quad \text{'is leaf'}
\end{array}
$$

distinct set elements:

$$u, v \in T \quad u \neq v \rightarrow key(u) \neq key(v)$$

binary search trees (BST's): definition and notations



$isl(v)$ $\quad isr(w)$ $\quad isroot(r)$

notation (Java):

$$
\begin{array}{rcll}
l(u) & = & u.l & \text{left son} \\
r(u) & = & u.r & \text{right son} \\
p(u) & = & u.p & \text{parent} \\
L(u) & = & T(l(u)) & \text{subtree rooted in left son of } u \\
R(u) & = & T(r(u)) & \text{subtree rooted in right son of } u \\
T(u) & : & \text{subtree of T with root u} & \\
h(u) & = & \text{heigt of } u & \\
h(T) & = & h(root) &
\end{array}
$$

predicates:

$$
\begin{array}{rcll}
isl(u) & \equiv & y = l(p(y)) & \text{'is left son'} \\
isr(u) & \equiv & y = r(p(y)) & \text{'is right son'} \\
isroot(u) & \equiv & u.p = null & \text{'is root'} \\
isleaf(u) & \equiv & u.l = null \wedge u.r = null & \text{'is leaf'}
\end{array}
$$

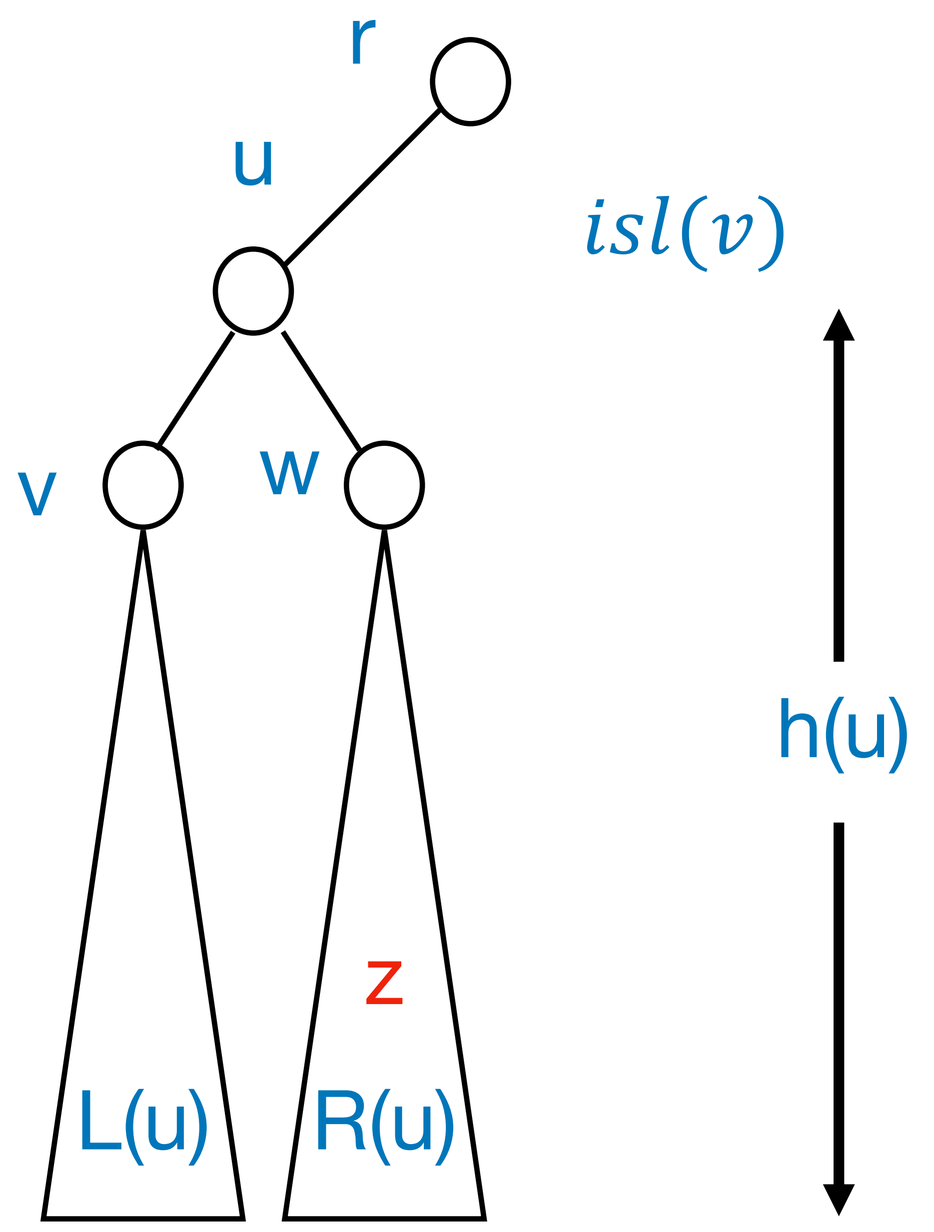
distinct set elements:

$$u, v \in T \quad u \neq v \rightarrow key(u) \neq key(v)$$

BST-property:

$$z \in L(u) \rightarrow key(z) < key(u)$$

binary search trees (BST's): definition and notations



$isl(v)$ $isr(w)$ $isroot(r)$

notation (Java):

$$
\begin{array}{rcll}
l(u) & = & u.l & \text{left son} \\
r(u) & = & u.r & \text{right son} \\
p(u) & = & u.p & \text{parent} \\
L(u) & = & T(l(u)) & \text{subtree rooted in left son of } u \\
R(u) & = & T(r(u)) & \text{subtree rooted in right son of } u \\
T(u) & : & \text{subtree of T with root u} & \\
h(u) & = & \text{heigt of } u & \\
h(T) & = & h(root) &
\end{array}
$$

predicates:

$$
\begin{array}{rcll}
isl(u) & \equiv & y = l(p(y)) & \text{'is left son'} \\
isr(u) & \equiv & y = r(p(y)) & \text{'is right son'} \\
isroot(u) & \equiv & u.p = null & \text{'is root'} \\
isleaf(u) & \equiv & u.l = null \wedge u.r = null & \text{'is leaf'}
\end{array}
$$

distinct set elements:

$u, v \in T \quad u \neq v \rightarrow key(u) \neq key(v)$

BST-property:

$z \in L(u) \rightarrow key(z) < key(u)$

$z \in R(u) \rightarrow key(z) > key(u)$

binary search trees (BST's): example

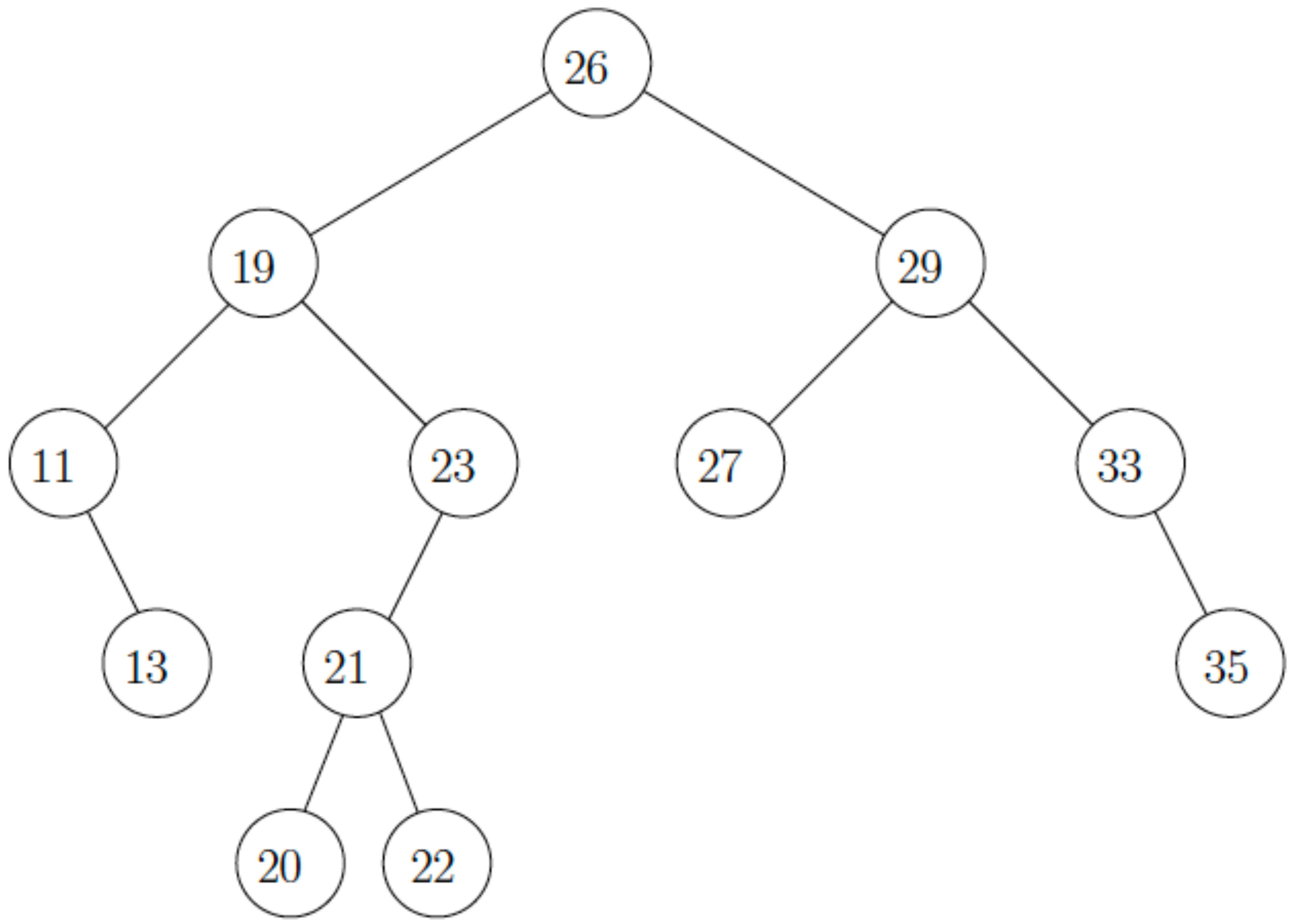


Figure 6.1: A binary search tree

BST-property:

$z \in L(u) \rightarrow key(z) < key(u)$

$z \in R(u) \rightarrow key(z) > key(u)$

binary search trees (BST’s): more notation

u v

$u \notin T(v) \land v \notin T(u)$

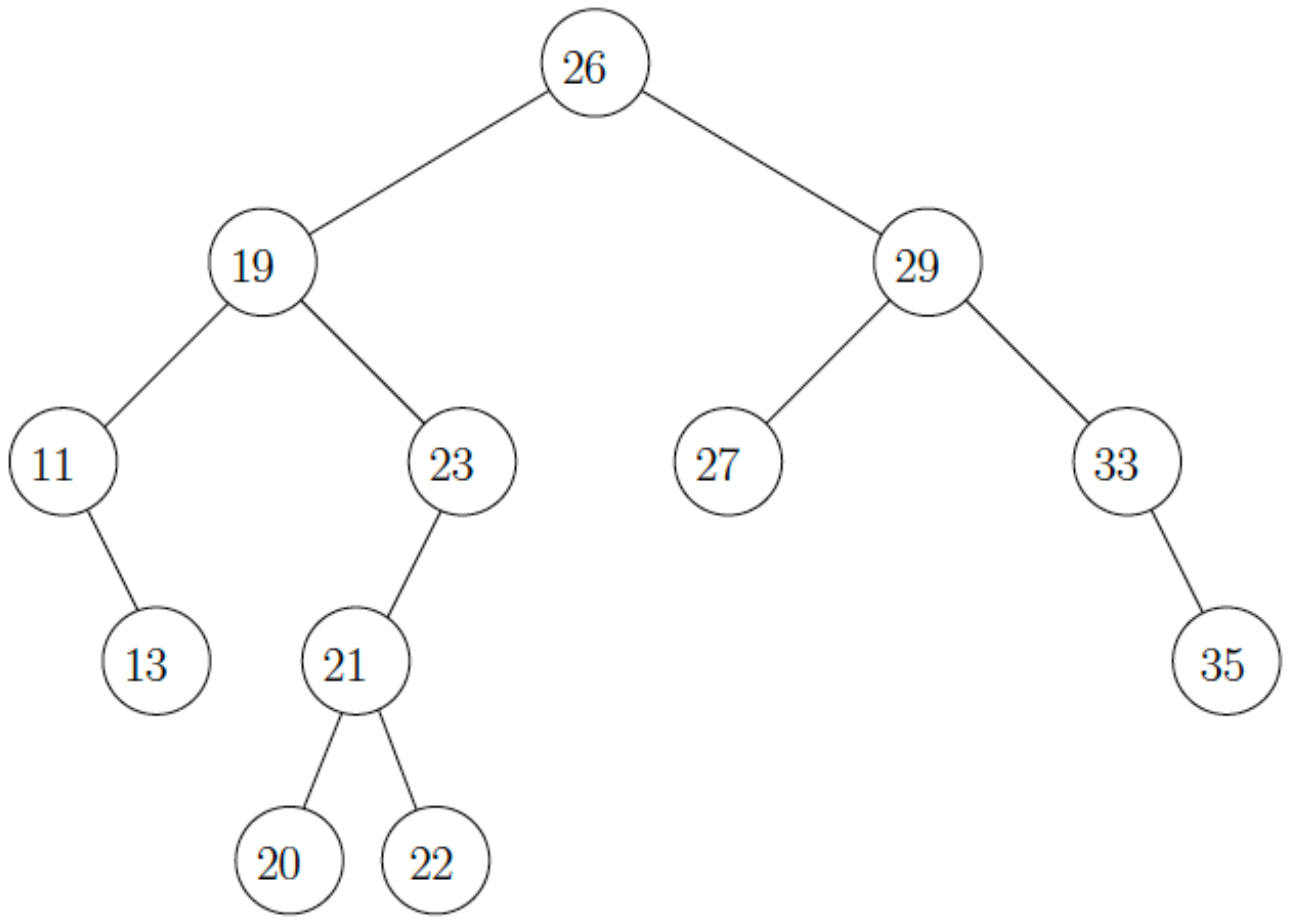


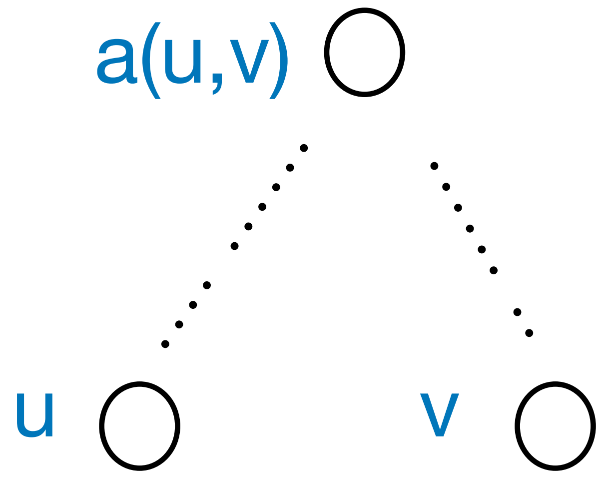
Figure 6.1: A binary search tree

BST-property:

$z \in L(u) \rightarrow key(z) < key(u)$

$z \in R(u) \rightarrow key(z) > key(u)$

binary search trees (BST's): more notation



$u \notin T(v) \wedge v \notin T(u)$

$a(u, v)$: lowest common ancestor

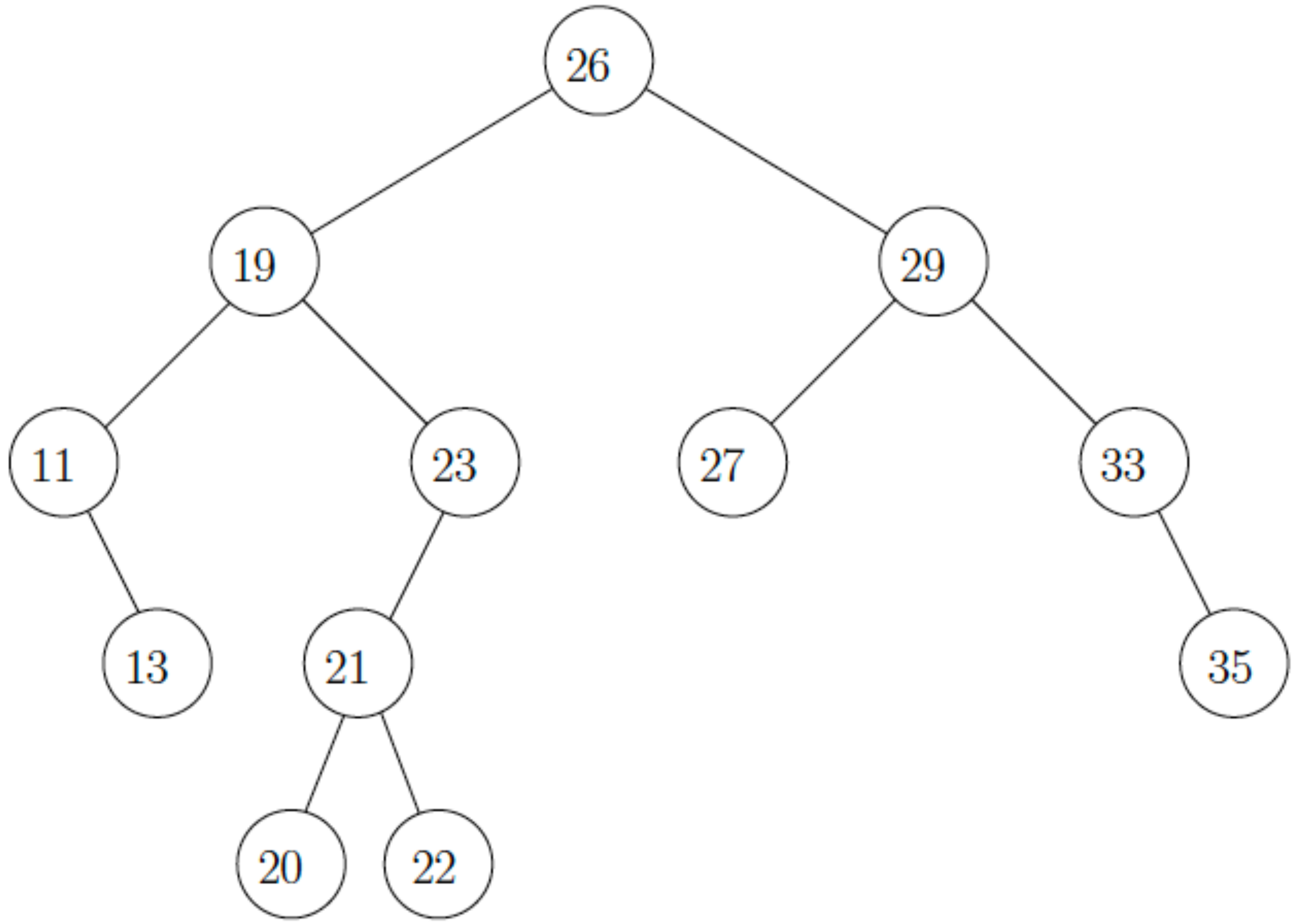


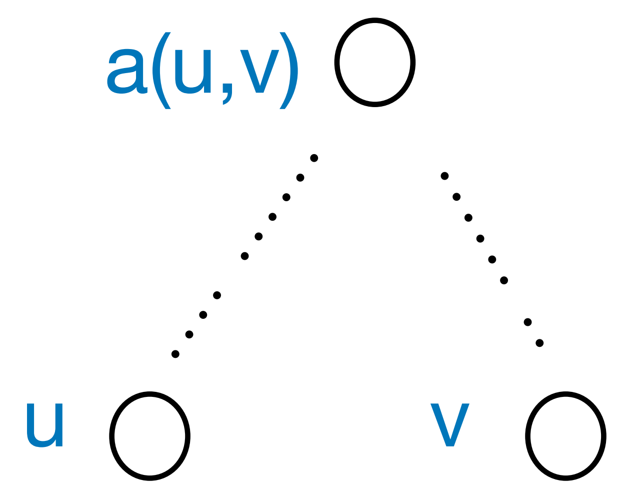
Figure 6.1: A binary search tree

BST-property:

$z \in L(u) \rightarrow key(z) < key(u)$

$z \in R(u) \rightarrow key(z) > key(u)$

binary search trees (BST's): more notation



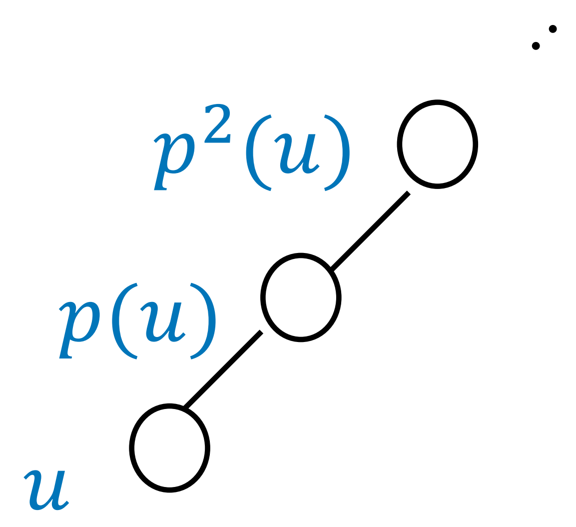
$u \notin T(v) \wedge v \notin T(u)$

$a(u,v)$: lowest common ancestor

iterated parent:

$$p^0(u) = u$$

$$p^{i+1}(u) = p(p^i(u))$$



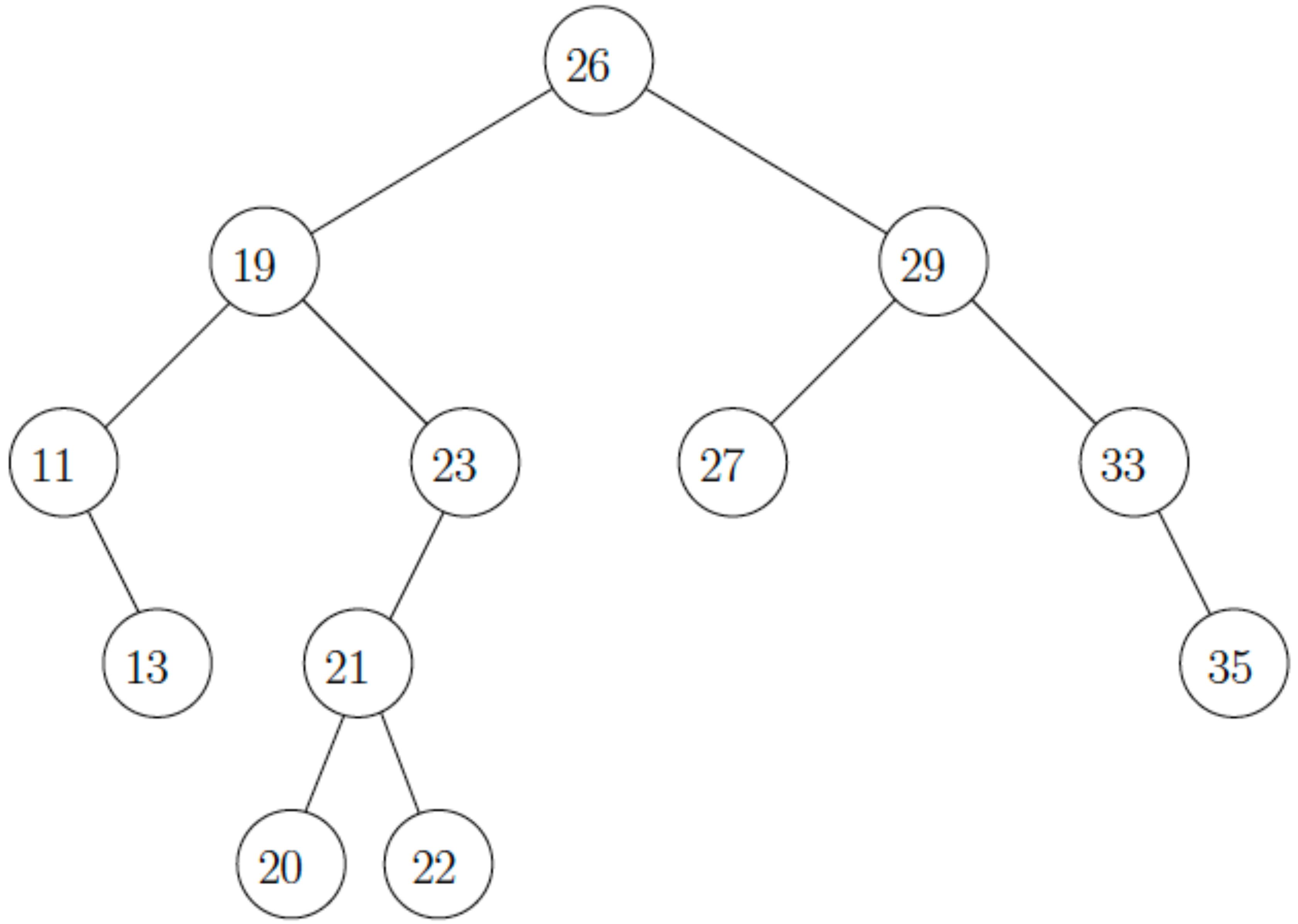


Figure 6.1: A binary search tree

BST-property:

$z \in L(u) \rightarrow key(z) < key(u)$

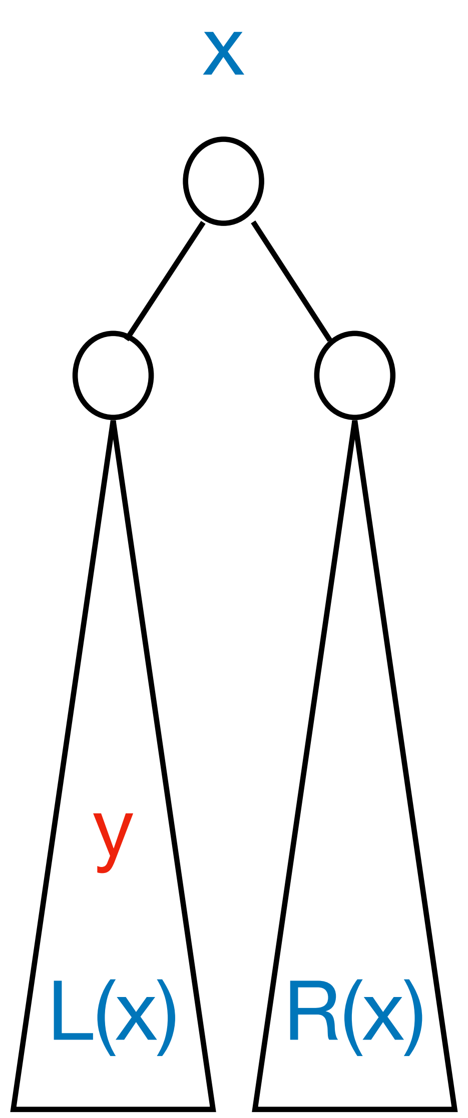
$z \in R(u) \rightarrow key(z) > key(u)$

binary search trees (BST's): lemmas

Lemma 1.

$$y \in T(x) \wedge key(y) < key(x) \rightarrow y \in L(x)$$

Proof. BST-condition, part 2 $\square$



BST-property:

$z \in L(x) \rightarrow key(z) < key(x)$

$z \in R(x) \rightarrow key(z) > key(x)$

binary search trees (BST's): lemmas

Lemma 1.

$$y \in T(x) \wedge key(y) < key(x) \rightarrow y \in L(x)$$

Proof. BST-condition, part 2 $\square$

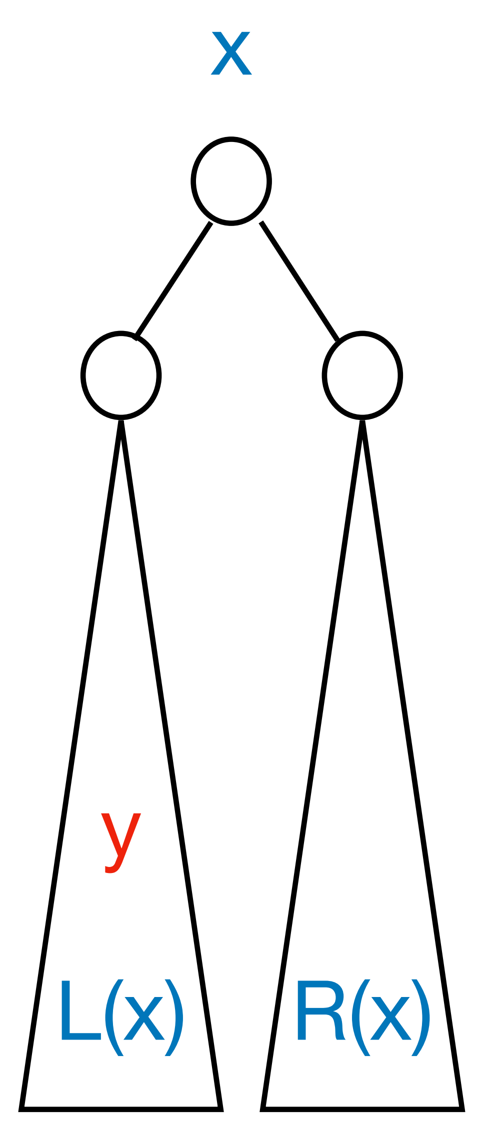
Lemma 2.

$$y \notin T(z) \wedge z \notin T(y) \wedge key(y) < key(z) \rightarrow y \in L(a(y,z))$$

Proof. Let $w = a(y,z)$. Then b/c not in each others subtrees.

$$\big(y \in L(w) \wedge z \in R(w)\big) \vee \big(y \in R(w) \wedge z \in L(w)\big)$$

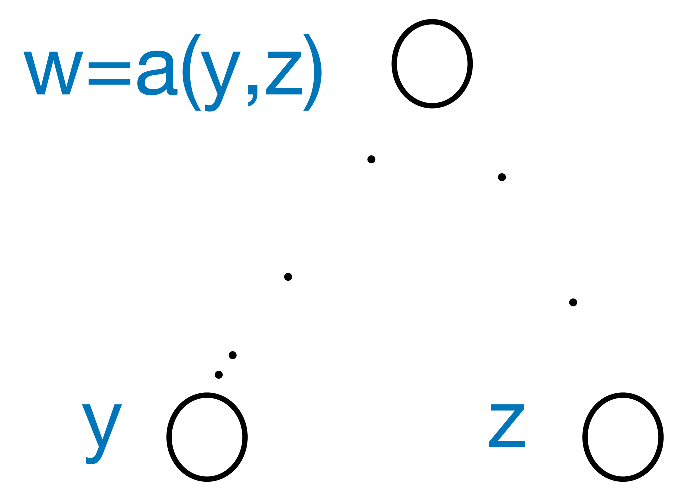
second case impossible by BST-condition, part 2



BST-property:

$z \in L(x) \rightarrow key(z) < key(x)$

$z \in R(x) \rightarrow key(z) > key(x)$



binary search trees (BST's): search

- input: node x, key k
- output:

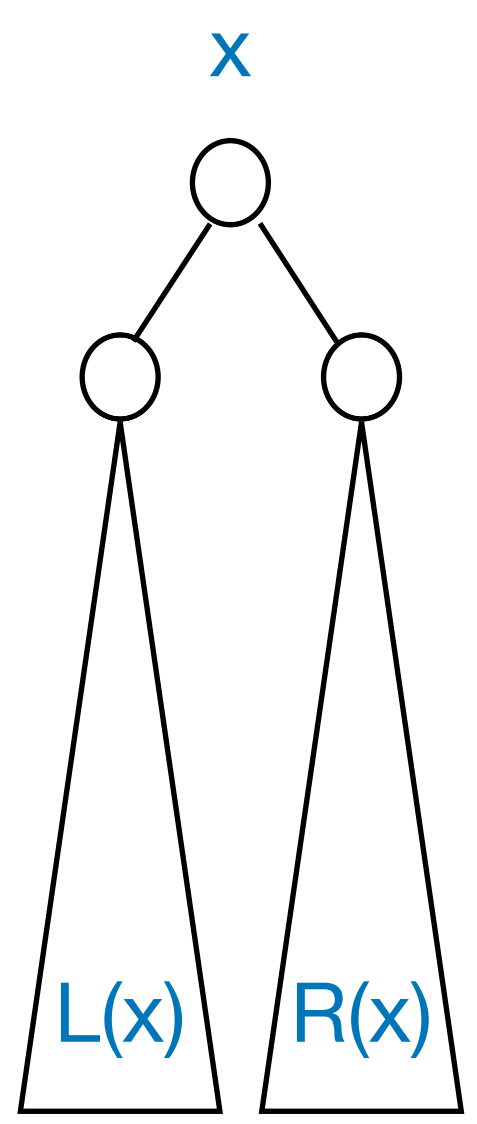
$$search(x,k) = \begin{cases} y \in T(x) \text{ with } key(y) = k & \text{if it exists} \\ NULL & \text{otherwise} \end{cases}$$

```
if key(x)= k {return x};
if key(x)< k {if x.l = null {return NULL} else {search(l(x),k)}};
if key(x)>k {if x.r = null {return NULL} else {search(l(x),k)}}
```

BST-property:

$z \in L(x) \rightarrow key(z) < key(x)$

$z \in R(x) \rightarrow key(z) > key(x)$



binary search trees (BST's): search

- input: node x, key k
- output:

$$search(x,k) = \begin{cases} y \in T(x) \text{ with } key(y) = k & \text{if it exists} \\ NULL & \text{otherwise} \end{cases}$$

```
if key(x)= k {return x};
if key(x)< k {if x.l = null {return NULL} else {search(l(x),k)}};
if key(x)>k {if x.r = null {return NULL} else {search(l(x),k)}}
```

correctness proof by induction on $h(x)$

- $h(x) = 0$ x is leaf. $x.l = x.l = 0$
- $h(x) > 0$

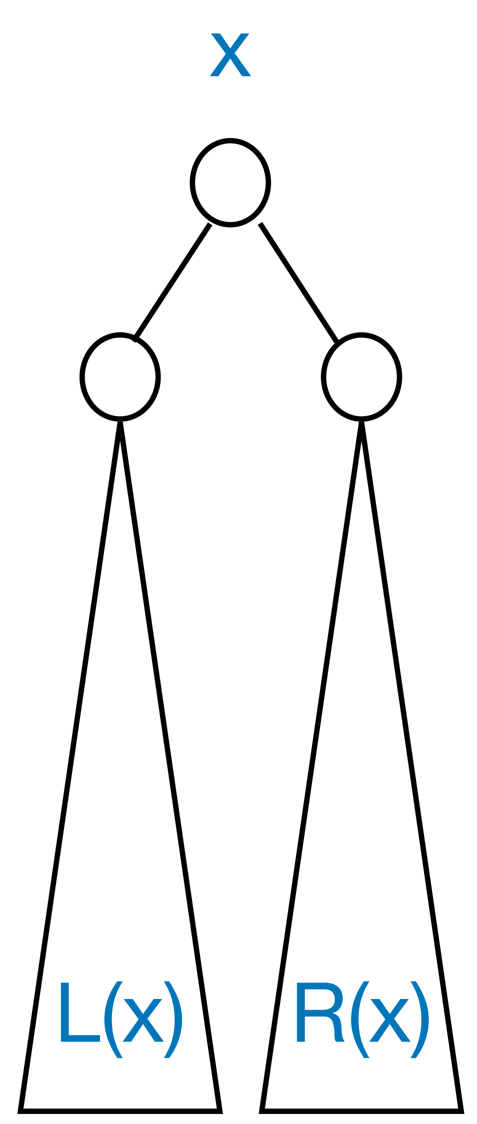
$$k < key(x) \rightarrow y \in L(x) \quad \text{if it exists; BST-property, part 2}$$

$$k > key(x) \rightarrow y \in R(x) \quad \text{if it exists; BST-property, part 1}$$

BST-property:

$z \in L(x) \rightarrow key(z) < key(x)$

$z \in R(x) \rightarrow key(z) > key(x)$



binary search trees (BST's): search

- input: node x, key k
- output:

$$search(x,k) = \begin{cases} y \in T(x) \text{ with } key(y) = k & \text{if it exists} \\ NULL & \text{otherwise} \end{cases}$$

```
if key(x)= k {return x};
if key(x)< k {if x.l = null {return NULL} else {search(l(x),k)}};
if key(x)>k {if x.r = null {return NULL} else {search(l(x),k)}}
```

correctness proof by induction on $h(x)$

- $h(x) = 0$ x is leaf. $x.l = x.l = 0$
- $h(x) > 0$

$$k < key(x) \rightarrow y \in L(x) \quad \text{if it exists; BST-property, part 2}$$

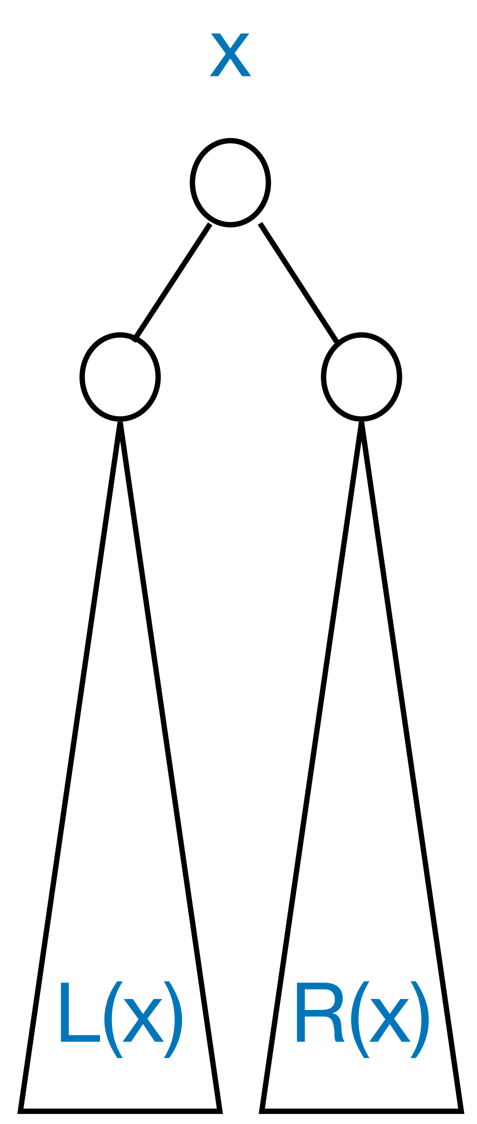
$$k > key(x) \rightarrow y \in R(x) \quad \text{if it exists; BST-property, part 1}$$

time O(h(x))

BST-property:

$z \in L(x) \rightarrow key(z) < key(x)$

$z \in R(x) \rightarrow key(z) > key(x)$



binary search trees (BST's): minimum

- input: node x
- output:

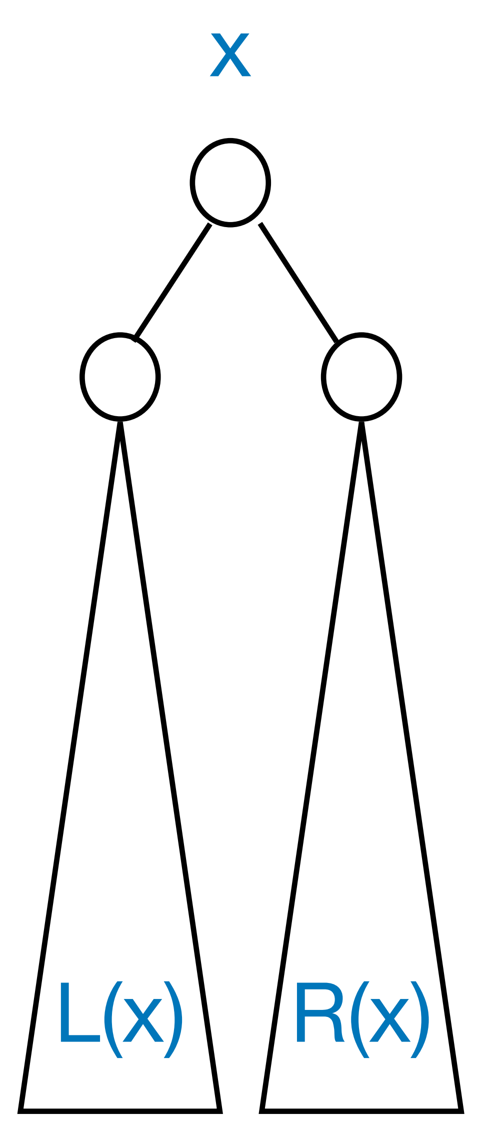
$$min(x) = min\{key(y) \mid y \in T(x))\}$$

```
if x.l=null {return key(x)} else {return min(l(x))}
```

BST-property:

$$z \in L(x) \rightarrow key(z) < key(x)$$

$$z \in R(x) \rightarrow key(z) > key(x)$$



binary search trees (BST's): minimum

BST-property:

$z \in L(x) \rightarrow key(z) < key(x)$

$z \in R(x) \rightarrow key(z) > key(x)$

- input: node x
- output:

$$min(x) = min\{key(y) \mid y \in T(x))\}$$

```
if x.l=null {return key(x)} else {return min(l(x))}
```

correctness proof by induction on $h(x)$

- $h(x) = 0$.

$$isleaf(x)\ ,\ x.l = null$$

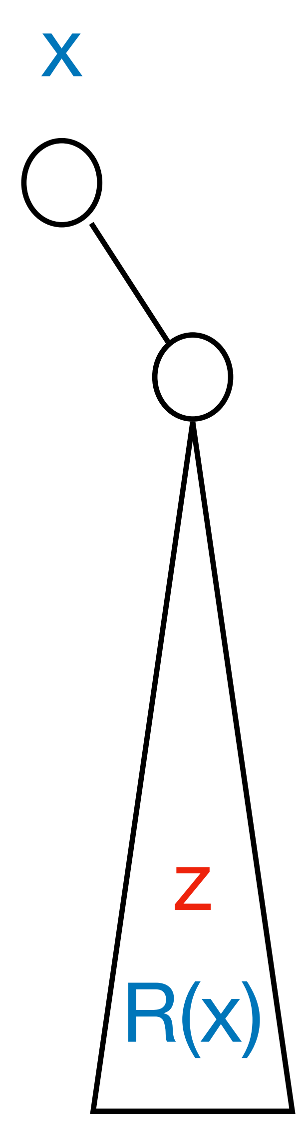
- $h(x) > 0$. If $x.l = null$, then

$$\forall z \in R(x).\ key(x) < key(z) \quad \text{BST-property, part 2}$$

If $x.l \neq null$ and $w = min(l(x))$ then

$$w = \min\{key(y) \mid y \in L(x)\} \quad \text{induction hypothesis}$$

$$z \in R(x) \rightarrow key(w) < key(x) < kex(z) \quad \text{BST-property, both parts}$$



binary search trees (BST's): minimum

BST-property:

$z \in L(x) \rightarrow key(z) < key(x)$

$z \in R(x) \rightarrow key(z) > key(x)$

- input: node x
- output:

$$min(x) = min\{key(y) \mid y \in T(x))\}$$

```
if x.l=null {return key(x)} else {return min(l(x))}
```

correctness proof by induction on $h(x)$

- $h(x) = 0$.

$$isleaf(x) \text{ , } x.l = null$$

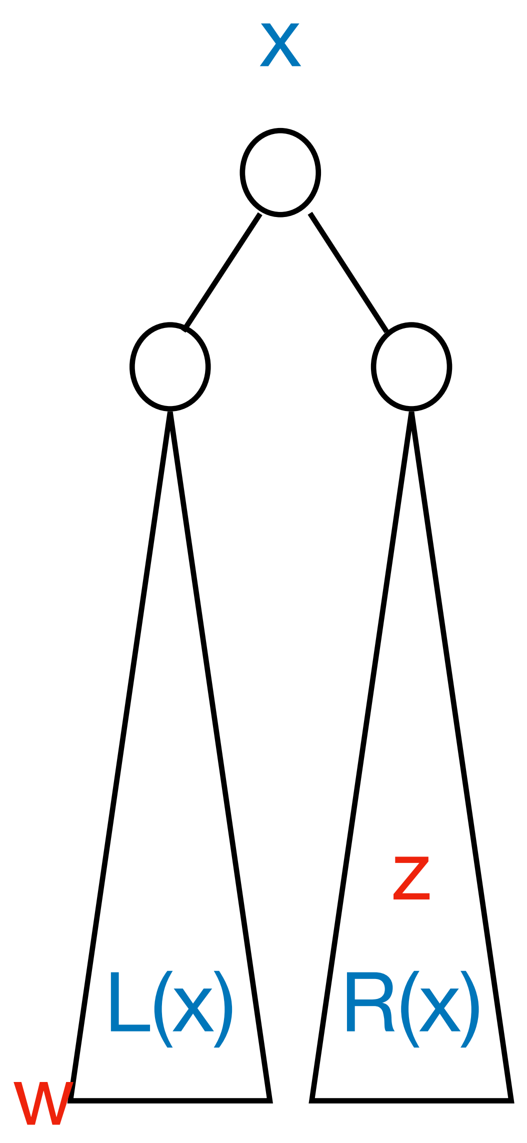
- $h(x) > 0$. If $x.l = null$, then

$$\forall z \in R(x).\ key(x) < key(z) \quad \text{BST-property, part 2}$$

If $x.l \neq null$ and $w = min(l(x))$ then

$$w = \min\{key(y) \mid y \in L(x)\} \quad \text{induction hypothesis}$$

$$z \in R(x) \rightarrow key(w) < key(x) < kex(z) \quad \text{BST-property, both parts}$$



binary search trees (BST's): minimum

- input: node x
- output:

$$min(x) = min\{key(y) \mid y \in T(x))\}$$

```
if x.l=null {return key(x)} else {return min(l(x))}
```

BST-property:

$$z \in L(x) \rightarrow key(z) < key(x)$$

$$z \in R(x) \rightarrow key(z) > key(x)$$

correctness proof by induction on $h(x)$

- $h(x) = 0$.

$$isleaf(x)\ ,\ x.l = null$$

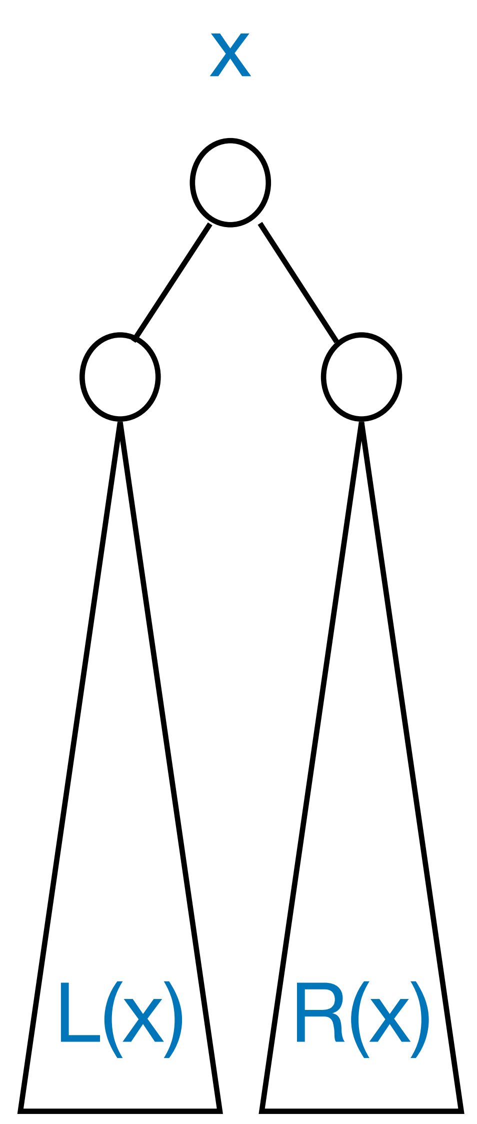
- $h(x) > 0$. If $x.l = null$, then

$$\forall z \in R(x).\ key(x) < key(z) \quad \text{BST-property, part 2}$$

  If $x.l \neq null$ and $w = min(l(x))$ then

$$w = \min\{key(y) \mid y \in L(x)\} \quad \text{induction hypothesis}$$

$$z \in R(x) \rightarrow key(w) < key(x) < kex(z) \quad \text{BST-property, both parts}$$



time O(h(x))

binary search trees (BST's): maximum

exercise

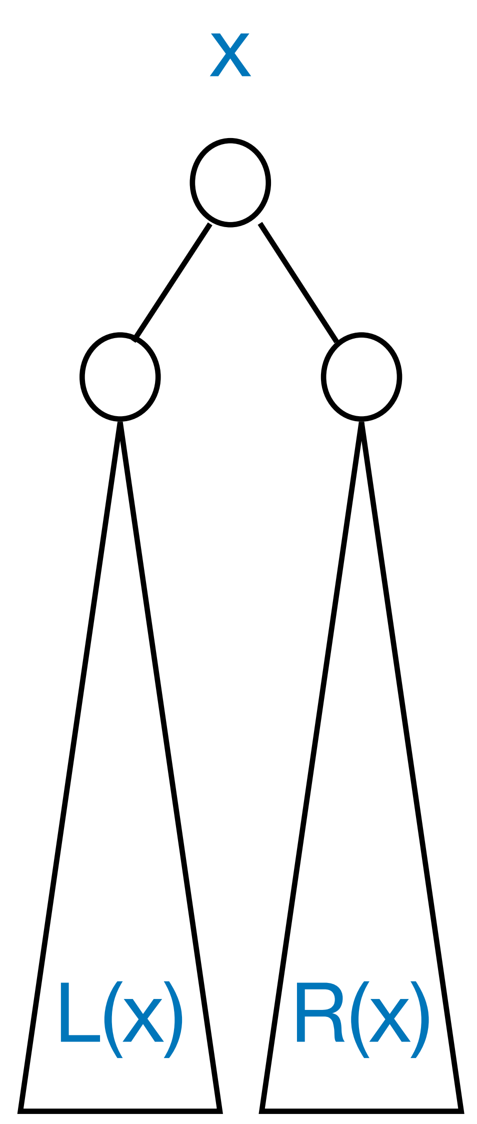
- input: node x
- output:

$$max(x) = max\{key(y) \mid y \in T(x))\}$$

BST-property:

$z \in L(x) \rightarrow key(z) < key(x)$

$z \in R(x) \rightarrow key(z) > key(x)$



binary search trees (BST's): successor

- input: node x
- output:

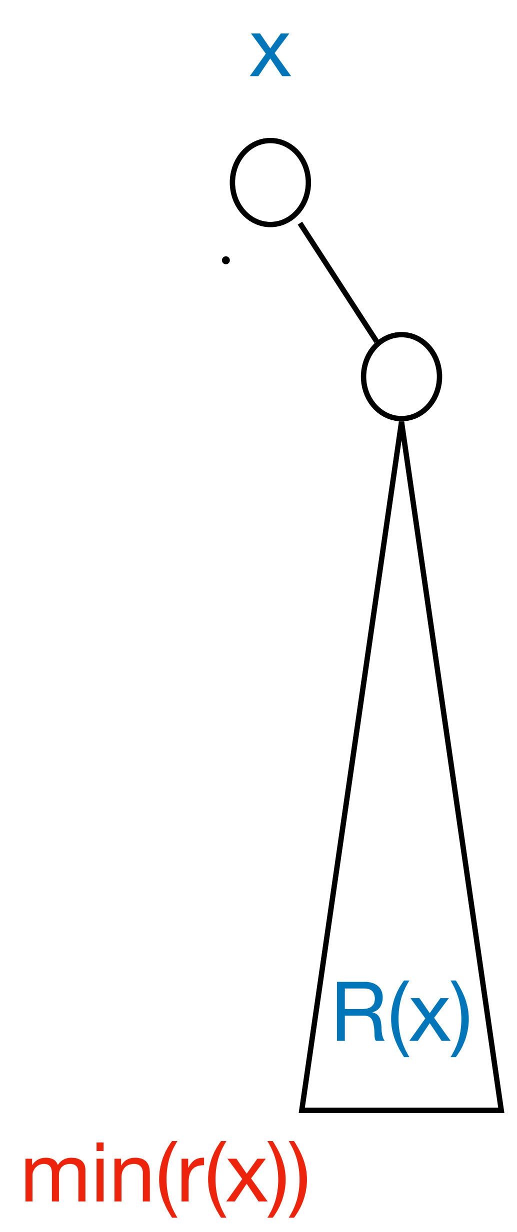
$$succ(x) = \begin{cases} y \in T \text{ with } key(y) = \min\{key(z) \mid z \in T\ ,\ key(z) > key(x)\} & \text{if it exists} \\ NULL & \text{otherwise} \end{cases}$$

```
if x.r != Null {return min(r(x))} else
{let i = min {j>=0 | /isl(p^j(x));  /*parent chasing of right sons*/
u = p^i(x);
return (isroot(u)? NULL:p(u))
}
```

BST-property:

$z \in L(x) \rightarrow key(z) < key(x)$

$z \in R(x) \rightarrow key(z) > key(x)$



binary search trees (BST's): successor

exercise

- input: node x
- output:

$$succ(x) = \begin{cases} y \in T \text{ with } key(y) = \min\{key(z) \mid z \in T\ ,\ key(z) > key(y)\} & \text{if it exists} \\ NULL & \text{otherwise} \end{cases}$$

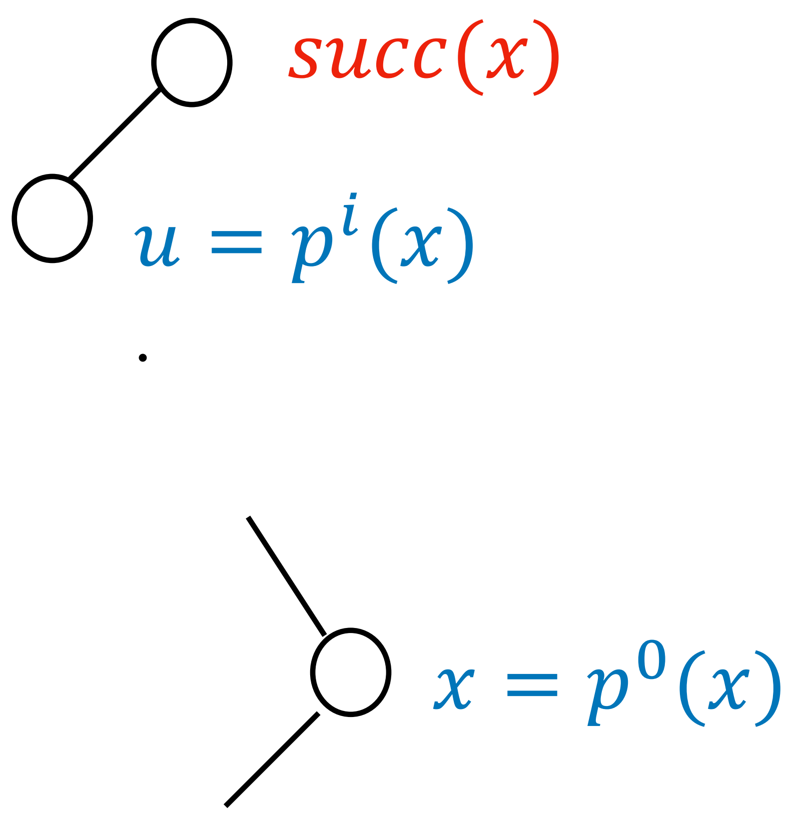
```
if x.r != Null {return min(r(x))} else
{let i = min {j>=0 | /isl(p^j(x));  /*parent chasing of right sons*/
u = p^i(x);
return (isroot(u)? NULL:p(u))
}
```

BST-property:

$z \in L(x) \to key(z) < key(x)$

$z \in R(x) \to key(z) > key(x)$

x



time O(max(h(x), d(x)))

binary search trees (BST's): successor correctness

exercise

- input: node x
- output:

$$succ(x) = \begin{cases} y \in T \text{ with } key(y) = \min\{key(z) \mid z \in T\ ,\ key(z) > key(y)\} & \text{if it exists} \\ NULL & \text{otherwise} \end{cases}$$

```
if x.r != Null {return min(r(x))} else
{let i = min {j>=0 | /isl(p^j(x));  /*parent chasing of right sons*/
u = p^i(x);
return (isroot(u)? NULL:p(u))
}
```

correctness: $x.r \neq null$

by contradiction. Let $y = min(r(x))$ and assume

$$key(x) < key(z) < key(y)$$

- $z \in T(x)$:

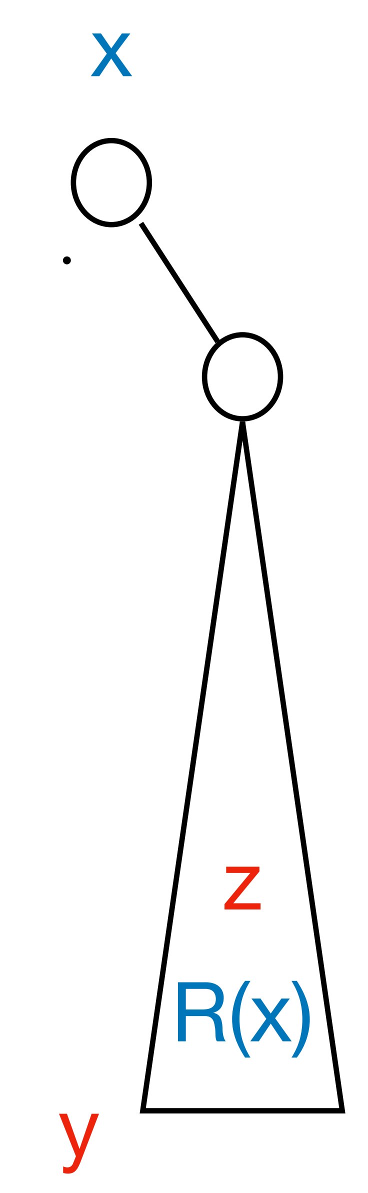
$$key(z) > key(x) \rightarrow z \in R(x) \quad \text{BST-condition, part 1}$$

$$z \in R(x) \rightarrow key(y) < key(z) \quad \text{correctness of min}$$

BST-property:

$$z \in L(x) \rightarrow key(z) < key(x)$$

$$z \in R(x) \rightarrow key(z) > key(x)$$



binary search trees (BST's): successor correctness

exercise

- input: node x
- output:

$$succ(x) = \begin{cases} y \in T \text{ with } key(y) = \min\{key(z) \mid z \in T \ , \ key(z) > key(y)\} & \text{if it exists} \\ NULL & \text{otherwise} \end{cases}$$

```
if x.r != Null {return min(r(x))} else
{let i = min {j>=0 | /isℓ(p^j(x));   /*parent chasing of right sons*/
u = p^i(x);
return (isroot(u)? NULL:p(u))
}
```

correctness: $x.r \neq null$

by contradiction. Let $y = min(r(x))$ and assume

$$key(x) < key(z) < key(y)$$

- $x \in T(z)$

$$key(z) > key(x) \rightarrow x \in L(z) \quad \text{lemma 1}$$

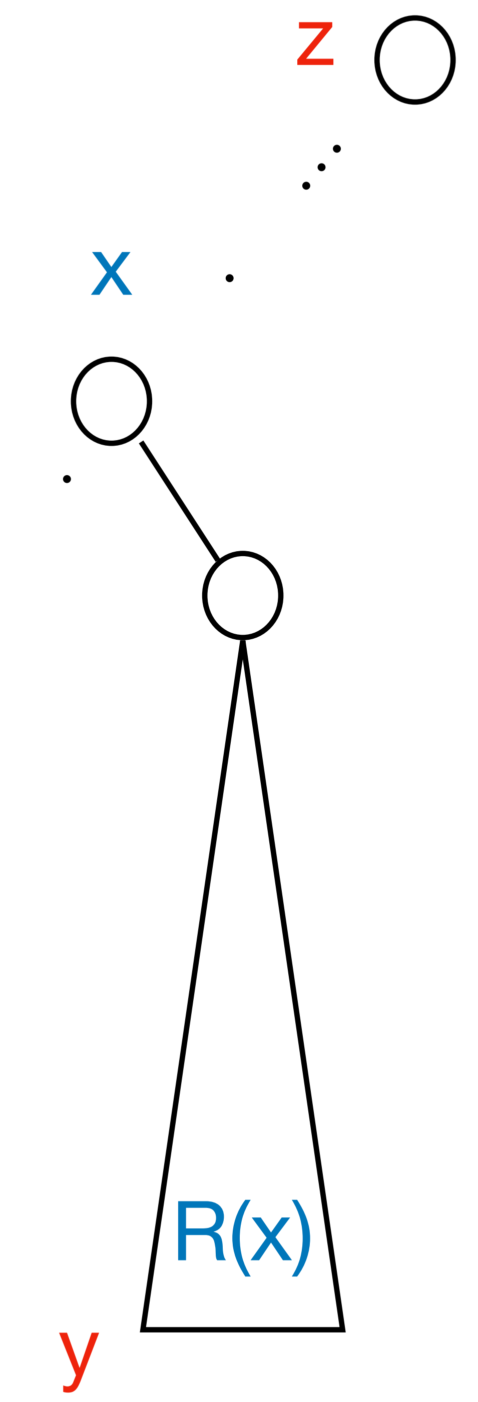
$$\rightarrow y \in L(z)$$

$$\rightarrow key(y) < key(z) \quad \text{BST-condition, part 1}$$

BST-property:

$z \in L(x) \rightarrow key(z) < key(x)$

$z \in R(x) \rightarrow key(z) > key(x)$



binary search trees (BST's): successor correctness

- input: node x
- output:

$$succ(x) = \begin{cases} y \in T \text{ with } key(y) = \min\{key(z) \mid z \in T\ ,\ key(z) > key(y)\} & \text{if it exists} \\ NULL & \text{otherwise} \end{cases}$$

```
if x.r != Null {return min(r(x))} else
{let i = min {j>=0 | /isl(p^j(x);  /*parent chasing of right sons*/
u = p^i(x);
return (isroot(u)? NULL:p(u))
}
```

correctness: $x.r \neq null$

by contradiction. Let $y = min(r(x))$ and assume

$$key(x) < key(z) < key(y)$$

- otherwise: let $u = a(x,z)$

$$key(x) < key(z) \ \rightarrow x \in L(u) \quad \text{lemma 2}$$

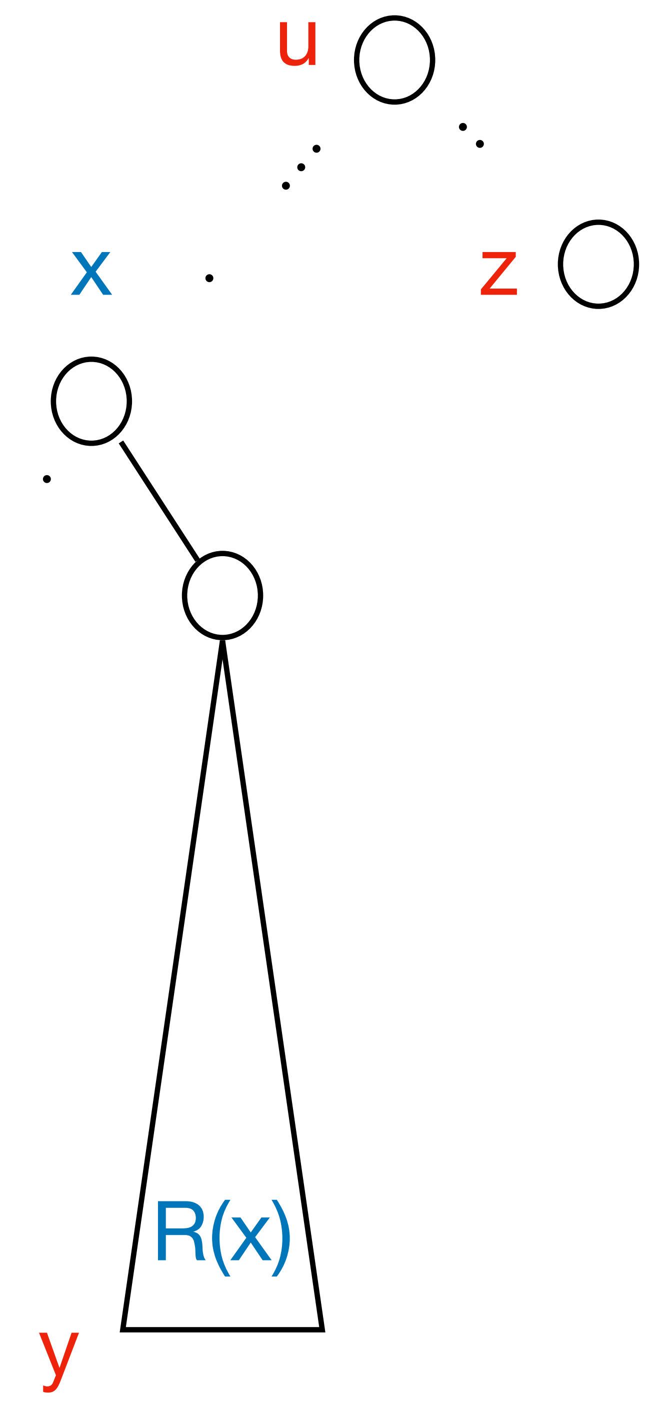
$$\rightarrow y \in L(u) \quad \& \quad z \in R(u)$$

$$\rightarrow key(y) < key(u) < key(z) \quad \text{BST-condition, both parts}$$

BST-property:

$z \in L(x) \rightarrow key(z) < key(x)$

$z \in R(x) \rightarrow key(z) > key(x)$



binary search trees (BST's): successor

- input: node x
- output:

$$succ(x) = \begin{cases} y \in T \text{ with } key(y) = \min\{key(z) \mid z \in T\ ,\ key(z) > key(y)\} & \text{if it exists} \\ NULL & \text{otherwise} \end{cases}$$

```
if x.r != Null {return min(r(x))} else
{let i = min {j>=0 | /isl(p^j(x);   /*parent chasing of right sons*/
u = p^i(x);
return (isroot(u)? NULL:p(u))
}
```

correctness: $x.r = null$

by contradiction. Let $y = p(u)$ and assume

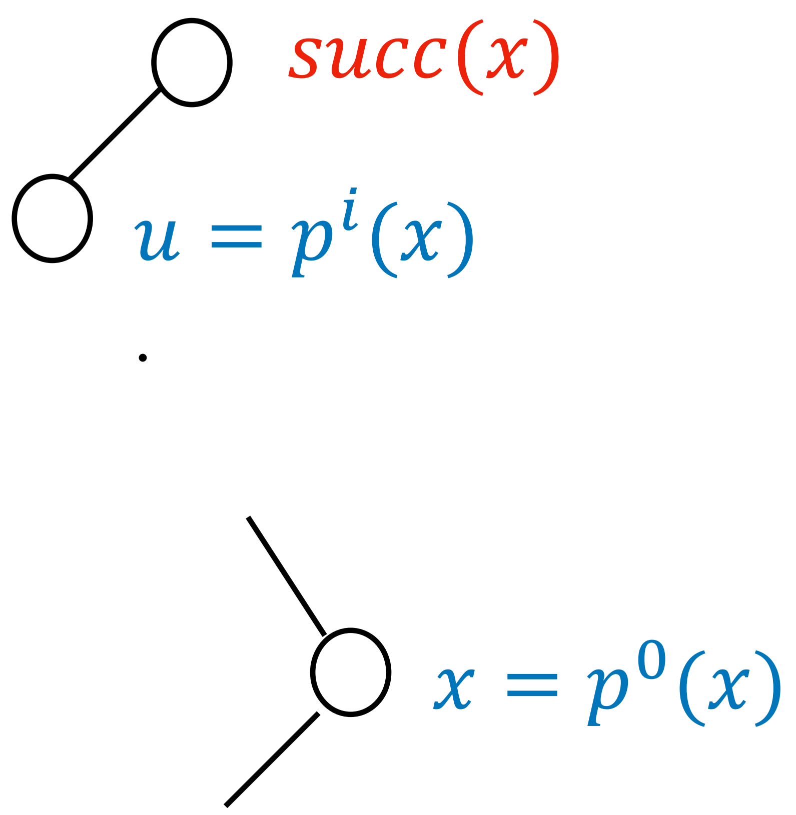
$$key(x) < key(z) < key(y)$$

BST-property:

$z \in L(x) \rightarrow key(z) < key(x)$

$z \in R(x) \rightarrow key(z) > key(x)$

x



binary search trees (BST's): successor

BST-property:

$z \in L(x) \rightarrow key(z) < key(x)$

$z \in R(x) \rightarrow key(z) > key(x)$

- input: node x
- output:

$$succ(x) = \begin{cases} y \in T \text{ with } key(y) = \min\{key(z) \mid z \in T\ ,\ key(z) > key(y)\} & \text{if it exists} \\ NULL & \text{otherwise} \end{cases}$$

```
if x.r != Null {return min(r(x))} else
{let i = min {j>=0 | /isl(p^j(x));  /*parent chasing of right sons*/
u = p^i(x);
return (isroot(u)? NULL:p(u))
}
```

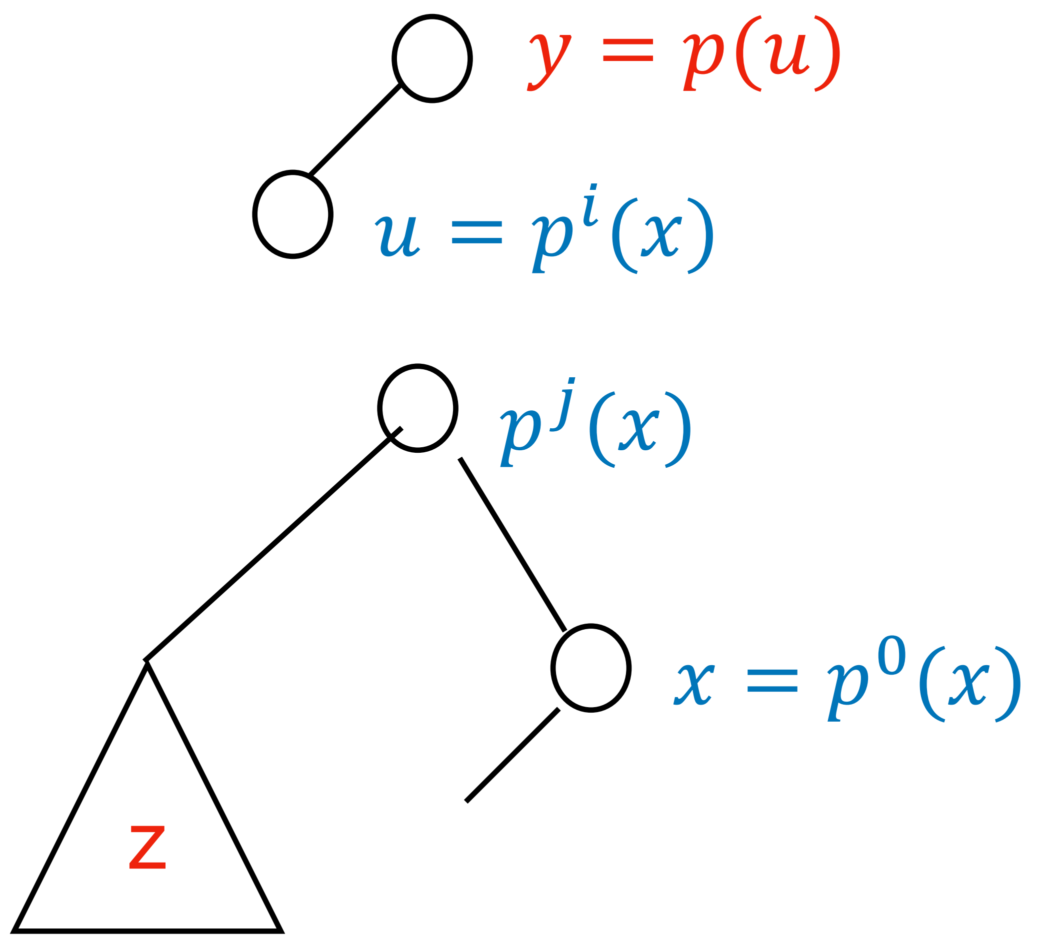
correctness: $x.r = null$

by contradiction. Let $y = p(u)$ and assume

$$key(x) < key(z) < key(y)$$

- $\exists j.\ 0 < j \leq i\ \wedge\ z \in T(p^j(x))$

$$key(z) \leq key(p^j(x)) < key(x) \quad \text{BST-condition, both cases}$$



binary search trees (BST's): successor

BST-property:

$z \in L(x) \rightarrow key(z) < key(x)$

$z \in R(x) \rightarrow key(z) > key(x)$

- input: node x
- output:

$$succ(x) = \begin{cases} y \in T \text{ with } key(y) = \min\{key(z) \mid z \in T\ ,\ key(z) > key(y)\} & \text{if it exists} \\ NULL & \text{otherwise} \end{cases}$$

```
if x.r != Null {return min(r(x))} else
{let i = min {j>=0 | /isl(p^j(x));  /*parent chasing of right sons*/
u = p^i(x);
return (isroot(u)? NULL:p(u))
}
```

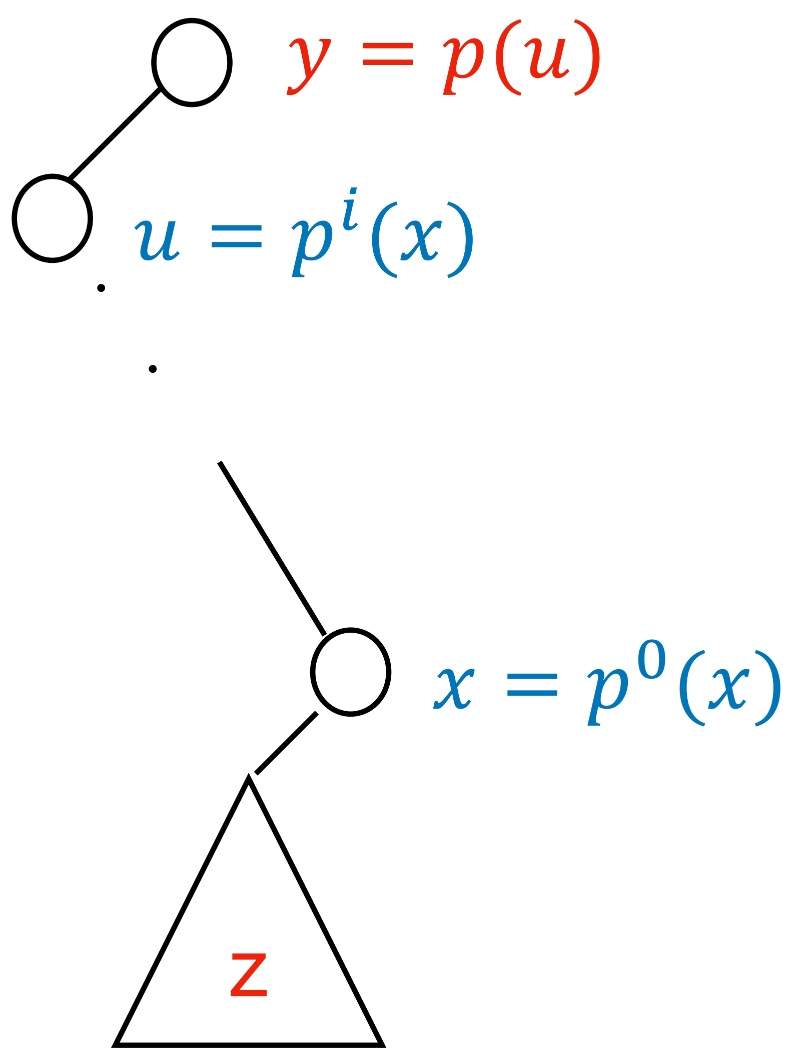
correctness: $x.r = null$

by contradiction. Let $y = p(u)$ and assume

$$key(x) < key(z) < key(y)$$

- $z \in T(x)$, as $x.r = \text{null}$

  $\rightarrow z \in L(x) \rightarrow key(z) < key(x)$ BST-condition, part 1



binary search trees (BST's): successor

- input: node x
- output:

$$succ(x) = \begin{cases} y \in T \text{ with } key(y) = \min\{key(z) \mid z \in T \ , key(z) > key(y)\} & \text{if it exists} \\ NULL & \text{otherwise} \end{cases}$$

```
if x.r != Null {return min(r(x))} else
{let i = min {j>=0 | /isℓ(p^j(x);  /*parent chasing of right sons*/
u = p^i(x);
return (isroot(u)? NULL:p(u))
}
```

correctness: $x.r = null$

by contradiction. Let $y = p(u)$ and assume

$$key(x) < key(z) < key(y)$$

- $y \in T(z)$

$$key(z) < key(y) \rightarrow y \in R(z) \quad \text{BST-condition, part 1}$$

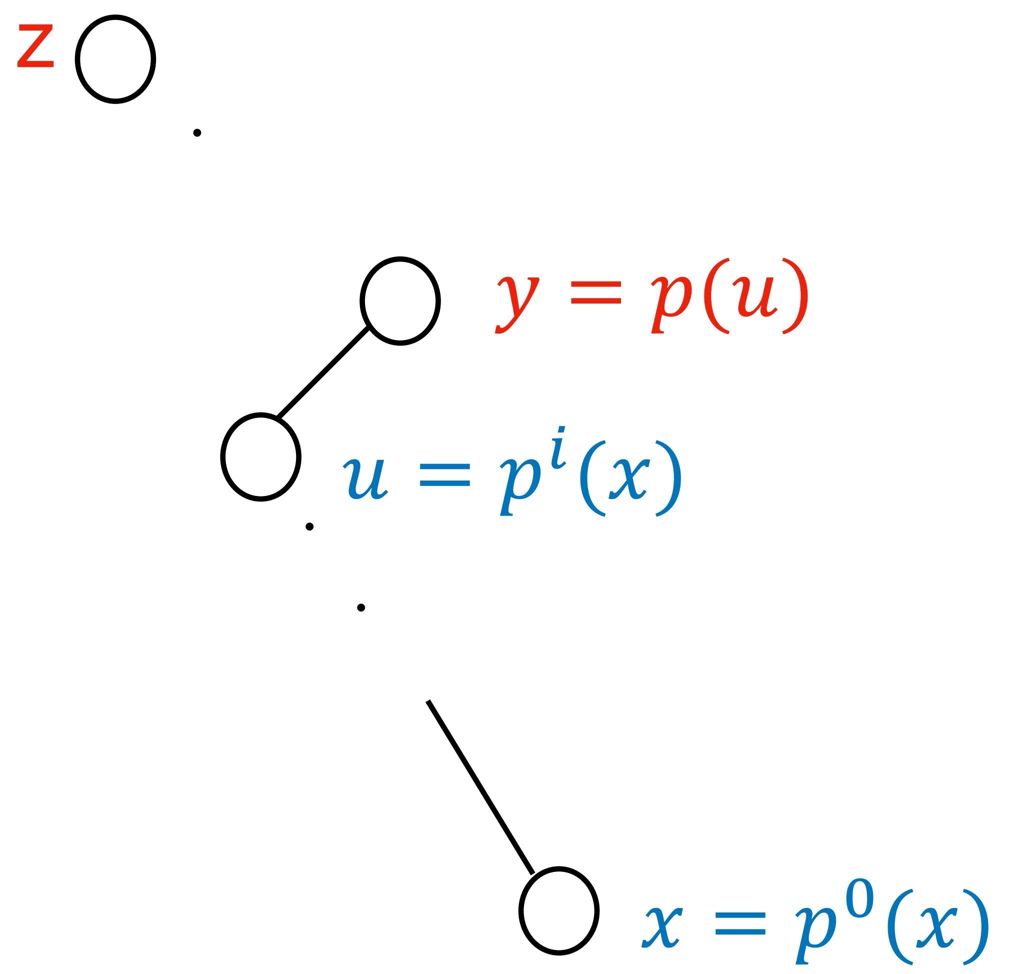
$$\rightarrow x \in R(z)$$

$$\rightarrow key(z) < key(x) \quad \text{BST-condition, part 2}$$

BST-property:

$z \in L(x) \rightarrow key(z) < key(x)$

$z \in R(x) \rightarrow key(z) > key(x)$



binary search trees (BST's): successor

- input: node x
- output:

$$succ(x) = \begin{cases} y \in T \text{ with } key(y) = \min\{key(z) \mid z \in T \ , \ key(z) > key(y)\} & \text{if it exists} \\ NULL & \text{otherwise} \end{cases}$$

```
if x.r != Null {return min(r(x))} else
{let i = min {j>=0 | /isl(p^j(x));   /*parent chasing of right sons*/
u = p^i(x);
return (isroot(u)? NULL:p(u))
}
```

BST-property:

$z \in L(x) \rightarrow key(z) < key(x)$

$z \in R(x) \rightarrow key(z) > key(x)$

correctness: $x.r = null$

by contradiction. Let $y = p(u)$ and assume

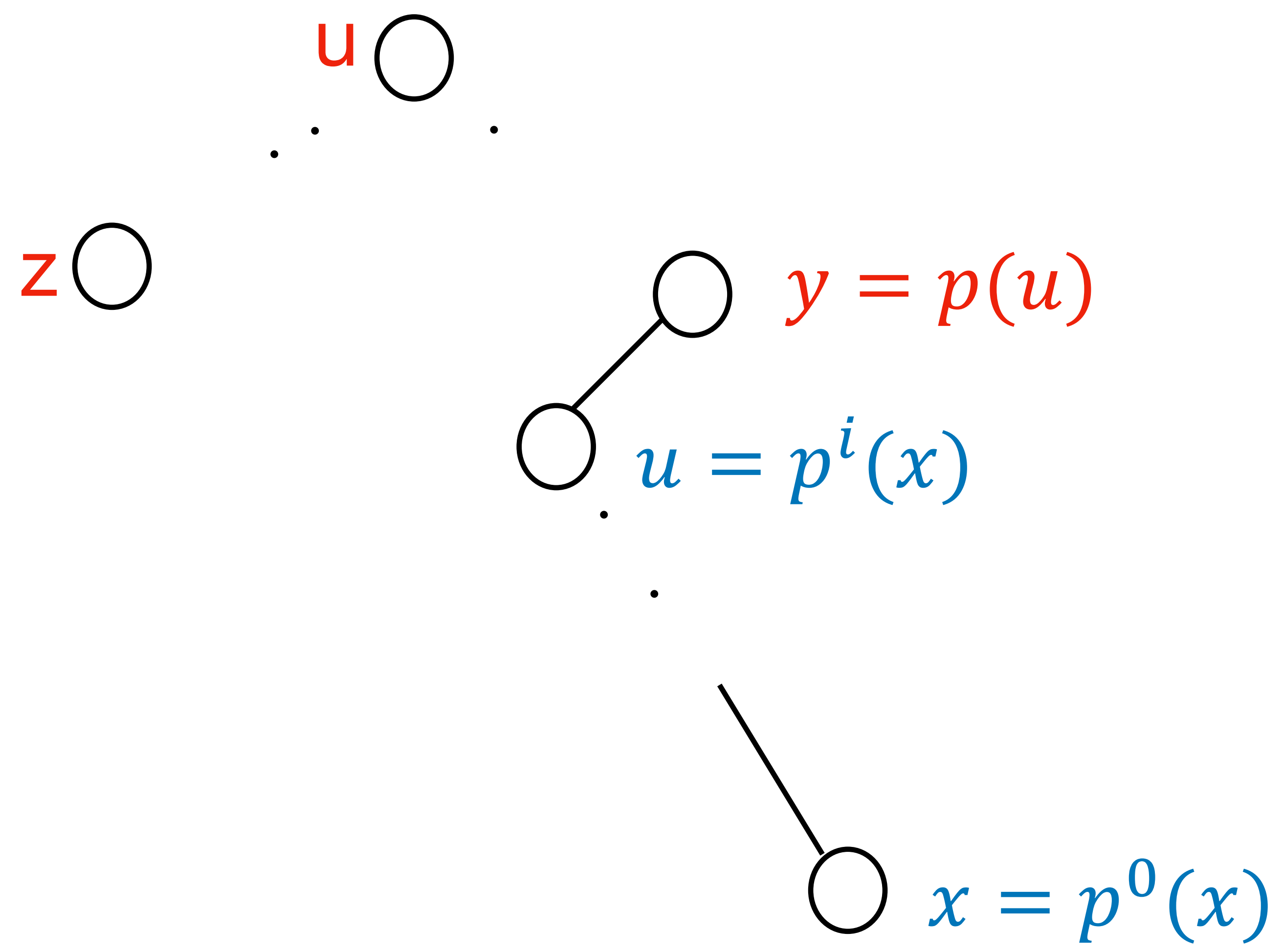
$$key(x) < key(z) < key(y)$$

- $y \notin T(z) \ \wedge \ z \notin T(y)$. Let $u = a(y,z)$.

$$key(z) < key(y) \rightarrow z \in L(u) \ \wedge y \in R(u) \quad \text{lemma 2}$$

$$\rightarrow x \in R(u)$$

$$\rightarrow key(z) < key(u) < key(x) \quad \text{BST-condition, both parts}$$



binary search trees (BST's): successor

- input: node x
- output:

$$succ(x) = \begin{cases} y \in T \text{ with } key(y) = \min\{key(z) \mid z \in T\ ,\ key(z) > key(y)\} & \text{if it exists} \\ NULL & \text{otherwise} \end{cases}$$

```
if x.r != Null {return min(r(x))} else
{let i = min {j>=0 | /isl(p^j(x));   /*parent chasing of right sons*/
u = p^i(x);
return (isroot(u)? NULL:p(u))
}
```

correctness: $x.r = null$

by contradiction. Let $y = p(u)$ and assume

$$key(x) < key(z) < key(y)$$

- $y \notin T(z) \wedge z \notin T(y)$. Let $u = a(y,z)$.

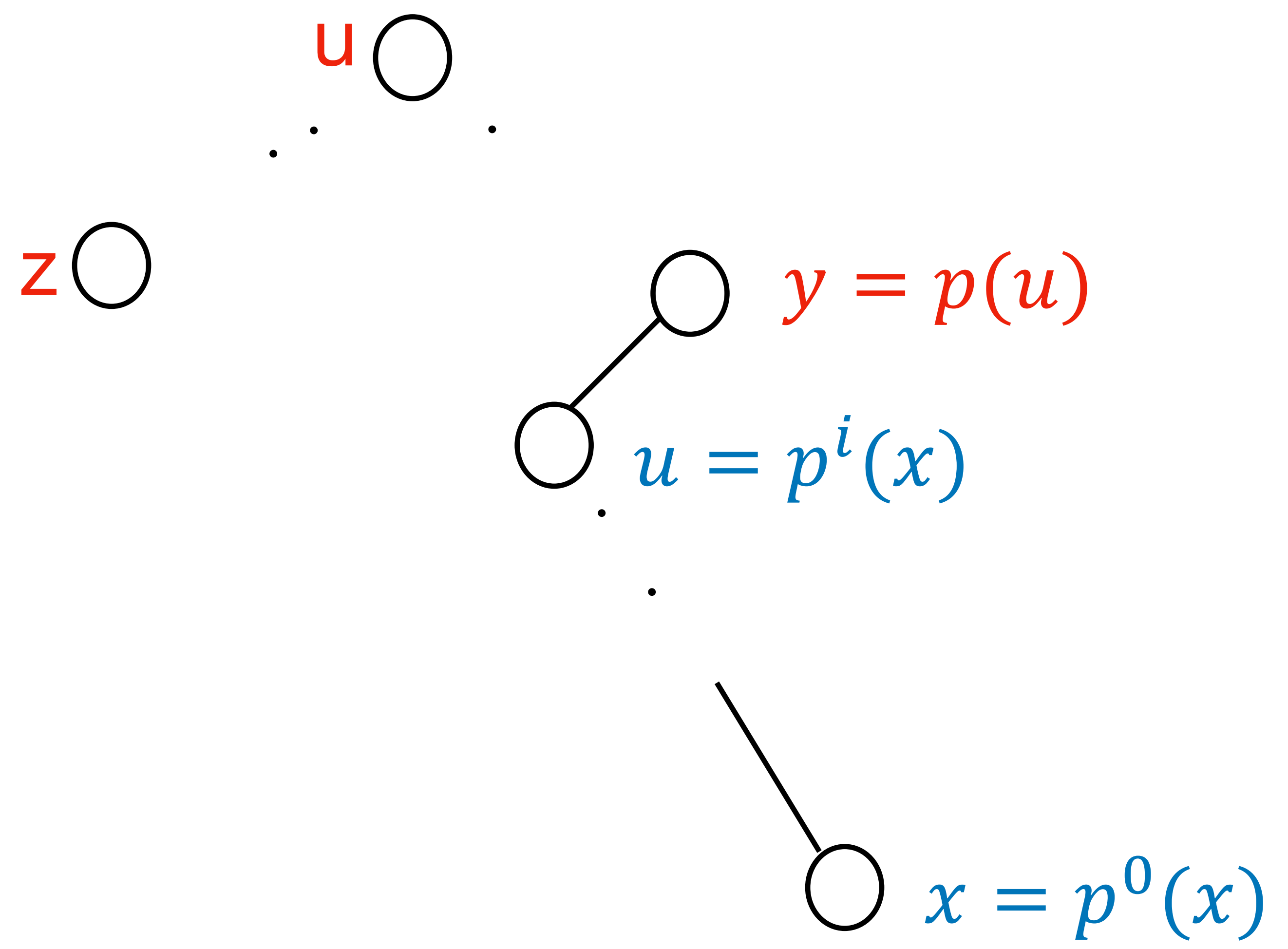
$$key(z) < key(y) \to z \in L(u) \wedge y \in R(u) \quad \text{lemma 2}$$
$$\to x \in R(u)$$
$$\to key(z) < key(u) < key(x) \quad \text{BST-condition, both parts}$$

time O(h(T))

BST-property:

$z \in L(x) \to key(z) < key(x)$

$z \in R(x) \to key(z) > key(x)$

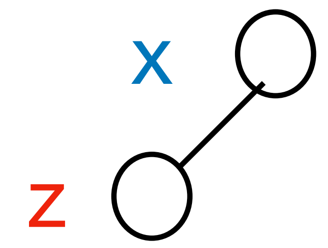


binary search trees (BST's): insert

$z \in T(x)$

- input: node $x \in T$, node$z \notin T$ with new key $k = key(z)$
- output: T with z inserted and BST-property maintained

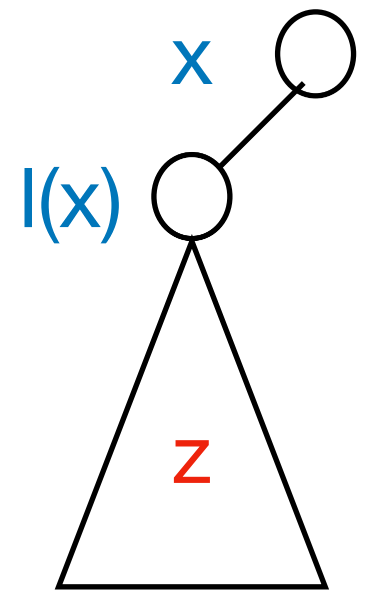
```
if key(z) < key(x) {if x.l=null {p(z) = x; x.l=z}
                             else {insert(l(x),z)}}
if key(z) > key(x) {if x.r=null {p(z) = x; x.r=z}
                             else {insert(r(x),z)}}
```



binary search trees (BST's): insert

- input: node $x \in T$, node$z \notin T$ with new key $k = key(z)$
- output: T with z inserted and BST-property maintained

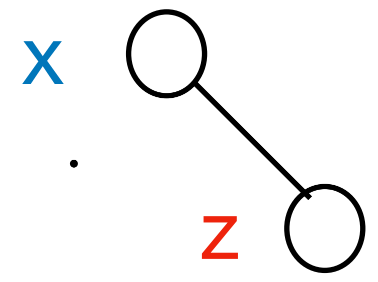
```
if key(z) < key(x) {if x.l=null {p(z) = x; x.l=z}
                              else {insert(l(x),z)}}
if key(z) > key(x) {if x.r=null {p(z) = x; x.r=z}
                              else {insert(r(x),z)}}
```



binary search trees (BST's): insert

- input: node $x \in T$, node$z \notin T$ with new key $k = key(z)$
- output: T with z inserted and BST-property maintained

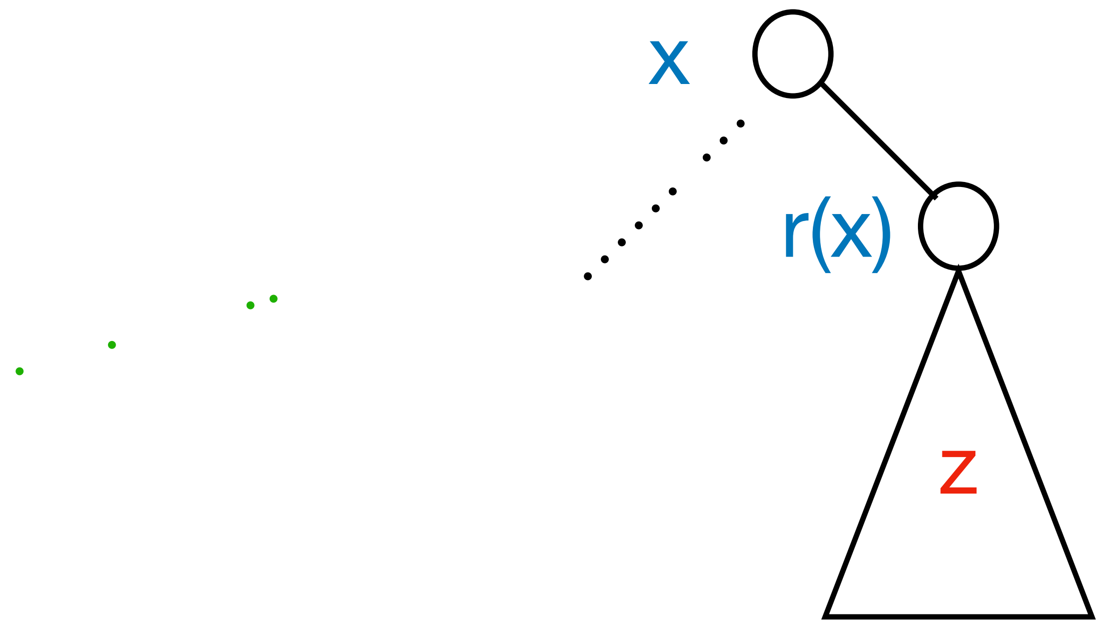
```
if key(z) < key(x) {if x.l=null {p(z) = x; x.l=z}
                              else {insert(l(x),z)}}
if key(z) > key(x) {if x.r=null {p(z) = x; x.r=z}
                              else {insert(r(x),z)}}
```



binary search trees (BST's): insert

- input: node $x \in T$, node$z \notin T$ with new key $k = key(z)$
- output: T with z inserted and BST-property maintained

```
if key(z) < key(x) {if x.l=null {p(z) = x; x.l=z}
                            else {insert(l(x),z)}}
if key(z) > key(x) {if x.r=null {p(z) = x; x.r=z}
                            else {insert(r(x),z)}}
```

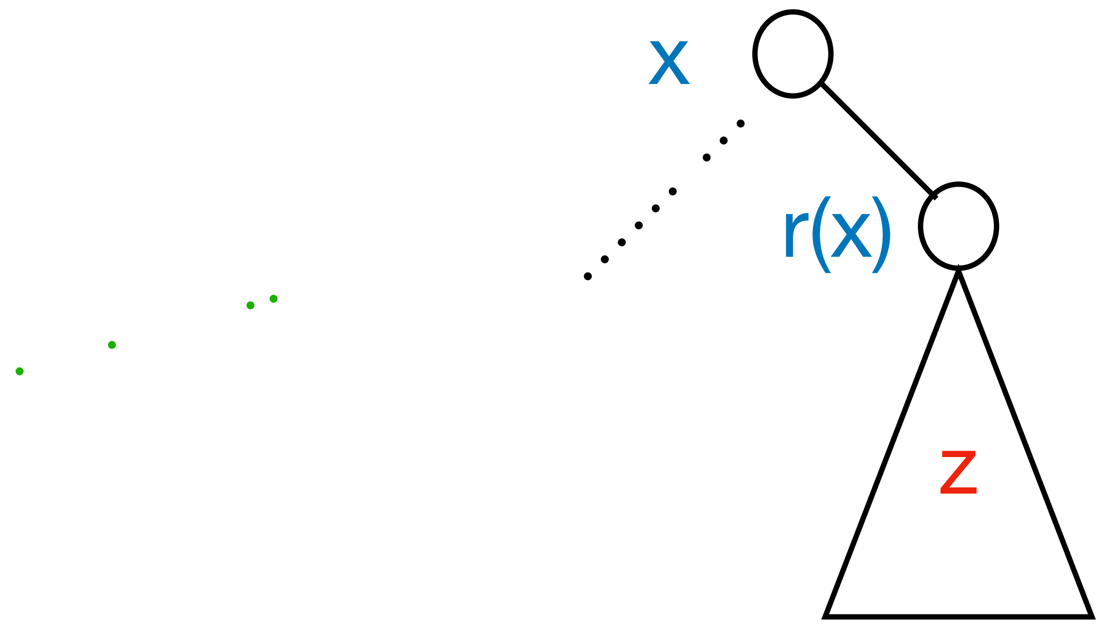


binary search trees (BST's): insert

- input: node $x \in T$, node$z \notin T$ with new key $k = key(z)$
- output: T with z inserted and BST-property maintained

```
if key(z) < key(x) {if x.l=null {p(z) = x; x.l=z}
                              else {insert(l(x),z)}}
if key(z) > key(x) {if x.r=null {p(z) = x; x.r=z}
                              else {insert(r(x),z)}}
```

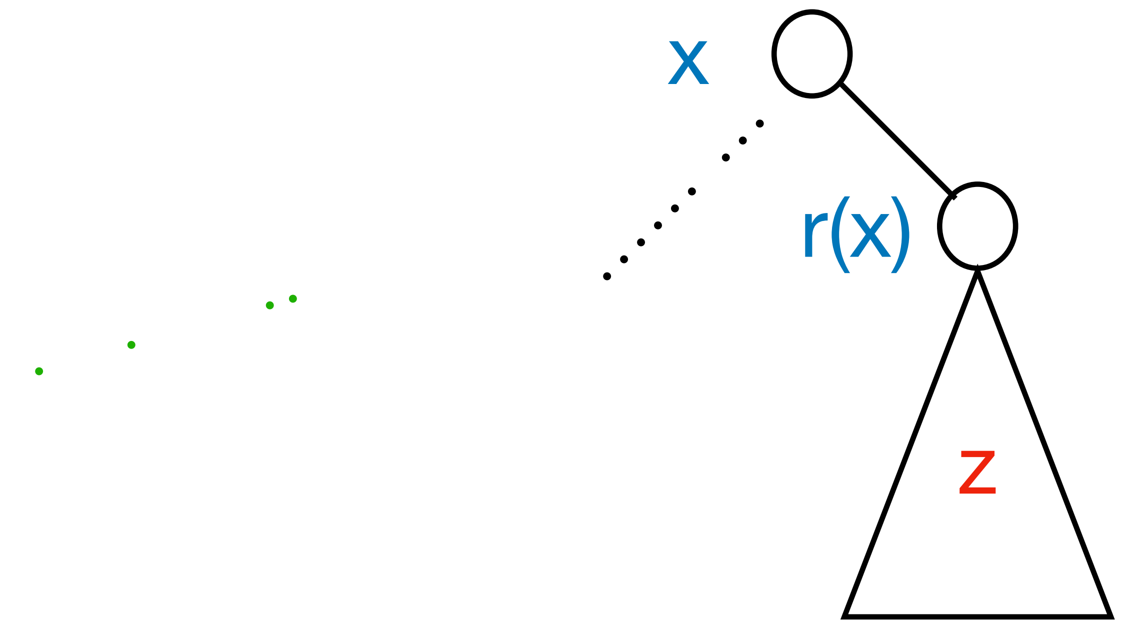
time O(h(x))



binary search trees (BST's): insert

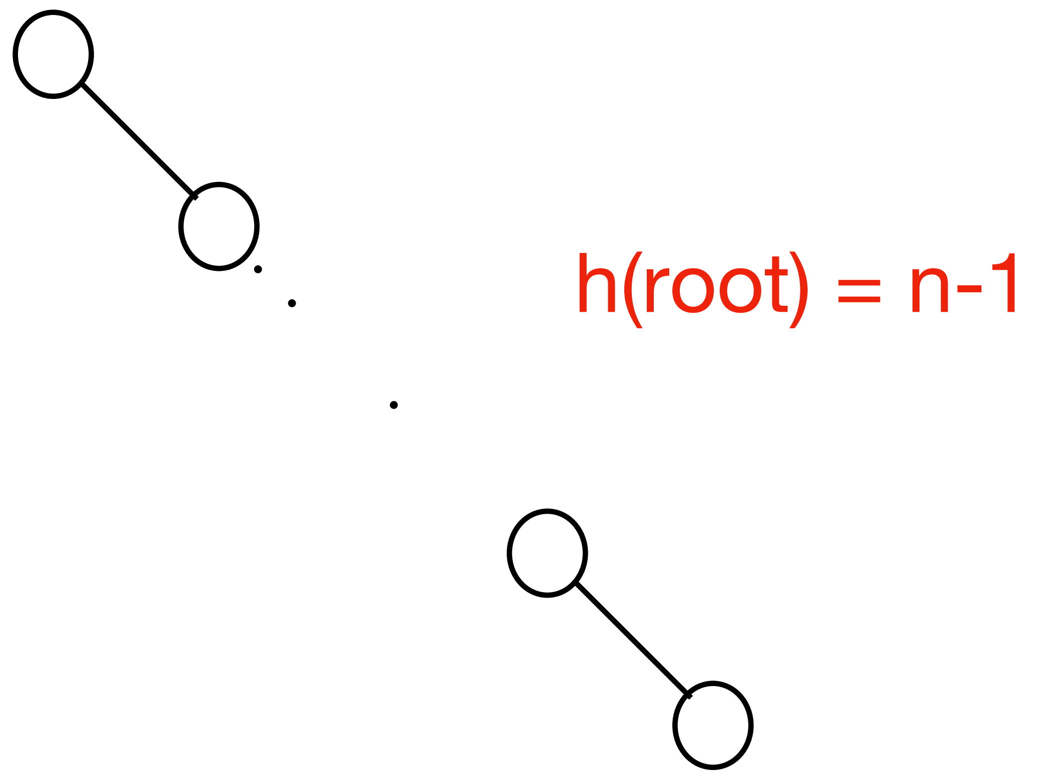
- input: node $x \in T$, node$z \notin T$ with new key $k = key(z)$
- output: T with z inserted and BST-property maintained

```
if key(z) < key(x) {if x.l=null {p(z) = x; x.l=z}
                             else {insert(l(x),z)}}
if key(z) > key(x) {if x.r=null {p(z) = x; x.r=z}
                             else {insert(r(x),z)}}
```



time O(h(x))

problem: insert n nodes with keys in increasing order



binary search trees (BST's): delete

- input: node $x \in T$
- output: T with x deleted and BST-property maintained

binary search trees (BST's): delete

- input: node $x \in T$
- output: T with x deleted and BST-property maintained

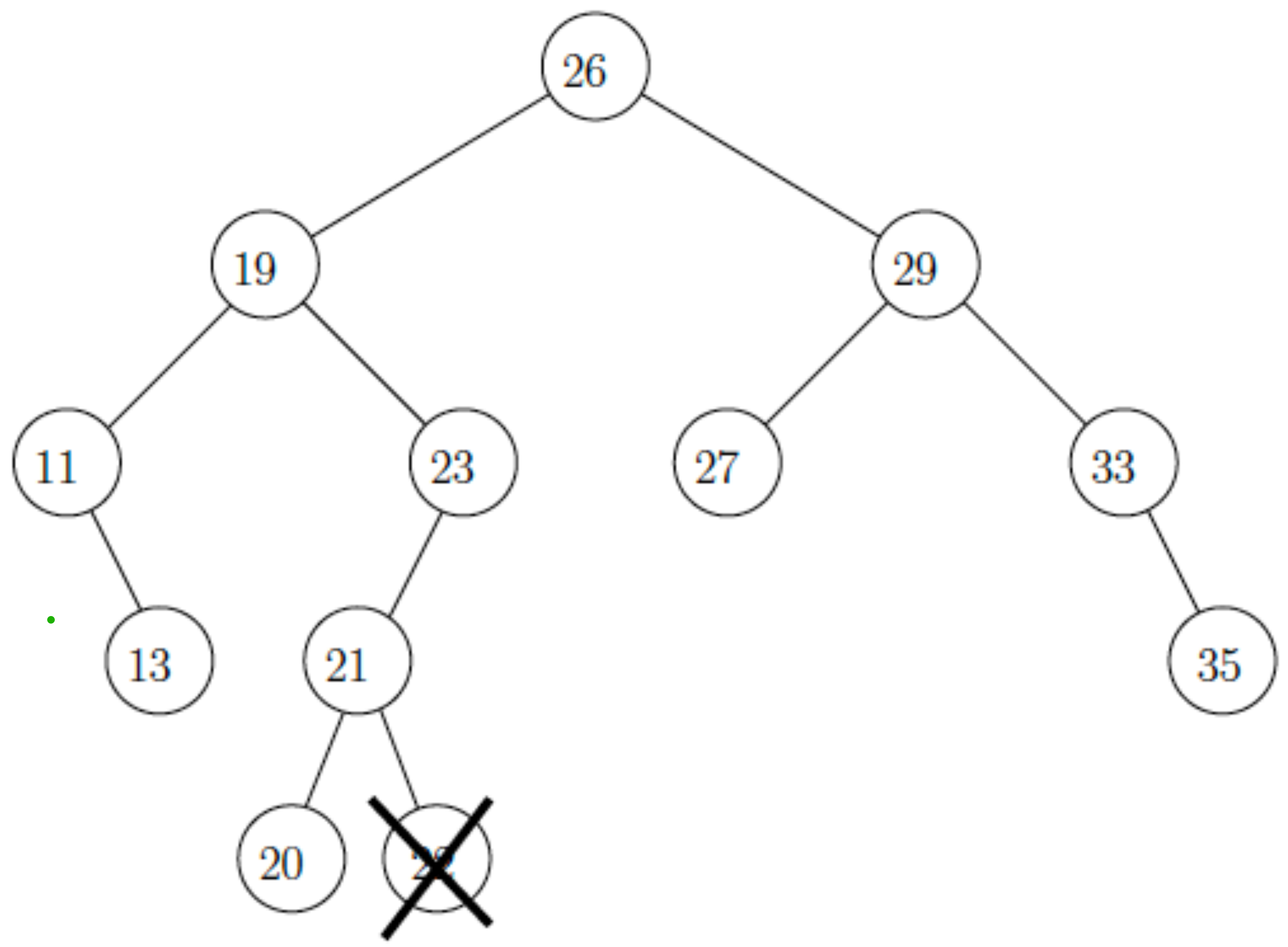
```
1.if isleaf(x)
  {if x = l(p(x)) {p(x).l = null} else p(x).r = null}}
        /* drop it, done*/

2.a. if x.l != 0 null & x.r = null& !isroot(x)
     /* x has left son, no right son, is not root*/
         {l(x).p = p(x};
         /*hook son to paent of x*/
2.b  if x.l != null & x.r = null& isroot(x)
/* x has left son, no right son, is root*/
        {root= l(x);  p(l(x))=null };
        /* son becomes new root*/

2.c    x has right son, no left son: analogous
2.d

3. x.l != null & x.r !=0
/* x has 2 sons. then y = succ(x) is a leaf */
{ y = succ(x);
key(x) = key(y)  /* move successor to place of x*/;
{if y = l(p(y)) {p(y).l = null} else p(y).r = null}}
/*drop y as in case 1* or 2
```

has at most one son as succ(x) = min R(x).



binary search trees (BST's): delete

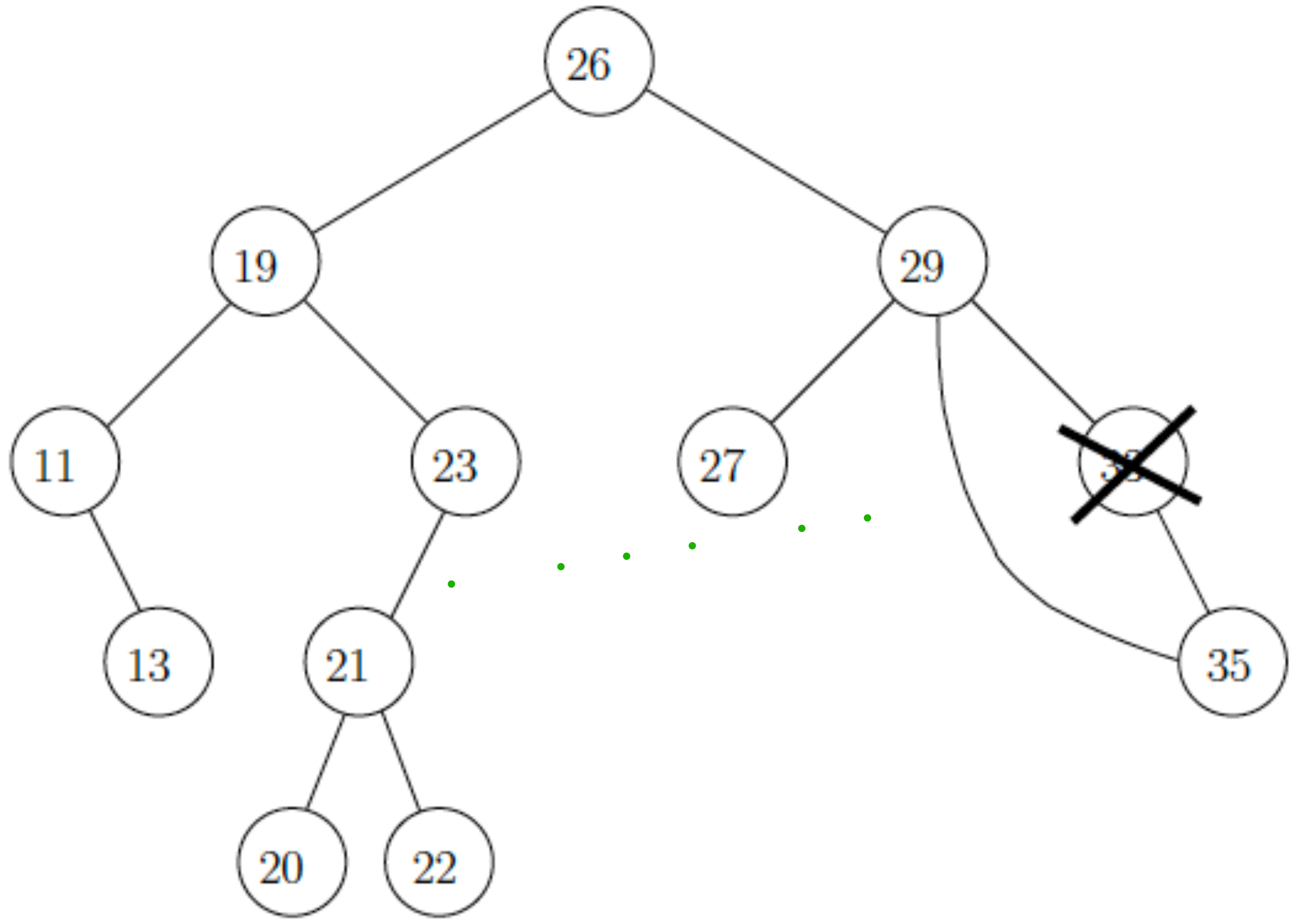
- input: node $x \in T$
- output: T with x deleted and BST-property maintained

```
1.if isleaf(x)
  {if x = l(p(x)) {p(x).l = null} else p(x).r = null}}
        /* drop it, done*/

2.a. if x.l != 0 null & x.r = null& !isroot(x)
     /* x has left son, no right son, is not root*/
         {l(x).p = p(x};
         /*hook son to paent of x*/
2.b  if x.l != null & x.r = null& ~~!~~isroot(x)
/* x has left son, no right son, is root*/
        {root= l(x);  p(l(x))=null };
        /* son becomes new root*/

2.c    x has right son, no left son: analogous
2.d

3. x.l != null & x.r !=0              has at most one son
/* x has 2 sons. then y = succ(x) ~~is a leaf */~~
{ y = succ(x);
key(x) = key(y)  /* move successor to place of x*/;
~~{if y = l(p(y)) {p(y).l = null} else p(y).r = null}}~~
/*drop y as in case 1*/ or 2.
```



binary search trees (BST's): delete

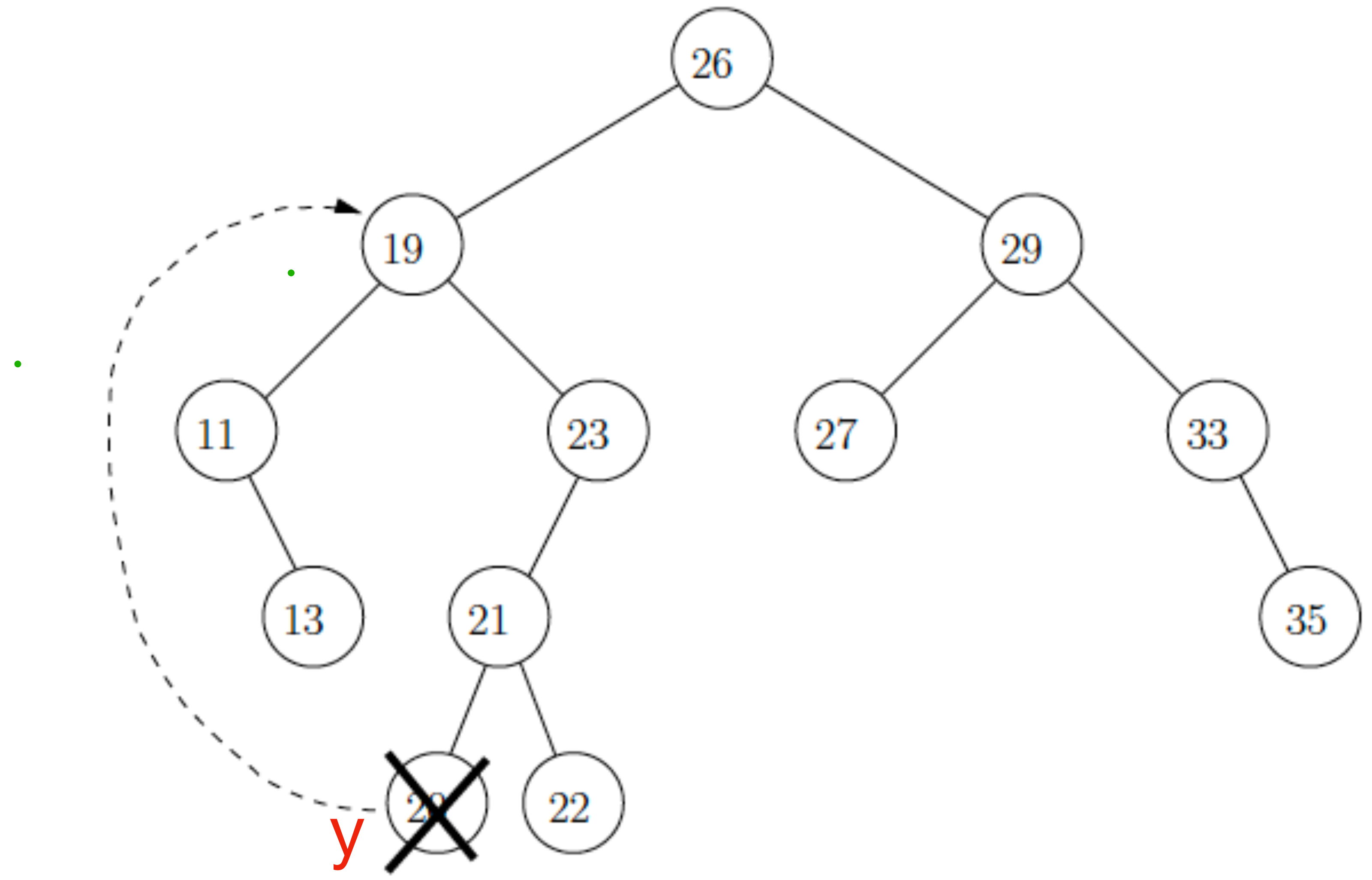
- input: node $x \in T$
- output: T with x deleted and BST-property maintained

```
1.if isleaf(x)
  {if x = l(p(x)) {p(x).l = null} else p(x).r = null}}
        /* drop it, done*/

2.a. if x.l != 0 null & x.r = null& !isroot(x)
     /* x has left son, no right son, is not root*/
         {l(x).p = p(x)};
         /*hook son to paent of x*/
2.b  if x.l != null & x.r = null& isroot(x)
/* x has left son, no right son, is root*/
        {root= l(x);  p(l(x))=null };
        /* son becomes new root*/

2.c    x has right son, no left son: analogous
2.d

3. x.l != null & x.r !=0
/* x has 2 sons. then y = succ(x) ~~is a leaf~~ */   has at most one child
{ y = succ(x);
key(x) = key(y)  /* move successor to place of x*/;
~~{if y = l(p(y)) {p(y).l = null} else p(y).r = null}}~~
/*drop y as in case ~~1~~ 2 or 1.
```



binary search trees (BST’s): delete

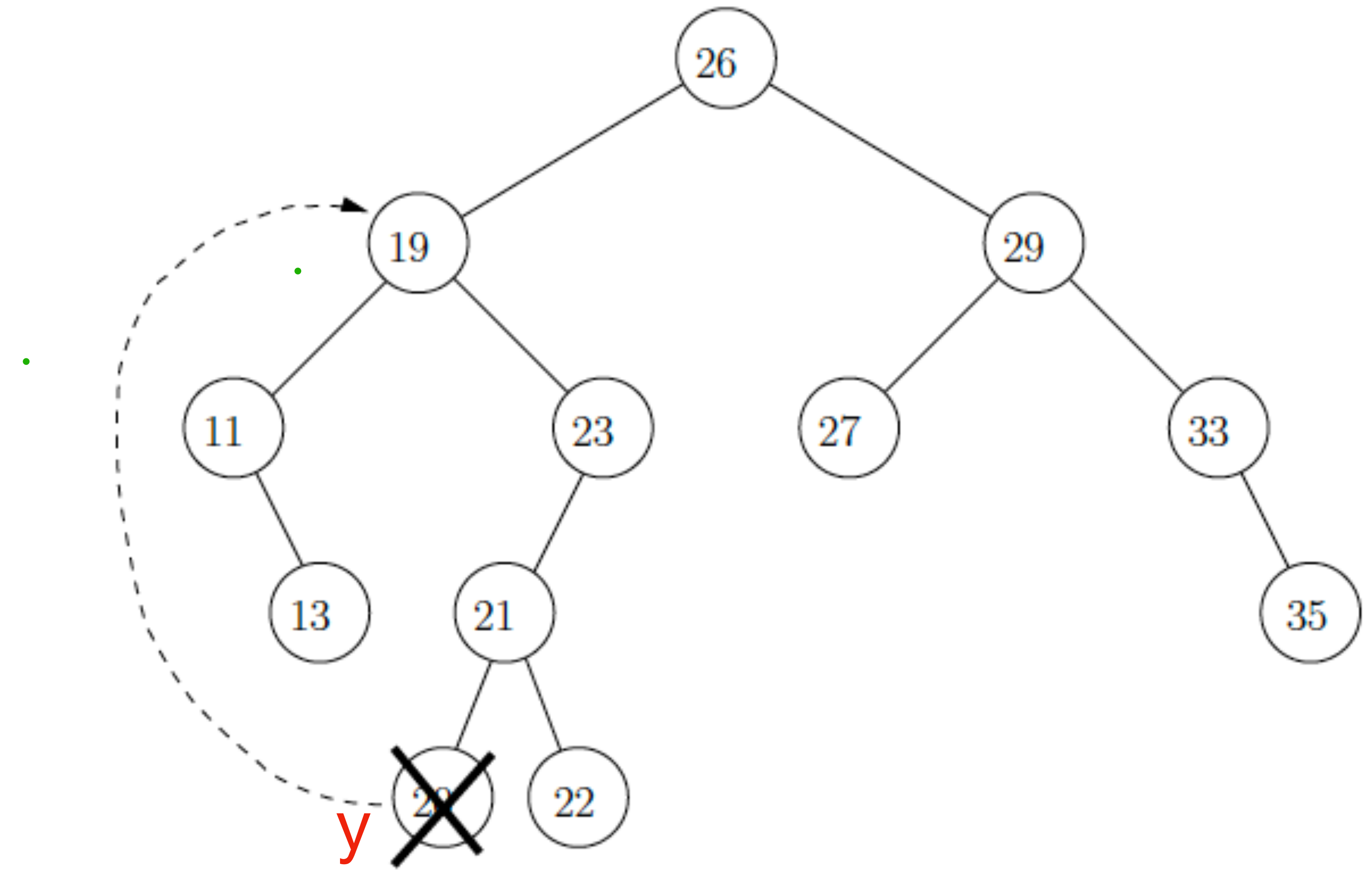
- input: node $x \in T$
- output: T with x deleted and BST-property maintained

1.if isleaf(x)
  {if x = l(p(x)) {p(x).l = null} else p(x).r = null}}
        /* drop it, done*/

2.a. if x.l != 0 null & x.r = null& !isroot(x)
     /* x has left son, no right son, is not root*/
         {l(x).p = p(x)};
         /*hook son to paent of x*/
2.b if x.l != null & x.r = null& isroot(x)
/* x has left son, no right son, is root*/
        {root= l(x); p(l(x))=null };
        /* son becomes new root*/

2.c   x has right son, no left son: analogous
2.d

3. x.l != null & x.r !=0
/* x has 2 sons. then y = succ(x) ~~is a leaf */~~
{ y = succ(x);
key(x) = key(y) /* move successor to place of x*/;
~~{if y = l(p(y)) {p(y).l = null} else p(y).r = null}}~~
/*drop y as in case 1*/ or 2



time O(h(x))