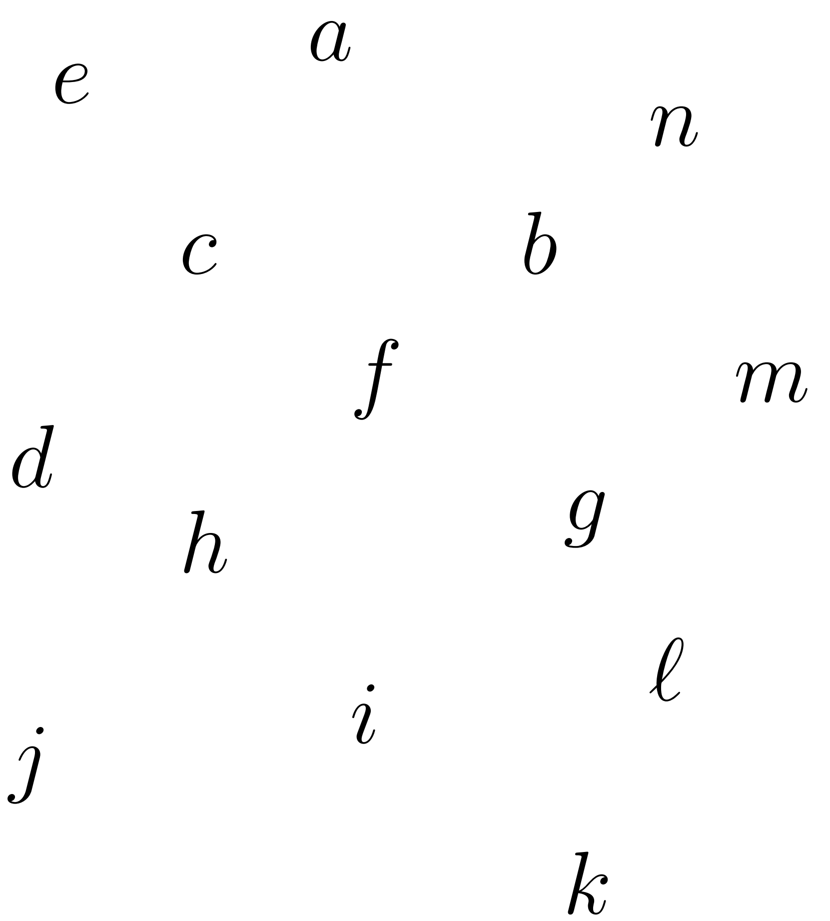
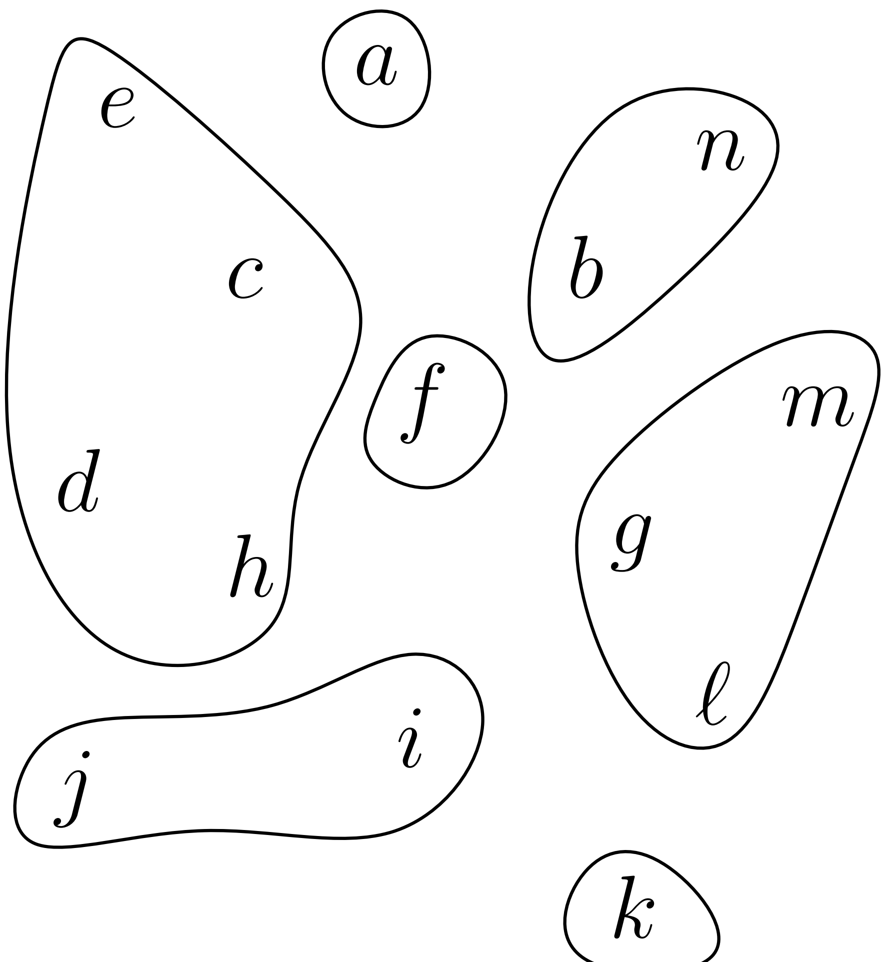
Union Find, Path Compression, and the Inverse Ackermann Function

The Data

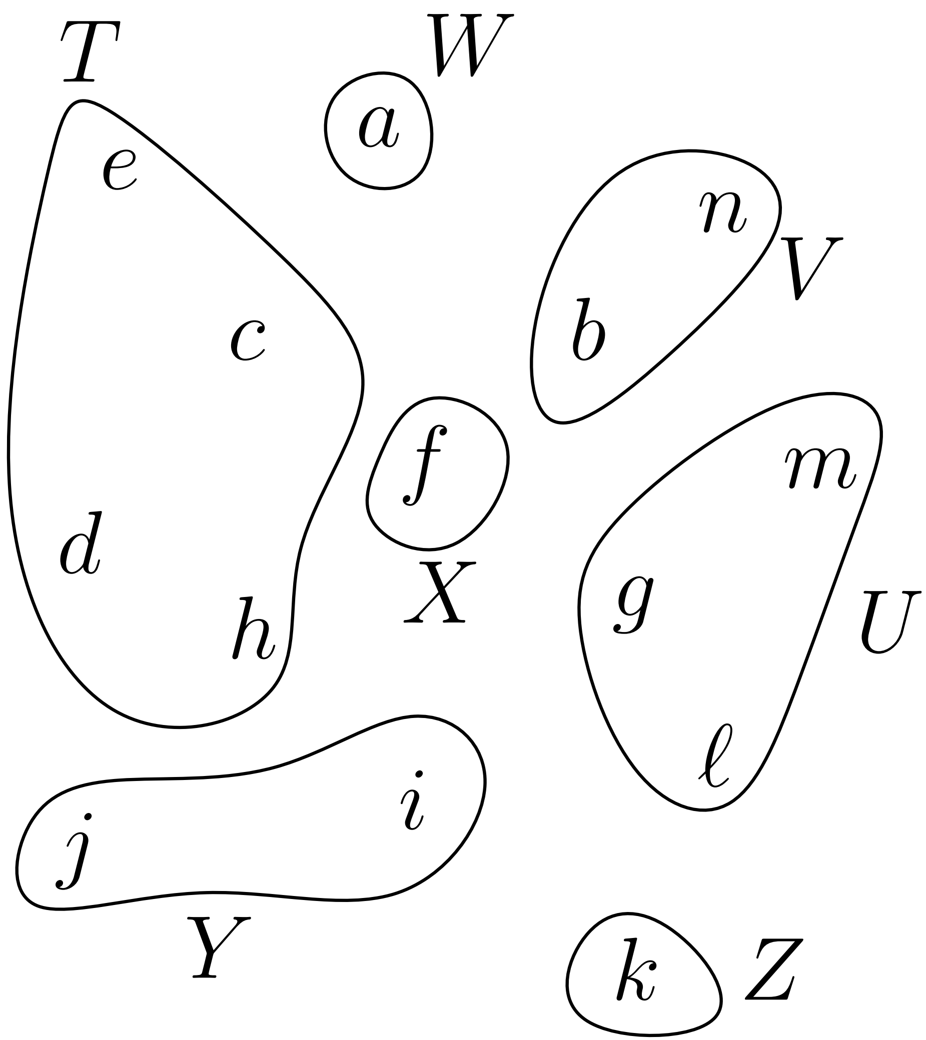
The Data



The Data

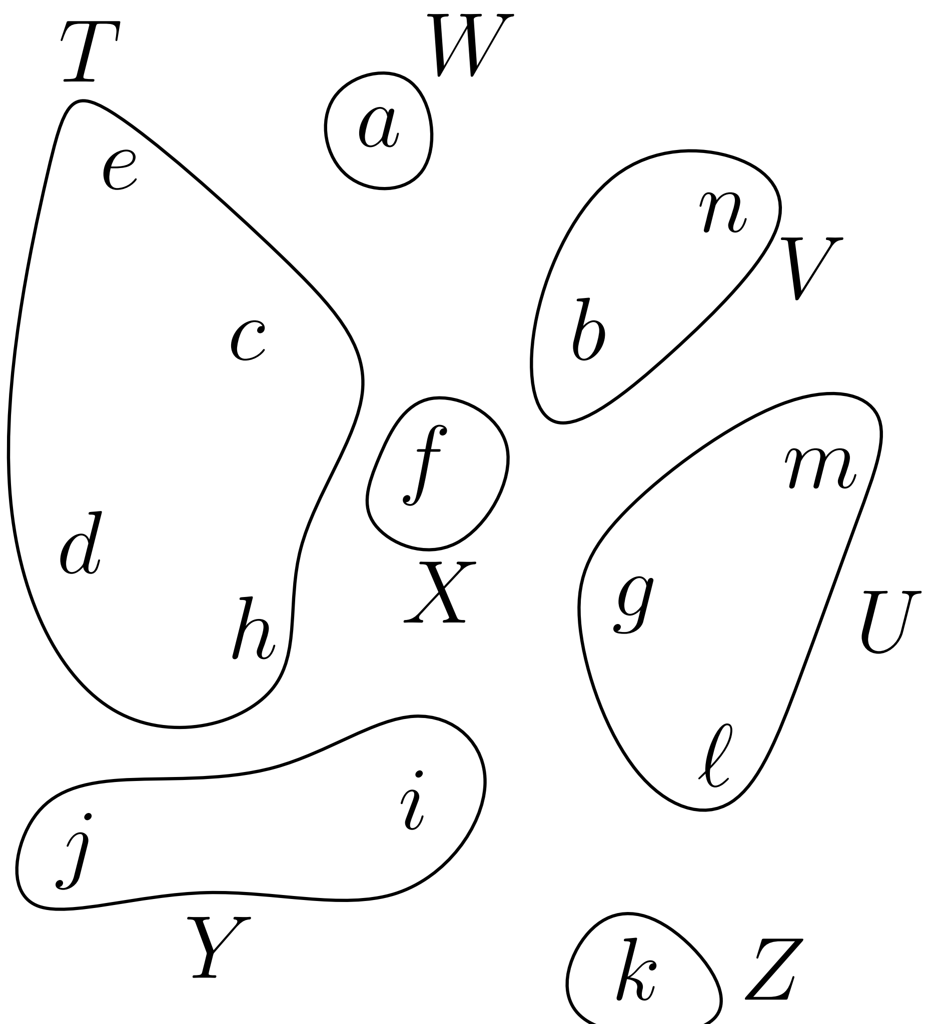


The Data

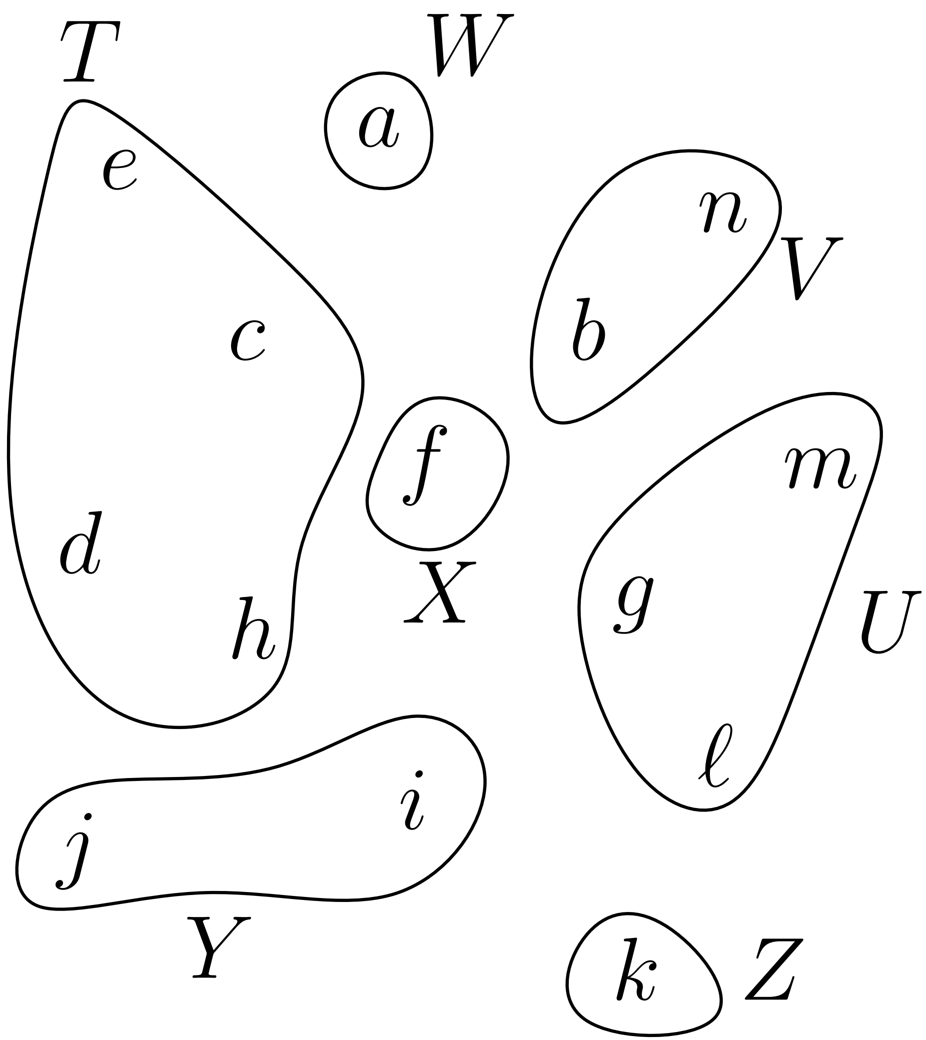


The Data

The Operations



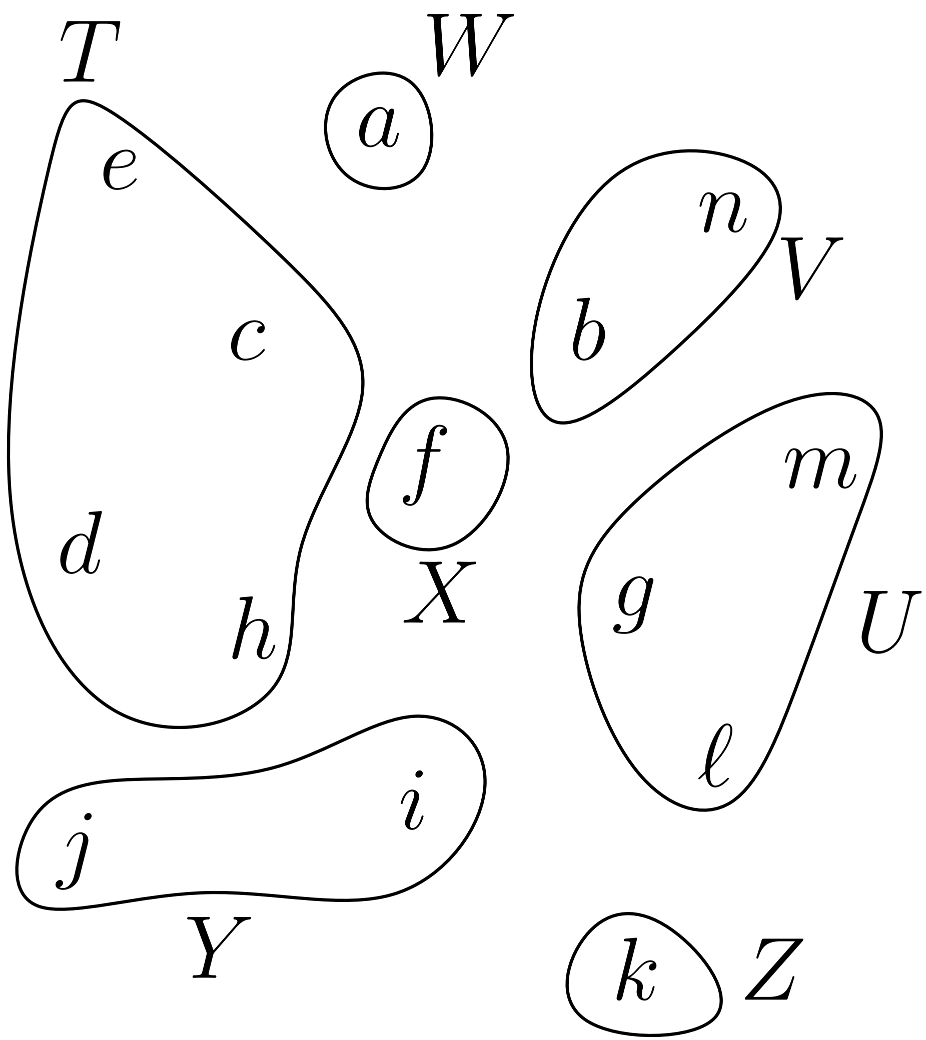
The Data



The Operations

$>>>$ `find`(h)

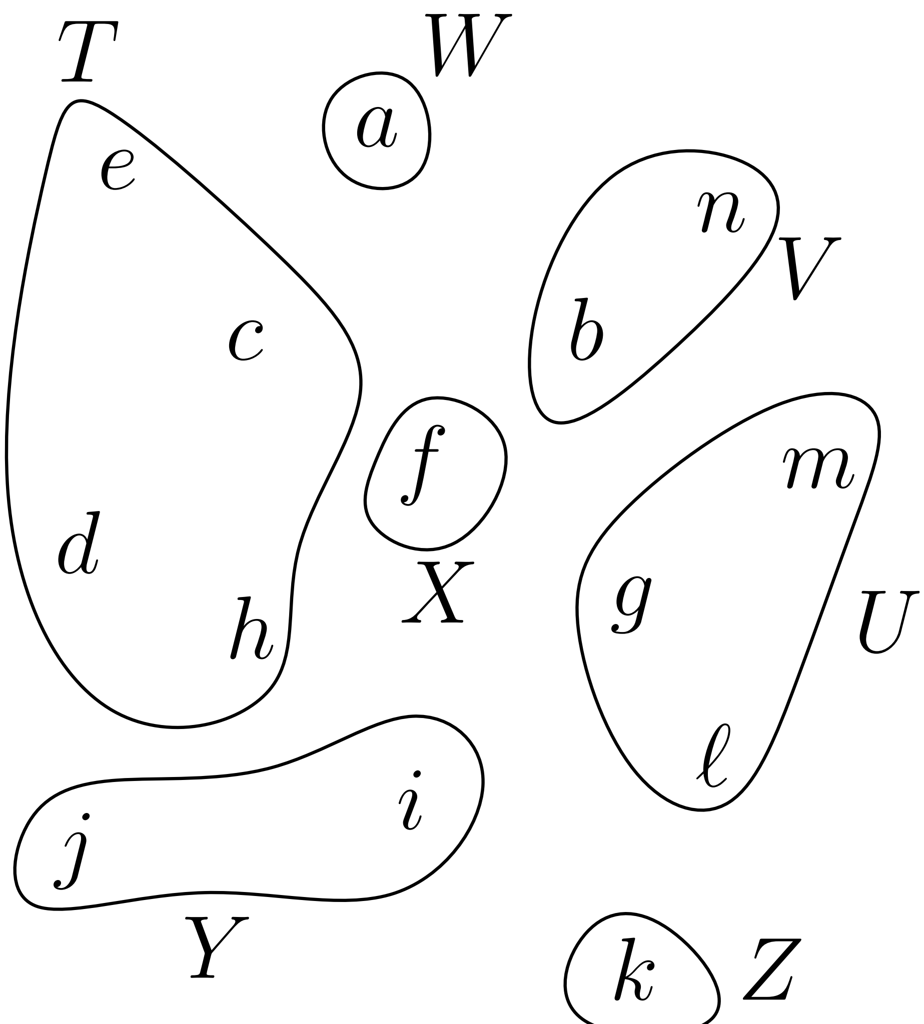
The Data



The Operations

```
>>> find(h)
T
>>>
```

The Data



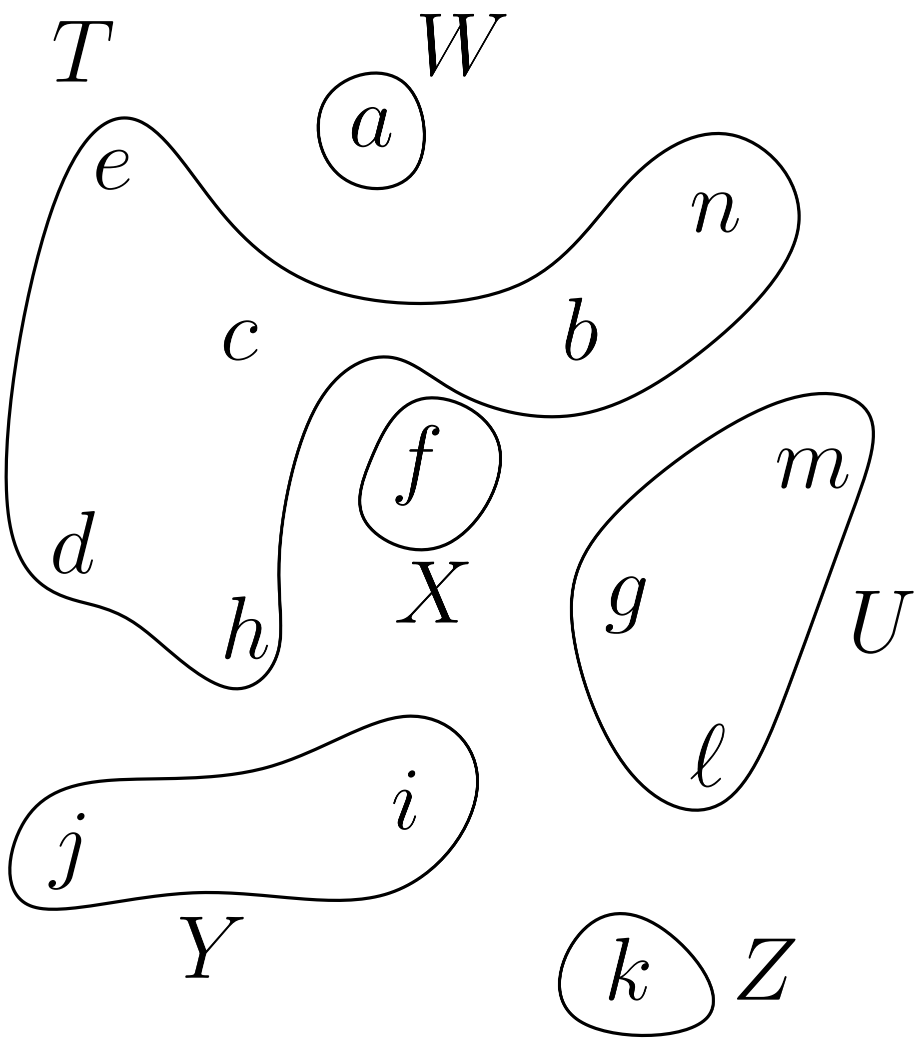
The Operations

$>>>$ `find`(h)

T

$>>>$ `union`(T, V)

The Data



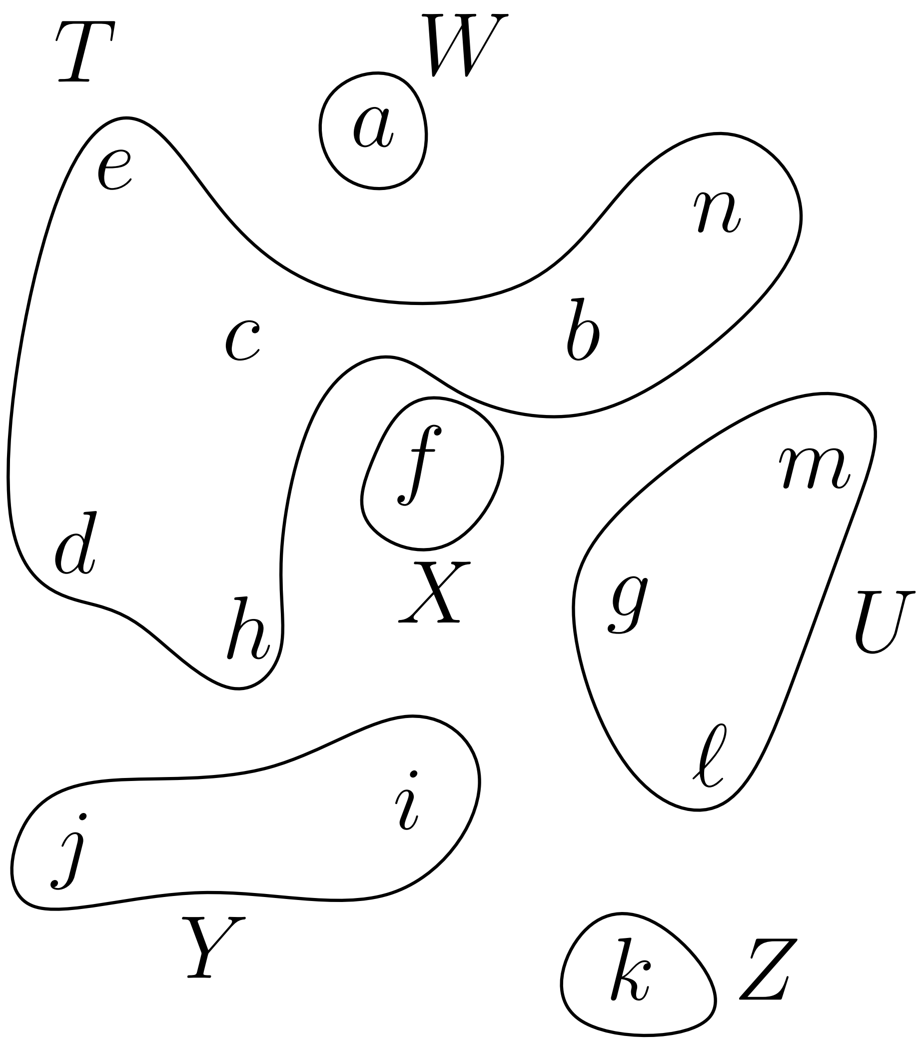
The Operations

$>>>$ `find`(h)

T

$>>>$ `union`(T, V)

The Data



The Operations

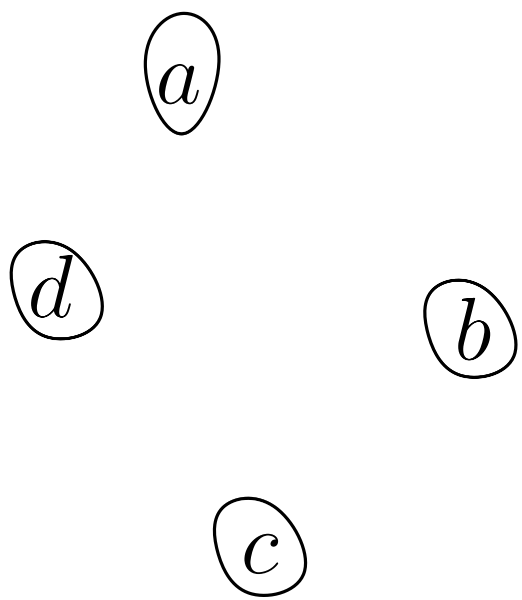
```
>>> find(h)
T
>>> union(T, V)
>>>
```

The Data

The Operations

$>>>$ `init`$(\{a, b, c, d\})$

The Data

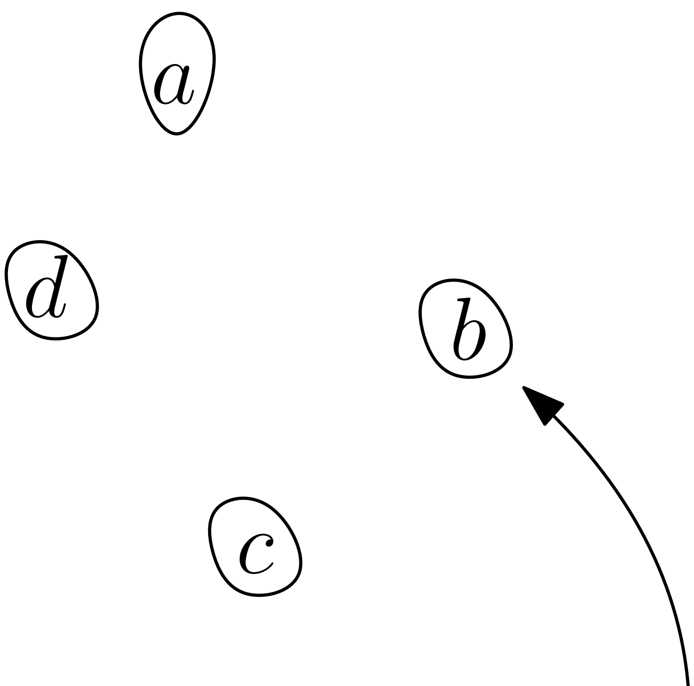


The Operations

$>>>\ \texttt{init}(\{a, b, c, d\})$

$>>>$

The Data



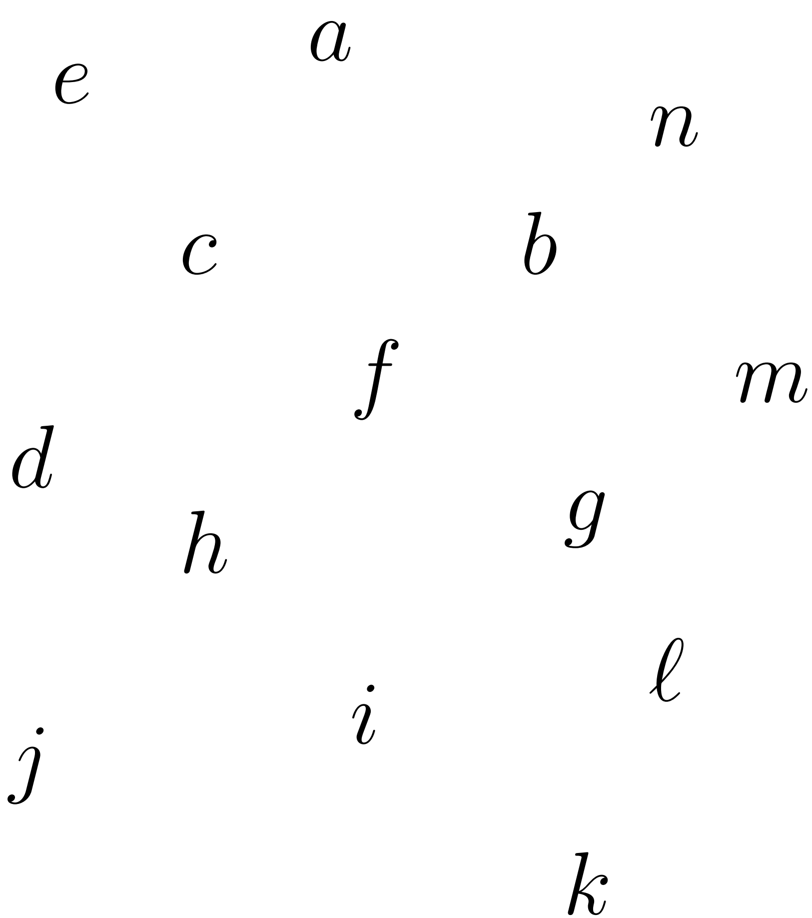
The names are arbitrary

The Operations

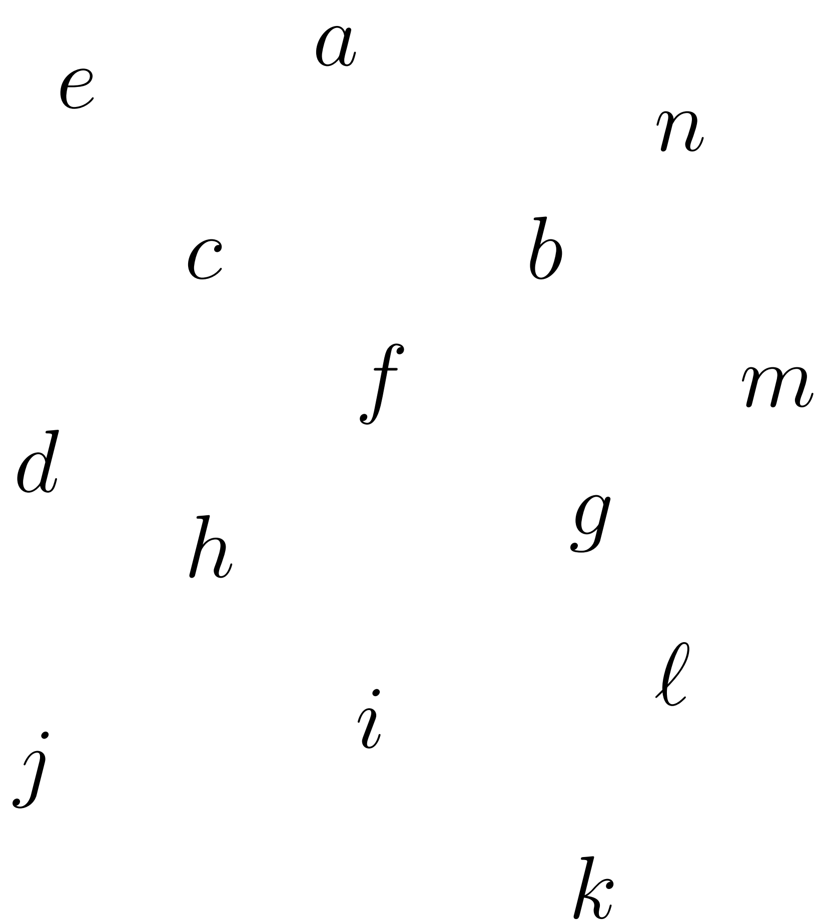
```
>>> init({a, b, c, d})
>>>
```

The Data Structure: Union by Rank

The Data Structure: Union by Rank

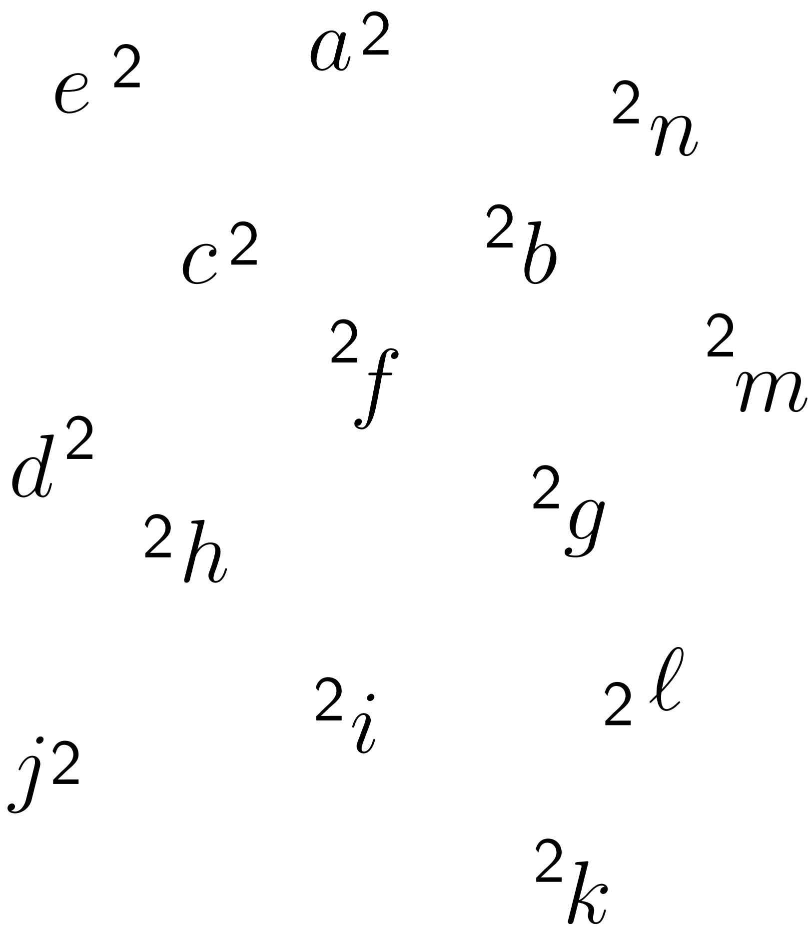


The Data Structure: Union by Rank



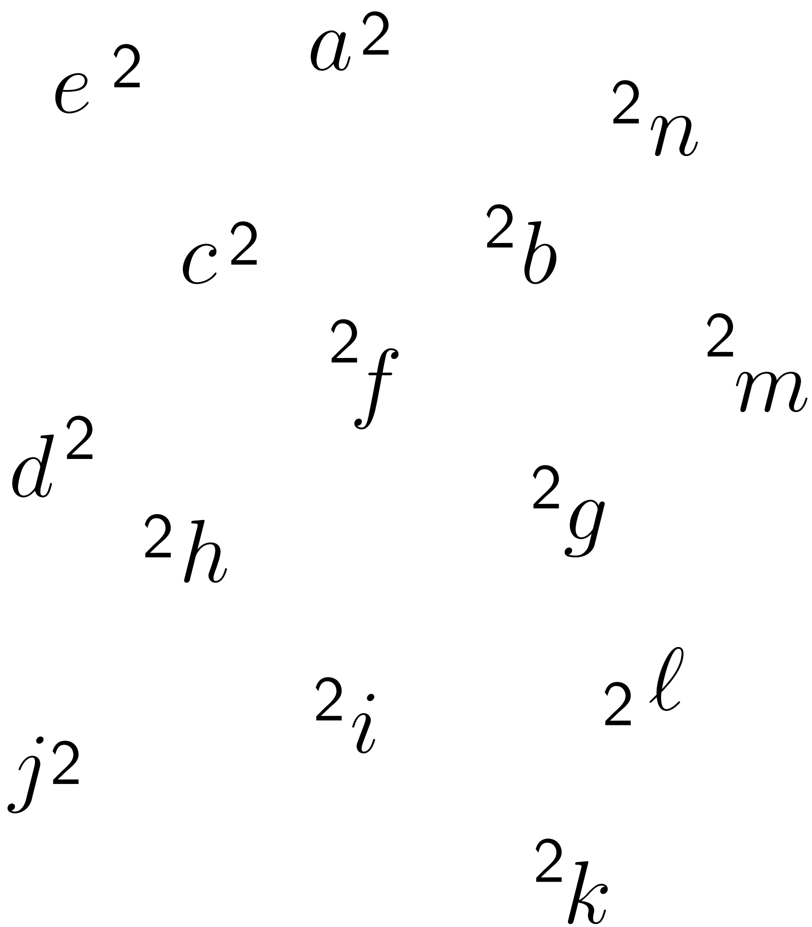
Every element has a *rank*. It is initialized to 2 at the beginning.

The Data Structure: Union by Rank



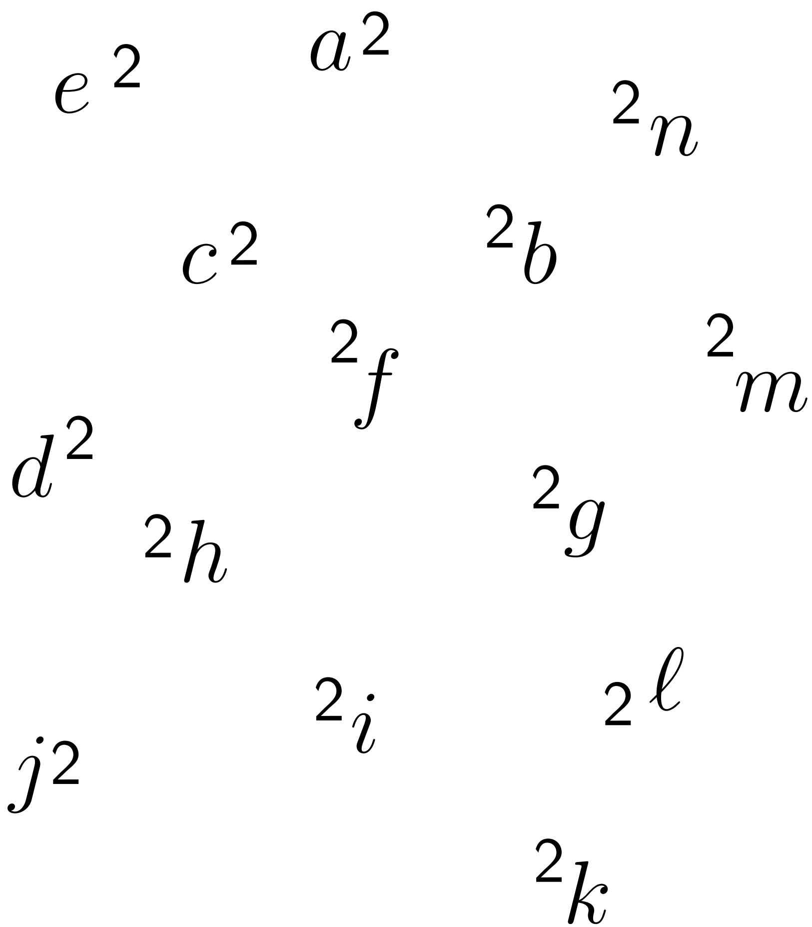
Every element has a *rank*. It is initialized to 2 at the beginning.

The Data Structure: Union by Rank

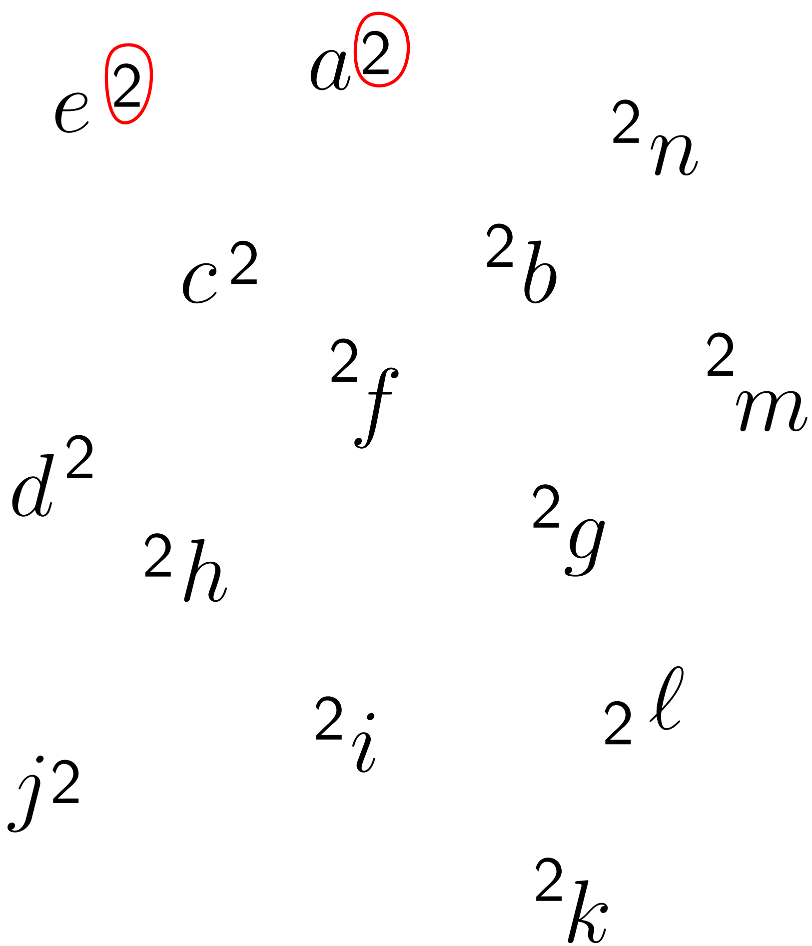


The Data Structure: Union by Rank

$>>>$ `union`(e, a)



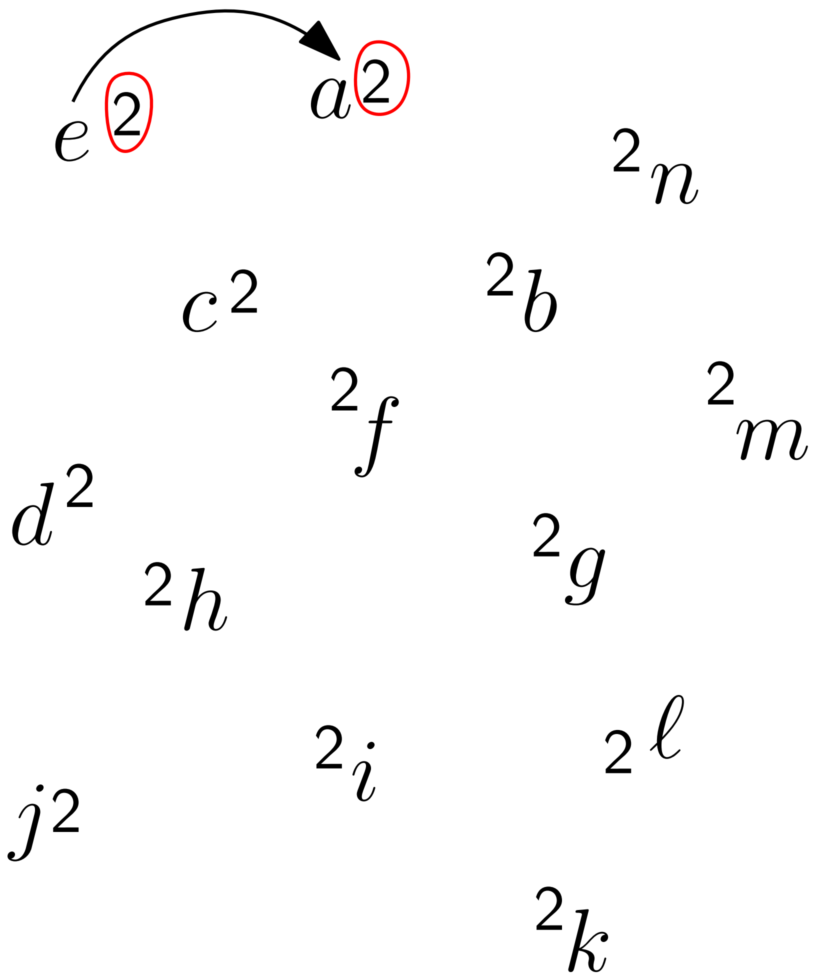
The Data Structure: Union by Rank



$>>>$ `union`(e, a)

if $\mathrm{rank(e)} = \mathrm{rank(a)}$:

The Data Structure: Union by Rank

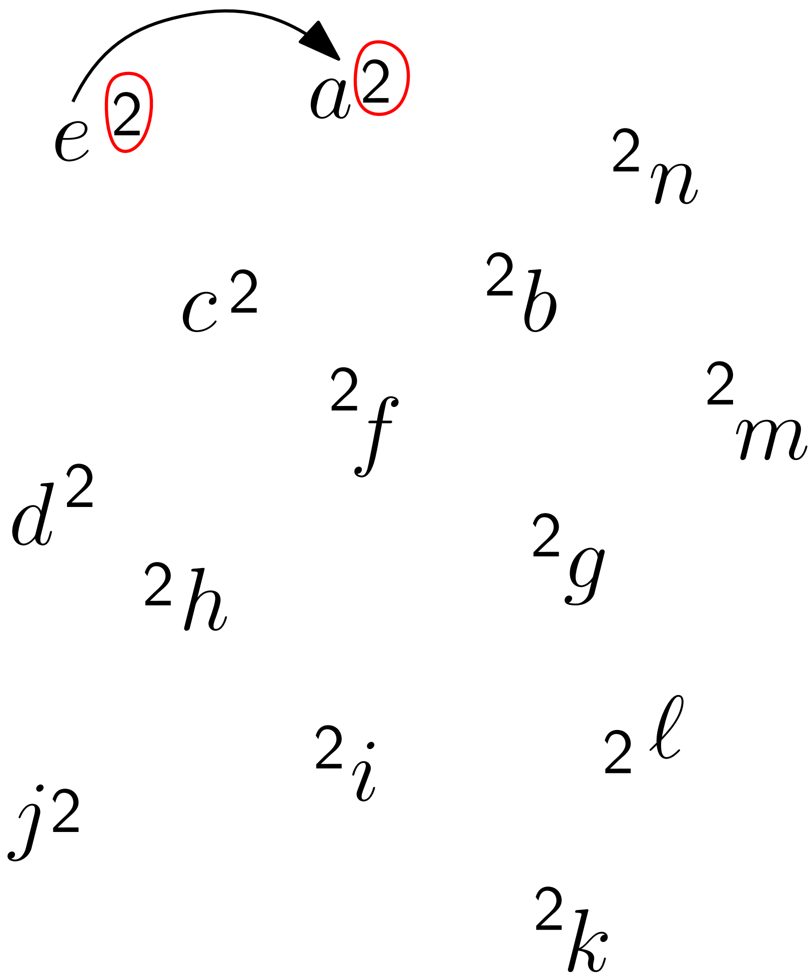


$>>>$ `union`(e, a)

if $\mathrm{rank(e)} = \mathrm{rank(a)}$:

edge from e to a

The Data Structure: Union by Rank



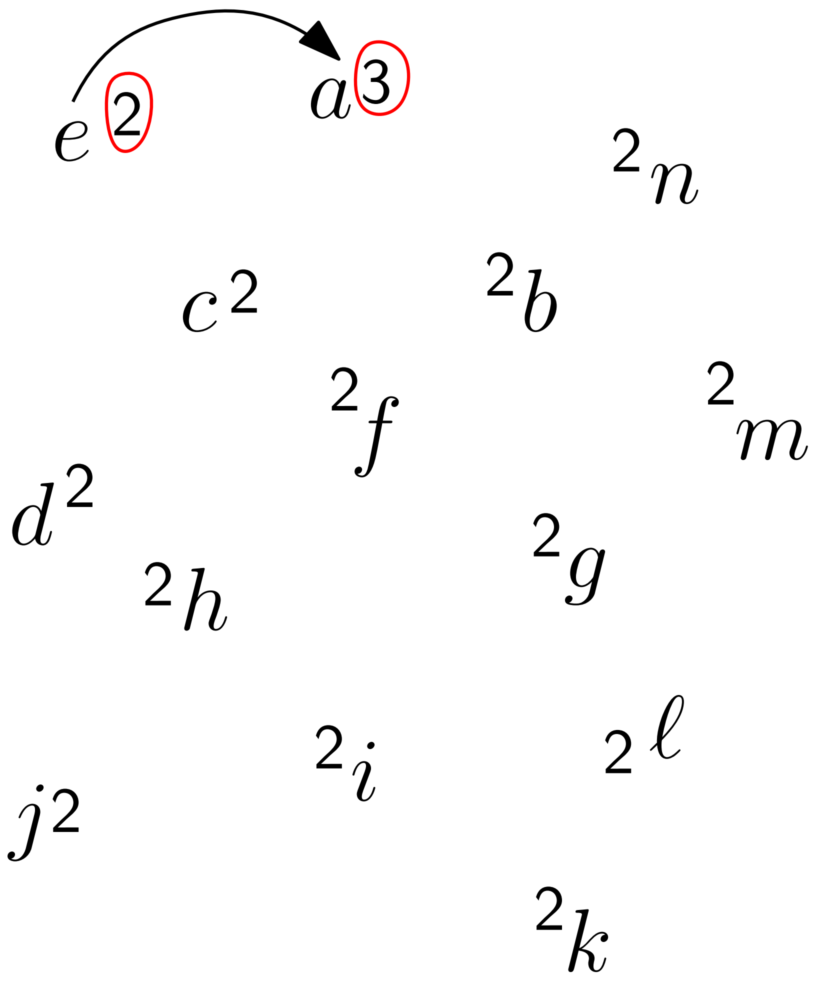
$>>>$ `union`(e, a)

if $\mathrm{rank(e)} = \mathrm{rank(a)}$:

edge from e to a

increase rank of a

The Data Structure: Union by Rank



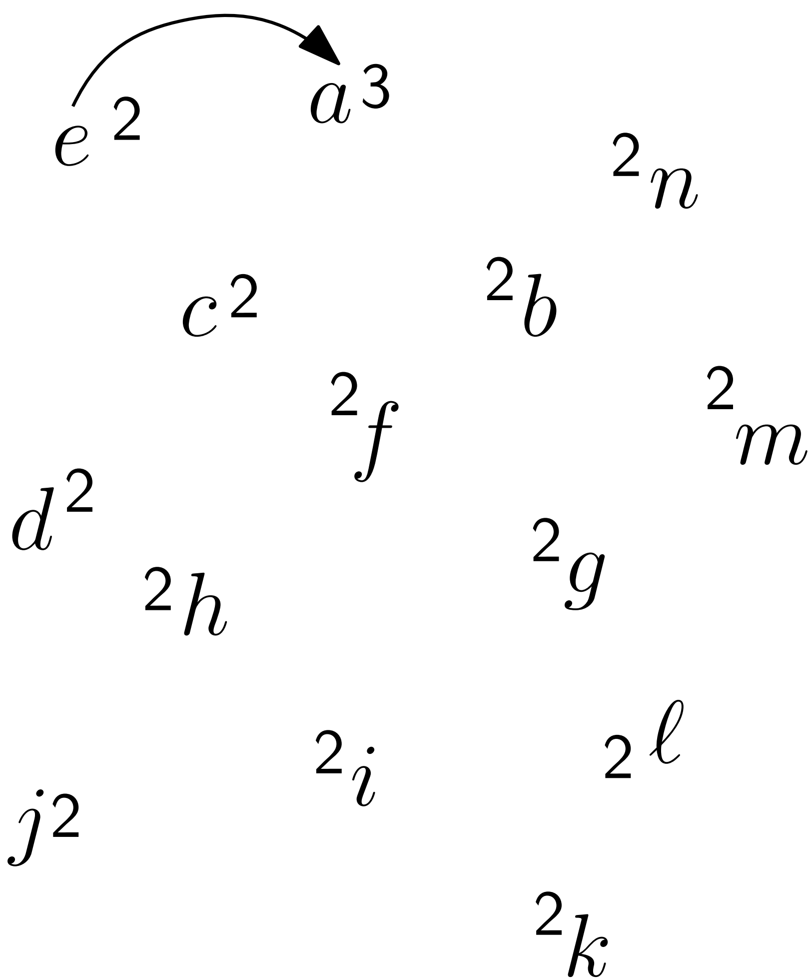
$>>>$ `union`(e, a)

if $\text{rank(e)} = \text{rank(a)}$:

edge from e to a

increase rank of a

The Data Structure: Union by Rank



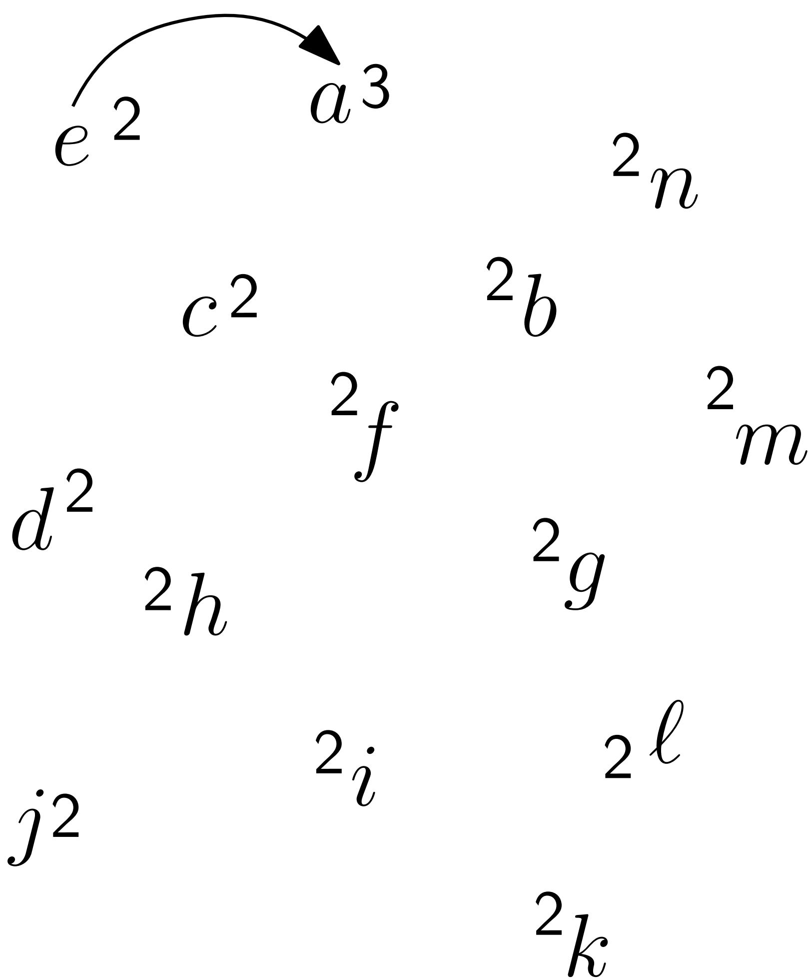
>>> `union`(e, a)

if $\mathrm{rank(e)} = \mathrm{rank(a)}$:

edge from e to a

increase rank of a

The Data Structure: Union by Rank



$>>>$ `union`(e, a)

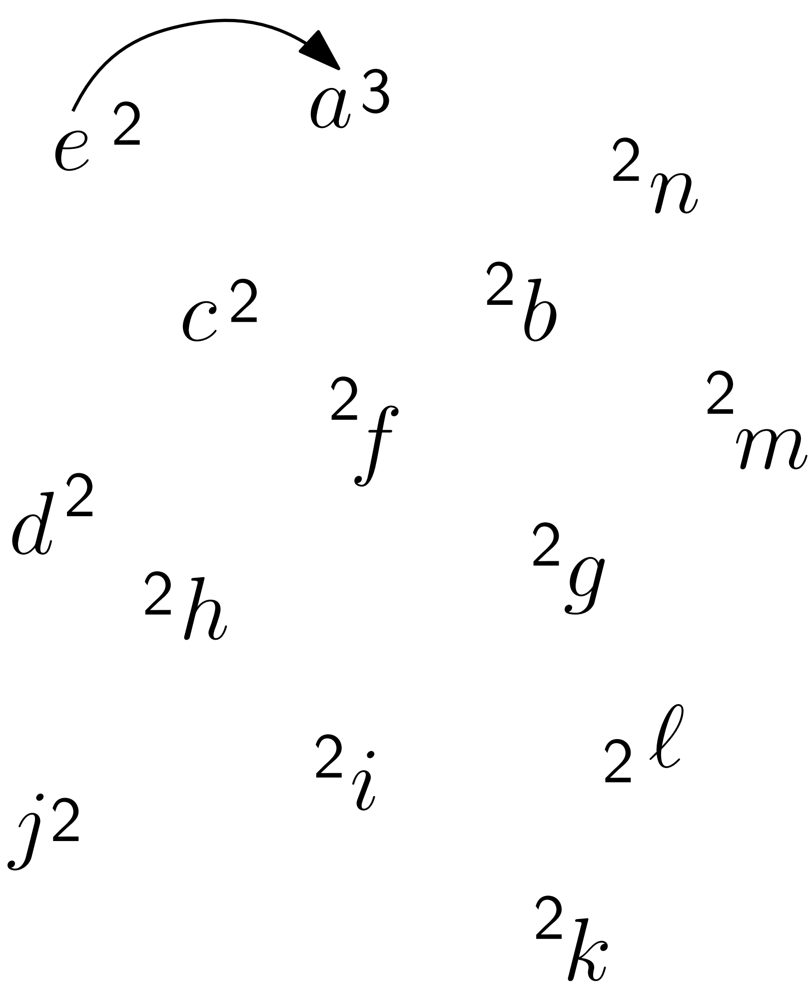
if $\mathrm{rank(e)} = \mathrm{rank(a)}$:

  edge from e to a

  increase rank of a

$>>>$ `union`(a, c)

The Data Structure: Union by Rank



$>>>$ `union`(e, a)

if $\mathrm{rank(e)} = \mathrm{rank(a)}$:

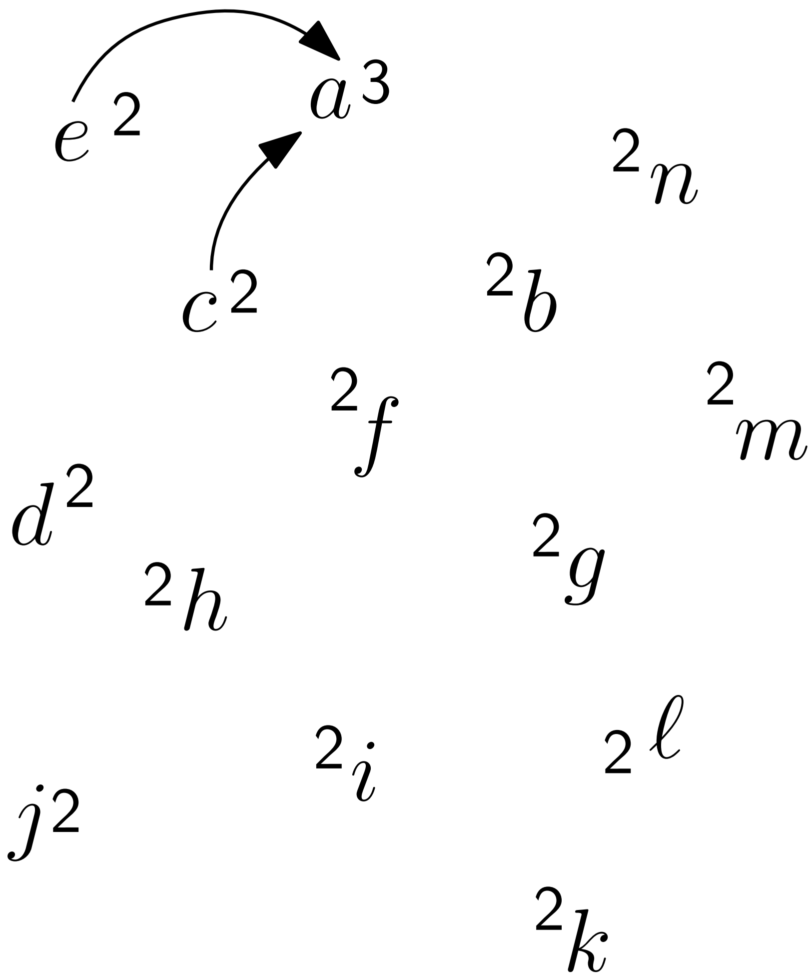
edge from e to a

increase rank of a

$>>>$ `union`(a, c)

if $\mathrm{rank(e)} \neq \mathrm{rank(a)}$:

The Data Structure: Union by Rank



>>> `union`(e, a)

if $\mathrm{rank(e)} = \mathrm{rank(a)}$:

edge from e to a

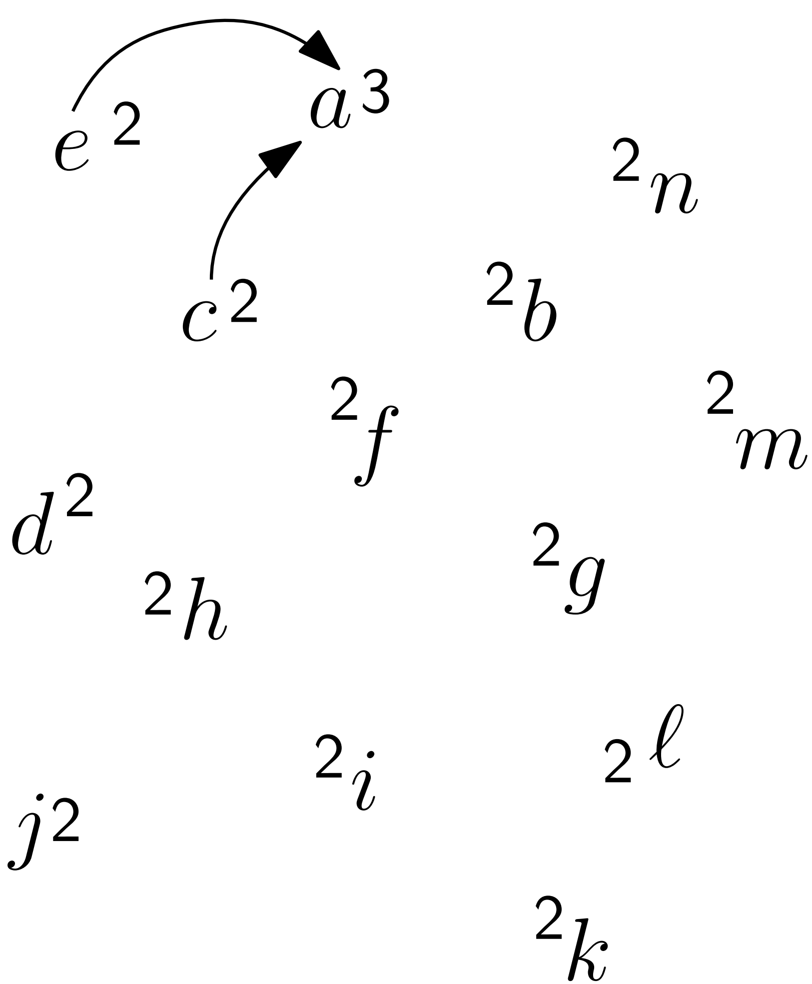
increase rank of a

>>> `union`(a, c)

if $\mathrm{rank(e)} \neq \mathrm{rank(a)}$:

smaller to larger

The Data Structure: Union by Rank



$>>>$ `union`(e, a)

if $\text{rank(e)} = \text{rank(a)}$:

edge from e to a

increase rank of a

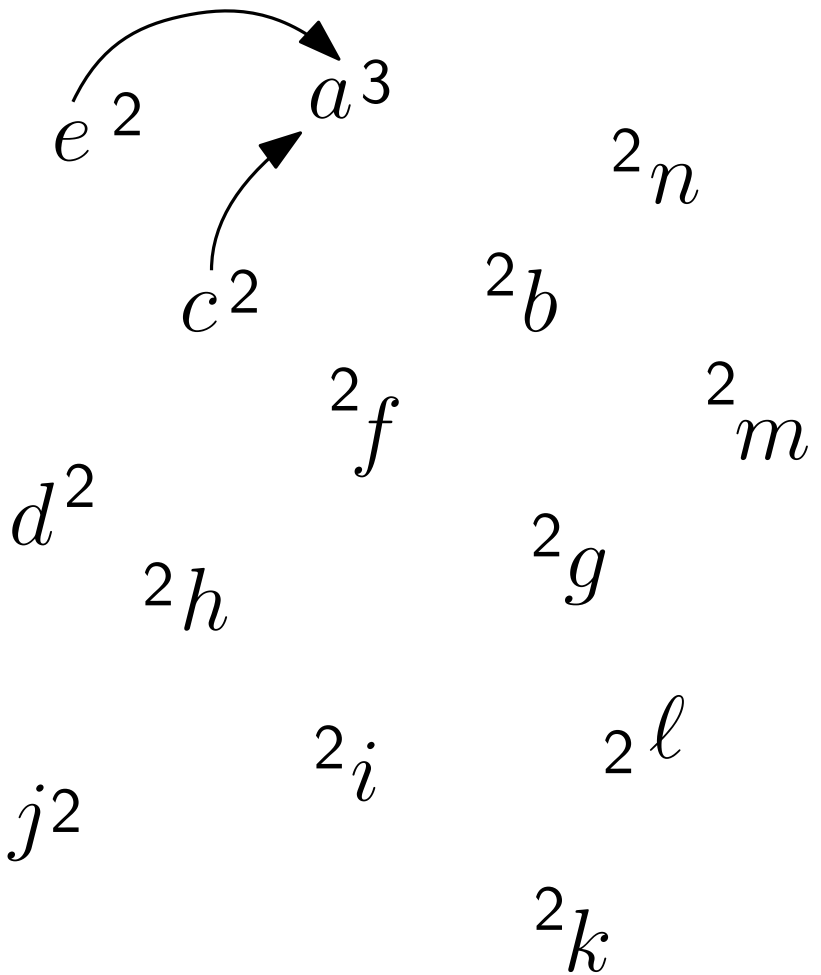
$>>>$ `union`(a, c)

if $\text{rank(e)} \neq \text{rank(a)}$:

smaller to larger

don't change rank

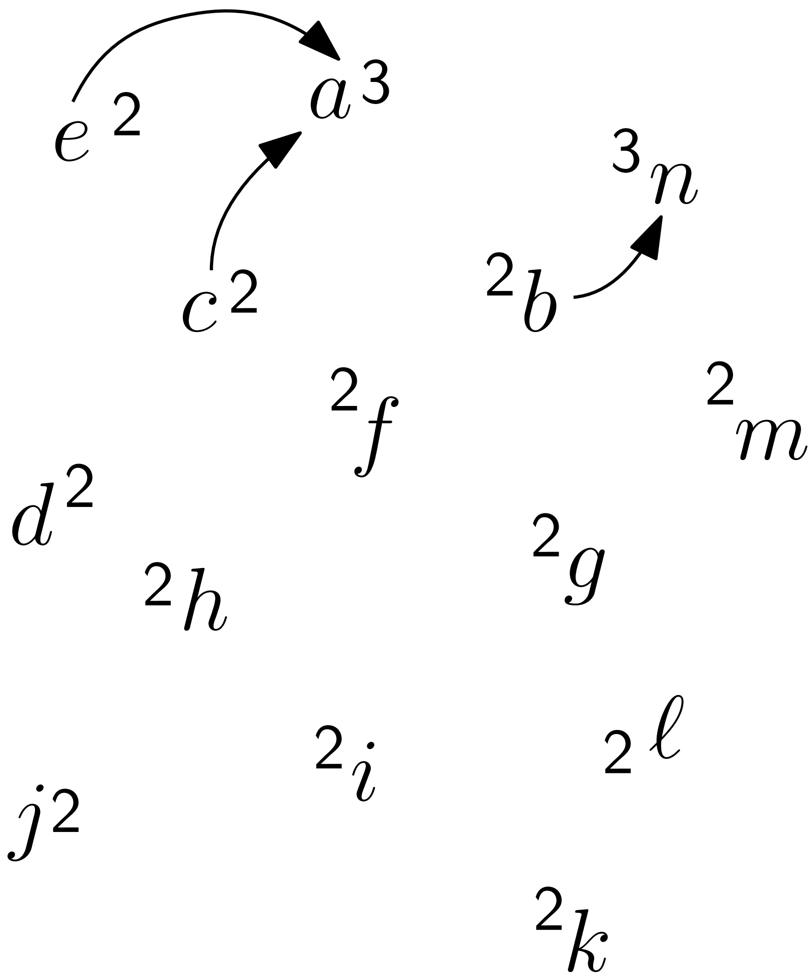
The Data Structure: Union by Rank



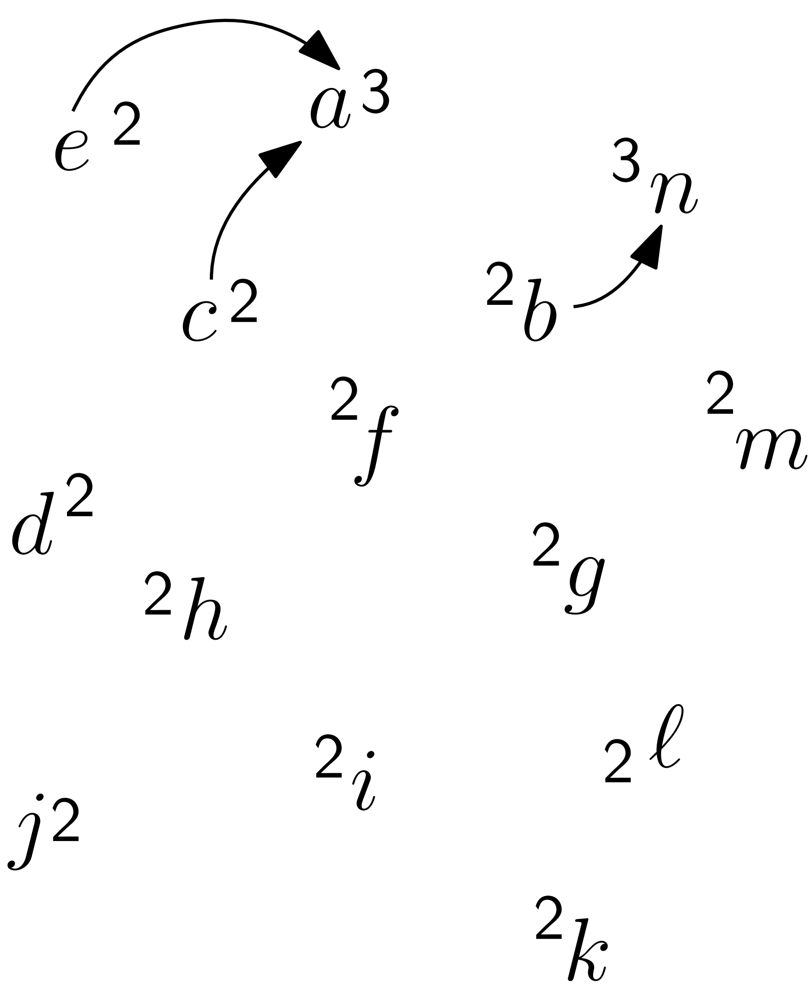
$>>>$ `union`(b, n)

The Data Structure: Union by Rank

$>>>$ `union`(b, n)



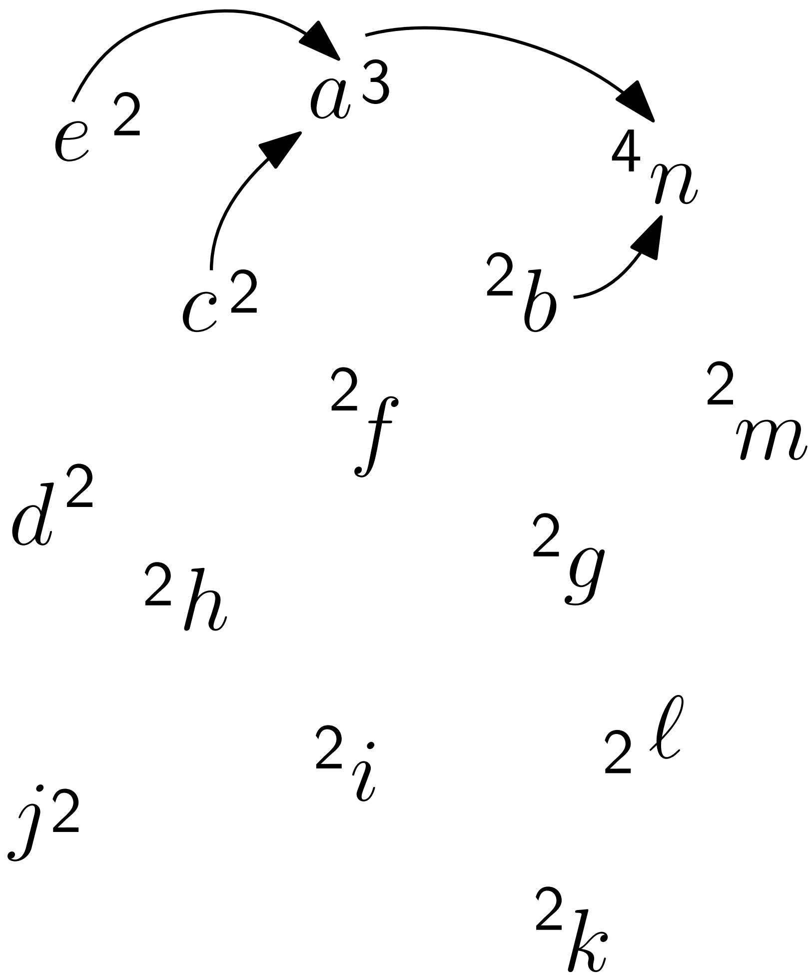
The Data Structure: Union by Rank



$>>>$ `union`(b, n)

$>>>$ `union`(a, n)

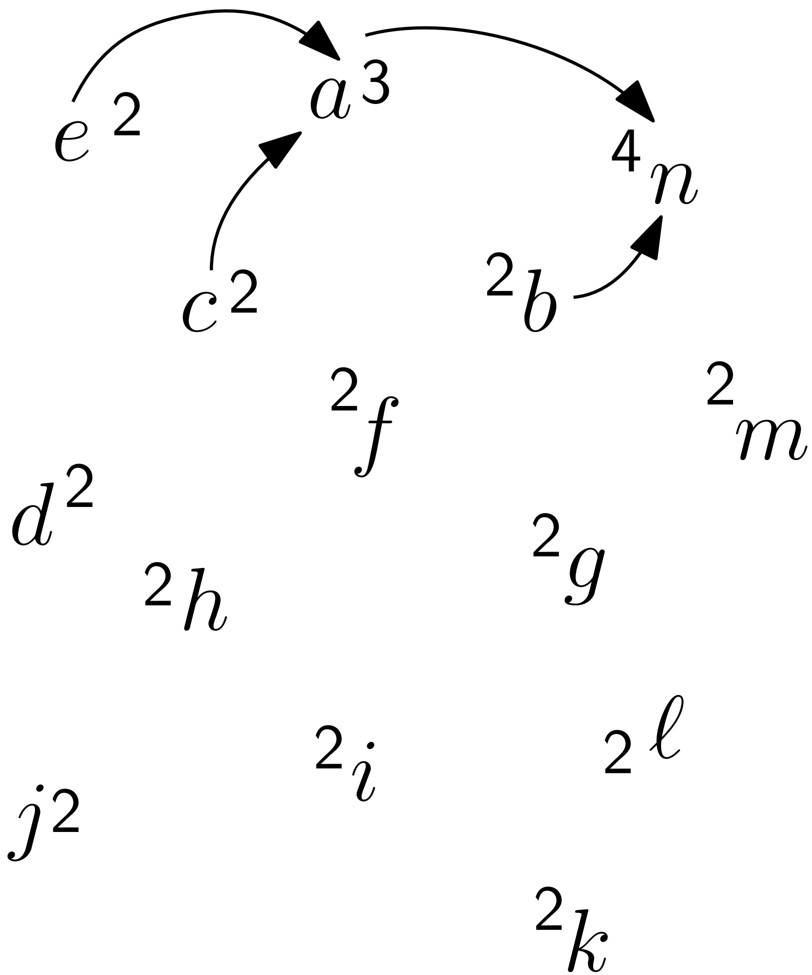
The Data Structure: Union by Rank



$>>>$ `union`(b, n)

$>>>$ `union`(a, n)

The Data Structure: Union by Rank

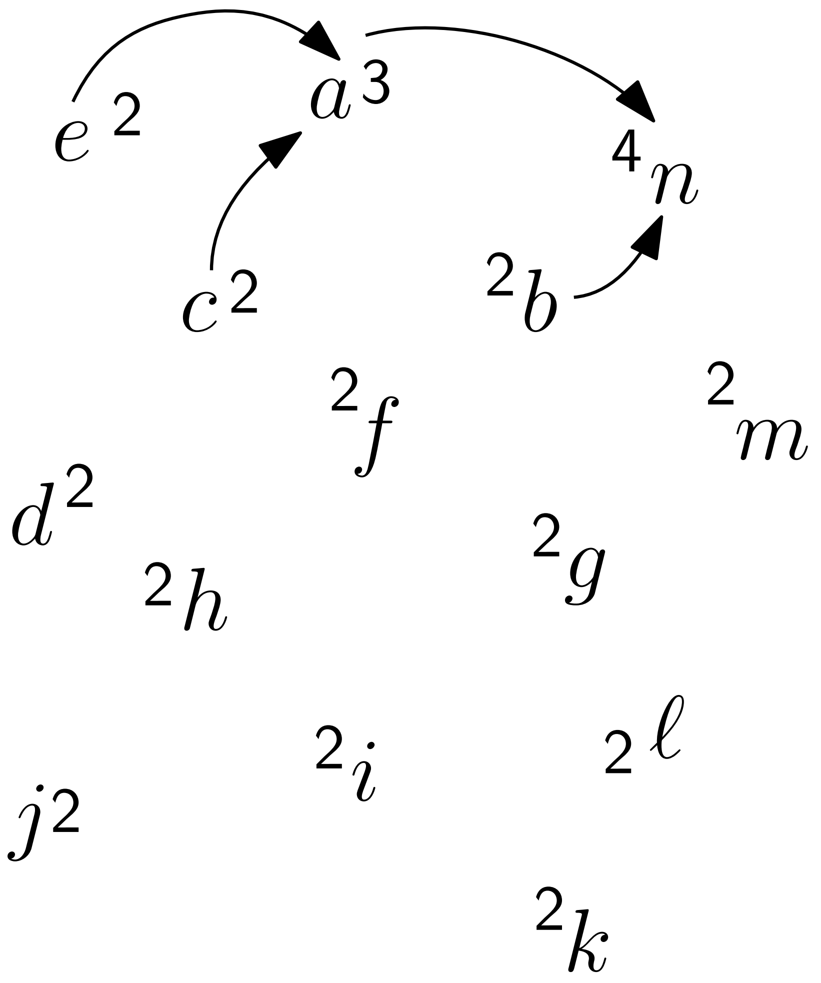


$>>>$ `union`(b, n)

$>>>$ `union`(a, n)

cost: $O(1)$

The Data Structure: Union by Rank



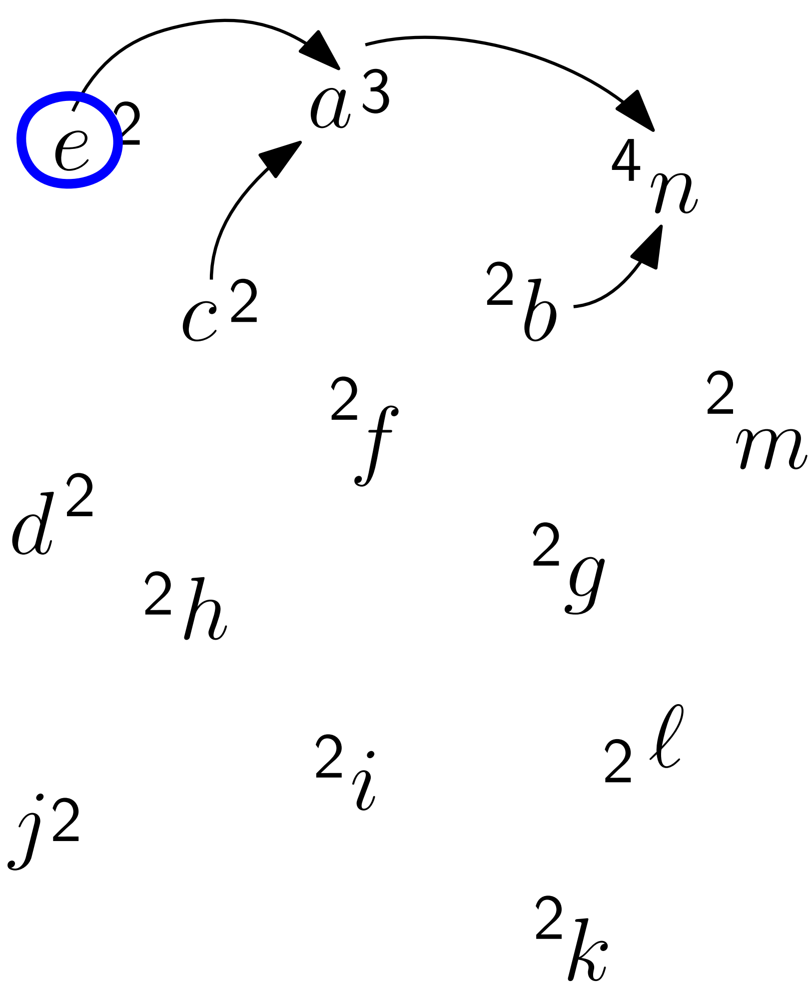
$>>>$ `union`(b, n)

$>>>$ `union`(a, n)

cost: $O(1)$

$>>>$ `find`(e)

The Data Structure: Union by Rank



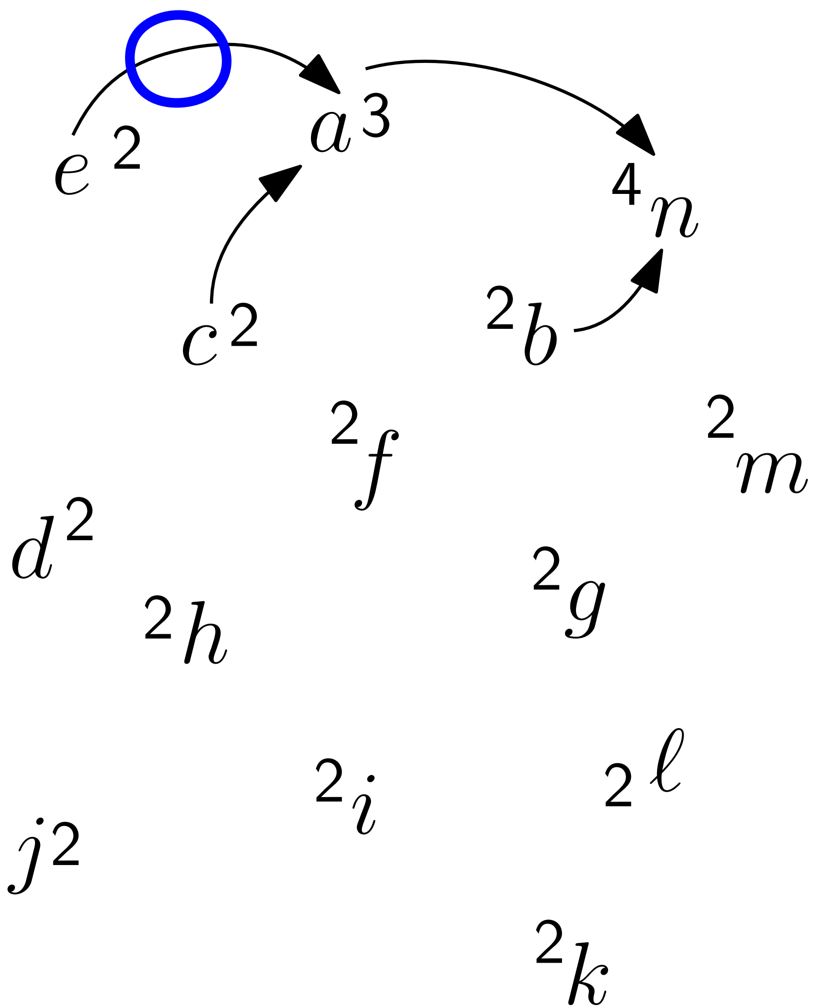
$>>>$ `union`(b, n)

$>>>$ `union`(a, n)

cost: $O(1)$

$>>>$ `find`(e)

The Data Structure: Union by Rank



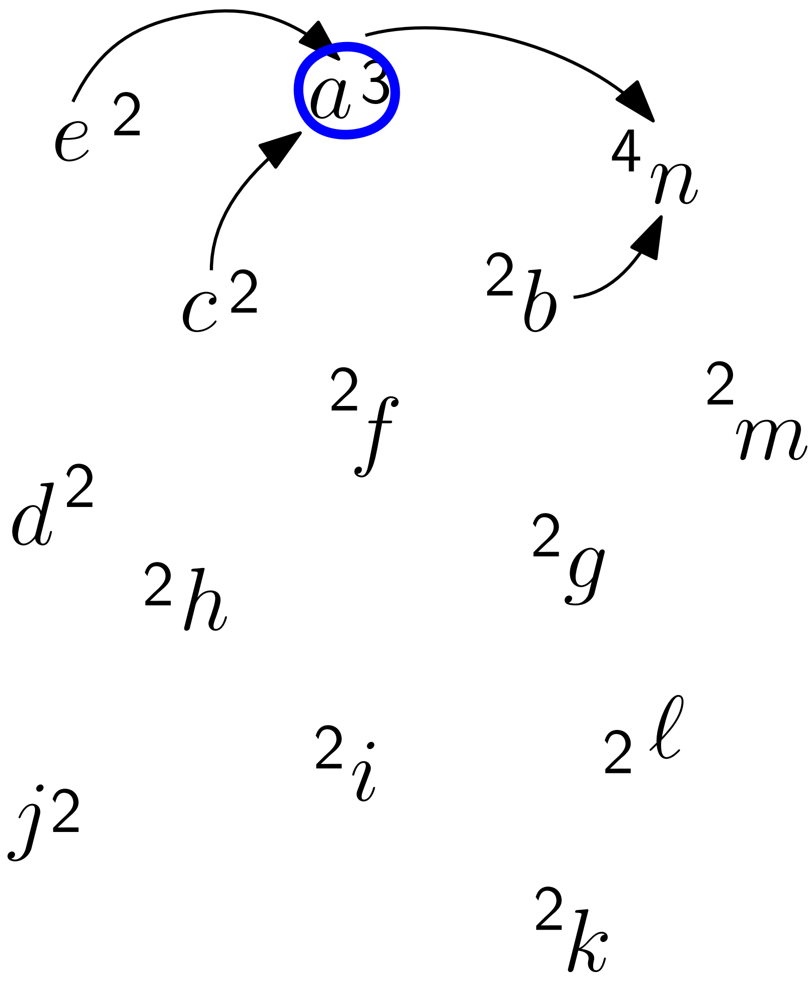
$>>>$ `union`(b, n)

$>>>$ `union`(a, n)

cost: $O(1)$

$>>>$ `find`(e)

The Data Structure: Union by Rank



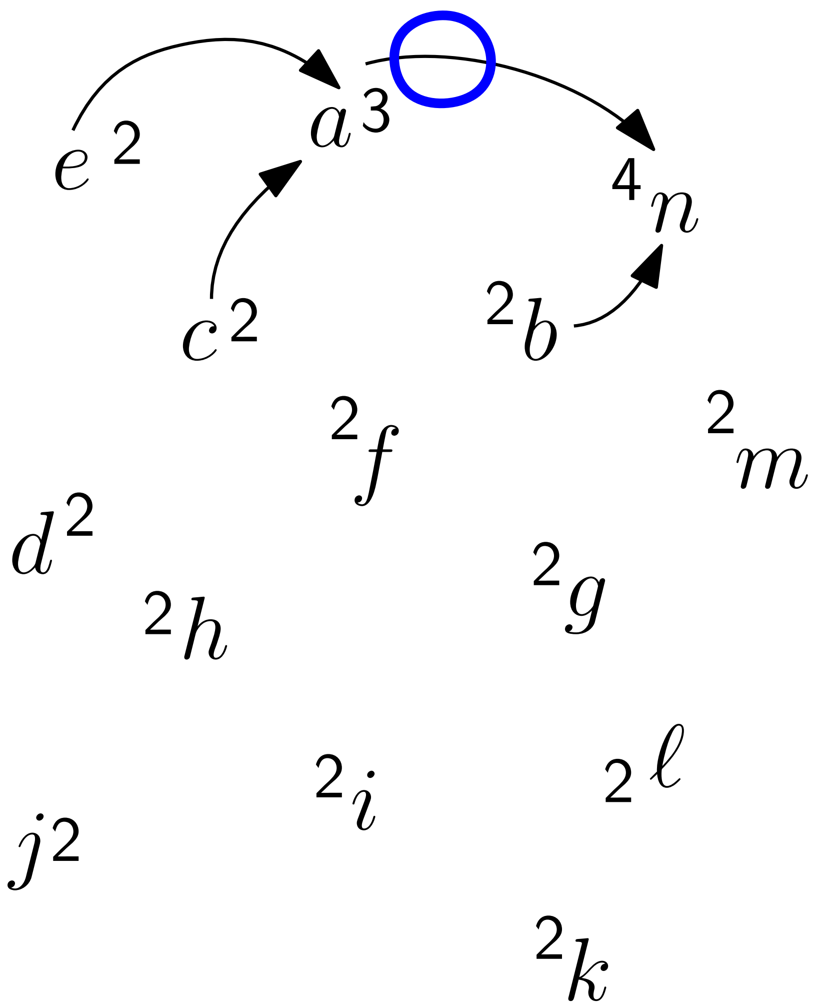
$>>>$ `union`(b, n)

$>>>$ `union`(a, n)

cost: $O(1)$

$>>>$ `find`(e)

The Data Structure: Union by Rank



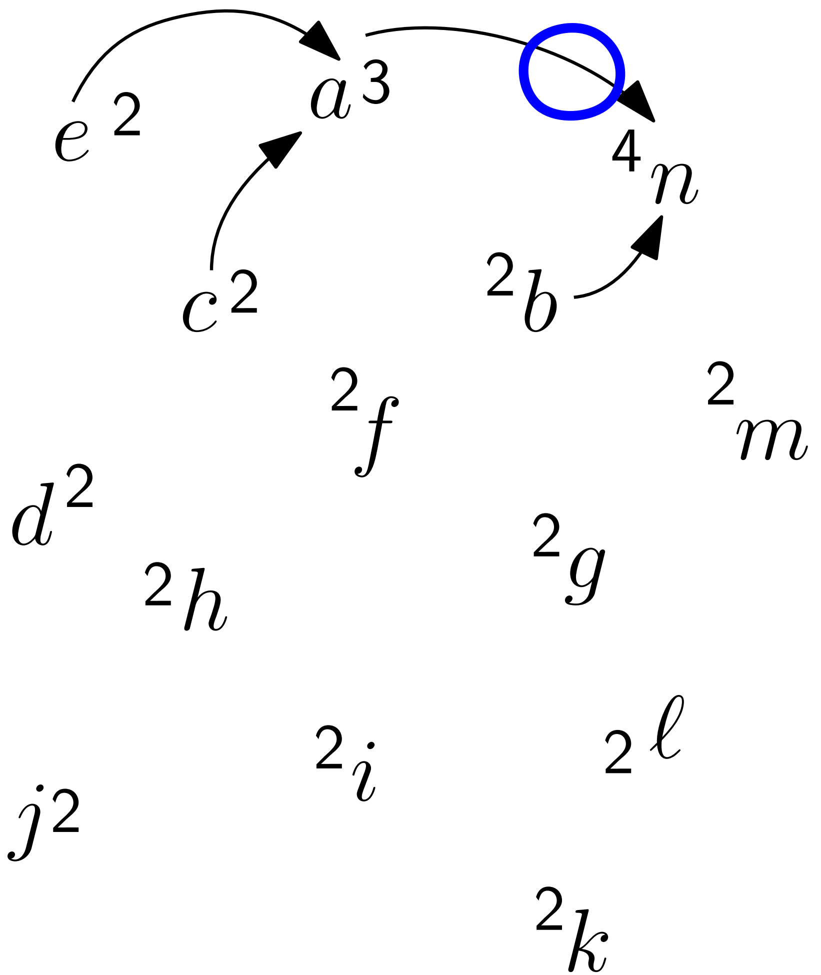
$>>>$ `union`(b, n)

$>>>$ `union`(a, n)

cost: $O(1)$

$>>>$ `find`(e)

The Data Structure: Union by Rank



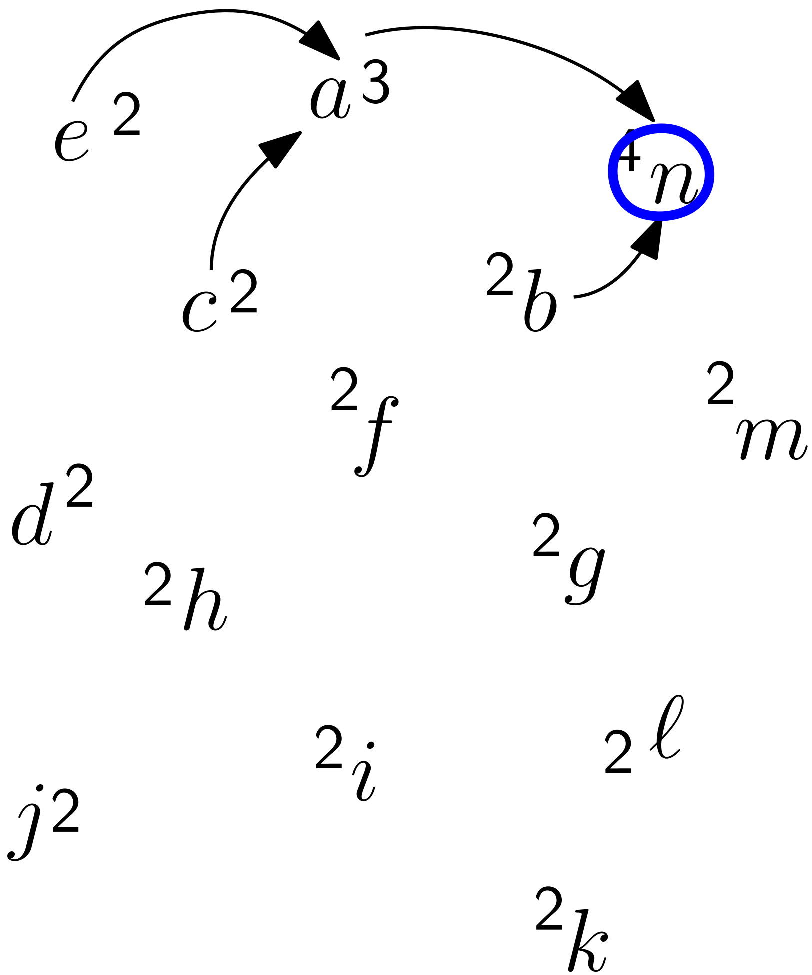
$>>>$ `union`(b, n)

$>>>$ `union`(a, n)

cost: $O(1)$

$>>>$ `find`(e)

The Data Structure: Union by Rank



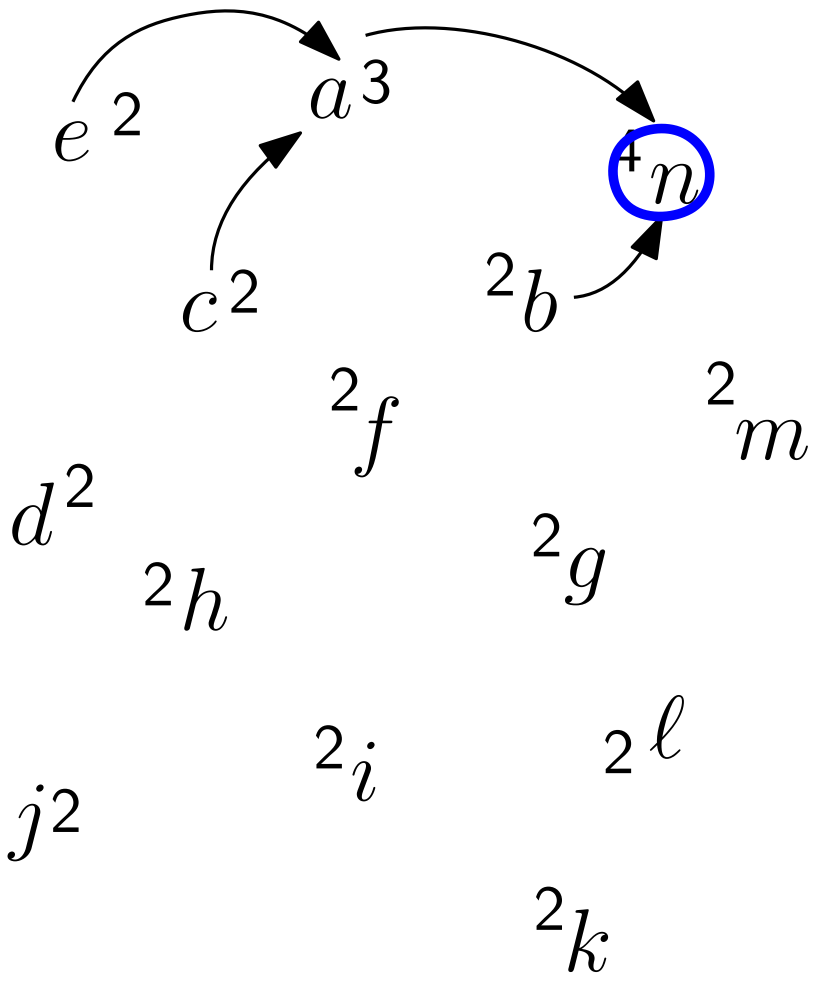
$>>>$ `union`(b, n)

$>>>$ `union`(a, n)

cost: $O(1)$

$>>>$ `find`(e)

The Data Structure: Union by Rank



$>>>$ `union`(b, n)

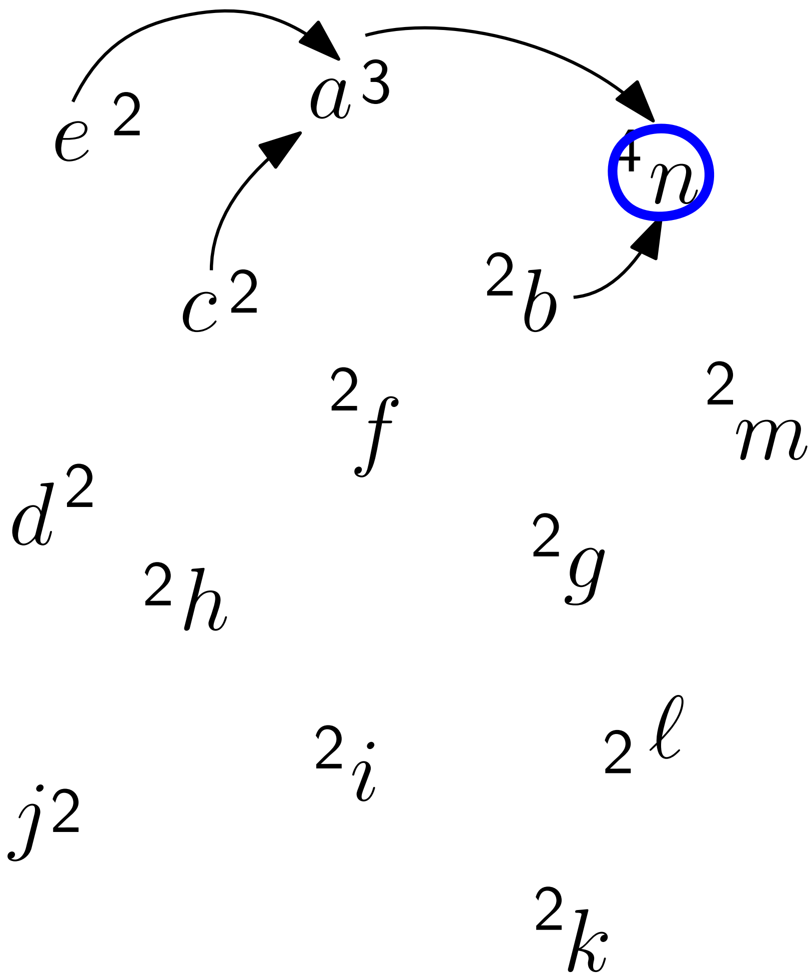
$>>>$ `union`(a, n)

cost: $O(1)$

$>>>$ `find`(e)

n

The Data Structure: Union by Rank



$>>>$ `union`(b, n)

$>>>$ `union`(a, n)

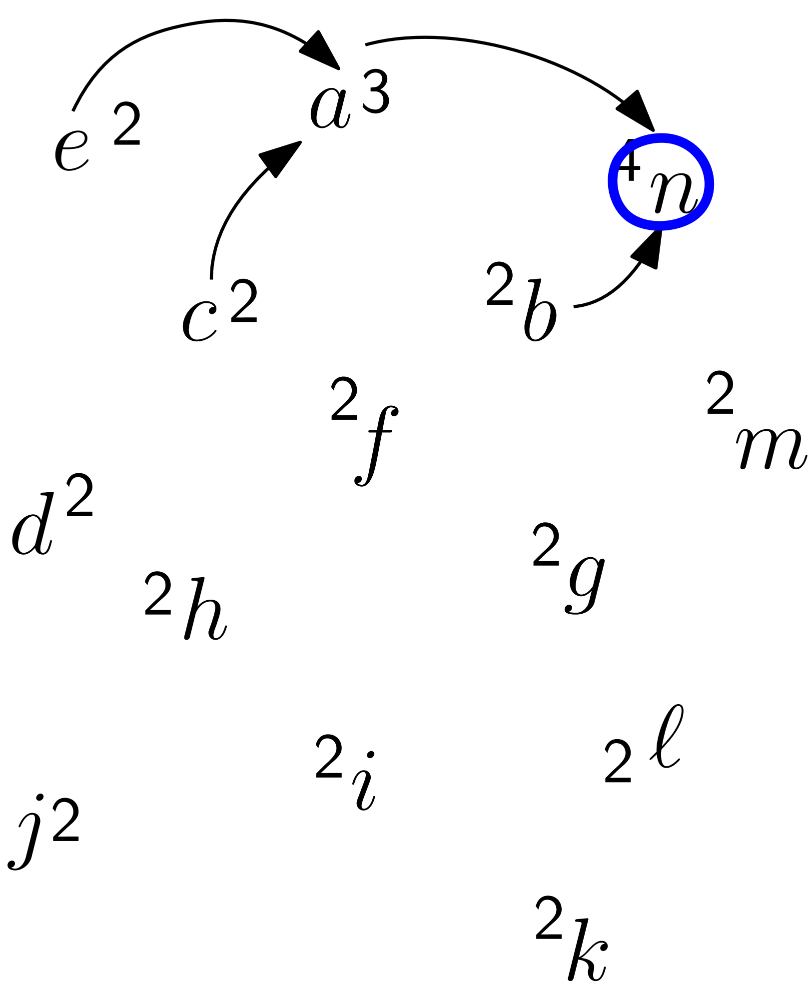
cost: $O(1)$

$>>>$ `find`(e)

n

cost: $O($ length of path$)$

The Data Structure: Union by Rank



$>>>$ `union`(b, n)

$>>>$ `union`(a, n)

cost: $O(1)$

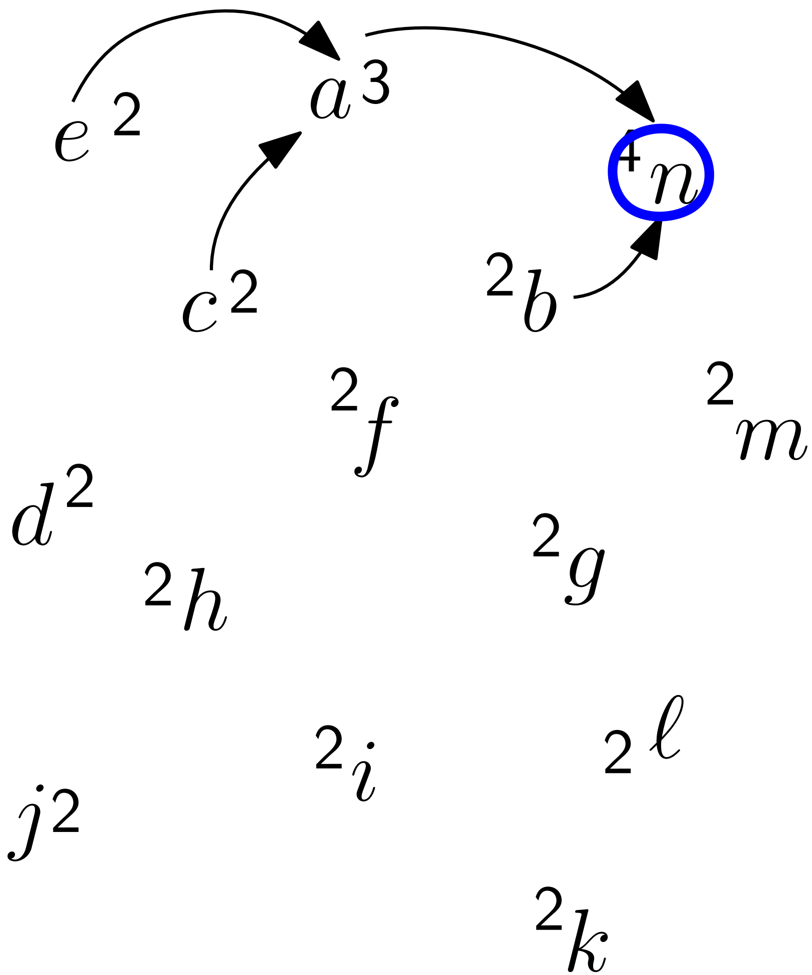
$>>>$ `find`(e)

n

cost: $O($ length of path$)$

$\leq O($ height of tree$)$

The Data Structure: Union by Rank



>>> $\texttt{union}(b, n)$

>>> $\texttt{union}(a, n)$

cost: $O(1)$

>>> $\texttt{find}(e)$

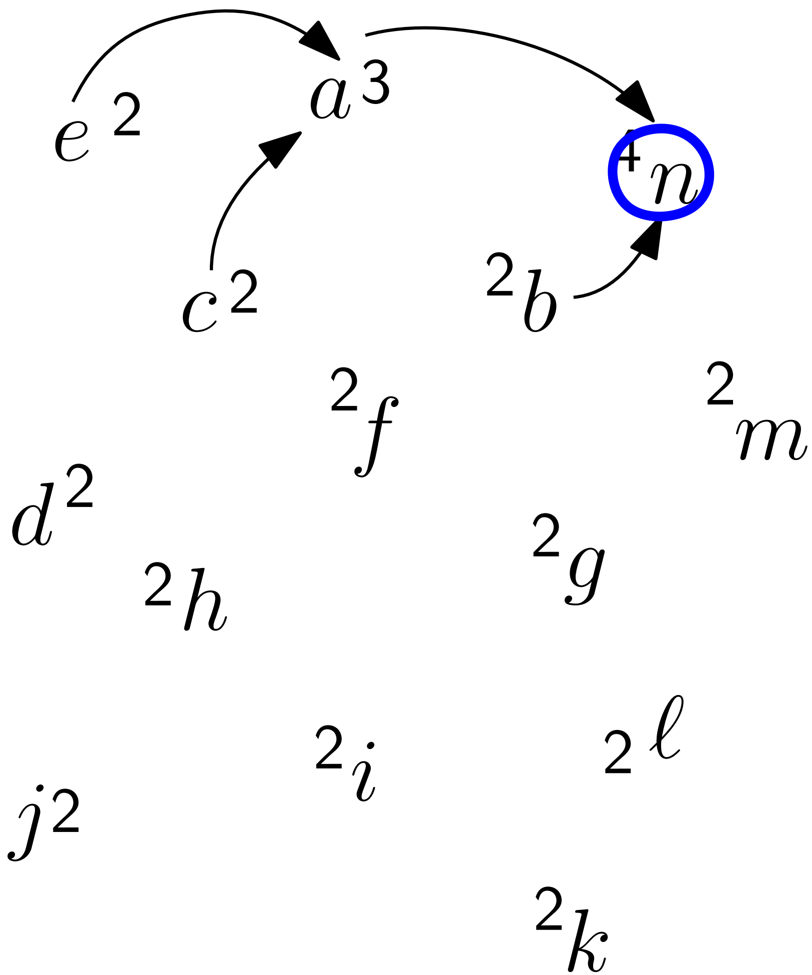
n

cost: $O($ length of path$)$

$\leq O($ height of tree$)$

$\leq O($ rank(n)$)$

The Data Structure: Union by Rank



$>>>$ `union`(b, n)

$>>>$ `union`(a, n)

cost: $O(1)$

$>>>$ `find`(e)

n

cost: $O($ length of path$)$

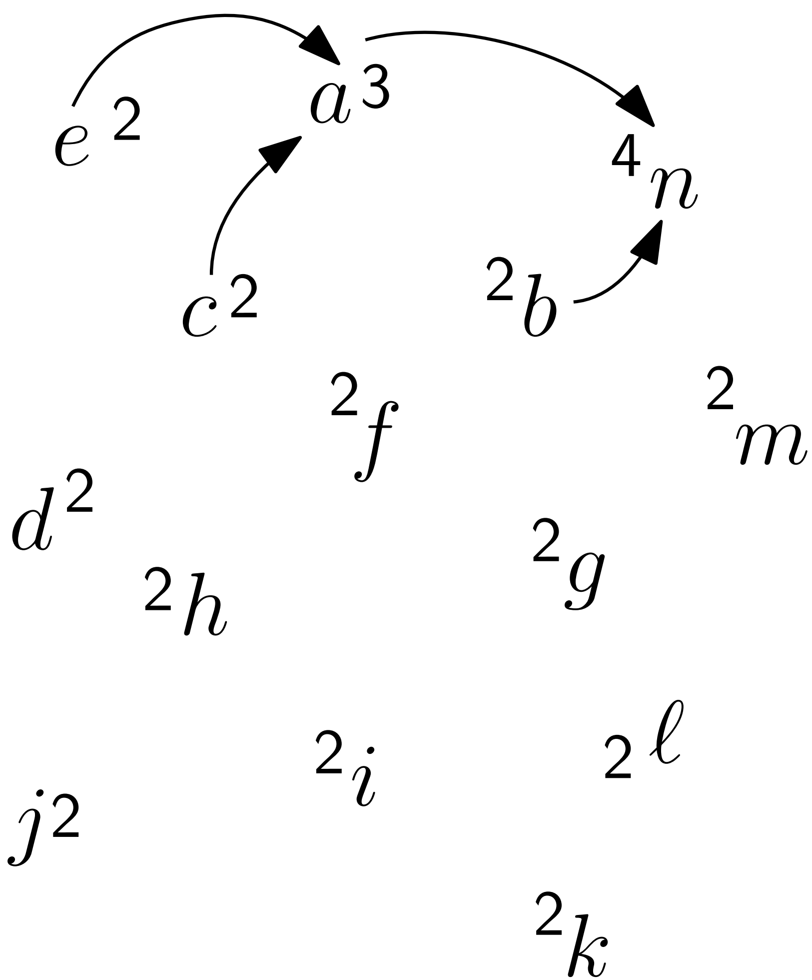
$\leq O($ height of tree$)$

$\leq O($ rank(n)$)$

because

Lemma: The height of the subtree rooted at x is $\text{rank}(x) - 2$.

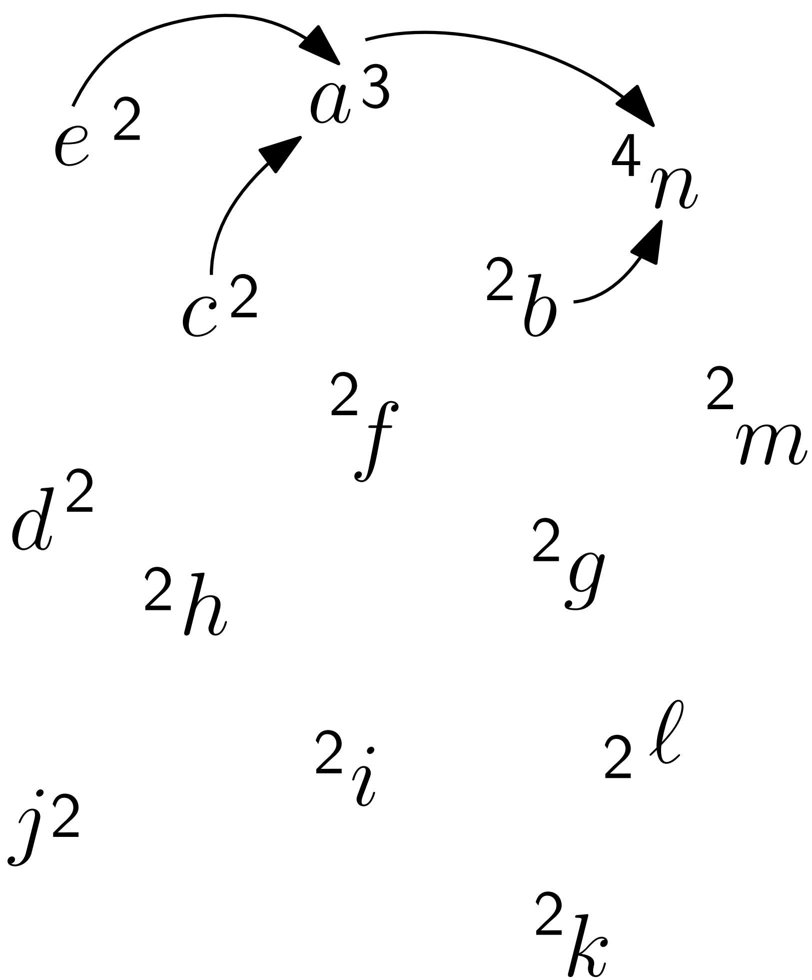
The Data Structure: Union by Rank



$>>>$ `union`(b, n)
$>>>$ `union`(a, n)

$>>>$ `find`(e)
n

The Data Structure: Union by Rank



$>>>$ `union`(b, n)

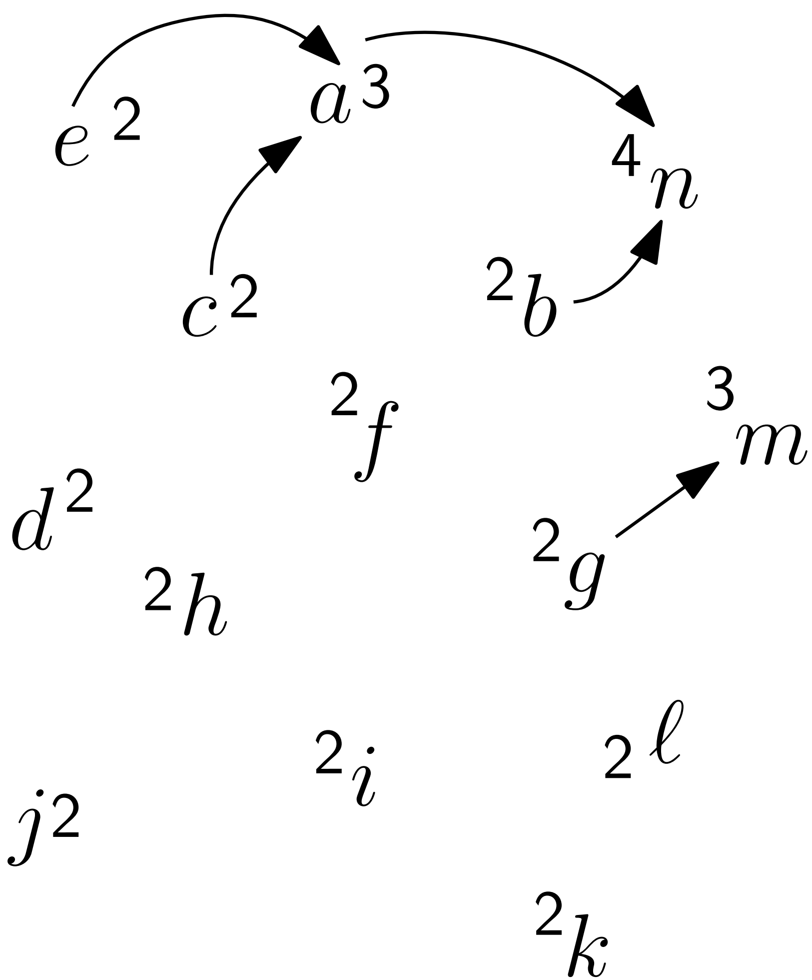
$>>>$ `union`(a, n)

$>>>$ `find`(e)

n

$>>>$ `union`(g, m)

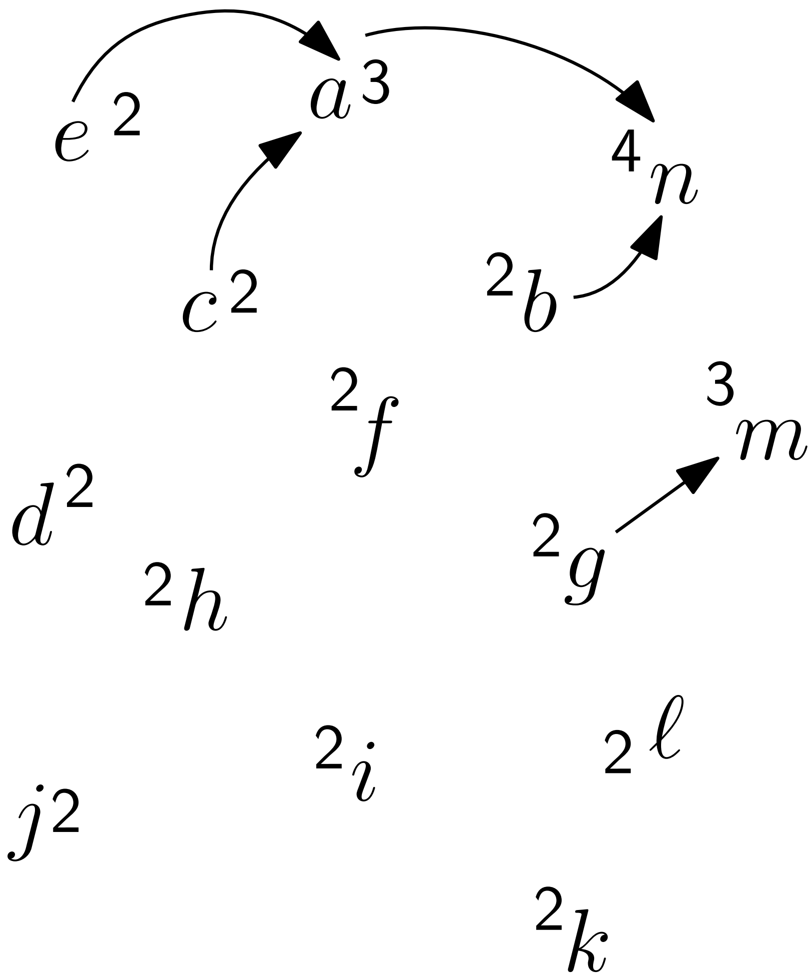
The Data Structure: Union by Rank



>>> $\texttt{union}(b, n)$
>>> $\texttt{union}(a, n)$

>>> $\texttt{find}(e)$
n
>>> $\texttt{union}(g, m)$

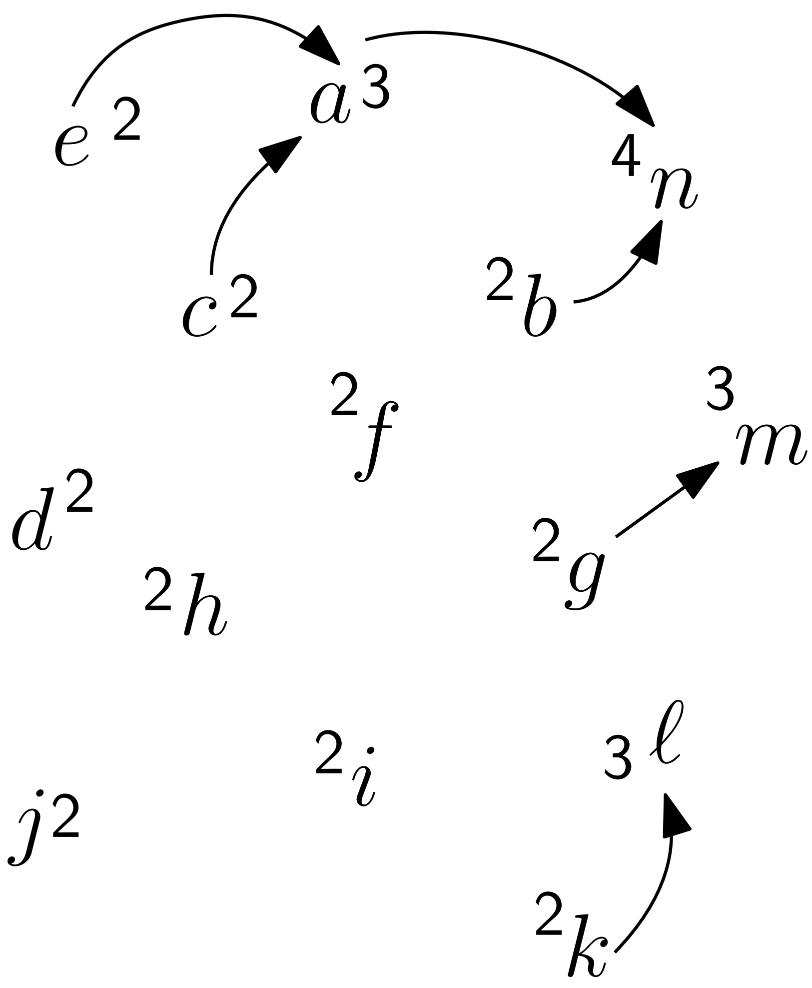
The Data Structure: Union by Rank



$>>>$ `union`(b, n)
$>>>$ `union`(a, n)

$>>>$ `find`(e)
n
$>>>$ `union`(g, m)
$>>>$ `union`(k, ℓ)

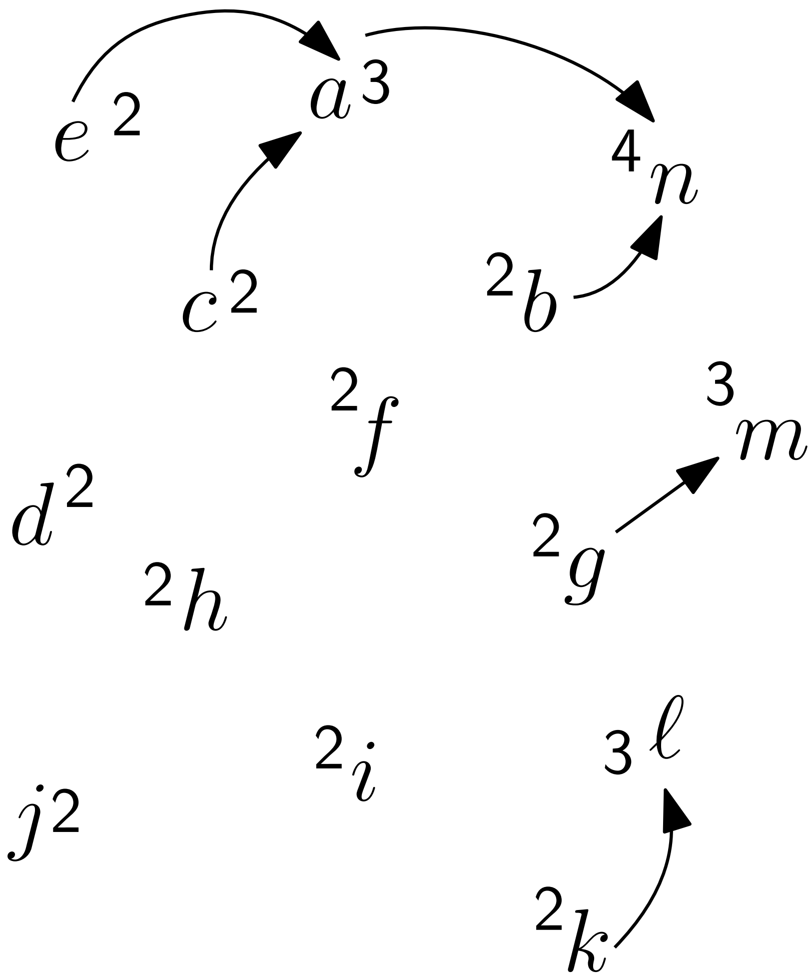
The Data Structure: Union by Rank



$>>>$ `union`(b, n)
$>>>$ `union`(a, n)

$>>>$ `find`(e)
n
$>>>$ `union`(g, m)
$>>>$ `union`(k, ℓ)

The Data Structure: Union by Rank



$>>>$ `union`(b, n)

$>>>$ `union`(a, n)

$>>>$ `find`(e)

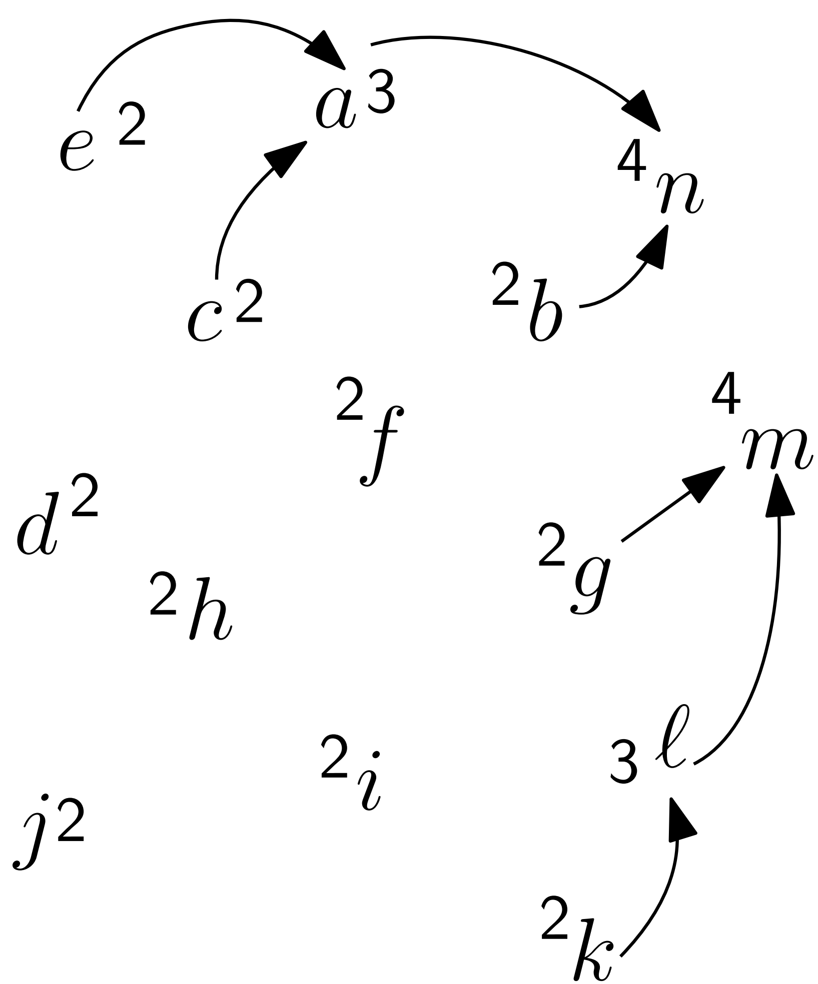
n

$>>>$ `union`(g, m)

$>>>$ `union`(k, ℓ)

$>>>$ `union`(ℓ, m)

The Data Structure: Union by Rank



$>>>$ `union`(b, n)

$>>>$ `union`(a, n)

$>>>$ `find`(e)

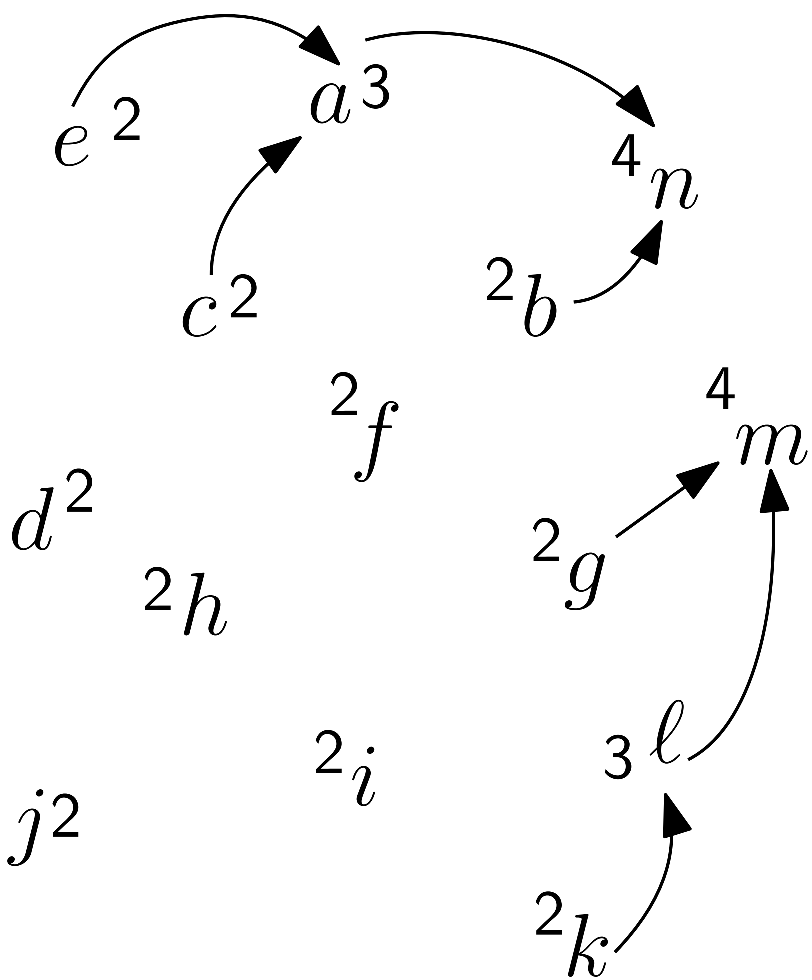
n

$>>>$ `union`(g, m)

$>>>$ `union`(k, ℓ)

$>>>$ `union`(ℓ, m)

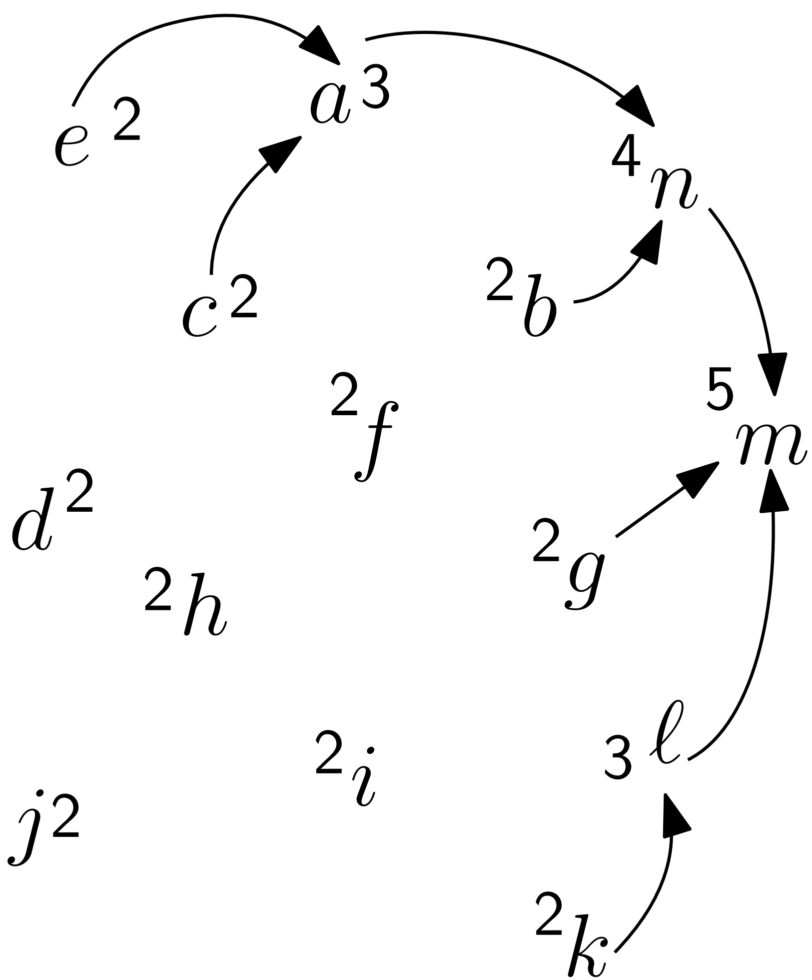
The Data Structure: Union by Rank



$>>>$ `union`(b, n)
$>>>$ `union`(a, n)

$>>>$ `find`(e)
n
$>>>$ `union`(g, m)
$>>>$ `union`(k, ℓ)
$>>>$ `union`(ℓ, m)
$>>>$ `union`(n, m)

The Data Structure: Union by Rank



$>>>$ `union`(b, n)

$>>>$ `union`(a, n)

$>>>$ `find`(e)

n

$>>>$ `union`(g, m)

$>>>$ `union`(k, ℓ)

$>>>$ `union`(ℓ, m)

$>>>$ `union`(n, m)

Lemma. The height of the subtree rooted at x is $\text{rank}(x) - 2$.

Lemma. The height of the subtree rooted at x is $\mathrm{rank}(x) - 2$.

Lemma. The number of nodes in x's subtree is at least $2^{\mathrm{rank(x)}-2}$.

Lemma. The height of the subtree rooted at x is $\mathrm{rank}(x) - 2$.

Lemma. The number of elements with rank r is at most $\frac{n}{2^{r-2}} = \frac{4n}{2^r}$.

Lemma. The number of nodes in x's subtree is at least $2^{\mathrm{rank(x)}-2}$.

Lemma. The height of the subtree rooted at x is $\mathrm{rank}(x) - 2$.

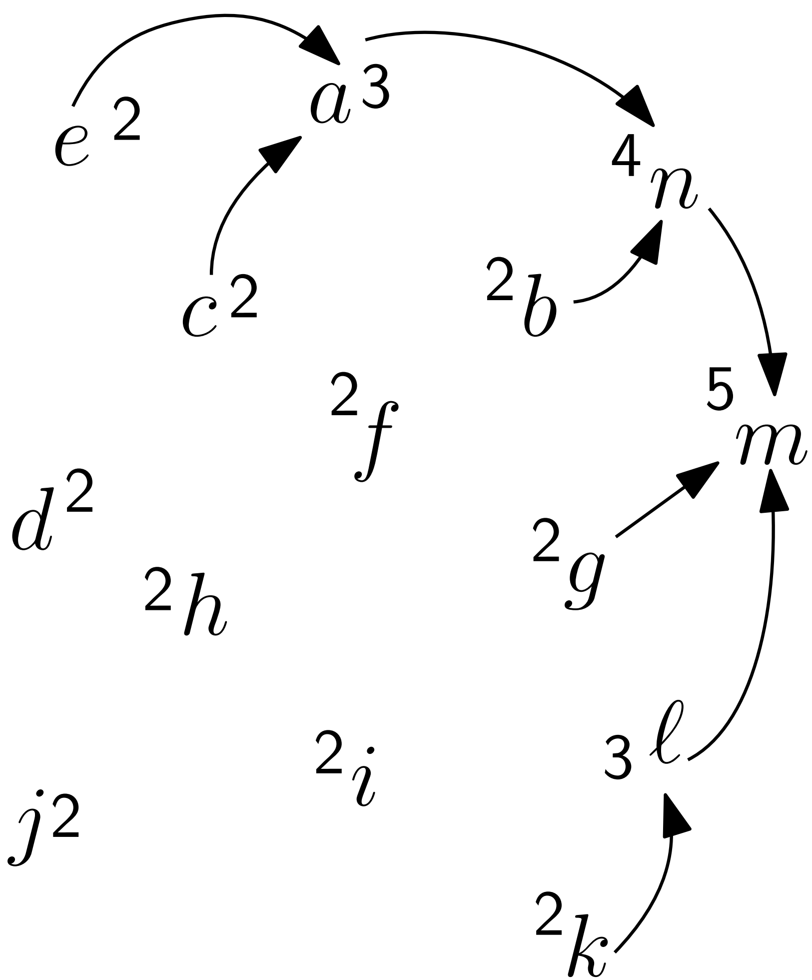
Lemma. The number of nodes in x's subtree is at least $2^{\mathrm{rank(x)}-2}$.

Lemma. The number of elements with rank r is at most $\frac{n}{2^{r-2}} = \frac{4n}{2^r}$.

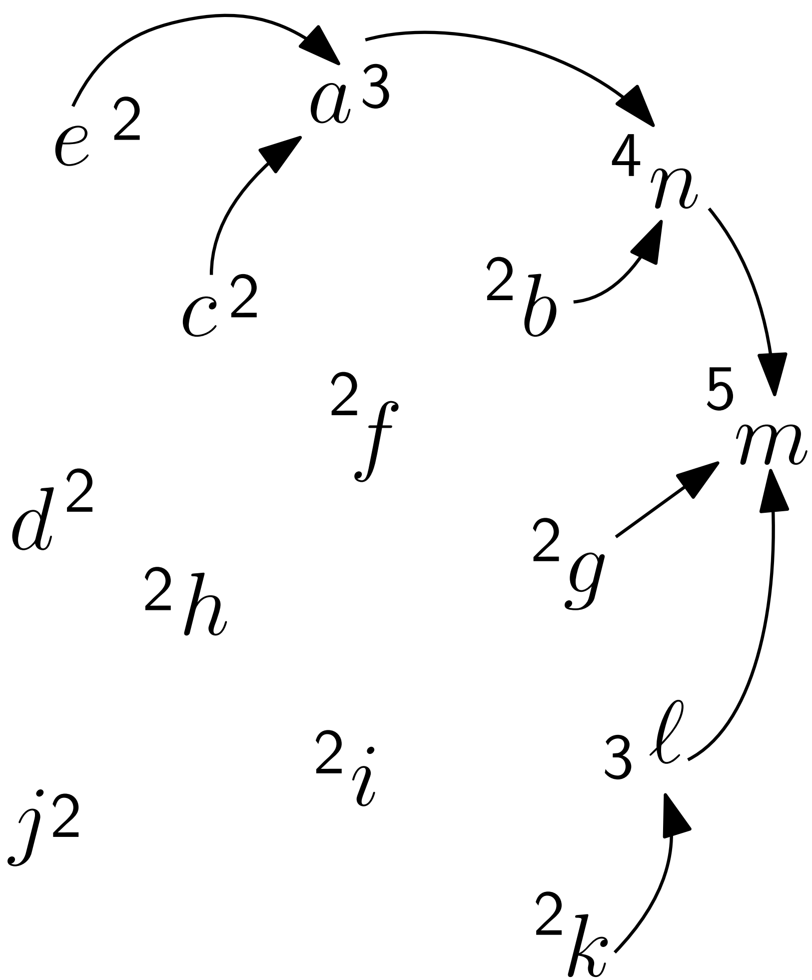
Corollary. The maximum rank is at most $\log(n) + 2$. The maximum height is at most $\log(n)$. The operation $\mathrm{find}(x)$ takes $O(\log n)$ steps.

Path Compression

The Data Structure: Union by Rank

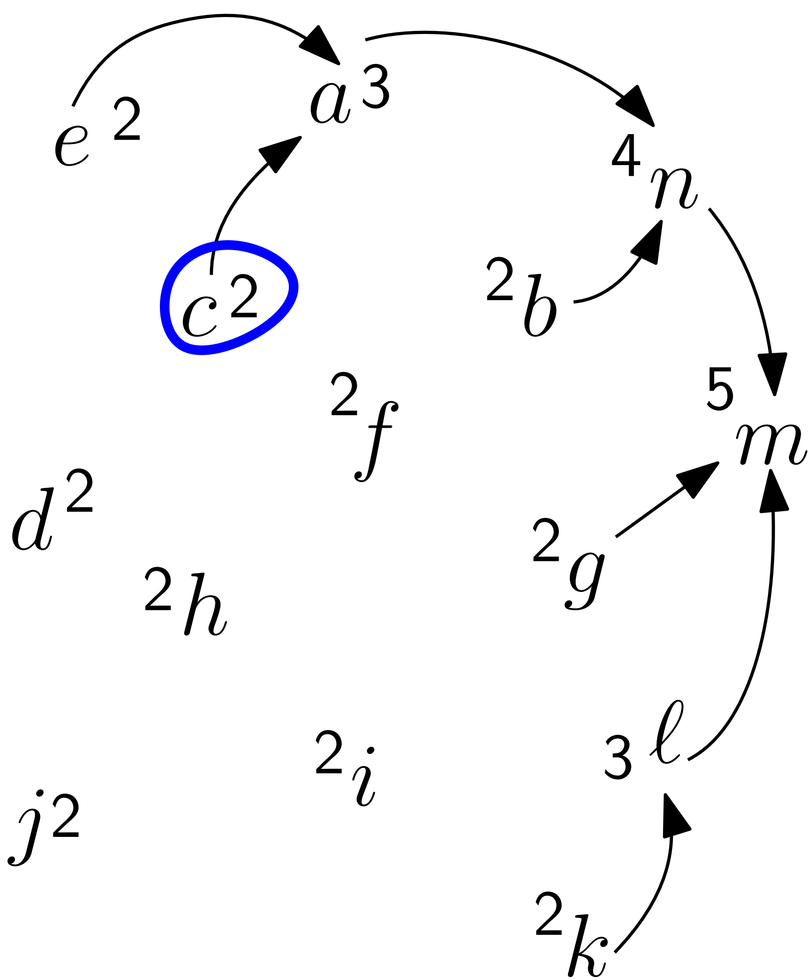


The Data Structure: Union by Rank



$>>>$ `find`(c)

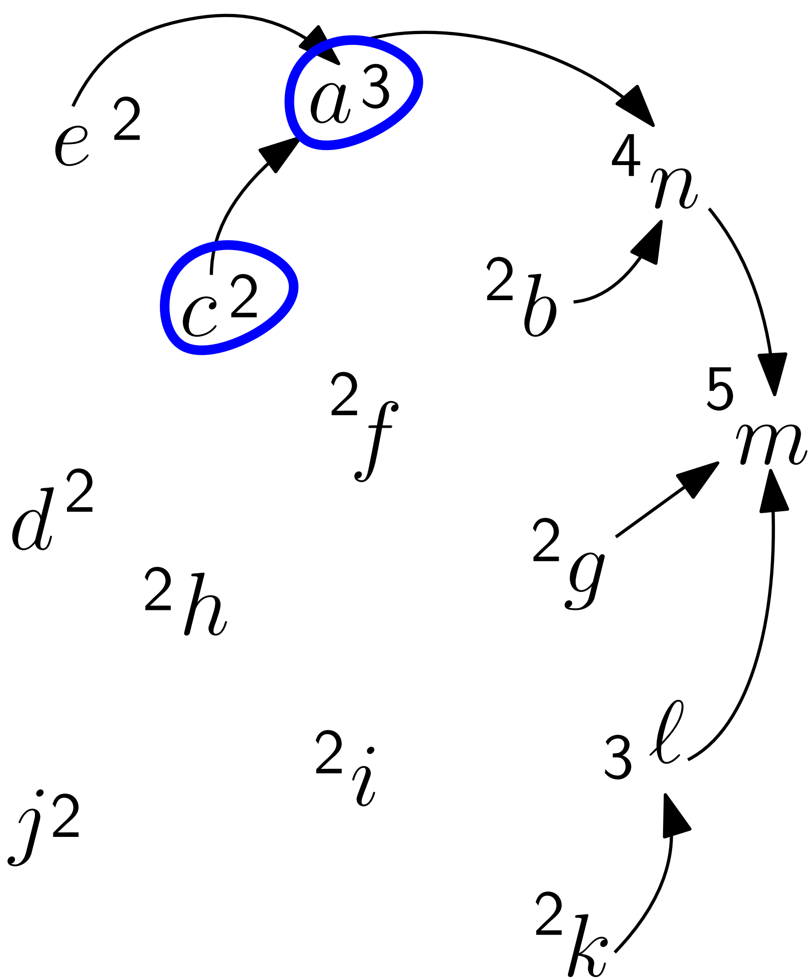
The Data Structure: Union by Rank



$>>>$ `find`(c)

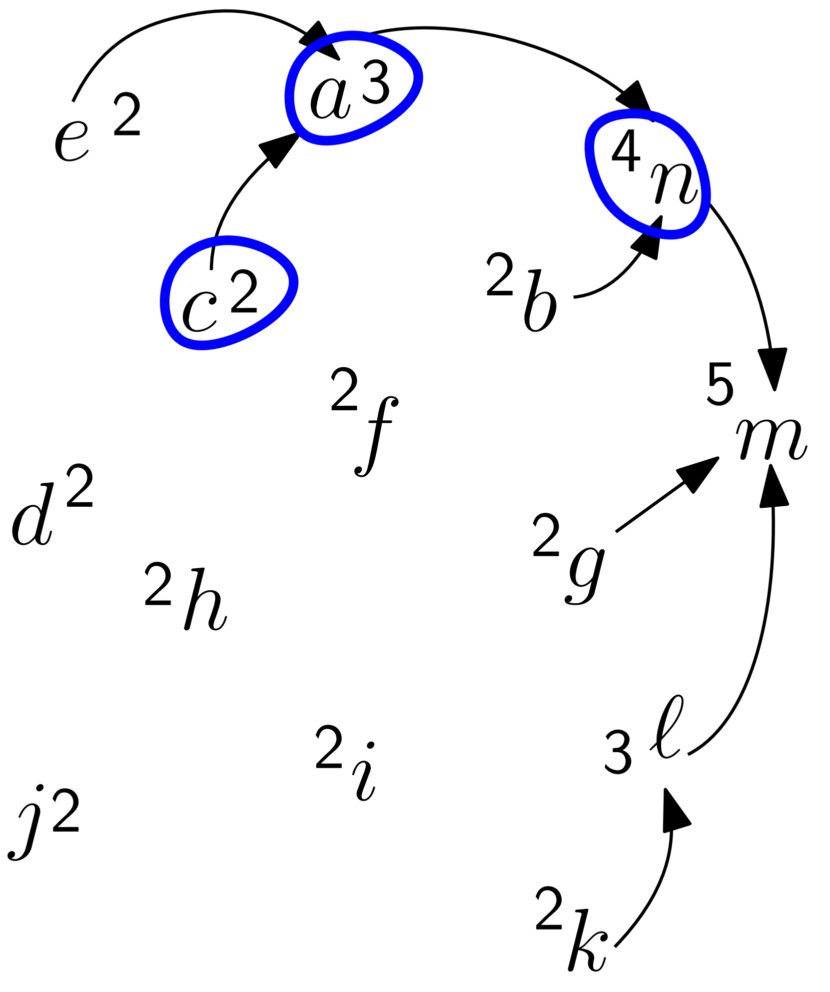
The Data Structure: Union by Rank

$>>>$ `find`(c)

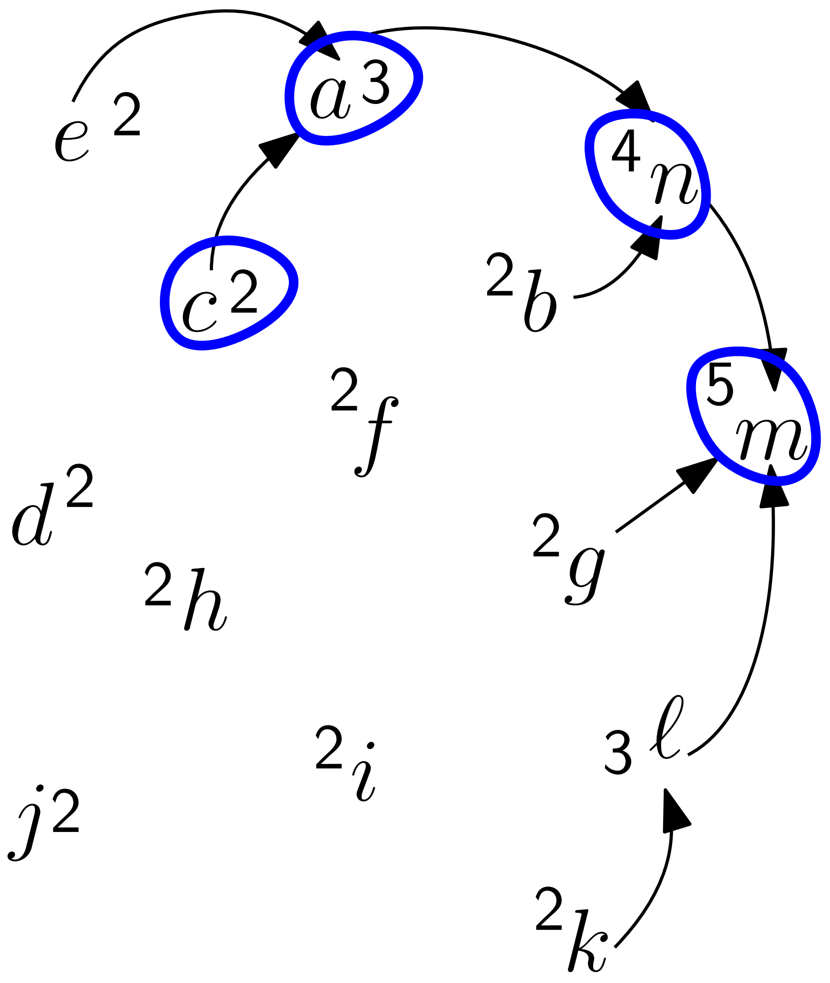


The Data Structure: Union by Rank

$>>>$ `find`(c)

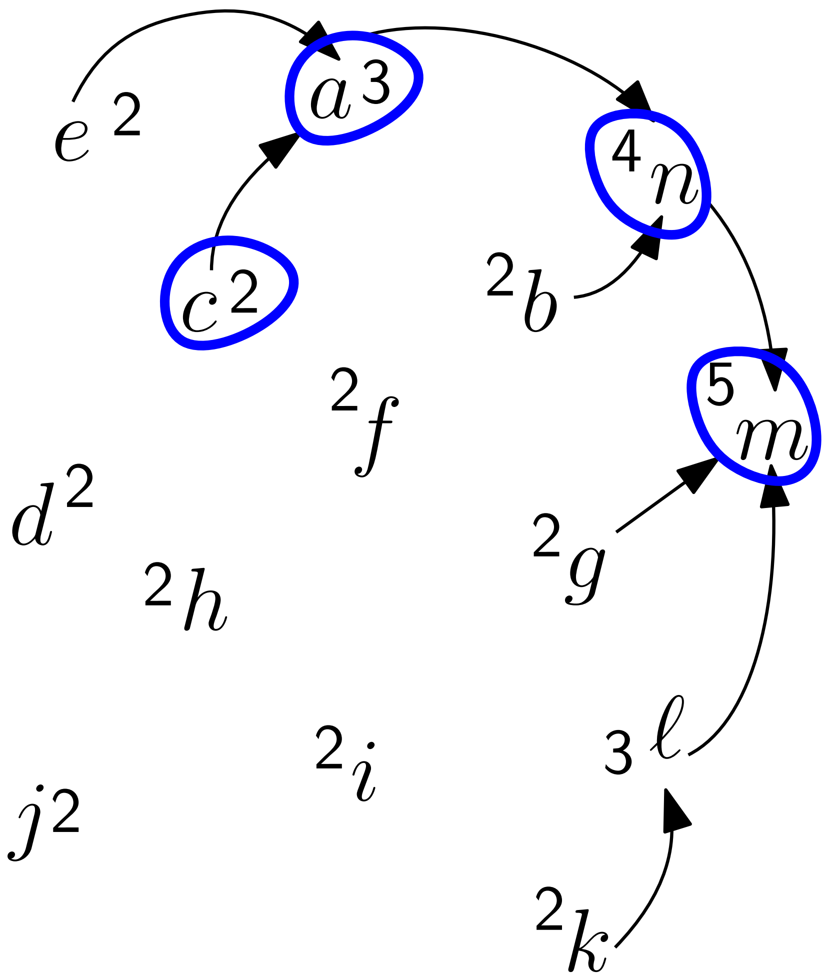


The Data Structure: Union by Rank



$>>>$ `find`(c)

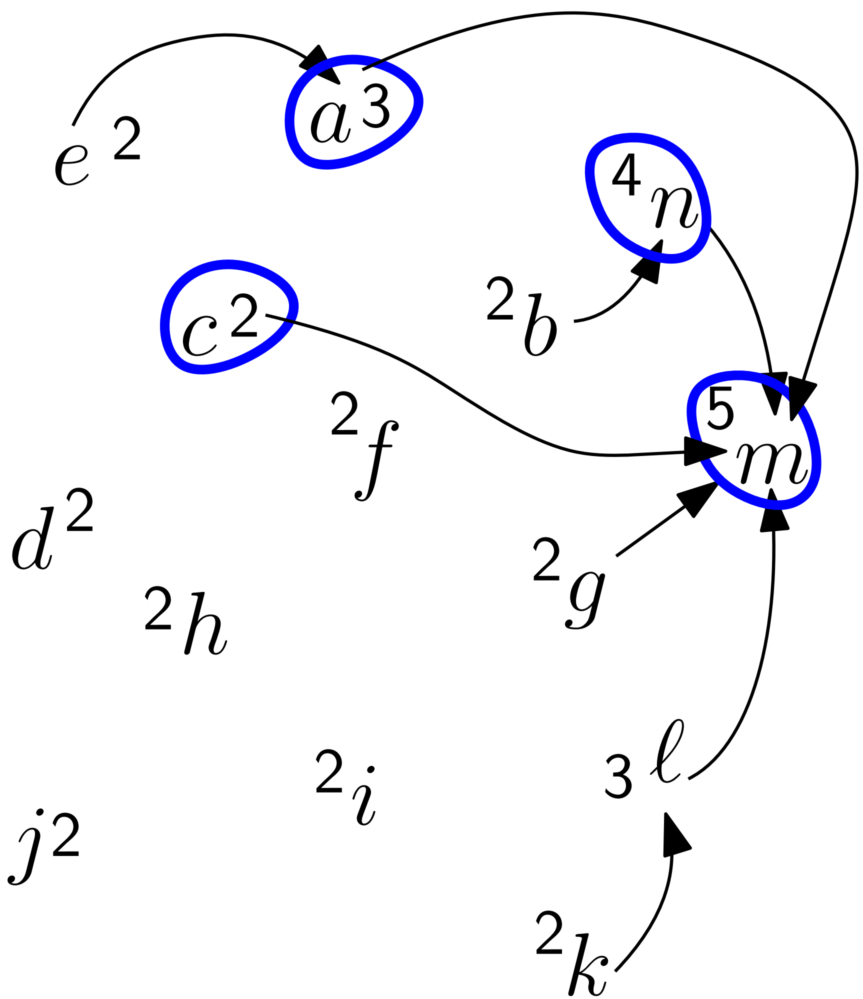
The Data Structure: Union by Rank



$>>>$ `find`(c)

m

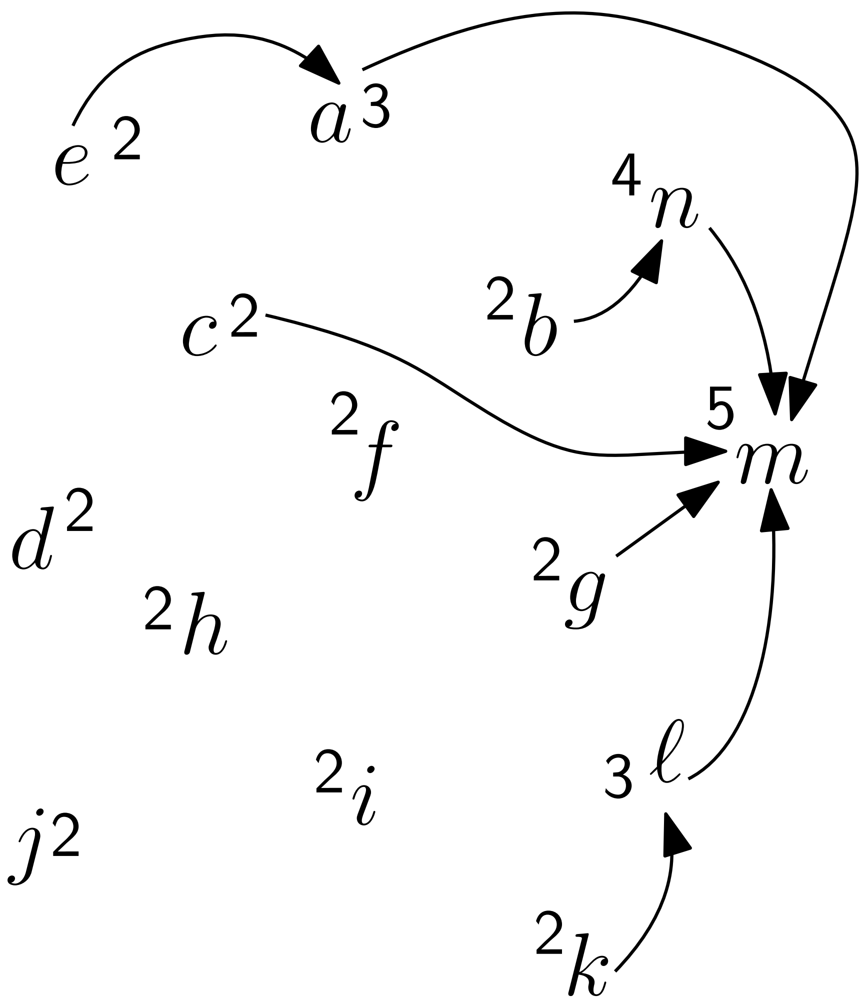
The Data Structure: Union by Rank



$>>>$ `find`(c)

m

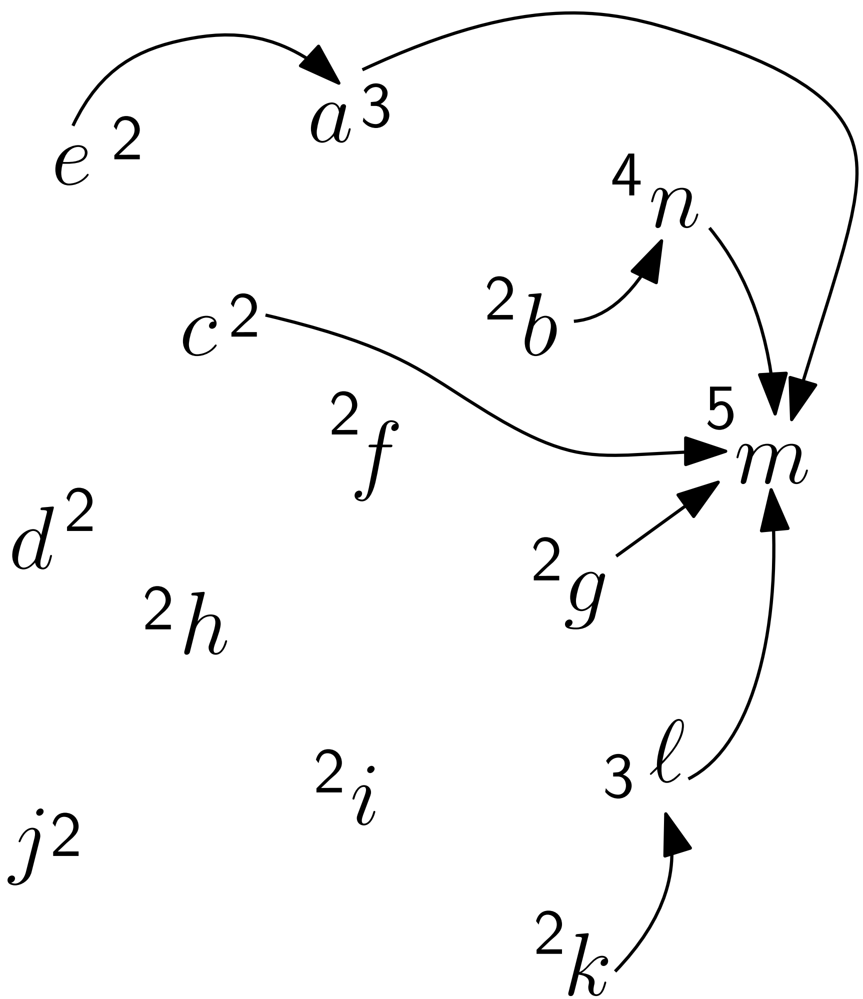
The Data Structure: Union by Rank



$>>>$ `find`(c)

m

The Data Structure: Union by Rank



$>>>$ `find`(c)

m

$\mathrm{rank}(n)$ is still 4. But it's subtree has height 1 and consists of 2 nodes only.

Lemma. The height of the subtree rooted at x is $\mathrm{rank}(x) - 2$.

Lemma. The number of elements with rank r is at most $\frac{n}{2^{r-2}} = \frac{4n}{2^r}$.

Lemma. The number of nodes in x's subtree is at least $2^{\mathrm{rank(x)}-2}$.

Corollary. The maximum rank is at most $\log(n) + 2$. The maximum height is at most $\log(n)$. The operation $\mathrm{find}(x)$ takes $O(\log n)$ steps.

Lemma. The height of the subtree rooted at x is $\mathrm{rank}(x) - 2$.

Lemma. The number of nodes in x's subtree is at least $2^{\mathrm{rank(x)}-2}$.

Lemma. The number of elements with rank r is at most $\frac{n}{2^{r-2}} = \frac{4n}{2^r}$.

Corollary. The maximum rank is at most $\log(n) + 2$. The maximum height is at most $\log(n)$. The operation $\mathrm{find}(x)$ takes $O(\log n)$ steps.

Lemma. The height of the subtree rooted at x is $\mathrm{rank}(x) - 2$.

Lemma. The number of nodes in x's subtree is at least $2^{\mathrm{rank(x)}-2}$.

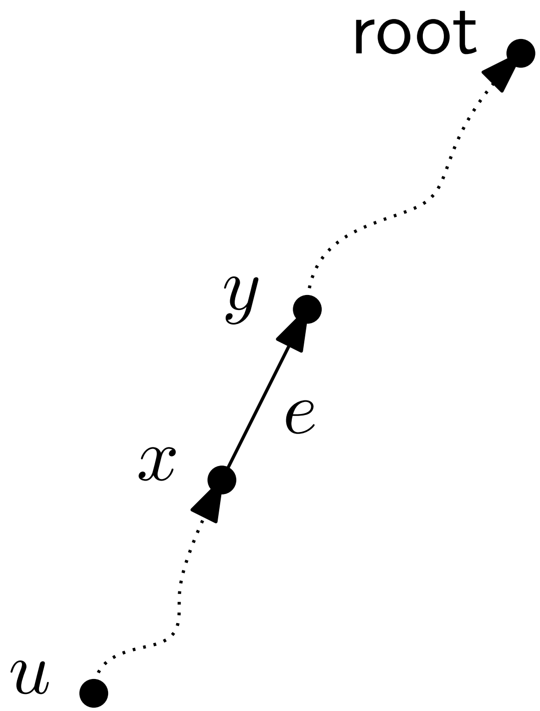
Lemma. The number of elements with rank r is at most $\frac{n}{2^{r-2}} = \frac{4n}{2^r}$.

Corollary. The maximum rank is at most $\log(n) + 2$. The maximum height is at most $\log(n)$. The operation $\mathrm{find}(x)$ takes $O(\log n)$ steps.

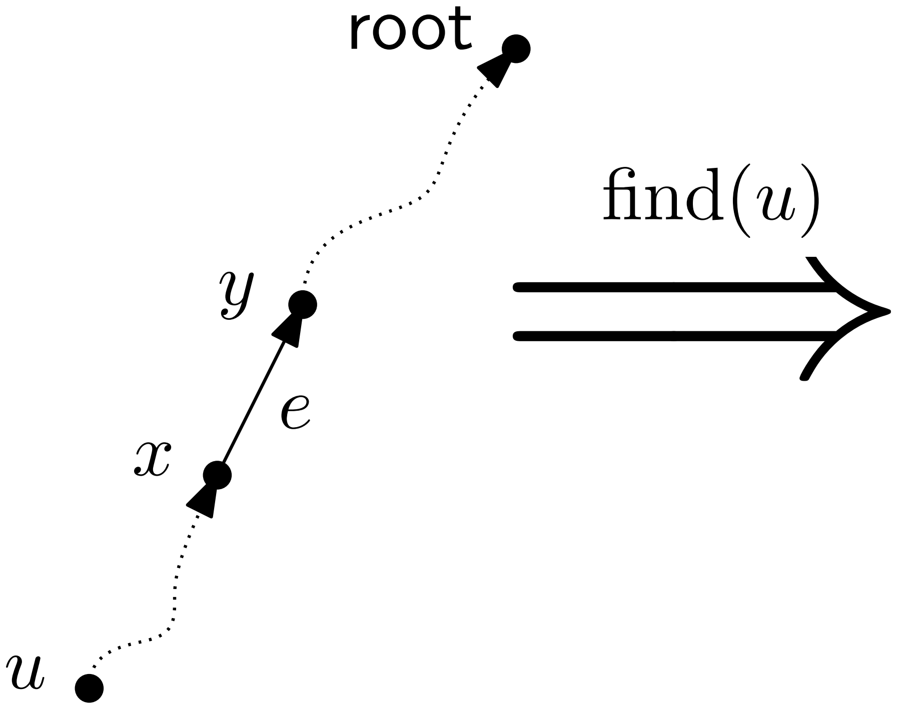
Running Time Analysis of the Path Compression Data Structure

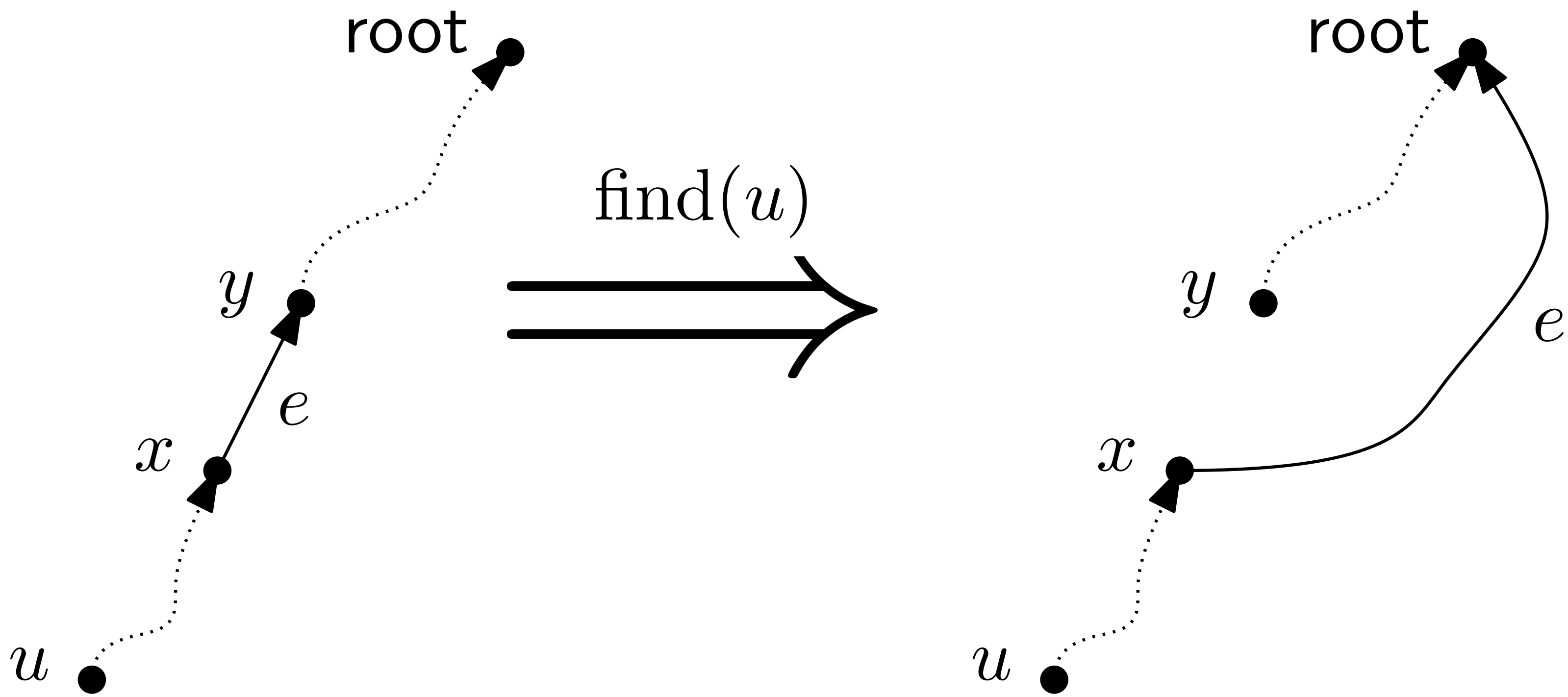
Running Time Analysis of the Path Compression Data Structure

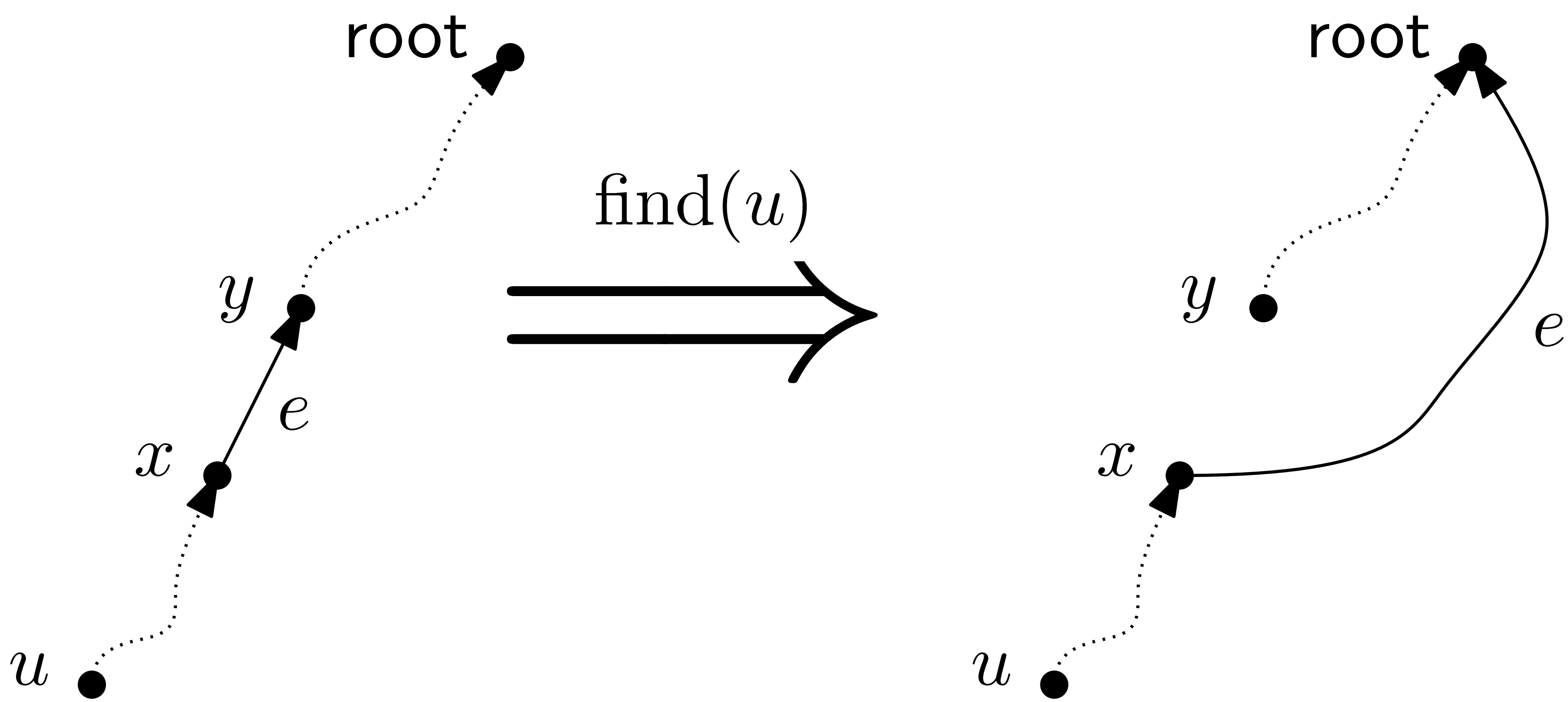
First Attempt



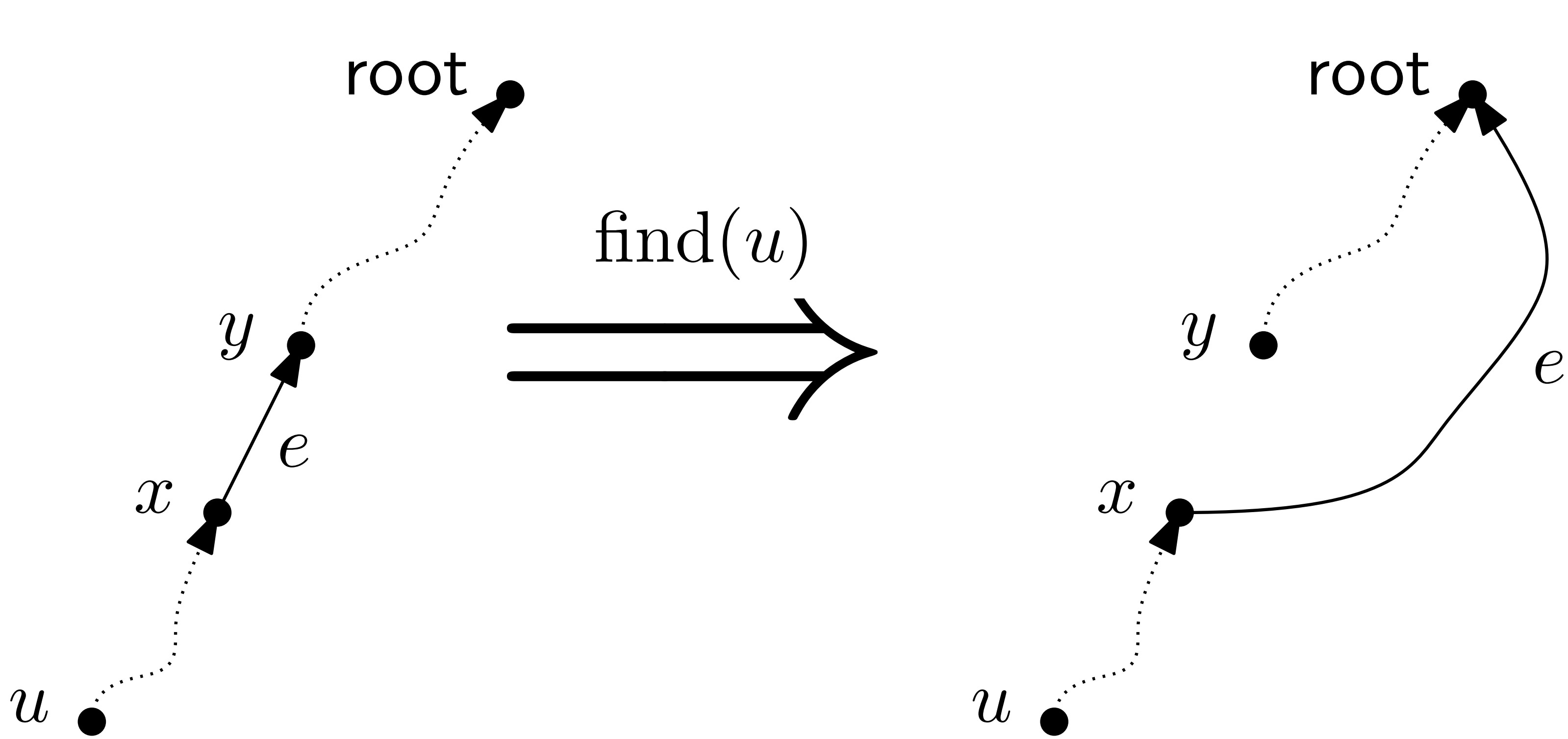
root
y
e
x
u


root
find(u)
y
e
x
u


root
y
e
x
u
find(u)
root
y
e
x
u

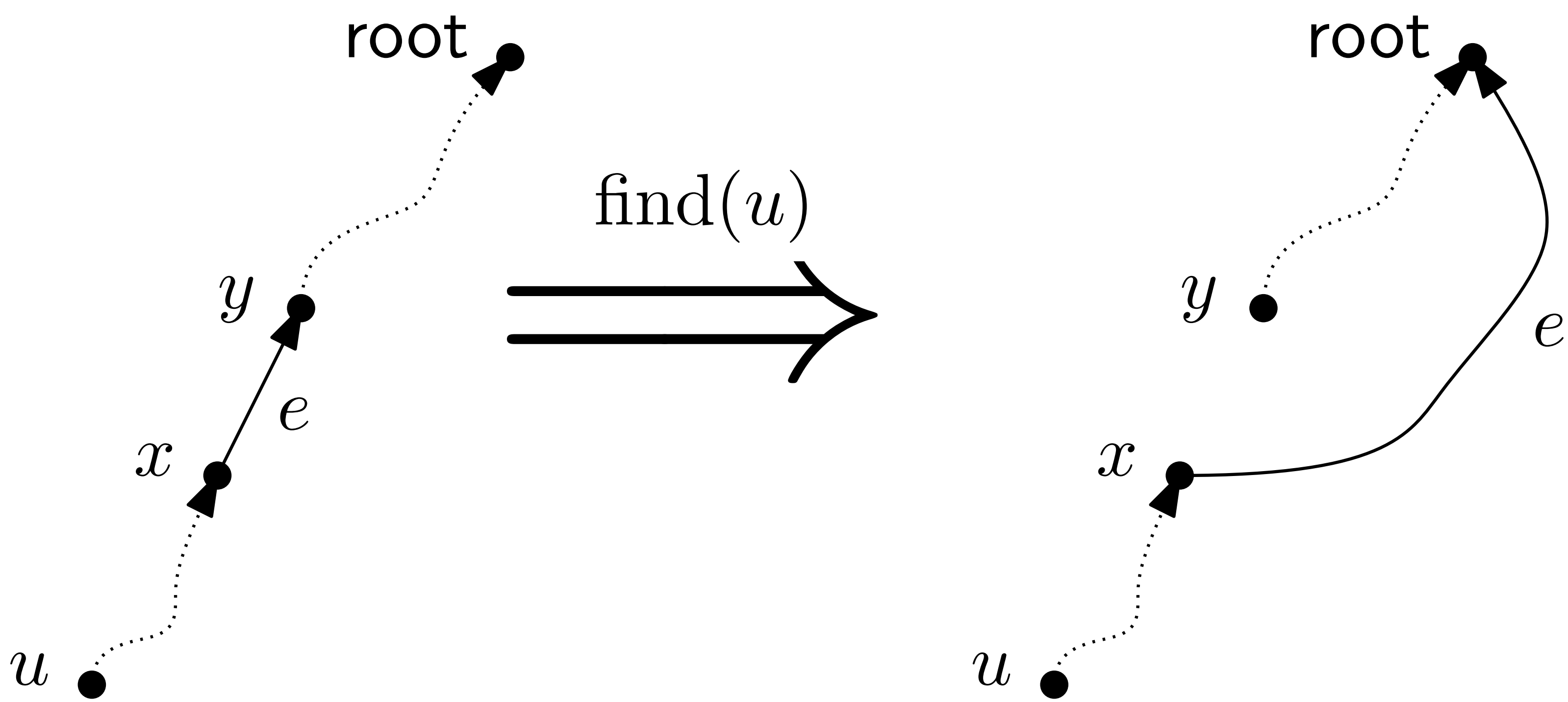


Edge e gets redirected.



Edge e gets redirected.

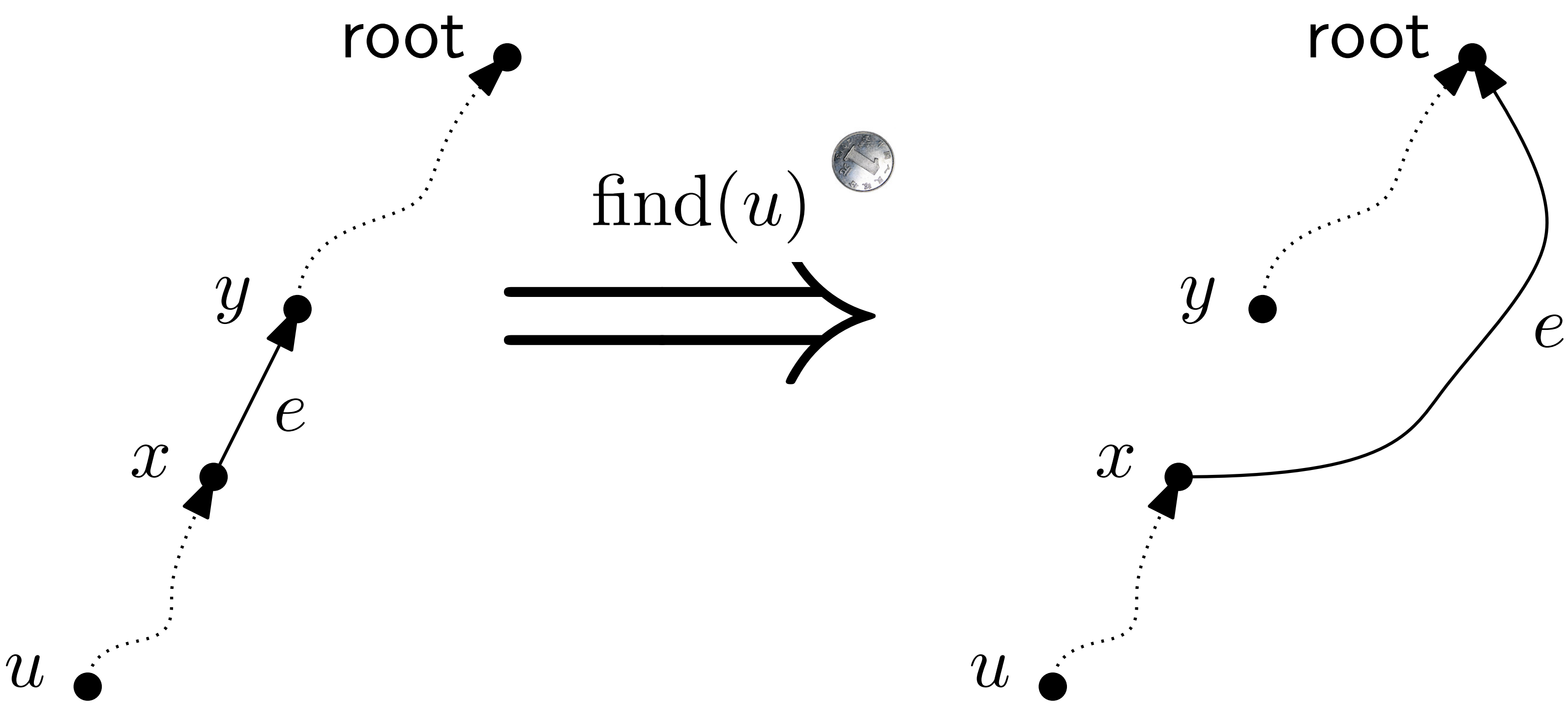
This operation costs 1 



Edge e gets redirected.

This operation costs 1 

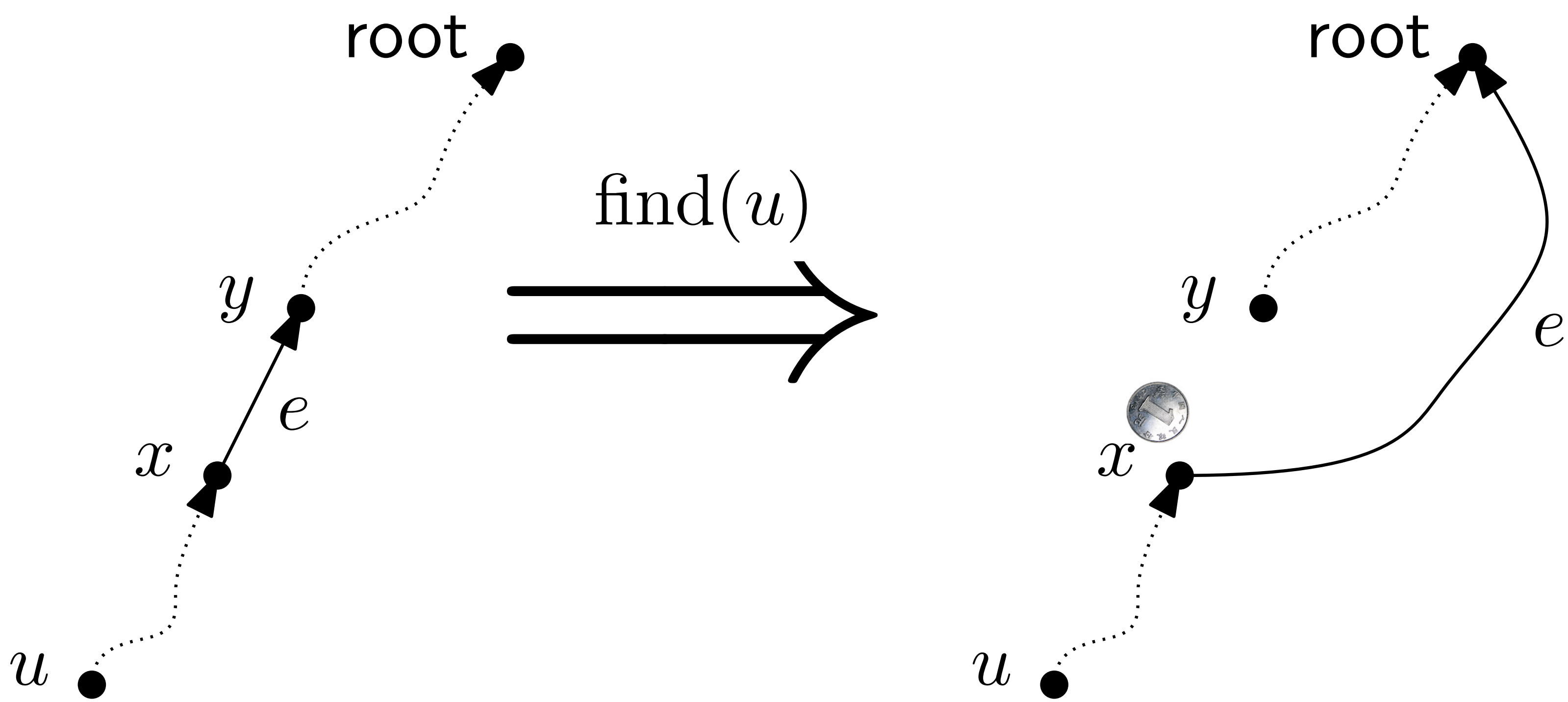
Who has to pay?



Edge e gets redirected.

This operation costs 1 

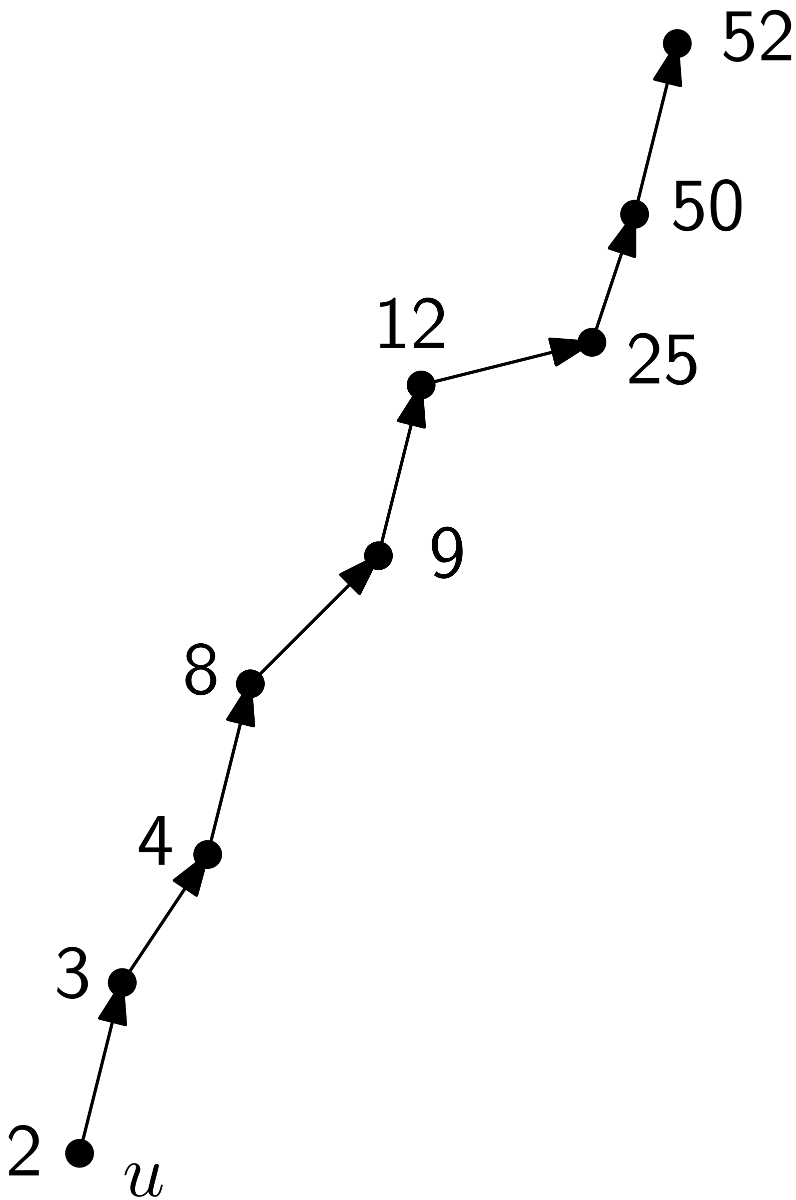
Who has to pay?



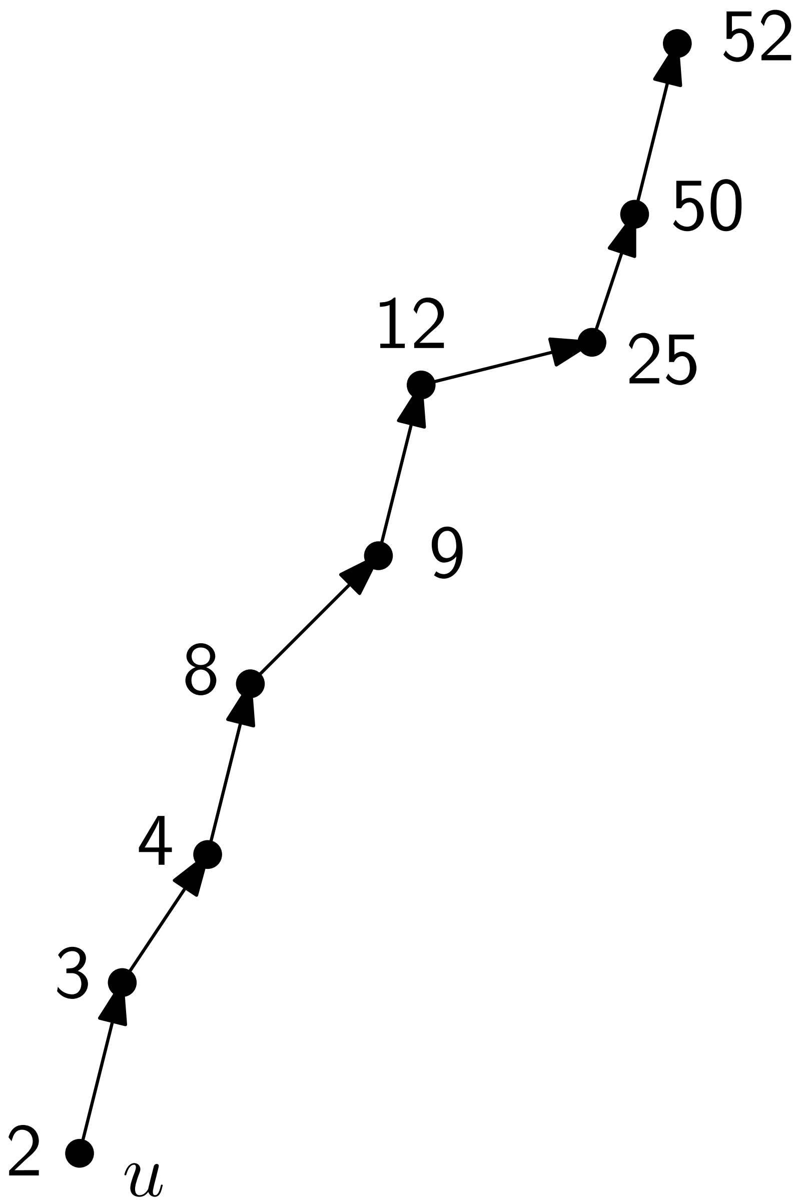
Edge e gets redirected.

This operation costs 1 

Who has to pay?

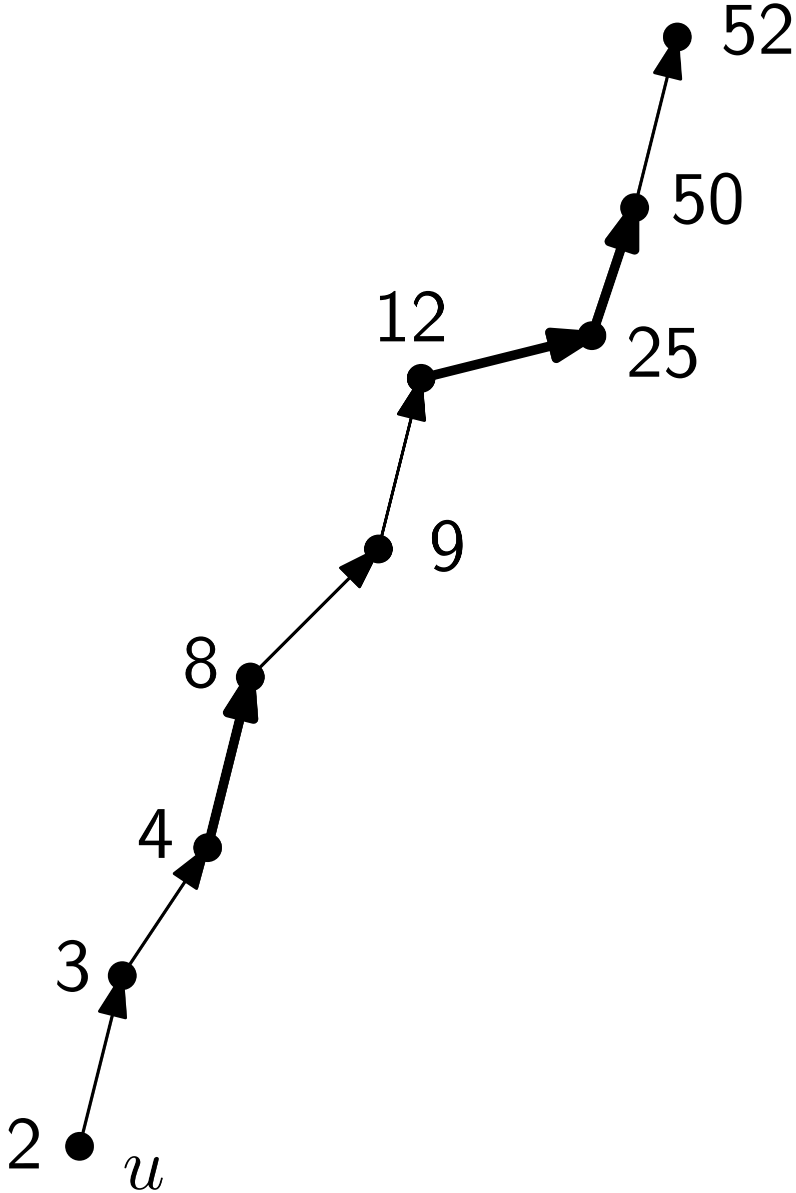


52
50
12
25
9
8
4
3
2
u



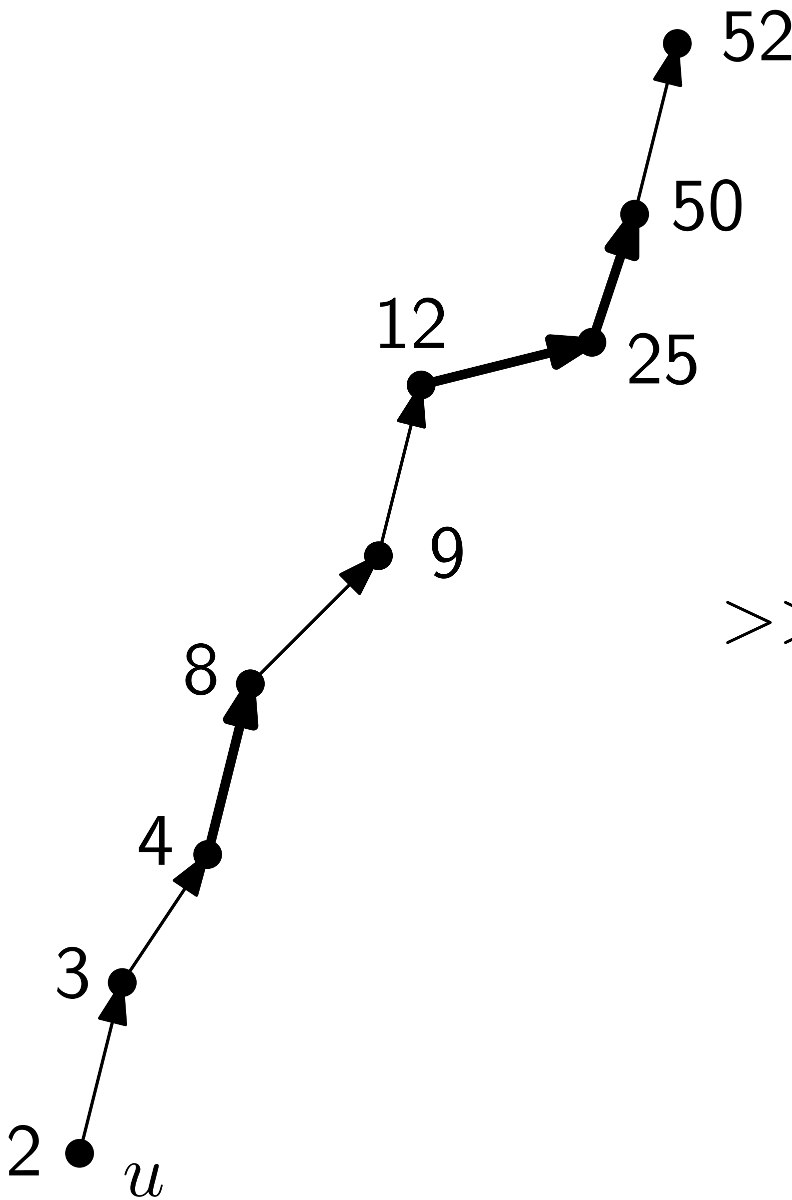
Definition. An edge

is *thick* if $\mathrm{rank}(y) \geq 2\ \mathrm{rank}(x)$.



Definition. An edge 

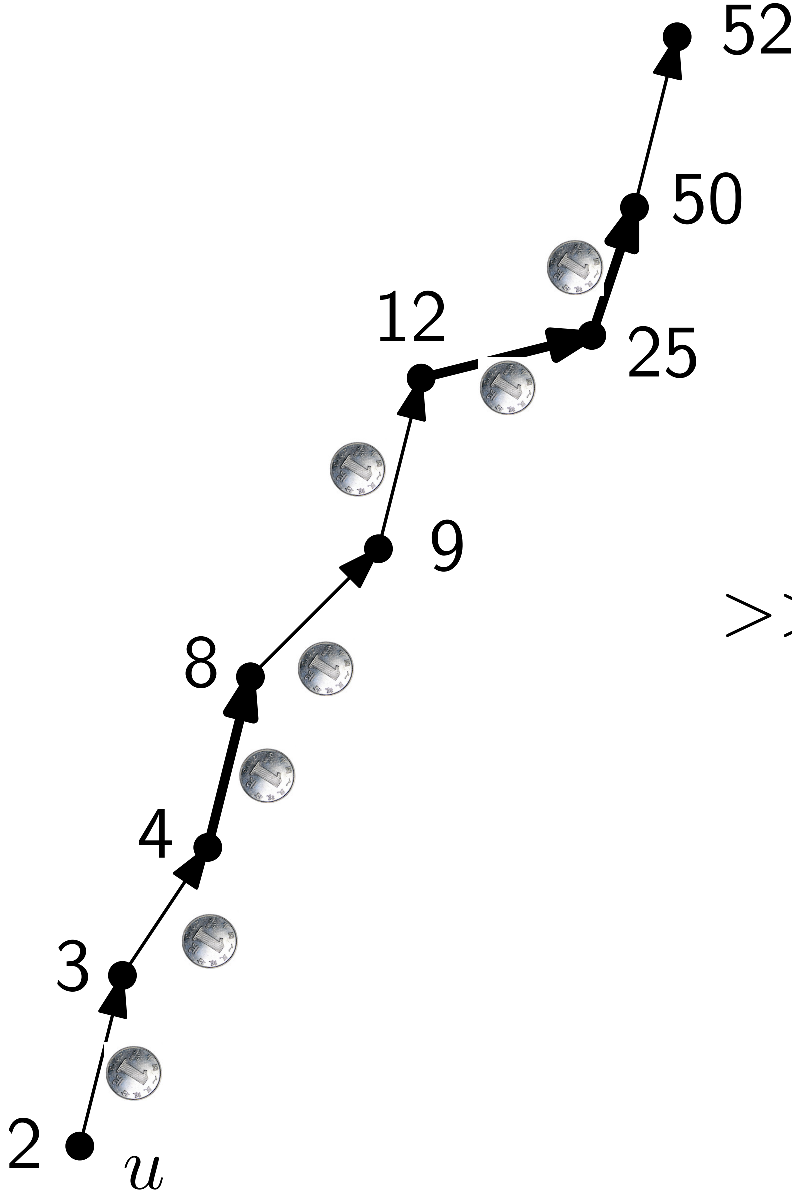
is *thick* if $\mathrm{rank}(y) \geq 2\ \mathrm{rank}(x)$.



Definition. An edge 

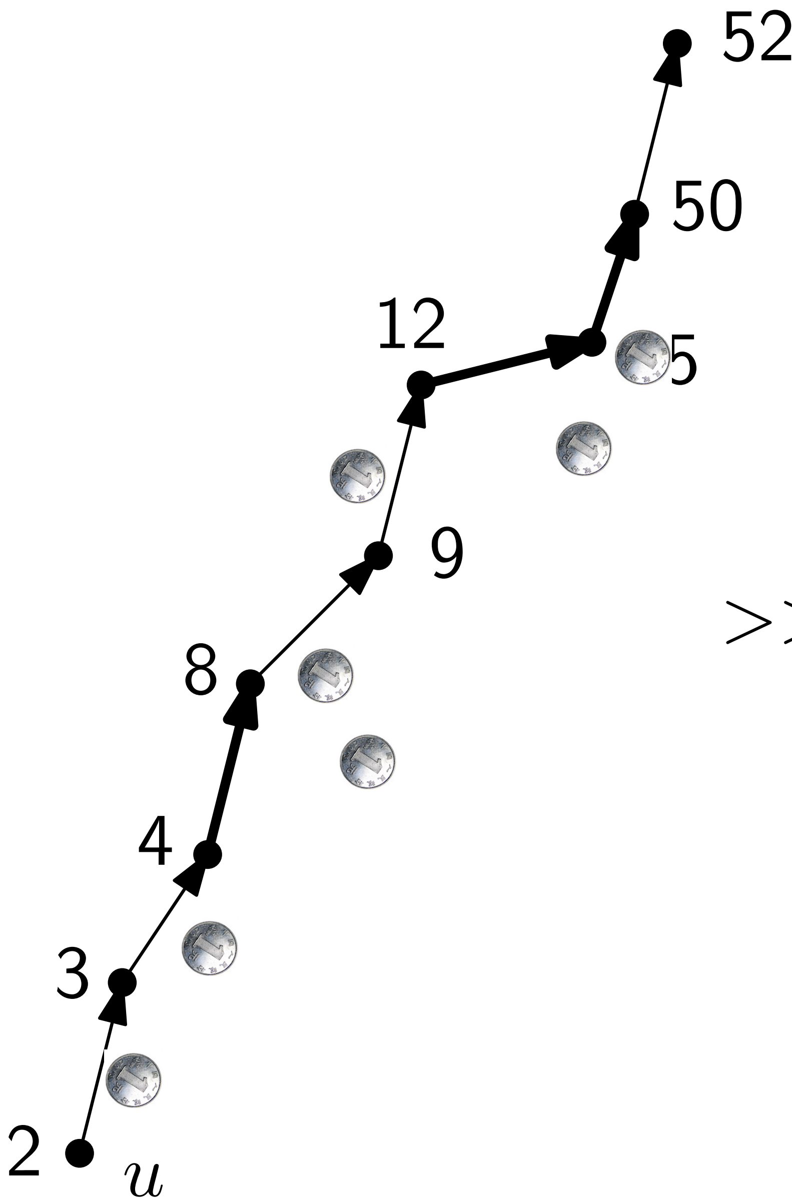
is *thick* if $\mathrm{rank}(y) \geq 2\ \mathrm{rank}(x)$.

$>>>$ `find`(u)



Definition. An edge $x \to y$ is *thick* if $\mathrm{rank}(y) \geq 2\ \mathrm{rank}(x)$.

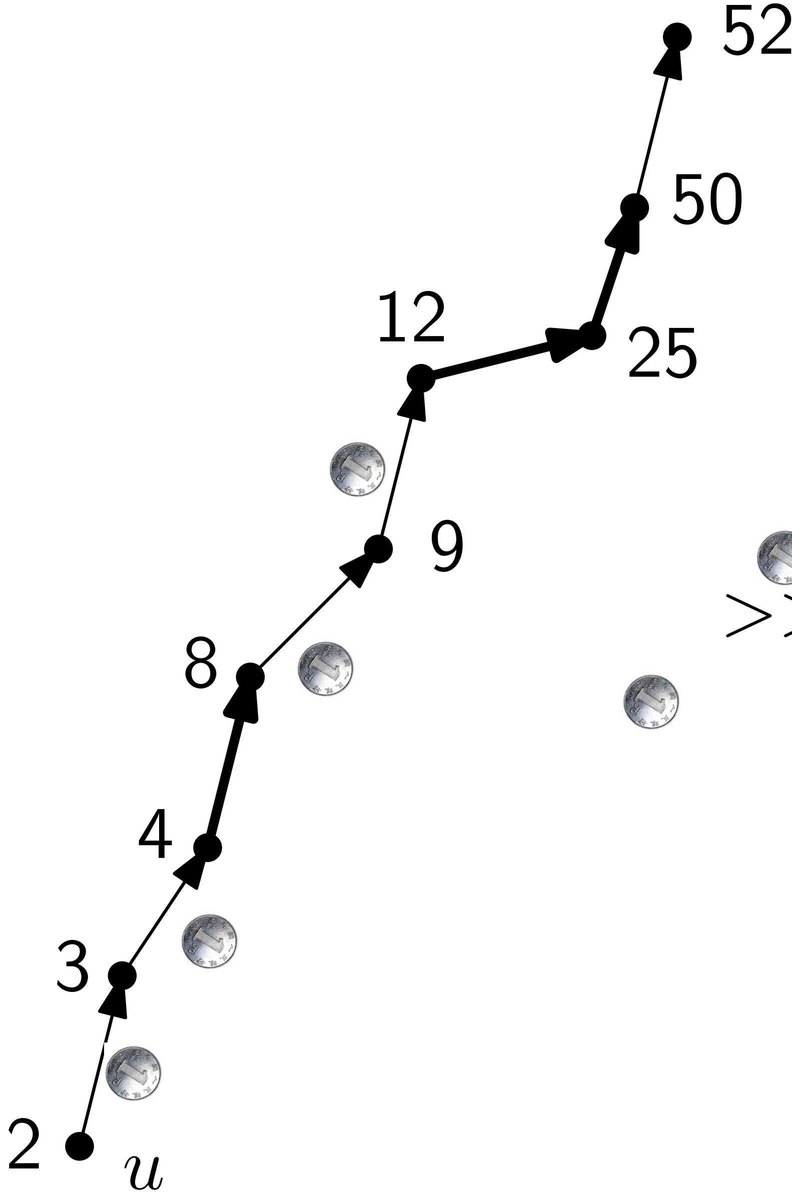
$>>>$ `find`(u)



Definition. An edge 

is *thick* if $\mathrm{rank}(y) \geq 2\ \mathrm{rank}(x)$.

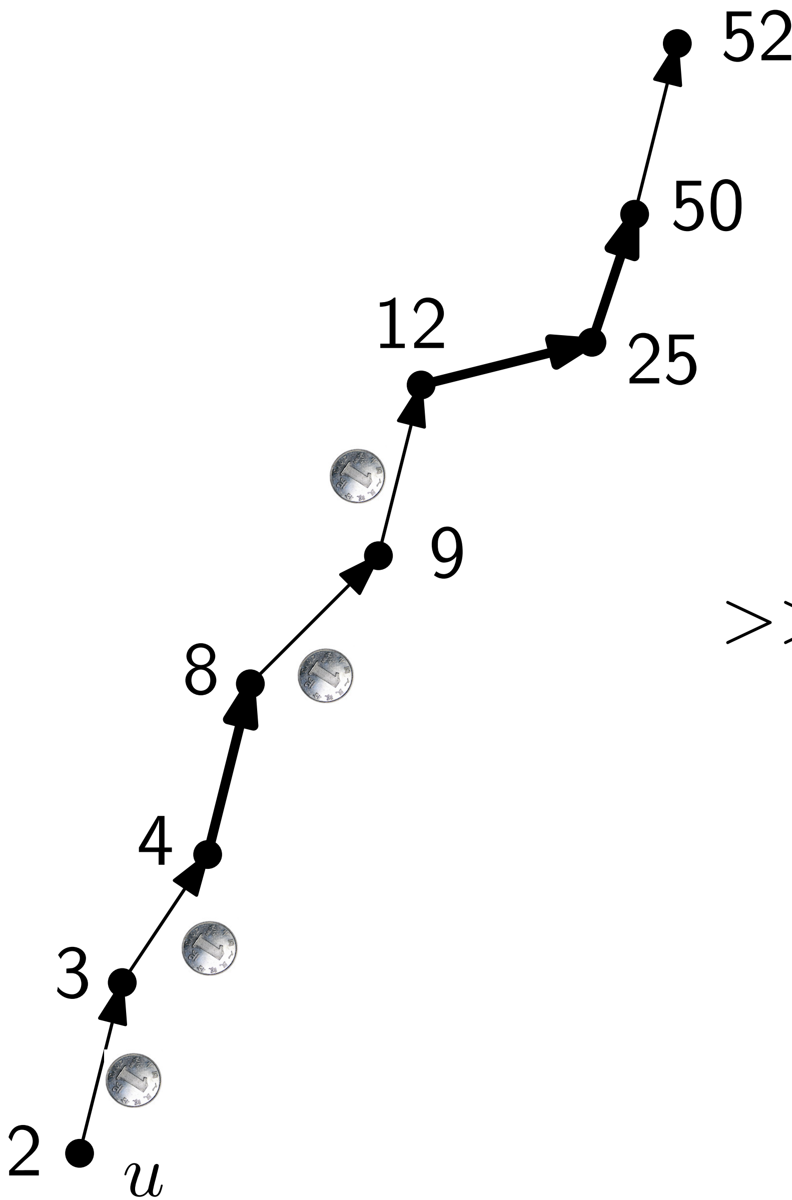
$>>>$ `find`(u)



Definition. An edge 

is *thick* if $\mathrm{rank}(y) \geq 2\ \mathrm{rank}(x)$.

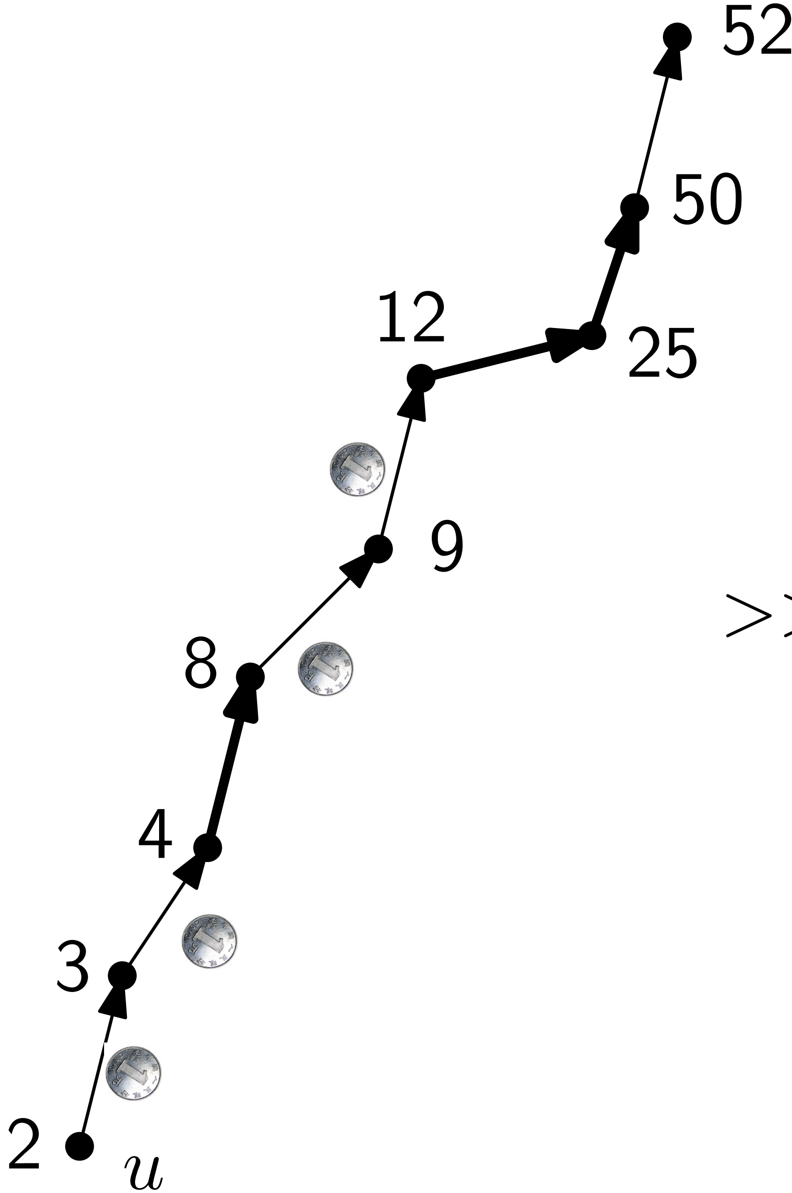
$>>>$ `find`(u)



Definition. An edge 

is *thick* if $\mathrm{rank}(y) \geq 2\ \mathrm{rank}(x)$.

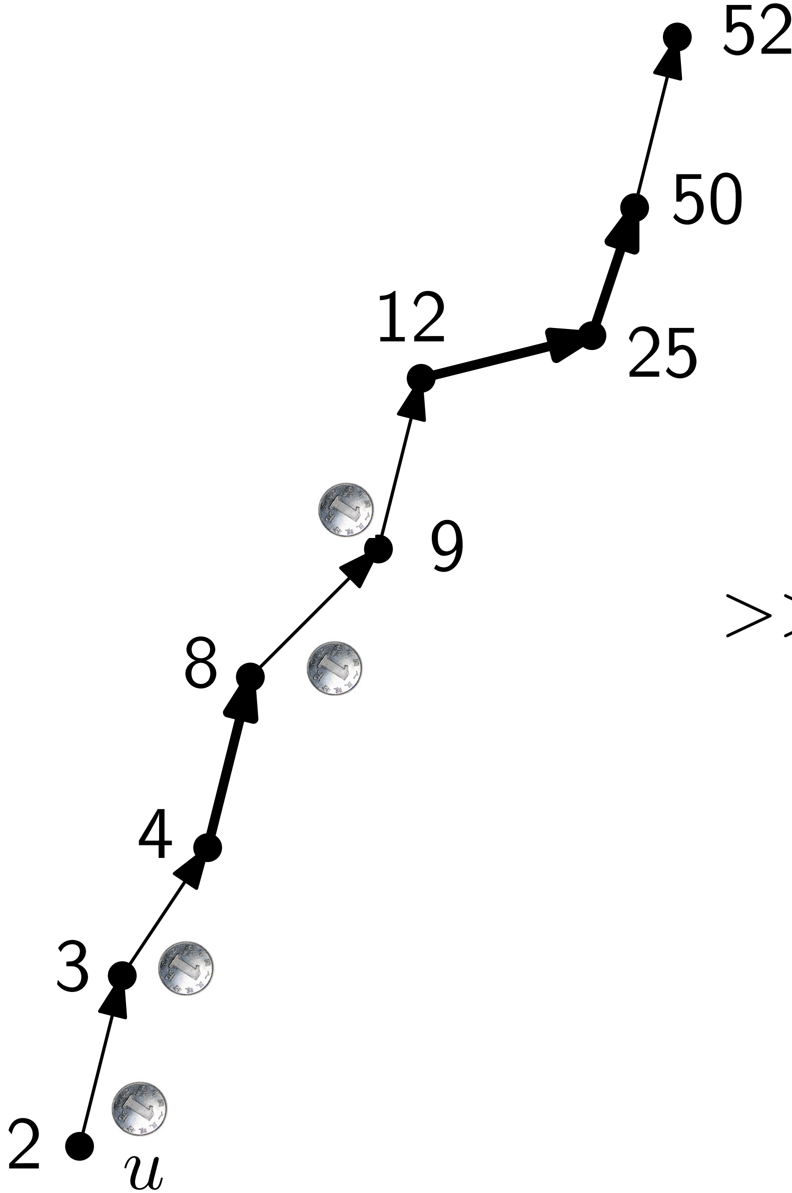
$>>>$ `find`(u)



Definition. An edge 

is *thick* if $\mathrm{rank}(y) \geq 2\ \mathrm{rank}(x)$.

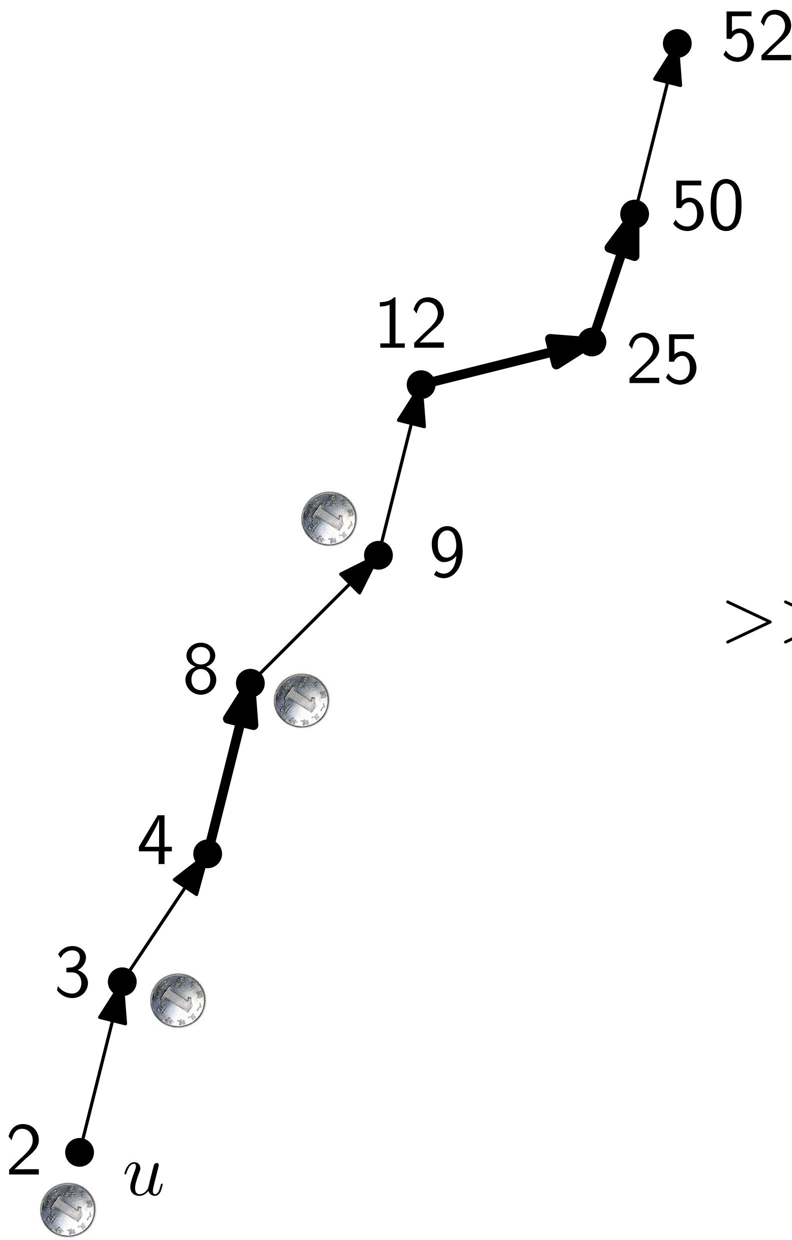
$>>>$ `find`(u) 



Definition. An edge 

is *thick* if $\mathrm{rank}(y) \geq 2\,\mathrm{rank}(x)$.

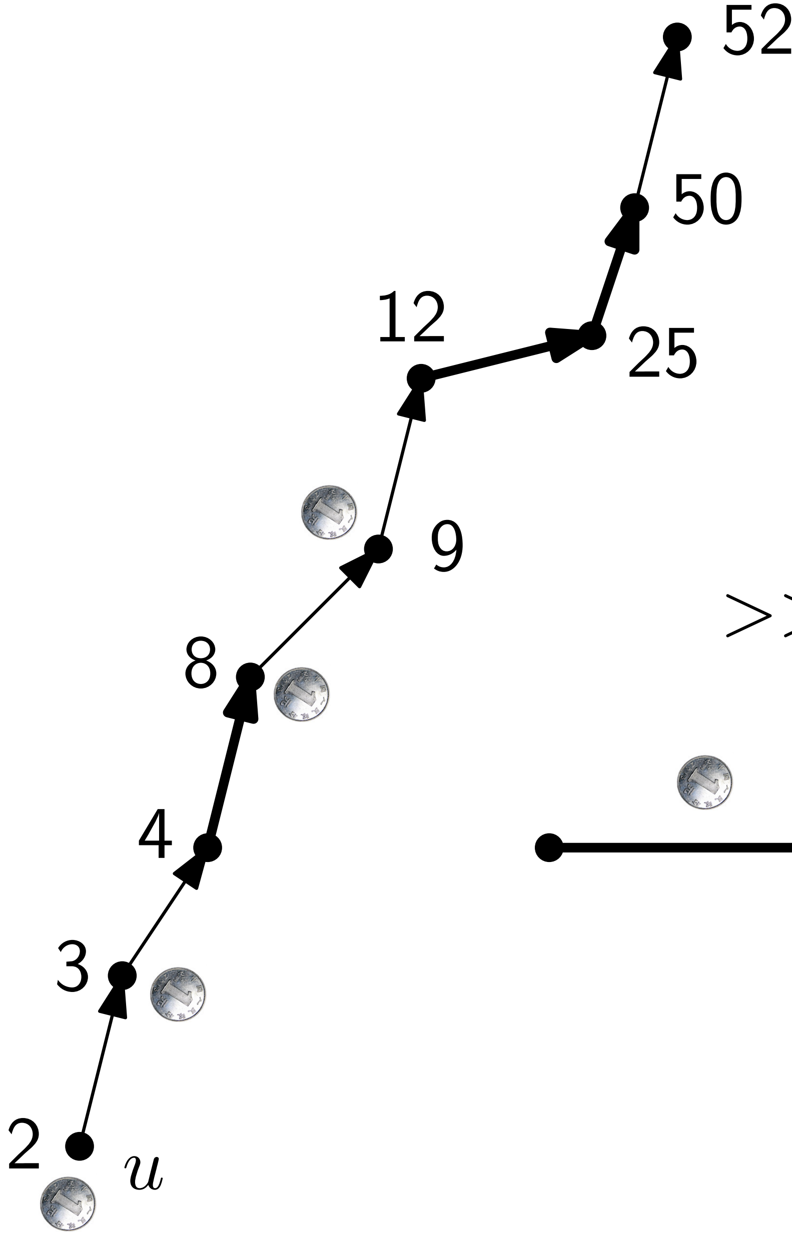
$>>>$ `find`(u) 



Definition. An edge 

is *thick* if $\mathrm{rank}(y) \geq 2\ \mathrm{rank}(x)$.

$>>>$ $\texttt{find}(u)$

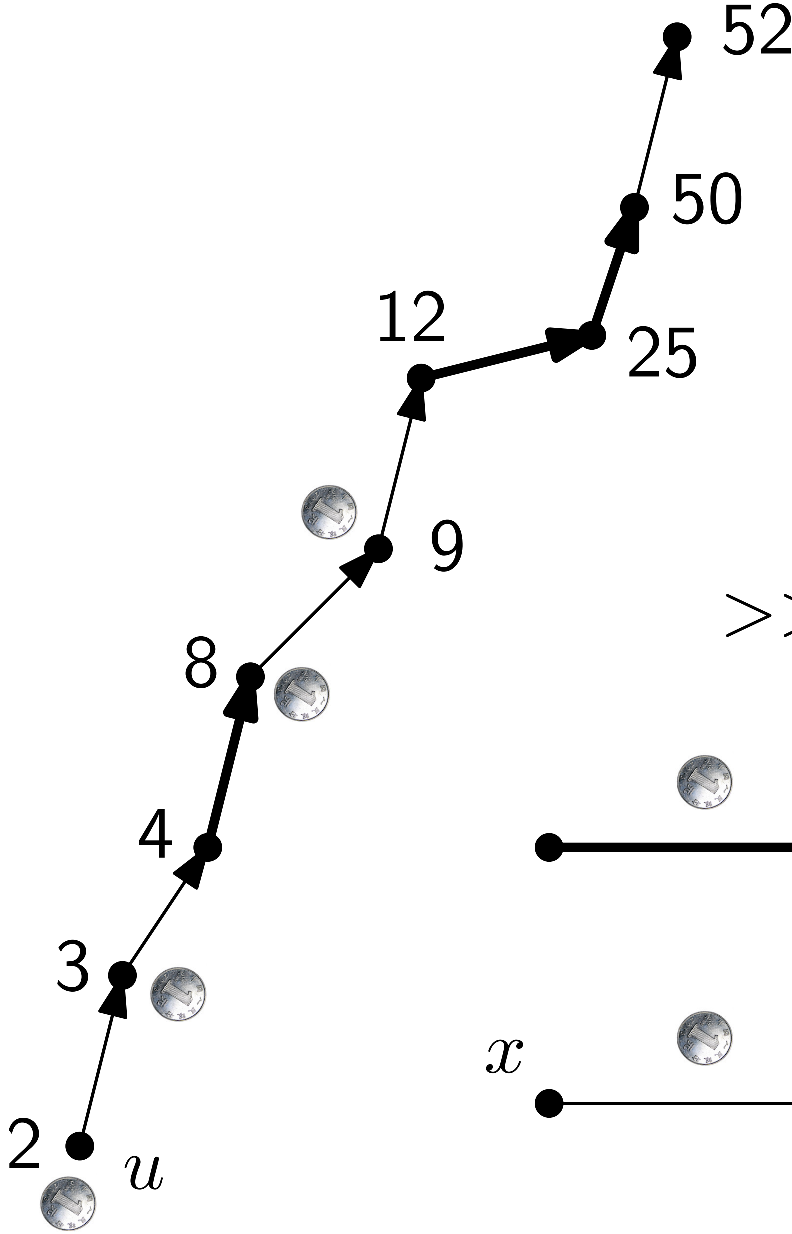


Definition. An edge 

is *thick* if $\mathrm{rank}(y) \geq 2\ \mathrm{rank}(x)$.

$>>>$ `find`(u) 

is paid by `find`



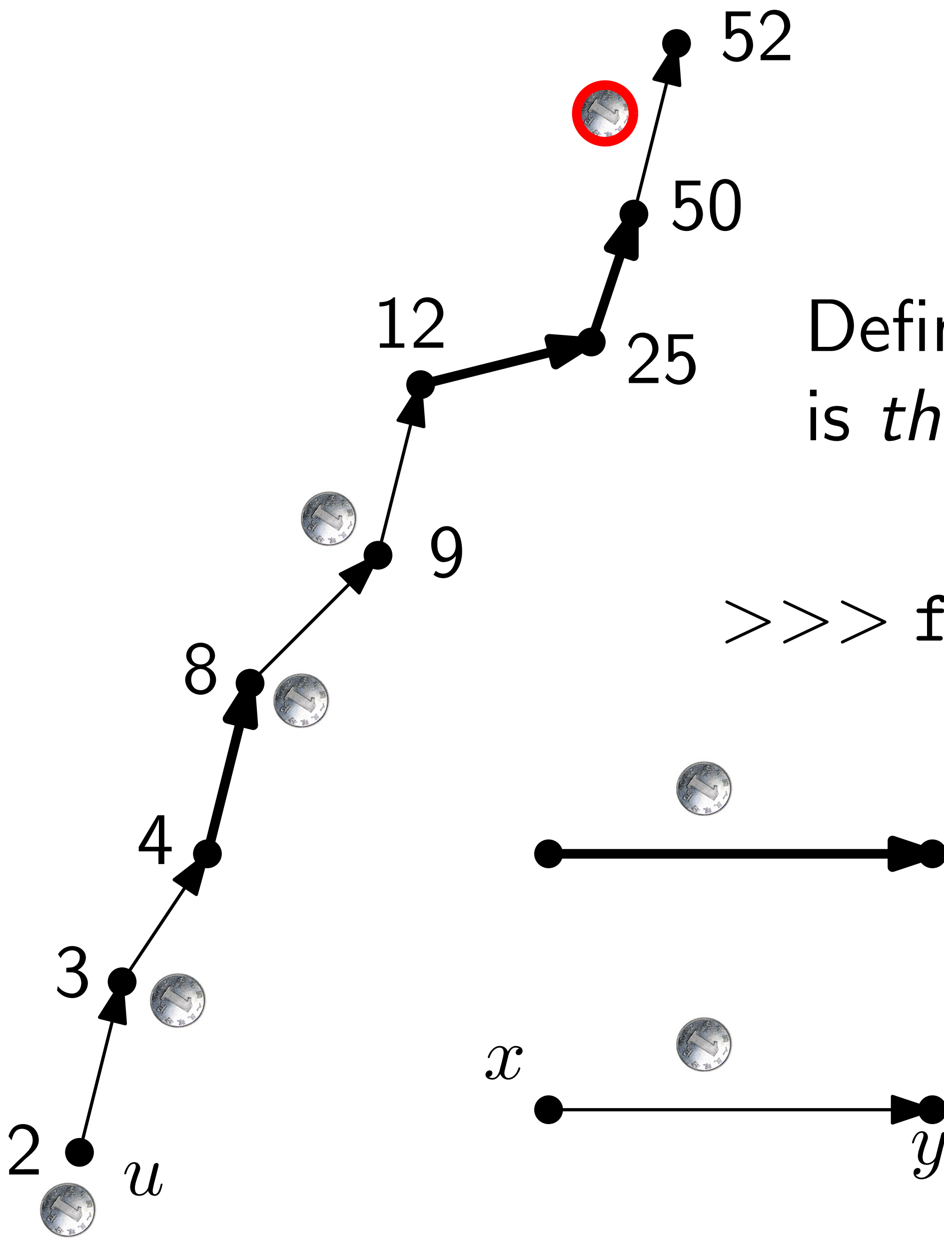
Definition. An edge 

is *thick* if $\mathrm{rank}(y) \geq 2\,\mathrm{rank}(x)$.

$>>>$ `find`(u) 

is paid by `find`

is paid by x



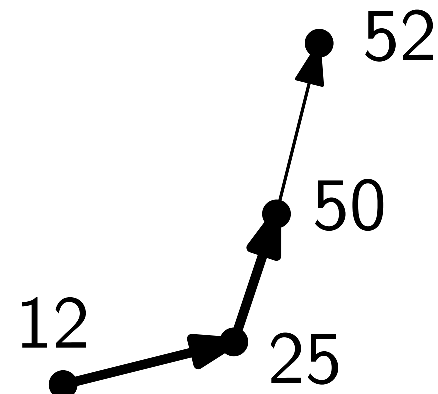
Definition. An edge 

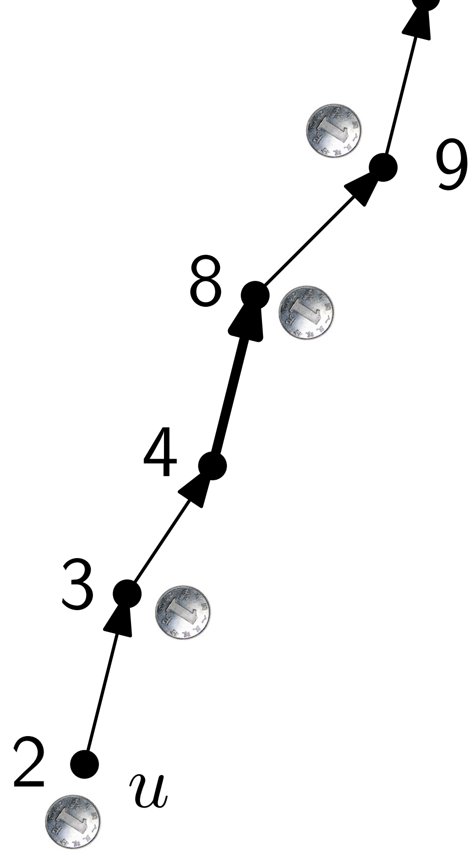
is *thick* if $\mathrm{rank}(y) \geq 2\ \mathrm{rank}(x)$.

$>>>$ `find`(u)

is paid by `find`

is paid by x



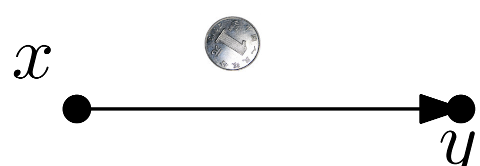


Definition. An edge 

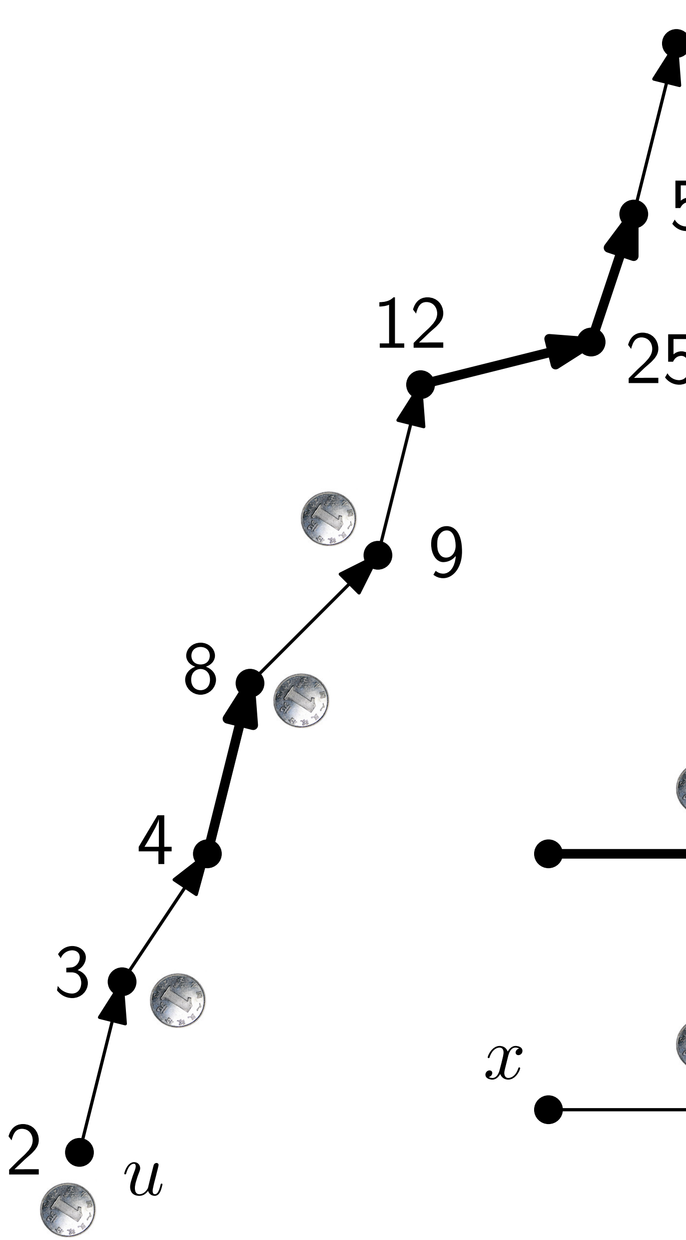
is *thick* if $\mathrm{rank}(y) \geq 2\,\mathrm{rank}(x)$.

$>>>$ `find`(u) 

 is paid by `find`



is paid by x

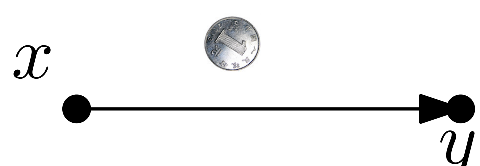


Definition. An edge 

is *thick* if $\mathrm{rank}(y) \geq 2\ \mathrm{rank}(x)$.

$>>>$ `find`(u) 

 is paid by `find`

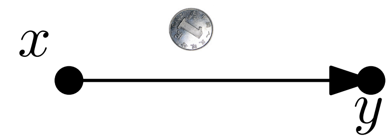


is paid by x

Definition. An edge 

is *thick* if $\mathrm{rank}(y) \geq 2\ \mathrm{rank}(x)$.

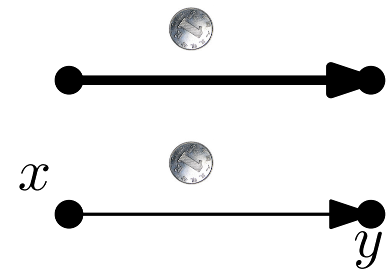

is paid by `find`



is paid by x

Definition. An edge 

is *thick* if $\text{rank}(y) \geq 2\ \text{rank}(x)$.



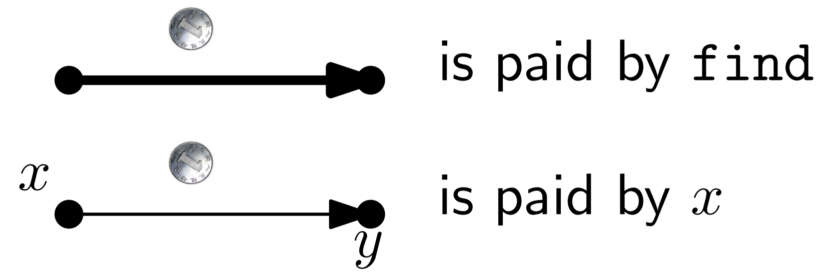
is paid by `find`

is paid by x

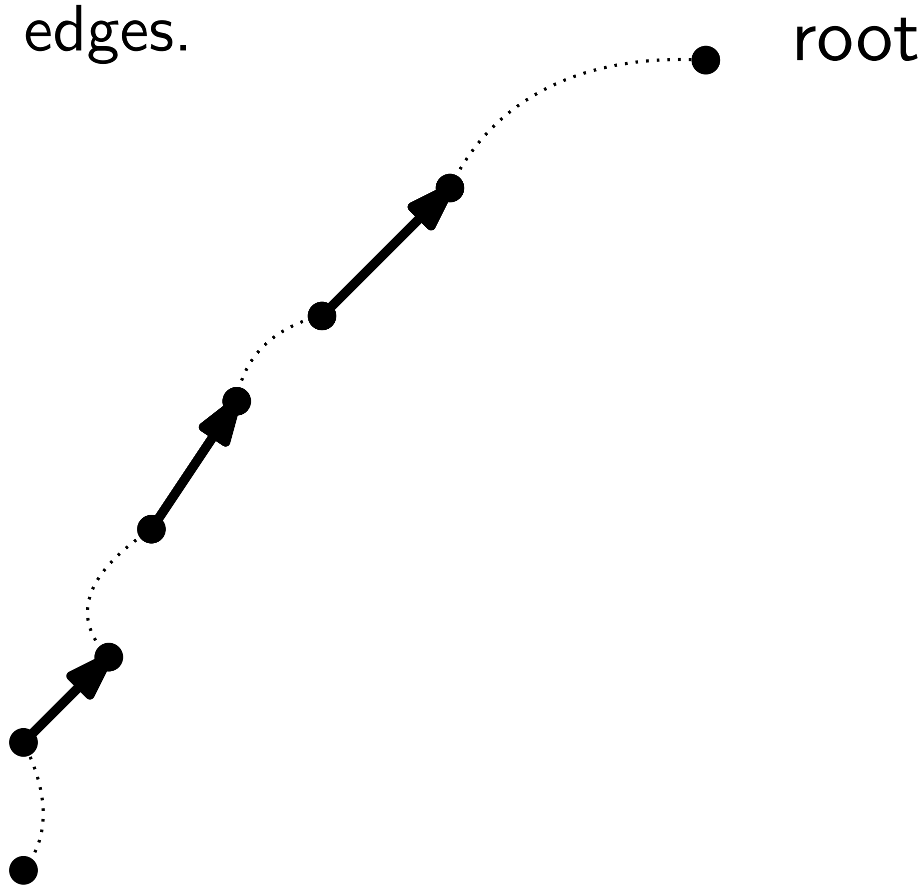
Lemma. Every `find` operation redirects at most $\log\log(n)$ thick edges.

Definition. An edge 

is *thick* if $\mathrm{rank}(y) \geq 2\ \mathrm{rank}(x)$.

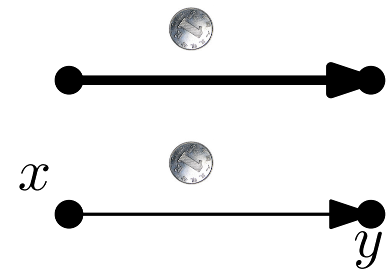


Lemma. Every `find` operation redirects at most $\log\log(n)$ thick edges.



Definition. An edge 

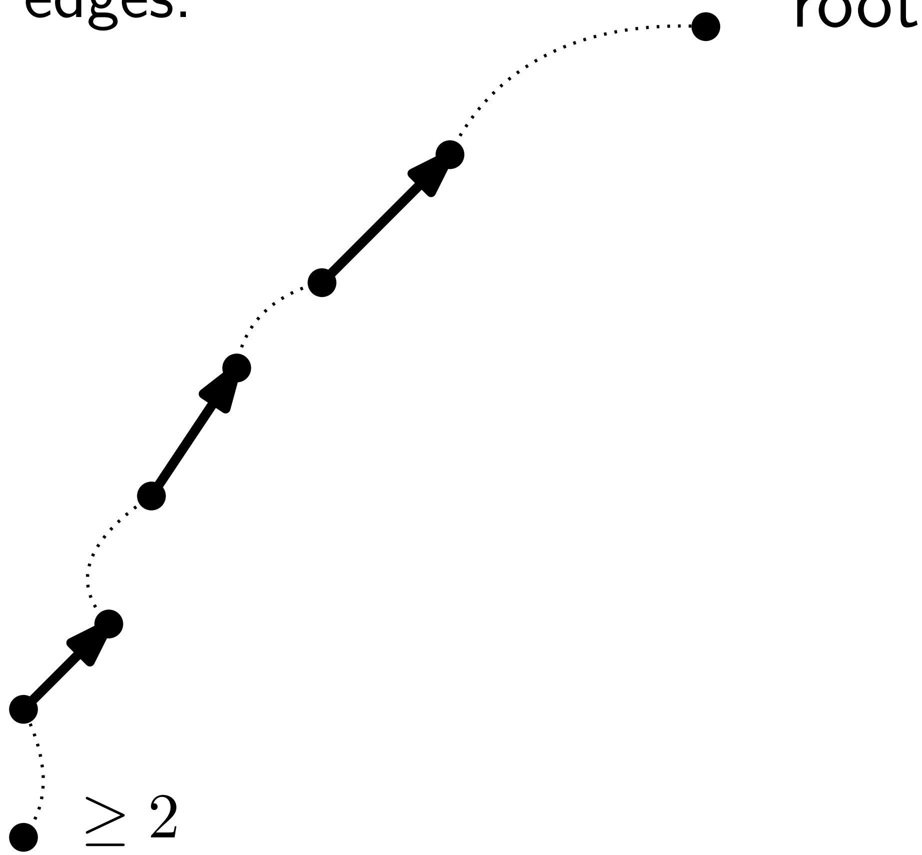
is *thick* if $\mathrm{rank}(y) \geq 2\ \mathrm{rank}(x)$.



is paid by `find`

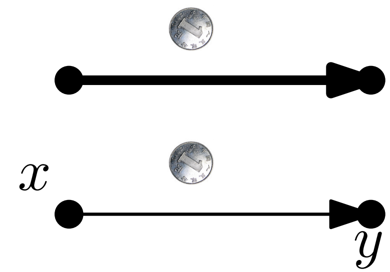
is paid by x

Lemma. Every `find` operation redirects at most $\log\log(n)$ thick edges.



Definition. An edge 

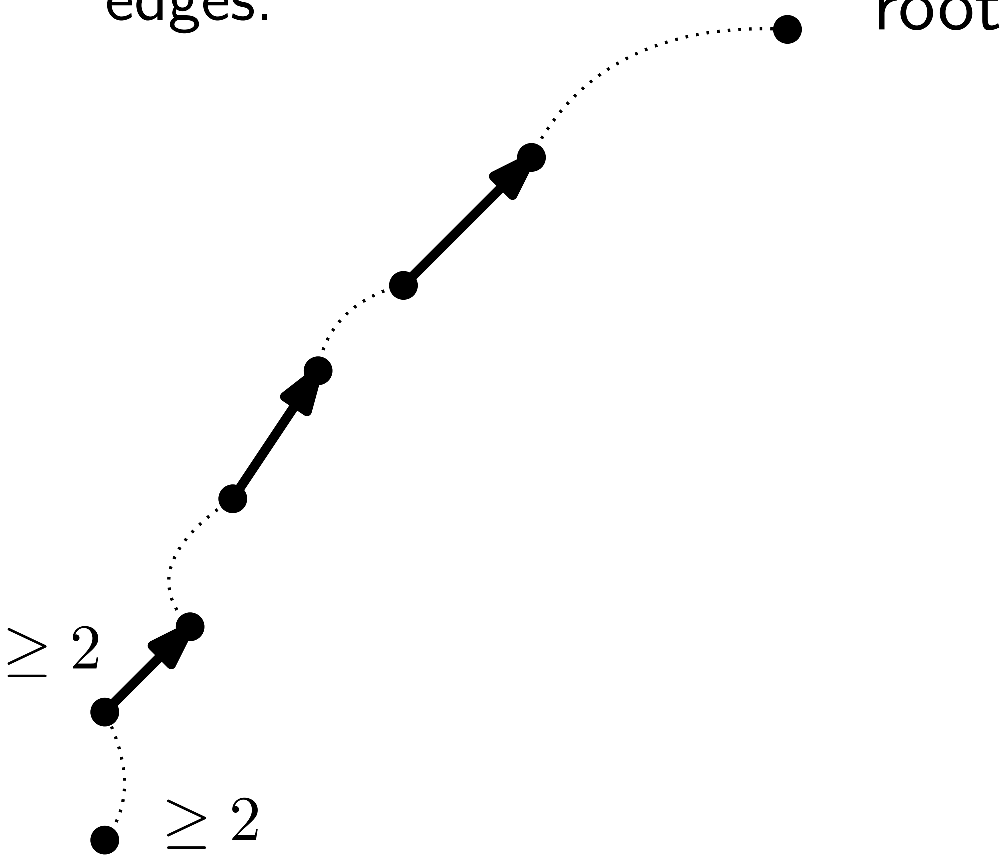
is *thick* if $\mathrm{rank}(y) \geq 2\ \mathrm{rank}(x)$.



is paid by `find`

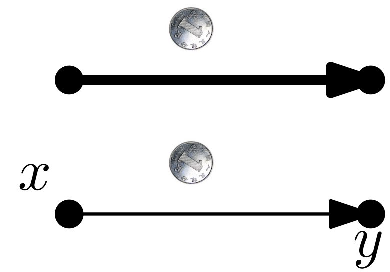
is paid by x

Lemma. Every `find` operation redirects at most $\log\log(n)$ thick edges.



Definition. An edge 

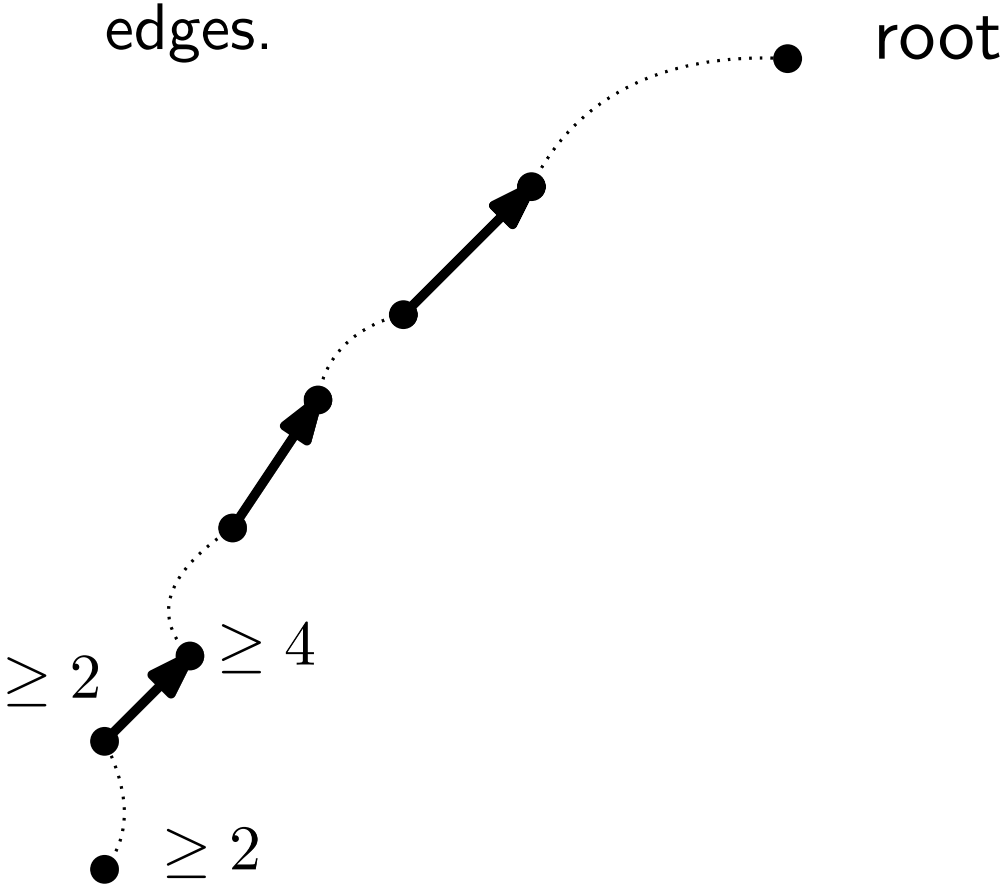
is *thick* if $\mathrm{rank}(y) \geq 2\,\mathrm{rank}(x)$.



is paid by `find`

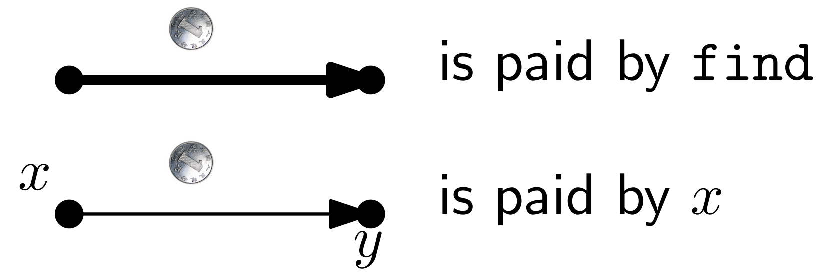
is paid by x

Lemma. Every `find` operation redirects at most $\log\log(n)$ thick edges.

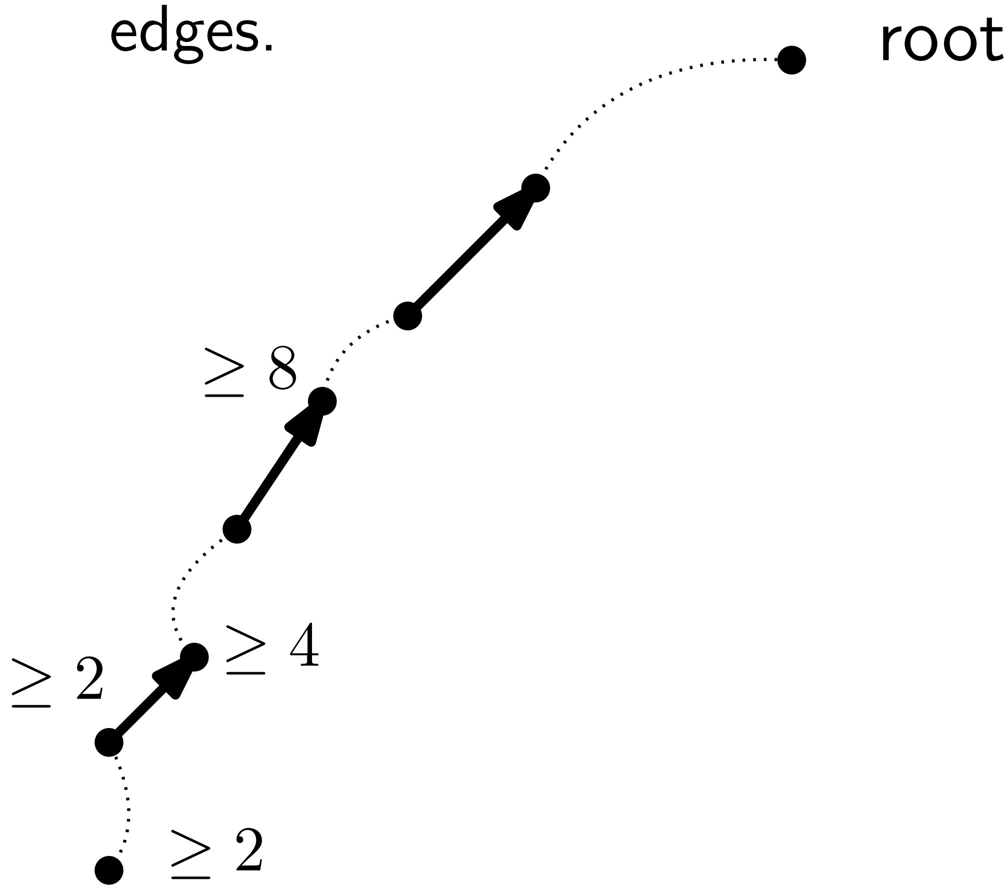


Definition. An edge 

is *thick* if $\mathrm{rank}(y) \geq 2\ \mathrm{rank}(x)$.

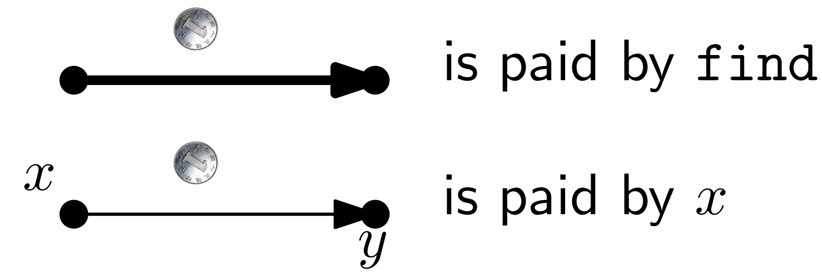


Lemma. Every `find` operation redirects at most $\log \log(n)$ thick edges.

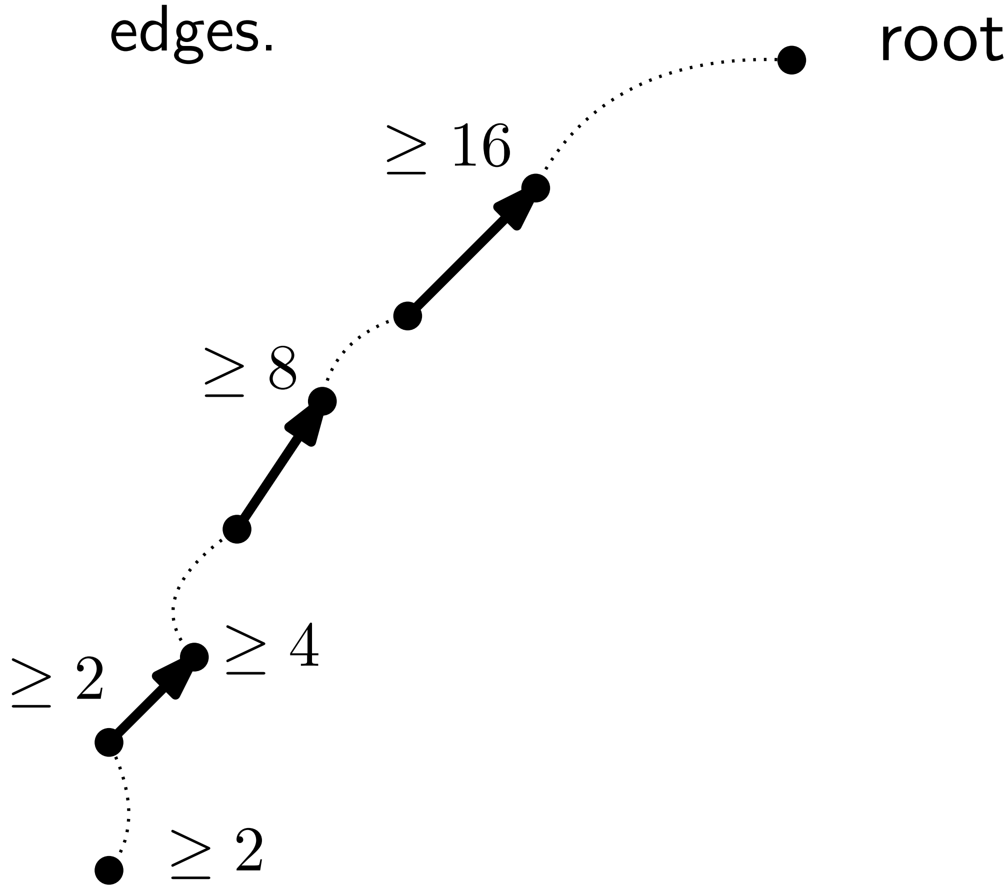


Definition. An edge 

is *thick* if $\mathrm{rank}(y) \geq 2\ \mathrm{rank}(x)$.

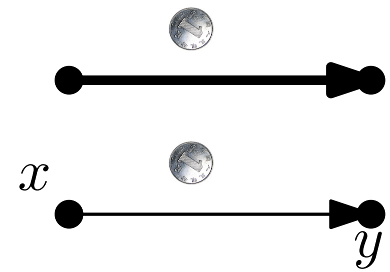


Lemma. Every `find` operation redirects at most $\log\log(n)$ thick edges.



Definition. An edge 

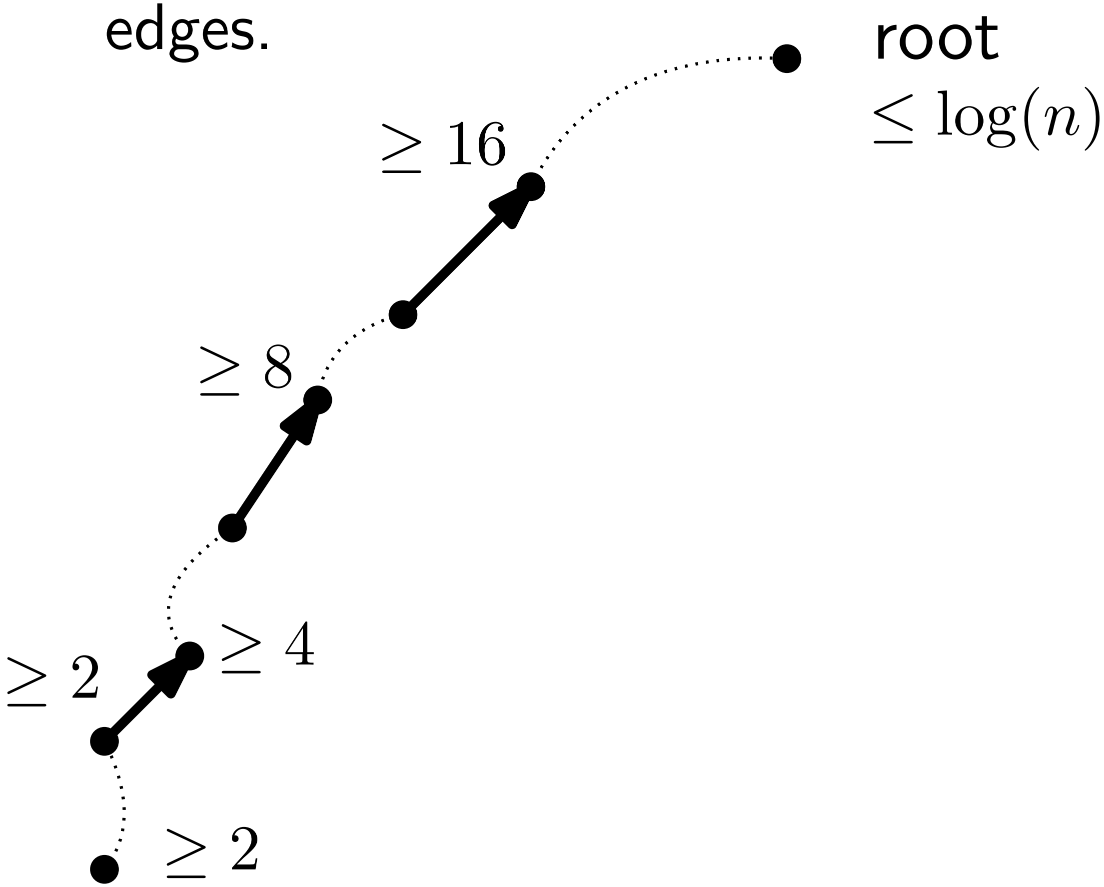
is *thick* if $\mathrm{rank}(y) \geq 2\ \mathrm{rank}(x)$.



is paid by `find`

is paid by x

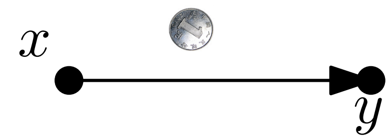
Lemma. Every `find` operation redirects at most $\log\log(n)$ thick edges.



Definition. An edge 

is *thick* if $\mathrm{rank}(y) \geq 2\ \mathrm{rank}(x)$.


is paid by `find`

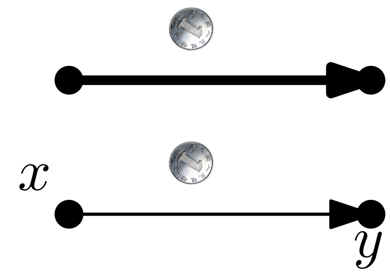


is paid by x

Lemma. Every `find` operation redirects at most $\log\log(n)$ thick edges.

Definition. An edge 

is *thick* if $\mathrm{rank}(y) \geq 2\ \mathrm{rank}(x)$.



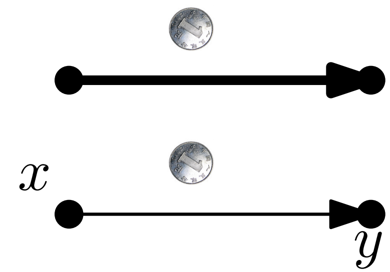
is paid by `find`

is paid by x

Lemma. Every `find` operation redirects at most $\log\log(n)$ thick edges. Thus, every `find` operation has to pay at most $\log\log(n)$ 

Definition. An edge 

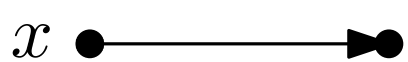
is *thick* if $\mathrm{rank}(y) \geq 2\ \mathrm{rank}(x)$.



is paid by `find`

is paid by x

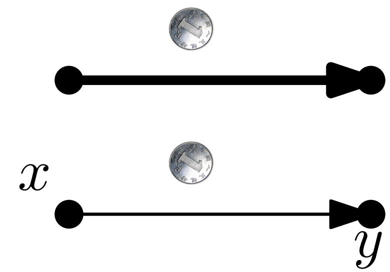
Lemma. Every `find` operation redirects at most $\log\log(n)$ thick edges. Thus, every `find` operation has to pay at most $\log\log(n)$ 

Lemma. Suppose edge 

is thin. It can be redirected at most $\mathrm{r} := \mathrm{rank}(x)$ times before it becomes thick.

Definition. An edge 

is *thick* if $\mathrm{rank}(y) \geq 2\ \mathrm{rank}(x)$.



is paid by `find`

is paid by x

Lemma. Every `find` operation redirects at most $\log\log(n)$ thick edges. Thus, every `find` operation has to pay at most $\log\log(n)$ 

Lemma. Suppose edge 

is thin. It can be redirected at most $\mathrm{r} := \mathrm{rank}(x)$ times before it becomes thick.

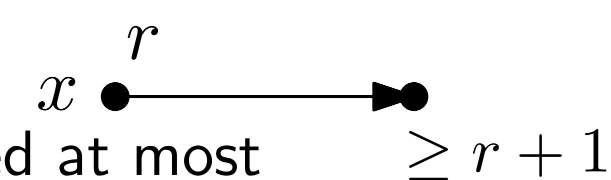
Definition. An edge 

is *thick* if $\mathrm{rank}(y) \geq 2\ \mathrm{rank}(x)$.


is paid by `find`

is paid by x

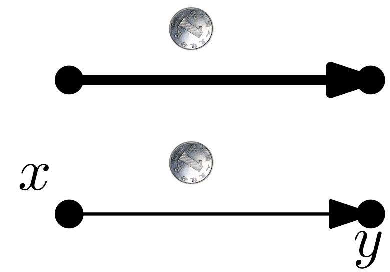
Lemma. Every `find` operation redirects at most $\log\log(n)$ thick edges. Thus, every `find` operation has to pay at most $\log\log(n)$ 

Lemma. Suppose edge 

is thin. It can be redirected at most $\mathrm{r} := \mathrm{rank}(x)$ times before it becomes thick.

Definition. An edge 

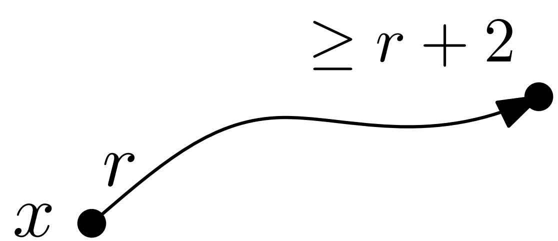
is *thick* if $\mathrm{rank}(y) \geq 2\ \mathrm{rank}(x)$.



is paid by `find`

is paid by x

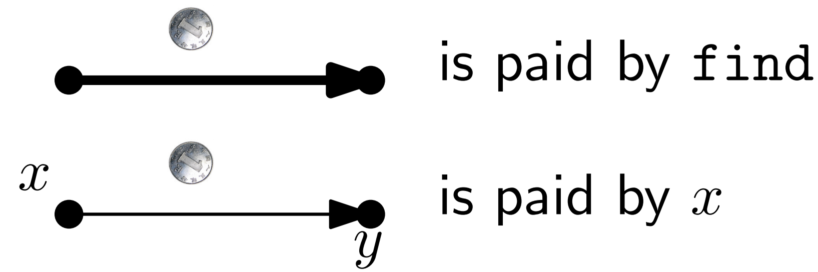
Lemma. Every `find` operation redirects at most $\log\log(n)$ thick edges. Thus, every `find` operation has to pay at most $\log\log(n)$ 



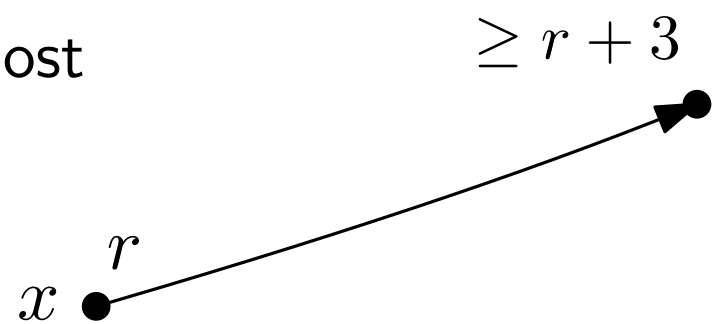
Lemma. Suppose edge is thin. It can be redirected at most $\mathrm{r} := \mathrm{rank}(x)$ times before it becomes thick.

Definition. An edge 

is *thick* if $\mathrm{rank}(y) \geq 2\ \mathrm{rank}(x)$.



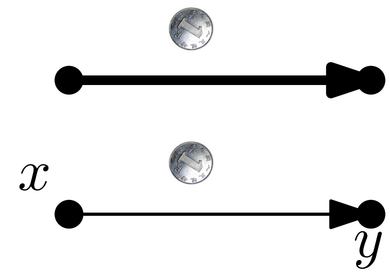
Lemma. Every `find` operation redirects at most $\log\log(n)$ thick edges. Thus, every `find` operation has to pay at most $\log\log(n)$ 

Lemma. Suppose edge 

is thin. It can be redirected at most $\mathrm{r} := \mathrm{rank}(x)$ times before it becomes thick.

Definition. An edge 

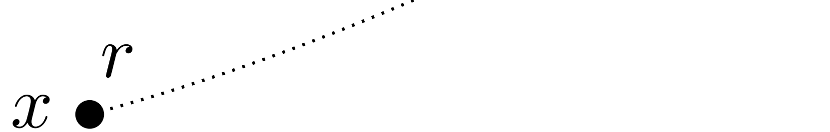
is *thick* if $\mathrm{rank}(y) \geq 2\ \mathrm{rank}(x)$.



is paid by `find`

is paid by x

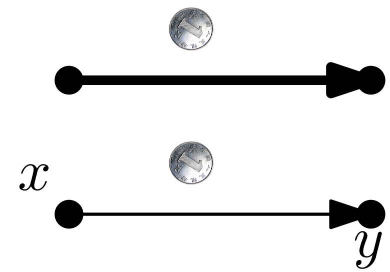
Lemma. Every `find` operation redirects at most $\log\log(n)$ thick edges. Thus, every `find` operation has to pay at most $\log\log(n)$

Lemma. Suppose edge 

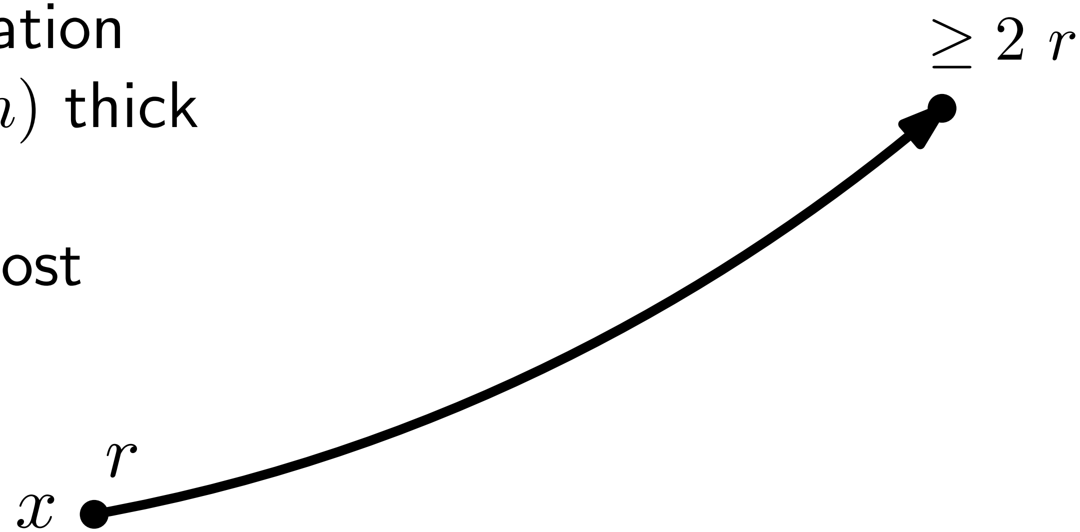
is thin. It can be redirected at most $\mathrm{r} := \mathrm{rank}(x)$ times before it becomes thick.

Definition. An edge 

is *thick* if $\mathrm{rank}(y) \geq 2\ \mathrm{rank}(x)$.



Lemma. Every `find` operation redirects at most $\log\log(n)$ thick edges. Thus, every `find` operation has to pay at most $\log\log(n)$ 

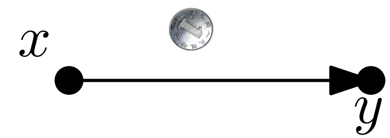


Lemma. Suppose edge is thin. It can be redirected at most $\mathrm{r} := \mathrm{rank}(x)$ times before it becomes thick.

Definition. An edge 

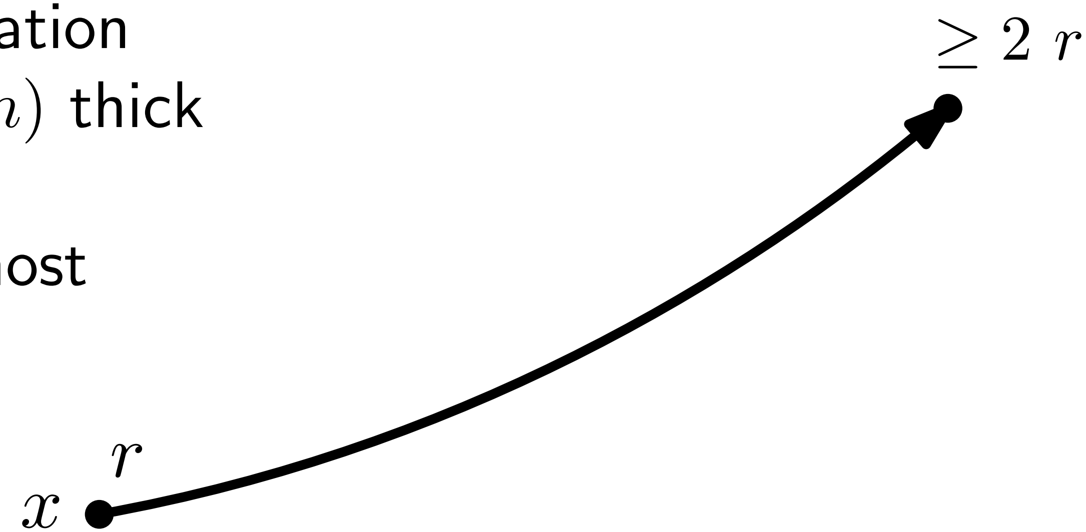
is *thick* if $\mathrm{rank}(y) \geq 2\ \mathrm{rank}(x)$.

 is paid by `find`



is paid by x

Lemma. Every `find` operation redirects at most $\log\log(n)$ thick edges. Thus, every `find` operation has to pay at most $\log\log(n)$ 



Lemma. Suppose edge is thin. It can be redirected at most $\mathrm{r} := \mathrm{rank}(x)$ times before it becomes thick. Thus, x has to pay at most $\mathrm{rank}(x)$ 

Lemma. Every `find` operation redirects at most $\log\log(n)$ thick edges. Thus, every `find` operation has to pay at most $\log\log(n)$ 

Lemma. Suppose edge is thin. It can be redirected at most $\mathrm{r} := \mathrm{rank}(x)$ times before it becomes thick. Thus, x has to pay at most $\mathrm{rank}(x)$ 

m find operations: $\leq m \log\log(n)$

Lemma. Every `find` operation redirects at most $\log\log(n)$ thick edges. Thus, every `find` operation has to pay at most $\log\log(n)$

Lemma. Suppose edge is thin. It can be redirected at most $\mathrm{r} := \mathrm{rank}(x)$ times before it becomes thick. Thus, x has to pay at most $\mathrm{rank}(x)$

m find operations: $\leq m \log\log(n)$

Lemma. Every `find` operation redirects at most $\log\log(n)$ thick edges. Thus, every `find` operation has to pay at most $\log\log(n)$

paid by elements

Lemma. Suppose edge is thin. It can be redirected at most $\mathrm{r} := \mathrm{rank}(x)$ times before it becomes thick. Thus, x has to pay at most $\mathrm{rank}(x)$

m find operations: $\leq m \log\log(n)$

Lemma. Every `find` operation redirects at most $\log\log(n)$ thick edges. Thus, every `find` operation has to pay at most $\log\log(n)$

\# paid by elements

$\leq \sum_x \mathrm{rank}(x)$

Lemma. Suppose edge is thin. It can be redirected at most $\mathrm{r} := \mathrm{rank}(x)$ times before it becomes thick. Thus, x has to pay at most $\mathrm{rank}(x)$

m find operations: $\leq m \log\log(n)$

Lemma. Every `find` operation redirects at most $\log\log(n)$ thick edges. Thus, every `find` operation has to pay at most $\log\log(n)$

Lemma. Suppose edge is thin. It can be redirected at most $\mathrm{r} := \mathrm{rank}(x)$ times before it becomes thick. Thus, x has to pay at most $\mathrm{rank}(x)$

paid by elements

$\leq \sum_x \mathrm{rank}(x)$

$= \sum_r r \cdot |\{\mathrm{rank}{-}r{-}\mathrm{elements}\}|$

m find operations: $\leq m \log\log(n)$

Lemma. Every `find` operation redirects at most $\log\log(n)$ thick edges. Thus, every `find` operation has to pay at most $\log\log(n)$

Lemma. Suppose edge is thin. It can be redirected at most $\mathrm{r} := \mathrm{rank}(x)$ times before it becomes thick. Thus, x has to pay at most $\mathrm{rank}(x)$

paid by elements

$$\leq \sum_x \mathrm{rank}(x)$$

$$= \sum_r r \cdot |\{\mathrm{rank}{-}r{-}\mathrm{elements}\}|$$

$$\leq \sum_r r \cdot \tfrac{n}{2^{r-2}}$$

m find operations: $\leq m \log\log(n)$

Lemma. Every `find` operation redirects at most $\log\log(n)$ thick edges. Thus, every `find` operation has to pay at most $\log\log(n)$

Lemma. Suppose edge is thin. It can be redirected at most $\mathrm{r} := \mathrm{rank}(x)$ times before it becomes thick. Thus, x has to pay at most $\mathrm{rank}(x)$

paid by elements

$\leq \sum_x \mathrm{rank}(x)$

$= \sum_r r \cdot |\{\mathrm{rank}{-}r{-}\mathrm{elements}\}|$

$\leq \sum_r r \cdot \frac{n}{2^{r-2}} = 6\,n$

Union-by-Rank: $O(n + m \log n)$

Union-by-Rank: $O(n + m \log n)$

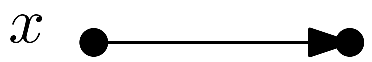
Union-by-Rank with path compression: $O(n + m \log \log n)$

Union-by-Rank: $O(n + m \log n)$

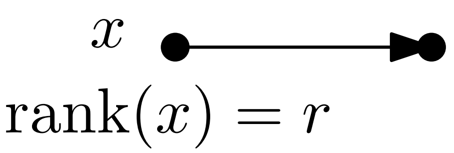
Union-by-Rank with path compression: $O(n + m \log \log n)$

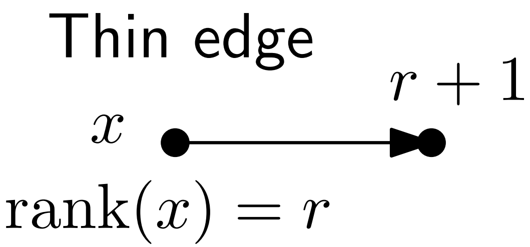
Even Better Analysis

Thin edge

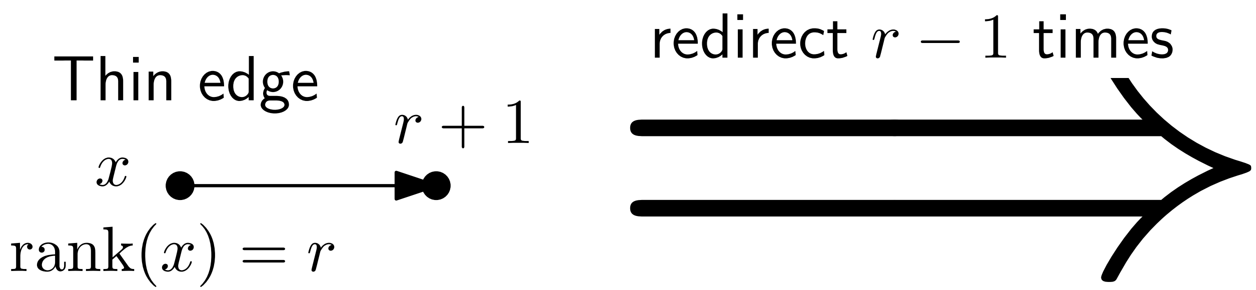


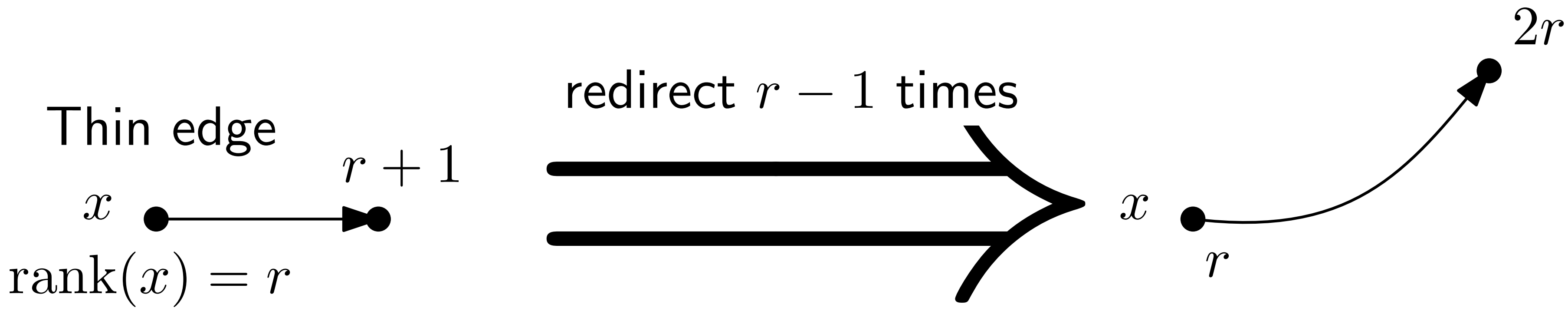
Thin edge



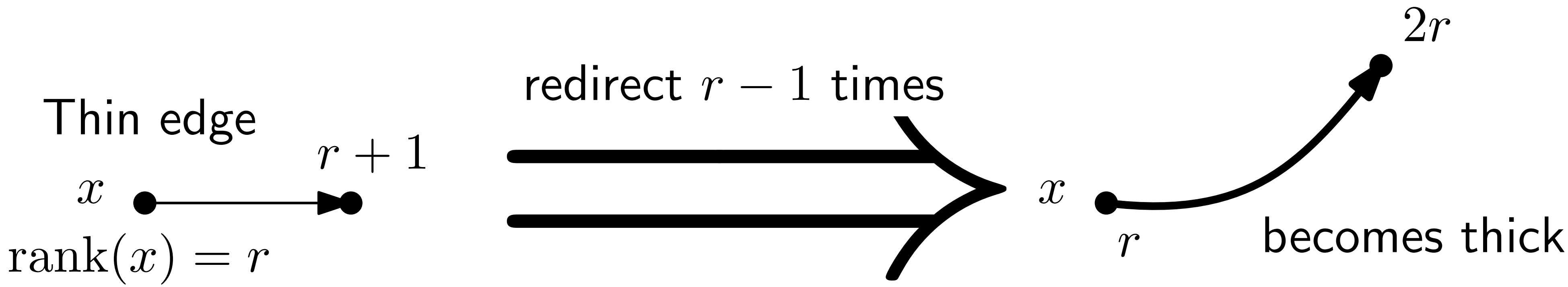


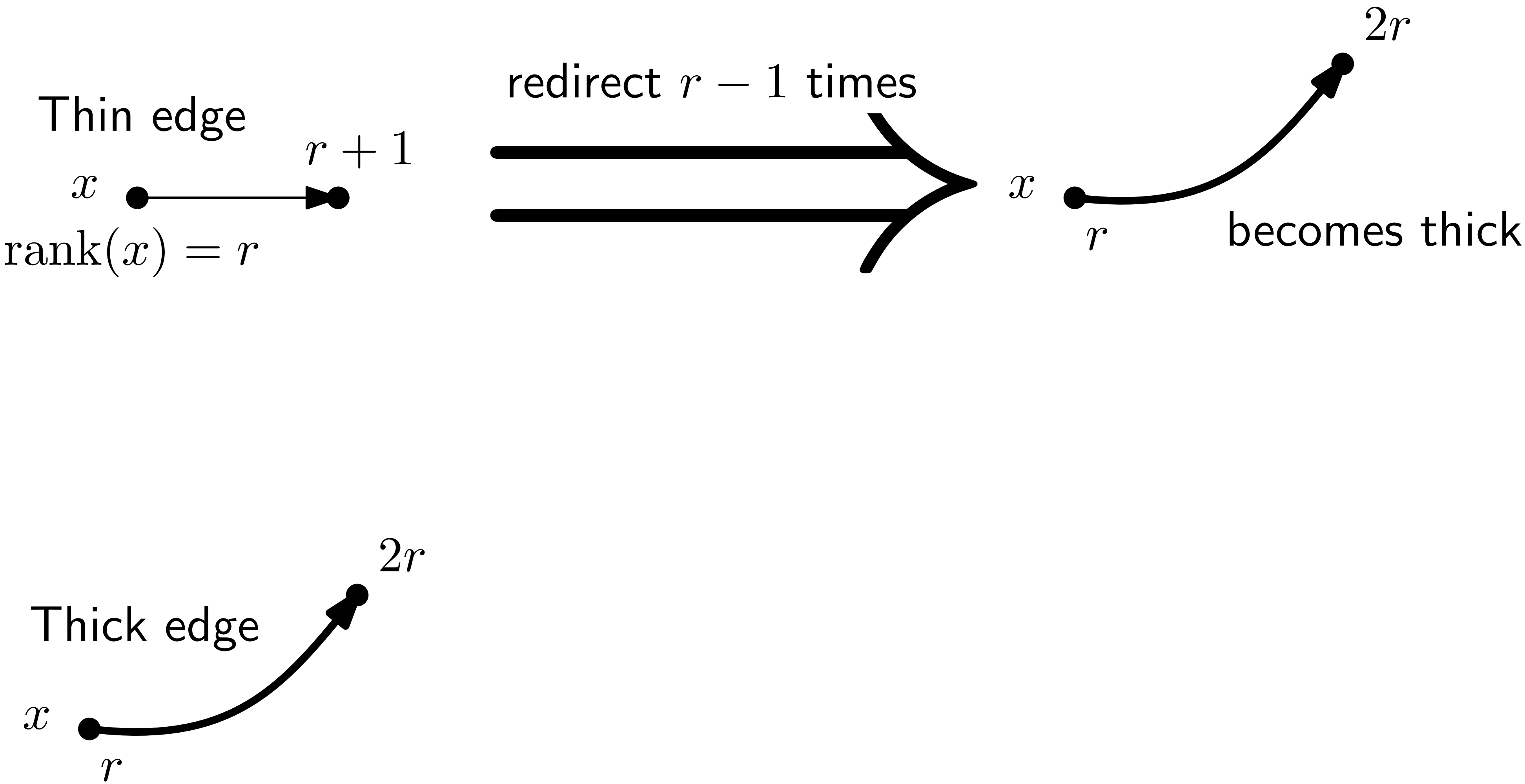
Thin edge
$r + 1$
x
$\mathrm{rank}(x) = r$

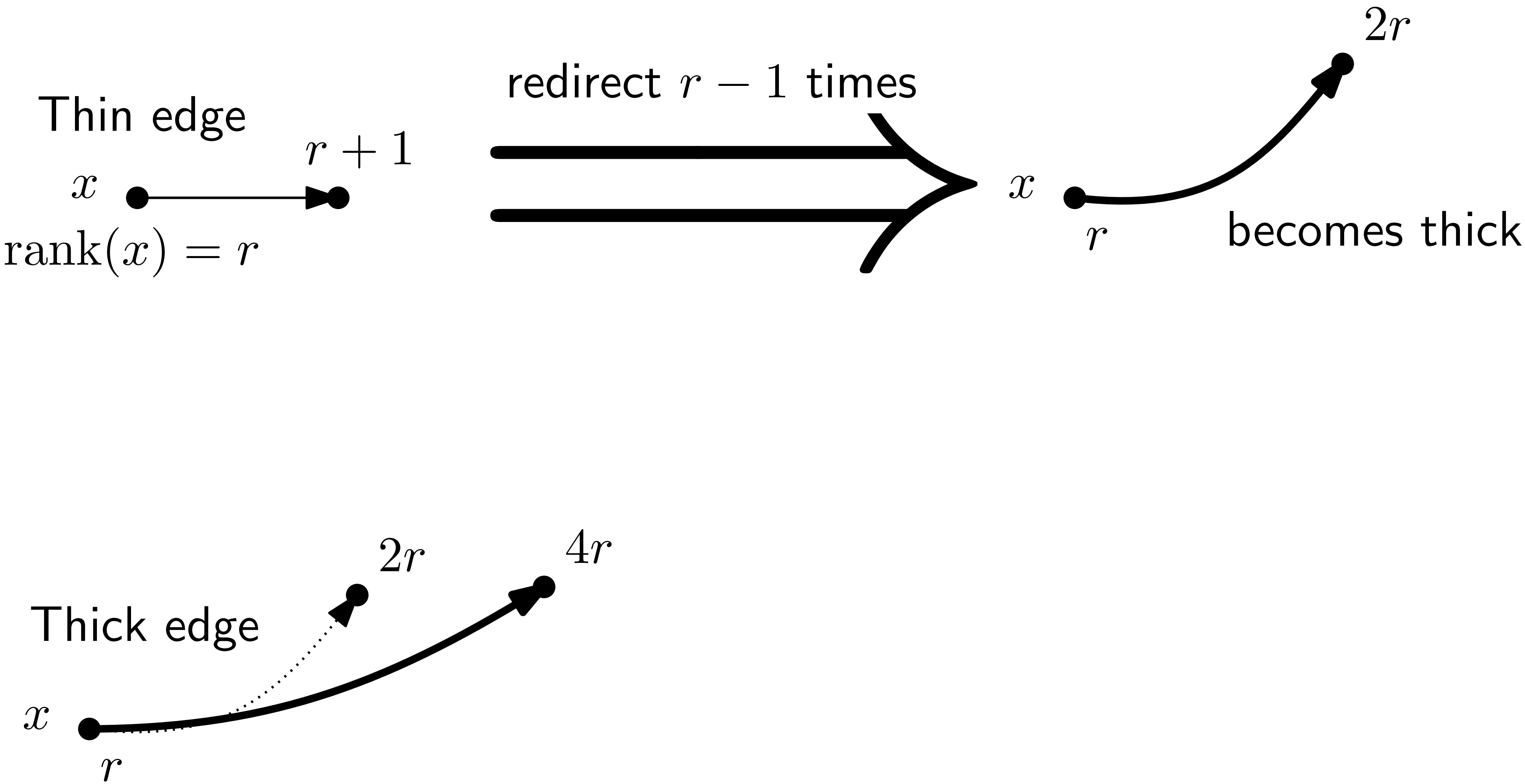

Thin edge
x
$r+1$
$\mathrm{rank}(x) = r$
redirect $r-1$ times



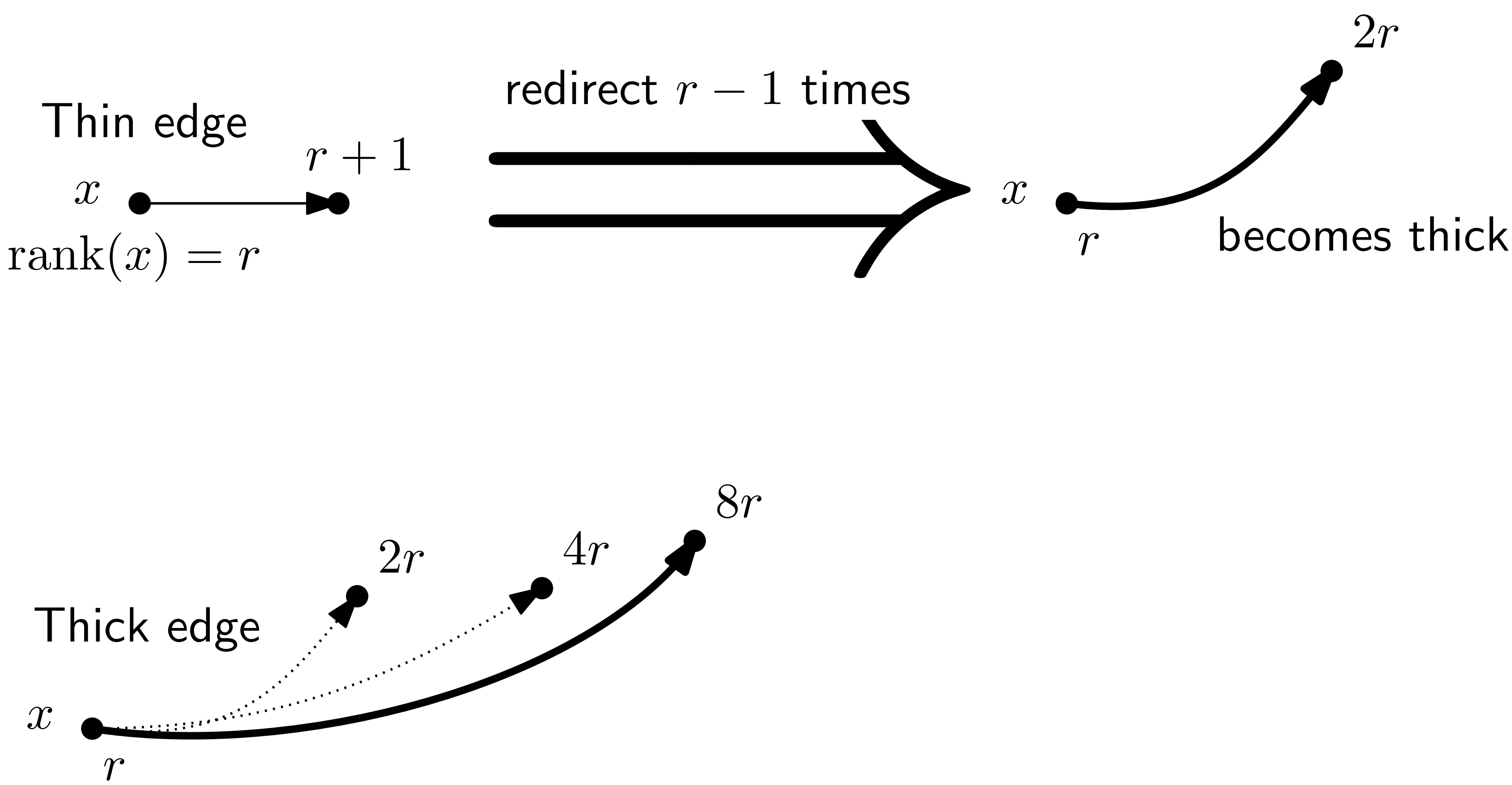
Thin edge
x
$r+1$
$\mathrm{rank}(x) = r$
redirect $r-1$ times
x
r
$2r$

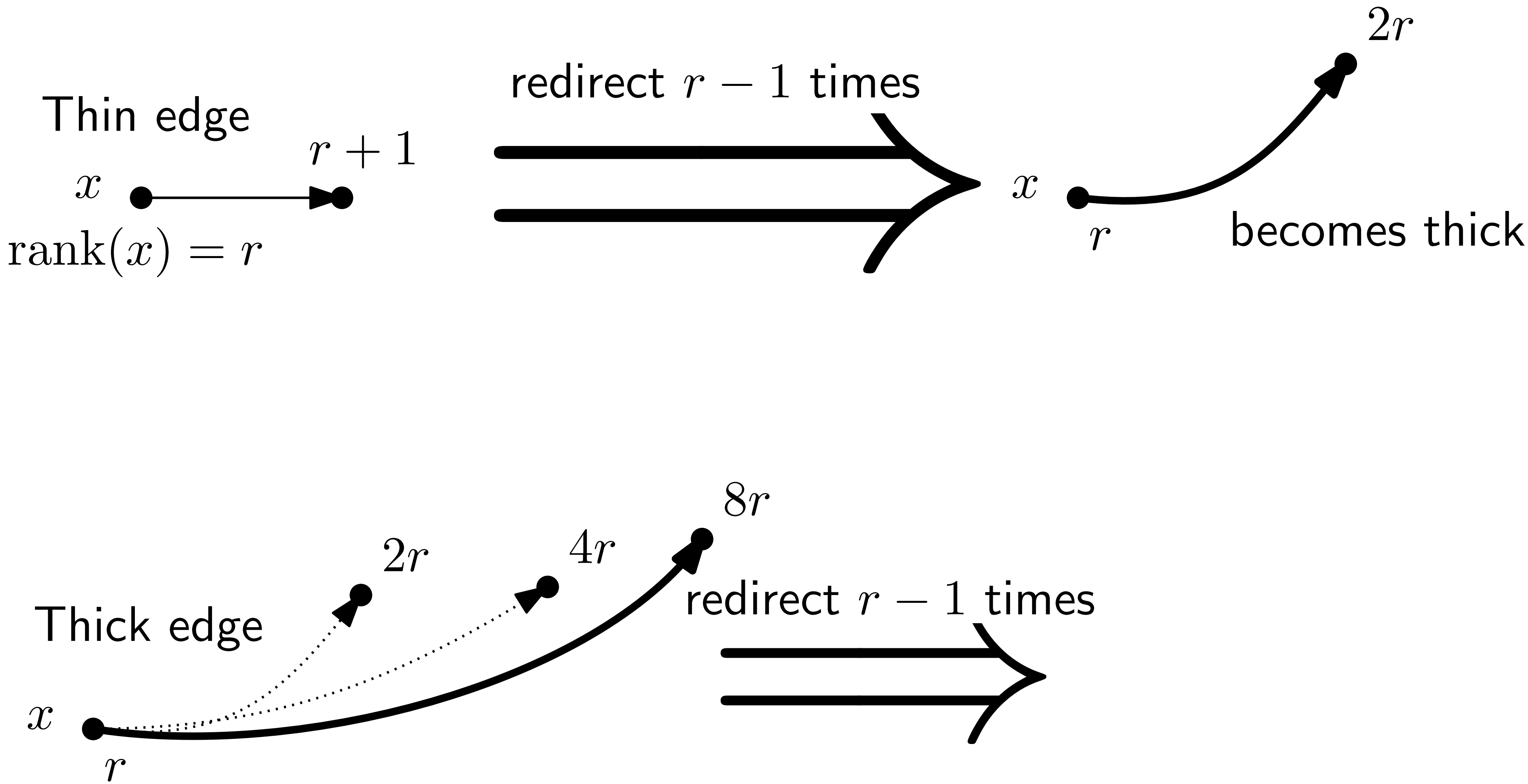

Thin edge
$r+1$
x
$\mathrm{rank}(x) = r$
redirect $r-1$ times
$2r$
x
r
becomes thick

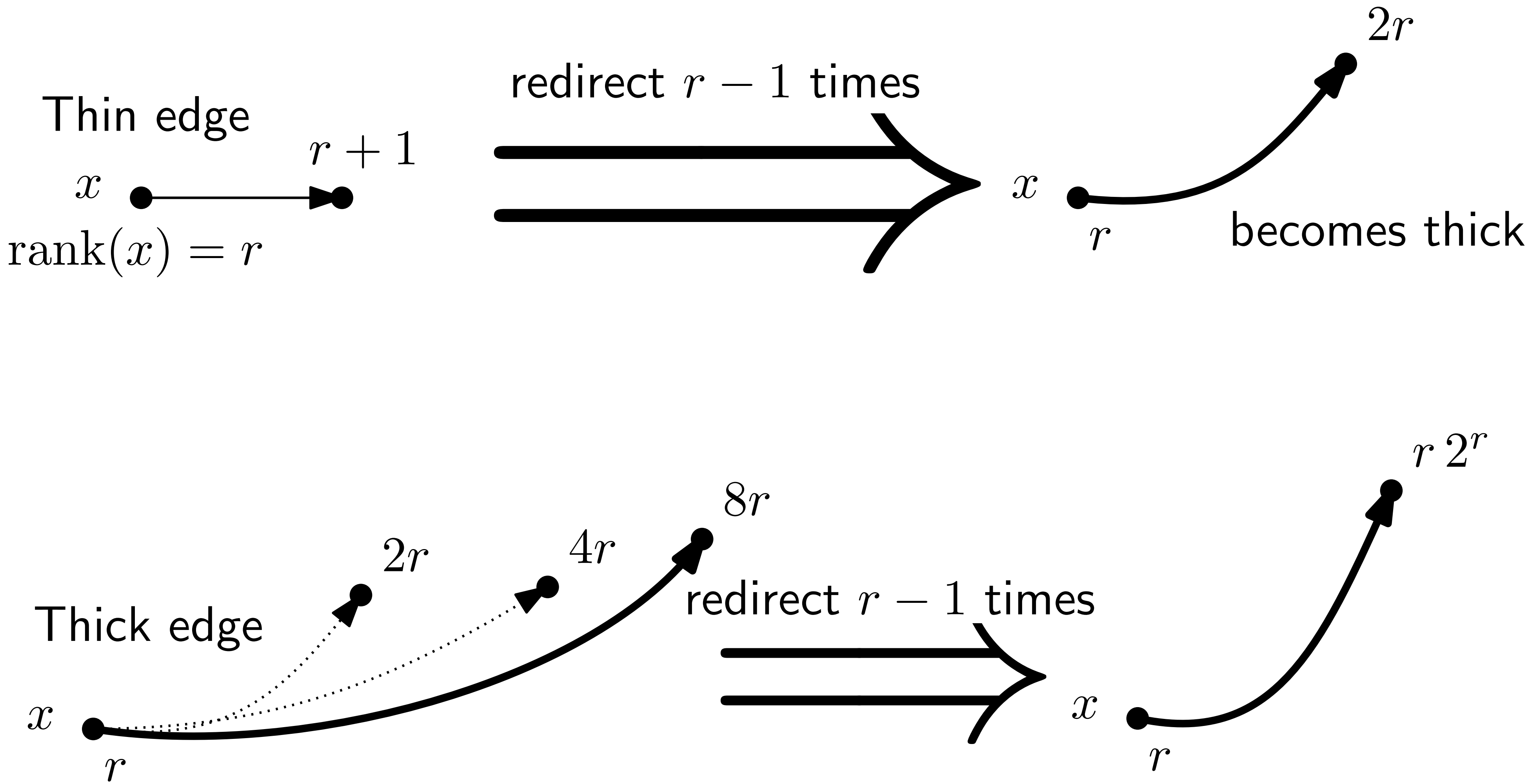

Thin edge
x
$r+1$
$\mathrm{rank}(x) = r$
redirect $r-1$ times
x
$2r$
r
becomes thick
Thick edge
x
$2r$
r



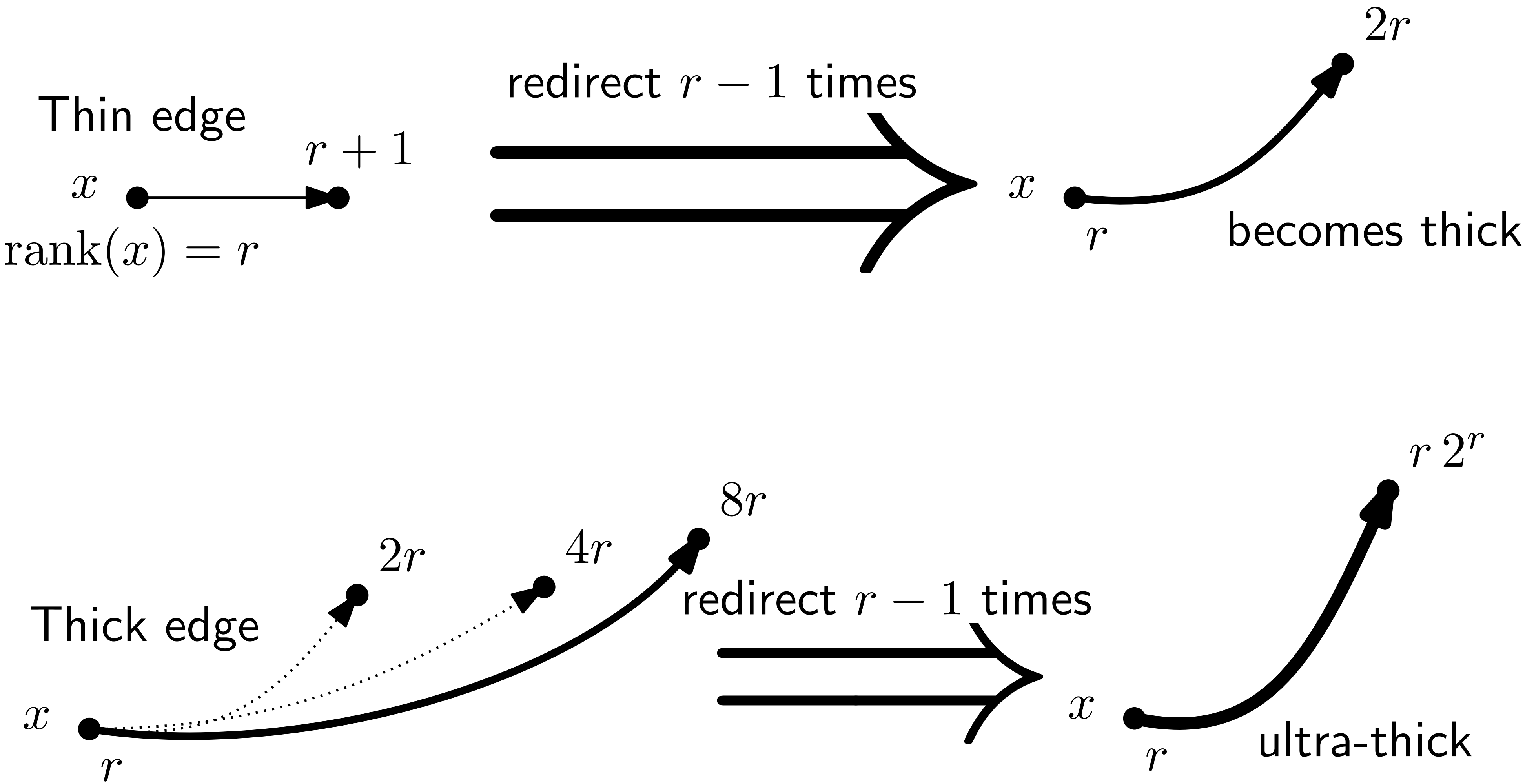
Thin edge
x
$r+1$
$\mathrm{rank}(x) = r$
redirect $r-1$ times
x
r
$2r$
becomes thick
Thick edge
x
r
$2r$
$4r$


Thin edge
x
$r+1$
$\mathrm{rank}(x) = r$
redirect $r-1$ times
x
$2r$
r
becomes thick
Thick edge
x
r
$2r$
$4r$
$8r$

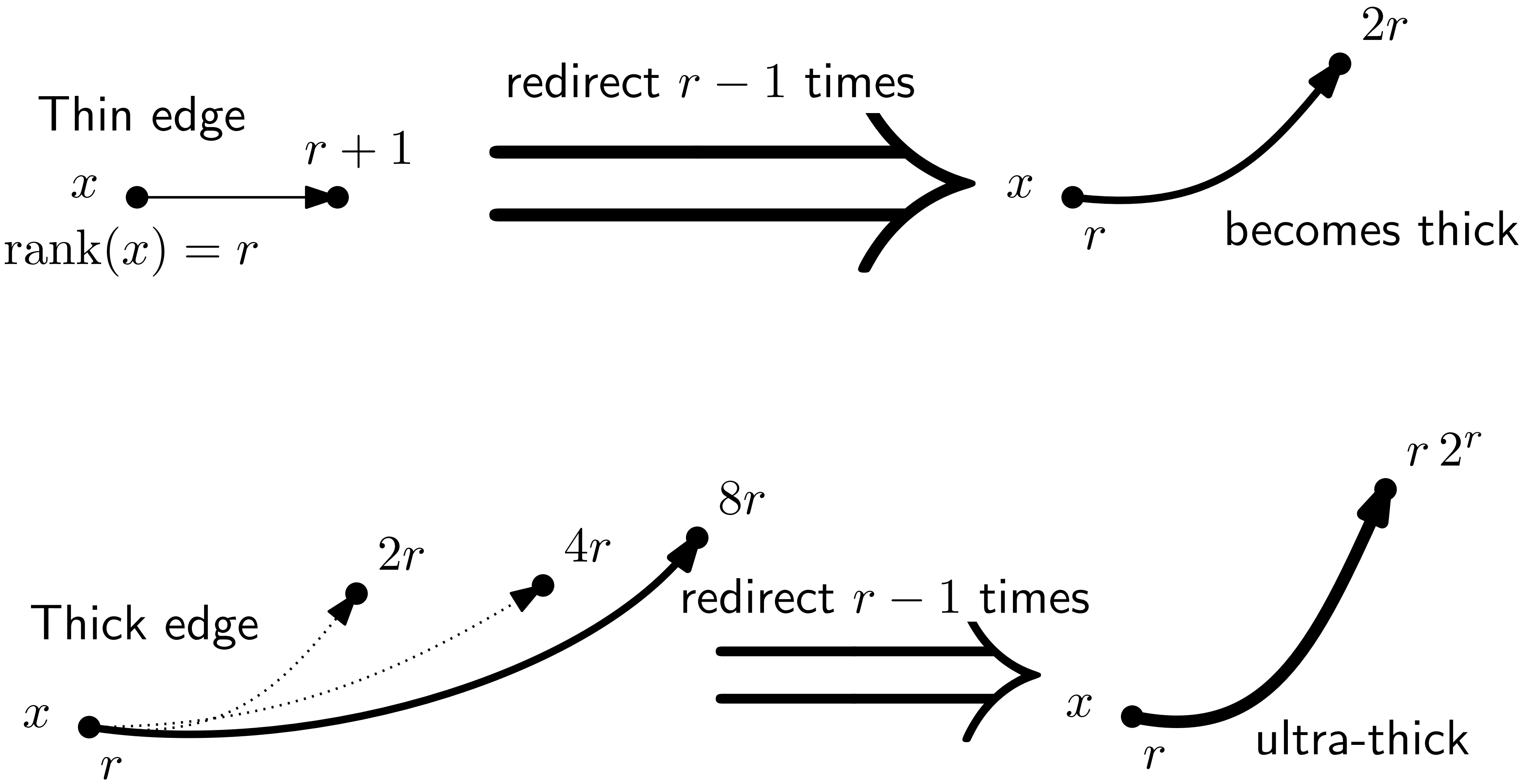

Thin edge
x
$r+1$
$\mathrm{rank}(x) = r$
redirect $r-1$ times
x
r
$2r$
becomes thick
Thick edge
x
r
$2r$
$4r$
$8r$
redirect $r-1$ times



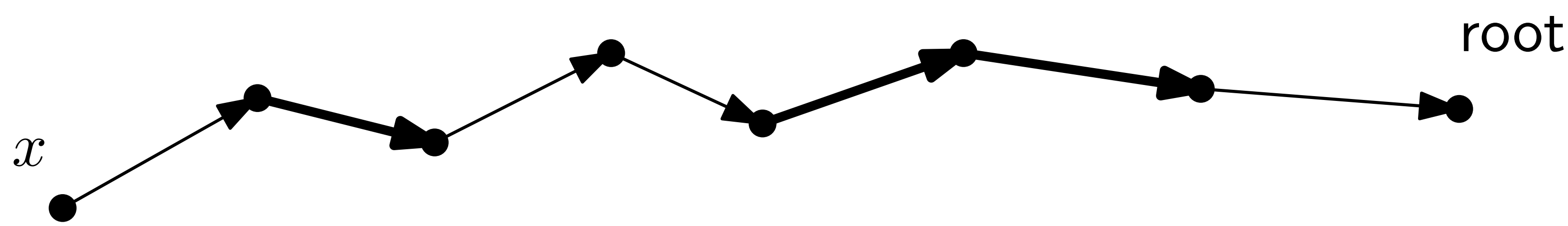
Thin edge
x
$r+1$
$\mathrm{rank}(x) = r$
redirect $r-1$ times
x
r
$2r$
becomes thick
Thick edge
x
r
$2r$
$4r$
$8r$
redirect $r-1$ times
x
r
$r\,2^r$

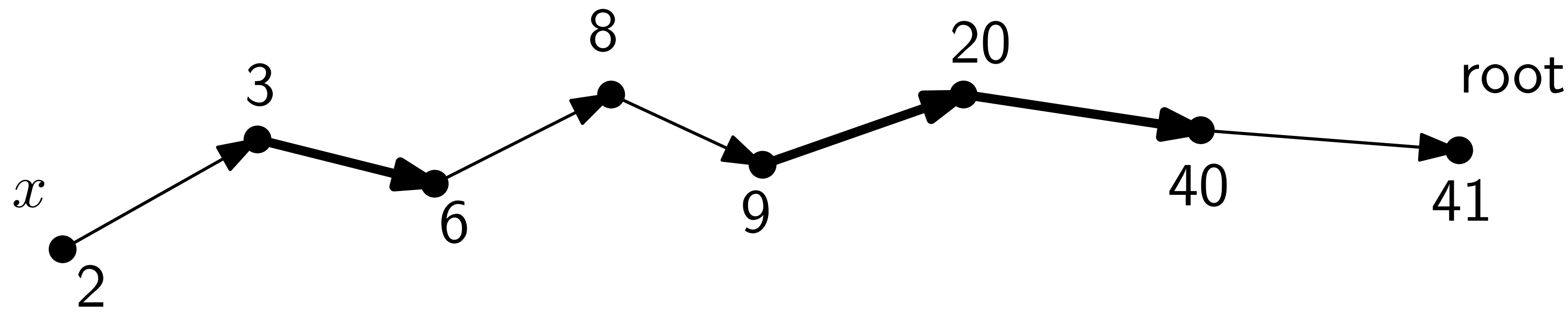


Thin edge
x
$r+1$
$\mathrm{rank}(x) = r$
redirect $r-1$ times
x
r
$2r$
becomes thick
Thick edge
x
r
$2r$
$4r$
$8r$
redirect $r-1$ times
x
r
$r\,2^r$
ultra-thick

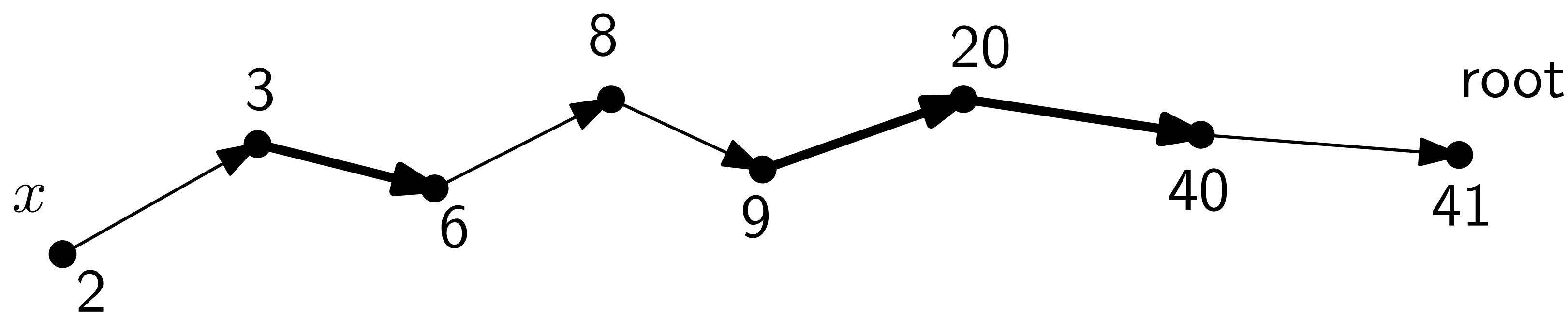


Spot the error!

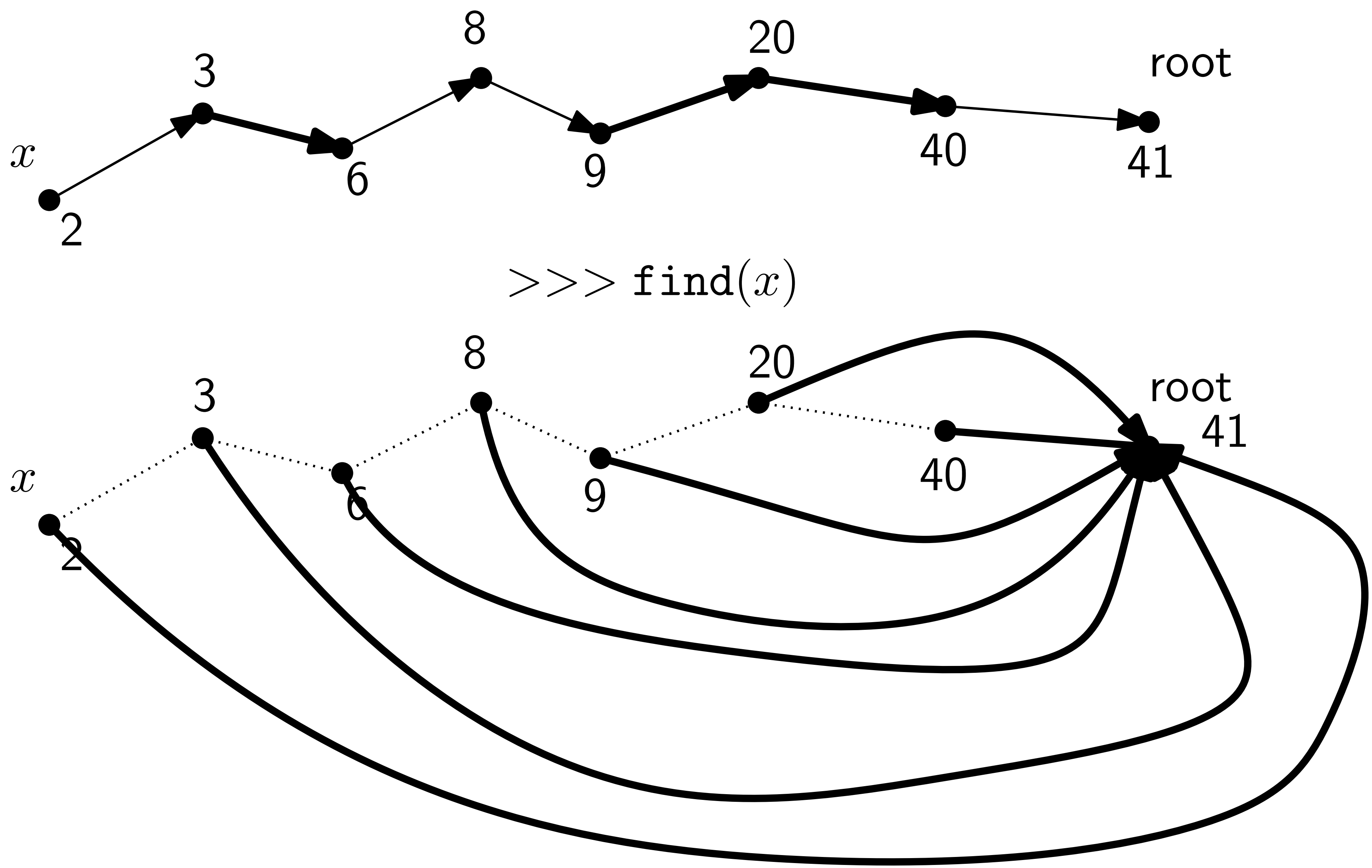

x
root



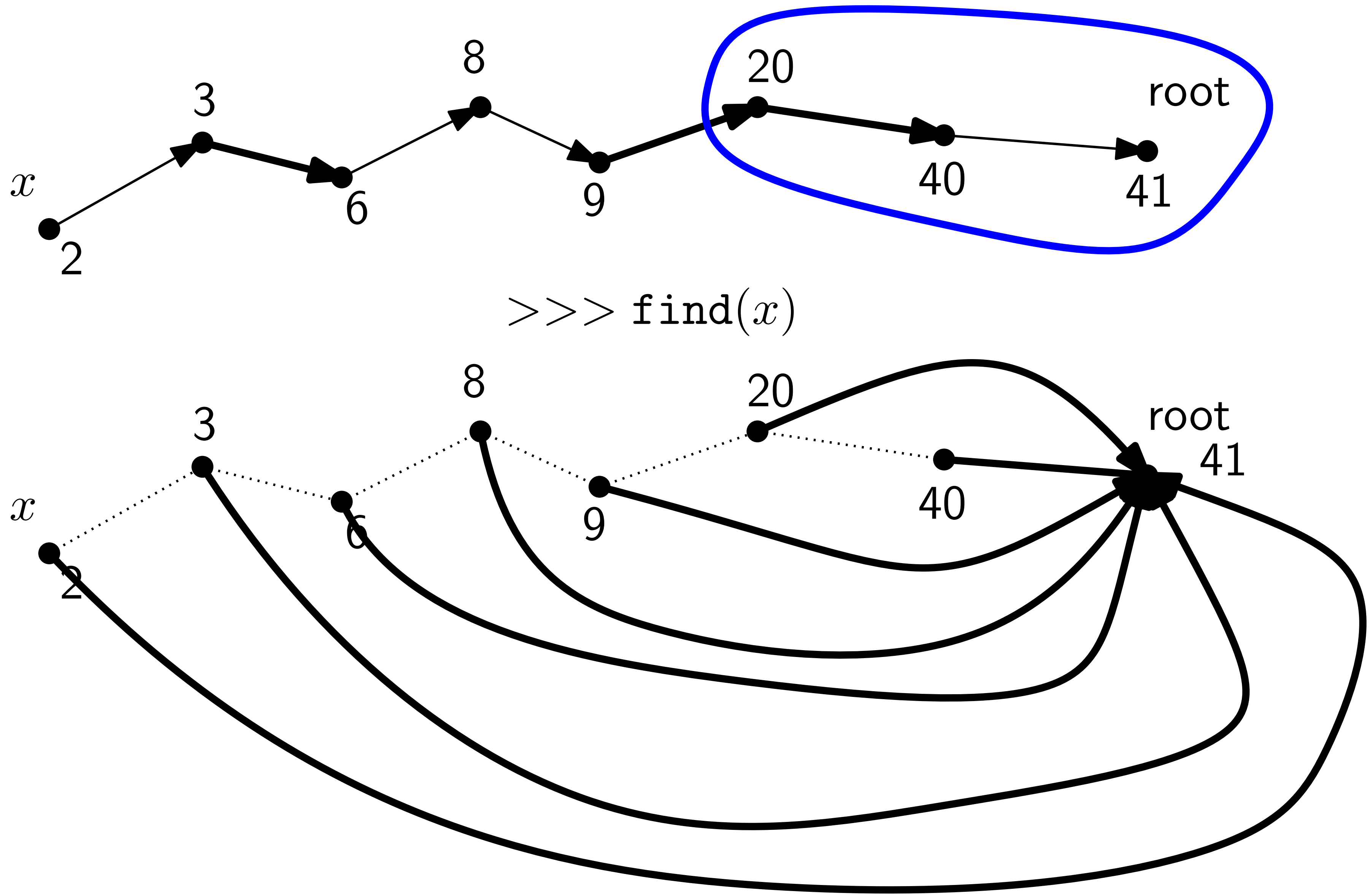
8
20
3
root
x
40
6
9
41
2

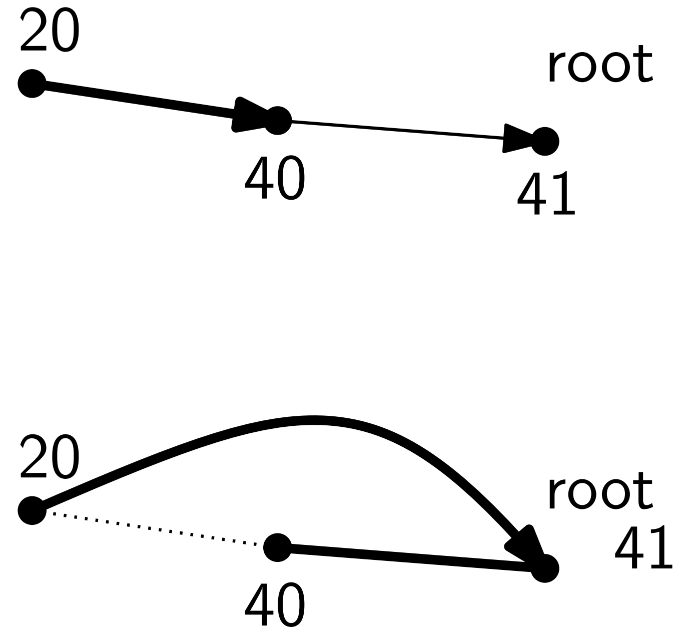


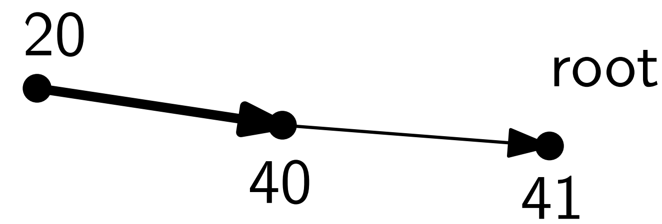
$$>>> \texttt{find}(x)$$

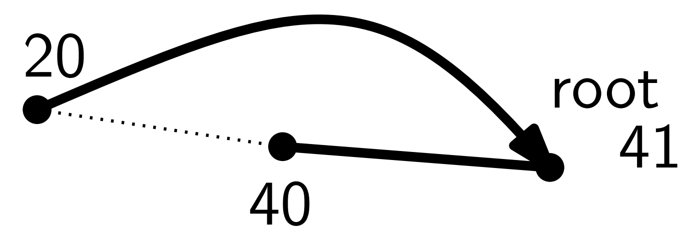


root
8
20
3
x
2
6
9
40
41
$>>> \texttt{find}(x)$
8
20
root
3
41
x
6
9
40
2

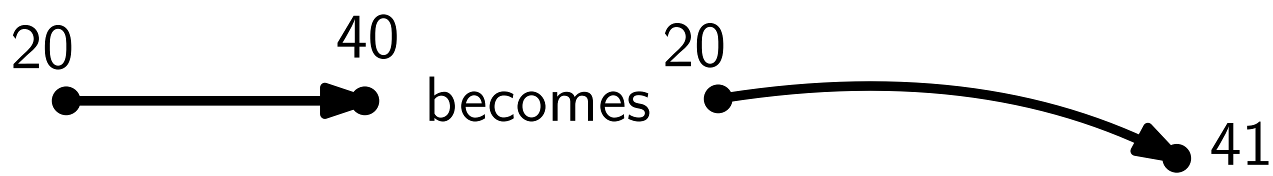

x
2
3
6
8
9
20
40
41
root
>>> find(x)
x
2
3
6
8
9
20
40
41
root

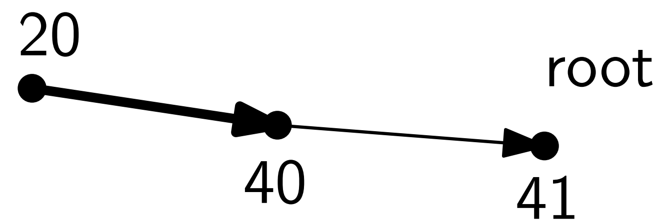

20
root
40
41
20
root
41
40

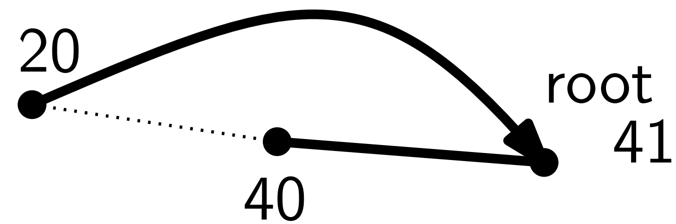




Problem:



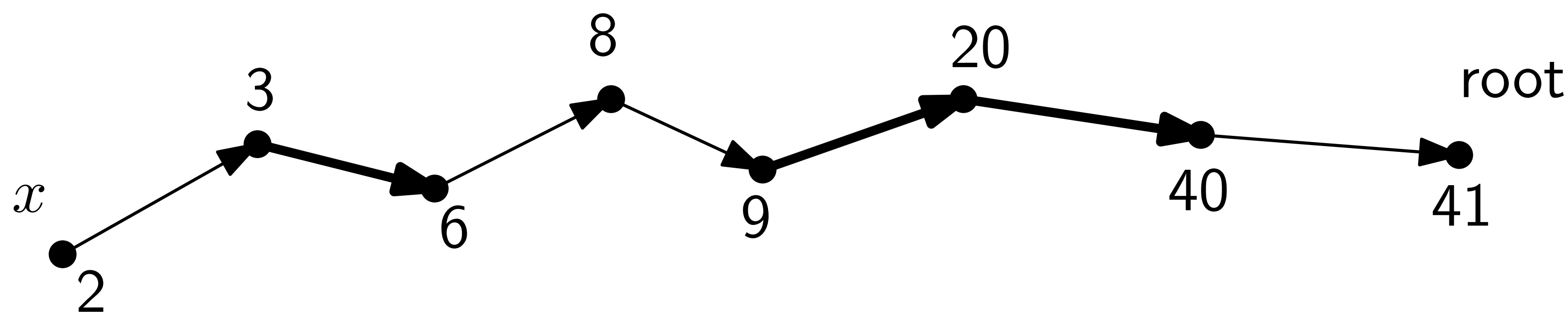




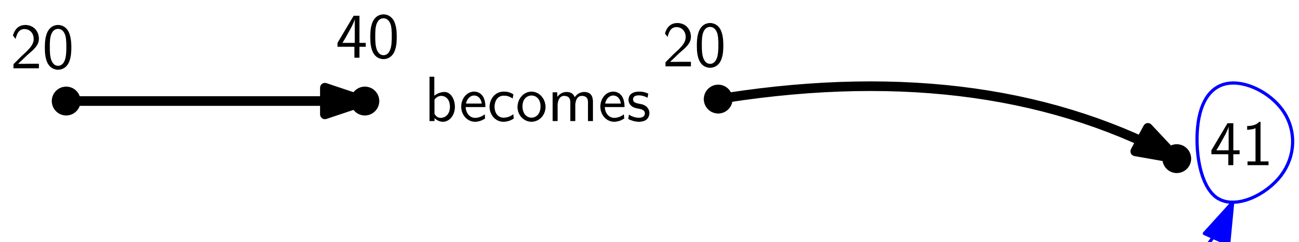
Problem:

becomes

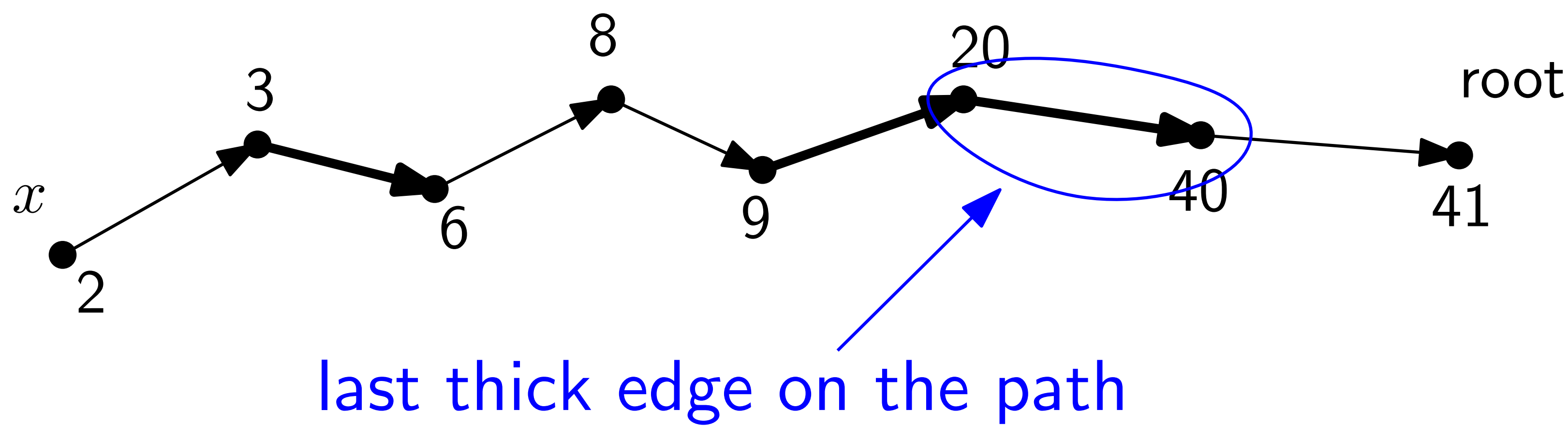
But I claimed this would be at least 80



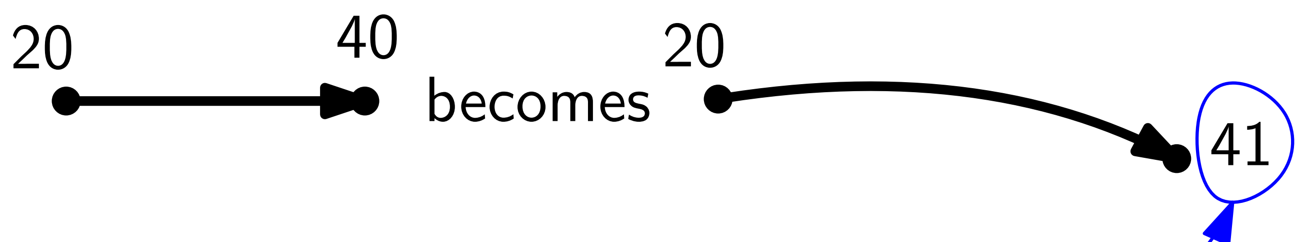
Problem:



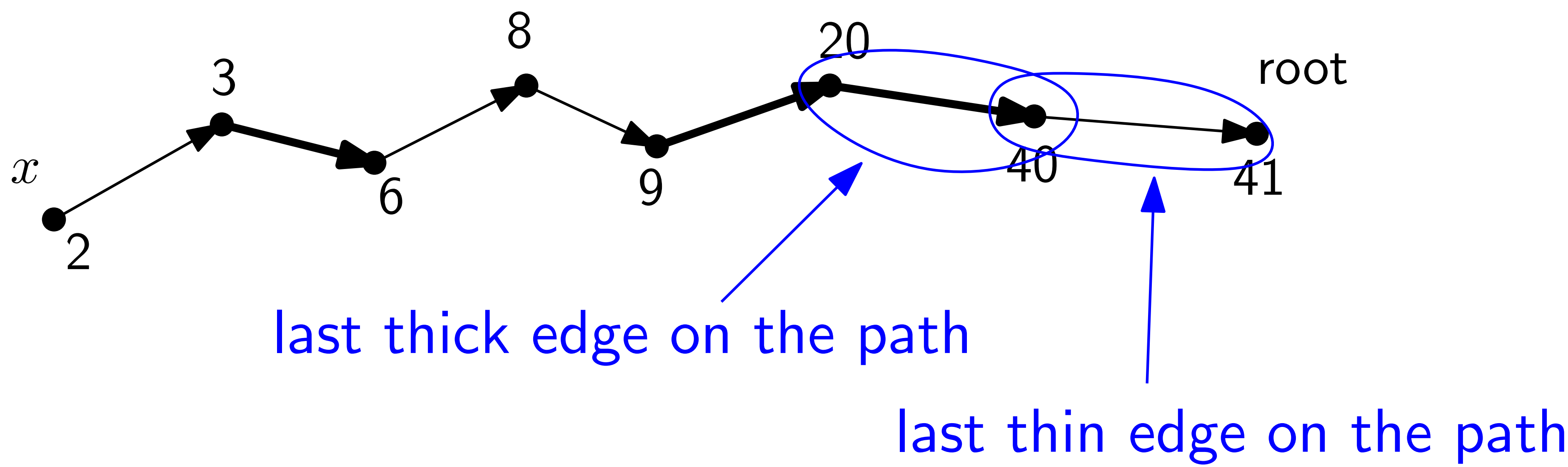
But I claimed this would be at least 80



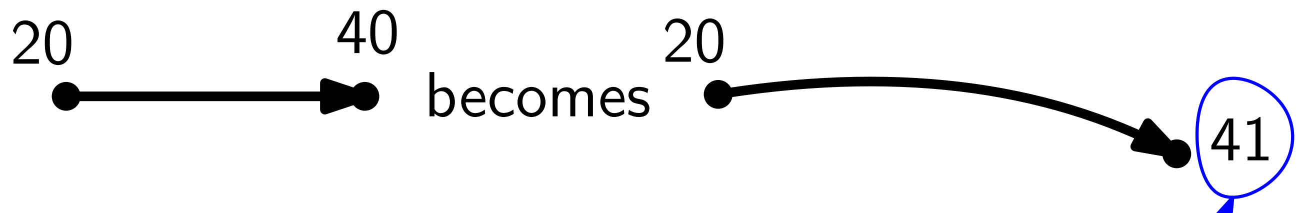
Problem:



But I claimed this would be at least 80



Problem:



But I claimed this would be at least 80

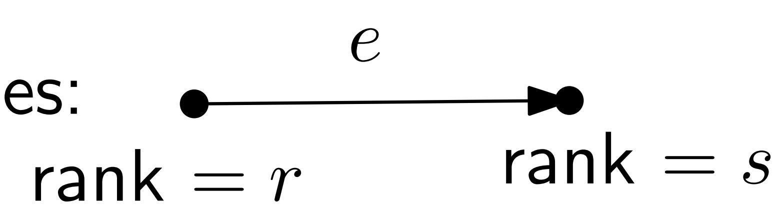
Let’s summarize and generalize

Let's summarize and generalize

Different thickness types:

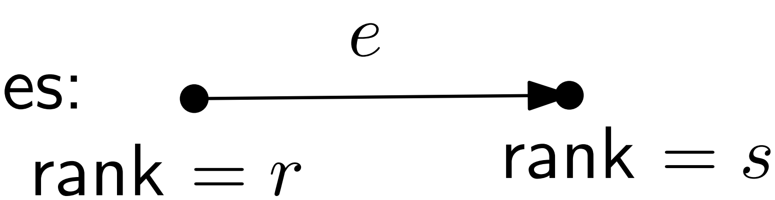
Let's summarize and generalize

Different thickness types:



Let's summarize and generalize

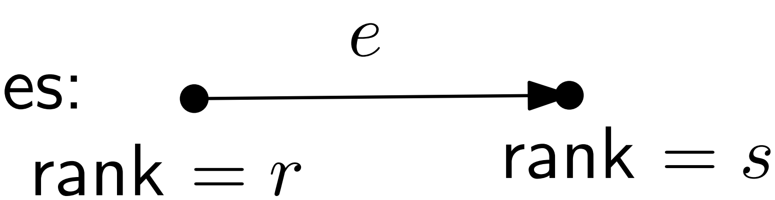
Different thickness types:



$r + 1 \leq s < 2r$: thickness 1

Let's summarize and generalize

Different thickness types:

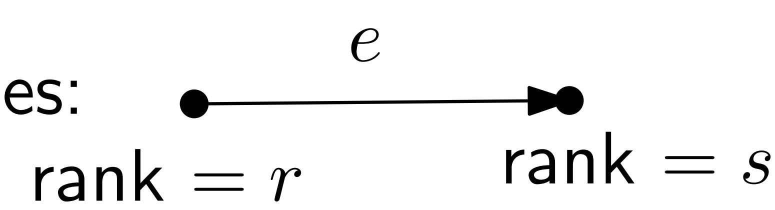


$r + 1 \leq s < 2r$: thickness 1

$2r \leq s < r\,2^r$: thickness 2

Let's summarize and generalize

Different thickness types:



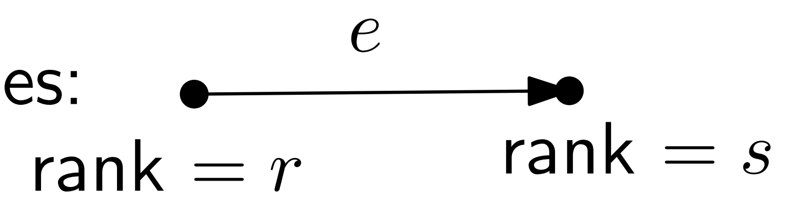
$r+1 \leq s < 2r$: thickness 1

$2r \leq s < r\,2^r$: thickness 2

$r\,2^r \leq s$: thickness 3

Let's summarize and generalize

Different thickness types:



$r + 1 \leq s < 2r$: thickness 1

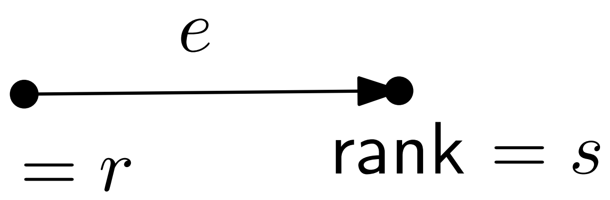
$2r \leq s < r\,2^r$: thickness 2

$r\,2^r \leq s$: thickness 3

Why stop here?

Let's summarize and generalize

Different thickness types:



$f_1(r)$

$r+1 \leq s < 2r$: thickness 1

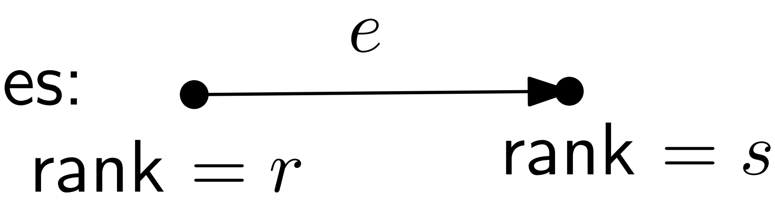
$2r \leq s < r\,2^r$: thickness 2

$r\,2^r \leq s$: thickness 3

Why stop here?

Let's summarize and generalize

Different thickness types:



$f_1(r)$

$r+1 \le s < 2r$: thickness 1

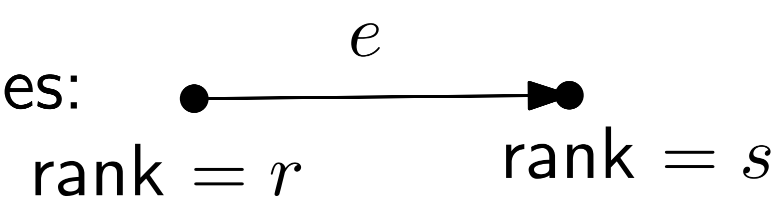
$f_2(r)$ $2r \le s < r\,2^r$: thickness 2

$r\,2^r \le s$: thickness 3

Why stop here?

Let's summarize and generalize

Different thickness types:



$f_1(r)$

$r + 1 \leq s < 2r$: thickness 1

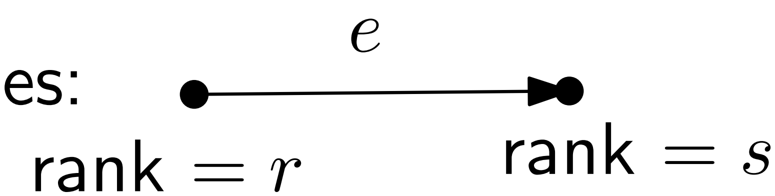
$f_2(r)$ $2r \leq s < r\,2^r$: thickness 2

$f_3(r)$ $r\,2^r \leq s$: thickness 3

Why stop here?

Let's summarize and generalize

Different thickness types:



$f_1(r)$

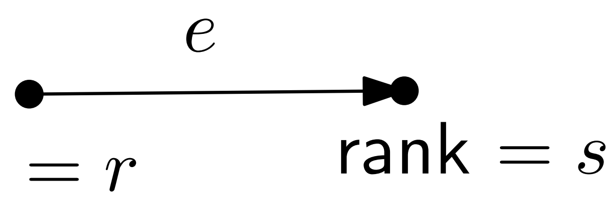
$r + 1 \leq s < 2r$: thickness 1

$f_2(r)$ $2r \leq s < r\,2^r$: thickness 2

$f_3(r)$ $r\,2^r \leq s < f_4(r)$: thickness 3

Let's summarize and generalize

Different thickness types:



$f_1(r)$

$r + 1 \leq s < 2r$: thickness 1

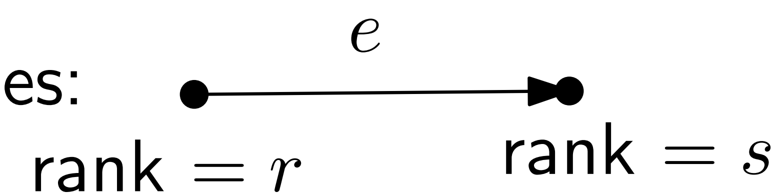
$f_2(r)$ $2r \leq s < r\,2^r$: thickness 2

$f_3(r)$ $r\,2^r \leq s < f_4(r)$: thickness 3

$f_4(r) \leq s < f_5(r)$: thickness 4

Let's summarize and generalize

Different thickness types:



$f_1(r)$

$r+1 \leq s < 2r$: thickness 1

$f_2(r)$ $2r \leq s < r\,2^r$: thickness 2

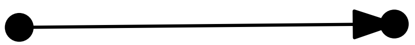
$f_3(r)$ $r\,2^r \leq s < f_4(r)$: thickness 3

$f_4(r) \leq s < f_5(r)$: thickness 4

$\vdots$

Let's summarize and generalize

Different thickness types:



$f_1(r)$

$r+1 \leq s < 2r$: thickness 1

$f_2(r) = f_1(f_1(\ldots f_1(r)\ldots))$

$f_2(r)$ $2r \leq s < r\,2^r$: thickness 2

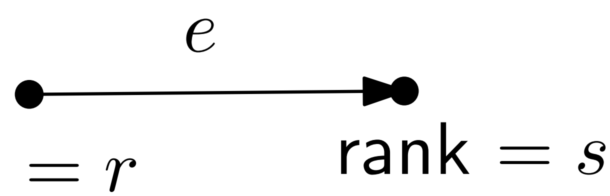
$f_3(r)$ $r\,2^r \leq s < f_4(r)$: thickness 3

$f_4(r) \leq s < f_5(r)$: thickness 4

$\vdots$

Let's summarize and generalize

Different thickness types:



$f_1(r)$

$r+1 \le s < 2r$: thickness 1

$f_2(r)$

$2r \le s < r\,2^r$: thickness 2

$f_3(r)$

$r\,2^r \le s < f_4(r)$: thickness 3

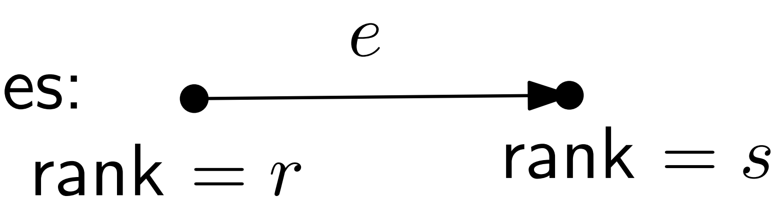
$f_4(r) \le s < f_5(r)$: thickness 4

$\vdots$

$$f_2(r) = \underbrace{f_1(f_1(\ldots f_1}_{r \text{ times}}(r)\ldots))$$

Let's summarize and generalize

Different thickness types:



$f_1(r)$

$r+1 \leq s < 2r$: thickness 1

$f_2(r)$ $2r \leq s < r\,2^r$: thickness 2

$f_3(r)$ $r\,2^r \leq s < f_4(r)$: thickness 3

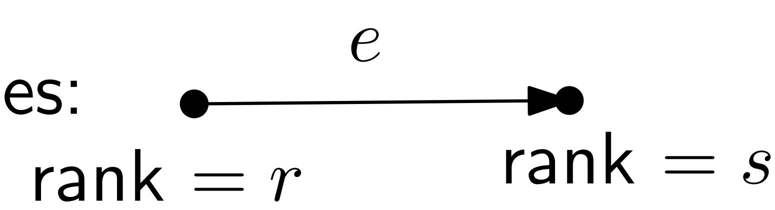
$f_4(r) \leq s < f_5(r)$: thickness 4

$\vdots$

$$f_2(r) = \overbrace{f_1(f_1(\ldots f_1}^{r \text{ times}}(r)\ldots)) = f_1^{(r)}(r)$$

Let's summarize and generalize

Different thickness types:



$f_1(r)$

$r+1 \le s < 2r$: thickness 1

$f_2(r)$ $2r \le s < r\,2^r$: thickness 2

$f_3(r)$ $r\,2^r \le s < f_4(r)$: thickness 3

$f_4(r) \le s < f_5(r)$: thickness 4

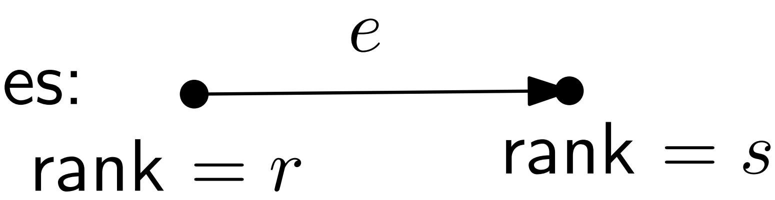
$\vdots$

$$f_2(r) = \underbrace{f_1(f_1(\dots f_1}_{r \text{ times}}(r)\dots)) = f_1^{(r)}(r)$$

$$f_3(r) = f_2^{(r)}(r)$$

Let's summarize and generalize

Different thickness types:



$f_1(r)$: $r+1 \leq s < 2r$: thickness 1

$f_2(r)$: $2r \leq s < r\,2^r$: thickness 2

$f_3(r)$: $r\,2^r \leq s < f_4(r)$: thickness 3

$f_4(r) \leq s < f_5(r)$: thickness 4

$\vdots$

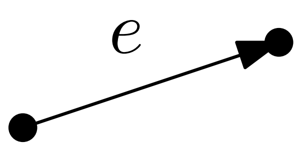
$$f_2(r) = \underbrace{f_1(f_1(\ldots f_1}_{r \text{ times}}(r)\ldots)) = f_1^{(r)}(r)$$

$$f_3(r) = f_2^{(r)}(r)$$

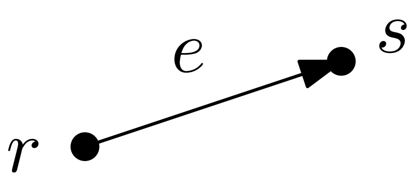
$$f_4(r) = f_3^{(r)}(r)$$

Weak and Strong Edge Redirections

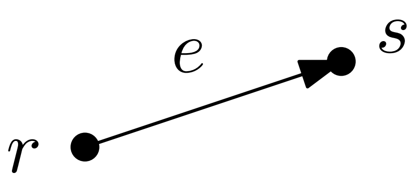
Weak and Strong Edge Redirections



Weak and Strong Edge Redirections

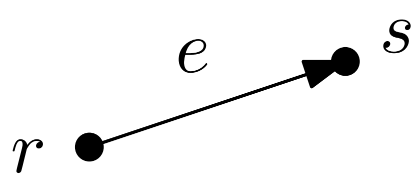


Weak and Strong Edge Redirections



$$\text{thickness}(e) = i$$

Weak and Strong Edge Redirections



$$\text{thickness}(e) = i$$

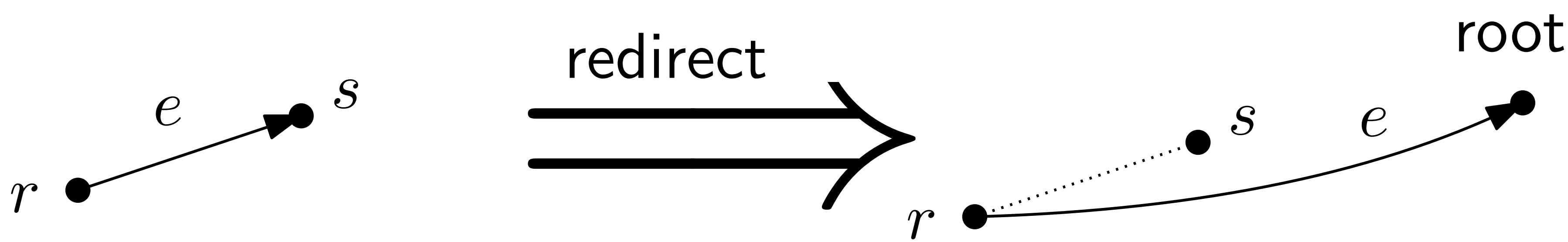
$$f_i(r) \leq s < f_{i+1}(r)$$

Weak and Strong Edge Redirections

$$\text{thickness}(e) = i$$

$$f_i(r) \leq s < f_{i+1}(r)$$

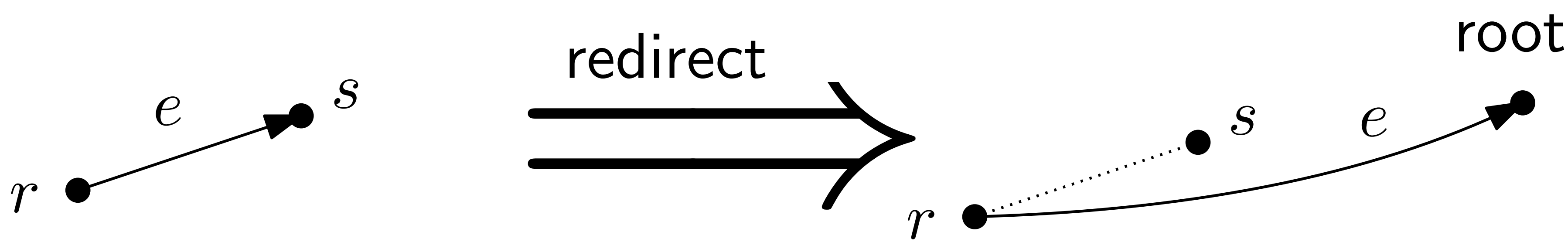
Weak and Strong Edge Redirections



$\mathrm{thickness}(e) = i$

$f_i(r) \leq s < f_{i+1}(r)$

Weak and Strong Edge Redirections

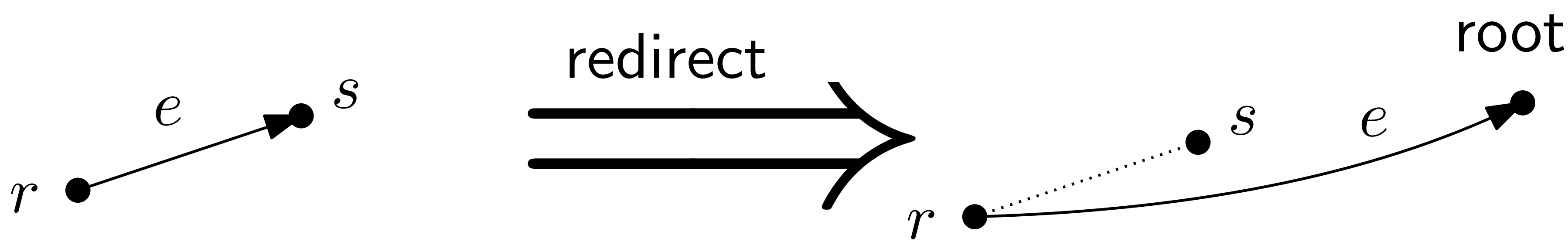


$\text{thickness}(e) = i$

$f_i(r) \leq s < f_{i+1}(r)$

Case 1: e is the last edge of thickness i on this path.

Weak and Strong Edge Redirections



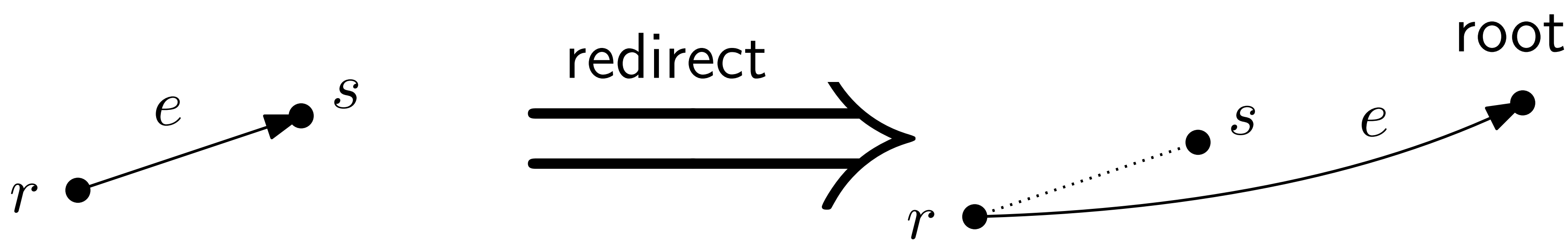
$\text{thickness}(e) = i$

$f_i(r) \leq s < f_{i+1}(r)$

Case 1: e is the last edge of thickness i on this path.

Then `find` operation pays one  for this.

Weak and Strong Edge Redirections



$\text{thickness}(e) = i$

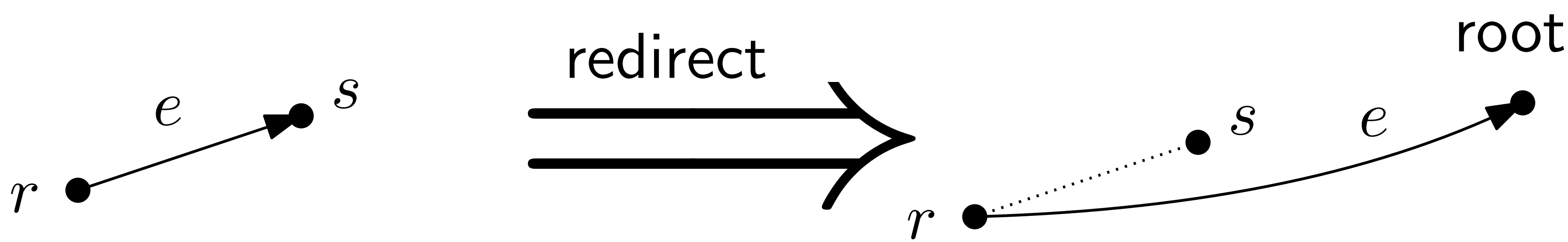
$f_i(r) \leq s < f_{i+1}(r)$

Case 1: e is the last edge of thickness i on this path.

Then `find` operation pays one  for this.

This is a *weak* edge redirection.

Weak and Strong Edge Redirections



$\mathrm{thickness}(e) = i$

$f_i(r) \leq s < f_{i+1}(r)$

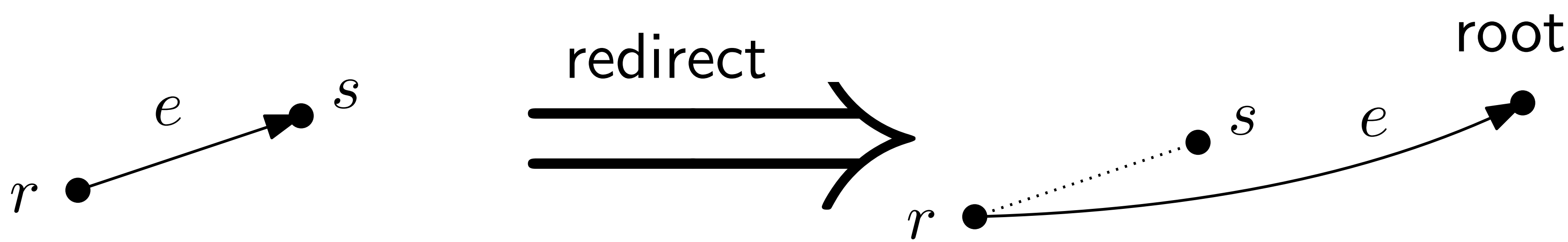
Case 1: e is the last edge of thickness i on this path.

Then `find` operation pays one  for this.

This is a *weak* edge redirection.

Total cost: number of thickness types.

Weak and Strong Edge Redirections

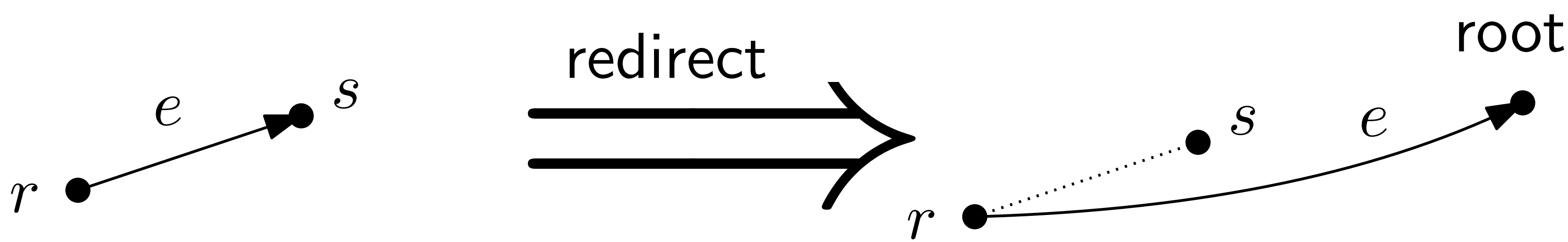


$\text{thickness}(e) = i$

$f_i(r) \leq s < f_{i+1}(r)$

Case 2: e is not the last edge of thickness i on this path.

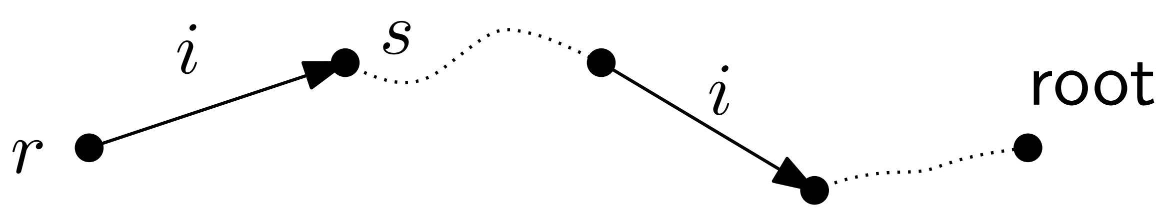
Weak and Strong Edge Redirections



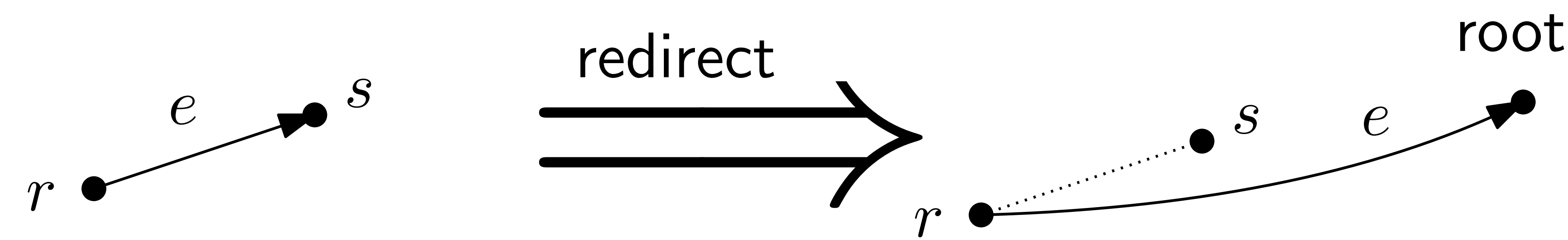
$\text{thickness}(e) = i$

$f_i(r) \leq s < f_{i+1}(r)$

Case 2: e is not the last edge of thickness i on this path.



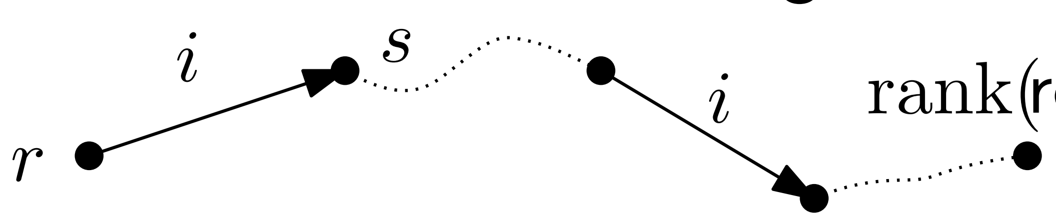
Weak and Strong Edge Redirections



$\text{thickness}(e) = i$

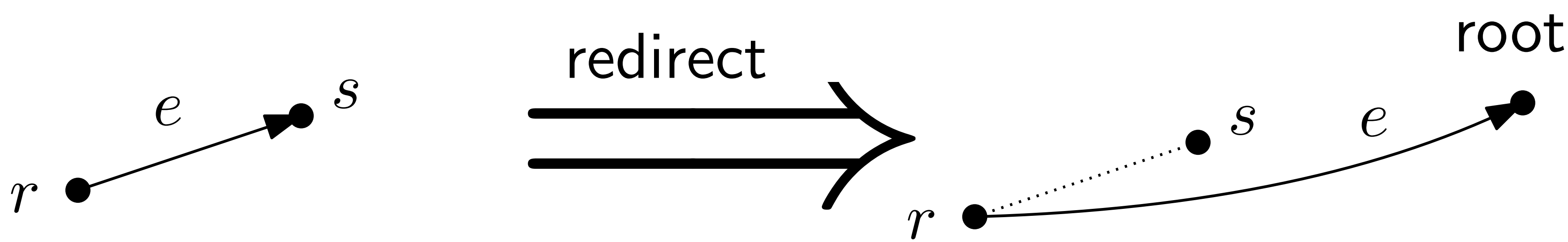
$f_i(r) \leq s < f_{i+1}(r)$

Case 2: e is not the last edge of thickness i on this path.



$\text{rank}(\text{root}) \geq f_i(f_i(r))$

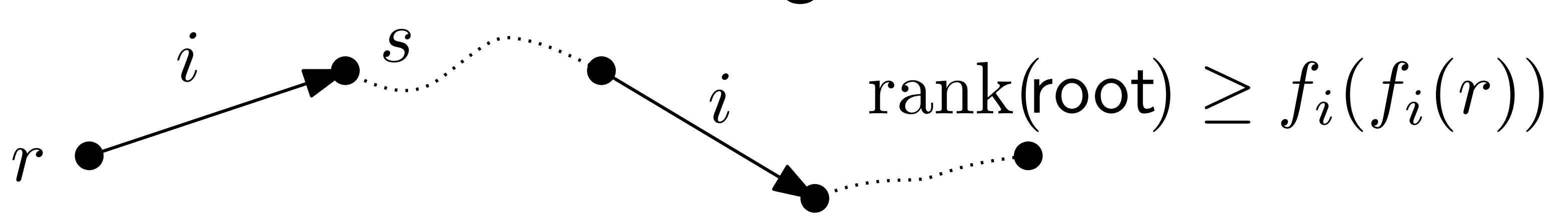
Weak and Strong Edge Redirections



$\text{thickness}(e) = i$

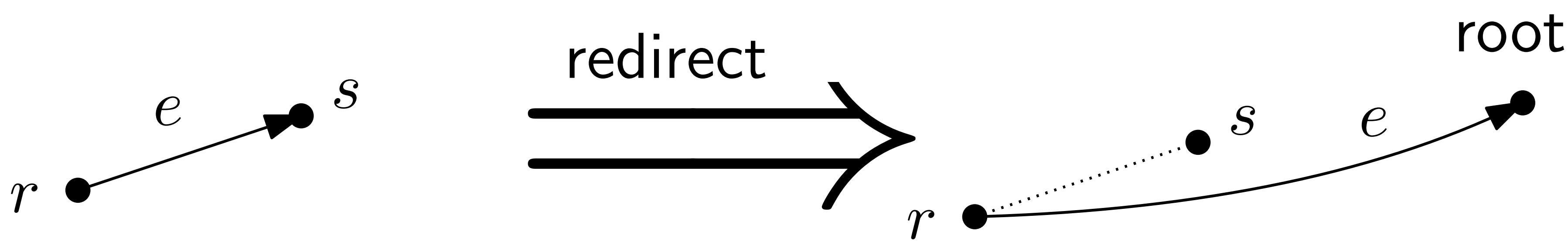
$f_i(r) \leq s < f_{i+1}(r)$

Case 2: e is not the last edge of thickness i on this path.



strong redirection

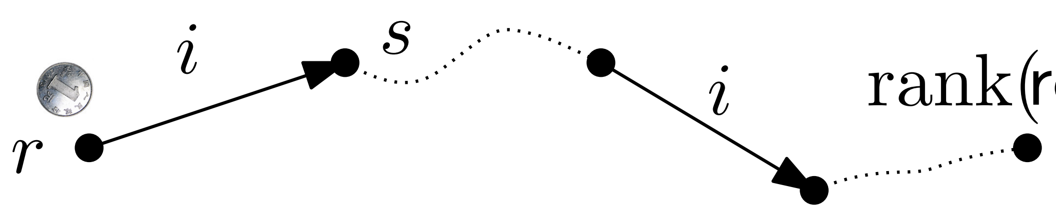
Weak and Strong Edge Redirections



$\mathrm{thickness}(e) = i$

$f_i(r) \leq s < f_{i+1}(r)$

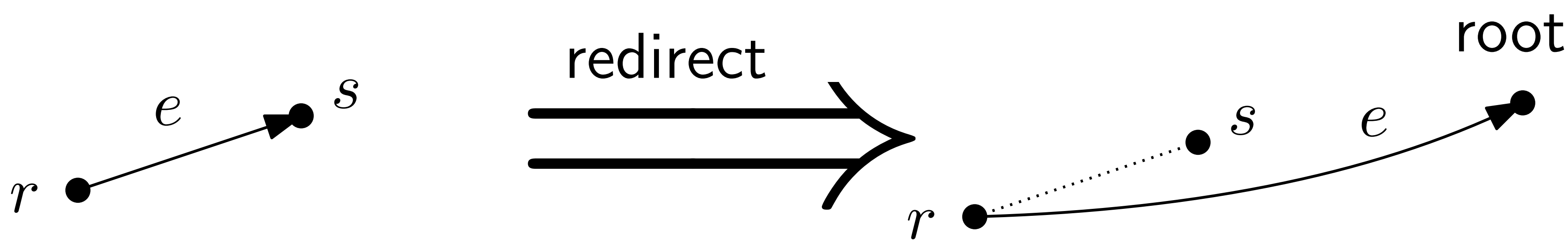
Case 2: e is not the last edge of thickness i on this path.



$\mathrm{rank}(\text{root}) \geq f_i(f_i(r))$

strong redirection

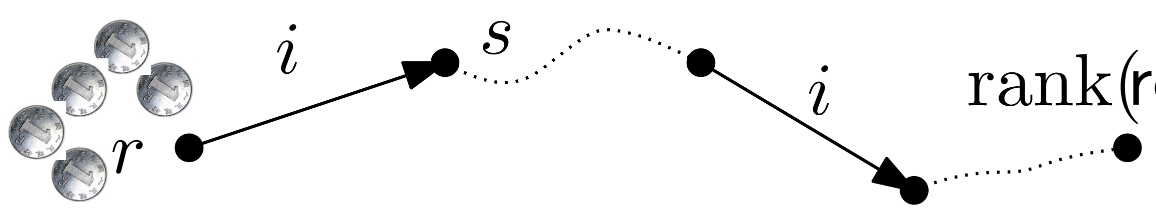
Weak and Strong Edge Redirections



$\text{thickness}(e) = i$

$f_i(r) \leq s < f_{i+1}(r)$

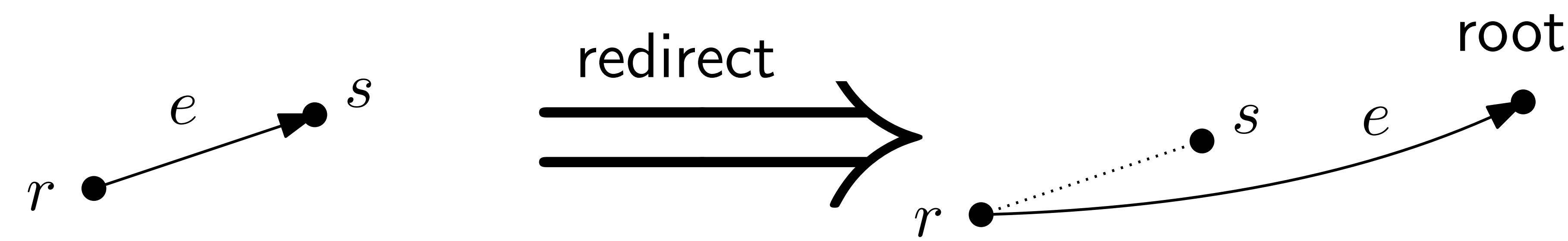
Case 2: e is not the last edge of thickness i on this path.



$\text{rank}(\text{root}) \geq f_i(f_i(r))$

after $r-1$ *strong* redirections: $\text{rank}(\text{root}) \geq f_i^{(r)}(r) = f_{i+1}(r)$

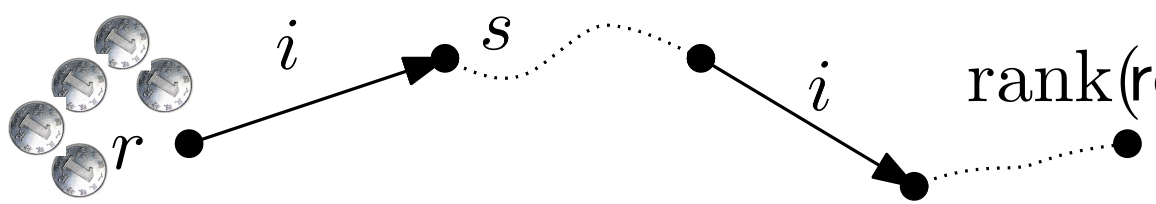
Weak and Strong Edge Redirections



$$\text{thickness}(e) = i$$

$$f_i(r) \leq s < f_{i+1}(r)$$

Case 2: e is not the last edge of thickness i on this path.



$\text{rank}(\text{root}) \geq f_i(f_i(r))$

after $r-1$ *strong* redirections: $\text{rank}(\text{root}) \geq f_i^{(r)}(r) = f_{i+1}(r)$

and the thickness of e increases to $i+1$

Putting Everything Together

Putting Everything Together

let ℓ be the number of thickness types occurring.

Putting Everything Together

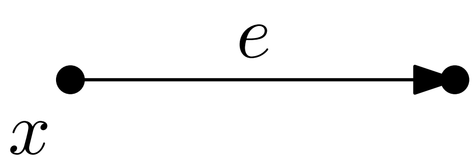
let ℓ be the number of thickness types occurring.

each `find` pays at most ℓ 

Putting Everything Together

let ℓ be the number of thickness types occurring.

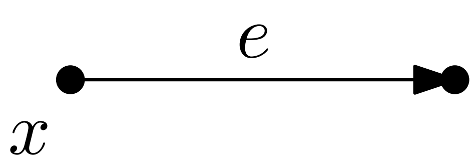
each `find` pays at most ℓ 



Putting Everything Together

let ℓ be the number of thickness types occurring.

each `find` pays at most ℓ

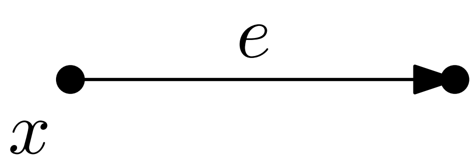


x pays at most $\mathrm{rank}(x)$
before the thickness of e increases.

Putting Everything Together

let ℓ be the number of thickness types occurring.

each `find` pays at most ℓ



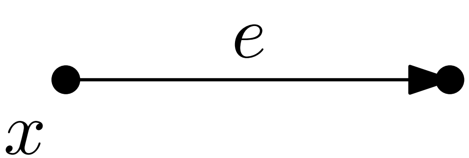
x pays at most $\mathrm{rank}(x)$
before the thickness of e increases.

Overall, x pays at most $\ell \cdot \mathrm{rank}(x)$

Putting Everything Together

let ℓ be the number of thickness types occurring.

each `find` pays at most ℓ 



x pays at most $\mathrm{rank}(x)$ 
before the thickness of e increases.

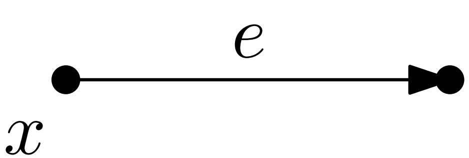
Overall, x pays at most $\ell \cdot \mathrm{rank}(x)$ 

$$\sum_x \ell \cdot \mathrm{rank}(x) = \ell \cdot \sum_r |\{\text{elements of rank } r\}|$$

Putting Everything Together

let ℓ be the number of thickness types occurring.

each `find` pays at most ℓ 



x pays at most $\mathrm{rank}(x)$ 
before the thickness of e increases.

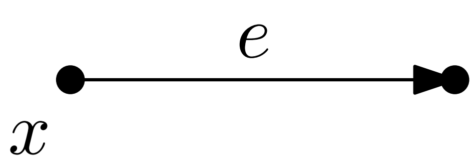
Overall, x pays at most $\ell \cdot \mathrm{rank}(x)$ 

$$\sum_x \ell \cdot \mathrm{rank}(x) = \ell \cdot \sum_r |\{\text{elements of rank } r\}|$$
$$\leq \ell \cdot \sum_r \frac{n}{2^{r-2}}$$

Putting Everything Together

let ℓ be the number of thickness types occurring.

each `find` pays at most ℓ 



x pays at most $\mathrm{rank}(x)$ 
before the thickness of e increases.

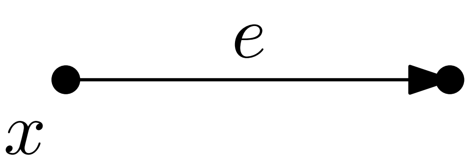
Overall, x pays at most $\ell \cdot \mathrm{rank}(x)$ 

$$\begin{aligned}\textstyle\sum_x \ell \cdot \mathrm{rank}(x) &= \ell \cdot \textstyle\sum_r |\{\text{elements of rank } r\}| \\ &\le \ell \cdot \textstyle\sum_r \frac{n}{2^{r-2}} \\ &= 6\,\ell n.\end{aligned}$$

Putting Everything Together

let ℓ be the number of thickness types occurring.

each `find` pays at most ℓ 



x pays at most $\mathrm{rank}(x)$ 
before the thickness of e increases.

Overall, x pays at most $\ell \cdot \mathrm{rank}(x)$ 

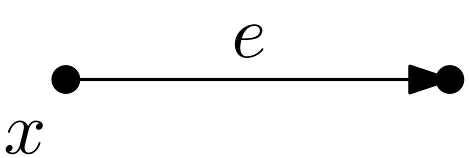
$$\begin{aligned} \textstyle\sum_x \ell \cdot \mathrm{rank}(x) &= \ell \cdot \textstyle\sum_r |\{\text{elements of rank } r\}| \\ &\le \ell \cdot \textstyle\sum_r \frac{n}{2^{r-2}} \\ &= 6\,\ell n. \end{aligned}$$

Overall cost is $O(\ell \cdot (n+m))$.

Putting Everything Together

let ℓ be the number of thickness types occurring.

each `find` pays at most ℓ



x pays at most $\mathrm{rank}(x)$
before the thickness of e increases.

Overall, x pays at most $\ell \cdot \mathrm{rank}(x)$

$$\sum_x \ell \cdot \mathrm{rank}(x) = \ell \cdot \sum_r |\{\text{elements of rank } r\}|$$
$$\leq \ell \cdot \sum_r \frac{n}{2^{r-2}}$$
$$= 6\,\ell n.$$

How large can ℓ become?

Overall cost is $O(\ell \cdot (n+m))$.

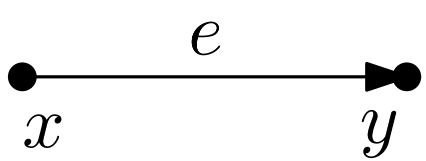
Overall running time is $O(\ell\,(n+m))$

Overall running time is $O(\ell\,(n+m))$

ℓ is the number of thickness levels

Overall running time is $O(\ell\,(n+m))$

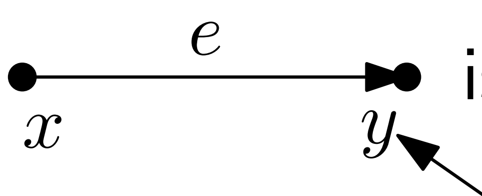
ℓ is the number of thickness levels

Imagine 

is an edge of thickness 5.

Overall running time is $O(\ell\,(n+m))$

ℓ is the number of thickness levels

Imagine 

is an edge of thickness 5.

$$\mathrm{rank}(y) \geq f_5(\mathrm{rank}(x)) \geq f_5(2)$$

What is $f_5(2)$?

What is $f_5(2)$?

$f_1(n) = n + 1$

What is $f_5(2)$?

$f_1(n) = n + 1$

$f_1(2) = 3$

What is $f_5(2)$?

$f_1(n) = n + 1$

$f_2(n) = 2n$

$f_1(2) = 3$

What is $f_5(2)$?

$$f_1(n) = n + 1 \qquad\qquad f_1(2) = 3$$

$$f_2(n) = 2n \qquad\qquad f_2(2) = 4$$

What is $f_5(2)$?

$$f_1(n) = n + 1$$

$$f_2(n) = 2n$$

$$f_3(n) = n\,2^n$$

$$f_1(2) = 3$$

$$f_2(2) = 4$$

What is $f_5(2)$?

$f_1(n) = n + 1$ $f_1(2) = 3$

$f_2(n) = 2n$ $f_2(2) = 4$

$f_3(n) = n\,2^n$ $f_3(2) = 8$

What is $f_5(2)$?

$$f_1(n) = n + 1 \qquad f_1(2) = 3$$

$$f_2(n) = 2n \qquad f_2(2) = 4$$

$$f_3(n) = n\,2^n \qquad f_3(2) = 8$$

$$f_4(2) = f_3(f_3(2)) = f_3(8)$$

What is $f_5(2)$?

$$f_1(n) = n + 1 \qquad f_1(2) = 3$$

$$f_2(n) = 2n \qquad f_2(2) = 4$$

$$f_3(n) = n\,2^n \qquad f_3(2) = 8$$

$$f_4(2) = f_3(f_3(2)) = f_3(8) = 8 \cdot 2^8 = 2048.$$

What is $f_5(2)$?

$f_1(n) = n + 1$ $\quad$ $f_1(2) = 3$

$f_2(n) = 2n$ $\quad$ $f_2(2) = 4$

$f_3(n) = n\, 2^n$ $\quad$ $f_3(2) = 8$

$$f_4(2) = f_3(f_3(2)) = f_3(8) = 8 \cdot 2^8 = 2048.$$

$$f_5(2) = f_4(f_4(2)) = f_4(2048)$$

What is $f_5(2)$?

$f_1(n) = n + 1$

$f_2(n) = 2n$

$f_3(n) = n\,2^n$

$f_1(2) = 3$

$f_2(2) = 4$

$f_3(2) = 8$

$$f_4(2) = f_3(f_3(2)) = f_3(8) = 8 \cdot 2^8 = 2048.$$

$$f_5(2) = f_4(f_4(2)) = f_4(2048) \geq f_3(f_3(f_3(\ldots f_3(2048)\ldots)))$$

What is $f_5(2)$?

$f_1(n) = n + 1$ $\qquad$ $f_1(2) = 3$

$f_2(n) = 2n$ $\qquad$ $f_2(2) = 4$

$f_3(n) = n\, 2^n$ $\qquad$ $f_3(2) = 8$

$$f_4(2) = f_3(f_3(2)) = f_3(8) = 8 \cdot 2^8 = 2048.$$

$$f_5(2) = f_4(f_4(2)) = f_4(2048) \geq \underbrace{f_3(f_3(f_3(\ldots f_3}_{2048 \text{ times}}(2048)\ldots)))$$

What is $f_5(2)$?

$f_1(n) = n + 1$ $\qquad$ $f_1(2) = 3$

$f_2(n) = 2n$ $\qquad$ $f_2(2) = 4$

$f_3(n) = n\,2^n$ $\qquad$ $f_3(2) = 8$

$$f_4(2) = f_3(f_3(2)) = f_3(8) = 8 \cdot 2^8 = 2048.$$

$$f_5(2) = f_4(f_4(2)) = f_4(2048) \geq \underbrace{f_3(f_3(f_3(\ldots f_3}_{2048 \text{ times}}(2048)\ldots)))$$

$$\geq 2^{2^{2^{\cdot^{\cdot^{\cdot^{2^{2048}}}}}}}$$

What is $f_5(2)$?

$f_1(n) = n + 1$ $\qquad$ $f_1(2) = 3$

$f_2(n) = 2n$ $\qquad$ $f_2(2) = 4$

$f_3(n) = n\,2^n$ $\qquad$ $f_3(2) = 8$

$$f_4(2) = f_3(f_3(2)) = f_3(8) = 8 \cdot 2^8 = 2048.$$

$$f_5(2) = f_4(f_4(2)) = f_4(2048) \geq \underbrace{f_3(f_3(f_3(\ldots f_3}_{2048 \text{ times}}(2048)\ldots)))$$

$$f_5(2) \geq 2^{2^{2^{\cdot^{\cdot^{\cdot^{2^{2048}}}}}}}$$

What is $f_5(2)$?

$f_1(n) = n + 1$ $\qquad$ $f_1(2) = 3$

$f_2(n) = 2n$ $\qquad$ $f_2(2) = 4$

$f_3(n) = n\, 2^n$ $\qquad$ $f_3(2) = 8$

$$f_4(2) = f_3(f_3(2)) = f_3(8) = 8 \cdot 2^8 = 2048.$$

$$f_5(2) = f_4(f_4(2)) = f_4(2048) \geq \underbrace{f_3(f_3(f_3(\ldots f_3}_{2048 \text{ times}}(2048)\ldots)))$$

$$\text{rank}(y) \geq f_5(2) \geq 2^{2^{2^{\cdot^{\cdot^{\cdot^{2^{2048}}}}}}}$$

What is $f_5(2)$?

$$f_1(n) = n + 1 \qquad\qquad f_1(2) = 3$$

$$f_2(n) = 2n \qquad\qquad f_2(2) = 4$$

$$f_3(n) = n\,2^n \qquad\qquad f_3(2) = 8$$

$$f_4(2) = f_3(f_3(2)) = f_3(8) = 8 \cdot 2^8 = 2048.$$

$$f_5(2) = f_4(f_4(2)) = f_4(2048) \geq \underbrace{f_3(f_3(f_3(\ldots f_3}_{\text{2048 times}}(2048)\ldots)))$$

$$\log(n) \geq \text{rank}(y) \geq f_5(2) \geq 2^{2^{2^{\cdot^{\cdot^{\cdot^{2^{2048}}}}}}}$$