UCSC Silicon Valley Extension

Advanced C Programming

Fibonacci Heaps

Radhika Grover

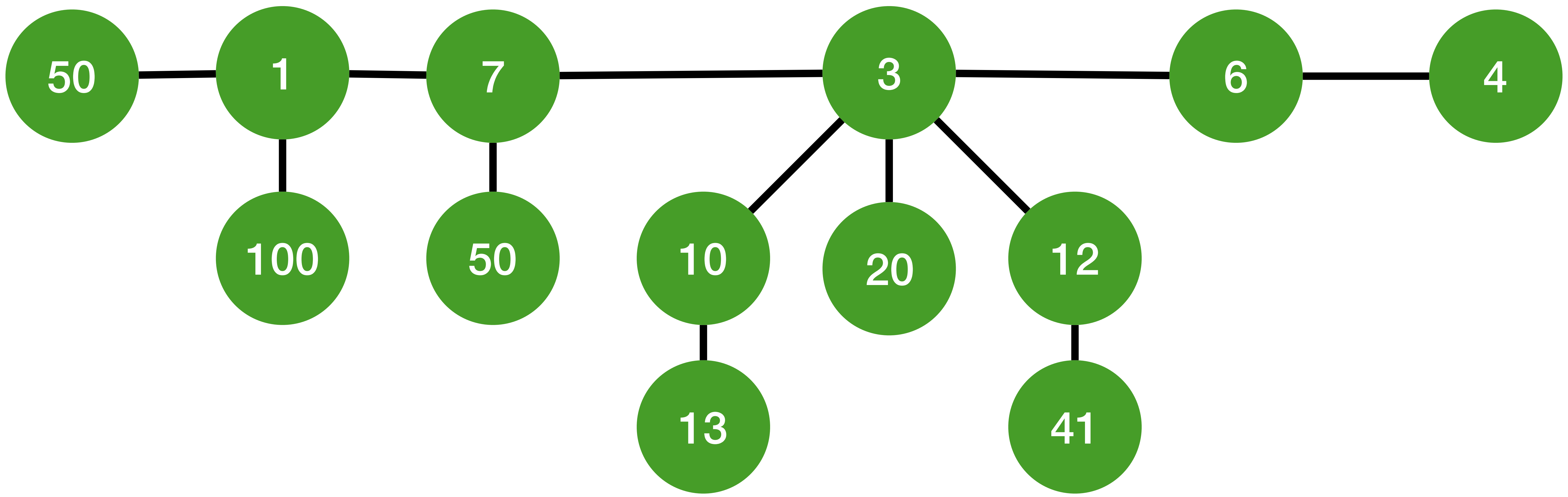
Fibonacci heaps

- Also used in minimum spanning tree
- Less structure than binomial heap
- Decrease-key and union operation O(1) time

 © SPEL Technologies, Inc. All rights reserved

Fibonacci heaps : properties

- Collection of min-heap-ordered tree
- Trees in a heap are unordered
- Children of node are linked in circular, doubly linked list
    - Removes a node in O(1) time



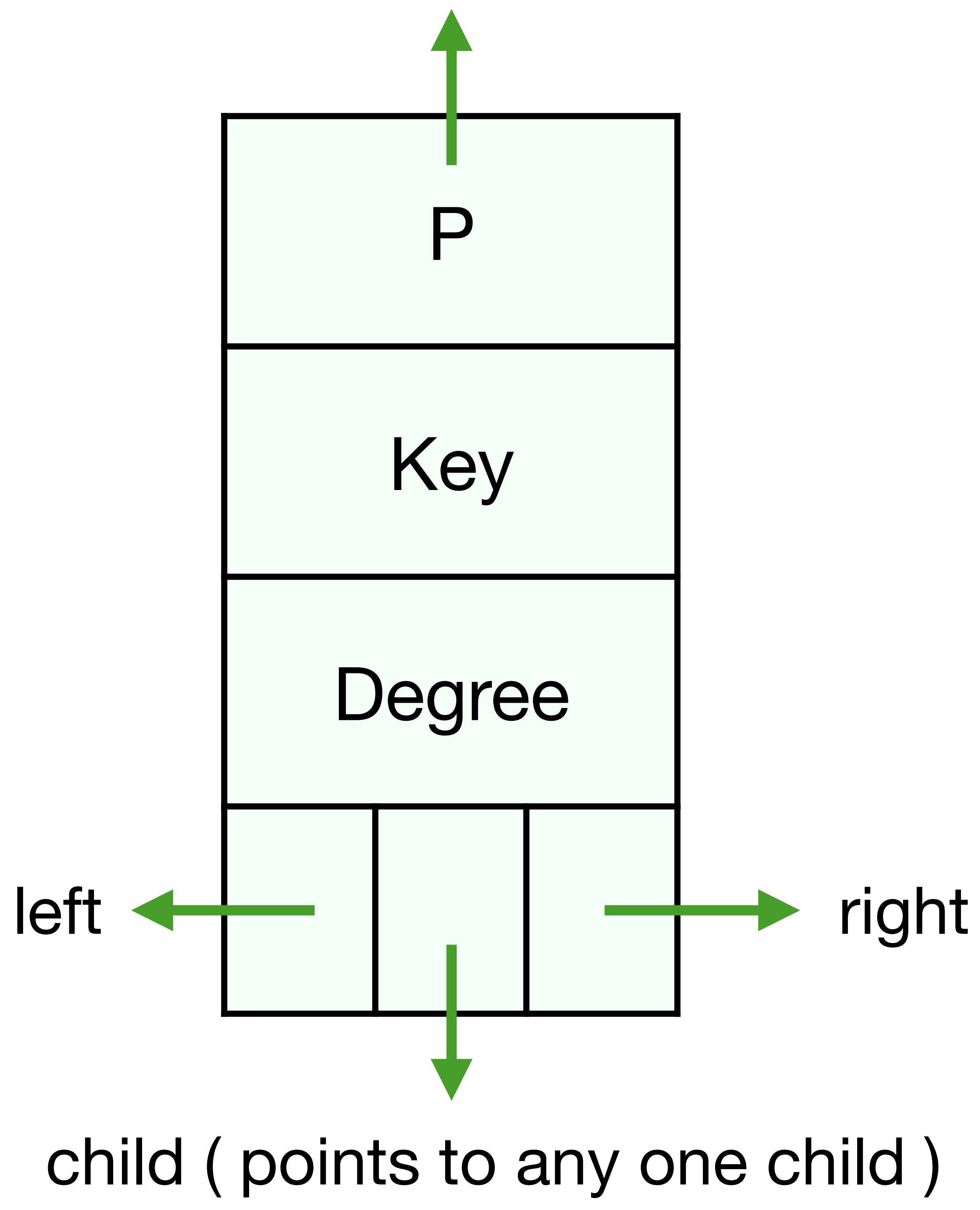
 © SPEL Technologies, Inc. All rights reserved

Fibonacci heaps: properties

- Tree can have any shape (can be a single node as well)
- Fibonacci numbers are used in analysis of running times (hence the name)
- Every node has a degree of at most $O(\log(n))$
- Size of subtree rooted in node of degree k is at least F_{k+2}, where F_k is the kth Fibonacci number.

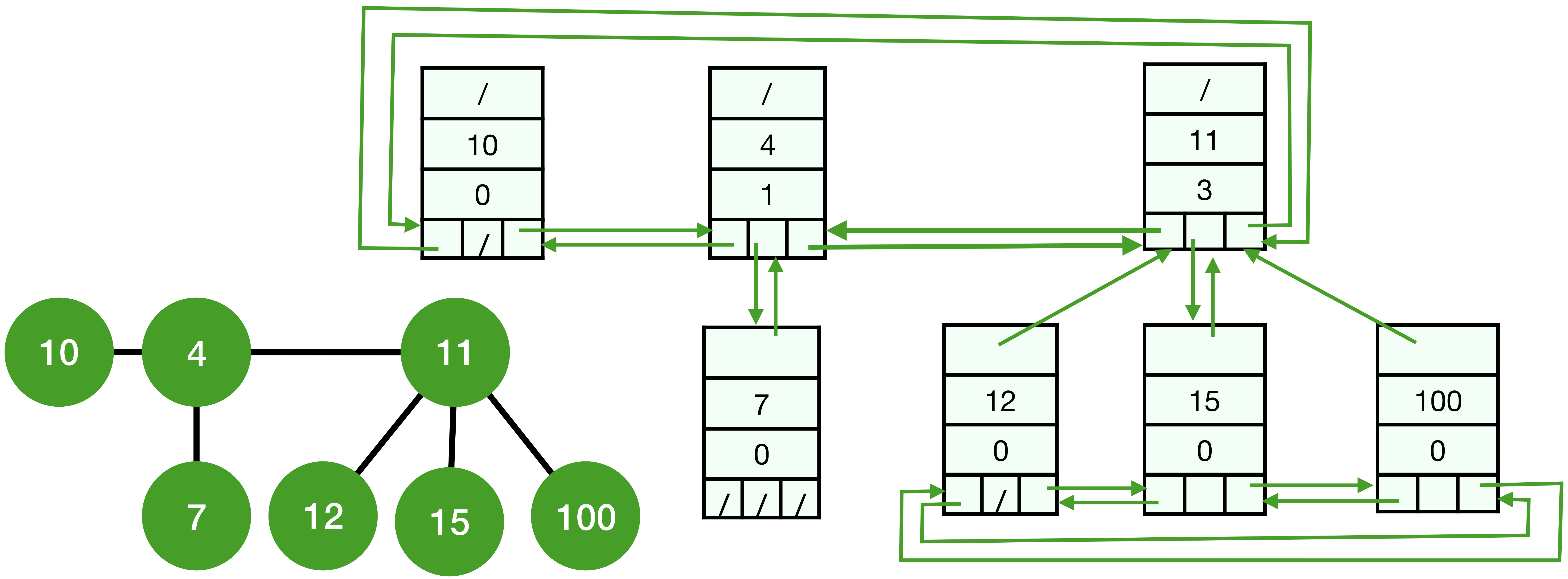
© SPEL Technologies, Inc. All rights reserved

Node structure in Fibonacci heaps



© SPEL Technologies, Inc. All rights reserved

Node structure in Fibonacci heaps

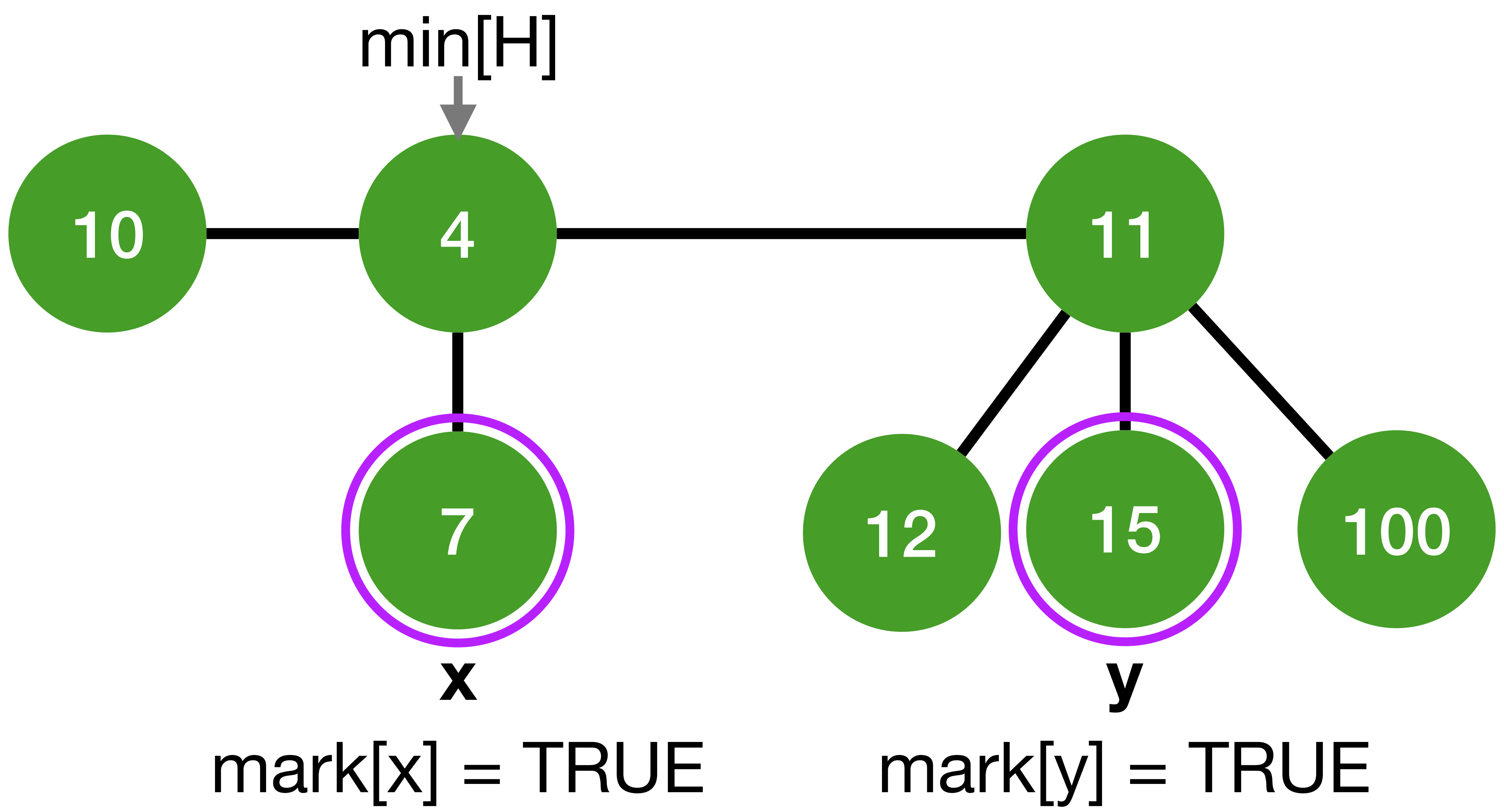


root-list : doubly linked list linking all roots

© SPEL Technologies, Inc. All rights reserved

Fibonacci heap : improvements

- Store pointer *min* to node with minimum key
- Each node x stores a boolean value mark[x] set to FALSE
  - Updated in the DECREASE - KEY operation



© SPEL Technologies, Inc. All rights reserved

Amortized Analysis

- More sophisticated than looking at worst-case bounds for operations.
- Shares cost of a single expensive operation with many other cheaper operations.

 © SPEL Technologies, Inc. All rights reserved

Fibonacci heap : potential function

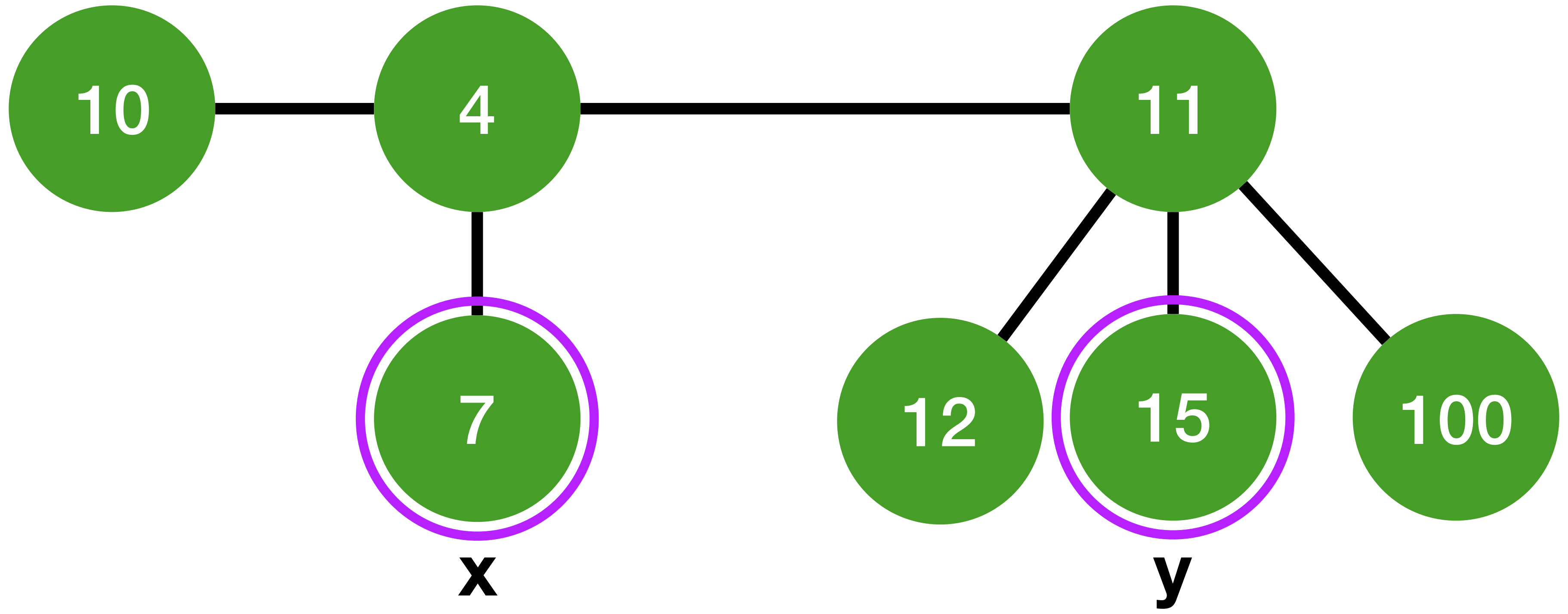
- Used to analyze performance

Ø(H) : potential function

t(H) : number of trees in H

m(H) : number of marked nodes

Ø(H) = t(H) + 2m(H)



t(H) = 3 , m(H) = 2 => Ø(H) = 3 + (2 x 2) = 7

© SPEL Technologies, Inc. All rights reserved

Fibonacci heap : constant time operations

- Find element with minimum key
- Merge two root lists together
- Add a node to root list
- Remove a node from root list

© SPEL Technologies, Inc. All rights reserved

Fibonacci heap : create

```
MakeFibHeap(){
   create empty heap H;
   min[H] = NULL ;
   return H;
}
```

 © SPEL Technologies, Inc. All rights reserved

Fibonacci heap : insert

```
Note: trees with same rank are not merged

//insert node x into heap H
FibHeapInsert(H, x){
   Add root list with x to H;
   if(min(H) == NULL || key[x] < key[min[H]])
      min[H] = x;
}

// x is defined as follows
   degree[x] = 0;
   P[x] = child[x] = NULL;
   left[x] = right[x] = x;
   mark[x] = FALSE;
```


© SPEL Technologies, Inc. All rights reserved

Fibonacci heap : insert potential

Actual Cost : O(1)

potential before insert : t(H) + 2m(H)

potential after insert : t(H) + 1 + 2m(H)

change in potential = 1

amortized cost = cost + change in potential = O(1)

© SPEL Technologies, Inc. All rights reserved

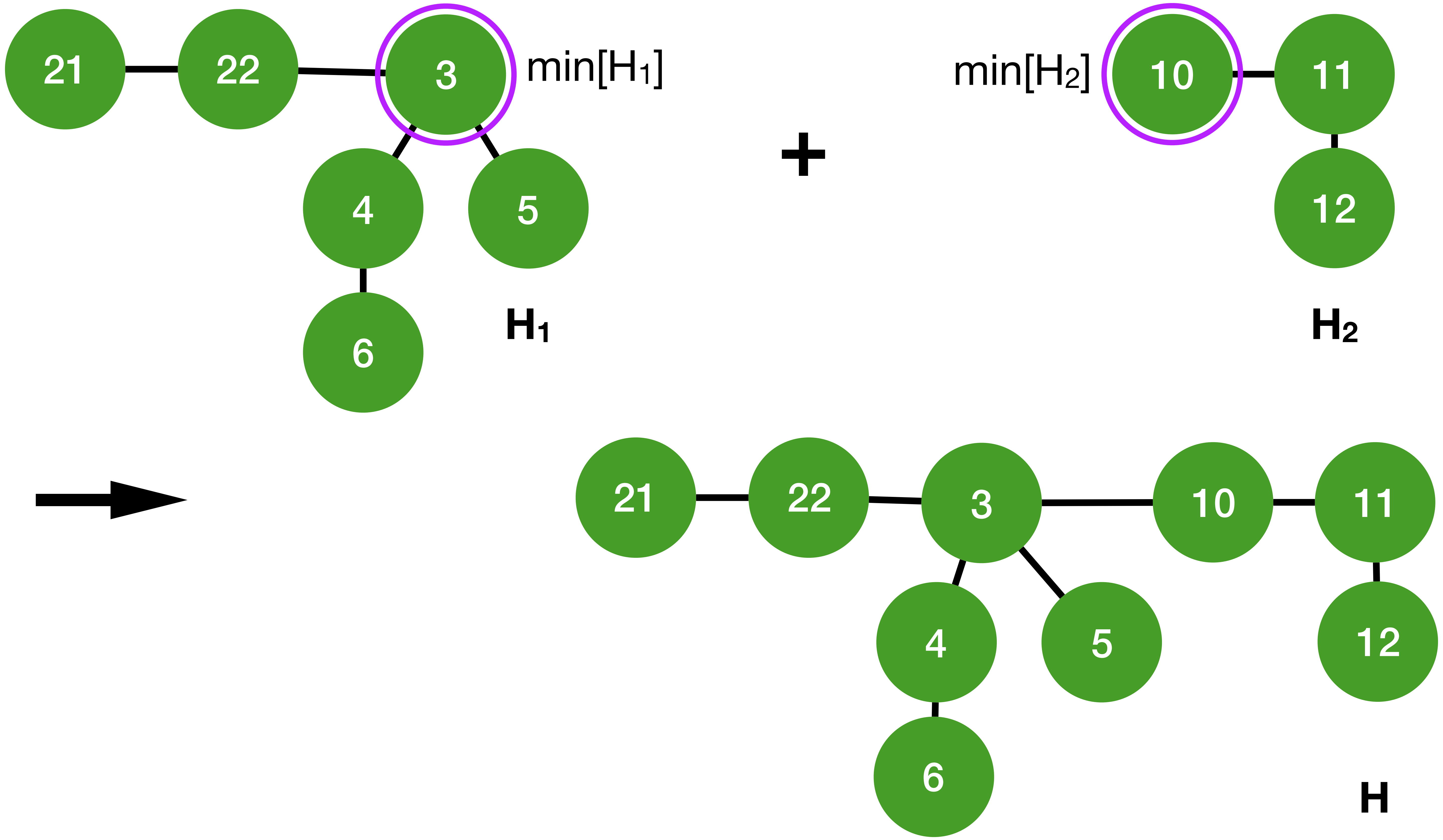
Fibonacci heap : finding the minimum

Pointer min(H) points to the node with minimum key cost : O(1)

© SPEL Technologies, Inc. All rights reserved

Fibonacci heap : union

```
//concatenate the root list and update min
```



© SPEL Technologies, Inc. All rights reserved

Fibonacci heap : union

```
FibHeapUnion(H₁,H₁){
  H = MakeFibHeap();
  H = Concatenate root lists of H₁ and H₂
  min[H] = smaller of key[min[H₁]]
           and key[min[H₂]]
  free H₁ and H₂ ;
  return H;
}
```

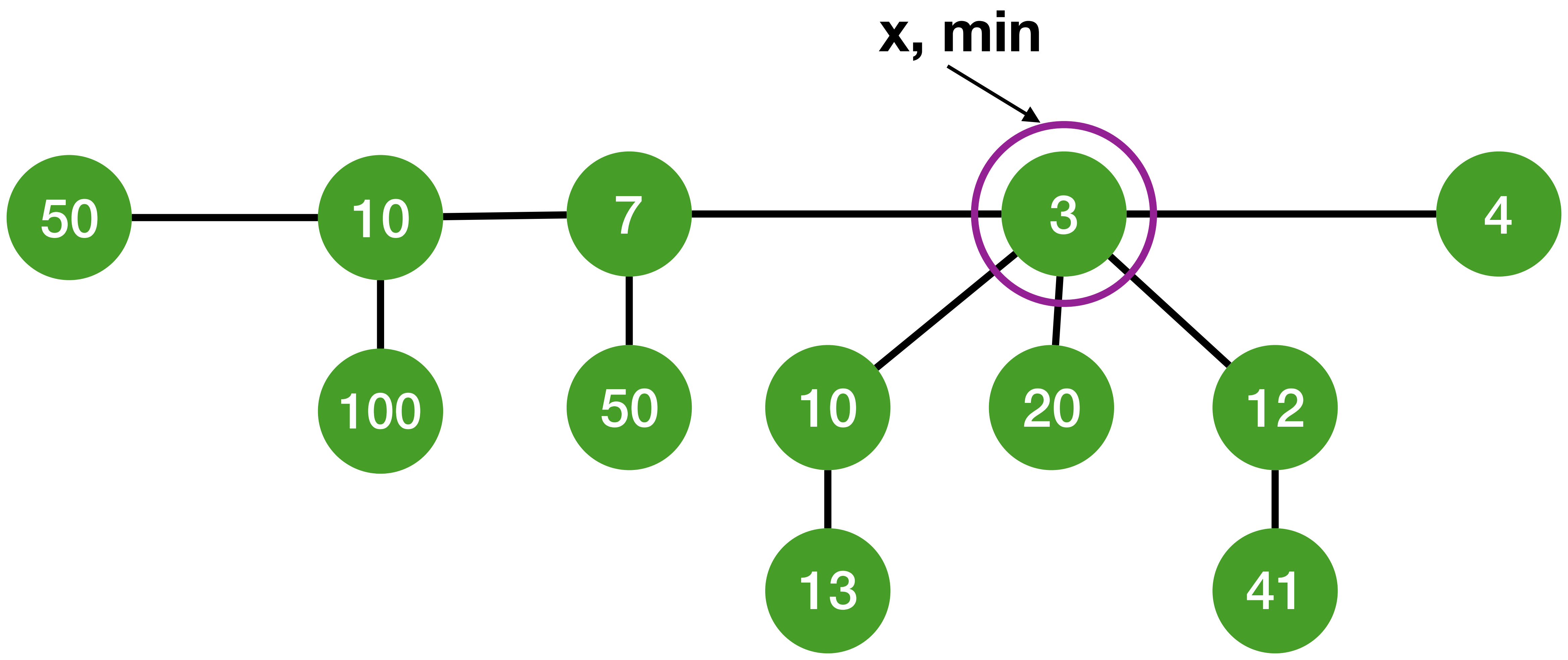

© SPEL Technologies, Inc. All rights reserved

Fibonacci heap : delete min

- Delete node with smallest key
- Consolidate trees so all have different degrees

 © SPEL Technologies, Inc. All rights reserved

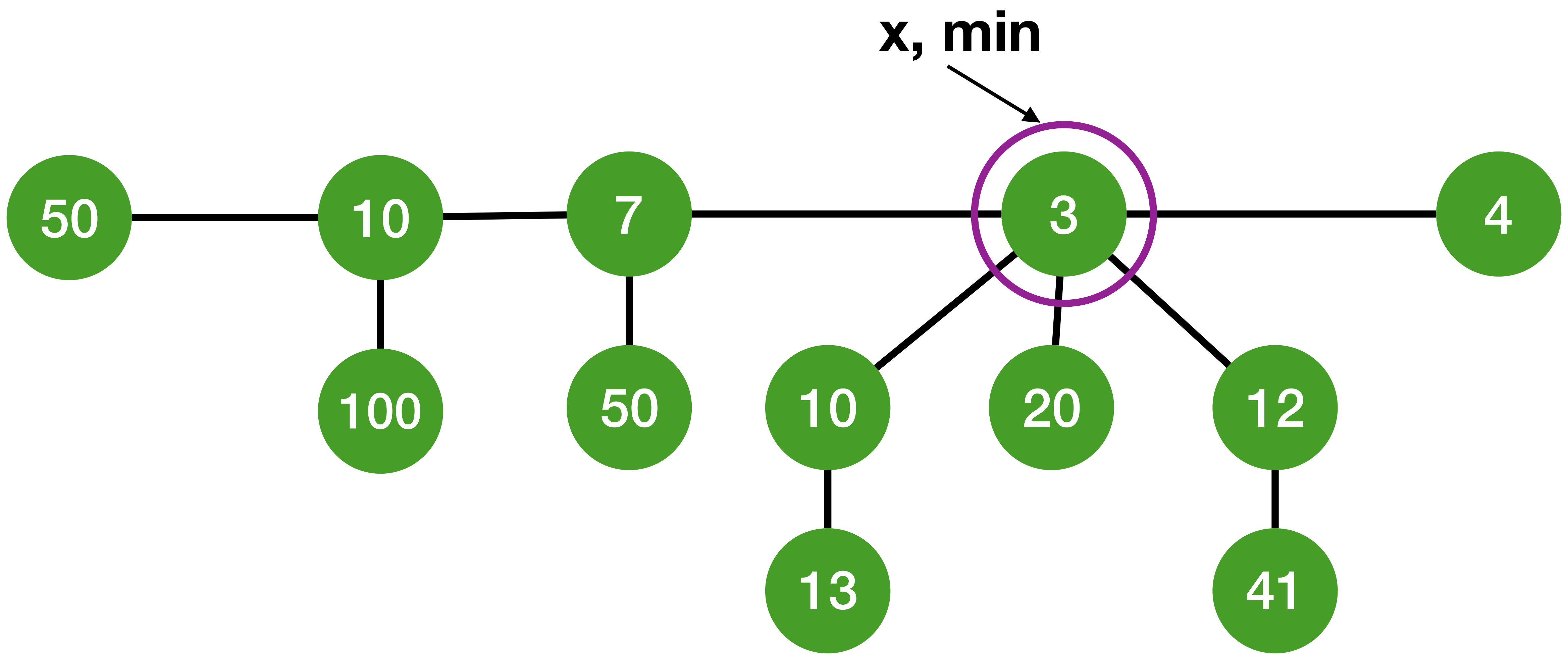
Fibonacci heap : delete min



Delete Node x

© SPEL Technologies, Inc. All rights reserved

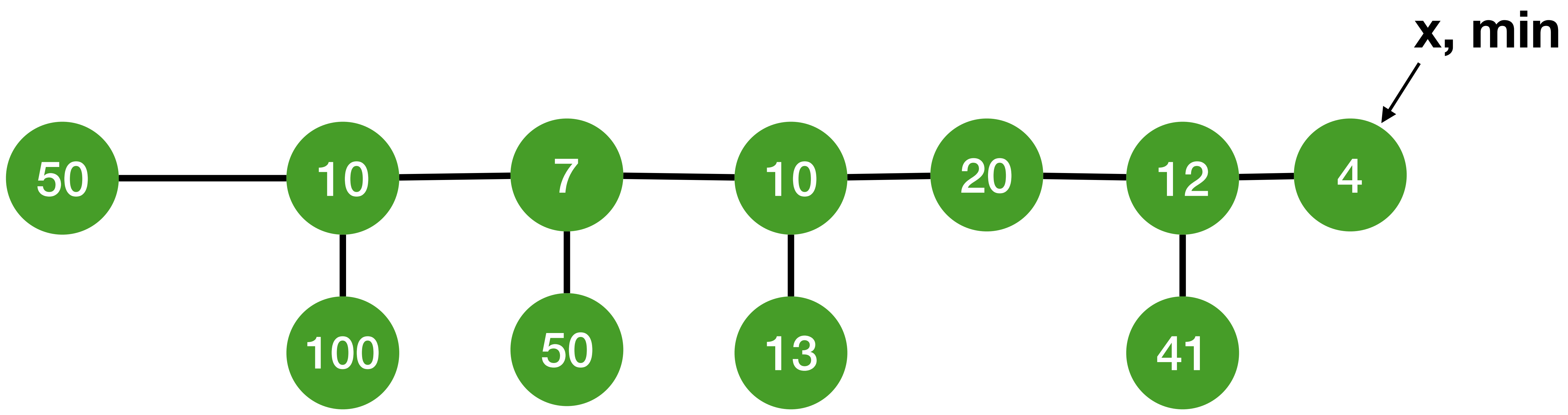
Fibonacci heap : delete min



Delete Node x

© SPEL Technologies, Inc. All rights reserved

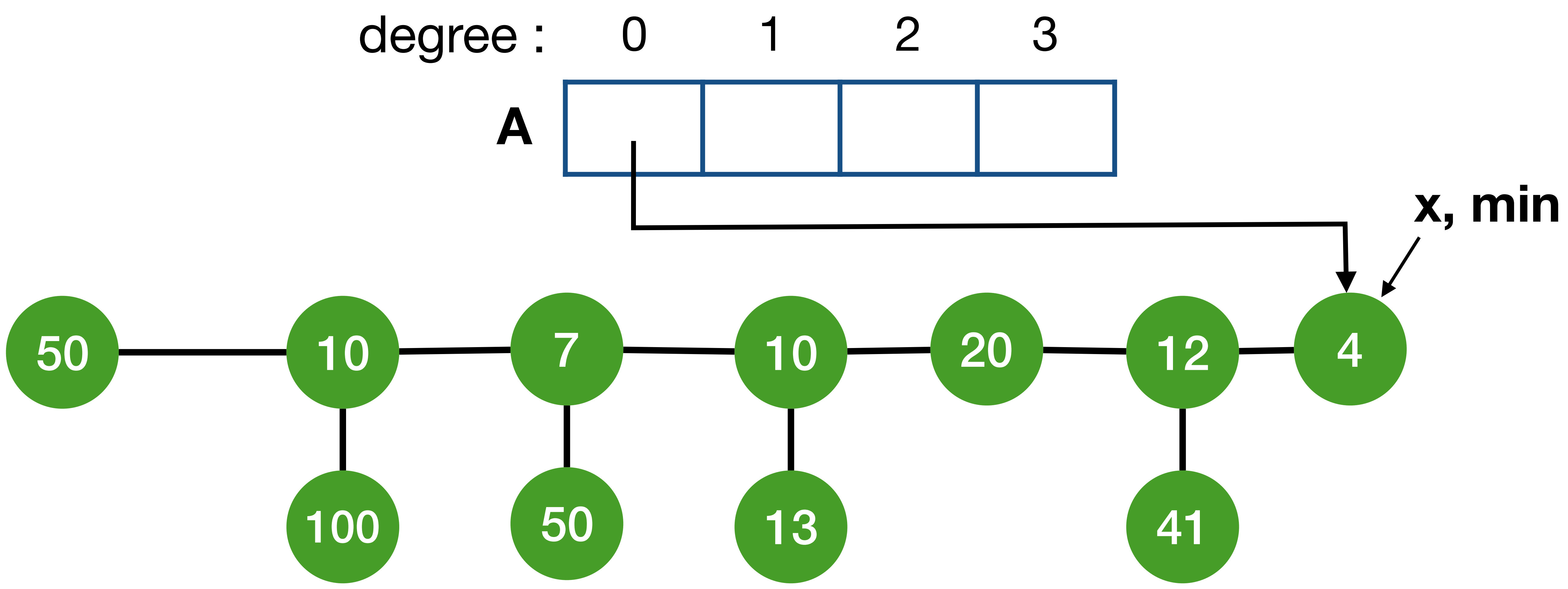
Fibonacci heap : delete min



Add children of Node x to root list update min, x points to min

© SPEL Technologies, Inc. All rights reserved

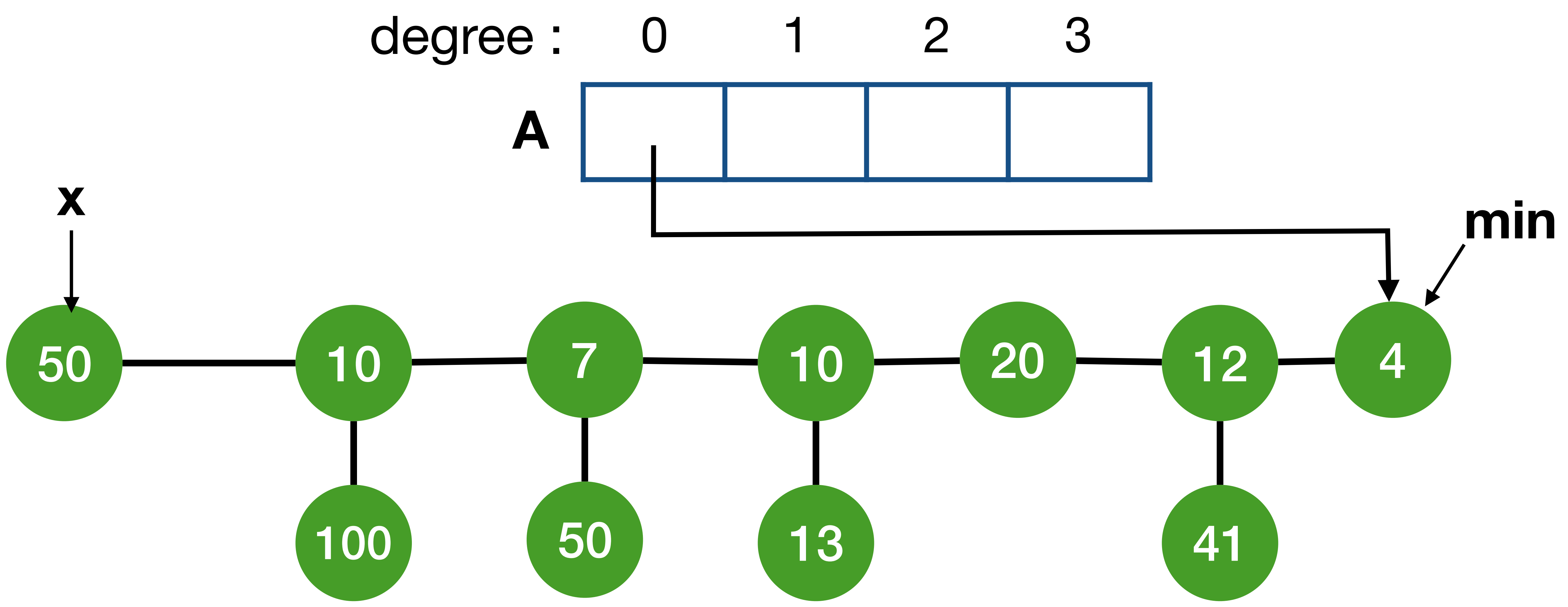
Fibonacci heap : delete min



x has degree 0 - update A

© SPEL Technologies, Inc. All rights reserved

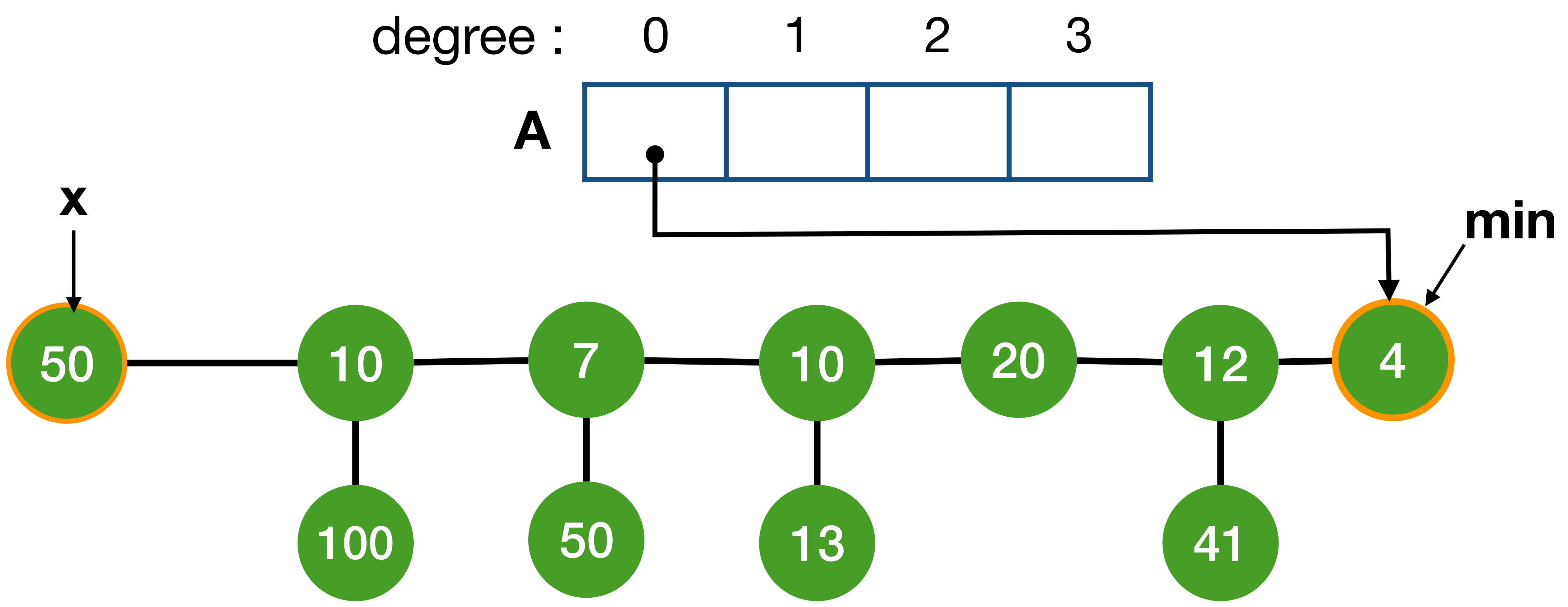
Fibonacci heap : delete min



move x to next node to link tree with same degree together

© SPEL Technologies, Inc. All rights reserved

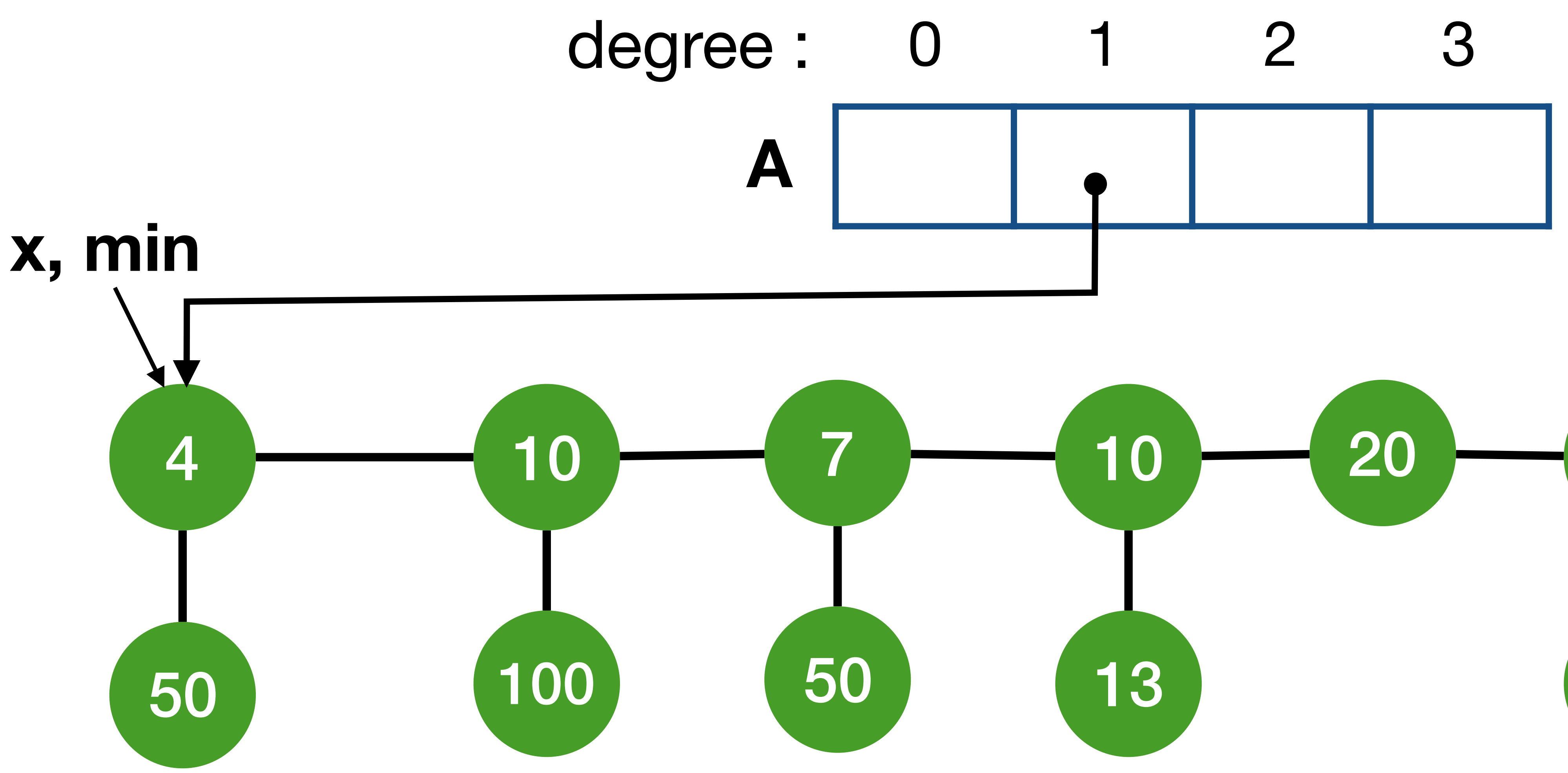
Fibonacci heap : delete min



50 also has degree 0 as 4

© SPEL Technologies, Inc. All rights reserved

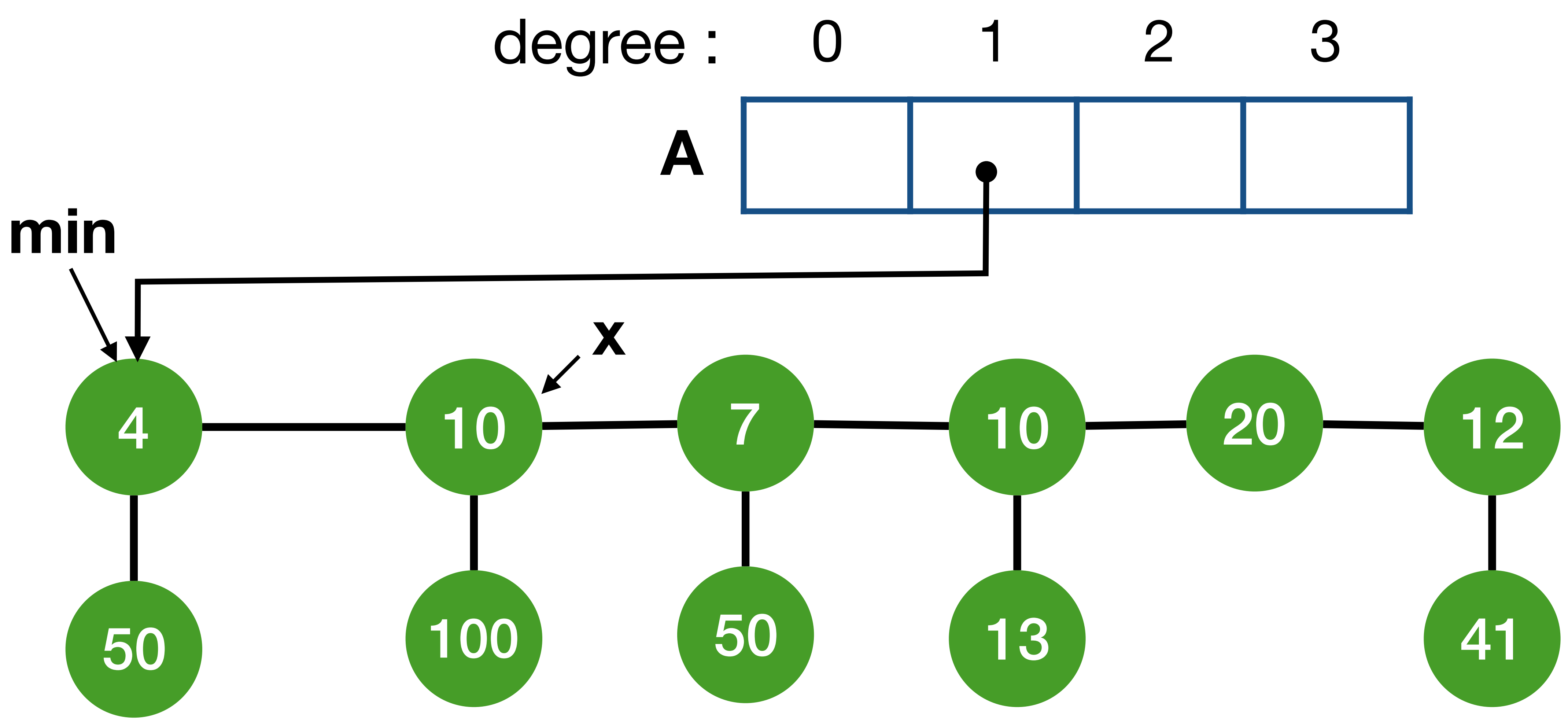
Fibonacci heap : delete min



link with 50 with 4 and update A

© SPEL Technologies, Inc. All rights reserved

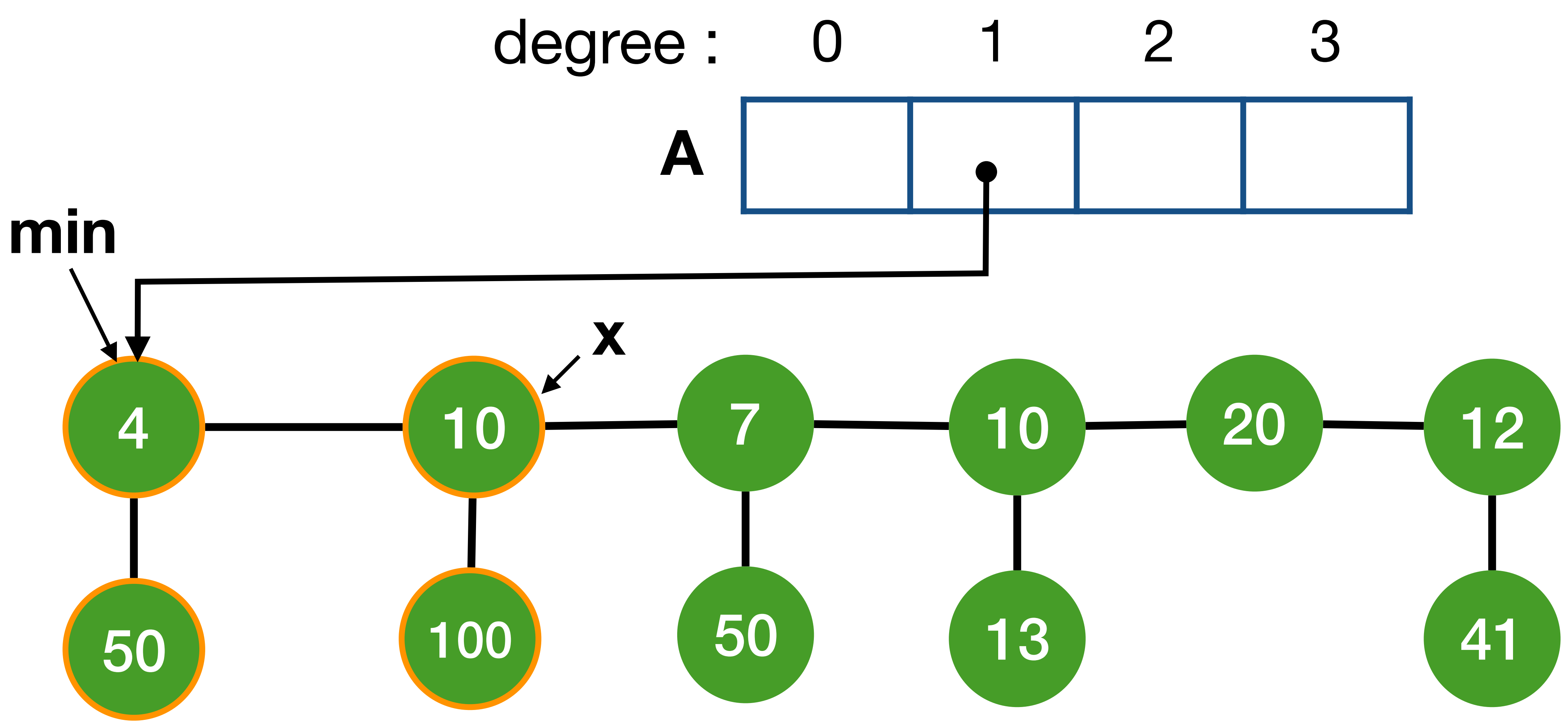
Fibonacci heap : delete min



Move x to next node

© SPEL Technologies, Inc. All rights reserved

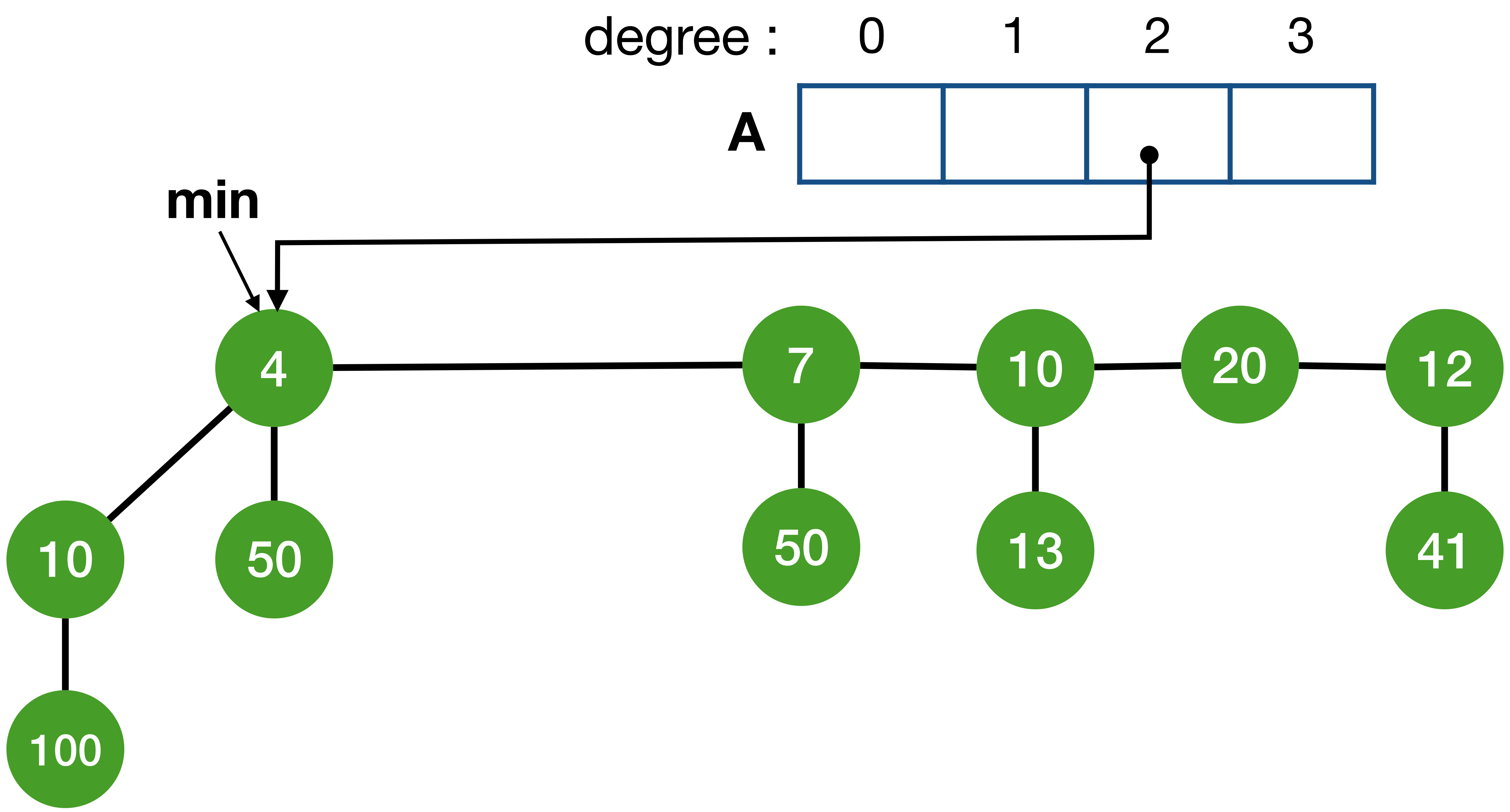
Fibonacci heap : delete min



10 has same degree as 4

 © SPEL Technologies, Inc. All rights reserved

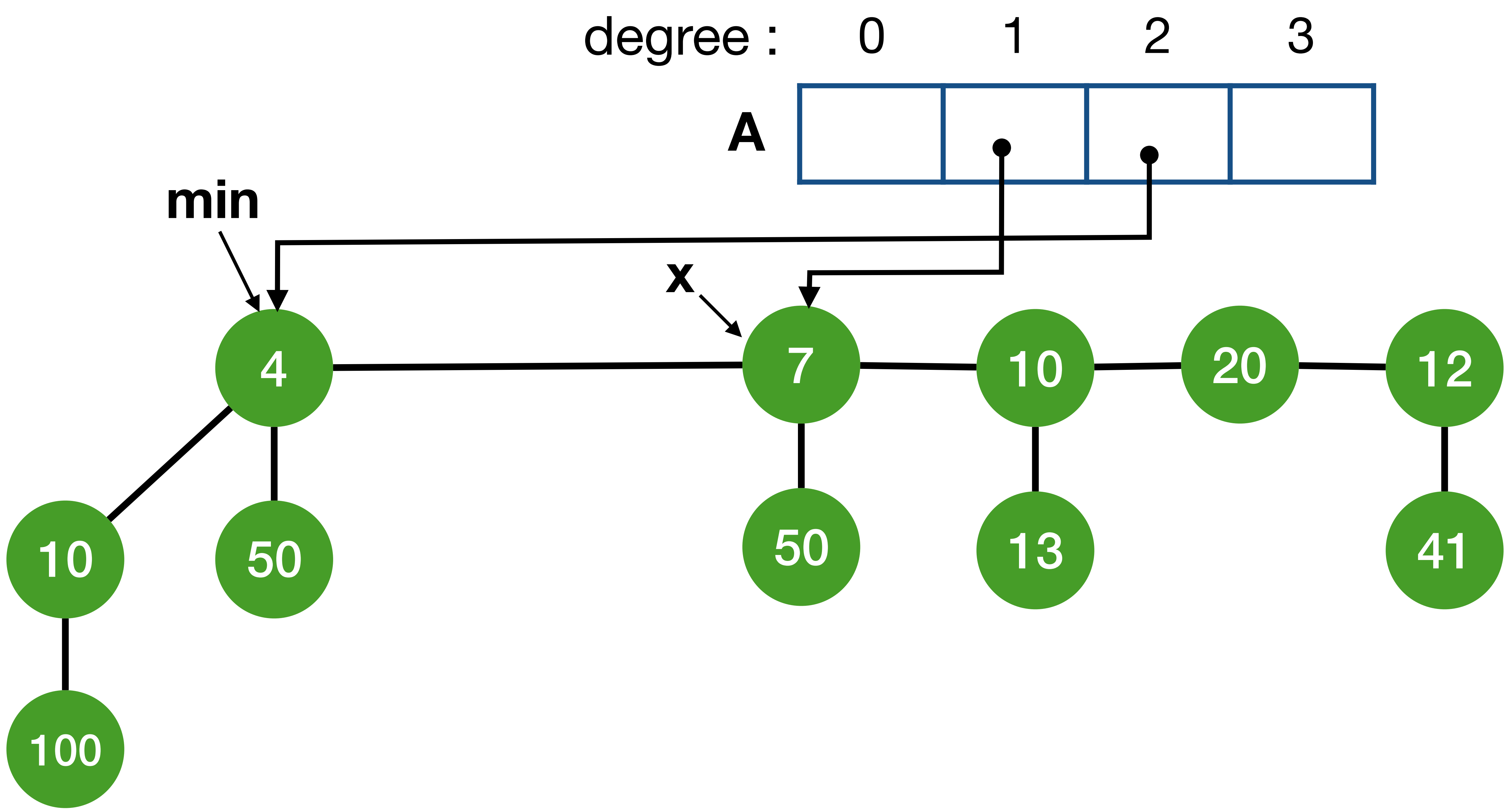
Fibonacci heap : delete min



Link 10 with 4 and update A

 © SPEL Technologies, Inc. All rights reserved

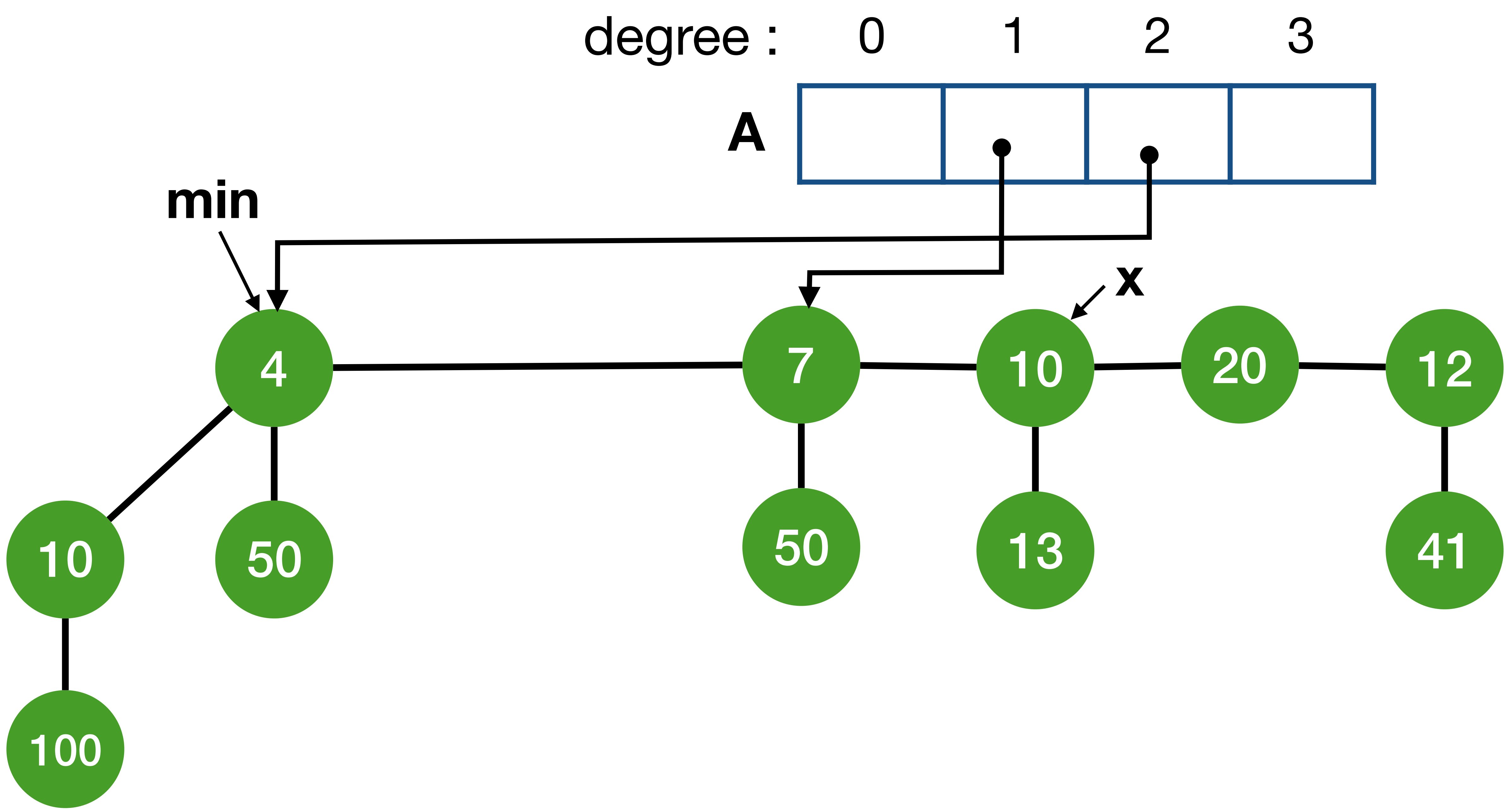
Fibonacci heap : delete min



Move x to next node, 7 has degree 1 update A

© SPEL Technologies, Inc. All rights reserved

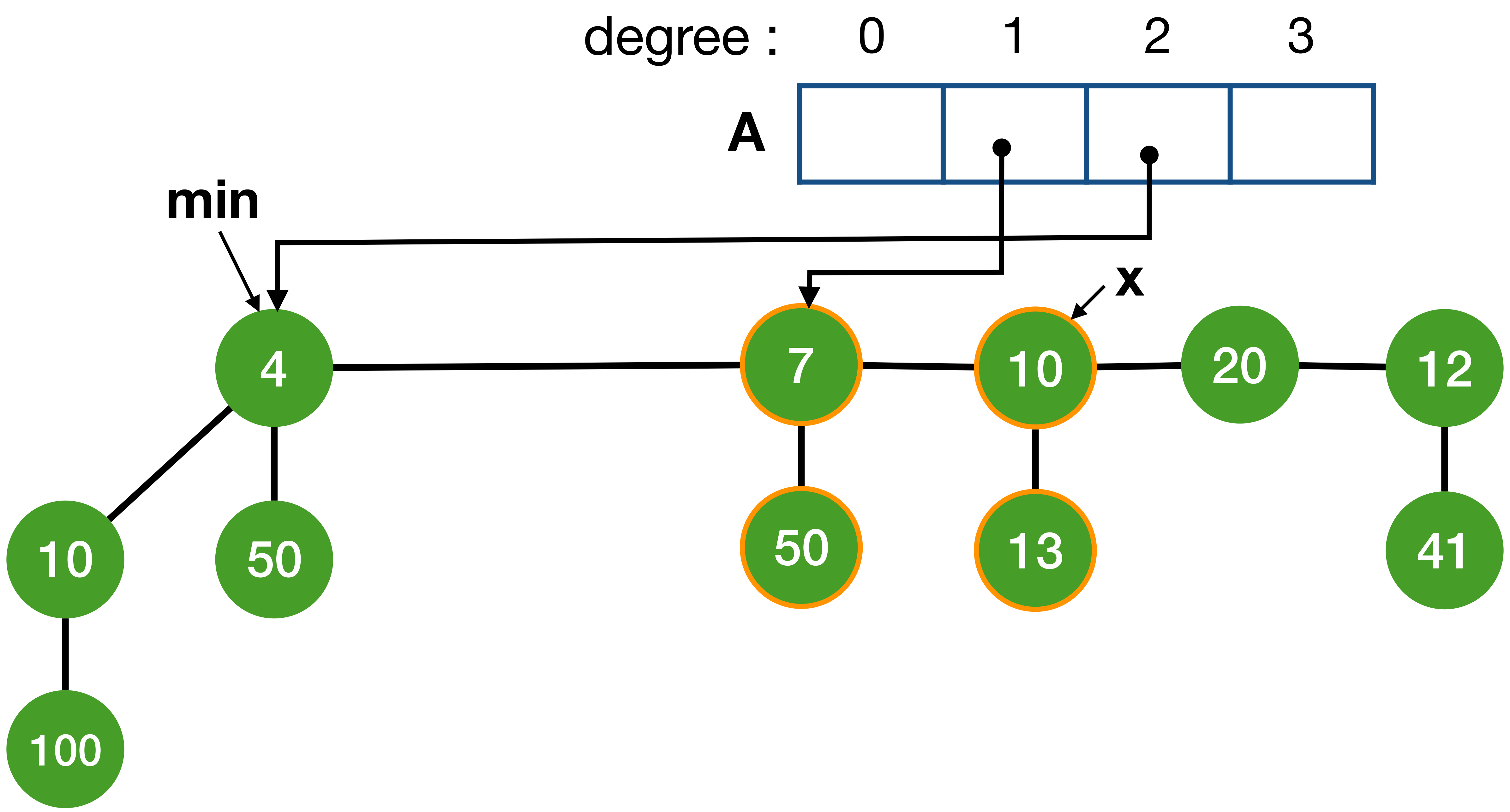
Fibonacci heap : delete min



Move x to next node

 © SPEL Technologies, Inc. All rights reserved

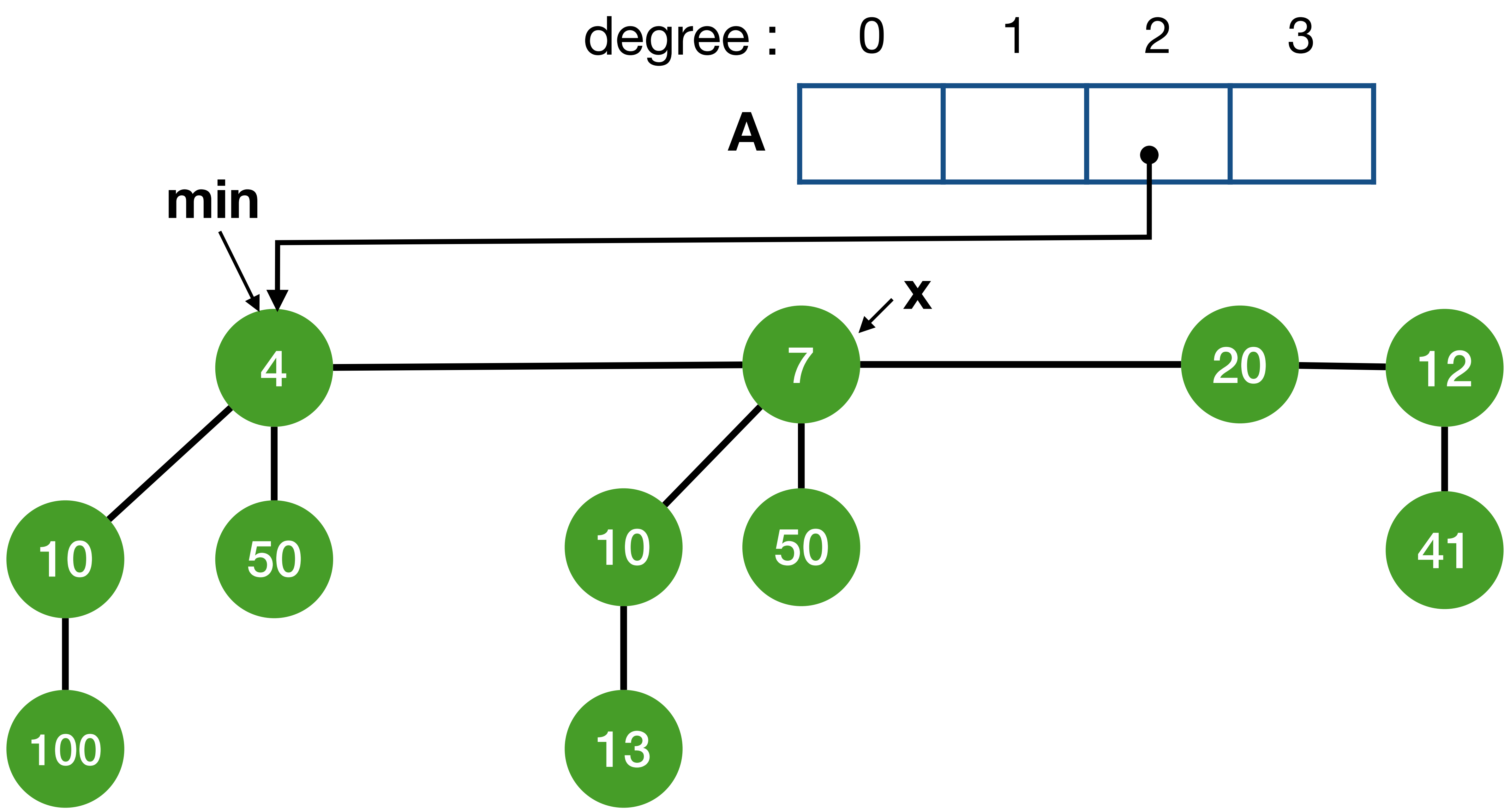
Fibonacci heap : delete min



10 has same degree as 7

© SPEL Technologies, Inc. All rights reserved

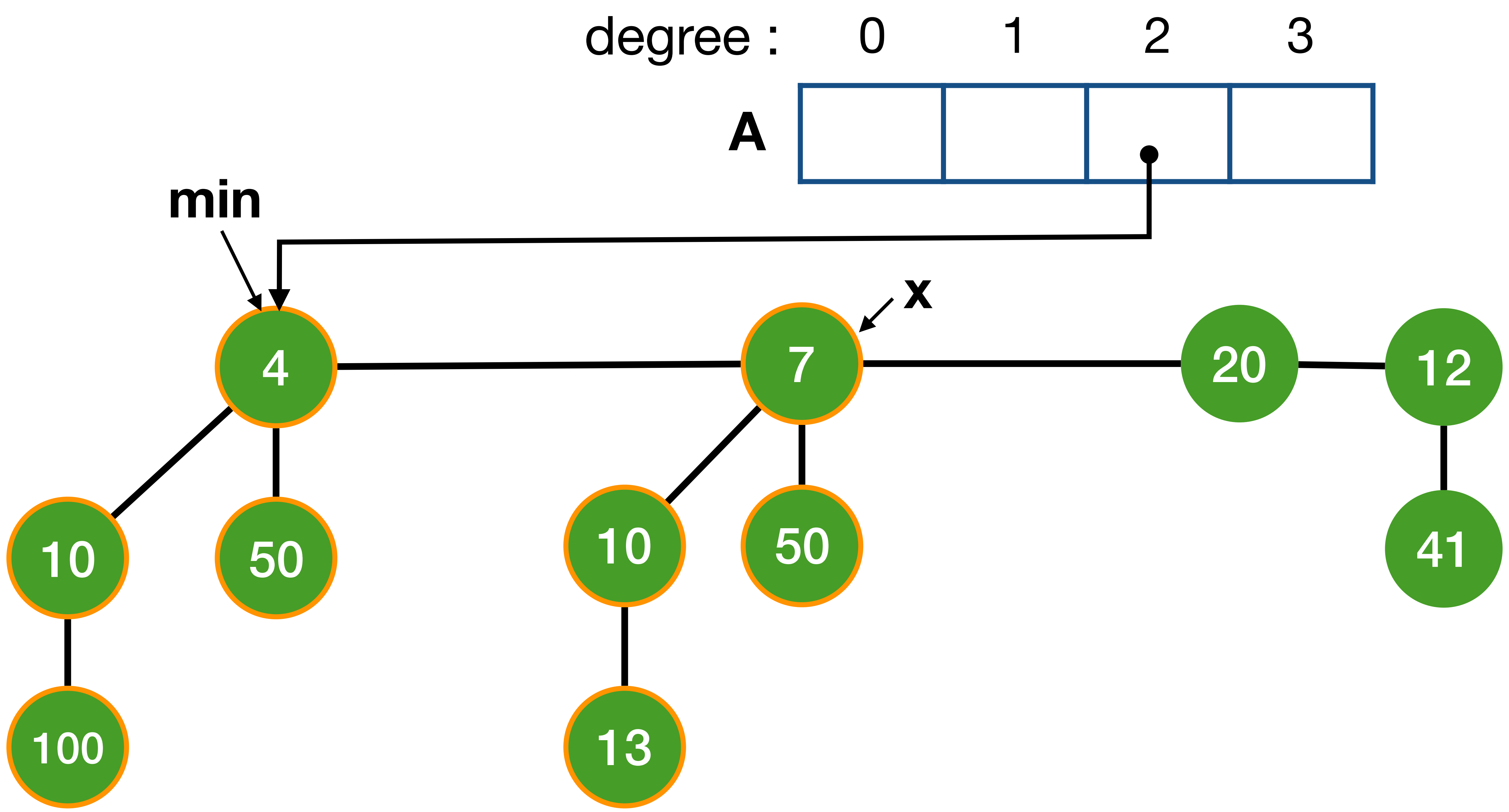
Fibonacci heap : delete min



Link 10 with 7

© SPEL Technologies, Inc. All rights reserved

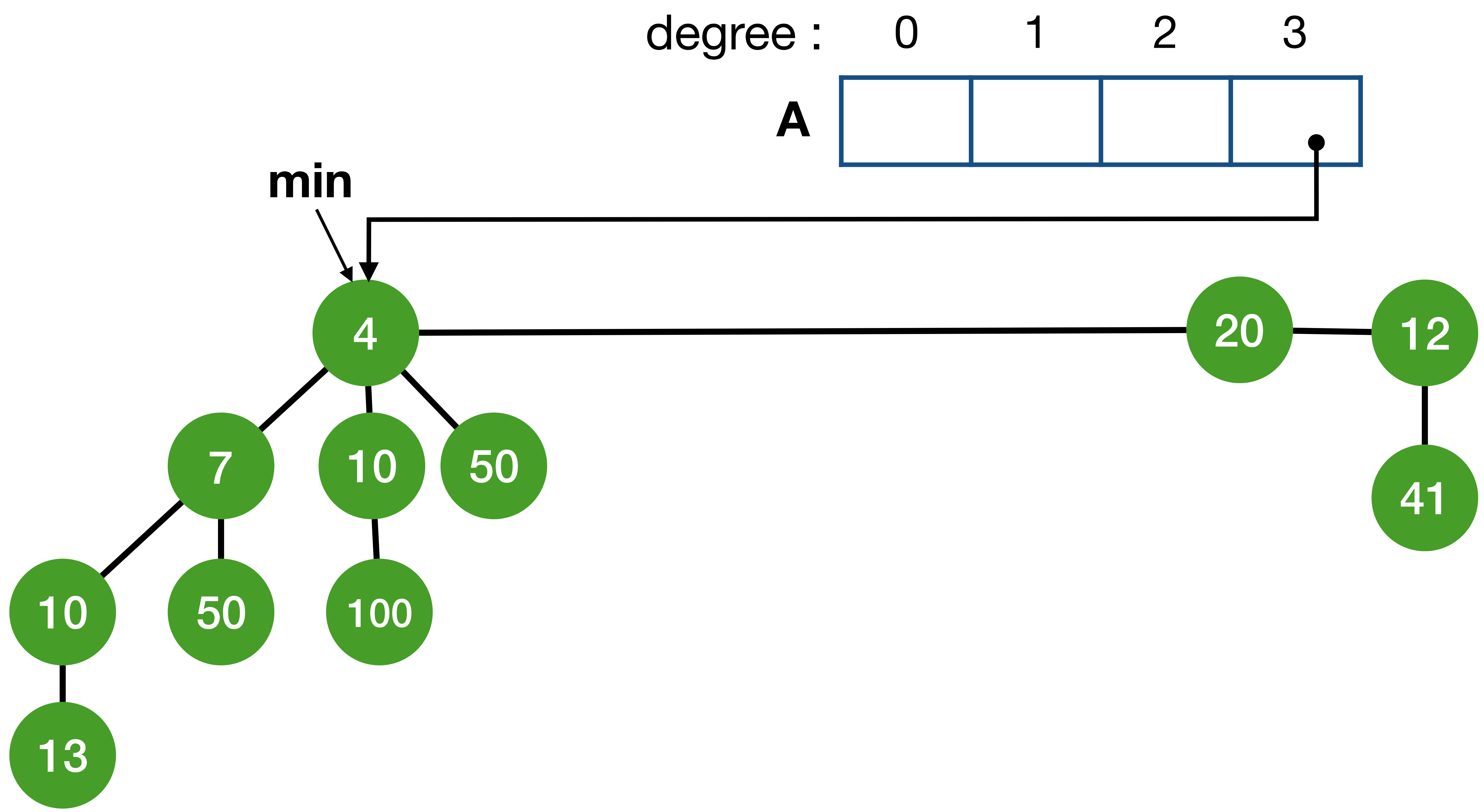
Fibonacci heap : delete min



7 has same degree as 4

© SPEL Technologies, Inc. All rights reserved

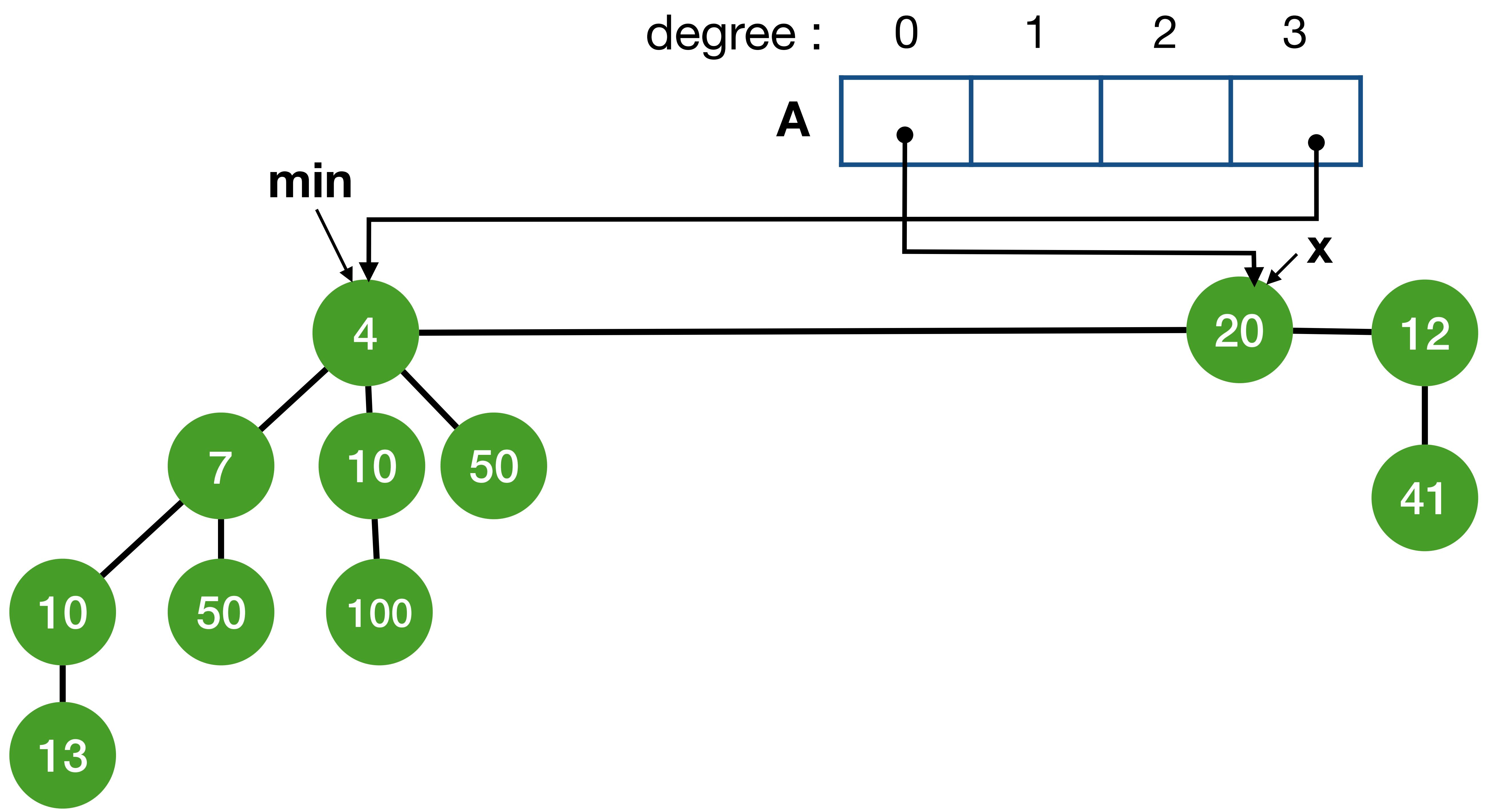
Fibonacci heap : delete min



Link 7 with 4 and update A

© SPEL Technologies, Inc. All rights reserved

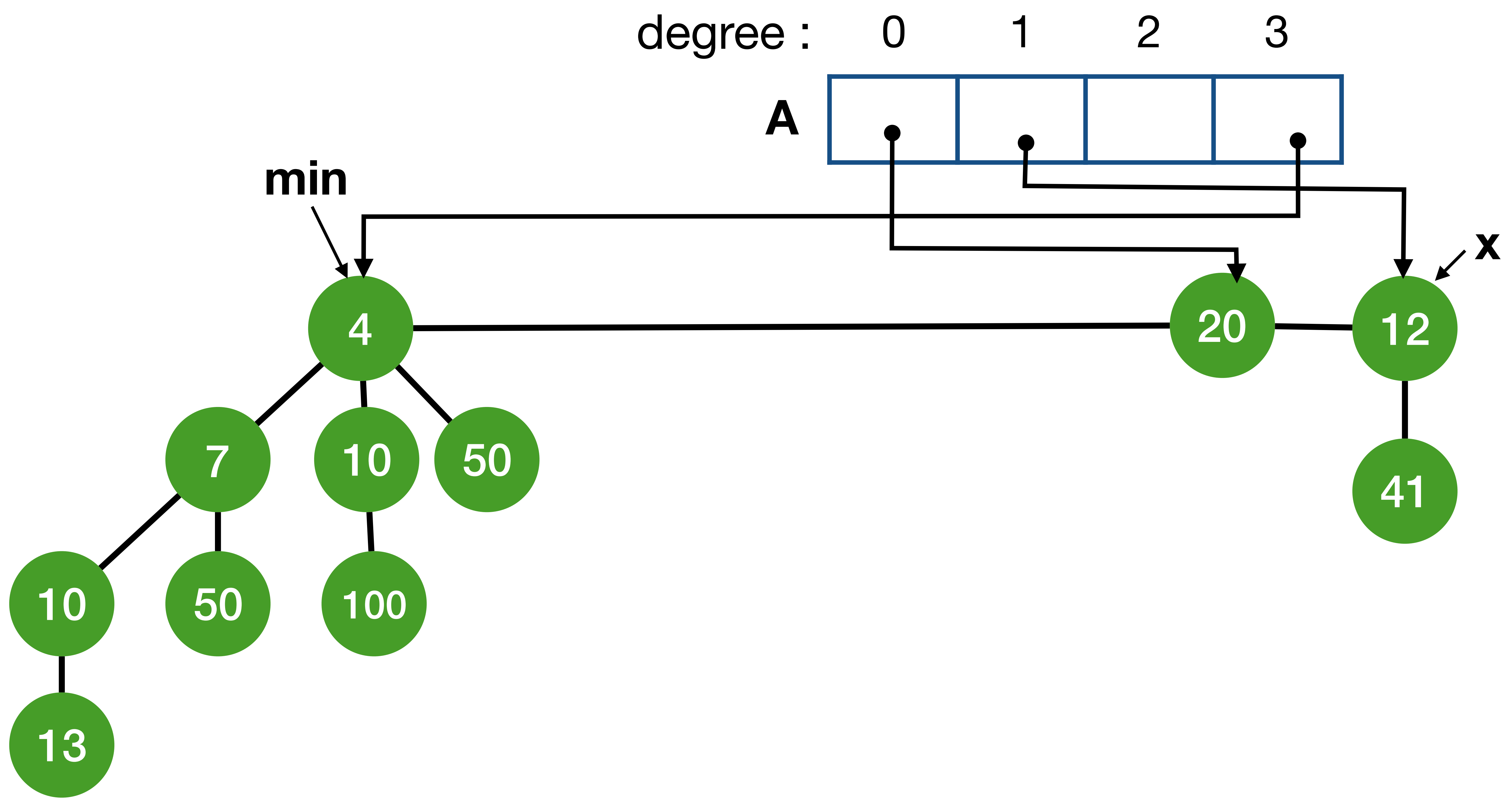
Fibonacci heap : delete min



Move x, 20 has degree 0 - update A

© SPEL Technologies, Inc. All rights reserved

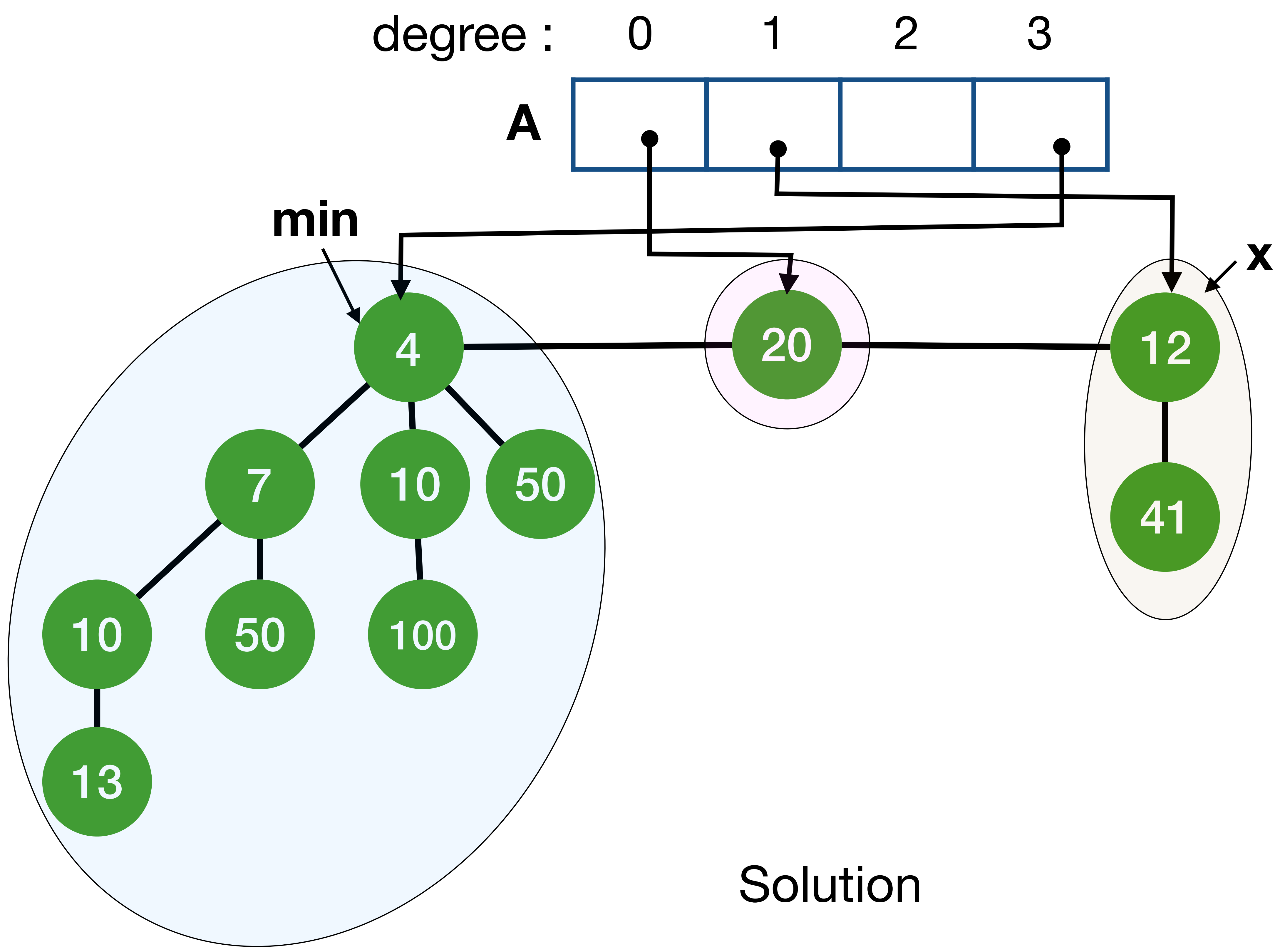
Fibonacci heap : delete min



Move x, 12 has degree 1- update A

© SPEL Technologies, Inc. All rights reserved

Fibonacci heap : delete min



Solution

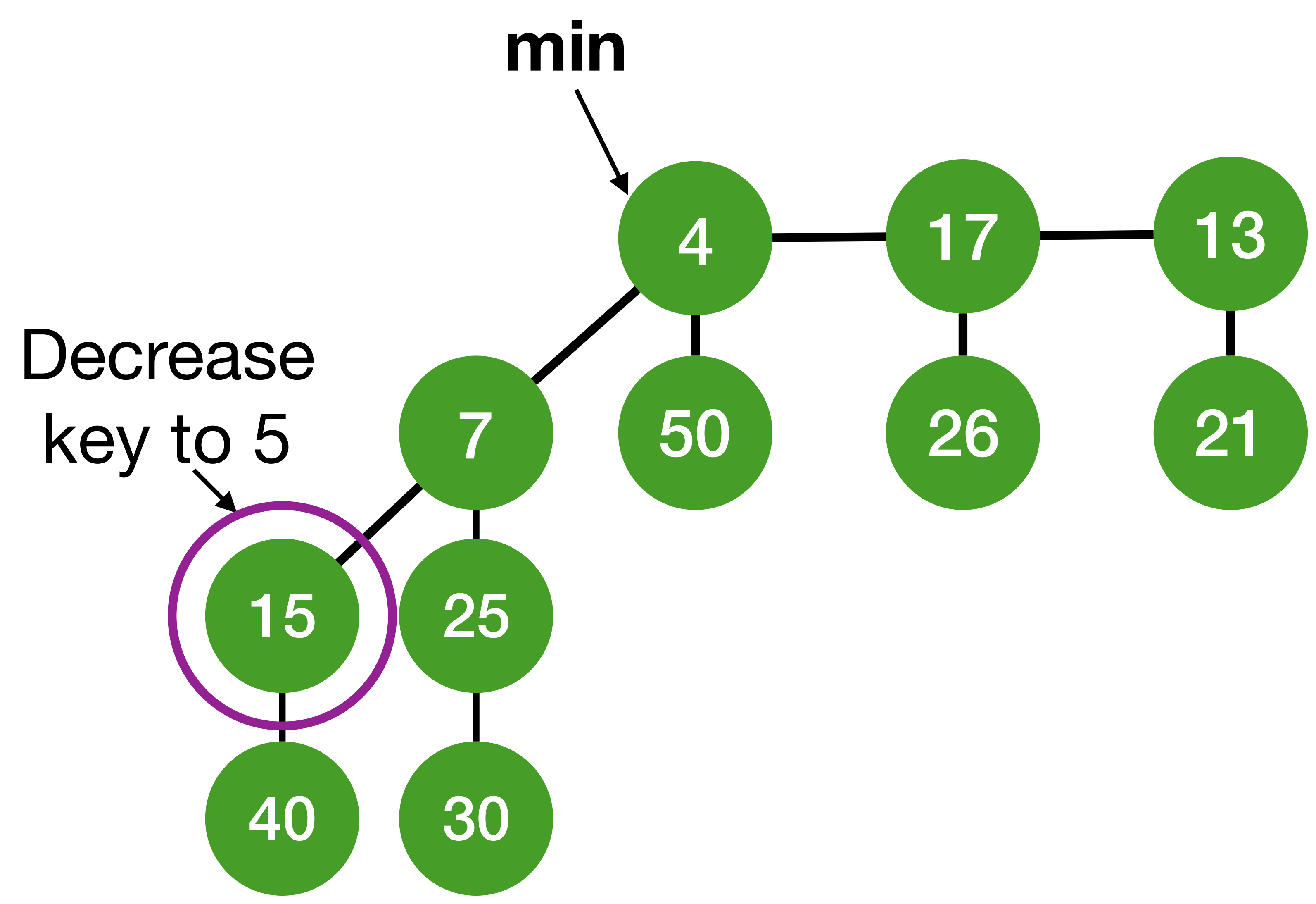
© SPEL Technologies, Inc. All rights reserved

Fibonacci heap :decrease key

- Decrease key of node x :
  - If heap order is violated :
    - Cut tree rooted at x and add to root list
    - If second child of x's parent p has been cut, then cut tree rooted at p and add to root list (cascading cut helps keep tree shape similar to binomial tree)
  - Otherwise, no change

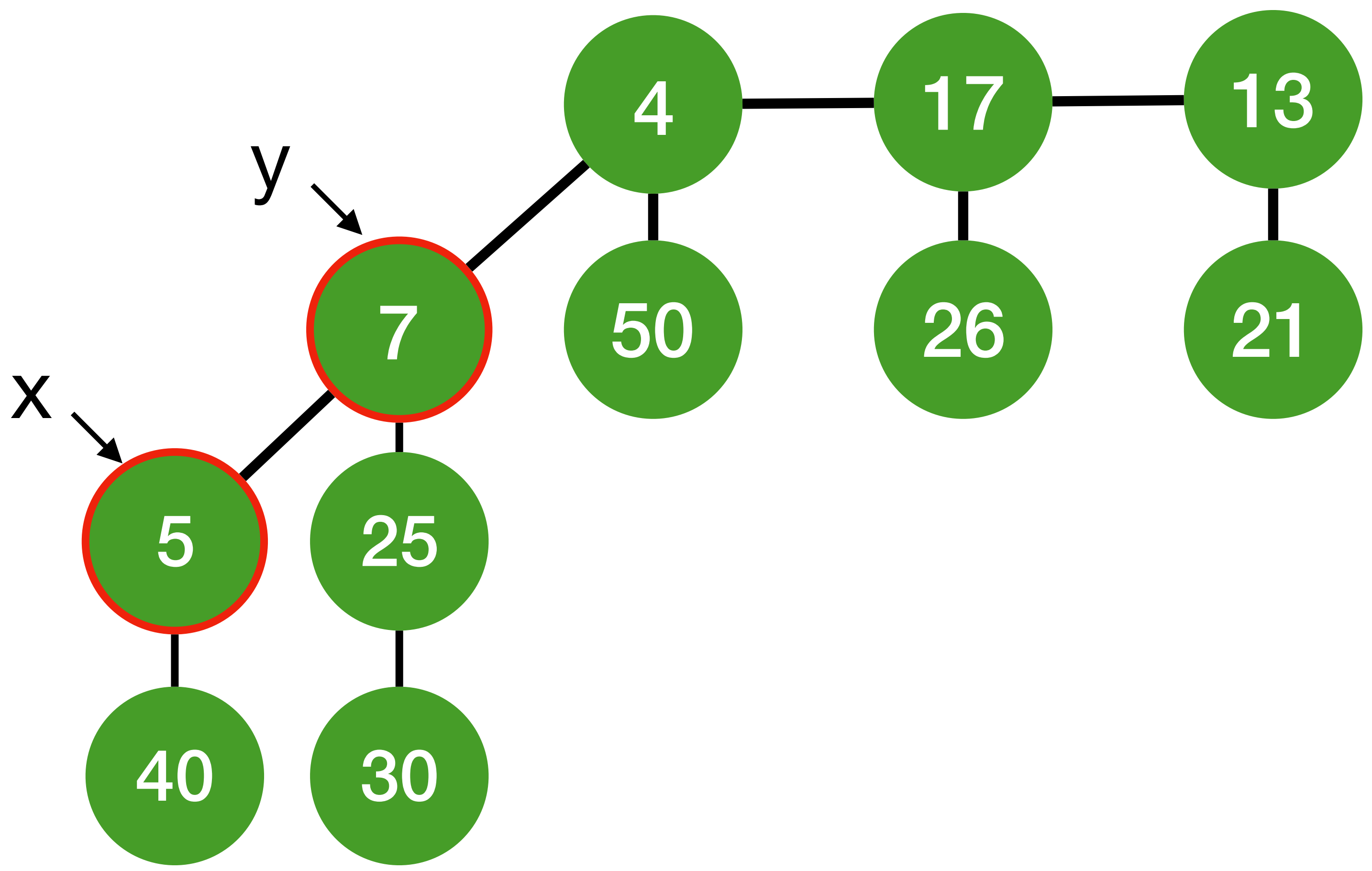
© SPEL Technologies, Inc. All rights reserved

Fibonacci heap :decrease key



© SPEL Technologies, Inc. All rights reserved

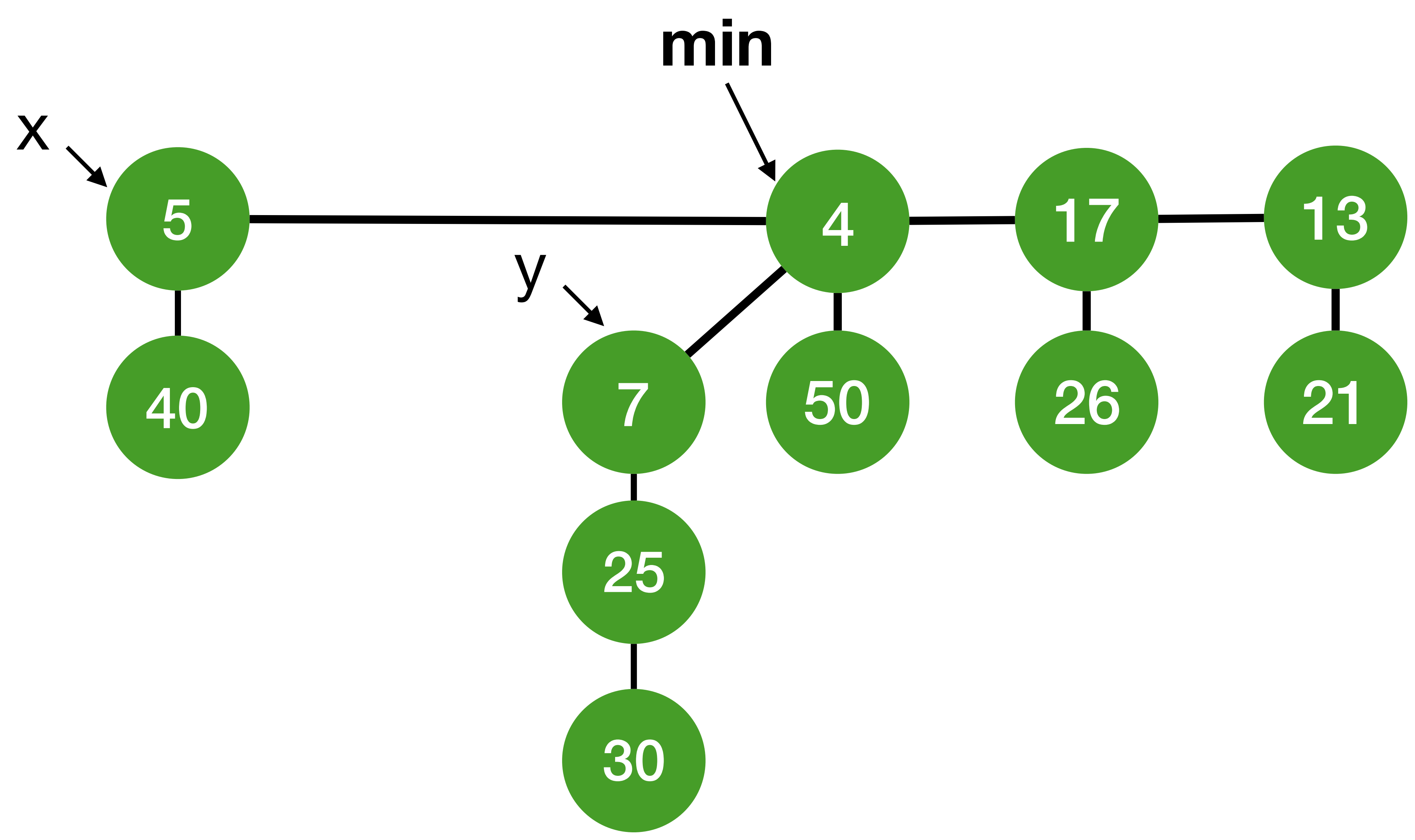
Fibonacci heap :decrease key



Key [x] < key[y] (heap order violated)

© SPEL Technologies, Inc. All rights reserved

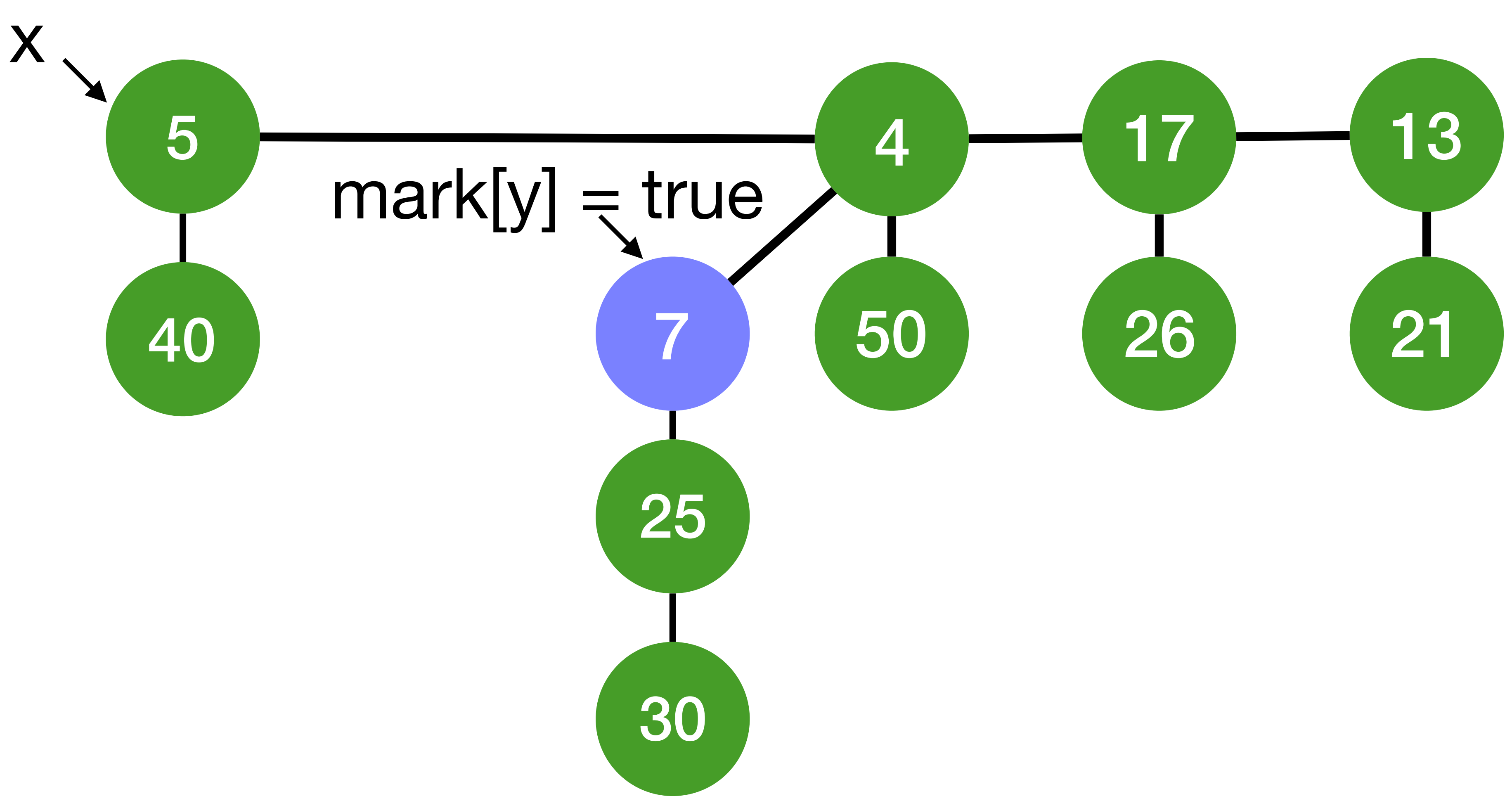
Fibonacci heap :decrease key



Cut x and add to root list

© SPEL Technologies, Inc. All rights reserved

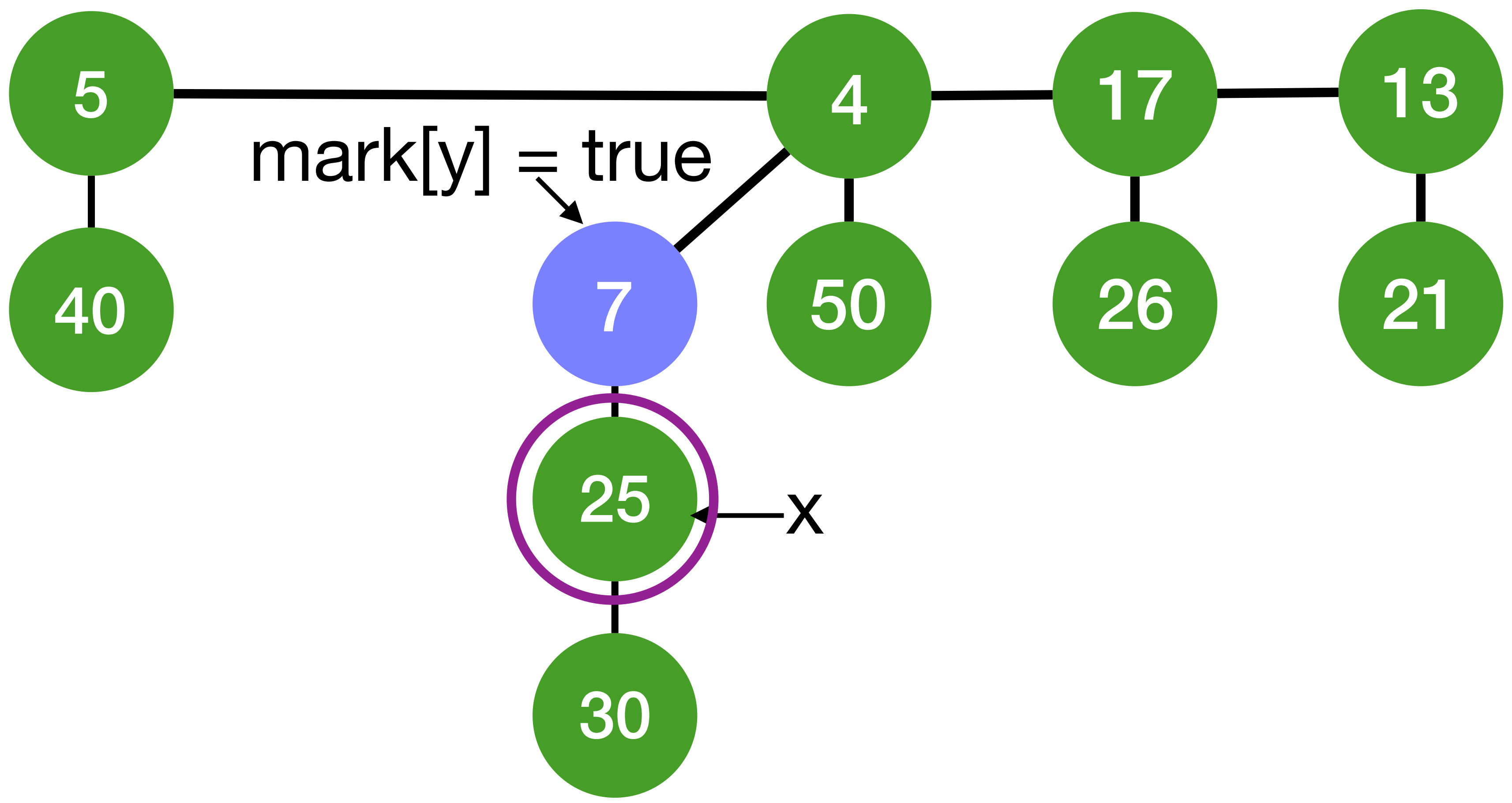
Fibonacci heap :decrease key



First child cut from y, set mark[y] = true

© SPEL Technologies, Inc. All rights reserved

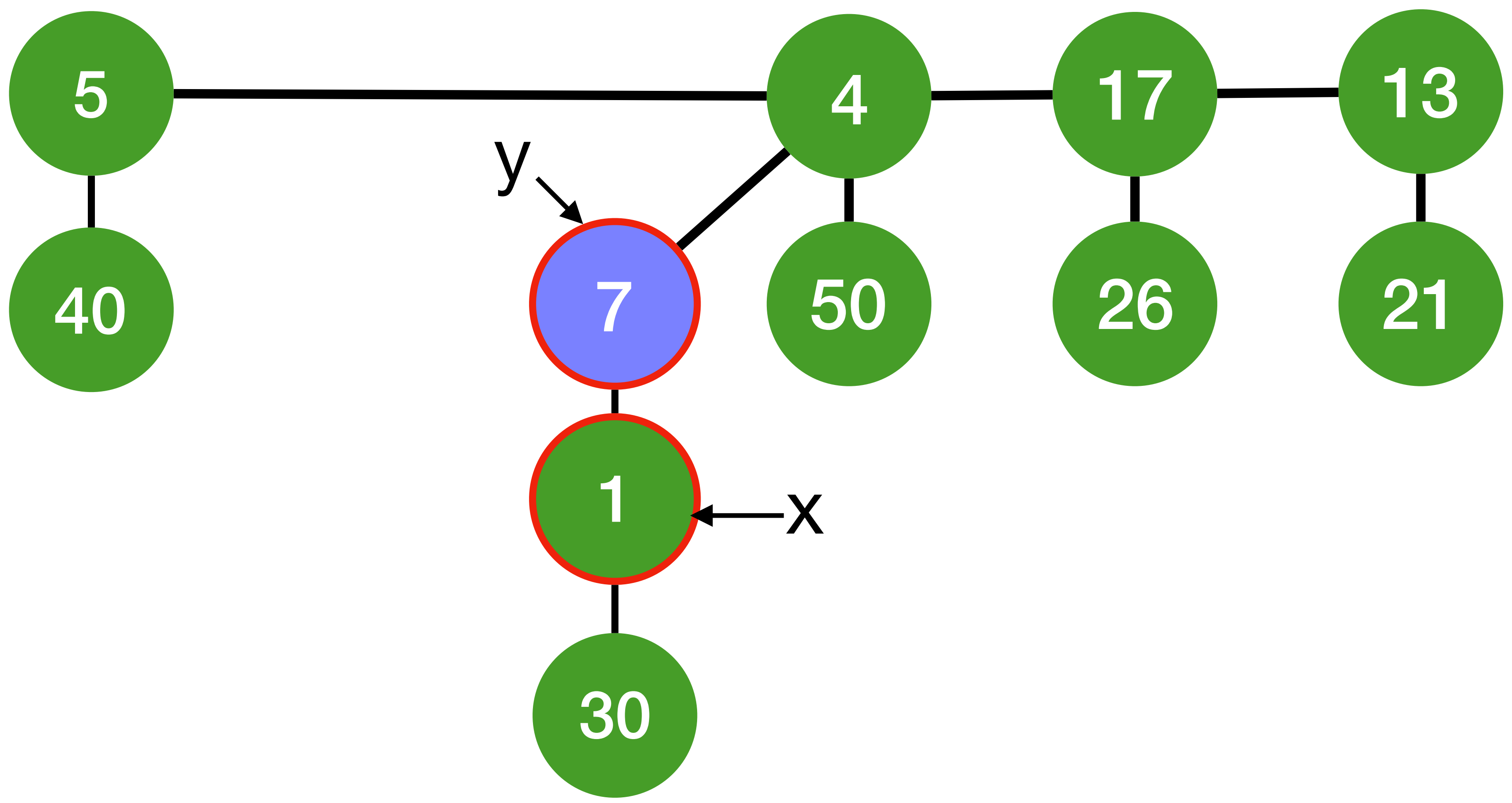
Fibonacci heap :decrease key



Decrease x to 1

© SPEL Technologies, Inc. All rights reserved

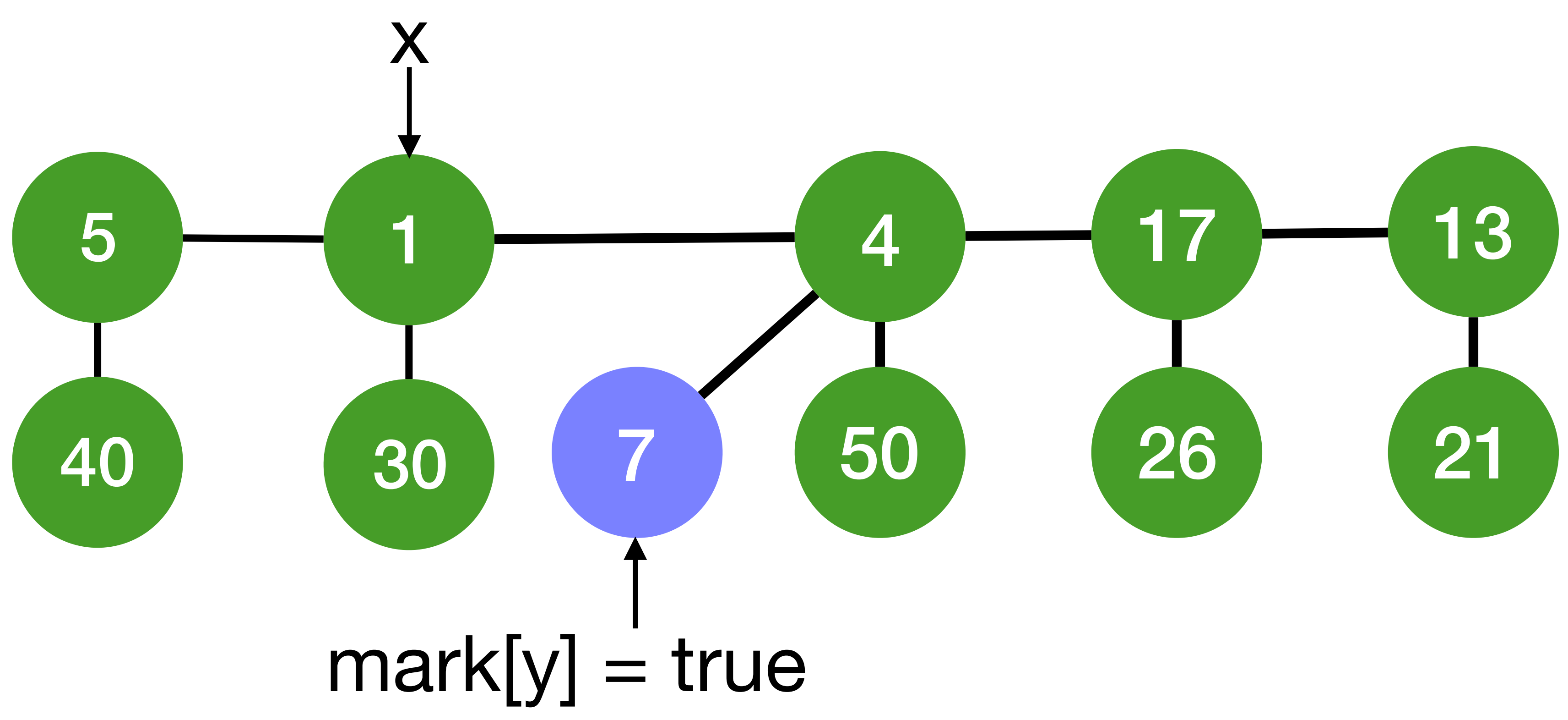
Fibonacci heap :decrease key



Key [x] < key[y] (heap order violated)

© SPEL Technologies, Inc. All rights reserved

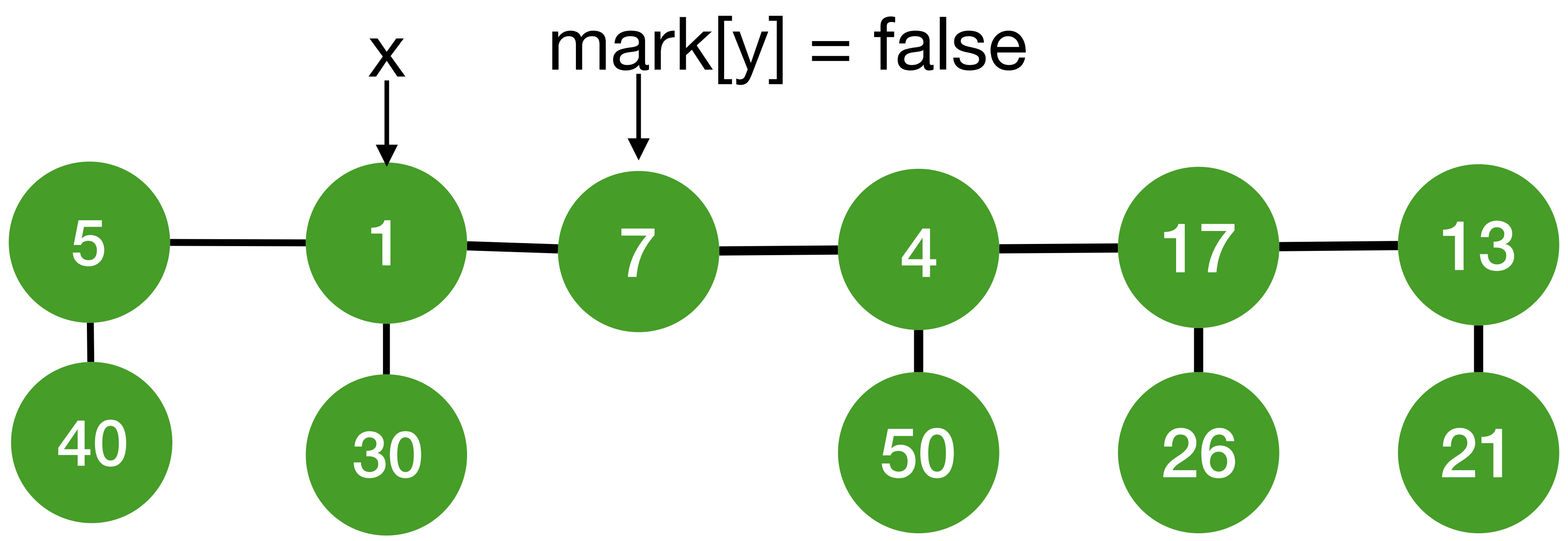
Fibonacci heap :decrease key



Cut x and add to root list
Second child cut from y

© SPEL Technologies, Inc. All rights reserved

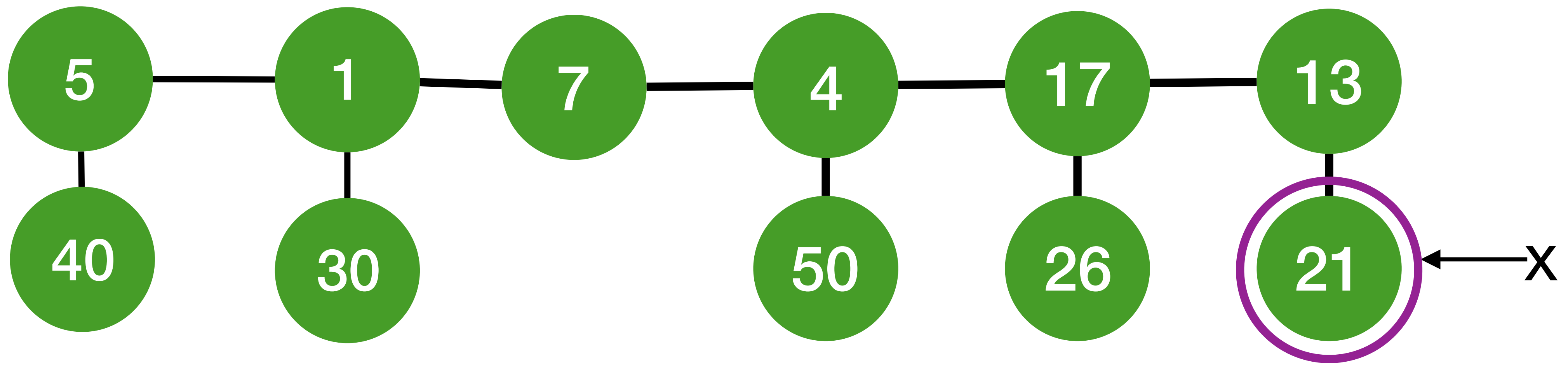
Fibonacci heap :decrease key



Cut y, add to root list and mark[y] = false
Continue this cascading cut until root is reached

 © SPEL Technologies, Inc. All rights reserved

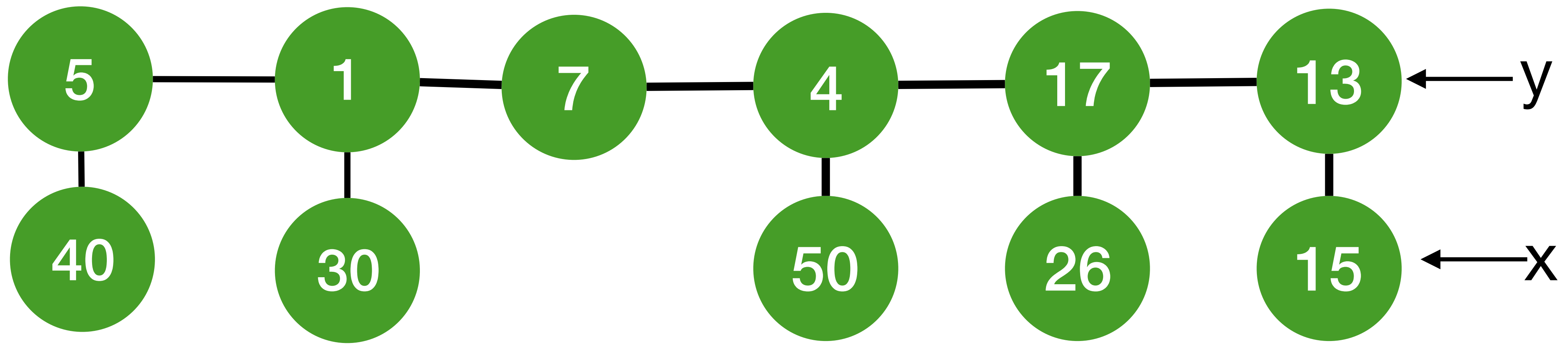
Fibonacci heap :decrease key



Decrease x to 15

© SPEL Technologies, Inc. All rights reserved

Fibonacci heap : decrease key



Decrease x to 15
Heap order not violated - no change

© SPEL Technologies, Inc. All rights reserved

Amortized analysis of decrease key

Actual cost : O(x), x = number of cascading cuts

Potential function = tree(H) + 2 * marks(H)

Amortized cost = actual cost + change in potential

tree(H') = tree(H) + x

marks(H') ≤ marks(H) - x + 1   (a cascading cut unmarks a node, last cascading cut marks a node)

Change in potential ≤ tree(H') + 2 * marks(H') - tree(H) - 2 * marks(H) = 2 - x

Actual cost = O(1)

Running times

Operation	Binary heap (worst-case)	Binomial heap (worst-case)	Fibonacci heap (amortized)
`find_min`	$\Theta(1)$	$\Theta(\log N)$	$\Theta(1)$
`insert`	$O(\log N)$	$\Theta(1)$	$\Theta(1)$
`merge`	$\Theta(N)$	$O(\log N)$	$\Theta(1)$
`delete_min`	$\Theta(\log N)$	$\Theta(\log N)$	$O(\log N)$
`decrease_key`	$\Theta(\log N)$	$\Theta(\log N)$	$\Theta(1)$
`make_heap`		$O(1)$	$O(1)$
`union`		$O(\log N)$	$O(1)$
`Minimum`		$O(\log N)$	$O(\log N)$

References

- Cormen, Thomas H., et al. *Introduction to Algorithms*. The MIT Press, 2014.

 © SPEL Technologies, Inc. All rights reserved