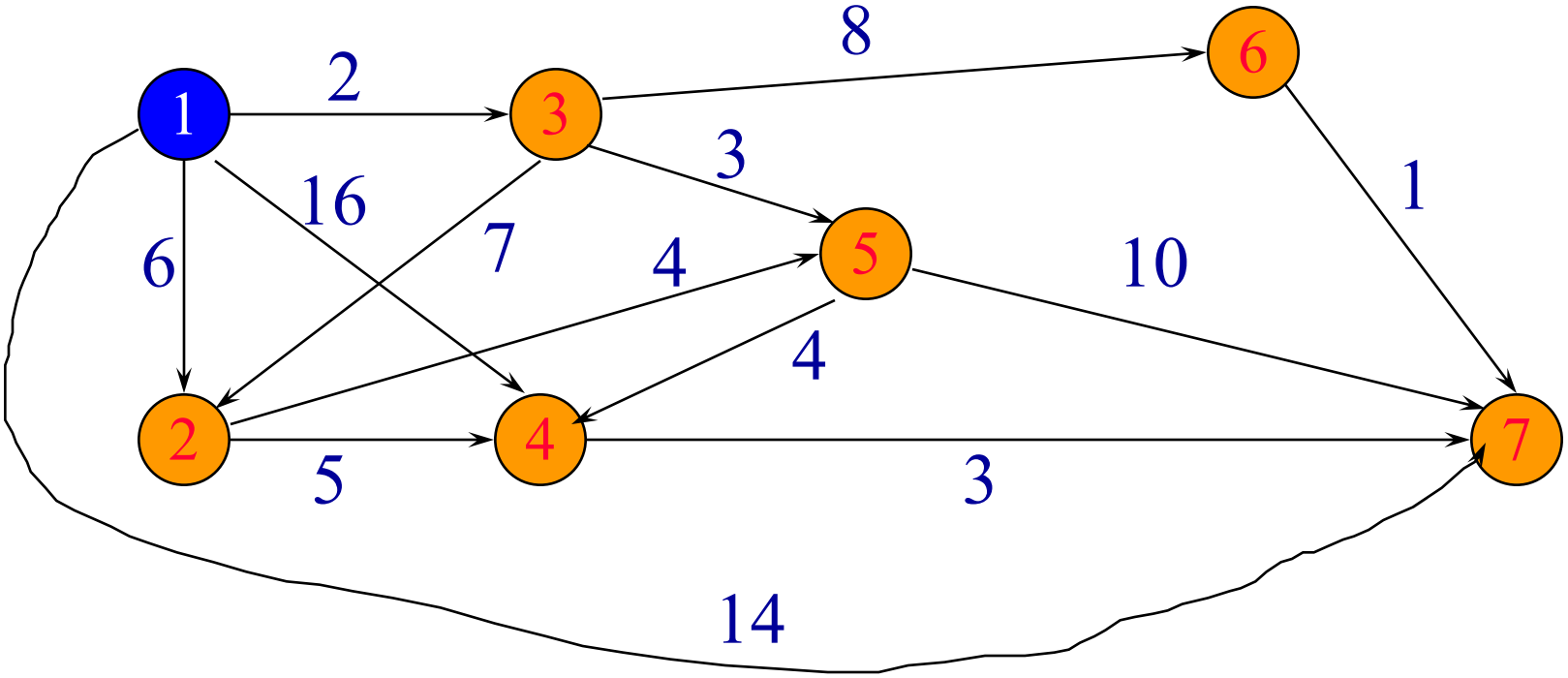
Fibonacci Heaps

	Actual	Amortized
Insert	O(1)	O(1)
Remove min (or max)	O(n)	O(log n)
Meld	O(1)	O(1)
Remove	O(n)	O(log n)
Decrease key (or increase)	O(n)	O(1)

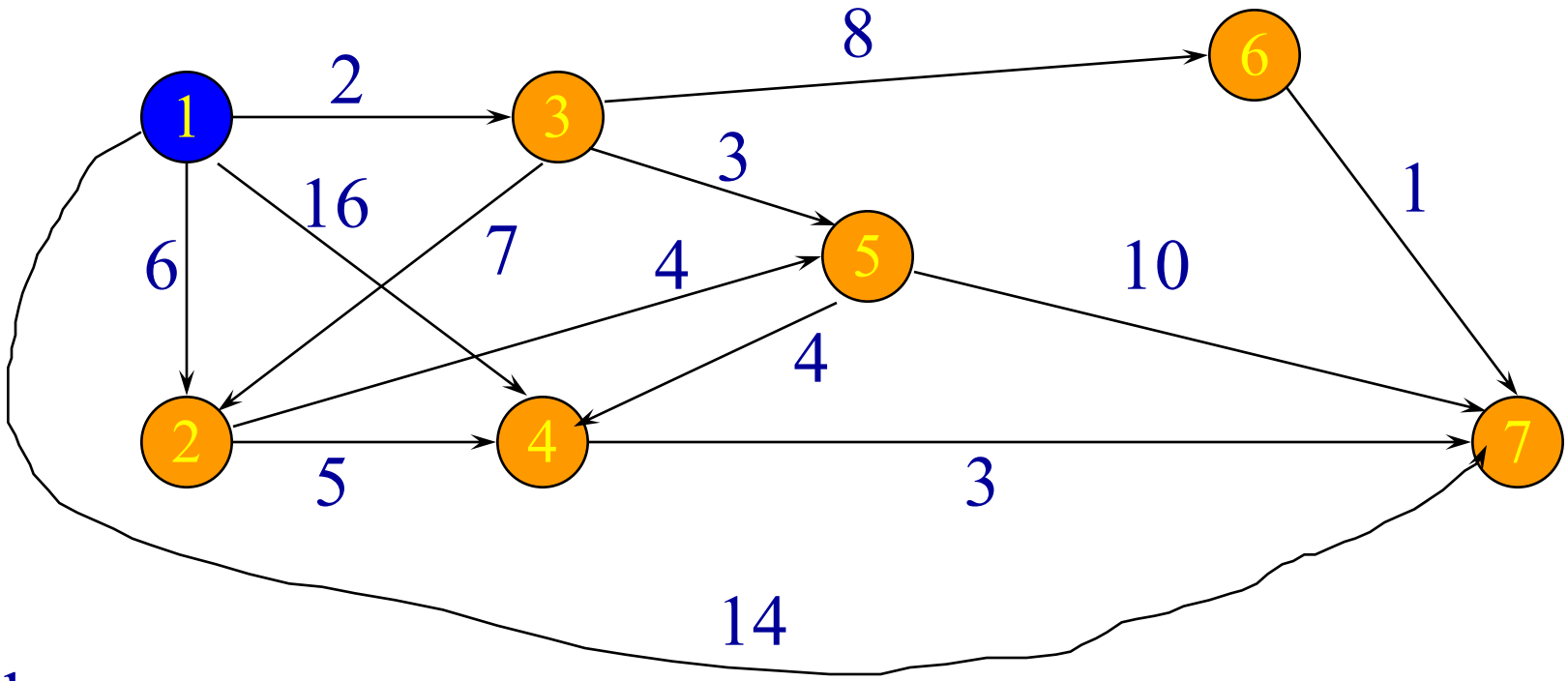
Single Source All Destinations Shortest Paths



Greedy Single Source All Destinations

- Known as Dijkstra's algorithm.
- Let d(i) be the length of a shortest one edge extension of an already generated shortest path, the one edge extension ends at vertex i.
- The next shortest path is to an as yet unreached vertex for which the d() value is least.
- After the next shortest path is generated, some d() values are updated (decreased).

Greedy Single Source All Destinations



Path	Length
1	0
1 → 3	2
1 → 3 → 5	5
1 → 2	6
1 → 3 → 5 → 4	9
1 → 3 → 6	10
1 → 3 → 6 → 7	11

Operations On d()

- Remove min.
  - Done O(n) times, where n is the number of vertices in the graph.
- Decrease d().
  - Done O(e) times, where e is the number of edges in the graph.
- Array.
  - $O(n^2)$ overall complexity.
- Min heap.
  - O(nlog n + elog n) overall complexity.
- Fibonacci heap.
  - O(nlog n + e) overall complexity.

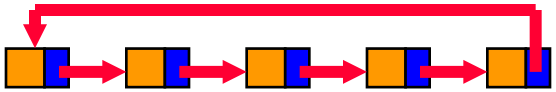
Prim's Min-Cost Spanning Tree Algorithm

- Array.
  - $O(n^2)$ overall complexity.
- Min heap.
  - $O(n \log n + e \log n)$ overall complexity.
- Fibonacci heap.
  - $O(n \log n + e)$ overall complexity.

Min Fibonacci Heap

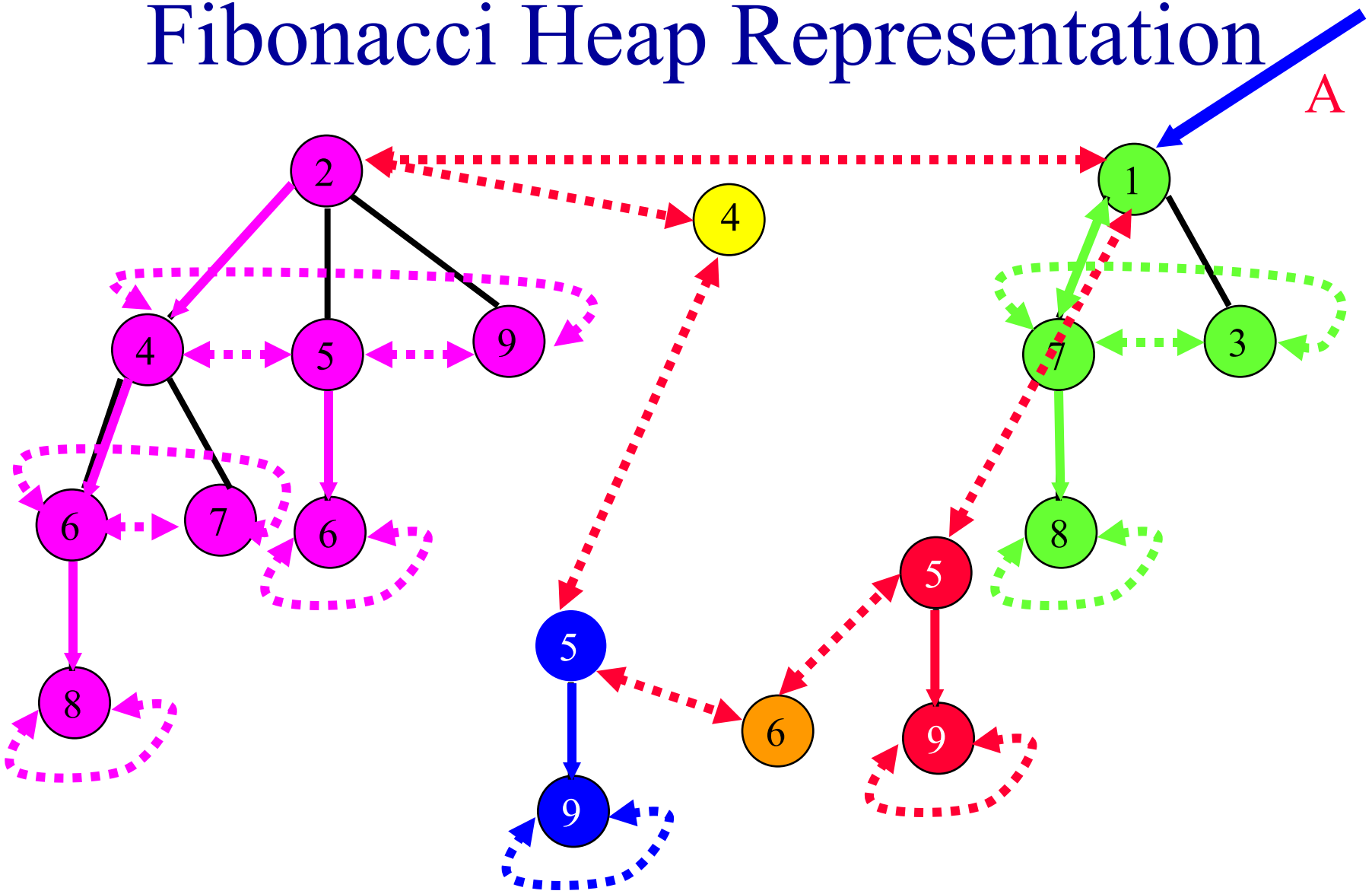
- Collection of min trees.
- The min trees need not be Binomial trees.

Node Structure



- Degree, Child, Data
- Left and Right Sibling
  - Used for circular doubly linked list of siblings.
- Parent
  - Pointer to parent node.
- ChildCut
  - True if node has lost a child since it became a child of its current parent.
  - Set to false by remove min, which is the only operation that makes one node a child of another.
  - Undefined for a root node.

Fibonacci Heap Representation



- Degree, Parent and ChildCut fields not shown.

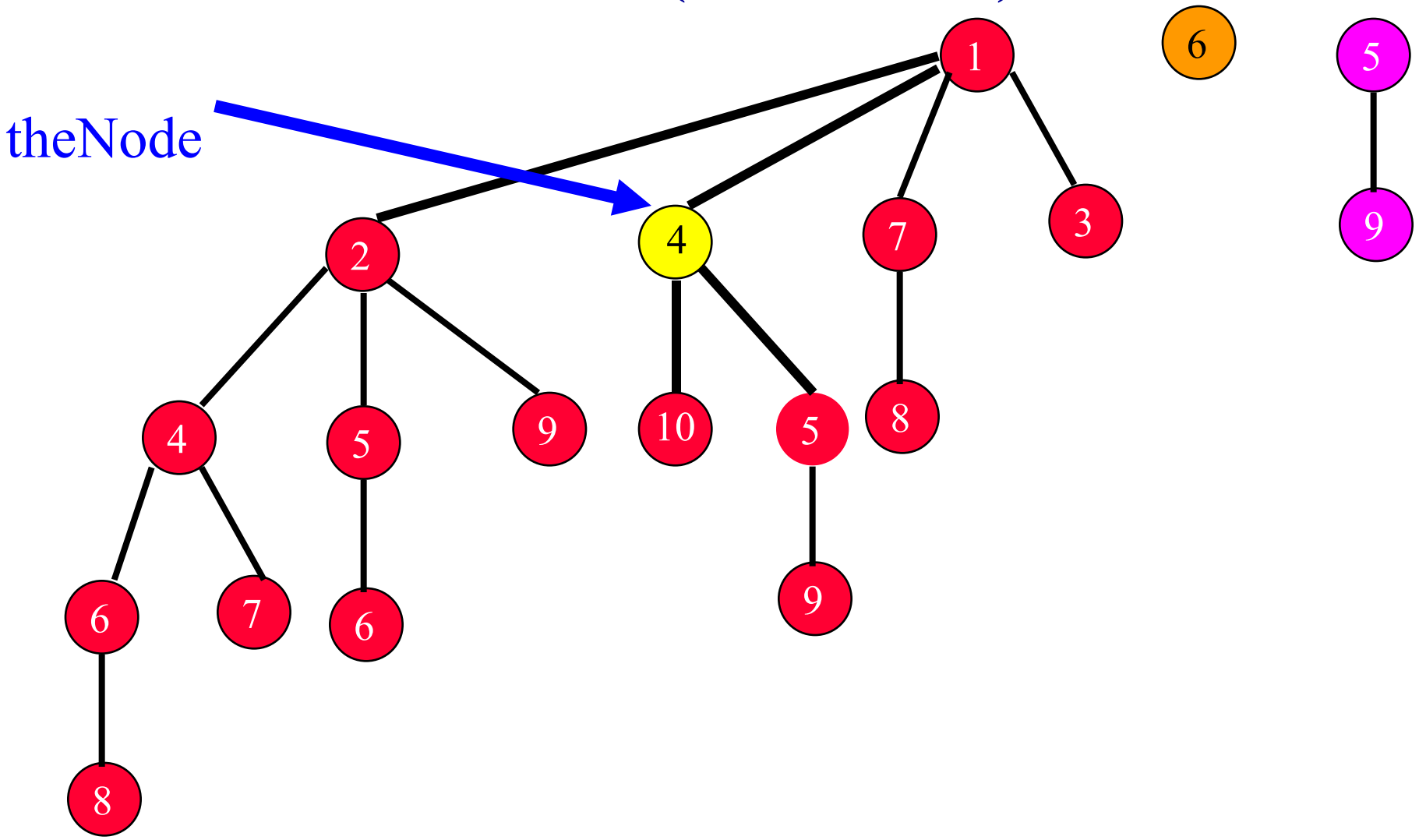
Remove(theNode)

- theNode points to the Fibonacci heap node that contains the element that is to be removed.
- theNode points to min element => do a remove min.
  - In this case, complexity is the same as that for remove min.

Remove(theNode)

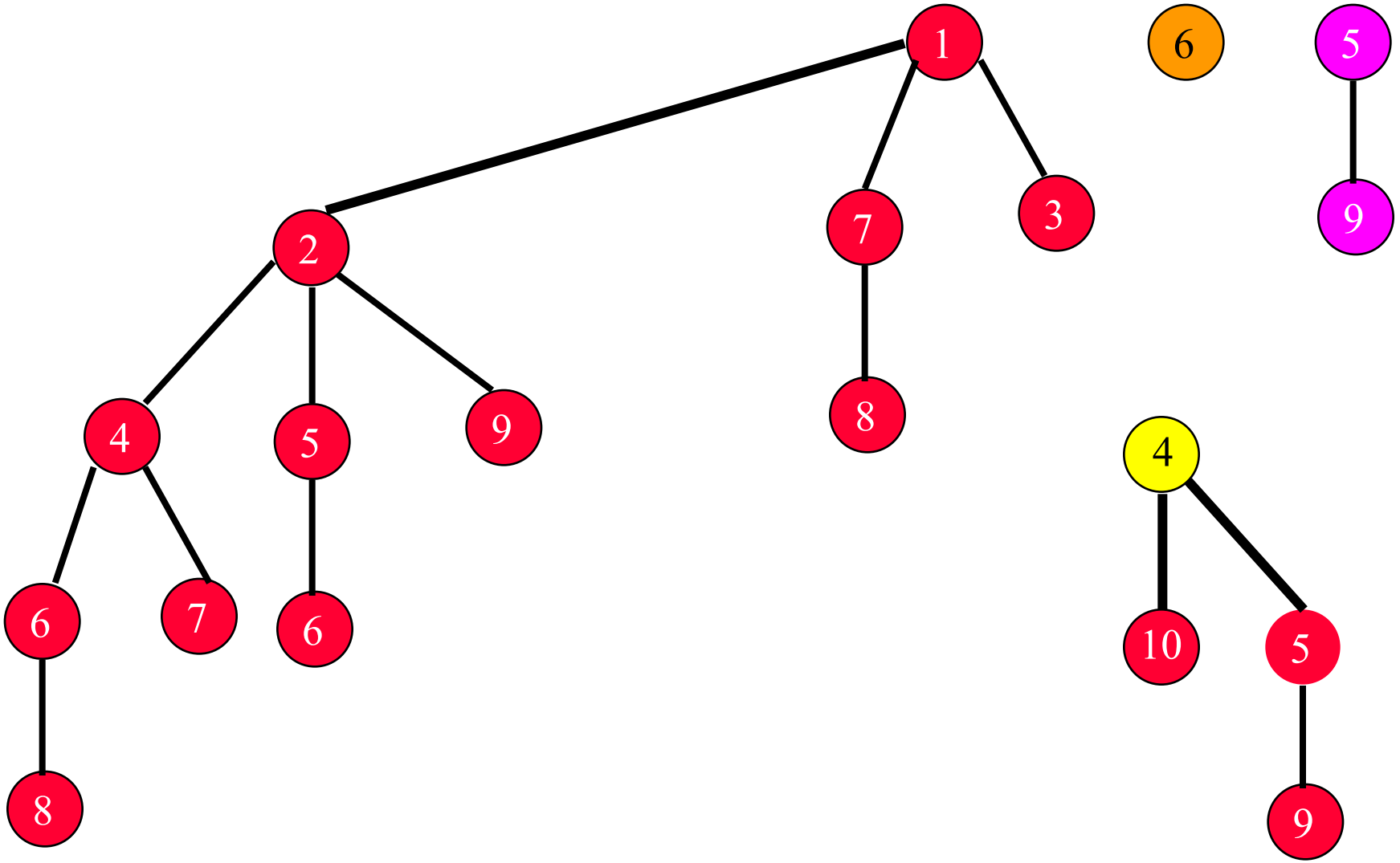
- theNode points to an element other than the min element.
  - Remove theNode from its doubly linked sibling list.
  - Change parent's child pointer if necessary.
  - Set parent field of theNode's children to null.
  - Combine top-level list and children list of theNode; do not pairwise combine equal degree trees.
  - Free theNode.
- In this case, actual complexity is O(log n) (assuming theNode has O(log n) children).

Remove(theNode)



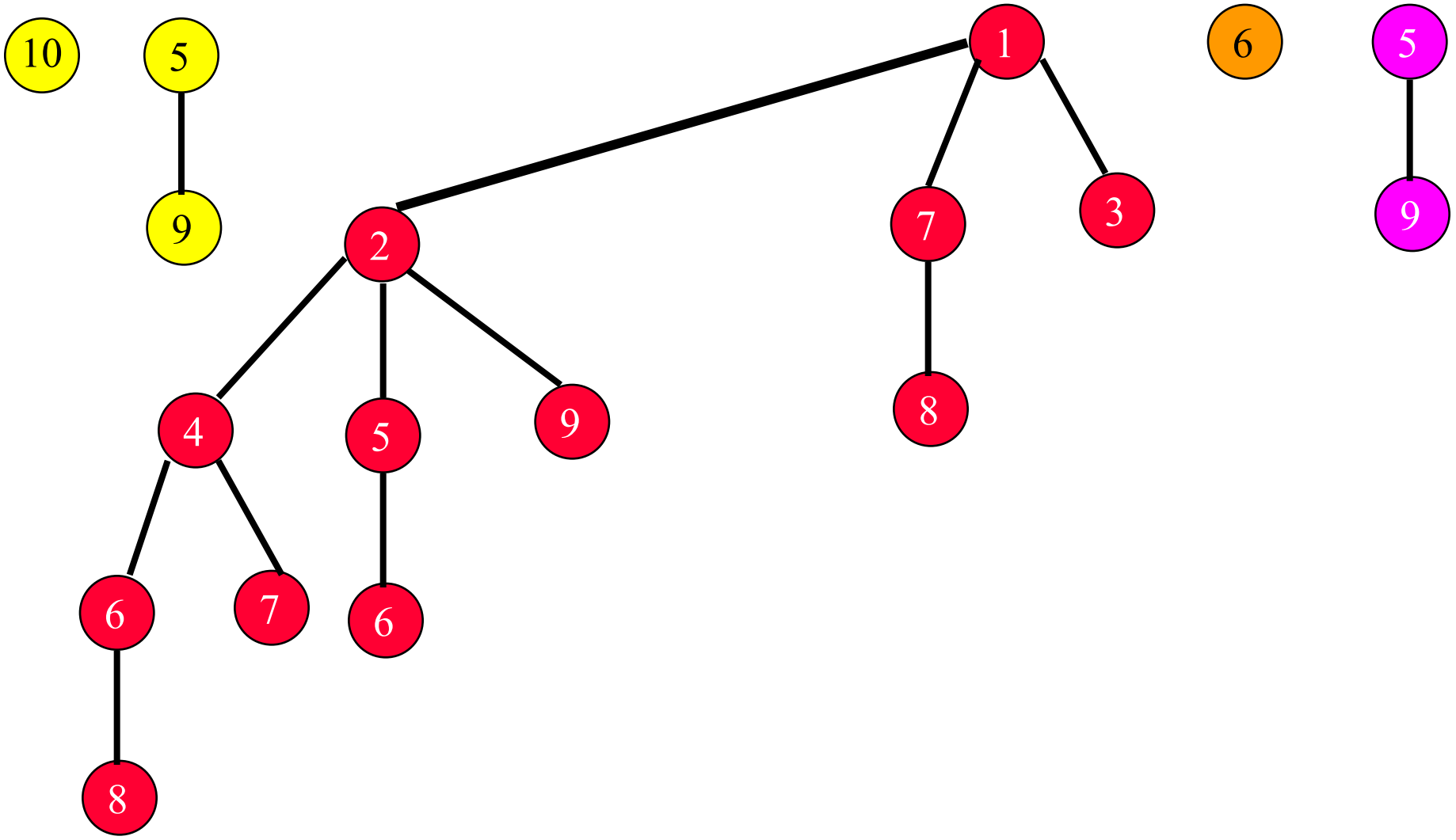
Remove theNode from its doubly linked sibling list.

Remove(theNode)

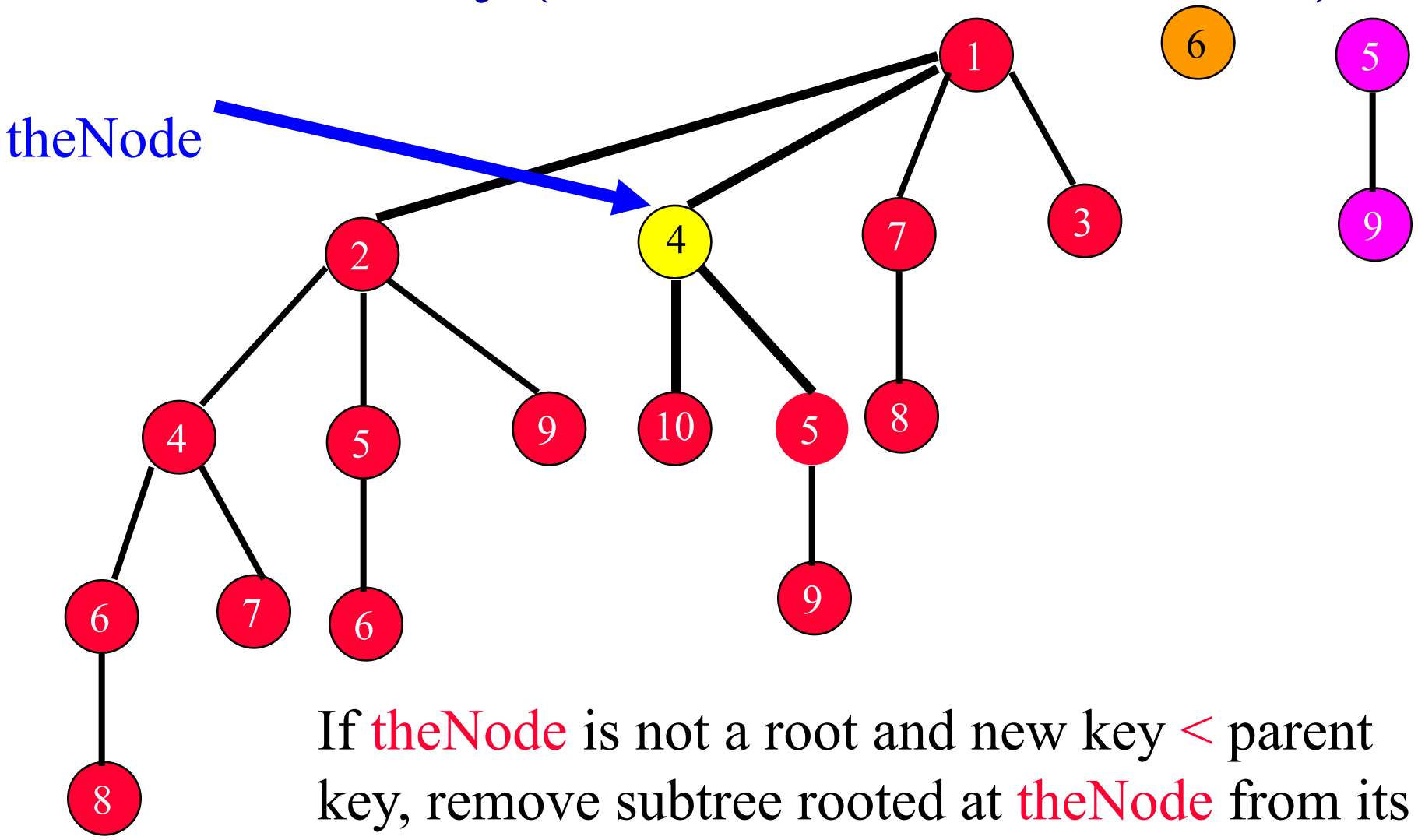


Combine top-level list and children of theNode setting parent pointers of the children of theNode to null.

Remove(theNode)



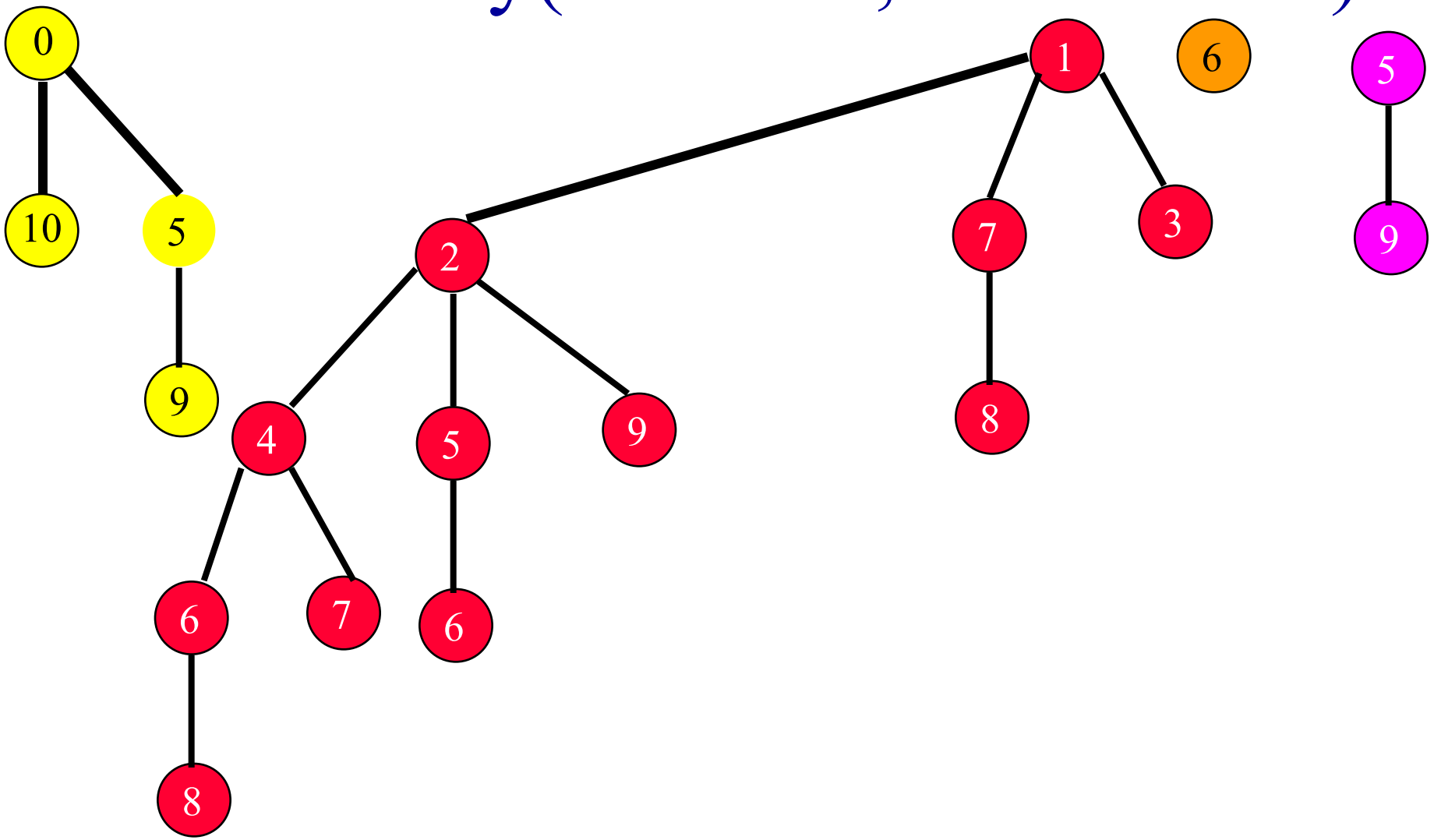
DecreaseKey(theNode, theAmount)



If theNode is not a root and new key < parent key, remove subtree rooted at theNode from its doubly linked sibling list.

Insert into top-level list.

DecreaseKey(theNode, theAmount)

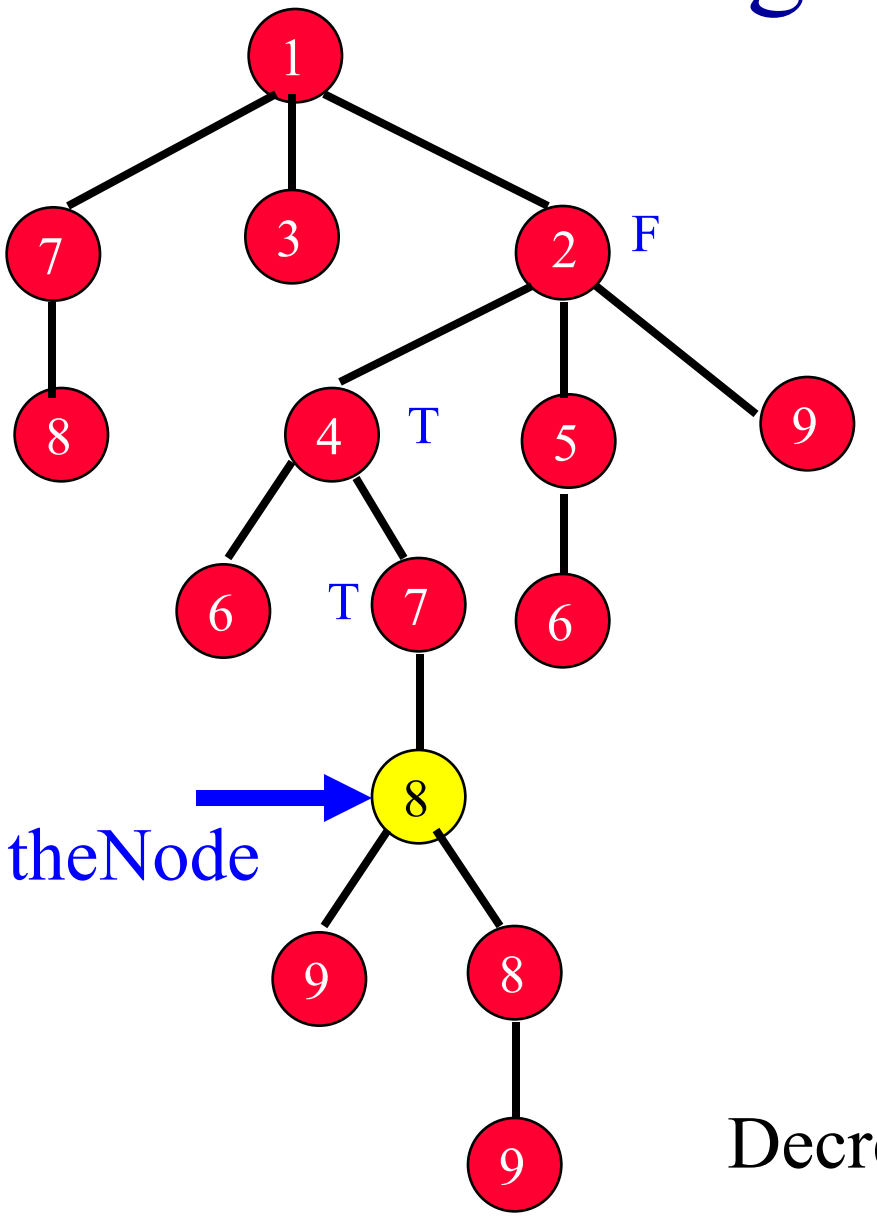


Update heap pointer if necessary

Cascading Cut

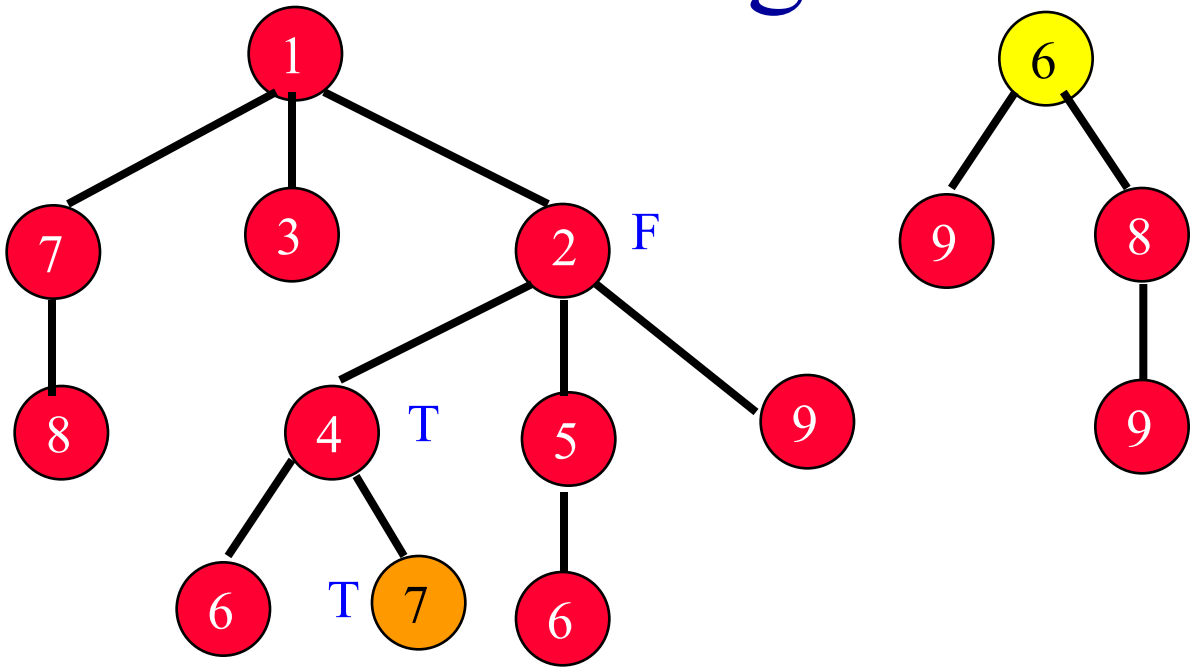
- When theNode is cut out of its sibling list in a remove or decrease key operation, follow path from parent of theNode to the root.
- Encountered nodes (other than root) with ChildCut = true are cut from their sibling lists and inserted into top-level list.
- Stop at first node with ChildCut = false.
- For this node, set ChildCut = true.

Cascading Cut Example

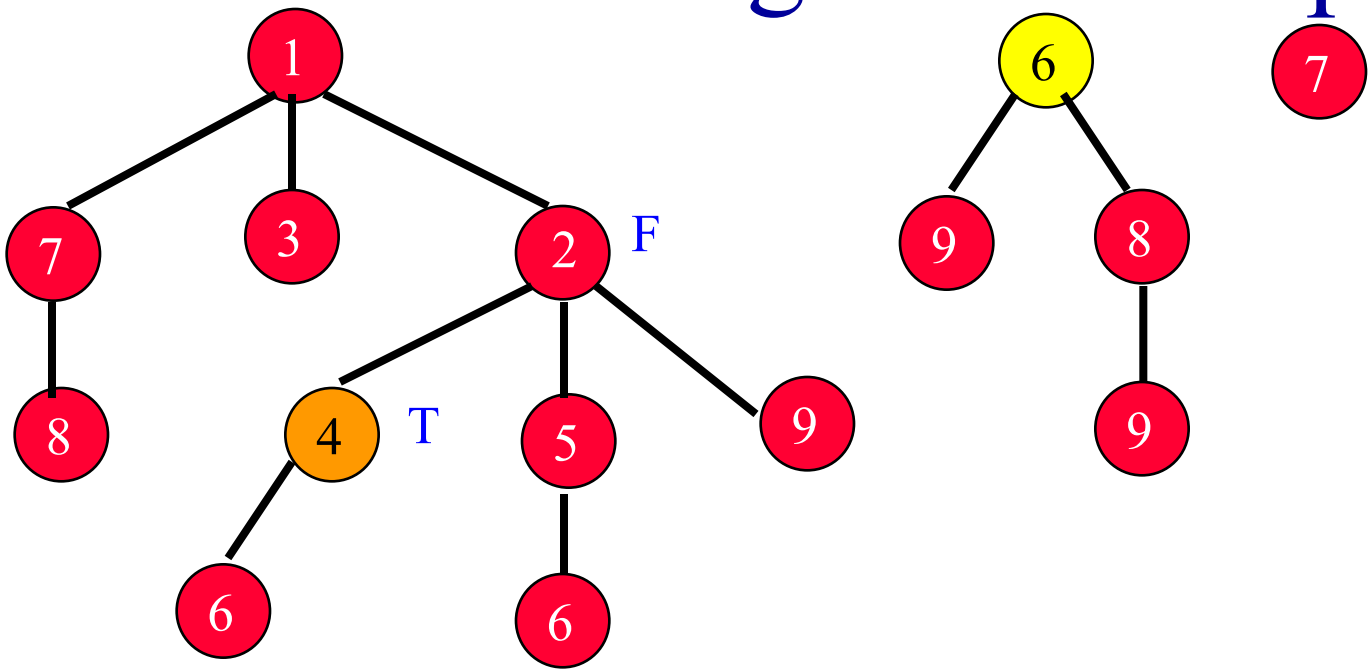


Decrease key by 2.

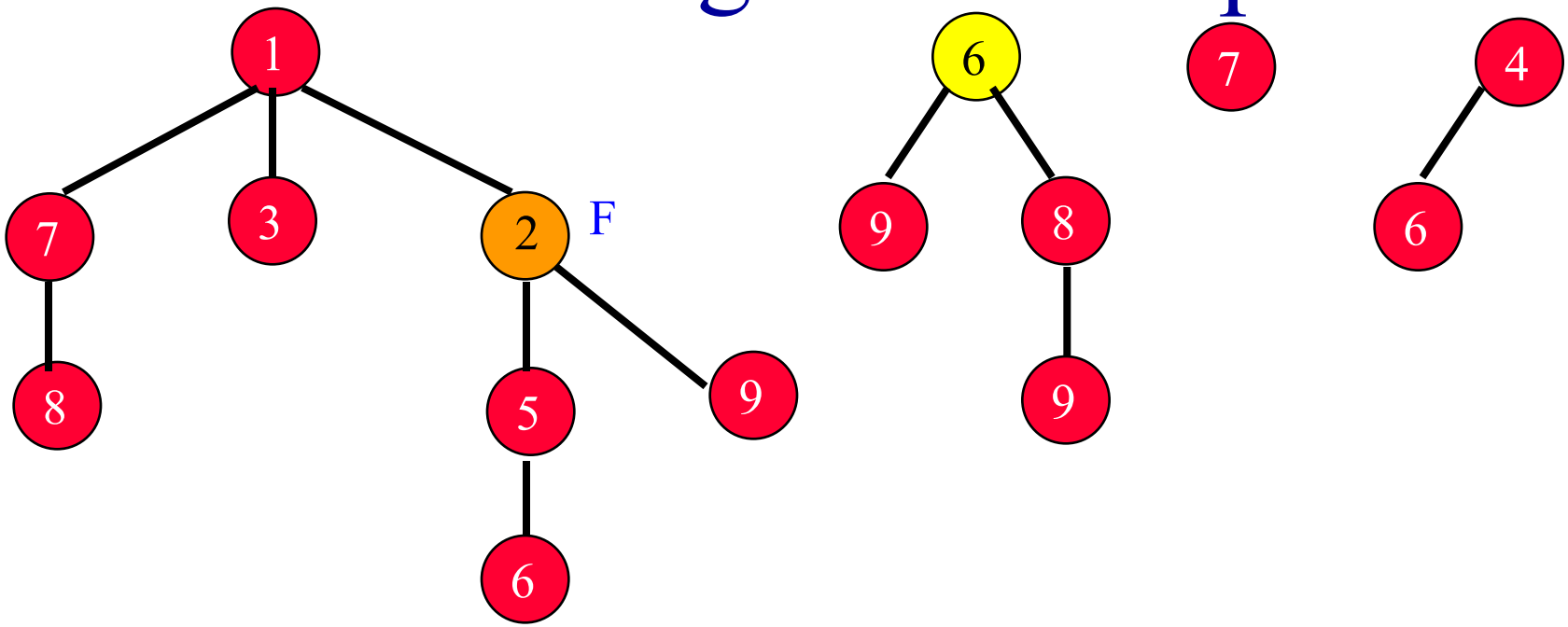
Cascading Cut Example



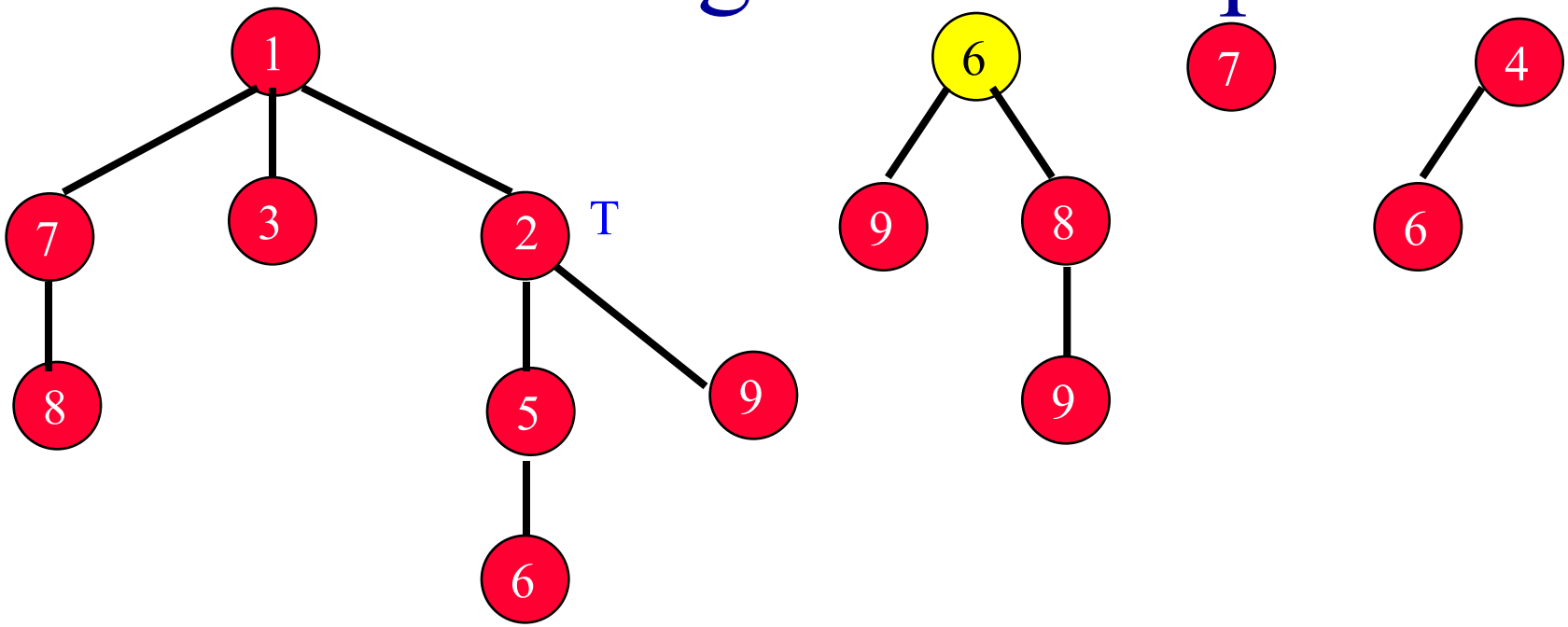
Cascading Cut Example



Cascading Cut Example



Cascading Cut Example



Actual complexity of cascading cut is O(h) = O(n).