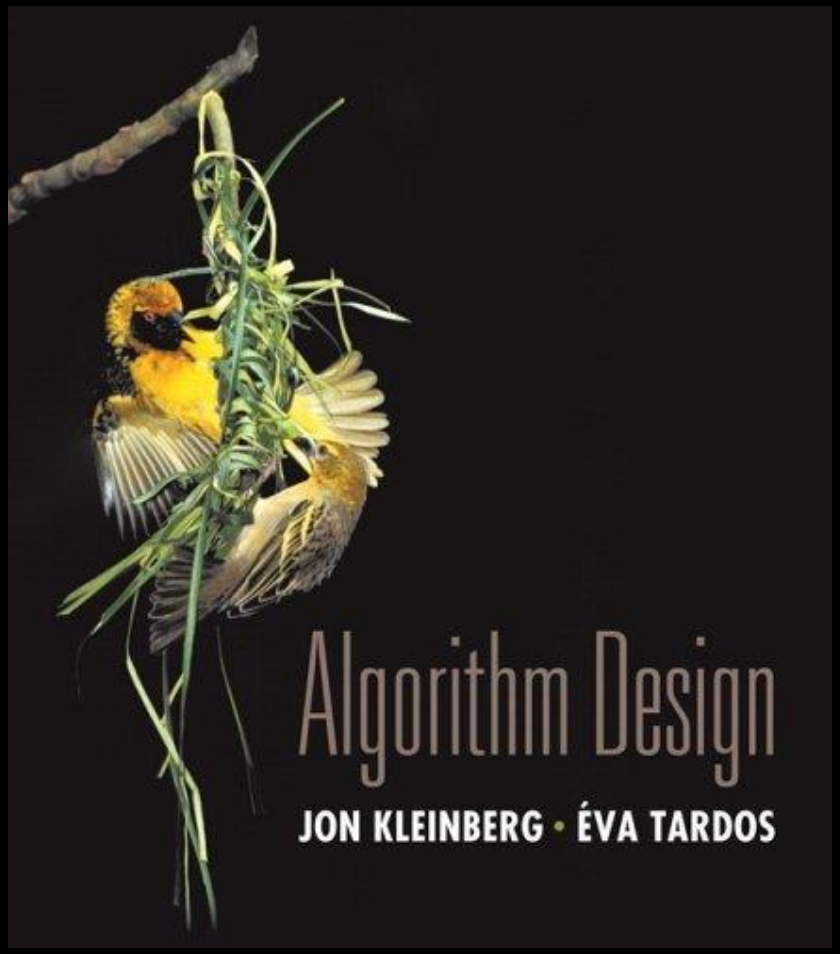


Chapter 5

Greedy Algorithms

Slides by Kevin Wayne.
Copyright © 2005 Pearson-Addison Wesley.
All rights reserved.

2.5 Priority Queues

Min/Max Priority Queue

- Collection of elements.
- Each element has a priority or key.
- Supports following operations:
  - isEmpty
  - size
  - insert an element into the priority queue
  - find an element with min/max priority
  - delete an element with min/max priority

Complexity of Operations

Two good implementations are: heaps and leftist trees.

isEmpty, size, and find => O(1) time

insert and delete => O(log n) time where n is the size of the priority queue

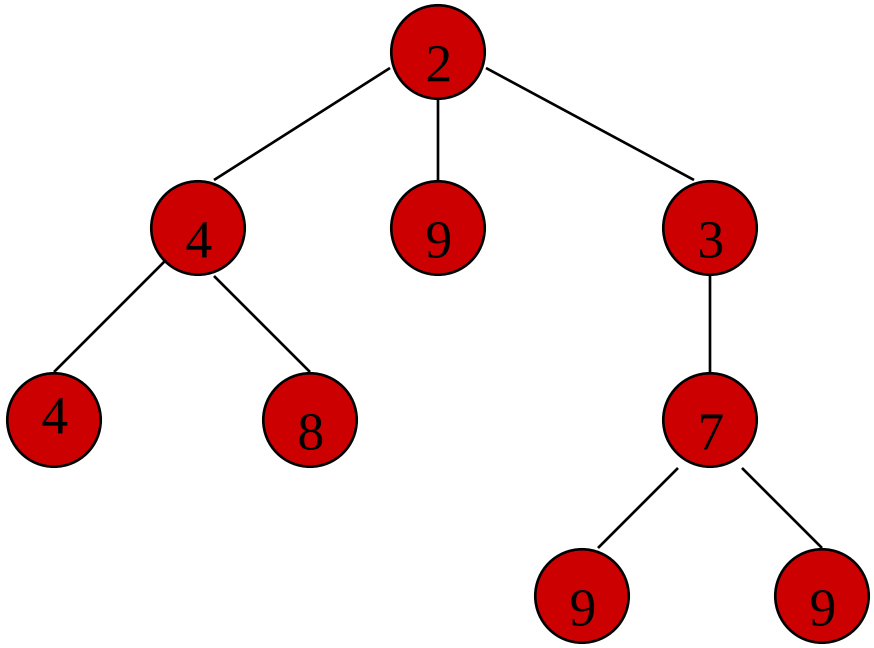
Min Tree Definition

Each tree node has a value.

Value in any node is the minimum value in the subtree for which that node is the root.

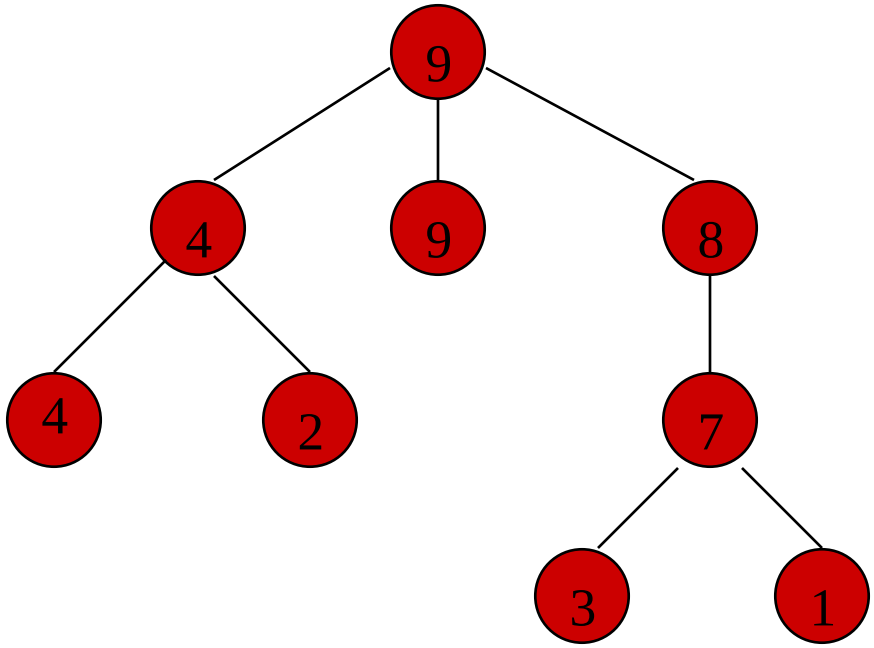
Equivalently, no descendent has a smaller value.

Min Tree Example



Root has minimum element.

Max Tree Example

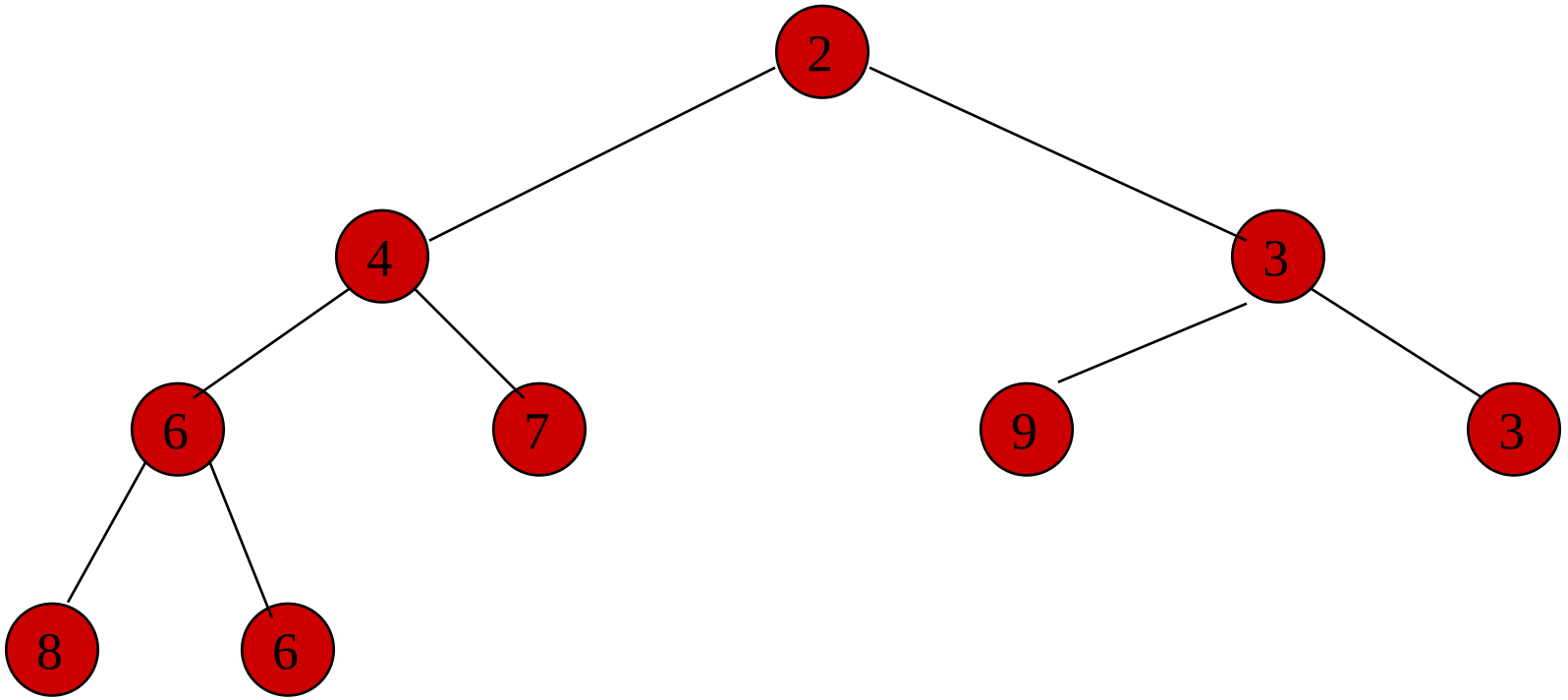


Root has maximum element.

Min Heap Definition

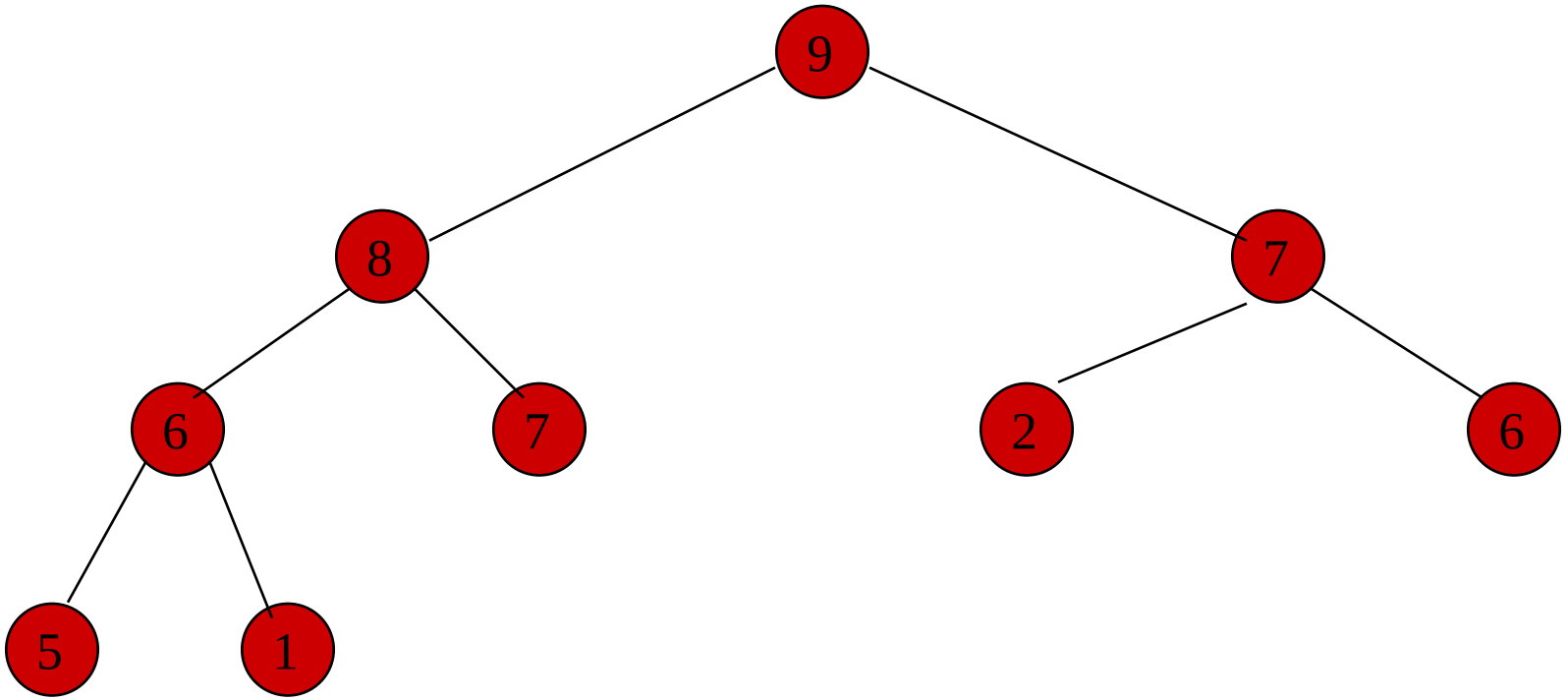
- complete binary tree
- min tree

Min Heap With 9 Nodes



Complete binary tree with 9 nodes that is also a min tree.

Max Heap With 9 Nodes

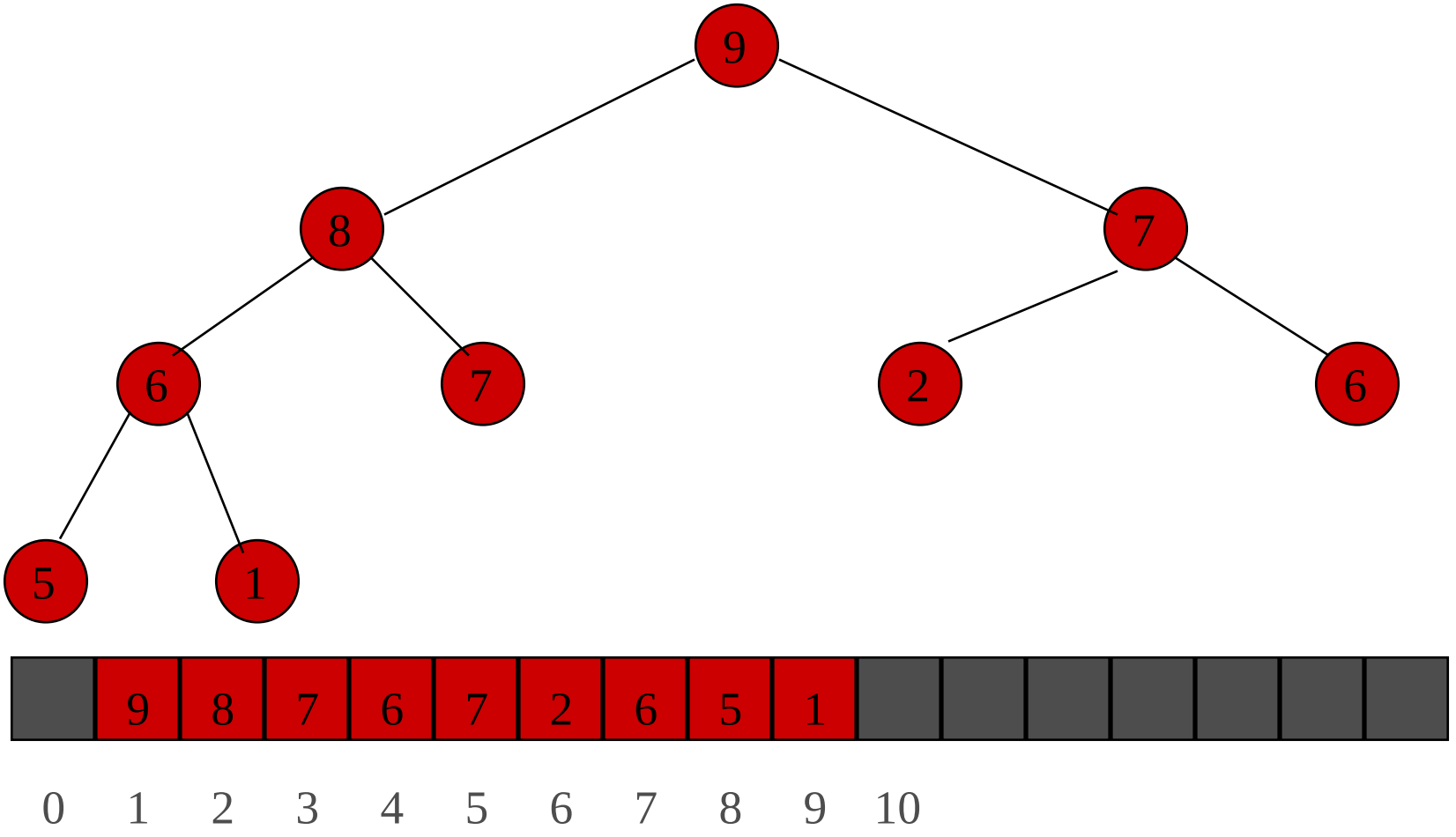


Complete binary tree with 9 nodes that is also a max tree.

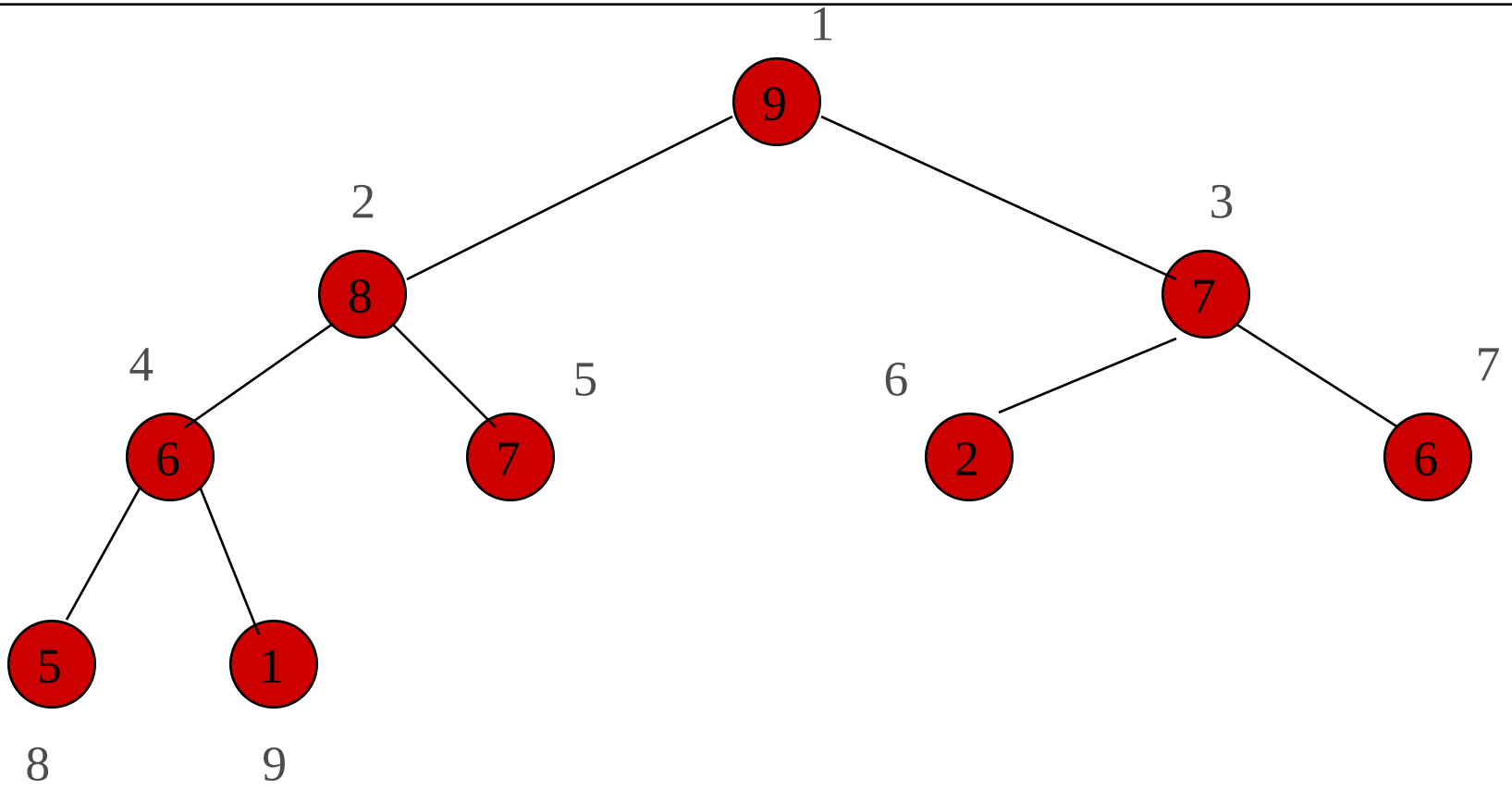
Heap Height

Since a heap is a complete binary tree, the height of an n node heap is $\log_2 (n+1)$.

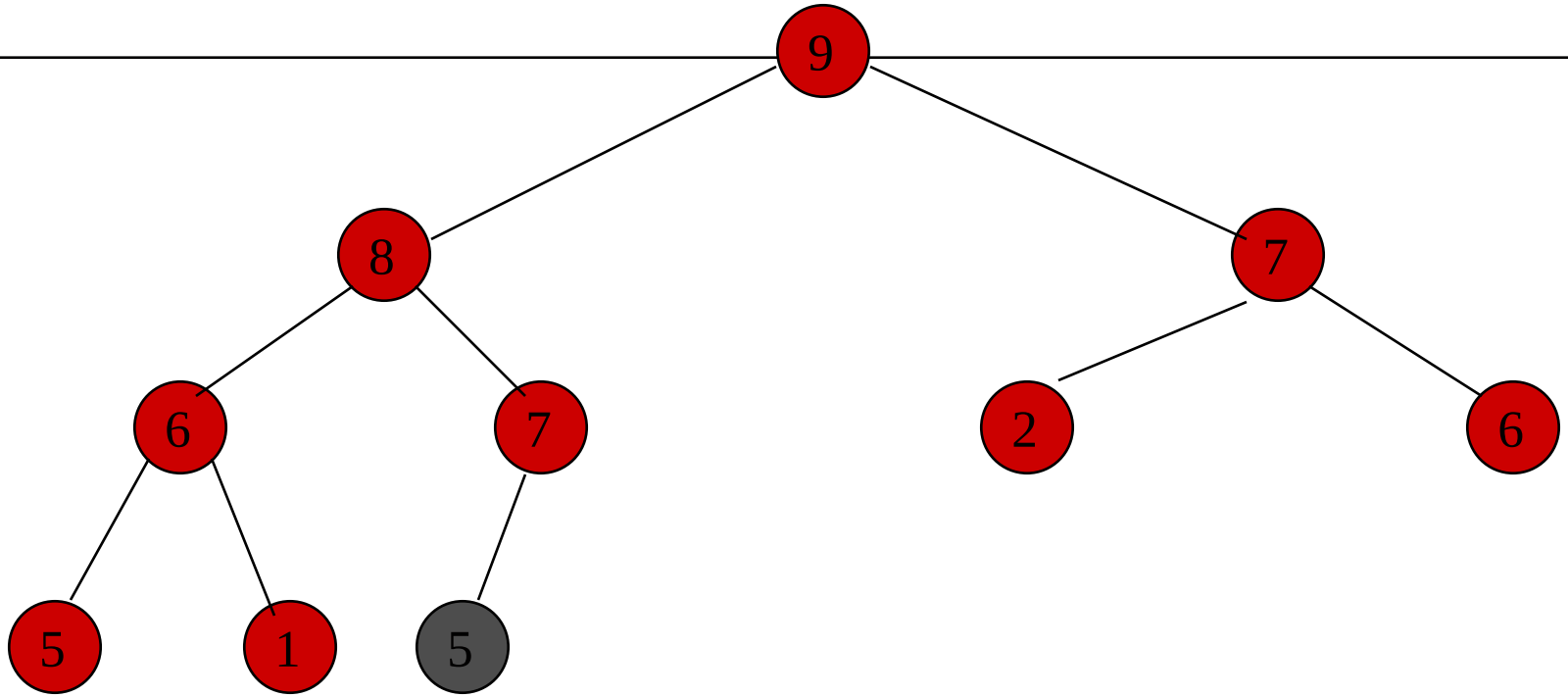
A Heap is Efficiently Represented as an Array



Moving Up and Down a Heap



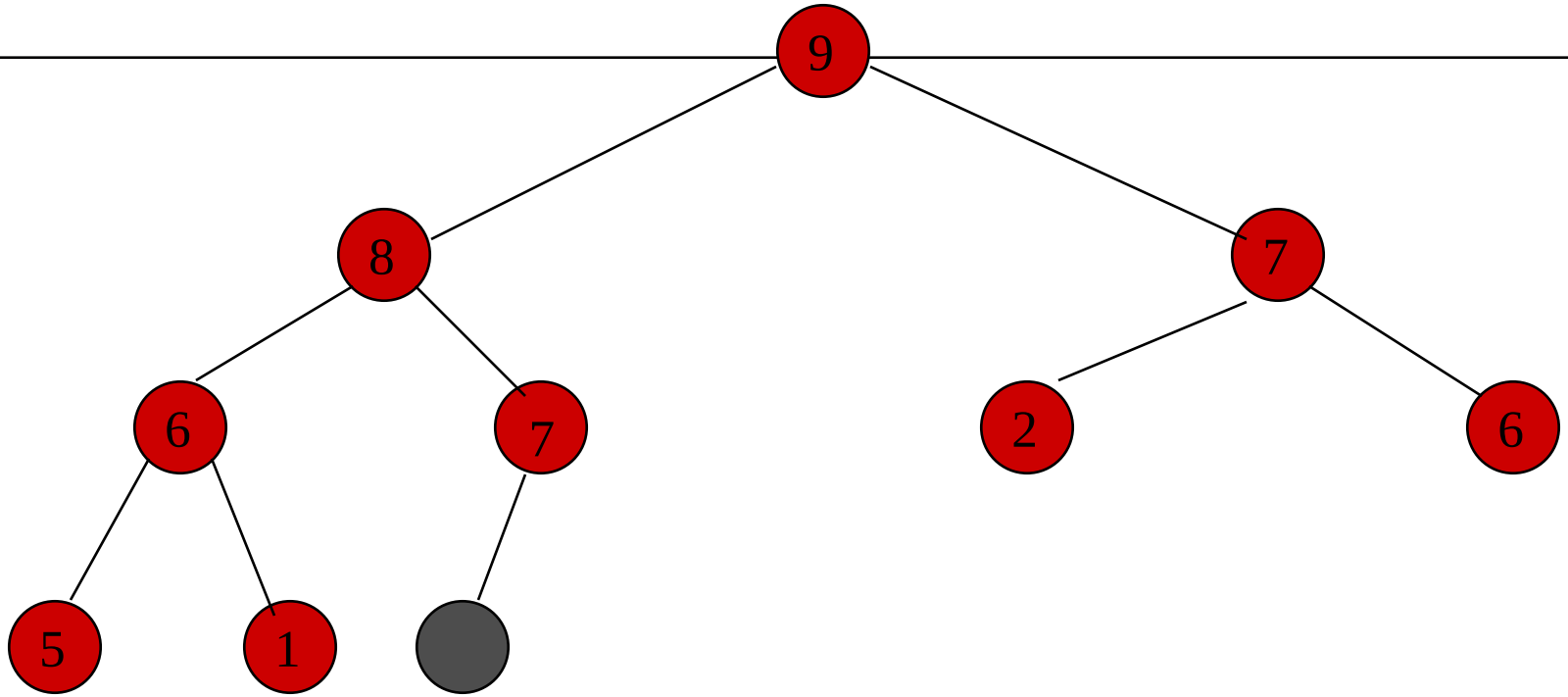
Inserting An Element Into A Max Heap



Complete binary tree with 10 nodes.

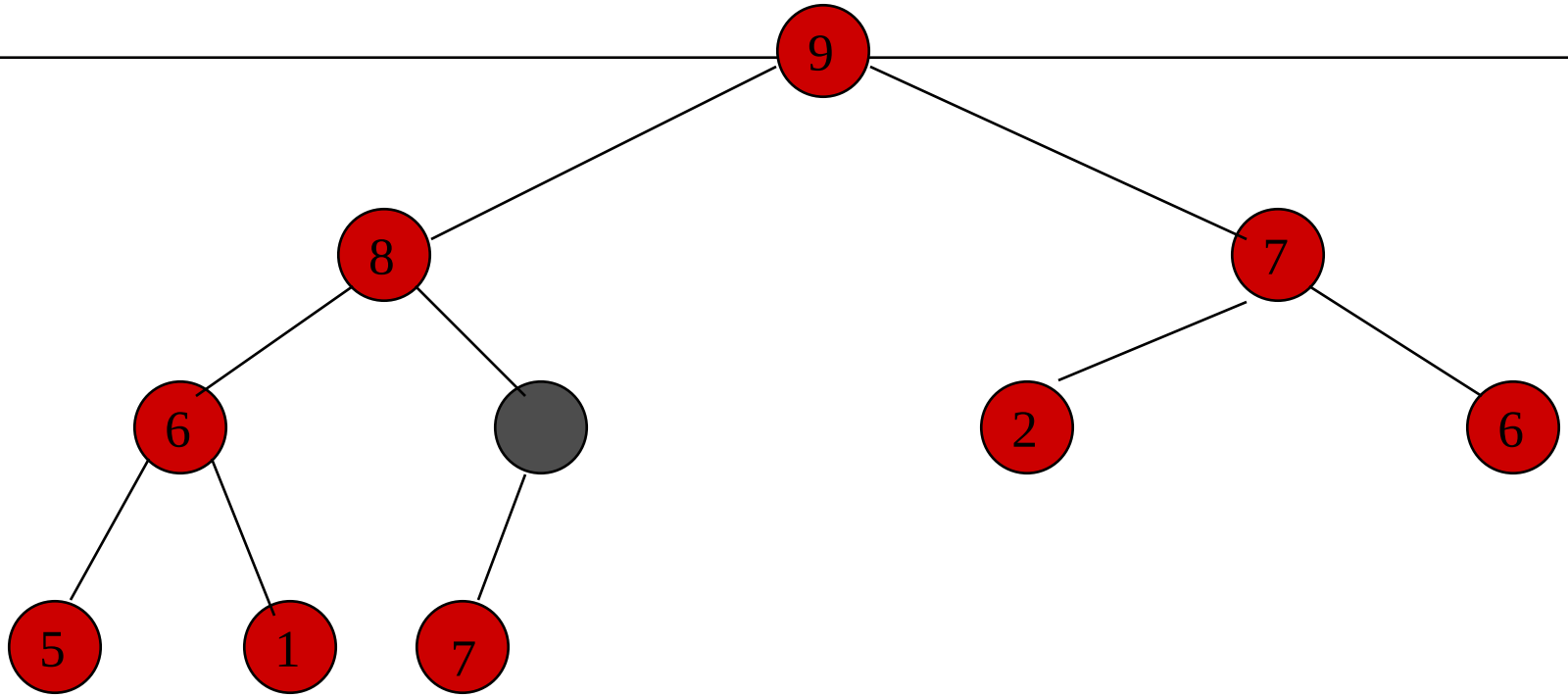
New element is 5.

Inserting An Element Into A Max Heap



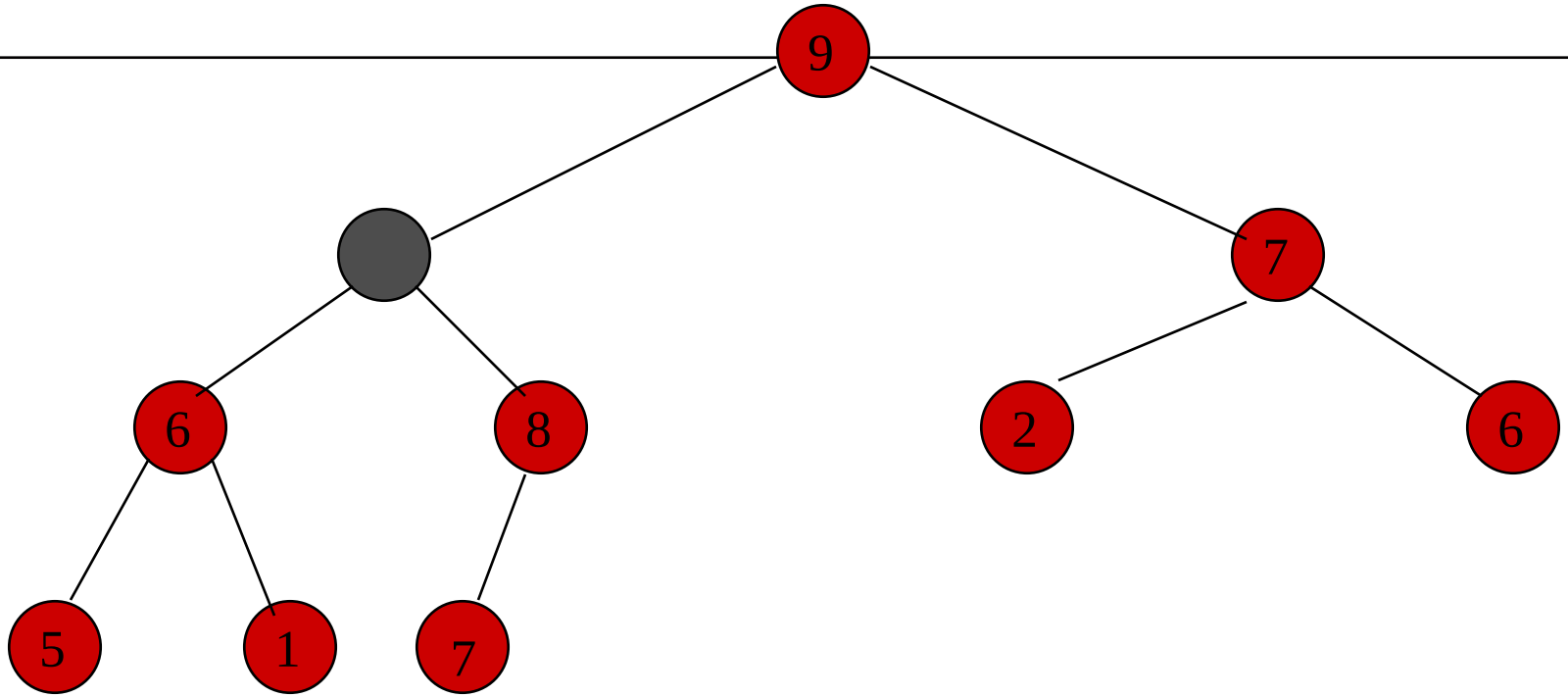
New element is 20.

Inserting An Element Into A Max Heap



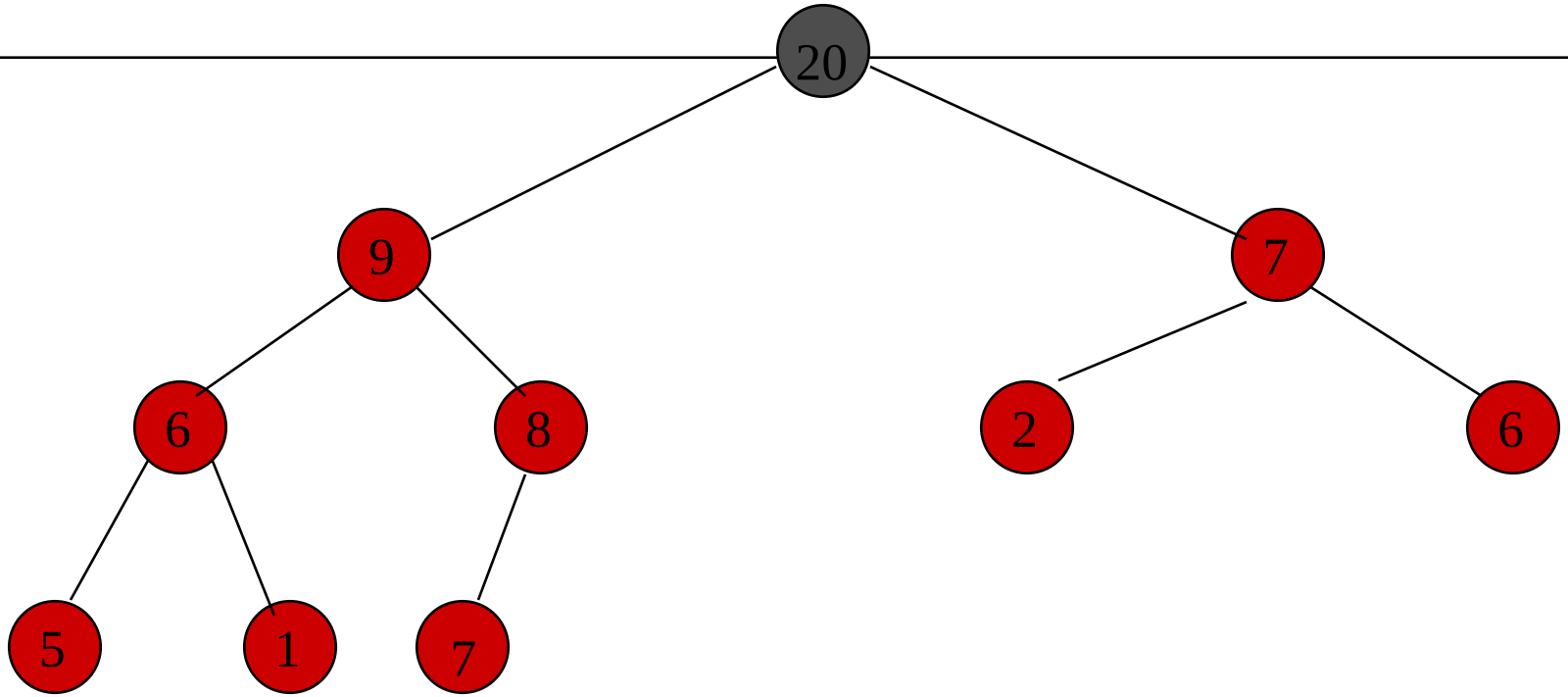
New element is 20.

Inserting An Element Into A Max Heap



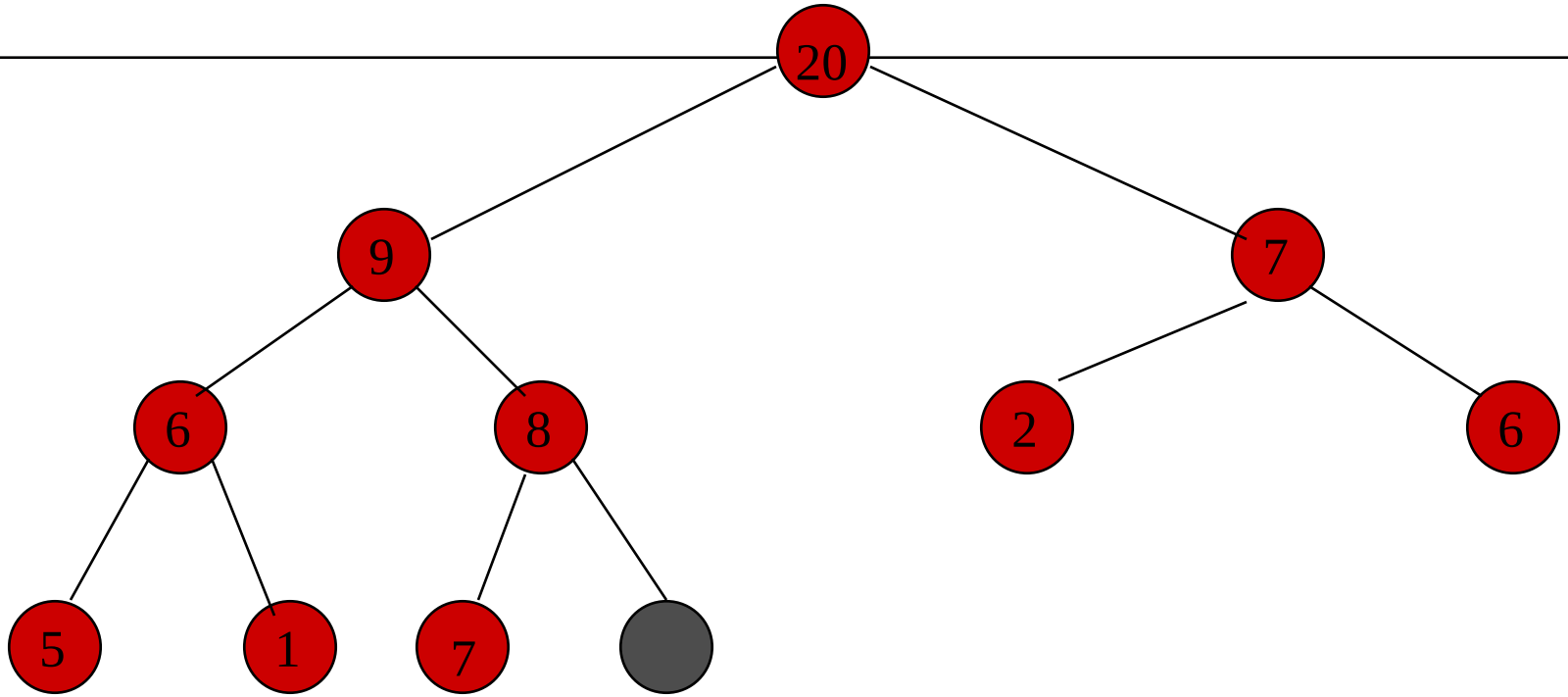
New element is 20.

Inserting An Element Into A Max Heap



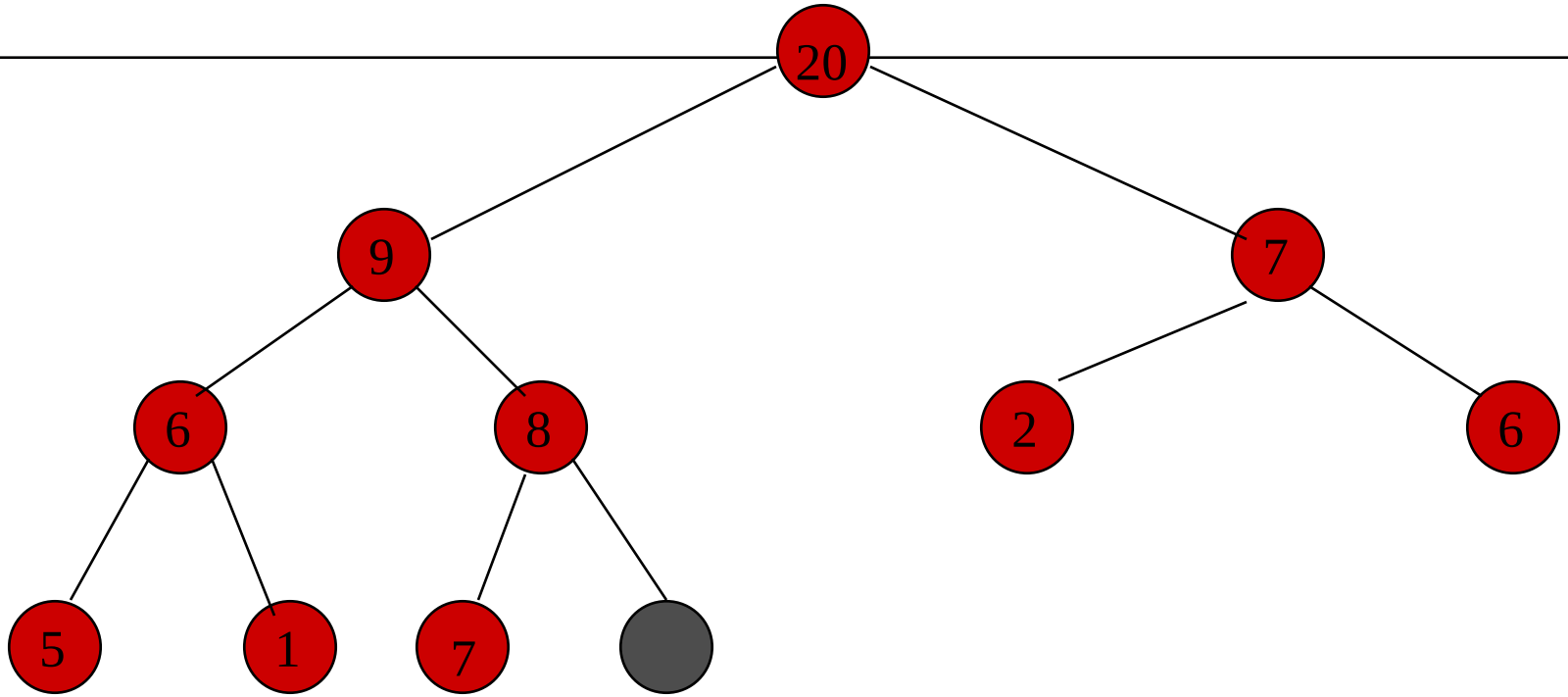
New element is 20.

Inserting An Element Into A Max Heap



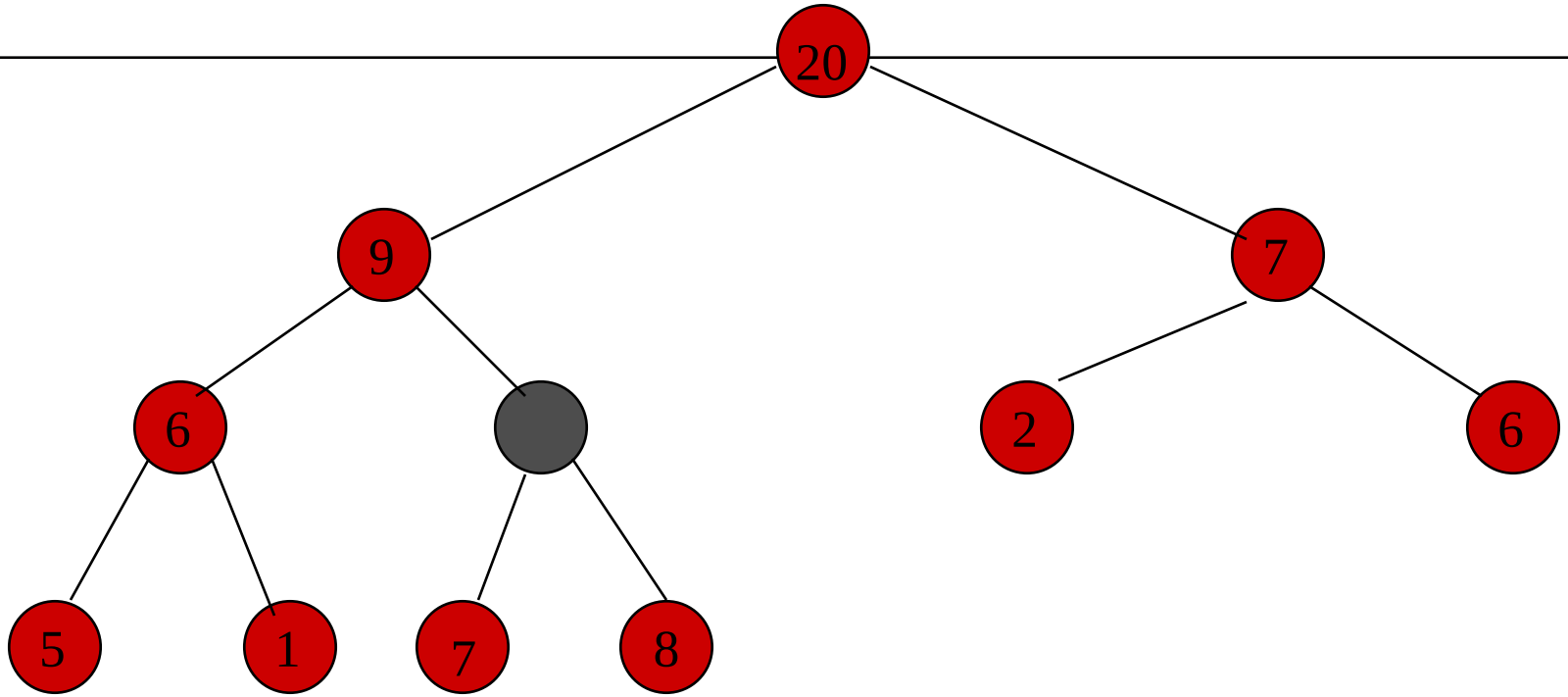
Complete binary tree with 11 nodes.

Inserting An Element Into A Max Heap



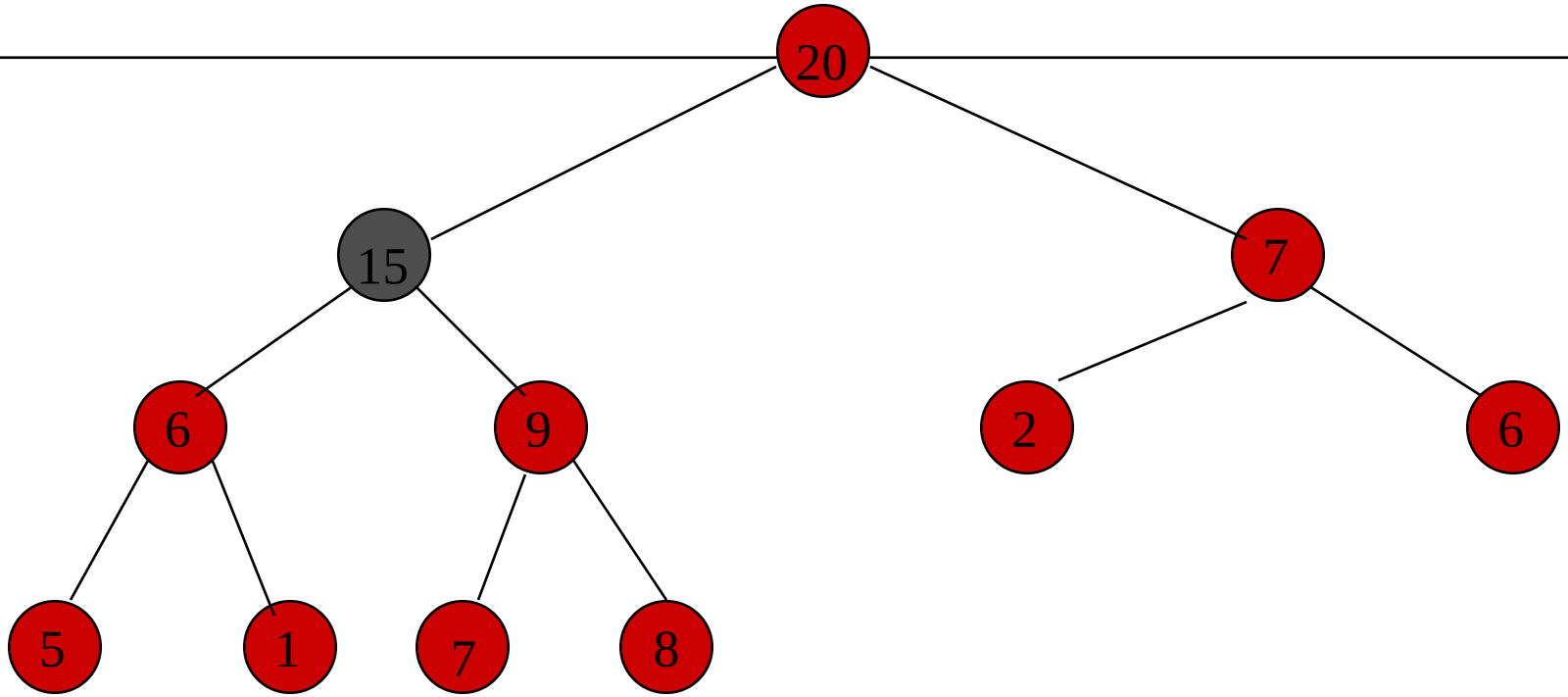
New element is 15.

Inserting An Element Into A Max Heap



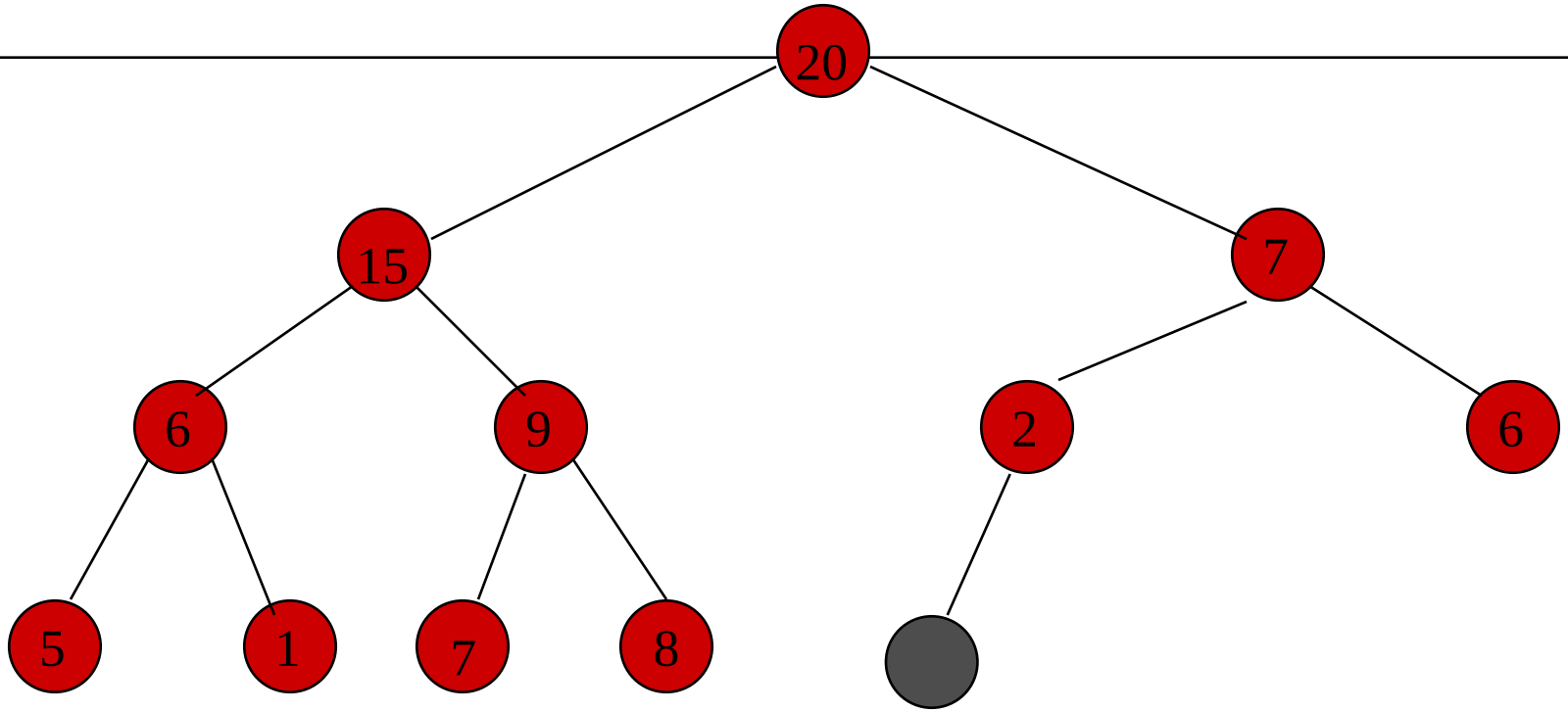
New element is 15.

Inserting An Element Into A Max Heap



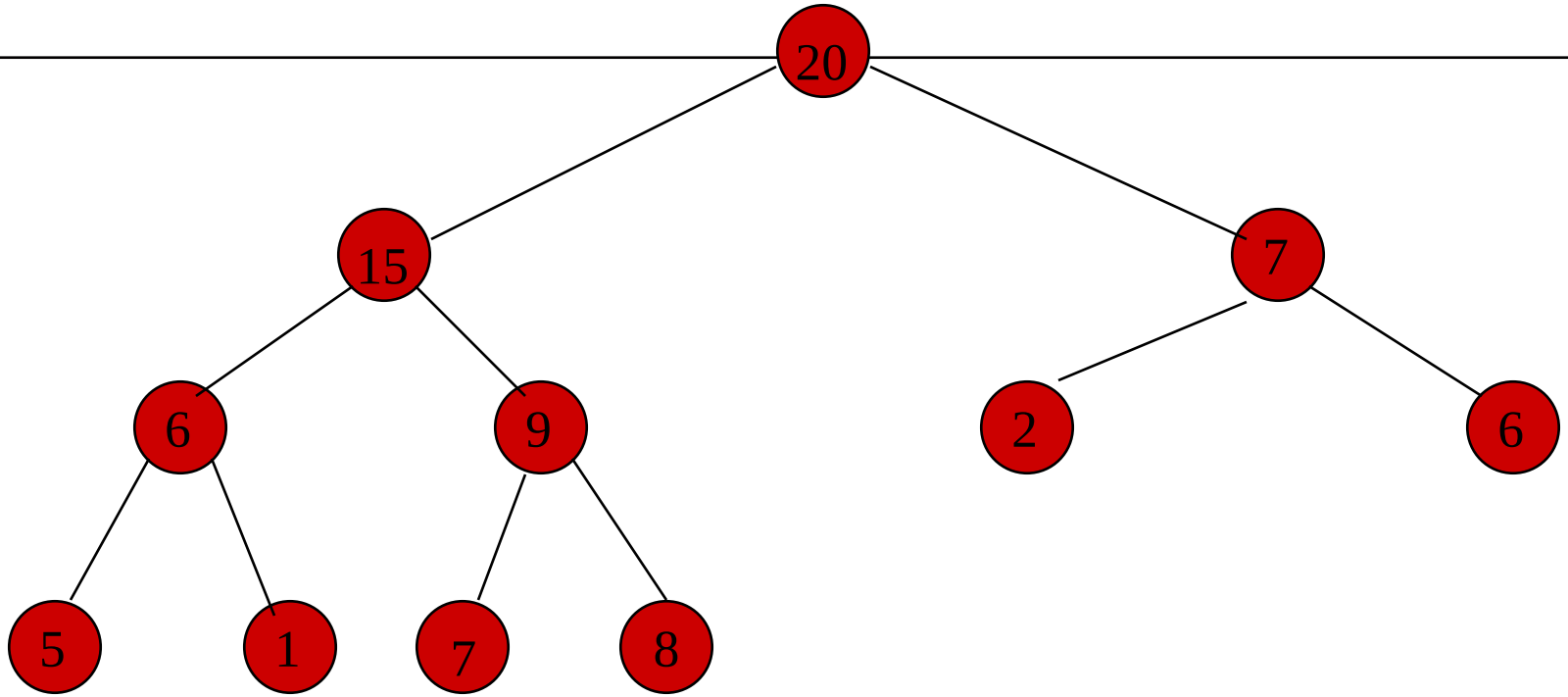
New element is 15.

Complexity Of Insert



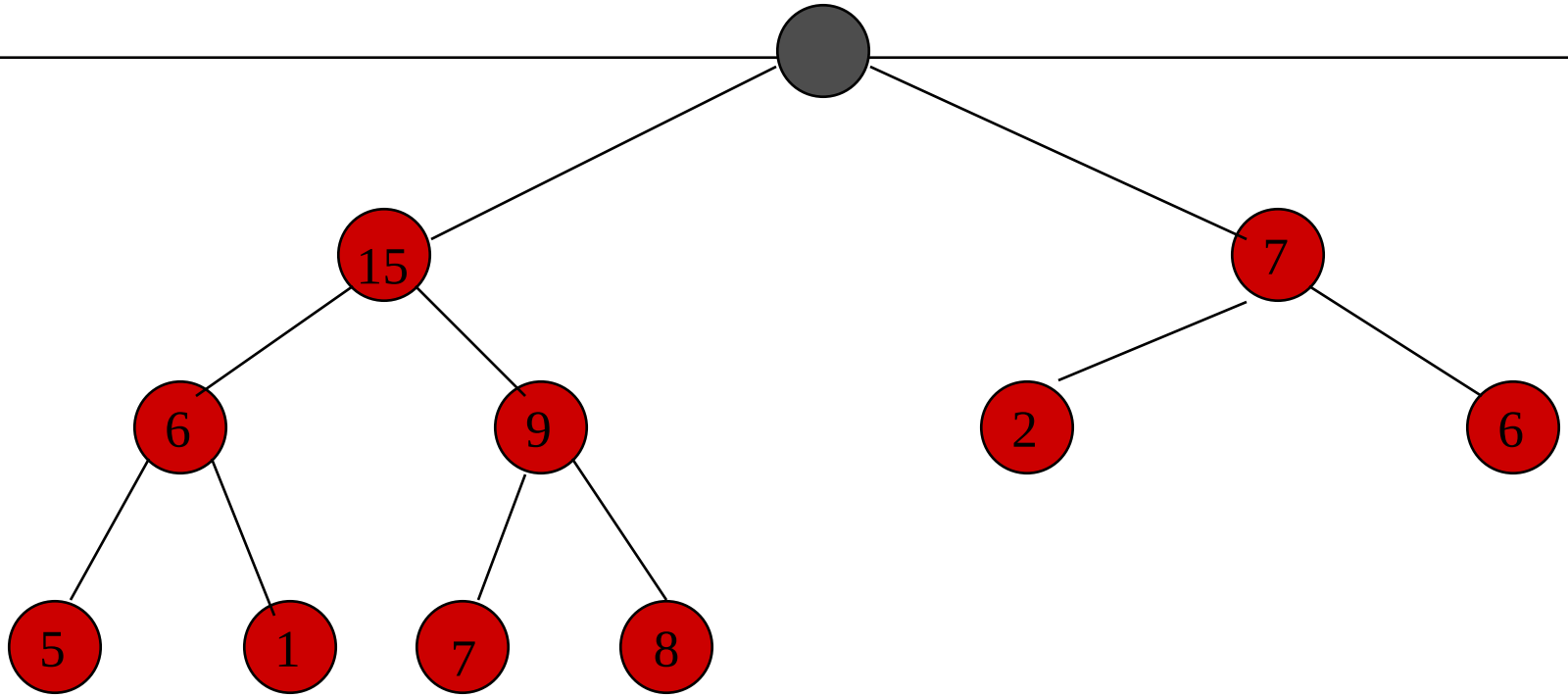
Complexity is O(log n), where n is heap size.

Removing The Max Element



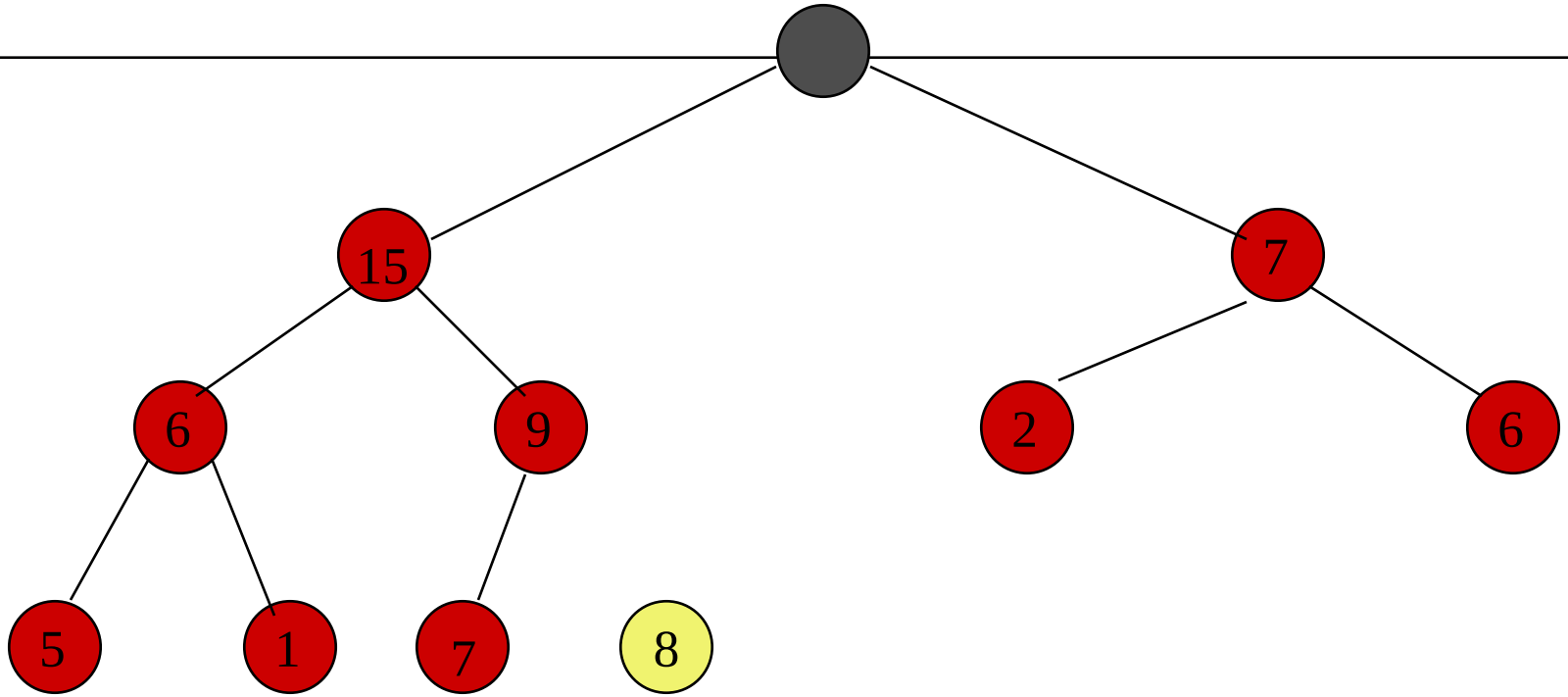
Max element is in the root.

Removing The Max Element



After max element is removed.

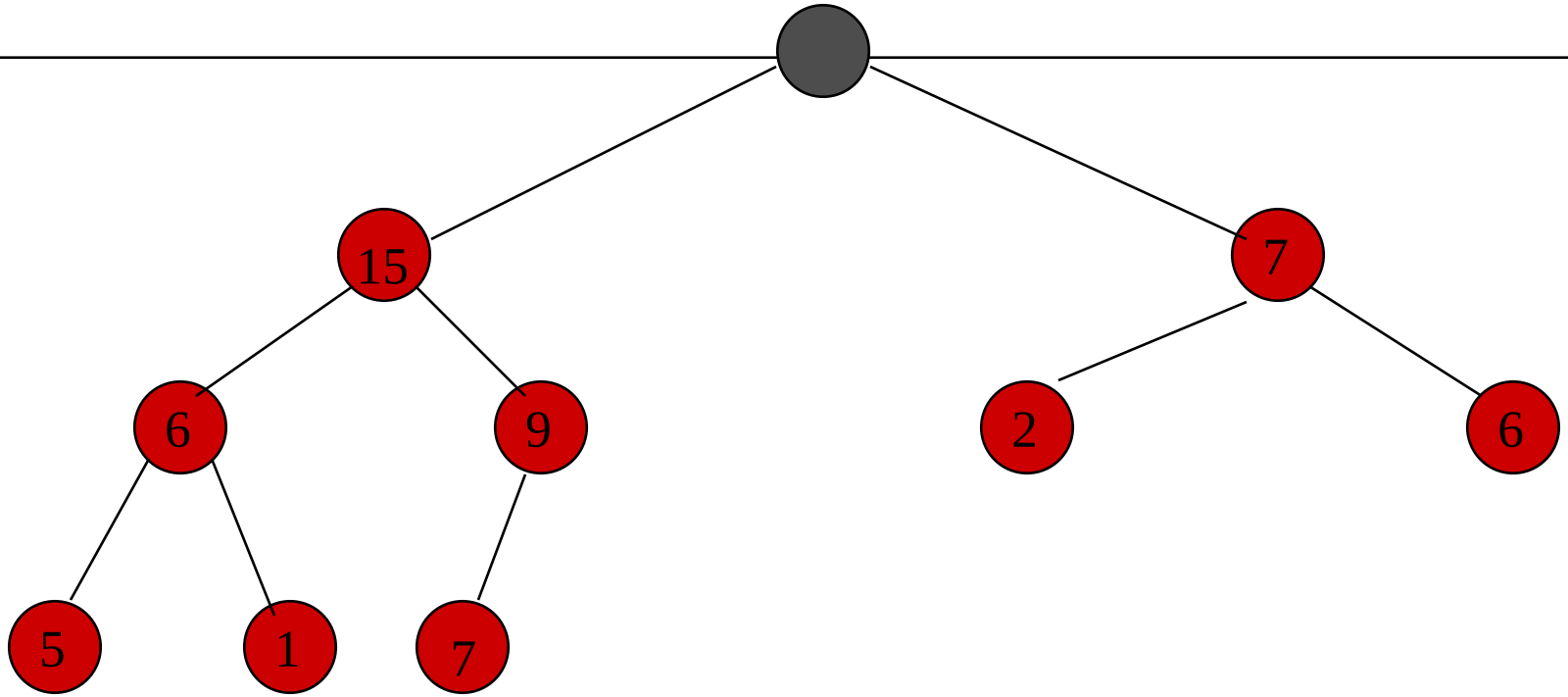
Removing The Max Element



Heap with 10 nodes.

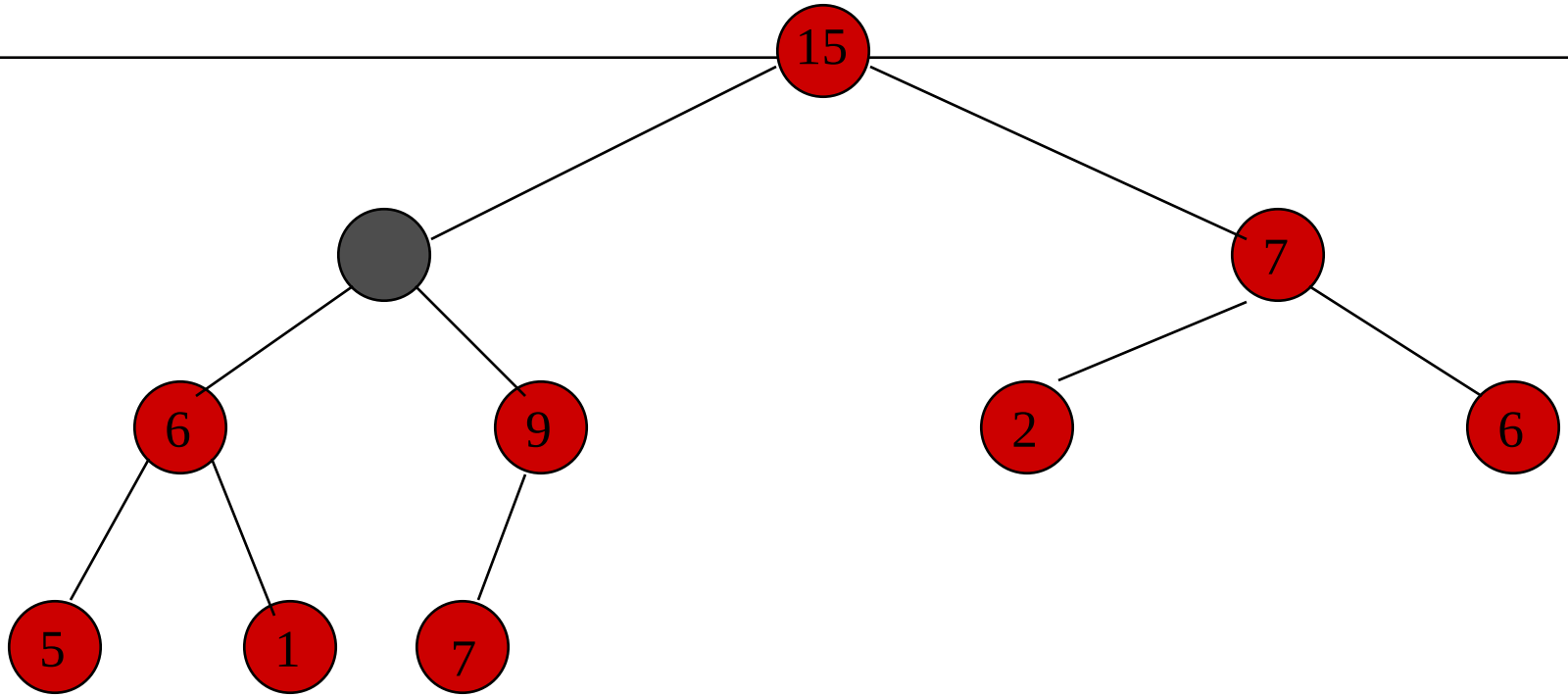
Reinsert 8 into the heap.

Removing The Max Element



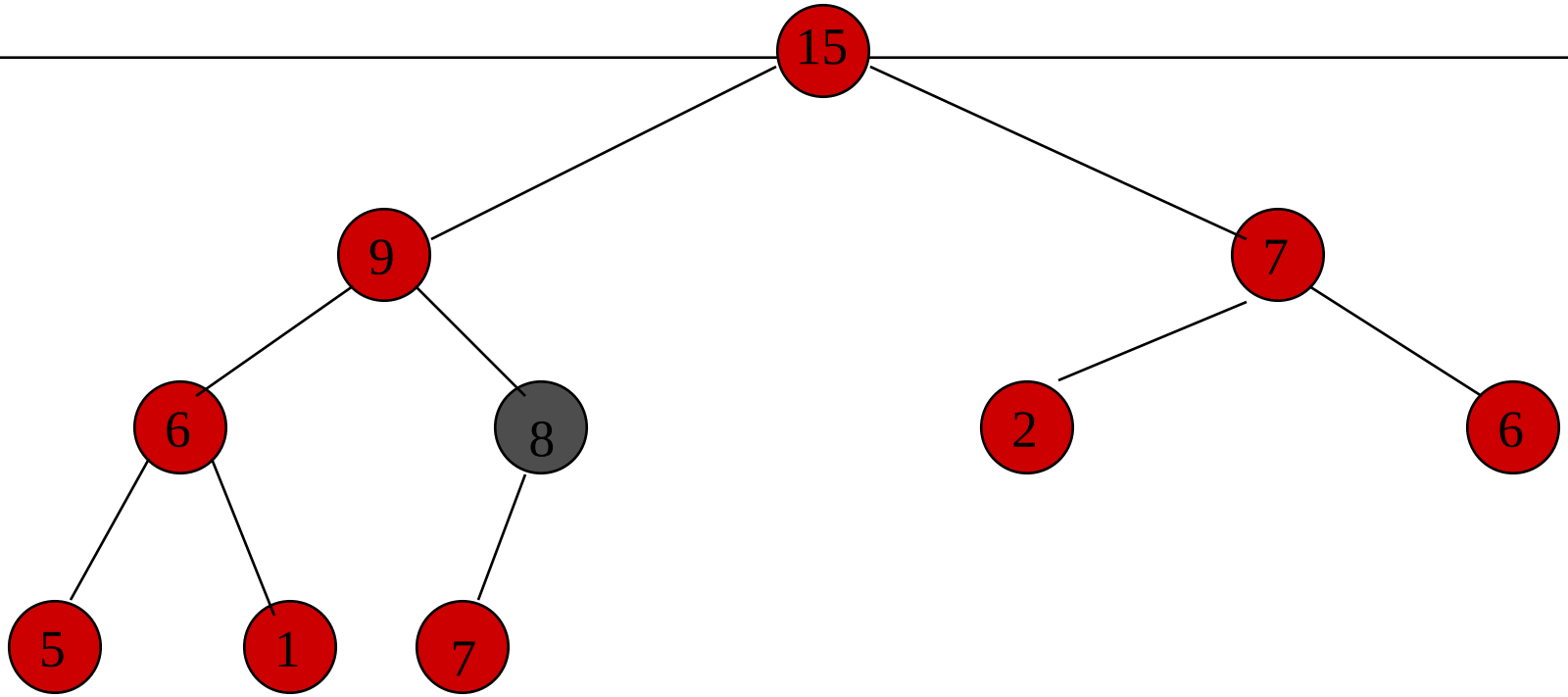
Reinsert 8 into the heap.

Removing The Max Element



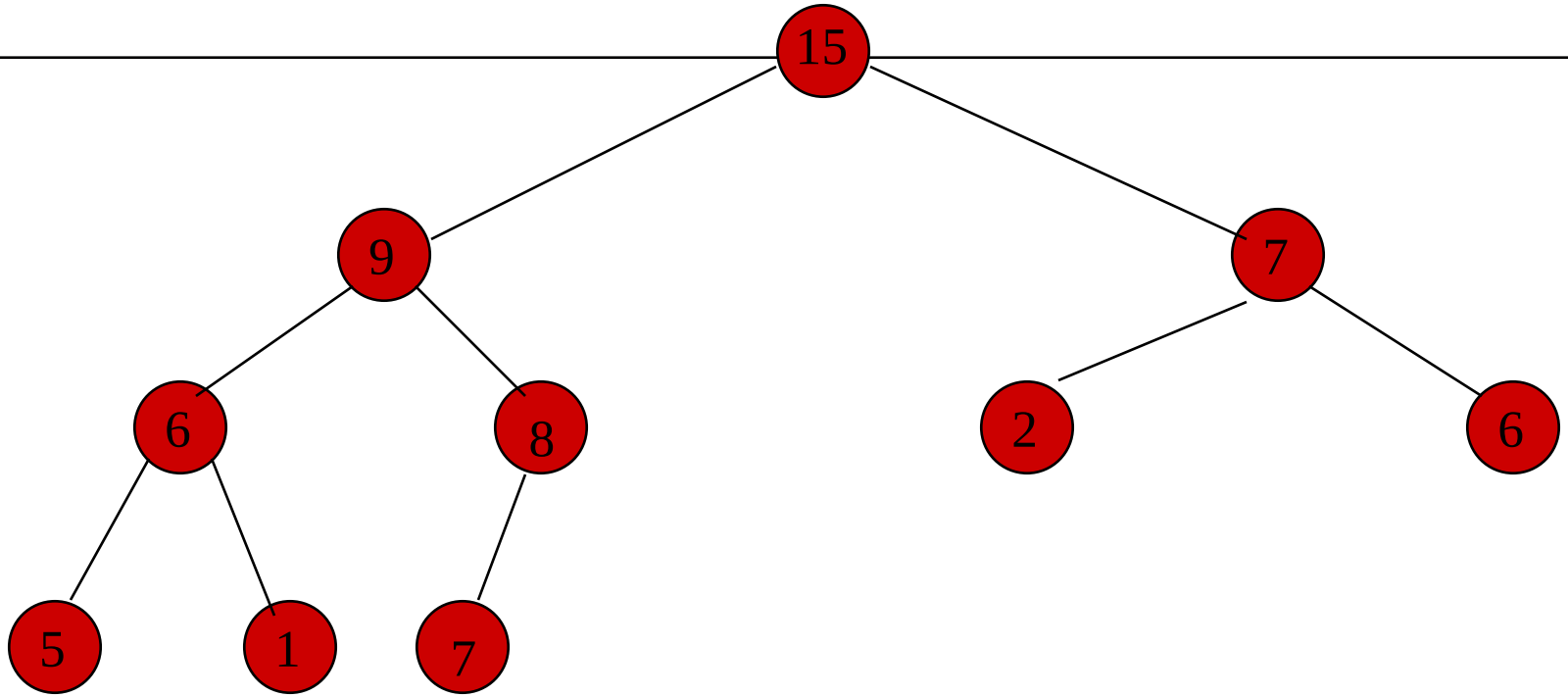
Reinsert 8 into the heap.

Removing The Max Element



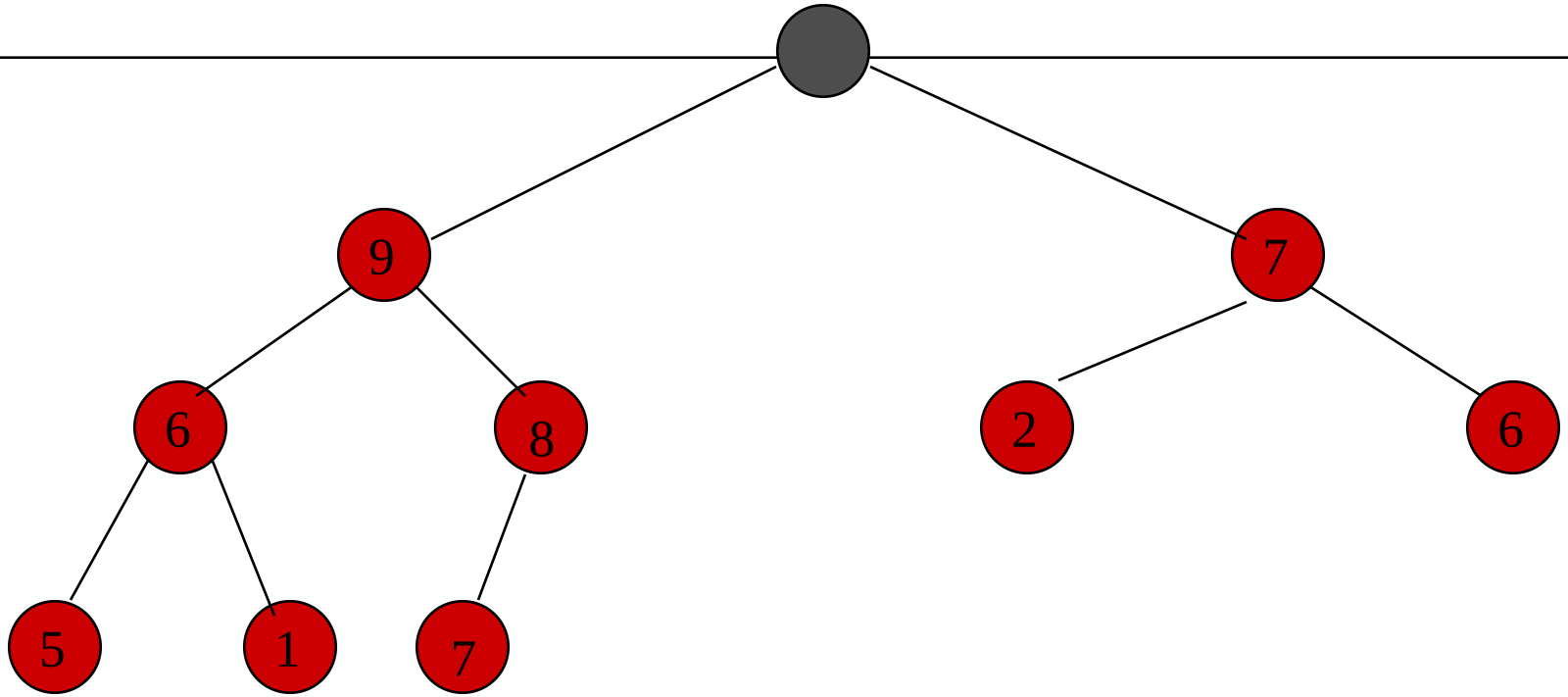
Reinsert 8 into the heap.

Removing The Max Element



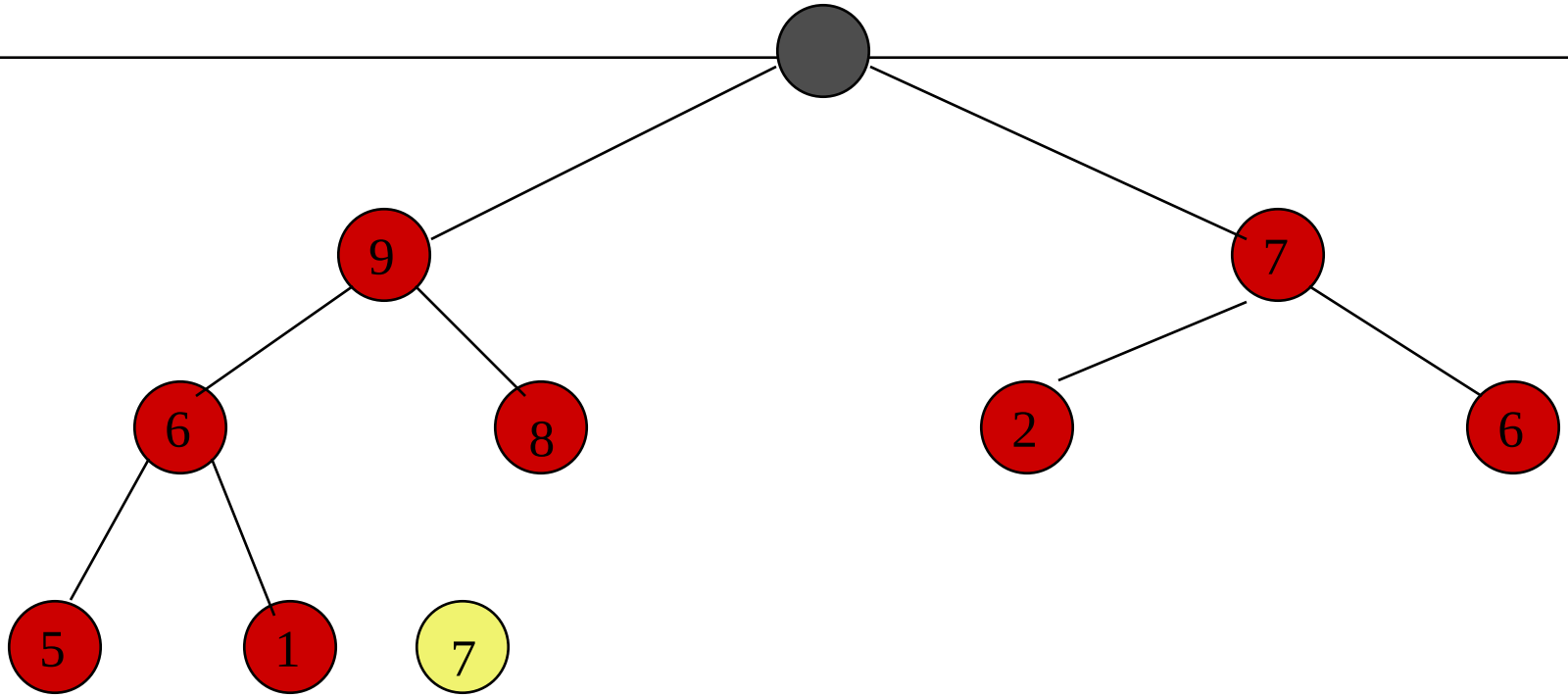
Max element is 15.

Removing The Max Element



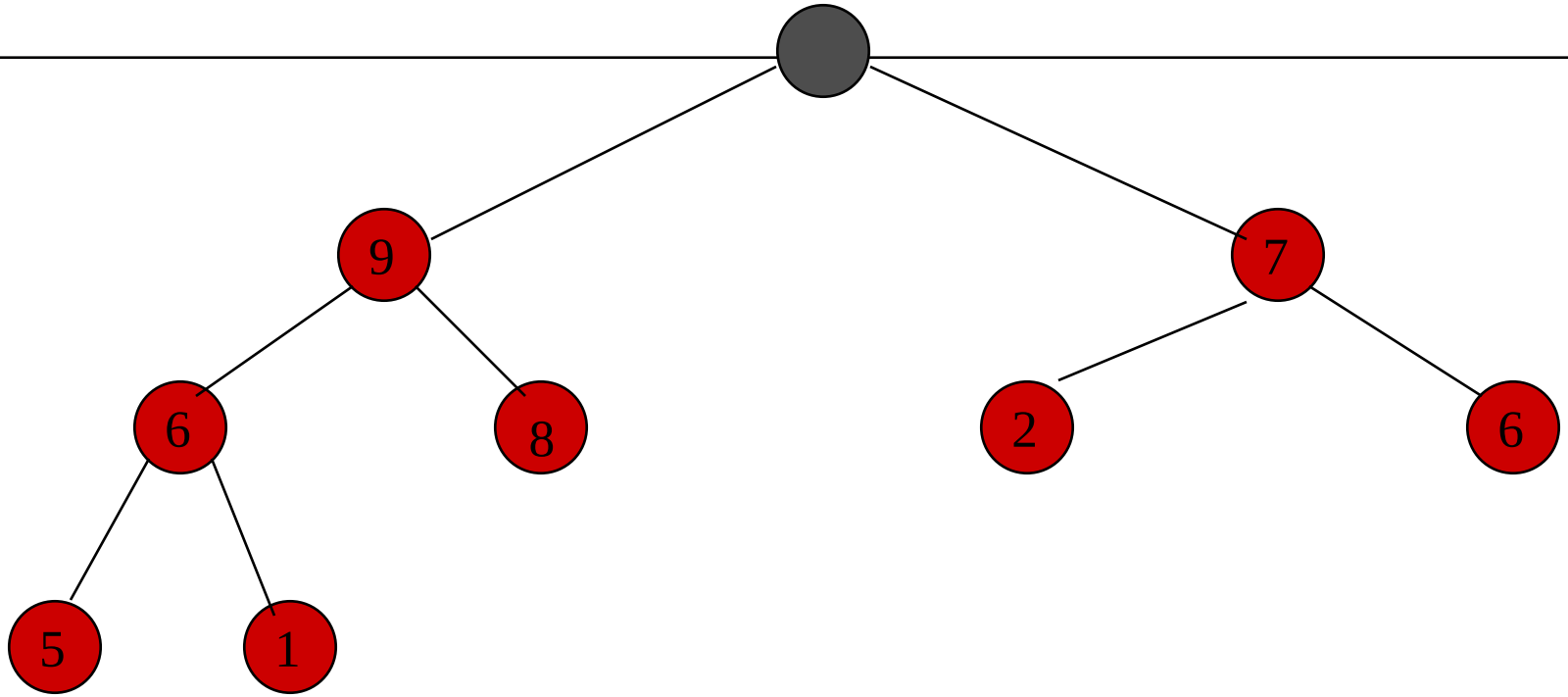
After max element is removed.

Removing The Max Element



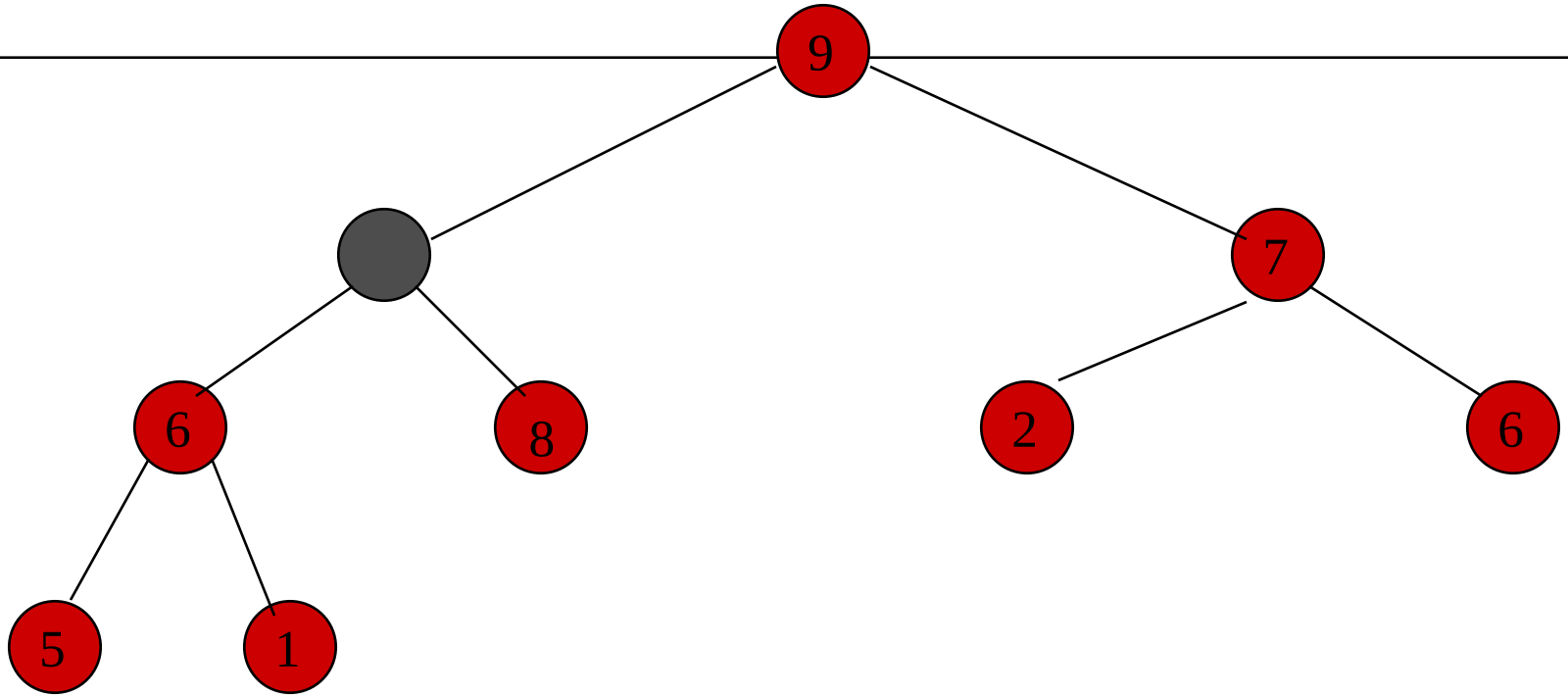
Heap with 9 nodes.

Removing The Max Element



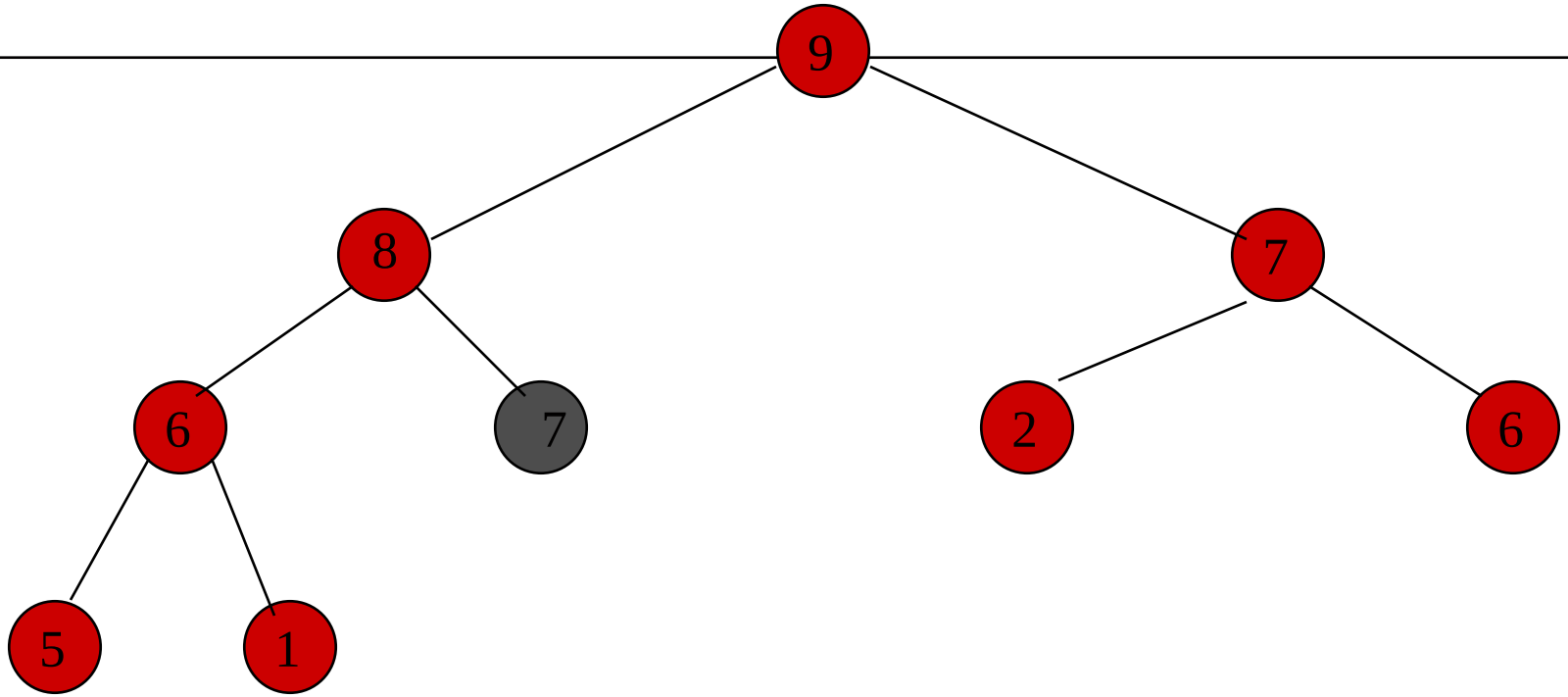
Reinsert 7.

Removing The Max Element



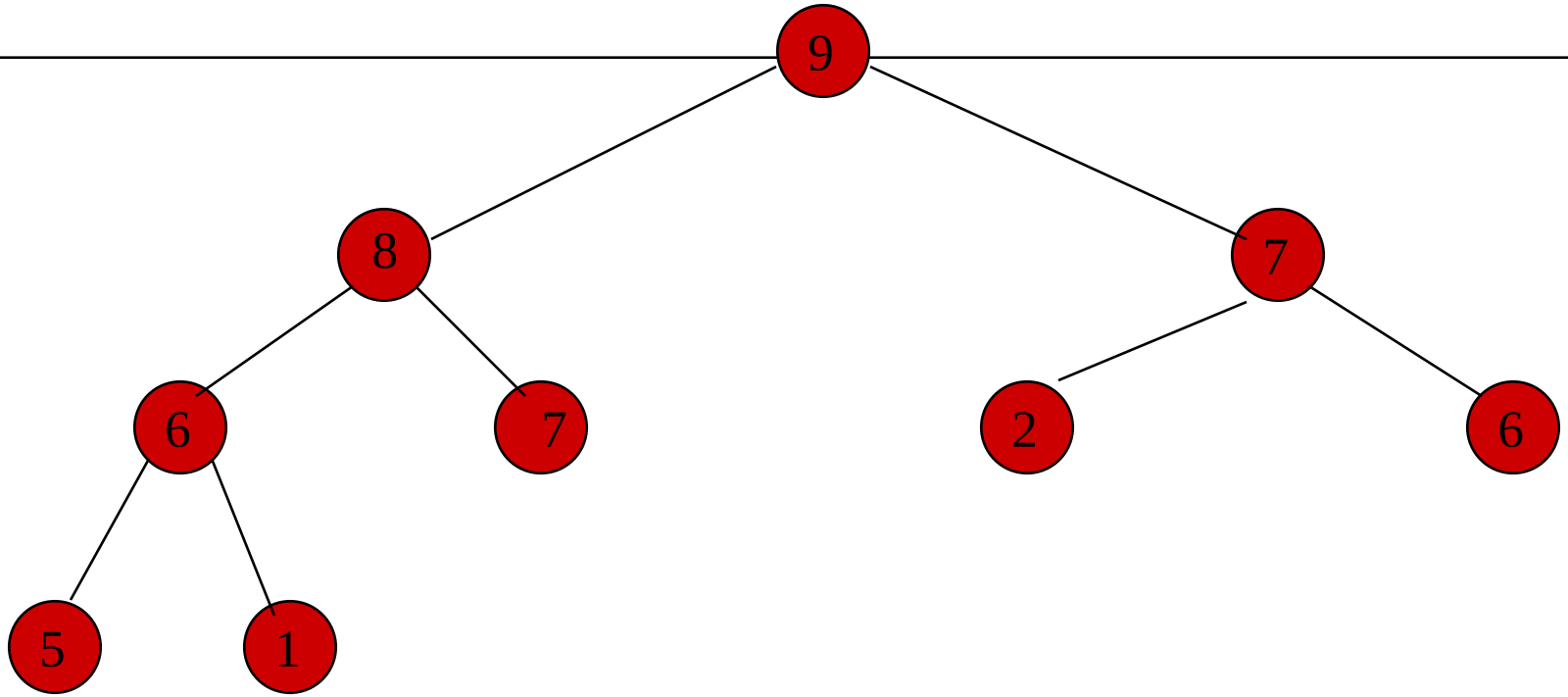
Reinsert 7.

Removing The Max Element



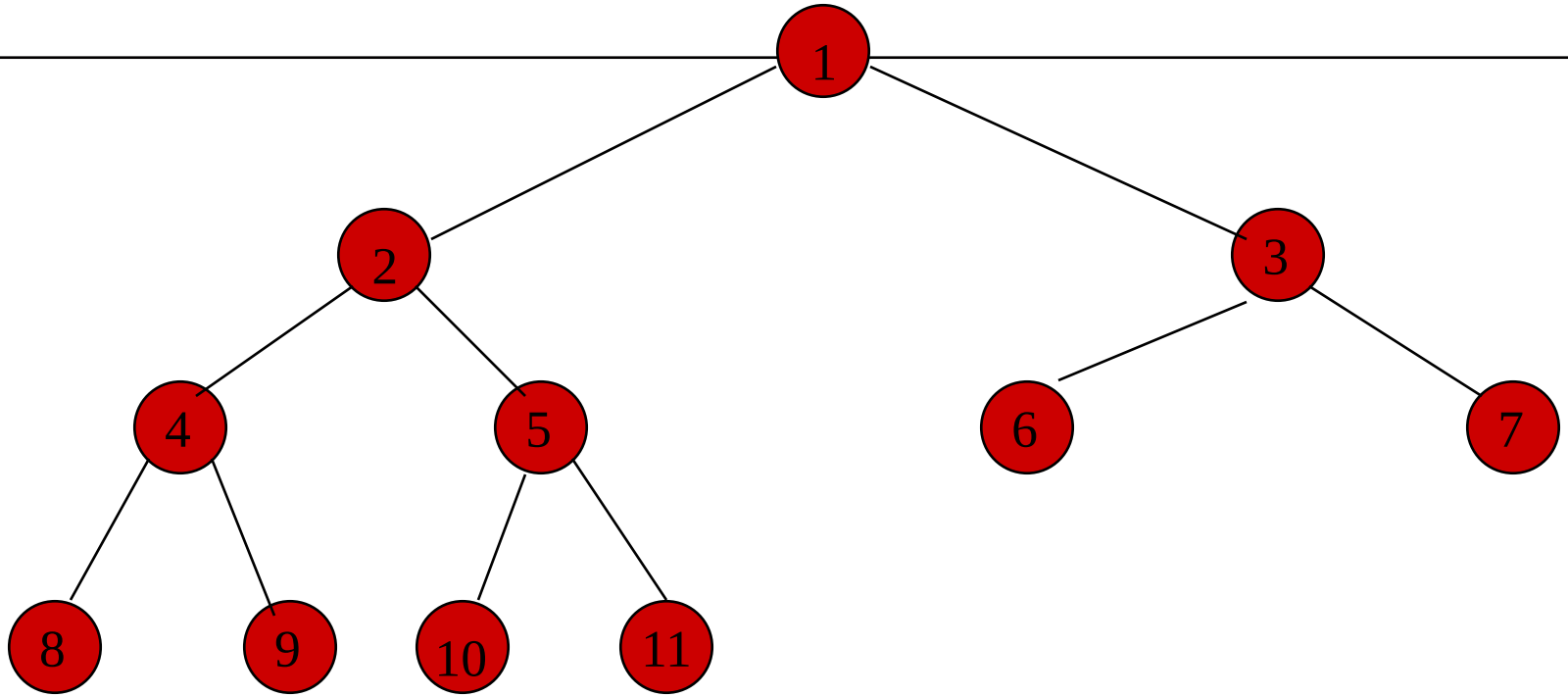
Reinsert 7.

Complexity Of Remove Max Element



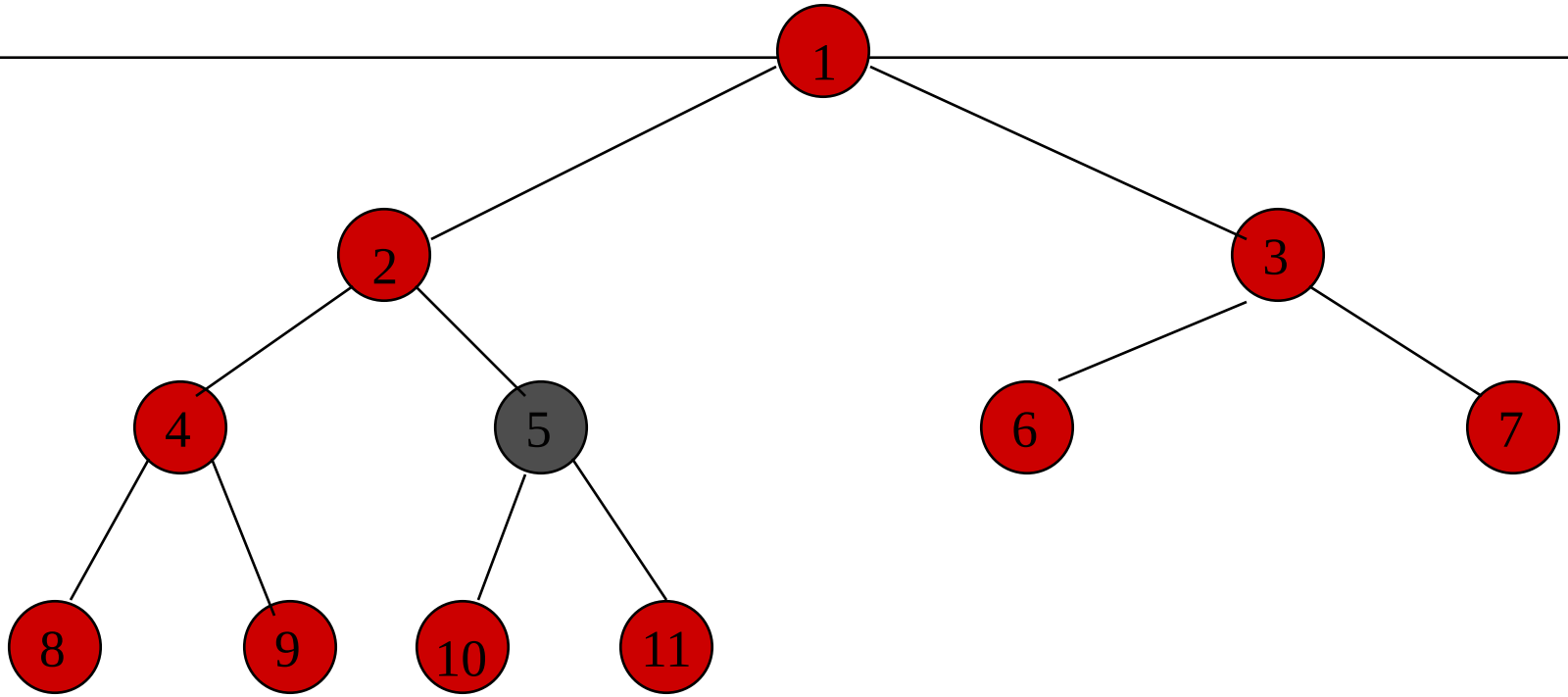
Complexity is $O(\log n)$.

Initializing A Max Heap



input array = [-, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11]

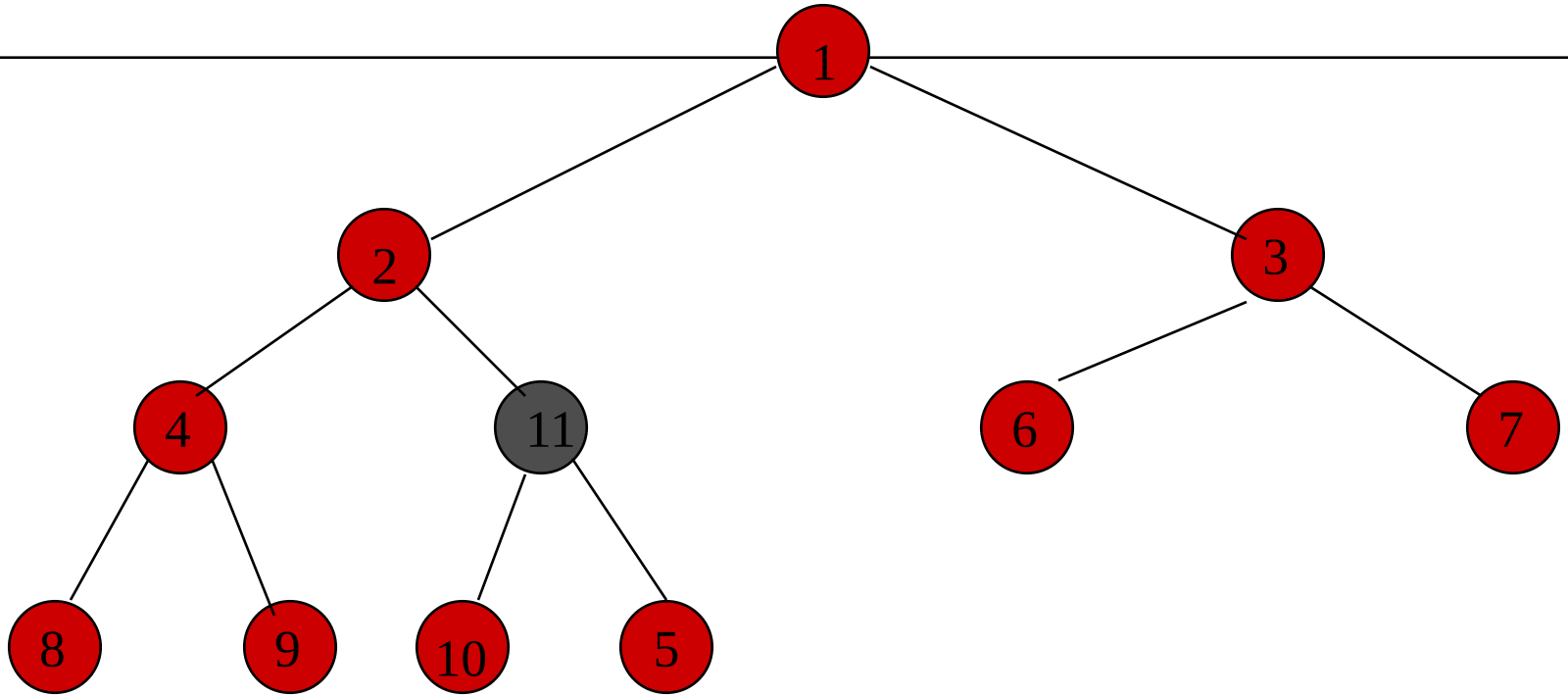
Initializing A Max Heap



Start at rightmost array position that has a child.

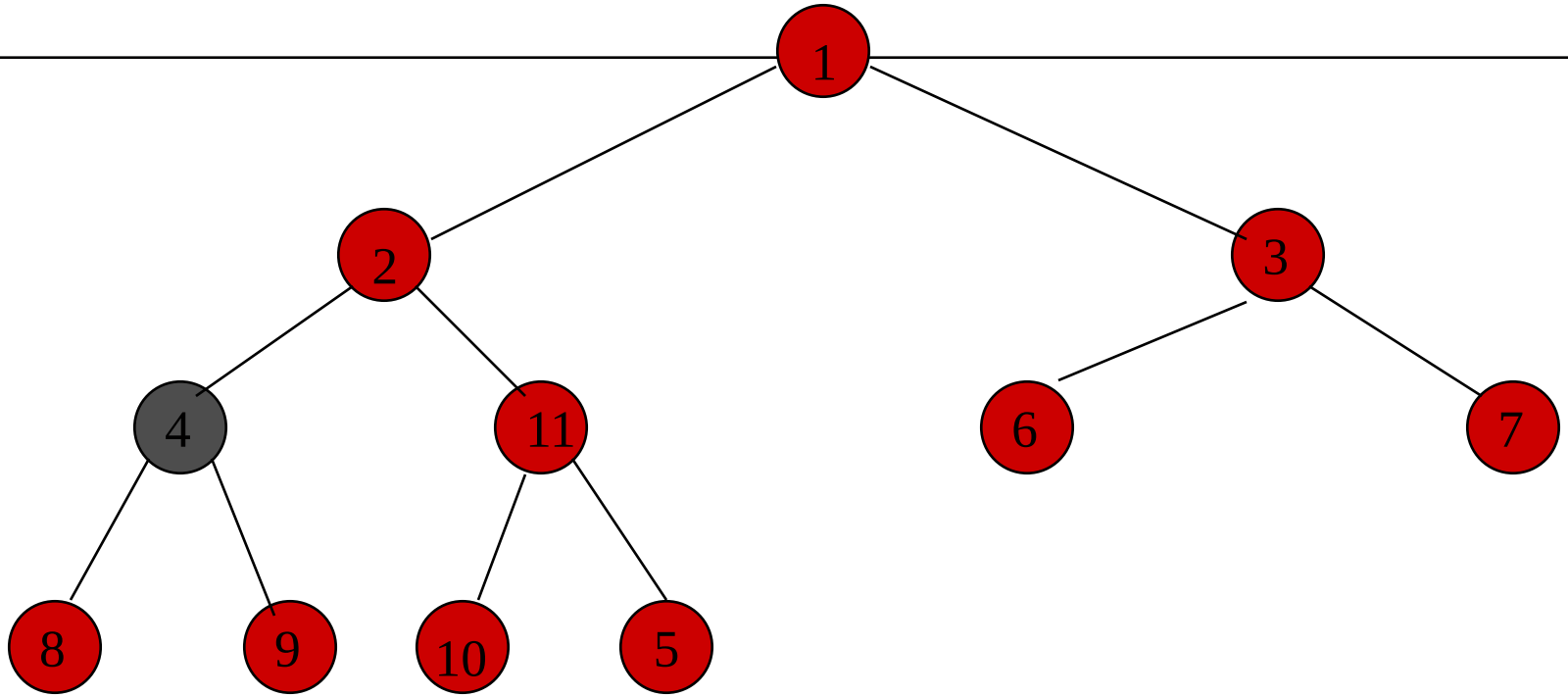
Index is n/2.

Initializing A Max Heap

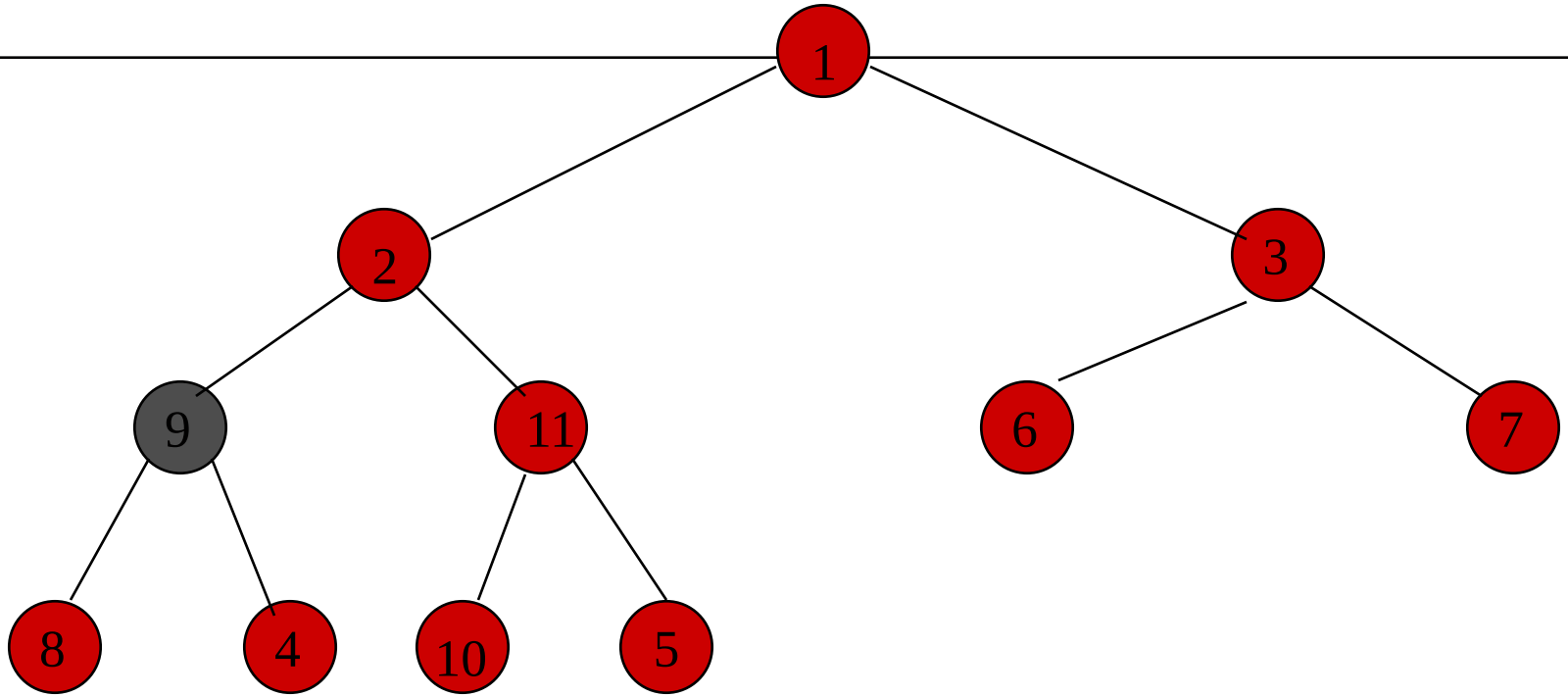


Move to next lower array position.

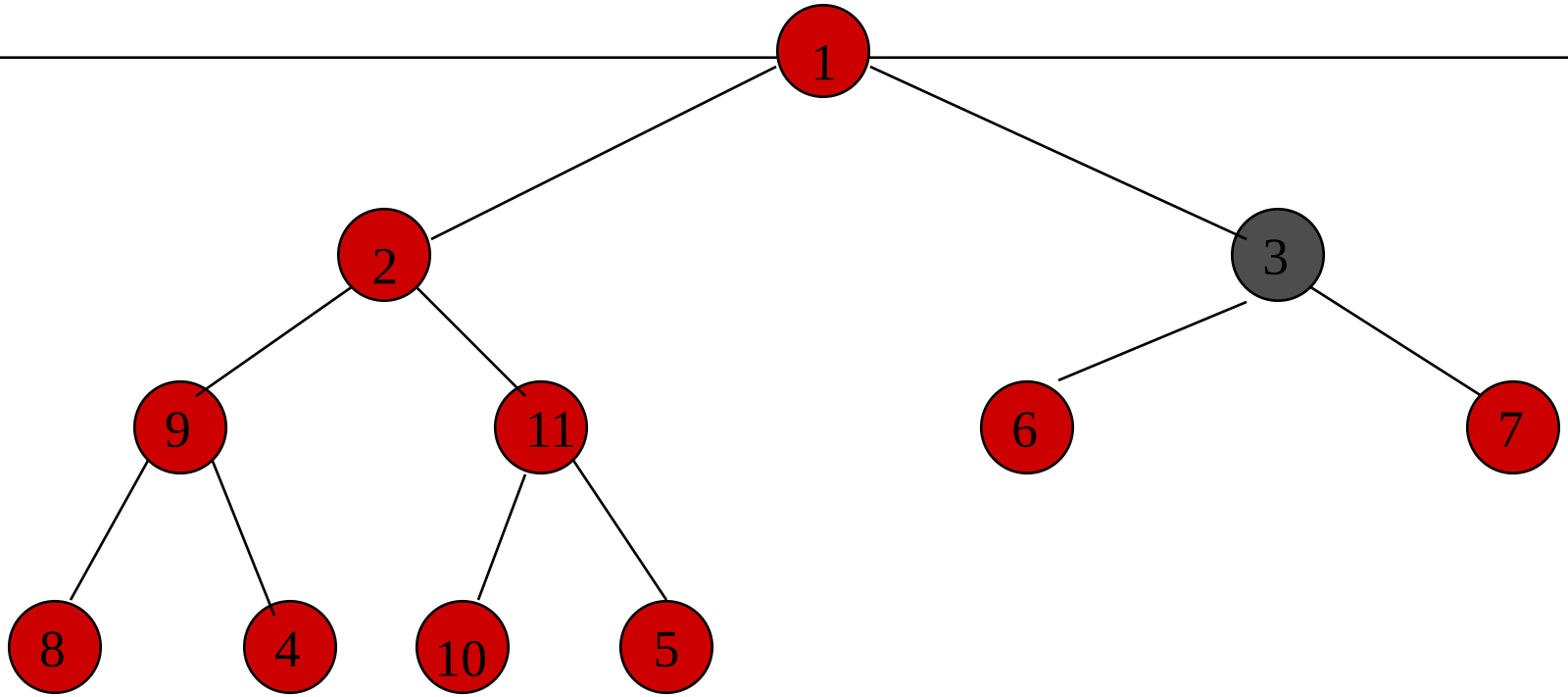
Initializing A Max Heap



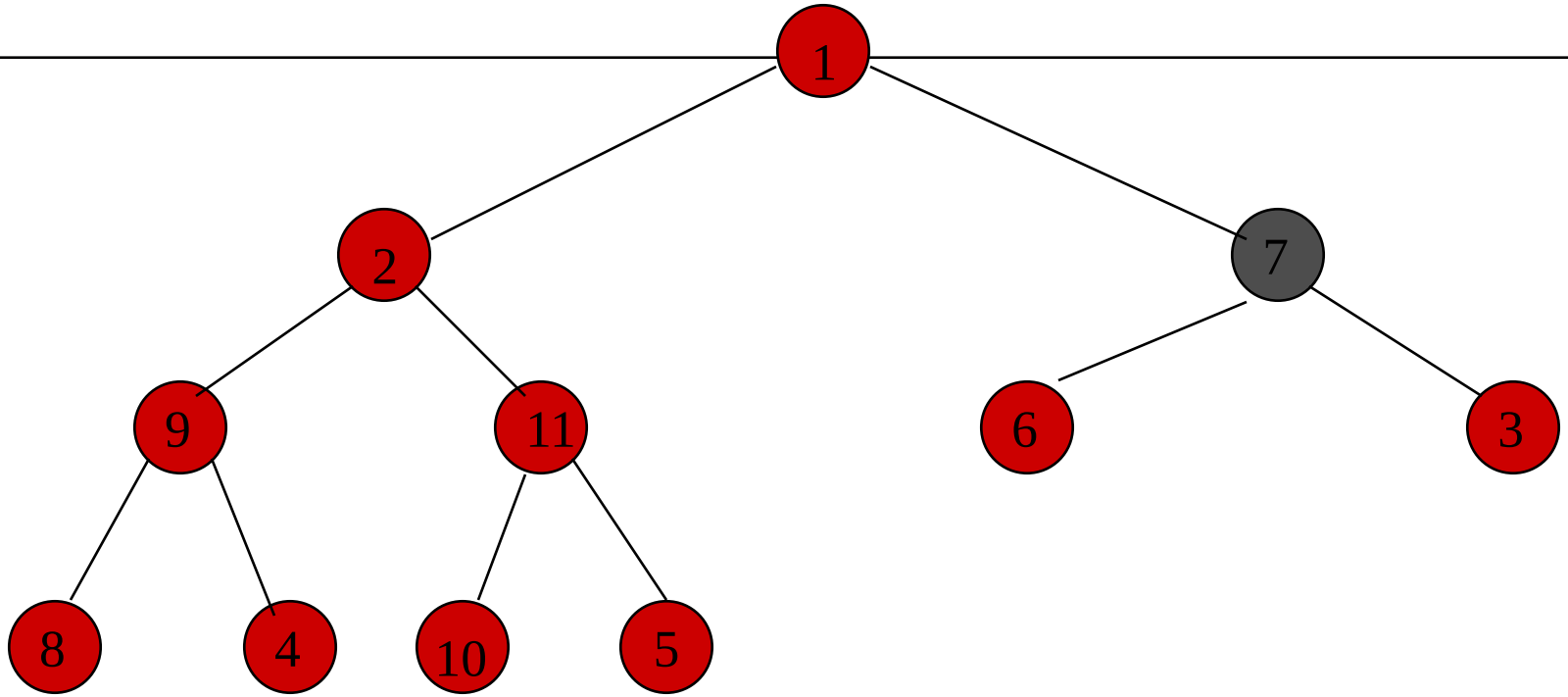
Initializing A Max Heap



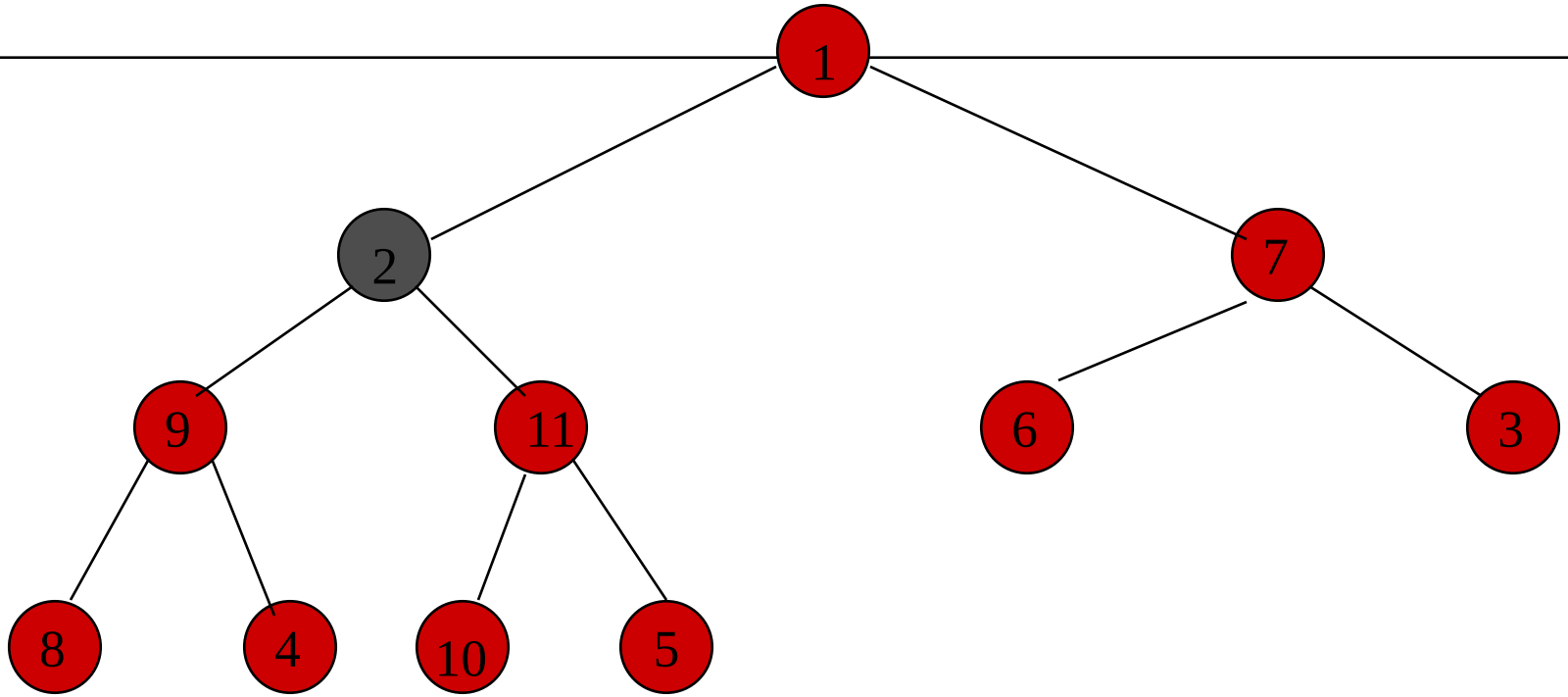
Initializing A Max Heap



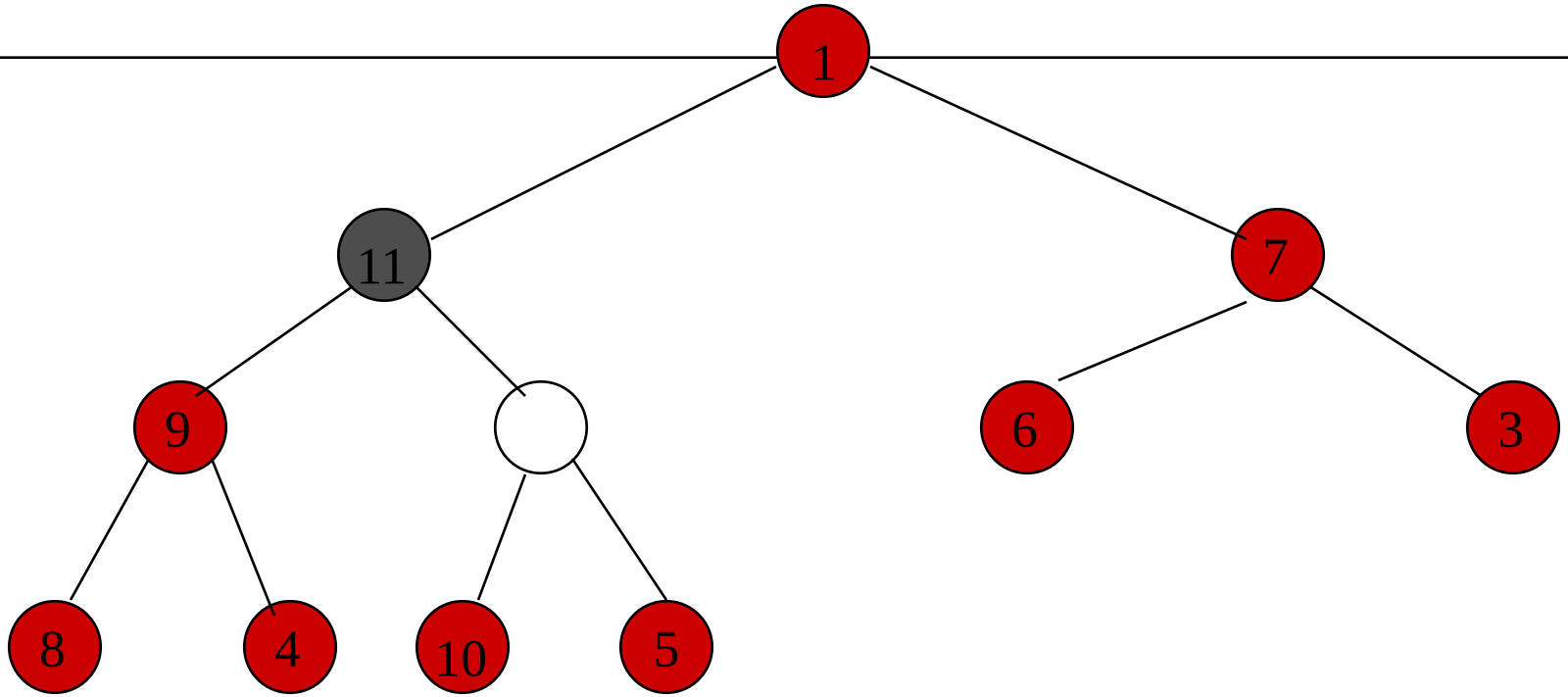
Initializing A Max Heap



Initializing A Max Heap

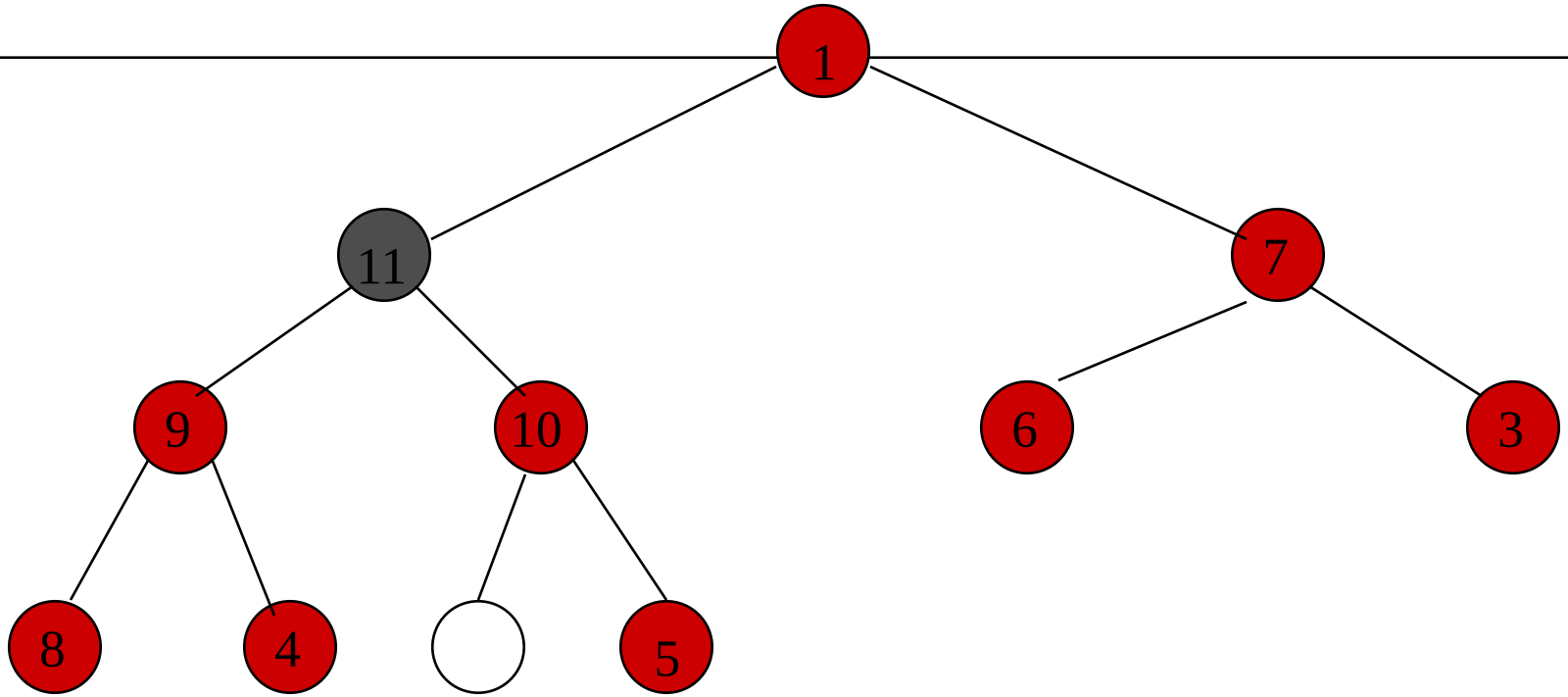


Initializing A Max Heap



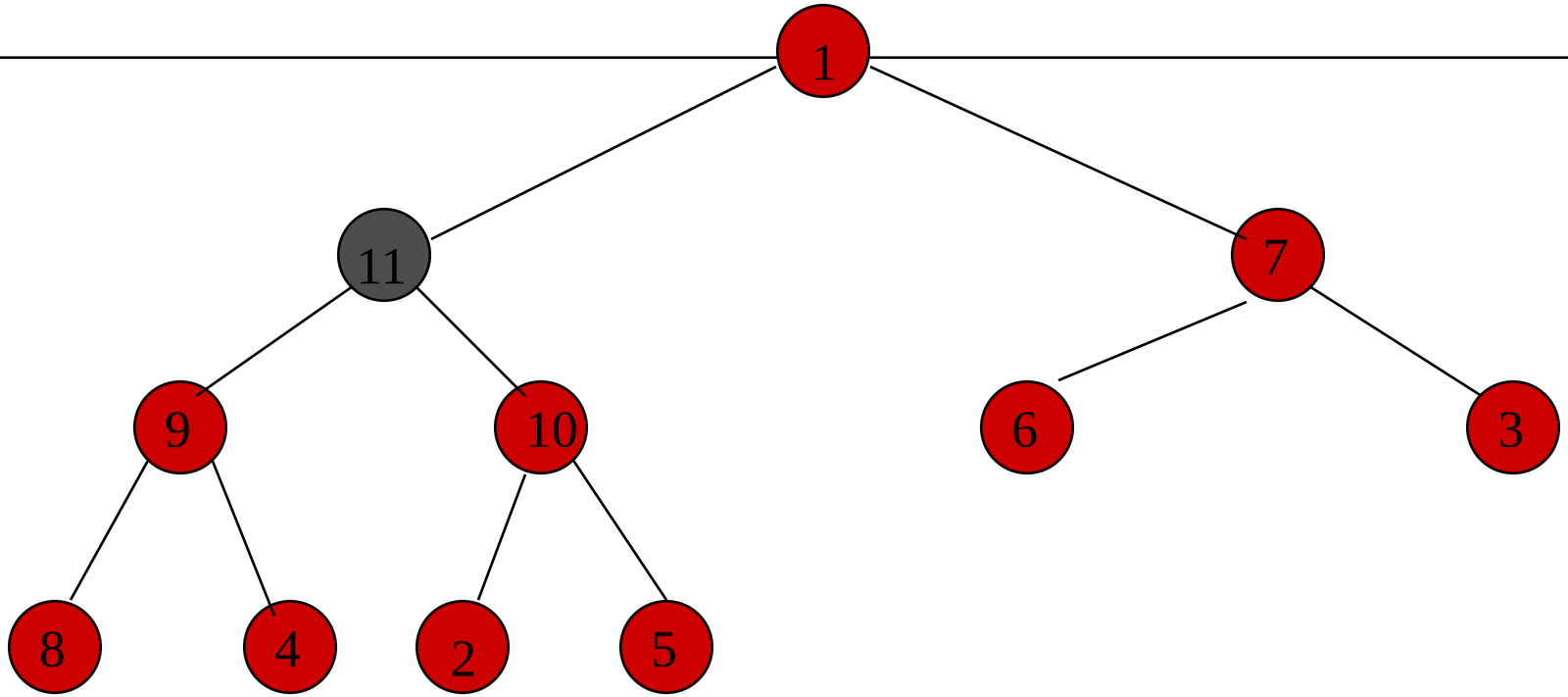
Find a home for 2.

Initializing A Max Heap



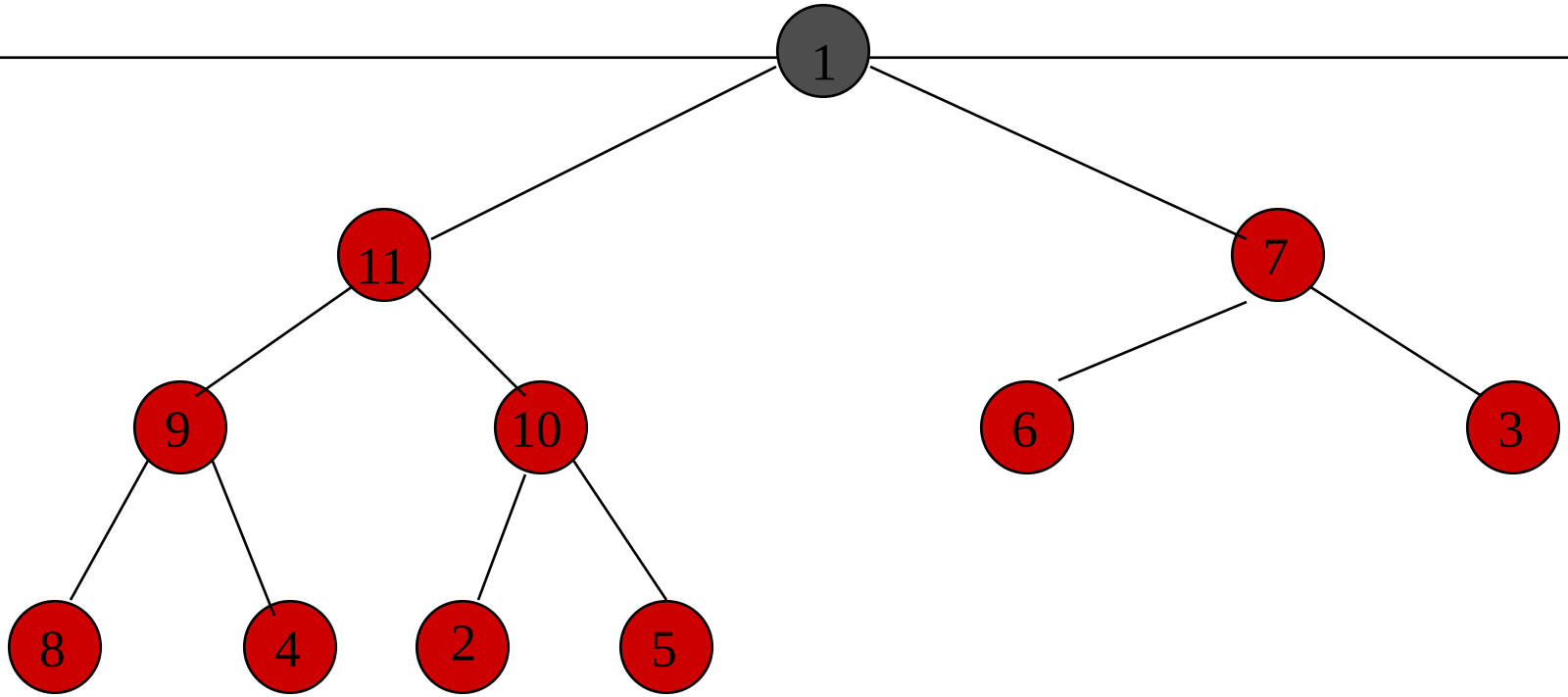
Find a home for 2.

Initializing A Max Heap



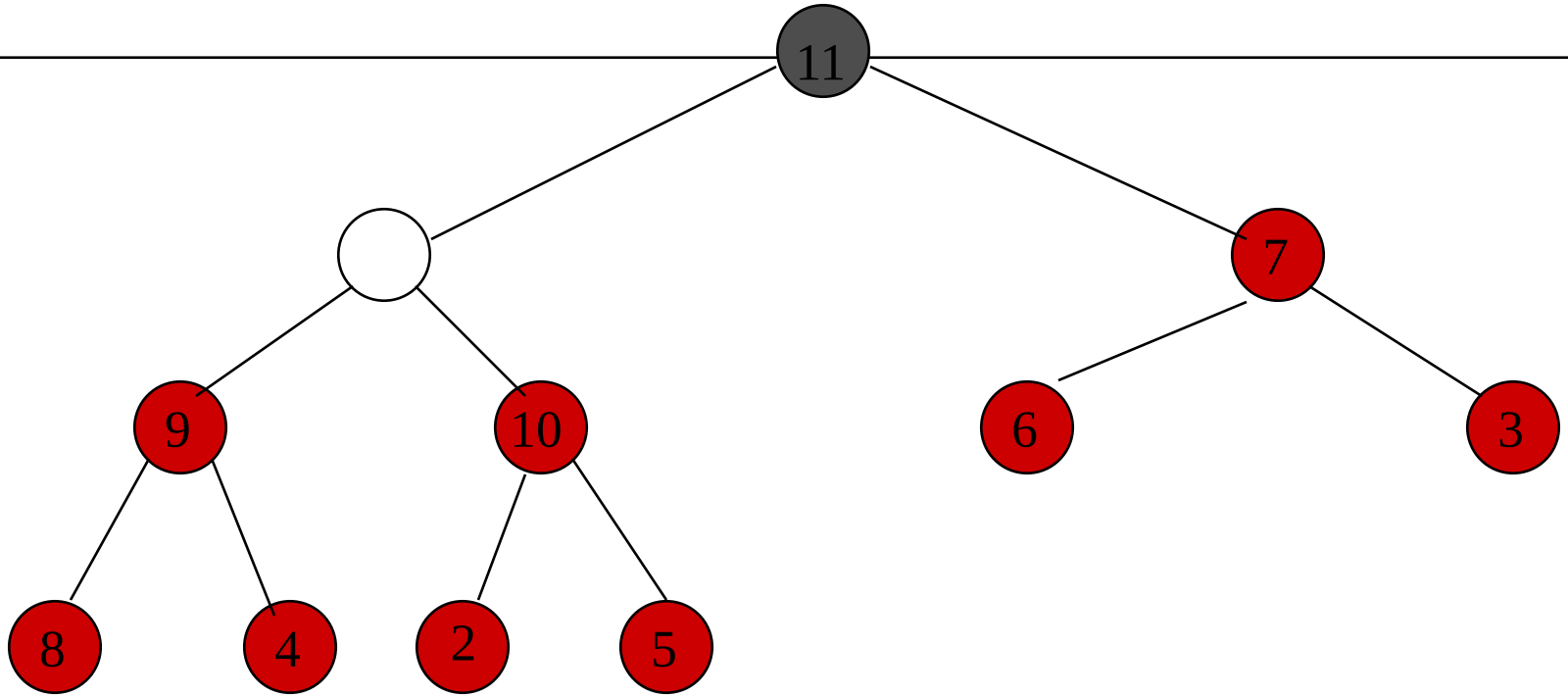
Done, move to next lower array position.

Initializing A Max Heap



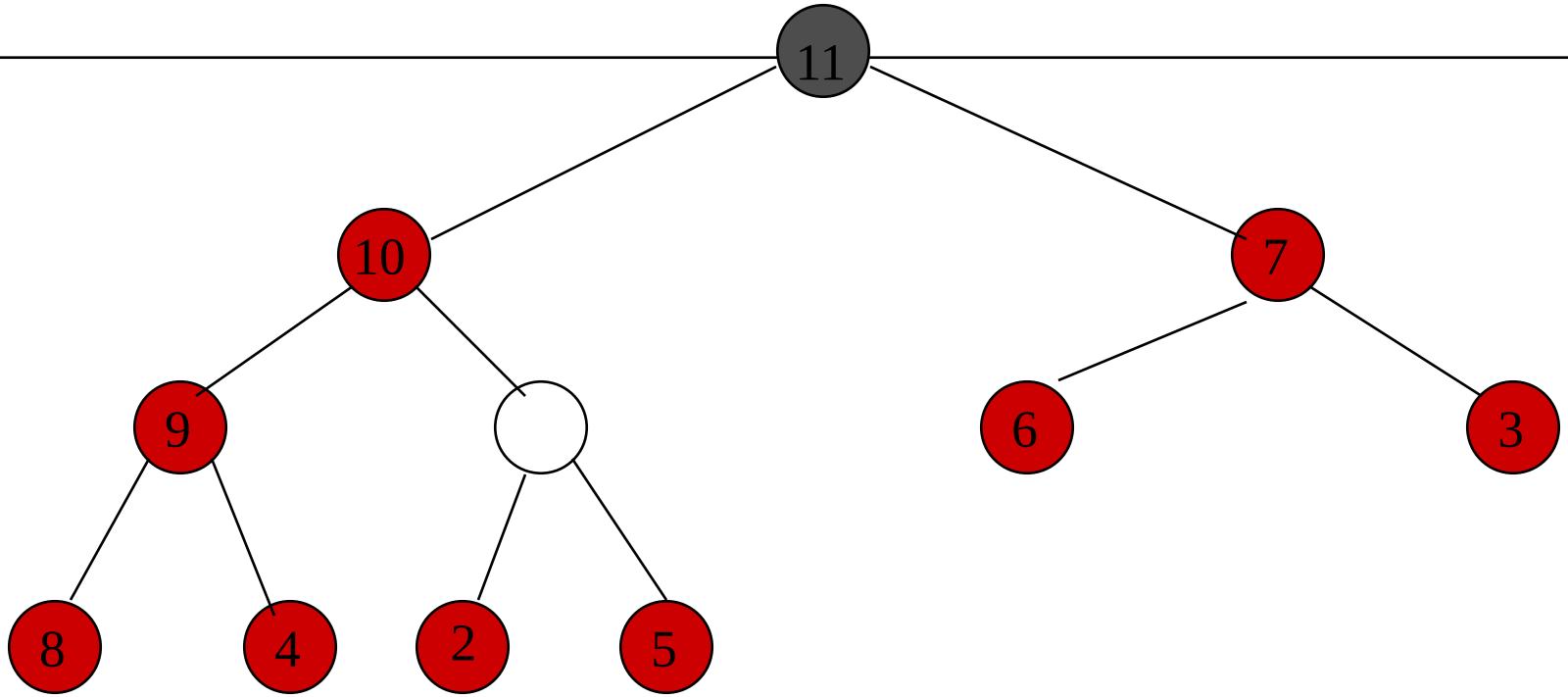
Find home for 1.

Initializing A Max Heap



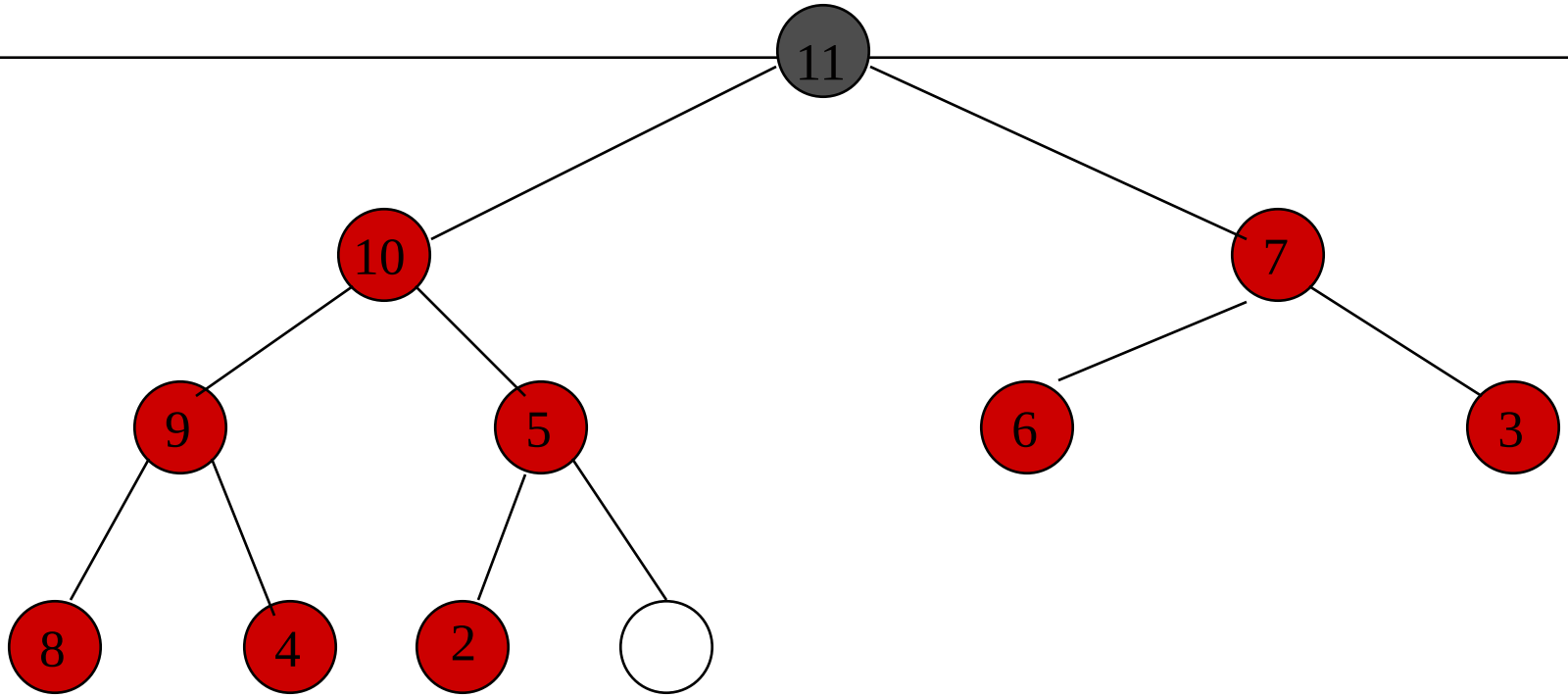
Find home for 1.

Initializing A Max Heap



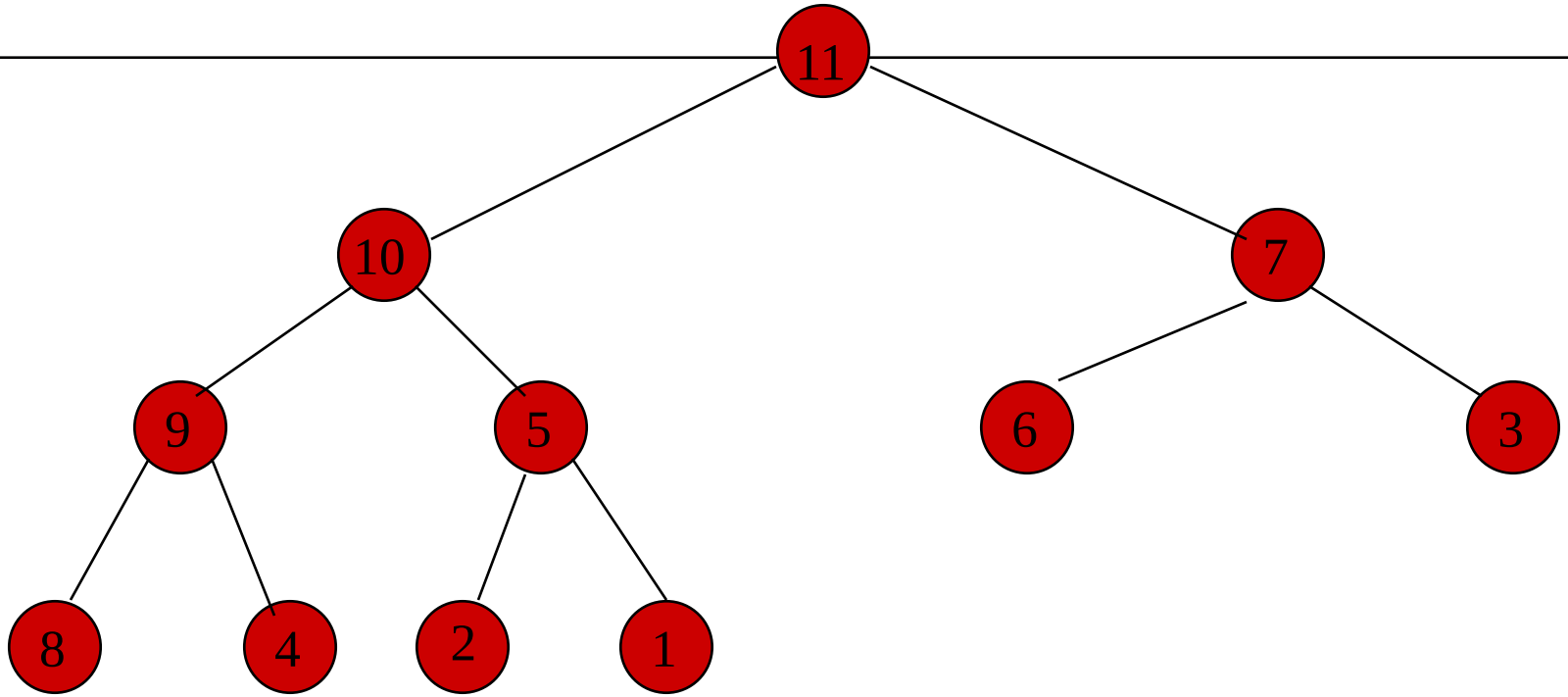
Find home for 1.

Initializing A Max Heap



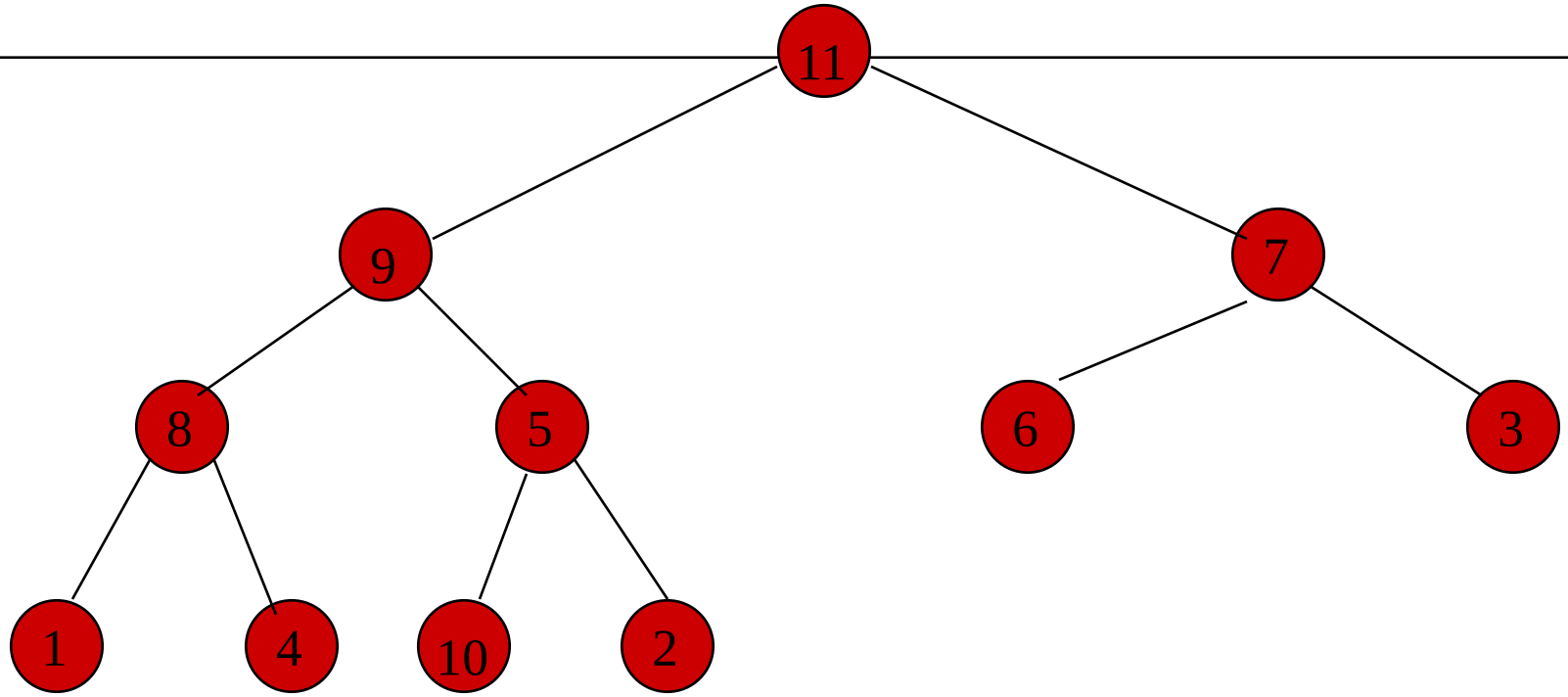
Find home for 1.

Initializing A Max Heap



Done.

Time Complexity



Height of heap = h.

Number of subtrees with root at level j is <= 2^{j-1}.

Time for each subtree is O(h-j+1).

Complexity

Time for level j subtrees is $<= 2^{j-1}(h-j+1) = t(j)$.

Total time is $t(1) + t(2) + \ldots + t(h-1) = O(n)$.

Priority queue operations (Min Heap)

PQ Operation	Array	Binary heap	d-way Heap	Fib heap †
Insert	1/n	log n	$d \log_d n$	1
ExtractMin	n/1	log n	$d \log_d n$	log n
DecreaseKey	1/n	log n	$\log_d n$	1
IsEmpty	1	1	1	1
Size	1	1	1	1

† Individual ops are amortized bounds

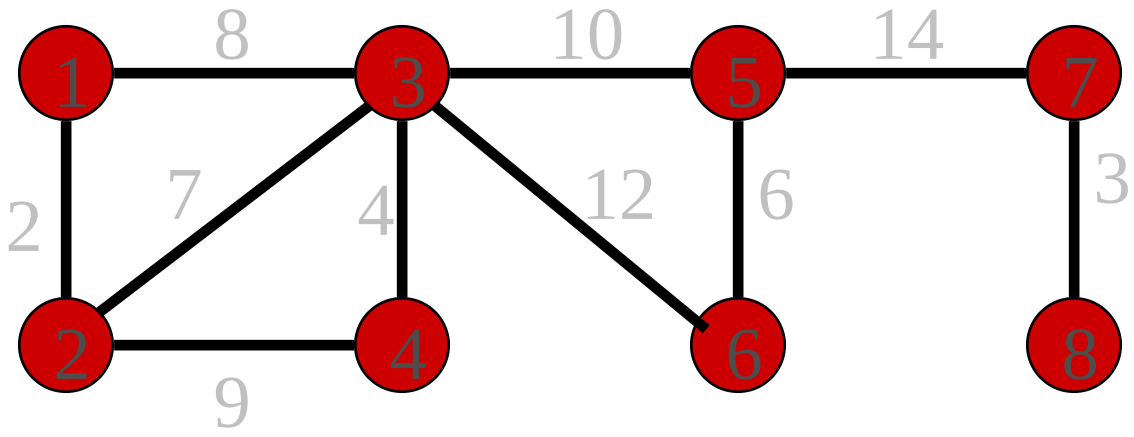
5.5 Minimum Spanning Tree (MST)

Minimum-Cost Spanning Tree

Given a weighted connected undirected graph:

find a spanning tree that has minimum cost
(cost of spanning tree is sum of edge costs)

Example



Network has 10 edges.

Spanning tree has only n - 1 = 7 edges.

Need to either select 7 edges or discard 3.

Edge Selection Greedy Strategies

Kruskal's method.

Start with an n-vertex 0-edge forest. Consider edges in ascending order of cost. Select edge if it does not form a cycle together with already selected edges.

Prim's method.

Start with a 1-vertex tree and grow it into an n-vertex tree by repeatedly adding a vertex and an edge. When there is a choice, add a least cost edge.

Sollin's method.

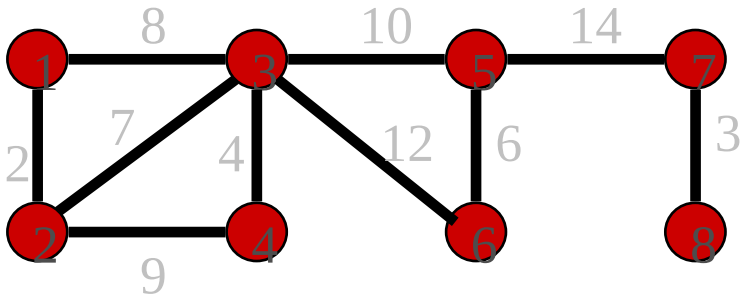
Start with an n-vertex forest. Each component/tree selects a least cost edge to connect to another component/tree. Eliminate duplicate selections and possible cycles. Repeat until only 1 component/tree is left.

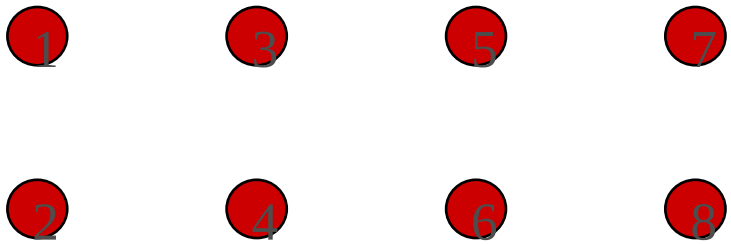
Edge Rejection Greedy Strategies

- Start with the connected graph. Repeatedly find a cycle and eliminate the highest cost edge on this cycle. Stop when no cycles remain.

- Consider edges in descending order of cost. Eliminate an edge provided this leaves behind a connected graph.

Kruskal's Method

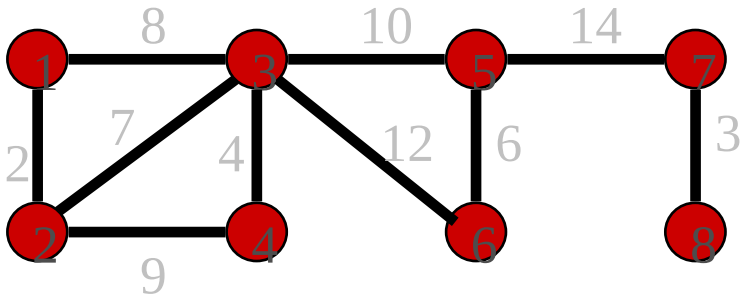


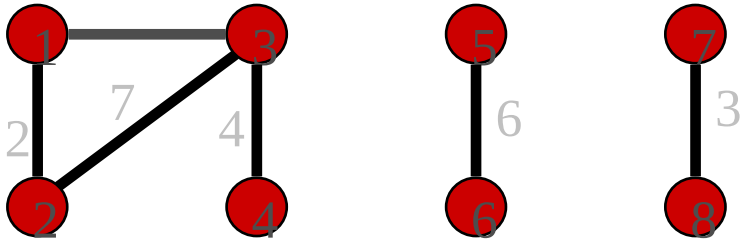


Start with a forest that has no edges.

- Consider edges in ascending order of cost.
- Edge (1,2) is considered first and added to the forest.

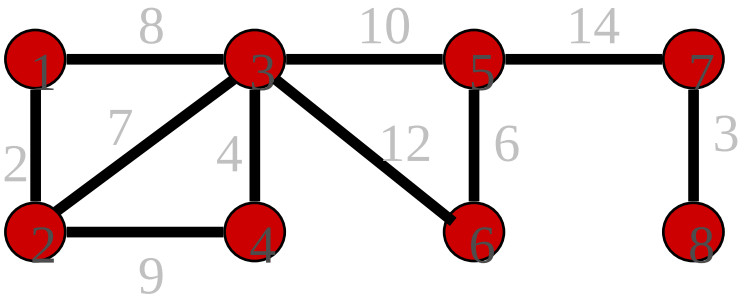
Kruskal's Method

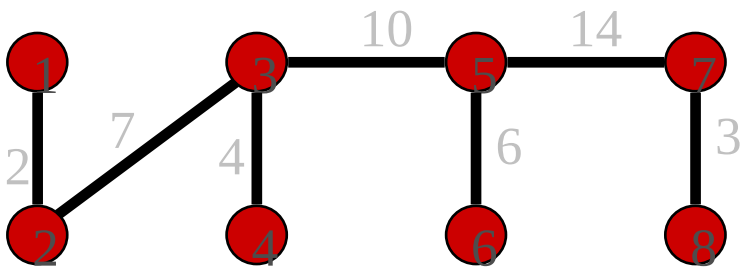




- Edge (7,8) is considered next and added.
- Edge (3,4) is considered next and added.
- Edge (5,6) is considered next and added.
- Edge (2,3) is considered next and added.
- Edge (1,3) is considered next and rejected because it creates a cycle.

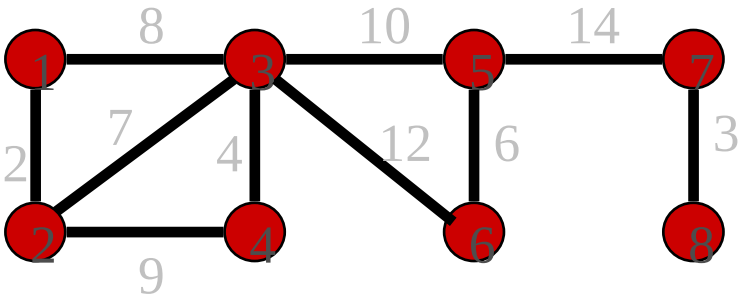
Kruskal's Method

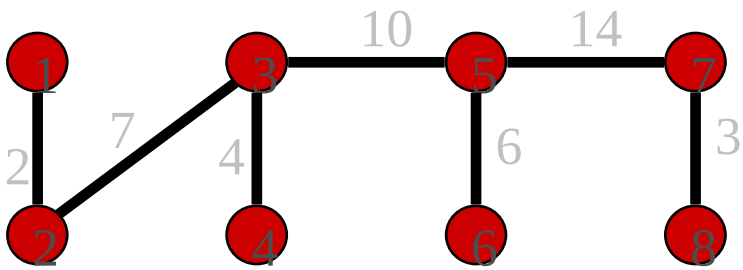




- Edge (2,4) is considered next and rejected.
- Edge (3,5) is considered next and added.
- Edge (3,6) is considered next and rejected.
- Edge (5,7) is considered next and added.

Kruskal's Method





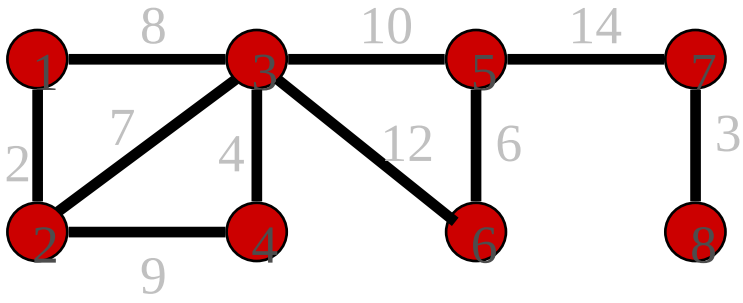
n - 1 edges have been selected and no cycle formed.

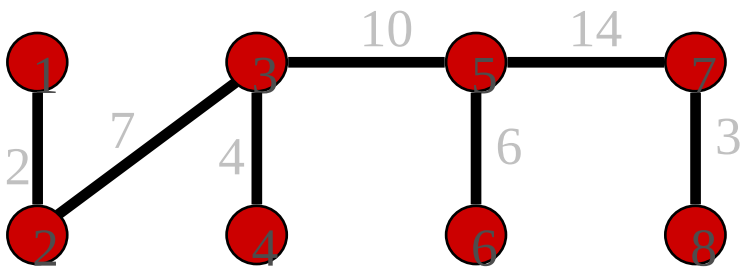
So, we must have a spanning tree.

Cost is 46.

Min-cost spanning tree is unique when all edge costs are different.

Prim's Method

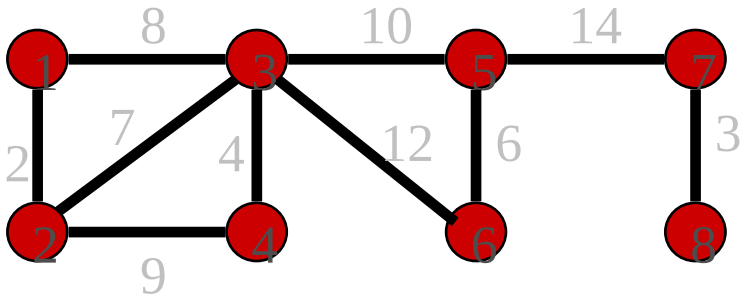


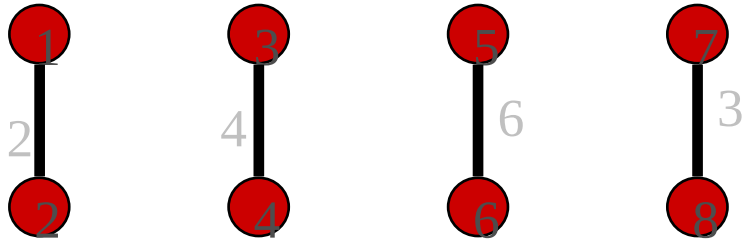


Start with any single vertex tree.

- Get a 2-vertex tree by adding a cheapest edge.
- Get a 3-vertex tree by adding a cheapest edge.
- Grow the tree one edge at a time until the tree has n - 1 edges (and hence has all n vertices).

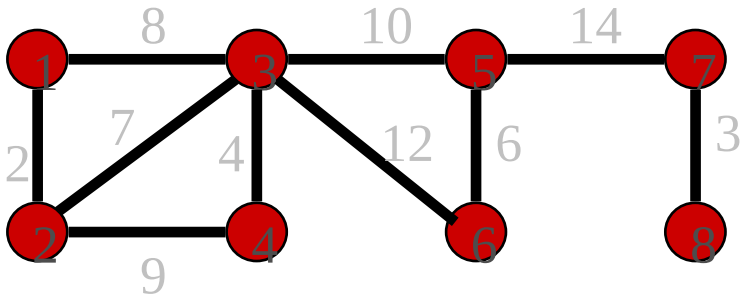
Sollin's Method

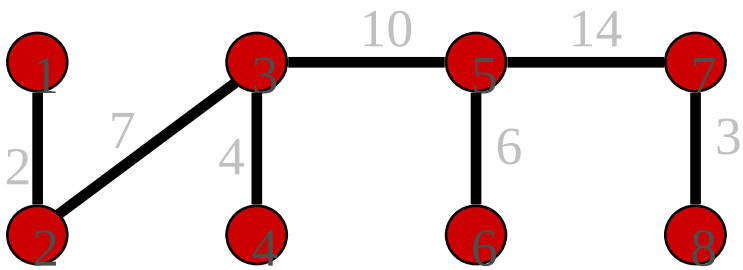




- Start with a forest that has no edges.
- Each component selects a least cost edge with which to connect to another component.
- Duplicate selections are eliminated.
- Cycles are possible when the graph has some edges that have the same cost.

Sollin's Method





- Each component that remains selects a least cost edge with which to connect to another component.

- Beware of duplicate selections and cycles.

Greedy Minimum-Cost Spanning Tree Methods

Can prove that all the algorithms return a minimum-cost spanning tree.

Prim's method is fastest.

- $O(n^2)$ or $O(m \log n)$ using array or binary heap, using an implementation similar to that of Dijkstra's shortest-path algorithm.
- $O(m + n \log n)$ using a Fibonacci heap.

Kruskal's uses union-find trees to run in $O(n + m \log m)$ time.

Pseudocode For Kruskal's Method

```
Start with an empty set T of edges

while (E is not empty && |T| != n-1)
{
    Let (u,v) be a least-cost edge in E
    E = E - {(u,v)}
    if ((u,v) does not create a cycle in T)
       Add edge (u,v) to T
}

if (| T | == n-1) T is a min-cost spanning tree
else Network has no spanning tree
```

Data Structures For Kruskal's Method

Edge set E.

Operations are:

- Is E empty?
- Select and remove a least-cost edge.

Use a min heap of edges.

- Initialize -- O(m) time.
- Remove and return least-cost edge -- O(log m) time.

Data Structures For Kruskal's Method

Set of selected edges T.

Operations are:

- Does T have n - 1 edges?
- Does the addition of an edge (u, v) to T result in a cycle?
- Add an edge to T.

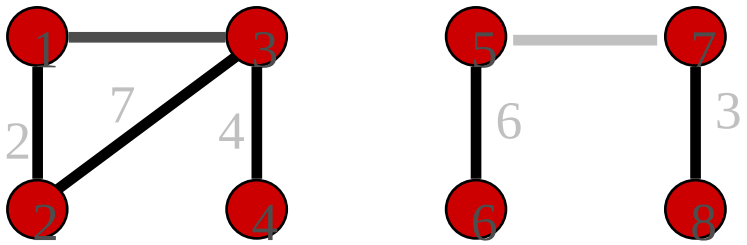
Data Structures For Kruskal's Method

Use an array for the edges of T.

- Does T have n - 1 edges?
  - Check size of the array -- O(1) time.
- Does the addition of an edge (u, v) to T result in a cycle?
  - Not easy.
- Add an edge to T.
  - Add at right end of the array -- O(1) time.

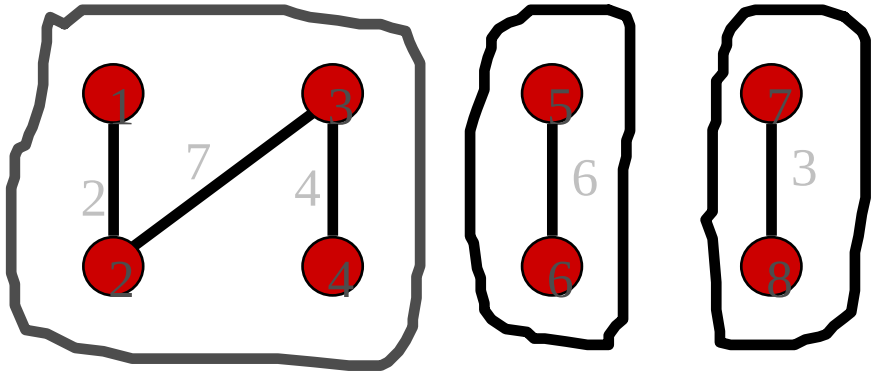
Data Structures For Kruskal's Method

Does the addition of an edge (u, v) to T result in a cycle?



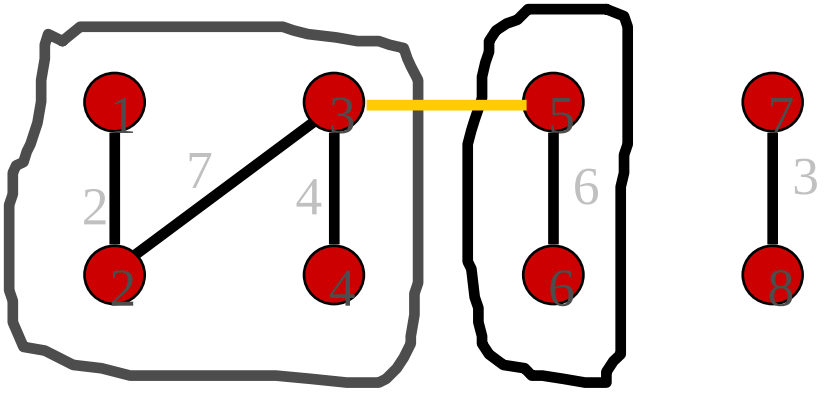
- Each component of T is a tree.
- When u and v are in the same component, the addition of the edge (u,v) creates a cycle.
- When u and v are in the different components, the addition of the edge (u,v) does not create a cycle.

Data Structures For Kruskal's Method



- Each component of T is defined by the vertices in the component.
- Represent each component as a set of vertices.
  - {1, 2, 3, 4}, {5, 6}, {7, 8}
- Two vertices are in the same component iff they are in the same set of vertices.

Data Structures For Kruskal's Method

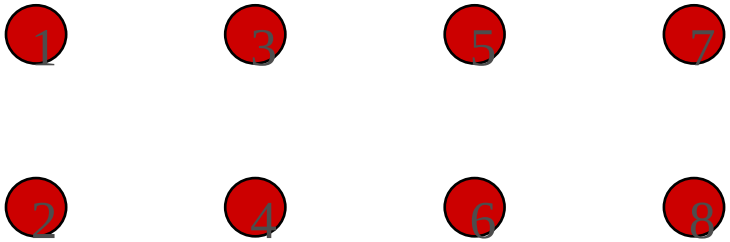


- When an edge (u, v) is added to T, the two components that have vertices u and v combine to become a single component.

- In our set representation of components, the set that has vertex u and the set that has vertex v are united.
  - {1, 2, 3, 4} + {5, 6} => {1, 2, 3, 4, 5, 6}

Data Structures For Kruskal's Method

- Initially, T is empty.



- Initial sets are:
  - {1} {2} {3} {4} {5} {6} {7} {8}
- Does the addition of an edge (u, v) to T result in a cycle? If not, add edge to T.

```
s1 = find(u); s2 = find(v);

if (s1 != s2) union(s1, s2);
```

Data Structures For Kruskal's Method

- Use FastUnionFind.
- Initialize.
  - O(n) time.
- At most 2m finds and n-1 unions.
  - Very close to O(n + m).
- Min heap operations to get edges in increasing order of cost take O(m log m).
- Overall complexity of Kruskal's method is O(n + m log m).

Huffman codes

5.8 Huffman Coding

Fixed length codes

Want to represent data as a sequence of 0's and 1's

For example: BACADAEAFABBAAAGAH

A-000 B-001 C-010 D-011 E-100 F-101 G-110 H-111

001 000 010 000 011 000 100 000 101 000 001 001 000 000 000 110 000 111 (length 54)

This is a fixed length code.

Can we make the sequence of 0's and 1's shorter ?

Variable length codes

A 0 B 100 C 1010 D 1011

E 1100 F 1101 G 1110 H 1111

10001010010110110001101010010000111001111

(length 42)

This is a variable length code.

How do we decode ?

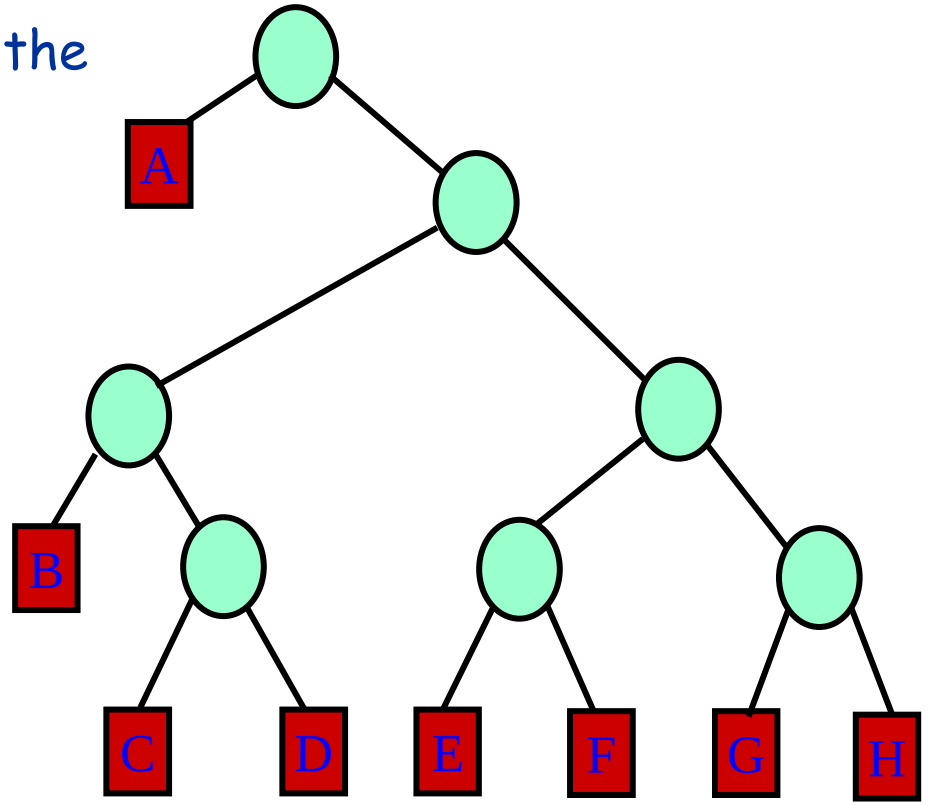
Use prefix codes: No codeword is a prefix of the other

Representing prefix codes

A 0 B 100 C 1010 D 1011

E 1100 F 1101 G 1110 H 1111

A prefix code corresponds to a binary tree with the symbols at the leaves and vice versa.



Construction Algorithm

Compute the character frequencies

Insert the frequencies into a min heap

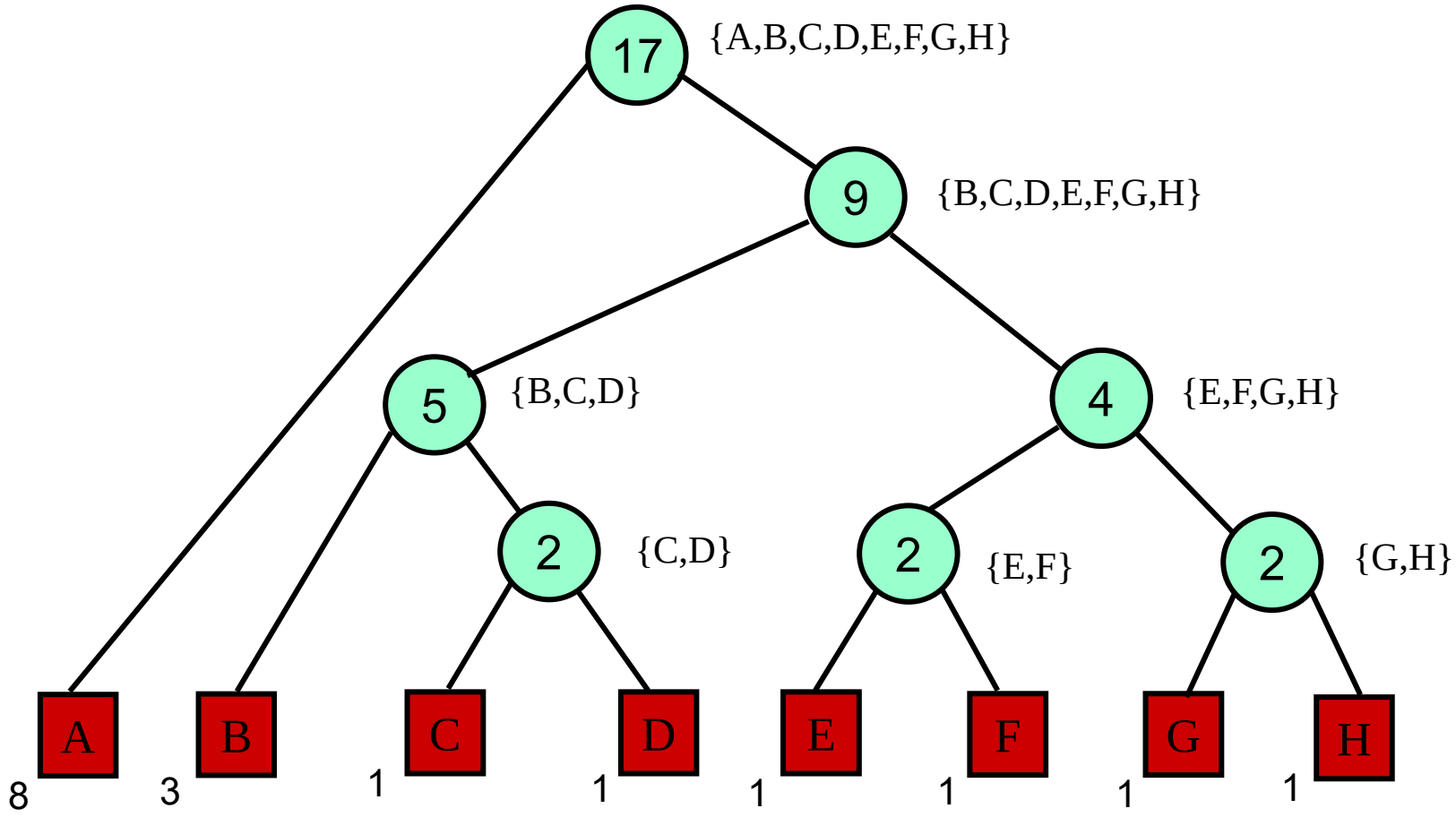
Repeat

- Delete the two minimum elements, f1 and f2 from the heap
- insert f1+f2 into the heap

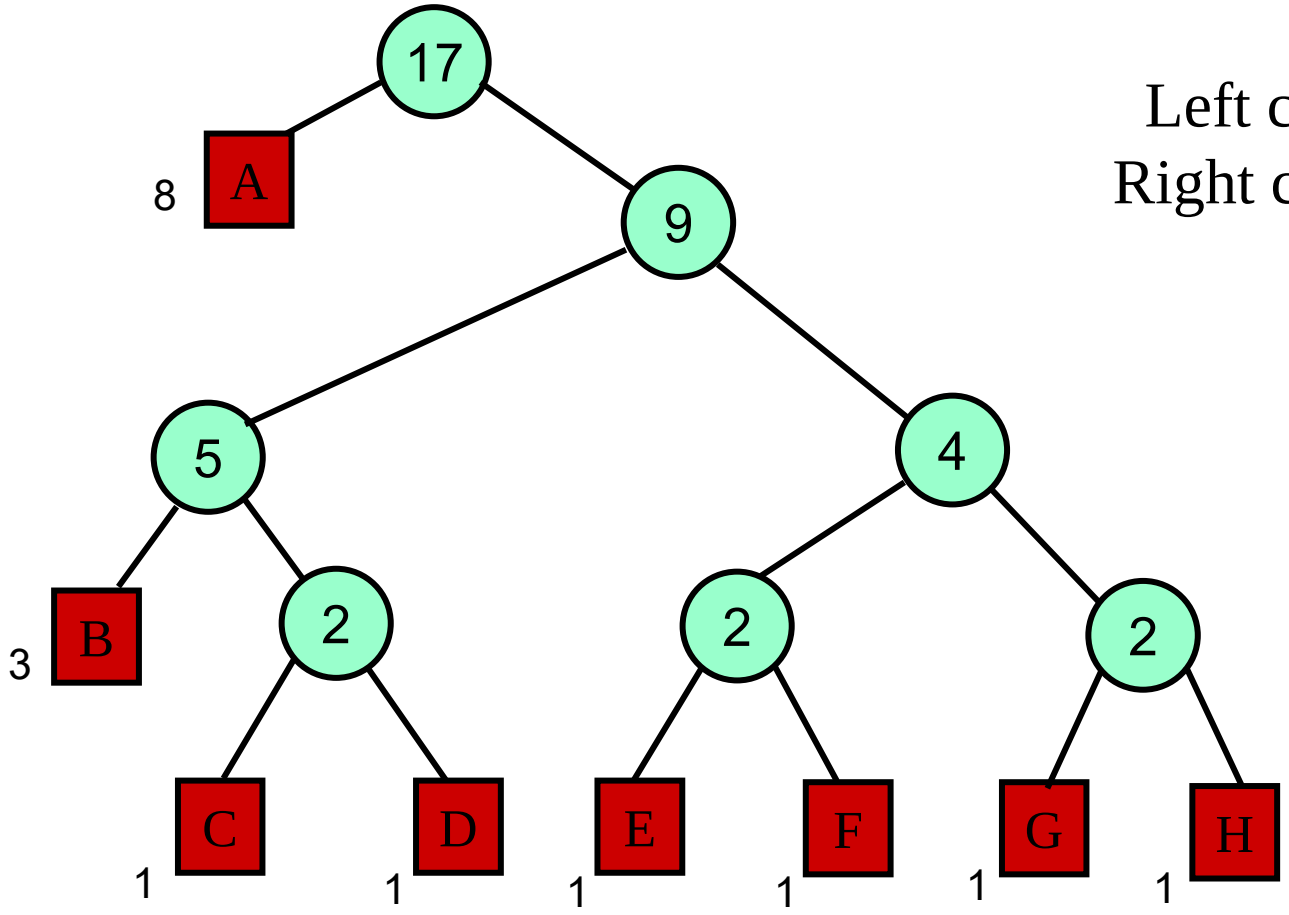
Complexity: $O(n \log n)$.

Theorem: Huffman algorithm is optimal for a symbol-by-symbol coding.

Construction of Huffman tree



Representation



Left child – 0
Right child – 1

code(D) = 1011; code(F) = 1101