6.006 Recitation

Build 2008.6

Outline

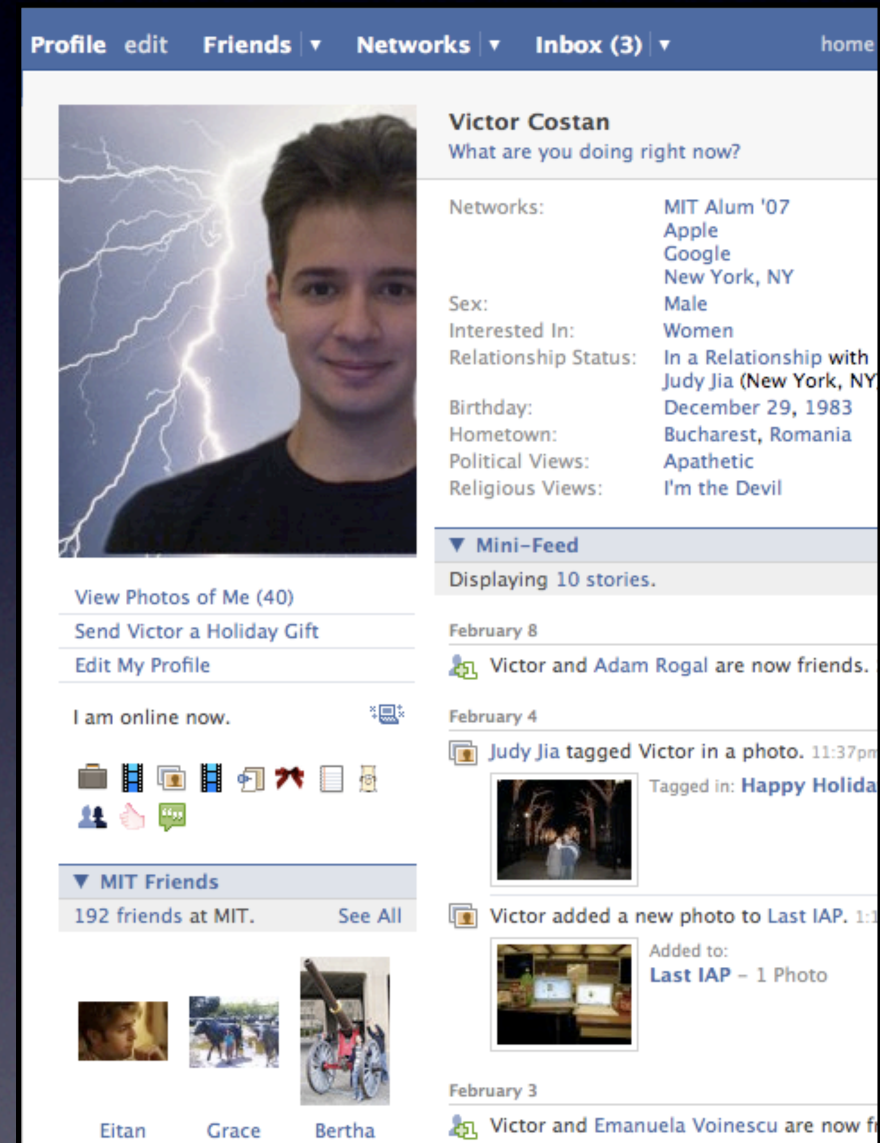
- The Lame Speech (Where & Why)
- Binary Search Trees
  - Principles
  - Algorithms & Python Code
- Augmenting Binary Search Trees
  - Rank computation

Orientation

- Binary Search Trees (BSTs)
  - Time/op: O(lg(N)) avg, O(N) max
- Balanced BSTs
  - Time/op: O(lg(N)) guaranteed
- Hash Tables
  - Time/op: O(1) avg, O(N) worst

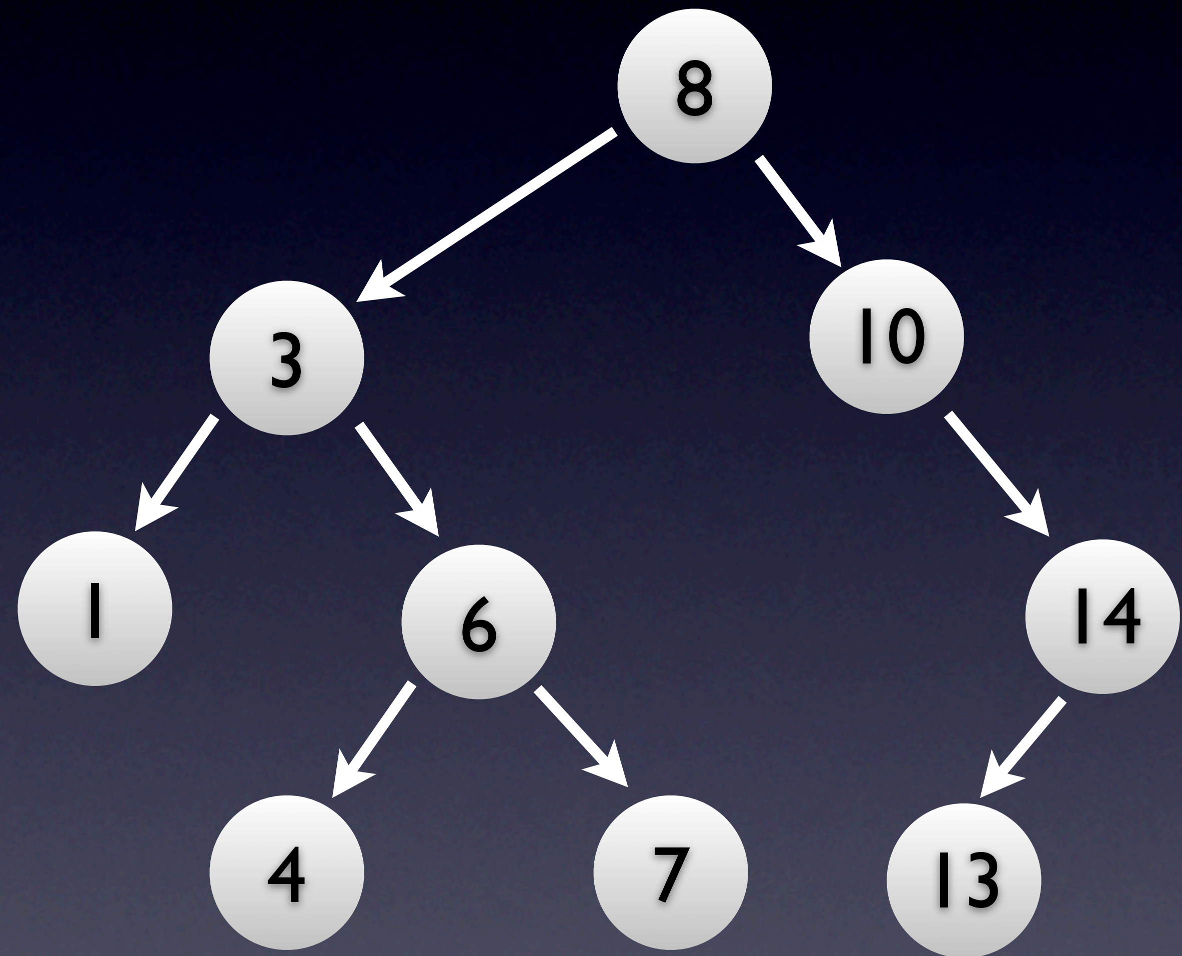
Motivation: Web Sites

- Many millions of DAILY visitors, billions of queries (searches)
- Run on SQL databases
- SQL indexes are mainly
  - Tree indexes
  - Hash indexes



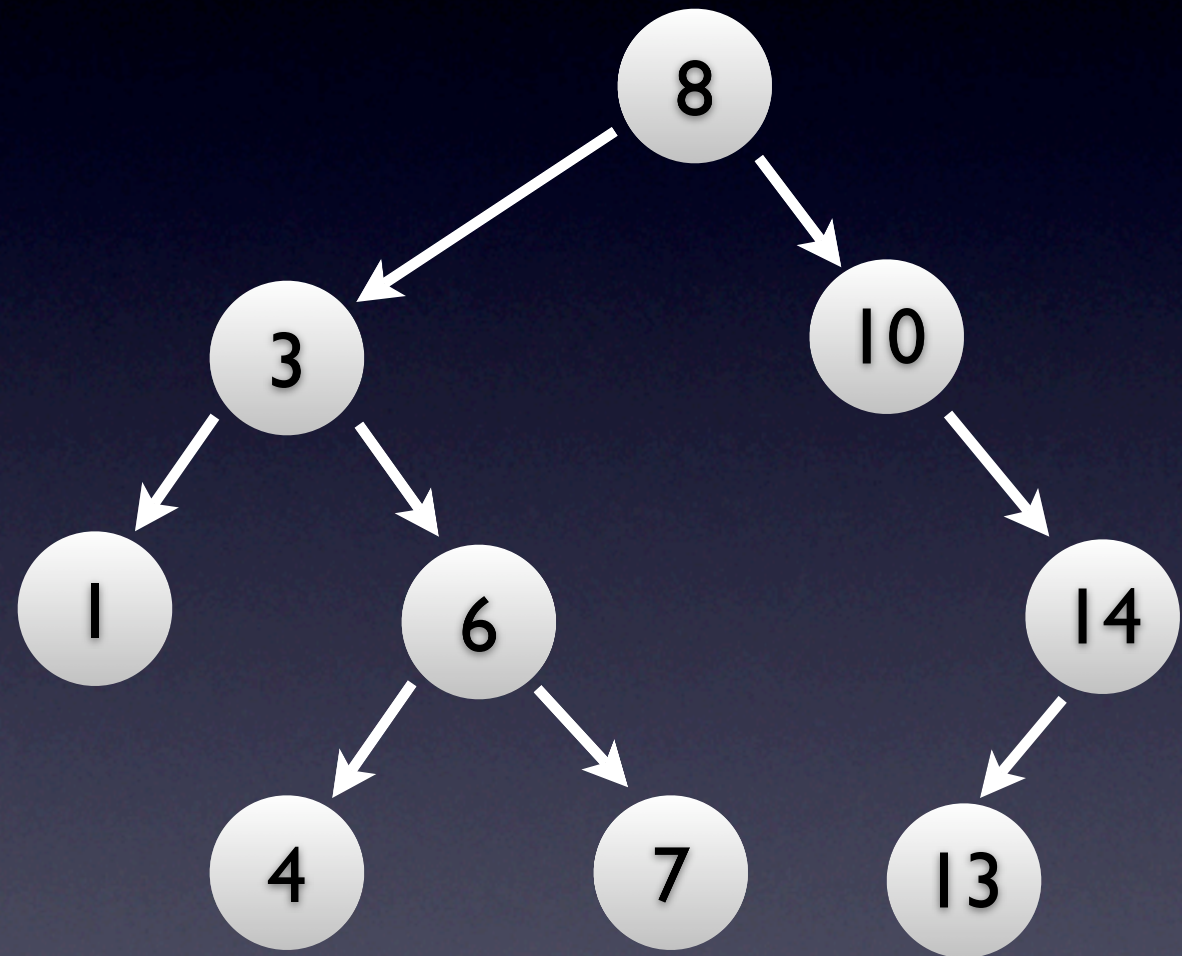
BST Invariants

- Binary rooted tree
- All left descendants have keys < node's key
- All right descendants have keys > node's key

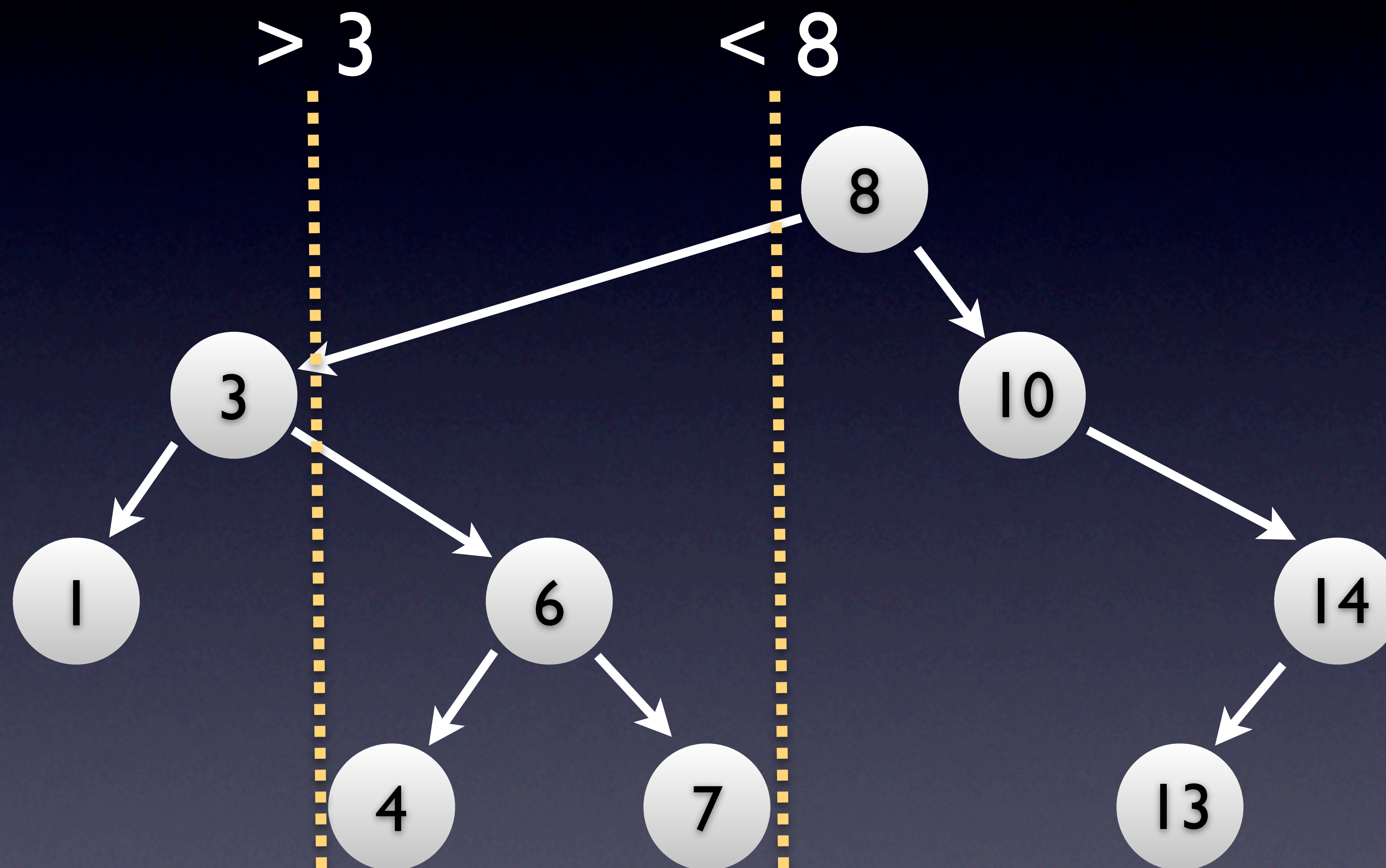


BST Conclusions

- No key shows up twice
- Each subtree contains all and only the keys with values in an interval
  - left subtree: upper bound
  - right subtree: lower bound



BST Subtree Intervals



Invariants Rock!

Invariants Rock!

You can mess up a BST infinitely; as long as you maintain the invariants, it works

Algorithms & Python

‘cause you need to know how to build this

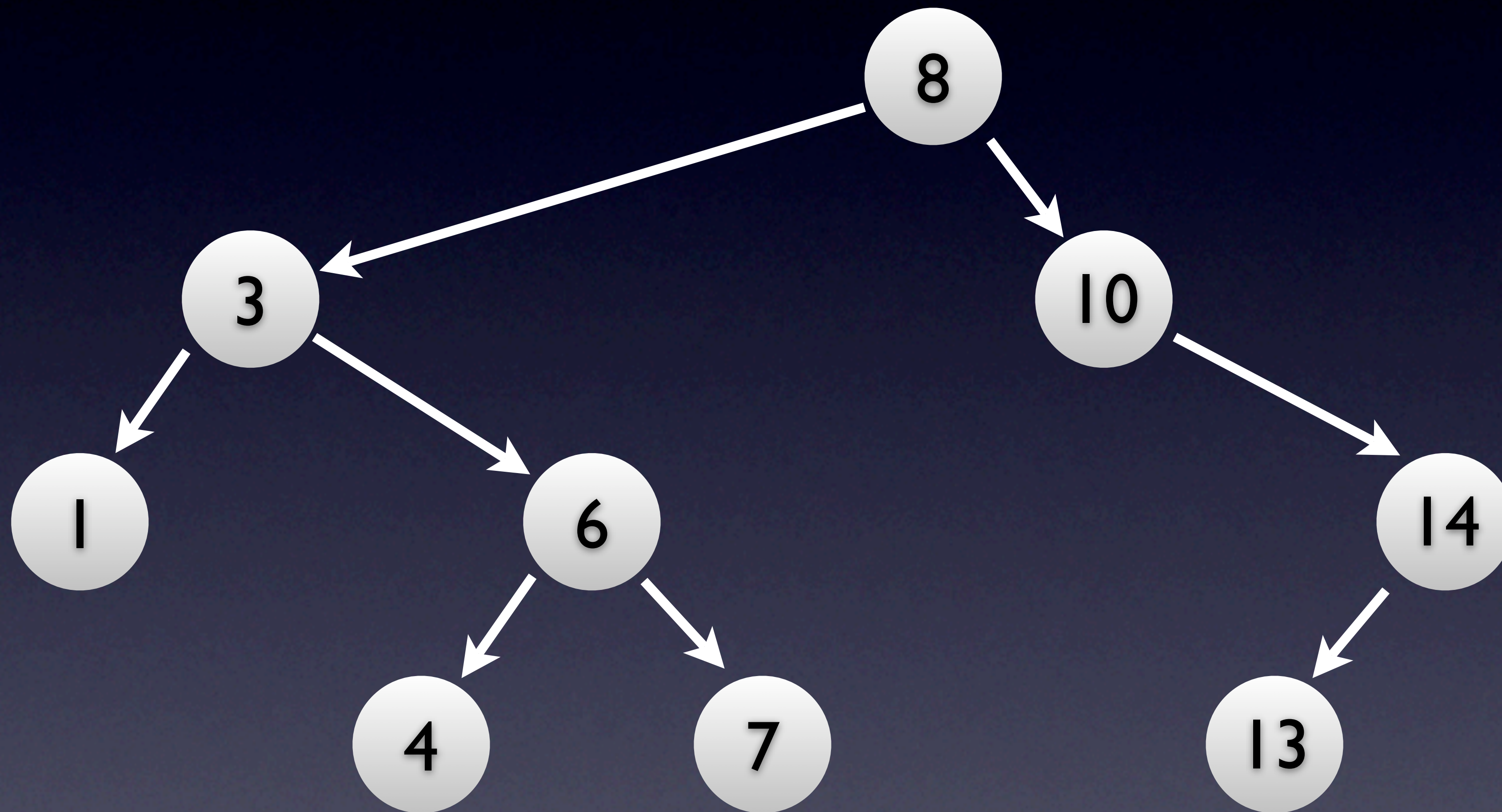
BST Design

- BSTnode
  - attributes: key, children (left & right)
  - methods: insert, find (subtree rooted at)
- BST
  - attributes: root of the tree
  - methods: same as above

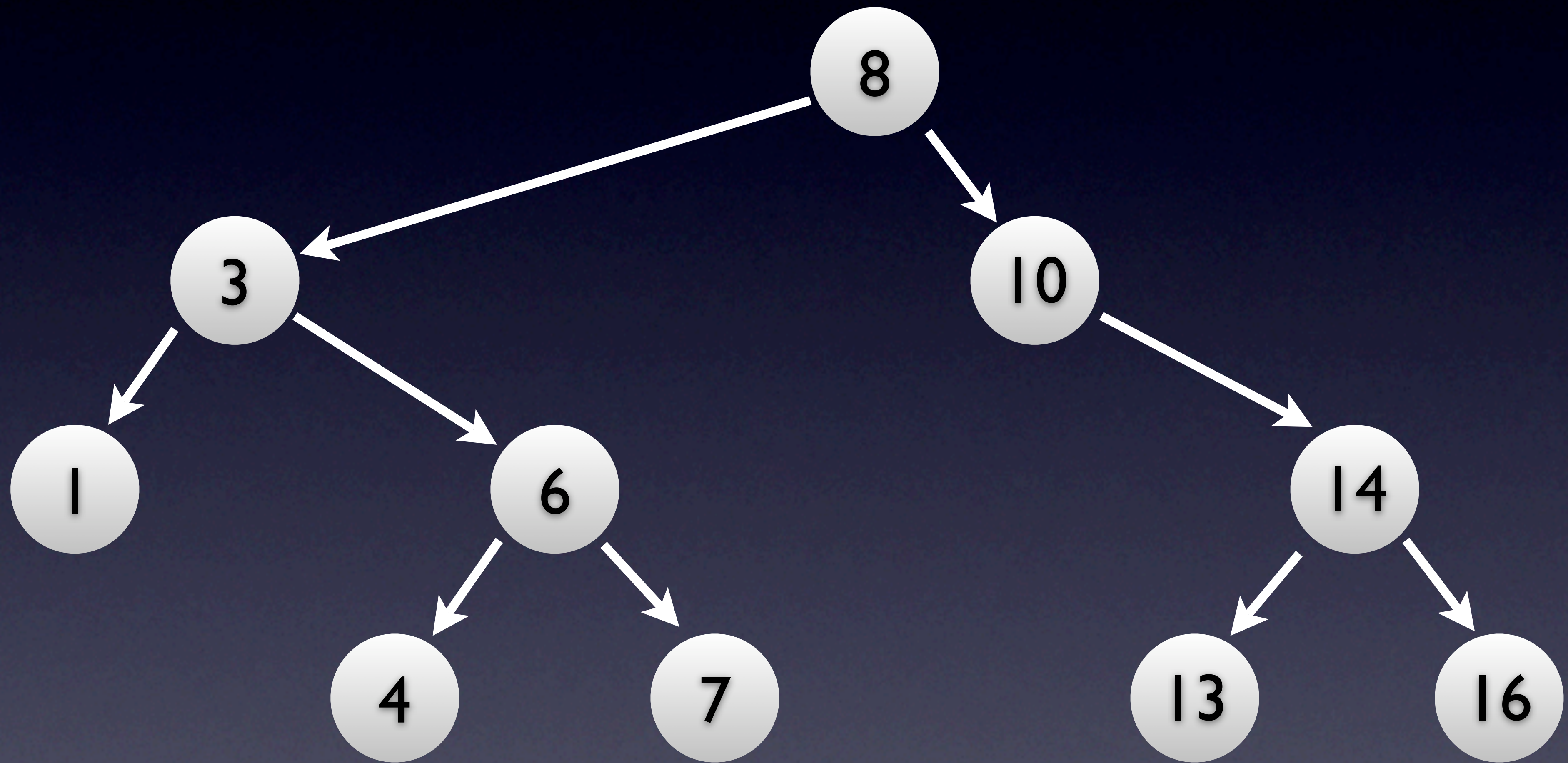
BST Search

```
1 class BSTnode(object):
2    def __init__(self, parent, t):
3        self.key = t
3        self.parent = parent
4        self.left = None
5        self.right = None
6
7    def find(self, t):
8        if t == self.key:
9             return self
10       elif t < self.key:
11           if self.left is None:
12               return None
13           else:
14               return self.left.find(t)
15       else:
16           if self.right is None:
17               return None
18           else:
19               return self.right.find(t)
```

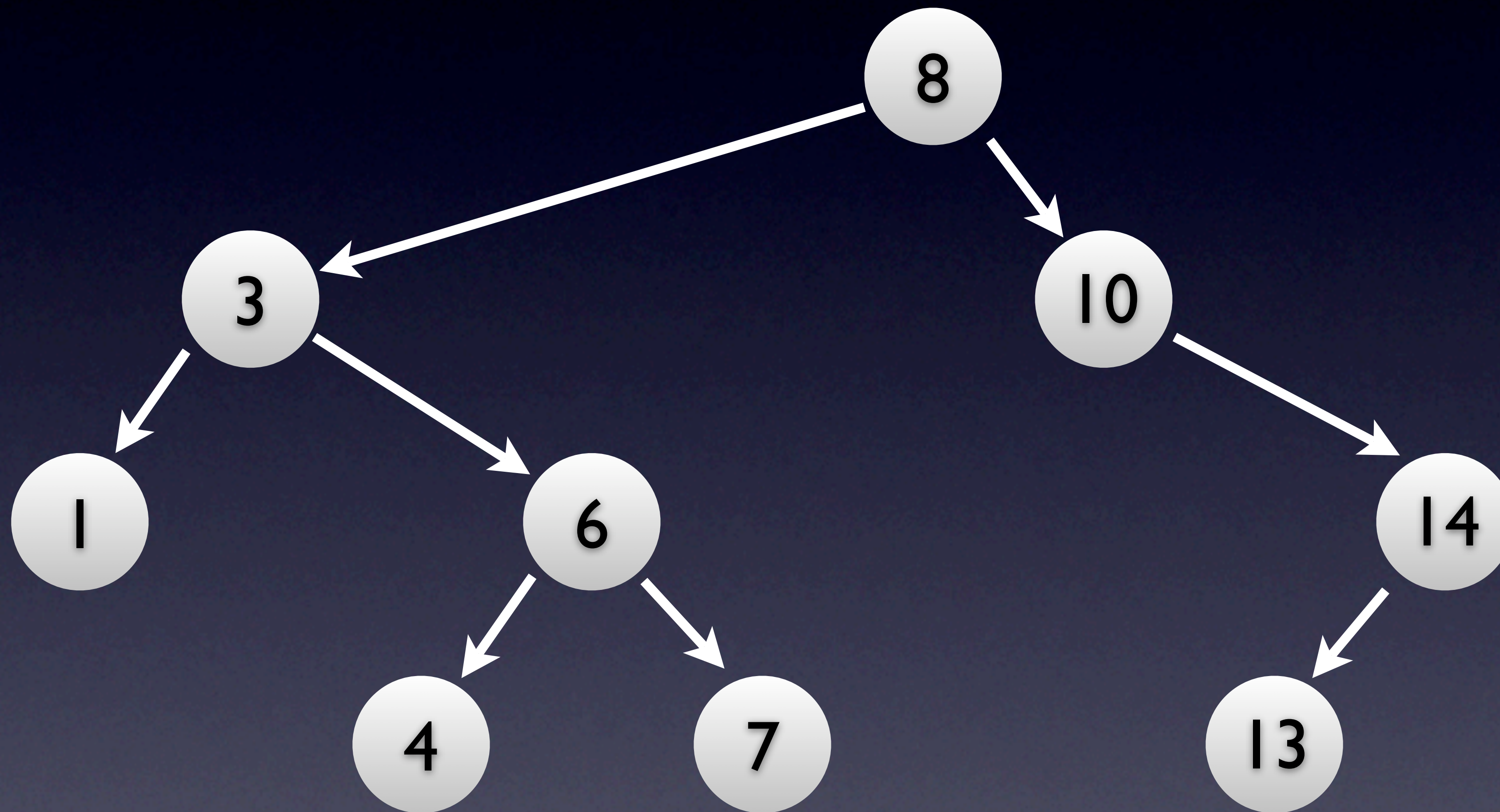
Insert 16



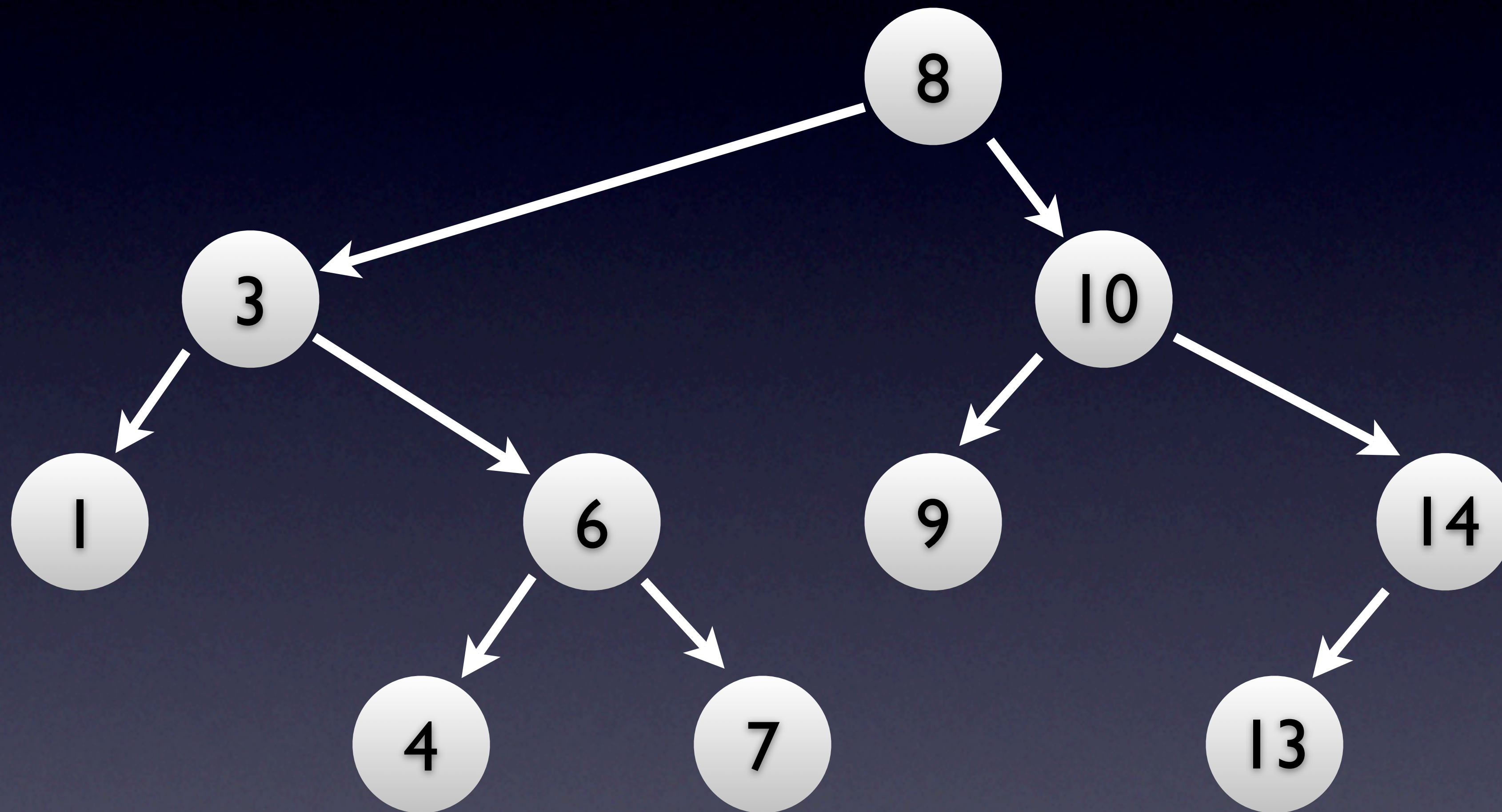
Insert 16



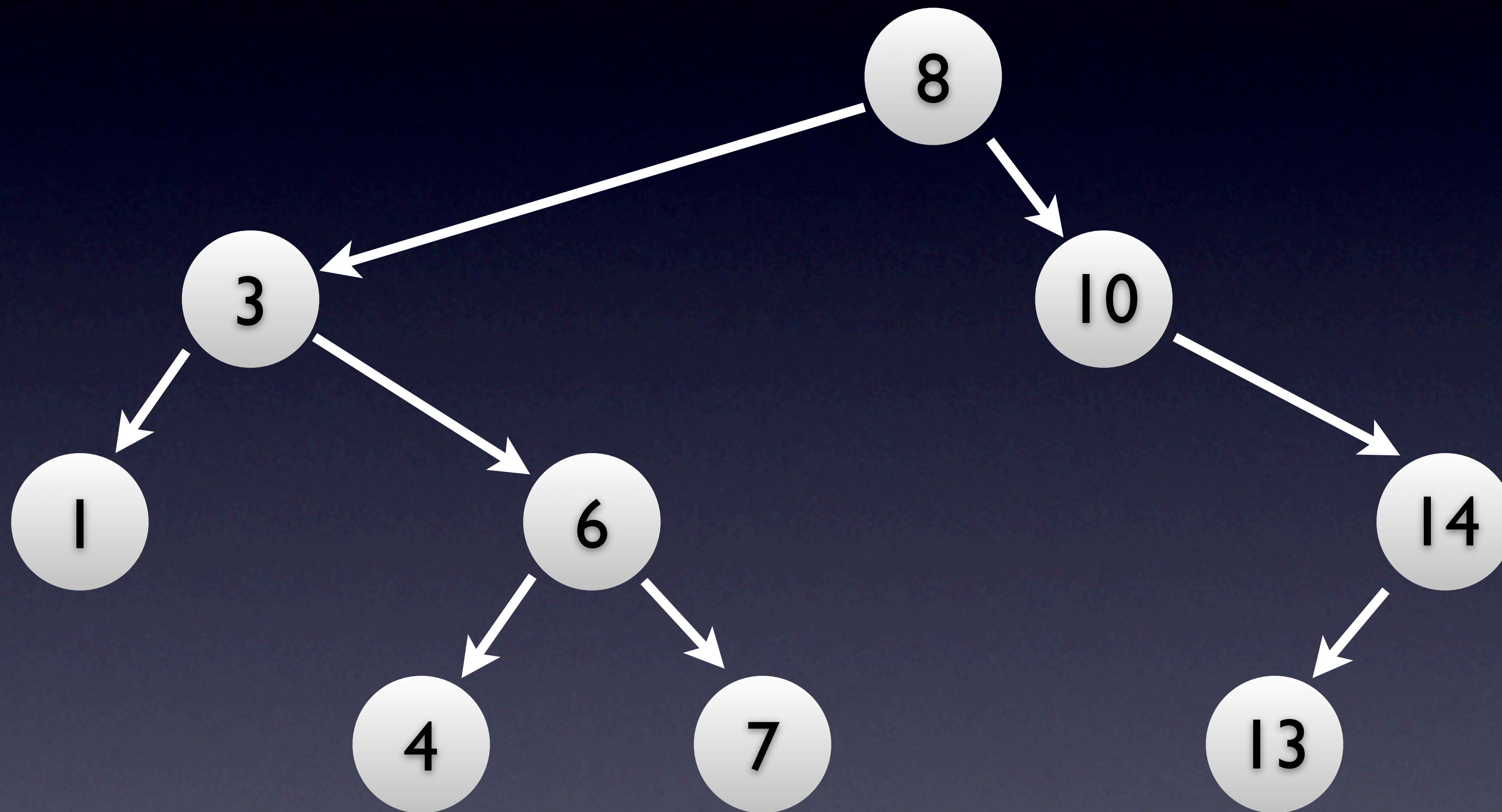
Insert 9



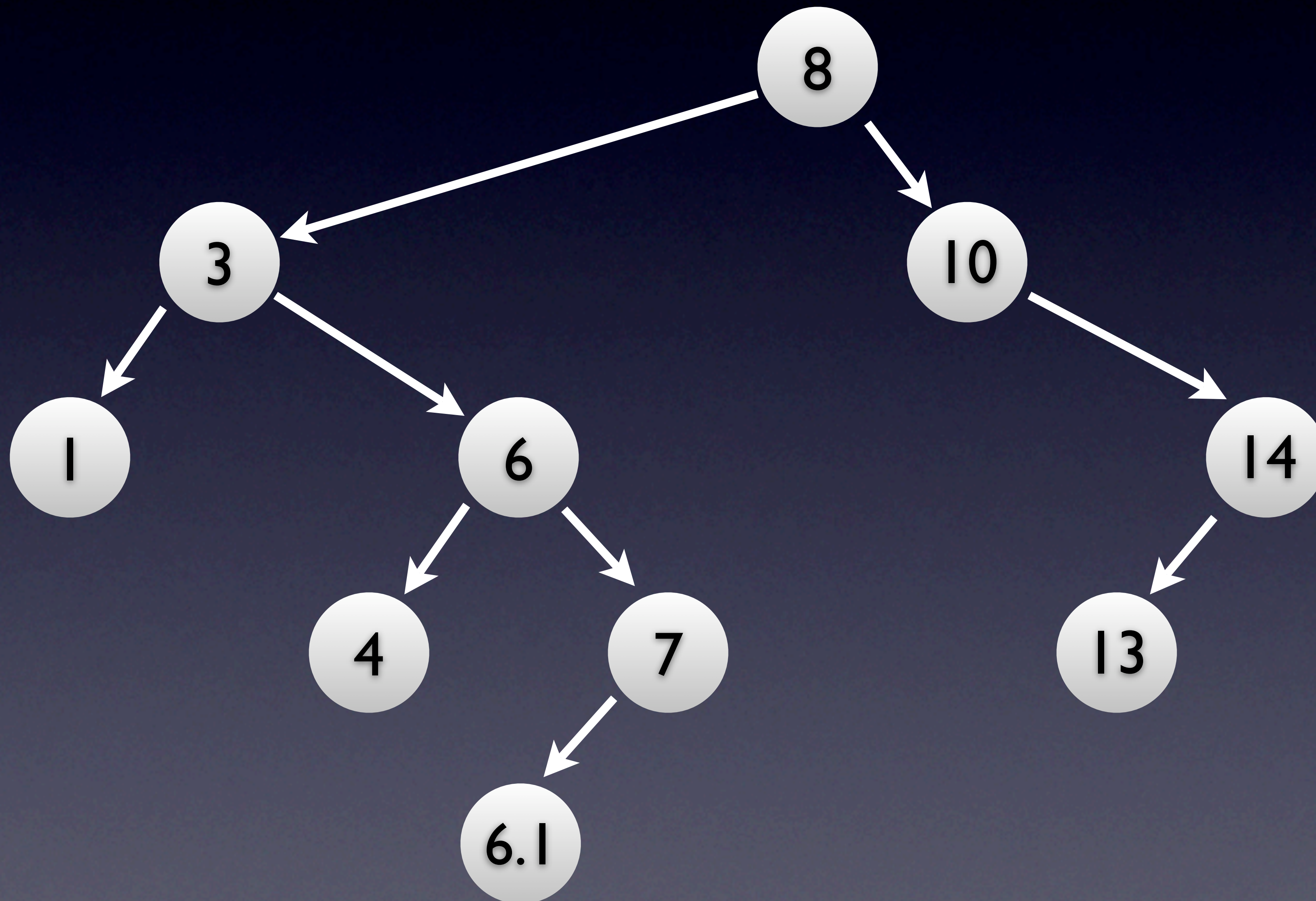
Insert 9



Insert 6.1



Insert 6.1



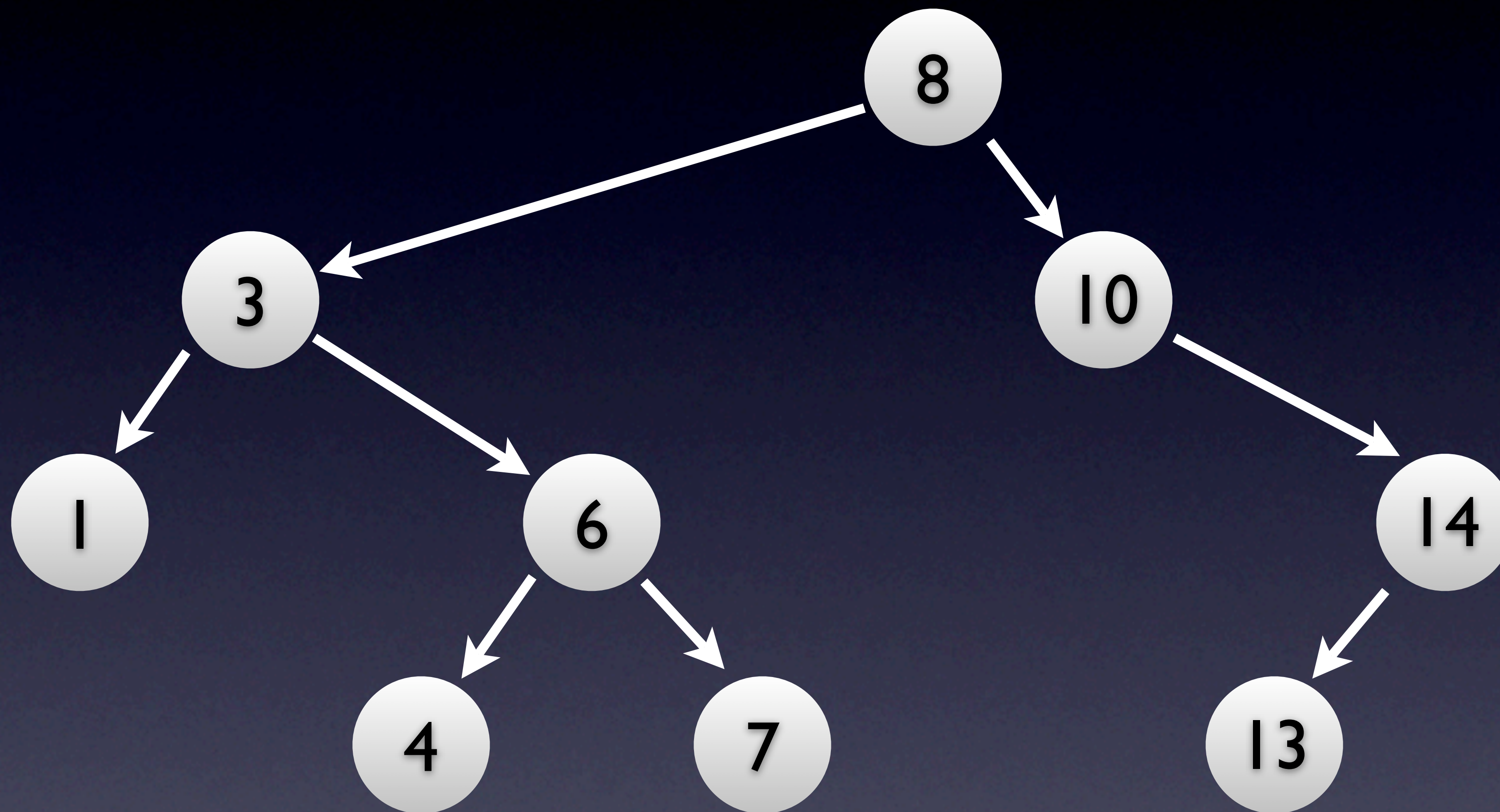
BST Insertion

```
1  class BSTnode(object):
2     def __init__(self, parent, t):
3         self.key = t
4         self.parent = parent
4         self.left = None
5         self.right = None
6
7     def insert(self, t):
8         if t < self.key:
9             if self.left is None:
10                self.left = BSTnode(self, t)
11            else:
12                self.left.insert(t)
13        else:
14            if self.right is None:
15                self.right = BSTnode(self, t)
16            else:
17                self.right.insert(t)
```

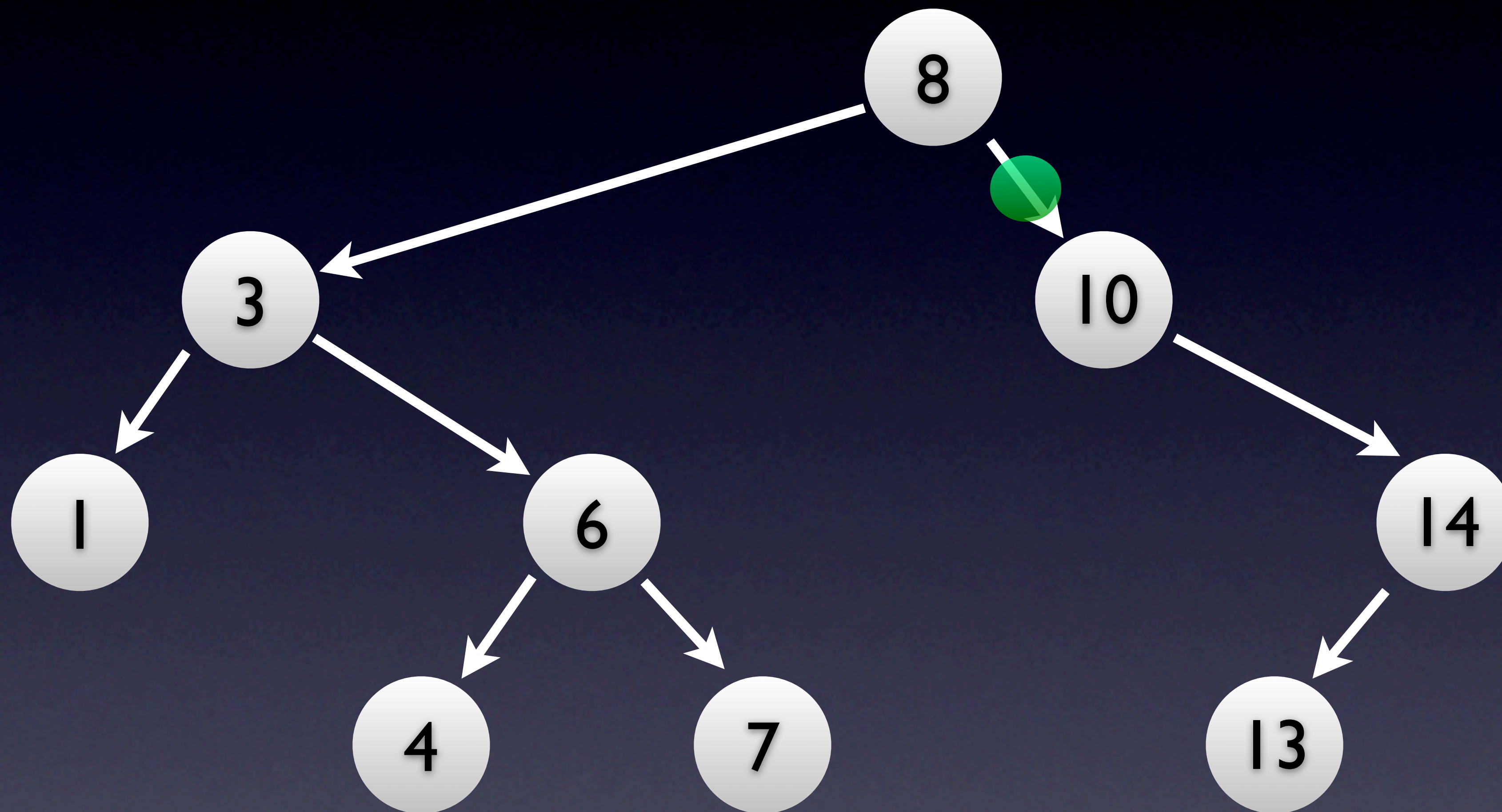
The BST Wrapper

```
1 class BST(object):
2     def __init__(self):
3         self.root = None
4 
5     def insert(self, t):
6         if self.root is None:
7             self.root = BSTnode(None, t)
8         else:
9             self.root.insert(t)
10
11    def find(self, t):
12        if self.root is None:
13            return None
14        else:
15            return self.root.find(t)
```

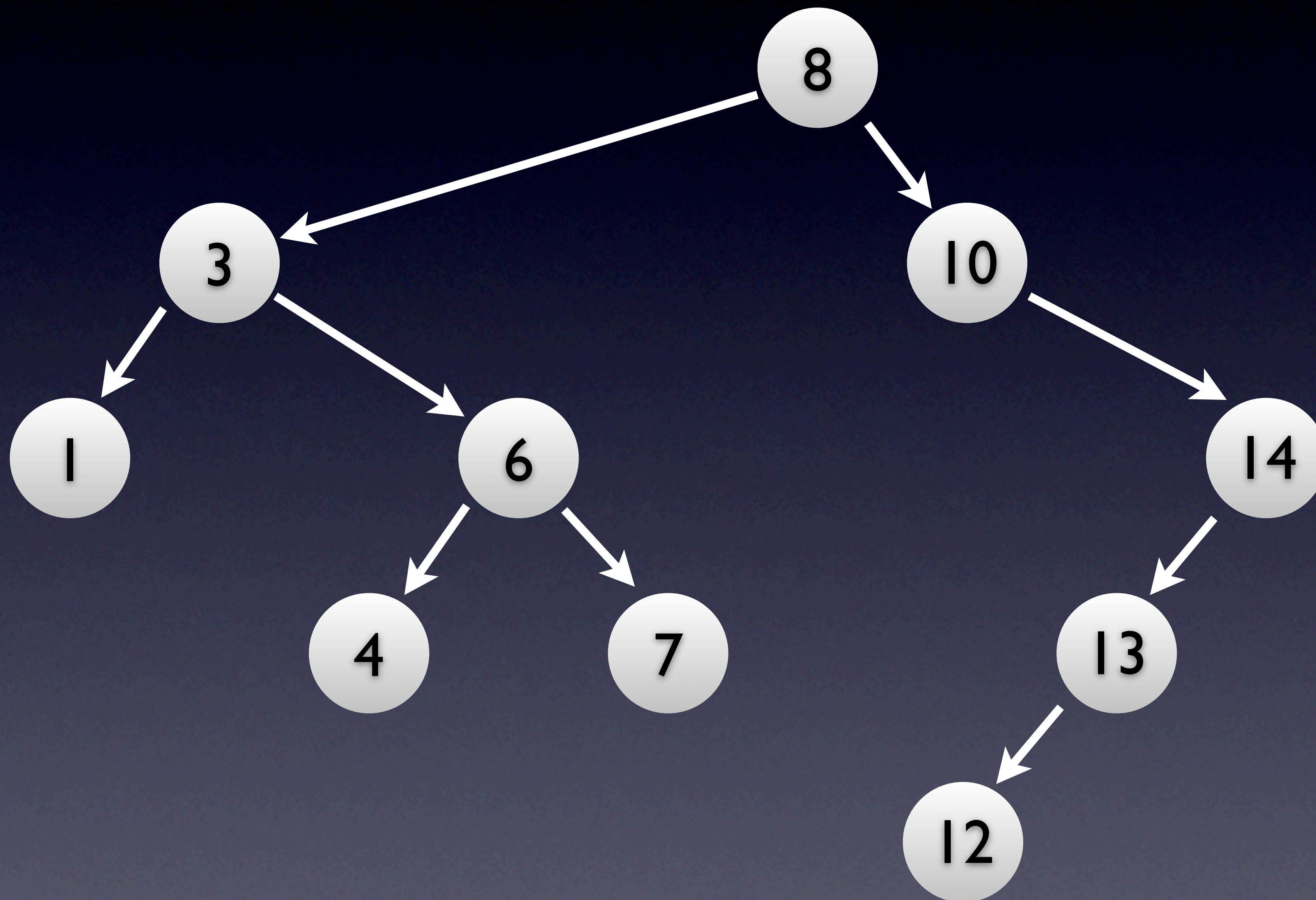
Successor of 8



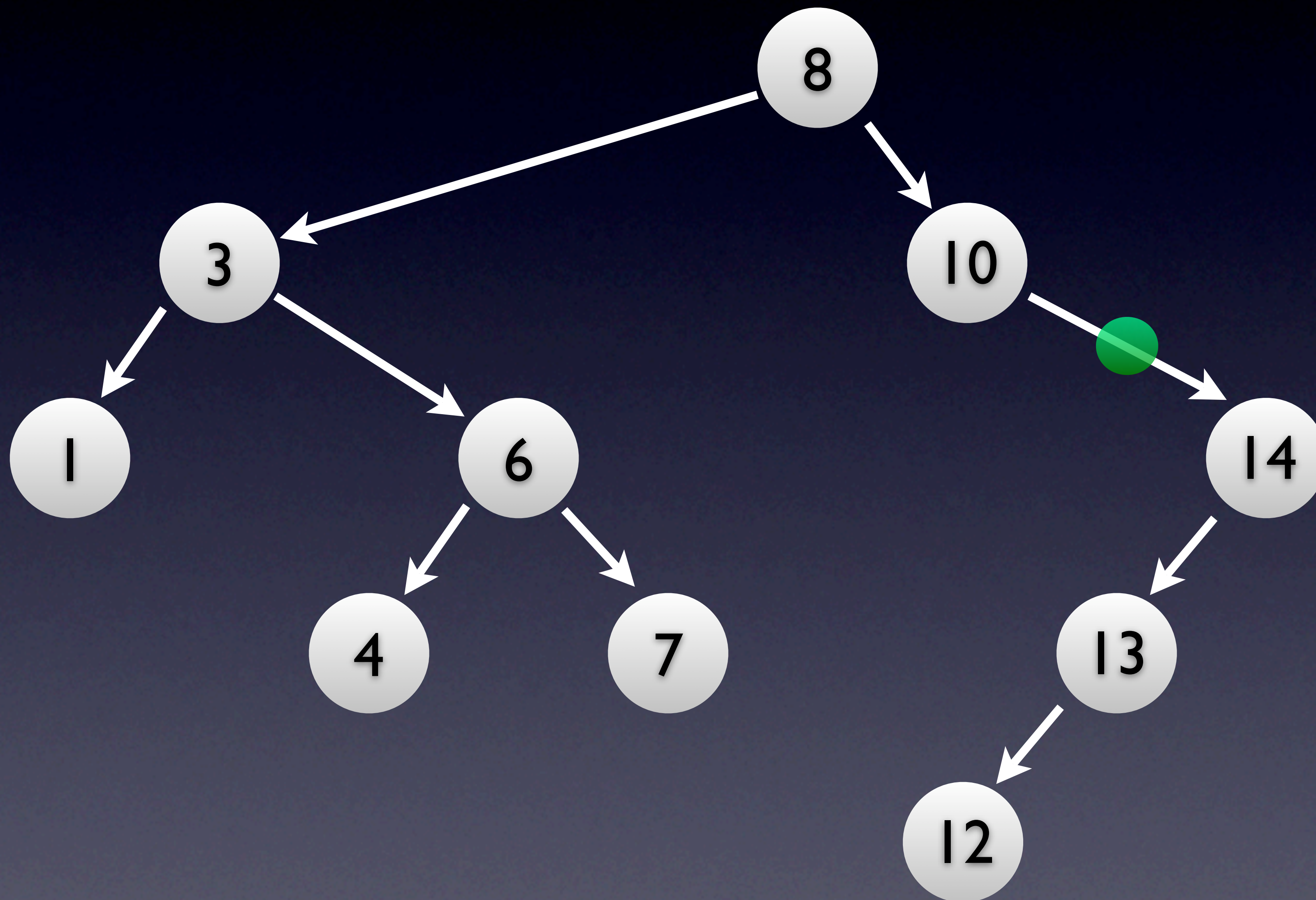
Successor of 8



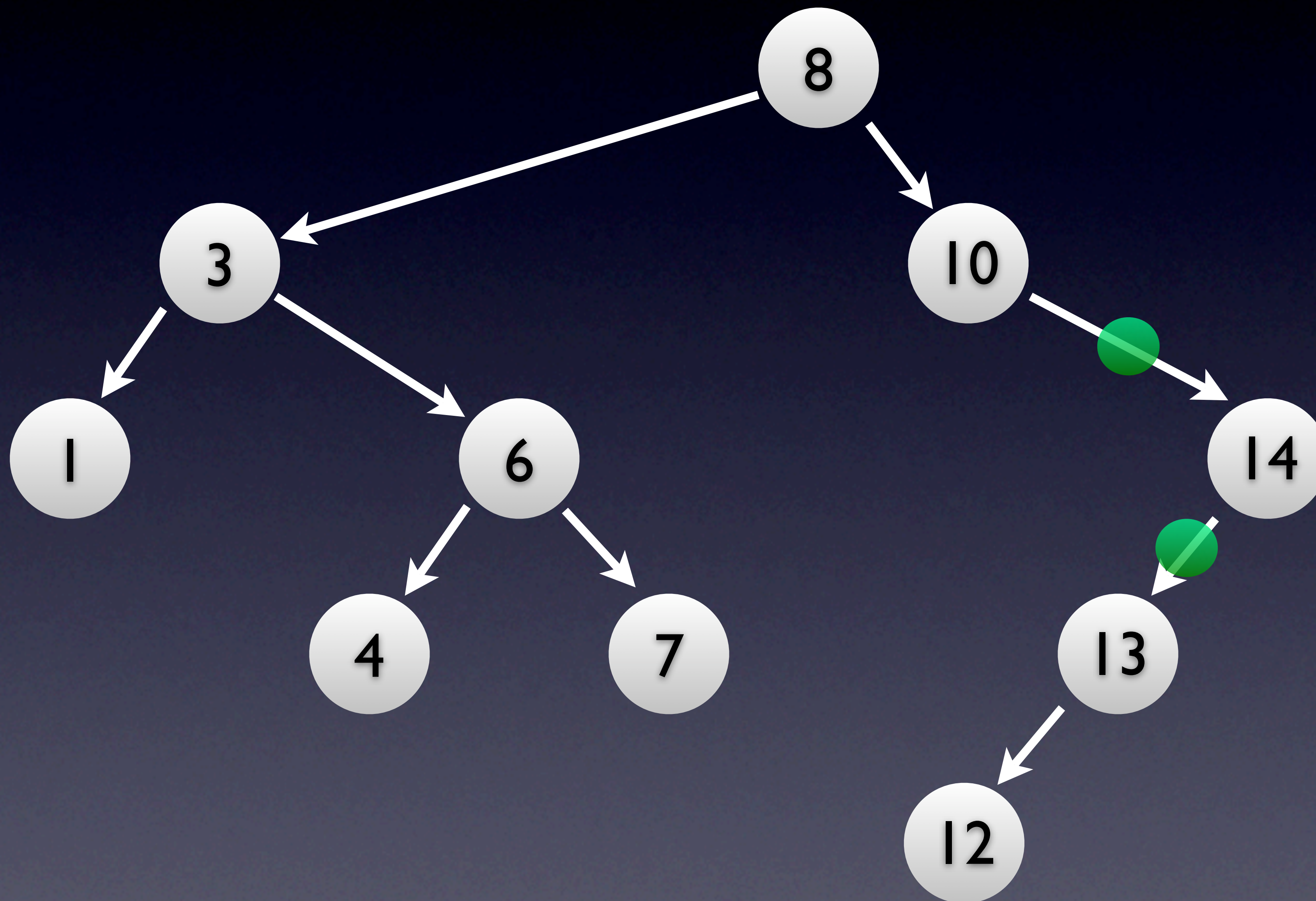
Successor of 10



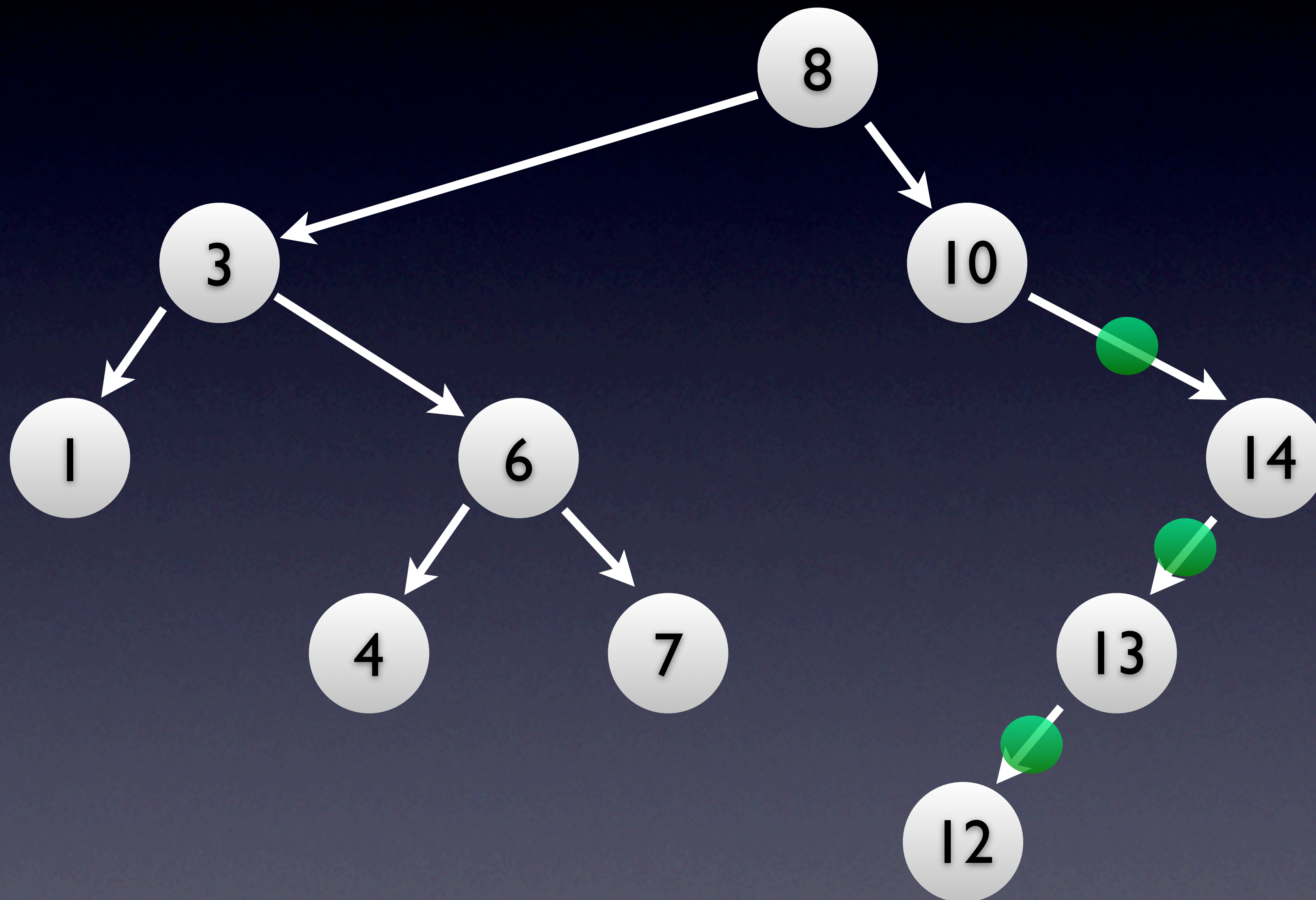
Successor of 10



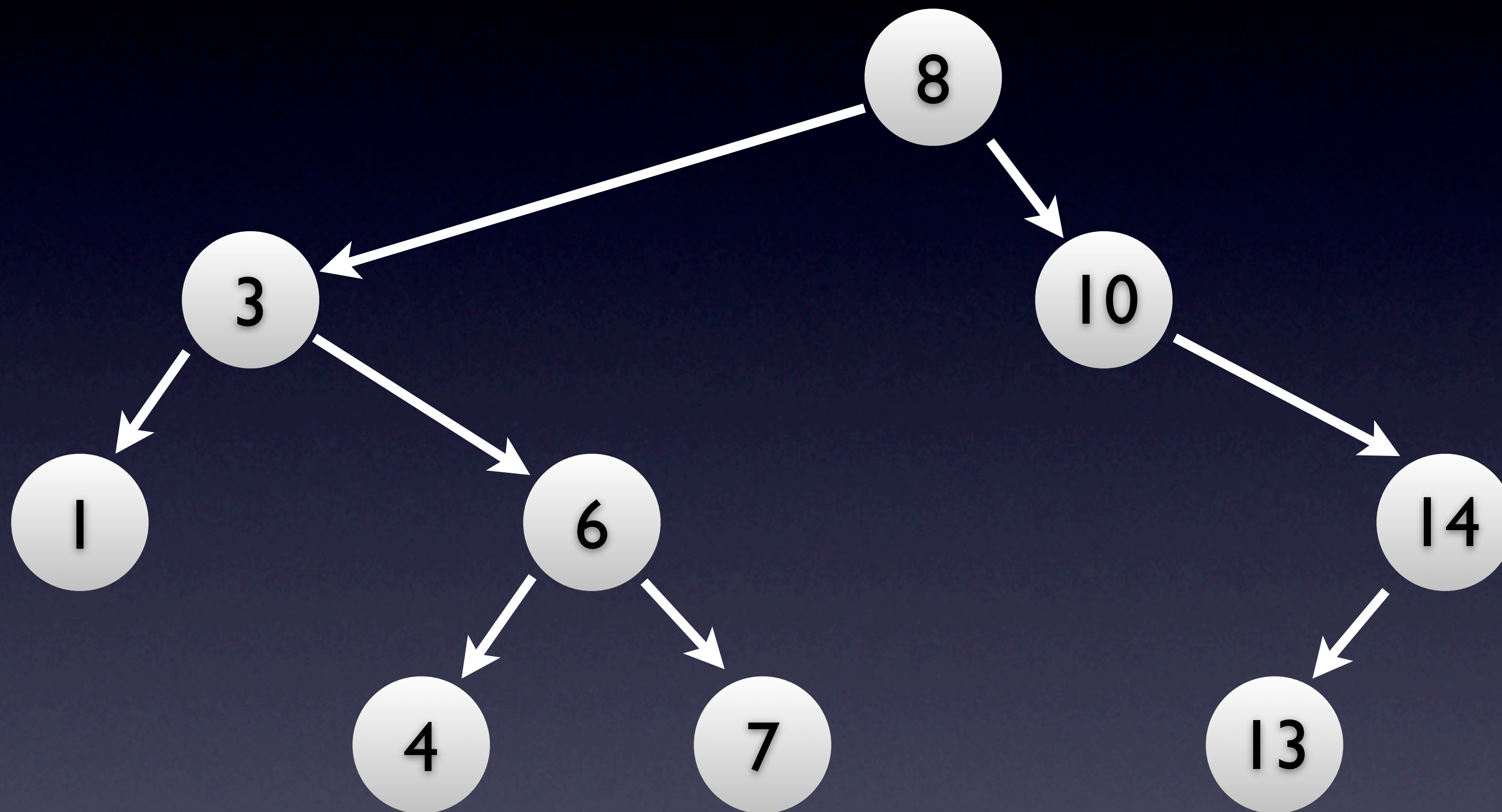
Successor of 10



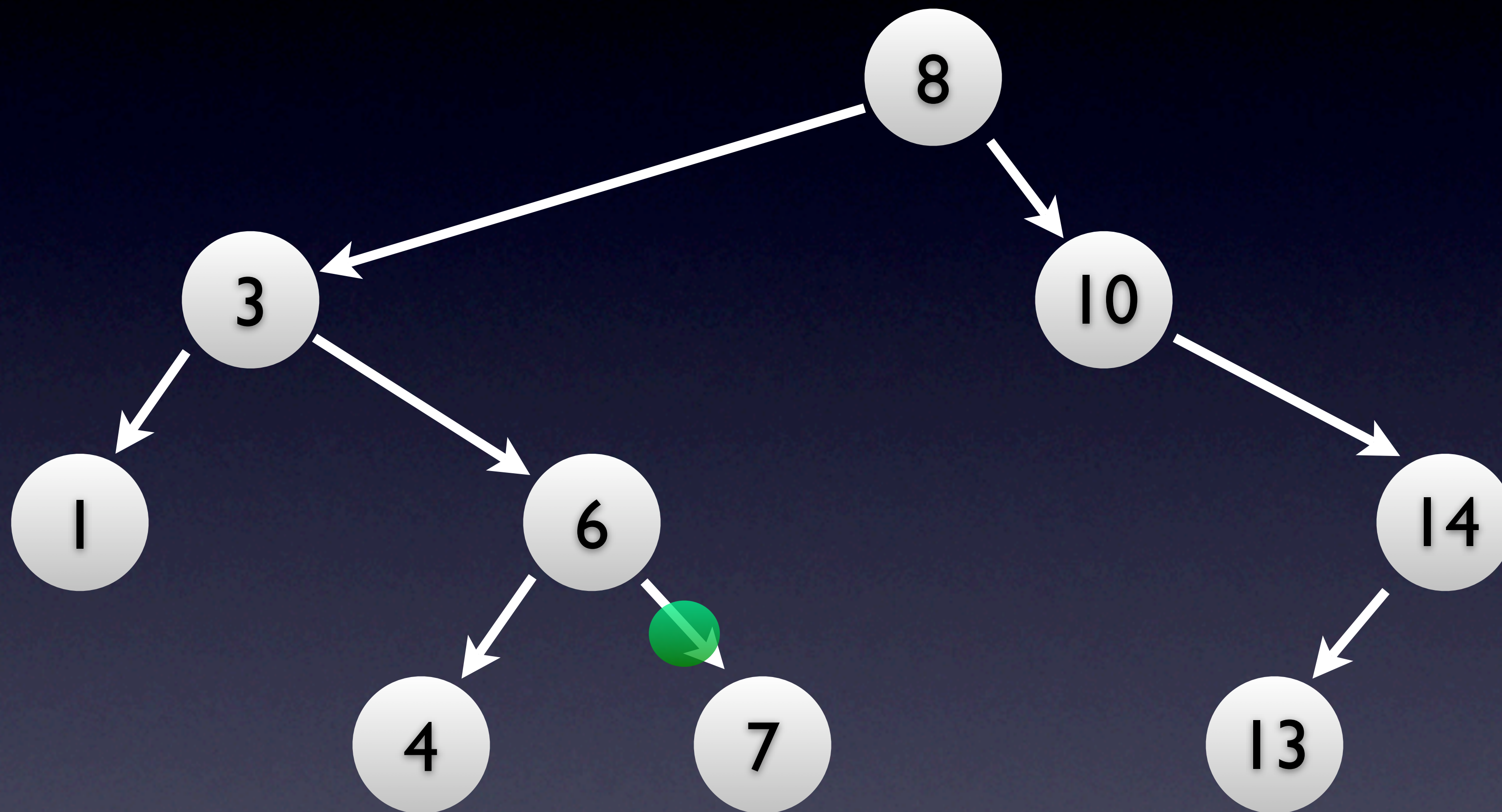
Successor of 10



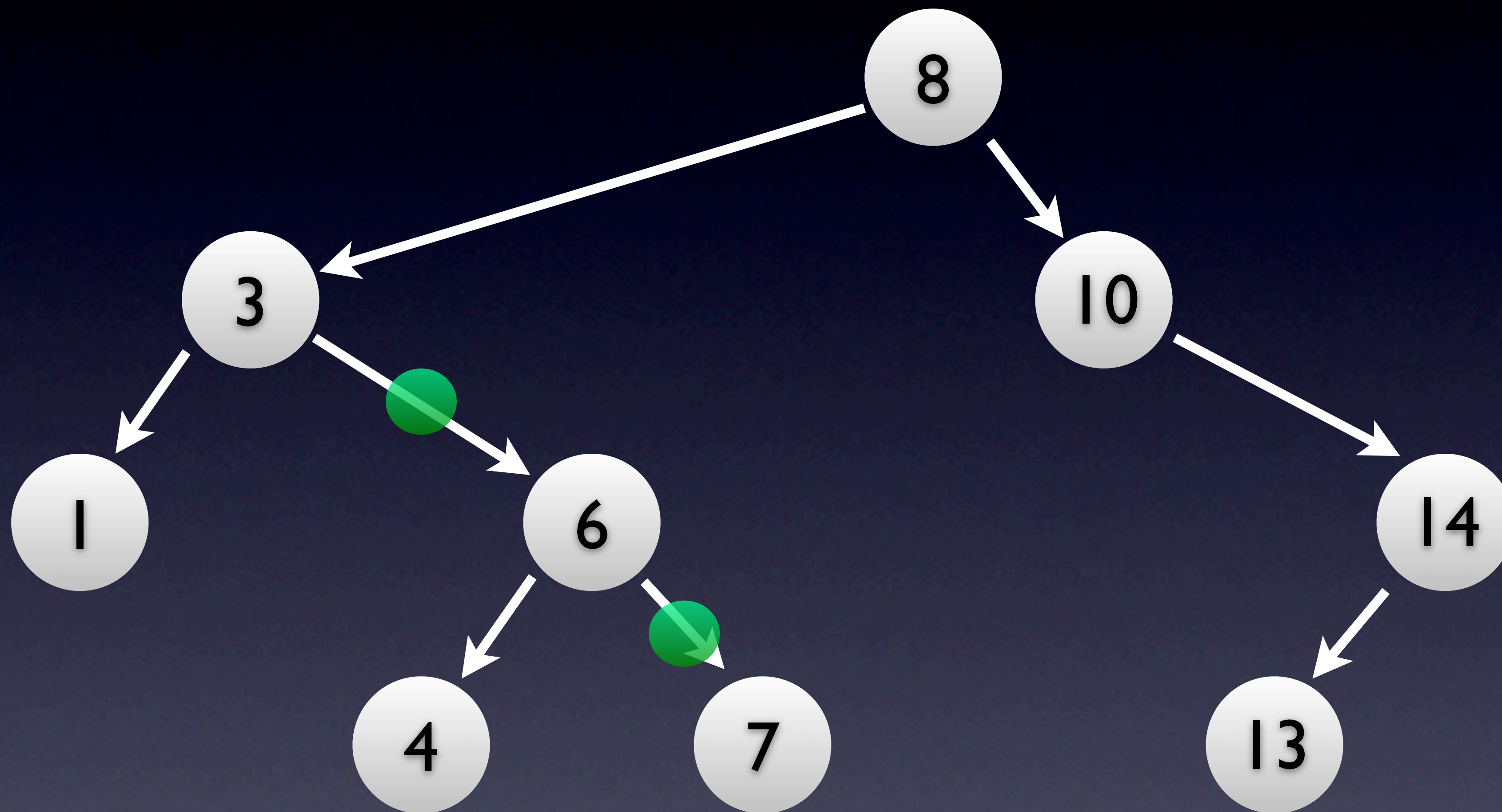
Successor of 7



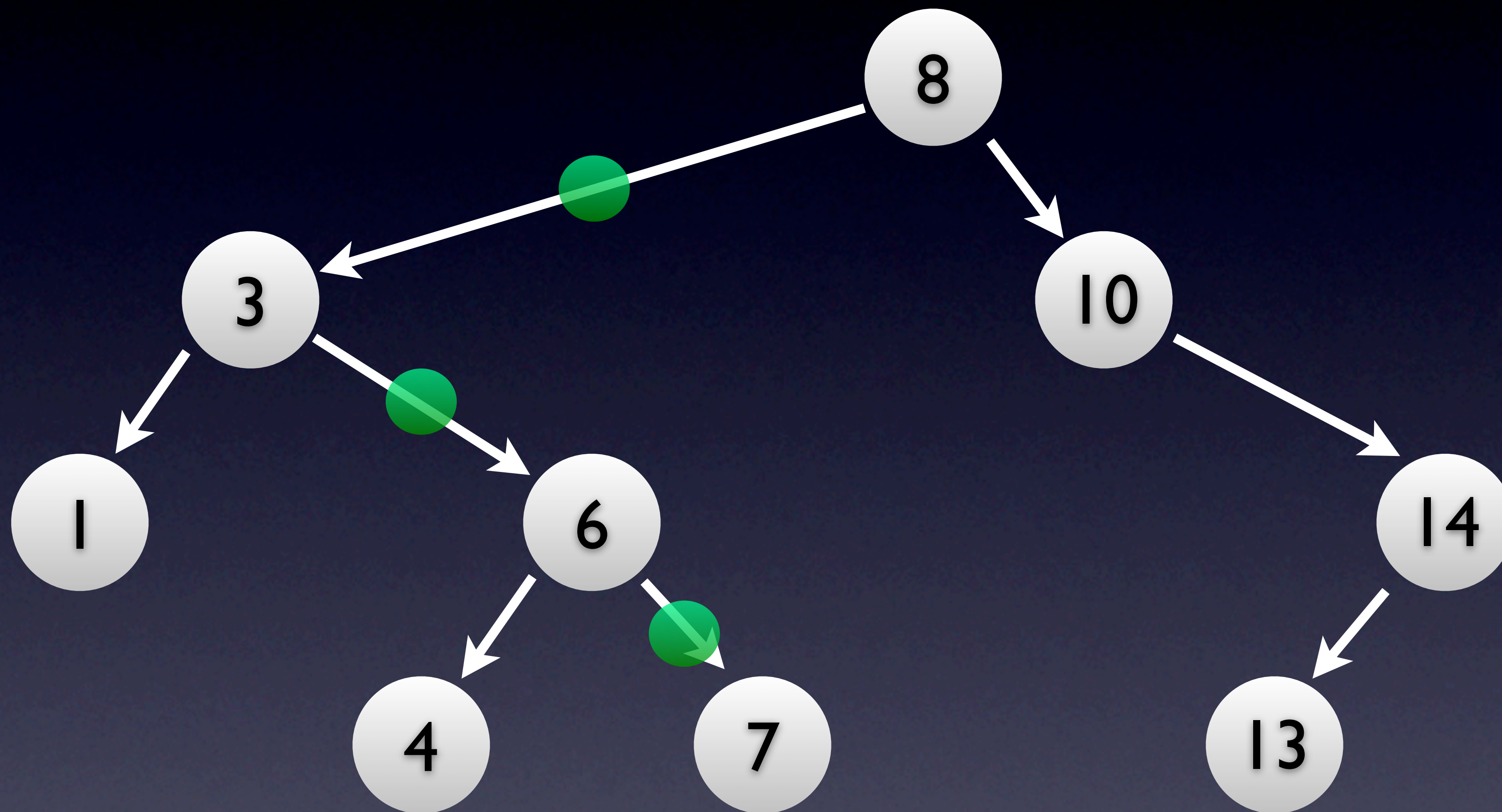
Successor of 7



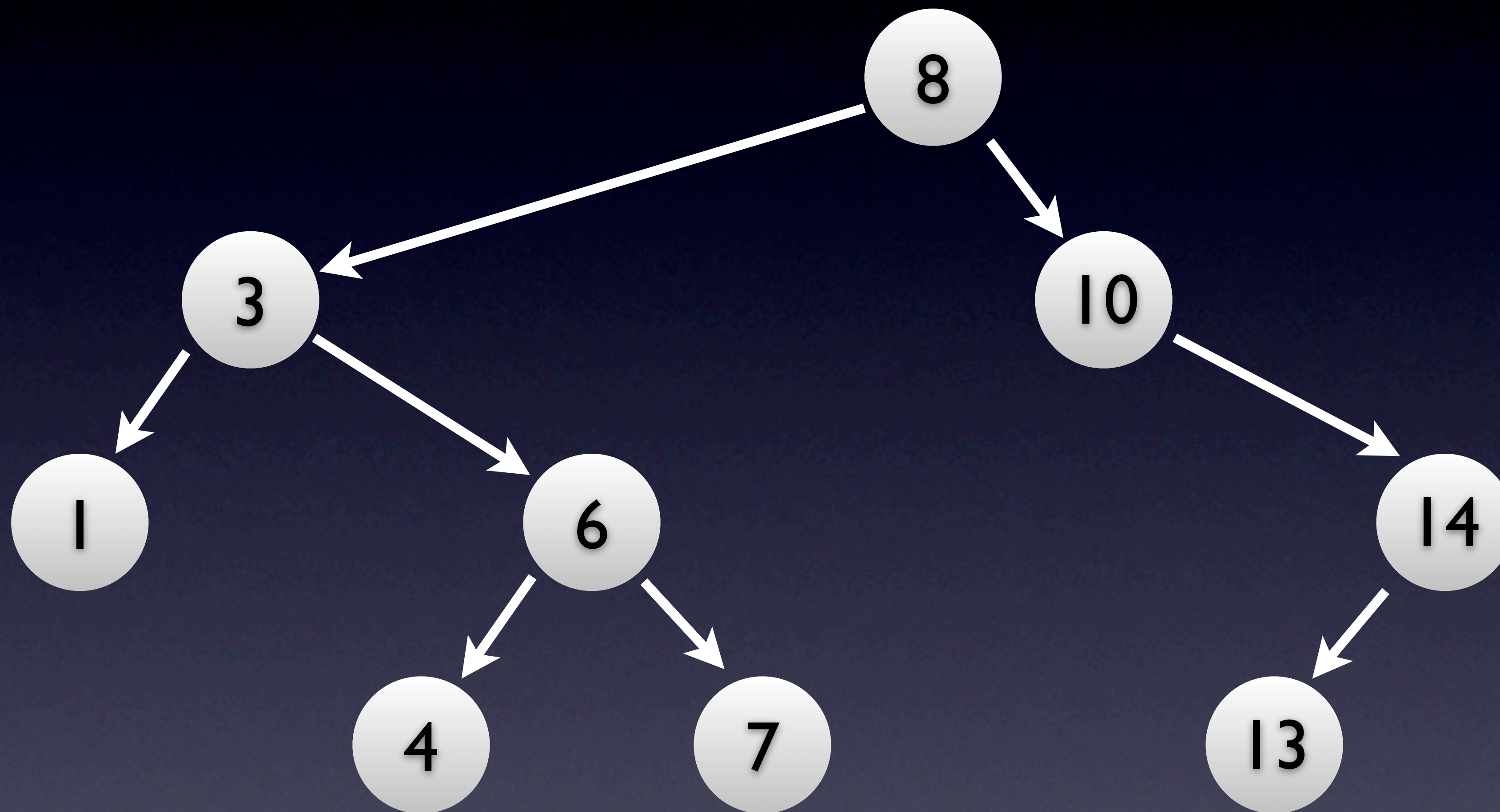
Successor of 7



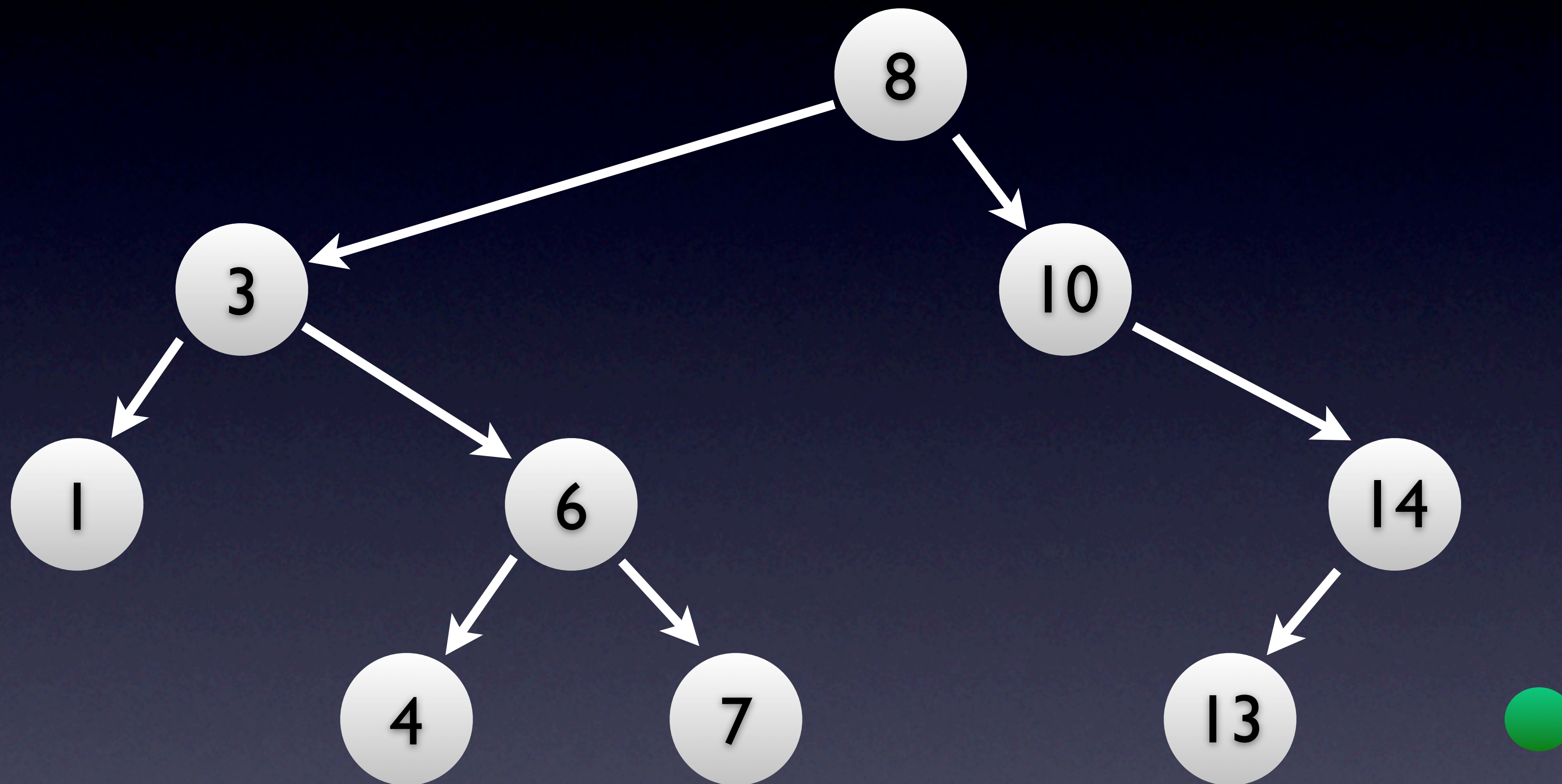
Successor of 7



Successor of 14



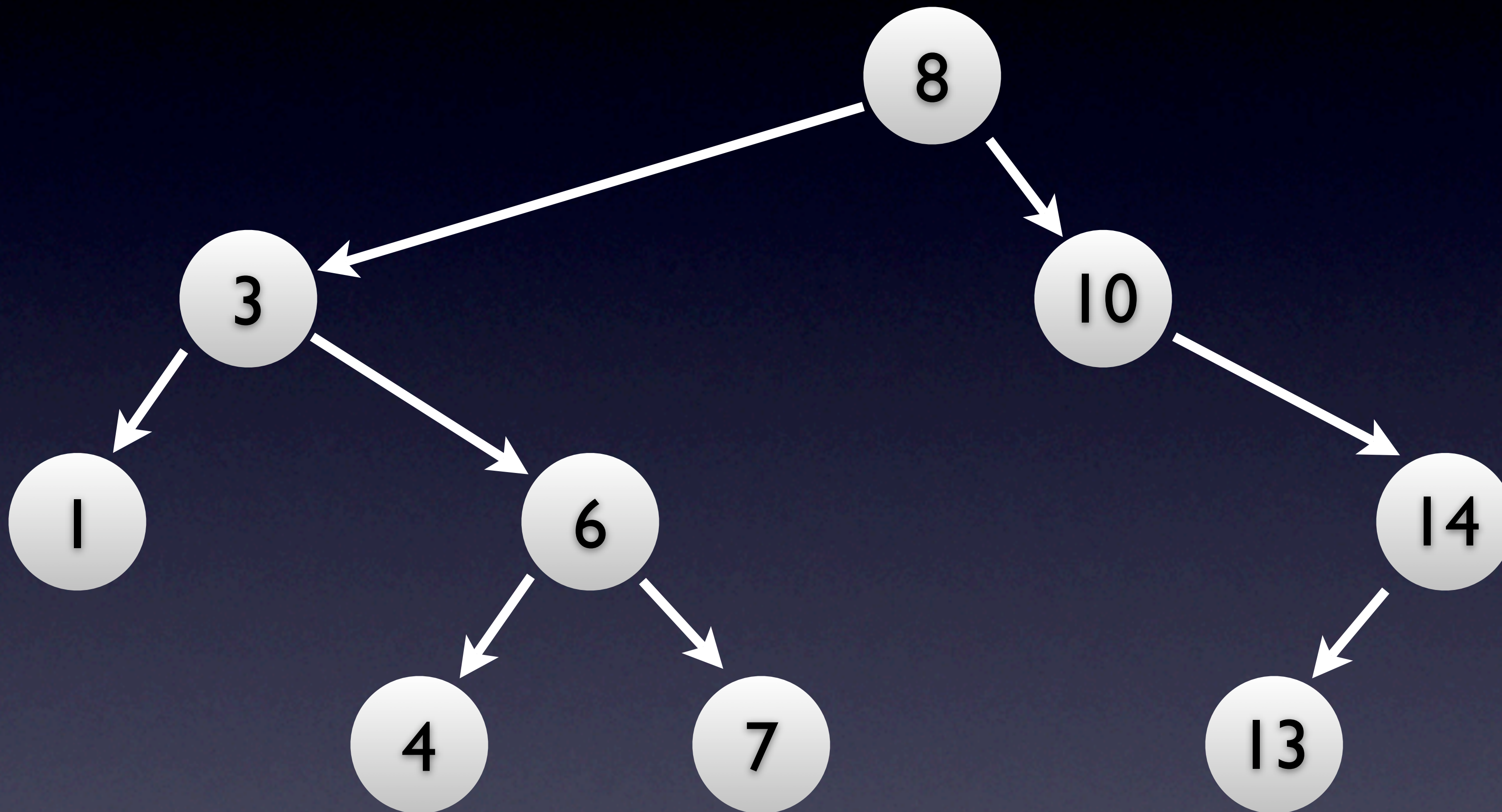
Successor of 14



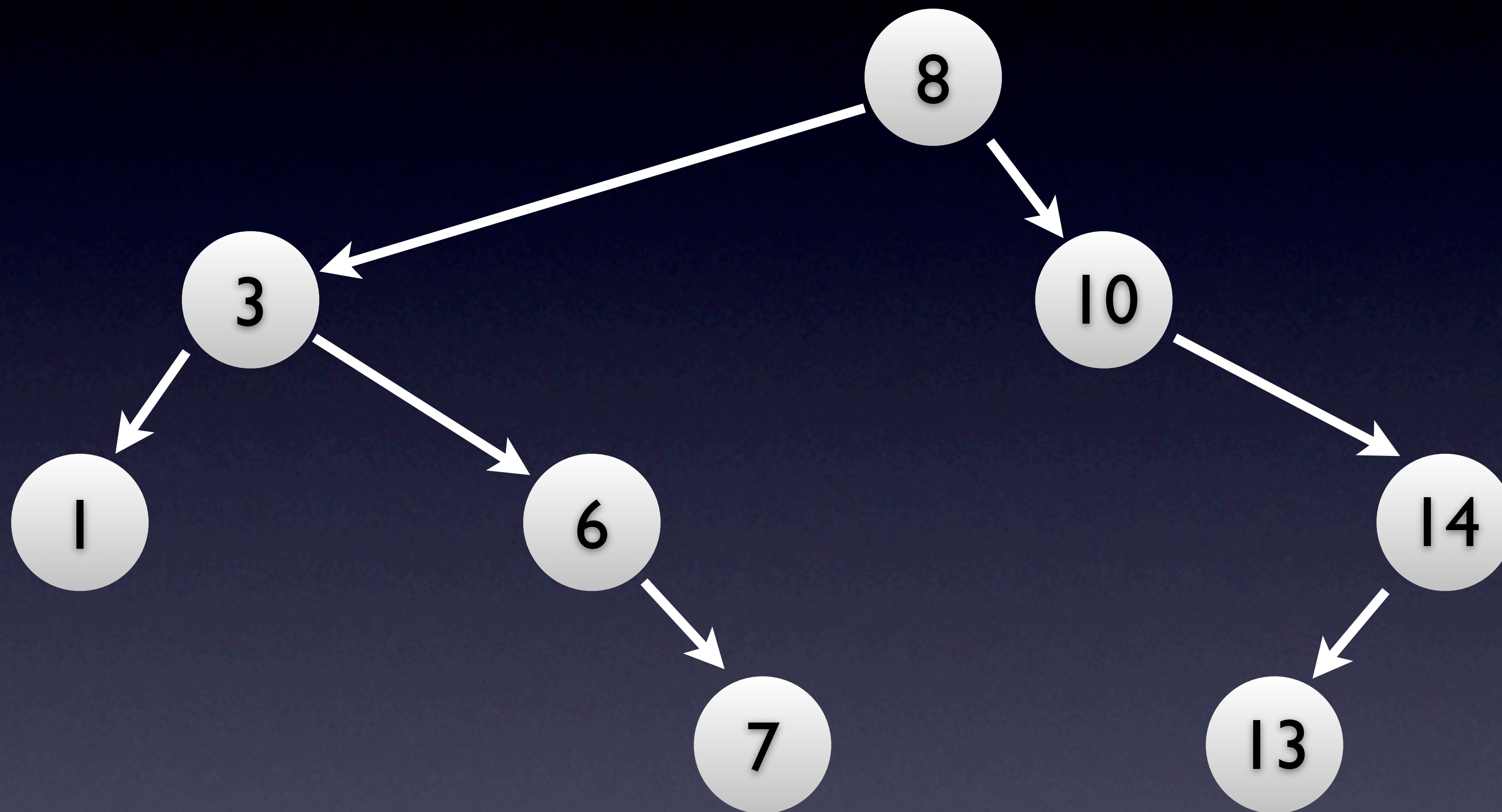
BST Successor

```
1 class BSTnode(object):
2    def __init__(self, parent, t):
3        self.key = t
3        self.parent = parent
4        self.left = None
5        self.right = None
6
7    def minimum(self):
8        current = self
9        while current.left is not None:
10           current = current.left
11       return current
12
13   def successor(self):
14       if self.right is not None:
15           return self.right.minimum()
16       current = self
17       while current.parent is not None and current.parent.right is current:
18           current = current.parent
19       return current.parent
```

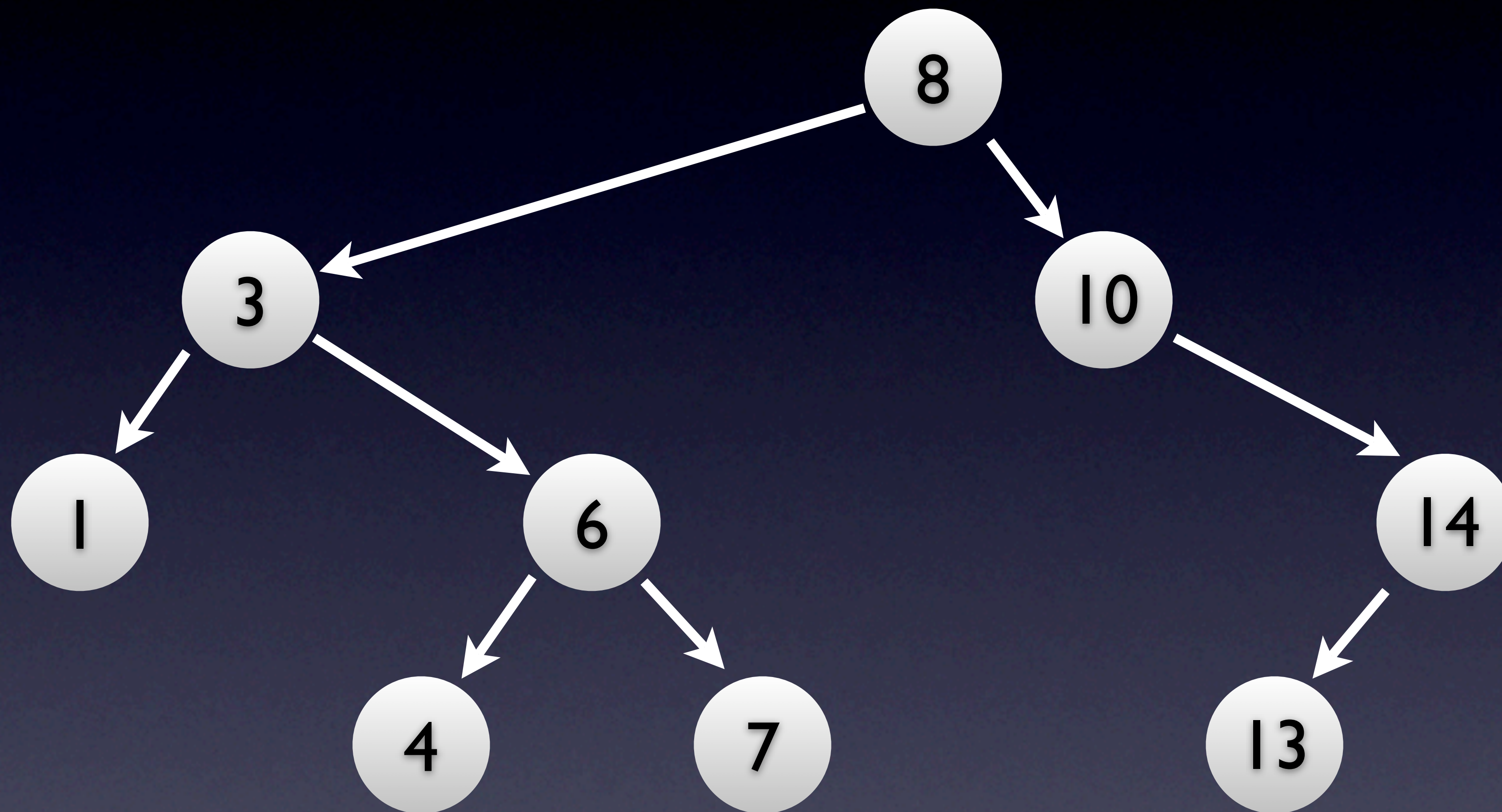
Delete 4



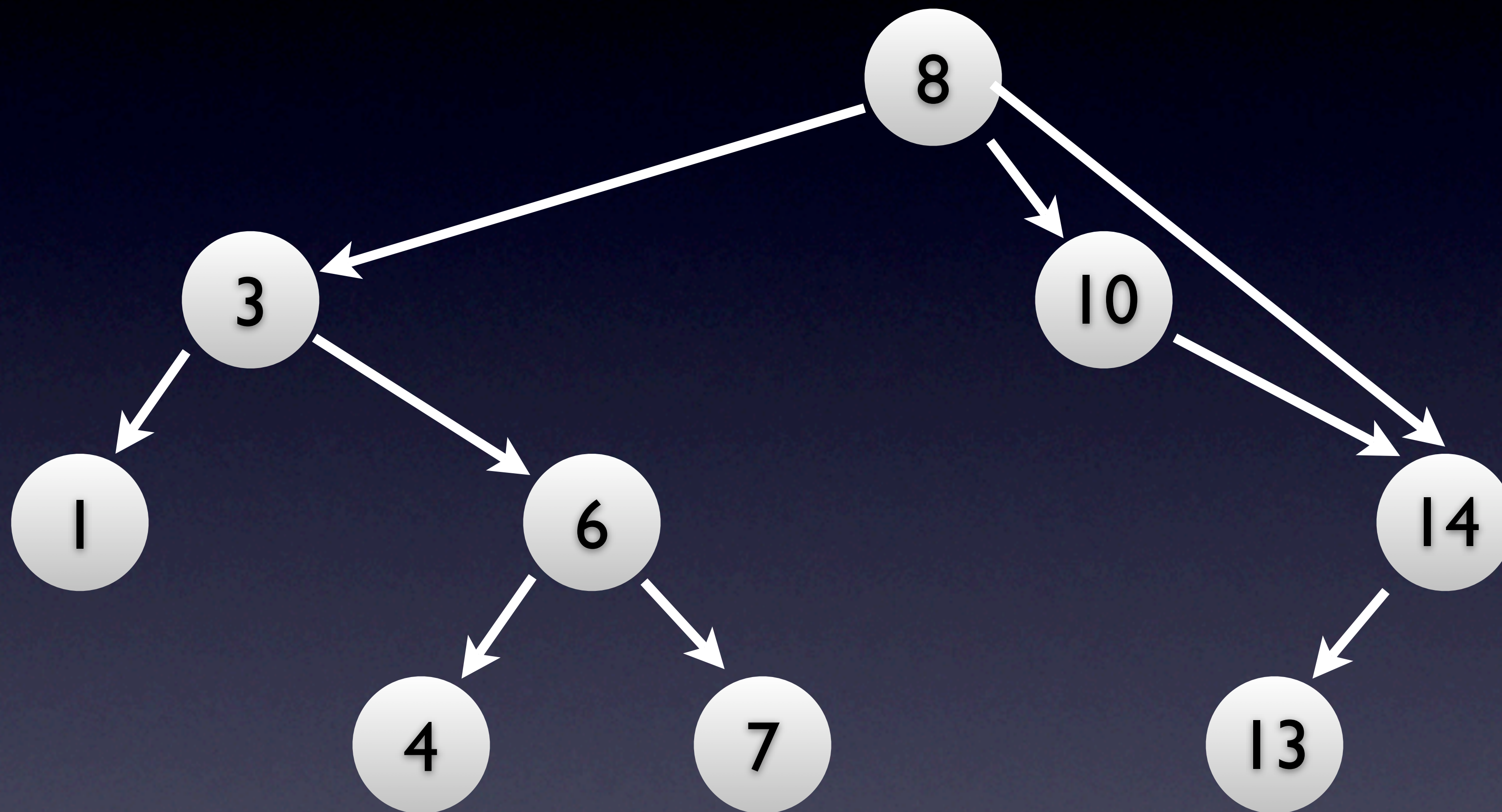
Delete 4



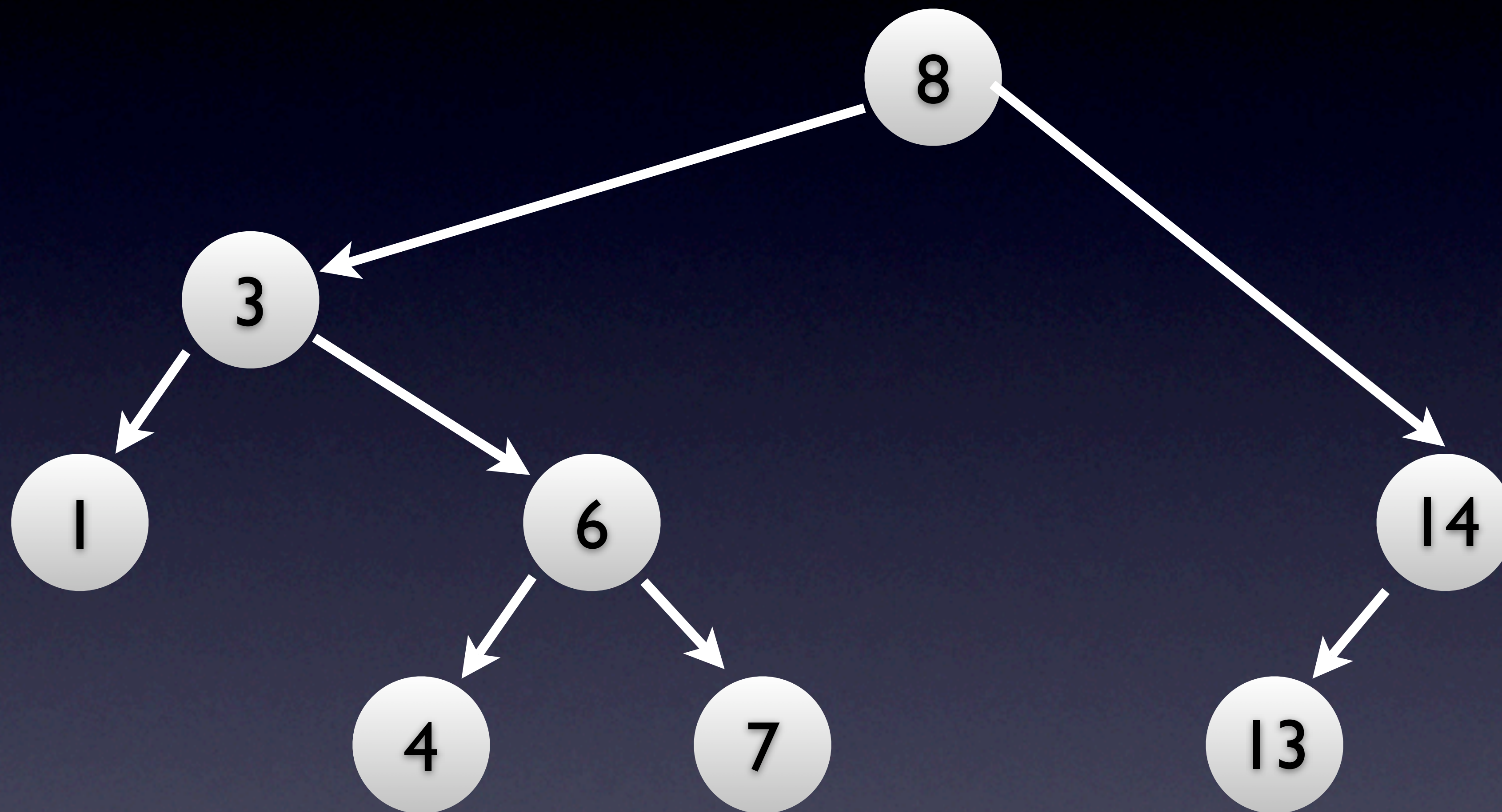
Delete 10



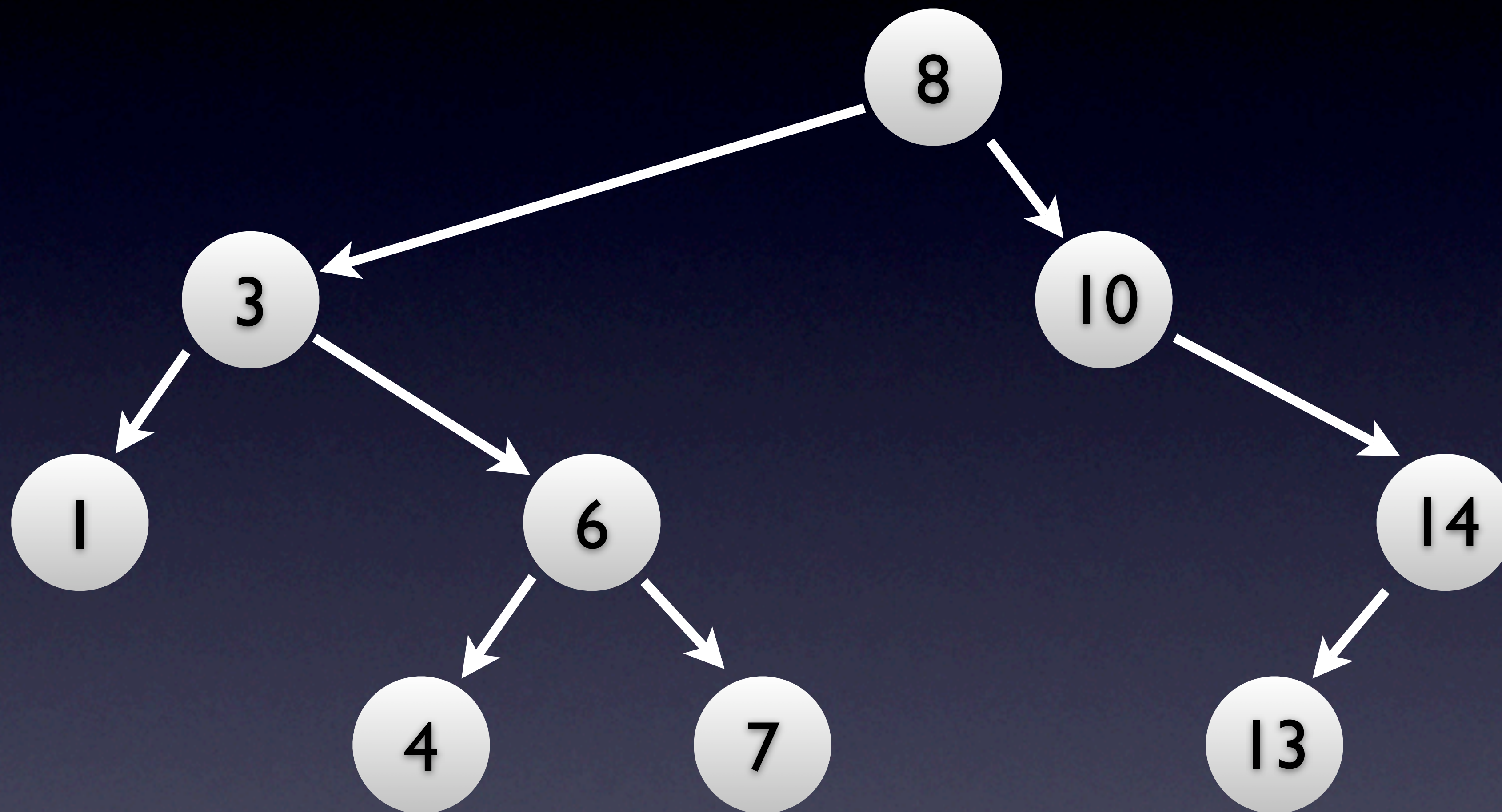
Delete 10



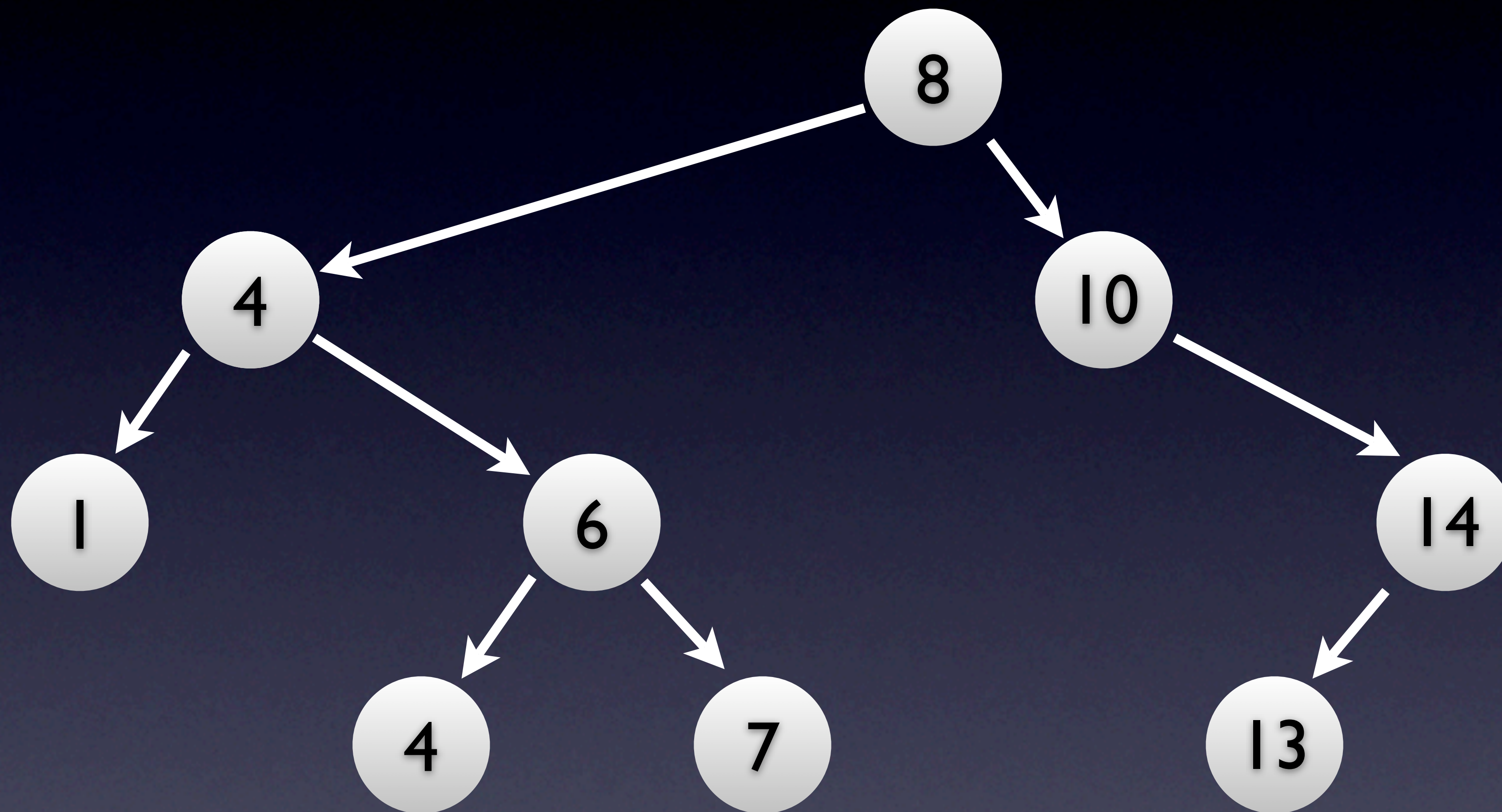
Delete 10



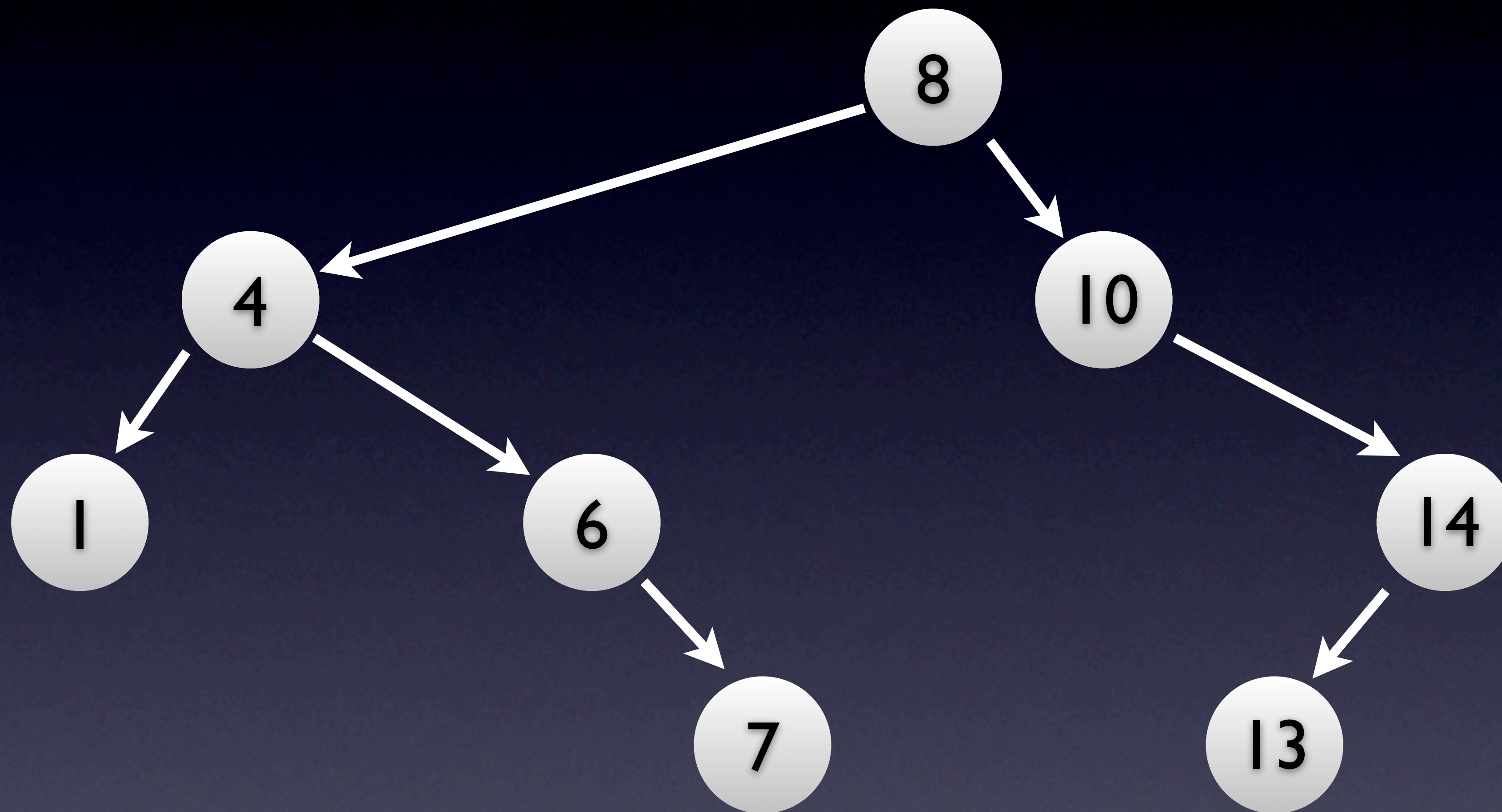
Delete 3



Delete 3



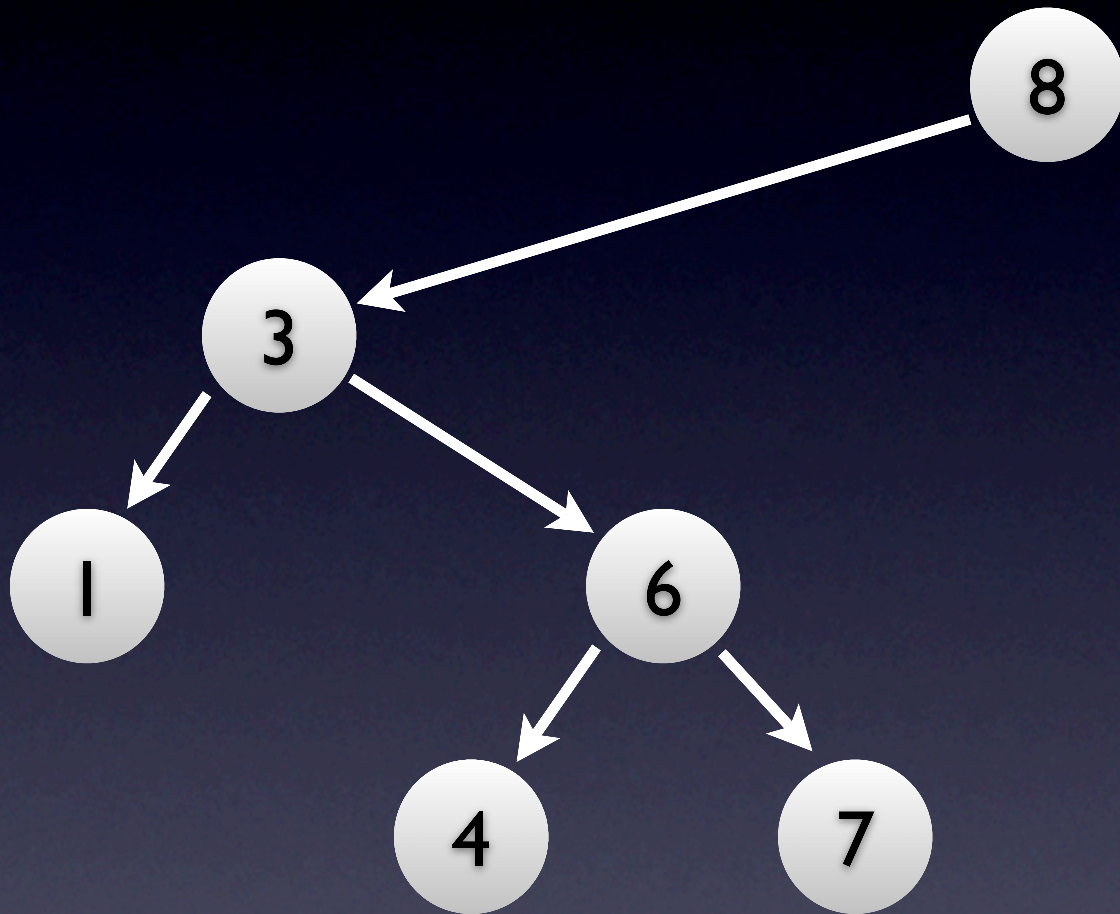
Delete 3



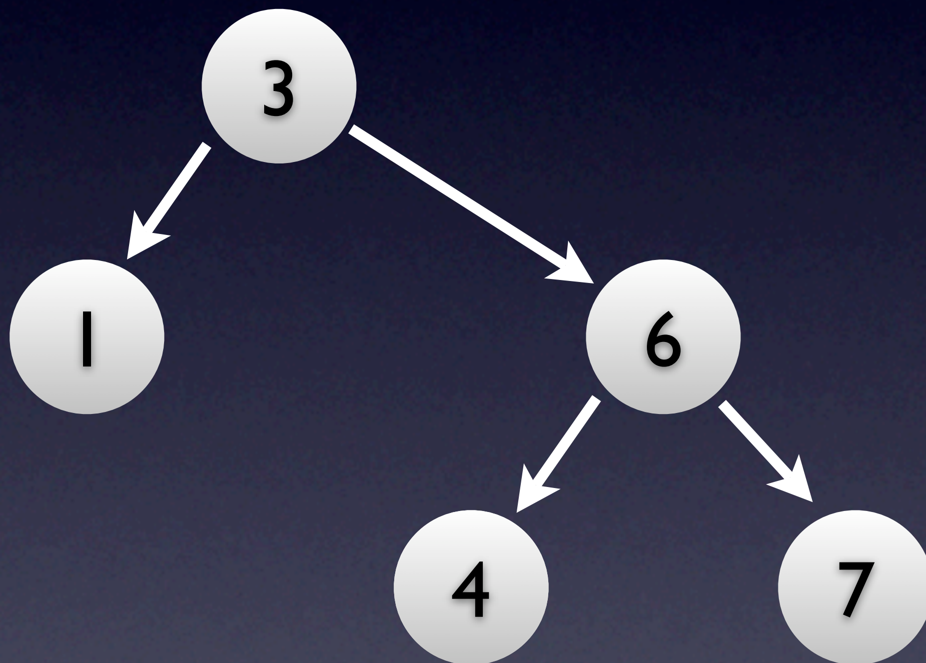
BST Deletion

```
1 class BSTnode(object):
2    def delete(self):
3        if self.left is None or self.right is None:
4            if self is self.parent.left:
5                self.parent.left = self.left or self.right
6                if self.parent.left is not None:
7                    self.parent.left.parent = self.parent
8            else:
9                self.parent.right = self.left or self.right
10               if self.parent.right is not None:
11                   self.parent.right.parent = self.parent
12           return self
13       else:
14           s = self.successor()
15           self.key, s.key = s.key, self.key
16           return s.delete()
```

Delete 8

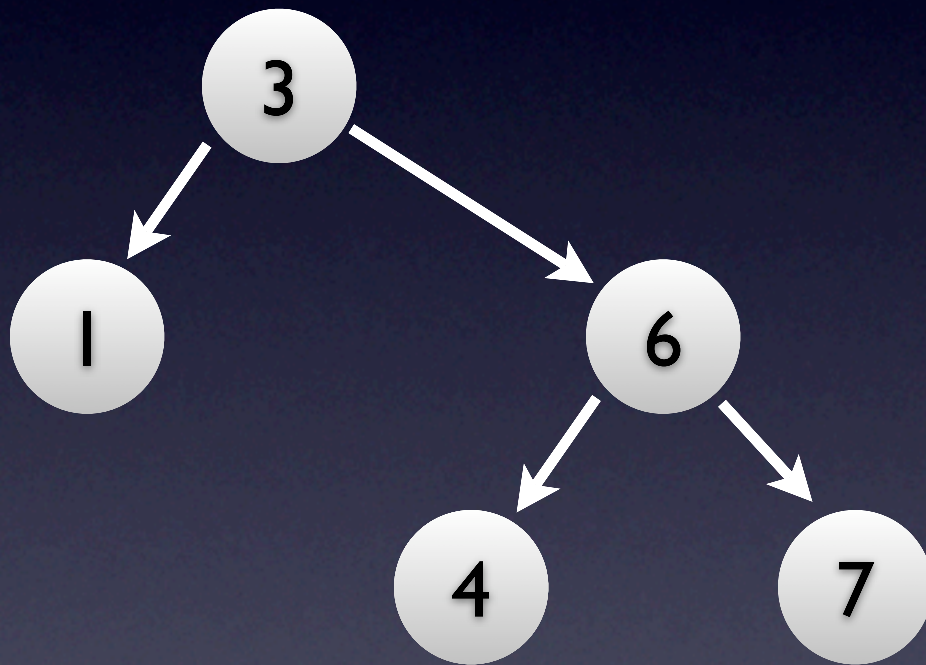


Delete 8

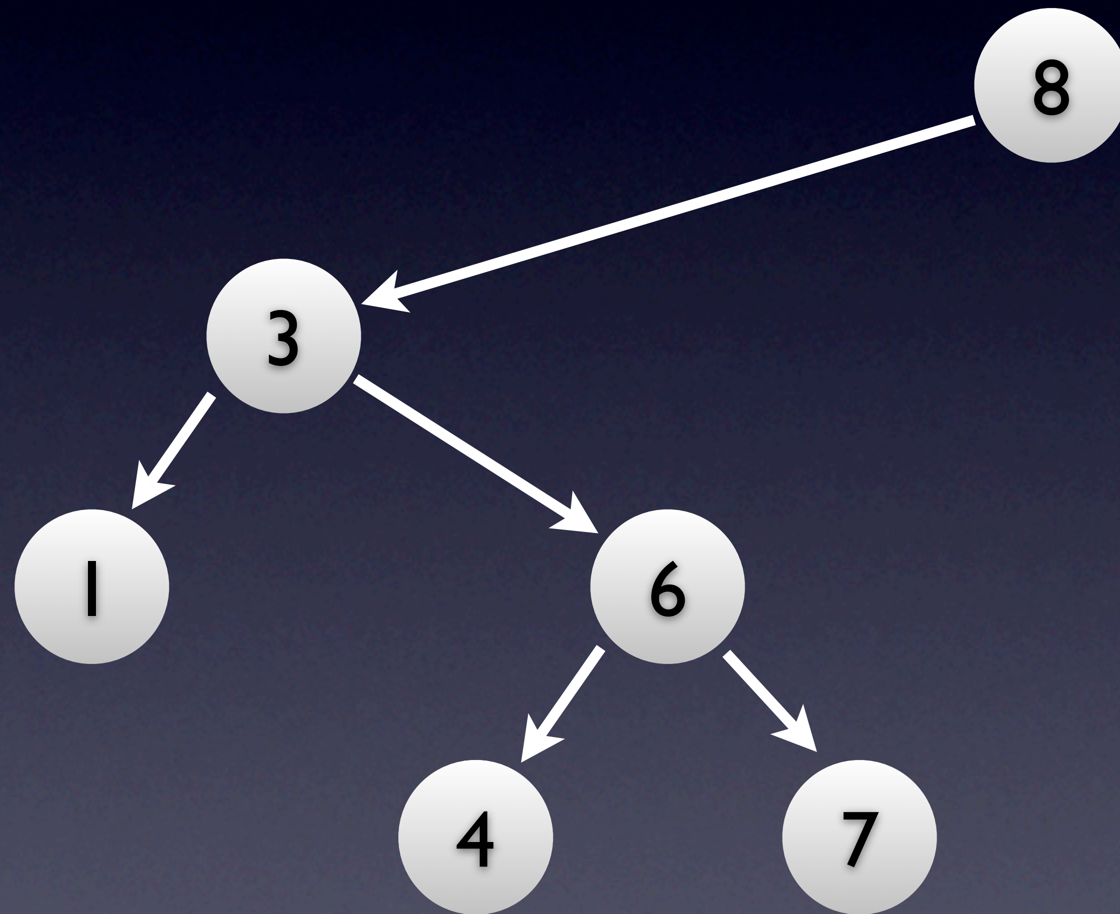


Delete 8

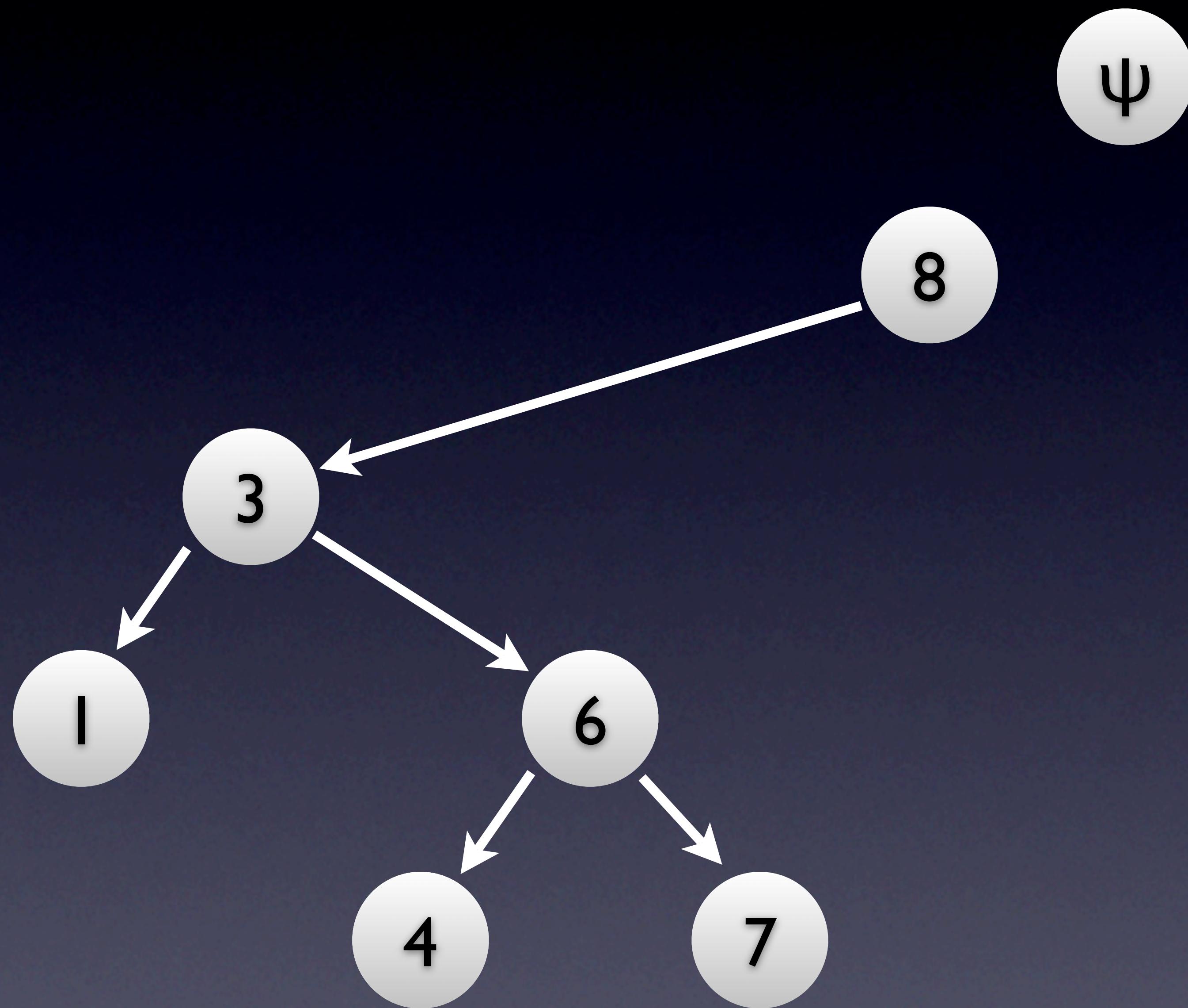
BST: Dude, where's self.root ?!



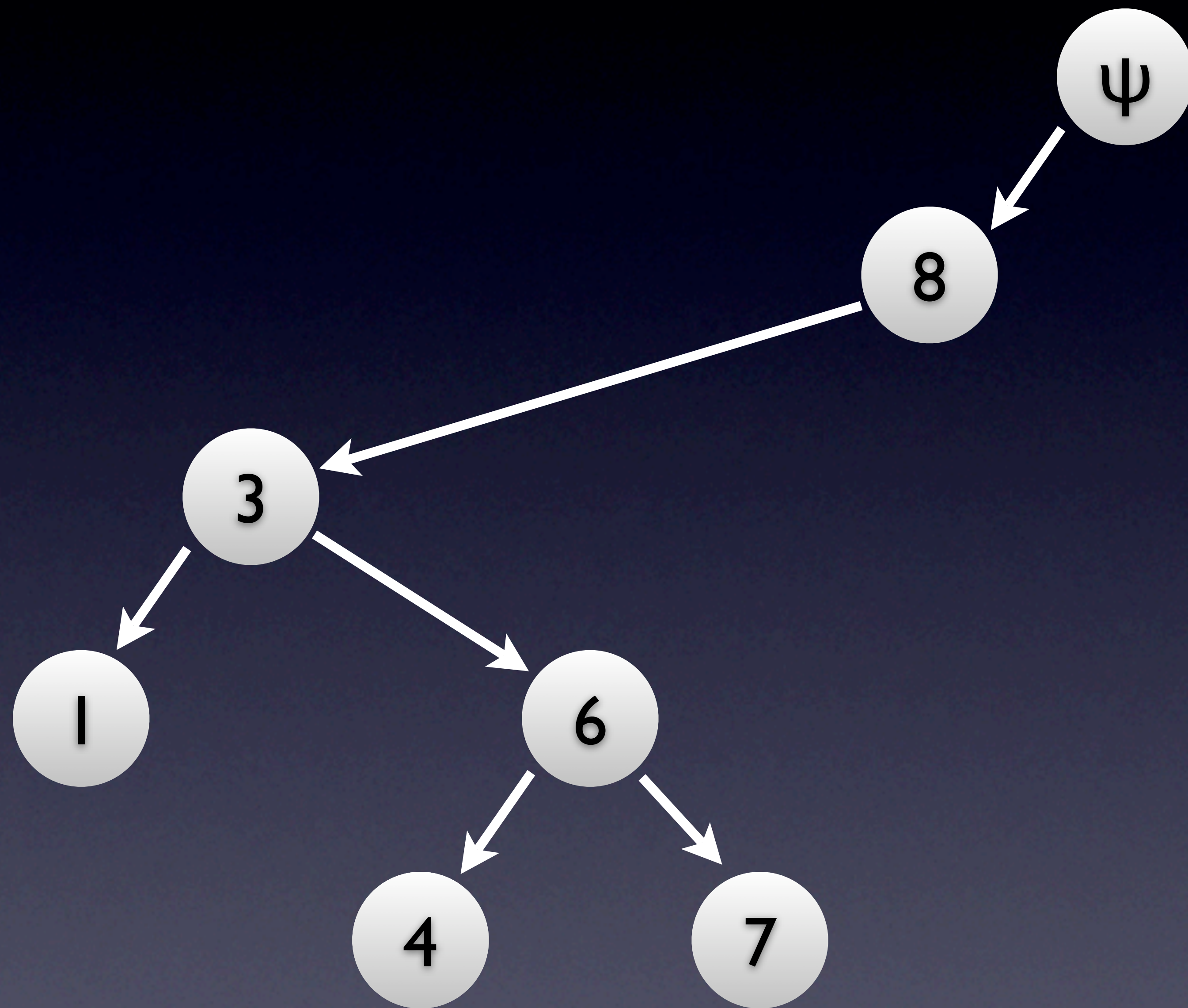
Delete 8, Take 2



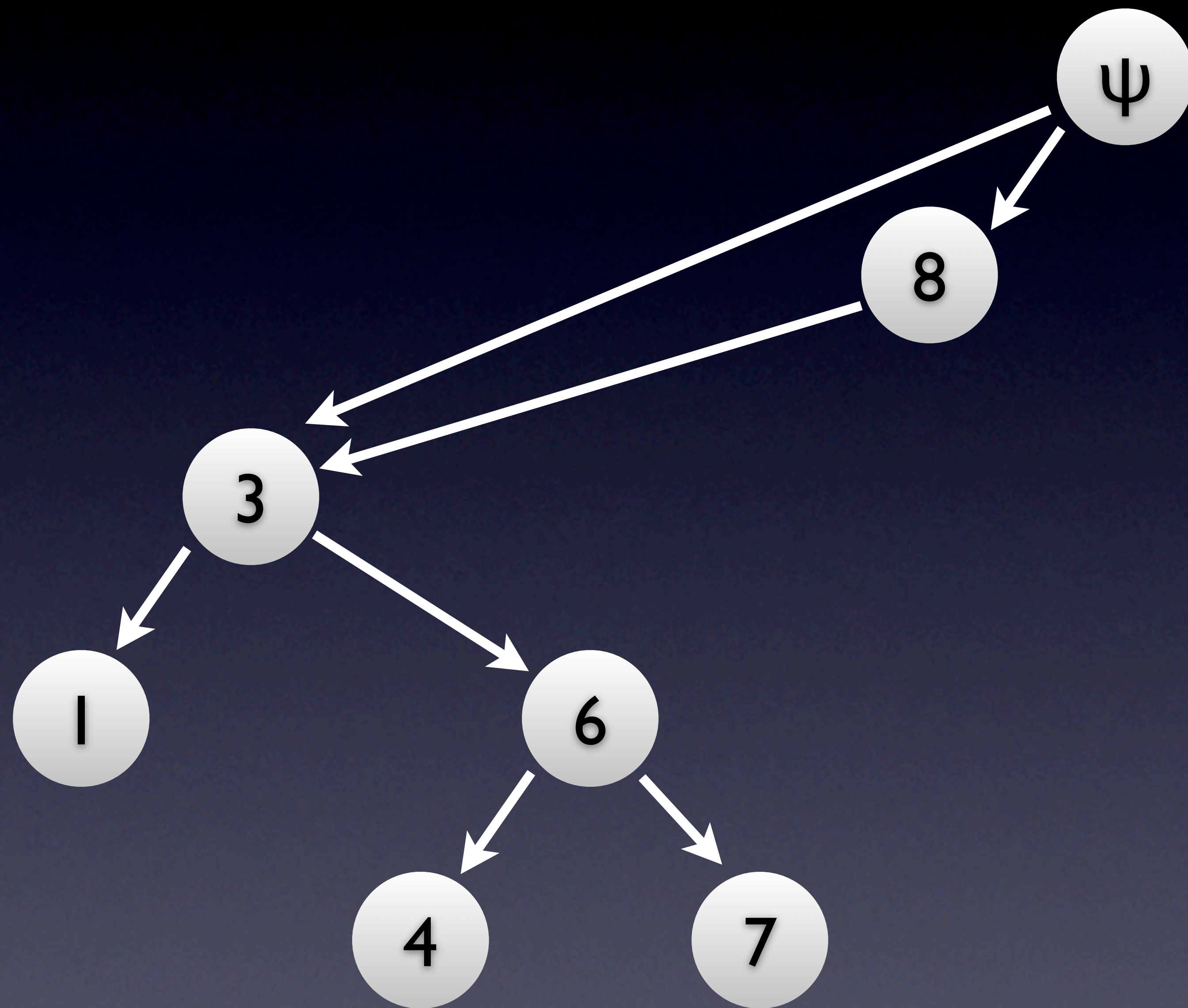
Delete 8, Take 2



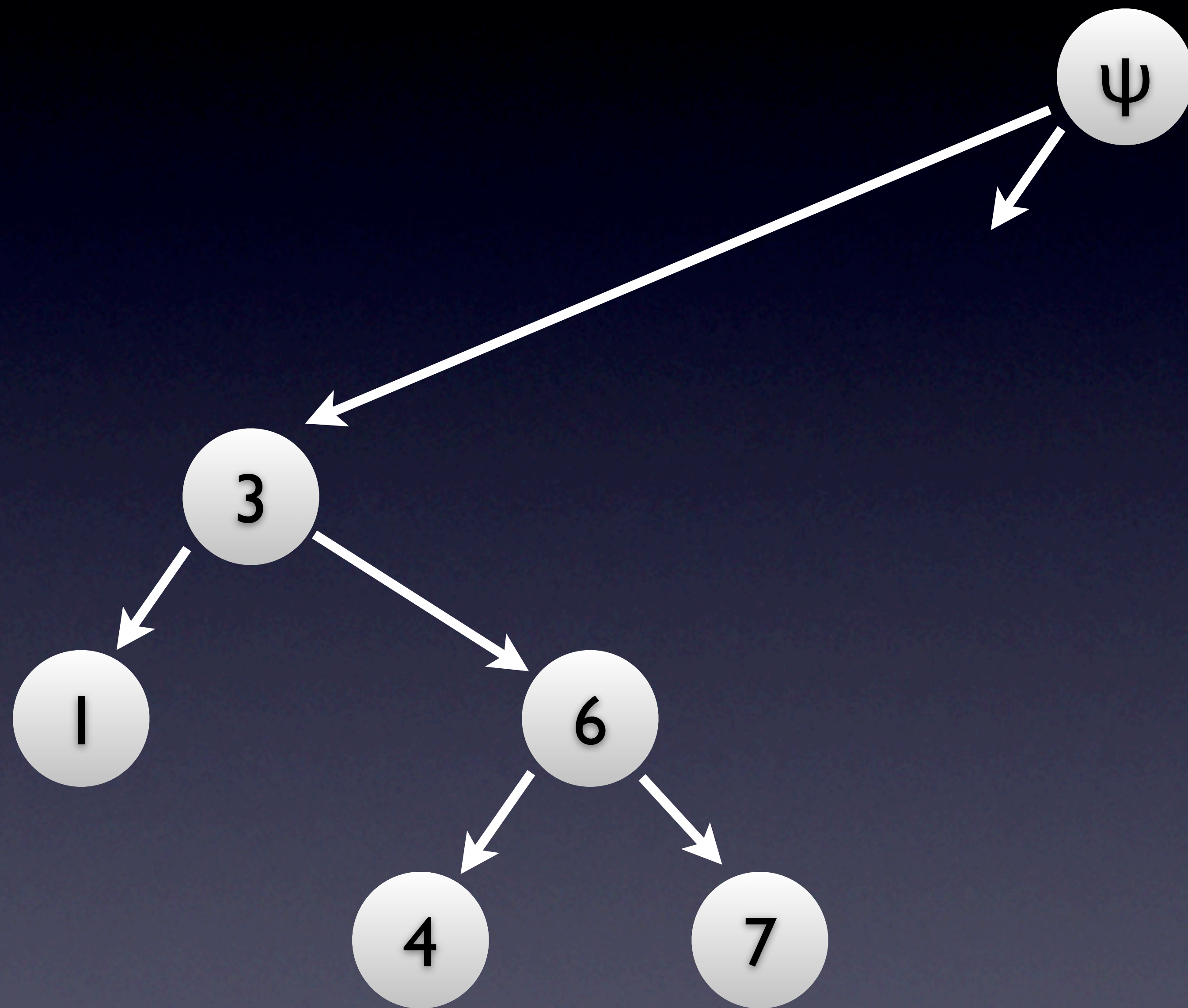
Delete 8, Take 2



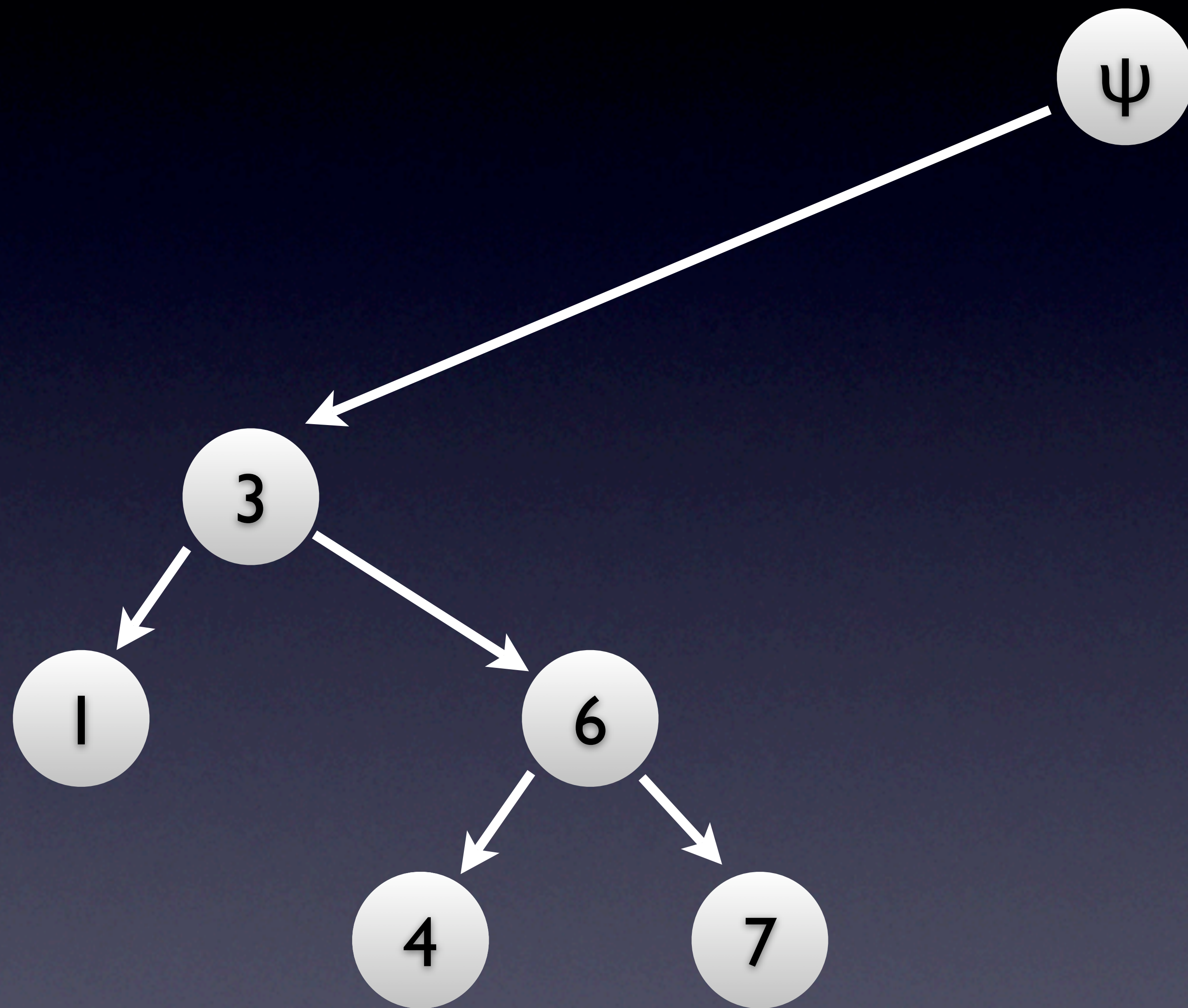
Delete 8, Take 2



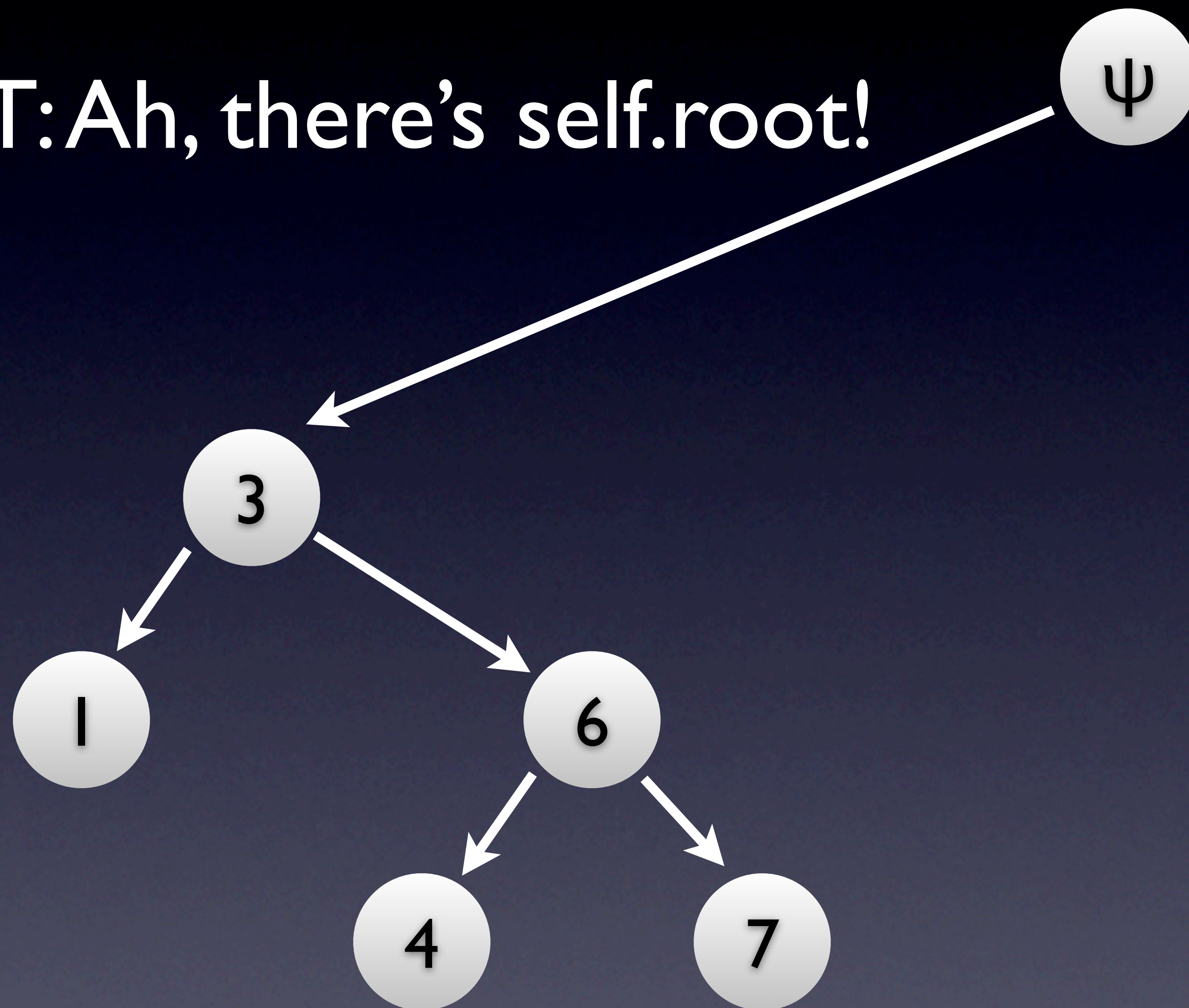
Delete 8, Take 2



Delete 8, Take 2

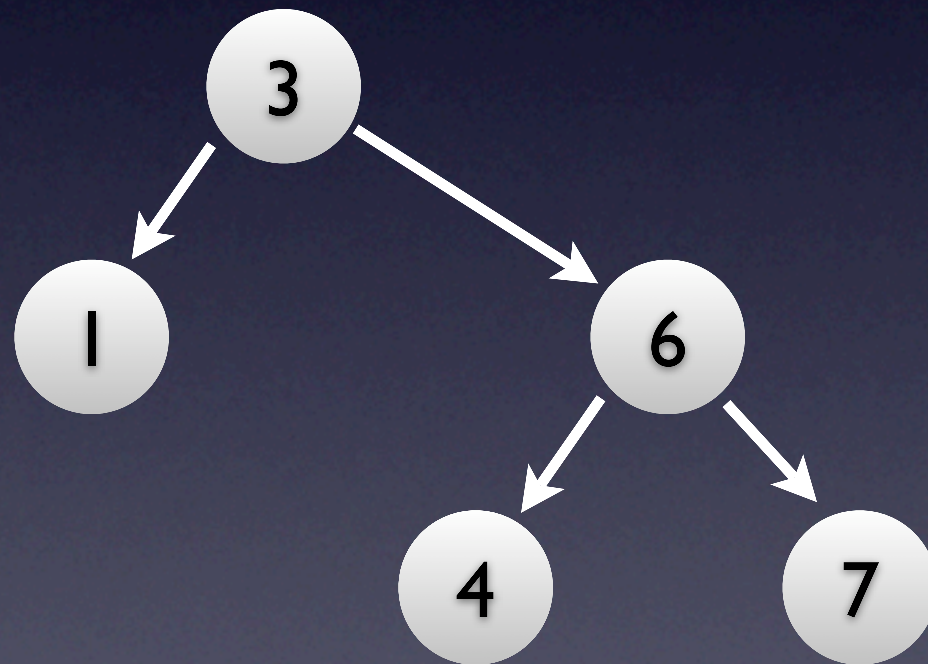


Delete 8, Take 2



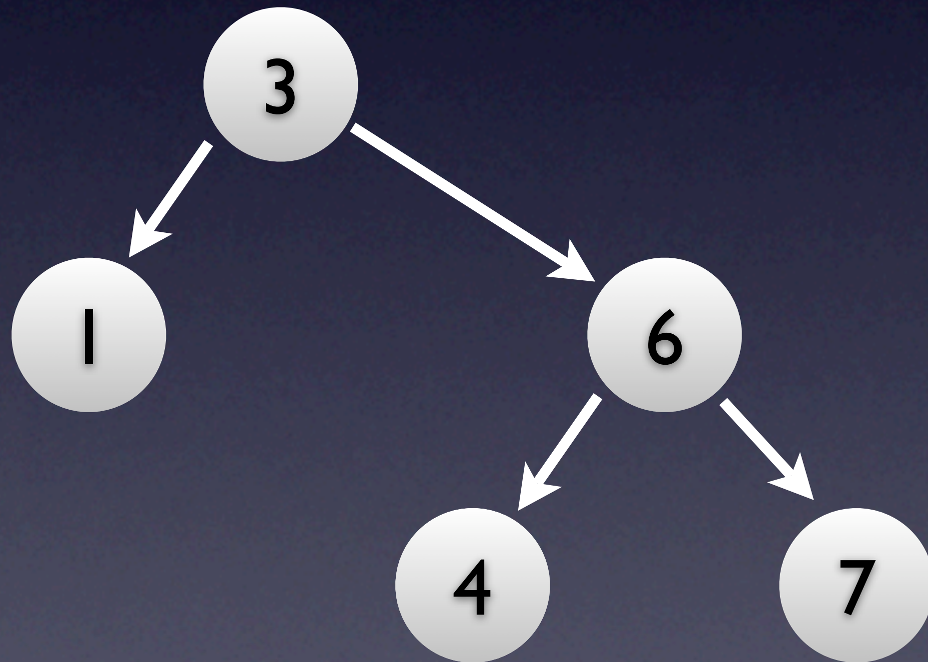
Delete 8, Take 2

BST: Ah, there's self.root!



Delete 8, Take 2

BST: Ah, there's self.root!



BST Deletion Hack

```
 1 class BST(object):
 2    def __init__(self):
 3        self.root = None
 4
 5    def delete(self, t):
 6        node = self.find(t)
 7        if node is self.root:
 8            pseudoroot = BSTnode(None, 0)
 9            pseudoroot.left = self.root
10            self.root.parent = pseudoroot
11            deleted = self.root.delete()
12            self.root = pseudoroot.left
13            self.root.parent = None
14            return deleted
15        if node is not None:
16            return node.delete()
```

Augmenting BSTs

‘cause you don’t wanna reinvent the wheel
for every new feature

Case Study: Rank

- Want to implement a data structure with the following operations
  - given a set S (initially empty)
  - **insert**(x): add x to S
  - **delete**(x): remove x from S
  - **rank**(x): # of $y \in S$ such that $y \leq x$

Implementing Rank

- Remember that BSTs will kick ass when we learn how to balance them
- Remember that BSTs are good with order relationships

Implementing Rank

- Remember that BSTs will kick ass when we learn how to balance them
- Remember that BSTs are good with order relationships

(again) BST Search

```
1 class BSTnode(object):
2     def __init__(self, parent, t):
3         self.key = t
3         self.parent = parent
4         self.left = None
5         self.right = None
6
7     def find(self, t):
8         if t == self.key:
9              return self
10        elif t < self.key:
11            if self.left is None:
12                return None
13            else:
14                return self.left.find(t)
15        else:
16            if self.right is None:
17                return None
18            else:
19                return self.right.find(t)
```

BST Search +Size

```
 1 class BSTnode(object):
 2    def __init__(self, parent, t):
 3        self.key = t
 3        self.parent = parent
 4        self.left = None
 5        self.right = None
+A        self.size = 1
 6
 7    def find(self, t):
 8        if t == self.key:
 9             return self
10        elif t < self.key:
11            if self.left is None:
12                return None
13            else:
14                return self.left.find(t)
15        else:
16            if self.right is None:
17                return None
18            else:
19                return self.right.find(t)
```

(again) BST Insertion

```
1 class BSTnode(object):
2    def __init__(self, parent, t):
3        self.key = t
4        self.parent = parent
4        self.left = None
5        self.right = None
6
7    def insert(self, t):
8        if t < self.key:
9            if self.left is None:
10               self.left = BSTnode(self, t)
11           else:
12               self.left.insert(t)
13       else:
14           if self.right is None:
15               self.right = BSTnode(self, t)
16           else:
17               self.right.insert(t)
```

BST Insertion +Size

```
 1 class BSTnode(object):
 2    def __init__(self, parent, t):
 3        self.key = t
 4        self.parent = parent
 4        self.left = None
 5        self.right = None
+A        self.size = 1
 6
 7    def insert(self, t):
+B        self.size += 1
 8        if t < self.key:
 9            if self.left is None:
10                self.left = BSTnode(self, t)
11            else:
12                self.left.insert(t)
13        else:
14            if self.right is None:
15                self.right = BSTnode(self, t)
16            else:
17                self.right.insert(t)
```

(again) BST Wrapper

```
 1 class BST(object):
 2     def __init__(self):
 3         self.root = None
 4
 5     def insert(self, t):
 6         if self.root is None:
 7             self.root = BSTnode(None, t)
 8         else:
 9             self.root.insert(t)
10
11     def find(self, t):
12         if self.root is None:
13             return None
14         else:
15             return self.root.find(t)
```

BST Wrapper +Size

```
1 class BST(object):
2     def __init__(self):
3         self.root = None
4 
5     def insert(self, t):
6         if self.root is None:
7             self.root = BSTnode(None, t)
8         else:
9             self.root.insert(t)
10
11    def find(self, t):
12        if self.root is None:
13            return None
14        else:
15            return self.root.find(t)
```

(again) BST Successor

```
 1 class BSTnode(object):
 2    def __init__(self, parent, t):
 3        self.key = t
 3        self.parent = parent
 4        self.left = None
 5        self.right = None
 6
 7    def minimum(self):
 8        current = self
 9        while current.left is not None:
10            current = current.left
11        return current
12
13    def successor(self):
14        if self.right is not None:
15            return self.right.minimum()
16        current = self
17        while current.parent is not None and current.parent.right is current:
18            current = current.parent
19        return current.parent
```

BST Successor +Size

```
 1 class BSTnode(object):
 2    def __init__(self, parent, t):
 3        self.key = t
 3        self.parent = parent
 4        self.left = None
 5        self.right = None
+A        self.size = 1
 6
 7    def minimum(self):
 8        current = self
 9        while current.left is not None:
10            current = current.left
11        return current
12
13    def successor(self):
14        if self.right is not None:
15            return self.right.minimum()
16        current = self
17        while current.parent is not None and current.parent.right is current:
18            current = current.parent
19        return current.parent
```

(again) BST Deletion

```
1 class BSTnode(object):
2    def delete(self):
3        if self.left is None or self.right is None:
4            if self is self.parent.left:
5                self.parent.left = self.left or self.right
6                if self.parent.left is not None:
7                    self.parent.left.parent = self.parent
8            else:
9                self.parent.right = self.left or self.right
10               if self.parent.right is not None:
11                   self.parent.right.parent = self.parent
12           return self
13       else:
14           s = self.successor()
15           self.key, s.key = s.key, self.key
16           return s.delete()
```

BST Deletion +Size

```
1  class BSTnode(object):
2     def delete(self):
3         if self.left is None or self.right is None:
4             if self is self.parent.left:
5                 self.parent.left = self.left or self.right
6                 if self.parent.left is not None:
7                     self.parent.left.parent = self.parent
8             else:
9                 self.parent.right = self.left or self.right
10                if self.parent.right is not None:
11                    self.parent.right.parent = self.parent
+A            current = self.parent
+B            while current is not None:
+C                current.size -= 1
+D                current = current.parent
15            return self
16        else:
17            s = self.successor()
18            self.key, s.key = s.key, self.key
19            return s.delete()
```

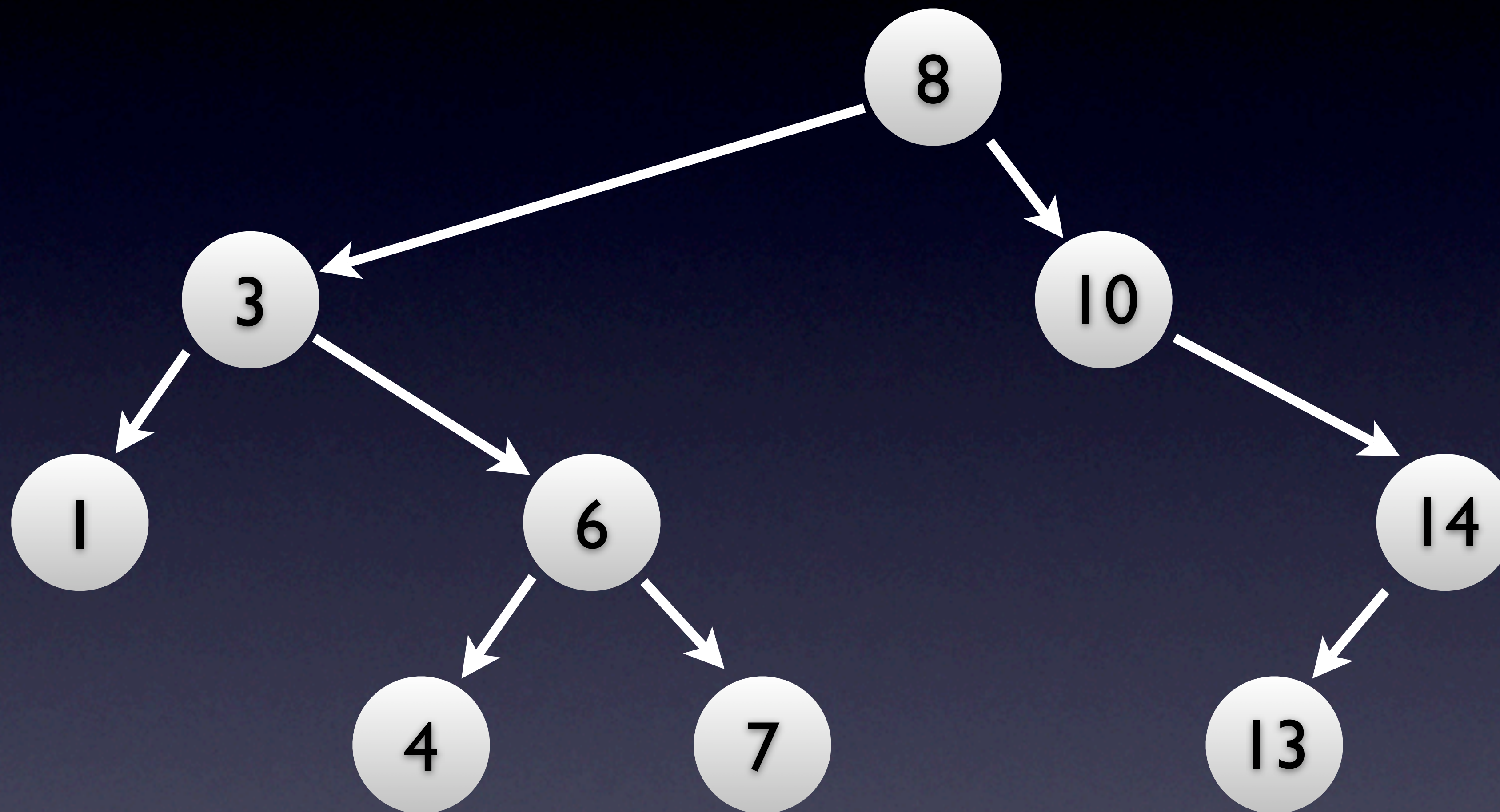
(again) Deletion Hack

```
1 class BST(object):
2    def __init__(self):
3        self.root = None
4
5    def delete(self, t):
6        node = self.find(t)
7        if node is self.root:
8            pseudoroot = BSTnode(None, 0)
9            pseudoroot.left = self.root
10           self.root.parent = pseudoroot
11           deleted = self.root.delete()
12           self.root = pseudoroot.left
13           self.root.parent = None
14           return deleted
15       if node is not None:
16           return node.delete()
```

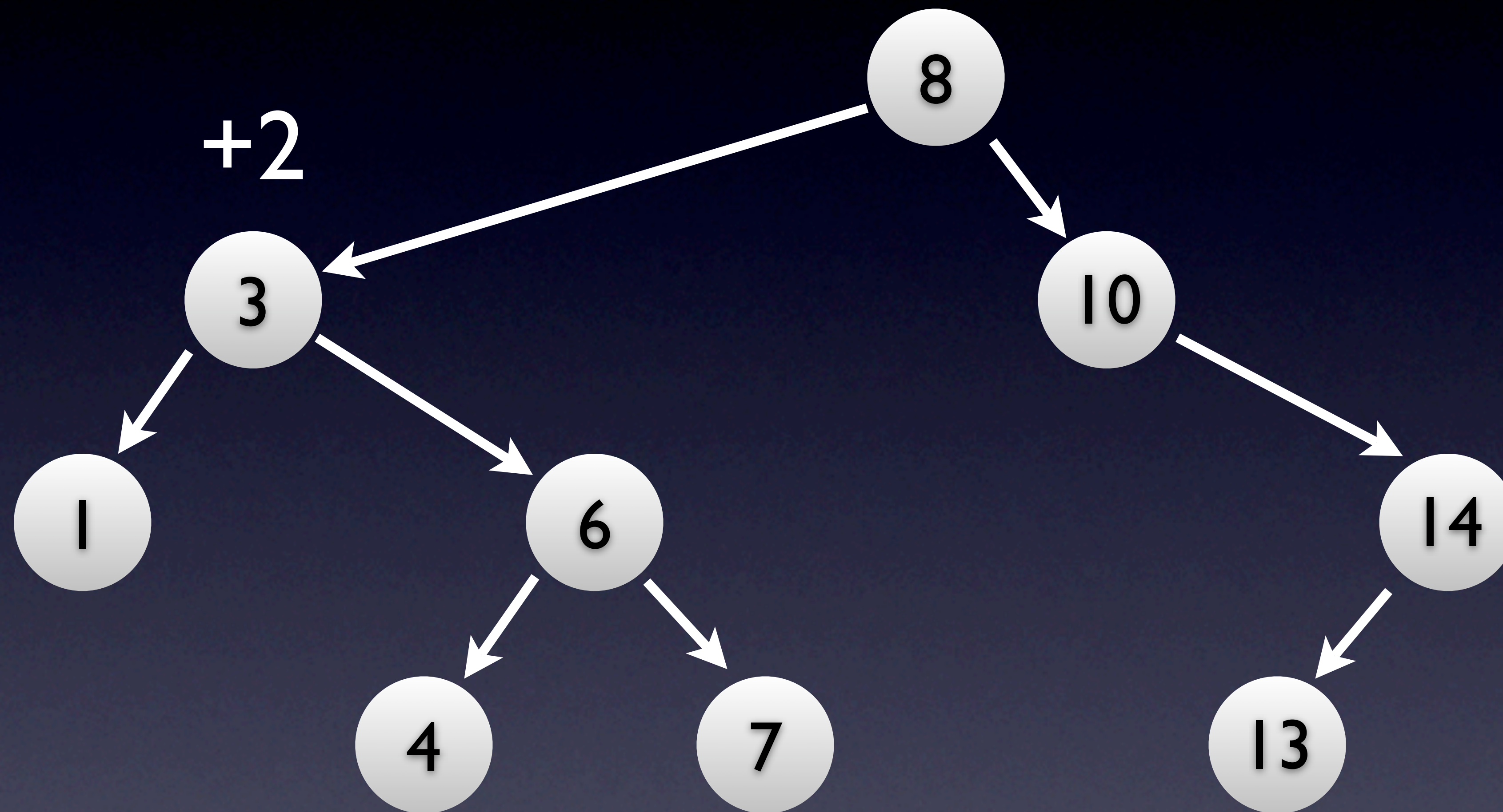
Deletion Hack +Size

```
 1 class BST(object):
 2    def __init__(self):
 3        self.root = None
 4
 5    def delete(self, t):
 6        node = self.find(t)
 7        if node is self.root:
 8            pseudoroot = BSTnode(None, 0)
 9            pseudoroot.left = self.root
10            self.root.parent = pseudoroot
11            deleted = self.root.delete()
12            self.root = pseudoroot.left
13            self.root.parent = None
14            return deleted
15        if node is not None:
16            return node.delete()
```

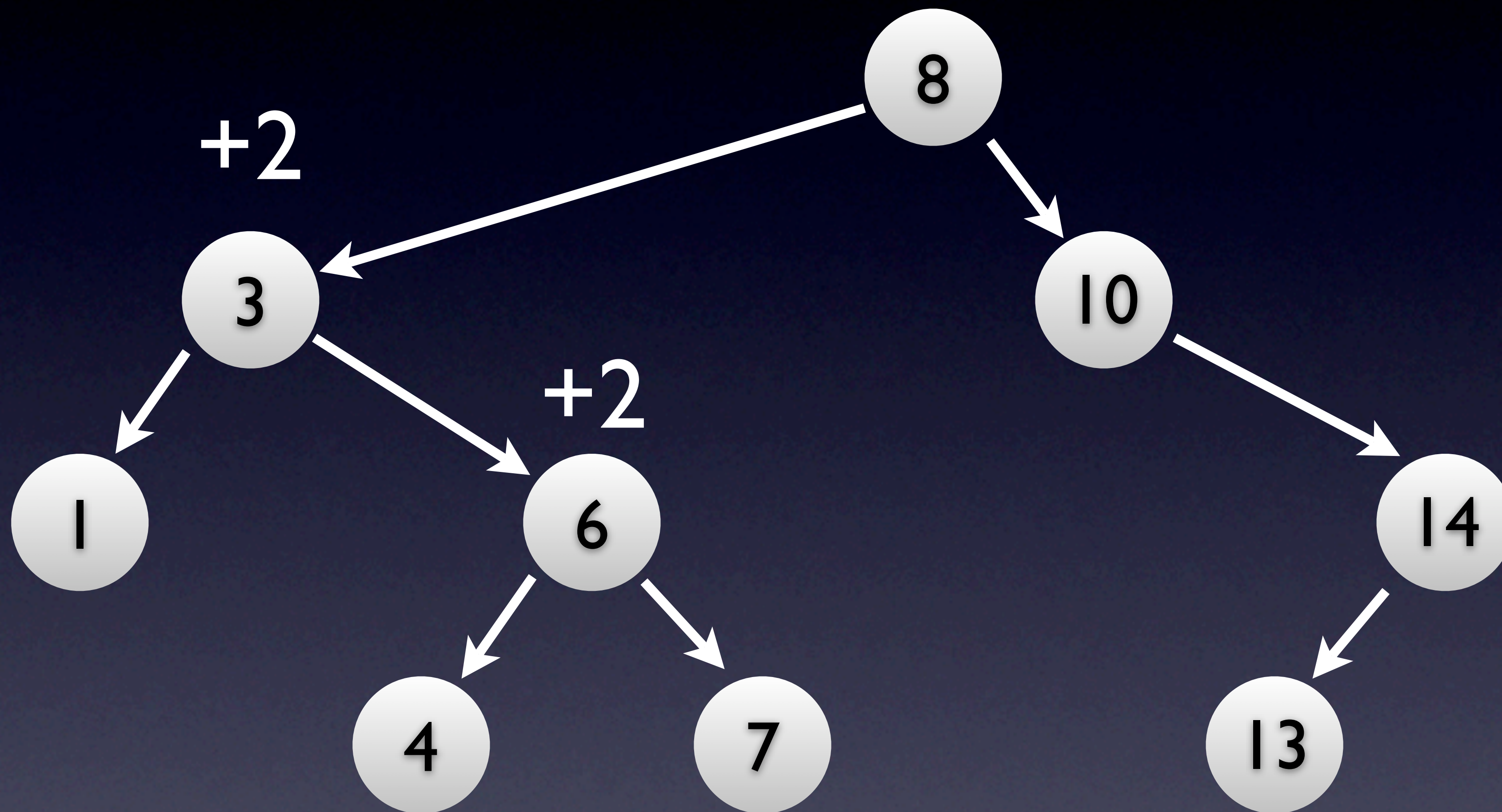
Rank of 7



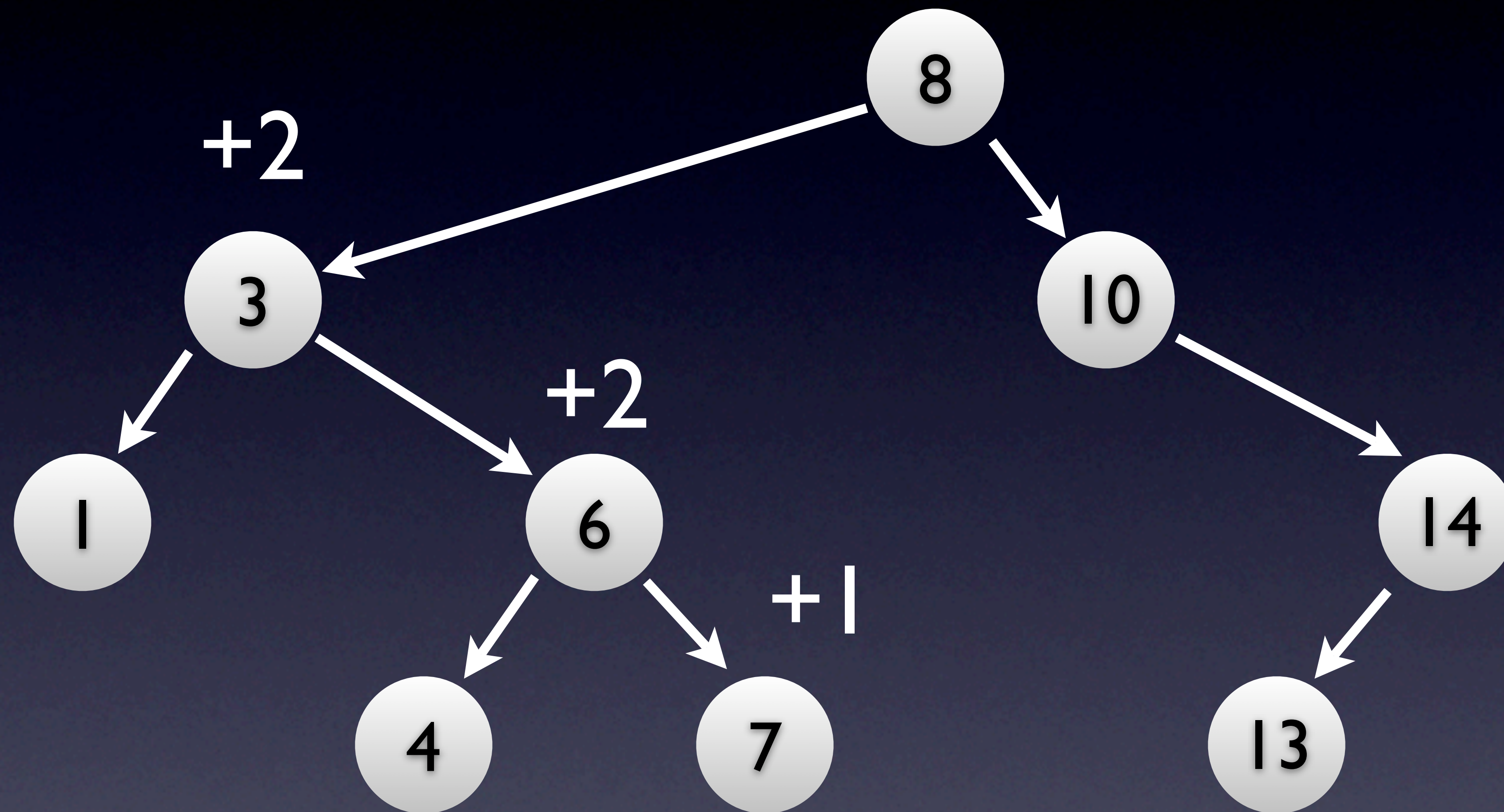
Rank of 7



Rank of 7



Rank of 7



BST Search +Size

```
1 class BSTnode(object):
2    def __init__(self, parent, t):
3        self.key = t
3        self.parent = parent
4        self.left = None
5        self.right = None
6        self.size = 1
7
8    def find(self, t):
9        if t == self.key:
10            return self
11        elif t < self.key:
12            if self.left is None:
13                return None
14            else:
15                return self.left.find(t)
16        else:
17            if self.right is None:
18                return None
19            else:
20                return self.right.find(t)
```

Finally, Rank!

```
1 class BSTnode(object):
2    def __init__(self, parent, t):
3        self.key = t
3        self.parent = parent
4        self.left = None
5        self.right = None
6        self.size = 1
7
8    def rank(self, t):
9        left_size = 0 if self.left is None else self.left.size
10       if t == self.key:
11           return left_size + 1
12       elif t < self.key:
13           if self.left is None:
14               return 0
15           else:
16               return self.left.rank(t)
17       else:
18           if self.right is None:
19               return left_size + 1
20           else:
21               return self.right.rank(t) + left_size + 1
```

And we're done!

- costan@mit.edu
- (617) 230-9694, no voicemail
- AIM: victorcostan
- Google Talk: costan@gmail.com
- 32G-8th Floor

v.Next

- Use a better name than ‘wrapper’ for BST
- Explain rank by example not by math
- This cannot be covered in 1 recitation