Упражнения Обходы дерева поиска

1. Какой *обход* дерева поиска дает упорядоченную последовательность ключей?
2. В результате *прямого* (pre-order) обхода некоторого дерева поиска T получена последовательность $A = [30, 20, 10, 15, 25, 23, 39, 35, 42]$. Каков результат *обратного* (post-order) обхода того же дерева?

Упражнения Поиск-1

В некотором бинарном дереве выполняется поиск значения $\mathbf{43}$. Какие последовательности просмотра ключей возможны, а какие нет? Почему?

1. $61, 52, 14, 17, 43$
2. $2, 3, 50, 40, 60, 43$
3. $10, 65, 31, 48, 37, 43$
4. $17, 77, 27, 66, 18, 43$

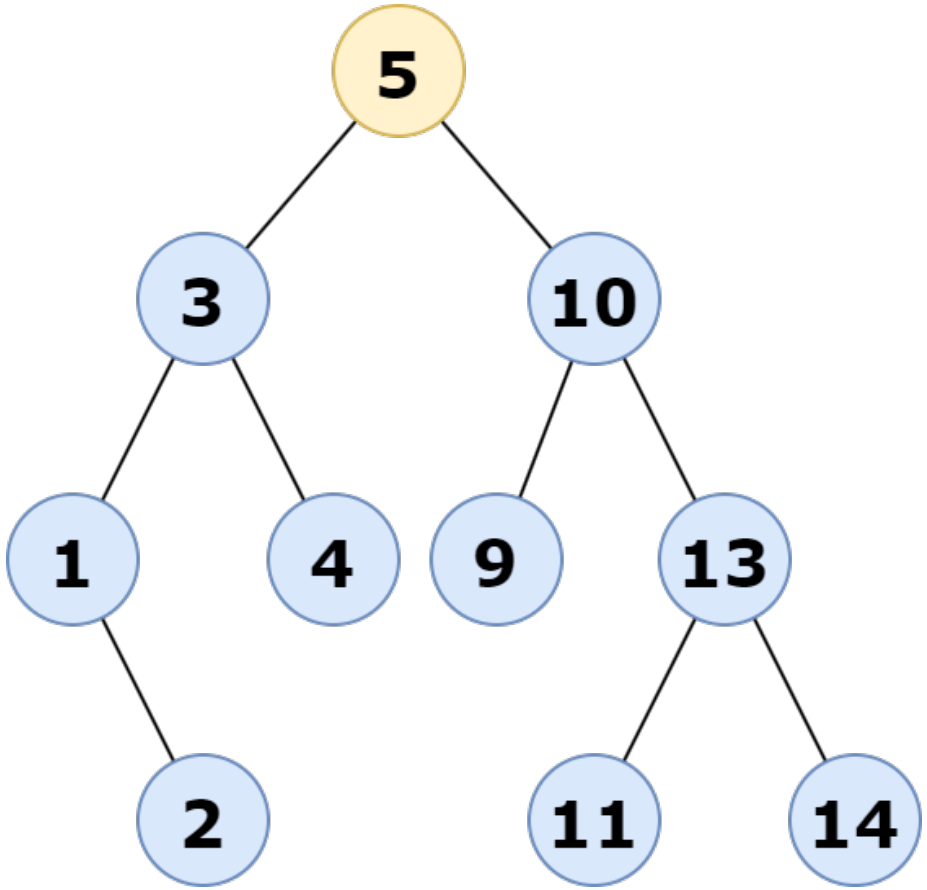
Упражнения Поиск-2

Где в бинарном дереве поиска находится *минимальный* и *максимальный* элементы?

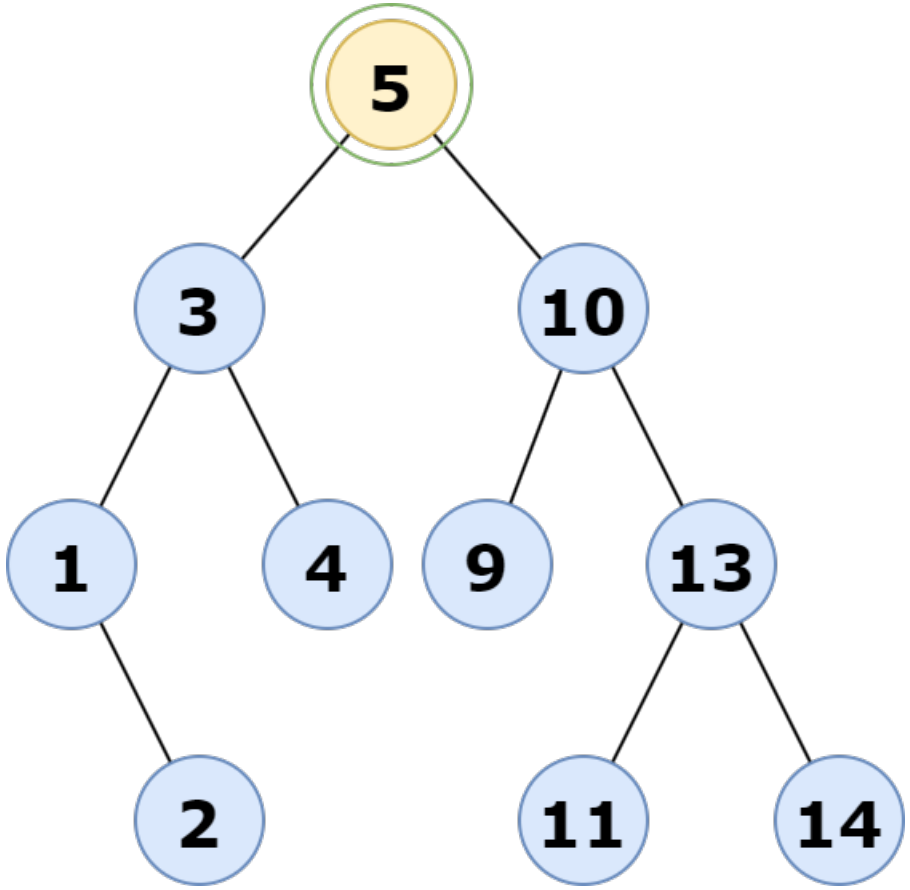
Разработайте *нерекурсивные* алгоритмы **`treeMax(…)`** и **`treeMin(…)`** для решения этих задач.

Итератор для бинарного дерева поиска

Предыдущий `treePredecessor`

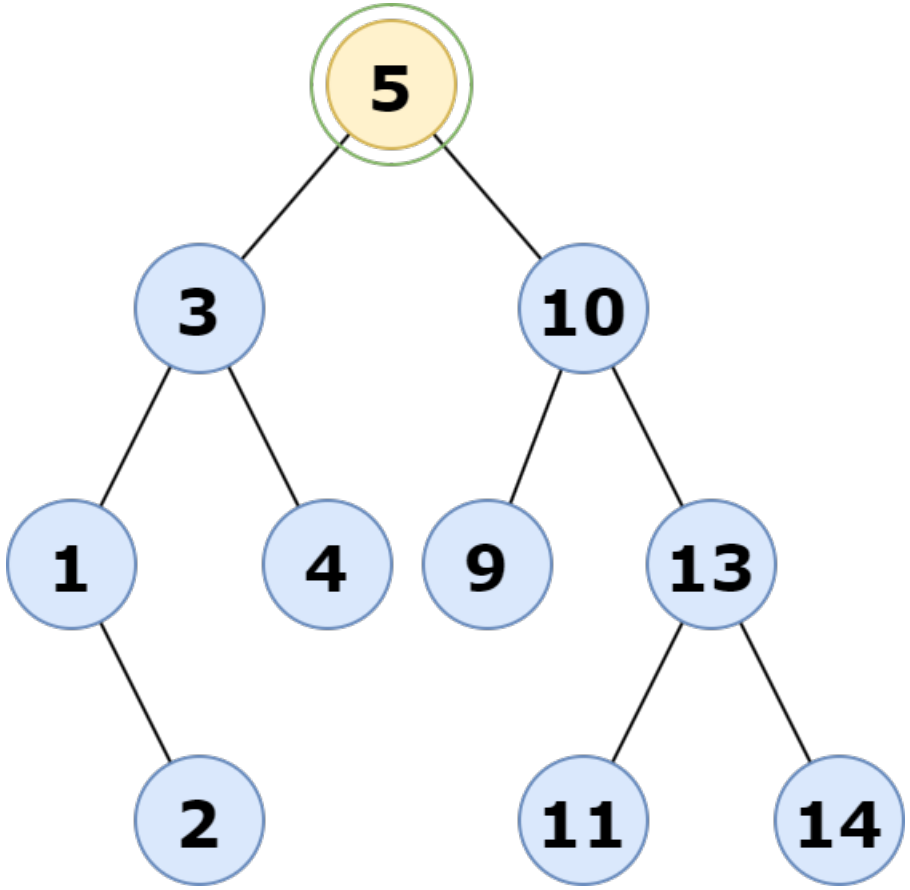


Предыдущий **treePredecessor** case 1



Необходимо найти предыдущий по порядку элемент для ключа **5**...

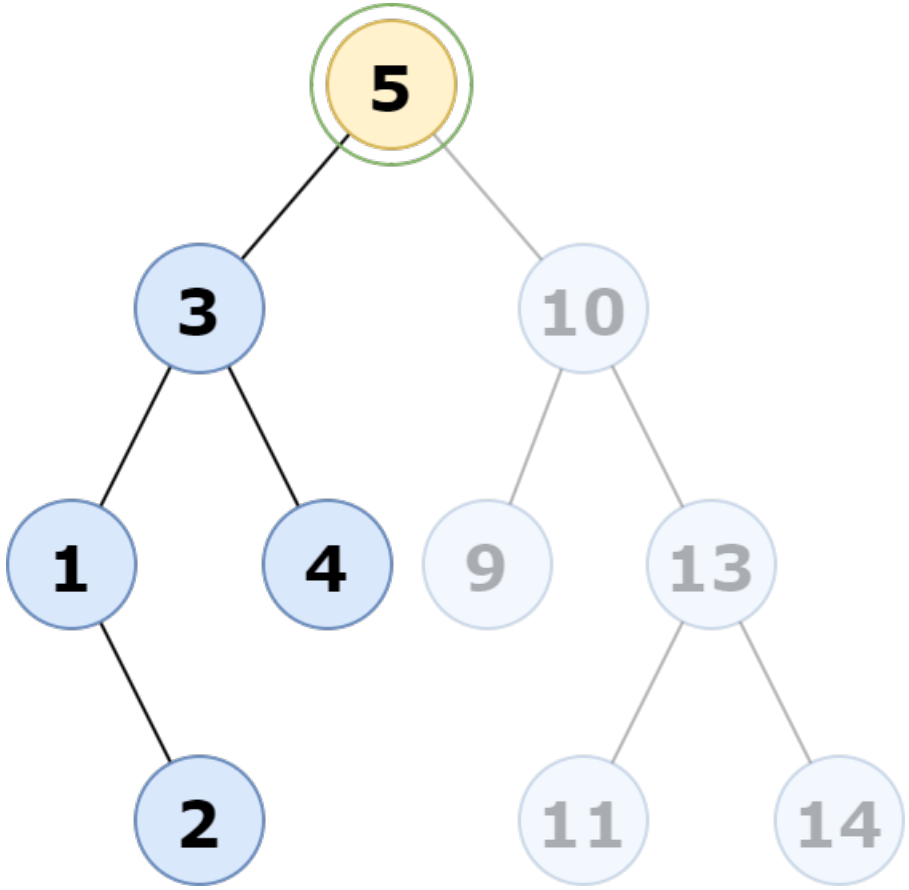
Предыдущий **treePredecessor** case 1



Необходимо найти предыдущий по порядку элемент для ключа **5**...

В каком под-дереве его искать?

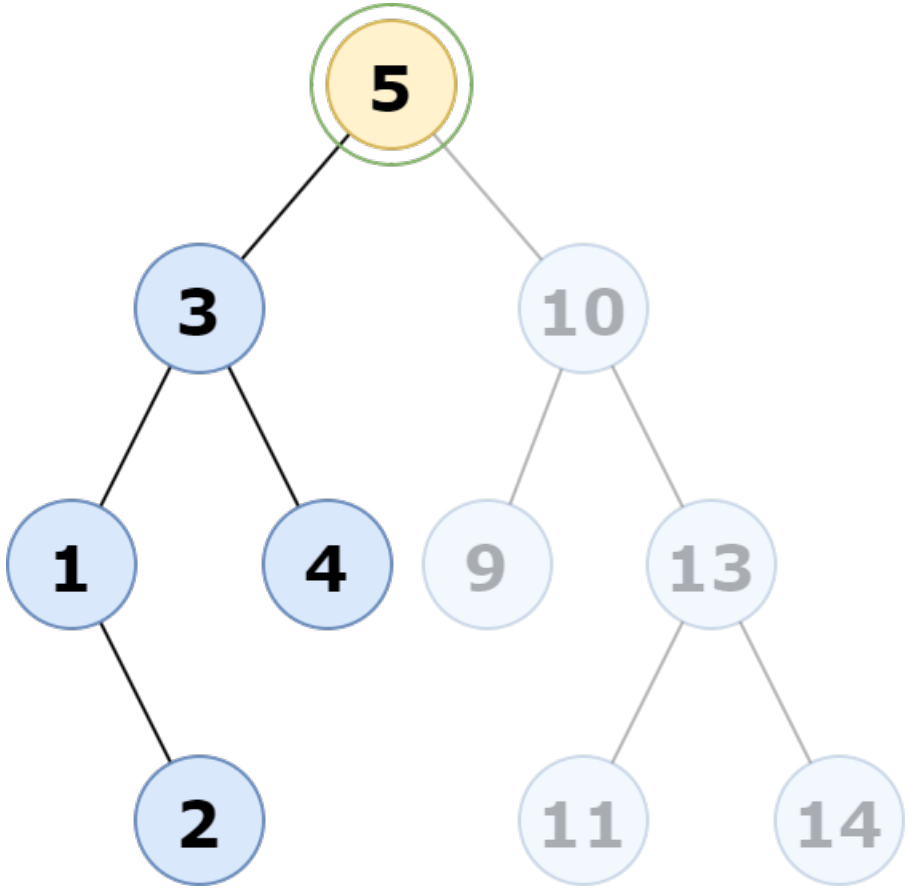
Предыдущий **treePredecessor** case 1



Необходимо найти предыдущий по порядку элемент для ключа **5**...

В каком под-дереве его искать?

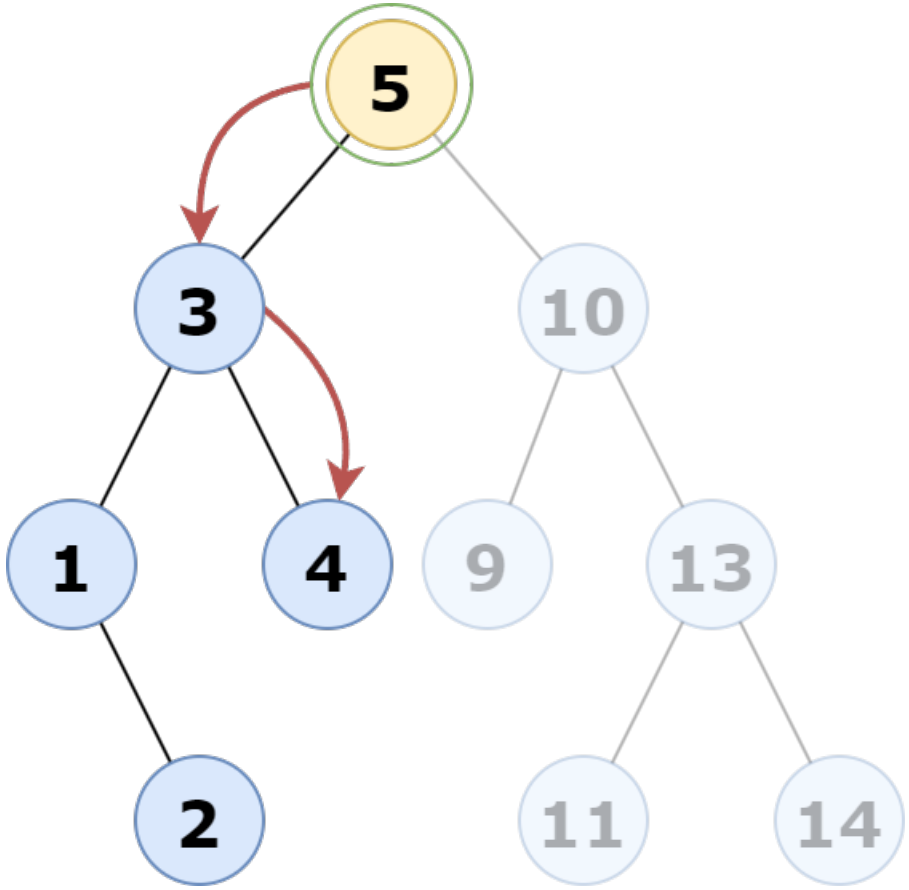
Предыдущий `treePredecessor` case 1



Необходимо найти предыдущий по порядку элемент для ключа **5**...

В этом под-дереве нужно найти максимум (**самый правый** элемент)

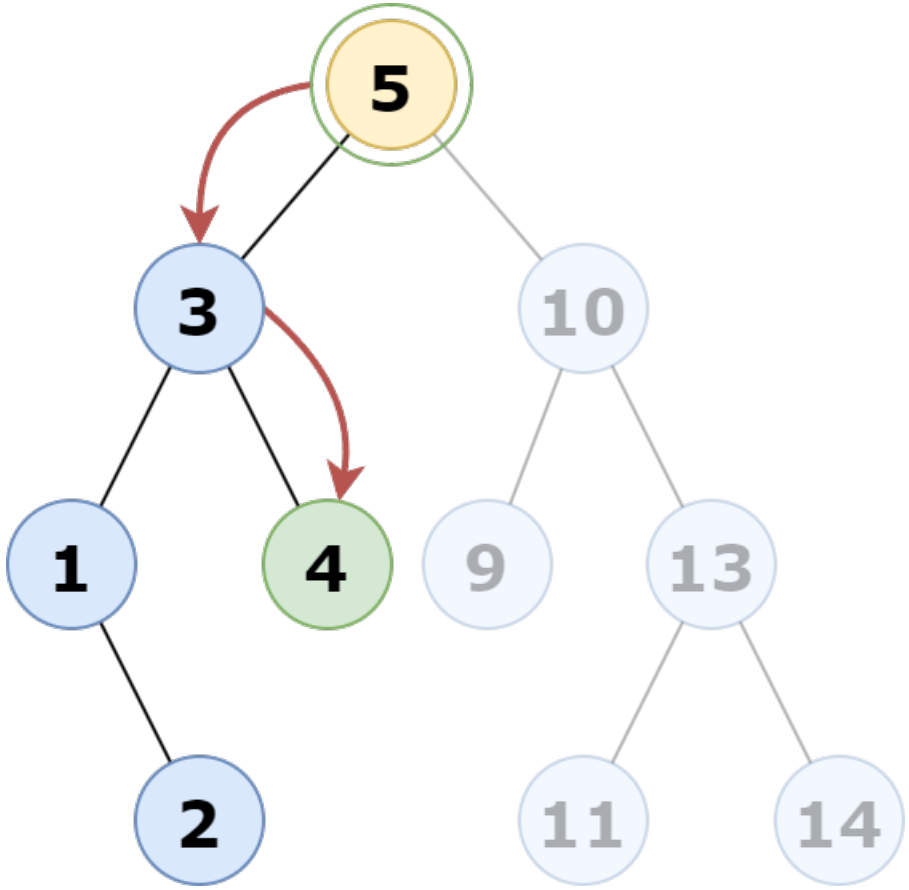
Предыдущий **`treePredecessor`** case 1



Необходимо найти предыдущий по порядку элемент для ключа **5**...

В этом под-дереве нужно найти максимум (**самый правый** элемент)

Предыдущий **treePredecessor** case 1

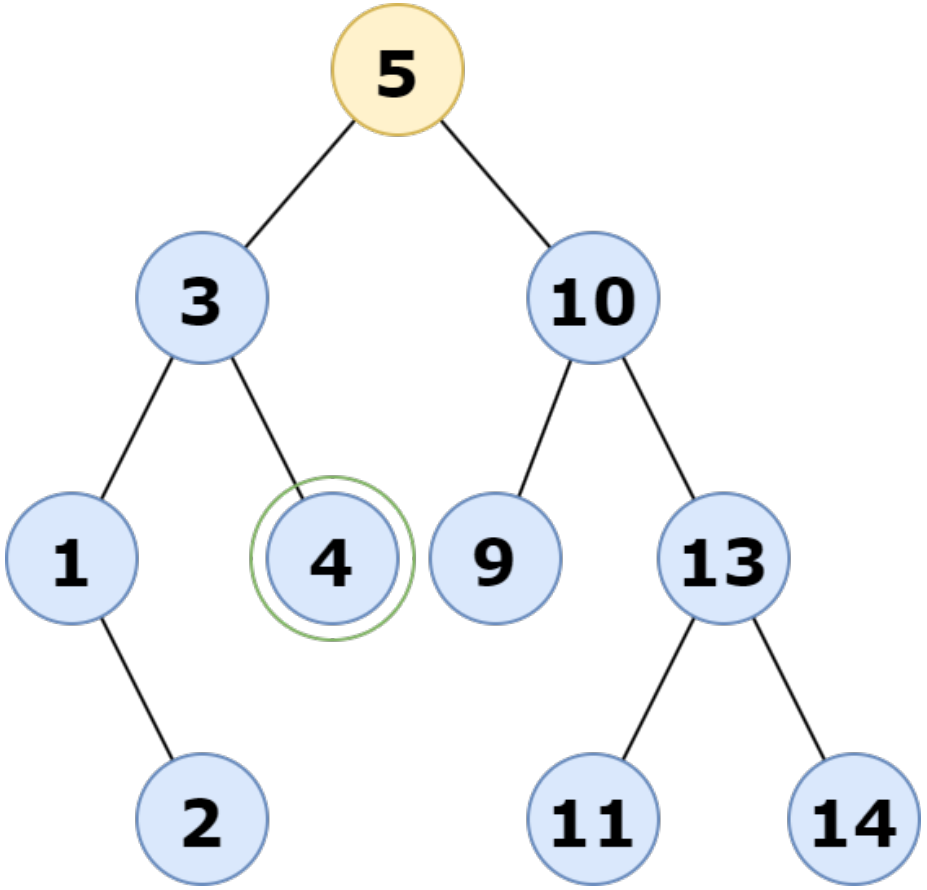


Необходимо найти предыдущий по порядку элемент для ключа **5**...

В этом под-дереве нужно найти максимум (**самый правый** элемент)

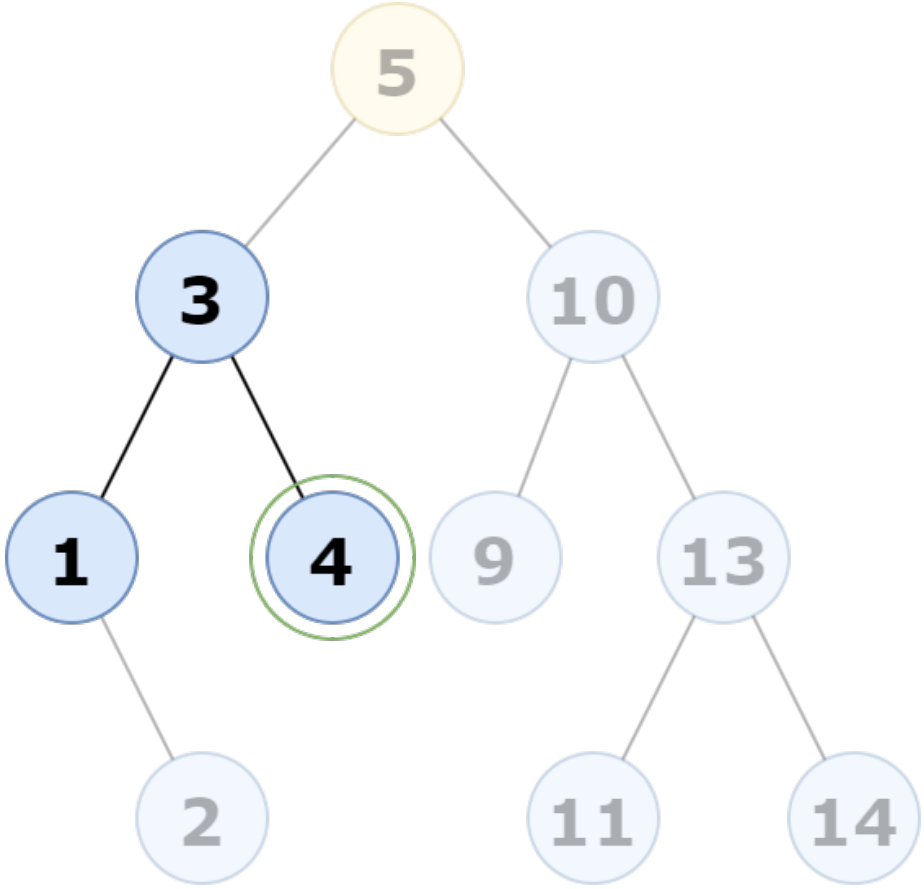
```
treePredecessor(5) -> 4
```

Предыдущий **treePredecessor** case 2



Необходимо найти предыдущий по порядку элемент для ключа **4**...

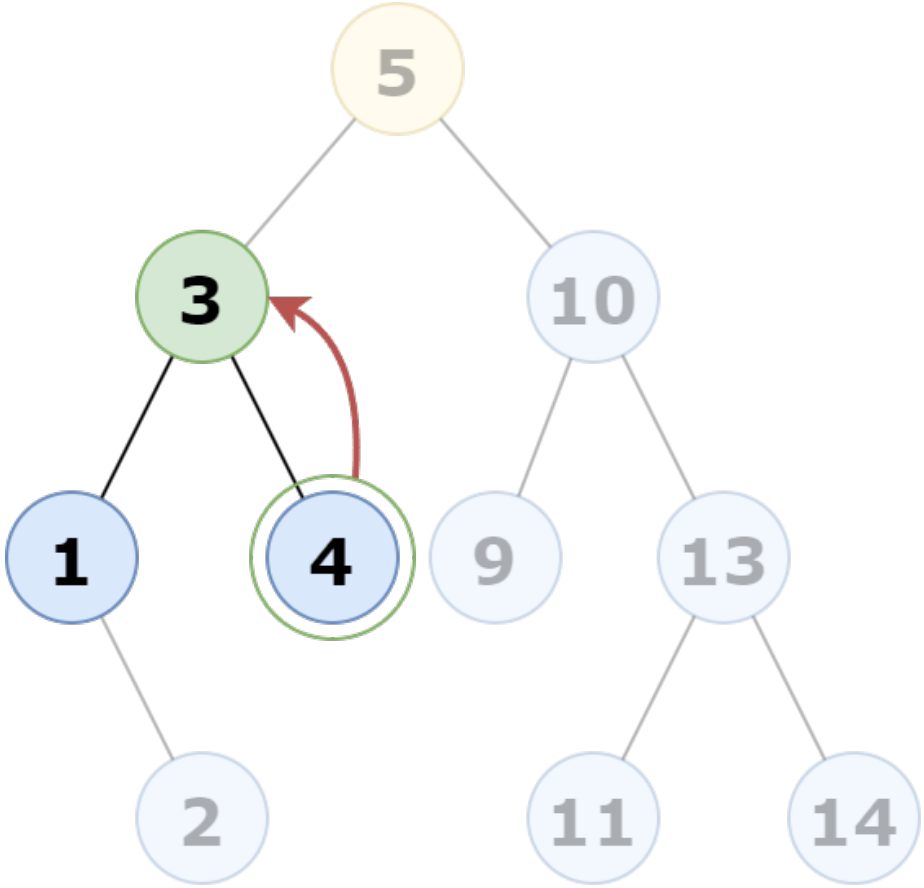
Предыдущий **treePredecessor** case 2



Необходимо найти предыдущий по порядку элемент для ключа **4**...

В этом случае мы ограничены **тремя узлами**

Предыдущий **treePredecessor** case 2

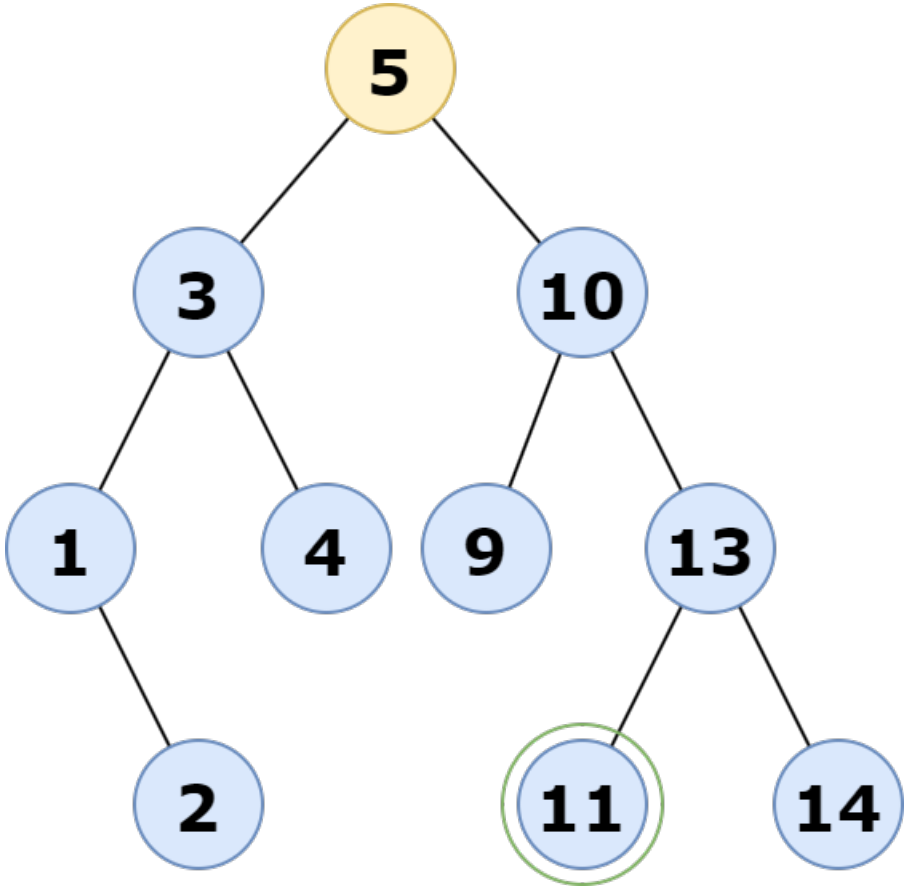


Необходимо найти предыдущий по порядку элемент для ключа **4**...

Нас интересует **прямой родитель** узла

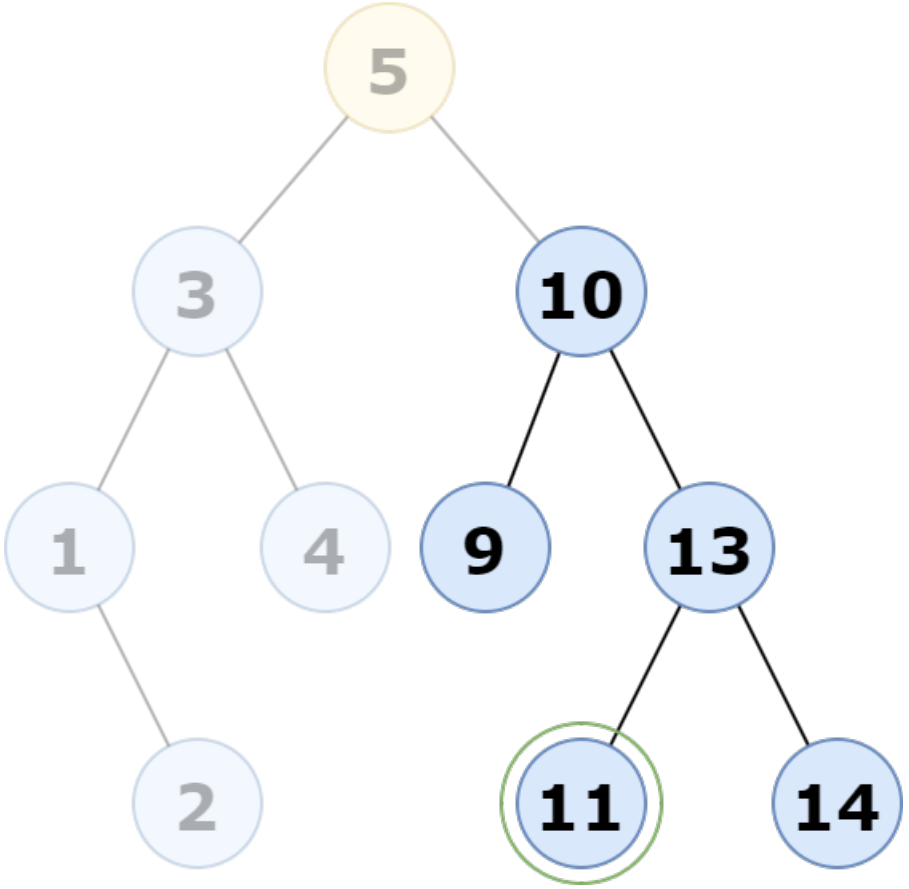
```
treePredecessor(4) -> 3
```

Предыдущий **treePredecessor** case 3



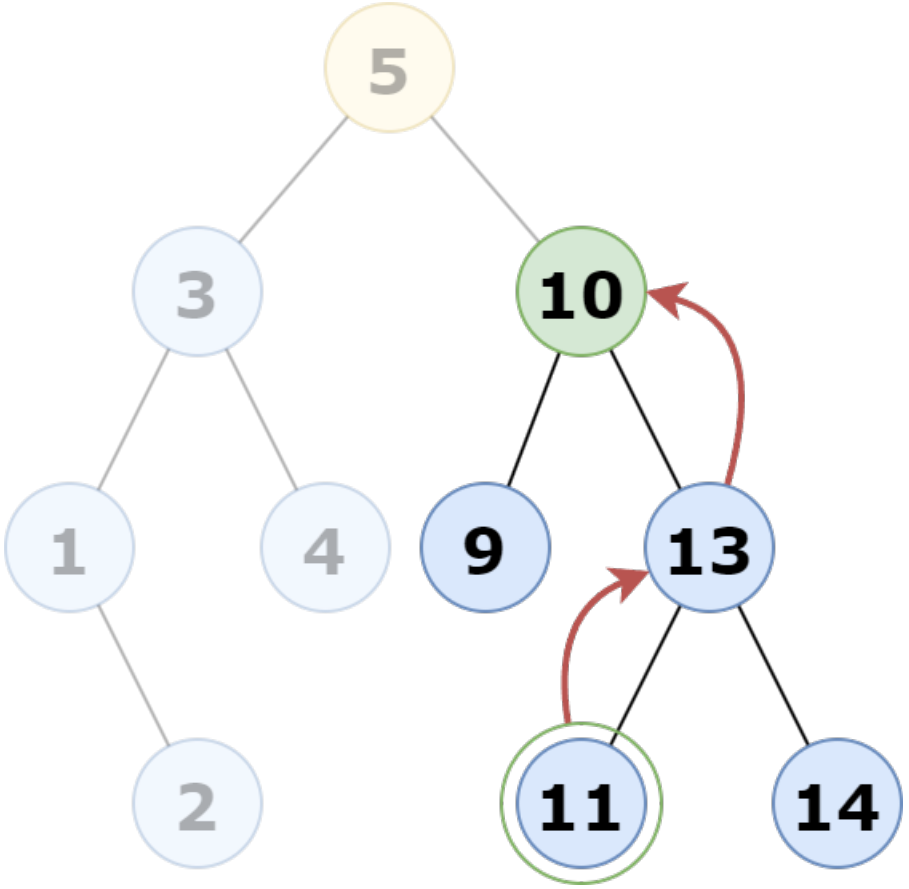
Необходимо найти предыдущий по порядку элемент для ключа **11**...

Предыдущий **treePredecessor** case 3



Необходимо найти предыдущий по порядку элемент для ключа **11**...

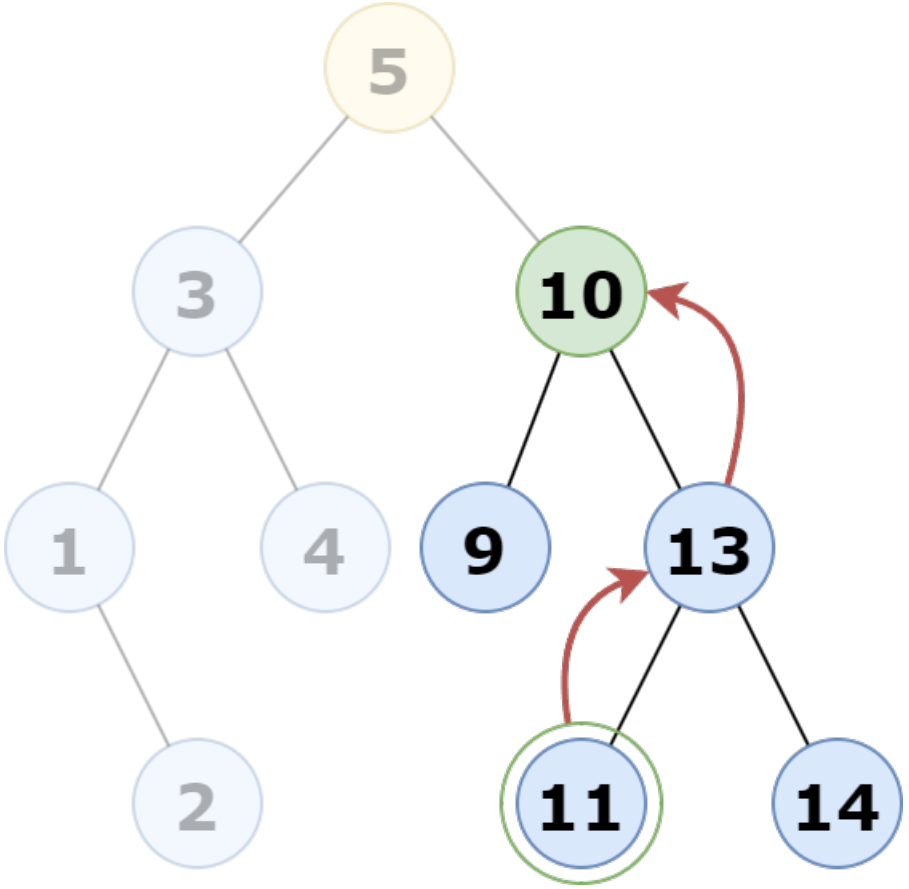
Предыдущий **treePredecessor** case 3



Необходимо найти предыдущий по порядку элемент для ключа **11**...

Как сформулировать условие остановки подъема наверх?

Предыдущий **treePredecessor** case 3



Необходимо найти предыдущий по порядку элемент для ключа **11**...

Пока узел не будет **правым ребенком** своего родителя

```
treePredecessor(4) -> 3
```

Предыдущий `treePredecessor`

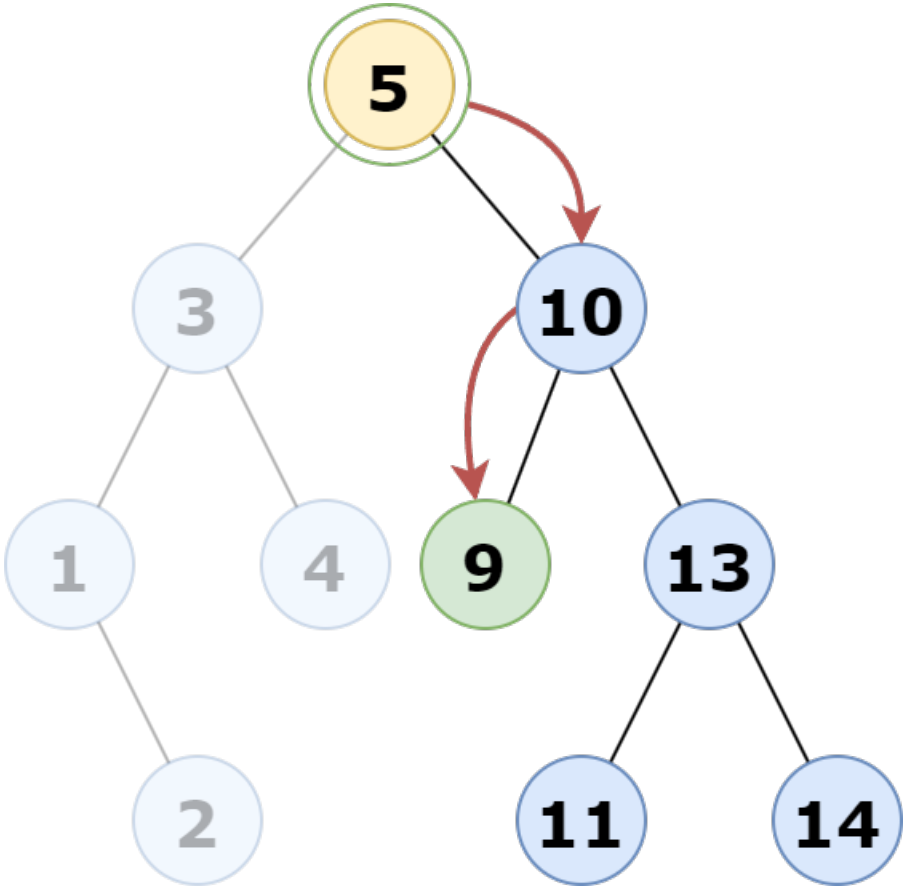
Итак...

1. Определяющим фактором в поиске предыдущего значения будет наличие/отсутствие **левого поддерева** у выбранного нами узла
2. Отдельного внимания заслуживает минимальный элемент

Следующий **`treeSuccessor`**

Все действия по поиску следующего для выбранного нами узла выполняются симметрично...

Следующий **treeSuccessor** case 1



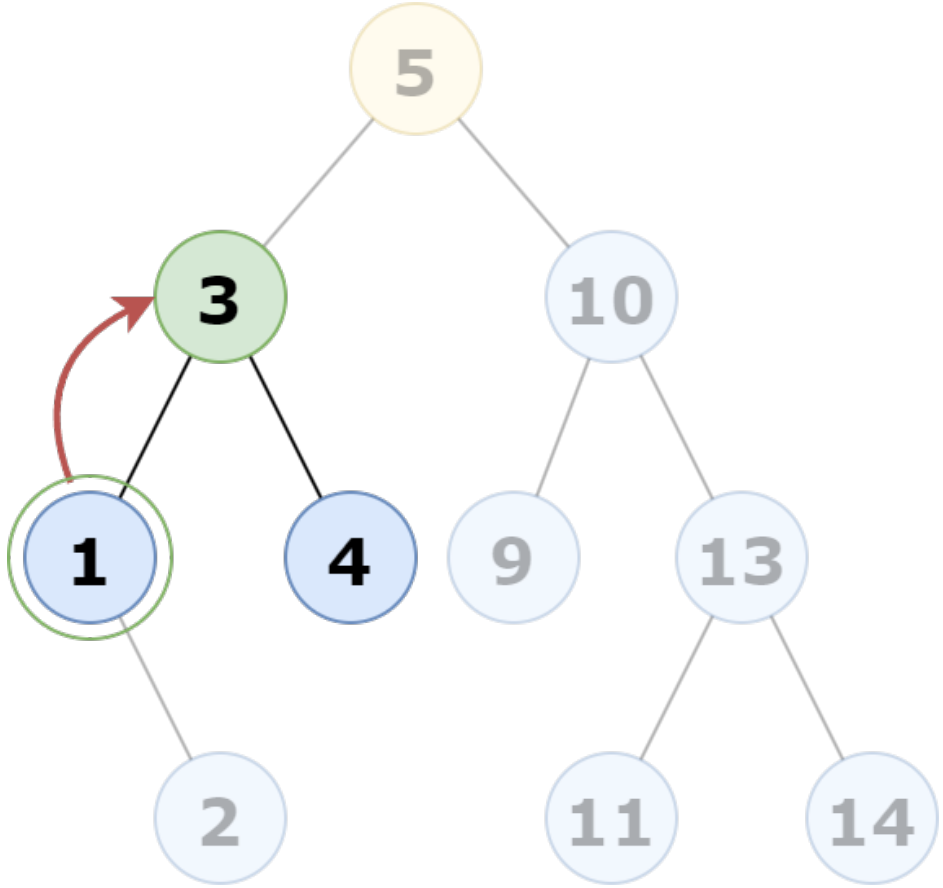
Необходимо найти предыдущий по порядку элемент для ключа **5**...

В этом под-дереве нужно найти минимум (**самый левый** элемент)

```
treeSuccessor(5) -> 9
```

Следующий **treeSuccessor**

case 2



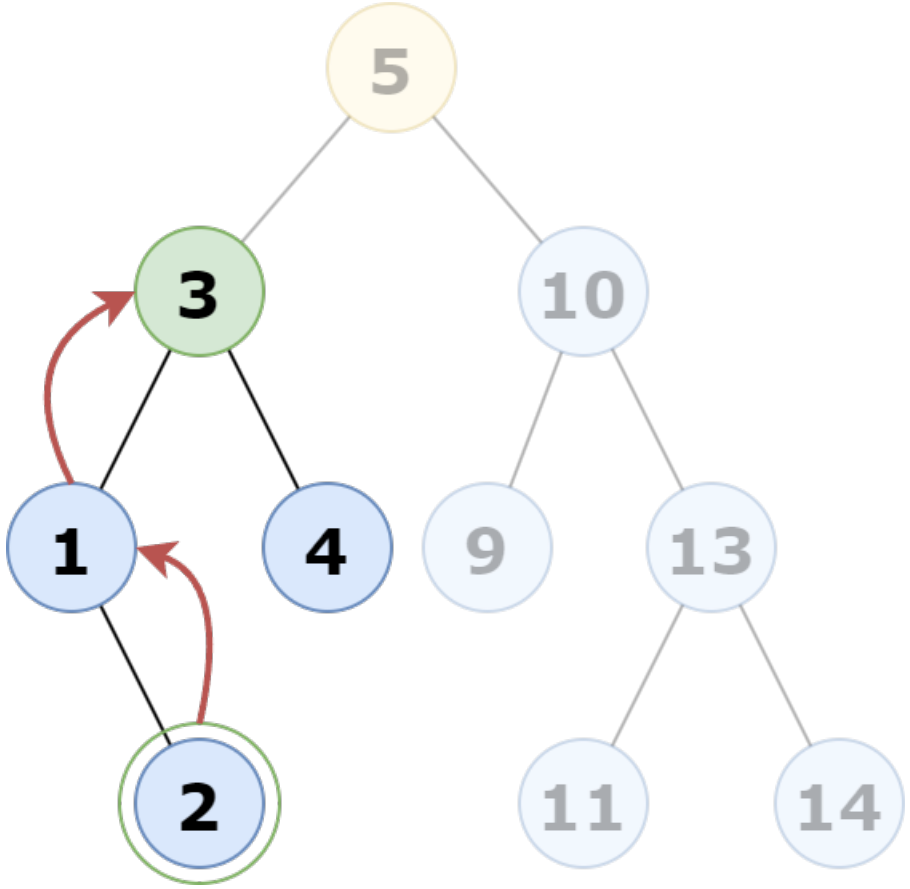
Необходимо найти предыдущий по порядку элемент для ключа **1**...

Прямой родитель для левого ребенка

```
treeSuccessor(5) -> 9
```

Следующий **treeSuccessor**

case 3



Необходимо найти предыдущий по порядку элемент для ключа **1**...

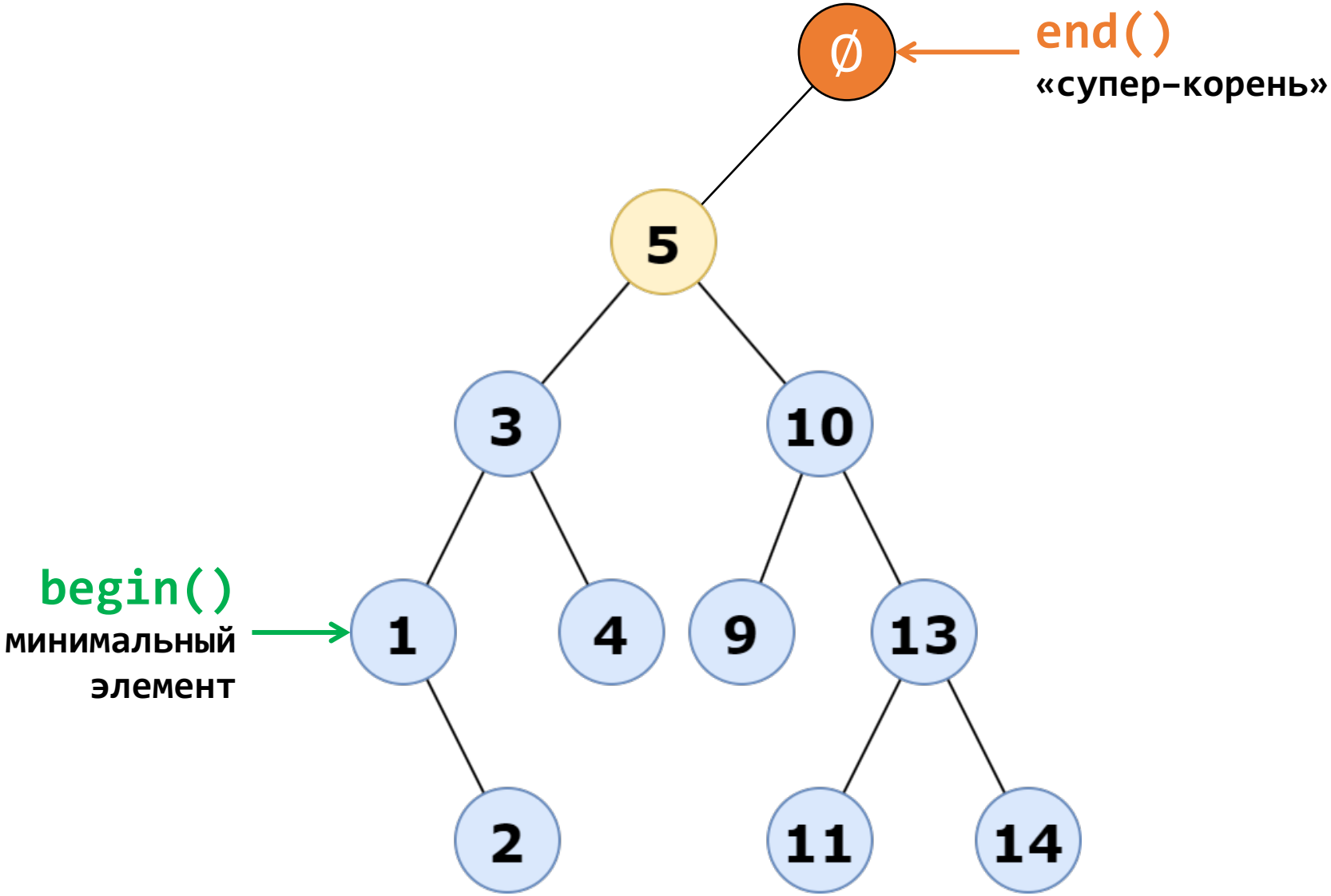
Пока узел не будет левым ребенком **своего родителя**

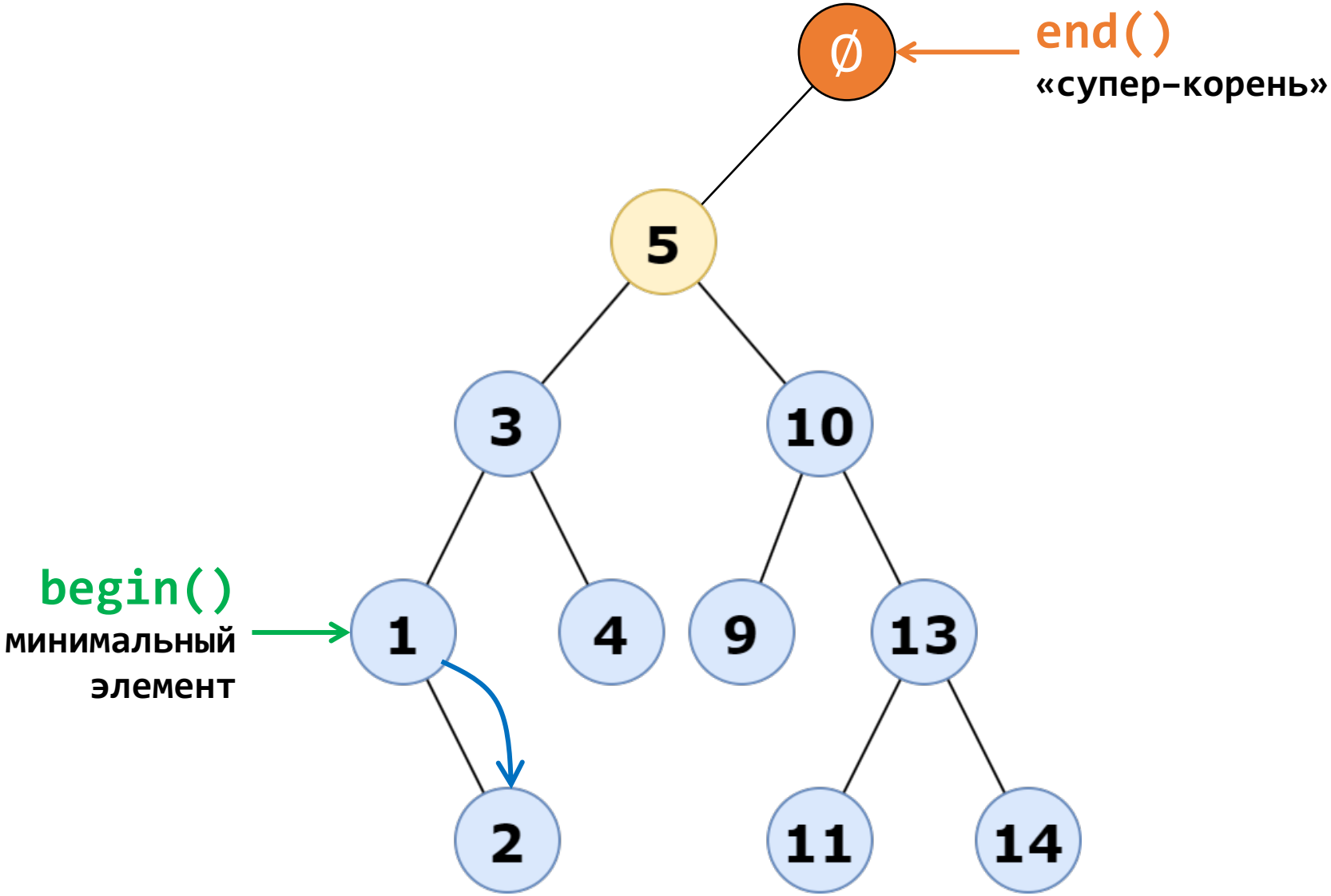
```
treeSuccessor(5) -> 9
```

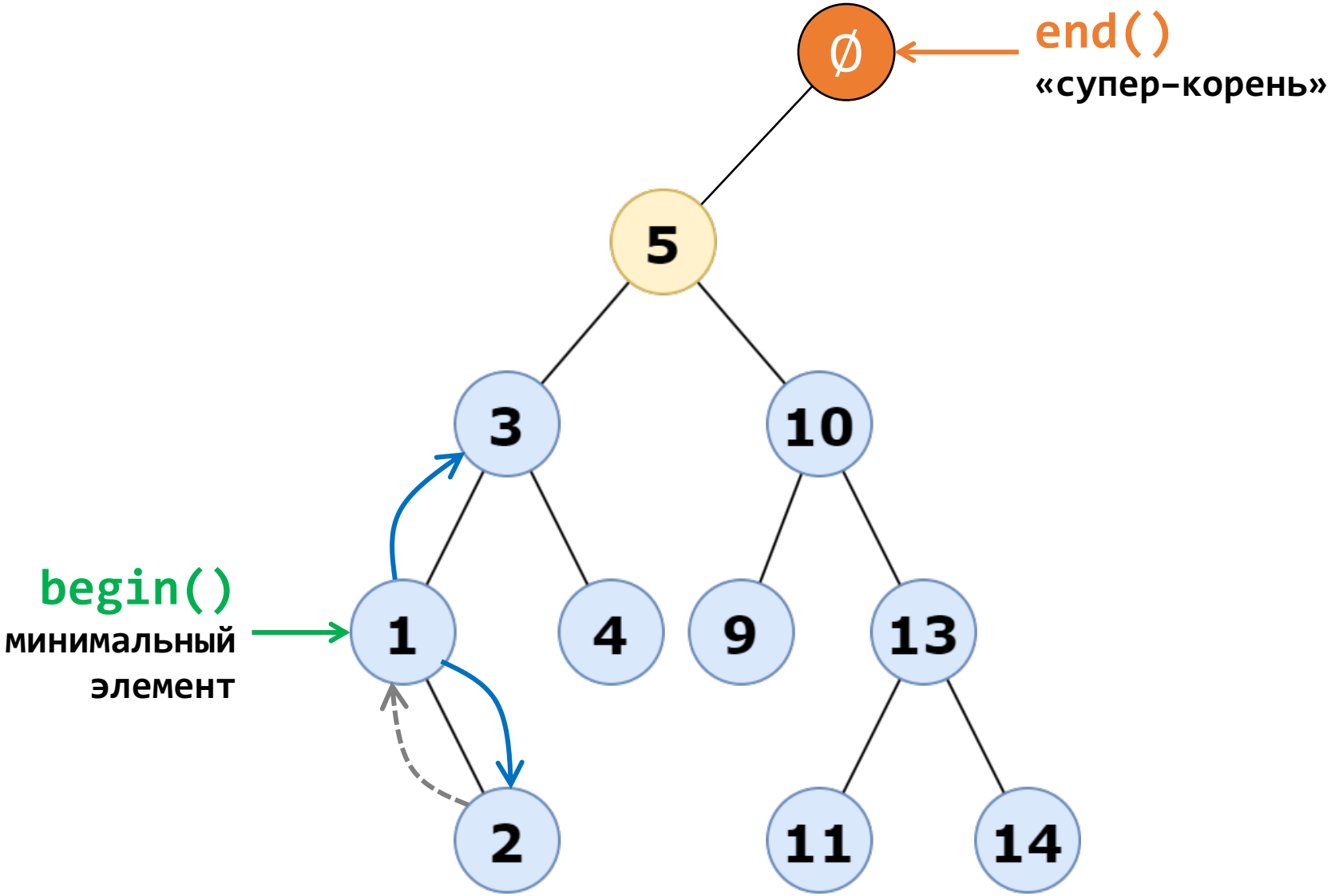
Следующий `treeSuccessor`

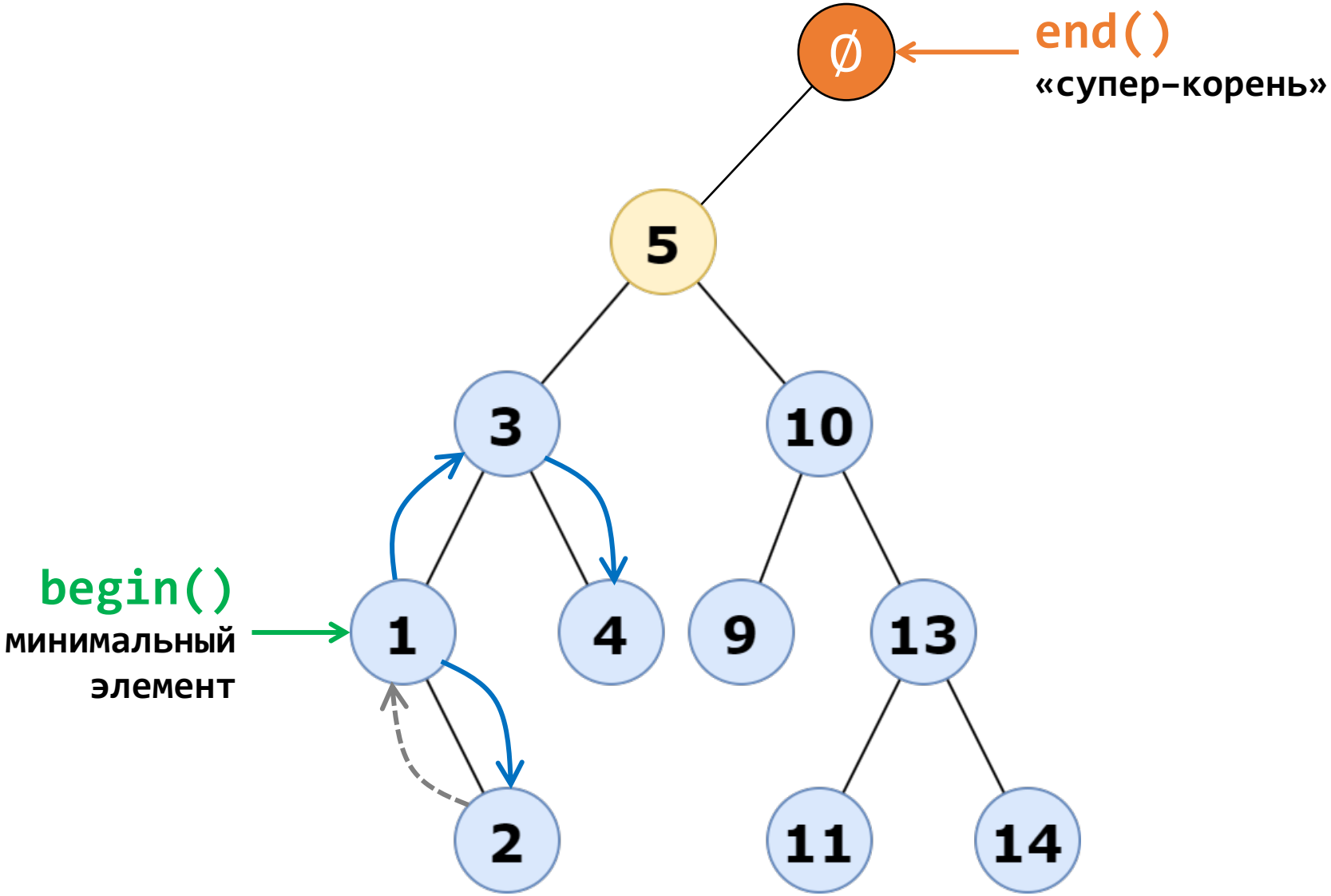
Итак...

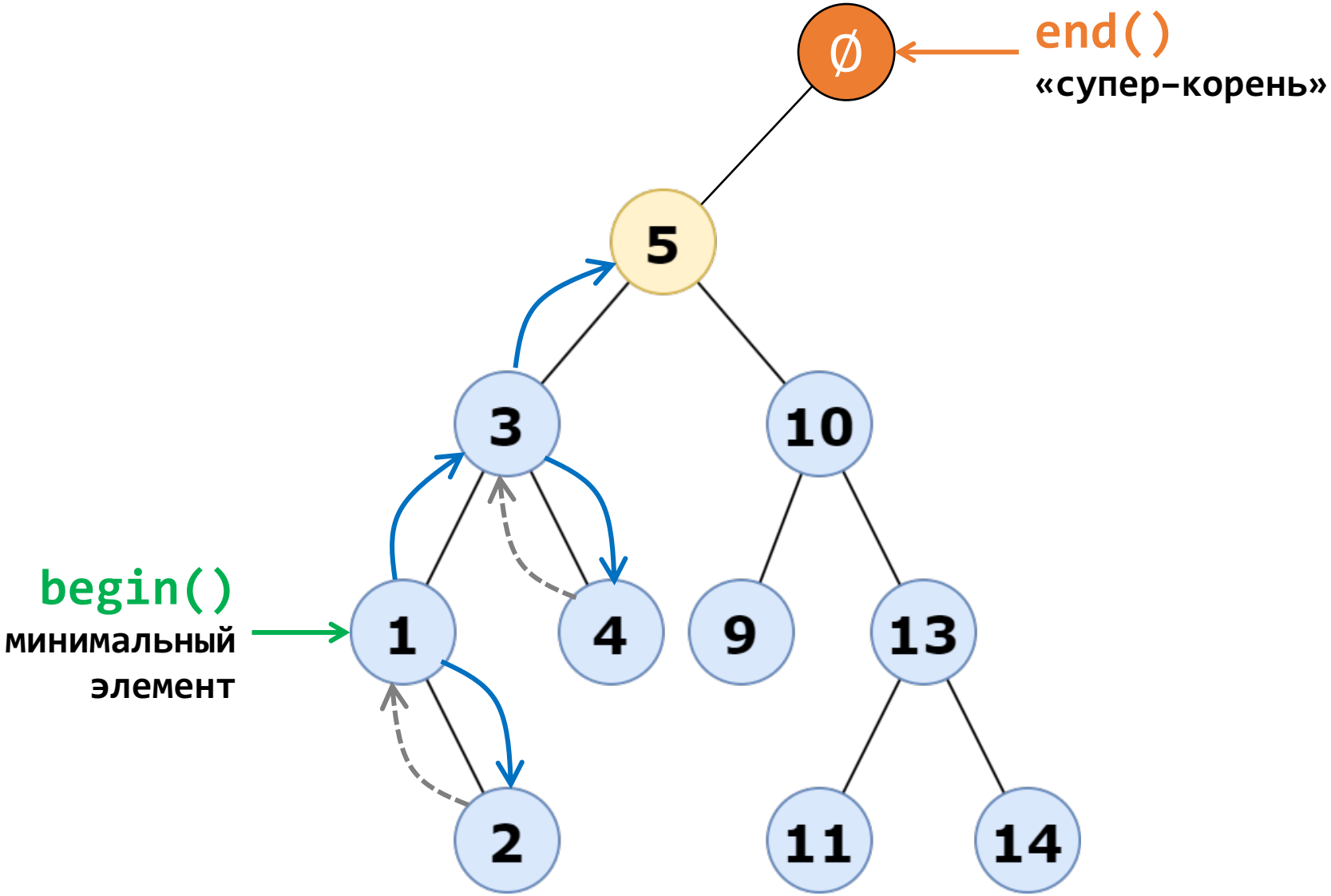
1. Определяющим фактором в поиске предыдущего значения будет наличие/отсутствие **правого поддерева** у выбранного нами узла
2. Отдельного внимания заслуживает максимальный элемент

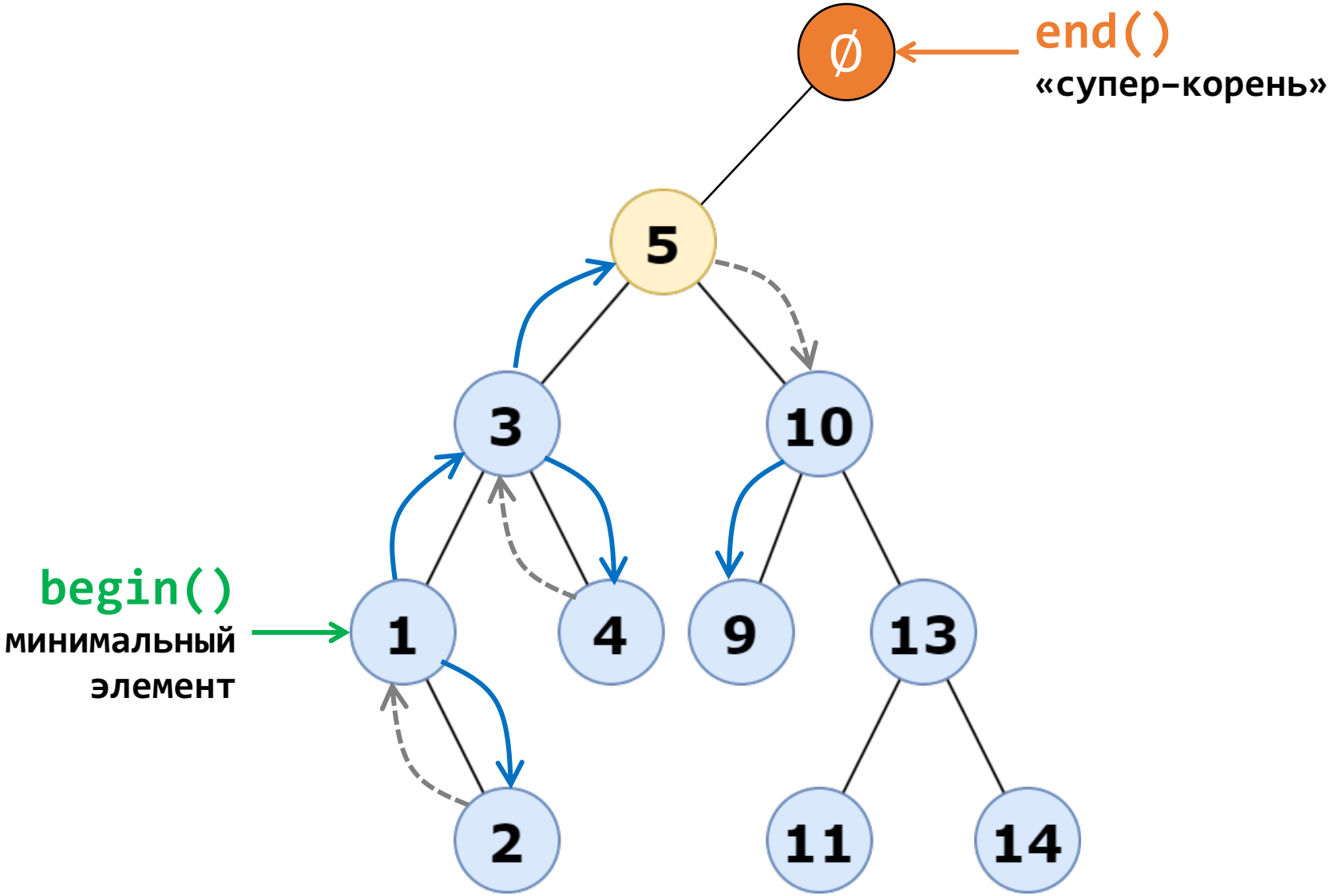

end()
«супер-корень»
∅
5
3
10
begin()
минимальный
элемент
1
4
9
13
2
11
14

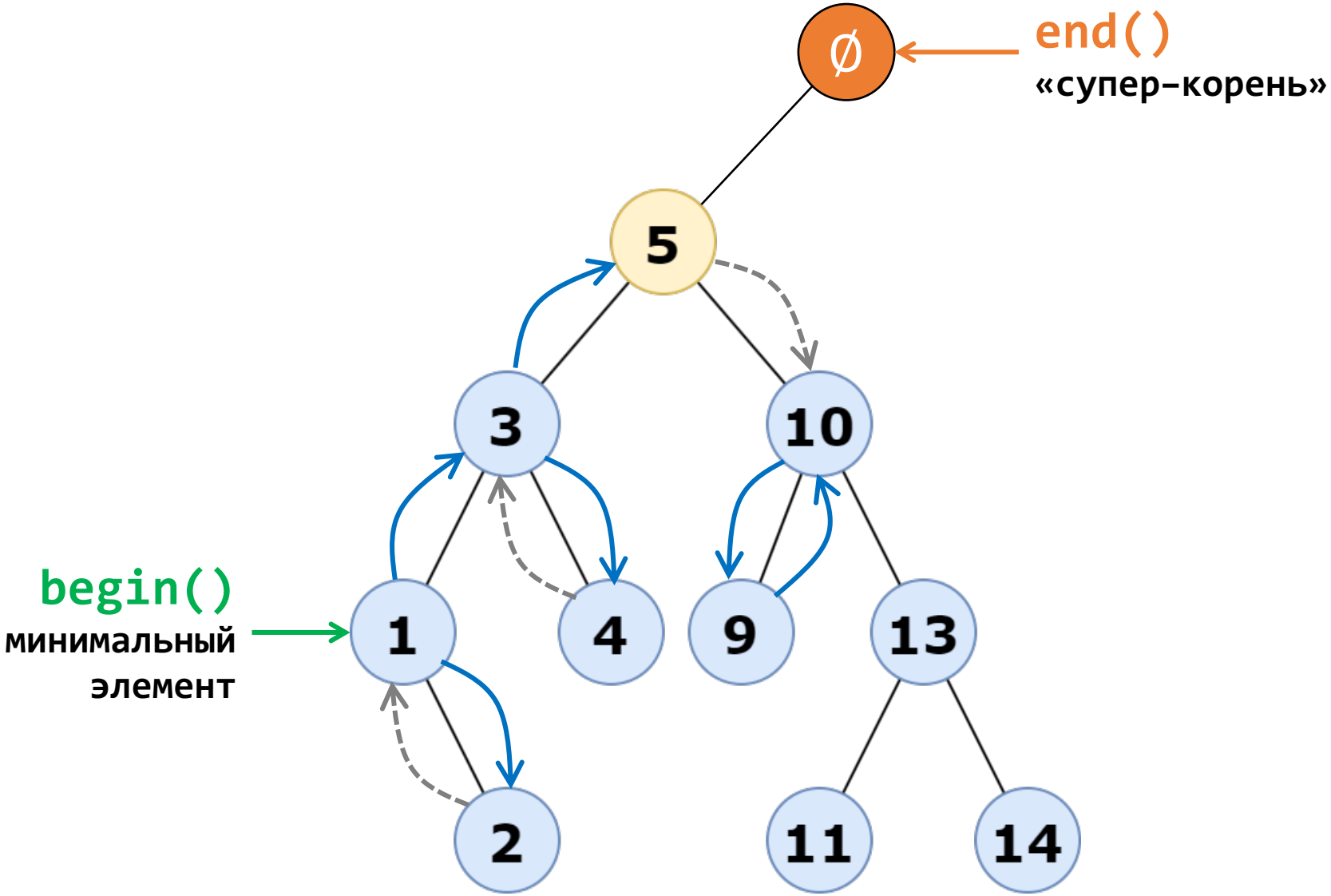

end()
«супер-корень»
∅
5
3
10
begin()
минимальный
элемент
1
4
9
13
2
11
14

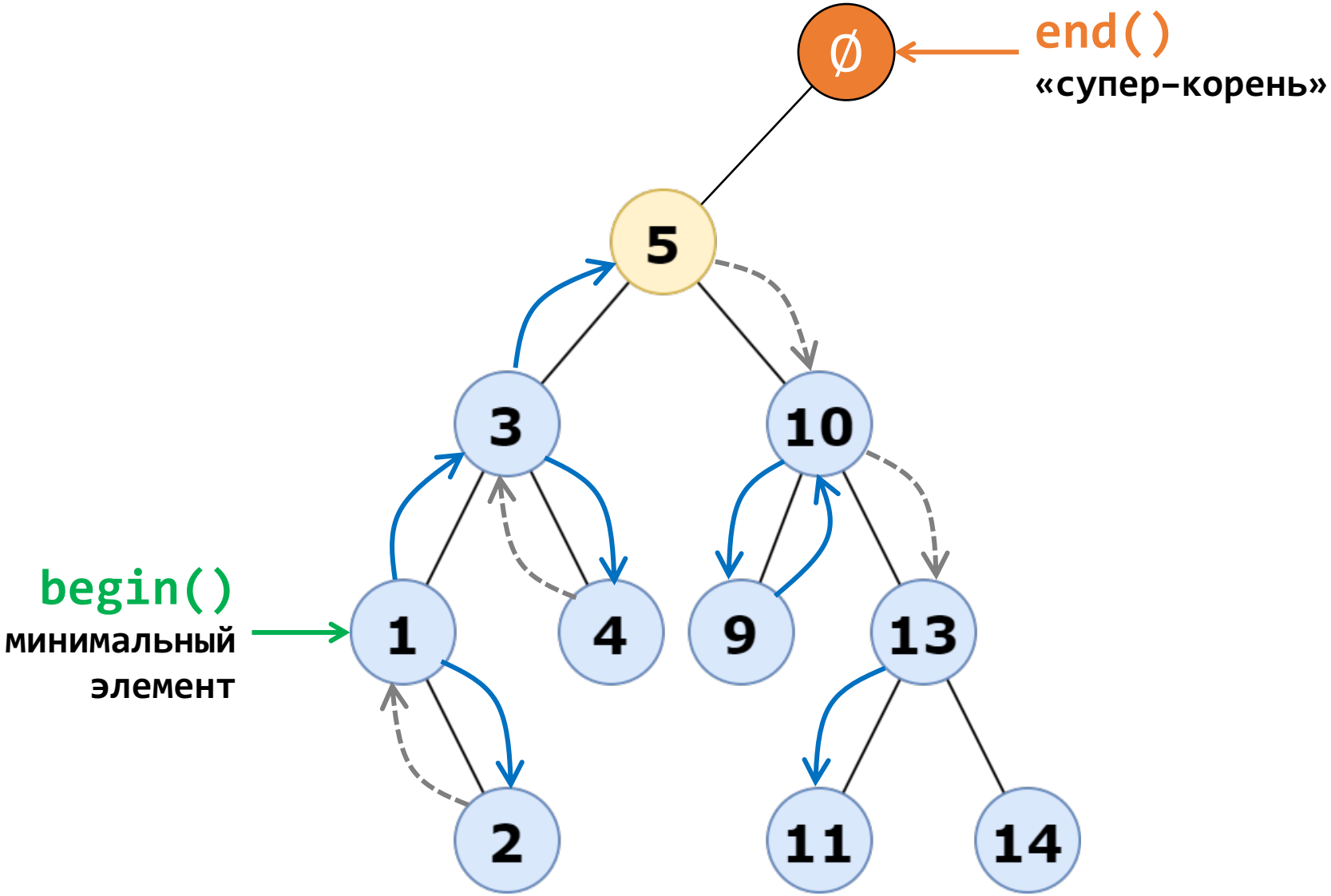

end()
«супер-корень»
∅
5
3
10
begin()
минимальный
элемент
1
4
9
13
2
11
14

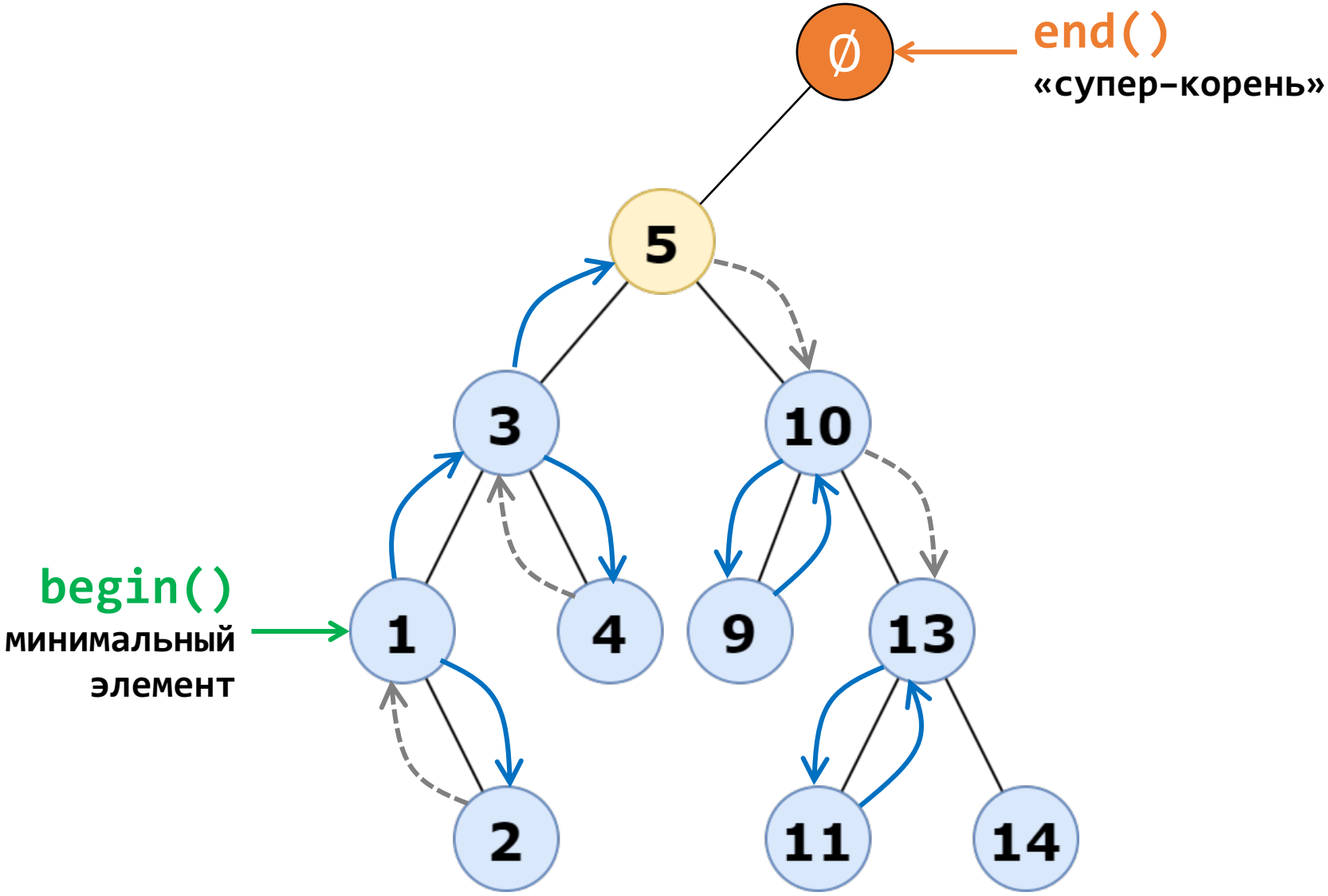

end()
«супер-корень»
∅
5
3
10
begin()
минимальный
элемент
1
4
9
13
2
11
14

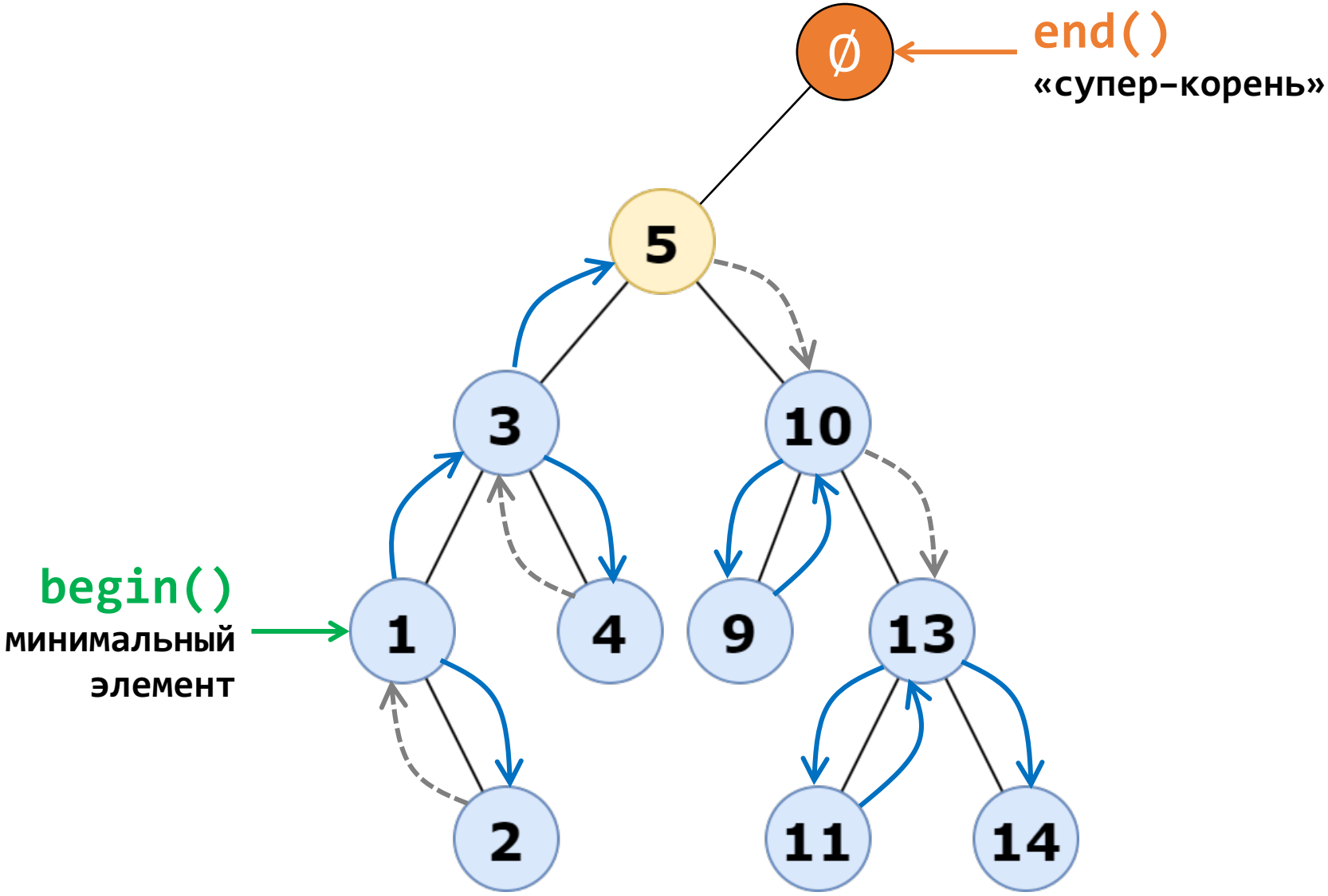

end()
«супер-корень»
∅
5
3
10
begin()
минимальный
элемент
1
4
9
13
2
11
14

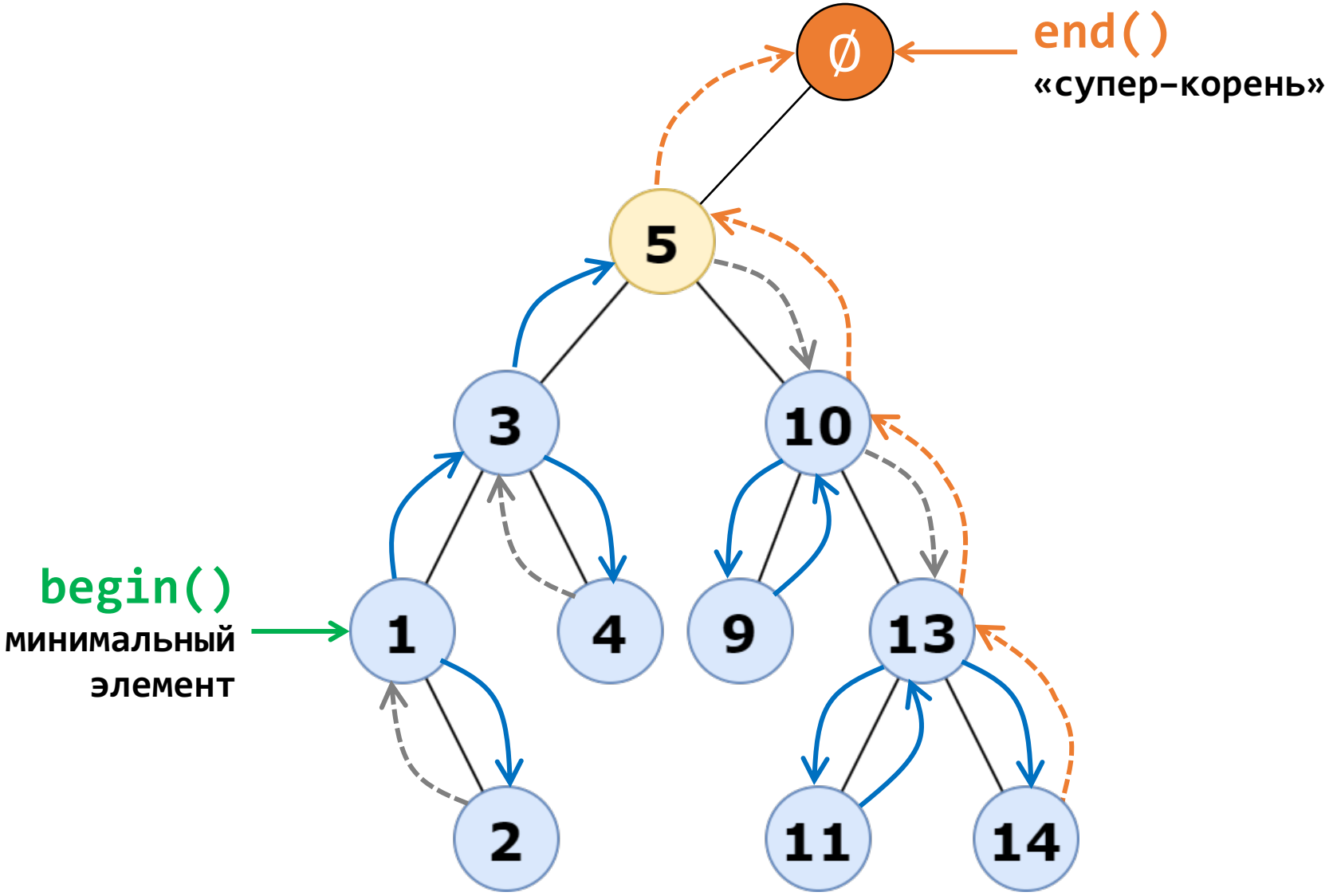

end()
«супер-корень»
∅
5
3
10
begin()
минимальный
элемент
1
4
9
13
2
11
14


end()
«супер-корень»
∅
5
3
10
begin()
минимальный
элемент
1
4
9
13
2
11
14


end()
«супер-корень»
∅
5
3
10
begin()
минимальный
элемент
1
4
9
13
2
11
14


end()
«супер-корень»
∅
5
3
10
begin()
минимальный
элемент
1
4
9
13
2
11
14


end()
«супер-корень»
∅
5
3
10
begin()
минимальный
элемент
1
4
9
13
2
11
14


end()
«супер-корень»
∅
5
3
10
begin()
минимальный
элемент
1
4
9
13
2
11
14

К ~~барьеру~~ реализации!