Алгоритмы и структуры данных

Сложность алгоритма Дейкстры, Фибоначчиева пирамида

Алгоритм Дейкстры поиска кратчайших путей

Оценка сложности:

1. Обрабатываем каждую вершину 1 раз
2. Для каждой вершины перебираем ребра
3. На каждом шаге алгоритма ищем минимум

$$O(|V| * log(|V|) + |E| * log(|V|))$$

Для случая пирамиды - T = O(log(|V|))

Но нужно еще и пометки обновить, а это как раз decrease_priority(...) - каждая тоже за O(log(|V|))

Но можем ли мы еще лучше?

Биномиальная пирамида

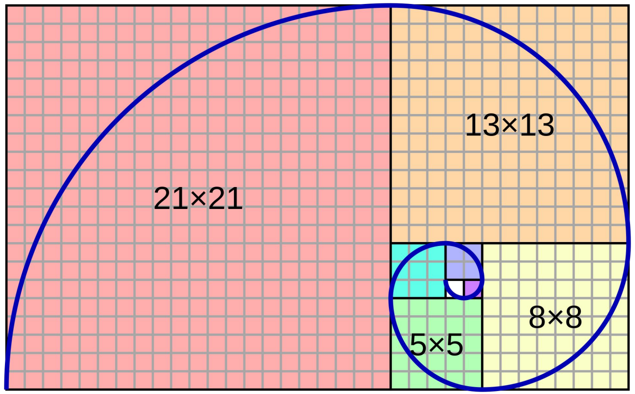
Множество значений: пары <priority: int, value: T>

Операции:

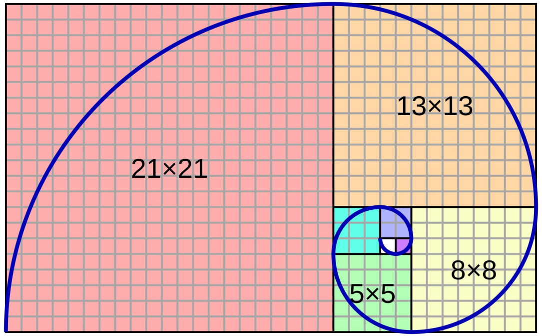
1. insert(value) -> O(logN)
2. peek_min() -> O(logN) 🥴
3. extract_min() -> O(logN)
4. decrease_key(s, k) -> O(logN)
5. merge(H1, H2) -> O(logN) 🥰
6. delete(s) -> O(logN)

Как реализовать?
~~Бинарная куча!~~
Биномиальная!

Фибоначчиева пирамида



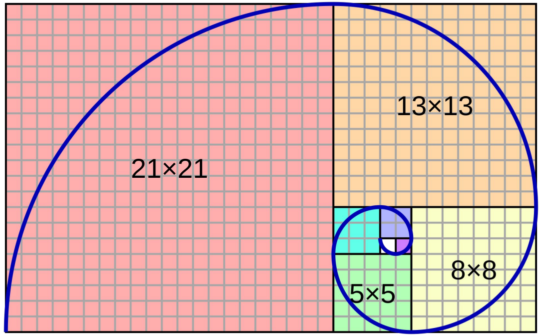
Фибоначчиева пирамида



Операции:

1. insert(value)
2. peek_min()
3. extract_min()
4. decrease_key(s, k)

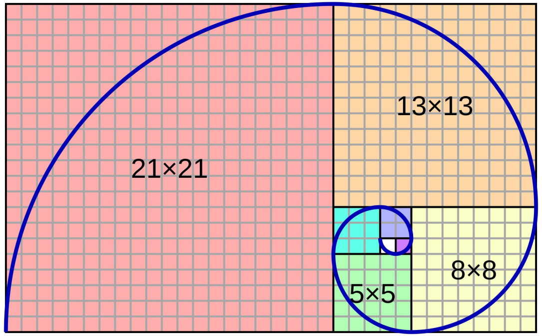
Фибоначчиева пирамида



Операции:

1. `insert(value)`
2. `peek_min()`
3. `extract_min()`
4. `decrease_key(s, k)`
5. `merge(f1, f2)`
6. `delete(s)`

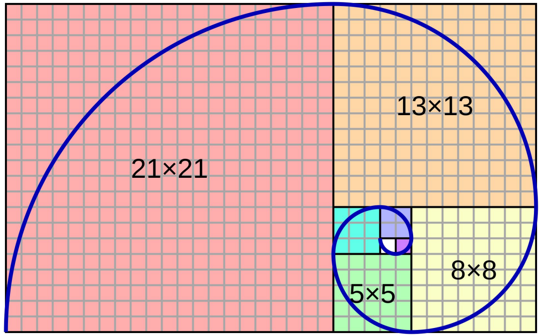
Фибоначчиева пирамида



Операции:

```
1.  insert(value)          -> O(1)
2.  peek_min()             -> O(1)
3.  extract_min()
4.  decrease_key(s, k)
5.  merge(f1, f2)          -> O(1)
6.  delete(s)
```

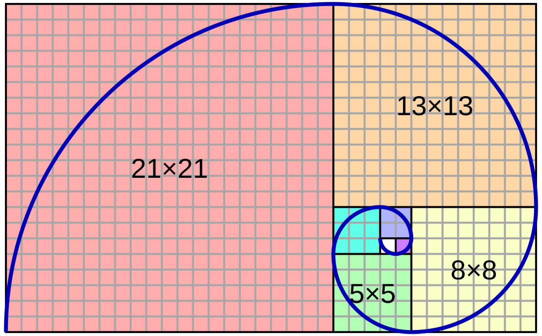
Фибоначчиева пирамида



Операции:

```
1.  insert(value)         -> O(1)
2.  peek_min()            -> O(1)
3.  extract_min()
4.  decrease_key(s, k)  -> O*(1)     (амортизированное)
5.  merge(f1, f2)         -> O(1)
6.  delete(s)
```

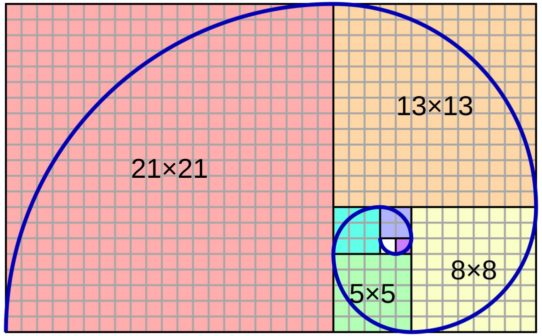
Фибоначчиева пирамида



Операции:

```
1.  insert(value)        -> O(1)
2.  peek_min()           -> O(1)
3.  extract_min()        -> O*(logN) (амортизированное)
4.  decrease_key(s, k)   -> O*(1)    (амортизированное)
5.  merge(f1, f2)        -> O(1)
6.  delete(s)            -> O*(logN) (амортизированное)
```

Фибоначчиева пирамида



Операции:

```
1.  insert(value)       -> O(1)
2.  peek_min()          -> O(1)
3.  extract_min()       -> O*(logN) -> O(N) (в худшем)
4.  decrease_key(s, k)  -> O*(1)    -> O(N) (в худшем)
5.  merge(f1, f2)       -> O(1)
6.  delete(s)           -> O*(logN) -> O(N) (в худшем)
```

Алгоритм Дейкстры поиска кратчайших путей

Оценка сложности:

1. Обрабатываем каждую вершину 1 раз
2. Для каждой вершины перебираем ребра
3. На каждом шаге алгоритма ищем минимум

$$O(|V| * log(|V|) + |E| * log(|V|))$$

Но можем ли мы еще лучше?

Для случая пирамиды - T = O(log(|V|))

Но нужно еще и пометки обновить, а это как раз decrease_priority(...) - каждая тоже за O(log(|V|))

Алгоритм Дейкстры поиска кратчайших путей

Оценка сложности:

1. Обрабатываем каждую вершину 1 раз
2. Для каждой вершины перебираем ребра
3. На каждом шаге алгоритма ищем минимум

$$O(|V| * log(|V|) + |E| * log(|V|))$$

Для Фибоначчиевой кучи обновление пометки - O*(1)

Алгоритм Дейкстры поиска кратчайших путей

Оценка сложности:

1. Обрабатываем каждую вершину 1 раз
2. Для каждой вершины перебираем ребра
3. На каждом шаге алгоритма ищем минимум

$$O^*(|V| * log(|V|) + |E|)$$

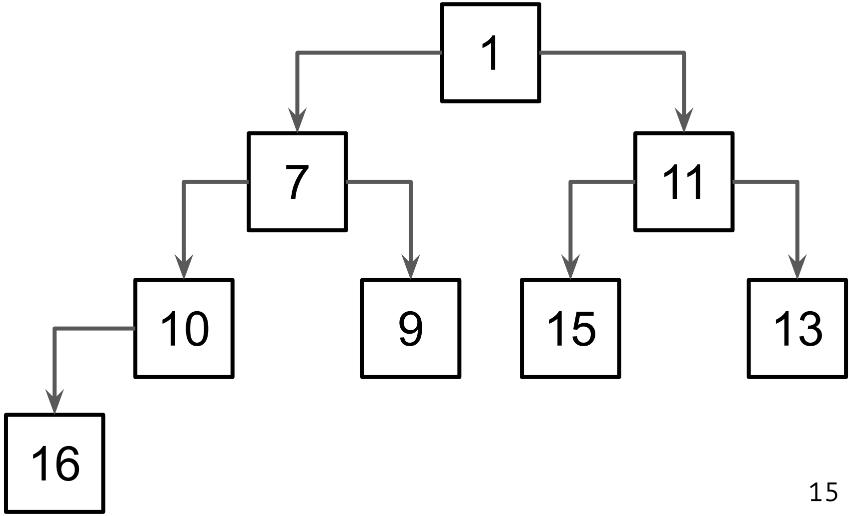
Для Фибоначчиевой кучи обновление пометки - O*(1)

Фибоначчиева пирамида

Определение: будем говорить, что дерево удовлетворяет свойству неубывающей пирамиды, если значение каждого узла меньше либо равно значений всех своих детей

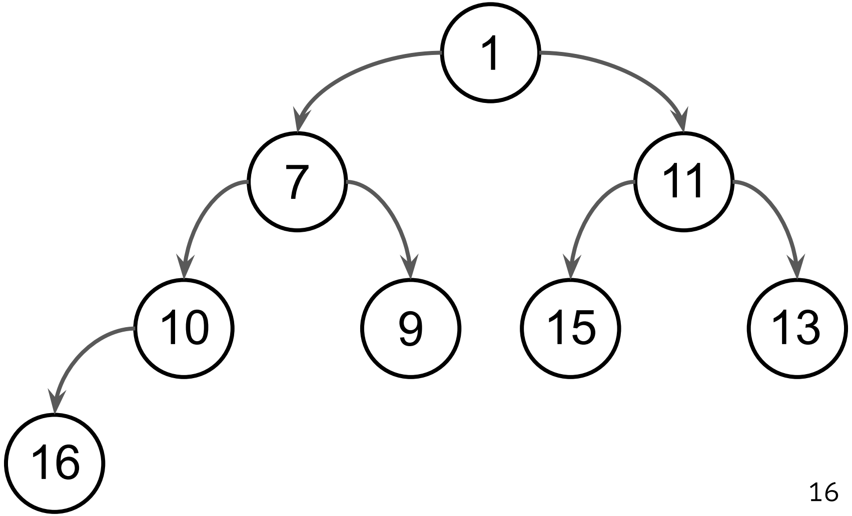
Фибоначчиева пирамида

Определение: будем говорить, что дерево удовлетворяет свойству неубывающей пирамиды, если значение каждого узла меньше либо равно значений всех своих детей



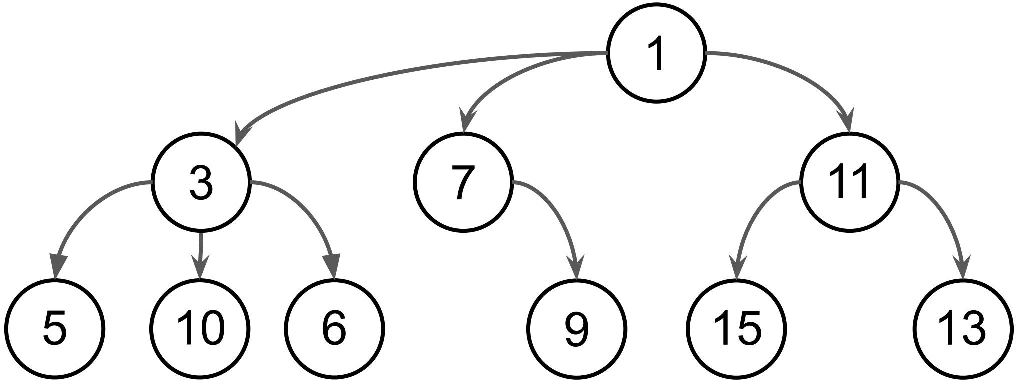
Фибоначчиева пирамида

Определение: будем говорить, что дерево удовлетворяет свойству неубывающей пирамиды, если значение каждого узла меньше либо равно значений всех своих детей



Фибоначчиева пирамида

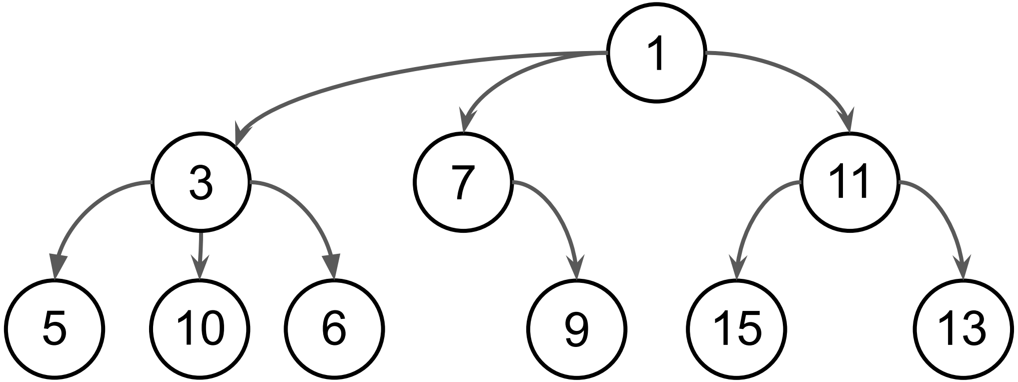
Определение: будем говорить, что дерево удовлетворяет свойству неубывающей пирамиды, если значение каждого узла меньше либо равно значений всех своих детей



Фибоначчиева пирамида

Определение: будем говорить, что дерево удовлетворяет свойству неубывающей пирамиды, если значение каждого узла меньше либо равно значений всех своих детей

Отличие от бинарных пирамид - детей может быть много

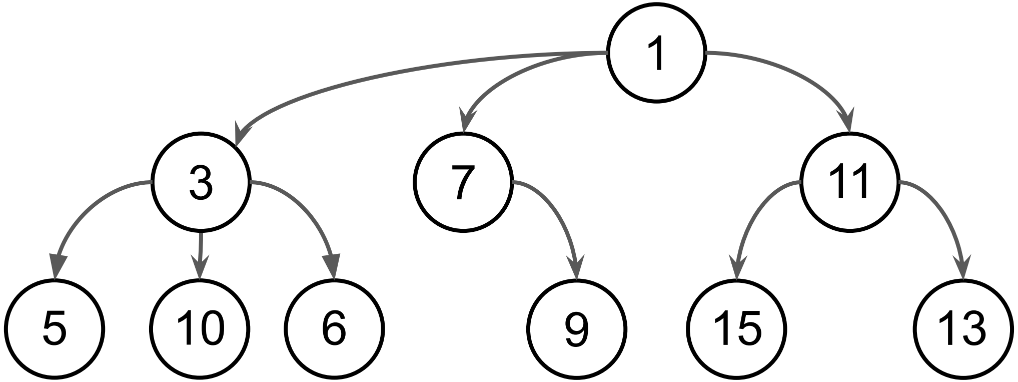


Фибоначчиева пирамида

Определение: будем говорить, что дерево удовлетворяет свойству неубывающей пирамиды, если значение каждого узла меньше либо равно значений всех своих детей

Отличие от бинарных пирамид - детей может быть много =>

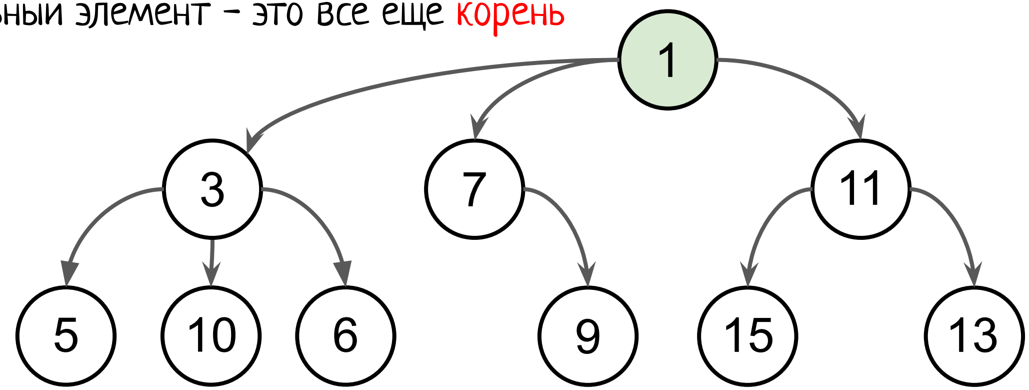
представлением в виде массива уже не воспользуешься



Фибоначчиева пирамида

Определение: будем говорить, что дерево удовлетворяет свойству неубывающей пирамиды, если значение каждого узла меньше либо равно значений всех своих детей

Минимальный элемент - это все еще корень



Отличие от бинарных пирамид - детей может быть много =>

представлением в виде массива уже не воспользуешься

Фибоначчиева пирамида

Определение: будем говорить, что дерево удовлетворяет свойству неубывающей пирамиды, если значение каждого узла меньше либо равно значений всех своих детей

Определение: Фибоначчиевой кучей назовем набор деревьев удовлетворяющих свойству неубывающей пирамиды.

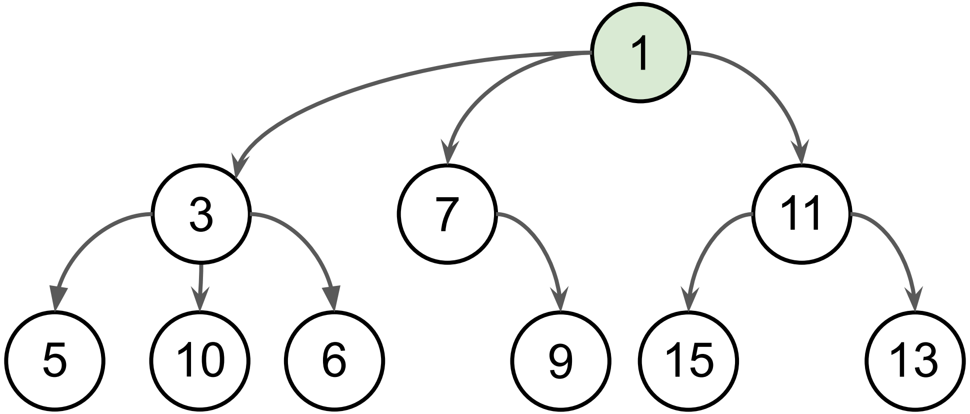
Фибоначчиева пирамида

Определение: будем говорить, что дерево удовлетворяет свойству неубывающей пирамиды, если значение каждого узла меньше либо равно значений всех своих детей

Определение: Фибоначчиевой кучей назовем набор деревьев удовлетворяющих свойству неубывающей пирамиды.

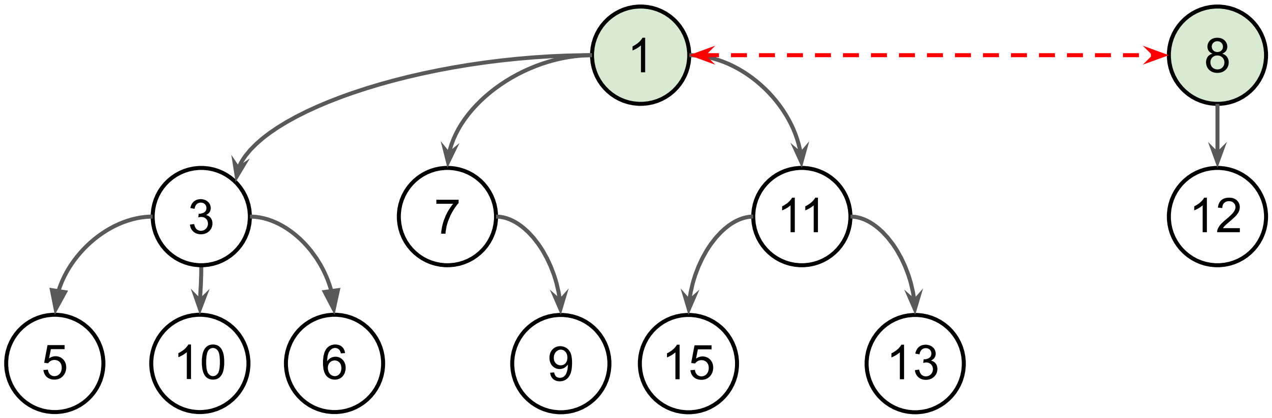
Корни этих деревьев хранятся в двусвязном списке.

Фибоначчиева пирамида



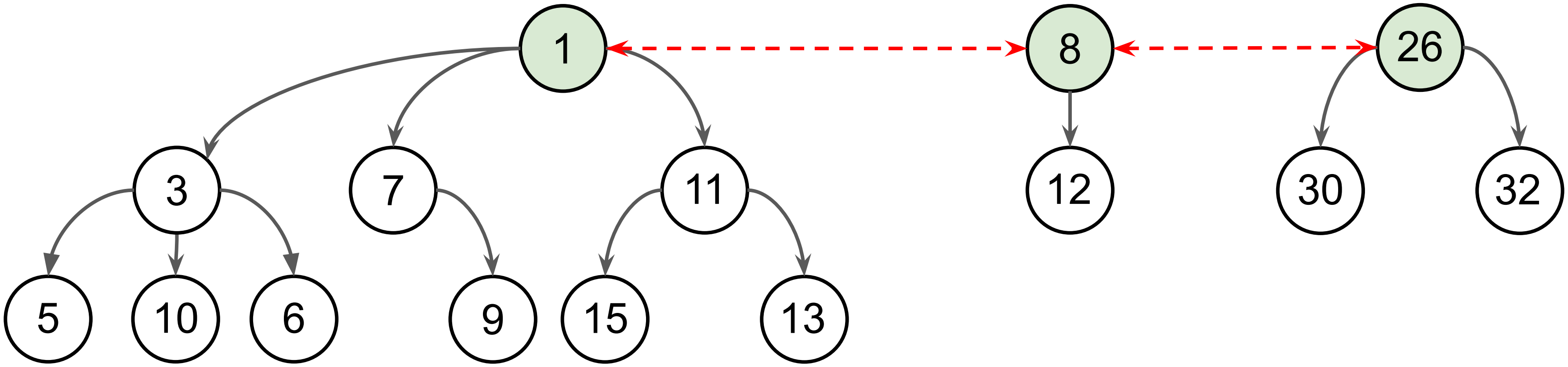
Это Фибоначчиева пирамида

Фибоначчиева пирамида



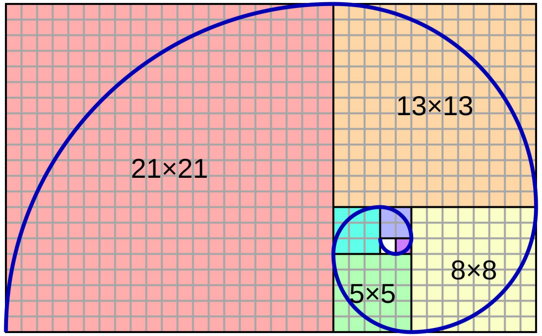
Это Фибоначчиева пирамида

Фибоначчиева пирамида



Это Фибоначчиева пирамида

Фибоначчиева пирамида

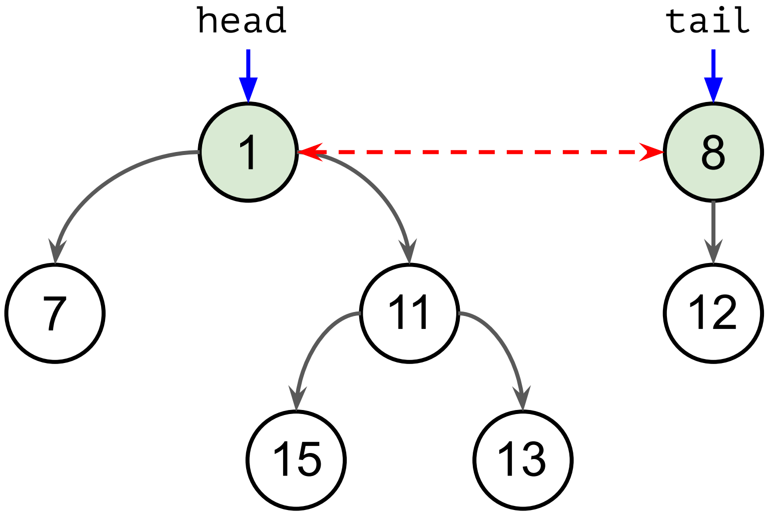


Операции:

1. insert(value)
2. peek_min()
3. extract_min()
4. decrease_key(s, k)
5. ? merge(f1, f2)
6. delete(s)

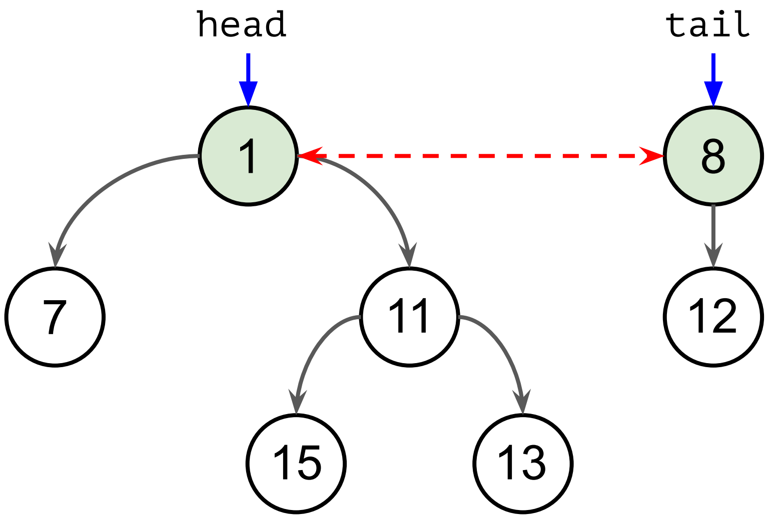
Как слить две Фибоначчиевы пирамиды в одну?

Фибоначчиева пирамида

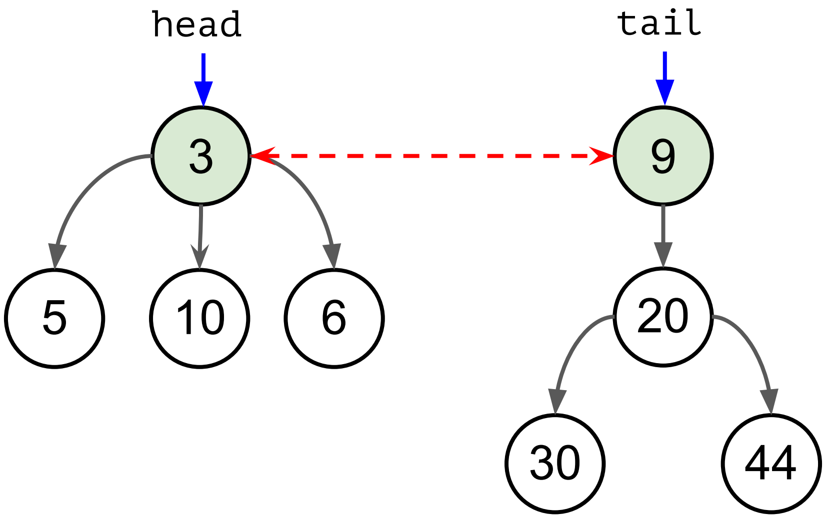


Это Фибоначчиева пирамида

Фибоначчиева пирамида



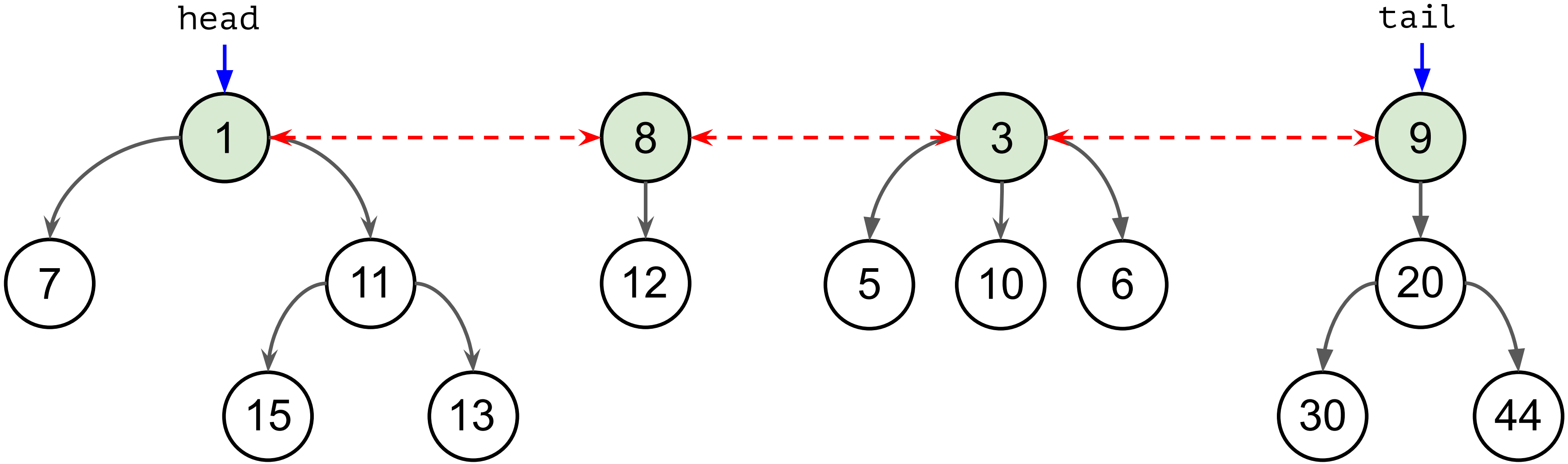
Это Фибоначчиева пирамида



И это Фибоначчиева пирамида

Как их слить в одну?

Фибоначчиева пирамида



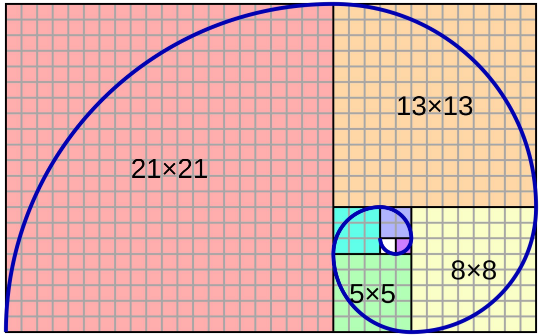
Это Фибоначчиева пирамида

И это Фибоначчиева пирамида

Как их слить в одну?

Объединение списков корней за O(1)

Фибоначчиева пирамида

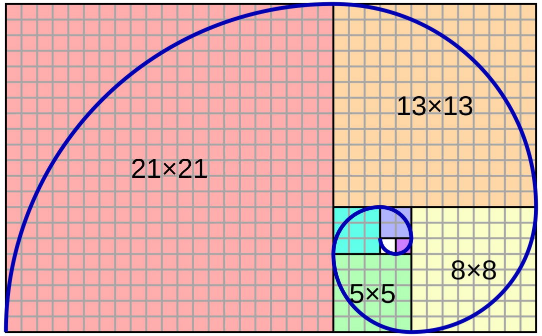


Операции:

```
? 1.  insert(value)
  2.  peek_min()
  3.  extract_min()
  4.  decrease_key(s, k)
✓ 5.  merge(f1, f2)        -> O(1)
  6.  delete(s)
```

Как добавить новый элемент в Фибоначчиеву пирамиду?

Фибоначчиева пирамида

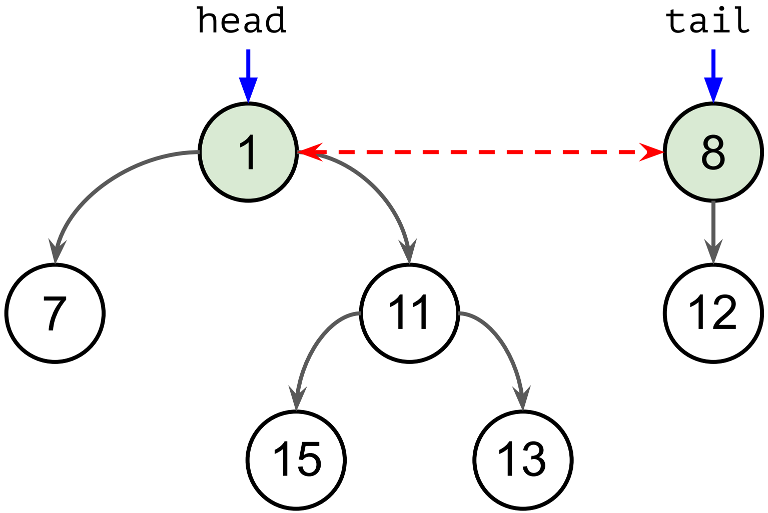


Операции:

? 1. insert(value)
2. peek_min()
3. extract_min()
4. decrease_key(s, k)
✓ 5. merge(f1, f2) -> O(1)
6. delete(s)

Как добавить новый элемент в Фибоначчиеву пирамиду?
Один элемент он ведь сам себе пирамида.

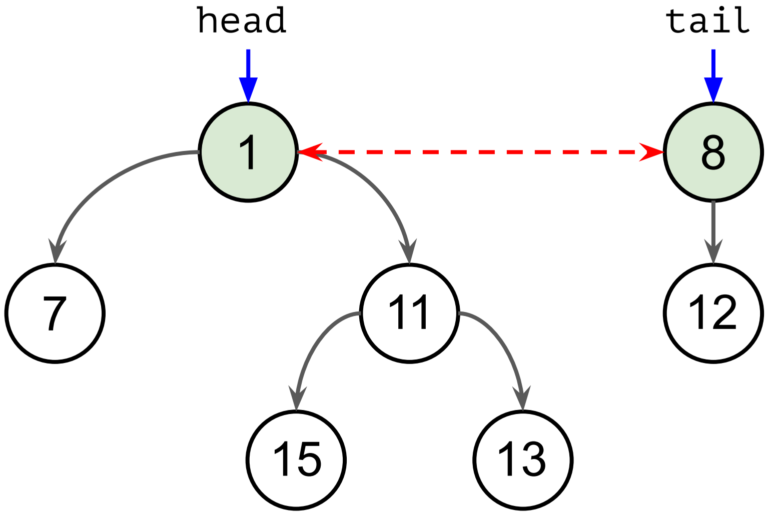
Фибоначчиева пирамида

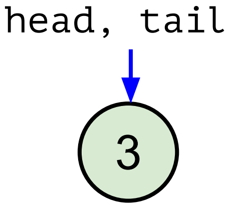


Это Фибоначчиева пирамида

insert(3)?

Фибоначчиева пирамида



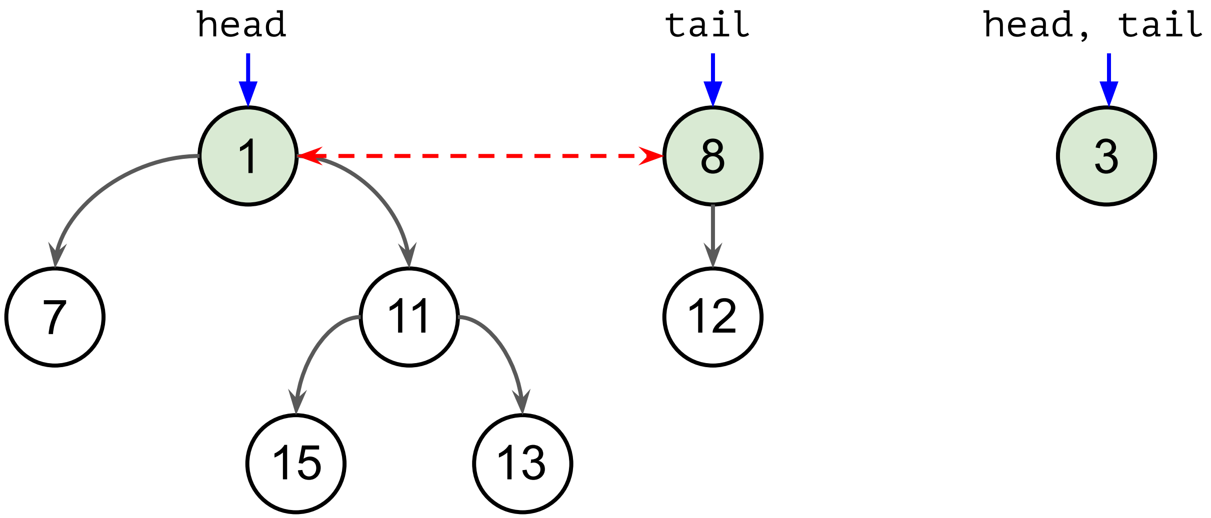


Это Фибоначчиева пирамида

И это Фибоначчиева пирамида

insert(3)?

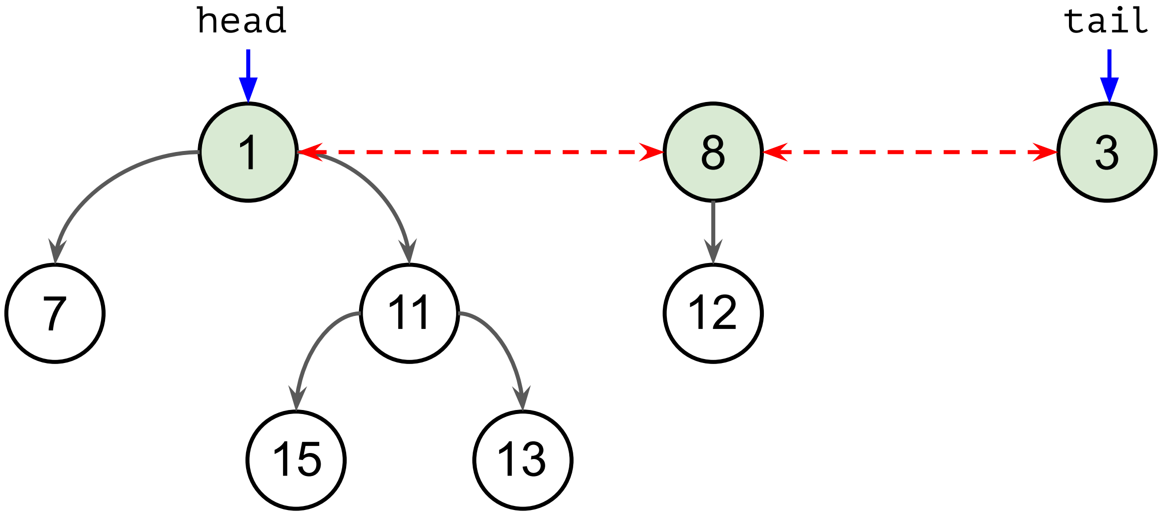
Фибоначчиева пирамида



Это Фибоначчиева пирамида   И это Фибоначчиева пирамида

insert(3): создаем новую пирамиду из 1 элемента => merge

Фибоначчиева пирамида

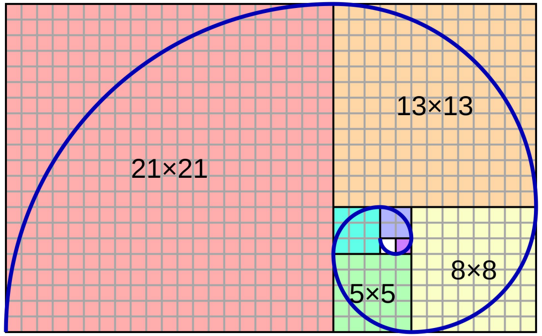


Это Фибоначчиева пирамида   И это Фибоначчиева пирамида

insert(3): создаем новую пирамиду из 1 элемента => merge

Снова операция за O(1)

Фибоначчиева пирамида

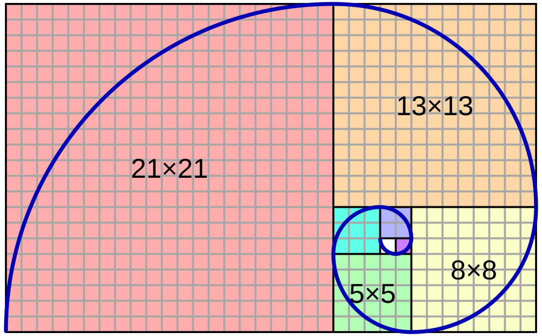


Операции:

```
✓ 1.  insert(value)         -> O(1)
  2.  peek_min()            -> ???
  3.  extract_min()
  4.  decrease_key(s, k)
✓ 5.  merge(f1, f2)         -> O(1)
  6.  delete(s)
```

Как узнать минимальный элемент?

Фибоначчиева пирамида

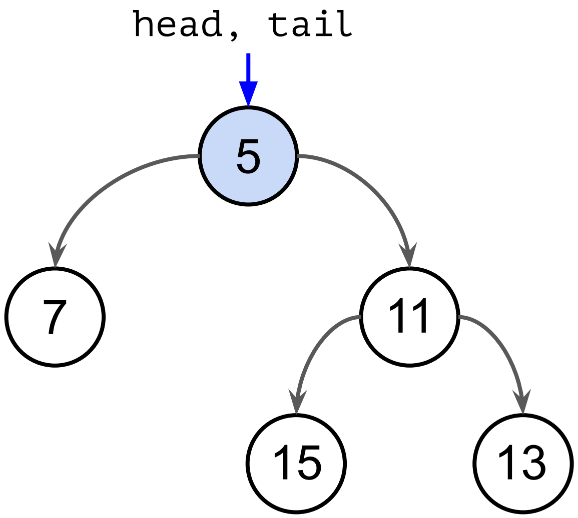


Операции:

```
✓ 1.  insert(value)          -> O(1)
  2.  peek_min()             -> ???
  3.  extract_min()
  4.  decrease_key(s, k)
✓ 5.  merge(f1, f2)          -> O(1)
  6.  delete(s)
```

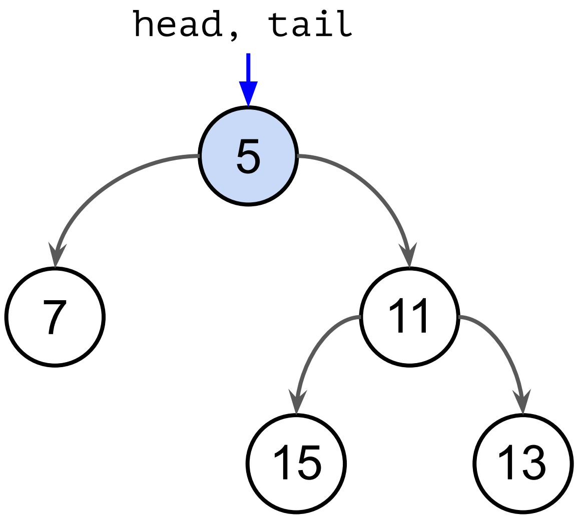
Как узнать минимальный элемент? Он явно где-то среди корней. Давайте хранить указатель на него и обновлять, когда надо.

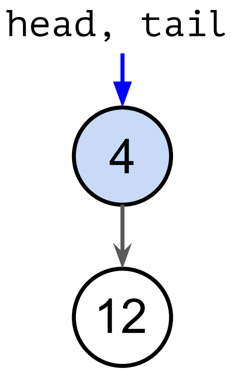
Фибоначчиева пирамида



Это Фибоначчиева пирамида

Фибоначчиева пирамида

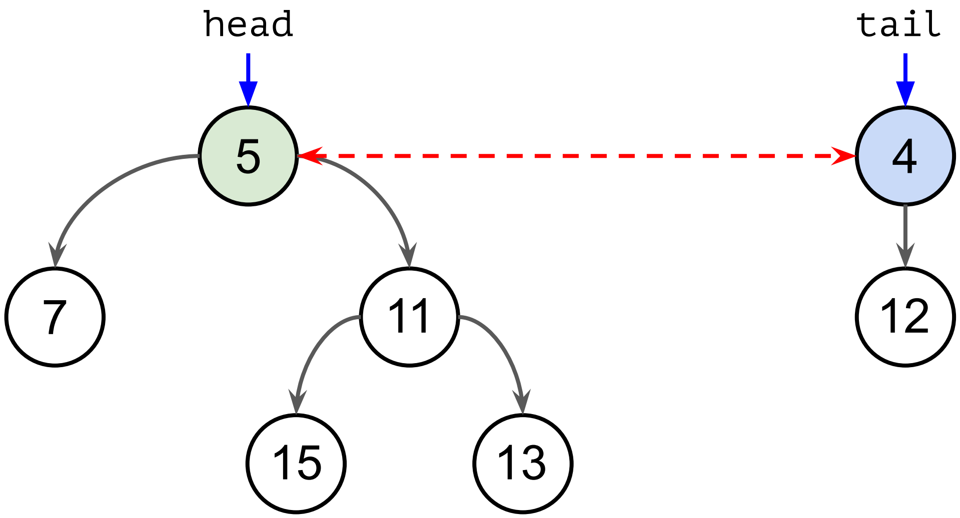




Это Фибоначчиева пирамида

И это Фибоначчиева пирамида

Фибоначчиева пирамида

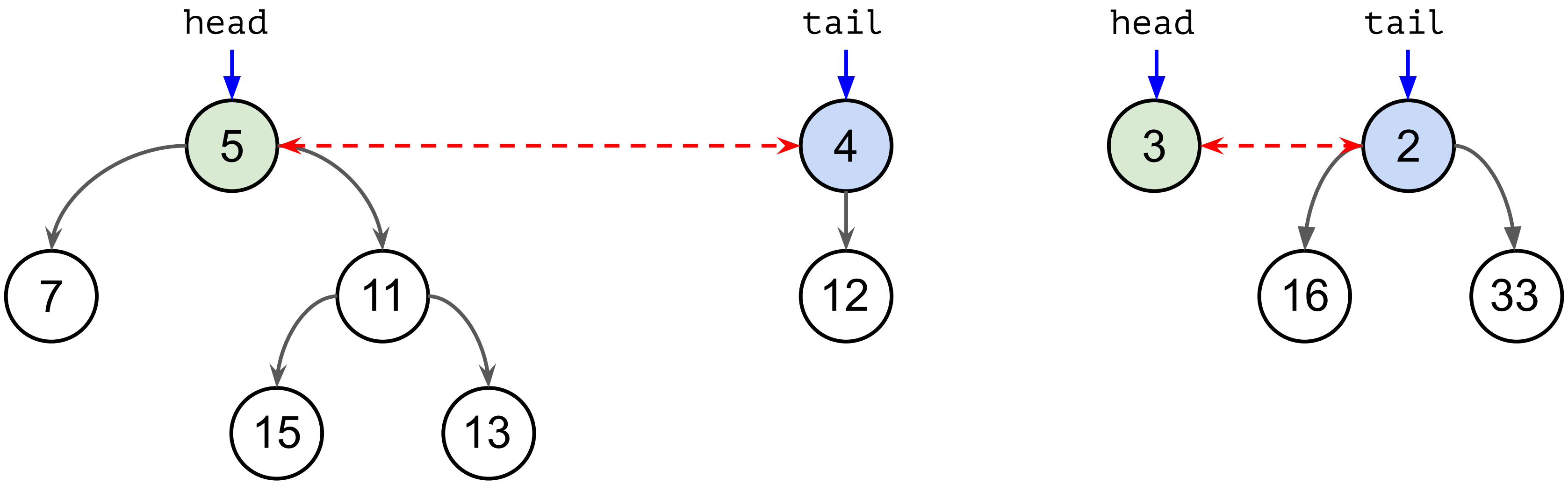


Это Фибоначчиева пирамида

И это Фибоначчиева пирамида

При merge обновим минимальный на минимум из корней

Фибоначчиева пирамида

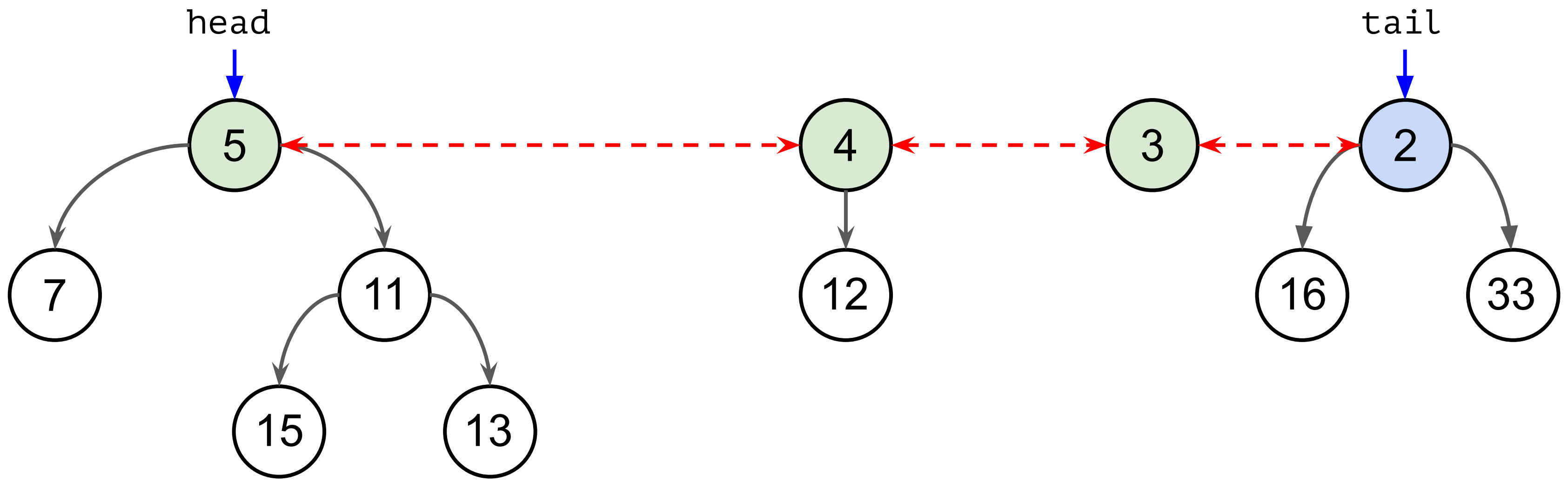


Это Фибоначчиева пирамида

И это Фибоначчиева пирамида

При merge обновим минимальный на минимум минимумов

Фибоначчиева пирамида

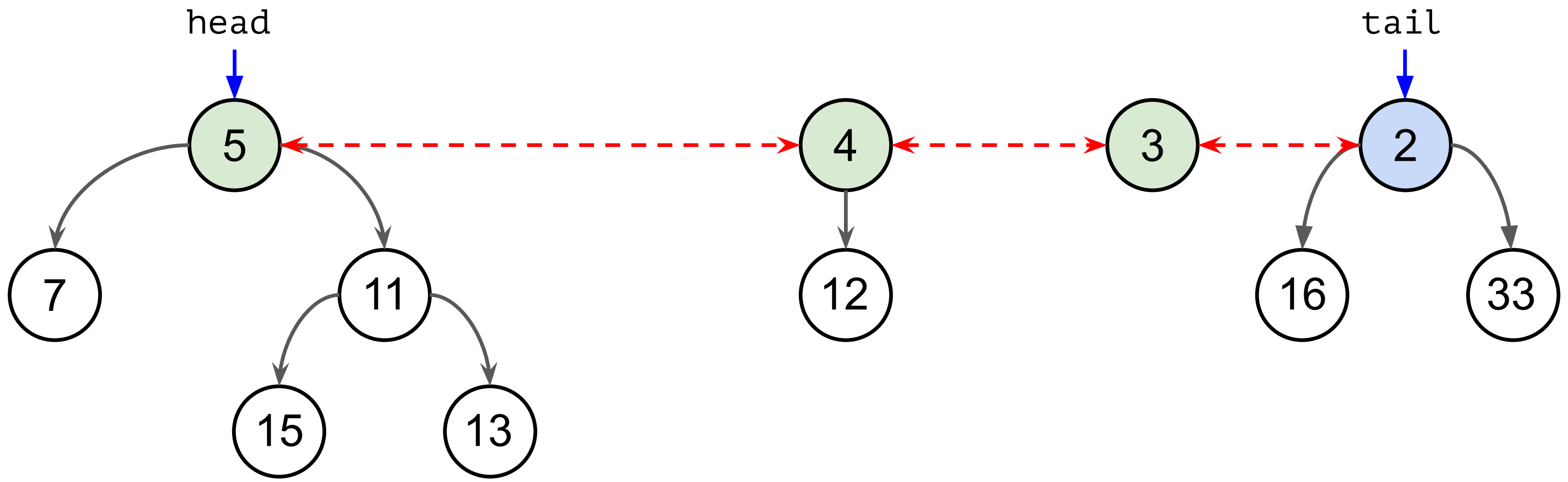


Это Фибоначчиева пирамида

И это Фибоначчиева пирамида

При merge обновим минимальный на минимум минимумов

Фибоначчиева пирамида



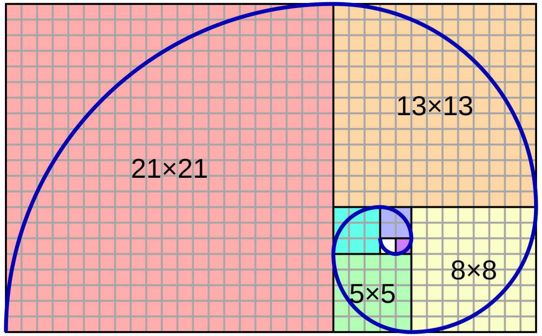
Это Фибоначчиева пирамида

И это Фибоначчиева пирамида

При merge обновим минимальный на минимум минимумов

Тогда узнать минимальный элемент - операция за O(1), просто возьмем синий

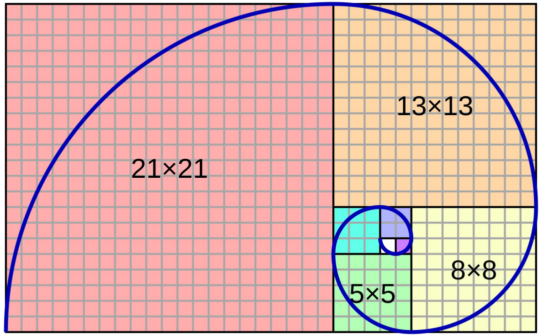
Фибоначчиева пирамида



Операции:

```
✓ 1.  insert(value)         -> O(1)
✓ 2.  peek_min()            -> O(1)
? 3.  extract_min()         -> ???
  4.  decrease_key(s, k)
✓ 5.  merge(f1, f2)         -> O(1)
  6.  delete(s)
```

Фибоначчиева пирамида

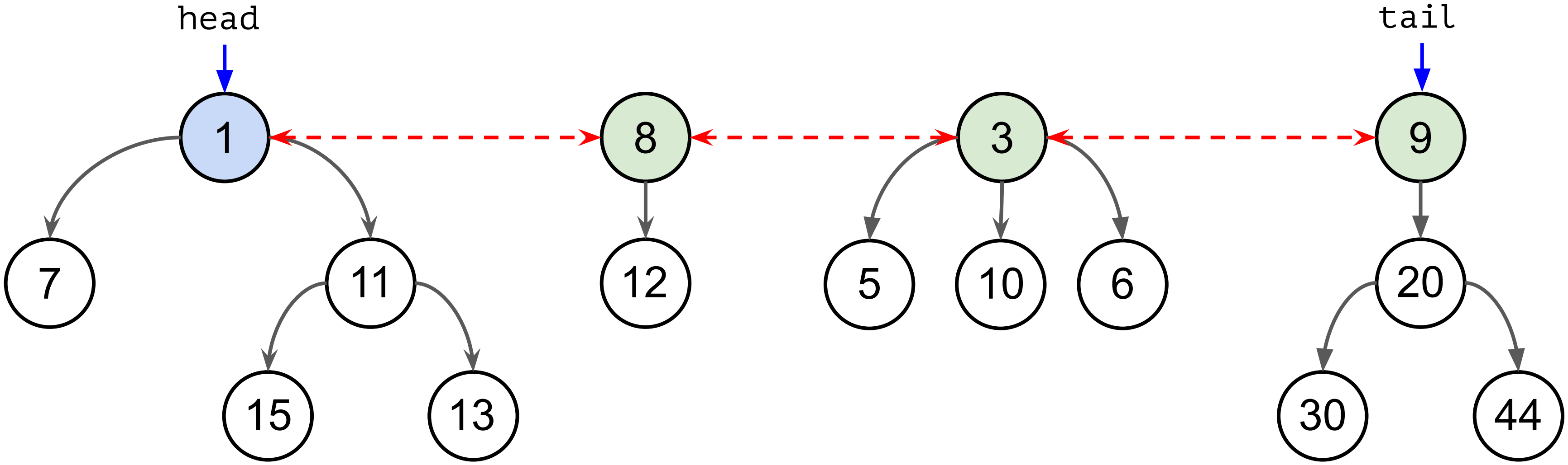


Операции:

```
✓ 1.  insert(value)          -> O(1)
✓ 2.  peek_min()             -> O(1)
? 3.  extract_min()          -> ???
  4.  decrease_key(s, k)
✓ 5.  merge(f1, f2)          -> O(1)
  6.  delete(s)
```

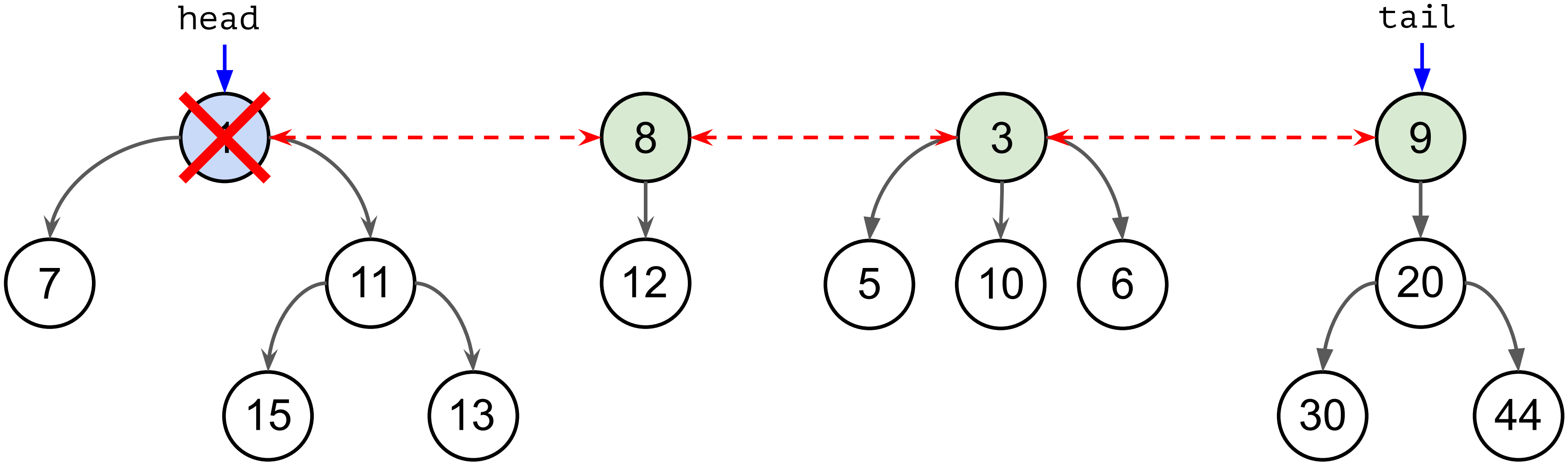
Идея простая: давайте снова пользоваться процедурой merge.

Фибоначчиева пирамида



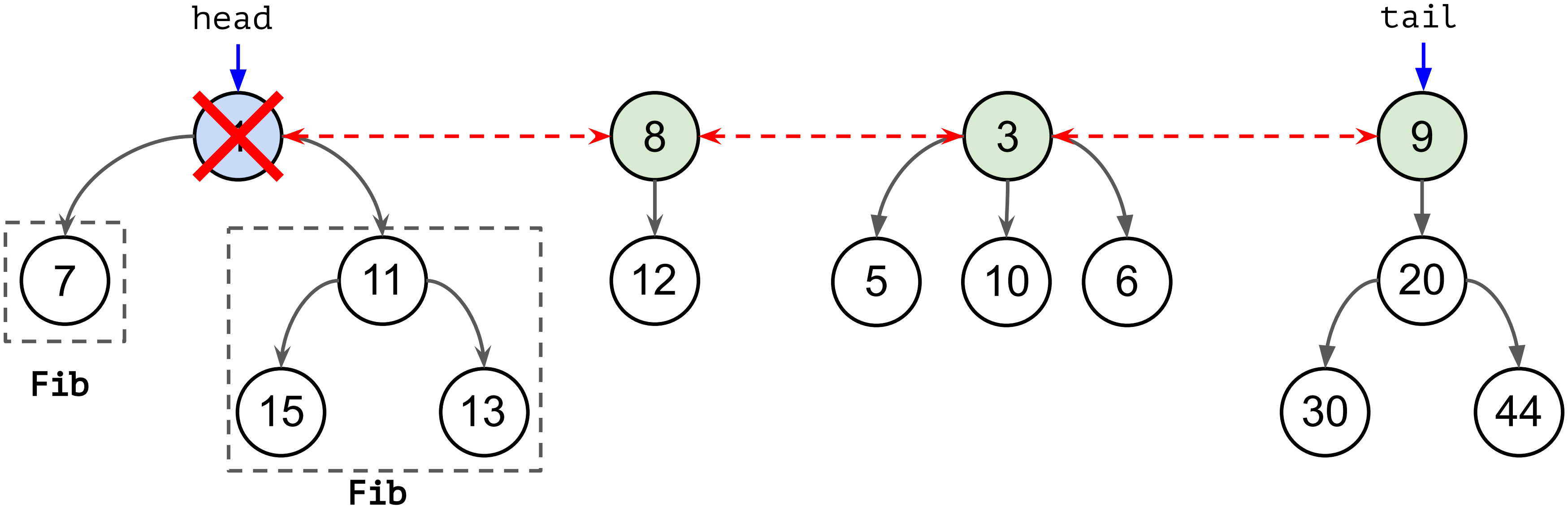
Хочу достать минимальный элемент из пирамиды.

Фибоначчиева пирамида



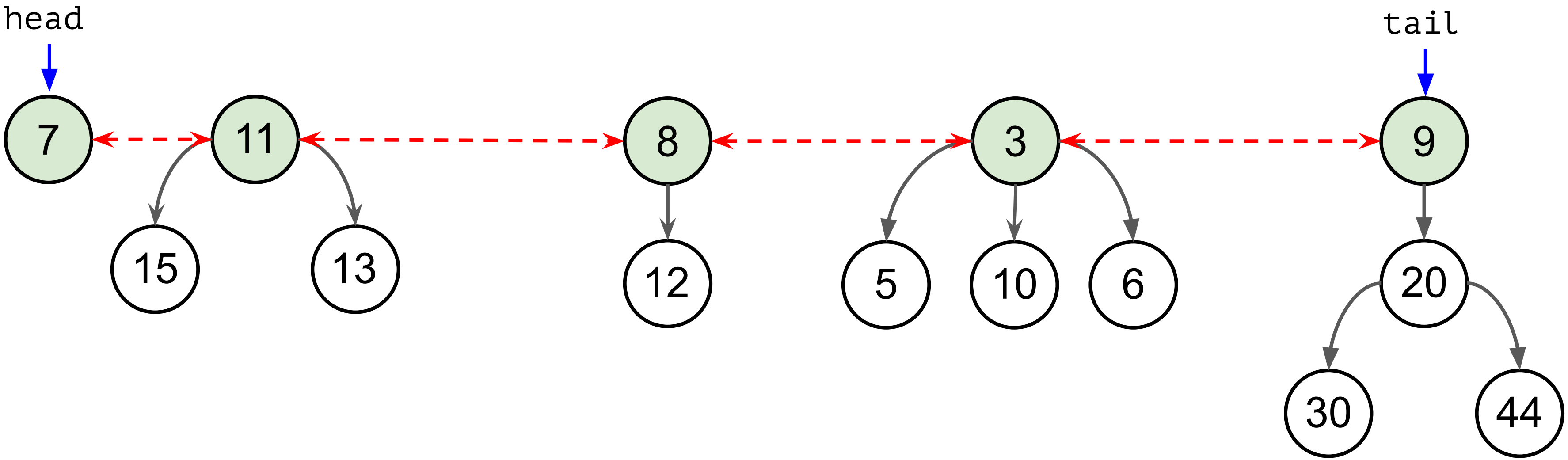
Хочу достать минимальный элемент из пирамиды.

Фибоначчиева пирамида



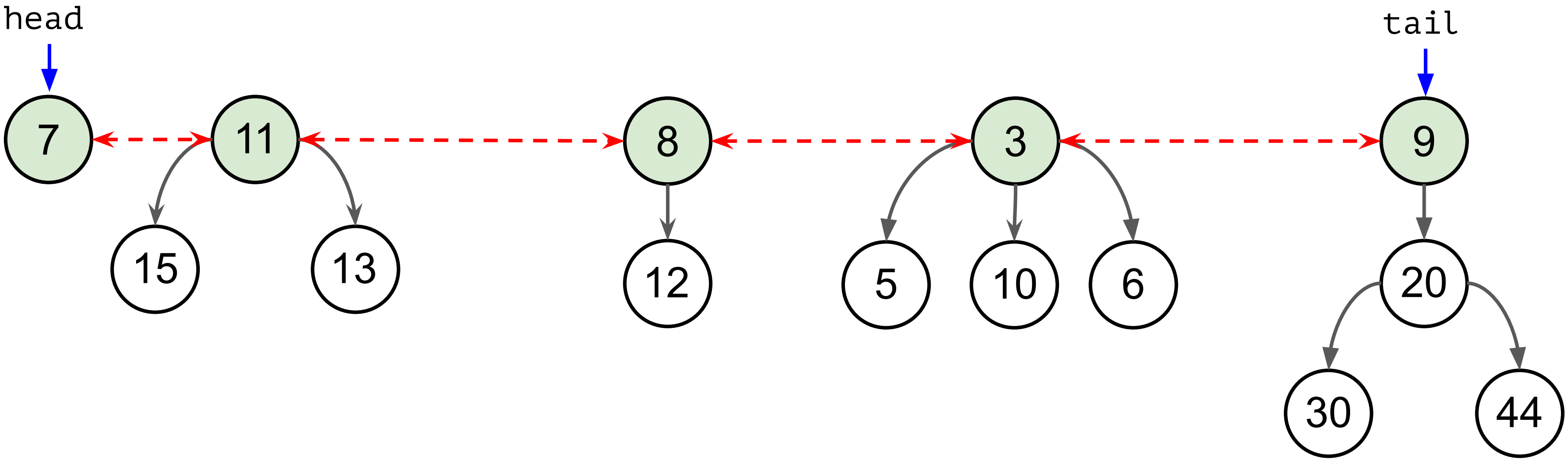
Хочу достать минимальный элемент из пирамиды.
Удаляем корень, разбираемся с детьми.

Фибоначчиева пирамида



Хочу достать минимальный элемент из пирамиды.
Удаляем корень, разбираемся с детьми (два раза вызывая merge).

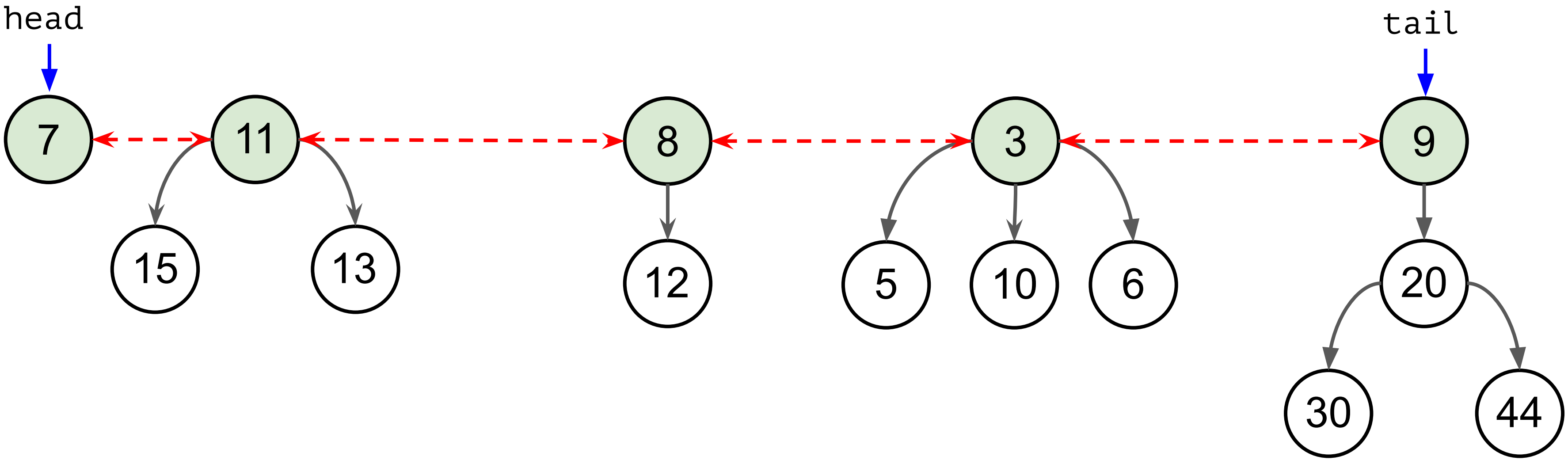
Фибоначчиева пирамида



Хочу достать минимальный элемент из пирамиды.
Удаляем корень, разбираемся с детьми (два раза вызывая merge).

А где новый минимальный элемент теперь?

Фибоначчиева пирамида



Хочу достать минимальный элемент из пирамиды.
Удаляем корень, разбираемся с детьми (два раза вызывая merge).

А где новый минимальный элемент теперь? Как скоро это превратится в список?

Фибоначчиева пирамида

Определение: для узла V будем говорить, что степень **degree(V)** - это количество его детей

Будем хранить в каждом узле его степень.

Фибоначчиева пирамида

Определение: для узла V будем говорить, что степень **degree(V)** - это количество его детей

Будем хранить в каждом узле его степень.

Определение: для дерева T будем говорить, что степень **degree(T)** - это degree(root), где root - корень дерева

Фибоначчиева пирамида

Определение: для узла V будем говорить, что степень **degree(V)** - это количество его детей

Будем хранить в каждом узле его степень.

Определение: для дерева T будем говорить, что степень **degree(T)** - это degree(root), где root - корень дерева

Обозначение: **D(N)** - это ограничение сверху на степень корня в Фибоначчиевой пирамиде из **N** элементов.

Фибоначчиева пирамида

Введем процедуру Consolidate(Fib) -> Fib.

Цель данной процедуры - добиться того, чтобы у всех корней в Фибоначчиевой пирамиде было разная степень degree(V).

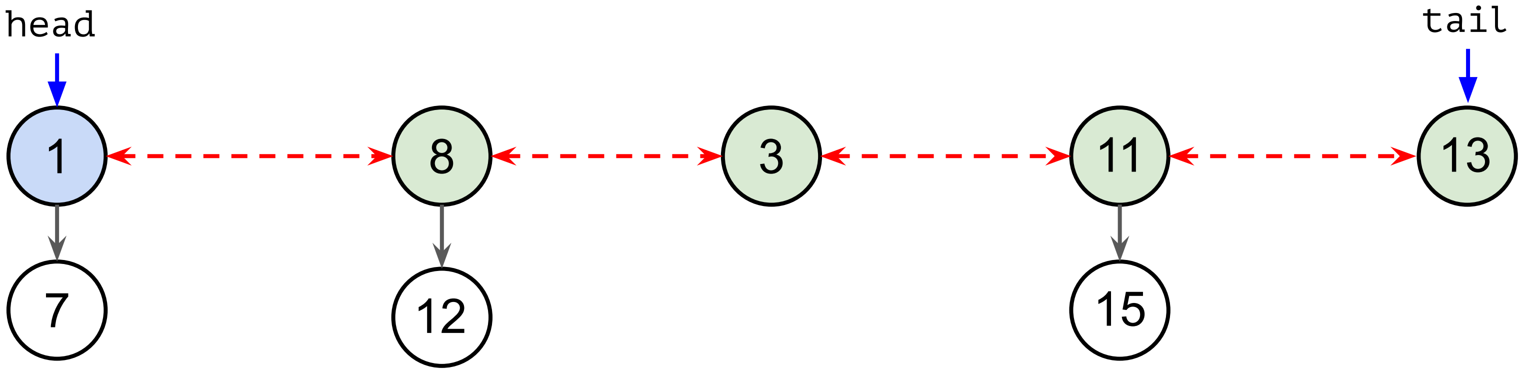
Фибоначчиева пирамида

Введем процедуру Consolidate(Fib) -> Fib.

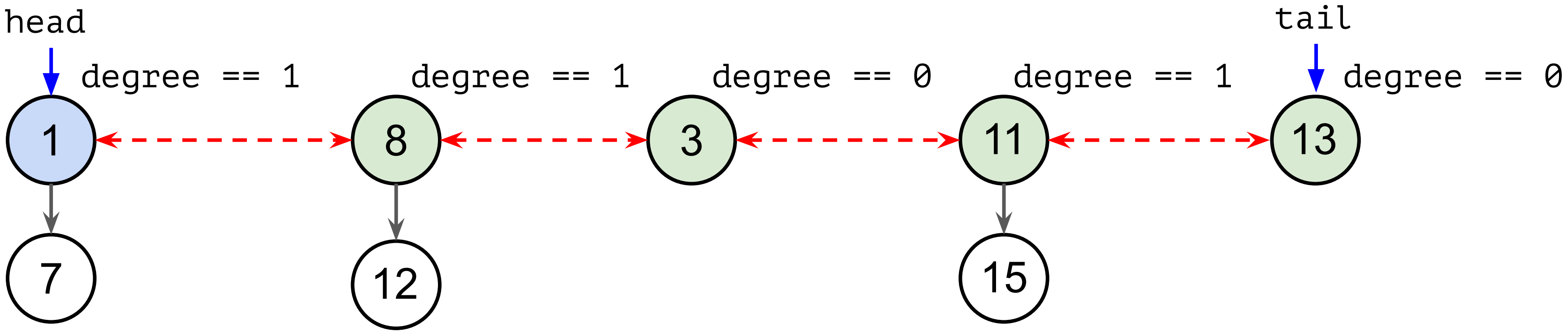
Цель данной процедуры - добиться того, чтобы у всех корней в Фибоначчиевой пирамиде было разная степень degree(V).

На уровне интуиции: она слишком расхлябистую пирамиду приводит к более собранному виду.

Фибоначчиева пирамида

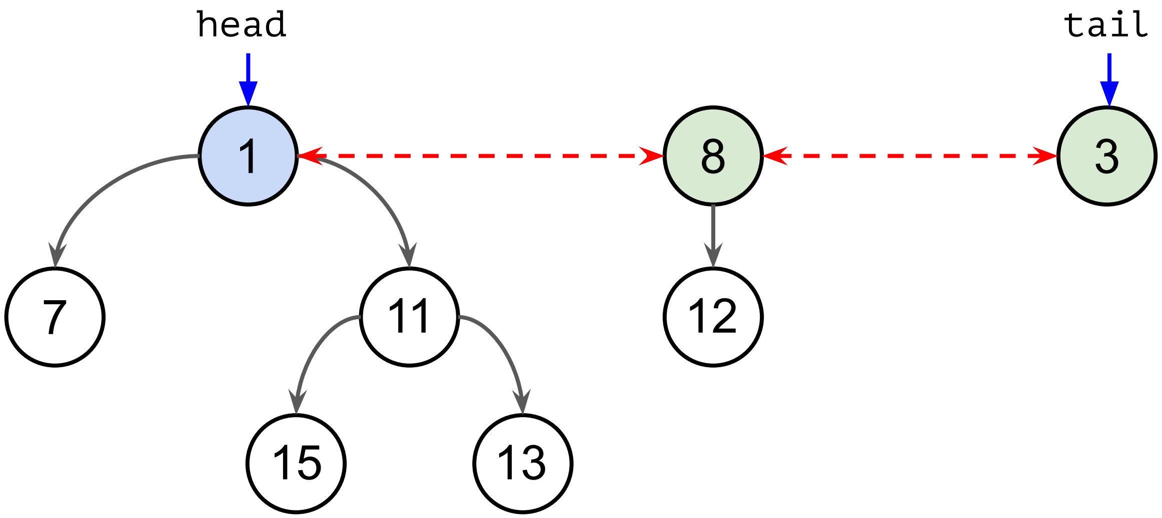


Фибоначчиева пирамида

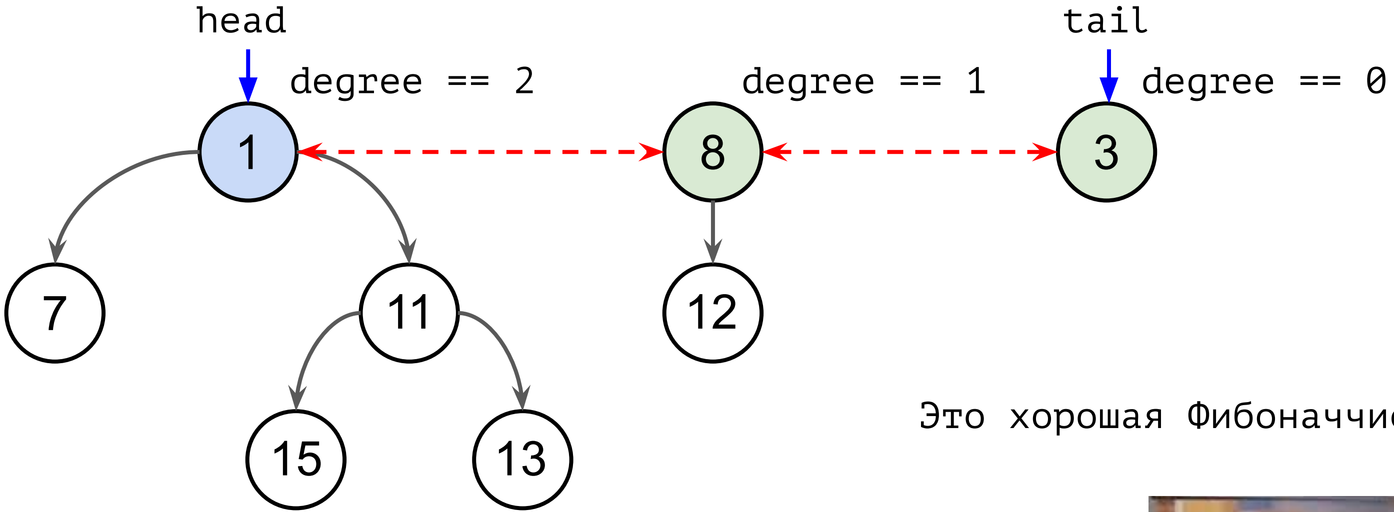


Это плохая Фибоначчиева пирамида

Фибоначчиева пирамида



Фибоначчиева пирамида



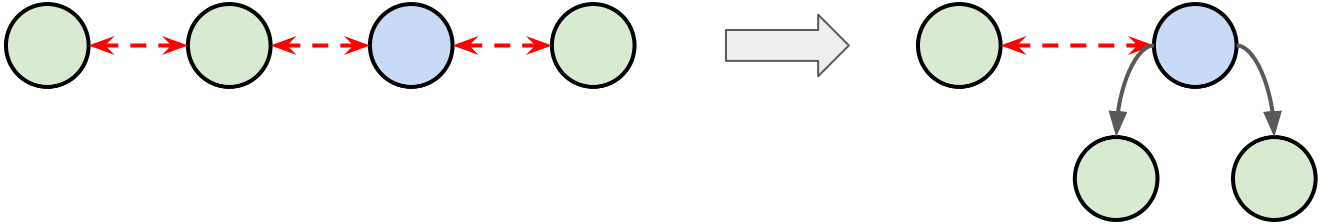
Это хорошая Фибоначчиева пирамида

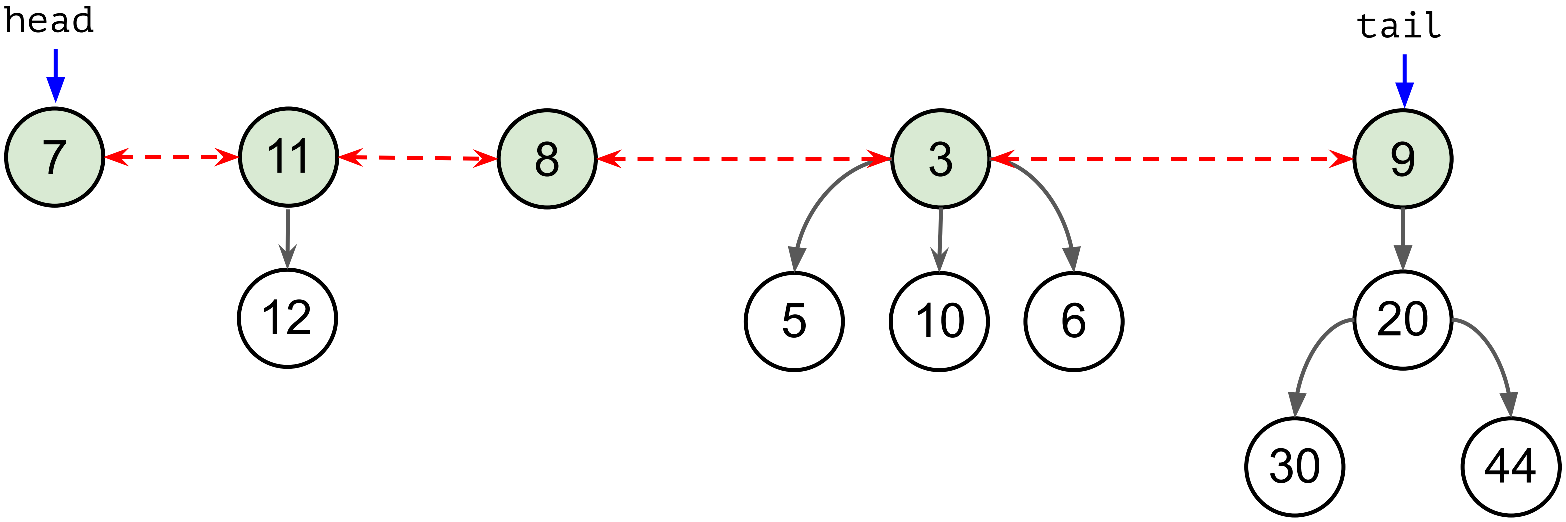
Фибоначчиева пирамида

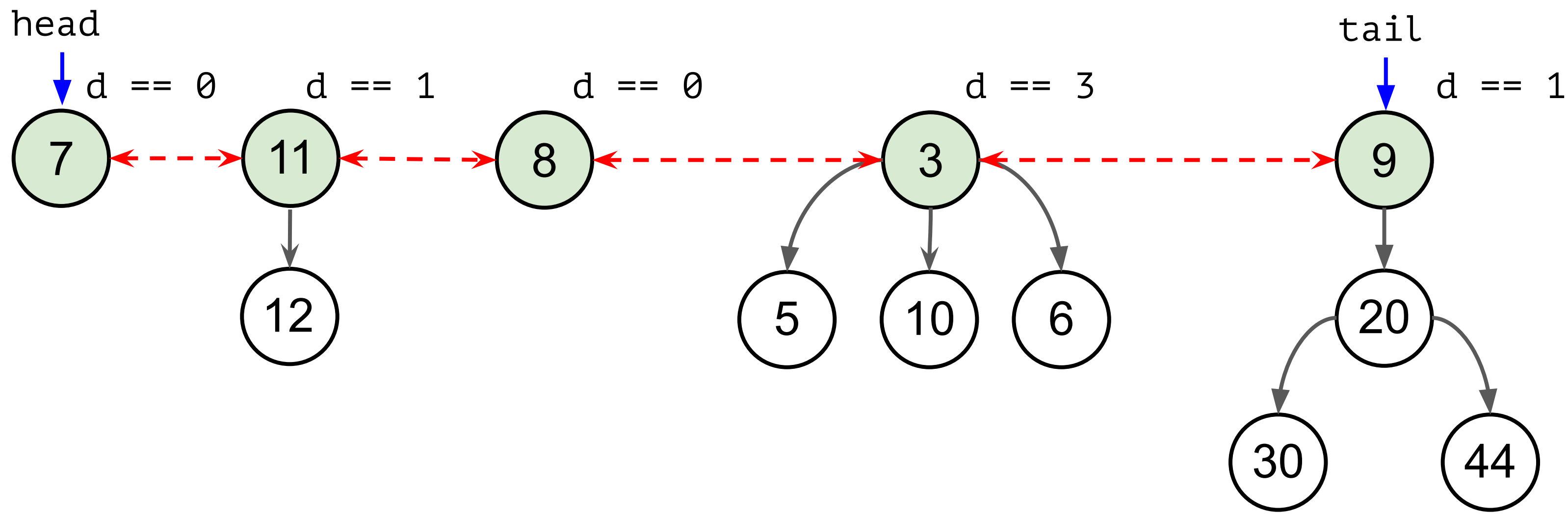
Введем процедуру Consolidate(Fib) -> Fib.

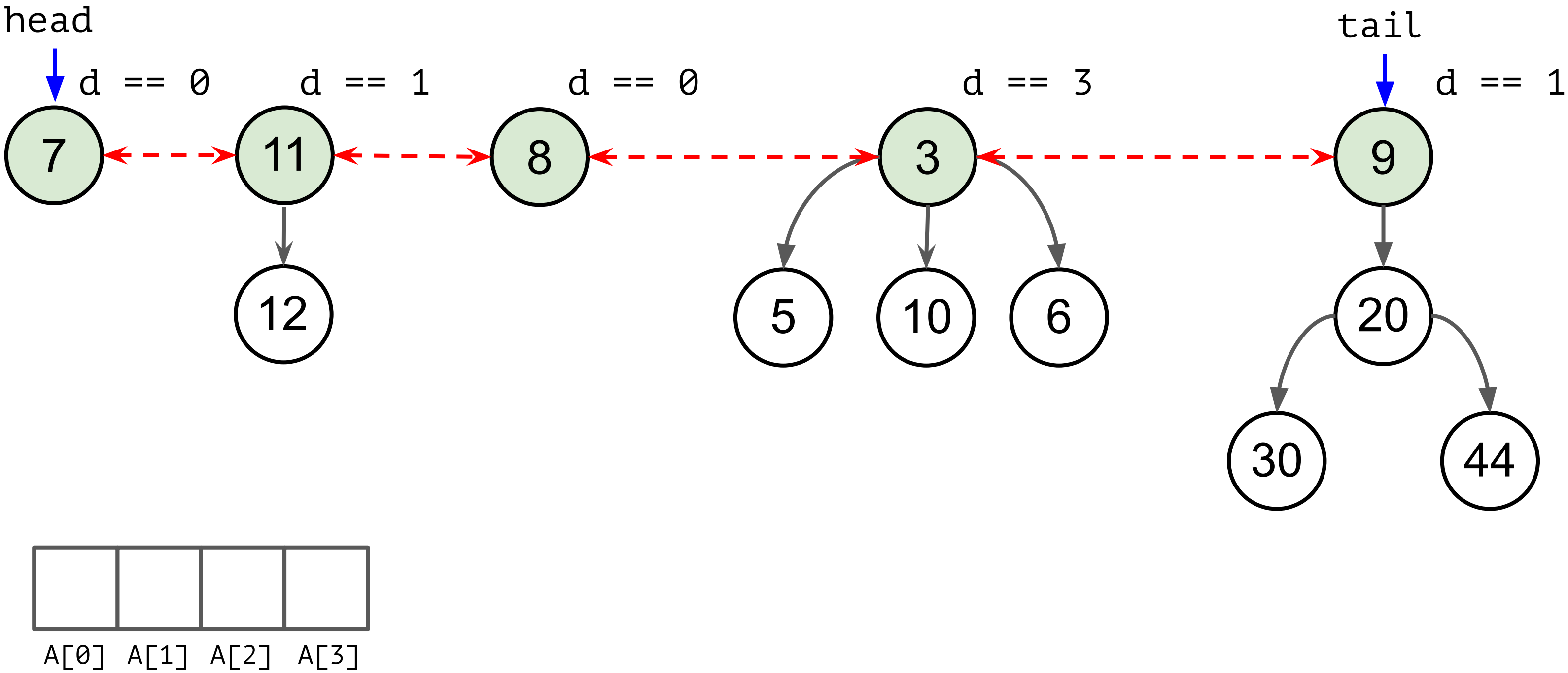
Цель данной процедуры - добиться того, чтобы у всех корней в Фибоначчиевой пирамиде было разная степень degree(V).

На уровне интуиции: она слишком расхлябистую пирамиду приводит к более собранному виду.

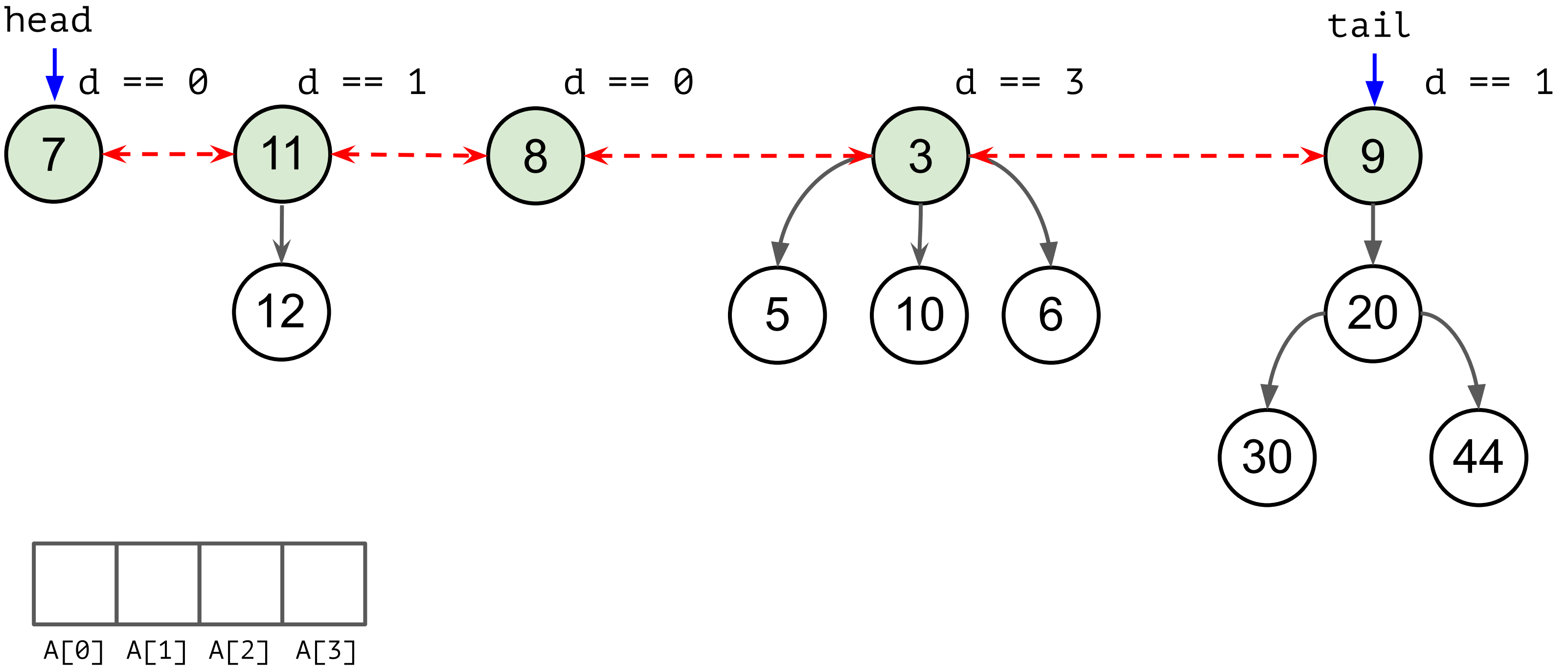



head
tail
7
11
8
3
9
12
5
10
6
20
30
44


head
d == 0
d == 1
d == 0
d == 3
tail
d == 1
7
11
8
3
9
12
5
10
6
20
30
44

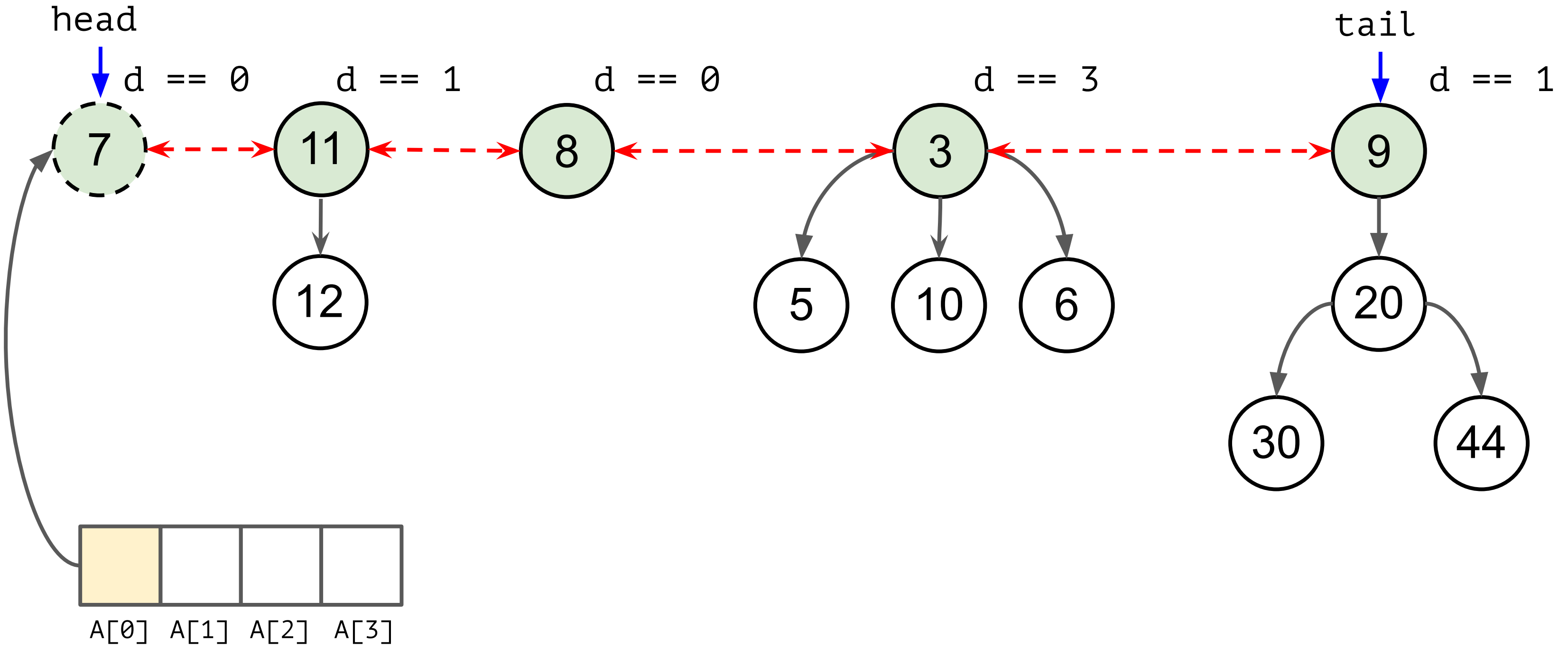


Заводим массив указателей на корни A размера D(N)



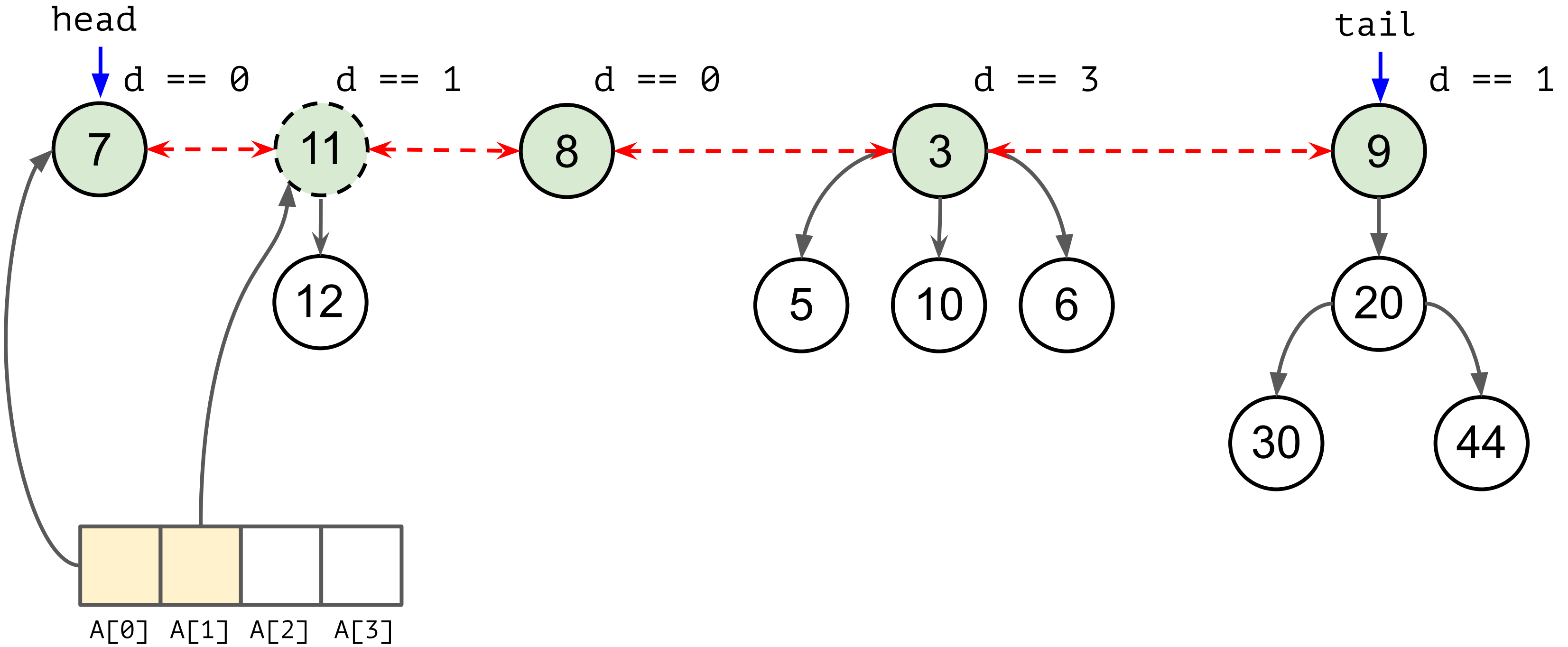
Заводим массив указателей на корни A размера D(N)

Пытаемся для каждого корня R записать указатель на него в A[degree(R)]



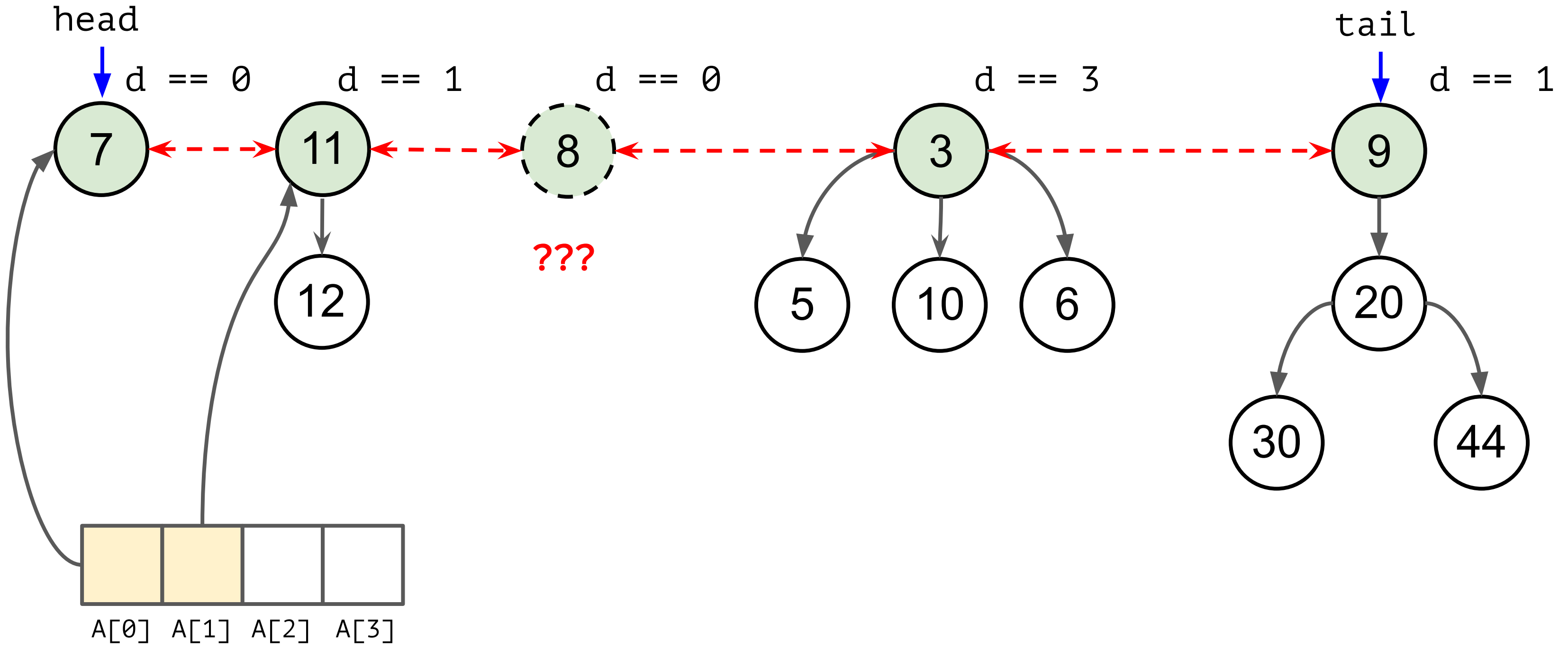
Заводим массив указателей на корни A размера D(N)

Пытаемся для каждого корня R записать указатель на него в A[degree(R)]



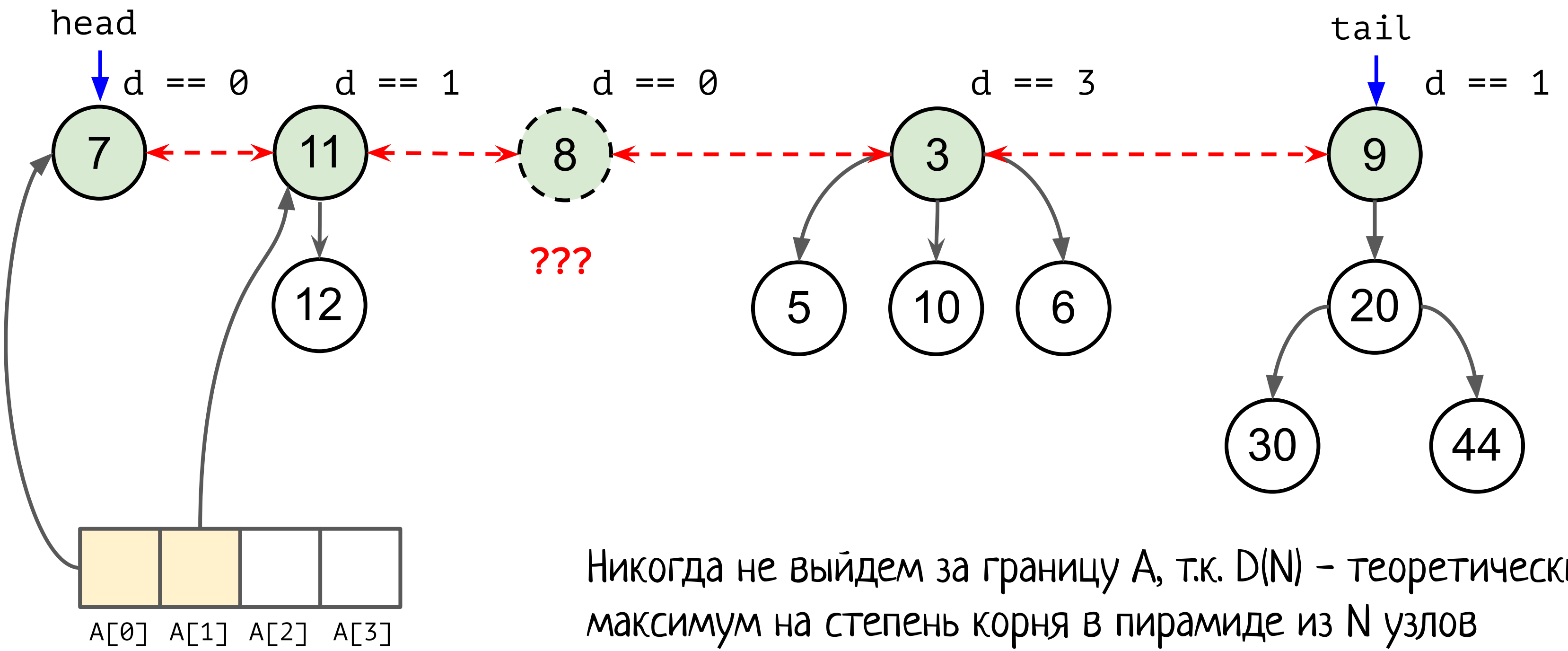
Заводим массив указателей на корни A размера D(N)

Пытаемся для каждого корня R записать указатель на него в A[degree(R)]



Заводим массив указателей на корни A размера D(N)

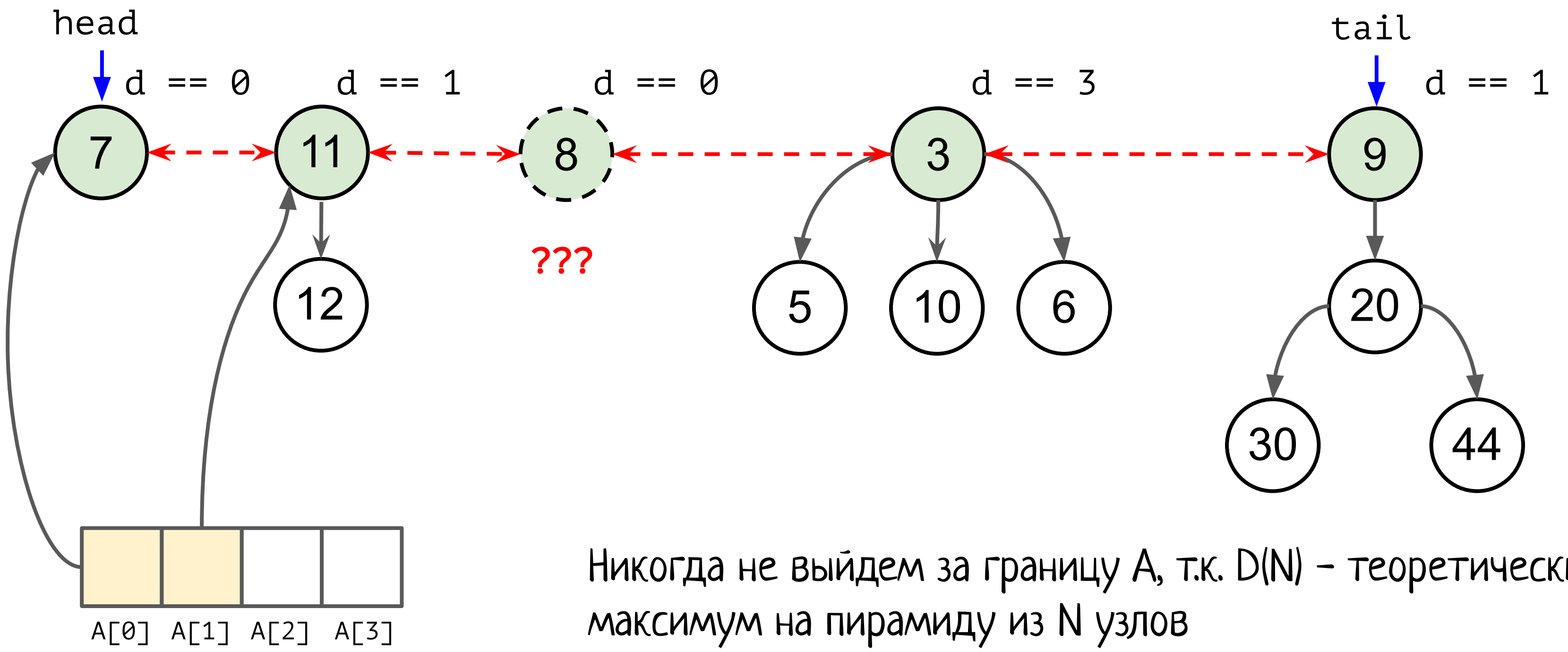
Пытаемся для каждого корня R записать указатель на него в A[degree(R)]



Никогда не выйдем за границу A, т.к. D(N) – теоретический максимум на степень корня в пирамиде из N узлов

Заводим массив указателей на корни A размера D(N)

Пытаемся для каждого корня R записать указатель на него в A[degree(R)]

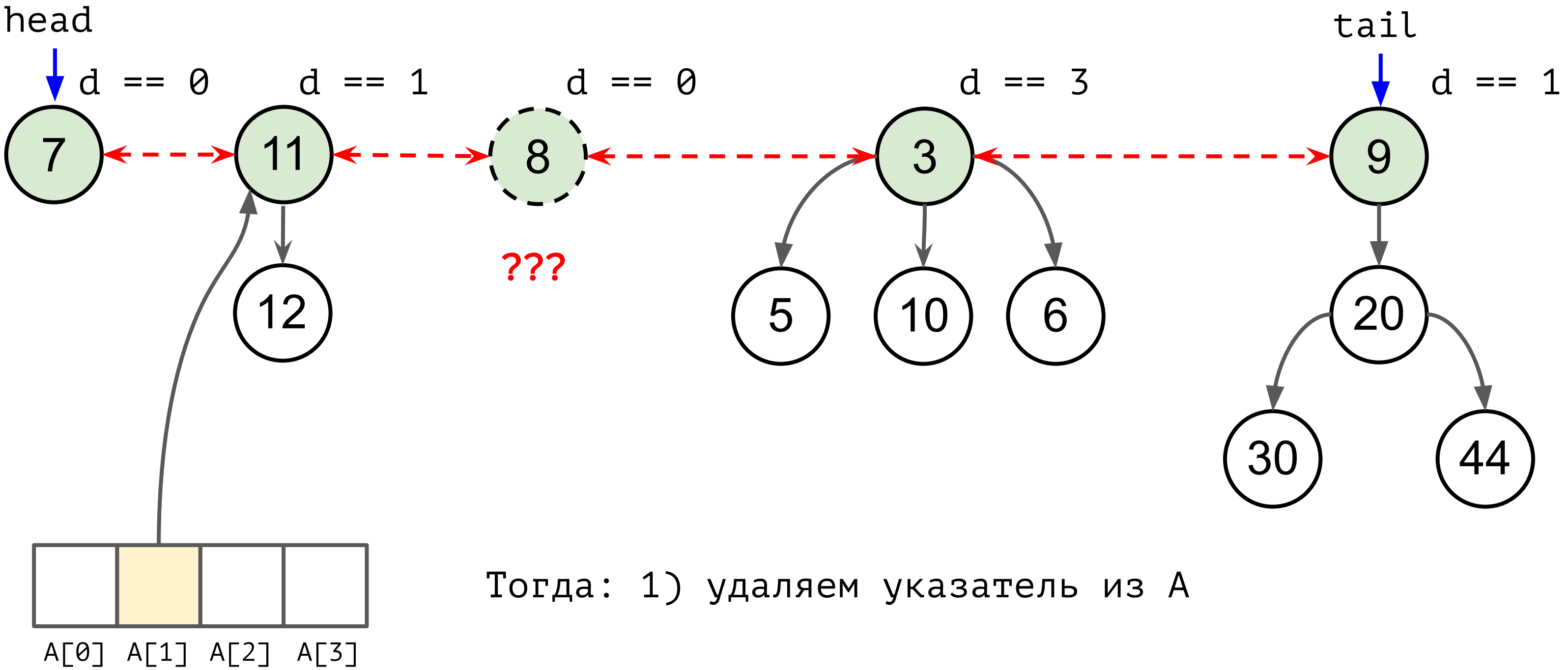


Никогда не выйдем за границу A, т.к. D(N) – теоретический максимум на пирамиду из N узлов

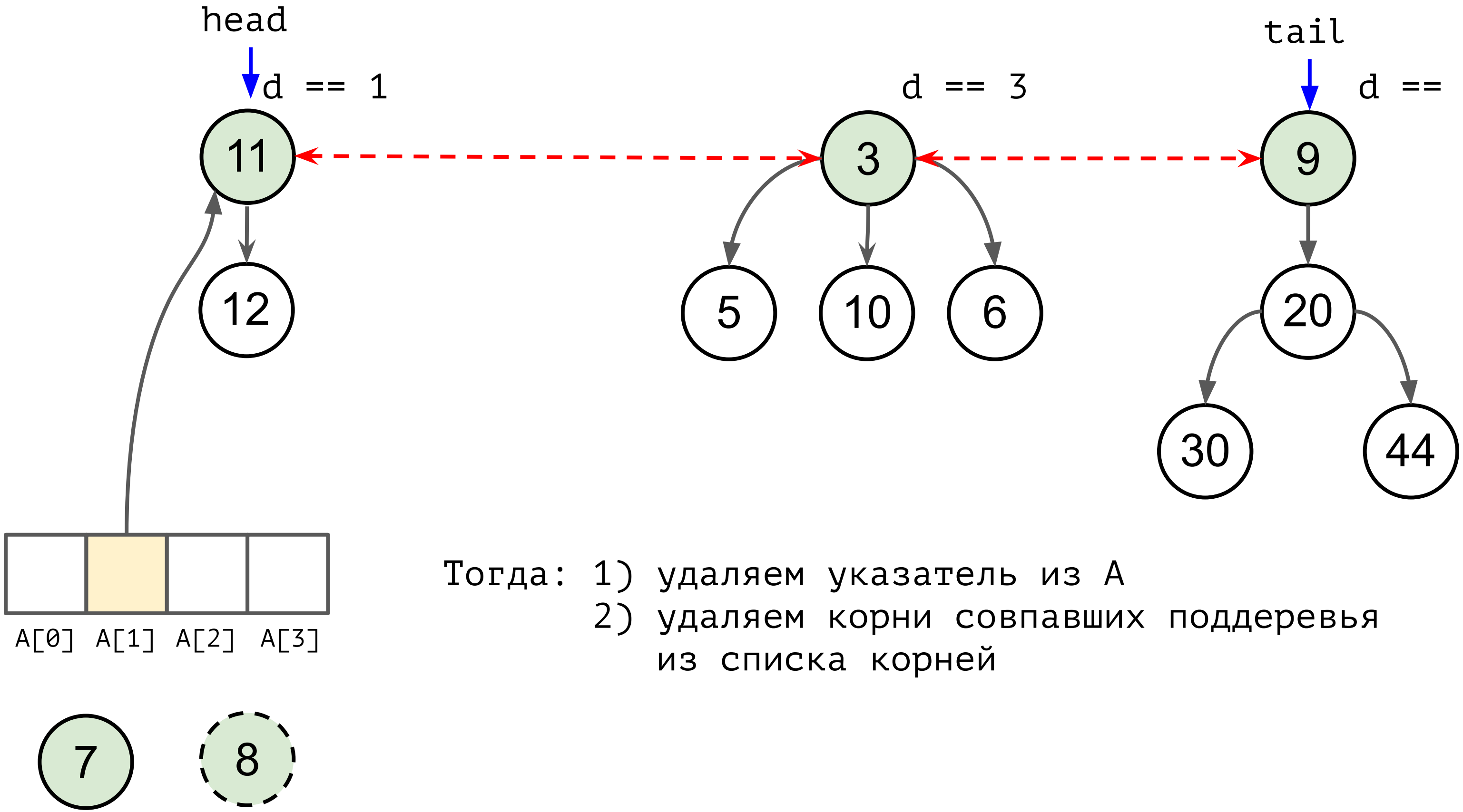
Заводим массив указателей на корни A размера D(N)

Пытаемся для каждого корня R записать указатель на него в A[degree(R)]

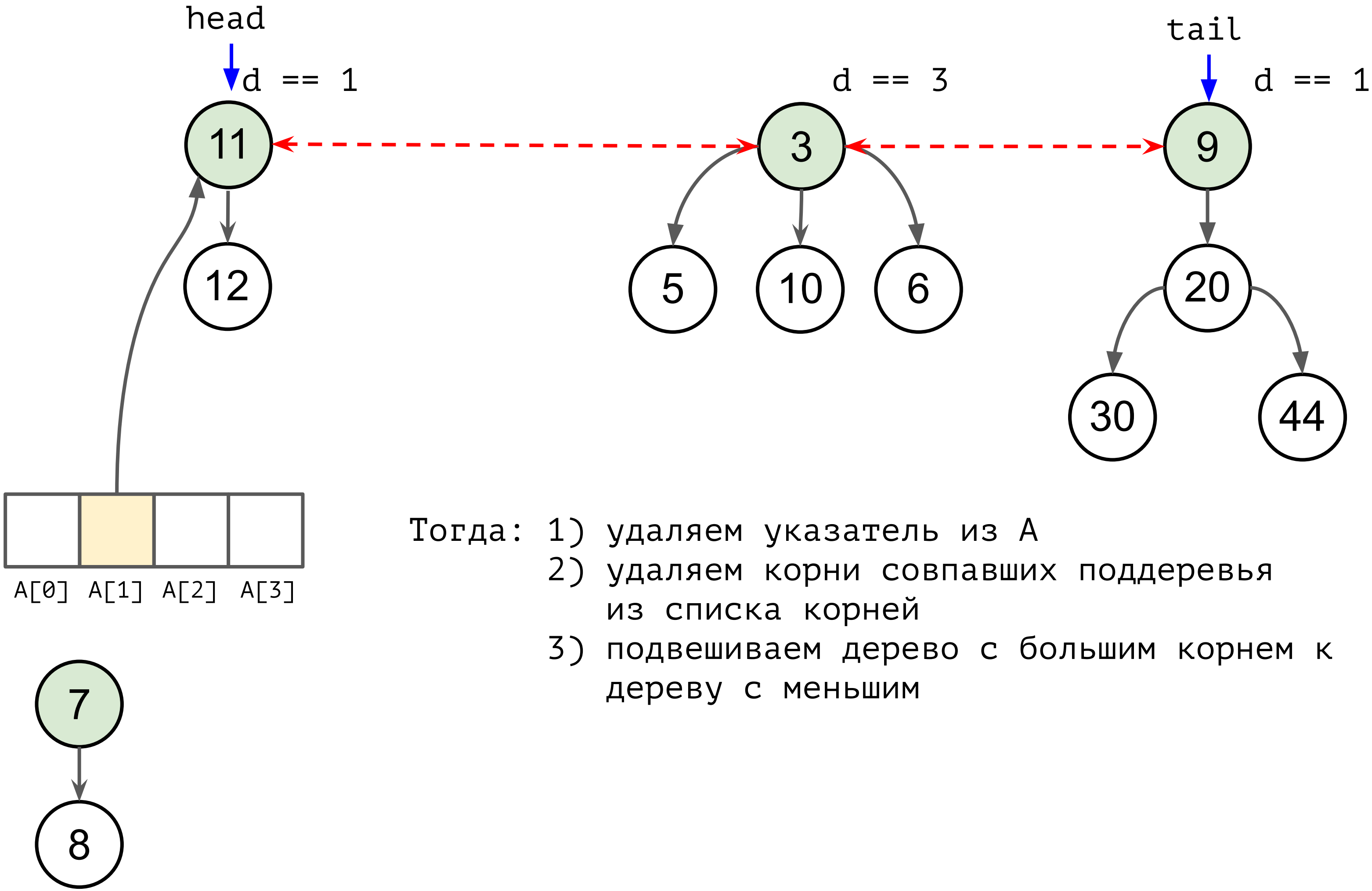
Если занято, то мы имеем дело с деревьями с одинаковыми degree!



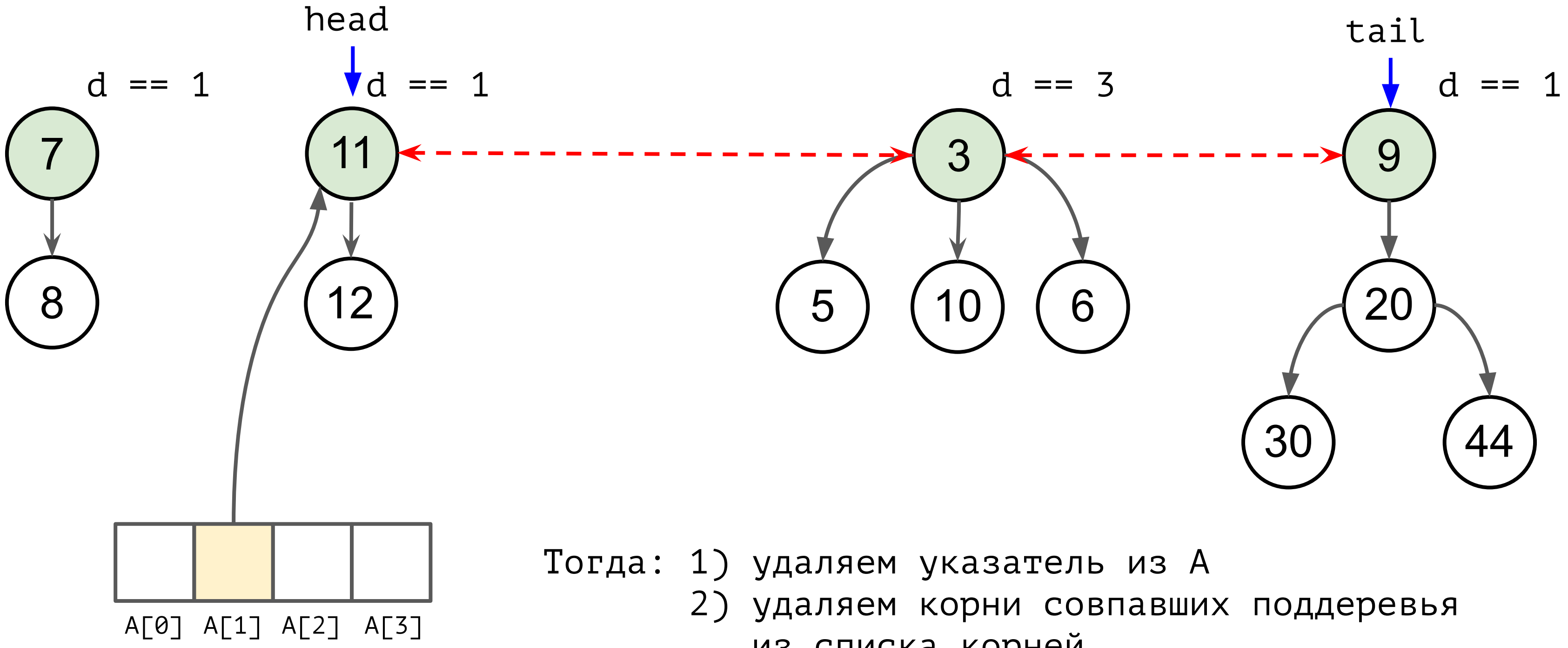
Тогда: 1) удаляем указатель из A



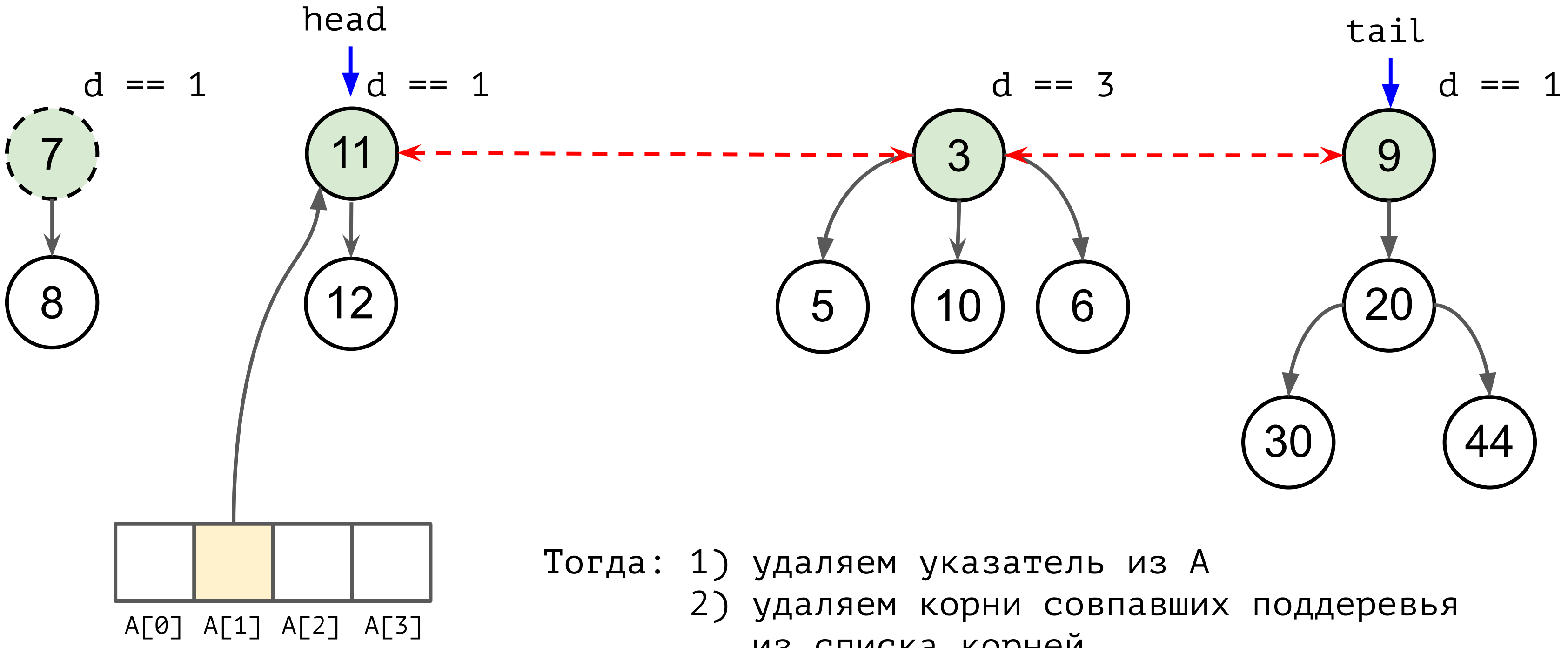
Тогда: 1) удаляем указатель из A
2) удаляем корни совпавших поддеревья из списка корней



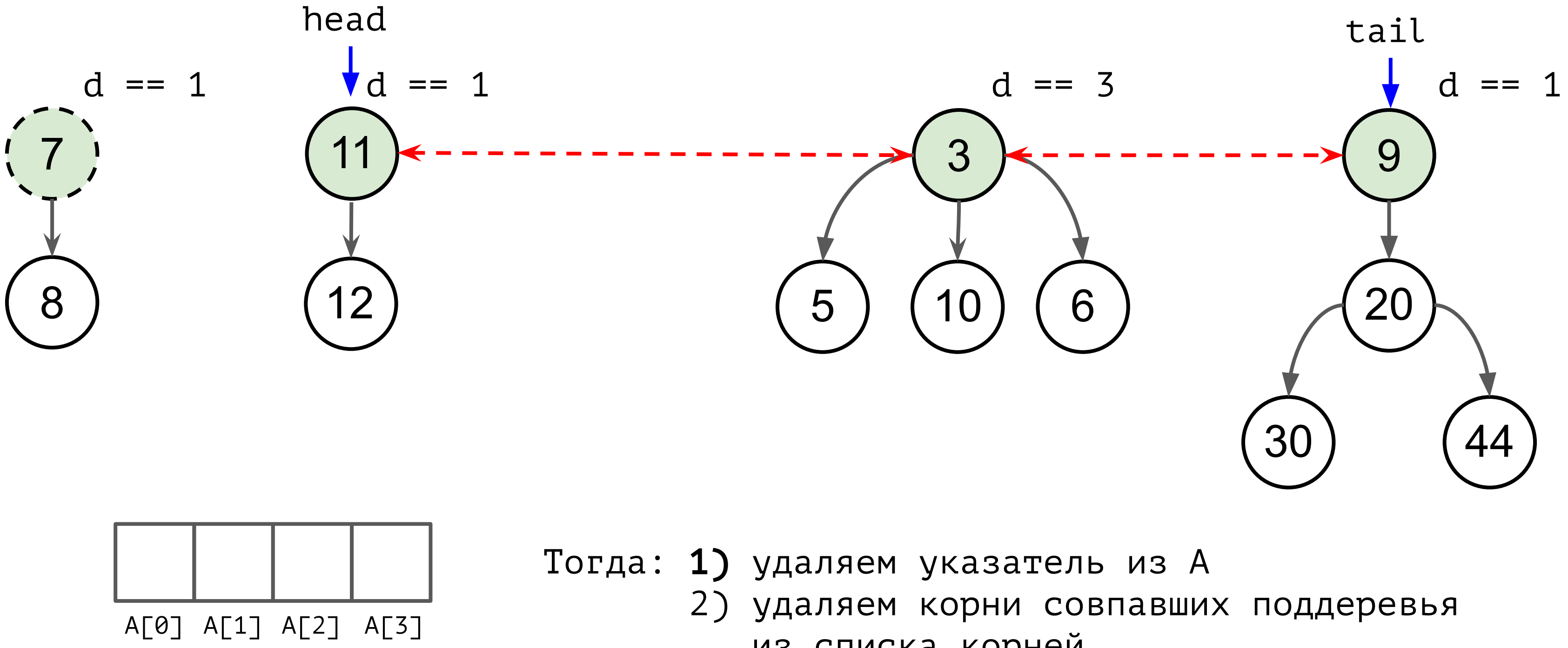
Тогда: 1) удаляем указатель из A
2) удаляем корни совпавших поддеревья из списка корней
3) подвешиваем дерево с большим корнем к дереву с меньшим



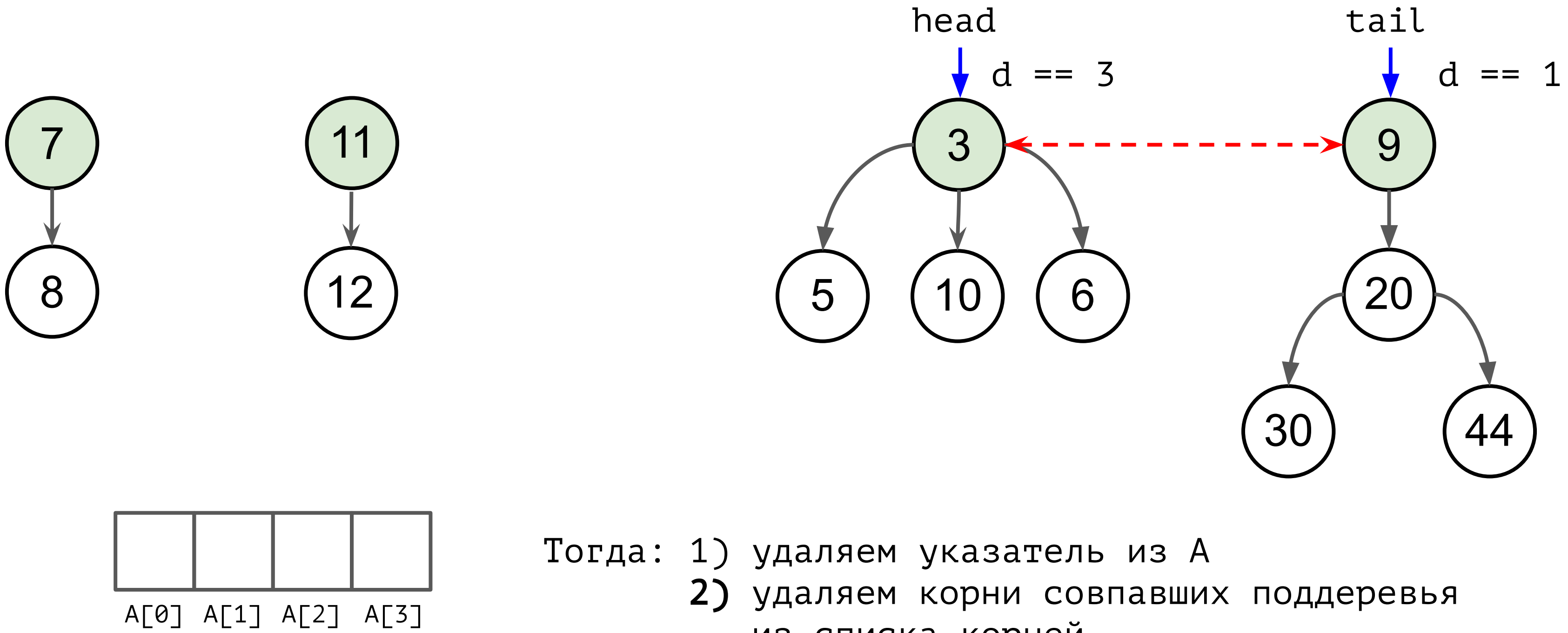
Тогда: 1) удаляем указатель из A
2) удаляем корни совпавших поддеревья из списка корней
3) подвешиваем дерево с большим корнем к дереву с меньшим



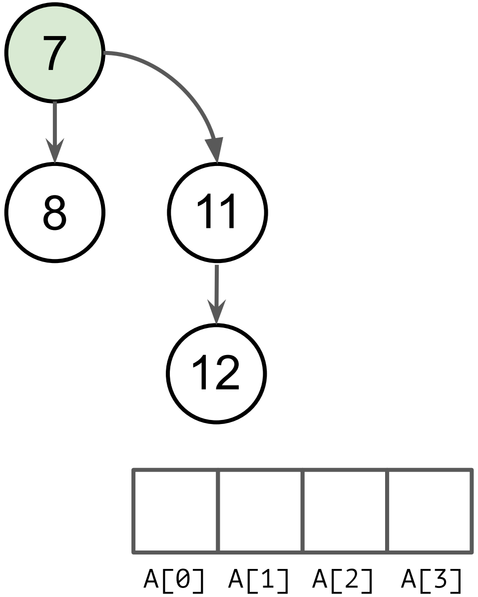
Тогда: 1) удаляем указатель из A
2) удаляем корни совпавших поддеревья из списка корней
3) подвешиваем дерево с большим корнем к дереву с меньшим
4) пробуете снова добавить в A!

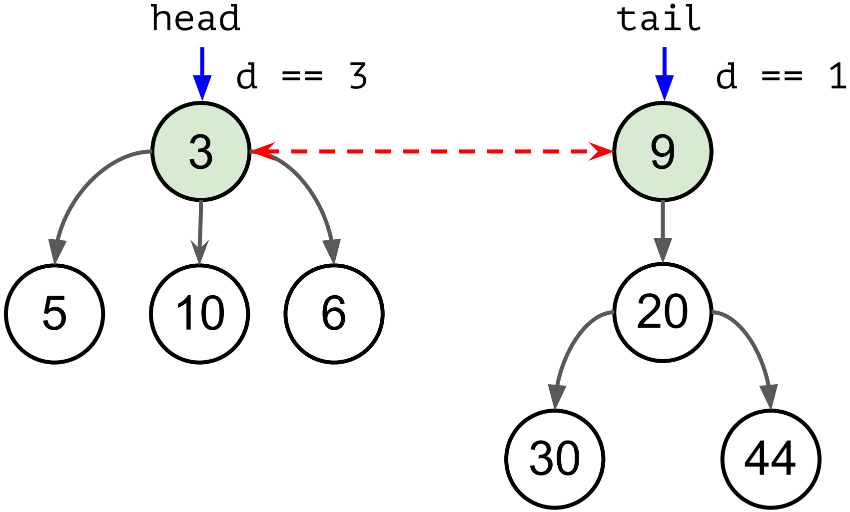


Тогда: **1)** удаляем указатель из А
2) удаляем корни совпавших поддеревья из списка корней
3) подвешиваем дерево с большим корнем к дереву с меньшим
4) пробуете снова добавить в А!

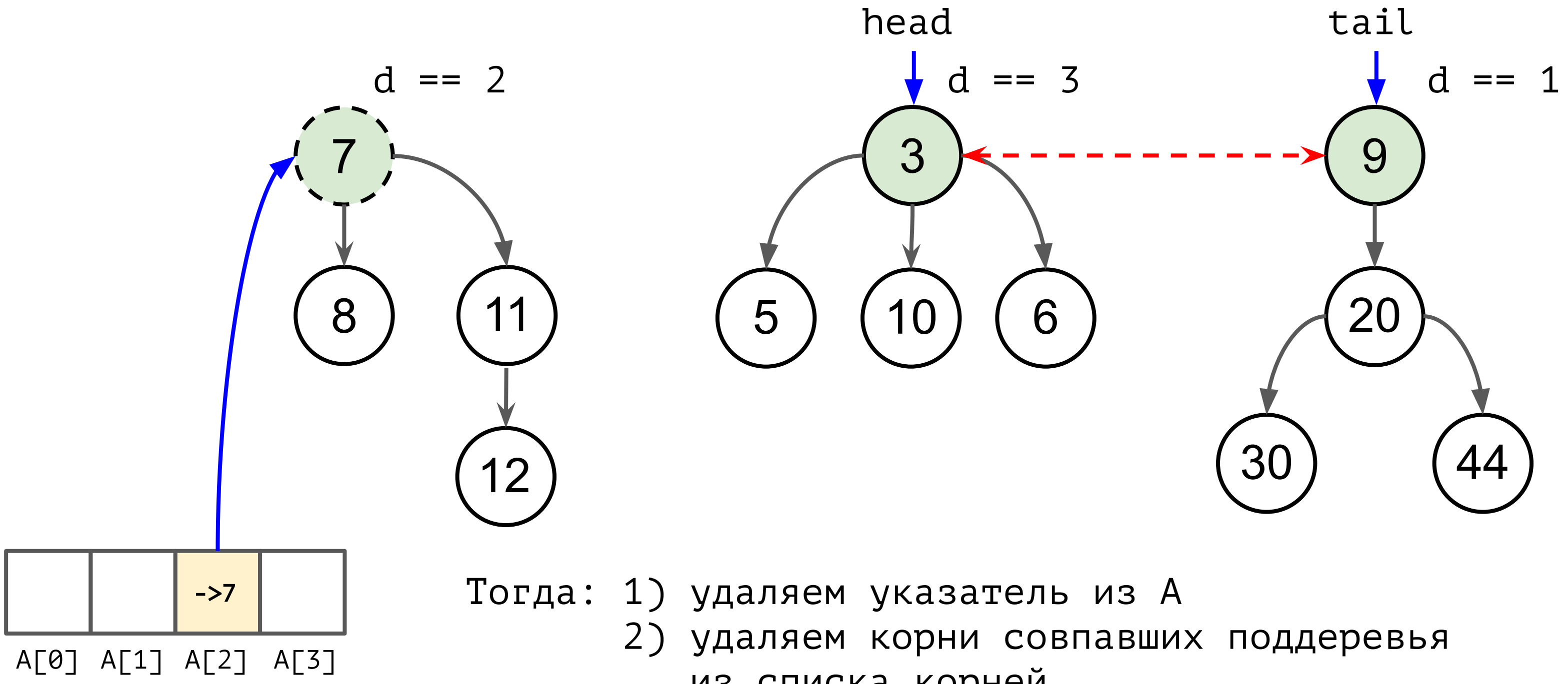


Тогда: 1) удаляем указатель из A
2) удаляем корни совпавших поддеревья из списка корней
3) подвешиваем дерево с большим корнем к дереву с меньшим
4) пробуете снова добавить в A!

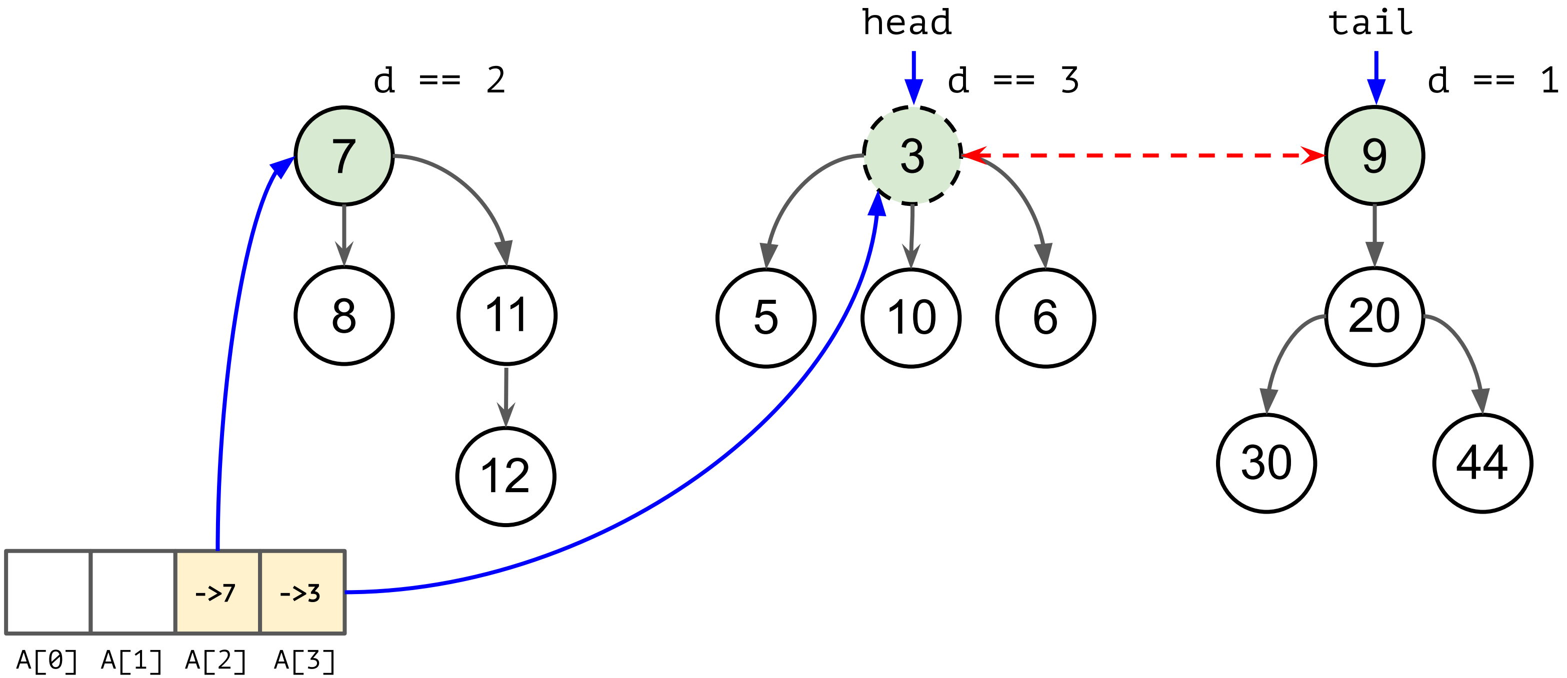




Тогда: 1) удаляем указатель из A
2) удаляем корни совпавших поддеревья из списка корней
3) подвешиваем дерево с большим корнем к дереву с меньшим
4) пробуете снова добавить в A!



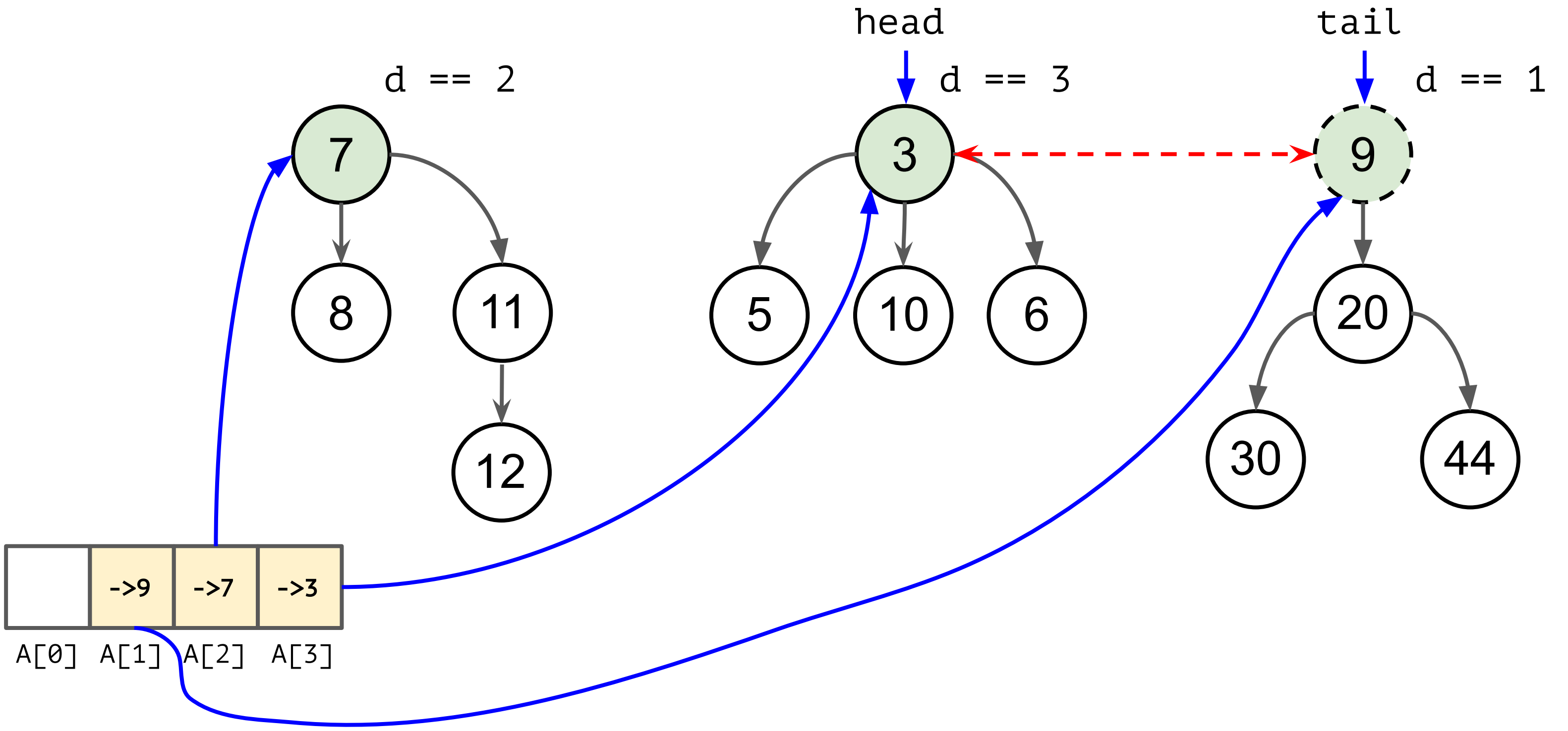
Тогда: 1) удаляем указатель из A
2) удаляем корни совпавших поддеревья из списка корней
3) подвешиваем дерево с большим корнем к дереву с меньшим
4) пробуете снова добавить в A!



Заводим массив указателей на корни A размера D(N)

Пытаемся для каждого корня R записать указатель на него в A[degree(R)]

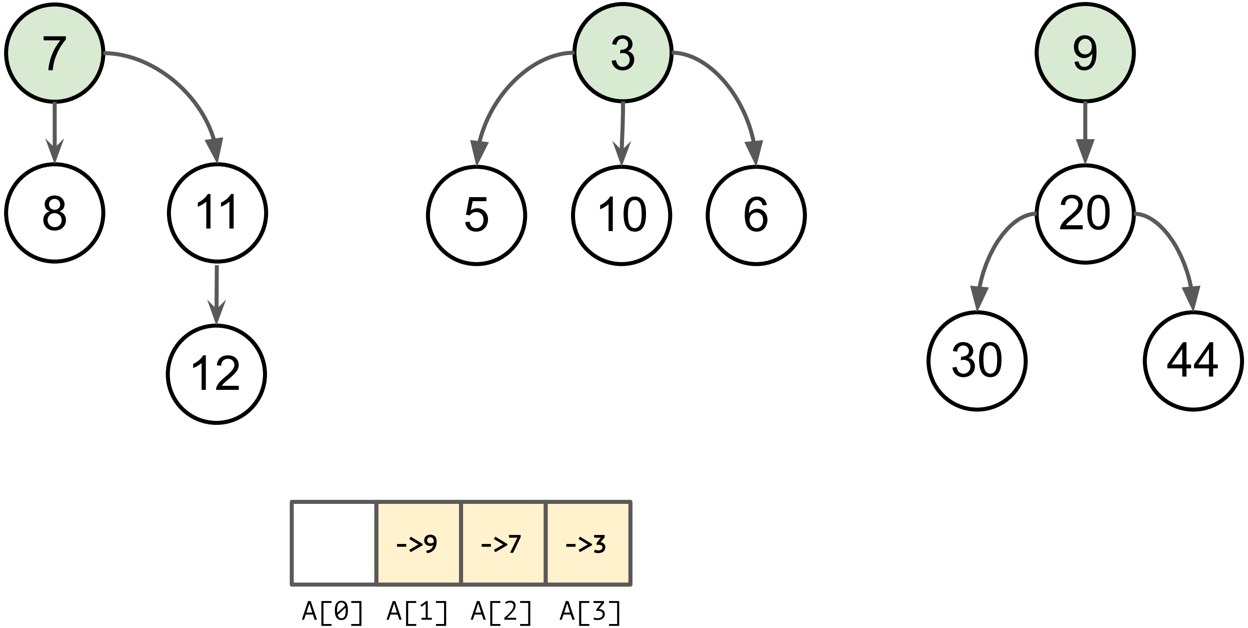
Если занято, то мы имеем дело с деревьями с одинаковыми degree!



Заводим массив указателей на корни A размера D(N)

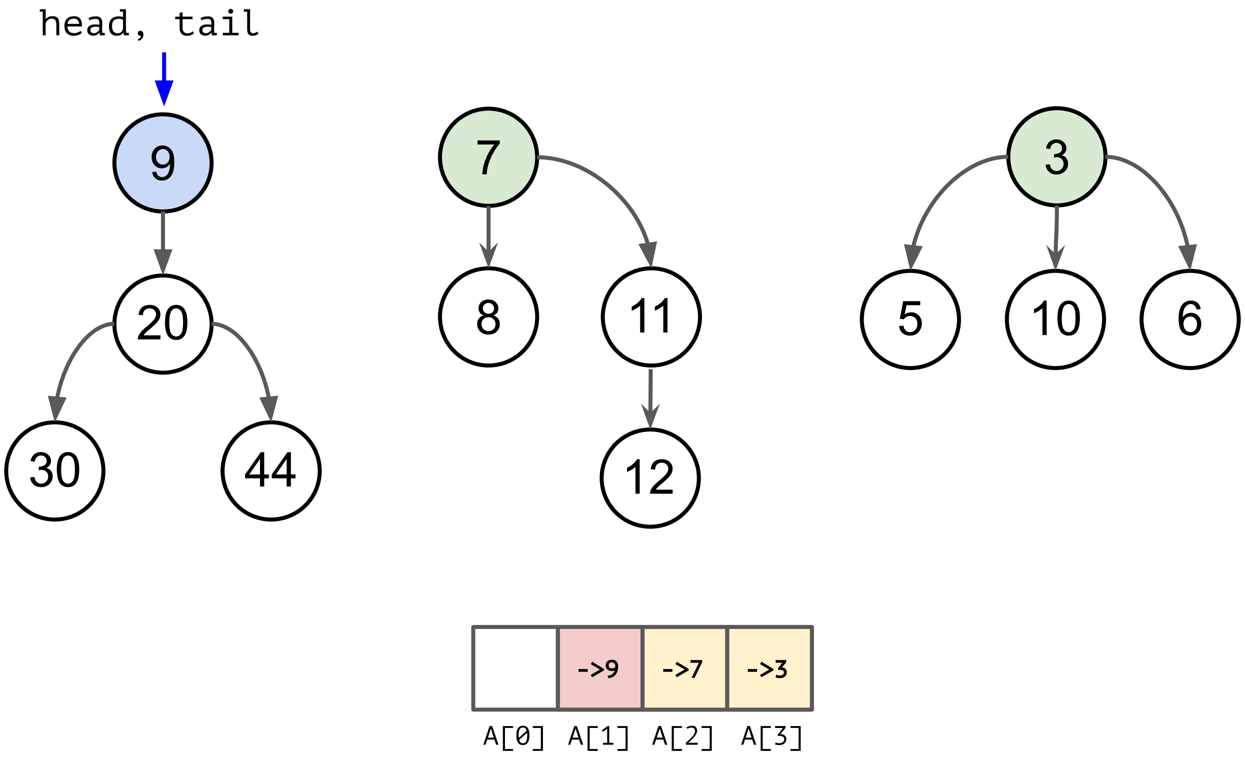
Пытаемся для каждого корня R записать указатель на него в A[degree(R)]

Если занято, то мы имеем дело с деревьями с одинаковыми degree!



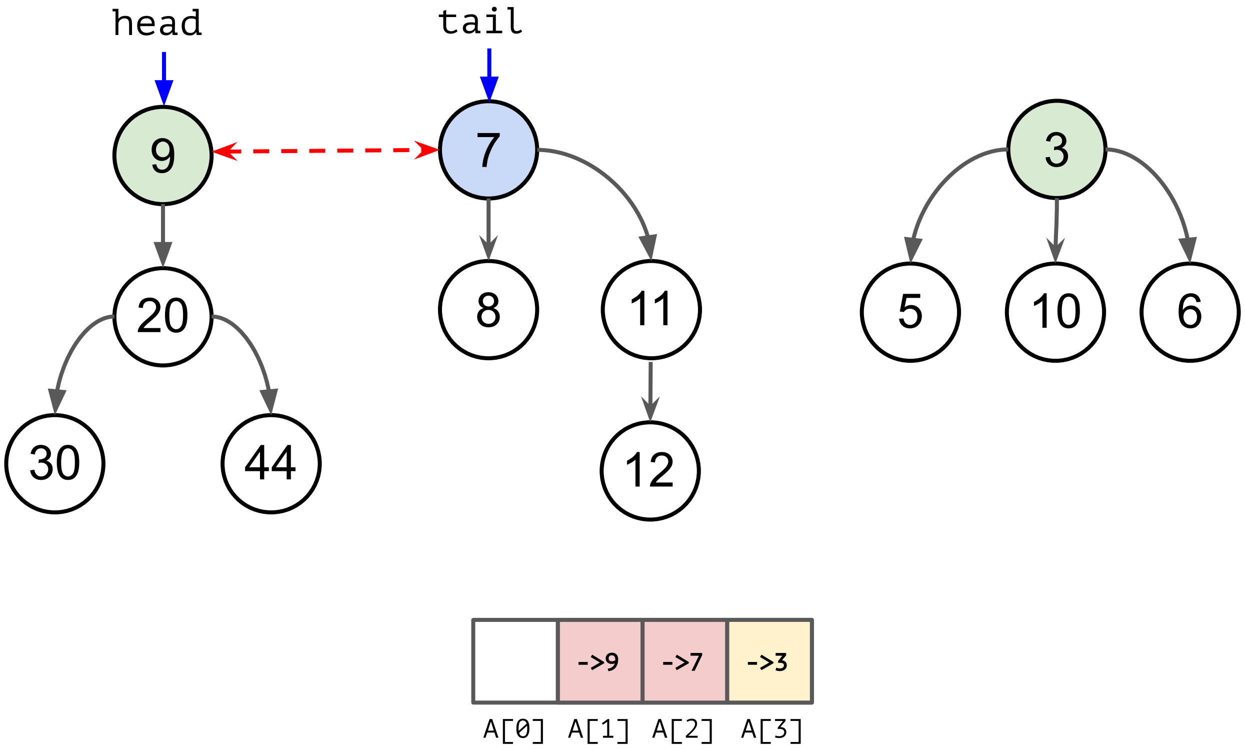
После того, как все текущие корни обработаны, проходим по массиву и перестраиваем весь список корней.

А заодно ищем новый минимум!



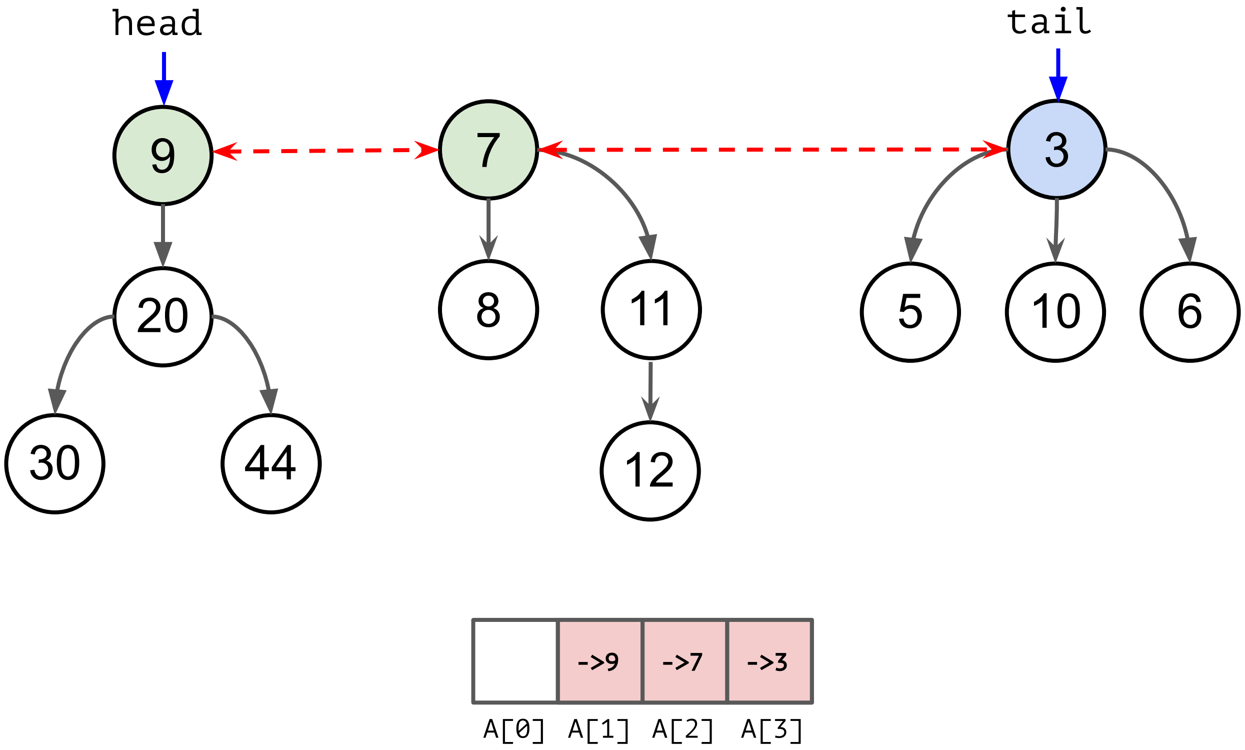
После того, как все текущие корни обработаны, проходим по массиву и перестраиваем весь список корней.

А заодно ищем новый минимум!



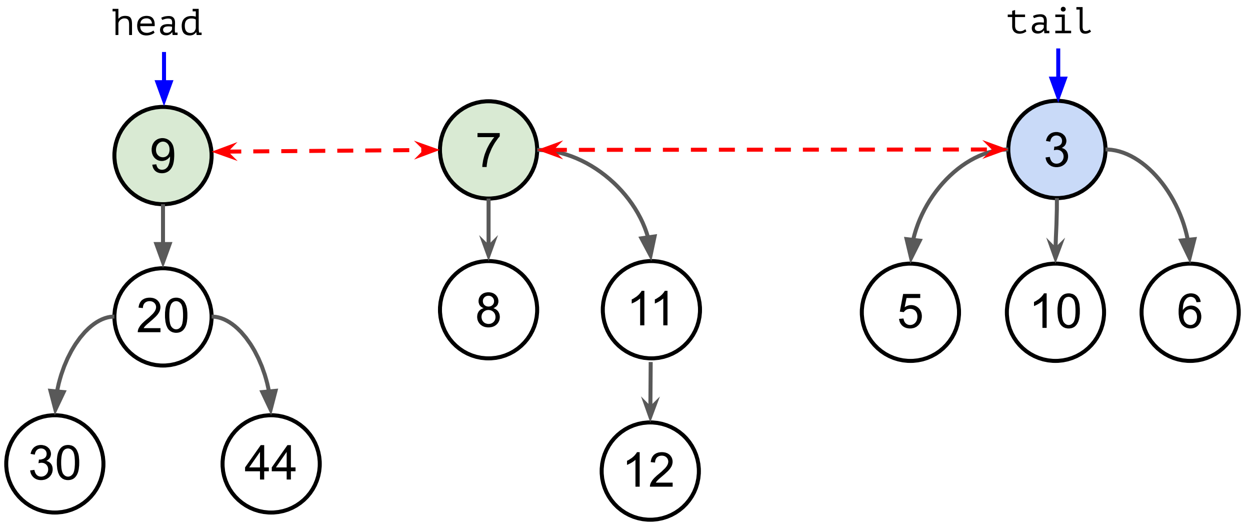
После того, как все текущие корни обработаны, проходим по массиву и перестраиваем весь список корней.

А заодно ищем новый минимум!



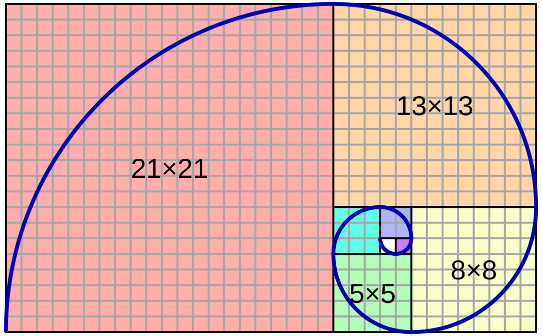
После того, как все текущие корни обработаны, проходим по массиву и перестраиваем весь список корней.

А заодно ищем новый минимум!



Это хорошая Фибоначчиева пирамида

Фибоначчиева пирамида

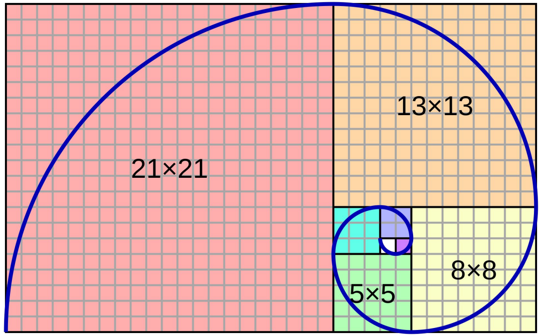


Операции:

```
✓ 1.  insert(value)         -> O(1)
✓ 2.  peek_min()            -> O(1)
? 3.  extract_min()         -> ???
  4.  decrease_key(s, k)
✓ 5.  merge(f1, f2)         -> O(1)
  6.  delete(s)
```

Идея простая: давайте снова пользоваться процедурой merge.

Фибоначчиева пирамида

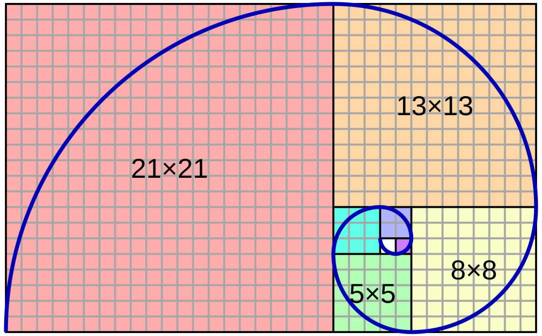


Операции:

```
✓ 1.  insert(value)          -> O(1)
✓ 2.  peek_min()             -> O(1)
? 3.  extract_min()          -> ???
  4.  decrease_key(s, k)
✓ 5.  merge(f1, f2)          -> O(1)
  6.  delete(s)
```

Идея простая: давайте снова пользоваться процедурой merge.
выкидываем корень => вмерживаем сыновей => вызываем consolidate.

Фибоначчиева пирамида

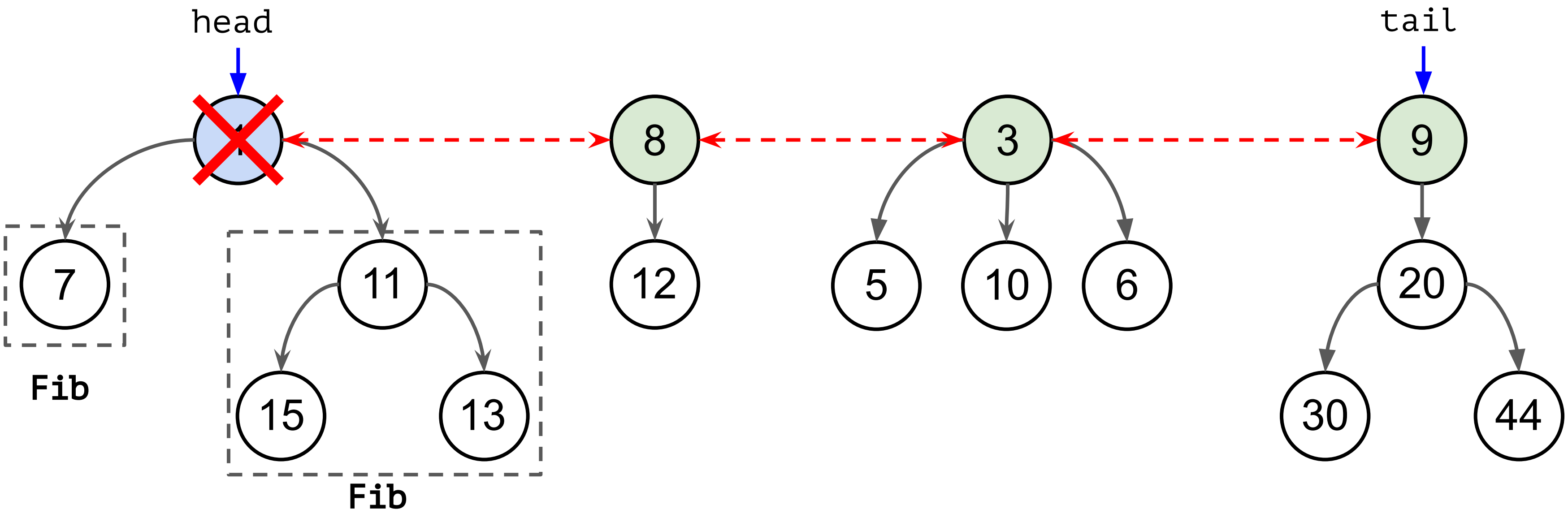


Операции:

```
✓ 1.  insert(value)        -> O(1)
✓ 2.  peek_min()           -> O(1)
? 3.  extract_min()        -> ???
  4.  decrease_key(s, k)
✓ 5.  merge(f1, f2)        -> O(1)
  6.  delete(s)
```

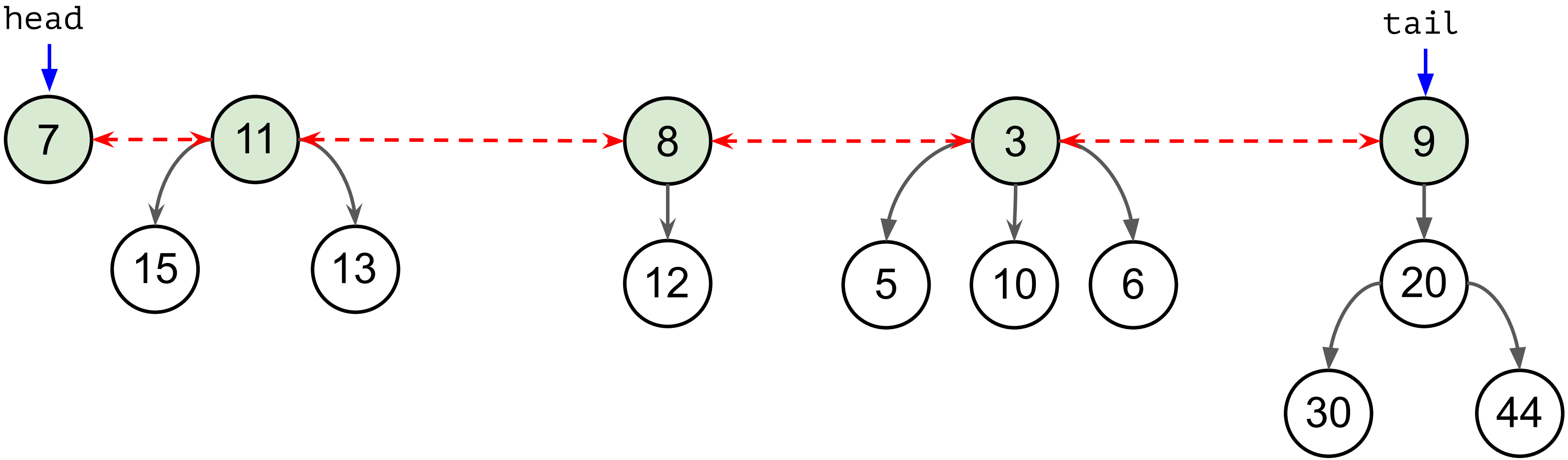
Идея простая: давайте снова пользоваться процедурой merge. выкидываем корень => вмерживаем сыновей => вызываем consolidate. Сложность?

Фибоначчиева пирамида



Хочу достать минимальный элемент из пирамиды.
Удаляем корень, разбираемся с детьми.

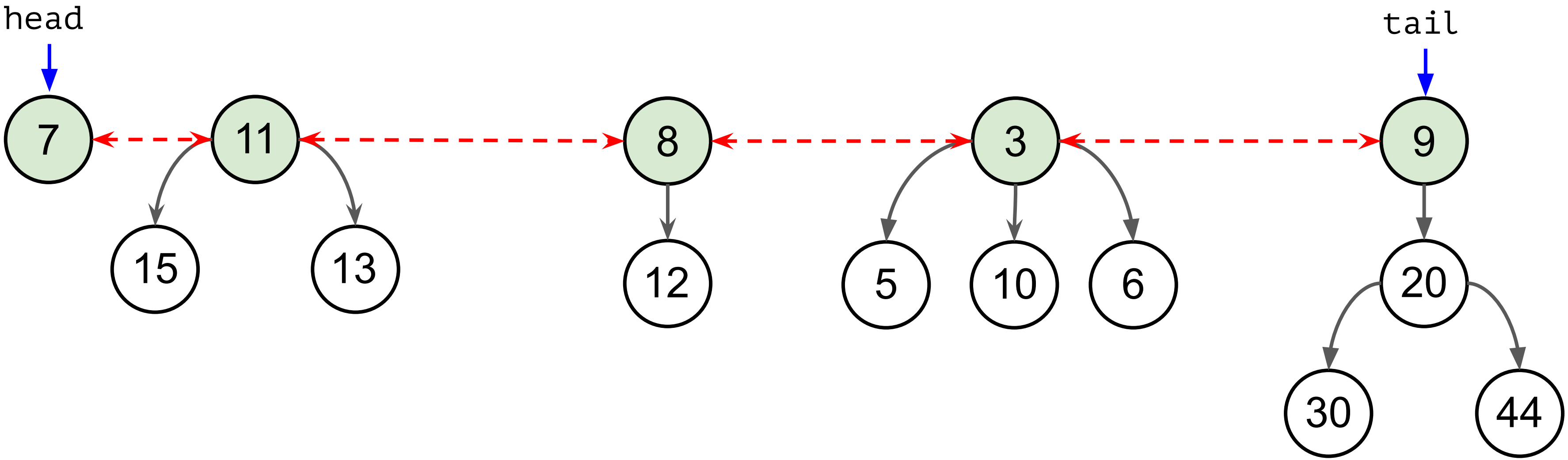
Фибоначчиева пирамида



Хочу достать минимальный элемент из пирамиды.
Удаляем корень, разбираемся с детьми (два раза вызывая merge).

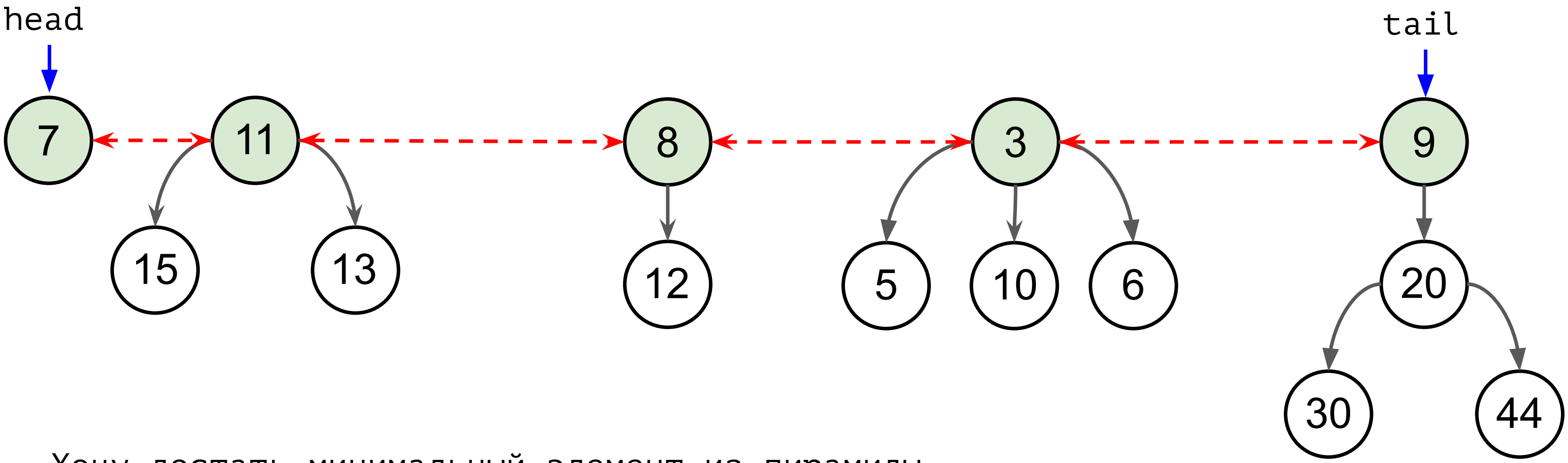
Фибоначчиева пирамида

D(N) - это ограничение сверху на степень корня в Фибоначчиевой пирамиде из **N** элементов.



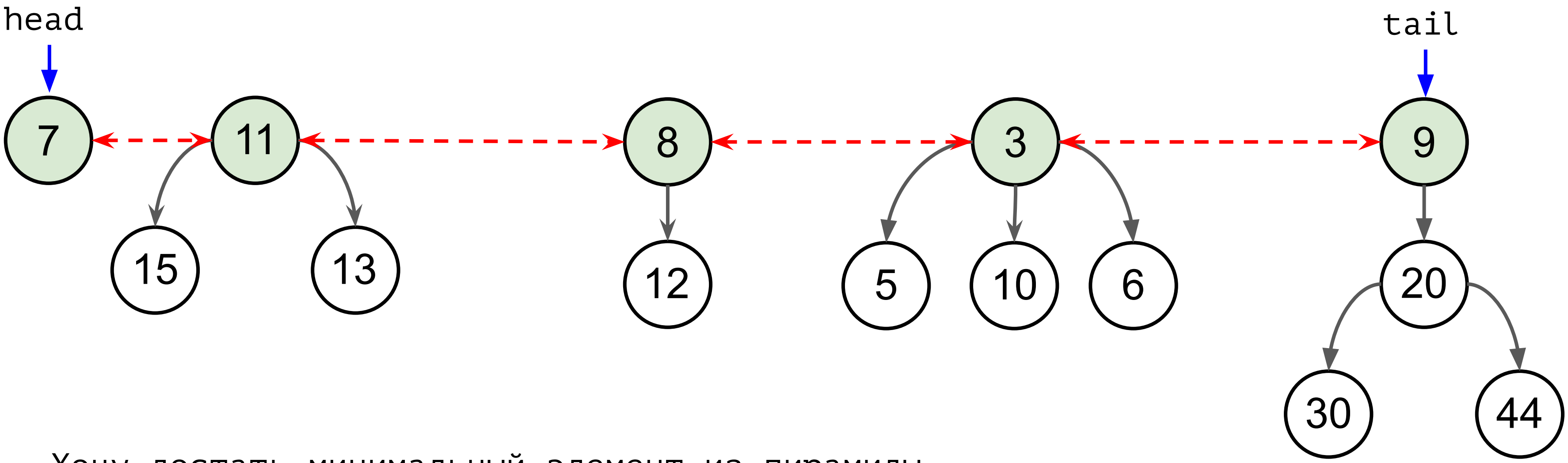
Хочу достать минимальный элемент из пирамиды.
Удаляем корень, разбираемся с детьми (два раза вызывая merge).

После этого вершин в списке корней не больше, чем **D(N) + t(Fib) - 1**,
где t(Fib) - количество корней в списке изначально.



Хочу достать минимальный элемент из пирамиды.
Удаляем корень, разбираемся с детьми (два раза вызывая merge).

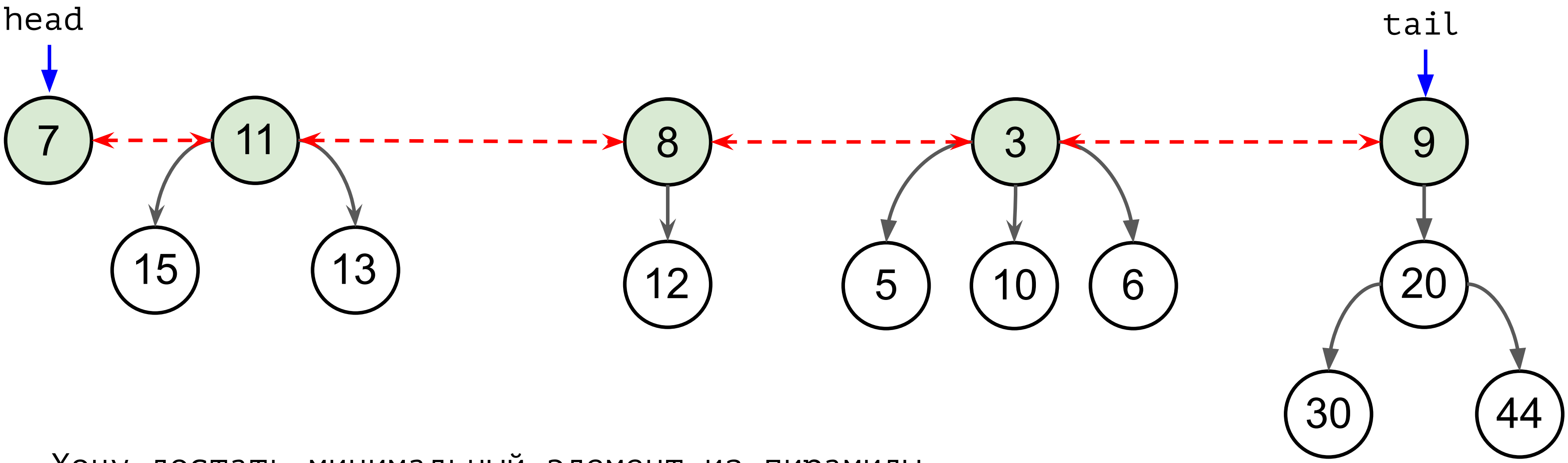
После этого вершин в списке корней не больше, чем **D(N) + t(Fib) - 1**,
где t(Fib) - количество корней в списке изначально.




```
Хочу достать минимальный элемент из пирамиды.
Удаляем корень, разбираемся с детьми (два раза вызывая merge).

После этого вершин в списке корней не больше, чем D(N) + t(Fib) - 1,
где t(Fib) - количество корней в списке изначально.

Заметим, что в процедуре Consolidate принятие решения (и слияние) будет
происходить не более, чем пропорционально количеству элементов в списке
корней. Т.е. процедура закончится, когда закончится, что сливать.
```



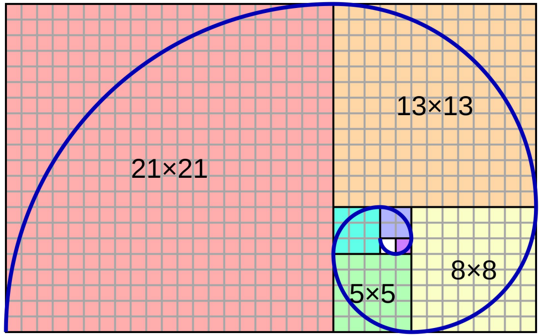
Хочу достать минимальный элемент из пирамиды.
Удаляем корень, разбираемся с детьми (два раза вызывая merge).

После этого вершин в списке корней не больше, чем **D(N) + t(Fib) - 1**, где t(Fib) - количество корней в списке изначально.

Заметим, что в процедуре Consolidate принятие решения (и слияние) будет происходить не более, чем пропорционально количеству элементов в списке корней. Т.е. процедура закончится, когда закончится, что сливать.

Т.е. порядка **D(N) + t(Fib)** раз.

Фибоначчиева пирамида

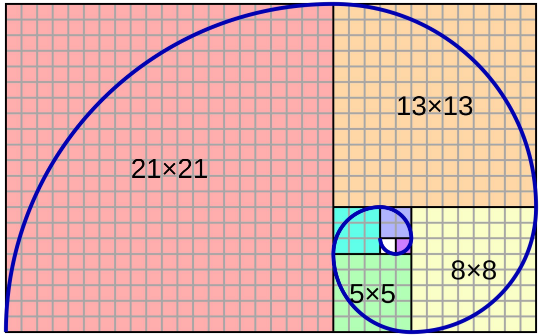


Операции:

```
✓ 1.  insert(value)        -> O(1)
✓ 2.  peek_min()           -> O(1)
? 3.  extract_min()        -> ???
  4.  decrease_key(s, k)
✓ 5.  merge(f1, f2)        -> O(1)
  6.  delete(s)
```

Идея простая: давайте снова пользоваться процедурой merge. выкидываем корень => вмерживаем сыновей => вызываем consolidate. Сложность?

Фибоначчиева пирамида



Операции:

✓ 1. insert(value) -> O(1) Пусть потратит x2 и дает

✓ 2. peek_min() -> O(1)

? 3. extract_min() -> ???

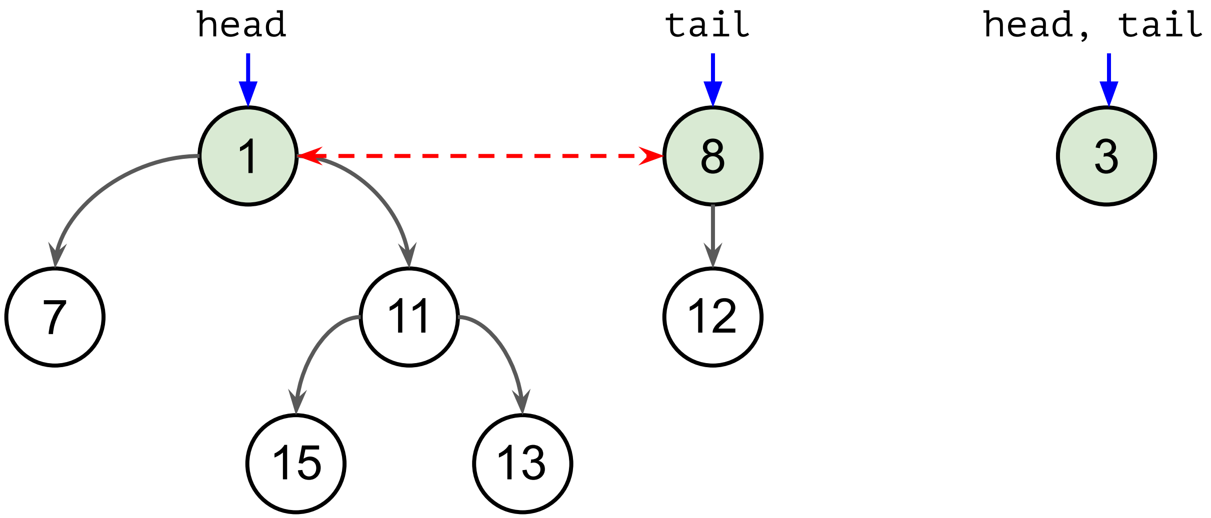
4. decrease_key(s, k)

✓ 5. merge(f1, f2) -> O(1)

6. delete(s)

Идея простая: давайте снова пользоваться процедурой merge.
выкидываем корень => вмерживаем сыновей => вызываем consolidate. Сложность?

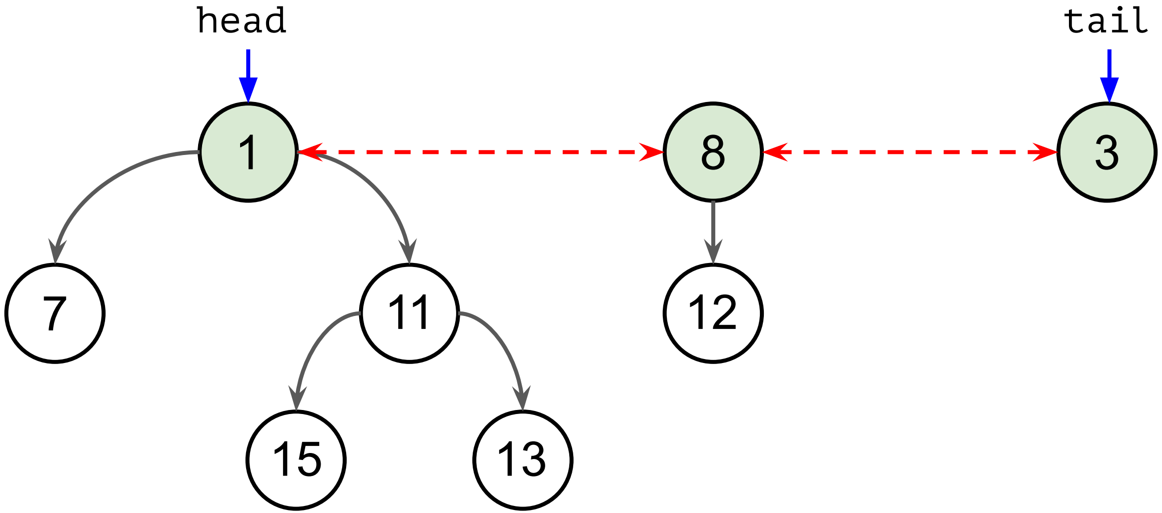
Фибоначчиева пирамида



`Это Фибоначчиева пирамида` `И это Фибоначчиева пирамида`

`insert(3): создаем новую пирамиду из 1 элемента => merge`

Фибоначчиева пирамида

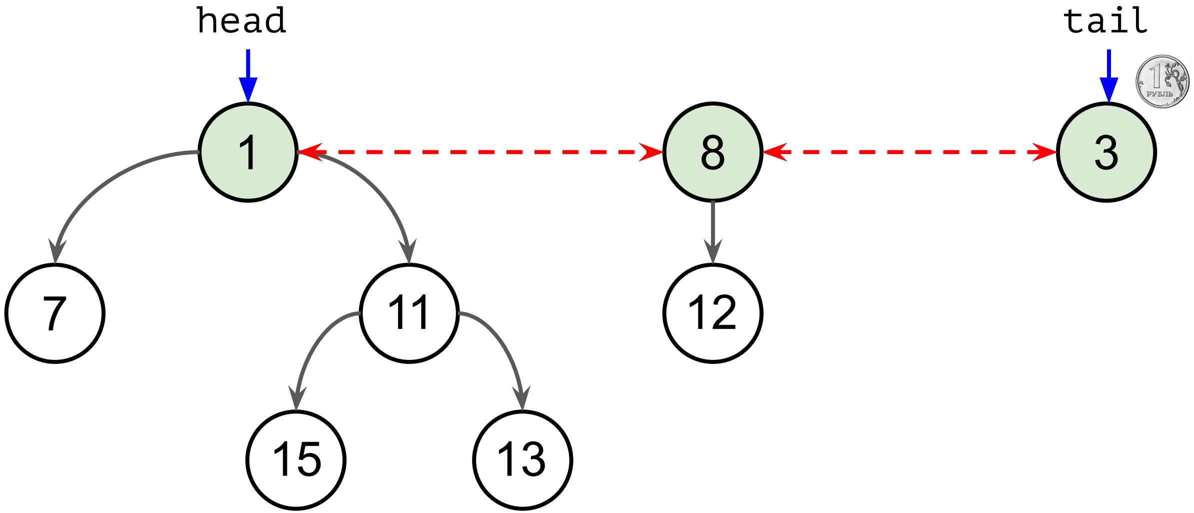


Это Фибоначчиева пирамида И это Фибоначчиева пирамида

insert(3): создаем новую пирамиду из 1 элемента => merge

Снова операция за O(1)

Фибоначчиева пирамида

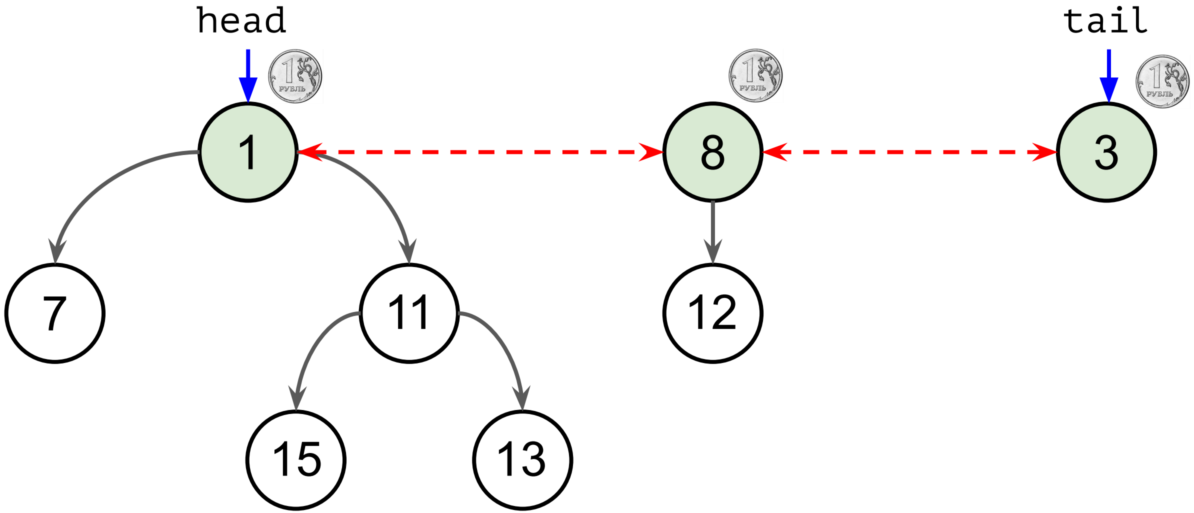


Это Фибоначчиева пирамида   И это Фибоначчиева пирамида

insert(3): создаем новую пирамиду из 1 элемента => merge

Снова операция за O(1) и плюс монетку, не жалко ведь!

Фибоначчиева пирамида

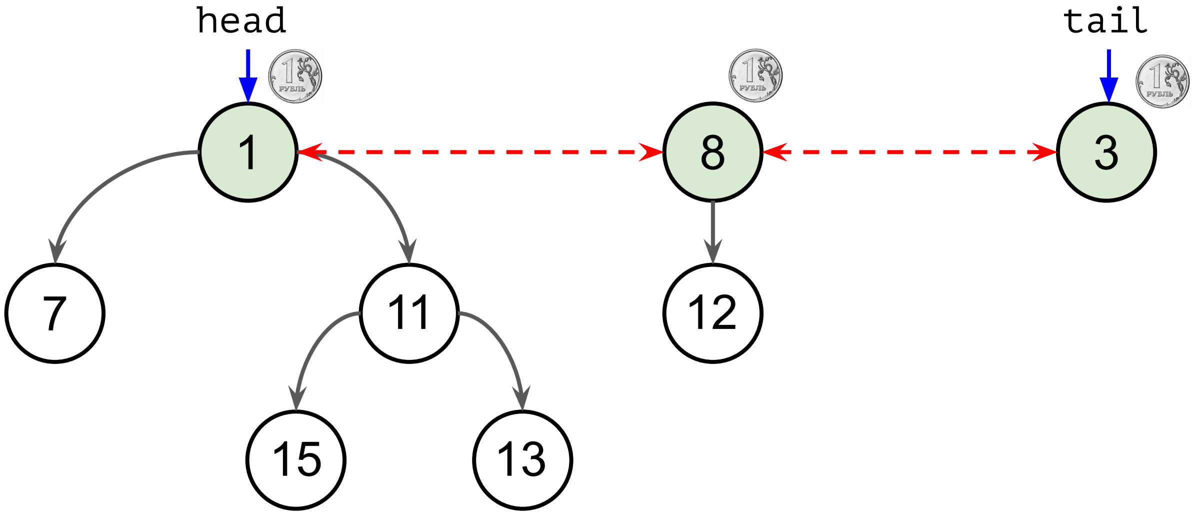


Это Фибоначчиева пирамида   И это Фибоначчиева пирамида

insert(3): создаем новую пирамиду из 1 элемента => merge

Снова операция за O(1) и плюс монетку, не жалко ведь!

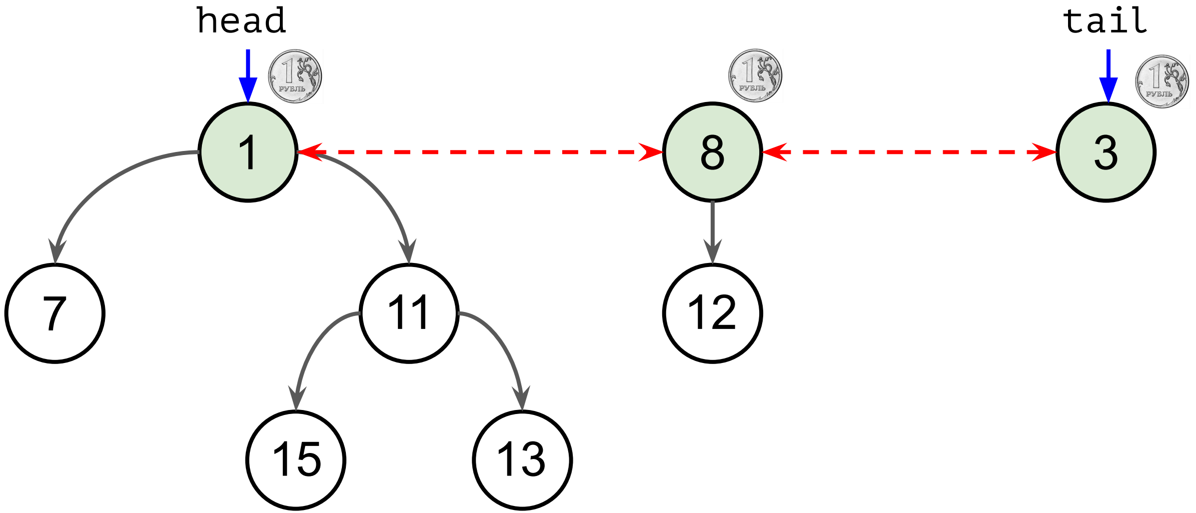
Фибоначчиева пирамида



Снова операция за O(1) и плюс монетку, не жалко ведь!

Откуда еще могут появиться корни?

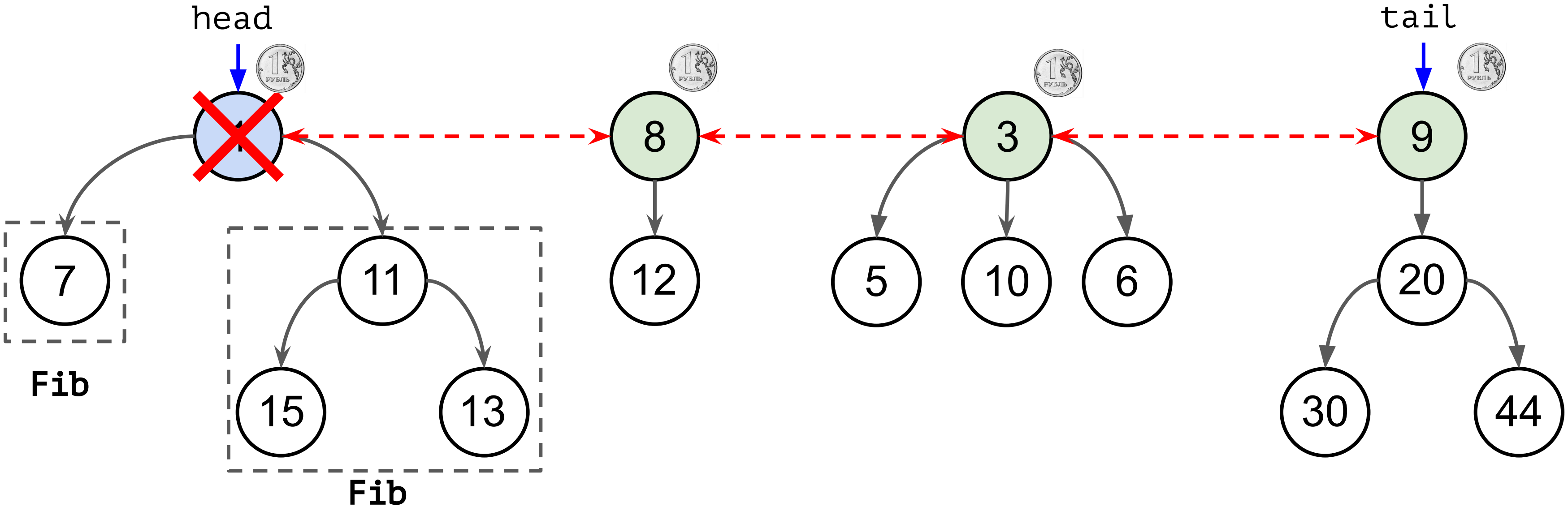
Фибоначчиева пирамида



Снова операция за O(1) и плюс монетку, не жалко ведь!

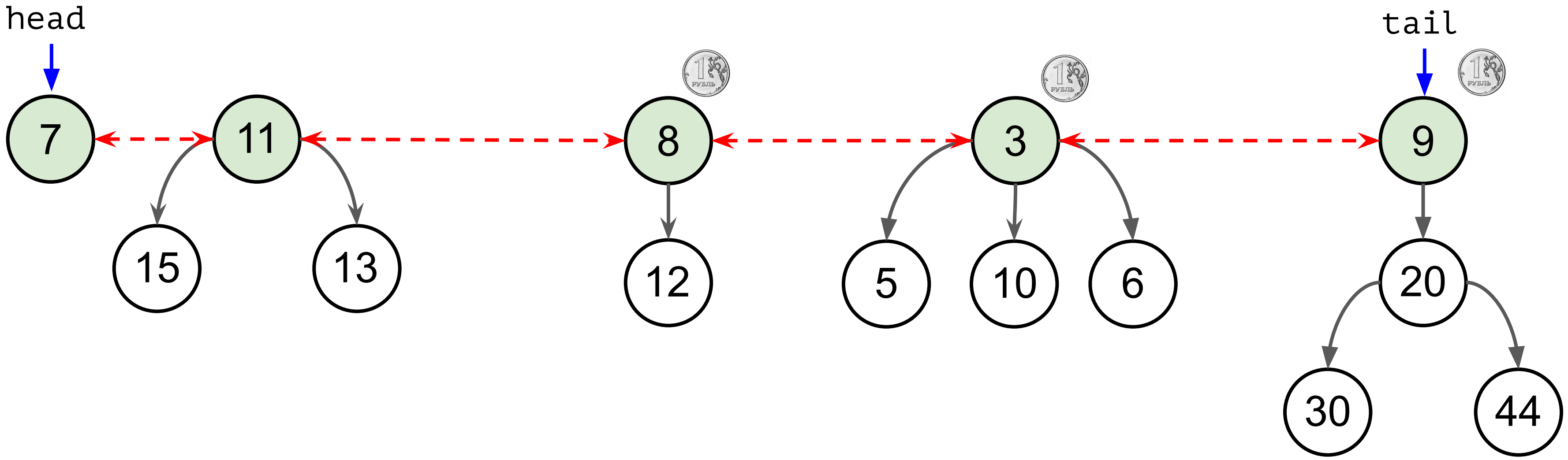
Откуда еще могут появиться корни? Из extract_min.

Фибоначчиева пирамида



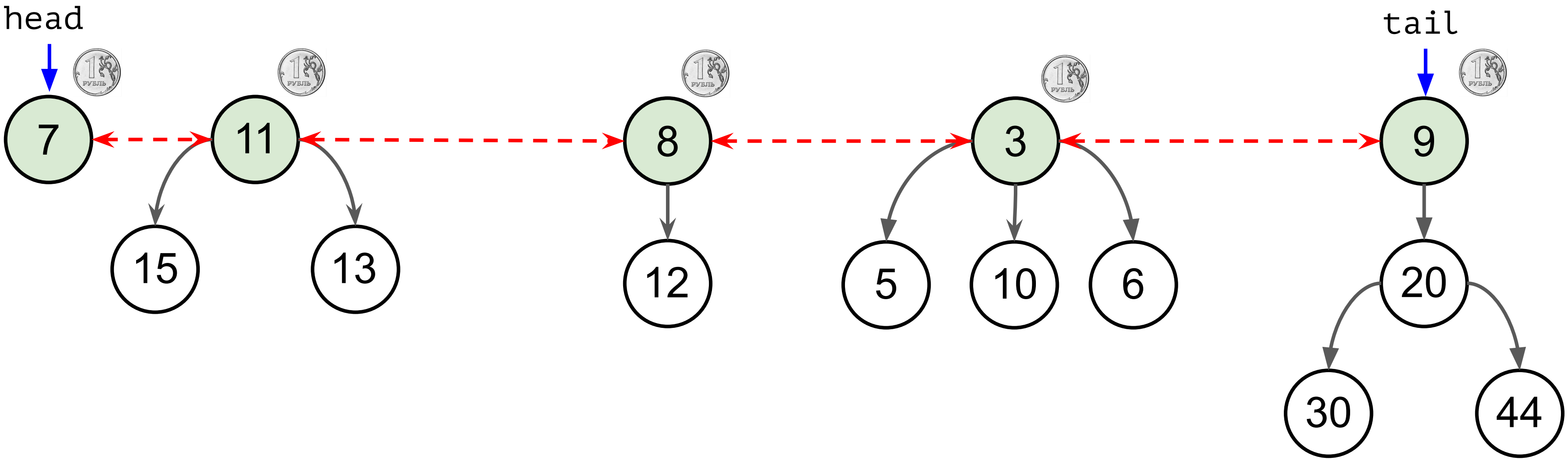
Хочу достать минимальный элемент из пирамиды.
Удаляем корень, разбираемся с детьми.

Фибоначчиева пирамида



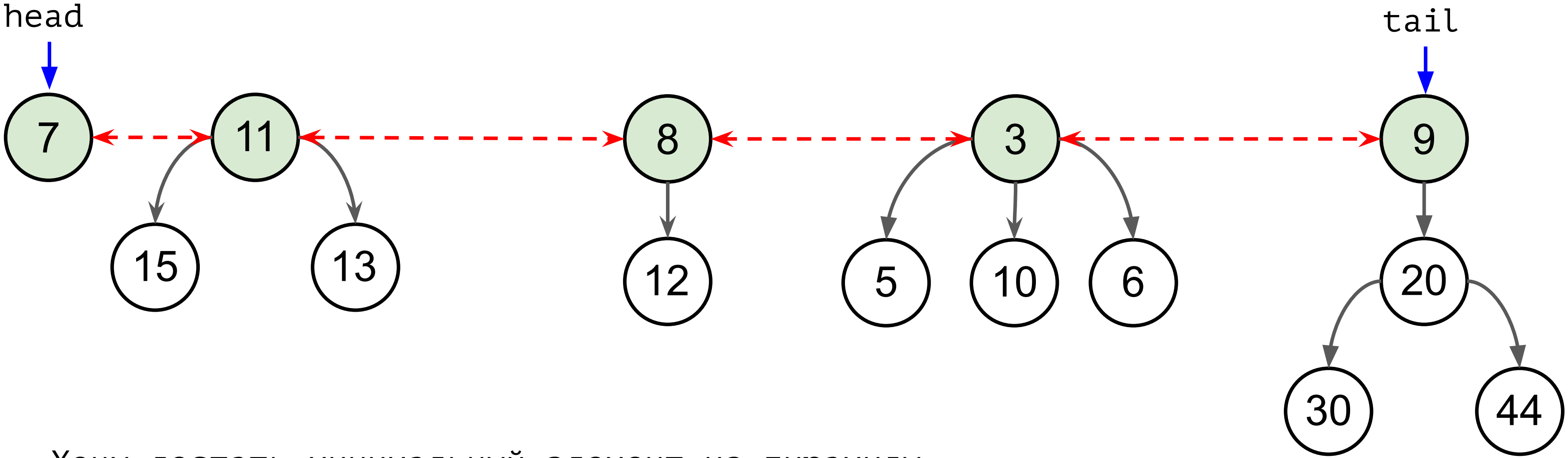

```
Хочу достать минимальный элемент из пирамиды.
Удаляем корень, разбираемся с детьми (два раза вызывая merge).
```

Фибоначчиева пирамида



Хочу достать минимальный элемент из пирамиды.
Удаляем корень, разбираемся с детьми (два раза вызывая merge).

Мы здесь в любом случае тратим O(D(N)) времени, чтобы добавить новые корни, давайте же и D(N) монеток тоже положим.

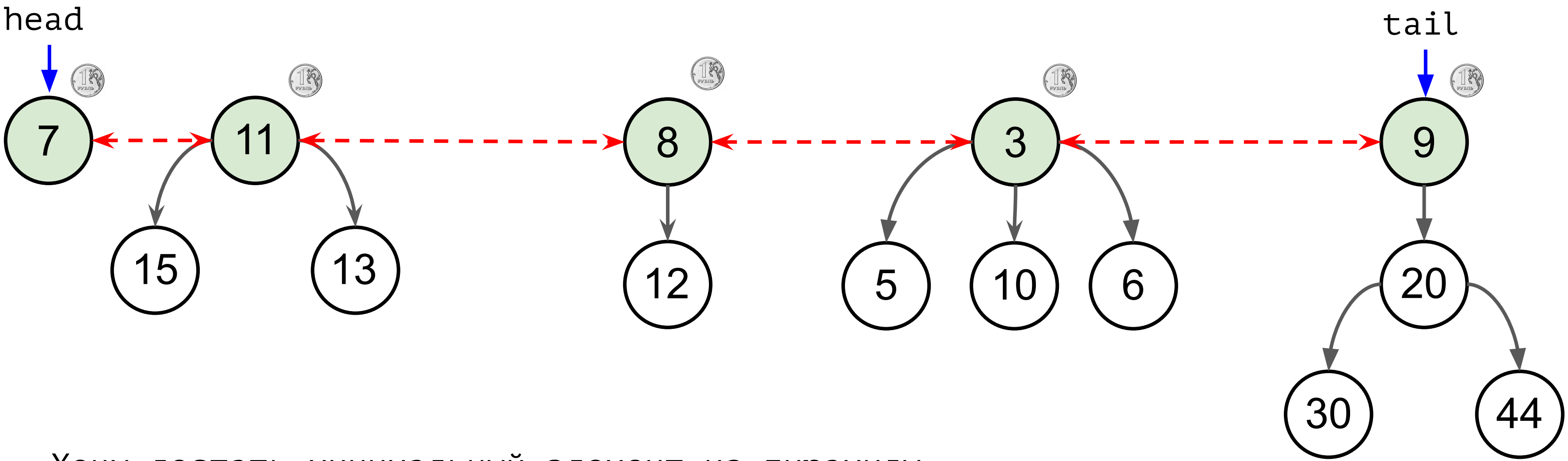


Хочу достать минимальный элемент из пирамиды.
Удаляем корень, разбираемся с детьми (два раза вызывая merge).

После этого вершин в списке корней не больше, чем **D(N) + t(Fib) - 1**, где t(Fib) - количество корней в списке изначально.

Заметим, что в процедуре Consolidate принятие решение (и слияние) будет происходить не больше раз, чем количество элементов в списке корней.

Т.е. порядка **D(N) + t(Fib)** раз.

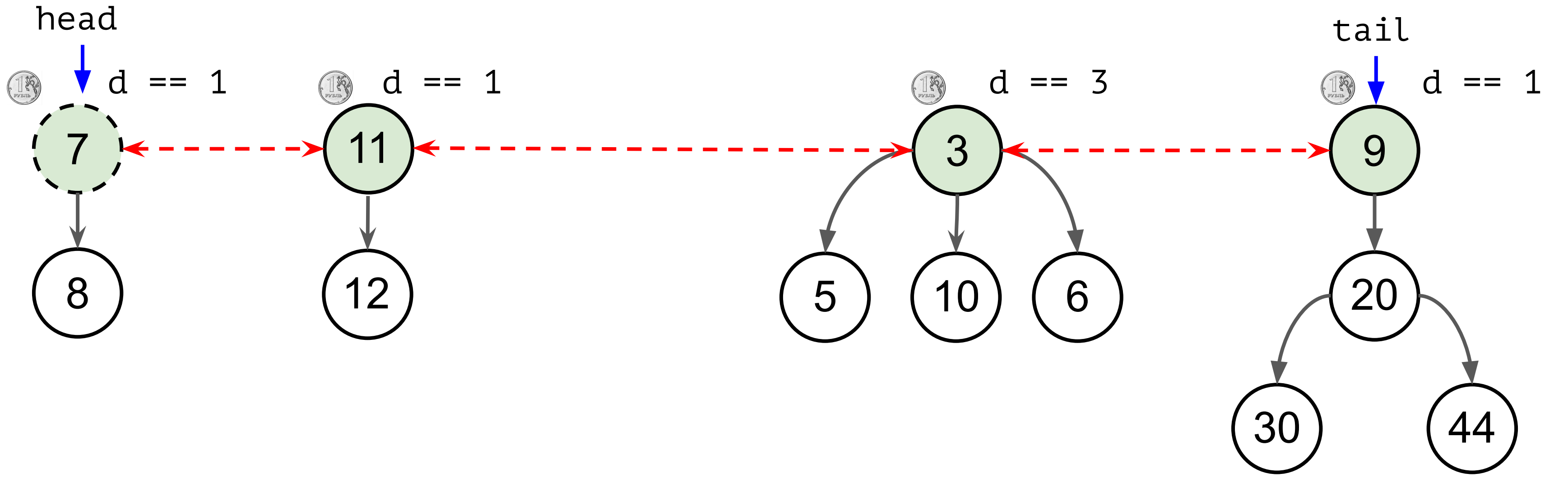


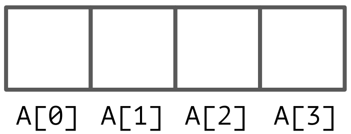
Хочу достать минимальный элемент из пирамиды.
Удаляем корень, разбираемся с детьми (два раза вызывая merge).

После этого вершин в списке корней не больше, чем **D(N) + t(Fib) - 1**,
где t(Fib) - количество корней в списке изначально.

Заметим, что в процедуре Consolidate принятие решение (и слияние) будет
происходить не больше раз, чем количество элементов в списке корней.

Т.е. порядка **D(N) + t(Fib)** раз. Но на всех корнях точно лежат **монетки**.





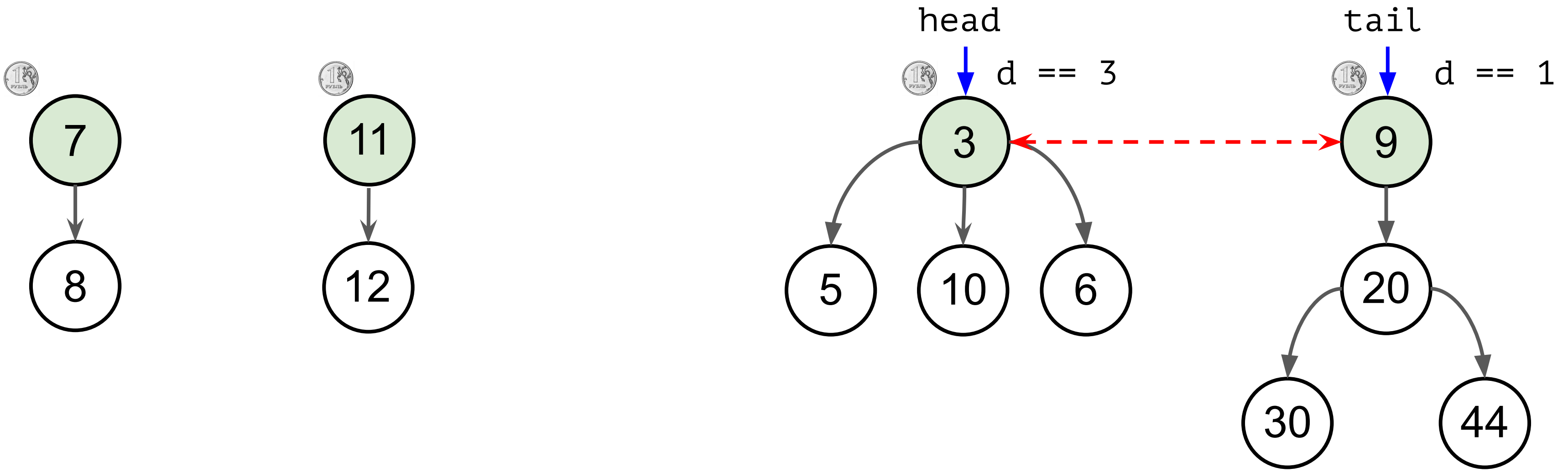
Тогда: **1)** удаляем указатель из A

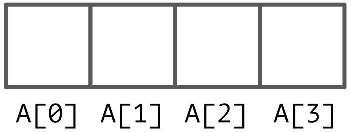
2) удаляем корни совпавших поддеревья из списка корней

3) подвешиваем дерево с большим корнем к дереву с меньшим

4) возвращаете получившийся корень в список корней

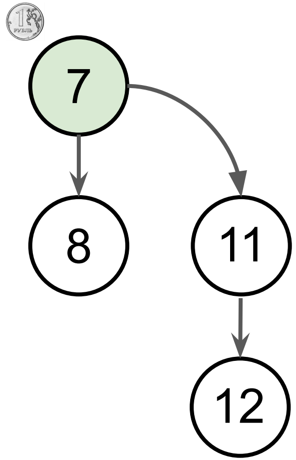
5) пробуете снова добавить в A!





Тогда: 1) удаляем указатель из A
2) удаляем корни совпавших поддеревья из списка корней
3) подвешиваем дерево с большим корнем к дереву с меньшим
4) возвращаете получившийся корень в список корней

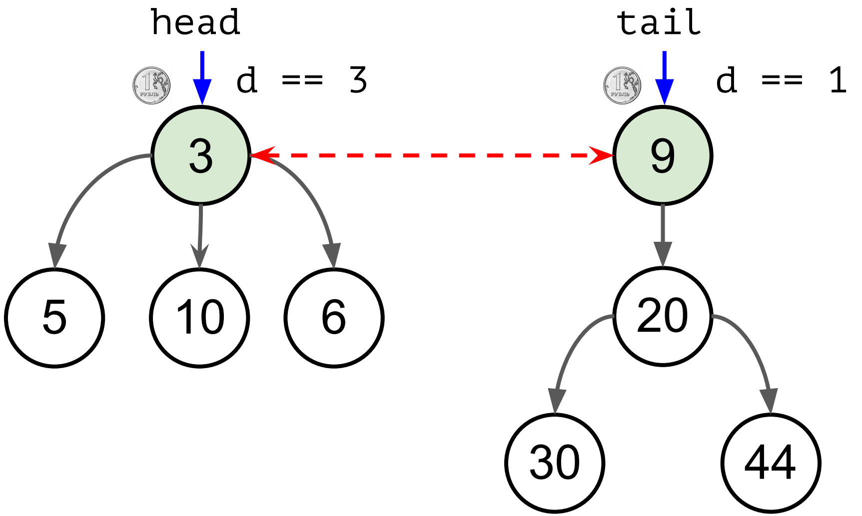
5) пробуете снова добавить в A!

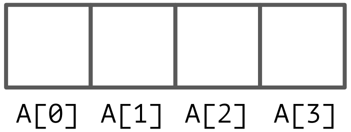


Подвешивание - операция за O(1).

Одну монетку потратим на нее.

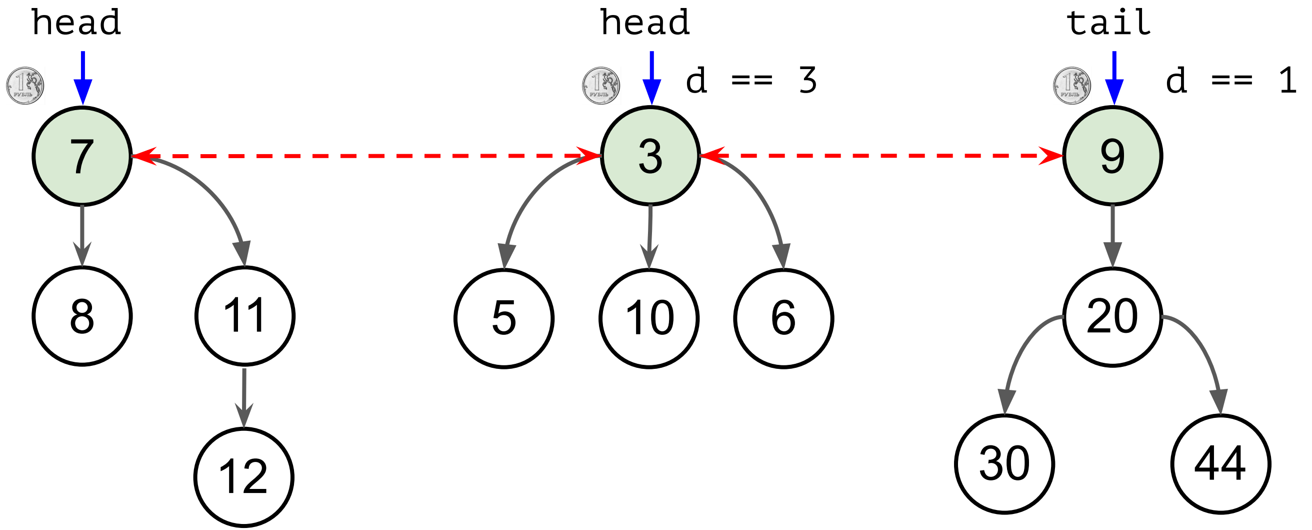
Вторую - оставим в корне нового дерева.



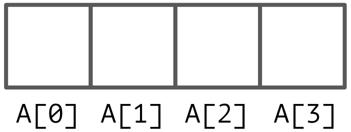


Тогда:
1) удаляем указатель из A
2) удаляем корни совпавших поддеревья из списка корней
3) подвешиваем дерево с большим корнем к дереву с меньшим
4) возвращаете получившийся корень в список корней

5) пробуете снова добавить в A!

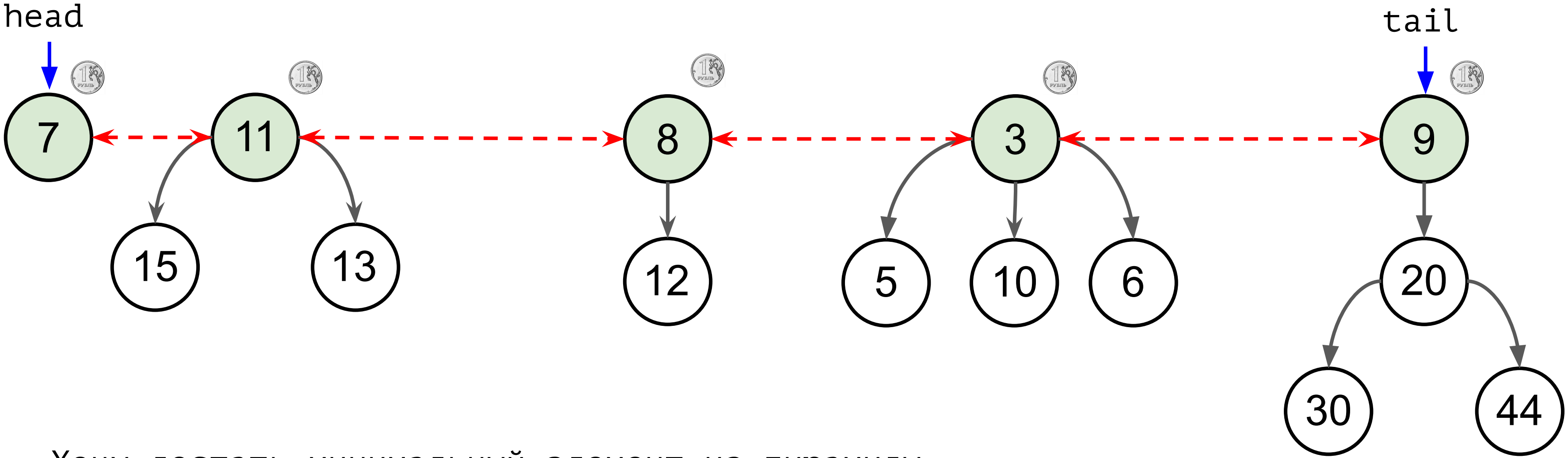


В каждом корне осталось по монетке!



Тогда:
1) удаляем указатель из A
2) удаляем корни совпавших поддеревья из списка корней
3) подвешиваем дерево с большим корнем к дереву с меньшим
4) возвращаете получившийся корень в список корней

5) пробуете снова добавить в A!



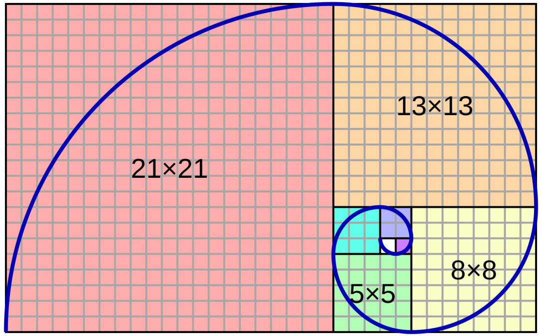
Хочу достать минимальный элемент из пирамиды.
Удаляем корень, разбираемся с детьми (два раза вызывая merge).

После этого вершин в списке корней не больше, чем **D(N) + t(Fib) - 1**,
где t(Fib) - количество корней в списке изначально.

Заметим, что в процедуре Consolidate принятие решение (и слияние) будет
происходить не больше раз, чем количество элементов в списке корней.

Т.е. порядка **D(N) + t(Fib)** раз. Но на всех корнях точно лежат **монетки**.
Потратим их за счет долга от t(Fib) и получим, что сложность **O*(D(N))**.

Фибоначчиева пирамида

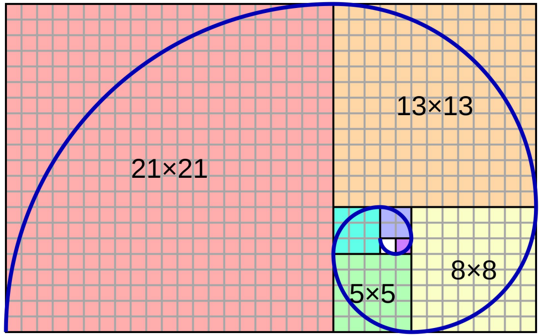


Операции:

```
✓ 1. insert(value)         -> O(1)  Пусть потратит x2 и дает
✓ 2. peek_min()            -> O(1)
? 3. extract_min()         -> ???
  4. decrease_key(s, k)
✓ 5. merge(f1, f2)         -> O(1)
  6. delete(s)
```

Идея простая: давайте снова пользоваться процедурой merge.
выкидываем корень => вмерживаем сыновей => вызываем consolidate. Сложность?

Фибоначчиева пирамида

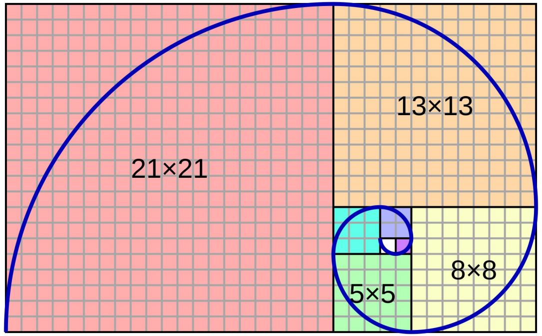


Операции:

```
✓   1. insert(value)        -> O(1)  Пусть потратит x2 и дает
✓   2. peek_min()           -> O(1)
??? 3. extract_min()        -> O*(D(N))
    4. decrease_key(s, k)
✓   5. merge(f1, f2)        -> O(1)
    6. delete(s)
```

Идея простая: давайте снова пользоваться процедурой merge. выкидываем корень => вмерживаем сыновей => вызываем consolidate. Сложность: O*(D(N))

Фибоначчиева пирамида



Операции:

✓ 1. insert(value) -> O(1) Пусть потратит x2 и дает

✓ 2. peek_min() -> O(1)

??? 3. extract_min() -> O*(D(N))

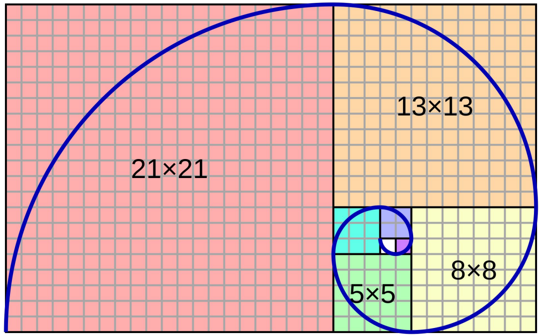
? 4. decrease_key(s, k)

✓ 5. merge(f1, f2) -> O(1)

6. delete(s)

Идея простая: давайте снова пользоваться процедурой merge. выкидываем корень => вмерживаем сыновей => вызываем consolidate. Сложность: O*(D(N))

Фибоначчиева пирамида



Операции:

✓ 1. insert(value) -> O(1) Пусть потратит x2 и дает

✓ 2. peek_min() -> O(1)

??? 3. extract_min() -> O*(D(N))

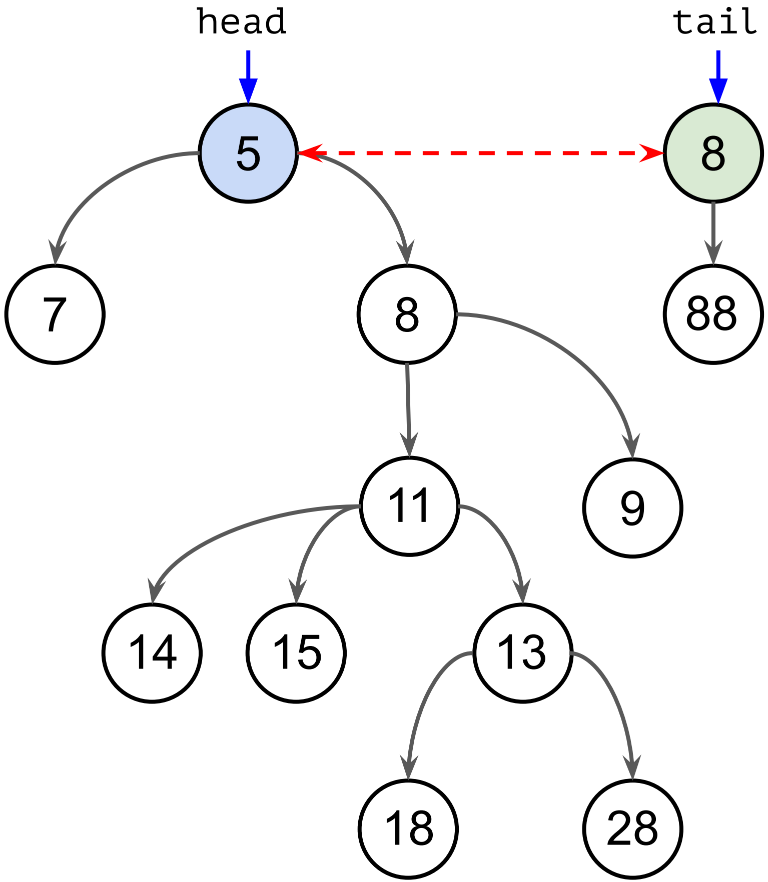
? 4. decrease_key(s, k)

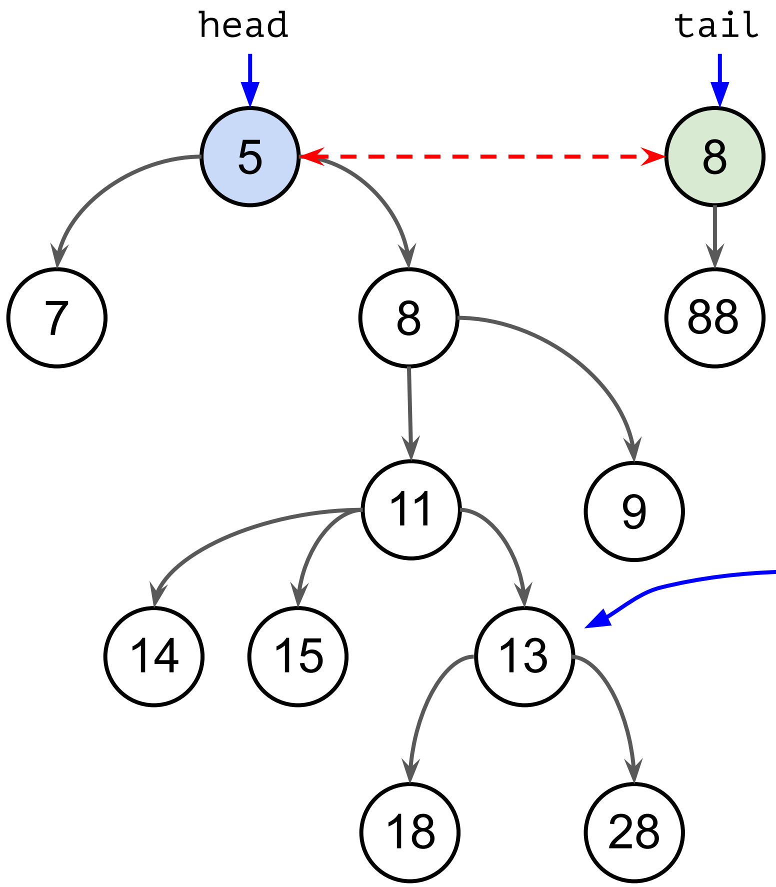
✓ 5. merge(f1, f2) -> O(1)

6. delete(s)

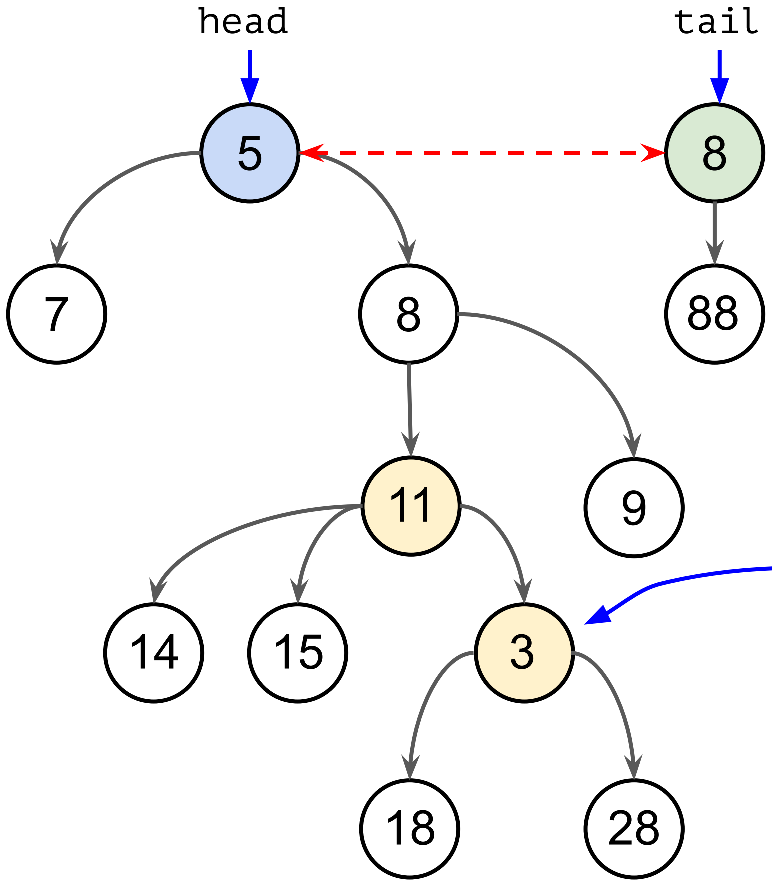
Идея: 1) пометки на вершинах

2) отсаживание поддеревьев при нарушении свойств пирамиды


head
tail
5
8
7
8
88
11
9
14
15
13
18
28

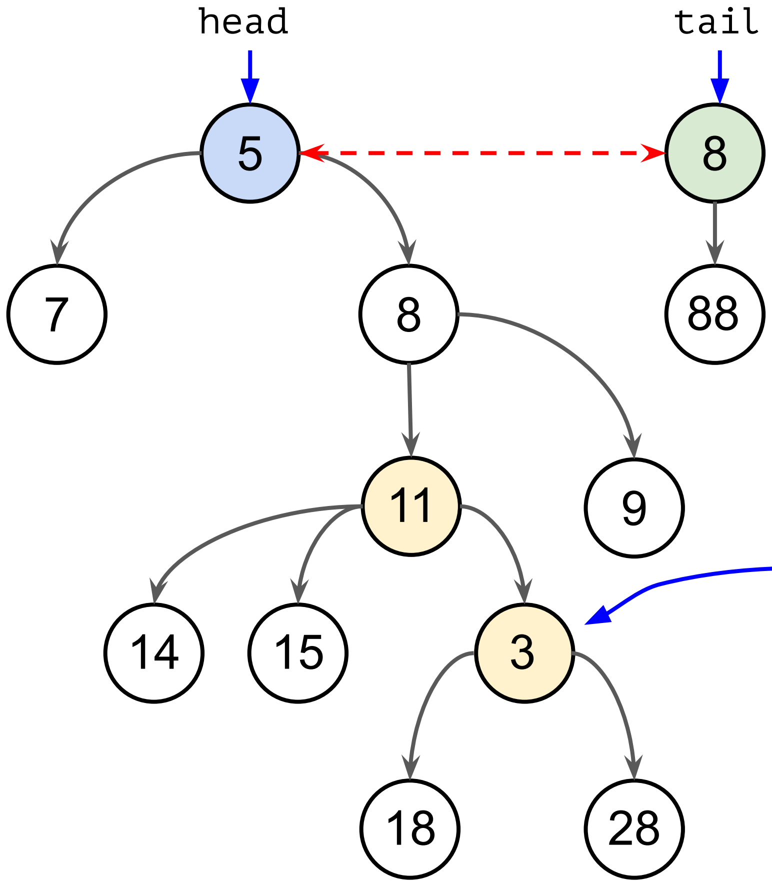


Пусть пришел запрос: decrease_key(s, 3)



Пусть пришел запрос: `decrease_key(s, 3)`

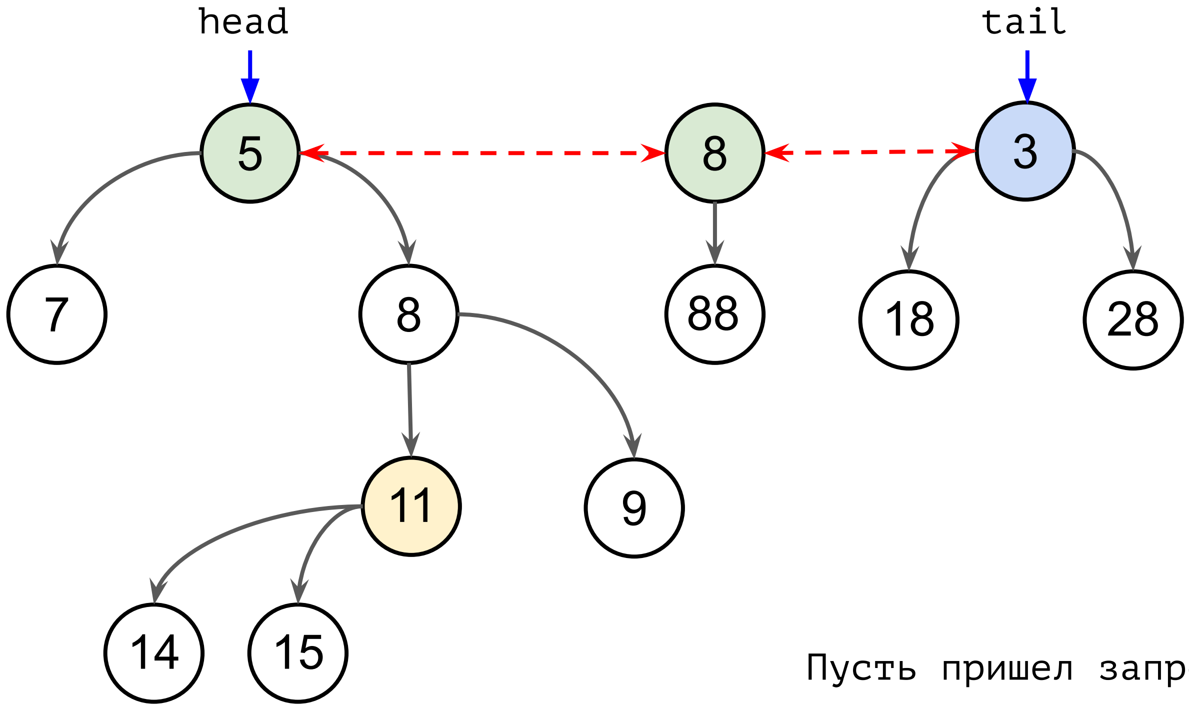
Если получилось нарушение свойства неубывающей пирамиды, то нужно что-то менять.



Пусть пришел запрос: decrease_key(s, 3)

Если получилось нарушение свойства неубывающей пирамиды, то нужно что-то менять.

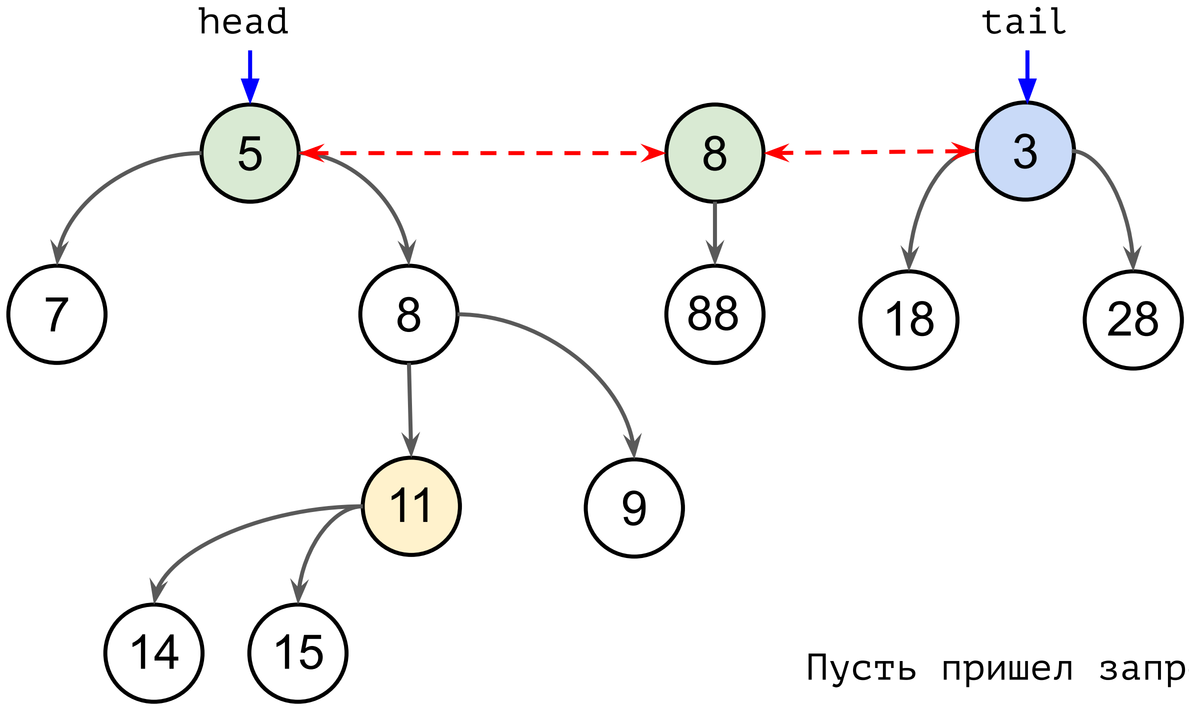
Но вместо просеивания - отсадим поддерево!



Пусть пришел запрос: decrease_key(s, 3)

Если получилось нарушение свойства неубывающей пирамиды, то нужно что-то менять.

Но вместо просеивания - отсадим поддерево!



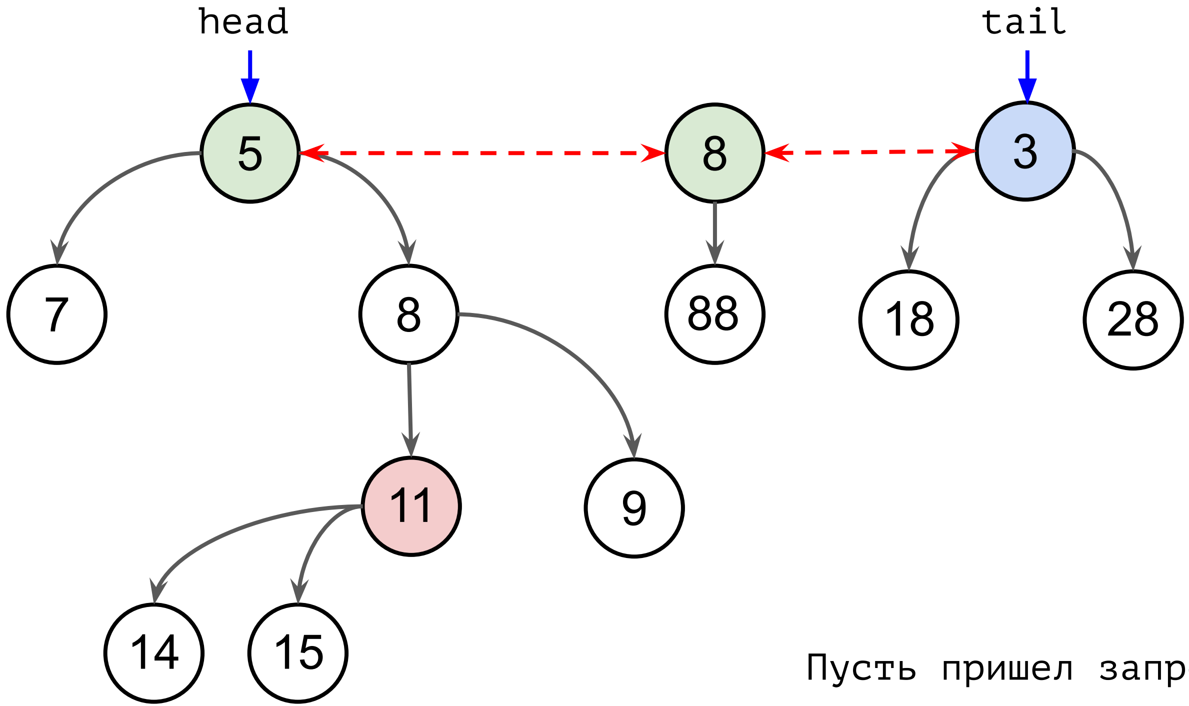
Действительно, это поддерево все еще удовлетворяет свойству неубывающей пирамиды, ведь ключ только уменьшался.

Могут появиться деревья одной степени, но и ладно! extract_min потом все исправит.

Пусть пришел запрос: decrease_key(s, 3)

Если получилось нарушение свойства неубывающей пирамиды, то нужно что-то менять.

Но вместо просеивания - отсадим поддерево!



Действительно, это поддерево все еще удовлетворяет свойству неубывающей пирамиды, ведь ключ только уменьшался.

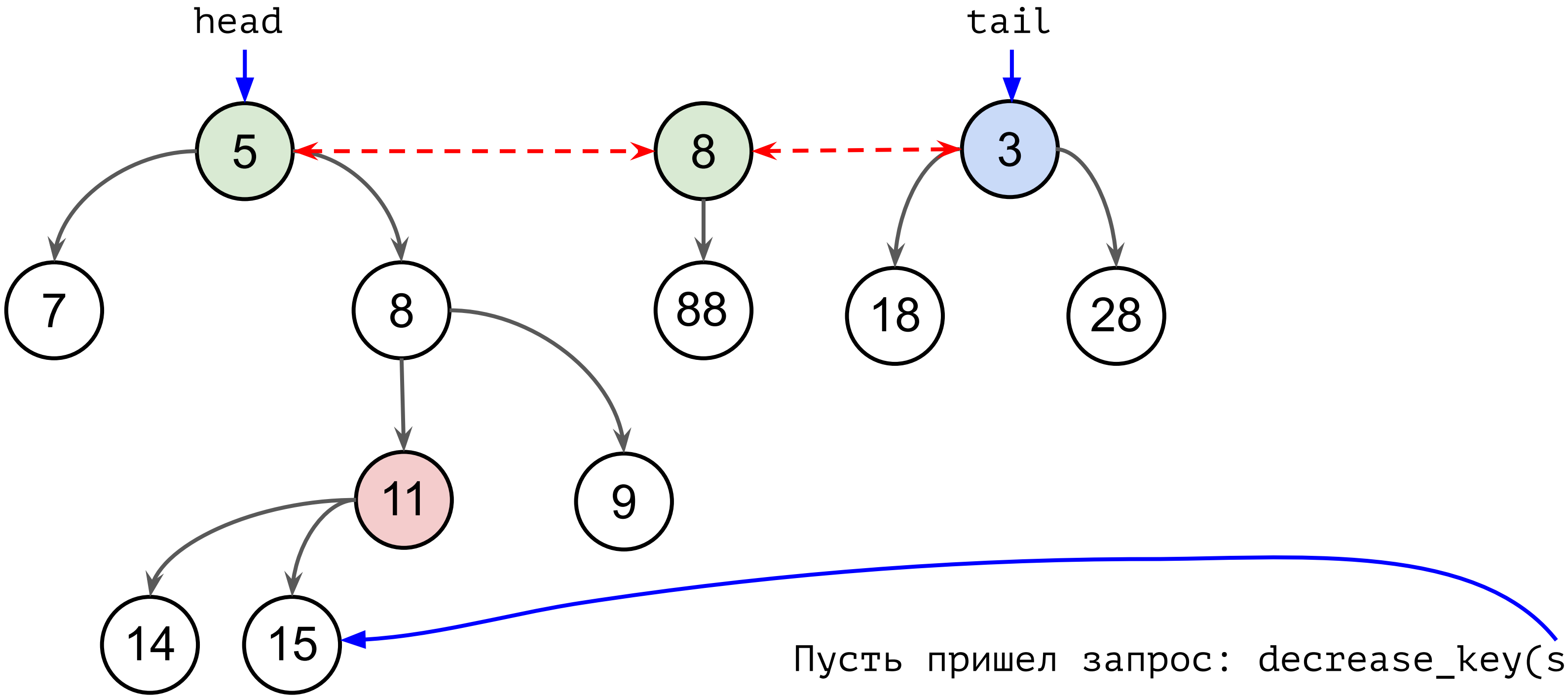
Могут появиться деревья одной степени, но и ладно! extract_min потом все исправит.

Пусть пришел запрос: decrease_key(s, 3)

Если получилось нарушение свойства неубывающей пирамиды, то нужно что-то менять.

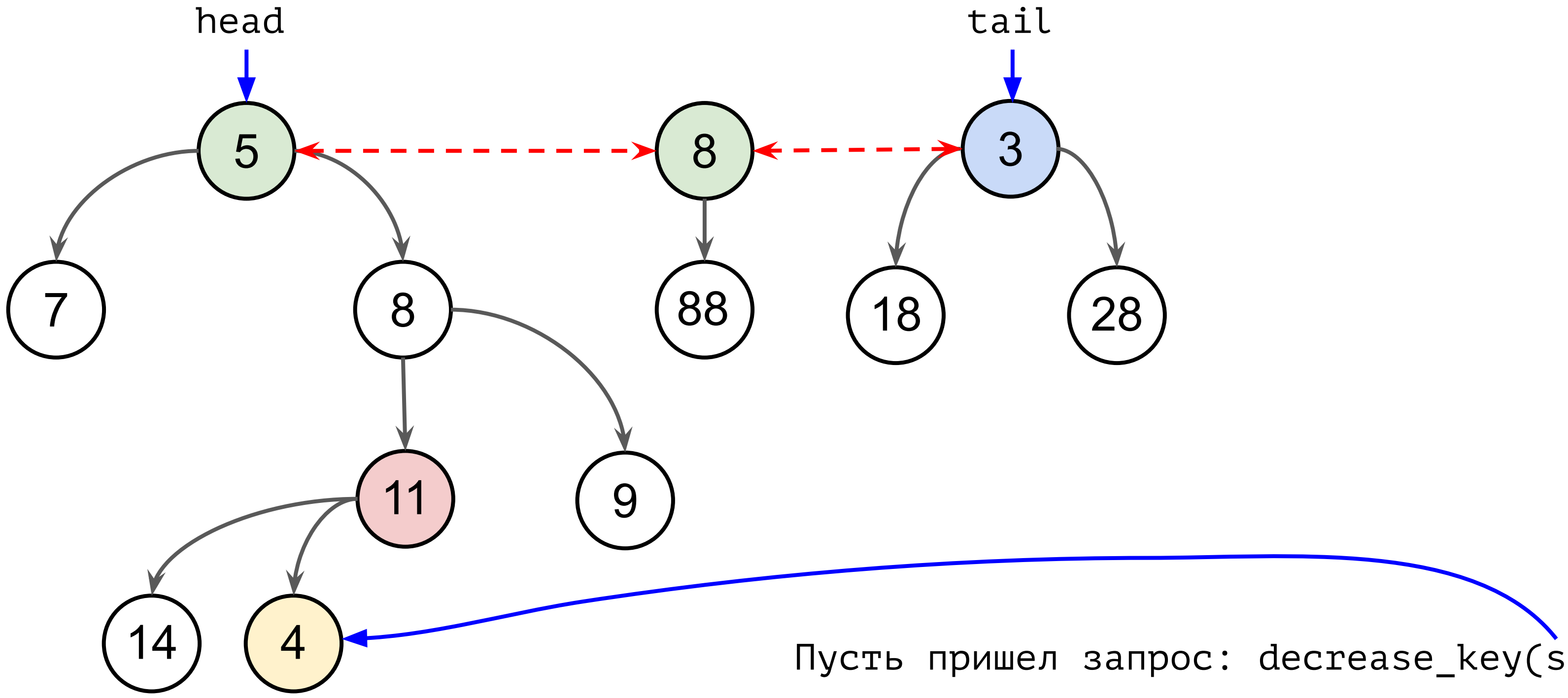
Однако, оставим пометку на вершине, от которой отсадили ребенка

Но вместо просеивания - отсадим поддерево!



Пусть пришел запрос: decrease_key(s, 4)

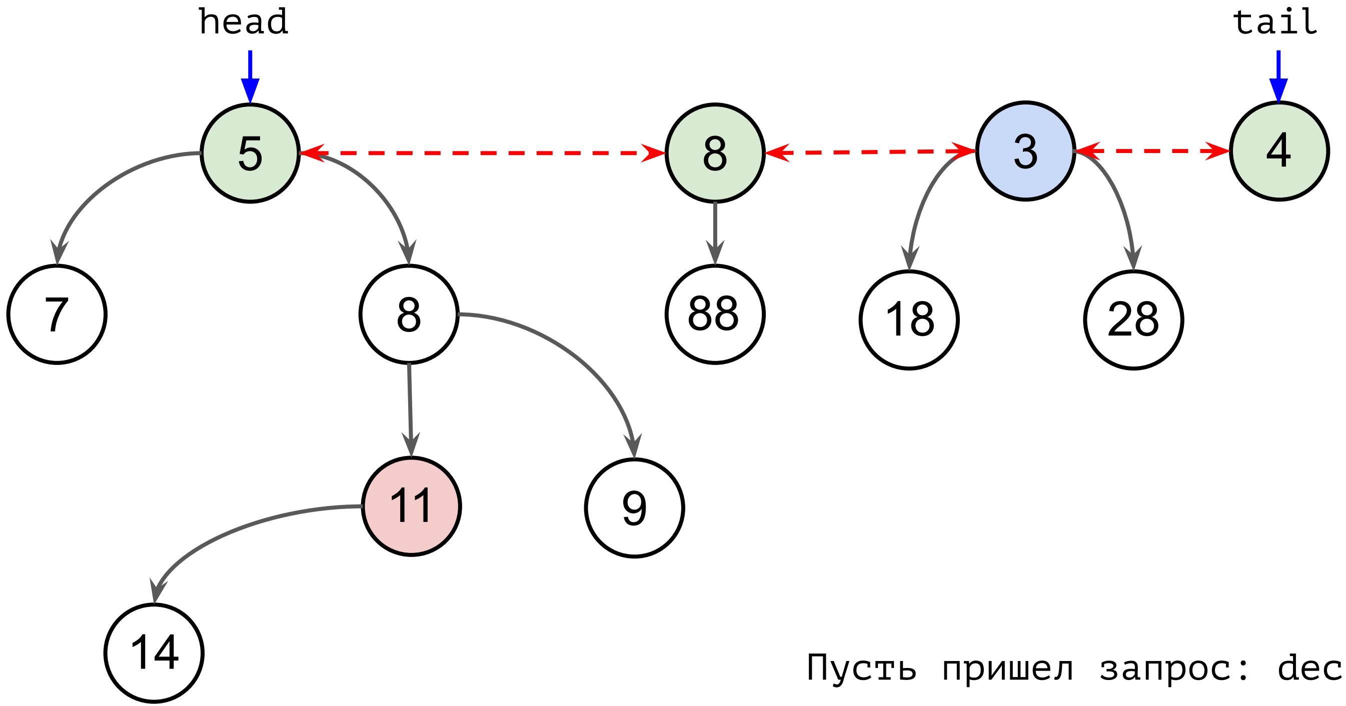
Однако, оставим пометку
на вершине, от которой
отсадили ребенка



Пусть пришел запрос: decrease_key(s, 4)

Во-первых, повторим те же действия.

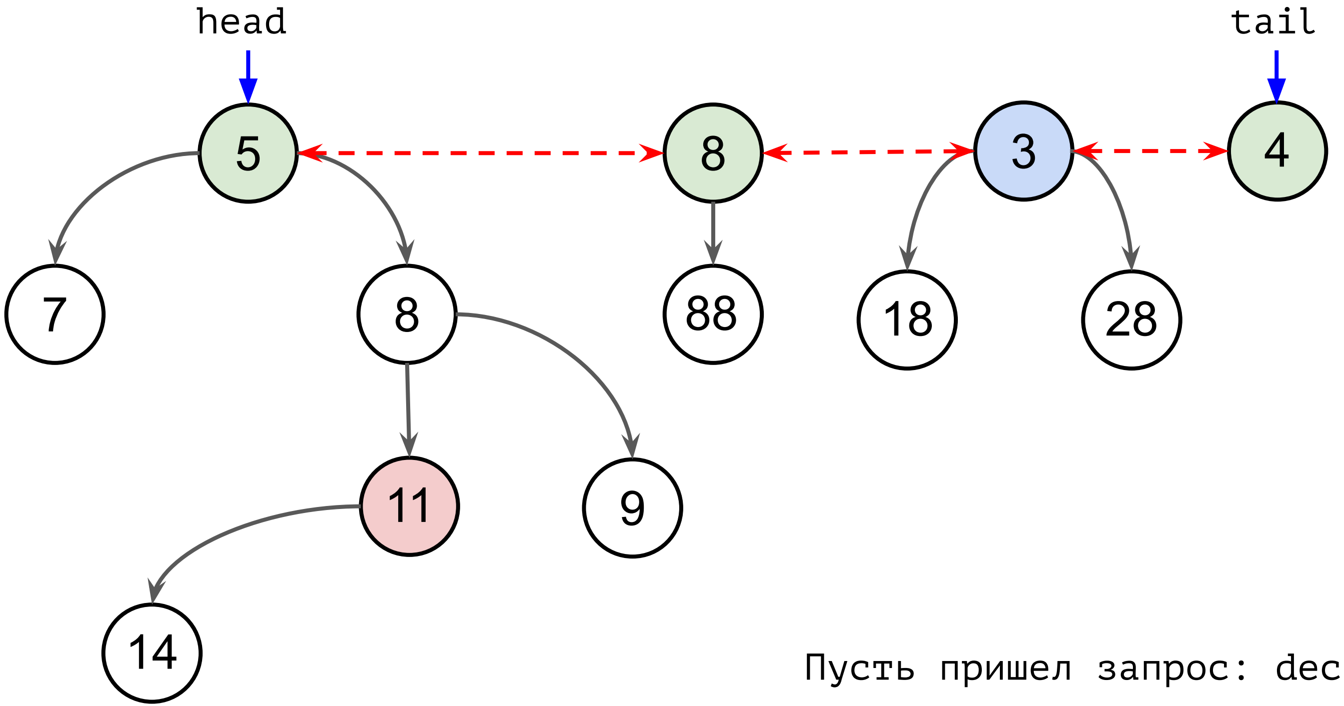
Однако, оставим пометку
на вершине, от которой
отсадили ребенка



Пусть пришел запрос: decrease_key(s, 4)

Во-первых, повторим те же действия.

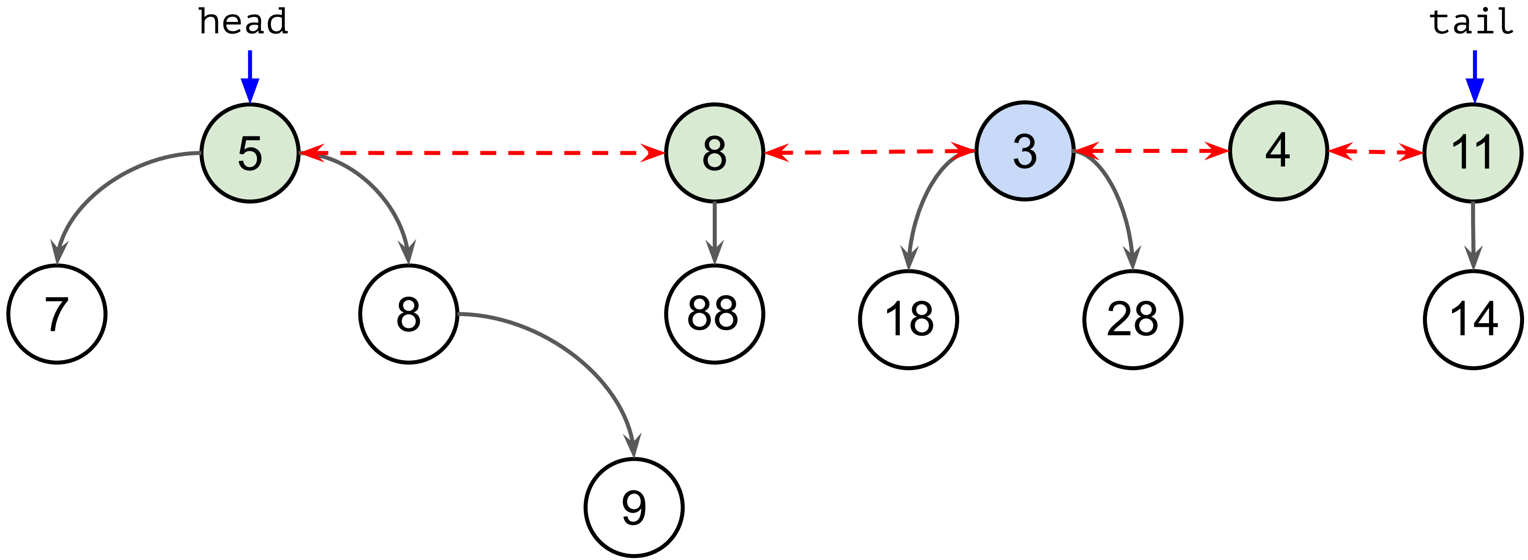
Однако, оставим пометку
на вершине, от которой
отсадили ребенка



Пусть пришел запрос: decrease_key(s, 4)

Во-первых, повторим те же действия.
Во-вторых, отсадим и само дерево, т.к. корень помечен!

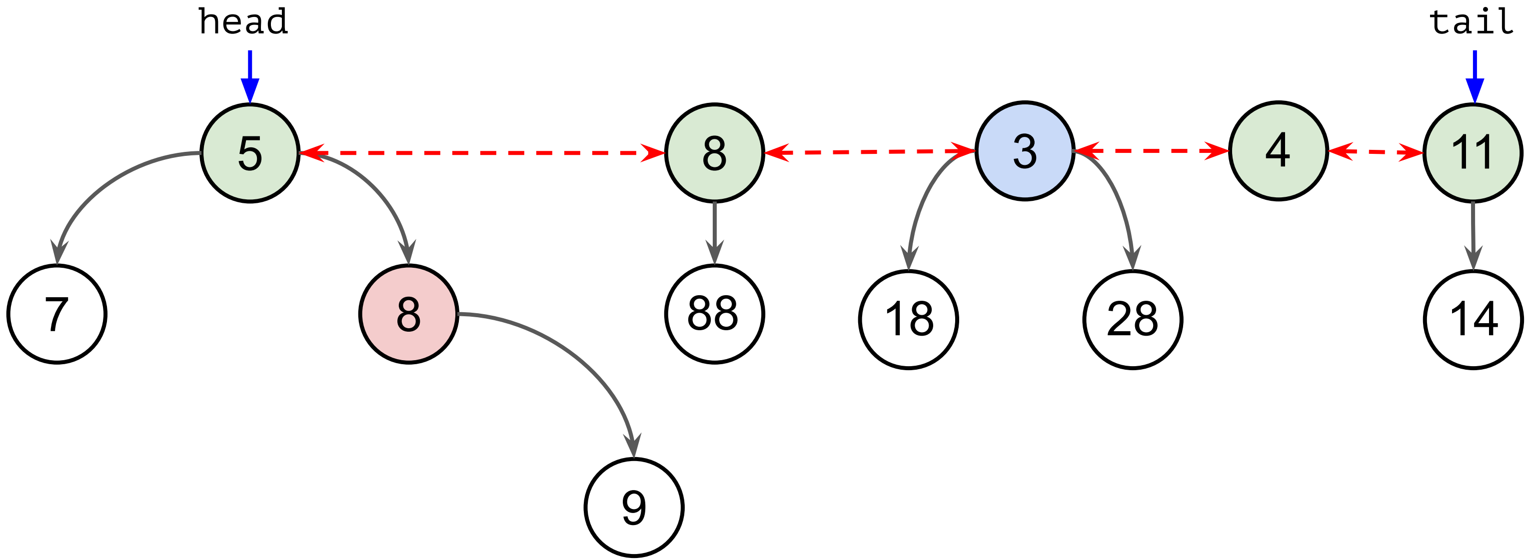
Однако, оставим пометку на вершине, от которой отсадили ребенка



Пусть пришел запрос: decrease_key(s, 4)

Во-первых, повторим те же действия.
Во-вторых, отсадим и само дерево, т.к. корень помечен!

Однако, оставим пометку на вершине, от которой отсадили ребенка

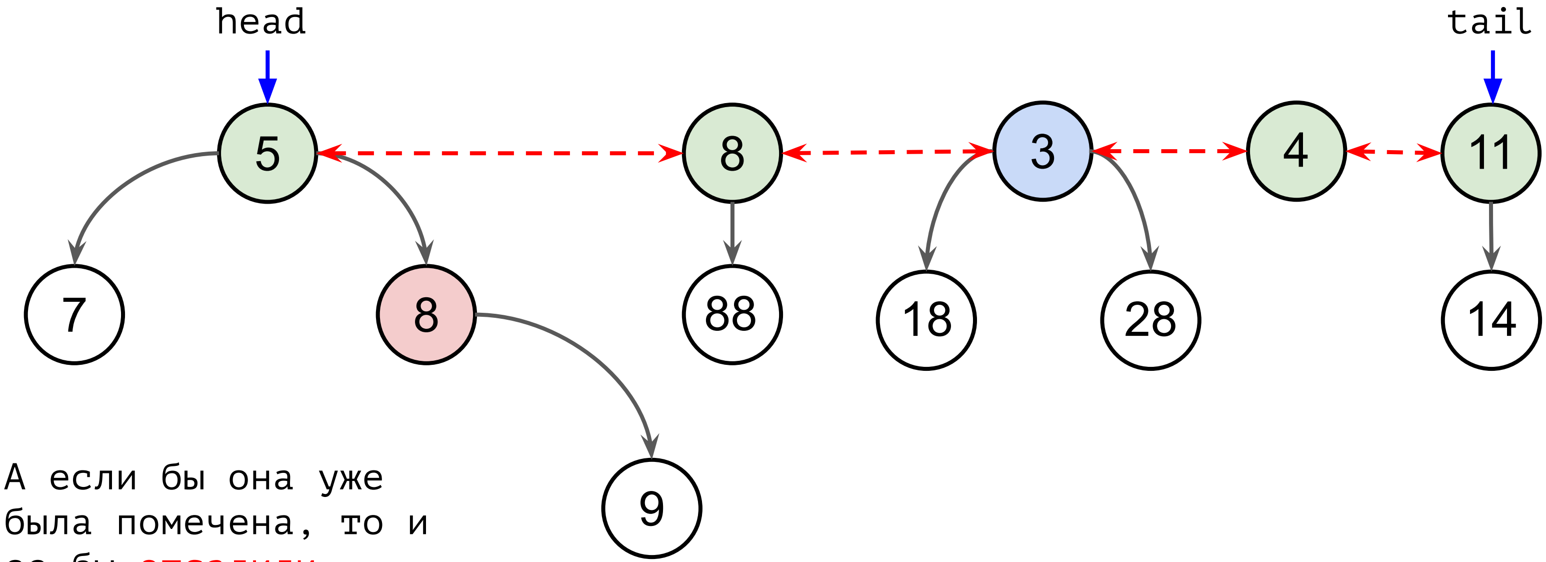


Однако, оставим пометку на вершине, от которой отсадили ребенка

Пусть пришел запрос: decrease_key(s, 4)

Во-первых, повторим те же действия.
Во-вторых, отсадим и само дерево, т.к. корень помечен!

Не забываем пометить вершину, откуда забрали.



А если бы она уже была помечена, то и ее бы отсадили.

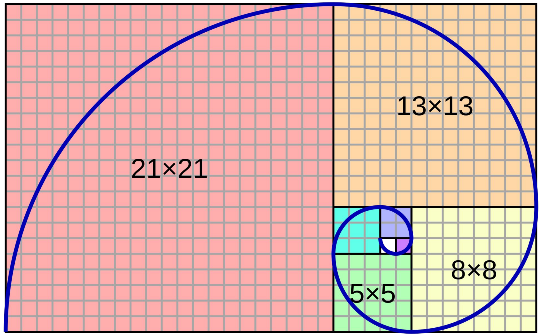
Пусть пришел запрос: decrease_key(s, 4)

Во-первых, повторим те же действия.
Во-вторых, отсадим и само дерево, т.к. корень помечен!

Однако, оставим пометку на вершине, от которой отсадили ребенка

Не забываем пометить вершину, откуда забрали.

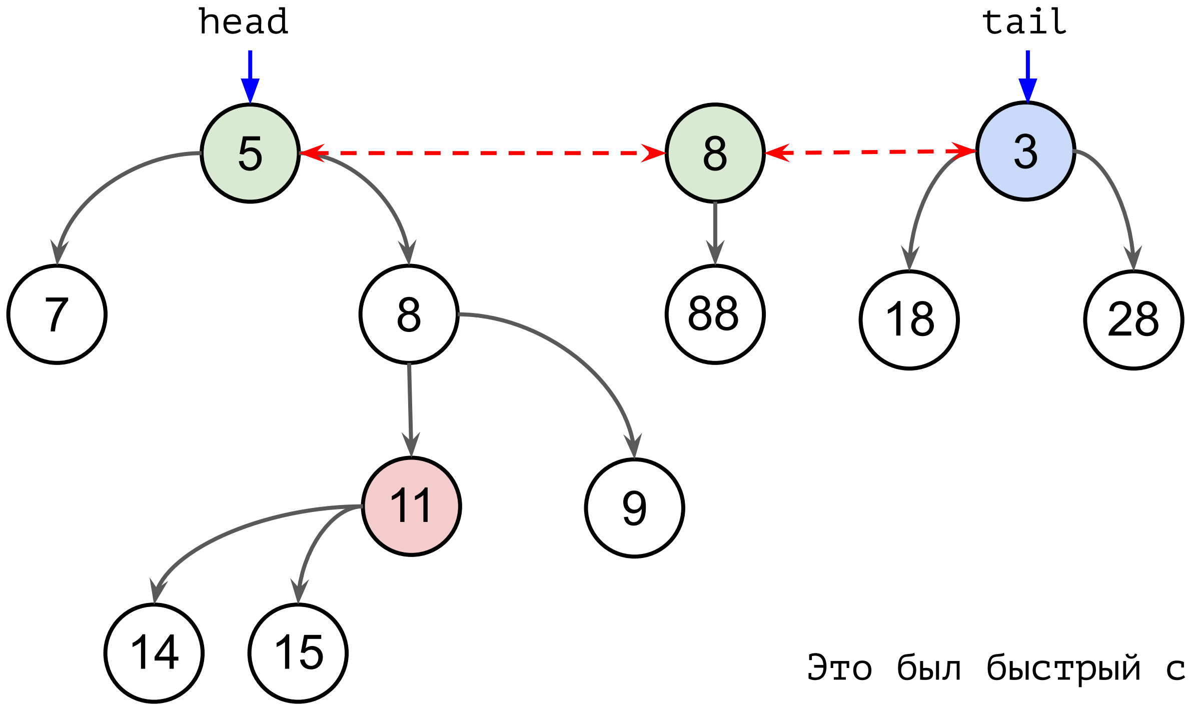
Фибоначчиева пирамида



Операции:

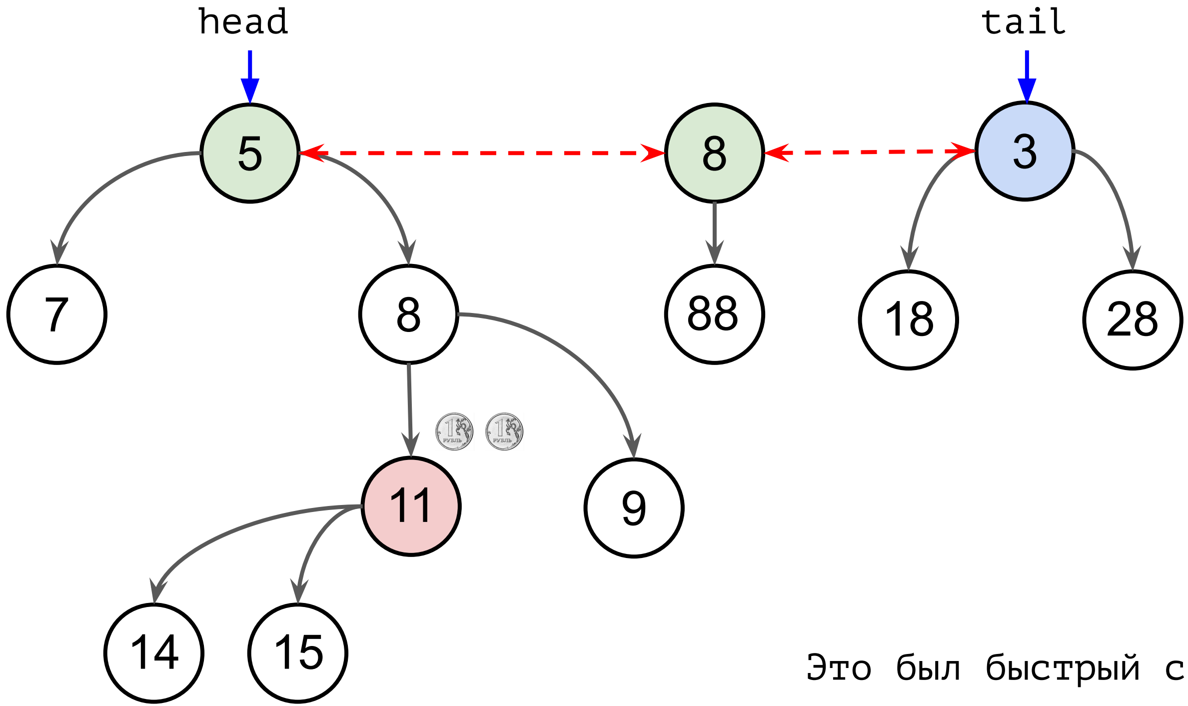
```
✓ 1.  insert(value)          -> O(1)
✓ 2.  peek_min()             -> O(1)
??? 2.  extract_min()          -> O*(D(N))
 ? 3.  decrease_key(s, k)
✓ 4.  merge(f1, f2)          -> O(1)
  5.  delete(s)
```

Идея: 1) пометки на вершинах
2) отсаживание поддеревьев при нарушении свойств пирамиды. Сложность?



Это был быстрый сценарий, отсадили за O(1).

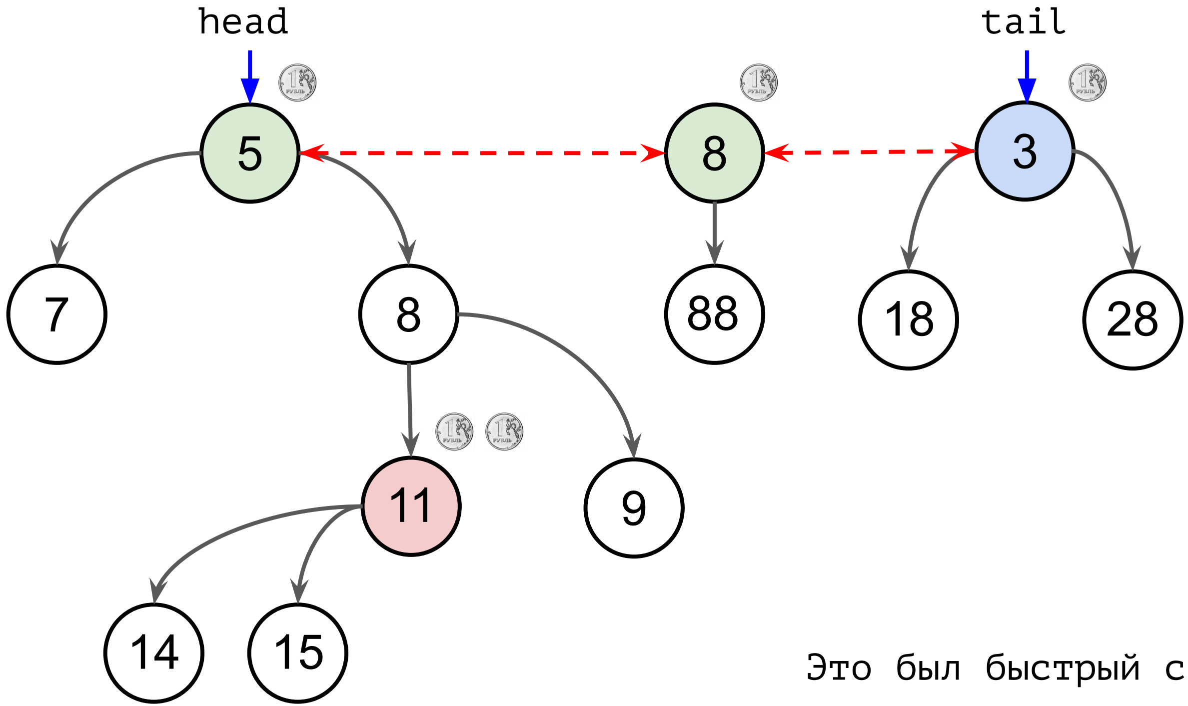
Однако, оставим пометку
на вершине, от которой
отсадили ребенка



Это был быстрый сценарий, отсадили за O(1).

Поэтому давайте здесь оставим предоплату.
Две монетки.

Однако, оставим пометку на вершине, от которой отсадили ребенка

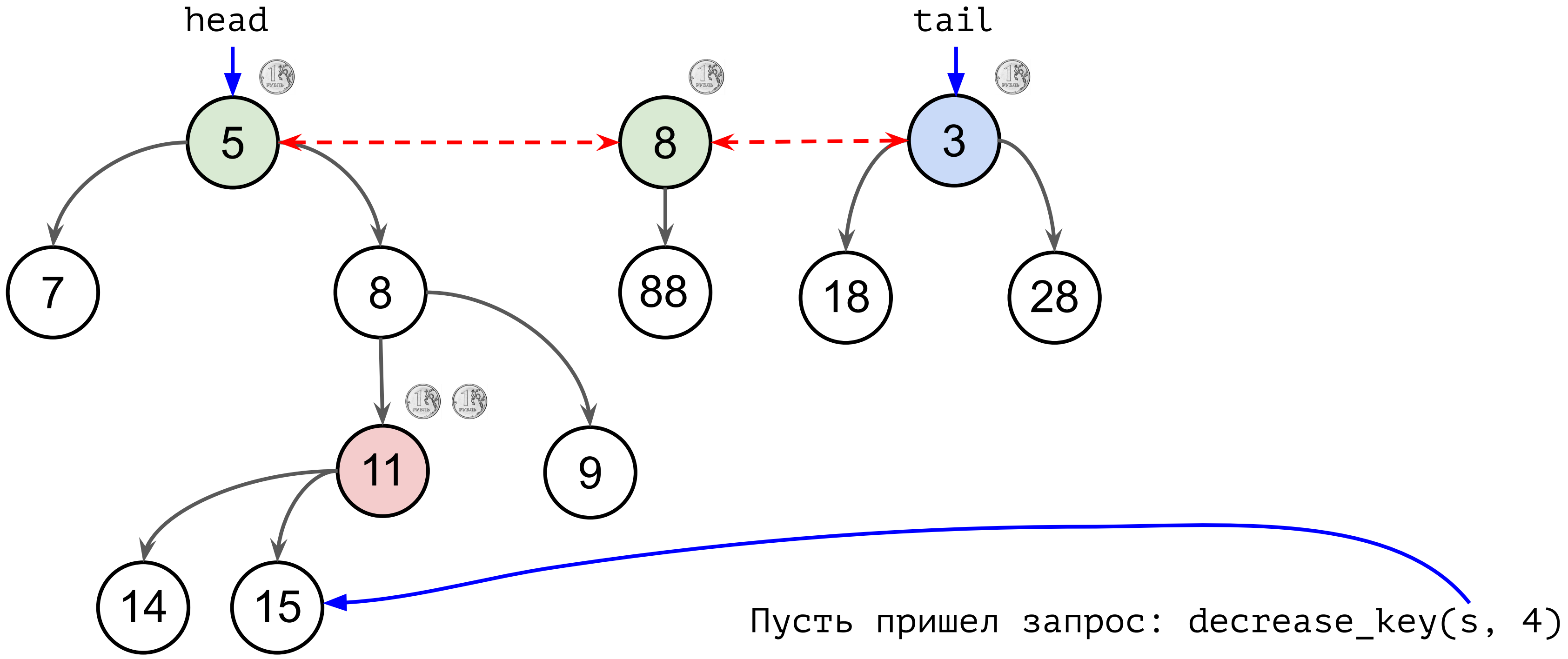


Однако, оставим пометку
на вершине, от которой
отсадили ребенка

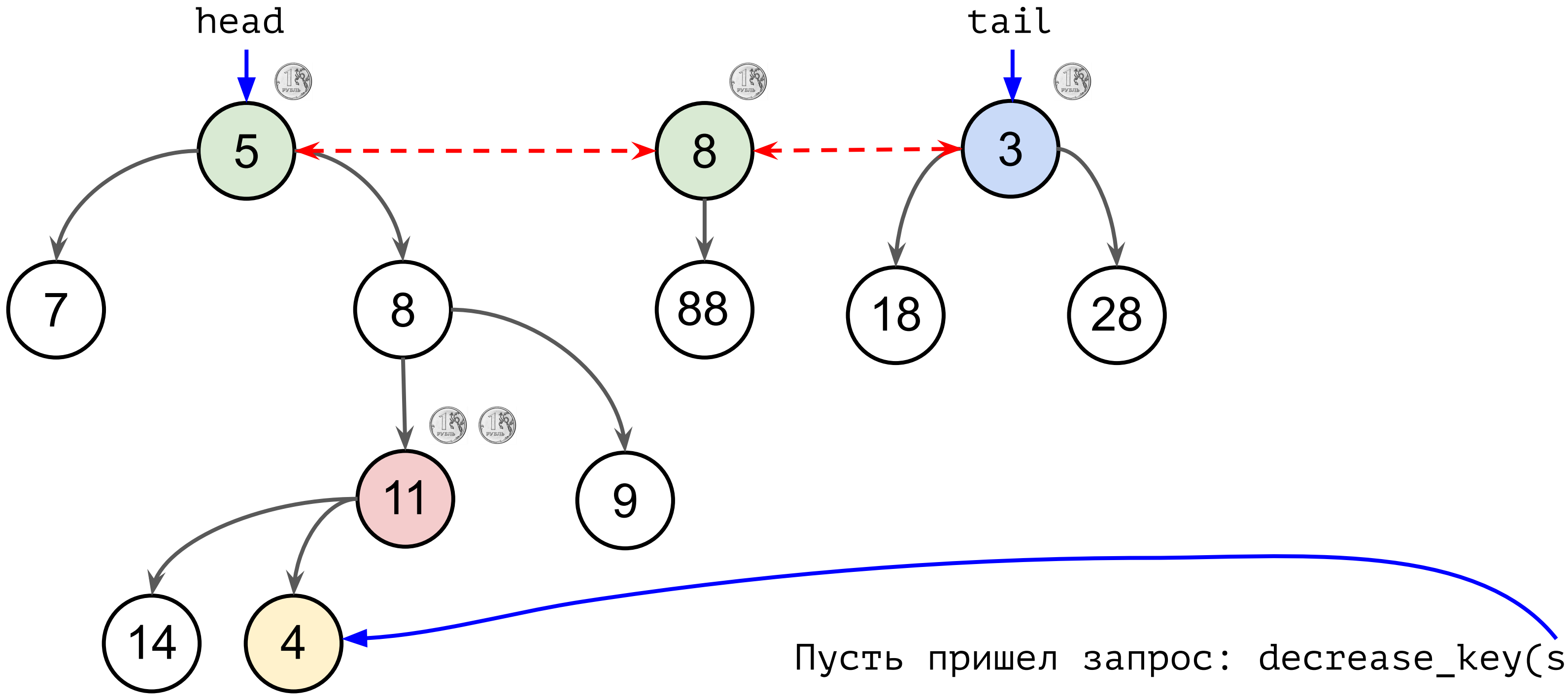
Это был быстрый сценарий, отсадили за O(1).

Поэтому давайте здесь оставим предоплату.
Две монетки.

Не забываем, что на всех корнях - тоже
монетки.



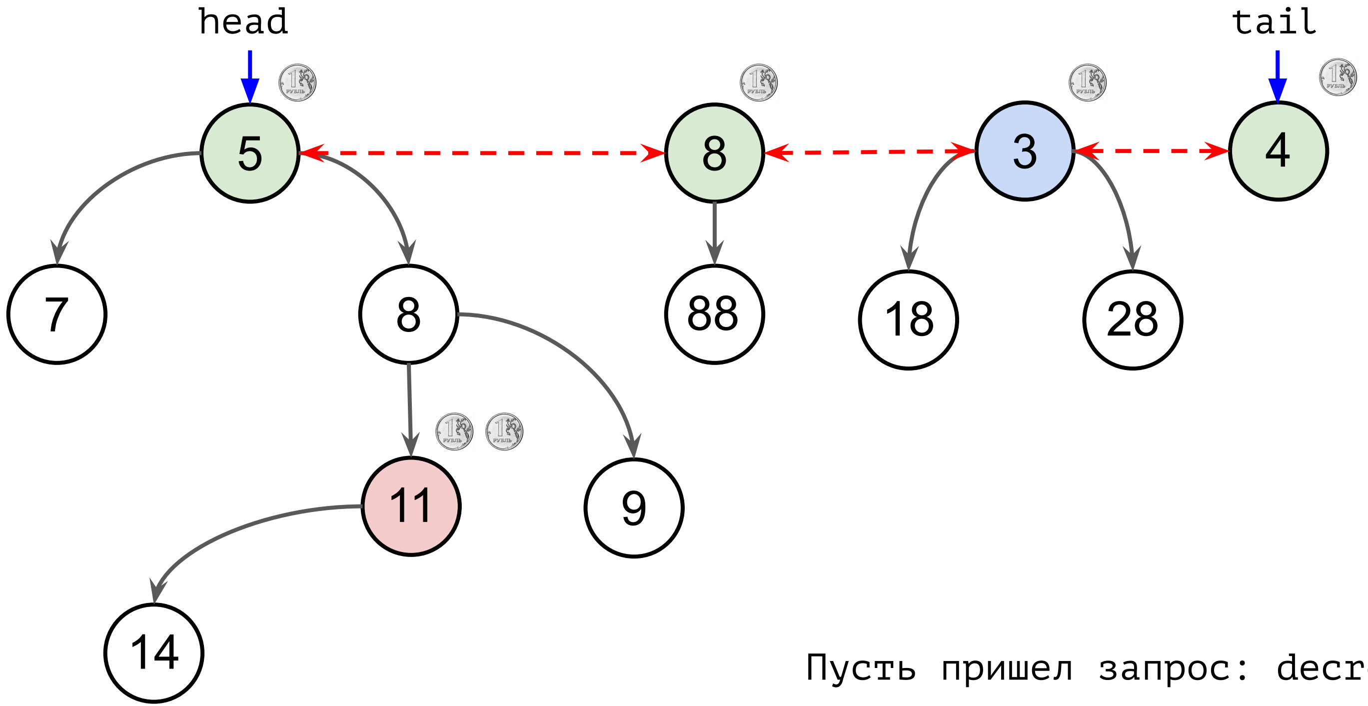
Однако, оставим пометку
на вершине, от которой
отсадили ребенка



Пусть пришел запрос: decrease_key(s, 4)

Во-первых, повторим те же действия.

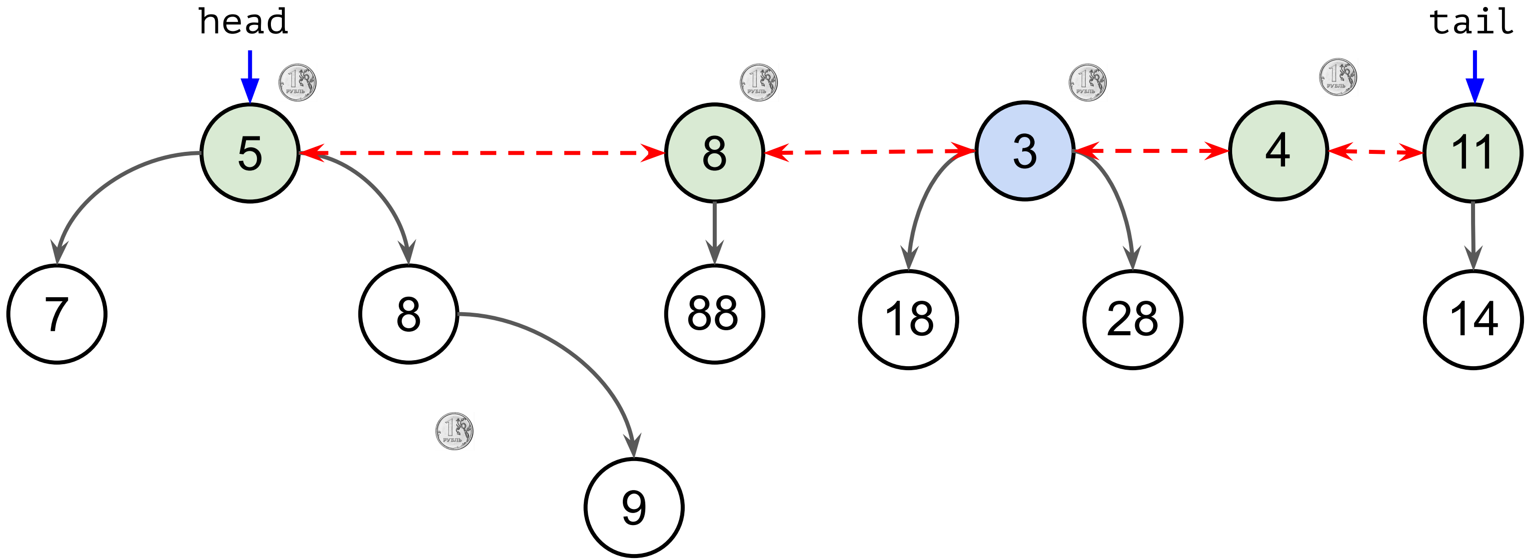
Однако, оставим пометку
на вершине, от которой
отсадили ребенка



Пусть пришел запрос: decrease_key(s, 4)

Во-первых, повторим те же действия.

Однако, оставим пометку на вершине, от которой отсадили ребенка

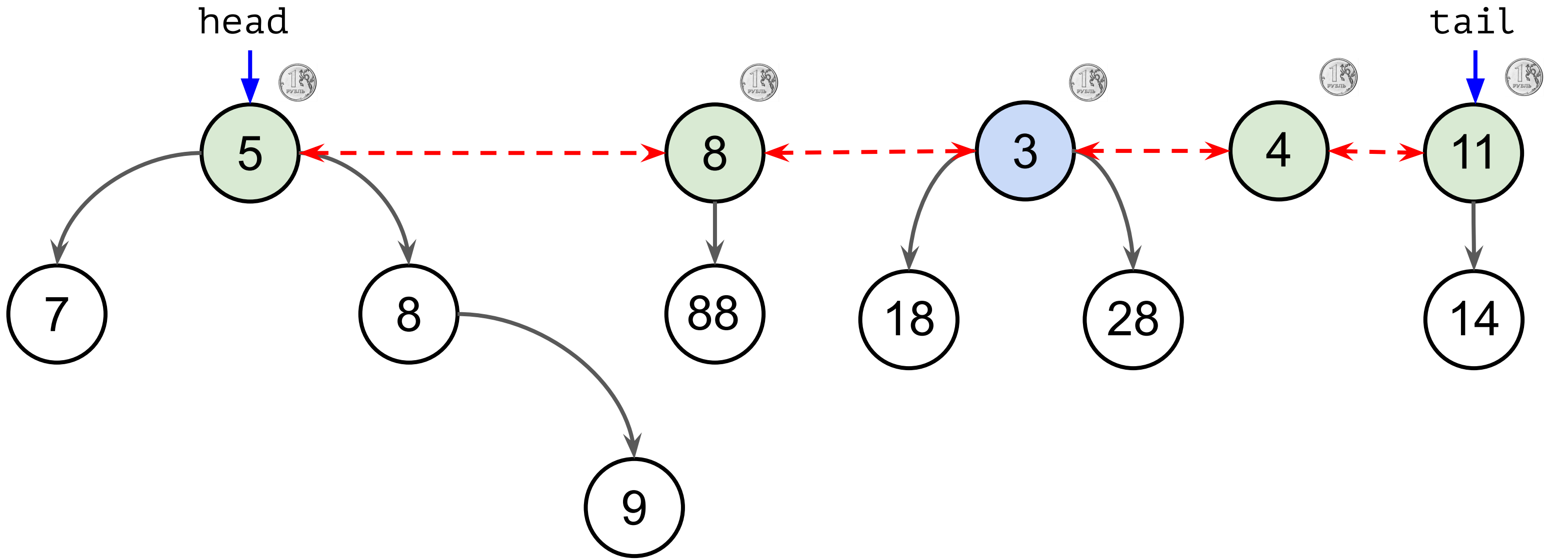


Пусть пришел запрос: decrease_key(s, 4)

Теперь расплачиваемся за перенос дерева нашей предоплатой.

Однако, оставим пометку на вершине, от которой отсадили ребенка

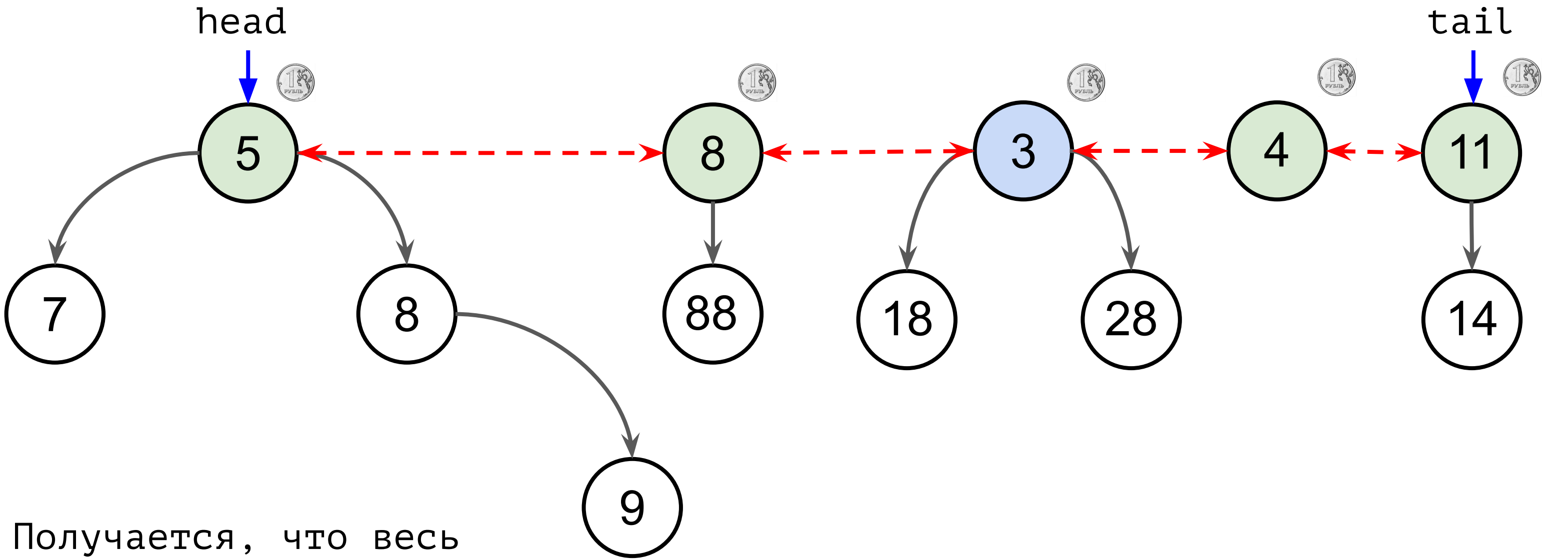
1 монета - за сам перенос.



Пусть пришел запрос: decrease_key(s, 4)

Теперь расплачиваемся за перенос дерева нашей предоплатой.

1 монета - за сам перенос.
2 монета - кладется на новый корень.

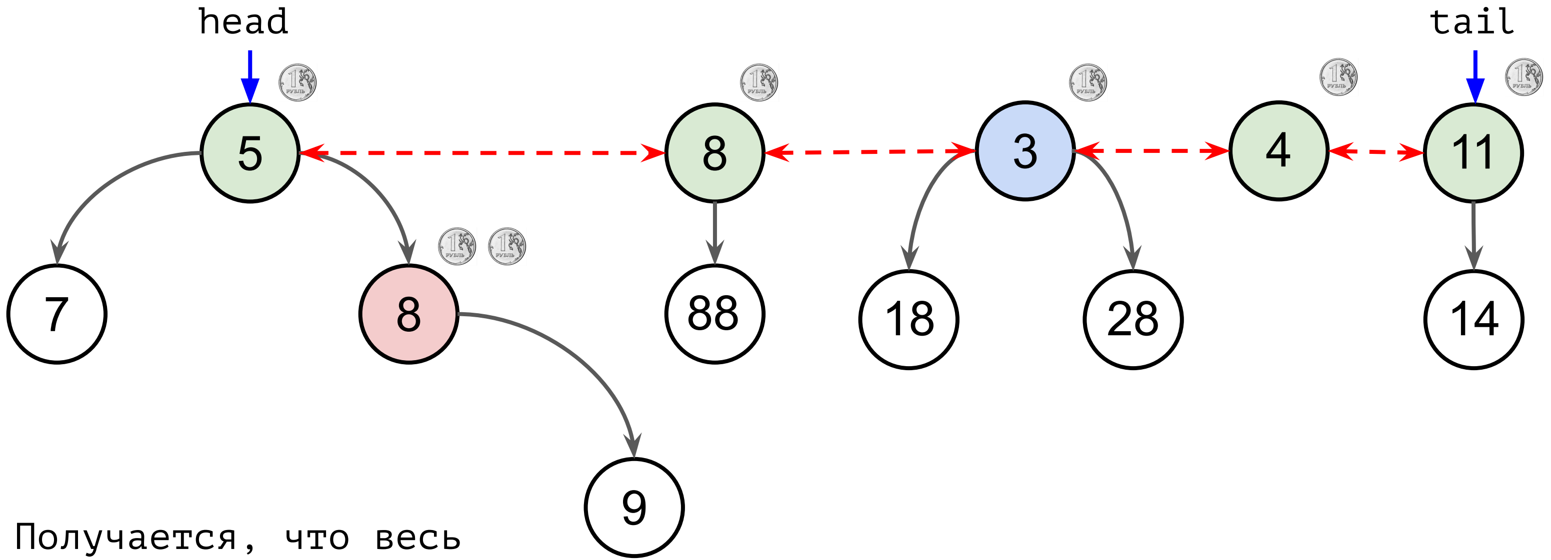


Получается, что весь рекурсивный проход вверх до первой не помеченной вершины - бесплатный.

Пусть пришел запрос: decrease_key(s, 4)

Теперь расплачиваемся за перенос дерева нашей предоплатой.

1 монета - за сам перенос.
2 монета - кладется на новый корень.



Получается, что весь рекурсивный проход вверх до первой не помеченной вершины - бесплатный.

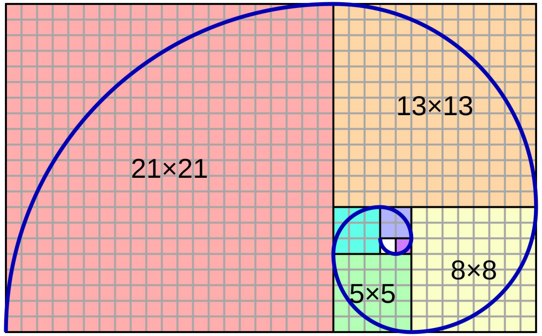
Не забываем положить две монетки (это запланированная предоплата).

Пусть пришел запрос: decrease_key(s, 4)

Теперь расплачиваемся за перенос дерева нашей предоплатой.

1 монета - за сам перенос.
2 монета - кладется на новый корень.

Фибоначчиева пирамида

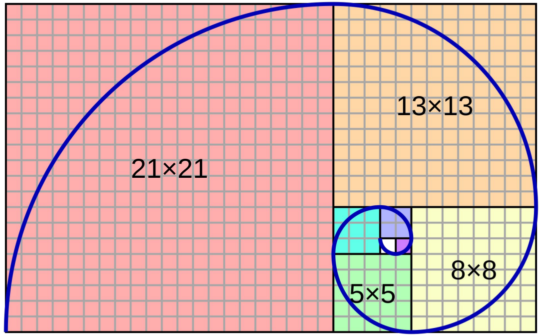


Операции:

```
✓ 1.  insert(value)        -> O(1)
✓ 2.  peek_min()           -> O(1)
??? 2.  extract_min()      -> O*(D(N))
? 3.  decrease_key(s, k)
✓ 4.  merge(f1, f2)        -> O(1)
  5.  delete(s)
```

Идея: 1) пометки на вершинах
2) отсаживание поддеревьев при нарушении свойств пирамиды. Сложность?

Фибоначчиева пирамида

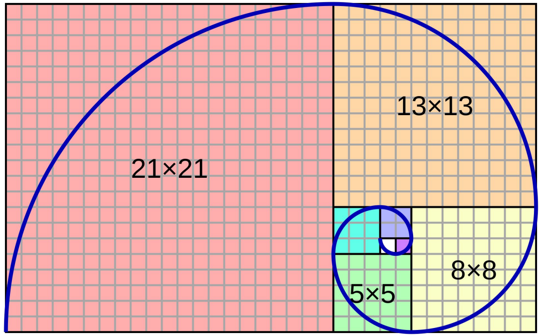


Операции:

```
✓ 1. insert(value)          -> O(1)
✓ 2. peek_min()             -> O(1)
??? 2. extract_min()        -> O*(D(N))
✓ 3. decrease_key(s, k)     -> O*(1)
✓ 4. merge(f1, f2)          -> O(1)
  5. delete(s)
```

Идея: 1) пометки на вершинах
2) отсаживание поддеревьев при нарушении свойств пирамиды. Сложность? O*(1)

Фибоначчиева пирамида

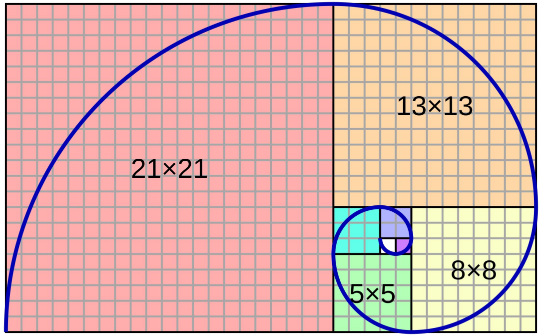


Операции:

```
✓ 1.  insert(value)        -> O(1)
✓ 2.  peek_min()           -> O(1)
??? 2.  extract_min()      -> O*(D(N))
✓ 3.  decrease_key(s, k)   -> O*(1)
✓ 4.  merge(f1, f2)        -> O(1)
  5.  delete(s)
```

delete(s) = decrease_key(s, -∞) и extract_min()

Фибоначчиева пирамида

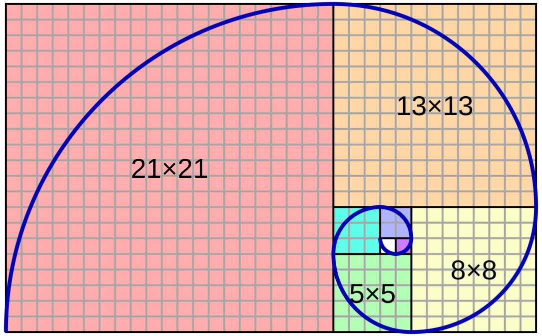


Операции:

```
✓ 1.  insert(value)        -> O(1)
✓ 2.  peek_min()           -> O(1)
??? 2.  extract_min()      -> O*(D(N))
✓ 3.  decrease_key(s, k)   -> O*(1)
✓ 4.  merge(f1, f2)        -> O(1)
??? 5.  delete(s)          -> O*(D(N))
```

```
delete(s) = decrease_key(s, -∞) и extract_min()
```

Фибоначчиева пирамида



Операции:

```
✓ 1.  insert(value)         -> O(1)
✓ 2.  peek_min()            -> O(1)
??? 2.  extract_min()       -> O*(D(N))
✓ 3.  decrease_key(s, k)    -> O*(1)
✓ 4.  merge(f1, f2)         -> O(1)
??? 5.  delete(s)           -> O*(D(N))
```

Последний вопрос: правда ли, что D(N) в построенной нами структуре данных - это logN? Тогда все сойдется.

Числа Фибоначчи

$$F_k = \begin{cases} 0, & if\ k = 0, \\ 1, & if\ k = 1, \\ F_{k-1} + F_{k-2}, & if\ k \geq 2. \end{cases}$$

Числа Фибоначчи

Числа Фибоначчи

$$F_k = \begin{cases} 0, & if\ k = 0, \\ 1, & if\ k = 1, \\ F_{k-1} + F_{k-2}, & if\ k \geq 2. \end{cases}$$

Числа Фибоначчи

Лемма #1: $F_{k+2} = 1 + \sum_{i=0}^{k} F_i$

Числа Фибоначчи

$$F_k = \begin{cases} 0, & if\ k = 0, \\ 1, & if\ k = 1, \\ F_{k-1} + F_{k-2}, & if\ k \geq 2. \end{cases}$$

Числа Фибоначчи

Лемма #1: $F_{k+2} = 1 + \sum_{i=0}^{k} F_i$

Доказательство: индукция по k. База: $1 + \sum_{i=0}^{0} F_i = 1 + F_0 = 1 + 0 = 1 = F_2$

Числа Фибоначчи

$$F_k = \begin{cases} 0, & if\ k = 0, \\ 1, & if\ k = 1, \\ F_{k-1} + F_{k-2}, & if\ k \geq 2. \end{cases}$$

Числа Фибоначчи

Лемма #1: $F_{k+2} = 1 + \sum_{i=0}^{k} F_i$

Доказательство: индукция по k. База: $1 + \sum_{i=0}^{0} F_i = 1 + F_0 = 1 + 0 = 1 = F_2$

Шаг: пусть $F_{k+1} = 1 + \sum_{i=0}^{k-1} F_i$

Числа Фибоначчи

$$F_k = \begin{cases} 0, & if\ k = 0, \\ 1, & if\ k = 1, \\ F_{k-1} + F_{k-2}, & if\ k \geq 2. \end{cases}$$

Числа Фибоначчи

Лемма #1: $F_{k+2} = 1 + \sum_{i=0}^{k} F_i$

Доказательство: индукция по k. База: $1 + \sum_{i=0}^{0} F_i = 1 + F_0 = 1 + 0 = 1 = F_2$

Шаг: пусть $F_{k+1} = 1 + \sum_{i=0}^{k-1} F_i$

Тогда: $F_{k+2} = F_k + F_{k+1} = F_k + (1 + \sum_{i=0}^{k-1} F_i) = 1 + \sum_{i=0}^{k} F_i$ □

Числа Фибоначчи

$$F_k = \begin{cases} 0, & if\ k = 0, \\ 1, & if\ k = 1, \\ F_{k-1} + F_{k-2}, & if\ k \geq 2. \end{cases}$$

Числа Фибоначчи

Лемма #2: $F_{k+2} \geq \phi^k$, где $\phi = \frac{1+\sqrt{5}}{2}$

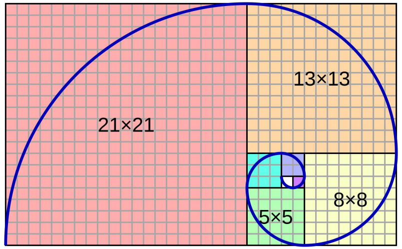
Числа Фибоначчи

$$F_k = \begin{cases} 0, & if\ k = 0, \\ 1, & if\ k = 1, \\ F_{k-1} + F_{k-2}, & if\ k \geq 2. \end{cases}$$

Числа Фибоначчи

Лемма #2: $F_{k+2} \geq \phi^k$, где $\phi = \frac{1+\sqrt{5}}{2}$

Доказательство: индукция по k, в качестве упражнения.



Фибоначчиева пирамида

Обозначение: **D(N)** - это ограничение сверху на степень корня в Фибоначчиевой пирамиде из **N** элементов.

Фибоначчиева пирамида

Обозначение: **D(N)** - это ограничение сверху на степень корня в Фибоначчиевой пирамиде из **N** элементов.

Лемма #3: пусть x - произвольный узел в фибоначчиевой пирамиде, а $y_1, y_2, \ldots, y_k$ - дочерние узлы x в порядке их связывания с x, начиная с более ранних.

Фибоначчиева пирамида

Обозначение: **D(N)** - это ограничение сверху на степень корня в Фибоначчиевой пирамиде из **N** элементов.

Лемма #3: пусть x - произвольный узел в фибоначчиевой пирамиде, а $y_1, y_2, \ldots, y_k$ - дочерние узлы x в порядке их связывания с x, начиная с более ранних.

Тогда $degree[y_1] \geq 0, degree[y_i] \geq i-2;\ i = 2, 3, \ldots, k$

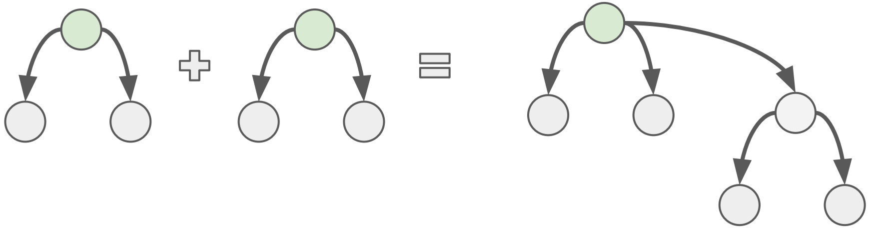
Фибоначчиева пирамида

Лемма #3: пусть x - произвольный узел в фибоначчиевой пирамиде, а $y_1, y_2, \ldots, y_k$ - дочерние узлы x в порядке их связывания с x, начиная с более ранних.

Тогда $degree[y_1] \geq 0, degree[y_i] \geq i - 2;\ i = 2, 3, \ldots, k$

Действительно:

1) мы подвешиваем одно дерево к другому только тогда, когда их degree совпадают.



Фибоначчиева пирамида

Лемма #3: пусть x - произвольный узел в фибоначчиевой пирамиде, а $y_1, y_2, ..., y_k$ - дочерние узлы x в порядке их связывания с x, начиная с более ранних.

Тогда $degree[y_1] \geq 0, degree[y_i] \geq i - 2;\ i = 2, 3, \ldots, k$

Действительно:

1) мы подвешиваем одно дерево к другому только тогда, когда их degree совпадают.

2) В момент подвешивания y_i у x уже были дети $y_1, y_2, ... y_{i-1}$, значит $degree(x) \geq i - 1$ (возможно их было даже больше, но кого-то удалили)

Фибоначчиева пирамида

Лемма #3: пусть x - произвольный узел в фибоначчиевой пирамиде, а $y_1, y_2, \ldots, y_k$ - дочерние узлы x в порядке их связывания с x, начиная с более ранних.

Тогда $degree[y_1] \geq 0, degree[y_i] \geq i - 2;\ i = 2, 3, \ldots, k$

Действительно:

1) мы подвешиваем одно дерево к другому только тогда, когда их degree совпадают.

2) В момент подвешивания y_i у x уже были дети $y_1, y_2, \ldots y_{i-1}$, значит $degree(x) \geq i - 1$ (возможно их было даже больше, но кого-то удалили)

3) Тогда $degree(y_i) \geq i - 1$ в момент добавления.

Фибоначчиева пирамида

Лемма #3: пусть x - произвольный узел в фибоначчиевой пирамиде, а $y_1, y_2, \ldots, y_k$ - дочерние узлы x в порядке их связывания с x, начиная с более ранних.

Тогда $degree[y_1] \geq 0, degree[y_i] \geq i-2;\ i = 2, 3, \ldots, k$

Действительно:

1) мы подвешиваем одно дерево к другому только тогда, когда их degree совпадают.

2) В момент подвешивания y_i у x уже были дети $y_1, y_2, \ldots y_{i-1}$, значит $degree(x) \geq i-1$ (возможно их было даже больше, но кого-то удалили)

3) Тогда $degree(y_i) \geq i-1$ в момент добавления.
А удалить ребенка y_i могли только 1 раз, потом отсаживают.

Фибоначчиева пирамида

Лемма #3: пусть x - произвольный узел в фибоначчиевой пирамиде,
а $y_1, y_2, \ldots, y_k$ - дочерние узлы x в порядке их связывания с x, начиная с более ранних.

Тогда $degree[y_1] \geq 0, degree[y_i] \geq i-2;\ i = 2, 3, \ldots, k$

Действительно:

1) мы подвешиваем одно дерево к другому только тогда, когда их degree совпадают.

2) В момент подвешивания y_i у x уже были дети $y_1, y_2, \ldots y_{i-1}$, значит $degree(x) \geq i-1$ (возможно их было даже больше, но кого-то удалили)

3) Тогда $degree(y_i) \geq i-1$ в момент добавления.
А удалить ребенка y_i могли только 1 раз => $degree(y_i) \geq i-2$ $\square$

Фибоначчиева пирамида

Лемма #4: пусть x - произвольный узел в фибоначчиевой пирамиде, $k = degree(x)$ - его степень, а $size(x)$ - количество узлов в поддереве, корнем которого является x.

Тогда: $size(x) \geq F_{k+2} \geq \phi^k$

Фибоначчиева пирамида

Лемма #4: пусть x - произвольный узел в фибоначчиевой пирамиде, $k = degree(x)$ - его степень, а $size(x)$ - количество узлов в поддереве, корнем которого является x.

Тогда: $size(x) \geq F_{k+2} \geq \phi^k$

Доказательство: введем $s_k =$ минимальное возможное количество элементов в поддереве степени k.

Фибоначчиева пирамида

Лемма #4: пусть x - произвольный узел в фибоначчиевой пирамиде, $k = degree(x)$ - его степень, а $size(x)$ - количество узлов в поддереве, корнем которого является x.

Тогда: $size(x) \geq F_{k+2} \geq \phi^k$

Доказательство: введем $s_k =$ минимальное возможное количество элементов в поддереве степени k.

тогда $size(x) \geq s_k$

Лемма #4: пусть x - произвольный узел в фибоначчиевой пирамиде, $k = degree(x)$ - его степень, а $size(x)$ - количество узлов в поддереве, корнем которого является x.

Тогда: $size(x) \geq F_{k+2} \geq \phi^k$

Доказательство: введем $s_k =$ минимальное возможное количество элементов в поддереве степени k.

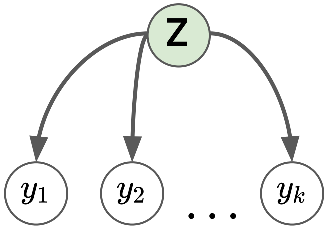
тогда $size(x) \geq s_k$;

Лемма #4: пусть x - произвольный узел в фибоначчиевой пирамиде, $k = degree(x)$ - его степень, а $size(x)$ - количество узлов в поддереве, корнем которого является x.

Тогда: $size(x) \geq F_{k+2} \geq \phi^k$

Доказательство: введем $s_k =$ минимальное возможное количество элементов в поддереве степени k.

тогда $size(x) \geq s_k$; Пусть z - вершина дерева, на которой достигается s_k т.е. degree(z) = k, size(x) = s_k



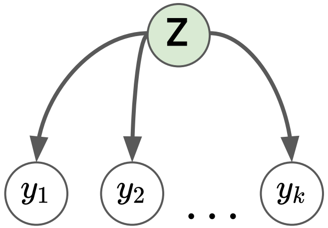
Лемма #4: пусть x - произвольный узел в фибоначчиевой пирамиде, $k = degree(x)$ - его степень, а $size(x)$ - количество узлов в поддереве, корнем которого является x.

Тогда: $size(x) \geq F_{k+2} \geq \phi^k$

Доказательство: введем $s_k =$ минимальное возможное количество элементов в поддереве степени k.

тогда $size(x) \geq s_k$; Пусть z - вершина дерева, на которой достигается s_k т.е. degree(z) = k, size(x) = s_k

тогда $s_k \geq 1 + 1 + \sum_{i=2}^{k} s_{degree[y_i]}$



Лемма #4: пусть x - произвольный узел в фибоначчиевой пирамиде, $k = degree(x)$ - его степень, а $size(x)$ - количество узлов в поддереве, корнем которого является x.

Тогда: $size(x) \geq F_{k+2} \geq \phi^k$

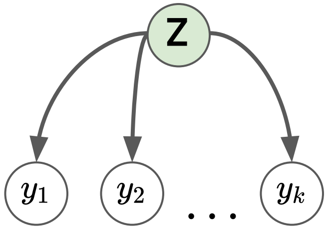
Доказательство: введем $s_k =$ минимальное возможное количество элементов в поддереве степени k.

тогда $size(x) \geq s_k$; Пусть z - вершина дерева, на которой достигается s_k т.е. degree(z) = k, size(x) = s_k

тогда $s_k \geq 1 + 1 + \sum_{i=2}^{k} s_{degree[y_i]}$

за сам z

за y1 для которого
минимум degree по лемме 3
равен 0



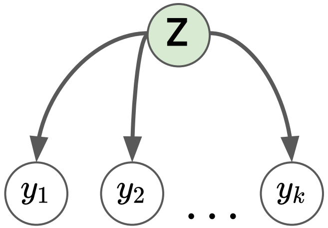
Лемма #4: пусть x - произвольный узел в фибоначчиевой пирамиде, $k = degree(x)$ - его степень, а $size(x)$ - количество узлов в поддереве, корнем которого является x.

Тогда: $size(x) \geq F_{k+2} \geq \phi^k$

Доказательство: введем $s_k =$ минимальное возможное количество элементов в поддереве степени k.

тогда $size(x) \geq s_k$; Пусть z - вершина дерева, на которой достигается s_k т.е. degree(z) = k, size(x) = s_k

тогда $s_k \geq 1 + 1 + \sum_{i=2}^{k} s_{degree[y_i]} \geq 2 + \sum_{i=2}^{k} s_{i-2}$



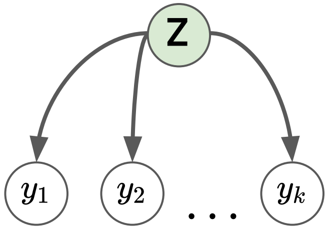
Лемма #4: пусть x - произвольный узел в фибоначчиевой пирамиде, $k = degree(x)$ - его степень, а $size(x)$ - количество узлов в поддереве, корнем которого является x.

Тогда: $size(x) \geq F_{k+2} \geq \phi^k$

Доказательство: введем $s_k =$ минимальное возможное количество элементов в поддереве степени k.

тогда $size(x) \geq s_k$; Пусть z - вершина дерева, на которой достигается s_k т.е. degree(z) = k, size(x) = s_k

тогда $s_k \geq 1 + 1 + \sum_{i=2}^{k} s_{degree[y_i]} \geq 2 + \sum_{i=2}^{k} s_{i-2}$



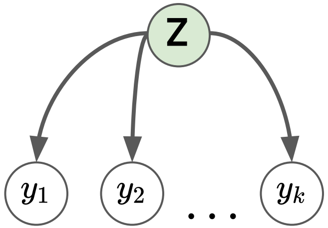
т.к. по лемме #3 $degree[y_i] \geq i - 2$ а функция s_k монотонная (увеличение степени не может уменьшить размер).

Лемма #4: пусть x - произвольный узел в фибоначчиевой пирамиде, $k = degree(x)$ - его степень, а $size(x)$ - количество узлов в поддереве, корнем которого является x.

Тогда: $size(x) \geq F_{k+2} \geq \phi^k$

Доказательство: введем $s_k =$ минимальное возможное количество элементов в поддереве степени k.

$$s_k \geq 2 + \sum_{i=2}^{k} s_{i-2}$$



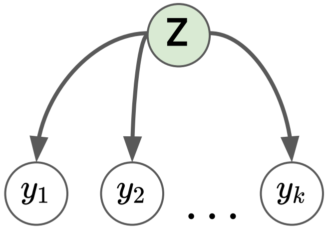
Лемма #4: пусть x - произвольный узел в фибоначчиевой пирамиде, $k = degree(x)$ - его степень, а $size(x)$ - количество узлов в поддереве, корнем которого является x.

Тогда: $size(x) \geq F_{k+2} \geq \phi^k$

Доказательство: введем $s_k =$ минимальное возможное количество элементов в поддереве степени k.

$$s_k \geq 2 + \sum_{i=2}^{k} s_{i-2}$$

Покажем теперь, что $s_k \geq F_{k+2}$.



Лемма #4: пусть x - произвольный узел в фибоначчиевой пирамиде, $k = degree(x)$ - его степень, а $size(x)$ - количество узлов в поддереве, корнем которого является x.

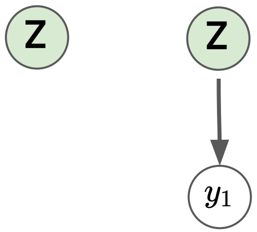
Тогда: $size(x) \geq F_{k+2} \geq \phi^k$

Доказательство: введем $s_k =$ минимальное возможное количество элементов в поддереве степени k.

$$s_k \geq 2 + \sum_{i=2}^{k} s_{i-2}$$

Покажем теперь, что $s_k \geq F_{k+2}$.

Индукция: база для k = 0, k = 1 - верно.



Лемма #4: пусть x - произвольный узел в фибоначчиевой пирамиде, $k = degree(x)$ - его степень, а $size(x)$ - количество узлов в поддереве, корнем которого является x.

Тогда: $size(x) \geq F_{k+2} \geq \phi^k$

Доказательство: введем $s_k =$ минимальное возможное количество элементов в поддереве степени k.

$$s_k \geq 2 + \sum_{i=2}^{k} s_{i-2}$$

Покажем теперь, что $s_k \geq F_{k+2}$.

Индукция: база для k = 0, k = 1 - верно.

Шаг: пусть $s_i \geq F_{i+2}$ для $i = 0, 1, \ldots, k-1$

Лемма #4: пусть x - произвольный узел в фибоначчиевой пирамиде, $k = degree(x)$ - его степень, а $size(x)$ - количество узлов в поддереве, корнем которого является x.

Тогда: $size(x) \geq F_{k+2} \geq \phi^k$

Доказательство: введем $s_k =$ минимальное возможное количество элементов в поддереве степени k.

$$s_k \geq 2 + \sum_{i=2}^{k} s_{i-2}$$

Покажем теперь, что $s_k \geq F_{k+2}$.

Индукция: база для k = 0, k = 1 - верно.

Шаг: пусть $s_i \geq F_{i+2}$ для $i = 0, 1, \ldots, k-1$. Тогда:

$$s_k \geq 2 + \sum_{i=2}^{k} S_{i-2} \geq 2 + \sum_{i=2}^{k} F_i = 1 + \sum_{i=0}^{k} F_i = F_{k+2} \quad \square$$

По лемме #1

Лемма #4: пусть x - произвольный узел в фибоначчиевой пирамиде, $k = degree(x)$ - его степень, а $size(x)$ - количество узлов в поддереве, корнем которого является x.

Тогда: $size(x) \geq F_{k+2} \geq \phi^k$

Доказательство: введем $s_k =$ минимальное возможное количество элементов в поддереве степени k.

$$s_k \geq 2 + \sum_{i=2}^{k} s_{i-2} \geq F_{k+2}$$

Лемма #4: пусть x - произвольный узел в фибоначчиевой пирамиде, $k = degree(x)$ - его степень, а $size(x)$ - количество узлов в поддереве, корнем которого является x.

Тогда: $size(x) \geq F_{k+2} \geq \phi^k$

Доказательство: введем $s_k =$ минимальное возможное количество элементов в поддереве степени k.

$$s_k \geq 2 + \sum_{i=2}^{k} s_{i-2} \geq F_{k+2}$$

Тогда: $size(x) \geq s_k \geq F_{k+2} \geq \phi^k$ $\square$

Фибоначчиева пирамида

Обозначение: **D(N)** - это ограничение сверху на степень корня в Фибоначчиевой пирамиде из **N** элементов.

Лемма #4: пусть x - произвольный узел в фибоначчиевой пирамиде, $k = degree(x)$ - его степень, а $size(x)$ - количество узлов в поддереве, корнем которого является x.

Тогда: $size(x) \geq F_{k+2} \geq \phi^k$

Доказательство: введем $s_k =$ минимальное возможное количество элементов в поддереве степени k.

$$s_k \geq 2 + \sum_{i=2}^{k} s_{i-2} \geq F_{k+2}$$

Тогда: $size(x) \geq s_k \geq F_{k+2} \geq \phi^k$ $\square$

Следствие: максимальная степень D(N) произвольного узла в фибоначчиевой пирамиде с n узлами равна O(logN).

Лемма #4: пусть x - произвольный узел в фибоначчиевой пирамиде, $k = degree(x)$ - его степень, а $size(x)$ - количество узлов в поддереве, корнем которого является x.

Тогда: $size(x) \geq F_{k+2} \geq \phi^k$

Доказательство: введем $s_k =$ минимальное возможное количество элементов в поддереве степени k.

$$s_k \geq 2 + \sum_{i=2}^{k} s_{i-2} \geq F_{k+2}$$

Тогда: $size(x) \geq s_k \geq F_{k+2} \geq \phi^k$ $\square$

Следствие: максимальная степень D(N) произвольного узла в фибоначчиевой пирамиде с n узлами равна O(logN).

Доказательство: пусть x - произвольный узел в куче из n узлов, k = degree(x).

Лемма #4: пусть x - произвольный узел в фибоначчиевой пирамиде, $k = degree(x)$ - его степень, а $size(x)$ - количество узлов в поддереве, корнем которого является x.

Тогда: $size(x) \geq F_{k+2} \geq \phi^k$

Доказательство: введем $s_k =$ минимальное возможное количество элементов в поддереве степени k.

$$s_k \geq 2 + \sum_{i=2}^{k} s_{i-2} \geq F_{k+2}$$

Тогда: $size(x) \geq s_k \geq F_{k+2} \geq \phi^k$ $\square$

Следствие: максимальная степень D(N) произвольного узла в фибоначчиевой пирамиде с n узлами равна O(logN).

Доказательство: пусть x - произвольный узел в куче из n узлов, k = degree(x). Тогда справедливо: $n \geq size(x) \geq s_k \geq F_{k+2} \geq \phi^k$

Лемма #4: пусть x - произвольный узел в фибоначчиевой пирамиде, $k = degree(x)$ - его степень, а $size(x)$ - количество узлов в поддереве, корнем которого является x.

Тогда: $size(x) \geq F_{k+2} \geq \phi^k$

Доказательство: введем $s_k =$ минимальное возможное количество элементов в поддереве степени k.

$$s_k \geq 2 + \sum_{i=2}^{k} s_{i-2} \geq F_{k+2}$$

Тогда: $size(x) \geq s_k \geq F_{k+2} \geq \phi^k$ $\square$

Следствие: максимальная степень D(N) произвольного узла в фибоначчиевой пирамиде с n узлами равна O(logN).

Доказательство: пусть x - произвольный узел в куче из n узлов, k = degree(x). Тогда справедливо: $n \geq size(x) \geq s_k \geq F_{k+2} \geq \phi^k$

Логарифмируем по основанию ϕ: $k \leq log_{\phi} n$

Лемма #4: пусть x - произвольный узел в фибоначчиевой пирамиде, $k = degree(x)$ - его степень, а $size(x)$ - количество узлов в поддереве, корнем которого является x.

Тогда: $size(x) \geq F_{k+2} \geq \phi^k$

Доказательство: введем $s_k =$ минимальное возможное количество элементов в поддереве степени k.

$$s_k \geq 2 + \sum_{i=2}^{k} s_{i-2} \geq F_{k+2}$$

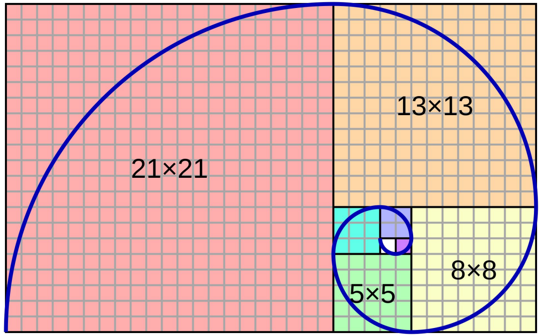
Тогда: $size(x) \geq s_k \geq F_{k+2} \geq \phi^k$ $\square$

Следствие: максимальная степень D(N) произвольного узла в фибоначчиевой пирамиде с n узлами равна O(logN).

Доказательство: пусть x - произвольный узел в куче из n узлов, k = degree(x). Тогда справедливо: $n \geq size(x) \geq s_k \geq F_{k+2} \geq \phi^k$

Логарифмируем по основанию ϕ: $k \leq log_\phi n$. Тогда $D(N) = O(logN)$ $\square$

Фибоначчиева пирамида

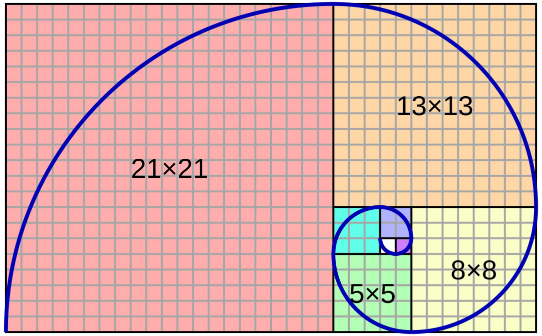


Операции:

```
✓ 1.   insert(value)          -> O(1)
✓ 2.   peek_min()             -> O(1)
??? 2. extract_min()          -> O*(D(N))
✓ 3.   decrease_key(s, k)     -> O*(1)
✓ 4.   merge(f1, f2)          -> O(1)
??? 5. delete(s)              -> O*(D(N))
```

Последний вопрос: правда ли, что D(N) в построенной нами структуре данных - это logN? Тогда все сойдется.

Фибоначчиева пирамида

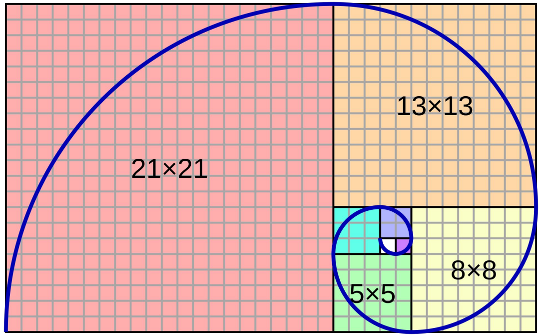


Операции:

```
✓ 1.  insert(value)          -> O(1)
✓ 2.  peek_min()             -> O(1)
✓ 2.  extract_min()          -> O*(logN)
✓ 3.  decrease_key(s, k)     -> O*(1)
✓ 4.  merge(f1, f2)          -> O(1)
✓ 5.  delete(s)              -> O*(logN)
```

Последний вопрос: правда ли, что D(N) в построенной нами структуре данных - это logN? Да, это так.

Фибоначчиева пирамида



Операции:

```
✓ 1.  insert(value)          -> O(1)
✓ 2.  peek_min()             -> O(1)
✓ 2.  extract_min()          -> O*(logN) -> O(N) (в худшем)
✓ 3.  decrease_key(s, k)     -> O*(1)    -> O(N) (в худшем)
✓ 4.  merge(f1, f2)          -> O(1)
✓ 5.  delete(s)              -> O*(logN) -> O(N) (в худшем)
```

Последний вопрос: правда ли, что D(N) в построенной нами структуре данных - это logN? Да, это так.

Алгоритм Дейкстры поиска кратчайших путей

Оценка сложности:

1. Обрабатываем каждую вершину 1 раз
2. Для каждой вершины перебираем ребра
3. На каждом шаге алгоритма ищем минимум

$$O(|V| * log(|V|) + |E| * log(|V|))$$

Но можем ли мы еще лучше?

Для случая пирамиды - T = O(log(|V|))

Но нужно еще и пометки обновить, а это как раз decrease_priority(...) - каждая тоже за O(log(|V|))

Алгоритм Дейкстры поиска кратчайших путей

Оценка сложности:

1. Обрабатываем каждую вершину 1 раз
2. Для каждой вершины перебираем ребра
3. На каждом шаге алгоритма ищем минимум

$$O^*(|V| * log(|V|) + |E|)$$

Для Фибоначчиевой кучи обновление пометки - O*(1)

Алгоритм Дейкстры поиска кратчайших путей

Оценка сложности:

1. Обрабатываем каждую вершину 1 раз
2. Для каждой вершины перебираем ребра
3. На каждом шаге алгоритма ищем минимум

$$O^*(|V| * log(|V|) + |E|)$$

Для Фибоначчиевой кучи обновление пометки - O*(1)
(но константы могут быть просто ужасны, это тоже стоит учитывать)

Сливаемые пирамиды

Множество значений: пары <priority: int, value: T>

Операции:

```
1.  insert(value)          -> O(logN)
2.  peek_min()             -> O(1) 🥰
3.  extract_min()          -> O(logN)
4.  decrease_key(s, k)     -> O(logN)
5.  merge(H1, H2)          -> O(N)
6.  delete(s)              -> O(logN)
```

Как реализовать?
Бинарная куча!

O(N) – неприятно, хочется оптимизировать merge

Биномиальная пирамида

Множество значений: пары <priority: int, value: T>

Операции:

1. insert(value) -> O(logN)
2. peek_min() -> O(logN) 🥴
3. extract_min() -> O(logN)
4. decrease_key(s, k) -> O(logN)
5. merge(H1, H2) -> O(logN) 🥰
6. delete(s) -> O(logN)

Как реализовать?

~~Бинарная куча!~~

Биномиальная!

Биномиальная пирамида

Множество значений: пары <priority: int, value: T>

Операции:

```
1.  insert(value)         -> O(1) 🥰
2.  peek_min()            -> O(1) 🥰
3.  extract_min()         -> O*(logN)
4.  decrease_key(s, k)    -> O*(1)
5.  merge(H1, H2)         -> O(1)
6.  delete(s)             -> O*(logN)
```

Как реализовать?

~~Бинарная куча!~~

~~Биномиальная!~~

Фибоначчиева!

Биномиальная пирамида

Множество значений: пары <priority: int, value: T>

Операции:

1. insert(value) -> O(1) 🥰
2. peek_min() -> O(1) 🥰
3. extract_min() -> O*(logN)
4. decrease_key(s, k) -> O*(1)
5. merge(H1, H2) -> O(1)
6. delete(s) -> O*(logN)

Как реализовать?

~~Бинарная куча!~~

~~Биномиальная!~~

Фибоначчиева!

Но бойтесь констант!

Takeaways

- Алгоритм Дейкстры можно и нужно оптимизировать через пирамиды.

Takeaways

- Фибоначчиева пирамида, как пример потрясающей работы с амортизационной сложностью.
- Определение и правила менее жесткие, чем у биномиальной, но реализовывать не проще.
- Красивые оценки, но бойтесь констант.