Мини-задача #46 (1 балл, дополнительная)

Решите задачу поиска чисел >= k на отрезке с помощью персистентного дерева отрезков.

Мини-задача #47 (2 балл, дополнительная)

Реализуйте очередь с полной персистентностью.

Оцените временную и емкостную сложность операций.

Сложнее, чем может показаться!

Алгоритмы и структуры данных

Персистентные структуры данных

Персистентность

Задача: пусть есть некоторая структура данных, поддерживающая запросы на изменение.

Персистентность

Задача: пусть есть некоторая структура данных, поддерживающая запросы на изменение.

Нужно уметь получать доступ к состоянию в разные моменты времени

Персистентность

Задача: пусть есть некоторая структура данных, поддерживающая запросы на изменение.

Нужно уметь получать доступ к состоянию в разные моменты времени (т.е. хранить историю изменений и прошлые версии структуры данных)

Персистентность

Задача: пусть есть некоторая структура данных, поддерживающая запросы на изменение.

Нужно уметь получать доступ к состоянию в разные моменты времени (т.е. хранить историю изменений и прошлые версии структуры данных). Такие структуры данных назовем персистентными.

Персистентность

Задача: пусть есть некоторая структура данных, поддерживающая запросы на изменение.

Нужно уметь получать доступ к состоянию в разные моменты времени (т.е. хранить историю изменений и прошлые версии структуры данных). Такие структуры данных назовем персистентными.

Персистентность

Задача: пусть есть некоторая структура данных, поддерживающая запросы на изменение.

Нужно уметь получать доступ к состоянию в разные моменты времени (т.е. хранить историю изменений и прошлые версии структуры данных). Такие структуры данных назовем персистентными.

Используются:

1) в базах данных,
2) системах контроля версий,
3) для систем с одновременным доступом, когда новая версия еще только считается;

Виды персистентности

Виды персистентности

1) Частичная персистентность: можем посмотреть на любую версию в прошлом, но изменять (создавая новую версию) можно только последнюю текущую версию.

Виды персистентности

1) Частичная персистентность: можем посмотреть на любую версию в прошлом, но изменять (создавая новую версию) можно только последнюю текущую версию.

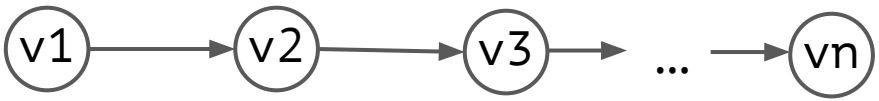
Виды персистентности

1) Частичная персистентность: можем посмотреть на любую версию в прошлом, но изменять (создавая новую версию) можно только последнюю текущую версию.

add(...)

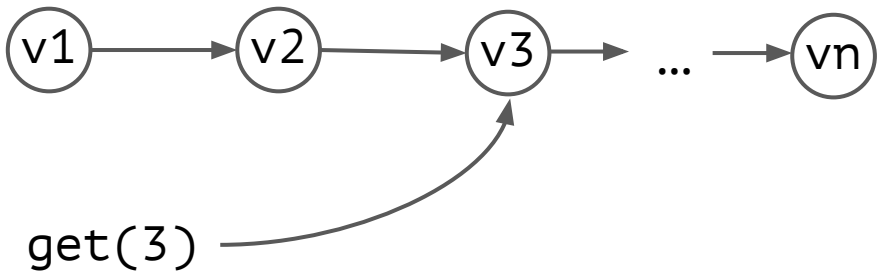
Виды персистентности

1) Частичная персистентность: можем посмотреть на любую версию в прошлом, но изменять (создавая новую версию) можно только последнюю текущую версию.



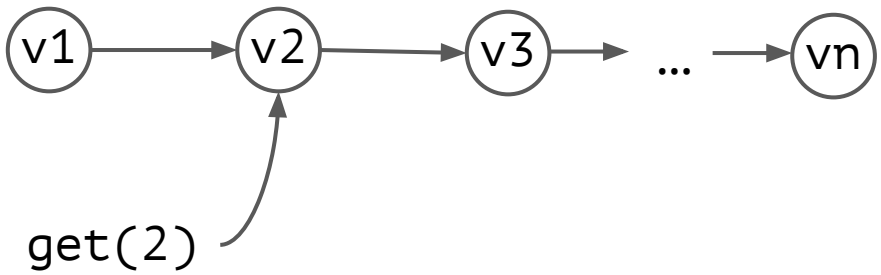
Виды персистентности

1) Частичная персистентность: можем посмотреть на любую версию в прошлом, но изменять (создавая новую версию) можно только последнюю текущую версию.



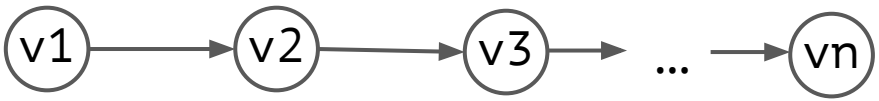
Виды персистентности

1) Частичная персистентность: можем посмотреть на любую версию в прошлом, но изменять (создавая новую версию) можно только последнюю текущую версию.



Виды персистентности

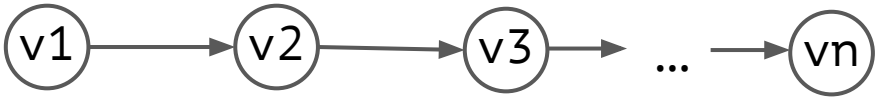
1) Частичная персистентность: можем посмотреть на любую версию в прошлом, но изменять (создавая новую версию) можно только последнюю текущую версию.



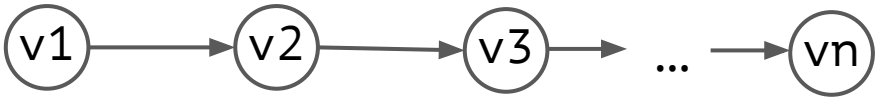
2) Полная персистентность: можем посмотреть любую версию в прошлом и изменить (создавая новую версию) любую версию!

Виды персистентности

1) Частичная персистентность: можем посмотреть на любую версию в прошлом, но изменять (создавая новую версию) можно только последнюю текущую версию.

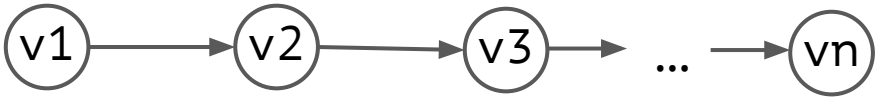


2) Полная персистентность: можем посмотреть любую версию в прошлом и изменить (создавая новую версию) любую версию!

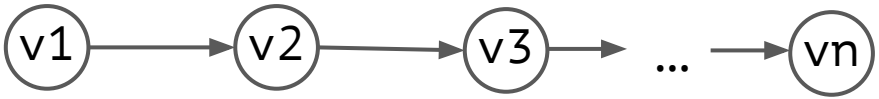


Виды персистентности

1) Частичная персистентность: можем посмотреть на любую версию в прошлом, но изменять (создавая новую версию) можно только последнюю текущую версию.



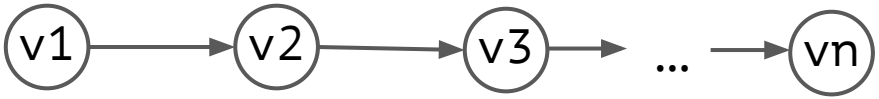
2) Полная персистентность: можем посмотреть любую версию в прошлом и изменить (создавая новую версию) любую версию!



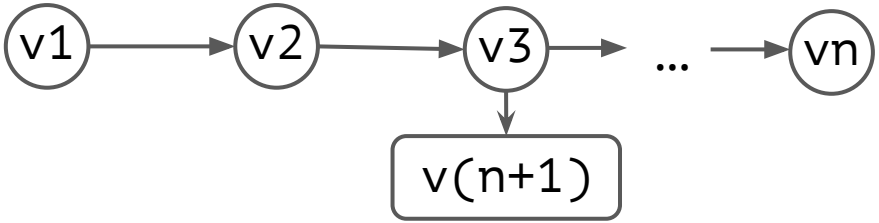

```
add(3, ...)
```

Виды персистентности

1) Частичная персистентность: можем посмотреть на любую версию в прошлом, но изменять (создавая новую версию) можно только последнюю текущую версию.



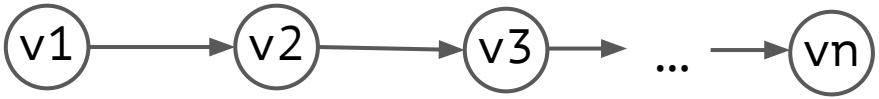
2) Полная персистентность: можем посмотреть любую версию в прошлом и изменить (создавая новую версию) любую версию!



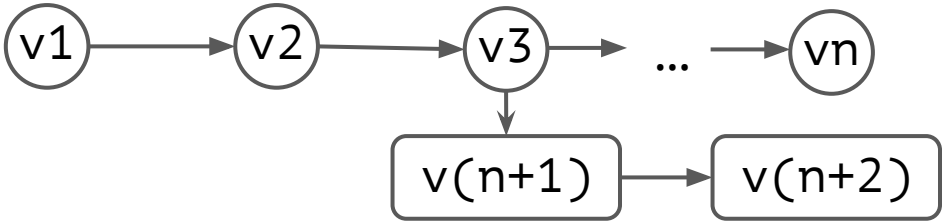

```
add(3, ...)
```

Виды персистентности

1) Частичная персистентность: можем посмотреть на любую версию в прошлом, но изменять (создавая новую версию) можно только последнюю текущую версию.



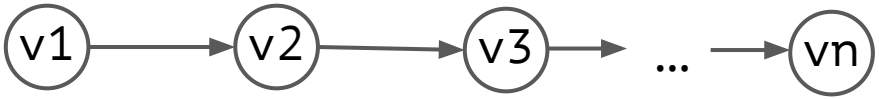
2) Полная персистентность: можем посмотреть любую версию в прошлом и изменить (создавая новую версию) любую версию!



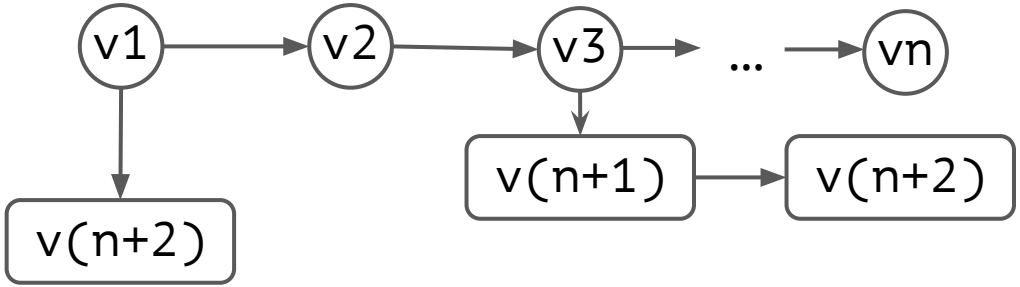

```
add(n + 1, ...)
```

Виды персистентности

1) Частичная персистентность: можем посмотреть на любую версию в прошлом, но изменять (создавая новую версию) можно только последнюю текущую версию.



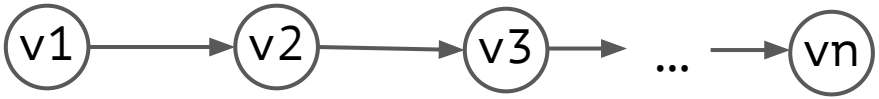
2) Полная персистентность: можем посмотреть любую версию в прошлом и изменить (создавая новую версию) любую версию!



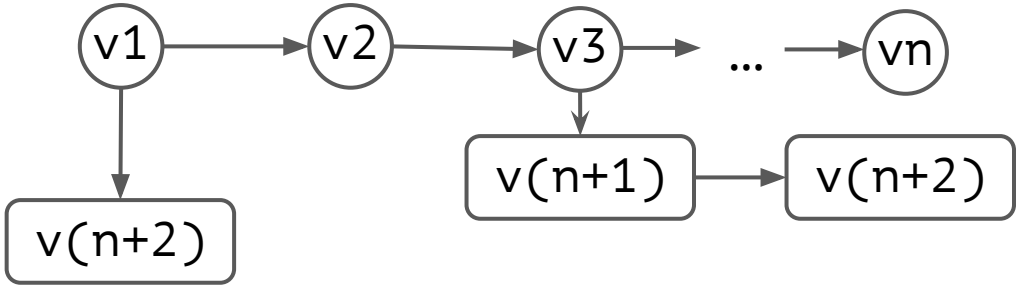

```
add(1, ...)
```

Виды персистентности

1) Частичная персистентность: можем посмотреть на любую версию в прошлом, но изменять (создавая новую версию) можно только последнюю текущую версию.



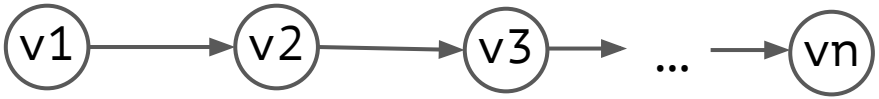
2) Полная персистентность: можем посмотреть любую версию в прошлом и изменить (создавая новую версию) любую версию!



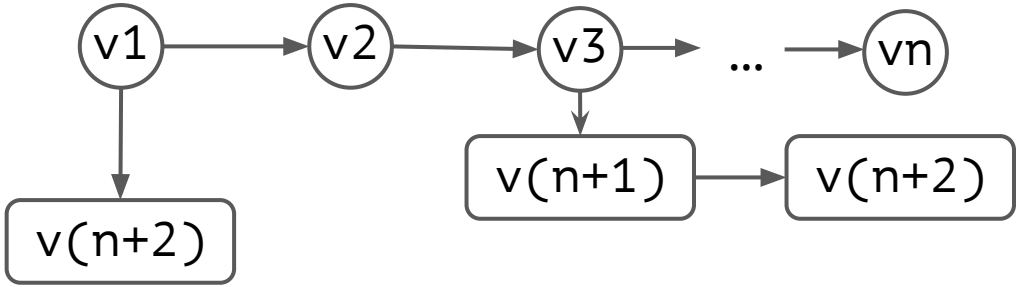
На что похоже?

Виды персистентности

1) Частичная персистентность: можем посмотреть на любую версию в прошлом, но изменять (создавая новую версию) можно только последнюю текущую версию.



2) Полная персистентность: можем посмотреть любую версию в прошлом и изменить (создавая новую версию) любую версию!



На что похоже?

Виды персистентности

Частичная персистентность $\Subset$ Полная персистентность

Виды персистентности

Частичная персистентность $\Subset$ Полная персистентность

Еще есть конфлюэнтные структуры данных (полная персистентность + мерж версий), про них говорить не будем.

Виды персистентности

Частичная персистентность $\Leftarrow$ Полная персистентность

Еще есть конфлюэнтные структуры данных (полная персистентность + мерж версий), про них говорить не будем.

Иногда, сюда же отправляют (чисто) функциональные структуры данных. В них вообще ничего не меняется (даже внутренняя структура), поэтому они по определению персистентны.

Персистентный массив

Пусть есть массив: a0, a1, …, an

Персистентный массив

Пусть есть массив: a0, a1, …, an

Есть операция update(i, value), которая присваивает в i-ый элемент значение value.

Персистентный массив

Пусть есть массив: a0, a1, …, an

Есть операция update(i, value), которая присваивает в i-ый элемент значение value. Это инкрементирует версию.
Изначальная версия - 0.

Персистентный массив

Пусть есть массив: a0, a1, …, an

Есть операция update(i, value), которая присваивает в i-ый элемент значение value. Это инкрементирует версию.
Изначальная версия - 0.

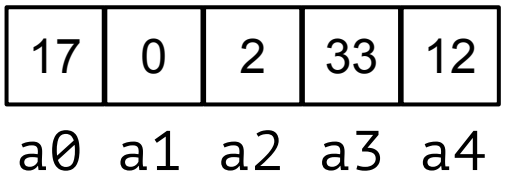
get(t, i): взять i-ый в версии t.

Персистентный массив

Пусть есть массив: a0, a1, …, an

Есть операция update(i, value), которая присваивает в i-ый элемент значение value. Это инкрементирует версию. Изначальная версия - 0.

get(t, i): взять i-ый в версии t. Как решать?



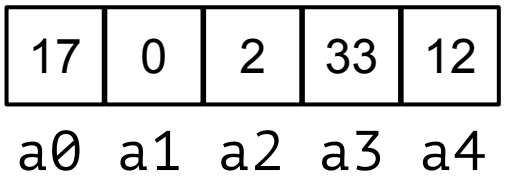
Персистентный массив

Пусть есть массив: a0, a1, …, an

Есть операция update(i, value), которая присваивает в i-ый элемент значение value. Это инкрементирует версию. Изначальная версия - 0.

get(t, i): взять i-ый в версии t. Как решать?

Замечание: всегда можно решать полным копированием.



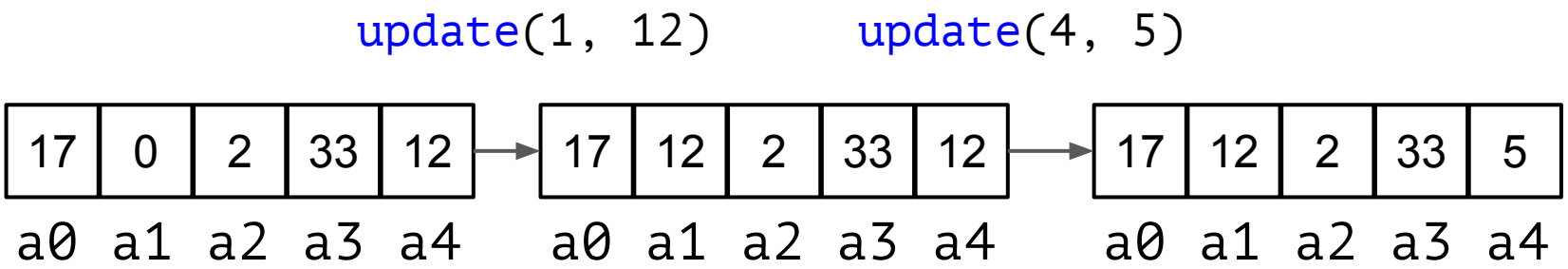
Персистентный массив

Пусть есть массив: a0, a1, …, an

Есть операция update(i, value), которая присваивает в i-ый элемент значение value. Это инкрементирует версию. Изначальная версия - 0.

get(t, i): взять i-ый в версии t. Как решать?

Замечание: всегда можно решать полным копированием.



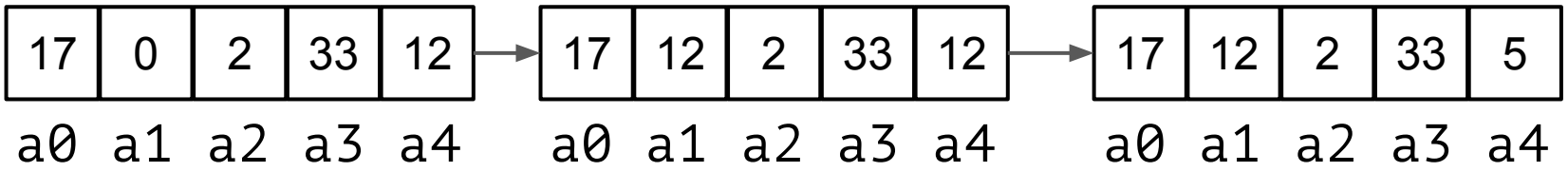
Персистентный массив

Пусть есть массив: a0, a1, …, an

Есть операция update(i, value), которая присваивает в i-ый элемент значение value. Это инкрементирует версию. Изначальная версия - 0.

get(t, i): взять i-ый в версии t. Как решать?

Замечание: всегда можно решать полным копированием. Но это стоит O(N*K) памяти, где K - количество update-ов. Дорого!

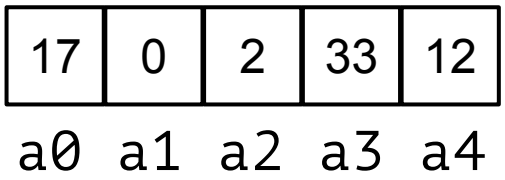


Персистентный массив

Пусть есть массив: a0, a1, …, an

Есть операция update(i, value), которая присваивает в i-ый элемент значение value. Это инкрементирует версию. Изначальная версия - 0.

get(t, i): взять i-ый в версии t. Как решать?

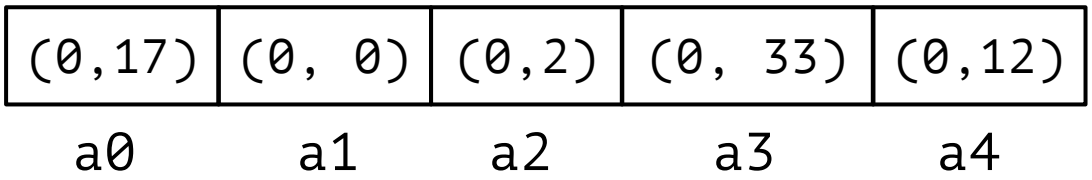


Персистентный массив

Пусть есть массив: a0, a1, …, an

Есть операция update(i, value), которая присваивает в i-ый элемент значение value. Это инкрементирует версию. Изначальная версия - 0.

get(t, i): взять i-ый в версии t. Как решать? Для каждого элемента хранить все версии!

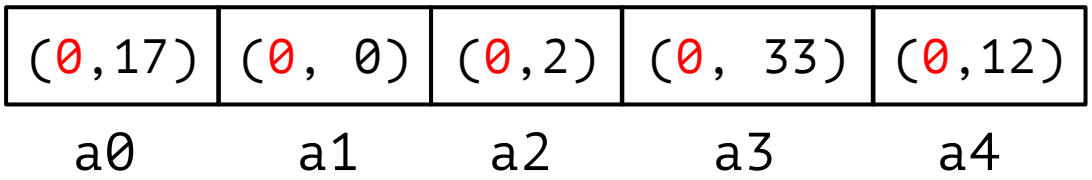


Персистентный массив

Пусть есть массив: a0, a1, …, an

Есть операция update(i, value), которая присваивает в i-ый элемент значение value. Это инкрементирует версию. Изначальная версия - 0.

get(t, i): взять i-ый в версии t. Как решать? Для каждого элемента хранить все версии!

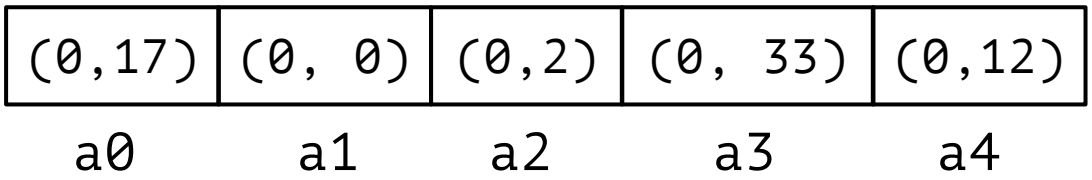


Персистентный массив

Пусть есть массив: a0, a1, …, an

Есть операция update(i, value), которая присваивает в i-ый элемент значение value. Это инкрементирует версию. Изначальная версия - 0.

get(t, i): взять i-ый в версии t. Как решать? Для каждого элемента хранить все версии!



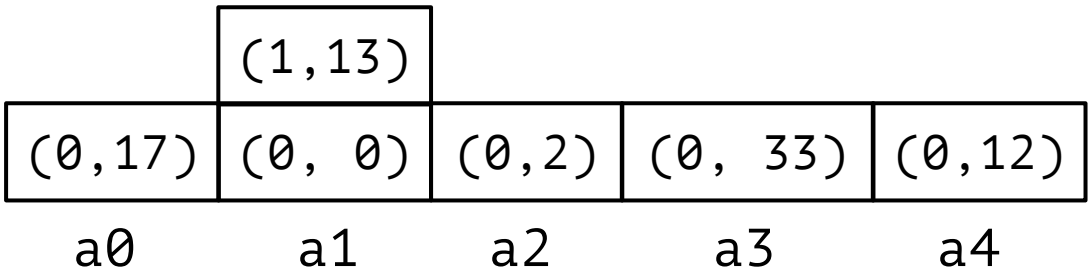
update(1, 13)

Персистентный массив

Пусть есть массив: a0, a1, …, an

Есть операция update(i, value), которая присваивает в i-ый элемент значение value. Это инкрементирует версию. Изначальная версия - 0.

get(t, i): взять i-ый в версии t. Как решать? Для каждого элемента хранить все версии!



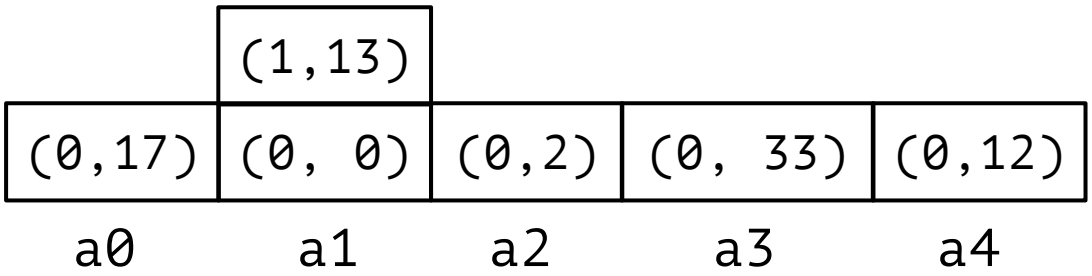
update(1, 13)

Персистентный массив

Пусть есть массив: a0, a1, …, an

Есть операция update(i, value), которая присваивает в i-ый элемент значение value. Это инкрементирует версию. Изначальная версия - 0.

get(t, i): взять i-ый в версии t. Как решать? Для каждого элемента хранить все версии!



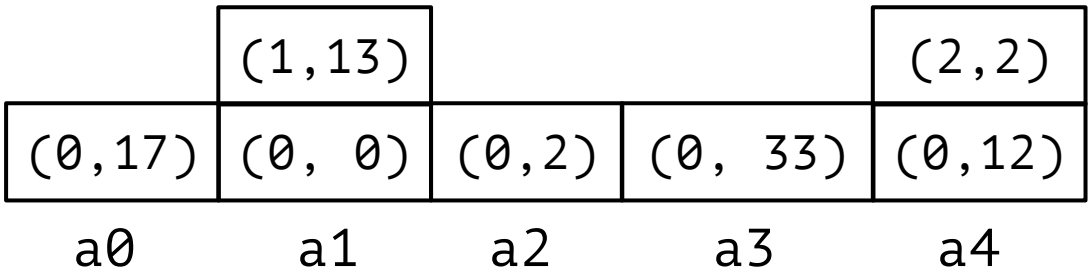
update(4, 2)

Персистентный массив

Пусть есть массив: a0, a1, …, an

Есть операция update(i, value), которая присваивает в i-ый элемент значение value. Это инкрементирует версию. Изначальная версия - 0.

get(t, i): взять i-ый в версии t. Как решать? Для каждого элемента хранить все версии!



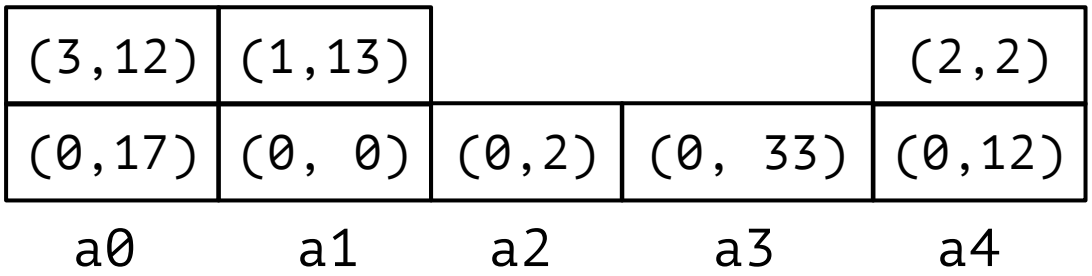
update(4, 2)

Персистентный массив

Пусть есть массив: a0, a1, …, an

Есть операция update(i, value), которая присваивает в i-ый элемент значение value. Это инкрементирует версию. Изначальная версия - 0.

get(t, i): взять i-ый в версии t. Как решать? Для каждого элемента хранить все версии!



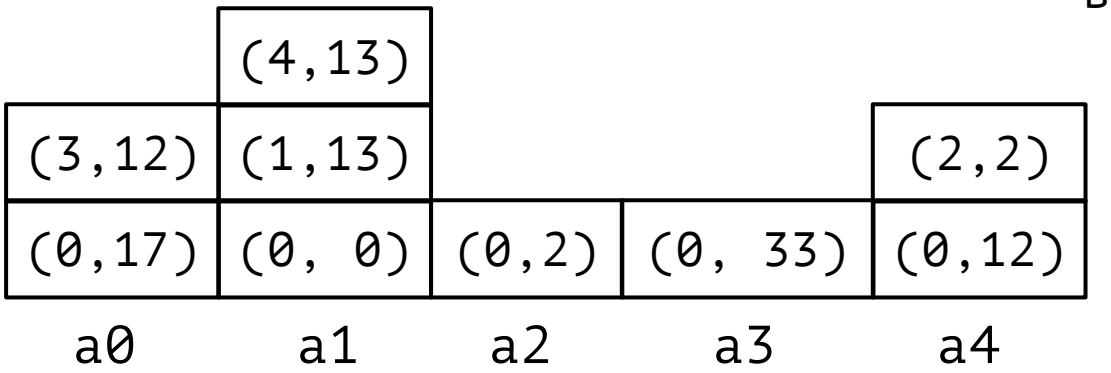
update(0, 12)

Персистентный массив

Пусть есть массив: a0, a1, …, an

Есть операция update(i, value), которая присваивает в i-ый элемент значение value. Это инкрементирует версию. Изначальная версия - 0.

get(t, i): взять i-ый в версии t. Как решать? Для каждого элемента хранить все версии!



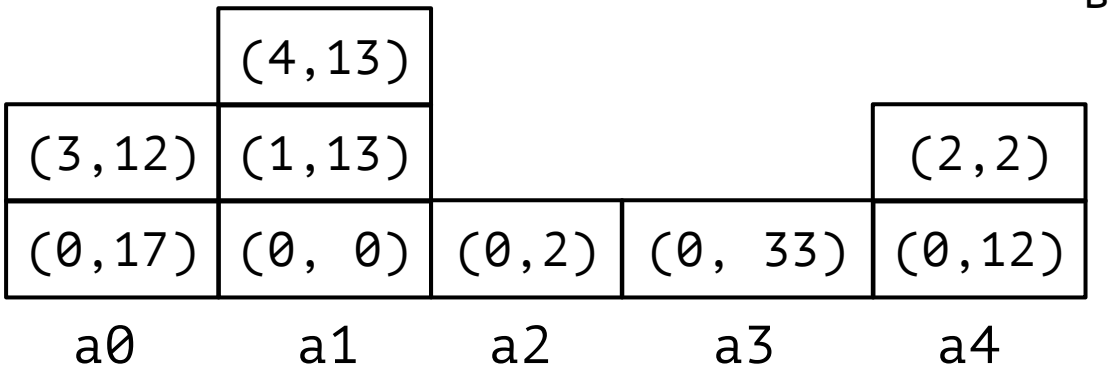
update(1, 16)

Персистентный массив

Пусть есть массив: a0, a1, …, an

Есть операция update(i, value), которая присваивает в i-ый элемент значение value. Это инкрементирует версию. Изначальная версия - 0.

get(t, i): взять i-ый в версии t. Как решать? Для каждого элемента хранить все версии!



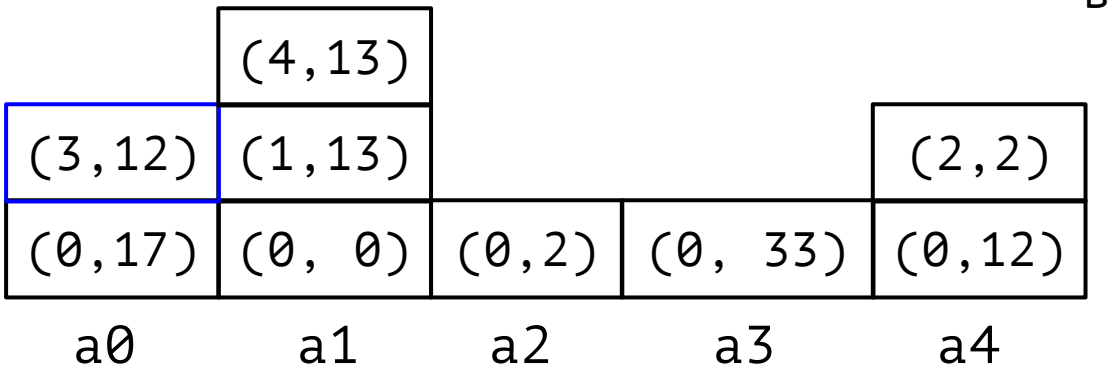
get(3, 0)?

Персистентный массив

Пусть есть массив: a0, a1, …, an

Есть операция update(i, value), которая присваивает в i-ый элемент значение value. Это инкрементирует версию. Изначальная версия - 0.

get(t, i): взять i-ый в версии t. Как решать? Для каждого элемента хранить все версии!



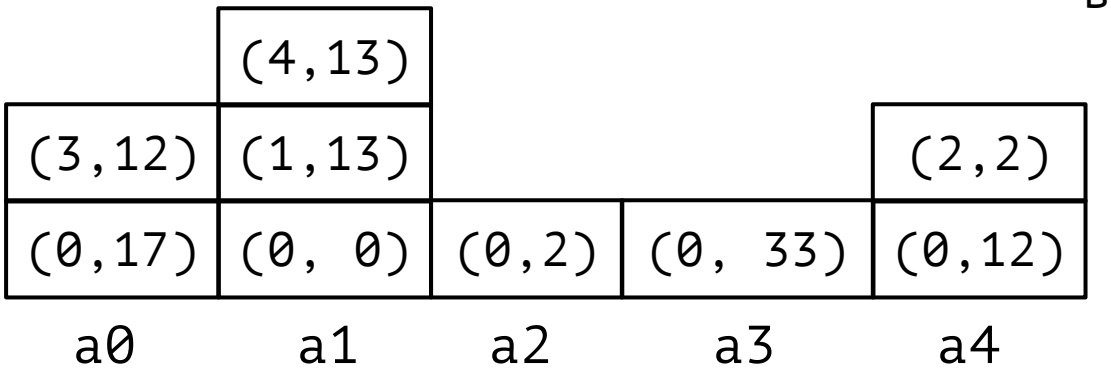
get(3, 0)?

Персистентный массив

Пусть есть массив: a0, a1, …, an

Есть операция update(i, value), которая присваивает в i-ый элемент значение value. Это инкрементирует версию. Изначальная версия - 0.

get(t, i): взять i-ый в версии t. Как решать? Для каждого элемента хранить все версии!



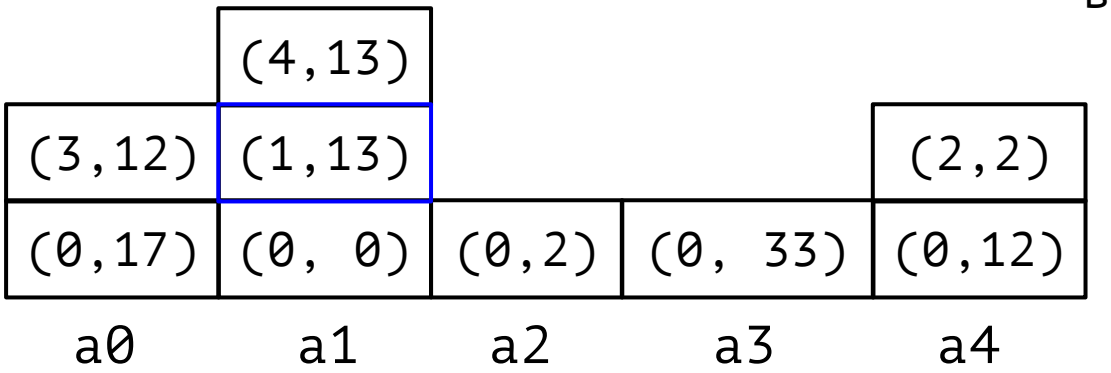
get(2, 1)?

Персистентный массив

Пусть есть массив: a0, a1, …, an

Есть операция update(i, value), которая присваивает в i-ый элемент значение value. Это инкрементирует версию. Изначальная версия - 0.

get(t, i): взять i-ый в версии t. Как решать? Для каждого элемента хранить все версии!



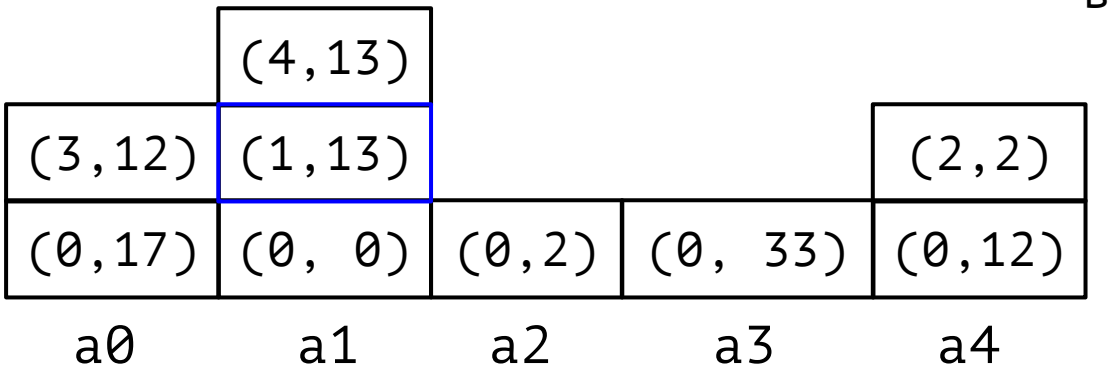
get(2, 1)?

Персистентный массив

Пусть есть массив: a0, a1, …, an

Есть операция update(i, value), которая присваивает в i-ый элемент значение value. Это инкрементирует версию. Изначальная версия - 0.

get(t, i): взять i-ый в версии t. Как решать? Для каждого элемента хранить все версии!



При каждой операции находим бинарным поиском ближайшую версию <= нужной.

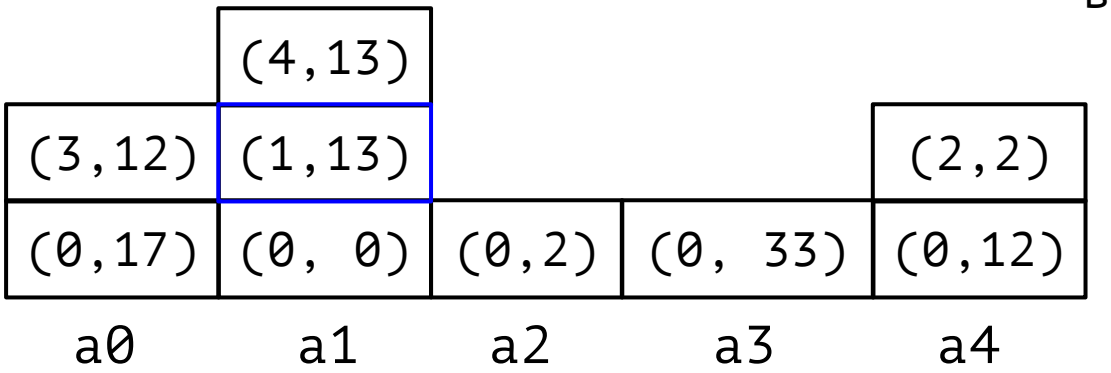
Персистентный массив

Время: ?
Память: ?

Пусть есть массив: a0, a1, …, an

Есть операция update(i, value), которая присваивает в i-ый элемент значение value. Это инкрементирует версию. Изначальная версия - 0.

get(t, i): взять i-ый в версии t. Как решать? Для каждого элемента хранить все версии!



При каждой операции находим бинарным поиском ближайшую версию <= нужной.

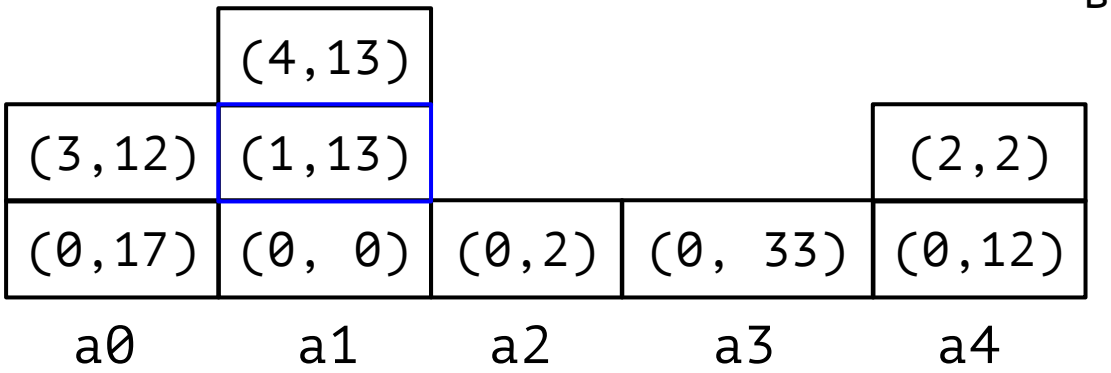
Персистентный массив

Время: get(...) -> logK
Память: N + K
K - количество запросов

Пусть есть массив: a0, a1, …, an

Есть операция update(i, value), которая присваивает в i-ый элемент значение value. Это инкрементирует версию.
Изначальная версия - 0.

get(t, i): взять i-ый в версии t. Как решать? Для каждого элемента хранить все версии!



При каждой операции находим бинарным поиском ближайшую версию <= нужной.

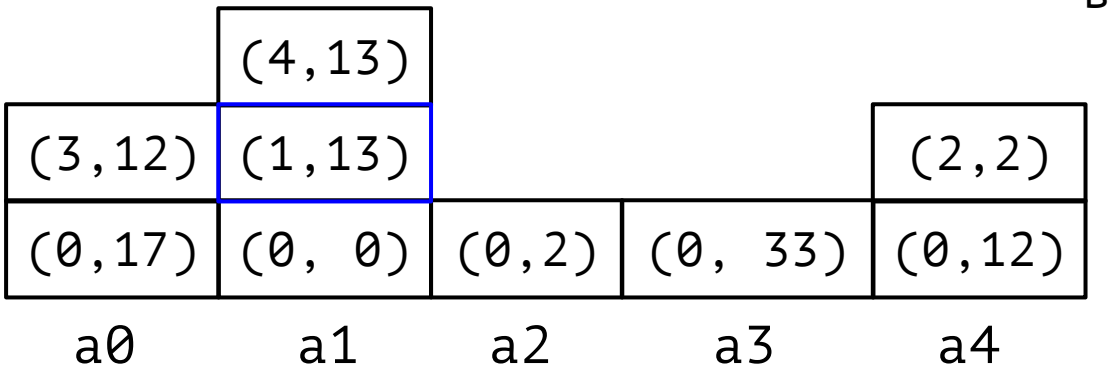
Персистентный массив

Время: get(...) -> logK
Память: N + K
K - количество запросов
Персистентность?

Пусть есть массив: a0, a1, …, an

Есть операция update(i, value), которая присваивает в i-ый элемент значение value. Это инкрементирует версию. Изначальная версия - 0.

get(t, i): взять i-ый в версии t. Как решать? Для каждого элемента хранить все версии!



При каждой операции находим бинарным поиском ближайшую версию <= нужной.

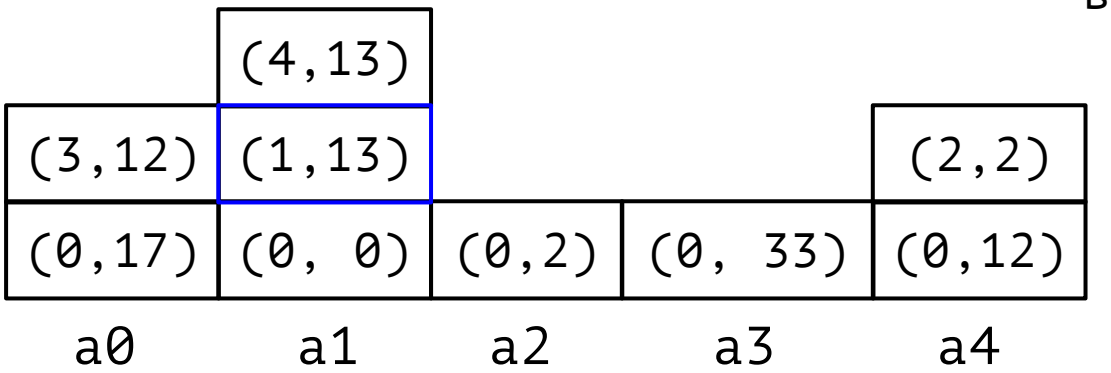
Персистентный массив

Время: get(...) -> logK
Память: N + K
K - количество запросов
Частичная персистентность

Пусть есть массив: a0, a1, …, an

Есть операция update(i, value), которая присваивает в i-ый элемент значение value. Это инкрементирует версию.
Изначальная версия - 0.

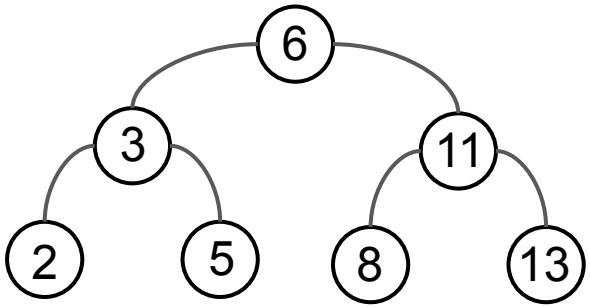
get(t, i): взять i-ый в версии t. Как решать? Для каждого элемента хранить все версии!



При каждой операции находим бинарным поиском ближайшую версию <= нужной.

Персистентные деревья

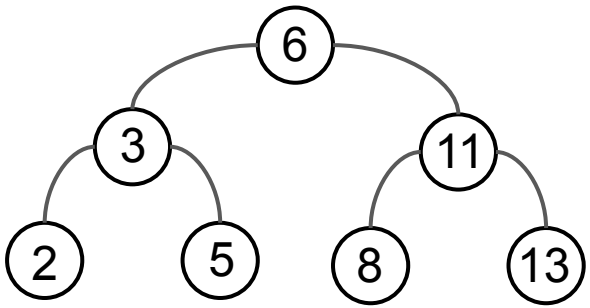
Пусть есть бинарное дерево поиска



Персистентные деревья

Пусть есть бинарное дерево поиска. Реализовать операции:

insert(value), при этом каждый insert инкрементирует версию
find(t, value), где t - это номер версии

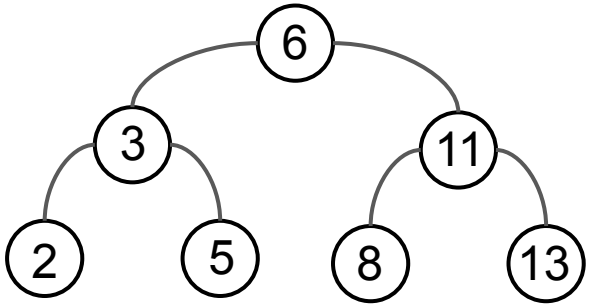


Персистентные деревья

Пусть есть бинарное дерево поиска. Реализовать операции:

insert(value), при этом каждый insert инкрементирует версию
find(t, value), где t - это номер версии

Т.е. опять путешествуем во времени, пока только для поиска

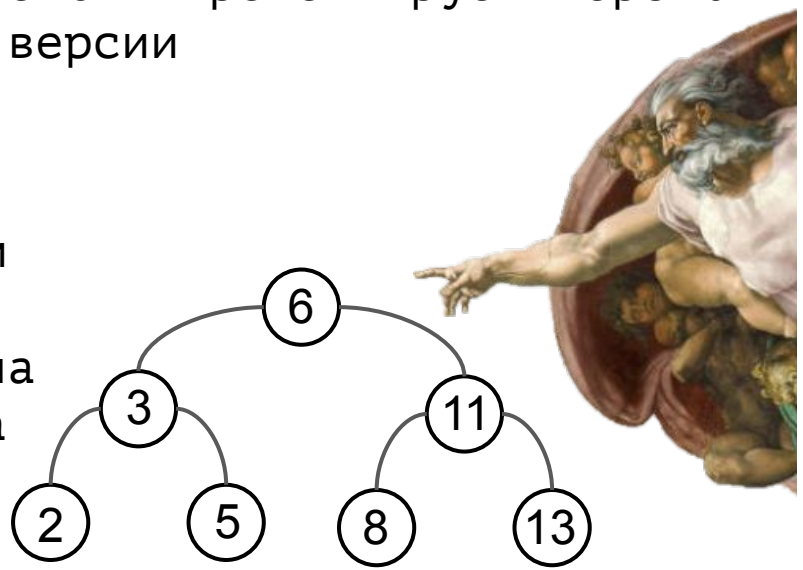


Персистентные деревья

Пусть есть бинарное дерево поиска. Реализовать операции:

insert(value), при этом каждый insert инкрементирует версию
find(t, value), где t - это номер версии

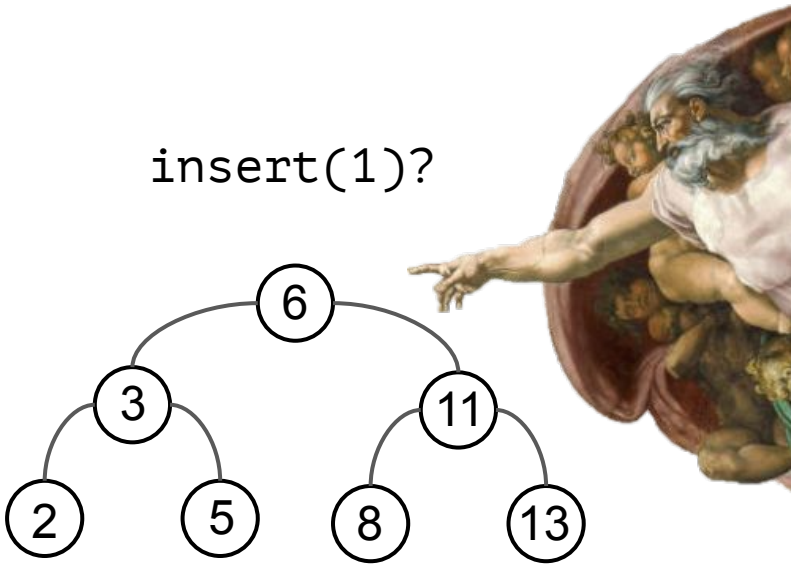
Для простоты будем считать, что эта версия существовала от сотворения мира



Персистентные деревья

Пусть есть бинарное дерево поиска. Реализовать операции:

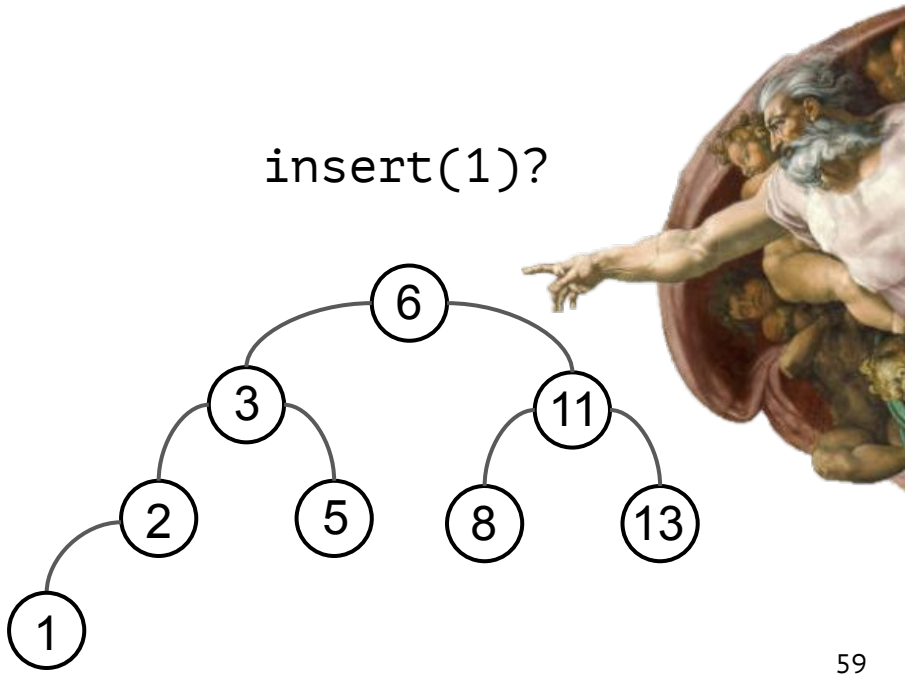
```
insert(value)
find(t, value)
```



Персистентные деревья

Пусть есть бинарное дерево поиска. Реализовать операции:

```
insert(value)
find(t, value)
```

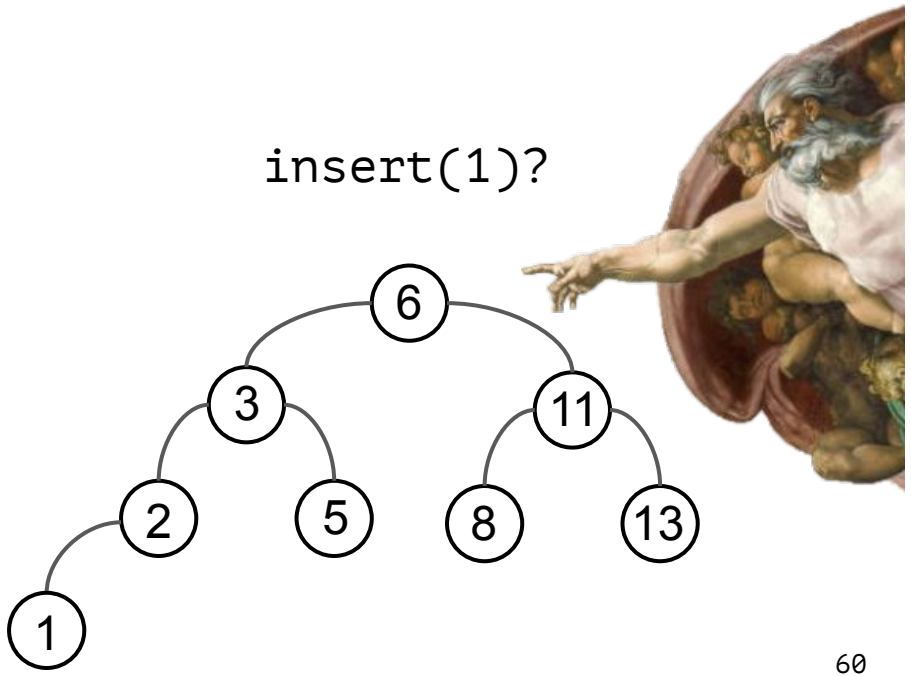


хотелось бы
вставить сюда

Персистентные деревья

Пусть есть бинарное дерево поиска. Реализовать операции:

```
insert(value)
find(t, value)
```

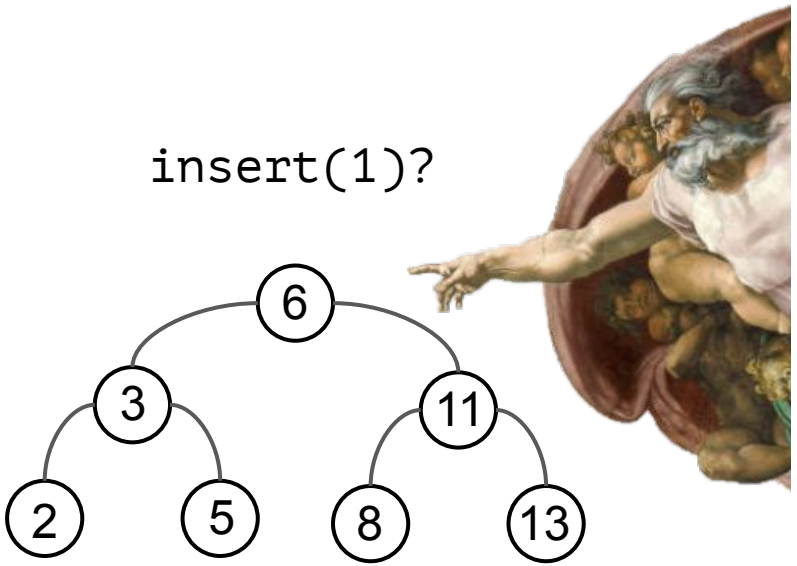


хотелось бы
вставить сюда,
но тогда нужно
менять предка

Персистентные деревья

Пусть есть бинарное дерево поиска. Реализовать операции:

```
insert(value)
find(t, value)
```

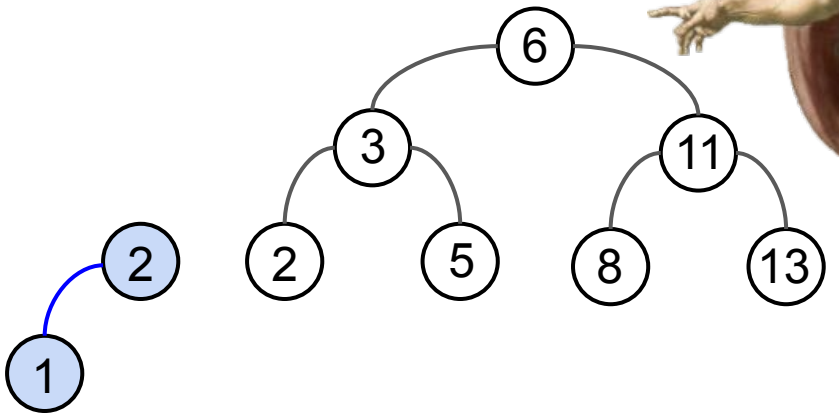


Персистентные деревья

Пусть есть бинарное дерево поиска. Реализовать операции:

```
insert(value)
find(t, value)
```

insert(1)?



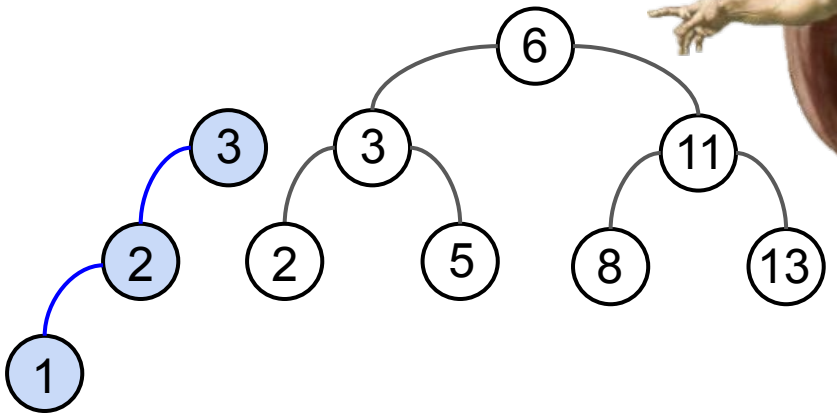
менять предка мы не будем,
вместо этого мы его скопируем

Персистентные деревья

Пусть есть бинарное дерево поиска. Реализовать операции:

```
insert(value)
find(t, value)
```

insert(1)?



менять предка мы не будем,
вместо этого мы его скопируем

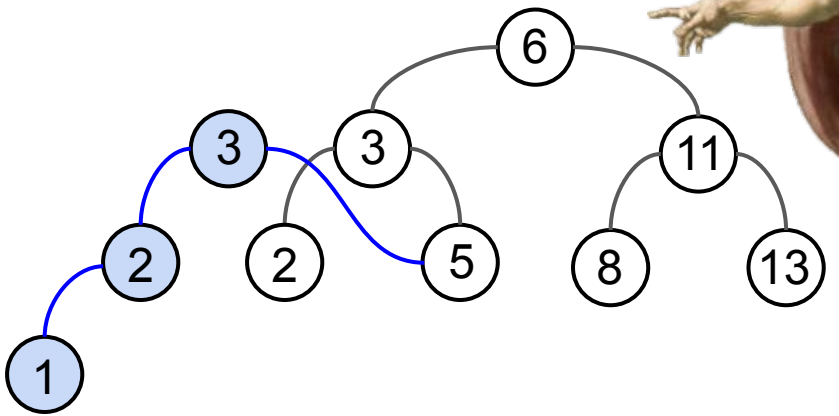
но придется повторить и выше

Персистентные деревья

Пусть есть бинарное дерево поиска. Реализовать операции:

```
insert(value)
find(t, value)
```

insert(1)?



менять предка мы не будем,
вместо этого мы его скопируем

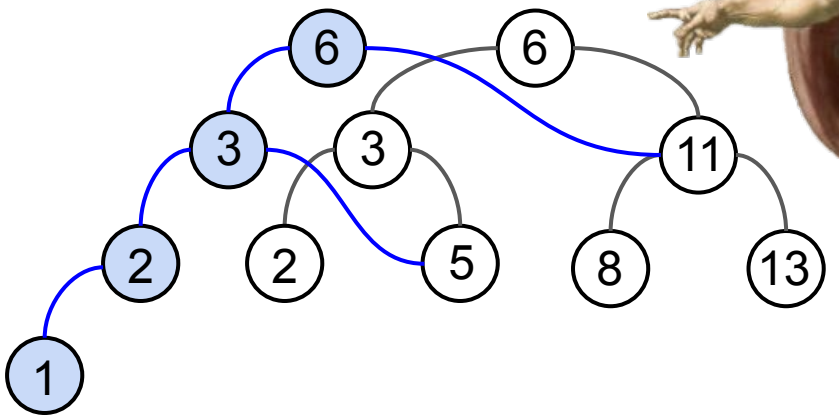
но придется повторить и выше,
при этом где можем, ссылаемся
на старые вершины

Персистентные деревья

Пусть есть бинарное дерево поиска. Реализовать операции:

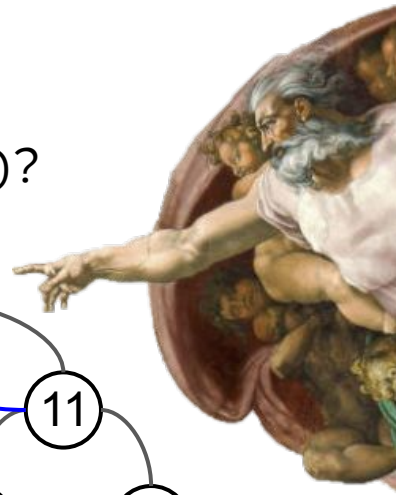
```
insert(value)
find(t, value)
```

insert(1)?



менять предка мы не будем,
вместо этого мы его скопируем

но придется повторить и выше,
при этом где можем, ссылаемся
на старые вершины

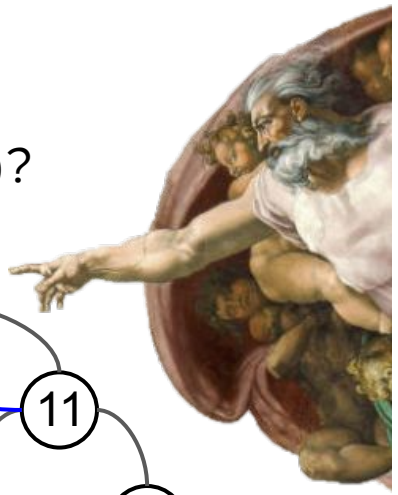


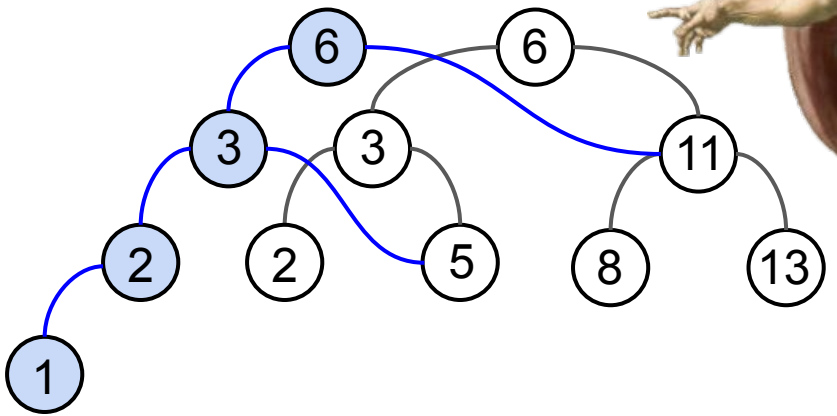
Персистентные деревья

Пусть есть бинарное дерево поиска. Реализовать операции:

```
insert(value)
find(t, value)
```

insert(12)?





менять предка мы не будем,
вместо этого мы его скопируем

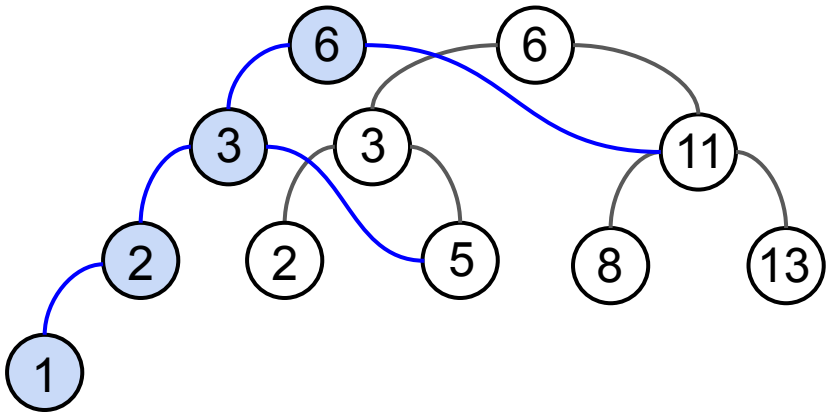
но придется повторить и выше,
при этом где можем, ссылаемся
на старые вершины

Персистентные деревья

Пусть есть бинарное дерево поиска. Реализовать операции:

```
insert(value)
find(t, value)
```

insert(12)?



менять предка мы не будем,
вместо этого мы его скопируем

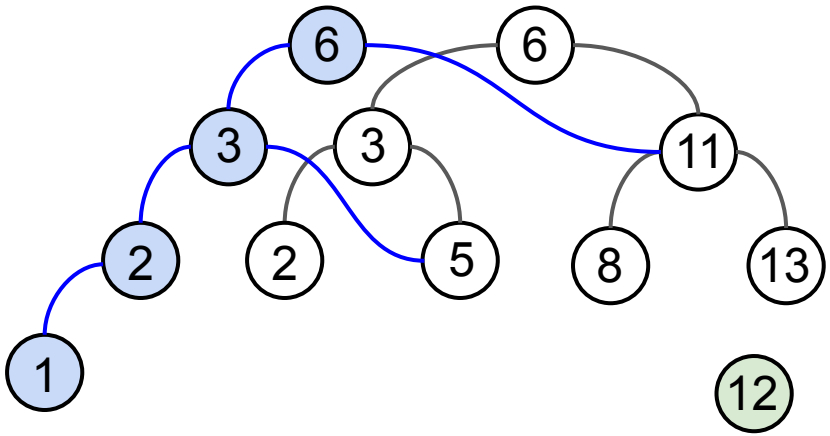
но придется повторить и выше,
при этом где можем, ссылаемся
на старые вершины

Персистентные деревья

Пусть есть бинарное дерево поиска. Реализовать операции:

```
insert(value)
find(t, value)
```

insert(12)?



менять предка мы не будем,
вместо этого мы его скопируем

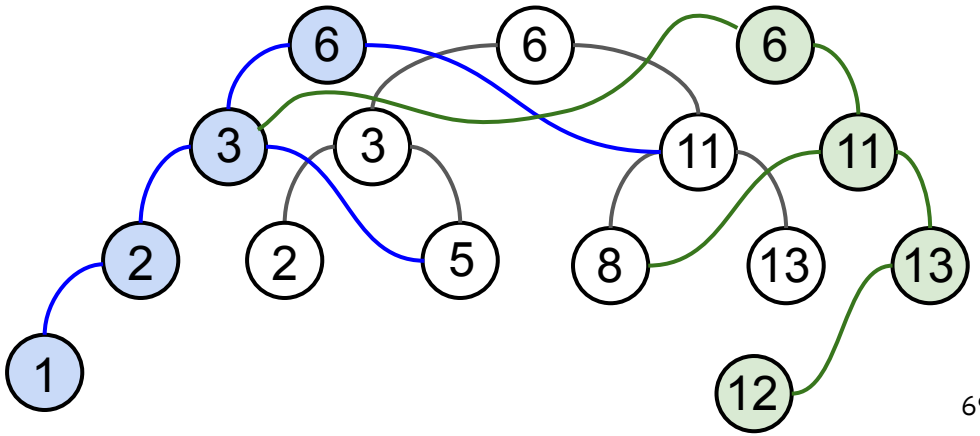
но придется повторить и выше,
при этом где можем, ссылаемся
на старые вершины

Персистентные деревья

Пусть есть бинарное дерево поиска. Реализовать операции:

```
insert(value)
find(t, value)
```

insert(12)?



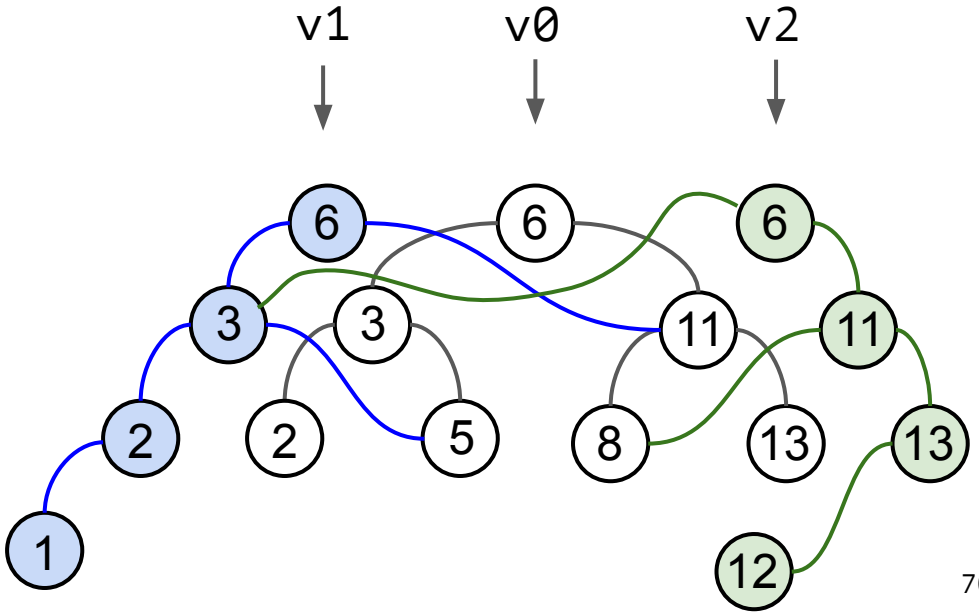
менять предка мы не будем,
вместо этого мы его скопируем

но придется повторить и выше,
при этом где можем, ссылаемся
на старые вершины

Персистентные деревья

Пусть есть бинарное дерево поиска. Реализовать операции:

```
insert(value)
find(t, value)
```

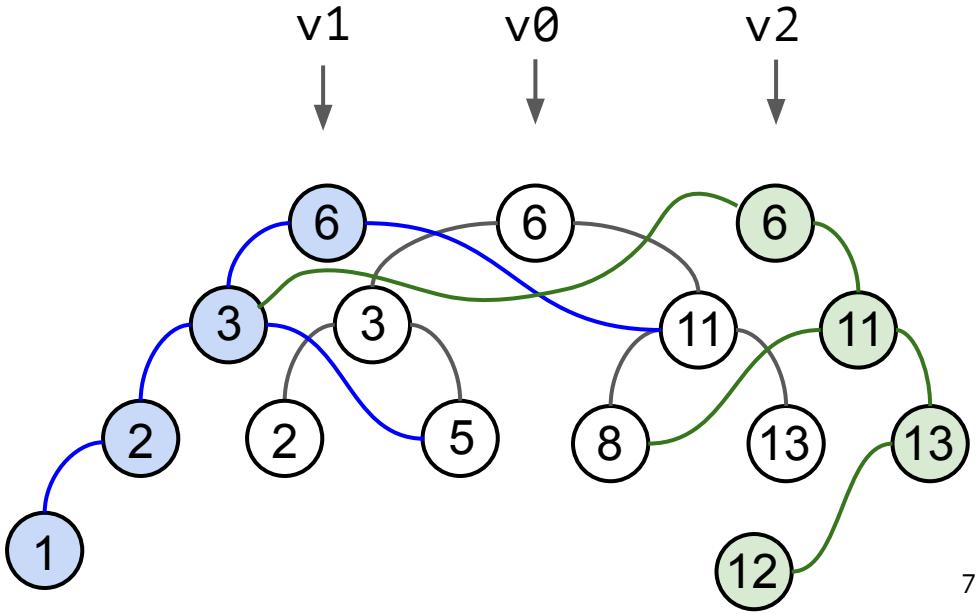


Персистентные деревья

Пусть есть бинарное дерево поиска. Реализовать операции:

```
insert(value)
find(t, value)
```

будем хранить в массиве v указатели
на root-ы деревьев разных версий



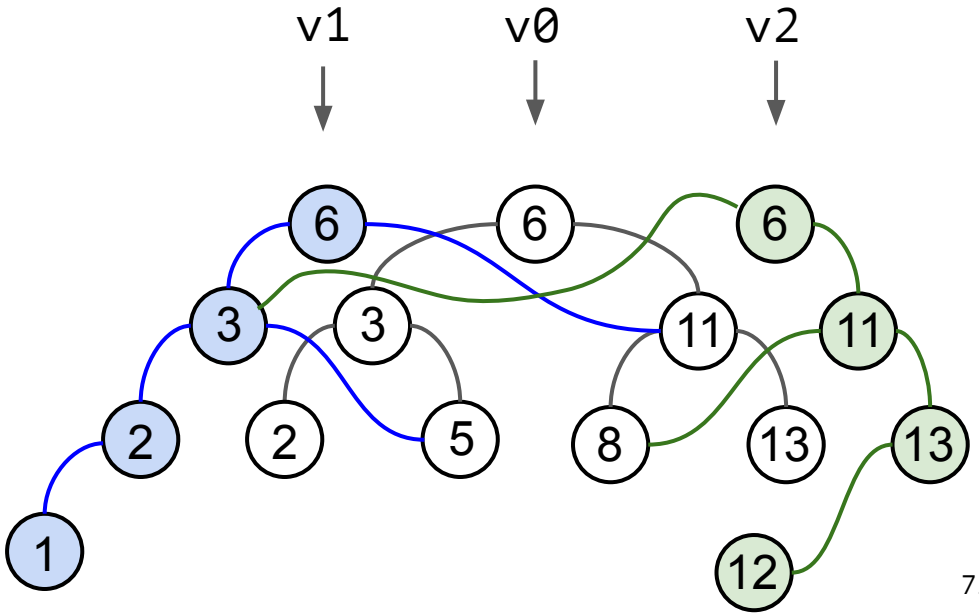
Персистентные деревья

Пусть есть бинарное дерево поиска. Реализовать операции:

```
insert(value)
find(t, value)
```

```
find(2, 1)?
```

будем хранить в массиве v указатели
на root-ы деревьев разных версий

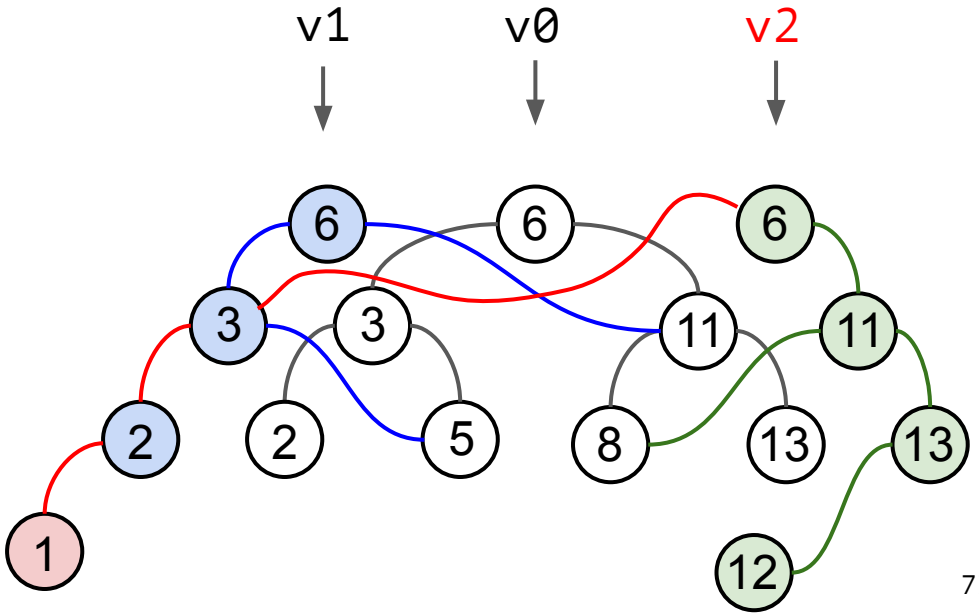


Персистентные деревья

Пусть есть бинарное дерево поиска. Реализовать операции:

```
insert(value)
find(t, value)
```

будем хранить в массиве v указатели на root-ы деревьев разных версий




```
find(2, 1)?
```

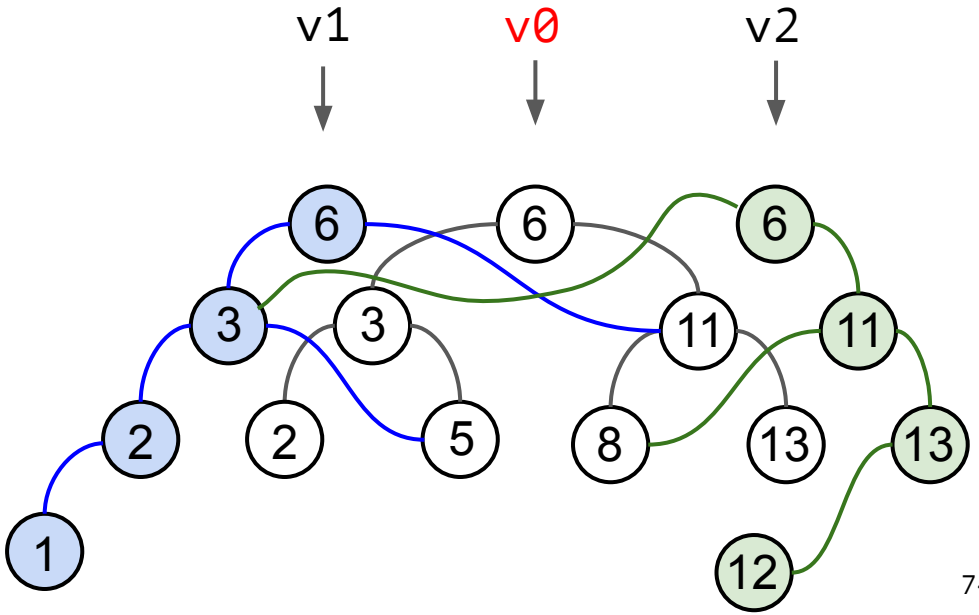
Запускаем обычный поиск элемента в дереве с корнем v[2]

Персистентные деревья

Пусть есть бинарное дерево поиска. Реализовать операции:

```
insert(value)
find(t, value)
```

будем хранить в массиве v указатели на root-ы деревьев разных версий




```
find(0, 1)?
```

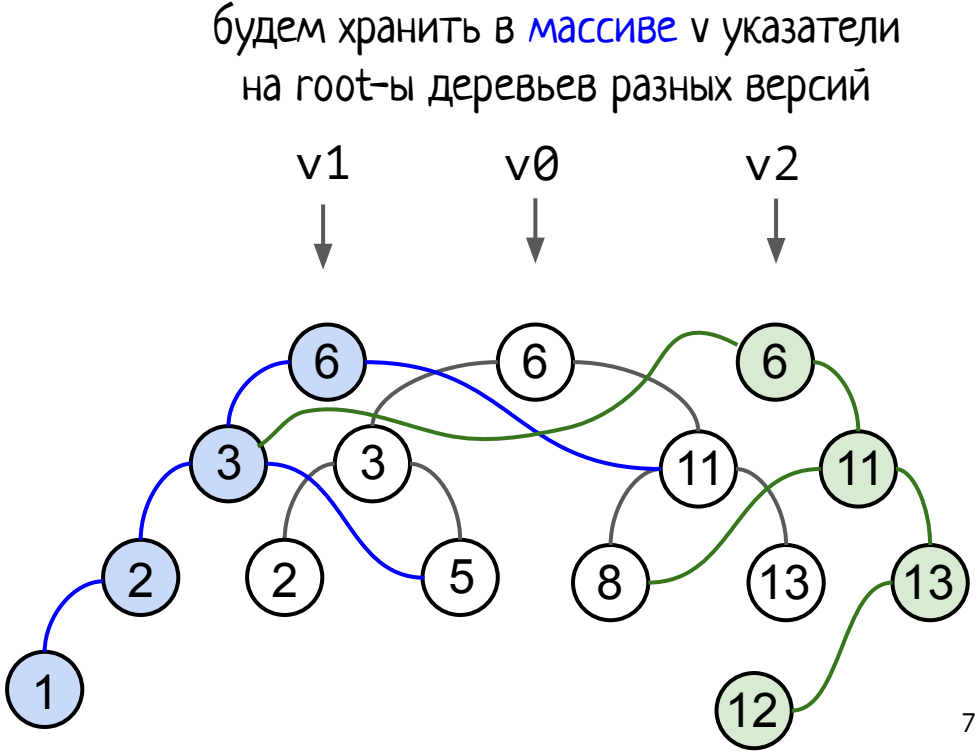
Запускаем обычный поиск элемента в дереве с корнем v[0]

Персистентные деревья

Пусть есть бинарное дерево поиска. Реализовать операции:

```
insert(value)
find(t, value)
```

Такая техника называется "копирование путей".



Персистентные деревья

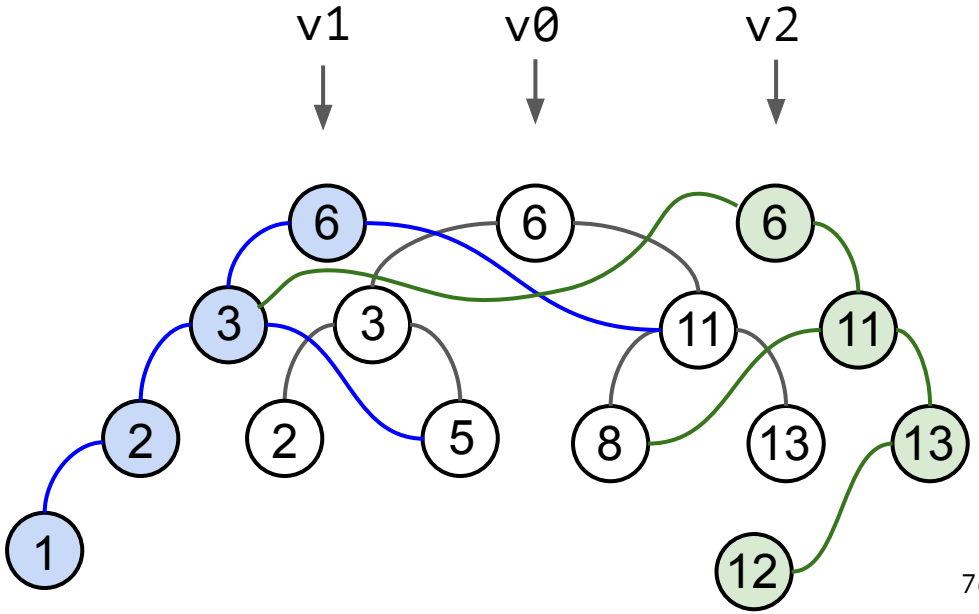
Пусть есть бинарное дерево поиска. Реализовать операции:

```
insert(value)
find(t, value)
```

Такая техника называется "копирование путей".

Время: ?
Память: ?

будем хранить в массиве v указатели на root-ы деревьев разных версий



Персистентные деревья

Пусть есть бинарное дерево поиска. Реализовать операции:

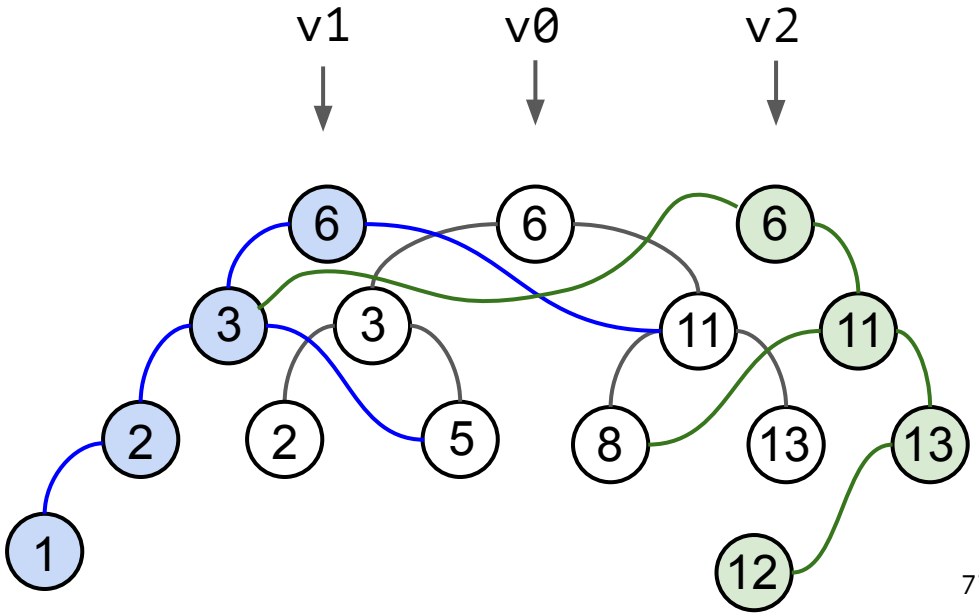
```
insert(value)
find(t, value)
```

Такая техника называется "копирование путей".

h - высота дерева

Время: h(N)
Память: ?

будем хранить в массиве v указатели на root-ы деревьев разных версий



Персистентные деревья

Пусть есть бинарное дерево поиска. Реализовать операции:

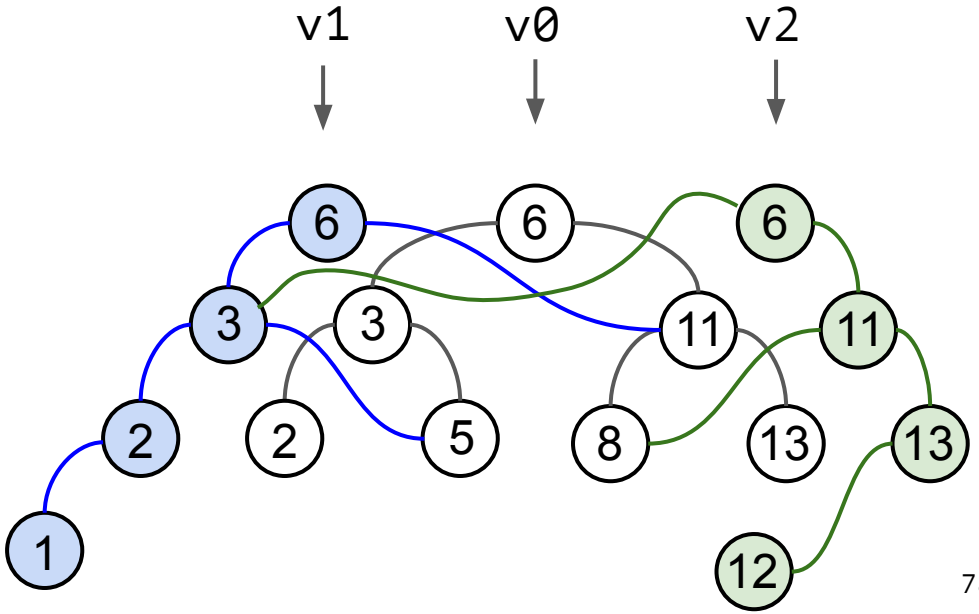
```
insert(value)
find(t, value)
```

Такая техника называется "копирование путей".

h - высота дерева

Время: h(N)
Память: N + K*h(N)
K - количество insert

будем хранить в массиве v указатели на root-ы деревьев разных версий



Персистентные деревья

Пусть есть бинарное дерево поиска. Реализовать операции:

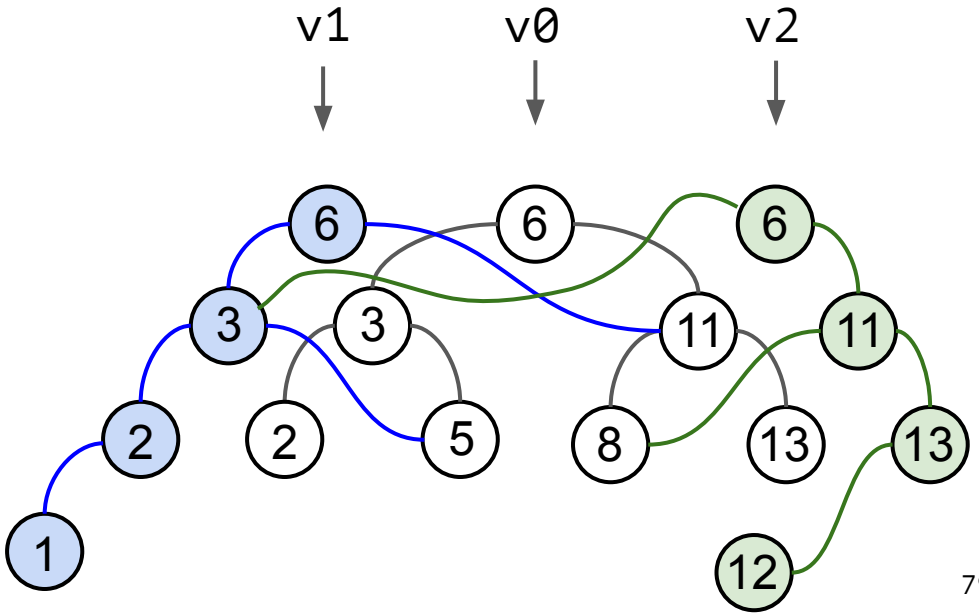
```
insert(value)
find(t, value)
```

Такая техника называется "копирование путей".

h - высота дерева

Время: h(N)
Память: N + K*h(N)
K - количество insert
Какая персистентность?

будем хранить в массиве v указатели на root-ы деревьев разных версий



Персистентные деревья

Пусть есть бинарное дерево поиска. Реализовать операции:

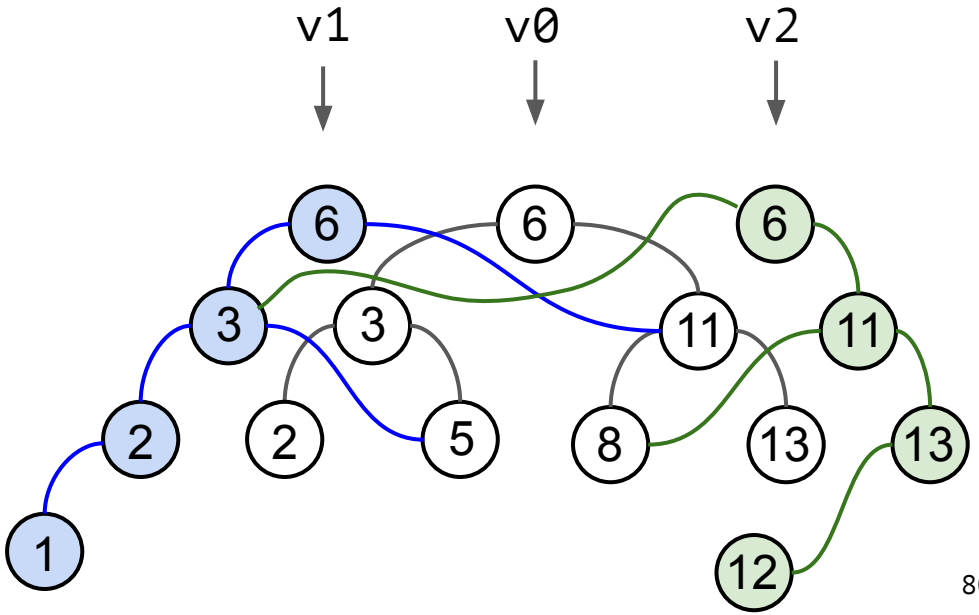
```
insert(value)
find(t, value)
```

Такая техника называется "копирование путей".

h - высота дерева

Время: h(N)
Память: N + K*h(N)
K - количество insert
Частичная персистентность

будем хранить в массиве v указатели на root-ы деревьев разных версий



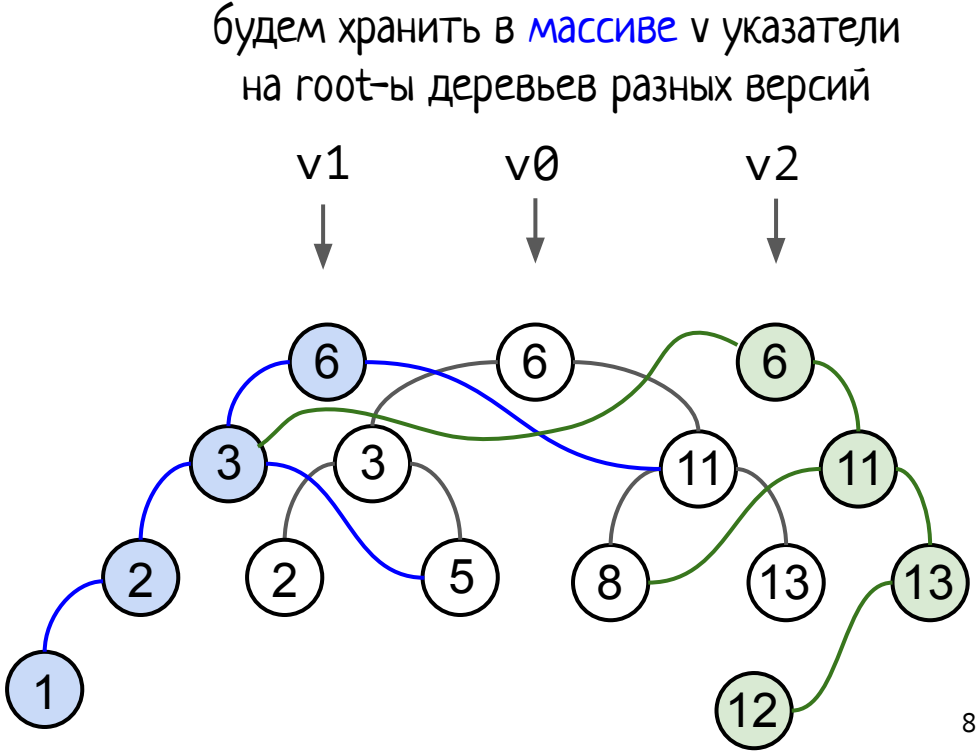
Персистентные деревья

Пусть есть бинарное дерево поиска. Реализовать операции:

```
insert(t, value)
find(t, value)
```

Такая техника называется "копирование путей".

insert(0, 0)?



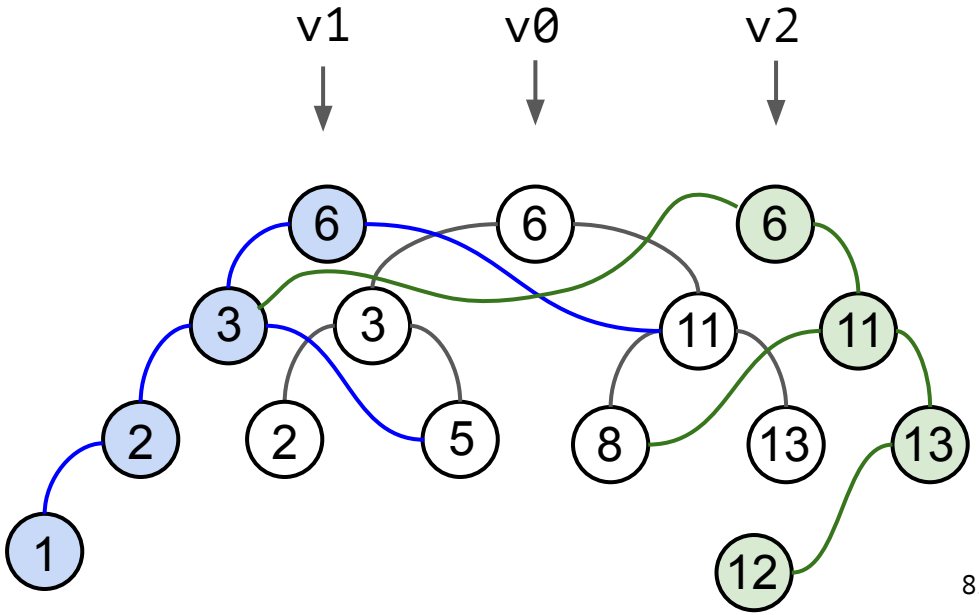
Персистентные деревья

Пусть есть бинарное дерево поиска. Реализовать операции:

```
insert(t, value)
find(t, value)
```

Такая техника называется "копирование путей".

будем хранить в массиве v указатели на root-ы деревьев разных версий




```
insert(0, 0)?
```

Все тоже самое! Начинаем с v[0] и вставляем (с копированием путей)

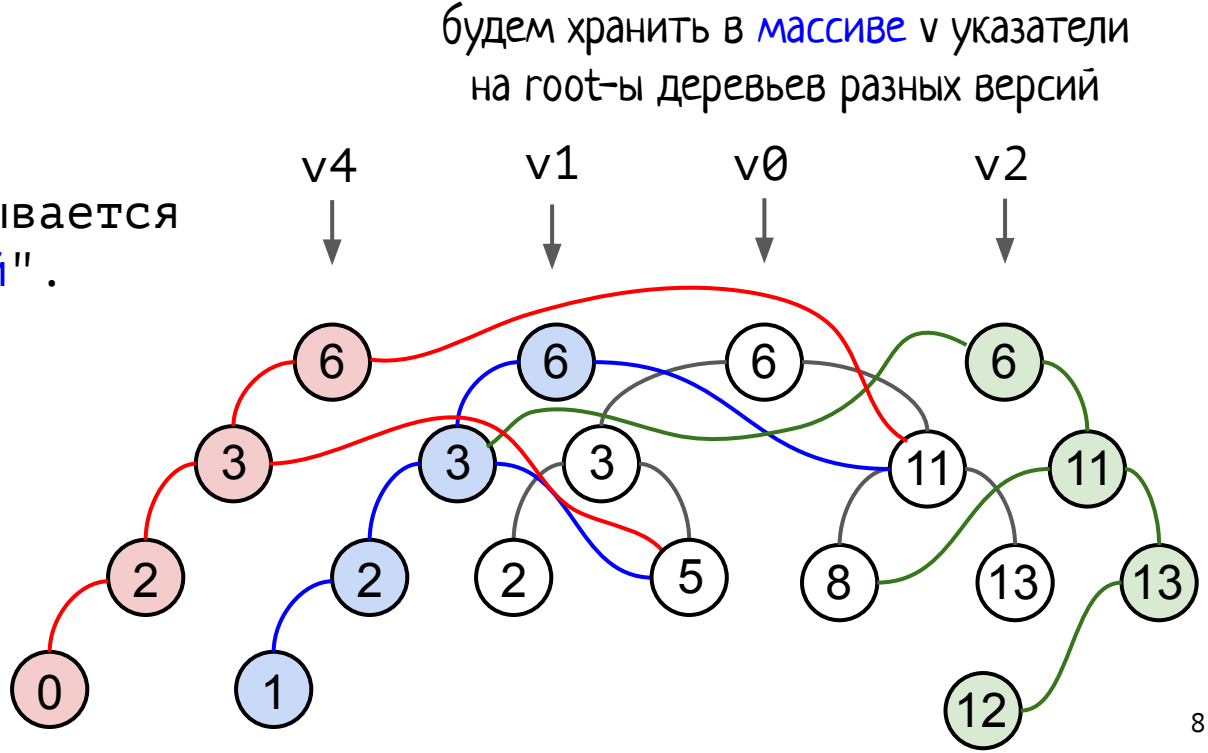
Персистентные деревья

Пусть есть бинарное дерево поиска. Реализовать операции:

```
insert(t, value)
find(t, value)
```

Такая техника называется "копирование путей".

insert(0, 0)?



Персистентные деревья

Пусть есть бинарное дерево поиска. Реализовать операции:

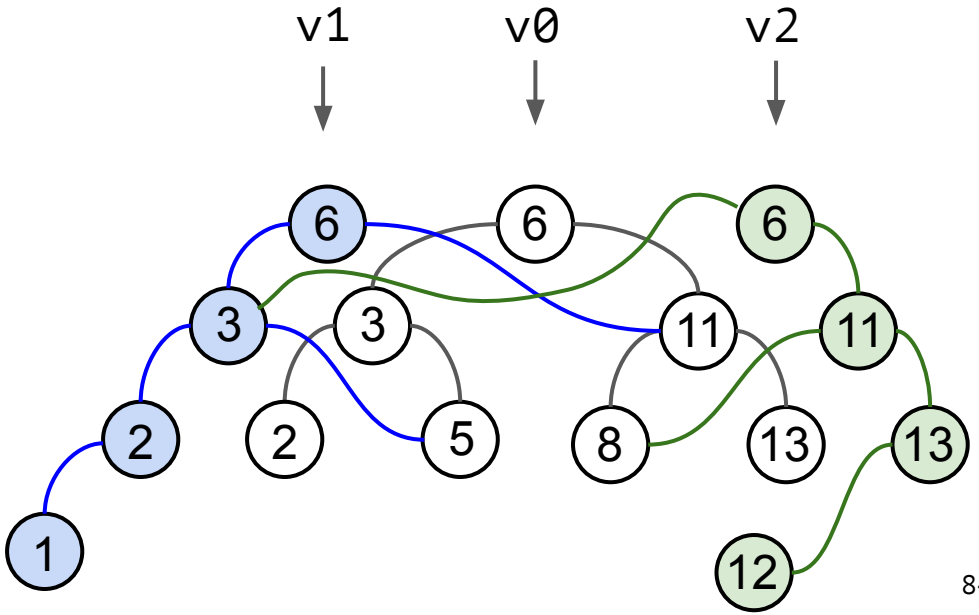
```
insert(value)
find(t, value)
```

Такая техника называется "копирование путей".

h - высота дерева

Время: h(N)
Память: N + K*h(N)
K - количество insert
Частичная персистентность

будем хранить в массиве v указатели на root-ы деревьев разных версий



Персистентные деревья

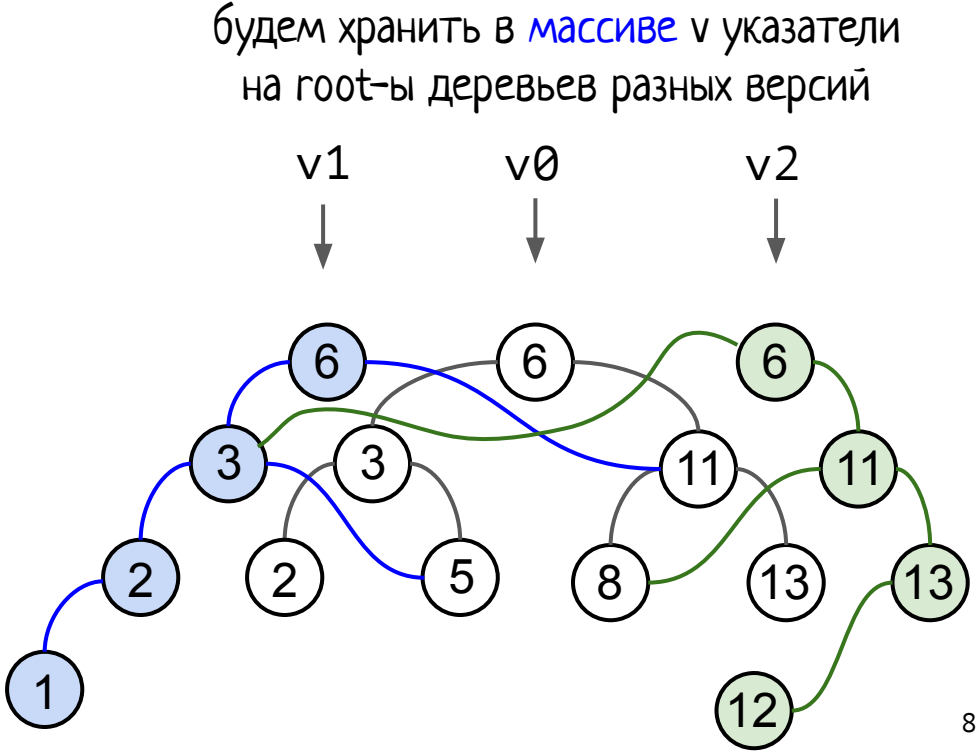
Пусть есть бинарное дерево поиска. Реализовать операции:

```
insert(t, value)
find(t, value)
```

Такая техника называется "копирование путей".

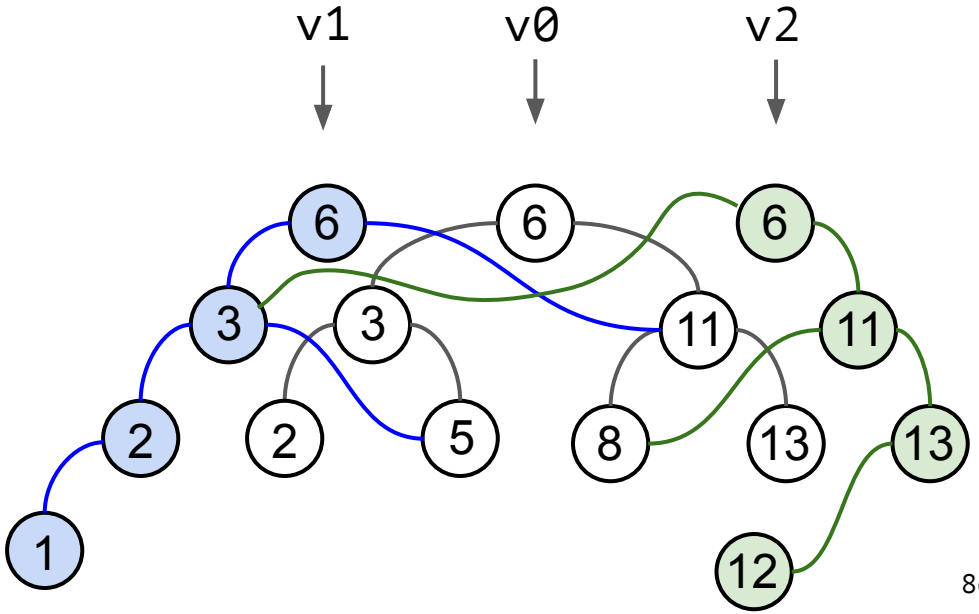
h - высота дерева

Время: h(N)
Память: N + K*h(N)
K - количество insert
Полная персистентность!



Персистентные деревья

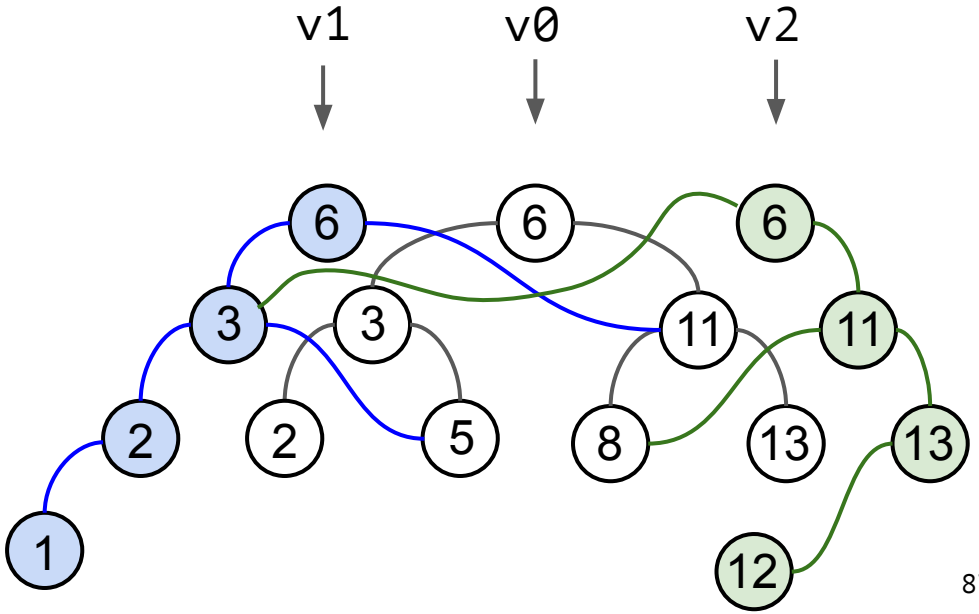
Такая техника называется "копирование путей".



Персистентные деревья

Такая техника называется "копирование путей".

А так только с BST можно?

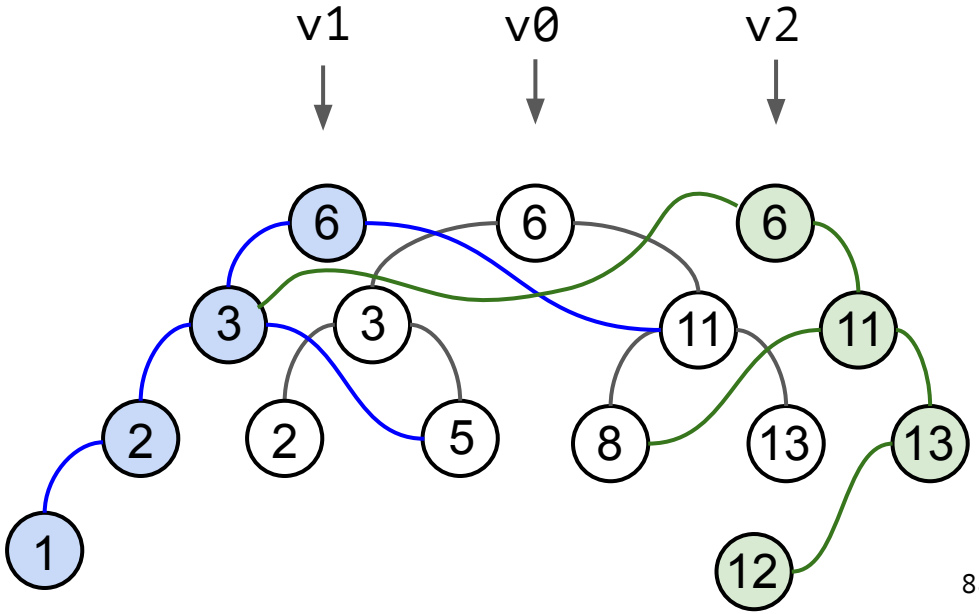


Персистентные деревья

Такая техника называется "копирование путей".

А так только с BST можно?

Конечно нет!



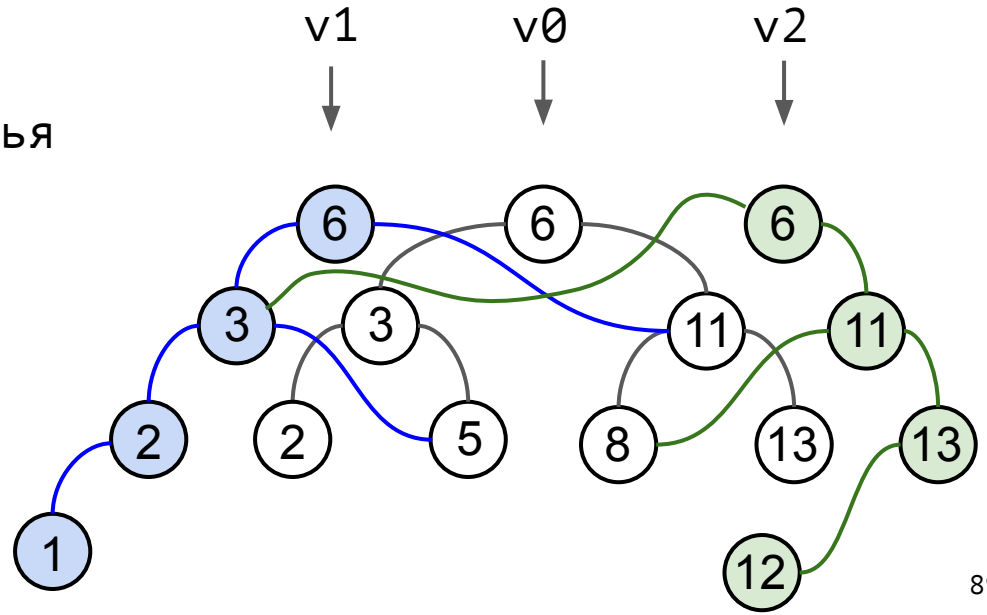
Персистентные деревья

Такая техника называется "копирование путей".

А так только с BST можно?

Конечно нет!

- Сбалансированные деревья
- Декартовы деревья
- Деревья отрезков!
- …



Персистентные деревья

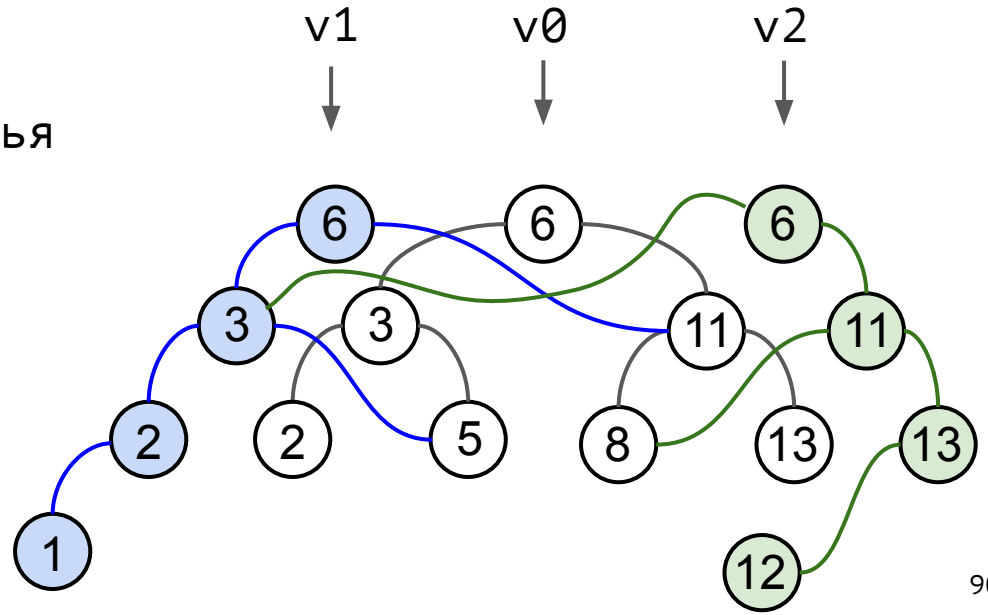
Такая техника называется "копирование путей".

А так только с BST можно?

Конечно нет!

- Сбалансированные деревья
- Декартовы деревья
- Деревья отрезков!
- …

Да, операции внутри становятся сложнее, но правило такое же: не меняем, а копируем путь.



Персистентные деревья

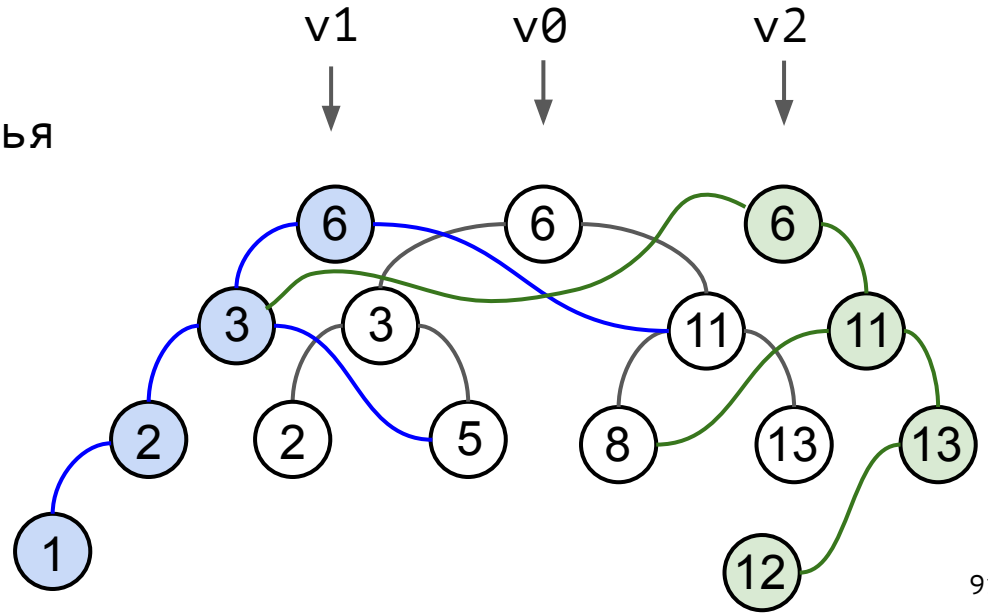
Такая техника называется "копирование путей".

А так только с BST можно?

Конечно нет!

- Сбалансированные деревья
- Декартовы деревья
- **Деревья отрезков** !
- …

Да, операции внутри становятся сложнее, но правило такое же: не меняем, а копируем путь.



Поиск количества чисел на отрезке >= k

Задача: пусть есть массив фиксированного размера n. Пусть для простоты n - это степень двойки.

$$a_1, a_2, a_3, \ldots, a_n$$

Необходимо поддержать операцию следующего вида:

- gte(l, r, k) - получить количество элементов на отрезке, которые больше либо равны k

46 11 [40 8 2 42] 65 10 gte(2, 5, 10) -> 2

Хочется использовать дерево отрезков, но как?

Поиск количества чисел на отрезке >= k

Задача: пусть есть массив фиксированного размера n. Пусть для простоты n - это степень двойки.

$$a_1, a_2, a_3, \ldots, a_n$$

Необходимо поддержать операцию следующего вида:

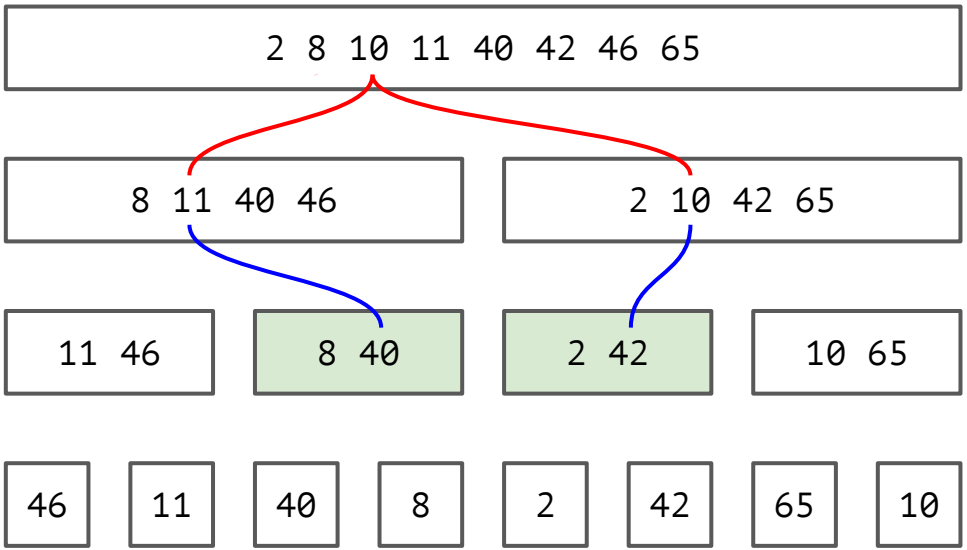
- gte(l, r, k) - получить количество элементов на отрезке, которые больше либо равны k

46 11 [40 8 2 42] 65 10 gte(2, 5, 10) -> 2

В прошлый раз использовали merge sort tree + оптимизации

Fractal cascading

gte(2, 5, 10) = 2



Пусть теперь каждый элемент в каждом массиве хранит индексы на первого, кто >= чем он, в детях

Тогда больше не нужно каждый раз запускать бинарный поиск! Только один раз в начале.

Первый бинарный поиск ищет позицию в отсортированном массиве. Он хранит нужные нам элемент в сыновьях. Дальше при спуске используем уже именно их (и так далее, каждый раз ищем в детях подходящий элемент за O(1)).

Поиск количества чисел на отрезке >= k

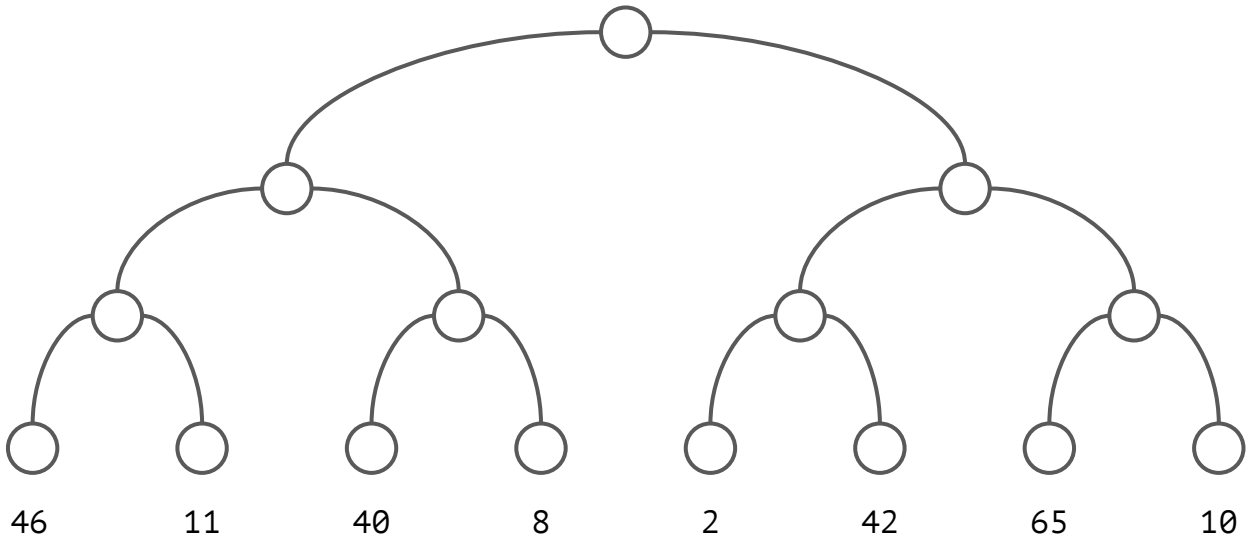
Задача: пусть есть массив фиксированного размера n.
a_1, a_2, a_3, …, a_n

Необходимо поддержать операцию следующего вида:

- gte(l, r, k) - получить количество элементов на отрезке, которые больше либо равны k

gte(l, r, k) - количество элементов на [l,r], которые >= k

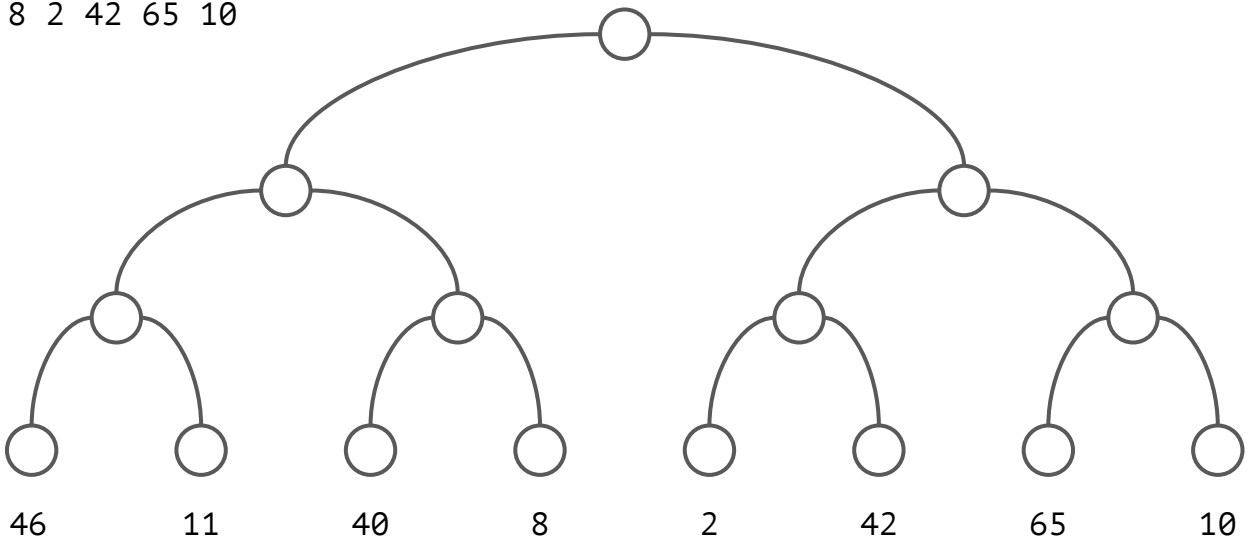
46 11 40 8 2 42 65 10



gte(l, r, k) - количество элементов на [l,r], которые >= k

1) Введем для каждого элемента свойство: жив он или нет

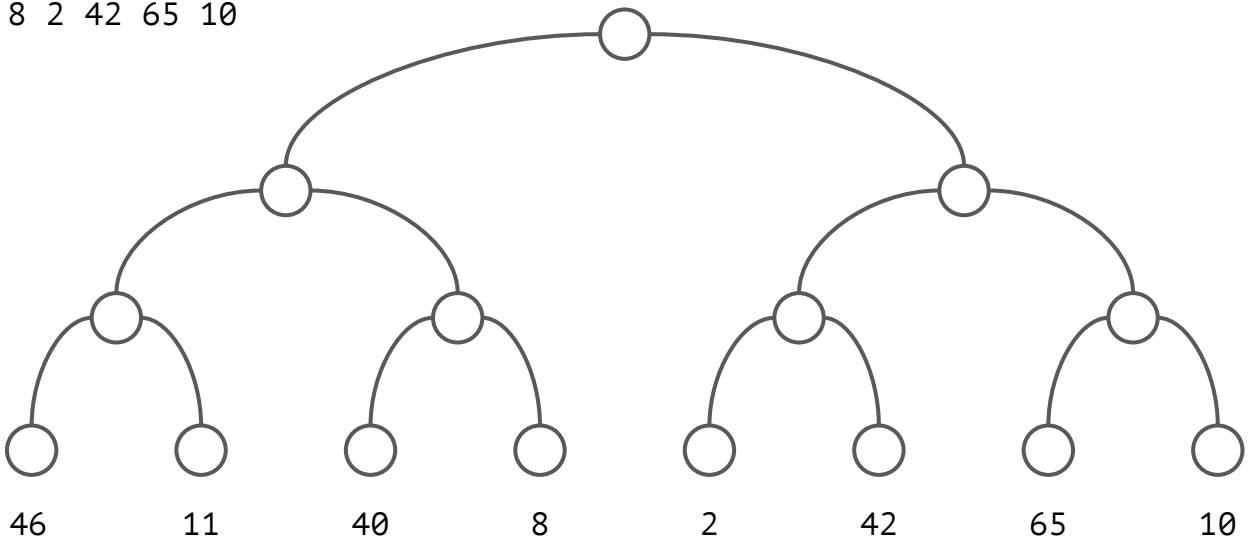
Массив: 46 11 40 8 2 42 65 10



gte(l, r, k) - количество элементов на [l,r], которые >= k

1) Введем для каждого элемента свойство: жив он или нет
2) В дереве будем считать количество живых на отрезке
3) Изначально все элементы мертвы

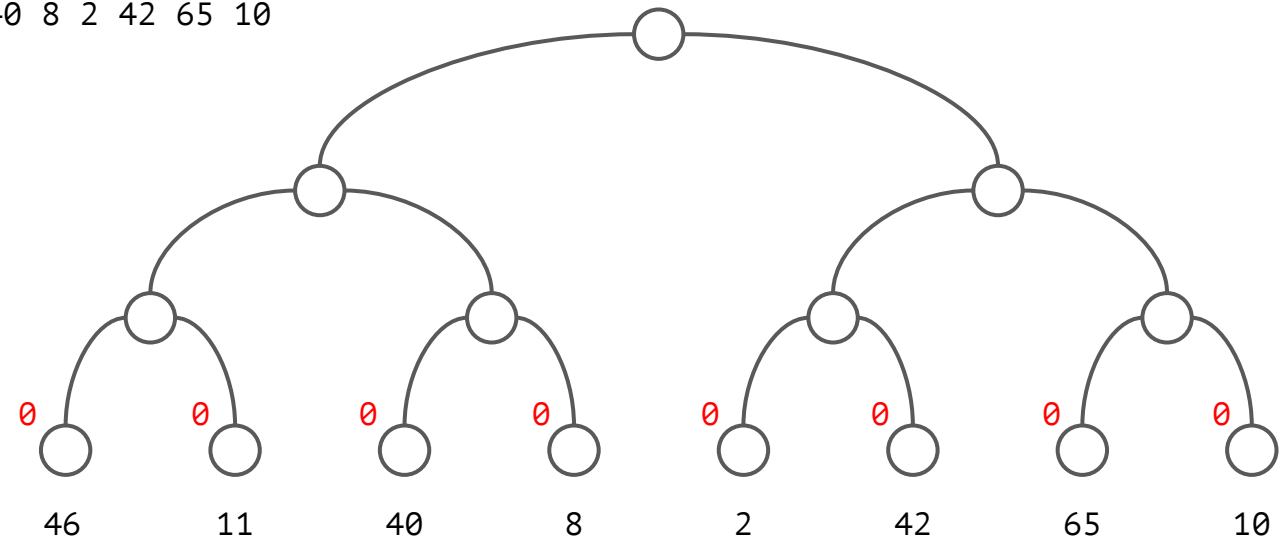
Массив: 46 11 40 8 2 42 65 10



gte(l, r, k) - количество элементов на [l,r], которые >= k

1) Введем для каждого элемента свойство: жив он или нет
2) В дереве будем считать количество живых на отрезке
3) Изначально все элементы мертвы

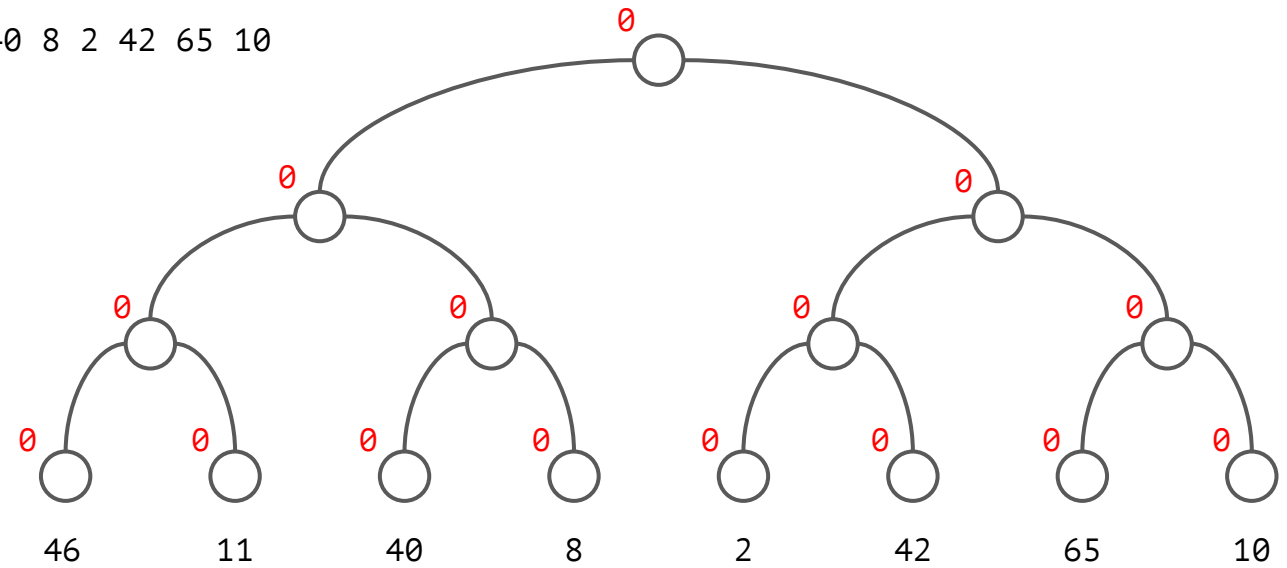
Массив: 46 11 40 8 2 42 65 10



gte(l, r, k) - количество элементов на [l,r], которые >= k

1) Введем для каждого элемента свойство: жив он или нет
2) В дереве будем считать количество живых на отрезке
3) Изначально все элементы мертвы

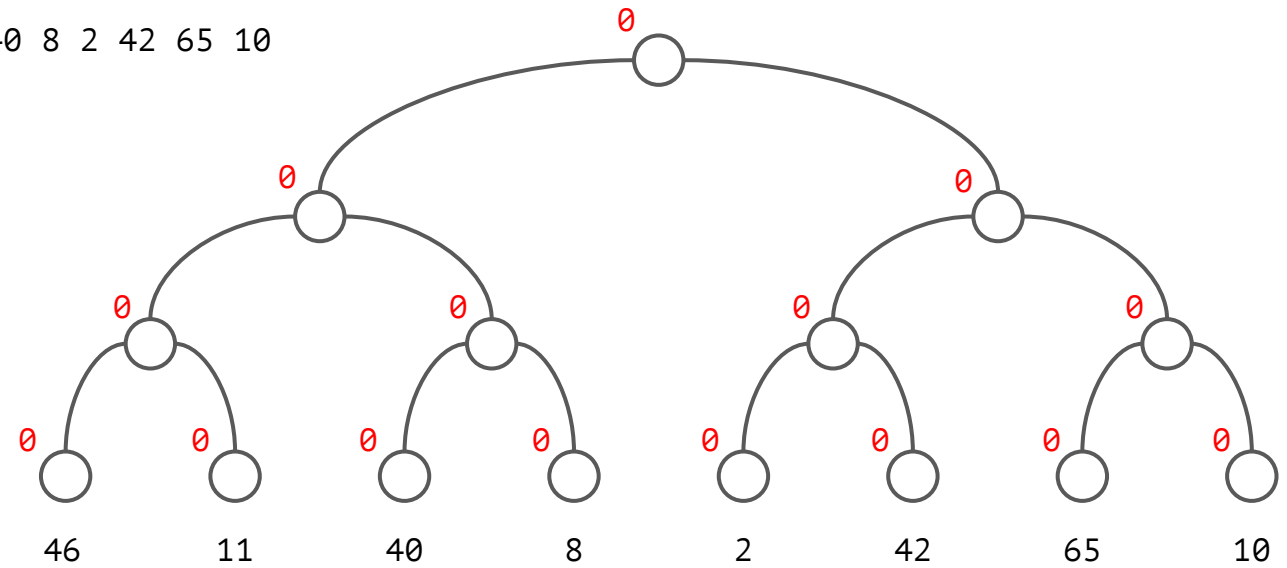
Массив: 46 11 40 8 2 42 65 10



gte(l, r, k) - количество элементов на [l,r], которые >= k

1) Введем для каждого элемента свойство: жив он или нет
2) В дереве будем считать количество живых на отрезке
3) Изначально все элементы мертвы
4) Начинаем воскрешать элемент с большего

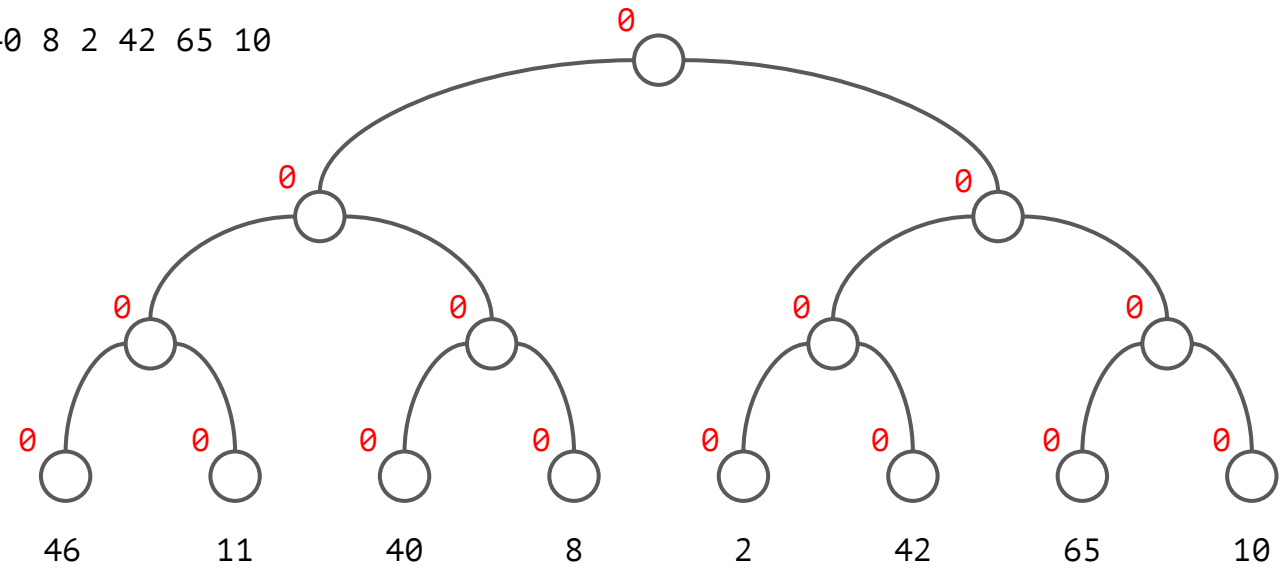
Массив: 46 11 40 8 2 42 65 10



gte(l, r, k) - количество элементов на [l,r], которые >= k

1) Введем для каждого элемента свойство: жив он или нет
2) В дереве будем считать количество живых на отрезке
3) Изначально все элементы мертвы
4) Начинаем воскрешать элемент с большего

Массив: 46 11 40 8 2 42 65 10

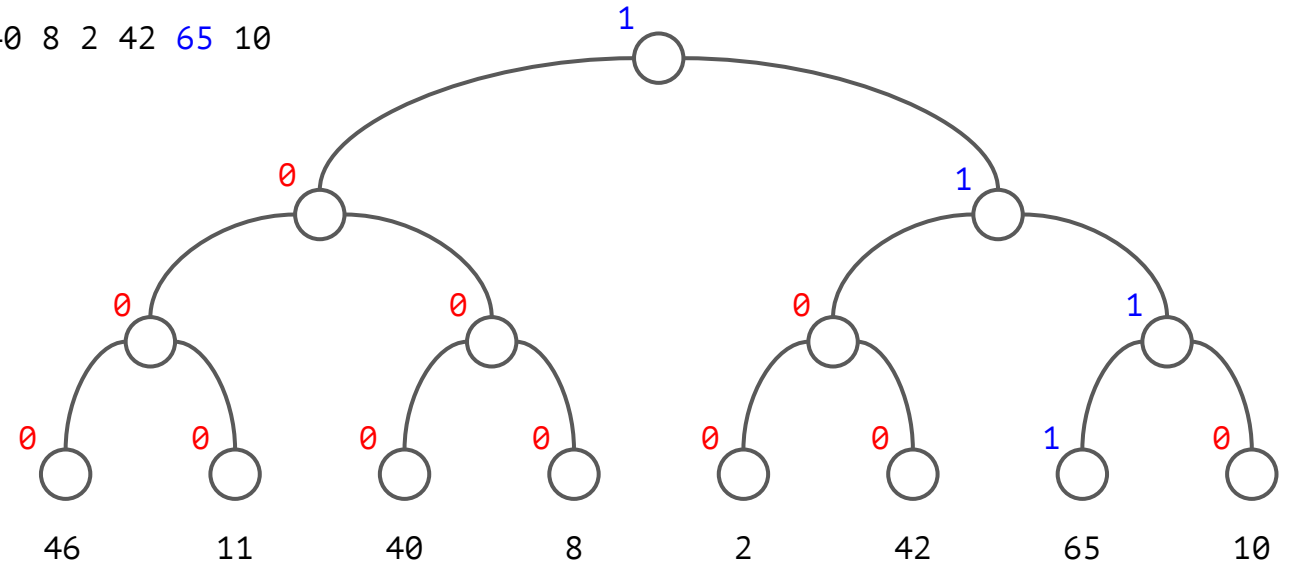


0 версия

gte(l, r, k) - количество элементов на [l,r], которые >= k

1) Введем для каждого элемента свойство: жив он или нет
2) В дереве будем считать количество живых на отрезке
3) Изначально все элементы мертвы
4) Начинаем воскрешать элемент с большего

Массив: 46 11 40 8 2 42 65 10

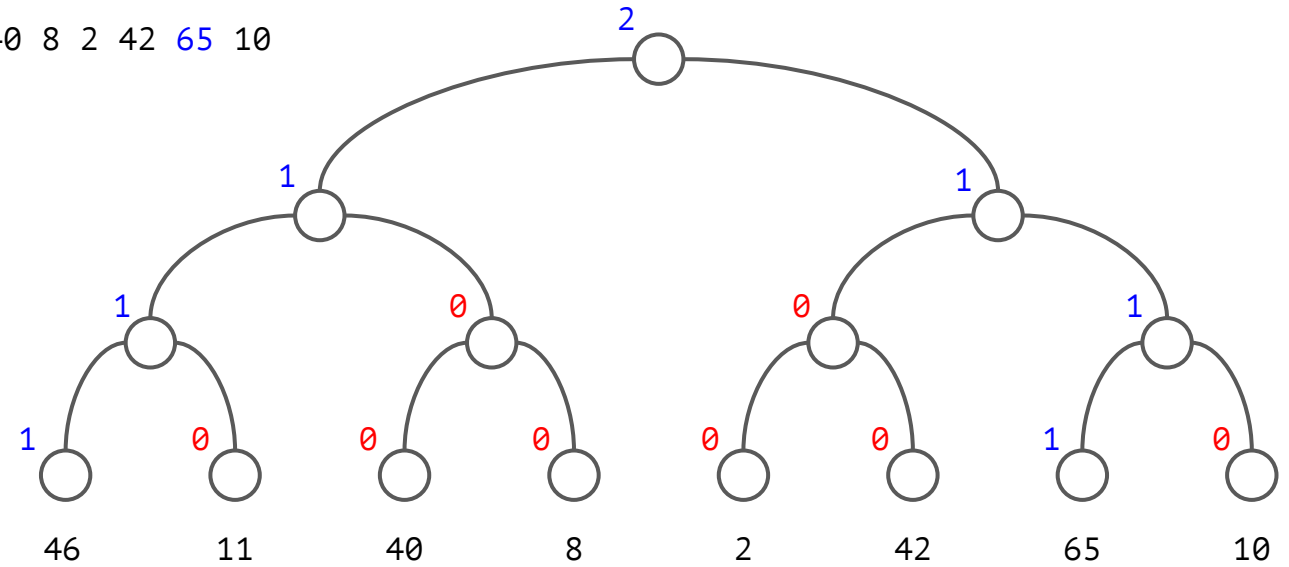


1 версия

gte(l, r, k) - количество элементов на [l,r], которые >= k

1) Введем для каждого элемента свойство: жив он или нет
2) В дереве будем считать количество живых на отрезке
3) Изначально все элементы мертвы
4) Начинаем воскрешать элемент с большего

Массив: 46 11 40 8 2 42 65 10

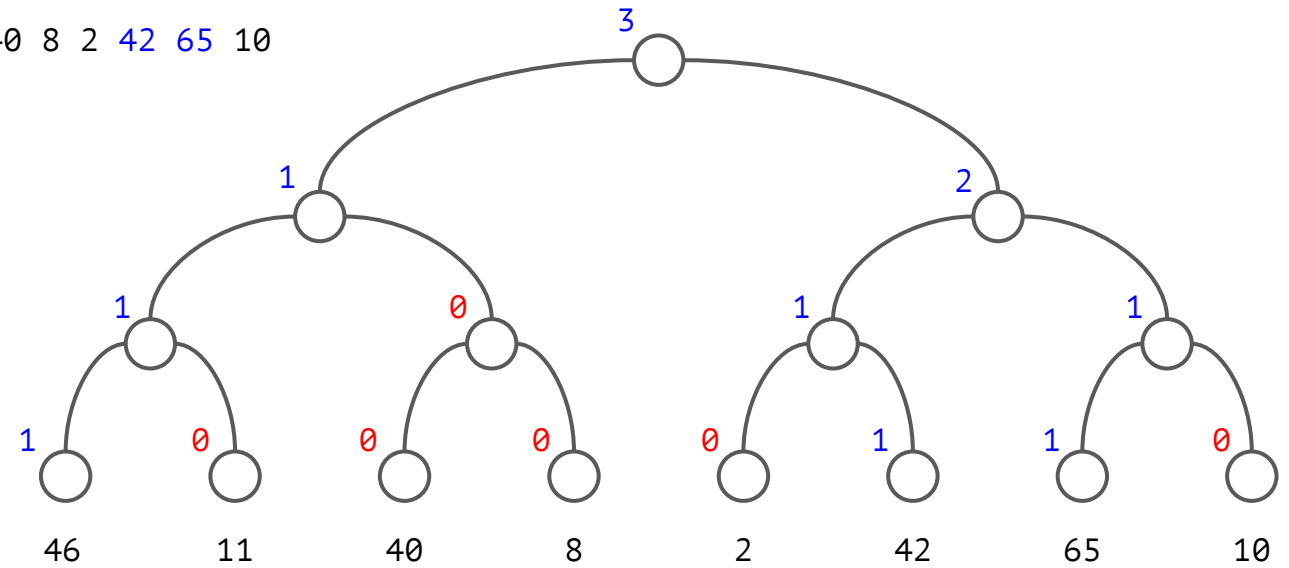


2 версия

gte(l, r, k) - количество элементов на [l,r], которые >= k

1) Введем для каждого элемента свойство: жив он или нет
2) В дереве будем считать количество живых на отрезке
3) Изначально все элементы мертвы
4) Начинаем воскрешать элемент с большего

Массив: 46 11 40 8 2 42 65 10

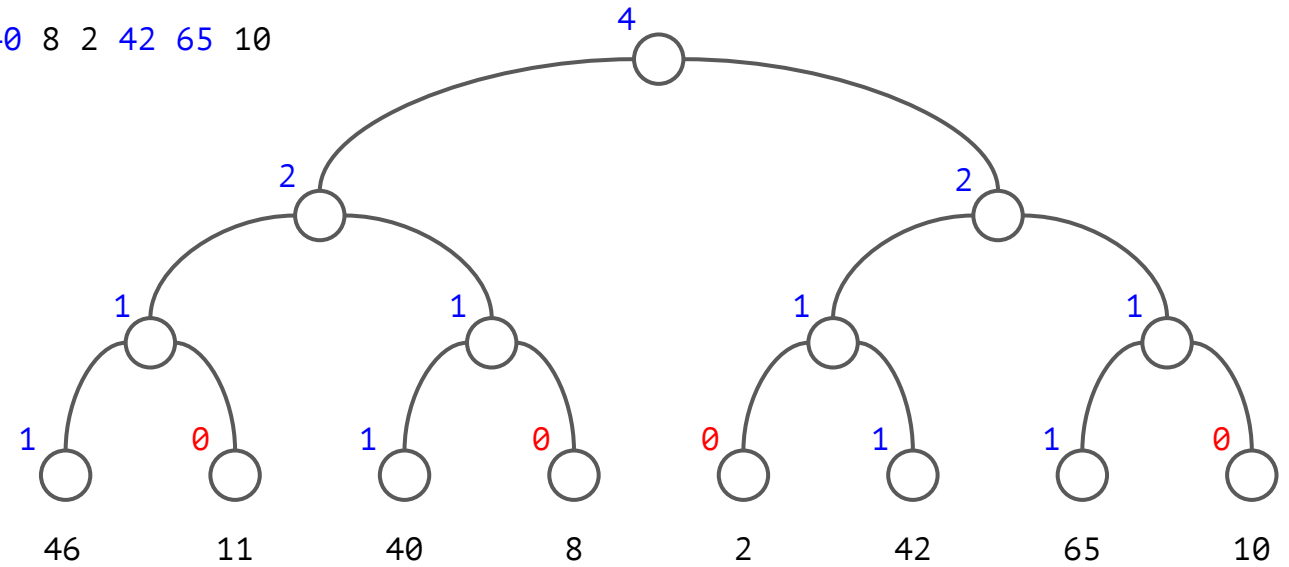


3 версия

gte(l, r, k) - количество элементов на [l,r], которые >= k

1) Введем для каждого элемента свойство: жив он или нет
2) В дереве будем считать количество живых на отрезке
3) Изначально все элементы мертвы
4) Начинаем воскрешать элемент с большего

Массив: 46 11 40 8 2 42 65 10

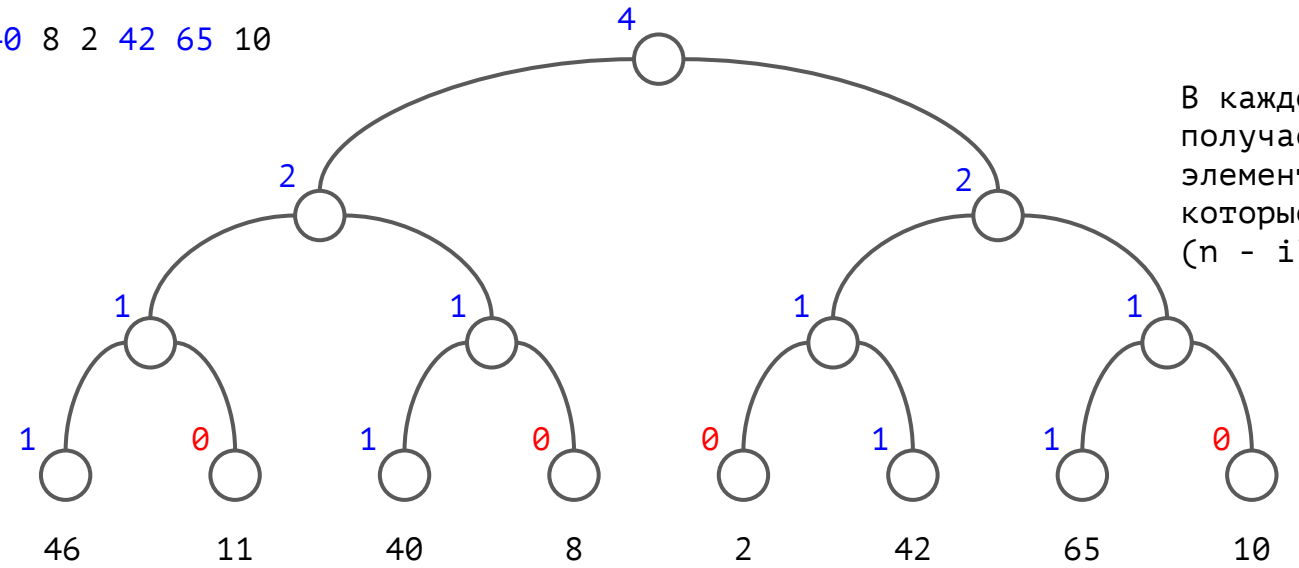


4 версия

gte(l, r, k) - количество элементов на [l,r], которые >= k

1) Введем для каждого элемента свойство: жив он или нет
2) В дереве будем считать количество живых на отрезке
3) Изначально все элементы мертвы
4) Начинаем воскрешать элемент с большего

Массив: 46 11 40 8 2 42 65 10



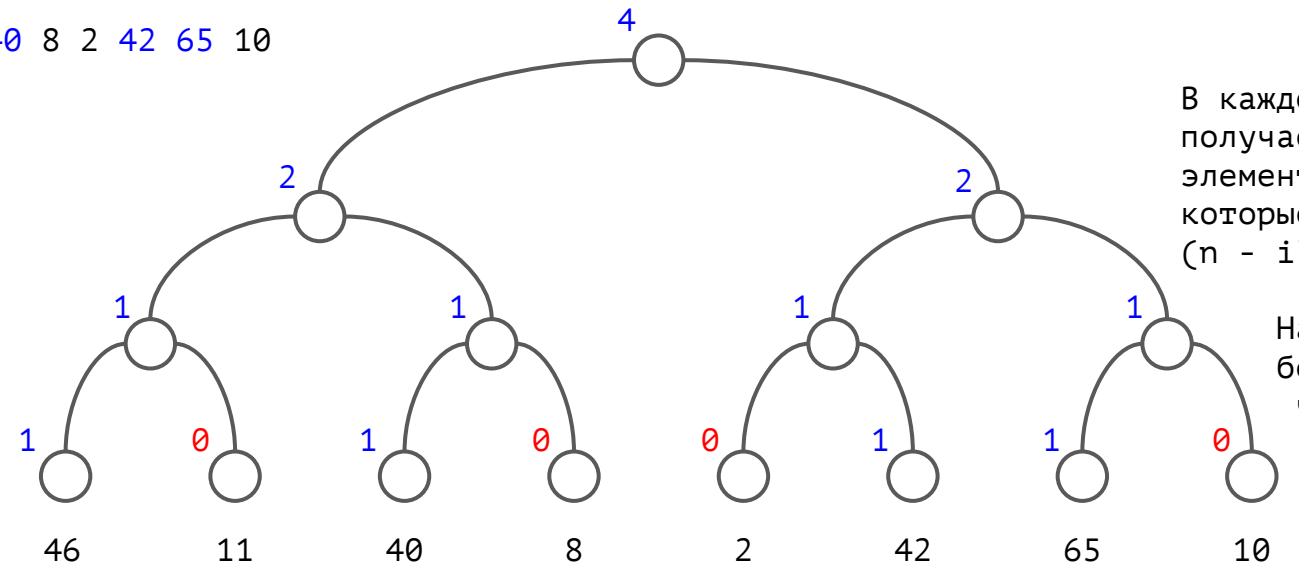
4 версия

В каждой версии i получаем количество элементов на отрезках, которые больше > (n - i)-ая статистика

gte(l, r, k) - количество элементов на [l,r], которые >= k

1) Введем для каждого элемента свойство: жив он или нет
2) В дереве будем считать количество живых на отрезке
3) Изначально все элементы мертвы
4) Начинаем воскрешать элемент с большего

Массив: 46 11 40 8 2 42 65 10



4 версия

В каждой версии i получаем количество элементов на отрезках, которые больше > (n - i)-ая статистика

Например, здесь больше, чем 8-4: чем 4-ая статистика (11)

gte(l, r, k) - количество элементов на [l,r], которые >= k

1) Введем для каждого элемента свойство: жив он или нет
2) В дереве будем считать количество живых на отрезке
3) Изначально все элементы мертвы
4) Начинаем воскрешать элемент с большего

5) При этом дерево отрезков делаем персистентным, каждая версия хранится в памяти.

gte(l, r, k) - количество элементов на [l,r], которые >= k

1) Введем для каждого элемента свойство: жив он или нет
2) В дереве будем считать количество живых на отрезке
3) Изначально все элементы мертвы
4) Начинаем воскрешать элемент с большего

5) При этом дерево отрезков делаем персистентным, каждая версия хранится в памяти.

6) Тогда, чтобы решить изначальную задачу:

   a) сортируем массив за O(N*logN)

gte(l, r, k) - количество элементов на [l,r], которые >= k

1) Введем для каждого элемента свойство: жив он или нет
2) В дереве будем считать количество живых на отрезке
3) Изначально все элементы мертвы
4) Начинаем воскрешать элемент с большего

5) При этом дерево отрезков делаем персистентным, каждая версия хранится в памяти.

6) Тогда, чтобы решить изначальную задачу:

    а) сортируем массив за O(N*logN)
    б) персистентное дерево строим тоже за O(N*logN)

gte(l, r, k) - количество элементов на [l,r], которые >= k

1) Введем для каждого элемента свойство: жив он или нет
2) В дереве будем считать количество живых на отрезке
3) Изначально все элементы мертвы
4) Начинаем воскрешать элемент с большего

5) При этом дерево отрезков делаем персистентным, каждая версия хранится в памяти.

6) Тогда, чтобы решить изначальную задачу:

   а) сортируем массив за O(N*logN)
   б) персистентное дерево строим тоже за O(N*logN)
   в) каждый запрос = бин. поиск версии за O(logN) + поиск по дереву - тоже O(logN)

gte(l, r, k) - количество элементов на [l,r], которые >= k

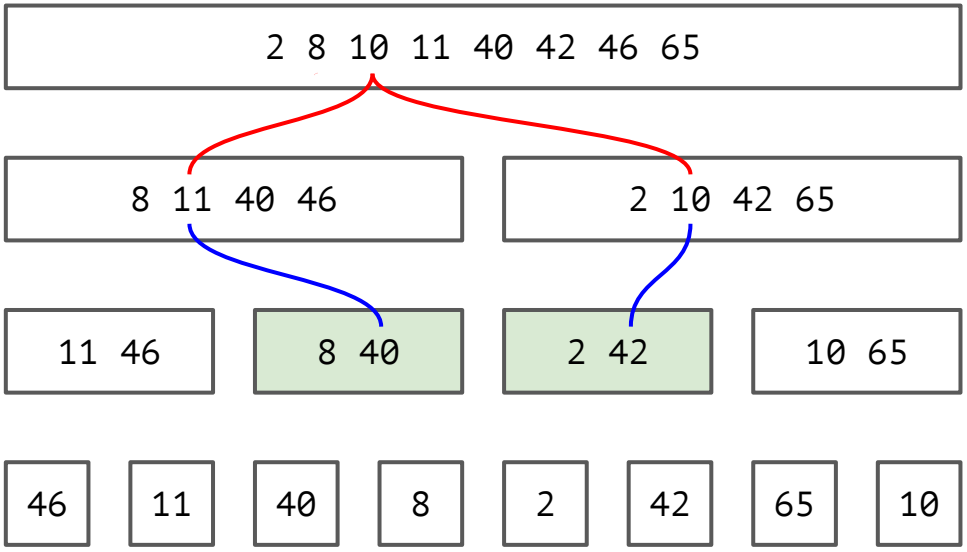
1) Введем для каждого элемента свойство: жив он или нет
2) В дереве будем считать количество живых на отрезке
3) Изначально все элементы мертвы
4) Начинаем воскрешать элемент с большего

5) При этом дерево отрезков делаем персистентным, каждая версия хранится в памяти.

6) Тогда, чтобы решить изначальную задачу:

   а) сортируем массив за O(N*logN)
   б) персистентное дерево строим тоже за O(N*logN)
   в) каждый запрос = бин. поиск версии за O(logN) + поиск по дереву - тоже O(logN)
   г) память: O(N + N*logN) = O(N*logN)
      (N версий, каждая добавляет logN новых вершин)

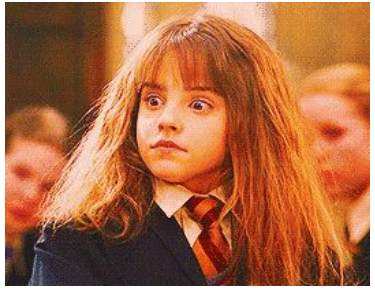
Fractal cascading



Пусть теперь каждый элемент в каждом массиве хранит индексы на первого, кто >= чем он, в детях

Тогда больше не нужно каждый раз запускать бинарный поиск! Только один раз в начале.

Сложность построения: не меняется O(N*logN)
Память: x3, но остается O(N*logN)
Сложность запроса: O(logN) !!!



Мини-задача #46 (1 балл, дополнительная)

Решите задачу поиска чисел >= k на отрезке с помощью персистентного дерева отрезков.

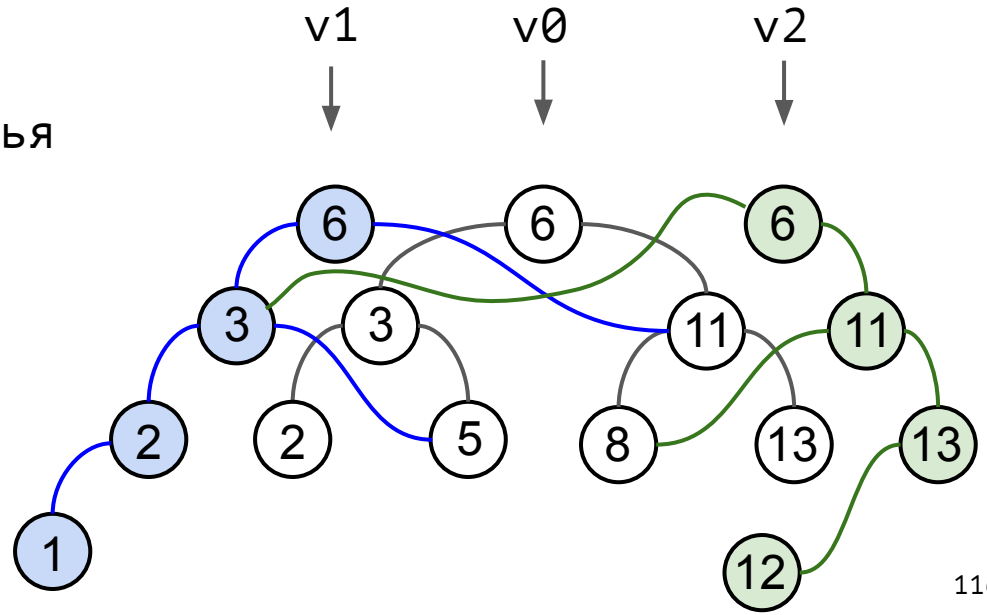
Персистентные деревья

Такая техника называется "копирование путей".

А так только с BST можно?

Конечно нет!

- Сбалансированные деревья
- Декартовы деревья
- Деревья отрезков!
- …



Персистентный массив

Пусть есть массив: a0, a1, …, an

Есть операция update(i, value), которая присваивает в i-ый элемент значение value. Это инкрементирует версию.
Изначальная версия - 0.

get(t, i): взять i-ый в версии t.

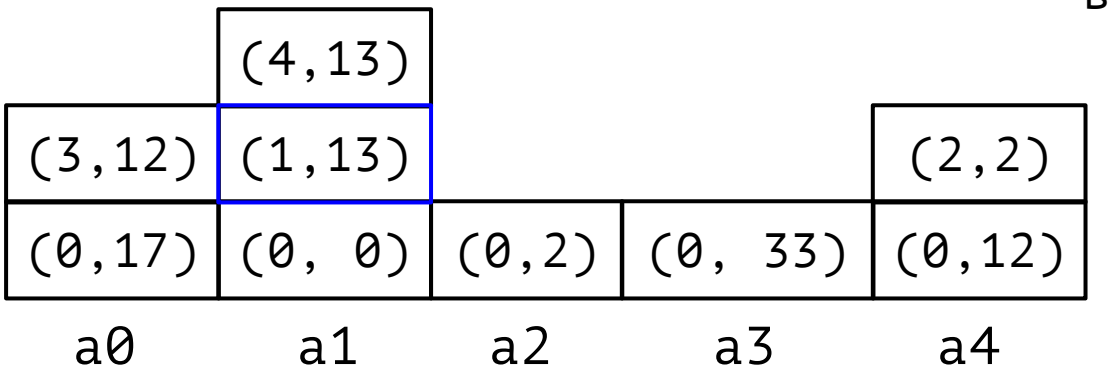
Персистентный массив

Время: get(...) -> logK
Память: N + K
K - количество запросов
Частичная персистентность

Пусть есть массив: a0, a1, …, an

Есть операция update(i, value), которая присваивает в i-ый элемент значение value. Это инкрементирует версию.
Изначальная версия - 0.

get(t, i): взять i-ый в версии t. Как решать? Для каждого элемента хранить все версии!



При каждой операции находим бинарным поиском ближайшую версию <= нужной.

Персистентный массив

Пусть есть массив: a0, a1, …, an

Есть операция update(t, i, value), которая присваивает в i-ый элемент значение value, базируясь при этом на версии t и создавая новую версию.

get(t, i): взять i-ый в версии t.

Персистентный массив

Пусть есть массив: a0, a1, …, an

Есть операция update(t, i, value), которая присваивает в i-ый элемент значение value, базируясь при этом на версии t и создавая новую версию.

get(t, i): взять i-ый в версии t.

Другими словами: как получить полную персистентность?

Персистентный массив

46 11 40 8 2 42 65 10

Персистентный массив

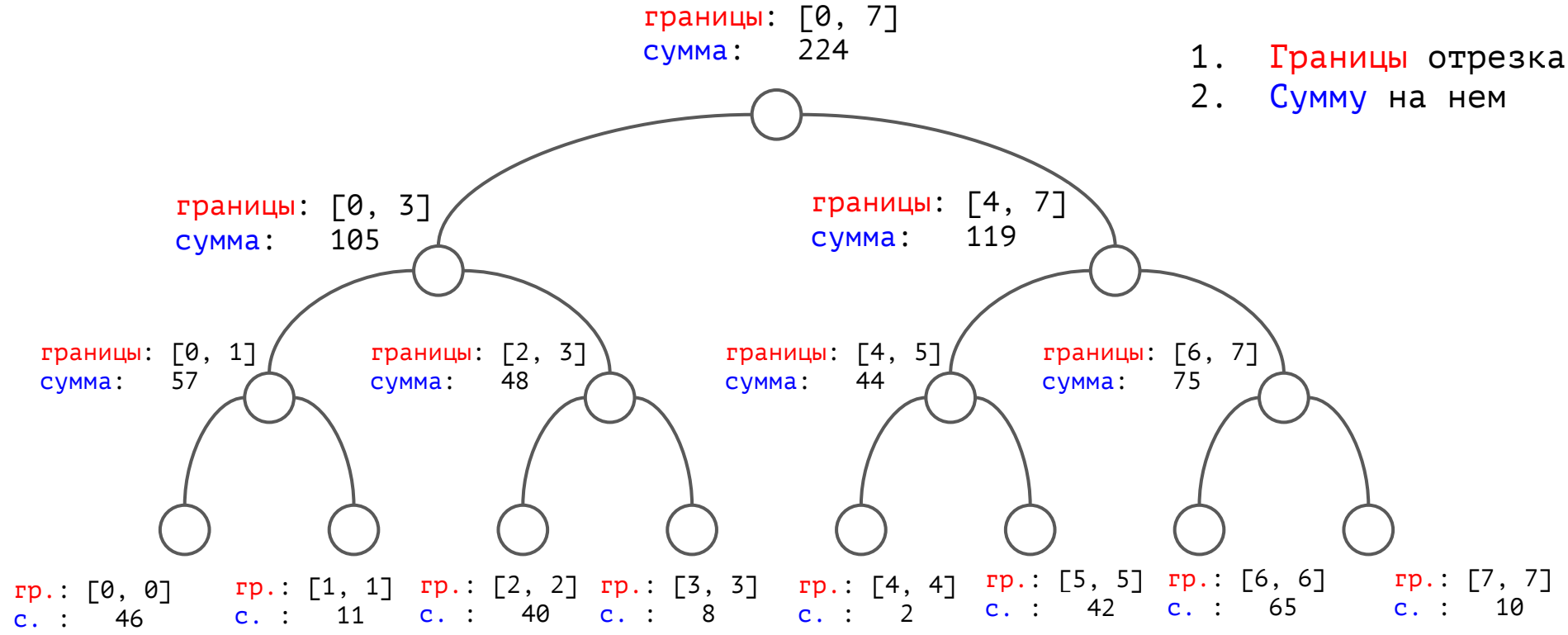
46 11 40 8 2 42 65 10

46 11 40 8 2 42 65 10

46 11 40 8 2 42 65 10

На самом деле в каждой вершине будем хранить:

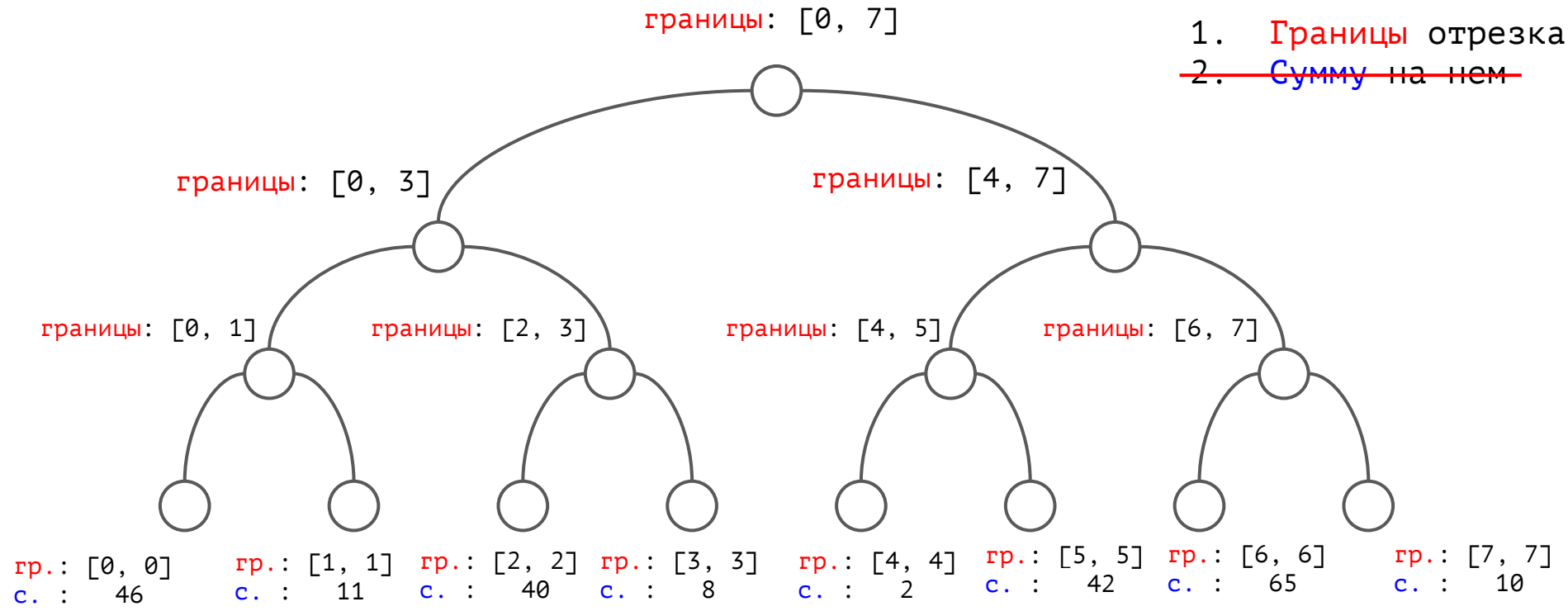
1. Границы отрезка
2. Сумму на нем



46 11 40 8 2 42 65 10

На самом деле в каждой вершине будем хранить:

1. Границы отрезка
2. ~~Сумму на нем~~

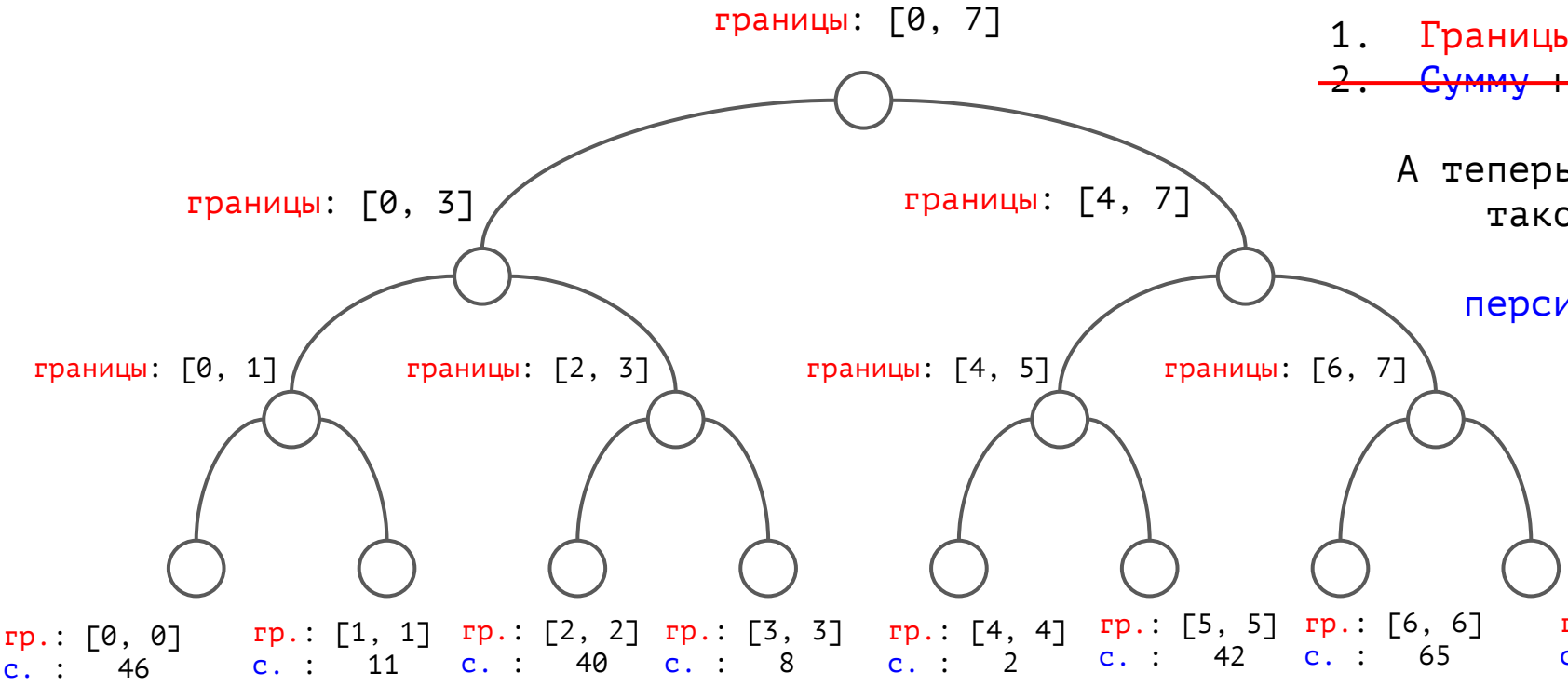


46 11 40 8 2 42 65 10

На самом деле в каждой вершине будем хранить:

1. Границы отрезка
2. ~~Сумму на нем~~

А теперь сделаем такое дерево отрезков персистентным



Персистентный массив

Пусть есть массив: a0, a1, …, an

Есть операция update(t, i, value), которая присваивает в i-ый элемент значение value, базируясь при этом на версии t и создавая новую версию.

get(t, i): взять i-ый в версии t.

Другими словами: как получить полную персистентность?

Делаем персистентное дерево отрезков на массиве.

Персистентный массив

Пусть есть массив: a0, a1, …, an

Есть операция update(t, i, value), которая присваивает в i-ый элемент значение value, базируясь при этом на версии t и создавая новую версию.

get(t, i): взять i-ый в версии t.

Другими словами: как получить полную персистентность?

Делаем персистентное дерево отрезков на массиве.

Время: ?
Память: ?

Персистентный массив

Пусть есть массив: a0, a1, …, an

Есть операция update(t, i, value), которая присваивает в i-ый элемент значение value, базируясь при этом на версии t и создавая новую версию.

get(t, i): взять i-ый в версии t.

Другими словами: как получить полную персистентность?

Делаем персистентное дерево отрезков на массиве.

Время: logN (как и всегда в дереве отрезков)
Память: ?

Персистентный массив

Пусть есть массив: a0, a1, …, an

Есть операция update(t, i, value), которая присваивает в i-ый элемент значение value, базируясь при этом на версии t и создавая новую версию.

get(t, i): взять i-ый в версии t.

Другими словами: как получить полную персистентность?

Делаем персистентное дерево отрезков на массиве.

Время: logN (как и всегда в дереве отрезков)
Память: O(N + K*logN)

Персистентный массив

Пусть есть массив: a0, a1, …, an

Есть операция update(t, i, value), которая присваивает в i-ый элемент значение value, базируясь при этом на версии t и создавая новую версию.

get(t, i): взять i-ый в версии t.

Персистентные массивы дают возможность делать реализованные через них структуры данных персистентными.

Персистентный массив

Пусть есть массив: a0, a1, …, an

Есть операция update(t, i, value), которая присваивает в i-ый элемент значение value, базируясь при этом на версии t и создавая новую версию.

get(t, i): взять i-ый в версии t.

Персистентные массивы дают возможность делать реализованные через них структуры данных персистентными.

Пример: union-find! Реализуется через несколько массивов.

Персистентный массив

Пусть есть массив: a0, a1, …, an

Есть операция update(t, i, value), которая присваивает в i-ый элемент значение value, базируясь при этом на версии t и создавая новую версию.

get(t, i): взять i-ый в версии t.

Персистентные массивы дают возможность делать реализованные через них структуры данных персистентными.

Пример: union-find! Реализуется через несколько массивов. Персистентным его так сделать можно, но вот почти-линейность пропадет, т.к. не получится сокращать пути (упражнение: подумайте, почему)

Персистентный стек

Абстрактный тип данных: стек

Множество значений: элементы заданного типа T

Операции:

1. `push(value: T)` — добавление нового элемента
2. `pop() -> T` — удаление последнего добавленного элемента и возвращение его значения

Персистентный стек

Операции:

1. `push(t: int, value: T)`
2. `pop(t: int)`

И пушаем и достаем
из версии `t`

Персистентный стек

Операции:

1. push(t: int, value: T)
2. pop(t: int)

Как будем делать?

И пушаем и достаем из версии t

Персистентный стек

Операции:

1. push(t: int, value: T)
2. pop(t: int)

И пушаем и достаем
из версии t

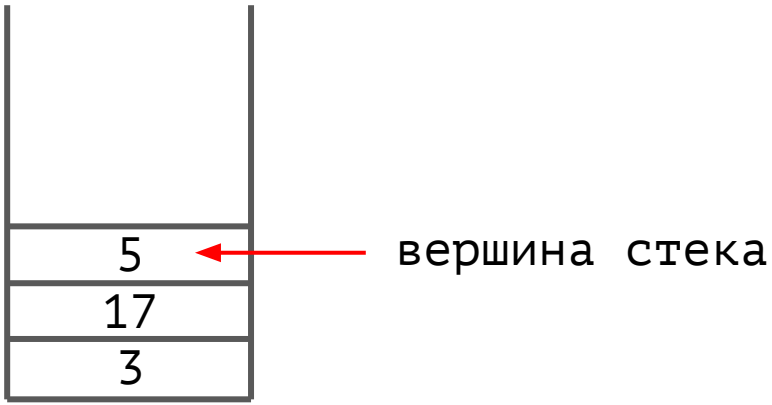
Как будем делать? Можно вспомнить, что стек - это массив, но лучше подумать про него,... как про дерево!

Персистентный стек

Операции:

1. `push(t: int, value: T)`
2. `pop(t: int)`

И пушаем и достаем из версии `t`

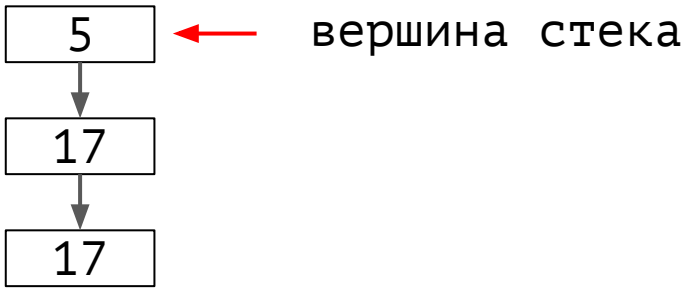


Персистентный стек

Операции:

1. `push(t: int, value: T)`
2. `pop(t: int)`

И пушаем и достаем из версии `t`

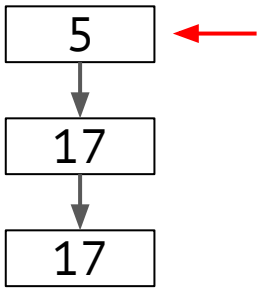


Персистентный стек

Операции:

1. push(t: int, value: T)
2. pop(t: int)

И пушаем и достаем
из версии t



вершина стека

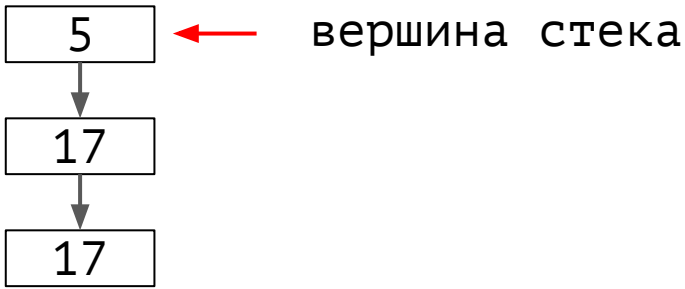
Что, если применить здесь "копирование путей"?

Персистентный стек

Операции:

1. push(t: int, value: T)
2. pop(t: int)

И пушаем и достаем из версии t



Что, если применить здесь "копирование путей"?

Мы всегда работаем только с root-ом, на него никто не ссылается!

Операции:

1. push(t: int, value: T)
2. pop(t: int)

И пушаем и достаем из версии t

Будем хранить в каждой версии значение и предыдущую версию.

Версия	0
value	None
Prev	None

Операции:

1. push(t: int, value: T)
2. pop(t: int)

И пушаем и достаем из версии t

Будем хранить в каждой версии значение и предыдущую версию.

Версия	0
value	None
Prev	None

push(0, 13)

Операции:

1. push(t: int, value: T)
2. pop(t: int)

И пушаем и достаем из версии t

Будем хранить в каждой версии значение и предыдущую версию.

Версия	0	1
value	None	13
Prev	None	0

push(0, 13)

Операции:

1. push(t: int, value: T)
2. pop(t: int)

И пушаем и достаем из версии t

Будем хранить в каждой версии значение и предыдущую версию.

Версия	0	1
value	None	13
Prev	None	0

```
push(0, 13)
push(1, 42)
```

Операции:

1. push(t: int, value: T)
2. pop(t: int)

И пушаем и достаем из версии t

Будем хранить в каждой версии значение и предыдущую версию.

Версия	0	1	2
value	None	13	42
Prev	None	0	1

```
push(0, 13)
push(1, 42)
```

Операции:

1. push(t: int, value: T)
2. pop(t: int)

И пушаем и достаем из версии t

Будем хранить в каждой версии значение и предыдущую версию.

Версия	0	1	2	3
value	None	13	42	5
Prev	None	0	1	0

```
push(0, 13)
push(1, 42)
push(0, 5)
```

Операции:

1. push(t: int, value: T)
2. pop(t: int)

И пушаем и достаем из версии t

Будем хранить в каждой версии значение и предыдущую версию.

Версия	0	1	2	3
value	None	13	42	5
Prev	None	0	1	0

```
push(0, 13)
push(1, 42)
push(0, 5)
 pop(2)?
```

Операции:

1. push(t: int, value: T)
2. pop(t: int)

И пушаем и достаем из версии t

Будем хранить в каждой версии значение и предыдущую версию.

Версия	0	1	2	3	4
value	None	13	42	5	13
Prev	None	0	1	0	0

```
push(0, 13)
push(1, 42)
push(0, 5)
 pop(2)
```

Операции:

1. push(t: int, value: T)
2. pop(t: int)

И пушаем и достаем из версии t

Будем хранить в каждой версии значение и предыдущую версию.

Версия	0	1	2	3	4	5
value	None	13	42	5	13	None
Prev	None	0	1	0	0	None

```
push(0, 13)
push(1, 42)
push(0, 5)
 pop(2)
 pop(1)
```

Операции:

1. push(t: int, value: T)
2. pop(t: int)

И пушаем и достаем из версии t

Будем хранить в каждой версии значение и предыдущую версию.

Версия	0	1	2	3	4	5	6
value	None	13	42	5	13	None	99
Prev	None	0	1	0	0	None	4

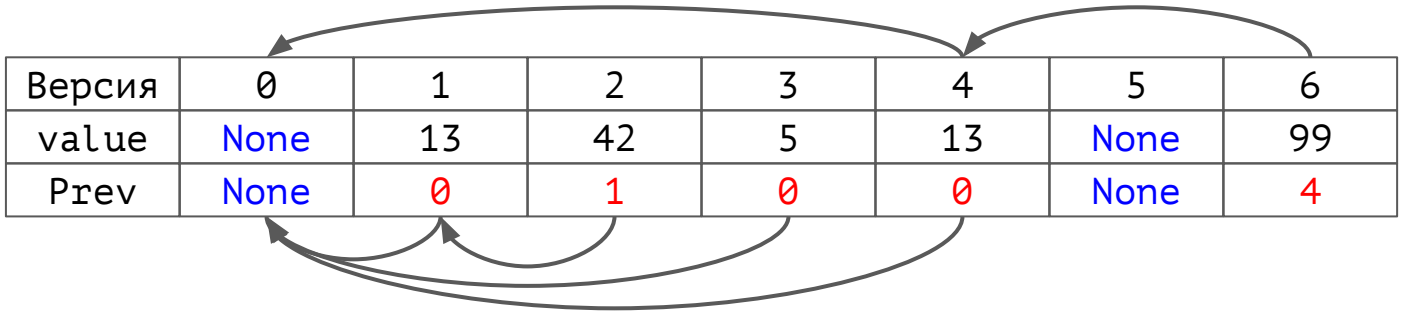
```
push(0, 13)
push(1, 42)
push(0, 5)
 pop(2)
 pop(1)
push(4, 99)
```

Операции:

1. push(t: int, value: T)
2. pop(t: int)

И пушаем и достаем из версии t

Будем хранить в каждой версии значение и предыдущую версию.



Версия	0	1	2	3	4	5	6
value	None	13	42	5	13	None	99
Prev	None	0	1	0	0	None	4

```
push(0, 13)
push(1, 42)
push(0, 5)
 pop(2)
 pop(1)
push(4, 99)
```

Операции:

1. push(t: int, value: T)
2. pop(t: int)

И пушаем и достаем из версии t

Будем хранить в каждой версии значение и предыдущую версию.

Версия	0	1	2	3	4	5	6
value	None	13	42	5	13	None	99
Prev	None	0	1	0	0	None	4

```
push(0, 13)
push(1, 42)
push(0, 5)
 pop(2)
 pop(1)
push(4, 99)
```

Время: O(1)
Память: O(K)
K - количество запросов
Полная персистентность

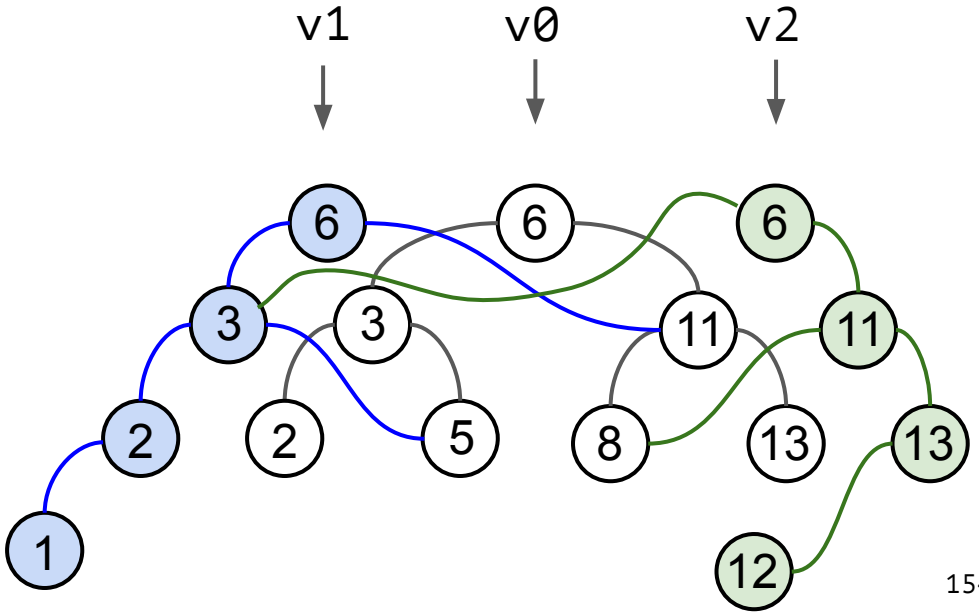
Персистентные деревья

Пусть есть бинарное дерево поиска. Реализовать операции:

```
insert(value)
find(t, value)
```

Такая техника называется "копирование путей".

будем хранить в массиве v указатели на root-ы деревьев разных версий



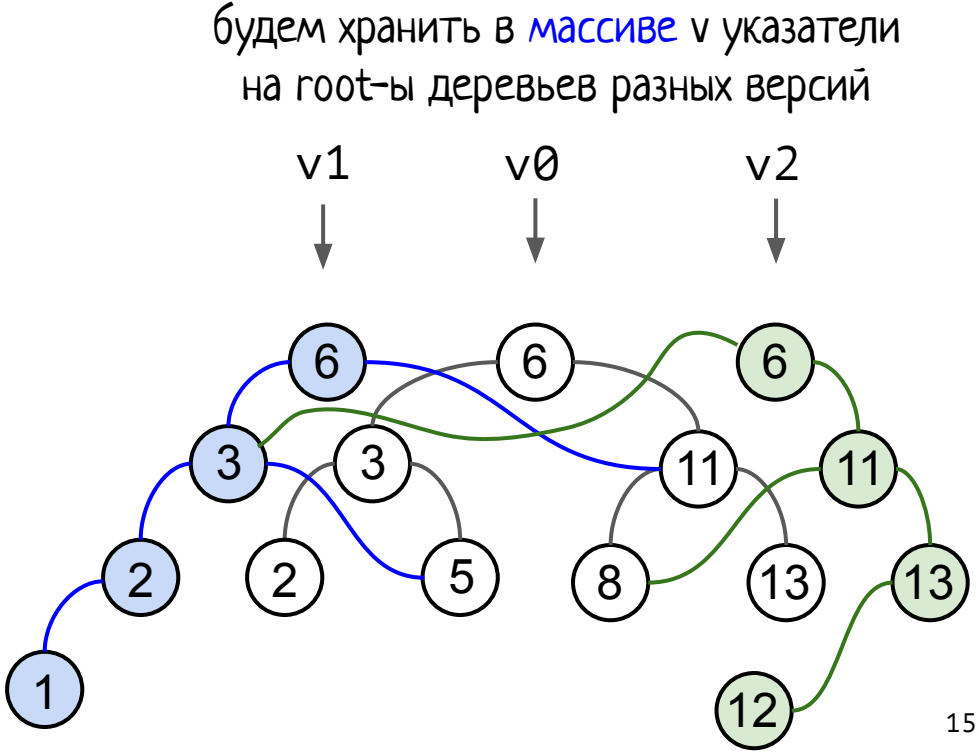
Персистентные деревья

Пусть есть бинарное дерево поиска. Реализовать операции:

```
insert(value)
find(t, value)
```

Такая техника называется "копирование путей".

Обобщим: какие конкретно вершины нужно копировать вместо изменения вершины q?



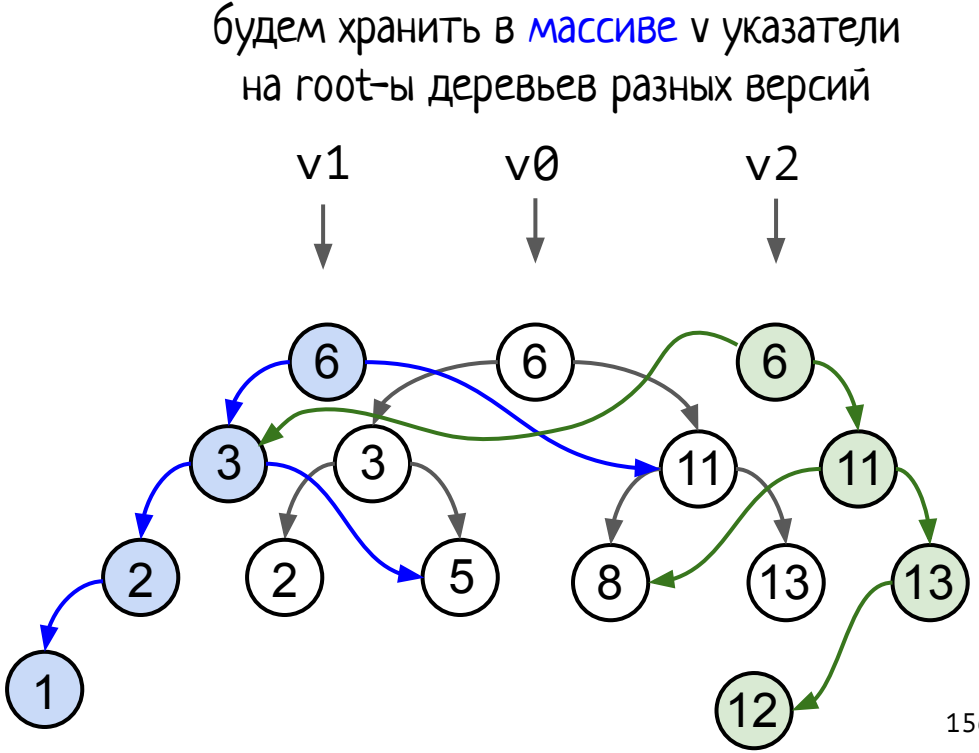
Персистентные деревья

Пусть есть бинарное дерево поиска. Реализовать операции:

```
insert(value)
find(t, value)
```

Такая техника называется "копирование путей".

Обобщим: какие конкретно вершины нужно копировать вместо изменения вершины q?



Персистентные деревья

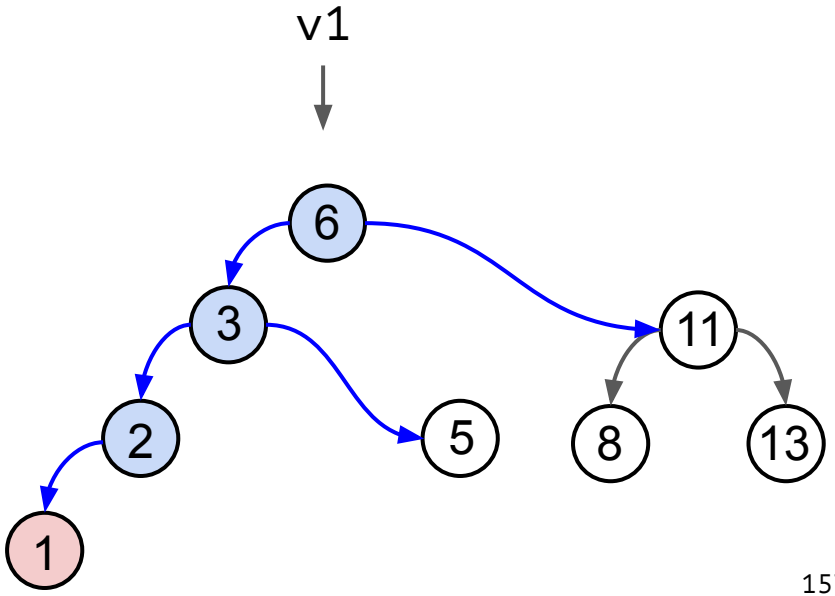
Пусть есть бинарное дерево поиска. Реализовать операции:

```
insert(value)
find(t, value)
```

Такая техника называется "копирование путей".

Обобщим: какие конкретно вершины нужно копировать вместо изменения вершины q?

Копируем все, из которых q достижима в текущем дереве!



Персистентные деревья

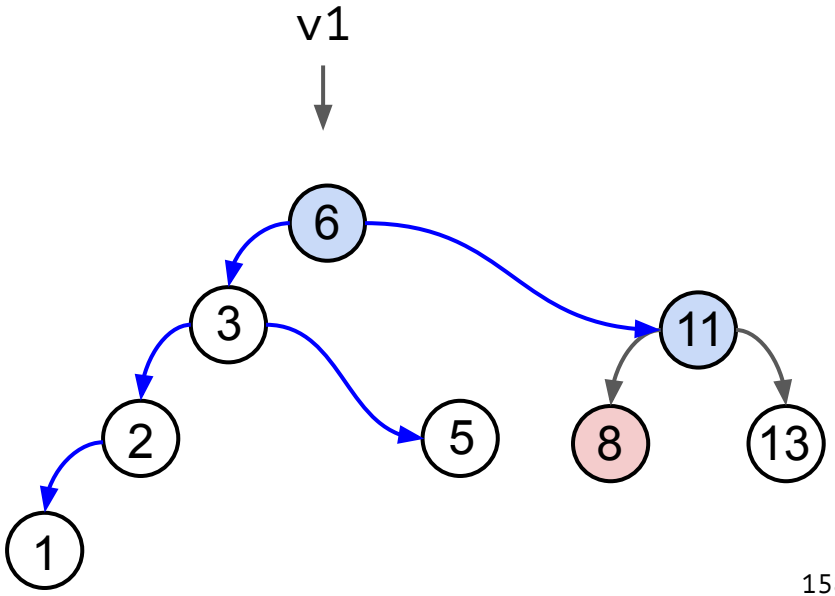
Пусть есть бинарное дерево поиска. Реализовать операции:

```
insert(value)
find(t, value)
```

Такая техника называется "копирование путей".

Обобщим: какие конкретно вершины нужно копировать вместо изменения вершины q?

Копируем все, из которых q достижима в текущем дереве!



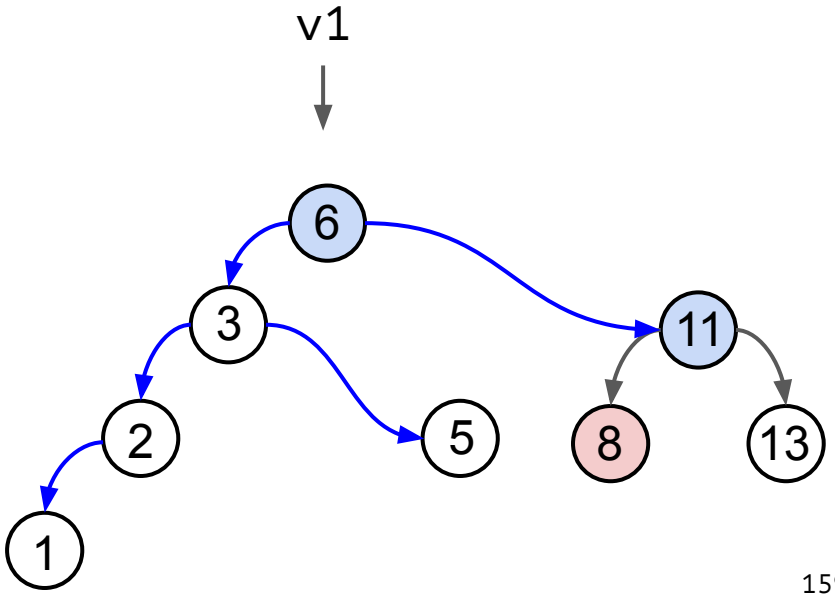
Персистентные деревья

Такая техника называется "копирование путей".

Обобщим: какие конкретно вершины нужно копировать вместо изменения вершины q?

Копируем все, из которых q достижима в текущем дереве!

Понимая это, легко построить контр-пример на эту технику.



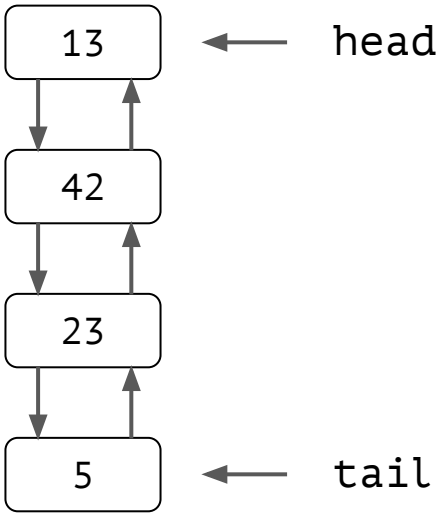
Персистентные деревья

Такая техника называется "копирование путей".

Обобщим: какие конкретно вершины нужно копировать вместо изменения вершины q?

Копируем все, из которых q достижима в текущем дереве!

Понимая это, легко построить контр-пример на эту технику.



Персистентные списки

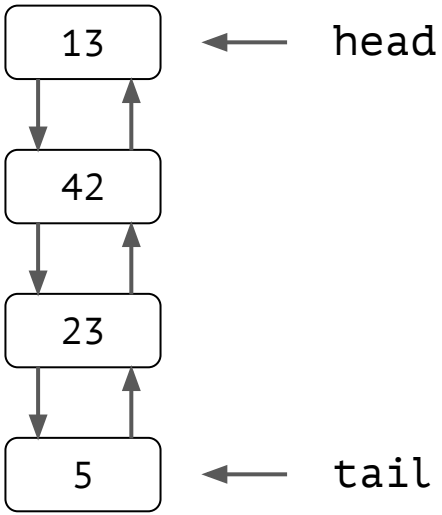
Такая техника называется "копирование путей".

Обобщим: какие конкретно вершины нужно копировать вместо изменения вершины q?

Копируем все, из которых q достижима в текущем дереве!

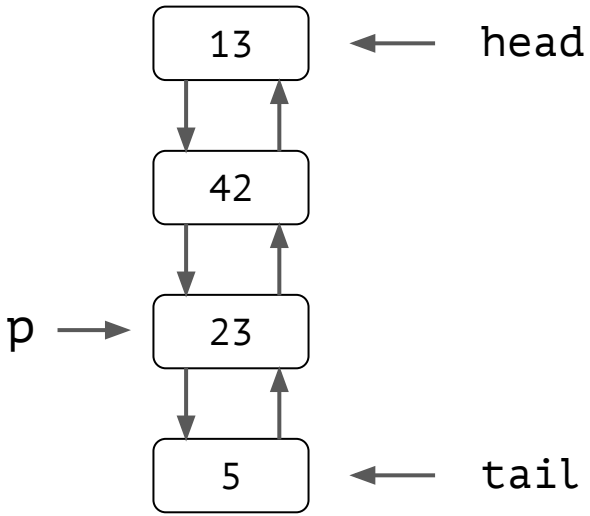
Понимая это, легко построить контр-пример на эту технику.

Двусвязный список!



Персистентные списки

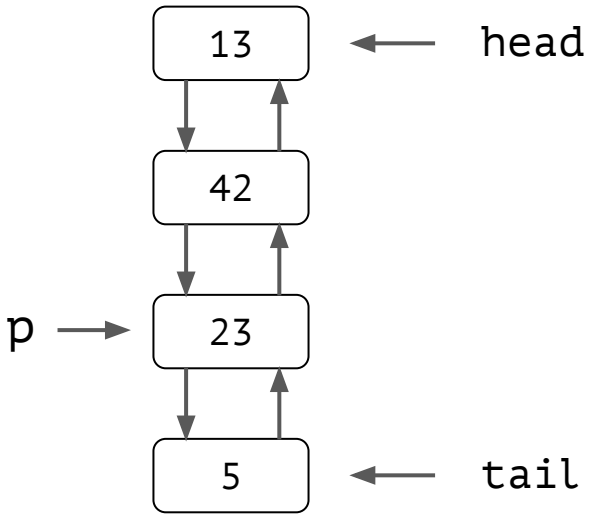
Двусвязный список: update(p, value)
get(t, p)



Персистентные списки

Двусвязный список: update(p, value)
get(t, p)

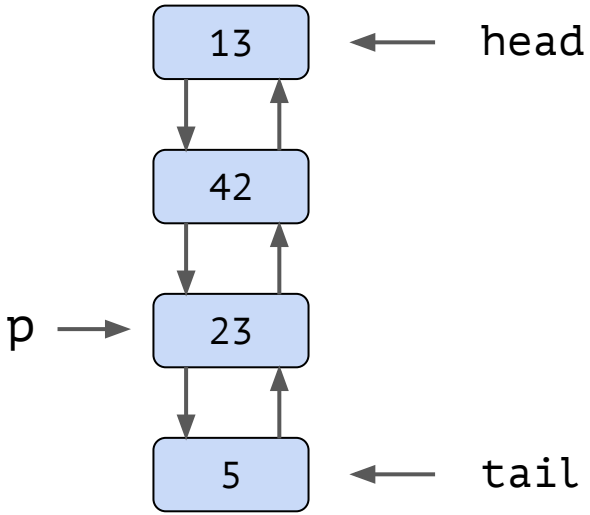
Из каких элементов p достижим?



Персистентные списки

Двусвязный список: update(p, value)
get(t, p)

Из каких элементов p достижим? А из всех!

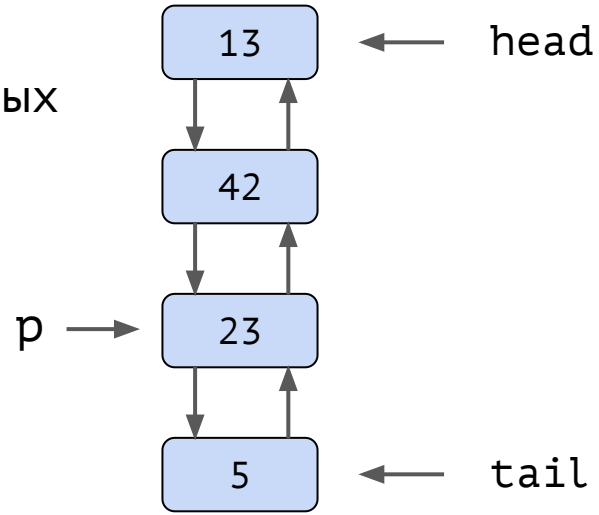


Персистентные списки

Двусвязный список: update(p, value)
get(t, p)

Из каких элементов p достижим? А из всех!

Получается, придется **все** копировать,
т.е.это персистентная структура данных
через полное копирование



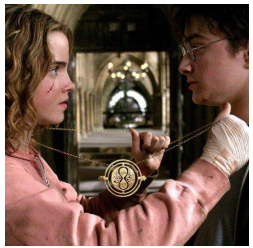
Персистентные списки

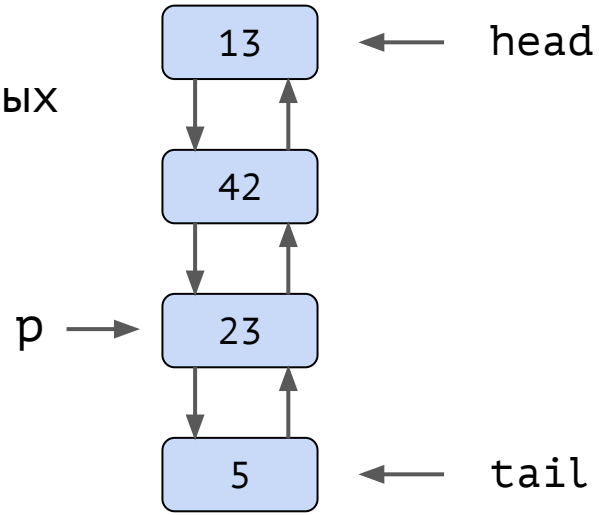
Двусвязный список: update(p, value)
get(t, p)

Из каких элементов p достижим? А из всех!

Получается, придется **все** копировать,
т.е.это персистентная структура данных
через полное копирование

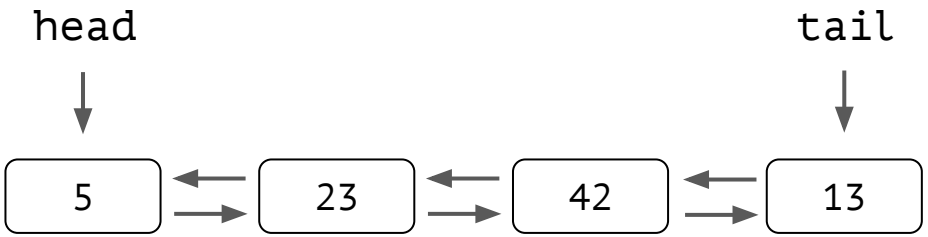
Может можно как-то еще?





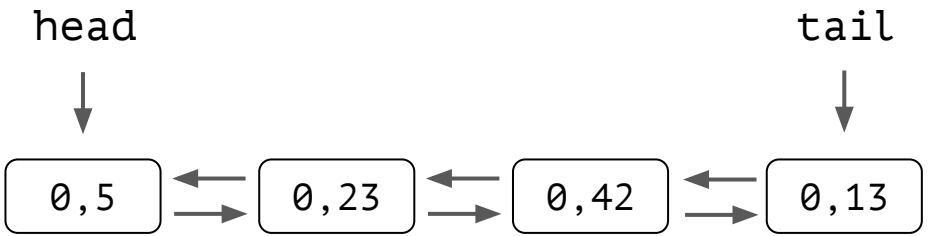
Персистентные списки: толстые узлы

Двусвязный список: update(p, value)
get(t, p)



Персистентные списки: толстые узлы

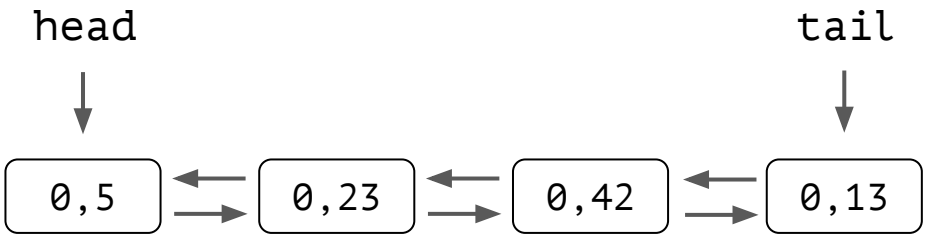
Двусвязный список: update(p, value)
get(t, p)



Храним не только значения, но и версию

Персистентные списки: толстые узлы

Двусвязный список: update(p, value)
get(t, p)

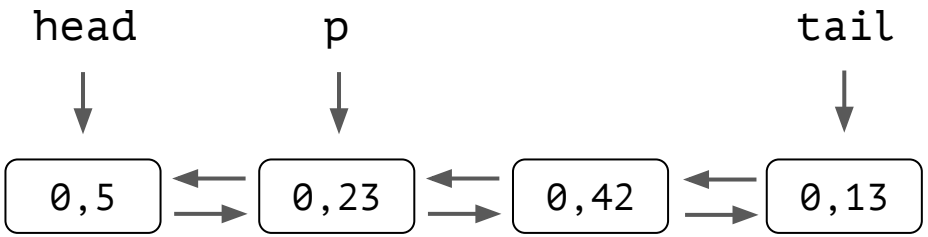


Храним не только значения, но и версию

update инкрементирует версию и раздувает ноду.

Персистентные списки: толстые узлы

Двусвязный список: update(p, value)
get(t, p)



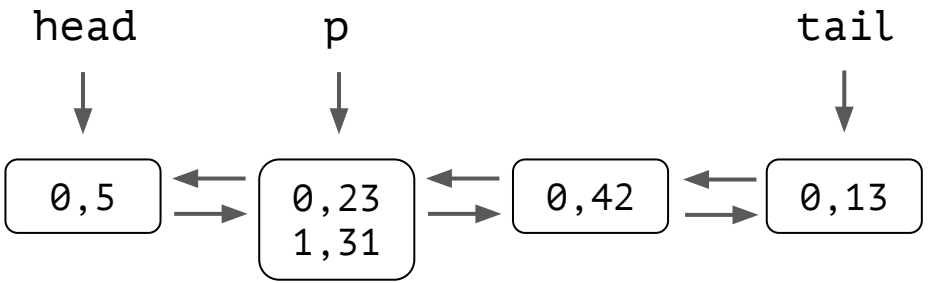
Храним не только значения, но и версию

update инкрементирует версию и раздувает ноду.

update(p, 31)

Персистентные списки: толстые узлы

Двусвязный список: update(p, value)
get(t, p)



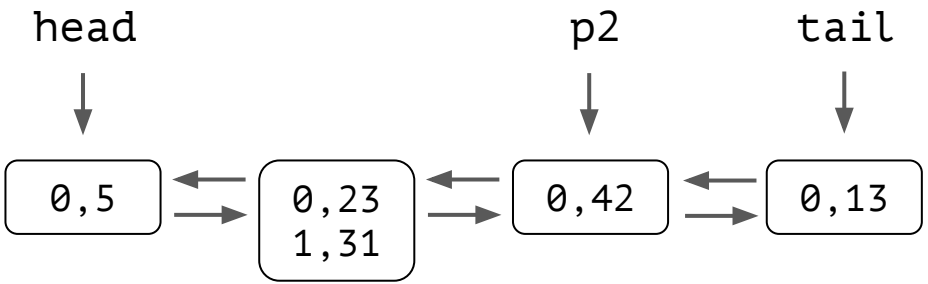
Храним не только значения, но и версию

update инкрементирует версию и раздувает ноду.

update(p, 31)

Персистентные списки: толстые узлы

Двусвязный список: update(p, value)
get(t, p)



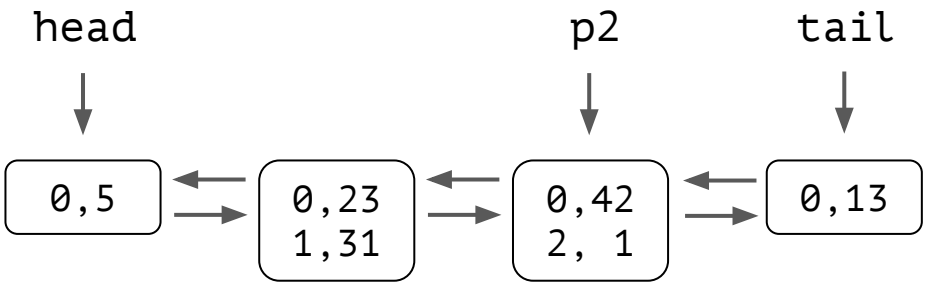
Храним не только значения, но и версию

update инкрементирует версию и раздувает ноду.

update(p, 31)
update(p2, 1)

Персистентные списки: толстые узлы

Двусвязный список: update(p, value)
get(t, p)



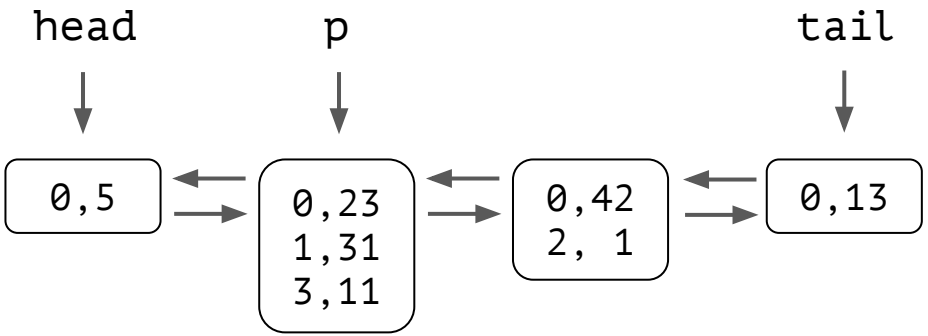
Храним не только значения, но и версию

update инкрементирует версию и раздувает ноду.

update(p, 31)
update(p2, 1)

Персистентные списки: толстые узлы

Двусвязный список: update(p, value)
get(t, p)



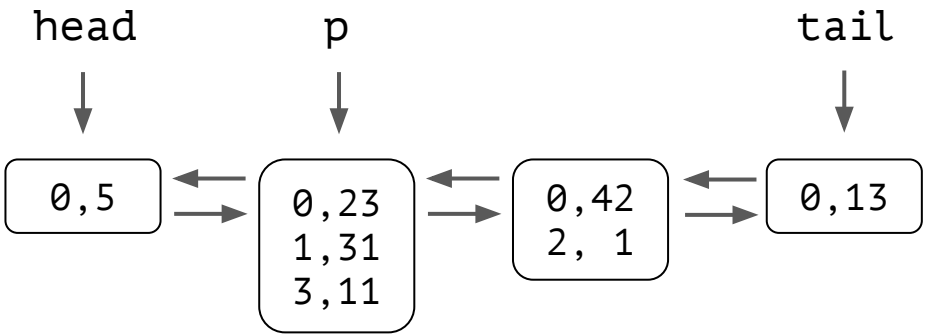
Храним не только значения, но и версию

update инкрементирует версию и раздувает ноду.

update(p, 31)
update(p2, 1)
update(p, 11)

Персистентные списки: толстые узлы

Двусвязный список: update(p, value)
get(t, p)



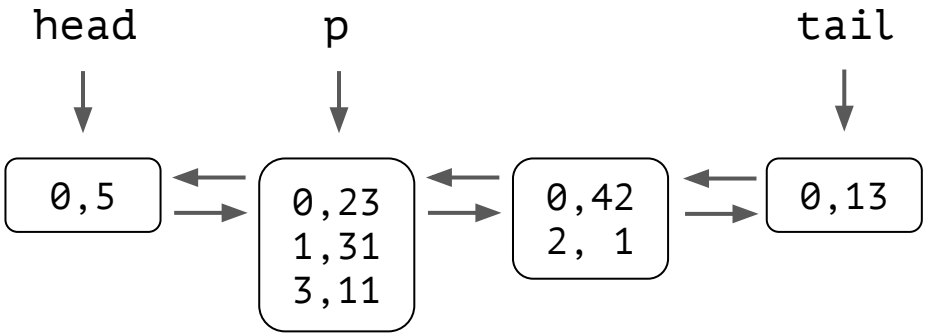
Храним не только значения, но и версию

update инкрементирует версию и раздувает ноду.

Как тогда работает get?

Персистентные списки: толстые узлы

Двусвязный список: update(p, value)
get(t, p)



get(2, p)?

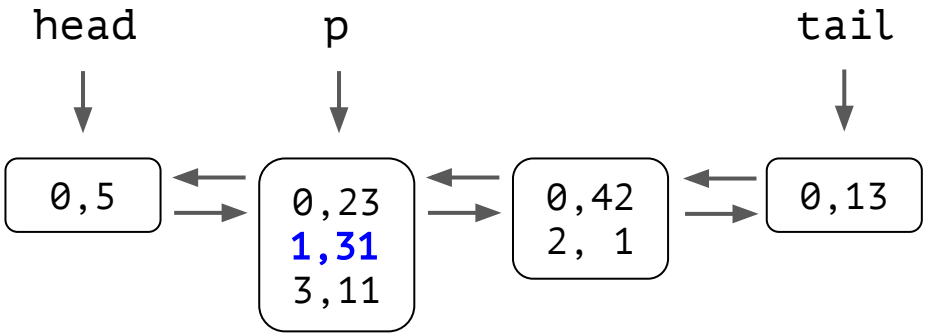
Храним не только значения, но и версию

update инкрементирует версию и раздувает ноду.

Как тогда работает get?

Персистентные списки: толстые узлы

Двусвязный список: update(p, value)
get(t, p)



get(2, p)?

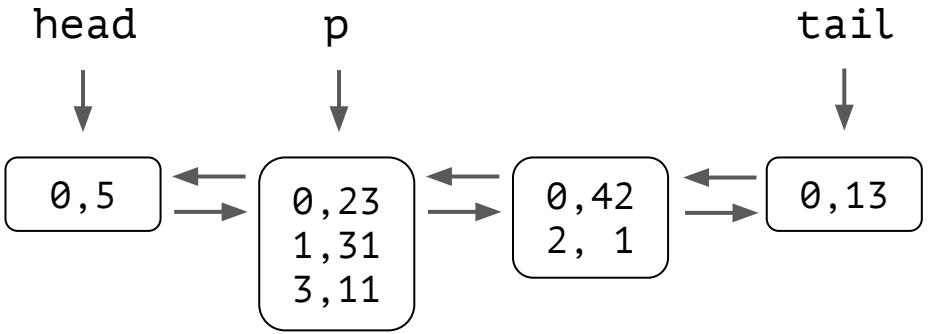
Храним не только значения, но и версию

update инкрементирует версию и раздувает ноду.

Как тогда работает get? Бинарный поиск до ближайшего <= чем то, что мы ищем.

Персистентные списки: толстые узлы

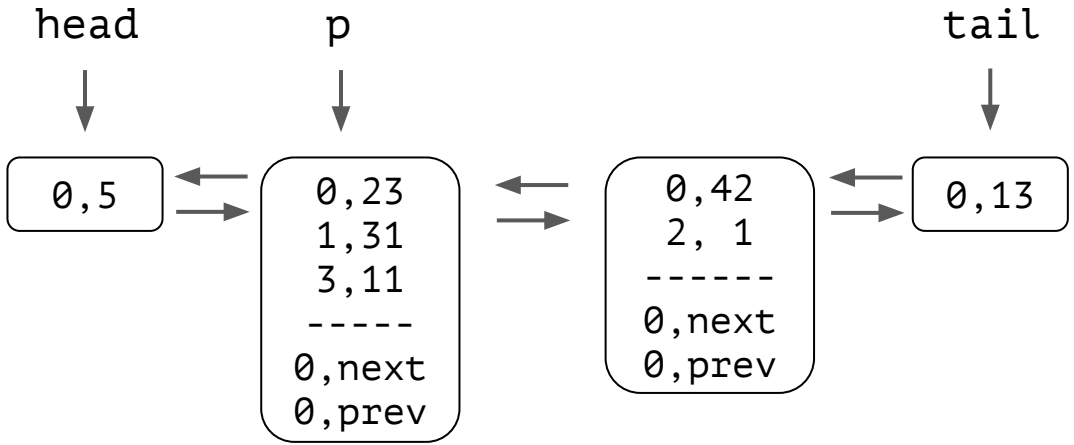
Двусвязный список: update(p, value)
get(t, p)
insert(p, value) <- вставить новую ноду после p?



insert(p, 21)

Персистентные списки: толстые узлы

Двусвязный список: update(p, value)
get(t, p)
insert(p, value) <- вставить новую ноду после p?

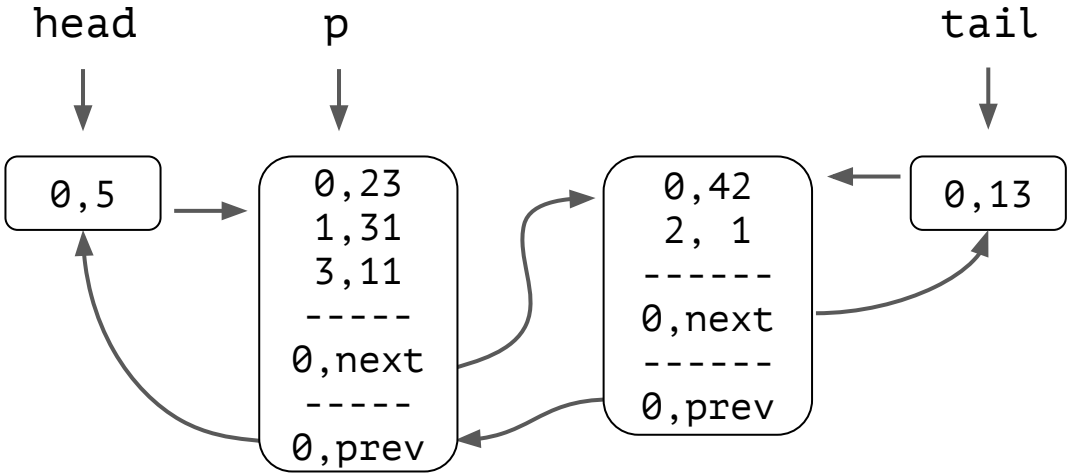


insert(p, 21)

next и prev - это ведь тоже поля

Персистентные списки: толстые узлы

Двусвязный список: update(p, value)
get(t, p)
insert(p, value) <- вставить новую ноду после p?

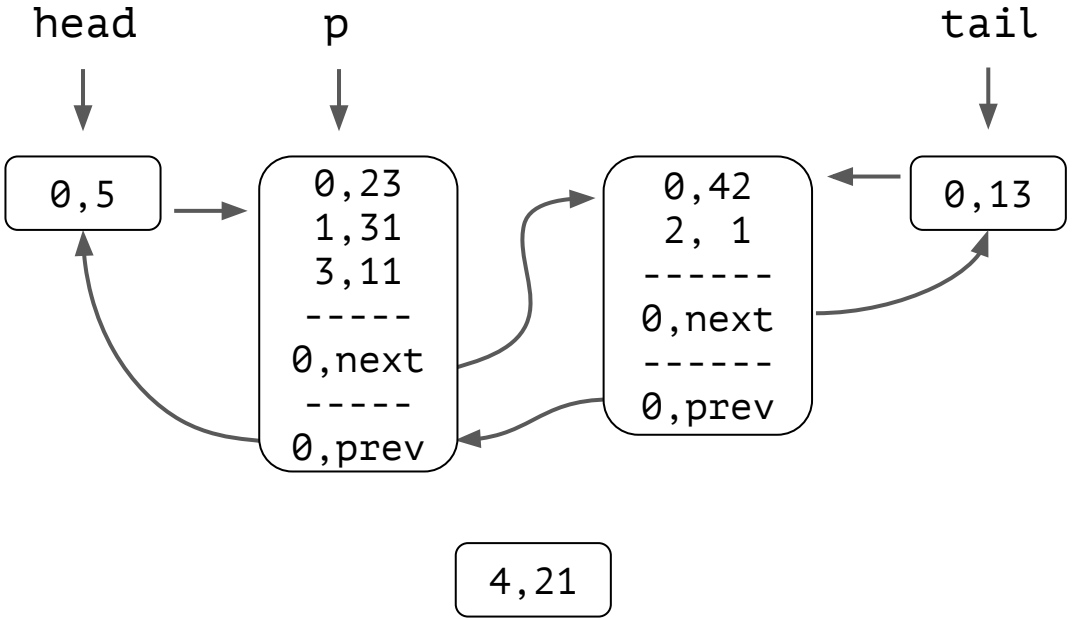


insert(p, 21)

next и prev - это ведь тоже поля

Персистентные списки: толстые узлы

Двусвязный список: update(p, value)
get(t, p)
insert(p, value) <- вставить новую ноду после p?

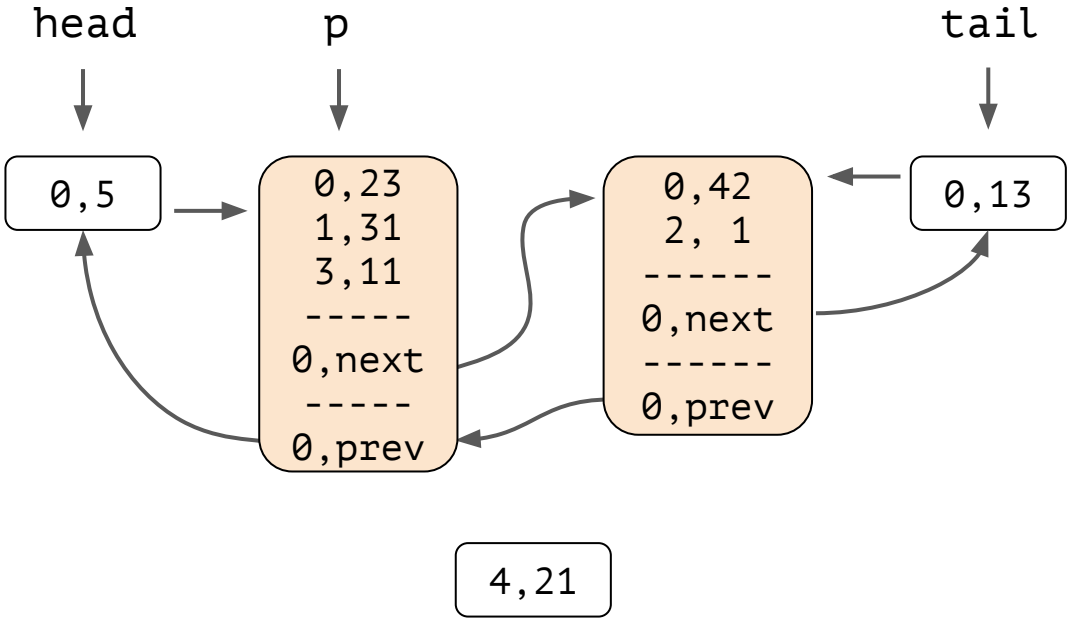


insert(p, 21)

next и prev - это ведь тоже поля

Персистентные списки: толстые узлы

Двусвязный список: update(p, value)
get(t, p)
insert(p, value) <- вставить новую ноду после p?

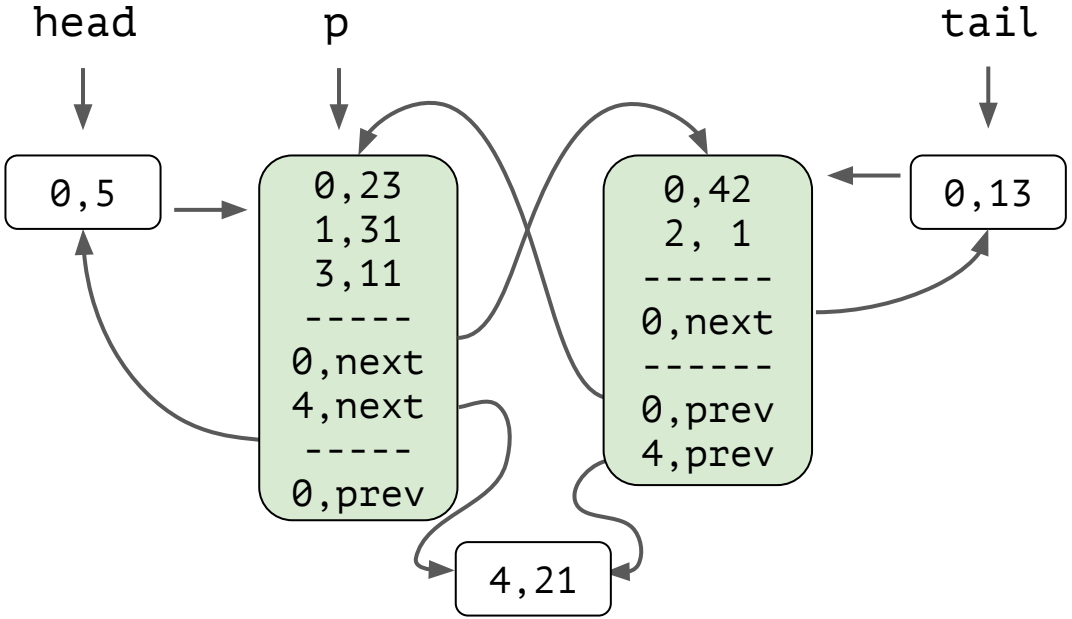


insert(p, 21)

next и prev - это ведь тоже поля

Персистентные списки: толстые узлы

Двусвязный список: update(p, value)
get(t, p)
insert(p, value) <- вставить новую ноду после p?



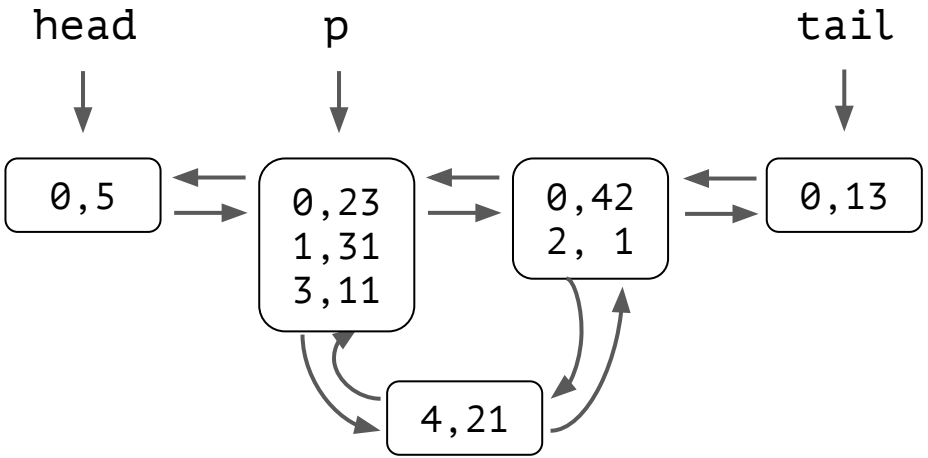
insert(p, 21)

next и prev - это ведь тоже поля

их утолщаем по тем же самым правилам

Персистентные списки: толстые узлы

Двусвязный список: update(p, value)
get(t, p)
insert(p, value) <- вставить новую ноду после p?



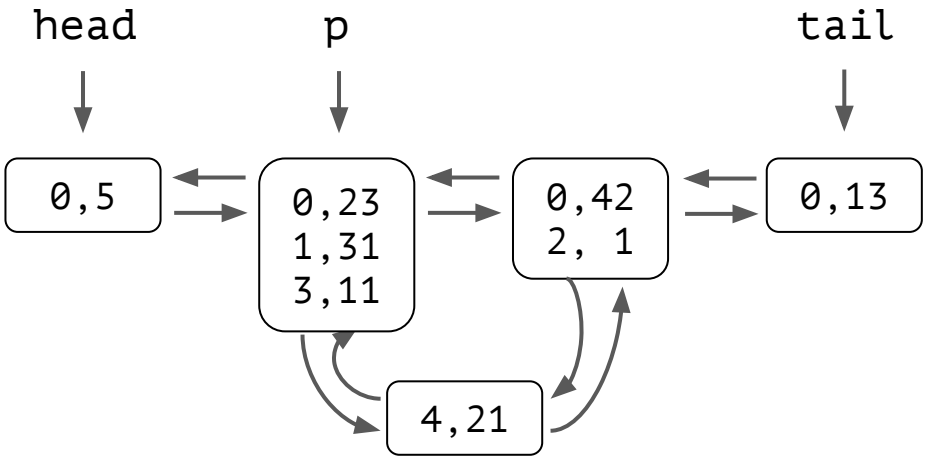
insert(p, 21)

next и prev - это ведь тоже поля

их утолщаем по тем же самым правилам

Персистентные списки: толстые узлы

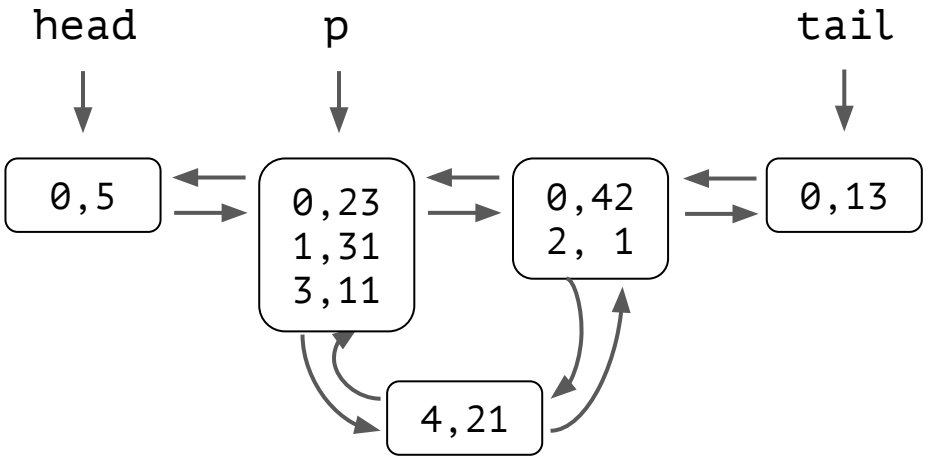
Двусвязный список: update(p, value)
get(t, p)
insert(p, value) <- вставить новую ноду после p?



Время: f -> f * logK
Память: O(K)
K - кол-во запросов
Част. персистентность

Персистентные списки: толстые узлы

Двусвязный список: update(p, value)
get(t, p)
insert(p, value) <- вставить новую ноду после p?

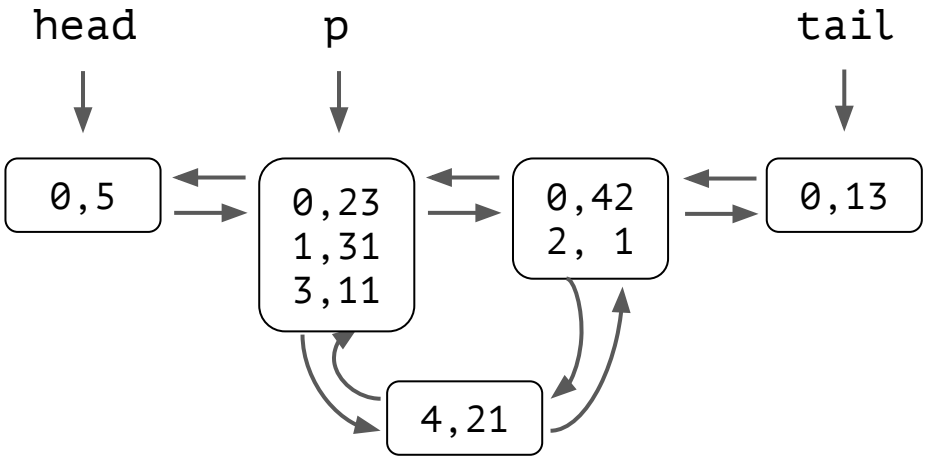


Время: f -> f * logK
Память: O(K)
K - кол-во запросов
Част. персистентность

Можем ли мы лучше?

Толстые узлы + копирование путей

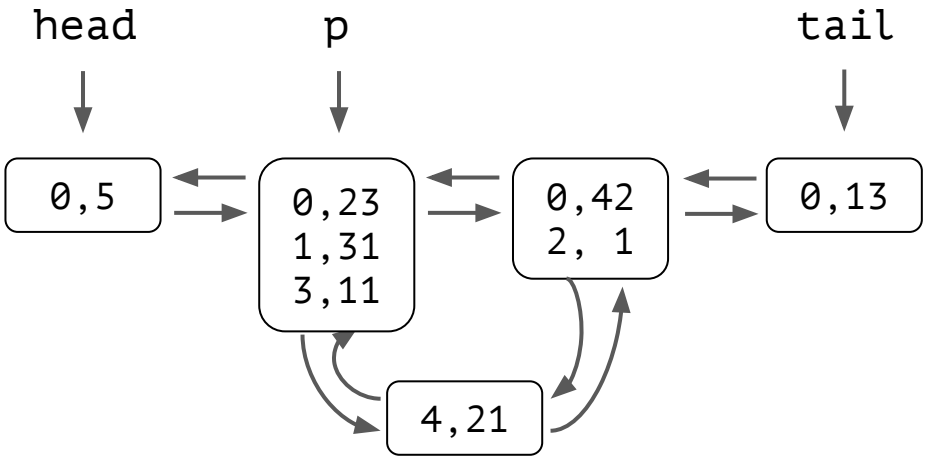
Двусвязный список: update(p, value)
get(t, p)
insert(p, value) <- вставить новую ноду после p?



Идея: давайте ограничим количество версий в каждом толстом узле константой.

Толстые узлы + копирование путей

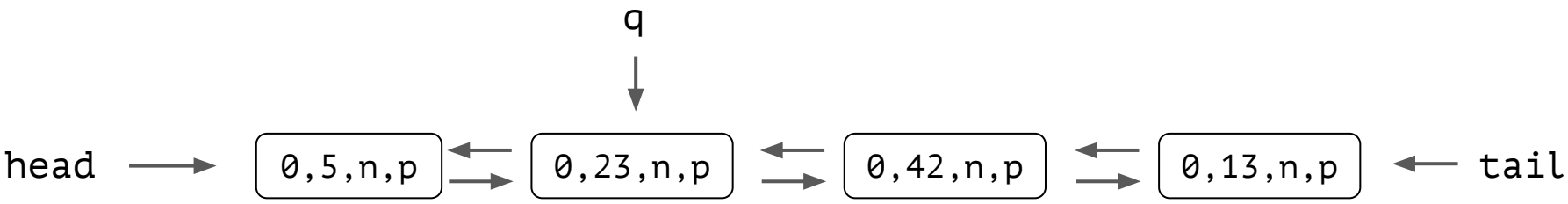
Двусвязный список: update(p, value)
get(t, p)
insert(p, value) <- вставить новую ноду после p?



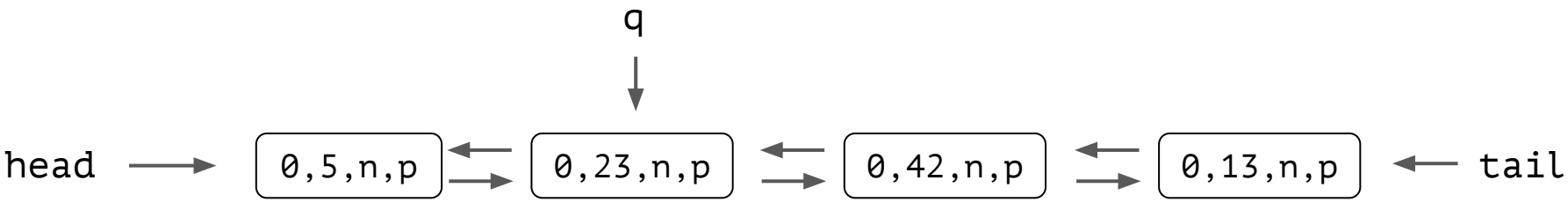
Идея: давайте ограничим количество версий в каждом толстом узле константой.

Например, константой 2.

Толстые узлы + копирование путей

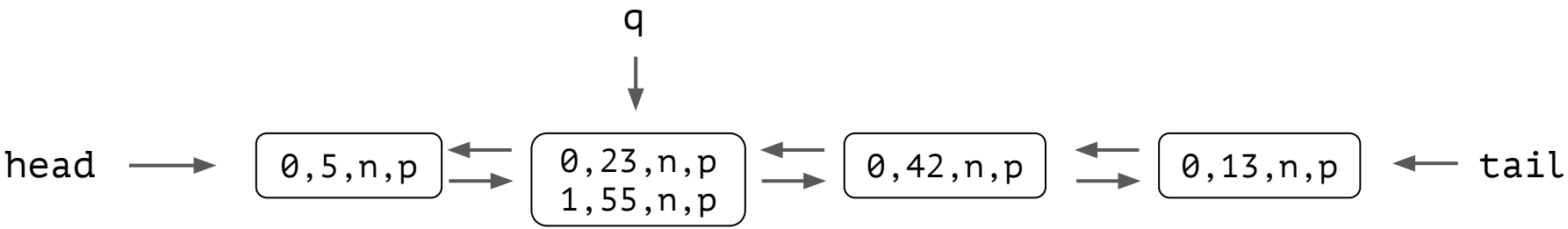


Толстые узлы + копирование путей



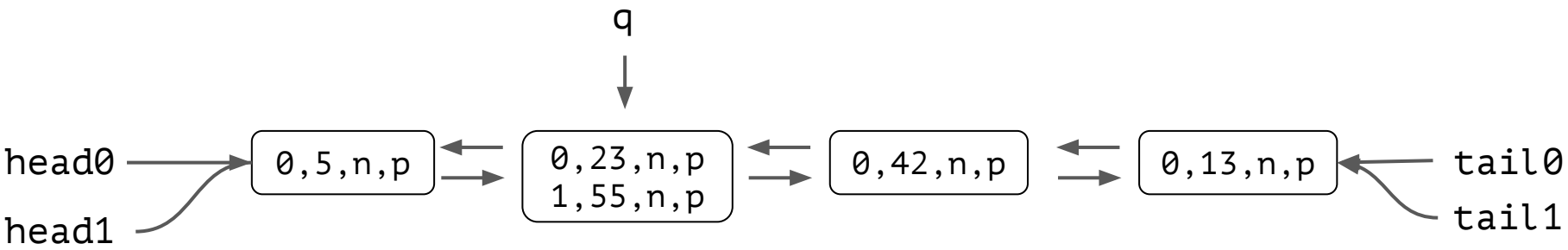

```
update(q, 55)
```

Толстые узлы + копирование путей



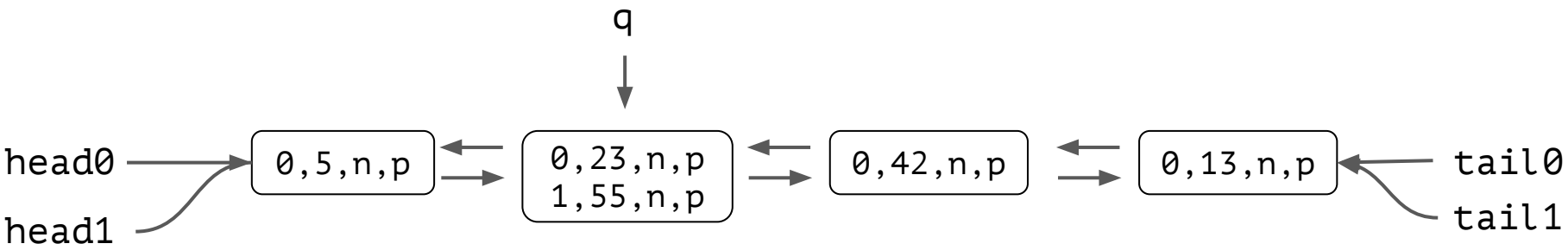

```
update(q, 55)
```

Толстые узлы + копирование путей



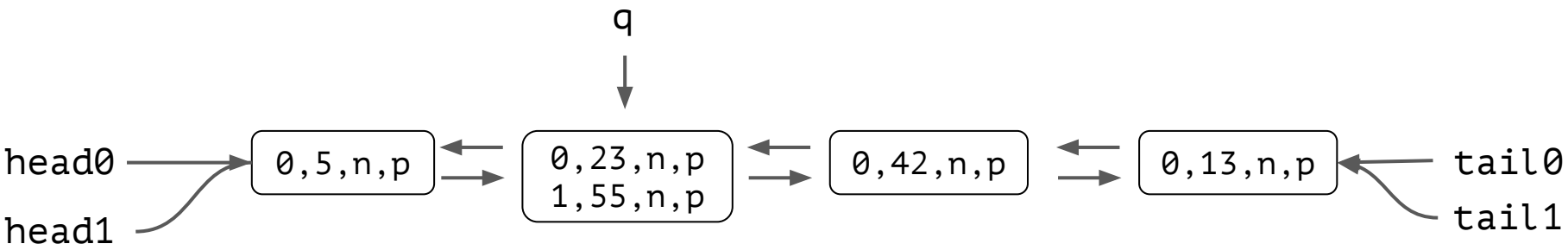
update(q, 55) для каждой версии храним head/tail

Толстые узлы + копирование путей



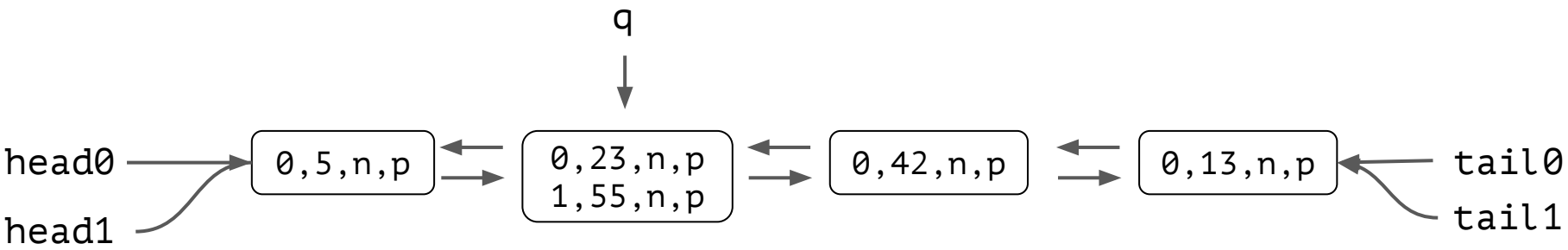

```
update(q, 55)
update(q, 33)
```

Толстые узлы + копирование путей



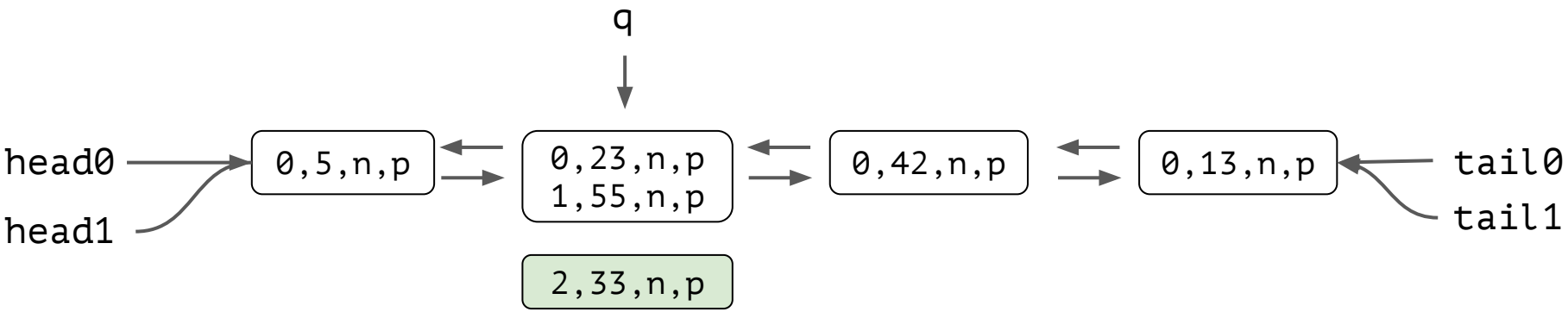

```
update(q, 55)
update(q, 33)   раздувать больше не хочется
```

Толстые узлы + копирование путей



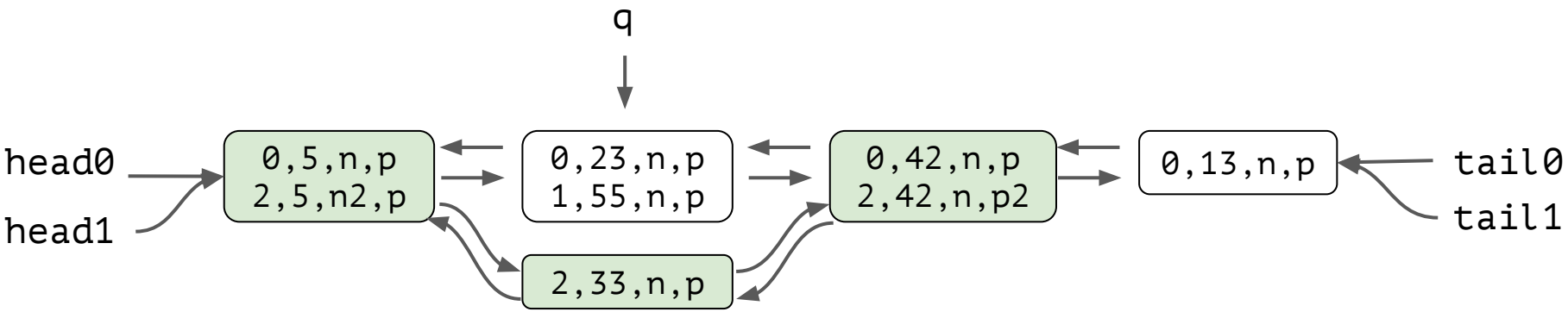

```
update(q, 55)
update(q, 33)   раздувать больше не хочется,
                поэтому мы создадим новый узел!
```

Толстые узлы + копирование путей



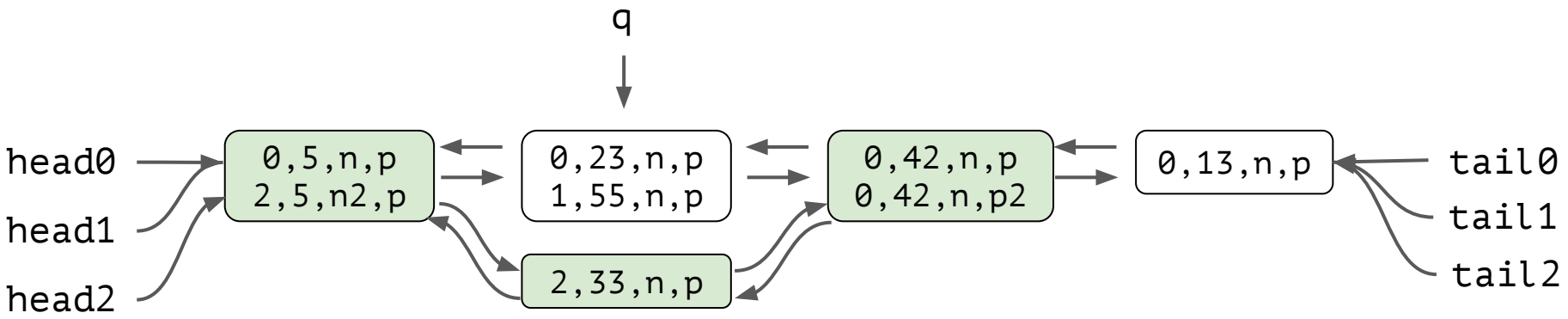

```
update(q, 55)
update(q, 33)   раздувать больше не хочется,
                поэтому мы создадим новый узел!
```

Толстые узлы + копирование путей



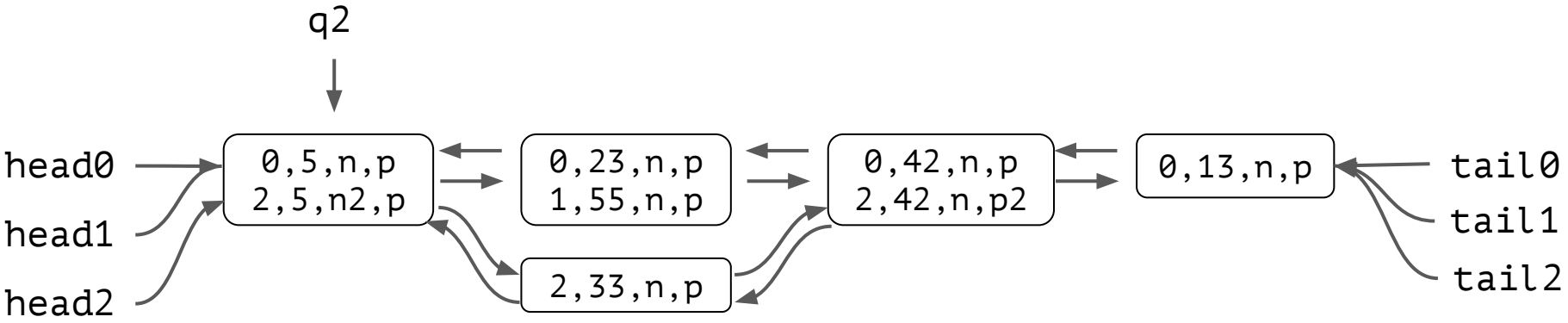

```
update(q, 55)
update(q, 33)   раздувать больше не хочется,
                поэтому мы создадим новый узел!
```

Толстые узлы + копирование путей



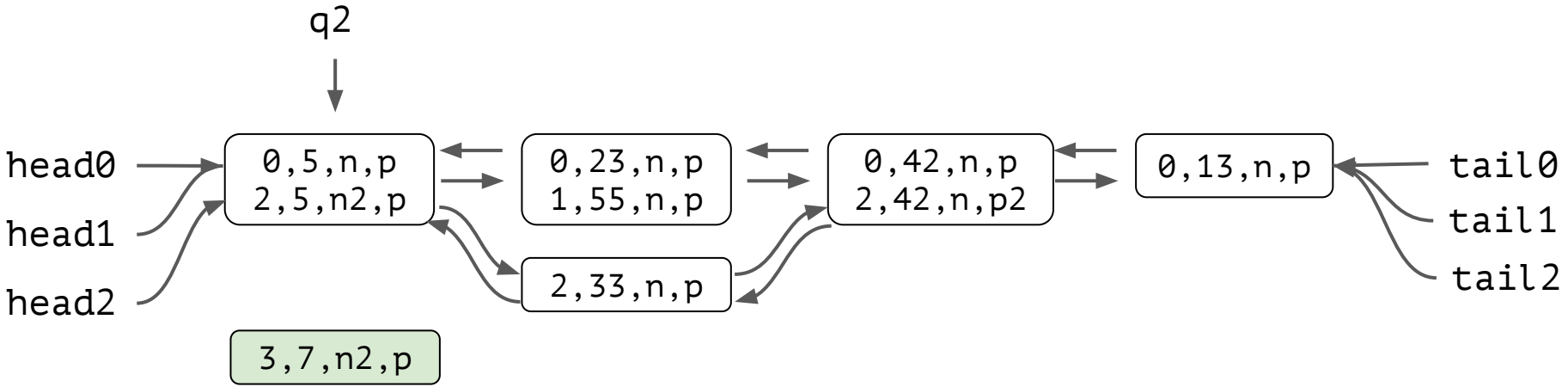

```
update(q, 55)
update(q, 33)   раздувать больше не хочется,
                поэтому мы создадим новый узел!
```

Толстые узлы + копирование путей



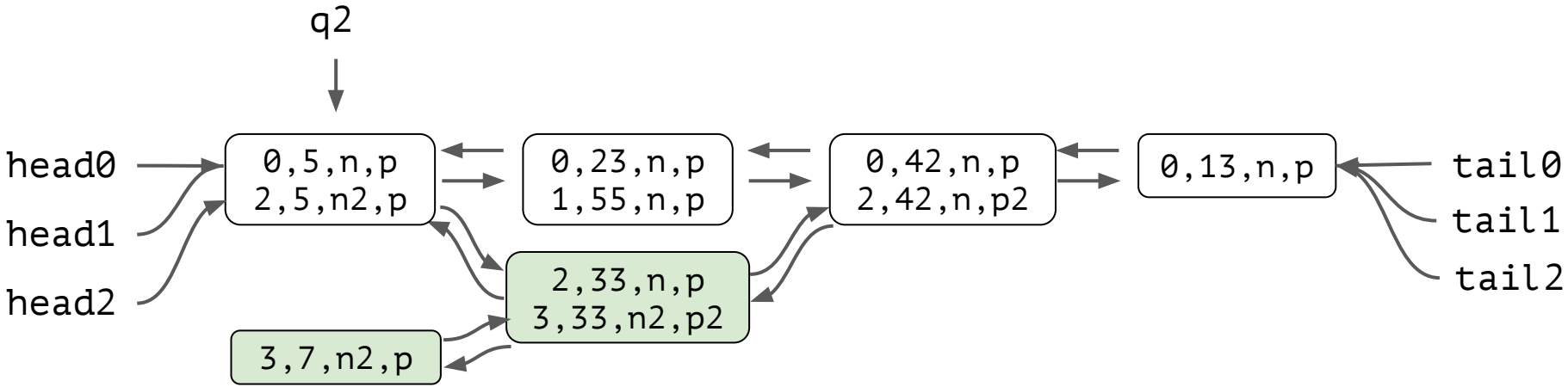

```
update(q, 55)
update(q, 33)
update(q2, 7)
```

Толстые узлы + копирование путей



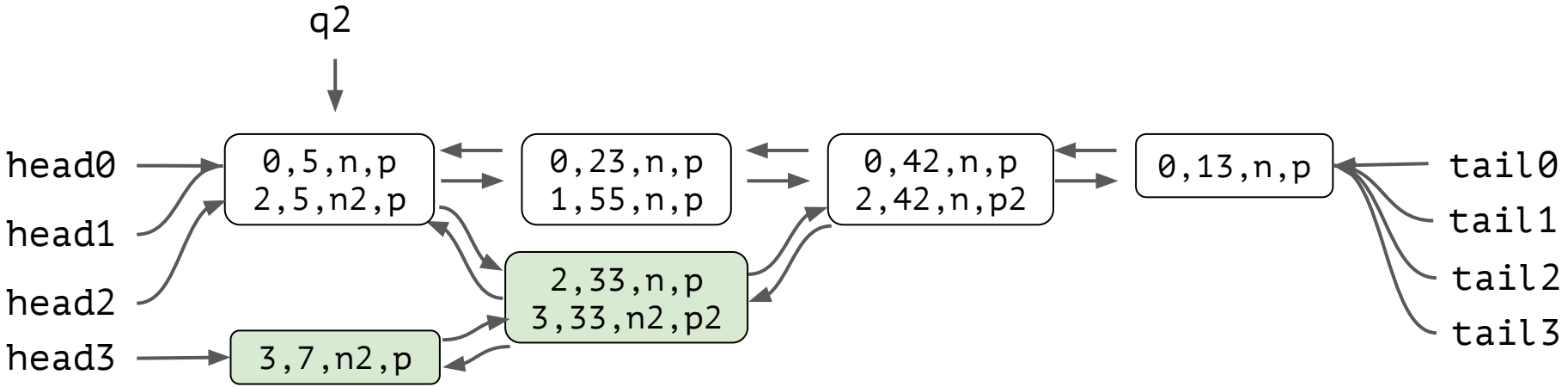

```
update(q, 55)
update(q, 33)
update(q2, 7)
```

Толстые узлы + копирование путей



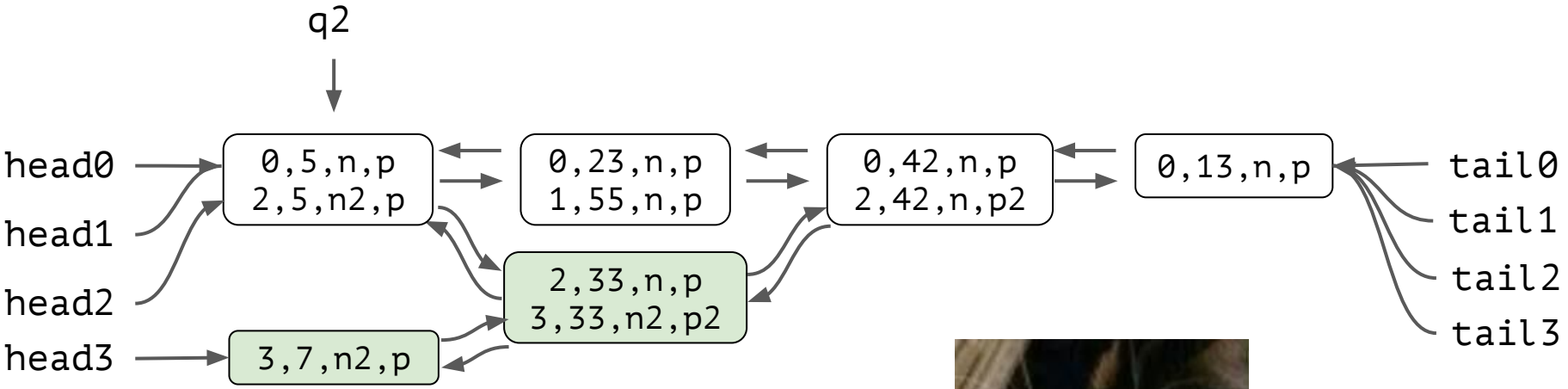

```
update(q, 55)
update(q, 33)
update(q2, 7)
```

Толстые узлы + копирование путей



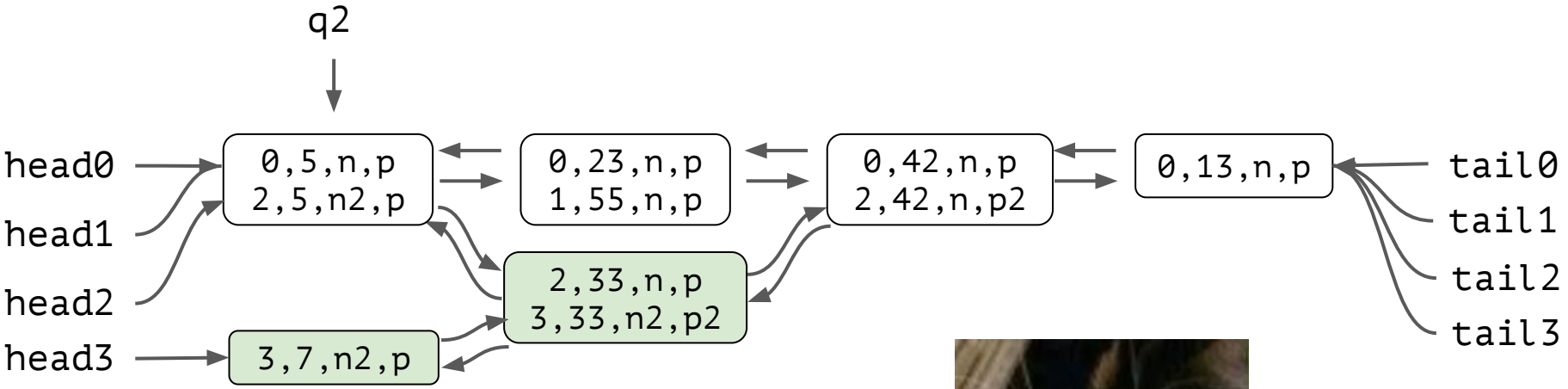

```
update(q, 55)
update(q, 33)
update(q2, 7)
```

Толстые узлы + копирование путей




```
update(q, 55)
update(q, 33)
update(q2, 7)
```

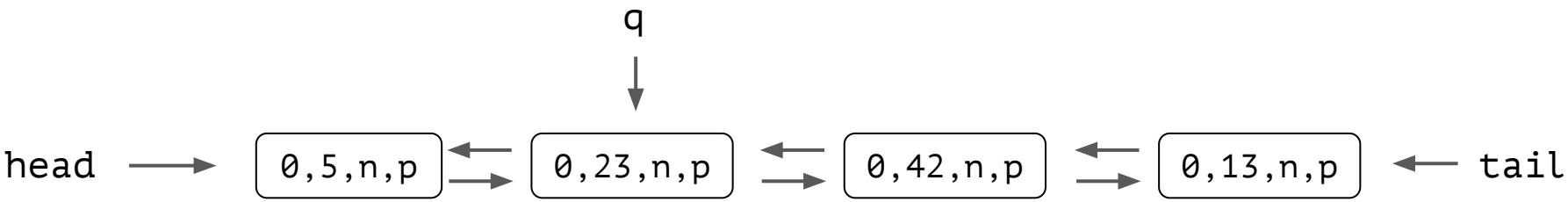
Толстые узлы + копирование путей




```
update(q, 55)
update(q, 33)
update(q2, 7)
```

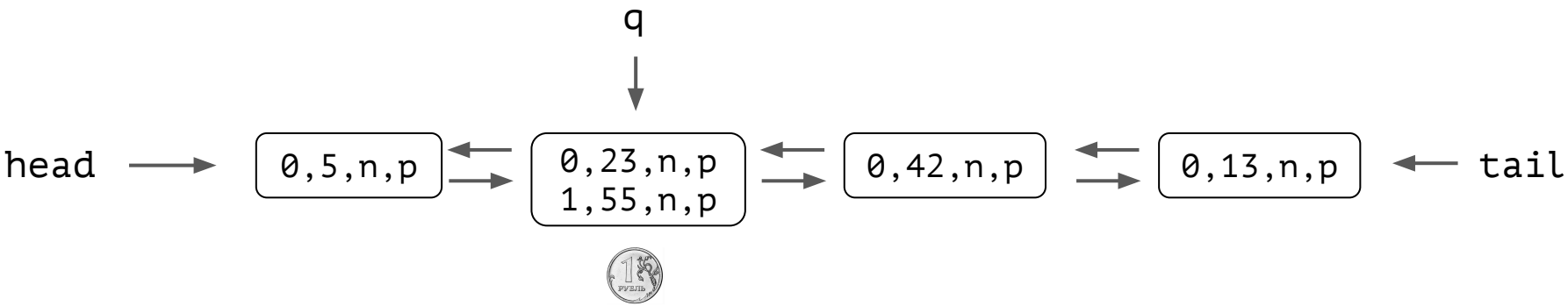
Ради амортизации!

Толстые узлы + копирование путей




```
update(q, 55)
```

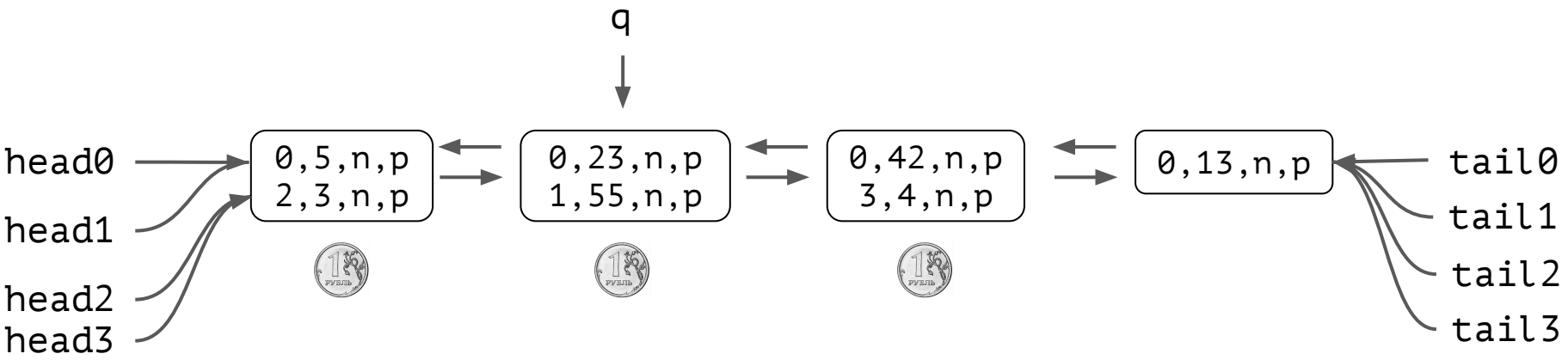
Толстые узлы + копирование путей



update(q, 55)

Дописать версию без перекопирования - быстрое действие, оставляем монету

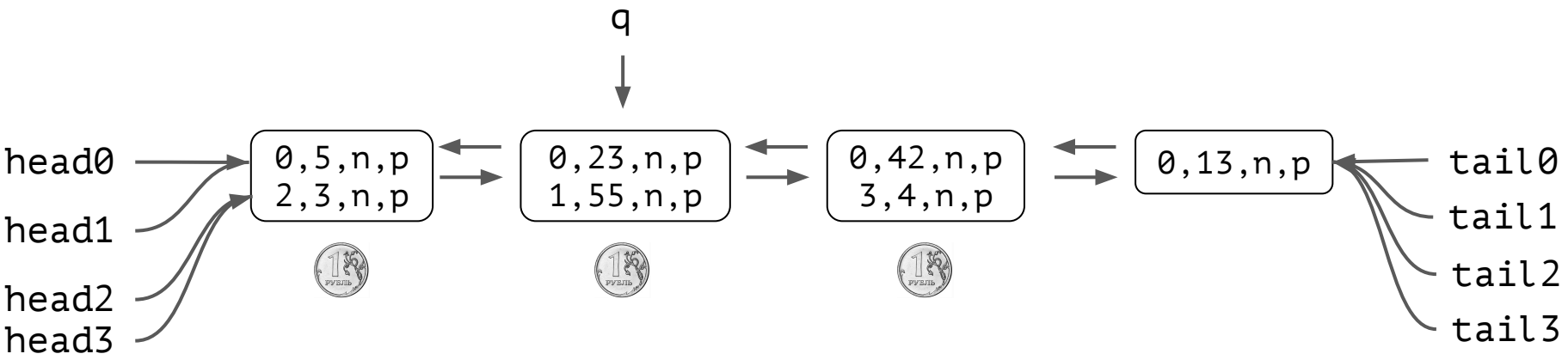
Толстые узлы + копирование путей




```
update(q, 55)
update(q.p, 3)
update(q.n, 4)
```

Дописать версию без перекопирования - быстрое действие, оставляем монету

Толстые узлы + копирование путей

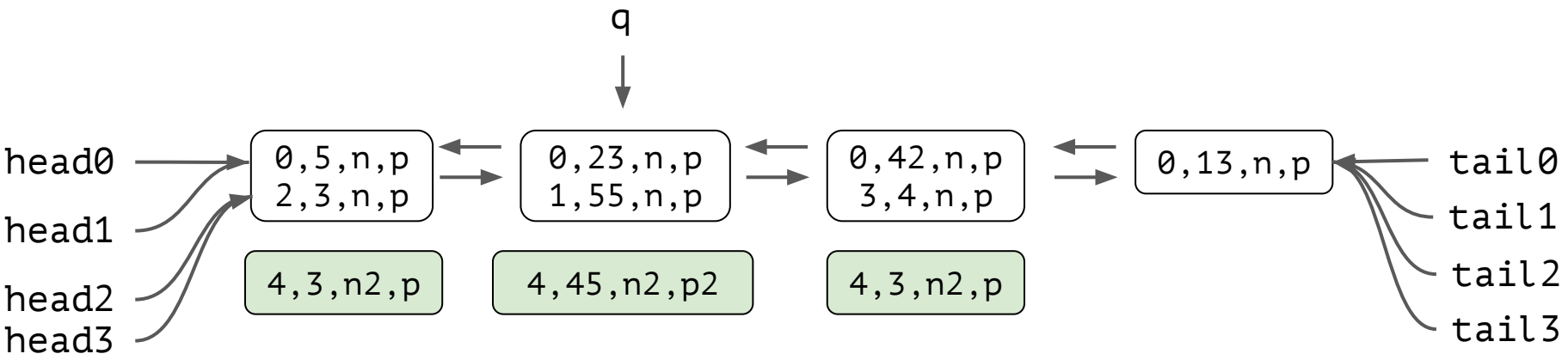



```
update(q, 55)
update(q.p, 3)
update(q.n, 4)
update(q, 45)
```

вызовет каскадное копирование

Дописать версию без перекопирования - быстрое действие, оставляем монету

Толстые узлы + копирование путей

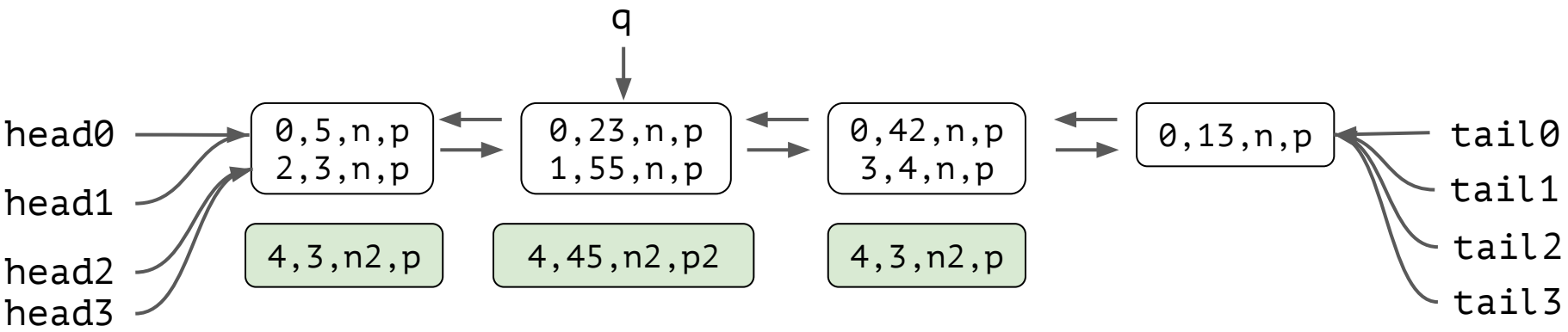



```
update(q, 55)
update(q.p, 3)
update(q.n, 4)
update(q, 45)
```

Дописать версию без перекопирования - быстрое действие, оставляем монету

вызовет каскадное копирование, но копироваться будут только толстые узлы, а на них есть монетки!

Толстые узлы + копирование путей




```
update(q, 55)
update(q.p, 3)
update(q.n, 4)
update(q, 45)
```

Дописать версию без перекопирования - быстрое действие, оставляем монету

вызовет каскадное копирование, но копироваться будут только толстые узлы, а на них есть монетки!

Что дает нам амортизационную сложность операций O*(1) 🎉🎉

Толстые узлы + копирование путей

- Больше не боимся большой степени достижимости в структурах данных! 🎉🎉

Толстые узлы + копирование путей

- Больше не боимся большой степени достижимости в структурах данных! 🎉🎉
- Алгоритм обобщается и до случая полной персистентности

Толстые узлы + копирование путей

- Больше не боимся большой степени достижимости в структурах данных! 🎉🎉
- Алгоритм обобщается и до случая полной персистентности
- Алгоритм обобщается до любой pointer machine структуры данных (с ограничением на количество входных дуг в каждую вершину)

Takeaways

- Персистентные структуры данных, как концепция
- Частичная и полная персистентности
- Реализация: методы полного копирования, копирования путей, толстых узлов и их комбинации.

Мини-задача #47 (2 балла, дополнительная)

Реализуйте очередь с полной персистентностью.

Оцените временную и емкостную сложность операций.

Сложнее, чем может показаться!