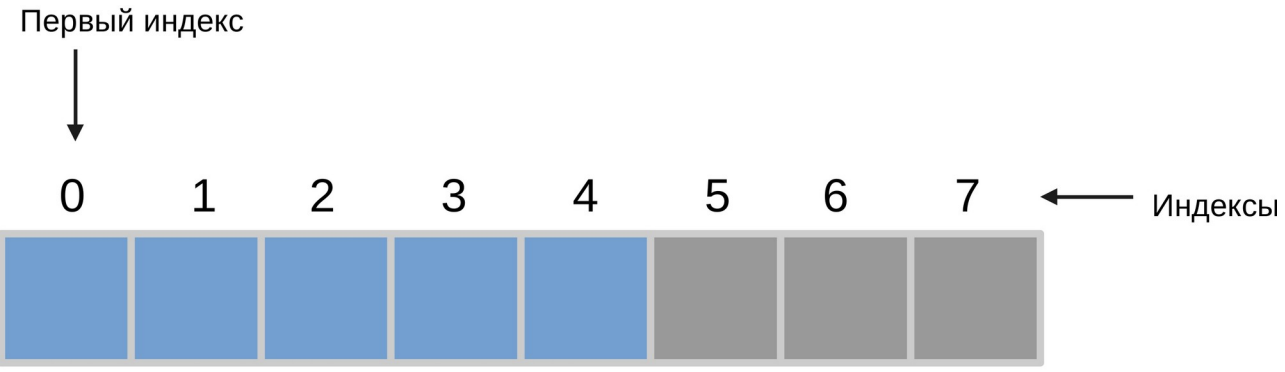
Технология и методы программирования

Тема 8. Структуры данных.

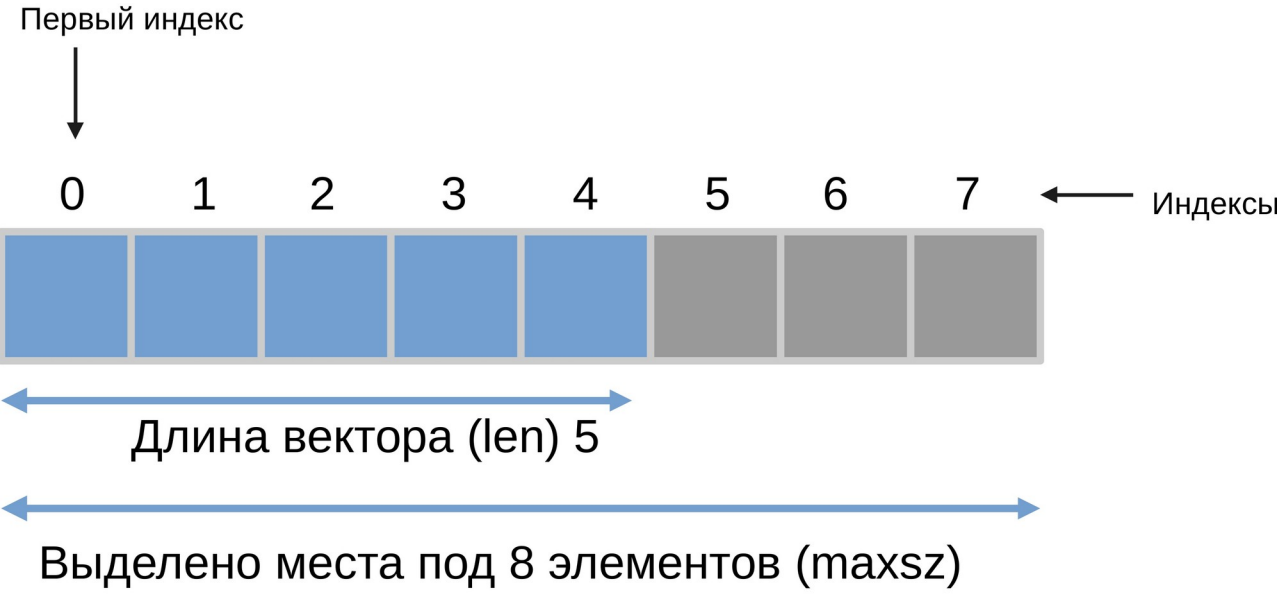
Вектор (vector)




```
struct vector {
        size_t len;
        size_t maxsz;
        T *data;
};
```

Метод	Сложность
get(index)	???
push(item)	???
pop()	???
insert(index)	???
remove(index)	???
find(node)	???

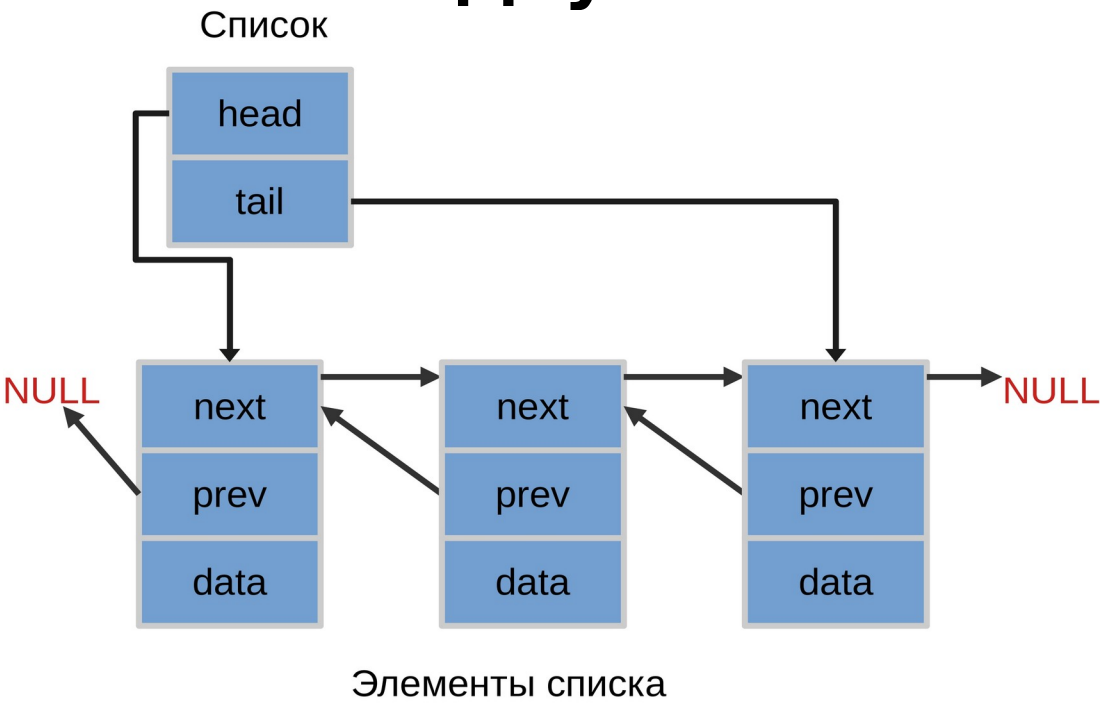
Вектор (vector)




```
struct vector {
        size_t len;
        size_t maxsz;
        T *data;
};
```

Метод	Сложность
get(index)	O(1)
push(item)	O(1)
pop()	O(1)
insert(index)	O(n)
remove(index)	O(n)
find(node)	O(n)

Двусвязный список (double linked list)



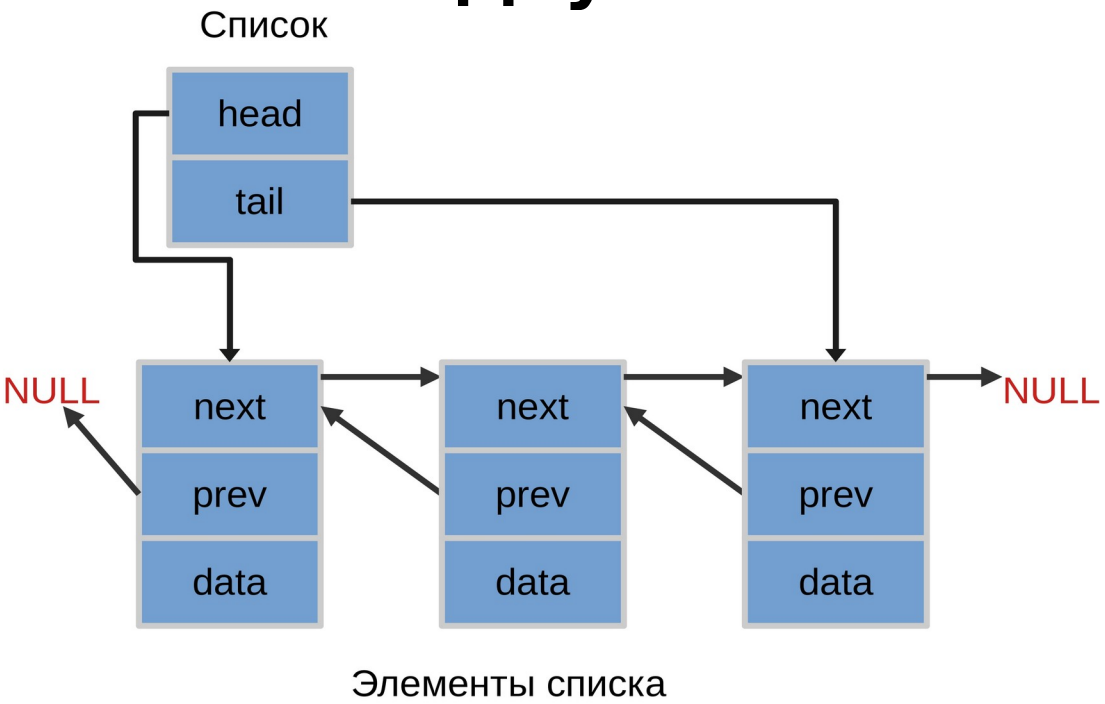
Элементы списка

```
struct list {
        list_node *head;
        list_node *tail;
};
struct list_node {
        struct list_node *next;
        struct list_node *prev;
};
```

Метод	Сложность
push_front(n) push_back(n)	???
pop_front() pop_back()	???
get(index)	???
insert_node(n)	???
remove_node(n)	???
find_node(l)	???

* -- в случае если у нас есть указатель на элемент перед/после которого нужно вставлять

Двусвязный список (double linked list)



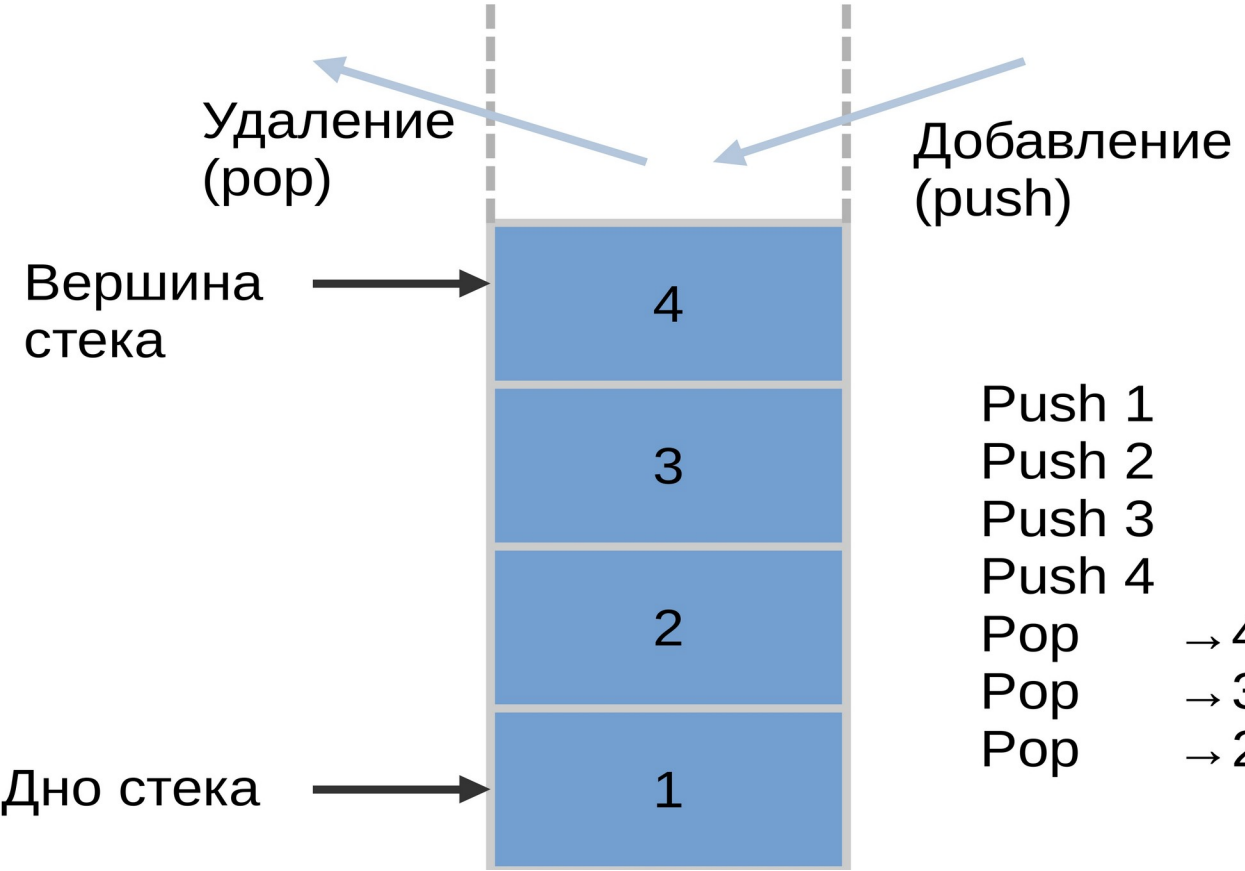
Элементы списка

```
struct list {
        list_node *head;
        list_node *tail;
};
struct list_node {
        struct list_node *next;
        struct list_node *prev;
};
```

Метод	Сложность
push_front(n) push_back(n)	O(1)
pop_front() pop_back()	O(1)
get(index)	O(n)
insert_node(n)	O(1) *
remove_node(n)	O(1)
find_node(l)	O(n)

* -- в случае если у нас есть указатель на элемент перед/после которого нужно вставлять

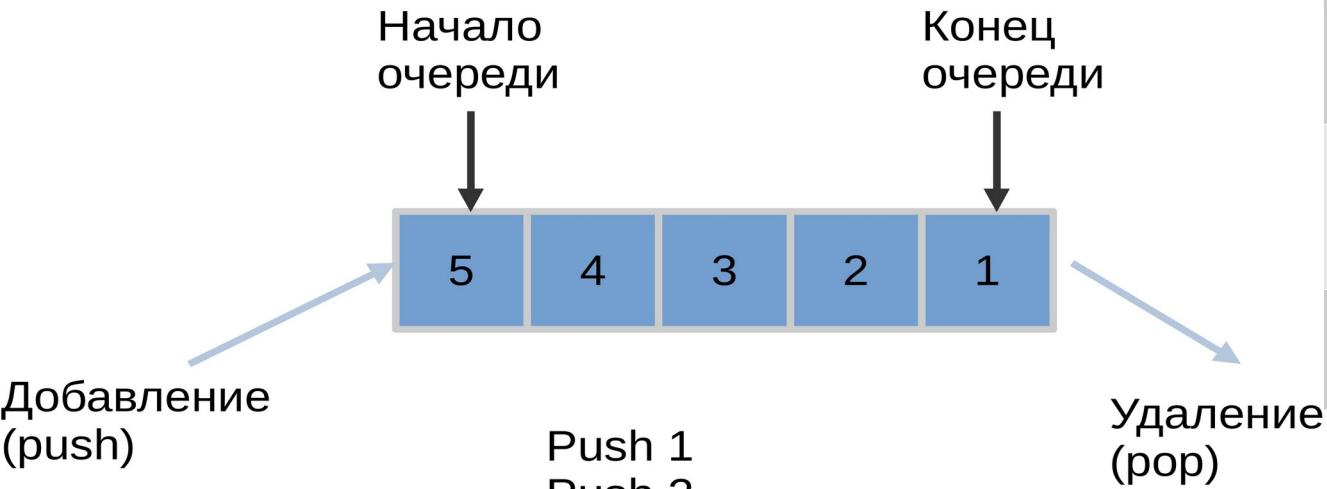
Стек (stack, FILO)



Push 1
Push 2
Push 3
Push 4
Pop →4
Pop →3
Pop →2

Метод	Сложность
push(n)	O(1)
pop()	O(1)
peek()	O(1)

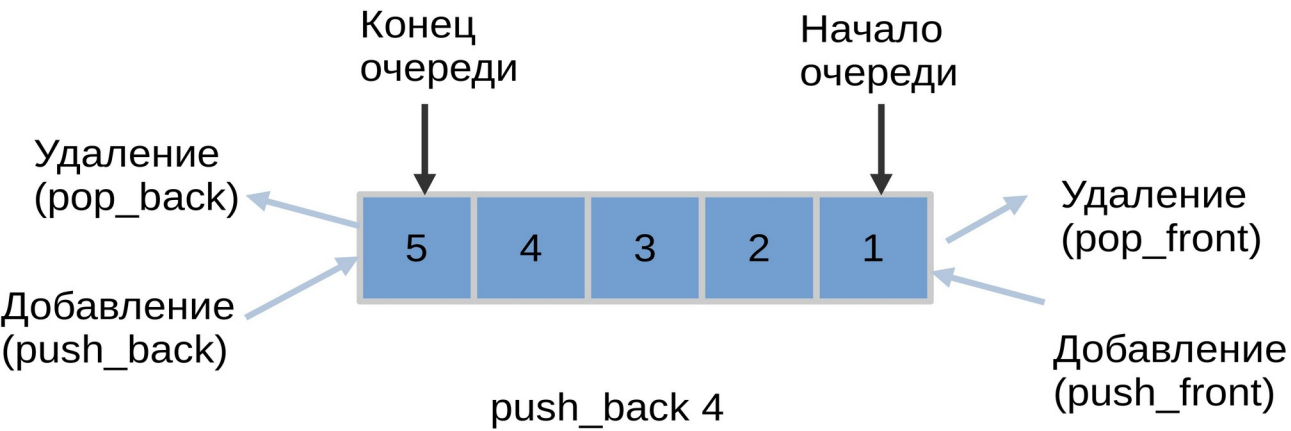
Очередь (queue, FIFO)



Push 1
Push 2
Push 3
Push 4
Push 5
Pop → 1
Pop → 2
Pop → 3

Метод	Сложность
push(n)	O(1)
pop()	O(1)
peek()	O(1)

Дек (deque)




```
push_back 4
push_back 5
push_front 3
push_front 2
push_front 1
pop_front    → 1
pop_back     → 5
pop_front    → 2
```

Метод	Сложность
push_back(n) push_front	O(1)
pop_front() pop_back	O(1)
front() back()	O(1)

Хеш-таблица

Метод	Сложность
insert(n)	O(1)
remove(n)	O(1)
find(n)	O(1)

Хеш-функция

- Функция, которая принимает данные произвольной длины, и возвращает данные фиксированной длины

  Пример хеш-функции:

```
int
stupid_hash(unsigned char *in, int sz)
{
        int hash = 0;
        for (i = 0; i < sz; i++)
                hash += in[i];
        return hash
}
```

  Хеширование «невозвратно», по выходу нельзя понять что было на входе.

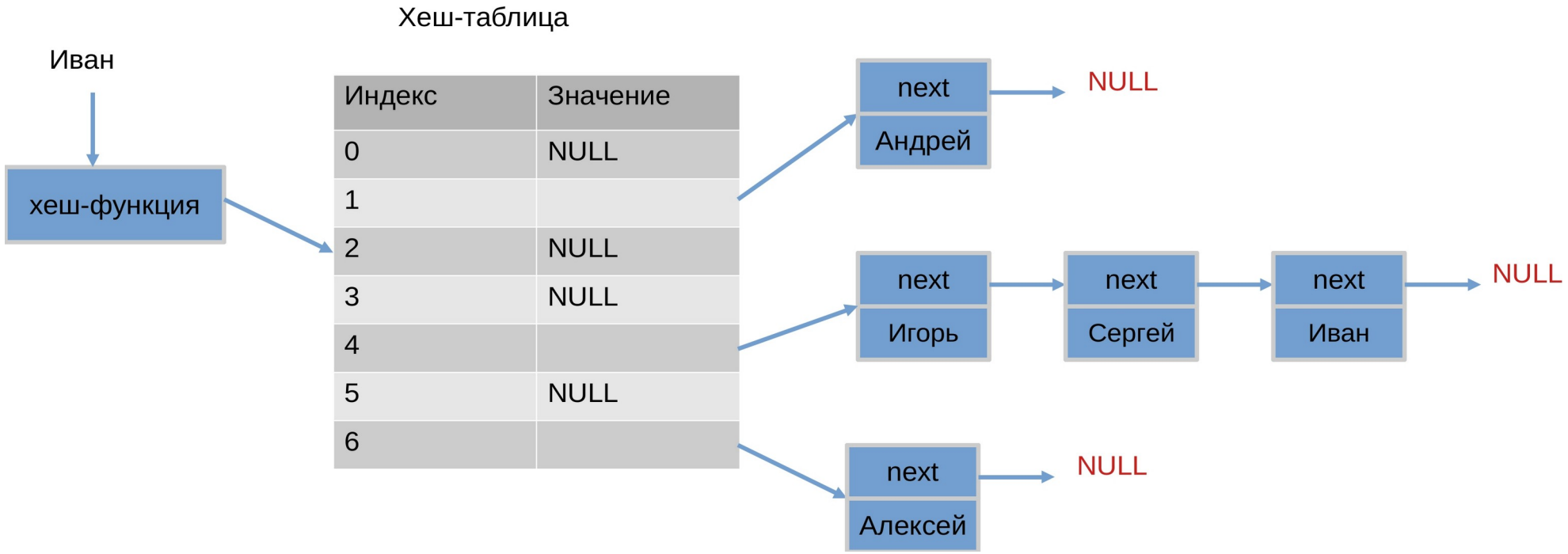
  Коллизия — ситуация, когда при разных входных данных хеш-функция генерирует один результат.

Хеш-функция. FNV-хеш

```
const unsigned FNV_32_PRIME = 0x01000193;

unsigned int FNV1Hash (char *in, size_t sz)
{
        unsigned int hval = 0x811c9dc5;
        while (sz--) {
                hval *= FNV_32_PRIME;
                hval ^= (unsigned int)*in++;
        }
        return hval;
}
```

Хеш-таблица


Хеш-таблица
Иван
хеш-функция
Индекс
Значение
0
NULL
1
2
NULL
3
NULL
4
5
NULL
6
next
Андрей
NULL
next
Игорь
next
Сергей
next
Иван
NULL
next
Алексей
NULL

hash_table_insert

```
void hash_table_insert(struct hash_table *table,
                struct item *item)
{
        int hash = calc_hash(item->key, strlen(item->key));
        int index = hash % table->size;
        struct hash_bucket *b = table->buckets[index];

        while (b != NULL) {
                if (strcmp(b->item->key, item->key) == 0) {
                        b->item = item; //replace
                        return;
                }
                b = b->next;
        }
        b = new_bucket();
        b->next = table->buckets[index];
        b->item = item;
        table->buckets[index] = b;
}
```

Очередь с приоритетом

Нужна, когда необходимо ввести дополнительный параметр, «вес» который влияет на очерёдность.

Поддерживаются методы

- insert() -- добавить элемент
- get_min() -- получить элемент с минимальным весом
- delete_min() -- удалить элемент с минимальным весом

Обычно реализуется с помощью двоичной кучи.

Двоичная (бинарная) куча — древовидная структура данных, для которой выполняется 2 условия:

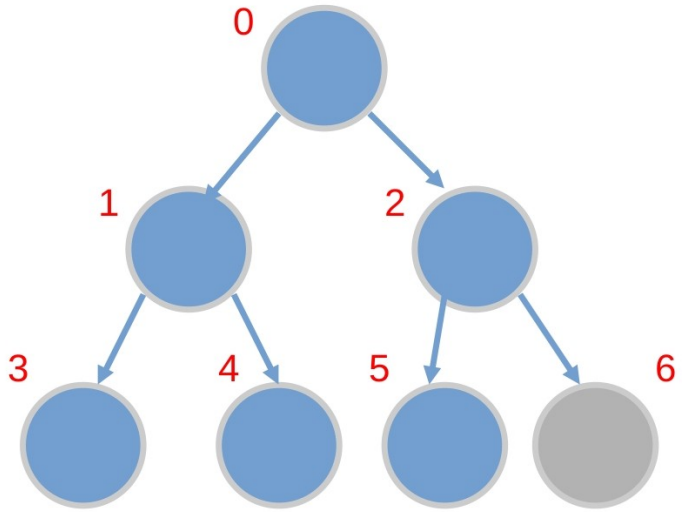
- Уровни кучи заполняются последовательно, слева направо.
- Каждое значение в вершине меньше чем дочерние

Bonus: https://www.youtube.com/watch?v=sv1eSC1puCg

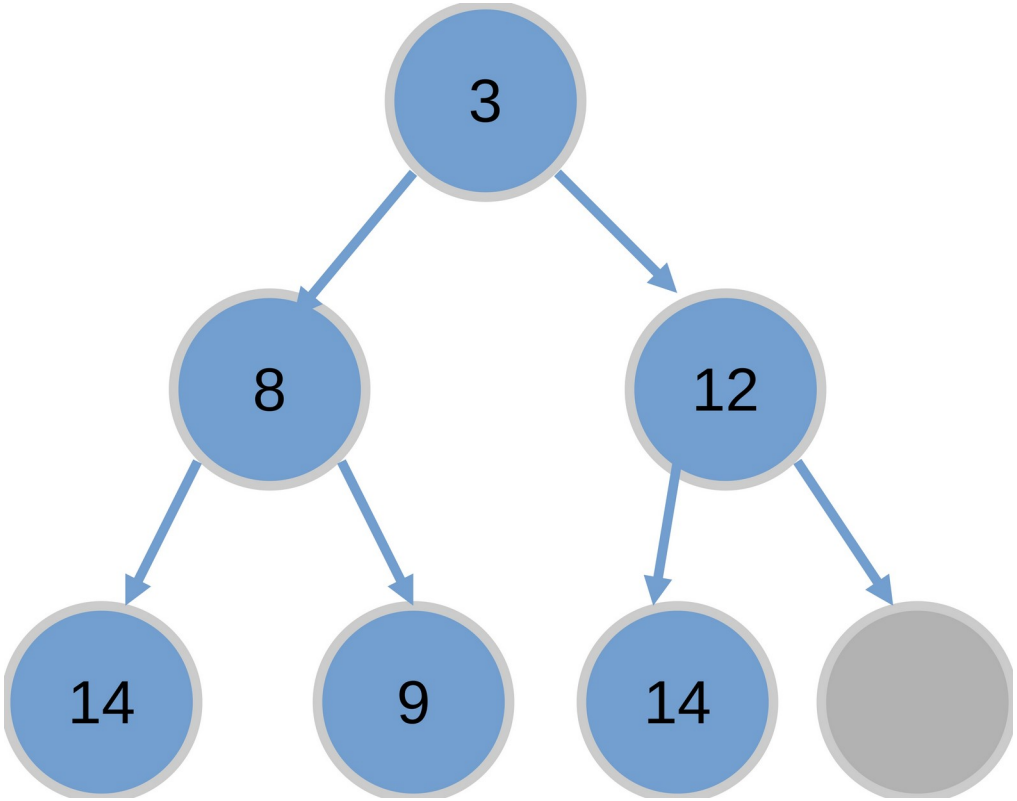
Квазиполное бинарное дерево

- Неполным может быть только нижний слой
- В неполном слое могут отсутствовать вершины справа
- Квазиполные бинарные деревья можно хранить в массивах!

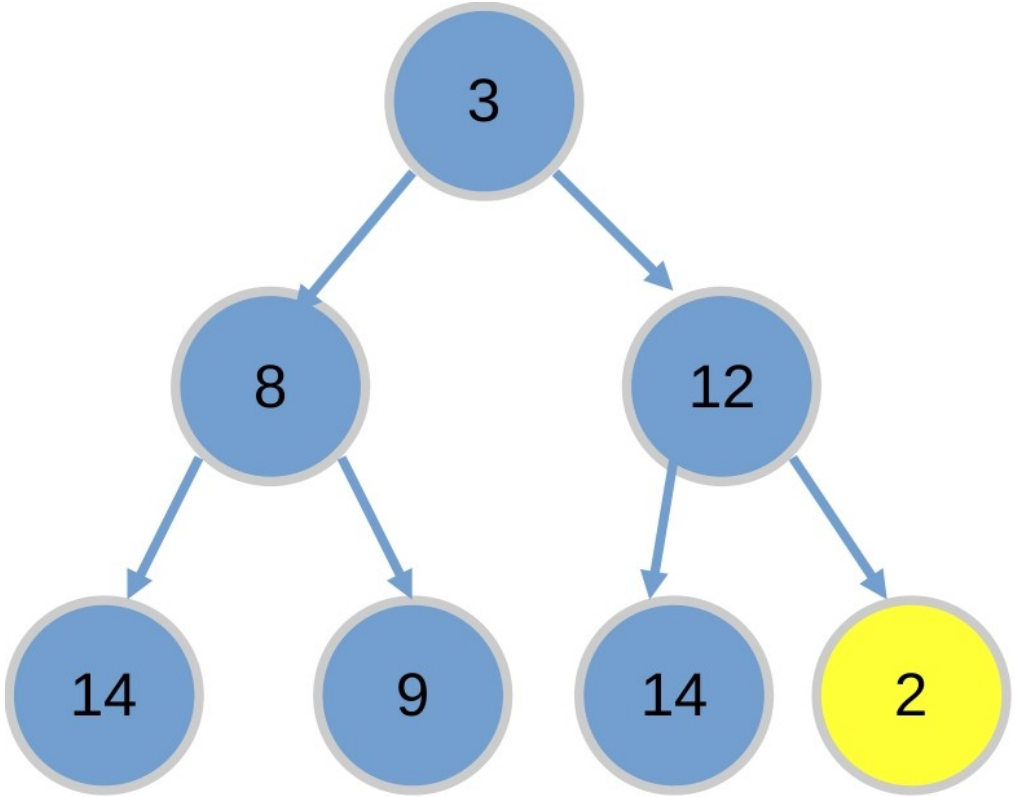
child_left = 2 * n + 1
child_right = 2 * n + 2
parent = (n - 1) / 2



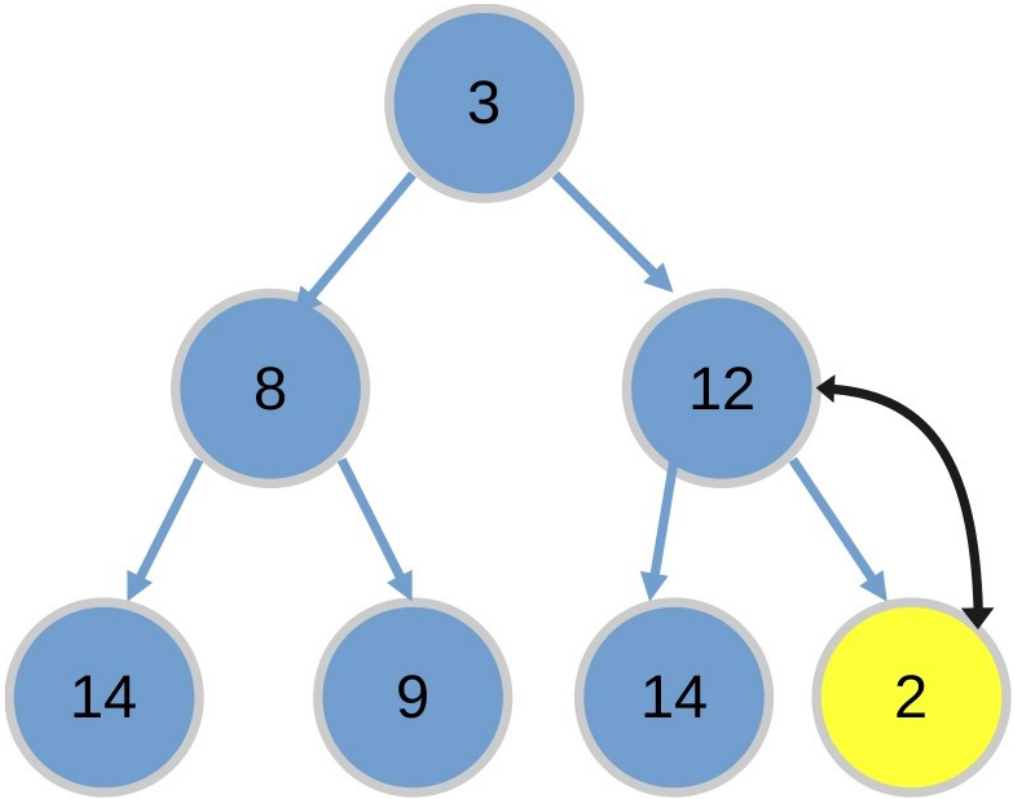
Двоичная куча



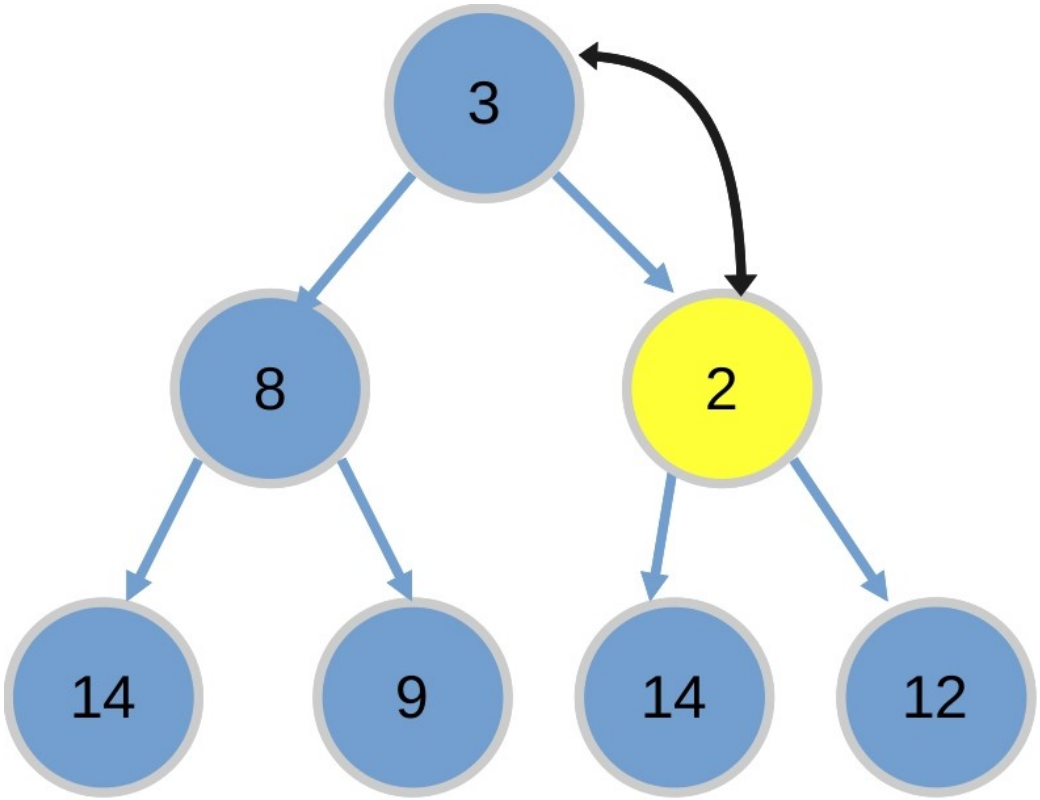
Добавление элемента (операция sift up)



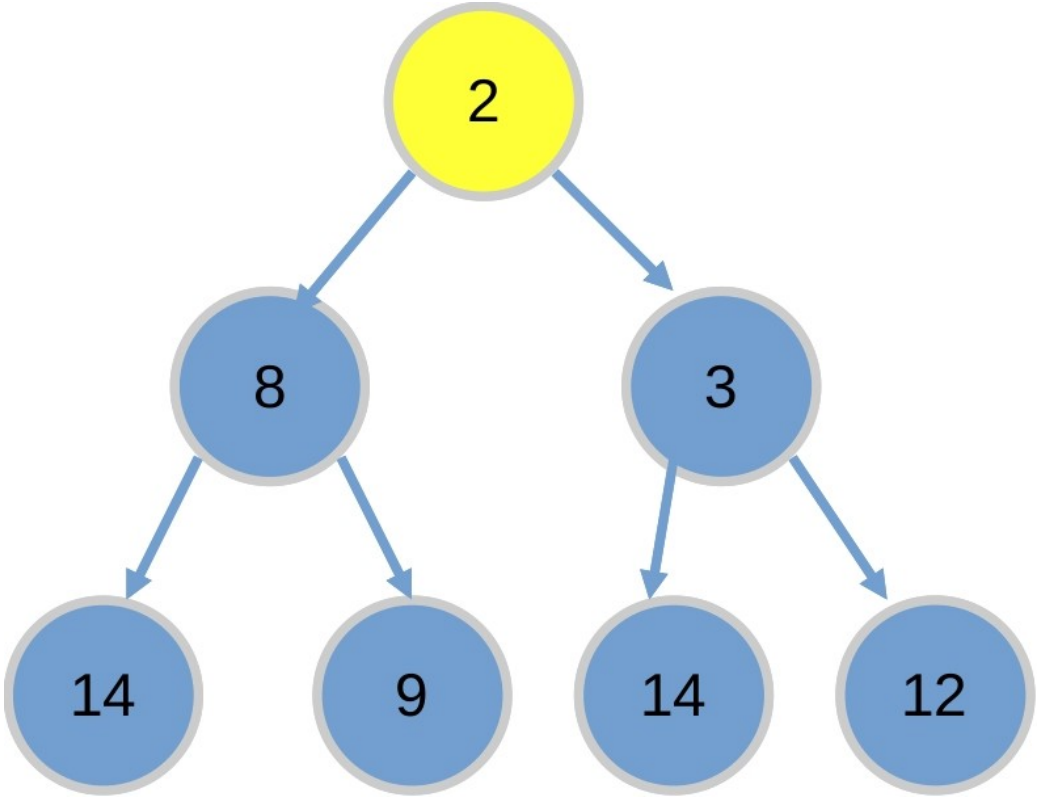
Добавление элемента (операция sift up)



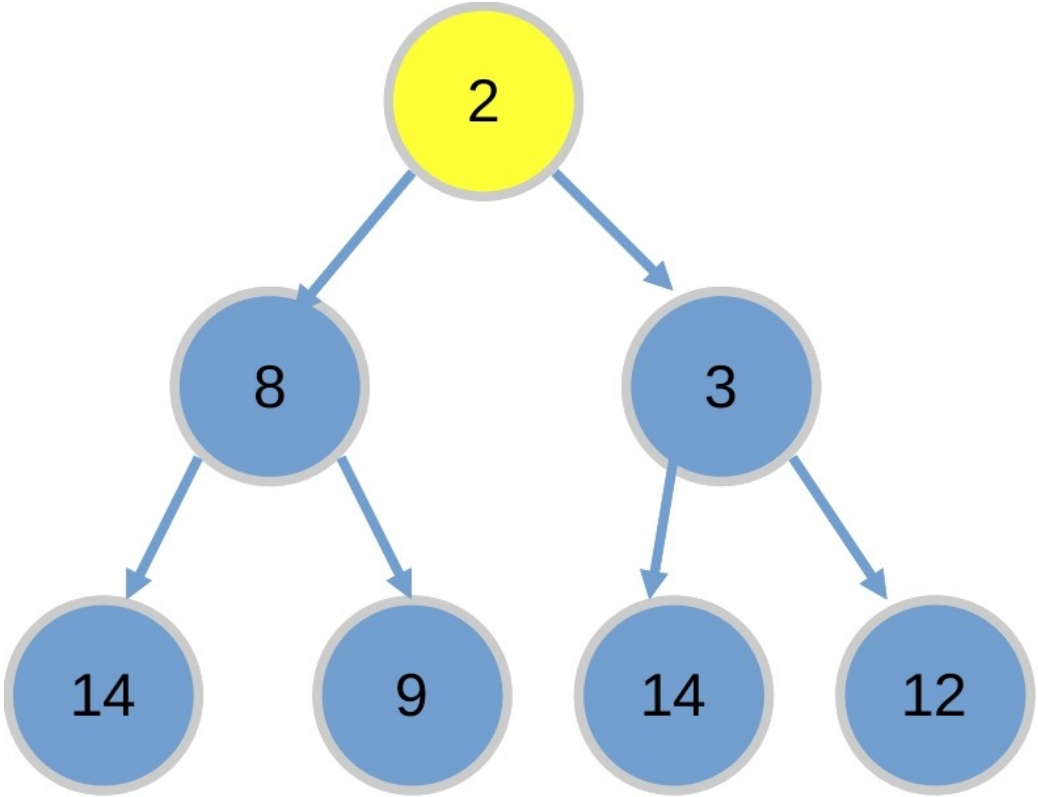
Добавление элемента (операция sift up)



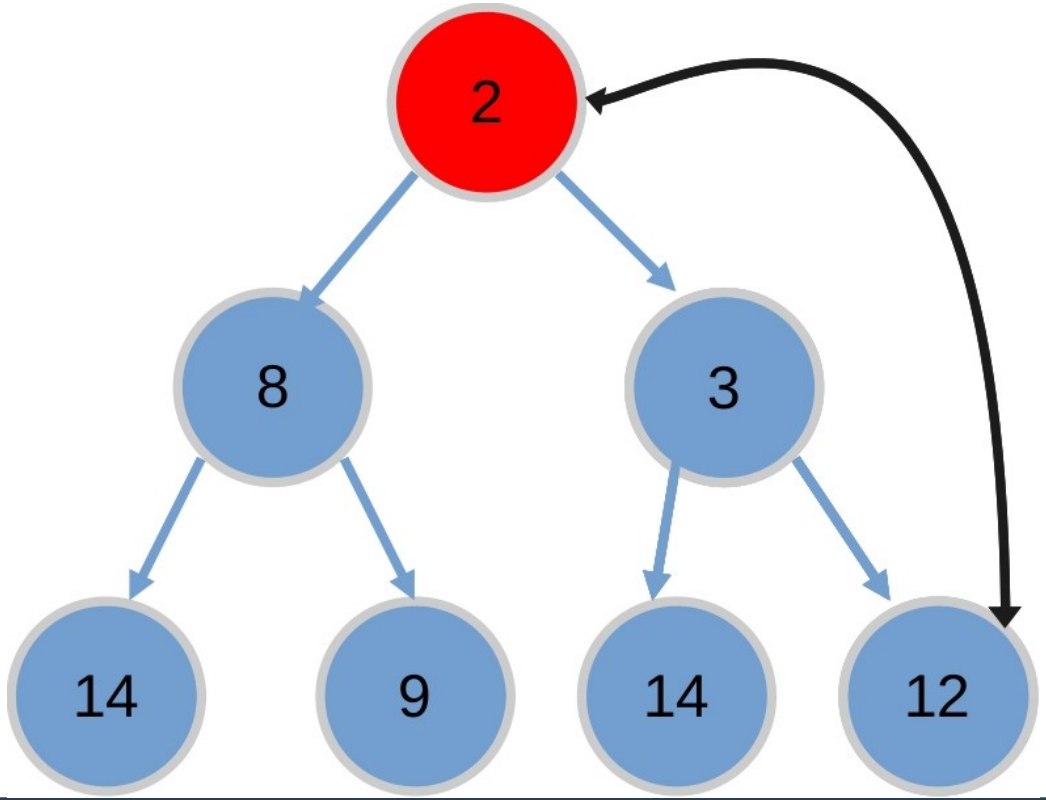
Добавление элемента (операция sift up)



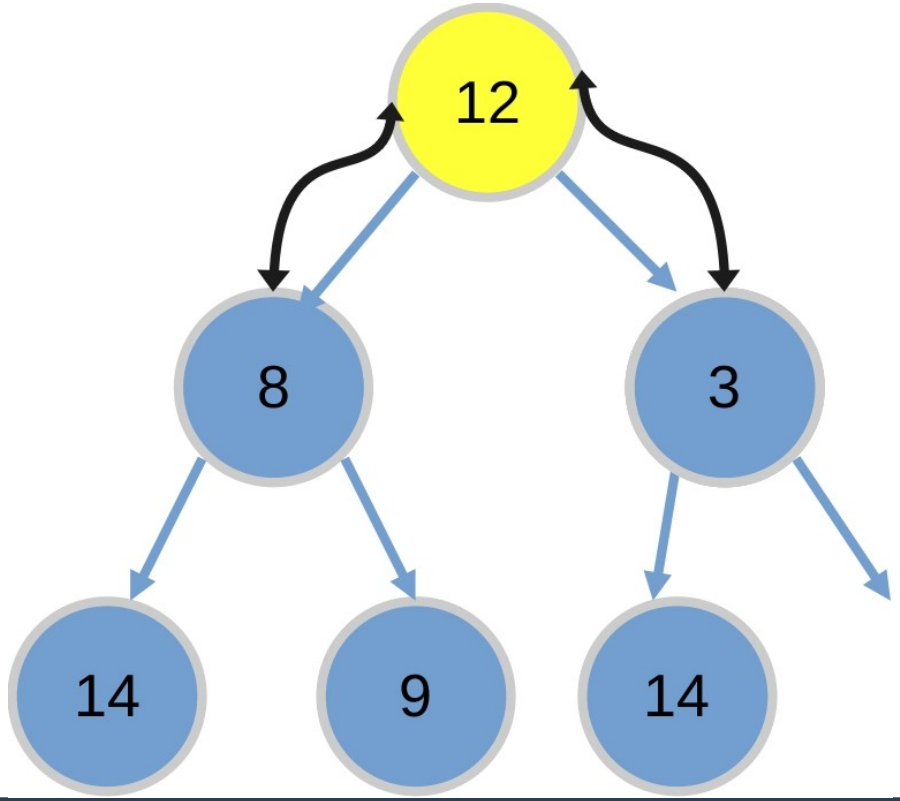
Удаление элемента (операция sift down)



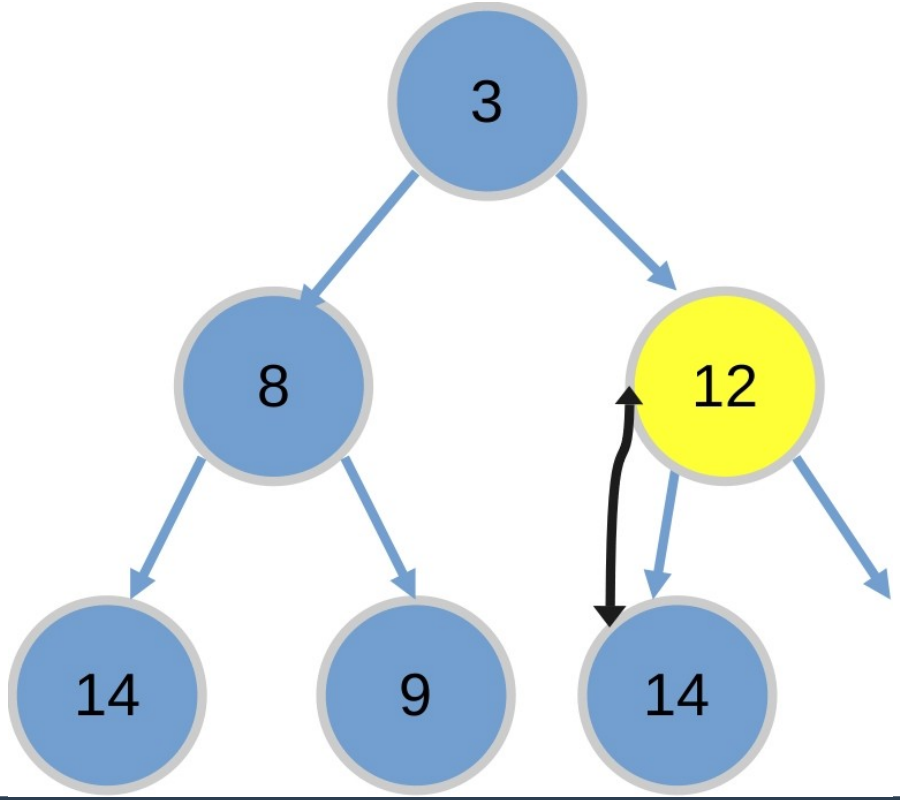
Удаление элемента (операция sift down)



Удаление элемента (операция sift down)



Удаление элемента (операция sift down)



Двоичная куча

Метод	Сложность
get_min(n)	O(1)
insert(n)	O(log(n))
delete_min(n)	O(log(n))

Реализации

Структура	C++ STL	Python
Вектор	std::vector	list
список	std::list	
стек	std::stack	list (*)
очередь	std::queue	collections.deque(*)
дек	std::deque	collections.deque
хеш-таблица	std::unordered_map	dict
Очередь с приоритетом	Std::priority_queue	heapq модуль

Дополнительные материалы

- Понятия и оценка алгоритмов (youtube лекция)
- Базовые структуры данных (youtube лекция)
- Исходники CTL(C template library)