

При удалении элемента его следует удалить со всех уровней, дополнительная балансировка не производится. Опишите список с пропусками как шаблон класса на языке C++.

```
1 template <typename T> class SkipList;
```

Класс должен удовлетворять следующим требованиям:

1. наличие конструктора по умолчанию для пустого списка без элементов;
2. наличие конструктора от диапазона итератора (output iterator), на любую другую структуру данных, например `std::unordered_set`;
3. наличие конструктора копирования и оператора копирующего присваивания;
4. наличие эффективных конструктора перемещения и перемещающего присваивания;
5. наличие метода **empty**, возвращающего **true**, если список пуст, иначе - **false**;
6. наличие метода **size**, возвращающего текущее количество элементов в списке за $O(1)$;
7. наличие метода **insert** для вставки нового значения с амортизированной сложностью $O(\log N)$;
8. наличие перегруженного метода **insert** для вставки диапазона итератора (output iterator);
9. наличие итератора для обхода списка в отсортированном порядке для использования в стандартных алгоритмах STL, итератор должен удовлетворять требованию **bidirectional iterator**;
10. наличие специального итератора для обхода списка в обратном направлении (reverse iterator);
11. наличие метода **find** для поиска положения ключа в списке: если ключ найден - вернуть итератор на значение, если ключ не найден - вернуть **past-the-end** итератор, равный вызову **end**;

12. наличие метода **count**, возвращающего количество вхождений заданного ключа;
13. наличие метода **lower_bound**, возвращающего итератор на первый элемент *не меньше заданного*;
14. наличие метода **upper_bound**, возвращающего итератор на первый элемент *строго больше заданного*;
15. наличие метода **clear** для удаления всех элементов списка;
16. наличие метода **erase** для удаления элемента из списка по итератору, при удалении полученные ранее итераторы на другие элементы списка должны оставаться верными;
17. наличие перегруженного метода **erase** для удаления промежутка итераторов, при удалении полученные ранее итераторы на другие элементы списка должны оставаться верными;
18. предусмотрите возможность создания списка с произвольным компаратором;
19. предусмотрите возможность хранения повторяющихся элементов в списке: метод **find** возвращает первый из одинаковых элементов и метод **count** возвращает количество вхождений повторяющегося элемента, при удалении одного из повторяющихся элементов другие остаются доступны;
20. предусмотрите метод **equal_range**, возвращающий промежуток итераторов на равные заданному ключу элементы.

Важно! Во всех случаях поведение методов при передаче неверных итераторов или неверных промежутков не определено!

Исследование списка с пропусками.

Опираясь на созданную реализацию списка с пропусками, проведите тестирование структуры данных и оцените асимптотическую сложность операций поиска, вставки и удаления. Как меняется время поиска в зависимости от p - вероятности создания копии элемента в списке более высокого уровня?