Лабораторная работа по теме «Стандартная библиотека шаблонов».

Реализация списка с пропусками.

Список с пропусками - это вероятностная динамическая структура данных, позволяющая выполнять операции добавления, удаления и поиска элемента с ожидаемой средней асимптотической сложностью $O(\log N)$, где N - количество элементов в списке. Технически список с пропусками представляет собой несколько отсортированных списков, каждый из которых содержит часть элементов и только один содержит все элементы. Схематически список с пропусками можно представить следующим образом:

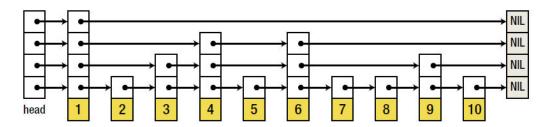


Рис. 1: Схематическое изображение списка с пропусками

Поиск элемента в такой структуре данных осуществляется сначала в самом «редком» списке простым последовательным просмотром узлов. Как только текущий узел оказывается больше ключа, поиск возвращается к предыдущему элементу и опускается на уровень ниже. Процедура повторяется пока не достигнут самый «подробный» список.

Идеально сбалансированный список с пропусками на каждом уровне содержит вдвое меньше элементов, что гарантирует асимптотическую сложность O(logN). Однако балансировка заняла бы слишком много времени. Вместо этого предлагается вероятностный механизм: при добавлении нового элемента случайным образом с вероятностью p выбирается один из двух вариантов ($\xi = \{1-p,p\}$): ничего не делать или добавить элемент на уровень выше. Процедура добавления продолжается до тех пор, пока случайная величина не выбросит «ничего не делать» или мы не добавим элемент на уровень K+1, где K - текущее количество уровней.

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

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 армортизационной 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?