4.27. Контейнеры map и set, их внутреннее устройство с алгоритмической точки зрения: что хранится внутри, какие алгоритмы и структуры данных используются для реализации методов. Понятие компараторов, пример использования нестандартных компараторов. Асимптотика обхода map итератором с объяснением, как он (обход) работает. Правила инвалидации итераторов в map.

Map - упорядоченный ассоциативный массив, который хранит пару: ключ-значение; в c++ это красно-черное дерево (сбалансированное двоичное дерево поиска). В нем тоже есть структура **Node**, которая хранит ключ-значение как пару, указатель на родителя, указатели на двух детей, bool red. Итератор в map - указатель на **Node**. Под итератором лежит пара ключ-значение.

Еще есть **компаратор** (для ключей) - функция, которой на вход можно передать два элемента для сравнения, а на выходе получить true или false в зависимости от результатов сравнения; по стандарту предполагается, что компаратор задаёт weak ordering на множестве (транзитивное бинарное отношение, которое для любых двух элементов либо xRy, либо yRx). Пример нестандартного компаратора - сравнение пар элементов.

Ключ обязан иметь **константный тип** (const key), так как от ключа зависит положение в дереве.

Если в insert положить пару с неконстантным key, то произойдёт const_cast; если положить такой элемент, что key уже существует, то bool в паре будет false. Если по [key] не существует такого элемента, то он создастся по умолчанию, а если через at - то бросает исключение.

От map нельзя вызывать std::sort, next_permutation - CE.

Структура	Параметры	Описание
map	<Key, Value, Comp = std::less<Key> >	Красно-чёрное дерево
unordered_mop	<Key, Value, Hash = std::hash<Key>, EqualTo = std::equal_to <Key> >	Хеш-таблица, массив связных списков
set	<Key, Compare = std::less<Key> >	Красно-чёрное дерево

Name	iterator find	pair<iterator, bool> insert
map	(const Key&), O(log N)	(const pair<const Key, Value>&), O(log N)
set	O(log N)	O(log N)

Name	bool erase	Value& operator[]	Value& op-r at
map	(const Key&), O(log N)	(const Key&), O(log N)	(const Key&), O(log N)
set	O(log N)	O(log N)	O(log N)

Обход map итератором:

Строго говоря, инкремент в итераторе map работает за логарифм (как и find по ключу), но в итоге обход мап итератором (см. код) линеен:

```
for (auto it = m.begin(); it != m.end(); ++it)
  cout << it->first << it->second;
```

Он линеен, так как в итоге каждую вершину дерева мы посещаем не более шести раз: в неё входит одно ребро (от родителя), выходят два ребра (к детям), значит, по ней при обходе можно будет пройтись не более шести раз по этим рёбрам.

Пример оптимального использования итераторов в map:

```
// Неоптимально:

if (m.count(5))
  m[5] = 3;

// Неоптимально, так как мы два раза спустились по дереву
// Оптимально:

auto x = m.find(5);
if (x != m.end())
  x->second = 3;
```

Инвалидация итераторов в map:

Метод	Инвалидация
All read only operations, swap, std::swap	Никогда
clear, rehash, reserve, operator=	Всегда
insert, emplace, emplace_hint, operator[]	Только если был rehash
erase	Только относительно удалённого элемента