std::map

Контейнер **std::map** в C++ представляет собой реализацию ассоциативного массива, который представлен в виде отображения ключей на значения. Ключи могут быть любого типа, для которого определен оператор сравнения, а значения могут быть любого типа.

std::map реализован как бинарное дерево поиска, где каждый узел содержит пару ключ-значение. Ключи в дереве отсортированы в порядке возрастания и используются для быстрого поиска значения. Поиск, вставка и удаление элементов в **std::map** имеют логарифмическую сложность, что делает его эффективным для большинства задач.

Операции, которые можно выполнить над **std::map**, включают:

- **insert(key, value)** - добавляет элемент в отображение.
- **erase(key)** - удаляет элемент из отображения по ключу.
- **find(key)** - возвращает итератор на элемент с указанным ключом.
- **clear()** - удаляет все элементы из отображения.
- **size()** - возвращает количество элементов в отображении.

Кроме того, **std::map** предоставляет методы для обхода элементов отображения в отсортированном порядке, такие как **begin()**, **end()**, **rbegin()**, **rend()**.

Стандарт C++ также предоставляет другие ассоциативные контейнеры, такие как **std::multimap**, **std::set** и **std::multiset**, которые основаны на том же принципе бинарного дерева поиска, что и **std::map**. Однако, они используются для решения других задач и имеют свои особенности.

Бинарное дерево – это структура данных, в которой каждый узел имеет не более двух потомков: левого и правого. Каждый потомок сам является корнем бинарного дерева, которое в свою очередь также может иметь не более двух потомков и т.д.

Каждый узел бинарного дерева содержит значение (ключ) и может иметь до двух потомков. В зависимости от правил построения дерева, ключи могут быть упорядочены, например, в порядке возрастания или убывания.

Одним из примеров использования бинарного дерева является бинарное дерево поиска, в котором ключи упорядочены таким образом, что для каждого узла все значения в левом поддереве меньше, чем значение ключа узла, а все значения в правом поддереве больше, чем значение ключа узла. Бинарное дерево поиска может использоваться для реализации структур данных, таких

как словари или множества, где ключи используются для быстрого доступа к соответствующим значениям.

В бинарном дереве есть несколько основных операций, включая добавление нового узла, удаление узла, поиск узла по ключу, обход дерева и т.д.

Пример кода бинарного дерева, представлен в файле ***BinTree.cpp***. Этот код определяет структуру узла бинарного дерева, а также функции для вставки, поиска и обхода дерева в порядке возрастания значений (inorder traversal). В **main** функции создается дерево и выполняются операции поиска и обхода.

Красно-черное дерево поиска (Red-Black Tree) — это сбалансированное двоичное дерево поиска, в котором каждый узел имеет один из двух цветов: красный или черный.

Основные свойства красно-черных деревьев:

- Каждый узел является либо красным, либо черным.
- Корень и листья (NIL) дерева являются черными.
- Если узел красный, то его дочерние узлы обязательно черные.
- Для каждого узла все пути от него до листьев (NIL) должны содержать одинаковое количество черных узлов.

Каждый узел содержит ключ и ссылки на двух дочерних узла и родительский узел. Хранение дополнительной информации в узлах не предусмотрено.

Операции:

- Вставка узла
- Удаление узла
- Поиск узла
- Обход дерева (inorder, preorder, postorder)

Вставка: При вставке нового узла он сначала вставляется в дерево как в обычном бинарном дереве поиска. Затем новый узел окрашивается в красный цвет и выполняется процедура перекрашивания и поворотов, чтобы сохранить свойства красно-черного дерева.

Удаление: Удаление узла из красно-черного дерева также начинается с удаления его из обычного двоичного дерева поиска. Затем происходит перебалансировка дерева, чтобы сохранить свойства красно-черного дерева.

Поиск: Поиск в красно-черном дереве происходит так же, как и в обычном двоичном дереве поиска: начиная с корня, сравниваются значения ключей в узлах, и если ключ равен искомому, возвращается ссылка на узел. Если значение искомого ключа меньше значения ключа текущего узла, то поиск продолжается в левом поддереве, иначе в правом.

Обход дерева: Обходы дерева (inorder, preorder, postorder) также происходят аналогично обычному двоичному дереву поиска. Однако, так как красно-черное дерево сбалансировано, обходы выполняются за логарифмическое время в зависимости от размера дерева.

Пример реализации красно-черного дерева поиска *RedBlackTree.cpp*