***Теория по Бинарному Дереву:***

**Дерево (свободное) – непустая коллекция вершин и ребер, удовлетворяющих определяющему свойству дерева.**

**Определяющее свойство дерева – существование только одного пути, соединяющего два узла.**

**Дерево с корнем** – дерево, в котором один узел выделен и назначен корнем дерева. Существует только один путь между корнем и каждым из других узлов дерева.

**Высота** (глубина) дерева с корнем – количество вершин в самом длинном пути от корня

Каждый узел (за исключением корня) имеет только один узел, расположенный над ним. Такой узел называется **родительским**. Узлы, расположенные непосредственно под данным узлом, называются его **дочерними узлами**. Узлы, не имеющие *дочерних узлов*, называются **листьями**.

**Двоичное (бинарное) дерево** – это дерево, в котором *степени вершин не превосходят 3*. **Двоичное (бинарное) дерево с корне**м – это дерево, в котором каждая вершина имеет не более двух дочерних вершин.

**СД «Двоичное дерево»** - представление двоичного дерева с корнем. Узел – структура, содержащая данные и указатели на левый и правый дочерний узел. Также может содержать указатель на родительский узел.

**Двоичное дерево поиска** – это двоичное дерево, с каждым узлом которого связан ключ, и выполняется следующее дополнительное условие: ключ в любом узле **X** больше или равен ключам во всех узлах левого поддерева **X** и меньше или равен ключам во всех узлах правого поддерева **X**.

***Функции по Бинарному Дереву:***

**Инициализация узла: (Создано из понимания конструктора)**

Выделить новую память для узла, в поле Ключ внести значение, в поля адресов на Левое и Правое поддерево и на Родителя внести nullptr.

Вернуть адрес нового узла.

**Создание дерева: (Создано из понимания работа с СД «СПИСОК»)**

Присвоить адресу корня дерева nullptr, переменной, отвечающей за проверку создания дерева, присвоить утвердительное значение.

**Поиск по ключу: (Было в Лекции – «Find»)**

Проверить есть ли узел с ключом K в дереве с корнем X, и если да, то вернуть указатель на этот узел

Если дерево пусто сообщить, что узел не найден и остановиться, иначе:

• Если К=Х, вернуть указатель на этот узел и остановиться

• Если К>Х, искать ключ К в правом поддереве

• Если К<Х, искать ключ К в левом поддереве

*Время работы: О(h), где h – глубина дерева*

**Поиск минимального ключа: (Было в Лекции – «FindMin»)**

Найти узел с минимальным значением ключа K в дереве с корнем **X**

Переходить в левый дочерний узел, пока такой существует

*Время работы: О(h), где h – глубина дерева*

**Поиск максимального ключа: (Сделано по аналогии с минимумом)**

Найти узел с максимальным значением ключа K в дереве с корнем **X**

Переходить в правый дочерний узел, пока такой существует

*Время работы: О(h), где h – глубина дерева*

**Добавление узла: (Было в Лекции – «Insert» + некоторые доработки, основа взята из лекции)**

Вставить узел с ключом K в дерево с корнем X (возможно появление дубликатов)

Если дерево пусто, заменить его на дерево с одним корневым узлом и остановиться, иначе сравнить **К** с **ключом корневого узла Х**. Если К < Х, рекурсивно добавить К в левое поддерево Х, иначе рекурсивно вставить в правое поддерево Х.

*Время работы: О(h), где h – глубина дерева*

**Удаление узла: (Сделано по описанию с лекции)**

Удалить узел с ключом K из дерева с корнем X

Если дерево пусто, остановиться, иначе сравнить К с ключом корневого узла Х:

• Если К < Х, рекурсивно удалить К из левого поддерева Х

• Если К > Х, рекурсивно удалить К из правого поддерева Х

• Если К == Х, то:

• Дочерних узлов нет – удаляем узел Х

• Один дочерний узел – переносим дочерний узел в Х

• Оба дочерних узла есть – Заменяем ключ удаляемого узла на ключ минимального узла из правого поддерева, удаляя последний.

*Время работы: О(h), где h – глубина дерева*

**Удаление дерева: (Было в Лекции – «DeepFirstSearch» + некоторые доработки, основа взята из лекции)**

Если узла нет, остановиться, иначе:

• Рекурсивно пройти в глубь левого поддерева, потом рекурсивно пройти в глубь правого поддерева

• Удалить узел

***Общая характеристика Бинарного Дерева:***

|  |  |  |
| --- | --- | --- |
|  | **В среднем** | **В худшем случае** |
| **Расход памяти** | O(n) | O(n) |
| **Поиск** | O(log n) | O(n) |
| **Вставка** | O(log n) | O(n) |
| **Удаление** | O(log n) | O(n) |

***Теория по Декартовому Дереву:***

**Декартово дерево (eng. Treap)** – структура данных, объединяющая в себе двоичное дерево поиска и двоичную кучу (tree+heap = treap). **Декартово дерево** – двоичное дерево, в узлах которого хранятся пары, где x – ключ, а y – это приоритет. *Все х и все у являются различными.* Если некоторый элемент дерева содержит (х0, у0), то у всех элементов в левом поддереве х < x0, а также в левом и правом поддереве y < y0.

**В компьютерных науках куча (heap)** — это специализированная структура данных типа дерево, которая удовлетворяет свойству кучи: если B является узлом-потомком узла A, то ключ(A) ≥ ключ(B).

Приоритеты могут повторяться и назначаются **случайным образом** (лучше по равномерному распределению).

В декартовом дереве из n узлов, приоритеты которого являются случайными величинами с равномерным распределением, **средняя глубина дерева O(log n)**. Операции:

• Разрезание – Split

• Слияние – Merge

**Разрезание – Split.** Операция разрезания позволяет разрезать декартово дерево Т по ключу К и получить 2 других дерева: Т1 и Т2, причем в Т1 находятся все ключи дерева Т, не большие К, а в Т2 – большие К.

*Время работы: О(h), где h – глубина дерева*

**Слияние – Merge**. Операция слияния позволяет слить 2 декартовых дерева в одно. *Причем все ключи в левом дереве должны быть меньше, чем ключи в правом.* В результате получается дерево, в котором есть все ключи из первого и второго деревьев.

*Время работы: О(h), где h – глубина дерева*

***Функции по Декартовому дереву:***

**Инициализация узла: (Создано из понимания конструктора)**

Выделить новую память для узла, во все поля внести подаваемые значения при вызове функции.

Вернуть адрес нового узла.

**Создание дерева: (Создано из понимания работа с СД «СПИСОК»)**

Присвоить адресу корня дерева nullptr, переменной, отвечающей за проверку создания дерева, присвоить утвердительное значение.

**Поиск по ключу: (Было в Лекции – «Find»)**

Проверить есть ли узел с ключом K в дереве с корнем X, и если да, то вернуть указатель на этот узел

Если дерево пусто сообщить, что узел не найден и остановиться, иначе:

• Если К=Х, вернуть указатель на этот узел и остановиться

• Если К>Х, искать ключ К в правом поддереве

• Если К<Х, искать ключ К в левом поддереве

*Время работы: О(h), где h – глубина дерева*

**Вставка узла:**

Добавляется элемент <x,y>, где x – ключ, а y – это приоритет.

Элемент <x,y> - дерево из одного элемента. Для того, чтобы добавить его в декартово дерево Т, их нужно слить. Но Т может содержать ключи как меньше, так и больше ключа х, поэтому сначала нужно разрезать Т по ключу х.

***Реализация Неоптимизированная (1): (1 Split, 2 Merge)*** – (**Функция взята из интернета по рекомендации в конце лекции + свои доработки на основе из лекции):**

• Разрезаем Т по ключу х на L и R

• Сливаем первое дерево L с новым элементом.

• Сливаем получившееся дерево с деревом R.

***Реализация Оптимизированная (2): (1 Split => Merge вообще не используется)*** –**(Сделано по описанию с лекции)**

• Сначала спускаемся по дереву (как в обычном бинарном дереве поиска по х), но останавливаемся на первом элементе, в котором значение приоритета оказалось меньше у.

• Разрезаем поддерево данного элемента на L и R

• Полученные деревья записываем в качестве левого и правого сына добавляемого элемента

• Полученное дерево вставляем на место элемента, найденного в первом пункте.

**Удаление узла:**

Удаляется элемент с ключом х

***Реализация Неоптимизированная (1): (2 Split, 1 Merge)*** – (**Функция взята из интернета по рекомендации в конце лекции + свои доработки на основе из лекции):**

• Разобьем дерево по ключу х на L и R

• Теперь отделяем от первого дерева L элемент х разбивая по ключу:

(х-е) *(В нашем случае (x-1))*

• Сливаем измененное дерево L со вторым R

***Реализация Оптимизированная (2): (1 Merge => Split не используется)*** –**(Сделано по описанию с лекции)**

• Спускаемся по дереву (как в обычном бинарном дереве поиска по х), ища удаляемый элемент

• Вызываем слияние его левого и правого сыновей

• Результат ставим на место удаляемого элемента

**Удаление дерева: (Было в Лекции – «DeepFirstSearch» + некоторые доработки, основа взята из лекции)**

Если узла нет, остановиться, иначе:

• Рекурсивно пройти в глубь левого поддерева, потом рекурсивно пройти в глубь правого поддерева

• Удалить узел

***Общая характеристика Декартового Дерева:***

