|  |  |
| --- | --- |
| ДИСЦИПЛИНА | Алгоритмы и структуры данных |
| ИНСТИТУТ | Институт перспективных технологий и индустриального программирования |
| КАФЕДРА | Кафедра индустриального программирования |
| ВИД УЧЕБНОГО МАТЕРИАЛА | Текущий контроль |
| ПРЕПОДАВАТЕЛЬ | Преснецова Виктория Юрьевна, Яковлев Михаил Сергеевич, Дворецкий Артур Геннадьевич, Гиматдинов Дамир Маратович |
| СЕМЕСТР | 2 семестр, 2024-2025 гг. |

**Рабочая тетрадь 2.**

**Сбалансированные двоичные деревья поиска**

|  |
| --- |
| **Требования** |
| 1. Язык программирования - C++. 2. Код должен быть оптимизирован для производительности и использования ресурсов. 3. Необходимо избегать избыточных вычислений и памяти. 4. Комментарии должны объяснять сложные участки кода и логику работы программы. |

**1. АВЛ-деревья**

|  |
| --- |
| **Теоретический материал** |
| **Подробные материалы о всех видах сбалансированных деревьев и операциях с ними (добавление и удаление элементов) смотрите в слайдах лекции 2**  **АВЛ-деревья**  АВЛ-дерево - сбалансированное по высоте двоичное дерево поиска: для каждой его вершины высота её двух поддеревьев различается не более чем на 1. Для АВЛ-деревьев **сбалансированность определяется разностью высот правого и левого поддеревьев любого узла**. Если эта разность по модулю не превышает 1, то дерево считается сбалансированным. Данное условие проверяется после каждого добавления или удаления узла, и определен минимальный набор операций перестройки дерева, который приводит к восстановлению свойства сбалансированности, если оно оказалось нарушено.  **Для каждого узла дерева в структуру вводится переменная – показатель баланса (разность высот правого и левого поддеревьев)**  Структура данных, описывающая узел АВЛ-дерева, имеет вид:  Struct Node{  int key;  Node \* left;  Node \* right;  Node \* parent;  int balans;  };  Для упрощения работы алгоритмов в структуру допускается ввести указатель на родительский узел.  Алгоритм добавления узла:  1. Выполнить обычную вставку, как в бинарном дереве поиска.  2. Обновить высоту текущего узла.  3. Вычислить баланс-фактор.  4. Если баланс-фактор выходит за пределы [-1, 1], выполнить поворот.  Picture background **Отличия АВЛ-деревьев от бинарных деревьев поиска**  |  |  |  | | --- | --- | --- | | **Критерий** | **АВЛ-дерево (AVL)** | **Бинарное дерево поиска (BST)** | | *Балансировка* | Поддерживает баланс автоматически после каждой вставки или удаления. | Может быть несбалансированным, если элементы добавляются в порядке возрастания или убывания. | | *Баланс-фактор* | Разница высот левого и правого поддерева всегда -1, 0 или 1. | Баланс не контролируется. Разница может быть любой. | | *Операции вставки/удаления* | Требует дополнительного времени на балансировку (вращения). | Вставка и удаление проще, но может привести к сильному дисбалансу. | | *Поиск* | O(log n) в худшем случае. | O(n) в худшем случае (при сильном дисбалансе). | | *Повороты* | Использует левый, правый и двойные повороты для балансировки. | Не требует поворотов. | | *Структура* | Гарантирует, что дерево остается сбалансированным. | Может превращаться в список при неудачном порядке вставки. | | Э*ффективность* | Быстрее на больших объемах данных за счет сбалансированной структуры. | Может деградировать до неэффективного случая O(n) при несбалансированности. | |
|  |

|  |  |
| --- | --- |
| **Пример 1** | |
| ***Задача:*** | |
|  | Реализовать АВЛ-дерево с поддержкой симметричного обхода. |
| ***Решение:***  Баланс всех узлов был установлен в 0 просто как заглушка, поскольку балансировка не выполняется. | |
|  |  |
| ***Ответ:*** | |
|  |  |

|  |  |
| --- | --- |
| **Задание 1 (1 балл)** | |
| ***Задача:*** | |
|  | Создать двоичное дерево поиска (в узлах хранятся целые положительные числа). Рассчитать для каждого узла построенного дерева показатель баланса |
| ***Решение:*** | |
|  |  |
| ***Ответ:*** | |
|  |  |
| **Задание 2 (2 балла)** | |
| ***Задача:*** | |
|  | Создать АВЛ-дерево (узлы добавляются как при стандартном алгоритме вставки узла, при нарушении баланса производится необходимая перестройка). |
| ***Решение:*** | |
|  |  |
| ***Ответ:*** | |
|  |  |
| **Задание 3\*** | |
| ***Задача:*** | |
|  | Для АВЛ-дерева (построенного в задании 2) реализовать операции удаления узла дерева с учетом требования сохранения сбалансированности дерева после удаления узла. |
| ***Решение:*** | |
|  |  |
| ***Ответ:*** | |
|  |  |

**2. Красно-черные деревья**

|  |
| --- |
| **Теоретический материал** |
| Красно-черное дерево(англ. red-black tree, RB tree) — один из видов самобалансирующихся двоичных деревьев поиска, гарантирующих логарифмический рост высоты дерева от числа узлов и позволяющее быстро выполнять основные операции дерева поиска: добавление, удаление и поиск узла. Сбалансированность достигается за счёт введения дополнительного атрибута узла дерева — «цвета». Этот атрибут может принимать одно из двух возможных значений — «чёрный» или «красный».  ***Свойства КЧ-деревьев:***  1. каждый узел либо красный, либо черный;  2. каждый лист (фиктивный) – черный;  3. если узел красный, то оба его сына – черные;  4. все пути, идущие от корня к любому фиктивному  листу, содержат одинаковое количество черных узлов;  5. корень – черный.    Структура данных, описывающая узел красно-черного дерева, имеет вид:  Struct Node{  int key;  Node \* left;  Node \* right;  Node \* parent;  char color;  };  Для упрощения работы алгоритмов в структуру допускается ввести указатель на родительский узел. |

|  |  |
| --- | --- |
| **Задание 4 (2 балла)** | |
| ***Задача:*** | |
|  | Создать двоичное дерево поиска (в узлах хранятся целые положительные числа). Проверить для построенного дерева свойства красно-черного дерева. |
| ***Решение:*** | |
|  |  |
| ***Ответ:*** | |
|  |  |
| **Задание 5\*** | |
| ***Задача:*** | |
|  | Создать красно-черное дерево (узлы добавляются как при стандартном алгоритме вставки узла, при нарушении свойств красно-черного дерева производится необходимая перестройка и перекрашивание). Реализовать операцию удаления узла из красно-черного дерева. |
| ***Решение:*** | |
|  |  |
| ***Ответ:*** | |
|  |  |

**3. Самоперестраивающиеся деревья (splay деревья)**

|  |
| --- |
| **Теоретический материал** |
| Самоперестраивающееся дерево является двоичным деревом поиска, в котором поддерживается свойство сбалансированности. Это дерево принадлежит классу «саморегулирующихся деревьев», которые поддерживают необходимый баланс ветвления дерева, чтобы обеспечить выполнение операций поиска, добавления и удаления за логарифмическое время от числа хранимых элементов. Это реализуется без использования каких-либо дополнительных полей в узлах дерева (как, например, в Красно-чёрных деревьях или АВЛ-деревьях, где в вершинах хранится, соответственно, цвет вершины и глубина поддерева). Вместо этого «расширяющие операции» (splay operation), частью которых являются вращения, выполняются при каждом обращении к дереву.  Операция splay(T,k) выполняется следующим образом. Сначала производится поиск узла с ключом k в дереве обычным способом, спускаясь вниз, начиная с корня. При этом запоминается пройденный путь. В итоге, получаем указатель на узел дерева либо с ключом k, либо с его предшественником или последователем, на котором закончился поиск. Далее, происходит возвращение назад по запомненному пути, с перемещением этого узла к корню. Для того, чтобы при этом сохранялись свойства двоичного дерева поиска, необходимы повороты. |

|  |  |
| --- | --- |
| **Задание 6\*** | |
| ***Задача:*** | |
|  | Создать самоперестраивающееся дерево, для которого добавление нового узла и удаление существующего узла будет производиться на основе операции splay. |
| ***Решение:*** | |
|  |  |
| ***Ответ:*** | |
|  |  |