**Лабораторная работа 3.** **Балансировка двоичного дерева поиска**

**Цель работы: Закрепление теоретических знаний и получение практических навыков создания программ для балансировки бинарных деревьев.**

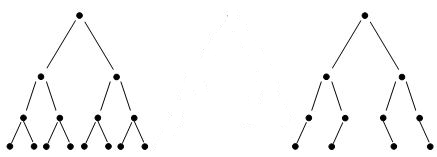
**Краткие теоретические сведения**

1. **Идеально сбалансированные бинарные деревья**

    Пусть требуется построить бинарное дерево с **n** узлами и минимальной высотой (максимально "ветвистое" и "низкое"). Такие деревья имеют большое практическое значение, так как их использование сокращает машинное время, требуемое на выполнение различных алгоритмов.

***Определение.***

Бинарное дерево называется ***идеально сбалансированным***, если для ***каждой его вершины высота левого и правого поддеревьев одинаковы***.

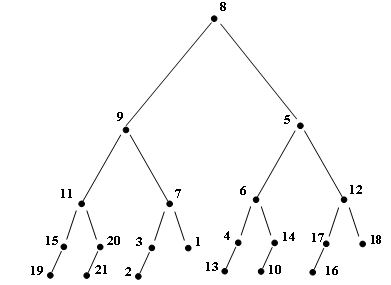
  
Рис.1. Примеры идеально сбалансированных бинарных деревьев

В сбалансированном бинарном дереве поиска сложность поиска O(log n).

При построении идеально сбалансированного дерева необходимо учитывать, что для достижения минимальной высоты при заданном числе вершин, нужно располагать максимально возможное число вершин на всех уровнях, кроме самого нижнего. Это можно сделать, если распределять все поступающие в дерево вершины поровну слева и справа от каждой вершины.

    Суть алгоритма

* взять одну вершину в качестве корня.
* построить левое поддерево с **nl = n DIV 2** вершинами тем же способом.
* построить правое поддерево с **nr = n-nl-1** вершинами тем же способом.
* Предположим, например, что имеется следующий набор ключей для построения дерева с 21 вершиной: 8, 9, 11, 15, 19, 20, 21, 7, 3, 2, 1, 5, 6, 4, 13, 14, 10, 12, 17, 16, 18. Результат работы программы следующий:

  
Рис.2. Результат построения идеально сбалансированного дерева

* *НО при формировании идеально сбалансированного дерева по вышеприведенному алгоритму оно может не быть бинарным деревом поиска.*

1. **Балансированные по высоте деревья (АВЛ-деревья)**

***Определение***.

Бинарное дерево поиска называется ***балансированным по высоте***, если ***для каждой его вершины высоты ее двух поддеревьев различаются не более, чем на 1***.

Деревья, удовлетворяющие этому условию, часто называют ***АВЛ-деревьями*** (по первым буквам фамилий их изобретателей ***Г.М.Адельсона-Вельского*** и ***Е.М.Ландиса***).

    Показателем балансированности вершины бинарного дерева называется  ***разность высоты его правого и левого поддерева***.

**bal=hR – hL**

Пусть **hR** - высота правого поддерева, **hL** - высота левого. Если дерево АВЛ-сбалансировано, то для каждого узла выполняется: **|hR – hL| <= 1**.

Показатель баланса **hR- hL** в сбалансированном АВЛ-дереве может принимать следующие значения:

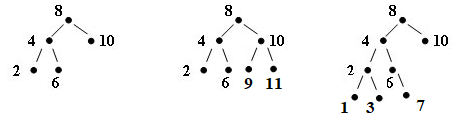
* -1, если левое поддерево на единицу выше правого;
* 0, если высоты обоих поддеревьев одинаковы;
* 1, если правое поддерево на единицу выше левого.

Если хотя бы для одного узла дерева это не так, то дерево не является АВЛ-сбалансированным. Примеры АВЛ-сбалансированного и АВЛ-несбалансированного дерева (у каждого узла указан показатель сбалансированности):

|  |  |
| --- | --- |
| АВЛ-сбалансированное дерево | АВЛ-несбалансированное дерево |

1. **Алгоритмы балансировки.** 
   1. **Общие положения**

    Рассмотрим следующее дерево:

  
Рис.3. Пример АВЛ дерева

    Вершины с ключами 9 и 11 можно включить в дерево, не нарушив его сбалансированности.

Однако включение значений 1, 3, 7 требует последующей балансировки.

    Алгоритм включения узла и балансировки существенно зависит от того, каким способом хранится информация о сбалансированности дерева. В дальнейшем мы будем хранить в каждой вершине показатель сбалансированности.

**bal=hR – hL**

Таким образом, класс узла АВЛ-дерева будет следующей:

***public class Node***

***{***

***public int Key; // Данные, используемые в качестве ключа***

***public int bal;***

***public Node leftChild; // Левый потомок узла***

***public Node rightChild; // Правый потомок узла***

***public Node parent; //Родитель***

***public void displayNode()***

***{***

***// (Тело метода )***

***}***

***} //Конец класса Node***

А класс самого дерева можно представить в виде

***class Tree***

***{***

***private Node root; // Единственное поле данных***

***public void Insert(int Key)***

***{***

***}***

***public void Delete(int Key)***

***{***

***}***

***public void Find(int Key)***

***{***

***}***

***// Другие методы***

***} // Конец класса Tree***

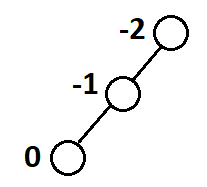
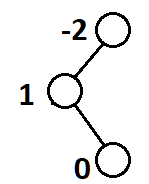
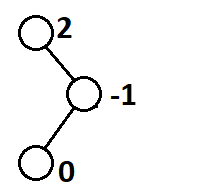
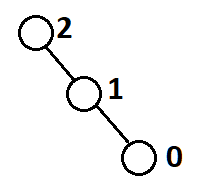
    Таким образом, можно сформулировать алгоритм включения узла в дерево:

* проход по дереву, чтобы убедиться, что включаемого значения в дереве нет;
* включение нового узла и определение результирующего показателя сбалансированности;
* "отступление" по пути поиска и проверка в каждом узле показателя сбалансированности. При необходимости - балансировка.

        После включения нового узла в АВЛ-дерево оно должно оставаться сбалансированным. Рассмотрим, в каких случаях потребуется балансировка дерева после включения узла. Во всех случаях будем указывать показатель сбалансированности корневого узла.

|  |  |  |
| --- | --- | --- |
| **До включения (дерево АВЛ-сбалансировано)** | **После включения** | |
| **В левое поддерево** | **В правое поддерево** |
| hR = hL  hR - hL= 0 | |  |  |  | | --- | --- | --- | |  | hR < hL  hR – hL= -1 |  |   Дерево сбалансировано | |  |  |  | | --- | --- | --- | |  | hR > hL  hR – hL= 1 |  |   Дерево сбалансировано |
| hR < hL  hR - hL= -1 | |  |  |  | | --- | --- | --- | |  | hR < hL  hR - hL= -2 |  |   Дерево не сбалансировано | hR = hL  hR - hL= 0  Дерево сбалансировано |
| hR > hL  hR - hL= 1 | hR = hL  hR - hL= 0  Дерево сбалансировано | |  |  |  | | --- | --- | --- | |  | hR > hL  hR - hL= 2 |  |   Дерево не сбалансировано |

Таким образом, возможны 4 варианта нарушения балансировки:

Каждому варианту нарушения балансировки соответствует своя комбинация значений показателей разбалансированности вершин.

* 1. **Однократный LL-поворот. (Правый поворот)**

**Простейшее дерево**

На рисунке 4. показано исходное несбалансированное дерево, направление правого поворота одной из его вершин и результирующее сбалансированное дерево

|  |  |  |
| --- | --- | --- |
|  |  |  |
| Исходное дерево | Правый поворот | Сбалансированное дерево |

Для балансировки этого дерева необходимо выполнить следующие операции

1. Сделать узел 3 правым потомком узла 2.

2. Удалить ссылку на левого потомка в узле 3.

3. Сделать узел 2 корневым - ссылку на дерево (на корневой узел 3) установить на узел 2.

4. Пересчитать показатели сбалансированности всех узлов.

    Программа, реализующая алгоритм балансировки может иметь следующий вид:

***p2.rightChild = p3;***

***p3.leftChild=null;***

***Tree=p2;***

***p3.parent=p2;***

***p2.parent=null***

***p2.bal=0;***

***p3.bal=0;***

В общем случае схему правого поворота можно представить в следующем виде

|  |  |
| --- | --- |
|  |  |
| До поворота (дерево не сбалансировано) | После поворота (дерево сбалансировано) |

Фрагмент программы, реализующий поворот может иметь вид:

***P.leftChild=L.rightChild;***

***L.rightChild=P;***

***Tree=L;***

***P.bal=0;***

***L.bal=0;***

***L.parent=null;***

***P.parent=L;***

* 1. **Однократный RR-поворот. (Левый поворот)**

    Этот алгоритм ничем не отличается от рассмотренного на предыдущем шаге, только поворот осуществляется в другую сторону.

|  |  |  |
| --- | --- | --- |
|  |  |  |
| Исходное дерево | Левый поворот | Сбалансированное дерево |

Для балансировки этого дерева необходимо выполнить следующие операции

1. Сделать узел 1 левым потомком узла 2.

2. Удалить ссылку на правого потомка в узле 1.

3. Сделать узел 2 корневым - ссылку на дерево (на корневой узел 1) установить на узел 2.

4. Пересчитать показатели сбалансированности всех узлов.

    Программа, реализующая алгоритм балансировки может иметь следующий вид:

***p2.leftChild = p1;***

***p1.righChild=null;***

***Tree=p2;***

***p1.parent=p2;***

***p2.parent=null***

***p2.bal=0;***

***p1.bal=0;***

В общем случае схему левого поворота можно представить в следующем виде

|  |  |
| --- | --- |
|  |  |
| До поворота (дерево не сбалансировано) | После поворота (дерево сбалансировано) |

Фрагмент программы, реализующий поворот может иметь вид:

***P.rightChild=L.leftChild;***

***L.leftChild=P;***

***Tree=L;***

***P.bal=0;***

***L.bal=0;***

***L.parent=null;***

***P.parent=L;***

* 1. **Двухкратный RL-поворот. (Большой левый поворот).**

    В этом случае для балансировки дерева необходимо выполнить два поворота.

|  |  |  |
| --- | --- | --- |
|  |  |  |
| Исходное дерево | Право-левый поворот | Сбалансированное дерево |

Для балансировки этого дерева необходимо выполнить следующие операции

1. Сделать узел 2 – правым потомком узла 3, а узел 3 правым потомком узла 1. (поменять узлы 3 и 2 местами

2. Удалить ссылку на левого потомка в узле 2. (Правый поворот)

3. Сделать узел 1 левым потомком узла 3.

4. Удалить ссылку на правого потомка узла 1. (Левый поворот)

3. Сделать узел 3 корневым - ссылку на дерево (на корневой узел 1) установить на узел 3.

4. Пересчитать показатели сбалансированности всех узлов.

    Программа, реализующая алгоритм балансировки может иметь следующий вид:



|  |  |
| --- | --- |
| ***p3.rightChild = p1.rightChild;*** |  |
| ***p1.rightChild=p2.leftChild***  ***p2.leftChild=null;*** |  |
| ***p3.leftChild=p1;***  ***p1.rightChild=null;*** |  |

***p2.parent=p3;***

***p1.parent=p3;***

***Tree=p3;***

***p3.parent=null;***

***p2.bal=0;***

***p1.bal=0;***

В общем случае схему право- левого поворота можно представить в следующем виде

|  |  |
| --- | --- |
|  |  |
| До поворота (дерево не сбалансировано) | После поворота (дерево сбалансировано) |

Фрагмент программы, реализующий поворот может иметь вид:

***P.rightChild=L.leftChild;***

***L.leftChild=P;***

***Tree=L;***

***P.bal=0;***

***L.bal=0;***

***L.parent=null;***

***P.parent=L;***

* 1. **Двухкратный LR-поворот. (Большой правый поворот).**

    Этот алгоритм ничем не отличается от рассмотренного на предыдущем шаге, только поворот осуществляется в другую сторону.

|  |  |  |
| --- | --- | --- |
|  |  |  |
| Исходное дерево | Право-левый поворот | Сбалансированное дерево |

Для балансировки этого дерева необходимо выполнить следующие операции

1. Сделать узел 2 – левым потомком узла 3, а узел 3 – левым потомком узла 1. (поменять узлы 3 и 2 местами

2. Удалить ссылку на правого потомка в узле 2. (Правый поворот)

3. Сделать узел 1 правым потомком узла 3.

4. Удалить ссылку на левого потомка узла 1. (Левый поворот)

3. Сделать узел 3 корневым - ссылку на дерево (на корневой узел 1) установить на узел 3.

4. Пересчитать показатели сбалансированности всех узлов.

    Программа, реализующая алгоритм балансировки может иметь следующий вид:



|  |  |
| --- | --- |
| ***p3.leftChild = p1.leftChild;*** |  |
| ***p1.leftChild=p2.rightChild***  ***p2.rightChild=null;*** |  |
| ***p3.rightChild=p1;***  ***p1.leftChild=null;*** |  |

***Tree=p3;***

***p2.parent=p3;***

***p1.parent=p3;***

***p3.parent=null;***

***p2.bal=0;***

***p1.bal=0;***

В общем случае схему право- левого поворота можно представить в следующем виде

|  |  |
| --- | --- |
|  |  |
| До поворота (дерево не сбалансировано) | После поворота (дерево сбалансировано) |

Фрагмент программы, реализующий поворот может иметь вид:

***P.rightChild=L.leftChild;***

***L.leftChild=P;***

***Tree=L;***

***P.bal=0;***

***L.bal=0;***

***L.parent=null;***

***P.parent=L;***

Во всех случаях балансировка АВЛ-дерева сводится к переопределению ссылок в некоторых узлах.

Поэтому сложность балансировки составляет O(1).

**2. Задание на лабораторную работу.**

* + 1. Построить двоичное дерево поиска из последовательности целых чисел (по варианту), внося их в дерево последовательно слева направо и изобразить это дерево.
    2. Определить тип поворота для балансировки дерева.
    3. Записать алгоритм балансировки.
    4. Изобразить дерево после каждой операции внесения изменений в дерево.
    5. Создать программу для реализации алгоритма (п.3).

Варианты задания

|  |  |
| --- | --- |
| Вариант | Последовательность чисел |
| 1 | 4, 6, 2, 8, 11 |
| 2 | 10, 12, 8, 3, 2 |
| 3 | 22, 25, 28, 15, 13, 5, 2 |
| 4 | 12, 3, 15, 17, 1, 23, 43 |
| 5 | 18, 16, 28, 12, 14 |
| 6 | 32, 50, 55, 30, 51 |
| 7 | 57, 59, 35, 30, 70, 25, 28 |
| 8 | 78, 55, 92, 96, 30, 98, 97 |
| 9 | 12, 20, 6, 24, 30 |
| 10 | 40, 51, 35, 30, 27 |
| 11 | 33, 36, 41, 27, 25, 13, 7 |
| 12 | 72, 16, 77, 85, 12, 95, 91 |
| 13 | 77, 81, 53, 45, 33, 39, 99 |
| 14 | 54, 64, 70, 43, 77 |
| 15 | 49, 47, 54, 45, 46 |

**3. Содержание отчета по лабораторной работе**

- наименование лабораторной работы и ее цель;

- задание на лабораторную работу согласно варианту;

- изображения дерева после создания и после каждой операции внесения изменений;

- тип поворота для балансировки созданного дерева;

- алгоритм балансировки для созданного дерева в текстовом виде;

- листинг программы для реализации алгоритма балансировки;

- выводы.

1. Контрольные вопросы
2. Какое дерево называется идеально сбалансированным?
3. Что такое АВЛ-сбалансированное дерево?
4. Как определяется разбалансированность АВЛ-дерева?
5. Что такое левый (правый) поворот дерева?
6. Для чего производится балансировка дерева?
7. Какова сложность (в O-нотации) операций балансировки дерева?