Министерство науки и высшего образования Российской Федерации
Калужский филиал
федерального государственного бюджетного образовательного
учреждения высшего образования
**«Московский государственный технический университет
имени Н.Э. Баумана
(национальный исследовательский университет)»**
(КФ МГТУ им. Н.Э. Баумана)

Ю.С. Белов, С.С. Гришунов

СОЗДАНИЕ И ОБРАБОТКА ДРЕВОВИДНЫХ СТРУКТУР ДАННЫХ

Методические указания к выполнению лабораторной работы
по курсу «Типы и структура данных»

Калуга – 2019

УДК 004.62
ББК 32.972.5
Б435

Методические указания составлены в соответствии с учебным планом КФ МГТУ им. Н.Э. Баумана по направлению подготовки 09.03.04 «Программная инженерия» кафедры «Программного обеспечения ЭВМ, информационных технологий».

Методические указания рассмотрены и одобрены:

- Кафедрой «Программного обеспечения ЭВМ, информационных технологий» (ИУ4-КФ) протокол № 51.4/5 от «23» января 2019 г.

Зав. кафедрой ИУ4-КФ _____ к.т.н., доцент Ю.Е. Гагарин

- Методической комиссией факультета ИУ-КФ протокол № 7 от «28» 01 2019 г.

Председатель методической
комиссии факультета ИУ-КФ _____ к.т.н., доцент М.Ю. Адкин

- Методической комиссией
КФ МГТУ им.Н.Э. Баумана протокол № 4 от «5» 02 2019 г.

Председатель методической комиссии
КФ МГТУ им.Н.Э. Баумана _____ д.э.н., профессор О.Л. Перерва

Рецензент:
к.т.н., доцент кафедры ИУ6-КФ _____ А.Б. Лачихина

Авторы
к.ф.-м.н., доцент кафедры ИУ4-КФ _____ Ю.С. Белов
ассистент кафедры ИУ4-КФ _____ С.С. Грищунов

Аннотация

Методические указания к выполнению лабораторной работы по курсу «Типы и структуры данных» содержат сведения о динамических древовидных структурах данных, с целью ознакомления с работой сложных динамических структур данных, используемых для создания программных комплексов обработки информации с элементами принятия решений. Рассматриваются приемы создания и обработки сбалансированных деревьев и деревьев поиска.

Предназначены для студентов 2-го курса бакалавриата КФ МГТУ им. Н.Э. Баумана, обучающихся по направлению подготовки 09.03.04 «Программная инженерия».

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ

Настоящие методические указания составлены в соответствии с программой проведения лабораторных работ по курсу «Типы и структуры данных» на кафедре «Программное обеспечение ЭВМ, информационные технологии» факультета «Информатика и управление» Калужского филиала МГТУ им. Н.Э. Баумана.

Методические указания к выполнению лабораторной работы по курсу «Типы и структуры данных» содержат сведения о видах деревьев, их создании и способах снижения трудоемкости поиска информации в древовидных структурах. Рассматриваются алгоритмы основных операций при работе с деревьями, а также их балансировка. Для закрепления теоретического и практического материала лабораторной работы в качестве задания приводится индивидуальный вариант для каждого студента, в котором необходимо реализовать все изученные алгоритмы работы с деревьями.

Предназначены для студентов 2-го курса бакалавриата КФ МГТУ им Н.Э. Баумана, обучающихся по направлению подготовки 09.03.04 «Программная инженерия».

ЦЕЛЬ И ЗАДАЧИ РАБОТЫ, ТРЕБОВАНИЯ К РЕЗУЛЬТАТАМ ЕЕ ВЫПОЛНЕНИЯ

Целью лабораторной работы является формирование практических навыков создания алгоритмов обработки древовидных структур данных.

Основными задачами выполнения лабораторной работы являются:

1. Изучить виды деревьев.
2. Научиться стоить двоичные деревья, деревья поиска.
3. Изучить способы балансировки деревьев.
4. Познакомиться с основными алгоритмами обработки деревьев.
5. Реализовать основные алгоритмы обработки древовидных структур данных (создание, удаление, поиск, добавление и удаление элемента), а также алгоритм согласно полученному варианту.

Результатами работы являются:

- консольное приложение, написанное с применением технологии ООП и запускаемое из командной строки;
- построенное двоичное дерево поиска;
- отчет

КРАТКАЯ ХАРАКТЕРИСТИКА ОБЪЕКТА ИЗУЧЕНИЯ, ИССЛЕДОВАНИЯ

Общие сведения

Древовидная модель оказывается довольно эффективной для представления динамических данных с целью быстрого поиска информации.

Деревья являются одними из наиболее широко распространенных структур данных в информатике и программировании, которые представляют собой иерархические структуры в виде набора связанных узлов.

Дерево – это структура данных, представляющая собой совокупность элементов и отношений, образующих иерархическую структуру этих элементов (рис. 1). Каждый элемент дерева называется *вершиной* (*узлом*) дерева. Вершины дерева соединены направленными дугами, которые называют *ветвями* дерева. Начальный узел дерева называют *корнем* дерева, ему соответствует нулевой уровень. *Листьями* дерева называют вершины, в которые входит одна ветвь и не выходит ни одной ветви.

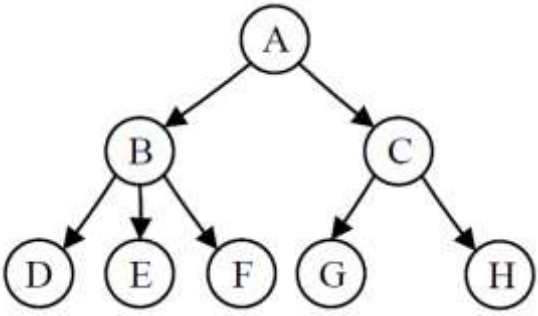

Рис. 1. Дерево

Каждое дерево обладает следующими свойствами:

- существует узел, в который не входит ни одной дуги (корень)
- в каждую вершину, кроме корня, входит одна дуга.

Деревья особенно часто используют на практике при изображении различных иерархий. Например, популярны генеалогические деревья.

Все вершины, в которые входят ветви, исходящие из одной общей вершины, называются *потомками*, а сама вершина – *предком*. Для каждого предка может быть выделено несколько. Уровень потомка на единицу превосходит уровень его предка. Корень дерева не имеет предка, а листья дерева не имеют потомков.

Высота (*глубина*) дерева определяется количеством уровней, на которых располагаются его вершины. Высота пустого дерева равна нулю, высота дерева из одного корня – единице. На первом уровне дерева может быть только одна вершина – корень дерева, на втором – потомки корня дерева, на третьем – потомки потомков корня дерева и т.д.

Поддерево – часть древообразной структуры данных, которая может быть представлена в виде отдельного дерева.

Степенью вершины в дереве называется количество дуг, которое из нее выходит. *Степень дерева* равна максимальной степени вершины, входящей в дерево. При этом листьями в дереве являются вершины, имеющие степень нуль.

По величине степени дерева различают два типа деревьев:

- двоичные – степень дерева не более двух;
- сильноветвящиеся – степень дерева произвольная.

Упорядоченное дерево – это дерево, у которого ветви, исходящие из каждой вершины, упорядочены по определенному критерию.

Деревья являются рекурсивными структурами, так как каждое поддерево также является деревом. Таким образом, дерево можно определить как рекурсивную структуру, в которой каждый элемент является:

- либо пустой структурой;
- либо элементом, с которым связано конечное число поддеревьев.

Действия с рекурсивными структурами удобнее всего описываются с помощью рекурсивных алгоритмов.

Списочное представление деревьев основано на элементах, соответствующих вершинам дерева. Каждый элемент имеет поле данных и два поля указателей: указатель на начало списка потомков вершины и указатель на следующий элемент в списке потомков текущего уровня. При таком способе представления дерева обязательно следует сохранять указатель на вершину, являющуюся корнем дерева.

Для того, чтобы выполнить определенную операцию над всеми вершинами дерева необходимо все его вершины просмотреть. Такая задача называется *обходом дерева*.

Обход дерева – это упорядоченная последовательность вершин дерева, в которой каждая вершина встречается только один раз.

При обходе все вершины дерева должны посещаться в определенном порядке. Существует несколько способов обхода всех вершин дерева. Выделим три наиболее часто используемых способа обхода дерева (рис. 2):

- *прямой* – сначала посещается корень, затем в прямом порядке узлы левого поддерева, далее все узлы правого поддерева;
- *симметричный* – сначала в симметричном порядке посещаются все узлы левого поддерева, затем корень, после чего в симметричном порядке все узлы правого поддерева;
- *обратный* – сначала посещаются в обратном порядке все узлы левого поддерева, затем в обратном порядке узлы правого поддерева, последним посещается корень.

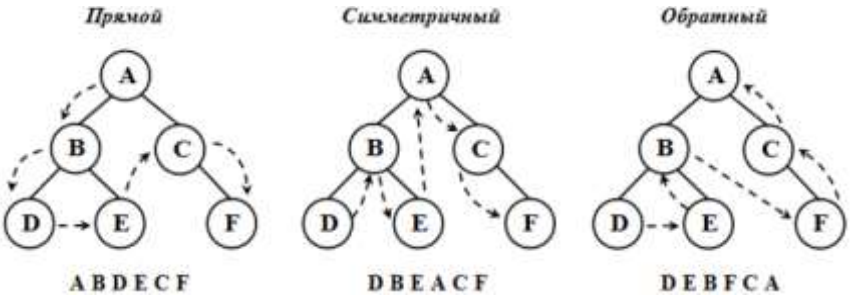

Рис. 2. Обходы дерева

Существует большое многообразие древовидных структур данных. Выделим самые распространенные из них: бинарные (двоичные) деревья, красно-черные деревья, B-деревья, АВЛ-деревья.

Бинарные деревья

Бинарные деревья являются деревьями со степенью не более двух.

Бинарное (двоичное) дерево – это динамическая структура данных, представляющее собой дерево, в котором каждая вершина имеет не более двух потомков (рис. 3). Таким образом, бинарное дерево состоит из элементов, каждый из которых содержит информационное поле и не более двух ссылок на различные бинарные поддеревья. На каждый элемент дерева имеется ровно одна ссылка.

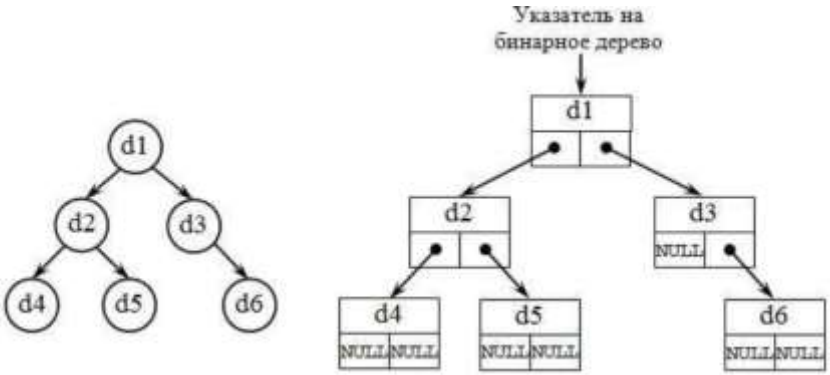

Рис. 3. Бинарное дерево и его организация

Каждая вершина бинарного дерева является структурой, состоящей из минимум трех видов полей. Содержимым этих полей будут соответственно:

- информационное поле (ключ вершины);
- служебное поле (их может быть несколько или ни одного);
- указатель на левое поддерево;
- указатель на правое поддерево.

По степени вершин бинарные деревья делятся на (рис. 4):

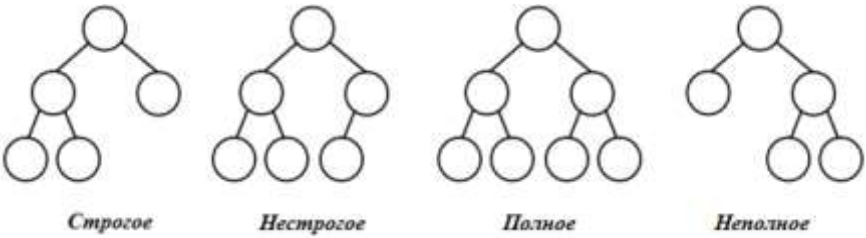

Рис. 4. Виды бинарных деревьев

Бинарное дерево может представлять собой пустое множество. Бинарное дерево может выродиться в список (рис. 5).

Рис. 5. Список как частный случай бинарного дерева

Двоичные деревья поиска

Двоичное дерево поиска (ДДП) упорядоченно, если для любой его вершины *x* справедливы такие свойства (рис. 6):

- все элементы в левом поддереве меньше элемента, хранимого в x,
- все элементы в правом поддереве больше элемента, хранимого в x,
- все элементы дерева различны.

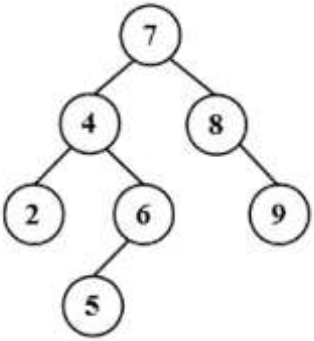

Рис. 6. Двоичное упорядоченное дерево

Если в дереве выполняются первые два свойства, но встречаются одинаковые элементы, то такое дерево является *частично упорядоченным*. В дальнейшем будет идти речь только о двоичных упорядоченных деревьях.

Операции для работы с ДДП

Основными операциями, производимыми с упорядоченным деревом, являются:

- добавление вершины;
- удаление вершины;
- вывод (печать) дерева;
- поиск вершины;
- поиск минимума;
- поиск максимума;
- удаление дерева.

ДДП строится по мере поступления элементов в дерево относительно корня. Далее, следуя законам ДДП, элементы, значения которых меньше корня, уходят в левое поддерево, а элементы, значения которых больше корня, – в правое.

Алгоритм удаления элемента более трудоемкий, так как необходимо соблюдать упорядоченность дерева. При удалении могут возникнуть следующие случаи:

- удаляемый элемент – лист;
- удаляемый элемент имеет одного потомка (потомок может быть как в правом поддереве, так и в левом);
- удаляемый элемент имеет двух потомков.

В последнем случае вершина имеет ссылки на реально существующие поддеревья. Эти поддеревья терять нельзя, а присоединить два поддерева на одно освободившееся после удаления место невозможно. Поэтому необходимо поместить на освободившееся место либо самый правый элемент из левого поддерева, либо самый левый из правого поддерева. Упорядоченность дерева при этом не

нарушится. Удобно придерживаться одной стратегии, например, заменять самый левый элемент из правого поддерева.

Исходя из свойств ДДП, к самому левому элементу из правого поддерева может привести поиск максимума, а к самому правому элементу из левого поддерева – поиск минимума.

простейшем случае вывод дерева можно организовать, повернув его на 90° вправо, обращаясь рекурсивно к правому поддереву, корню и левому поддереву.

Удаление дерева происходит, также рекурсивно обращаясь левому поддереву, правому поддереву и корню.

Сбалансированные деревья

В худшем случае, когда дерево вырождено в линейный список, хранение данных в упорядоченном бинарном дереве никакого выигрыша в сложности операций по сравнению с массивом или линейным списком не дает. В лучшем случае, когда дерево сбалансировано, для всех операций получается логарифмическая сложность, что гораздо лучше. Идеально сбалансированным называется дерево, у которого для каждой вершины выполняется требование: число вершин в левом и правом поддеревьях различается не более чем на 1.

Однако идеальную сбалансированность довольно трудно поддерживать. В некоторых случаях при добавлении или удалении элементов может потребоваться значительная перестройка дерева, не гарантирующая логарифмической сложности. В 1962 году два советских математика Г.М. Адельсон-Вельский и Е.М. Ландис ввели менее строгое определение сбалансированности и доказали, что при таком определении можно написать программы добавления и/или удаления, имеющие логарифмическую сложность и сохраняющие дерево сбалансированным. Дерево считается *сбалансированным по АВЛ* (сокращения от фамилий Г.М. Адельсон-Вельский и Е.М. Ландис, 1962), если для каждой вершины выполняется требование: высота левого и правого поддеревьев различаются не более, чем на 1. Не всякое

сбалансированное по АВЛ дерево идеально сбалансировано, но всякое идеально сбалансированное дерево сбалансировано по АВЛ.

При операциях добавления и удаления может произойти нарушение сбалансированности дерева. В этом случае потребуются некоторые преобразования, не нарушающие упорядоченности дерева и способствующие лучшей сбалансированности.

Алгоритм балансировки сводится к двум действиям:

1. дерево перестраивается в лозу (рис. 7);

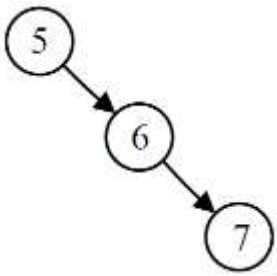

Рис. 7. Лоза

2. лоза перестраивается в дерево (рис. 8).

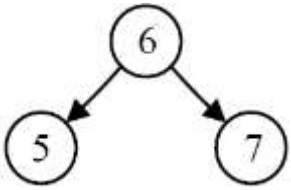

Рис. 8. Дерево

Для поддержания балансировки деревьев необходимо пользоваться еще одним служебным полем в каждом узле дерева – *высотой*. Высота каждый раз обновляется при добавлении узла дерево или при его удалении. Высота листа равна 0. Высота пустого поддерева = -1.

Далее, анализируя высоту корня поддерева, а точнее разницу высот левого и правого поддеревьев, к нему применяют повороты. Различают следующие типы поворотов:

– одиночный правый поворот (R-rotation);
– одиночный левый поворот (L-rotation);
– двойной лево-правый поворот (LR-rotation);
– двойной право-левый поворот (RL-rotation).

Одиночный правый поворот

Рассмотрим следующий пример. Имеется дерево следующего вида (рис. 9):

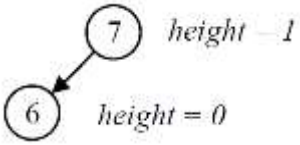

Рис. 9.

Добавляем узел 5. Получаем (рис. 10):

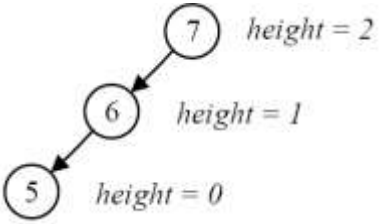

Рис. 10.

Разница левого и правого поддерева равна 2. Дерево разбалансировано, следовательно, необходимо применить поворот. Поскольку перегружена левая часть, необходимо применить одиночный правый поворот (рис. 11).

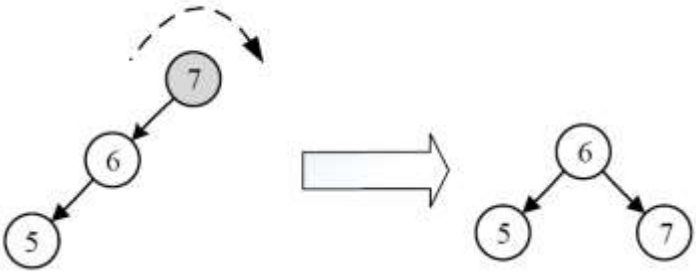

Рис. 11. Одиночный правый поворот

Одиночный левый поворот

Аналогично применяется простой поворот влево, но тогда, когда узлы располагаются в правом поддереве (Рис. 12). А разница высот проверяется как разница между правым и левым поддеревом.

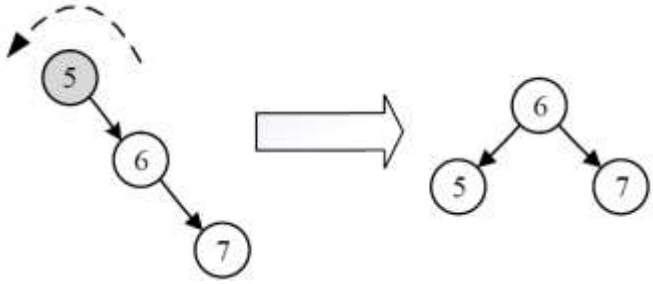

Рис. 12. Одиночный левый поворот

Двойной лево-правый поворот

Двойные повороты применяются, когда сначала нужно получить лозу, а затем перестроить в дерево (Рис. 13). Двойной лево-правый поворот применяется, когда добавляемый узел поступает в правое поддерево левого поддерева.

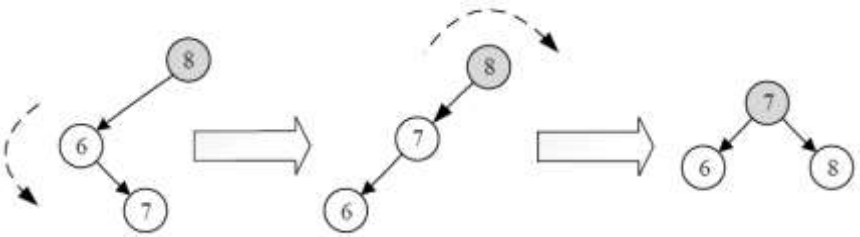

Рис. 13. Двойной лево-правый поворот

Двойной право-левый поворот

Двойной право-левый поворот применяется, когда добавляемый узел поступает в левое поддерево правого поддерева (рис. 14).

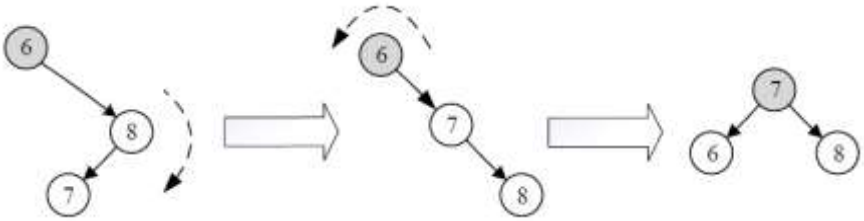

Рис. 14. Двойной право-левый поворот

ТЕСТИРОВАНИЕ И РЕАЛИЗАЦИЯ ДЕРЕВЬЕВ

Реализация операций над двоичными деревьями поиска приведена в виде отдельных методов на языке C++.

Первое, что необходимо сделать, это определить структуру узла в дереве, которая должна состоять как минимум из одного информационного поля и двух адресных полей, указывающих на левое и правое поддерево:

```
struct BSTree {
int _data;
BSTree *_left;
BSTree *_right;
}
```

Добавление вершины в ДДП

```
BSTree * Insert(BSTree *, int data)
{
    if (t == NULL)
    {
          return CreateNode(data);
    }

    if (data < t->_data)
    {
          t->_left = Insert(t->_left, data);
    }
    else if (data > t->_data)
    {
        t->_right = Insert(t->_right, data);
    }

    return t;
}
```

Удаление вершины из ДДП

```
BSTree * DeleteNode(BSTree *_root, int _data)
{
    if (_root == nullptr)
          return nullptr;
    if (_data == _root->data)
    {
        if (_root->left == nullptr &&
              _root->right == nullptr)
        {
              delete _root;
              return nullptr;
        }
        if (_root->left == nullptr &&
              _root->right != nullptr)
        {
              BSTree * temp = _root->right;
              delete _root;
              return temp;
        }
        if (_root->right == nullptr &&
              _root->left != nullptr)
        {
              BSTree * temp = _root->left;
              delete _root;
              return temp;
        }
        _root->data = GetMinimum(_root->right)->data;
        _root->right = DeleteNode(_root->right,
              _root->data);
        return _root;
    }
    if (_data < _root->data)
    {
        _root->left = DeleteNode(_root->left,_data);
        return _root;
    }
    if (_data > _root->data)
    {
        _root->right = DeleteNode(_root->right, _data);
        return _root;
    }
  return _root;
}
```

Вывод (печать) дерева

В данном методе `t` является указатель на корень дерева, а `n` – определяет количество пробелов, выводимых при печати узла дерева.

```
void PrintBSTree(BSTree *t, int n)
{
    int i;
    if (t != NULL) {
          PrintBSTree(t->_right, n + 1);
          for (i = 0; i < n; i++)
                cout << " ";
          cout << t->_data;
          PrintBSTree(t->_left, n + 1);
    }
    else
          cout << endl;
}
```

Поиск вершины в дереве

```
BSTree *FindNode(int data) {
    BSTree *current = root;
    while (current != NULL)
          if (data == current->_data)
                return current;
          else
                current = data < current->_data ?
                    current->_left : current->_right;
    return NULL;
}
```

Поиск минимума в ДДП

```
BSTree * GetMinimum(BSTree *_root)
{
    if (!_root->left)
    {
          return _root;
    }
    return GetMinimum(_root->left);
}
```

Поиск максимума в ДДП

```
BSTree * GetMaximum(BSTree *_root){
    if (!_root->right)
    {
          return _root;
    }
    return GetMaximum(_root->right);
   }
```

Удаление дерева

```
void DeleteTree(BSTree *t) {
    if (t != NULL) {
          DeleteTree(t->_right);
          DeleteTree(t->_left);
          delete t;
    }
}
```

Балансировка ДДП

Как упоминалось выше, для создания сбалансированного дерева поиска необходимо предусмотреть наличие в каждом узле дерева еще одного служебного поля – высоты дерева. Поэтому определим новую структуру, характерную для сбалансированных деревьев по АВЛ.

```
struct AvlTree
{
    int _data;
    int _height;
    AvlTree * _left, * _right;
}
```

Для определения высоты дерева узла t необходима функция Height, определяющая высоту дерева как существующее значение поля _height узла t или как -1, в случае, если поддерева не существует.

```
int Height(AvlTree *t)
{
    return (t != NULL) ? t->_height : -1;
}
```

Поле `_height` узла `t` определяется как сумма максимальной из высот левого и правого поддеревьев и 1 при добавлении его в дерево:

```
t->_height = MaxHeight(Height(t->_left),
     Height(t->_right)) + 1,
```

где

```
int MaxHeight(int h1, int h2)
{
    return h1 > h2 ? h1 : h2;
}
```

Далее рассмотрим повороты. Задача поворотов заключается в переопределении ссылок узла `t` на потомков, согласно схеме поворота.

```
AvlTree * RightRotate(AvlTree * t)
{
    AvlTree * left;

    left = t->_left;
    t->_left = left->_right;
    left->_right = t;

    t->_height = MaxHeight(Height(t->_left),
          Height(t->_right)) + 1;
    left->_height = MaxHeight(Height(left->_left),
          t->_height) + 1;
    return left;
}


AvlTree * LeftRotate(AvlTree * t)
{
    AvlTree * right;
    right = t->_right;
    t->_right = right->_left;
    right->_left = t;

    t->_height = MaxHeight(Height(t->_left),
          Height(t->_right)) + 1;
```

```
    right->_height = MaxHeight(Height(right->_right),
          t->_height) + 1;
    return right;
}
AvlTree * LeftRightRotate(AvlTree *t)
{
    t->_left = LeftRotate(t->_left);
    return RightRotate(t);
}
AvlTree * RightLeftRotate(AvlTree *t)
{
    t->_right = RightRotate(t->_right);
    return LeftRotate(t);
}
```

Балансировка дерева происходит в процессе добавления узлов в дерево или удаления из дерева. Рассмотрим реализацию добавления узла в дерево.

```
AvlTree * CreateNode(int data)
{
    AvlTree * node = new AvlTree;
    node->_data = data;
    node->_left = NULL;
    node->_right = NULL;
    node->_height = 0;

    return node;
}

AvlTree * Insert(AvlTree *t, int data)
{
    if (t == NULL)
    {
        return CreateNode(data);
    }

    if (data < t->_data)
    {
        t->_left = Insert(t->_left, data);
        if (Height(t->_left) - Height(t->_right) == 2)
        {
```

```
            // дерево разбалансировано
            if (data < t->_left->_data)
            {
               t = RightRotate(t);
            }
            else
            {
               t = LeftRightRotate(t);
            }
         }
      }
      else if (data > t->_data)
      {
            t->_right = Insert(t->_right, data);
            if (Height(t->_right) - Height(t->_left) == 2)
            {
           // дерево разбалансировано
               if (data > t->_right->_data)
               {
                  t = LeftRotate(t);
               }
               else
               {
                  t = RightLeftRotate(t);
               }
            }
      }
      t->_height = MaxHeight(Height(t->_left),
          Height(t->_right)) + 1;

      return t;
   }
```

ЗАДАНИЕ НА ЛАБОРАТОРНУЮ РАБОТУ

1. Разработать консольное приложение, написанное с помощью объектно-ориентированной технологии. Индивидуальное задание предусмотрено вариантом, который назначит преподаватель.
2. Приложение необходимо запускать для демонстрации из командной строки с указанием названий приложения и трех файлов:
3. все входные данные (например, последовательности чисел, коэффициенты многочленов и т.д.) считать из первого файла;
   - все выходные данные записать во второй файл;
   - все возникшие ошибки записать в третий файл – файл ошибок.
   - все основные сущности вашего приложения представить в виде отдельных классов.
4. Необходимо предусмотреть пользовательское меню, содержащее набор команд всех основных операций для работы с двоичным деревом, а также команду для запуска индивидуального задания.
5. В приложении также должны быть учтены все критические ситуации, обработанные с помощью класса исключений.

ВАРИАНТЫ ЗАДАНИЙ

1. Построить двоичные деревья поиска для ключей из диапазона 1..10, изменяя порядок, в котором величины вставляются в дерево. Для каждого из деревьев определить его длину и вывести на экран деревья максимальной и минимальной длины.
2. Построить бинарное дерево следующего выражения 7 + (8 * (4 - 1) и вывести его на экран. Написать процедуры постфиксного, инфиксного и префиксного обхода дерева и вывести соответствующие выражения.

3. Построить бинарное дерево следующего выражения ((6 * 3) + (8*7) * (6*5) и вывести его на экран. Написать процедуры постфиксного, инфиксного и префиксного обхода дерева и вывести соответствующие выражения.
4. Построить бинарное дерево следующего выражения ((3 + 4) * (8 (3 DIV 2))) * (9 * (7 + 4)) и вывести его на экран. Написать процедуры постфиксного, инфиксного и префиксного обхода дерева и вывести соответствующие выражения.
5. Построить бинарное дерево следующего выражения ((((5 * 2) +3) * 4) + 5) *6 и вывести его на экран. Написать процедуры постфиксного, инфиксного и префиксного обхода дерева и вывести соответствующие выражения.
6. Построить бинарное дерево следующего выражения ((1 + 2) * (3 + 4)) * ((5 +6) * (7 +8)) и вывести его на экран. Написать процедуры постфиксного, инфиксного и префиксного обхода дерева и вывести соответствующие выражения.
7. Построить бинарное дерево следующего выражения 9 + (8 * (7 + (6 * (5 + (4 * 3))))) и вывести его на экран. Написать процедуры постфиксного, инфиксного и префиксного обхода дерева и вывести соответствующие выражения.
8. Построить двоичное дерево, в вершинах которого находятся слова. Написать процедуры:
    a) обхода дерева сверху вниз;
    b) определения количества вершин дерева, содержащих слова, начинающиеся на одну и туже букву;
    c) вставки слова в дерево, если такого слова в дереве нет.
9. Написать программу-словарь, позволяющую пользователю проверять правильность написания слов. Реализация программы должна обеспечить представление словаря в виде дерева двоичного поиска, словарь хранится в виде текстового файла. При каждом выполнении программы словарь может быть модифицирован, если пользователь решил добавить в него слова. Программа должна обеспечивать диалоговое взаимодействие с пользователем.

10. В файловой системе каталог файлов организован в виде сбалансированного бинарного дерева. Каждый узел обозначает файл, содержащий имя и атрибуты файла, в том числе и дату последнего обращения к файлу. Написать программу, которая обходит дерево и удаляет все файлы, последнее обращение к которым происходило до определенной даты, при этом сбалансированность бинарного дерева сохраняется.
11. Написать программу, которая определяет последовательность из n чисел, для которой формируется идеально сбалансированное дерево двоичного поиска. Последовательность чисел и дерево поиска вывести на экран монитора.
12. В заданном непустом бинарном дереве подсчитать число вершин на *n*-ом уровне, считая корень вершиной 0-го уровня.
13. Написать процедуру построения сбалансированного дерева, в узлах которого расположены целые числа в диапазоне 1...20. Удалить из дерева все четные числа с сохранением сбалансированности дерева.
14. Построить двоичное дерево, каждый узел которого содержит букву и целое число, являющееся частотой обращения к данному узлу. Написать процедуру, которая создает новое дерево, в котором узлы располагаются в порядке убывания счетчика частоты обращений и вывести его на экран. Определить среднюю длину пути в исходном дереве и после реорганизации дерева.
15. Построить сбалансированное дерево из букв латинского алфавита и вывести его на экран слева направо. Написать процедуру удаления из дерева гласных букв с сохранением сбалансированности дерева.
16. Построить сбалансированное дерево из целых чисел и вывести его на экран снизу-вверх. Написать процедуру удаления элементов из дерева с сохранением сбалансированности дерева. Найти дерево и короткую последовательность удалений, вызывающую появление всех четырех ситуаций балансировки хотя бы по одному разу.

17. Построить сбалансированное дерево из целых чисел и вывести его на экран сверху вниз. Определить среднюю длину пути в дереве.
18. В текстовом файле записаны целые числа. Построить двоичное дерево, элементами которого являются числа из файла. Написать процедуры:
    a) обхода дерева слева направо;
    b) удаления из дерева всех нечетных чисел.
19. В текстовом файле записаны целые числа. Построить двоичное дерево, элементами которого являются числа из файла. Написать нерекурсивную процедуру обхода дерева сверху вниз, а также процедуру, которая вычисляет сумму элементов дерева, кратных 3.
20. В текстовом файле записаны целые числа. Построить двоичное дерево, элементами которого являются числа из файла. Написать нерекурсивную процедуру обхода дерева слева направо, а также процедуру, которая определяет число узлов дерева на каждом уровне.
21. В текстовом файле записаны целые числа. Построить двоичное дерево, элементами которого являются числа из файла. Написать нерекурсивную процедуру обхода дерева снизу-вверх, а также процедуру, которая определяет число узлов в левом и правом поддеревьях двоичного дерева.
22. Построить двоичное дерево из букв строки и написать следующие процедуры:
    a) обход дерева сверху вниз;
    b) удаление из дерева повторяющихся букв.
23. Построить двоичное дерево из букв строки и написать следующие процедуры:
    a) обход дерева слева направо;
    b) удаление из дерева согласных букв.

24. Построить двоичное дерево из букв строки и написать следующие процедуры:
    a) обход дерева снизу-вверх;
    b) вывод самого правого элемента левого поддерева и самого левого элемента правого поддерева.
25. Построить двоичное дерево из целых чисел и написать следующие процедуры:
    a) вывод элементов дерева по уровням;
    b) удаление из дерева отрицательных элементов.
26. Построить двоичное дерево из целых чисел и написать следующие процедуры:
    a) вывод элементов дерева сверху вниз;
    b) удаление из дерева нечетных элементов.
27. Построить двоичное дерево из целых чисел и написать следующие процедуры:
    a) вывод элементов дерева снизу-вверх;
    b) все отрицательные элементы дерева заменить на 0.
28. Построить двоичное дерево из целых чисел и написать следующие процедуры:
    a) вывод элементов дерева по уровням;
    b) определение количества элементов дерева на каждом уровне.
29. Построить двоичное дерево из букв латинского алфавита и написать следующие процедуры:
    a) вывод элементов дерева слева направо;
    b) удаление из дерева гласных букв.
30. В заданном непустом бинарном дереве подсчитать число вершин на *n*-ом уровне, считая корень вершиной 0-го уровня.
31. Заданную последовательность целых чисел, оканчивающуюся нулем, представить в виде дерева поиска, затем преобразовать его, добавив еще одно целое число; распечатать преобразованное дерево в прямом, обратном, симметричном порядке.

32. Построить двоичное дерево, каждый узел которого содержит букву и целое число, являющееся частотой обращения к данному узлу. Написать процедуру, которая создает новое дерево, в котором узлы располагаются в порядке убывания счетчика частоты обращений и вывести его на экран. Определить среднюю длину пути в исходном дереве и после реорганизации дерева.
33. Написать процедуру построения сбалансированного дерева, в узлах которого расположены целые числа в диапазоне 1...20. Удалить из дерева все нечетные числа с сохранением сбалансированности дерева.
34. В заданном бинарном дереве подсчитать число его листьев и напечатать их значения:
    a) при прямом обходе дерева;
    b) при обратном обходе дерева;
    c) при симметричном обходе дерева;
35. Написать программу-словарь, позволяющую пользователю проверять правильность написания слов. Реализация программы должна обеспечить представление словаря в виде дерева двоичного поиска, словарь хранится в виде текстового файла. При каждом выполнении программы словарь может быть модифицирован, если пользователь решил добавить в него слова. Программа должна обеспечивать диалоговое взаимодействие с пользователем.
36. В заданном бинарном дереве найти первое вхождение заданного элемента и напечатать пройденные при поиске узлы дерева:
    a) при прямом обходе дерева;
    b) при обратном обходе дерева;
    c) при концевом обходе дерева.

КОНТРОЛЬНЫЕ ВОПРОСЫ И ЗАДАНИЯ

1. Дайте определение дерева, двоичного дерева, дерева поиска.
2. Опишите представление деревьев в памяти компьютера.
3. Перечислите и раскройте способы задания дерева.
4. Назовите три способа обхода деревьев.
5. Раскройте алгоритм вставки элемента в дерево.
6. Объясните алгоритм удаления элемента из дерева.
7. Опишите процесс поиска элемента в дереве.
8. Объясните принцип поиска максимального и минимального элемента в двоичном дереве.
9. Опишите процесс удаления дерева.
10. Опишите о способы балансировки деревьев.

ФОРМА ОТЧЕТА ПО ЛАБОРАТОРНОЙ РАБОТЕ

На выполнение лабораторной работы отводится 4 занятия (8 академических часов: 7 часов на выполнение и сдачу практического задания и 1 час на подготовку отчета).

Номер варианта студенту выдается преподавателем.

Отчет на защиту предоставляется в печатном виде.

Структура отчета (на отдельном листе(-ах)): титульный лист, формулировка задания, этапы выполнения работы, результаты выполнения, выводы.

ОСНОВНАЯ ЛИТЕРАТУРА

1. Алексеев В.Е. Графы и алгоритмы. Структуры данных. Модели вычислений [Электронный ресурс]/ В.Е. Алексеев, В.А. Таланов. — Электрон. текстовые данные. — М.: Интернет-Университет Информационных Технологий (ИНТУИТ), 2016. — 153 с. — Режим доступа: http://www.iprbookshop.ru/52186.html

2. Вирт Никлаус. Алгоритмы и структуры данных [Электронный ресурс]/ Никлаус Вирт— Электрон. текстовые данные. — Саратов: Профобразование, 2017. — 272 с.— Режим доступа: http://www.iprbookshop.ru/63821.html

3. Самуйлов С.В. Алгоритмы и структуры обработки данных [Электронный ресурс]: учебное пособие/ С.В. Самуйлов. — Электрон. текстовые данные. — Саратов: Вузовское образование, 2016. — 132 с.— Режим доступа: http://www.iprbookshop.ru/47275.html

ДОПОЛНИТЕЛЬНАЯ ЛИТЕРАТУРА

4. Костюкова Н.И. Графы и их применение [Электронный ресурс]/ Н.И. Костюкова. — Электрон. текстовые данные. — М.: Интернет-Университет Информационных Технологий (ИНТУИТ), 2016. — 147 с. — Режим доступа: http://www.iprbookshop.ru/52185.html

5. Сундукова Т.О. Структуры и алгоритмы компьютерной обработки данных [Электронный ресурс]/ Т.О. Сундукова, Г.В. Ваныкина. — Электрон. текстовые данные. — М.: Интернет-Университет Информационных Технологий (ИНТУИТ), 2016. — 749 с.— Режим доступа: http://www.iprbookshop.ru/57384.html

Электронные ресурсы:

6. Научная электронная библиотека http://elibrary.ru
7. Электронно-библиотечная система «ЛАНЬ» http://e.lanbook.com
8. Электронно-библиотечная система «IPRbooks» http://www.iprbookshop.ru
9. Электронно-библиотечная система «Юрайт» http://www.biblio-online.ru
10. Электронно-библиотечная система «Университетская библиотека ONLINE» http://www.biblioclub.ru