Лабораторная работа № 12

Абстрактный тип данных – Бинарное дерево поиска

Под бинарным деревом понимают иерархическую структуру данных (дерево), в которой каждый узел (родительский узел) имеет не более двух ветвей: левой и правой. Узел, не имеющий родителя, будем называть корневым узлом (root), а узлы, не имеющие ветвей, будем называть листьями.

Пусть узлы бинарного дерева содержат некоторые уникальные значения, называемые ключами, для которых определена операция сравнения.

Бинарное дерево поиска (binary search tree, BST) — это бинарное дерево с ключами со следующими дополнительными свойствами:

- для каждого узла дерева обе ветви (левая и правая) являются бинарными деревьями поиска;
- у всех узлов левой ветви произвольного узла X значения ключей меньше, чем значение ключа самого узла X;
- у всех узлов правой подветви произвольного узла X значения ключей больше, чем значение ключа самого узла X.

Рассмотрим, как бинарное дерево поиска может быть использовано для представления иерархического процесса поиска в соответствии с алгоритмом бинарного поиска.

Алгоритм бинарного поиска в массиве позволяет быстро найти элемент данных в массиве при условии, что каждый элемент данных имеет уникальный ключ, и что элементы данных в массиве упорядочены по ключам. Например, для указанного ниже массива ключей

Индекс	0	1	2	3	4	5	6
Ключ	16	20	31	43	65	72	86

бинарный поиск элемента данных с ключом 31 начнется в середине массива с элемента с ключом 43. Так как 31 меньше 43, элемент с ключом 31 должен находиться в первой половине массива (среди элементов с индексами 0÷2). Ключ в середине этого подмассива равен 20, поэтому элемент с ключом 31 находится в верхней половине этого подмассива (элемент с индексом 2). Элемент с индексом 2 содержит ключ 31 и поиск на этом закончен.

Хотя конкретные сравнения в алгоритме бинарного поиска зависят от ключа, относительный порядок, в котором сравниваются значения, инвариантен для заданного массива элементов. Например, в предыдущем примере мы всегда сравниваем ключ поиска со значением 43, затем с 20 или 72. порядок сравнений показан ниже:

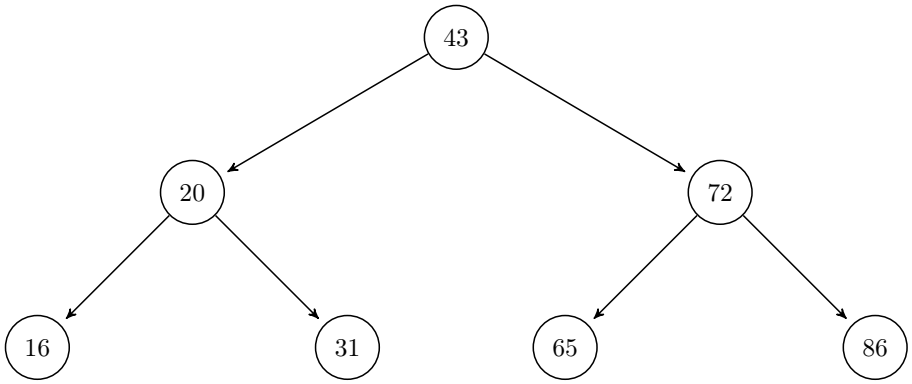
Индекс	0	1	2	3	4	5	6
Ключ	16	20	31	43	65	72	86
Порядок	3	2	3	1	3	2	3

Ранее мы использовали бинарный поиск в массиве при реализации класса OrdList:

```
bool OrdList::binarySearch ( int searchKey, int &index ) {
    int low = 0, // Low index of current search range
        high = size - 1; // High index of current search range
```

```
    bool result; // Result returned
    while ( low <= high ) {
        index = ( low + high ) / 2; // Compute midpoint
        if ( searchKey < dataItems[index].getKey() )
            high = index - 1; // Search lower half
        else if ( searchKey > dataItems[index].getKey() )
            low = index + 1; // Search upper half
        else
            break; // searchKey found
    }
    if ( low <= high )
        result = true; // searchKey found
    else {
        index = high; // searchKey not found, adjust index
        result = false;
    }
    return result;
}
```

Иерархическая природа сравнений ключей в алгоритме бинарного поиска может быть показана на следующем дереве:



Обратите внимание, что для каждого ключа K в этом дереве все ключи в левом поддереве меньше, чем K, и все ключи в правом поддереве больше, чем K. Таким образом, мы имеем бинарное дерево поиска.

При поиске заданного ключа в бинарном дереве поиска мы начинаем с корневого узла и движемся вниз вдоль соответствующей ветви, пока не находим узел с нужным ключом или не достигаем листа (терминального узла), не найдя ключа. Каждое движение вдоль ветви соответствует разделению массива в алгоритме бинарного поиска. В каждом узле движемся влево, если ключ поиска меньше значения ключа в узле, или вправо, если ключ поиска больше значения ключа в узле.

Для реализации бинарного дерева поиска будем использовать два класса: один класс для узлов дерева (класс BSTreeNode), другой класс для общей структуры дерева (класс BSTree). Каждый узел в дереве, являющийся объектом класса BSTreeNode, содержит элемент данных dataItem общего типа DT, имеющий ключ общего типа KF, и два указателя left и right на дочерние узлы (если таковые имеются). В классе BSTree поддерживается указатель root на корневой узел бинарного дерева.

```
template < class DT, class KF > // DT : tree data item
class BSTree                    // KF : key field
{
public:
    BSTree ();  // Constructor
    ~BSTree (); // Destructor
    // Binary search tree status operations
    bool isEmpty () const;                  // Tree is empty
    // Binary search tree manipulation operations
    void insert ( const DT &newDataItem ); // Insert data item
    bool retrieve ( KF searchKey, DT &searchDataItem ) const; // retreive data item
    bool remove ( KF deleteKey );           // Remove data item
    void clear ();                          // Clear tree
```

```
    int getHeight () const;                  // Height of tree
    int getCount () const;                   // Number of nodes in tree
    void writeLessThan ( const KF& searchKey ) const; // Output keys < searchKey
    // Output the tree structure
    void show () const;
private:
    class BSTreeNode       // Inner class: facilitator for the BSTree class
    {
    public:
        BSTreeNode ( const DT &nodeDataItem, // Constructor
                     BSTreeNode *leftPtr, BSTreeNode *rightPtr );
        // Data members
        DT dataItem;       // Binary search tree data item
        BSTreeNode *left,  // Pointer to the left child
                   *right; // Pointer to the right child
    };
    // Recursive partners of the public member functions
    void insertHelper(const DT& newDataItem, BSTreeNode*& p);
    void showHelper ( BSTreeNode *p, int level ) const;
    // insert your prototypes of these functions here...
    // Data member
    BSTreeNode *root;    // Pointer to the root node
};
```

Задание 12.1. (5 баллов) Дан работающий шаблон интерактивной программы для работы с бинарным деревом поиска, поддерживающий следующие команды с консоли:

Команда	Действия
+x	Поместить элемент x в дерево
?x	Найти и вывести элемент xна консоль
-x	Удалить элемент x из дерева
C	Очистить дерево
E	Вывести, является ли дерево пустым
G	Вывести общее число узлов в дереве
H	Вывести высоту дерева
<x	Вывести ключи, которые меньше x
Q	Выйти из программы

Разработайте один из методов класса BSTree (в зависимости от варианта) и оттестируйте разработанный метод при помощи интерактивной программы:

Вариант 1. Метод retrieve

Вариант 2. Метод remove

Вариант 3. Метод clear

Вариант 4. Метод getHeight

Вариант 5. Метод getCount

Вариант 6. Метод writeLessThan

Напоминание

Выполненные задания отправляются со своей электронной почты на почту `labs_it@mail.ru`. Тема письма содержит номер группы, фамилию студента и номер задания, например:

НПИ-01-18 Иванов 4.1

В письмо включается текст программы, скриншот всего экрана с результатом работы программы (с системной датой и временем), пояснения (при необходимости).

Если Вы не успели выполнить задание полностью во время лабораторной работы, то в конце занятия Вы должны отправить на почту файл с текущим состоянием проекта (программы) и указанием выполненных и невыполненных работ. В этом случае сохраняется прежний порядок оценки работы (задание оценивается с максимальной оценкой, если программа корректно решает поставленную задачу и задание отправлено в ходе занятия или в день проведения занятия).

Если в конце занятия файл с программой на почту не поступил или за время лабораторной работы никакие работы не выполнены, то задание оценивается исходя из 75% своей максимальной оценки.

Будьте готовы презентовать Ваше решение в начале следующего занятия.