Лабораторная работа №8

Деревья

Цель работы: реализовать работу с бинарными деревьями на языке C#.

Теоретическая часть

Бинарные деревья

Бинарное дерево — это иерархическая структура данных, в которой каждый узел имеет значение и ссылки на левого и правого потомков. Узел, находящийся на самом верхнем уровне, называется корнем. Узлы, не имеющие потомков, называются листьями.

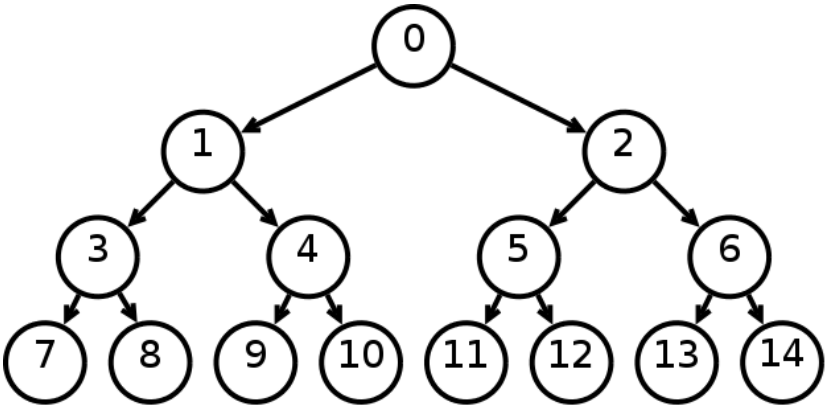

Рисунок 1 – Бинарное дерево

Бинарное дерево поиска — это бинарное дерево, обладающее дополнительными свойствами: значение левого потомка меньше значения родителя, а значение правого потомка больше значения родителя для каждого узла дерева. То есть, данные в бинарном дереве поиска хранятся в отсортированном виде. При каждой операции вставки нового или удаления существующего узла отсортированный порядок дерева сохраняется. При поиске элемента сравнивается искомое значение с корнем. Если искомое больше корня, то поиск продолжается в правом потомке корня, если меньше, то в левом, если равно, то значение найдено и поиск прекращается.

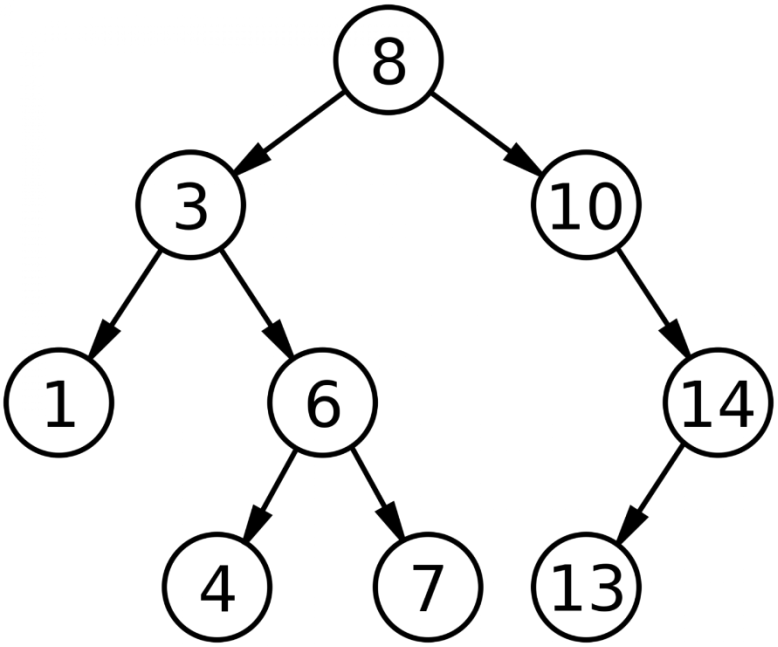

Рисунок 2 – Бинарное дерево поиска

Для целей реализации бинарное дерево поиска можно определить так:

- Бинарное дерево состоит из узлов (вершин) — записей вида (data, left, right), где data — некоторые данные, привязанные к узлу, left и right — ссылки на узлы, являющиеся детьми данного узла — левый и правый сыновья соответственно. Для оптимизации алгоритмов конкретные реализации предполагают также определения поля parent в каждом узле (кроме корневого) — ссылки на родительский элемент.
- Данные (data) обладают ключом (key), на котором определена операция сравнения «меньше». В конкретных реализациях это может быть пара (key, value) — (ключ и значение), или ссылка на такую пару, или простое определение операции сравнения на необходимой структуре данных или ссылке на неё.
- Для любого узла X выполняются свойства дерева поиска: key[left[X]] < key[X] ≤ key[right[X]], то есть ключи данных родительского узла больше ключей данных левого сына и нестрого меньше ключей данных правого.

Основные операции

По сути, двоичное дерево поиска — это структура данных, способная хранить таблицу пар (key, value) и поддерживающая три операции: FIND, INSERT, REMOVE. Кроме того, интерфейс двоичного дерева включает ещё три дополнительных операции обхода узлов дерева: INFIX_TRAVERSE, PREFIX_TRAVERSE и POSTFIX_TRAVERSE. Первая из них позволяет обойти узлы дерева в порядке неубывания ключей.

Поиск элемента (FIND)

Дано: дерево T и ключ K.

Задача: проверить, есть ли узел с ключом K в дереве T, и если да, то вернуть ссылку на этот узел.

Алгоритм:

- Если дерево пусто, сообщить, что узел не найден, и остановиться.
- Иначе сравнить K со значением ключа корневого узла X.
    - Если K=X, выдать ссылку на этот узел и остановиться.
    - Если K>X, рекурсивно искать ключ K в правом поддереве T.
    - Если K<X, рекурсивно искать ключ K в левом поддереве T.

Добавление элемента (INSERT)

Дано: дерево T и пара (K, V).

Задача: вставить пару (K, V) в дерево T (при совпадении K, заменить V).

Алгоритм:

- Если дерево пусто, заменить его на дерево с одним корневым узлом ((K, V), null, null) и остановиться.

- Иначе сравнить K с ключом корневого узла X.
    - Если K>X, рекурсивно добавить (K, V) в правое поддерево T.
    - Если K<X, рекурсивно добавить (K, V) в левое поддерево T.
    - Если K=X, заменить V текущего узла новым значением.

Удаление узла (REMOVE)

Дано: дерево T с корнем n и ключом K.

Задача: удалить из дерева T узел с ключом K (если такой есть).

Алгоритм:

- Если дерево T пусто, остановиться;
- Иначе сравнить K с ключом X корневого узла n.
    - Если K>X, рекурсивно удалить K из правого поддерева T;
    - Если K<X, рекурсивно удалить K из левого поддерева T;
    - Если K=X, то необходимо рассмотреть три случая.
        - Если обоих детей нет, то удаляем текущий узел и обнуляем ссылку на него у родительского узла;
        - Если одного из детей нет, то значения полей ребёнка m ставим вместо соответствующих значений корневого узла, затирая его старые значения, и освобождаем память, занимаемую узлом m;
        - Если оба ребёнка присутствуют, то
            - Если левый узел m правого поддерева отсутствует (n->right->left)
                - Копируем из правого узла в удаляемый поля K, V и ссылку на правый узел правого потомка.
            - Иначе

- Возьмём самый левый узел m, правого поддерева n->right;
- Скопируем данные (кроме ссылок на дочерние элементы) из m в n;
- Рекурсивно удалим узел m.

Обход дерева (TRAVERSE)

Рассмотрим операцию прямого обхода.

Дано: дерево T и функция f

Задача: применить f ко всем узлам дерева T в порядке возрастания ключей

Алгоритм:

- Если дерево пусто, остановиться.
- Иначе
  - Рекурсивно обойти левое поддерево T.
  - Применить функцию f к корневому узлу.
  - Рекурсивно обойти правое поддерево T.

В простейшем случае функция f может выводить значение пары (K, V).

Практическая часть

Необходимо реализовать обобщенное бинарное дерево поиска. Использовать рекурсивные алгоритмы для реализации основных операций над деревом.

1) Создать класс `Node<T>` для хранения информации об узле дерева. Для удобства выполнения операции сравнения узлов можно добавить ограничение `where T : IComparable<T>`.

2) Создать класс `Tree<T>`. Класс должен реализовать интерфейс `IEnumerable<T>`.

3) Реализовать в классе дерева основные операции:

   a. `Find (T element)`

   b. `Insert (T element)`

   c. `Remove (T element)`

   d. `GetEnumerator()` – для выполнения прямого обхода

4) Продемонстрировать работу с деревьями на примерах `Tree<int>` и `Tree<Item>` (из ПР 7).

Содержание отчета

1. Титульный лист
2. Цель работы
3. Задание
4. Код программы
5. Результат выполнения