Алгоритмы и структуры данных

Лекция 11. Бинарные деревья поиска (BST)

Антон Штанюк (к.т.н, доцент)

13 мая 2021 г.

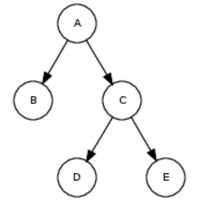
Нижегородский государственный технический университет им. Р.Е. Алексеева
Институт радиоэлектроники информационных технологий
Кафедра "Компьютерные технологии в проектировании и производстве"

Содержание

Общие сведения о деревьях

Общие сведения о деревьях

Дерево (англ. Tree) - иерархическая структура данных, состоящая из узлов, расположенных на уровнях и соединённых ветвями.



Основное преимущество дерева перед списком: скорость доступа к информации. Если в списке доступ осуществляется в среднем за $O(n)$, то в бинарном дереве: $O(log_2(n))$

Общие сведения о деревьях

Рассмотрим пример.

Общие сведения о деревьях

Допустим, у нас есть база данных сотрудников организации и данные в ней связаны друг с другом в виде дерева. Принцип, по которому данные расположены в дереве следующий: каждый узел разделяет фамилии по алфавитному принципу, то есть содержимое в левом поддереве находится по списку выше содержимого правого поддерева.

Проверить наличие среди сотрудников человека по фамилии ”Задорожный”. При организации записей в виде массива или списка, нужно выполнить 7 сравнений, прежде чем убедиться, что такого человека нет. Если организовать данные в виде дерева, то количество сравнений уменьшится до 3, по количеству уровней в дереве.

Общие сведения о деревьях

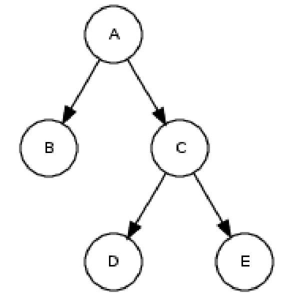
Древовидная структура характеризуется множеством узлов, происходящих из единственного начального узла, называемого **корнем**. Сыновья узла и сыновья сыновей называются **потомками** узла, а родители и прародители - **предками**. Каждый некорневой узел имеет только одного родителя и каждый родитель имеет 0 и более сыновей. Узел, не имеющий детей, называется **листом**.

Каждый узел дерева является корнем **поддерева**, которое определяется данным узлом и всеми потомками этого узла.

Прохождение от родительского узла к его дочернему узлу и к другим потомкам осуществляется вдоль пути. Тот факт, что каждый некорневой узел имеет единственного родителя, гарантирует, что существует единственный путь из любого узла к его потомкам. Путь от корня к узлу дает меру, называемому уровнем узла. Уровень есть длина пути от корня к этому узлу.

Глубина (высота) дерева есть максимальный уровень любого его узла или длина самого длинного пути от корня до узла.

Общие сведения о деревьях



Для дерева, изображённого на рисунке: A - корневой узел (нулевой уровень). B,C - потомки первого уровня. D,E - потомки второго уровня. B,D,E - листы. Глубина дерева - 2.

Классификация деревьев

- по максимальному количеству потомков у одного узла:
  - бинарные (2 потомка);
  - тернарные (3 потомка);
  - М-арные (М-потомков);
- по структуре:
  - симметричные;
  - сбалансированные;
  - несбаллансированные;
  - полные;
  - вырожденные;
- по характеру данных:
  - с упорядоченными данными;
  - с неупорядоченными данными;
- по виду:
  - бинарные с упорядоченными данными (поиска);
  - В-деревья;
  - Красно-чёрные (с балансировкой);

Бинарные деревья

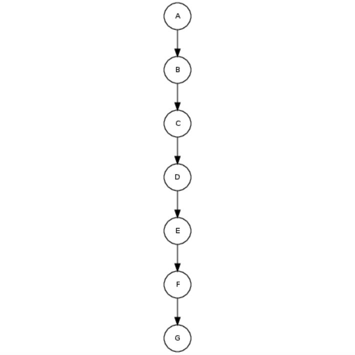
Бинарные деревья

Одним из наиболее распространенным видом деревьев являются **бинарные**, которые допускают разнообразные алгоритмы прохождения и эффективный доступ к элементам.

У каждого узла бинарного дерева может быть 0,1 или 2 потомка. По отношению к левому узлу применяется термин **левый** потомок, по отношению к узлу справа - **правый** потомок. Бинарное дерево является рекурсивной структурой. Каждый узел является корнем своего собственного поддерева.

Бинарные деревья

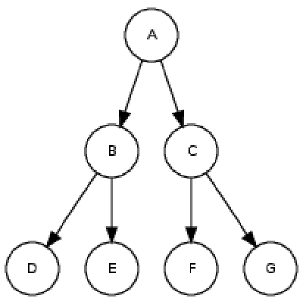
На любом уровне N бинарное дерево может содержать от 1 до 2^N узлов. **Вырожденным** будет называться такое дерево, у которого один лист и каждый узел имеет одного сына. Вырожденное бинарное дерево эквивалентно связанному списку.



Вырожденные деревья являются крайней мерой плотности размещения

Бинарные деревья

Другая крайность - **полные** бинарные деревья, в них каждый узел имеет обоих потомков.



Для бинарного дерева существует несколько операций, важнейшими из которых являются создание дерева, и обход его узлов. Обе процедуры рекурсивны.

Обход бинарного дерева

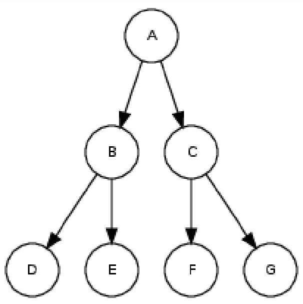
Существует несколько методов прохождения дерева для доступа к его элементам. К ним относятся **прямой**, **обратный** и **симметричный**.

При прохождении дерева используется рекурсия, поскольку каждый узел является корнем своего поддерева. Каждый алгоритм выполняет в узле три действия: заходит в узел, рекурсивно спускается по левому и по правому поддереву. Спуск прекращается при достижении пустого поддерева (нулевой указатель).

Обход бинарного дерева

- Порядок действий при прямом обходе:
  1. Обработка данных узла.
  2. Прохождение левого поддерева.
  3. Прохождение правого поддерева.
- Порядок действий при обратном обходе:
  1. Прохождение левого поддерева.
  2. Прохождение правого поддерева.
  3. Обработка данных узла.
- Порядок действий при симметричном обходе:
  1. Прохождение левого поддерева.
  2. Обработка данных узла.
  3. Прохождение правого поддерева.

Обход бинарного дерева



Последовательность обработки узлов:

1. прямой обход: ABDECFG
2. симметричный обход: DBEAFCG
3. обратный обход: DEBFGCA

Бинарные деревья с упорядоченными данными

При создании дерева часто используется свойство упорядоченности. Свойство упорядоченности. Пусть *x* - произвольная вершина бинарного дерева. Если вершина y находится в левом поддереве вершины *x*, то $y \to data \leq x \to data$. Если вершина y находится в правом поддереве вершины *x*, то $y \to data \geq x \to data$.

В этом случае симметричный обход дерева позволяет обрабатывать элементы в порядке возрастания их значений. Именно этот принцип используется в двух рассматриваемых ниже программах обработки символов и строк.

Программная реализация

Программная реализация

Структура бинарного дерева построена из узлов. Как и в связанном списке эти узлы содержат поля данных и указатели на другие узлы в коллекции. Узел дерева содержит поле данных и два поля с указателями, которые называются левым и правым указателями. Значение **nullptr** является признаком пустого поддерева.

Дерево, по сути, является **рекурсивной** структурой данных. В результате, множество операций будут реализованы через рекурсивные функции. У таких функций будет обязательный служебный параметр - адрес узла текущего уровня. Но для пользователя дерева такие параметры не нужны и их нужно задекорировать. Поэтому мы используем специальные функции-обертки для взаимодействия с пользователем, которые, в свою очередь, вызывают рекурсивные функции дерева.

Очередь на списке

```
template<typename T>
class BST
{
public:
    struct Node
    {
        T value;
        int count;
        Node *left;
        Node *right;
    };
```

Очередь на списке

```
private:
    Node* root;
    Node* addNode(Node *, T);
    void  printTree(Node*);
    int   depthTree(Node*);
    int   searchNode(Node*,T);
    void  delTree(Node*);
    Node* delNode(Node*,int);
public:
    BST();
    ~BST();
    void add(T);
    void print();
    int  depth();
    int  search(T);
    void clear();
    void remove(int);
};
```

Программная реализация

Перечислим основные рекурсивные функции для работы с деревом:

- **addNode()** - добавление нового узла в дерево.
- **printTree()** - симметричный обход и печать данных дерева.
- **depthTree()** - вычисление глубины (высоты) дерева.
- **searchNode()** - поиск узла в дереве.
- **delTree()** - удаление дерева.
- **delNode()** - удаление определенного узла в дереве.

Конструктор и деструктор класса

```
template<typename T>
BST<T>::BST():root(nullptr) {}

template<typename T>
BST<T>::~BST()
{
    if(root)
        delTree(root);
}
```

Добавление данных в дерево

Наиболее ответственная операция: добавление в дерево нового узла. Суть алгоритма добавления в следующем: мы начинаем работу с корня всего дерева. Если дерево пустое (корень нулевой), то тогда сразу создается новый узел и его адрес возвращается в качестве корня дерева. Если корень не пустой, то по результату сравнения вставляемых данных и данных в узле дерева мы идем либо в левое, либо в правое поддерево. Далее, либо мы достигаем листа и добавляем новый узел в качестве его потомка, либо находим узел, значение данных в котором совпадает с добавляемым значением. В таком случае добавлять узел не надо, а только увеличить счетчик в найденном узле.

Добавление данных в дерево

```
template<typename T>
typename BST<T>::Node* BST<T>::addNode(Node *root, T value) {
   if(root==nullptr) {
      root=new Node;
      root->value=value;
      root->count=1;
      root->left=root->right=nullptr;
   }
   else if(root->value>value) {
     root->left=addNode(root->left,value);
   }
   else if(root->value<value) {
     root->right=addNode(root->right,value);
   }
   else
    root->count++;
   return root;
}
```

Добавление данных в дерево

```
template<typename T>
void BST<T>::add(T value) {
   root=addNode(root,value);
}
```

Обход дерева

Обход и печать данных в узлах. Здесь реализован симметричный обход, когда мы сначала входим в левое поддерево, потом обрабатываем данные узла (печать), потом идет спуск в правое поддерево.

Обход дерева

```
template<typename T>
void BST<T>::printTree(Node* root) {
    if(root==nullptr)
        return;
    printTree(root->left);
    for(int i=0;i<root->count;i++)
        std::cout<<root->value<<"␣";
    printTree(root->right);
}

template<typename T>
void BST<T>::print() {
   printTree(root);
}
```

Удаление всех узлов (очистка дерева)

```
template<typename T>
void  BST<T>::delTree(Node* root)
{
   if(root==nullptr)
       return;
   else
   {
       delTree(root->left);
       delTree(root->right);
       delete root;
   }
}

template<typename T>
void BST<T>::clear()
{
    if(root)
    {
        delTree(root);
        root=nullptr;
    }
}
```

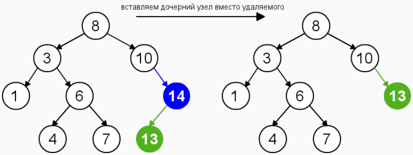
Удаление узла с определенными данными

Как ни странно, это одна из сложнейших операций над деревом. Проблема в том, что после удаления, не должен нарушаться принцип упорядоченности данных. Это заставляет нас рассмотреть четыре случая:

- У удаляемого узла нет потомков. Тогда мы можем освободить память, занимаемую узлом, а у его родителя выставить **nullptr** в указателе на потомка.

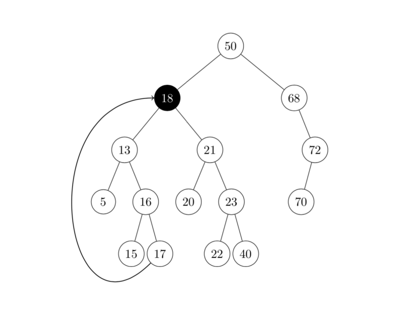
Удаление узла с определенными данными

- Удаляемый узел имеет одного потомка (левого или правого). В этом случае мы присваиваем адрес потомка указателю нашего родителя, вместо адреса текущего узла.



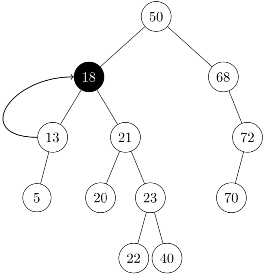
Удаление узла с определенными данными

- Удаляемый узел имеет двух потомков, причем у левого потомка есть свое правое поддерево. В этом случае нужно найти в этом правом поддереве наибольший элемент и вставить его вместо удаляемого узла.



Удаление узла с определенными данными

- Удаляемый узел имеет двух потомков, причем у левого потомка нет правого поддерева.



Удаление узла с определенными данными

```
template<typename T>
typename BST<T>::Node* BST<T>::delNode(typename BST<T>::Node* root,int
     value)
{
    Node* p,*v;
    // случай 0: поддерево пустое
    if(root==nullptr)
        return root;
    // ищем удаляемый узел либо в левом, либо в правом поддереве
    else if(value<root->value)
        root->left=delNode(root->left,value);
    else if(value>root->value)
        root->right=delNode(root->right,value);
    else
    {
```

Удаление узла с определенными данными

```
// случай 1,2: у узла есть только один потомок или узел — лист
 p=root;
 if(root->right==nullptr)
     root=root->left;
 else if(root->left==nullptr)
     root=root->right;
 else
 {
```

Удаление узла с определенными данными

```
// случай 3,4: у узла есть 2 потомка
v=root—>left; // будем просматривать левое поддерево
//  случай 3. У левого потомка есть правый потомок
if(v—>right)
{
    // ищем самый большой элт— в левом поддереве
    while(v—>right—>right)
        v=v—>right;
    // меняем найденное значение с корнем
    root—>value=v—>right—>value;
    root—>count=v—>right—>count;
    p=v—>right; // этот элт— мы удалим
    v—>right=v—>right—>left;
}
```

Удаление узла с определенными данными

```
        else // случай 4. У левого потомка удаляемого узла нет правого
поддерева или( нет сыновей)
        {
            root->value=v->value;
            root->count=v->count;
            p=v;
            root->left=root->left->left;
        }
    }
    delete p;
  }
  return root;
}
```

Удаление узла с определенными данными

```
template<typename T>
void BST<T>::remove(int value)
{
   if(root)
     root=delNode(root,value);
}
```

Список литературы

Список литературы I

- Кормен Т.,Лейзерсон Ч., Ривест Р.
  Алгоритмы: построение и анализ
  МЦНМО, Москва, 2000
- **Кормен Т., Лейзерсон Ч., Ривест Р., Штайн К. Алгоритмы: построение и анализ.**
  2-е изд. — М.: «Вильямс», 2006
- **Википедия**
  Алгоритм
  http://ru.wikipedia.org/wiki/Алгоритм
- **Википедия**
  Список алгоритмов
  http://ru.wikipedia.org/wiki/Список_алгоритмов
- **Традиция**
  Задача коммивояжёра
  http://traditio.ru/wiki/Задача

Список литературы ii

- **Википедия**
  NP-полная задача
  http://ru.wikipedia.org/wiki/NP-полная
- **Серджвик Р.**
  Фундаментальные алгоритмы на C++. Части 1-4
  Diasoft,2001
- **Седжвик Р.**
  Фундаментальные алгоритмы на C. Анализ/Структуры данных/Сортировка/Поиск
  СПб.: ДиаСофтЮП, 2003
- **Седжвик Р.**
  Фундаментальные алгоритмы на C. Алгоритмы на графах
  СПб.: ДиаСофтЮП, 2003
- **Ахо А., Хопкрофт Д., Ульман Д. Структуры данных и алгоритмы.**
  Издательский дом «Вильямс», 2000

Список литературы iii

- **Кнут Д.**
  Искусство программирования, том 1. Основные алгоритмы
  3-е изд. — М.: «Вильямс», 2006
- **Кнут Д.**
  Искусство программирования, том 2. Получисленные методы
  3-е изд. — М.: «Вильямс», 2007
- **Кнут Д.**
  Искусство программирования, том 3. Сортировка и поиск
  2-е изд. — М.: «Вильямс», 2007
- **Кнут Д.**
  Искусство программирования, том 4, выпуск 3. Генерация всех сочетаний и разбиений
  М.: «Вильямс», 2007

Список литературы iv

- **Кнут Д.**
  Искусство программирования, том 4, выпуск 4. Генерация всех деревьев. История комбинаторной генерации
  М.: «Вильямс», 2007