acm.mipt.ru

олимпиады по программированию на Физтехе

Раздел «Язык Си» . CoffeBTree :

Поиск

Поиск

Раздел «Язык Си»

Главная
Зачем учить С?
Определения
Инструменты:
Поиск
Изменения
Index
Статистика

Разделы

Информация
Алгоритмы
Язык Си
Язык Ruby
Язык Ассемблера
El Judge
Парадигмы
Образование
Сети
Objective C

Logon>>

Бинарное дерево (поиска)

Бинарным деревом называют структуру данных, состоящую из узлов, каждый из которых может иметь от 0 до 2 детей - узлов уровнем ниже (различают правого и левого ребенка).

Дерево растет сверху вниз.

Вверху находится корень (программисты ничего не понимаю в сельском хозяйстве).

Узел без детей называют листом. Листья находятся внизу дерева.

Рисунок для пояснения терминологии: корень, листья, левый ребенок, правый ребенок, уровень.

Построение дерева

- Если узла нет, добавляем узел.
- Если число меньше текущего узла, добавляем число в поддерево левее.
- Если число больше текущего узла, добавляем число в поддерево правее.

Построение бинарного дерева поиска

Построим дерево, добавляя в него числа 7, 3, 2, 1, 9, 5, 4, 6, 8

- Сначала дерево пустое
- Число 7
  - стало корнем дерева
- Число 3
  - корень занят
  - 3 < 7, значит добавляем в левое поддерево.
  - левого ребенка у узла 7 нет, так что 3 стало левым ребенком.
- Число 2
  - корень занят
  - 2 < 7, значит добавляем в левое поддерево.
  - левый ребенок у узла 7 уже есть, так что считаем этот узел 3 корнем поддерева и продолжаем добавлять.
  - 2 < 3, значит добавляем в левое поддерево узла 3.
  - левого ребенка у узла 3 нет, так что 2 стало левым ребенком.
- Число 1
  - аналогично стало левым ребенком узла 2
- Число 9
  - корень занят
  - 9 > 7, значит добавляем в правое поддерево.
  - правого ребенка у узла 7 нет, так что 9 стало правым ребенком.
- Число 5
  - корень занят
  - 5 < 7, значит добавляем в левое поддерево.
  - левый ребенок у узла 7 уже есть, так что считаем этот узел 3 корнем поддерева и продолжаем добавлять.
  - 5 > 3, значит добавляем в правое поддерево.
  - правого ребенка у узла 3 нет, так что 5 стало правым ребенком.
- аналогично добавляем числа 4, 6, 8 и получаем дерево

Объявление структур

Структурная единица дерева - его узел. В узле есть даные (поле data), указатель на левого и правого ребенка. Если ребенка нет, то вместо ребенка указываем на NULL.

```
typedef int Data;
struct Node {
    Data data;              // данные в узле
    struct Node * left;     // указатель на левого ребенка или NULL
    struct Node * right;    // указатель на правого ребенка или NULL
};
```

Печать дерева

Разберемся, как печатать дерево.

Пусть у нас есть пустое дерево:

```
struct Node * tree = NULL;    // tree - указатель на пустое дерево
```

В пустом дереве нет никаких элементов и ничего печатать не нужно.

Если дерево состоит ровно из 1 элемента, то надо его напечатать.

```
struct Node a = {7, NULL, NULL};
struct Node * tree = &a;
```

Нужно напечатать только содержимое узла a, то есть:

```
printf("%d ", tree->data);   // печать содержимого узла
```

Если в дереве в корне лежит число 7 и есть другие узлы, то нужно:

- сначала напечать числа, меньшие 7 (они все лежат в левом поддереве);
- напечатать содержимое корня (число 7);
- потом напечатать все числа, большие 7 (они лежат в правом поддереве).

Числа, лежащие в левом поддереве узла tree, пусть полностью печатает функция tree_print (это ее предназначение). Корнем левого поддерева будет узел tree->left.

Объединим все в один код:

```
если дерева нет,
    ничего делать не нужно
иначе
    напечатать левое поддерево (1, 2, 3, 4, 5, 6)
    напечатать значение в корне (7)
    напечатать правое поддерево (8, 9)
```

Запишем это на языке C:

```
void tree_print(struct Node * tree) {
    if (tree == NULL)
          // ничего не делать
    else {
        tree_print(tree->left);     // печатаем левое поддерево (меньшие числа)
        printf("%d ", tree->data);  // печатаем корень
        tree_print(tree->right);     // печатаем правое поддерево (большие числа)
    }
}
```

Выглядит коряво и не компилируется в части "дальше ничего не делать". Перепишем то же самое в другой форме:

```
void tree_print(struct Node * tree) {
    if (tree == NULL)
          return;   // дальше ничего не делать
    // "после return жизни нет" - далее код
    tree_print(tree->left);     // печатаем левое поддерево (меньшие числа)
    printf("%d ", tree->data);  // печатаем корень
    tree_print(tree->right);     // печатаем правое поддерево (большие числа)
}
```

Все остальные функции будут похожи на tree_print – они будут тоже рекурсивно вызывать те же функции для левых и правых детей.

Добавление узла в дерево

Пусть дерево пустое. В него хотим добавить число 7. Напишем это отдельным кодом:

```
struct Node * tree = NULL;      // дерево пустое, корень указывает на NULL

// добавляем в пустое дерево число 7
struct Node * new_node = malloc(sizeof(struct Node));
new_node -> data = 7;
new_node->left = new_node->right = NULL;
tree = new_node;
```

Напишем функцию tree_add так, чтобы при добавлении числа 7 в пустое дерево был выполнен именно этот код. Вызов функции будет таким:

```
struct Node * tree = NULL;
tree = tree_add(tree, 7);
```

Значит, функция tree_add должна получать параметры "указатель на корень поддерева" и "число для добавления", а возвращать "указатель на новый узел".

```
struct Node * tree_add(struct Node * tree, Data d) {
    if (tree == NULL) {
        struct Node * new_node = malloc(sizeof(struct Node));
        new_node -> data = d;
        new_node->left = new_node->right = NULL;
        return new_node;
    }
    // прочий код (дерево не пусто)
    ...
}
```

Пусть дерево состоит только из узла 7. Добавим в него число 3. Новое число меньше и должно быть добавлено как левый ребенок корня.

Это должно делать следующее:

```
if (3 < 7)
    tree->left = tree_add(NULL, 3);
```

Заметим, что до того, как мы вызывали tree_add, значение tree->left = NULL. Перепишем эту строку:

```
if (3 < tree->data)
    tree->left = tree_add(tree->left, 3);
```

Аналогично добавим число 2.

Если число меньше значения в узле (2 < 7), то 2 нужно добавлять в левое поддерево (если получится - левым ребенком). Левый ребенок уже занят? (Ссылается на узел с числом 3) Тогда рассмотрим левое поддерево (корень - 3) и добавим в него число 2.

Аналогично будем поступать, если наши числа будут больше значения в узле - будем добавлять их в правое поддерево.

Запишем алгоритм в виде функции.

```
struct Node * tree_add(struct Node * tree, Data d) {
    if (tree == NULL) {
        struct Node * new_node = malloc(sizeof(struct Node));
        new_node -> data = d;
        new_node->left = new_node->right = NULL;
        return new_node;
    }
    // прочий код (дерево не пусто)
    if (d < tree->data)
        tree->left = tree_add(tree->left, d);
    else if (d > tree->data)
        tree->right = tree_add(tree->right, d);
    return tree;   // чтобы не сломать значение переменной,
                   // указывающей на корень дерева
                   // при вызове tree = tree_add(3);
}
int main() {
    struct Node * tree = NULL;
    tree = tree_add(tree, 7);   // здесь значение tree изменится с NULL на указатель на корневой узел
    tree = tree_add(tree, 3);   // здесь значение tree не изменяется
    tree = tree_add(tree, 2);
    tree = tree_add(tree, 1);
    tree = tree_add(tree, 9);
    tree = tree_add(tree, 5);
    tree = tree_add(tree, 4);
    tree = tree_add(tree, 6);
    tree = tree_add(tree, 8);
    tree_print(tree); // напечатает 1 2 3 4 5 6 7 8 9
    // здесь нужно освобождать память
```

Остальные функции работы с деревом напишите самостоятельно.

Печать дерева

Построим вручную дерево, которое показано на рисунке.

Будем демонстрировать работу дерева на основе целых чисел.

Для описания узла объявим структуру Node.

```
#include <stdio.h>
#include <stdlib.h>

typedef int Data;

struct Node {
   Data data;             // данные в узле
   struct Node * left;       // указатель на левого ребенка
   struct Node * right;       // указатель на правого ребенка
};
```

Сделаем дерево из чисел 7, 3, 2, 9 руками.

```
void test0() {
   struct Node a, b, c, d, e, f, g, h, p;
   a.data = 7;  a.left = &b;   a.right = &d;
   b.data = 3;  b.left = &c;   b.right = NULL;
   c.data = 2;  c.left = NULL;   c.right = NULL;
   d.data = 9;  d.left = NULL;   d.right = NULL;

   tree_print(&a);
}
```

Подумаем, как можно напечатать дерево, если есть только указатель на его корень.

- Если дерева нет tree=NULL, то печатать ничего не нужно.
- Напечатать меньшие числа (те, что лежат в левом узле и ниже, то есть в левом поддереве).
- Напечатать число в узле.
- Напечатать большие числа (те, что лежат в правом узле и ниже, то есть в правом поддереве).

```
void tree_print (struct Node * tree) {
   // Если дерева нет, то печатать ничего не нужно
   if (tree == NULL)
      return;

   tree_print(tree->left);      // печатаем меньшие числа - левое поддерево
   printf("%d ", tree->data);   // печатаем само число в узле
   tree_print(tree->right);   // печатаем большие числа - правое поддерево
}
```

Проверим, что построенное дерево печатается верно: 2 3 5 9.

Добавление узла в дерево

Добавляем число в дерево следующим образом:

- Если узла нет (дерево пустое, NULL), сделаем узел с этим числом и он будет вместо NULL.
- Если узел есть и число меньше, попробуем его добавить на место левого ребенка.
- Если узел есть и число больше, попробуем его добавить на место правого ребенка.

```
struct Node * tree_add(struct Node * tree, Data d) {
   if (tree == NULL) {
      struct Node * new_node = malloc(sizeof(struct Node));
      new_node->data = d;
      new_node->left = NULL;
      new_node->right = NULL;
      return new_node;
   }
   if (d < tree->data) {
      tree->left = tree_add(tree, d);
   else if (d > tree->data) {
      tree->right = tree_add(tree, d);
   }
   return tree;
}
```

-- TatyanaDerbysheva - 18 Feb 2019