Работа со структурами данных в языках Си и Python: Часть 9. Красно-черные деревья

Сергей Яковлев
Консультант
независимый специалист

07.06.2011

В статье рассматриваются характеристики красно-черных деревьев (еще одной разновидности сбалансированных двоичных деревьев) и алгоритмы для работы с ними, реализованные на языке Си.

Введение

В данной статье рассматривается разновидность бинарных поисковых деревьев - красно-черные деревья, которые также относятся к сбалансированным деревьям. Такие деревья используются для решения самых разных задач, например, в одной из реализаций планировщика ядра ОС Linux (completely fair scheduler) или для создания ассоциативных массивов. В статье, как обычно, будут разбираться особенности реализации красно-черных деревьев и алгоритмов для работы с ними.

Красно-черные деревья

Красно-черное дерево (англ. red-black tree) - это еще одна форма сбалансированного бинарного поискового дерева. Впервые оно было представлено в 1972 году как еще одна разновидность сбалансированного бинарного дерева. Время поиска, вставки или удаления узла для красно-черного дерева является логарифмической функцией от числа узлов.

Данный тип деревьев отличается от других реализаций следующими свойствами:

- каждый узел ассоциируется с определенным цветом - красным или черным;
- корневой узел может быть любого цвета;
- красные узлы могут иметь только черные дочерние узлы;
- все пути от узла до любого листа, расположенного ниже в дереве, содержат одно и то же количество черных узлов.

Высота красно-черного дерева, состоящего из **N** узлов, лежит в диапазоне от двоичного логарифма `log(N+1)` до `2 * log(N+1)`.

В листинге 1 приведены структуры на языке Си, описывающие узлы красно-черного и AVL-деревьев.

© Copyright IBM Corporation 2011

Листинг 1. Исходный код узлов, использующихся в различных типах BST-деревьев

```
/* структура, описывающая узел красно-черного дерева */
struct rb_node
{
 int red;
 int data;
 struct rb_node *link[2];
};

/* структура, описывающая узел AVL-дерева */
struct avl_node
{
 struct avl_node * link[2];
 int data;
 short bal;
};
```

Как можно заметить, структура узла AVL-дерева очень похожа на структуру, использующуюся для хранения информации об узле красно-черного дерева. Только коэффициент сбалансированности **bal** в красно-черном дереве заменен на переменную **red**, определяющую цвет узла (красный или черный). Но функции вставки/удаления узлов для красно-черного дерева значительно отличаются от аналогичных операций для AVL-дерева и должны быть рассмотрены отдельно.

Также для работы с красно-черным деревом потребуется вспомогательная структура из листинга 2.

Листинг 2. Структура, описывающая красно-черное дерево

```
struct rb_tree
{
 struct rb_node *root; // указатель на корневой узел
 int count; // количество узлов в дереве
};
```

Вставка узла в красно-черное дерево

При вставке нового узла в цветное дерево (другое название красно-черного дерева) для него изначально устанавливается красный цвет. Затем для добавляемого элемента выполняется поиск родительского узла и проверка его цвета. Если цвет родителя - черный, то основной критерий цветного дерева сохраняется. Если же родитель - красного цвета, то выполняется итеративная (нерекурсивная) балансировка дерева. В худшем случае время вставки может достичь значения логарифма от числа узлов в красном дереве.

Для упрощения алгоритма предполагается, что листья (узлы, не имеющие потомков) имеют черный цвет. В листинге 3 приведен исходный код функции для определения цвета узла.

Листинг 3. Функция для определения цвета узла

```
int is_red ( struct rb_node *node )
{
 return node != NULL && node->red == 1;
}
```

Для ротации узлов в дереве будут использоваться функции, представленные в листинге 4. Первая функция меняет местами два узла, вторая функция выполняет два таких обмена:

Листинг 4. Функции для ротации узлов в красно-черном дереве

```
/* функция для однократного поворота узла */
struct rb_node *rb_single ( struct rb_node *root, int dir )
{
 struct rb_node *save = root->link[!dir];

 root->link[!dir] = save->link[dir];
 save->link[dir] = root;

 root->red = 1;
 save->red = 0;

 return save;
}

/* функция для двукратного поворота узла */
struct rb_node *rb_double ( struct rb_node *root, int dir )
{
 root->link[!dir] = rb_single ( root->link[!dir], !dir );
 return rb_single ( root, dir );
}
```

В листинге 5 приведен исходный код функции для создания нового узла.

Листинг 5. Функция для создания нового узла

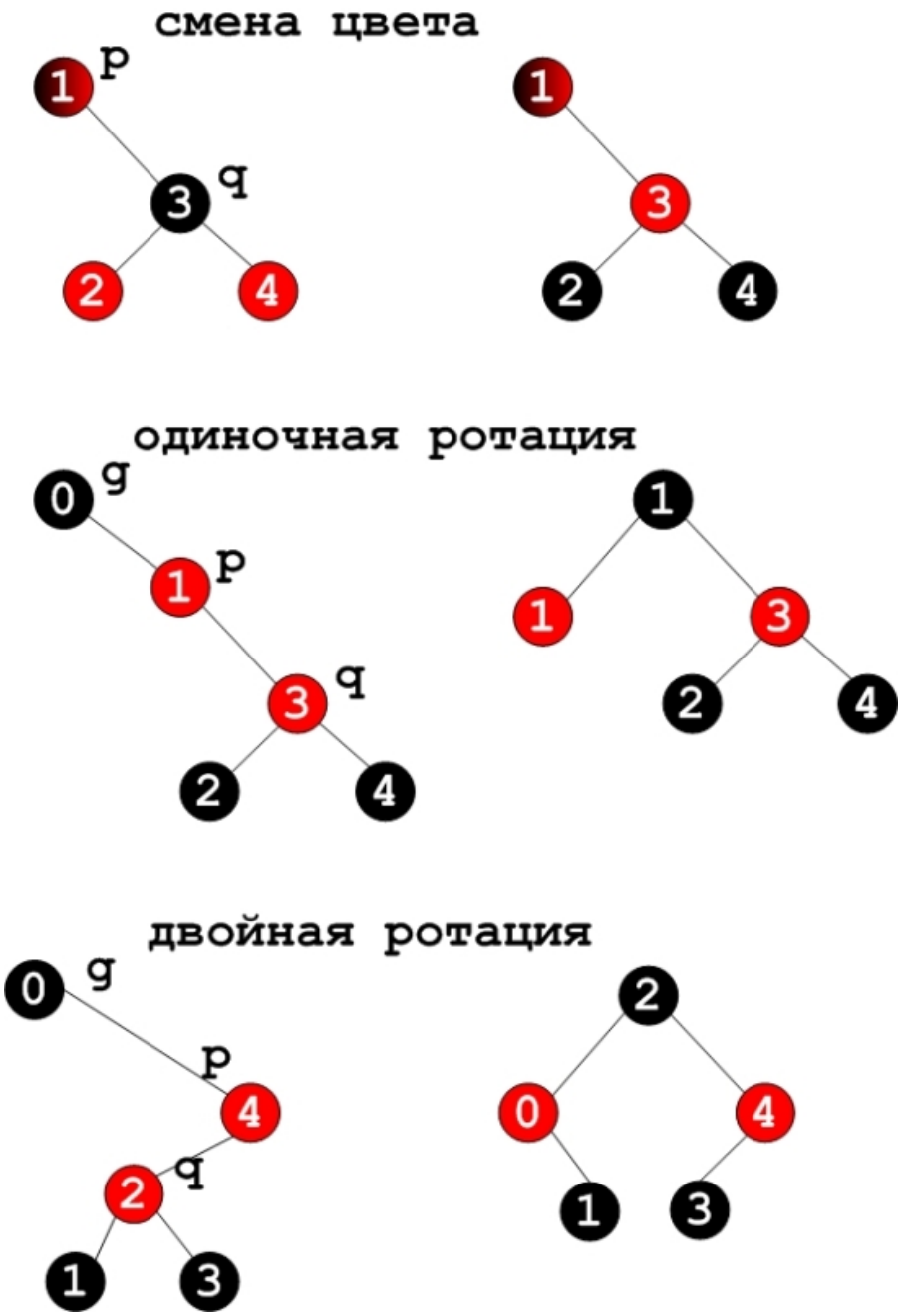
```
struct rb_node *make_node ( int data )
{
 struct rb_node *rn = malloc ( sizeof *rn );

 if ( rn != NULL ) {
  rn->data = data;
  rn->red = 1; /* –инициализация красным цветом */
  rn->link[0] = NULL;
  rn->link[1] = NULL;
 }
 return rn;
}
```

При вставке узла в цветное дерево не требуется прибегать к рекурсии, так как это можно сделать за один проход. В общем случае при вставке нового узла возможны три варианта.

1. происходит изменение цвета;
2. требуется сделать один поворот;
3. требуется сделать двойной поворот.

Для этого в памяти кроме текущего узла нужно хранить еще три уровня дерева: «родителя», «деда» и «прадеда» текущего узла.

Рисунок 1. Вставка узла в красно-черное дерево.



На рисунке 1 в первом варианте у добавляемого узла (**q**) проверяются дочерние узлы. Если дочерние узлы - красного цвета, а добавляемый узел – черного, то выполняется смена цветов. Необходимо также проверить на совпадение цвета 2 узла (второй и третий варианты) - добавляемый узел (**q**) и его родителя (**p**). Если они оба красного цвета, то выполняется одиночная либо двойная ротация.

Листинг 6. Функция для вставки узла в красно-черное дерево

```
int rb_insert ( struct rb_tree *tree, int data )
 {
```

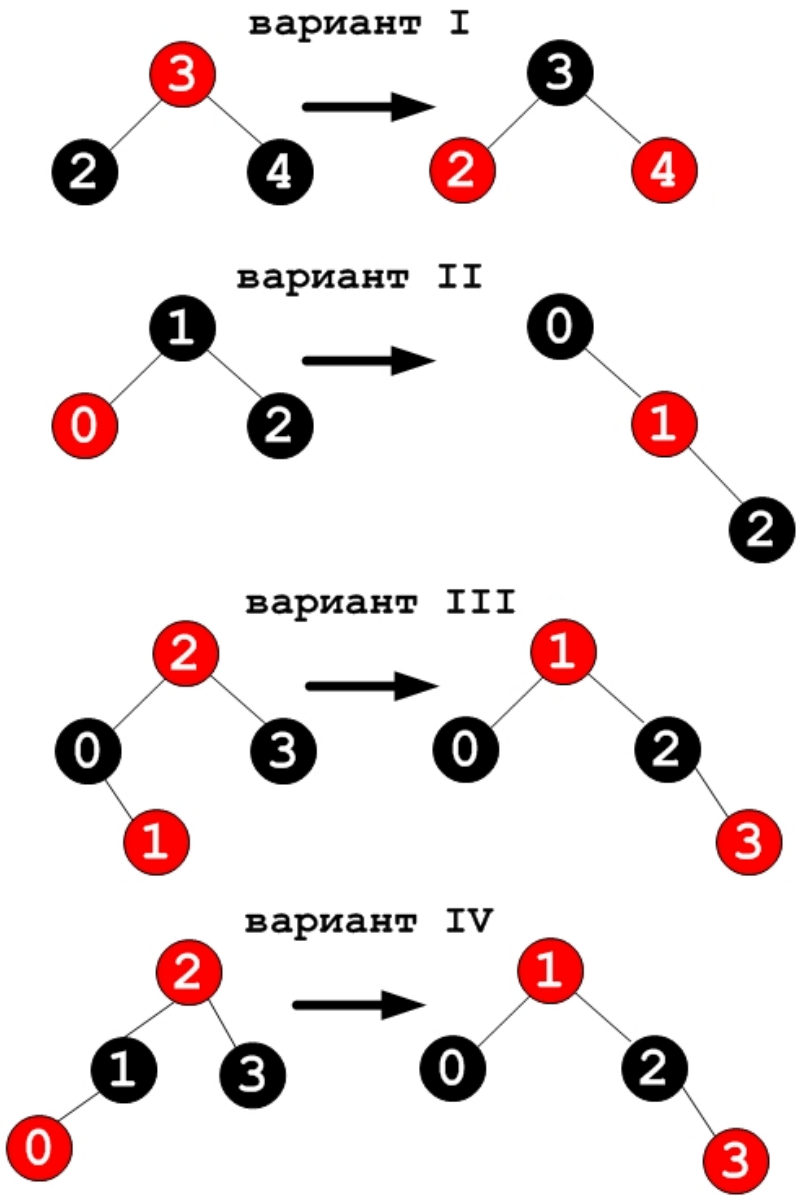
```
  /* если добавляемый элемент оказывается первым – то ничего делать не нужно*/
  if ( tree->root == NULL ) {
    tree->root = make_node ( data );
    if ( tree->root == NULL )
      return 0;
  }
  else {
    struct rb_node head = {0}; /* временный корень дерева*/
    struct rb_node *g, *t;     /* дедушка и родитель */
    struct rb_node *p, *q;     /* родитель и итератор */
    int dir = 0, last;

    /* вспомогательные переменные */
    t = &head;
    g = p = NULL;
    q = t->link[1] = tree->root;

    /* основной цикл прохода по дереву */
    for ( ; ; )
    {
      if ( q == NULL ) {
        /* вставка ноды */
        p->link[dir] = q = make_node ( data );
        tree->count ++ ;
        if ( q == NULL )
          return 0;
      }
      else if ( is_red ( q->link[0] ) && is_red ( q->link[1] ) )
      {
        /* смена цвета */
        q->red = 1;
        q->link[0]->red = 0;
        q->link[1]->red = 0;
      }
       /* совпадение 2-х красных цветов */
      if ( is_red ( q ) && is_red ( p ) )
      {
        int dir2 = t->link[1] == g;

        if ( q == p->link[last] )
          t->link[dir2] = rb_single ( g, !last );
        else
          t->link[dir2] = rb_double ( g, !last );
      }

      /* такой узел в дереве уже есть  - выход из функции*/
      if ( q->data == data )
      {
        break;
      }

      last = dir;
      dir = q->data < data;

      if ( g != NULL )
        t = g;
      g = p, p = q;
      q = q->link[dir];
    }

    /* обновить указатель на корень дерева*/
    tree->root = head.link[1];
  }
  /* сделать корень дерева черным */
  tree->root->red = 0;

  return 1;
```

```
}
```

Удаление узла из красно-черного дерева

При удалении узла из цветного дерева цвет родительского узла не изменяется. Удаление красного узла не влечет никаких последствий, коллизию может вызвать только удаление узла черного цвета. Поэтому при удалении черного узла для обеспечения целостности дерева необходимо использовать различные операции: простую смену цвета (если это возможно) или одну или несколько ротаций. На рисунке 2 представлены 4 ситуации, возможных при удалении черного узла.

Рисунок 2. Удаление узла из красно-черного дерева



Если при удалении черного узла, его «брат» (узел, находящийся на том же уровне, что и удаляемый) и все их четыре потомка имеют черный цвет (это вариант I на рисунке 2), то

выполняется изменение цвета. Если «брат» удаляемого узла окрашен в красный цвет, то производится ротация, изображенная на варианте II. Если удаляемый узел и его «брат» черного цвета, а правый потомок «брата» - красного, то выполняется двойная ротация, изображенная на варианте 4. Если левый потомок «брата» красного, то выполняется одиночная ротация, как в пятом варианте.

Листинг 7. Функция для удаления узла из красно-черного дерева

```
int br_remove ( struct rb_tree *tree, int data )
 {
   if ( tree->root != NULL )
   {
     struct rb_node head = {0}; /* временный указатель на корень дерева */
     struct rb_node *q, *p, *g; /* вспомогательные переменные */
     struct rb_node *f = NULL;  /* узел, подлежащий удалению */
     int dir = 1;

     /* инициализация вспомогательных переменных */
     q = &head;
     g = p = NULL;
     q->link[1] = tree->root;

     /* поиск и удаление */
     while ( q->link[dir] != NULL ) {
       int last = dir;

       /* сохранение иерархии узлов во временные переменные */
       g = p, p = q;
       q = q->link[dir];
       dir = q->data < data;

       /* сохранение найденного узла */
       if ( q->data == data )
         f = q;

       if ( !is_red ( q ) && !is_red ( q->link[dir] ) ) {
         if ( is_red ( q->link[!dir] ) )
           p = p->link[last] = rb_single ( q, dir );
         else if ( !is_red ( q->link[!dir] ) ) {
           struct rb_node *s = p->link[!last];


           if ( s != NULL ) {
             if ( !is_red ( s->link[!last] ) && !is_red ( s->link[last] ) ) {
               /* смена цвета узлов */
               p->red = 0;
               s->red = 1;
               q->red = 1;
             }
             else {
               int dir2 = g->link[1] == p;

               if ( is_red ( s->link[last] ) )
                 g->link[dir2] = rb_double ( p, last );
               else if ( is_red ( s->link[!last] ) )
                 g->link[dir2] = rb_single ( p, last );

               /* корректировка цвета узлов */
               q->red = g->link[dir2]->red = 1;
               g->link[dir2]->link[0]->red = 0;
               g->link[dir2]->link[1]->red = 0;
             }
           }
         }
```

```
      }
    }

    /* удаление найденного узла */
    if ( f != NULL ) {
      f->data = q->data;
      p->link[p->link[1] == q] =
        q->link[q->link[0] == NULL];
      free ( q );
    }

    /* обновление указателя на корень дерева */
    tree->root = head.link[1];
    if ( tree->root != NULL )
      tree->root->red = 0;
  }

  return 1;
}
```

Сравнение AVL и красно-черных деревьев

У AVL и цветных деревьев есть как общие черты, так и отличия. Например, совпадает время, требующееся для вставки удаления/узлов в оба типа деревьев, которое в общем случае равно двоичному логарифму от общего числа узлов в дереве.

Для модификации обоих типов деревьев требуется выполнение дополнительных ротаций. Но вставка в AVL-дерево требует не более одной ротации, а удаление узла уже может потребовать большого количества ротаций, в общем случае, двоичный логарифм от числа узлов в дереве. Модификация красно-черных деревьев выполняется легче, так как вставка требует не более 2-х ротаций, а удаление - не более трех.

Также в случае, когда общее число узлов дерева] одинаково, максимальная высота AVL-дерева всегда будет меньше, чем максимальная высота красно-черного дерева. Высота красно-черного дерева может превышать высоту AVL-дерева в 1.38 раз, поэтому для выполнения поиска по красно-черному дереву требуется больше времени.

AVL-дерево должно хранить высоту в каждом узле, в то время как в узле красно-черного дерева хранится только дополнительный бит, определяющий цвет узла. Поэтому для хранения красно-черного дерева требуется меньше памяти, чем для AVL-дерева такого же размера.

Сравнение производительности AVL и красно-черных деревьев

В этом разделе сравнивается производительность вставки узлов в AVL-дерево и красно-черное дерево. Как уже было сказано, оба этих типа относятся к самобалансирующимся бинарным поисковым деревьям. Поэтому, если при вставке нарушается критерий сбалансированности (высота дерева становится неоптимальной, т.е. превышает минимально возможное значение), то выполняется автоматическая корректировка дерева.

В ходе сравнительного тестирования в дерево добавляются узлы, сгенерированные случайным образом, до тех пор, пока количество узлов в дереве не станет равно пяти

миллионам. Такое большое значение было выбрано специально, чтобы наглядно выявить разницу в скорости работы двух типов деревьев, но для более слабых или наоборот более мощных процессоров можно использовать другие значения.

Реализация красно-черного дерева на языке Си была представлена в этой статье, а реализацию AVL-дерева можно взять из предыдущей статьи. В листинге 8 представлена программа для проверки производительности AVL-дерева, а в листинге 9 – эта же программа, но уже для красно-черного дерева.

Листинг 8. Тестирование производительности AVL-дерева

```
int main()
{

  struct avl_tree * my_tree = tree_create();

  int res = 0;
  int i = 0 ;
  int rnd = 0;
  int count = 10000000;

  time_t start,time2;
  volatile long unsigned t;
  start = time(NULL);

  srand (time (NULL));
  for( i = 0; i < count; i++)
  {
    rnd = (rand() % count);
    res = avl_insert(my_tree, rnd);
    if(my_tree->count==5000000) break;
  }

 time2 = time(NULL);
 printf("\n затрачено %f секунд.\n", difftime(time2, start));

 printf("\n высота=%d количество узлов=%d\n",height(my_tree->root),my_tree->count);

 return 0 ;
}
```

Листинг 9. Тестирование производительности красно-черного дерева

```
int  main()
{

  int count = 10000000;
  int res = 0;
  int i = 0 ;
  int rnd = 0;

  struct rb_tree * my_tree = tree_create();

  time_t start,time2;
  volatile long unsigned t;
  start = time(NULL);

  srand (time (NULL));
  for( i = 0; i < count; i++)
  {
    rnd = (rand() % count);
```

```
    res = rb_insert ( my_tree, rnd );
    if(my_tree->count==5000000) break;
  }

  time2 = time(NULL);
  printf("\n затрачено %f секунд.\n", difftime(time2, start));

  printf("\n высота=%d количество узлов=%d\n",height(my_tree->root),my_tree->count);

  return 0 ;
}
```

В ходе тестирования были получены следующие результаты: одному и тому же компьютеру потребовалось 16 секунд на вставку 5 миллионов узлов в AVL-дерево и 20 секунд на вставку такого же количества узлов в красно-черное дерево. При этом высота AVL-дерева оказалась равной 27, а цветного дерева - 28.

Заключение

Таким образом, теория была подтверждена практикой: высота AVL-дерева оказалась чуть меньше высоты красно-черного дерева, что положительно сказалось на скорости вставки узлов в AVL-дерево. Используя представленные материалы, предлагается выполнить «обратное» тестирование на скорость удаления узлов из дерева, чтобы проверить утверждение, что удаление из AVL-дерева требует больше времени, нежели удаление из цветного дерева.

Об авторе

Сергей Яковлев

Яковлев Сергей — независимый разработчик с многолетним опытом прикладного и системного программирования; вносит вклад в развитие open-source на своем персональном сайте www.iakovlev.org. Консультант.

© Copyright IBM Corporation 2011
(www.ibm.com/legal/copytrade.shtml)
Торговые марки
(www.ibm.com/developerworks/ru/ibm/trademarks/)