**Анализ красно-черного дерева**

В этой структуре баланс достигается за счет поддержания раскраски вершин в два цвета, а именно красный и черный.

struct rb\_node//структура узла

{

int red;//красное

int data;//тип данных

struct rb\_node\* link[2];

};

struct rb\_tree//структура узла

{

struct rb\_node\* root; // указатель на корневой узел

int count; // количество узлов в дереве

};

Возьмем самый глубокий лист. Пусть он находится на глубине h. Из-за правила 1, как минимум половина вершин на пути из корня будет черными, то есть черная высота дерева будет не меньше h/2. Можно показать, что в таком дереве будет не менее 2(h/2)-1 черных вершин (так как у каждой черной вершины с черной глубиной k, если она не лист, должно быть как минимум два потомка с черной глубиной k+1). Тогда 2(h/2)-1 <= n или h <= 2\*log2(n+1).

int rb\_insert(struct rb\_tree\* tree, int data)

{

if (tree->root == NULL) //если не пустое

{

tree->root = make\_node(data);//добавляем элемент

if (tree->root == NULL)//если он первый то

return 0;//ничего

}

else //иначе

{

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;

for (; ;)//цикл для прохода по дереву

{

if (q == NULL)

{

// вставка узла

p->link[dir] = q = make\_node(data);

tree->count++;//увеличиваем количество узлов

if (q == NULL)

return 0;

}

else if (is\_red(q->link[0]) && is\_red(q->link[1]))//если левая и правая ссылки красные

{

// смена цвета

q->red = true;

q->link[0]->red = false;

q->link[1]->red = false;

}

if (is\_red(q) && is\_red(p))//если два цвета совпадают

{

int dir2 = t->link[1] == g;//кладем второе значение в переменную и запомниаем

if (q == p->link[last])

t->link[dir2] = rb\_single(g, !last);

else

t->link[dir2] = rb\_double(g, !last);

}

if (q->data == data)//если узел такой есть

{

break;//выходим из функции

}

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 = false;

return 1;

}

Как можно видеть в начале запускается бесконечный цикл for( ; ;). Но он не является бесконечным, т.к в общем случае будет идти вниз, а высота дерева, как сказано выше, есть величина h <= 2\*log2(n+1). Поэтому добавление будет осуществляться за O(h) или же за O(log(n)). Операции, которые выполняются за константное время нас не интересуют.

int br\_remove(struct rb\_tree\* tree, int data)

{

if (tree->root != NULL)//если не пустой то ходим бродим

{

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;//правая ссылка

// дополнительные переменные для удобства

q = &head;//сюда передаем голову

g = p = NULL;//это значение теперь пустое

q->link[1] = tree->root;//сюда сохраняем первое значение

// поиск и удаление

while (q->link[dir] != NULL) //если не пустое то идем вправо

{

int last = dir;

// сохранение узлов во временные переменные

g = p, p = q;

q = q->link[dir];

dir = q->data < data;

if (q->data == data)//если есть такое значение

f = q;//сохраняем узел который нашли

if (!is\_red(q) && !is\_red(q->link[dir]))

{

if (is\_red(q->link[!dir]))

p = p->link[last] = rb\_single(q, dir);//вращаем 1 раз

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])) {//не красные левая и правая ссылки,то

// меняем цвета узлов при перестановки

p->red = false;

s->red = true;

q->red = true;

}

else {

int dir2 = g->link[1] == p;

if (is\_red(s->link[last]))

g->link[dir2] = rb\_double(p, last);//двойная поворот узла

else if (is\_red(s->link[!last]))

g->link[dir2] = rb\_single(p, last);//одинарный поворот узла

// корректировка цвета узлов

q->red = g->link[dir2]->red = true;

g->link[dir2]->link[0]->red = false;

g->link[dir2]->link[1]->red = false;

}

}

}

}

}

// удаление найденного узла

if (f != NULL)//если узел не пустой

{

f->data = q->data;//переприсваиваем значения

p->link[p->link[1] == q] =

q->link[q->link[0] == NULL];//переприсваиваем ссылки

free(q);//освобождаем память

}

// передаем указатель на новый узел

tree->root = head.link[1];//теперь первый узел и есть новый

if (tree->root != NULL)//если первый не пустой

tree->root->red = false;

}

return 1;

}

Вывод дерева будет осуществляться симметрично на консоль

void Show(rb\_node\* node, char dir)//симметричный вывод дерева на консоль

{

if (!node) return;

Show(node->link[0], 'l');//направляемся в левую ветку

cout << dir << " " << node->data << " " << (node->red ? "red" : "black") << endl;//выводим и присваимваем узлам цвет ветки

Show(node->link[1], 'r');////направляемся в правую ветку

}