**Анализ АВЛ**

avl\* avl\_tree::balance(avl\* t)//балансировка дерева

{

int bal\_factor = difference(t);

if (bal\_factor > 1) //если не сбалансированно

{

if (difference(t->l) > 0)//если левая

t = ll\_rotat(t);//влево влево

else

t = lr\_rotat(t);//влево вправо

}

else if (bal\_factor < -1)//вращаем в другую сторону

{

if (difference(t->r) > 0)//если больше нуля

t = rl\_rotat(t);//вправо влево

else

t = rr\_rotat(t);//вправо вправо

}

return t;//возвращаемся

}

Рассмотрим операцию балансировки. Функции ll\_rotat, lr\_rotat, rl\_rotat, rr\_rotat будут выполняться за O(1), т.к в них просто переприсваиваются ссылки. Difference выполняется также за O(1), потому что в этой функции считается разница между двумя вершинами. Операции сравнения в самом методе выполняются также за константное время, поэтому общее время выполнения данного метода будет O(1).

АВЛ- дерево является сбалансированным деревом, поэтому его высота будет составлять log(n). Рассмотрим операцию добавления.

avl\* avl\_tree::insert(avl\* r, int v) {

if (r == NULL) {//если нет корня, то создаем

r = new avl;

r->d = v;

r->l = NULL;

r->r = NULL;

return r;

}

else if (v < r->d) {//если меньше

r->l = insert(r->l, v);//то идем влево

r = balance(r);//потом балансируем

}

else if (v >= r->d) {//если болоьше

r->r = insert(r->r, v);//то идем вправо

r = balance(r);//потом балансируем

} return r;

}

Как можно видеть, все операторы выполняются за O(1), но при проходе дерева алгоритм пройдет h узлов, а значит время будет составлять O(h) или же O(log(n)).

Аналогично и с операцией удаления. мы будем идти в левый правый угол, и тем самым пройдем h узлов, время будет составлять O(h) или же O(log(n)).

avl\* avl\_tree:: remove(avl \*p, int k) // удаление ключа k из дерева p

{

if (!p) return 0;

if (k < p->d)//если значение меньше

p->l = remove(p->l, k);//идем влево

else if (k > p->d)//иначе

p->r = remove(p->r, k);//идем вправо

else // если нашли

{

avl\* q = p->l;//сохраняем левую

avl \*r = p->r;//и правую ссылки

delete p;//удаляем p

if (!r) return q;

avl\* min = findmin(r);//ищем минимальный

min->r = removemin(r);//удаляем его

min->l = q;//перекидываем ссылочку

return balance(min);//балансируем

}

return balance(p);

}

Обход дерева будет осуществляться за O(n), так как будут пройдены все узлы.