Для красно-чёрного дерева реализовать операции: добавления в дерево вершины, удаления вершины, поиска вершины и желательно вывода дерева на экран :

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <stdarg.h>

int T; // \* Тип элемента для хранения \* /

#define compLT(a,b) (a < b)

#define compEQ(a,b) (a == b)

// \* Красно-черного дерева Описание \* /

enum { BLACK, RED } nodeColor;

struct Node\_ {

struct Node\_ \*left; // \* Левый потомок \* /

struct Node\_ \*right; /\* правый потомок \*/

struct Node\_ \*parent; /\* родители \*/

nodeColor color; /\* узел цвет (черный , красный) \*/

T data; /\* данные, сохраненные в узле \*/

} Node;

#define NIL &sentinel /\* все листья являются стражами \*/

Node sentinel = { NIL, NIL, 0, BLACK, 0};

Node \*root = NIL; /\* квадратный из красно-черного дерева \*/

void rotateLeft(Node \*x) {

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* вращать узел х налево \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

Node \*y = x->right;

/\* establish x->right link \*/

x->right = y->left;

if (y->left != NIL) y->left->parent = x;

/\* establish y->parent link \*/

if (y != NIL) y->parent = x->parent;

if (x->parent) {

if (x == x->parent->left)

x->parent->left = y;

else

x->parent->right = y;

} else {

root = y;

}

/\* link x and y \*/

y->left = x;

if (x != NIL) x->parent = y;

}

void rotateRight(Node \*x) {

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* вращать узел X направо \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

Node \*y = x->left;

/\* establish x->left link \*/

x->left = y->right;

if (y->right != NIL) y->right->parent = x;

/\* establish y->parent link \*/

if (y != NIL) y->parent = x->parent;

if (x->parent) {

if (x == x->parent->right)

x->parent->right = y;

else

x->parent->left = y;

} else {

root = y;

}

/\* ссылка х и у \*/

y->right = x;

if (x != NIL) x->parent = y;

}

void insertFixup(Node \*x) {

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

поддерживать красно-черного дерева баланс \*

\* После вставки узла X \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* проверить Красно-черные свойства \*/

while (x != root && x->parent->color == RED) {

/\* имеет место нарушение \*/

if (x->parent == x->parent->parent->left) {

Node \*y = x->parent->parent->right;

if (y->color == RED) {

/\* дядя RED \*/

x->parent->color = BLACK;

y->color = BLACK;

x->parent->parent->color = RED;

x = x->parent->parent;

} else {

/\* дядя BLACK \*/

if (x == x->parent->right) {

/\* сделать Xлевого ребенка \*/

x = x->parent;

rotateLeft(x);

}

/\* Перекрасить и вращать \*/

x->parent->color = BLACK;

x->parent->parent->color = RED;

rotateRight(x->parent->parent);

}

} else {

/\* зеркальное изображение выше код \*/

Node \*y = x->parent->parent->left;

if (y->color == RED) {

/\*Дядя RED \*/

x->parent->color = BLACK;

y->color = BLACK;

x->parent->parent->color = RED;

x = x->parent->parent;

} else {

/\* дядя BLACK \*/

if (x == x->parent->left) {

x = x->parent;

rotateRight(x);

}

x->parent->color = BLACK;

x->parent->parent->color = RED;

rotateLeft(x->parent->parent);

}

}

}

root->color = BLACK;

}

Node \*insertNode(T data) {

Node \*current, \*parent, \*x;

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* выделить узел для данных и вставьте его на дереве \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* найти, где принадлежит узел \*/

current = root;

parent = 0;

while (current != NIL) {

if (compEQ(data, current->data)) return (current);

parent = current;

current = compLT(data, current->data) ?

current->left : current->right;

}

/\* Установка нового узла \*/

if ((x = (Node \*) malloc (sizeof(\*x))) == 0) {

printf ("insufficient memory (insertNode)\n");

exit(1);

}

x->data = data;

x->parent = parent;

x->left = NIL;

x->right = NIL;

x->color = RED;

/\* Вставить узел в дереве \*/

if(parent) {

if(compLT(data, parent->data))

parent->left = x;

else

parent->right = x;

} else {

root = x;

}

insertFixup(x);

return(x);

}

void deleteFixup(Node \*x) {

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

поддерживать красно-черного дерева баланс \*

\* После удаления узла X

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

while (x != root && x->color == BLACK) {

if (x == x->parent->left) {

Node \*w = x->parent->right;

if (w->color == RED) {

w->color = BLACK;

x->parent->color = RED;

rotateLeft (x->parent);

w = x->parent->right;

}

if (w->left->color == BLACK && w->right->color == BLACK) {

w->color = RED;

x = x->parent;

} else {

if (w->right->color == BLACK) {

w->left->color = BLACK;

w->color = RED;

rotateRight (w);

w = x->parent->right;

}

w->color = x->parent->color;

x->parent->color = BLACK;

w->right->color = BLACK;

rotateLeft (x->parent);

x = root;

}

} else {

Node \*w = x->parent->left;

if (w->color == RED) {

w->color = BLACK;

x->parent->color = RED;

rotateRight (x->parent);

w = x->parent->left;

}

if (w->right->color == BLACK && w->left->color == BLACK) {

w->color = RED;

x = x->parent;

} else {

if (w->left->color == BLACK) {

w->right->color = BLACK;

w->color = RED;

rotateLeft (w);

w = x->parent->left;

}

w->color = x->parent->color;

x->parent->color = BLACK;

w->left->color = BLACK;

rotateRight (x->parent);

x = root;

}

}

}

x->color = BLACK;

}

void deleteNode(Node \*z) {

Node \*x, \*y;

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Удаление узла из Z \* дерево

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

if (!z || z == NIL) return;

if (z->left == NIL || z->right == NIL) {

/\* у имеет NIL узел в качестве дочернего \*/

y = z;

} else {

/\* найти дерево с преемникомNIL узел в качестве дочернего \*/

y = z->right;

while (y->left != NIL) y = y->left;

}

/\* X является единственным ребенком у автора \*/

if (y->left != NIL)

x = y->left;

else

x = y->right;

/\* Y удалить из родительской цепи \*/

x->parent = y->parent;

if (y->parent)

if (y == y->parent->left)

y->parent->left = x;

else

y->parent->right = x;

else

root = x;

if (y != z) z->data = y->data;

if (y->color == BLACK)

deleteFixup (x);

free (y);

}

Node \*findNode(T data) {

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* найти узел, содержащий данные \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

Node \*current = root;

while(current != NIL)

if(compEQ(data, current->data))

return (current);

else

current = compLT (data, current->data) ?

current->left : current->right;

return(0);

}

void main(int argc, char \*\*argv) {

int a, maxnum, ct;

Node \*t;

// \* Командной строки :

//Процесс записи 2000

maxnum = atoi(argv[1]);

for (ct = maxnum; ct; ct--) {

a = rand() % 9 + 1;

if ((t = findNode(a)) != NULL) {

deleteNode(t);

} else {

insertNode(a);

}

}

}