```
#include <iostream>

#include <vector>

using namespace std;




class node {

	//for simplicity, we again assume all numbers are distinct

public:

	int num_values;

	vector<int> value;  //it can contain 1 ... 3 values

	node * parent;

	vector<node *> child;  //child[i] ( i = 0 ... 3) is a pointer to child node i

	int child_state; //a node can have up to 4 child nodes: child 0, 1, 2, 3.  child_state i (0 ... 3)
means this node is child i of its parent

	int is_leaf; //1 if this is a leaf node; otherwise 0

	node() { num_values = 0; is_leaf = 1; child.assign(4, nullptr); parent = nullptr; value.assign(3, -
999); }

	void add_value(int k);  //add value to a node of less than 3 value

	void remove_value(int k); //remove value from a node of more than 1 value

};


void node::add_value(int k)

{

	if (num_values == 0)

	{

		value[0] = k;

	}

	else if (num_values == 1)

	{
```

```
                if (value[0] > k)
                {
                        value[1] = value[0];
                        value[0] = k;
                }
                else
                        value[1] = k;
        }
        else if (num_values == 2)
        {
                if (value[0] > k)
                {
                        value[2] = value[1];
                        value[1] = value[0];
                        value[0] = k;
                }
                else if (k > value[0] && k < value[1])
                {
                        value[2] = value[1];
                        value[1] = k;
                }
                else if (k > value[1])
                        value[2] = k;
                else
                        return;
        }
        num_values++;
}
```

```
void node::remove_value(int k)

{

        if (value[0] == k)

        {

                value[0] = value[1];

                value[1] = value[2];

                value[2] = -999;

        }

        else  if (value[1] == k)

        {

                value[1] = value[2];

                value[2] = -999;

        }

        else if (value[2] == k)

        {

                value[2] = -999;

        }

        num_values--;

}

class two34_tree {

public:

        node * root;

        two34_tree() { root = nullptr; }

        void add(int k);

        node * find(int k); //find a node to add value k; invoked by add

        void break_3_value_node(node * p); //to be invoked by find

        void remove(int k);

        node * find_1(int k); //find a node to replace k once k is removed; invokde by remove

        void expand_1_value_node(node *p); //to be invoked by find_1
        void fusion(node *p); //to be invoked by exapnd_1_value_node
        void rotation(node *p); ////to be invoked by exapnd_1_value_node
        void in_order_traversal(node * p);
};

void two34_tree::break_3_value_node(node *p)
{
        node * pnode = p;
        int val = pnode->value[1];
        int pnodechildstate = pnode->child_state;
        node*n2 = new node();
        bool itsroot = false;
        if (pnode == root)
                itsroot = true;
        n2->value[0] = pnode->value[0];
        pnode->value[0] = pnode->value[2];
        if (pnode->child[0] != nullptr)
        {
                n2->child[0] = pnode->child[0];
                n2->child[0]->child_state = 0;
                n2->child[0]->parent = n2;
                n2->is_leaf = 0;
        }
        else
        {
                n2->child[0] = nullptr;
                n2->is_leaf = 1;
        }
        if (pnode->child[1] != nullptr)
```

```
{
        n2->child[1] = pnode->child[1];
        n2->child[1]->child_state = 1;
        n2->child[1]->parent = n2;
        n2->is_leaf = 0;
}
else
{
        n2->child[1] = nullptr;
        n2->is_leaf = 1;
}

if (pnode->child[2] != nullptr)
{
        pnode->child[0] = pnode->child[2];
        pnode->child[0]->child_state = 0;
}
else
{
        pnode->child[0] = nullptr;
        pnode->is_leaf = 1;
}

if (pnode->child[3] != nullptr)
{
        pnode->child[1] = pnode->child[3];
        pnode->child[1]->child_state = 1;
}
else
```

```
{
        pnode->child[1] = nullptr;
        pnode->is_leaf = 1;
}

n2->num_values = 1;
pnode->num_values = 1;
pnode->child[2] = nullptr;
pnode->child[3] = nullptr;
pnode->value[1] = -999;
pnode->value[2] = -999;
n2->parent = pnode->parent;

if (itsroot == true)
{
        node * n1 = new node();
        n1->value[0] = val;
        n1->num_values++;
        n1->parent = nullptr;
        n1->child_state = 0;
        n1->is_leaf = 0;
        n1->child[0] = n2;
        n1->child[1] = pnode;
        n2->parent = n1;
        pnode->parent = n1;
        n2->child_state = 0;
        pnode->child_state = 1;
        root = n1;
        root->child_state = -1;
```

```
}
else
{
        if (pnodechildstate == 0)
        {
                pnode->parent->value[2] = pnode->parent->value[1];
                pnode->parent->value[1] = pnode->parent->value[0];
                pnode->parent->value[0] = val;
                pnode->parent->num_values++;

                if (pnode->parent->child[2] != nullptr)
                {
                        pnode->parent->child[3] = pnode->parent->child[2];
                        pnode->parent->child[3]->child_state = 3;
                }
                if (pnode->parent->child[1] != nullptr)
                {
                        pnode->parent->child[2] = pnode->parent->child[1];
                        pnode->parent->child[2]->child_state = 2;
                }
                pnode->parent->child[0] = n2;
                n2->child_state = 0;
                pnode->parent->child[1] = pnode;
                pnode->child_state = 1;
        }
        else if (pnodechildstate == 1)
        {
                pnode->parent->value[2] = pnode->parent->value[1];
                pnode->parent->value[1] = val;
                pnode->parent->num_values++;

                if (pnode->parent->child[2] != nullptr)
                {
                    pnode->parent->child[3] = pnode->parent->child[2];
                    pnode->parent->child[3]->child_state = 3;
                }
                pnode->parent->child[1] = n2;
                n2->child_state = 1;
                pnode->parent->child[2] = pnode;
                pnode->child_state = 2;
            }
            else if (pnodechildstate == 2)
            {
                pnode->parent->value[2] = val;
                pnode->parent->num_values++;
                pnode->parent->child[2] = n2;
                n2->child_state = 2;
                pnode->parent->child[3] = pnode;
                pnode->child_state = 3;
            }
    }
}

node* two34_tree::find(int k)
{
    int tnodestate = 0;
    node * tnode = root;
    while (true)
```

```
{
        if (tnode->num_values == 3)
        {
                tnodestate = tnode->child_state;
                break_3_value_node(tnode);
                if (tnodestate == 0||tnodestate==-1)
                {
                        if (tnode->parent->value[0] == k)
                                return nullptr;
                        if (tnode->parent->value[0] > k)
                        {
                                tnode = tnode->parent->child[0];
                        }
                        else
                        {
                                tnode = tnode->parent->child[1];
                        }

                }
                else if (tnodestate == 1)
                {
                        if (tnode->parent->value[1] == k)
                                return nullptr;
                        if (tnode->parent->value[1] > k)
                        {
                                tnode = tnode->parent->child[1];
                        }
                        else
                        {
```

```
                tnode = tnode->parent->child[2];
            }
        }
        else if (tnodestate == 2)
        {
            if (tnode->parent->value[2] == k)
                return nullptr;
            if (tnode->parent->value[2] > k)
                tnode = tnode->parent->child[2];
            else
                tnode = tnode->parent->child[3];
        }
    }
    else
    {
        if (tnode->num_values == 1)
        {
            if (tnode->value[0] > k)
            {
                if (tnode->child[0] != nullptr)
                    tnode = tnode->child[0];
                else
                    return tnode;
            }
            else
            {
                if (tnode->child[1] != nullptr)
                    tnode = tnode->child[1];
                else
```

```
                        return tnode;
                }
        }
        else if (tnode->num_values == 2)
        {
                if (tnode->value[0] > k)
                {
                        if (tnode->child[0] != nullptr)
                                tnode = tnode->child[0];
                        else
                                return tnode;
                }
                else if (k > tnode->value[0] && k < tnode->value[1])
                {
                        if (tnode->child[1] != nullptr)
                        {
                                tnode = tnode->child[1];
                        }
                        else
                                return tnode;
                }
                else if (k > tnode->value[1])
                {
                        if (tnode->child[2] != nullptr)
                                tnode = tnode->child[2];
                        else
                                return tnode;
                }
        }
```

```
                }
        }
}

void two34_tree::rotation(node *p)
{
        if (p->child_state == 0)
        {
                p->value[1] = p->parent->value[0];
                p->num_values++;
                if (p->parent->num_values == 3)
                {
                        p->parent->value[0] = p->parent->value[1];
                        p->parent->value[1] = p->parent->value[2];
                        p->parent->value[2]= p->parent->child[1]->value[0];

                }
                else if (p->parent->num_values == 2)
                {
                        p->parent->value[0] = p->parent->value[1];
                        p->parent->value[1] = p->parent->child[1]->value[0];
                }
                else
                {
                        p->parent->value[0] = p->parent->child[1]->value[0];
                }

                for (int i =1; i < p->parent->child[1]->num_values; i--)
                {
```

```
                p->parent->child[1]->value[i - 1] = p->parent->child[1]->value[i];
        }
        p->parent->child[1]->num_values--;
}
else if (p->child_state == 1)
{
        p->value[1] = p->parent->value[0];
        p->num_values++;
        if (p->parent->num_values == 3)
        {
                p->parent->value[0] = p->parent->value[1];
                p->parent->value[1] = p->parent->value[2];
                p->parent->value[2] = p->parent->child[2]->value[0];

        }
        else if (p->parent->num_values == 2)
        {
                p->parent->value[0] = p->parent->value[1];
                p->parent->value[1] = p->parent->child[2]->value[0];
        }
        else
        {
                p->parent->value[0] = p->parent->child[2]->value[0];
        }
        for (int i =1; i < p->parent->child[2]->num_values; i--)
        {
                p->parent->child[2]->value[i - 1] = p->parent->child[2]->value[i];
        }
        p->parent->child[2]->num_values--;
```

```
}
else if (p->child_state == 2)
{
        p->value[1] = p->parent->value[0];
        p->num_values++;
        if (p->parent->num_values == 3)
        {
                p->parent->value[0] = p->parent->value[1];
                p->parent->value[1] = p->parent->value[2];
                p->parent->value[2] = p->parent->child[3]->value[0];

        }
        else if (p->parent->num_values == 2)
        {
                p->parent->value[0] = p->parent->value[1];
                p->parent->value[1] = p->parent->child[3]->value[0];
        }
        else
        {
                p->parent->value[0] = p->parent->child[3]->value[0];
        }
        for (int i = 1; i < p->parent->child[3]->num_values; i--)
        {
                p->parent->child[3]->value[i - 1] = p->parent->child[3]->value[i];
        }
        p->parent->child[3]->num_values--;
}
else if (p->child_state == 3)
{
		}

			p->value[1] = p->value[0];
			p->value[0] = p->parent->value[0];
			p->num_values++;
			if (p->parent->num_values == 3)
			{
				p->parent->value[2] = p->parent->value[1];
				p->parent->value[1] = p->parent->value[0];
				p->parent->value[0] = p->parent->child[2]->value[p->parent->child[2]-
>num_values-1];
				p->parent->child[2]->value[p->parent->child[2]->num_values - 1] = -999;
				p->parent->child[2]->num_values--;


			}
			else if (p->parent->num_values == 2)
			{
				p->parent->value[0] = p->parent->value[1];
				p->parent->value[1] = p->parent->child[3]->value[0];
				p->parent->value[0] = p->parent->child[2]->value[p->parent->child[2]-
>num_values - 1];
				p->parent->child[2]->value[p->parent->child[2]->num_values - 1] = -999;
				p->parent->child[2]->num_values--;
			}
			else
			{
				p->parent->value[0] = p->parent->child[2]->value[p->parent->child[2]-
>num_values - 1];
				p->parent->child[2]->value[p->parent->child[2]->num_values - 1] = -999;
				p->parent->child[2]->num_values--;
			}
		}
```

```
}
void two34_tree::fusion(node *p)
{
        node* pnode = p;
        int cs = p->child_state;
        node *n = new node();
        {
                n->value[0] = pnode->value[0];
                n->value[1] = pnode->parent->value[cs];
                n->value[2] = pnode->parent->child[cs + 1]->value[0];
                n->parent = pnode->parent;
                if (pnode->is_leaf ==0)
                {
                        n->child[0] = pnode->child[0];
                        n->child[0]->parent = n;
                        n->child[1] = pnode->child[1];
                        n->child[1]->parent = n;
                        n->child[2] = pnode->parent->child[cs + 1]->child[0];
                        n->child[2]->parent = n;
                        n->child[3] = pnode->parent->child[cs + 1]->child[1];
                        n->child[3]->parent = n;
                }
                else
                        n->child[0] = nullptr;



                if (cs == 2)
                {
```

```
        n->parent->value[cs] = -999;

        n->parent->child[cs] = n;

        n->parent->num_values--;

        n->parent->child[cs+1] = nullptr;

    }

    else if (cs == 1)

    {

        if (n->parent->num_values == 3)

        {

            n->parent->child[cs+1] =n->parent->child[cs + 2];

            n->parent->value[cs] = n->parent->value[cs + 1];

            n->parent->child[cs + 2] =nullptr;

            n->parent->value[cs + 1] = -999;

            n->parent->child[cs] = n;

            n->parent->num_values--;

        }

        else if (n->parent->num_values == 2)

        {

            n->parent->value[cs] = -999;

            n->parent->child[cs] =n;

            n->parent->num_values--;

        }

    }

    else if (cs == 0)

    {

        if (n->parent->num_values == 3)

        {

            n->parent->value[cs] = n->parent->value[cs + 1];

            n->parent->child[cs + 1] = n->parent->child[cs + 2];
            n->parent->child[cs + 2] = n->parent->child[cs + 3];


            n->parent->value[cs + 1] = n->parent->value[cs + 2];


            n->parent->child[cs + 3] = nullptr;

            n->parent->value[cs + 2] = -999;

            n->parent->child[cs] = n;

            n->parent->num_values--;

        }

        else if (n->parent->num_values == 2)

        {

            n->parent->child[cs + 1] = n->parent->child[cs + 2];

            n->parent->value[cs] = n->parent->value[cs + 1];

            n->parent->child[cs + 2] = nullptr;

            n->parent->value[cs + 1] = -999;

            n->parent->child[cs] = n;

            n->parent->num_values--;


        }

        else if (n->parent->num_values == 1)

        {

            int cc = n->parent->child_state;

            if (n->child_state != -1)

            {

                    n->parent = n->parent->parent;

                    n->parent->child[cs] = n;

            }

            else

            {
```

```
                                        root = n;
                                        n->child_state = -1;
                                        n->parent = nullptr;
                                }
                        }
                }

                /*if (pnode->child[1]->child[0] != nullptr)
                {
                        pnode->child[2] = pnode->child[1]->child[0];
                        pnode->child[2]->parent = pnode;
                        pnode->child[2]->child_state = 2;
                }
                else
                        pnode->child[2] = nullptr;

                if (pnode->child[1]->child[1] != nullptr)
                {
                        pnode->child[3] = pnode->child[1]->child[1];
                        pnode->child[3]->parent = pnode;
                        pnode->child[3]->child_state = 3;
                }
                else
                        pnode->child[3] = nullptr;*/
        }p = n;
}
void two34_tree::add(int k)
{
        if (root == nullptr)
        {
                root = new node();
                root->add_value(k);
                root->parent = nullptr;
                root->child_state = -1;
                return;
        }
        node * result_node = new node();
        result_node =find(k);
        if(result_node != nullptr)
                result_node->add_value(k);
}

node * two34_tree::find_1(int k)
{
        node * rnode = root;
        int v1 = 0;
        bool gotit = false,itsdone=false;
        node *troot = root;
        int val = 0;
        while (itsdone == false)
        {
                if (troot != nullptr) {
                        if (troot->is_leaf == 0)
                        {
                                int r = 0;
                                while (troot != nullptr)
                                {
```

```
bool useit = false;
if (troot->value[r] == k)
{
        node * temp = new node();
        temp = troot;
        if (troot->is_leaf == 0)
        {
                troot = troot->child[r];
                while (useit == false)
                {
                        if (troot->is_leaf == 0)
                        {
                troot = troot->child[troot->num_values];
                        }
                        else
                                useit = true;
                }

                if (troot->num_values == 3)
                {
                        int val = troot->value[2];
                        troot->value[2] = -999;
                        troot->num_values--;
                        temp->value[r] = val;
                        return nullptr;
                }
                else if (troot->num_values == 2)
                {
                        int val = troot->value[1];
```

```
    troot->value[2] = -999;
    troot->num_values--;
    temp->value[r] = val;
    return nullptr;
}
else
{
    if (troot->num_values == 0)
    {
    //if(troot->parent->num_values > 1)
        //troot->value[1] = troot->
if (troot->parent->child[1]->num_values > 1)
        {
            rotation(troot);
            return troot;
        }
        else
        {
            fusion(troot);
            return troot;
        }
    }
    else
    {
if (troot->parent->child[0]->num_values > 1)
        {
            rotation(troot);
            return troot;
        }
```

```
            else
            {
                fusion(troot);
                return troot;
            }
        }

    }

    /*if (troot->num_values == 1)
    {
        while (true)
        {
            if (troot->parent != root)
            {
                //troot->parent->
            }
            else
            {

            }
        }
    }*/
}
else
{
    if (troot->num_values == 3)
    {
        int val = troot->value[2];
        troot->value[2] = -999;
        troot->num_values--;
        temp->value[r] = val;
        return nullptr;
    }
    else if (troot->num_values == 2)
    {
        int val = troot->value[1];
        troot->value[2] = -999;
        troot->num_values--;
        temp->value[r] = val;
        return nullptr;
    }
    else
    {
        if (troot->num_values == 0)
        {
        //if(troot->parent->num_values > 1)
//troot->value[1] = troot->
    if (troot->parent->child[1]->num_values > 1)
            {
                rotation(troot);
                return troot;
            }
            else
            {
                fusion(troot);
                return troot;
            }
```

```
                    }
                    else
                    {
                if (troot->parent->child[0]->num_values > 1)
                        {
                                rotation(troot);
                                return troot;
                        }
                        else
                        {
                                fusion(troot);
                                return troot;
                        }
                    }
                }

        }
}
else

{
        if (troot->value[r] > k)
                troot = troot->child[r];
        else
        {
                troot = troot->child[r + 1];

        }
}
```

```
                        }
                    }
                    else
                    {
                        if (troot->value[0] == k)
                        {
                            troot->value[0] = troot->value[1];
                            troot->value[1] = troot->value[2];
                            troot->value[2] = -999;
                        }
                        else if (troot->value[1] != -999 && troot->value[1] == k)
                        {
                            troot->value[1] = troot->value[2];
                            troot->value[2] = -999;
                        }
                        else if (troot->value[2] != -999 && troot->value[2] == k)
                        {
                            troot->value[2] = -999;
                        }
                    }



                }
                else return nullptr;
            }
            return nullptr;
```

```
}
void two34_tree::remove(int k)
{
	node * n1 = new node();
	n1 = find_1(k);
	if (n1 != nullptr)
		n1->remove_value(k);
}
void two34_tree::in_order_traversal(node * p) {
	cout << endl;
	node * p1 = p;
	if (p1 == nullptr) return;
	int i;
	for (i = 0; i < p1->num_values; i++) {
		in_order_traversal(p1->child[i]);
		cout << "  " << p1->value[i] << "  " << "child_state = " << p1->child_state;
	}
	in_order_traversal(p1->child[i]);
}



int main() {
	two34_tree t1;
	t1.add(40);
	t1.add(30);
	t1.add(20);
	t1.in_order_traversal(t1.root);
	t1.add(100);
	t1.add(120);
```

```
        t1.in_order_traversal(t1.root);

        t1.add(200);

        t1.in_order_traversal(t1.root);

        t1.add(400);

        t1.in_order_traversal(t1.root);

        t1.add(600);

        t1.in_order_traversal(t1.root);

        t1.remove(20);

        t1.in_order_traversal(t1.root);

        t1.remove(200);

        t1.in_order_traversal(t1.root);

        t1.remove(100);

        t1.in_order_traversal(t1.root);

        getchar();

        getchar();

        return 0;


}
```