Федеральное государственное автономное образовательное учреждение высшего образования «Казанский (Приволжский) федеральный университет»

Семестровая работа

по Алгоритмам и структурам данных

Тема: реализация на языке программирования C++ структуры данных АА-дерева, используемая для хранения и эффективного извлечения упорядоченных данных.

Выполнила студентка группы

11-002 Института ИТИС

Хасанова Л.Р.

Казань 2021

**АA-дерево** (англ. AA-Tree) — структура данных, представляющая собой сбалансированное двоичное дерево поиска, которое является разновидностью красно-черного дерева с дополнительными ограничениями. АA-дерево названо по первым буквам имени и фамилии изобретателя, Арне Андерссона, который впервые предложил данную модификацию красно-черного дерева в 1993 году. Это одно из самых быстрых бинарных деревьев с простой реализацией.

Как известно, основная проблема бинарного дерева — его балансировка, то есть противодействие вырождению в обычный связный список. Обычно при балансировке дерева возникает много вариантов взаимного расположения вершин, каждый из которых нужно учесть в алгоритме отдельно. Так Арне Андерссон придумал АА-дерево, которое упрощает балансировку дерева.

**Основной принцип устройства и особенности АА-дерева.**

Для упрощения балансировки дерева было введено понятие **уровень**(*level*) вершины. Если представить дерево растущим сверху вниз от корня (то есть «стоящим на листьях»), то уровень любой листовой вершины будет равен 1. В своей работе Арне Андерссон приводит простое правило, которому должно удовлетворять AA-дерево:

*«К одной вершине можно присоединить другую вершину того же уровня, но только одну и только справа».*

**Свойства АА-дерева:**

• Уровень каждого листа равен 1.

• Уровень каждого левого ребенка ровно на один меньше, чем у его родителя.

• Уровень каждого правого ребенка равен или на один меньше, чем у его родителя.

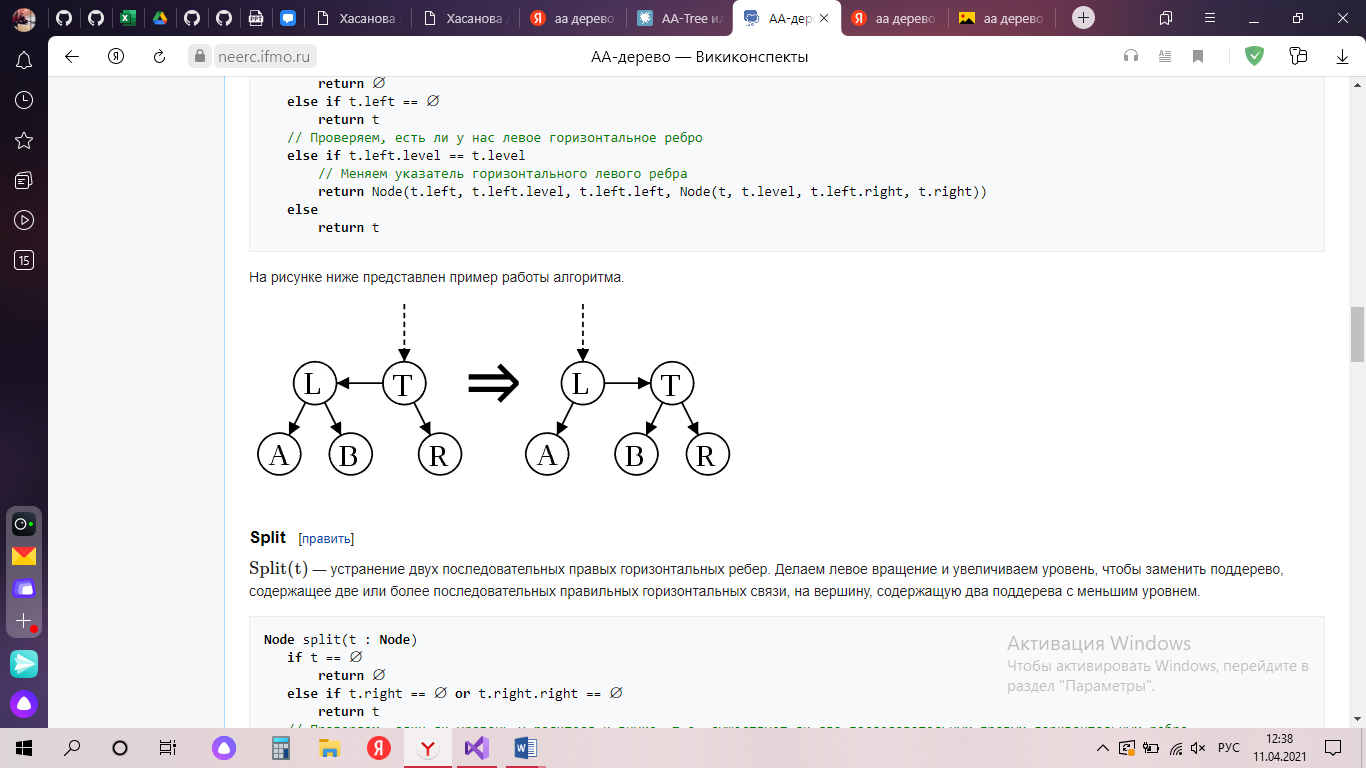
• Уровень каждого правого внука строго меньше, чем у его прародителя.

• Каждая вершина с уровнем больше 1 имеет двоих детей.

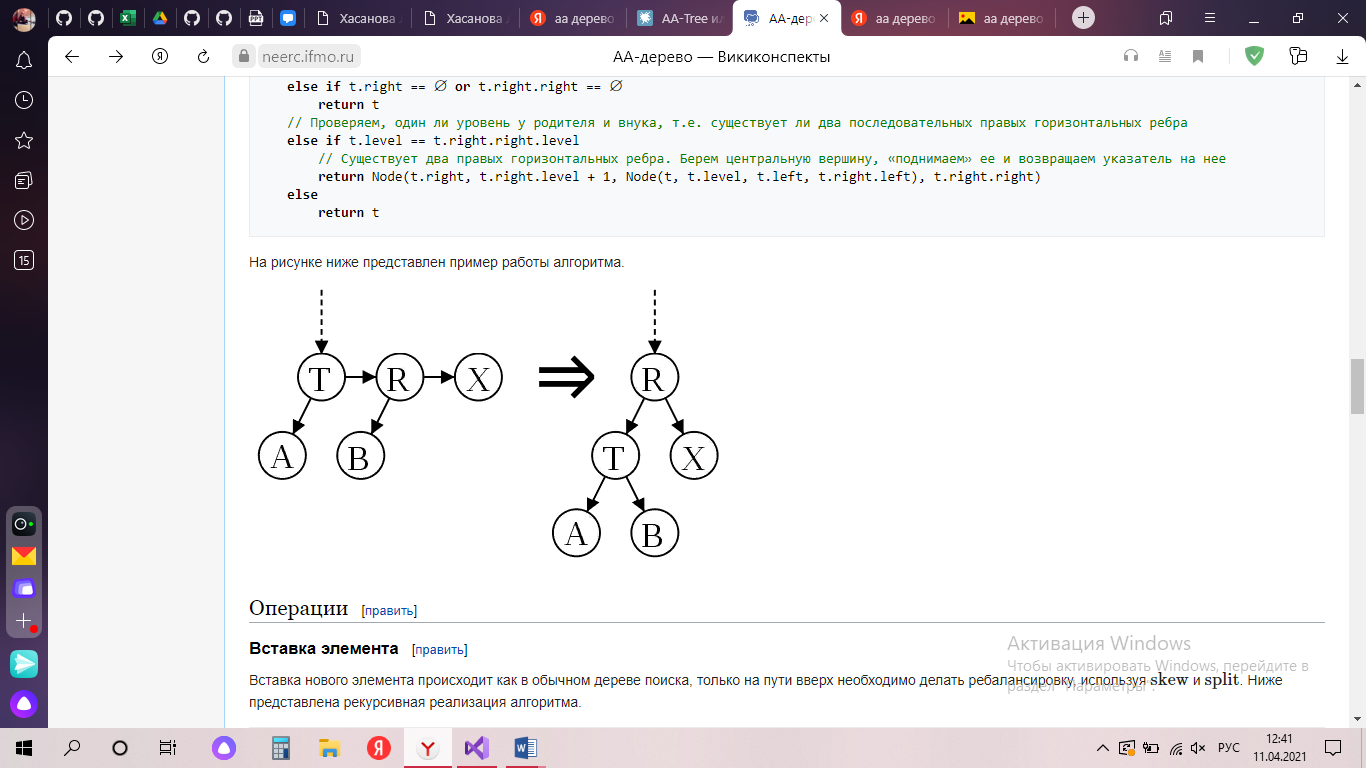
**Балансировка АА-дерева:**

Для балансировки АА-дерева нужны следующие две операции:

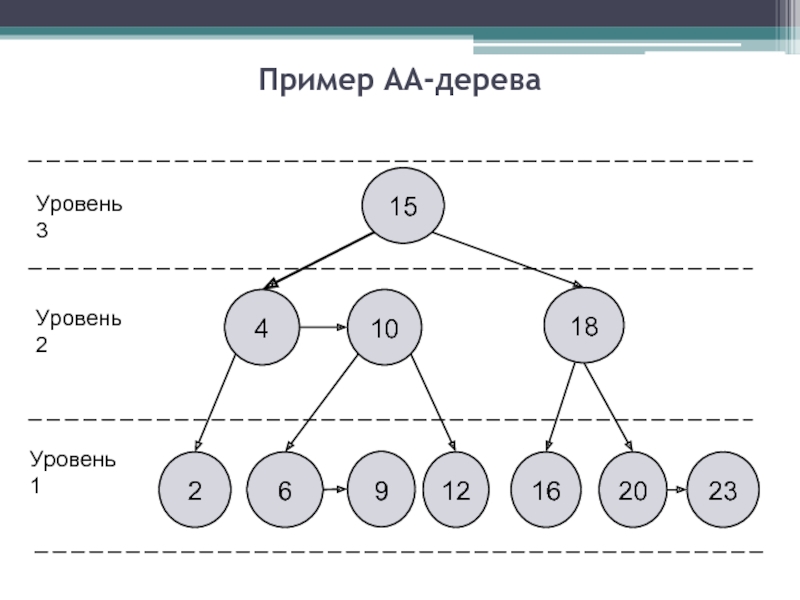
**1)** **Skew()** — устранение левого горизонтального ребра. Делаем правое вращение, чтобы заменить поддерево, содержащее левую горизонтальную связь, на поддерево, содержащее разрешенную правую горизонтальную связь. На рисунке ниже представлен пример работы алгоритма.



**2)** **Split()** — устранение двух последовательных правых горизонтальных ребер. Делаем левое вращение и увеличиваем уровень, чтобы заменить поддерево, содержащее две или более последовательных правильных горизонтальных связи, на вершину, содержащую два поддерева с меньшим уровнем. На рисунке ниже представлен пример работы алгоритма.



**Пример АА-дерева:**



**Оценка временной сложности АА-дерева.**

Операции поиска/вставки/удаления в АА-дереве имеют временную сложность O(log(N)).

АА-дерево, которое содержит n внутренних узлов, имеет высоту O(log(N)).

**Обозначения:**

• h(v) – высота поддерева с корнем в v

• bh(v) – число узлов, не расположенных на том же уровне, что и их родители (не считая v{\displaystyle v}, если он как раз правый потомок) от {\displaystyle v}v до любого листа в поддереве.

**Лемма:** Поддерево с корнем в узле v имеет не менее  внутренних узлов.

**Доказательство леммы** (индукцией по высоте):

1) Основание индукции: {\displaystyle h(v)=0}h(v) = 0. Если поддерево имеет нулевую высоту, то {\displaystyle v}v должен быть null, поэтому {\displaystyle bh(''v'')=0}bh(v) = 0. Следовательно,  **= = = 0**.

2) Индукционный шаг: пусть узел {\displaystyle v}v такой, что h(v) = k{\displaystyle h(v)=k}h(v)hh и поддерево имеет не менее  {\displaystyle 2^{bh(v)}-1}

внутренних узлов. Покажем, что тогда {\displaystyle v'}v’, для которого h(v’) = k+1{\displaystyle h(v')=k+1}h(jbksjc, имеет не менее  внутренних {\displaystyle 2^{bh(v')}-1}узлов. Так как {\displaystyle v'}v’ имеет h(v’) > 0{\displaystyle h(v')>0}hр, это внутренний узел. Как таковой он имеет два потомка, оба из которых имеют высоту bh(v’){\displaystyle bh(v')}bh(, либо bh(v’)-1{\displaystyle bh(v')-1} (зависит от того, является {\displaystyle v'}v’ узлом, расположенным на одном уровне с родителем, или нет). По индукционному предположению каждый потомок имеет не менее  {\displaystyle 2^{bh(v')-1}-1} внутренних узлов, поэтому {\displaystyle v'}v’ имеет не менее

**{\displaystyle 2^{bh(v')-1}-1}+ + 1 = {\displaystyle 2^{bh(v')-1}-1}{\displaystyle 2^{bh(v')-1}-1}**внутренних{\displaystyle 2^{bh(v')-1}-1+2^{bh(v')-1}-1+1=2^{bh(v')}-1}узлов.

{\displaystyle 2^{bh(v)}-1=2^{0}-1=1-1=0}Используя эту лемму, мы можем показать, что дерево имеет логарифмическую высоту. Высота корня, не имеющая узлов, расположенных на одном уровне с родителями, не менее h(root)/2. По лемме имеем:

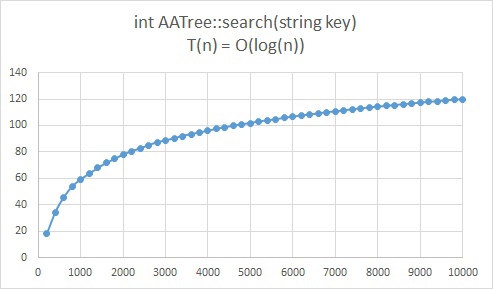
**n > - 1 ⬄ log2(n+1) ≥ ⬄ h(root) ≤ 2log2(n+1)**

Поэтому высота корня O(log(n)). Следовательно, операции поиска/вставки/удаления в АА-дереве имеют временную сложность O(log(N)), так как они по времени пропорциональны высоте дерева.

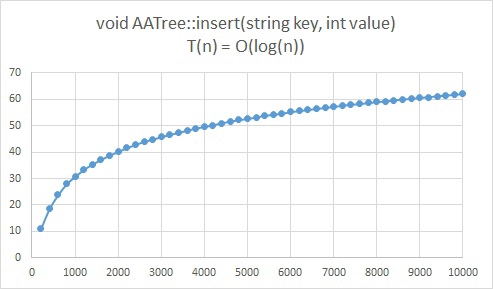
**Графики.**

Сравнивая с ниже представленные графики с графиком log2 (N) можем заметить, что существенных отличий от графика log2 (N) нет.

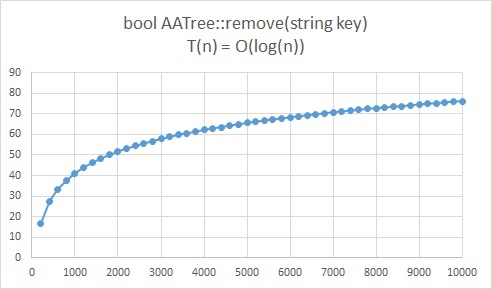
• График алгоритма поиска вершины:



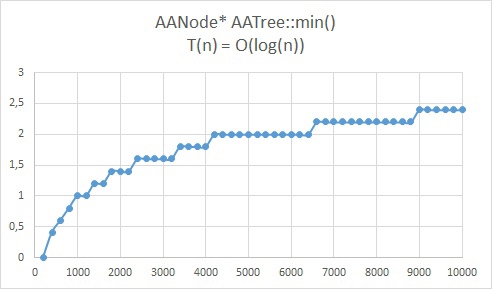
• График алгоритма вставки вершины:



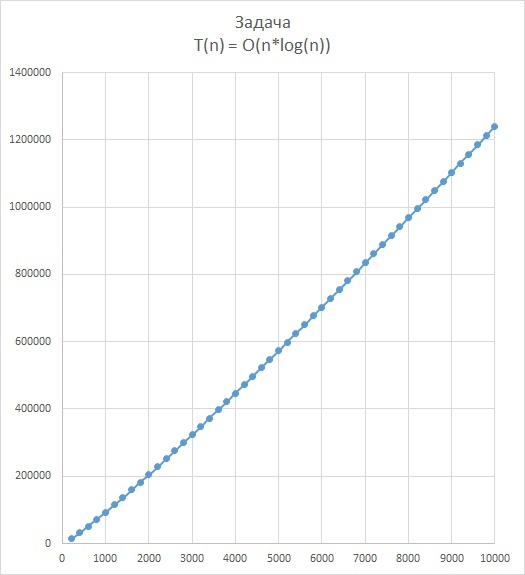
• График алгоритма удаления вершины:



• График алгоритма поиска минимальной вершины:



• График временной сложности задачи.



**Плюсы и минусы АА-дерева.**

**Плюсы:**

• Меньше оборотов, в отличие от красно-черного дерева, поэтому их легче кодировать, особенно удаления (устраняет около половины случаев вращения);

• Устраняет многие условия, которые необходимо учитывать для поддержания красно-черного дерева;

• Только два случая должны быть рассмотрены и исправлены для поддержания баланса АА-дерева;

• Для восстановления равновесия в дереве необходимы только две отдельные операции: skew и split.

• Легко проверить правильность дерева и выполнить любые вращения по мере необходимости. Если возникает горизонтальная левая связь, будет выполнен skew, а если возникают две горизонтальные правые связи, будет выполнено split, возможно, увеличивающее уровень нового корневого узла текущего поддерева.

**Минусы:**

• В то время как красно-черные деревья требуют одного бита балансирующих метаданных на узел (цвет), АА-деревья требуют O(log(log(N)) битов метаданных на узел в виде целого "уровня";

• АА-дерево имеет много свойств и ограничений, несоблюдение хотя бы 1 из них ведет к нарушению структуры дерева;

• Не смотря на то, что самые простые алгоритмы вставки и удаления узлов у АА-дерева, однако, если вставка и удаление узлов встречаются гораздо реже, чем поиск, то предпочтительнее использовать красно-черные деревья.

**Вывод:**

Производительность АА-дерева эквивалентна производительности красно-черного дерева. В то время как АА-дерево делает больше вращений, чем красно-черное дерево, более простые алгоритмы, как правило, работают быстрее, и все это уравновешивается, чтобы привести к аналогичной производительности. Красно-черное дерево более последовательно по своей производительности, чем дерево АА, но дерево АА имеет тенденцию быть более плоским, что приводит к несколько более быстрому времени поиска.

В своей работе Арне Андерссон делает вывод, что если сравнивать по производительности 4 типа двоичных деревьев поиска, а именно:

• АВЛ-дерево;

• красно-черное дерево;

• 2-3 дерево;

• АА-дерево;

То можно сделать вывод, что сбалансированность (и скорость поиска) лучше всего у АВЛ-дерева, чуть хуже у красно-черного дерева и еще чуть хуже у 2-3 дерева и эквивалентного ему АА-дерева.

Алгоритмы балансировки очень сложны для АВЛ-дерева и 2-3 дерева, поэтому на практике предпочитают использовать красно-черные и АА-деревья. Самые простые алгоритмы вставки и удаления узлов у АА-дерева, однако, если вставка и удаление узлов встречаются гораздо реже, чем поиск, то красно-черные деревья будут предпочтительнее.

Преимущества АА-дерева по сравнению с красно-черным деревом является то, что алгоритмы, используемые при вставке и удалении узла в АА-дереве, а также балансировка дерева, существенно проще, чем в красно-черном дереве.

**Список использованной литературы (источников).**

1) <https://en.wikipedia.org/wiki/AA_tree>;

2) [https://neerc.ifmo.ru/wiki/index.php?title=AA-дерево#:~:text=АA-дерево%20(англ.%20AA-Tree)%20—%20структура,красно- черного%20дерева%20с%20дополнительными%20 ограничениями](https://neerc.ifmo.ru/wiki/index.php?title=AA-дерево#:~:text=АA-дерево%20(англ.%20AA-Tree)%20—%20структура,красно-черного%20дерева%20с%20дополнительными%20ограничениями);

3) <https://web.eecs.umich.edu/~sugih/courses/eecs281/f11/lectures/12-AAtrees+Treaps.pdf>;

4) <https://ycpcs.github.io/cs350-fall2017/lectures/AA-tree_lecture.pdf>;

5) <http://www.cse.chalmers.se/edu/year/2016/course/DIT960/slides/9.pdf>;

6) <http://espressocode.top/aa-trees-set-1-introduction/>;

7) <https://www.nayuki.io/page/aa-tree-set>;

8) <https://vk.com/doc-127988977_442328811?hash=932e14b24ef3e06ebf&dl=f08881177666976433>;

9) <https://vk.com/doc179585542_437543470?hash=a3caec405846c395d4&dl=d34bee6f7a087e529c>;

**Код программы.**

/\*

Задача:

Посчитать количество различных слов в тексте и вывести список слов с числом их

вхождений. Для решения данной задачи использовалась структура данных AA-дерево.

1) Слова из исходного текста добавляются в AA-дерево при помощи метода

void insert(string key, int value). При этом в качестве ключа используется само

слово, а в качестве значения - сколько раз оно встречалось, то есть при первом

добавлении слова value равно 1, а при каждом последующем value увеличивается на 1.

Перед вставкой слова в дерево вызывается метод int search(string key) для

определения текущего значения value для добавляемого слова.

2) После вставки всех слов в AA-дерево, структура данных будет хранить в своих

узлах весь набор слов текста (с соответствующим числом вхождений). Для вывода

слов в алфавитном порядке используется метод AANode\* min(), извлекающий из

AA-дерева узел с наименьшим значением ключа (так как ключи типа string, то их

сравнение происходит в лексикографическом порядке). После нахождения узла с

минимальным ключом производится печать в консоль ключа и значения, хранившихся в

узле, и удаление данного узла из AA-дерева при помощи метода bool remove(string key).

Замечание 1. Исходный текст для задачи может быть дан в виде строки, введённой из

консоли, или же сгенерирован каким-либо образом. Слова из исходного текста перед

вставкой в AA-дерево парсятся в vector<string> words. В тестах, написанных к данной

задаче, вектор words заполняется словами, сгенерированными псевдослучайным образом

при помощи метода string generate\_word(), реализованного на основе работы метода rand().

Замечание 2. Время работы методов AA-дерева, использующихся для решения выше описанной

задачи, а также время, затраченное на всё решение задачи, измеряется при помощи

инструментов стандартной библиотеки chrono.

\*/

#include <iostream>

#include <sstream>

#include <vector>

#include <chrono>

using namespace std;

using namespace std::chrono;

struct AANode {

int level;

string key;

int value;

AANode\* right;

AANode\* left;

AANode\* parent;

};

class AATree {

public:

AATree();

~AATree();

void insert(string, int);

bool remove(string);

int search(string key);

AANode\* min();

AANode\* max();

private:

AANode\* root;

void dealloc\_memory(AANode\* N);

void skew(AANode\*);

bool split(AANode\*);

void balance(AANode\*);

AANode\* insert\_helper(AANode\*, AANode\*);

bool remove\_helper(AANode\*, AANode\*, string);

int search\_helper(AANode\*, string);

};

/\*

конструктор класса AATree

\*/

AATree::AATree() {

root = new AANode;

root = nullptr;

}

/\*

деструктор класса AATree

\*/

AATree::~AATree() {

dealloc\_memory(root);

}

void AATree::dealloc\_memory(AANode\* N) {

if (N == nullptr) {

return;

}

dealloc\_memory(N->left);

dealloc\_memory(N->right);

delete N;

}

/\*

реализация метода insert, который выполняет алгоритм "вставки" узлов;

алгоритм осуществляется следующим способом:

1. добавление узла на первый уровень

2. вызов метода skew (для устранения горизонтальной левой связи)

3. вызов метода split (для устранения двух горизонтальных правых связей,

возможно, увеличивающее уровень нового корневого узла текущего поддерева)

\*/

void AATree::insert(string key, int value) {

AANode\* temp = new AANode;

temp->key = key;

temp->value = value;

temp->level = 1;

temp->left = nullptr;

temp->right = nullptr;

temp->parent = nullptr;

insert\_helper(root, temp);

}

AANode\* AATree::insert\_helper(AANode\* temp, AANode\* ins) {

if (root == nullptr) {

ins->parent = nullptr;

ins->left = nullptr;

ins->right = nullptr;

root = ins;

return root;

}

if (ins->key < temp->key) {

if (temp->left) {

return insert\_helper(temp->left, ins);

}

temp->left = ins;

ins->parent = temp;

balance(ins);

return ins;

}

if (ins->key > temp->key) {

if (temp->right) {

return insert\_helper(temp->right, ins);

}

temp->right = ins;

ins->parent = temp;

balance(ins);

return ins;

}

if (ins->key == temp->key) {

temp->value = ins->value;

}

delete ins;

return temp;

}

/\*

реализация метода remove, который осуществляет удаление узлов в дереве

\*/

bool AATree::remove(string key) {

return this->remove\_helper(nullptr, root, key);

}

bool AATree::remove\_helper(AANode\* parent, AANode\* current, string key) {

if (!current) {

return false;

}

if (current->key == key) {

if (current->left == nullptr || current->right == nullptr) {

AANode\* temp = current->left;

if (current->right) {

temp = current->right;

}

if (parent) {

if (parent->left == current) {

parent->left = temp;

}

else {

parent->right = temp;

}

}

else {

root = temp;

}

}

else {

AANode\* replacement = current->right;

while (replacement->left) {

replacement = replacement->left;

}

string temp = current->key;

current->key = replacement->key;

replacement->key = temp;

return remove\_helper(current, current->right, temp);

}

delete current;

return true;

}

return remove\_helper(current, current->left, key) ||

remove\_helper(current, current->right, key);

}

/\*

реализация метода skew, который устраняет левую связь на одном уровне; данный

метод устроняет горизонтальную левую связь при помощи вращения узла вправо

каждый раз, когда найдена горизонтальная левая связь

\*/

void AATree::skew(AANode\* temp) {

AANode\* ptr = temp->left;

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

temp->parent->left = ptr;

}

else {

temp->parent->right = ptr;

}

ptr->parent = temp->parent;

temp->parent = ptr;

temp->left = ptr->right;

if (temp->left != nullptr) {

temp->left->parent = temp;

}

ptr->right = temp;

temp->level = (temp->left ? temp->left->level + 1 : 1);

}

/\*

реализация метода split, который устраняет две правых связи на одном уровне;

данный метод устроняет две последовательные правые горизонтальные связи при

помощи вращения узла влево на одно, содержащее две меньших последовательных

правых горизонтальных связи

\*/

bool AATree::split(AANode\* temp) {

AANode\* ptr = temp->right;

if (ptr && ptr->right && (ptr->right->level == temp->level)) {

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

temp->parent->left = ptr;

}

else {

temp->parent->right = ptr;

}

ptr->parent = temp->parent;

temp->parent = ptr;

temp->right = ptr->left;

if (temp->right != nullptr) {

temp->right->parent = temp;

}

ptr->left = temp;

ptr->level = temp->level + 1;

return true;

}

return false;

}

/\*

реализация метода balance, который делает дерево сбалансированным при помощи

методов split и skew (достаточные методы для сбалансированного дерева)

\*/

void AATree::balance(AANode\* temp) {

temp->left = nullptr;

temp->right = nullptr;

temp->level = 1;

for (temp = temp->parent; temp != root; temp = temp->parent) {

if (temp->level != (temp->left ? temp->left->level + 1 : 1)) {

skew(temp);

if (temp->right == nullptr) {

temp = temp->parent;

}

else if (temp->level != temp->right->level) {

temp = temp->parent;

}

}

if (temp->parent != root) {

if (!split(temp->parent)) {

break;

}

}

}

}

/\*

реализация метода search, токорый осуществляет поиск вершины в дереве

\*/

int AATree::search(string key) {

return search\_helper(root, key);

}

int AATree::search\_helper(AANode\* temp, string key) {

if (temp == nullptr) {

return 0;

}

if (key == temp->key) {

return temp->value;

}

else if (key < temp->key) {

return search\_helper(temp->left, key);

}

else if (key > temp->key) {

return search\_helper(temp->right, key);

}

return 0;

}

/\*

поиск минимальной вершины в дереве

\*/

AANode\* AATree::min() {

AANode\* N = root;

if (N != nullptr) {

while (N->left != nullptr) {

N = N->left;

}

}

return N;

}

/\*

поиск максимальной вершины в дереве

\*/

AANode\* AATree::max() {

AANode\* N = root;

if (N != nullptr) {

while (N->right != nullptr) {

N = N->right;

}

}

return N;

}

bool is\_punctuation(char symbol) {

if (ispunct(symbol) || isspace(symbol)) {

return true;

}

return false;

}

vector<string> split\_string(string& str) {

vector<string> words;

string temp;

for (size\_t i = 0; i < str.size(); i++) {

char letter = str[i];

if (!is\_punctuation(letter)) {

if (isupper(letter)) {

letter = tolower(letter);

}

temp += letter;

}

else {

if (temp.length() > 0) {

words.push\_back(temp);

}

temp.clear();

}

}

if (temp != "") {

words.push\_back(temp);

}

return words;

}

string generate\_word() {

const char\* letters[] = { "q", "w", "e", "r", "t", "y", "u", "i", "o", "p", "a", "s", "d",

"f", "g", "h", "j", "k", "l", "z", "x", "c", "v", "b", "n", "m" };

int length\_of\_word = (rand() % 7) + 1;

string word = "";

for (int j = 0; j < length\_of\_word; j++) {

word += letters[rand() % (sizeof letters / sizeof(char\*))];

}

return word;

}

void task\_test(vector<string>& words, int number\_of\_test) {

cout << "Test #" << number\_of\_test << endl;

cout << "Input data size: " << number\_of\_test \* 200 << " elements" << endl;

for (int j = 0; j < 5; j++) {

auto start = high\_resolution\_clock::now();

AATree\* tree = new AATree();

for (int i = 0; i < words.size() - 1; i++) {

int count = tree->search(words[i]) + 1;

tree->insert(words[i], count);

}

auto startSearch = high\_resolution\_clock::now();

int count = tree->search(words[words.size() - 1]) + 1;

auto stopSearch = high\_resolution\_clock::now();

cout << "search: " << duration\_cast<microseconds>(stopSearch - startSearch).count() << "; ";

auto startInsert = high\_resolution\_clock::now();

tree->insert(words[words.size() - 1], count);

auto stopInsert = high\_resolution\_clock::now();

cout << "insert: " << duration\_cast<microseconds>(stopInsert - startInsert).count() << "; ";

auto startMin = high\_resolution\_clock::now();

AANode\* min\_node = tree->min();

auto stopMin = high\_resolution\_clock::now();

cout << "min: " << duration\_cast<microseconds>(stopMin - startMin).count() << "; ";

if (min\_node != nullptr) {

//cout << min\_node->key << ": " << min\_node->value << endl;

auto startRemove = high\_resolution\_clock::now();

tree->remove(min\_node->key);

auto stopRemove = high\_resolution\_clock::now();

cout << "remove: " << duration\_cast<microseconds>(stopRemove - startRemove).count() << "; ";

min\_node = tree->min();

}

while (min\_node != nullptr) {

//cout << min\_node->key << ": " << min\_node->value << endl;

tree->remove(min\_node->key);

min\_node = tree->min();

}

tree->~AATree();

auto stop = high\_resolution\_clock::now();

cout << "task: " << duration\_cast<microseconds>(stop - start).count() << endl;

}

}

int main() {

vector<string> words;

//string text;

//getline(cin, text);

//words = split\_string(text);

for (int test = 1; test <= 50; test++) {

for (int i = 0; i < 200; i++) {

words.push\_back(generate\_word());

}

task\_test(words, test);

}

return 0;

}

**Таблицы полученных значений времени работы в зависимости от размера данных.**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| https://sun9-3.userapi.com/impg/sy-3wexL_jGTe6Rp2o-zxeXVLz3TYgTUNzKxNQ/M8gw9iW9tac.jpg?size=248x1080&quality=96&sign=6c42d612b2956f6f01c5bfcfc2ea4fc3&type=album | https://sun9-76.userapi.com/impg/G50B-IptJAShUjfqteAMT2lVnmjgkY9OKUeTQA/dEA4gkMQtzQ.jpg?size=248x1080&quality=96&sign=343b9b64375c8d055824272391d970a2&type=album | https://sun9-47.userapi.com/impg/gmP648i5N_SzTNle0dpCKxW9zE0iGJmukxUteA/RXVr3EKyuFI.jpg?size=248x1080&quality=96&sign=25daed668ca7bab06ea41f70899d4949&type=album | https://sun9-73.userapi.com/impg/TASwK4mHxMNahn5RIS9oJJF2pVW00tE72q1PmQ/uVuVn8o7NHQ.jpg?size=248x1080&quality=96&sign=06e16662013fbf04d2cec01919722e35&type=album | https://sun9-58.userapi.com/impg/hr_uKOtJE1FRaKK8dMOP_okAsu0Y2pz_jtBgdA/UICVjN8CLX4.jpg?size=420x1080&quality=96&sign=f9a13da85916eaa459948f27c3d3be27&type=album |

**Все входные данные.**

Test #1

Input data size: 200 elements

search: 16; insert: 12; min: 0; remove: 18; task: 14227

search: 18; insert: 9; min: 0; remove: 16; task: 13333

search: 19; insert: 10; min: 0; remove: 15; task: 13965

search: 20; insert: 11; min: 0; remove: 17; task: 14525

search: 18; insert: 13; min: 0; remove: 18; task: 14906

Test #2

Input data size: 400 elements

search: 35; insert: 17; min: 0; remove: 27; task: 32668

search: 32; insert: 19; min: 1; remove: 27; task: 29379

search: 35; insert: 16; min: 0; remove: 29; task: 33571

search: 36; insert: 22; min: 0; remove: 26; task: 34017

search: 34; insert: 19; min: 1; remove: 27; task: 32614

Test #3

Input data size: 600 elements

search: 42; insert: 22; min: 1; remove: 34; task: 51800

search: 47; insert: 25; min: 0; remove: 35; task: 50288

search: 49; insert: 24; min: 1; remove: 31; task: 49621

search: 46; insert: 24; min: 0; remove: 34; task: 54455

search: 45; insert: 24; min: 1; remove: 33; task: 52403

Test #4

Input data size: 800 elements

search: 57; insert: 29; min: 1; remove: 39; task: 68738

search: 51; insert: 26; min: 1; remove: 35; task: 74116

search: 53; insert: 30; min: 1; remove: 38; task: 72490

search: 56; insert: 28; min: 1; remove: 37; task: 72645

search: 52; insert: 27; min: 0; remove: 39; task: 71751

Test #5

Input data size: 1000 elements

search: 58; insert: 30; min: 1; remove: 41; task: 90363

search: 59; insert: 32; min: 1; remove: 44; task: 95895

search: 57; insert: 29; min: 1; remove: 40; task: 94906

search: 62; insert: 28; min: 1; remove: 41; task: 93742

search: 60; insert: 35; min: 1; remove: 40; task: 90670

Test #6

Input data size: 1200 elements

search: 64; insert: 34; min: 1; remove: 46; task: 116530

search: 67; insert: 32; min: 1; remove: 44; task: 111539

search: 59; insert: 36; min: 1; remove: 43; task: 114125

search: 61; insert: 33; min: 1; remove: 41; task: 118676

search: 66; insert: 32; min: 1; remove: 46; task: 112936

Test #7

Input data size: 1400 elements

search: 66; insert: 37; min: 1; remove: 47; task: 138755

search: 70; insert: 35; min: 2; remove: 45; task: 136698

search: 71; insert: 35; min: 1; remove: 46; task: 135638

search: 67; insert: 33; min: 1; remove: 48; task: 131373

search: 65; insert: 37; min: 1; remove: 45; task: 139151

Test #8

Input data size: 1600 elements

search: 72; insert: 37; min: 1; remove: 47; task: 157783

search: 69; insert: 38; min: 1; remove: 49; task: 159332

search: 73; insert: 38; min: 1; remove: 49; task: 163195

search: 70; insert: 36; min: 2; remove: 48; task: 155189

search: 74; insert: 37; min: 1; remove: 49; task: 162737

Test #9

Input data size: 1800 elements

search: 77; insert: 40; min: 2; remove: 51; task: 186828

search: 76; insert: 40; min: 0; remove: 53; task: 180409

search: 71; insert: 39; min: 1; remove: 50; task: 182563

search: 74; insert: 39; min: 2; remove: 48; task: 178535

search: 75; insert: 36; min: 2; remove: 48; task: 179726

Test #10

Input data size: 2000 elements

search: 80; insert: 39; min: 1; remove: 52; task: 205928

search: 76; insert: 40; min: 2; remove: 51; task: 198894

search: 75; insert: 42; min: 2; remove: 52; task: 204224

search: 78; insert: 42; min: 1; remove: 53; task: 205212

search: 81; insert: 38; min: 1; remove: 51; task: 209870

Test #11

Input data size: 2200 elements

search: 81; insert: 43; min: 1; remove: 55; task: 231041

search: 77; insert: 41; min: 1; remove: 52; task: 227800

search: 85; insert: 39; min: 2; remove: 54; task: 225394

search: 80; insert: 42; min: 1; remove: 53; task: 227825

search: 79; insert: 43; min: 2; remove: 52; task: 228454

Test #12

Input data size: 2400 elements

search: 84; insert: 42; min: 2; remove: 54; task: 249219

search: 86; insert: 39; min: 1; remove: 55; task: 254685

search: 78; insert: 44; min: 2; remove: 52; task: 253987

search: 82; insert: 43; min: 2; remove: 54; task: 248229

search: 83; insert: 46; min: 1; remove: 57; task: 251730

Test #13

Input data size: 2600 elements

search: 83; insert: 44; min: 2; remove: 55; task: 279213

search: 88; insert: 45; min: 2; remove: 56; task: 272745

search: 84; insert: 42; min: 1; remove: 55; task: 271640

search: 79; insert: 45; min: 1; remove: 58; task: 274787

search: 91; insert: 44; min: 2; remove: 54; task: 278215

Test #14

Input data size: 2800 elements

search: 92; insert: 44; min: 1; remove: 57; task: 301627

search: 84; insert: 43; min: 2; remove: 57; task: 297684

search: 85; insert: 44; min: 2; remove: 56; task: 304527

search: 89; insert: 44; min: 2; remove: 55; task: 298728

search: 86; insert: 49; min: 1; remove: 59; task: 296080

Test #15

Input data size: 3000 elements

search: 88; insert: 47; min: 2; remove: 57; task: 328962

search: 91; insert: 45; min: 2; remove: 58; task: 319316

search: 85; insert: 46; min: 1; remove: 57; task: 327057

search: 87; insert: 46; min: 2; remove: 60; task: 323572

search: 92; insert: 45; min: 1; remove: 57; task: 318100

Test #16

Input data size: 3200 elements

search: 93; insert: 45; min: 1; remove: 59; task: 342739

search: 90; insert: 49; min: 2; remove: 59; task: 344878

search: 94; insert: 43; min: 2; remove: 59; task: 350701

search: 88; insert: 48; min: 1; remove: 56; task: 353911

search: 86; insert: 48; min: 2; remove: 61; task: 346722

Test #17

Input data size: 3400 elements

search: 93; insert: 48; min: 2; remove: 58; task: 371262

search: 88; insert: 45; min: 1; remove: 61; task: 374123

search: 97; insert: 47; min: 2; remove: 58; task: 370143

search: 91; insert: 48; min: 2; remove: 62; task: 372913

search: 91; insert: 49; min: 2; remove: 60; task: 373265

Test #18

Input data size: 3600 elements

search: 90; insert: 48; min: 2; remove: 61; task: 398886

search: 96; insert: 50; min: 2; remove: 59; task: 398766

search: 89; insert: 50; min: 1; remove: 63; task: 394982

search: 94; insert: 47; min: 2; remove: 60; task: 400335

search: 98; insert: 46; min: 2; remove: 60; task: 391392

Test #19

Input data size: 3800 elements

search: 95; insert: 50; min: 2; remove: 63; task: 425285

search: 93; insert: 48; min: 1; remove: 61; task: 421317

search: 99; insert: 51; min: 2; remove: 61; task: 419188

search: 97; insert: 47; min: 2; remove: 63; task: 423461

search: 89; insert: 49; min: 2; remove: 59; task: 419316

Test #20

Input data size: 4000 elements

search: 94; insert: 50; min: 2; remove: 62; task: 445049

search: 98; insert: 50; min: 2; remove: 63; task: 448100

search: 98; insert: 49; min: 2; remove: 62; task: 442783

search: 93; insert: 51; min: 2; remove: 61; task: 449434

search: 97; insert: 48; min: 1; remove: 63; task: 448470

Test #21

Input data size: 4200 elements

search: 91; insert: 49; min: 2; remove: 63; task: 474715

search: 100; insert: 49; min: 2; remove: 60; task: 472316

search: 98; insert: 52; min: 2; remove: 65; task: 471355

search: 96; insert: 50; min: 2; remove: 63; task: 469588

search: 102; insert: 51; min: 2; remove: 64; task: 470421

Test #22

Input data size: 4400 elements

search: 101; insert: 52; min: 2; remove: 65; task: 493776

search: 96; insert: 50; min: 2; remove: 64; task: 501998

search: 95; insert: 51; min: 1; remove: 61; task: 500655

search: 103; insert: 51; min: 3; remove: 66; task: 497950

search: 98; insert: 51; min: 2; remove: 62; task: 490604

Test #23

Input data size: 4600 elements

search: 97; insert: 51; min: 2; remove: 65; task: 526743

search: 102; insert: 53; min: 2; remove: 66; task: 520926

search: 103; insert: 50; min: 2; remove: 67; task: 523780

search: 96; insert: 54; min: 2; remove: 61; task: 519674

search: 101; insert: 50; min: 2; remove: 63; task: 520939

Test #24

Input data size: 4800 elements

search: 100; insert: 54; min: 2; remove: 64; task: 546501

search: 101; insert: 50; min: 2; remove: 63; task: 551664

search: 98; insert: 53; min: 2; remove: 62; task: 546843

search: 104; insert: 51; min: 2; remove: 69; task: 545198

search: 101; insert: 53; min: 2; remove: 67; task: 548614

Test #25

Input data size: 5000 elements

search: 100; insert: 53; min: 2; remove: 66; task: 576105

search: 99; insert: 53; min: 3; remove: 65; task: 573654

search: 108; insert: 52; min: 2; remove: 65; task: 569998

search: 97; insert: 53; min: 1; remove: 66; task: 568481

search: 105; insert: 52; min: 2; remove: 65; task: 578872

Test #26

Input data size: 5200 elements

search: 98; insert: 54; min: 2; remove: 66; task: 599009

search: 103; insert: 51; min: 2; remove: 69; task: 597914

search: 107; insert: 55; min: 2; remove: 65; task: 600841

search: 104; insert: 52; min: 2; remove: 64; task: 601812

search: 103; insert: 54; min: 2; remove: 67; task: 595479

Test #27

Input data size: 5400 elements

search: 104; insert: 53; min: 1; remove: 70; task: 628987

search: 109; insert: 56; min: 2; remove: 66; task: 626643

search: 99; insert: 53; min: 2; remove: 66; task: 619092

search: 103; insert: 55; min: 2; remove: 67; task: 622733

search: 104; insert: 52; min: 3; remove: 65; task: 626529

Test #28

Input data size: 5600 elements

search: 103; insert: 56; min: 2; remove: 67; task: 651291

search: 105; insert: 52; min: 2; remove: 68; task: 648448

search: 105; insert: 54; min: 3; remove: 70; task: 651184

search: 100; insert: 57; min: 2; remove: 65; task: 653070

search: 110; insert: 52; min: 1; remove: 67; task: 649455

Test #29

Input data size: 5800 elements

search: 110; insert: 54; min: 2; remove: 68; task: 678789

search: 98; insert: 57; min: 2; remove: 67; task: 671589

search: 106; insert: 56; min: 2; remove: 66; task: 677112

search: 109; insert: 50; min: 2; remove: 71; task: 676974

search: 106; insert: 56; min: 2; remove: 67; task: 679454

Test #30

Input data size: 6000 elements

search: 104; insert: 55; min: 2; remove: 68; task: 705250

search: 104; insert: 56; min: 2; remove: 69; task: 702082

search: 108; insert: 55; min: 2; remove: 67; task: 698039

search: 111; insert: 57; min: 2; remove: 68; task: 706203

search: 106; insert: 53; min: 2; remove: 70; task: 702098

Test #31

Input data size: 6200 elements

search: 103; insert: 58; min: 2; remove: 67; task: 723701

search: 108; insert: 55; min: 2; remove: 72; task: 731251

search: 109; insert: 55; min: 2; remove: 70; task: 732938

search: 108; insert: 55; min: 2; remove: 68; task: 730460

search: 110; insert: 55; min: 2; remove: 67; task: 726050

Test #32

Input data size: 6400 elements

search: 109; insert: 56; min: 2; remove: 71; task: 759300

search: 106; insert: 55; min: 2; remove: 69; task: 752195

search: 114; insert: 59; min: 2; remove: 69; task: 754301

search: 108; insert: 54; min: 2; remove: 69; task: 754362

search: 104; insert: 56; min: 2; remove: 69; task: 755434

Test #33

Input data size: 6600 elements

search: 104; insert: 57; min: 2; remove: 70; task: 779017

search: 112; insert: 59; min: 3; remove: 68; task: 785246

search: 110; insert: 54; min: 2; remove: 70; task: 777472

search: 110; insert: 58; min: 2; remove: 69; task: 781167

search: 109; insert: 54; min: 2; remove: 72; task: 784943

Test #34

Input data size: 6800 elements

search: 112; insert: 55; min: 2; remove: 73; task: 811705

search: 108; insert: 56; min: 2; remove: 69; task: 805062

search: 108; insert: 60; min: 2; remove: 68; task: 811494

search: 111; insert: 57; min: 2; remove: 74; task: 802085

search: 111; insert: 56; min: 3; remove: 68; task: 809540

Test #35

Input data size: 7000 elements

search: 110; insert: 58; min: 3; remove: 71; task: 830208

search: 114; insert: 61; min: 2; remove: 70; task: 836012

search: 106; insert: 56; min: 2; remove: 72; task: 836719

search: 109; insert: 57; min: 2; remove: 69; task: 830548

search: 114; insert: 54; min: 2; remove: 72; task: 838405

Test #36

Input data size: 7200 elements

search: 112; insert: 57; min: 2; remove: 72; task: 858799

search: 112; insert: 58; min: 3; remove: 70; task: 862137

search: 114; insert: 59; min: 2; remove: 71; task: 856283

search: 107; insert: 58; min: 2; remove: 73; task: 867240

search: 112; insert: 56; min: 2; remove: 70; task: 860826

Test #37

Input data size: 7400 elements

search: 108; insert: 59; min: 2; remove: 69; task: 888099

search: 113; insert: 58; min: 1; remove: 75; task: 889596

search: 110; insert: 56; min: 3; remove: 73; task: 883729

search: 113; insert: 58; min: 2; remove: 70; task: 885799

search: 117; insert: 59; min: 3; remove: 71; task: 890824

Test #38

Input data size: 7600 elements

search: 113; insert: 58; min: 2; remove: 74; task: 919531

search: 113; insert: 57; min: 2; remove: 71; task: 918809

search: 114; insert: 59; min: 2; remove: 71; task: 909517

search: 114; insert: 59; min: 3; remove: 72; task: 909647

search: 110; insert: 58; min: 2; remove: 72; task: 914067

Test #39

Input data size: 7800 elements

search: 114; insert: 61; min: 3; remove: 73; task: 944874

search: 116; insert: 58; min: 2; remove: 70; task: 940415

search: 111; insert: 58; min: 2; remove: 75; task: 940289

search: 113; insert: 57; min: 2; remove: 71; task: 938061

search: 114; insert: 59; min: 2; remove: 73; task: 941903

Test #40

Input data size: 8000 elements

search: 112; insert: 60; min: 2; remove: 73; task: 967836

search: 115; insert: 59; min: 3; remove: 71; task: 966741

search: 115; insert: 58; min: 2; remove: 72; task: 971786

search: 116; insert: 60; min: 2; remove: 74; task: 969494

search: 113; insert: 58; min: 2; remove: 74; task: 964321

Test #41

Input data size: 8200 elements

search: 115; insert: 60; min: 2; remove: 72; task: 999089

search: 110; insert: 61; min: 2; remove: 75; task: 995411

search: 116; insert: 60; min: 3; remove: 73; task: 989244

search: 115; insert: 57; min: 2; remove: 73; task: 995055

search: 118; insert: 58; min: 2; remove: 73; task: 995716

Test #42

Input data size: 8400 elements

search: 116; insert: 59; min: 2; remove: 75; task: 1019310

search: 114; insert: 57; min: 2; remove: 74; task: 1027030

search: 114; insert: 60; min: 2; remove: 75; task: 1024430

search: 119; insert: 60; min: 3; remove: 72; task: 1016411

search: 114; insert: 62; min: 2; remove: 70; task: 1022398

Test #43

Input data size: 8600 elements

search: 115; insert: 62; min: 3; remove: 74; task: 1050813

search: 118; insert: 60; min: 2; remove: 73; task: 1052879

search: 115; insert: 59; min: 2; remove: 73; task: 1048141

search: 116; insert: 61; min: 2; remove: 73; task: 1042086

search: 116; insert: 58; min: 2; remove: 76; task: 1051345

Test #44

Input data size: 8800 elements

search: 118; insert: 61; min: 2; remove: 74; task: 1075368

search: 116; insert: 59; min: 2; remove: 75; task: 1078520

search: 117; insert: 61; min: 3; remove: 74; task: 1079709

search: 115; insert: 59; min: 2; remove: 76; task: 1075117

search: 117; insert: 61; min: 2; remove: 72; task: 1071501

Test #45

Input data size: 9000 elements

search: 117; insert: 61; min: 2; remove: 76; task: 1106084

search: 120; insert: 62; min: 3; remove: 73; task: 1101307

search: 117; insert: 60; min: 2; remove: 75; task: 1102475

search: 117; insert: 60; min: 2; remove: 73; task: 1105291

search: 115; insert: 60; min: 3; remove: 75; task: 1100939

Test #46

Input data size: 9200 elements

search: 116; insert: 60; min: 3; remove: 75; task: 1131999

search: 115; insert: 58; min: 2; remove: 75; task: 1130397

search: 121; insert: 63; min: 2; remove: 75; task: 1131908

search: 118; insert: 62; min: 3; remove: 75; task: 1123332

search: 119; insert: 61; min: 2; remove: 75; task: 1134803

Test #47

Input data size: 9400 elements

search: 120; insert: 63; min: 2; remove: 72; task: 1152591

search: 114; insert: 63; min: 3; remove: 78; task: 1158645

search: 119; insert: 59; min: 2; remove: 73; task: 1159232

search: 120; insert: 60; min: 3; remove: 76; task: 1153403

search: 118; insert: 61; min: 2; remove: 77; task: 1164997

Test #48

Input data size: 9600 elements

search: 119; insert: 62; min: 2; remove: 76; task: 1185397

search: 123; insert: 62; min: 2; remove: 75; task: 1180861

search: 116; insert: 62; min: 3; remove: 80; task: 1189784

search: 119; insert: 59; min: 2; remove: 74; task: 1189122

search: 118; insert: 62; min: 3; remove: 73; task: 1180415

Test #49

Input data size: 9800 elements

search: 118; insert: 60; min: 3; remove: 79; task: 1210183

search: 116; insert: 60; min: 2; remove: 72; task: 1212570

search: 119; insert: 63; min: 2; remove: 75; task: 1208000

search: 120; insert: 61; min: 3; remove: 76; task: 1214107

search: 124; insert: 65; min: 2; remove: 78; task: 1217894

Test #50

Input data size: 10000 elements

search: 122; insert: 62; min: 2; remove: 77; task: 1241066

search: 119; insert: 63; min: 3; remove: 80; task: 1239214

search: 123; insert: 61; min: 3; remove: 75; task: 1241148

search: 119; insert: 64; min: 2; remove: 75; task: 1236791

search: 116; insert: 60; min: 2; remove: 74; task: 1241811