**Прізвище: Пагута**

**Ім’я: Віталій**

**Група:** КН-108

**Варіант: 22**

**Дата захисту:**

**Кафедра:** САП

**Дисципліна:** Алгоритмізація та програмування. Ч.2

**Перевірив:** Андрійчук М.І.

**Звіт**

До лабораторної роботи №6

На тему “ ДИНАМІЧНІ ОБ’ЄКТИ СКЛАДНОЇ СТРУКТУРИ. БІНАРНІ ДЕРЕВА ПОШУКУ”

**Мета роботи**: Ознайомитися iз способом подання даних в оперативнiй пам'ятi ЕОМ у виглядi збалансованих бінарних (АВЛ) дерев. Оволодiти методами роботи iз збалансованими бінарними деревами.

**Відповіді на контрольні запитання:**

1. *Які дерева називаються виродженими?*

Вироджене дерево-дерево, яке є лінійним списком елементів, впорядкованих зазбільшенням або зменшенням інформаційних полів.

1. *Яке бінарне дерево називається ідеально збалансованим?*Бінарне дерево називається ідеально збалансованим, якщо кількість вершин в йоголівому і правому піддеревах відрізняється не більш, ніж на 1.
2. *Як можна оцiнити кiлькiсть варiантiв структур бiнарних дерев? Скільки серед них будуть ідеально збалансованими?*

 Кiлькiсть варiантiв структур бiнарних дерев можна приблизно оцiнитиза допомогою формули Стирлiнґа:



Якщо кiлькiсть ключів n=15±серед отриманих 9’694’845 бiнарнихдерев лише одне буде повнiстю збалансованим. Якщо кiлькiсть ключів n=16±серед 35’357’670 бiнарних дерев лише вiсiм будуть повністю збалансованими.

1. *Чому на практиці ідеально збалансовані дерева пошуку використовуються рідко?*

На практиці ідеально збалансовані дерева пошуку використовуються вкрай рідко тому, що підтримка дерева пошуку в ідеально збалансованому стані вимагає істотного ускладнення операцій включення і виключення ключів. Крім того, при рівномірному розподілі значень ключів, що додаються і видаляються, використання ідеально збалансованих дерев пошуку дає виграш не більше 30%.

1. *Що таке АВЛ-дерево?*

АВЛ-дерево — збалансоване по висоті двійкове дерево пошуку: для кожної його вершини висота її двох піддерев відрізняється не більше ніж на 1.

1. *Як визначити показник зблансованостi вузла?*

Щодо АВЛ-дерева балансуванням вершини називається операція, яка у разі різниці висот лівого і правого піддерев = 2, змінює зв'язку предок-нащадок в піддереві даної вершини так, що різниця стає <= 1, інакше нічого не змінює.

1. *Як здiйснюється додавання вузла до АВЛ-дерева? Як виконуються процедури збалансування?*

Показник збалансованості в подальшому будемо інтерпретувати як різниця між висотою лівого і правого піддерева, а алгоритм буде заснований на типі TAVLTree, описаному вище. Безпосередньо при вставці (листу) присвоюється нульовий баланс. Процес включення вершини складається з трьох частин:

1. Прохода по шляху пошуку, поки не переконаємося, що ключа в дереві немає.
2. Включення нової вершини у дерево і визначення результуючих показників балансування.
3. «Відступи» назад по шляху пошуку і перевірки в кожній вершині показника збалансованості. Якщо необхідно — балансування
4. *Як здiйснюється видлення вузла з АВЛ-дерева?*

Для простоти опишемо рекурсивний алгоритм видалення. Якщо вершина — листок, то видалимо її і викличемо балансування всіх її предків в порядку від батька до кореня. Інакше знайдемо саму близьку за значенням вершину в піддереві найбільшої висоти (правому або лівому) і перемістимо її на місце видаляється вершини, при цьому викликавши процедуру її видалення. Спрощений варіант видалення можна описати таким чином:

node\* remove(node\* p, int k) // видалення ключа k з дерева p

{

if( !p )

return 0;

if( k < p->key )

p->left = remove(p->left,k);

else if( k > p->key )

p->right = remove(p->right,k);

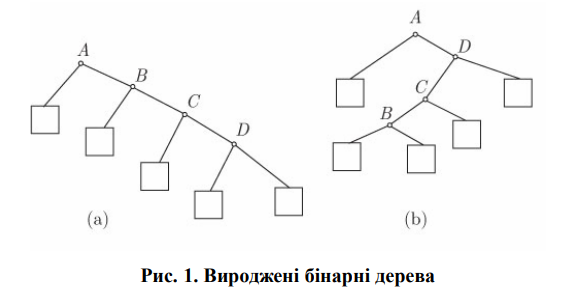
}

1. *Як здійснити доступ до елементу за індексом у АВЛ-дереві?*

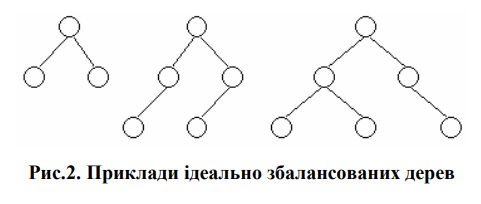
Для того, щоб організувати доступ за індексом, нам буде потрібно зберігати у кожній вершині додаткове числове поле − кількість вершин в лівому піддереві даної вершини, тобто кількість її нащадків, які передують їй по порядку (всі елементи в лівому піддереві є меншими за кореневу вершину).

**Короткі теоретичні відомості:**

Для зберігання великої кількості впорядкованих інформаційних об’єктів звичайно використовують двійкові дерева пошуку [1, 2, 3]. Процедура та алгоритм включення у таке дерево дають хороші результати при використанні випадкових вхідних даних, але все-таки існує можливість того, що при цьому буде побудоване вироджене дерево (рис.1).



Щоб уникнути цього, можна було б розробити алгоритм, що підтримує дерево в оптимальному стані весь час. Бінарне дерево називається ідеально збалансованим, якщо кількість вершин в його лівому і правому піддеревах відрізняється не більш, ніж на 1 (легко бачити, що при дотриманні цієї умови довжини шляху до будь-якої листової вершини дерева відрізняються не більше, ніж на 1). Приклади ідеально збалансованих дерев показані на рис.2.



**Індивідуальне завдання:**

22. Написати програму, яка створює збалансоване бінарне дерево. Написати рекурсивну процедуру, яка визначає кількість входжень заданого елемента у дерево.

Програма:

#include <iostream>

#include<iomanip>

#include <time.h>

#include <vector>

#include <fstream>

using namespace std;

vector<int> numbers;

//Структура, що описує вузол

struct node

{

public:

int height, data;

node\* leftChild, \* rightChild;

};

//Початковий вказівник для батьківського вузла

node\* root = NULL;

//Визначення висоти дерева

int height(node\* p)

{

if (p == NULL)

return -1;

else

{

p->height = max(height(p->leftChild), height(p->rightChild)) + 1;

}

return p->height;

}

//Допоміжна функція

node\* newNode(int element)

{

node\* newPtr = new node;

newPtr->data = element;

newPtr->leftChild = NULL;

newPtr->rightChild = NULL;

newPtr->height = 1;

return newPtr;

}

//Функція правого повороту

node\* rightRotate(node\* p)

{

node\* child = p->leftChild;

p->leftChild = child->rightChild;

child->rightChild = p;

p->height = height(p);

child->height = height(child);

return child;

}

//Функція лівого повороту

node\* leftRotate(node\* p)

{

node\* child = p->rightChild;

p->rightChild = child->leftChild;

child->leftChild = p;

p->height = height(p);

child->height = height(child);

return child;

}

//Функція збалансування

int getBalance(node\* p)

{

if (p == NULL)

return 0;

else

return height(p->leftChild) - height(p->rightChild);

}

node\* insert(node\*& p, int element)

{

if (p == NULL)

return(newNode(element));

if (element < p->data) {

p->leftChild = insert(p->leftChild, element);

}

else

{

p->rightChild = insert(p->rightChild, element);

}

p->height = height(p);

int balance = getBalance(p);

if (balance > 1 && element < p->leftChild->data)

{

return rightRotate(p);

}

if (balance < -1 && element > p->rightChild->data)

{

return leftRotate(p);

}

if (balance > 1 && element > p->leftChild->data)

{

p->leftChild = leftRotate(p->leftChild);

return rightRotate(p);

}

if (balance < -1 && element < p->rightChild->data)

{

p->rightChild = rightRotate(p->rightChild);

return leftRotate(p);

}

return p;

}

//Функції обходу дерева

void inorder(node\* p)

{

if (p != NULL)

{

inorder(p->leftChild);

cout << p->data << ", ";

inorder(p->rightChild);

}

}

void preorder(node\* p)

{

if (p != NULL)

{

cout << p->data << ", ";

preorder(p->leftChild);

preorder(p->rightChild);

}

}

//Роздрук елементів

void print(node\* root)

{

cout << endl << "Inorder: ";

inorder(root);

cout << endl << endl << endl << endl;

}

//Функція зчитування з файлу

void read() {

int num;

ifstream file\_save("data.txt");

if (file\_save.is\_open())

{

while (!file\_save.eof())

{

file\_save >> num;

numbers.push\_back(num);

}

file\_save.close();

} else {

cout << "Error in opening file!!" << endl;

}

}

void ShowBackSymmetric(node\* current, int l)

{

if (current != NULL)

{

ShowBackSymmetric(current->rightChild, l + 1);

for (int i = 0; i < l; i++)

cout << "\t";

cout << current->data << endl;

ShowBackSymmetric(current->leftChild, l + 1);

}

}

int CalculateAmountOf(int info) {

node\* current = root;

int flag = 0;

while (current != NULL)

{

if (info < current->data)

current = current->leftChild;

else if (info > current->data)

current = current->rightChild;

else {

flag++;

}

}

return flag;

}

int amountOf = 0;

void CalculateAmountOfRekurs(node\* p, int info) {

if (p != NULL) {

if (info < p->data)

CalculateAmountOfRekurs(p->leftChild, info);

else if (info > p->data)

CalculateAmountOfRekurs(p->rightChild, info);

else {

amountOf++;

CalculateAmountOfRekurs(p->leftChild, info);

CalculateAmountOfRekurs(p->rightChild, info);

}

}

}

//Приклад головної функції

int main()

{

double duration;

time\_t begin = time(0);

read();

int x = 0;

int track = 0;

for (std::vector<int>::const\_iterator i = numbers.begin(); i != numbers.end(); ++i)

{

root = insert(root, numbers[x]);

x++;

track++;

/\*if ((track % 10000) == 0)

{

cout << track << " iterations" << endl;

time\_t now = time(0);

cout << now - begin << " seconds" << endl;

}\*/

}

time\_t end = time(0);

duration = end - begin;

print(root);

ShowBackSymmetric(root,0);

/\*cout << "-----------------------------------------------------------------------\n";

root = insert(root, 100);

ShowBackSymmetric(root, 0);

cout << "-----------------------------------------------------------------------\n";

root = insert(root, 110);

ShowBackSymmetric(root, 0);

cout << "-----------------------------------------------------------------------\n";

root = insert(root, 120);

ShowBackSymmetric(root, 0);

cout << "-----------------------------------------------------------------------\n";\*/

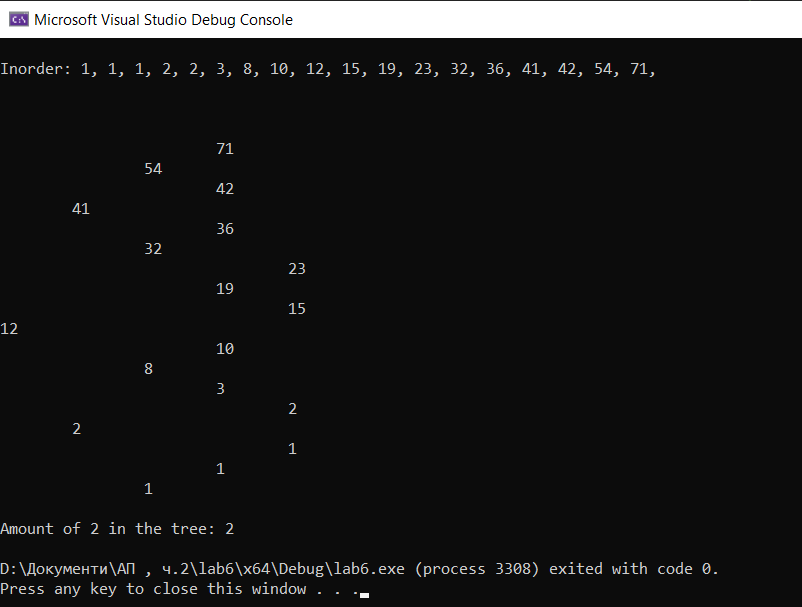
CalculateAmountOfRekurs(root, 2);

cout << endl << "Amount of 2 in the tree: " << amountOf << endl;

return 0;

}

Результат виконання програми:



**Висновок:** я ознайомився iз способом подання даних в оперативнiй пам'ятi ЕОМ у виглядi збалансованих бінарних (АВЛ) дерев та оволодiв методами роботи iз збалансованими бінарними деревами.