**Методичні рекомендації**

**до виконання лабораторної роботи №№13\_14**

**Тема:**Складання та налагодження програм побудови збалансованого бінарного дерева пошуку та реалізація пошуку ключового елементу.

**Мета:** Отримання навиків в організації динамічної структури даних – збалансованого бінарного дерева пошуку реалізації пошуку ключового елементу.

ТЕОРЕТИЧНІ ВІДОМОСТІ

АВЛ-дерево - це насамперед бінарне дерево пошуку, ключі якого задовольняють стандартній властивості: ключ будь-якого вузла дерева не менше будь-якого ключа в лівому піддереві даного вузла й не більше будь-якого ключа в правому піддереві цього вузла. Це значить, що для пошуку потрібного ключа в АВЛ-дереві можна використати стандартний алгоритм. Для простоти подальшого викладу будемо вважати, що всі ключі в дереві цілочисельні й не повторюються.

Бінарне дерево називається збалансованим(АВЛ-деревом), якщо висота лівого піддерева кожного вузла відрізняється від висоти правого не більше ніж на одиницю.

Будемо представляти вузли АВЛ-дерева наступною структурою:

struct node

{

int info;

int h;

node \* left;

node \* right;

};

Поле info зберігає ключ вузла, поле h - висоту піддерева з коренем у даному вузлі, полючиleft й rіght - покажчики на ліве й праве піддерева.

Традиційно, вузли АВЛ-дерева зберігають не висоту, а різницю висот правого й лівого піддерев (так званий balancefactor), що може приймати тільки три значення -1, 0 й 1.

Визначимо три допоміжні функції, пов'язані з висотою.

***Перша*** є обгорткою для поля h, вона може працювати й з нульовими покажчиками (з порожніми деревами). Вона повертає висоту дерева(піддерева):

intheight(node\* p)

{ return p->h;}

***Друга*** обчислює balancefactor заданого вузла (і працює тільки з ненульовими покажчиками):

intbfactor (node\* p)

{ returnheight(p->right)-height(p->left); }

***Третя*** функція відновлює коректне значення поля h заданого вузла (за умови, що значення цього поля в правом і левом дочірніх вузлах є коректними):

voidfixheight (node\* p)

{ int h1=height(p->left);

int h2=height(p->right);

if (h1>h2) p->h=h1+1;

else p->h=h2+1;

2 Балансування вузлів

У процесі додавання або видалення вузлів в АВЛ-дереві можливе виникнення ситуації, коли balancefactor деяких вузлів виявляється рівними 2 або -2, тобто виникає розбалансуванняпіддерева. Для виправлення ситуації застосовуються повороти навколо тих або інших вузлів дерева. ***Простий поворот*** вправо (уліво) робить наступну трансформацію дерева (рисунок 2):

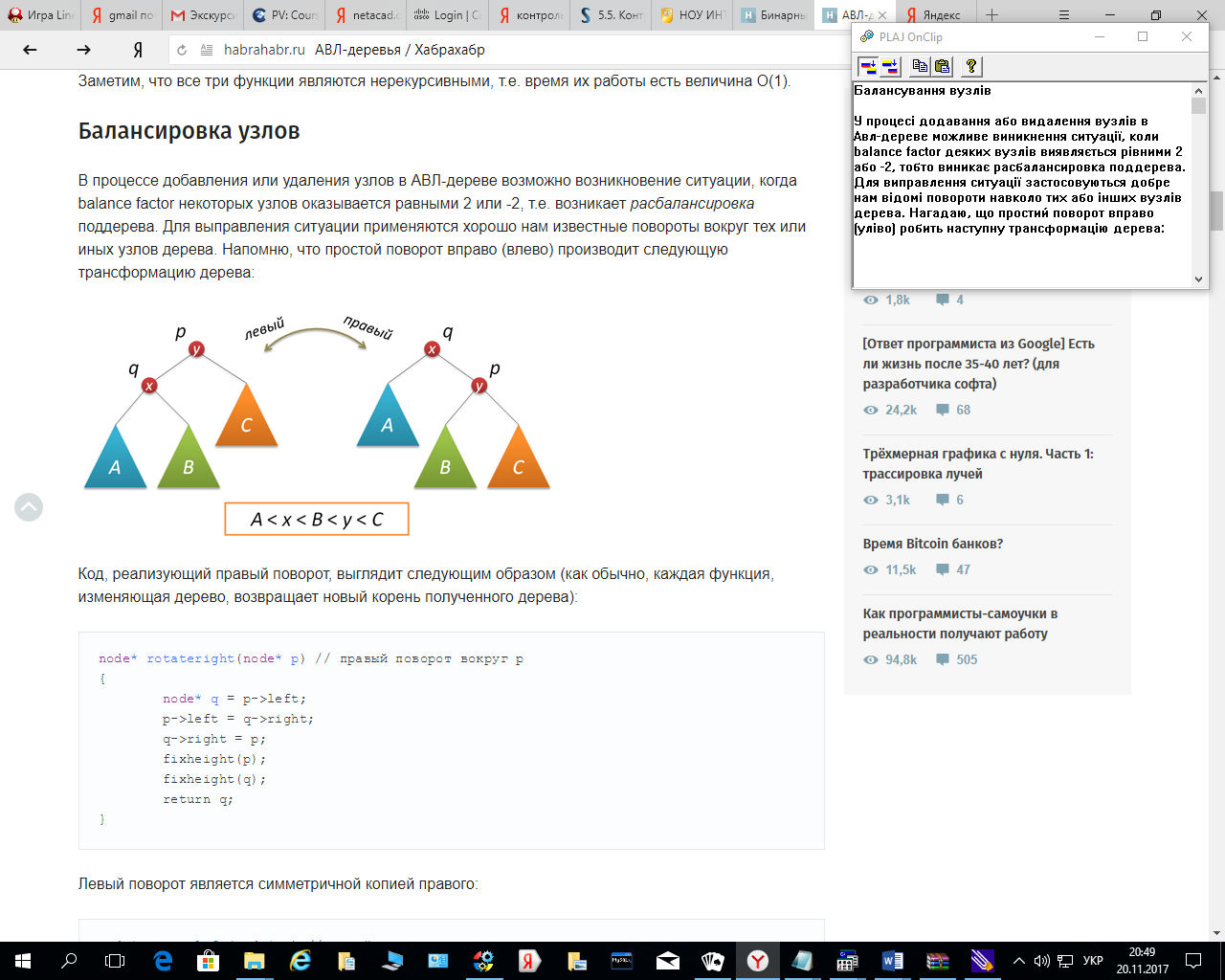


Рис. 2. Правий та лівий прості повороти.

Код, що реалізує правий поворот, виглядає в такий спосіб(лістинг 1).Як звичайно, кожна функція, що змінює дерево, повертає новий корінь отриманого дерева:

Лістинг 1. Реалізація правого простого повороту.

node\* rotateright(node\* p) // правий поворот навколо p

{

node\* q = p->left;

p->left = q->right;

q->right = p;

fixheight(p);

fixheight(q);

return q;

}

Лівий поворот є симетричною копією правого (лістинг 2):

Лістинг 2. Реалізація лівого просторо повороту.

node\* rotateleft(node\* q) // левый поворот вокруг q

{

node\* p = q->right;

q->right = p->left;

p->left = q;

fixheight(q);

fixheight(p);

return p;

}

Розглянемо тепер ситуацію дисбалансу, коли висота правого піддерева вузла p на 2 більше висоти лівого піддерева (зворотний випадок є симетричним і реалізується аналогічно). Нехай q - правий дочірній вузол вузла p, а s - лівий дочірній вузол вузла q (рисунок 3).

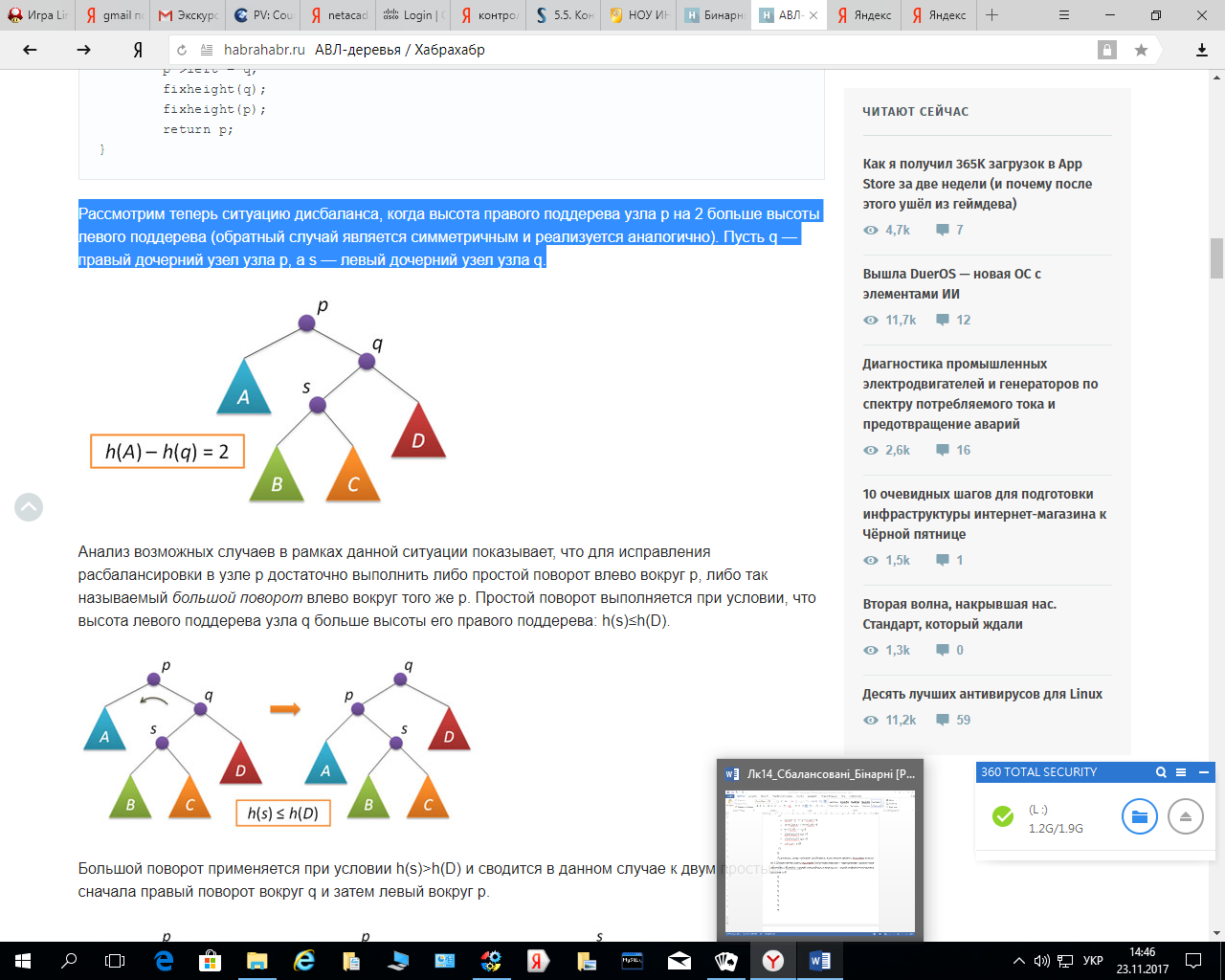


Рис. 3. Розбалансоване бінарне дерево пошуку.

Аналіз можливих випадків у рамках даної ситуації показує, що для виправлення розбалансування у вузлу p досить виконати або простий поворот уліво навколо p, або так званий великий поворот уліво навколо того ж p. Простий поворот виконується за умови, що висота лівого піддерева вузла q більше висоти його правого піддерева: h(s)≤(D) (рисунок 4).

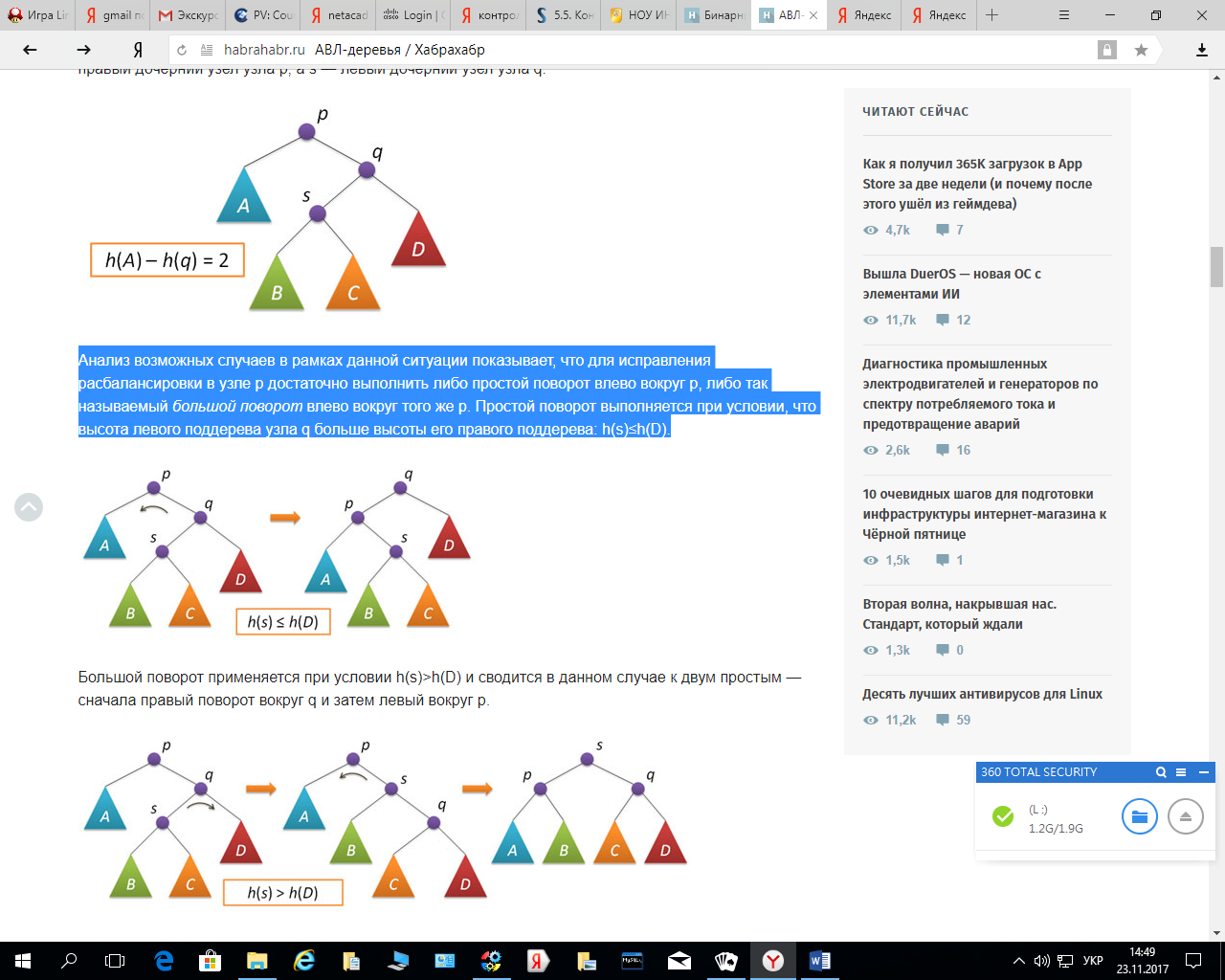


Рис. 4. Простий лівий поворот навколо вузла Р.

Великий поворот застосовується за умови h(s)>h(D) і зводиться в цьому випадку до двох простих поворотів - спочатку правий поворот навколо q і потім лівий навколо p (рисунок 5).

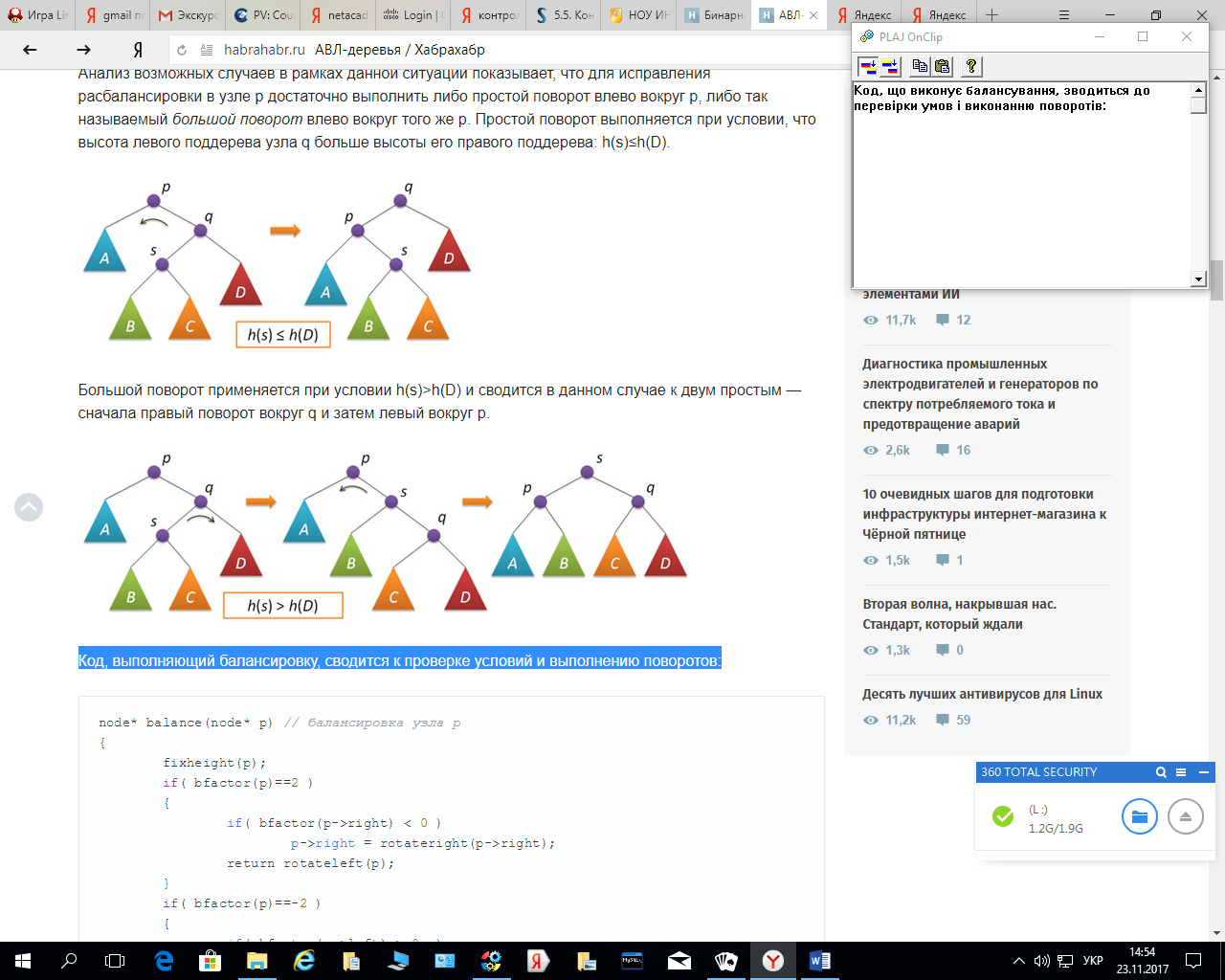


Рис. 5. Великий поворот навколо Р.

Код, що виконує балансування, зводиться до перевірки умов і виконанню поворотів наведено у лістингу 3.

Лістинг 3. Функція балансування вузлу бінарного дерева пошуку.

node\* balance(node\* p) // балансировкаузла p

{

fixheight(p);

if( bfactor(p)==2 )

{

if( bfactor(p->right) < 0 )

p->right = rotateright(p->right);

returnrotateleft(p);

}

if( bfactor(p)==-2 )

{

if( bfactor(p->left) > 0 )

p->left = rotateleft(p->left);

returnrotateright(p);

}

return p; // балансировка не нужна

}

Описані функції поворотів і балансування також не містять ні циклів, ні рекурсії, а значить виконуються за постійний час, що не залежить від розміру Авл-дерева.

3 Вставка ключів

Вставка нового ключа в АВЛ-дерево виконується, по великому рахунку, так само, як це робиться в простих деревах пошуку: спускаємося вниз по дереву, вибираючи правий або лівий напрямок руху залежно від результату порівняння ключа в поточному вузлі й ключа, що вставляється. Єдина відмінність полягає в тім, що при поверненні з рекурсії (тобто після того, як ключ вставлений або в праве, або в ліве поддерево, і це дерево збалансоване) виконується балансування поточного вузла. Строго доводиться, що виникаючий при такій вставці дисбаланс у будь-якому вузлі по шляху руху не перевищує двох, а значить застосування вищеописаної функції балансування є коректним.

Код функції вставки нового вузла в збалансоване дерево пошуку наведено в лістингу 4.

Лістинг 4. Функція вставки нового вузла.

node\* insert(node\* p, int k) // вставка ключа k в дерево с корнем p

{

if( p==NULL )

{ p=new node;

p->info=k;

p->h=0;

p->left=NULL;

p->right=NULL;

}

else

if( k<p->info )

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

else

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

return balance(p);

}

ЗМІСТ І ПОСЛІДОВНІСТЬ ВИКОНАННЯ ЗАВДАНЬ

1. Провести аналіз поставленої задачі.

Загальна постановка завдання:

Скласти та налагодити програму створення збалансованого бінарного дерева на основі нелінійного списку та пошуку ключового елементу. Програма повинна задовольняти наступним вимогам:

* організувати користувацьке меню, яке повинно містити наступні пункти:

1. Створення збалансованого бінарного дерева.
2. Перегляд вмісту бінарного дерева.
3. Пошук ключового елементу.
4. Видалення дерева.

* забезпечити коректне введення користувачем вхідних даних;

− при обробці дерева враховувати, що шукані елементи можуть бути відсутні. В цьому випадку вивести користувачеві відповідне повідомлення;

− введення та виведення вхідних та вихідних даних повинно містити необхідні для користувача повідомлення.

1. Розробити та налагодити програму рішення задачі.
2. Оформити звіт з лабораторної роботи.

**Звіт повинен містити наступні розділи:**

1. Постановка задачі.
2. Текст програми з відповідними коментарями.
3. Копії вікон виконання програми.

Цей розділ повинен вміщати наступні скріншоти:

* введення невпорядкованої вхідної послідовності даних;
* побудоване збалансоване бінарне дерево пошуку;
* результати пошуку ключового (заданого користувачем) елементу.

1. Висновок.

КОНТРОЛЬНІ ПИТАННЯ

1. Дайте визначення АВЛ-дерева.
2. Чи є АВЛ-дерево деревом пошуку?
3. Яка висота АВЛ-дерева?
4. Для чого потрібні повороти АВЛ-дерева?
5. Яка трудомісткість включення й виключення вершини АВЛ-дерева?
6. Яке дерево називається деревом пошуку?
7. У чому складається практична важливість використання дерев пошуку?
8. Які переваги має використання дерев пошуку для зберігання впорядкованих даних у порівнянні з масивами й списками?
9. Приведіть алгоритм пошуку в дереві пошуку.
10. Як програмно реалізується пошук у дереві пошуку?

Розробив: Старосельцева О.В.

Розглянуто та схвалено

на засіданні предметної (циклової) комісії

комп’ютерної інженерії

Протокол № 1 від 31.08. 2017 р.

Голова комісії \_\_\_\_\_\_\_\_ Старосельцева О.В.