МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ  
«ХАРКІВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ»

Кафедра «Системний аналіз та інформаційно-аналітичні технології»

Лабораторна робота № 3  
з дисципліни «Алгоритми та структури даних»

(Варіант III)

Виконав:

Олексій ДРАБЧАК

студент групи КН-320Г

Перевірила:

Євгенія МОШКО

Харків 2022

**Тема лабораторної роботи.** Бінарні дерева та червоно-чорні дерева.

**Мета:** Набуття практичних вмінь та навичок опрацювання динамічних

структур даних, представлених у вигляді бінарних та червоно-чорних дерев.

**Порядок виконання роботи:**

1. Написати програму, в якій дані варіанту (таблиця 1) структури записуються в бінарне дерево (використати три поля для вузла – текстове дане та два числові. Наприклад, вузол дерева містить таку корисну інформацію: прізвище студента, рік народження, оцінка по іспиту). Ввести з клавіатури декілька "студентів" у двійкове дерево, організоване за порядком текстового поля. Роздрукувати отримане бінарне дерево.

Індивідуальний варіант представлено на рис. 3.1.

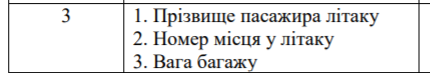


Рисунок 3.1 – Індивідуальний варіант.

1. Знайти середнє значення одного з числових полів, зчитуючи дані з дерева.
2. Надрукувати значення бінарного дерева:

а) при прямому обході дерева;

б) при зворотному обході дерева;

в) при симетричному обході дерева.

1. Дописати функцію видалення з пам’яті всього бінарного дерева.
2. Створити рекурсивну функцію, яка:
   * видаляє ліве піддерево, і ліву гілку занулює;
   * видаляє праве піддерево, і праву гілку занулює;
   * видаляє сам вузол, потім зануливши вказівник на нього.

Наприкінці програми видалити з пам’яті дерево.

1. "Пересипати" дані з першого дерева у друге дерево того ж типу, тільки організованого за першим числовим ключем (напр., рік народження) та роздрукувати його (а перше дерево стерти).
2. Розфарбувати вершини заданого бінарного дерева в червоний і чорний кольори так, щоб воно стало червоно-чорним деревом.
3. Провести перевірку властивості червоно-чорного дерева, а саме:

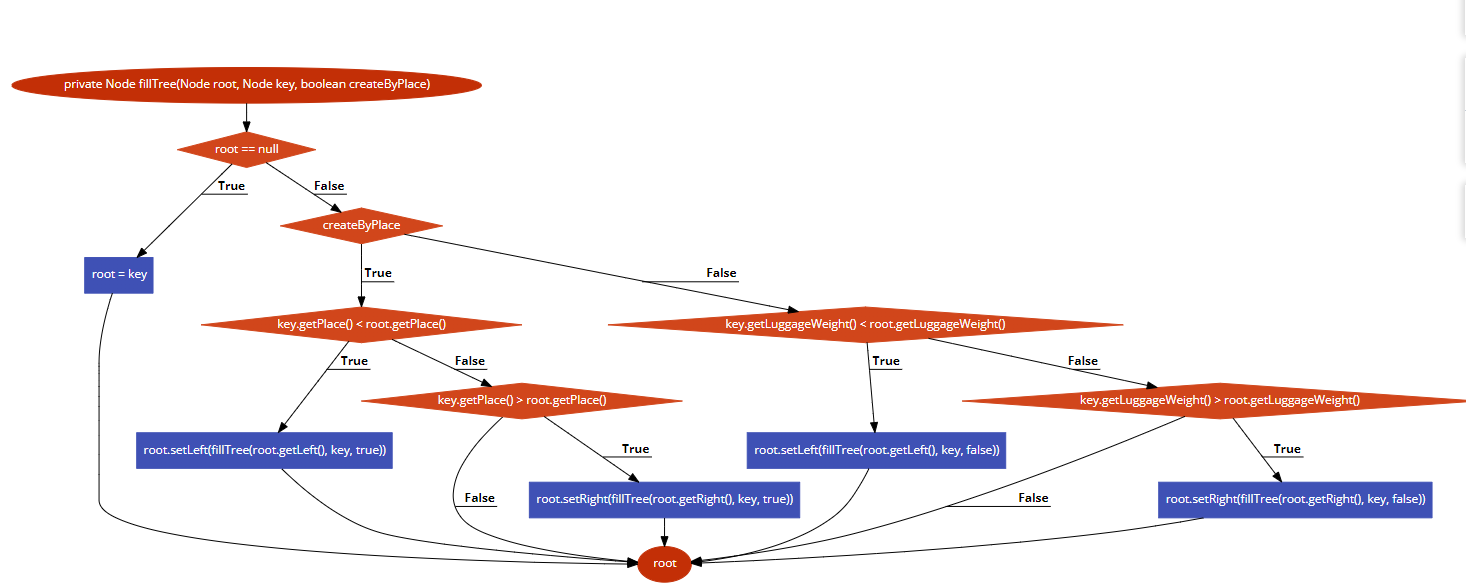
* кожен вузол промаркований червоним або чорним кольором;
* корінь дерева – чорний;
* кінцеві вузли дерева – чорні;
* біля червоного вузла батьківський вузол — чорний;
* усі прості шляхи з будь-якого вузла до листя містять однакову кількість чорних вузлів.

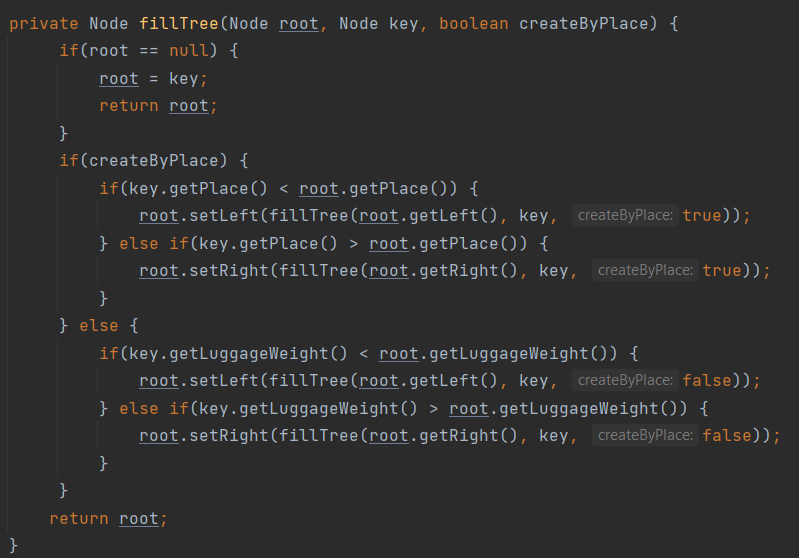
9. У червоно-чорному дереві виконати додавання і видаленні вузлів та провести перевірку властивості червоно-чорного дерева.

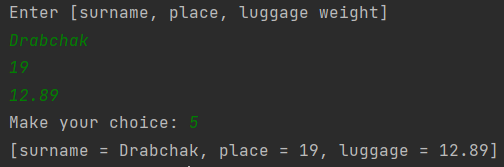
Наприкінці програми видалити з пам’яті дерево.

**Блок-схеми для методів для роботи з двійковим деревом:**

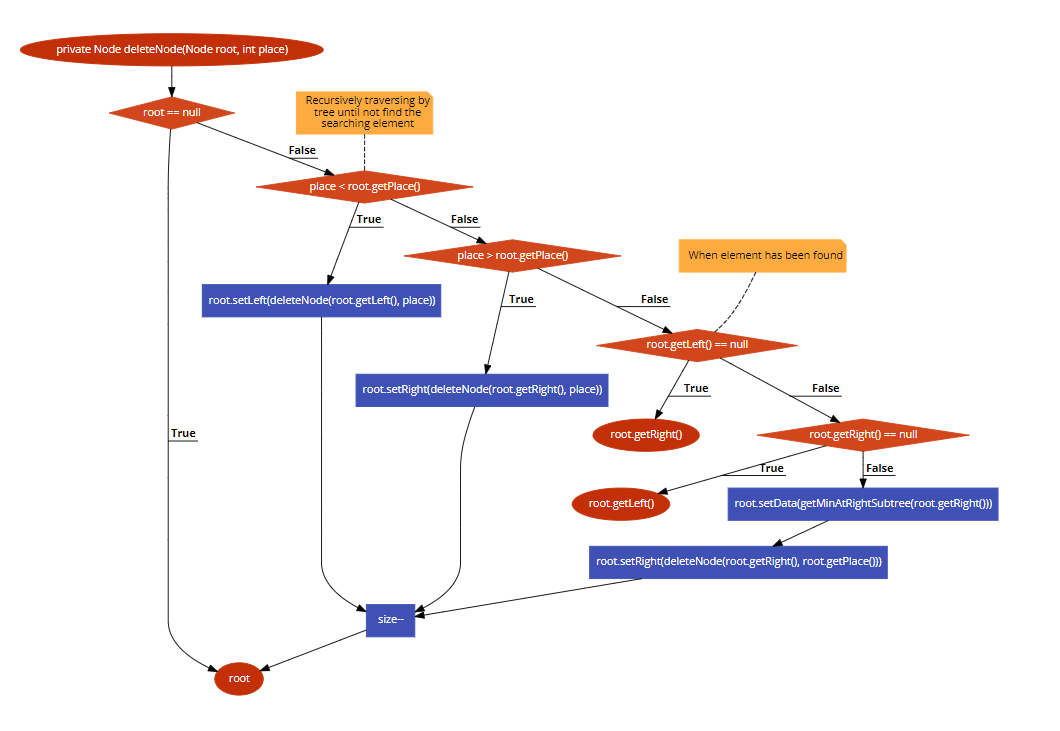
1. Додати елемент в дерево.

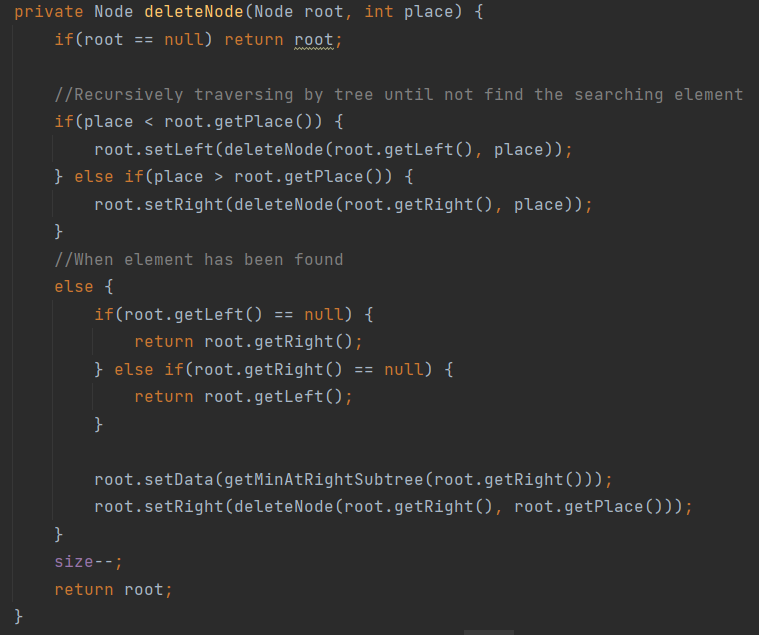


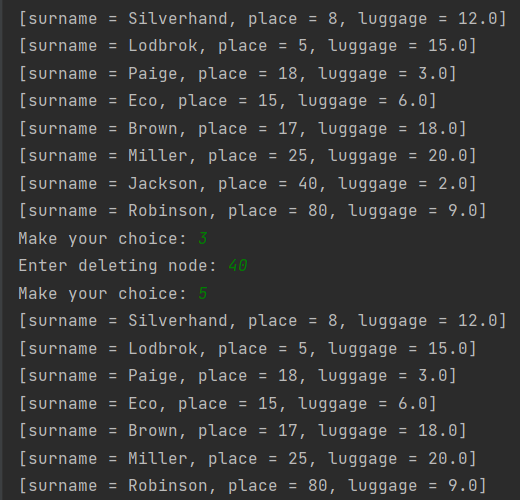




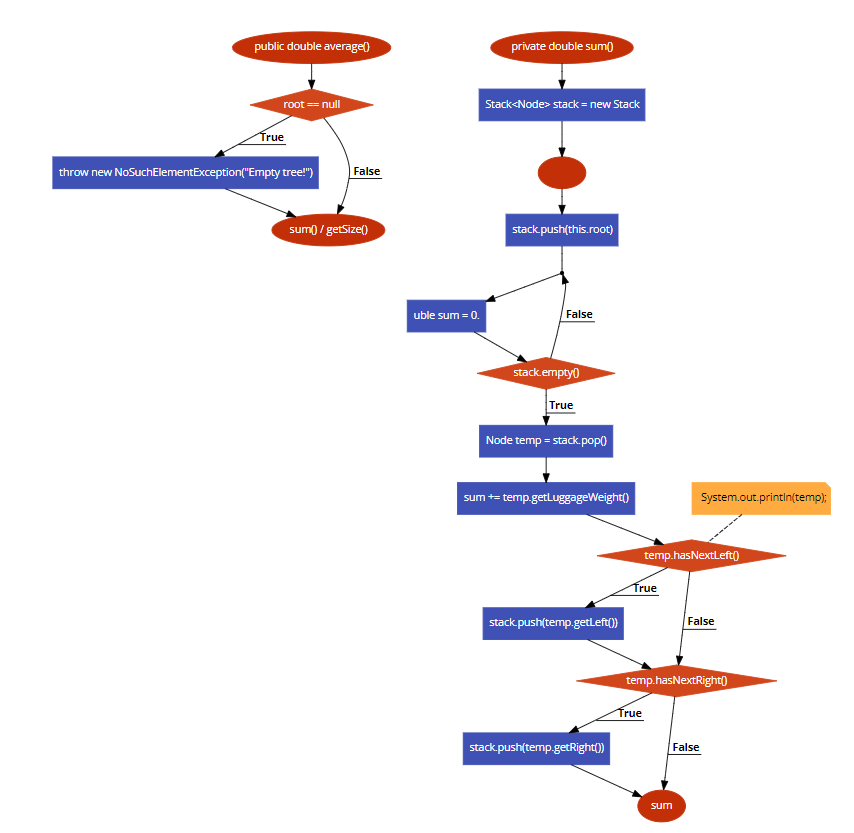
1. Видалення елемента.



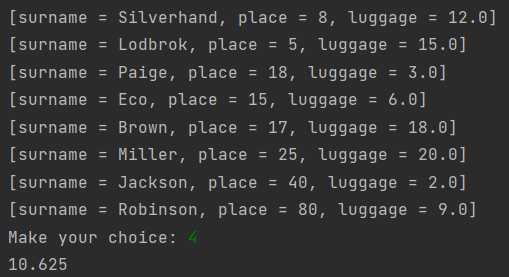




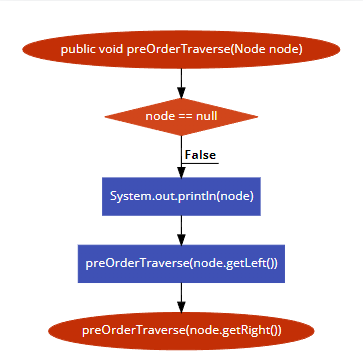
1. Знаходження середнього значення.

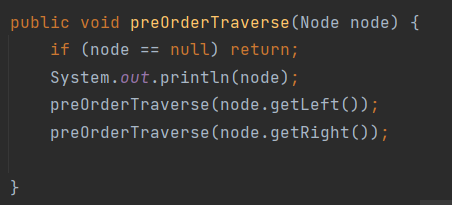


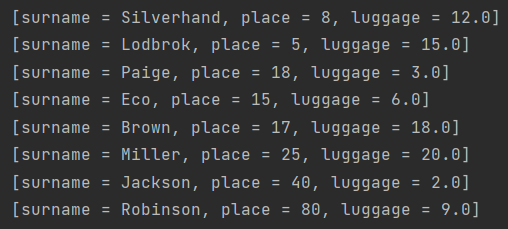




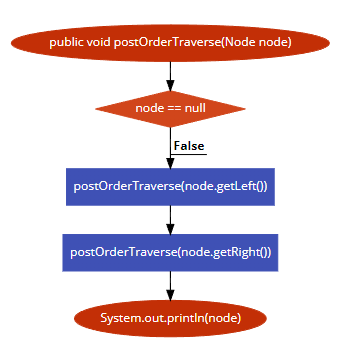
1. Виведення pre-order.

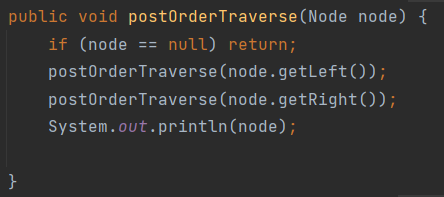


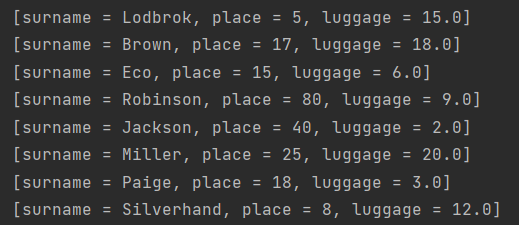




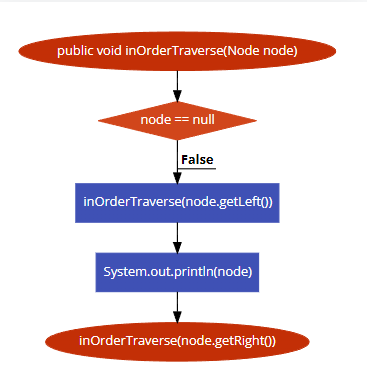
1. Виведення post-order.

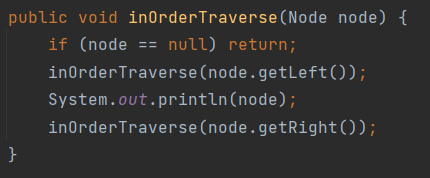


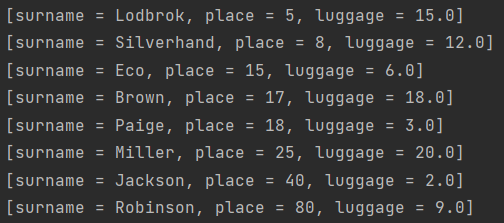




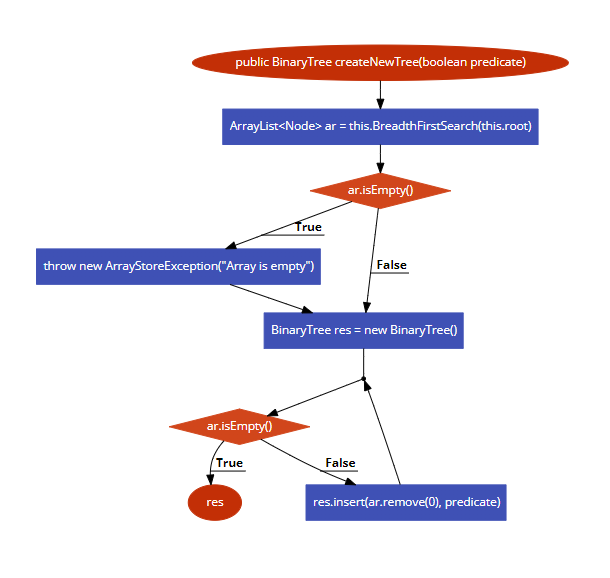
1. Виведення in-order.

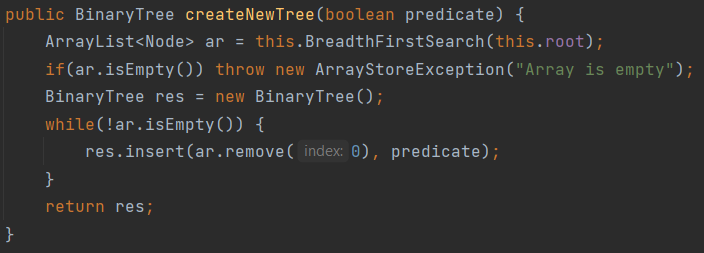


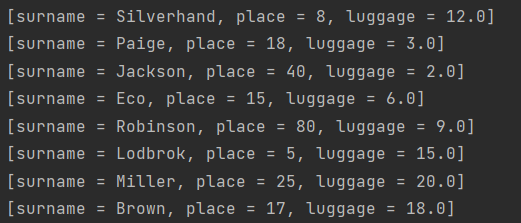




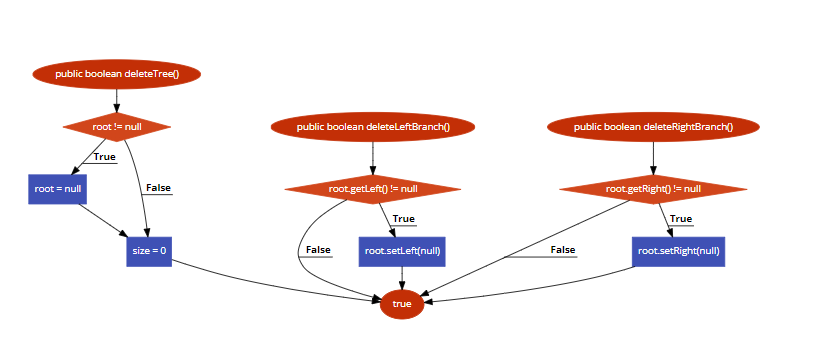
1. Перегрупування дерева по іншому ключу.







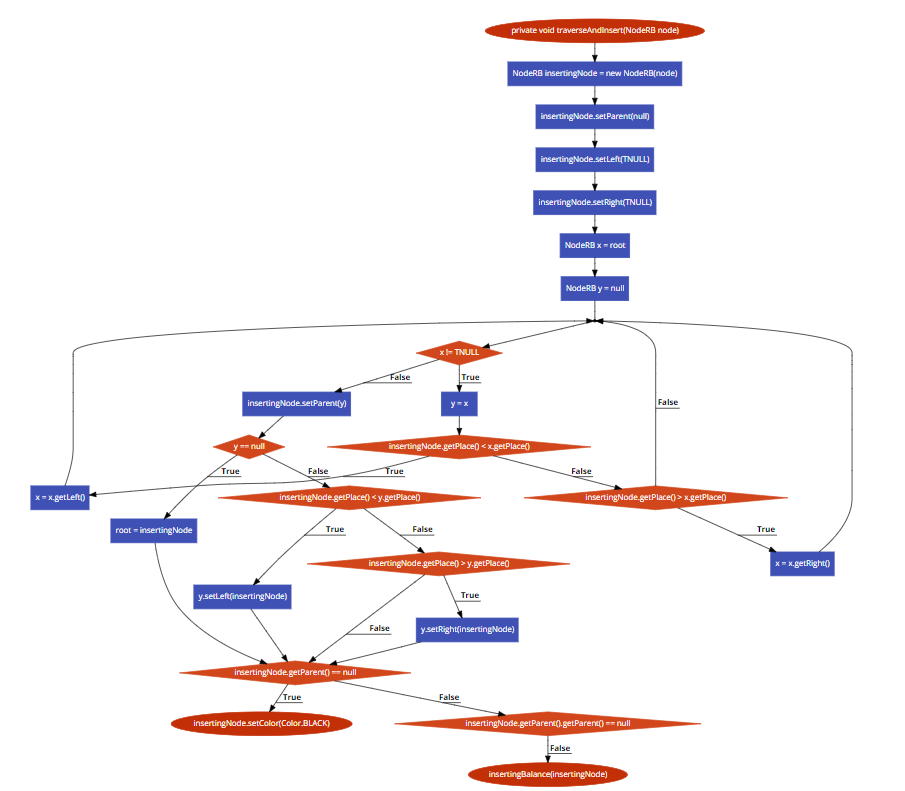
1. Видалення дерева і видалення лівої та правої гілок дерева.

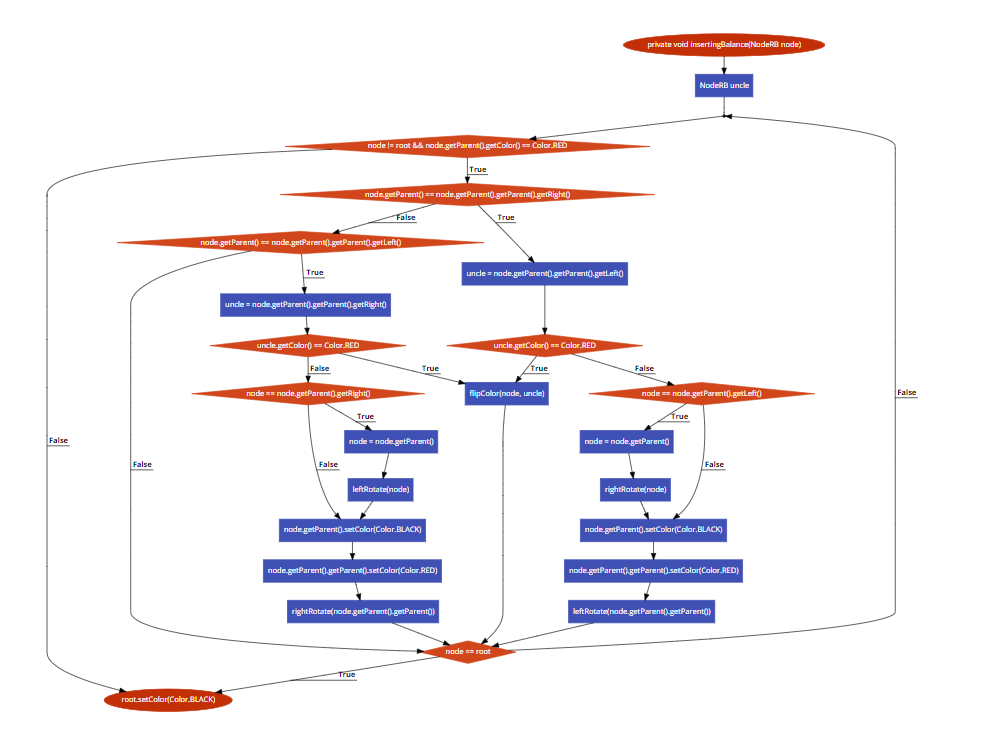


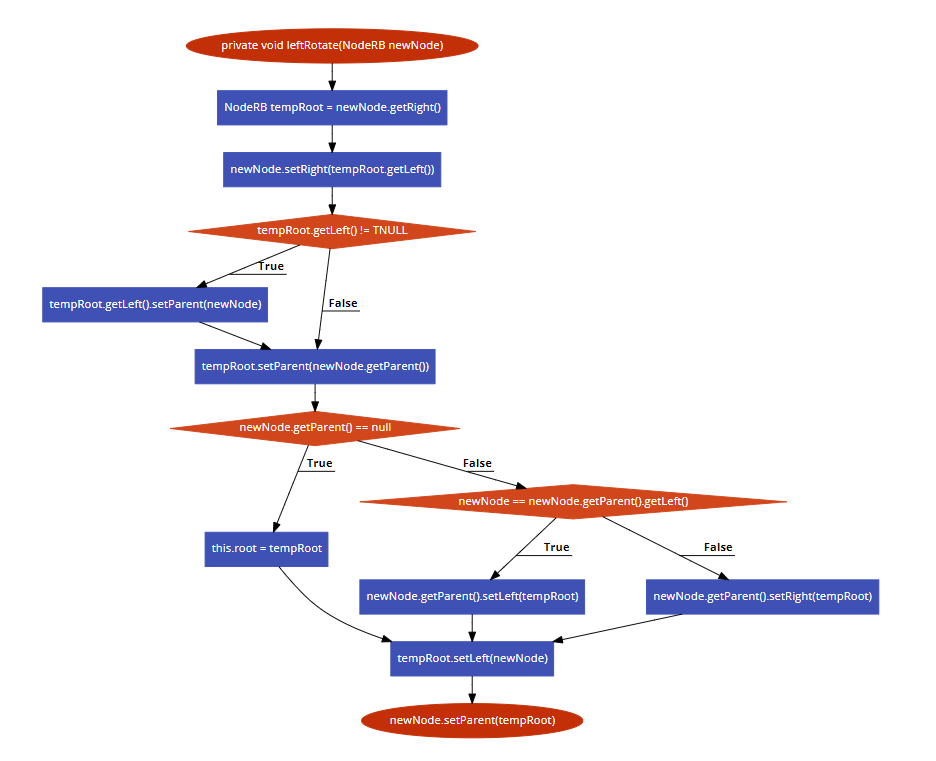


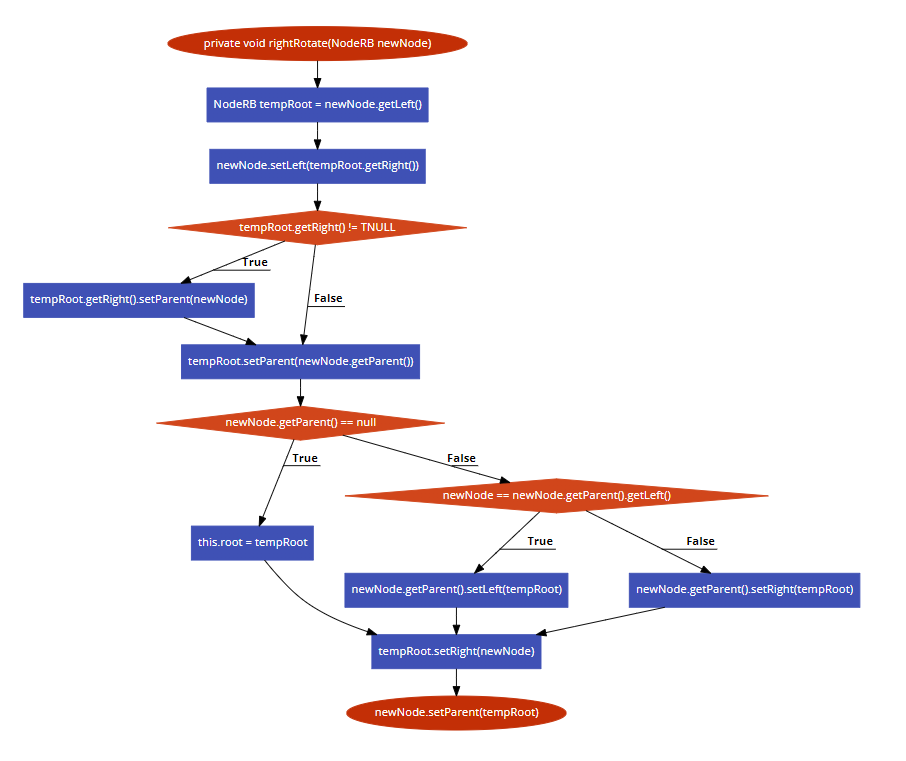
**Блок-схеми для методів для роботи з червоно-чорним деревом:**

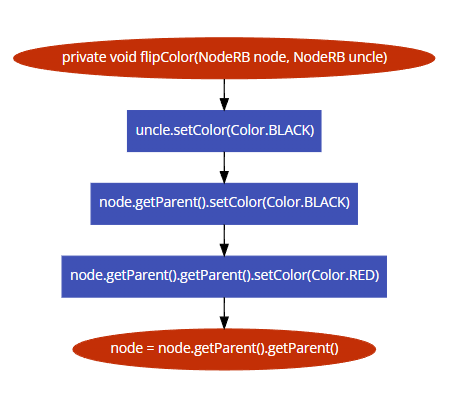
1. Додавання в червоно-чорне дерево.



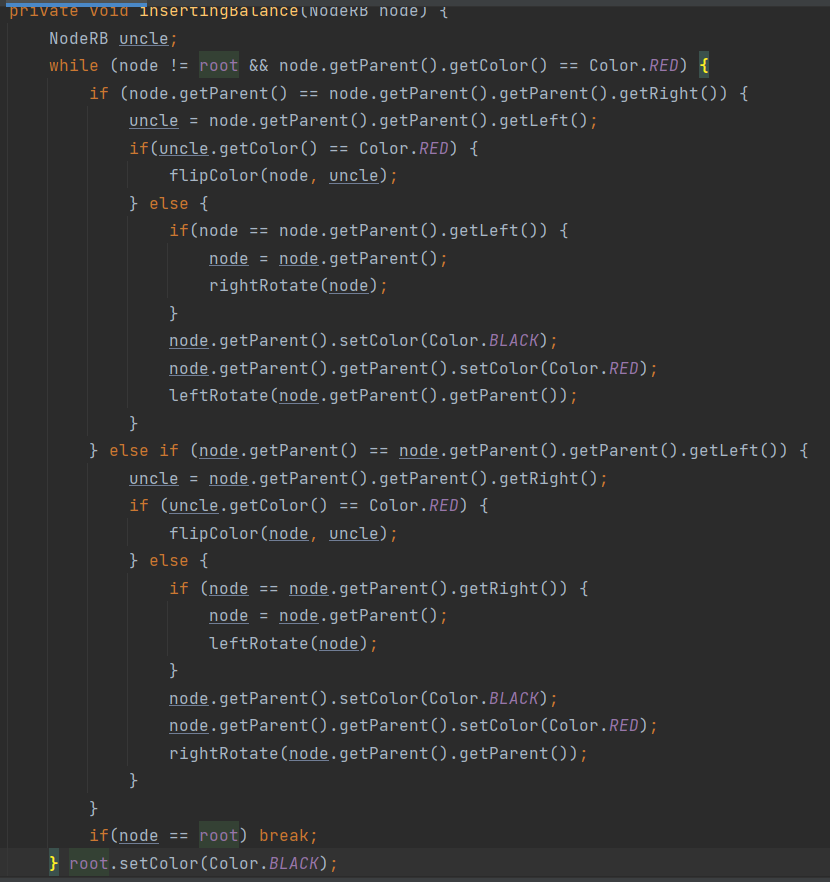


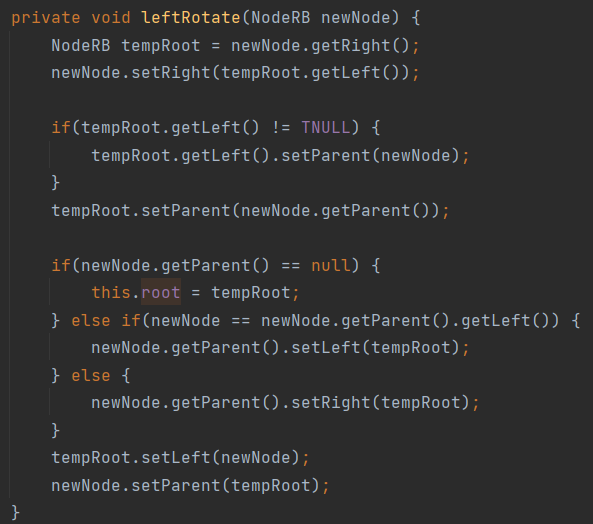




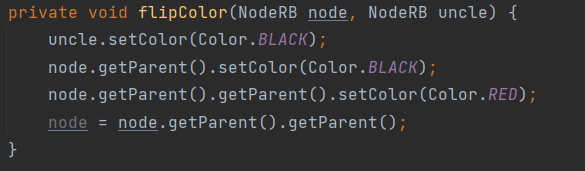


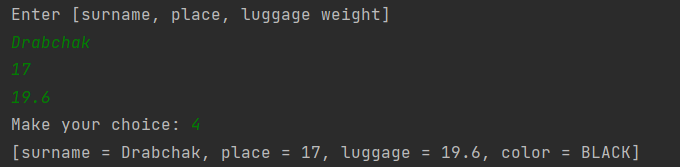




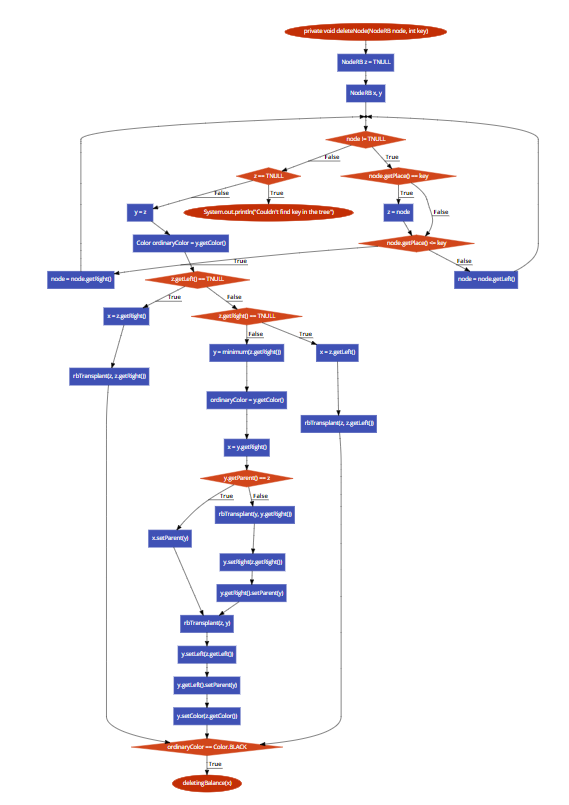


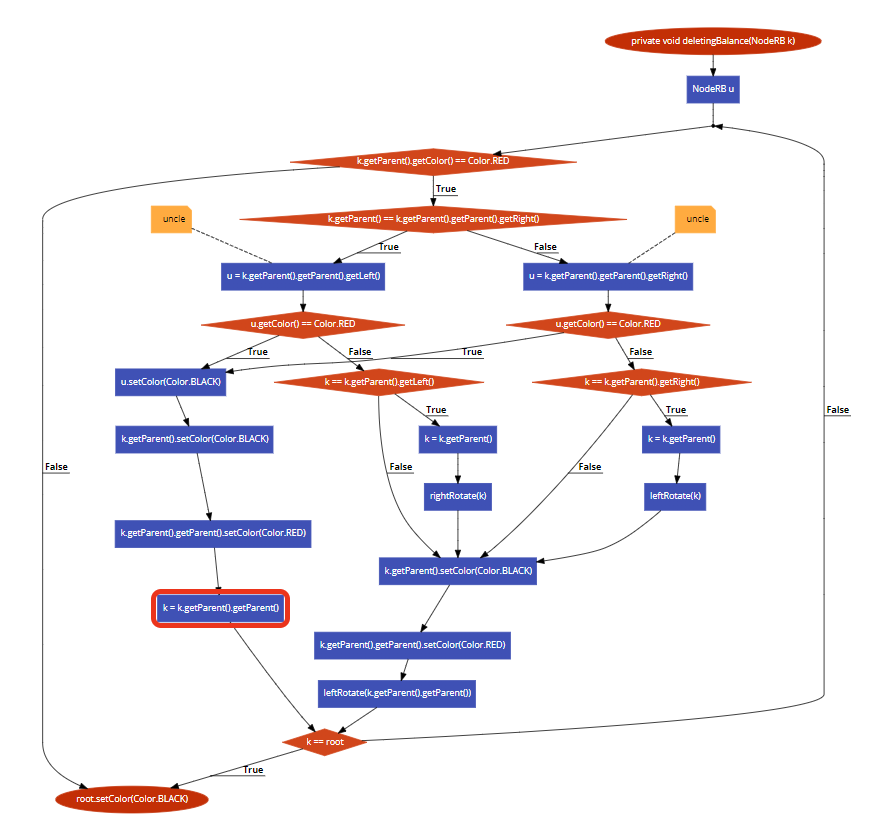


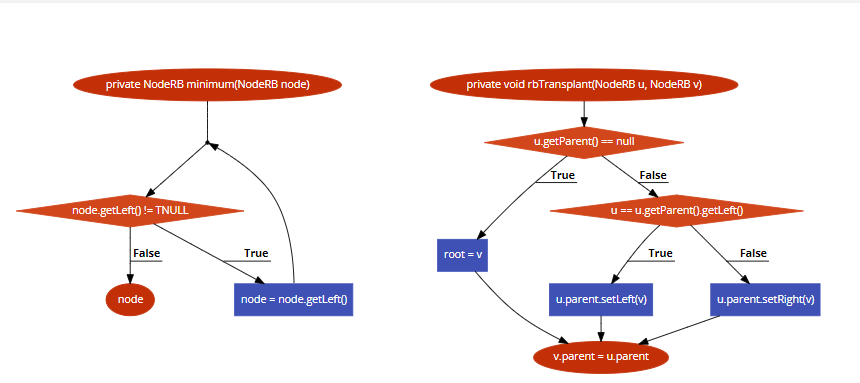




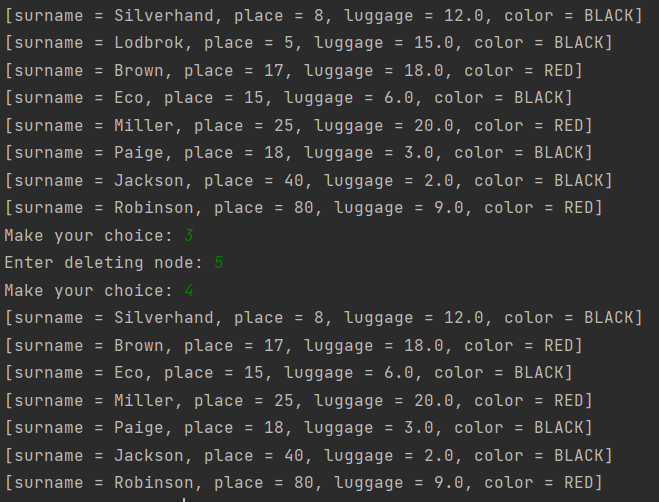
1. Видалення з червоно-чорного дерева.







private void deleteNode(NodeRB node, int key) {  
 NodeRB z = TNULL;  
 NodeRB x, y;  
 while(node != TNULL) {  
 if(node.getPlace() == key) {  
 z = node;  
 }  
 if(node.getPlace() <= key) {  
 node = node.getRight();  
 } else {  
 node = node.getLeft();  
 }  
 }  
 if(z == TNULL) {  
 System.*out*.println("Couldn't find key in the tree");  
 return;  
 }  
 y = z;  
 Color ordinaryColor = y.getColor();  
 if(z.getLeft() == TNULL) {  
 x = z.getRight();  
 rbTransplant(z, z.getRight());  
 } else if(z.getRight() == TNULL) {  
 x = z.getLeft();  
 rbTransplant(z, z.getLeft());  
 } else {  
 y = minimum(z.getRight());  
 ordinaryColor = y.getColor();  
 x = y.getRight();  
 if(y.getParent() == z) {  
 x.setParent(y);  
 } else {  
 rbTransplant(y, y.getRight());  
 y.setRight(z.getRight());  
 y.getRight().setParent(y);  
 }  
 rbTransplant(z, y);  
 y.setLeft(z.getLeft());  
 y.getLeft().setParent(y);  
 y.setColor(z.getColor());  
 }  
 if(ordinaryColor == Color.*BLACK*) {  
 deletingBalance(x);  
 }  
}  
private void deletingBalance(NodeRB k) {  
 NodeRB u;  
 while(k.getParent().getColor() == Color.*RED*) {  
 if(k.getParent() == k.getParent().getParent().getRight()) {  
 u = k.getParent().getParent().getLeft(); //uncle  
 if(u.getColor() == Color.*RED*) {  
 u.setColor(Color.*BLACK*);  
 k.getParent().setColor(Color.*BLACK*);  
 k.getParent().getParent().setColor(Color.*RED*);  
 k = k.getParent().getParent();  
 } else {  
 if(k == k.getParent().getLeft()) {  
 k = k.getParent();  
 rightRotate(k);  
 }  
 k.getParent().setColor(Color.*BLACK*);  
 k.getParent().getParent().setColor(Color.*RED*);  
 leftRotate(k.getParent().getParent());  
 }  
 } else {  
 u = k.getParent().getParent().getRight(); //uncle  
 if(u.getColor() == Color.*RED*) {  
 u.setColor(Color.*BLACK*);  
 k.getParent().setColor(Color.*BLACK*);  
 k.getParent().getParent().setColor(Color.*RED*);  
 k = k.getParent().getParent();  
 } else {  
 if(k == k.getParent().getRight()) {  
 k = k.getParent();  
 leftRotate(k);  
 }  
 k.getParent().setColor(Color.*BLACK*);  
 k.getParent().getParent().setColor(Color.*RED*);  
 leftRotate(k.getParent().getParent());  
 }  
 }  
 if (k == root) {  
 break;  
 }  
 }  
 root.setColor(Color.*BLACK*);  
}  
private NodeRB minimum(NodeRB node) {  
 while (node.getLeft() != TNULL) {  
 node = node.getLeft();  
 }  
 return node;  
}  
private void rbTransplant(NodeRB u, NodeRB v) {  
 if (u.getParent() == null) {  
 root = v;  
 } else if (u == u.getParent().getLeft()) {  
 u.parent.setLeft(v);  
 } else {  
 u.parent.setRight(v);  
 }  
 v.parent = u.parent;  
}



**Висновки**

При виконання лабораторної роботи я розглянув та освоїв такі структури даний як бінарні та червоно-чорні дерева і основні методи роботи з ними.

Двійкове дерево пошуку в інформатиці — двійкове дерево, в якому кожній вершині x зіставлене певне значення . При цьому такі значення повинні задовольняти умові впорядкованості:

* нехай — довільна вершина двійкового дерева пошуку. Якщо вершина y знаходиться в лівому піддереві вершини , то
* Якщо у знаходиться у правому піддереві , то

Таке структурування дозволяє надрукувати усі значення у зростаючому порядку за допомогою простого алгоритму центрованого обходу дерева.

Представляється таке дерево вузлами наступного вигляду:

Доступ до дерева T здійснюється за допомогою посилання .

Бінарні дерева пошуку набагато ефективніші в операціях пошуку, аніж лінійні структури, в яких витрати часу на пошук пропорційні , де — розмір масиву даних, тоді як в повному бінарному дереві цей час пропорційний в середньому або, де — висота дерева (хоча гарантувати, що h не перевищує можна лише для збалансованих дерев, які є ефективнішими в алгоритмах пошуку, аніж прості бінарні дерева пошуку).

Червоно-чорнедерево - в інформатиці - різновид самозбалансованого бінарного дерева пошуку, вершини якого мають додаткові властивості (RB-властивості), зокрема «колір» (червоний або чорний). Ці біти кольору використовуються для забезпечення того, щоб дерево залишалося приблизно збалансованим при виконанні операцій вставки та видалення.

Червоно-чорні дерева — різновид збалансованих дерев, в яких за допомогою спеціальних трансформацій гарантується, що висота дерева не буде перевищувати Зважаючи на те, що час виконання основних операцій на бінарних деревах (пошук, видалення, додавання елементу) є , ці структури даних на практиці є набагато ефективнішими, аніж звичайні бінарні дерева пошуку.

Бінарне дерево називається червоно-чорним, якщо воно має такі властивості:

* кожна вершина або червона, або чорна
* корінь дерева — чорний
* кожний лист (NIL) — чорний
* якщо вершина червона, обидві її дочірні вершини чорні (інакше, батько червоної вершини — чорний)
* усі прості шляхи від будь-якої вершини до листів мають однакову кількість чорних вершин