Лабораторна робота №7

Робота з бінарним та червоно-чорним деревами пошуку

Метою виконання лабораторної роботи є набуття практичних навичок із проектування, реалізації, тестування та аналізу алгоритмів пошуку, вставки та видалення елементів із деревоподібних структур даних. В якості структур даних для задачі пошуку розглядаються бінарне дерево пошуку та червоно-чорне дерево

Зміст:

1. Основні відомості про бінарні та червоно-чорні дерева пошуку.
(Див. у презентації лекції)

2. Вставка вузла в бінарне дерево пошуку

Операція вставки вузла в бінарне дерево пошуку працює *аналогічно* пошуку вузла, відмінністю є те, що якщо виявлено у вузла відсутність дочірнього елемента необхідно «підвісити» на нього елемент, що вставляється.
Розглянемо функцію побудови бінарного дерева пошуку (binary search tree, BST)

```
Func buildBST (a : int[n]):
for i = 1 to n
Node insert(x, a[i])

Node insert(x : Node, z : T):   // x — корінь піддерева, z — ключ, що вставляється
        if x == null
          return Node(z)                    // вставляємо Node з key = z
        else if z < x.key
          x.left = insert(x.left, z)
        else if z > x.key
          x.right = insert(x.right, z)
        return x
```

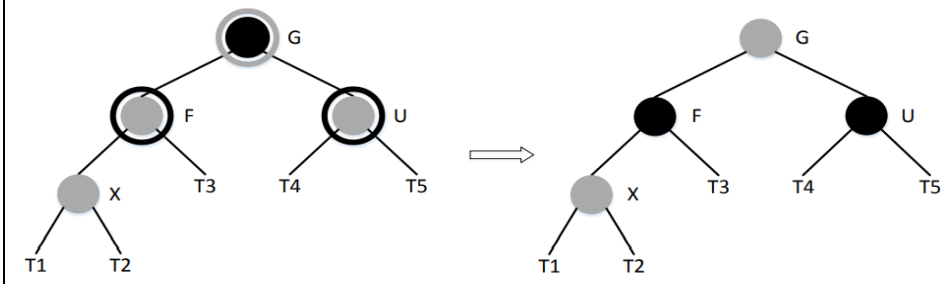
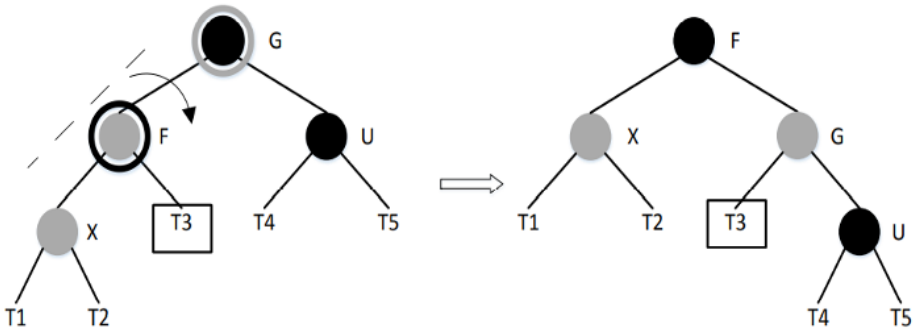
У циклі по елементах масиву запускається функція **Node** insert(x, a[i]). Бінарне дерево пошуку будується так: перший елемент стає кореневим елементом дерева, після чого наступні елементи порівнюються з кореневим. Якщо елемент менше кореневого – він йде в ліве піддерево і порівнюється з лівим нащадком кореневого елемента. Якщо елемент більше кореневого – він іде у праве піддерево і порівнюється з правим нащадком кореневого елемента. Це триває доки для елемента не знайдеться вільне місце.

3. Вставка елементів у червоно-чорне дерево (red-black tree RB tree) (3 випадки)

Виконується за наступним алгоритмом:
Знаходимо лист для вставки нового елемента
Створюємо елемент і фарбуємо його у червоний колір
Перефарбовуючи вузли та виконуючи повороти, відновлюємо структуру червоно-чорного дерева

При вставці вузла *X* **(червоного)** у ЧЧД можливі 6 випадків, що порушують властивості ЧЧД, при цьому 3 випадки **симетричні** іншим трьом (Таблиця 7.1).

Таблиця 7.1 Правила балансування при вставці вузла *X* **(червоного)** у ЧЧД

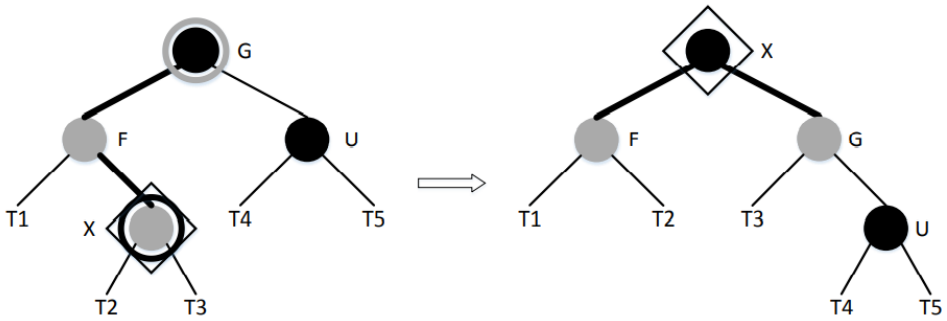
<table>
<tr><td>Випадок 1
Дядько U вузла X, що додається, червоний
Розв'язання:
перефарбування F та U у чорний колір, а G у червоний</td><td> 
</td></tr>
<tr><td colspan="2">Далі балансування виконується відносно вузла G</td></tr>
<tr><td>Випадок 2
Дядько U вузла X, що додається, чорний і при цьому ланцюжок вузлів X-F-G утворює пряму лінію
Розв'язання:
перефарбування F та G й одинарний правий поворот</td><td> 
</td></tr>
</table>

Правий поворот виконується «навколо» зв'язку між *G* і *F*, роблячи *F* новим коренем піддерева, правим дочірнім вузлом якого стає *G* , а колишній правий нащадок вузла *F (T3)* – лівим нащадком *G*.

Випадок 3

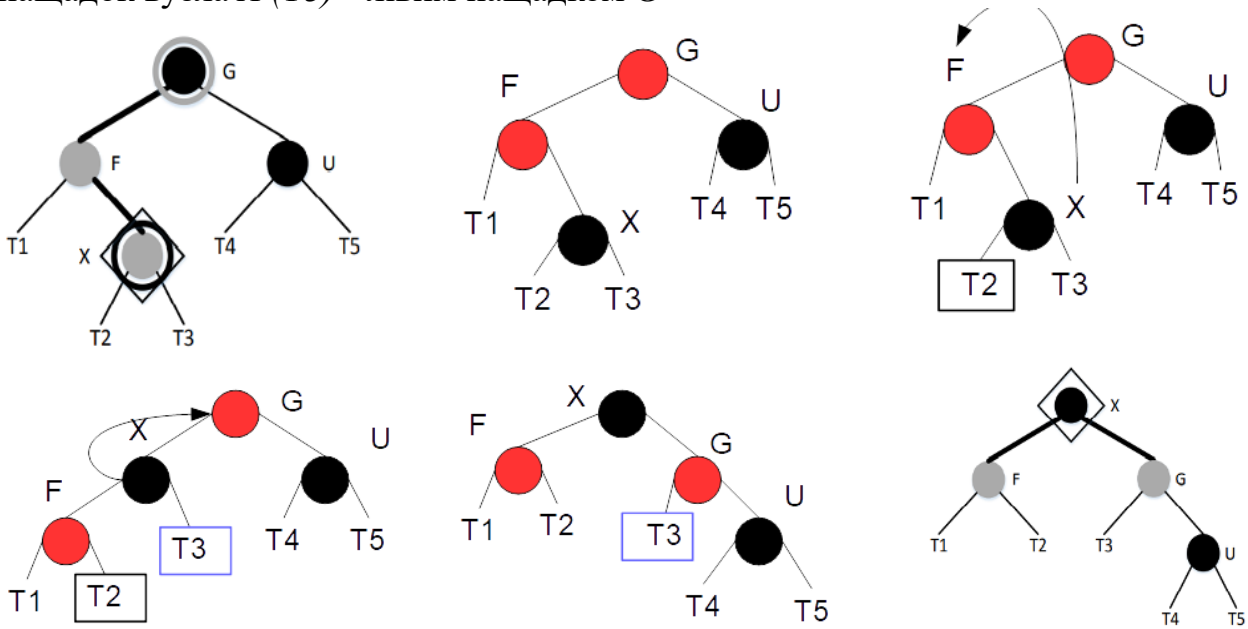
Дядько *U* вузла *X*, що додається, чорний і при цьому ланцюжок вузлів *X-F-G* утворює кут

Розв'язання: перефарбування *X* та *G*, і подвійний (лівий та правий) поворот комбінації *X-F-G*, коли нижній вузол *(X)* опиняється нагорі комбінації.



Лівий поворот виконується «навколо» зв'язку між *F* і *X*, роблячи *X* новим коренем піддерева, лівим дочірнім вузлом якого стає *F*, а колишній лівий нащадок вузла *X (T2)* – правим нащадком *F*

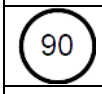
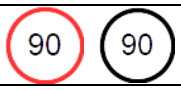
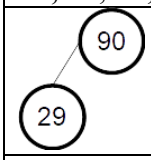
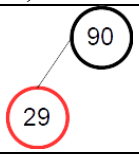
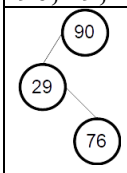
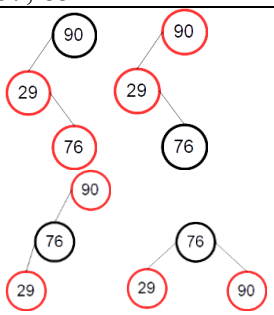
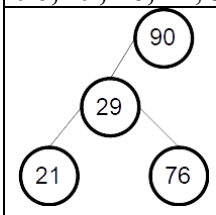
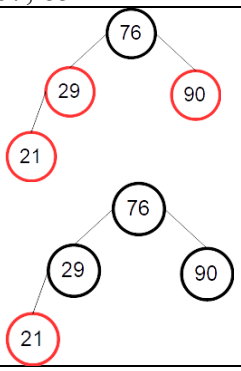
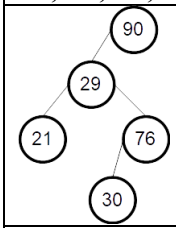
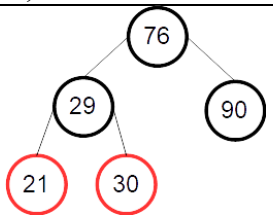
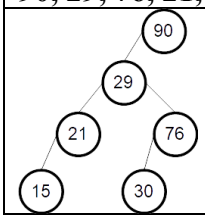
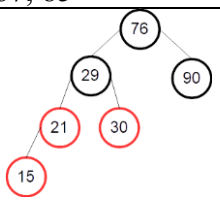
Правий поворот виконується «навколо» зв'язку між *G* і *X*, роблячи *X* новим коренем піддерева, правим дочірнім вузлом якого стає *G*, а колишній правий нащадок вузла *X (T3)* - лівим нащадком *G*

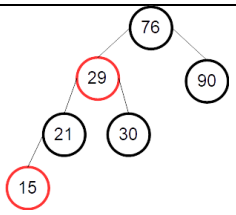
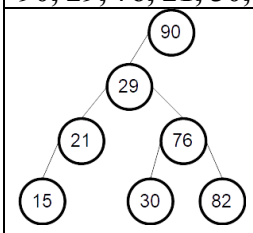
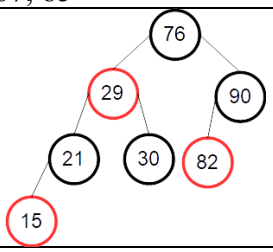
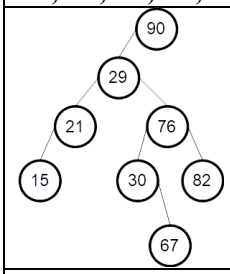
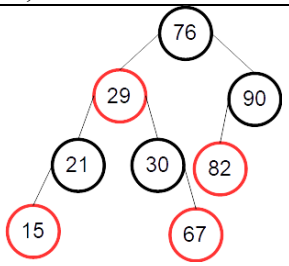
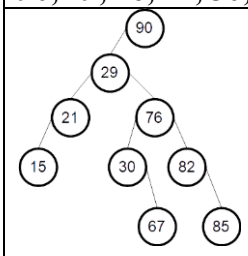
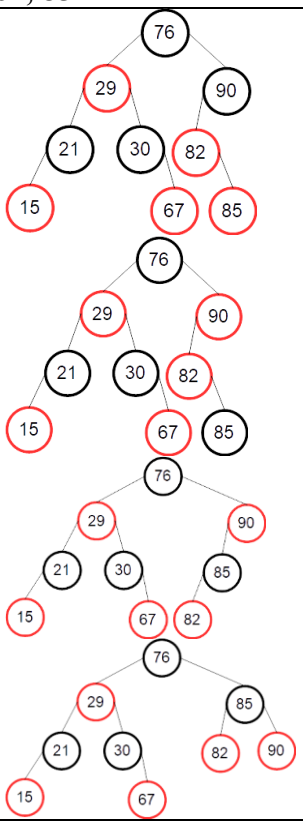


4. Приклади додавання елементів

У таблиці 7. 2 показаний процес побудови бінарного дерева пошуку та червоно-чорного дерева за допомогою вставки вузла для наступної множини елементів: 90, 29, 76, 21, 30, 15, 82, 67, 85. Вигляд отриманих дерев показаний на рисунку 7. 1.

Таблиця 7. 2 Побудова бінарного та червоно-чорного дерева пошуку

90, 29, 76, 21, 30, 15, 82, 67, 85		
Побудова BST Додаткових пояснень не потребує	Побудова RB tree. Пояснення дій за алгоритмом вставки обов'язкові	
		Вставляємо елемент 90. Випадок 1 – перефарбування в чорний колір
90, 29, 76, 21, 30, 15, 82, 67, 85		
		Вставляємо елемент 29. Він стає лівим сином елемента 90. Оскільки батько чорний, немає порушень, нічого не міняємо.
90, 29, 76, 21, 30, 15, 82, 67, 85		
		Додавання елемента 76. Він стає правим сином елемента 29. Червоне-червоне порушення. **Випадок 3** - утворився кут. Перефарбовуємо 76 чорним кольором, а 90 червоним кольором. Робимо лівий поворот навколо вузла 76, а потім правий поворот навколо вузла 76, вузол 76 став кореневим.
90, 29, 76, 21, 30, 15, 82, 67, 85		
		Додавання елемента 21. Він стає червоним лівим сином вузла 29. Червоне-червоне порушення. **Випадок 1 -** утворення прямої лінії, дядько доданого вузла червоний. Перефарбовуємо батька та дядька вузла 21 у чорний колір.
90, 29, 76, 21, 30, 15, 82, 67, 85		
		Додавання елемента 30. Він стає червоним правим сином вузла 29. Порушень немає
90, 29, 76, 21, 30, 15, 82, 67, 85		
		Додавання елемента 15. Він стає червоним лівим сином вузла 21. Червоне-червоне порушення. **Випадок 1** - утворюється прямої лінії, дядько доданого вузла червоний.

		Перефарбовуємо батька (21) та дядька (30) вузла 15 чорним кольором, а дідуся вузла 15 вузел (29) червоним кольором. Порушень відносно вузла 29 немає. Вимоги стосовно чорної висоти виконуються
90, 29, 76, 21, 30, 15, 82, 67, 85		
		Додавання елемента 82. Він стає червоним лівим сином вузла 90. Порушень немає
90, 29, 76, 21, 30, 15, 82, 67, 85		
		Додавання елемента 67. Він стає червоним правим сином вузла 30. Порушень немає
90, 29, 76, 21, 30, 15, 82, 67, 85		
		Додавання елемента 85. Він стає правим сином елемента 82. Червоне-червоне порушення. **Випадок 3** утворення кута. Перефарбовуємо 85 чорним кольором, а 90 червоним кольором. Робимо лівий поворот навколо вузла 82, а потім правий поворот навколо вузла 90

Дерево побудовано. Висота бінарного дерева пошуку дорівнює 5. Чорна висота червоно-чорного дерева дорівнює 2, а висота 4

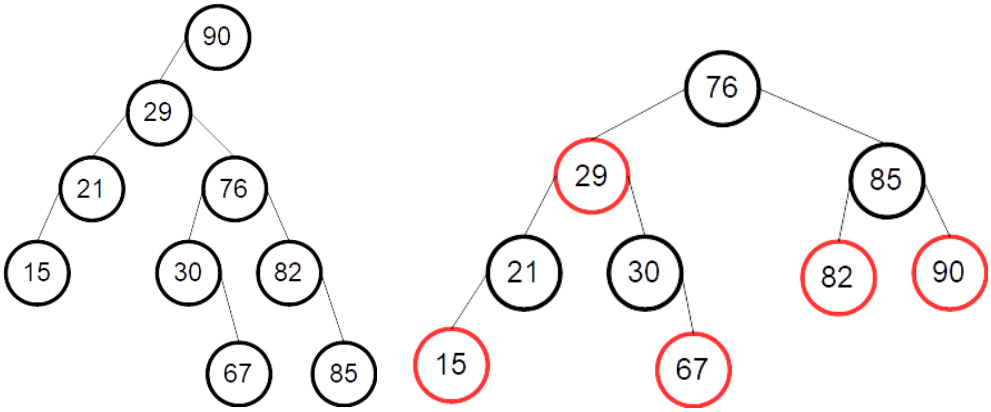


Рисунок 7.1 Вигляд BST(а) та RB tree (б) побудованих із {90, 29, 76, 21, 30, 15, 82, 67, 85}

5. Обходи бінарного та червоно-чорного дерев пошуку

Є три операції обходу вузлів дерева, що відрізняються порядком обходу вузлів:

inorderTraversal — обхід вузлів у симетричному (відсортованому) порядку,

preorderTraversal — обхід вузлів у прямому порядку: вершина, ліве піддерево, праве піддерево,

postorderTraversal — обхід вузлів у зворотному порядку: ліве піддерево, праве піддерево, вершина.

Обхід вузлів у симетричному (відсортованому) порядку

```
func inorderTraversal(x : Node):
    if x != null
      inorderTraversal(x.left)
      print x.key
      inorderTraversal(x.right)
```

Відповідно до inorderTraversal алгоритму порядок обходу елементів побудованого бінарного дерева пошуку у симетричному порядку наступний: 15, 21, 29, 30, 67, 76, 82, 85, 90.

При цьому стек викликів функції inorderTraversal на кожній ітерації буде виглядати наступним чином:

Стек виклику симетричного обходу

1	90				
2	90	29			
3	90	29	21		
4	90	29	21	**15**	
5	90	29	**21**		
6	90	**29**			
7	90	29	76		
8	90	29	76	**30**	
9	90	29	76	30	**67**
10	90	29	76	30	
11	90	29	**76**		

12	90	29	76	**82**	
13	90	29	76	82	**85**
14	90	29	76	82	
15	90	29	76		
16	90	29			
17	**90**				

Відповідно до inorderTraversal алгоритму порядок обходу елементів червоно-чорного дерева у симетричному порядку наступний: 15, 21, 29, 30, 67, 76, 82, 85, 90.

При цьому стек викликів функції inorderTraversal на кожній ітерації буде виглядати наступним чином:

1	76			
2	76	29		
3	76	29	21	
4	76	29	21	**15**
5	76	29	**21**	
6	76	**29**		
7	76	29	**30**	
8	76	29	30	**67**
9	76	29	30	
10	76	29		
11	**76**			
12	76	85		
13	76	85	**82**	
14	76	**85**		
15	76	85	**90**	
16	76	85		
17	76			

Як бачимо в першому и другому випадках кількість ітерацій при симетричному обході однакова. Але в першому випадку в стеку викликів на 9 та 13 ітераціях міститься по 5 викликів відповідно на відміну від другого випадку, де 4 виклики - максимальна кількість викликів. Тому для реалізації inorderTraversal для BST знадобилося більше витрат пам'яті ніж для RB tree.

Обхід вузлів у прямому порядку

```
func preorderTraversal(x : Node)
  if x != null
    print x.key
    preorderTraversal(x.left)
    preorderTraversal(x.right)
```

Відповідно до preorderTraversal алгоритму порядок обходу елементів бінарного дерева у прямому порядку наступний: 90,29,21,15,76,30,67,82,85.

Відповідно до preorderTraversal алгоритму порядок обходу елементів червоно-чорного дерева у прямому порядку наступний:76,29,21,15,30,67,85,82,90.

Обхід вузлів у зворотному порядку
func postorderTraversal(x : **Node**)
  if x != *null*
    postorderTraversal(x.left)
    postorderTraversal(x.right)
    print x.key

Відповідно до postorderTraversal алгоритму порядок обходу елементів бінарного дерева у зворотному порядку наступний:15,21,67,30,85,82,76,29,90.
Відповідно до postorderTraversal алгоритму порядок обходу елементів червоно-чорного дерева у зворотному порядку наступний:15,21,67,30,29,82,90,85,76.

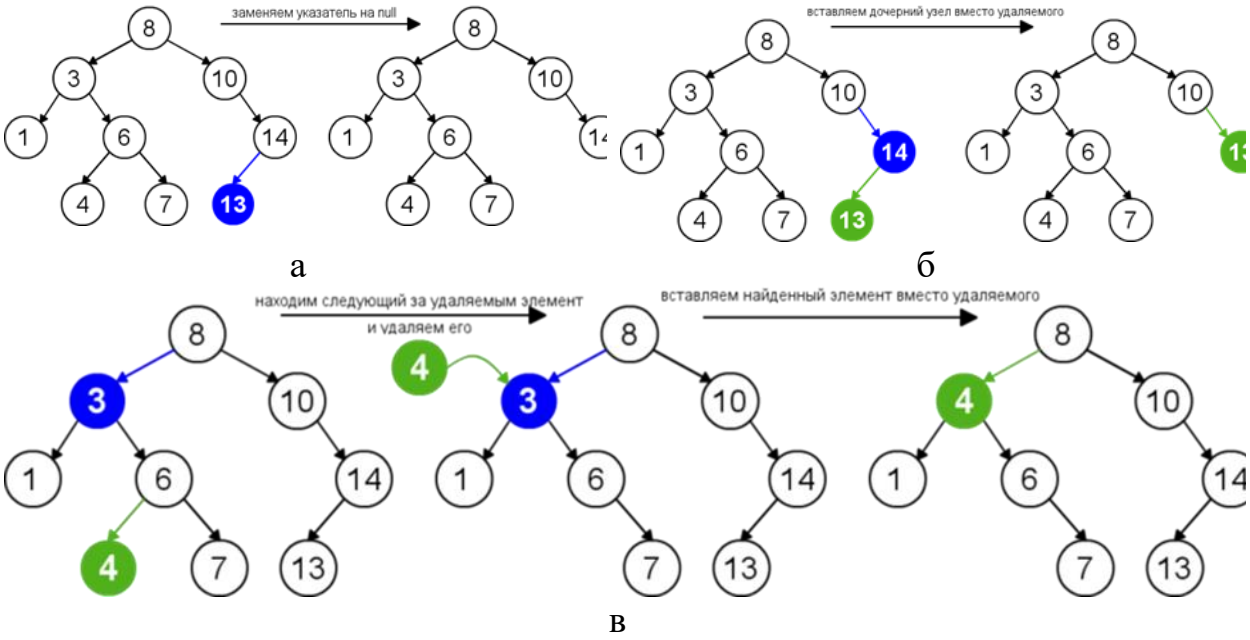
6. Видалення елементів з бінарного дерева пошуку

Рекурсивне видалення вузла із BST
Node delete(root : **Node**, z : **T**): // корінь піддерева, ключ, що видаляється
 if root == *null*
  return root
 if z < root.key //елемент, що видаляється, знаходиться в лівому піддереві
  root.left = delete(root.left, z)
 else if z > root.key //елемент, що видаляється, знаходиться в правому піддереві
  root.right = delete(root.right, z)
 else if root.left != *null* **and** root.right != *null* //елемент, що видаляється, знаходиться в корені і має два дочірні вузли
  root.key = minimum(root.right).key
  root.right = delete(root.right, root.key)
 else //елемент, що видаляється, має один дочірній вузол, потрібно замінити його нащадком
  if root.left != *null*
   root = root.left
  else if root.right != *null*
   root = root.right
  else
   root = *null*
 return root

При рекурсивному видаленні вузла з бінарного дерева слід розглянути три випадки (Рис.7.2):

1. елемент, що видаляється, знаходиться в лівому піддереві поточного піддерева,
2. елемент, що видаляється, знаходиться в правому піддереві
3. елемент, що видаляється, знаходиться в корені.

У перших двох випадках потрібно рекурсивно видалити елемент з потрібного піддерева.
Якщо елемент, що видаляється, знаходиться в корені поточного піддерева і має два дочірні вузли, то потрібно замінити його мінімальним (крайнім лівим) елементом з правого піддерева і рекурсивно видалити **цей** мінімальний елемент з правого піддерева. Інакше, якщо елемент, що видаляється, має один дочірній вузол, потрібно замінити його нащадком. Час роботи алгоритму — O(h). Рекурсивна функція, що повертає дерево з видаленим елементом z
Наприклад



а б в

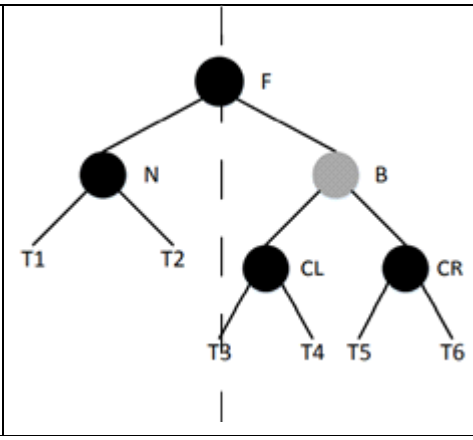
Рисунок – 7.2 Видалення вузла із бінарного дерева пошуку

7. Видалення елементів із червоно-чорного дерева

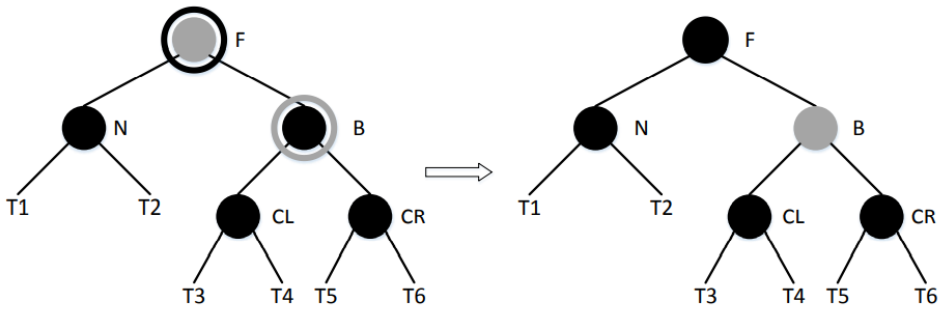
Видалення вузла з ЧЧД, так само, як і вставка вузла, відбувається у два етапи: власне видалення за допомогою алгоритму видалення з BST, та відновлення властивостей ЧЧД, якщо вони були порушені.
Якщо вузол, що видаляється з дерева, червоний, червоно-чорні властивості дерева зберігаються.
Таким чином, відновлення властивостей дерева може знадобитися тільки в тому випадку, якщо вузол, що видаляється, – *чорний*. Якщо при цьому син вузла, що видаляється, червоний, то після видалення достатньо буде перефарбувати сина видаленого вузла в чорний колір, щоб відновити кількість чорних вузлів на цьому шляху.
Розглянемо 5 випадків конфігурації дерева після видалення *чорного* вузла, у якого син – *чорний* (Таблиця 3).

Таблиця 7. 3 Правила балансування ЧЧД після видалення *чорного* вузла, у якого син – *чорний*

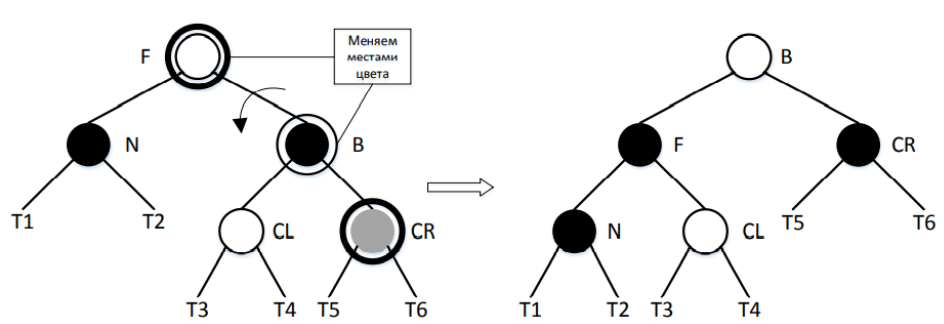
Позначимо сина видаленого вузла за *N*, а батька видаленого вузла за *F*. Після видалення *F* став новим батьком *N*. Позначимо за *B* нового брата *N*, а за *CL* та *CR* – лівого та правого синів *B*, відповідно. На те, якою буде процедура відновлення, впливає тільки комбінація з цих 5 вузлів. Будемо вважати, що *N* – лівий син *F*. Інакше, усі зображення будуть симетричними щодо вертикальної осі, яка проходить через *F*.



Випадок 1. Батько *F* вузла *N* червоний, інші вузли чорні. (*Батько F вузла N червоний: перефарбування B та F*)

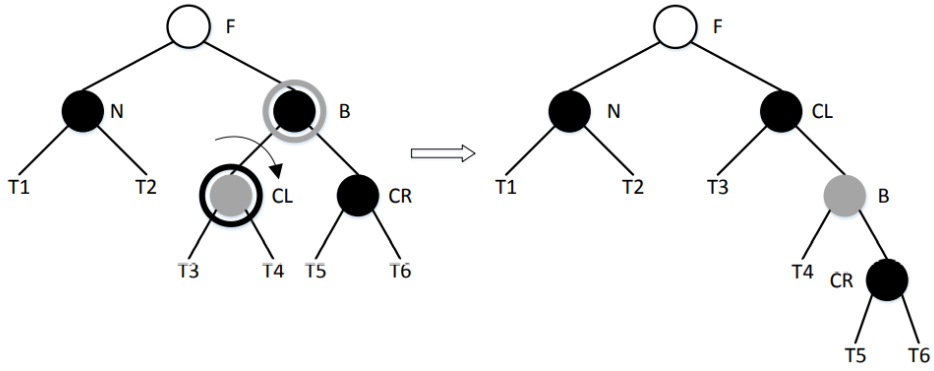
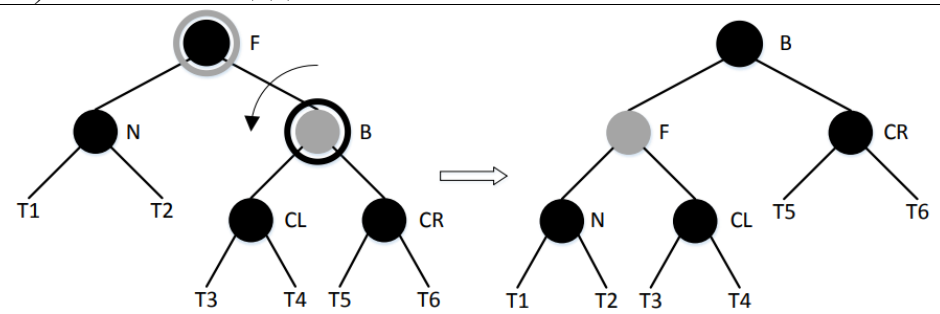
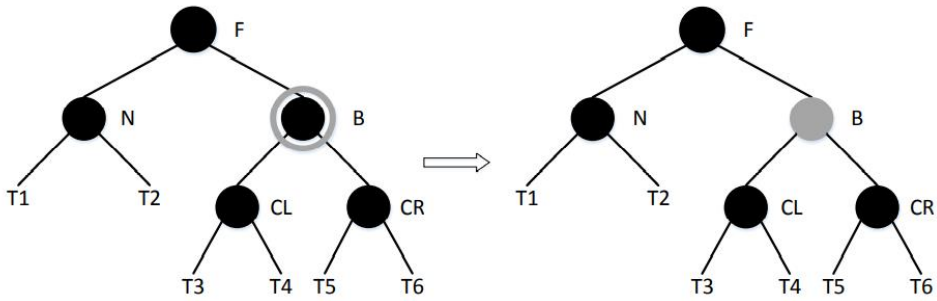


Випадок 2. Брат *B* вузла *N* чорний, а його правий син *CR* червоний. Незалежно від того, чи *F* є чорним або червоним, властивості ЧЧД відновлюються після лівого повороту *F* навколо *B*, перефарбування *CR* в чорний колір і обміну кольорами *F* і *B*.



Лівий поворот навколо F та B, перефарбування CR в чорний та обмін кольорами F та B

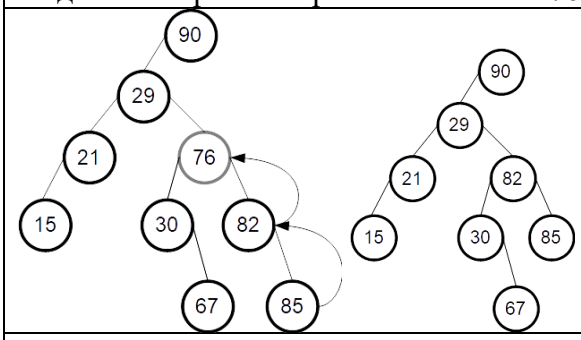
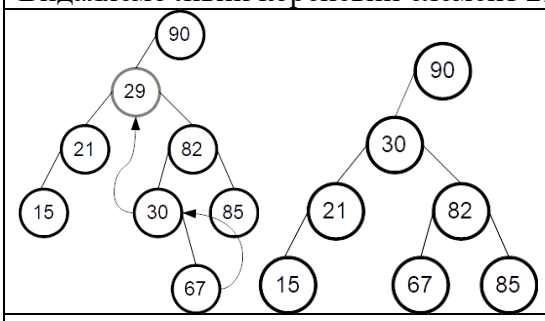
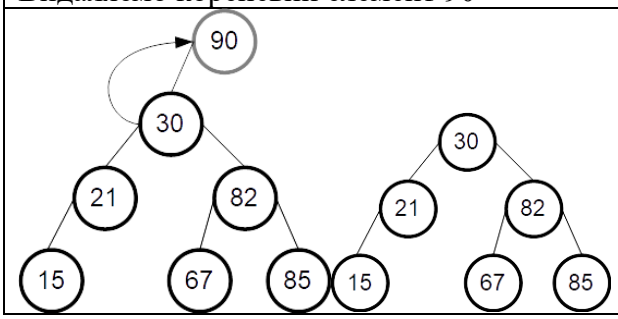
Лівий поворот виконується «навколо» зв'язку між *F* і *B*, роблячи *B* новим коренем піддерева, лівим дочірнім вузлом якого стає *F*, а колишній лівий нащадок вузла *B (CL)* - правим нащадком *F*.

Випадок 3. Брат *B* вузла *N* чорний, його правий син *CR* чорний, а лівий син CL червоний.	 *Правий поворот CL навколо B, перефарбування CL та B*
Правий поворот виконується «навколо» зв'язку між *B* і *CL*, роблячи *CL* новим коренем піддерева, правим дочірнім вузлом якого стає *B*, а колишній правий нащадок вузла *CL (T4)* – лівим нащадком *B*	
Випадок 4. Брат *B* вузла *N* червоний. У цьому випадку *F*, *CL* та *CR* можуть бути лише чорними	 *Лівий поворот F навколо B та перефарбування F та B*
Лівий поворот виконується «навколо» зв'язку між *F* і *B*, роблячи *B* новим коренем піддерева, лівим дочірнім вузлом якого стає *F*, а колишній лівий нащадок вузла *B (CL)* - правим нащадком *F*	
Випадок 5. Усі вузли комбінації (брат *B*, його діти *CL* та *CR*, а також батько *F* вузла *N*) чорні.	 *Перефарбування вузла B у червоний колір та продовження процедури нагору*

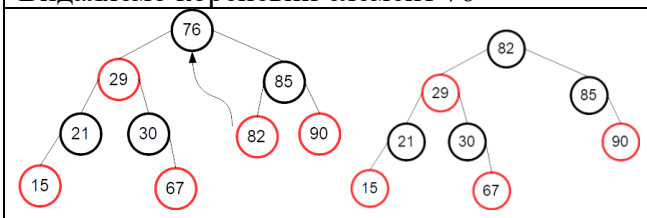
8. Приклади видалення елементів

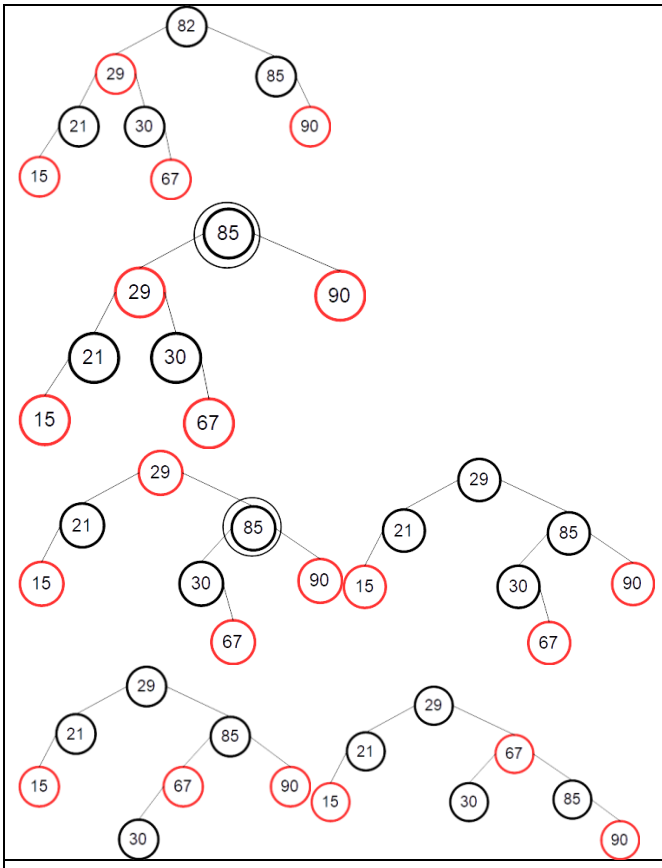
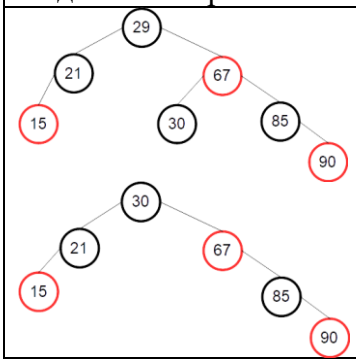
У таблиці 7. 4 показаний процес почергово видалення кореневих елементів *правого* та *лівого* піддерев, а також основного кореня із побудованого бінарного дерева пошуку (див. рисунок 7.1) У таблиці 7. 5 показаний процес почергово видалення із з побудованого червоно-чорного дерева (див. рисунок 7.1) трьох чорних елементів, які мають переважно чорних синів

Таблиця 7. 4 Видалення вузлів із бінарного дерева пошуку

Видаляємо правий кореневий елемент 76	
	Кореневий елемент правого піддерева 76, що видаляється, має два дочірніх вузла (елементи 30 та 82). Шукаємо крайній лівий елемент правого піддерева – це елемент 82 з правим нащадком – елементом 85. Тому елемент 82 стає на місце елемента 76, а елемент 85 на місце елемента 82
Видаляємо лівий кореневий елемент 29	
	Кореневий елемент лівого піддерева 29, що видаляється, має два дочірніх вузла (елементи 21 та 82). Шукаємо крайній лівий елемент правого піддерева – це елемент 30 з правим нащадком – елементом 67. Тому елемент 30 стає на місце елемента 29, а елемент 67 на місце елемента 30
Видаляємо кореневий елемент 90	
	Кореневий елемент дерева 90, що видаляється, має тільки один лівий дочірній вузол (елемент 30). Тому він займає місце кореневого елемента 90.

Таблиця 7. 5 Видалення вузлів із червоно-чорного дерева

Видаляємо кореневий елемент 76	
	Кореневий елемент піддерева 76, що видаляється, має два дочірніх вузла (червоний 29 та чорний 82). Шукаємо крайній лівий елемент правого піддерева – це червоний елемент 82 – ставимо його на місце 76. Він кореневий тому перефарбовуємо його в чорний колір. Чорну висоту (вл. 4) не порушено
Видаляємо кореневий елемент 82	

	Кореневий елемент дерева 82, що видаляється, має два дочірніх вузла (червоний 29 та чорний 85). Шукаємо крайній лівий елемент правого піддерева – це чорний елемент 85 – ставимо його на місце 82. Маємо порушення чорної висоти. Крім того елемент 85 стає «двічі» чорнім. (Випадок 3) Робимо правий поворот навколо зв'язку вузлів 29 та 85, 29 стає новим коренем, правим дочірнім вузлом стає 85, колишні правий нащадок вузла 29 вузел 30 стає лівим нащадком вузла 85. Перефарбування 29 та 85 (віддає чорний колір) Порушення чорної висоти. Лівий поворот навколо 67 та 30 та правий поворот навколо 67 та 85
Видаляємо кореневий елемент 29	
	Кореневий елемент дерева 29, що видаляється, має два дочірніх вузла (чорний 21 та червоний 67). Шукаємо крайній лівий елемент правого піддерева – це чорний елемент 30 – ставимо його на місце 29. Порушень немає

9. Завдання:

1. Побудувати бінарне дерево пошуку та червоно-чорне дерево за варіантами завдань наводячи покроковий опис (зразок таблиця 7.2)
2. Навести результати симетричного, прямого та зворотного обходів побудованого бінарного дерева пошуку та червоно-чорне дерева. Для симетричного обходу навести стан стека викликів рекурсивної функції inorderTraversal. Порівняйте результати, зробить висновки
3. Видаліть по черзі та опишіть процедуру видалення із побудованого бінарного дерева пошуку трьох кореневих елементів в такому порядку: праве, ліве піддерева, основний кореневий елемент (зразок таблиця 7.4)
4. Видаліть по черзі та опишіть процедуру видалення із побудованого червоно-чорного дерева трьох чорних елементів, які мають переважно чорних синів (зразок таблиця 7.5)

Варіанти завдань:

Варіант	Послідовність
1	53, 12, 68, 63, 3, 47, 50, 61, 41
2	98, 40, 42, 88, 61, 87, 79, 97, 82
3	70, 80, 61, 92, 12, 23, 67, 65, 50
4	29, 1, 24, 69, 52, 97, 27, 10, 88
5	95, 43, 98, 41, 68, 67, 10, 7, 69
6	61, 32, 27, 45, 75, 58, 5, 50, 99
7	78, 77, 4, 62, 8, 69, 46, 11, 49
8	28, 76, 27, 10, 5, 35, 95, 16, 33
9	31, 40, 22, 50, 53, 68, 97, 12, 15
10	18, 20, 2, 88, 61, 17, 79, 97, 82
11	41, 52, 47, 65, 95, 38, 15, 50, 99
12	11, 42, 67, 55, 65, 78, 25, 50, 69
13	53, 5, 44, 47, 35, 83, 82, 85, 28
14	77, 89, 74, 68, 70, 49, 5, 62, 51
15	19, 75, 43, 31, 5, 66, 62, 34, 76
16	50, 57, 78, 34, 41, 68, 47, 61, 38
17	86, 36, 14, 50, 64, 21, 2, 83, 82
18	80, 27, 37, 36, 91, 53, 86, 66, 98
19	21, 44, 22, 50, 63, 68, 97, 12, 15
20	7, 89, 4, 68, 70, 49, 10, 62, 51
21	54, 65, 7, 33, 86, 29, 11, 91, 12
22	50, 80, 19, 86, 35, 7, 60, 48, 51
23	10, 90, 95, 30, 45, 60, 57, 28, 5
24	90, 10, 15, 80, 100, 6, 57, 5, 29
25	53, 100, 44, 74, 53, 38, 82, 65, 28
26	47, 50, 61, 41,53, 12, 68, 63, 3
27	87, 79, 97, 82,98, 40, 42, 88, 61
28	12, 23, 67, 65, 50, 70, 80, 61, 92
29	69, 52, 97, 27, 10, 88,29, 1, 24
30	41, 68, 67, 10, 7, 69, 95, 43, 98
31	58, 5, 50, 99, 61, 32, 27, 45, 75
32	46, 11, 49, 78, 77, 4, 62, 8, 69
33	35, 95, 16, 33, 28, 76, 27, 10, 5
34	68, 97, 12, 15, 31, 40, 22, 50, 53
35	79, 97, 82, 18, 20, 2, 88, 61, 17
36	38, 15, 50, 99, 41, 52, 47, 65, 95

10.Оформлення звіту.

Звіт повинен містити:

– хід побудови бінарного дерева пошуку та червоно-чорного дерева за варіантами завдань (зразок таблиця 7.2)

– результати прямого та зворотного обходів побудованого бінарного дерева пошуку та червоно-чорного дерева; для симетричного обходу стан стека викликів рекурсивної функції inorderTraversal та результати обходу.

– результати видалення із бінарного дерева пошуку трьох кореневих елементів в такому порядку: праве, ліве піддерева, основний кореневий елемент (зразок таблиця 7. 4)

– результати видалення із червоно-чорного дерева трьох чорних елементів, які мають переважно чорних синів (зразок таблиця 7. 5)