МНІСТЕРСТВО НАУКИ ТА ОСВІТИ УКРАЇНИ
Національний Авіаційний Університет
Факультет комп'ютерних наук та технологій
Кафедра прикладної математики

"Алгоритми та структури даних"

Лабораторна робота №2.1

9 листопада 2023 р.

Виконав:
студент №14 групи ПМ 2516 НАУ
Владислав Реган Володимирович

Прийняв:
доцент, к.т.н.
Чолишкіна Ольга Генадіївна

Київ 2023

Зміст

1. Тема і мета роботи

Тема: Типи даних ієрархічної структури. Двійкові дерева. Піраміди.
Мета: Ознайомитися з основними алгоритмами обробки типів даних ієрархічної структури.

2. Постановка задачі

1. Вивчити основні принципи побудови дерев довічного пошуку та алгоритми їх обробки.
2. Вивчити тип даних піраміда і основні алгоритми обробки.
3. Реалізувати програмно ієрархічну структуру даних і відповідно до варіанту. Організувати обробку заданої структури відповідно до запиту користувача.
4. Оцінити обчислювальну складність алгоритмів обробки.

Варіант 4 (варіант $= N \mod 5$ де $N = 14$)

Для класу двійкового дерева пошуку (з цілими ключами і даними) реалізувати програмно наступні алгоритми:

- Вставка елемента (пари - ключ / значення).
- Пошук максимального елемента.
- Пошук елемента по заданому ключу.
- Видалення елемента із заданим ключем.

3. Теоретичні відомості

Вміст цієї секції взято з [1].

3.1. Дерева

3.1.1. Що таке дерево?

Дерево - це структура даних, подібна до зв'язаного списку, але замість того, щоб кожна вершина вказувала просто на наступну вершину в лінійному порядку, кожна вершина вказує на декілька вершин. Дерево є прикладом нелінійної структури даних. Деревоподібна структура - це спосіб представлення ієрархічної природи структури в графічній формі.

У деревах ADT (Abstract Data Type - абстрактний тип даних) порядок елементів не є важливим. Якщо нам потрібна впорядкована інформація, можна використовувати лінійні структури даних, такі як зв'язані списки, стеки, черги тощо.

3.1.2. Словник

- Корінь дерева - це вершина, яка не має батьків. У дереві може бути не більше однієї кореневої вершини (вершина A у наведеному вище прикладі).
- Ребро - це зв'язок від батька до дочірнього елемента (всі зв'язки на рисунку).
- Вузол без нащадків називається листовим (E,J,K,H та I).
- Діти одного батька називаються братами і сестрами (B, C, D - брати і сестри A, а E, F - брати і сестри B).
- Вершина p є предком вершини q, якщо існує шлях від кореня до q і на цьому шляху зустрічається p на цьому шляху. Вершина q називається нащадком p. Наприклад, A, C і G є її предками.
- Множина всіх вершин на заданій глибині називається рівнем дерева (B, C і D - це один і той же рівень). Коренева вершина знаходиться на нульовому рівні.

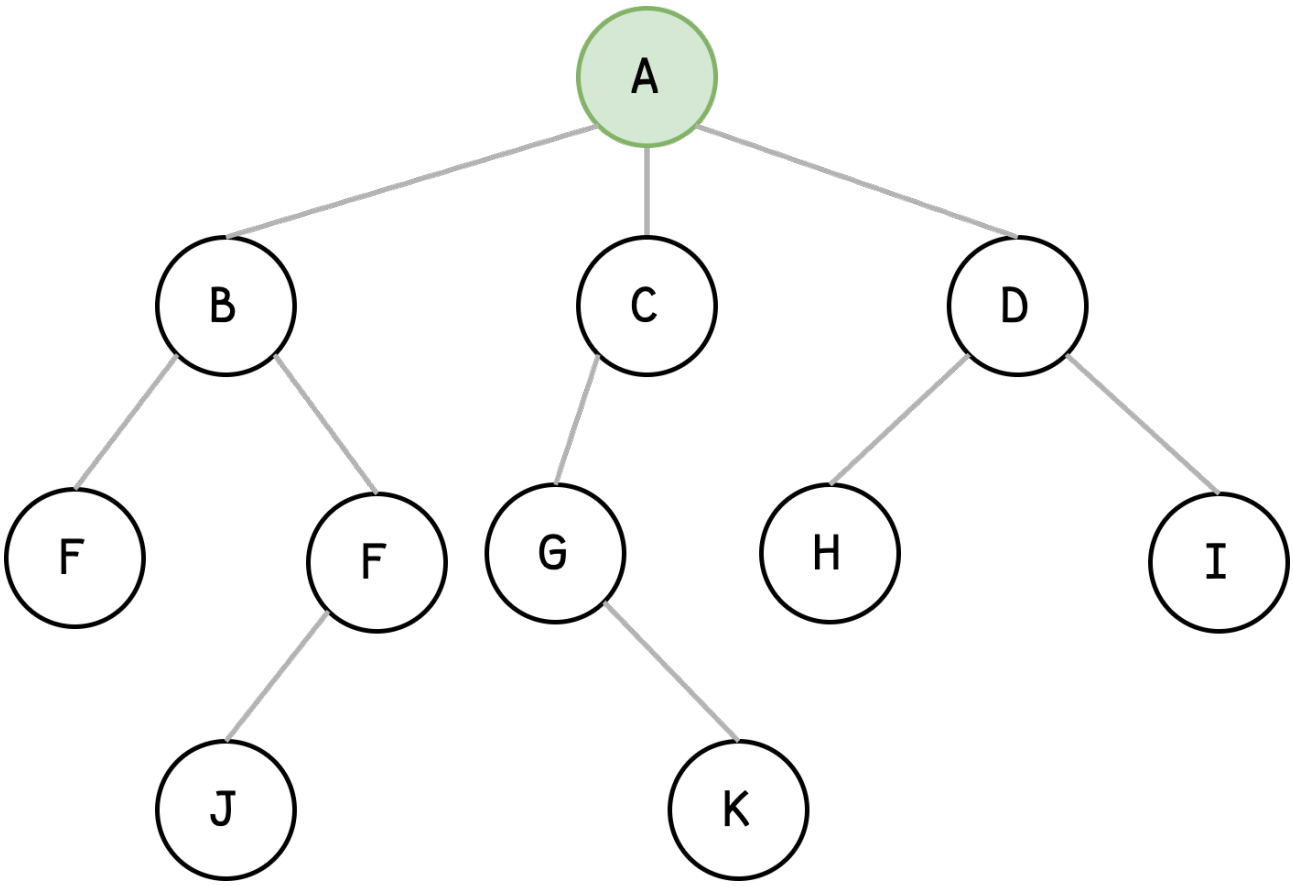

Рис. 1. Приклад бінарного дерева

- Глибина вершини - це довжина шляху від кореня до вершини (глибина G дорівнює 2, A - C - G).
- Висота вершини - це довжина шляху від цієї вершини до найглибшої вершини. Висота дерева - це довжина шляху від кореня до найглибшої вершини дерева. Дерево з одним коренем (кореневим) має нульову висоту. У попередньому прикладі висота B дорівнює 2 (B - F - J).
- Висота дерева - це максимальна висота серед усіх вершин дерева, а глибина дерева - максимальна глибина серед усіх вершин дерева. Для заданого дерева глибина і висота повертають однакове значення. Але для окремих вершин ми можемо отримати різні результати.
- Розмір вузла - це кількість нащадків, які він має, включаючи самого себе (розмір піддерева C дорівнює 3).
- Якщо кожна вершина дерева має лише одного нащадка (крім листків), то ми називаємо такі дерева деревами нахилу. Якщо кожна вершина має лише одного лівого нащадка, то такі дерева називаються деревами лівого нахилу. Аналогічно, якщо кожна вершина має тільки правого нащадка, то ми називаємо такі дерева деревами правого нахилу.

3.1.3. Бінарні дерева

Дерево називається бінарним, якщо кожна вершина має нульового нащадка, одного нащадка або двох нащадків. Порожнє дерево також є правильним бінарним деревом. Ми можемо візуалізувати бінарне дерево як таке, що складається з кореня і двох роз'єднаних бінарних дерев, які називаються лівим і правим піддеревами кореня.

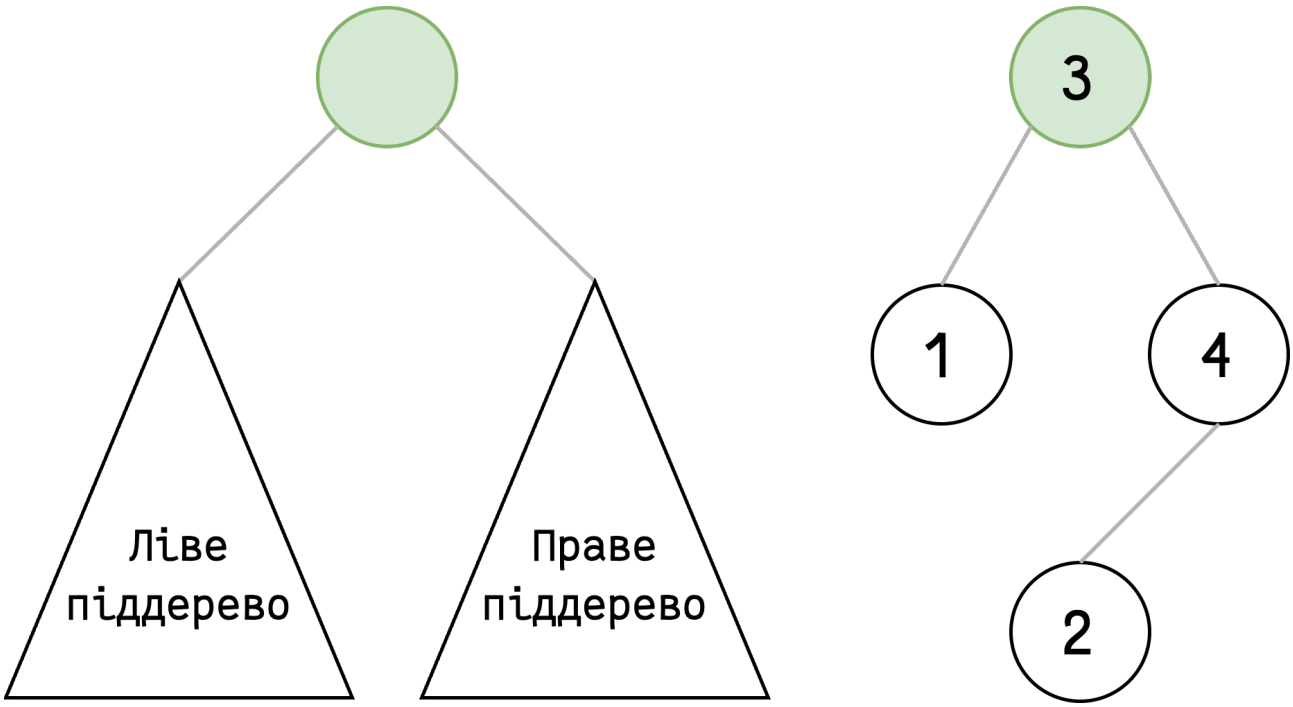

Рис. 2. Бінарні дерева

Бінарні дерева пошуку (BST)

У попередніх розділах ми обговорювали різні представлення дерев, і у всіх них ми не накладали жодних обмежень на дані про вузли. Як наслідок, для пошуку елемента нам потрібно шукати його і в лівому піддереві, і в правому піддереві. Через це у найгіршому випадку складність операції пошуку становить $O(n)$.

У цьому розділі ми розглянемо інший варіант бінарних дерев: Бінарні дерева пошуку (Binary Search Trees, BST). Як випливає з назви, основне використання цього подання - це пошук. У цьому поданні ми накладаємо обмеження на тип даних, які може містити вузол. В результаті, це зменшує середню операцію пошуку у найгіршому випадку до $O(\log n)$.

Властивості бінарних дерев пошуку

У бінарних деревах пошуку всі елементи лівого піддерева мають бути меншими за кореневі дані, а всі елементи правого піддерева - більшими за кореневі дані. Це називається властивістю бінарного дерева пошуку. Зауважте, що ця властивість повинна виконуватись у кожній вершині дерева.

- Ліве піддерево вузла містить лише вузли з ключами, меншими за ключ вузла.
- Праве піддерево вершини містить лише вершини з ключами, більшими за ключ вершини.
- І ліве, і праве піддерева також повинні бути бінарними деревами пошуку.

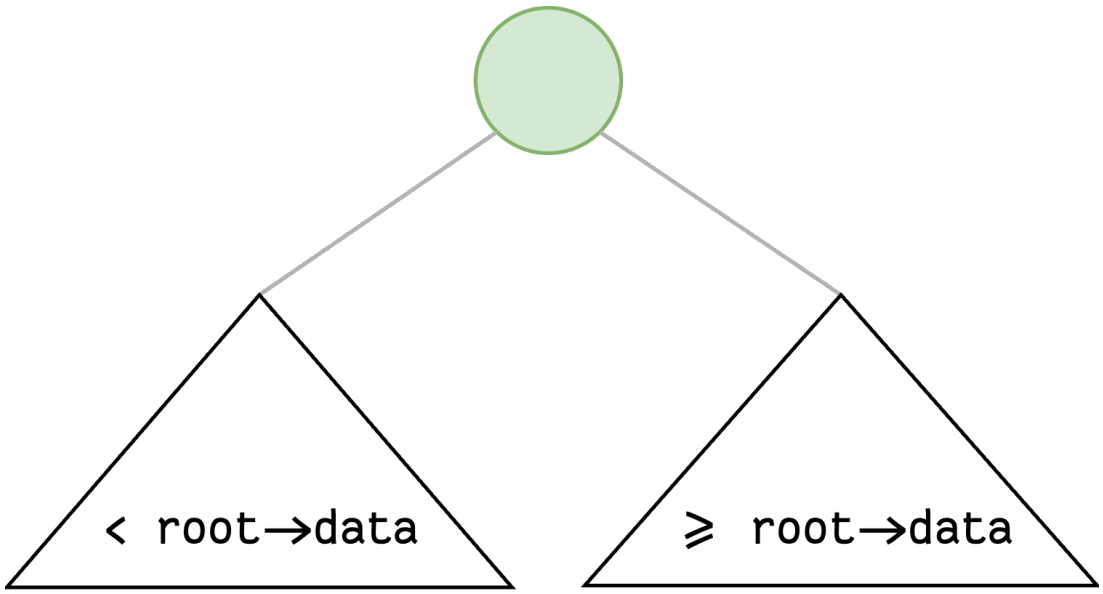

Рис. 3. Бінарні дерева пошуку

Операції над бінарними деревам

- Знайти/Знайти мінімальний/Знайти максимальний елемент у бінарному дереві пошуку
- Вставка елемента в бінарне дерево пошуку
- Видалення елемента з бінарного дерева пошуку
- Знаходження k-го найменшого елемента в дереві
- Сортування елементів бінарного дерева пошуку та багато іншого

Важливі зауваження щодо бінарних дерев пошуку

- Оскільки кореневі дані завжди знаходяться між даними лівого та правого піддерева, обхід у зворотному порядку по бінарному дереву пошуку призводить до отримання відсортованого списку.
- Розв'язуючи задачі на бінарних деревах пошуку, спочатку ми обробляємо ліве піддерево, потім кореневі дані, і, нарешті, праве піддерево. Це означає, що в залежності від задачі змінюється лише проміжний крок (обробка кореневих даних), а перший і третій кроки ми не чіпаємо.
- Якщо ми шукаємо елемент, і кореневі дані лівого піддерева менші за елемент, який ми хочемо знайти, то пропускаємо його. Те ж саме відбувається і з правим піддеревом. Завдяки цьому бінарні дерева пошуку займають менше часу для пошуку елемента, ніж звичайні бінарні дерева. Іншими словами, бінарні дерева пошуку розглядають або ліве, або праве піддерево для пошуку елемента, але не обидва.
- Основними операціями, які можна виконувати над бінарним деревом пошуку (БДП), є вставка елемента, видалення елемента та пошук елемента. При виконанні цих операцій над БДП висота дерева щоразу змінюється. Таким чином, існують відмінності у часових складностях найкращого, середнього та найгіршого випадків.
- Основні операції над бінарним деревом пошуку займають час, пропорційний висоті дерева. Для повного бінарного дерева з вершиною n такі операції виконуються за $O(\log n)$ часу у найгіршому випадку. Якщо ж дерево є лінійним ланцюжком з n вузлів (skew-дерево), то ті ж самі операції займають $O(n)$ часу в найгіршому випадку.

4. Про програму

Програма написана на мові програмування C++ з використанням крос-платформового інструментарію розробки програмного забезпечення Qt та інтегрованого середовища розробки (IDE) QtCreator, яке спрощує розробку GUI-додатків.

4.1. Про Qt

Qt (вимовляється як "кьют") - це крос-платформне програмне забезпечення для створення графічних інтерфейсів користувача, а також крос-платформних додатків, які працюють на різних програмних і апаратних платформах, таких як Linux, Windows, macOS, Android або вбудованих системах, з невеликими змінами або без змін в базовій кодовій базі, залишаючись при цьому нативним додатком з нативними можливостями і швидкістю.

Наразі Qt розробляється The Qt Company, публічною компанією, що котирується на біржі, та Qt Project з відкритим вихідним кодом, за участю окремих розробників та організацій, що працюють над розвитком Qt. Qt доступна як під комерційними ліцензіями, так і під ліцензіями з відкритим кодом GPL 2.0, GPL 3.0 та LGPL 3.0.

4.2. Структура

`bintree.h`
: Декларація класу `bintree<typename T>`, вузла `node<typename T>` та його методів.

`food.h`
: Декларація 100-елементного масиву рядків з назвами страв.

`mainwindow.h`
: Декларація наслідування класу `MainWindow` з `QMainWindow`, структура даних `nodeData` типу `T` для `bintree<typename T>`. Включення заголовків `Qt` та `bintree.h`.

`main.cpp`
: Ініціалізація `QApplication` та `MainWindow`.

`mainwindow.cpp`
: Ініціалізація методів `MainWindow`, логіка слотів графічного інтерфейсу користувача.

`mainwindow.ui`
: XML-опис графічного інтерфейсу користувача.

`recources.qrc`
: Ресурс-файл Qt з медіа, які необхідні для програми.

`Lab1.pro`
: Pro-файл QtCreator.

4.3. Методи `bintree.h`

```
1  template <typename T>
2  struct node {
3      T* data;
4      int key;
5
6      node<T>* left;
7      node<T>* right;
8  };
9
10 template <typename T>
11 class bintree {
```

```
12 public:
13     bintree();
14     ~bintree();
15 
16     node<T>* insert(int key);
17     node<T>* search(int key);
18     node<T>* searchMax();
19     node<T>* searchMin();
20     int leftDepth();
21     int rightDepth();
22 
23     static int leftDepth(node<T>* leaf);
24     static int rightDepth(node<T>* leaf);
25     static int depth(node<T>* leaf);
26     static int getMaxWidth(node<T>* leaf);
27     void destroy(int key);
28 
29     void destroyTree();
30 
31     void inorderPrint();
32     void postorderPrint();
33     void preorderPrint();
34 
35     node<T>* root;
36 private:
37     void destroyTree(node<T>* leaf);
38 
39     node<T>* insert(int key, node<T>* leaf);
40     node<T>* search(int key, node<T>* leaf);
41         node<T>* searchMax(node<T>* leaf);
42         node<T>* searchMin(node<T>* leaf);
43     static int getWidth(node<T>* leaf, int level);
44     node<T>* destroy(int* key, node<T>* leaf);
45 
46     void inorderPrint(node<T>* leaf);
47     void postorderPrint(node<T>* leaf);
48     void preorderPrint(node<T>* leaf);
49 };
```

Для реалізації методів, див. файл `bintree.h`.

Публічні нестатичні методи

`node<T>* insert(int key)`
: Створює вузол з вагою `k`, вставляє його в дерево та повертає вказівник на нього. Складність: $O(n)$.

`node<T>* search(int key)`
: Шукає вузол з вагою `k` та повертає його адрес. Якщо вузол не вдалося знайти, то повертає `nullptr`. Складність: $O(n)$.

`node<T>* searchMax()`
: Шукає вузол з найбільшою вагою, та повертає вказівник на нього. Якщо такий вузол не вдалося знайти, то повертає `nullptr`. Складність: $O(n)$.

`node<T>* searchMin()`
: Шукає вузол з найменшою вагою і та повертає вказівник на нього. Якщо такий вузол не вдалося знайти, то повертає `nullptr`. Складність: $O(n)$.

`int leftDepth()`
: Максимальна кількість вузлів вліво після кореня. Складність: $O(n)$.

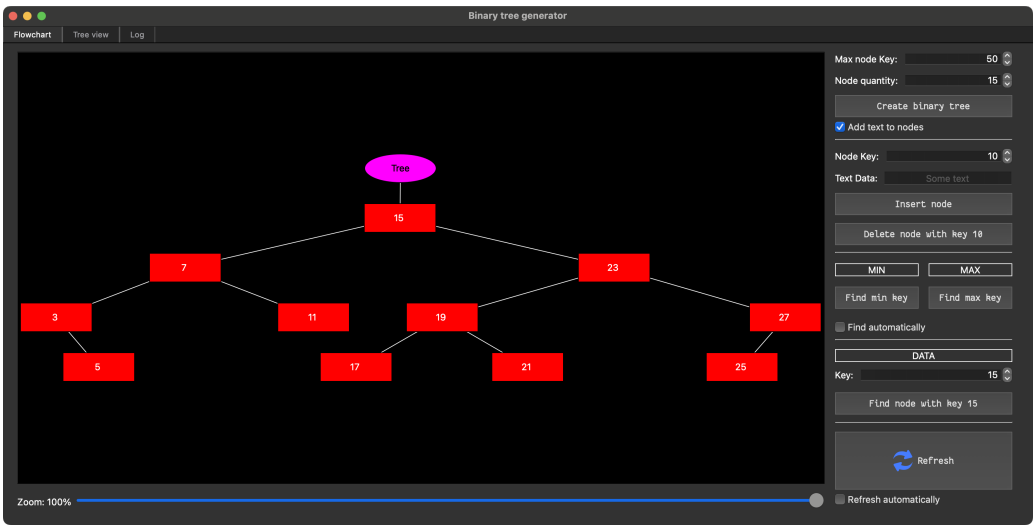

Рис. 4. Інтерфейс програми

`int rightDepth()`
: Максимальна кількість вузлів вправо після кореня. Складність: $O(n)$.

`void destroy(int key)`
: Видаляє вузол з вагою `key`. Якщо такого вузла немає, то нічого не відбувається. Складність: $O(n)$.

`void destroyTree()`
: Видаляє дерево. Складність: $O(n)$.

`void inorderPrint()`
: Інордерний вивід вмісту дерева в стандартний потік. Складність: $O(n)$.

`void postorderPrint()`
: Постордерний вивід вмісту дерева в стандартний потік. Складність: $O(n)$.

`void preorderPrint()`
: Преордерний вивід вмісту дерева в стандартний потік. Складність: $O(n)$.

Публічні поля

`node<T>* root`
: Адрес кореневого вузла.

Статичні методи

`static int leftDepth(node<T>* leaf)`
: Максимальна кількість вузлів вліво після вузла `leaf`. Складність: $O(n)$.

`static int rightDepth(node<T>* leaf)`
: Максимальна кількість вузлів вправо після вузла `leaf`. Складність: $O(n)$.

`static int depth(node<T>* leaf)`
: Глибина дерева після вузла `leaf`. Складність: $O(n)$.

`static int depth(node<T>* leaf)`
: Максимальна ширина дерева після вузла `leaf`.

4.4. Опис GUI

Код збирається під операційну систему, тому фінальний формат файлу додатку може бути різним на різних операційних системах.

① Графічне середовище `QGraphicsView`:

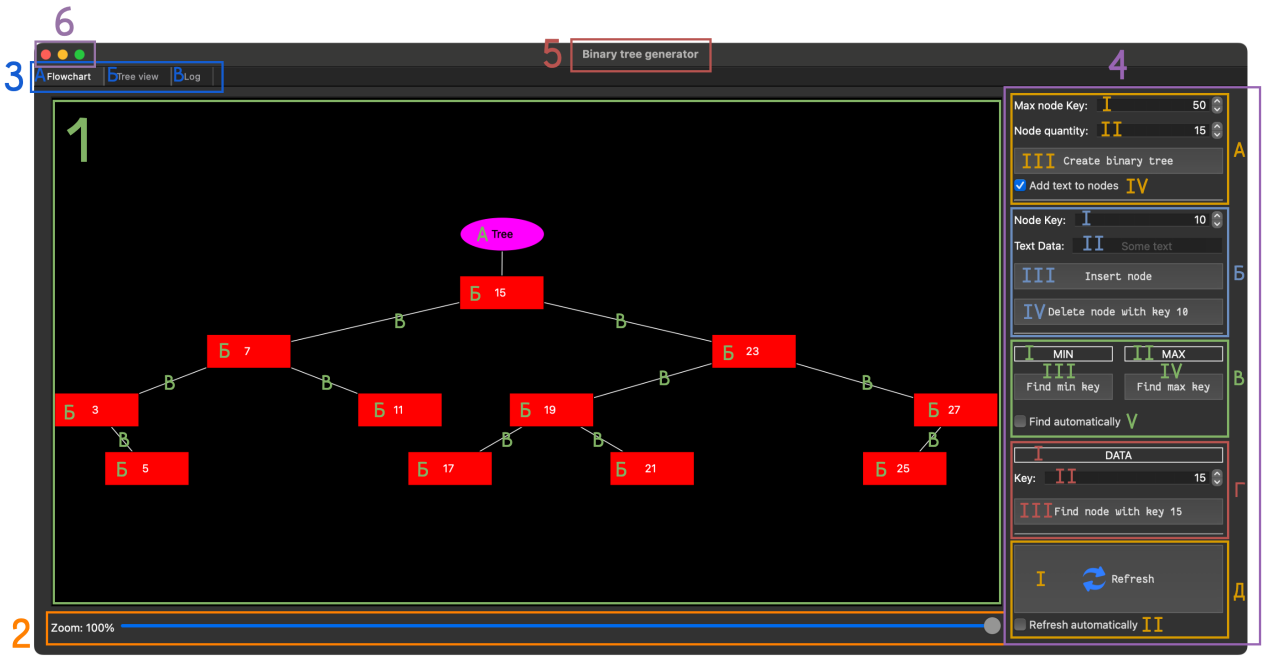

Рис. 5. Мапа інтерфейсу на рис. 4

- Ⓐ Позначка дерева.
- Ⓑ Вузли.
- Ⓒ Ребра.

2. Слайдер маштабування графічного середовища.
3. Вкладки:
   - Ⓐ Діаграма. (Ця вкладка відкрита наразі).
   - Ⓑ Вигляд в формі дерева.
   - Ⓒ Лог подій в програмі.
4. Панель управління:
   - Ⓐ Створення списку:
     i. Найбільший можливий індекс.
     ii. Кількість вузлів.
     iii. Кнопка створення дерева.
     iv. Прапорець "Додавати текст до вузлів".
   - Ⓑ Видалення та вставка вузла:
     i. Ключ (вага) вузла.
     ii. Текстові дані. Це поле використовується лише для вставки.
     iii. Кнопка вставки вузла.
     iv. Кнопка видалення вузла.
   - Ⓒ Пошук найбільшого та найменшого вузлів:
     i. Найменший вузол.
     ii. Найбільший вузол.
     iii. Кнопка знаходження найменшого вузла.
     iv. Кнопка знаходження найбільшого вузла.
     v. Прапорець автоматичного пошуку найменшого та найбільшого при будь-якій

зміні дерева.

Г) Пошук вузла за ключем:

i. Текстові дані з знайденого вузла.

ii. Поле для введення ключа (ваги) шуканого вузла.

iii. Кнопка пошуку.

Д) Оновлення дерева:

i. Кнопка оновлення дерева.

ii. Прапорець автоматичного оновлення при зміні дерева.

5) Назва програми

6) Елементи управління вікном.

5. Тестування

Згенеруємо довільне дерево (див. рис. 6, стр. 10). Для цього:

1) Виберемо максимальну кількість вузлів 10 за допомогою `4-А-II`.
2) Створимо нове дерево нажавши на `4-А-III`.
3) Оновимо дерево на екрані за допомогою `4-Д-I`.

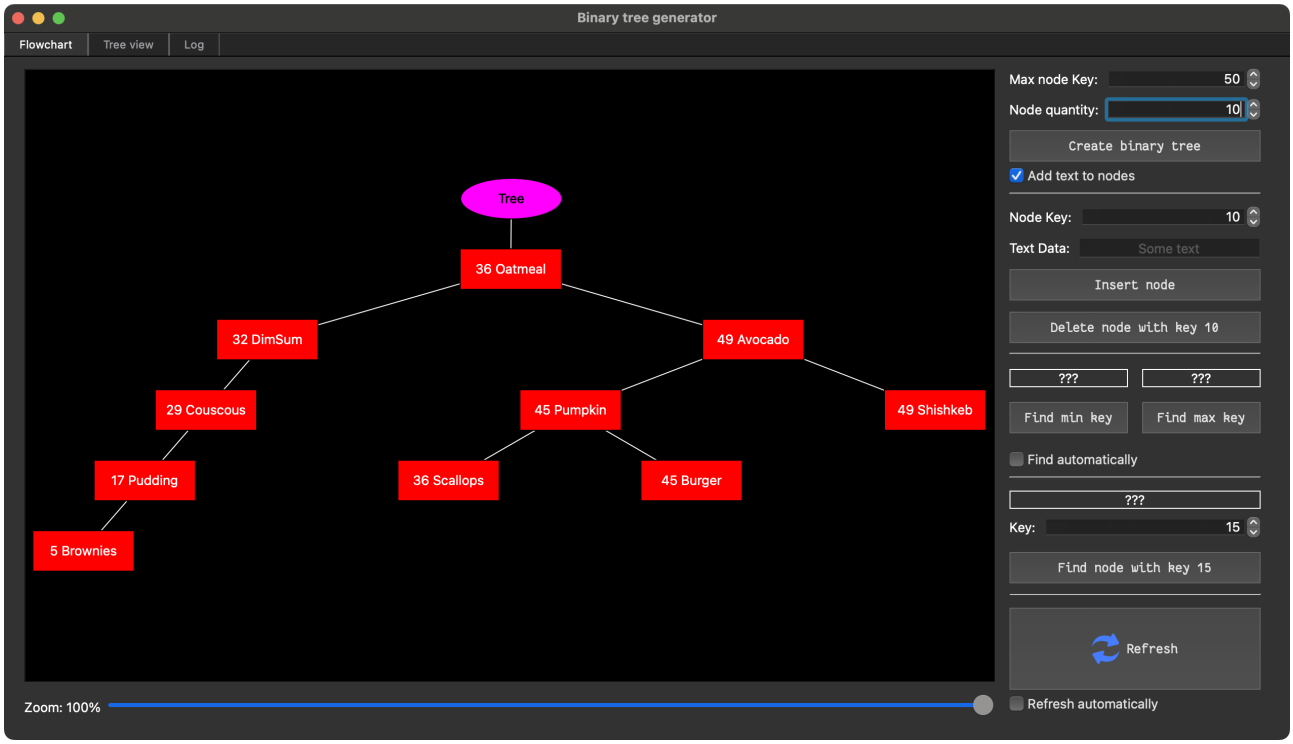

Рис. 6. Отримане дерево

Видалимо з дерева на рис. 6 вузли з вагою 32 та 49. Знайдемо вузли з найбільшим та найменшим ключами (див. рис. 7, стр. 11). Для цього:

1) Введемо 32 у `4-Б-I`.
2) Нажмемо на `4-Б-IV`.
3) Введемо 49 у `4-Б-I`.
4) Нажмемо на `4-Б-IV`.
5) Нажмемо на `4-В-III` та `4-В-IV`.
6) Оновимо дерево на екрані за допомогою `4-Д-I`.

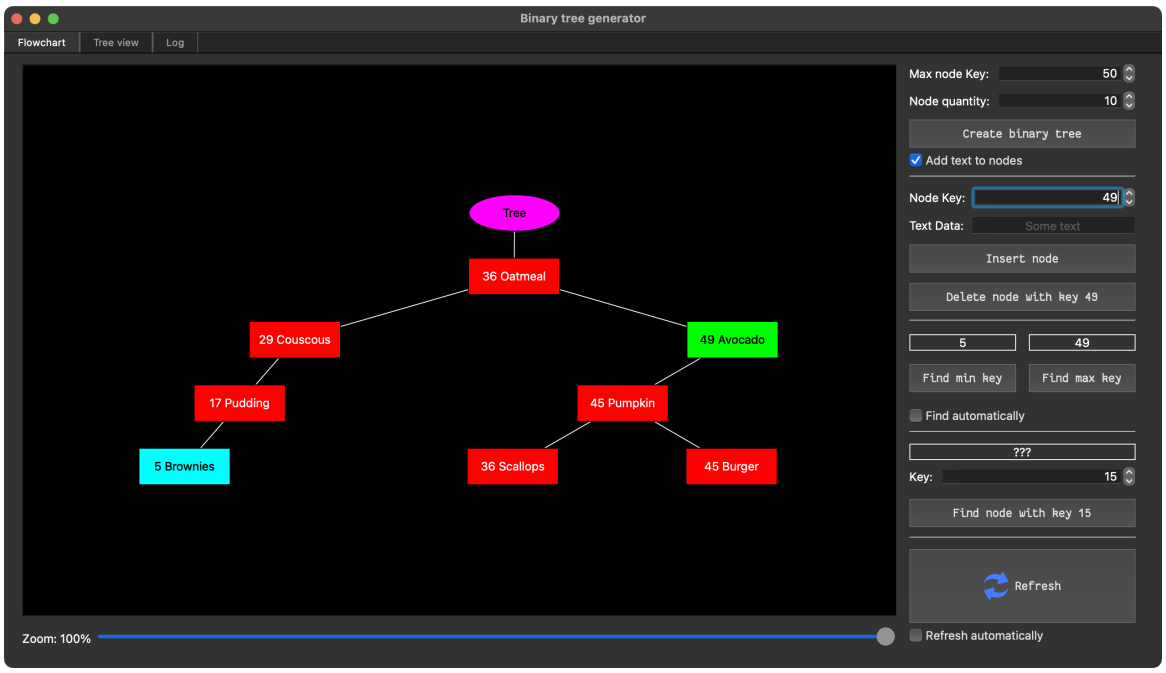

Рис. 7. Видалення та пошук найбільшого і наймешого

Для полегшення наступних операцій, увімкнемо прапорці `4-В-V` та `4-Д-II`.

Втсавимо вузел з ключем 30 та знайдемо вузел з вагою 45 на рис. 7. Для цього (див. рис. 8, стр. 11):

1. Введемо 30 у `4-Б-I`.
2. Нажмемо `4-Б-III`.
3. Введемо 45 у `4-Г-II`.
4. Нажмемо на `4-Г-III`.

Так як ми увімкнули прапорці `4-В-V` та `4-Д-II` то нажимати на `4-В-III` та `4-В-IV` і `4-Д-I` не потрібно.

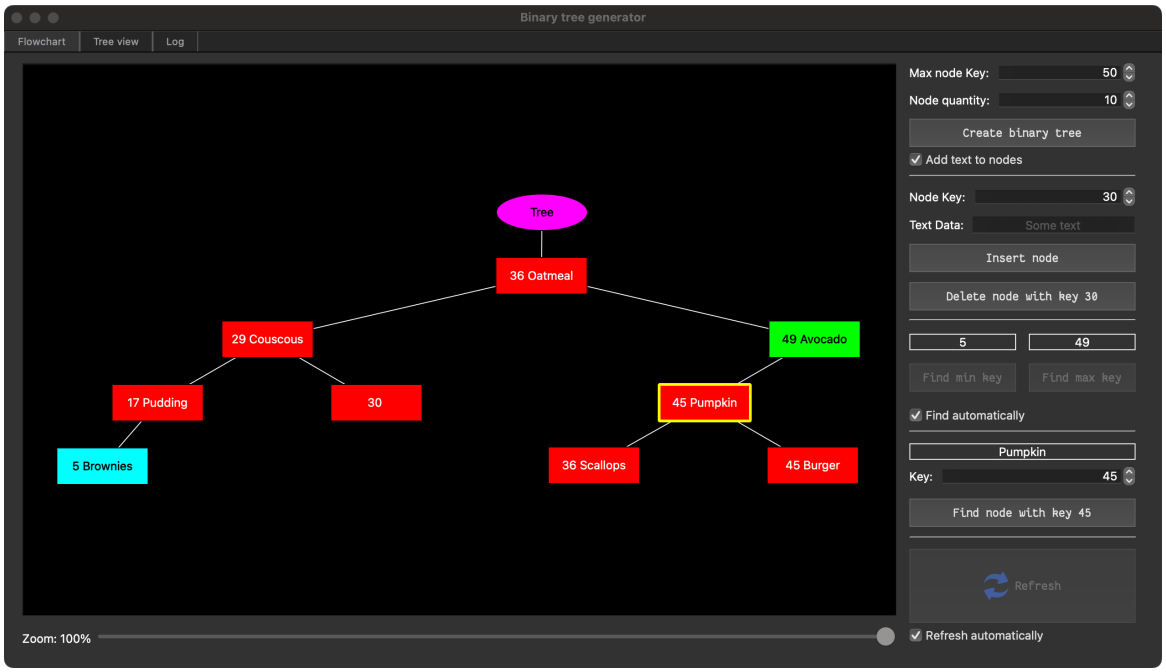

Рис. 8. Вставка і пошук елементу

Тим часом вкладка 3-Б набула наступного вигляду:

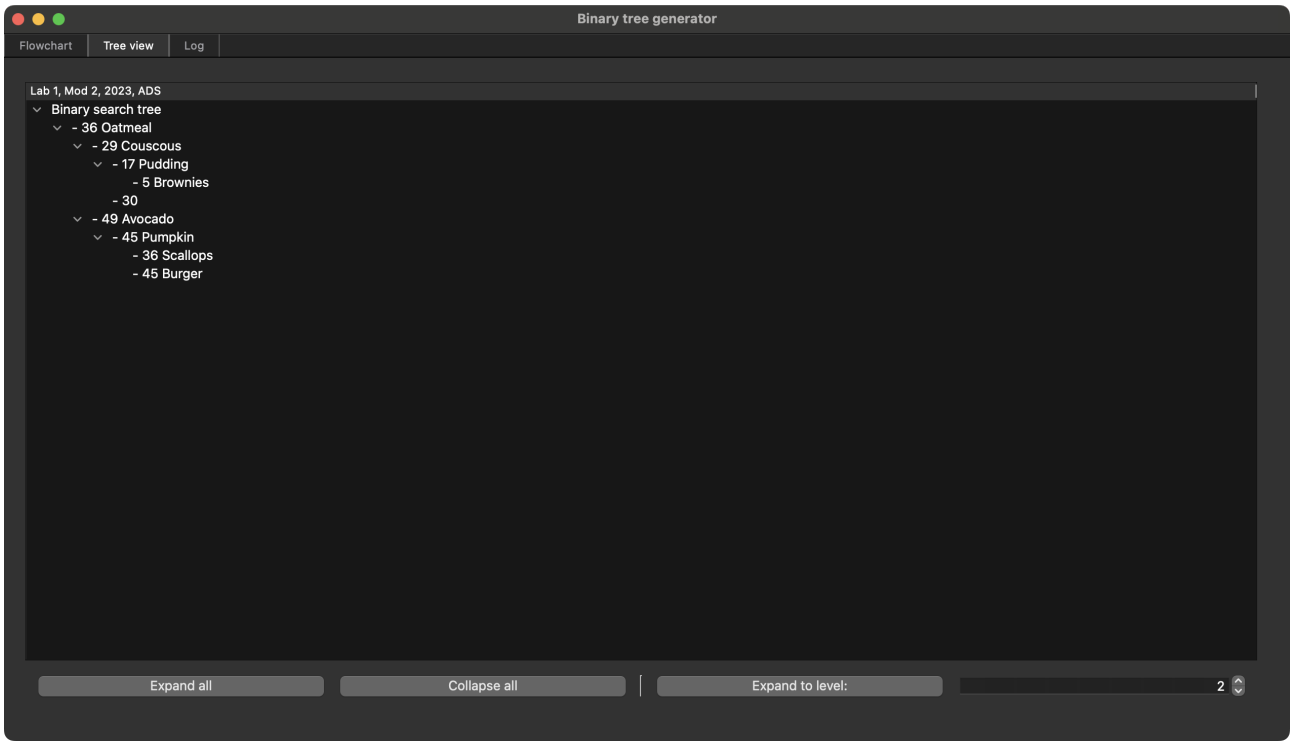

Рис. 9. Вкладка Tree View

6. Висновки

- Ознайомився:
  - З основними алгоритмами обробки типів даних ієрархічної структури.
    - ◁ Пошуком найменшого елементу.
    - ◁ Пошуком найбільшого елементу.
    - ◁ Способами ітерації по дереву.
    - ◁ Вставкою в дерево.
    - ◁ Видаленням з дерева.
    - ◁ Пошуком глибини дерева.
  - З виведенням дерева на екран.
  - З розробкою додатків за допомогою `Qt`.
- Вивчив:
  - Основні принципи побудови дерев двійкового пошуку та алгоритми їх обробки.
  - Тип даних піраміда і основні алгоритми обробки.
- Реалізував:
  - Створення випадкового дерева.
  - Видалення елементу з дерева.
  - Вставку елемента в дерево.
  - Пошук елемента в дереві.
  - Графічний інтерфейс користувача.
  - Виведення дерева на екран.

Перелік джерел посилань

[1] Narasimha Karumanchi. *Data Structures and algorithms made easy: Concepts, problems, Interview Questions*. CareerMonk Publications, 2020.

[2] Ткачук В.М. *Алгоритми та структура даних*. Видавництво Прикарпатського національного університету імені Василя Стефаника, 2016, с. 286.

[3] Панік Леонід Олександроович Ільман Валерій Михайлович Іванов Олександр Петрович. *АЛГОРИТМИ ТА СТРУКТУРИ ДАНИХ Навчальний посібник*. Дніпропетровський національний університет залізничного транспорту імені академіка В. Лазаряна, 2019, с. 134.