МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Харківський національний університет імені В.Н.Каразіна

Факультет математики і інформатики

Кафедра теоретичної та прикладної інформатики

Індивідуальна робота № 2

з курсу «Алгоритми і структури даних»
(назва дисципліни)

Виконала: студентка 2 курсу групи мф-21
напряму підготовки (спеціальності)
Комп'ютерні науки
122 Комп'ютерні науки
(шифр і назва напряму підготовки (спеціальності))
Лобанова Д.В.
(прізвище й ініціали студента)

Харків – 2024

Мета програми:
Задане бінарне дерево пошуку
a) перевірити, чи є це дерево AVL деревом;
b) перетворити задане бінарне дерево пошуку у відсортований масив.

Вхідні дані:

1. Кількість елементів у дереві.
2. Список значень, які потрібно вставити в дерево.

Вихідні дані:

1. Інформація про те, чи є дерево AVL.
2. Відсортований масив, що містить елементи дерева.

Обґрунтування вибору структури даних:

Бінарне дерево пошуку - це структура даних корисна для виконання операцій вставки, пошуку та видалення елементів. Вона забезпечує впорядковане зберігання елементів, що дозволяє здійснювати сортування за допомогою обходу в порядку зростання.

AVL-дерево - це бінарне дерево пошуку, де після кожної операції вставки чи видалення перевіряється, чи задовольняє дерево умови AVL (різниця висот лівого і правого піддерев будь-якого вузла не повинна бути більше 1).

Масив - оскільки вхідні дані мають бути відсортовані, використання масиву дозволяє легко вивести елементи у відсортованому вигляді після обробки дерева за допомогою обходу.

Опис програми

```
#include <iostream>
#include <algorithm>
using namespace std;
```

Структура вузла дерева

```
struct Tree {
    int val;
    Tree* left;
    Tree* right;

    // Конструктор для ініціалізації вузла
    Tree(int x) : val(x), left(nullptr), right(nullptr) {}
};
```

Функція для вставки елемента в бінарне дерево пошуку

```
Tree* addElem(Tree* root, int val) {
    if (!root) {
        return new Tree(val);
    }
    if (val < root->val) { // Якщо значення менше за значення кореня, вставляємо в ліве піддерево
        root->left = addElem(root->left, val);
    } else { // Якщо значення більше або рівне значенню кореня, вставляємо в праве піддерево
        root->right = addElem(root->right, val);
    }
    return root;
}
```

Функція для обчислення висоти дерева

```
int heightTree(Tree* root) {
    if (!root) return 0;
    return 1 + max(heightTree(root->left), heightTree(root->right)); // Висота = 1 + максимальна висота лівого і правого піддерева
}
```

Функція для перевірки, дерево AVL чи ні

```
bool isAVL(Tree* root, int& height) {
    if (!root) {
        height = 0;
        return true;
    }

    int leftHeight = 0, rightHeight = 0;
    // Рекурсивно перевіряємо ліве та праве піддерево (якщо хоча б одне з піддерев не є AVL, то все дерево не буде AVL)
    if (!isAVL(root->left, leftHeight) || !isAVL(root->right, rightHeight))
        return false;

    height = 1 + max(leftHeight, rightHeight); // Обчислюємо висоту поточного дерева
    return abs(leftHeight - rightHeight) <= 1; // перевірка на AVL умову
}
```

Функція для підрахунку кількості вузлів у дереві

```
int countNodes(Tree* root) {
    if (!root) return 0;
    return 1 + countNodes(root->left) + countNodes(root->right); // Підраховуємо вузли лівого та правого піддерев і додаємо 1
}
```

Функція для заповнення масиву відсортовоного при обході бінарного дерева пошуку

```
void inOrderArrTree(Tree* root, int arr[], int& index) {
    if (!root) return;
    inOrderArrTree(root->left, arr, index); // Спочатку обходимо ліве піддерево
    arr[index++] = root->val; // Додаємо значення вузла в масив
    inOrderArrTree(root->right, arr, index); // Потім обходимо праве піддерево
}
```

Функція для перевірки вводу числа

```
int getValidInt(const string& prompt) {
    string input;
    int result;

    while (true) {
        cout << prompt;
        getline(cin, input);

        try {
            result = stoi(input); // Перетворення рядка на число
            return result; // Повертаємо результат
        } catch (const invalid_argument&) {
            cout << "Invalid input. Please enter a valid number.\n";
        } catch (const out_of_range&) {
            cout << "The number is out of range. Please try again.\n";
        }
    }
}
```

Функція для звільнення пам'яті для дерева

```
void deleteTree(Tree* root) {
    if (!root) return;

    deleteTree(root->left); // Видаляємо ліве піддерево
    deleteTree(root->right); // Видаляємо праве піддерево
    delete root; // Видаляємо поточний вузол
}
```

Основна функція програми

```
int main() {
    Tree* root = nullptr;
    int n;

    // Введення кількості елементів з перевіркою
```

```
    do {
        n = getValidInt("Enter the number of elements: ");
        if (n < 0) {
            cout << "Please enter a positive integer.\n";
        }
    } while (n < 0);

    // Введення значень для дерева з перевіркою вводу
    cout << "Enter values for the binary search tree:\n";
    for (int i = 0; i < n; i++) {
        int val = getValidInt("Enter value " + to_string(i + 1) + ": ");
        root = addElem(root, val);
    }

    // Перевірка чи дерево є AVL
    int height = 0;
    if (isAVL(root, height)) {
        cout << "The tree is AVL\n";
    } else {
        cout << "The tree is not AVL\n";
    }

    // Перетворення дерева у відсортований масив
    int n_count = countNodes(root);
    int* sorted_arr = new int[n_count];
    int index = 0;
    inOrderArrTree(root, sorted_arr, index);

    // Виведення відсортованого масиву
    cout << "Sorted arr: ";
    for (int i = 0; i < n_count; ++i) {
        cout << sorted_arr[i] << " ";
    }
    cout << endl;

    // Звільнення пам'яті
    delete[] sorted_arr;
    // Звільнення пам'яті для дерева
    deleteTree(root);

    return 0;
}
```

Код програми

```
/* Задане бінарне дерево пошуку
* a) перевірити, чи є це дерево AVL деревом;
* b) перетворити задане бінарне дерево пошуку у відсортований масив.
* Daryna Iobanova mf-21
*/
#include <iostream>
#include <algorithm>
using namespace std;

// Структура вузла дерева
struct Tree {
   int val;
   Tree* left;
   Tree* right;

   // Конструктор для ініціалізації вузла
   Tree(int x) : val(x), left(nullptr), right(nullptr) {}
};

// Функція для вставки елемента в бінарне дерево пошуку
Tree* addElem(Tree* root, int val) {
   if (!root) {
       return new Tree(val);
   }
   if (val < root->val) { // Якщо значення менше за значення кореня, вставляємо в ліве піддерево
       root->left = addElem(root->left, val);
   } else { // Якщо значення більше або рівне значенню кореня, вставляємо в праве піддерево
       root->right = addElem(root->right, val);
   }
   return root;
}

// Функція для обчислення висоти дерева
int heightTree(Tree* root) {
   if (!root) return 0;
   return 1 + max(heightTree(root->left), heightTree(root->right)); // Висота = 1 + максимальна висота
лівого і правого піддерева
}

// Функція для перевірки, дерево AVL чи ні
bool isAVL(Tree* root, int& height) {
   if (!root) {
       height = 0;
       return true;
   }

   int leftHeight = 0, rightHeight = 0;
   // Рекурсивно перевіряємо ліве та праве піддерево (якщо хоча б одне з піддерев не є AVL, то все дерево
не буде AVL)
   if (!isAVL(root->left, leftHeight) || !isAVL(root->right, rightHeight))
       return false;

   height = 1 + max(leftHeight, rightHeight); // Обчислюємо висоту поточного дерева
   return abs(leftHeight - rightHeight) <= 1; // перевірка на AVL умову
}

// Функція для підрахунку кількості вузлів у дереві
int countNodes(Tree* root) {
   if (!root) return 0;
   return 1 + countNodes(root->left) + countNodes(root->right); // Підраховуємо вузли лівого та правого
піддерев і додаємо 1
}

// Функція для заповнення масиву відсортовоного при обході бінарного дерева пошуку
void inOrderArrTree(Tree* root, int arr[], int& index) {
   if (!root) return;
   inOrderArrTree(root->left, arr, index); // Спочатку обходимо ліве піддерево
   arr[index++] = root->val; // Додаємо значення вузла в масив
   inOrderArrTree(root->right, arr, index); // Потім обходимо праве піддерево
}

// Функція для перевірки вводу числа
```

```cpp
int getValidInt(const string& prompt) {
    string input;
    int result;

    while (true) {
        cout << prompt;
        getline(cin, input);

        try {
            result = stoi(input); // Перетворення рядка на число
            return result; // Повертаємо результат
        } catch (const invalid_argument&) {
            cout << "Invalid input. Please enter a valid number.\n";
        } catch (const out_of_range&) {
            cout << "The number is out of range. Please try again.\n";
        }
    }
}

// Функція для звільнення пам'яті для дерева
void deleteTree(Tree* root) {
    if (!root) return;

    deleteTree(root->left); // Видаляємо ліве піддерево
    deleteTree(root->right); // Видаляємо праве піддерево
    delete root; // Видаляємо поточний вузол
}

int main() {
    Tree* root = nullptr;
    int n;

    // Введення кількості елементів з перевіркою
    do {
        n = getValidInt("Enter the number of elements: ");
        if (n < 0) {
            cout << "Please enter a positive integer.\n";
        }
    } while (n < 0);

    // Введення значень для дерева з перевіркою вводу
    cout << "Enter values for the binary search tree:\n";
    for (int i = 0; i < n; i++) {
        int val = getValidInt("Enter value " + to_string(i + 1) + ": ");
        root = addElem(root, val);
    }

    // Перевірка чи дерево є AVL
    int height = 0;
    if (isAVL(root, height)) {
        cout << "The tree is AVL\n";
    } else {
        cout << "The tree is not AVL\n";
    }

    // Перетворення дерева у відсортований масив
    int n_count = countNodes(root);
    int* sorted_arr = new int[n_count];
    int index = 0;
    inOrderArrTree(root, sorted_arr, index);

    // Виведення відсортованого масиву
    cout << "Sorted arr: ";
    for (int i = 0; i < n_count; ++i) {
        cout << sorted_arr[i] << " ";
    }
    cout << endl;

    // Звільнення пам'яті
    delete[] sorted_arr;
    // Звільнення пам'яті для дерева
    deleteTree(root);

    return 0;
}
```

Результати програми

Тест 1. Дерево є AVL

```
Enter the number of elements:3
 Enter values for the binary search tree:
Enter value 1:10
 Enter value 2:5
 Enter value 3:15
 The tree is AVL
Sorted arr: 5 10 15

Process finished with exit code 0
```

```
Enter the number of elements:4
 Enter values for the binary search tree:
Enter value 1:10
 Enter value 2:5
 Enter value 3:15
 Enter value 4:3
 The tree is AVL
Sorted arr: 3 5 10 15

Process finished with exit code 0
```

```
Enter the number of elements:5
 Enter values for the binary search tree:
Enter value 1:10
 Enter value 2:5
 Enter value 3:15
 Enter value 4:3
 Enter value 5:7
 The tree is AVL
Sorted arr: 3 5 7 10 15

Process finished with exit code 0
```

Тест 2. Дерево не є AVL

```
Enter the number of elements:3
 Enter values for the binary search tree:
Enter value 1:5
 Enter value 2:10
 Enter value 3:15
 The tree is not AVL
Sorted arr: 5 10 15

Process finished with exit code 0
```

```
Enter the number of elements:4
 Enter values for the binary search tree:
Enter value 1:5
 Enter value 2:10
 Enter value 3:15
 Enter value 4:7
 The tree is not AVL
Sorted arr: 5 7 10 15

Process finished with exit code 0
```

```
Enter the number of elements:5
 Enter values for the binary search tree:
Enter value 1:7
 Enter value 2:10
 Enter value 3:3
 Enter value 4:8
 Enter value 5:9
 The tree is not AVL
Sorted arr: 3 7 8 9 10

Process finished with exit code 0
```

Тест 3. Дерево з декількома елементами з однаковими значеннями

```
Enter the number of elements:7
 Enter values for the binary search tree:
Enter value 1:5
 Enter value 2:3
 Enter value 3:7
 Enter value 4:3
 Enter value 5:7
 Enter value 6:8
 Enter value 7:7
 The tree is not
 AVL
Sorted arr: 3 3 5 7 7 7 8

Process finished with exit code 0
```

```
Enter the number of elements:5
 Enter values for the binary search tree:
Enter value 1:5
 Enter value 2:3
 Enter value 3:7
 Enter value 4:3
 Enter value 5:10
 The tree is AVL
Sorted arr: 3 3 5 7 10

Process finished with exit code 0
```

Тест 4. Перевірка на некоректне введення кількості елементів

```
Enter the number of elements:kk
 Invalid input. Please enter a valid number.
Enter the number of elements:num
 Invalid input. Please enter a valid number.
Enter the number of elements:77777777777777777777777777777777777777777777777777777777777777777777777777777777777777777
 The number is out of range. Please try again.
Enter the number of elements:0999999999999999999999999ghhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhh
 The number is out of range. Please try again.
Enter the number of elements:
```

Тест 5. Введення некоректних значень

```
Enter the number of elements:7
 Enter values for the binary search tree:
Enter value 1:rrrrrrrrrrrr
 Invalid input. Please enter a valid number.
Enter value 1:12222222222233333333333333333333333333333333333333333333333333333333333
 The number is out of range. Please try again.
Enter value 1:kkkkkkkkkkkkkkkkkkkkkkkk7777777777777777777777777777777777777777777777777777
 Invalid input. Please enter a valid number.
Enter value 1:num
 Invalid input. Please enter a valid number.
Enter value 1:something
 Invalid input. Please enter a valid number.
Enter value 1:
```