Міністерство освіти і науки України
Національний технічний університет України «Київський політехнічний інститут імені Ігоря Сікорського"
Факультет інформатики та обчислювальної техніки

Кафедра

Звіт

з лабораторної роботи № 5 з дисципліни
«Теорія алгоритмів»

«Деревовидні структури даних»

Виконали ІА-31 Клим'юк В.Л, Самелюк А.С, Дук М.Д, Сакун Д.С
(шифр, прізвище, ім'я, по батькові)

Перевірив Степанов А.С
(прізвище, ім'я, по батькові)

Київ 2024

ЗМІСТ

1 МЕТА ЛАБОРАТОРНОЇ РОБОТИ

Мета роботи – вивчити основні підходи формалізації та імплементації алгоритмів побудови та обробки базових деревовидних структур даних.

2 ЗАВДАННЯ

Розробити алгоритм розв’язання задачі відповідно до варіанту. Виконати програмну реалізацію задачі. Не використовувати вбудовані деревовидні структури даних (контейнери). Зробити висновок по лабораторній роботі.

Варіанти завдань.

1. Задано фрагмент програми мовою C++. Надрукувати в алфавітному порядку всі ідентифікатори цієї програми, вказавши для кожного з них число входжень у текст програми. Для збереження ідентифікаторів використати структуру типу дерева, елементами якого є ідентифікатор і число його входжень у текст.

2. Побудувати дерево, елементами якого є дійсні числа. Обчислити середнє арифметичне усіх його елементів.

3. Побудувати дерево, що відображає формулу (a*(b+c))/d, де коренем дерева та його підкоренями є операції "*, +, -, /", а листками - змінні a, b, c, d. Надрукувати відповідне дерево .

4. Побудувати дерево, елементами якого є числа. Надрукувати дерево. Визначити кількість від'ємних та додатніх елементів дерева.

5. Побудувати двійкове дерево, елементами якого є символи. Визначити, чи знаходиться у цьому дереві елемент, значення якого вводиться з клавіатури. Якщо елемент знайдений, то підрахувати число його входжень.

6. Побудувати двійкове дерево пошуку з літер заданого рядка. Видалити з дерева літери, що зустрічаються більше одного разу. Вивести елементи дерева, що залишилися, при його постфіксному обході.

7. Заданий текст. Підрахувати кількість повторень кожного слова. Побудувати дерево із слів тексту. Слова, що зустрічаються найчастіше розмістити на верхньому рівні, на інших рівнях дерева розмістити слова з меншою кількістю повторень.

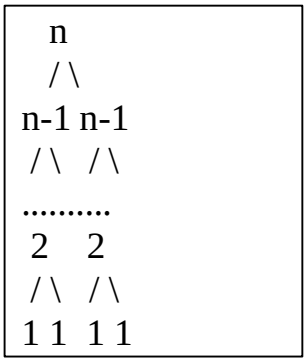
8. Побудувати бінарне дерево, елементами якого є дійсні числа. Знайти значення найбільшого елемента цього дерева та надрукувати його.

9. Заданий рядок символів латинського алфавіту. Побудувати дерево, в якому значеннями вершин є символи, що розміщуються на рівнях відповідно до кількості їх повторень у рядку.

10. Побудувати дерево, елементами якого є цілі числа. Визначити кількість вершин на n-му його рівні та кількість рівнів.

11. Побудувати дерево, що відображає формулу ((a+b)*c-d), де коренем дерева та його підкоренями є операції " +, -, *, /", а листками є змінні a, b, c, d . Вивести значення дерева-формули. Надрукувати відповідні піддерева y1=a+b, y2=y1*c, y3=y2-d.

12. Побудувати дерево наступного виду:



, де n -додатне ціле число

13. Побудувати бінарне дерево для зберігання даних виду: деталь, її кількість, постачальник. Забезпечити виконання операцій додавання нового елемента у дерево в діалоговому режимі та визначення постачальника найбільшої кількості деталей.

14. Побудувати дерево, елементами якого є символи. Визначити максимальну глибину дерева (число гілок на найбільшому з маршрутів від кореня дерева до листків).

15. Побудувати бінарне дерево, елементами якого є цілі числа. Підрахувати кількість вершин на n-му рівні цього дерева (нульовий рівень - корінь цього дерева) та надрукувати ці елементи.

16. Побудувати дерево, що відображає формулу ((a+b)/c)*d, де коренем дерева та його підкоренями є операції, а листками - змінні. Ввести значення змінних та визначити значення дерева-формули. Надрукувати відповідні

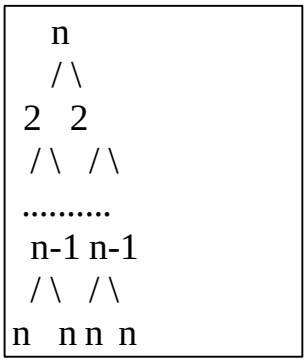
піддерева, наприклад: y1=a+b, y2:= y1/c, y3=y2*d.

17. Відповідно до виразу, що вводиться з клавіатури та може містити операції +,-,*,/, побудувати дерево-формулу та обчислити значення цієї формули.

18. Побудувати бінарне дерево, елементами якого є слова. Знайти у ньому значення слова, введеного з клавіатури, визначивши номер відповідного рівня.

19. Побудувати дерево, елементами якого є символи. Знайти довжину шляху (число гілок) від кореня до значення символа, введеного з клавіатури. Різновид дерева вибрати самостійно.

20. Побудувати дерево наступного типу:



, де n -додатне ціле число

21. Побудувати два бінарних дерева, елементами якого є цілі числа. Об'єднати їх, уникаючи дублювання елементів в сумарному дереві.

22. Побудувати дерево, елементами якого є дійсні числа. Поміняти місцями найбільше та найменше значення дерева.

23. Побудувати бінарне дерево для зберігання даних виду: найменування товару, його кількість, вартість одиниці. Забезпечити виконання операцій додавання нового елемента у дерево в діалоговому режимі та підрахунку загальної вартості вказаного товару.

24. Побудувати двійкове дерево пошуку, в вершинах якого знаходяться слова введені з клавіатури. Визначити кількість вершин дерева, що містять слова, які починаються на зазначену букву.

25. Побудувати дерево, елементами якого є цілі числа. Визначити кількість вузлових вершин (не листків) даного дерева та надрукувати їх координати

(номер рівня та номер гілки).

26. Написати програму, що будує дерево-формулу та перетворює в ньому всі піддерева, що відповідають формулам $((f_1 \pm f_2)*f_3)$, на піддерева виду $((f_1*f_3)\pm(f_2*f_3))$.

27. Побудувати дерево, елементами якого є символи. Визначити і вивести на друк усі термінальні вершини (листя) цього дерева.

28. Побудувати і вивести на екран бінарне дерево наступного виразу: 9 + 8 * (7 + (6 * (5 + 4) – (3 – 2)) +1)). Реалізувати постфіксний, інфіксний та префіксний обходи дерева і вивести відповідні вирази на екран.

29. Побудувати двійкове дерево пошуку, в вершинах якого знаходяться цілі числа. Визначити максимальну висоту дерева, тобто число ребер в найдовшому шляху від кореня до листків.

30. Побудувати двійкове дерево пошуку, в вершинах якого знаходяться цілі числа. Визначити висоту дерева, тобто число ребер в шляху від кореня до заданої вершини.

31. Побудувати дерево, елементами якого є цілі числа. Видалити з дерева усі листки, надрукувати початкове та модифіковане дерево.

32. Побудувати двійкове дерево пошуку, в вершинах якого знаходяться цілі числа. Визначити чи є дерево збалансованим.

33. Побудувати бінарне дерево, елементами якого є цілі числа. Вивести усі вузли, які мають лише правого потомка і усі вузли, які мають лише лівого потомка.

34. Побудувати дерево, елементами якого є цілі числа. Видалити з дерева усі вузли, які є батьками листків, надрукувати початкове та модифіковане дерево.

35. Реалізувати структуру даних бінарна піраміда на основі бінарного дерева, передбачити можливість додавання та видалення елементів.

3 ВИКОНАННЯ

3.1 Псевдокод алгоритмів

```
  BinaryTree
{
    Root { get; set; } = null;

    Function AddIdentifier(identifier)
    {
        if (_Root == null)
        {
            _Root = new TreeNode(identifier, 1);
            return;
        }

        _AddIdentifier(_Root, identifier);
    }

    Function _AddIdentifier(current, identifier)
    {
        if (current.Identifier == identifier)
        {
            current.Number++;
            return;
        }

        leftCompare = String.Compare(identifier, current.Identifier);

        if (leftCompare < 0)
        {
            if (current.Left == null)
                current.Left = new TreeNode(identifier, 1);
            else
                _AddIdentifier(current.Left, identifier);

        }
        else
        {
            if (current.Right == null)
                current.Right = new TreeNode(identifier, 1);
            else
                _AddIdentifier(current.Right, identifier);
        }
    }

    Function  TreeTraversal()
    {
        result = new List<Tuple<string, int>> ();

        if (_Root != null)
        {
            _TreeTraversal(result, _Root);
        }

        return result;
    }

    Function  _TreeTraversal(result, current)
    {
        if (current.Left != null)
            _TreeTraversal(result, current.Left);

        result.Add(new Tuple<string, int>(current.Identifier, current.Number));
```

```
            if (current.Right != null)
                _TreeTraversal(result, current.Right);
        }
    }
}
```

3.2 Програмна реалізація

```
namespace Classes
{
public class BinaryTree
{
    private TreeNode? _Root { get; set; } = null;

    public void AddIdentifier(string identifier)
    {
        if (_Root == null)
        {
            _Root = new TreeNode(identifier, 1);
            return;
        }

        _AddIdentifier(_Root, identifier);
    }

    private void _AddIdentifier(TreeNode current, string identifier)
    {
        if (current.Identifier == identifier)
        {
            current.Number++;
            return;
        }

        int leftCompare = String.Compare(identifier, current.Identifier);

        if (leftCompare < 0)
        {
            if (current.Left == null)
                current.Left = new TreeNode(identifier, 1);
            else
                _AddIdentifier(current.Left, identifier);

        }
        else
        {
            if (current.Right == null)
                current.Right = new TreeNode(identifier, 1);
            else
                _AddIdentifier(current.Right, identifier);
        }
    }

    public List<Tuple<string, int>> TreeTraversal()
    {
        List<Tuple<string, int>> result = new();

        if (_Root != null)
        {
            _TreeTraversal(result, _Root);
        }
```

```
        return result;
    }

    private void _TreeTraversal(List<Tuple<string, int>> result, TreeNode current)
    {
        if (current.Left != null)
            _TreeTraversal(result, current.Left);

        result.Add(new Tuple<string, int>(current.Identifier, current.Number));

        if (current.Right != null)
            _TreeTraversal(result, current.Right);
    }
}
}
```

1.1 BinaryTree

```
using System.Text.RegularExpressions;

namespace Classes
{
public class FileReader
{
    private static readonly HashSet<string> ExcludeList = new HashSet<string>()
    {
        "include", "return", "void", "class", "public", "private", "protected",
"new"
    };

    public async Task<BinaryTree> ReadFile(string file)
{
    BinaryTree result = new BinaryTree();

    using FileStream stream = File.OpenRead(file);
    using StreamReader reader = new StreamReader(stream);

    Regex regex = new Regex("(\\\"\\s*[a-zA-Z_0-9]+\\s*\\\")|([a-zA-Z_0-9]+)");

    while (!reader.EndOfStream)
    {
        string line = (await reader.ReadLineAsync())!;

        var matches = regex.Matches(line);

        foreach(var m in matches)
        {
            Match match = (Match)m;
            string value = match.Value;

            if (value.StartsWith('\"') || char.IsNumber(value[0]) || value[0] ==
'<' || value.EndsWith("::"))
                continue;

            if (!ExcludeList.Contains(value))
                result.AddIdentifier(value);
        }
    }

    return result;
}
```

```
}
}
```

1.2 FileReader

```
namespace Classes
{
public class TreeNode
{
    public string Identifier { get; set; }
    public int Number { get; set; }
    public TreeNode? Left { get; set; } = null;
    public TreeNode? Right { get; set; } = null;

    public TreeNode(string identifier, int number)
    {
        Identifier = identifier;
        Number = number;
    }
}
}
```

1.3 TreeNode

2.1.1 Вихідний код

```
using Classes;

namespace Lab5
{
internal class Program
{
    static async Task Main(string[] args)
{
    FileReader reader = new FileReader();

    var result = await reader.ReadFile("C:\\Users\\Дима\\Desktop\\KPI\\Теорія
Алгоритмів\\5\\teoriya-algoritmiv-5-main\\Lab5\\Files\\test1.cpp");

    foreach(var r in result.TreeTraversal())
    {
        Console.WriteLine(r.Item1 + " " + r.Item2);
    }
}
}
}
```

2.1.2 Приклади роботи

На рисунках 3.1 і 3.2 показані приклади роботи програми.

```cpp
int main(){
    int a = 1;
    int b = 1;
    int c = a + b + 1;

    std::cout << c << std::endl;

    unsigned short d = 1u;
    double e = 1.0;
    e = e * e * e;

    std::cout << e << std::endl;

    double f = e / 2;

    float g = (float)(f * e / 2.0);

    std::cout << aboba(100, 200) << std::endl;

    std::string str1 = "aboba";
    std::string str2 = "   aboba   ";

    return 0;
}

void aboba(int a, int b) {
    return a + b;
}
```

```
a 4
aboba 2
b 4
c 2
cout 3
d 1
double 2
e 8
endl 3
f 2
float 2
g 1
int 6
iostream 1
main 1
short 1
std 8
str1 1
str2 1
string 2
unsigned 1
utility 1
```

Рисунок 3.1 – тест 1 файлу

Прочие файлы

```cpp
int sum(int a, int b) {
    return a + b;
}

int subtract(int a, int b) {
    return a - b;
}

int multiply(int a, int b) {
    return a * b;
}

int divide(int a, int b) {
    return a / b;
}

int main(){
    int a = 100;
    int b = 200;

    int c = sum(a, b);

    int d = subtract(c, a);

    c = multiply(c, d);

    int e = divide(c, a);

    return 0;
}
```

```
a 12
b 10
c 5
d 2
divide 2
e 1
int 18
main 1
multiply 2
subtract 2
sum 2
```

Рисунок 3.2 – тест 2 файлу

Прочие файлы

```
1  class Aboba {
2  private:
3      int x = 0;
4      int y = 0;
5
6  public:
7      Aboba(int x, int y) {
8          this->x = x;
9          this->y = y;
10     }
11
12     int getSum() {
13         return x + y;
14     }
15 };
16
17 int main(){
18     Aboba a = new Aboba(100, 200);
19
20     std::cout << a.getSum() << std::endl;
21
22     return 0;
23 }
```

```
a 2
Aboba 4
cout 1
endl 1
getSum 2
int 6
main 1
std 2
this 2
x 5
y 5
```

Рисунок 3.3 – тест 3 файлу

ВИСНОВОК

При виконанні даної лабораторної роботи ми навчились працювати з деревовидними структурами даних. Застосували структуру даних “бінарне дерево” для пошуку всіх ідентифікаторів у файлі нашої програми написаної на мові програмування C++ . Також ми вивчили основні підходи формалізації та імплементації алгоритмів побудови та обробки базових деревовидних структур дан