Міністерство освіти і науки України

Національний технічний університет України

“Київський політехнічний інститут”

Кафедра АСОІУ

ЗВІТ

про виконання лабораторної роботи № 4

з дисципліни

“Дискретні структури”

Тема:Дерева.Бінарні дерева пошуку. АВЛ-дерева.

|  |  |  |
| --- | --- | --- |
| Прийняв: |  | Виконав: |
| [ст.вик. Москаленко Н. В.](http://rozklad.kpi.ua/Schedules/ViewSchedule.aspx?v=98f24a67-7c5b-4cd0-b696-d43e4af682ea) |  | студент 1-го курсу  гр. ІП-81 ФІОТ  Педоренко Микита Русланович |

Київ 2018

Зміст

[1 МЕТА РОБОТИ 3](#_Toc2507274)

[2 ПОСТАНОВКА ЗАДАЧІ 4](#_Toc2507275)

[3 БЛОК СХЕМА АЛГОРИТМУ 5](#_Toc2507276)

[4 ТЕКСТ ПРОГРАМНОГО КОДУ 6](#_Toc2507277)

[5 РЕЗУЛЬТАТИ РОБОТИ ПРОГРАМИ 7](#_Toc2507278)

[ВИСНОВКИ 8](#_Toc2507279)

[Додаток 9](#_Toc2507280)

1 МЕТА РОБОТИ

*Мета роботи* – опанувати теоретичні положення бінарних дерев пошуку та АВЛ-дерев як збалансованих дерев пошуку. Знати базові методи обходу бінарних дерев. Вміти релізувати АВЛ-дерево на алгоритмічній мові програмування.

1. ПОСТАНОВКА ЗАДАЧІ

Реалізувати на алгоритмічній мові програмування АВЛ-дерево пошуку з операціями вставки, пошуку та видалення елементів, а також префіксним, постфіксним та інфіксним обходами.

**Завдання**:  
Знайти ширину піддерева k-ї вершини.

3 БЛОК СХЕМА АЛГОРИТМУ



4 ТЕКСТ ПРОГРАМНОГО КОДУ

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace LAB4DS

{

class Program

{

static AVLTree EnterTree()

{

AVLTree tree = new AVLTree();

while (true)

{

Console.WriteLine("===================================================>GeneneratingTree<===================================================");

Console.WriteLine("1.Add 2.Remove 3.Stop");

Console.Write("Infix:"); tree.Infix(tree.top); Console.WriteLine();

Console.Write("Prefix:"); tree.Prefix(tree.top); Console.WriteLine();

Console.Write("Postfix:"); tree.Postfix(tree.top); Console.WriteLine();

Console.Write("Choose operation: ");

switch (Console.ReadLine())

{

case "1":

Console.Write("Add: ");

tree.Add(int.Parse(Console.ReadLine()));

break;

case "2":

Console.Write("Remove: ");

tree.Delete(int.Parse(Console.ReadLine()));

break;

case "3":

Console.Clear();

return tree;

default:

break;

}

Console.Clear();

}

}

static void Main(string[] args)

{

AVLTree tree = EnterTree();

Console.WriteLine("Tree top " + tree.top.data);

Console.Write("Infix:"); tree.Infix(tree.top); Console.WriteLine();

Console.Write("Prefix:"); tree.Prefix(tree.top); Console.WriteLine();

Console.Write("Postfix:"); tree.Postfix(tree.top); Console.WriteLine();

Console.Write("Enter Level: ");

Console.WriteLine("Width: "+tree.GetWidth(tree.top,int.Parse(Console.ReadLine())));

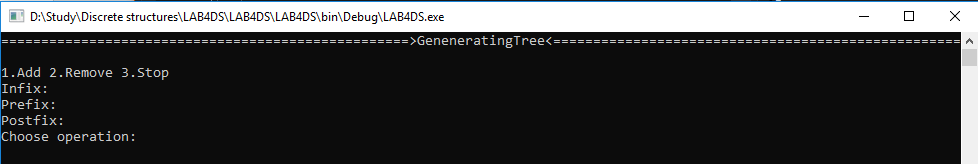
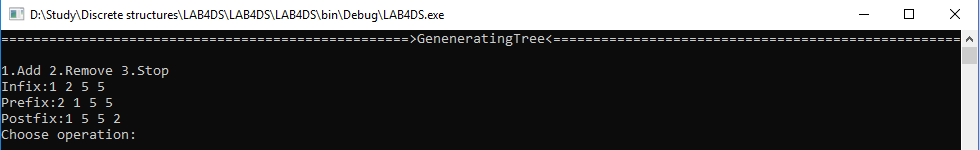
Console.ReadLine();

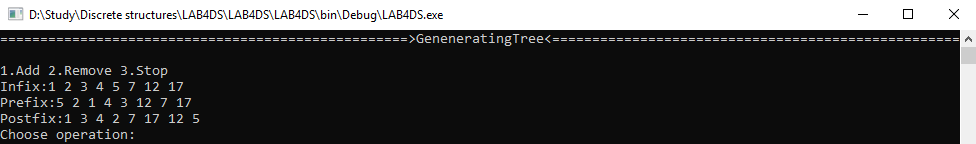
}

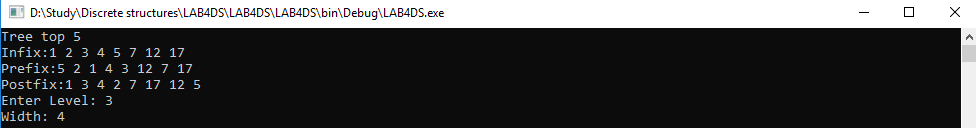
}

}

5 РЕЗУЛЬТАТИ РОБОТИ ПРОГРАМИ







ВИСНОВКИ

Під час виконання даної лабораторної роботи я опанував теоретичні положення бінарних дерев пошуку та АВЛ-дерев як збалансованих дерев пошуку. Вивчив базові методи обходу бінарних деревю. Реалізував АВЛ-дерево на алгоритмічній мові програмування.

# Додаток А. Код програми.

**Program.cs:**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace LAB4DS

{

class Program

{

static AVLTree EnterTree()

{

AVLTree tree = new AVLTree();

while (true)

{

Console.WriteLine("===================================================>GeneneratingTree<===================================================");

Console.WriteLine("1.Add 2.Remove 3.Stop");

Console.Write("Infix:"); tree.Infix(tree.top); Console.WriteLine();

Console.Write("Prefix:"); tree.Prefix(tree.top); Console.WriteLine();

Console.Write("Postfix:"); tree.Postfix(tree.top); Console.WriteLine();

Console.Write("Choose operation: ");

switch (Console.ReadLine())

{

case "1":

Console.Write("Add: ");

tree.Add(int.Parse(Console.ReadLine()));

break;

case "2":

Console.Write("Remove: ");

tree.Delete(int.Parse(Console.ReadLine()));

break;

case "3":

Console.Clear();

return tree;

default:

break;

}

Console.Clear();

}

}

static void Main(string[] args)

{

AVLTree tree = EnterTree();

Console.WriteLine("Tree top " + tree.top.data);

Console.Write("Infix:"); tree.Infix(tree.top); Console.WriteLine();

Console.Write("Prefix:"); tree.Prefix(tree.top); Console.WriteLine();

Console.Write("Postfix:"); tree.Postfix(tree.top); Console.WriteLine();

Console.Write("Enter Level: ");

Console.WriteLine("Width: "+tree.GetWidth(tree.top,int.Parse(Console.ReadLine())));

Console.ReadLine();

}

}

}

**AVLTree.cs:**

namespace LAB4DS

{

class AVLTree

{

public class Node

{

public int data;

public int height;

public Node left, right;

public Node(int data = 0)

{

this.data = data;

this.height = 1;

}

}

public Node top;

public int Heigth(Node element)

{

if (element != null) return element.height;

return 0;

}

public int BalanceFactor(Node element)

{

return Heigth(element.right) - Heigth(element.left);

}

public void CalcHeight(Node element)

{

int hLeft = Heigth(element.left);

int hRight = Heigth(element.right);

element.height = (hLeft > hRight ? hLeft : hRight) + 1;

}

public Node leftRotare(Node q)

{

Node p = q.right;

q.right = p.left;

p.left = q;

CalcHeight(q);

CalcHeight(p);

return p;

}

public Node rightRotate(Node p)

{

Node q = p.left;

p.left = q.right;

q.right = p;

CalcHeight(p);

CalcHeight(q);

return q;

}

public Node Balancing(Node element)

{

CalcHeight(element);

if (BalanceFactor(element) == 2)

{

if (BalanceFactor(element.right) < 0)

element.right = rightRotate(element.right);

return leftRotare(element);

}

else if (BalanceFactor(element) == -2)

{

if (BalanceFactor(element.left) > 0)

element.left = leftRotare(element.left);

return rightRotate(element);

}

return element;

}

public Node Insert(Node element, int data)

{

if (element == null)

return new Node(data);

if (data < element.data)

element.left = Insert(element.left, data);

else

element.right = Insert(element.right, data);

return Balancing(element);

}

public void Add(int data)

{

top = Insert(top, data);

}

public Node FindLeft(Node element)

{

if (element.left != null) return FindLeft(element.left);

return element;

}

public Node RemoveMin(Node element)

{

if (element.left == null)

return element.right;

element.left = RemoveMin(element.left);

return Balancing(element);

}

public Node Remove(Node element, int data)

{

if (element == null) return null;

if (data < element.data)

element.left = Remove(element.left, data);

else if (data > element.data)

element.right = Remove(element.right, data);

else

{

Node q = element.left;

Node r = element.right;

if (r == null) return q;

Node min = FindLeft(r);

min.left = RemoveMin(r);

min.left = q;

return Balancing(min);

}

return Balancing(element);

}

public void Delete(int data)

{

top = Remove(top, data);

}

public void Infix(Node top)

{

if (top == null) return;

Infix(top.left);

System.Console.Write(top.data + " ");

Infix(top.right);

}

public void Prefix(Node top)

{

if (top == null) return;

System.Console.Write(top.data + " ");

Prefix(top.left);

Prefix(top.right);

}

public void Postfix(Node top)

{

if (top == null) return;

Postfix(top.left);

Postfix(top.right);

System.Console.Write(top.data + " ");

}

public bool Find(int data)

{

Node p = top;

while (p != null)

{

if (data < p.data) p = p.left;

if (data > p.data) p = p.right;

if (p.data == data) return true;

}

return false;

}

public int GetWidth(Node top, int level)

{

if (top == null) return 0;

if (level == 1) return 1;

else if (level > 1)

return GetWidth(top.left, level - 1) + GetWidth(top.right, level - 1);

return 0;

}

}

}