**Міністерство освіти і науки України**

**Національний технічний університет України «Київський політехнічний інститут імені Ігоря Сікорського"**

**Факультет інформатики та обчислювальної техніки**

**Кафедра ІПІ**

**Звіт**

з лабораторної роботи № 6 з дисципліни

«Основи програмування 2.

Модульне програмування»

Варіант 25

**Виконав(ла)**

(шифр, прізвище, ім'я, по батькові)

*ІП-15 Плугатирьов Дмитро Валерійович ійович*

**Перевірив**

(прізвище, ім'я, по батькові)

*Вєчерковська Анастасія Сергіївна*

Київ 2022

**ДЕРЕВА**

*Мета роботи* – вивчити особливості організації та обробки дерев.

**Варіант 25**

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

**Код програми**

**C#**

namespace Labwork\_6

{

    public class BinaryTree

    {

        public Node Root { get; set; }

        public bool Add(ProductModel product)

        {

            Node previous = null;

            Node next = Root;

            while (next != null)

            {

                previous = next;

                if (product.Price < next.Data.Price)

                {

                    next = next.LeftNode;

                }

                else if (product.Price > next.Data.Price)

                {

                    next = next.RightNode;

                }

                else

                {

                    return false;

                }

            }

            Node newNode = new Node(product);

            if (this.Root == null)

            {

                this.Root = newNode;

            }

            else

            {

                if (product.Price < previous.Data.Price)

                {

                    previous.LeftNode = newNode;

                }

                else

                {

                    previous.RightNode = newNode;

                }

            }

            return true;

        }

        public Node Find(decimal price)

        {

            return this.Find(price, this.Root);

        }

        public void Remove(int value)

        {

            this.Root = Remove(this.Root, value);

        }

        private Node Remove(Node parent, decimal price)

        {

            if (parent == null)

            {

                return parent;

            }

            if (price < parent.Data.Price)

            {

                parent.LeftNode = Remove(parent.LeftNode, price);

            }

            else if (price > parent.Data.Price)

            {

                parent.RightNode = Remove(parent.RightNode, price);

            }

            else

            {

                if (parent.LeftNode == null)

                {

                    return parent.RightNode;

                }

                else if (parent.RightNode == null)

                {

                    return parent.LeftNode;

                }

                parent.Data.Price = GetMinValue(parent.RightNode);

                parent.RightNode = Remove(parent.RightNode, parent.Data.Price);

            }

            return parent;

        }

        private decimal GetMinValue(Node node)

        {

            decimal minPrice = node.Data.Price;

            while (node.LeftNode != null)

            {

                minPrice = node.LeftNode.Data.Price;

                node = node.LeftNode;

            }

            return minPrice;

        }

        private Node Find(decimal price, Node parent)

        {

            if (parent != null)

            {

                if (price == parent.Data.Price)

                {

                    return parent;

                }

                else if (price < parent.Data.Price)

                {

                    return Find(price, parent.LeftNode);

                }

                else

                {

                    return Find(price, parent.RightNode);

                }

            }

            return null;

        }

        public int GetTreeDepth()

        {

            return GetTreeDepth(Root);

        }

        private int GetTreeDepth(Node parent)

        {

            return (parent == null) ? 0 : Math.Max(GetTreeDepth(parent.LeftNode),

                GetTreeDepth(parent.RightNode)) + 1;

        }

        public decimal GetTotalPrice(Node parent)

        {

            decimal totalPrice = parent.Data.GetTotalPrice();

            if (parent.LeftNode != null)

            {

                totalPrice += GetTotalPrice(parent.LeftNode);

            }

            if (parent.RightNode != null)

            {

                totalPrice += GetTotalPrice(parent.RightNode);

            }

            return totalPrice;

        }

        public void TraversePreOrder(Node parent)

        {

            if (parent != null)

            {

                Console.WriteLine(parent.Data);

                TraversePreOrder(parent.LeftNode);

                TraversePreOrder(parent.RightNode);

            }

        }

        public void TraverseInOrder(Node parent)

        {

            if (parent != null)

            {

                TraverseInOrder(parent.LeftNode);

                Console.WriteLine(parent.Data);

                TraverseInOrder(parent.RightNode);

            }

        }

        public void TraversePostOrder(Node parent)

        {

            if (parent != null)

            {

                TraversePostOrder(parent.LeftNode);

                TraversePostOrder(parent.RightNode);

                Console.WriteLine(parent.Data);

            }

        }

    }

}

namespace Labwork\_6

{

    public class Node

    {

        public Node LeftNode { get; set; }

        public Node RightNode { get; set; }

        public ProductModel Data { get; set; }

        public Node(ProductModel product)

        {

            Data = product;

        }

        public Node()

        {

        }

    }

}

namespace Labwork\_6

{

    public class ProductModel

    {

        private string \_name;

        public string Name

        {

            get => \_name;

            set

            {

                if (value == string.Empty)

                {

                    throw new ArgumentException("The name of product mustn't be empty");

                }

                \_name = value;

            }

        }

        private int \_count;

        public int Count

        {

            get => \_count;

            set

            {

                if (value < 0)

                {

                    throw new ArgumentOutOfRangeException(nameof(value),

                        "The count mustn't be less than 0");

                }

                \_count = value;

            }

        }

        private decimal \_price;

        public decimal Price

        {

            get => \_price;

            set

            {

                if (value < 0)

                {

                    throw new ArgumentOutOfRangeException(nameof(value),

                        "The price mustn't be less than 0");

                }

                \_price = value;

            }

        }

        public ProductModel(string name, decimal price, int count = 0)

        {

            Name = name;

            Count = count;

            Price = price;

        }

        public ProductModel()

        {

        }

        public override string ToString()

        {

            return $"Name - {Name}, Price - {Price}, Count - {Count}";

        }

        public decimal GetTotalPrice()

        {

            return Price \* Count;

        }

    }

}

namespace Labwork\_6

{

    class Program

    {

        public static void Main(string[] args)

        {

            BinaryTree binaryTree = new BinaryTree();

            CaptureProducts(binaryTree);

            System.Console.WriteLine($"The depth of tree: {binaryTree.GetTreeDepth()}");

            Console.WriteLine("PreOrder traversal:");

            binaryTree.TraversePreOrder(binaryTree.Root);

            PrintHorizontalRule();

            Console.WriteLine("InOrder traversal:");

            binaryTree.TraverseInOrder(binaryTree.Root);

            PrintHorizontalRule();

            Console.WriteLine("PostOrder traversal:");

            binaryTree.TraversePostOrder(binaryTree.Root);

            PrintHorizontalRule();

            System.Console.Write("Total price: ");

            System.Console.WriteLine(binaryTree.GetTotalPrice(binaryTree.Root));

        }

        static void PrintHorizontalRule()

        {

            System.Console.WriteLine(new string('-', 80));

        }

        public static void CaptureProducts(BinaryTree binaryTree)

        {

            bool exceptionIsCaught = true;

            do

            {

                ProductModel product = new();

                exceptionIsCaught = false;

                try

                {

                    System.Console.Write("Please, enter the product name: ");

                    product.Name = Console.ReadLine();

                    System.Console.Write("Please, enter the product price: ");

                    product.Price = int.Parse(Console.ReadLine());

                    System.Console.Write("Please, eter the product count: ");

                    product.Count = int.Parse(Console.ReadLine());

                    binaryTree.Add(product);

                    PrintHorizontalRule();

                }

                catch (ArgumentException ex)

                {

                    System.Console.WriteLine(ex.Message);

                    exceptionIsCaught = true;

                }

                catch (FormatException ex)

                {

                    System.Console.WriteLine(ex.Message);

                    exceptionIsCaught = true;

                }

                if (!exceptionIsCaught)

                {

                    System.Console.WriteLine("Press <Backspace> if you want to end typing or any key to continue");

                }

            } while (exceptionIsCaught || Console.ReadKey().Key != ConsoleKey.Backspace);

        }

    }

}

**Висновок**

На цій лабораторній роботі я побудував дерево та реалізував прохід по ньому кількома способами, видалення і вставку вузла. Я зрозумів, що дерева використовуються для представлення або маніпуляції ієрархічних даних.