**Доклад**

**по**

**Алгоритми и структури от данни**

**Тема: Дървовидни структури от данни**

1. **Дървовидни структури от данни. Увод в теорията на графите. Съпоставяне на дърво и граф**
2. **Реализация на двоично дърво**
3. **Обхождане на двоично дърво**
4. **Балансиране на двоично дърво за претърсване**
5. **Източници**

Изготвил: Ваня Ванева 12а

1. **Дървовидни структури от данни. Увод в теорията на графите. Съпоставяне на дърво и граф**

Дървовидните структури от данни представляват начин за организация на данни във вид на дърво, където всеки елемент има родител и деца, образувайки йерархия. В теорията на графите, графът е структура, която представлява съвкупност от върхове и ребра, които ги свързват.

Дърво:

• Всеки елемент има само един родител.

• Няма цикли (затворени пътища) в структурата.

• Подходящи за организация на йерархична информация.

Граф:

• Върховете могат да имат множество родители и деца.

• Възможни са цикли в структурата.

• Приложими за моделиране на различни взаимоотношения и връзки.

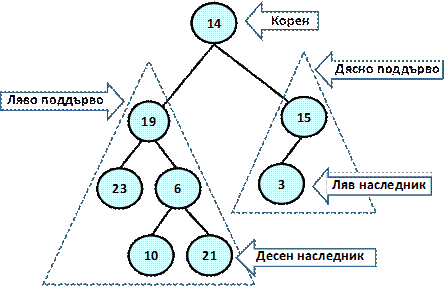
Съпоставянето:

• Корен на дърво ≈ Начален връх на граф.

• Ребрата в дървото ≈ Връзките между върховете в графа.

• Листа в дървото ≈ Върхове с нула деца в графа.

• Височина на дървото ≈ Дължина на най-дългия път в графа.



1. **Реализация на двоично дърво**

**using System;**

**public class BinaryTreeNode<T>**

**{**

**public T Data { get; set; }**

**public BinaryTreeNode<T> Left { get; set; }**

**public BinaryTreeNode<T> Right { get; set; }**

**public BinaryTreeNode(T data)**

**{**

**Data = data;**

**Left = null;**

**Right = null;**

**}**

**}**

**public class BinaryTree<T>**

**{**

**public BinaryTreeNode<T> Root { get; set; }**

**public BinaryTree()**

**{**

**Root = null;**

**}**

**// Метод за добавяне на възел в дървото**

**public void Add(T data)**

**{**

**Root = AddRecursive(Root, data);**

**}**

**private BinaryTreeNode<T> AddRecursive(BinaryTreeNode<T> current, T data)**

**{**

**if (current == null)**

**{**

**return new BinaryTreeNode<T>(data);**

**}**

**// Рекурсивно добавяне на възела в подходящата поддървовидна част**

**if (Comparer<T>.Default.Compare(data, current.Data) < 0)**

**{**

**current.Left = AddRecursive(current.Left, data);**

**}**

**else if (Comparer<T>.Default.Compare(data, current.Data) > 0)**

**{**

**current.Right = AddRecursive(current.Right, data);**

**}**

**return current;**

**}**

**// Метод за обхождане на дървото в инордер (ляво-корен-дясно)**

**public void InOrderTraversal(BinaryTreeNode<T> node)**

**{**

**if (node != null)**

**{**

**InOrderTraversal(node.Left);**

**Console.WriteLine(node.Data);**

**InOrderTraversal(node.Right);**

**}**

**}**

**}**

**class Program**

**{**

**static void Main(string[] args)**

**{**

**BinaryTree<int> binaryTree = new BinaryTree<int>();**

**binaryTree.Add(5);**

**binaryTree.Add(3);**

**binaryTree.Add(7);**

**binaryTree.Add(1);**

**binaryTree.Add(4);**

**binaryTree.Add(6);**

**binaryTree.Add(9);**

**binaryTree.InOrderTraversal(binaryTree.Root);**

**}**

**}**

Този пример дефинира два класа: BinaryTreeNode<T> за възлите на двоичното дърво и BinaryTree<T> за самото дърво. Класът BinaryTreeNode<T> съдържа данни и референции към лявото и дясното дете на възела. Класът BinaryTree<T> има свойство Root и методи за добавяне на възел и обхождане на дървото в инордер. Методът за добавяне е рекурсивен и добавя възлите според тяхната стойност (по-малките в лявото поддърво, по-големите в дясното).

1. **Обхождане на двоично дърво**

В двоичното дърво се извършват няколко вида обхождания: инордер (ляво-корен-дясно), предордер (корен-ляво-дясно) и постордер (ляво-дясно-корен). Ето как може да се реализират тези обхождания в C#:

**using System;**

**public class BinaryTreeNode<T>**

**{**

**public T Data { get; set; }**

**public BinaryTreeNode<T> Left { get; set; }**

**public BinaryTreeNode<T> Right { get; set; }**

**public BinaryTreeNode(T data)**

**{**

**Data = data;**

**Left = null;**

**Right = null;**

**}**

**}**

**public class BinaryTree<T>**

**{**

**public BinaryTreeNode<T> Root { get; set; }**

**public BinaryTree()**

**{**

**Root = null;**

**}**

**// Инордер обхождане: ляво-корен-дясно**

**public void InOrderTraversal(BinaryTreeNode<T> node)**

**{**

**if (node != null)**

**{**

**InOrderTraversal(node.Left);**

**Console.WriteLine(node.Data);**

**InOrderTraversal(node.Right);**

**}**

**}**

**// Предордер обхождане: корен-ляво-дясно**

**public void PreOrderTraversal(BinaryTreeNode<T> node)**

**{**

**if (node != null)**

**{**

**Console.WriteLine(node.Data);**

**PreOrderTraversal(node.Left);**

**PreOrderTraversal(node.Right);**

**}**

**}**

**// Постордер обхождане: ляво-дясно-корен**

**public void PostOrderTraversal(BinaryTreeNode<T> node)**

**{**

**if (node != null)**

**{**

**PostOrderTraversal(node.Left);**

**PostOrderTraversal(node.Right);**

**Console.WriteLine(node.Data);**

**}**

**}**

**}**

**class Program**

**{**

**static void Main(string[] args)**

**{**

**BinaryTree<int> binaryTree = new BinaryTree<int>();**

**binaryTree.Root = new BinaryTreeNode<int>(1);**

**binaryTree.Root.Left = new BinaryTreeNode<int>(2);**

**binaryTree.Root.Right = new BinaryTreeNode<int>(3);**

**binaryTree.Root.Left.Left = new BinaryTreeNode<int>(4);**

**binaryTree.Root.Left.Right = new BinaryTreeNode<int>(5);**

**Console.WriteLine("InOrder traversal:");**

**binaryTree.InOrderTraversal(binaryTree.Root);**

**Console.WriteLine("PreOrder traversal:");**

**binaryTree.PreOrderTraversal(binaryTree.Root);**

**Console.WriteLine("PostOrder traversal:");**

**binaryTree.PostOrderTraversal(binaryTree.Root);**

**}**

**}**

Този код дефинира двоично дърво и реализира методи за инордер, предордер и постордер обхождане. Когато се извика всяко от тези методи с подаден корен на дървото, те ще обходят дървото в съответния ред и ще изведат данните на възлите на конзолата.

Балансирането на двоично дърво за претърсване (BST) е процес, при който се оптимизира структурата на дървото, за да се гарантира, че височините на двете поддървета на всеки възел са близки до еднакви. Това улеснява бързото търсене в дървото и осигурява по-балансирано разпределение на данните.

1. **Балансиране на двоично дърво за претърсване**

Един от най-популярните методи за балансиране на BST е методът на въртенето. Този метод се използва за преместване на възлите във високите слоеве на дървото, за да се постигне баланс. Ето примерна имплементация на метода за въртене:

**using System;**

**public class BinaryTreeNode<T>**

**{**

**public T Data { get; set; }**

**public BinaryTreeNode<T> Left { get; set; }**

**public BinaryTreeNode<T> Right { get; set; }**

**public BinaryTreeNode(T data)**

**{**

**Data = data;**

**Left = null;**

**Right = null;**

**}**

**}**

**public class BinarySearchTree<T>**

**{**

**public BinaryTreeNode<T> Root { get; set; }**

**public BinarySearchTree()**

**{**

**Root = null;**

**}**

**// Вмъкване на елемент в дървото**

**public void Insert(T data)**

**{**

**Root = InsertRecursive(Root, data);**

**}**

**private BinaryTreeNode<T> InsertRecursive(BinaryTreeNode<T> node, T data)**

**{**

**if (node == null)**

**{**

**return new BinaryTreeNode<T>(data);**

**}**

**if (Comparer<T>.Default.Compare(data, node.Data) < 0)**

**{**

**node.Left = InsertRecursive(node.Left, data);**

**}**

**else if (Comparer<T>.Default.Compare(data, node.Data) > 0)**

**{**

**node.Right = InsertRecursive(node.Right, data);**

**}**

**return node;**

**}**

**// Ротация на дървото надляво**

**private BinaryTreeNode<T> RotateLeft(BinaryTreeNode<T> node)**

**{**

**BinaryTreeNode<T> newRoot = node.Right;**

**node.Right = newRoot.Left;**

**newRoot.Left = node;**

**return newRoot;**

**}**

**// Ротация на дървото надясно**

**private BinaryTreeNode<T> RotateRight(BinaryTreeNode<T> node)**

**{**

**BinaryTreeNode<T> newRoot = node.Left;**

**node.Left = newRoot.Right;**

**newRoot.Right = node;**

**return newRoot;**

**}**

**// Балансиране на дървото**

**public BinaryTreeNode<T> Balance(BinaryTreeNode<T> node)**

**{**

**if (node == null)**

**{**

**return null;**

**}**

**// Балансиране на лявото поддърво**

**if (GetHeight(node.Left) - GetHeight(node.Right) > 1)**

**{**

**if (GetHeight(node.Left.Left) >= GetHeight(node.Left.Right))**

**{**

**node = RotateRight(node);**

**}**

**else**

**{**

**node.Left = RotateLeft(node.Left);**

**node = RotateRight(node);**

**}**

**}**

**// Балансиране на дясното поддърво**

**else if (GetHeight(node.Right) - GetHeight(node.Left) > 1)**

**{**

**if (GetHeight(node.Right.Right) >= GetHeight(node.Right.Left))**

**{**

**node = RotateLeft(node);**

**}**

**else**

**{**

**node.Right = RotateRight(node.Right);**

**node = RotateLeft(node);**

**}**

**}**

**return node;**

**}**

**// Връща височината на дървото**

**private int GetHeight(BinaryTreeNode<T> node)**

**{**

**if (node == null)**

**{**

**return 0;**

**}**

**return 1 + Math.Max(GetHeight(node.Left), GetHeight(node.Right));**

**}**

**// Извеждане на дървото**

**public void PrintInOrder(BinaryTreeNode<T> node)**

**{**

**if (node != null)**

**{**

**PrintInOrder(node.Left);**

**Console.Write(node.Data + " ");**

**PrintInOrder(node.Right);**

**}**

**}**

**}**

**class Program**

**{**

**static void Main(string[] args)**

**{**

**BinarySearchTree<int> bst = new BinarySearchTree<int>();**

**bst.Insert(5);**

**bst.Insert(3);**

**bst.Insert(7);**

**bst.Insert(2);**

**bst.Insert(4);**

**bst.Insert(6);**

**bst.Insert(8);**

**Console.WriteLine("Before balancing:");**

**bst.PrintInOrder(bst.Root);**

**Console.WriteLine();**

**bst.Root = bst.Balance(bst.Root);**

**Console.WriteLine("After balancing:");**

**bst.PrintInOrder(bst.Root);**

**}**

**}**

Тази имплементация на двоично дърво за претърсване включва методи за вмъкване на елементи в дървото, изчисляване на височината на възлите, методи за ротация на дървото наляво и надясно и метод за балансиране на дървото.

1. **Източници**

<https://introprogramming.info/english-intro-csharp-book/read-online/chapter-17-trees-and-graphs/>

<https://www.geeksforgeeks.org/types-of-trees-in-data-structures/>

<https://tutorials.eu/binary-trees-in-c-sharp/>

<https://www.c-sharpcorner.com/UploadFile/19b1bd/binarysearchtreewalkstraversal-implementation-using-C-Sharp/>

<https://stackoverflow.com/questions/2893318/building-a-balanced-binary-search-tree>

<https://www.geeksforgeeks.org/convert-normal-bst-balanced-bst/>