Pojęcie drzewa w strukturach danych

1. Definicja drzewa:

- Drzewo to struktura danych składająca się z węzłów połączonych krawędziami. Jest ona jednym z podstawowych elementów w informatyce, wykorzystywanym do przechowywania danych i wykonywania różnych operacji.

2. Podstawowe terminy:

- Korzeń: Węzeł nadrzędny, od którego rozchodzą się pozostałe węzły drzewa.

- Węzły: Elementy składowe drzewa, mogące mieć zero lub więcej dzieci.

- Liście: Węzły, które nie posiadają dzieci, stanowiące końcowe elementy w strukturze drzewa.

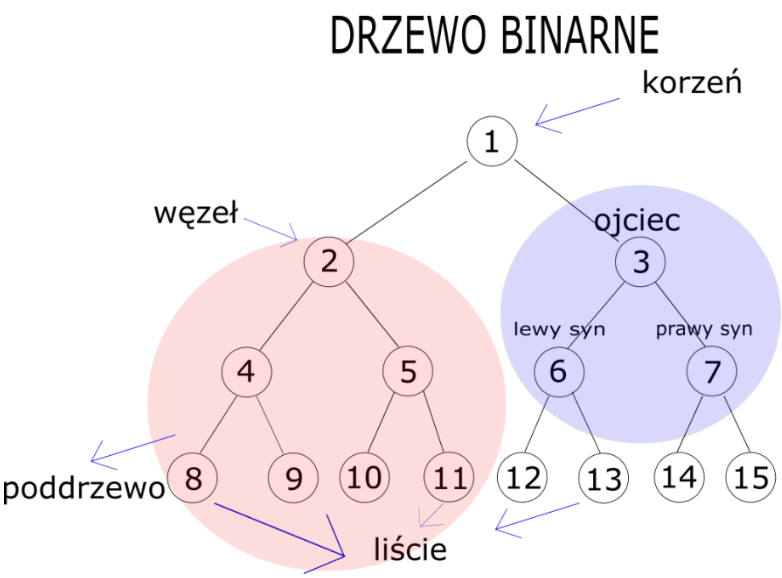
- Krawędzie: Relacje między węzłami, łączące węzeł z jego dziećmi.

- Głębokość: Odległość między danym węzłem a korzeniem.

- Wysokość: Maksymalna głębokość w drzewie, czyli odległość między korzeniem a liściem.

- Poziom: Liczba krawędzi między węzłem a korzeniem, plus jeden.

- Stopień węzła: Liczba dzieci danego węzła.



https://www.algorytm.edu.pl/struktury-danych/drzewo-binarne

3. Różne rodzaje drzew:

- Drzewo binarne: Każdy węzeł ma maksymalnie dwóch potomków.

- Drzewo BST (Binary Search Tree): Drzewo binarne, w którym dla każdego węzła lewe poddrzewo zawiera tylko klucze mniejsze od klucza węzła, a prawe poddrzewo zawiera tylko klucze większe.

- Drzewo AVL: Zrównoważone drzewo binarne, w którym różnica wysokości poddrzew dla każdego węzła wynosi co najwyżej jeden.

- Drzewo wyrażeń arytmetycznych: Drzewo binarne reprezentujące strukturę wyrażenia arytmetycznego, np. (a + b) * (c - d).

- Drzewo czerwono-czarne: Rodzaj samobalansującego się drzewa binarnego, w którym każdy węzeł ma kolor czerwony lub czarny, a spełnione są pewne reguły dotyczące koloru węzłów.

- Drzewo B: Drzewo binarne, w którym każdy węzeł może mieć więcej niż dwa dzieci, a klucze są przechowywane w węzłach wewnętrznych, a nie tylko w liściach.

- Drzewo Trie: Struktura danych, która reprezentuje zbiór danych tekstowych w formie drzewa, w którym każda krawędź odpowiada pojedynczemu znakowi.

- Drzewo Huffmana: Drzewo binarne używane w algorytmie Huffmana do kompresji danych, w którym częściej występujące symbole są bliżej korzenia, co pozwala na uzyskanie krótszych kodów dla częściej używanych symboli.

- Drzewo Merkle'a: Drzewo używane w kryptografii, w którym każdy węzeł jest haszem kryptograficznym swoich dzieci, co umożliwia weryfikację integralności danych.

- Drzewo wyrażeń: Drzewo, które reprezentuje składnię wyrażenia matematycznego, gdzie liście reprezentują wartości, a węzły reprezentują operatory.

- Drzewo Sufiksowe: Struktura danych używana w przetwarzaniu tekstu do przechowywania wszystkich sufiksów tekstu w formie drzewa, które pozwala na efektywne wyszukiwanie wzorców.

Drzewa binarne

Drzewa binarne są strukturą danych, która składa się z węzłów, z których każdy może mieć co najwyżej dwóch potomków: lewego i prawego. Drzewo rozpoczyna się od korzenia, a węzły tworzą gałęzie, które mogą prowadzić do kolejnych węzłów lub liści. Liście są węzłami końcowymi, które nie posiadają potomków.

Drzewa binarne są wykorzystywane w różnych dziedzinach informatyki, takich jak:

- Przechowywanie danych w hierarchicznej strukturze.

- Wyszukiwanie danych w efektywny sposób.

- Analiza i przetwarzanie danych w algorytmach.

Implementacja drzew binarnych w C++

Biblioteka STL udostępnia gotowe narzędzia do implementacji drzew binarnych w języku C++, takie jak std::set i std::map.

Kontenery std::set i std::map z biblioteki standardowej C++:

std::set:

https://en.cppreference.com/w/cpp/container/set

- std::set jest kontenerem asocjacyjnym, który przechowuje unikalne, uporządkowane elementy.

- Każdy element w std::set występuje tylko raz, co oznacza, że nie ma powtórzeń.

- Elementy w std::set są automatycznie sortowane, co oznacza, że są przechowywane w porządku rosnącym.

- Struktura danych wewnętrznie jest implementowana jako drzewo poszukiwań binarnych lub równoważne struktury danych, co zapewnia wysoką efektywność wyszukiwania, dodawania i usuwania elementów.

std::map:

https://en.cppreference.com/w/cpp/container/map

- std::map jest kontenerem asocjacyjnym, który przechowuje pary klucz-wartość, uporządkowane według klucza.

- Każdy klucz w std::map jest unikalny, a wartość może być dowolna.

- Elementy w std::map są automatycznie sortowane według klucza, co pozwala na efektywne wyszukiwanie, dodawanie i usuwanie elementów na podstawie klucza.

- Struktura danych wewnętrznie jest implementowana jako drzewo poszukiwań binarnych lub równoważne struktury danych, co zapewnia wysoką efektywność operacji na mapie.

Porównanie:

- Zarówno std::set, jak i std::map opierają się na drzewach poszukiwań binarnych lub podobnych strukturach danych, co zapewnia efektywne wyszukiwanie, dodawanie i usuwanie elementów.

- Główna różnica między nimi polega na tym, że std::set przechowuje tylko klucze, podczas gdy std::map przechowuje pary klucz-wartość.

- Jeśli potrzebujesz przechowywać tylko unikalne wartości, std::set jest odpowiednią opcją.

- Jeśli potrzebujesz mapować klucze na wartości, std::map jest odpowiednią opcją.

W obu przypadkach, std::set i std::map są przydatnymi narzędziami do pracy z danymi, które wymagają unikalności i/lub mapowania kluczy na wartości.

Implementacja drzewa binarnego za pomocą std::set:

std::set jest kontenerem asocjacyjnym, który przechowuje unikalne, uporządkowane elementy.

```
#include <iostream>
#include <set>

int main() {
    std::set<int> binaryTree;

    // Dodawanie elementów do drzewa binarnego
    binaryTree.insert(5);
```

```
    binaryTree.insert(3);
    binaryTree.insert(7);
    binaryTree.insert(1);
    binaryTree.insert(4);

    // Wyświetlanie drzewa binarnego
    for (int num : binaryTree) {
        std::cout << num << " ";
    }
    std::cout << std::endl;

    return 0;
}
```

Implementacja drzewa binarnego za pomocą std::map:

std::map jest kontenerem asocjacyjnym, który przechowuje pary klucz-wartość, uporządkowane według klucza.

```
#include <iostream>
#include <map>

int main() {
    std::map<int, bool> binaryTree;

    // Dodawanie elementów do drzewa binarnego
    binaryTree[5] = true;
    binaryTree[3] = true;
    binaryTree[7] = true;
    binaryTree[1] = true;
    binaryTree[4] = true;

    // Wyświetlanie drzewa binarnego
    for (const auto& pair : binaryTree) {
        std::cout << pair.first << " ";
    }
    std::cout << std::endl;

    return 0;
}
```

W obu przypadkach, std::set i std::map automatycznie zajmują się zachowaniem uporządkowania elementów, co odpowiada właściwościom drzewa binarnego. Jednakże, warto zauważyć, że te implementacje nie pozwalają na takie działania jak usuwanie konkretnych węzłów, a jedynie dodawanie i sprawdzanie obecności elementów. Dla bardziej zaawansowanych operacji na drzewie binarnym, konieczne może być użycie niestandardowych struktur danych lub własnych implementacji drzewa binarnego.

Implementacja obiektowa drzewa binarnego

```
#include <iostream>

// Struktura węzła drzewa binarnego
struct TreeNode {
    int data;
```

```
    TreeNode* left;
    TreeNode* right;

    TreeNode(int value) : data(value), left(nullptr), right(nullptr) {}
};

// Klasa drzewa binarnego
class BinaryTree {
private:
    TreeNode* root;

public:
    BinaryTree() : root(nullptr) {}

    // Metoda do dodawania węzła do drzewa
    void insert(int value) {
        root = insertRecursive(root, value);
    }

    // Metoda rekurencyjna do dodawania węzła
    TreeNode* insertRecursive(TreeNode* node, int value) {
        if (node == nullptr) {
            return new TreeNode(value);
        }

        if (value < node->data) {
            node->left = insertRecursive(node->left, value);
        } else if (value > node->data) {
            node->right = insertRecursive(node->right, value);
        }

        return node;
    }

    // Metoda do wyświetlania drzewa w porządku inorder
    void inorderTraversal(TreeNode* node) {
        if (node != nullptr) {
            inorderTraversal(node->left);
            std::cout << node->data << " ";
            inorderTraversal(node->right);
        }
    }

    // Metoda publiczna do wyświetlania drzewa w porządku inorder
    void inorderTraversal() {
        inorderTraversal(root);
    }
};

int main() {
    BinaryTree tree;

    // Dodawanie elementów do drzewa
    tree.insert(50);
    tree.insert(30);
    tree.insert(70);
    tree.insert(20);
```

```
    tree.insert(40);

    // Wyświetlanie drzewa w porządku inorder
    std::cout << "Drzewo binarne w porządku inorder: ";
    tree.inorderTraversal();
    std::cout << std::endl;

    return 0;
}
```

Metody przeszukiwania drzew binarnych

Trzy główne metody przeszukiwania drzew binarnych: PREORDER, INORDER i POSTORDER. Każda z tych metod różni się kolejnością, w jakiej odwiedzane są węzły drzewa.

1. PREORDER (Przeszukiwanie w porządku przedrostkowym):

- Metoda PREORDER polega na odwiedzeniu najpierw korzenia drzewa, a następnie rekurencyjnym przeszukiwaniu lewego i prawego poddrzewa.

- Kolejność odwiedzania węzłów w metodzie PREORDER jest następująca: najpierw korzeń, potem lewe poddrzewo, a na końcu prawe poddrzewo.

2. INORDER (Przeszukiwanie w porządku środkowym):

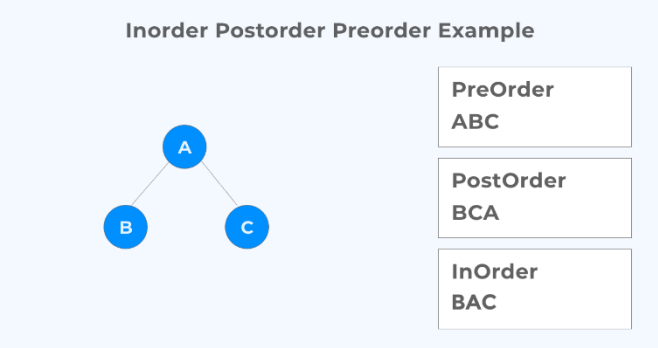
- Metoda INORDER polega na rekurencyjnym przeszukiwaniu lewego poddrzewa, odwiedzeniu korzenia, a następnie rekurencyjnym przeszukiwaniu prawego poddrzewa.

- Kolejność odwiedzania węzłów w metodzie INORDER jest następująca: najpierw lewe poddrzewo, potem korzeń, a na końcu prawe poddrzewo.

3. POSTORDER (Przeszukiwanie w porządku końcowym):

- Metoda POSTORDER polega na rekurencyjnym przeszukiwaniu lewego i prawego poddrzewa, a następnie odwiedzeniu korzenia.

- Kolejność odwiedzania węzłów w metodzie POSTORDER jest następująca: najpierw lewe poddrzewo, potem prawe poddrzewo, a na końcu korzeń.



https://prepinsta.com/data-structures-algorithms/inorder-postorder-preorder-examples/

Zadania:

Zadanie 1.

Z zapisanego poniżej ciągu liczb utwórz drzewo BST (drzewo poszukiwań binarnych).

2, 1, 30, 10, 5, 4, 11, 18, 7, 40

Nie pisz algorytmu tylko wynik przedstaw graficznie.

Aby utworzyć drzewo BST na podstawie podanego ciągu liczb, wykonuje się następujące kroki:

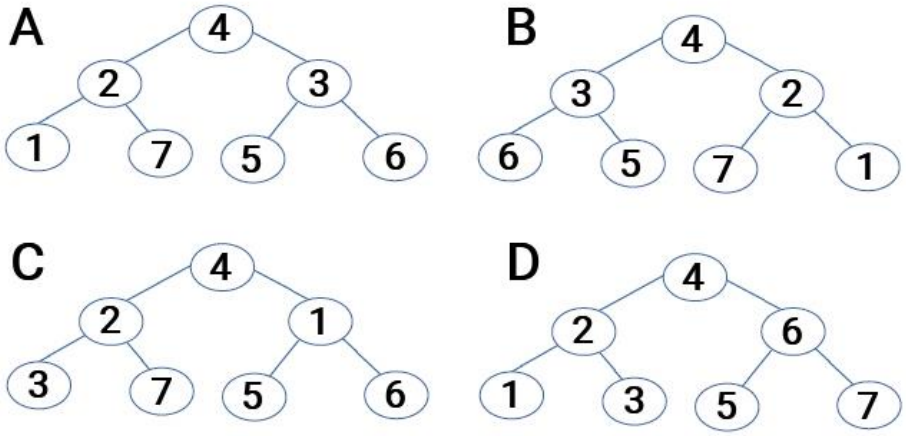
- Pierwsza liczba w ciągu staje się korzeniem drzewa.
- Kolejne liczby są dodawane do drzewa poprzez porównanie ich z wartościami już istniejących węzłów:
- Jeśli liczba jest mniejsza niż wartość węzła, to zostaje wstawiona do lewego poddrzewa.
- Jeśli liczba jest większa niż wartość węzła, to zostaje wstawiona do prawego poddrzewa.

Proces dodawania liczby do drzewa jest powtarzany dla każdej liczby w ciągu.

Dla liczby 5 podaj: głębokość, wysokość i stopień

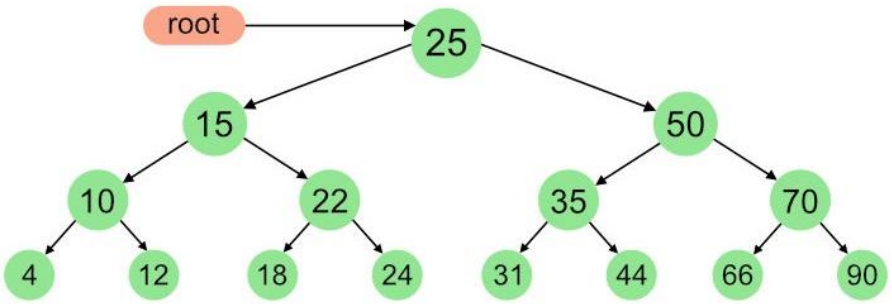
Zadanie 2

Do drzewa binarnego należy wstawić kolejno następujące liczby: 4, 2, 1, 3, 6, 5, 7. Która wersja przedstawia prawidłowe rozmieszczenie liczb, zgodne z regułą drzewa?



Zadanie 3.

- Zmodyfikuj program ze str. 2 i zapisz rekurencyjnie metody:
  PREORDER, POSTORDER przeglądania drzewa binarnego.
- Na postawie poniższego drzewa wypisz wartości wyświetlania metodą PREORDER, POSTORDER, INORDER



Zadanie 4

Napisz funkcję sprawdzającym czy dany element występuje w drzewie binarnym.

Zadanie 5

Sprawdzanie czy drzewo jest drzewem BST (Binary Search Tree): Napisz funkcję sprawdzającą, czy dane drzewo binarne jest drzewem BST. Zadanie polega na sprawdzeniu, czy dla każdego węzła drzewa lewe poddrzewo zawiera wartości mniejsze od wartości węzła, a prawe poddrzewo zawiera wartości większe.