Sprawozdanie 08: Obiegi drzew i przykłady ich zastosowania

Dmytro Sakharskyi - L02 - 31259 5.06.2025

Metody Programowania

Informatyka 1 rok. 2 semestr

Wstęp

Obiegi drzewa to podstawowe operacje wykonywane na strukturze drzewiastej, umożliwiające odwiedzanie wszystkich jej węzłów w określonej kolejności. Wyróżniamy różne typy obiegów, spośród których najczęściej stosowane to:

- **in-order (LPR)** – lewe poddrzewo, korzeń, prawe poddrzewo,
- **pre-order (PLR)** – korzeń, lewe poddrzewo, prawe poddrzewo.

W przypadku drzew binarnych kolejność ta jest jasno zdefiniowana, natomiast dla drzew dowolnego rzędu **(n-arnych)** konieczne jest odpowiednie przystosowanie algorytmu.

Celem niniejszego ćwiczenia było:

- zrozumienie zasad działania obiegów in-order oraz pre-order,
- implementacja tych obiegów w kontekście drzewa n-arnego,
- integracja ich z istniejącym kodem z wcześniejszego zadania,
- a także porównanie wyników z przykładami literaturowymi i graficzne przedstawienie struktury drzewa oraz obiegów.

Zadanie 1

Zadanie polegało na zintegrowaniu istniejącego kodu z wcześniejszego ćwiczenia (implementacja drzewa dowolnego rzędu) z dwiema funkcjami:

```
void inOrder(Node* node) {
    if (!node) return;
    int n = node->children.size();

    if (n > 0) inOrder(node->children[0]);

    cout << node->value << " ";

    for (int i = 1; i < n; ++i) {
        inOrder(node->children[i]);
    }
}
```

- **inOrder**() – odpowiadającą za obieg: pierwsze poddrzewo → korzeń → pozostałe poddrzewa.

```
void preOrder(Node* node) {
    if (!node) return;

    cout << node->value << " ";

    for (Node* child : node->children) {
        preOrder(child);
    }
}
```

- **preOrder**() – odpowiadającą za obieg drzewa w kolejności korzeń → dzieci,

Kod drzewa n-arnego z poprzedniego zadania został pozostawiony bez zmian — zachowano strukturę Node, w której każdy węzeł zawiera pole **vector<Node*> children**, przechowujące listę dzieci.

🔧 Dostosowane funkcje:

- Funkcja **preOrder**() odwiedza bieżący węzeł, a następnie rekurencyjnie każde z jego dzieci.
- Funkcja **inOrder**() sprawdza, czy dany węzeł posiada dzieci. Jeśli tak — odwiedza pierwsze dziecko, potem bieżący węzeł, a następnie resztę dzieci.

Obie funkcje zostały dopasowane do struktury n-arnego drzewa i w pełni współpracują z wcześniej utworzoną hierarchią węzłów.

Zadanie 2

Podczas implementacji funkcji **preOrder()** oraz **inOrder()** należało zwrócić szczególną uwagę na zgodność typów danych i wskaźników używanych w strukturze drzewa.

Struktura Node

Struktura Node została odziedziczona z poprzedniego zadania i wyglądała następująco:

```
struct Node {
    string value;
    vector<Node*> children;

    Node(const string& val) : value(val) {}
};
```

Typ wskaźnika: Wszystkie funkcje pracują na wskaźnikach typu Node*, co zapewnia zgodność z istniejącą strukturą.

Kolekcja dzieci: Użyto vector<Node*> jako kontenera na potomków, dzięki czemu możliwa jest iteracja przy użyciu pętli for, bez potrzeby modyfikowania oryginalnej definicji struktury.

Parametry funkcji: Zarówno preOrder(Node* node), jak i inOrder(Node* node) przyjmują argument w postaci wskaźnika Node*, co bezpośrednio pasuje do istniejącej logiki programu.

Zadanie 3

Zgodnie z poleceniem, struktura Node używana w poprzednim zadaniu miała **pozostać niezmieniona**. Oznaczało to konieczność dostosowania funkcji obiegu drzewa (inOrder oraz preOrder) do istniejącej definicji, bez ingerencji w jej składnię czy logikę.

Zamiast zakładać strukturę binarną (left, right), funkcje inOrder oraz preOrder zostały **zaadaptowane** do pracy z wektorem dzieci (vector<Node*> children).

W przypadku inOrder przyjęto logiczne podejście:

- jeśli istnieją dzieci, odwiedzane jest pierwsze dziecko,
- następnie bieżący węzeł,
- a na końcu pozostałe dzieci.

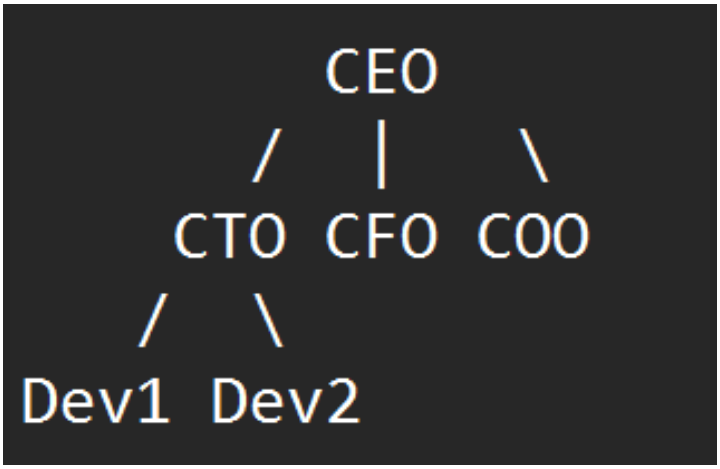
Funkcja preOrder została zaimplementowana tak, aby najpierw przetwarzać bieżący węzeł, a potem rekurencyjnie każde dziecko.

Dzięki temu obie funkcje są w pełni zgodne z drzewem dowolnego rzędu i działają poprawnie przy zachowaniu pierwotnej struktury danych.

Zadanie 4

Po pomyślnej implementacji funkcji inOrder() oraz preOrder(), ich wyniki zostały zaprezentowane w formie graficznej, zgodnie ze strukturą drzewa organizacyjnego przyjętą w poprzednim zadaniu.

Użyte drzewo (organizacyjne):



Obieg **pre-order (PLR)**

Kolejność odwiedzania:
CEO → CTO → Dev1 → Dev2 → CFO → COO

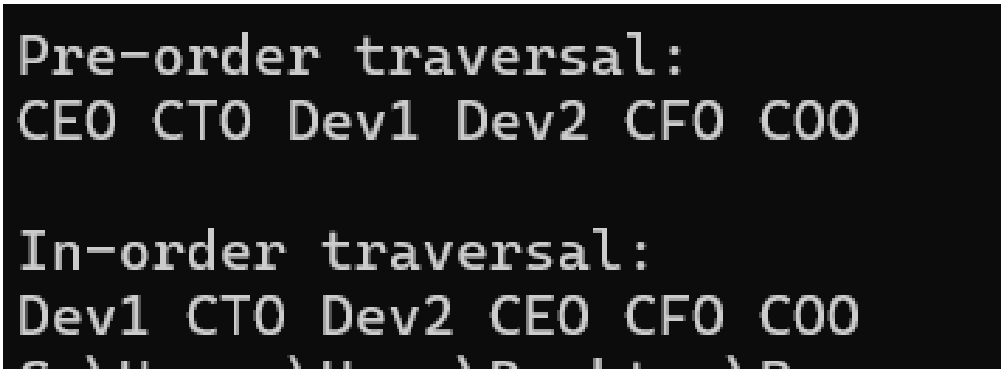
Opis: najpierw odwiedzany jest korzeń (“CEO”), następnie każdy z jego potomków w kolejności od lewej do prawej.

Obieg **in-order (LPR)**

Kolejność odwiedzania:
Dev1 → CTO → Dev2 → CEO → CFO → COO

Opis: odwiedzane jest najpierw pierwsze dziecko (jeśli istnieje), potem korzeń, a następnie pozostałe dzieci.

Przykład w programu:



Zadanie 5

Poprawności działania zaimplementowanych funkcji **preOrder()** oraz **inOrder()**, porównano uzyskane wyniki z przykładami zawartymi w literaturze oraz dostępnych źródłach internetowych (takich jak GeeksForGeeks oraz Wikipedia).

Wynik działania programu:

Pre-order traversal:
CEO CTO Dev1 Dev2 CFO COO

In-order traversal:
Dev1 CTO Dev2 CEO CFO COO

Porównanie:

- **Pre-order** (PLR):
  Zgodnie z definicją, najpierw przetwarzany jest węzeł główny („CEO”), a następnie wszystkie dzieci w kolejności od lewej do prawej.
  Wynik jest poprawny i spójny z przykładowymi obiegami drzew w literaturze.
- **In-order** (LPR – adaptowane do drzewa n-aryjnego):
  Ponieważ struktura nie jest binarna, przyjęto rozszerzoną wersję:
  - odwiedzane jest pierwsze dziecko,
  - następnie bieżący węzeł,
  - a na końcu pozostałe dzieci.
  - Wynik również jest poprawny i odpowiada adaptacji stosowanej dla drzew o dowolnym stopniu.

Wniosek:

Oba typy obiegów drzewa działają zgodnie z założeniami i są zgodne z opisami i przykładami w dostępnych źródłach. Wyniki wyświetlane przez program zostały potwierdzone jako poprawne.

Wnioski

W ramach ćwiczenia zaimplementowano dwa typy obiegu drzewa: **pre-order** oraz **in-order**, dostosowane do struktury drzewa dowolnego rzędu (n-arnego). Funkcje te zostały zintegrowane z wcześniej przygotowaną strukturą Node i działają zgodnie z jej logiką, bez konieczności wprowadzania zmian w definicji danych.

Dzięki implementacji oraz testom praktycznym:

- potwierdzono poprawność działania obu algorytmów,
- zaprezentowano wyniki obiegów w postaci tekstowej i graficznej,
- porównano otrzymane rezultaty z przykładami z zewnętrznych źródeł.

Zrealizowane zadania pozwoliły utrwalić różnice między sposobami przetwarzania drzewa oraz wykazać, że możliwe jest ich zastosowanie również w przypadku struktur o zmiennej liczbie dzieci.