Universitas Negeri Surabaya

Binary Tree

Presentasi oleh Kelompok 6

Anggota kelompok

- Kamaulina Fanni Wahyuningsih (2403021408 8)
- Nirmala Rosyidatul Luthfiana (2403021412 0)
- Ahmad Alvin Harsani (2403021414 9)
- Ilmia Marita (2403021406 0)
- Nabilah Febri N.A (2403021415 3)

Pendahuluan

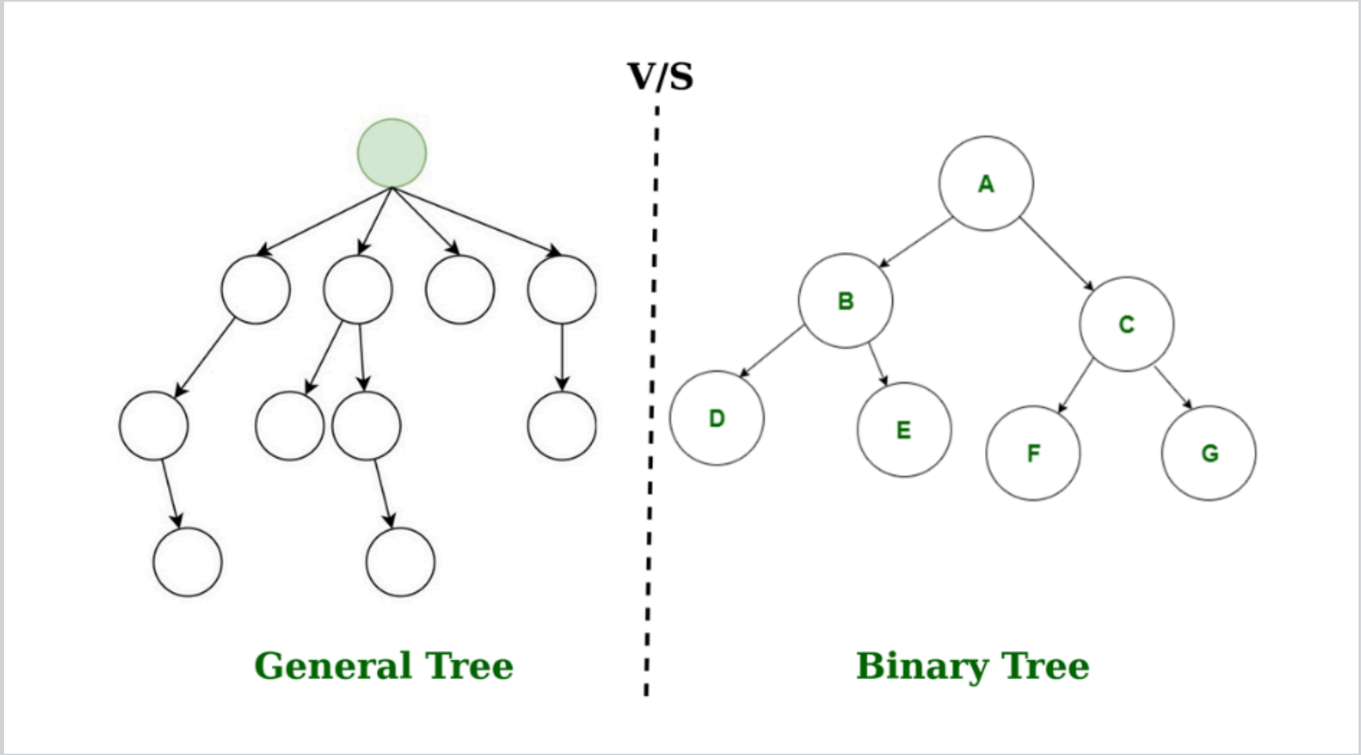
- **Binary Tree** adalah bentuk khusus dari tree dimana setiap node memiliki maksimal dua anak, yaitu anak kiri (left) dan anak kanan (right)
- Berdasarkan bentuknya, Binary Tree dapat berupa:
  - **Vine-like**
    Struktur menyerupai linked list (tidak seimbang).
  - **Bushy**
    Struktur yang seimbang dengan cabang kiri dan kanan relatif sama tinggi.

perbedaan general tree dengan binary tree

GENERAL TREE VS BINARY TREE

Aspek	GENERAL TREE	BINARY TREE
Jumlah Anak	0 atau lebih dari satu	Maksimal 2 anak
Derajat Node	Tidak terbatas	Maksimal 2
Subtree	Dapat lebih dari dua	Terdiri dari subtree kiri dan kanan
Pengurutan	Tidak memiliki urutan tertentu	Dapat diurutkan (misalnya pada BST)
Out-degree Maksimal	n	2

GENERAL TREE VS BINARY TREE



Struktur Data Binary Tree

- Setiap node memiliki maksimal dua anak, yaitu anak kiri (left) dan anak kanan (right)
- Node paling atas disebut root.
- Operasi dasar yang umum dilakukan meliputi:
  - Searching (pencarian)
  - Inserting (penambahan)
  - Removing (Penghapusan)
  - Traversing (Penelusuran)

Implementasi Python

- Binary Tree dapat diimplementasikan menggunakan **linked list**.
- Setiap node memiliki tiga atribut utama:
  - data
  - left (anak kiri)
  - right (anak kanan)
- Contoh Kelas:

```
class BinaryTree:
    def __init__(self, data=None):
        self.data = data
        self.left = None
        self.right = None
```

Operasi pencarian

Langkah algoritma:

1. Periksa node root.
2. Jika belum ditemukan, periksa subtree kiri secara rekursif.
3. Jika masih belum ditemukan, lanjutkan ke subtree kanan.
4. Kembalikan node jika ditemukan, atau None jika tidak ada hasil.

Operasi penambahan

- **Insert Left** → Menambahkan node baru sebagai anak kiri.
- **Insert Right** → Menambahkan node baru sebagai anak kanan.
- Langkah umum:
  - Temukan node referensi.
  - Tambahkan anak kiri atau kanan jika posisi masih kosong.

Operasi penghapusan

- **Remove Left/Right** digunakan untuk menghapus anak kiri atau kanan beserta seluruh keturunannya.
- Implementasi dilakukan dengan cara mengosongkan referensi anak menggunakan None.

Traversal Binary Tree

- **Preorder:** Root → Kiri → Kanan
  Umumnya digunakan untuk menyalin atau mengkontruksi tree.
- **Inorder:** Kiri → Root → Kanan
  Pada Binary Search Tree, hasil traversal bersifat terurut.
- **Postorder:** Kiri → Kanan → Root
  Digunakan untuk menghapus seluruh node tree

Contoh Traversal

```
# Struktur Tree:
#       D
#     /   \
#    B     F
#   /\     /\
#  A  C   E  G
```

Hasil Traversal

- **Preorder** : D B A C F E G
- **Inorder** : A B C D E F G
- **Postorder** : A C B E G F D

Hasil menunjukkan bahwa urutan traversal tergantung pada posisi dan urutan penelusuran node.

Studi Kasus

Dalam sistem akademik kampus, bagian administrasi ingin menyimpan data NIM mahasiswa agar pencarian data bisa dilakukan dengan cepat dan lebih efisien. Jika data disimpan dalam daftar biasa, proses pencarian akan lama karena memeriksa data satu per satu.

Untuk mengatasi hal tersebut, digunakan Binary Search Tree (BST) agar data tersusun rapi dan mudah dicari. Misalnya, data NIM mahasiswa yang akan dimasukkan adalah:
[123, 111, 135, 109, 120, 140, 130]

Link google colab

https://colab.research.google.com/drive/16lepk0Ai6sqsYbSho-lBZTnYswlpzYl_?usp=sharingda

Kompleksitas

Operasi	Waktu (Best)	Waktu (Worst)	Keterangan
insert ( )	O(log n)	O(n)	Pencarian posisi penyisipan
Search ( )	O(log n)	O(n)	Bandingkan kiri-kanan rekursif
Inorder ( )	O(n)	O(n)	Kunjungi semua node satu kali

Apakah ada pertanyaan?

Sesi

Tanya

Jawab

Terima Kasih

Presentasi oleh Kelompok 6