Tango Trees

Presented by-
Garima Kamra (2024CSB1115)
Palak (2024CSB1137)
Aarushi Goel (2024MCB1283)
Archita (2024MCB1288)
Komal Goyal (2024MCB1298)

Tango Trees

A dynamic binary search tree optimized using preferred paths and auxiliary splay trees to achieve competitive search performance.

Proposed in **2004**

by _Erik D. Demaine, Dion Harmon, John Iacono, and Mihai Pătraşcu.

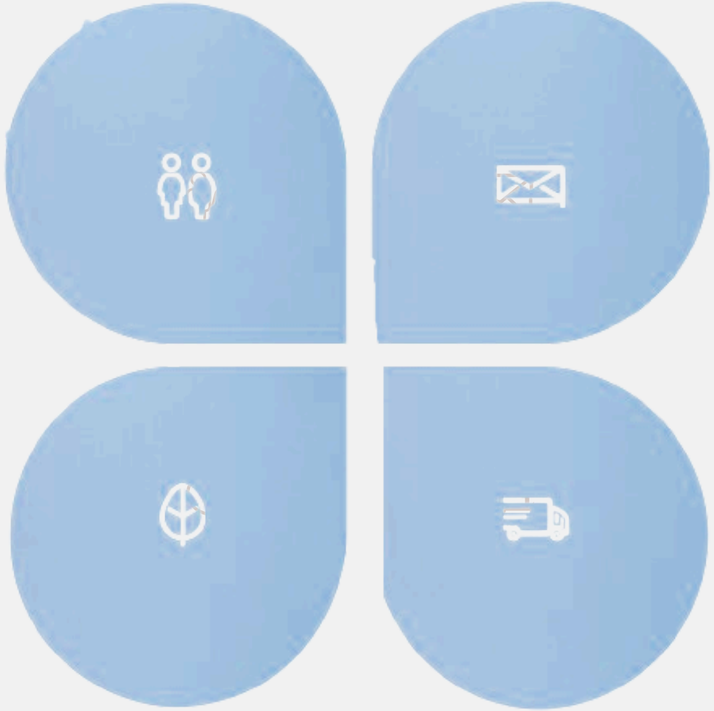
Why name ‘Tango’?

- Searches create preferred paths.
- Deviations flip edges and reshape the structure.
- Paths keep splitting and merging like a rhythmic dance.
- Hence the name “Tango Tree.”

Intuition: Like a Dance

Changing Partners

Nodes change preferred children.



Dynamic Paths

Paths join and break dynamically.

Adaptive Structure

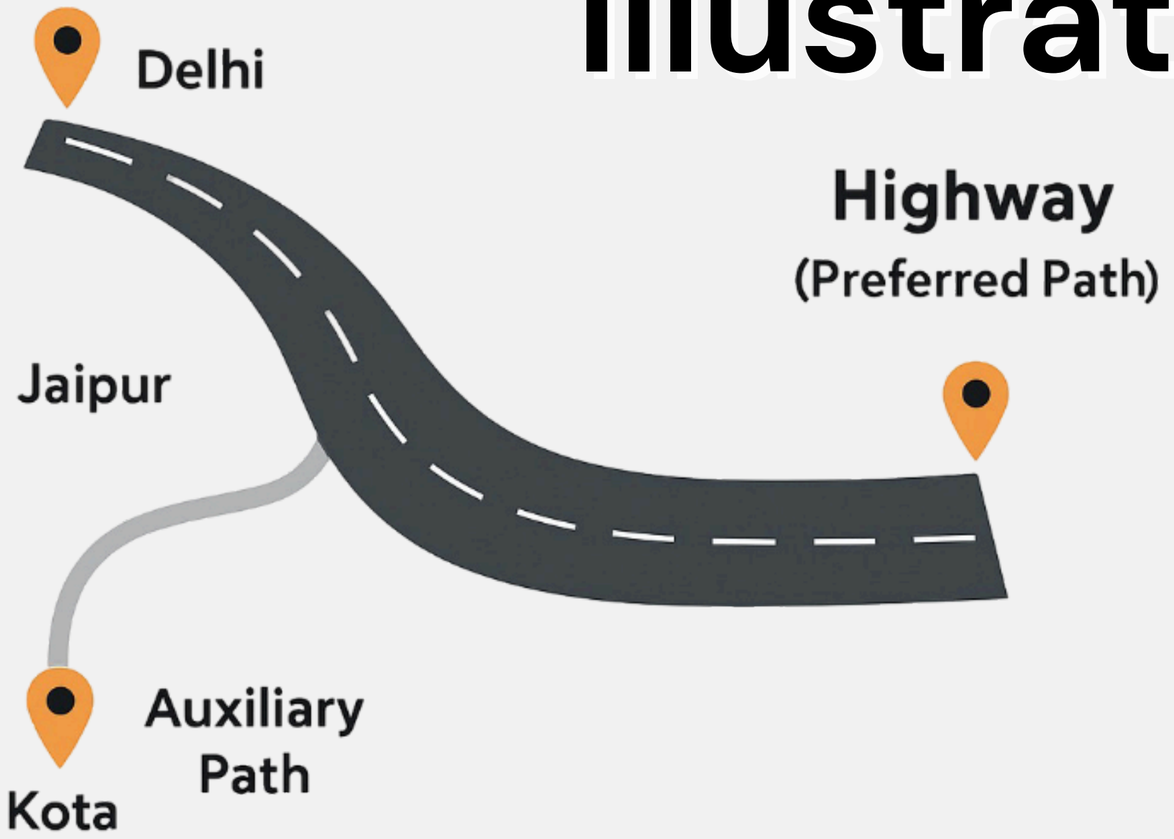
Adapts based on search rhythm.

Fast Movement

Coordinated and rapid movement.

Hence the name: Tango Tree.

Illustration



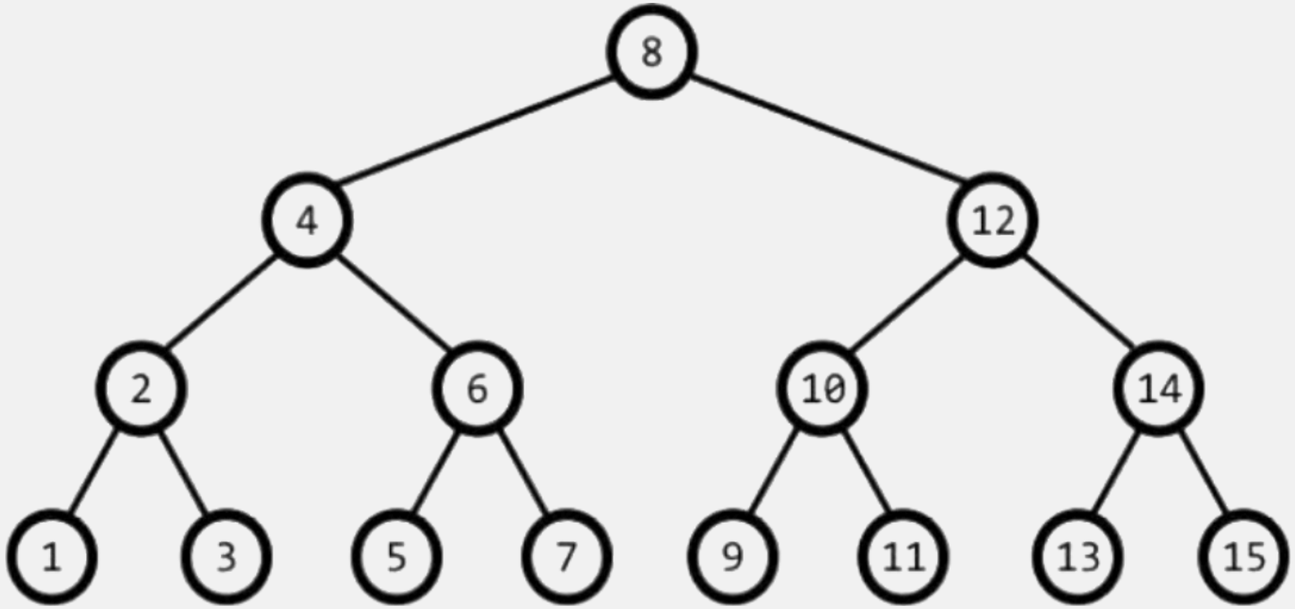
A Tango Tree works like a **road system**:

- The path that is used most often becomes a **highway** → fast and direct.
- Less frequently visited paths remain as **smaller side roads.**

What are Tango Trees?

Tango Tree is a **self-adjusting BST** that uses preferred paths based on recent searches.

- It splits the reference BST into small auxiliary trees for faster navigation.
- Each operation takes O(log n) worst-case time.
- Overall, it achieves a O(log log n) competitive ratio, making it faster than standard balanced BSTs.



Key Concepts

Preferred Child

01. The child through which the most recent search passed.

Preferred Path

02. A chain of nodes connected only via preferred-child edges.

Auxilliary Tree

03. Each preferred path is stored as a splay tree (balanced, self-adjusting BST).

Important Properties

- **Dynamic:** Adjusts continuously with each search.
- **Online Algorithm:** Uses only past information; no future knowledge required.
- **Self-Balancing**
- **Efficient**
- **Search**: O(log n)
- **Competitive Ratio**: O(log log n)

Node Structure

- Key Value
- Left Child Pointer
- Right Child Pointer
- Preferred Child Pointer
- Pointer to its auxiliary tree

```
class TangoNode {
 int value;
 TangoNode *left, *right;
 TangoNode(int v): value(v),
left(nullptr), right(nullptr) {}
};
```

Searching

Insertion

Core Operation

Deletion

Searching Process

1.

Follow BST Path

Navigate as in a normal BST.

2.

Splay Node

Splay the node within its auxiliary tree.

3.

Flip Edge

If next step differs from preferred child, flip the preferred edge.

4.

Split & Join

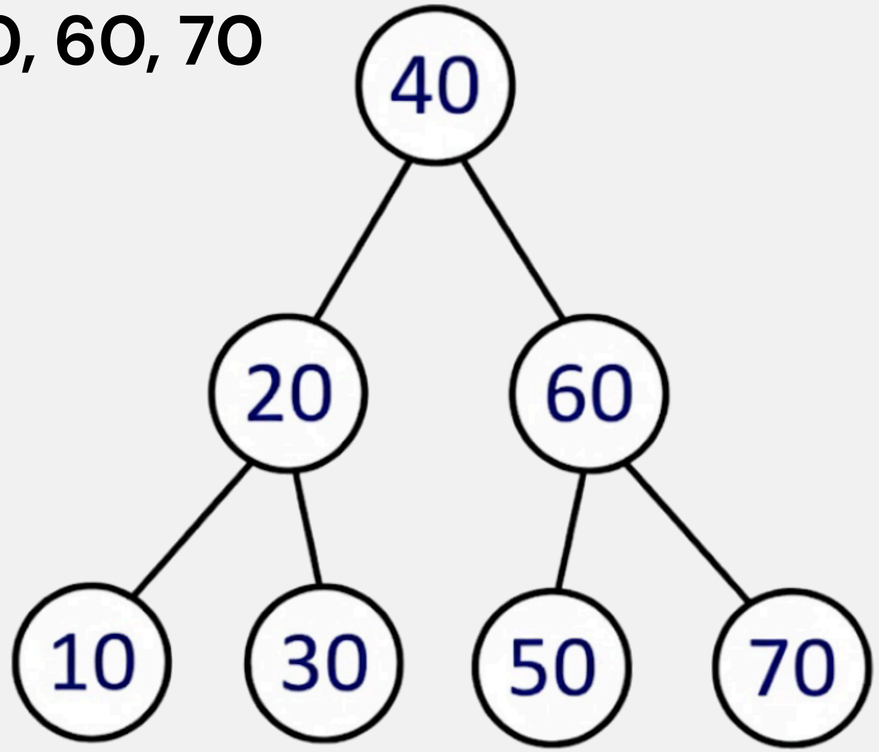
Split and join auxiliary trees accordingly.

The target is splayed to the top of its auxiliary tree.

Searching Process
Example

Nodes- 10, 20, 30, 40, 50, 60, 70

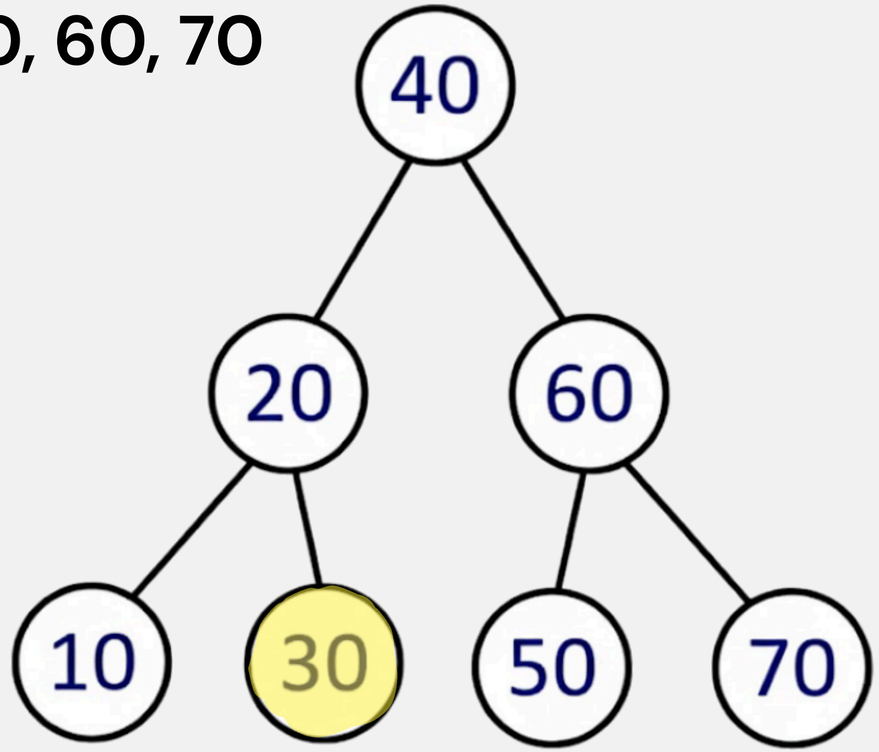
STEP-1
Search for 30



Searching Process
Example

Nodes- 10, 20, 30, 40, 50, 60, 70

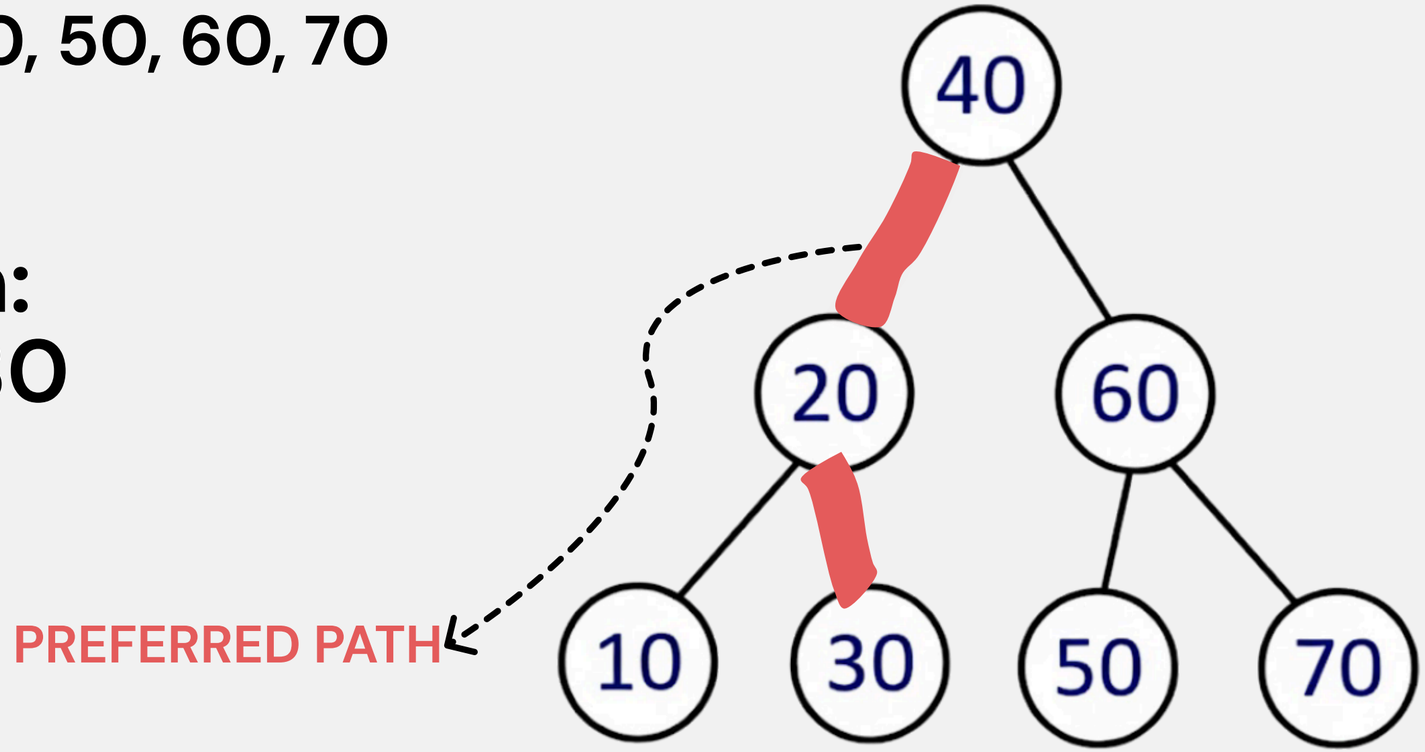
STEP-1
Search for 30



Searching Process
Example

Nodes- 10, 20, 30, 40, 50, 60, 70

Search Path:
40 → 20 → 30

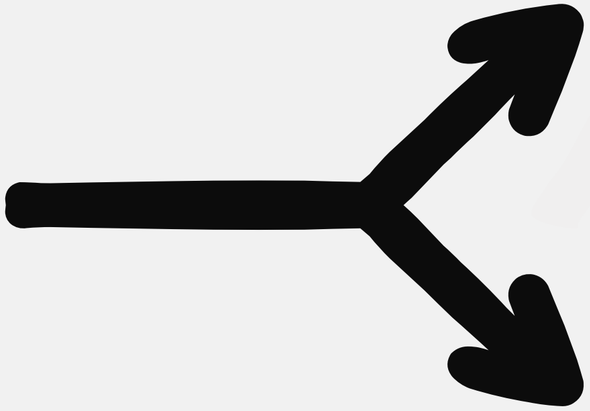


Searching Process
Example

STEP-2
Auxiliary Tree

Searching Process
Example

STEP-3
Future search behavior



Searching 20 and 30 is now very fast

if path deviates, for e.g. searching 60, the tree adjusts dynamically

Operations: Insertion & Deletion

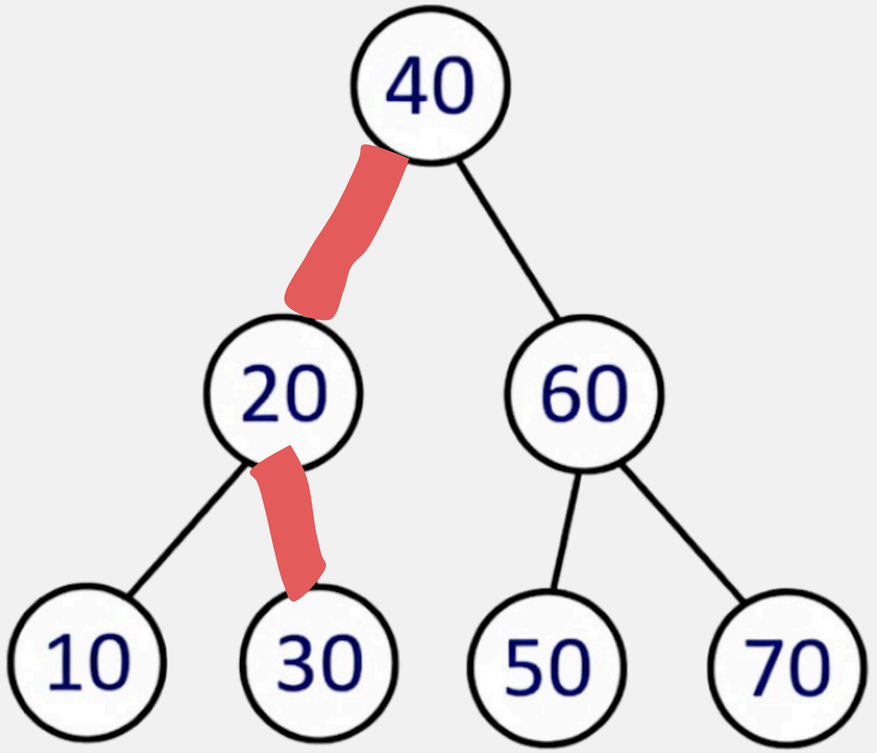
Insertion & Deletion

- Insert like a BST
- Rebuild/update auxiliary trees.
- Maintain preferred-child invariants.

Insertion Process
Example

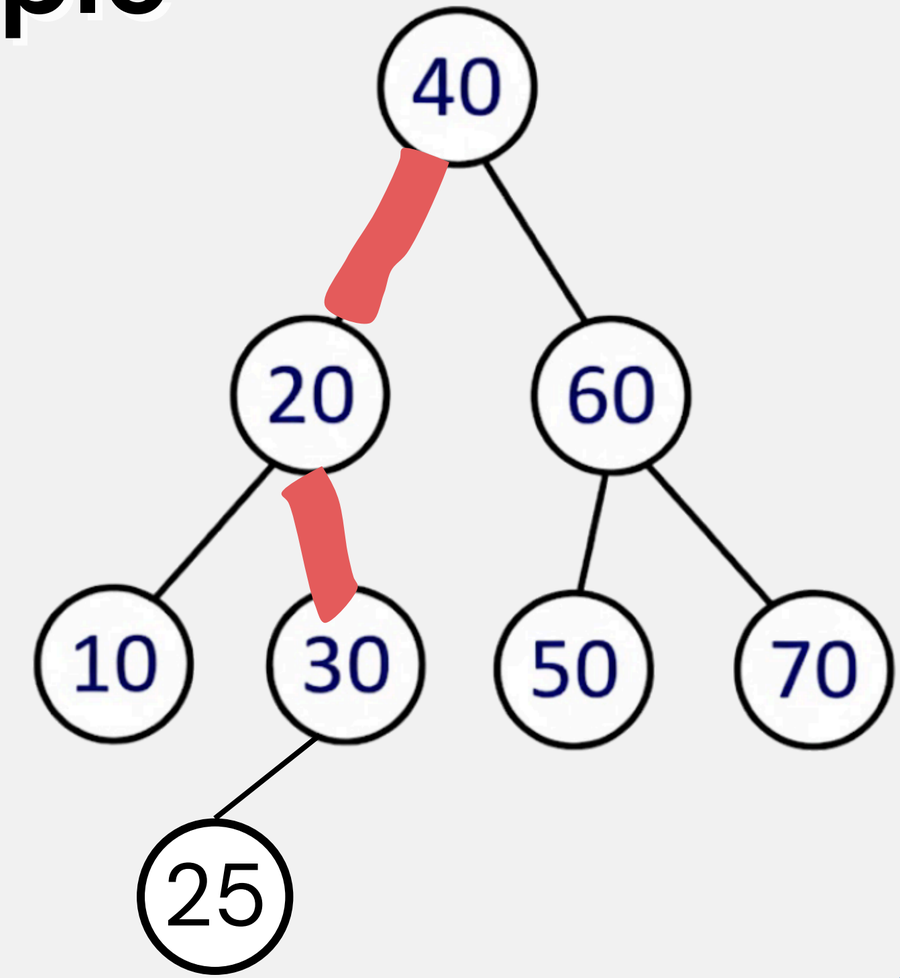
Nodes- 10, 20, 30, 40, 50, 60, 70

STEP-1
Insert 25



Insertion Process
Example

Nodes- 10, 20, 30, 40, 50, 60, 70



STEP-2
Insertion like normal BST

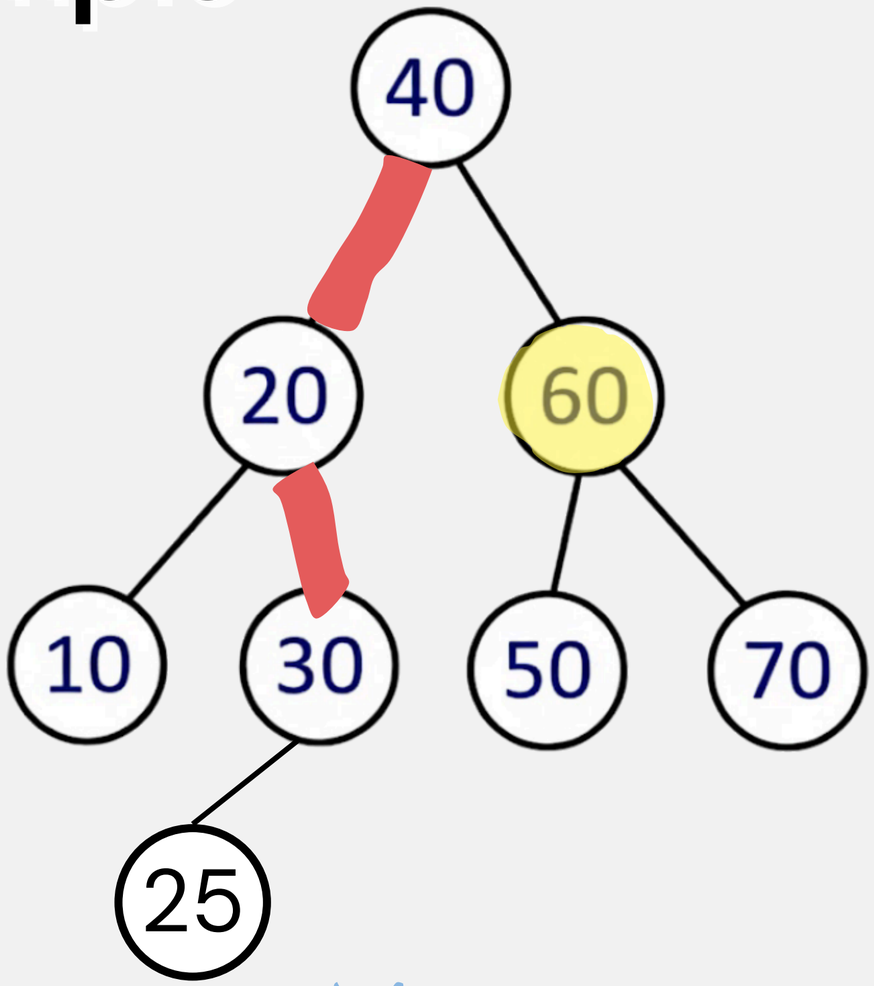
PREFERRED PATH does not change during insertion

Deletion Process

Example

Nodes- 10, 20, 30, 40, 50, 60, 70

STEP-1
Delete 60

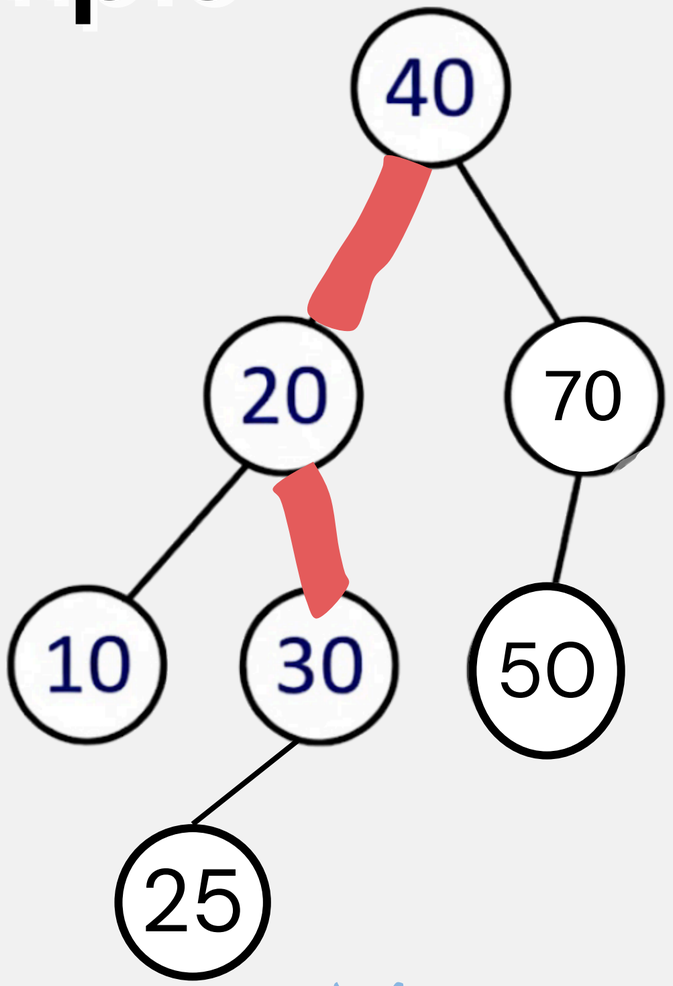


Deletion Process

Example

Nodes- 10, 20, 30, 40, 50, 60, 70

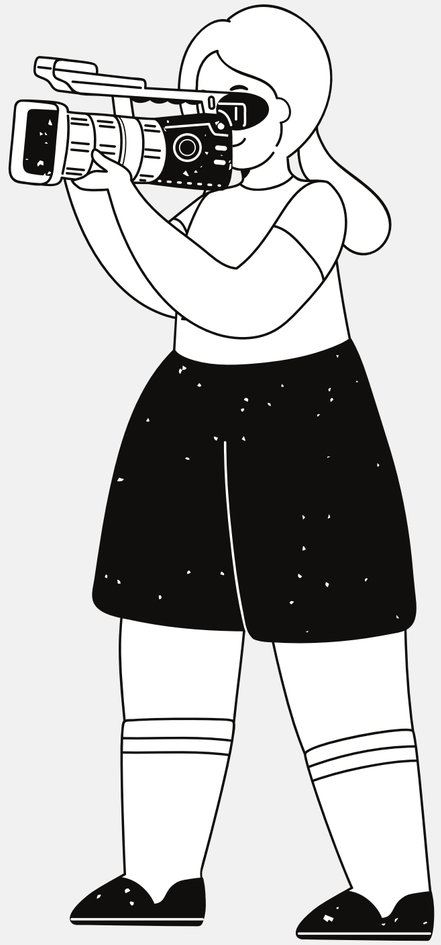
Normal BST Deletion



Advantages

Efficiency:

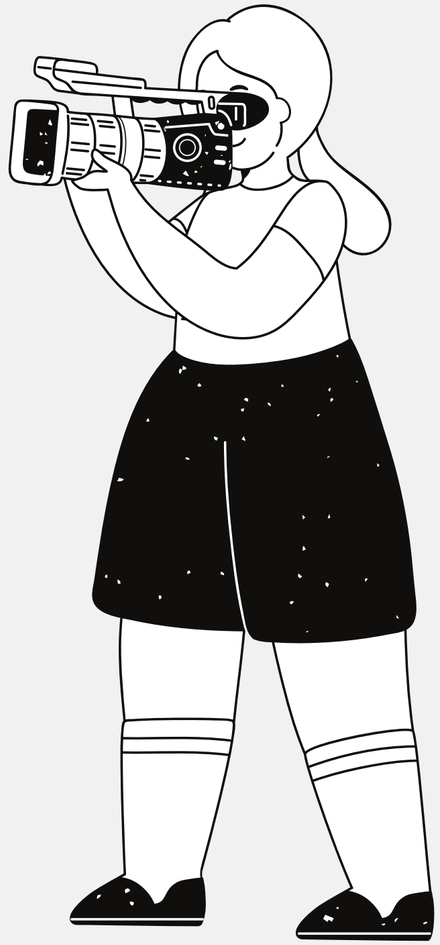
- Adapts to search patterns for faster queries.
- Offers better competitive performance than standard balanced trees.



Advantages

Versatility:

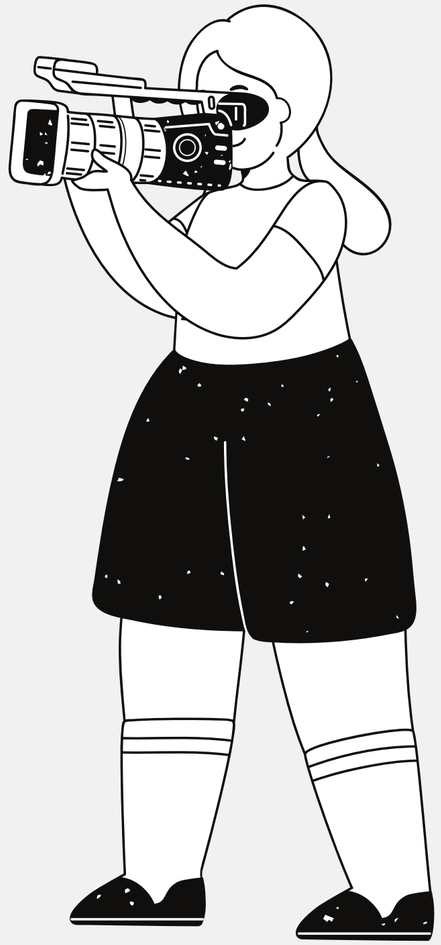
- Works well for dynamic datasets.
- Useful in competitive programming, dynamic queries, and algorithmic research.



Advantages

Space Optimization:

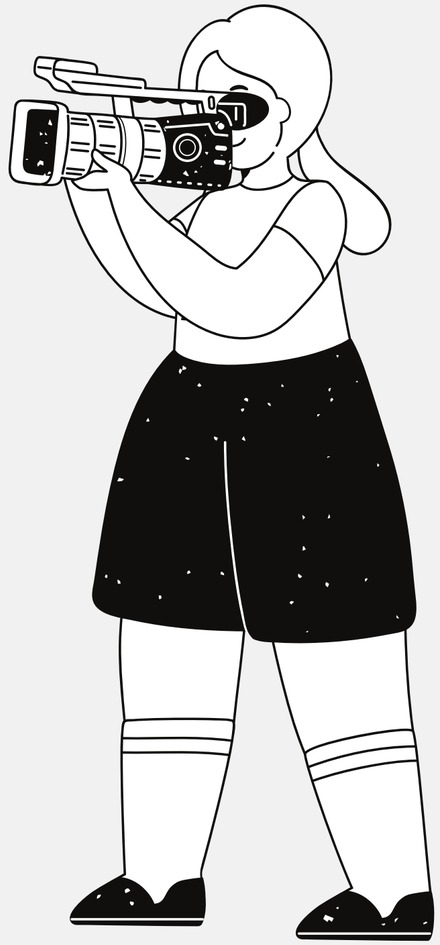
- Maintains multiple structures without excessive memory overhead.
- Efficiently manages auxiliary splay trees.



Advantages

Elegant Implementation:

- Combines simple concepts (preferred paths + splay trees).
- Once understood, the structure is clean and intuitive to use.



Thank you very much!

– Group 10