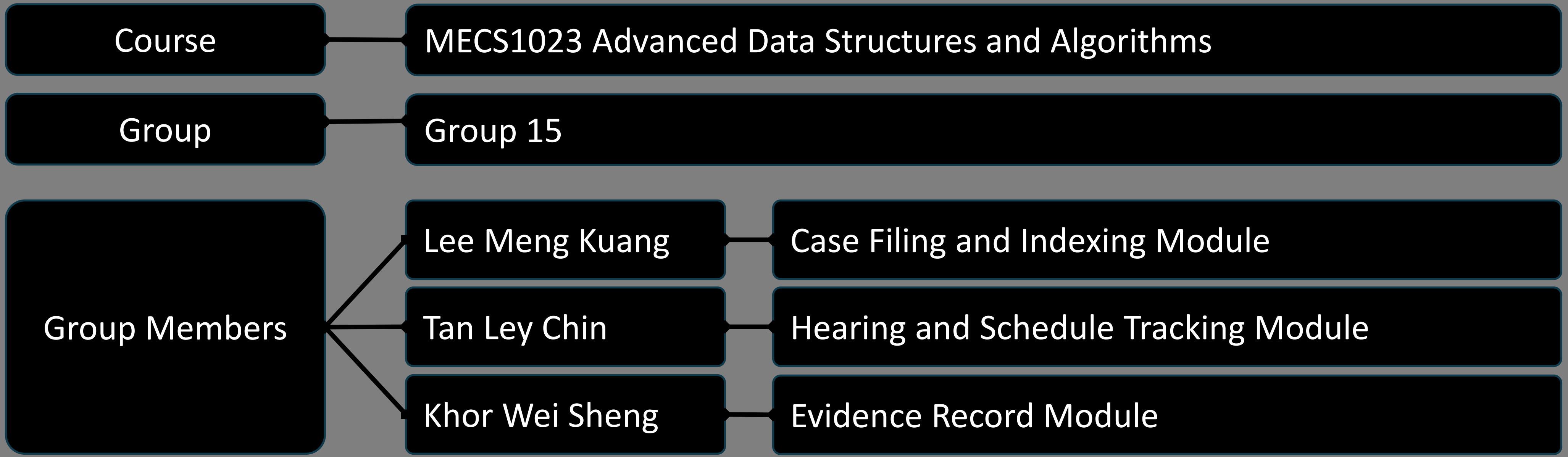
E-JUDICIARY

CASE MANAGEMENT SYSTEM

ASSIGNMENT Solution 1 - BST



Course	MECS1023 Advanced Data Structures and Algorithms	
Group	Group 15	
Group Members	Lee Meng Kuang	Case Filing and Indexing Module
	Tan Ley Chin	Hearing and Schedule Tracking Module
	Khor Wei Sheng	Evidence Record Module

BACKGROUND AND PROBLEM STATEMENT

- Judiciary case databases contain thousands of records
- Slow retrieval of case records
- Difficulty managing large case databases
- Poor indexing and time wasted on administrative tasks

E-JUDICIARY

CASE FILLING & INDEXING SYSTEM

Programming Language: Python

ASSIGNMENT Solution 1

Prepared by	Group 15 – Lee Meng Kuang MEC255019
Course	MECS1023 Advanced Data Structures and Algorithms

SYSTEM OVERVIEW

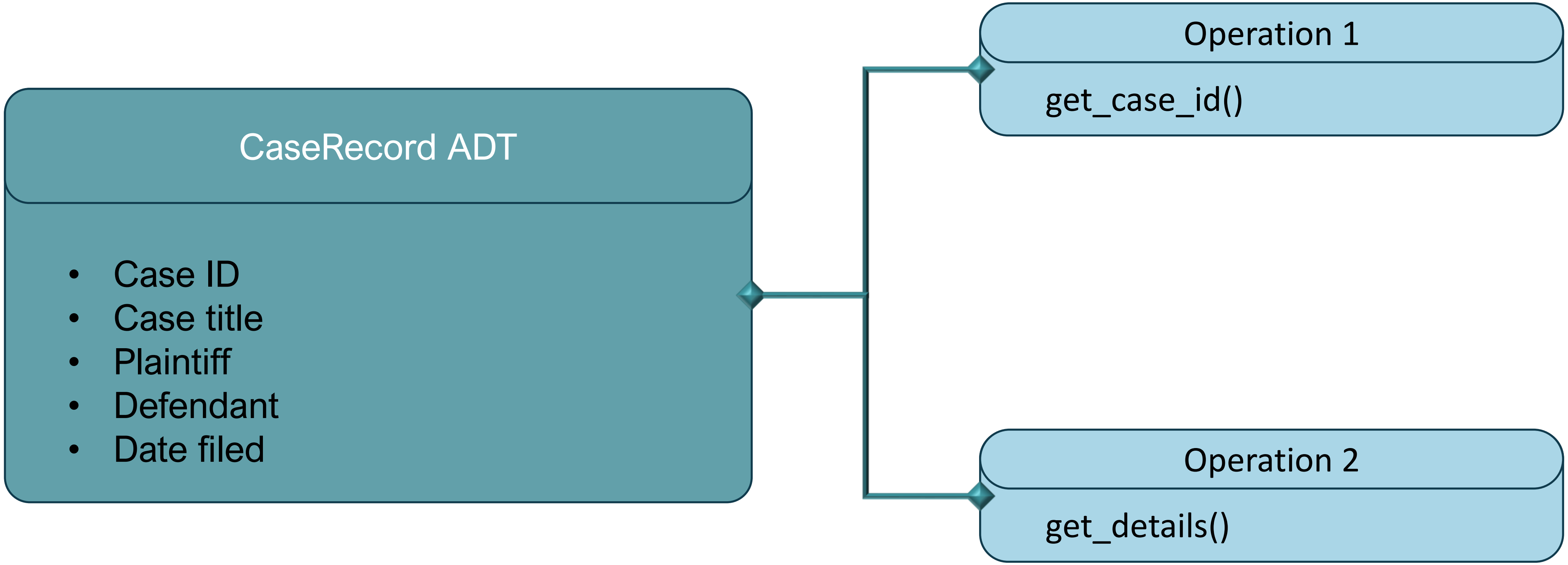
The system is built around 2 important Abstract Data Types (ADTs):

- CaseRecord ADT
- CaseIndexBST ADT

Programming Language:

🟩 Python

SYSTEM OVERVIEW (Con't)



CaseRecord ADT

- Case ID
- Case title
- Plaintiff
- Defendant
- Date filed

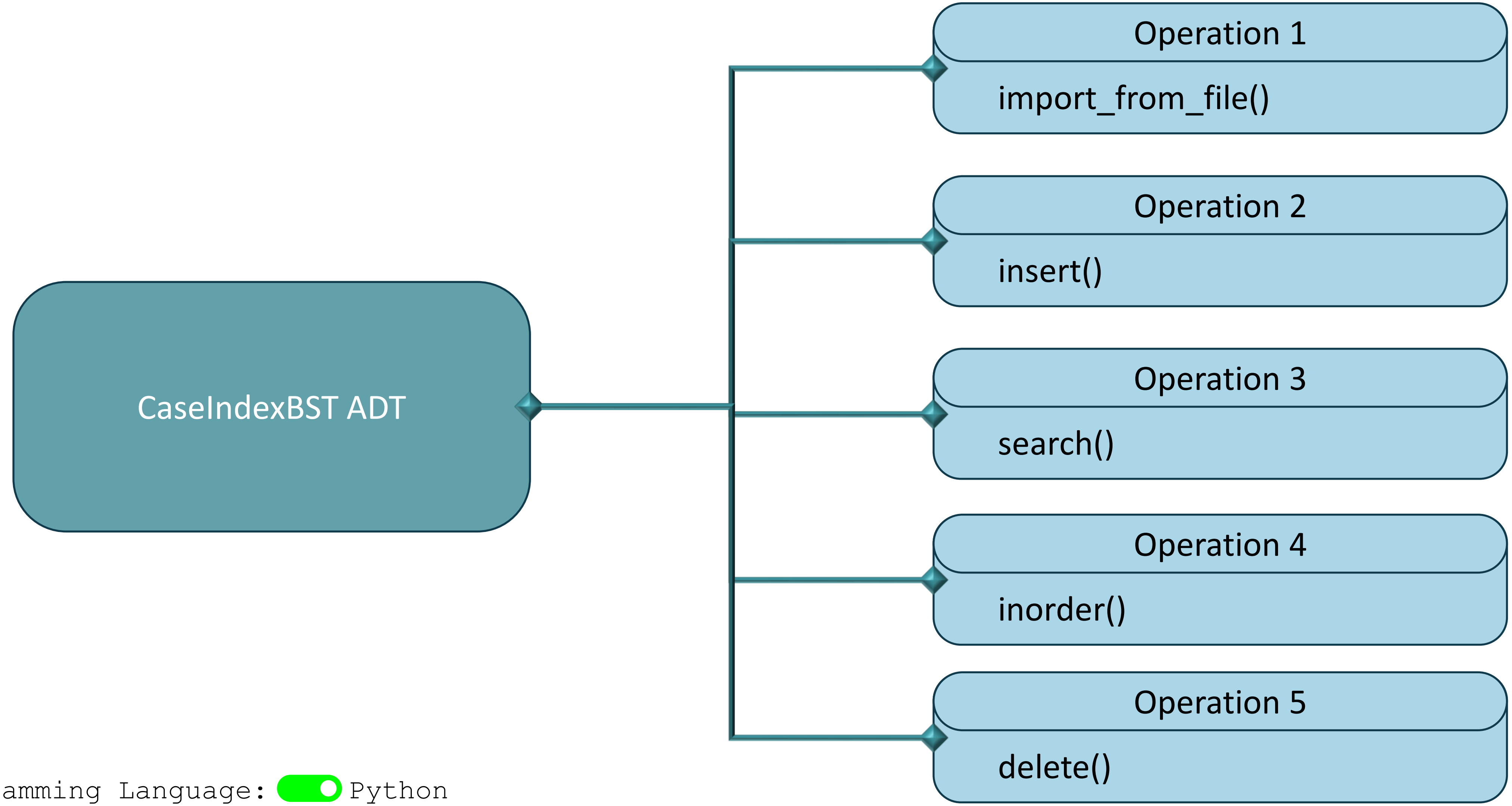
Operation 1

get_case_id()

Operation 2

get_details()

SYSTEM OVERVIEW (Con't)



CaseIndexBST ADT

Operation 1

import_from_file()

Operation 2

insert()

Operation 3

search()

Operation 4

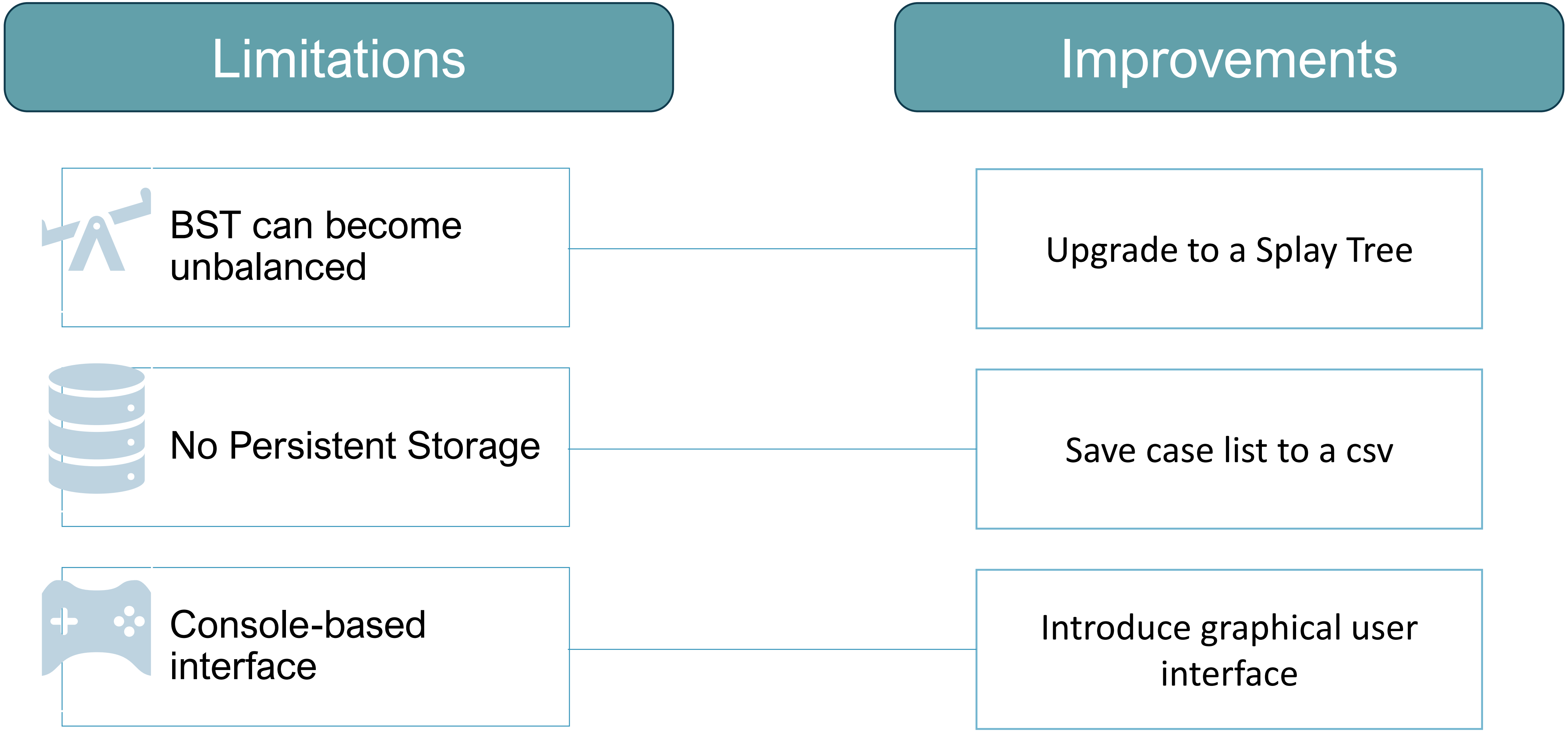
inorder()

Operation 5

delete()

Programming Language: ● Python

Limitations and Improvements



Limitations	Improvements
BST can become unbalanced	Upgrade to a Splay Tree
No Persistent Storage	Save case list to a csv
Console-based interface	Introduce graphical user interface

MECS1023-52(ADVANCED DATA STRUCTURE AND ALGORITHM))

G15 E-JUDICIARY CASE MANAGEMENT SYSTEM

Task 2 (Group) — Solution 1: Binary Search Tree (BST)

Module: Hearing and Schedule Tracking Module

STUDENT: TAN LEY CHIN MEC245059

PROJECT OVERVIEW

Project Goal:

Develop three independent BST-based modules for judiciary operations:

- Case Filing and indexing Module
- **Hearing & Schedule Tracking Module**
- Evidence Record Module

Solution 1 Requirement:

Implement a **BST-based search solution** using Abstract Data Type (ADT) structure, including:

- Insert
- Search
- Update
- Display (Traverse)
- CSV → BST Loading

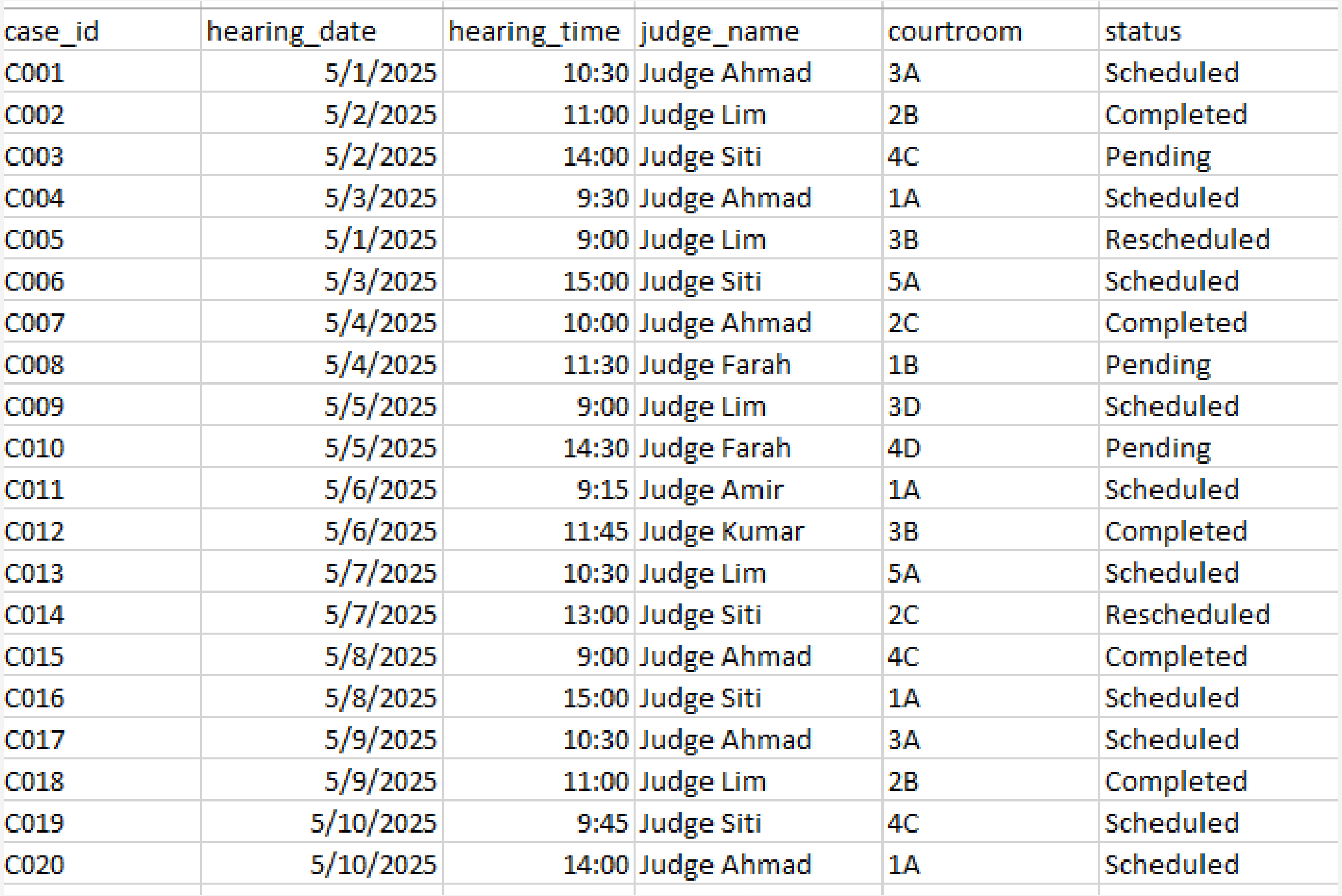
WHY BST?

- Efficient **logarithmic search** in average case
- Ordered storage based on Case ID
- Easy to insert and retrieve hearing schedules
- ADT structure matches assignment requirements
- Ideal for small–medium datasets (20–200 cases)

Hearing & Schedule Module uses a dataset of **20 hearing records**, with fields:

- case_id
- hearing_date
- hearing_time
- judge_name
- courtroom
- status

Dataset (CSV Input)

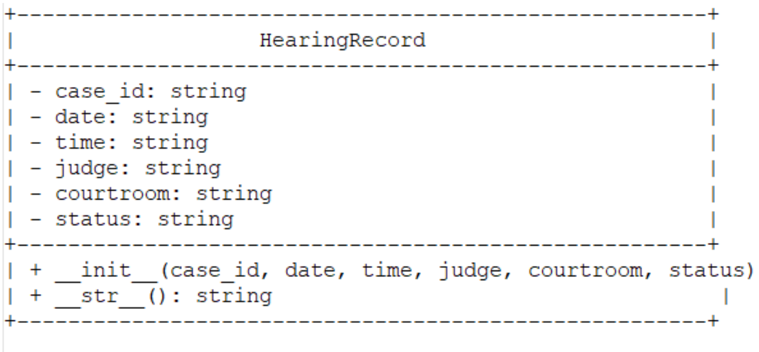


case_id	hearing_date	hearing_time	judge_name	courtroom	status
C001	5/1/2025	10:30	Judge Ahmad	3A	Scheduled
C002	5/2/2025	11:00	Judge Lim	2B	Completed
C003	5/2/2025	14:00	Judge Siti	4C	Pending
C004	5/3/2025	9:30	Judge Ahmad	1A	Scheduled
C005	5/1/2025	9:00	Judge Lim	3B	Rescheduled
C006	5/3/2025	15:00	Judge Siti	5A	Scheduled
C007	5/4/2025	10:00	Judge Ahmad	2C	Completed
C008	5/4/2025	11:30	Judge Farah	1B	Pending
C009	5/5/2025	9:00	Judge Lim	3D	Scheduled
C010	5/5/2025	14:30	Judge Farah	4D	Pending
C011	5/6/2025	9:15	Judge Amir	1A	Scheduled
C012	5/6/2025	11:45	Judge Kumar	3B	Completed
C013	5/7/2025	10:30	Judge Lim	5A	Scheduled
C014	5/7/2025	13:00	Judge Siti	2C	Rescheduled
C015	5/8/2025	9:00	Judge Ahmad	4C	Completed
C016	5/8/2025	15:00	Judge Siti	1A	Scheduled
C017	5/9/2025	10:30	Judge Ahmad	3A	Scheduled
C018	5/9/2025	11:00	Judge Lim	2B	Completed
C019	5/10/2025	9:45	Judge Siti	4C	Scheduled
C020	5/10/2025	14:00	Judge Ahmad	1A	Scheduled

■ ADT CLASS STRUCTURE (UML)

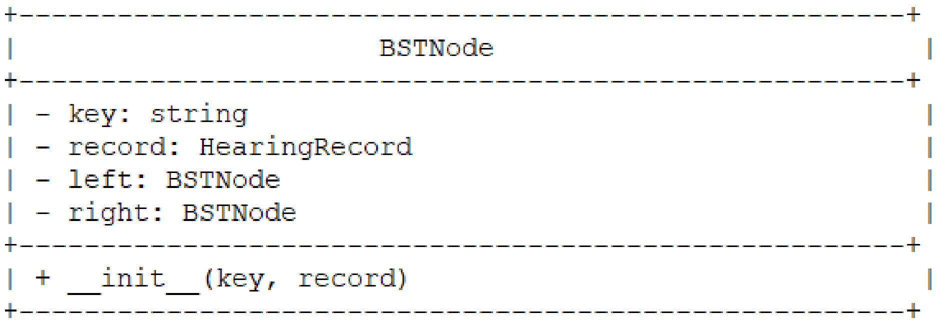
Class Implemented:

1) Hearing Record: Stores hearing attributes



```
+-------------------------------------------------------+
|                    HearingRecord                      |
+-------------------------------------------------------+
| - case_id: string                                     |
| - date: string                                        |
| - time: string                                        |
| - judge: string                                       |
| - courtroom: string                                   |
| - status: string                                      |
+-------------------------------------------------------+
| + __init__(case_id, date, time, judge, courtroom, status)
| + __str__(): string                                    |
+-------------------------------------------------------+
```

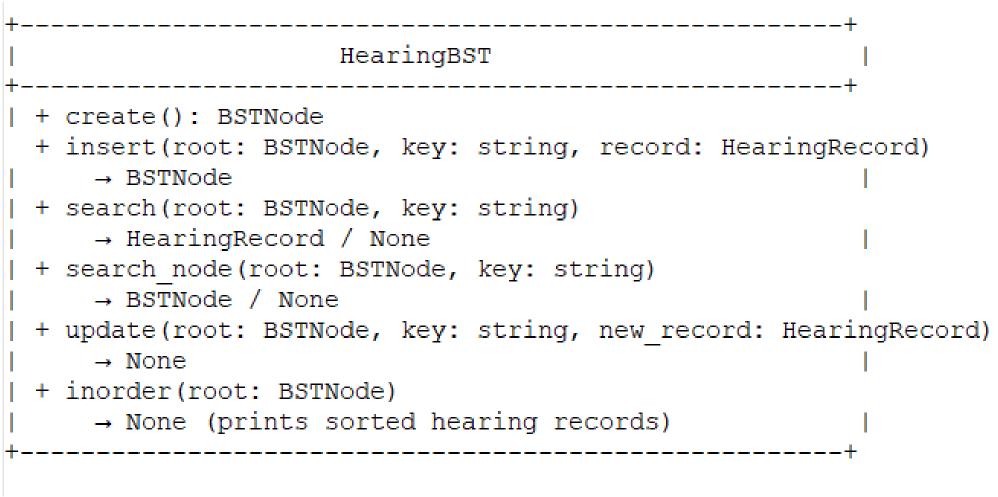
2) BST Node: Defines the structure of each node in the BST.



```
+-------------------------------------------------------+
|                        BSTNode                         |
+-------------------------------------------------------+
| - key: string                                          |
| - record: HearingRecord                                |
| - left: BSTNode                                        |
| - right: BSTNode                                       |
+-------------------------------------------------------+
| + __init__(key, record)                                |
+-------------------------------------------------------+
```

3) HearingBST (ADT) :

The ADT that supports Insert, Search, Update, and Inorder Traversal operations



```
+-------------------------------------------------------+
|                      HearingBST                        |
+-------------------------------------------------------+
| + create(): BSTNode
  + insert(root: BSTNode, key: string, record: HearingRecord)
|     → BSTNode                                          |
| + search(root: BSTNode, key: string)
|     → HearingRecord / None                             |
| + search_node(root: BSTNode, key: string)
|     → BSTNode / None                                   |
| + update(root: BSTNode, key: string, new_record: HearingRecord)
|     → None                                             |
| + inorder(root: BSTNode)
|     → None (prints sorted hearing records)             |
+-------------------------------------------------------+
```

▪ ADT OPERATIONS

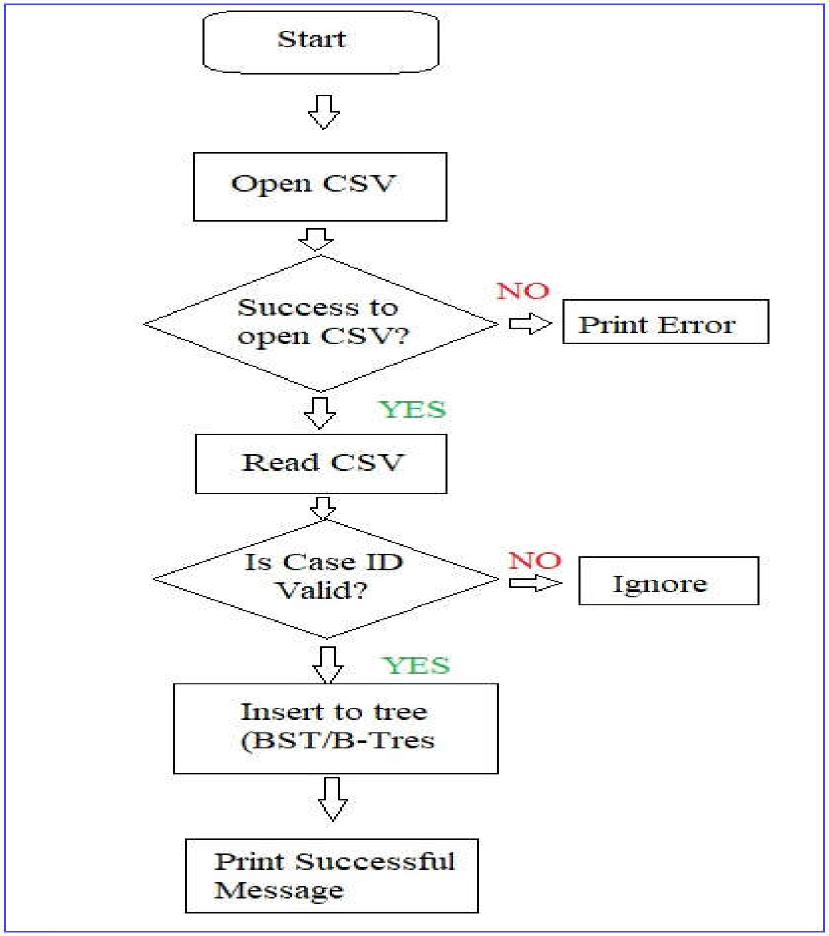
BST ADT Functions Used:

- **Create()** - Initialize empty BST
- **Insert(record) –** Add hearing record by Case ID
- **Search(key) –** Find record by Case ID
- **Update(key, new_record) –** Modify existing record
- **Traverse() –** Inorder (sorted display)

FLOW CHART: Load CSV (Create + Insert Loop)

Steps:

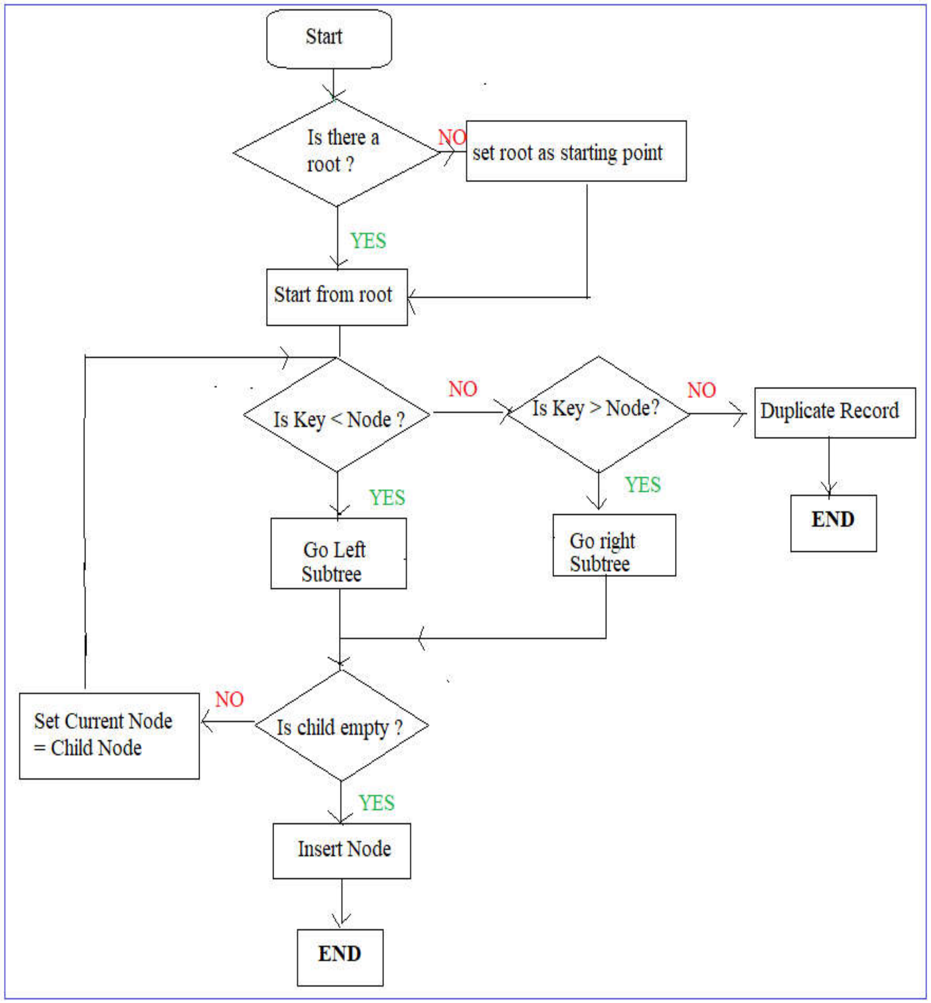
- Open CSV
- Validate row
- Insert into BST
- Continue until end of file



FLOW CHART: BST INSERT

Steps:

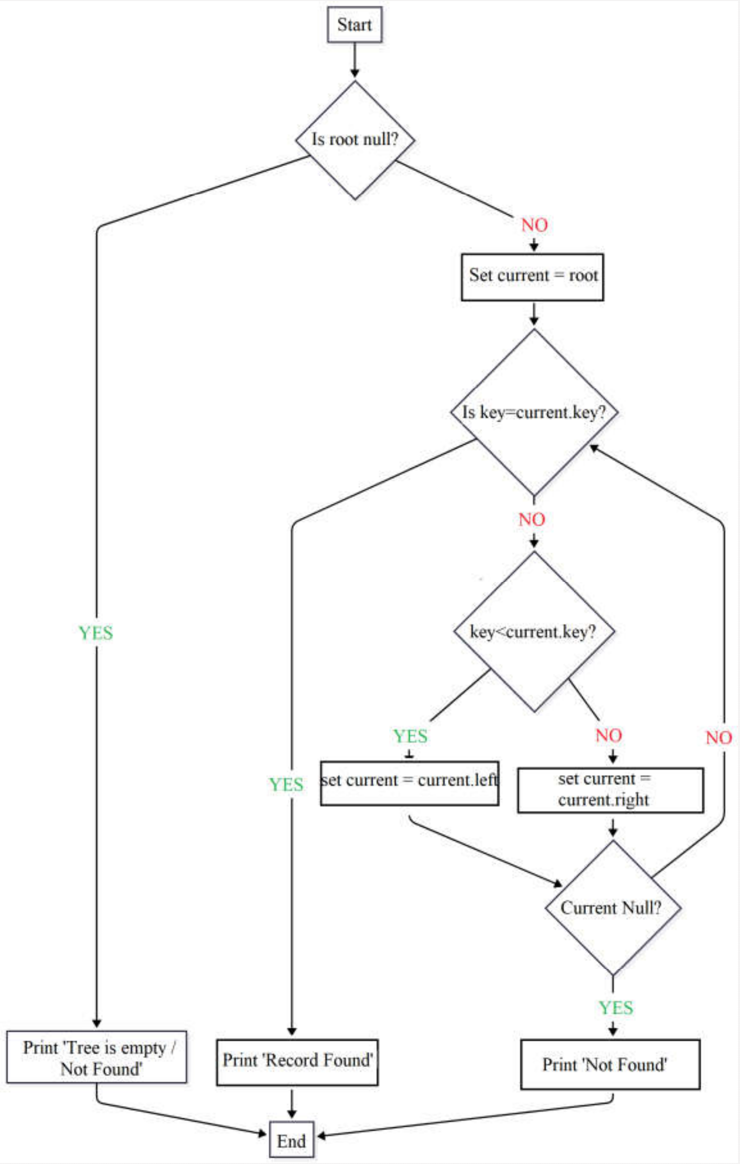
- Compare key
- Go left / go right
- Insert node
- Reject duplicates



FLOW CHART: BST SEARCH

Steps:

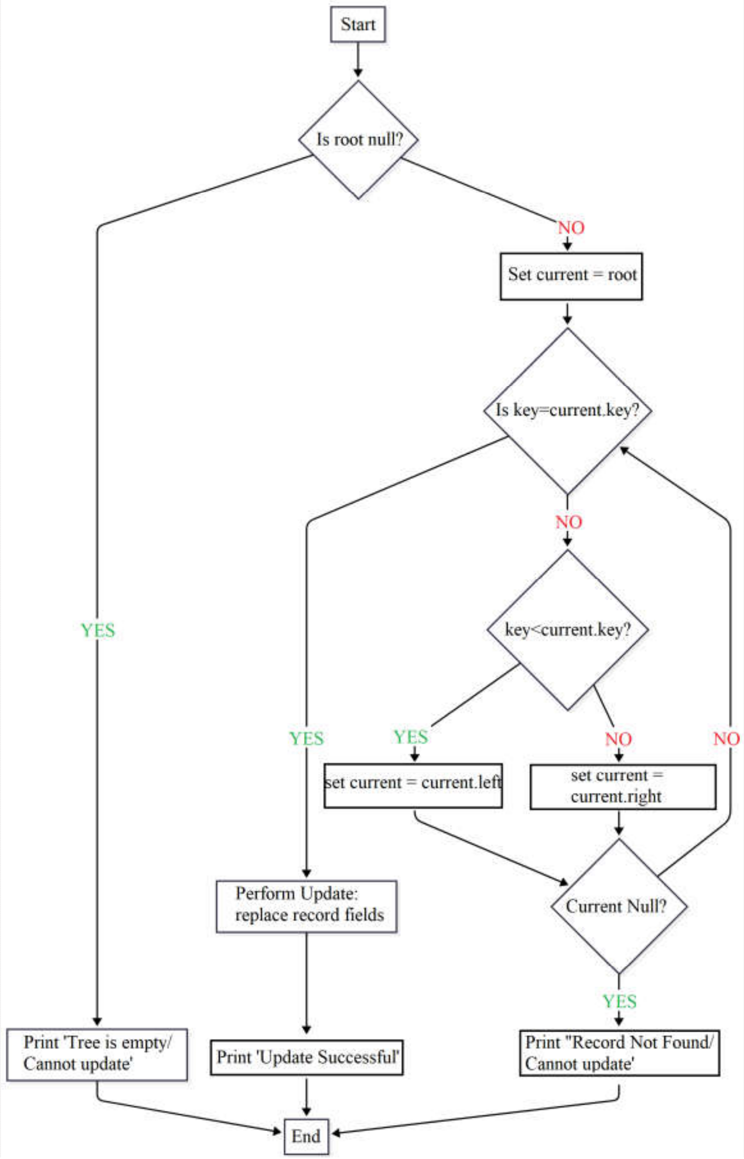
- Compare key
- Search left/right subtree
- Return record or “Not Found



FLOW CHART: UPDATE RECORD

Steps:

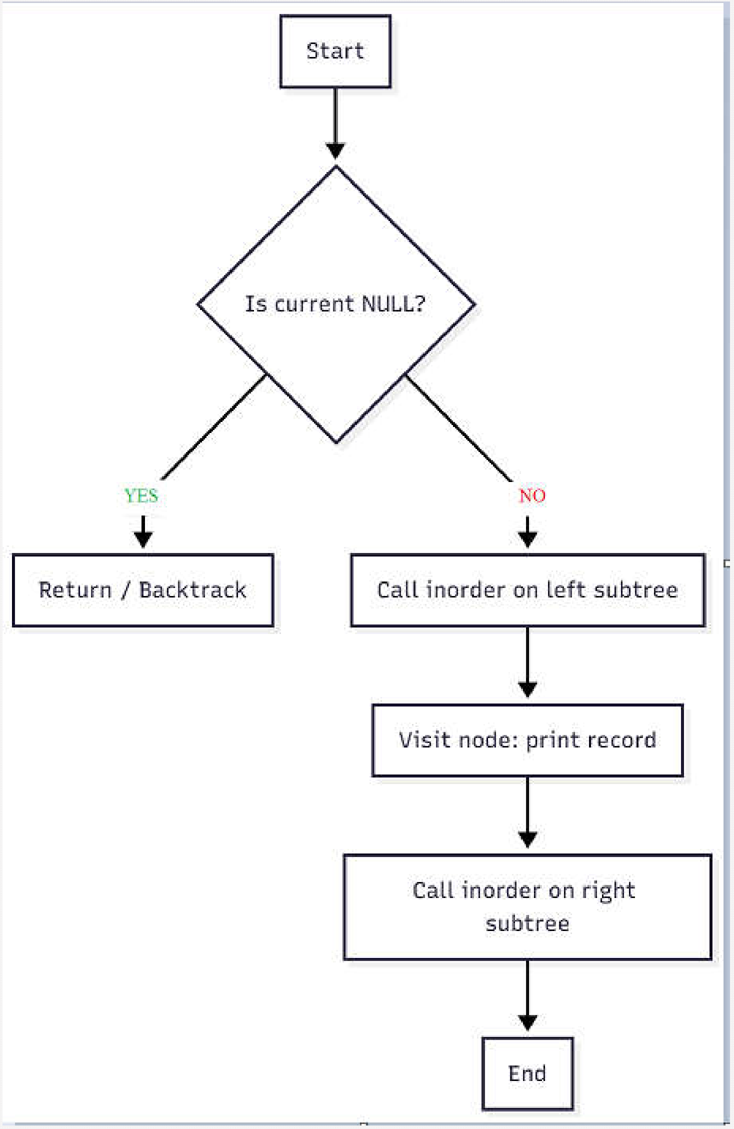
- Search for key
- Replace record attributes
- Print updated message


Start
Is root null?
NO
Set current = root
Is key=current.key?
NO
key<current.key?
YES
YES
YES
NO
NO
set current = current.left
set current = current.right
Current Null?
YES
Perform Update: replace record fields
Print 'Tree is empty/ Cannot update'
Print 'Update Successful'
Print "Record Not Found/ Cannot update'
End

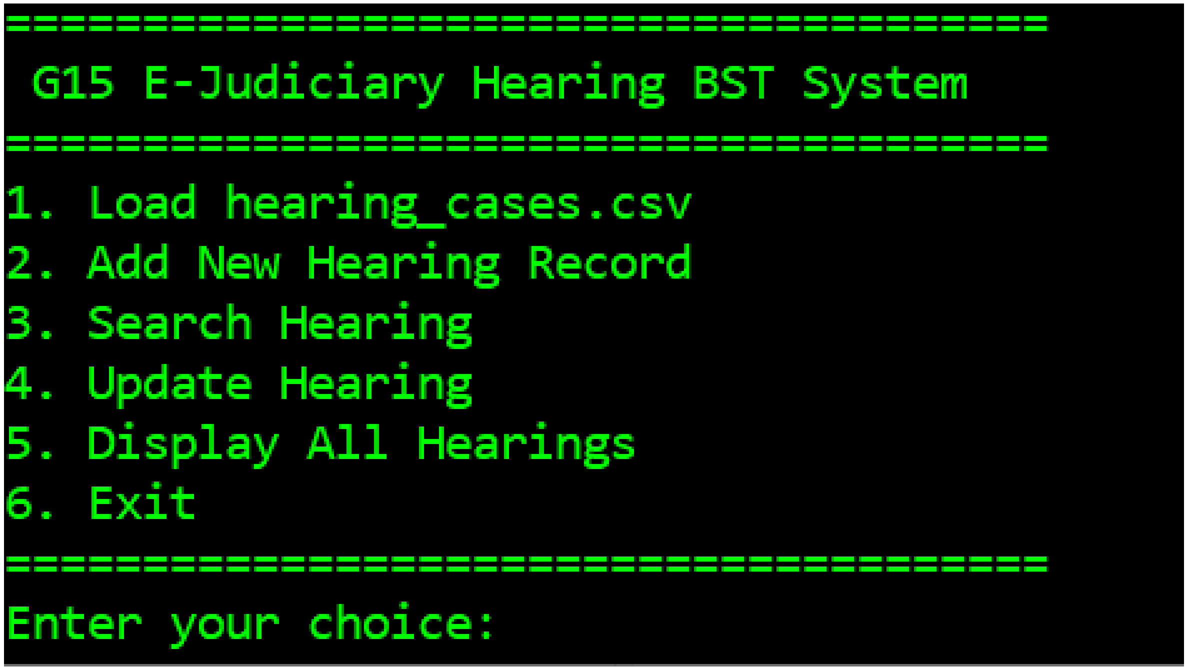
FLOW CHART: TRAVERSE (INORDER)

Steps:

- Left → Node → Right
- Displays sorted hearing list

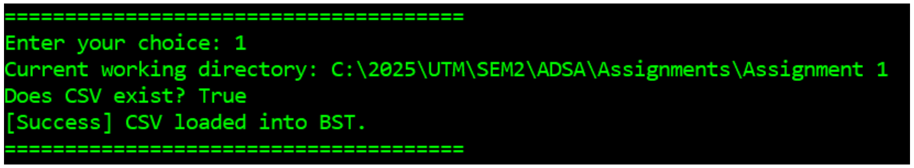


SYSTEM DEMO

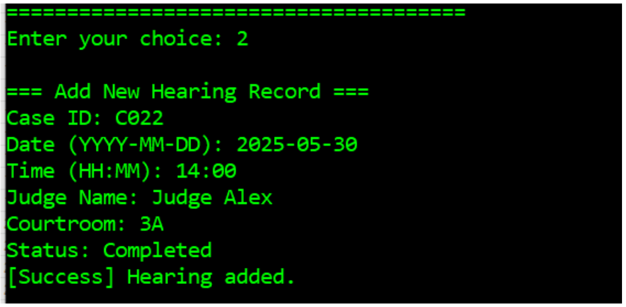


```
======================================
 G15 E-Judiciary Hearing BST System
======================================
1. Load hearing_cases.csv
2. Add New Hearing Record
3. Search Hearing
4. Update Hearing
5. Display All Hearings
6. Exit
======================================
Enter your choice:
```

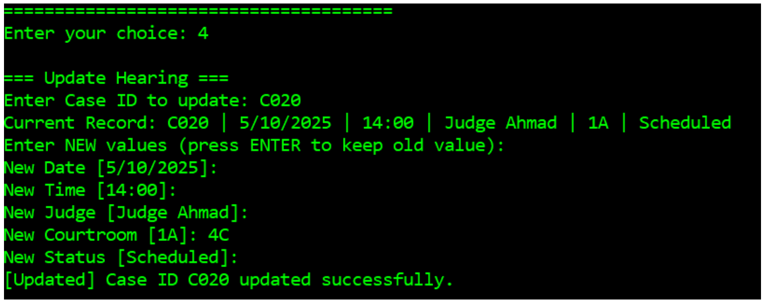
EXAMPLE OUTPUT



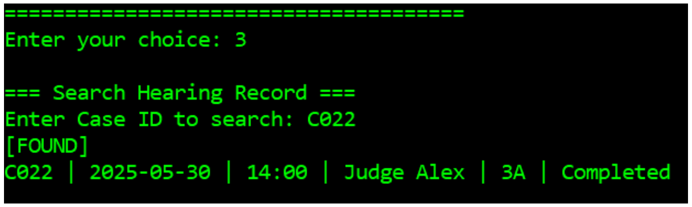
```
======================================
Enter your choice: 1
Current working directory: C:\2025\UTM\SEM2\ADSA\Assignments\Assignment 1
Does CSV exist? True
[Success] CSV loaded into BST.
======================================
```



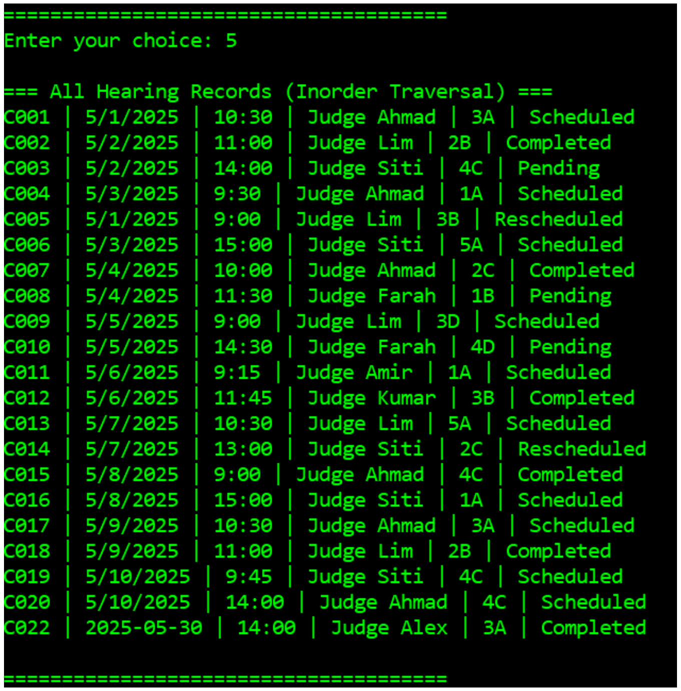
```
======================================
Enter your choice: 2

=== Add New Hearing Record ===
Case ID: C022
Date (YYYY-MM-DD): 2025-05-30
Time (HH:MM): 14:00
Judge Name: Judge Alex
Courtroom: 3A
Status: Completed
[Success] Hearing added.
```



```
======================================
Enter your choice: 4

=== Update Hearing ===
Enter Case ID to update: C020
Current Record: C020 | 5/10/2025 | 14:00 | Judge Ahmad | 1A | Scheduled
Enter NEW values (press ENTER to keep old value):
New Date [5/10/2025]:
New Time [14:00]:
New Judge [Judge Ahmad]:
New Courtroom [1A]: 4C
New Status [Scheduled]:
[Updated] Case ID C020 updated successfully.
```



```
======================================
Enter your choice: 3

=== Search Hearing Record ===
Enter Case ID to search: C022
[FOUND]
C022 | 2025-05-30 | 14:00 | Judge Alex | 3A | Completed
```



```
=====================================
Enter your choice: 5

=== All Hearing Records (Inorder Traversal) ===
C001 | 5/1/2025 | 10:30 | Judge Ahmad | 3A | Scheduled
C002 | 5/2/2025 | 11:00 | Judge Lim | 2B | Completed
C003 | 5/2/2025 | 14:00 | Judge Siti | 4C | Pending
C004 | 5/3/2025 | 9:30 | Judge Ahmad | 1A | Scheduled
C005 | 5/1/2025 | 9:00 | Judge Lim | 3B | Rescheduled
C006 | 5/3/2025 | 15:00 | Judge Siti | 5A | Scheduled
C007 | 5/4/2025 | 10:00 | Judge Ahmad | 2C | Completed
C008 | 5/4/2025 | 11:30 | Judge Farah | 1B | Pending
C009 | 5/5/2025 | 9:00 | Judge Lim | 3D | Scheduled
C010 | 5/5/2025 | 14:30 | Judge Farah | 4D | Pending
C011 | 5/6/2025 | 9:15 | Judge Amir | 1A | Scheduled
C012 | 5/6/2025 | 11:45 | Judge Kumar | 3B | Completed
C013 | 5/7/2025 | 10:30 | Judge Lim | 5A | Scheduled
C014 | 5/7/2025 | 13:00 | Judge Siti | 2C | Rescheduled
C015 | 5/8/2025 | 9:00 | Judge Ahmad | 4C | Completed
C016 | 5/8/2025 | 15:00 | Judge Siti | 1A | Scheduled
C017 | 5/9/2025 | 10:30 | Judge Ahmad | 3A | Scheduled
C018 | 5/9/2025 | 11:00 | Judge Lim | 2B | Completed
C019 | 5/10/2025 | 9:45 | Judge Siti | 4C | Scheduled
C020 | 5/10/2025 | 14:00 | Judge Ahmad | 4C | Scheduled
C022 | 2025-05-30 | 14:00 | Judge Alex | 3A | Completed

=====================================
```

LIMITATIONS

1. BST may become unbalanced with large datasets

2. No filtering by judge/date yet

3. No delete operation

4. No GUI (currently CLI only)

IMPROVEMENT

1. Main Improvement (Required for Task 3):

- Implement B-Tree (Solution 2) for better performance and balanced structure.

2. Additional enhancements:

- Date Range Search
- Search by judge or courtroom
- Save updated records back to CSV
- GUI version for user-friendly operation

CONCLUSION

- BST successfully implemented for Hearing Module
- All ADT operations demonstrated
- Flowcharts match code logic
- System runs correctly with CSV → BST → Search/Update
- Ready to expand to **B-Tree** in Task 3 for performance benchmarking

THANK YOU

30 Nov 2025

Evidence Record Module

Evidence record and search using BST (Binary Search Tree)

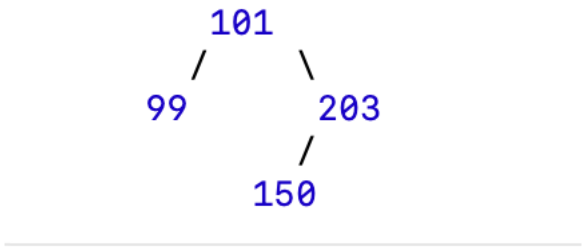
Evidence Tag

- To tag the evidence with meaningful information, the application mark information with the following:
    1. Evidence ID
    2. Case ID
    3. Type of documentation
    4. Date Input
    5. Description

BST method of search and insert in evidence

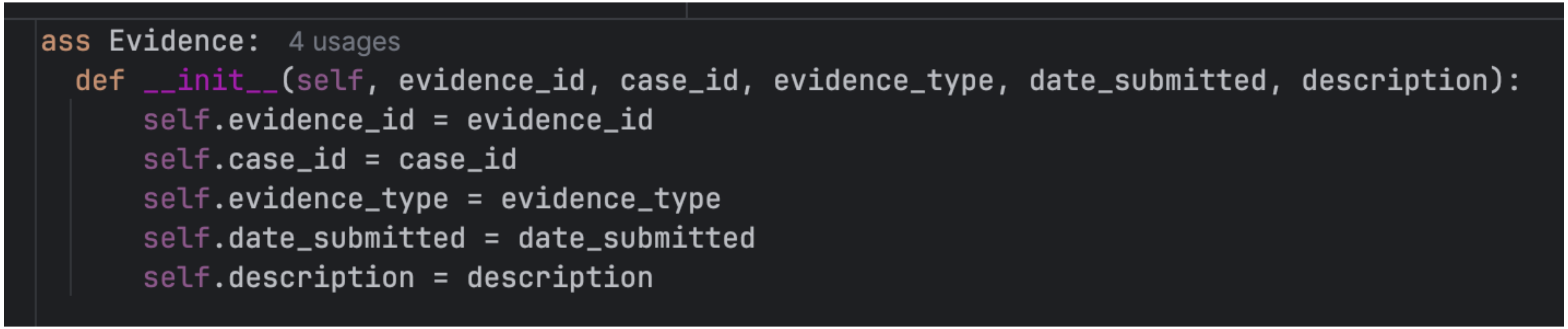
A binary search tree method is employing a pseudo binary search method by not sorting the index but build it in a tree.

For easy implementation, the evidence ID is set to be an integer data type and upon a new data input, the tree built by comparing the evidence ID. An example as below figure.



```
     101
    /   \
  99     203
         /
      150
```

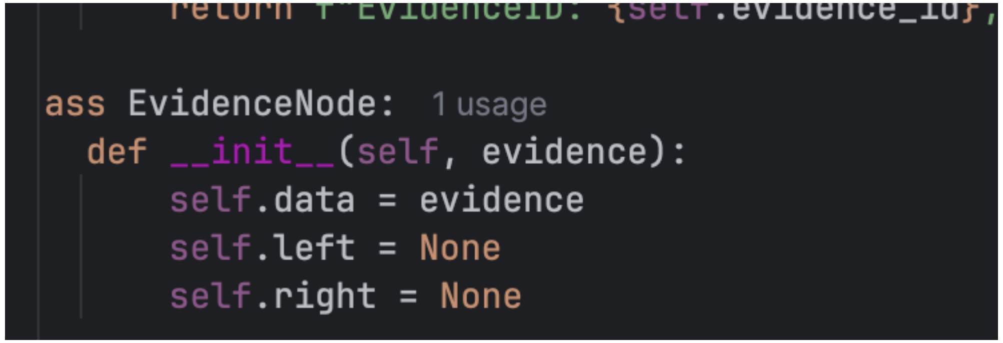
code



```
ass Evidence:  4 usages
  def __init__(self, evidence_id, case_id, evidence_type, date_submitted, description):
      self.evidence_id = evidence_id
      self.case_id = case_id
      self.evidence_type = evidence_type
      self.date_submitted = date_submitted
      self.description = description
```

Declare an evidence class for information storage

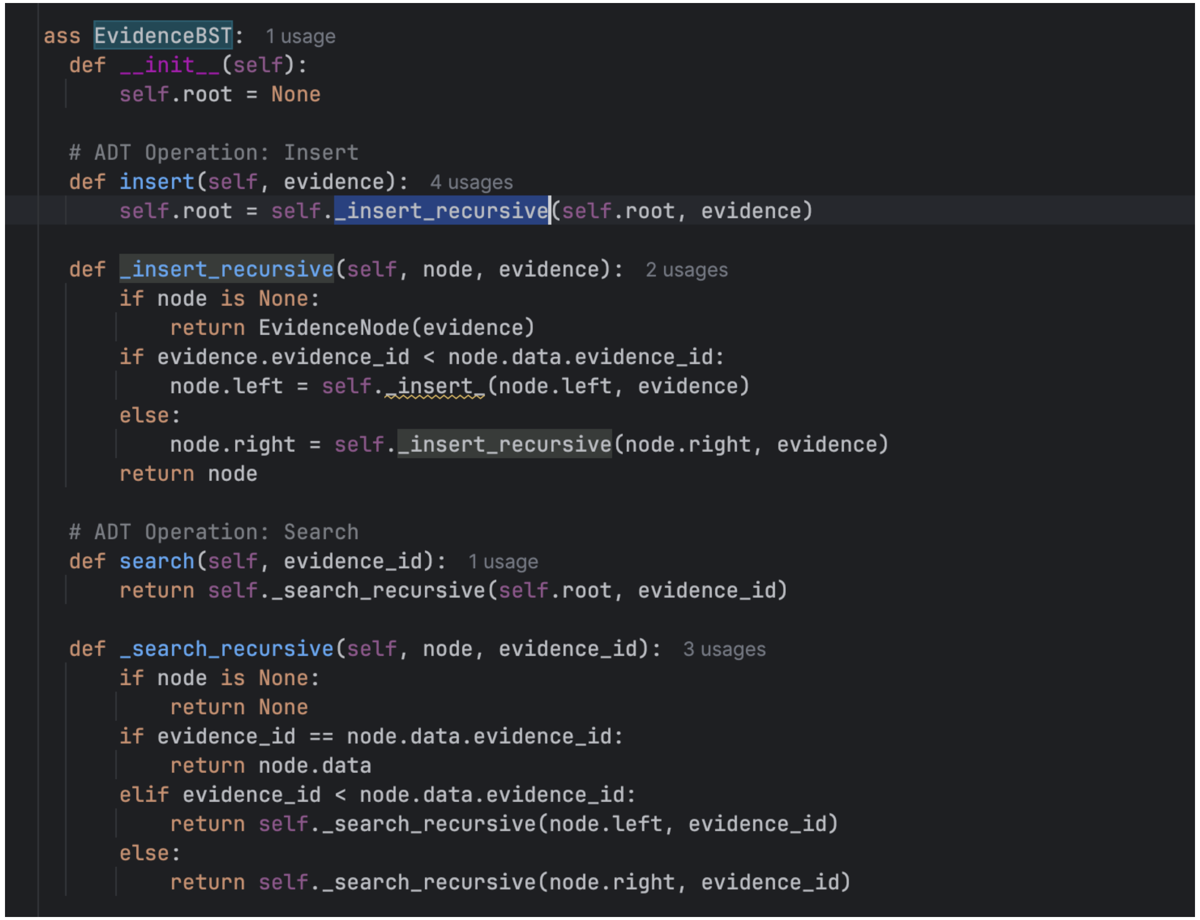
code



```
        return f"EvidenceID: {self.evidence_id},
ass EvidenceNode:  1 usage
    def __init__(self, evidence):
        self.data = evidence
        self.left = None
        self.right = None
```

Declare a Linked list for binary search tree, where the evidence is use as a data to be compared and define the left and right nodes

code



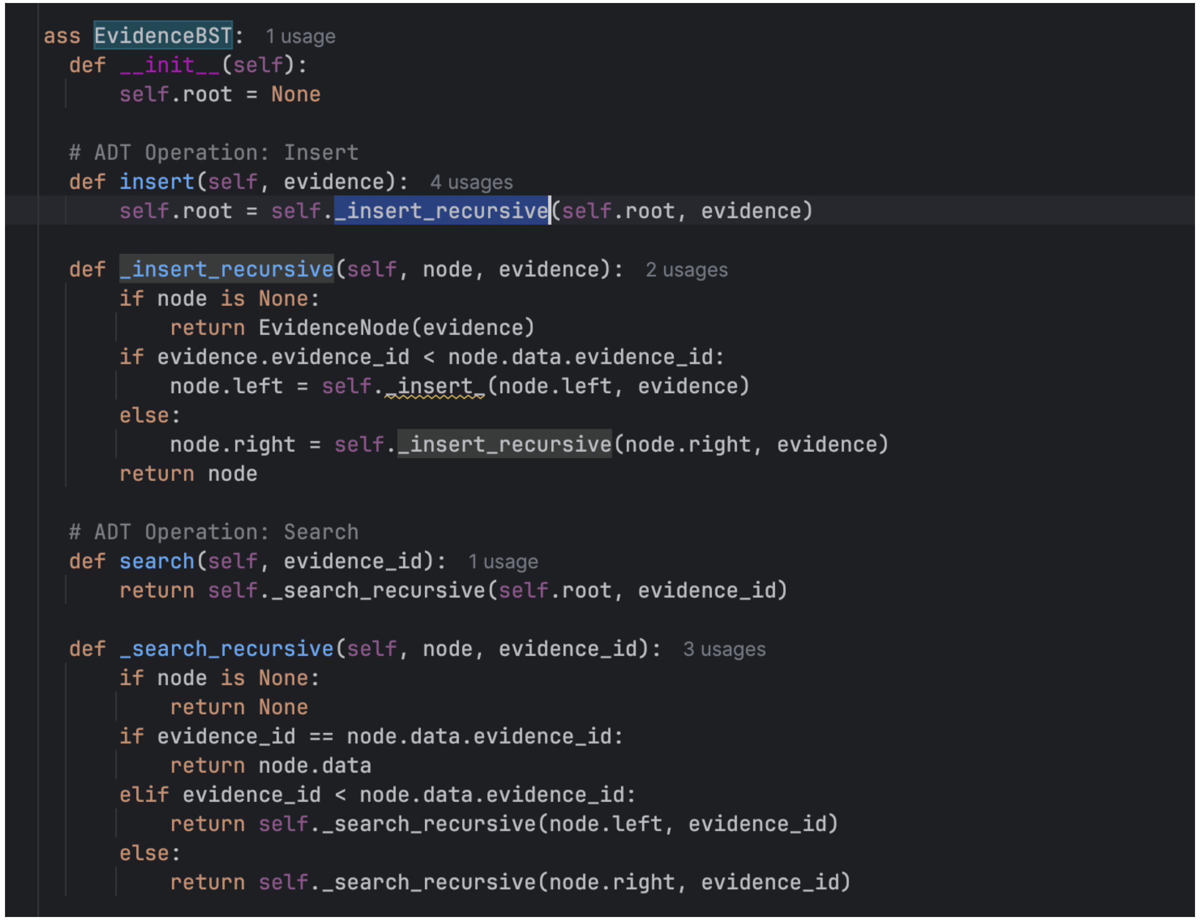
```
ass EvidenceBST:  1 usage
  def __init__(self):
      self.root = None

  # ADT Operation: Insert
  def insert(self, evidence):  4 usages
      self.root = self._insert_recursive(self.root, evidence)

  def _insert_recursive(self, node, evidence):  2 usages
      if node is None:
          return EvidenceNode(evidence)
      if evidence.evidence_id < node.data.evidence_id:
          node.left = self._insert_(node.left, evidence)
      else:
          node.right = self._insert_recursive(node.right, evidence)
      return node

  # ADT Operation: Search
  def search(self, evidence_id):  1 usage
      return self._search_recursive(self.root, evidence_id)

  def _search_recursive(self, node, evidence_id):  3 usages
      if node is None:
          return None
      if evidence_id == node.data.evidence_id:
          return node.data
      elif evidence_id < node.data.evidence_id:
          return self._search_recursive(node.left, evidence_id)
      else:
          return self._search_recursive(node.right, evidence_id)
```

1. Upon a first data comes in, the root of the node will be defined.

2. The following data will be insert whether left of right by comparing the root and the following node if the following node is not None.

3. The search algorithm employ a recursive search on the node by comparing the evidence ID.

code



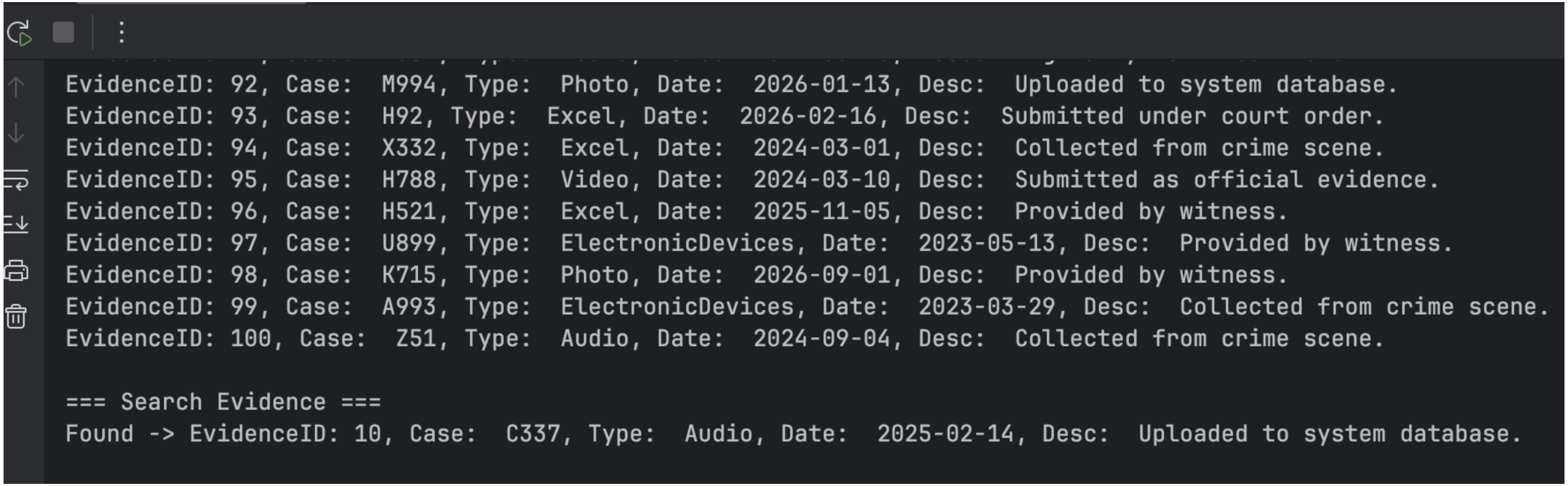
```
ass EvidenceBST:  1 usage
    def __init__(self):
        self.root = None

    # ADT Operation: Insert
    def insert(self, evidence):  4 usages
        self.root = self._insert_recursive(self.root, evidence)

    def _insert_recursive(self, node, evidence):  2 usages
        if node is None:
            return EvidenceNode(evidence)
        if evidence.evidence_id < node.data.evidence_id:
            node.left = self._insert_(node.left, evidence)
        else:
            node.right = self._insert_recursive(node.right, evidence)
        return node

    # ADT Operation: Search
    def search(self, evidence_id):  1 usage
        return self._search_recursive(self.root, evidence_id)

    def _search_recursive(self, node, evidence_id):  3 usages
        if node is None:
            return None
        if evidence_id == node.data.evidence_id:
            return node.data
        elif evidence_id < node.data.evidence_id:
            return self._search_recursive(node.left, evidence_id)
        else:
            return self._search_recursive(node.right, evidence_id)
```

1. Upon a first data comes in, the root of the node will be defined.

2. The following data will be insert whether left of right by comparing the root and the following node if the following node is not None.

3. The search algorithm employ a recursive search on the node by comparing the evidence ID.

Input simulated data and perform search



```
EvidenceID: 92, Case:  M994, Type:  Photo, Date:  2026-01-13, Desc:  Uploaded to system database.
EvidenceID: 93, Case:  H92, Type:  Excel, Date:  2026-02-16, Desc:  Submitted under court order.
EvidenceID: 94, Case:  X332, Type:  Excel, Date:  2024-03-01, Desc:  Collected from crime scene.
EvidenceID: 95, Case:  H788, Type:  Video, Date:  2024-03-10, Desc:  Submitted as official evidence.
EvidenceID: 96, Case:  H521, Type:  Excel, Date:  2025-11-05, Desc:  Provided by witness.
EvidenceID: 97, Case:  U899, Type:  ElectronicDevices, Date:  2023-05-13, Desc:  Provided by witness.
EvidenceID: 98, Case:  K715, Type:  Photo, Date:  2026-09-01, Desc:  Provided by witness.
EvidenceID: 99, Case:  A993, Type:  ElectronicDevices, Date:  2023-03-29, Desc:  Collected from crime scene.
EvidenceID: 100, Case:  Z51, Type:  Audio, Date:  2024-09-04, Desc:  Collected from crime scene.

=== Search Evidence ===
Found -> EvidenceID: 10, Case:  C337, Type:  Audio, Date:  2025-02-14, Desc:  Uploaded to system database.
```

Q&A