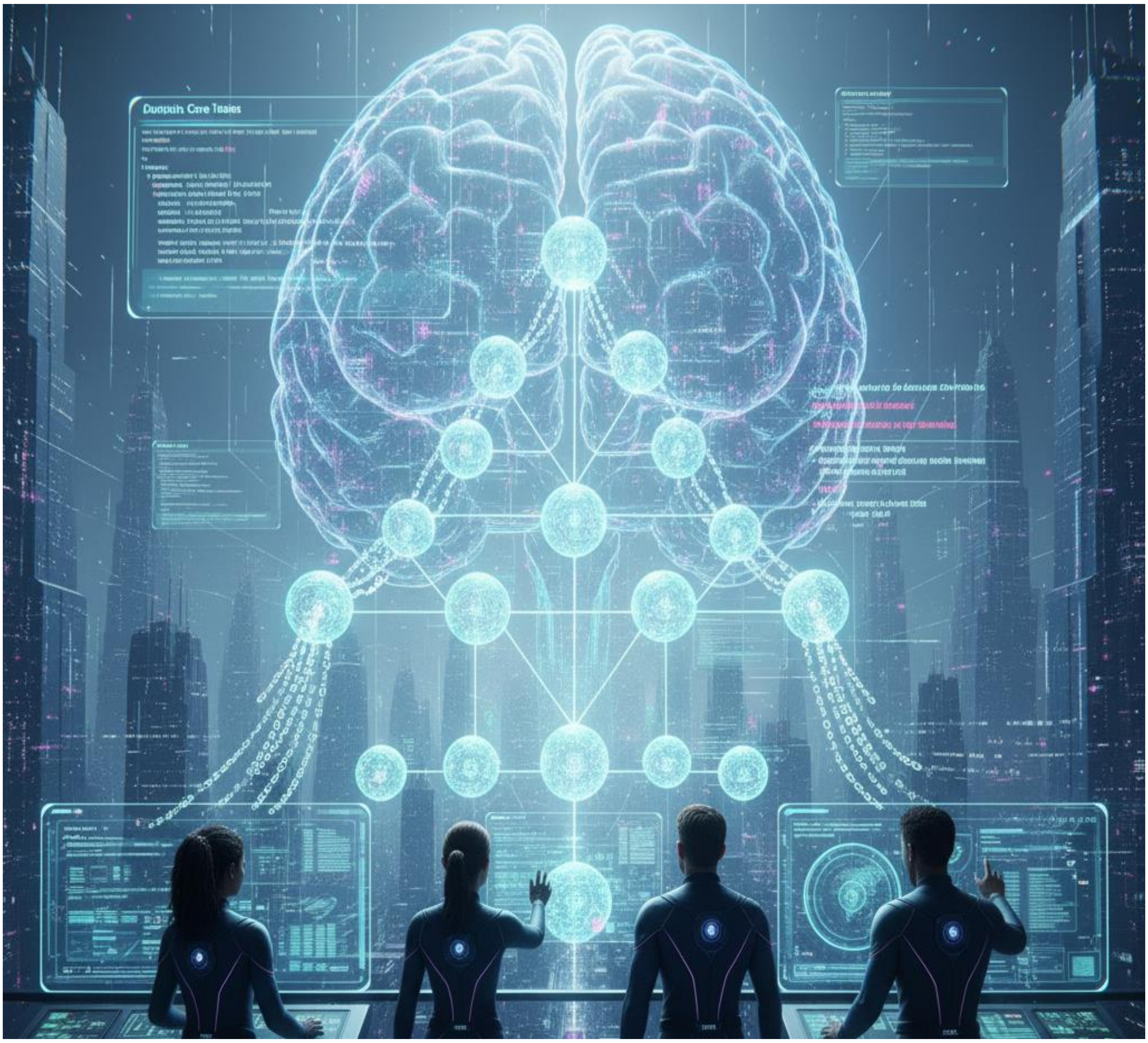


Neural Nexus Core AI

A course project for **CS210 Data Structures and Algorithms with Prof. Basit Qureshi**

Posted: Tuesday, 28 October 2025

Due: Saturday, 8 November 2025 (23:59)

Weight: 15 %

Test site: https://www.hackerrank.com/251cs210project

Submission on: https://lms.psu.edu.sa/

This is the year 2057. The age of artificial intelligence started decades ago; the autonomous warriors of the galaxy built the **Neural Nexus Core AI** that stands as the core of global knowledge; a vast digital brain connecting every document, image, and signal ever produced. It learns, adapts, and evolves every second, grows its knowledge and helps humanity in tremendous ways.

But such intelligence demands structure. Without balance and order, even the most powerful tools fall into chaos. You were chosen by the warriors of the galaxy and are now part of the **Index Core Team**, the architects responsible for maintaining the Nexus's **Cognitive Index**, the intelligent system that stores, searches, and synchronizes the world's information.

Your mission:

- Build the **Cognitive Index** using advanced data structures including Linked Lists, and AVL Trees.
- Your goal is to optimize speed, maintain balance, and ensure that every query keeps the **Neural Nexus Core AI** in perfect harmony.

You must create the **Cognitive Index**, a next generation search engine with the following powers:

1. **Index** the Archives: When new *archives* (text files) are added, you must read and store every *token* (word) and the *archives* they appear in. The index is your lifeblood — make it fast. Use a AVL Tree with an worst case runtime of O(log n).
2. Search the Lore: Allow visiting warriors to search for *tokens* (words or phrases). For each token query, list which *archive* contain the word and on which line.
3. Purge the Dangerous tokens: Occasionally, the warriors will ban a token. You must *remove* all traces of it from the index — as if it never existed.
4. List all tokens: *Traverse* through the archives to list the frequency of all the tokens. The warriors have decided to traverse using only the *post-order* method.

Your program reads input from the console. It reads the text file token by token. A token can be any word having removed any spaces or punctuation marks etc.

Your index is made of a *AVL* tree. Each AVL node (*AVLNode*) stores a String called *token*, an integer value called the *frequency*, i.e. the count of the number of times this token appears in all the *archives* (files); and a singly linked list (S*List*). The SList consists of Linked list nodes (*ListNode*) where each node contains the *filename* and *line-number* where the *token* is found. The following figure illustrates the data structure:

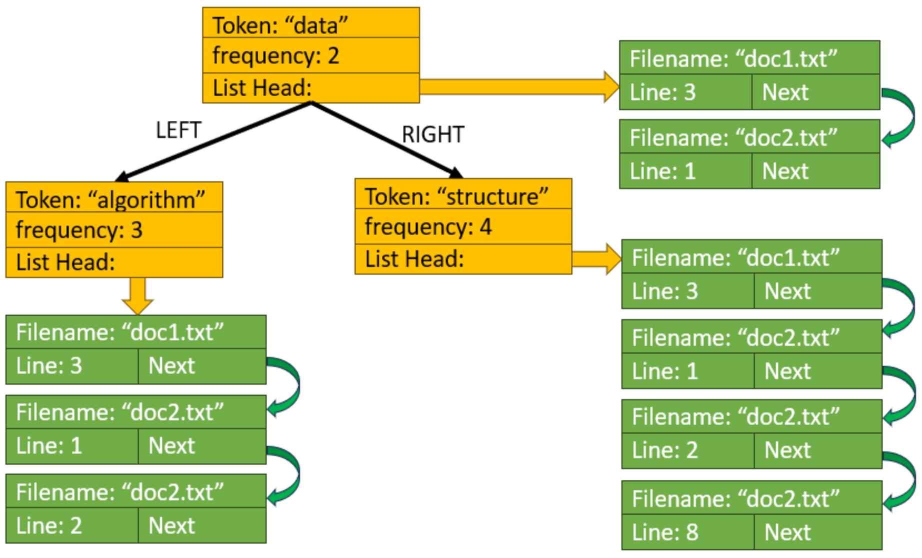


Figure 1: The “Cognitive Index” data structure is a AVL tree with embedded singly linked lists. Each AVLNode consists of a singly linked list.

Warning! The warriors requires that the Cognitive Index must be written in the scared Java language, however they do not trust the Collections Framework. You must avoid the use of Java Collections Framework at any cost.

Students are expected to use the following API to develop their software.

`Index`

`AVLNode root`
`int size`

`Index()`
`Insert(String)`
`Remove(String)`
`Search(String)`
`Traverse()`
`String toString()`
`//appropriate methods`

.

`AVLNode`

`String token`
`int height`
`int frequency`
`SList List`
`AVLNode left`
`AVLNode right`

`AVLNode()`
`//appropriate methods`

.

`SList`

`int size`
`Node head`

`Insert(Node)`
`Remove(Node)`
`Search(String s)`
`String toString()`
`//appropriate methods`

.

`Node`

`String filename`
`int lineNumber`
`Node next`

`//appropriate methods`

.

`Solution`

`main method`
`//appropriate methods for reading and writing`

Students are allowed to add/modify the API to suit their needs such as inclusion of setters and getters, however the attributes and methods names in the above classes must persist. In addition to these classes, students will also write a tester program that reads from the console, processes the input and produces the correct output for this software considering the following conditions.

Input Format

- A line starts with a integer Command followed by instructions.
- If the Command is **1**, it is followed by an integer S which represents the number of archives / files to read. For each Si, the line reads the filename and a integer value N, which represents the number of lines in the archive Si. Your program reads all this information in the data structure.
- If the Command is **2**, it is followed by a token T. You program searchers for token T in the data structure and prints a count P i.e. how many times did the token T appear in the data structure followed by P number of lines. Each Pj line consists of the filename and the line number where the token was found.
- If the Command is **3**, it is followed by a token T. You are required to remove all instances of T from the data structure.
- If the Command is **4**, you program prints the Post-order traversal of the binary tree.
- For any invalid input, the program prints -1.
- 1 <= Command <= 4
- 1 <= S <= 100000
- 1 <= P <= 100

Running your program

The following is a test run with explanation

Standard Input: Consider the following input:

```
1 3
File1.txt 3
data structures are cool
algorithms are fun
this is a cool project
File2.txt 2
search the data
time is bubble
File3.txt 1
data in data is fun
2 data
3 in
4
```

Standard Output: The above generates the following output

```
4
File1.txt 1
File2.txt 1
File3.txt 1
File3.txt 1
a algorithms bubble cool are fun is search the time this structure project data
```

Explanation

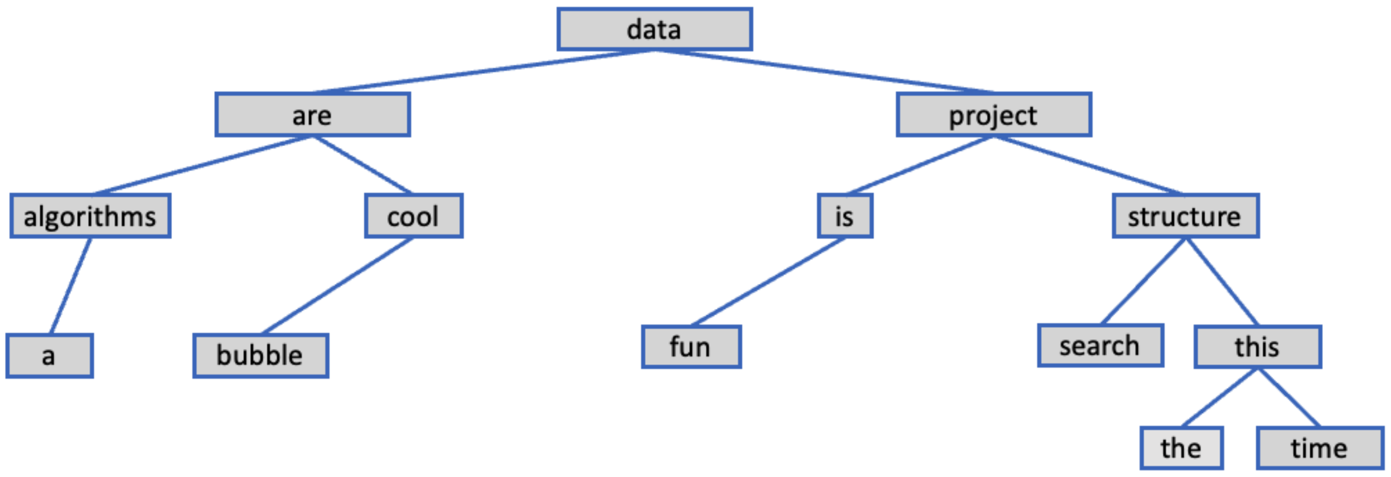
`1 3`
`File1.txt 3`
`data structures are cool`
`algorithms are fun`
`this is a cool project`
`File2.txt 2`
`search the data`
`time is bubble`
`File3.txt 1`
`data in data is fun`

The first line reads 1 3. 1 means insert. 3 means there are 3 files/scrolls to read.
The first file name is File1.txt and it has 3 lines of text.
`data structures are cool`
`algorithms are fun`
`this is a cool project`
 	
The next file name is File2.txt and it has 2 lines of text.
`search the data`
`time is bubble`
 	
The next file name is File3.txt and it has 1 line of text.
`data in data is fun`

2 data

2 data -> requires you to print all the occurrences of data. In the next line, you print 4 because data appears 4 times; followed by details for each occurrence (filename and line number).
 	
`4`
`File1.txt 1`
`File2.txt 1`
`File3.txt 1`
`File3.txt 1`

3 in
3 are -> requires you to search in and remove it from the data-structure.
4
4 -> requires you to print the Post-order traversal of this tree. **At this moment the tree looks like this:**



Note: Node with (in) is removed
Postorder traversal of this tree is:

a algorithms bubble cool are fun is search the time this structure project data

Evaluation:

You are allowed to work as a group with maximum 2 members in a group. Your work's evaluation would be based the following criterion.

Step 1: Verify correctness of your program on Hackerrank

You have unlimited chances to submit your code to hackerrank before the deadline. When you submit your code, the hackerrank platform tests your program against pre-built (hidden) testcases and grades your work. You need to fix all the mistakes in the project and try to earn a full score. For this project, there are no limitations on time and memory usage.

Step 2: Writing Report

Complete a report consisting of the following sections:

1. **Cover:** Use the coversheet. Add personal details and a screenshot from Hackerrank showing your score.
2. **Introduction**: Explain why you think the data structure you implemented is a good choice for implementing the Index.
3. **Performance**:
    a. Give the runtime of your program using big Oh notation.
    b. Give the best case and worst case scenario using big Oh notation.
    c. Compare *Index* data structure to a Doubly Linked list.
    d. Compare *Index* data structure to a BST Tree.
4. **Conclusion**: Give reasons why or why not the *Index* data structure is better than a doubly linked list or a BST implementation.
5. **Code**: Copy paste all of your java code. This code must be identical to the one submitted to Hackerrank.

Step 3: Upload your final submission to LMS. The submission will consist of

- One Microsoft word file consisting of your report. The code must have the following structure.

```
public class Solution {
}
class Index {…}
class BSTNode {…}
class SList {…}
class Node {…}
```

Interview:

Student would be requested to demonstrate their work. The instructor may ask the student to modify the code to satisfy any test-case(s) there in.

Evaluation and Grading:

- Correctness of program verified on hackerrank - 5 points
- Completeness of report – 5 points

- Interview – 5 points
- Total = 15 points

Warning: Not allowed to use java collections framework or any of its classes such as ArrayList etc.

The instructor reserves the right to determine the scores of each test case. Test-cases will be posted on hackerrank, students will have unlimited number of opportunities to post and test their project until the due date. **The system will not take any submissions after the due date.**

Code Inspection and plagiarism:

The code would be inspected by the instructor. The instructor would use the MOSS tool (https://theory.stanford.edu/~aiken/moss/) to determine the originality of submission.

If the code similarity is above 40%, the students would earn ZERO score on the project. This includes *all* group members for all teams involved as well (i.e. all other groups with similar code)

Submission Dead-Line:

The submission deadline is final. Late Submissions will be awarded ZERO points.

Important Notes:

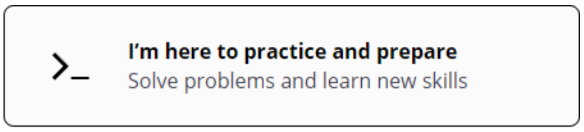
- It is the student's responsibility to check/test/verify/debug the code before submission.
- It is the student's responsibility to check/test/verify all submitted work (including jar files)
- It is the student's responsibility to verify that all files have been uploaded to the LMS.
- Incomplete or wrong file types that do not execute will NOT be graded.
- For each project, instructor will provide a few sample test-cases to verify the execution of your program.
- After an assignment/project has been graded, re-submission with an intention to improve an assignments score **will not be allowed**.

TUTORIAL:

Submission on Hacker Rank

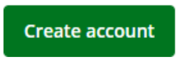
Step 1. Register on https://www.hackerrank.com/

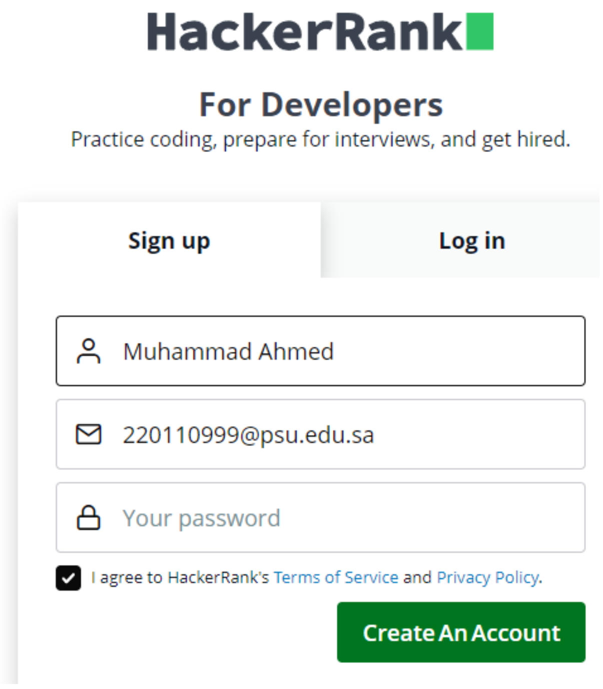
Make sure your username as your PSU ID number.



I'm here to practice and prepare

Solve problems and learn new skills





For Developers

Practice coding, prepare for interviews, and get hired.

Muhammad Ahmed

220110999@psu.edu.sa

Your password

I agree to HackerRank's Terms of Service and Privacy Policy.

2. Join contest "251CS210project"

Start working on your project.