Competitive programming

Assignment-05

2303A51463

Batch-07

1: Building and Utilizing Tries for String Problems

Problem: Library Book Title Search Using Trie

A digital library stores a list of book titles.

When a student types the first few letters of a title, the system immediately displays matching books.

Since many book titles begin with common prefixes, the library uses a Trie (Prefix Tree) to efficiently:

• Store book titles

• Search for a complete title

• Check whether any title starts with a given prefix

Task

Write a program using a Trie to support:

• Insert operation: Add a book title

• Search operation: Check whether a book title exists

• Prefix query: Check whether any book title starts with a given prefix

Example Test Case 3

Input

• Number of book titles: 5

• Book titles:

• ["Data", "Database", "DataScience", "Design", "Development"]

Operations:

1. Search for book "Data"

2. Search for book "DataMining"

3. Check prefix "Data"

4. Check prefix "Dev"

Output

• Book "Data" found → True

• Book "DataMining" not found → False

• Prefix "Data" exists → True

• Prefix "Dev" exists → True

Explanation

Book titles inserted into the Trie

Data, Database, DataScience, Design, Development

Conceptual Trie Structure

26th Feb,

2025

5:00PM

(root)

/ \

D (other)

|

a

|

t

|

a*

/ \

b S

| |

a c

| |

s i

| |

e e

|

(end)

From 'D'

|

e

|

s

|

i

|

g

|

n*

|

v

|

e

|

l

|

o

|

p

|

m

|

e

|

n

|

t*

(* indicates end of a complete word)

Operation 1: Search "Data"

Characters D → a → t → a exist

End-of-word marker is present

Book found

Operation 2: Search "DataMining"

Characters D → a → t → a exist

Character M does not continue the path

Book not found

Operation 3: Prefix "Data"

Characters D → a → t → a exist

Matching titles:

• Data

• Database

• DataScience

Prefix exists

Operation 4: Prefix "Dev"

Characters D → e → v exist

Matching title:

• Development

Prefix exists

Algorithm:

```
1. Set current = root.
2. For each character c in title:
```

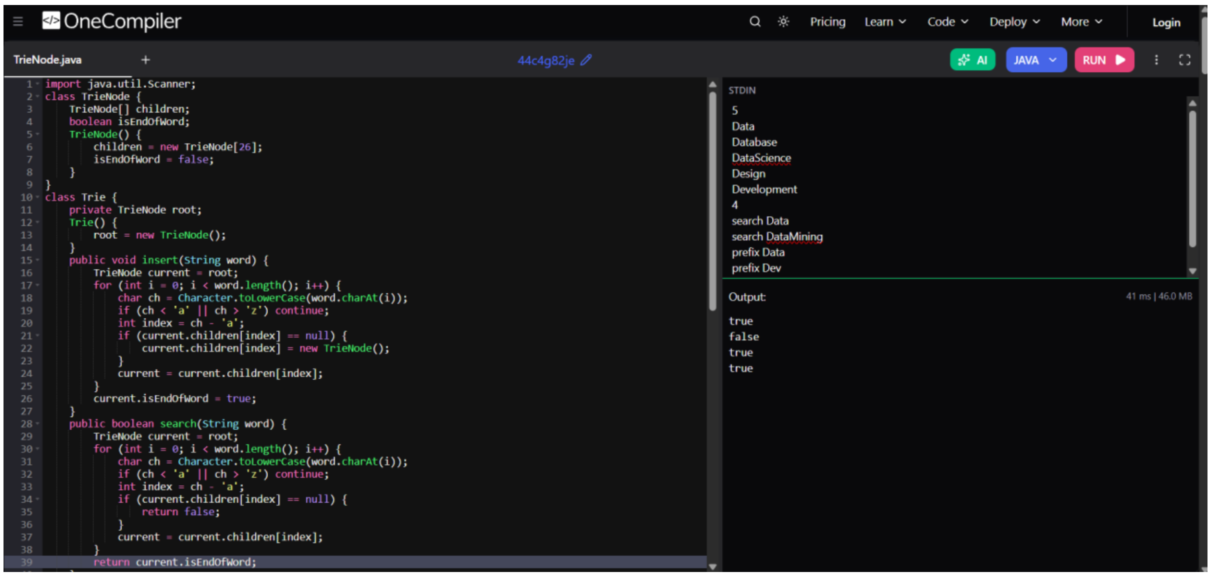
- Convert c to lowercase.
- Compute index = c - 'a'.
- If current.children[index] is NULL:
    - Create a new Trie node there.
- Move current to current.children[index].

3.After the last character:

- Mark current.isEndOfWord = true.

Code:

```
import java.util.Scanner;
class TrieNode {
   TrieNode[] children;
   boolean isEndOfWord;
   TrieNode() {
      children = new TrieNode[26];
      isEndOfWord = false;
   }
}
class Trie {
   private TrieNode root;
   Trie() {
      root = new TrieNode();
   }
   public void insert(String word) {
      TrieNode current = root;
      for (int i = 0; i < word.length(); i++) {
         char ch = Character.toLowerCase(word.charAt(i));
         if (ch < 'a' || ch > 'z') continue;
         int index = ch - 'a';
         if (current.children[index] == null) {
            current.children[index] = new TrieNode();
        }
        current = current.children[index];
    }
    current.isEndOfWord = true;
}
public boolean search(String word) {
    TrieNode current = root;
    for (int i = 0; i < word.length(); i++) {
        char ch = Character.toLowerCase(word.charAt(i));
        if (ch < 'a' || ch > 'z') continue;
        int index = ch - 'a';
        if (current.children[index] == null) {
            return false;
        }
        current = current.children[index];
    }
    return current.isEndOfWord;
}
public boolean startsWith(String prefix) {
    TrieNode current = root;
    for (int i = 0; i < prefix.length(); i++) {
        char ch = Character.toLowerCase(prefix.charAt(i));
        if (ch < 'a' || ch > 'z') continue;
        int index = ch - 'a';
        if (current.children[index] == null) {
            return false;
        }
        current = current.children[index];
        }
        return true;
    }
}
public class Main {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        Trie trie = new Trie();
        int n = sc.nextInt();
        sc.nextLine();
        for (int i = 0; i < n; i++) {
            String title = sc.nextLine();
            trie.insert(title);
        }
        int q = sc.nextInt();
        sc.nextLine();
        for (int i = 0; i < q; i++) {
            String line = sc.nextLine();
            String[] parts = line.split(" ");

            String operation = parts[0];
            String value = parts[1];

            if (operation.equalsIgnoreCase("search")) {
                System.out.println(trie.search(value));
            }
            else if (operation.equalsIgnoreCase("prefix")) {
                System.out.println(trie.startsWith(value));
            }
        }
        sc.close();
    }
}
```



TrieNode.java

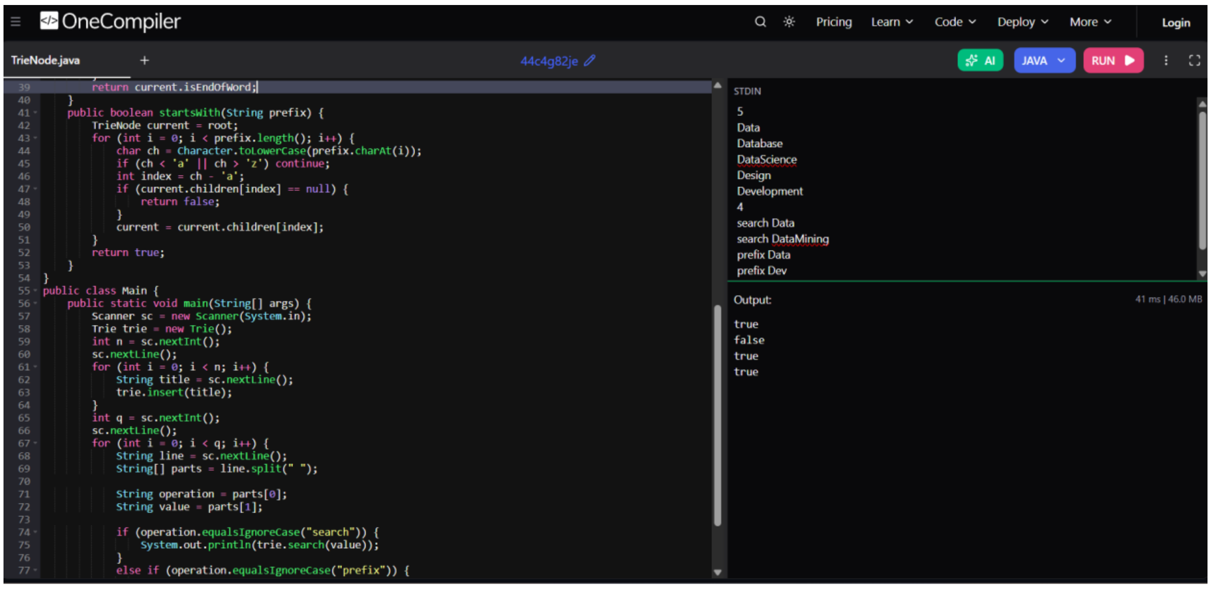
```
import java.util.Scanner;
class TrieNode {
    TrieNode[] children;
    boolean isEndOfWord;
    TrieNode() {
        children = new TrieNode[26];
        isEndOfWord = false;
    }
}
class Trie {
    private TrieNode root;
    Trie() {
        root = new TrieNode();
    }
    public void insert(String word) {
        TrieNode current = root;
        for (int i = 0; i < word.length(); i++) {
            char ch = Character.toLowerCase(word.charAt(i));
            if (ch < 'a' || ch > 'z') continue;
            int index = ch - 'a';
            if (current.children[index] == null) {
                current.children[index] = new TrieNode();
            }
            current = current.children[index];
        }
        current.isEndOfWord = true;
    }
    public boolean search(String word) {
        TrieNode current = root;
        for (int i = 0; i < word.length(); i++) {
            char ch = Character.toLowerCase(word.charAt(i));
            if (ch < 'a' || ch > 'z') continue;
            int index = ch - 'a';
            if (current.children[index] == null) {
                return false;
            }
            current = current.children[index];
        }
        return current.isEndOfWord;
```

STDIN

```
5
Data
Database
DataScience
Design
Development
4
search Data
search DataMining
prefix Data
prefix Dev
```

Output: 41 ms | 46.0 MB

```
true
false
true
true
```



TrieNode.java

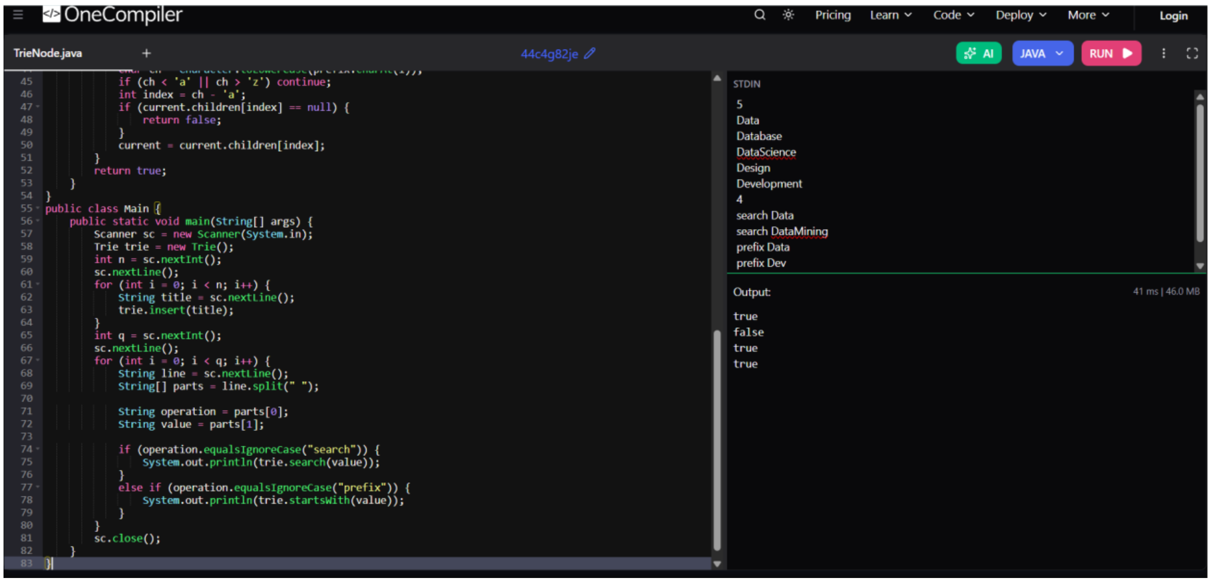
```
        return current.isEndOfWord;
    }
    public boolean startsWith(String prefix) {
        TrieNode current = root;
        for (int i = 0; i < prefix.length(); i++) {
            char ch = Character.toLowerCase(prefix.charAt(i));
            if (ch < 'a' || ch > 'z') continue;
            int index = ch - 'a';
            if (current.children[index] == null) {
                return false;
            }
            current = current.children[index];
        }
        return true;
    }
}
public class Main {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        Trie trie = new Trie();
        int n = sc.nextInt();
        sc.nextLine();
        for (int i = 0; i < n; i++) {
            String title = sc.nextLine();
            trie.insert(title);
        }
        int q = sc.nextInt();
        sc.nextLine();
        for (int i = 0; i < q; i++) {
            String line = sc.nextLine();
            String[] parts = line.split(" ");

            String operation = parts[0];
            String value = parts[1];

            if (operation.equalsIgnoreCase("search")) {
                System.out.println(trie.search(value));
            }
            else if (operation.equalsIgnoreCase("prefix")) {
```

STDIN

```
5
Data
Database
DataScience
Design
Development
4
search Data
search DataMining
prefix Data
prefix Dev
```

Output: 41 ms | 46.0 MB

```
true
false
true
true
```



```java
            if (ch < 'a' || ch > 'z') continue;
            int index = ch - 'a';
            if (current.children[index] == null) {
                return false;
            }
            current = current.children[index];
        }
        return true;
    }
}
public class Main {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        Trie trie = new Trie();
        int n = sc.nextInt();
        sc.nextLine();
        for (int i = 0; i < n; i++) {
            String title = sc.nextLine();
            trie.insert(title);
        }
        int q = sc.nextInt();
        sc.nextLine();
        for (int i = 0; i < q; i++) {
            String line = sc.nextLine();
            String[] parts = line.split(" ");

            String operation = parts[0];
            String value = parts[1];

            if (operation.equalsIgnoreCase("search")) {
                System.out.println(trie.search(value));
            }
            else if (operation.equalsIgnoreCase("prefix")) {
                System.out.println(trie.startsWith(value));
            }
        }
        sc.close();
    }
}
```

STDIN

```
5
Data
Database
DataScience
Design
Development
4
search Data
search DataMining
prefix Data
prefix Dev
```

Output: 41 ms | 46.0 MB

```
true
false
true
true
```

2.Problem: Mobile Contact Name Search Using Trie

Problem Statement

A mobile phone stores a list of contact names.

When a user types the first few letters of a contact name, the phone should instantly check whether matching contacts exist.

Since many contact names share common prefixes, the mobile phone uses a Trie (Prefix Tree) to efficiently:

• Store contact names

• Search for a complete contact name

• Check whether any contact name starts with a given prefix

Task

Write a program using a Trie to support the following operations:

• Insert Operation

Add a contact name to the Trie.

• Search Operation

Check whether a complete contact name exists in the Trie.

• Prefix Query Operation

Check whether any contact name starts with a given prefix.

Example Test Case

Input

• Number of contact names: 5

• Contact names:

["Anil", "Anita", "Anand", "Suresh", "Sunil"]

Operations:

1. Search for contact "Anil"

2. Search for contact "Anitha"

3. Check prefix "Ani"

4. Check prefix "Su"

Output

• Contact "Anil" found → True

• Contact "Anitha" not found → False

• Prefix "Ani" exists → True

• Prefix "Su" exists → True

Explanation

Contact Names Inserted into the Trie

• Anil

• Anita

• Anand

• Suresh

• Sunil

Conceptual Trie Structure

(root)

/ \

A S

| |

n u

/ \ \

```
i a r
/ \ | \
l* t* n e
| |
d* s
|
h*
```

(from S → u → n → i → l*)

* indicates the end of a complete contact name.

Operation-wise Explanation

Operation 1: Search "Anil"

• Characters followed: A → n → i → l

• End-of-word marker found

Contact found

Operation 2: Search "Anitha"

• Characters A → n → i → t exist

• Character h does not continue the Trie path

Contact not found

Operation 3: Prefix Query "Ani"

• Characters A → n → i exist

• Matching contacts:

o Anil

o Anita

Prefix exists

Operation 4: Prefix Query "Su"

• Characters S → u exist

• Matching contacts:

o Suresh

o Sunil

Prefix exists

Algorithm 1: INSERT(ContactName)

Input: String name

Steps:

1. Set current = root.
2. For each character c in name:
    - Convert to lowercase.
    - index = c - 'a'
    - If current.children[index] is NULL:
        - Create a new node.
    - Move current to that child.
3. After the last character:
    - Mark current.isEndOfWord = true.

Algorithm 2: SEARCH(ContactName)

Input: String name
Output: True if contact exists, else False

Steps:

1. Start at root.
2. For each character:
    - Convert to lowercase.
    - Compute index.
    - If child does not exist → return False.
3. After traversal:
    - If isEndOfWord is true → return True
    - Else → return False.

Algorithm 3: PREFIX QUERY

Input: String prefix
Output: True if any contact starts with prefix, else False

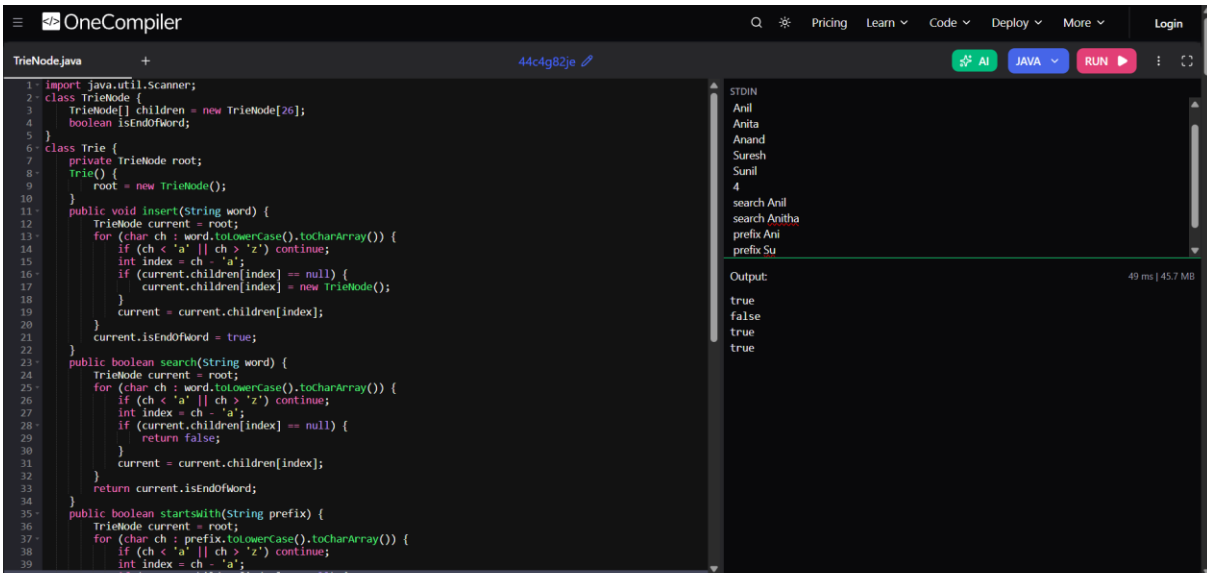
Steps:

1. Start at root.
2. Traverse characters of prefix.
3. If any character path is missing → return False.
4. If all matched → return True.

Code:

```
import java.util.Scanner;

class TrieNode {
   TrieNode[] children = new TrieNode[26];
   boolean isEndOfWord;
}

class Trie {
   private TrieNode root;
   Trie() {
      root = new TrieNode();
   }
   public void insert(String word) {
      TrieNode current = root;
      for (char ch : word.toLowerCase().toCharArray()) {
         if (ch < 'a' || ch > 'z') continue;
         int index = ch - 'a';
         if (current.children[index] == null) {
            current.children[index] = new TrieNode();
         }
         current = current.children[index];
      }
```

```
        current.isEndOfWord = true;
    }
    public boolean search(String word) {
        TrieNode current = root;
        for (char ch : word.toLowerCase().toCharArray()) {
            if (ch < 'a' || ch > 'z') continue;
            int index = ch - 'a';
            if (current.children[index] == null) {
                return false;
            }
            current = current.children[index];
        }
        return current.isEndOfWord;
    }
    public boolean startsWith(String prefix) {
        TrieNode current = root;
        for (char ch : prefix.toLowerCase().toCharArray()) {
            if (ch < 'a' || ch > 'z') continue;
            int index = ch - 'a';
            if (current.children[index] == null) {
                return false;
            }
            current = current.children[index];
        }
        return true;
    }
}
public class Main {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        Trie trie = new Trie();
        int n = sc.nextInt();
        sc.nextLine();
        for (int i = 0; i < n; i++) {
            String name = sc.nextLine();
            trie.insert(name);
        }
        int q = sc.nextInt();
        sc.nextLine();
        for (int i = 0; i < q; i++) {
            String[] parts = sc.nextLine().split(" ");
            String operation = parts[0];
            String value = parts[1];
            if (operation.equalsIgnoreCase("search")) {
                System.out.println(trie.search(value));
            }
            else if (operation.equalsIgnoreCase("prefix")) {
                System.out.println(trie.startsWith(value));
            }
        }
        sc.close();
    }
}
```



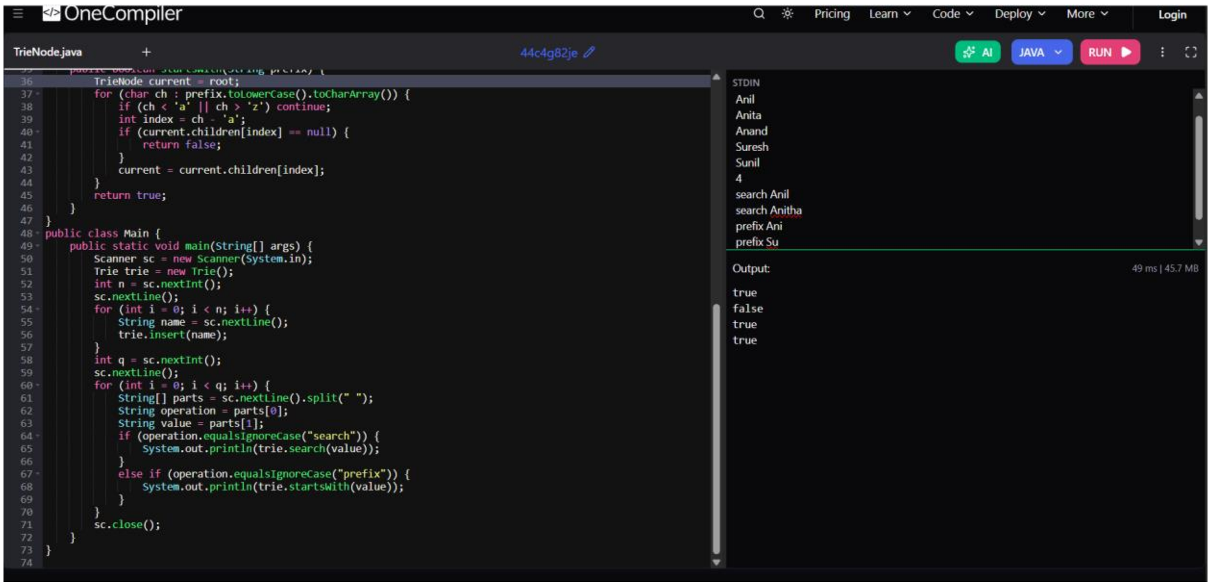
```java
import java.util.Scanner;
class TrieNode {
    TrieNode[] children = new TrieNode[26];
    boolean isEndOfWord;
}
class Trie {
    private TrieNode root;
    Trie() {
        root = new TrieNode();
    }
    public void insert(String word) {
        TrieNode current = root;
        for (char ch : word.toLowerCase().toCharArray()) {
            if (ch < 'a' || ch > 'z') continue;
            int index = ch - 'a';
            if (current.children[index] == null) {
                current.children[index] = new TrieNode();
            }
            current = current.children[index];
        }
        current.isEndOfWord = true;
    }
    public boolean search(String word) {
        TrieNode current = root;
        for (char ch : word.toLowerCase().toCharArray()) {
            if (ch < 'a' || ch > 'z') continue;
            int index = ch - 'a';
            if (current.children[index] == null) {
                return false;
            }
            current = current.children[index];
        }
        return current.isEndOfWord;
    }
    public boolean startsWith(String prefix) {
        TrieNode current = root;
        for (char ch : prefix.toLowerCase().toCharArray()) {
            if (ch < 'a' || ch > 'z') continue;
            int index = ch - 'a';
```

STDIN

```
Anil
Anita
Anand
Suresh
Sunil
4
search Anil
search Anitha
prefix Ani
prefix Su
```

Output: 49 ms | 45.7 MB

```
true
false
true
true
```



```java
        TrieNode current = root;
        for (char ch : prefix.toLowerCase().toCharArray()) {
            if (ch < 'a' || ch > 'z') continue;
            int index = ch - 'a';
            if (current.children[index] == null) {
                return false;
            }
            current = current.children[index];
        }
        return true;
    }
}
public class Main {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        Trie trie = new Trie();
        int n = sc.nextInt();
        sc.nextLine();
        for (int i = 0; i < n; i++) {
            String name = sc.nextLine();
            trie.insert(name);
        }
        int q = sc.nextInt();
        sc.nextLine();
        for (int i = 0; i < q; i++) {
            String[] parts = sc.nextLine().split(" ");
            String operation = parts[0];
            String value = parts[1];
            if (operation.equalsIgnoreCase("search")) {
                System.out.println(trie.search(value));
            }
            else if (operation.equalsIgnoreCase("prefix")) {
                System.out.println(trie.startsWith(value));
            }
        }
        sc.close();
    }
}
```

STDIN

```
Anil
Anita
Anand
Suresh
Sunil
4
search Anil
search Anitha
prefix Ani
prefix Su
```

Output: 49 ms | 45.7 MB

```
true
false
true
true
```