Saturday, April 19, 2025 6:51 PM

Implement Trie

Implement **Trie** class and complete **insert(), search()** and **isPrefix()** function for the following queries :

- Type 1 : (1, word), calls insert(word) function and insert word in the Trie
- Type 2 : (2, word), calls search(word) function and check whether word exists in Trie or not.
- Type 3 : (3, word), calls isPrefix(word) function and check whether word exists as a prefix of any string in Trie or not.

Examples :

Input: query[][] = [[1, "abcd"], [1, "abc"], [1, "bcd"], [2, "bc"], [3, "bc"], [2, "abc"]]
Output: [false, true, true]
Explanation: string "bc" does not exist in the trie, "bc" exists as prefix of the word "bcd" in the trie, and "abc" also exists in the trie.

Input: query[][] = [[1, "gfg"], [1, "geeks"], [3, "fg"], [3, "geek"], [2, "for"]]
Output: [false, true, false]
Explanation: The string "for" is not present in the trie, "fg" is not a valid prefix, while "geek" is a valid prefix of the word "geeks" in the trie.

Constraints:
$1 \leq \text{query.size()} \leq 10^4$
$1 \leq \text{word.size()} \leq 10^3$

```
// User function template for C++
class TrieNode{
    public:
    TrieNode* ptr[26];
    int word;
    TrieNode(){
        for (int i = 0; i < 26; ++i)
            ptr[i] = nullptr; // Initialize all child pointers to nullptr
        word = 0;
    }
};
class Trie {
  public:
    TrieNode* root;
    Trie() {
        // implement Trie
        root = new TrieNode();
    }

    void insert(string &word) {
        // insert word into Trie
        TrieNode* cur=root;
        for(char ch: word){
            if(cur->ptr[ch-'a'] == nullptr){
                cur->ptr[ch-'a']=new TrieNode();
            }
            cur=cur->ptr[ch-'a'];
        }
        cur->word++;
    }

    bool search(string &word) {
        // search word in the Trie
          TrieNode* cur=root;
        for(char ch: word){
            if(cur->ptr[ch-'a'] == nullptr){
                return 0;
            }
            cur=cur->ptr[ch-'a'];
        }
        return cur->word?1:0;
    }

    bool isPrefix(string &word) {
        // search prefix word in the Trie
        TrieNode* cur=root;
        for(char ch: word){
            if(cur->ptr[ch-'a'] == nullptr){
                return 0;
            }
            cur=cur->ptr[ch-'a'];
        }
        return 1;
    }
};
```