Implement Trie (Prefix Tree)

Implement a trie with `insert`, `search`, and `startsWith` methods.

Note:
You may assume that all inputs are consist of lowercase letters `a-z`.

Solution 1

```
class TrieNode
{
public:
    TrieNode *next[26];
    bool is_word;

    // Initialize your data structure here.
    TrieNode(bool b = false)
    {
        memset(next, 0, sizeof(next));
        is_word = b;
    }
};

class Trie
{
    TrieNode *root;
public:
    Trie()
    {
        root = new TrieNode();
    }

    // Inserts a word into the trie.
    void insert(string s)
    {
        TrieNode *p = root;
        for(int i = 0; i < s.size(); ++ i)
        {
            if(p -> next[s[i] - 'a'] == NULL)
                p -> next[s[i] - 'a'] = new TrieNode();
            p = p -> next[s[i] - 'a'];
        }
        p -> is_word = true;
    }

    // Returns if the word is in the trie.
    bool search(string key)
    {
        TrieNode *p = find(key);
        return p != NULL && p -> is_word;
    }

    // Returns if there is any word in the trie
    // that starts with the given prefix.
    bool startsWith(string prefix)
    {
        return find(prefix) != NULL;
    }

private:
    TrieNode* find(string key)
    {
        TrieNode *p = root;
```

```
        for(int i = 0; i < key.size() && p != NULL; ++ i)
            p = p -> next[key[i] - 'a'];
        return p;
    }
};
```

written by makuiyu original link here

Solution 2

Detailed explanation after code!

```
class TrieNode {
    public char val;
    public boolean isWord;
    public TrieNode[] children = new TrieNode[26];
    public TrieNode() {}
    TrieNode(char c){
        TrieNode node = new TrieNode();
        node.val = c;
    }
}

public class Trie {
    private TrieNode root;
    public Trie() {
        root = new TrieNode();
        root.val = ' ';
    }

    public void insert(String word) {
        TrieNode ws = root;
        for(int i = 0; i < word.length(); i++){
            char c = word.charAt(i);
            if(ws.children[c - 'a'] == null){
                ws.children[c - 'a'] = new TrieNode(c);
            }
            ws = ws.children[c - 'a'];
        }
        ws.isWord = true;
    }

    public boolean search(String word) {
        TrieNode ws = root;
        for(int i = 0; i < word.length(); i++){
            char c = word.charAt(i);
            if(ws.children[c - 'a'] == null) return false;
            ws = ws.children[c - 'a'];
        }
        return ws.isWord;
    }

    public boolean startsWith(String prefix) {
        TrieNode ws = root;
        for(int i = 0; i < prefix.length(); i++){
            char c = prefix.charAt(i);
            if(ws.children[c - 'a'] == null) return false;
            ws = ws.children[c - 'a'];
        }
        return true;
    }
}
```

With my solution I took the simple approach of giving each TrieNode a 26 element array of each possible child node it may have. I only gave 26 children nodes because we are only working with lowercase 'a' - 'z'. If you are uncertain why I made the root of my Trie an empty character this is a standard/typical approach for building out a Trie it is somewhat arbitrary what the root node is.

For insert I used the following algorithm. Loop through each character in the word being inserted check if the character is a child node of the current TrieNode i.e. check if the array has a populated value in the index of this character. If the current character ISN'T a child node of my current node add this character representation to the corresponding index location then set current node equal to the child that was added. However if the current character IS a child of the current node only set current node equal to the child. After evaluating the entire String the Node we left off on is marked as a *word* this allows our Trie to know which words exist in our "dictionary"

For search I simply navigate through the Trie if I discover the current character isn't in the Trie I return false. After checking each Char in the String I check to see if the Node I left off on was marked as a word returning the result.

Starts with is identical to search except it doesn't matter if the Node I left off was marked as a word or not if entire string evaluated i always return true;

written by mlblount45 original link here

Solution 3

```
    /**
     ** author: cxq
     ** weibo: http://weibo.com/chenxq1992
     **/

    class TrieNode {
    public:
        char content;   // the character included
        bool isend;     // if the node is the end of a word
        int shared;     // the number of the node shared ,convenient to implement
delete(string key), not necessary in this problem
        vector<TrieNode*> children; // the children of the node
        // Initialize your data structure here.
        TrieNode():content(' '), isend(false), shared(0) {}
        TrieNode(char ch):content(ch), isend(false), shared(0) {}
        TrieNode* subNode(char ch) {
            if (!children.empty()) {
                for (auto child : children) {
                    if (child->content == ch)
                        return child;
                }
            }
            return nullptr;
        }
        ~TrieNode() {
            for (auto child : children)
                delete child;
        }
    };

    class Trie {
    public:
        Trie() {
            root = new TrieNode();
        }

        // Inserts a word into the trie.
        void insert(string s) {
            if (search(s)) return;
            TrieNode* curr = root;
            for (auto ch : s) {
                TrieNode* child = curr->subNode(ch);
                if (child != nullptr) {
                    curr = child;
                } else {
                    TrieNode *newNode = new TrieNode(ch);
                    curr->children.push_back(newNode);
                    curr = newNode;
                }
                ++curr->shared;
            }
            curr->isend = true;
        }
```

```
    // Returns if the word is in the trie.
    bool search(string key) {
        TrieNode* curr = root;
        for (auto ch : key) {
            curr = curr->subNode(ch);
            if (curr == nullptr)
                return false;
        }
        return curr->isend == true;
    }

    // Returns if there is any word in the trie
    // that starts with the given prefix.
    bool startsWith(string prefix) {
        TrieNode* curr = root;
        for (auto ch : prefix) {
            curr = curr->subNode(ch);
            if (curr == nullptr)
                return false;
        }
        return true;
    }
    ~Trie() {
        delete root;
    }
private:
    TrieNode* root;
};
```

written by cxq1992 original link here