```cpp
    // return count of all prefix of s;
    int countPrefix(string s){
        Node* cur = base;
        int ans = 0;
        for (int i = 0; i < s.size(); i++)
        {
            if(cur->edge[s[i]]==NULL) return 0;
            cur = cur->edge[s[i]];
            ans += cur->prefixCount;
        }
        return ans;
    }
};

```

TrieTree (for delete case):

```cpp
struct trieNode
{
    struct trieNode *child[2];
    int cnt;
    trieNode(){
        child[0] = child[1] = NULL;
        cnt = 0;
    }
} *Root;

trieNode *remove(trieNode *root, string key, int depth = 0)
{

    root->cnt--;
    if (depth == key.size()){
        if (root->cnt == 0){
            delete (root);
            root = NULL;
        }
        return root;
    }
    int index = key[depth] - '0';
    root->child[index] = remove(root->child[index], key, depth + 1);
    if (root->cnt == 0){
        delete (root);
        root = NULL;
    }
    return root;
}

void Add(string s)
{
    trieNode *Cur = Root;
    for (int i = 0; i < s.size(); i++)
    {
        int idx = s[i] - '0';
        if (!Cur->child[idx])
            Cur->child[idx] = new trieNode;
        Cur = Cur->child[idx];
        Cur->cnt++;
    }
}
```