

```

63 // return count of all prefix of s;
64 int countPrefix(string s){
65     Node* cur = base;
66     int ans = 0;
67     for (int i = 0; i < s.size(); i++)
68     {
69         if(cur->edge[s[i]]==NULL) return 0;
70         cur = cur->edge[s[i]];
71         ans += cur->prefixCount;
72     }
73     return ans;
74 }
75 };
76
77
78 TrieTree (for delete case):
79
80 struct trieNode
81 {
82     struct trieNode *child[2];
83     int cnt;
84     trieNode(){
85         child[0] = child[1] = NULL;
86         cnt = 0;
87     }
88 } *Root;
89
90 trieNode *remove(trieNode *root, string key, int depth = 0)
91 {
92     root->cnt--;
93     if (depth == key.size()){
94         if (root->cnt == 0){
95             delete (root);
96             root = NULL;
97         }
98         return root;
99     }
100     int index = key[depth] - '0';
101     root->child[index] = remove(root->child[index], key, depth + 1);
102     if (root->cnt == 0){
103         delete (root);
104         root = NULL;
105     }
106     return root;
107 }
108
109 void Add(string s)
110 {
111     trieNode *Cur = Root;
112     for (int i = 0; i < s.size(); i++)
113     {
114         int idx = s[i] - '0';
115         if (!Cur->child[idx])
116             Cur->child[idx] = new trieNode;
117         Cur = Cur->child[idx];
118         Cur->cnt++;
119     }
120 }

```