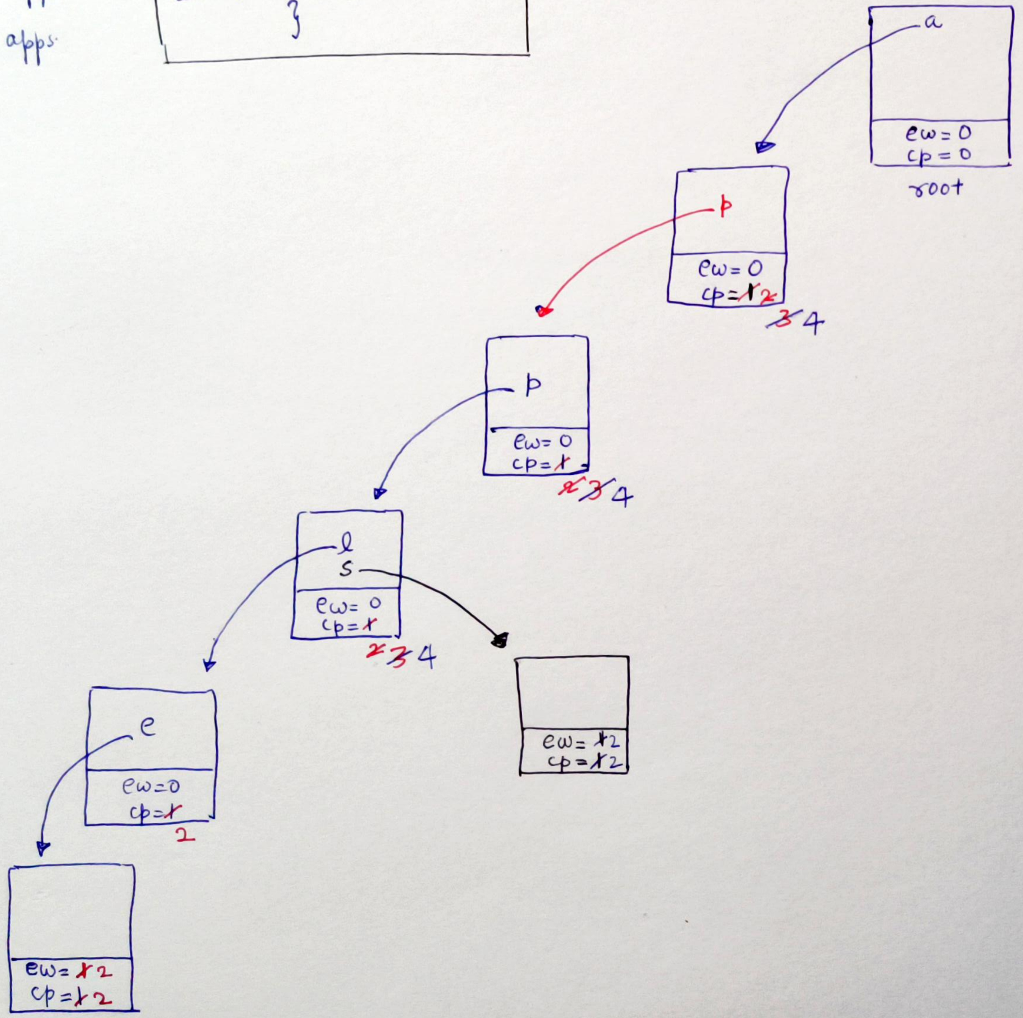
TRIE → INSERT() | CountsWordsEqualTo() | countWordsStartingWith() | erase() } → Delete the given word from TRIE.

CountsWordsEqualTo() → return how many times the given word is present in Trie

countWordsStartingWith() → return how many words are there in this TRIE that have the string PREFIX as a prefix.

```
Trie {
        link [26];
end with    ← ew = 0
count prefix ← cp = 0
}
```

- → apple
- → apple
- → apps
- → apps.



```
Class Node {
  public:
    Node* link[26];
    int ew=0; //endswith
    int cp=0; // count-prefix

  bool contain(char ch)
  {
    return link[ch-'a'] != NULL;
  }

  void put(char ch, Node* node)
  {
      link[ch-'a'] = node;
  }

  ~~void~~
  Node* get(char ch)
  {
     return link[ch-'a'];
  }

  int get_ew();
  {
    return ew;
  }

  int get-cp();

 ////ly
  ~~int~~ inc-ew();
     inc-cp();
  dec-ew();
  dec-cp();
};
```

Does ch have a reference trie.

Make a reference Trie of ch.

Move to reference trie of ch.

```
class Trie {
    Node* root;
    public:

    Trie() {
        root = new Node();
    }

  void ~~inset~~ insert(string &word)
  {
     Node* node = root;
     for (int i=0 ; i<word.size(); i++)
     {
        if (!node -> contain(word[i]))
        {
             node -> put(word[i], new Node());
        }

         node = node -> get(word[i]);
         node -> inc-cp();
     }

     node -> inc-ew();
  }
```

```
int  countWordsEqualTo ( string &word)
{
    Node * node = root;

    for (int i=0; i< word. size(); i++)
    {
        if ( !node -> contain (word[i]) return 0;

        node = node -> get (word[i]);
    }
    return node -> getew();
}


int countWordsStartingWith (string &word)
{
    Node * node = root;
    for (int i=0; i<word.size(); i++)
    {
        if (!node-> contain(word [i]) return 0;
        node = node-> get (word [i]);
    }

    return node->get-cp ();
}
```

```
void erase (string &word)
{
    Node * node = root;

    for (int i=0; i<word.size(); i++)
    {
        if (! node -> contain (word [i]))to return;

        node = node -> get (word [i]);
        node -> dec-cp();
    }

    node -> dec-ew();
}
```