

Tries (reTRIEval)

- A TRIE represents strings over some alphabet Σ by its tree of prefixes.
- Each node in a TRIE is represented by a boolean variable and a vector of length m , where m is the number of elements in Σ .
- A function, normally computed in constant time, transforms each element in Σ into an integer in $\{0, 1, \dots, m - 1\}$.
- For example, $\Sigma = \{a, b, c\}$ and $f(a) = 0$, $f(b) = 1$, and $f(c) = 2$.
- The EMPTY string is always in the set (cannot be deleted).


```
#include <iostream>
#include <string>
using namespace std;
const int TrieMaxElem = 26;
const int StrMaxElem = 80;

class Trie;

class TrieNode {
private:
    bool StrEnds;
    TrieNode *ptr[TrieMaxElem];
public:
    TrieNode();
    void SetStrEnds(){StrEnds = true;}
    void UnSetStrEnds(){StrEnds = false;}
    bool GetStrEnds(){return StrEnds;}
    void SetPtr(int i, TrieNode* j){ptr[i]=j;}
    TrieNode* GetPtr(int i){return ptr[i];}
};
```

```
TrieNode::TrieNode(){
    StrEnds = false;
    for(int i=0; i<TrieMaxElem; i++)
        ptr[i] = 0;
}
```

```
class Trie {
public:
    Trie() ;
    void Readlist();
    void Insert(char x[]);
    bool Member(char x[]);
    void Delete(char x[]);
private:
    TrieNode *root;
    bool Delete(char x[], int i,
                TrieNode *current );
    bool CheckTrieNodeEmpty(TrieNode *current);
};

Trie::Trie(){
    root = new TrieNode();
    root->SetStrEnds();
}
```

```
void Trie::Readlist(){
    int numtimes;
    char x[StrMaxElem];
    cout << endl;
    cout << "num of times" << endl;
    cin >> numtimes;
    for ( int i = 1 ; i <= numtimes ; i = i + 1 )
        { cout << "Input String" << endl;
          cin >> x;
          cout << x << endl;
          Insert( x );
        }
}
```

```
void Trie::Insert(char x[]) {
    TrieNode *current = root;
    int i = 0;
    while ( x[i] != '\0' )
        { int j = x[i] - 'a';
          if ( current->GetPtr(j) == 0 )
              current->SetPtr(j, new TrieNode());
          current = current->GetPtr(j);
          i = i + 1;
        }
    current->SetStrEnds();
}
```

```
void Trie::Delete(char x[])
{
    TrieNode *current = root;
    int i = 0;
    if (x[i] == '\0') return;
    bool j = Delete(x,i,current);
}
```



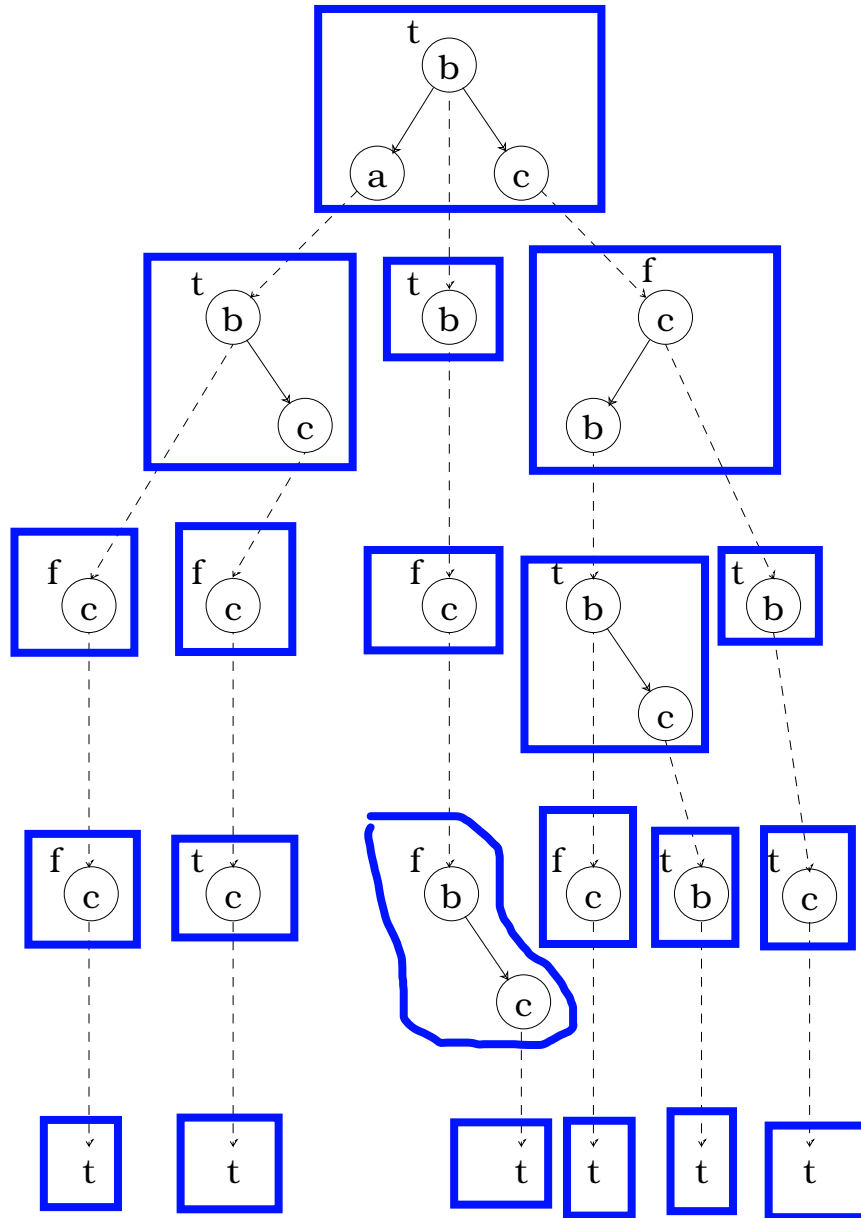
```
bool Trie::Delete(char x[], int i, TrieNode *current){
    if (current != 0)
        {if (x[i] == '\0')
            {current -> UnSetStrEnds();
             if (CheckTrieNodeEmpty(current))
                 {delete current; return true;} }
            else {if (Delete(x,i+1,current->GetPtr(x[i] - 'a'))
                {current->SetPtr(x[i] - 'a', 0);
                 if (i != 0 && CheckTrieNodeEmpty(current))
                     {delete current; return true;} }
                }
            }
        return false;
    }
```

```
bool Trie::CheckTrieNodeEmpty
                (TrieNode *current)
{
    if (current -> GetStrEnds()) return false;
    for(int i = 0; i<TrieMaxElem; i++)
        if (current->GetPtr( i )) return false;
    return true;
}
```

```
bool Trie::Member(char x[]) {
    TrieNode *current = root;
    int i = 0;
    while (x[i] != '\0')
        { int j = x[i] - 'a';
          if ( current->GetPtr(j) == 0 )
              return false ;
          else current = current->GetPtr(j);
          i = i+1;
        }
    return current->GetStrEnds() ;
}
```

```
main()
{ int numtests;
  char newelem[StrMaxElem];
  Trie t;
  t.Readlist();
  cout << endl;
  cout << "num of tests" << endl;
  cin >> numtests;
  for ( int i = 1 ; i <= numtests ; i = i + 1 )
    { if (i == (i/2)*2)
      { cout << "Input Test String" << endl;
        cin >> newelem;
        cout << newelem << "IS A MEMBER?" <<
          t.Member(newelem) << endl; }
      else{ cout << "Delete Test String" << endl;
            cin >> newelem;
            t.Delete(newelem);
            cout << newelem << " Deleted" << endl; }
    }
}
```


Binary Search Tree: Clampett 1964



Other Rep.

- Balanced binary search trees
- Hybrid Representations
- Elimination of Leaf nodes