Trie (Prefix Tree)

Bach. Rodolfo Mercado Gonzales
Universidad Nacional de Ingeniería

Trie (Prefix Tree)

- Estructura de datos representada por un árbol, la cual permite almacenar un diccionario con claves de tipo string y realizar sus operaciones fundamentales de manera eficiente.

- Usada para tareas de Information Re**trie**val, campo que estudia técnicas de recuperación de cierta información desde una gran cantidad de datos. Ejm: búsqueda de documentos en una PC, de un e-mail, etc.

Trie vs Otras estructuras

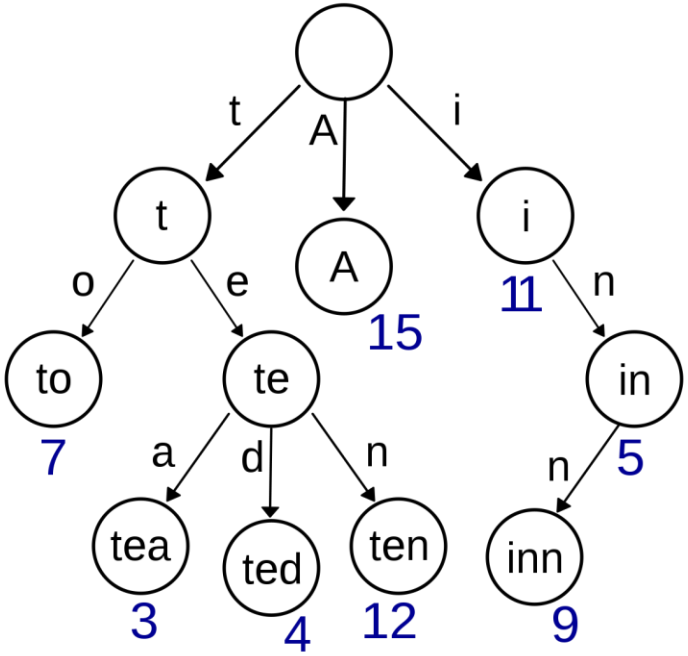
- El **trie** nos permite insertar, buscar y eliminar una palabra (clave) en $O(L)$, donde L es la longitud de la palabra.
- Además de permitirnos saber si una palabra se encuentra en el diccionario, nos permite saber qué palabras tienen un determinado prefijo en común.

Construcción del Trie

- Es un árbol donde un nodo a distancia k de la raíz representa un prefijo de longitud k (o una palabra) del diccionario.
- La raíz representa el string vacío (“”).
- Las aristas indican si un prefijo se puede extender con alguna letra del alfabeto.
- El prefijo que representa un nodo está formado por los caracteres del camino que existe desde la raíz hasta dicho nodo.
- Se denomina nodo terminal a aquel que representa a una palabra.

Construcción del Trie

Clave	valor
A	15
to	7
tea	3
ted	4
ten	12
i	11
in	5
inn	9

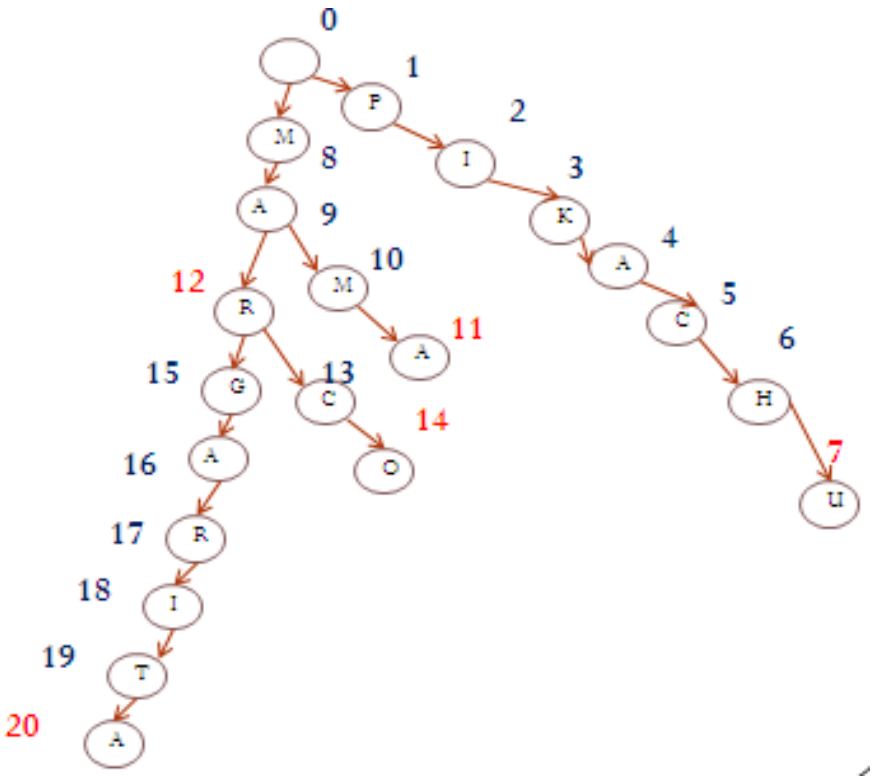


Las aristas son caracteres del alfabeto y los nodos representan prefijo o palabras.

Construcción del Trie

Construyamos un trie
solo con palabras (clave)

PIKACHU
MAMA
MAR
MARCO
MARGARITA
PIKACHU



Observaciones

- La cantidad de nodos en el trie es a lo más la suma de caracteres de cada palabra del diccionario + 1 (raíz).
- El número de hijos de cada nodo es como máximo el tamaño del alfabeto.
- Los nodos terminales no necesariamente son una hoja.

Implementación Trie

- La raíz del trie será el nodo 0.
- Por cada nodo tendremos un arreglo del tamaño del alfabeto, que funcionará como lista de adyacencia.
- Memoria: $\boldsymbol{O(N * ALPH)}$

```
#include <bits/stdc++.h>
#define N 100000 //cantidad máxima de nodos
#define ALPH 26 // tamaño del alfabeto
using namespace std;

int trie[N][ALPH]; // una especie de lista de adyacencia
/* Dado el alfabeto ['a'-'z'] representado de [0, 25] y
   el nodo u representa el prefijo P:
   si trie[u][i] = 0, entonces no existe el prefijo P+('a'+i),
   caso contrario trie[u][i] será igual a un nodo v. */
bool term[N]; //indica si un nodo es terminal o no
int nodos = 1; //cantidad de nodos en el trie
```

Implementación Trie

```
void init(int cnt){
    nodos = 1;
    for(int i=0; i<cnt; ++i){
        term[i] = 0;
        for(int j=0; j<ALPH; ++j) trie[i][j] = 0;
    }
}

void addWord(string &s){
    int sz = s.size();
    int u = 0; //empezamos en la raíz
    for(int i=0; i<sz; ++i){
        int c = s[i] - 'a';
        if( trie[u][c] == 0) trie[u][c] = nodos++; //si no existe pref creamos nodo
        u = trie[u][c];
    }
    term[u] = 1; //nodo terminal
}
```

$O(L)$, *donde* L *es longitud de la palabra*

Implementación Trie

```
bool isPrefix(string &s){
    int sz = s.size();
    int u = 0;
    for(int i=0; i<sz; ++i){
        int c = s[i] - 'a';
        if( trie[u][c] == 0) return 0;
        u = trie[u][c];
    }
    return 1;
}
```

```
bool isWord(string &s){
    int sz = s.size();
    int u = 0;
    for(int i=0; i<sz; ++i){
        int c = s[i] - 'a';
        if( trie[u][c] == 0) return 0;
        u = trie[u][c];
    }
    return term[u]; //validamos si es nodo terminal
}
```

$O(L), donde\ L\ es\ longitud\ de\ la\ palabra$

Observaciones

- Si deseamos trabajar con sufijos, podríamos hacer el trie de las inversas de las cadenas.
- Para hacer la eliminación tenemos que analizar los nodos de abajo hacia arriba, pero primero hay que dominar la funciones presentadas hasta el momento.

Problemas

SPOJ PHONELST – Phone List

SPOJ TAP2012D – Designing T-Shirts

UVA 12526 – Cellphone Typing

Codeforces 455B – A lot of Games

Referencias

- Using Tries, Topcoder
- Algorithms, Sedgewick y Wayne

¡ Good luck and have fun !