Laborator 11 - Trie

Responsabili

- Isabella Mincă [mailto:mailto:isabella.minca@gmail.com]
- Cosmin Petrișor [mailto:mailto:cosmin.ioan.petrisor@gmail.com]

Obiective

În urma parcurgerii articolului, studentul va fi capabil să:

- înţeleagă noţiunea şi structura unui trie
- construiască, în limbajul C++, un trie
- utilizeze un trie

De ce trie?

Trie-ul, cunoscut și sub numele de arbore de prefixe (prefix tree) este un arbore de căutare care permite operații de inserare și căutare de elemente in complexitate O(L) (L - lungimea cheii).

De obicei trie-ul este folosit pentru a stoca string-uri și valori asociate acestora. Pe parcursul semestrului am studiat alte 2 structuri de date care pot efectua aceleași operații: hashtable-ul si arborele binar de căutare. De ce am studia înca o structură de date care poate realiza aceleași operații?

- Trie-ul poate insera și căuta elemente in O(L), având o complexitate mai bună decât un arbore binar de căutare balansat O(logN) (ținând cont că în general lungimile cuvintelor sunt mult mai mici decât numărul de cuvinte stocate) și rezultate comparabile (uneori mai bune) decât un hashtable.
- Trie-ul permite operații de căutare mai avansate, putând efectua căutari eficiente după prefix sau pentru cuvinte cu un număr de caractere lipsă sau greșite (autocomplete).

Care este dezavantajul acestei structuri?

- Memoria utilizată variază în funcție de prefixele pe care le au in comun, însă în cazul cel mai defavorabil, numărul de noduri este egal cu suma lungimilor cheilor.

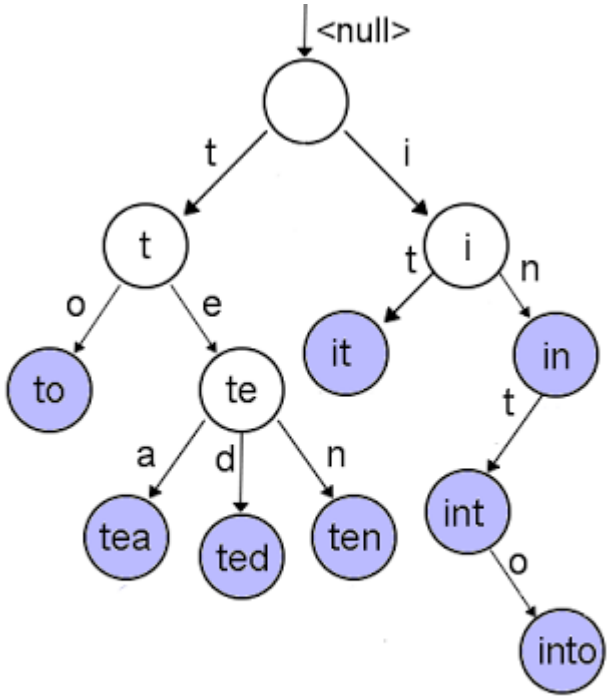
Ce este un trie?

Un trie este o structură de tip arbore care reține asocieri de tip cheie - valoare. Cheia este reprezentată în general de un cuvânt sau un prefix al unui cuvânt, dar poate fi orice listă ordonată (ex: reprezentarea binară a numerelor - bitwise trie)

Radacina utilizează pe post de cheie un string vid (””). Diferența de lungime dintre cheia asociată unui nod și cheile copiilor săi este de 1. Astfel, copiii rădăcinii sunt noduri cu chei de dimensiune 1, iar copiii acestora au chei de dimensiune 2, etc.

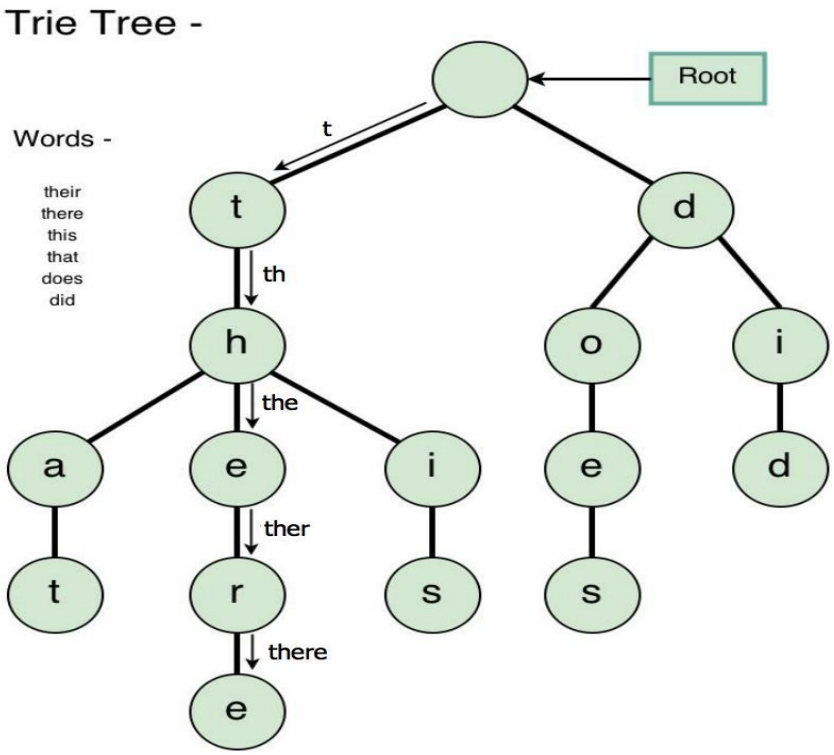
În concluzie, putem spune că pentru un nod aflat la distanța k de radacină, acesta are o cheie de lungime k. De asemenea, dacă nodul n1 este strămoș al lui n2 atunci cheia asociată lui n1 este prefix al cheii asociate lui n2.

Puteți observa în desenul de mai jos cum arată un trie. De observat că nodurile ce conțin valori sunt colorate cu albastru (nodurile corespunzătoare cheilor “in” și “int” conțin și ele valori, chiar dacă nu sunt noduri frunză)



Implementare

Fiind un arbore, trie-ul va respecta formatul standard al acestei structuri de date, însa cheia nu este reținută în mod explicit. Fiecare nod conține un vector de dimensiunea alfabetului, reprezentând copiii nodului. În cadrul implementării din laborator, cheile vor conține numai litere mici al alfabetului englez. Poziția unui nod in vectorul părintelui său este dată de poziția în alfabet a literei prin care se diferențiază de cheia parintelui său. În exemplul de mai jos, nodul cu cheia "the" este pe a 5-a poziție in vectorul de copii al nodului cu cheia "th", deoarece 'e' este a 5-a litera din alfabet.



Căutare

În cazul în care cheia are lungime 0, se reține valoarea asociată nodului și se întoarce true daca nodul curent reprezintă sfârșit de cuvânt, altfel se apelează recursiv metoda search pe nodul care continuă cheia părintelui său cu prima litera a cuvântului primit ca parametru.

Pseudocod

```
search(key) {
    if key == ""
        // The value of the node is the searched one
        return isEndOfWord

    nextNode = child corresponding to the first letter of the key

    if nextNode does not exist
        return false

    return nextNode->search(key without first letter)
}
```

Inserare

Dacă lungimea cheii este 0, am ajuns la nodul a cărui cheie se vrea a fi inserată. Altfel, se apelează recursiv metoda insert pe nodul a cărui cheie diferă de cheia părintelui său prin prima literă a cuvântului primit ca parametru.

Pseudocod

```
insert(key, value) {  // Key is the rest of the initial key to be processed
    if length of key == 0
        this->value = value
        isEndOfWord = true
        return

    nextNode = child corresponding to the first letter of the key

    if nextNode does not exist
        create nextNode
        count++

    nextNode->insert(key without first letter, value)
}
```

Ștergere

Metoda întoarce true dacă nodul poate fi șters de către părinte, fiindcă nu este prefixul vreunui cuvânt inserat și fals altfel. Dacă nodul curent este cel ce trebuie șters, se șterge valoarea asociată. În caz contrar, se apelează recursiv metoda remove pe nodul a cărui cheie are ca ultimă literă prima litera a cuvântului primit ca parametru.

Pseudocod

```
remove(key) {
    if length of key == 0
        clear value
        if node can be deleted   // has no children and is not end of word
            return true

    nextNode = child corresponding to the first letter of the key
```

```
    if nextNode exists
        if nextNode->remove(key without first letter) == true
            delete nextNode
            count--
            if node can be deleted
                return true

    return false
}
```

Schelet

Schelet

Exerciții

Fiecare laborator va avea unul sau doua exerciții publice si un pool de subiecte ascunse, din care asistentul poate alege cum se formeaza celelalte puncte ale laboratorului.

1) [**5p**] Implementați funcțiile de inserare și căutare a unei chei în Trie

2) [**3p**] Implementați funcția de ștergere un nod din Trie

3) [**2p**] Feedback

4) [**Bonus 2p**] Implementați funcția numWordsWithPrefix care întoarce numărul de cuvinte din Trie având ca prefix cuvântul dat ca parametru

Interviu

- Cum ați implementa funcționalitatea de auto-complete?
- Cum ați sorta eficient un array de șiruri de caratere?

Bibliografie obligatorie

1. Trie [https://www.wikiwand.com/en/Trie]
2. Trie insert & search [http://www.geeksforgeeks.org/trie-insert-and-search/]
3. Trie delete [http://www.geeksforgeeks.org/trie-delete/]

Bibliografie recomandată

1. Prefix tree [https://leetcode.com/problems/implement-trie-prefix-tree/description/]
2. Trie (hackerrank) [https://www.hackerrank.com/domains/data-structures/trie]
3. Trie (leetcode) [https://leetcode.com/tag/trie/]

sd-ca/2018/laboratoare/lab-11.txt · Last modified: 2019/02/01 13:29 by teodora.serbanescu