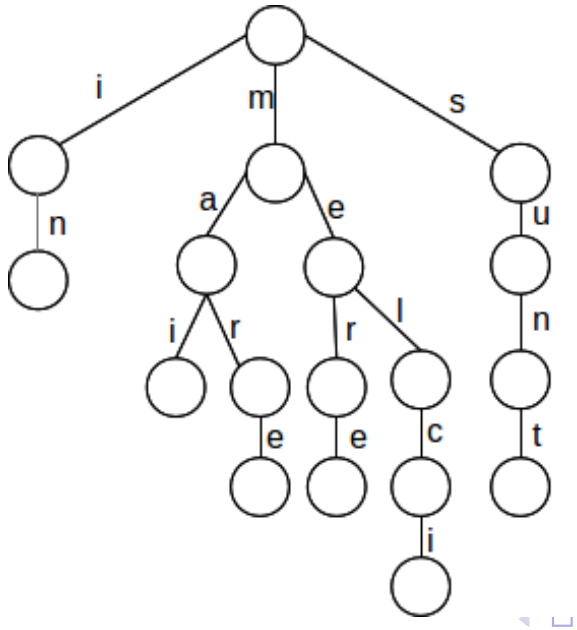
Arbori digitali

SD 2019/2020

Conținut

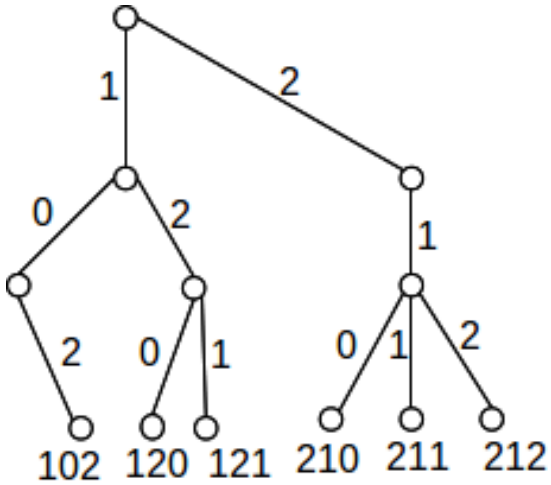
Arbori digitali (*Tries*)

- *Information re***trie***val*
- O structură de date pentru a lucra cu șiruri de caractere care beneficiază de proprietățile structurale ale acestora
- Spațiul de memorie necesar reprezentării unui dicționar este redus: rădăcina comună este reprezentată o singură dată
- Economie de memorie cand există multe prefixe comune



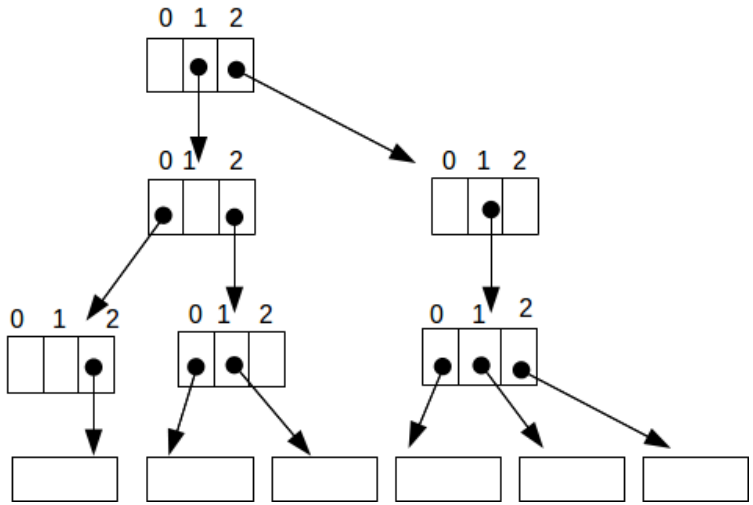
Arbori digitali

- Structură de date care se bazează pe reprezentarea digitală a elementelor
- Un arbore cu rădacină ordonat k-ar, unde k este numărul de cifre (litere din alfabet)
- Se presupune că elementele sunt reprezentate prin secvențe de cifre (litere) de aceeași lungime m ($|U| = m^k$)



Arbori digitali - Structura de date

- O colecție de noduri, fiecare nod având k fii
- Presupunem alfabetul $\{0, \ldots, k-1\}$; elementele din S sunt chei, iar nodurile de pe frontieră memorează informațiile asociate acestor chei



Arbori digitali

Un arbore digital care memorează o colecție de cuvinte S, $|S| = n$ dintr-un alfabet de mărime k, are următoarele proprietăți:

- orice nod intern are cel mult k fii
- arborele are n noduri externe
- înălțimea arborelui este egală cu lungimea celui mai mare cuvânt din S

Arbori digitali - Căutarea

- Caută un element a în structura t: parcurge drumul descris de secvența $a[0], \ldots a[m-1]$

Function *cauta*(a, m, t)
begin
    $i \leftarrow 0$
    $p \leftarrow t$
    while $(p \neq NULL\ AND\ i < m)$ **do**
        $p \leftarrow p \rightarrow succ[a[i]]$
        $i \leftarrow i + 1$
    return p
end

- Complexitatea timp pentru cazul cel mai nefavorabil: $O(m)$

Arbori digitali - Inserarea

- Inserarea unui cuvant x în structura t: simulează parcurgerea drumului descris de secvența $x[0], \ldots x[m-1]$; pentru componentele pentru care nu există noduri în t, adaugă un nod nou

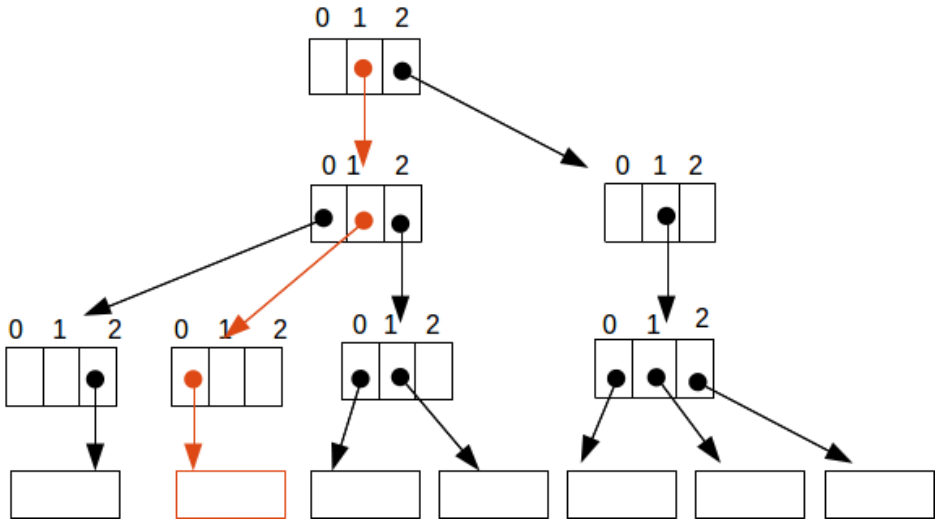


Figura: Inserarea cheii 110

Arbori digitali - Ștergerea

- Un element x care trebuie eliminat este împărțit în:
  - un prefix comun
  - un sufix care nu mai aparține niciunui element
- Se parcurge drumul descris de x și se memorează într-o stivă
- Se parcurge acest drum înapoi și dacă pentru un nod toți succesorii sunt *nil*, atunci se elimină nodul

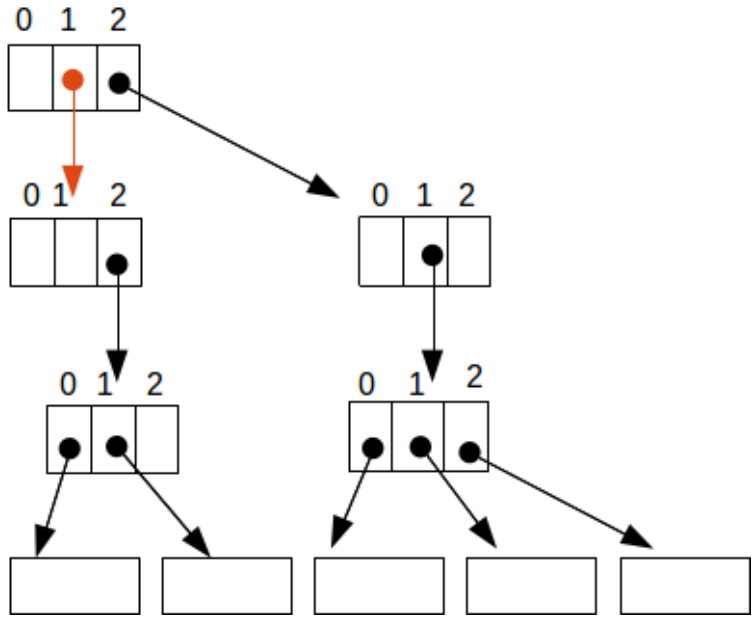
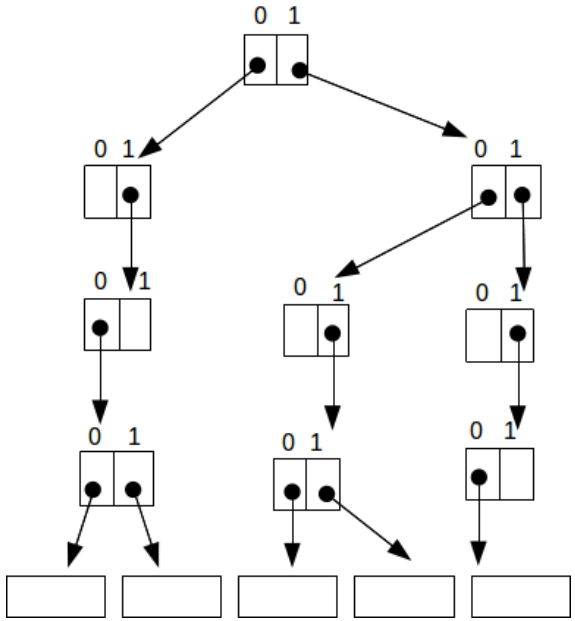


Figura: Eliminarea cheii 102

Conținut

Compactarea lanțurilor

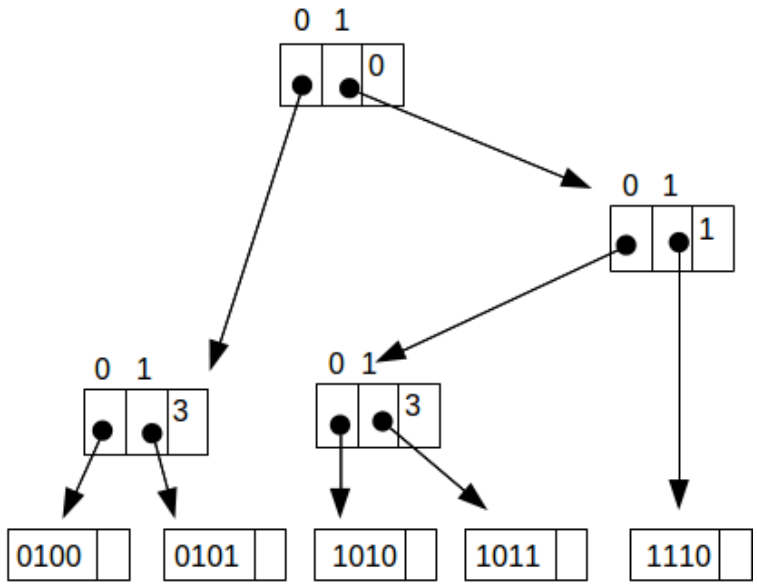
- O structură rară memorează puține chei; există multe lanțuri din noduri intermediare cu un succesor



- Eliminarea acestor lanțuri ar duce la îmbunătățirea complexității spațiu și timp; păstram doar nodurile intermediare cu cel puțin 2 succesori

Compactarea lanțurilor

- În structura compactată se memorează:
  - în nodurile interne poziția de la care diferă cheile cu un prefix comun
  - în nodurile de pe frontieră cheia.



Compactarea lanțurilor

Un arbore digital compactat care memorează o colecție de cuvinte S, $|S| = n$ dintr-un alfabet de mărime k are următoarele proprietăți:

- fiecare nod intern are cel puțin 2 fii și cel mult k fii
- arborele are n noduri externe
- numărul de noduri al arborelui este $O(n)$

Compactarea lanțurilor - Operații

- **Căutarea**: similară cazului necompactat, cu deosebirea ca în fiecare nod este testată valoarea de pe poziția memorată de nod, iar cand se ajunge pe un nod de pe frontieră se testează cheia

- **Inserarea** presupune o operație de căutare a cheii urmată de inserarea unui nou nod care să distingă după poziția pe care diferă ultimul, respectiv noul nod

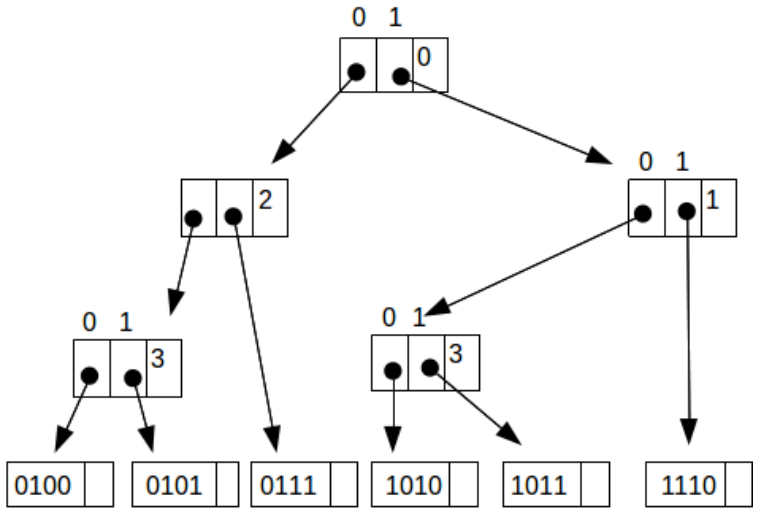


Figura: Inserarea cheii 0111

Compactarea lanțurilor - Operații

- **Stergerea**: într-o manieră asemănătoare cazului necompactat;
  - se caută nodul care memorează cheia și se memorează drumul într-o stivă;
  - se șterge nodul cu cheia respectivă
  - se parcurge drumul înapoi, eliminandu-se nodurile cu un singur succesor

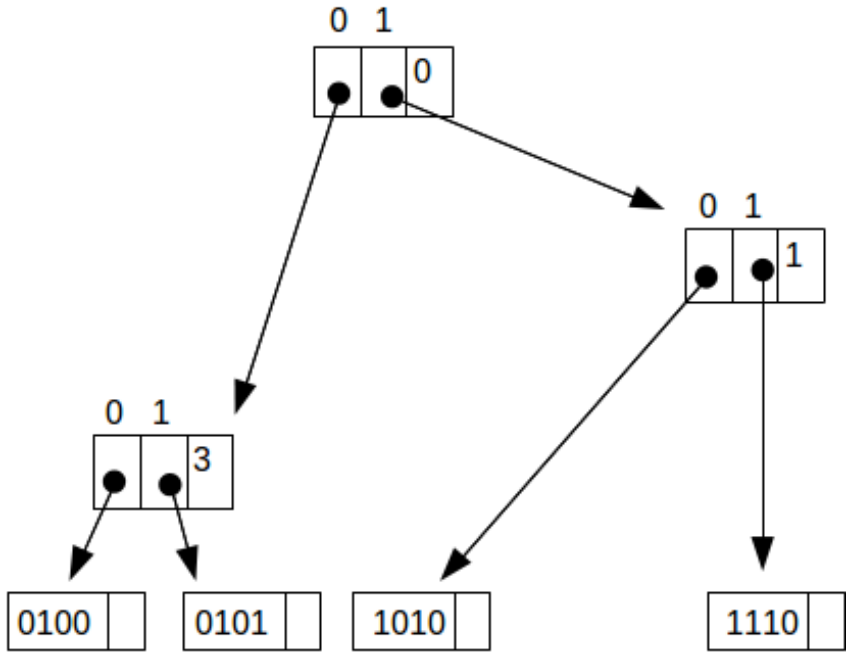


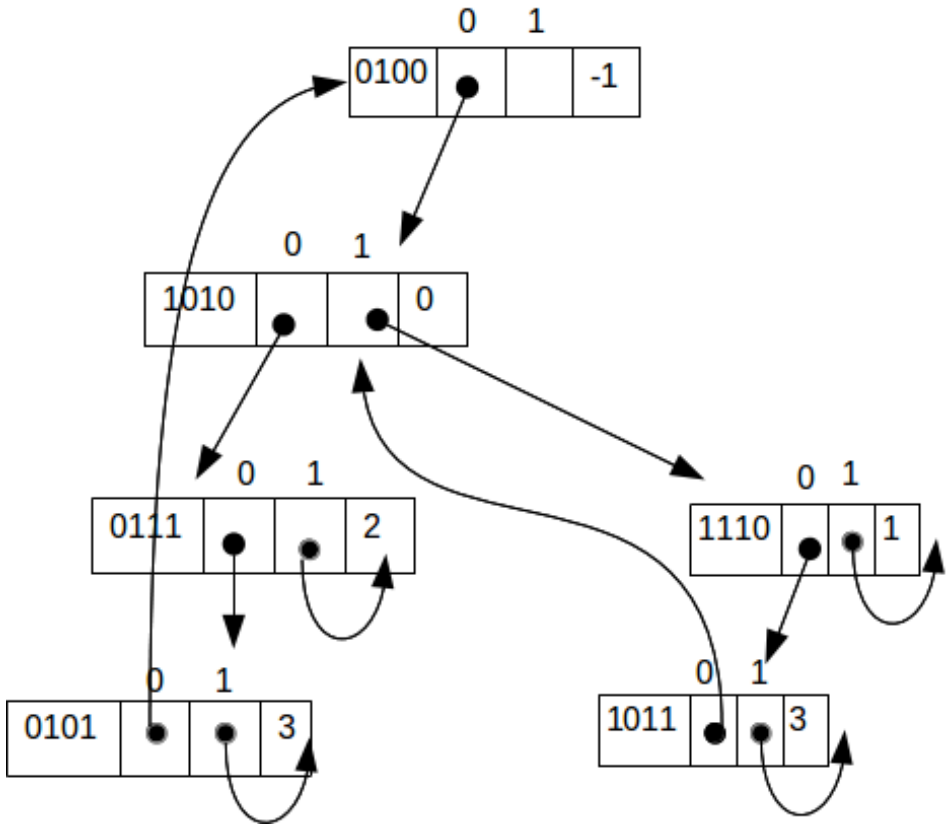
Figura: Ștergerea cheii 1011

Conținut

Structuri Patricia

- *Practical Algorithm to Retrieve Information Coded in Alphanumeric*
- Un *arbore digital binar* este un arbore digital peste alfabetul $\{0, 1\}$.
- Numărul nodurilor de pe frontieră este cu 1 mai mare decat numărul nodurilor interne.
- Dacă mai adăugam un nod intern, putem memora cheile în nodurile interne. Nodul adăugat va fi rădăcina și va avea un singur fiu (stanga). Poziția din rădăcină va fi -1.

Structuri Patricia - Exemplu



Structuri Patricia - Operații

- **Căutarea** - se poate termina cand poziția curentă este mai mică decat ultima poziție testată
- **Inserarea**: se caută cheia în structură, se identifică prima poziție j pe care x și cheia diferă;

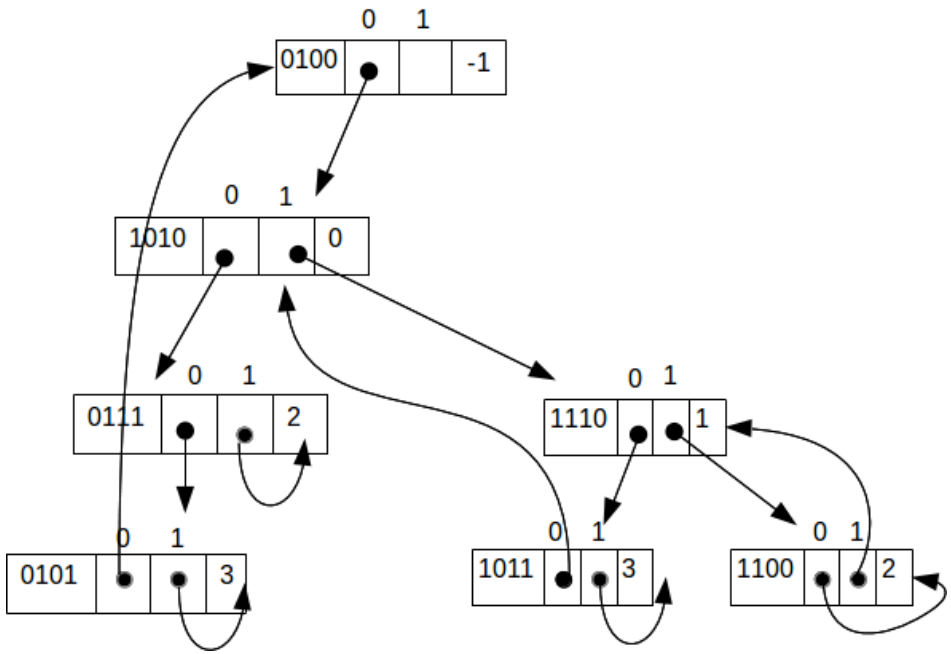


Figura: Inserarea cheii 1100

Structuri Patricia - Operații

- **Ștergerea** - exemplu

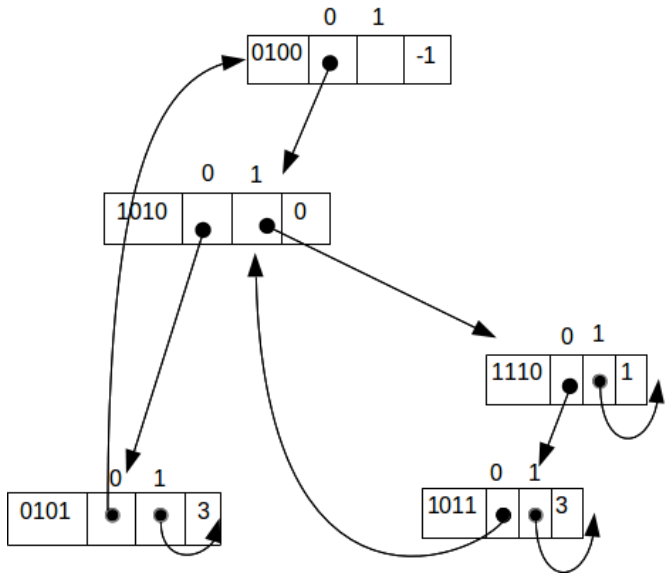


Figura: Ștergerea cheii 0111

Teoremă:

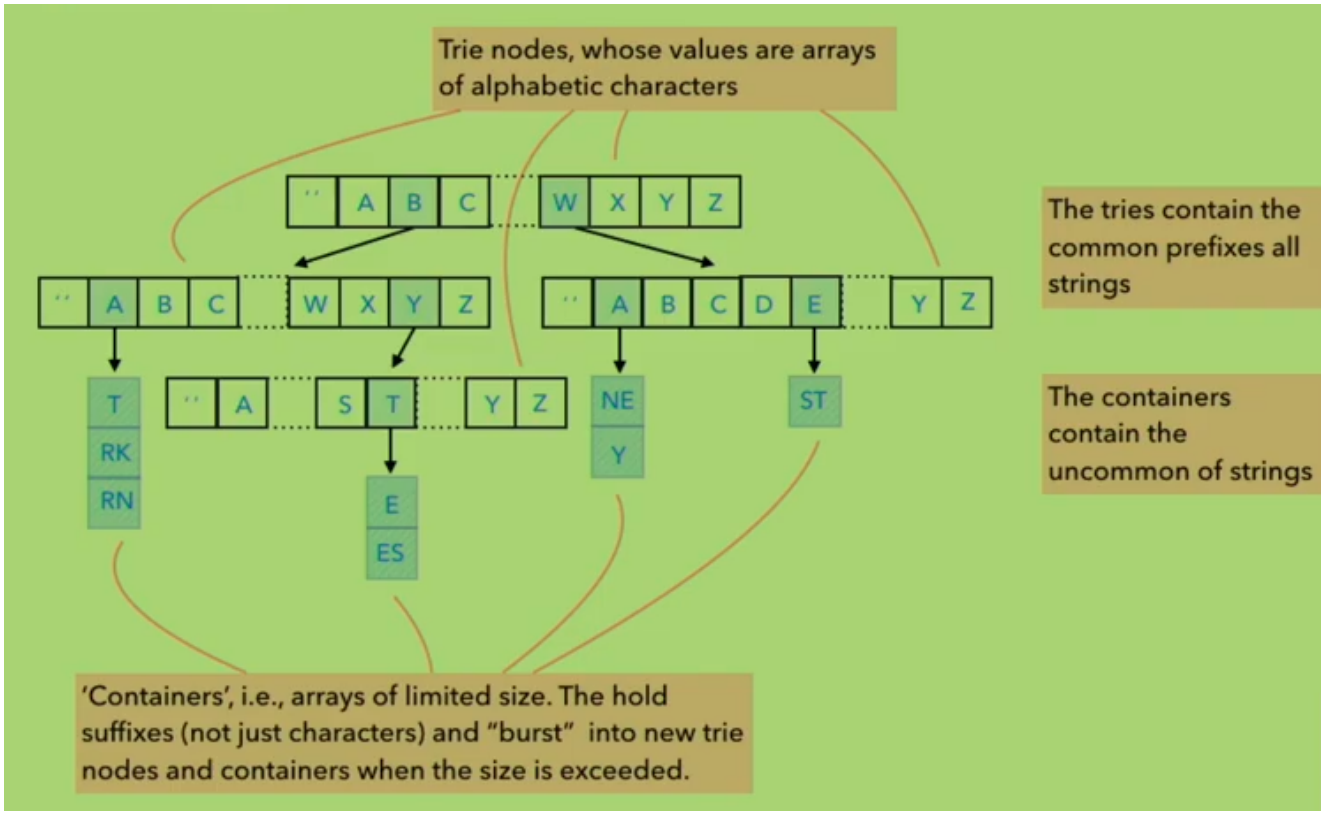
Presupunem ca se creează o structură Patricia pornind de la o structură vidă prin n inserări de chei generate aleator. Atunci operația de căutare necesită $O(\log\ n)$ comparații în medie.

Arbori digitali - Aplicații

- Determină toate cheile care încep cu un prefix dat (funcția *autocomplete*)
- Determină cea mai lungă cheie dintr-o tabelă de simboluri care este prefix al unui șir de caractere (exemplu: pentru a trimite pachete, alege adresa IP care este cel mai lung prefix)
- Determină toate cuvintele care corespund unei secvențe de numere

- Căutarea în baze de date, în rețele P2P, în fisiere XML
- Biologie computațională

Burst Tries



Burstsort

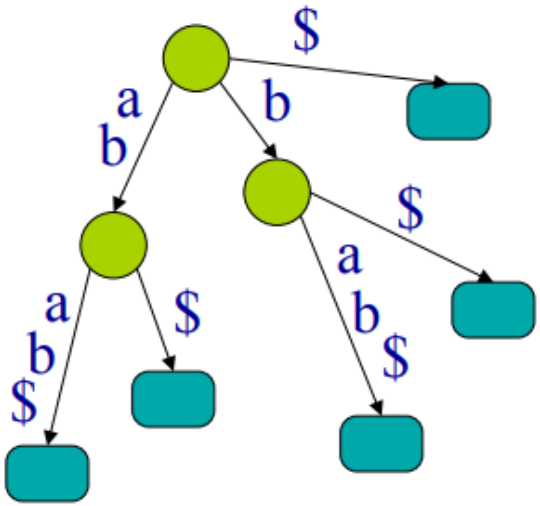
- complexitatea timp pentru inserare
- DFS (inordine)

Conținut

Arbori de sufixe

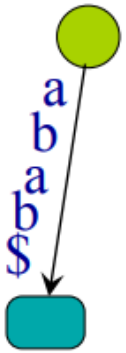
Un *arbore de sufixe* pentru un sir de caractere s este un arbore digital compactat al sufixelor lui s.

Exemplu: $s = abab$; multimea de sufixe: $\{\$, b\$, ab\$, bab\$, abab\$\}$.

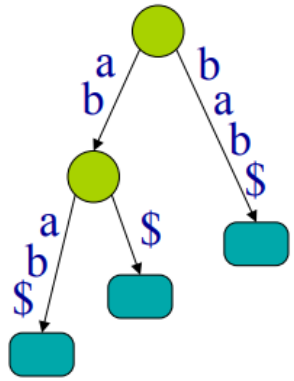


Constructia unui arbore de sufixe

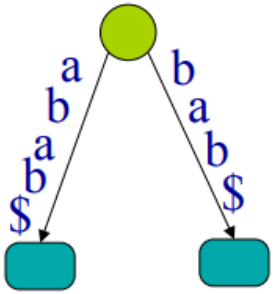
Adauga cel mai mare sufix:



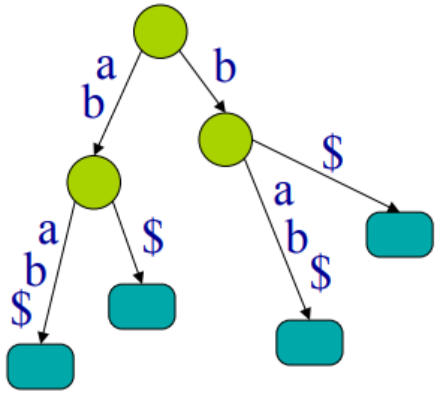
Adauga sufixul *ab*$:



Adauga sufixul *bab*$:

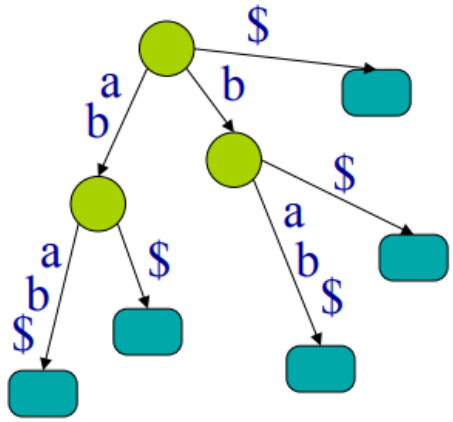


Adauga sufixul *b*$:

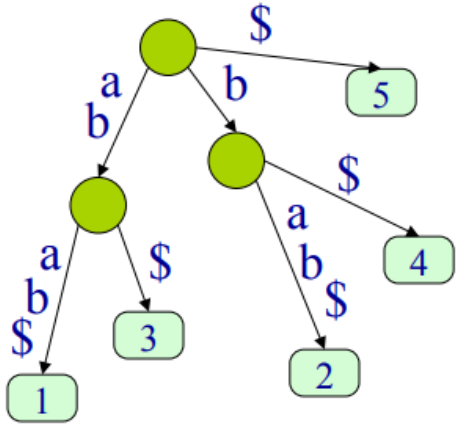


Constructia unui arbore de sufixe

Adauga sufixul $:



Eticheteaza nodurile frunza (cu indexul sufixului corespunzator):



Complexitate timp: $O(n^2)$. Algoritmul lui Ukkonen: $O(n)$.

Utilizare: string matching

Verifica daca un subsir de caractere (*pattern*) $P(|P| = m)$ apare intr-un sir $T(|T| = n)$.

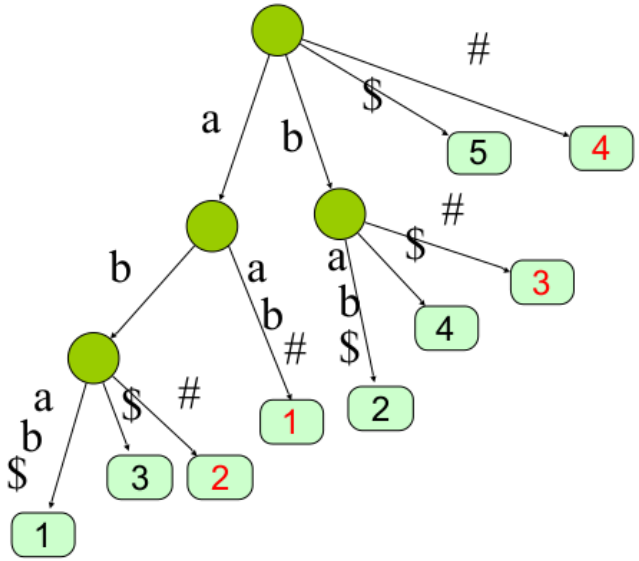
- construieste un arbore de sufixe pentru T $(O(n))$
- traverseaza arborele conform subsirului P
- fiecare frunza din subarborele identificat corespunde unei aparitii (k aparitii)

Complexitate: $O(n + k)$.

Utilizare: cel mai lung subsir comun

Un *arbore generalizat de sufixe* pentru o multime de siruri S este un arbore digital compactat al sufixelor $s \in S$.

Exemplu: $s_1 = abab$, $s_2 = aab$; $\{\$, b\$, ab\$, bab\$, abab\$\}$, $\{\#, b\#, ab\#, aab\#\}$.



Utilizare: cel mai lung subsir comun

Exemplu: $s_1 = abab$, $s_2 = aab$; $\{\$, b\$, ab\$, bab\$, abab\$\}$, $\{\#, b\#, ab\#, aab\#\}$.

Orice nod care are \$ si # in subarborele sau corespunde unui subsir comun pentru s_1 si s_2. Cel mai lung subsir comun: nodul care satisface proprietatea si care are cel mai lung subsir.