Cel mai mic strămoș comun

Olivotto Alexandra

Facultatea de Automatică și Calculatoare, Universitatea Politehnica București, Grupa 325CD
alexandra.olivotto18@gmail.com

Abstract. Lucrarea își propune analiza unor soluții aplicabile problemei găsirii celui mai mic strămoș comun, într-un timp de complexitate cel puțin O(h), unde h este înălțimea arborelui.

Cuvinte-cheie: arbori binari - cel mai mic strămoș comun .

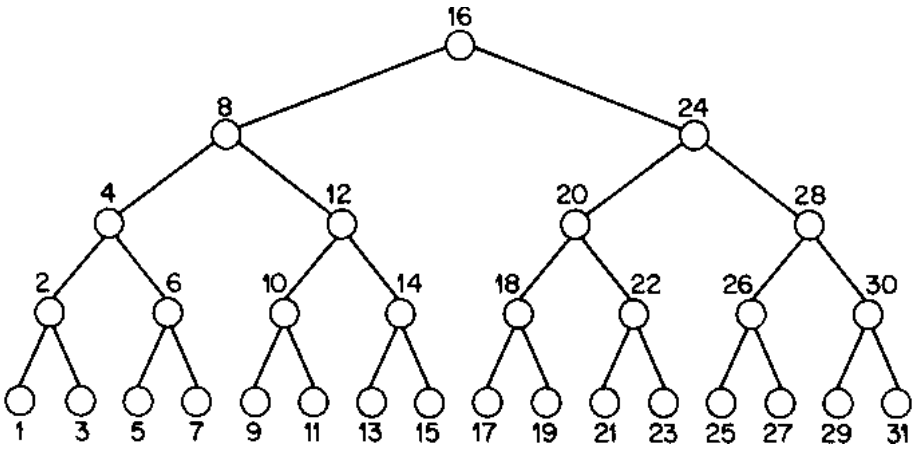
I. Introducere

- **Descrierea problemei rezolvate**

Dat fiind un arbore și două noduri (u și v) care aparţin acestuia, se pune problema identificării celui mai mic strămoș comun al celor două noduri. Prin cel mai mic strămoș comun se înțelege nodul cel mai îndepărtat de rădăcina care este strămoș atât pentru u, cât și pentru v.

O aplicație practică și actuală a problemei celui mai mic strămoș comun se poate reduce la identificarea persoanei care influențează eficient oameni în mediul online. De exemlu, putem crea un grup cu persoanele care au comentat sau distribuit conținut pe marginea unui anumit subiect. Prin traversarea unui grafic alcătuit pe baza conexiunilor dintre persoane (prieteni, pagini urmărite, timp) putem găsi persoana care a influențat anumite două persoane.

Se va lucra pe un arbore in care se adauga nodurile din input pe fiecare nivel pana la completarea acestuia. Va rezulta un arbore de forma celui din figura:



- **Specificarea soluțiilor alese**

  i. **Determinarea LCA folosind pointeri către părinți**

  ii. Determinarea LCA folosind problema identificarii elementului minim pe un interval (RMQ)

- **Criterii de evaluare**

Setul de teste are în vedere evaluarea corectitudinii și eficienței solutiei ținând cont de raportul dintre numărul de interogări necesare (M) și numărul nodurilor arborelui (N).

Cazul cel mai defavorabil se întâlnește când u și v reprezintă cele două extremități (frunză cea mai din stânga, respectiv frunză cea mai din dreapta).

II. Prezentarea solutiilor

- **Modul de functionare al algoritmilor**

1. Pointeri catre parinti

Identificarea celui mai mic stramos comun se simplifica semnificativ daca avem acces la parintele fiecarui nod, iar algoritmul care se foloseste de acest avantaj abordeaza problema astfel: fiind date u si v, doua noduri ale arborelui, acesta salveaza intr- o lista toti stramosii nodului u, dupa care parcurge toti stramosii v si il returneaza pe primul care se regaseste printre stramosii lui u.

2. RMQ

În implementarea acestei soluții se au în vedere trei vectori utilizați astfel: un vector E[] care va reține parcurgerea în adâncime (DFS) a arborelui, un vector L care pe poziția i va păstra nivelul nodului stocat în E[i] și un vector H care păstrează prima apariție a nodului i în vectorul E.

Rezolvarea propriu zisă presupune reducerea vectorului L la un subinterval dat de valorile lui H[u], respectiv H[v], astfel încât se restrânge aria de căutare la calea prin care se poate ajunge la nodul v pornind din nodul u. Tot ce rămâne de făcut este să căutăm minimul din intervalul obținut, fie acesta L[min], iar cel mai mic strămoș comun se va află pe poziția *min* din vectorul E.

- **Complexitatea solutiilor**

1. Pointeri catre parinti

Construirea listei de stramosi a nodului u implica o complexitate O(1) pentru adaugarea unui element si adauga maxim h elemente, in cazul cel mai defavorabil

cand u este frunza a arborelui. Complexitatea per interogare este O(1) si se realizeaza cel mult h interogari, in cazul in care v este tot o frunza a arborelui in extremitatea opusa lui v, deci in cazul in care au un unic stramos comun: radacina.

2. RMQ

Aceasta implementare presupune mai inatai construirea a trei vectori, fiecare avand complexitatea in timp O(n), rezultand un total de O(n). Complexitatea per interogare este de O(1), iar in cazul cel mai defavorabil vom avea h interogari, unde h este inaltimea arborelui.

- **Avantajele si dezavantajele solutiilor**

1. Pointeri catre parinti

Avantajul acestei implementari consta utilizarea unui spatiu redus pentru rezolvarea problemei si de asemenea si timpul alocat construirii structurii auxiliare este de proportii minimizate. Implementarea este dezavantajoasa daca avem in vedere retinerea unui volum (variabil, dar de proportii mari in cazul cel mai defavorabil) de informatii nenecesare si iterarea peste acestea, intrucat nu stim exact ce cautam, ci avem nevoie de interogari la fiecare pas.

2. RMQ

Implementarea este avantajoasa pentru ca restrange semnificativ spatiul de cautare, de la un arboore intreg la calea de la u la v, interval cunoscut din care stim ca ne este utila valoarea minima, deci vom sti exact ce trebuie sa cautam. Dezavantajos este spatiul de care avem nevoie, deoarece folosim 3 vectori de dimensiuni care depind de n, ceea ce afecteaza si timpul in care este rezolvata problema.

III. Evaluare

1. Construirea setului de teste

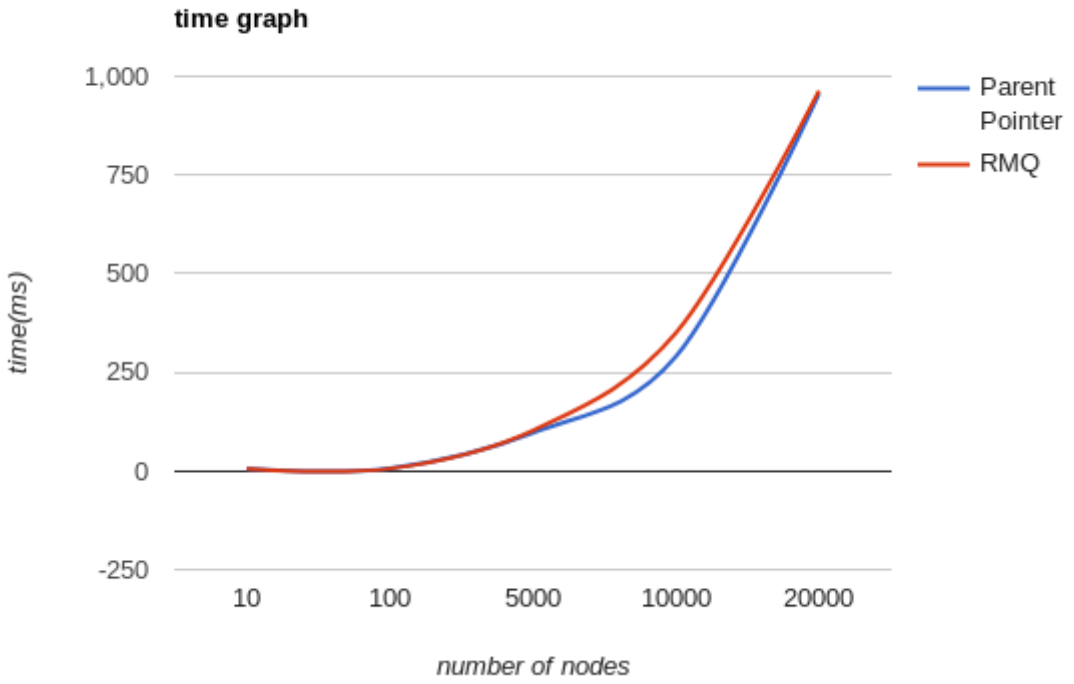
Am construit setul de teste folosind un generator de numere care primea la input numarul de noduri si intervalul in care trebuia sa se afle valoarea fiecarui nod. La output returna n numere care reprezinta nodurile si pe urmatoarea linie doua numere u si v care erau de asemenea numere returnate random dintre cele care formau deja arborele. Pentru a putea testa si in functie de proportionalitatea lui M si N cu numarul de interogari, am folosit acelasi arbore generat random pentru 3 teste si am schimbat valorile pentru u si v manual astfel incat sa indeplineasca conditiile.

2. Specificatiile sistemului

Operating System: Ubuntu 16.04 64-bit
Processor: Intel(R) Core(TM) i5-8265U CPU @ 1.60GHz
Memory (total): 7978172
Memory(available): 5373912

3. Ilustrarea rezultatelor evaluarii

Numar de noduri	Timp pointeri catre parinti	Timp RMQ
10	5ms	5ms
100	7ms	6ms
5000	97ms	103ms
10000	290ms	350ms
20000	957ms	964ms



4. Prezentarea valorilor obtinute

Se observa ca implementarea **RMQ** este mai eficienta din punct de vedere al timpului de executie pe inputuri de dimensiuni mici. Motivul pentru care consider ca timpul de executie creste o data cu inputul pentru aceasta implementare este parcurgerea vectorului H a carui dimensiune este egala cu cheia maxima din arbore (pe masura ce creste numarul de

noduri, trebuie sa creasca si valorile cheilor deoarece acestea trebuie sa fie unice in arbore). Valorile sunt de fiecare data cele asteptate, dovedindu-se astfel corectitudinea algoritmilor.

IV. Concluzii

Orice algoritm prezinta simultan avantaje si dezavantaje, iar alegerea trebuie facuta in functie de resursele disponibile si asteptarile pe care le avem. In vederea rezolvarii problemei practice mentionate in introducerea lucrarii, consider ca algoritmul potrivit este RMQ, deoarece in retelele de socializare care merg pe prinicipul de “follow” legaturile dintre urmaritori si urmariti nu sunt reciproce, astfel ca s-ar putea crea confunzii intr-o parcurgere de tipul celei care foloseste pointeri catre parinti.

V. Referinte

[1] D. Harel and R.E. Tarjan. *Fast Algorithms For Finding Nearest Common Ancestors.* 1984.
[2] Michael A. Bender and Martin Farach-Colton. *The LCA Problem Revisited*. 2000.
[3] Daniel P. *Range Minimum Query and Lowest Common Ancestor*. WebLink