Cel mai mic strămoș comun

Matei Barbu

Universitatea Politehnica București
Facultatea de Automatică și Calculatoare
matei.barbu1905@stud.acs.upb.ro

Abstract. This paper presents a comparison between two classic algorithms, Tarjan-offline and RMQ, for the Lowest Common Ancestor problem. They are analyzed from the standpoint of time complexity, taking into account both phases of preprocessing and query answering. The comparison places focus not only on the reductibility of LCA to RMQ, but also on the flexibility of an established data structure, union-find.

Keywords: Algorithm · Complexity · Preprocess · Query

1 Introducere

1.1 Descrierea problemei

Dintr-un arbore cu rădăcină, $A = (V, E)$, din care alegem două noduri, $u, v \in V$, dorim să aflăm un nod, w, care este și strămoșul lui u și al lui v. Dacă nodul w are adâncime maximă atunci spunem că *w este cel mai mic strămoș comun al lui u și v.* În decursul lucrării vom folosi și prescurtarea din engleză a problemei, lowest common ancestor, *L.C.A.*, dar și *C.M.M.S.C.*
Noțiunile de strămoș și adâncime vor fi folosite în sensul definițiilor de mai jos.

Definiția 1. *Lungimea unui lanț dintre două noduri este egală cu numărul de muchii ce unesc nodurile lanțului.*

Definiția 2. *Distanța cea mai scurtă dintre două noduri, $d : (V \times V) \to \mathbf{N}$.*

$$d(u,v) = \begin{cases} 0 & \text{dacă } x = y \\ min\{\text{lungimea oricărui lanț dintre } x, y\} & \text{dacă } x \neq y \end{cases}$$

Definiția 3. *Adâncimea unui nod într-un arbore este egală cu distanța de la rădăcină la acel nod.*

Definiția 4. *Un nod w, este strămoș al lui v, dacă w aparține lanțului de la rădăcina arborelui la v și adâncimea lui v este mai mare decât al lui w.*

Exemplu: Cel mai mic strămoș comun al lui x și y (vezi fig. 1) este colorat cu verde închis, iar toți strămoșii comuni sunt colorați verde deschis. Adâncimea lui x este 3, iar a lui y, 4.

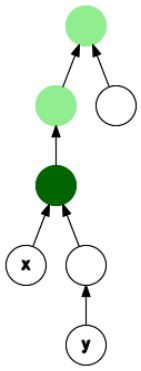

Fig. 1. [7]

1.2 Aplicații practice

Genetica și taxonomia sunt două domenii în care problema de față este utilă pentru a identifica trăsături comune în seturi de date sau contaminări culturale în genomuri. O implementare care utilizează direct C.M.M.S.C. este programul BASTA, despre care se pot citi mai multe în revista din referințe. [1]

Moștenirea claselor în programarea orientată pe obiecte este un exemplu clasic în care este utilă determinarea relațiilor de dependență. O discuție pe tema determinării celei mai apropiate clase comune moștenită de două obiecte se află în referințe. [5]

1.3 Soluții propuse

Cele două soluții alese pentru rezolvarea problemei sunt: algoritmul off-line al lui Tarjan și algoritmul de determinare cu ajutorul elementului minim dintr-un interval (R.M.Q. - Range Minimum Query).

În articolul din referințe de D. Harel și R. E. Tarjan, un contribuitor la importante rezultate legate de pădurile de mulțimi disjuncte, vezi [4] pag. 585, a fost printre primele studii importante pe această tema, în care au demonstrat că este necesară pentru un răspuns în timp constant o preprocesare liniară. [2]

În ceea ce privește reducerea C.M.M.S.C. la E.M.I., adică la determinarea elementului minimului dintr-un interval, aceasta se poate face în timp liniar, deși aparent problemele nu sunt asemănătoare. E.M.I. se poate reduce la rândul lui la C.M.M.S.C. [6] Implementarea soluției se va face prin metoda programării dinamice.

1.4 Criterii de evaluarea a soluțiilor

Algoritmii propuși se diferențiază în primul rând prin complexitățiile lor spațiale și temporale, dar mai ales prin modul de tratare a interogărilor. Algoritmul lui Tarjan necesită ca toate interogările să fie date în avans. Răspunsurile la întrebări se dau în timp ce procesăm graful, de aici și denumirea de *off-line*. Abordarea online ne prezintă fiecare interogare doar după ce am răspuns la cea anterioară.

Astfel, în evaluarea teoretică a calității soluțiilor nu este suficientă caracterizarea printr-un singur termen, corespunzător complexității temporale asimptotice, ci trebuie să luăm în calcul, separate una de cealaltă, complexitatea preprocesării și cea a interogărilor. Spre deosebire, complexitatea spațială adițională va fi în general caracterizată de memoria structurii de date necesare etapei de preprocesare.

Testele vor încerca să sublinieze calitățile individuale ale fiecărui algoritm și mai ales să răspundă la următoarele întrebări: Ce soluție performează când avem un număr de interogări mic, respectiv mare? Este justificată memoria necesară preprocesării?

2 Prezentarea soluțiilor

2.1 Modul de funcționare

Algoritmul lui Tarjan off-line se bazează pe următoare observație ca să aflăm cel mai mic strămoș dintre un nod, x, pe care îl vom colora *albastru*, și orice alt nod, z: pornim din z și urcăm din părinte în părinte, până când intersectăm lanțul, verde, de la x, la rădăcina, r, *roșie*, astfel nodul de intersecție este C.M.M.S.C., vezi fig. 2. Ca să otimizăm această soluție brutală, pentru fiecare nod y, verde, de pe lanțul de la nodul x la r, îi luăm fii lui y și îi punem într-o nouă mulțime disjunctă dintr-o pădure al cărei strămoș îl marcăm y. Astfel pentru orice nod z, negru, trebuie doar să aflăm care este strămoșul acelei mulțimi.

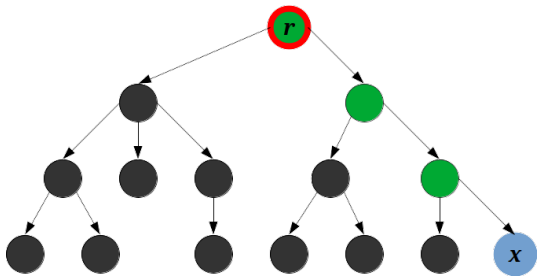

Fig. 2. Ideea din spatele algoritmului lui Tarjan

Pornim din rădăcina arborelui, o procedură recursivă, cf. alg. 1, după cum este prezentat în referințe [4], pag. 584. Pentru fiecare fiu al nodului curent procesează mulțimea disjunctă a fiului și răspunde la anumite interogări, iar apoi unește mulțimea fiului disjunctă cu cea a nodului curent. Pentru fiecare interogare care conține nodul curent și un alt nod vizitat deja, află strămoșul celuilalt nod. Vom reține o listă ascoiată fiecărui nod în care salvăm toate interorările în care acesta apare. Vom folosi implementarea de pădure de mulțimi disjuncte cu două euristici: reuniunea după rang și comprimarea drumurilor. Vezi [4] pag. 571-581.

Algoritmul cu determinarea E.M.I. se bazează pe următoarea observație, într-o parcurgere în adâncime a arborelui, între vizitele dintre două noduri, u și v, cel mai adânc nod vizitat este C.M.M.S.C. Deci, reducem problema la E.M.I.

cu tehnica turului eulerian. Arborele este privit precum un graf orientat, între două noduri vecine inserându-se două arce orientate diferit. Se pornește din tatăl arborelui și se parcurge graful mai în adâncime, inserându-se tatăl în tur după fiecare fiu vizitat. Vezi alg 2. La final vom avea $2N-1$ noduri în tur, dat de numărul de noduri, adică N, plus numărul de arce incidente fiecărui tată, adică, $N-1$.

Pe reprezentarea vectorială a turului eulerian, *tur*, o să observăm că pentru două noduri, v și u, care au prima apariție în *tur*, i, respectiv j, cu $i \leq j$, cel mai mic strămoș comun al lor este nodul dintre i și j cu adâncimea minimă.

Vom determina E.M.I. cu tehnica programrii dinamice, folosind un tabel rar, având N noduri în arborele inițial, pentru $0 \leq i \leq 2(N-1)$ și $0 < j \leq [log_2(N)]$, vom reține în tabel:

t(i)(j) = cel mai mic adânc nod din subsecvența tur(i, i + 2^j – 1)

Vom folosi următoarea recurență pentru a genera tabelul:

$$t(i,j) = \begin{cases} tur(i) & \text{dacă } j = 0 \\ t(i, j-1) & \text{dacă } nivel(t(i,j-1)) < nivel(t(i+2^{j-1}, j-1)) \\ t(i+2^{j-1}, j-1) & \text{altfel} \end{cases}$$

Deci pentru a afla C.M.M.S.C. dintre u și v, care au prima apariție în *tur*, i, respectiv j, cu $i \leq j$, iar $len = j - i + 1$, și $l_2 = log_2(len)$

$$CMMSC(u,v) = \begin{cases} t(i, l_2) & \text{dacă } nivel(t(i,l_2)) < nivel(t(j-2^{l_2}+1, l_2)) \\ t(j-2^{l_2}+1, l_2) & \text{altfel} \end{cases}$$

2.2 Analiza complexității soluțiilor

Algoritmul lui Tarjan îl vom analiza pas cu pas.

Operația *formează_mulțime* are complexitatea amortizată $\Theta(1)$, cf. *lemei 21.11*, pag. 579, [4].

Operația *găsește_mulțime* are complexitatea amortizată $O(\alpha(N))$, cf. lemei 21.13, pag. 580, [4], unde α este inversa funcției lui Ackermann.

Operația *unește* are complexitatea amortizată $O(\alpha(N))$, deoarece doar apelează la rândul său în spate două operații de tip *găsește_mulțime*, și apoi execută o reuniune după rang în $\Theta(1)$.

Operația *marchează_strămoș* are complexitatea amortizată $O(\alpha(N))$, deoarece doar apelează la rândul său în spate două operații de tip *găsește_mulțime*. Apoi execută operații cu cost $\Theta(1)$.

Astfel, complexitatea **temporală** amortizată a **preprocesării** din alg. 1, liniile 1-8, este dată de operațiile de mai sus din parcurgerea în adâncime, deci $O(N\alpha(N))$. Complexitatea **spațială** a preprocesării este $\Theta(N)$, deoarece folosim doar doi vectori pentru reprezentarea taților și a rangurilor.

Pentru fiecare din cele M interogări, se efectuează o operație de *găsește_mulțime*, aceasta însemnând o complexitate **temporală** amortizată pentru toate răspunsurile de $O(M\alpha(N))$. Complexitatea **spațială** pentru răspunsuri este $\Theta(M)$, deoarece

folosim câte o listă pentru fiecare nod, cu interogările care îl conțin, și doar un vector de răspunsuri.

Algoritmul cu determinarea E.M.I. îl vom analiza pas cu pas.

Pentru a precalcula turul eulerian vom avea nevoie de $\Theta(N)$, **timp** deoarece avem doar de făcut o parcurgere în adâncime într-un arbore, care are $N-1$ muchii. În ceea ce privește **memoria**, compexitatea este $\Theta(N)$, deoarece vom avea în final $2N-1$ noduri în tur, iar vectorii de reținere a adâncimii și a primei apariții a unui nod în tur, aparțin aceleași clase de complexitate, $\Theta(N)$.

Pentru a precalcula tabelul vom avea nevoie de $\Theta(Nlog(N))$ **memorie** și **timp**, din motivele specificate când am definit recurența de generare.

Pentru fiecare din cele M interogări, vom răspunde în $\Theta(1)$, deoarece determinarea C.M.M.S.C. se face doar din instrucțiuni de decizie, cum am specificat mai sus. Deci în total $\Theta(M)$ timp de răspuns, și precum această soluție poate funcționa *on-line*, ne putem lipsi de stocarea interogărilor.

2.3 Avantaje și dezavantaje

Algoritmul lui Tarjan are principalul avantaj că folosește doar $\Theta(N)$ memorie adițională pentru precalcul. Deși inversa funcției lui Ackermann crește **foarte lent**, constantele de timp din acest algoritm nu sunt neglijabile în practică, când avem **multe** interogări.

Algoritmul cu determinarea E.M.I. are principalul avantaj că răspunde instant la interogări fără memorie adițională pentru ele. Dezavantajul se materializează într-un cost mai mare de memorie și timp pentru precalcul, când avem **puține** interogări.

3 Evaluare

3.1 Proiectarea testelor

Mai întâi am folosit testele de pe site-ul din referințe [8], pentru a verifica corectitudinea implementărilor.

Apoi, am scris un script Octave, care primește numărul de noduri și interogări, și generează folosind o bibliotecă de numere pseudo-aleatore fișierele de intrare. Pentru vectorul de tați, pe poziția i, am generat un tatăl, t, respectând relația, $t < i$. Astfel se evită ciclurile. Interogările sunt generate complet aleator.

3.2 Specificațiile sistemului de evaluare

```
System:    Host: Kernel: 4.19.0-13-amd64 x86_64 bits: 64
           base: Debian 10.2 buster
           compiler: gcc v: 8.3.0
CPU:       Topology: Quad Core model: Intel Core i7-8550U bits: 64
           cache: 8192 KiB
Graphics:  Intel UHD Graphics 620
Memory:    16GB type: DDR4
Disk:      type: SSD
```

3.3 Ilustrarea rezultatelor

Pentru fiecare din testele de pe infoarena am realizat un grafic care ilustreaza secundele de rulare.

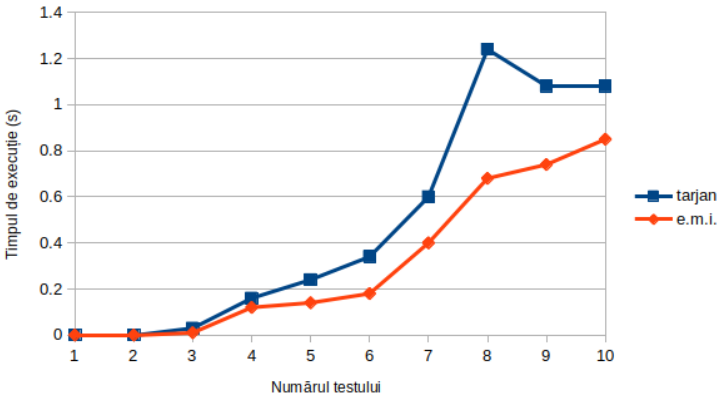

Fig. 3. Performanță pe testele de corectitudine

În ceea ce privește testele proprii le-am dat următoarele dimensiuni:

Nr. test	N	M
0	10^3	10^6
1	10^3	10^6
2	10^6	10^3
3	10^6	10^3
4	10^6	10^6
5	10^6	10^6

Și am obținut următaorele rezultate:

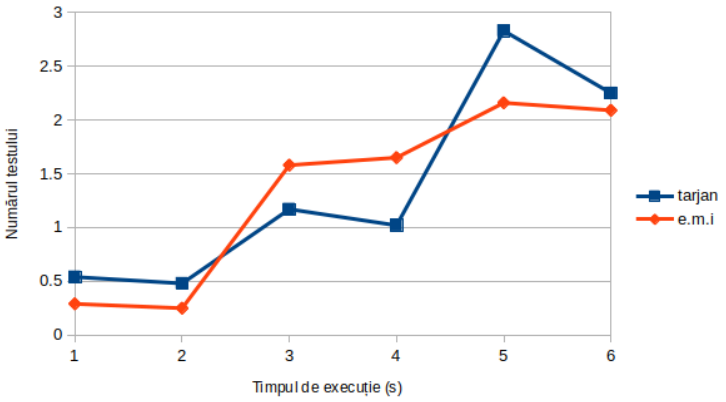

Fig. 4. Performanță pe testele speciale

3.4 Valori obținute

Observăm că alg. cu determinarea E.M.I. își păstrează un nivel de performanță aproape constant în funcție doar de dimensiunea testelor, pe când alg. lui Tarjan depinde și de valorile din teste care influențează acele constante de timp. Dar pe testele unde avem mai multe interogări decât noduri, soluția cu păduri de mulțimi disjuncte prelevează.

Folosind utilitarul valgrind de pe sistemul de operare GNU/Linux, avem pe testul special 4, pentru implementarea cu E.M.I., `307,883,208` octeți de memorie alocată dinamic. Iar pentru cealaltă implementare, `106,316,252`.

Așadar, intuițiile noastre s-au confirmat în termeni de timp și memorie.

4 Concluzii

Deși reducerea problemei celui mai mic strămoș comun la determinarea elementului minim dintr-un interval, are cea mai bună performanță de timp, ceea ce ne demonstrează ingeniozitatea soluției propuse de Tarjan este flexibilitatea unei structuri de date care poate fi adaptată și altor nevoi, păstrând o logică arborescentă.

5 Algoritmi

Algoritm 1 Soluția off-line a lui Tarjan

```
1: procedură ocmmmsc(nod)
2:     formează_mulțime(nod)
3:     pentru fiecare f fiu al lui nod execută
4:         ocmmsc(f)
5:         unește(f, nod)
6:         marchează_strămoș(nod, nod);
7:     final pentru
8:     nod.culoare ← NEGRU
9:     pentru fiecare nod v a.î. {v, nod} ∈ I execută
10:        dacă v.culoare = NEGRU atunci
11:            scrie "CMMSC de (" v "," nod ") =" găsește_mulțime(v)
12:        final dacă
13:    final pentru
14: final procedură
```

Algoritm 2 Algoritmul pentru tur eulerian

```
1: procedură tur_eulerian(nod, tur)
2:     nod.vizitat ← adevărat
3:     adaugă(tur, nod)
4:     pentru fiecare f fiu al lui nod execută
5:         dacă f.vizitat = fals atunci
6:             tur_eulerian(f, tur)
7:             adaugă(tur, nod)
8:         final dacă
9:     final pentru
10: final procedură
```

Referințe

1. Kahlke, T., Ralph P.J.: BASTA – Taxonomic classification of sequences and sequence bins using last common ancestor estimations. Methods in Ecology and Evolution **10**(1), 100–103 (2019). https://doi.org/10.1111/2041-210X.13095
2. Harel, D.; Tarjan, R. E.: Fast algorithms for finding nearest common ancestors. SIAM J. Comput. **13**(5), 338–355 (1984)
3. Popescu, D.R.: Combinatorică și teoria grafurilor. Societatea de Științe Matematice din România, București (2005)
4. Cormen, T.H.; Leiserson, C.E.; Rivest, R.L.; Stein, C.: Introduction to Algorithms. 3rd edn. MIT Press and McGraw-Hill, USA (2009)
5. Stack Overflow Question, https://stackoverflow.com/questions/15788725/how-to-determine-the-closest-common-ancestor-class, Last accessed 1 Oct 2020
6. Algopedia, Note de curs, http://algopedia.ro/wiki/index.php/Note_de_curs,_clasele_11-12,_13_martie_2014, Last accessed 1 Oct 2020
7. Wikimedia Foundation, https://commons.wikimedia.org/wiki/File:Lowest_common_ancestor.svg, Last accessed 1 Oct 2020
8. Infoarena, https://www.infoarena.ro/problema/lca?action=attach-list, Last accessed 18 Dec 2020