Cel mai mic stramos, comun (” Lowest Common Ancestor”)

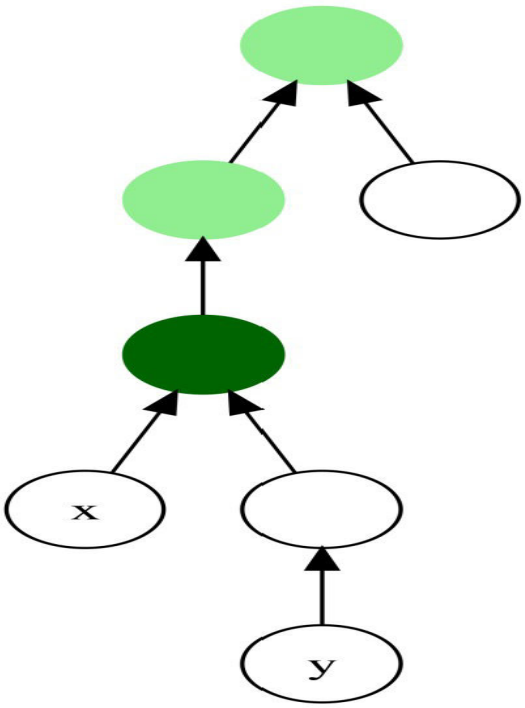
Maria Vidrasc
Grupa 324CA, Facultatea de Automatica si Calculatoare
Universitatea Politehnica Bucuresti
vidrascmaria@gmail.com

1 Introducere

1.1 Descrierea problemei rezolvate

In teorie grafică, cel mai mic strămoș comun (LCA) a două noduri v și w într-un arbore sau grafic aciclic direcționat (DAG) T este cel mai mic (adică cel mai profund) nod care are atât v și w ca descendenți, unde noi definește fiecare nod pentru a fi descendent de la sine (deci dacă v are o legătură directă de la w, w este cel mai mic strămoș comun).

LCA din v și w în T este strămoșul comun al lui v și w care se află cel mai îndepărtat de rădăcină. Calculul strămoșilor comuni mai mici poate fi util, de exemplu, ca parte a unei proceduri pentru determinarea distanței dintre perechile de noduri dintr-un copac: distanța de la v la w poate fi calculată ca distanța de la rădăcină la v, plus distanța de la rădăcină la w, minus de două ori distanța de la rădăcină la strămoșul lor cel mai mic.



Arborele binar este o structură de date arbore în care fiecare nod are cel mult doi copii, care sunt numiți copilul stâng și copilul drept. O definiție recursivă care folosește doar noțiuni de teorie a setului este aceea că un arbore binar (non-gol) este un tuple (L, S, R), unde L și R sunt arbori binari sau setul gol și S este un set singleton. Unii autori permit ca arborele binar să fie și setul gol.

1.2 Exemple de aplicatii practice pentru problema aleasa

Arbore de căutare binar - utilizat în multe aplicații de căutare în care datele intră / părăsesc constant, cum ar fi harta și obiectele setate în bibliotecile multor limbi.

Arbore binar - utilizat în aproape fiecare router cu lățime mare de bandă pentru stocarea tabelelor de router.

1.3 Specificarea solutiilor alese

RMQ(Range Minimum Query):

Range Minimum Query (Interogarea minima a intervalului) este utilizat pe tablouri pentru a găsi poziția unui element cu valoarea minimă între doi indici specificați.

Timpul de preprocesare al unui segment de arbore este O (n), pe când timpul pentru interogarea minimă a intervalului este O (Logn).
Spațiul suplimentar necesar este O (n) pentru a stoca arborele de segmente.

Binary Search:

Dacă ni se oferă un BST în care fiecare nod are indicatorul părinte, atunci LCA
poate fi determinat cu ușurință parcurgând cu indicatorul părinte și prin tipărirea
primului nod intersectant.

1.4 Specificarea criteriilor de evaluare alese pentru validarea solutiilor

Reducand LCA la RMQ:

Ideea rezolvarii RMQ este de a traversa arborele pornind de la rădăcină printr-un tur Euler (traversarea drumului intre doua noduri fara a ridica creionul de pe foaie), care este de tip DFS cu caracteristici de traversare preordine.

Să ne imaginam: nodul nostru este nodul la cel mai mic nivel și singurul nod la acest nivel dintre toate nodurile care apar între apariții consecutive (oricare) din u și v în turul Euler din T.

Reducand LCA intr-un BST:

Putem rezolva această problemă folosind proprietăți BST. Putem traversa recursiv
BST de la root. Ideea principală a soluției este că, în timp ce parcurgem de sus în jos,
primul nod n pe care îl întâlnim cu valoare între n1 și n2, adică n1 mai mic decat n2 sau la fel
ca unul dintre n1 sau n2, este LCA al n1 și n2 (presupunând că n1 < n2). Așadar, traversam
BST în mod recursiv, dacă valoarea nodului este mai mare decât n1 și n2, atunci LCA-ul
nostru se află în partea stângă a nodului, dacă este mai mic decât atât n1 cât și n2,
LCA se află în partea dreaptă. În caz contrar, rădăcina este LCA (presupunând că atât
n1 cât și n2 sunt prezente în BST).

2 Prezentarea solutiilor

2.1 Descrierea modului in care functioneaza algoritmii alesi

Un arbore binar este format din noduri, unde fiecare nod conține un indicator „stânga”, un indicator „dreapta” și un element de date. Indicatorul „rădăcină” indică nodul cel mai de sus din copac. Indicatorii din stânga și din dreapta indică recursiv „subtreze” mai mici de pe ambele părți. Un indicator nul reprezintă un arbore binar fără elemente - copacul gol. Definiția recursivă formală este: un arbore binar este fie gol (reprezentat de un indicator nul), sau este format dintr-un singur nod, unde indicatorii stânga și dreapta (definiție recursivă înainte) fiecare punct către un arbore binar.

Pentru un arbore binar – RMQ (Range Minimum Query) :

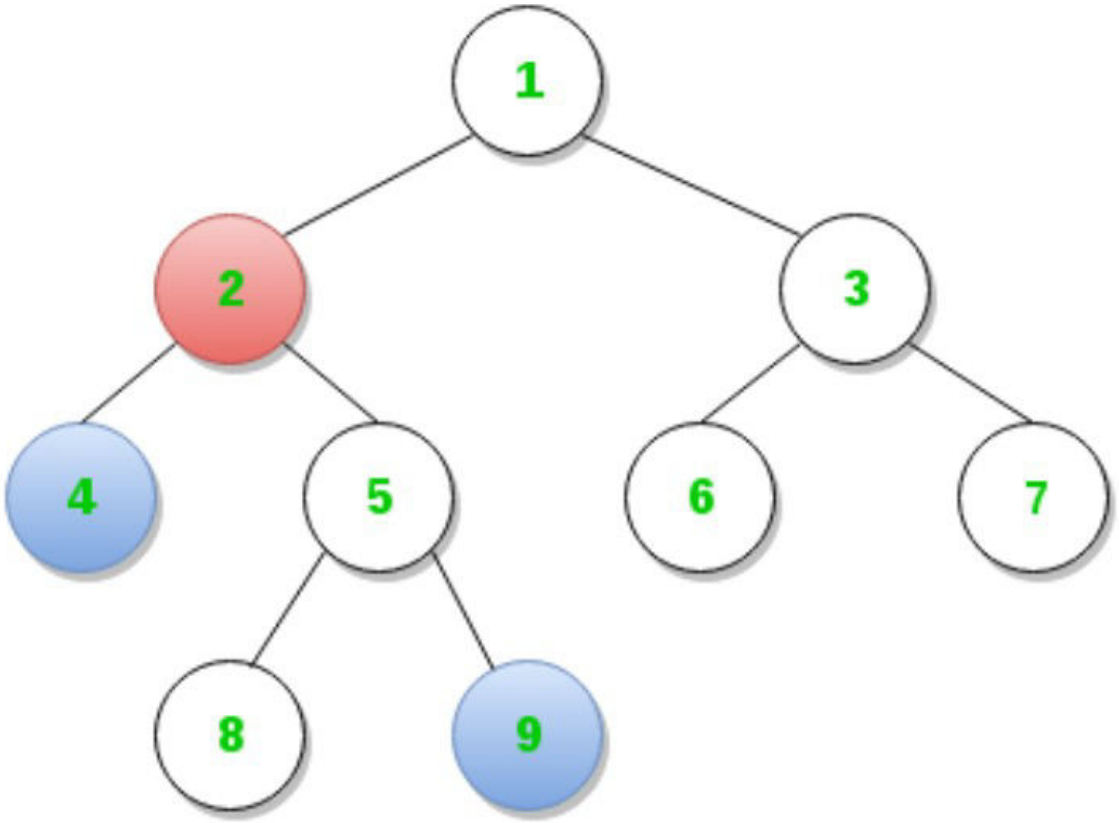
Pentru implementarea RMQ avem nevoie de trei tablouri:
1. Nodurile vizitate în ordine in turul Euler din arbore.
2. Nivelul fiecărui nod vizitat în turul Euler din arbore.
3. Indexul primei apariții a unui nod în turul Euler din arbore

Facem un tur Euler pe arbore și completam tablourile de noduri vizitate, nivele și prima apariție.

Utilizând primul tablou de apariție, obținem indicii corespunzători celor două noduri care vor fi colțurile intervalului din tabloul de nivel care este alimentat cu algoritmul RMQ pentru valoarea minimă.
Odată ce algoritmul returnează indicele nivelului minim din interval, îl utilizăm pentru a determina LCA folosind tabloul traversat DFS.

Algoritm de implementare:

```
int findLCA(Node *root, int u, int v)
{
   /* Marcheaza toate nodurile nevizitate. Rețineți că dimensiunea de
      firstOcurrence este 1 ca valori ale nodului care variază de la
      1 la 9 sunt utilizate ca indici */
   memset(firstOccurrence, -1, sizeof(int)*(V+1));
   ind = 0;
   eulerTour(root, 0);

   int *st = constructST(level, 2*V-1);

   /* Dacă v înaintea u în turul Euler. Pentru ca RMQ să funcționeze, mai întâi
      parametrul „u” trebuie să fie mai mic decât al doilea „v” */
   if (firstOccurrence[u]>firstOccurrence[v])
      std::swap(u, v);

   // Pornirea și sfârșitul indexurilor intervalului de interogare
   int qs = firstOccurrence[u];
   int qe = firstOccurrence[v];

   int index = RMQ(st, 2*V-1, qs, qe);

   /* returneaza nodul LCA */
   return euler[index];
}
```

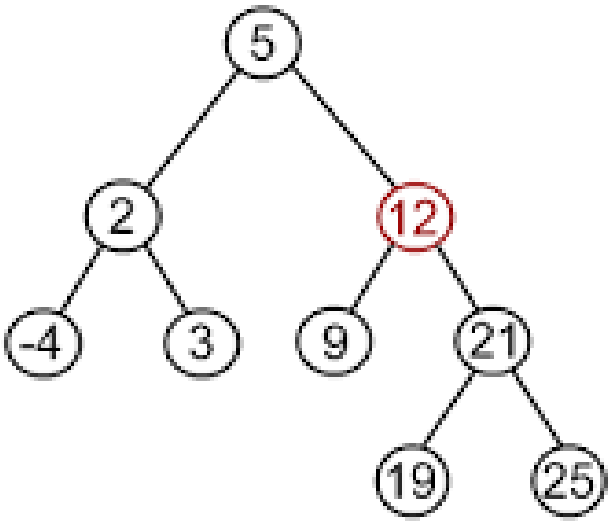


Exemple: Cel mai mic stramos comun intre 4 si 9 este 2.

Pentru un arbore binar de cautare:

```
node *lca_recursive(node* root, int n1, int n2)
{
    if (root == NULL) return NULL;

    // Dacă atât n1 cât și n2 sunt mai mari decât
    // rădăcină, atunci LCA se află în dreapta
    if (root->data < n1 && root->data < n2)
        return lca_recursive(root->right, n1, n2);

    // Dacă ambele n1 și n2 sunt mai mici
    // decât rădăcină, atunci LCA se află în stânga
    if (root->data > n1 && root->data > n2)
        return lca_recursive(root->left, n1, n2);

    return root;
```

}



Exemplu: Cel mai mic stramos comun intre 9 si 25 este 12.

2.2 Analiza complexitatii solutiilor

RMQ:

Observatie:
Presupunem că nodurile interogate sunt prezente în arbore.
De asemenea, am presupus că dacă în arbore există noduri V, tastele (sau datele)
acestor noduri sunt cuprinse între 1 și V.

Complexitatea in timp:
Turul Euler: numărul de noduri este V. Pentru un copac, E = V-1. Turul Euler (DFS)
va lua O (V + E) care este O (2 * V) care poate fi scris ca O (V).
Segment Construcția arborelui: O (n) unde n = V + E = 2 * V - 1.
Interval Interogare minimă: O (jurnal (n));

În general, această metodă durează O (n) timp pentru preprocesare, dar durează O (Log n) timp pentru interogare. Prin urmare, poate fi util atunci când avem un singur arbore pe care dorim să efectuăm un număr mare de interogări LCA (Rețineți că LCA este util pentru a găsi cea mai scurtă cale între două noduri din Arbore binar).

Complexitate spatiala:
Matricea turistică Euler: O(n) unde n = 2 * V - 1
Matrice niveluri noduri: O(n)
Matricea de întâmplare: O(V)
Arborele segmentului: O(n)
În general: O(n)

O altă observație este că elementele adiacente din tabloul de nivel diferă cu 1.

Aceasta poate fi folosită pentru a converti o problemă RMQ într-o problemă LCA.

BST:

Căutare: pentru căutarea elementului 1, trebuie să traversăm toate elementele (în ordinea 3, 2, 1). Prin urmare, căutarea în arborele de căutare binară are cea mai gravă complexitate a cazurilor de O (n). În general, complexitatea timpului este O (h) unde h este înălțimea BST.

Inserare: Pentru inserarea elementului 0, trebuie să fie introdus ca copilul stâng de 1. Prin urmare, trebuie să traversăm toate elementele (în ordinea 3, 2, 1) pentru a insera 0 care are cea mai gravă complexitate a cazului O (n). În general, complexitatea timpului este O (h).

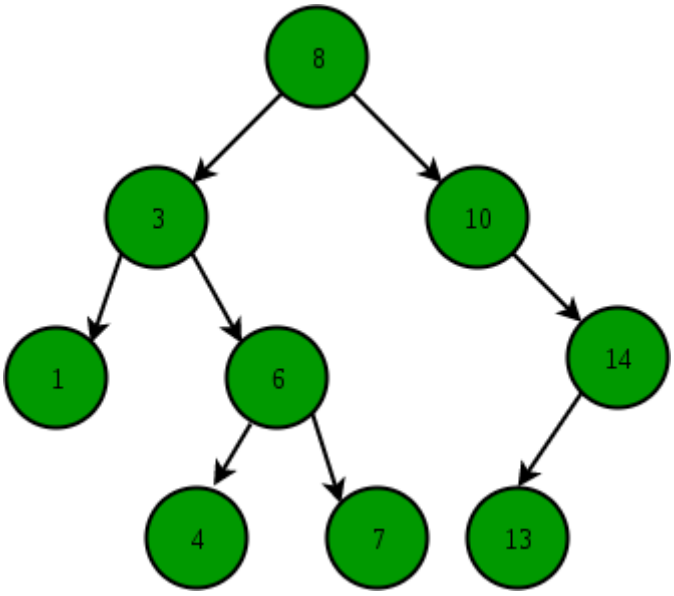
Ștergere: Pentru ștergerea elementului 1, trebuie să traversăm toate elementele pentru a găsi 1 (în ordinea 3, 2, 1). Prin urmare, ștergerea din arborele binar are cea mai gravă complexitate a cazurilor de O (n). În general, complexitatea timpului este O (h).
Complexitatea spatiala pt LCA:
In general: O(log(n))

2.3 Prezentarea principalelor avantaje si dezavantaje pentru solutiile luate in considerare

LCA in arbore binar de cautare:

Avantaje:
Timp de cautare mult mai mic.
Algoritmul este usor de inteles.

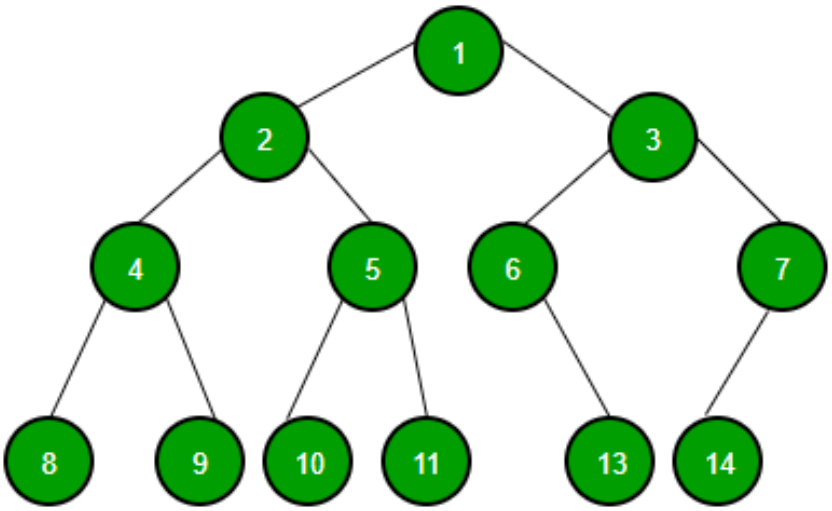
Dezavantaje:
Trebuie sa fie arbore binar de cautare, nu orice fel de arbore binar.
Poate ocupa mult spatiu; uneori aceasta structura poate fi ineficienta.



LCA folosind RMQ:

Avantaje:
Merge pe orice tip de arbore binar.
Ocupa mereu cat de putin spatiu posibil.

Dezavantaje:
Poate dura mai mult.
Algoritmul este mai complex.



3. Evaluare

3.1 Descrierea modalitatii de construire a setului de teste folosite pentru validare

Am realizat si testat testele manual. Nu am reusit sa realizez un generator de teste. De asemenea referintele pt output sunt reaizate tot manual.

3.2 Mentionati specificatiile sistemului de calcul pe care ati rulat testele

Testele au fost rulate pe un sistem avand urmatoarele specificatii:
Operating System: Ubuntu 18.04.3 LTS
Processor: Intel(R) Core(TM) i5-7300HQ CPU @ 2.50GHz
Memory: 3925 MB RAM
Available OS Memory: 1525MB RAM

3.3 Ilustrarea, folosind grafice, a rezultatelor evaluarii solutiilor pe setul de teste

Nr. testului	Nr. de noduri introduse	Nr. de perechi date spre cautare	Timp BST	Timp RMQ
0	5	1	0,05	0,054
1	9	3	0,13	0,17
2	9	5	0,15	0,2

3	12	12	0,57	0,64
4	16	3	0,37	0,41
5	9	3	0,13	0,17
6	10	4	0,15	0,21

Fig.

Fig. prezinta timpul de executie per numarul de noduri pentru cele doua structuri de date analizate in aceasta lucrare, abore binar si si arbore binar de cautare.

3.4 Prezentarea valorilor obtinute pe teste

Dupa cum se poate vedea, arborii binari de cautare ofera un timp mai bun decat arborii binari normali, chiar si pentru date de intrare de dimensiuni mari. Trebuie insa sa avem in vedere ca input-urile sunt facute sa genereze arbori binari de cautare eficienti. Acest lucu reiese si din Fig7, ce prezinta timpii obtinuti.

4 Concluzii

Dupa cum am prezentat mai sus, nu exista un algoritm ideal din punct de vedere al complexitatii timp-spatiu. Rmq ramane, totusi, un algoritm ce se poate aplica pe orice tip de arbore binar.

Surse:

https://www.geeksforgeeks.org/lowest-common-ancestor-in-a-binary-search-tree/
https://www.geeksforgeeks.org/find-lca-in-binary-tree-using-rmq/
https://www.geeksforgeeks.org/complexity-different-operations-binary-tree-binary-search-tree-avl-tree/
https://www.geeksforgeeks.org/difference-between-binary-tree-and-binary-search-tree/