Incremental DFS algorithms: a theoretical and experimental study

Iojică Mattia

decembrie 2023

Surender Baswana, Ayush Goel, Shahbaz Khan. Incremental DFS algorithms: a theoretical and experimental study. SODA '18

1 Preliminarii

Pentru toate experimentele descrise, adaugăm grafului G un nod s conectat la toate nodurile sale. Algoritmii vor incepe mereu din nodul s, iar la final, graful obținut va avea ca rădăcină nodul s, iar fiecare subarbore înrădăcinat la orice copil al lui s va fi o componentă conexă a lui G.

Având graful G, vom folosi următoarele notații:

- T este un arbore DFS a lui G.
- *path(x, y)* este drumul de la x la y în T.
- *T(x)* este subarborele cu rădăcina în x.
- *LCA(u, v)* este cel mai mic strămoș comun al lui u și v în T.
- De-a lungul lucrării ne vom întâlni cu 2 tipuri de grafuri aleatoare:
  1. *G(n, m)*: având n noduri și m muchii dintr-o permutare aleatoare a muchiilor din graf.
  2. *G(n, p)*: având n noduri, iar fiecare muchie are probabilitatea p să apară in graf.

2 Introducere

Parcurgerea în adâncime (Depth First Search) este un algoritm folosit pentru a parcurge o structură de date de tip graf. Aceasta a jucat un rol estențial în proiectarea algoritmilor eficienti pentru cu grafuri cum ar fi: componente conexte, sortare topologică, potrivire bipartită, și multe altele.

În ziua de azi, majoritatea aplicațiilor ce țin de grafuri, lucrează cu grafuri ce se schimbă constant prin adăugarea sau ștergerea nodurilor sau muchiilor din acestea. Astfel, dorim să obținem o structură de date ce va avea timpi mai buni de rulare pentru actualizarea unei soluții după o modificare a muchiilor din graf, decât o parcurgere pe un graf static. Un algoritm se numește *algoritm dinamic* (dynamic algorithm) dacă permite doar inserarea de muchii în graf.

Cu toate că este un concept ce stă la baza teoriei grafurilor, foarte puțini algoritmi incrementali au fost proiectați pentru a menține un arbore DFS. Autorii articolului prezintă acesti algoritimi, analizând funcționalitatea și performanțele (atât teoretice, cât și practice) pe care acestea le obțin pe diferite grafuri. La final, aceștia vor propune un algoritm modificat, ce va încerca să obțină performanțe mai bune decât algoritmii existenți.

Algoritmi existenți

Algoritmi DFS Statici (SDFS și SDFS-Int)

SDFS este un algoritm ce presupune reluarea algoritmului DFS de la început, de fiecare dată când o muchie noua este adăugată.

SDFS-Int este un algoritm derivat din SDFS ce se oprește atunci când toate nodurile din graf au fost vizitate. Acesta obtine performanțe mult mai bune decât SDFS pentru grafuri aleatoare.

DFS Incremental pentru Grafuri Orientate Aciclice (FDFS)

Algoritmul **FDFS** mentine numerotarea post-order (DFN) a nodurilor într-un arbore DFS. La inserarea unei muchii *(x,y)* in graf, se verifica dacă muchia este de tipul *anti-cross*, verificând dacă *DFN[x] < DFN[y]*.

În cazul în care muchia *(x, y)* nu este de tipul *anti-cross*, graful se actualizează și se termină procesul.

În cazul în care muchia *(x, y)* este de tipul *anti-cross*, se va face un DFS parțial pentru nodurile în care se poate ajunge din *y* în subgraful indus de nodurile avand DFN-ul cuprins între DFN[x] și DFN[y]. Astfel, se vor selecta nodurile aflate în subarborele din dreapta *path(LCA(x, y), x)* sau din stânga *path(LCA(x, y), y)*. FDFS elimină aceste vârfuri din subarborele corespunzător și calculează arborele DFS înrădăcinat nodului *y* din T prin muchia *(x, y)*. Pentru fiecare nod se va recalcula DNF-ul.

DFS Incremental pentru Grafuri Neorientate (ADFS)

Algoritmul **ADFS** menține o structură de date ce răspunde întrebărilor pentru LCA și nivelul strămoșilor. La inserarea în graf a unei muchii *(x, y)*, algoritmul ADFS verifică dacă muchia este de tipul *cross-edge* prin calcularea *w = LCA(x, y)*, și asigurându-se că w nu este egal cu *x* sau *y*.

În cazul în care muchia *(x, y)* este de tipul *back-edge*, se actualizează graful și se termină procesul.

În caz contrar, fie *u* și *v* copii ai lui *w*, unde $x \in T(u)$ și $y \in T(v)$, iar *x* va fi mai mic decât *y* în *T*. ADFS reconstruiește *T(v)*, atașându-l la muchia *(x, y)*. Acesta va inversa *path(x, y)*, ce convertește muchii de tip *back-edge* în muchii de tip *cross-edge*. Ulterior va colecta toate muchiile de tip *cross-edge* obținute, inserându-le ulterior in graf.

Singura diferență dintre ADFS1 și ADFS2 este în modul în care acești algoritmi procesează muchiile de tipul *cross-edge* colectate.

DFS Incremental având worst-case garantat (WDFS)

Cu toate că există mai mulți algoritmi de menținere incrementală a DFS, cel mai rău caz de timp pentru actualizarea arborelui DFS după o inserare a unei muchii este tot *O(m)*. Algoritmul **WDFS** vine însă cu un worst-case garantat de complexitate de timp $O(n \log^3 n)$.

Algoritmul construiește o structură de date folosind arborele DFS curent, care este folosit pentru a reconstrui eficient arborele DFS după adăugarea unei muchii. Chiar dacă construirea unei astfel de structuri se realizează în *O(m)*, ea trebuie actualizată periodic, odată la aproimativ *O(m/n)* operații, acestă technică numindu-se *reconstrucție periodică suprapusă* (overlapped periodic rebuilding).

3 Experimente pe Grafuri Neorientate Aleatoare

Experimentele fac referire la o comparație în ceea ce priveste rezultatele obținute de algoritmii de menținere incrementală a anui arbore DFS pe un graf neorientat aleatoriu.

În ceea ce privește experimentele, autorii au ales să compare rezultatele obținute pe baza numărului de muchii total procesate, și a numărului de muchii procesate la fiecare update. Astfel, în urma rezultatelor obținute, s-a remarcat faptul că performanțele algoritmilor SDFS, SDFS-Int și WDFS sunt foarte apropiate de limitele teoretice.

Surprinzător, performanțele algoritmilor ADFS1 și ADFS2 au fost diferite de cele așteptate. Atât ADFS1, cât și ADFS2 au performat mult mai rapid decât ceilalți algoritmi. Mai mult, algoritmii ADFS1 și ADFS2 au performat similar, în ciuda faptului că au complexități diferite ($O(n^{3/2}\sqrt{m})$ și $O(n^2/m)$). De notat este că algoritmii ADFS1 și ADFS2 au performat la fel de bine și mult mai rapid decât ceilalți algoritmi.

S-a mai observat că algoritmii ADFS procesează aproximativ 2 muchii per inserție după ce au fost inserate $O(n)$ muchii, iar când grafurile devin dense, numărul de muchii procesate de ADFS pentru actualizarea arborelui DFS ajunge asimptotic la 0.

4 Structura unui arbore DFS

Cum algoritmii SDFS, SDFS-Int și WDFS reconstruiesc arborele DFS de la început, odată cu adăugarea unei muchii noi, autorii au ales să se folosească de algoritmul ADFS, care reconstruiește arborele doar daca o muchie de tipul *cross-edge* este adăugată.

Fie *T* un arbore DFS pentru un graf aleatoriu *G(n, m)*. Fie p_c probabilitatea ca urmatoarea muchie adăugată să fie de tipul cross-edge in *T*. În urma unui studiu experimental, s-a observat că valoarea lui p_c scade atunci când graful devine mai dens. Asfel, ADFS se folosește de comportamentul pe care p_c îl are într-un graf aleatoriu.

Din studiile făcute pe structura arborilor DFS pentru grafuri aleatoare, s-a observat o proprietate a structurilor, numită **broomstick structure**.

Broomstick Structure

Structura unui arbore DFS poate fii descrisă ca una a unei maturi în felul următor: plecând de la rădăcina arborelui, există o cale în jos unde nu se găsesc ramificații, iar fiecare nod are exact un copil. Ne referim la această cale drept "**coadă**" (stick). Toate celelalte noduri și muchii vor fi de acum "**perii**" măturii (bristles).

Fie l_s lungimea *cozii* în structura arborelui DFS. În urma unor teste, s-a observat că *coada* apare după aproximativ $n \log n$ muchii (până atunci l_s este 0). Dupa aceasta, lungimea cozii crește rapid la aproape 90% din înălțimea ei după aproximativ $3n \log n$ muchii, urmând o creștere lentă, apropiindu-se de înălțimea maximă la aproximativ $O(n^2)$ muchii.

Lungimea cozii

Pentru a analiza lungimea lui l_s pentru $m = \Omega(n \log n)$ muchii, se demonstreaza mai întâi o limită succintă a probabilității existenței unui drum fără ramificații în timpul unei traversări DFS în *G(n, p)* prin teorema următoare.

Teoremă: Pentru un graf aleatoriu *G(n, p)*, cu $p = (\log n_0 + c)/n_0$, $n_0 \leq n$ și $c \geq 1$, exisă o cale fără remificații având lungimea de cel puțin $n - n_0$ în arborele DFS al lui *G*, cu probabilitatea de cel puțin $1 - 2e^{-c}$.

Pentru a se stabili un *tight-bound* a lunigimii cozii, trebuie selectată cea mai mică valoare pentru n_0 ce va satiface următoare condiție: Odată ce avem un drum DFS de lungime $n - n_0$ fără ramificări, subgraful indus de restul de n_0 vârfuri și ultimul vârf al acestei căi *v* va rămâne conectat.

La sfârșit s-a demonstrat că pentru graful *G(n, p)*, probabilitatea ca arborele DFS al său este o *mătură* (broomstick) cu lungimea cozii $\geq n - n_0$, este $1 - 3e^{-c}$.

Proprietățile structurii de coadă de matură

În urma analizei testelor obținute, autorii identifică 2 proprietăți pe care le are *structura de mătura* asupra algoritmilor ADFS:

- ADFS va reconstrui arborele DFS doar în cazul în care o muchie de tipul *cross-edge* a fost inserată.
- ADFS va modifica doar *perii* (bristles) arborelui DFS pentru a păstra coada intactă.

5 Algoritmi noi pentru grafuri aleatoare

Varianta simplă a SDFS (SDFS2) pentru grafuri neorientate aleatorii

Văzând performanțele pe care ADFS le obține datorită proprietăților enunțate mai devreme, se propune o variantă pentru SDFS inpirată dintr-o contrucție bazată pe peri, ce va satisface proprietățile ADFS. Practic, se vor reconstrui doar perii arborelui DFS după inserarea unor muchii cross-edge. Acesta se va realiza prin marcarea nodurilor din perii măturii ca fiind nevizitați, și performând DFS de la radpcina perilor.

Ca rezultat, se va obține un algoritm SDFS2 care va procesa doar muchiile din peri. Astfel, timpul pentru a adauga a insera $m = 2n \log n$ muchii va fi $O(m^2) = O(n^2 \log^2 n)$.

Autorii au observat că SDFS2 procesează $O(n^2)$ muchii similar cu ADFS. Acesta obține performanțe mult mai bune decât WDFS si SDFS-Int. Interesant este că în ciuda diferenței mare în numărul de muchii procesate de SDFS2 și ADFS, SDFS2 este mai rapid decât ADFS2 și este echivalent cu ADSF1 în practică.

Experimente pe grafuri orientate

Algoritmul propus SFDS2 functionează si pentru grafuri orientate. Astfel, proprietățile structurii măturii și analiza SDFS2 pot fi demonstrate și pentru grafurile orientate folosind argumente similare.

În urma unor teste, s-a observat că SFDS2 obține rezultate similare cu FDSF (algoritm specific grafurilor orientate). Din nou, în ciuda diferenței uriașe de muchii procesate de SDFS2 față de FDSF, acesta rămâne echivalent lui FDSF.

6 DFS Incremental pe grafuri reale

Majoritatea grafurilor care există în viața reală sunt diferite de cele generate aleatoriu pentru testarea algoritmilor. Cum pentru grafuri aleatoare, perii reprezentau intregul arbore DFS până la insearea a $\Theta(n \log n)$ muchii. Acest lucru forța SDFS2 sa reconstruiască întregul arbore, lucru ce dura $\Omega(n^2)$ timp chiar si pentru grafuri aleatoare, pe când ADFS și FDSF, care reconstruiau doar parțial arborele DFS, erau mult mai rapide. Astfel, atât ADFS și FDFS obțin performanțe mai bune decât SDFS2 sau alți algoritmi existenți.

Algoritm pentru grafuri reale (SDFS3)

Cum reconstruirea parțiala făcută de SFDS2 este semnificativă doar când *mătura* are o dimensiune apreciabilă, lucru care nu se întamplă foarte des cu grafurile reale, autorii propun crearea unui nou algoritm ce reconstruiește doar partea afectatp de muchia adaugată in arborele DFS.

Grafuri neorientate

La adăugarea unei muchii de tipul cross-edge *(x, y)*, ADFS reconstruiește unul din 2 subarbori candidați care se leagă de $LCA(x, y)$. Autorii propun algoritmul SDFS3 care va reconstrui doar subarborele care are cele mai puține noduri dintre cei doi subarbori. Nodurile acestuia vor fi marcate drept nevizitate, iar rezultatul DFS va fi legat muchiei *(x, y)*.

Grafuri orientate

La adăugarea unei muchii de tipul anti-cross *(x, y)*, FDSF reconstruieste vârfurile accesibile din *y* în subgraful indus de un *set candidat* de subarbori. Autorii propun ca SDFS3 să marcheze toți subarborii din acest set de candidați ca fiind nevizitați, urmând să continue din *(x, y)* cu traversarea.

Performanțe

În urma testelor făcute pe noul algoritm, autorii observă că SFDS3 este mai rapid de peste 10 ori decât SDFS2, dar mai lent de peste 30 de ori decât ADFS.

Observații pentru *grafuri neorientate*:

- ADFS depășește toți ceilalți algoritmi cu o marjă uriașă, ADFS1 fiind ușor mai bun decât ADFS2.
- SDFS2 îmbunătățește ușor SDFS, în timp ce SDFS3 îmbunătățește semnificativ SDFS2.
- WDFS are performanțe mai slabe decât SDFS pentru grafuri reale.

Observații pentru *grafuri orientate*:

- FDFS și SDFS3 depășesc toți ceilalți algoritmi, cu excepția cazului în care densitatea este mare, unde SDFS este mai bun.
- SDFS3 are performanțe mai bune decât FDFS în grafuri dense.
- SDFS2 îmbunătățește ușor SDFS, iar SDFS3 îmbunătățește ușor SDFS2.

7 Concluzii

În urma studiului produs de autori, aceștia au descoperit o proprietate importantă a arborilor DFS pentru un graf aleatoriu: *structura de mătură* (broomstick structure). Aceștia au demonstrat teoretic variația în lungime a cozii arborelui DFS pe măsură ce densitatea graficului crește, lucru ce s-a potrivit cu rezultatele experimentale. Aceasta a dus la crearea unui nou algoritm extrem de simplu SDFS2. Acest algoritm se potrivește teoretic și depășește experimental algoritmul de ultimă generație în grafice aleatorii dense. În cele din urmă, pentru grafurile din lumea reală, au propus un nou algoritm SDFS3 care funcționează mult mai bine decât SDFS2. În ciuda faptului că este extrem de simplu, aproape întotdeauna se potrivește cu performanța FDFS în grafurile orientate. Cu toate acestea, pentru grafurile neorientate, s-a constatat că ADFS depășește toți algoritmii (inclusiv SDFS3) cu o marjă uriașă motivând utilizarea sa în practică.