



# Algoritmi DFS

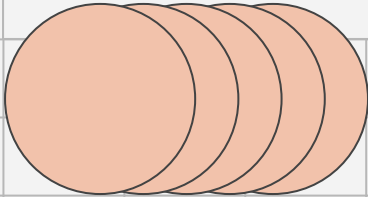
## Incrementali

Nicoleta Ciaușu

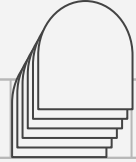


# Introducere

- Ce este algoritmul DFS?
- Ce facem atunci cand graful se modifică?
  - In mod normal, rerulăm DFS => ineficient
  - Ce ne dorim: Algoritmi online, cu o performanta mai buna la update/delete
- Vom vorbi despre algoritmi incrementali (doar insert)



# Observații



## Tipuri de muchii intr-un graf

O muchie  $(u,v)$  se raporteaza astfel față de arborele DFS al unui graf, poate fi:

- Forward edge: leaga nodul u de un descendent al acestuia, **compun arborele DFS**
- Back edge: leaga nodul u de un strămoș al acestuia
- Cross edge: leaga doua noduri care apartin a 2 subarbori diferiti - **inserarea lor in graf poate cere modificarea arborelui DFS**

# Muchii Cross-edge

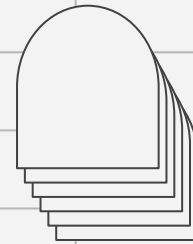
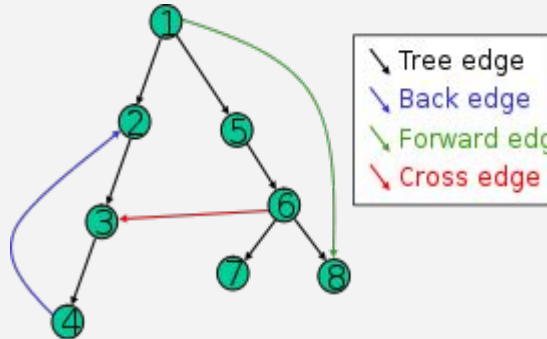
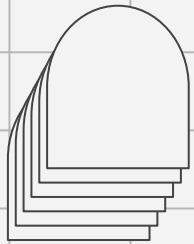
## Grafuri Orientate

Cross edge: de la un subarbor aflat în dreapta la un subarbor aflat în stanga -> **nu modifica arborele DFS**

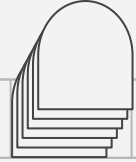
Anti-cross edge: de la stanga la dreapta, insertia în graf a unei astfel de muchii cere **recalcularea (parțială) a arborelui DFS**

## Grafuri Neorientate

Cross edge: muchie care conectează oricare doi subarbori ai arborelui DFS, insertia ei în graf cere **recalcularea (parțială) a arborelui DFS**



# SDFS/SDFS-Int



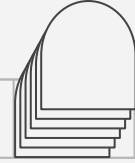
## I. SDFS

- La adaugarea unei muchii noi în graf, se recalculează întregul arbore DFS
- Folosit ca baseline pentru comparare

## II. SDFS-Int

- Optimizare a SDFS - oprește parcurgerea atunci când toate nodurile au fost vizitate

# FDFS

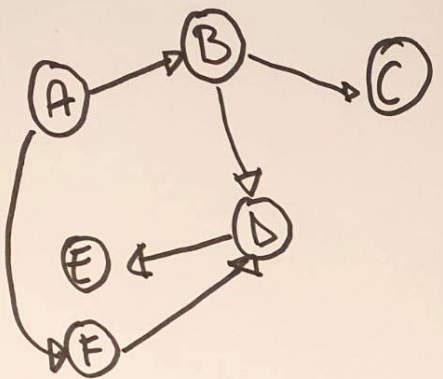


Algoritm pentru grafuri aciclice orientate (DAG)

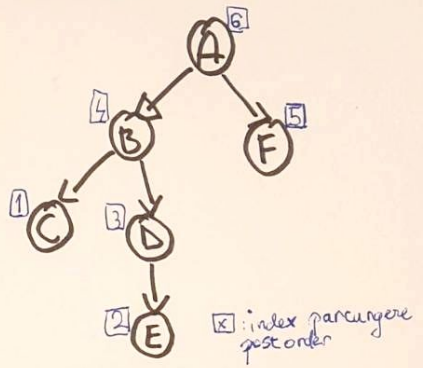
1. Se calculează arborele DFS uzual și numerotarea parcurgerii post-order a acestuia.
2. La adăugarea unei muchii noi în graf  $(x,y)$ , va fi identificat dacă aceasta este anti-cross edge.
  - a. **Dacă nu e**, poate fi adăugată în graf fără modificări ale arborelui DFS.
  - b. **Dacă este**, se vor determina nodurile afectate de modificare, vor fi marcate ca nevizitate, și se va rerula DFS pe acestea.

I

Graf initial:



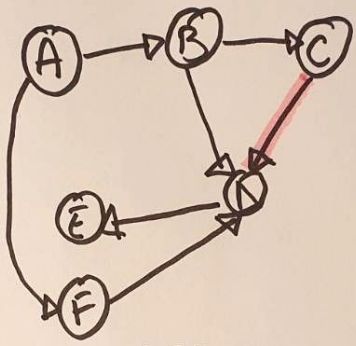
Arbore DFS:



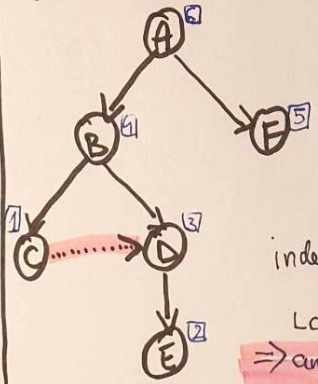
II

inserare muchie

$C \rightarrow D$  : Graful devine:



Arbore DFS:



$\text{index}(C) < \text{index}(D)$

$\text{LCA}(C, D) \neq C$  sau  $A$

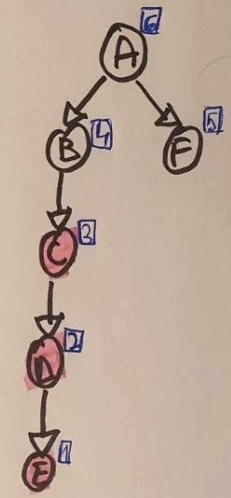
$\Rightarrow$  anti-cross edge!

III

Update DFS

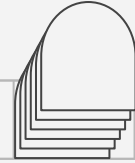
- se recalculeaza  
DFS doar pe nodurile  
cu index-ul mai mare  
 $\text{index}(C)$  si  $\text{index}(D)$   
(Visited = 0 si recalculat  
dfs).

Rezultat:



\* se redistribuie indexurile  
nodurilor modificate

# ADFS



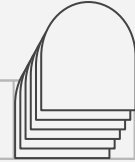
- Algoritm pentru grafuri neorientate
- Inspirat din FDFS
- Mentine, pe langa arbore, o structura de date ce raspunde eficient la query-uri  $LCA(x,y)$  si  $LA(x,k)$  (level ancestor - al k-lea stramos al lui x)

Cum funcționează - la inserarea unei muchii  $(x, y)$  in graf:

1. Determin daca muchia este cross edge (daca  $w = LCA(x,y)$ ,  $w \neq x$ ,  $w \neq y$ )
2. Daca este, atunci fie  $u$  radacina subarborelui ce il contine pe  $x$ ,  $v$  radacina subarborelui ce il contine pe  $y$ .



# ADFS



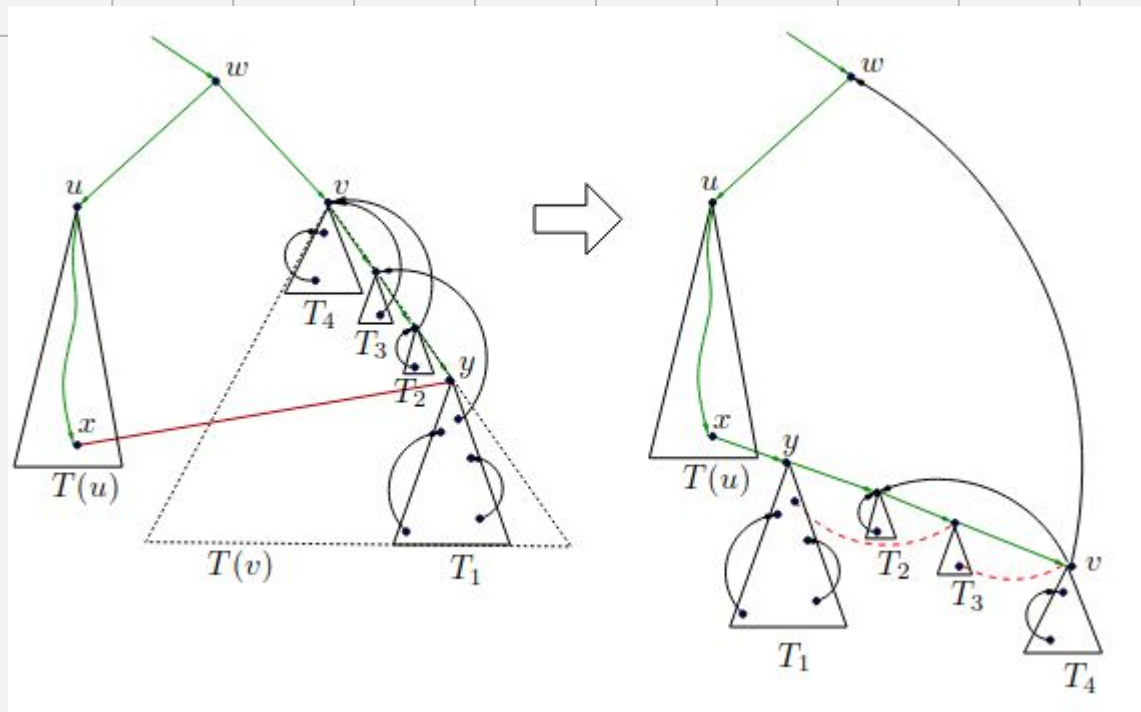
3. Fie  $L(x) > L(y)$  (unde  $L$  = nivelul in arbore).  $\Rightarrow$  subarborele cu radacina in  $y$  se va atasa la  $x$ . (din DFS traversal)

4. Se va “inversa”  $\text{path}(v, y)$ , adica subarborii ce apar pe nodurile ce compun  $\text{path}(v, y)$  vor fi reordonati.

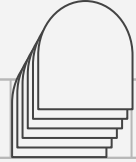
5. Prin reordonare, unele muchii back-edge din  $\text{path}(v, y)$  se transforma in muchii cross-edge, iar algoritmul va fi aplicat și pe acestea.

6. Algoritmul se termina atunci cand nu mai exista muchii cross edge in graf.

# ADFS



# ADFS 1 & ADFS 2



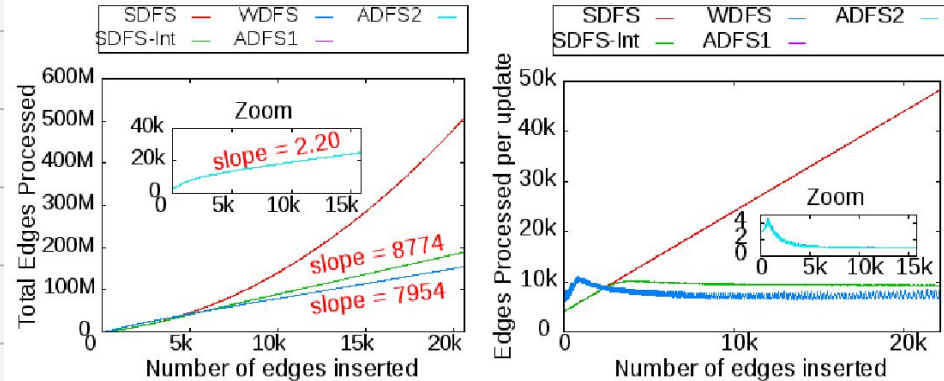
Ordinea in care sunt procesate muchiile cross-edge determina doua variante ale ADFS:

**ADFS1:** Muchiile sunt procesate arbitrar.

**ADFS2:** Este mentinuta o structura de date care ordoneaza muchiile a.i. sa fie procesate in ordinea nivelului in graf.

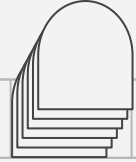
# Analiza experimentală a algoritmilor incrementalii

Se observa ca performanta in practica a ADFS1/2 este mai buna decat cea calculata teoretic, iar numarul de muchii modificate per update scade in loc sa creasca - ce se intampla?



Algorithm	Graph	Update time	Total time
SDFS [53]	Any	$O(m)$	$O(m^2)$
SDFS-Int [29]	Random	$O(n \log n)$ expected	$O(mn \log n)$ expected
FDFS [19]	DAG	$O(n)$ amortized	$O(mn)$
ADFS1 [6]	Undirected	$O(n^{3/2}/\sqrt{m})$ amortized	$O(n^{3/2}\sqrt{m})$
ADFS2 [6]	Undirected	$O(n^2/m)$ amortized	$O(n^2)$
WDFS [4]	Undirected	$O(n \log^3 n)$	$O(mn \log^3 n)$

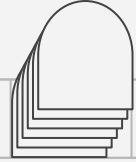
# Forma arborelui DFS



Se observa ca în arborele DFS asociat unui graf aleator exista un drum fără ramificări (adica: fiecare nod al drumului are un singur nod copil)

Cu putina imaginatie, acest drum poate fi văzut ca fiind "coada" unei mături (broomstick), celelalte noduri si muchii din graf constituind "perii" (bristles).

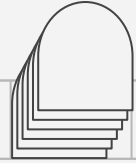
# Forma arborelui DFS



Autorii demonstreaza in mod probabilist ca:

- inserand muchii intr-un graf, pana cand acesta este conex, probabilitatea ca arborele DFS sa aiba forma de matura este 0.
- Din momentul in care arborele devine conex, probabilitatea incepe sa creasca, tinzand catre 1 cu cat graful este mai dens.

# Forma arborelui DFS



- In plus, cu cat graful este mai bine conectat, cu atat lungimea cozii crește.
- Aceasta forma de “coada de matura” este în spatele performanței foarte bune a ADFS. Cum coada devine din ce în ce mai lungă, iar ADFS rearanjeaza arborele doar cand atunci cand este inserat un cross-edge, de la un punct incolo, din ce în ce mai multe muchii vor fi back edges ce leaga un nod din “perii cozii” cu unul din “coada”.



# Noi algoritmi DFS incremental (SDFS 2)



In urma observatiei anterioare, se modifica SDFS astfel:

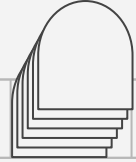
- La adaugarea unei noi muchii in graf, daca aceasta este un cross-edge pentru arborele DFS, se va recalcula doar partea de “matura” a grafului, ignorandu-se “coada”.
- In plus, se vor ignora muchiile care leaga “perii” arborelui de “coada” (fiind back-edges).

Acest algoritm simplu de implementat atinge performante mai bune decat SDFS, si comparabile cu ADFS, pe grafurile orientate aleatoare, avand o complexitate de  $O(n^2 \log^2 n)$ .





# Teste pe grafuri reale



✧ Toate testele anterioare au fost facute pe grafuri aleatoare. Pe grafuri reale, ADFS are performante mult mai bune decat SDFS2.

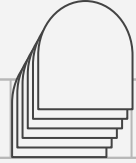
Ca imbunatatire, se incearca o euristica: la adaugarea unei noi muchii cross-edge  $(x, y)$  (unde  $T(x)$  subarborele din care face parte  $x$ ,  $T(y)$  subarborele din care face parte  $y$ ), SDFS3 va reconstrui doar subarborele cu numar mai mic de noduri.

La fel, nodurile acestuia vor fi marcate ca nevizitate, se va rerula DFS, iar arborele rezultat va fi atasat la muchia  $(x, y)$ .

Algoritmul rezultat este numit SDFS3.



# Concluzii



- A fost descoperita importanta structurii de mătură (broomstick structure) a arborilor DFS, si implicatiile acesteia.
- Au fost propusi doi algoritmi noi, SDFS2 si SDFS3, cu performanta foarte buna in cazul grafurilor orientate.
- Desi in cazul grafurilor neorientate, ADFS este superior din punct de vedere al performantei, SDFS3 se remarca prin simplitatea implementarii.