PL/SQL

—

Parcurgerea In Adancime: Comanda *Connect By*

Theodor Moroianu

January 5, 2021

Contents

1 Introducere

În acest scurt referat o să prezint implementarea si testarea performantelor algoritmului de DFS – numit si algoritmul de parcurgere in adancime.

Parcurgerea in adancime se refera la parcurgerea unui arbore in modul firesc: parcurgem primul fiu, urmat de al doilea etc.

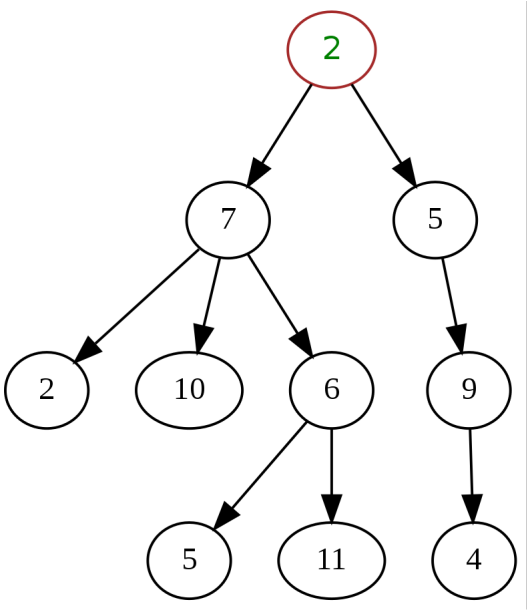

Figure 1: Reprezentarea unui arbore

2 Implementarea Bazei de Date pentru Testare

Pentru a putea testa diferite abordari ale algoritmului DFS, trebuie mai intai sa ne construim o baza de date care sa reprezinte un arbore.

Astfel, ne construim o relatie de *Tata*, adica pentru fiecare nod tinem minte care este tatal acestuia:

```
CREATE TABLE Adiacenta (
    Tata        NUMBER,              -- Tatal nodului
    Nod         NUMBER PRIMARY KEY   -- Nodul
);

-- Adaugam radacina, care nu are tata
INSERT INTO Adiacenta VALUES(NULL, 0);
```

Pentru a popula baza de date cu un numar relevant de noduri, am scris urmatorul script de *Python*:

```
import random

with open("script.sql", "w") as fout:
    for i in range(1, 10000):
        fout.write("INSERT INTO Adiacenta VALUES("
                + str(random.randint(0, i - 1))
                + ", " + str(i) + ");\n")
```

Se observa imediat ca adaugam in tabel valori de tipul (K, i), cu $k < i$, pentru fiecare i de la 1 la 10^4.

3 Parcurgerea DFS în PL/SQL

Codul care parcurge arborele, scris cu ajutorul unei functii recursive in PL/SQL este urmatorul:

```
CREATE OR REPLACE FUNCTION Dfs (
    Nod_p     NUMBER)
RETURN NUMBER
IS
    Numar     NUMBER := 1;
BEGIN
    FOR c IN (SELECT Nod
                FROM Adiacenta
                WHERE Tata = Nod_p) LOOP
        Numar := Numar + Dfs(c.Nod);
    END LOOP;

    RETURN Numar;
END;
/
```

Se poate vedea ca practic DFS-ul pentru un nod cauta toti fii acestuia, si se apeleaza recursiv.

4 Parcurgerea DFS în SQL

Codul care parcurge arborele, scris in SQL pur este surprinzator de simplu:

```
SELECT COUNT(*)
FROM Adiacenta
START WITH Nod = 0
CONNECT BY PRIOR Nod = Tata;
```

Selectul incepe de la nodul 0, si se extinde dupa urmatoarea regula: alege toate elementele din tabel care au *Tata* egal cu un nod deja vizitat.

5 Timpul de Executie

Am executat cei doi algoritmi, si le-am pus timpul de executare in tabelul alaturat.

Linia *SQL** corespunde codului cu instructiunea *CONNECT BY*, dar cu inlocuirea comenzii *COUNT(*)* cu *COUNT(1)* – o mica optimizare.

Tip	Timp De Executie
PL/SQL	1463 ms
SQL	14 ms
SQL*	12 ms

6 Concluzie

Uitandu-ne la timpii de executie, am extras urmatoarele concluzii:

- Baza de date Oracle reuseste sa execute *Select*-ul cu *Connect By* extrem de rapid, in opinia mea in timp liniar, adica $O(n)$. Pentru a face asta fie are cativa algoritmi complecsi in spatele comenzii *Connect By*, fie isi tine mai multe reprezentari ale tabelului *Adiacanta*, care sa usureze parcurgerea acestuia.
- DFS-ul implementat in PL/SQL face exact ceea ce i-am spus sa faca, adica parcurge tabelul intr-o complexitate de $O(n^2)$, adica pentru fiecare element din tabel isi cauta prin tot tabelul fiii.