Στρατηγική για παράλληλη εκτέλεση του αλγορίθμου Floyd-Warshall

Στα πλαίσια της αναφοράς αυτής ασχολούμαστε με 3 σειριακές εκδοχές του αλγορίθμου:

- Standard
- Recursive και
- Tiled

Για κάθε μία από τις παραπάνω θα προτείνουμε έναν τρόπο σκέψης που οδηγεί στην παράλληλη εκτέλεση.

Σε όλες τις εκδοχές λόγω της φύσης του αλγορίθμου χρειάζεται να υπολογιστεί ο πίνακας A_k πριν τον υπολογισμό του πίνακα A_k-1

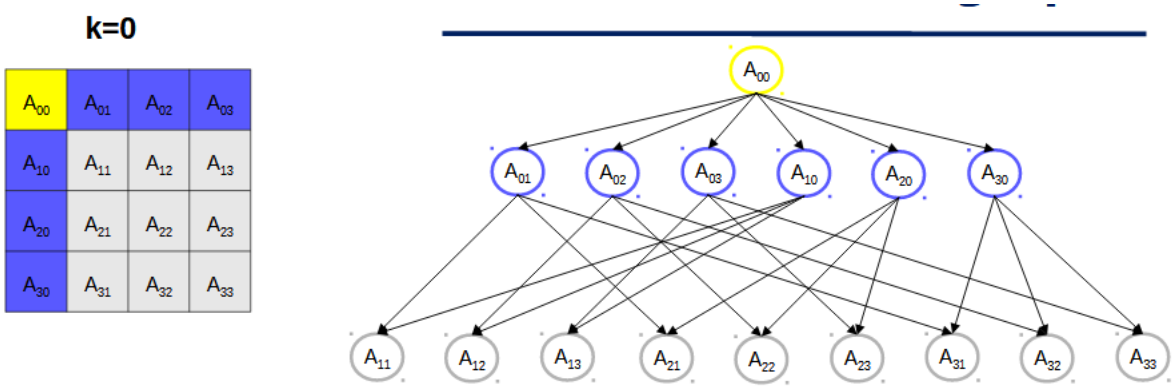
Standard έκδοση

Με το γράφο γειτνίασης στον πίνακα A μπορούμε να θεωρήσουμε τον αλγόριθμο ως εξής

```
for (k=0; k<N; k++){
    for (i=0; i<N; i++){
        for (j=0; j<N; j++){
            A[i][j] = min(A[i][j], A[i][k]+A[k][j]);
        }
    }
}
```

Όπου το **k** αναφέρεται στο χρονικό βήμα του αλγορίθμου και τα **i-j** τα ζευγάρια κόμβων.

Εάν οπτικοποιήσουμε τον πίνακα γειτνίασης A ως εξής, βλέπουμε ότι τα τετράγωνα που είναι στην ίδια στήλη και την ίδια γραμμή με το A[i][j] μπορούν να υπολογίζονται ταυτόχρονα, αφού δεν υπάρχει εξάρτηση μεταξύ τους παρά μόνο με το χρονικό βήμα k. Οπότε μπορούμε να πούμε πως οι λούπες i και j μπορούν να εκτελεστούν παράλληλα.



Recursive έκδοση

Θα φανταζόμαστε και εδώ πως χωρίζουμε τον πίνακα γειτνίασης ως εξής:

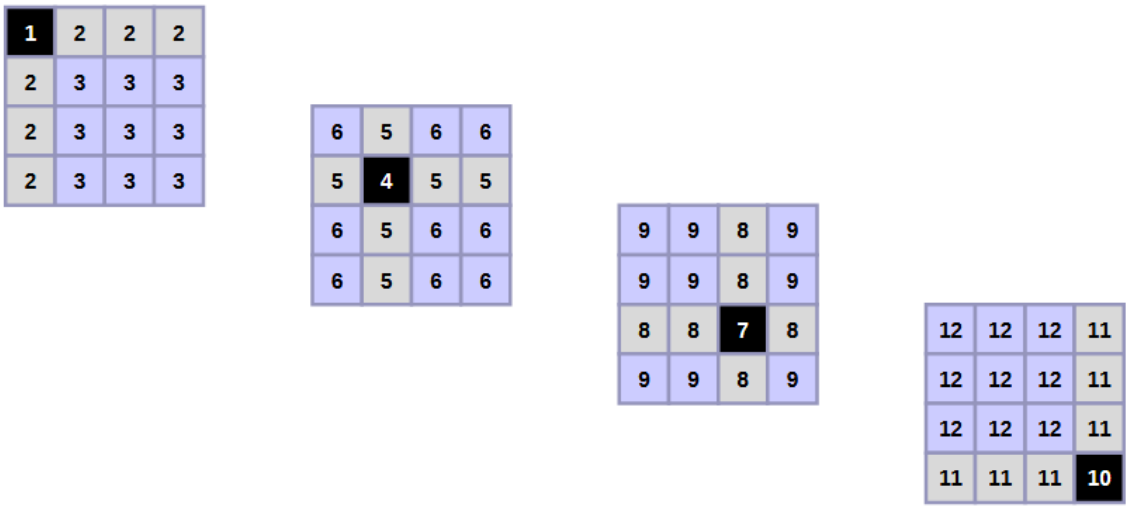
Ισχύει ότι ο πίνακας A_{00} πρέπει να υπολογιστεί πριν από τον A_{11}. Οι πίνακες A_{01} και A_{10} όμως είναι ανεξάρτητοι και άρα μπορούν να επεξεργαστούν ταυτόχρονα.

Η εκτέλεση του αλγορίθμου μοιάζει:

```
FWR (A, B, C)
    if (base case)
        FWI (A, B, C)
    else
        FWR (A00, B00, C00);
        FWR (A01, B00, C01); // Αυτές οι κλήσεις μπορούν να εκτελεστούν
παράλληλα!
        FWR (A10, B10, C00); // Αυτές οι κλήσεις μπορούν να εκτελεστούν
παράλληλα!
        FWR (A11, B10, C01);
        FWR (A11, B10, C01);
        FWR (A10, B10, C00); // Και αυτές οι κλήσεις μπορούν να εκτελεστούν
παράλληλα!
        FWR (A01, B00, C01); // Και αυτές οι κλήσεις μπορούν να εκτελεστούν
παράλληλα!
        FWR (A00, B00, C00);
```

Tiled έκδοση

Ο σειριακός αλγόριθμος tiled δουλεύει ως εξής: Χωρίζει σε υποπίνακες το πρόβλημα έστω μεγέθους `size`. Σε κάθε βήμα **k** αφού ανανεωθεί το A[i][j] στοιχείο, ανανεώνονται όλα τα στοιχεία στην ίδια στήλη και γραμμή, και στο τέλος ανανεώνει όλα τα υπόλοιπα στοιχεία του tile.



Βλέποντας την εξέλιξη του αλγόριθμου όπως παραπάνω, αρχίζουμε να σκεφτόμαστε τα πράγματα που θα μπορούσαν να τρέχουν παράλληλα. Όταν ο αλγόριθμος εξετάζει το tile 1, τότε σίγουρα δεν μπορούμε να πειράξουμε τα tiles της ομάδας 3 γιατί υπάρχει εξάρτηση. Μπορούμε όμως να υπολογίσουμε *παράλληλα* τα tiles της ομάδας 2, για τα οποία δεν υπάρχει εξάρτηση να μας κρατά πίσω

Αντίστοιχα στα υπόλοιπα βήματα, τα tiles της ίδιας στήλης και γραμμής μπορούν να επεξεργάζονται παράλληλα, ενώ μόλις τελειώσει η επεξεργασία και γίνει κάποιος συγχρονισμός στα δεδομένα τους, μπορούν να επεξεργαστούν και πάλι παράλληλα τα υπόλοιπα (διαγώνια) tiles του πίνακα.

Θα μπορούσαμε να δώσουμε έναν ψευδωκώδικα όπως:

```
for (k = 0; k < matrix_size; k++){
    FW(init_tile)
    for (tile in same row or column){
        new task FW(tile)
    sync_data()  \\ some kind of synchronization
    }
    for (tile in rest of tiles){
        new task FW(tile)
    }
    sync_data()  \\ some kind of synchronization
}
```

Καλή μας τύχη στο κομμάτι της υλοποίησης!