Εθνικό και Καποδιστριακό Πανεπιστήμιο Αθηνών
Τμήμα Φυσικής
ΔΠΜΣ Τμημάτων Φυσικής και Πληροφορικής & Τηλεπικοινωνιών
Κατεύθυνση Ηλεκτρονικού Αυτοματισμού

Ονοματεπώνυμο: Ιωάννης Κούτουλας
Αριθμός Μητρώου: 7110132300204
Μάθημα: "Δομές Δεδομένων & Αλγόριθμοι"
Εξάμηνο: Χειμερινό 2023-2024

Project 3

Παράλληλος Αλγόριθμος Levialdi για Μέτρηση και Labelling Connected-Component

Εισαγωγή

Σε αυτό το κείμενο θα αναπτυχθεί συνοπτικά ο παράλληλος αλγόριθμος του Stefano Levialdi. Ο αλγόριθμος, όταν παρουσιάστηκε αρχικά [1], είχε ως στόχο την μέτρηση connected component σε έναν πίνακα - matrix n x n (όπου n είναι δύναμη του 2). Το matrix αποτελείται από στοιχεία – pixel τα οποία λαμβάνουν είτε την τιμή '0' είτε την τιμή '1'. Στη δημοσίευση αναφέρεται το τί ορίζεται ως connected component σε μία τέτοια δομή και έπειτα αναπτύσσεται ο αλγόριθμος. Ο αλγόριθμος αυτός έχει περεταίρω χρήση, από την αρχική που προτάθηκε, για labeling ομάδων pixel, θεωρώντας την ομάδα σαν ένα connected component [2]. Η ερμηνεία αυτή βρίσκει χρήση στην παράλληλη επεξεργασία ψηφιακών εικόνων.

Αρχικά θα γίνει περιγραφή του αλγορίθμου για την μέτρηση connected component και έπειτα για component-labeling. Θα παρατεθούν και μερικά αποτελέσματα κώδικα που έγραψα για να στηρίξω το κείμενο. Ο κώδικας έχει γραφεί σε γλώσσα Python 3 [a], χρησιμοποιώντας το Jupyter Notebook [b]. Για την παράλληλη εκτέλεση του κώδικα χρησιμοποιείται η βιβλιοθήκη Joblib [c]. Γενικά ο υπολογιστής μου διαθέτει περιορισμένο αριθμό πυρήνων για την εκτέλεση του παράλληλου αλγορίθμου, όπως περιγράφεται στις αναφορές, αλλά η εν λόγω βιβλιοθήκη προσομοιώνει την παράλληλη εκτέλεση, δίνοντας σωστά αποτελέσματα. Κάποιες συναρτήσεις που χρησιμοποιώ εκτελούνται σειριακά και όχι παράλληλα. Οπότε η ταχύτητα εκτέλεσης του παράλληλου αλγορίθμου δεν προσομοιώνεται. Παραδίδω τον κώδικα χωριστά ενώ εδώ παρουσιάζω μόνο αποτελέσματά του, για να στηρίξω όσα αναφέρω.

Περιγραφή Δομής

Η δομή στην οποία εφαρμόζεται ο αλγόριθμος είναι ένας τετραγωνικός πίνακας – matrix n στοιχείων, άρα γίνεται λόγος για πίνακα διαστάσεων $\sqrt{n} \times \sqrt{n}$. Κάθε στοιχείο του πίνακα θεωρείται ως pixel που μπορεί να λάβει τις τιμές '0' ή '1'. Μία ομάδα στοιχείων με τιμή '1' μπορεί να θεωρηθεί connected component αν τα ενώνει αλληλουχία συνεχόμενων στοιχείων με τιμή '1'. Συνεχόμενα στοιχεία ορίζονται μέσω δύο ορισμών απόστασης από γειτονικά στοιχεία που έχουν την ίδια τιμή: d_1 και d_2[3]. Πρακτικά d_1-συνδεσιμότητα στοιχείων σημαίνει ό,τι στοιχείο με τιμή '1' θεωρείται συνδεδεμένο με άλλο στοιχείο με τιμή '1', αν αυτό είναι το αμέσως πάνω, κάτω, δεξιά η αριστερά στοιχείο. Η d_2-συνδεσιμότητα τα θεωρεί συνδεδεμένα στις ίδιες προαναφερθείσες περιπτώσεις, αλλά ισχύει και αν το πάνω δεξιά ή αριστερά, ή κάτω δεξιά ή αριστερά στοιχείο έχει την τιμή '1'. Ο αλγόριθμος αυτός μπορεί να δουλέψει και για τις δύο περιπτώσεις, αλλά περιγράφεται με d_2-συνδεσιμότητα.

Αλγόριθμος Levialdi για τη μέτρηση Connected Component

Στόχος είναι να βρεθεί το πλήθος των connected component στον πίνακα. Ο αλγόριθμος που προτείνεται είναι παράλληλος αλγόριθμος και προϋποθέτει δυνατότητα παράλληλης επεξεργασίας όλων των n στοιχείων του πίνακα. Οπότε σε κάθε κύκλο επεξεργασίας – σε κάθε βήμα του αλγορίθμου - μπορούν να έχουν ενημερωθεί όλα τα στοιχεία του πίνακα με νέες τιμές, ταυτόχρονα. Στόχος είναι σε κάθε βήμα του αλγορίθμου κάθε connected component να συρρικνώνεται, έως ότου μετά από κάποιους κύκλους να γίνουν μοναδικά στοιχεία με τιμή '1', τα οποία στον επόμενο κύκλο θα γίνουν μηδενικά στοιχεία και τότε θα μετρηθούν.

Η κατάσταση του κάθε στοιχείου του πίνακα (δηλαδή το τι τιμή έχει σε κάθε βήμα) εκφράζεται σαν συνάρτηση των τιμών ενός "παραθύρου" γειτονικών στοιχείων. Εδώ επιλέγω ως παράθυρο ένα πίνακα 2 x 2 όπου το στοιχείο προς επεξεργασία βρίσκεται στην πάνω δεξιά γωνία, αλλά γενικά μπορεί να επιλέξει κανείς οποιοδήποτε 2 x 2 παράθυρο με το προς επεξεργασία στοιχείο στη μία γωνία. Η τιμή του κάθε στοιχείου εξαρτάται από τις τιμές του αριστερά, κάτω και κάτω αριστερά γειτονικού στοιχείου, δηλαδή των υπολοίπων στοιχείων του παραθύρου. Ο Levialdi περιγράφει μία συνάρτηση κατάστασης φ που χρησιμοποιεί τη συνάρτηση Heaviside [d] και σε κάθε κύκλο εφαρμόζεται παράλληλα σε όλα τα στοιχεία του πίνακα.

$$\varphi(a_{i,j}) = h\left(h(a_{i,j-1} + a_{i,j} + a_{i+1,j} + -1) + h(a_{i,j} + a_{i+1,j-1} - 1)\right)$$

Η πρακτική περιγραφή είναι ότι αν το στοιχείο έχει την τιμή '1' στον επόμενο κύκλο επεξεργασίας θα αποκτήσει την τιμή '0', αν ο αριστερά, κάτω και κάτω αριστερά γείτονας έχουν την τιμή '0'. Αν έχει την τιμή '1' θα αποκτήσει την τιμή '0', στο επόμενο βήμα του αλγορίθμου, αν ο αριστερά και ο κάτω γείτονας έχουν την τιμή 1.

```
When the window is :
 [[0 1]
 [0 0]]
The upper right will change from 1 to 0

When the window is :
 [[1 0]
 [0 1]]
The upper right will change from 0 to 1

When the window is :
 [[1 0]
 [1 1]]
The upper right will change from 0 to 1
```

Βασική παραδοχή είναι ότι αν κάποιο στοιχείο βρίσκεται στα σύνορα του πίνακα, τότε θεωρείται ότι έχει γειτονικά στοιχεία με τιμή '0' στη θέση των συνόρων. Άρα ένα στοιχείο στο αριστερό σύνορο του πίνακα θεωρείται ότι αριστερά, πάνω αριστερά και κάτω αριστερά γειτνιάζει με στοιχεία με τιμή '0'. Μία άλλη βασική παραδοχή, που αναφέρθηκε ήδη, είναι ότι ένα connected component που έχει συρρικνωθεί σε ένα στοιχείο με τιμή '1' πρέπει να μετρηθεί και στο επόμενο βήμα να διαγραφεί, αλλιώς ο αλγόριθμος μπορεί να δώσει λάθος αποτελέσματα. Ένα παράδειγμα εξέλιξης του αλγορίθμου δίνεται παρακάτω για έναν πίνακα 4 x 4.

```
Matrix before:
 [[1 1 1 0]
 [0 0 0 0]
 [1 1 0 1]
 [1 1 0 1]]

Matrix after 1 iterrations:
 [[0 1 1 0]
 [0 0 0 0]
 [1 1 0 1]
 [0 1 0 0]]
Element with indexes i = 2 , j = 3
Component count: 1
```

```
Matrix after 2 iterrations:
 [[0 0 1 0]
 [0 0 0 0]
 [0 1 0 0]
 [0 0 0 0]]
Element with indexes i = 0 , j = 2
Element with indexes i = 2 , j = 1
Component count: 2


Matrix after 3 iterrations:
 [[0 0 0 0]
 [0 0 0 0]
 [0 0 0 0]
 [0 0 0 0]]
Component count: 0


General component number:  3
```

Προκειμένου η συρρίκνωση τον connected components να είναι επιτυχημένη πρέπει, μετά από οσαδήποτε βήματα, να ικανοποιεί δύο συνθήκες. Πρέπει να μη χωρίζει στοιχεία που ανήκουν στο ίδιο connected component σε χωριστά connected components, δηλαδή να μη “σπάει” τα connected components. Από την άλλη πρέπει και να μην ενώνει ένα η περισσότερα connected components, τα οποία πριν την εφαρμογή του αλγορίθμου δεν ήταν ενωμένα. Αν μία από αυτές τις συνθήκες δεν ικανοποιείται θα μετρηθεί λάθος πλήθος connected components.

Η απόδειξη ότι ο αλγόριθμος δεν “σπάει” τα connected components βασίζεται στην παρατήρηση ότι υπάρχει μόνο ένας συνδυασμός τιμών στοιχείων γύρω από ένα δεδομένο στοιχείο που ενδέχεται να σπάσει ένα connected component. Στο συνδυασμό αυτό υπάρχει ένα στοιχείο με τιμή ‘1’, του οποίου η τιμή αν αλλάξει σε ‘0’ το connected component θα σπάσει σε δύο στοιχεία. Αυτός φαίνεται στο αποτέλεσμα του παρακάτω κώδικα. Στον αρχικό πίνακα (Matrix before) το στοιχείο ενδιαφέροντος είναι το $\alpha_{2,2}$ (θεωρώ ότι η πρώτη σειρά και η πρώτη στήλη έχουν την αρίθμηση 1). Βλέπουμε ότι ο αλγόριθμος όμως δεν σπάει τα connected component σε αυτήν την περίπτωση. Στον πίνακα που προκύπτει στο επόμενο βήμα (Matrix after), όταν το $\alpha_{2,2}$ γίνει ‘0’, μερικά άλλα γειτονικά στοιχεία θα γίνουν ‘1’ από τις ιδιότητες του αλγορίθμου και το component παραμένει d_2-συνδεδεμένο.

```
Matrix before:
 [[1 0 0]
 [0 1 0]
 [0 0 1]]

Matrix after:
 [[1 1 0]
 [0 0 1]
 [0 0 1]]
```

Κίνδυνος να ενωθούν δύο connected component υπάρχει όταν ένα στοιχείο αλλάζει τιμή από ‘0’ σε ‘1’. Αυτό γίνεται αν το στοιχείο έχει αριστερά και κάτω γειτονικά στοιχεία με τιμή ‘1’ και στον επόμενο κύκλο η τιμή του θα γίνει ‘1’. Αν το στοιχείο έχει πάνω δεξιά γείτονα με τιμή ‘1’, αυτός θα ανήκει σε άλλο connected component, μα αν το στοιχείο μας αλλάξει τιμή σε ‘1’ μπορεί να ενωθούν τα αρχικά χωριστά connected components που τα αποτελούσαν. Αυτός είναι και ο μόνος συνδυασμός που παρουσιάζει κίνδυνο γιατί σε οποιαδήποτε άλλη περίπτωση είτε τα δύο στοιχεία θα ανήκαν ήδη στο ίδιο connected component ή δε θα υπήρχε κίνδυνος ένωσης. Ο αλγόριθμος όμως δε θα το επιτρέψει αυτό, μηδενίζοντας την τιμή του πάνω δεξιά στοιχείου στον επόμενο κύκλο. Το στοιχείο ενδιαφέροντος στους πίνακες του κάτω κώδικα είναι το $\alpha_{3,2}$.

```
Matrix before:
 [[0 1 1 1]
 [0 0 1 1]
 [1 0 0 1]
 [1 1 0 0]]

Matrix after:
 [[0 1 1 1]
 [0 0 0 1]
 [1 1 0 1]
 [1 1 0 0]]
```

Καθώς ο αλγόριθμος εκτελείται μπορούμε να θεωρήσουμε ότι κάθε απομονωμένο component (που εξ ορισμού είναι ένας σχηματισμός '1'-στοιχείων περιτριγυρισμένων από '0'-στοιχεία) έχει ένα μέτωπο κάτω και αριστερά στοιχείων τα οποία σε κάθε βήμα μετατρέπονται σε μηδενικά στοιχεία. Άλλα στοιχεία, που βρίσκονται στο κέντρο του component και είναι περιτριγυρισμένα από '1'-στοιχεία διατηρούν την τιμή τους, έως ότου το μέτωπο των κάτω αριστερά στοιχείων τα φτάσει και αρχίσει να τα μηδενίζει. Νησίδες '0'-στοιχείων, περιτριγυρισμένων από '1' στοιχεία θα μετατραπούν σε '1' στοιχεία, μέχρι να ξαναμετατραπούν σε '0' στοιχεία όταν τα φτάσει το κάτω αριστερά μέτωπο. Τέλος υπάρχει και ένα μέτωπο άνω δεξιά στοιχείων τα οποία ενδέχεται στην αρχή να είναι μηδενικά, αλλά θα μετατραπούν σε '1'-στοιχεία καθώς συνορεύουν με '1'-στοιχεία του κέντρου του component και από την ιδιότητα του αλγορίθμου οι κάτω αριστερά γείτονες τείνουν γενικώς να μεταδίδουν την κατάστασή τους στους άνω δεξιά. Εν τέλει όλα τα στοιχεία θα καταλήξουν στο άνω δεξιά σύνορο, έως ότου συρρικνωθούν σε ένα μόνο στοιχείο με τιμή '1', το οποίο τελικά αφανίζεται. Το συμπέρασμα είναι ότι η συρρίκνωση γίνεται από κάτω αριστερά προς τα πάνω δεξιά.

```
Matrix before:
 [[1 1 1 0]
 [1 0 0 1]
 [1 1 0 1]
 [0 1 1 1]]
```

```
Matrix after 1 iterrations:
 [[1 1 1 1]
 [1 1 0 1]
 [0 1 1 1]
 [0 0 1 1]]
```

```
Matrix after 2 iterrations:
 [[1 1 1 1]
 [0 1 1 1]
 [0 0 1 1]
 [0 0 0 1]]
```

```
Matrix after 3 iterrations:
 [[0 1 1 1]
 [0 0 1 1]
 [0 0 0 1]
 [0 0 0 0]]
```

```
Matrix after 4 iterrations:
 [[0 0 1 1]
 [0 0 0 1]
 [0 0 0 0]
 [0 0 0 0]]
```

```
Matrix after 5 iterrations:
 [[0 0 0 1]
 [0 0 0 0]
 [0 0 0 0]
 [0 0 0 0]]
```

```
Matrix after 6 iterrations:
 [[0 0 0 0]
 [0 0 0 0]
 [0 0 0 0]
 [0 0 0 0]]
```

Βάσει της προηγούμενης περιγραφής υπολογίζονται και τα μέγιστα βήματα για τη συρρίκνωση ενός connected component. Μπορούμε να θεωρήσουμε ότι κάθε απομονωμένο component ανήκει σε ένα δικό του υποπίνακα, τέτοιον ώστε να το περιέχει ακριβώς (ουσιαστικά ένα περιγεγραμμένο τετράγωνο στο component). Τα περισσότερα στοιχεία του κάτω αριστερά μετώπου που περιέγραψα πριν θα ανήκουν στο κάτω αριστερά σύνορο του τετραγώνου, και σε κάθε βήμα θα μετατρέπονται όλα σε '0' στοιχεία. Ένα από αυτά τα στοιχεία θα έχει τη μέγιστη d_1-απόσταση από το στοιχείο που βρίσκεται στην άνω δεξιά γωνία του περιεγραμμένου τετραγώνου, και θα το αναφέρω ως *στοιχείο γωνίας*. Αυτό το στοιχείο το αναφέρω γιατί είναι σημαντικό: βάσει των προαναφερθέντων ο αλγόριθμος θα συρρικνώσει το component σε ένα στοιχείο, το οποίο πράγματι θα είναι το ανώτερο δεξιά στοιχείο του υποπίνακα. Σε κάθε βήμα το κάτω αριστερά μέτωπο μειώνεται και η απόσταση του πιο απομακρυσμένου στοιχείου από το στοιχείο γωνίας μειώνεται κατά 1, έως ότου μηδενιστεί όταν το ίδιο στοιχείο γωνίας αφανιστεί και μετρηθεί το component. Άρα η d_1-απόσταση του πιο απομακρυσμένου στοιχείου από το στοιχείου γωνίας συμπίπτει με τα βήματα που απαιτούνται για να υπολογιστεί το component! Στη χειρότερη περίπτωση λοιπόν όπου ο συνολικός πίνακας $\sqrt{n} \times \sqrt{n}$ θα αποτελείται εξ ολοκλήρου από ένα μόνο connected – component θα χρειαστούν $2\sqrt{n} - 1$ βήματα. Άρα ο αλγόριθμος είναι πολυπλοκότητας $\mathcal{O}(\sqrt{n})$. Η χειρότερη περίπτωση παρουσιάζεται στον κάτω πίνακα για έναν πίνακα 4 x 4. Απαιτούνται $2\sqrt{16} - 1 = 7$ βήματα

```
Matrix before:
 [[1 1 1 1]
 [1 1 1 1]
 [1 1 1 1]
 [1 1 1 1]]

Matrix after 1 iterrations:
 [[1 1 1 1]
 [1 1 1 1]
 [1 1 1 1]
 [0 1 1 1]]

Matrix after 2 iterrations:
 [[1 1 1 1]
 [1 1 1 1]
 [0 1 1 1]
 [0 0 1 1]]

Matrix after 3 iterrations:
 [[1 1 1 1]
 [0 1 1 1]
 [0 0 1 1]
 [0 0 0 1]]

Matrix after 4 iterrations:
 [[0 1 1 1]
 [0 0 1 1]
 [0 0 0 1]
 [0 0 0 0]]

Matrix after 5 iterrations:
 [[0 0 1 1]
 [0 0 0 1]
 [0 0 0 0]
 [0 0 0 0]]

Matrix after 6 iterrations:
 [[0 0 0 1]
 [0 0 0 0]
 [0 0 0 0]
 [0 0 0 0]]

Matrix after 7 iterrations:
 [[0 0 0 0]
 [0 0 0 0]
 [0 0 0 0]
 [0 0 0 0]]
```

Χρήση του Αλγορίθμου Levialdi για Component-Labelling

Οι ψηφιακές εικόνες μπορούν να αναπαρασταθούν ως πίνακες από pixel και στην επεξεργασία τους συχνά το ζητούμενο είναι ο εντοπισμός δομών, οι οποίες διαφοροποιούνται από τις μεγαλύτερες ή μικρότερες τιμές των pixel τους (στην πιο ακραία περίπτωση ένα pixel θα έχει είτε τη μέγιστη δυνατή τιμή ή την ελάχιστη, οπότε μπορούν να αναλογιστούν με τιμές ‘1’ ή ‘0’). Οι δομές αυτές μπορούν να θεωρηθούν σαν connected-components και ο εντοπισμός τους γίνεται με την απόδοση label σε κάθε pixel· στοιχεία με ίδιο label ανήκουν στο ίδιο connected component.

Ο αλγόριθμος του Levialdi μπορεί να χρησιμοποιηθεί για component-labelling, με μερικές προσθήκες. Ο αλγόριθμος πλέον θα χωρίζεται στη *φάση της συρρίκνωσης* και στη *φάση της επέκτασης*. Η φάση της συρρίκνωσης είναι ο αλγόριθμος ακριβώς όπως περιγράφηκε προηγουμένως με τη διαφορά ότι όταν το component συρρικνωθεί μέχρι να αφανιστεί, αντί να μετρηθεί, αποδίδεται ένα label στο τελευταίο μοναδικό ‘1’-στοιχείου του. Στη φάση της επέκτασης απλώς γίνεται η ανάποδη διαδικασία, και τα component ξεκινάνε από το τελικό ‘1’-στοιχείο τους και επεκτείνονται στην αρχική τους δομή. Η προσθήκη είναι ότι από το τελικό ‘1’-στοιχείο αποδίδεται και σε όλα τα γειτονικά στοιχεία του παραθύρου του το label που έλαβε κατά την συρρίκνωση. Σε κάθε βήμα επέκτασης κάθε στοιχείο περνάει το label του σε όλα τα γειτονικά στοιχεία παραθύρου του που αποκτούν την τιμή ‘1’.

Η φάση της επέκταση απαιτεί ακριβώς τον ίδιο αριθμό βημάτων με τη φάση της συρρίκνωσης. Άρα στη χειρότερη περίπτωση απαιτούνται και για τις δύο φάσεις $\left(2\sqrt{n}-1\right)+\left(2\sqrt{n}-1\right)=4\sqrt{n}-2$ βήματα. Άρα η πολυπλοκότητα του αλγορίθμου παραμένει $\mathcal{O}(\sqrt{n})$. Για να επιτευχθεί η φάση της επέκτασης πρέπει κάθε επεξεργαστής (ή κάθε διεργασία, νήμα κτλ.) που επεξεργάζεται την κατάσταση του κάθε στοιχείου - pixel να έχει τη δυνατότητα να αποθηκεύει όλες τις τιμές του pixel, από την αρχική του τιμή μέχρι την λήξη της φάσης της συρρίκνωσης, άρα στη χειρότερη $2\sqrt{n}$ τιμές. Στη φάση της επέκτασης σε κάθε κύκλο απλώς θα επιστρέφει στις προηγούμενες τιμές του, που έχουν αποθηκευτεί – μία σε κάθε βήμα.

Βιβλιογραφικές Αναφορές

[1] S. Levialdi. On Shrinking Binary Picture Patterns. *Laboratorio di Cibernetica del C.N.R. Napoli, Italy,* January 1972

[2] F. Thomson Leighton. Introduction to Parallel Algorithms and Architectures. *Morgan Kaufmann Publishers, San Mateo, California,* 1992

[3] A. Rosenfeld, J.L. Pfaltz. Distance functions on digital pictures. Pattern Recognition. July 1968

Άλλες Αναφορές

[a] Python 3 documentation: https://docs.python.org/3.0/

[b] Jupyter Notebook documentation: https://docs.jupyter.org/

[c] Joblib Library documentation: https://joblib.readthedocs.io/

[d] Heaviside step function: https://en.wikipedia.org/wiki/Heaviside_step_function