ΠΑΡΑΛΛΗΛΗ ΕΠΕΞΕΡΓΑΣΙΑ

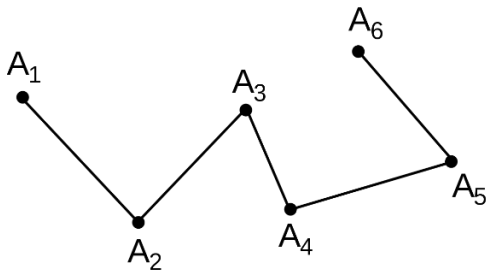
ΑΚΑΔΗΜΑΪΚΟ ΕΤΟΣ 2018/2019

Εισαγωγή

Στα πλαίσια της άσκησης θα ασχοληθούμε με το πρόβλημα της απλούστευσης τεθλασμένων γραμμών (polylines ή «πολυγραμμών»), το οποίο αποτελεί απαραίτητη επεξεργασία στην χαρτογραφία, αλλά η χρήση του μπορεί να γενικευθεί και σε άλλα πεδία.

Ο αλγόριθμος Ramer–Douglas–Peucker

Ο αλγόριθμος Ramer–Douglas–Peucker (συχνά αναφέρεται και ως αλγόριθμος Douglas-Peucker) [1] χρησιμοποιείται για την απλοποίηση μιας «πολυγραμμής» (polyline). Μια πολυγραμμή είναι το σχήμα που προκύπτει από την ένωση σημείων με ευθύγραμμα τμήματα, όπως στο παρακάτω σχήμα:



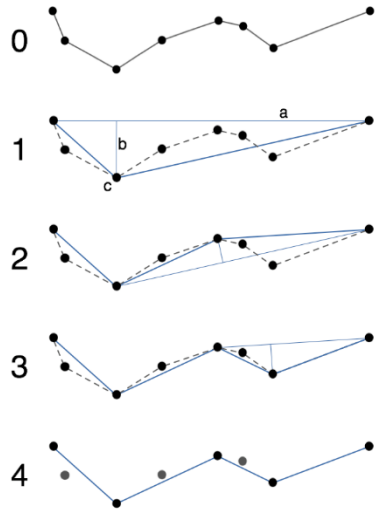
Σκοπός χρήσης του αλγορίθμου είναι τυπικά η εξάλειψη λεπτομερειών που δεν είναι απαραίτητες. Για παράδειγμα, τα σύνορα μιας χώρας και των περιφερειών της μπορούν να αναπαρασταθούν ως περισσότερες πολυγραμμές, δεδομένων των συντεταγμένων των σημείων που αποτελούν τα σύνορα αυτά. Ωστόσο, η απεικόνιση των συνόρων σε έναν χάρτη μπορεί να είναι υπερβολικά λεπτομερής αν χρησιμοποιηθούν όλα τα σημεία που ορίζουν τα σύνορα (αριστερό τμήμα της εικόνας που ακολουθεί). Η απεικόνιση είναι ικανοποιητική και με μικρότερη ακρίβεια, δηλαδή με την αφαίρεση κάποιων σημείων (δείτε δεξί τμήμα της εικόνας που ακολουθεί).

(a) Original data

(b) Tolerance is 10.0 kilometers

Ο αλγόριθμος Ramer–Douglas–Peucker αποφασίζει ποια σημεία της πολυγραμμής μπορούν να αφαιρεθούν ώστε να επιτευχθεί προσέγγιση της αρχικής πολυγραμμής με μια νέα πολυγραμμή που περιέχει λιγότερα σημεία και με συγκεκριμένη μέγιστη απόκλιση των σημείων που αφαιρούνται.

Η λογική του αλγορίθμου είναι απλή. Δεδομένης μιας πολυγραμμής, ο αλγόριθμος θεωρεί την ευθεία που συνδέει το πρώτο με το τελευταίο σημείο της. Για όλα τα ενδιάμεσα σημεία βρίσκει αυτό που απέχει περισσότερο από αυτήν την ευθεία. Αν η απόσταση αυτού του σημείου από την προαναφερθείσα ευθεία είναι μικρότερη από την επιτρεπτή απόκλιση, τότε όλα τα ενδιάμεσα σημεία αφαιρούνται και η πολυγραμμή αντικαθίσταται από την ευθεία. Αν η απόσταση είναι μεγαλύτερη, τότε η αρχική πολυγραμμή χωρίζεται σε δύο μικρότερες πολυγραμμές. Η πρώτη αποτελείται από όλα τα σημεία της αρχικής πολυγραμμής ξεκινώντας από το πρώτο μέχρι και το σημείο που έχει την μεγαλύτερη απόσταση από την ευθεία. Η δεύτερη πολυγραμμή αποτελείται από το σημείο που έχει την μεγαλύτερη απόσταση από την ευθεία και όλα τα σημεία της αρχικής πολυγραμμής μέχρι και το τελευταίο. Ο αλγόριθμος επαναλαμβάνεται στην συνέχεια αναδρομικά για κάθε μια από τις νέες πολυγραμμές.



Ας πάρουμε το παράδειγμα του παραπάνω σχήματος. Στην αρχική πολυγραμμή συνδέουμε το πρώτο και το τελευταίο σημείο (ευθεία a) και υπολογίζουμε το σημείο c με την μεγαλύτερη απόσταση b από την ευθεία αυτή. Υποθέτοντας πως η απόσταση αυτή είναι μεγαλύτερη από την μέγιστη επιτρεπτή απόκλιση, χωρίζουμε την πολυγραμμή σε δύο μικρότερες πολυγραμμές και επαναλαμβάνουμε την διαδικασία για κάθε μια από τις νέες πολυγραμμές, όπως αναφέρθηκε παραπάνω. Στο τέλος του συγκεκριμένου παραδείγματος έχουν αφαιρεθεί 3 σημεία από την αρχική πολυγραμμή.

Θεωρώντας πως μια πολυγραμμή αναπαρίσταται ως μια ακολουθία (λίστα) από συντεταγμένες σημείων (vertex), ο αλγόριθμος μπορεί να εκφραστεί όπως παρακάτω:

```
Input:  AllPolylines     # List than contains lists of coordinates, a.k.a. multiple polylines
Input:  epsilon          # Maximum allowed distance of vertex from line
Output: SimpPolylines    # List that contains the simplified polylines

function RDP(polyline, epsilon)
Input Arguments:   polyline to be simplified
                   epsilon
Return value:      Simplified polyline

     start ← first vertex of polyline
     end   ← last vertex of polyline
     maxDist ← 0
     for each vertex in polyline
          dist ← distance of vertex from line defined by start and end
          if dist > maxDist
               maxDist ← dist
               remote_vertex ← vertex
          end if
     end for
     if maxDist > epsilon
          polyline1 ← vertices of polyline from start until and including remote_vertex
          polyline2 ← vertices of polyline from remote_vertex until and including end
          simple_polyline1 ← RDP(polyline1, epsilon)
          simple_polyline2 ← RDP(polyline2, epsilon)
          return simple_polyline1 – last vertex of simple_polyline1 + simple_polyline2
     else
          return start + end
     end if
end function

for each polyline in AllPolylines
     simple_polyline ← RDP(polyline, epsilon)
     print simple_polyline
     SimpPolylines ← SimpPolylines + simple_polyline
end for
```

Αναφορές

[1] D. H. Douglas and T. K. Peucker, "Algorithms for the reduction of the number of points-required to represent a line or its caricature". The Canadian Cartographer, vol. 10, no. 2, 1973, pp. 112-122.

Ζητούμενα της άσκησης

Στα πλαίσια της άσκησης θα σας δοθεί μια σειριακή υλοποίηση του αλγορίθμου γραμμένη σε C++. Σκοπός σας θα είναι να δημιουργήσετε ένα σύνολο παράλληλων υλοποιήσεων του αλγορίθμου και να τις συγκρίνετε ως προς την επίδοση τους.

1) **(10%) Στατική ανάθεση φόρτου εργασίας σε βρόχο for (Πρόγραμμα "Static")**

   Ζητείται να παραλληλοποιήσετε τον αλγόριθμο Ramer–Douglas–Peucker ως προς τις πολλαπλές πολυγραμμές που περιέχονται στην λίστα AllPolylines και πρέπει να επεξεργαστούν. Ας πάρουμε για παράδειγμα μια περίπτωση στην οποία η λίστα AllPolylines περιέχει 100 πολυγραμμές και πως στο πρόγραμμα που θα υλοποιήσετε έχουν δημιουργηθεί 4 νήματα. Η πρώτη λογική παραλληλοποίησης που θα πρέπει να ακολουθήσετε είναι κάθε νήμα να αναλάβει με **στατικό τρόπο ανάθεσης** να κάνει απλοποίηση για 100/4 = 25 συνεχόμενες πολυγραμμές στην λίστα AllPolylines. Στην συνέχεια, κάθε νήμα θα εκτελεί σειριακά τον αλγόριθμο Ramer–Douglas–Peucker για κάθε πολυγραμμή που ανέλαβε να επεξεργαστεί.

   - Ποιες οδηγίες του προτύπου OpenMP πρέπει να αξιοποιήσετε για να υλοποιήσετε αυτήν την προσέγγιση;
   - Χρονομετρήστε τον χρόνο εκτέλεσης κάθε νήματος χωριστά. Τι παρατηρείτε ως προς τους χρόνους αυτούς; Τι είναι αυτό που προκαλεί το συγκεκριμένο φαινόμενο;

2) **(10%) Δυναμική ανάθεση φόρτου εργασίας σε βρόχο for (Πρόγραμμα "Dynamic")**

   Ακολουθείστε την ίδια λογική με το προηγούμενο ερώτημα, όμως αυτή τη φορά η παραλληλοποίησης που θα πρέπει να ακολουθήσετε είναι κάθε νήμα να αναλάβει με **δυναμικό τρόπο ανάθεσης** να κάνει απλοποίηση ενός συνόλου πολυγραμμών. Στην συνέχεια, κάθε νήμα θα εκτελεί σειριακά τον αλγόριθμο Ramer–Douglas–Peucker για κάθε πολυγραμμή που ανέλαβε να επεξεργαστεί.

   - Ποιες οδηγίες του προτύπου OpenMP πρέπει να αξιοποιήσετε για να υλοποιήσετε αυτήν την προσέγγιση;
   - Ποιες παράμετροι των οδηγιών αυτών πρέπει να καθοριστούν πειραματικά ώστε να δώσουν τον καλύτερο δυνατό χρόνο εκτέλεσης της εφαρμογής;
   - Χρονομετρήστε τον χρόνο εκτέλεσης κάθε νήματος χωριστά. Τι παρατηρείτε ως προς τους χρόνους αυτούς και σε σύγκριση με το πρώτο ερώτημα;

3) **(40%) Παραλληλοποίηση με χρήση έργων (tasks) (Πρόγραμμα "Task1")**

   Όπως προαναφέρθηκε, όταν η μεγαλύτερη απόσταση ενός σημείου από την ευθεία που ορίζεται από το πρώτο και τελευταίο σημείο της πολυγραμμής είναι μεγαλύτερη από μια συγκεκριμένη τιμή, τότε δημιουργούνται δύο νέες, μικρότερες πολυγραμμές και επαναλαμβάνεται αναδρομικά ο αλγόριθμος για κάθε μια από τις νέες πολυγραμμές.

   Υλοποιήστε μια παράλληλη έκδοση του αλγορίθμου ώστε όταν μια πολυγραμμή διασπαστεί σε δύο μικρότερες, τότε για την πρώτη θα δημιουργείται ένα νέο έργο (task), ενώ την δεύτερη θα την επεξεργάζεται το τρέχων έργο (task).

- Ποιες οδηγίες του προτύπου OpenMP πρέπει να αξιοποιήσετε για να υλοποιήσετε αυτήν την προσέγγιση;

Θα πρέπει να δείξετε μεγάλη προσοχή στην δημιουργία της λίστας που θα αποθηκεύει την απλοποιημένη πολυγραμμή.

4) (40%) Παραλληλοποίηση με χρήση έργων (tasks) (Πρόγραμμα "Task2")

Υλοποιήστε μια παράλληλη έκδοση του αλγορίθμου ώστε όταν μια πολυγραμμή διασπαστεί σε δύο μικρότερες, τότε θα δημιουργείται ένα νέο έργο (task) και για τις δύο νέες πολυγραμμές.

- Ποιες οδηγίες του προτύπου OpenMP πρέπει να αξιοποιήσετε για να υλοποιήσετε αυτήν την προσέγγιση;
- Τι παρατηρείτε ως προς τον χρόνο εκτέλεσης της εφαρμογής, ειδικά σε σχέση με τις προηγούμενες προσεγγίσεις; Που οφείλεται αυτό το φαινόμενο; Πως θεωρείτε πως μπορεί να βελτιωθεί;

Θα πρέπει να δείξετε μεγάλη προσοχή στην δημιουργία της λίστας που θα αποθηκεύει την απλοποιημένη πολυγραμμή.

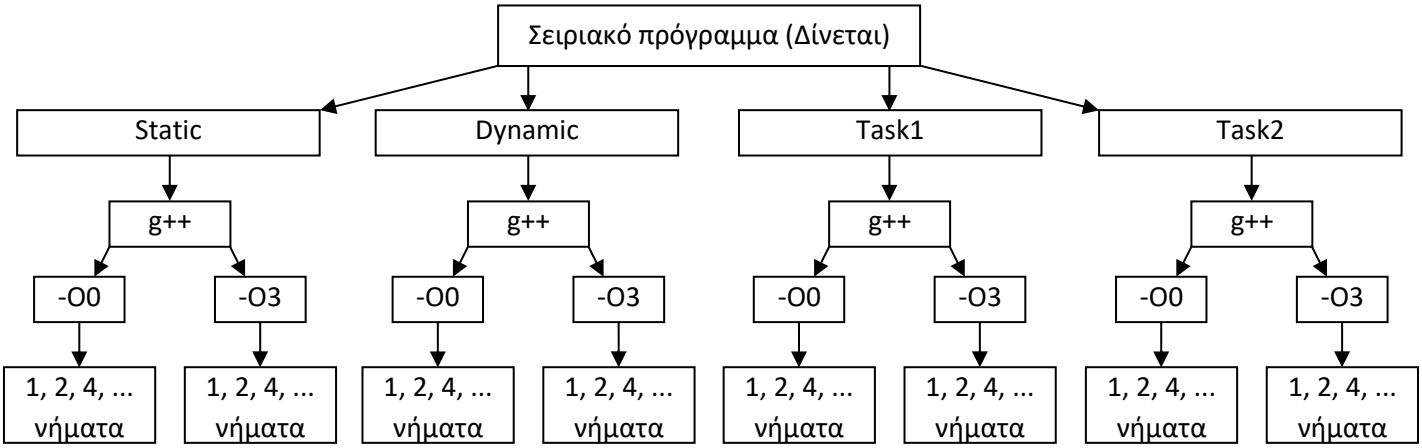
Εκτός όλων των παραπάνω είσαστε προφανώς ελεύθεροι να βελτιστοποιήσετε και να παραλληλοποιήσετε τον κώδικα με όποιον άλλο τρόπο θέλετε. Αν θεωρήσετε πως μια οποιαδήποτε αναδιοργάνωση του κώδικα μπορεί να οδηγήσει σε καλύτερη παραλληλοποίηση και απόδοση είστε ελεύθεροι να το κάνετε. Ωστόσο, θεωρείται αυτονόητο πως η υλοποίηση θα πρέπει να είναι σωστή (να δίνει **πάντα** τα ίδια αποτελέσματα με την ακολουθιακή έκδοση του προγράμματος) και πως **στην αναφορά που θα παραδώσετε θα πρέπει να αιτιολογήσετε γιατί παραλληλοποιήσατε με τον συγκεκριμένο τρόπο την εφαρμογή.**

Οδηγίες για την μεταγλώττιση και εκτέλεση του ακολουθιακού κώδικα που σας δίνεται και του παράλληλου κώδικα που θα φτιάξετε δίνονται στο Παράρτημα Β της εκφώνησης. Εκεί επίσης αναφέρεται για ποιες παραμέτρους θα πρέπει να τρέξετε την κάθε έκδοση του προγράμματος και ποια αποτελέσματα θα συμπεριλάβετε στην αναφορά σας.

Συγκεντρωτικά, θα πρέπει στην αναφορά σας να παραθέσετε μετρήσεις και να σχολιάσετε τις παρακάτω περιπτώσεις:

1) Πρόγραμμα "Static", "Dynamic", "Task1" και "Task2"
2) Μεταγλώττιση και εκτέλεση κάθε προγράμματος χωρίς βελτιστοποιήσεις (-O0) και με μέγιστες βελτιστοποιήσεις (-O3).
3) Εκτέλεση κάθε περίπτωσης με 1, 2 και 4 νήματα τουλάχιστον. Αν το σύστημα σας διαθέτει περισσότερους πυρήνες, ακόμα καλύτερα!

Για κάθε έναν από τους παραπάνω συνδυασμούς συμπεριλάβετε στην αναφορά σας πίνακες με τους χρόνους εκτέλεσης του βασικού αλγόριθμου (υπάρχει έτοιμο στον κώδικα που σας δίνεται) **και διαγράμματα της χρονοβελτίωσης**. Συγκεντρωτικά, όλες οι περιπτώσεις που θα πρέπει να συμπεριλάβετε φαίνονται στο παρακάτω σχήμα:


Σειριακό πρόγραμμα (Δίνεται)
Static
Dynamic
Task1
Task2
g++
g++
g++
g++
-O0
-O3
-O0
-O3
-O0
-O3
-O0
-O3
1, 2, 4, ... νήματα
1, 2, 4, ... νήματα
1, 2, 4, ... νήματα
1, 2, 4, ... νήματα
1, 2, 4, ... νήματα
1, 2, 4, ... νήματα
1, 2, 4, ... νήματα
1, 2, 4, ... νήματα

Παράρτημα Β

1) Για την εκτέλεση του σειριακού προγράμματος που σας δίνεται και των παράλληλων προγραμμάτων που θα φτιάξετε απαιτείται το πέρασμα παραμέτρων από την γραμμή εντολής. Κατά την διάρκεια ανάπτυξης και αποσφαλμάτωσης των προγραμμάτων σας μπορείτε να χρησιμοποιείται τις παρακάτω παραμέτρους, οι οποίες οδηγούν σε σχετικά μικρούς χρόνους εκτέλεσης:

```
polylines_small.txt 0.1 0
polylines_small.txt 0.01 0
polylines_small.txt 0.001 0
polylines_small.txt 0.0001 0
```

Με αυτές μπορείτε να ελέγχετε γρήγορα αν οι αλλαγές που κάνετε για την παραλληλοποίηση είναι σωστές. Για τις μετρήσεις που θα περιλαμβάνονται στην αναφορά σας θα χρησιμοποιήσετε τις παρακάτω παραμέτρους, που απαιτούν περισσότερο χρόνο:

```
polylines_large.txt 0.1 0
polylines_large.txt 0.01 0
polylines_large.txt 0.001 0
polylines_large.txt 0.0001 0
```

Λόγω μεγέθους, το αρχείο polylines_large.txt δεν περιλαμβάνεται στο συμπιεσμένο αρχείο με όλα τα υπόλοιπα απαραίτητα αρχεία για την εργασία. Θα πρέπει να το μεταφορτώσετε από τον παρακάτω σύνδεσμο:

https://drive.google.com/open?id=1Sc_mOqag7wjEiupZEJ30p6jW95Go_HlY

Μόλις μεταγλωττίσετε το σειριακό πρόγραμμα (βλέπε παρακάτω) μπορείτε να το τρέξετε χωρίς παραμέτρους για να δείτε μια σύντομη περιγραφή των παραμέτρων που χρησιμοποιούμε. **Σημειώνεται πως αν θέλετε να εμφανίζονται τα αποτελέσματα της εκτέλεσης του προγράμματος (αρχικές και απλοποιημένες πολυγραμμές) θα πρέπει στην τελευταία παράμετρο να αντικαταστήσετε το 0 με το 1.**

2) Για την μεταγλώττιση του σειριακού προγράμματος που σας δίνεται μπορείτε να χρησιμοποιήσετε την παρακάτω εντολή:

```
g++ -O3 (ή –O0) -Wall -Wextra -o RDP RDP.cpp
```

Αυτή θα δημιουργήσει ένα εκτελέσιμο αρχείο με το όνομα “RDP”, το οποίο μπορείτε να εκτελέσετε με την εντολή:

```
./RDP <Παράμετροι γραμμής εντολής>
```

Οι παράμετροι γραμμής εντολής είναι κάποιο από τα σύνολα που δόθηκαν παραπάνω. **Συνιστάται να τρέξετε την σειριακή εφαρμογή με όλα τα παραπάνω σύνολα παραμέτρων (ορίζοντας το κατάλληλο όνομα αρχείου εξόδου κάθε φορά), ώστε να έχετε τα αποτελέσματα σαν αναφορά για τις αλλαγές που θα κάνετε κατά την παραλληλοποίηση. Μην ξεχνάτε πως θα πρέπει να παίρνετε τα ίδια αποτελέσματα από το σειριακό και το παράλληλο πρόγραμμα!**

3) Για την μεταγλώττιση του παράλληλου προγράμματος με OpenMP που θα φτιάξετε μπορείτε να χρησιμοποιήσετε την παρακάτω εντολή (θεωρώντας πως το όνομα του αρχείου που φτιάξατε είναι “RDP_omp.cpp”):

```
g++ -O3 (ή –O0) –fopenmp -Wall -Wextra -o RDP_omp RDP_omp.cpp
```

Αυτή θα δημιουργήσει ένα εκτελέσιμο αρχείο με το όνομα “RDP_omp”. Μπορείτε να ορίσετε το πλήθος των νημάτων που θα δημιουργούνται σε κάθε παράλληλη περιοχή και στην συνέχεια να το εκτελέσετε με τις εντολές:

```
export OMP_NUM_THREADS=<Πλήθος νημάτων ανά παράλληλη περιοχή>
./RDP_omp <Παράμετροι γραμμής εντολής>
```