

Δεύτερο σύνολο Θεωρητικών και  
Προγραμματιστικών ασκήσεων  
Αποκεντρωμένου υπολογισμού

---

ΑΓΓΕΛΟΠΟΥΛΟΣ ΓΕΩΡΓΙΟΣ ΑΜ 1067345

ΤΣΙΡΩΝΗΣ ΑΝΔΡΕΑΣ ΑΜ 1063428

## Περιεχόμενα

Θεωρητικές Ασκήσεις.....	1
Άσκηση 1 .....	1
Άσκηση 2 .....	3
Άσκηση 3 .....	7
Προγραμματιστικές Ασκήσεις .....	9
Άσκηση 1 .....	9
Graphic Interface .....	11
Χρόνοι Σταθεροποίησης του Αλγορίθμου.....	15
Σχολιασμός Αποτελεσμάτων .....	22
Άσκηση 2 .....	23
Μέρος 1 .....	23
Μέρος 2 .....	24
Κώδικας προγραμματιστικών Ασκήσεων.....	27
Άσκηση 1 .....	27
Άσκηση 2 .....	33

## Θεωρητικές Ασκήσεις

### Άσκηση 1

Αναζητούμε να βρούμε εάν ο πληθυσμός μας συνολικά έχει ένα χαρακτηριστικό, συγκεκριμένα εάν έχει άρτιο πλήθος από αρχική κατάσταση 1 ή όχι. Δεν μας ενδιαφέρει συγκεκριμένα εάν πρέπει να το γνωρίζει ένας πράκτορας ή όλοι.

Πάμε να δούμε ποιους από τα πρωτόκολλα που είδαμε στο μάθημα ποια θα μπορούσαν να ταιριάζουν. Καθώς πρέπει να πάρουμε όλους τους πράκτορες υπόψη, θα απορρίψουμε κατευθείαν τον 3-state Approximate Majority Protocol, καθώς δεν μας νοιάζει να κυριαρχήσει κάποια κατάσταση ενάντια σε κάποια άλλη, αλλά να βρούμε εάν για το σύνολο των πρακτόρων, η αρχική τους κατάσταση υπακούει στην ιδιότητα που μας ενδιαφέρει, δηλαδή να βρούμε εάν υπάρχει άρτιος ή περιττός αριθμός από 1.

Όταν έρχονται δύο πράκτορες σε επαφή, μία πράξη  $xor$  είναι αρκετή για να μας βρει εάν είναι περιττός ή άρτιος αριθμός το σύνολο των καταστάσεων, 0 για άρτιο αριθμό κατάστασης 1 και 1 για περιττό. Έτσι, θα μπορούσαν να ενημερώνουν και οι δύο την κατάσταση τους, με βάση το αποτέλεσμα της  $xor$ . Το πρόβλημα όμως με αυτό την σκέψη

είναι ότι εάν έρθει ένας πράκτορας με αρχική κατάσταση 0 και άλλος με αρχική κατάσταση 1, και αλλάξουν και οι δύο την κατάσταση τους σε 1, τότε εάν έρθουν ξανά σε επαφή, θα γίνουν και οι δύο 0, οπότε το 1 θα έχει σβηστεί. Επίσης δίνει δυνατότητα σε κάποιο πράκτορα με αρχική κατάσταση 1 σε ένα πληθυσμό που έχει πολύ λίγους πράκτορες με αρχική κατάσταση 1, να μολύνει το υπόλοιπο δίκτυο, πριν αρχίζουν 1 να σβήνονται.

Οπότε για άρτιο συνολικό πρακτόρων, θα τείνει τελικώς ο πληθυσμός να συμφωνεί πάντα ότι η αρχική κατάσταση 1 των πρακτόρων ήταν άρτια σε αριθμό, καθώς όλα οι καταστάσεις θα έχουν γίνει 0. Αυτό θα ισχύσει μόνο εάν έχουμε uniform δρομολογητή, γιατί με strongly fair δρομολογητή δεν θα μπορεί να έρθουν ποτέ οι πράκτορες σε συμφωνία, καθώς θα μπορεί να ξανά αναπαράγει τα 1, διαλέγοντας να συνδέσει πράκτορα κατάσταση 1 με πράκτορες κατάσταση 0 για να ξανά μολύνει τον αλγόριθμο. Για περιττό αριθμό πρακτόρων, ακόμα και με uniform δρομολογητή, δεν θα μπορεί να φτάσει σε τελική συμφωνία, εάν υπάρχει έστω και ένα 1, λόγω αναπαραγωγής. Με αυτά βλέπουμε ότι τα πρωτόκολλα flock of birds και Epidemics, δεν θα μπορούσαν να λειτουργήσουν όπως είναι, γιατί διαδίδουν τις καταστάσεις τους και όλοι φτάνουν σε τελική απόφαση, κάτι που καταλήγουμε ότι δεν μπορεί να εφαρμοστεί στο δικό μας

Μία καλύτερη επιλογή, που διορθώνει το πρόβλημα αναπαραγωγής, είναι τα πρωτόκολλα της εκλογής αρχηγού, συγκεκριμένα την εκλογή αρχηγού υπόλοιπο mod  $m$ , το οποίο μετράει των αριθμών των πράκτορα που είναι σε μία ειδική κατάσταση  $A$ , και παίρνουμε το υπόλοιπο της διαίρεσης με μία σταθερά  $m$ , που στην περίπτωση μας θα είναι 2. Κάθε πράκτορας θα έχει δύο καταστάσεις, η μία θα δηλώνει εάν είναι αρχηγός ή ακόλουθος και η άλλη θα είναι εάν η συνολική κατάσταση του πληθυσμού είναι άρτια ή περιττή, με βάση τον εαυτό του και όσους έχει συναντήσει. Σε κάθε επαφή μεταξύ δύο αρχηγών. ο ένας θα γίνει ακόλουθος, δηλαδή θα απενεργοποιηθεί, με συνολική κατάσταση δικτύου 0 και άλλος θα παραμείνει αρχηγός, δηλαδή θα μείνει ενεργός, και θα κάνει υπολογισμό των καταστάσεων συνολική κατάσταση πληθυσμού και των δύο πρακτόρων mod 2, και θα αποθηκεύει αυτήν την τιμή στην κατάσταση του. Έτσι θα καταλήξουμε σε έναν αρχηγό που έχει γνώση εάν το σύνολο των 1 στον πληθυσμό είναι άρτιο (0) ή περιττό (1). Εμείς μπορούμε να το χρησιμοποιήσουμε, καθώς το αν ένας αριθμός  $k$ , είναι άρτιος αν  $k \bmod 2 = 0$  και περιττός εάν  $k \bmod 2 = 1$ , και εάν αθροιστούν τα υπόλοιπα δύο περιττών αριθμών, θα είναι  $2 \bmod 2 = 0$ . Μπορούμε τυπικά να θεωρήσουμε ότι το leader είναι ενεργοποιημένος πράκτορας και το follower απενεργοποιημένος.

Η εκλογή αρχηγού υπόλοιπο mod  $m$  είναι μία πολύ καλή επιλογή πρωτόκολλου, αλλά θα μπορούσαμε να απλοποιήσουμε το πρωτόκολλο, διώχνοντας την ανάγκη να έχουμε κατάσταση leader και follower. Αφού το να έρθουν δύο πράκτορες σε επαφή που έχουν αρχική κατάσταση 1 και οι δύο είναι το ίδιο με το να έχουν και οι δύο πράκτορες αρχική κατάσταση μηδέν, τότε μπορούμε να αποφύγουμε την αναπαραγωγή, με την μόνη αλλαγή κατάστασης να είναι ότι όταν βρεθούν δύο πράκτορες με κατάσταση 1, θα γίνουν και οι δύο μηδέν. Οπότε εάν υπάρχει άρτιος αριθμός από πράκτορες με κατάσταση 1, θα γίνουν όλοι μηδέν και εάν είναι περιττός, θα μείνει ένα μόνος πράκτορας με κατάσταση 1. Σε μία πρακτική υλοποίηση, θα μπορούσαμε να θεωρήσουμε ότι η κατάσταση 0 είναι η ανενεργή κατάσταση και 1 η ενεργή, και περιμένουμε να δούμε εάν θα μείνει ένας πράκτορας με ενεργή κατάσταση ή θα απενεργοποιηθούν όλοι, βλέποντας ότι είναι περιττό ή άρτιο ταυτόχρονα.

Οπότε το πρωτόκολλο θα είναι:

- States :  $Q = \{0,1\}$
- Initial states  $I(q_0) = 0$  και  $I(q_1)=1$
- Output:  $O(q_0)=0$  και  $O(q_1)=1$
- Transitions:  $(q_1,q_1) \rightarrow (q_0,q_0)$

Αφού χρησιμοποιήσουμε uniform δρομολογητή παραλληλοποιημένο, οπότε σε κάθε γύρο όλοι οι ενεργοποιημένοι κόμβοι (κατάσταση 1) θα συμμετέχουν σε μία αλληλεπίδραση με ένα άλλο ενεργοποιημένο πράκτορα (στους περιττούς ένας θα μείνει εκτός), οπότε οι μισοί ενεργοποιημένοι κόμβοι σε κάθε γύρο θα απενεργοποιούνται, μέχρι να μείνει μόνο ένας ή κανένας. Η χειρότερη περίπτωση είναι να είναι όλοι 1, καθώς εκεί θα χρειαστεί να γίνουν οι περισσότερες μεταβάσεις, οπότε η χρονική περιπλοκότητα είναι η  $O(\log(n))$ . Προφανής απαίτηση, που πάντα την θεωρούμε δεδομένη, είναι ότι ο δρομολογητής ακολουθεί την αρχή της προόδου, όπου ο δρομολογητής πρέπει να κάνει μία δράση που αλλάζει την κατάσταση του πληθυσμού (οπότε θα γίνονται μόνο οι αλληλεπιδράσεις που ορίσαμε στο transitions )

## Άσκηση 2

Λόγω του ότι μιλάμε για εθνικές εκλογές, το σύστημα δεν γίνεται να υλοποιηθεί με permissionless blockchain διότι χρειάζεται ένας τρόπος για να επιβεβαιωθεί ότι ο χρήστης είναι πολίτης και μπορεί όντως να ψηφίσει! Σε περίπτωση που έχουμε permissionless blockchain αυτό δεν γίνεται οπότε ο καθένας που μπορεί να συνδεθεί στο δίκτυο θα μπορεί και να ψηφίζει, κάτι το οποίο νοθεύει την διαδικασία και δεν το θέλουμε.

Επίσης θέλουμε να διατηρήσουμε την ανωνυμία της διαδικασίας ώστε ο κάθε χρήστης που ψηφίζει να μην αντιστοιχίζεται με τα αληθινά του στοιχεία. Παρόλα αυτά, πρέπει να υπάρχει ένας έλεγχος στο παρασκήνιο για τον χρήστη και την επαλήθευση των στοιχείων του το οποίο μπορεί να γίνει ως εξής:

Σε μία ξεχωριστή βάση δεδομένων θα είναι ασφαλώς αποθηκευμένα όλα τα αναγνωριστικά όλων των πολιτών (πχ αριθμοί ταυτότητας). Κάθε χρήστης που θα θέλει να δημιουργήσει λογαριασμό στο σύστημα πρέπει πρώτα να επαληθεύσει τα στοιχεία του. Αφού γίνει η επαλήθευση αυτή, τότε αποκτάει ένα μοναδικό public key με το οποίο θα αναγνωρίζεται στην εφαρμογή ενώ ταυτόχρονα το ίδιο αναγνωριστικό δεν θα μπορεί να ξαναφτιάξει λογαριασμό για την τρέχουσα υλοποίηση. Αυτό το public key θα παράγεται από τα στοιχεία του χρήστη (όπως πχ τον αριθμό ταυτότητας) μαζί με μία τυχαία μοναδική τιμή (όπως πχ την ημερομηνία και την ώρα εγγραφής)

μέσω SHA-256. Έτσι όλοι οι χρήστες του συστήματος εγγυημένα δικαιούνται να ψηφίσουν αλλά και μέσω του ελέγχου απορρίπτονται ενδεχόμενα για Sybil Attacks τα οποία θα νοθεύαν σημαντικά την όλη διαδικασία.

Αυτό σαν υλοποίηση μας εξασφαλίζει και το Zero-Knowledge Proof που θα χρειαστούμε στην συνέχεια. Σαν μηχανισμό συναίνεσης και επιλογής των Block Producers απορρίπτουμε τα PoW και PoS για λόγους δαπάνης ενέργειας και απόδοσης για το πρώτο και για το PoS λόγω της σωστής υλοποίησης. Φανταζόμαστε την κάθε ψήφο στην μορφή ενός νομίσματος, άρα σε περίπτωση ενός PoS μοντέλου, οι εκλεγόμενοι θα ήταν και η εφορευτική επιτροπή, άρα δεν μας κάνει. Ύστερα από λίγη μελέτη καταλήξαμε στην υλοποίηση ενός pBFT (practical Byzantine Fault Tolerance) μηχανισμού το οποίο μας επιτρέπει:

- Μεγάλη απόδοση σε throughput από Transactions και block production
- Χαμηλή κατανάλωση ενέργειας
- Αρκετά μεγάλη αξιοπιστία αποτελεσμάτων αφού ο μηχανισμός είναι σχεδιασμένος να αντέχει σε κακόβουλες συμπεριφορές μέσω της πλειοψηφίας 2/3.

Η εκλογή των Validators θα γίνεται από πριν από κρατικά όργανα ή και πολίτες με τέτοιο τρόπο ώστε να τηρείται μία ισορροπία μεταξύ τους και μία αρκετά μεγάλη ομάδα από Validators να μην είναι biased ως προς ένα ή περισσότερους εκλεγόμενους.

Η ψήφος μπορεί να υλοποιηθεί ως ένα Transaction από τον κάθε χρήστη προς το κόμμα ή το άτομο που ψηφίζει χρησιμοποιώντας το public key του επιλεγθέντα. Κάθε user θα μπορεί να κάνει μόνο ένα τέτοιο Transaction. Μια επιλογή είναι σε κάθε λογαριασμό να δοθεί από ένα ελάχιστον νόμισμα το οποίο θα λειτουργεί σαν ψήφος και με το οποίο θα μπορεί να κάνει συναλλαγή με οποιόν άλλον χρήστη θέλει, δηλαδή να το δώσει σε χρήστη που εκπροσωπεί ένα κόμμα.

Έτσι όλο το ιστορικό των ψήφων μπορεί να καταγραφεί ψευδοανώνυμα χωρίς να προδίδεται η ταυτότητα κανενός και να ελέγχεται από όλους η εγκυρότητα της διαδικασίας. Στο τέλος των εκλογών απλά μετράμε τα Beans του καθενός και έτσι βγαίνουν τα αποτελέσματα.

#### **Από την μεριά του προγραμματιστή:**

- Το σύστημα θα είναι ένα Permissioned Blockchain.
- Θα υπάρχει μια εξωτερική δομή αποθήκευσης και ελέγχου των στοιχείων του ατόμου που θέλει να γραφτεί για την εγκυρότητα της διαδικασίας.

- Κάθε user θα δεσμεύεται με ένα public key και θα έχει από την αρχή ένα νόμισμα (**Bean**) που μπορεί να ξοδέψει.

- Θα υπάρχουν προκαθορισμένοι κομβοί που θα ελέγχουν τις ψήφους και την εγκυρότητα τους (**Validators**). Αυτοί θα είναι ελεγχόμενοι από έμπιστα άτομα, θα λειτουργήσουν ως εφορευτική επιτροπή. Για την καταγραφή ενός μπλοκ στην αλυσίδα θα χρειάζεται μια πλειοψηφία 2/3 των Validators να συμφωνούν σε ένα block.

- Μια ψήφος είναι ένα Transaction μεταφοράς ενός Bean σε ένα λογαριασμό. Αυτή η διαδικασία χωρίζεται σε 3 στάδια:

- Αρχικά το στάδιο της προ-προετοιμασίας: Ο validator προτείνει ένα Tx και το μεταδίδει σε όλους τους validators. Όταν λάβει ένας validator την πρόταση, τσεκάρει αν ισχύει για να την βάλει στο block.

- Το στάδιο της προετοιμασίας: Οι validators ετοιμάζουν ένα "prepare" μήνυμα και το στέλνουν σε όλους τους άλλους Validators. Όταν λάβει ένας prepare μήνυμα για ένα Tx από τα 2/3 των validators, προχωράει στο επόμενο βήμα.

- Το στάδιο της εισαγωγής: Κάθε Validator στέλνει ένα "commit" μήνυμα στους υπολοίπους. Μόλις λάβει "commit" μήνυμα από τα 2/3 των Validators, προσθέτει το Tx στο block και μεταδίδει το block σε όλο το δίκτυο. Στο τέλος θα έχει μεταδοθεί το τελικό block με όλες τις έγκυρες συναλλαγές με βάση την πλειοψηφία των Validators.

- Το μέγεθος του Block θα μπορούσε να είναι 1024 transactions μιας και πρόκειται για εκλογές και με την δομή που έχουμε θα έχουμε μεγάλο throughput από Transactions. Αν βάζαμε πολύ μικρό τότε μάλλον πολλές αλυσίδες θα δημιουργόντουσαν και μερικά transactions θα χάνονταν, με αποτέλεσμα να χρειαστεί αυτοί οι χρήστες να ξαναψηφίσουν, κάτι που δεν θέλουμε.

- Σαν μηχανισμό επιλογής επομένου block θα χρησιμοποιήσουμε το longest chain διότι αυτή η επιλογή μας επιτρέπει να απορρίψουμε όσο το δυνατόν λιγότερα έγκυρα Transactions σε σχέση με τις υπόλοιπες επιλογές αφού περιέχει τα περισσότερα έγκυρα blocks.

- Σε αυτό το μοντέλο υποθέτουμε ότι οι Block Producers είναι δίκαιοι και παίζουν με τους κανόνες μιας και είναι η εφορευτική επιτροπή. Ακόμα κι αν μερικοί κάνουν κακόβουλες πράξεις, η πλειοψηφία θα τους επισκιάσει. Στην περίπτωση του PBFT πρέπει να επικρατεί η πλειοψηφία 2/3 για να περάσει ένα Tx οπότε θεωρούμε ότι είναι αρκετά ισχυρό.

#### **Από την μεριά του Χρήστη:**

- Αρχικά ο χρήστης που θέλει να ψηφίσει πρέπει να γραφτεί στο σύστημα

- Εκεί θα πρέπει να τακτοποιηθεί ως δικαιούχος ψήφου στην εξωτερική δομή αποθήκευσης μέσω των αναγνωριστικών του και θα αποκτά ένα μοναδικό public key το οποίο δεν θα προδίδει την ταυτότητα του.
- Με αυτό το public key θα είναι εμφανίσιμος στο σύστημα και όλοι θα μπορούν να δουν και να αλληλοεπιδράσουν με αυτόν.
- Με το που δημιουργηθεί ο λογαριασμός, θα του δίνεται ένα νόμισμα (έστω **BEan**)
- Όταν θέλει να κάνει μια ψήφο, θα ξέρει από πριν το public key του κόμματος που θα θέλει να ψηφίσει και θα κάνει ένα transaction μεταφοράς του Bean του σε αυτόν τον χρήστη.
- Θα μπορεί να δει μετρά από ένα χρονικό διάστημα αν αυτή η συναλλαγή του καταγράφηκε ή αν πρέπει να την ξανακάνει.
- Όταν λάβει ένα block που καταγραφεί την συναλλαγή του και δεν ακυρώνεται για ένα χρονικό διάστημα πχ μιας μέρας τότε η ψήφος του έγινε.
- Αν θέλει μπορεί να ανατρέξει ζωντανά κατά τη διάρκεια της ψηφοφορίας και να δει τα τωρινά αποτελέσματα διαβάζοντας τα καταγεγραμμένα Transactions και βλέποντας ποιο κόμμα είναι μπροστά κτλ.
- Έτσι επιτυγχάνεται εξαιρετικά μεγάλος βαθμός ελευθέριας και δημοκρατικότητας αφού ο καθένας έχει την δύναμη της ψήφου του, την ευχέρεια να δει την πορεία των εκλογών και να κρίνει ανάλογα για την ψήφο του και τα θέλω του και υπάρχει διασφάλιση της εγκυρότητας των εκλογών από όλους τους χρήστες.

#### **Αναφορικά:**

Με αυτήν την υλοποίηση υπάρχει ευκολία στην χρήση και στην υλοποίηση έχοντας κάνει κάποιες βασικές υποθέσεις και παραδοχές όπως ότι:

- Ο χρήστης μπορεί να ψηφίσει μόνο έναν χρήστη στο σύστημα και εμπιστευόμαστε ότι θα διαλέξει το κατάλληλο public key.
- Δεν γίνεται να παρθεί πίσω μία ψήφος και να δοθεί κάπου αλλού.
- Δεν έχει υπολογιστεί η δυνατότητα για 2ο γύρο εκλογών ή νέων εκλογών. Σε τέτοια περίπτωση υποθέτουμε ότι μία λύση είναι η διαγραφή όλων των χρηστών και η επαναδημιουργία των λογαριασμών τους όπως την πρώτη φορά με τις ίδιες προϋποθέσεις. Λίγο απλοϊκό στην σκέψη αλλά θεωρητικά θα δούλευε αφού μιλάμε για ένα permissioned blockchain.
- Όσο αναφορά το blockchain, επειδή ο σκοπός της διαδικασίας δεν είναι η καταγραφή και απόκτηση currency αλλά της ψηφοφορίας, δεν χρειάζεται να παράγεται χρήμα στις συναλλαγές. Επίσης η επιλογή της συναίνεσης γίνεται με την υπόθεση ότι όλοι ή έστω οι περισσότεροι θα παίζουν τίμια. Αν τους block producers τους διαλέγουν ισάριθμα τα

κόμματα μεταξύ τους (αφού όλα τους θέλουν το ατομικό καλό και όχι την προβιάσει των άλλων) όλοι τους θα προσπαθούν να παίξουν τίμια γιατί στην τελική δεν έχουν κανέναν άλλον να στηρίζουν και δεν θα είναι πολλοί στο ίδιο συμφέρον για να αλλοιώσουν την διαδικασία.

## Άσκηση 3

Παρόλο που ο αλγόριθμος μιλάει για το πώς θα επιλεγθεί ένας συγκεκριμένος χρήστης για να προτείνει το επόμενο μπλοκ, εμάς μας νοιάζει να ξέρουμε με σχεδόν απόλυτη σιγουριά εάν θα επιλεγθεί ένα νόμισμα που το έχει στοιχηματίσει κάποιος, ώστε να μπορέσει να προτείνει μπλοκ. Υπάρχουν πολλές υποδιαίρεσεις των νομισμάτων στα κρυπτονομίσματα, οπότε διαλέγοντας ένα νόμισμα ελάχιστης αξίας, μπορούμε να υπολογίσουμε την πιθανότητα επιλογής σε κάθε εκτέλεση του αλγορίθμου, βάση του ποσού των χρημάτων που έχουν στοιχηματισθεί σε σχέση με το σύνολο του δικτύου. Επίσης, το πλήθος των νομισμάτων του blockchain δεν θα αλλάξει, καθώς μέχρι να βρεθεί χρήστης που θα προτείνει, δεν θα γίνει coinbase transaction για να κόψει νέα νομίσματα.

Πολλά πράγματα στα blockchain και στα κρυπτονομίσματα, πχ το collusion resistance στο hashing, λειτουργούν στοχαστικά, αλλά η πιθανότητα να μην γίνει το γεγονός που θέλουμε είναι τόσο μικρό, ώστε να μην υπάρχει πρακτικά φόβος αποτυχίας (χρειάζονται να  $2^{130} + 1$  δοκιμές για να βρούμε δύο εισόδους που δίνουν ίδια έξοδο). Επίσης, το ίδιο βλέπουμε να ισχύει για την αναζήτηση του nonce για bitcoin mining, όπου έχει ένα μέσο όρο των δέκα λεπτών να βρεθεί ένα block και ακολουθεί Poisson distribution, το οποίο από ένα σημείο και μετά είναι πρακτικά βέβαιο ότι θα βρεθεί μπλοκ.

Εμείς αυτό που ψάχνουμε είναι ότι θα έχουμε μία επιτυχή επιλογή μετά από ένα πεπερασμένο αριθμό εκτελέσεων του αλγορίθμου, με μία πιθανότητα που προσεγγίζει ικανοποιητικά κοντά στο 1, δηλαδή στην βεβαιότητα ότι θα γίνει. Θα διαλέξουμε την γεωμετρική κατανομή, όπου βρίσκουμε την πρώτη επιτυχία μετά από ένα αριθμό αποτυχιών για εύρεση υποψήφιου πρότασης μπλοκ, όπου κάθε επανάληψη είναι ανεξάρτητη από την άλλη, καθώς είναι μία ειδική περίπτωση των binomial distribution. Δεν έχουμε ένα συγκεκριμένο αριθμό εκτελέσεων που θα κάνουμε, αλλά ότι με έναν αριθμό εκτελέσεων και πάνω θα έχουμε στοχαστικά σίγουρη επιλογή υποψήφιου.

Θα ορίσουμε την πιθανότητα επιλογής νόμιμου πράκτορα για την πρόταση νέου μπλοκ σε μία εκτέλεση του αλγορίθμου, δηλαδή μέσω ομοιόμορφης τυχαίας επιλογής να διαλέξει ένα νόμισμα που έχει στοιχηματισθεί για το επόμενο μπλοκ:

$$p = \frac{\text{αριθμός χρημάτων που έχουν στοιχηματισθεί για το επόμενο μπλοκ}}{\text{αριθμός χρημάτων που υπάρχουν στο blockchain}}$$



Μπορούμε να αντιληφθούμε ότι όσο περισσότερος αριθμός νομισμάτων στοιχηματίζουν οι χρήστες του blockchain, τόσο πιο γρήγορα θα βρεθεί ένας υποψήφιος χρήστης για να προτείνει το επόμενο μπλοκ. Για αυτό θέλουμε η λειτουργία του blockchain να το ενθαρρύνει, που ένας πολύ καλός τρόπος είναι η επιβράβευση σε νομίσματα που δίνεται σε αυτόν που πρότεινε το κόμβο. Το αν θα ενθαρρύνει να συμμετέχουν πολύ χρήστες ή λίγοι χρήστες με πολλά νομίσματα είναι επιλογή υλοποίησης του blockchain. Θα θέσουμε σαν ικανοποιητική πιθανότητα το να είναι η συσσωρευτική πιθανότητα επιτυχίας ίση και μεγαλύτερη του 0.99

Δεν θα χρησιμοποιήσουμε κάποια συγκεκριμένα μεγέθη, θα κρατήσουμε την απόδειξη και ένα ενδεικτικό αριθμό εκτελέσεων γενικά. Αλλά κρίνοντας από το μεγαλύτερο proof of stake κρυπτονόμισμα, το Ethereum, βλέπουμε ότι 27% περίπου των νομισμάτων τους γίνεται staked <https://www.coinbase.com/en-fr/earn/staking/ethereum>.

Για να ψάξουμε να βρούμε τον αριθμό επαναλήψεων που χρειάζεται για να πετύχουμε μία επιθυμητή πιθανότητα, το οποίο βρίσκεται από τον αντίστροφο της συνάρτησης άθροισης πιθανότητας της κατανομής (Cumulative Distribution Function) ή αλλιώς quantile function. Υπολογίζοντας το inverse cdf της γεωμετρικής κατανομής μας με το πάνω ποσοστό στην matlab, δίνει ότι θέλει 14 εκτελέσεις ο αλγόριθμος για να φτάσει στην συσσωρευτική πιθανότητα επιτυχίας 0.99.

Το CDF της γεωμετρικής κατανομής, με  $k$  συγκεκριμένο αριθμό γύρων, θα είναι:

$$F_X = P(X \leq k) = 1 - (1 - p)^k$$

Το οποίο για ένα  $k$  γύρο βρίσκει την πιθανότητα να έχει υπάρξει επιτυχία για  $k$  και λιγότερους γύρους.

Με  $P(X \leq k) = 0.99$  και επίλυση ως  $k$ , το inverse cdf θα διαμορφωθεί ως εξής;

$$F_X^{-1}(p) = k = \frac{\log(1 - 0.99)}{\log(1 - p)}$$

Μπορούμε να αποδείξουμε και ότι ο αλγόριθμος θα έχει βέβαιη επιλογή χρήστη για να προτείνει μπλοκ, αποδεικνύοντας ότι για εκτελέσεις  $k$  που τείνουν στο άπειρο, η συσσωρευμένη πιθανότητα των γύρων θα είναι 1, εάν έχει στοιχηματισθεί έστω και ένα νόμισμα:

$$\lim_{k \rightarrow \infty} F_X = \lim_{k \rightarrow \infty} 1 - (1 - p)^k$$

Έχουμε  $0 < 1 - p < 1$ , το οποίο το οποίο υψώνουμε στην δύναμη  $k$ . Όσο το  $k$  τείνει στο άπειρο, τόσο το αποτέλεσμα θα τείνει στο μηδέν. Οπότε:

$$\lim_{k \rightarrow \infty} 1 - (1 - p)^k = 1 - 0 = 1$$

Άρα θα μπορεί να επιλεγθεί πάντα κάποιος χρήστης που έχει στοιχηματίσει το επόμενο μπλοκ.

## Προγραμματιστικές Ασκήσεις

### Άσκηση 1

Ο αλγόριθμος που υλοποιούμε είναι ένας σιωπηλός αυτοστεθοποιητικός αλγόριθμος όπου μέσω Breadth First Search (BFS) βρίσκει το spanning tree, δηλαδή ένα υποδιάγραμμα όπου είναι ένα δέντρο που εμπεριέχει όλες τις κορυφές, σε συνεκτικό μη κατευθυνόμενο διάγραμμα. Σαν ρίζα επιλέγεται ένα τυχαίος κόμβος. Ο αλγόριθμος εκτελείται σε λούπα για κάθε κόμβο ξεχωριστά, ώστε να μπορεί να επανέλθει από διαταραχές. Χρειάζεται να γνωρίζεται από πριν η διάμετρος του διαγράμματος. Δεν υπάρχει η ανάγκη να χρησιμοποιηθεί κάποιο είδους αναγνωριστικό για το κάθε κόμβο, μόνο να ξεχωρίζουμε ποια είναι η ρίζα.

Η περιγραφή του αλγόριθμου κατηγοριοποιείται στις ρίζες και στους υπόλοιπους κόμβους. Παρόλα αυτά όλοι οι κόμβοι εμπεριέχουν μία μεταβλητή για την απόσταση που έχουν από την ρίζα, όπου μπορούν να πάρουν τιμές από το 0 (η ίδια η ρίζα) μέχρι κάποια τιμή D, όπου η τιμή D είναι ίση ή μεγαλύτερη από την διάμετρο του γραφήματος. Η τιμή D είναι η μέγιστη τιμή που μπορεί να πάρει ένας κόμβος σαν απόσταση από την ρίζα, χωρίς κάποια διαταραχή. Όλοι οι κόμβοι ξέρουν τους γείτονες τους. Όλοι οι κόμβοι, εκτός της ρίζας, έχουν δείκτη σε κάποιο γείτονα τους όπου έχει απόσταση κατά ένα λιγότερο από τον ίδιο τον κόμβο. Το μονοπάτι που δημιουργεί ο δείκτης των δεικτών καταλήγει πάντα στην ρίζα.

Η εκτέλεση του αλγόριθμου στον κόμβο της ρίζας:

- Εάν η απόσταση από την ρίζα είναι διάφορο του μηδενός, τότε το αλλάζουμε την απόσταση σε μηδέν

Η εκτέλεση του αλγόριθμου εκτός του κόμβου της ρίζας:

- Αναζητείται μεταξύ των γειτόνων του κόμβου η ελάχιστη απόσταση από την ρίζα που μπορεί να βρεθεί.
- Ελέγχεται εάν η απόσταση του κόμβου από τη ρίζα μείον 1 είναι ίσο με την απόσταση του γείτονα με την ελάχιστη απόσταση από τη ρίζα, που βρήκαμε πριν

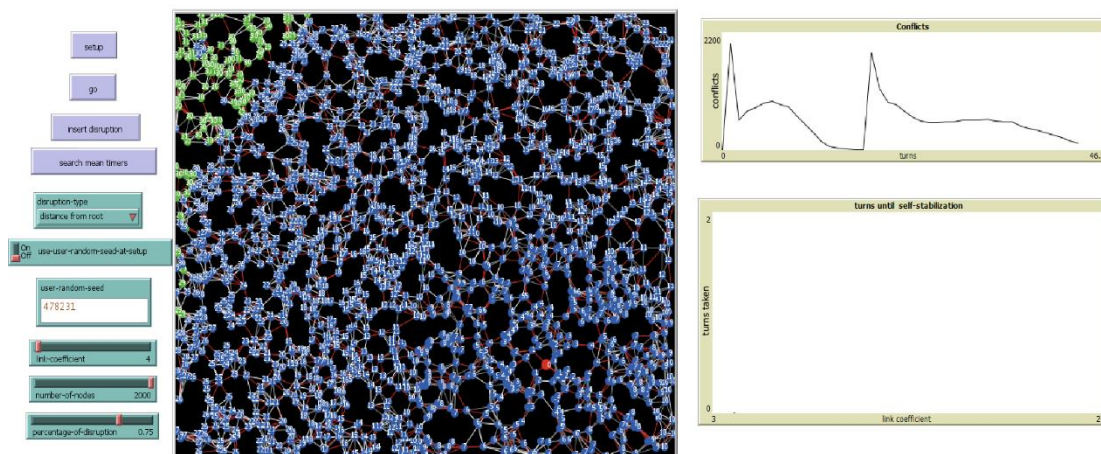
- Εάν δεν είναι ίσο, ορίζουμε την απόσταση του κόμβου από τη ρίζα ίσο είτε με την ελάχιστη απόσταση γείτονα από τη ρίζα είτε η τιμή D.
- Εάν είναι ίσο, τσεκάρουμε να δούμε εάν ο δείκτης δείχνει σε κάποιον κόμβο (δεν είναι κενό) και εάν δείχνει σε κόμβο που έχει την απόσταση που έχει ο κόμβος από που προέρχεται ο δείκτης μείον 1
- Εάν κάτι από τα δύο δεν ισχύει, τότε στον δείκτη δείχνουμε ένα γείτονα που έχει σαν απόσταση από την ρίζα την ελάχιστη αποστάση από την ρίζα των γειτόνων του κόμβου από πού προέρχεται ο δείκτης(σε πολλαπλές επιλογές διάλεξε τυχαία)

Έτσι λειτουργεί ο αλγόριθμος θεωρητικά και έχει υλοποιηθεί ανάλογα στην netlogo. Ο κενός δείκτης έχει προστεθεί γιατί ενώ δεν υπάρχει θεωρητικά, η υλοποίηση του συστήματος ξεκινάει ορίζοντας τους δείκτες κενούς, και είναι μία πιθανή πηγή διαταραχής. Σε κάθε αλλαγή, θέτουμε την μεταβλητή *stabilized* ίσο με *false*. Αν δεν υπάρχει κάποια αλλαγή σε κάποια μεταβλητή, τότε ορίζουμε την μεταβλητή *stabilized* ίσο με *true*. Αυτό το κάνουμε για να μπορούν να προστεθούν παραπάνω είδη διαταραχών, χωρίς να χρειαζόμαστε να προσθέτουμε ειδικές μεταβλητές που θα δείχνουν εάν υπήρξε αλλαγή σε κάποια μεταβλητή ή όχι (αν και δύο κύριες μπορούν να βρεθούν). Εάν το σύστημα έχει φτάσει στο σύνολο της έχει σταθεροποιηθεί στους κόμβους τους, σημαίνει ότι έχει φτάσει στην σωστή λειτουργία και δεν θα ξεφύγει από αυτό, εκτός αν υπάρξει διαταραχή. Υπάρχει ένα *redundancy* ενός γύρου για να οριστικοποιηθούν ότι όλοι οι κόμβοι είναι σταθεροποιημένοι.

Υλοποιήσαμε δύο είδη διαταραχών, που μπορούμε να θεωρήσουμε και έως κοινές. Η πρώτη είναι η διαταραχή δείκτη, όπου ο δείκτης είτε γίνεται κενός (εάν δεν είναι είδη), είτε αλλάζει τον κόμβο στο οποίο δείχνει. Η δεύτερη διαταραχή είναι η διαταραχή απόστασης από κόμβο, όπου η απόσταση από κόμβο παίρνει μία τυχαία τιμή μεταξύ 0 και δύο φορές την τιμή D, ώστε να προκύψουν τιμές πάνω από D.

Εμείς θα αναλύσουμε τον αριθμό των γύρων που κάνει ο αλγοριθμός για να φτάσει σε *self-realization* (*self-stabilization*) σε τρεις περιπτώσεις: Όταν πρωτοδημιουργείται το διάγραμμα, στην διαταραχή δείκτη μετά από σταθεροποίηση και διαταραχή απόστασης από ρίζα μετά από σταθεροποίηση. Θα τρέξουμε για βαθμό κόμβου από 4 έως 20, δέκα φορές για κάθε βαθμό κόμβου. Θα τρέξουμε όλους τους βαθμούς κόμβων με βάση το ποσοστό των κόμβων που επηρεάζονται στο 75%. Το *random-seed* που θα χρησιμοποιήσουμε θα είναι το 478231

## Graphic Interface



Τα γραφικά στοιχεία του διαγράμματος είναι τα εξής:

- Μπλε κόμβοι: σταθεροποιημένοι κόμβοι
- Πράσινοι κόμβοι: μη σταθεροποιημένοι κόμβοι (απόσταση ή δείκτης άλλαξε)
- Κόκκινος κόμβος: Ρίζα του δέντρου
- Κόκκινες ακμές: Υπόδειξη ότι η ακμή είναι δείκτης από ένα κόμβο προς ένα άλλο, κρίνοντας από την απόσταση από τη ρίζα που προέρχεται και που δείχνει
- Λευκές ακμές: Απλές ακμές
- Νούμερο του κόμβου: Υποδηλώνει την απόσταση του κόμβου από τη ρίζα

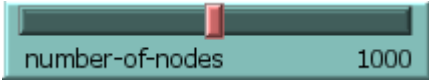
Τα κουμπιά του interfaces είναι τα εξής:

- setup: Εκκαθαρίζει ότι υπάρχει πριν και δημιουργεί ένα γράφημα με βάση τις μεταβλητές number-of-nodes (αριθμό κόμβων) και link-coefficient (βαθμός κόμβων)
- go: Τρέχει τον αλγόριθμο στους κόμβους που δημιουργήθηκαν από το setup. Σταματάει όταν έχουν σταθεροποιηθεί όλοι οι κόμβοι.
- Insert disruption: Εισάγει τον τύπο της διαταραχής που έχει επιλεγεί στο disruption type και σε ένα ποσοστό κόμβων που επηρεάζονται από το percentage of disruption. Χρησιμοποιείται σαν συνάρτηση για το search mean times
- Search mean timers: Ψάχνει να βρει τον μέσο χρόνο που θα κάνει ο αλγόριθμος να φτάσει σε αυτό-σταθεροποίηση, με βάση ένα συγκεκριμένο αριθμό κόμβων, βαθμό ακμών από 4 έως 20 και ένα τύπο διαταραχής, που ορίζεται στο disruption type. Συγκεκριμένα, δημιουργεί το γράφημα από την αρχή, το τρέχει μέχρι να

σταθεροποιηθεί και έπειτα εφαρμόζει την insert-distruption, και μετράει τους γύρους που πήρε για να σταθεροποιηθεί. Εάν επιλεχθεί στο distruption-type η επιλογή none, απλώς μετράμε τους γύρους που παίρνει το πρώτο κομμάτι. Για κάθε βαθμό ακμών, εκτελούμε τον αλγόριθμο με καινούργιο γράφημα δέκα φορές και παίρνουμε τον μέσο όρο των γύρων που μας πήρε και το βάζουμε στο διάγραμμα turns until self-stabilaization. Κρατάμε το seed σταθερό για τις κάθε δέκα επαναλήψεις

Οι επιλογές τιμών είναι οι εξής

- number-of-nodes: Αριθμός κόμβων



A slider control for the variable 'number-of-nodes'. The slider bar is green with a red knob. Below the slider, the text 'number-of-nodes' is followed by the value '1000'.

Slider

Global variable: number-of-nodes

Minimum: 10 Increment: 1.0 Maximum: 2000


min, increment, and max may be numbers or reporters

Value: 1000 Units (optional):

☐ vertical?

OK Apply Cancel

- link-coefficient: Βαθμός των κόμβων



A slider control for the variable 'link-coefficient'. The slider bar is green with a red knob. Below the slider, the text 'link-coefficient' is followed by the value '15'.

Slider

Global variable: link-coefficient

Minimum: 3 Increment: 1 Maximum: 25

min, increment, and max may be numbers or reporters

Value: 15 Units (optional):

☐ vertical?

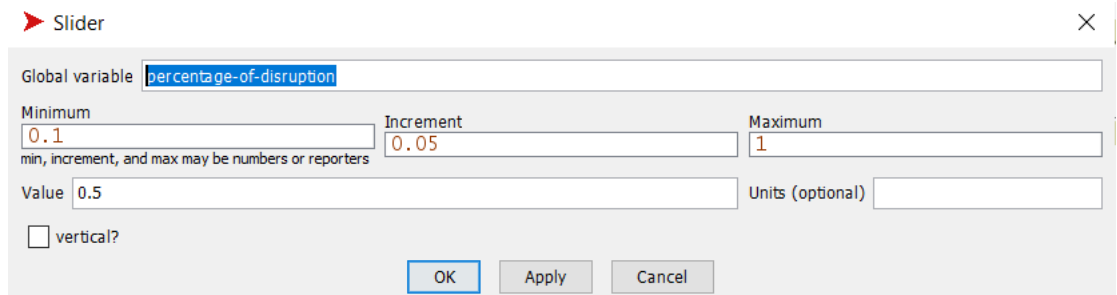
OK Apply Cancel

- percentage of disruption: Ποσοστό των κόμβων σε σχέση με το σύνολο που θα διαταραχθεί όταν χρησιμοποιήσουμε το insert-distruption command



A slider control for the variable 'percentage-of-disruption'. The slider bar is green with a red knob. Below the slider, the text 'percentage-of-disruption' is followed by the value '0.50'.

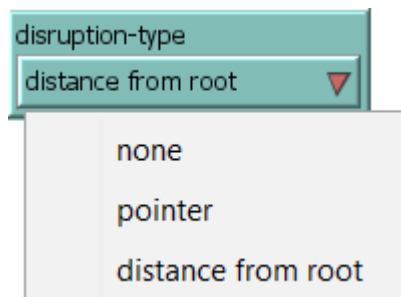
percentage-of-disruption 0.50



Slider dialog box with the following fields:

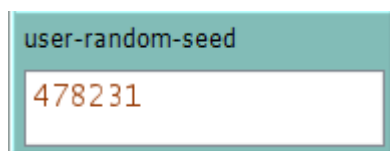
- Global variable: `percentage-of-disruption`
- Minimum: `0.1`
- Increment: `0.05`
- Maximum: `1`
- Value: `0.5`
- Units (optional):
- ☐ vertical?
- Buttons: OK, Apply, Cancel

- `disruption-type` Επιλογή μεταξύ του να επηρεάσουμε τους δείκτες, τις αποστάσεις ή τίποτα από αυτά



disruption-type dropdown menu with options: none, pointer, distance from root

- none: Δεν προστίθεται κάποια διαταραχή. Σημαντικό για να μετρήσουμε τους γύρους που παίρνει η αρχική σύγκλιση, από δημιουργία του συστήματος έως σε self-stabilization
  - pointer: Διαταραχή δείκτη
  - distance from root: Διαταραχή απόστασης από ρίζα.
- `user-random-seed`: Seed το οποίο θα χρησιμοποιηθεί για το search mean timers και αν το `use-user-random-seed-at-setup` για το να χρησιμοποιηθεί στο setup.



user-random-seed input field with value: 478231

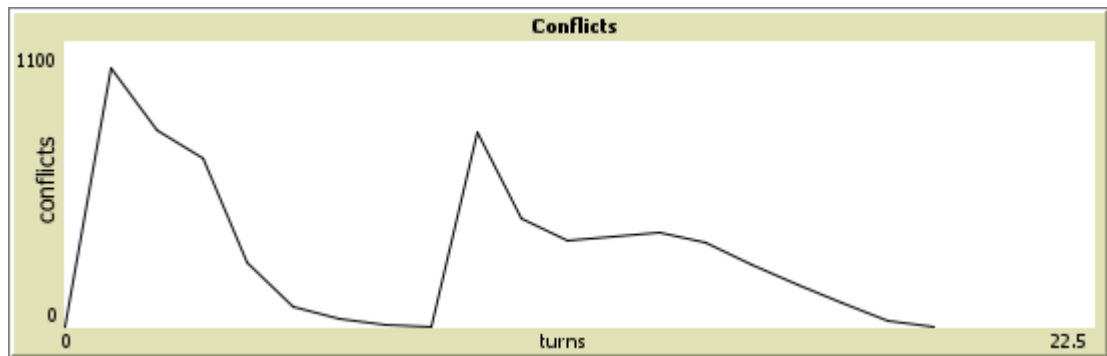
- `use-user-random-seed-at-setup`: Αν επιλεγθεί να είναι on θα χρησιμοποιήσουμε το `user-random-seed`, αλλιώς ένα τυχαίο seed.



use-user-random-seed-at-setup toggle switch with On/Off indicator

Γραφήματα που χρησιμοποιούνται:

- Conflicts: Δείχνει τους μη σταθεροποιημένους κόμβους που υπάρχουν σε κάθε εκτέλεση του αλγορίθμου. Αφορά το setup, go, και insert disruption κουμπί, όπου βλέπουμε για ένα συγκεκριμένο διάγραμμα



Plot

Name: Conflicts

X axis label: turns X min: 0 X max: 10

Y axis label: conflicts Y min: 0 Y max: 10

☒ Auto scale? ☐ Show legend?

Plot setup commands

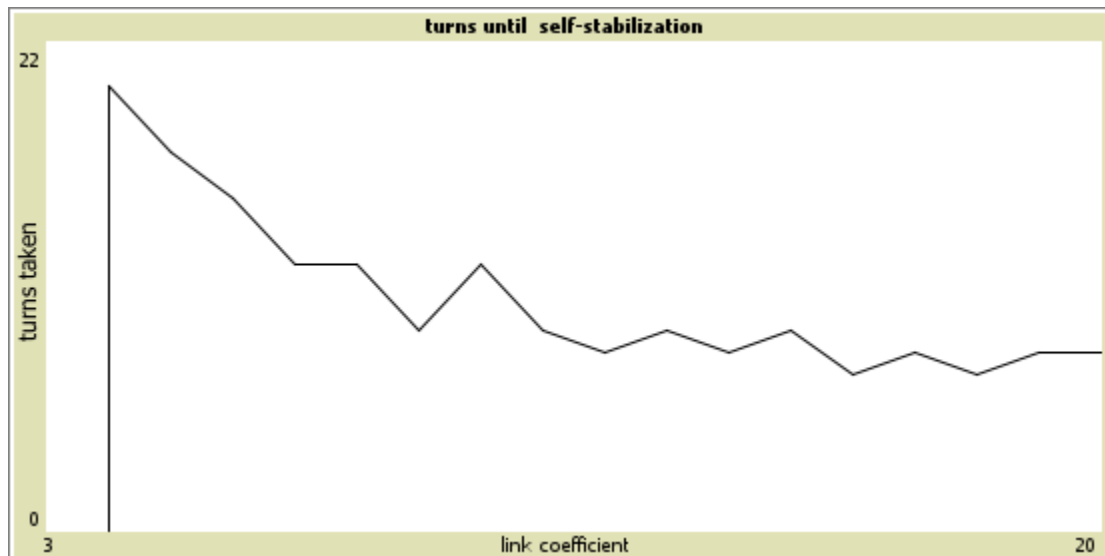
Plot update commands

Color	Pen name	Pen update commands	
	default	plot count turtles with [stabilized = false]	

Add Pen

OK Apply Help Cancel

- turns until self-stabilization: Δείχνει τον μέσο αριθμό γύρων που παίρνουν όλοι οι κόμβοι να αυτό σταθεροποιηθούν, με βάση ένα συγκεκριμένο αριθμό κόμβων, που θέτουμε στο number-of-nodes, ένα percentage-of-disruption και link-coefficient από 4 έως 20. Εμφανίζει τα αποτελέσματα του Search mean timers



Plot

Name: turns until self-stabilization



X axis label: link coefficient X min: 3 X max: 20

Y axis label: turns taken Y min: 0 Y max: 2

☒ Auto scale? ☐ Show legend?

Plot setup commands

Plot update commands

Color	Pen name	Pen update commands	
	default	plotxy link-coefficient turns-taken	 

Add Pen

OK Apply Help Cancel

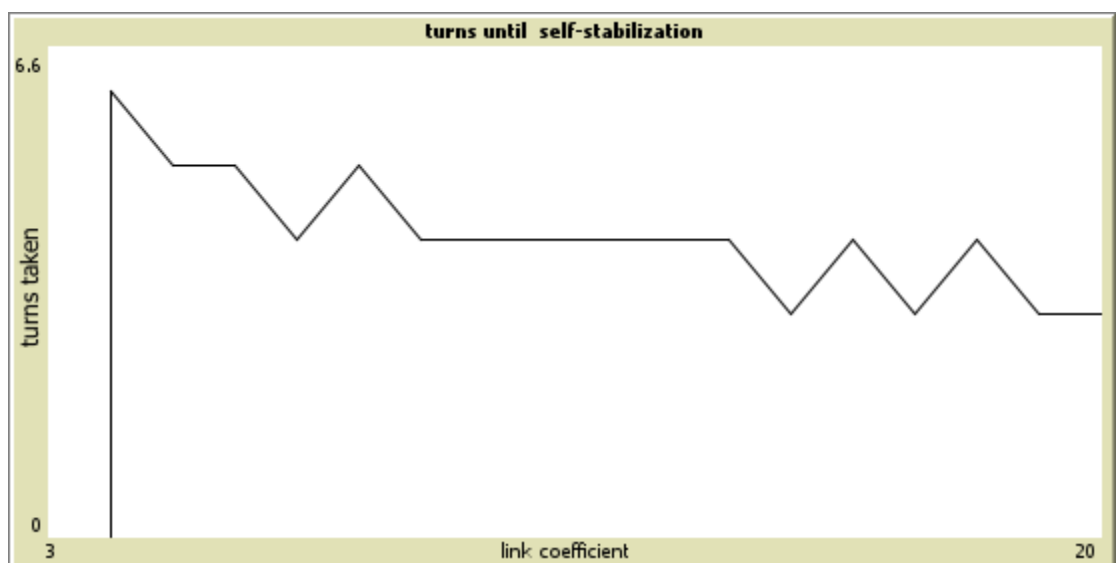
## Χρόνοι Σταθεροποίησης του Αλγορίθμου

Θα αναζητήσουμε τους χρόνους για number-of-nodes: 100,500,1000 και 2000 αριθμοί κόμβων και percentage-of-disruption :0,75 ,οπότε το 75% θα επηρεαστεί. Θα τρέξουμε και για τα τρία disruption type, ώστε να μπορεί να συγκριθεί η αρχική σύγκλιση, δηλαδή το none, με τα δύο είδη διαταραχών που έχουμε επιλέξει. Για κάθε βαθμό κόμβου, το τρέχουμε 10 φορές και παίρνουμε το μέσο όρο. Μειώνουμε το συνολικό αριθμό γύρων που πήρε ο αλγόριθμος κάθε φορά κατά 1,για να βγάλουμε το redundancy που είπαμε στην αρχή.



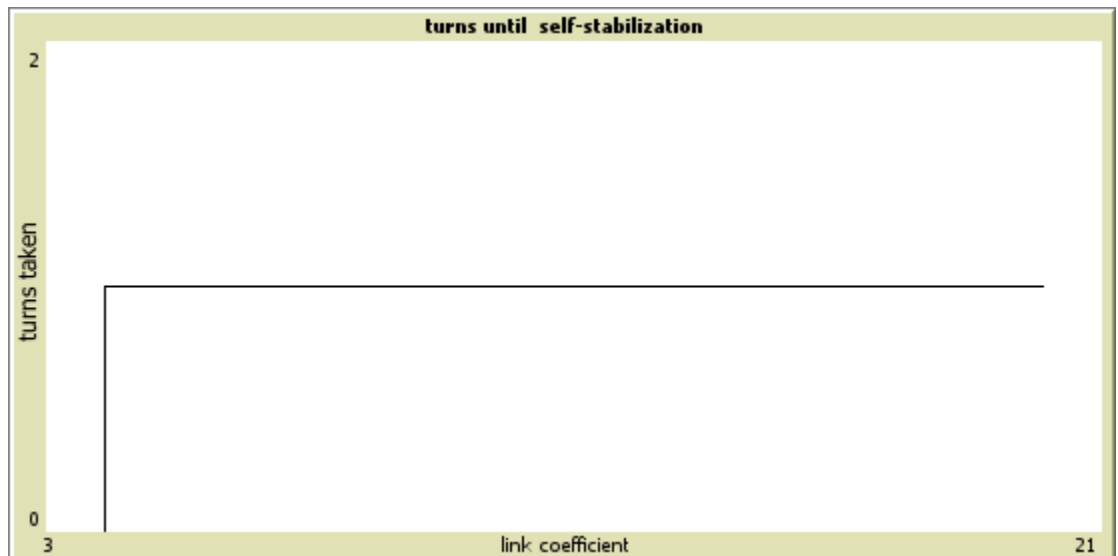
number-of-nodes=100

- disruption-type:"none"



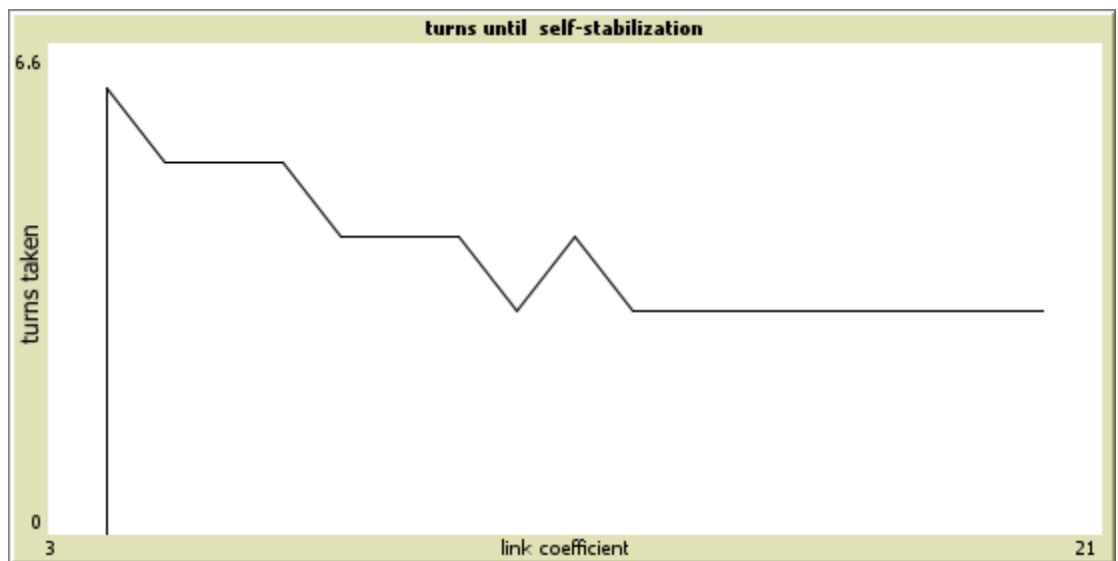
- link coefficient=4, turns taken=6
- link coefficient=12, turns taken=4
- link coefficient=20, turns taken=3

- disruption-type:"pointers"



- Σταθερά 1 γύρος για όλα τα link-coefficient

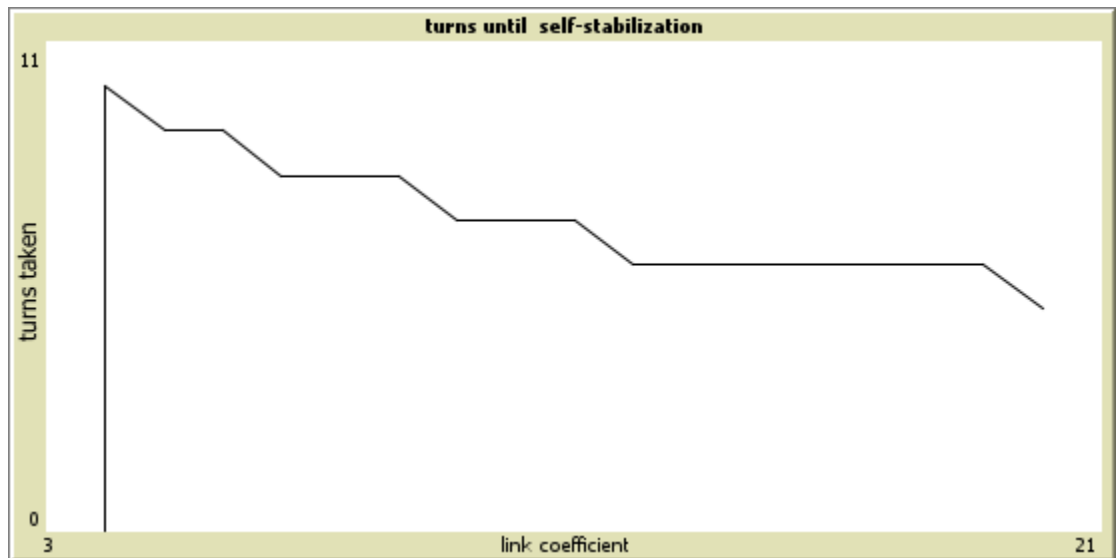
- Disruption-type: "distance from root"



- link coefficient=4, turns taken=6
- link coefficient=12, turns taken=4
- link coefficient=20, turns taken=3

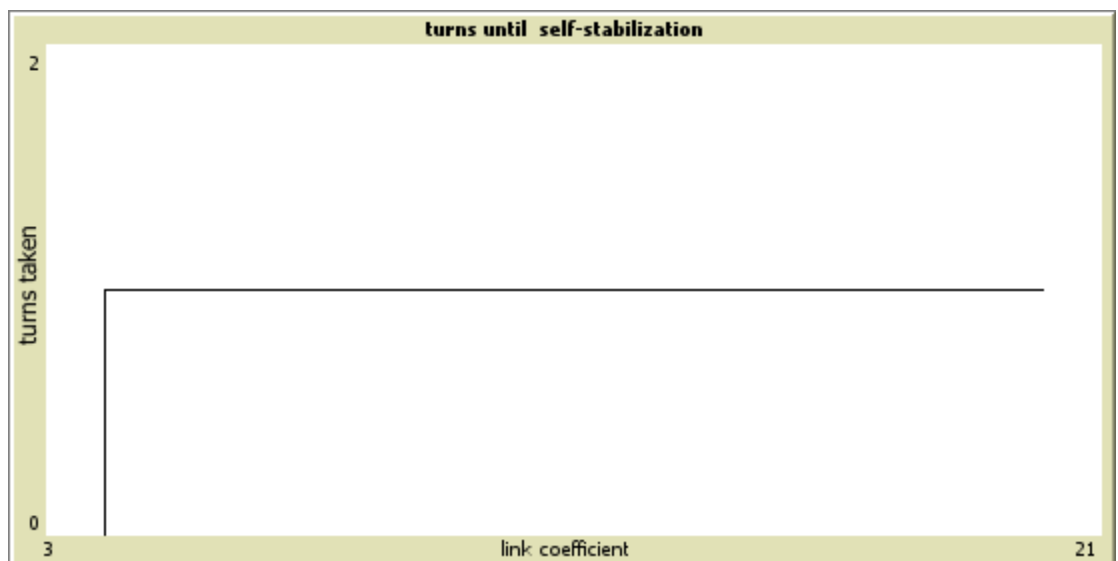
number-of-nodes=500

- Disruption-type: "none"



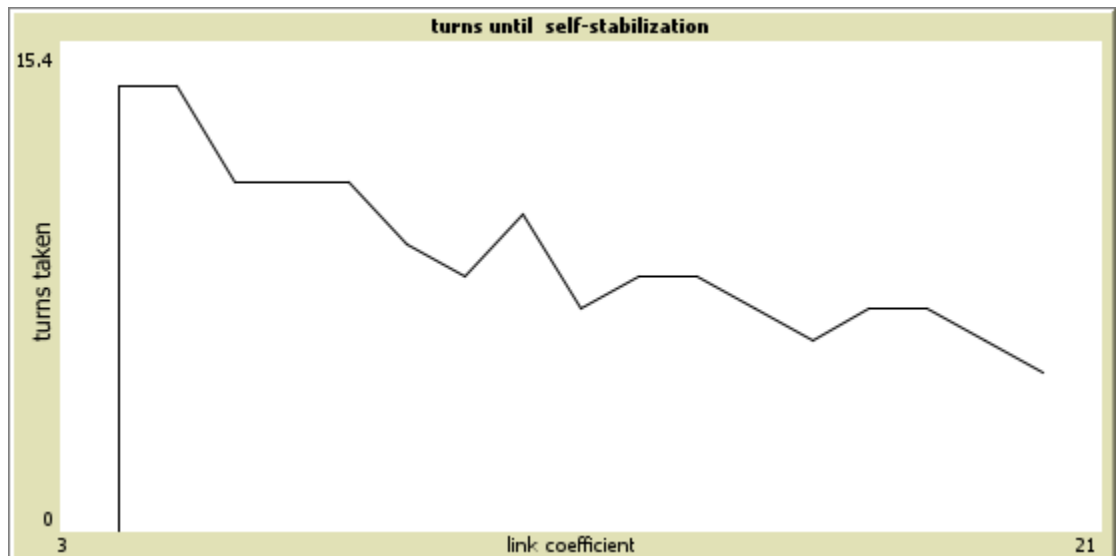
- link coefficient=4, turns taken=10
- link coefficient=12, turns taken=7
- link coefficient=20, turns taken=5

- disruption-type: "pointers"



- Σταθερά 1 γύρος για όλα τα link-coefficient

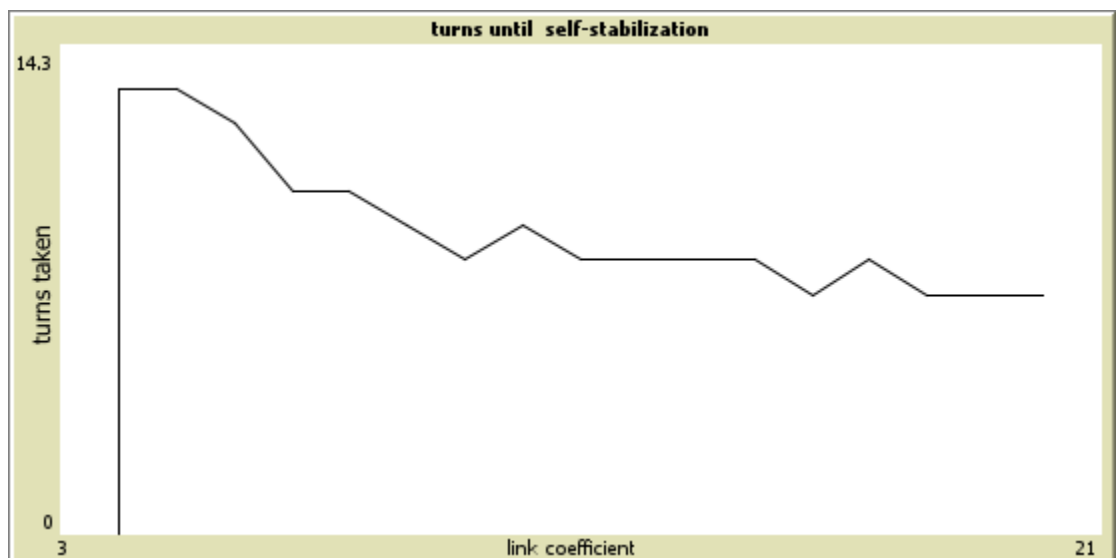
- Disruption-type: "distance from root"



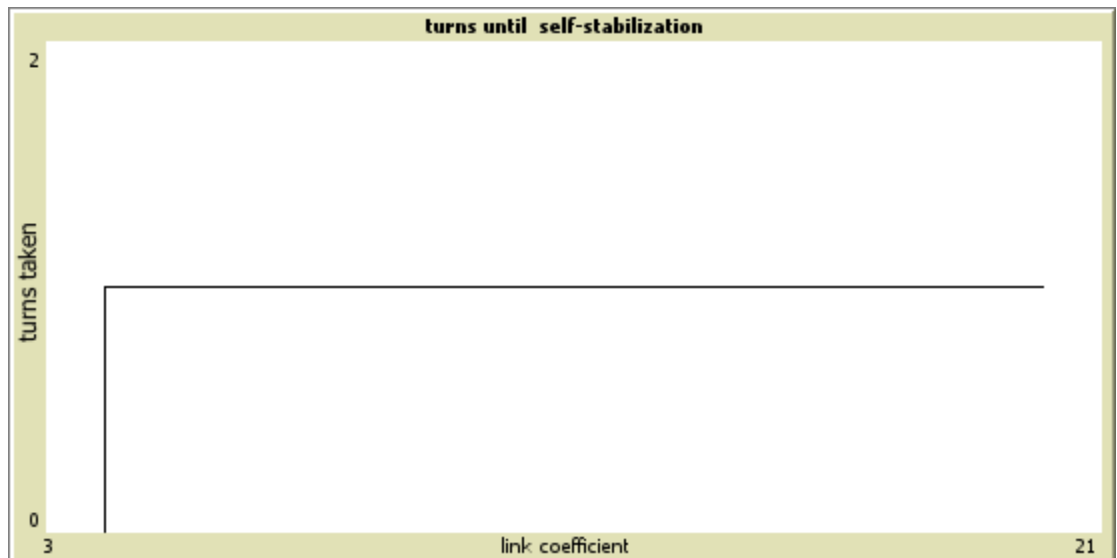
- link coefficient=4, turns taken=14
- link coefficient=12, turns taken=7
- link coefficient=20, turns taken=5

number-of-nodes=1000

- Disruption-type: "none"

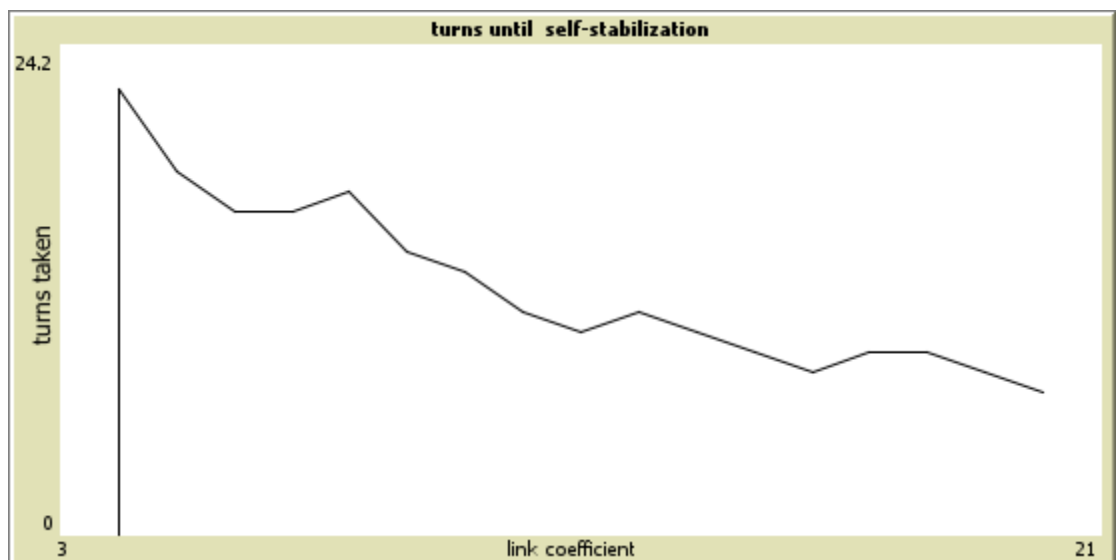


- link coefficient=4, turns taken=13
- link coefficient=12, turns taken=8
- link coefficient=20, turns taken=7
- disruption-type: "pointers"



- Σταθερά 1 γύρος για όλα τα link-coefficient

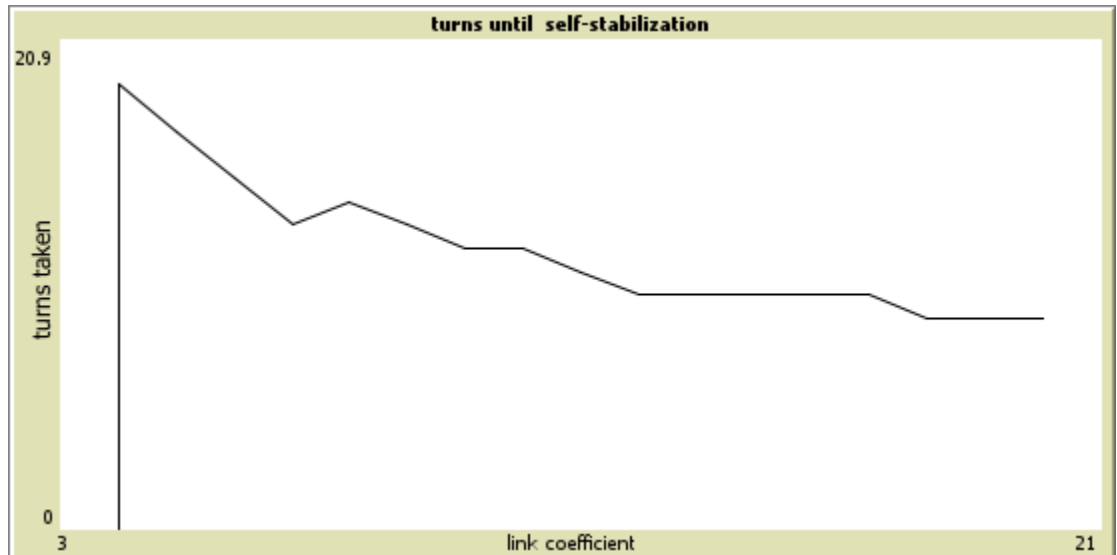
- Disruption-type: "distance from root"



- link coefficient=4, turns taken=22
- link coefficient=12, turns taken=10
- link coefficient=20, turns taken=7

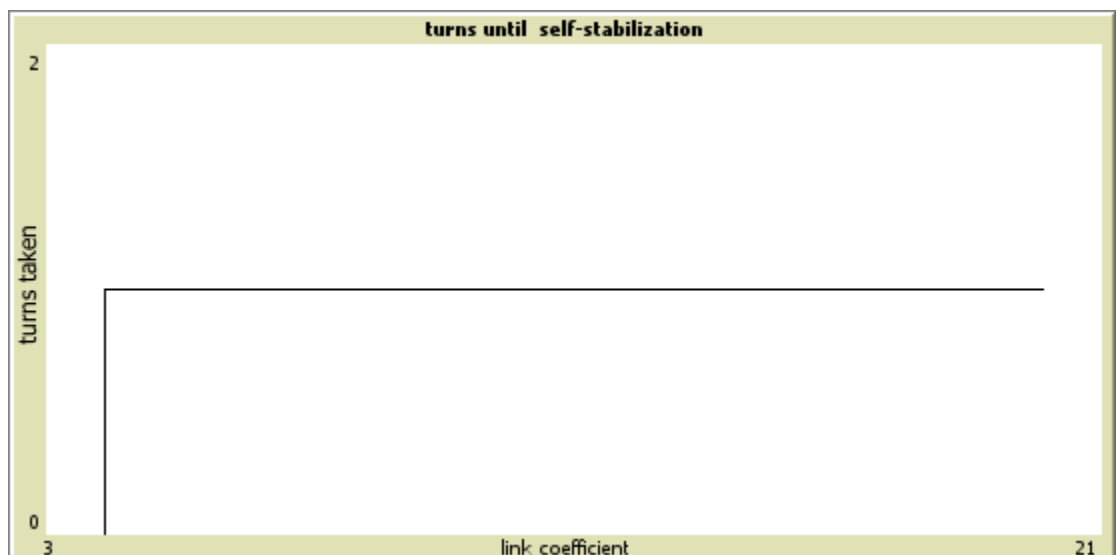
number-of-nodes=2000

- Disruption-type: "none"



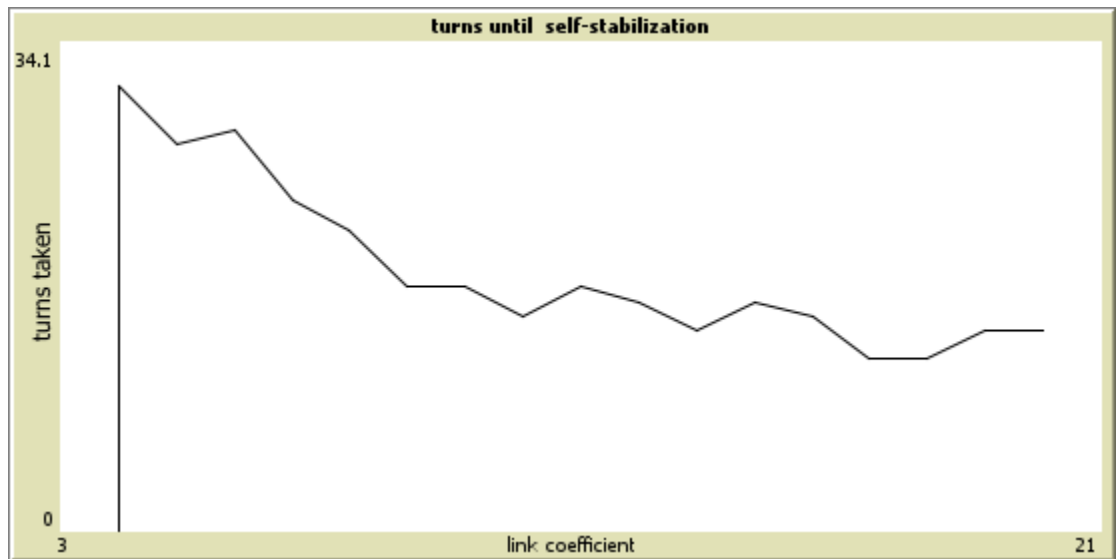
- link coefficient=4, turns taken=19
- link coefficient=12, turns taken=11
- link coefficient=20, turns taken=9

- disruption-type: "pointers"



- Σταθερά 1 γύρος για όλα τα link-coefficient

- Disruption-type: "distance from root"



- link coefficient=4, turns taken=31
- link coefficient=12, turns taken=17
- link coefficient=20, turns taken=14

## Σχολιασμός Αποτελεσμάτων

Μπορούμε να παρατηρήσουμε ότι το pointer disruption από μόνο του επιλύεται πάρα πολύ γρήγορα. Αυτό συμβαίνει γιατί ο κάθε κόμβος ξέρει ποια είναι η πραγματική του απόσταση, δεν επηρεάζεται αυτή η απόσταση από το που δείχνει ο δείκτης στον αλγόριθμο μας. Μας νοιάζει να έχει ακμές το δέντρο σε όλους τους κόμβους και να δούμε πόσους κόμβους θα διασχίσουμε για να φτάσουμε στην ρίζα, όχι το ποιο θα είναι το μονοπάτι. Η αυτοσταθεροποίηση γίνεται σε ένα γύρο.

Για το distance from root disruption, έχει ενδιαφέρον το γεγονός ότι η διαταραχή απόστασης από την ρίζα παίρνει παραπάνω γύρους, σε όλους σχεδόν τους βαθμούς κόμβων, από την σταθεροποίηση του συστήματος όταν πρωτοδημιουργείται το σύστημα, παρόλο που κάποιοι κόμβοι θα έχουν την σωστή απόσταση.

Εάν τρέξουμε το σύστημα βήμα-βήμα για ένα τυχαίο σύστημα, μπορούμε να δούμε ότι ενώ στην πρώτη σταθεροποίηση του συστήματος, οι κόμβοι πιο κοντά

στην ρίζα διαδίδουν την σωστή απόσταση και δείκτες κατά ένα βάθος και παραπάνω ανά γύρο. Λόγω του ότι η netlogo τρέχει κατά σειρά και όχι παράλληλα, τις περισσότερες φορές η διάδοση γίνεται πιο γρήγορα.

Αλλά στο distance from root disruption, υπάρχει κίνδυνος ένας κόμβος να του τεθεί τυχαία απόσταση από την ρίζα, μικρότερη από αυτήν που πρέπει να έχει κανονικά. Επειδή η μικρότερη τιμή κυριαρχεί στους γείτονες, παίρνει παραπάνω χρόνο για να επανέλθει στη σωστή τιμή.

## Άσκηση 2

### Μέρος 1

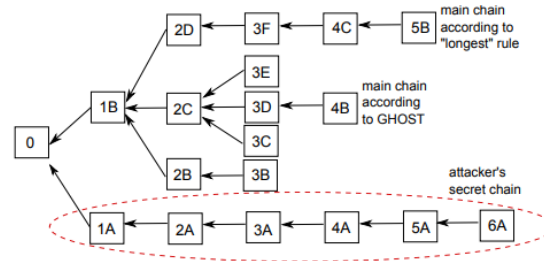
Όταν έχεις ένα σύστημα blockchain πρέπει να αποφασίσεις στον μηχανισμό επιλογής επόμενου block και επικρατέστερης αλυσίδας. Μερικές φορές το ένα προκύπτει από το άλλο, πχ σε περίπτωση Proof of Work συστημάτων η επιλογή της μακρύτερης αλυσίδας είναι η πιο συχνή λόγω του ότι για τα μπλοκ της αλυσίδας ξοδεύτηκαν αρκετοί πόροι και δεν θέλουμε πρώτον σπατάλη και δεύτερον είναι πάρα πολύ δύσκολο κάποιος να την αλλοιώσει με κακόβουλες βλέψεις.

Αυτό όμως δεν λειτουργεί ακριβώς όπως θα θέλαμε με άλλους μηχανισμούς όπως Proof of Stake, όπου κάποιος με το μεγαλύτερο stake θα μπορεί να παράξει μόνος του την μακρύτερη αλυσίδα και να την αλλάξει με σκοπό το δικό του συμφέρον. Άλλοι μηχανισμοί επιλογής της κύριας αλυσίδας είναι η επιλογή της **βαρύτερης αλυσίδας** και το πρωτόκολλο **GHOST** ( Greedy Heaviest-Observed Sub Tree ) [1] [\[https://eprint.iacr.org/2013/881.pdf\]](https://eprint.iacr.org/2013/881.pdf).

Στην επιλογή της **βαρύτερης αλυσίδας**, διαλέγεται να συνεχιστεί η αλυσίδα που έχει αθροιστικά το μεγαλύτερο βάρος από τις άλλες ως προς μία μετρική (πχ effort σε PoW και stake σε PoS). Το καλό αυτής της επιλογής είναι ότι είναι πιο ευέλικτη από την πιο απλή επιλογή της μακρύτερης αλυσίδας καθώς ορίζεται ανάλογα με τις ανάγκες η μετρική του βάρους αλλά επίσης δεν είναι τόσο εύκολο να λάβει κάποιο αυτόβουλο μέρος στην αλυσίδα καθώς θα χρειαζόταν σημαντικό μέρος από την εκάστη μετρική (πχ υπολογιστική ισχύ ή οικονομία). Σε αντιστοιχία όμως ανάλογα με την μετρική υπάρχει ακόμα κίνδυνος για μερική κεντροποίηση λόγω της συσσώρευσης κάποιων πόρων από λίγους χρήστες. Επίσης δεν γίνεται να διαβαθμιστεί τόσο καλά, χτυπάει ένα ταβάνι απόδοσης όπως και το longest chain.



Από την άλλη μεριά, το **GHOST** επιλέγει το επόμενο block από το υπό-δέντρο των αλυσίδων με το πιο βαρύ υποδέντρο, αξιοποιώντας έτσι την ύπαρξη και γειτονικών αλυσίδων στο υπο-δέντρο για την επιλογή.



**Fig. 3.** A block tree in which the longest chain and the chain selected by GHOST differ. An attacker's chain is able to switch the longest chain, but not the one selected by GHOST.

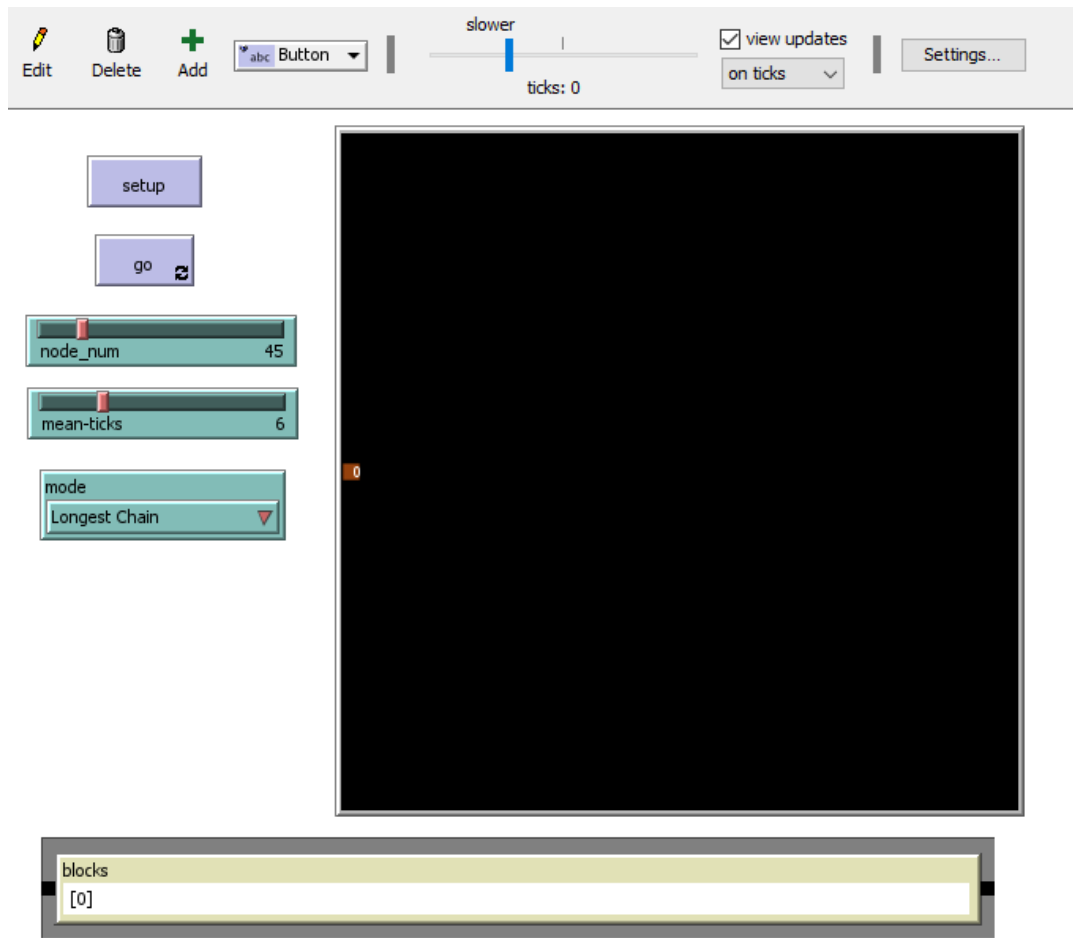
Πχ στο παράδειγμα πιο πάνω σύμφωνα με το longest chain protocol η επιλογή θα γινόταν ή στον attacker ή στο longest chain αλλά με το GHOST θα γίνει στην μεσαία αλυσίδα αφού φαίνεται ότι οι περισσότεροι προτίμησαν εκείνο το μονοπάτι για να προσθέσουν το block τους όταν ήρθε η ώρα της επιλογής. Με αυτόν τον τρόπο επιτυγχάνεται πιο γενική αξιολόγηση της αλυσίδας και προσμετρείται το βάρος κάθε block παιδιού του κάθε genesis block. Επίσης είναι αρκετά πιο scalable από το longest chain καθώς σύμφωνα με το [1] επιτυγχάνεται αύξηση του throughput και των συνολικών transactions μέσα σε ένα block χωρίς να υπονομευτεί η ασφάλεια. Αρνητικό αυτής της υλοποίησης είναι ότι είναι αρκετά πιο περίπλοκο στην υλοποίηση και τον υπολογισμό του βάρους κάθε υπο-δέντρου.

[1] - <https://eprint.iacr.org/2013/881.pdf>

## Μέρος 2

Για την υλοποίηση του δεύτερου μέρους κάναμε μερικές καίριες υποθέσεις για την διευκόλυνση μας όπως προτείνεται και από την εκφώνηση.

Αρχικά να τονιστεί ότι ο κώδικας στην netlogo είναι γεμάτος επεξηγητικά σχόλια για τυχόν έξτρα πληροφορίες που μπορεί να χρειαστούν. Ακολουθεί screenshot από το περιβάλλον υλοποίησης μετά από ένα setup:

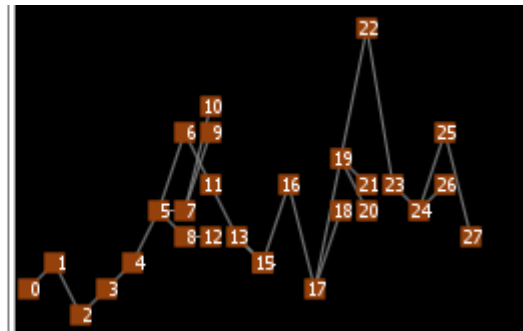
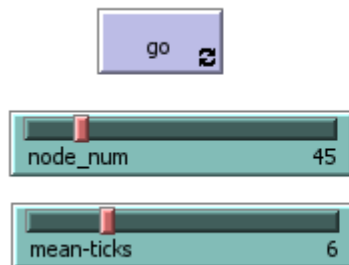


Οι υποθέσεις που κάναμε για την υλοποίηση είναι

- ◆ Στο μοντέλο μας κάθε **κόμβος** και κάθε **block** είναι ένας ξεχωριστός agent που δημιουργείται μέσω των *breeds* για καλύτερη διαχείριση μέσα στον κώδικα.
- ◆ Το **blockchain** στην ολότητα του είναι μία **global λίστα** που περιέχει όλα τα id (headers) των blocks που δημιουργήθηκαν, προσομοιώνοντας την άμεση γνώση του κάθε κόμβου του blockchain και τον υποτιθέμενων συναλλαγών που έγιναν. Επίσης λόγω του ότι είναι global, σε κάθε βήμα κάθε κόμβος ξέρει όλη την αλυσίδα, μία ακόμα παραδοχή που κάναμε.
- ◆ Κάθε block αποθηκεύει το id του και το id του block στο οποίο δείχνει. Έτσι προσομοιώνεται πιστά η φύση της αλυσίδας και των διακλαδώσεων όπου κάθε σε block μπορεί να δείχνει παραπάνω από ένα blocks.
- ◆ Για χάρη της υλοποίησης και ορθής λειτουργίας, στο setup δημιουργείται ένα αρχικό block από τον observer και έπειτα τρέχει το μοντέλο όπου κάθε κόμβος προσπαθεί να φτιάξει το δικό του block.
- ◆ Όπως δίνεται να εννοηθεί από την εκφώνηση, το σύστημα συναίνεσης στην προκειμένη είναι ένα **Proof of Work** σύστημα καθώς όπως καταλάβαμε ο χρόνος παραγωγής ενός block εξαρτάται από μια τιμή της τυχαίας μεταβλητής (**nonce**) που ακολουθεί την κατανομή Poisson. Άρα με μία αναμενόμενη μέση τιμή ως χρόνο παραγωγής (**mean-ticks**) κάθε μπλοκ προσπαθεί να φτιάξει αυτόνομα από τους υπόλοιπους το block του ανάλογα με το δικό του *nonce*.

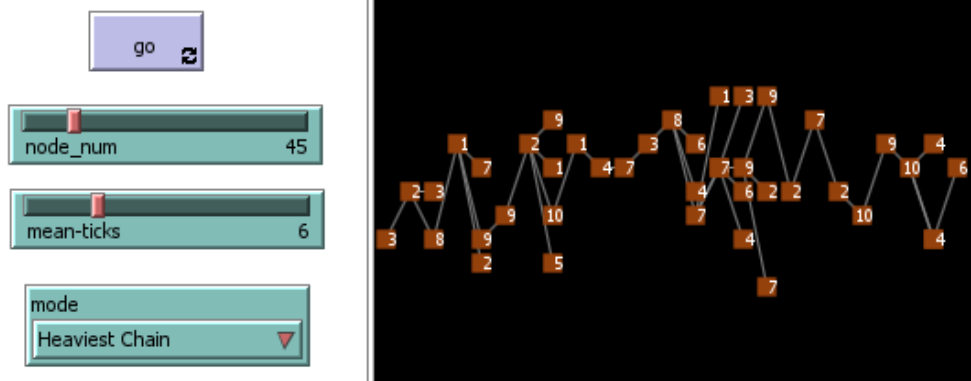
- ◆ Υλοποιήσαμε την διαδικασία με το σκεπτικό ότι σε κάθε tick περνάει και μία χρονική μονάδα. Όταν δημιουργείται ένα block, όλοι οι κόμβοι μαθαίνουν για αυτό και ξεκινούν από την αρχή και με νέο nonce να φτιάχνουν το νέο block. Το nonce συγκρίνεται από κάθε κόμβο με τα ticks που πέρασαν από την δημιουργία του τελευταίου block και αν είναι ίσα, θεωρούμε ότι ο κόμβος βρήκε κατάλληλο nonce και προτείνει το επόμενο block στην αλυσίδα.
- ◆ Για λόγους παρατήρησης, οι αλυσίδες αναπαρίστανται ως σύνδεσμοι μεταξύ των blocks. Κάθε block έχει ως συντεταγμένη x το βάθος του στην αλυσίδα (μόνο αυξάνεται) και σαν y ένα τυχαίο y (αυτό για λόγους ευκολίας της υλοποίησης, ήταν πιο δύσκολο να θέσουμε σαν y το y του block που συνδέεται το νέο block και να μην φτάσει στα όρια του κόσμου όταν αυτά αυξάνονται). Έτσι για να δείξουμε σε πιο block συνδέεται το κάθε νέο block του είπαμε να δημιουργήσει ένα link με το block που έχει σαν id το parent\_id του νέου block.
- ◆ Όσον αφορά την μέθοδο επιλογής του block για την σύνδεση με το επόμενο, υλοποιήσαν 2 μέθοδοι: Της **μακρύτερης αλυσίδας** και της **Βαρύτερης αλυσίδας**.
  - Στην περίπτωση της **μακρύτερης αλυσίδας**, κάθε κόμβος αναζητά στην συνολική αλυσίδα τους κόμβους με το μεγαλύτερο βάθος και επιλέγει τυχαία έναν από αυτούς ως το ιδανικό. Η επιλογή γίνεται τυχαία για να προσομοιώσει κάπως την μερική άγνοια της καλύτερης επιλογής της αλυσίδας από όλους τους κόμβους την ίδια χρονική στιγμή.

Σαν αποτέλεσμα έχει μερικές φορές να δημιουργούνται παράλληλες αλυσίδες που πιθανόν να τρέξουν παράλληλα για μερικές γεννήσεις blocks αλλά όταν σε μία από αυτές προστεθεί ένα μόνο block σε ένα βήμα, τότε η κύρια αλυσίδα θα θεωρηθεί αυτή και θα συνεχιστεί από εκεί.



Πχ στην συγκεκριμένη περίπτωση στο block 5 συνδέθηκαν 3 blocks και σε αυτά συνδέθηκαν συνολικά 4 νέα. Όμως μόνο ένα συνδέθηκε στο 13 και έπειτα η αλυσίδα συνεχίστηκε από αυτό. Άρα όντως λειτουργεί με τον κανόνα της μακρύτερης αλυσίδας.

- Για την Βαρύτερη αλυσίδα έπρεπε κάπως να προσομοιώσουμε το βάρος κάθε block. Αυτό γίνεται μέσω μιας μεταβλητής κάθε block που παίρνει τυχαία τιμές μεταξύ 0 και 1. Για λόγους ευανάγνωστης, σαν label όταν τίθεται σαν mode λειτουργίας το Heaviest Chain, ανατίθεται το πρώτο δεκαδικό ψηφίο του βάρους.



Αυτό που παρατηρήθηκε κατά την υλοποίηση είναι ότι δεν υπήρχαν παράλληλες αλυσίδες, όπου υπήρχαν διακλαδώσεις ήταν ένα μόνο το block με το μεγαλύτερο βάρος οπότε ήταν η πρώτη επιλογή κάθε κόμβου. Αφού δεν συνεχίστηκαν παράλληλες αλυσίδες το θέμα της βαρύτερης αλυσίδας μετατρέπεται σε βαρύτερο block, που στην πραγματικότητα επειδή κάθε κόμβος δεν μπορεί να ξέρει όλη την αλυσίδα δεν γίνεται αλλά στα πλαίσια της υλοποίησης μας βγάζει νόημα.

Για να καταφέρει η μακρύτερη αλυσίδα να μην έχει πολλές διακλαδώσεις πρέπει να βρεθεί μία ισορροπία στο πλήθος των κόμβων που προσπαθούν να φτιάξουν blocks και της μέσης τιμής του nonce που θέτουμε. Αν ένα από τα δύο ξεφεύγει σαν τιμή τότε ή πάρα πολλά παράλληλα blocks θα δημιουργούνται (χαμηλό nonce και μεγάλο πλήθος nodes) ή πολλή σπάνια δημιουργία blocks (στην αντίστροφη περίπτωση) άρα και μη παρατήρηση διακλαδώσεων.

Αυτό σαν παρατήρηση **επιβεβαιώνεται** από την πραγματικότητα που το **Bitcoin** προσπαθεί συνεχώς να προσαρμόζει την δυσκολία των *hash puzzles* για τα *nonce* (αντίστοιχα το *mean-ticks* στο μοντέλο μας) ώστε να μένει σε ένα *σταθερό χρόνο* η παραγωγή κάθε νέου block όσο αυξάνονται οι κόμβοι ή η υπολογιστική ισχύς αυτών (αυξάνουμε το πλήθος των nodes αντίστοιχα)

## Κώδικας προγραμματιστικών Ασκήσεων

### Άσκηση 1

```
turtles-own [
  is-root ;check if it the root node or not
  distance-from-root ;distance from the root that the node knows at this
```

```

moment
  pointer ;pointer to a neighbor one distance lower
  stabilized ;check if any property have changed at the current turn
]
globals [diameter
  turns-taken ; for the conflict-timer-search,it show the amount of turns
the system has to stabilize itself
]

to setup
  clear-all
  reset-ticks
  resize-world -25 25 -20 20 ;to fit more nodes
  ifelse use-user-random-seed-at-setup ;either chosen from input seed or
random
    [random-seed user-random-seed]
    [random-seed new-seed]
  setup-graph
end

to go
  if count turtles with [stabilized = false] = 0 [stop] ;stop forever go
loop when system is stabilized
  go-algorithm
  tick
  check-for-disconnect-error ticks

end

;creating the system
to setup-graph
  set-default-shape turtles "circle"
  set diameter number-of-nodes ;we don't care about the exact diameter, it
is just essential to be bigger than any distance from root can get
naturally from the algorithm
  create-turtles number-of-nodes[
    set color blue
    set size 0.75
    set is-root false
    set stabilized false
    set color green; green means no stabilized
    set distance-from-root diameter ;it could be empty, but taking a surely
bigger than anything else distance from root, it will make it sure that it
will change immediattely
    set pointer nobody
    set label distance-from-root
    setxy random-xcor random-ycor
  ]
  set diameter count turtles

  move-to one-of patches with [not any? other turtles in-radius 1] ;to
leave some space between nodes
  create-connections
  ask one-of turtles[ ;set randomly a node as root
    set color red
    set size 1 ; make it a bit bigger to see it better
    set is-root true
    set distance-from-root 0
    set stabilized true
  ]

```

```

]
ask turtles [
  set label distance-from-root
]

```

end

to create-connections ; create the edges of the graph, with each nodes creating link-coefficient (degree of node) edges to it's closest nodes

```

ask turtles [

  let remaining-available-turtles count turtles - count my-links - 1
;it checks if any other nodes have already created edges going to the
particular node
  if remaining-available-turtles > 0 [ ; check if new edges can be
created without creating multiple links between two nodes
    let closest-turtles nobody
    ifelse remaining-available-turtles < link-coefficient ;if we have
less available nodes that the choosen node-degree, we create edges to the
available turtles
      ;not link-neighbor? myself means than
      [
        set closest-turtles min-n-of remaining-available-turtles other
turtles with [not link-neighbor? myself] [distance myself]
      ]
      [
        set closest-turtles min-n-of link-coefficient other turtles with
[not link-neighbor? myself] [distance myself]
      ]
      create-links-with closest-turtles

    ]
  ]
end

```

to go-algorithm ;the algorithm shown in theory has been created entirely at the two commands below

```

rootProcess
notRootProcess
ask turtles [
  set label distance-from-root ;for each node, show the distance
]
end

```

to rootProcess ;root node can be affected by distance from root disruption ,but not pointer

```

ask turtles with [is-root = true][
  ifelse distance-from-root != 0 [
    set distance-from-root 0
    set stabilized false
  ]
  [
    set stabilized true ;if nothing changed, the node is stabilized and
without disruption, this will change that way.
  ]
]
end

```

;Initially, we check if the distance from root is the minimum distance found from neighbors plus 1.If it is false, we fix it. If it is true, we check if the pointer shows to a neighbor that it is one step closer to the root. If it is false,we point the pointer to a random neighbor with the minimum minimum distance.  
;For each turn, at every node, only one action of the will hapen, even though it would make sense.This happens because the book says in preliminaries section, that each action is excluding each other

to notRootProcess

```
ask turtles with [is-root = false][

  let is-distance-from-neighbors-ok true
  let is-pointer-ok true
  let min-distance [distance-from-root] of min-one-of link-neighbors
  [distance-from-root]

  if distance-from-root - 1 != min-distance [set is-distance-from-
neighbors-ok false ]

  if not is-distance-from-neighbors-ok [
    let true-distance min-distance
    if min-distance > diameter [set true-distance diameter - 1] ;we keep
the diameter as an upper limit
    set distance-from-root true-distance + 1
  ]
  if pointer = nobody or (is-distance-from-neighbors-ok and [distance-
from-root] of pointer != distance-from-root - 1 ) [set is-pointer-ok
false]

  if not is-pointer-ok [
    if pointer != nobody [ask link [who] of self [who] of pointer [set
color white]] ; we return the color of the pointer to the normal color of
red
    set pointer one-of link-neighbors with [distance-from-root = min-
distance]
    ask link [who] of self [who] of pointer [set color red] ;as we cannot
have both directed and directed links at the same time without breeds.And
we can see who is pointer to whom,based of distance from root
  ]

  ifelse is-distance-from-neighbors-ok = true and is-pointer-ok = true ;if
nothing changed, the node is stabilized and without disruption, this will
change that way.
  [
    set stabilized true
    set color blue
  ]
  [
    set stabilized false
    set color green
  ]
]
end
```

to insert-disruption ;it is change a value of the nodes that have randomly

choosed, the amount of nodes that it will get taken is based at percentage of distrupction

```
let disrupted-nodes number-of-nodes * percentage-of-disruption ;a
percentage of the total nodes
set disrupted-nodes ceiling disrupted-nodes ;we don't want float numbers
if disrupted-nodes < 1 [set disrupted-nodes 1] ; to take at least a node
to insert the disruption
ask n-of disrupted-nodes turtles [ ;some nodes will get choosen even at
choice none
```

```
if disruption-type = "pointer" [
  if is-root = false[ ;root doesn't have pointer

    ifelse pointer = nobody ;if it hasn't been assigned yet, assign a
    random neighbor.
    [set pointer one-of link-neighbors ]
    [
      ask link [who] of self [who] of pointer[set color white] ; a
      purely visual change, to show that it is not the pointer
      let p-pointer-null random-float 1
      ifelse p-pointer-null < 0.5 ;we choose between destroying the
      pointer or changing the pointer to somewhere else
      [set pointer nobody]
      [set pointer one-of link-neighbors who-are-not pointer]
    ]

    if pointer != nobody [ask link [who] of self [who] of pointer[set
    color red]]
  ]

]
if disruption-type = "distance from root"
[
  set distance-from-root random diameter * 2 ;it can be bigger than
  diameter,We don't want to make choosing a smallen number a minority
  if is-root = false [set color green]
  set label distance-from-root
]
set stabilized false
if is-root = false [set color green]
]

end
```

to search-mean-timers ;it has a behavior similar to behavior search, to see the turns taken to reash stabilization

```
set link-coefficient 4 ;experimentally, it is the smallest number that
hasn't yet given asubgraph unconnected to the main graph
clear-all
reset-ticks
while [link-coefficient <= 20] [ ;we could have even bigger node degree,
but firstly,it would take more time to run the experiments and second, a
complete graph can reach stability very quickly
```



```

set turns-taken 0 ;

random-seed user-random-seed ; we assigned random seed 478231
repeat 10 [
  clear-turtles
  setup-graph
  conflict-timer-search ;
]
set turns-taken ceiling (turns-taken / 10)
tick
set link-coefficient link-coefficient + 1
]
end

to conflict-timer-search ; it is the command responsible for the individual
execution of each system for search-mean-timers
let current-turns-taken 0

while [count turtles with [stabilized = false ] > 0] [
  go-algorithm ; we run the algorithm without ticks
  set current-turns-taken current-turns-taken + 1
]
check-search-conflict-timers-for-errors ;check for errors for a not legit
system configuration

if disruption-type != "none"[ ;if disruption-type=none, we need to take
into account the current-turns-taken of the first part of the loop
set current-turns-taken 0 ;reset the counter
insert-disruption
while [count turtles with [stabilized = false ] > 0 ][
  go-algorithm
  set current-turns-taken current-turns-taken + 1
  check-for-disconnect-error current-turns-taken
]
check-search-conflict-timers-for-errors ;check for errors for a not
legit system configuration
]

set turns-taken turns-taken + (current-turns-taken - 1 )
end

```

to check-search-conflict-timers-for-errors ;Function that check the experiments for the case of no legit conditions,after the end of an execution

```

ask turtles with [is-root = true and distance-from-root > 0] [
  user-message "Root nose has distance from root over zero."
  stop
]
ask turtles with [is-root = false] [
  let min-distance [distance-from-root] of min-one-of link-neighbors
[distance-from-root]
  if distance-from-root - 1 != min-distance[
    user-message "distance from root from not root node is not correct."
    stop
  ]
  if pointer = nobody [
    user-message "pointer is empty."
    stop
  ]
]

```

```

]
if [distance-from-root] of pointer != distance-from-root - 1 [
  user-message "pointer point to an invalid neighbor."
  stop
]
]
end

```

**to** check-for-disconnect-error [current-turns];check for disconnected subgraph. It has to be called during the execution, to stop the execution going forever,so it will used a seperate command

```

  if current-turns > 1000 and count turtles with [stabilized = false] != 0
  [ ;if a subgraph not having any connection to the rest of the network,
  throw an error and stop the execution
    user-message "The graph is disconnected"
    stop
  ]
end

```

## Άσκηση 2

```

breed [blocks block]
breed [nodes node]

```

```

blocks-own [
  id ;; the header of the block
  parent_id ;; the header of the genesis block
  depth ;; local var showing how deep in the chain the block is
  weight ;; var that holds the weight of the block
  ;;has-next? ;; boolean to check if there's next block
]

```

```

nodes-own [
  nonce ;; symbolic var that holds the value of the poisson distribution
  ;; emulating the proof of work mechanism
  prod_block? ;; boolean showing if a node produces a block
  block_con_id ;; the id of the block to connect the new block to
]

```

```

globals [
  blockchain ;; list that holds all the ids of all blocks
  forks ;; list that holds the ids of all blocks that produce a fork in
the chain
  new_b? ;; flag used to recalculate nonce for all and restart production
  max_depth ;; var that shows the deepest block(s) depth
  new_block_tick ;; mark the tick that a new block started being produced
  y ;; holds the y coordinate of the main blockchain, used to position
correctly
]

```

```

;;;;;;;;;;;;;
;;;;;;;;; SETUP ;;;;;;;;;;
;;;;;;;;;;;;;

```

**to** setup

```

clear-all

set y 5
set blockchain [] ;; set the list of all block ids
setup-nodes
set max_depth 0 ;;
produce_block 0 0 0;; create the first block, Υποθεση για σωστη
λειτουργια του κωδικα
ask blocks [
  set depth 1
  set y 5
]
set new_b? true ;; start with all nodes getting a nonce
block_label ;; set labels based on the chosen mechanism (id / weights)

reset-ticks
end

to setup-nodes
  create-nodes node_num
  ask nodes [
    set color black
    set shape "circle"
    set size 0.5
    setxy random-xcor -10
    set prod_block? false
    set nonce 100 ;; give a default value to avoid initial state errors
  ]
end

;;;;;;;;;;;;;
;;;;;;;;; GO ;;;;;;;;;
;;;;;;;;;;;;;

to go

  ;; if a new block was produced, all nodes restart the production of a new
  block
  if new_b? [
    set new_b? false
    set max_depth max_depth + 1
    set new_block_tick ticks ;; mark the tick that a new block started
    being produced
    ask nodes [set nonce 1 + random-poisson mean-ticks] ;; get a new nonce
    for the new block
  ]

  ;; all nodes check each tick if they want to propose a block
  ask nodes [
    ;; use the current tick and the new_block_tick to check if a node
    produces a block
    let chance (ticks - new_block_tick)
    if nonce = chance [
      set prod_block? true
      set block_con_id suitable_block
    ]
  ]

  ;; temp var that holds the id of the genesis block

```

```

let fork_block position (last blockchain) blockchain

;; iterate through all the nodes that want to produce a block
repeat count nodes with [ prod_block? = true ] [
  let num_of_blocks count nodes with [ prod_block? = true ]

  ;; ask each node one at the time to produce their block
  ;; connecting it to the most suitable block they can find
  ask one-of nodes with [ prod_block? = true ] [
    set prod_block? false ;; deactivate the flag
    set fork_block block_con_id ;; keep in the local var the id of the
suitable block
  ]
  ;; the id of the block to be produced
  let new_block (last blockchain + 1)

  ;; produce the new block with from a certain parent with a new id
  produce_block new_block fork_block num_of_blocks
]

tick
end

;; reports the id of the most suitable block based on the mechanism we
choose
to-report suitable_block
  let s_block 0 ;; holds the id to return

  if mode = "Longest Chain" [
    set s_block [id] of one-of blocks with [depth = max_depth]
  ]

  if mode = "Heaviest Chain" [
    ;; list of all the weights of the deepest blocks
    let weights_list []

    ;; ask each deepest block to put their weight in the list
    ask blocks with [depth = max_depth] [
      set weights_list lput weight weights_list
    ]

    ;; suitable block is the one with the deepest block with the biggest
weight
    set s_block [id] of one-of blocks with [depth = max_depth and weight =
max weights_list]
  ]

  report s_block
end

to produce_block [ new_id new_parent_id block_num ]
  set new_b? true ;; flag to show a new block was produced

  create-blocks 1 [
    set blockchain lput new_id blockchain ;; update the blockchain
    set id new_id ;; id of the new block
    set parent_id new_parent_id ;; id of parent block
  ]

```

```

    set depth max_depth + 1 ;; depth of the new block in the chain
    set shape "square"
    set color 23
    set size 1
    set weight random-float 1
]

block_label ;; set labels based on the chosen mechanism (id / weights)
position_block new_id new_parent_id
end

to position_block [ b_id p_id ]
    let y_par random-normal 0 3
    ; last [ycor] of blocks with [id = p_id] ;; var that holds the ycor of the
    parent block
    ; if any? other blocks with [depth = max_depth] [
    ;     set y_par y_par + count blocks with [depth = max_depth]
    ; ]

    ;; used for the first block
    if b_id = p_id [
        set y_par 0
    ]

    ;; the block that runs the function sets itself in the grid
    ask blocks with [ id = b_id ] [
        setxy (-16 + max_depth) (y_par)
    ]

    ;; create link with the parent block for better visualization of the
    chain
    ask blocks with [ id = b_id and id != p_id] [
        create-link-with one-of blocks with [id = p_id]
    ]
end

to block_label
    if mode = "Longest Chain" [
        ;; for better identification
        ask blocks [set label id]
    ]

    if mode = "Heaviest Chain" [
        ;; for better identification
        ask blocks [set label round ( 10 * weight)]
    ]
end

```

