Αλγοριθμικές Τεχνικές για Δεδομένα Ευρείας Κλίμακας

3ο Φυλλάδιο Ασκήσεων

Προθεσμία υποβολής: 13/06/2025, 23:59

Ενότητα Α: Έλεγχος συνεκτικότητας γραφήματος **(Μονάδες: 3)**

Σε αυτήν την ενότητα θα ασχοληθούμε με (μη-κατευθυνόμενα) γραφήματα, με μέγιστο βαθμό **Δ=10**.

Το πρόβλημα που θα μας απασχολήσει είναι το εξής. Ας υποθέσουμε ότι ξεκινούμε από ένα κενό γράφημα (δηλαδή χωρίς ακμές) με **n** κόμβους, που είναι αριθμημένοι από **1** μέχρι και **n**. Τότε, κατασκευάζουμε τυχαίες ακμές, και τις προσθέτουμε στο γράφημα. Για την ακρίβεια, πρώτα επιλέγουμε τυχαία έναν αριθμό **x** μεταξύ **1** και **n**, και έπειτα επιλέγουμε τυχαία έναν αριθμό **y** μεταξύ **1** και **n**. Αν τύχει να έχουμε **y=x**, τότε επαναλαμβάνουμε την διαδικασία για το **y**, μέχρι να τύχουμε ένα που είναι διάφορο του **x**. Τότε, ελέγχουμε αν η ακμή **(x,y)** ανήκει στο γράφημα: αν ανήκει, τότε επαναλαμβάνουμε την δειγματοληψία ακμής (ξεκινώντας και πάλι δηλαδή από την επιλογή των **x** και **y**). Ειδάλλως, ελέγχουμε αν η προσθήκη της **(x,y)** θα παραβίαζε το όριο στον μέγιστο βαθμό. Δηλαδή, αν έχουμε ότι είτε ο **x** είτε ο **y** έχει ακριβώς **Δ** γείτονες, τότε δεν επιτρέπεται να προσθέσουμε την ακμή **(x,y)**, οπότε επαναλαμβάνουμε την δειγματοληψία ακμής. Ειδάλλως, προσθέτουμε την ακμή **(x,y)** στο γράφημα. Συνεχίζουμε αυτήν την διαδικασία, μέχρι να καταλήξουμε σε ένα συνεκτικό γράφημα.

Εφαρμόζοντας αυτήν την πιθανοτική διαδικασία, είναι εύλογο να αναρωτηθούμε: ποιο είναι το αναμενόμενο πλήθος εισαγωγών ακμών που θα πραγματοποιηθούν μέχρι να τερματιστεί η διαδικασία;

Φυσικά, υπάρχει το ενδεχόμενο αυτή η διαδικασία να σκαλώσει σε ατέρμονα βρόχο. Ο λόγος για αυτό είναι ότι, λόγω του περιορισμού στον μέγιστο βαθμό, μπορεί να τύχει όλες οι συνεκτικές συνιστώσες του γραφήματος να αποτελούν **Δ-κανονικά** γραφήματα (δηλαδή όλοι τους οι κόμβοι να έχουν βαθμό ακριβώς **Δ**), οπότε να μην γίνεται να προστεθεί καμία επιπλέον ακμή στο γράφημα, οπότε θα επαναλαμβάνονται δειγματοληψίες ακμής δίχως τελειωμό.

Αυτό είναι εύκολο να συμβεί αν το **Δ** είναι αρκετά μικρό. Για παράδειγμα, για **n=8** και **Δ=3**, αν εφαρμόσουμε την διαδικασία που περιγράψαμε, είναι πιθανό να καταλήξουμε στο ακόλουθο γράφημα, που αποτελείται από δύο συνεκτικές συνιστώσες που είναι πλήρη γραφήματα με τέσσερις κόμβους:

Όμως στην πράξη, ειδικά για το **Δ** που επιλέξαμε, και για τα **n** που θα δούμε, φαίνεται πως η πιθανότητα του ενδεχομένου να πέσουμε σε ατέρμονα βρόχο είναι αμελητέα. (Θα ήταν ένας ενδιαφέρων μαθηματικός γρίφος να υπολογιστεί ακριβώς αυτή η πιθανότητα, συναρτήσει των **n** και **Δ**, αλλά τέτοια ερωτήματα ξεφεύγουν κατά πολύ από το πλαίσιο του μαθήματός μας.)

Μέχρις εδώ, για να υλοποιήσουμε την διαδικασία που περιγράψαμε, χρειαζόμαστε έναν αλγόριθμο για έλεγχο συνεκτικότητας. Έναν τέτοιον αλγόριθμο, γραμμικού χρόνου (ή σχεδόν γραμμικού χρόνου), θα χρειαστούμε οπωσδήποτε.

Οπότε, η πρώτη μας δουλειά είναι να υλοποιήσουμε μία ρουτίνα για έλεγχο συνεκτικότητας γραφήματος. Αυτή δέχεται ως είσοδο ένα γράφημα, που αναπαρίσταται με μορφή λιστών γειτνίασης, και επιστρέφει "ΝΑΙ" ή "ΟΧΙ", ανάλογα με το αν το γράφημα είναι συνεκτικό, αντίστοιχα. Στην υλοποίηση, ίσως σας βολέψει περισσότερο να αριθμείτε τους κόμβους από **0** μέχρι και **n – 1** (αυτό δεν αλλάζει ουσιαστικά τίποτα).

Η ρουτίνα θα κάνει έλεγχο συνεκτικότητας ως εξής. Θα επιλέγει τυχαία έναν κόμβο, και θα εφαρμόζει BFS από αυτόν. Αν η BFS καταφέρει να εξερευνήσει όλους τους κόμβους, τότε το γράφημα είναι συνεκτικό. Ειδάλλως, δεν είναι συνεκτικό.

Εδώ επιτρέπουμε οποιαδήποτε υλοποίηση γραμμικού ή σχεδόν γραμμικού χρόνου, δηλαδή χρονικής πολυπλοκότητας είτε **O(n+m)** είτε **O((n+m)logn)**, όπου **m** είναι το πλήθος ακμών του γραφήματος.

Όμως, θα δοθούν 0.25 μονάδες μόνο αν υλοποιήσετε αλγόριθμο γραμμικού χρόνου. Συγκεκριμένα, το εξής σημείο χρειάζεται λιγάκι προσοχή. Για να πραγματοποιήσετε BFS, θα πρέπει για κάθε νέο κόμβο που συναντάτε, να ελέγχετε αν τον έχετε ξανασυναντήσει. Μία δομή ευρετηρίου(-συνόλου) μπορεί να δώσει εγγύηση χρόνου **O(logn)** για έλεγχο ύπαρξης στοιχείου, οπότε αυτό δεν μας κάνει. Για να πετύχετε γραμμικό χρόνο, θα πρέπει να έχετε διαθέσιμο έναν boolean πίνακα μεγέθους **n**, αρχικοποιημένο με "False", που σε κάθε θέση **i** θα έχει την τιμή "True" μόνο αν έχετε συναντήσει τον κόμβο **i**. Όμως, και αυτό από μόνο του δεν είναι αρκετό. Επειδή μετά από κάθε εισαγωγή ακμής θα τρέχουμε κατευθείαν έναν αλγόριθμο συνεκτικότητας, θα προτιμούσαμε να μην χρειάζεται να αρχικοποιούμε αυτόν τον boolean πίνακα ξανά απ' την αρχή. Οπότε, θα κάνουμε την εξής υλοποίηση. Προτού ξεκινήσει η όλη διαδικασία, αναγκαστικά θα αρχικοποιήσουμε αυτόν τον πίνακα με "False". Έπειτα, κάθε φορά που θα καλείται ο αλγόριθμος για έλεγχο συνεκτικότητας, θα διατηρούμε σε μία λίστα όλους τους κόμβους που συναντήσαμε, ούτως ώστε, όταν ο έλεγχος τερματιστεί, να ξαναθέσουμε σε "False" μονάχα εκείνες τις θέσεις του πίνακα που αντιστοιχούν σε κόμβους που εξερευνήθηκαν στην τρέχουσα κλήση. Δεν είναι απαραίτητο για τα επόμενα να έχετε μπει στην διαδικασία αυτής της υλοποίησης, αλλά αν το πετύχετε θα κερδίσετε:
(Μονάδες: 0.25)

Για να ελέγξετε την υλοποίησή σας, τρέξτε την διαδικασία που περιγράψαμε για **n=100**. Το πλήθος εισαγωγών ακμών μέχρι να καταλήξετε σε συνεκτικό γράφημα θα βρείτε να είναι συνήθως μεταξύ 200 και 500.

Τώρα, αν επιχειρήσουμε να τρέξουμε αυτήν την διαδικασία για αρκετά μεγαλύτερο **n**, π.χ. για **n=100.000**, τότε ο αλγόριθμος γραμμικού χρόνου για έλεγχο συνεκτικότητας δεν θα μπορέσει να μας πάει και πολύ μακριά (με την έννοια ότι, μόλις θα έχουν προστεθεί αρκετές ακμές, θα τρέχει πάρα πολύ αργά μετά από κάθε νέα εισαγωγή ακμής). (Προτείνουμε να το δοκιμάσετε λιγάκι αυτό.)

Όμως, ενδέχεται ακόμη και μετά από πολλές εισαγωγές ακμών, τα γραφήματα να έχουν τόσο μεγάλη απόσταση απ' το να είναι συνεκτικά (με την έννοια απόστασης απ' την συνεκτικότητα που είδαμε στο μάθημα), ώστε να συμφέρει να τρέξουμε πρώτα τον αλγόριθμο υπογραμμικού χρόνου που έχουμε δει.

Ο αλγόριθμος υπογραμμικού χρόνου που είδαμε, βασίζεται σε μία ρουτίνα τοπικού ελέγχου συνεκτικότητας. Συγκεκριμένα, αυτή η ρουτίνα δέχεται ως είσοδο το γράφημα (ή καλύτερα: έναν δείκτη στο γράφημα), που αναπαρίσταται με μορφή λιστών γειτνίασης, έναν αρχικό κόμβο (seed) **s**, και ένα budget **B** για εξερεύνηση κόμβων (όπου **B < n**).

Η ρουτίνα αυτή ξεκινά ένα BFS από το **s**, μέχρι να εξερευνήσει **B+1** κόμβους, ή μέχρι να καταφέρει να ολοκληρώσει την διερεύνηση, οπότε έχει εντοπίσει μία συνεκτική συνιστώσα που περιέχει το **s**. (Εδώ πρέπει να διευκρινήσουμε, ότι ως "πλήθος κόμβων που έχουν εξερευνηθεί", μετράμε τις νέες ανακαλύψεις κόμβων. Δηλαδή, αν ο ίδιος κόμβος συναντηθεί πάνω από μία φορά, θεωρούμε ότι συνεισφέρει μόνο μία στο "πλήθος κόμβων που έχουν εξερευνηθεί".) Άρα, αν καταφέρει να εξερευνήσει **B+1** κόμβους, τερματίζει την διερεύνηση και επιστρέφει "ΝΑΙ" (που σημαίνει ότι το γράφημα μπορεί να είναι συνεκτικό). Ειδάλλως, επιστρέφει "ΟΧΙ", που σημαίνει ότι το γράφημα είναι οπωσδήποτε μη-συνεκτικό. Εδώ δεχόμαστε υλοποιήσεις αυτής της ρουτίνας με χρονική πολυπλοκότητα το πολύ **O(BΔlogn)**.
(Μονάδες: 0.25)

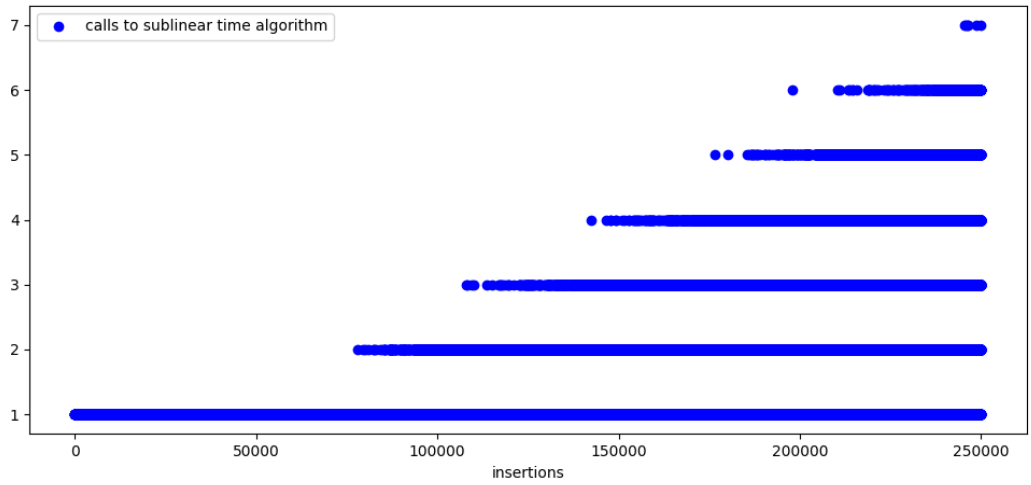
Τώρα, αφού διαθέτουμε την ρουτίνα τοπικού ελέγχου συνεκτικότητας, μπορούμε να υλοποιήσουμε τους αλγορίθμους υπογραμμικού χρόνου που είδαμε στο μάθημα για έλεγχο συνεκτικότητας. Υπενθυμίζουμε ότι αυτοί οι αλγόριθμοι παραμετροποιούνται με **ε** (και **Δ**), και έχουν χρονική πολυπλοκότητα **$O(1/(\varepsilon^2\Delta))$** και **$O((1/\varepsilon)\cdot\log(1/(\varepsilon\Delta))^2)$**, αντίστοιχα. Θα λέμε ότι ο πρώτος αλγόριθμος είναι ο "απλός", και ο δεύτερος ο "εκλεπτυσμένος".

Ο σκοπός μας είναι να υλοποιήσουμε αυτούς τους δύο αλγορίθμους, και να τους χρησιμοποιήσουμε για να (δούμε αν μπορούμε να) επιταχύνουμε τον έλεγχο συνεκτικότητας καθώς προστίθενται ακμές στο γράφημα.

Για αρχή, θα υλοποιήσουμε τον "απλό" αλγόριθμο υπογραμμικού χρόνου, και θα τον αξιοποιήσουμε ως εξής. Μετά από κάθε εισαγωγή ακμής, πρώτα θα τρέχουμε αυτόν τον αλγόριθμο για έλεγχο συνεκτικότητας, δοκιμάζοντας ολοένα και μικρότερα **ε**. Συγκεκριμένα, θα ξεκινούμε από **ε=1/Δ**, και θα το υποδιπλασιάζουμε, μέχρι να κρίνουμε ότι δεν συμφέρει να συνεχίσουμε τον υποδιπλασιασμό, και θα ήταν καλύτερα να τρέξουμε τον αλγόριθμο γραμμικού χρόνου. (Θα αφήσουμε να κρίνετε εσείς ποιο είναι ένα κατάλληλο **ε** απ' το οποίο και πέρα είναι καλύτερα να περάσουμε στον αλγόριθμο γραμμικού χρόνου.)

Πιο συγκεκριμένα, η διαδικασία θα είναι η εξής. Για κάθε **ε** (που κρίνουμε ότι συμφέρει να θεωρήσουμε), θα τρέχουμε τον αλγόριθμο υπογραμμικού χρόνου. Αν αυτός επιστρέψει "ΟΧΙ", τότε ξέρουμε ότι το γράφημα δεν είναι συνεκτικό, οπότε τερματίζουμε τον έλεγχο (και περνάμε σε νέα προσθήκη ακμής). Ειδάλλως, αν ο αλγόριθμος υπογραμμικού χρόνου πει "ΝΑΙ", τότε δεν μπορούμε να συναγάγουμε με βεβαιότητα την συνεκτικότητα, οπότε αναγκαστικά θα υποδιπλασιάσουμε το **ε** (δηλαδή θα θέσουμε **ε ← ε/2**), ή, αν κρίνουμε ότι το **ε** έγινε πολύ μικρό, θα εφαρμόσουμε τον αλγόριθμο γραμμικού χρόνου (που θα δώσει την σωστή απάντηση).

Θα εφαρμόσουμε αυτήν την διαδικασία για **n=100.000**, **Δ=10**, και για **250.000** εισαγωγές ακμών. (Θα παρατηρήσετε ότι τόσες εισαγωγές τυχαίων ακμών συνήθως δεν φτάνουν προκειμένου να έχουμε συνεκτικό γράφημα.) Μετά από κάθε εισαγωγή ακμής, ο έλεγχος συνεκτικότητας που περιγράψαμε θα καταγράφει το πλήθος των κλήσεων που έγιναν στον αλγόριθμο υπογραμμικού χρόνου (που είναι το πλήθος υποδιπλασιασμών του **ε**, +1). Θα πρέπει να σχηματίσετε και την αντίστοιχη γραφική παράσταση από αυτά τα δεδομένα, που θα μοιάζει κάπως έτσι:



Προφανώς, κάθε εκτέλεση θα δίνει διαφορετική γραφική παράσταση, αλλά σε γενικές γραμμές αναμένεται να λαμβάνετε ένα ξεκάθαρο μοτίβο.
(Μονάδες: 1)

Εδώ βλέπουμε ότι μέχρι και στις πρώτες περίπου 75.000 εισαγωγές ακμών, μόνο μία κλήση του αλγορίθμου υπογραμμικού χρόνου αρκεί για να εντοπίσει την μη-συνεκτικότητα. Όμως, καθώς οι εισαγωγές αυξάνονται, η απόσταση του γραφήματος απ' το να είναι συνεκτικό αναμένεται να μικραίνει, άρα γενικά απαιτούνται περισσότερες κλήσεις (δηλαδή να δοκιμαστούν μικρότερα **ε**), ώστε να εντοπίσουμε την μη-συνεκτικότητα. Όταν το πλήθος των εισαγωγών έχει φτάσει τις 250.000, βλέπουμε ότι και πάλι το πολύ 7 κλήσεις (δηλαδή 6 υποδιπλασιασμοί του **ε**) αρκούν. Βέβαια, όπως φαίνεται και στο σχήμα, καμιά φορά συνεχίζει να αρκεί μονάχα μία κλήση, όμως αυτό γίνεται όλο και πιο σπάνια (κάτι που δεν φαίνεται ξεκάθαρα, λόγω της πυκνότητας των δεδομένων).

Τώρα επαναλάβετε την ίδια διαδικασία, έχοντας αντικαταστήσει τον "απλό" αλγόριθμο υπογραμμικού χρόνου με τον "εκλεπτυσμένο". Εδώ θα πρέπει να έχετε την πολυτέλεια να δοκιμάσετε και 300.000 εισαγωγές ακμών (προτού εξαντληθεί η υπομονή σας). Θέλουμε και πάλι να σχεδιάσετε την αντίστοιχη γραφική παράσταση όπως και πριν. Επιπλέον, θέλουμε να δοκιμάσετε και τις παραμέτρους **n=1.000.000**, **Δ=100**, και 2.000.000 εισαγωγές ακμών. Θα ήταν εύκολο να διαπιστώσετε ότι 2.000.000 εισαγωγές ακμών (σχεδόν ποτέ) δεν επαρκούν προκειμένου να έχουμε ένα συνεκτικό γράφημα, αν μετά από κάθε εισαγωγή ακμής δεν εφαρμόζατε πρώτα τον αλγόριθμο υπογραμμικού χρόνου (αλλά εφαρμόζατε κατευθείαν τον αλγόριθμο γραμμικού χρόνου); Επίσης, βλέπετε να είναι προτιμότερος ο "απλός" ή ο "εκλεπτυσμένος" αλγόριθμος υπογραμμικού χρόνου; (Δεν εννοούμε ότι αυτά τα ερωτήματα θα έπρεπε να μπορείτε να τα απαντήσετε a priori. Απλώς κάντε μερικά πειράματα για να διαπιστώσετε τι συμβαίνει.)
(Μονάδες: 1)

Τέλος, μπορεί να υποψιάζεστε ότι υπάρχει και πολύ πιο γρήγορος τρόπος να πραγματοποιήσουμε τα πειράματά μας. Πράγματι, υπάρχει ένας αλγόριθμος συνολικά σχεδόν γραμμικού χρόνου (που δηλαδή μπορεί να επεξεργαστεί **m** εισαγωγές ακμών, μέχρι να φτάσουμε σε συνεκτικό γράφημα, σε χρόνο περίπου **O(m)**). Αν υλοποιήσετε έναν τέτοιον αλγόριθμο, και καταφέρετε να τρέξετε την διαδικασία για **n=1.000.000** και **Δ=100** θα εξασφαλίσετε:
(Μονάδες: 0.5)

Σας δίνουμε και μία υπόδειξη για αυτόν τον αλγόριθμο. Αρκεί να χρησιμοποιήσετε μία δομή εύρεσης-ένωσης, και να διατηρείτε το πλήθος των συνεκτικών συνιστωσών μετά από κάθε εισαγωγή ακμής. (Οπότε ξεκινάτε με **n** συνεκτικές συνιστώσες, και η διαδικασία τερματίζεται όταν έχετε φτάσει στο σημείο να έχετε μόνο μία.)

Επίσης, σας αναφέρουμε ότι για **n=1.000.000** και **Δ=100**, συνήθως αρκούν γύρω στις 6.000.000 με 8.000.000 εισαγωγές ακμών για να φτάσουμε σε συνεκτικό γράφημα.

Βέβαια, μπορεί να αναρωτιέται κανείς, για ποιον λόγο να υλοποιήσουμε τους αλγορίθμους υπογραμμικού χρόνου, αφού τελικά υπάρχει και πολύ πιο γρήγορη μέθοδος για την όλη διαδικασία; Η απάντηση είναι (πέραν του ότι έτσι ελέγχουμε πειραματικά την επίδοση ορισμένων αλγορίθμων) ότι θα μπορούσαμε να είχαμε συμπεριλάβει στο ρεπερτόριο των αλλαγών στο γράφημα και διαγραφές ακμών (π.χ., με πιθανότητα 10%, να επιχειρούμε να διαγράψουμε ακμή, αντί να προσθέσουμε), οπότε θα χρειαζόταν να υλοποιήσει κανείς πολύ εξεζητημένους αλγορίθμους για διατήρηση του πλήθους των συνεκτικών συνιστωσών σε δυναμικά γραφήματα που υφίστανται ανάμικτα προσθαφαιρέσεις ακμών. Εδώ η γρήγορη μέθοδος (που βασίζεται σε μια δομή εύρεσης-ένωσης) δουλεύει ακριβώς επειδή περιοριζόμαστε (χάριν απλότητας) μονάχα σε εισαγωγές ακμών.

Ενότητα Β: Έλεγχος μονοτονικότητας πίνακα **(Μονάδες: 2)**

Άσκηση 1: Δειγματοληψία χωρίς επανάθεση*

Έστω ότι θέλουμε να κάνουμε δειγματοληψία **k** αριθμών από το σύνολο **{1 ,2, . . ., n}**, με ομοιόμορφη κατανομή πιθανότητας.

Τότε, υπάρχουν δύο διαφορετικές έννοιες αυτής της δειγματοληψίας: με επανάθεση, και χωρίς.

Η δειγματοληψία χωρίς επανάθεση είναι πολύ απλή: επαναλαμβάνουμε **k** φορές την διαδικασία του να επιλέγουμε τυχαία έναν αριθμό από το σύνολο **{1, 2, . . ., n}**. Δηλαδή, σε κάθε επανάληψη, κάθε αριθμός έχει πιθανότητα **1/n** να επιλεγεί. Αυτή η μέθοδος λέγεται "δειγματοληψία με επανάθεση" επειδή είναι σαν, κάθε φορά που επιλέγουμε έναν αριθμό, να τον τοποθετούμε ξανά στο σύνολο για δειγματοληψία. Αυτό έχει ως συνέπεια ότι μπορεί στο τέλος να έχουμε πάρει λιγότερα από **k** διακεκριμένα στοιχεία.

Στην δειγματοληψία χωρίς επανάθεση, ο σκοπός είναι να επιλεγεί τυχαία ένα υποσύνολο **k** στοιχείων από το σύνολο **{1, 2, . . ., n}**. Άρα, κάθε δεδομένο σύνολο **k** στοιχείων έχει πιθανότητα **1 / (n k)** να επιλεγεί, όπου με **(n k)** συμβολίζουμε την ποσότητα **n! / (k! · (n-k)!)**.

Ένας τρόπος να επιτευχθεί κάτι τέτοιο είναι ο εξής (που εφαρμόσαμε ουσιαστικά και στην άσκηση 1, της ενότητας Α, του 2ου φυλλαδίου):
1) Αρχικοποιούμε έναν πίνακα **{1, 2, . . ., n}**.
2) Διατρέχουμε με την σειρά τις πρώτες **k** θέσεις του πίνακα. Για κάθε θέση **i**, επιλέγουμε τυχαία έναν δείκτη **j** από **i** μέχρι και **n**, κρατούμε (για να επιστρέψουμε, ως ένα στοιχείο της δειγματοληψίας) τον αριθμό στην θέση **j**, και πραγματοποιούμε ανταλλαγή στοιχείων στις θέσεις **i**, **j**.

Ο αλγόριθμος που περιγράψαμε επιτυγχάνει αυτό που θέλουμε, αλλά έχει ένα μειονέκτημα: απαιτεί να έχουμε κατασκευάσει έναν πίνακα με **n** αριθμούς, και αυτό σε κάποιες εφαρμογές μπορεί να μην συμφέρει.

Υπάρχει όμως και ένας εναλλακτικός αλγόριθμος για δειγματοληψία **k** αριθμών χωρίς επανάθεση. Ο αλγόριθμος είναι ο εξής:

Αλγόριθμος δειγματοληψίας χωρίς επανάθεση
Αρχικοποιούμε ένα σύνολο **S** (το δείγμα που θα επιστρέψουμε) ως **S ← ∅**.
Έπειτα, επαναλαμβάνουμε **k** φορές την ακόλουθη διαδικασία:
1. Επιλέγουμε τυχαία έναν αριθμό **x** από **1** μέχρι και **n**.
2. Αν **x ∈ S**, τότε επαναλαμβάνουμε το βήμα 1.
3. Ειδάλλως, εισάγουμε το **x** στο **S**.

*Προσοχή: το ζητούμενο σε αυτήν την άσκηση είναι καθαρά πιθανοθεωρητικής φύσεως. Παρά ταύτα, σας παροτρύνουμε να μελετήσετε έστω την εκφώνηση, προκειμένου να καταλάβετε ποιο είναι το πρόβλημα, διότι στην επόμενη άσκηση θα χρειαστεί να υλοποιήσετε αλγόριθμο δειγματοληψίας χωρίς επανάθεση.

Τώρα, το προφανές μειονέκτημα του αλγορίθμου που περιγράψαμε, είναι ότι μπορεί να τρέξει υπερβολικά πολλές φορές τον βρόχο που σχηματίζουν τα βήματα 1 και 2, σε κάποια επανάληψη. Αυτό είναι πολύ πιθανό να συμβεί αν το **k** είναι κοντά στο **n**. Παρά ταύτα, αν το **k** είναι μικρό σε σχέση με το **n**, τότε αυτός ο αλγόριθμος αναμένεται να τερματίσει πολύ γρήγορα (ίσως και ακριβώς σε **k** επαναλήψεις).

Αποδείξτε ότι αυτός ο αλγόριθμος πραγματοποιεί όντως δειγματοληψία χωρίς επανάθεση. ***(Μονάδες: 0.25)***

Αν και προτείνουμε να σκεφτείτε μόνοι σας αυτό το πρόβλημα, στο τέλος του φυλλαδίου παρέχουμε μία υπόδειξη για αυτήν την άσκηση.

Άσκηση 2: Ένας ανταγωνιστικός πίνακας εισόδου

Έστω ένας πίνακας ακεραίων $\mathbf{A = [2, 1, 4, 3, 6, 5, \ldots, 10^{100}, 10^{100} - 1]}$, με $\mathbf{10^{100}}$ στοιχεία.

Παρατηρήστε ότι αυτός ο πίνακας έχει απόσταση **1/2** απ' το να είναι ταξινομημένος σε αύξουσα σειρά (με την έννοια απόστασης από την μονοτονικότητα που είδαμε στο μάθημα).

Θέλουμε να κάνουμε μία δειγματοληψία **N** θέσεων αυτού του πίνακα, ώστε να αποκαλύψουμε την μη-μονοτονικότητά του (σε αύξουσα σειρά). Συγκεκριμένα, προτείνουμε τον εξής αλγόριθμο:

1) Κάνουμε δειγματοληψία **N** αριθμών από το **1** μέχρι και το $\mathbf{10^{100}}$, χωρίς επανάθεση. (Δείτε την προηγούμενη άσκηση για το πώς κάνουμε τέτοια δειγματοληψία[†].)
2) Έστω $\mathbf{i_1, i_2, \ldots, i_N}$ οι αριθμοί που λάβαμε.
3) Τότε, ταξινομούμε τους αριθμούς αυτούς σε αύξουσα σειρά.
4) Δίχως βλάβη της γενικότητας, ας υποθέσουμε ότι $\mathbf{i_1 < i_2 < \ldots < i_N}$.
5) Τότε, ελέγχουμε αν $\mathbf{A[i_1] < A[i_2] < \ldots < A[i_N]}$.

Υλοποιήστε αυτόν τον αλγόριθμο, δοκιμάζοντας διάφορα **N** (οσοδήποτε μεγάλα θέλετε). ***(Μονάδες: 0.25)***

Αναμένετε να καταφέρετε να αποκαλύψετε την μη-μονοτονικότητα του πίνακα; (Αν όχι, γιατί;) ***(Μονάδες: 0.125)***

Εικάζουμε ότι μάλλον θα χρειαστεί αρκετή ώρα για να κατασκευάσετε τον πίνακα **A** (ανάλογα με τις δυνατότητες του υπολογιστή σας, φυσικά). Οπότε, προτείνουμε να μην επιχειρήσετε κάτι τέτοιο. Αυτό δεν είναι πρόβλημα, διότι σκεφτείτε πως αρκεί να

---

† Σημαντική σημείωση: όπως εξηγήσαμε και στην προηγούμενη άσκηση, υπάρχει το ενδεχόμενο ο αλγόριθμος δειγματοληψίας χωρίς επανάθεση να πάρει πολύ χρόνο να τρέξει, διότι ενδέχεται να επαναλάβει πολλές φορές έναν συγκεκριμένο βρόχο. Παρά ταύτα, εδώ δεν αναμένεται να συμβεί κάτι τέτοιο. Μάλιστα, αν τύχει έστω και μία φορά, κατά την διάρκεια του αλγορίθμου δειγματοληψίας χωρίς επανάθεση, να τύχετε έναν αριθμό που είδατε και πριν, τότε, παρότι γενικά σας συμβουλεύουμε να αποφεύγετε τον τζόγο, ίσως θα ήταν καλή ιδέα σήμερα να παίξετε και ένα τζόκερ.

μπορείτε να επιστρέψετε μία θέση του πίνακα μονάχα όταν σας ζητηθεί. Και παρατηρήστε ότι οι εγγραφές του πίνακα δίνονται από τον εξής τύπο:
A[i] = i+1, αν **i mod 2 = 1**,
A[i] = i-1, αν **i mod 2 = 0**.

Άσκηση 3: Υλοποίηση μιας αποδοτικής μεθόδου ελέγχου μονοτονικότητας

Σε αυτήν την άσκηση θα ασχοληθούμε με τέσσερις διαφορετικούς πίνακες ακεραίων με 10^{100} θέσεις (αριθμημένες από το **1** μέχρι και το 10^{100}). (Όπως και στην προηγούμενη άσκηση, προτείνουμε να μην μπείτε στην διαδικασία να κατασκευάσετε αυτούς τους πίνακες.)

Ο πίνακας **A** είναι αυτός που είδαμε στην προηγούμενη άσκηση.

Ο πίνακας **B** δίνεται από τις σχέσεις:
B[i] = 0, αν **i mod 100 = 20**,
B[i] = ⌊i/100⌋+1, αν **i mod 100 ≠ 20**.

Ο πίνακας **C** δίνεται από τις σχέσεις:
C[i] = ⌊i/100⌋, αν **i mod 100 = 20**,
C[i] = ⌊i/100⌋+1, αν **i mod 100 ≠ 20**.

Ο πίνακας **D** είναι στοχαστικός, και εξαρτάται από μία παράμετρο **p**, με **0 ≤ p ≤ 1**, που δίνεται ως είσοδος. Δεδομένου του **p**, ο πίνακας **D** δίνεται από τις σχέσεις:
D[i] = ⌊i/100⌋, αν **i mod 100 = 20**, με πιθανότητα **p** για το **i**,
D[i] = ⌊i/100⌋+1, αν **i mod 100 ≠ 20**, ή αν **i mod 100 = 20** και η πιθανότητα **p** δεν πραγματοποιήθηκε για το **i**.

Δηλαδή, για κάθε **i** για το οποίο ισχύει **i mod 100 = 20**, με πιθανότητα **p** θα έχουμε **D[i] = ⌊i/100⌋**, και με πιθανότητα **1 - p** θα έχουμε **D[i] = ⌊i/100⌋+1**. (Παρατηρήστε ότι για **p=1**, ο **D** ταυτίζεται με τον **C**.)

Οπότε, μπορεί να έχουμε **D[20]=1**, **D[120]=2**, **D[220]=3**, **D[320]=3**, **D[420]=5**, κ.ο.κ.

Υπολογίστε την απόσταση των πινάκων **B**, **C**, **D** απ' το να είναι ταξινομημένοι (σε αύξουσα σειρά). Για τον πίνακα **D**, εφόσον αυτός είναι στοχαστικός, και η στοχαστικότητά του εξαρτάται από μία παράμετρο **p**, αρκεί να δώσετε μία καλή εκτίμηση της απόστασής του απ' το να είναι ταξινομημένος, συναρτήσει του **p**.
(Μονάδες: 0.125)

Ο φυσικός αλγόριθμος ελέγχου μονοτονικότητας που είδαμε στην προηγούμενη άσκηση, θεωρείτε ότι θα είχε καλύτερη επίδοση στον πίνακα **B** ή τον **C** (ή και στους δύο θα είχε την ίδια επίδοση); (Όταν λέμε επίδοση, εννοούμε πόσο μεγάλο πρέπει να γίνει το **N** ώστε να έχουμε καλή πιθανότητα να αποκαλύψουμε την μη-μονοτονικότητα.) Αν για κάποιον από τους δύο πίνακες έχετε την υποψία ότι κάποιο εύλογα μικρό **N** θα μπορούσε να δουλέψει, τότε επιλέξτε ένα κατάλληλο **N** (για τον

συγκεκριμένο πίνακα), ώστε να έχουμε πιθανότητα επιτυχίας τουλάχιστον **99%**, και υλοποιήστε τον αλγόριθμό σας. ***(Μονάδες: 0.25)***

Τέλος, θα υλοποιήσουμε τον αποδοτικό αλγόριθμο για έλεγχο μονοτονικότητας που είδαμε στο μάθημα, ο οποίος βασίζεται σε δυαδική αναζήτηση. Εφαρμόστε αυτόν τον αλγόριθμο στους πίνακες **A**, **B**, **C**. Υπενθυμίζουμε ότι αυτός ο αλγόριθμος παραμετροποιείται με ένα **ε** (που αποτελεί μία εκτίμηση ή εικασία για την απόσταση του πίνακα απ' το να είναι ταξινομημένος). Προτείνουμε να επιλέξετε κατάλληλα το **ε** με βάση την απόσταση που βρήκατε ότι έχουν αυτοί οι πίνακες απ' το να είναι ταξινομημένοι. (Ειδάλλως, δώστε για **ε** ό,τι θεωρείτε ότι θα ήταν μία καλή εκτίμηση.) Μας αρκεί να έχουμε πιθανότητα επιτυχίας τουλάχιστον **75%**. Φροντίστε να τρέξετε αρκετές φορές το πρόγραμμά σας, ώστε να σιγουρευτείτε ότι παίρνετε εύλογα αποτελέσματα.
(Μονάδες: 0.5)

Φυσικά, μπορεί να αναρωτιέστε "γιατί να μην κάνουμε το ίδιο και για τον πίνακα **D**"; Βεβαίως, αυτή θα είναι η τελευταία πρόκληση αυτής της άσκησης. Όμως, ο συγκεκριμένος πίνακας έχει την ιδιαιτερότητα ότι δεν είναι όλες του οι εγγραφές προκαθορισμένες, αλλά κάποιες καθορίζονται από μια πηγή τυχαιότητας. Σε μια πραγματική εφαρμογή, αυτό μάλλον δεν θα ήταν πρόβλημα (αφού ο πίνακας θα ήταν ήδη αποθηκευμένος κάπου), όμως αυτόν τον πίνακα θα πρέπει να τον κατασκευάσουμε εμείς, και αυτό προτιμάμε να το αποφύγουμε.

Παρά ταύτα, εφόσον οι θέσεις του πίνακα που δεν είναι προκαθορισμένες εξ αρχής καθορίζονται η μία ανεξάρτητα από την άλλη (με μία πιθανοτική παράμετρο **p**), μπορούμε να προσομοιώσουμε την διαδικασία του να τρέχαμε τον αλγόριθμο πάνω σε πραγματικά δεδομένα, αρκεί να κατασκευάζουμε "on the fly" τις εγγραφές στις οποίες θα απαιτήσει πρόσβαση ο αλγόριθμός μας. Αυτό όμως χρειάζεται λίγη προσοχή: αν ο αλγόριθμός μας, κατά την διάρκεια της εκτέλεσής του, απαιτήσει πρόσβαση σε μία θέση **i** με **i mod 100 = 20** (δηλαδή σε μία θέση που καθορίζεται από την πιθανοτική παράμετρο), τότε, αν έχουμε ξαναπαιτήσει πρόσβαση σε αυτήν την θέση, θα πρέπει να θεωρήσουμε ότι έχει την ίδια τιμή που είχε και πριν. (Ενώ, αν συναντούμε το **i** για πρώτη φορά, αρκεί να καθορίσουμε εμείς την τιμή του, με βάση την πιθανοτική παράμετρο **p**.)

Εδώ ας δοκιμάσουμε την προσομοίωση με τιμές **p = 50%, 25%, 10%, 1%, 0.1%**, και θα επιλέγουμε το **ε** κατάλληλα, με βάση την αναμενόμενη απόσταση του **D** απ' το να είναι ταξινομημένος (ή έστω με μια καλή εκτίμηση για αυτήν την απόσταση). Προτείνουμε αυτές τις τιμές για δικό σας πειραματισμό. Το μόνο που απαιτούμε για να ελέγξουμε την λύση σας, είναι να έχετε αφήσει κάπου ξεκάθαρα την επιλογή για το **p** στον κώδικά σας.
(Μονάδες: 0.5)

Ενότητα Γ: Έλεγχος διμερότητας γραφήματος **(Μονάδες: 2)**

Έστω ένας θετικός ακέραιος **n**, και ένας πραγματικός αριθμός **p**, με **0 ≤ p ≤ 1**.

Θα λέμε ότι ένα γράφημα είναι τυχαίο με παραμέτρους **n** και **p**, αν αποτελείται από **n** κόμβους, και κάθε δυνατή ακμή μεταξύ των **n** κόμβων υπάρχει με πιθανότητα **p**.

Με άλλα λόγια, μπορούμε να περιγράψουμε την τυχαία επιλογή ενός τέτοιου γραφήματος ως εξής. Για αρχή, θεωρούμε έναν πίνακα μεγέθους **n × n**, όπου όλες του οι θέσεις είναι μηδενισμένες, και που πρόκειται να αποτελέσει τον πίνακα γειτνίασης του γραφήματος. (Υποθέτουμε ότι οι κόμβοι του γραφήματος είναι αριθμημένοι από **1** μέχρι και **n**.) Έπειτα, για κάθε πιθανό ζευγάρι κόμβων **(x,y)**, με **x < y**, επιλέγουμε με πιθανότητα **p** αν θα θέσουμε την τιμή **1** στις θέσεις **(x,y)** και **(y,x)** του πίνακα γειτνίασης.

Σε αυτήν την ενότητα, θα μας απασχολήσει το εξής ερώτημα: πόσο πιθανό είναι ένα τυχαίο γράφημα με παραμέτρους **n** και **p** να είναι διμερές;

Η διαίσθησή μας είναι, ότι αν το **p** είναι αρκετά μεγάλο, τότε το γράφημα θα είναι αρκετά πυκνό, και άρα πολύ απίθανο να είναι διμερές. Όμως, αν το **p** μικρύνει αρκετά, τότε το γράφημα θα είναι αρκετά αραιό – και συγκεκριμένα, θα τείνει να μην περιέχει καν κύκλους –, οπότε με πολύ μεγάλη πιθανότητα θα είναι διμερές.

Για να εξετάσουμε αυτήν την υπόθεση, θα εφαρμόσουμε την μέθοδο Monte Carlo για εκτίμηση της πιθανότητας ένα τυχαίο γράφημα με παραμέτρους **n** και **p** να είναι διμερές. Συγκεκριμένα, θα δοκιμάσουμε τις τιμές **n=100**, **1000**, **10000**, και το **p** να ξεκινάει από **1** και να μειώνεται εκθετικά (π.χ., να υποδιπλασιάζεται), μέχρι να γίνει μικρότερο από μία συγκεκριμένη τιμή (π.χ., 0.1%). Τότε, για κάθε σταθεροποιημένο **n**, και κάθε προεπιλεγμένη πιθανότητα **p**, θα παράγουμε 100 τυχαία γραφήματα με παραμέτρους **n** και **p**, θα τα εξετάζουμε ένα-ένα ως προς την διμερότητα, και θα επιστρέφουμε το ποσοστό αυτών που είναι διμερή.

Άσκηση 1: Υλοποίηση του αλγορίθμου γραμμικού χρόνου

Για αρχή, θα χρειαστεί να υλοποιήσουμε τον αλγόριθμο γραμμικού χρόνου για τον έλεγχο διμερότητας γραφήματος. Για να ελέγξουμε την διμερότητα ενός γραφήματος, υπολογίζουμε τις συνεκτικές του συνιστώσες, και για κάθε μία ελέγχουμε αν είναι διμερής (εφαρμόζοντας τον αλγόριθμο που είδαμε στο μάθημα για έλεγχο διμερότητας σε συνεκτικά γραφήματα). Αν υπάρχει έστω και μία που δεν είναι διμερής, τότε το γράφημα δεν είναι διμερές. Ειδάλλως, είναι διμερές.

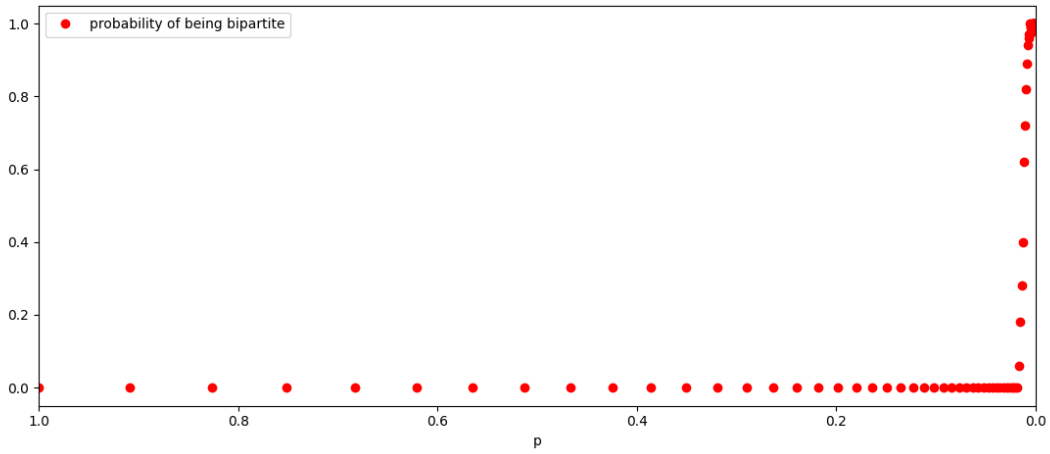
Αυτό το κομμάτι είναι απαραίτητο και για την επόμενη άσκηση, αφού ο αλγόριθμος υπογραμμικού χρόνου που θα αξιοποιήσουμε εκεί προϋποθέτει ότι έχουμε μία ρουτίνα για έλεγχο διμερότητας σε γραμμικό χρόνο.

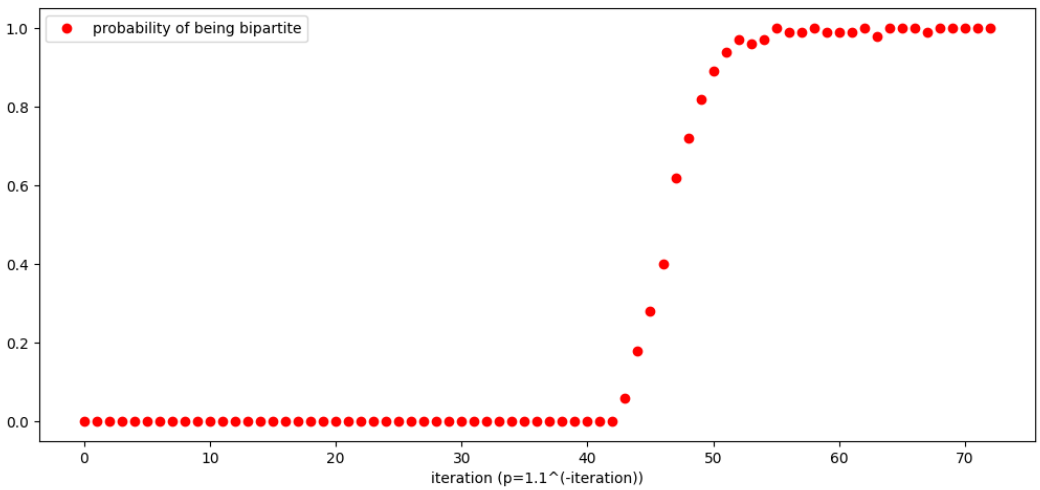
Τις λεπτομέρειες της συγκεκριμένης υλοποίησης τις αφήνουμε σε εσάς. Ειδικότερα, το αν το γράφημα εισόδου σε αυτήν την ρουτίνα θα το αναπαριστάτε με πίνακα

γειτνίασης ή με λίστες γειτνίασης, αφήνουμε να το επιλέξετε εσείς, με βάση το τι σας φαίνεται ευκολότερο Σε κάθε περίπτωση, όμως, θέλουμε η υλοποίηση να είναι γραμμικού χρόνου. Επιπλέον, ίσως σας βολέψει να αριθμείτε τους κόμβους από το **0** μέχρι και το **n – 1** (αυτό δεν αλλάζει ουσιαστικά τίποτα).

Μόλις ολοκληρώσετε την υλοποίησή σας, δοκιμάστε την μέθοδο Monte Carlo που περιγράψαμε σε τυχαία γραφήματα με **100** κόμβους, όπου η πιθανότητα **p** ξεκινάει από το **1**, και κάθε φορά γίνεται το **1/1.1** της προηγούμενης, για όσο έχουμε **p > 0.1%**. (Άρα, θα πρέπει να λάβετε αποτελέσματα για **73** διαφορετικά **p**, όπου για κάθε ένα θα κάνετε 100 επαναλήψεις, και θα επιστρέφετε το ποσοστό των γραφημάτων που είναι διμερή.)

Με τα αποτελέσματά σας, φτιάξτε γραφικές παραστάσεις όπως τις εξής:





Και στις δύο γραφικές παραστάσεις έχουμε **73** διαφορετικά σημεία, όπου ο άξονας των **y** δείχνει την πιθανότητα που εκτιμήσαμε για να έχουμε διμερότητα. Στην πρώτη γραφική παράσταση, στον άξονα των **x** έχουμε τις διαφορετικές τιμές του **p**: **1**, **1/1.1**, **1/1.1²**, **1/1.1³**, **. . ..** Στην δεύτερη γραφική παράσταση, ο άξονας των **x** είναι αριθμημένος ως **0**, **1**, **2**, **3**, **. . .**, και αντιστοιχεί στις διαφορετικές επιλογές του **p**.

Παρατηρούμε αυτό ακριβώς που υποψιαζόμασταν: όσο το **p** είναι αρκετά μεγάλο (μεγαλύτερο από **1%**), ένα τυχαίο γράφημα με παραμέτρους **100** και **p** είναι σχεδόν απίθανο να είναι διμερές. Όμως, καθώς το **p** μικραίνει, υπάρχει μία μικρή περιοχή απότομης αύξησης της πιθανότητας το γράφημα να είναι διμερές, μέχρι που πολύ απότομα γίνεται σχεδόν βέβαιο ότι το γράφημα θα είναι διμερές.
(Μονάδες: 0.5)

Από καθαρή περιέργεια, δοκιμάστε να δείτε αν θα σας ήταν εύκολο να κάνετε τις ίδιες γραφικές παραστάσεις και για **n=1000**, όταν το **p** ξεκινάει από **1**, και υποδιπλασιάζεται, μέχρι να γίνει μικρότερο από **0.1%**.

Ίσως παρατηρείτε ότι εδώ μας παίρνει πολύ περισσότερο χρόνο για να βγάλουμε τα ανάλογα αποτελέσματα. Μπορείτε να σκεφτείτε γιατί; (Εδώ δεν είναι απαραίτητο να κάνετε τα πειράματα για **n=1000**. Ο σκοπός είναι απλώς να διαπιστώσετε την δυσκολία.)

Άσκηση 2: Αξιοποίηση του αλγορίθμου υπογραμμικού χρόνου

Με βάση τα αποτελέσματα που είδαμε στην προηγούμενη άσκηση, εικάζουμε ότι, όταν το **p** είναι αρκετά μεγάλο, τότε, όχι απλώς τα γραφήματα είναι πρακτικά αδύνατο να είναι διμερή, αλλά θα έχουν και μεγάλη απόσταση από την διμερότητα (με την έννοια απόστασης από την διμερότητα που είδαμε στο μάθημα).

Αν αυτό ισχύει, τότε, όταν το **p** είναι αρκετά μεγάλο, ο αλγόριθμος υπογραμμικού χρόνου για τον έλεγχο διμερότητας που έχουμε δει, θα πρέπει, από ένα σχετικά μικρό δείγμα, να είναι σε θέση να συμπεράνει ότι το γράφημα δεν είναι διμερές.

Όπως θυμόμαστε, το μέγεθος του δείγματος που θα πάρει ο αλγόριθμος εξαρτάται από μία παράμετρο **ε**, η οποία συνιστά μία εκτίμηση ή εικασία για την απόσταση του γραφήματος απ' το να είναι διμερές. Εμείς θα ξεκινούμε με **ε=1/2**, και θα το υποδιπλασιάζουμε, μέχρι να φτάσουμε σε ένα σημείο όπου είναι ασύμφορο να κάνουμε δειγματοληψία με τόσο μικρό **ε**, και άρα αναγκαστικά θα εφαρμόσουμε τον αλγόριθμο γραμμικού χρόνου σε ολόκληρο το γράφημα. (Θα το αφήσουμε στην δική σας κρίση να επιλέξετε ένα κατάλληλο τέτοιο σημείο.)

Συγκεκριμένα, σε αυτήν την άσκηση θα δουλέψουμε με **n=1000** και **p ∈ {1, 1/2, 1/4, 1/8, . . . }**, για όσο ισχύει ότι **p > 0.09%**. (Άρα, θα έχουμε **11** διαφορετικά **p**.) Για κάθε **p**, θέλουμε να φτιάξουμε 100 τυχαία γραφήματα με παραμέτρους **n** και **p**, και να εφαρμόσουμε τον εξής αλγόριθμο για έλεγχο διμερότητας:

Για **ε ∈ {1/2, 1/4, 1/8, . . }**, μέχρι να κρίνουμε ότι δεν έχει νόημα να συνεχίσουμε να μικραίνουμε το **ε** περισσότερο, εφαρμόζουμε τον αλγόριθμο υπογραμμικού χρόνου με παράμετρο **ε**. Αν έστω μία φορά λάβουμε "ΟΧΙ" (δηλαδή ότι το γράφημα δεν είναι διμερές), τότε τερματίζουμε τον αλγόριθμο για το συγκεκριμένο γράφημα, διότι αυτό συνεπάγεται με σιγουριά ότι το γράφημα δεν είναι διμερές. Αν όμως για όλα τα **ε** (για τα οποία κρίναμε ότι έχει νόημα να προσπαθήσουμε) ο αλγόριθμος υπογραμμικού χρόνου πει "ΝΑΙ", τότε δεν μπορούμε να συμπεράνουμε με σιγουριά ότι το γράφημα

είναι διμερές, οπότε αναγκαστικά θα τρέξουμε τον αλγόριθμο γραμμικού χρόνου σε ολόκληρο το γράφημα.

Οπότε, ο σκοπός είναι και πάλι να εκτιμήσουμε με την μέθοδο Monte Carlo την πιθανότητα ένα τυχαίο γράφημα με παραμέτρους **n** και **p** να είναι διμερές. Απλώς η ιδέα εδώ είναι ότι, δοκιμάζοντας πρώτα τον αλγόριθμο υπογραμμικού χρόνου, θα καταφέρουμε, για τα περισσότερα (αρκετά μεγάλα) **p**, να εντοπίσουμε πολύ γρήγορα ότι ένα γράφημα δεν είναι διμερές (για τα περισσότερα γραφήματα που δεν είναι διμερή).

Όμως, για να δουλέψει σωστά αυτή η ιδέα, θα πρέπει να είμαστε λιγάκι προσεκτικοί. Παρατηρήστε ότι δεν θα είχε νόημα να κατασκευάζουμε ολόκληρο το γράφημα, για κάθε μια από τις 100 επαναλήψεις, για κάθε **p**, και μετά να εφαρμόζουμε τον αλγόριθμο υπογραμμικού χρόνου, διότι τότε δεν θα κερδίζαμε ουσιαστικά τίποτα, αφού ήδη θα μας έπαιρνε γραμμικό χρόνο για να κατασκευάσουμε το γράφημα (αποφασίζοντας, με πιθανότητα **p**, για κάθε μια από τις **n(n-1)/2** πιθανές ακμές του γραφήματος, αν θα την συμπεριλάβουμε στο γράφημα).

Οπότε, η ιδέα είναι να αποφασίζουμε αν θα συμπεριλάβουμε μία ακμή μονάχα αν χρειαστεί να την εξετάσουμε. Για την ακρίβεια, ας υποθέσουμε π.χ. ότι ξεκινάμε με **ε=1/2**. Τότε, κάνουμε δειγματοληψία ορισμένου πλήθους κόμβων (που εξαρτάται από το **ε**), και σχηματίζουμε το υπογράφημα που επάγεται από αυτούς τους κόμβους. Εφόσον το αν θα υπάρχει μια ακμή καθορίζεται ανεξάρτητα από τις υπόλοιπες, σε αυτήν την φάση μπορούμε να κατασκευάσουμε "on the fly" τις ακμές του γραφήματος που επάγονται από το δείγμα. Όμως, η απόφαση για το αν μια ακμή θα συμπεριληφθεί ή όχι θα πρέπει να είναι δεσμευτική για όσο τρέξει ο αλγόριθμος πάνω σε αυτό το γράφημα (που ουσιαστικά το κατασκευάζουμε "on the fly"). Δηλαδή, αν χρειαστεί να περάσουμε στο **ε=1/4**, τότε, αν τύχει στο δείγμα κόμβων που κάναμε δειγματοληψία να εμφανιστούν δύο κόμβοι **x,y** που εμφανίστηκαν και πριν, θα πρέπει να έχουμε αποθηκεύσει αναγκαστικά το αν η ακμή **(x,y)** υπάρχει στο γράφημα, και άρα τώρα δεν επιτρέπεται να το επανακαθορίσουμε αυτό. Στο τέλος, αν όλες οι εκτελέσεις του αλγορίθμου υπογραμμικού χρόνου κατέληξαν σε αποτυχία (δηλαδή, αν όλες τους είπαν "ΝΑΙ"), θα πρέπει να συμπληρώσουμε τις υπόλοιπες ακμές που απομένει να εξετάσουμε στο γράφημα (αν έχουν απομείνει τέτοιες), ώστε να τις καθορίσουμε και αυτές, για να τρέξουμε τελικά τον αλγόριθμο γραμμικού χρόνου.

Για να γίνει λοιπόν αυτό το πείραμα αποδοτικά, προτείνουμε στην αρχή να αρχικοποιήσουμε τον πίνακα γειτνίασης με **-1** σε όλες τις **n × n** θέσεις του. Αυτό το "**-1**" έχει την έννοια ότι "ακόμη δεν καθορίσαμε το αν θα υπάρχει αυτή η ακμή ή όχι". (Αυτή η αρχικοποίηση γίνεται μονάχα μία φορά στην αρχή, προτού ξεκινήσουμε το όλο πείραμα.) Οπότε, κάθε φορά που τρέχει ο αλγόριθμος, θα καθορίζει τις θέσεις στις οποίες αποκτά πρόσβαση και ακόμη έχουν **-1**, αλλά δεν θα πειράζει τις υπόλοιπες που έχει ήδη καθορίσει (με **0** ή **1**). Κατά την διάρκειά του, θα πρέπει να αποθηκεύουμε σε μία λίστα όλες τις θέσεις που καθορίσαμε, ώστε στο τέλος να τις μετατρέψουμε σε **-1** ξανά, ώστε να ξεκινήσει η επόμενη εκτέλεση (που αφορά ένα νέο τυχαίο γράφημα).

Έτσι αναμένουμε να προσομοιώσουμε τα Monte Carlo πειράματα πολύ αποδοτικά. Μονάχα όταν το **p** γίνει αρκετά μικρό θα πάψουμε να έχουμε πλεονέκτημα, διότι ο

αλγόριθμος θα καταλήγει αναγκαστικά να τρέξει τον αλγόριθμο γραμμικού χρόνου (γιατί τα περισσότερα γραφήματα θα είναι διμερή). Όμως, εφόσον υποδιπλασιάζουμε το **p**, και συνεχίζουμε για όσο **p > 0.09%**, τα περισσότερα πειράματα θα εκτελεστούν σχετικά γρήγορα.

Ο σκοπός είναι να κάνουμε τις αντίστοιχες γραφικές παραστάσεις για **n=1000** με αυτές που κάναμε στην προηγούμενη άσκηση (που αφορούσαν το **n=100**, και διαφορετικές τιμές του **p**). Για να επαληθεύσετε ότι τα αποτελέσματά σας είναι εύλογα, εφαρμόστε πρώτα αυτήν την ιδέα για **n=100** (και με τα ίδια **p** με πριν), διότι θα πρέπει το πρόγραμμά σας για την περίπτωση **n=100** να τρέχει σχεδόν εξ ίσου γρήγορα με πριν.
(Μονάδες: 1.25)

Τέλος, δοκιμάστε μερικές τιμές του **p** και για **n=10000** (προτείνουμε **p∈{1, 1/2, 1/4, 1/8}**), απλώς για να δείτε ότι, για αυτά τα αρκετά μεγάλα **p**, μπορούμε και πάλι να διαπιστώσουμε αποδοτικά την πρακτικά μηδενική πιθανότητα να έχουμε διμερές γράφημα. (Δεν χρειάζεται να φτιάξετε γραφική παράσταση για την περίπτωση **n=10000**, αφού δοκιμάζουμε πολύ λίγες τιμές του **p**, απλώς ενδεικτικά.) Βλέπετε να έχουμε σημαντικό κέρδος στον χρόνο απ' ό,τι αν εφαρμόζαμε σε κάθε τυχαίο γράφημα κατευθείαν τον αλγόριθμο γραμμικού χρόνου; Τέλος, σημειώνουμε ότι υπάρχει το ενδεχόμενο εδώ ο αλγόριθμός μας (που πρώτα δοκιμάζει τον αλγόριθμο υπογραμμικού χρόνου για **ε=1/2**, **1/4**, **1/8**, . . .) να τρέχει αισθητά πιο αργά απ' ό,τι για την περίπτωση **n=1000**. Ισχυριζόμαστε ότι κάτι τέτοιο δεν θα έπρεπε να συμβαίνει κανονικά (αφού το πλήθος βημάτων του αλγορίθμου υπογραμμικού χρόνου καθορίζεται μονάχα από το μέγεθος του δείγματος, και το μέγεθος του δείγματος καθορίζεται μονάχα από το **ε**, και όχι από το **n**), όμως ίσως είναι εύλογο αυτό που συμβαίνει. Αν όντως παρατηρείτε αυτό το φαινόμενο, πώς το εξηγείτε;
(Μονάδες: 0.25)

Παράρτημα: Υπόδειξη για το πιθανοθεωρητικό ερώτημα της Άσκησης 1, της Ενότητας Β.

Θέλουμε να αποδείξουμε ότι ο αλγόριθμος που περιγράψαμε (για δειγματοληψία χωρίς επανάθεση) είναι ένας σωστός αλγόριθμος για δειγματοληψία ενός υποσυνόλου **k** στοιχείων από το σύνολο **{1, 2, . . ., n}**.

Άρα, πρέπει να αποδείξουμε ότι κάθε υποσύνολο $\{x_1, x_2, \ldots, x_k\}$ του **{1, 2, . . ., n}**, έχει πιθανότητα ακριβώς **1 / (n k) = (k! · (n-k)!) / n!** να επιστραφεί από τον αλγόριθμο (ως το σύνολο **S**).

Δίχως βλάβη της γενικότητας, μπορούμε να εστιάσουμε στο υποσύνολο **{1, 2, . . ., k}** (αφού αυτά τα νούμερα δεν έχουν κάτι ιδιαίτερο, οπότε αυτό είναι απλώς ένα ενδεικτικό σύνολο **k** στοιχείων).

Παρατηρήστε ότι υπάρχουν πολλά διαφορετικά σενάρια που θα μπορούσαν να επιστρέψουν αυτό το σύνολο. Π.χ., θα μπορούσε το πρώτο στοιχείο που κάναμε δειγματοληψία να ήταν το **4**, μετά το **2**, μετά το **10**, μετά το **4** (οπότε το αγνοούμε), μετά το **6**, μετά το **2** (οπότε το αγνοούμε), κ.ο.κ.

Για να αποφύγουμε την πλήρη γενικότητα αυτής της σεναριολογίας, μία πρώτη ιδέα είναι να εστιάσουμε στην περίπτωση που τα στοιχεία μπαίνουν στο **S** με την σειρά **1, 2, 3, . . ., k**. Φυσικά, εδώ και πάλι μπορεί να είχαμε ενδιαμέσως επαναλήψεις στοιχείων που αγνοήσαμε (π.χ., μετά το **5**, μπορεί να δεχτήκαμε το **2** (οπότε το αγνοήσαμε), μετά πάλι το **2** (οπότε το αγνοήσαμε ξανά), μετά το **3** (οπότε το αγνοήσαμε και αυτό), και μετά το **6** (οπότε το βάλαμε στο σύνολο), κ.ο.κ.), όμως έχουμε περιορίσει σημαντικά τα πιθανά σενάρια.

Όμως, επειδή τώρα θεωρούμε μία συγκεκριμένη διάταξη του συνόλου **{1, 2, . . ., k}**, (και επειδή υπάρχουν **k!** τέτοιες διατάξεις), θα πρέπει να δείξουμε ότι η πιθανότητα αυτά τα στοιχεία να μπουν στο σύνολο **S** με αυτήν την σειρά είναι **(1 / (n k)) / k! = (n-k)! / n!**.

Και τώρα, η ιδέα είναι να χρησιμοποιήσουμε επαγωγή. Δηλαδή, έστω ότι η πιθανότητα να σχηματιστεί το **S** με τα στοιχεία **{1, 2, . . ., k-1}**, όπου αυτά έχουν μπει με αυτήν την συγκεκριμένη σειρά, είναι **(n-k+1)! / n!**. (Η βάση της επαγωγής, για **k-1=1**, είναι προφανής, αφού έχουμε πιθανότητα **1 / n = (n-1)! / n!** να πετύχουμε το **1**.)

Οπότε, έστω ότι στο **S** έχουν μπει τα στοιχεία **{1, 2, . . . , k-1}**, με αυτήν την σειρά, και έστω ότι αυτό το σενάριο έχει πιθανότητα ακριβώς **p = (n-k+1)! / n!**. Τώρα απομένει να απαντήσουμε το ερώτημα: ποια είναι η πιθανότητα το επόμενο στοιχείο που θα μπει στο **S** να είναι το **k**; Αν βρούμε ότι αυτή η πιθανότητα είναι **q**, τότε η τελική πιθανότητα (για τον σχηματισμό του **{1, 2, . . ., k}**, με αυτήν την σειρά) θα είναι **p·q** (το οποίο θα πρέπει να βρούμε να είναι **(n-k)! / n!**).

Θα σας αφήσουμε σε αυτό το σημείο. (Ήδη το προχωρήσαμε πάρα πολύ.) Σε περίπτωση που μπορεί να νιώσετε ότι σας χρειάζεται, σας υπενθυμίζουμε τον τύπο για το άθροισμα όρων γεωμετρικής προόδου: $1 + \alpha + \alpha^2 + \ldots = 1 / (1 - \alpha)$, όταν $0 < \alpha < 1$.