Αλγοριθμικές Τεχνικές για Δεδομένα Ευρείας Κλίμακας

2ο Φυλλάδιο Ασκήσεων

Προθεσμία υποβολής: 25/05/2025, 23:59

Ενότητα Α: Υπολογισμός Heavy Hitters σε ροές δεδομένων

Άσκηση 1: Η περίπτωση των εισαγωγών στοιχείων

Ένας ντετερμινιστικός αλγόριθμος

Έστω ότι πρόκειται να δεχτούμε μία ροή στοιχείων, και θέλουμε να υπολογίσουμε τα **φ-hitters** της ροής (δηλαδή τα στοιχεία που η συχνότητα εμφάνισής τους στην ροή ξεπερνά το **φ**). Τότε, υπάρχει ένας ντετερμινιστικός αλγόριθμος που μπορεί να αποθηκεύσει τα **φ-hitters**, χωρίς να χρειαστεί να αποθηκεύσει όλα τα διακεκριμένα στοιχεία της ροής.

Συγκεκριμένα, έστω ένας ακέραιος $\mathbf{k \geq 1}$. Τότε, εφαρμόζουμε τον εξής αλγόριθμο.

Αρχικοποίηση: μία λίστα **L**, αρχικά κενή. Αυτή η λίστα είναι για να αποθηκεύει στοιχεία της ροής (επιλεκτικά), και για κάθε στοιχείο που αποθηκεύει κρατά και έναν μετρητή για αυτό.

Εισαγωγή στοιχείου x:
1) Αν το **x** βρίσκεται στην λίστα, τότε αυξάνουμε τον μετρητή του κατά **1**.
2) Αν δεν ισχύει το 1), τότε, αν η λίστα περιέχει λιγότερα από **k** στοιχεία, προσθέτουμε το **x** στην λίστα με μετρητή **1**.
3) Αν δεν ισχύει ούτε το 1) ούτε το 2), τότε διατρέχουμε όλα τα στοιχεία της λίστας, και μειώνουμε τον μετρητή τους κατά **1**.
Αν για κάποια στοιχεία ο μετρητής τους δείχνει πλέον **0**, τότε τα αφαιρούμε από την λίστα (μαζί με τους μετρητές τους).

Τώρα ισχυριζόμαστε το εξής:

στο τέλος της ροής, κάθε **(1/(k+1))-hitter** βρίσκεται στην λίστα.

Αποδείξτε τον παραπάνω ισχυρισμό. ***(Μονάδες: 0.5)***
*(Υπόδειξη: ίσως είναι πιο εύκολο να αποδείξετε την ισοδύναμη συνθήκη: αν ένα στοιχείο **δεν** βρίσκεται στην λίστα, τότε **δεν** είναι* **(1/(k+1))-hitter***.)*

Κατασκευή μίας ροής στοιχείων

Τώρα ο σκοπός μας είναι να φτιάξουμε μία ροή στοιχείων για να δοκιμάσουμε τον παραπάνω αλγόριθμο. Συγκεκριμένα, θέλουμε μία εντελώς τυχαία ροή 1.000.000 αριθμών από το 1 μέχρι και το 10.000.000, με τα εξής χαρακτηριστικά: ένας αριθμός θα εμφανιστεί 50.000 φορές, ένας άλλος θα εμφανιστεί 2.000 φορές, και οι υπόλοιποι

θα εμφανιστούν 100 φορές ο καθένας. Επομένως, η ροή θα αποτελείται από 9482 διακεκριμένα στοιχεία.

Τώρα, για να κατασκευάσουμε μία τέτοια ροή, προτείνουμε τον εξής αλγόριθμο. Αρχικά, φτιάχνουμε έναν πίνακα **A** στον οποίο τοποθετούμε όλους τους αριθμούς από το 1 μέχρι και το 10.000.000. Έπειτα, πραγματοποιούμε μία τυχαία μετάθεση αυτού του πίνακα (δηλαδή ένα τυχαίο ανακάτεμα των στοιχείων του). Για να το πετύχουμε αυτό, διατρέχουμε μία-μία όλες τις θέσεις του πίνακα, από την πρώτη μέχρι και την προτελευταία. Για κάθε θέση **i** που θεωρούμε, επιλέγουμε μία τυχαία θέση **j** που είναι μεγαλύτερη από την **i**, ή ίση με αυτήν, και πραγματοποιούμε ανταλλαγή στοιχείων μεταξύ αυτών των δύο θέσεων. Αυτό εξασφαλίζει μία τυχαία μετάθεση του πίνακα **A**. Μολονότι η πλήρης μετάθεση δεν κοστίζει πολύ (από άποψη χρόνου), εμάς μας αρκεί απλώς να εφαρμόσουμε αυτόν τον αλγόριθμο για τις πρώτες 9482 θέσεις, διότι θέλουμε να επιλέξουμε 9482 τυχαία στοιχεία μεταξύ 1 και 10.000.000.

Τώρα κατασκευάζουμε έναν πίνακα **B** με 1.000.000 στοιχεία ως εξής. Διατρέχουμε τις πρώτες 9480 θέσεις του (μετατεθειμένου) πίνακα **A**, και κάθε στοιχείο που συναντούμε το τοποθετούμε 100 φορές στον πίνακα **B**. Έπειτα, το στοιχείο στην θέση 9481 του **A** το τοποθετούμε 2.000 φορές στον πίνακα **B**, και το στοιχείο στην θέση 9482 του **A** το τοποθετούμε 50.000 φορές στον πίνακα **B**. Τώρα ο **B** έχει γεμίσει με 1.000.000 στοιχεία. Τέλος, πραγματοποιούμε μία τυχαία μετάθεση του πίνακα **B**. Τώρα σχηματίζεται μία ροή με τις επιθυμητές ιδιότητες αν θεωρήσουμε ένα-ένα τα στοιχεία του πίνακα **B**.

Υλοποιήστε την διαδικασία που προτείναμε για την κατασκευή μίας ροής στοιχείων με τα χαρακτηριστικά που περιγράψαμε. ***(Μονάδες: 0.5)***

Υλοποιήστε τον ντετερμινιστικό αλγόριθμο που αναφέραμε, ώστε να συγκρατεί τα 0.1%-hitters. Σιγουρευτείτε ότι στην έξοδό του περιέχονται οπωσδήποτε τα δύο μοναδικά 0.1%-hitters της ροής (αν και μπορεί να υπάρχουν και πολλά ακόμη έξτρα στοιχεία).
(Μονάδες: 0.5)

Αυτά τα έξτρα στοιχεία που επιστρέφει ο ντετερμινιστικός αλγόριθμος (που δεν είναι 0.1%-hitters) είναι κάπως ενοχλητικά, διότι δίνουν την εντύπωση πως είναι πολύ σημαντικά, ενώ δεν είναι. Προτείνουμε την εξής ιδέα για να ξεκαθαρίσουμε κάπως το τοπίο. Για κάθε στοιχείο που είναι αποθηκευμένο στην λίστα **L**, κρατούμε και έναν δεύτερο μετρητή, ο οποίος αρχικοποιείται με **1** όταν το στοιχείο μπαίνει για πρώτη φορά στην λίστα, και αυξάνεται κατά **1** όταν το συναντούμε ξανά στην ροή (και υπάρχει ήδη στην λίστα), και δεν μειώνεται ποτέ (εκτός αν το στοιχείο βγει από την λίστα, οπότε αναγκαστικά ξεχνάμε πλέον τα πάντα για αυτό). Μπορούμε να αξιοποιήσουμε κάπως αυτόν τον δεύτερο μετρητή για να έχουμε μία εκτίμηση για το πραγματικό πλήθος εμφανίσεων των στοιχείων στην ροή; Υπάρχει εγγύηση ότι, αν ένα στοιχείο είναι 0.1%-hitter, τότε αυτός ο μετρητής θα πρέπει π.χ. να είναι μεγαλύτερος από κάποια τιμή (σε σχέση με το μέγεθος της ροής); Προτείνετε μερικές εύλογες σκέψεις πάνω σε αυτά τα ερωτήματα. Για ό,τι θεωρείτε ότι δεν ισχύει, δώστε και ένα αντιπαράδειγμα (με την μορφή κάποιας ανταγωνιστικής ροής αριθμών, οσοδήποτε μεγάλου μεγέθους). ***(Μονάδες: 0.25)***

Εφαρμογή του CountMin για το ίδιο πρόβλημα

Υλοποιήστε τον αλγόριθμο που αναφέραμε στο μάθημα (και που περιγράφεται και στην εργασία "An improved data stream summary: the count-min sketch and its applications" των Cormode και Muthukrishnan), για να υπολογίσετε τα 0.1%-hitters της ροής που προτείναμε. Φροντίστε να κάνετε μια όσο το δυνατόν πιο οικονομική επιλογή των παραμέτρων του CountMin, ώστε να εξασφαλίσετε ότι: με πιθανότητα τουλάχιστον 99%, όλα τα στοιχεία που θα επιστραφούν θα είναι τουλάχιστον 0.02%-hitters. Αυτή η αρχικοποίηση θα πρέπει να γίνει ανεξάρτητα από την γνώση που έχετε για την κατανομή των στοιχείων στην ροή που θα δεχτεί ο αλγόριθμος. Η μόνη γνώση που σας επιτρέπεται να αξιοποιήσετε είναι ότι η ροή θα αποτελείται από αριθμούς μεταξύ 1 και 10.000.000. Σε περίπτωση που μπορεί να νιώσετε ότι σας χρειάζεται, σας δίνουμε την πληροφορία ότι ο p = 10.000.019 είναι ένας πρώτος αριθμός.
(Μονάδες: 2)

Άσκηση 2: Ανάκτηση των Heavy Hitters από το σκιαγράφημα του CountMin

Ένας φίλος σας σκέφτηκε να εφαρμόσει έναν από τους αλγορίθμους της εργασίας "An improved data stream summary: the count-min sketch and its applications", των Cormode και Muthukrishnan, προκειμένου να υπολογίσει τα **Top-3** στοιχεία μίας ροής αριθμών μεταξύ του **0** και του $\mathbf{2^{100} - 1}$.

Αυτό που του έδωσε την αυτοπεποίθηση να χρησιμοποιήσει τον CountMin ήταν ότι είχε την πληροφόρηση ότι η ροή που πρόκειται να δεχτεί θα έχει τα ακόλουθα χαρακτηριστικά:
1) Ακριβώς τρία στοιχεία θα είναι τουλάχιστον 10%-hitters
2) Τα υπόλοιπα στοιχεία δεν θα είναι ούτε καν 1%-hitters.

Οπότε, με βάση αυτά που διάβασε, είχε την εξής ιδέα. Αρχικοποίησε 100 διαφορετικές δομές CountMin, ας τις πούμε CM_1, CM_2, . . ., CM_{100}, όπου κάθε δομή έχει 15 γραμμές (δηλαδή πίνακες κατακερματισμού) των 277 θέσεων. Οι συναρτήσεις κατακερματισμού που χρησιμοποίησε ήταν τυχαίες επιλογές από την οικογένεια Vec_{277}. (Σημειώνουμε ότι ο αριθμός 277 είναι πρώτος.)

Συγκεκριμένα, κάθε συνάρτηση κατακερματισμού καθορίστηκε από μία τυχαία επιλογή από 13 αριθμούς μεταξύ 0 και 276. Κάθε τέτοια συνάρτηση κατακερματισμού, που δίνεται από τους αριθμούς $(\alpha_1, \alpha_2, \ldots, \alpha_{13})$, δίνει την τιμή $(\alpha_1x_1 + \alpha_2x_2 + \ldots + \alpha_{13}x_{13})\text{mod}277$, πάνω σε οποιαδήποτε 13-άδα αριθμών $(x_1, x_2, \ldots, x_{13})$.

Τώρα, η διαδικασία που εφάρμοζε ο φίλος σας, σε κάθε αριθμό **x** της ροής, ήταν η εξής. Η πρώτη δουλειά ήταν να μετατρέψει τον **x** στο δυαδικό σύστημα αρίθμησης, ενδεχομένως προσθέτοντας στο τέλος αρκετά μηδενικά, ώστε να σχηματιστεί μία συμβολοσειρά των 100 bits. (Οπότε, το πρώτο bit της συμβολοσειράς ήταν το λιγότερο σημαντικό bit του αριθμού. Για παράδειγμα, η συμβολοσειρά 1011000000...00 συμβολίζει τον αριθμό 13.) Έπειτα, για κάθε CM_i, κρατούσε τα πρώτα i bits της συμβολοσειράς, και αυτά τα μετέτρεπε σε έναν αριθμό (όπου και πάλι το πρώτο bit της (υπο)συμβολοσειράς είναι το λιγότερο σημαντικό bit). Έπειτα, αυτόν τον αριθμό τον μετέτρεπε στο σύστημα αρίθμησης με βάση το 277, και έτσι λάμβανε μία 13-άδα

αριθμών (προσθέτοντας στις τελευταίες θέσεις μηδενικά, αν χρειαζόταν). (Ώστε και πάλι, το πρώτο στοιχείο αυτής της 13-άδας, είναι το λιγότερο σημαντικό ψηφίο στην αναπαράσταση του αριθμού στο σύστημα αρίθμησης με βάση το 277.) Έπειτα, απλώς εφάρμοζε τις 15 συναρτήσεις κατακερματισμού του CM_i πάνω σε αυτήν την 13-άδα, και τροποποιούσε ανάλογα το κελί που έδειχνε η κάθε μία. Συγκεκριμένα, η ροή αποτελούταν από εντολές εμφάνισης στοιχείου (οπότε προσετίθετο +1 στον αντίστοιχο μετρητή) ή διαγραφής στοιχείου (όπου αφαιρούταν μία μονάδα από τον αντίστοιχο μετρητή), αλλά ποτέ δεν γινόταν διαγραφή στοιχείου που δεν είχε απομείνει έστω μία εμφάνισή του. (Το πόσες εντολές έλαβαν χώρα συνολικά, δεν το γνωρίζουμε.)

Στο τέλος της ροής, ο φίλος σας μάζεψε όλα τα δεδομένα του CountMin σε ένα αρχείο, αλλά συνάντησε ένα πρόβλημα: επειδή ό,τι έκανε το εφάρμοσε μηχανικά, τώρα δεν γνωρίζει πώς να ανακτήσει τα **Top-3** στοιχεία που τον ενδιαφέρουν. Επειδή όμως έμαθε ότι εσείς παρακολουθήσατε ένα σχετικό μάθημα, σκέφτηκε να ζητήσει την βοήθειά σας. Για να θυμηθείτε κι όλας την απόγνωσή του, ακολουθεί και ένα απόσπασμα από την συνομιλία που είχατε:

"...
[ο φίλος σας]: *δεν έχω ιδέα τι να κάνω... μου φαίνονται όλα τυχαίοι αριθμοί* 😭😭😭
..."

Στην σελίδα του μαθήματος στο ecourse, στον φάκελο CountMinSketch, θα βρείτε δύο αρχεία με όλα τα δεδομένα που συγκράτησε ο φίλος σας. Το αρχείο hash_functions.txt περιέχει τους αριθμούς που ορίζουν τις συναρτήσεις κατακερματισμού που χρησιμοποίησε. Συγκεκριμένα, αυτό το αρχείο αποτελείται από 1500 γραμμές, όπου η κάθε γραμμή αντιστοιχεί σε μία συνάρτηση κατακερματισμού, και αποτελείται από τους 13 αριθμούς μεταξύ 0 και 276 που την ορίζουν. Η πρώτη 15-άδα γραμμών δίνει τις συναρτήσεις κατακερματισμού του CM_1, η δεύτερη 15-άδα γραμμών δίνει τις συναρτήσεις κατακερματισμού του CM_2, κ.ο.κ.

Στο αρχείο sketch.txt βρίσκονται οι τιμές των μετρητών των CountMin. Αυτό το αρχείο αποτελείται από 1500 γραμμές, όπου η κάθε γραμμή αντιστοιχεί σε μια γραμμή ενός CountMin, και άρα αποτελείται από 277 αριθμούς. Η πρώτη 15-άδα γραμμών δίνει τις γραμμές του CM_1, η δεύτερη 15-άδα γραμμών δίνει τις γραμμές του CM_2, κ.ο.κ., και όλα είναι σε πλήρη αντιστοίχιση με τις συναρτήσεις κατακερματισμού του πρώτου αρχείου.

Βοηθήστε τον φίλο σας να βρει τους **Top-3** αριθμούς της ροής. Έπειτα, δώστε μία εκτίμηση για το πλήθος των εμφανίσεών τους στην ροή, και υποστηρίξτε ότι η εκτίμηση που δώσατε για κάθε έναν από αυτούς έχει πιθανότητα μεγαλύτερη του 99.99% να μην απέχει παραπάνω από 1000 από την πραγματική τους τιμή.
(Μονάδες: 2.5)

Άσκηση 3: Αλγόριθμος για τον υπολογισμό του F_∞

Πρόκειται να δεχτούμε μία ροή που αφορά εισαγωγές και διαγραφές αριθμών από το **1** μέχρι και το **n**. Μας δίνεται ως εγγύηση ότι, στο τέλος της ροής, θα έχει απομείνει τουλάχιστον ένας **φ-hitter**, για κάποιο **φ** για το οποίο ισχύει ότι $\mathbf{0 < \varphi < 1}$.

Δείξτε ότι, χρησιμοποιώντας χώρο $\mathbf{O(poly(1/\varphi,\ 1/\varepsilon,\ \log(1/\delta),\ \log(n)))}$, μπορούμε να υπολογίσουμε μία εκτίμηση **Y** για το F_∞ της ροής, για την οποία ισχύει ότι
$\mathbf{F_\infty \le Y \le (1+\varepsilon)F_\infty}$, με πιθανότητα τουλάχιστον $\mathbf{1 - \delta}$, όπου $\boldsymbol{\varepsilon, \delta > 0}$ είναι δύο προεπιλεγμένοι αριθμοί. *(Υπόδειξη: χρησιμοποιήστε τον CountMin)*
Όσο ενδελεχέστερα υποστηρίξετε το φράγμα που δώσατε, τόσο περισσότερες μονάδες μπορείτε να διεκδικήσετε.
(Μονάδες: 1)

Αυτό το αποτέλεσμα φαίνεται να αναιρεί αυτό που ξέρουμε για το lower bound στην χρήση μνήμης για τον υπολογισμό του F_∞. Ισχύει κάτι τέτοιο; Τεκμηριώστε την απάντησή σας.
(Μονάδες: 0.25)

Άσκηση 4: Υπολογισμός **Top** στοιχείων σε εισόδους που ακολουθούν κατανομή Zipf

Στην σελίδα του μαθήματος στο ecourse, θα βρείτε ένα αρχείο κειμένου με τίτλο "vilfredo.txt", που περιέχει το κείμενο ενός βιβλίου (σε κάπως ανακριβή μορφή, καθώς αποτελεί μηχανική αποτύπωση σκαναρισμένης εικόνας σε κείμενο).

Είναι γνωστό ότι οι συχνότητες εμφάνισης των λέξεων σε έργα ανθρώπων τείνουν να ακολουθούν κατανομή Zipf (ή Pareto). Αυτό ίσως διευκολύνει αρκετά τον CountMin να βρει με μεγαλύτερη ακρίβεια τα **Top** στοιχεία σε ροές που σχηματίζονται από αυτά τα έργα, χρησιμοποιώντας αρκετά λιγότερο χώρο απ' όσο απαιτείται συνήθως σε γενικές ροές δεδομένων, προκειμένου να έχουμε ικανοποιητικά καλές εγγυήσεις (στην προσέγγιση, και την πιθανότητα επιτυχίας).

Αυτό σκοπεύουμε να το εξετάσουμε με το συγκεκριμένο κείμενο. Η πρώτη δουλειά θα είναι ακριβώς να σχηματίσουμε μία ροή από αυτό. Για να το πετύχουμε αυτό, για αρχή θα εξαγάγουμε όλες τις λέξεις από το κείμενο. "Λέξη" για εμάς σημαίνει μία συμβολοσειρά λατινικών χαρακτήρων. Επειδή δεν έχει γίνει απολύτως ακριβής μετατροπή της αρχικής εικόνας σε κείμενο, υπάρχουν διάφορα σημεία που δεν ανταποκρίνονται καν σε λέξεις της αγγλικής γλώσσας. Εμείς θα δουλέψουμε ως εξής. Θα μαζέψουμε όλες τις μεγιστικές συμβολοσειρές που δεν περιέχουν κενό " ". Για κάθε τέτοια συμβολοσειρά, θα μετατρέψουμε όλα τα κεφαλαία γράμματα σε μικρά, θα αφαιρέσουμε οποιονδήποτε χαρακτήρα δεν αποτελεί λατινικό γράμμα, και θα κρατήσουμε το αποτέλεσμα σε συμπτυγμένη μορφή. (Π.χ., έτσι θα μετατρέπαμε την συμβολοσειρά "a,Bc..s12b" στην "abcsb".) Επειδή μπορεί να εμφανιστούν ορισμένες τεράστιες συμβολοσειρές, θα απορρίπτουμε (δηλαδή δεν θα λαμβάνουμε υπ' όψιν ως μέρος της ροής) κάθε συμβολοσειρά με πάνω από 15 χαρακτήρες.

Εφόσον έχουμε εφαρμόσει αυτήν την διαδικασία σε κάθε μεγιστική συμβολοσειρά που συναντήσαμε που δεν περιέχει κενό " ", (και απορρίψαμε τις τεράστιες συμβολοσειρές), έχουμε σχηματίσει μία ροή "λέξεων". Αποθηκεύστε αυτήν την ροή σε ένα αρχείο stream.txt, ώστε να μην χρειαστεί να ξαναδιαβάσετε το βιβλίο. Τώρα, εφόσον έχετε διαθέσιμη την ροή, υπολογίστε το πλήθος εμφανίσεων της κάθε λέξης, και ταξινομήστε τες σε φθίνουσα σειρά ως προς το πλήθος εμφανίσεων. Αποθηκεύστε το αποτέλεσμά σας σε ένα αρχείο distinct_words_with_count.txt, όπου κάθε γραμμή θα έχει την μορφή "[word] [count]". Για παράδειγμα, εμείς βρήκαμε ότι η πρώτη γραμμή είναι η "the 19038", που σημαίνει ότι η λέξη "the" εμφανίστηκε 19038 φορές, και είναι το **Top-1** στοιχείο της ροής. (Για εσάς το αποτέλεσμα μπορεί να διαφέρει λιγάκι, διότι αυτό για εμάς προέκυψε από κάποιες αυτοματοποιημένες ρουτίνες της Python.) Ήδη για τον κόπο σας αν έχετε μπει σε αυτήν την διαδικασία, μέχρις εδώ δίνουμε:
(Μονάδες: 0.25)

Τώρα θα κάνουμε μία άμεση χρήση του CounMin για να επιχειρήσουμε να βρούμε τα **Top-10** στοιχεία σε αυτήν την ροή. Συγκεκριμένα, διατηρούμε απλώς μία (αρχικά κενή) λίστα 10 στοιχείων. Όσο η λίστα δεν έχει φουλάρει, προσθέτουμε απλώς κάθε λέξη που συναντούμε (αρκεί να μην υπάρχει ήδη μέσα). Σε κάθε περίπτωση, βάζουμε τον CountMin να ανανεώσει τους μετρητές της λέξης. Τώρα, αν η λέξη δεν βρίσκεται στην λίστα, ζητούμε κατευθείαν από τον CountMin μία εκτίμηση για το πλήθος εμφανίσεων της λέξης, και συγκρίνουμε αυτήν την εκτίμηση με όλες τις εκτιμήσεις για τα στοιχεία που υπάρχουν ήδη στην λίστα. Αν η εκτίμηση της λέξης ξεπερνάει την μικρότερη εκτίμηση στοιχείου της λίστας, τότε αντικαθιστούμε την λέξη της λίστας με την μικρότερη εκτίμηση (για το πλήθος εμφανίσεών της) με την τωρινή λέξη που εμφανίστηκε στην ροή. Στο τέλος της ροής, εκτυπώστε τα 10 στοιχεία της λίστας με τις εκτιμήσεις τους. Προτείνουμε να πειραματιστείτε με διάφορα **ε** και **δ** για τον CountMin, και κρατήστε κάποια που σας φαίνεται ότι δουλεύουν αρκετά καλά. (Μπορείτε να έχετε μία τέλεια εικόνα για την ποιότητα της λύσης, εφόσον έχετε τα ακριβή αποτελέσματα στο αρχείο distinct_words_with_count.txt.) Επαναλάβετε την ίδια διαδικασία και για τα **Top-100** στοιχεία.

Οι συναρτήσεις κατακερματισμού που θα χρησιμοποιήσετε θα πρέπει να είναι από μία οικογένεια της μορφής $\mathbf{Vec_p}$, για κάποιον πρώτο αριθμό **p**. Εφόσον το σύγχρονο λατινικό αλφάβητο περιέχει 26 γράμματα, μπορείτε να δείτε την κάθε λέξη της ροής ως έναν αριθμό στο σύστημα αρίθμησης με βάση το 26. Εφόσον οι "λέξεις" μας είναι συμβολοσειρές με το πολύ 15 χαρακτήρες, μπορούμε λοιπόν να δούμε την ροή ως μια ροή αριθμών από το 0 μέχρι και το $26^{15} - 1$.
(Μονάδες: 1.25)

Ενότητα Β: Ανάκτηση 1-αραιού διανύσματος

Άσκηση 1: Κατασκευή μίας κατανομής εισόδου

Σε αυτήν την άσκηση, ο σκοπός είναι να κατασκευάσουμε μία μη-τετριμμένη κατανομή εισόδου για τον μηχανισμό ανάκτησης 1-αραιού διανύσματος. Συγκεκριμένα, θέλουμε η κατανομή να έχει το χαρακτηριστικό ότι το διάνυσμα μεταβάλλεται με σχεδόν τυχαίο τρόπο, όμως σχετικά συχνά εξασφαλίζεται κι όλας ότι υπάρχει ακριβώς μία μη-μηδενική συντεταγμένη, η οποία θα πρέπει να ποικίλλει.

Για να έχουμε αυτά τα χαρακτηριστικά, προτείνουμε μία γεννήτρια εντολών που τροποποιούν ένα (αρχικά μηδενικό) διάνυσμα **x** με 10.000 συντεταγμένες, αριθμημένες από το **1** μέχρι και το **10.000**. Ο σκοπός της γεννήτριας θα είναι να παράγει εντολές της μορφής **(i, c)**, όπου μία τέτοια εντολή προσθέτει στην συντεταγμένη **i** την τιμή **c** (όπου ο **c** είναι ακέραιος).

Η λειτουργία της γεννήτριας διακρίνεται σε τέσσερις φάσεις, τις οποίες συμβολίζουμε με **Ο**, **Α**, **Β**, **Γ**, και τις οποίες ονομάζουμε αντίστοιχα: "φάση 0", "κανονική εξέλιξη", "επιστροφή στο 0", και "ξαφνική έκρηξη".

Να σημειώσουμε ότι, καθώς η γεννήτρια παράγει τις εντολές της, υποθέτουμε ότι τις εφαρμόζει κι όλας στο διάνυσμα **x**, οπότε έχει πλήρη γνώση του τι έχει συμβεί σε αυτό.

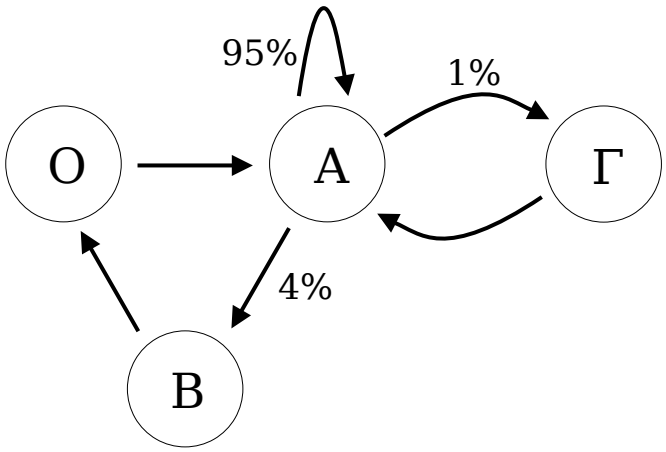
Η γεννήτρια ξεκινάει από την φάση 0. Σε αυτήν την φάση επιλέγεται τυχαία μία συντεταγμένη **s** από το 1 μέχρι και το 10.000, και παράγεται η εντολή **(s,1)**. Η **s** είναι μια ειδική συντεταγμένη, που αρχικοποιείται κάθε φορά στην φάση 0. Έπειτα από την φάση 0, γίνεται μετάβαση στην φάση **Α**.

Η φάση **Α** χωρίζεται με την σειρά της σε τρεις φάσεις, **Α1**, **Α2**, **Α3** (όπου μόλις ολοκληρώνεται η μία περνούμε κατευθείαν στην άλλη). Στην φάση **Α1**, αν το διάνυσμα **x** περιέχει κάποια μη-μηδενική συντεταγμένη διάφορη της **s**, τότε με πιθανότητα 1/3 παίρνεται η απόφαση να μηδενιστεί μία τυχαίως επιλεγμένη από αυτές. Οπότε, αν αποφασιστεί να μηδενιστεί μία τέτοια συντεταγμένη **i**, παράγεται η εντολή **(i,-x_i)**. Στην φάση **Α2**, με πιθανότητα 1/3 παίρνεται η απόφαση να προστεθεί μία τιμή σε μία συντεταγμένη διάφορη της **s**. Οπότε, αν αποφασιστεί κάτι τέτοιο, για αρχή επιλέγεται εντελώς τυχαία μία συντεταγμένη **i∈{1, 2, . . ., 10.000}\{s}**. Έπειτα, η τιμή που θα προστεθεί στην **i**, είναι ένας τυχαίος μη-μηδενικός ακέραιος **c** μεταξύ -10 και 10. Οπότε, εδώ παράγεται η εντολή **(i,c)**. Τέλος, στην φάση **Α3**, με πιθανότητα 1/2 παίρνεται η απόφαση να πειραχτεί η συντεταγμένη **s**. Αν αποφασιστεί κάτι τέτοιο, τότε επιλέγεται τυχαία να προστεθεί στην **s** είτε το 2 είτε το -2. Οπότε, παράγεται μια εντολή της μορφής **(s,c)**, όπου **c∈{-2,2}**. Μόλις ολοκληρωθεί και η φάση **Α3**, παίρνεται η απόφαση είτε να συνεχίσουμε με την κανονική εξέλιξη, είτε να μεταβούμε σε κάποια από τις φάσεις **Β** ή **Γ**. Συγκεκριμένα, με πιθανότητα 95% συνεχίζουμε στην φάση **Α**, με πιθανότητα 4% μεταβαίνουμε στην φάση Β, και με πιθανότητα 1% πηγαίνουμε στην φάση **Γ**.

Η φάση **Β** λέγεται "επιστροφή στο 0", γιατί ο σκοπός της είναι ακριβώς να μηδενίσει όλες τις συντεταγμένες του διανύσματος. Οπότε, όσο υπάρχει μη-μηδενική συντεταγμένη, επιλέγεται εντελώς τυχαία μία από αυτές, έστω η **i**, και παράγεται η εντολή **(i,-x_i)**. Μόλις το διάνυσμα μηδενιστεί, μεταβαίνουμε στην φάση 0.

Τέλος, στην φάση **Γ** γίνεται μία απότομη έκρηξη αλλαγών στο διάνυσμα. Συγκεκριμένα, για 100 συνεχόμενους γύρους, επιλέγεται τυχαία μία συντεταγμένη διάφορη της **s**, και προστίθεται σε αυτήν μία τυχαία μη-μηδενική τιμή μεταξύ -10 και 10 (και παράγεται και η σχετική εντολή). Μόλις αυτή η φάση ολοκληρωθεί, μεταβαίνουμε στην φάση **Α**.

Συνοπτικά, στο ακόλουθο σχήμα βλέπουμε το διάγραμμα μεταβάσεων μεταξύ των τεσσάρων φάσεων.



Χρησιμοποιήστε την γεννήτρια εντολών που περιγράψαμε για να κατασκευάσετε μία ροή 10.000.000 εντολών.
(Μονάδες: 1.25)

Άσκηση 2: Υλοποίηση του μηχανισμού ανάκτησης 1-αραιού διανύσματος

Τώρα θα πρέπει να υλοποιήσετε τον μηχανισμό ανάκτησης 1-αραιού διανύσματος, και να τον τροφοδοτήσετε με την ροή που παράγει η γεννήτρια που περιγράψαμε στην προηγούμενη άσκηση. (Ωστε αυτός ο μηχανισμός πρόκειται να διαχειριστεί ένα διάνυσμα με 10.000 συντεταγμένες, αριθμημένες από το 1 μέχρι και το 10.000.)

Προτείνουμε πρώτα να χρησιμοποιήσετε την γεννήτρια για να φτιάξετε τις 10.000.000 εντολές, να τις αποθηκεύσετε σε έναν πίνακα, και έπειτα να τις τροφοδοτήσετε στον μηχανισμό μία προς μία.

Για αυτόν τον μηχανισμό θα χρησιμοποιήσουμε τον πρώτο αριθμό p=20.011 (που είναι ο μικρότερος πρώτος αριθμός μεγαλύτερος του 20.000). Αυτό που έχει σημασία είναι να επιλέξουμε το πλήθος **Τ** των αριθμών $\mathbf{r_1}$, $\mathbf{r_2}$, . . ., $\mathbf{r_T}$, που ο καθένας είναι τυχαία επιλεγμένος από το σύνολο **{1, 2, . . . , p – 1}**, και που χρησιμεύουν για να μειωθεί η

πιθανότητα false-positive του μηχανισμού ανάκτησης. (Υπενθυμίζουμε ότι false-positive εδώ σημαίνει ο μηχανισμός να ισχυρίζεται ότι το διάνυσμά του περιέχει ακριβώς μία μη-μηδενική συντεταγμένη, χωρίς να ισχύει κάτι τέτοιο. Από την άλλη, δεν υπάρχουν false-negatives. Δηλαδή, οποτεδήποτε το διάνυσμα περιέχει ακριβώς μία μη-μηδενική συντεταγμένη, ο μηχανισμός είναι σε θέση να την εντοπίσει, μαζί με την τιμή της.)

Το **T**, για να είμαστε σωστοί, θα πρέπει να το επιλέξουμε με βάση την θεωρητική ανάλυση που είδαμε, προκειμένου να εξασφαλίσουμε μία καλή πιθανότητα επιτυχίας. Παρά ταύτα, εμείς εδώ θα δοκιμάσουμε μονάχα ορισμένες μικρές τιμές του, ξεκινώντας από **T=1**, μέχρι και **T=10**.

Για κάθε εντολή (της μορφής **(i,c)**) που λαμβάνουμε, πρώτα την τροφοδοτούμε στον μηχανισμό, και έπειτα θέτουμε κατευθείαν το ερώτημα για το αν το διάνυσμα περιέχει ακριβώς μία μη-μηδενική συντεταγμένη (το ποια ακριβώς είναι αυτή, δεν μας ενδιαφέρει εδώ πέρα, αλλά θα πρέπει να τσεκάρετε μόνοι σας κάπως ότι η υλοποίησή σας είναι σωστή, και ότι αν σας ζητούταν και η μη-μηδενική συντεταγμένη θα μπορούσατε να την επιστρέψετε εύκολα). Το πρόγραμμά σας θα πρέπει, μετά από κάθε εντολή, να τυπώνει, σε μία γραμμή (στην γραμμή εντολών), τον αριθμό της τρέχουσας εντολής (δηλαδή αν είναι η 1η, η 2η, κτλ.), και το πλήθος των σφαλμάτων που έχουν γίνει συνολικά μέχρι εκείνο το σημείο. (Οπότε, μία γραμμή εξόδου μπορεί π.χ. να έχει την μορφή "9000050: 17", που σημαίνει ότι εκτελέστηκε η 9.000.050-η εντολή, και συνολικά έχουν λάβει χώρα 17 false-positives.)
(Μονάδες: 1.5)

Θα θέλαμε να έχουμε πιθανότητα τουλάχιστον 99% να μην υπάρξει ούτε ένα σφάλμα κατά την διάρκεια της ροής. Με βάση την θεωρητική ανάλυση που γνωρίζετε, τι τιμή θα πρέπει να δώσουμε στο **T**;
(Μονάδες: 0.125)

Παρά τις απαιτήσεις της θεωρητικής ανάλυσης, ίσως παρατηρήσατε ότι ήδη για πολύ μικρές τιμές του **T** (ίσως π.χ. και για **T=2**), δεν υπάρχει ούτε ένα σφάλμα κατά την διάρκεια της ροής. Δώστε μία εύλογη εξήγηση για την απαισιοδοξία της θεωρητικής ανάλυσης.
(Μονάδες: 0.125)

Ενότητα Γ: Δειγματοληψία μη-μηδενικής συντεταγμένης

Άσκηση 1: Η περίπτωση των μη-αρνητικών τιμών

α) Έστω μία ροή εντολών της μορφής **(i, c)**, όπου κάθε τέτοια εντολή προσθέτει την τιμή **c** στην **i** συντεταγμένη ενός (αρχικά μηδενικού) διανύσματος που οι συντεταγμένες του είναι αριθμημένες από το **1** μέχρι και το **n**.

Θέλουμε να διατηρήσουμε ένα σκιαγράφημα του διανύσματος, ώστε ανά πάσα ώρα και στιγμή να είμαστε σε θέση να επιστρέψουμε μία μη-μηδενική συντεταγμένη του (αν υπάρχει) με πιθανότητα τουλάχιστον $\mathbf{1 - \delta}$, για κάποιο προκαθορισμένο $\boldsymbol{\delta > 0}$.

Στην περίπτωση που έχουμε την εγγύηση ότι, όταν λάβουμε εντολή για δειγματοληψία, καμία συντεταγμένη του διανύσματος δεν έχει αρνητική τιμή, τότε μπορούμε να δώσουμε μία πιο απλή κατασκευή για τον non-zero sampler, επειδή με αυτήν την εγγύηση απλοποιείται το πρόβλημα της ανάκτησης 1-αραιού διανύσματος. (Τονίζουμε ότι οι αυξομειώσεις στις συντεταγμένες επιτρέπονται, απλώς δεν θα πρέπει καμία συντεταγμένη να έχει αρνητική τιμή την στιγμή της δειγματοληψίας.)

Συγκεκριμένα, με την εγγύηση των μη-αρνητικών τιμών, για το πρόβλημα της ανάκτησης 1-αραιού διανύσματος υπάρχει ένας πολύ απλός ντετερμινιστικός αλγόριθμος, που διατηρεί απλώς τρεις αριθμούς:

$\boldsymbol{\alpha = x_1 + x_2 + \ldots + x_n}$
$\boldsymbol{\beta = 1 \cdot x_1 + 2 \cdot x_2 + \ldots + n \cdot x_n}$
$\boldsymbol{\gamma = 1^2 \cdot x_1 + 2^2 \cdot x_2 + \ldots + n^2 \cdot x_n}$,

όπου **x** είναι το διάνυσμα που τροποποιεί η ροή.

Σε αυτήν την περίπτωση, την στιγμή που πρόκειται να ρωτηθούμε αν το διάνυσμα είναι 1-αραιό, αρκεί να ελέγξουμε ότι:

$\boldsymbol{\alpha \neq 0}$, ο $\boldsymbol{\beta/\alpha}$ είναι ακέραιος, και $\boldsymbol{\gamma \cdot \alpha = \beta^2}$.

Ισχύει ότι το διάνυσμα **x** είναι 1-αραιό αν και μόνο αν ικανοποιείται η παραπάνω συνθήκη, και τότε η μοναδική μη-μηδενική συντεταγμένη του **x** είναι η $\boldsymbol{\beta/\alpha}$ (που έχει τιμή $\boldsymbol{\alpha}$). (Για την απόδειξη αυτού του ισχυρισμού, παραπέμπουμε στην εργασία "Sampling in dynamic data streams and applications", των G. Frahling, P. Indyk, και C. Sohler.)

Επομένως, το πρόβλημα της κατασκευής του non-zero sampler απλοποιείται σημαντικά: αρκεί να έχουμε $\boldsymbol{\lceil \log_2(n+1) \rceil}$ επίπεδα δειγματοληψίας, και σε κάθε επίπεδο **T** επαναλήψεις, όπου το **T** καθορίζεται πλήρως από το **δ**. (Δείτε τις σχετικές διαφάνειες που αφορούν τον non-zero sampler.)

Σε αυτήν την άσκηση, εφόσον ασχολούμαστε με μη-αρνητικές τιμές, θα θεωρήσουμε ένα διάνυσμα που αναπαριστά μια συμβολοσειρά των 10.000 bits. Με άλλα λόγια,

έχουμε ένα διάνυσμα **x**, που οι συντεταγμένες του είναι αριθμημένες από το **1** μέχρι και το **10.000**, και η τιμή σε κάθε συντεταγμένη είναι είτε 0 ή 1. Για αρχή, φτιάξτε ένα τέτοιο διάνυσμα με τον εξής τρόπο: πολύ απλά, διατρέξτε μία-μία τις συντεταγμένες, και σε κάθε θέση βάλτε τιμή 0 με πιθανότητα 3/4.

Τώρα αρχικοποιήστε έναν non-zero sampler (με την απλούστερη λύση για το πρόβλημα της ανάκτησης 1-αραιού διανύσματος που περιγράψαμε), επιλέγοντας το **T** με τέτοιον τρόπο ώστε, σύμφωνα με την θεωρητική ανάλυση που γνωρίζετε, να εξασφαλιστεί ότι ο δειγματολήπτης θα μπορέσει να απαντήσει 10.000 ερωτήματα σωστά με πιθανότητα τουλάχιστον 99%. (Δηλαδή, πρόκειται να του θέσουμε το πολύ 10.000 ερωτήματα, και θέλουμε η πιθανότητα να τα απαντήσει όλα σωστά να είναι τουλάχιστον 99%.)

Τώρα, εφόσον έχετε το διάνυσμα **x**, και αφού αρχικοποιήσατε τον δειγματολήπτη, διατρέξτε όλες τις συντεταγμένες του **x**, και, για κάθε συντεταγμένη **i** που είναι μη-μηδενική, τροφοδοτήστε τον δειγματολήπτη με την εντολή **(i, 1)**.

Τώρα αρχίστε να διατρέχετε τις συντεταγμένες του **x** ξανά. Για κάθε μη-μηδενική συντεταγμένη **i** που συναντάτε, θα τροφοδοτείτε τον δειγματολήπτη με την εντολή **(i,-1)**, και θα θέτετε κατευθείαν ένα ερώτημα για μη-μηδενική συντεταγμένη. Αν ο δειγματολήπτης αποτύχει (όπου αφήνουμε να κρίνετε εσείς - και να εξηγήσετε - τι σημαίνει "αποτυχία" στην προκειμένη περίπτωση), τότε θα πρέπει να τυπωθεί στην γραμμή εντολών το μήνυμα "error", και να τερματιστεί η εκτέλεση του προγράμματος.
(Μονάδες: 2)

Φυσικά, δεδομένου ότι επιλέξατε το **T** κατάλληλα, η πιθανότητα αποτυχίας θα πρέπει να είναι το πολύ 1%. (Άρα, αν ο δειγματολήπτης σας αποτυγχάνει συχνά, αυτό σημαίνει ότι είτε κάνατε λάθος στην υλοποίηση, ή στην επιλογή του **T**, ή στην ερμηνεία του τι σημαίνει αποτυχία για τον συγκεκριμένο δειγματολήπτη.)

Ακόμη περισσότερο: αν κάνετε μερικές δοκιμές όπου το **T** είναι πολύ μικρότερο από αυτό που επιτάσσει η θεωρία (π.χ., κάτω από 5), τότε και πάλι μπορεί να διαπιστώσετε ότι (πρακτικά) το ενδεχόμενο αποτυχίας είναι πολύ σπάνιο. Βέβαια, αν το αρχικό διάνυσμα είναι αρκετά αραιό (δηλαδή, αν επιλέξετε αρχικά να βάζετε την τιμή 0 σε κάθε συντεταγμένη με πιθανότητα π.χ. 99/100), τότε ίσως χρειάζεται να αυξήσετε λιγάκι το πλήθος των επαναλήψεων **T**, αλλά και πάλι: μπορεί να είναι αρκετά μικρότερο από αυτό που επιτάσσει η θεωρία, και όμως η πιθανότητα επιτυχίας θα είναι σημαντικά μεγάλη. Παρατηρείτε όντως κάτι τέτοιο; Αν ναι, δώστε μία εύλογη εξήγηση για αυτά τα δύο φαινόμενα.
(Μονάδες: 0.25)

β) Με αφορμή το θέμα **α)** αυτής της άσκησης, ένας φίλος σας (πιθανότατα ο ίδιος που είδαμε και σε προηγούμενη άσκηση) είχε μια ενδιαφέρουσα ιδέα. Επειδή ο ενθουσιασμός του είναι αμίμητος, προτιμάμε να σας μεταφέρουμε τα λόγια του αυτούσια:

"[ο φίλος σας]: *Βρήκα έναν τέλειο αλγόριθμο για συμπίεση πληροφορίας! Μπορούμε να αποθηκεύουμε κάθε συμβολοσειρά των* ***n*** *bits σε χώρο* ***O(polylog(n))****, ώστε να την ανακτούμε ακέραια με πιθανότητα επιτυχίας τουλάχιστον 99%! Δεν είναι υπέροχο;*"

Φυσικά, αυτός ο ισχυρισμός σάς κάνει λίγο καχύποπτους, δεδομένου ότι γνωρίζετε πως κάτι τέτοιο είναι αδύνατο αν χρησιμοποιήσουμε έναν ντετερμινιστικό αλγόριθμο (που δηλαδή ανακτά την συμβολοσειρά με πιθανότητα επιτυχίας 100%). Αποδείξτε αυτόν τον ισχυρισμό που αφορά ντετερμινιστικούς αλγορίθμους. ***(Μονάδες: 0.125)***

Παρά ταύτα, δεδομένου ότι ο φίλος σας υπόσχεται μονάχα πιθανοτική εγγύηση, ίσως αξίζει λιγάκι να ακούσουμε την ιδέα του. Μάλιστα, με μία πρώτη ματιά, φαίνεται αρκετά εύλογη.

Η ιδέα είναι η εξής. Πολύ απλά, χρησιμοποιούμε τον δειγματολήπτη που περιγράψαμε στην προηγούμενη άσκηση, τον αρχικοποιούμε με τέτοιον τρόπο ώστε με πιθανότητα τουλάχιστον 99% να μην κάνει ούτε ένα λάθος σε **n** ερωτήματα, και τον τροφοδοτούμε με την συμβολοσειρά των **n** bits. Έπειτα, θέτουμε απανωτά ερωτήματα για μία μη-μηδενική συντεταγμένη: μετά από κάθε τέτοιο ερώτημα, επιστρέφουμε την συντεταγμένη που λάβαμε – ας την πούμε **i** – και κατευθείαν τροφοδοτούμε τον δειγματολήπτη με την εντολή **(i, -1)**, ώστε να την μηδενίσει και να μην την ξαναεπιστρέψει. Αυτό συνεχίζεται μέχρι ο δειγματολήπτης να ανακοινώσει ότι το διάνυσμά του πλέον έχει μηδενιστεί. Οπότε, αν δούμε το πράγμα συνολικά, έχουμε επιστρέψει, με πιθανότητα επιτυχίας τουλάχιστον 99%, όλες τις μη-μηδενικές θέσεις της συμβολοσειράς (και μόνον αυτές), οπότε ουσιαστικά την έχουμε ανακτήσει αυτούσια.

Δοκιμάστε να υλοποιήσετε αυτήν την ιδέα για το διάνυσμα του μέρους **α**) αυτής της άσκησης (που δηλαδή αποτελείται από 10.000 συντεταγμένες, όπου η κάθε μία είναι 0 με πιθανότητα 3/4). (Θα πρέπει να παραδώσετε και τον κώδικα με τον οποίο πραγματοποιήσατε αυτό το πείραμα.) Βλέπετε αυτή η ιδέα να δουλεύει; Αν όχι, δώστε μια εύλογη εξήγηση γιατί αποτυγχάνει, και εξηγήστε γιατί στο μέρος **α)** αυτής της άσκησης (όπου φαίνεται πως κάναμε ουσιαστικά το ίδιο, αφού και εκεί απλώς ζητούσαμε επανειλημμένα απ' τον δειγματολήπτη μία μη-μηδενική συντεταγμένη, και έπειτα μηδενίζαμε μία που ήταν μη-μηδενική) είχαμε πολύ καλύτερη πιθανότητα επιτυχίας.
(Μονάδες: 1.25)

Φυσικά, ακόμη και αν η συγκεκριμένη ιδέα αποτυγχάνει, παρά ταύτα θα μπορούσε ίσως θεωρητικά να υπάρχει ένας αλγόριθμος με τις εγγυήσεις που περιγράφει ο φίλος σας. Εσείς πιστεύετε ότι μπορεί να υπάρχει τέτοιος αλγόριθμος; Κάντε μία βιβλιογραφική έρευνα ώστε να δείτε αν αυτό είναι κάποιο γνωστό αποτέλεσμα, ή αν υπάρχει κάποιος λόγος που απαγορεύει την ύπαρξη ενός αλγορίθμου με αυτές τις εγγυήσεις. Αν σκεφτείτε αυτό το πρόβλημα μόνοι σας και καταφέρετε να αποδείξετε ότι δεν γίνεται να υπάρχει αλγόριθμος με αυτές τις εγγυήσεις, θα έχετε +1 μονάδα στην τελική βαθμολογία.
(Μονάδες: 0.375)

Ενότητα Δ: Υπολογισμός συνεκτικών συνιστωσών

Άσκηση 1: Υλοποίηση του αλγορίθμου Borůvka

Για να μπορέσουμε να υλοποιήσουμε τον αλγόριθμο που είδαμε στο μάθημα για τον υπολογισμό των συνεκτικών συνιστωσών σε γραφήματα που δίνονται με μορφή ροής δεδομένων, θα πρέπει πρώτα να είμαστε σε θέση να υλοποιήσουμε τον αλγόριθμο του Borůvka. Αυτός δεν είναι και ο πιο απλός αλγόριθμος που υπάρχει για τον υπολογισμό των συνεκτικών συνιστωσών, και χρειάζεται λίγη σκέψη για να έχουμε μια καλή υλοποίησή του.

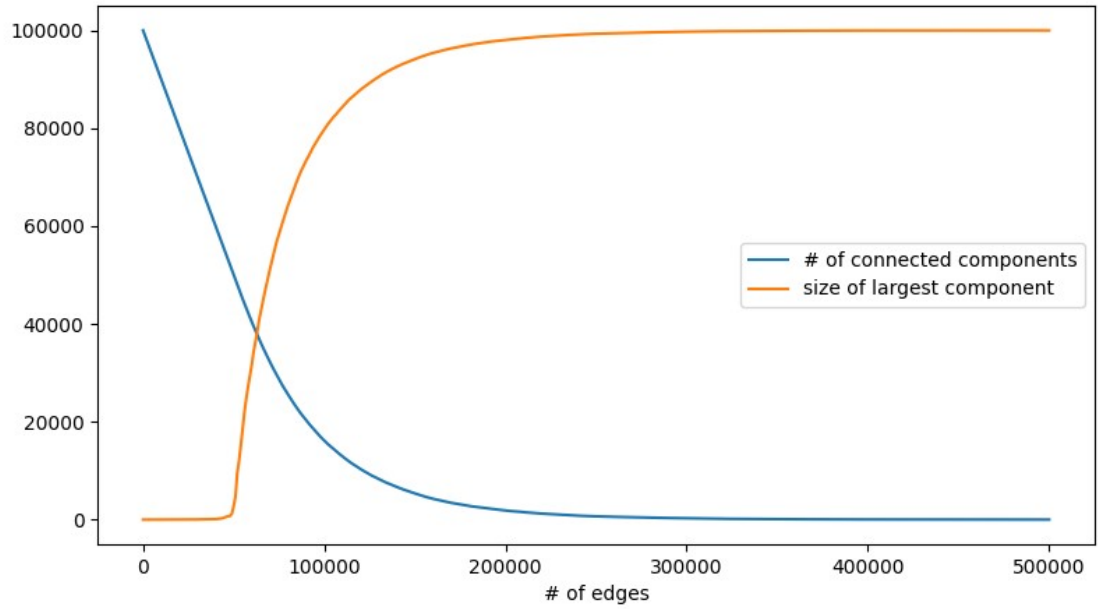
Επομένως, σε μία πρώτη φάση, δεν θα χρειαστεί να υλοποιήσουμε τον non-zero sampler που είδαμε στο μάθημα. Θα χρησιμοποιήσουμε μονάχα έναν τετριμμένο L_0-sampler, που απλώς διατηρεί αυτούσιο ένα διανύσμα (ακριβέστερα: τις μη-μηδενικές συντεταγμένες του), και, όποτε του ζητηθεί, επιλέγει εντελώς τυχαία μία μη-μηδενική συντεταγμένη (αν υπάρχει), και την επιστρέφει (ή ανακοινώνει ότι το διάνυσμα είναι το μηδενικό). Αυτή η λύση μάς κάνει για την ώρα (απλώς δεν θα μπορούσε να διαχειριστεί μία τεράστια ροή δεδομένων, με αυξημένες απαιτήσεις στην μνήμη). Οπότε η πρόκληση εδώ θα είναι μονάχα η υλοποίηση του αλγορίθμου του Borůvka.

Το γράφημά μας υποθέτουμε ότι αποτελείται από 100.000 κόμβους, και στην αρχή δεν έχει καμία ακμή. Για αρχή, θα φτιάξουμε μία ροή από 500.000 εντελώς τυχαίες εισαγωγές ακμών. Οπότε, σε κάθε βήμα, θα φτιάχνουμε ένα ζεύγος **(x,y)**, όπου τα **x** και **y** είναι ακέραιοι μεταξύ 1 και 100.000, και απλώς θα πρέπει να ισχύει **x≠y**. (Η πιθανή ύπαρξη παράλληλων ακμών στο γράφημα δεν μας πειράζει.) Κατά την διάρκεια της ροής, θα πρέπει να διατηρούμε τις λίστες πρόσπτωσης των κόμβων, με την μορφή που περιγράψαμε στο μάθημα. Οπότε, για κάθε κόμβο **v**, υπάρχει ένα διάνυσμα με 100.000^2 συντεταγμένες, αριθμημένες από το 1 μέχρι και το 100.000^2, που αναπαριστά τις ακμές που προσπίπτουν σε αυτόν (με μια κατάλληλη κωδικοποίηση). Οι τιμές σε αυτές τις συντεταγμένες είναι ακέραιοι, πιθανώς αρνητικοί. Οπότε, κάθε εισαγωγή ακμής **(x,y)** θα πρέπει να ανανεώσει τα διανύσματα πρόσπτωσης για τους κόμβους **x** και **y**. Επειδή τα διανύσματα αυτά είναι τεράστια, κρατούμε απλώς τις μη-μηδενικές συντεταγμένες τους (οι οποίες, αφού η ροή αποτελείται από 500.000 εισαγωγές, θα είναι το πολύ 500.000 – και πιθανότατα, για το διάνυσμα κάθε συγκεκριμένου κόμβου, πολύ λιγότερες). Στην επόμενη άσκηση, που θα υλοποιήσουμε τον non-zero sampler, θα κρατούμε απλώς σκιαγραφήματα αυτών των διανυσμάτων.

Θα πρέπει να προγραμματίσετε μία ρουτίνα που θα υλοποιεί τον αλγόριθμο του Borůvka, και θα δουλεύει με αυτές τις λίστες πρόσπτωσης, με τον τρόπο που θα λειτουργούσε αν είχατε έναν non-zero sampler. Επομένως, σε κάθε γύρο του αλγορίθμου Borůvka, θα διατηρείτε μία διαμέριση του συνόλου των κόμβων (ξεκινώντας από την διαμέριση όπου κάθε κόμβος συνιστά ένα μονοσύνολο), και κάθε σύνολο αυτής της διαμέρισης θα έχει το αντίστοιχο διάνυσμα πρόσπτωσής του (για την ακρίβεια: ένα σύνολο με τις μη-μηδενικές συντεταγμένες αυτού του διανύσματος), το οποίο προκύπτει από το άθροισμα των διανυσμάτων των κόμβων που το αποτελούν. Δεδομένου ενός τέτοιου διανύσματος, απλώς επιλέγουμε τυχαία μία μη-μηδενική συντεταγμένη του (ή, αν είναι μηδενικό, αναφέρουμε ότι βρήκαμε μία συνεκτική συνιστώσα).

Ανά 1000 εισαγωγές ακμών, θα εφαρμόζετε τον αλγόριθμο του Borůvka για να υπολογίζετε τις συνεκτικές συνιστώσες του τρέχοντος γραφήματος. Στην γραμμή εντολών

θα πρέπει να τυπώνετε: τον αριθμό της τρέχουσας εισαγωγής, το πλήθος των συνεκτικών συνιστωσών, και το μέγεθος της μεγαλύτερης συνεκτικής συνιστώσας (όπου το μέγεθος ορίζεται ως το πλήθος κόμβων που περιέχονται σε αυτήν). Οπότε, κάθε γραμμή που θα τυπώνετε θα έχει π.χ. την μορφή "150000: 5371 94123" (που σημαίνει ότι, μετά από 150.000 εισαγωγές ακμών, έχουμε 5371 συνεκτικές συνιστώσες, όπου η μεγαλύτερη περιέχει 94123 κόμβους). Προφανώς αυτά τα νούμερα θα εξαρτώνται από την αρχικοποίηση της συνάρτησης που παράγει την τυχαιότητα, όμως γενικά θα πρέπει να εμφανίζεται ένα ξεκάθαρο μοτίβο. Θα πρέπει να κάνετε και μία γραφική παράσταση με αυτά τα δεδομένα, η οποία θα πρέπει να μοιάζει κάπως έτσι:



Για να επαληθεύσετε την ορθότητα της υλοποίησής σας, προτείνουμε να τρέχετε παράλληλα και έναν απλό αλγόριθμο της επιλογής σας για τον υπολογισμό των συνεκτικών συνιστωσών, ώστε να βεβαιωθείτε ότι λαμβάνετε τα ίδια νούμερα για το πλήθος των συνεκτικών συνιστωσών, και της μεγαλύτερης συνεκτικής συνιστώσας. (Σημειώνουμε ότι, παρότι ο αλγόριθμος του Borůvka κάνει τυχαίες επιλογές, εντούτοις το αποτέλεσμα που θα δίνει θα πρέπει πάντοτε να είναι σωστό.) Το πιο πιθανό είναι ότι ο δικός σας αλγόριθμος θα τρέχει πιο γρήγορα από τον αλγόριθμο του Borůvka. Αυτό είναι το πιο αναμενόμενο. (Σε αντίθετη περίπτωση, προτείνουμε να προβληματιστείτε λιγάκι.) Όμως, αν η υλοποίηση που κάνατε για τον αλγόριθμο του Borůvka καθυστερεί υπερβολικά πολύ (σε σημείο π.χ. που μπορεί να χρειάζεται αρκετά λεπτά για να επεξεργαστεί όλη την ροή), τότε μάλλον δεν κάνατε αποδοτική υλοποίησή του. (Ήτοι, μπορεί η υλοποίησή σας να είναι σωστή, αλλά κάποια πράγματα θα μπορούσατε να τα είχατε κάνει και πιο έξυπνα, και θα χάσετε μερικές μονάδες αν δεν έχετε κάνει μια αποδοτική υλοποίηση.)

Έπειτα, θα δοκιμάσουμε ροές που περιλαμβάνουν και διαγραφές ακμών. Συγκεκριμένα, η ροή θα αποτελείται από 5.000.000 εντολές (εισαγωγές/διαγραφές). Σε κάθε βήμα, επιλέγουμε με μία πιθανότητα **p** αν θα διαγράψουμε ακμή. Αν αποφασίσουμε να διαγράψουμε ακμή, τότε δοκιμάζουμε να διαγράψουμε μία με τον εξής τρόπο. Επιλέγουμε εντελώς τυχαία έναν κόμβο. Αν αυτός ο κόμβος έχει προσπίπτουσα ακμή, τότε επιλέγουμε εντελώς τυχαία μία από αυτές και την διαγράφουμε. Ειδάλλως, αν τύχει ο κόμβος που

επιλέξαμε να μην έχει προσπίπτουσα ακμή, τότε, αντί για διαγραφή, θα πραγματοποιήσουμε απλώς εισαγωγή. Οι εισαγωγές ακμών γίνονται όπως και πριν. Θα δοκιμάσουμε αυτό το ρεπερτόριο για **p=3/4** και **p=9/10**, και θα υπολογίζουμε τις συνεκτικές συνιστώσες κάθε 10.000 βήματα. Θα πρέπει και πάλι το πρόγραμμά μας να τυπώνει στην γραμμή εντολών τα αντίστοιχα δεδομένα όπως και πριν, και θα σχεδιάσουμε και πάλι τις σχετικές γραφικές παραστάσεις.
(Μονάδες: 2.125)

Στην προηγούμενη περίπτωση των 500.000 εισαγωγών τυχαίων ακμών, αναμένεται, όπως αναφέραμε, να λαμβάνετε μία γραφική παράσταση όπως και εκείνη που έχουμε στο παραπάνω σχήμα. Φυσικά, οι ακριβείς τιμές εξαρτώνται από την αρχικοποίηση που κάνατε στην γεννήτρια τυχαίων αριθμών, αλλά αυτό το μοτίβο θα δείτε να εμφανίζεται συστηματικά. Πώς το εξηγείτε αυτό; Ειδικότερα, διατυπώστε μία εύλογη θεωρία σε σχέση με το γιατί, όσον αφορά το μέγεθος της μεγαλύτερης συνεκτικής συνιστώσας, φαίνεται να υπάρχει αρχικά μία σύντομη περίοδος σταθερότητας, που ακολουθείται από μία απότομη αύξηση, και σταθεροποίηση τελικά στο πλήθος όλων των κόμβων σχεδόν του γραφήματος.
(Μονάδες: 0.375)

Άσκηση 2: Υλοποίηση του πλήρους αλγορίθμου (με χρήση non-zero sampler)

Αφού μπορέσατε να δώσετε μια καλή υλοποίηση του αλγορίθμου Borůvka, τώρα θα πρέπει να υλοποιήσετε τον πλήρη αλγόριθμο που υπολογίζει τις συνεκτικές συνιστώσες στο μοντέλο ροής δεδομένων, χρησιμοποιώντας τον non-zero sampler που περιγράψαμε στον μάθημα.

Θα εργαστείτε με τις ίδιες ροές εντολών όπως και στην προηγούμενη άσκηση (δηλαδή και πάλι έχουμε να κάνουμε με ένα γράφημα 100.000 κόμβων που αρχικά είναι κενό, και πρώτα θα δοκιμάσουμε εισαγωγές 500.000 τυχαίων ακμών, και έπειτα τα δύο ρεπερτόρια 5.000.000 εισαγωγών/διαγραφών που περιγράψαμε). Εδώ θα χρειαστεί να τυπώνετε και πάλι τα ίδια δεδομένα στην γραμμή εντολών (ανά 1000 εισαγωγές στην πρώτη περίπτωση, και ανά 10.000 εντολές (εισαγωγές/διαγραφές) στην δεύτερη), αλλά δεν χρειάζεται να κάνετε τις αντίστοιχες γραφικές παραστάσεις.

Θα είναι δική σας δουλειά να καθορίσετε κατάλληλα τις παραμέτρους του non-zero sampler, ώστε να εξασφαλίζει με πιθανότητα τουλάχιστον 99% ότι ο υπολογισμός των συνεκτικών συνιστωσών δεν θα αποτύχει ούτε μία φορά στις 500 που θα πραγματοποιηθεί.
(Μονάδες: 2.5)

Για μια καλή επιλογή των παραμέτρων, σας δίνονται 0.5 από τις παραπάνω μονάδες (ανεξάρτητα απ' το αν θα καταφέρετε να υλοποιήσετε κάτι). Από την άλλη, αν καταφέρετε να υλοποιήσετε σωστά τον όλο αλγόριθμο, θα κερδίσετε τις 2 μονάδες (συγκεκριμένα: ο non-zero sampler από μόνος του δίνει 1.25 μονάδες, και η ενσωμάτωσή του στον πλήρη αλγόριθμο για τον υπολογισμό των συνεκτικών συνιστωσών δίνει 0.75), ανεξάρτητα από το αν ακολουθήσατε πολύ πιστά την θεωρητική ανάλυση για την επιλογή των παραμέτρων ώστε να εξασφαλιστούν οι επιθυμητές πιθανοτικές εγγυήσεις. Εξ άλλου, όπως είδαμε και σε προηγούμενες ασκήσεις, οι επιλογές που προτείνει η θεωρία φαίνεται να είναι αρκετά απαισιόδοξες στην πράξη.