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)