Αλγοριθμικές Τεχνικές για Δεδομένα Ευρείας Κλίμακας

1ο Φυλλάδιο Ασκήσεων

Προθεσμία υποβολής: 10/04/2025, 23:59 (ώρα Ελλάδος)

Ενότητα Α: Μετρητές Morris

Θεωρούμε έναν μετρητή τύπου Morris που εκτιμά το πλήθος στοιχείων n σε μια ροή δεδομένων (εκτιμώντας το πλήθος των bits στην δυαδική αναπαράσταση του n), ο οποίος περιγράφεται από τον εξής ψευδοκώδικα:

```
integer C ← 0
void insert()
{
   C ← C+1, with probability 1/2^C
}
```

```
integer query()
{
   return 2^C - 1
}
```

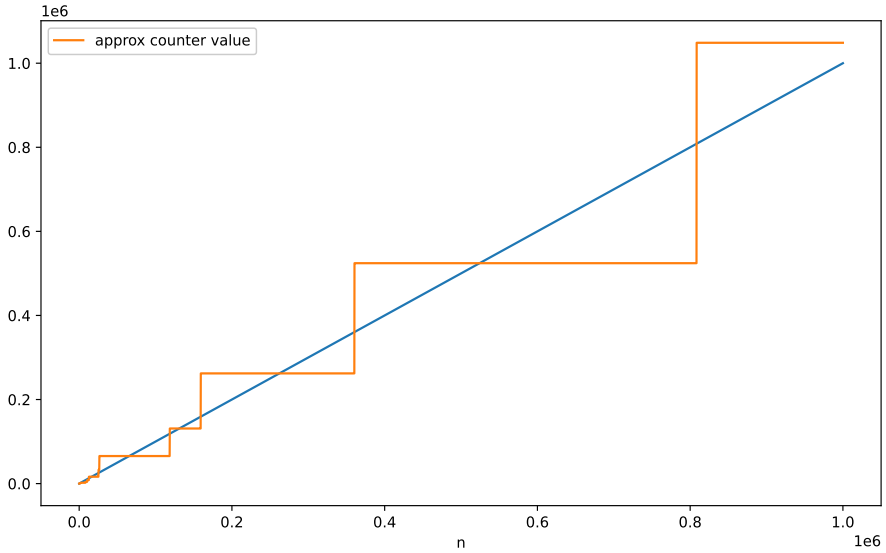
Ορίζουμε το C_n να είναι η τυχαία μεταβλητή που η τιμή της είναι η τιμή της μεταβλητής C μετά από n εισαγωγές στοιχείων.
(Παρατηρήστε ότι $C_n \in \{1, 2, \ldots, n\}$.)

Άσκηση 1

α) *(Μονάδες: 0.75)*
Υλοποιήστε τον παραπάνω μετρητή Morris, και τρέξτε τον για 1.000.000 εισαγωγές στοιχείων. Έπειτα από κάθε εισαγωγή, εκτυπώστε την εκτίμηση του μετρητή (δηλαδή το $2^C - 1$).

Σχεδιάστε μία αντίστοιχη γραφική παράσταση που δείχνει τις τιμές του μετρητή καθώς γινόντουσαν οι εισαγωγές. Το αποτέλεσμά σας θα πρέπει να είναι όπως φαίνεται στην ακόλουθη εικόνα*:

*προφανώς η γραφική παράσταση θα ποικίλλει ανάλογα με την αρχικοποίηση που κάνατε στην συνάρτηση που παράγει την τυχαιότητα



Παράδειγμα εκτιμήσεων του μετρητή Morris για το πλήθος εισαγωγών n, όσο το n αυξάνεται από 1 μέχρι και 1.000.000.

β1) *(Μονάδες: 0.75)*
Μπορούμε να βελτιώσουμε τον παραπάνω μετρητή ως εξής. Αντί να κρατούμε μόνο μία μεταβλητή C, μπορούμε να κρατούμε αρκετές τέτοιες μεταβλητές (οι οποίες ανανεώνονται ανεξάρτητα), και να επιστρέφουμε τον μέσο όρο ή τον διάμεσο από το αντίστοιχο αποτέλεσμα που δίνουν. (Π.χ., αν κρατούμε τις μεταβλητές C1, C2, C3, τότε επιστρέφουμε ως εκτίμηση του n το $(2^{C1} + 2^{C2} + 2^{C3} - 3)/3$.)

Υλοποιήστε αυτήν την ιδέα, για τον μέσο όρο και για τον διάμεσο, χρησιμοποιώντας 5 ανεξάρτητες μεταβλητές, και κάντε τις αντίστοιχες γραφικές παραστάσεις όπως και στο α).

β2) *(Μονάδες: 0.25)*
Δοκιμάστε να τρέξετε αρκετές φορές τις υλοποιήσεις που κάνατε στο β1 (χρησιμοποιώντας κάθε φορά νέα τυχαιότητα). Θεωρείτε ότι είναι καλύτερο να επιστρέφετε τον μέσο όρο ή τον διάμεσο; (Ίσως χρειαστεί να προσθέσετε περισσότερες μεταβλητές C για να παρατηρήσετε ένα πιο ξεκάθαρο μοτίβο.)

Προφανώς είναι ανούσιο στην πράξη να κρατά κανείς πέντε διαφορετικές μεταβλητές C για να εκτιμήσει το πλήθος στοιχείων όταν το n φτάνει μέχρι το 1.000.000. Πρώτον, τεκμηριώστε αυτόν τον ισχυρισμό. Δεύτερον, για ποια n θα είχε νόημα να κρατά κανείς 5 μεταβλητές C, δεδομένου ότι θέλουμε να έχουμε κέρδος στον χώρο; (Θεωρήστε ότι μόνο οι μεταβλητές C είναι αυτές που κοστίζουν χώρο μνήμης.)

(Υπόδειξη: χρειαζόμαστε $\lceil \log_2(n+1) \rceil$ bits για να αναπαραστήσουμε αριθμούς από το 0 μέχρι και το n.)

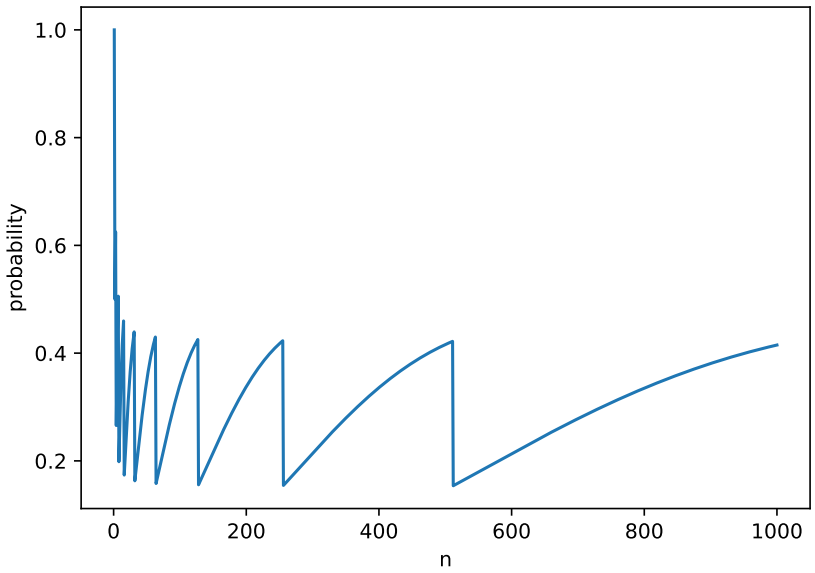
Άσκηση 2

α) *(Μονάδες: 0.125)*

Υπολογίστε με απόλυτη ακρίβεια την πιθανότητα η C_{1000} να έχει τιμές στο σύνολο {8, 9, 10, 11}. Δηλαδή, αυτή η πιθανότητα θα πρέπει να υπολογιστεί ως κλάσμα ακεραίων. Να αναφέρετε το αποτέλεσμα σε δεκαδική μορφή (με ακρίβεια τουλάχιστον 5 δεκαδικών ψηφίων), και υποβάλετε τον κώδικα με τον οποίο υπολογίσατε αυτήν την πιθανότητα.

β) *(Μονάδες: 0.125)*

Υπολογίστε τις πιθανότητες $P(C_k = \lceil \log_2(k+1) \rceil)$, για $k \in \{1, \ldots, 1000\}$. Σχεδιάστε μια γραφική παράσταση που αναπαριστά αυτές τις πιθανότητες. Το αποτελεσμά σας θα πρέπει να είναι κάπως έτσι:



Θα πρέπει να υποβάλετε τον κώδικα με τον οποίο υπολογίσατε τις πιθανότητες, και τον κώδικα με τον οποίο φτιάξατε την γραφική παράσταση.

γ) *(Μονάδες: 0.125)*

Κάντε ακριβώς ό,τι και στο β), μόνο που αυτήν την φορά θα πρέπει να υπολογίσετε τις πιθανότητες $P(C_k \in \{l(k)-1, l(k), l(k)+1\})$, όπου $l(k) = \lceil \log_2(k+1) \rceil$, για $k \in \{1, 2, \ldots, 1000\}$.

Αναφέρετε κάποιο χρήσιμο συμπέρασμα που βγαίνει από αυτήν την γραφική παράσταση, ειδικά αν την συγκρίνουμε με αυτήν που λάβαμε στο β).

δ) *(Μονάδες: 0.625)*
Μέχρι τώρα υπολογίσαμε ορισμένες πιθανότητες της μορφής $P(C_k = i)$. Αυτό μας λέει την πιθανότητα, αν γίνουν k εισαγωγές στοιχείων, η μεταβλητή C_k (που θέλουμε να είναι κοντά στο $\lceil \log_2(k+1) \rceil$) να έχει την τιμή i. Αυτή η πιθανότητα όμως δεν μας λέει ακριβώς τι γινόταν *καθόλη την πορεία των εισαγωγών*. Δηλαδή, μπορεί μεν η C_k να τύχει να είναι κοντά στο $\lceil \log_2(k+1) \rceil$, όμως μπορεί για ορισμένα μικρότερα k' η C να ήταν πολύ μακριά απ' το $\lceil \log_2(k'+1) \rceil$.

Εργαστείτε όπως και στα β) και γ), μόνο που αυτήν την φορά υπολογίστε την πιθανότητα:
1) η C_k να είναι ακριβώς $l(k)$ *για όλα τα* $k \in \{1, 2, \ldots, 1000\}$, και
2) η C_k να ανήκει στο σύνολο $\{l(k)-1, l(k), l(k)+1\}$, *για όλα τα* $k \in \{1, 2, \ldots, 1000\}$,
όπου $l(k) = \lceil \log_2(k+1) \rceil$.

Άσκηση 3 *(Μονάδες: 0.75)*

Έστω ότι θέλουμε έναν μετρητή τύπου Morris για να μετρήσουμε μέχρι το 1.000.000, και έστω ότι έχουμε 8 bits μνήμης στην διάθεσή μας. Παρατηρήστε ότι αυτά είναι περισσότερα απ' όσα χρειαζόμαστε (και τεκμηριώστε αυτόν τον ισχυρισμό). Θα θέλαμε λοιπόν να εκμεταλλευτούμε όλα τα διαθέσιμα bits.

Για τον σκοπό αυτό, μπορούμε να κάνουμε την εξής τροποποίηση στον αλγόριθμο. Αντί να αυξάνουμε το C με πιθανότητα $1/2^C$, μπορούμε να το αυξάνουμε με πιθανότητα $1/\alpha^C$, για κάποιο α με $1 \leq \alpha \leq 2$. (Παρατηρήστε ότι για $\alpha = 1$ έχουμε απλώς κανονική καταμέτρηση, με απόλυτη ακρίβεια.) Έπειτα, μια εύλογη επιλογή είναι να επιστρέψουμε ως εκτίμηση του n το $(1/(\alpha - 1)) \cdot (\alpha^C - 1)$ (γιατί;).

Βρείτε ένα κατάλληλο α ώστε να αξιοποιηθούν όσο το δυνατόν περισσότερο τα 8 bits, φτιάξτε μια γραφική παράσταση όπως και στην Άσκηση 1, και εξετάστε αν όντως υπάρχει κάποια βελτίωση στην ποιότητα της προσέγγισης.

Ενότητα Β: Καταμέτρηση διακεκριμένων στοιχείων

Άσκηση 1

α) (Μονάδες: 0.875)
Ο σκοπός είναι να υλοποιήσουμε τον αλγόριθμο προσεγγιστικής καταμέτρησης διακεκριμένων στοιχείων που φαίνεται στον ακόλουθο ψευδοκώδικα:

```
integer R ← 0
let h: [0,...,2^d-1] → [0,...,2^d-1] be a hash function
void insert(integer x)
{
   let r be the number of trailing zeroes of h(x)
   if (r ≥ R)
   {
      R ← r
   }
}
```

```
integer query()
{
   return 2^R
}
```

Για αρχή, θα δημιουργήσουμε μία ροή εντελώς τυχαίων στοιχείων, και δεν θα χρησιμοποιήσουμε καμία συνάρτηση κατακερματισμού (ή, αν θέλετε, η συνάρτηση κατακερματισμού h θα είναι η h(x) = x).

Συγκεκριμένα, δημιουργήστε μία ροή 1.000.000 εντελώς τυχαίων αριθμών από το 0 μέχρι και το 1.000.000. Αποδείξτε ότι το αναμενόμενο πλήθος των διακεκριμένων αριθμών που θα έχουν εμφανιστεί μέχρι το τέλος της ροής είναι ~630.000.

Το πρόγραμμά σας θα εκτυπώνει μετά από κάθε παραγωγή νέου στοιχείου: 1) το πόσα στοιχεία έχουν παραχθεί μέχρι τώρα, 2) το πραγματικό πλήθος διακεκριμένων στοιχείων μέχρι τώρα, και 3) την προσέγγιση που δίνει ο αλγόριθμος που περιγράψαμε.

Για παράδειγμα, την στιγμή που θα έχουν παραχθεί 100.000 στοιχεία, το πρόγραμμα θα εκτυπώνει π.χ. την γραμμή: 100000 63212 65536.

Προκειμένου να υλοποιηθεί αποδοτικά το 2) θα πρέπει είτε να χρησιμοποιήσετε κάποια έτοιμη δομή δεδομένων, είτε να υλοποιήσετε εσείς μία (π.χ., προτείνουμε να αποθηκεύετε τα διακεκριμένα στοιχεία σε ένα trie). Η επιτυχής υλοποίηση του 2) πιάνει συνολικά 0.25 μονάδες,

αλλά οι 0.125 από αυτές δίνονται μόνο αν υλοποιήσετε εσείς την δική σας δομή δεδομένων.

β) *(Μονάδες: 0.875)*
Εδώ θα κάνουμε ό,τι και στο α), μόνο που θα αλλάξουμε την κατανομή εισόδου, και επίσης θα χρησιμοποιήσουμε μία (τυχαία επιλεγμένη) συνάρτηση κατακερματισμού.

Συγκεκριμένα, τώρα θα παράγουμε 1.000.000 αριθμούς κατασκευάζοντας συμβολοσειρές των 20 bits ως εξής. Κάθε ένα από τα 5 πρώτα bits είναι 1 με πιθανότητα 1/4. Κάθε ένα από τα bits 6 μέχρι και 10 είναι 1 με πιθανότητα 1/8. Κάθε ένα από τα bits 11 μέχρι και 15 είναι 1 με πιθανότητα 1/16. Και κάθε ένα από τα bits 16 μέχρι και 20 είναι 1 με πιθανότητα 1/32.

Πρώτα κάντε ό,τι και στο α) (με αυτήν εδώ την κατανομή εισόδου). Ίσως παρατηρείτε ότι η προσέγγιση του πλήθους διακεκριμένων στοιχείων δεν είναι καθόλου καλή. Δώστε μία εξήγηση για αυτό.

Τώρα επαναλάβετε αυτήν την διαδικασία, χρησιμοποιώντας μία συνάρτηση κατακερματισμού h που ορίζεται ως εξής. Έστω $p = 1048583$. (Ο p είναι ο μικρότερος πρώτος αριθμός που είναι μεγαλύτερος από το 2^{20}.) Επιλέξτε τυχαία έναν αριθμό α μεταξύ 1 και $p - 1$, και έναν αριθμό β μεταξύ 0 και $p - 1$. Τότε, το $h(x)$ ορίζεται ως $h(x) = (\alpha x + \beta) \bmod p$.

Άσκηση 2 *(Μονάδες: 1.75)*
Εδώ θα υλοποιήσουμε έναν διαφορετικό αλγόριθμο για την προσεγγιστική καταμέτρηση διακεκριμένων στοιχείων, που δίνει καλύτερες εγγυήσεις από αυτόν που περιγράψαμε στην Άσκηση 1.

Συγκεκριμένα, θα εφαρμόσουμε τον αλγόριθμο 1 που περιγράφεται στην εργασία "Counting distinct elements in a data stream" των Bar-Yossef, Jayram, Kumar, Sivakumar, και Trevisan (BJKST), RANDOM, 2002. Ο αλγόριθμος αυτός λαμβάνει ως είσοδο ένα ε, και δίνει μία εκτίμηση N του πλήθους F_0 των διακεκριμένων στοιχείων σε μια ροή στοιχείων, η οποία ικανοποιεί την ανισότητα $(1 - \varepsilon)F_0 \leq N \leq (1 + \varepsilon)F_0$, με πιθανότητα τουλάχιστον $2/3 - 1/m$, χρησιμοποιώντας χώρο $O((1/\varepsilon^2)\log(m))$, όπου m είναι ο μέγιστος αριθμός που μπορεί να εμφανιστεί στην ροή. (Αρχικά εξηγήστε από ποιες απόψεις αυτές οι εγγυήσεις είναι καλύτερες από εκείνες που γνωρίζετε για τον αλγόριθμο που περιγράψαμε στην Άσκηση 1.)

Θα εφαρμόσετε αυτόν τον αλγόριθμο και στις δύο κατανομές εισόδου που περιγράφονται στα α) και β) της Άσκησης 1, και θα εργαστείτε ακριβώς όπως σε αυτές τις ασκήσεις, απλώς χρησιμοποιώντας τον BJKST αλγόριθμο.

Ο αλγόριθμος BJKST περιγράφεται από τα εξής βήματα.

1) Επιλέγουμε μία συνάρτηση κατακερματισμού $h : [1,...,m] \rightarrow [1,...,M]$, όπου $M = m^3$.

2) Θέτουμε $t = \lceil 96/\varepsilon^2 \rceil$.

3) Για κάθε x που εμφανίζεται στην ροή, υπολογίζουμε την τιμή h(x). Διατηρούμε (με μια αποδοτική δομή δεδομένων) τις t μικρότερες διακεκριμένες τιμές που συναντήσαμε, καθόλη την διάρκεια της ροής.

4) Ως εκτίμηση του πλήθους διακεκριμένων στοιχείων, επιστρέφουμε την τιμή $N = tM/v$, όπου v είναι η t-οστή μικρότερη διακεκριμένη τιμή που έχουμε αποθηκεύσει. (Ας αρχίσουμε να εκτυπώνουμε το N μονάχα απ' την στιγμή που θα έχουμε συγκεντρώσει τουλάχιστον t διακεκριμένες τιμές.)

Η συνάρτηση κατακερματισμού θα έχει την ίδια μορφή όπως και στο β) της Άσκησης 1, μόνο που το p εδώ θα είναι το $q_1 \cdot q_2 \cdot q_3$, όπου $q_1 = 100.003$, $q_2 = 100.019$, και $q_3 = 100.043$, είναι ο πρώτος, δεύτερος, και τρίτος μικρότερος πρώτος αριθμός μεγαλύτερος του 100.000, αντίστοιχα. (Τα α και β θα επιλέγονται τυχαία μεταξύ 1 και $p - 1$, και 0 και $p - 1$, αντίστοιχα.) Ως αποτέλεσμα h(x) θα επιστρέφεται η τιμή
$((\alpha x + \beta) \bmod p) \bmod m$.

Υλοποιήσετε τον αλγόριθμο για $\varepsilon=0.1$ και $\varepsilon=0.01$, και τρέξτε τον στις κατανομές εισόδου που περιγράψαμε.

Έπειτα, βελτιώστε την πιθανοτική εγγύηση του αλγορίθμου, εφαρμόζοντας την τεχνική του διαμέσου. Πρώτα δοκιμάστε 101 επαναλήψεις του αλγορίθμου, και έπειτα χρησιμοποιήστε έναν κατάλληλο αριθμό επαναλήψεων ώστε (ακολουθώντας μια θεωρητική ανάλυση) να αυξήσετε την πιθανότητα επιτυχίας στο 99.9%. Αν και πάλι δεν βλέπετε η προσεγγιστική τιμή N να ικανοποιεί την ανισότητα $(1 - \varepsilon)F_0 \leq N \leq (1 + \varepsilon)F_0$, μην απογοητευτείτε πολύ. Δώστε μία εύλογη εξήγηση για αυτήν την αποτυχία.

Ενότητα Γ: Φίλτρα Bloom

Άσκηση 1: Υπολογισμός τομής

Έστω ότι έχουμε μία συλλογή από τεράστια αρχεία που βρίσκονται κατανεμημένα σε δύο servers A και B. Ορισμένα αρχεία περιέχονται και στους δύο servers (ως ακριβή αντίγραφα), και θα θέλαμε να τα βρούμε. Μας δίνεται ως πληροφορία ότι αυτά τα κοινά αρχεία είναι πολύ λιγότερα απ' όσα περιέχονται στους A και B. Η επικοινωνία μεταξύ των servers κοστίζει πολύ, οπότε θα θέλαμε να υπολογίσουμε τα κοινά αρχεία των A και B ελαχιστοποιώντας την επικοινωνία μεταξύ τους. Για τον σκοπό αυτό, μπορούμε να χρησιμοποιήσουμε φίλτρα Bloom ως εξής.

Ο A θα υπολογίσει ένα φίλτρο Bloom για τα αρχεία που περιέχονται σε αυτόν, και θα το στείλει στον B για να τσεκάρει ποια από τα δικά του βρίσκονται στον A. Έπειτα, ο B θα στείλει στον A αυτά που πέρασαν το φίλτρο (δηλαδή αυτά που το φίλτρο ανέφερε ότι ανήκουν στον A), και ο A θα τσεκάρει ποια από αυτά ανήκουν όντως σε αυτόν. (Υποθέτουμε ότι ο A μπορεί να κάνει γρήγορο έλεγχο ύπαρξης, έχοντας οργανώσει τα αρχεία του σε μια αποδοτική δομή δεδομένων, π.χ., σε tries, ή, έχοντας ταξινομήσει τα αρχεία του, κάνοντας δυαδική αναζήτηση.)

Ας πούμε ότι η παραπάνω διαδικασία είναι "ένας γύρος επικοινωνίας". Προφανώς κάποια αρχεία που ο B έστειλε στον A δεν ανήκουν στον A, αλλά στάλθηκαν λόγω του false positive bias του φίλτρου Bloom. Παρά ταύτα, κατά πάσα πιθανότητα πετύχαμε κάτι καλύτερο απ' το να στέλναμε όλα τα αρχεία του B στον A.
Υπάρχει όμως το ενδεχόμενο να μπορούμε να κάνουμε και κάτι πολύ καλύτερο. Αντί ο B στον πρώτο γύρο να στείλει τα αρχεία του, θα στείλει ένα δικό του φίλτρο Bloom στον A, που σχηματίζεται από τα αρχεία του B που πέρασαν από το φίλτρο του A. Έπειτα ο A θα στείλει ένα καινούριο φίλτρο στον B με βάση αυτό που του έστειλε ο B, και τώρα ο B θα στείλει στον A όσα αρχεία περνούν από αυτό το δεύτερο φίλτρο, όπως και πριν. Ας πούμε ότι αυτή η διαδικασία συνιστά "δύο γύρους επικοινωνίας".

Αυτό το σχήμα επικοινωνίας γενικεύεται άμεσα σε "k γύρους επικοινωνίας".

α) *(Μονάδες: 1.75)*
Υλοποιήστε την παραπάνω ιδέα ως εξής.

Πρώτα απ' όλα, θα φροντίσουμε οι Α και Β να περιέχουν 100.000 αρχεία ο καθένας, αλλά μόνο 10 θα είναι κοινά σε αυτούς. Τα "αρχεία" για εμάς θα είναι απλώς συμβολοσειρές των 100 bits.

Για να υλοποιήσουμε αυτήν την κατάσταση, πρώτα θα φτιάξουμε 10 τυχαία αρχεία. (Δηλαδή, 10 εντελώς τυχαίες συμβολοσειρές των 100 bits.) Έπειτα θα φτιάξουμε 99.990 τυχαία αρχεία για τον Α, 99.990 τυχαία αρχεία για τον Β, και θα προσθέσουμε στους Α και Β επιπλέον τα 10 αρχεία που φτιάξαμε στην αρχή.

Ισχυριζόμαστε ότι αυτή η διαδικασία εξασφαλίζει με πρακτικά απόλυτη βεβαιότητα ότι μόνο τα 10 αρχεία που φτιάξαμε στην αρχή θα ανήκουν στους Α και Β ταυτόχρονα. Τεκμηριώστε αυτόν τον ισχυρισμό, εκτιμώντας την πιθανότητα τα κοινά αρχεία στους Α και Β, μετά από την όλη αυτή διαδικασία, να είναι πάνω από 10. *(Μονάδες: 0.125).*

Κάθε συνάρτηση κατακερματισμού που θα χρησιμοποιήσουμε για τα φίλτρα Bloom κατασκευάζεται ως εξής. Έστω $p = 1048583$, και έστω N το πλήθος των bits του φίλτρου Bloom. (Ο p είναι ο μικρότερος πρώτος αριθμός που είναι μεγαλύτερος από το 2^{20}.) Αρχικά, επιλέγουμε τυχαία έναν αριθμό α μεταξύ 1 και $p - 1$, και έναν τυχαίο αριθμό β μεταξύ 0 και $p - 1$. Αυτές οι τέσσερις παράμετροι (α, β, p, N) καθορίζουν μια συνάρτηση κατακερματισμού h. Έστω τώρα ένα αρχείο X, και ο σκοπός είναι να περιγράψουμε πώς υπολογίζεται το h(X). Αρχικά, χωρίζουμε το X σε 5 συνεχόμενα κομμάτια B_1, B_2, B_3, B_4, B_5 των 20 bits. Θεωρούμε κάθε ένα από αυτά τα κομμάτια ως έναν αριθμό μεταξύ 0 και $2^{20} - 1$, που η δυαδική του αναπαράσταση δίνεται από τα bits που συνιστούν το κομμάτι. (Π.χ., ένα κομμάτι 00000000000000001101 ερμηνεύεται ως ο αριθμός 13.) Αρχικοποιούμε μια τιμή $c_0 \leftarrow 0$, και υπολογίζουμε αναδρομικά την τιμή $c_i = (\alpha(B_i + c_{i-1}) + \beta) \bmod p$, για κάθε $i \in \{1,2,3,4,5\}$. Τελικά, επιστρέφουμε το $c_5 \bmod N$.

Για αρχή, ας υποθέσουμε ότι $N = 500.000$. (Οπότε, κάθε φίλτρο Bloom που στέλνεται από τον Α στον Β, ή αντίστροφα, αποτελείται από 500.000 bits. Παρατηρήστε όμως ότι ο κάθε ένας από τους Α και Β θα πρέπει να στέλνει στον άλλον και τις συναρτήσεις κατακερματισμού που χρησιμοποίησε.)

Υλοποιήστε την μέθοδο που περιγράψαμε για τον υπολογισμό των κοινών αρχείων των Α και Β. Το πρόγραμμά σας αρκεί να εκτυπώνει μονάχα το

πλήθος των positives που υπολόγισε ο Β στο τελευταίο φίλτρο Bloom που του έστειλε ο Α. Ας ονομάσουμε αυτήν την παράμετρο ύψιστου ενδιαφέροντος **count**. (Όμως τσεκάρετε κι όλας ότι στα αρχεία που ο Β πήρε positive στο τέλος περιλαμβάνονται τα 10 που ανήκουν και στον Α, και εκτυπώστε "true" σε αυτήν την περίπτωση. *(Μονάδες για την υλοποίηση αυτού του ελέγχου: 0.25)*)

Η υλοποίησή σας θα πρέπει να αφήνει την ελευθερία στον χρήστη να επιλέξει: 1) το πλήθος των συναρτήσεων κατακερματισμού (που θα αφορά όλα τα φίλτρα Bloom που θα δημιουργηθούν), και 2) το πλήθος των γύρων επικοινωνίας.

Επιπλέον, δώστε σαν επιλογή, αντί όλα τα φίλτρα Bloom να αποτελούνται από 500.000 bits, το πλήθος των bits να αλλάζει με βάση το πόσα αρχεία εκτίμησε ο ένας να περιέχονται στον άλλον. Συγκεκριμένα, αν π.χ. πρόκειται να έχουμε πάνω από έναν γύρο επικοινωνίας, τότε ο Β, αν στον πρώτο γύρο επικοινωνίας βρει ότι N_1 αρχεία του περνούν το φίλτρο που του έστειλε ο Α, τότε θα στείλει ένα φίλτρο Bloom στον Α με $5N_1$ bits. Έπειτα, αν ο Α βρει N_2 αρχεία του να περνούν το φίλτρο που του έστειλε το Β, τότε στέλνει ένα φίλτρο Bloom στον Β με $5N_2$ bits, κ.ο.κ. Θεωρείτε ότι αυτό θα βελτιώσει ή θα χειροτερέψει το αποτέλεσμα **count**; (Για το πρόβλημά μας, προφανώς το "χειροτέρεμα" είναι το **count** να αυξάνεται.)

Δοκιμάστε τις εξής παραλλαγές (σε όλους τους δυνατούς συνδυασμούς): i) το πλήθος των συναρτήσεων κατακερματισμού να είναι 1, 2, 3 ,5, 10, και ii) το πλήθος των γύρων επικοινωνίας να είναι 1, 2, 3, 4, 5. Για ποιον συνδυασμό βλέπετε να έχουμε το βέλτιστο **count**;

β) *(Μονάδες: 0.25)*
Ίσως παρατηρήσατε (ή υποψιάζεστε) ότι όσο το πλήθος των γύρων επικοινωνίας αυξάνεται, τόσο περισσότερο αναμένεται να βελτιώνεται (δηλαδή να μειώνεται) το πλήθος των αρχείων που πρέπει τελικά ο Β να στείλει στον Α (δηλαδή το **count**). Δώστε μία πειστική μαθηματική εξήγηση για αυτήν την παρατήρηση (ή υποψία), ή καταρρίψτε αυτόν τον ισχυρισμό.

Άσκηση 2: Ιχνηλάτηση πακέτου (Packet Tracing)

Έστω ότι έχουμε ένα δίκτυο από routerάκια, τα οποία προωθούν μηνύματα το ένα στο άλλο. Ανά πάσα ώρα και στιγμή, ένα routerάκι

μπορεί να δημιουργήσει ένα μήνυμα, το οποίο θα το στείλει σε κάποιο από τα routerάκια με τα οποία είναι συνδεδεμένο ως πομπός. (Το ποιος μπορεί να στείλει μήνυμα σε ποιον εξαρτάται από μία προκαθορισμένη τοπολογία.) Ένα routerάκι που δέχεται μήνυμα, μπορεί να αναρωτιέται ποιος ήταν αυτός που το δημιούργησε αρχικά. Σε αυτήν την περίπτωση, μπορεί να ρωτήσει ένα-ένα τα routerάκια τα οποία συνδέονται με αυτό (ως πομποί του), ώστε να μάθει ποιο του έστειλε το μήνυμα, και έπειτα αυτό να ρωτήσει με την σειρά του τους πομπούς του. Το πρόβλημα είναι ότι τα routerάκια έχουν πολύ περιορισμένη μνήμη, οπότε δεν μπορούν να κρατούν αυτούσια τα μηνύματα που δέχονται.

Σε ένα τέτοιο σενάριο, τα φίλτρα Bloom μπορούν να φανούν πολύ χρήσιμα. Υποθέτουμε ότι κάθε routerάκι κρατάει ένα φίλτρο Bloom για τα μηνύματα που δέχτηκε, και άρα η ιχνηλάτηση του μηνύματος θα γίνει χρησιμοποιώντας τα φίλτρα Bloom που είναι αποθηκευμένα στα routerάκια. Παρατηρήστε ότι αυτό θα κάνει ορισμένα routerάκια να ισχυρίζονται ότι ένα μήνυμα πέρασε από αυτά, ή δημιουργήθηκε από αυτά, χωρίς να ισχύει κάτι τέτοιο. Αν όμως καταλήξουμε σε μοναδικό routerάκι που ισχυρίζεται ότι δημιούργησε το μήνυμα, τότε θεωρούμε την ιχνηλάτηση πετυχημένη.

α) *(Μονάδες: 2.25)*
Ο σκοπός είναι να προσομοιώσουμε την διαδικασία προώθησης μηνυμάτων και ιχνηλάτησής τους, χρησιμοποιώντας φίλτρα Bloom, και να υπολογίσουμε το πλήθος των πετυχημένων ιχνηλατήσεων.

Συγκεκριμένα, έχουμε διάφορα δίκτυα από routerάκια, τα οποία έχουν την μορφή κατευθυνόμενου γραφήματος. Τα routerάκια είναι αριθμημένα ως 1, 2, 3, ... Υποθέτουμε ότι μόνο τα 1, 2, . . . , 10 δημιουργούν μηνύματα, και το τελευταίο router (δηλαδή αυτό με το μεγαλύτερο νούμερο) είναι ο παραλήπτης των μηνυμάτων που εκκινεί την διαδικασία ιχνηλάτησης.

Τα γραφήματα περιέχονται στον φάκελο Networks, στην σελίδα του μαθήματος στο ecourse, και δίνονται σε μορφή αρχείου κειμένου που έχει την εξής μορφή. Η πρώτη γραμμή έχει την μορφή "n m", όπου n είναι το πλήθος των κόμβων (δηλαδή των router), και m είναι το πλήθος των ακμών (συνδέσεων). Κάθε μια από τις επόμενες m γραμμές έχει την μορφή "x y", που σημαίνει ότι ο x μπορεί να στέλνει μηνύματα στον y. Τα γραφήματα αυτά δεν περιέχουν κύκλους (οπότε η προώθηση μηνύματος δεν πρόκειται να πέσει σε ατέρμονα βρόχο), και από κάθε έναν από τους

κόμβους 1, 2, . . . , 10 υπάρχει (κατευθυνόμενο) μονοπάτι προς τον n (τον τελικό κόμβο-παραλήπτη). Επιπλέον, τα routerάκια 1, 2, . . ., 10 έχουν μόνο εξερχόμενες συνδέσεις προς άλλα routerάκια (δηλαδή λειτουργούν μονάχα ως πομποί, και όχι ως δέκτες).

Η διαδικασία δημιουργίας και προώθησης μηνύματος λειτουργεί ως εξής. Επιλέγουμε να παραχθούν 100.000 μηνύματα, όπου κάθε μήνυμα συνιστά μια συμβολοσειρά των 100 bits. Κάθε μήνυμα παράγεται τυχαία, και με τυχαίο τρόπο επιλέγεται ένα από τα routerάκια 1, 2, . . ., 10 ως αυτό που το δημιούργησε. Έπειτα, δεδομένου ότι το μήνυμα βρίσκεται σε ένα router x που δεν είναι το τερματικό (δηλαδή το n), επιλέγουμε τυχαία ένα από τα router-δέκτες του x (με άλλα λόγια: έναν κόμβο y για τον οποίο υπάρχει ακμή (x,y)), και στέλνουμε το μήνυμα στο y. Κάθε router έχει το δικό του φίλτρο Bloom, και κάθε φορά που περνάει μήνυμα από αυτό (ή που δημιουργείται από αυτό), το φίλτρο Bloom του καταγράφει ότι πέρασε από εκεί αυτό το μήνυμα.

Μόλις σταλθούν και τα 100.000 μηνύματα, το τελευταίο routerάκι αρχίζει να τα ιχνηλατεί ένα προς ένα. Για κάθε μήνυμα, ρωτάει τους πομπούς του αν το δέχτηκαν. Όσοι είπαν "ΝΑΙ", κάνουν το ίδιο, μέχρι κάποιο από τα αρχικά routerάκια (δηλαδή κάποιο από τα 1, 2, . . ., 10) να πει "ΝΑΙ". Σε περίπτωση που μόνο ένα από τα αρχικά routerάκια καταλήξει να έχει πει "ΝΑΙ", θεωρούμε την ιχνηλάτηση πετυχημένη.

Στο τέλος, θα πρέπει να εκτυπώνουμε το πλήθος των πετυχημένων ιχνηλατήσεων. (Προφανώς αυτό θα ποικίλλει, ανάλογα με την τυχαιότητα στα φίλτρα Bloom, αλλά αναμένεται ίσως να υπάρχει κάποια ξεκάθαρη τιμή για κάθε γράφημα, γύρω απ' την οποία συγκεντρώνονται οι τιμές από κάθε πείραμα ιχνηλάτησης.)

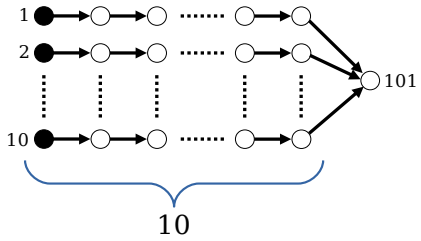
Θα δώσουμε σε κάθε routerάκι 10.000 bits μνήμης για το φίλτρο Bloom του, και για αρχή ας δοκιμάσουμε μόνο μία συνάρτηση κατακερματισμού (μία διαφορετική, για κάθε φιλτράκι). Οι συναρτήσεις κατακερματισμού θα είναι ακριβώς του ίδιου τύπου όπως και στην Άσκηση 1. Έπειτα δοκιμάστε περισσότερες συναρτήσεις κατακερματισμού (για κάθε φιλτράκι), και αναφέρετε για κάθε γράφημα ποιο πλήθος συναρτήσεων κατακερματισμού σάς φαίνεται ότι δίνει την καλύτερη απόδοση. (Η απόδοση εδώ μετράται ως το αναμενόμενο πλήθος των επιτυχημένων ιχνηλατήσεων.)

Ίσως μπορείτε να αντιληφθείτε ότι, κατά την διαδικασία της ιχνηλάτησης, ορισμένα routerάκια μπορεί να δεχτούν πάρα πολλά requests για έλεγχο ύπαρξης του μηνύματος στο φίλτρο Bloom τους. (Εξηγήστε γιατί μπορεί να συμβεί αυτό, και δώστε και ένα θεωρητικό παράδειγμα.) Αυτό θα ήταν αναπόφευκτο σε μια πραγματική εφαρμογή, και θα πρέπει να κρατήσετε αυτήν την υλοποίηση.

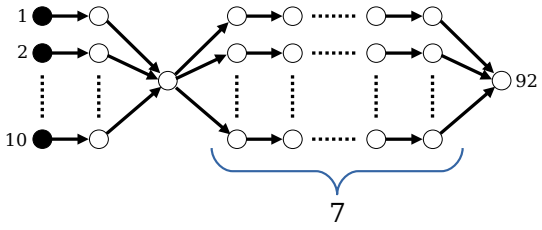
Παρά ταύτα, στο πλαίσιο της προσομοίωσης, μπορείτε να υλοποιήσετε επιπλέον και έναν πολύ πιο αποδοτικό αλγόριθμο για τον υπολογισμό των πετυχημένων ιχνηλατήσεων. Οι 0.5 μονάδες αυτής της άσκησης θα δοθούν για μία τέτοια αποδοτική υλοποίηση.

β) *(Μονάδες: 0.75)*
Ίσως παρατηρήσατε ότι σε ορισμένα γραφήματα η ιχνηλάτηση είναι γενικά πιο πετυχημένη απ' ό,τι σε κάποια άλλα. Πώς εξηγείτε την διαφορά στην επίδοση μεταξύ των διαφόρων γραφημάτων; Στην επόμενη σελίδα μπορείτε να δείτε μία αναπαράσταση όλων των γραφημάτων που βρίσκονται στον φάκελο Networks.

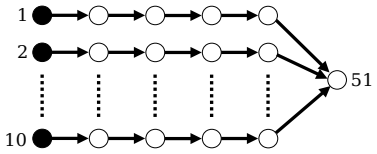
graphA1



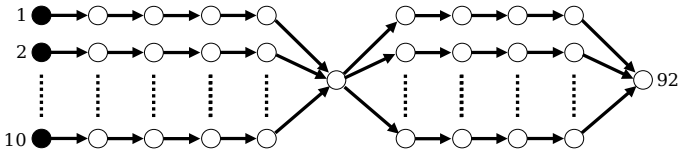
graphB1



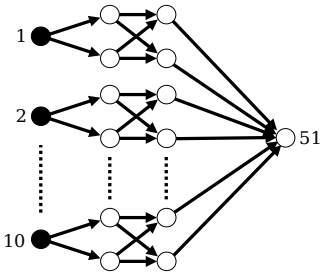
graphA2



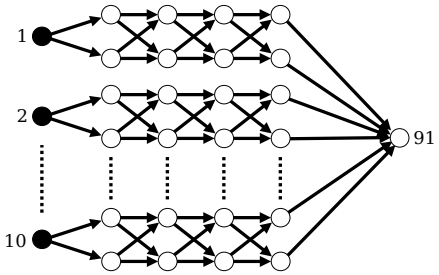
graphB2



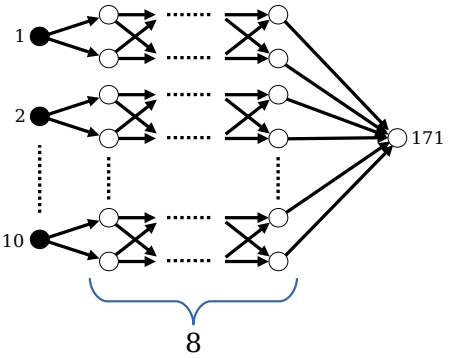
graphC1



graphC2



graphC3



graphD