Καθηγητής Π. Λουρίδας

Τμήμα Διοικητικής Επιστήμης και Τεχνολογίας

Οικονομικό Πανεπιστήμιο Αθηνών

Εκτίμηση Πλήθους

Έστω ότι έχουμε μια ακολουθία στοιχείων $(a_1, a_2, \dots, a_n)$, τα οποία λαμβάνουμε ένα-ένα, και θέλουμε να ξέρουμε το πλήθος των διαφορετικών στοιχείων μεταξύ τους, δηλαδή πόσα είναι χωρίς να λάβουμε υπόψην τυχόν επαναλαμβόμενες εμφανίσεις στοιχείων. Αν θέλουμε να είμαστε πιο ακριβείς, αν $A = \{a_1, a_2, \dots, a_n\}$ είναι τα σύνολο των μοναδικών στοιχείων της ακολουθίας, θέλουμε να ξέρουμε το μέγεθος του συνόλου $|A|$.

Βεβαίως η προφανής λύση είναι να κρατάμε κάπου τα στοιχεία που βλέπουμε και για κάθε στοιχείο να ελέγχουμε ότι το έχουμε δει· αν όχι, το προσθέτουμε σε αυτά που έχουμε δει και στο τέλος μετράμε το πλήθος τους. Αλλά τι μπορούμε να κάνουμε αν έχουμε περιορισμένη μνήμη και το δυνατό πλήθος των στοιχείων είναι πολύ μεγαλύτερο από όσα μπορούμε να διατηρήσουμε στη μνήμη μας;

Μπορούμε να εργαστούμε πιθανοθεωρητικά ώστε να πάρουμε μια καλή εκτίμηση του πλήθους. Χρησιμοποιούμε για αποθήκευση ένα χώρο περιορισμένου μεγέθους που ορίζουμε στην αρχή. Στην αποθήκη μας χώρο εισάγουμε στοιχεία όπως έρχονται πιθανοθεωρητικά· συνεπώς θα περιέχει ένα τυχαίο δείγμα των στοιχείων. Έστω ότι $A_t = \{a_1, a_2, \dots, a_t\}$ είναι τα πρώτα t μοναδικά στοιχεία τα οποία έχουν εισαχθεί στην αποθήκη. Τα περιεχόμενά της τη στιγμή t θα τα συμβολίζουμε με B_t. Χρησιμοποιούμε έναν τυχαίο αριθμό $0 \leq t < 1$ ώστε να εξασφαλίζουμε ότι:

$$P(a_j \in B_t) = p_t \qquad \text{για } 1 \leq j \leq t$$

Δηλαδή, τη στιγμή t, η αποθήκη μας περιέχει στοιχεία με πιθανότητα p. Τότε, η αναμενόμενη τιμή του αριθμού των στοιχείων στην αποθήκη, $|B_t|$, είναι $p_t|A_t|$, ή με άλλα λόγια:

$$\text{μια εκτίμηση του } |A_t| \text{ είναι το } |B_t|/p_t$$

Η μέθοδος αυτή είναι ο αλγόριθμος CVM, ο οποίος έχει ονομαστεί από τα αρχικά των ονομάτων των ερευνητών που τον ανακάλυψαν: Sourav Chakraborty, N. V. Vinodchandran, and Kuldeep S. Meel. Σε ψευδοκώδικα μπορούμε να τον αποδώσουμε ως εξής:

Algorithm 1: CVM.

`CVM(`B, E`)` $\rightarrow t$

Input: B, αποθηκευτικός χώρος με συγκεκριμένη χωρητικότητα
E, ακολουθία στοιχείων

Output: t, εκτίμηση του πλήθους των στοιχείων της E

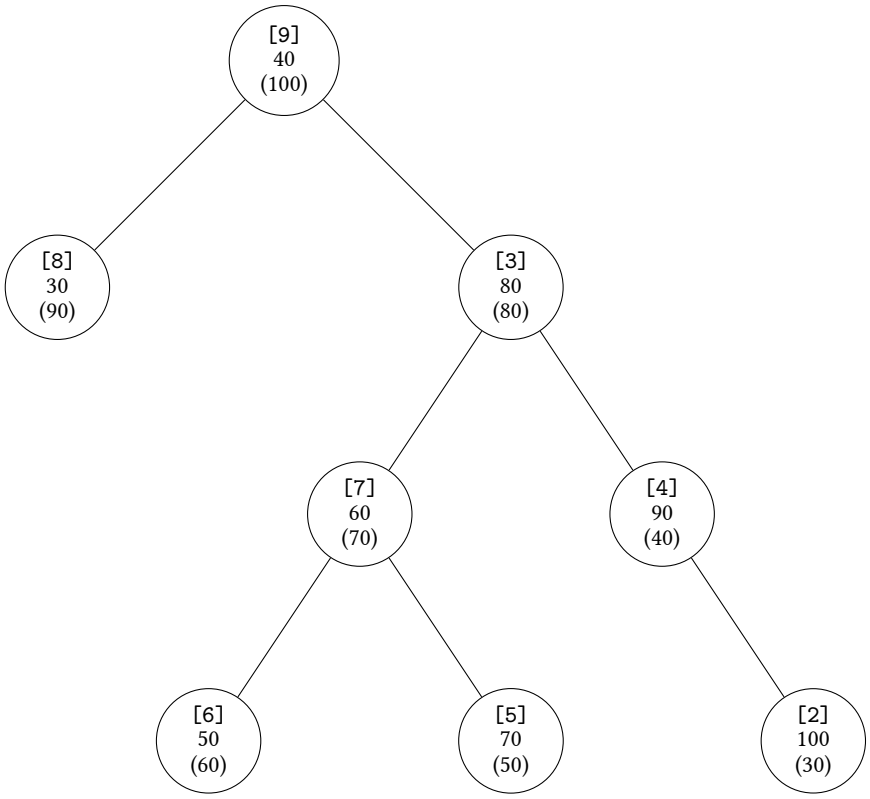
```
1  p ← 1
2  c ← 0
3  foreach e ∈ E do
4      if e ∈ B then
5          DeleteElement(B, e)
6          c ← t − 1
7      u ← Random()
8      if u ≤ p then
9          InsertElement(B, e, u)
10         c ← c + 1
11     if IsFull(B) then
12         p ← GetTopProbability(B)
13         d ← DeleteElements(B, p)
14         c ← c − d
15 t ← c/p
16 return t
```

Ο αλγόριθμος ξεκινάει με πιθανότητα $p = 1$. Για κάθε στοιχείο της ακολουθίας, ελέγχει αν είναι στην αποθήκη, και αν είναι το διαγράφει. Στη συνέχεια επιλέγει έναν τυχαίο αριθμό $0 \leq u < 1$. Αν ο τυχαίος αριθμός είναι μικρότερος ή ίσος της πιθανότητας p τον εισάγει στην αποθήκη. Σε περίπτωση που η αποθήκη γεμίσει, βρίσκουμε τη μέγιστη πιθανότητα με την οποία είναι αποθηκευμένο ένα στοιχείο και αφαιρούμε από την αποθήκη όλα τα d στοιχεία που έχουν αποθηκευτεί με αυτήν την πιθανότητα (μπορεί να τύχει περισσότερες από μία φορά το ίδιο u στη γραμμή 7). Στο τέλος επιστρέφουμε την εκτίμησή μας.

Για να υλοποιήσουμε τον αλγόριθμο αυτό, πρέπει να δούμε πώς θα λειτουργεί η αποθήκη B. Μια καλή επιλογή είναι η δομή δεδομένων *treap*. Το όνομα προέρχεται από το συνδυασμό των λέξεων tree και heap, διότι συνδυάζει τα χαρακτηριστικά αυτών των δύο. Συγκεκριμένα, ένα treap είναι ένα δυαδικό δέντρο με τα εξής χαρακτηριστικά:

- Κάθε κόμβος έχει ένα κλειδί και μια προτεραιότητα.
- Όλα τα παιδιά που βρίσκονται κάτω από το αριστερό κλαδί ενός κόμβου έχουν κλειδί που είναι μικρότερο από το κλειδί του κόμβου.
- Όλα τα παιδιά που βρίσκονται κάτω από το δεξί κλαδί ενός κόμβου έχουν κλειδί μεγαλύτερο ή ίσο από το κλειδί του κόμβου.
- Κάθε κόμβος έχει προτεραιότητα μεγαλύτερη από την προτεραιότητα των παιδιών του.

Παρακάτω βλέπουμε ένα treap. Σε κάθε κόμβο έχουμε το κλειδί του και σε παρένθεση την προτεραιότητά του. Σε αγκύλες έχουμε τον αριθμό του κόμβου (θα αναφερθούμε σε αυτόν αργότερα όταν αναλύσουμε πώς αποθηκεύουμε αυτό το δένδρο).

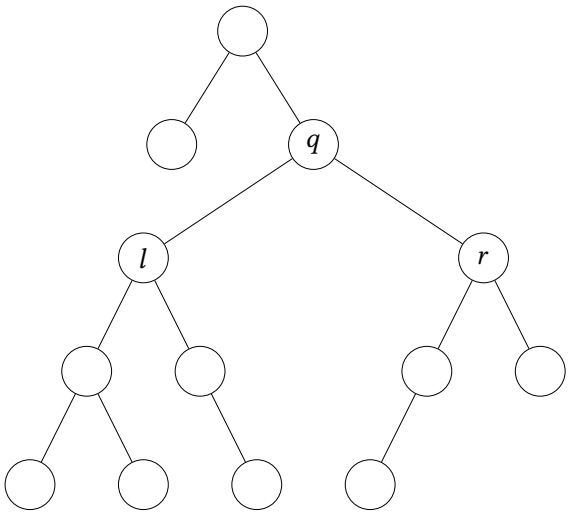


Αν έχουμε ένα treap τότε οι λειτουργίες του *B* που απαιτούνται από τον αλγόριθμο CVM υλοποιούνται αποτελεσματικά. Η αναζήτηση ενός στοιχείου στη γραμμή 4 του αλγορίθμου αντιστοιχεί στη διάσχιση του δένδρου από τη ρίζα. Αν το στοιχείο έχει μικρότερο παιδί πηγαίνουμε στα αριστερά, διαφορετικά πηγαίνουμε στα δεξιά. Η εύρεση του στοιχείου με τη μεγαλύτερη πιθανότητα στη γραμμή 12 του αλγορίθμου αντιστοιχεί στην εύρεση του κόμβου με τη μεγαλύτερη προτεραιότητα, που είναι η ρίζα του δένδρου.

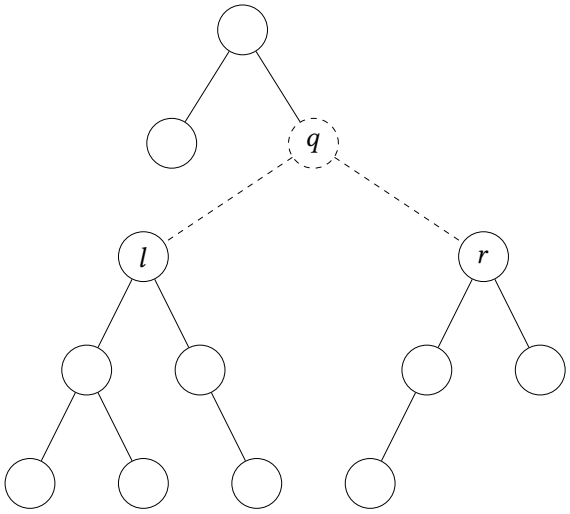
Αυτό που μένει είναι να δούμε είναι η διαγραφή στοιχείων, η εισαγωγή στοιχείων, και η αποθήκευση της δομής αυτής.

Διαγραφή Στοιχείου

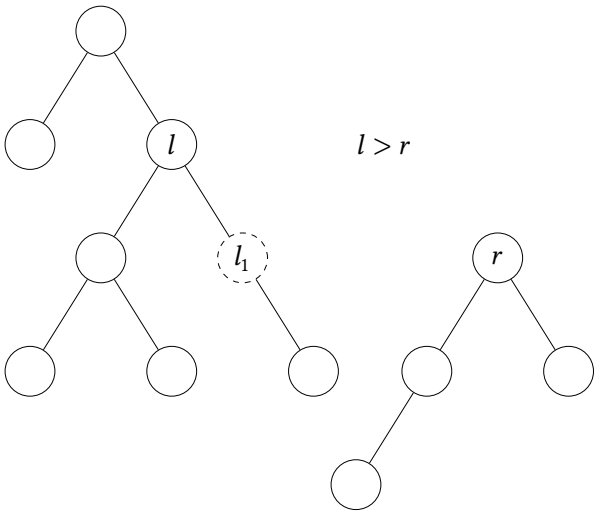
Για να διαγράψουμε ένα στοιχείο, αν αυτό είναι φύλλο στο δένδρο, απλώς το αφαιρούμε από αυτό. Αν είναι εσωτερικός κόμβος, τα πράγματα είναι πιο ενδιαφέροντα. Έστω ότι θέλουμε να διαγράψουμε τον κόμβο *q* στο παρακάτω δένδρο.



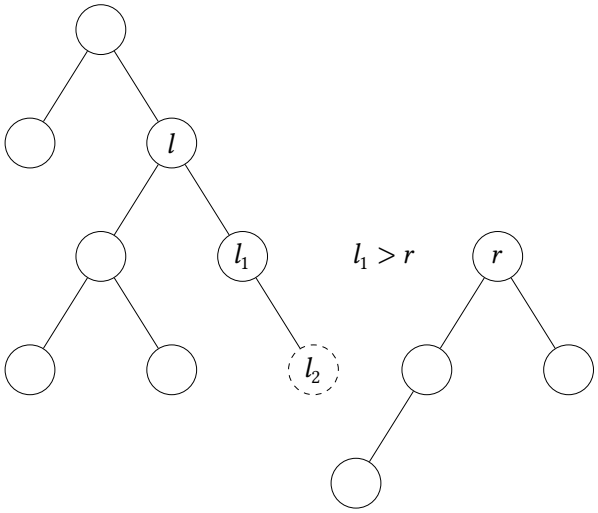
Αν αυτός διαγραφεί, παίρνουμε δύο επιμέρους δένδρα, με ρίζα l και r. Αυτά θα πρέπει να συγχωνευτούν και το αποτέλεσμα να μπει στη θέση του q.



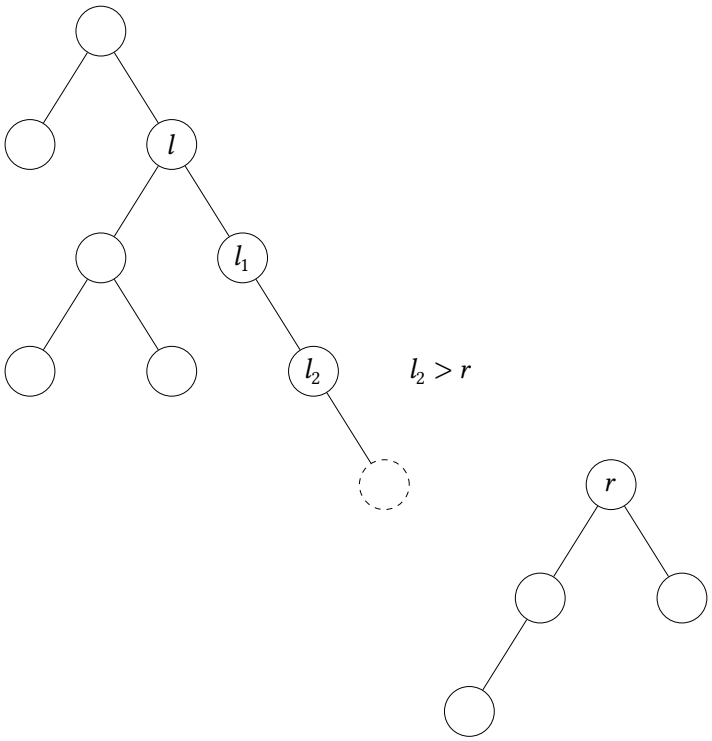
Αν η προτεραιότητα του l είναι μεγαλύτερη από αυτή του r, τότε ο κόμβος l θα είναι η ρίζα του συγχωνευμένου δένδρου και θα τοποθετηθεί στη θέση του q. Συνεχίζουμε τότε τη διαδικασία της συγχώνευσης ελέγχοντας το δεξί παιδί του l, l_1, με το δένδρο με ρίζα r.



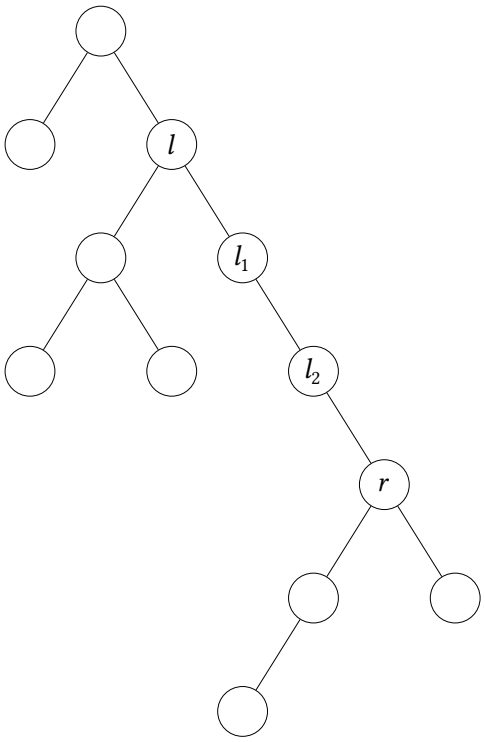
Αν η προτεραιότητα του l_1 είναι πάλι μεγαλύτερη από την προτεραιότητα του r, τότε το l_1 θα είναι η ρίζα του συγχωνευμένου δένδρου (άρα θα μείνει εκεί που είναι), θα συνεχίσουμε τη διαδικασία με το δεξί παιδί του, το l_2, και το αποτέλεσμα της συγχώνευσης θα μπει στη θέση του l_2.



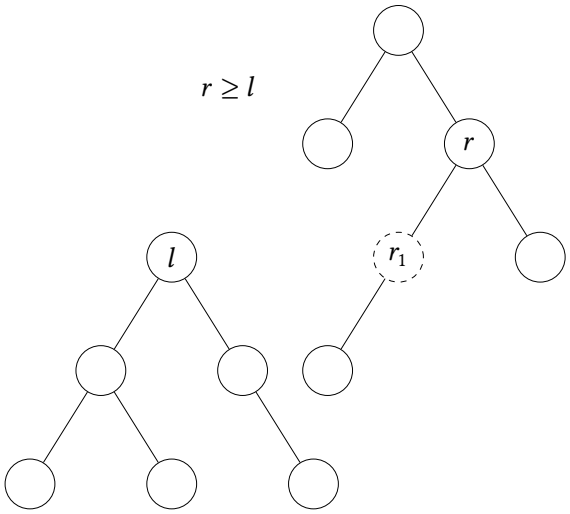
Αν και πάλι η προτεραιότητα του l_2 είναι μεγαλύτερη από την προτεραιότητα του r, το l_2 θα είναι η ρίζα του συγχωνευμένου δένδρου (θα μείνει εκεί που είναι). Συνεχίζουμε τη διαδικασία της συγχώνευσης, και το αποτέλεσμα της συγχώνευσης θα τοποθετηθεί στο δεξί παιδί του l_2.



Η συγχώνευση τελειώνει αφού πλέον υπάρχει μόνο ένα δένδρο· το τοποθετούμε στη θέση στα δεξιά του l_2.

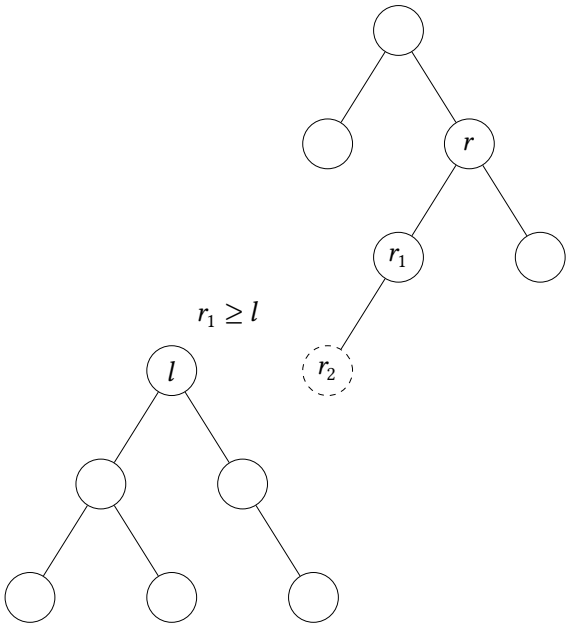


Αν η προτεραιότητα του l δεν είναι μεγαλύτερη από αυτή του r, τότε ο κόμβος r θα είναι η ρίζα του συγχωνευμένου δένδρου και θα τοποθετηθεί στη θέση του q. Συνεχίζουμε τότε τη διαδικασία της συγχώνευσης ελέγχοντας το αριστερό παιδί του r, r_1, με το δένδρο με ρίζα l. Το αποτέλεσμα της συγχώνευσης θα μπει στη θέση του r_1.

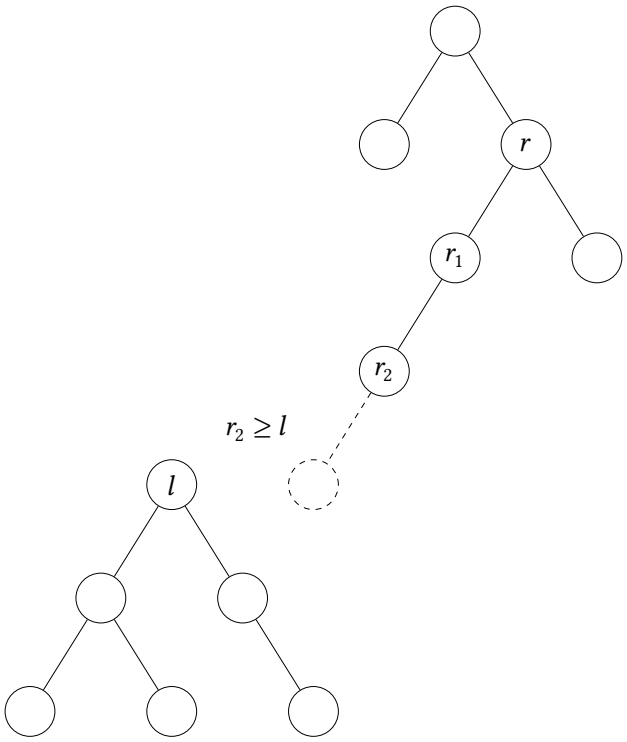


Έστω ότι πάλι $r_1 \geq l$, τότε το r_1 θα είναι η ρίζα του νέου δένδρου (άρα παραμένει εκεί

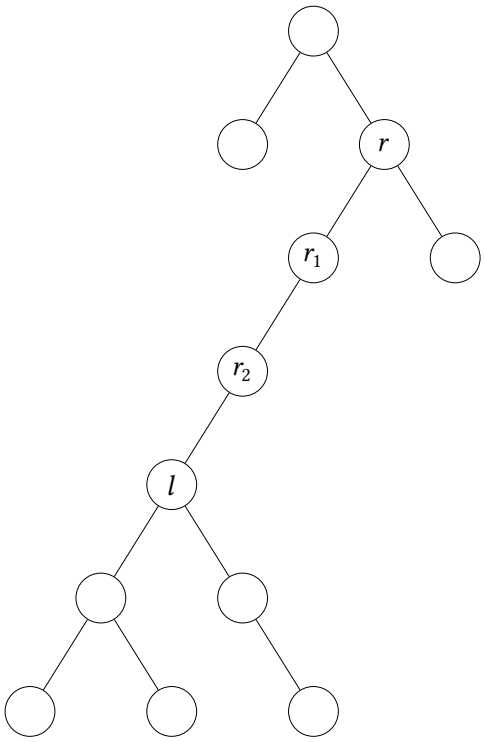
που είναι) και συνεχίζουμε τη διαδικασία της συγχώνευσης με το αριστερό παιδί του, το r_2. Το αποτέλεσμα της συγχώνευσης θα τοποθετηθεί στη θέση του r_2.



Αν και πάλι $r_2 \geq l$, τότε το r_2 θα είναι η ρίζα του νέου δένδρου (άρα μένει εκεί που είναι), και η συγχώνευση θα συνεχιστεί με το αποτέλεσμα της να αποθηκευτεί στο αριστερό παιδί του r_2.



Η συγχώνευση τελειώνει αφού πλέον υπάρχει μόνο ένα δένδρο· το τοποθετούμε στη θέση στα αριστερά του r_2.



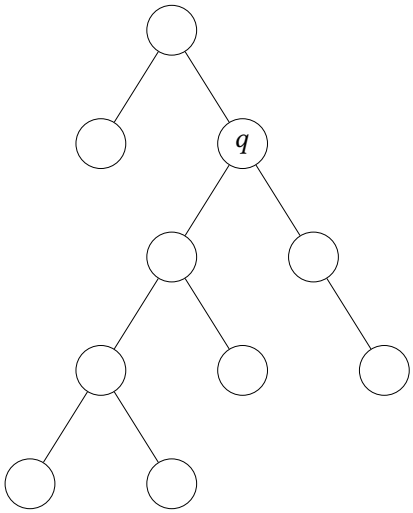
Εν ολίγοις, στη συγχώνευση διακρίνουμε τέσσερεις περιπτώσεις:

1. Θέλουμε να συγχωνεύσουμε ένα δένδρο με ένα κενό δεξί παιδί: το δένδρο γίνεται το δεξί παιδί και σταματάμε.
2. Θέλουμε να συγχωνεύσουμε ένα δένδρο με ένα κενό αριστερό παιδί: το δένδρο γίνεται το αριστερό παιδί και σταματάμε.
3. Θέλουμε να συγχωνεύσουμε δύο δένδρα, και η ρίζα του αριστερού δένδρου έχει μεγαλύτερη προτεραιότητα: γίνεται η ρίζα του νέου δένδρου και συνεχίζουμε με το δεξί παιδί του και το δεξί δένδρο.
4. Θέλουμε να συγχωνεύσουμε δύο δένδρα, και η ρίζα του δεξιού δένδρου έχει μεγαλύτερη ή ίση προτεραιότητα: γίνεται η ρίζα του νέου δένδρου και συνεχίζουμε με το αριστερό παιδί του και το αριστερό δένδρο.

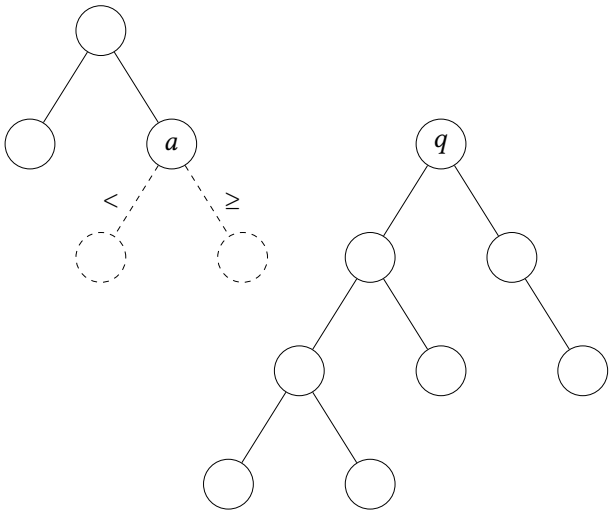
Εισαγωγή στοιχείου

Για την εισαγωγή ενός στοιχείου, ξεκινάμε από τη ρίζα του δένδρου και αναζητούμε τη θέση που θα το εισάγουμε. Σε κάθε κόμβο εξετάζουμε την προτεραιότητά του: αν η προτεραιότητα του κόμβου είναι μεγαλύτερη ή ίση από την προτεραιότητα του νέου στοιχείου, τότε αν το κλειδί του κόμβου είναι μεγαλύτερο από το κλειδί του νέου στοιχείου συνεχίζουμε τη διάσχιση του δένδρου πηγαίνοντας στο αριστερό παιδί, αλλιώς πηγαίνουμε στο δεξί παιδί.

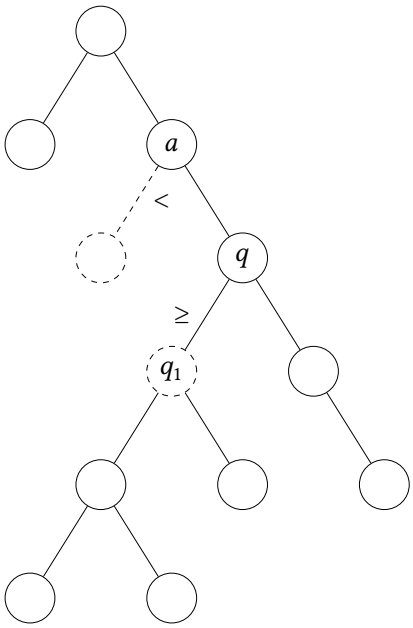
Αν με τη διάσχιση καταλήξουμε σε κενό αριστερό ή δεξί παιδί, απλώς εισάγουμε τον νέο κόμβο εκεί. Τα πράγματα είναι πιο ενδιαφέροντα αν σταματήσουμε σε εσωτερικό κόμβο. Έστω ότι ο νέος κόμβος έχει κλειδί a και θέλουμε να το εισάγουμε στη θέση του κόμβου με κλειδί q.



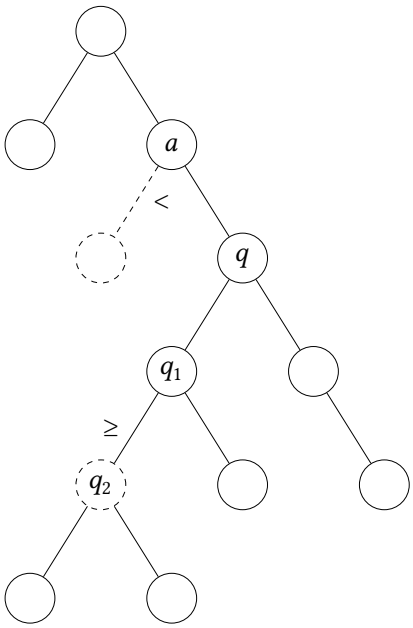
Βάζουμε τον νέο κόμβο στη θέση του q. Θα πρέπει τότε να δούμε που θα μπουν οι κόμβοι του δένδρου με ρίζα το q. Έχουμε δύο δυνατότητες, δύο θέσεις, στα αριστερά και δεξιά του a.



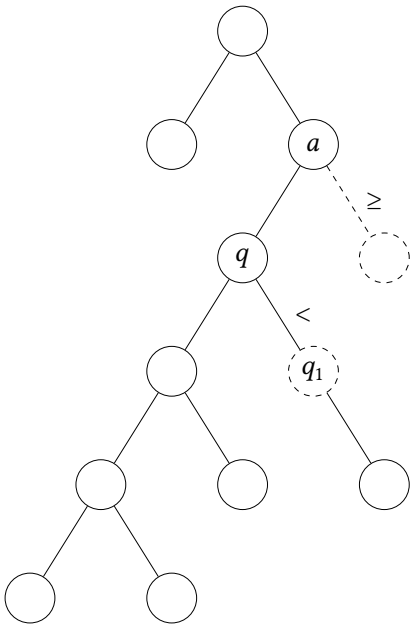
Ελέγχουμε αν το a είναι μικρότερο από το q. Αν είναι, τότε το δένδρο με ρίζα το q γίνεται το δεξί παιδί του κόμβου με κλειδί a και συνεχίζουμε τη διαδικασία με το δένδρο που ξεκινά από το αριστερό παιδί του, με κλειδί q_1.



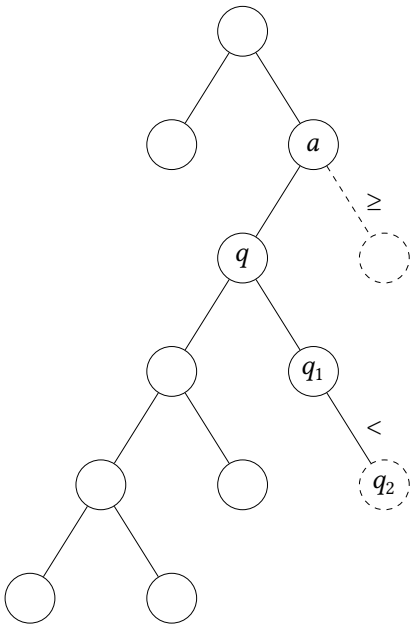
Αν πάλι το a είναι μικρότερο του q_1, τότε το q_1 παραμένει εκεί που είναι και συνεχίζουμε με το δένδρο που ξεκινά από το αριστερό παιδί του, με κλειδί q_2.



Πηγαίνοντας πίσω, αν το a δεν είναι μικρότερο του q, τότε ο κόμβος με κλειδί q θα γίνει το αριστερό κλειδί του a και θα συνεχίσουμε τη διαδικασία με το δένδρο που ξεκινά από το δεξί παιδί του q, q_1.



Αν τώρα $q_1 < a$, το q_1 παραμένει εκεί που είναι και συνεχίζουμε τη διαδικασία με το δένδρο που ξεκινά από το δεξί παιδί του q_1, q_2.



Αν τώρα το q_2 είναι μικρότερο του a μένει εκεί που είναι. Προχωράμε στο δεξί παιδί του q_2, που είναι όμως κενό, και σταματάμε. Αν το q_2 είναι μεγαλύτερο ή ίσο του a, θα το βάλουμε στη θέση στα αριστερά του a. Προχωράμε στο αριστερό παιδί του q_2

που είναι όμως κενό και σταματάμε.

Εν ολίγοις, στην εισαγωγή, όταν ξεκινάμε τη διαδικασία ορίζουμε δύο θέσεις, αριστερά και δεξιά του *a*, όπου μπορούν να μπουν οι κόμβοι που εξετάζουμε.

1. Αν το κλειδί του κόμβου που εξετάζουμε είναι μικρότερο του *a*, μπαίνει στην αριστερή θέση και η νέα αριστερή θέση είναι το δεξί παιδί της αριστερής θέσης.
2. Αν το κλειδί του κόμβου που εξετάζουμε είναι μεγαλύτερο ή ίσο του *a* μπαίνει στη δεξιά θέση και η νέα δεξιά θέση είναι το αριστερό παιδί της δεξιάς θέσης.
3. Συνεχίζουμε μέχρις ότου φτάσουμε σε κενό παιδί όπου εισάγουμε τον κόμβο και σταματάμε.

Αποθήκευση

Για την αποθήκευση των κόμβων του δένδρου χρησιμοποιούμε έναν πίνακα. Κάθε στοιχείο του πίνακα περιέχει έναν κόμβο. Έστω ότι χρησιμοποιούμε έναν πίνακα δέκα θέσεων. Ο πίνακας αρχικοποιείται με κόμβους με κενά κλειδιά και προτεραιότητες. Ο κάθε κόμβος δείχνει στον κόμβο στα αριστερά του, εκτός από τον αριστερότερο κόμβο. Επίσης διατηρούμε έναν δείκτη *avail* που δείχνει στον πρώτο διαθέσιμο (άδειο) κόμβο. Στην ουσία, ο δείκτης *avail* είναι η κεφαλή της λίστας των διαθέσιμων κόμβων. Ο πίνακας μας λοιπόν αρχικά θα είναι:

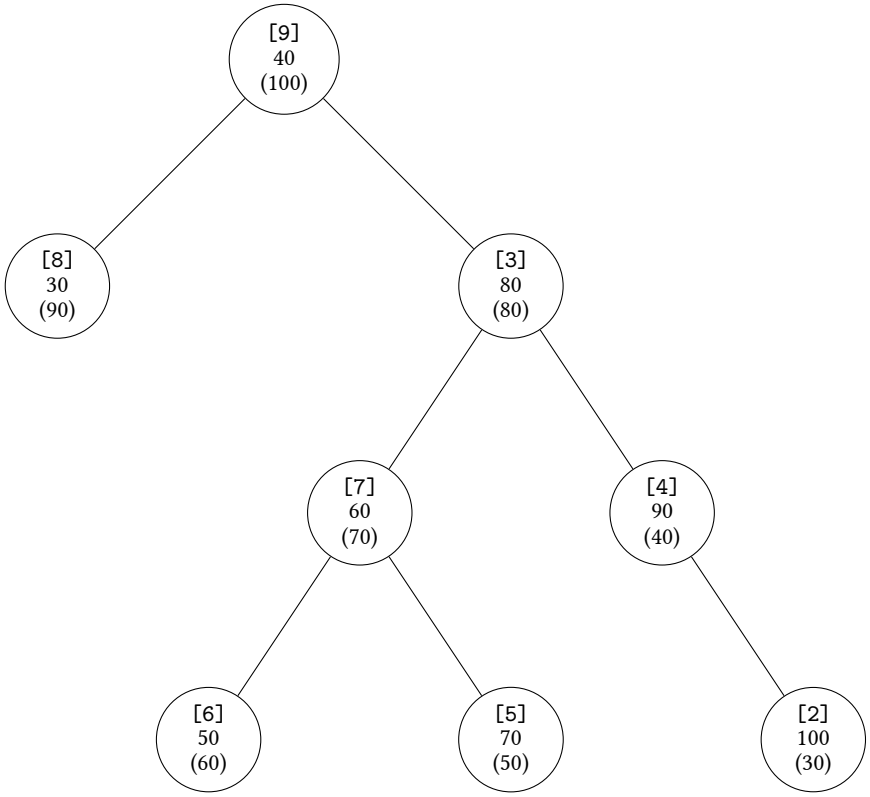
[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
(-1, -1)	(0, -1)	(1, -1)	(2, -1)	(3, -1)	(4, -1)	(5, -1)	(6, -1)	(7, -1)	(8, -1)

$avail = 9$

Εισάγουμε τα στοιχεία με τα παρακάτω κλειδιά και προτεραιότητες το ένα μετά το άλλο:

Κλειδί	Προτεραιότητα
40	100
30	90
60	70
50	60
70	50
90	40
80	80
100	30

Τότε θα προκύψει το παρακάτω δένδρο:

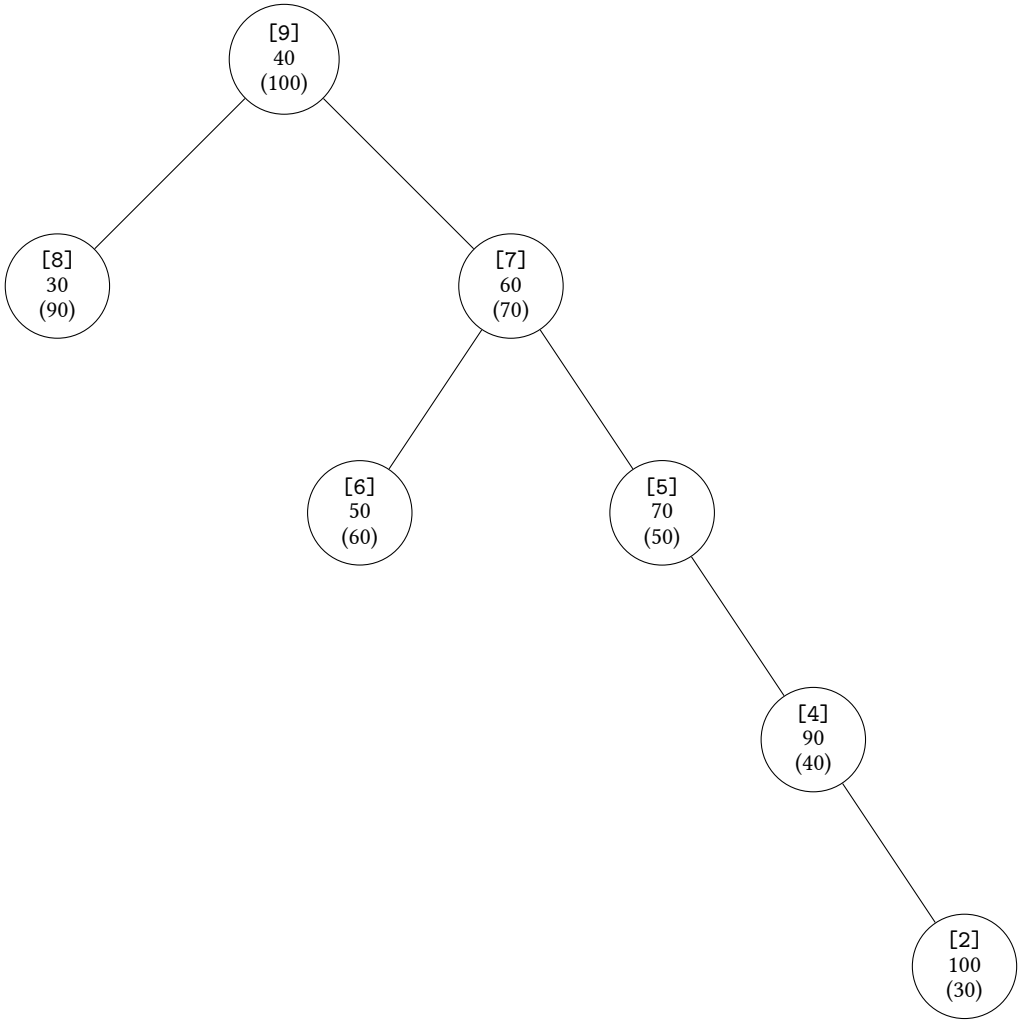


Το δένδρο αυτό θα αποθηκεύεται όπως φαίνεται στον παρακάτω πίνακα (για λόγους οικονομίας χώρου παραλείπουμε τις προτεραιότητες στον κάθε κόμβο). Κάθε φορά που εισάγουμε ένα στοιχείο στο δένδρο, αυτό μπαίνει στη θέση που δείχνει ο δείκτης *avail*, και ο δείκτης αυτός μεταφέρεται στο αριστερό παιδί του. Με άλλα λόγια, από τη λίστα των διαθέσιμων κόμβων παίρνουμε τον πρώτο και η κεφαλή της λίστας μεταφέρεται στον επόμενο ελεύθερο κόμβο.

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
-1 (-1, -1)	-1 (0, -1)	100 (-1, -1)	80 (7, 4)	90 (-1, 2)	70 (-1, -1)	50 (-1, -1)	60 (6, 5)	30 (-1, -1)	40 (8, 3)

$avail = 1$

Αν διαγράψουμε τον κόμβο με κλειδί 80, θα πάρουμε το παρακάτω δένδρο:



Ο πίνακας τότε θα είναι όπως παρακάτω. Όταν διαγράφουμε ένα στοιχείο από τον πίνακα, ο κόμβος του επιστρέφει στη λίστα των διαθέσιμων, επομένως το αριστερό παιδί του δείχνει στη θέση του *avail* και ο δείκτης *avail* ενημερώνεται να δείχνει στον μόλις απελευθερωμένο κόμβο.

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
-1 (-1, -1)	-1 (0, -1)	100 (-1, -1)	80 (1, 4)	90 (-1, 2)	70 (-1, 4)	50 (-1, -1)	60 (6, 5)	30 (-1, -1)	40 (8, 7)

$avail = 3$

Σκοπός της εργασίας είναι να υλοποιήσετε τον αλγόριθμο CVM όπως περιγράφεται εδώ, χρησιμοποιώντας treaps όπως περιγράφονται εδώ. Τυχόν άλλες υλοποιήσεις

δεν θα γίνουν δεκτές.

Απαιτήσεις Προγράμματος

Κάθε φοιτητής θα εργαστεί σε αποθετήριο στο GitHub. Για να αξιολογηθεί μια εργασία θα πρέπει να πληροί τις παρακάτω προϋποθέσεις:

- Για την υποβολή της εργασίας θα χρησιμοποιηθεί το ιδιωτικό αποθετήριο του φοιτητή που δημιουργήθηκε για τις ανάγκες του μαθήματος και του έχει αποδοθεί. Το αποθετήριο αυτό έχει όνομα του τύπου `username-algo-assignments`, όπου `username` είναι το όνομα του φοιτητή στο GitHub. Για παράδειγμα, το σχετικό αποθετήριο του διδάσκοντα θα ονομαζόταν `louridas-algo-assignments` και θα ήταν προσβάσιμο στο https://github.com/dmst-algorithms-course/louridas-algo-assignments. *Τυχόν άλλα αποθετήρια απλώς θα αγνοηθούν.*
- Μέσα στο αποθετήριο αυτό θα πρέπει να δημιουργηθεί ένας κατάλογος `assignment-2025-3`.
- Μέσα στον παραπάνω κατάλογο το πρόγραμμα θα πρέπει να αποθηκευτεί με το όνομα `cvm.py`.
- Δεν επιτρέπεται η χρήση έτοιμων βιβλιοθηκών γράφων ή τυχόν έτοιμων υλοποιήσεων των αλγορίθμων, ή τμημάτων αυτών, εκτός αν αναφέρεται ρητά ότι επιτρέπεται.
- Επιτρέπεται η χρήση δομών δεδομένων της Python όπως στοίβες, λεξικά, σύνολα, κ.λπ.
- Επιτρέπεται η χρήση των παρακάτω βιβλιοθηκών ή τμημάτων τους όπως ορίζεται:
  - `sys.argv`
  - `argparse`
  - `random`
- Το πρόγραμμα θα πρέπει να είναι γραμμένο σε Python 3.
- Η εργασία είναι αποκλειστικά ατομική. Δεν επιτρέπεται συνεργασία μεταξύ φοιτητών στην εκπόνησή της, με ποινή το μηδενισμό. Επιπλέον η εργασία δεν μπορεί να είναι αποτέλεσμα συστημάτων Τεχνητής Νοημοσύνης (όπως ChatGPT). Ειδικότερα όσον αφορά το τελευταίο σημείο προσέξτε ότι τα συστήματα αυτά χωλαίνουν στην αλγοριθμική σχέση, άρα τυχόν προτάσεις που κάνουν σε σχετικά θέματα μπορεί να είναι λανθασμένες. Επιπλέον, αν θέλετε να χρησιμοποιήσετε τέτοια συστήματα, τότε αν και κάποιος άλλος το κάνει αυτό, μπορεί οι προτάσεις που θα λάβετε να είναι παρόμοιες, οπότε οι εργασίες θα παρουσιάσουν ομοιότητες και άρα θα μηδενιστούν.
- Η έξοδος του προγράμματος θα πρέπει να περιλαμβάνει μόνο ό,τι φαίνεται στα παραδείγματα που παρατίθενται. *Η φλυαρία δεν επιβραβεύεται.*

Χρήση του GitHub

Όπως αναφέρθηκε το πρόγραμμά σας για να αξιολογηθεί θα πρέπει να αποθηκευθεί στο GitHub. Επιπλέον, θα πρέπει το GitHub να χρησιμοποιηθεί καθόλη τη διάρκεια της ανάπτυξης του.

Αυτό σημαίνει ότι *δεν θα ανεβάσετε στο GitHub απλώς την τελική λύση του προβλήματος* μέσω της λειτουργίας "Upload files". Στο GitHub θα πρέπει να φαίνεται *το ιστορικό της συγγραφής του προγράμματος*. Αυτό συνάδει και με τη φιλοσοφία του εργαλείου: λέμε "commit early, commit often". Εργαζόμαστε σε ένα πρόγραμμα και κάθε μέρα, ή όποια στιγμή έχουμε κάνει κάποιο σημαντικό βήμα, αποθηκεύουμε την αλλαγή στο GitHub. Αυτό έχει σειρά ευεργετικών αποτελεσμάτων:

- Έχουμε πάντα ένα αξιόπιστο εφεδρικό μέσο στο οποίο μπορούμε να ανατρέξουμε αν κάτι πάει στραβά στον υπολογιστή μας (μας γλιτώνει από πανικούς του τύπου: ένα βράδυ πριν από την παράδοση ο υπολογιστής μας ή ο δίσκος του πνέει τα λοίσθια, και εμείς τι θα υποβάλουμε στον Λουρίδα;).
- Καθώς έχουμε πλήρες ιστορικό των σημαντικών αλλαγών και εκδόσεων του προγράμματός μας, μπορούμε να επιστρέψουμε σε μία προηγούμενη αν συνειδητοποιήσουμε κάποια στιγμή ότι πήραμε λάθος δρόμο (μας γλιτώνει από πανικούς του τύπου: μα το πρωί δούλευε σωστά, τι στο καλό έκανα και τώρα δεν παίζει τίποτε;).
- Καθώς φαίνεται η πρόοδος μας στο GitHub, επιβεβαιώνουμε ότι το πρόγραμμα δεν είναι αποτέλεσμα της όποιας επιφοίτησης (μας γλιτώνει από πανικούς του τύπου: καλά, πώς το έλυσες το πρόβλημα αυτό με τη μία, μόνος σου;).
- Αν δουλεύετε σε μία ομάδα *που βεβαίως δεν είναι καθόλου η περίπτωση μας εδώ*, μπορούν όλοι να εργάζονται στα ίδια αρχεία στο GitHub εξασφαλίζοντας ότι ο ένας δεν γράφει πάνω στις αλλαγές του άλλου. Παρά το ότι η εργασία αυτή είναι ατομική, καλό είναι να αποκτάτε τριβή με το εργαλείο git και την υπηρεσία GitHub μιας και χρησιμοποιούνται ευρέως και όχι μόνο στη συγγραφή κώδικα.

Άρα επενδύστε λίγο χρόνο στην εκμάθηση των git / GitHub, των οποίων ο σωστός τρόπος χρήσης είναι μέσω γραμμής εντολών (command line), ή ειδικών προγραμμάτων (clients) ή μέσω ενοποίησης στο περιβάλλον ανάπτυξης (editor, IDE).

Τελικό Πρόγραμμα

Το πρόγραμμα θα καλείται ως εξής (όπου `python` η κατάλληλη εντολή στο εκάστοτε σύστημα):

```
python cvm.py [-s SEED] size filename
```

Το πρόγραμμα δέχεται μία προαιρετική και δύο υποχρεωτικές παραμέτρους:

- Η παράμετρος `seed`, αν δωθεί, είναι ο σπόρος της γεννήτριας τυχαίων αριθμών της Python. Αυτό μας επιτρέπει να μπορούμε για την ίδια είσοδο να αναπαράξουμε την ίδια έξοδο.

- Η παράμετρος `size` δίνει το μέγεθος του πίνακα που θα χρησιμοποιήσουμε για την αποθήκευση του treap.
- Η παράμετρος `filename` δίνει το όνομα του αρχείου που περιέχει τα στοιχεία που θέλουμε να καταμετρήσουμε.

Το πρόγραμμα στην έξοδο θα δίνει την εκτίμηση του αριθμού των μοναδικών στοιχείων που περιέχονται στο αρχείο `filename`.

Παραδείγματα

Παράδειγμα 1

Αν ο χρήστης του προγράμματος δώσει:

```
python -s 42 10000 seq_1.txt
```

το πρόγραμμά σας θα διαβάσει το αρχείο `seq_1.txt`. Το αρχείο αυτό περιέχει 1,000,000 τυχαίους αριθμούς, εκ των οποίων οι 946,618 είναι μοναδικοί. Το πρόγραμμά σας θα πρέπει να εμφανίσει ακριβώς:

```
946902
```

Παράδειγμα 1

Αν ο χρήστης του προγράμματος δώσει:

```
python -s 42 10000 seq_2.txt
```

το πρόγραμμά σας θα διαβάσει το αρχείο `seq_2.txt`. Το αρχείο αυτό περιέχει 1,000,000 αριθμούς από το 1 έως το 50,000 επαναλαμβανόμενους 20 φορές. Το πρόγραμμά σας θα πρέπει να εμφανίσει ακριβώς:

```
49030
```

Καλή Επιτυχία!