Εισαγωγή

Στην εργασία αυτή θα υλοποιήσετε ένα πρόγραμμα που θα δέχεται, θα επεξεργάζεται, θα καταγράφει και θα απαντάει ερωτήματα για Bitcoin συναλλαγές. Συγκεκριμένα θα υλοποιήσετε ένα σύνολο δομών (hash tables, linked lists, trees) που επιτρέπουν την εισαγωγή και επερωτήσεις σε μεγάλο όγκο εγγραφών τύπου `bitCoinTransaction`. Αν και τα δεδομένα της άσκησης θα προέρχονται από αρχεία, τελικά όλες οι εγγραφές θα αποθηκεύονται μόνο στην κύρια μνήμη.

Interface της εφαρμογής

Η εφαρμογή θα ονομάζεται `bitcoin` και θα χρησιμοποιείται ως εξής:

```
./bitcoin -a bitCoinBalancesFile -t transactionsFile -v bitCoinValue -h1
  senderHashtableNumOfEntries -h2 receiverHashtableNumOfEntries -b bucketSize
```

όπου:

- Η παράμετρος `bitCoinValue` λέει στην εφαρμογή ποια είναι η αξία (σε $) ενός bitcoin.
- Η παράμετρος `senderHashtable1NumOfEntries` είναι ο αριθμός θέσεων ενός πίνακα κατακερματισμού που θα κρατάει η εφαρμογή για την εντόπιση πληροφοριών αποστολέων συναλλαγών.
- Η παράμετρος `receiverHashtable1NumOfEntries` είναι ο αριθμός θέσεων ενός πίνακα κατακερματισμού που θα κρατάει η εφαρμογή για την εντόπιση πληροφοριών παραληπτών συναλλαγών.
- Η παράμετρος `bucketSize` είναι ο αριθμός των Bytes που δίνει το μέγεθος του κάθε bucket στους πίνακες κατακερματισμού.
- Το `bitCoinBalancesFile` (ή κάποιο άλλο όνομα αρχείου) είναι ένα αρχείο που περιλαμβάνει τα αρχικά balances των χρηστών που συμμετέχουν στο bitcoin network. Κάθε γραμμή του αρχείου αυτού είναι μια λίστα με τα bitCoin IDs που έχει στη κατοχή του ένας συγκεκριμένος χρηστής (userID). Για παράδειγμα αν τα περιεχόμενα του αρχείου είναι:

```
Mia 123 337 880 667
Kylian 456 767 898
Katerina 222 567 003
```

  σημαίνει πως έχουμε αρχικά τρεις χρήστες που συμμετέχουν στο network. Η Mia έχει τέσσερα bitcoins, με bitCoinIDs 123, 337, 880, 667, ο Kylian έχει τρία bitcoins με IDs 456, 767, 898, κοκ. Μπορείτε να θεωρήσετε πως το όνομα του χρήστη δεν περιέχει κενά.

- Το `transactionsFile` (ή κάποιο άλλο όνομα αρχείου) είναι ένα αρχείο που περιέχει μια σειρά από αιτήματα (συναλλαγές) προς επεξεργασία. Κάθε γραμμή του αρχείου αυτού περιγράφει μια συναλλαγή με τα userID του αποστολέα και του παραλήπτη και το ποσόν (σε $) που στέλνει ο αποστολέας. Για παράδειγμα αν τα περιεχόμενα του αρχείου είναι:

```
889 Maria Ronaldo 50 25-12-2018 20:08
776 Lionel Antonella 150 14-02-2019 10:05
```

  σημαίνει πως έχουμε δυο συναλλαγές όπου η Maria ζητάει να στείλει $50 στον Ronaldo στις 25-12-2018, ώρα 8:08 το βράδυ και ο Lionel ζητάει να στείλει στην Antonella $150 στις 14-2-2019, ώρα 10:05 το πρωί. Μπορείτε να θεωρήσετε πως τα περιεχόμενα του αρχειου με τα transactions θα είναι ταξινομημένα ανά ημερομηνία και ώρα.

  Συγκεκριμένα, μια εγγραφή/αίτημα συναλλαγής είναι μια γραμμή ASCII κειμένου που αποτελείται από τα εξής στοιχεία:

  1. `transactionID`: μια συμβολοσειρά (μπορεί να έχει και μόνο ψηφία) που με μοναδικό τρόπο καθορίζει την κάθε τέτοια εγγραφή.
  2. `senderWalletID`: μια συμβολοσειρά που αποτελείται από γράμματα.
  3. `receiverWalletID`: μια συμβολοσειρά που αποτελείται από γράμματα.
  4. `value`: το ποσό την συναλλαγής (μπορείτε να υποθέσετε πως είναι ακέραιος αριθμός).

5. `date`: ημερομηνία που γίνεται το αίτημα συναλλαγής. Πρέπει να έχει την μορφή DD-MM-YYYY όπου το DD εκφράζει την ημέρα, το MM το μήνα, και το YYYY το χρόνο του αιτήματος.
6. `time`: ώρα στο 24ωρο που γίνεται το αίτημα συναλλαγής. Πρέπει να έχει την μορφή HH:MM όπου το HH εκφράζει την ώρα και το MM το λεπτό.

Ξεκινώντας, η εφαρμογή σας θα πρέπει να ανοίξει τα αρχεία `bitCoinBalancesFile` και `transactionsFile`, να διαβάσει μία-μία τις γραμμές και να αρχικοποιήσει και να αποθηκεύσει στη μνήμη τις δομές δεδομένων που θα χρησιμοποιεί κατά την εκτέλεση ερωτημάτων. Θα Πρέπει να ελέγχετε πως τα στοιχεία στα αρχεία είναι έγκυρα. Για παράδειγμα, αν στο αρχείο `bitCoinBalancesFile`, υπάρχουν χρήστες που έχουν στη κατοχή τους το ίδιο bitcoin, θα πρέπει να χειριστείτε το λάθος παρουσιάζοντας το κατάλληλο μήνυμα και βγαίνοντας από την εφαρμογή. Επίσης, αν κατά τη διάρκεια επεξεργασίας του `transactionsFile` εντοπίσετε μια συναλλαγή που δεν είναι έγκυρη, τότε η εφαρμογή θα πρέπει να παρουσιάσει ένα μήνυμα πως η συναλλαγή δεν είναι εφικτή και ακυρώνεται. Μια τέτοια περίπτωση είναι όταν σε μια συναλλαγή, ο αποστολέας δεν έχει αρκετά χρήματα στο `wallet` του για να γίνει η συναλλαγή.

Όταν η εφαρμογή τελειώσει την επεξεργασία των `bitCoinBalancesFile` και `transactionsFile` αρχείων, θα περιμένει είσοδο από το χρήστη από το πληκτρολόγιο. Ο χρήστης θα μπορεί να δίνει τις ακόλουθες εντολές:

- `/requestTransaction senderWalletID receiverWalletID amount date time`

Ο χρήστης ζητά να σταλούν `amount` χρήματα από τον χρήστη με userID `senderWalletID` στον χρήστη με userID `receiverWalletID`. Το `date` και `time` θα πρέπει να είναι μεταγενέστερα της τελευταίας συναλλαγής που έχει καταγράψει η εφαρμογή, αλλιώς απορρίπτεται το αίτημα. Αν δεν έχουν δοθεί `date` και `time`, η εφαρμογή χρησιμοποιεί τη τρέχουσα ώρα για την καταγραφή ώρας εκτέλεσης της συναλλαγής. Επίσης, η εφαρμογή θα πρέπει να ελέγχει αν υπάρχουν επαρκή χρήματα για να εκτελεστεί επιτυχώς η συναλλαγή. Αν υπάρχουν, ενημερώνει τις κατάλληλες δομές (δείτε παρακάτω) και παρουσιάζει μήνυμα επιτυχίας εκτέλεσης στον χρήστη με τις λεπτομέρειες της συναλλαγής. Αν δεν υπάρχουν, η εφαρμογή παρουσιάζει μήνυμα αποτυχίας στον χρήστη.

- `/requestTransactions senderWalletID receiverWalletID amount date time;`
  `senderWalletID2 receiverWalletID2 amount2 date2 time2;`
  `…`
  `senderWalletIDn receiverWalletIDn amountn daten timen;`

Ο χρήστης ζητά μια σειρά από συναλλαγές να εκτελεστούν. Οι συναλλαγές διαχωρίζονται με ελληνικό ερωτηματικό. Η εφαρμογή ελέγχει την εγκυρότητα κάθε συναλλαγής και αναλόγως ενημερώνει τις δομές δεδομένων τις και παρουσιάζει μήνυμα αποτελέσματος στον χρήστη.

- `/requestTransactions inputFile`

Ο χρήστης ζητά να εκτελεστούν συναλλαγές που περιγράφονται στο `inputFile`. Οι συναλλαγές διαχωρίζονται με ελληνικό ερωτηματικό και έχουν την ίδια μορφή με συναλλαγές των ερωτημάτων `requestTransaction(s)`. Η εφαρμογή ελέγχει την εγκυρότητα κάθε συναλλαγής και αναλόγως, ενημερώνει τις δομές δεδομένων και παρουσιάζει μήνυμα αποτελέσματος στον χρήστη.

- `/findEarnings walletID [time1][year1][time2][year2]`

Η εφαρμογή πρώτα επιστρέφει το συνολικό ποσόν που έχει λάβει μέσω συναλλαγών ο χρηστής με userID `walletID` (με επιλογή στο εύρος χρόνου ή/και ημερομηνίας). Αν υπάρχει ορισμός για [time1] θα πρέπει να υφίσταται και ορισμός για [time2]. Επίσης το ίδιο ισχύει και για την χρήση των μη υποχρεωτικών

παραμέτρων [year1] και [year2]. Στη συνεχεία, παρουσιάζει όλες τις εγγραφές συναλλαγών του χρήστη (ως παραλήπτης) που εκτελέστηκαν επιτυχώς μέσα στο συγκεκριμένο διάστημα. Αν δεν ορίζεται διάστημα, τότε η εφαρμογή θα παρουσιάζει την πλήρη ιστορία συναλλαγών όπου το `walletID` είναι παραλήπτης.

- `/findPayments walletID [time1][year1][time2][year2]`
Η εφαρμογή επιστρέφει το συνολικό ποσόν που έχει στείλει επιτυχώς μέσω συναλλαγών ο χρηστής με userID `walletID` (με επιλογή στο εύρος χρόνου ή/και ημερομηνίας). Στη συνεχεία, παρουσιάζει όλες τις εγγραφές συναλλαγών του χρήστη (ως αποστολέας) που εκτελέστηκαν επιτυχώς μέσα στο διάστημα που έχει δοθεί στη γραμμή εντολής. Αν δεν ορίζεται διάστημα, τότε η εφαρμογή θα παρουσιάζει την πλήρη ιστορία συναλλαγών όπου το `walletID` είναι αποστολέας.

- `/walletStatus walletID`
Η εφαρμογή επιστρέφει το τρέχον ποσόν που είναι στο wallet `walletID`.

-`/bitCoinStatus bitCoinID`
Η εφαρμογή επιστρέφει την αρχική αξία του bitcoin me ID `bitCoinID`, τον αριθμό των συναλλαγών στις οποίες έχει χρησιμοποιηθεί, και το ποσόν του `bitCoinID` που έχει μείνει `unspent` (δηλαδή δεν έχει χρησιμοποιηθεί ακόμα σε συναλλαγή).

Παράδειγμα output: `124 10 50`
σημαίνει πως το bitcoin 124 έχει χρησιμοποιηθεί σε 10 transactions ενώ 50 μονάδες της αξίας του δεν έχει χρησιμοποιηθεί ακόμα σε συναλλαγή.

- `/traceCoin bitCoinID`
Η εφαρμογή επιστρέφει την ιστορία συναλλαγών στο οποίο εμπλέκεται το bitcoin `bitCoinID`.

Παράδειγμα output:

```
/tracecoin 124
889 Maria Ronaldo 50 25-12-2018 20:08
776 Lionel Antonella 150 14-02-2019 10:05
```

Η Maria έδωσε στον Ronaldo 50 μονάδες στις 25/12/2018 (μέσω συναλλαγής #889) και ο Lionel 150 μονάδες στην Antonella στις 14/2/2019 (μέσω συναλλαγής #776).

-`/exit`
Έξοδος από την εφαρμογή. Βεβαιωθείτε πως ελευθερώνετε σωστά όλη τη δεσμευμένη μνήμη.

Δομές δεδομένων

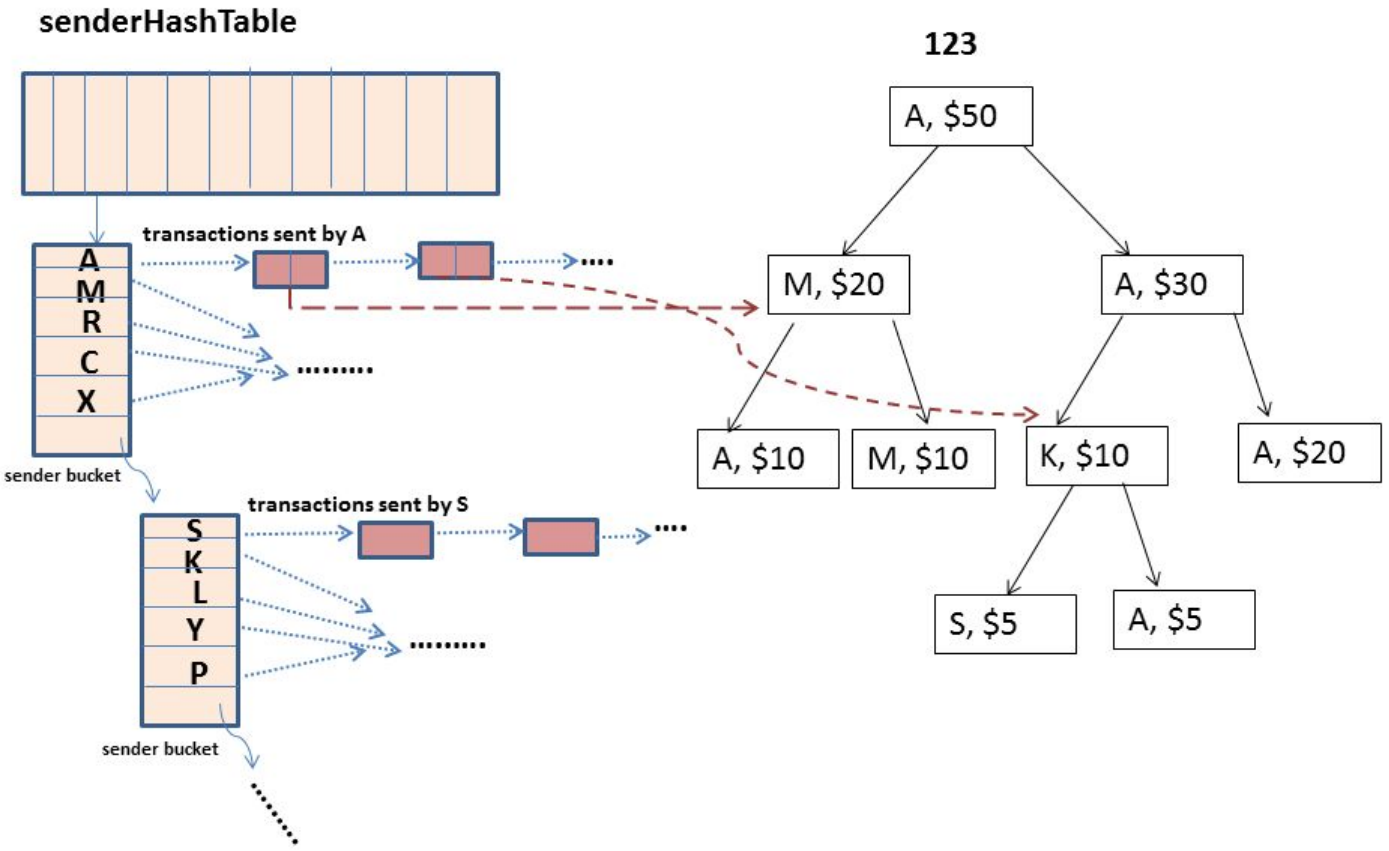
Για την υλοποίηση της εφαρμογής μπορείτε να χρησιμοποιήσετε C ή C++. Δεν μπορείτε να χρησιμοποιήσετε όμως την Standard Template Library (STL). Όλες οι δομές δεδομένων θα πρέπει να υλοποιηθούν από εσάς. Βεβαιωθείτε πως δεσμεύετε μόνο όση μνήμη χρειάζεστε, π.χ. η ακόλουθη τακτική δε συνιστάται:

```
int wallets[512];// store up to 512 wallets, but really we don't know how many
```

Επίσης βεβαιωθείτε πως απελευθερώνετε τη μνήμη σωστά και κατά την εκτέλεση του προγράμματός σας αλλά και κατά την έξοδο.

Για να ολοκληρώσετε την άσκηση θα χρειαστεί, μεταξύ άλλων, να υλοποιήσετε τις εξής δομές δεδομένων.

1. Δύο πίνακες κατακερματισμού (`senderHashTable` και `receiverHashTable`) που με index προσφέρουν γρήγορες προσπελάσεις σε στοιχεία συναλλαγών που έχουν εκτελεσθεί επιτυχώς. Ο πρώτος πίνακας κατακερματισμού ουσιαστικά παρέχει προσπέλαση για στοιχεία συναλλαγών όπου ο `walletID` είναι αποστολέας. Ο δεύτερος πίνακας κάνει το αντίστροφο: για κάθε `walletID` παρέχει προσπέλαση για στοιχεία συναλλαγών όπου ο `walletID` είναι παραλήπτης. Οι πίνακες κατακερματισμού θα χρησιμοποιούν κουβάδες για να εξυπηρετήσουν `walletIDs` που παρουσιάζουν «σύγκρουση»/collision (δηλαδή, το αποτέλεσμα της συνάρτησης κατακερματισμού οδηγεί στο ίδιο στοιχείο του hash table). Αν χρειάζονται πιο πολλοί από ένα κουβάδες για να αποθηκευτούν δεδομένα, δημιουργούνται δυναμικά και διατάσσονται σε μια λίστα.
2. Για κάθε `walletID` που γίνεται hashed σε ένα στοιχείο του `senderHashTable`, υπάρχει ένα σύνολο από συναλλαγές στις οποίες είναι αποστολέας. Αυτό το σύνολο τοποθετείται σε μια δυναμική linked list. Κάθε κόμβος της λίστας περιέχει προσπέλαση σε στοιχεία μιας συναλλαγής. Αντίστοιχες δομές θα πρέπει να φτιαχτούν και με το `receiverHashTable`.
3. Για κάθε `walletID`, μια δομή (δικής σας σχεδιαστικής επιλογής) που να περιέχει προσπέλαση σε στοιχεία του wallet (όπως π.χ., το συνολικό τρέχον balance του `walletID`, τα bitCoinIDs που έχει ο `walletID` στην κατοχή του και το υπόλοιπό τους, κλπ). Όταν η εφαρμογή επεξεργάζεται μια συναλλαγή, πρέπει να ελέγχει αν υπάρχουν αρκετά χρήματα συνολικά στο `wallet` του αποστολέα πριν προχωρήσει στην ολοκλήρωση της συναλλαγής. Στην περίπτωση που υπάρχουν, κατά την εκτέλεση της συναλλαγής, θα πρέπει να ενημερωθούν σωστά όλες οι κατάλληλες δομές δεδομένων.
4. Ένα δέντρο για κάθε `bitCoinID` το οποίο θα κρατάει όλη την ιστορία συναλλαγών του bitcoin και θα δημιουργείται δυναμικά καθώς επεξεργάζονται συναλλαγές που εμπλέκουν το συγκεκριμένο bitcoin. Η ρίζα-κόμβος του δέντρου θα περιέχει το `walletID` του αρχικού ιδιοκτήτη του `bitCoinID`, και την αρχική αξία του `bitCoinID`. Για την εκτέλεση μιας συναλλαγής, αν χρειαστούν κάποια χρήματα από το `bitCoinID` (από το `wallet` του αποστολέα), η εφαρμογή θα προσθέτει στο δέντρο ένα νέο κόμβο (φύλλο) στο δέντρο που θα περιέχει το `walletID` του παραλήπτη που έλαβε τα χρήματα και το ποσόν που έλαβε. Επίσης, σε περίπτωση που υπάρχει υπόλοιπο στο το bitcoin μετά την μεταφορά, θα πρέπει να προστεθεί ένας δεύτερος νέος κόμβος που κατοπτρίζει το υπόλοιπο του bitcoin που μένει στον αποστολέα. Για παράδειγμα, το Σχήμα 1 δείχνει ένα δέντρο για το bitcoinID 123 που ξεκινάει στην κατοχή του Α με αξία 50. Όταν ο Α στέλνει στον Μ $20, δημιουργείται ένας νέος κόμβος με στοιχεία Μ και $20 και ένας νέος κόμβος με στοιχεία Α και $30, διότι έμειναν $30 του bitcoin 123 στον Α. Με αυτόν τον τρόπο, μπορεί κανείς να διασχίσει το δέντρο ενός bitcoin και να βρει το ιστορικό των συναλλαγών στο οποίο εμπλέκεται το συγκεκριμένο bitcoin. Το άθροισμα των ποσών των φύλλων του δέντρου θα πρέπει πάντα να είναι ίσο με την αρχική αξία του bitcoin.



Σχήμα 1: Παράδειγμα Καποιων Δομών για την εφαρμογή bitCoin

Ο στόχος σας σε αυτή την άσκηση είναι να υπάρξει όσο το δυνατόν *μικρότερη επικάλυψη στοιχείων (data duplication)*. Βεβαιωθείτε πως για κάθε υπο-πρόβλημα που χρειάζεται να επιλύσετε κατά την υλοποίηση της άσκησης, χρησιμοποιείτε τον πιο αποτελεσματικό αλγόριθμο ή δομή δεδομένων. Όποιες σχεδιαστικές αποφάσεις και επιλογές κάνετε κατά την υλοποίηση, θα πρέπει να τις περιγράψετε στο README, στα παραδοτέα.

Παραδοτέα

- Μια σύντομη και περιεκτική εξήγηση για τις επιλογές που έχετε κάνει στο σχεδιασμό του προγράμματός σας. 1-2 σελίδες ASCII κειμένου είναι αρκετές. Συμπεριλάβετε την εξήγηση και τις οδηγίες για το compilation και την εκτέλεση του προγράμματός σας σε ένα αρχείο README μαζί με τον κώδικα που θα υποβάλετε.
- Οποιαδήποτε πηγή πληροφορίας, συμπεριλαμβανομένου και κώδικα που μπορεί να βρήκατε στο Διαδίκτυο ή αλλού θα πρέπει να αναφερθεί στον πηγαίο κώδικά σας αλλά και στο παραπάνω README.
- Όλη η δουλειά σας (πηγαίος κώδικας, Makefile και README) σε ένα tar.gz file με ονομασία OnomaEponymoProject1.tar.gz. Προσοχή να υποβάλετε μόνο κώδικα, Makefile, README και όχι τα binaries.

Διαδικαστικά

- Για επιπρόσθετες ανακοινώσεις, παρακολουθείτε το forum του μαθήματος στο piazza.com. Η πλήρης διεύθυνση είναι https://piazza.com/uoa.gr/spring2019/k24/home. Η παρακολούθηση του φόρουμ στο Piazza είναι υποχρεωτική.
- Το πρόγραμμά σας θα πρέπει να γραφεί σε C (ή C++). Στην περίπτωσή που χρησιμοποιήσετε C++ δεν μπορείτε να χρησιμοποιήσετε τις έτοιμες δομές της Standard Template Library (STL). Σε κάθε περίπτωση το πρόγραμμά σας θα πρέπει να τρέχει στα Linux workstations του Τμήματος.
- Ο κώδικάς σας θα πρέπει να αποτελείται από τουλάχιστον δύο (και κατά προτίμηση περισσότερα) διαφορετικά αρχεία. Η χρήση του separate compilation είναι επιτακτική και ο κώδικάς σας θα πρέπει να έχει ένα Makefile.
- Βεβαιωθείτε πως ακολουθείτε καλές πρακτικές software engineering κατά την υλοποίηση της άσκησης. Η οργάνωση, η αναγνωσιμότητα και η ύπαρξη σχολίων στον κώδικα αποτελούν κομμάτι της βαθμολογίας σας.
- Στα πρώτα δύο μαθήματα κυκλοφορεί hard-copy λίστα στην τάξη στην οποία θα πρέπει οπωσδήποτε να δώσετε το όνομά σας και το Unix user-id σας. Με αυτό τον τρόπο μπορούμε να γνωρίζουμε ότι προτίθεστε να υποβάλετε την παρούσα άσκηση και να προβούμε στις κατάλληλες ενέργειες για την τελική υποβολή της άσκησης.

Άλλες σημαντικές παρατηρήσεις

- Οι εργασίες είναι ατομικές.
- Όποιος υποβάλει / παρουσιάσει κώδικα που δεν έχει γραφτεί από την ίδια/τον ίδιο μηδενίζεται στο μάθημα.
- Αν και αναμένεται να συζητήσετε με φίλους και συνεργάτες το πώς θα επιχειρήσετε να δώσετε λύση στο πρόβλημα, αντιγραφή κώδικα (οποιασδήποτε μορφής) είναι κάτι που δεν επιτρέπεται. Οποιοσδήποτε βρεθεί αναμεμειγμένος σε αντιγραφή κώδικα απλά παίρνει μηδέν στο μάθημα. Αυτό ισχύει για όσους εμπλέκονται ανεξάρτητα από το ποιος έδωσε/πήρε κλπ.
- Οι ασκήσεις προγραμματισμού μπορούν να δοθούν με καθυστέρηση το πολύ 3 ημερών και με ποινή 5% για κάθε μέρα αργοπορίας. Πέραν των 3 αυτών ημερών, δεν μπορούν να κατατεθούν ασκήσεις.