Δεύτερη Σειρά ασκήσεων
Ημερομηνία Παράδοσης: 12 Μαΐου 2016, 12 μ.μ.

Οι παρακάτω ασκήσεις πρέπει να παραδοθούν μέχρι τις 12/5/2016, 12 μ.μ. . Στην υλοποίηση των κλάσεων σας δεν θα πρέπει να έχετε public πεδία. Βαθμοί θα αφαιρεθούν για προγράμματα που δεν είναι καλά γραμμένα, δηλαδή δεν είναι σωστά στοιχισμένα ή δεν έχουν καλά επιλεγμένα ονόματα μεταβλητών ώστε να διαβάζονται εύκολα.

Άσκηση 1

Για την άσκηση αυτή θα υλοποιήσετε ένα δυαδικό δέντρο αναζήτησης (Binary Search Tree). Μπορείτε να διαβάσετε περισσότερα για το δυαδικό δέντρο στο σύνδεσμο στη σελίδα του μαθήματος, καθώς και τους αλγορίθμους για τις μεθόδους που πρέπει να υλοποιήσετε.

Κατασκευάσετε την κλάση **BinarySearchTree** που υλοποιεί ένα δυαδικό δέντρο αναζήτησης. Η κλάση θα πρέπει να έχει τις εξής μεθόδους:

- Μία μέθοδο **contains** η οποία παίρνει σαν όρισμα ένα ακέραιο και επιστρέφει μια boolean τιμή αν αυτός ο ακέραιος ανήκει ή όχι στο δέντρο.
- Μία μέθοδο **insert** η οποία παίρνει σαν όρισμα ένα ακέραιο και τον προσθέτει στην κατάλληλη θέση στο δέντρο.
- Μία μέθοδο **print** η οποία τυπώνει τις τιμές του δέντρου σε αύξουσα σειρά. Για την υλοποίηση αυτής της μεθόδου συνίσταται να χρησιμοποιήσετε μια αναδρομική μέθοδο, όπου για κάθε κόμβο πρώτα τυπώνουμε το αριστερό παιδί, μετά την τιμή του κόμβου, και μετά το δεξί παιδί.
- Μία μέθοδο **remove** η οποία παίρνει σαν όρισμα ένα ακέραιο και εφόσον υπάρχει στο δέντρο τον αφαιρεί από τον δέντρο. Η μέθοδος αυτή είναι η πιο δύσκολη, με διάφορες υπο-περιπτώσεις που πρέπει να εξετάσετε. Η κάθε υπο-περίπτωση βαθμολογείται ξεχωριστά, οπότε βάλετε σχόλια μέσα στην μέθοδο που να προσδιορίζουν ποια περίπτωση χειρίζεται το κάθε κομμάτι του κώδικα.

Για την υλοποίηση θα χρειαστείτε και μια κλάση **TreeNode** η οποία κρατάει πληροφορία για ένα κόμβο του δέντρου. Η κλάση θα κρατάει την ακέραια τιμή, καθώς και τα παιδιά και τον πατέρα του κόμβου. Επίσης θα έχει τις εξής μεθόδους:

- Μεθόδους **πρόσβασης** και **μετάλλαξης** (accessor και mutator μεθόδους) για **όλα** τα πεδία της κλάσης.
- Μία μέθοδο **isLeaf** που επιστρέφει boolean τιμή για το αν ο κόμβος είναι φύλο ή όχι.
- Μια μέθοδο **isRightChild** που επιστρέφει μια boolean τιμή για το αν ο κόμβος είναι δεξί παιδί του πατέρα του ή όχι.
- Μια μέθοδο **isLeftChild** που επιστρέφει μια boolean τιμή για το αν ο κόμβος είναι αριστερό παιδί του πατέρα του ή όχι.

Ορίστε μια μέθοδο main μέσα στην BinarySearchTree για να τεστάρετε την κλάση σας. Για να σας βαθμολογήσουμε θα φτιάξουμε μια κλάση **BSTTest** που θα χρησιμοποιεί την BinarySearchTree που θα παραδώσετε. Ένα παράδειγμα δίνεται στην σελίδα του μαθήματος.

Υποδείξεις:

- Μπορεί να σας φανεί χρήσιμο να φτιάξετε στην BinarySearchTree μία private μέθοδο **find** η οποία παίρνει σαν όρισμα ένα ακέραιο και επιστρέφει τον κόμβο του δέντρου που περιέχει αυτή την τιμή ή αλλιώς null.
- Εκτός από την print και άλλες μέθοδοι μπορούν να υλοποιηθούν με αναδρομή. Αυτή είναι μια αποδεκτή υλοποίηση.

Άσκηση 2

Για την άσκηση αυτή θα υλοποιήσετε σε Java ένα πρόγραμμα που θα προσομοιώνει την κυκλοφορία αυτοκινήτων πάνω σε ένα οδικό δίκτυο. Το οδικό δίκτυο αποτελείται από N+1 πόλεις {0,1,…,N} που συνδέονται με δρόμους. Όλοι οι δρόμοι είναι **μονόδρομοι**, και έχουν κατεύθυνση από την «μικρότερη» πόλη προς την μεγαλύτερη. Υπάρχουν δρόμοι μεταξύ «γειτονικών πόλεων» (i, i+1), για i=0,1,…,N-1. Επίσης σε κάθε πόλη i <= N-3 υπάρχει και ένας δρόμος προς κάποια άλλη πόλη j > i+1 που ο προορισμός του επιλέγεται τυχαία. Ο δρόμος (i,j) έχει μήκος j-i χιλιόμετρα.

Πάνω σε αυτό το δίκτυο τοποθετούμε Κ αυτοκίνητα στην πόλη 0. Το κάθε αυτοκίνητο έχει μια ταχύτητα η οποία στην προσομοίωση μας ορίζεται από τον χρόνο που παίρνει στο όχημα για να διανύσει 1 χιλιόμετρο. Αυτή είναι μια τυχαία τιμή μεταξύ 1 και 10. Όλα τα αυτοκίνητα έχουν στόχο να πάνε στην πόλη N.

Η προσομοίωση θα γίνεται σε βήματα, όπου κάθε βήμα αντιστοιχεί σε μία χρονική στιγμή. Σε κάθε βήμα, αν υπάρχουν αυτοκίνητα σε κάποια πόλη, διαλέγουμε για κάθε όχημα έναν από τους εξερχόμενους δρόμους από αυτή την πόλη και τοποθετούμε το όχημα στον δρόμο. Στο σημείο αυτό (όπου όλα τα οχήματα είναι στο δρόμο) τυπώνουμε την κίνηση στους δρόμους του δικτύου μας. Στη συνέχεια για κάθε όχημα που είναι στο δρόμο, κάνουμε ένα βήμα (δηλαδή μειώνουμε το χρόνο να φτάσουν στην επόμενη πόλη κατά ένα). Τα οχήματα στον δρόμο είναι σε μία ουρά. Για να φτάσει ένα αυτοκίνητο στο τέλος του δρόμου, θα πρέπει να έχουν φτάσει τα οχήματα που προηγούνται από αυτό. Όταν κάποιο ή κάποια οχήματα φτάσουν στο τέλος του δρόμου τα αφαιρούμε από την ουρά και τα τοποθετούμε στην πόλη τερματισμού του δρόμου. Η προσομοίωση σταματάει όταν όλα τα οχήματα φτάσουν στην πόλη N.

Για την υλοποίηση της προσομοίωσης θα υλοποιήσετε τις εξής πέντε κλάσεις: **Car**, **Road**, **City**, **RoadNetwork** και **TrafficSimulation**.

Η κλάση **Car** κρατάει πληροφορίες για ένα όχημα. Για ένα όχημα χρειαζόμαστε να κρατάμε την ταχύτητα του (**timePerKlm**) η οποία παίρνει τιμή στον constructor. Επίσης όταν το όχημα είναι στο δρόμο πρέπει να ξέρουμε πόσο χρόνο (**timeToGo**) χρειάζεται ακόμη για να φτάσει στο τέλος του δρόμου. Αυτή η μεταβλητή αρχικοποιείται κατάλληλα όταν τοποθετούμε το όχημα στο δρόμο, και μειώνεται όταν το μετακινούμε κατά ένα. Ορίστε τις αντίστοιχες μεθόδους, καθώς και όποια άλλη μέθοδο χρειάζεστε.

Η κλάση **Road** κρατάει πληροφορία για ένα δρόμο. Χρειάζεται να κρατάει πληροφορία για τις πόλεις που είναι η αφετηρία και ο τερματισμός του δρόμου, την απόσταση, και την ουρά με τα αυτοκίνητα που είναι στον δρόμο αυτό. Η κλάση θα πρέπει να έχει και μια μέθοδο **computeAggregateTravelTime** η οποία θα επιστρέφει μια τιμή η οποία θα μας λέει πόσο «καλός» είναι ο δρόμος δεδομένων των οχημάτων που είναι σε αυτό το δρόμο. Π.χ., μπορείτε να επιστρέψετε την μέση ταχύτητα των οχημάτων στο δρόμο. Ή το μέγιστο χρόνο ανά χιλιόμετρο που απομένει στα οχήματα, ή κάτι άλλο που θα αποφασίσετε εσείς (σημειώσετε την επιλογή σας με σχόλια στον κώδικα). Μπορεί επίσης να εξαρτάται και από το όχημα που θέλουμε να προσθέσουμε στο δρόμο.
Η Road θα έχει επίσης μια μέθοδο **moveCars** η οποία θα μετακινεί τα οχήματα κατά ένα βήμα. Μετά την μετακίνηση, ξεκινώντας από την αρχή της ουράς αφαιρεί από την ουρά όσα οχήματα έχουν φτάσει στο τέλος του δρόμου και τα τοποθετεί στην πόλη τερματισμού του δρόμου. Σταματάει μόλις βρει το πρώτο όχημα στην ουρά που έχει μη μηδενικό timeToGo.
Χρειαζόμαστε επίσης μια μέθοδο που τυπώνει την κίνηση στο δρόμο, και μπορείτε να ορίσετε και όποιες άλλες μεθόδους χρειάζεστε.

Η κλάση **City** κρατάει πληροφορία για μία πόλη. Η κάθε πόλη χαρακτηρίζεται από ένα αριθμό που δίνεται στον constructor. Κρατάει επίσης ένα ArrayList με τους δρόμους που ξεκινούν από αυτή την πόλη και μια λίστα (ουρά) με τα οχήματα που είναι σε αυτή την πόλη. Η βασική μέθοδος της City είναι η **routeCars** η οποία για κάθε όχημα επιλέγει τον καλύτερο δρόμο και το τοποθετεί σε αυτόν. Ορίστε και όποιες άλλες μεθόδους χρειάζεστε.

Η κλάση **RoadNetwork** κρατάει πληροφορία για το οδικό δίκτυο. Έχει ένα πίνακα με τις πόλεις και ένα ArrayList με τους δρόμους. Το δίκτυο όπως το περιγράψαμε δημιουργείται στον constructor. Χρειάζονται ακόμη μέθοδοι για να μετακινεί τα οχήματα στους δρόμους, να δρομολογεί τα οχήματα στις πόλεις, να τυπώνει την κίνηση και να υπολογίζει τον αριθμό των οχημάτων πού φτάσανε στον τελικό προορισμό. Μπορείτε να ορίσετε και άλλες μεθόδους που χρειάζεστε.

Η **TrafficSimulation** υλοποιεί την προσομοίωση. Σας δίνεται ενδεικτικά μια υλοποίηση στη σελίδα του μαθήματος. Μπορείτε να κάνετε κάποιες τροποποιήσεις αν το κρίνετε απαραίτητο.

Υποδείξεις: Για την υλοποίηση της ουράς αυτοκινήτων στην κλάση Road και της λίστας αυτοκινήτων στην κλάση City προτείνεται να χρησιμοποιήσετε την δομή LinkedList που σας δίνει η Java. Συγκερκιμένα θα χρειαστείτε τις μεθόδους addLast, getFirst και remove (ή pop). Η δομή αυτή έχει το πλεονέκτημα ότι είναι εύκολο να προσθέσεις στο τέλος της ουράς και να αφαιρέσεις την αρχή. Γενικά να είσαστε προσεκτικοί να μην αφαιρείτε στοιχεία από μια λίστα ενώ την διατρέχετε.