Αναφορά Παράδοσης

Εργασία 1 – Δομές Δεδομένων (22-23)

1.

(Α) Υλοποίηση διεπαφής StringQueue με λίστα μονής σύνδεσης.

Αρχικά τροποποιήσαμε τη διεπαφή StringQueue που μας δώθηκε, έτσι ώστε να δέχεται στοιχεία του τύπου T (generics) αντί για τύπου String. Σκεφτήκαμε πως έτσι η δομή μας θα είναι πιο επεκτάσιμη. Δημιουργούμε την κλάση StringQueueImpl και κρατάει τρείς μεταβλητές: έναν δείκτη στον πρώτο κόμβο (head), έναν δείκτη στον τελευταίο κόμβο (tail) και το μήκος της ουράς (length). Στη μέθοδο isEmpty αρκεί να ελέγξουμε άν ο tail είναι κενός. Η μέθοδος put μας κάνει δύο δουλειές: 1) αν η ουρά είναι άδεια. φτιάχνει ένα νέο κόμβο, τον θέτει να έιναι head και tail ταυτόχρονα και μετά αυξάνει το μήκος, 2) αν υπάρχουν κόμβοι στην ουρά, φτιάχνει έναν νέο κόμβο, τον θέτει ώς τον επόμενο του tail, γίνεται αυτός ο καινούργιος tail και μετά αυξάνει το μήκος. Η μέθοδος get θέτει ως head τον επόμενο του head, μειώνει το μήκος και επιστρέφει τα στοιχεία του παλιού head. Στην peek απλά επιστρέφουμε τα δεδομένα του head. Στην size απλά επιστρέφουμε το μήκος της ουράς. Τέλος, η printQueue διατρέχει σειριακά την ουρά μέχρι να φτάσει στον tail και τυπώνει τα στοιχεία του καθε κόμβου.

(Α) Υλοποίηση διεπαφής StringStack με λίστα μονής σύνδεσης.

Ομοίως και εδώ κάναμε υλοποίηση με generics. Δημιουργούμε την κλάση StringStackImpl και κρατάει δύο μεταβλητές: έναν δείκτη στον πρώτο κόμβο (top) και το μήκος της στοίβας (length). Στη μέθοδο isEmpty αρκεί να ελέγξουμε άν ο top είναι κενός. Στην push δημιουργούμε έναν νέο κόμβο, του θέτουμε τον top ώς επόμενο, γίνεται αυτός ο καινούργιος top και αυξάνουμε το μήκος. Η μέθοδος pop θέτει ως top τον επόμενο του top, μειώνει το μήκος και επιστρέφει τα στοιχεία του παλιού top. Στην peek απλά επιστρέφουμε τα δεδομένα του top. Στην size απλά επιστρέφουμε το μήκος της ουράς. Τέλος, η printQueue διατρέχει σειριακά την στοίβα μέχρι να φτάσει στον κόμβο που έχει επόμενο το null και τυπώνει τα στοιχεία του καθε κόμβου.

(Γ) Υλοποίηση διεπαφής StringQueue με έναν μόνο δείκτη.

Ακολουθήσαμε την υπόδειξη και κάναμε την υλοποίηση με κυκλική λίστα μονής σύνδεσης. Έτσι, κρατήσαμε τον δείκτη tail και απλά φροντίσαμε ο επόμενός του κόμβος να δείχνει στην αρχή της ουράς (δλδ ισχύει head == tail.getNext()). Οι isEmpty, peek, length, και printQueue ακολουθούν την ίδια λογική με την προηγούμενη υλοποίηση. Εδώ η μέθοδος put μας κάνει δύο διαφορετικές λειτουργίες: 1) αν η ουρά είναι άδεια, φτιάχνει ένα νέο κόμβο, τον κάνει να δείχνει στον εαυτό του και γίνεται αυτός ο tail, 2) αν υπάρχουν κόμβοι στην ουρά, φτιάχνει ένα νεό κόμβο, τον κάνει να δείχνει στον επόμενο του tail (δηλαδή στην αρχή), θέτει ως επόμενο του tail τον καινούργιο και ο καινούργιος γίνεται ο tail. Στην get

έχουμε 3 περιπτώσεις: 1) η ουρά έχει μήκος = 1, οπότε ο tail διαγράφεται (γίνεται null), 2) η ουρά έχει μήκος = 2, οπότε ο επόμενος του tail γίνεται ο εαυτός του, 3) η ουρά έχει μήκος > 2, άρα ο επόμενος του tail γίνεται ο αμέσως επόμενος κόμβος απο αυτόν στον οποίο δείχνει.

2.

(Β) Χρήση στοίβας StringStackImpl

Δημιουργήσαμε μια κλάση Point, κάθε αντικείμενο της οποίας αποτελεί ένα ζεύγος τιμών (συντεταγμένες) x και y. Στην στοίβα pathToExit αποθηκεύουμε αντικείμενα τύπου Point ώστε να μπορούμε να αποθηκεύσουμε τις συντεταγμένες των κόμβων του πίνακα labyrinth τις οποίες έχουμε επισκευτεί. Αρχικά τοποθετούμε στην στοίβα το σημείο στο οποίο βρίσκεται το «E» (pathToExit.push(new Point(startX, startY);). Έπειτα, κάθε φορά που εκτελείται μια κίνηση (up,down,left,right) , τοποθετούμε στην κορυφή της στοίβας (με την μέθοδο push) το σημείο που πρόκειται να επισκεφτούμε (πχ για την κίνηση up εκτελείται η εντολή: pathToExit.push(new Point(x-1,y);). Όταν δεν μπορούμε να κάνουμε άλλη κίνηση και αναγκαστούμε να κάνουμε backtracking και να επιστρέψουμε στο προηγούμενο σημείο του πίνακα, αφαιρούμε από την στοίβα (με την μέθοδο pop) το τελευταίο της στοιχείο (top) και συνεχίζουμε την αναζήτηση από το στοιχείο που έχει πλέον βρεθεί στην κορυφή της στοίβας (solveLabyrinth(Labyrinth, pathToExit.peek().x, pathToExit.peek().y,visitedPoints).