Δομές Δεδομένων

Εργασία 1η

Μέρος Α:

Αρχικά δημιουργήσαμε τις κλάσεις StringStackImpl και StringQueueImpl οι οποίες υλοποιούν τις διεπαφές StringStack και StringQueue αντίστοιχα. Και για τις 2 χρησιμοποιήσαμε τη λίστα την οποία δημιουργήσαμε στο εργαστήριο με μία τροποποίηση στη μέθοδο print, έτσι ώστε να δέχεται ως όρισμα κάποιο stream, καθώς αυτό απαιτείται για την υλοποίηση των διεπαφών.

Στη **StringStackImpl** δημιουργήσαμε τη μέθοδο push χρησιμοποιώντας τη μέθοδο εισαγωγής στο μπροστά μέρος της λίστας(insertAtFront), καθώς στη στοίβα το κάθε καινούργιο στοιχείο πρέπει να προστίθεται σαν το πρώτο. Για τη δημιουργία της μεθόδου pop χρησιμοποιήσαμε τη μέθοδο της αφαίρεσης από το μπροστά μέρος της λίστας(removeFromFront) μετά από έλεγχο αν η στοίβα είναι κενή, αφού στη στοίβα πάντα αφαιρούμε το πρώτο στοιχείο, το οποίο είναι στο μπροστά μέρος της λίστας.

Κατόπιν, δημιουργήσαμε έναν μετρητή, ο οποίος αυξάνεται όταν προστίθεται ένα στοιχείο στη στοίβα και μειώνεται όταν κάποιο αφαιρείται. Αυτός ο μετρητής μας χρησίμευσε στη μέθοδο size, η οποία επιστρέφει το μέγεθος της στοίβας, δηλαδή τον αριθμό των στοιχείων που περιέχει.

Επιπρόσθετα δημιουργήσαμε τη μέθοδο isEmpty χρησιμοποιώντας την ομώνυμη μέθοδο της List για να ελέγχουμε αν η στοίβα είναι άδεια.

Τέλος, δημιουργήσαμε τη μέθοδο peek, με σκοπό να εμφανίζει το πρώτο στοιχείο της στοίβας, αφού πρώτα ελέγξει ότι δεν είναι άδεια. Αυτό πραγματοποιείται καλώντας τη μέθοδο getFirst της List.

Στη **StringQueueImpl** δημιουργήσαμε τη μέθοδο put χρησιμοποιώντας τη μέθοδο εισαγωγής στο πίσω μέρος της λίστας(insertAtBack), καθώς στην ουρά το κάθε καινούργιο στοιχείο πρέπει να προστίθεται σαν το τελευταίο. Για τη δημιουργία της μεθόδου get χρησιμοποιήσαμε τη μέθοδο της αφαίρεσης από το μπροστά μέρος της λίστας(removeFromFront) μετά από έλεγχο αν η ουρά είναι κενή, αφού στην ουρά πάντα αφαιρούμε το πρώτο στοιχείο, το οποίο είναι στο μπροστά μέρος της λίστας.

Και εδώ δημιουργήσαμε μετρητή με τον ίδιο σκοπό με της StringStackImpl, δηλαδή για να υλοποιηθεί η μέθοδος size. Αντίστοιχα φτιάξαμε, με τον ίδιο σκοπό με αυτόν της προαναφερθείσας κλάσης, τη μέθοδο isEmpty και τη μέθοδο peek.

Μέρος Β:

Κατά την εκκίνηση του προγράμματος **Thiseas**, φορτώνεται το αρχείο που υποδεικνύουμε σαν όρισμα κατά την εκτέλεση, χρησιμοποιώντας ένα Scanner. Στη συνέχεια, αξιοποιώντας τη μέθοδο nextInt, αποθηκεύουμε τους 4 αριθμούς σε μεταβλητές με σκοπό να τους επαληθεύσουμε. Κατόπιν ξεκινάμε μια διαδικασία αποθήκευσης των δεδομένων του λαβύρινθου σε έναν πίνακα, ελέγχοντας παράλληλα αν διαφέρουν τα χαρακτηριστικά του από εκείνα της κεφαλίδας. Σε περίπτωση που δεν ταιριάζουν, το πρόγραμμα ενημερώνει το χρήστη για το λάθος που προέκυψε και τερματίζεται.

<u>Προσοχή! Το αρχείο με το λαβύρινθο θα πρέπει να έχει την είσοδο με αγγλικό E, για να αναγνωριστεί ως σωστό, καθώς δεν υπάρχει υποστήριξη ελληνικών χαρακτήρων!</u>

Αφού ολοκληρωθεί ο έλεγχος και η αποθήκευση στον πίνακα, αρχίζει η διαδικασία έυρεσης εξόδου. Ξεκινώντας, αρχικοποιούμε μια στοίβα(**stoiva**) χρησιμοποιώντας την StringStackImpl από το μέρος Α. Η επίλυση του λαβίρυνθου ουσιαστικά βηματοδοτείται από τη μέθοδο **LabExit**, η οποία λαμβάνει ως ορίσματα τις συντεταγμένες της εισόδου σε αυτόν. Καλώντας, λοιπόν, τη **LabExit**, αποθηκεύουμε τις συντεταγμένες σε ένα array τύπου int, το οποίο μας θα μας διευκολύνει στην κλήση μεθόδων. Βάζουμε το σημείο εκκίνησης στη στοίβα με την κλήση της **SaveCoordinates** και κατόπιν μπαίνουμε σε ένα while loop, με τη συνθήκη η στοίβα να μην είναι άδεια! Σε αυτή τη φάση του προγράμματος, παίρνουμε τις τελευταίες συντεταγμένες της στοίβας με τη μέθοδο **LoadCoordinates** και ελέγχουμε αν έχουμε βρει έξοδο μέσω της συνάρτησης **isExit**, η οποία ελέγχει αν το σημείο το οποίο ελέγχουμε βρίσκεται στην περίμετρο του λαβύρινθου και το περιεχόμενό του είναι το "0". Αν το σημείο αυτό είναι όντως σημείο εξόδου ο χρήστης ενημερώνεται με σχετικό μήνυμα και το πρόγραμμα τερματίζεται. Σε διαφορετική περίπτωση ελέγχει σε ποιά κατεύθυνση θα μπορούσε να υπάρχει η έξοδος, μέσω της μεθόδου **addNeighbors**, και αποθηκεύει στη στοίβα κάθε δυνατή επιλογή, χωρίς να ξαναπροσθέσει το προηγούμενο σημείο από το οποίο πέρασε, αφού αλλάζει το περιερχόμενό του σε "2", άρα ο προηγούμενος έλεγχος πραγματοποιείται υπό τη συνθήκη ότι το περιεχόμενο είναι διαφορετικό του 2. Αν λοιπόν αδειάσει η στοίβα και βγεί από την επανάληψη σημαίνει ότι δεν έχει βρεθεί κάποια έξοδος και ο χρήστης ενημερώνεται με σχετικό μήνυμα. Τέλος, κλείνουμε το αρχείο του λαβύρινθου και το πρόγραμμα τερματίζεται.

Μέρος Γ:

Στην κλάση StringQueueWithOnePointer, που δημιουργήσαμε, έχουμε φτιάξει μια κυκλική λίστα που αναπαριστά την ουρά, με σκοπό να έχει έναν μόνο δείκτη, συγκεκριμένα το **firstNode**. Για την υλοποίησή της χρησιμοποιήσαμε την κλάση ListNode που δημιουργήσαμε στο εργαστήριο.

Μέσα στην κλάση αυτή δημιουργήσαμε τη μέθοδο **insert**, η οποία εισάγει ένα στοιχείο στην ουρά. Αυτό πραγματοποιείται ως εξής: Αρχικά ελέγχει αν η ουρά είναι άδεια μέσω της μεθόδου **isEmpty**. Αν είναι, εισάγεται το στοιχείο που έχουμε δώσει σαν όρισμα και ο δείκτης firstNode δείχνει το στοιχείο που εισήχθη, καθώς και ο firstNode.nextNode, δηλαδή ο επόμενος κόμβος από τον πρώτο. Σε διαφορετική περίπτωση, δηλαδή αν υπάρχουν στοιχεία μέσα, ο κόμβος που εισάγουμε συνδέεται ως εξής: Ο επόμενος του εισακτέου κόμβου θα είναι ο επόμενος του firstNode,ο εισακτέος θα γίνει ο επόμενος του firstNode και τελικά το firstNode θα γίνει ο εισακτέος ώστε να μπορούμε να βάζουμε και άλλους κόμβους.
Επίσης δημιουργήσαμε τη μέθοδο **remove** με την όποια αφαιρούμε πάντα το στοιχειό μετά το firstNode,λαμβάνοντας υπόψη την περίπτωση που δεν υπάρχει κανένα στοιχειό (**isEmpty)** ή υπάρχει μόνο ένα στοιχείο οπότε το firstNode γίνεται null.Αν η ουρά έχει περισσότερα από ένα στοιχεία ,τότε αφαιρούμε το κόμβο,θέτοντας τον επόμενο του firstNode τον μεθεπόμενο του.
Θα μπορούσε να πει κανείς ότι είναι λάθος που εισάγουμε φαινομενικά από την αρχή και αφαιρούμε από το τέλος , αλλά χρησιμοποιήσαμε κυκλική λίστα ώστε να μην υπάρχει πρώτο και τελευταίο στοιχειό Έτσι δε χρειάστηκε να κάνουμε σάρωση της λίστας για τις λειτουργίες αυτές,έχοντας αντι για O(N) πολυπλοκότητα την O(1) δηλαδή σταθερό κόστος!
Για τη μέθοδο **print** αναγκαστικά διατρεξαμε τη λίστα με επανάληψη ώστε να τυπώσουμε τα στοιχεία έτσι έχουμε O(N) μόνο στη **print,**όμως πρώτα ελέγξαμε αν είναι άδεια (**isEmpty)** και με ξεχωριστή εντολή έξω από τη loop τυπώσαμε το firstNode.
Σημειώνουμε ότι για την **isEmpty** που είναι τύπου boolean η συνθήκη είναι αν το firstNode ισούται null.