



ΕΡΓΑΣΙΑ 1

Δομές Δεδομένων

Ονοματεπώνυμο:

Κεχριώτη Ελένη

2023-2024

Μέρος Α

Για την υλοποίηση των διεπαφών `StringStackImpl` και `StringQueueImpl` χρησιμοποιήσαμε τη class `Node` η οποία υλοποιεί μια λίστα μονής σύνδεσης. Κάθε αντικείμενο τάξης `Node` περιέχει δεδομένα τύπου `T` (generic type), τα οποία εκχωρούνται στη στοίβα, και μια αναφορά τύπου `Node`, η οποία δείχνει στο επόμενο στοιχείο/ κόμβο της λίστας.

StringStackImpl

Όπως θα δείτε στον κώδικα του αρχείου `StringStackImpl.java` χρησιμοποιείται ένας δείκτης `top`, που δείχνει στο πρώτο στοιχείο της στοίβας, στο τελευταίο που ωθείται στη στοίβα. Στον constructor θέτουμε το `top` κάθε στοίβας που δημιουργείται ως `null`.

isEmpty(): επιστρέφει μια τιμή αληθής ή ψευδής αν το `top = null`, δηλαδή αν δεν υπάρχουν στοιχεία στη στοίβα, είτε επειδή δεν έχει εισαχθεί ακόμα κάποιο, είτε γιατί έχουν αφαιρεθεί όλα.

push(T item): δέχεται σαν όρισμα ένα αντικείμενο τύπου `T` και δεν επιστρέφει κάτι. Αν η στοίβα είναι άδεια τότε δημιουργούμε ένα νέο αντικείμενο τύπου `Node` με δεδομένα `item` και αναφορά στο επόμενο στοιχείο `null`, αφού δεν υπάρχει κάποιο. Αντίθετα, αν υπάρχει τουλάχιστον ένα στοιχείο ήδη τότε θέτουμε ως `top` ένα νέο στοιχείο τύπου `Node` με δεδομένα `item` και αναφορά στο επόμενο στοιχείο `top`, το στοιχείο που βρισκόταν πριν στη κορυφή της στοίβας.

pop(): αφαιρεί το πρώτο στοιχείο της στοίβας και επιστρέφει τα δεδομένα του. Αν η στοίβα είναι άδεια, τότε δεν γίνεται να αφαιρέσουμε στοιχείο της και πετάει εξαίρεση τύπου `NoSuchElementException`. Αντίθετα, σε μια μεταβλητή `data` αποθηκεύουμε τα δεδομένα του στοιχείου κορυφής της στοίβας, για να τα επιστρέψουμε μετά. Δημιουργούμε μια προσωρινή μεταβλητή `t`, η οποία αποθηκεύει το επόμενο στοιχείο της στοίβας από αυτό που βρίσκεται ήδη στη κορυφή και ύστερα αποθηκεύουμε το `t` στην `top` ως το στοιχείο κορυφής.

peek(): επιστρέφει τα δεδομένα του πρώτου στοιχείου της στοίβας, αν αυτή δεν είναι άδεια. Διαφορετικά, πετάει εξαίρεση τύπου `NoSuchElementException`.

printStack(PrintStream stream): εκτυπώνει το μήνυμα “List is empty” αν η λίστα είναι άδεια. Αλλιώς, δημιουργούμε μια νέα μεταβλητή τύπου `Node`, η οποία δείχνει στα περιεχόμενα της μεταβλητής `top`. Όσο το `n` είναι διάφορο του `null`, δηλαδή υπάρχει στοιχείο στη στοίβα, τότε εκτύπωσε τα δεδομένα του `n`. Ύστερα, στο `n` θέτουμε το επόμενο στοιχείο της στοίβας.

size(): επιστρέφει τη τιμή της μεταβλητής `stack_size`, η οποία αυξομειώνεται στις μεθόδους `push` και `pop` και δείχνει το μέγεθος της στοίβας.

StringQueueImp

Στην υλοποίηση της ουράς χρησιμοποιούμε δύο δείκτες, `head` και `top`, τους οποίους αρχικοποιούμε με `null`. Ακόμη θέτουμε τη μεταβλητή `size` ίση με 0, οποία λειτουργεί ως ένας μετρητής και μετράει το πλήθος των στοιχείων που υπάρχουν στην ουρά.

isEmpty(): ελέγχει αν η ουρά είναι άδεια βάσει του δείκτη `head` και επιστρέφει `true` αν είναι `head = null`, αν δηλαδή δεν υπάρχει κάποιο στοιχείο στην ουρά.

put(): δημιουργεί ένα νέο κόμβο `n` (αυξάνεται η μεταβλητή `size` κατά 1) και τον προσθέτει στο τέλος της ουράς. Αν η ουρά είναι άδεια και οι δυο δείκτες `tail` και `head` δείχνουν στον νέο κόμβο. Διαφορετικά, θέτουμε στο τωρινό τελευταίο κόμβο της ουράς ως επόμενο κόμβο το `n` και ύστερα θέτουμε ως τελευταίο κόμβο, δηλαδή στο `tail`, το `n`.

get(): επιστρέφει τα δεδομένα τύπου `T` του πρώτου κόμβου (εκεί που δείχνει το `head`) και τον αφαιρεί από την ουρά. Επίσης μειώνει τη μεταβλητή `size` κατά 1. Αν η ουρά είναι άδεια, πετάει εξαίρεση τύπου *NoSuchElementException*. Αν στην ουρά υπάρχει μόνο ένας κόμβος τότε οι δείκτες `head`, `tail` παίρνουν τη τιμή `null`. Αν ουρά έχει παραπάνω από ένα κόμβο τότε το `head` δείχνει στον επόμενο.

peek(): επιστρέφει τα δεδομένα του πρώτου κόμβου (`head`) χωρίς να διαγραφεί τον κόμβο. Αν η ουρά είναι κενή τότε πετάει εξαίρεση τύπου *NoSuchElementException*.

printQueue(Printstream stream): τυπώνει τα δεδομένα κάθε κόμβου της ουράς, αρχίζοντας από τον πρώτο. Αυτό το κάνουμε δημιουργώντας μια μεταβλητή `temp` τύπου `Node` ίση με τον δείκτη `head` και μέσω της εντολής `temp=temp.getNext()`; μετακινούμαστε από κόμβο σε κόμβο μέχρι το τέλος της ουράς. Όσο το `temp` δεν είναι `null`, δηλαδή όσο υπάρχει στοιχείο στην ουρά, τυπώνουμε τα δεδομένα του κόμβου `temp`. Αν η ουρά είναι κενή εμφανίζει αντίστοιχο μήνυμα.

size(): τυπώνει το πλήθος των κόμβων του πίνακα. Αυτό το έχουμε καταφέρει δημιουργώντας μια μεταβλητή `size=0` στη αρχή και αυξάνοντας την κατά ένα κάθε φορά που προσθέτουμε κάποιο κόμβο με τη μέθοδο `put()` και αφαιρώντας 1 κάθε φορά που διαγράφουμε κόμβο με τη μέθοδο `get()`.

Μέρος Β

Για την λύση του λαβυρίνθου, δημιουργήσαμε τη μέθοδο `solve_maze(String[][] maze, int coorX, int coorY)`. Η μέθοδος δέχεται σαν ορίσματα τον πίνακα που αποθηκεύσαμε το λαβύρινθο, και τις συντεταγμένες του σημείου που βρίσκεται το 'Ε', δηλαδή η είσοδος, και επιστρέφει `true` ή `false`, ανάλογα με το αν βρέθηκε ή όχι έξοδος από τον λαβύρινθο.

solve_maze

Δημιουργούμε το αντικείμενο `stack` της τάξης `StringStack`.

Στις μεταβλητές `x,y` αποθηκεύουμε τις συντεταγμένες της εισόδου και στις μεταβλητές `maxX, maxY`, τα όρια του πίνακα.

Ωθούμε ως πρώτο στοιχείο στη στοίβα μας τις συντεταγμένες της εισόδου. (Στη στοίβα αποθηκεύονται πίνακες `int[]`, με δεδομένα τις συντεταγμένες του σημείου που βρισκόμαστε) Ακόμα, χρειαζόμαστε να αποθηκεύσουμε κάπου (`int[] previous`) το προηγούμενο στοιχείο της στοίβας για να σιγουρευτούμε ότι δεν θα “πέσουμε” σε infinite loop (π.χ. ο αλγόριθμος να μετακινείται επ άπειρον δεξιά και αριστερά σε δύο διπλάνα στοιχεία). Εδώ βρίσκουμε χρήσιμη τη μέθοδο `peek()`.

Ξεκινάμε ένα infinite loop while, το οποίο θα τερματίσει όταν βρεθεί ότι υπάρχει ή δεν υπάρχει έξοδος από το λαβύρινθο.

Αν ισχύει κάποια από τις παρακάτω συνθήκες

1) `y == maxY` 2) `y == 0` 3) `x == maxX` 4) `x == 0` και `repeats > 1`

τότε αυτό σημαίνει ότι βρισκόμαστε σε ένα σημείο στα όρια του πίνακα και αυτό δεν είναι το ‘Ε’. Άρα βρισκόμαστε σε μία έξοδο, εκτυπώνουμε τις συντεταγμένες της εξόδου και επιστρέφουμε `true`.

Αλλιώς ξεκινάει η διαδικασία εύρεσης της εξόδου.

το `repeats` είναι ένας μετρητής που μετράει πόσες επαναλήψεις έχουν γίνει. Έτσι, αν έχει γίνει τουλάχιστον μια επανάληψη έχουμε εξασφαλίσει ότι έχουμε βρει έξοδο.

Ελέγχουμε ξεχωριστά κάθε ένα από τα όρια του λαβυρίνθου καθώς οι κινήσεις που μπορούμε να κάνουμε σε καθένα είναι περιορισμένες (δεν θέλουμε να ξεφύγουμε από τα όρια του πίνακα).

Σε κάθε προσπάθεια κίνησης ελέγχουμε αν το στοιχείο του πίνακα με συντεταγμένες το σημείο που θέλουμε να κινηθούμε ισούται με “0” και αν μια από τις δύο συντεταγμένες του σημείου που θέλουμε να κινηθούμε ισούται με την συντεταγμένη του προηγούμενου στοιχείου από αυτό που βρισκόμαστε τώρα (ελέγχουμε το μετά με τον πριν και όχι το τώρα)

Αν ισχύουν οι συνθήκες τότε μετακινούμαστε στο επόμενο στοιχείο και ετοιμαζόμαστε να το ωθήσουμε στη στοίβα. Πριν όμως, ελέγχουμε αν η στοίβα δεν είναι άδεια με τη μέθοδο `isEmpty()` και αποθηκεύουμε στο `previous` τις συντεταγμένες του σημείου που βρισκόμασταν μέχρι τώρα (τη κορυφή της στοίβας). Ο έλεγχος γίνεται για να μην οδηγηθούμε στη εξαίρεση `NoSuchElementException`. Τώρα είναι που ωθούμε το νέο σημείο στη στοίβα.

Ανάλογα με το πως κινούμαστε αυξομειώνουμε τις μεταβλητές `x,y` κατά 1.

Αφού ελέγξουμε τα όρια και πλέον είναι όλες οι κινήσεις επιτρεπτές επαναλαμβάνουμε τα ίδια βήματα με την προσθήκη μια επιπλέον `else if`. Αν δεν μπορούμε να κινηθούμε προς κάποια κατεύθυνση και η στοίβα μας δεν είναι άδεια τότε αλλάζουμε το περιεχόμενο του πίνακα στο σημείο που βρισκόμαστε με “1” (για να μην ξαναπεράσουμε απο εκεί). Ύστερα, αποθηκεύουμε τις συντεταγμένες του προηγούμενου σημείου ως νέες συντεταγμένες στα `x, y` και απωθούμε απο τη στοίβα το σημείο που βρισκόμασταν, με τη μέθοδο `pop()`, καθώς μας οδήγησε σε αδιέξοδο. Χρησιμοποιούμε δηλαδή τη λογική του `backtracking` και γυρίζουμε προς τα πίσω.

Αν δεν ισχύει ούτε η τελευταία συνθήκη, δηλαδή η στοίβα να είναι άδεια, τότε επιστρέφουμε `false`, καθώς δεν υπάρχει έξοδος στο λαβύρινθο.

ΜΕΡΟΣ Γ

StringQueueWithOnePointer

Σε αυτή τη κλάση έχουμε ένα μόνο pointer το `tail`, αρχικοποιημένο με `null`, που δείχνει στον τελευταίο κόμβο της ουράς. Επειδή έχουμε ένα pointer κάναμε την ουρά κυκλική , δηλαδή ο `pointer tail.next` δείχνει στο πρώτο κόμβο της ουράς.

isEmpty(): ελέγχει αν η ουρά είναι άδεια κοιτάζοντας αν ο δείκτης tail είναι null.

put(T data): δημιουργεί ένα νέο κόμβο n με δεδομένο= data και τον προσθέτει στο τέλος της ουράς. Επίσης αυξάνει τη μεταβλητή size κατά 1. Αν η ουρά είναι άδεια, τότε αφού έχουμε υλοποιήσει λίστα κυκλική σύνδεσης για την ουρά ο δείκτης tail θα πρέπει να δείχνει στον εαυτό του σαν επόμενο στοιχείο. Έτσι, θέτουμε ως tail τον νέο κόμβο n και ως next το n. Αν η ουρά δεν είναι άδεια, τότε στο n θέτουμε ως next το επόμενο κόμβο του tail, δηλαδή τον πρώτο κόμβο της ουράς, και θέτουμε το n ως επόμενο στοιχείο του tail, που ήταν μέχρι στιγμής το τελευταίο στοιχείο της ουράς. Τέλος, θέτουμε ως νέο tail τον n. Έτσι, ο προτελευταίος κόμβος δείχνει στον n και ο n έχει γίνει ο τελευταίος κόμβος, ο οποίος δείχνει στον πρώτο κόμβο της ουράς.

get(): αφαιρούμε τον πρώτο κόμβο της ουράς και επιστρέφουμε τα data του. Αν η ουρά είναι άδεια πετάει εξαίρεση τύπου NoSuchElementException. Αν η ουρά είναι έχει ένα κόμβο τότε το tail είναι ίσο με το επόμενο στοιχείο του (tail.next). Άρα παίρνουμε το δεδομένο από αυτόν και μετά τον αφαιρούμε από την ουρά κάνοντας τον tail ίσο με null. Αν η ουρά έχει παραπάνω από έναν κόμβο, τότε συνδέουμε τον τελευταίο κόμβο με τον δεύτερο κόμβο της ουράς, θέτοντας ως επόμενο στοιχείο του tail το tail.getNext().getNext().

peak(): παίρνουμε το δεδομένο του πρώτου κόμβου της ουράς, δηλαδή του κόμβου που δείχνει το tail.next αφού έχουμε κυκλική ουρά.

printQueue(Printstream stream): τυπώνει τα δεδομένα κάθε κόμβου της ουράς από τη αρχή μέχρι το τέλος της. Αυτό το κάνουμε δημιουργώντας μια μεταβλητή temp τύπου Node ίση με tail.next, δηλαδή ίση με το head και μέσω της εντολής temp=temp.getNext() μετακινούμαστε από κόμβο σε κόμβο μέχρι το τέλος της ουράς. Αν η ουρά είναι κενή εμφανίζει αντίστοιχο μήνυμα.

size(): τυπώνει το πλήθος των κόμβων του πίνακα. Αυτό το έχουμε καταφέρει δημιουργώντας μια μεταβλητή size=0 στη αρχή και αυξάνοντας την κατά ένα κάθε φορά που προσθέτουμε κάποιο κόμβο με τη μέθοδο put() και αφαιρώντας 1 κάθε φορά που διαγράφουμε κόμβο με τη μέθοδο get().