ΝΙΚΟΣ ΜΗΤΣΑΚΗΣ - Α.Μ. : 3210122
ΠΑΝΑΓΙΩΤΗΣ ΜΟΣΧΟΣ – Α.Μ. : 3210127

*1η ΕΡΓΑΣΙΑ ΣΤΟ ΜΑΘΗΜΑ ΤΩΝ ΔΟΜΩΝ ΔΕΔΟΜΕΝΩΝ*

# ΜΕΡΟΣ Δ

## *Για το μέρος Α:*

Δημιουργήσαμε μία κλάση με όνομα SinglyLinkedList στην οποία περιγράφουμε την έννοια της λίστας μονής σύνδεσης. Αυτό το κάναμε, για να υλοποιήσουμε τις δοσμένες διασυνδέσεις StringStack.java και StringQueue.java .

Ειδικότερα μέσα σε αυτήν την κλάση δημιουργήσαμε και μία στατική κλάση με όνομα Node για να προσομοιώσουμε τους κόμβους οι οποίοι απαρτίζουν μία ΑΤΔ όπως είναι η συνδεδεμένη λίστα μονής σύνδεσης. Η συγκεκριμένη κλάση έχει τις εξής μεθόδους: την getElement() η οποία επιστρέφει το στοιχείο στο οποίο δείχνει ο τωρινός κόμβος, την getNext() η οποία επιστρέφει τον επόμενο κόμβο από τον τωρινό και την setNext(Node n) η οποία αλλάζει τον αμέσως επόμενο κόμβο από τον τωρινό, δηλαδή του αλλάζει την άμεση σύνδεση με το επόμενο στοιχείο της λίστας.

Επιπροσθέτως, η κλάση της λίστας μονής σύνδεσης έχει 3 μεταβλητές αρχικοποίησης. Συγκεκριμένα τις 2 τύπου Node με όνομα head και tail που δείχνουν στο στοιχείο που βρίσκεται στην αρχής της λίστας ( ή αλλιώς κόμβος κεφάλι/head node ) και στο τέλος της λίστας ( ή αλλιώς κόμβος ουρά/tail node) αντίστοιχα που αρχικοποιούνται σε null και μία ακόμη τύπου int ονόματι size που αρχικοποιείται στην τιμή 0, η οποία αντιπροσωπεύει το πλήθος των κόμβων της λίστας ή πιο απλά το μέγεθός της.

Αναφορικά με τις μεθόδους της κλάσης της SinglyLinkedList, έχουμε γράψει τις εξής:

- size() -> επιστρέφει την τιμή της μεταβλητής size, δηλαδή το μέγεθος της
- isEmpty() -> επιστρέφει το αποτέλεσμα της λογικής πράξης αν η μεταβλητής size είναι ίση με το μηδέν, δηλαδή αν η λίστα είναι άδεια
- getFirst() -> επιστρέφει το πρώτο στοιχείο της λίστας χωρίς να το αφαιρέσει από αυτήν
- getLast() -> επιστρέφει το τελευταίο στοιχείο της λίστας, χωρίς να το αφαιρέσει από αυτήν
- addFirst(e) -> προσθέτει το στοιχείο e στην αρχής της λίστας και το συνδέει κατάλληλα
- addLast() -> προσθέτει το στοιχείο e στο τέλος της λίστας κα το συνδέει κατάλληλα
- removeFirst() -> Αφαιρεί από την λίστα το πρώτο στοιχείο και το επιστρέφει

Επομένως για να υλοποιήσουμε τις διασυνδέσεις StringStack και StringQueue δημιουργήσαμε δύο νέες κλάσεις, μία για κάθε διασύνδεση, με όνομα StringStackImlp καιStringQueueImpl αντίστοιχα.

- Για την StringStackImpl υλοποιούμε την βασική δομή δεδομένων χρησιμοποιώντας λίστα μονής σύνδεσης, τη στοίβα, μία δομή δεδομένων που ακολουθεί την αρχή LIFO (Last in First Out). Οπότε ορίσαμε ότι ο head κόμβος θα δείχνει στο στοιχείο που βρίσκεται στην κορυφή της στοίβας, ενώ ο κόμβος tail θα δείχνει στο στοιχείο που βρίσκεται στον πάτο της στοίβας.

Αναφορικά με τις μεθόδους της κλάσης StringStackImpl, έχουμε γράψει τις εξής:

- size() -> επιστρέφει το μέγεθος της στοίβας
- isEmpty() -> επιστρέφει το αποτέλεσμα της λογικής πράξης αν η μεταβλητής size είναι ίση με το μηδέν, δηλαδή αν η στοίβα είναι άδεια
- push(e) -> τοποθετεί το στοιχείο e στην κορυφή της στοίβας, δηλαδή το νέο head δείχνει τώρα στο στοιχείο e
- pop() -> αφαιρεί και επιστρέφει το πρώτο στοιχείο της στοίβας
- peek() -> επιστρέφει το πρώτο στοιχείο της στοίβας, χωρίς όμως να το αφαιρέσει
- printStack(stream) -> εκτυπώνει τα στοιχεία της στοίβας από την κορυφή προς τον πάτο. (Η μέθοδος εκτελείται αναδρομικά, βλ. σχόλια μέσα στα αρχεία κώδικα)

- Για την StringQueueImpl υλοποιούμε την βασική δομή δεδομένων χρησιμοποιώντας λίστα μονής σύνδεσης, την ουρά, μία δομή δεδομένων που ακολουθεί την αρχή FIFO (First in First Out). Οπότε ορίσαμε ότι ο head κόμβος θα δείχνει στο στοιχείο που βρίσκεται στην πρώτη θέση της ουράς, ενώ ο κόμβος tail θα δείχνει στο στοιχείο που βρίσκεται στην τελευταία θέση της ουράς.

Αναφορικά με τις μεθόδους της κλάσης StringStackImpl, έχουμε γράψει τις εξής:

- size() -> επιστρέφει το μέγεθος της ουράς
- isEmpty() -> επιστρέφει το αποτέλεσμα της λογικής πράξης αν η μεταβλητής size είναι ίση με το μηδέν, δηλαδή αν η ουρά είναι άδεια
- put(e) -> προσθέτει το στοιχείο e στην τελευταία θέση της ουράς, οπότε το νέο tail δείχνει τώρα στο στοιχείο e
- get() -> αφαιρεί και επιστρέφει το στοιχείο που βρίσκεται στην πρώτη θέση της ουράς
- peek() -> επιστρέφει το στοιχείο που βρίσκεται στην πρώτη θέση της ουράς, χωρίς όμως να το αφαιρέσει από αυτήν

- printQueue(stream) -> εκτυπώνει τα στοιχεία της ουράς από την αρχή προς το τέλος. (Η μέθοδος εκτελείται αναδρομικά, βλ. σχόλια μέσα στα αρχεία κώδικα)

## *Για το μέρος Γ:*

Δημιουργήσαμε μία κλάση με όνομα CircularlyLinkedList στην οποία περιγράφουμε την έννοια της κυκλικής λίστας μονής σύνδεσης. Αυτό το κάναμε, για να υλοποιήσουμε την βασική δομή δεδομένων της ουράς, στην κλάση StringQueueWithOnePointer όπως και στην κλάση StringQueueImpl με τη διαφορά ότι δεν διατηρούμε μία αναφορά στο head κόμβο της ουράς με κάποια μεταβλητή, παραμόνο στο tail κόμβο της.

Οι μέθοδοι της κλάσης CircularlyLinkedList είναι οι ίδιες μέθοδοι που έχουμε γράψει στην κλάση SinglyLinkedList με μία προσθήκη:

- rotate() -> Ουσιαστικά, πηγαίνει το πρώτο στοιχείο στο τέλος της λίστας (αν δεν είναι άδεια η λίστα προφανώς). Δηλαδή ο παλιός head κόμβος γίνεται ο νέος tail κόμβος

- Ειδικότερα ο τρόπος με τον οποίο έχουμε πάντα πρόσβαση στο head κόμβο της ουράς είναι ο εξής: head = tail.next , το οποίο next είναι attribute της ιδιωτικής στατικής κλάσης που βρίσκεται μέσα στην κλάση της CircularlyLinkedList.

Αναφορικά με τις μεθόδους της κλάσης StringQueueWithOnePointer, έχουμε γράψει τις εξής:

- size() -> επιστρέφει το μέγεθος της ουράς
- isEmpty() -> επιστρέφει το αποτέλεσμα της λογικής πράξης αν η μεταβλητής size είναι ίση με το μηδέν, δηλαδή αν η ουρά είναι άδεια
- put(e) -> προσθέτει στο τέλος της ουράς το στοιχείο e και το συνδέει κατάλληλα με τα υπόλοιπα
- get() -> αφαιρεί και επιστρέφει το πρώτο στοιχείο της ουράς

- peek() -> επιστρέφει το στοιχείο που βρίσκεται στην πρώτη θέση της ουράς, χωρίς όμως να το αφαιρέσει από αυτήν

- printQueue(stream) -> εκτυπώνει τα στοιχεία της ουράς από την αρχή προς το τέλος. (Η μέθοδος εκτελείται αναδρομικά, βλ. σχόλια μέσα στα αρχεία κώδικα)

ΝΙΚΟΣ ΜΗΤΣΑΚΗΣ - Α.Μ. : 3210122
ΠΑΝΑΓΙΩΤΗΣ ΜΟΣΧΟΣ – Α.Μ. : 3210127

## *Για το μέρος Β:*

Δημιουργήσαμε μία κλάση με όνομα Thiseas.java στην οποία χρησιμοποιούμε την υλοποίηση της στοίβας που φτιάξαμε στο Μέρος Α (συγκεκριμένα της StringStackImpl) για να διαβάσουμε ένα αρχείου κειμένου (.txt) το οποίο θα παρομοιάζει έναν λαβύρινθο, από τον οποίο θα αναζητήσει το πρόγραμμά μας κάποιο μονοπάτι για να βγει από έξω. Το μονοπάτι αυτό θα το αποθηκεύει σε μία στοίβα StringStackImpl.

Ειδικότερα μέσα στην main κλάση της συγκεκριμένης κλάσης κάνουμε τα εξής:

Αρχικά διαβάζουμε το αρχείο .txt του οποίου το path θα το έχει δώσει ο χρήστης ως όρισμα της main όταν καλεί το πρόγραμμά μας από την γραμμή εντολών.

Έχουμε προνοήσει με try – catch block για την περίπτωση που δεν δώσει σωστά ο χρήστης το path του αρχείου ή γενικά το path που δώσει δεν αντιστοιχεί σε κάποιο υπάρχον αρχείο. Στην περίπτωση που συμβεί κάτι τέτοιο εκτυπώνουμε κατάλληλο μήνυμα για να ενημερώσουμε τον χρήστη για το θέμα που προκλήθηκα και τερματίζουμε το πρόγραμμα.

Στην περίπτωση τώρα που το path του αρχείου που έχει δώσει ο χρήστης είναι εντάξει, δημιουργούμε ένα αντικείμενο BufferedReader το οποίο το χρησιμοποιούμε για να δημιουργήσουμε ένα αντικείμενο FileReader για το εν λόγω αρχείο.

Καθώς η δομή του αρχείου κειμένου που θα έχει δώσει ο χρήστης θα μας είναι γνωστή, διαβάζουμε ως ένα πίνακα από String την πρώτη γραμμή και εξάγουμε από αυτή τα πλήθη των γραμμών και των στηλών του λαβυρίνθου και τα αποθηκεύουμε σε 2 int μεταβλητές, row και columns αντίστοιχα. Εργαζόμαστε με τον ίδιο ακριβώς τρόπο και για την επόμενη γραμμή από την οποία εξάγουμε τις συντεταγμένες της εισόδου, σε μορφή (γραμμή, στήλη) και αποθηκεύουμε κάθε συντεταγμένη σε μία μεταβλητή int, με όνομα in_1 και in_2 αντίστοιχα.

Στη συνέχεια δημιουργούμε μία ArrayList με όνομα boardOfStrings που αποθηκεέι αντικείμενα τύπου String και ένα αντικείμενο στοίβας, με όνομα stack της κλάσης StringStackImpl που υλοποιήσαμε στο Μέρος Α. Συνεχίζουμε διαβάζοντας το υπόλοιπο αρχείο, γραμμή – γραμμή και αποθηκεύουμε κάθε γραμμή του ως ένα String χωρίς κενά μεταξύ των στοιχείων στο boardOfStrings. Έπειτα, για δικιά μας διευκόλυνση μετατρέπουμε το boardOfStrings σε κανονικό μονοδιάστατο πίνακα από String με όνομα array, τον οποίο τελικά τον μετατρέπουμε σε έναν δισδιάστατο πίνακα από char με όνομα maze, με ίδιες διαστάσεις όπως αυτές που έχει ο λαβύρινθος στο δοσμένο αρχείο. Οπότε ο πίνακας maze αντιπροσωπεύει τον λαβύρινθο για τον οποίο καλείται το πρόγραμμά μας να βρει κάποιο μονοπάτι προς κάποια έξοδο.

Στο σημείο αυτό, εκτελούμε όλους τους απαραίτητους ελέγχους που μας ζητάει η εκφώνηση του συγκεκριμένου μέρους της εργασίας. Αρχικά ελέγχουμε αν οι διαστάσεις του πίνακα που αναγράφονται στην πώτη γραμμή του αρχείου συμφωνούν με το μέγεθος του πίνακα (λαβυρίνθου) που διαβάσαμε από αυτό. Μετά ελέγχουμε αν οι συντεταγμένες της εισόδου που αναγράφονται στην δεύτερη

γραμμή του αρχείου συμφωνούν με τη θέση του συμβόλου Ε (το οποίο δείχνει την είσοδο) στον πίνακα (λαβύρινθο).

Στη συνέχεια για να μην αρχίσουμε άδικα το ψάξιμο για κάποιο μονοπάτι που θα οδηγεί σε κάποια έξοδο, ελέγχουμε πρώτα αν υπάρχει τουλάχιστον μία έξοδος, ενώ αν υπάρχουν πάνω από μία τις μετράμε και εκτυπώνουμε ένα μήνυμα στον χρήστη για να τον ενημερώσουμε από πόσες πιθανές εξόδους μπορεί να βγει από τον λαβύρινθο (Η έξοδος συμβολίζεται με κάποιο 0 στις πλευρές του πίνακα, δηλαδή κάποιο 0 στην πρώτη ή τελευταία γραμμή και πρώτη ή τελευταία στήλη του λαβυρίνθου).

Αν δεν βρεθεί καμία έξοδος από τον λαβύρινθο, εκτυπώνουμε κατάλληλο μήνυμα στον χρήστη και τερματίζουμε το πρόγραμμα.

Εφόσον έχει βρεθεί τουλάχιστον μία έξοδος από τον λαβύρινθο, προχωράμε στην αναζήτηση κάποιου μονοπατιού που θα οδηγεί στην έξοδο (ή σε μία από τις πολλές, αν υπάρχουν).

Πρώτιστα, «σπρώχνουμε» τις συντεταγμένες της εισόδου στην στοίβα μας, καθώς (προφανώς !) αποτελούν την αρχή οποιουδήποτε μονοπατιού τελικά ακολουθήσουμε.

Επίσης ενσωματώσαμε στο πρόγραμμά μας την λειτουργία του backtracking (οπισθοδρόμηση), το οποίο ουσιαστικά σημαίνει ότι αν βρεθεί σε κάποια αδιέξοδο, να έχει την ικανότητα να γυρίσω πίσω από το μονοπάτι που ήρθε. (Αξιοποιούμε την στοίβα που υλοποιήσαμε στο Μέρος Α)

Πρώτιστα, φτιάχνουμε ένα βρόγχο επανάληψης while στο οποίο μέσα θα κάνουμε τις απαραίτητες ενέργειες για να βρούμε ένα μονοπάτι που θα οδηγεί σε κάποια έξοδο από τον λαβύρινθο.

Αρχικά έχουμε γράψει μία εντολή if μέσω της οποίας ελέγχουμε αν βρισκόμαστε σε κάποιο ακριανό σημείο (δηλ. στην πρώτη ή τελευταία, είτε γραμμή, είτε στήλη) του λαβυρίνθου και η τιμή του σημείου αυτή είναι 0, το οποίο μας υποδεικνύει ότι βρισκόμαστε σε έξοδο, οπότε έχουμε βρει και μονοπάτι, άρα εκτυπώνουμε τις συντεταγμένες του σημείου, στο οποίο βρίσκεται η έξοδος που βρήκαμε και βγαίνουμε από το while, οπότε τερματίζουμε και το πρόγραμμα.

Στη συνέχεια έχουμε γράψει 4 άλλες εντολές if κάθε μία για τον έλεγχο ύπαρξης κάποιου 0 σε κάθε μία από τις τέσσερις κατευθύνσεις από το σημείο στο οποίο βρισκόμαστε κάθε φορά, ελέγχοντας παράλληλα ότι η επόμενη κίνησή μας δεν μας βγάζει έξω από τα όρια του πίνακα.

Εφόσον υπάρχει διαθέσιμο σημείο που μπορούμε να περπατήσουμε και δεν βρισκόμαστε σε κάποια έξοδο (δηλαδή 0 σε κάποια από τις τέσσερις κατευθύνσεις), «σπρώχνουμε» στην στοίβα τις συντεταγμένες του σημείου στο οποίο συνεχίζουμε και «σημαδεύουμε» το προηγούμενο σημείο στο οποίο ήμασταν (εκτός από το πρώτο βήμα, που το προηγούμενο σημείο ήταν η έξοδος και δεν θέλουμε να χάσουμε το

σημάδι της (Ε) ) μετατρέποντάς το από 0 σε Τ, το οποίο σημαίνει πως το έχουμε επισκεφτεί, αλλά ακόμη δεν ξέρουμε αν ανήκει σε κάποιο μονοπάτι το οποίο θα μας οδηγήσει σε αδιέξοδο. Δηλαδή το έχουμε «πατήσει» αλλά ακόμη βρισκόμαστε σε μονοπάτι το οποίο δεν μας έχει οδηγήσει σε κάποια αδιέξοδο.

Τέλος ενημερώνουμε τις συντεταγμένες της θέσης που βρισκόμαστε σε κάθε χρονική στιγμή (ως μεταβλητές για της συντεταγμένες της παρούσας κάθε φορά θέσης μας, χρησιμοποιούμε τις ίδιες που είχαμε ορίσει στην αρχή για τις συντεταγμένες της εισόδου, in_1 και in_2). Μετά από αυτές τις εντολές ακολουθεί μία εντολή else.

Μέσα στην εντολή αυτή, με μία ακόμη εντολή if ελέγχουμε αν η τωρινή μας θέση είναι κάποιο μη ακριανό 0, οπότε έχουμε οδηγηθεί σε αδιέξοδο. Αφού, έχουμε οδηγηθεί σε αδιέξοδο εφαρμόζουμε την διαδικασία του backtracking χρησιμοποιώντας την στοίβα του Μέρους Α. Αρχικά «σημαδεύουμε» την τωρινή μας θέση, μετατρέποντάς την από 0 σε Χ, το οποίο σημαίνει ότι το συγκεκριμένο σημείο το έχουμε επισκεφτεί και επιπλέον ανήκει σε κάποιο μονοπάτι που μας οδήγησε σε αδιέξοδο. Συνεχίζοντας, αφαιρούμε από την στοίβα το πρώτο (στοιχείο στη κορυφή), το οποίο είναι το ίδιο το στοιχείο το οποίο μόλις σημαδέψαμε με Χ. Επιπλέον, αντλούμε με την κλήση της μεθόδου peek στην στοίβα μας, το καινούργιο πρώτο στοιχείο της, το οποίο όπως έχουμε είναι ένα String το οποίο αποτελείται από δύο χαρακτήρες. Ο πρώτο δείχνει τον αριθμό της γραμμής και ο άλλος τον αριθμό της στήλης του κελιού που βρισκόταν το 0 που έχουμε πατήσει. Οπότε χρησιμοποιώντας τους χαρακτήρες ενημερώνουμε κατάλληλα τις τιμές των μεταβλητών μας in_1 και in_2. Επιπλέον, επαναφέρουμε το περιεχόμενο του κελιού που «δείχνουν» οι νέες τιμές των μεταβλητών αυτών σε 0 (δεν το κάνουμε αυτό μόνο στη περίπτωση που το συγκεκριμένο κελί είναι το κελί της εισόδου), για να μπορούμε να ξαναελέγξουμε πιθανά περπατήσιμα κελιά στις τέσσερις γνωστές κατευθύνσεις από το συγκεκριμένο κελί.

Έπειτα αρχίζουμε νέα εκτέλεση του βρόγχου επανάληψης while.

Τέλος με μία εντολή else (στην περίπτωση που έχει φτάσει σε αδιέξοδο και δεν μπορεί να κουνηθεί σε κανένα άλλο κελί), ενημερώνουμε τον χρήστη ότι δεν υπάρχει μονοπάτι που να οδηγεί σε κάποια από τις εξόδους του λαβυρίνθου, εκτυπώνοντας ένα κατάλληλο μήνυμα και τερματίζοντας το πρόγραμμα.

---

## *Για να τρέξετε το Thiseas.java:*

- Compile Program: javac -encoding "UTF-8" Thiseas.java

- Run Program: java Thiseas path of file (with .txt)

ΝΙΚΟΣ ΜΗΤΣΑΚΗΣ - Α.Μ. : 3210122
ΠΑΝΑΓΙΩΤΗΣ ΜΟΣΧΟΣ – Α.Μ. : 3210127