ΔΟΜΕΣ ΔΕΔΟΜΕΝΩΝ ΝΟΕΜΒΡΙΟΣ 2019 ΕΡΓΑΣΙΑ 1

ΚΩΤΣΗΣ-ΠΑΝΑΚΑΚΗΣ ΒΑΣΙΛΕΙΟΣ-ΕΚΤΩΡ (Ρ3180094)

ΓΕΩΡΓΙΟΥ ΑΛΕΞΙΟΣ - ΛΑΖΑΡΟΣ (Ρ3180027)

Μέρος Α (Υλοποίηση Στοίβας και Ουράς)

Για την υλοποίηση χρειαστήκαμε, μια κλάση Node που περιέχει τις βασικές μεθόδους και την υλοποίηση ενός κόμβου. Επίσης μια κλάση List, η οποία χρησιμοποιεί τις μεθόδους του κόμβου και υλοποίει μια βασική δομή δεδομένων λίστας μονής σύνδεσης με τις μεθόδους της. Είναι σημαντικό να αναφέρουμε πως η List περιέχει 2 δύο βασικούς κόμβους αναφοράς, ονομαζόμενοι head και tail. Με την κατάλληλη χρήση αυτών των 2 κόμβων μπορούμε να έχουμε πάντοτε αναφορά στην αρχή (head) και στο τέλος (tail) της λίστας, πράγμα που είναι αναγκαίο για την ορθή υλοποίησης των δομών.

Στην συνέχεια υλοποιείται η Ουρά (StringQueueImpl) και η Στοίβα (StringStackImpl) χρησιμοποιώντας τις μεθόδους της απλής λίστας. Ο ρόλος της κάθε μεθόδου μπορεί να βρεθεί στην μορφή σχόλιου στα δοσμένα .java αρχεία που περιέχουν τα interfaces για την κάθε κλάση. Όλες οι μέθοδοι των κλάσεων εκτός από τις μεθόδους εκτύπωσης γίνονται σε χρόνο O(1). Αυτό στην μέθοδο size() γίνεται με την βοήθεια ενός μετρητή που αυξομειώνεται κάθε φορά που κάνουμε insert ή remove ένα αντικείμενο, αποφεύγοντας έτσι την προσπέλαση της λίστας για την εύρεση του μεγέθους της.

Για την στοίβα:

Για την υλοποίηση της στοίβας φτιάχνουμε αρχικά ένα αντικείμενο τύπου List στο οποίο θα αποθηκεύουμε τα δεδομένα και περιέχει τις μεθόδους που θα αξιοποιηθούν. Στην στοίβα που υλοποιούμε, θα υποθέσουμε πως στην κορυφή της “δείχνει” το head της λίστας, ενώ στον πάτο της δείχνει το tail. Η πρόσθεση και αφαίρεση στοιχείων, που είναι LIFO, γίνεται από την κορυφή της λίστας, με τις μεθόδους push() Και pop() αντίστοιχα. Η push() χρησιμοποιεί την μέθοδο insertAtFront() της λίστας, η οποία θέτει τον επόμενο κόμβο του νέου κόμβου n ίσο με το head και ύστερα μεταφέρει τον δείκτη head στον νέο κόμβο n. Αντίστοιχα, η pop() χρησιμοποιεί την μέθοδο removeFromFront() της λίστας, η οποία κάνει τον head να δείχνει στον ακριβώς επόμενο κόμβο του, ώστε να μην υπάρχει δείκτης στον παλιό κόμβο που έδειχνε το head, και έτσι η Java να τον διαγράψει από την μνήμη με την χρήση του Garbage Collector. Μεταξύ άλλων, η pop() επιστρέφει και τα περιεχόμενα του διαγραμμένου κόμβου.

Για την ουρά:

Όπως και στην στοίβα, η ουρά υλοποιείτα με την δημιουργία του αντικειμένου τύπου List. Η διαφορά με την στοίβα είναι πως εφόσον η ουρά προσθαφαιρεί δεδομένα με τρόπο FIFO, θα πρέπει τα δεδομένα να προσθέτονται από την αρχή της ουράς, και να αφαιρούνται από το τέλος. Στην προκειμένη περίπτωση, υποθέτουμε πως το head δείχνει στο τέλος της ουράς (πρώτα δεδομένα σε προτεραιότητα), ενώ η tail στην αρχή(τελευταία δεδομένα σε προτεραιότητα). Η μέθοδος put(), με την χρήση της μεθόδου insertAtBack() της λιστας τοποθετεί τον νέο κόμβο n στο τέλος της λίστας, μετατρέποντας τον επόμενο κόμβο της tail στον n, και κάνοντας τον tail τελικά να

δείχνει στον n. Η get(), που αφαιρεί το δεδομένο στο τέλος της ουράς και επιστρέφει το περιεχόμενό του, χρησιμοποιεί την removeFromFront(), η οποία έχει ήδη αναλυθεί παραπάνω στην στοίβα.

Μέρος Β (Θησέας)

Υλοποίηση:

Πριν αρχίσει ο αλγόριθμος, κάνουμε κάποιες αλλαγές στα μηδενικά με ιδιότητες όπως αδιέξοδο, έξοδος, διασταύρωση με βάση των αριθμών γειτόνων μηδενικών στοιχείων (γείτονες 4 κατευθύνσεων).

Η Αδιέξοδος συμβολίζεται με “x” και είναι ένα μηδενικό το οποίο έχει μόνο ένα μηδενικό ως γείτονα και δεν είναι έξοδος, δηλαδή δεν βρίσκετε στην άκρη του λαβυρίνθου (2d- πίνακα).

Η Έξοδος συμβολίζεται με “e” και μπορεί να είναι οποιοδήποτε μηδενικό που βρίσκεται στην άκρη του λαβυρίνθου.

Η Διασταύρωση συμβολίζεται με “j” και είναι ένα μηδενικό που έχει 3 ή 4 γείτονες μηδενικά στοιχεία.

Επίσης γίνεται ένας έλεγχος της εισόδου από το txt αρχείο για το αν οι πληροφορίες έχουν αναπαρασταθεί σωστά στο λαβύρινθό. Συγκεκριμένα γίνεται έλεγχος για το αν η Είσοδος είναι στις σωστές δοσμένες συντεταγμένες ή αν δεν υπάρχει Είσοδος ή υπάρχουν περισσότερες από μια είσοδοι, για το αν οι διαστάσεις του πίνακα είναι σωστές και αν στο λαβύρινθο υπάρχουν ξένα στοιχεία που δεν είναι δηλαδή 0, 1 ή E.

Στην συνέχεια ο αλγόριθμος ελέγχει την θέση της εισόδου και με βάση την πλευρά που βρίσκεται κάνει το πρώτο βήμα (ελέγχοντας άμα δεν είναι 1). Από εκεί και πέρα όταν συναντάμε αδιέξοδο, γυρίζουμε πίσω (βγάζοντας τις πληροφορίες από τις στοίβες) στο προηγούμενο j (αν δεν έχει βρεθεί j το πρόγραμμα τερματίζει) αλλάζοντας στην επιστροφή πάλι το μονοπάτι σε 1. Αν βρεθεί έξοδος πριν συναντήσουμε κάποια διασταύρωση τότε το πρόγραμμα τερματίζει δίνοντας τις συντεταγμένες εξόδου. Τα βήματα γίνονται με βάση το προηγούμενο βήμα, ο αλγόριθμος “θυμάται” την προηγούμενη κίνηση και έτσι δεν γυρίζει πίσω από εκεί που ήρθε παρά μόνο άμα δεν υπάρχει άλλη επιλογή (προτεραιότητα κινήσεων). Όταν ο αλγόριθμος αναγκάζεται να γυρίσει στην προηγούμενη θέση του, τότε θέτει την τωρινή θέση του (πάντοτε διασταύρωση) σε αδιέξοδο “x”. Αυτό επαναλαμβάνεται μέχρι να μην υπάρχει λύση (εγκλωβισμός) ή βρεθεί η έξοδος.

Λειτουργία backtracking και χρήση στοίβας:

Ο αλγόριθμος χρησιμοποιεί 2 στοίβες xStack, yStack που η δομή τους έχει υλοποιηθεί στο μέρος Α για να αποθηκεύει τα βήματα του, όταν φτάνουμε σε μια αδιέξοδο γίνεται backtracking στην προηγουμένη διασταύρωση και απλά αφαιρούνται (pop) τα πάνω αντικείμενα τις στοίβας μέχρι να βρεθεί μια διασταύρωση. Οι στοίβες είναι παράλληλες για να διατηρούν τις συντεταγμένες κάθε βήματος (x, y).

Προϋποθέσεις:

Η Είσοδος (E) δεν μπορεί να βρίσκεται σε γωνία του λαβυρίνθου.

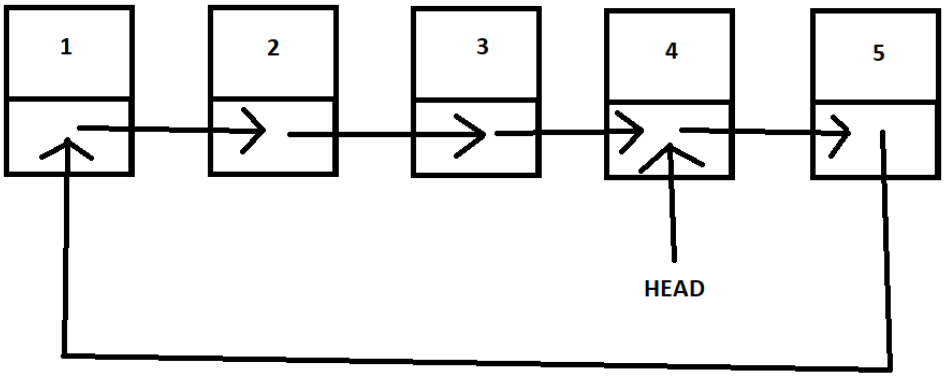
Το σύμβολο της Εισόδου είναι σε αγγλικούς χαρακτήρες (E).

Μια έξοδος δεν μπορεί να βρίσκετε διπλά στην είσοδο.

Θεωρούμε ότι ο λαβύρινθος δεν περιέχει κάποιο κυκλικό μονοπάτι μηδενικών ή συστάδες μηδενικών (σε τετράγωνα).

Μέρος Γ (Υλοποίηση ουράς με κυκλική λίστα μονής σύνδεσης)

Η υλοποίηση της ουράς σε αυτήν την περίπτωση δεν διαφέρει πολύ από το Μερος Α. Αντί των μεθόδων insertAtBack() και removeFromFront() της List που χρησιμοποιούνταν στο Α, εδώ θα γίνουν χρήση μόνο των μεθόδων insert() και remove() της CircularList (κυκλική λίστα). Η διαφορά της CircularList με την List είναι πως αντί για 2 δείκτες (head, tail) που χρησιμοποιούσε η List, έχει μόνον 1, έστω head, με τον οποίο μπορεί να υλοποιήσει μεθόδους εισαγωγής και εξαγωγής. Ο τρόπος με τον οποίο το κάνει αυτό οφείλεται στην δομή της κυκλικής λίστας, της οποίας ένα απλό παράδειγμα μπορεί να φανεί στο παρακάτω σχήμα (**Σχημα 1)**.



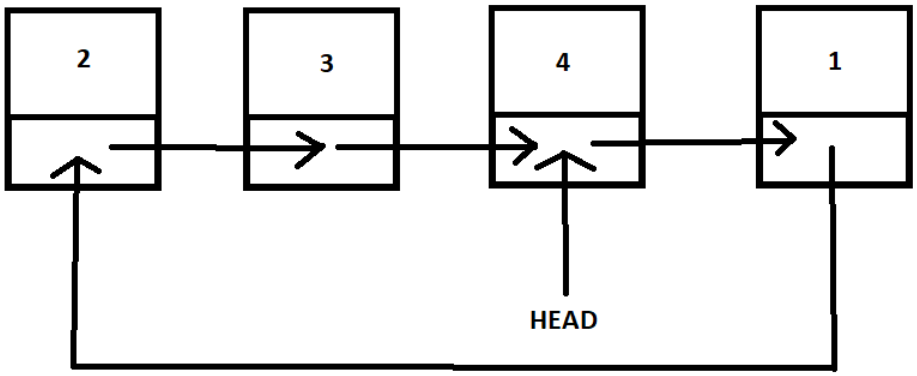
Σχήμα 1

Βλέπουμε δηλαδή πως έχουμε τον head να δείχνει στον κόμβο 4, ενώ ο επόμενος κόμβος του 5 να δείχνει στον κόμβο 1. Θα θεωρήσουμε πως ο κόμβος στον οποίο δείχνει ο head θα είναι πάντα το πιο πρόσφατο στοιχείο που έχουμε εισαγάγει στην ουρά (δηλαδή τελευταίας προτεραιότητας) και πως ο επόμενος (head.next) θα είναι ο πιο παλιός (δηλαδή πρώτης πρωτεραιότητας). Με αυτόν τον τρόπο μπορούμε πάντοτε με έναν κόμβο να έχουμε αναφορά στην “αρχή” και στο “τέλος” της κυκλικής λίστας.

Η put() χρησιμοποιεί την μέθοδο insert() της CircularList. Η insert() προσθέτει έναν κόμβο στην “αρχή” της λίστας. Δηλαδή, αν κοιτάξουμε το σχήμα, θα τοποθετηθεί δεξια από το 4 και αριστερά από το 5. Πιο συγκεκριμένα, η μέθοδος insert() θέτει τον επόμενο κόμβο του νέου στοιχείου n να είναι το επόμενο στοιχείο του head, ύστερα θέτει το επόμενο στοιχείο του κόμβου που δείχνει ο head να είναι το n, και τέλος βάζει το head να δείχνει στο n. Με αυτόν τον τρόπο, το n (που είναι και τώρα το νέο head) πλέον βρίσκεται στην αρχή της ουράς, και το επόμενο του στοιχείο είναι το στοιχείο στο τέλος της ουράς. Άρα επιτεύχθηκε η εισαγωγή του στοιχείου στην αρχή της ουράς.

Η get() κάνει χρήση της μεθόδου remove() της CircularList. Η remove() αφαιρεί τον κόμβο στο “τέλος” της λίστας, δηλαδή τον επόμενο κόμβο του head (Στο Σχήμα 1 είναι ο 5). Εμβαθύνοντας, ο τρόπος με τον οποίο δουλεύει η remove() είναι σχετικά απλός. Αρχικά σώζεται σε μια μεταβλητή επιστροφής το περιεχόμενο του κόμβου που πρόκειται να διαγραφθεί (έστω n), ώστε να μπορεί να επιστραφεί στο τέλος. Στη συνέχεια, ο επόμενος κόμβος του head (το πιο παλιο στοιχείο στην ουρά) θέτεται ίσος με τον δεύτερο επόμενό του (το δεύτερο πιο παλιό στοιχείο στην ουρά). Εφόσον δεν

υπάρχει κάποιος δείκτης στο στον επόμενο κόμβο του head, ο garbage collector της Java θα τον διαγράψει. Τέλος, επιστρέφεται ο σωσμένος n. Για να γίνει λίγο πιο κατανοητό, το τελικό αποτέλεσμα του παραπάνω συλλογισμού εφαρμόστηκε στο μοντέλο του Σχήματος 1 και μπορεί να φανεί στο **Σχήμα 2** παρακάτω.



Σχήμα 2

Φαίνεται λοιπόν πως με τον κόμβο 5 να έχει αφαιρεθεί, ο επόμενος κόμβος του head γίνεται ο κόμβος 1. Έτσι, το στοιχείο στον κόμβο 1 είναι τώρα το στοιχείο στην ουρά με την μεγαλύτερη προτεραιότητα (δηλαδή στο τέλος της), χωρίς να χαλάσει η δομή της κυκλικής λίστας και άρα της ουράς.

Με αυτήν την λίστα δηλαδή, μπορούμε να προσθαφαιρούμε δεδομένα σε μια ουρά με μόνο έναν δείκτη (head) σε χρόνο O(1), εφόσον έχουμε πάντοτε αναφορά στην αρχή και στο τέλος της λίστας με τον τρόπο που την έχουμε δομήσει.