ΝΙΚΟΛΕΤΑ ΠΑΠΑΓΕΩΡΓΙΟΥ – 3170234

ΘΑΝΑΣ ΚΟΥΡΟ – 3170078

Μέρος Α

Στην κλάση StringDoubleEndedQueueImpl δημιουργούμε μία κλάση Node η οποία δέχεται στον κατασκευαστή της ένα στοιχείο (item) τύπου <T>.

Σε κάθε κλήση της StringDoubleEndedQueueImpl δημιουργουνται τα head, tail που δείχνουν στην αρχή και στο τέλος αντίστοιχα, της λίστας.

Οι μέθοδοι addFirst(), removeFirst(), addLast(), removeLast(), getFirst(), getLast() ολοκληρώνονται σε χρόνο O(1).

Αυτό συμβαίνει, γιατί σε κάθε κλήση οποιασδήποτε μεθόδου ο χρόνος δεν εξαρτάται από το πλήθος των στοιχείων της λίστας. Σε κάθε κλήση μιας μεθόδου οι εντολές που θα εκτελεστούν είναι συγκεκριμένες σε πλήθος, ανεξάρτητες από το μέγεθος της λίστας και χωρίς χρήση κάποιας επανάληψης (while loop, for loop).

Όσο αφορά την μέθοδο size() είναι επίσης σε O(1). Αυτό συμβαίνει χάρη στη μεταβλητή totalItems , η οποία επεξεργάζεται και ενημερώνεται σε κάθε κλήση των μεθόδων που αναφέραμε παραπάνω. Συγκεκριμένα, εάν καλέσουμε την addFirst() αυξάνουμε κατά 1 την totalItems και αυτό σαφώς και δεν επηρεάζει την πολυπλοκότητα της addFirst(). Ομοίως και για την addLast(). Όταν διαγράφουμε κάποιο κόμβο-στοιχείο της λίστας με τη μέθοδο removeFirst() ή removeLast() μειώνουμε κατά 1 την τιμή της totalItems (εφόσον περιέχει τουλάχιστον 1 στοιχείο η λίστα). Έτσι, δεν επηρεάζεται η πολυπλοκότητα των removeFirst(), removeLast() και έχουμε αποθηκεύσει το μέγεθος της λίστας στην μεταβλητή totalItems. Όταν, λοιπόν, καλέσουμε την size() , το μόνο που θα κάνει είναι να επιστρέφει την τιμή της totalItems κάτι που γίνεται σε χρόνο O(1) με μία μόνο εντολή, χωρίς να χρησιμοποιήσουμε κάποια δομή επανάληψης (καθώς θα είχαμε χρόνο O(v)).

Σημείωση: Όλα τα αρχεία του src μεταγλωττίζονται από την γραμμή εντολών.

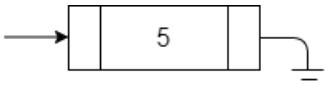
Μέρος Β

Αρχικά ζητάμε από τον χρήστη να εισάγει μια παράσταση και την αποθηκεύουμε σε ένα String με όνομα post.

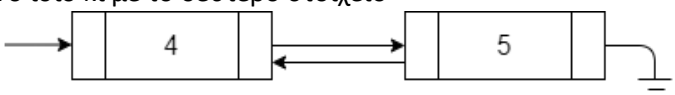
Αρχικοποιώντας τέσσερις μεταβλητές, char c = '0' , boolean isPostfix = true , int operands = 0 και int operators = 0, ξεκινάμε ένα for loop για τον έλεγχο της μεταθεματική παράστασης post .Ως «c» ορίζουμε σε κάθε επανάληψη το post.charAt(i) ,όπου i = 0, 1,2,..,n-1 με n = πλήθος χαρακτήρων στην παράσταση. Ελέγχουμε για κάθε χαρακτήρα c εάν είναι 0-9 ή κάποιο τελεστής (+,-,*,/). Εάν κάποιος χαρακτήρας δεν είναι έγκυρος κάνουμε την isPostfix = false, κάνουμε break και τυπώνουμε μήνυμα λάθους. Επιπλέον, σε κάθε επανάληψη, αυξάνουμε κατά 1 την operands εάν το c είναι αριθμός ή αυξάνουμε κατά 1 την operators εάν είναι τελεστής.
Μόλις βγούμε από την επανάληψη, εάν το isPostfix είναι true, σημαίνει ότι η παράσταση είναι έγκυρη. Όμως πρέπει να ελέγξουμε άλλες 2 περιπτώσεις.
1) Εφόσον isPostfix = true, ελέγχουμε τον τελευταίο χαρακτήρα της παράστασης. Εάν είναι τελεστής είμαστε εντάξει, διαφορετικά κάνουμε isPostfix = false και τυπώνουμε μήνυμα λάθους.
2) Στον δεύτερο έλεγχο πρέπει να ισχύει : operators = operands -1. Αν δεν ισχύει τυπώνουμε μήνυμα λάθους, διαφορετικά είμαστε (σχεδόν) έτοιμοι να μετατρέψουμε την παράσταση σε ενθεματική. Η συνθήκη πρέπει να ισχύει γιατί αν έχουμε π.χ ''568/'' θα επιστρέψει (6/8) αλλά το 5 θα μείνει μόνο του. Οπότε μετά από κάποια παραδείγματα συμπεραίνουμε ότι οι τελεστές πρέπει να είναι κατά 1 λιγότεροι από τους αριθμούς.
Εξαντλώντας όλες τις περιπτώσεις(εκτός από μία**) να είναι λάθος η μεταθεματική, πάμε να την μετατρέψουμε σε ενθεματική. (**την τελευταία περίπτωση θα την δούμε στην μετατροπή.)

Μετατροπή postfix σε infix:
Αρχικά δημιουργούμε ένα αντικείμενο s τύπου StringDoubleEndedQueueImp<String> , που είναι η ουρά που θα χρησιμοποιήσουμε για την μετατροπή.
Σε μια for loop διασχίζουμε την post (μεταθεματική παράσταση) και ελέγχουμε κάθε χαρακτήρα με τη σειρά. Θα χρησιμοποιήσουμε ένα παράδειγμα παράλληλα με την επεξήγηση του κώδικα. Έστω post = '' 54+96-- ''
Όσο οι χαρακτήρες είναι αριθμοί, τους τοποθετούμε στην ουρά με την μέθοδο addFirst(). Εάν συναντήσουμε τελεστή σημαίνει πως πρέπει να γίνει μία αλλαγή σε πράξη.

Διαβάζοντας το πρώτο στοιχείο ,αναγνωρίζει ότι είναι αριθμός και το προσθέτει στην αρχή της λίστας

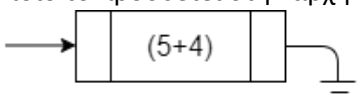


- Το ίδιο κι με το δεύτερο στοιχείο

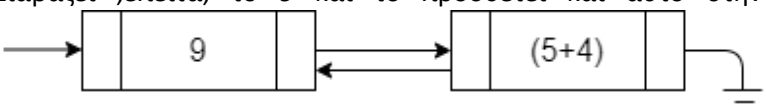


- Στην συνέχεια διαβάζοντας τον τελεστή + , πρέπει να γίνει μία πράξη και αφαιρεί τα δύο πρώτα στοιχεία της λίστας και ορίζει ως last = s.removeFirst() (δηλαδή 4 στο παράδειγμα) και first = s.removeFirst() (5 στο παράδειγμα). 'Ετσι, διαγράφουμε και

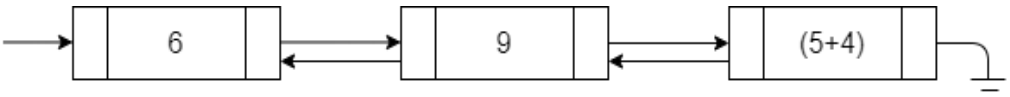
τα στοιχεία της ουράς. Σε μία μεταβλητή newOperand τύπου String βάζουμε το αποτέλεσμα: '(' + first + '+' + last + ')' , δηλαδή (5+4) όπου είναι ο νέος τελεστέος ,το οποίο το προσθέτει στην αρχή της λίστας με τη μέθοδο addFirst().



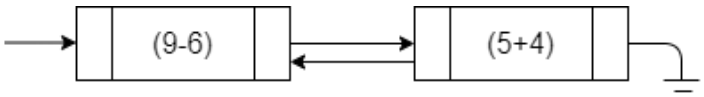
- Διαβάζει ,έπειτα, το 9 και το προσθέτει και αυτό στην αρχή της λίστας



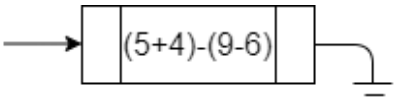
- Το ίδιο με το 6



- Διαβάζοντας τον δεύτερο τελεστή η μεταβλητή last ισούται με το 6 και η μεταβλητή first με το 9 κι έτσι η new_operand = (9-6). Οπότε η λίστα είναι :



- Ολοκληρώνοντας με τον τελευταίο τελεστή ' - ' η last = (9-6) και η first = (5+4) . Οπότε new_operand = (5+4)-(9-6)



Αφού βγούμε από την επανάληψη έχουμε μετατρέψει την μεταθεματική σε ενθεματική και την τυπώνουμε με την εντολή s.printQuee(System.out).

Υπάρχει ένα ενδεχόμενο η main να «πετάξει» εξαίρεση αν έχουμε κάποια παράσταση η οποία έχει μόνο 1 αριθμό πριν κάποιον τελεστέο. Π.χ '5+66*'. Παρότι πληροί τις προϋποθέσεις που αναφέραμε στον έλεγχο ορθότητας της εισόδου, βλέπουμε πως ο τελεστής ' + ' δεν έχει 2 στοιχεία να προσθέσει, αλλά μόνο 1, οπότε μέσω της removeFirst() της StringDoubleEndedQueueImp θα πετάξει την εξαίρεση. Με ένα try – catch block πιάνουμε το την εξαίρεση και τυπώνουμε μήνυμα και εδώ, πως η παράσταση δεν είναι σε έγκυρη μορφή.

<u>Πολυπλοκότητα</u>:

Αναφορικά με την έλεγχο για την ορθότητα του postfix (αν είναι έγκυροι όλοι οι χαρακτήρες), χρειαζόμαστε - όπως είπαμε - μια for loop με N επαναλήψεις. Έστω σταθερά c1 οι εντολές που θα εκτελεστούν στην χειρότερη περίπτωση στην for. Πολυπλοκότητα = O(c1 * N) = O(N).

Στην for loop για την μετατροπή, έχουμε μία for loop με N επαναλήψεις. Έστω c2 οι εντολές που θα εκτελεστούν στην χειρότερη περίπτωση στην for. Πολυπλοκότητα = O(c2 * N) = O(N).

Έστω c3 οι υπόλοιπες εντολές που θα εκτελεστούν σε όλο το πρόγραμμα, πέρα των επαναλήψεων. Πολυπλοκότητα = O(c3 * 1) = O(1).

Συνολική πολυπλοκότητα = O(N) + O(N) + O(1) = O(N).

Μέρος Γ

Ξεκινάμε με την συνάρτηση Complement,. Η συνάρτηση επιστρέφει το συμπλήρωμα κάθε νουκλεοτιδίου (για A,T,C,G). Αν το όρισμα που δέχεται η συνάρτηση δεν είναι επιτρεπτός χαρακτήρας νουκλεοτιδίου επιστρέφει 0.
(Πολυπλοκότητα = O(1)).

Όσο αφορά το κύριο πρόγραμμα, αρχικά ελέγχουμε την ορθότητα του string DNA που δόθηκε από τον χρήστη. Αυτό γίνεται με την μεταβλητή τύπου Boolean canBeDNA, όπου αρχικά ελέγχει αν το DNA που δόθηκε είναι σε κεφαλαία γράμματα και δε περιέχει κενό και παίρνει την κατάλληλη τιμή. Έπειτα ελέγχουμε εάν έχει κάποιο μη επιθυμητό χαρακτήρα με τη συνάρτηση Complement. Ελέγχουμε έναν -έναν τους χαρακτήρες και αν η Complement επιστρέψει 0 σημαίνει ότι δε βρήκε πουθενά A, T, C ή G επομένως η canBeDNA παίρνει τιμή false και τυπώνουμε ότι η είσοδος δεν είναι έγκυρη.

Εάν η canBeDNA είναι true τότε έχουμε σωστούς χαρακτήρες και ελέγχουμε αν το DNA είναι palindrome. Παρατηρούμε πως ένα DNA μπορεί να είναι Watson-Crick completed palindrome αν και μόνο αν έχει άρτιο πλήθος χαρακτήρων. Π.χ. το “AATCATT”. Το νουκλεοτίδιο C θα αλλάξει(όπως και όλα τα υπόλοιπα), αλλά το συγκεκριμένο είναι ακριβώς στη μέση και δε πρόκειται σε καμία περίπτωση να διαβαστεί ανάποδα το DNA και να είναι ίδιο με το αρχικό DNA.

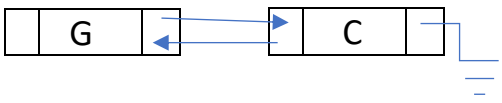
Έτσι, δεν χρειάζεται καν να ελέγξουμε το DNA και εφόσον έχει περιττό πλήθος χαρακτήρων, απλώς τυπώνουμε στην κονσόλα ότι δεν είναι Watson-Crick completed palindrome.

Πάμε λοιπόν στην περίπτωση που το πλήθος είναι άρτιο. Για την επεξήγηση του κώδικα θα χρησιμοποιούμε παράλληλα ένα παράδειγμα. Έστω DNA = “AAAGCTTT”.

i	0	1	2	3	4	5	6	7
DNA.charAt(i)	**A**	**A**	**A**	**G**	**C**	**T**	**T**	**T**

Σε μία for loop κάνουμε εισαγωγή των χαρακτήρων στη λίστα και παράλληλα ελέγχουμε αν είναι παλίνδρομη η είσοδος DNA.
Θεωρούμε ως μέσο του DNA το στοιχείο στη θέση DNA.length()/2 -1. Στο παράδειγμά μας στη θέση 3. (DNA.length() = 8 στο παράδειγμα).
Ξεκινάμε την επανάληψη από i= DNA.length()/2 -1 έως και i =0 με i--.

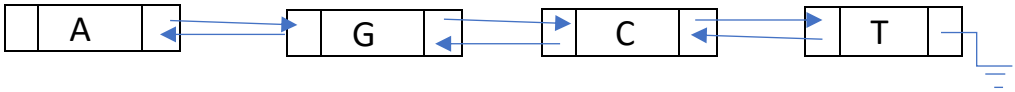
Στην 1η επανάληψη εισάγουμε στη λίστα με την μέθοδο addFirst() το στοιχείο στη θέση i και παράλληλα με την addLast() το στοιχείο στη θέση DNA.length() -1 – i. (θέση 3 και 4 αντίστοιχα.)
Στη λίστα έχουμε:



Ελέγχουμε αν το στοιχείο στη 1η ΄θεση είναι ίσο με το συμπλήρωμα του στοιχείου στη τελευταία θέση μέσω των getFirst(), getLast(), και της συνάρτησης Complement. Εφόσον είναι ίσα προχωράμε στην επόμενη επανάληψη. Αν δεν είναι ίσα δεν υπάρχει λόγος να συνεχίσουμε καθώς δεν πρόκειται να είναι παλίνδρομη, κάνουμε break και τυπώνουμε κατάλληλο μήνυμα.

Στην 2η επανάληψη(εφόσον δεν έχει γίνει break) προσθέτουμε το στοιχείο στη θέση i του DNA (όπου τώρα i = i-1) στην αρχή της λίστας και το στοιχείο στη θέση DNA.length() -1 – i στο τέλος. (στο παράδειγμά μας τα στοιχεία στη θέση 2 και 5)

Η λίστα τώρα



Συμπεριφερόμαστε όπως πριν. (συνεχίζουμε με τον ίδιο τρόπο)

Αν σε όλες τις επαναλήψεις το πρώτο στοιχείο είναι ίσο με το συμπλήρωμα του τελευταίου, το DNA είναι Watson-Crick completed palindrome και τυπώνει κατάλληλο μήνυμα.(Αν δεν είναι ίσα σε κάποια επανάληψη ,δεν υπάρχει λόγος να συνεχίσουμε, κάνουμε break και τυπώνουμε κατάλληλο μήνυμα.)

Με τον τρόπο αυτό, έχουμε καταφέρει να γίνει η εισαγωγή χαρακτήρων στη λίστα και ο έλεγχος του DNA σε μία for loop με N/2 επαναλήψεις.

<u>Πολυπλοκότητα</u>:
Αναφορικά με την έλεγχο για την ορθότητα του string DNA (αν είναι έγκυροι όλοι οι χαρακτήρες), χρειαζόμαστε - όπως είπαμε - μια for loop με N επαναλήψεις. Έστω σταθερά c1 οι εντολές που θα εκτελεστούν στην χειρότερη περίπτωση στην for. Πολυπλοκότητα = O(c1 * N) = O(N).

Στην for loop για τον έλεγχο, αν είναι παλίνδρομη η λέξη, έχουμε μία for loop με N/2 επαναλήψεις. Έστω c2 οι εντολές που θα εκτελεστούν στην χειρότερη περίπτωση στην for. Πολυπλοκότητα = O(c2 * N/2) = O(N).

Έστω c3 οι υπόλοιπες εντολές που θα εκτελεστούν σε όλο το πρόγραμμα, πέρα των επαναλήψεων. Πολυπλοκότητα = O(c3 * 1) = O(1)

Συνολική πολυπλοκότητα = O(N) + O(N) + O(1) = O(N).