***Projects: Μελέτη-Υλοποίηση και Πειραματική Αξιολόγηση Πολυδιάστατων Δομών Δεδομένων με Εφαρμογές τους***

*Σεργιάννης Παρασκευάς – Βασίλειος  
ΑΜ: 1067467  
Έτος: 5ο*

*Ερώτημα πρός υλοποίηση:*

*Ανάπτυξη γεωμετρικών πολυδιάστατων δομών*

1. **Interval trees** και **Segment trees** για interval και stabbing Queries αντίστοιχα. Μελέτη απόδοσης των βασικών πράξεων στις δύο δομές.

Για την υλοποίηση αυτόυ του ερωτήματος για αρχή κατασκευάσαμε τις 2 δομές δεδομένων με τις βασικές τους πράξεις στα αρχεια   
**IntevalTree.py** και **SegmentTree.py**.

**Interval Trees**

Ένα **Interval Tree** είναι μια προηγμένη δομή δεδομένων η οποία χρησιμοποιείται προκειμένου να καταγράψει και να αναζητήσει διαστήματα, ή intervals, με ιδιαίτερη αποτελεσματικότητα. Αυτό είναι ιδιαίτερα χρήσιμο για την αναζήτηση τμημάτων σε έναν μονοδιάστατο χώρο, και εν γένει σε περιπτώσεις όπου τα διαστήματα μπορεί να αλληλοεπικαλύπτονται ή να τέμνονται.  
  
Η δομή αυτή βρίσκει εφαρμογές σε πολλούς τομείς, όπως στην βιολογία υπολογιστών για την εύρεση γενών που είναι περιορισμένα μέσα σε κομμάτια DNA, και στη γεωγραφική πληροφορία για την εντοπίση επιφανειών που τέμνονται. Επίσης, είναι κρίσιμο εργαλείο για την επίλυση προβλημάτων που συνδέονται με τον προγραμματισμό εργασιών και την αναζήτηση ελεύθερων χρονικών διαστημάτων σε ένα ημερολόγιο.  
  
Σε γενικές γραμμές, το Interval Tree είναι αναντικατάστατο στην αντιμετώπιση προβλημάτων που έχουν να κάνουν με την αναζήτηση και τη διαχείριση διαστημάτων, επιτρέποντας την γρήγορη και ακριβή επίλυση αυτών, καθώς και την αποτελεσματική εξερεύνηση των συνδεδεμένων δεδομένων.

Πώς Λειτουργεί;

Κάθε κόμβος σε ένα Interval Tree αντιπροσωπεύει ένα διάστημα. Ο κόμβος έχει τρία χαρακτηριστικά: το διάστημα, ένα δείκτη προς το αριστερό υποδέντρο, και έναν δείκτη προς το δεξί υποδέντρο. Επίσης, κάθε κόμβος περιέχει το μέγιστο τέλος του διαστήματος του και των διαστημάτων των υποδέντρων του.

Βασικές Πράξεις:

Insert: Τα διαστήματα εισάγονται βάσει του αριστερού τελειώματός τους, σαν ένα κανονικό δυαδικό δέντρο αναζήτησης. Εφόσον τα διαστήματα εισάγονται βάσει του αριστερού τελειώματός τους, όπως σε ένα κανονικό δυαδικό δέντρο αναζήτησης, η χρονική πολυπλοκότητα είναι O(logn) σε μέση περίπτωση, όπου n είναι ο αριθμός των κόμβων στο δέντρο. Στη χειρότερη περίπτωση, μπορεί να είναι O(n).

Delete: Η διαγραφή ενός διαστήματος πραγματοποιείται χρησιμοποιώντας την αντικατάσταση με τον προηγούμενο ή τον επόμενο κόμβο στην αναζήτηση. Η χρονική πολυπλοκότητα για τη διαγραφή είναι, επίσης, O(logn) στη μέση περίπτωση και O(n) στη χειρότερη περίπτωση, λαμβάνοντας υπόψη την επαναϋπολογισμένη ισορροπία του δέντρου μετά τη διαγραφή.

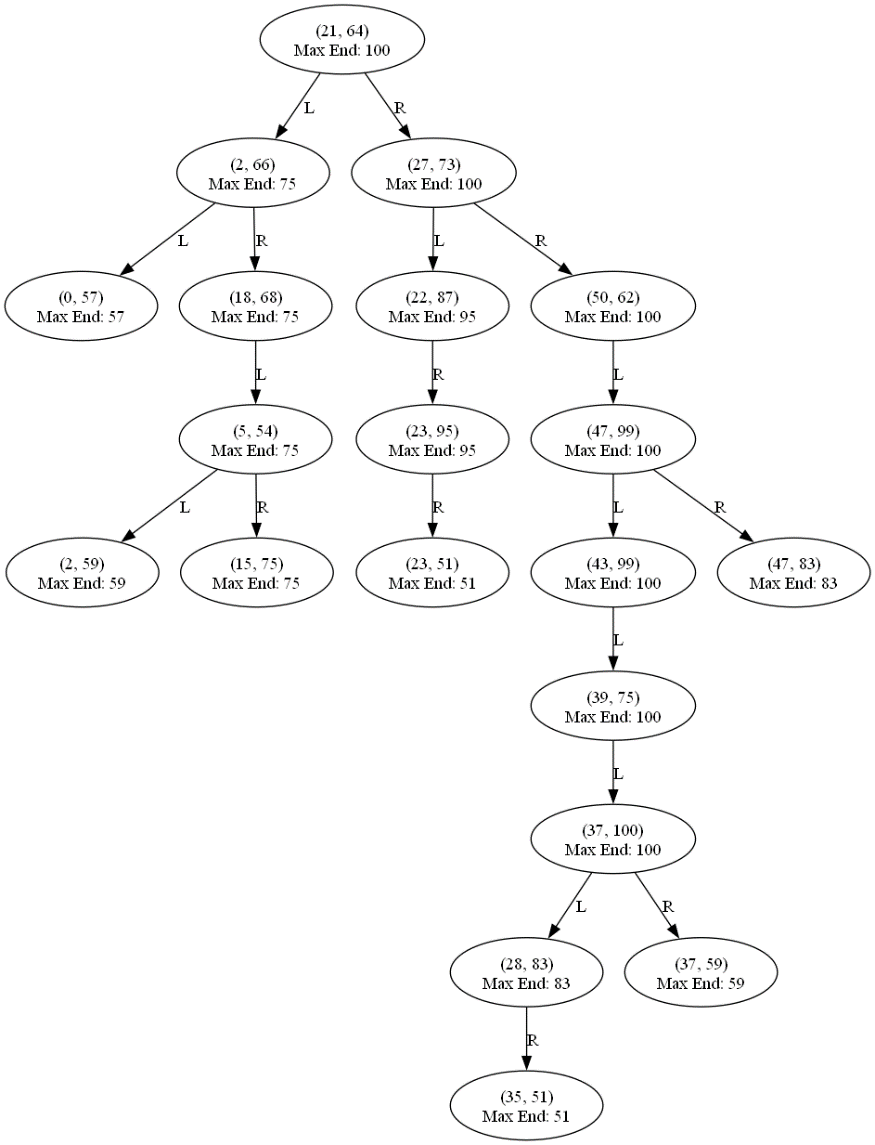
Update: Η ενημέρωση γίνεται με βάση έναν επιλεγμένο κόμβο. Αυτός διαγράφετι και αντικαθίσταται με έναν καινούριο που έχουμε ορίσει. Εφόσον η ενημέρωση περιλαμβάνει διαγραφή και εισαγωγή, η χρονική πολυπλοκότητα θα είναι, πρακτικά, η ίδια με αυτές των πράξεων διαγραφής και εισαγωγής, δηλαδή O(logn) στη μέση περίπτωση και O(n) στη χειρότερη.

Search Interval: Η αναζήτηση συνίσταται στο να βρίσκει τα διαστήματα που τέμνονται με ένα δοσμένο διάστημα. Η αναζήτηση διαστημάτων που τέμνονται με ένα δοσμένο διάστημα μπορεί να έχει χρονική πολυπλοκότητα O(logn+k), όπου k είναι το πλήθος των διαστημάτων που τέμνονται με το δοσμένο διάστημα.

Υλοποίηση:

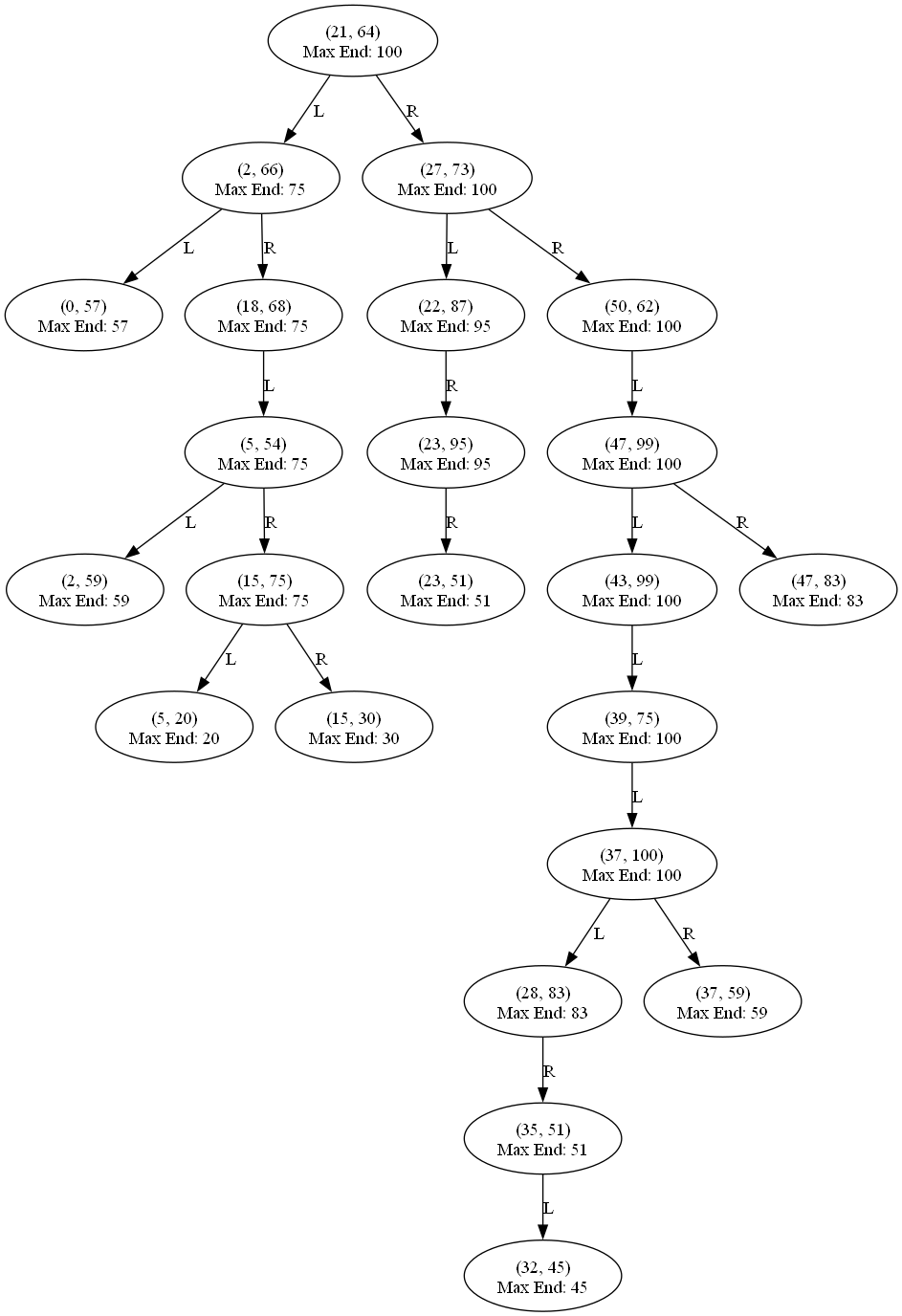
Στην υλοποίηση του κώδικα, κάθε κόμβος του δέντρου αντιπροσωπεύει ένα διάστημα και έχει επίσης έναν δείκτη προς το αριστερό και δεξί υποδέντρο. Η `insert` συνάρτηση προσθέτει ένα νέο κόμβο βάσει του αριστερού τελειώματός του διαστήματος. Η `delete` συνάρτηση διαγράφει έναν κόμβο και τον αντικαθιστά με τον προηγούμενο ή τον επόμενο κόμβο. Η `update` συνάρτηση ενημερώνει τις τιμές ενός κόμβου. Η `search` συνάρτηση στην ουσία είναι interval query και βρίσκει όλα τα διαστήματα που τέμνονται με ένα δοσμένο διάστημα.   
  
Σε κάθε στάδιο της υλοποίησης στον κώδικα δημιουργούμε ένα visualization του δέντρου. Για αρχή χρησιμοποιούμε 20 nodes, τυχαία αρχικοποιημένα για να κάνουμε build το δέντρο, το οποίο φαίνεται παρακάτω:

**Interval Tree after Build**



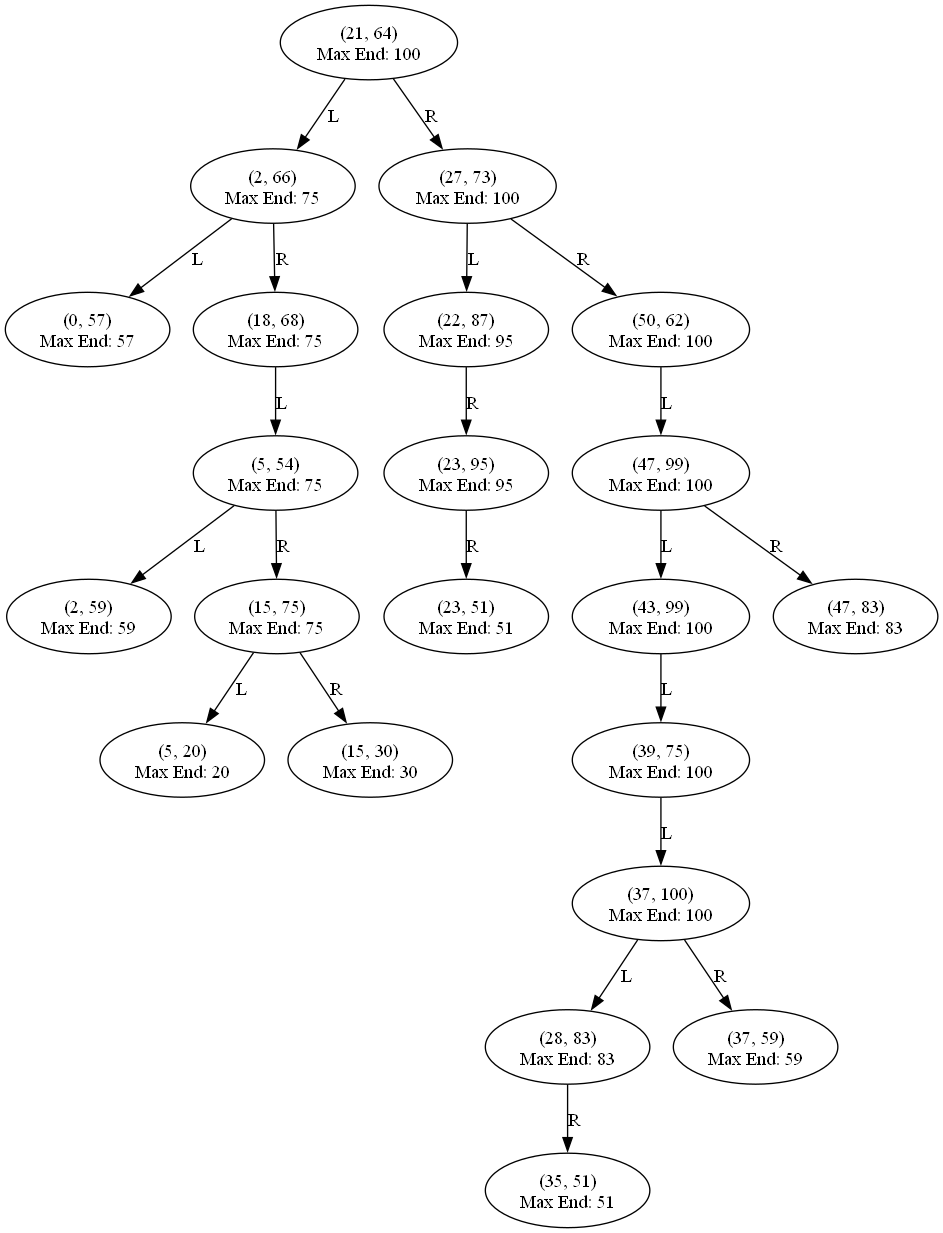
Στη συνέχεια ορίζουμε τα ακόλουθα intervals για να τα κάνουμε insert στο interval tree: [(15, 30), (32, 45), (5, 20)], και το δέντρο μετά τα inserts είναι το ακόλουθο:

**Interval Tree after Inserts**



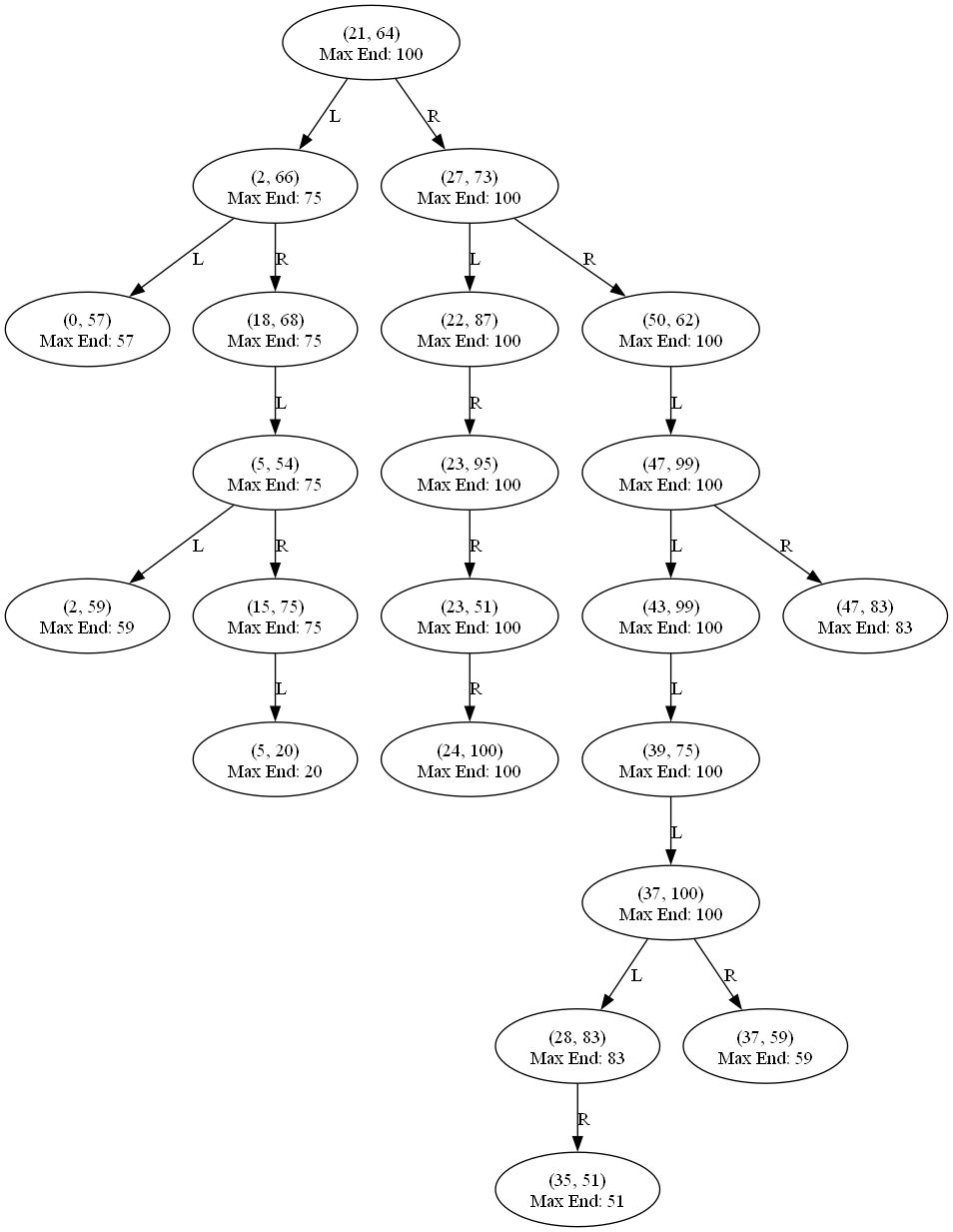
Έπειτα διαγράφουμε το node (32, 45) που κάναμε insert πρίν και παίρνουμε την καινούρια εικόνα:

**Interval Tree after Delete**



Τέλος κάνουμε update το interval (15, 30) που επίσης κάναμε insert προηγουμένως και το αντικαθιστούμε με το (24, 100).

**Interval Tree after Update**



Επίσης κάνουμε και ένα interval query με το (10, 20) για να βρούμε με ποια intervals κάνει intersect το συγκεκριμένο διάστημα και παίρνουμε τα ακόλουθα αποτελέσματα:

Found interval that intersects with (10, 20):   
[(2, 66), (0, 57), (18, 68), (5, 54), (2, 59), (15, 75), (5, 20)]  
  
  
  
  
  
Απόδοση:  
Καθόλη τη διάρκεια του τρεξίματος κρατάμε μετρήσεις σχετικά με το πόσο χρόνο και πόση μνήμη απαιτεί κάθε πράξη. Πιο αναλυτικά:

**Για n=20**

Memory used to build the tree: 0.0195312500 MiB

Time taken to build the tree: 0.0002216999 seconds

Memory used to insert interval (15, 30): 0.0039062500 MiB

Time taken to insert interval (15, 30): 0.0000528001 seconds

Memory used to insert interval (32, 45): 0.0039062500 MiB

Time taken to insert interval (32, 45): 0.0000290000 seconds

Memory used to insert interval (5, 20): 0.0000000000 MiB

Time taken to insert interval (5, 20): 0.0000164000 seconds

Memory used to delete interval (32, 45): 0.0000000000 MiB

Time taken to delete an interval: 0.0001052000 seconds

Memory used to update interval (15, 30) to (24, 100): 0.0000000000 MiB

Time taken to update an interval: 0.0000931000 seconds

Memory used to search an interval: 0.0000000000 MiB

Time taken to search an interval: 0.0000805000 seconds

**Για n=1000**

Memory used to build the tree: 0.1289062500 MiB

Time taken to build the tree: 0.0226000000 seconds

Memory used to insert interval (15, 30): 0.0039062500 MiB

Time taken to insert interval (15, 30): 0.0000939000 seconds

Memory used to insert interval (32, 45): 0.0000000000 MiB

Time taken to insert interval (32, 45): 0.0000559000 seconds

Memory used to insert interval (5, 20): 0.0000000000 MiB

Time taken to insert interval (5, 20): 0.0000520999 seconds

Memory used to delete interval (32, 45): 0.0000000000 MiB

Time taken to delete an interval: 0.0001594000 seconds

Memory used to update interval (15, 30) to (24, 100): 0.0000000000 MiB

Time taken to update an interval: 0.0002062999 seconds

Memory used to search an interval: 0.0078125000 MiB

Time taken to search an interval: 0.0008636999 seconds

**Για n=5000**

Memory used to build the tree: 0.5351562500 MiB

Time taken to build the tree: 0.4228657000 seconds

Memory used to insert interval (15, 30): 0.0000000000 MiB

Time taken to insert interval (15, 30): 0.0002634000 seconds

Memory used to insert interval (32, 45): 0.0000000000 MiB

Time taken to insert interval (32, 45): 0.0002305000 seconds

Memory used to insert interval (5, 20): 0.0000000000 MiB

Time taken to insert interval (5, 20): 0.0002086000 seconds

Memory used to delete interval (32, 45): 0.0000000000 MiB

Time taken to delete an interval: 0.0003974000 seconds

Memory used to update interval (15, 30) to (24, 100): 0.0000000000 MiB

Time taken to update an interval: 0.0005532999 seconds

Memory used to search an interval: 0.0781250000 MiB

Time taken to search an interval: 0.0044643000 seconds

Αξιολόγηση:

Με βάση αυτά τα αποτελέσματα oι χρόνοι εκτέλεσης και η χρησιμοποιούμενη μνήμη φαίνεται να αυξάνονται, όπως είναι λογικό, με την αύξηση των στοιχείων (n).

Χρόνος Εκτέλεσης:  
Ο χρόνος εκτέλεσης για την κατασκευή του δέντρου αυξάνεται από 0.0002216999 seconds για n=20 σε 0.4228657000 seconds για n=5000. Οι χρόνοι εισαγωγής, διαγραφής, ενημέρωσης, και αναζήτησης είναι αρκετά γρήγοροι, ακόμη και για n=5000, αλλά παρουσιάζουν μια ελαφρώς αυξανόμενη τάση.

Χρησιμοποιούμενη Μνήμη:  
Η χρησιμοποιούμενη μνήμη για την κατασκευή του δέντρου αυξάνεται από 0.0195312500 MiB για n=20 σε 0.5351562500 MiB για n=5000. Η χρησιμοποιούμενη μνήμη για τις άλλες λειτουργίες παραμένει σχετικά σταθερή.

**Συμπεράσματα**

Το interval tree που φτιάξαμε φαίνεται να έχει καλή απόδοση τόσο σε χρόνο όσο και σε μνήμη για τις παραπάνω περιπτώσεις. Είναι σημαντικό να σημειωθεί ότι, ενώ ο χρόνος και η μνήμη που χρησιμοποιούνται για την κατασκευή του δέντρου αυξάνονται με το μέγεθος του δέντρου, οι χρόνοι για τις βασικές λειτουργίες (εισαγωγή, διαγραφή, ενημέρωση, αναζήτηση) παραμένουν σε λογικά πλαίσια, ακόμη και για μεγαλύτερα n, το οποίο είναι ένα θετικό σημείο για την αποτελεσματικότητα του δέντρου.

**Segment Trees**

Ένα **Segment tree** είναι ένας τύπος δυαδικού δέντρου που χρησιμοποιείται για την αποθήκευση και αναζήτηση πληροφοριών σχετικά με διαστήματα ή τμήματα ενός μονοδιάστατου χώρου. Χρησιμοποιείται για την εκτέλεση ερωτημάτων εύρους σε έναν πίνακα ή λίστα δεδομένων, όπως η εύρεση του αθροίσματος, της μέγιστης ή της ελάχιστης τιμής μιας περιοχής στοιχείων. Οι κόμβοι φύλλων του δέντρου αντιπροσωπεύουν μεμονωμένα στοιχεία του πίνακα εισόδου και οι εσωτερικοί κόμβοι αντιπροσωπεύουν εύρη στοιχείων. Το δέντρο μπορεί να κατασκευαστεί σε χρόνο O(n log n) και υποστηρίζει ερωτήματα σε χρόνο O(log n), καθιστώντας το μια αποτελεσματική δομή δεδομένων για την επεξεργασία μεγάλων ποσοτήτων δεδομένων.  
  
  
Πως Δουλεύει:

Το Segment Tree διαίρει το σύνολο των στοιχείων του στη μέση και δημιουργεί δυο υποδένδρα, ένα για το αριστερό υποσύνολο και ένα για το δεξί υποσύνολο. Αυτή η διαδικασία επαναλαμβάνεται αναδρομικά μέχρι να φτάσουμε στα φύλλα του δένδρου, τα οποία αντιπροσωπεύουν τα στοιχεία της αρχικής ακολουθίας.

Βασικές Πράξεις:

Insert:   
Για να εισάγουμε ένα καινούργιο διάστημα, πρέπει να το τοποθετήσουμε στο σωστό φύλλο και, αν χρειάζεται, να ενημερώσουμε τις τιμές των γονικών κόμβων. Η πράξη αυτή έχει χρονική πολυπλοκότητα O(log n).

Delete:   
Η διαγραφή ενός διαστήματος πραγματοποιείται εντοπίζοντας το αντίστοιχο φύλλο και αφαιρώντας το από το δέντρο. Ενδέχεται να απαιτείται ενημέρωση των γονικών κόμβων. Αυτή η πράξη έχει επίσης χρονική πολυπλοκότητα O(log n).  
  
Update:  
Για να ενημερώσουμε την τιμή ενός διαστήματος, βρίσκουμε το αντίστοιχο φύλλο, αλλάζουμε την τιμή του, και ενημερώνουμε τις τιμές των γονικών κόμβων όπου χρειάζεται. Η πράξη αυτή έχει χρονική πολυπλοκότητα O(log n).

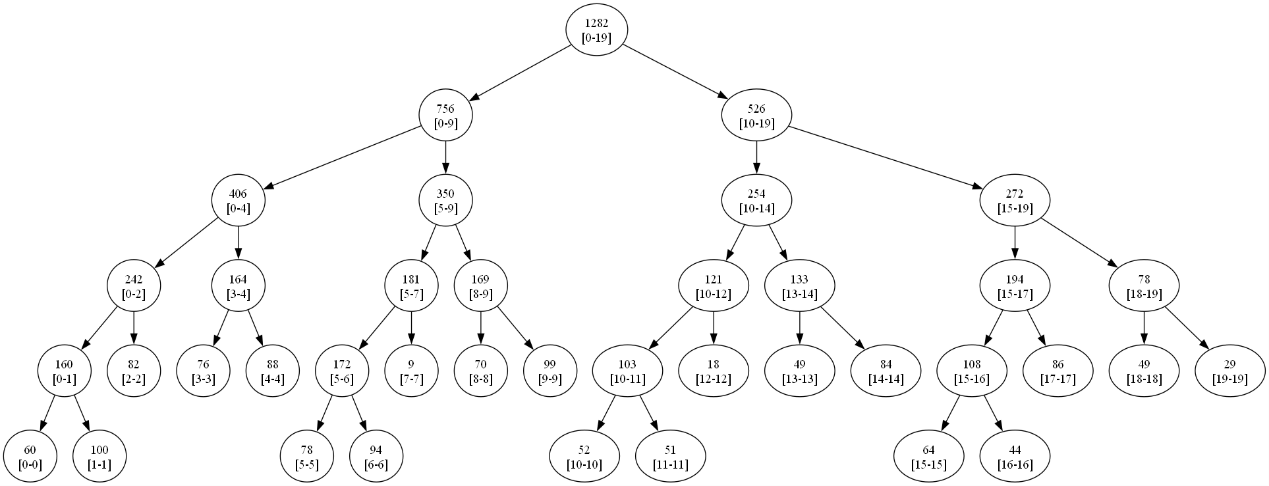
Stabbing Query:   
Ένα Stabbing Query αναζητά όλα τα διαστήματα που περιέχουν ένα συγκεκριμένο σημείο. Για παράδειγμα, μπορεί να ρωτάμε ποια διαστήματα περιέχουν το σημείο x. Η επεξεργασία ενός τέτοιου ερωτήματος σε ένα Segment Tree πραγματοποιείται ελέγχοντας τους κόμβους που τέμνονται με το σημείο και συλλέγοντας τα αποτελέσματα. Η χρονική πολυπλοκότητα για ένα τέτοιο ερώτημα είναι τάξης O(logn+k), όπου k είναι το πλήθος των τομών διαστημάτων.

Range Query:  
Ένα Range Query αναζητά το σύνολο των τιμών (ή άλλης ακολουθίας) μέσα σε ένα συγκεκριμένο διάστημα. Για παράδειγμα, μπορεί να ρωτάμε για το άθροισμα, το μέγιστο ή το ελάχιστο των τιμών μέσα στο διάστημα [a,b]. Η επεξεργασία ενός τέτοιου ερωτήματος σε ένα Segment Tree γίνεται εντοπίζοντας και συνδυάζοντας τις τιμές των κόμβων που καλύπτονται πλήρως ή μερικώς από το διάστημα ερωτήματος. Η χρονική πολυπλοκότητα για ένα τέτοιο ερώτημα είναι O(logn), καθώς μπορούμε να αγνοήσουμε τα φύλλα που δεν τέμνονται με το διάστημα ερωτήματος.

Υλοποίηση:

Ο κώδικας μας ορίζει μια κλάση SegmentTree με διάφορες μεθόδους όπως build, insert, delete, update, stabbing\_query και range\_query.   
- Η build χρησιμοποιείται για τη δημιουργία του δέντρου από έναν πίνακα.   
- Η insert και delete χρησιμοποιούνται για την εισαγωγή και διαγραφή στοιχείων αντίστοιχα.   
- Η update χρησιμοποιείται για την ενημέρωση τιμής ενός στοιχείου.   
- Η stabbing\_query χρησιμοποιείται για να επιστρέψει την τιμή ενός στοιχείου στον αρχικό πίνακα.  
- Η range\_query χρησιμοποιείται για να επιστρέψει το άθροισμα των στοιχείων σε ένα εύρος του αρχικού πίνακα.  
Παράλληλα χρονομετρούμε την κάθε συνάρτηση καθώς και τις χωρικές της απαιτήσεις κατά την υλοποίηση, όπως και κάνουμε οπτικοποίηση της δομής σε κάθε στάδιο υλοποίησης.  
Για αρχή χρησιμοποιούμε 20 nodes, τυχαία αρχικοποιημένα για να κάνουμε build το δέντρο όπως και το προηγούμενο, το οποίο φαίνεται παρακάτω:

**Segment Tree after Build**

****

Στη συνέχεια κάνουμε insert το node με τιμή 50: **Segment Tree after Insert**

**A diagram of a structure

Description automatically generated**  
  
  
  
  
Έπειτα διαγράφουμε το στοιχείο στο 2o index του δέντρου και ξαναφτιάχνουμε το δέντρο. **Segment Tree after Delete  
A diagram of a structure

Description automatically generated**

Στη συνέχεια κάνουμε update την τιμή του node στο index=2, σε 55 και ξαναζωγραφίζουμε το δέντρο. **Segment Tree after Update**

**A diagram of a structure

Description automatically generated**

**Τέλος εκτελούμε τα stabbing και range queries:**  
Stabbing query result for index 11: 18

Range query result for sum in range (2, 10): 596

Απόδοση:  
Καθόλη τη διάρκεια του τρεξίματος κρατάμε μετρήσεις σχετικά με το πόσο χρόνο και πόση μνήμη απαιτεί κάθε πράξη. Πιο αναλυτικά:  
  
**Για n=20**  
Memory used to build the tree: 0.0117187500 MiB

Time taken to build the tree: 0.0000710001 seconds

Memory used to insert an element: 0.0000000000 MiB

Time taken to insert an element: 0.0001494001 seconds

Memory used to delete an element: 0.0000000000 MiB

Time taken to delete an element: 0.0001248000 seconds

Memory used to update an element: 0.0000000000 MiB

Time taken to update an element: 0.0000702000 seconds

Memory used to perform a stabbing query: 0.0000000000 MiB

Time taken to perform a stabbing query: 0.0001331000 seconds

Memory used to perform a range query: 0.0078125000 MiB

Time taken to perform a range query: 0.0000961999 seconds  
  
  
**Για n=1000**Memory used to build the tree: 0.0117187500 MiB

Time taken to build the tree: 0.0024791000 seconds

Memory used to insert an element: 0.0156250000 MiB

Time taken to insert an element: 0.0046618001 seconds

Memory used to delete an element: 0.0000000000 MiB

Time taken to delete an element: 0.0032147000 seconds

Memory used to update an element: 0.0000000000 MiB

Time taken to update an element: 0.0001321000 seconds

Memory used to perform a stabbing query: 0.0000000000 MiB

Time taken to perform a stabbing query: 0.0001167000 seconds

Memory used to perform a range query: 0.0000000000 MiB

Time taken to perform a range query: 0.0001119999 seconds

**Για n=5000**

Memory used to build the tree: 0.0117187500 MiB

Time taken to build the tree: 0.0124086000 seconds

Memory used to insert an element: 0.2070312500 MiB

Time taken to insert an element: 0.0259879001 seconds

Memory used to delete an element: 0.0000000000 MiB

Time taken to delete an element: 0.0200135000 seconds

Memory used to update an element: 0.0000000000 MiB

Time taken to update an element: 0.0000792000 seconds

Memory used to perform a stabbing query: 0.0000000000 MiB

Time taken to perform a stabbing query: 0.0001764001 seconds

Memory used to perform a range query: 0.0039062500 MiB

Time taken to perform a range query: 0.0001123000 seconds

Αξιολόγηση:

Build:  
Η χρήση μνήμης και ο χρόνος αυξάνονται λογικά με το μέγεθος του δέντρου, είναι αναμενόμενο και σχετίζεται με την πολυπλοκότητα της κατασκευής του δέντρου O(nlogn).

Insert:  
Η χρήση μνήμης και ο χρόνος αυξάνονται ενώ το μέγεθος του δέντρου αυξάνεται. Σε μεγάλο δέντρο (n=5000), η χρήση της μνήμης καταγράφεται πολύ υψηλότερα, πιθανώς λόγω των αναδιοργανώσεων των δεδομένων και των επανακατασκευών τμημάτων του δέντρου.

Delete και Update:  
Οι χρόνοι εκτέλεσης είναι σχετικά χαμηλοί για όλα τα μεγέθη, με τη διαγραφή να είναι λίγο πιο αργή στα μεγάλα μεγέθη.  
  
Search:  
Ο χρόνος είναι πολύ χαμηλός και φαίνεται να είναι αρκετά αποδοτικός, ακόμη και για μεγάλες τιμές του n.

**Συμπεράσματα**

Συνολικά, το Segment Tree δείχνει να είναι αρκετά αποδοτικό για τις βασικές λειτουργίες, εκτός ίσως από την εισαγωγή στις περιπτώσεις όπου το n είναι μεγάλο, όπου η χρήση της μνήμης μπορεί να γίνει σημαντική.