ΥΠΟΛΟΓΙΣΤΙΚΗ ΓΕΩΜΕΤΡΙΑ

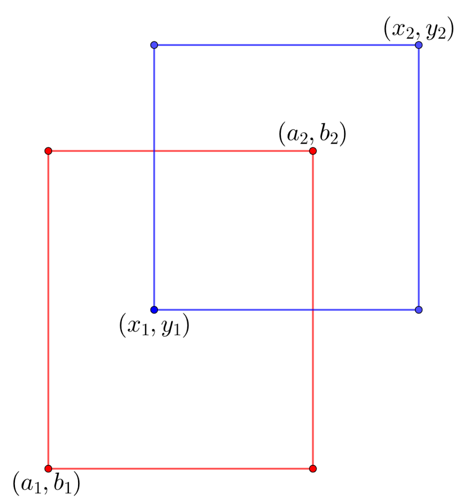
3ο Σετ Ασκήσεων

Καλαμαράκης Θεόδωρος: 2018030022

Άσκηση 1

Για την υλοποίηση θα χρησιμοποιήσουμε priority search tree και το πρόβλημα θα λυθεί ως μια παραλλαγή του προβλήματος αναζήτησης τριών πλευρών. Η συνθήκη για να τέμνονται δύο ορθογώνια με συντεταγμένες (a_1, b_1, a_2, b_2) και (x_1, y_1, x_2, y_2) (οπου $(a_1, b_1), (x_1, y_1)$ οι κάτω αριστερά κορυφές και $(a_2, b_2), (x_2, y_2)$ οι πάνω δεξιά), ειναι η:

$$(b_1 \leq y_2 \wedge b_2 \geq y_1) \wedge (a_1 \leq x_2 \wedge a_2 \geq x_1)$$



Σχήμα 1

Θα προσεγγίσουμε το πρόβλημα χρησιμοποιώντας την δομή priority search tree με τον εξής τρόπο:

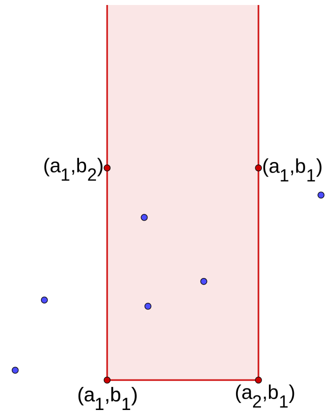
Έχουμε ένα σύνολο S με n rectangles (x_1, y_1, x_2, y_2) τα οποία τοποθετούμε στο priority search tree αλλά στην πράξη το κάθε rectangle θα αντιπροσωπεύεται μόνο απο το πάνω δεξιά σημείο του (x_2, y_2) δηλαδή θα είναι σαν να εφαρμόζουμε priority search tree στα σημεία (x_2, y_2). (θυμίζουμε οτι ενα rectangle το ορίζουμε χρησιμοποιώντας την κάτω αριστερα κορυφη του (x_1, y_1) και τη πάνω δεξια (x_2, y_2))

- Κάθε κόμβος έχει 2 κλειδιά, $x_{2p} \in \mathbb{R}$ και $q \in U$.
- $x_{2p} =$ η median x_2-συντεταγμένη
- $q = (x_1, y_1, x_2, y_2) \in U$ το σημείο με τη μεγαλύτερη y_2-συντεταγμένη
- Χωρίζουμε το $U - q$ ως προς το x_{2p} και φτιάχνουμε αναδρομικά τα υποδέντρα.
- binary search tree ως προς x_2 και priority heap ώς προς y_2

Η κατασκευή του δέντρου εχει κόστος $O(n \log n)$.

Έστω οτι θέλουμε να αναζητήσουμε τις τομές ενός rectangle απο το S με συντεταγμένες (a_1, b_1, a_2, b_2).

- Αφαιρούμε απο το δέντρο τον κόμβο που
  αντιστοιχεί στο rectangle $(a_1, b_1, a_2, b_2) \quad \rightarrow \quad O(\log n)$
- Κάνουμε αναζήτηση τριών πλευρών στο query $[a_1, a_2] \times [b_1, +\infty]$ όπως φαίνεται στο σχήμα. $\quad \rightarrow \quad O(\log n + k)$

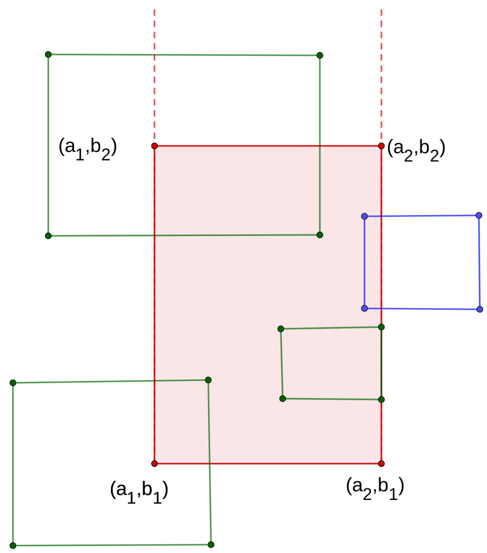


Σχήμα 2

- Σε αντίθεση με την απλή αναζήτηση τριών πλευρών, για κάθε σημείο q που εντοπίζουμε να βρίσκεται εντός των τριών πλευρών, πρίν το κάνουμε report, θα απαιτήσουμε επιπλέον να ισχύει $q.y_1 \leq b_2$

- Μόλις ολοκληρωθεί η αναζήτηση τριών πλευρών θα ξαναεισάγουμε το rectangle (a_1, b_1, a_2, b_2) στο δέντρο $\quad \rightarrow \quad O(\log n)$

Με την παραπάνω αναζήτηση επιτυγχάνουμε να εντοπίσουμε όλα τα rectangles (x_1, y_1, x_2, y_2) τα οποία τέμνουνε απο αριστερά το (a_1, b_1, a_2, b_2), είτε περικλείονται μέσα σε αυτο. (δηλαδή $a_1 \leq x_2 \leq a_2 \wedge (b_1 \leq y_2 \wedge b_2 \geq y_1))$. Τέτοια rectangles είναι τα πράσινα του παρακάτω σχήματος.



Σχήμα 3

Το κόστος της αναζήτησης των αριστερών τομών για το rectangle $i = (a_1, b_1, a_2, b_2)$ είναι

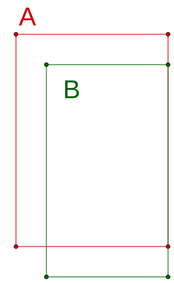
$$Q_i(n) = O(\log n) + O(\log n) + O(\log n + k_i) = O(\log n + k_i)$$

όπου k_i το πλήθος τών τομών. Η αναζήτηση αυτή θα επαναληφθέι για όλα τα rectangles του S, οπότε οι τομές σαν αυτη του κόκκινου με το μπλέ στο Σχήμα 3 θα εντοπιστούν όταν γίνει αναζήτηση τρίων πλευρών με βάση το μπλέ rectangle (θα γίνει report η τομή διότι το κόκκινο θα τέμνει απο αριστερά το μπλέ)

Το συνολικό κόστος αναζήτησης θα είναι

$$Q(n) = \sum_{i=1}^{n} O(\log n + k_i) = O(n \log n + k_1 + k_2 + ... + k_n) = O(n \log n + k_)$$

Πρωτού παραθέσουμε ψευδοκώδικα κάνουμε την εξής παρατήρηση: Η τομή δυο rectangles A και B για τα οποία ισχύει $A.x_2 = B.x_2$ θα μετρηθεί δύο φόρες διότι μια το A θα θεωρηθεί οτι τέμνει απο αριστερά το B και μία το B θα θεωρηθεί οτι τέμνει απο αριστερά το A



Σχήμα 4

Για να αντιμετωπίσουμε αυτό το πρόβλημα βάζουμε ένα επιπλέον πεδιό-flag σε κάθε κόμβο του δέντρου, που θα αντιστοιχεί στο rectangle q του συγκεκριμένου κόμβου και θα μας πληροφορεί για το αν το q έχει χρησιμοποιήθει για αναζήτηση τρίων πλευρών. Στην κατασκευή του δέντρου το flag θα είναι FALSE για όλα τα q και θα γίνεται TRUE στην επανεισαγωγή του q. Ο ψευδοκώδικας είναι ο παρακάτω:

Algorithm to find intersecting rectangles pairs

```
function FIND_INTERSECTING_RECTANGLES(SET OF RECTANGLES)
    for R in SET OF RECTANGLES do                    ▷ Construction of the tree in O(n log n)
        PST.insert(R, FLAG = FALSE)
    for R in SET OF RECTANGLES do                    ▷ Three side search for R
        PST.remove(R)
        SEARCH3S(R, PST)
        PST.insert(R, FLAG = TRUE)
function SEARCH3S(R, t)
    if t is None OR t.q.y_2 < R.y_1 then
        return
    if R.x_1 < t.x_2p then
        SEARCH3S(R, t.left)
    if R.x_1 ≤ t.q.x_2 ≤ R.x_2 AND t.q.y_2 ≥ R.y_1 AND t.q.y_1 ≤ R.y_2 then

        if NOT(R.x_2 = t.q.x_2 AND FLAG = TRUE ) then
            report(R, t.q)                           ▷ Report intersection pair (R, t.q)

    if R.x_2 > t.x_2p then
        SEARCH3S(R, t.right)
```

Το συνολικό κόστος του αλγορίθμου θα είναι το κόστος κατασκευής του δέντρου + το συνολικό κόστος αναζήτησης

$$O(n \log n) + O(n \log n + k) = O(n \log n + k)$$

Άσκηση 2

Η σχέση θα αποδειχθεί μέσω επαγωγής.

- για $d = 1$

$$S(1, n) = \Theta(n) = \Theta(n \log^{d-1} n)$$

- για $d = 2$

$$S(2, n) = 2S(2, n/2) + S(1, n) = 2S(2, n/2) + \Theta(n)$$

  Απο master theorem προκύπτει οτι $S(2, n) = \Theta(n \log n) = \Theta(n \log^{d-1} n)$

- Εστω οτι ισχύει για $d = k$ δηλαδή $S(k, n) = \Theta(n \log^{k-1} n)$
  Τότε αρκει να δείξουμε οτι για $d = k + 1$ ισχύει $S(k + 1, n) = \Theta(n \log^{k} n)$:

$$S(k+1, n) = 2S(k+1, n/2) + S(k, n) = 2S(k+1, n/2) + \Theta(n \log^{k-1} n)$$

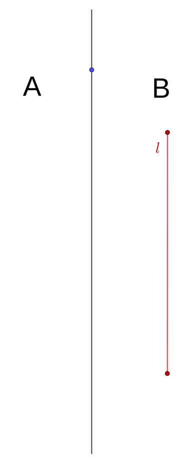
  Απο master theorem προκύπτει οτι $\gamma = \log_2 2 = 1$
  Αρα πρόκειται για την δεύτερη περίπτωση του master theorem, όπου $f(n) = \Theta(n \log^{k-1} n) = \Theta(n^{\gamma} \log^{k-1} n)$ με $k - 1 > -1$
  Συνεπώς $S(k + 1, n) = \Theta(n \log^{k} n)$

Άσκηση 3

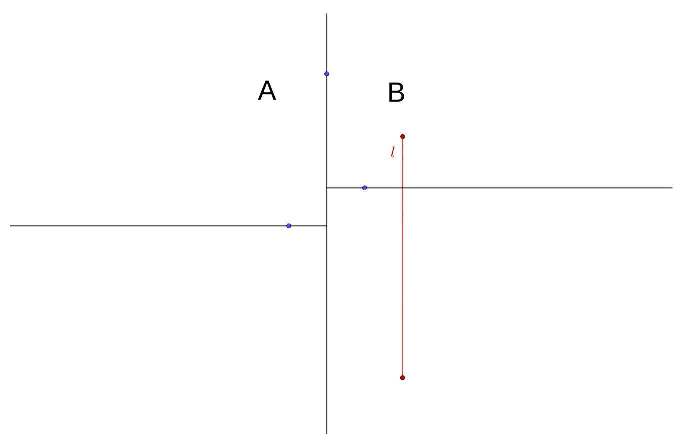
(a) Υπολογισμός κόστους αναζήτησης για αναζήτηση rectangular range query σε k-d-tree:

Το πρόβλημα αναζήτησης rectangular range query ανάγεται στο να εντοπίσουμε όλες τις περιοχές, στις οποίες χωρίζει τον χώρο το k-d-tree, οι οποίες τέμνονται απο τα σύνορα του query. Αυτό διότι, οτάν εντοπιστούν οι περιοχές που τέμνουνε τα σύνορα του query, τότε τα εσωτερικά σημεία του query (συμπεριλαμβανομένων και των σημείων που βρίσκονται σε περιοχές που περικλείονται απο το query) θα ανακτηθούν σε χρόνο $O(k)$
Για τον εντοπισμό των περιοχών στα σύνορα του query για δύο διαστάσεις ($d = 2$) εργαζόμαστε ώς εξής:
Θα υπολογίσουμε την αναδρομική σχέση για τις περιοχές που τέμνονται απο μια κέθετη πλευρά του τετραγωνικου query (για τις υπόλοιπες τρείς πλευρές η διαδικασία θα είναι ακριβώς η ίδια).

Οτάν κάνουμε κάθετο διαχωρισμό του χώρου στο πρώτο επίπεδο (δύο παιδιά τών $\frac{n}{2}$ στοιχείων) του δέντρου τότε η κάθετη γραμμή l θά τμήσει μόνο ένα απο τα δύο παιδιά (A και B όπως φαίνεται στο παρακάτω σχήμα).



Αντίθετα,όταν γίνει ο οριζόντιος διαχωρισμός του χώρου στό δεύτερο επίπεδο(τέσσερα παιδιά τών $\frac{n}{4}$ στοιχείων), τότε η γράμμη l θα τμήσει (στην χειρότερη περίπτωση) και τα δύο παιδιά του παιδιού B που ήδη έτεμνε απο πρίν, ενώ δεν θα τμήσει κανένα παιδι του παιδιού A.



Άρα η αναδρομή μέχρι το δεύτερο επίπεδο θα είναι

$$Q(n) = 2 + 2Q(\frac{n}{4})$$

Απο το δεύτερο επίπεδο και έπειτα ο οριζόντιος και κάθετος διαχωρισμός εναλλάσσονται κυκλικά, άρα η παραπάνω ανδρομή θα είναι και η τελική αναδρομή του αλγορίθμου για $d = 2$. Γενικεύοντας στις d διαστάσεις μπορούμε να πούμε οτι για κάθε διαχωρίσμο σε διάσταση παράλληλη στη πλευρά του query που μελετάμε, το πλήθος των περιοχών που θα τέμνονται απο αυτήν την πλευρα δεν θα αλλάζει. Αντίθετα, με διαχωρισμό σε οποιαδήποτε άλλη διάσταση, το πλήθος τών περιοχών που τέμνουνται απο αυτην την πλευρα θα διπλασιάζεται. Άρα κατεβαίνοντας d επίπεδα στο δέντρο, το πλήθος των περιοχών που θα τέμνονται απο την πλευρά του query θα είναι 2^{d-1} και κατά συνέπεια η αναδρομή θα είναι

$$Q(n) = 2^{d-1} + 2^{d-1}Q(\frac{n}{2^d})$$

εφαρμόζοντας master theorem έχουμε:

$$\gamma = \frac{\log 2^{d-1}}{\log 2^d} = \frac{d-1}{d}$$
$$\rightarrow n^{\frac{d-1}{d}} > 2^{d-1}$$

Αρα

$$Q(n) = O(n^{\frac{d-1}{d}}) = O(n^{1-\frac{1}{d}})$$

Σε συνδιασμό με τον χρόνο $O(k)$ που χρειαζόμαστε για την ανάκτηση των k εσωτερικών στοιχείων, η τελική πολυπλοκότητα αναζήτησης θα είναι $O(n^{1-\frac{1}{d}} + k)$

(b) Υπολογισμός κόστους κατασκευής k-d-tree:
Για την κατασκευή του δέντρου, υπολογίζουμε την median των n σημειών σε χρόνο $O(n)$ και χωρίζουμε τον χώρο στα σε δυο υποχωρους με $\frac{n}{2}$ σημεία βάσει του median. Επαναλαμβάνουμε την διαδικασία για τον κάθε υποχώρο. Συνεπώς η τελική αναδρομή θα είναι

$$T(n) = 2T(\frac{n}{2}) + O(n)$$

Αρα απο master theorem θα προκύψει οτι το κόστος κατασκευης θα είναι $O(n \log n)$

Άσκηση 4

Σε ένα απλό range tree ο χρόνος αναζήτησης ενός range query είναι $O(\log^d n + k)$, συνεπώς στην δική μας παραλλαγή θέλουμε να εξαλήψουμε τον όρο $O(k)$. Ο όρος $O(k)$ προκύπτει απο τις διαδοχικές αναφορές των σημείων που εντοπίζουμε εντός του range query, όταν έχουμε φτάσει στο 1-dimensional range tree. Για παράδειγμα, έστω οτι έχουμε το d-dimensional range tree στο οποίο τα κλειδιά βρίσκονται στα φύλλα και σε έναν εσωτερικό κόμβο i αποθηκεύουμε το range ρ_i και την median τιμή z_i του υποδέντρου με ρίζα τον κόμβο i. Όταν κάνουμε ένα ερώτημα $(\mathbf{a}, \mathbf{b})$ στο δέντρο $\mathcal{R}_d$, τότε όταν φτάνουμε σε έναν nested κόμβο i όπου $\rho_i \subseteq (a_d, b_d)$ σύνεχίζουμε ρωτώντας το δέντρο $\mathcal{R}_{d-1}$. Οταν φτάσουμε σε nested κόμβο του $\mathcal{R}_1$ τότε κάνουμε διαδοχικά report όλα του τα κλειδιά σε χρόνο $O(k)$, όπως ακριβώς θα κάναμε σε ένα απλό binary search tree. Στην δική μας παραλλαγή όμως, μας ενδιαφέρει μόνο το πλήθος τών κλειδιών εντός του range και οχι να τα προσδιορίσουμε ένα ένα. Άρα μπορουμε να αποφύγουμε την διαδικασία τών διαδοχικών reports προσθέτοντας απλά μια μεταβλητή c_i σε κάθε κόμβο i η οποία μας πληροφορεί για το πλήθος τών κλειδιών που βρίσκονται στο υποδέντρο με ρίζα το i. Ετσί όταν φτάσουμε σε nested κόμβο i του $\mathcal{R}_1$ άπλα κοιτάμε την μεταβλητή c_i και επιστρέφουμε χώρις να κάνουμε report τα κλειδία. Η αναζήτηση αυτή θα τερματίσει σε χρόνο $O(\log^d n)$.

Η κατασκευή αυτού του δέντρου θα είναι ακριβώς ίδια με του απλού range tree με την προσθήκη του υπολογισμού της μεταβλητής c_i το οποίο θέλει χρόνο $O(n)$ Αναλυτικότερα για το κόστος κατασκευής $T(n)$ έχουμε:

- Ρίζα: $\rho_0 = (-\infty, \infty)$
  $U_0 = U$
  $z_0 = \text{median}\{x_d | x \in U_0\} \quad \rightarrow \quad O(n)$
  $c_0 = |U_0|$ (το πλήθος τών $x \in U_0$) $\quad \rightarrow \quad O(n)$

- Παιδιά της ρίζας :
  $\rho_1 = (-\infty, z_0)$ και $\rho_2 = (z_0, \infty)$
  και αντίστοιχα U_1, U_2
  $U_1 = \{x \in U_0 | x_d \in (-\infty, z_0]\} \quad \rightarrow \quad T(n/2)$
  $U_2 = \{x \in U_0 | x_d \in (z_0, \infty)\} \quad \rightarrow \quad T(n/2)$

- Συνεχίζουμε αναδρομικά ώσπου $|U_i| = 1$

- Σε κάθε εσωτερικό κόμβο i συνδέουμε του δέντρο $\mathcal{R}_{d-1(U_i)}$

- Τα κλειδιά αποθηκεύοντα στα φύλλα

Αρα το τελικό κόστος κατασκευής θα προκύψει απο την αναδρομή

$$T(n) = 2T(n/2) + 2O(n) = 2T(n/2) + O(n)$$

το απο master theorem δίνει $T(n) = O(n \log n)$
