Randomized Binary Search Trees

CONRADO MARTÍNEZ AND SALVADOR ROURA

UNIVERSITAT POLITÈCNICA DE CATALUNYA, BARCELONA, CATALONIA, SPAIN

ΒΑΣΙΛΙΚΗ-ΜΑΡΙΑ ΚΑΤΣΙΟΥΛΗ
ΔΕΚΕΜΒΡΙΟΣ 2025

Βασικές Συνεισφορές Έρευνας

Παρουσιάζονται τυχαιοποιημένοι αλγόριθμοι για Binary Search Trees (BST) τέτοιοι ώστε:

- Η εισαγωγή οποιουδήποτε συνόλου κλειδιών, με οποιαδήποτε σειρά, σε ένα αρχικά άδειο Δέντρο ΠΑΝΤΑ να παράγει τυχαιοποιημένο BST (RBST)
- Η διαγραφή οποιουδήποτε κλειδιού από ένα RBST ΠΑΝΤΑ να παράγει RBST
- Οι τυχαίες επιλογές των αλγορίθμων να εξαρτώνται από το μέγεθος των subtrees του δέντρου - άρα να υπάρχει πρόσβαση κατά rank, χωρίς απαιτήσεις περισσότερου χώρου ή μετατροπές στις δομές των δεδομένων, και
- Το αναμενόμενο κόστος κάθε βασικής λειτουργίας, ως το πλήθος των επισκεπτόμενων nodes, να είναι O(logn), αλλά πλέον να είναι ανεξάρτητο από την κατανομή της εισόδου

Εισαγωγικές Πληροφορίες

Βασικές Λειτουργίες σε ένα BST:

- Αναζήτηση ενός στοιχείου και ανάκτηση των πληροφοριών του, εφόσον υπάρχει,
- Εισαγωγή στοιχείων,
- Διαγραφή στοιχείων

Οι πρώτες δύο γίνονται εύκολα κι απλά, με γραμμικό κόστος εξαρτώμενο από το ύψος του BST.

Για τα RBST υπάρχει εκτιμώμενο κόστος για αναζήτηση (επιτυχή ή όχι) και εισαγωγή O(logn), ήδη από προηγούμενες ερευνητικές εργασίες.

Αν το σύνολο εισόδου ΔΕΝ είναι τυχαίο, όμως, το κόστος μπορεί να υποβαθμιστεί σε γραμμικό.

Δεν υπάρχει, έως τότε, αλγόριθμος για διαγραφή στοιχείων που να διατηρεί την τυχαιότητα στα RBST!

Εισαγωγικές Πληροφορίες

Προσπάθεια υπέρβασης των ανωτέρω προβλημάτων με:

- την δημιουργία διαφόρων ειδών Ισορροπημένων Δέντρων (Balanced Search Trees),

  → Έχουν ως μειονεκτήματα τους αρκετά περίπλοκους αλγορίθμους εισαγωγής, τους σταθερούς παράγοντες που μπορούν να καταλήξουν μεγάλοι και την παραπάνω πληροφορία που πρέπει να φέρουν τα nodes ώστε να διατηρείται η ισορροπία.

- την χρήση τυχαιοποιημένων τεχνικών,

  → Έχουν ως πλεονεκτήματα την απλότητά τους, την εγγυημένη απόδοσή τους η οποία είναι ανεξάρτητη της κατανομής της εισόδου και το γεγονός ότι το worst-case scenario δεν μπορεί να κατασκευαστεί αν δεν είναι γνωστές οι τυχαίες επιλογές του αλγορίθμου.

Εισαγωγικές Πληροφορίες

Τυχαιοποιημένες τεχνικές έχουν ήδη χρησιμοποιηθεί για:

- Randomized Treaps και
- Skip lists

κι έχουν επιτύχει λογαριθμική αναμενόμενη απόδοση.

Στην συγκεκριμένη έρευνα τα BST που παράγονται μέσω των τυχαιοποιημένων αλγορίθμων ονομάζονται RBST και παραμένουν τέτοια ανεξαρτήτως της σειράς εισαγωγής ή διαγραφής των στοιχείων.

Επομένως, έχουν αναμενόμενη απόδοση O(logn), δεδομένου ότι οι τυχαίες επιλογές του αλγορίθμου παραμένουν άγνωστες σε κάποιον adversary που θέλει να πετύχει το worst-case scenario.

Εισαγωγικές Πληροφορίες

Κύρια διαφορά μεταξύ RBST και Randomized Treaps:

- Τα RBST χρησιμοποιούν «Δομικές πληροφορίες», αποθηκευμένες σε κάθε node, για να γίνουν οι τυχαίες επιλογές, βασιζόμενες στο n (μέγεθος Δέντρου), οπότε δεν χρειάζεται επιπλέον αποθηκευτικός χώρος ή μετατροπή των δομών δεδομένων.
- Τα Randomized Treaps χρησιμοποιούν έναν τυχαίο αριθμό στο πεδίο [0,1] ως τυχαία προτεραιότητα του node, άρα «μη-Δομικές πληροφορίες».

Αλγόριθμοι Εισαγωγής - Insertions

Υποθέτουμε ότι κάθε στοιχείο του Δέντρου αποτελείται από ένα key (χωρίς περαιτέρω πληροφορίες που να συνδέονται μ' αυτό) κι όλα τα keys είναι μη αρνητικοί ακέραιοι αριθμοί. Επίσης το άδειο δέντρο ή το external node συμβολίζεται ως □.

ΟΡΙΣΜΟΣ 2.1: Έστω T ένα BST μεγέθους n.

- Αν n=0, τότε T= □ και είναι RBST,
- Αν n>0, το T είναι RBST αν και μόνο αν και το αριστερό subtree L και το δεξί subtree R είναι ανεξάρτητα RBST, και

$$\Pr\{\text{size}(L) = i | \text{size}(T) = n\} = \frac{1}{n}, \qquad i = 0, \ldots, n-1, \qquad n > 0. \qquad (1)$$

Αλγόριθμοι Εισαγωγής - Insertions

Καίρια ιδιότητα των RBST, ως άμεση συνέπεια της (1), είναι ότι οποιοδήποτε key ενός Δέντρου μεγέθους $n>0$, έχει την ίδια πιθανότητα $1/n$ να αποτελεί ρίζα του.

Με άλλα λόγια, για να παράγουμε RBST, κάθε key που εισάγεται σε ένα RBST πρέπει να έχει κάποια πιθανότητα να γίνει ρίζα όλου του Δέντρου ή κάποιου subtree κ.ο.κ.

Ο παρακάτω αλγόριθμος είναι γραμμένος σε C-like γλώσσα, περιγράφει την διαδικασία εισαγωγής ενός νέου στοιχείου x σε ένα δέντρο μεγέθους n και ισχύει για κάθε $n\geq 0$.

Αλγόριθμοι Εισαγωγής - Insertions

Algorithm 1 Insertion

```
bst insert(int x, bst T) {
  int n, r;

  n = T→size;
  r = random(0,n);
  if (r == n)
      return insert_at_root(x,T);
  if (x < T→key)
      T→left = insert(x, T→left);
  else
      T→right = insert(x, T→right);
  return T;
}
```

Σημείωση: Στον Αλγόριθμο 1, για λόγους απλότητας, δεν φαίνεται η αλλαγή του μεγέθους του δέντρου από n σε n+1. Θα αναφερθεί παρακάτω.

Αλγόριθμοι Εισαγωγής - Insertions

Το ***insert_at_root(x, T)*** βασίζεται στην τεχνική split(x, T) και σε αλγόριθμο που υπήρχε ήδη (Stephenson 1980):

1. Το Δέντρο διαιρείται σε δύο Δέντρα $T_<$ και $T_>$,
2. Το $T_<$ περιέχει τα στοιχεία που είναι μικρότερα του x,
3. Το $T_>$ περιέχει τα στοιχεία που είναι μεγαλύτερα του x,
4. Το $T_<$ γίνεται το αριστερό subtree, ενώ το $T_>$ το δεξί

Αν η ρίζα του Δέντρου είναι μεγαλύτερη από το x, αυτή μαζί με το $T_>$ αποτελούν μέρος του Δεξιού subtree και για να δούμε ποια στοιχεία του $T_<$ παραμένουν σε αυτό και ποια μετακινούνται στο Δεξιό subtree καλούμε αναδρομικά το split(x, T→left).

Αλγόριθμοι Εισαγωγής - Insertions

Η split(x, T), ουσιαστικά, συγκρίνει τα ίδια στοιχεία του T με το x που θα συγκρίναμε σε μία ανεπιτυχή αναζήτηση του στοιχείου στο Δέντρο. Επομένως, το κόστος εισαγωγής στην ρίζα είναι ανάλογο του κόστους της εισαγωγής του ίδιου στοιχείου, στο ίδιο δέντρο με την χρήση των ήδη υπαρχόντων αλγορίθμων.

Αλγόριθμοι Εισαγωγής - Insertions

Algorithm 2 Insertion at the root

```
bst insert_at_root(int x, bst T) {
    bst S, G;

    split(x, T, &S, &G); /* S ≡ T<, G ≡ T> */
    T = new_node();
    T→key = x; T→left = S; T→right = G;
    return T;
}

void split(int x, bst T, bst *S, bst *G) {

    if (T == □) {
        *S = *G = □;
        return;
    }
    if (x < T→key) {
        *G = T;
        split(x, T→left, S, &(*G→left));
    }
    else { /* x > T→key */
        *S = T;
        split(x, T→right, &(*S→right), G);
    }
    return;
}
```

Αλγόριθμοι Εισαγωγής - Insertions

Ο αλγόριθμος της τυχαίας εισαγωγής ΠΑΝΤΑ διατηρεί την τυχαιότητα, ανεξαρτήτως του x που εισάγουμε.

Λήμμα 2.2: Έστω $T_<$ και $T_>$ είναι τα BSTs που προκύπτουν από το split(x, T). Αν το T είναι RBST που περιέχει το σύνολο των στοιχείων K, τότε τα $T_<$ και $T_>$ είναι ανεξάρτητα RBSTs που περιέχουν τα σύνολα στοιχείων $K_{<x} = \{y \in T | y<x\}$ και $K_{>x} = \{y \in T | y>x\}$, αντίστοιχα.

Αλγόριθμοι Εισαγωγής - Insertions

ΑΠΟΔΕΙΞΗ: Η απόδειξη χρησιμοποιεί επαγωγή στο μέγεθος n.

Αν n=0, τότε T=□ κι άρα η split(x, T) δίνει $T_< = T_> = □$, άρα το Λήμμα ισχύει.

Αν n>0 έστω y= T→key, L= T→left, R= T→right. Αν x>y:

Η ρίζα του $T_<$ είναι το y και το αριστερό subtree είναι το L.

Το $T_>$ και το subtree του $T_<$ (έστω R΄) υπολογίζονται αναδρομικά από την split που καλούμε στο R.

Μέσω της επαγωγικής υπόθεσης αυτή η διαδικασία split θα παράξει 2 δέντρα που είναι RBSTs, καθώς όλα τα υποδέντρα είναι ανεξάρτητα μεταξύ τους.

Αλγόριθμοι Εισαγωγής - Insertions

Επίσης για κάθε z ∈ $T_<$, η πιθανότητα να αποτελεί την ρίζα του $T_<$ είναι 1/m, όπου m είναι το μέγεθος του $T_<$.

$$\mathbb{P}[z \text{ is root of } T_< | \text{root of } T \text{ is } < x] = \frac{\mathbb{P}[z \text{ is root of } T \text{ and root of } T \text{ is } < x]}{\mathbb{P}[\text{root of } T \text{ is } < x]}$$

$$= \frac{1/n}{m/n} = \frac{1}{m}.$$

Η ίδια λογική ισχύει και για x<y.

Αλγόριθμοι Εισαγωγής - Insertions

Θεώρημα 2.3: Αν T: RBST που περιέχει σύνολο στοιχείων K και x∉K, τότε η insert(x, T) παράγει ένα RBST που περιέχει το σύνολο στοιχείων K∪{x}.

Απόδειξη: Αν n=0, τότε T=□ και η insert(x, T) παράγει ένα RBST με κενά υποδέντρα και το x ως ρίζα.

Αν n>0, θεωρούμε ότι το θεώρημα είναι αληθές για κάθε μέγεθος <n. Έστω στοιχείο y ∈ K.

Πιθανότητα να είναι ρίζα (πριν το x): 1/n άρα το T είναι RBST.

Πιθανότητα να μείνει ρίζα: πρέπει το x να εισαχθεί σε κάποιο υποδέντρο (πιθανότητα n/n+1) άρα η πιθανότητα το y να είναι ρίζα του νέου δέντρου T΄ είναι 1/n*n/n+1=1/n+1. Τα subtrees που δημιουργούνται είναι RBSTs.

Αν x ρίζα T΄: πιθανότητα 1/n+1 και παράγονται δύο RBST υποδέντρα (από το Λήμμα).

Αλγόριθμοι Εισαγωγής - Insertions

Πόρισμα 2.4: Έστω K=$\{x_1, x_2, \ldots, x_n\}$ οποιοδήποτε σύνολο στοιχείων

όπου n≥0. Έστω p= $x_{i_1}, \ldots, x_{i_n}$ οποιαδήποτε σταθερή σειρά εισαγωγής των στοιχείων του K. Τότε το RBST που παίρνουμε από την εισαγωγή των στοιχείων του p σε ένα αρχικά άδειο δέντρο είναι ένα RBST. Αλλιώς, αν

$$T = insert(x_{i_n}, insert(x_{i_{n-1}}, \ldots, insert(x_{i_1}, \square)\ldots)),$$

τότε T είναι RBST για το σύνολο των στοιχείων K.

Αντιθέτως, στα κλασικά BST, η τυχαιότητα επιτυγχάνεται μόνο αν η σειρά εισαγωγής είναι μια τυχαία μετάθεση.

Αλγόριθμοι Εισαγωγής - Insertions

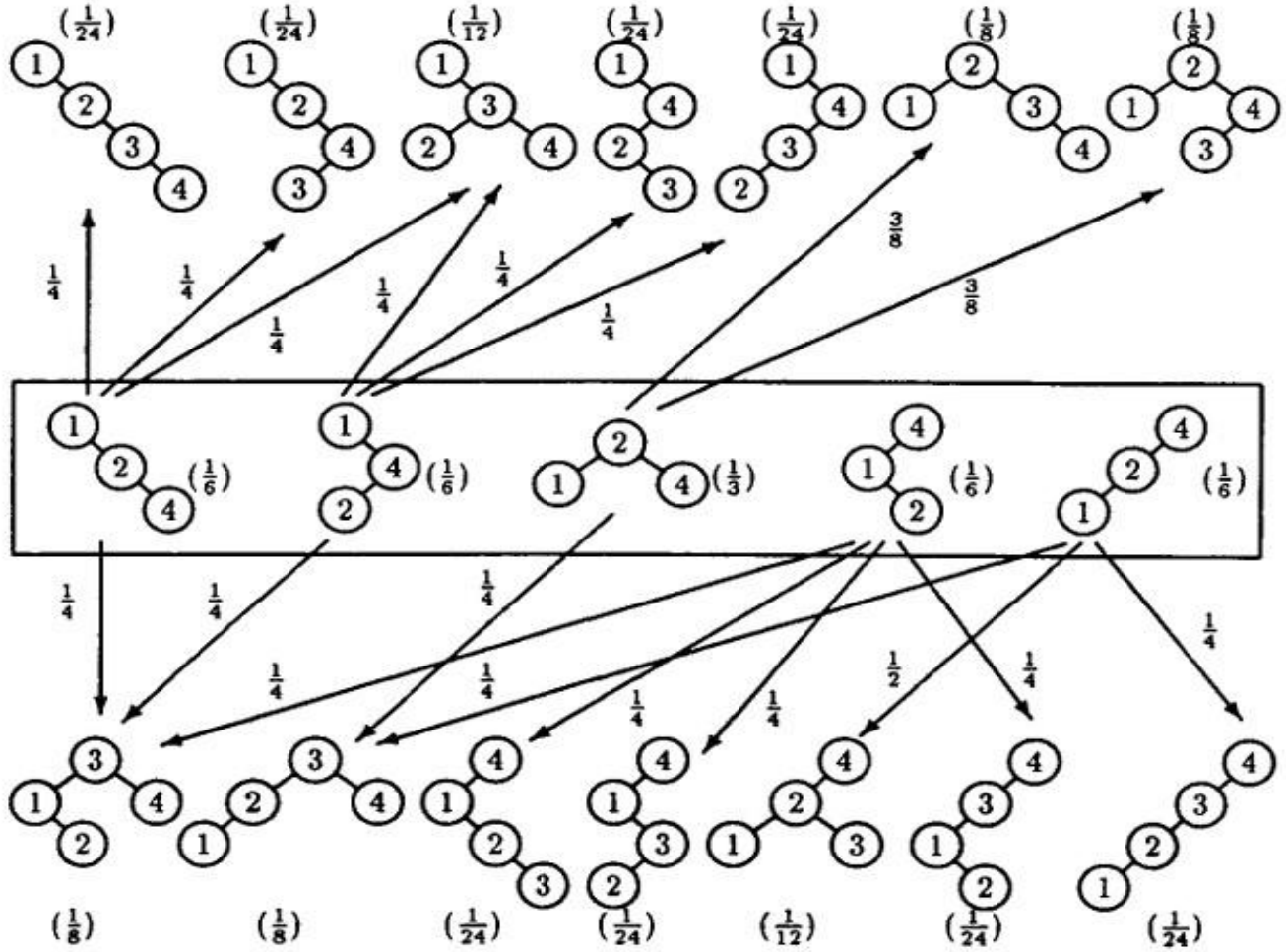


FIG. 1. Insertion of $x = 3$ in a random BST for the set $K = \{1, 2, 4\}$.

Αλγόριθμοι Διαγραφής - Deletions

Παρακάτω παρουσιάζεται ο αλγόριθμος διαγραφής ενός στοιχείου x:

Algorithm 3 Deletion

```
bst delete(int x, bst T) {
  bst Aux;

  if (T == □)
      return □;
  if (x < T→key)
      T→left = delete(x, T→left);
  else if (x > T→key)
      T→right = delete(x, T→right);
  else { /* x == T→key */
      Aux = join(T→left, T→right);
      free_node(T);
      T = Aux;
  }
  return T;
}
```

Αλγόριθμοι Διαγραφής - Deletions

Η διαγραφή ενός στοιχείου x, ξεκινάει με την αναζήτηση αυτού κι εφόσον βρεθεί, τα υποδέντρα που έχουν ως ρίζα το x ενώνονται μέσω της join(L, R) δημιουργώντας νέο δέντρο T΄.

Έστω τα L, R έχουν μέγεθος m>0 και n >0 αντίστοιχα. Αντί να πάρουμε το μεγαλύτερο στοιχείο του L ή το μικρότερο του R και να το κάνουμε τη νέα ρίζα του T΄, ο αλγόριθμος κάνει την ρίζα ενός subtree (a=L →key ή b=R→key), ρίζα του T΄ κι αναλόγως καλεί την join στα νέα υποδέντρα.

Ισχύει L_l = L→left, L_r = L→right, R_l = R→left, R_r = R→right και αν γίνει το a=T΄→key , L_l = T΄→left και join(L_r, R)= T΄→right. Αντίστοιχα και για το b.

Οι πιθανότητα καθένα από αυτά να είναι ρίζα του T΄ είναι m/m+n για το a και n/m+n για το b.

Αυτός ο αλγόριθμος διατηρεί την τυχαιότητα λόγω της λειτουργίας της join.

Αλγόριθμοι Διαγραφής - Deletions

Algorithm 4 Join of two RBSTs

```
bst join(bst L, bst R) {
  int m, n, r, total;

  m = L→size; n = R→size; total = m + n;
  if (total == 0) return □;
  r = random(0, total - 1);
  if (r < m) {                    /* with probability m/(m+n) */
      L→right = join(L→right, R);
      return L;
  }
  else {                          /* with probability n/(m+n) */
      R→left = join(L, R→left);
      return R;
  }
}
```

Αλγόριθμοι Διαγραφής - Deletions

Λήμμα 3.1: Έστω L και R είναι δύο ανεξάρτητα RBSTs, τέτοια ώστε τα στοιχεία του L να είναι αυστηρά μικρότερα από αυτά του R. Έστω K_L και K_R τα σύνολα των L, R αντιστοίχως. Τότε T=join(L, R) είναι ένα RBST που περιέχει τα στοιχεία K= $K_L \cup K_R$.

Απόδειξη: Αποδεικνύεται με επαγωγή στα μεγέθη m, n των L, R.

Αν m=0 ή n=0 τότε το Λήμμα ισχύει και επιστρέφει το μη μηδενικό subtree, αν υπάρχει, ή το □ αν και τα δύο είναι μηδενικά.

Αν m>0 και n>0 και a, b όπως τα έχουμε ορίσει παραπάνω:

Είτε a είτε b γίνονται η ρίζα όλου του δέντρου και καλούμε την αντίστοιχη join, όπως έχουμε παρουσιάσει. Τα subtrees που παράγονται στο νέο δέντρο είναι RBSTs καθώς διατηρούν την ιδιότητα τους αυτή από πριν και ταυτόχρονα παραμένουν ανεξάρτητα μεταξύ τους.

Η πιθανότητα κάποιου x∈L να είναι ρίζα του 1/m*m/m+n=1/m+n. Το αντίστοιχο και για κάποιο x∈R.

Αλγόριθμοι Διαγραφής - Deletions

Θεώρημα 3.2: Αν T είναι RBST που περιέχει τα στοιχεία K, τότε η delete(x,T) παράγει ένα RBST που περιέχει τα στοιχεία K-{x}.

Απόδειξη: Αν $x \notin T$, τότε η delete δεν αλλάζει το T και το θεώρημα ισχύει.

Αν $x \in T$, η απόδειξη πάλι χρησιμοποιεί επαγωγή στο μέγεθος n του T. Αν n=1, τότε η delete(x, T) παράγει ένα άδειο δέντρο άρα το θεώρημα ισχύει.

Αν n>1, θεωρούμε ότι το θεώρημα ισχύει για όλα τα μεγέθη <n και:

- Αν το x δεν είναι η ρίζα του T, τότε το διαγράφουμε από κάποιο subtree που λόγω επαγωγής είναι RBST.
- Αν το x είναι η ρίζα του T, T΄=delete(x, T) προέρχεται από το join των υποδέντρων και, λόγω του Λήμματος, είναι RBST.

Αλγόριθμοι Διαγραφής - Deletions

Επίσης για κάθε y ∈K-{x} ισχύει ότι η πιθανότητα να είναι ρίζα του T΄:

$$P[y= T' \rightarrow key]=$$

$$P[y= T' \rightarrow key \mid x= T \rightarrow key] * P[x= T \rightarrow key]+$$

$$P[y= T' \rightarrow key \mid x \neq T \rightarrow key] * P[x \neq T \rightarrow key]=$$

$$\frac{1}{n-1} * \frac{1}{n} + \frac{1}{n-1} * \frac{n-1}{n} =$$

$$\frac{1}{n-1}$$

Αλγόριθμοι Διαγραφής - Deletions

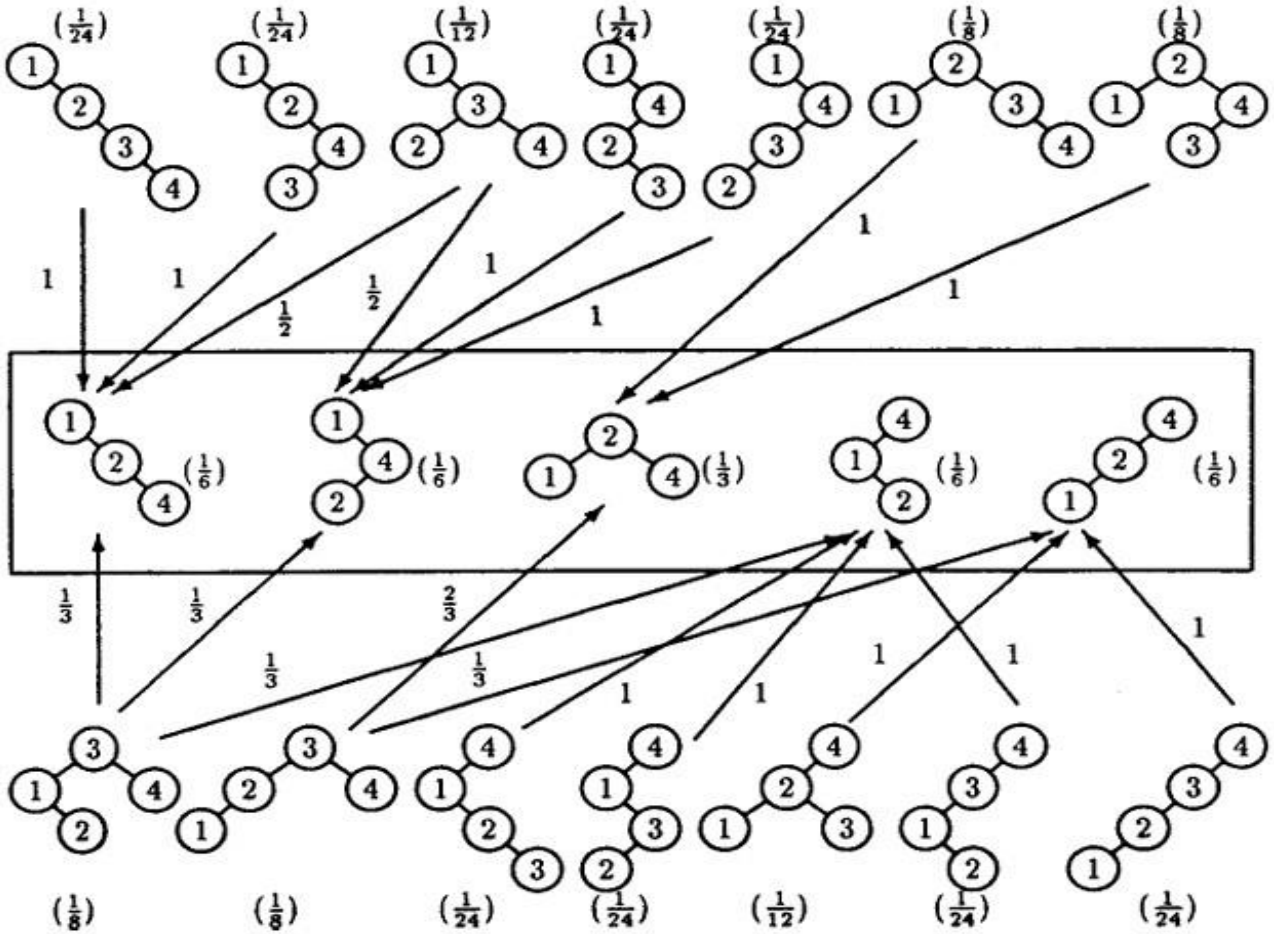


FIG. 2. Deletion of $x = 3$ from a random BST for the set $K = \{1, 2, 3, 4\}$.

Αλγόριθμοι Διαγραφής - Deletions

Συγκρίνοντας τα δύο Figures (Εισαγωγή και Διαγραφή) παρατηρούμε ότι για οποιοδήποτε σταθερό BST, με P[T] την πιθανότητα του T βάση του μοντέλου RBST, και T_1, T_2 οποιαδήποτε δοσμένα BSTs μεγέθους n και n+1, αντίστοιχα, ισχύει:

$$P[T_1]*P[\text{Εισάγοντας το x στο } T_1 \text{ δίνει } T_2]=$$

$$P[T_2]*P[\text{Διαγράφοντας το x στο } T_2 \text{ δίνει } T_1]$$

Επίσης, συνδυάζοντας τα Θεωρήματα 2.3 και 3.2 έχουμε:

Πόρισμα 3.3: Το αποτέλεσμα οποιασδήποτε αυθαίρετης σειράς εισαγωγών και διαγραφών, ξεκινώντας από ένα αρχικά άδειο Δέντρο, παράγει ΠΑΝΤΑ RBST. Επίσης, αν οι εισαγωγές είναι τυχαίες (όχι αυθαίρετες), τότε το αποτέλεσμα είναι πάλι RBST ακόμα κι αν χρησιμοποιούνται οι ήδη υπάρχοντες αλγόριθμοι εισαγωγής κι όχι οι τυχαίοι αλγόριθμοι εισαγωγής σε RBSTs.

Ανάλυση Απόδοσης

Η ανάλυση της απόδοσης είναι άμεση λόγω της ιδιότητας της τυχαιότητας.

Χρησιμοποιούμε τα ήδη γνωστά αποτελέσματα των RBSTs:

Αναμενόμενο βάθος του i-οστού internal node:

$$\mathbb{E}[\mathcal{D}_n^{(i)}] = H_i + H_{n+1-i} - 2, \qquad i = 1, \ldots, n;$$

Αναμενόμενο βάθος του i-οστού external node(leaf):

$$\mathbb{E}[\mathcal{L}_n^{(i)}] = H_{i-1} + H_{n+1-i}, \qquad i = 1, \ldots, n+1;$$

Συνολικό αναμενόμενο μήκος των spines των subtrees που έχουν ως ρίζα το i-οστό node:

$$\mathbb{E}[\mathcal{S}_n^{(i)}] = \mathbb{E}[\mathcal{L}_n^{(i)} + \mathcal{L}_n^{(i+1)} - 2(\mathcal{D}_n^{(i)} + 1)] = 2 - \frac{1}{i} - \frac{1}{n+1-i},$$

$$i = 1, \ldots, n;$$

όπου $H_n = \sum_{i \le j \le n} 1/j = \ln n + \gamma + O(1/n)$ και $\gamma = 0.577\ldots$ σταθερά Euler

Ανάλυση Απόδοσης

Έστω $S_n^{(i)}$ και $U_n^{(i)}$ το σύνολο των συγκρίσεων σε επιτυχή κι ανεπιτυχή αναζήτηση του i-οστου στοιχείου σε ένα δέντρο μεγέθους n. Ισχύει:

$$S_n^{(i)} = \mathcal{D}_n^{(i)} + 1, \qquad i = 1, \ldots, n.$$

$$U_n^{(i)} = \mathcal{L}_n^{(i)}, \qquad i = 1, \ldots, n+1.$$

Ανάλυση Απόδοσης

Για την εισαγωγή ενός i-οστού στοιχείου σε μέγεθος δέντρου n, η αναμενόμενη τιμή των επισκεπτόμενων nodes είναι $\mathbb{E}[\mathcal{L}_n^{(i)}] + 1$ καθώς επισκεπτόμαστε όλα τα nodes μιας μη επιτυχούς αναζήτησης συν ένα παραπάνω που αντιστοιχεί στο νέο node.

Όμως, το κόστος θα το χωρίσουμε σε δύο μέρη: το κόστος της καθόδου του στοιχείου μέχρι την θέση όπου θα εισαχθεί ($R_n^{(i)}$) και το κόστος της ανακατασκευής των υποδέντρων που θα το έχουν πλέον ρίζα ($I_n^{(i)}$) .

Τα μετράμε ως τα επισκεπτόμενα nodes κάθε φορά. Τα nodes από την ρίζα έως την θέση εισαγωγής είναι $R_n^{(i)}$ και τα nodes που επισκεπτόμαστε για να γίνει η εισαγωγή του ως ρίζα, είναι τα spines των subtrees του. Αυτή η εισαγωγή είναι επίσης τυχαία οπότε ισχύει:

$$R_n^{(i)} = \mathcal{D}_{n+1}^{(i)} + 1,\ I_n^{(i)} = \mathcal{S}_{n+1}^{(i)}, \qquad i = 1, \ldots, n+1$$

Ανάλυση Απόδοσης

Και η ακριβής εκτίμηση του αναμενόμενου κόστους είναι

$$\alpha\mathbb{E}[R_n^{(i)}] + \beta\mathbb{E}[I_n^{(i)}] = \alpha(H_i + H_{n+2-i} - 1) + \beta\left(2 - \frac{1}{i} - \frac{1}{n+2-i}\right)$$

με a, b σταθερές που δίνουν το αναμενόμενο κόστος κάθε μέρους της διαδικασίας εισαγωγής.

Ανάλυση Απόδοσης

Για την διαγραφή ενός i-οστού στοιχείου σε μέγεθος δέντρου n, το κόστος θα το χωρίσουμε, πάλι, σε δύο μέρη: το κόστος της αναζήτησης του στοιχείου ($F_n^{(i)}$) και το κόστος της ανακατασκευής (join) των υποδέντρων που θα το είχαν ρίζα ($J_n^{(i)}$) . Πάλι το εισαγόμενο δέντρο είναι τυχαίο οπότε ισχύει:

$$F_n^{(i)} = \mathcal{D}_n^{(i)} + 1, J_n^{(i)} = \mathcal{S}_n^{(i)}, \qquad i = 1, \ldots, n$$

Και με τον ίδιο τρόπο, όπως για την εισαγωγή, έχουμε σταθερές a΄ και b΄ για τα κόστη των φάσεων της διαγραφής, οπότε:

$$\alpha'(H_i + H_{n+1-i} - 1) + \beta'\left(2 - \frac{1}{i} - \frac{1}{n+1-i}\right)$$

Ανάλυση Απόδοσης

Συνολικά, λοιπόν, το κόστος κάθε αναζήτησης ή κάθε αλλαγής στα στοιχεία ενός RBST είναι $\Theta(\log n)$!

Άλλες Λειτουργίες

Άλλες λειτουργίες στις οποίες επεκτείνεται η εργασία είναι:

- Διπλότυπα στοιχεία
- Λειτουργίες συνόλων
- Αυτό-ρυθμιζόμενες τακτικές

Άλλες Λειτουργίες

Διπλότυπα στοιχεία:

- Σε περίπτωση που το εισαγόμενο στοιχείο, παίρνει την θέση της ρίζας ενός δέντρου (ή υποδέντρου) τότε αλλάζουμε την split ώστε να διαγράψει το διπλότυπο στοιχείο που υπάρχει πιο κάτω
- Σε περίπτωση που βρούμε το διπλότυπο στοιχείο στο πρώτο μέρος της διαδικασίας, ψάχνοντας θέση για το στοιχείο, αποφασίζουμε αν θα το αφήσουμε εκεί ή θα το «σπρώξουμε» παρακάτω. Αν δεν έχουμε αυτή την επιλογή κάθε στοιχείο που εισάγεται συχνά, θα φτάνει όλο και πιο κοντά στην ρίζα αλλάζοντας τις πιθανότητες του να γίνει ρίζα όλου του δέντρου (όχι RBST). Μ’ αυτό θα ασχοληθούμε και στις αυτό-ρυθμιζόμενες τακτικές παρακάτω.

Άλλες Λειτουργίες

Algorithm 5 Push down

```
bst push_down(bst T) {
/* T ≠ □ */
  bst P;
  int m, n, r, total;

  m = T→left→size; n = T→right→size;
  total = m + n;
  r = random(0, total);
  if (r < m) {                /* with probability m/(m+n+1) */
      P = T→left;
      T→left = T→left→right;
      P→right = push_down(T);
      return P;
  }
  else if (r < total) {       /* with probability n/(m+n+1) */
      P = T→right;
      T→right = T→right→left;
      P→left = push_down(T);
      return P;
  }
  return T;                   /* with probability 1/(m+n+1) */
}
```

Άλλες Λειτουργίες

Ο παραπάνω αλγόριθμος ικανοποιεί το παρακάτω Θεώρημα:

Θεώρημα 5.1.1: Έστω T ένα BST τέτοιο ώστε η ρίζα του είναι κάποιο γνωστό x, και τα υποδέντρα του είναι RBSTs. Τότε η push_down(T) παράγει ένα εντελώς τυχαίο BST (χωρίς πληροφορίες για την ρίζα του δέντρου).

Επίσης ισχύει η γενίκευση του Θεωρήματος 2.3:

Θεώρημα 5.1.2: Αν T ένα RBST που περιέχει τα στοιχεία K και το x είναι ένα οποιοδήποτε στοιχείο (είτε ανήκει είτε όχι στο K), τότε η insert(x, T) δημιουργεί ένα RBST με στοιχεία K∪{x}.

Άλλες Λειτουργίες

Λειτουργίες Συνόλων:

Για τις λειτουργίες της Ένωσης, Τομής και Διαφοράς έχουμε αντίστοιχους αλγορίθμους οι οποίοι εγγυόνται την δημιουργία RBST από την ένωση, τομή ή διαφορά RBSTs. Και για δύο RBSTs μεγέθους m και n η διαδικασία των παραπάνω έχει αναμενόμενη απόδοση $\Theta(m+n)$.

Άλλες Λειτουργίες

Algorithm 6 Union

```
bst union(bst A, bst B) {
  bst Al, Ar, Bl, Br;
  int m, n, u, total, rep;

  m = A→size; n = B→size; total = m + n;
  if (total == 0) return □;
  u = random(1, total);
  if (u <= m) {              /* with probability m/(m+n) */
      split(A→key, B, &Bl, &Br);
      A→left = union(A→left, Bl);
      A→right = union(a→right, Br);
      return A;
  }
  else {                     /* with probability n/(m+n) */
      rep = split(B→key, A, &Al, &Ar);
      B→left = union(B→left, Al);
      B→right = union(B→right, Ar);
      if (rep) return push_down(B);
       else    return B;
  }
}
```

Άλλες Λειτουργίες

Algorithm 7 Intersection

```
bst intersection(bst A, bst B) {
  bst Bl, Br, il, ir;
  int rep;

  if (A == □) {
      free_tree(B);
      return □;
  }
  rep = split(A→key, B, &Bl, &Br);
  il = intersection(A→left, Bl);
  ir = intersection(A→right, Br);
  if (rep) {
      A→left = il;
      A→right = ir;
      return A;
  }
  else {
      free_node(A);
      return join(il, ir);
  }
}
```

Άλλες Λειτουργίες

Algorithm 8 Difference

```
bst difference(bst A, bst B) {
  bst Bl, Br, dl, dr;
  int rep;

  if (a == □) {
      free_tree(B);
      return □;
  }
  rep = split(A→key, B, &Bl, &Br);
  dl = difference(A→left, Bl);
  dr = difference(A→right, Br);
  if (rep) {
      free_node(A);
      return join(dl, dr);
  }
  else {
      A→left = dl;
      A→right = dr;
      return A;
  }
}
```

Άλλες Λειτουργίες

Αυτό-ρυθμιζόμενες τακτικές:

- Τα RBSTs χρησιμοποιούν τις λειτουργίες split και join (ή ανάλογους μηχανισμούς) για την αναδιάρθρωση του δέντρου μετά από κάθε πράξη, με τρόπο που διατηρεί την ιδιότητα της τυχαιότητας.

- Η τυχαιοποίηση αυτή επιτρέπει την επίτευξη υψηλής απόδοσης χωρίς την ανάγκη για τους πολύπλοκους αλγορίθμους ενημέρωσης και τις μεγάλες σταθερές στα παραδοσιακά ισορροπημένα δέντρα.
- Ακόμα κι εκεί το αναμενόμενο κόστος είναι O(logn) για όλες τις βασικές λειτουργίες (αναζήτηση, εισαγωγή, διαγραφή) και η απόδοση είναι εγγυημένη ανεξάρτητα από την κατανομή της εισόδου, καθώς η τυχαιοποίηση είναι εσωτερική στον αλγόριθμο.

Θέματα Υλοποίησης

Αριθμός τυχαίων bits: Η πολυπλοκότητα ως προς το μέγεθος των τυχαίων bits είναι Θ(logn) καθώς κάθε φορά για να εισάγουμε ένα στοιχείο ως ρίζα ενός (υπο)δέντρου παράγουμε ένα τυχαίο αριθμό και το εκτιμώμενο πλήθος nodes που επισκεπτόμαστε για την διαδικασία αυτή είναι Θ(logn). Με κάποιες βελτιώσεις ίσως θα μπορούσε να γίνει Θ(1) αλλά είναι ένα κόστος που αξίζει έναντι πιο περίπλοκων αλγορίθμων. Για τις διαγραφές το κόστος αυτό είναι σταθερό.

Χώρος και πολυπλοκότητα: σε κάθε node αποθηκεύεται το μέγεθος του δεξιού ή του αριστερού υποδέντρου κάνοντας τον υπολογισμό πιο εύκολο και το αναμενόμενο κόστος αυτής της αποθήκευσης υπολογίζεται σε Θ(n), το οποίο είναι κατά πολύ μικρότερο από το Θ(nlogn) που χρησιμοποιείται για τα στοιχεία και τους pointers.

Τυπικό πλαίσιο και Αποδείξεις

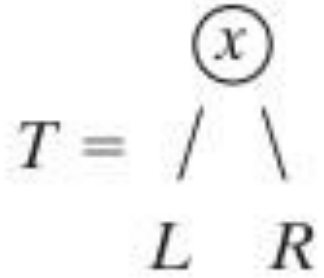
Η Αλγεβρική προσέγγιση των τυχαιοποιημένων αλγορίθμων βασίζεται στην κεντρική ιδέα πως:

Κάθε τυχαιοποιημένος αλγόριθμος F θεωρείται συνάρτηση από ένα σύνολο εισόδων A σε ένα σύνολο συναρτήσεων πιθανότητας (PF) επί του συνόλου των εξόδων B. Λέμε ότι η f είναι συνάρτηση πιθανότητας στο B αν και μόνο αν: f: $B \rightarrow [0,1]$ και $\sum_{y \in B} f(y) = 1$.

Τυπικό πλαίσιο και Αποδείξεις

Παρακάτω θα δώσουμε ορισμούς και σημειογραφίες για τα κυριότερα αντικείμενα με τα οποία ασχολούμαστε: στοιχεία, πιθανές διατάξεις σειρών στοιχείων (permutations) και δέντρων.

Ένα δέντρο T με ρίζα x και αριστερό υπόδεντρο το L και δεξί το R αναπαρίσταται ως:



Τυπικό πλαίσιο και Αποδείξεις

Συμβολίζουμε ως:

- B(K) = το σύνολο όλων των BSTs που περιέχουν τα στοιχεία του K,
- P(K) = το σύνολο όλων των σειρών (ή αλλιως permutations) (χωρίς επανάληψη) των στοιχείων του K
- λ = άδεια σειρά
- U|V = η αλληλουχία των στοιχείων U και V(χωρίς να έχουν κοινά στοιχεία)

Τυπικό πλαίσιο και Αποδείξεις

Έχοντας μια σειρά S συμβολίζουμε ως bst(S) το BST ως το αποτέλεσμα της εισαγωγής των στοιχείων του S από αριστερά προς τα δεξιά σε ένα αρχικά άδειο δέντρο:

$$\mathrm{bst}(\lambda) = \square, \qquad \mathrm{bst}(x|S) = \begin{array}{c} (x) \\ \diagup \quad \diagdown \\ \mathrm{bst}(\mathrm{sep}_{<}(x, S)) \quad \mathrm{bst}(\mathrm{sep}_{>}(x, S)) \end{array}$$

όπου το $\mathrm{sep}_{<}(x,S)$ επιστρέφει την υποσειρά των μικρότερων του x στοιχείων και αντίστοιχα το $\mathrm{sep}_{>}(x,S)$ τα μεγαλύτερα.

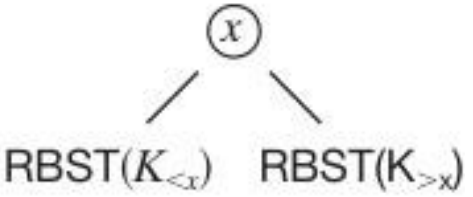
Τυπικό πλαίσιο και Αποδείξεις

Έστω Random_Perm μια συνάρτηση όπου για σύνολο K με στοιχεία n≥ 0 επιστρέφει μια τυχαία σειρά αυτών:

$$\text{Random\_Perm}(K) = \sum_{P \in \mathcal{P}(K)} \frac{1}{n!} \cdot P$$

Έτσι ο ορισμός των RBSTs γράφεται ως εξής:

$$\text{RBST}(K) = \text{bst}(\text{Random\_Perm}(K)) = \sum_{x \in K} \frac{1}{n}.$$



όπου προφανώς για n=0 RBST(K)= □

Τυπικό πλαίσιο και Αποδείξεις

- Εισάγουμε την συνάρτηση rm, η οποία διαγράφει ένα στοιχείο από μια σειρά χωρίς να αλλάζει την σειρά των υπολοίπων:

rm(2, 1423)=143, rm(5, 1423)=1423

- Εισάγουμε την συνάρτηση shuffle, η οποία δίνει ένα τυχαίο ανακάτεμα δύο σειρών, κρατώντας την σχετική επιμέρους σειρά των στοιχείων της κάθε μίας:

shuffle (21,ba)={21ba, 2b1a, 2ba1, b21a, b2a1, ba21}=

1/6*21ba + 1/6*2b1a + 1/6*2ba1 + 1/6*b21a + 1/6*b2a1 + 1/6*ba21

Αλλιώς γράφουμε: $\text{shuffle}(\lambda, \lambda) = \lambda$, $\text{shuffle}(\lambda, v|V) = v|V$, $\text{shuffle}(u|U, \lambda) = u|U$,

$$\text{shuffle}(u|U, v|V) = \frac{m}{m+n} \cdot u|\text{shuffle}(U, v|V) + \frac{n}{m+n} \cdot v|\text{shuffle}(u|U, V)$$

Με m=size→u|U και n=size →v|V

Τυπικό πλαίσιο και Αποδείξεις

Επίσης μπορούμε να γράψουμε για κάποιο x που δεν ανήκει στο K:

$$\text{Random\_Perm}(\emptyset) = \lambda,$$

$$\text{Random\_Perm}(K \cup \{x\}) = \text{shuffle}(x, \text{Random\_Perm}(K))$$

- Τέλος, εισάγουμε την συνάρτηση equiv η οποία, για μια σειρά S ∈ P(K), επιστρέφει ένα τυχαίο στοιχείο από το σύνολο των σειρών που παράγει το ίδιο BST με το S

E(S)={E∈ P(K)| bst(E)=bst(S)}

E(3124)={3124, 3142, 3412} και

equiv(3124)=1/3*3124+1/3*3142+1/3*3412 γιατί

bst(3124)=bst(3142)=bst(3412)

Τυπικό πλαίσιο και Αποδείξεις

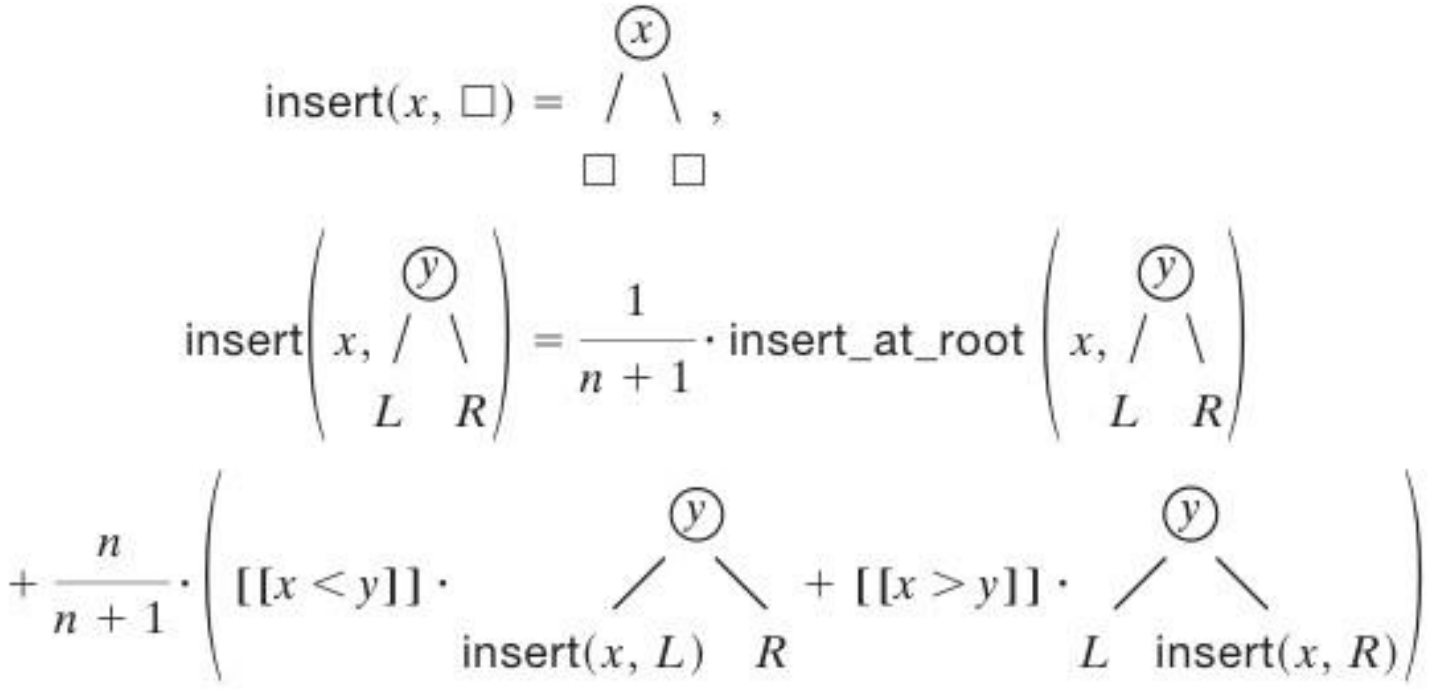
Για την equiv, λοιπόν, έχουμε:

$$\mathsf{equiv}(\lambda) = \lambda,$$

$$\mathsf{equiv}(x|S) = x|\mathsf{shuffle}(\mathsf{equiv}(\mathsf{sep}_{<}(x, S)), \mathsf{equiv}(\mathsf{sep}_{>}(x, S)))$$

Τυπικό πλαίσιο και Αποδείξεις

Όσο αφορά τους αλγόριθμους που χρησιμοποιήσαμε παραπάνω έχουμε:



Τυπικό πλαίσιο και Αποδείξεις

Για την split ισχύει

split(x, T)=[$split_<(x,T), split_>$(x, T)] και για $split_<$ έχουμε για x∉T:

$$\text{split}_<(x, \square) = \square,$$

$$\text{split}_<\left(x, \begin{array}{c} y \\ / \quad \backslash \\ L \quad R \end{array}\right) = [[x < y]] \cdot \text{split}_<(x, L) + [[x > y]] \cdot \begin{array}{c} y \\ / \quad \backslash \\ L \quad \text{split}_<(x, R) \end{array}$$

Οπότε παίρνουμε:

$$\text{insert\_at\_root}(x, T) = \begin{array}{c} x \\ / \quad \backslash \\ \text{split}_<(x, T) \quad \text{split}_>(x, T) \end{array}$$

Τυπικό πλαίσιο και Αποδείξεις

Για την delete ισχύει:

$$\text{delete}(x, \square) = \square,$$

$$\text{delete}\left(x, \begin{array}{c} (y) \\ / \quad \backslash \\ L \quad R \end{array}\right) = [[x < y]] \cdot \begin{array}{c} (y) \\ / \quad \backslash \\ \text{delete}(x, L) \quad R \end{array} + [[x, y]] \cdot \begin{array}{c} (y) \\ / \quad \backslash \\ L \quad \text{delete}(x, R) \end{array}$$

$$+ [[x = y]] \cdot \text{join}(L, R).$$

Και για την join: $\text{join}(\square, \square) = \square, \qquad \text{join}(L, \square) = L, \qquad \text{join}(\square, R) = R$

$$\text{join}(L, R) = \frac{m}{m+n} \cdot \begin{array}{c} (a) \\ / \quad \backslash \\ L_l \quad \text{join}(L_r, R) \end{array} + \frac{n}{m+n} \cdot \begin{array}{c} (b) \\ / \quad \backslash \\ \text{join}(L, R_l) \quad R_r \end{array}$$

Τυπικό πλαίσιο και Αποδείξεις

Κάνοντας χρήση όλων των παραπάνω έχουμε:

Λήμμα 7.4.1: Έστω S οποιοδήποτε permutation στοιχείων και x δεν ανήκει σε αυτά, τότε:

$$\text{split}(x, \text{bst}(S)) = [\text{bst}(sep_{<}(x, S)), bst(sep_{>}(x, S))]$$

Θεώρημα 7.4.2: Έστω K σύνολο στοιχείων και x δεν ανήκει στο K. Αν $K_{<x}$ και $K_{>x}$ περιέχουν τα στοιχεία του K μικρότερα και μεγαλύτερα του x αντίστοιχα, τότε:

$$\text{split}(x, \text{RBST}(K)) = [\text{RBST}(K_{<x}), \text{RBST}(K_{>x})]$$

Λήμμα 7.4.3: Έστω S οποιοδήποτε permutation στοιχείων και x δεν ανήκει σε αυτά, τότε:

$$\text{split}(x, \text{insert}(x, \text{bst}(S)) = \text{bst}(\text{shuffle}(x, \text{equiv}(S)))$$

Θεώρημα 7.4.4: Έστω K σύνολο στοιχείων και x δεν ανήκει στο K, τότε:

$$\text{insert}(x, \text{RBST}(K)) = \text{RBST}(K \cup \{x\})$$

Συμπεράσματα

Η εργασία παρουσίασε τους πρώτους αλγορίθμους για Binary Search Trees που εγγυώνται τη διατήρηση της ιδιότητας του Randomized BST.

- Η λειτουργία Εισαγωγής (Insertion) (με χρήση insert_at_root) παράγει **πάντα** ένα RBST, ανεξάρτητα από τη σειρά εισόδου.
- Ο αλγόριθμος Διαγραφής (Deletion) (με χρήση join) **επιλύει το παλαιό ανοικτό πρόβλημα** διαγραφής ενός κόμβου από ένα RBST έτσι ώστε το αποτέλεσμα να παραμένει RBST
- Το αποτέλεσμα **οποιασδήποτε αυθαίρετης ακολουθίας** εισαγωγών και διαγραφών είναι ένα **γνήσιο RBST.**

Συμπεράσματα

Επίσης:

- οι αλγόριθμοι (εισαγωγή, διαγραφή) είναι **πολύ απλούστεροι** από εκείνους των παραδοσιακών ισορροπημένων δέντρων
- Το αναμενόμενο κόστος κάθε βασικής λειτουργίας είναι O(logn) το οποίο επιτυγχάνεται χωρίς:
  - Την ανάγκη να υποθέσουμε τυχαία είσοδο
  - Τους μεγάλους σταθερούς παράγοντες των ισορροπημένων δέντρων
- Η χρήση των μεγεθών των υποδέντρων για τις τυχαίες επιλογές προσφέρει ένα **σημαντικό πλεονέκτημα**: υποστήριξη λειτουργιών κατά rank χωρίς την ανάγκη πρόσθετης αποθήκευσης ή τροποποίησης της βασικής δομής.

Συμπεράσματα

Για μελλοντικές εργασίες:

- Οι βασικοί αλγόριθμοι **split** και **join** μπορούν να χρησιμοποιηθούν για την κατασκευή πιο σύνθετων λειτουργιών συνόλων, όπως η ένωση (union), η τομή (intersection) και η διαφορά (difference) μεταξύ BSTs.
- Η εισαγωγή ενός **τυπικού αλγεβρικού πλαισίου** για την περιγραφή των τυχαιοποιημένων αλγορίθμων είναι μια σημαντική θεωρητική συνεισφορά (Ενότητα 7). Αυτό το πλαίσιο επιτρέπει την αυστηρή, συνοπτική και δομημένη απόδειξη της ιδιότητας της τυχαιότητας, το οποίο μπορεί να χρησιμοποιηθεί για τη μελέτη και άλλων τυχαιοποιημένων δομών δεδομένων πέρα από τα BSTs.

ΕΡΩΤΗΣΕΙΣ;

ΕΥΧΑΡΙΣΤΩ ΓΙΑ ΤΗΝ ΠΡΟΣΟΧΗ ΣΑΣ!!