Δομές Δεδομένων

15η Διάλεξη

Δέντρα Δυαδικής Αναζήτησης

και

Κατακερματισμός

Ε. Μαρκάκης

Περίληψη

- Υλοποιήσεις άλλων λειτουργιών σε ΔΔΑ:
- Επιλογή k-οστού μικρότερου
- Διαμέριση
- Αφαίρεση στοιχείου
- Ένωση 2 δέντρων
- Εισαγωγή στον κατακερματισμό
- Συναρτήσεις κατακερματισμού

Υλοποιήσεις άλλων λειτουργιών

- Επιλογή k-οστού μικρότερου κόμβου
  - Μετράμε τους κόμβους ανά υποδέντρο
  - Απαιτεί ύπαρξη μετρητή σε κάθε κόμβο: μετρά πόσους κόμβους έχει το υποδέντρο με ρίζα τον κόμβο
    - Η εισαγωγή και η αφαίρεση πρέπει να ενημερώνουν τον μετρητή!
    - Μειονέκτημα αν γίνονται πολλές εισαγωγές και αφαιρέσεις
  - Αναδρομή: Έστω ότι το αριστερό υποδέντρο έχει t κλειδιά
    - Αν k-1 < t ψάξε το k-οστό κλειδί στα αριστερά
    - Αν k-1 > t ψάξε το (k-t-1)-οστό κλειδί στα δεξιά
    - Αν k-1 = t τότε το k-οστό κλειδί είναι στη ρίζα

Υλοποιήσεις άλλων λειτουργιών

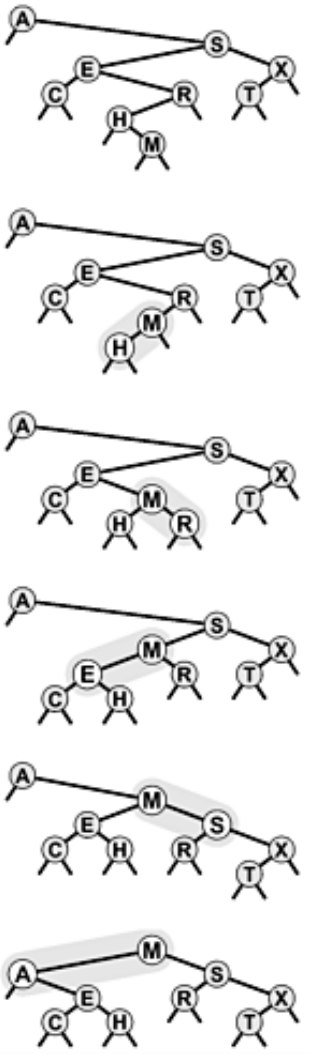
- Επιλογή k-οστού μικρότερου κόμβου

```
private ITEM selectR(Node h, int k) {
//για το k-οστό μικρότερο το όρισμα πρέπει να
  είναι k-1
  if (h == null) return null;
  int t = (h.l == null) ? 0 : h.l.N; /*πεδίο N
  σε κάθε κόμβο δηλώνει το μέγεθος του
  υποδέντρου με ρίζα τον κόμβο*/
  if (t > k) return selectR(h.l, k);
  if (t < k) return selectR(h.r, k-t-1);
  return h.item; }
ITEM select(int k) { return selectR(head, k); }
```

Υλοποιήσεις άλλων λειτουργιών

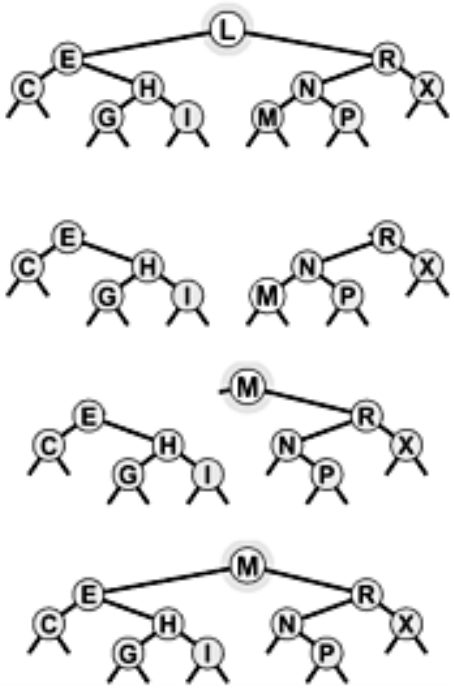
- Διαμέριση στον k-οστό κόμβο:
  - Αναδιάταξη ώστε να μπει το k-οστό μικρότερο στοιχείο στη ρίζα
  - Αρχικά εντοπίζουμε τον k-οστό κόμβο
  - Στη συνέχεια τον φέρνουμε στη ρίζα με περιστροφές
  - Παρόμοια τεχνική με την εισαγωγή στη ρίζα
  - Προσοχή: στο βιβλίο λείπουν οι εκχωρήσεις

```
Node partR(Node h, int k) {
  int t = (h.l == null) ? 0 : h.l.N;
  if (t > k) {
        h.l = partR(h.l, k);
        h = rotR(h); }
  if (t < k) {
        h.r = partR(h.r, k-t-1);
        h = rotL(h); }
  return h; }
```



Υλοποιήσεις άλλων λειτουργιών

- Αφαίρεση κόμβου από ΔΔΑ
  - Αναδρομική αφαίρεση από το κατάλληλο υποδέντρο
  - Τι γίνεται αν ο κόμβος είναι στη ρίζα ενός υποδέντρου;
  - Πώς πρέπει να ενώσουμε τα 2 υποδέντρα ώστε να αποκατασταθεί η ιδιότητα των ΔΔΑ;
  - Πολλές επιλογές. Π.χ.:
    - Εντοπίζουμε το μικρότερο κλειδί του δεξιού υποδέντρου (διαμέριση με k=0)
    - Το φέρνουμε στη ρίζα με περιστροφές
    - Θέτουμε ως αριστερό παιδί του το αριστερό υποδέντρο



Υλοποιήσεις άλλων λειτουργιών

- Υλοποίηση αφαίρεσης κόμβου από ΔΔΑ
  - Προσοχή: στο βιβλίο λείπουν οι εκχωρήσεις

```
private Node joinLR(Node a, Node b) {
  if (b == null) return a;
  b = partR(b, 0); //διαμέριση με k=0
  b.l = a; //το a θα γίνει το αριστερό υποδέντρο του b
  return b; }
private Node removeR(Node h, KEY v) {
  if (h == null) return null;
  KEY w = h.item.key();
  if (less(v, w)) h.l = removeR(h.l, v);
  if (less(w, v)) h.r = removeR(h.r, v);
  if (equals(v, w)) h = joinLR(h.l, h.r);
  return h; }
void remove(KEY v) { removeR(head, v); }
```

Υλοποιήσεις άλλων λειτουργιών

- Θα μπορούσαμε να βάλουμε ως ρίζα το μεγαλύτερο από το αριστερό υποδέντρο
- Το δέντρο που προκύπτει με τέτοιες προσεγγίσεις δεν είναι «εντελώς τυχαίο»
- Αν στη συνέχεια εφαρμοστούν ζεύγη από αφαιρέσεις και εισαγωγές το δέντρο δεν θα είναι ισορροπημένο (μέσο ύψος περίπου $\sqrt{N}$
- Ράθυμη αφαίρεση
  - Δεν αφαιρούμε, απλά μαρκάρουμε τον κόμβο ως μη ενεργό
  - Στην αναζήτηση αγνοούμε τους μαρκαρισμένους κόμβους
  - Περιοδικά ανακατασκευάζουμε τη δομή

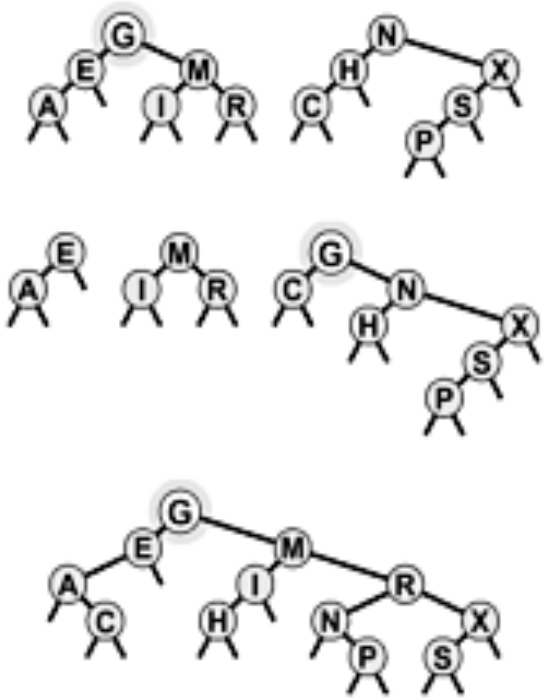
Υλοποιήσεις άλλων λειτουργιών

- Ένωση ή συγχώνευση ΔΔΑ
  - Έστω ότι έχουμε τα ΔΔΑ Α και Β και θέλουμε να τα ενώσουμε σε ένα δέντρο
  - Σε αντίθεση με την `joinLR` της αφαίρεσης, τώρα δεν ισχύει ότι όλα τα στοιχεία του Α είναι μικρότερα ή ίσα από τα στοιχεία του Β
  - Οι προφανείς λύσεις δεν είναι ικανοποιητικές
    - Παράδειγμα: εισαγωγή όλων των κόμβων του δέντρου Α στο Β
  - Αναδρομική συγχώνευση γραμμικού χρόνου:
    - Εισάγουμε τη ρίζα του Α ως ρίζα του Β
    - Συγχωνεύουμε το αριστερό υπόδενδρο του Β με το αριστερό του Α
    - Συγχωνεύουμε το δεξί υπόδενδρο του Β με το δεξί του Α
    - Κάθε κόμβος οδηγεί σε μία μόνο αναδρομική κλήση

Υλοποιήσεις άλλων λειτουργιών

- Υλοποίηση αναδρομικής συγχώνευσης

```
private Node joinR(Node a, Node b) {
  if (b == null) return a;
  if (a == null) return b;
  insertT(b, a.item);
  b.l = joinR(a.l, b.l);
  b.r = joinR(a.r, b.r);
  return b; }
public void join(ST b) {
  head = joinR(head, b.head); }
```



- Παράδειγμα
  - Συγχώνευση του αριστερού δέντρου στο δεξί

Υλοποιήσεις άλλων λειτουργιών

- Συμπεράσματα:
- Μην ξεχνάμε να ενημερώνουμε και τον μετρητή για το μέγεθος κάθε υποδέντρου, σε εισαγωγές, αφαιρέσεις, περιστροφές, κτλ
- ΔΔΑ αρκετά αποδοτική υλοποίηση πίνακα συμβόλων όταν ο χώρος δεν είναι πρόβλημα
- Οι βασικές λειτουργίες αναζήτησης και εισαγωγής έχουν καλές επιδόσεις κατά μέσο όρο
- Πρόβλημα όταν τα δέντρα αρχιζουν να γίνονται μη ισορροπημένα
- Υπάρχουν όμως μέθοδοι για την αναδιοργάνωση και τη μείωση του ύψους

Κεφάλαιο 14

Κατακερματισμός (Hashing)

Εισαγωγή

- Βασική ιδέα: άμεσος εντοπισμός στοιχείων
  - Μέχρι τώρα υποθέταμε ότι τα κλειδιά έχουν κάποια διάταξη
  - Οι αλγόριθμοι αναζήτησης που εξετάσαμε στηρίζονται σε λειτουργίες συγκρίσεων μεταξύ κλειδιών
    - Υπάρχει μέθοδος less()
  - Στα Δυαδικά Δέντρα αναζήτησης εκμεταλλευόμαστε τη διάταξη
    - Το αριστερό υποδέντρο έχει κλειδιά μικρότερα της ρίζας
  - Στον κατακερματισμό δεν εκμεταλλευόμαστε τη διάταξη
    - Αρκεί να υπάρχει μέθοδος equals()
  - Το κλειδί μετασχηματίζεται σε θέση (αριθμοδείκτη) πίνακα
    - Απαιτείται συνάρτηση κατακερματισμού για μετασχηματισμό
  - Πολλά κλειδιά μπορεί να μετασχηματίζονται στην ίδια θέση
    - Απαιτείται μέθοδος διαχείρισης συγκρούσεων (collision resolution) για τις θέσεις
    - Μπορεί να γίνει χρησιμοποιώντας λίστες

Εισαγωγή

- Ο κατακερματισμός αναδεικνύει τον ανταγωνισμό χώρου-χρόνου (time-space tradeoff)
  - Αν έχουμε πολύ μνήμη, μπορεί να απαιτεί λίγο χρόνο
  - Αν έχουμε πολύ χρόνο, μπορεί να απαιτεί λίγη μνήμη
  - Μπορούμε να επιτύχουμε όποια ισορροπία θέλουμε προσαρμόζοντας το μέγεθος του πίνακα κατακερματισμού
- Οι λειτουργίες εισαγωγής και αναζήτησης σε πίνακα συμβόλων με χρήση κατακερματισμού μπορεί να γίνουν ακόμα και σε σταθερό χρόνο και οι 2 (υπό κάποιες υποθέσεις)
  - Χρόνος εκτέλεσης όμως εξαρτάται και από μέγεθος κλειδιού
- Ο κατακερματισμός δεν παρέχει πολύ αποδοτικές υλοποιήσεις άλλων λειτουργιών όπως η επιλογή του k-οστού μικρότερου

Συναρτήσεις κατακερματισμού

- Μορφή συνάρτησης κατακερματισμού
  - A: σύνολο αντικειμένων, M: μέγεθος πίνακα
  - h: A → [0, M-1]
  - Συνήθως μετασχηματίζουμε μόνο το κλειδί
  - Ιδανικά η έξοδος πρέπει να φαίνεται τυχαία
    - Για κάθε αντικείμενο κάθε θέση είναι εξίσου πιθανή
- Τύποι κλειδιών
  - Κάθε τύπος απαιτεί διαφορετική συνάρτηση
    - Η διεπαφή του τύπου πρέπει να περιέχει τη συνάρτηση
  - Στη C χρησιμοποιούμε τη δυαδική μορφή των κλειδιών
    - Η πρακτική αυτή οδηγεί σε μη μεταφέρσιμο κώδικα
  - Στη Java η δυαδική μορφή είναι κρυμμένη
    - Πάντως όλα τα αντικείμενα έχουν μία μέθοδο hashCode!

Συναρτήσεις κατακερματισμού

- Κλειδιά κινητής υποδιαστολής
  - Έστω ότι ανήκουν στο διάστημα [0,1)
    - h(k) = floor(k*M)
  - Έστω ότι ανήκουν στο διάστημα [s,t)
    - h(k) = floor((k-s)/(t-s)*M)
  - Παράδειγμα με M = 97
  - 3 collisions, 17, 53, 76

.513870656	51
.175725579	17
.308633685	30
.534531713	53
.947630227	94
.171727657	17
.702230930	70
.226416826	22
.494766086	49
.124698631	12
.083895385	8
.389629811	38
.277230144	27
.368053228	36
.983458996	98
.535386205	53
.765678883	76
.646473587	64
.767143786	76
.780236185	78
.822962105	82
.151921138	15
.625476837	62
.314676344	31
.346903890	34

Συναρτήσεις κατακερματισμού

- Ακέραια κλειδιά w bits
  - Διαίρεση με 2^w για μετατροπή σε [0,1)
  - Συνεχίζουμε όπως για κλειδιά κινητής υποδιαστολής
  - Απλή λύση: h(k) = (k*M)<<w //δεξιά ολίσθηση για τη διαίρεση με 2^w
    - Αγνοεί πολλά από τα bits του κλειδιού
    - Μπορεί να δημιουργηθούν όμως πολλές συγκρούσεις
  - Καλύτερη λύση: h(k)= k mod M (συνάρτηση κατακερματισμού υπολοίπου)
    - Το M πρέπει να είναι πρώτος
  - Επιτυγχάνει καλύτερη διασπορά
  - Κάνει και για κινητή υποδιαστολή: h(k) = (k*2^w) mod M
    - Με w δυαδικά ψηφία στο δεκαδικό μέρος

Συναρτήσεις κατακερματισμού

- Άλλα αντικείμενα
  - Μετατρέπουμε το κλειδί σε δυαδική μορφή
  - Κάθε αντικείμενο είναι ουσιαστικά μία ακολουθία από bits
  - Χρησιμοποιούμε τη μέθοδο του υπολοίπου
- Γιατί το M πρέπει να είναι πρώτος;
  - Έστω ότι τα κλειδιά είναι συμβολοσειρές ASCII
    - Τα αντιμετωπίζουμε ως ακεραίους με βάση το 128 (7 bits)
  - Ιδανικά θέλουμε να χρησιμοποιούμε όλα τα bit
  - Έστω ότι διαλέγουμε $M = 64 = 2^6$
    - Ουσιαστικά χρησιμοποιούμε μόνο τα τελευταία 6 bit!
    - Είναι καλό να αποφεύγουμε δυνάμεις του 2
  - Με M πρώτο αποφεύγουμε κοινούς παράγοντες

Συναρτήσεις κατακερματισμού

- Χρήσιμοι πρώτοι αριθμοί
  - Οι δυναμικοί πίνακες με την πάροδο του χρόνου θα πρέπει να διπλασιάζονται ή να μειώνονται στο μισό ανάλογα με το πλήθος των εγγραφών
  - Αν το M όμως είναι πρώτος, θα πρέπει να βρούμε κάποιον άλλον πρώτο «κοντά» στο 2M
  - Συνήθης τακτική: επιλέγουμε πρώτους αριθμούς που είναι κοντά σε δυνάμεις του 2.
  - π.χ. Mersenne primes: πρώτοι της μορφής 2^m-1
  - Για αρκετές (αλλά όχι για όλες) τιμές του m, ο 2^m-1 είναι πρώτος αριθμός
  - Χρησιμοποιούμε πάντα τον πλησιέστερο πρώτο στο 2^n (από κάτω)

n	δ_n	$2^n - \delta_n$
8	5	251
9	3	509
10	3	1021
11	9	2039
12	3	4093
13	1	8191
14	3	16381
15	19	32749
16	15	65521
17	1	131071
18	5	262139
19	1	524287
20	3	1048573
21	9	2097143
22	3	4194301
23	15	8388593
24	3	16777213
25	39	33554393
26	5	67108859
27	39	134217689
28	57	268435399
29	3	536870909
30	35	1073741789
31	1	2147483647

Συναρτήσεις κατακερματισμού

- Εναλλακτική μέθοδος υπολοίπου
  - h(k) = floor(k*a) mod M
  - Με αυτή τη μέθοδο ο M δεν χρειάζεται πλέον να είναι πρώτος
  - Αρκεί να επιλέξουμε το κατάλληλο a!
    - Μία καλή επιλογή είναι φ = 0,618033
    - Ο φ είναι η χρυσή τομή (golden ratio)
- Παράδειγμα
  - Ακέραια κλειδιά των 16 bit σε δεκαδική μορφή
  - Με h(k) = k % 97 έχουμε καλή κατανομή
  - Με h(k) = k % 100 έχουμε κακή κατανομή
    - Ουσιαστικά χρησιμοποιούμε 2 δεκαδικά ψηφία
  - Με h(k) = (int) (φ * k) % 100 έχουμε καλή κατανομή
    - Το φ αναιρεί τα προβλήματα του 100

16838	57	38	6
5758	35	58	58
10113	25	13	50
17515	55	15	24
31051	11	51	90
5627	1	27	77
23010	21	10	20
7419	47	19	85
16212	13	12	19
4086	12	86	25
2749	33	49	98
12767	60	67	90
9084	63	84	14
12060	32	60	53
32225	21	25	16
17543	83	43	42
25089	63	89	5
21183	37	83	91
25137	14	37	35
25566	55	66	0
26966	0	66	65
4978	31	78	76
20495	28	95	66
10311	29	11	72
11367	18	67	25