Εθνικό Μετσόβιο Πολυτεχνείο
Σχολή ΗΜ&ΜΥ
Αλγόριθμοι και Πολυπλοκότητα
7ο εξάμηνο, Ροή Λ
Ακαδημαϊκή περίοδος: 2010-2011

1η Σειρά Γραπτών Ασκήσεων

Γερακάρης Βασίλης
<vgerak@gmail.com>
Α.Μ.: 03108092

8 Δεκεμβρίου 2011

1 Ασυμπτωτικός συμβολισμός, Αναδρομικές Σχέσεις

i) Ταξινόμηση

(1) $\log n^3 = 3\log n = \Theta(logn)$

(2) $\sqrt{n} * \log^{50} n$

(3) $\frac{n}{\log\log n}$

(4) $\log n! = \Theta(nlogn)$

(5) $n * \log^{10} n$

(6) $n^{1.01}$

(7) $5^{\log_2 n} = n^{log_2 5} = \Theta(n^{2.322})$

(8) $\sum_{k=1}^{n} k^5 = k^6$

(9) $\log^{\log n} n = n^{\log\log n}$

(10) $2^{\log_2^4 n}$

(11) $\log^{\sqrt{n}} n$

(12) $e^{\frac{n}{\ln n}}$

(13) $n3^n$

(14) 2^{2n}

(15) $\sqrt{n!}$

ii) Τάξη Μεγέθους

(1) $T(n) = 5T(n/7) + n\log n$, $(a = 5, b = 7)$
$\Rightarrow n^{\log_7 5} = n^{0.827} \Rightarrow n^{0.827} < n < n\log n$
$\Rightarrow T(n) \in \Theta(n\log n)$ M.Th (case #3)

(2) $T(n) = 4T(n/5) + n/\log^2 n$, $(a = 4, b = 5)$
$\Rightarrow n^{\log_5 4} = n^{0.861} \Rightarrow n^{0.861} < n/\log^2 n$
$\Rightarrow T(n) \in \Theta(n/\log^2 n)$ M.Th (case #3)

(3) $T(n) = T(n/3) + 3T(n/7) + n$ $(\frac{1}{3} + \frac{1}{7} + \frac{1}{7} + \frac{1}{7} < 1)$
$\Rightarrow T(n) \in \Theta(n)$ M.Th (special case)

(4) $T(n) = 6T(n/6) + n$
$\Rightarrow n^{\log_6 6} = n$
$\Rightarrow T(n) \in \Theta(n\log n)$ M.Th (sp. case #2)

(5) $T(n) = T(n/3) + T(2n/3) + n$
$\Rightarrow T(n) \in \Theta(n\log n)$

(6) $T(n) = 16T(n/4) + n^3\log^2 n$, $(a = 16, b = 4)$
$\Rightarrow n^{\log_4 16} = n^2 \Rightarrow n^2 < n^3\log^2 n$
$\Rightarrow T(n) \in \Theta(n^3\log^2 n)$ M.Th (case #3)

(7) $T(n) = T(\sqrt{n}) + \Theta(\log\log n)$ (Θέτω $k = \log n, f(k) = T(n)$)
$\Rightarrow T(\sqrt{n}) = f(\sqrt{k}) = f(\log\sqrt{n}) = f(\frac{\log n}{2}) = f(\frac{k}{2})$
$\Rightarrow f(k) = f(\frac{k}{2}) + \Theta(\log k)$
$\Rightarrow \Theta(n)$ (M.Th (case #1)

(8) $T(n) = T(n-3) + \log n$ $(\frac{n}{3} * \log n)$
$\Rightarrow T(n) \in \Theta(nlogn)$

2 Ταξινόμηση σε Πίνακα με Πολλά Ίδια Στοιχεία

Δημιουργούμε ένα 2-διάστατο πίνακα $B[\log^d n, 2]$ όπου η 1η γραμμή θα περιέχει τα διαφορετικά στοιχεία ταξινομημένα και η 2η το πλήθος εμφάνισης του κάθε στοιχείου.
Αρχίζουμε θέτονταςτο $B[1,1] = A[1]$, $B[1,2] = 1$, i=2.
Για κάθε $A[i]$ στον πίνακα, εφαρμόζουμε Binary Search στην πρώτη γραμμή του πίνακα B.
(Με κόστος $\Theta(nloglog^d n)$)

- Αν το στοιχείο δεν υπάρχει στον πίνακα B (Η binary search επιστρέψει 0)
  τότε εφαρμόζουμε Insertion sort του στοιχείου στον πίνακα B, τοποθετώντας το στη σωστή θέση (αντιμεταθέτωντας και τη 2η γραμμή όταν χρειάζεται)
  Με κόστος: $(\log^d n)^2$
- Αν το στοιχείο υπάρχει στον πίνακα (έστω στη θέση k), αυξάνουμε το $B[k,2]$ κατά 1.

Όταν τελειώσουμε, αναπτύσσουμε τα δεδομένα του (ταξινομημένου) πίνακα B (το στοιχείο $B[i,1]$ αναπτύσσεται $B[i,2]$ φορές) σε γραμμικό χρόνο.

Συνολικό κόστος: $(nloglog^d n) + (\log^d n)^2 + n = O(nloglog^d n)$

3 Δυαδική Αναζήτηση

i) Ξεκινάμε (για $i = 1$) συγκρίνοντας το $A[i]$ με το x. Όσο το x είναι μεγαλύτερος από αυτό, κρατάμε την τιμή του $(i+1)$ σε μια άλλη μεταβλητή k και διπλασιάζουμε το i. Μόλις φτάσουμε σε μεγαλύτερο αριθμό από τον x, εφαρμόζω ένα έλεγχο. Αν το στοιχείο $A[i]$ είναι διάφορο του ∞, τότε εφαρμόζω δυαδική αναζήτηση για το στοιχείο x στο τμήμα $[k..i]$ του πίνακα και έχω το αποτέλεσμα για πιθανή ύπαρξη & θέση του στοιχείου.
Αν το στοιχείο $A[i]$ είναι ίσο με ∞, τότε ελέγχω αν $k = i$ (τερματική συνθήκη) και αν όχι ελέγχω το $A[(k+i)div2]$.

- Αν είναι ∞ τότε θέτω $i = (k+i)div2$ και επαναλαβάνω.
- Αν είναι αριθμός m τον συγκρίνω με το x.
  - Αν $x > m$ τότε θέτω $k = (k+i)div2$ και επαναλαμβάνω.
  - Αν $x < m$ τότε εφαρμόζω δυαδική αναζήτηση στο τμήμα $A[k..i]$ του πίνακα.
  - Αν $x = m$ τότε είναι το ζητούμενο στοιχείο.

Η τελική πολυπλοκότητα είναι $O(\log n)$.

ii) Εργαζόμαστε με τα k πρώτα στοιχεία από κάθε πίνακα (συμπληρώνοντας τις κενές θέσεις με + ∞ , αν ο πίνακας δεν έχει τόσα στοιχεία). Συγκρίνουμε τα $A[k/2]$ με $B[k/2+1]$ και $A[k/2+1]$ με $B[k/2]$.

- Αν $A[k/2] < B[k/2{+}1]$ και $B[k/2] < A[k/2{+}1]$ τοτε το k-οστό στοιχείο είναι το $max\{A[k/2], B[k/2]\}$.
- Αν $A[k/2] < B[k/2+1]$ και $B[k/2] > A[k/2+1]$ τότε επαναλαμβάνουμε αναδρομικά τη διαδικασία στους υποπίνακες $A[k/2+1]..A[k]$ και $B[1]..B[k/2]$ για $k = \frac{k}{2}$.

Τελική πολυπλοκότητα: $O(logk)$

4 Συλλογή Comics

Ελέγχουμε αρχικά το 1ο bit για όλα τα τεύχη, χωρίζοντάς τα σε 2 σύνολα με πλήθος στοιχείων $\frac{n}{2}$ και $\frac{n}{2}-1$. Το τέυχος που λείπει είναι στο σύνολο με τα λιγότερα στοιχεία, οπότε αποθηκεύουμε την τιμή αυτού του bit.
Εφαρμόζουμε αναδρομικά τη μέθοδο για το επόμενο bit, κάθε φορά στο σύνολο με τα λιγότερα στοιχεία, κάνοντας τις μισές ερωτήσεις από το προηγούμενο βήμα σε κάθε αναδρομή.
Τελικά, θα καταλήξουμε στον κωδικό του τεύχους που λείπει από τη συλλογή, έχοντας ρωτήσει $n + \frac{n}{2} + \frac{n}{4} + .. + 1 \approx 2n$ φορές.

Πρακτικά, είναι: $(n-1) + (\frac{n}{2}-1) + (\frac{n}{4}-1) + .. = \sum_{i=0}^{\log n} (\frac{n}{2^i} - 1) = 2n - logn - 2$

5 Πολυκατοικίες χωρίς Θέα

Αρχίζουμε αρχικοποιώντας $B[1] = 0$, μιάς και ο 1ος δεν έχει κτήριο δυτικά του.
Για κάθε κτήριο i ανατολικά του, θέτουμε $k = i - 1$ και συγκρίνουμε το ύψος του $A[i]$ με του $A[k]$.

- Αν $A[i] < A[k]$, τότε $B[i] = k$ και συνέχιζουμε στο επόμενο δεξιά κτήριο ($i = i + 1$) .
- Αν $A[i] \geq A[k]$,
  - Αν $B[k] = 0$ τότε $B[i] = 0$ και συνέχιζουμε στο επόμενο δεξιά κτήριο.
  - Αλλιώς θέτουμε $k = B[k]$ και επαναλαμβάνουμε τον έλεγχο.

Θα αποδείξουμε ότι εκτελείται σε γραμμικό χρόνο. Η βασική ιδέα είναι ότι γίνεται ένα tradeoff μεταξύ του αριθμού των ελέγχων που γίνονται σε κάθε βήμα και των πιθανών ελέγχων που θα μπορούν να εκτελεστούν στα επόμενα.
Για κάθε στοιχείο i οι μέγιστοι πιθανοί έλεγχοι είναι i μειωμένοι κατά τους επιπλέον ελέγχους που πραγματοποιήθηκαν στα προηγούμενα βήματα (δεν καταμετρούνται οι έλεγχοι με το αμέσως δυτικότερα κτήριο).
Δηλαδή, αν k είναι ο μετρητής των επιπλέον ελέγχων που έγιναν, θα ισχύει:
MaxComparisons(i) = i - k -1
Έτσι, κάθε extra έλεγχος που γίνεται σε κάποιο βήμα, μειώνει τους πιθανούς ελέγχους όλων των υπολοίπων.