Αλγόριθμοι και Πολυπλοκότητα
1η Σειρά Γραπτών Ασκήσεων

Γιαννέλος Γιάννης
ΑΜ:03108088

8 Δεκεμβρίου 2011

Άσκηση 1: Ασυμπτωτικός Συμβολισμός, Αναδρομικές Σχέσεις

(α)

Ταξινομώντας τις συναρτήσεις της άσκησης σε αύξουσα σειρά τάξης μεγέθους προκύπτει η εξής διάταξη :

$g_1 = \log n^3$	$g_2 = \sqrt{n}(\log n)^{50}$	$g_3 = \frac{n}{\log\log n}$
$g_4 = \log n!$	$g_5 = n(\log n)^{10}$	$g_6 = n^{1.01}$
$g_7 = 5^{\log n}$	$g_8 = \sum\limits_{k=1}^{n} k^5$	$g_9 = (\log n)^{\log n} = n^{\log\log n}$
$g_{10} = 2^{(\log n)^4}$	$g_{11} = (\log n)^{\sqrt{n}}$	$g_{12} = e^{\frac{n}{\ln n}}$
$g_{13} = n \cdot 3^n$	$g_{14} = 2^{2n}$	$g_{15} = \sqrt{n!}$

(β)

1. Χρηση του Master Theorem:

   - $\alpha = 5,\ \beta = 7$ άρα $n^{\log_\beta \alpha} = n^{\log_7 5} \simeq n^{0.827}$
   - $f(n) = n\log n = \Omega(n^{\log_7 5+\epsilon})$
   - $5f(\frac{n}{7}) = 5\frac{n}{7}log(\frac{n}{7}) \leq \frac{5}{7}n\log(n)$

   Άρα σύμφωνα με την 3η περίπτωση του Master Theorem προκύπτει ότι: $T(n) = \Theta(n\log n)$

2. Χρηση του Master Theorem:

   - $T(n) = 4T(\frac{n}{5}) + \frac{n}{(\log n)^2}$
   - $n^{\log_\beta \alpha} = n^{\log_5 4} \simeq n^{0.861}$

   Άρα $T(n) = \Theta(\frac{n}{(\log n)^2})$

3. $T(n) = T(\frac{n}{3}) + 3T(\frac{n}{7}) + n \Longrightarrow T(n) = \Theta(n\log n)$

4. Χρηση του Master Theorem:

   - $\alpha = 6,\ \beta = 6$ άρα $n^{\log_\beta \alpha} = n^{\log_6 6} = n$
   - $f(n) = \Theta(n^{\log_6 6})$

Άρα σύμφωνα με την 2η περίπτωση του Master Theorem προκύπτει ότι: $T(n) = \Theta(n \log n)$

5. Χρηση του Master Theorem:
   - $T(n)$ της μορφής $T(n) = T(\gamma_1 n) + T(\gamma_2 n) + \Theta(n)$
   - $\gamma_1 + \gamma_2 = \frac{1}{3} + \frac{2}{3} = 1$

   Άρα ισχύει $T(n) = \Theta(n \log n)$

6. Χρηση του Master Theorem:
   - $\alpha = 16$, $\beta = 4$ άρα $n^{\log_\beta \alpha} = n^{\log_4 16} = n^2$
   - $f(n) = n^3 log(n)^2 = \Omega(n^{\log_{16} 4 + \epsilon})$
   - $\alpha f(\frac{n}{\beta}) = 16 f(\frac{n}{4}) = 16 \frac{n^3}{4^3} log(\frac{n}{4})^2 = \frac{16}{4^3} n^3 \log(\frac{n}{4})^2 \leq c f(n)$ με $c < 1$

   Άρα σύμφωνα με την 3η περίπτωση του Master Theorem προκύπτει ότι: $T(n) = \Theta(n^3 (\log n)^2)$

7. $T(n) = T(\sqrt{n}) + \Theta(\log \log n)$. Έστω $k = \log n$, $f(k) = T(n)$. Άρα $T(\sqrt{n}) = f(\log \sqrt{n}) = f(\frac{\log n}{2}) = f(\frac{k}{2}) \implies f(k) = f(k/2) + \Theta(\log k)$. Επομένως: $T(n) = \Theta(\frac{(\log k)^2 (\log k + 1)}{2})$

8. $T(n) = T(n-3) + \log n \implies T(n) = \Theta(n \log n)$

Άσκηση 2: Ταξινόμηση σε πίνακα με πολλά ίδια στοιχεία

(α)

Ο πίνακας $A[1...n]$ χαρακτήριζεται απο πολλές εμφανίσεις των στοιχείων του, με $O(\log(n)^d)$ (για κάποια σταθέρα $d \geq 1$) διαφορετικά στοιχεία. Επομένως ο πίνακας που αποτελείται απο τα διαφορετικά στοιχεία του A, έστω $B[1...k]$, θα έχει πλήθος στοιχείων $k \in O(\log(n)^d)$. Αν εκμεταλευτούμε την ιδιότητα του πίνακα A μπορούμε να κατασκευάσουμε συγκριτικό αλγόριθμο ταξινόμησης που θα έχει πολυπλοκότητα $(n \log \log n)$. Θεωρούμε χωρίς βλάβη της γενικότητας ότι ο A αποτελείται μόνο απο θετικά στοιχεία. Περιγραφικά, θα χρειαστούμε τις εξής διαδικασίες που θα απαρτίζουν τον αλγόριθμό μας:

- `freq_array()`: Συνάρτηση που δέχεται ως είσοδο έναν πίνακα (στην δικιά μας περίπτωση τον A) και μας επιστρέφει έναν πίνακα μεγέθους `max_array(A)` (μέγιστου στοιχείου του πίνακα) που για κάθε θέση του, έστω i, δείχνει την αντίστοιχη σύχνοτητα εμφάνισης του i στον A. Παράλληλα δημιουργεί εναν πίνακα B που περιέχει μόνο τα διαφορετικά στοιχεία του A και επιστρεφεί το diff_num που είναι το πλήθος των διαφορετικών στοιχείων.

- `qsort_array()`: Συνάρτηση που ταξινομεί έναν πίνακα με την quick sort.
- `reconstruct_array()`: Συνάρτηση που κατασκευάζει τον ζητούμενο ταξινομημένο πίνακα.

Ακολουθούν σε ψευδογλώσσα οι παραπάνω συναρτήσεις:

procedure $freq\_array(A)$
    $j = 1$
    $initialize(B)$ ▷ Αρχικοποίηση του πίνακα B με μηδενικά
    for i in $[1...n]$ **do**
        if $B[A[i]] = 0$ **then**
            $Unique[j] = A[i]$ ▷ Unique: πίνακας διαφορετικών στοιχείων
            $j \leftarrow j + 1$
        end if
        $B[A[i]] \leftarrow B[A[i]] + 1$
    end for
    $diff\_num \leftarrow j - 1$
    return $Unique, B, diff\_num$
end procedure

procedure $reconstruct\_array(B, Unique, diff\_num)$
    $init \leftarrow 1$
    $k \leftarrow diff\_num$
    for i in $[1...k]$ **do**
        for j in $[1...B[Unique[i]]]$ **do**
            $A[init] \leftarrow Unique[i]$
            $init \leftarrow init + 1$
        end for
    end for
    return A
end procedure

Η πολυπλοκότητα του αλγόριθμου μας θα ειναι $O(\log n^d \log\log n^d)$ που όμως είναι ίση με $O(n \log\log n)$ αφού $(\log n)^d \leq n$ (ακόμα και στην χειρότερη περίπτωση το πλήθος των διαφορετικών στοιχείων είναι μικρότερο απο το πλήθος όλων των στοιχείων). Αυτό συμβαίνει αφού αν καλέσουμε κατάλληλα τις παραπάνω συναρτήσεις για την ταξινόμηση του A, όλες είναι γραμμικής πολυπλοκότητας, εκτός απο την quick sort όπου θέλει $n \log n$ χρόνο. Ομως το μέγεθος του πίνακα που ταξινομεί είναι πολυλογαριθμικά μικρότερο απο αυτό του αρχικού πίνακα. Έτσι στην περίπτωση ταξινομήσης σαν και αυτή της άσκησης δεν ισχύει το κάτω φράγμα $\Omega(n \log n)$ γιατί εκμεταλλευόμαστε την ιδιότητα των στοιχείων μειώνοντας την χρονική πολυπλοκότητα αλλά αυξάνοντας πιθανά την χωρική (ο πίνακας B είναι μεγέθους `max_array(A)`, δηλαδή μέγεθος ίσο με το μέγιστο στοιχείο).

Άσκηση 3: Δυαδική Αναζήτηση

(α)

Ο αλγόριθμος για την επίλυση του προβλήματος θα αποτελέιται απο τα εξής βήματα:

- Εύρεση της τιμής του n (μέγεθος του πίνακα), σε λογαριθμικό χρόνο.
- Εκτέλεση δυαδικής αναζήτησης με δεδομένο το μέγεθος του πίνακα, πάλι σε λογαριθμικό χρόνο.

Έτσι συνολικά η πολυπλοκότητα του αλγορίθμου μας θα είναι $\log n + \log n$ δηλαδή $O(\log n)$. Για το 1ο βήμα:

- Ξεκινάμε με δείκτη $i = 1$ και τον αυξάνουμε με βήμα $2, 2^2, 2^3, 2^4, ...$ μέχρις ότου μας επιστραφεί το μήνυμα σφάλματος ∞.
- Έστω 2^k η θέση του πίνακα όπου κατά την προσπέλαση της εμφανίστηκε το μήνυμα σφάλματος.
- Εκτελούμε δυαδική αναζήτηση μεταξύ των θέσεων 2^{k-1} και 2^k μέχρι να βρούμε το ζητούμενο n, γνωρίζοντας ότι θα είναι το τελυταία στοιχείο που δεν θα εμφανιστεί το μήνυμα λάθους.

Στη συνέχεια, για το 2ο βήμα, εκτελούμε δυαδική αναζήτηση για το στοιχείο x στον πίνακα $A[1...n]$, αφού γνωρίζουμε πλέον και το μέγεθος του πίνακα A.

(β)

Για την επίλυση του προβλήματος σε αυτό το υποερώτημα θα χρειαστεί να εφαρμόσουμε την ιδέα της δυαδικής αναζήτησης στην ένωση 2 πινάκων. Αφού οι 2 πίνακες A,B είναι ταξινομημένοι και τα στοιχεία τους διαφορετικά, δεν ειναι δυνατό το ζητούμενο στοιχείο να βρίσκεται σε θέση μεγαλύτερη της k. Άρα η αναζήτηση περιορίζεται στα στοιχεία $A[1...k]$, $B[1...k]$.

```
procedure k_smallest(A[n], B[n])
    i ← k div 2
    j ← k − i
    step ← k div 4
    while step > 0 do
        if A[i] > B[j] then
            i ← i − step
            j ← j + step
        else
            i ← i + step
            j ← j − step
        end if
        step ← step div 2
    end while
    if A[i] > B[j] then
        x ← A[i]
    else
        x ← B[j]
    end if
    return x
end procedure
```

Άσκηση 4: Συλλογή comics

Η λύση του προβλήματος της Συλλογης Comics στηρίζεται εν μέρη στην ιδέα της δυαδικής αναζήτησης. Συγκεκριμένα, η ιδέα που θα βασιστούμε για την κατασκευή του αλγορίθμου είναι ότι μετά απο κάθε ανάγνωση των ψηφίων ενός comic θα ελλατώνουμε στο μισό τα ψηφία που θα πρέπει να εξεταστούν (δηλ. συμφωνα με την εκφώνηση, τις ερωτήσεις της μορφής *"Ποιό είναι το i-οστό bit στην αρίθμηση του τέυχους j"*) στην επόμενη επανάληψη. Έτσι σε κάθε βήμα χρειάζεται να διαβάζουμε n ψηφία τα οποία στην επόμενη αναζήτηση μειώνονται στο μισό άρα:

- $T(n) = n + T(n/2) \Longrightarrow T(n) = 2n - 2 + c1$
- Πολυπλοκοτητα: $n + \frac{n}{2} + \frac{n}{4} + \frac{n}{8} + ... = \sum_{k=0}^{\log n} \frac{n}{2k} \Longrightarrow O(n)$

Θεωρούμε ότι ο δείκτης i είναι αρχικά 0 και το σύνολο των τευχών που εξετάζουμε ειναι το $ComicsSet$ με αρχικά όλα τα τευχη. Ακόμα, έστω $ComicsNumber$ κάθε φορά το σύνολο των στοιχείων του $ComicsSet$. Τα βήματα του αλγορίθμου μας θα είναι τα εξής παρακάτω:

1. Διάβαζουμε το i-οστό (απο το τέλος) ψηφίο σε κάθε τέυχος του $ComicsSet$ και ανάλογα με το αν είναι 1 η 0 χωρίζουμε τα τεύχη σε 2 σύνολα.
2. Αν υπάρχουν περισσότερα 0 απο 1 τότε το τεύχος που λείπει βρίσκεται στα $\frac{1}{2} \cdot ComicsNumber$ μεγαλύτερα στοιχεία του τρέχοντος $ComicsSet$. Το τρέχον $ComicsSet$ γίνεται το σύνολο με τα τέυχη με 1 στο i-οστο ψηφίο.
3. Αν υπάρχουν περισσότερα 1 απο 0 τότε το τέυχος που λείπει βρίσκεται στα $\frac{1}{2} \cdot ComicsNumber$ μικρότερα στοιχεία του τρέχοντος $ComicsSet$. Το τρέχον $ComicsSet$ γίνεται το σύνολο με τα τέυχη με 0 στο i-οστο ψηφίο.
4. Πήγαινουμε ξανά στο βήμα 1 για $i = i + 1$ μέχρι το $ComicsNumber$ να γίνει 1.
5. Τέλος το στοιχείο που ψάχνουμε είναι το στοιχείο με αριθμό τεύχους ιδιο με αυτό του τελευταίου στοιχείου του $ComicsSet$, αν αλλάξουμε το Less Significan Bit στο συμπληρωματικό του.

Άσκηση 5: Πολυκατοικίες χωρίς θέα

Για την επίλυση του προβλήματος «Πολυκατοικιες χωρις θεα» θα χρησιμοποιήσουμε μια παραλλαγμένη μορφή του αλγόριθμου που βρίσκει τις κοντινότερες μικρότερες τιμές «All nearest smaller values» σε μια ακολουθία αριθμών (πολυπλοκότητα $O(n)$). Στην προκειμένη περίπτωση χρειάζεται να βρούμε όλες τις κοντινότερες μεγαλύτερες τιμές κάθε στοιχείου του πίνακα Α. Ένας αποδοτικός τρόπος για να υλοποιήσουμε το παραπάνω θα είναι χρησιμοποιώντας τη δομή δεδομένων "στοίβα" (χρησιμοποιούνται χωρίς να οριστούν οι συναρτήσεις $push()$, $pop()$, $top()$ για την εισαγωγή και εξαγωγή στοιχείων απο την στοιβα και για το κορυφαίο στοιχείο της). Ακολουθεί υλοποίηση του παραπάνω αλγόριθμου σε ψευδογλώσσα:

- Πολυπλοκοτητα Αλγοριθμου: $O(n)$

```
procedure NearestLargerValues(A[n])
    initialize(S)                                ▷ Αρχικοποίηση άδειας στοίβας S
    i ← 1
    for x in A[1...n] do
        while S is nonempty and top(S) ≤ x do
            pop(S)
        end while
        if empty(S) then
            x has no preceeding larger value
            B[i] ← 0
        else
            top(S) is the nearest larger value to x
            B[i] ← x
            i ← i + 1
        end if
        push(x)
    end for
    return B[n]
end procedure
```

Ο αλγόριθμος εκτελείται σε γραμμικό χρόνο, αν και αυτό δεν είναι εμφανές λόγω των εμφωλευμένων επαναλήψεων. Είναι γραμμικής πολυπλοκότητας όμως, αφού κάθε επανάληψη του εσωτερικού βρόγχου αφαιρεί ένα στοιχείο το οποίο είχε προστεθεί σε μια προηγούμενη επανάληψη του εξωτερικού βρόγχου.