Εθνικο Μετσοβιο Πολυτεχνειο

1η Γραπτή Άσκηση

Αλγόριθμοι & Πολυπλοκότητα

Σπουδαστής:
Διονύσης Ζηνδρος (06601)
<dionyziz@gmail.com>

Διδάσκοντες:
Στάθης Ζαχος
Δημήτρης Φωτακης

8 Δεκεμβρίου 2011

Άσκηση 1

(α)

Αρχικά χωρίζουμε τις συναρτήσεις σε πολυωνυμικές και μη πολυωνυμικές και στη συνέχεια ταξινομούμε τις επί μέρους συναρτήσεις. Παρατηρούμε τις εξής χρήσιμες σχέσεις:

$$\begin{aligned}
\log n^3 &\in \Theta(\log n) \\
\log(n!) &\in \Theta(n \log n) \\
5^{(\lg n)} &\in \Theta(n^{\frac{1}{\log_5 2}}) \\
(\log n)^{\log n} &\in \Theta(n^{\log \log n})
\end{aligned}$$

$$\begin{aligned}
\sum_{k=1}^{n} k^5 &\in O\left(\sum_{k=1}^{n} n^5\right) \\
&= O(n \times n^5) \\
&= O(n^6) \\
\sum_{k=1}^{n} k^5 &\in \Omega\left(\sum_{k=1}^{n} \left(\frac{n}{2}\right)^5\right) \\
&= \Omega\left(\left(\frac{n}{2}\right) \times \left(\frac{n}{2}\right)^5\right) \\
&= \Omega\left(\left(\frac{n}{2}\right)^6\right) = \Omega(n^6) \\
\sum_{k=1}^{n} k^5 &\in \Theta(n^6)
\end{aligned}$$

$$\begin{aligned}
\sqrt{n!} &\in O(\sqrt{n^n}) \\
&= O(n^{\frac{n}{2}}) \\
\sqrt{n!} &\in \Omega\left(\sqrt{(\frac{n}{2})^{\frac{n}{2}}}\right) \\
&= \Omega\left(\left(\frac{n}{2}\right)^{\frac{n}{4}}\right)
\end{aligned}$$

Με βάση αυτά καταλήγουμε στην παρακάτω διάταξη:

$$\begin{array}{lll}
\Theta(\log n^3) \subset & \Theta(\sqrt{n}\log^{50} n) \subset & \Theta\left(\frac{n}{\log\log n}\right) \subset \\
\Theta(\log(n!)) \subset & \Theta(n\log^{10} n) \subset & \Theta(n^{1.01}) \subset \\
\Theta(5^{(\lg n)}) \subset & \Theta\left(\sum_{k=1}^{n} k^5\right) \subset & \Theta((\log n)^{\log n}) = \\
\Theta(n^{\log\log n}) \subset & \Theta(2^{(\lg n)^4}) \subset & \Theta((\log n)^{\sqrt{n}}) \subset \\
\Theta(e^{\frac{n}{\ln n}}) \subset & \Theta(n3^n) \subset & \Theta(2^{2n}) \subset \\
\Theta(\sqrt{n!}) & &
\end{array}$$

(β)

1. Αφού $n\log n \in \Omega(n^{\log_7 5+\epsilon})$ άρα από το Master Theorem ϑα είναι:

$$T \in \Theta(n\log n)$$

2. Αφού $\frac{n}{log^2 n} \in \Omega(n^{log_5 4+\epsilon})$ άρα από το Master Theorem ϑα είναι:

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

3. Θα δείξω ότι $T \in O(n)$. Αρκεί να δείξω ότι $\forall n \in \mathbb{N} : T(n) \leq 21n$. Πράγματι, με ισχυρή επαγωγή έχουμε:

$$T(1) = 1 \leq 21$$

Έστω $\forall n < n_0 : T(n) \leq 21n$. Τότε:

$$\begin{aligned}
T(n_0) &= T\left(\frac{n_0}{3}\right) + 3T\left(\frac{n_0}{7}\right) + n_0 \\
&\leq 21\frac{n_0}{3} + 21 * 3 * \frac{n_0}{7} + n_0 \\
&= 7n_0 + 9n_0 + n_0 \leq 21n_0.
\end{aligned}$$

Άρα $T \in O(n)$.

Θα δείξω τώρα ότι $T \in \Omega(n)$. Αρκεί να δείξω ότι $\forall n \in \mathbb{N} : T(n) \geq n$. Πράγματι, με ισχυρή επαγωγή έχουμε:

$$T(1) = 1 \geq 1$$

Έστω $\forall n < n_0 : T(n) \geq n$. Τότε:

$$\begin{aligned} T(n_0) &= T\left(\frac{n_0}{3}\right) + 3T\left(\frac{n_0}{7}\right) + n_0 \\ &\geq \frac{n_0}{3} + 3\frac{n_0}{7} + n_0 \\ &\geq n_0 \end{aligned}$$

Άρα $T \in \Omega(n)$ και συνεπώς $T \in \Theta(n)$.

4. Αφού $n \in \Theta(n^{\log_6 6}) = \Theta(n)$ άρα από το Master Theorem θα είναι $T \in \Theta(n \log n)$.

5. Από το δέντρο της αναδρομής έχουμε τα φράγματα $T \in O(n \log_{3/2} n)$ και $T \in \Omega(n \log_3 n)$ και άρα $T \in \Theta(n \log n)$.

6. Αφού $n^3 \log^2 n \in \Omega(n^{2+\epsilon})$ άρα από το Master Theorem θα είναι $T \in \Theta(n^3 \log^2 n)$

7. Θα είναι:

$$\begin{aligned} T(n) &= \sum_{i=1}^{\lg\lg n} \lg\lg 2^{2^i} \\ &= \sum_{i=1}^{\lg\lg n} \lg 2^i \\ &= \sum_{i=1}^{\lg\lg n} i \\ &\in \Theta((\lg\lg n)^2) \end{aligned}$$

8. Έχουμε:

$$\begin{aligned} T(n) &= T(n-3) + \log n \\ \Rightarrow T(n) &= \sum_{i=0}^{\frac{n}{3}-\frac{1}{3}} \log(3i+1) \\ &= \log\left(\prod_{i=0}^{\frac{n}{3}-\frac{1}{3}} (3i+1)\right) \end{aligned}$$

Συνεπώς είναι:

$$\begin{aligned} T(n) &\in O(\log(n!)) \\ &= O(n \log n) \end{aligned}$$

Επιπλέον είναι:

$$\begin{aligned} T(n) &\in \Omega\left(\log(\frac{n}{3}!)\right) \\ &= \Omega\left(\log\left(\left(\frac{n}{6}\right)^{\frac{n}{6}}\right)\right) \\ &= \Omega(n \log n) \end{aligned}$$

Άρα $T(n) \in \Theta(n \log n)$.

Άσκηση 2

Χρησιμοποιούμε ένα AVL δέντρο για να κρατάμε τα στοιχεία μας. Ξεκινώντας με ένα άδειο δέντρο εισάγουμε όλα τα στοιχεία προσέχοντας να μην εισάγουμε πολλαπλά ταυτόσημα στοιχεία. Αντίθετα, κατά την εισαγωγή ενός ήδη υπάρχοντος στοιχείου, αυξάνουμε ένα μετρητή που δείχνει το πλήθος των εμφανίσεών του. Τέλος, εξάγουμε τα στοιχεία σε αύξουσα σειρά εισάγοντας το κατάλληλο πλήθος στον πίνακα που θα επιστρέψουμε. Καθώς το δέντρο θα περιέχει κάθε στιγμή το πολύ $O(\lg n)$ στοιχεία, οι λειτουργίες εισαγωγής στοιχείου και εξαγωγής ελαχίστου θα είναι $O(\lg \lg n)$.

Algorithm 1 Άσκηση 2

```
1:  procedure FastRepeatedSort(A, N)
2:      t ← EmptyBalancingTree()
3:      for i ← 1 to N do
4:          item ← TreeFind(t, A[i])
5:          if item ≠ ⊥ then
6:              item.count ← item.count + 1
7:          else
8:              TreeInsert(t, key : A[i], count : 1)
9:      j ← 1
10:     while ¬Empty(t) do
11:         item ← ExtractMin(t)
12:         for i ← 1 to item.count do
13:             A[j] ← item.key
14:             j ← j + 1
```

Το κάτω φράγμα στο χρόνο εκτέλεσης του συγκριτικού αλγορίθμου δεν ισχύει καθώς εισάγουμε μία υπόθεση για την κατανομή των στοιχείων με αποτέλεσμα να μπορούμε έτσι να μειώσουμε το πλήθος των πιθανών διατάξεων.

Άσκηση 3

(α)

Αρχικά εντοπίζουμε ένα άνω φράγμα για το πλήθος των στοιχείων του πίνακά μας εξετάζοντας αλλεπάλληλες δυνάμεις του 2. Αυτό επιτυγχάνεται σε χρόνο $\Theta(\log n)$. Στη συνέχεια χρησιμοποιούμε απλή δυαδική αναζήτηση για να εντοπίσουμε αν υπάρχει το εν λόγω στοιχείο σε χρόνο $\Theta(\log\lceil 2^{\log n}\rceil) = \Theta(\log n)$.

Algorithm 2 Άσκηση 3(α)

```
1: procedure SizeUpperBound(A)
2:     high ← 1
3:     while A[high] ≠ ∞ do
4:         high ← 2 × high
5:     return high
Require: sorted(A)
6: procedure BinarySearch(A, key, low, high)
7:     if low ≥ high then
8:         return ⊥
9:     mid ← ⌊(low+high)/2⌋
10:    if key = A[mid] then
11:        return mid
12:    if key < A[mid] then
13:        return BinarySearch(A, key, low, mid)
14:    if key > A[mid] then
15:        return BinarySearch(A, key, mid + 1, high)
Require: sorted(A)
16: procedure Find(A, key)
17:    N ← SizeUpperBound(A)
18:    return BinarySearch(A, key, 1, N)
```

(β)

Ξεκινάμε με δύο τμήματα των πινάκων A και B που περιέχουν σίγουρα το k-οστό στοιχείο. Αυτό σίγουρα θα βρίσκεται στα πρώτα k στοιχεία ενός εκ των δύο. Κρατάμε λοιπόν τα k πρώτα στοιχεία του A και τα k πρώτα στοιχεία του B. Στη συνέχεια, διαλέγουμε το μεσαίο στοιχείο των δύο τμημάτων τα οποία και συγκρίνουμε. Χρησιμοποιώντας αυτή τη σύγκριση μαζί με την μεταβατικότητα

της σύγκρισης και το γεγονός ότι τα δύο τμήματα είναι ταξινομημένα, απορρίπτουμε τον μισό και από τα δύο τμήματα και ενημερώνουμε το k αντίστοιχα βρίσκοντας το μισό του. Συνεχίζοντας αυτή τη διαδικασία αναδρομικά, μετά από $\log k$ επαναλήψεις καταλήγουμε στην περίπτωση όπου τα τμήματά μας έχουν λιγότερα από 1 στοιχείο το καθένα, και συνεπώς το στοιχείο που αναζητούμε είναι ένα από αυτά τα δύο.

Η λύση αυτή χρησιμοποιεί τη μέθοδο «Διαίρει και εξειδίκευε», καθώς κάθε φορά αφαιρεί το μισό κάθε ενός από τα δύο τμήματα που μένουν να εξετάσουμε. Έτσι η πολυπλοκότητά της είναι $\Theta(\log k)$.

Algorithm 3 Άσκηση 3(β)

Require: $sorted(A[l_A \dots l_A + k]), sorted(B[l_B \dots l_B + k])$

procedure KthELEMENT(A, l_A, B, l_B, k)
  $m_A \leftarrow l_A + \lfloor \frac{k-1}{2} \rfloor$
  $m_B \leftarrow l_B + \lfloor \frac{k-1}{2} \rfloor$
  if $k = 1$ **then**
    return $\min(A[l_A], B[l_B])$
  $k \leftarrow \lfloor \frac{k}{2} \rfloor$
  if $A[m_A] < B[m_B]$ **then**
    return KthELEMENT($A, m_A + 1, B, l_B, k$)
  return KthELEMENT($A, l_A, B, m_B + 1, k$)
procedure Kth(A, B, N, k)
  $k \leftarrow \min(N, k)$
  return KthELEMENT($A, 1, B, 1, k$)

Άσκηση 4

Παρατηρούμε ότι το τεύχος που λείπει θα πρέπει να διαφέρει σε τουλάχιστον ένα bit από κάθε τεύχος που έχουμε στην κατοχή μας. Ξεκινάμε βρίσκοντας το πρώτο bit του comicbook που λείπει. Αυτό γίνεται ευκολα κάνοντας xor στο πρώτο bit των τευχών που έχουμε, δηλαδή με $n-1$ ερωτήσεις. Στη συνέχεια, γνωρίζουμε ότι τα τεύχη που έχουν διαφορετικό το πρώτο bit είναι σίγουρα διάφορα από το τεύχος που λείπει. Συνεπώς, απομένει να ελέγξουμε τα τεύχη που έχουν ίδιο το πρώτο bit. Αυτά είναι $\frac{n}{2} - 1$ σε πλήθος. Στη συνέχεια μπορούμε να βρούμε το δεύτερο bit και να διαγράψουμε πάλι τα μισά τεύχη που διαφέρουν, συνεχίζοντας έτσι τη διαδικασία έως ότου γνωρίζουμε την τιμή όλων των bits. Καθώς κάθε φορά που μαθαίνουμε ένα bit αφαιρούμε τα μισά τεύχη που γνωρίζουμε ότι διαφέρουν, το πλήθος των ερωτήσεων δεν θα ξεπεράσει ποτέ το $n + \frac{n}{2} + \frac{n}{4} + \dots = 2n$.

Algorithm 4 Άσκηση 4

```
1: procedure MissingComicbook(n)
2:     candidates ← {x : 0 ≤ x < n − 1}
3:     result ← 0
4:     for j ← log n downto 2 do
5:         x ← 0
6:         candidates_0 ← {}
7:         candidates_1 ← {}
8:         for all i ∈ candidates do
9:             if Ask(i, j) = 1 then
10:                candidates_1 ← candidates_1 ∪ {i}
11:                x ← 1 − x
12:            else
13:                candidates_0 ← candidates_0 ∪ {i}
14:        if x = 1 then
15:            candidates ← candidates_1
16:            result ← result + 2^(j−1)
17:        else
18:            candidates ← candidates_0
19:    result ← result + Ask(⋃ candidates, 1)[1]
20:    return result
```

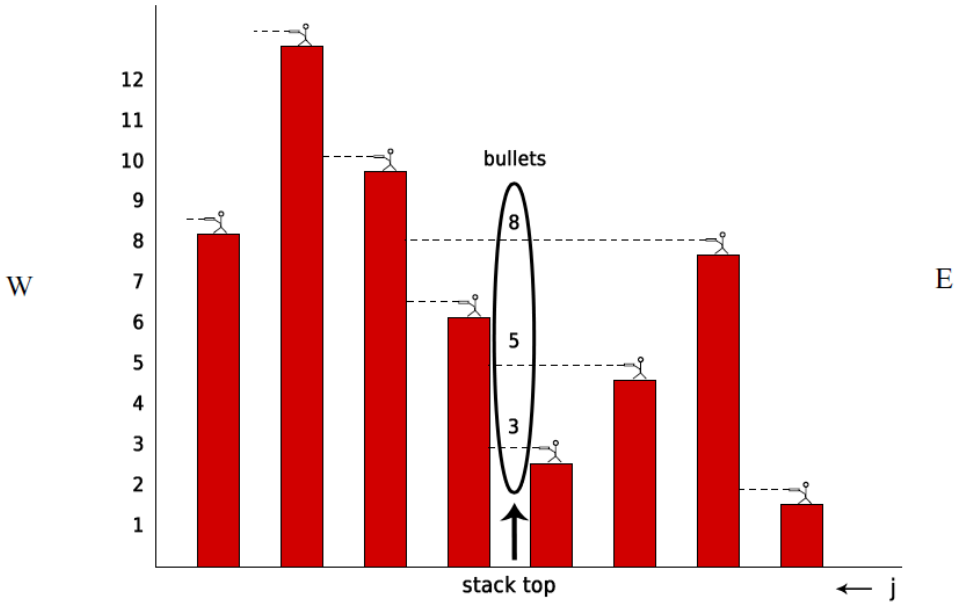
Άσκηση 5

Διασχίζουμε τις πολυκατοικίες από τα ανατολικά προς τα δυτικά. Κρατούμε κάθε στιγμή μία στοίβα που περιέχει ταξινομημένες σε σειρά αύξουσων υψών τις σφαίρες που κινούνται στον αέρα στη θέση που βρισκόμαστε. Φτάνοντας σε μία πολυκατοικία, ελέγχουμε το πρώτο και μικρότερο στοιχείο της στοίβας για να δούμε αν πρόκειται να υπάρξει κάποια σύγκρουση. Αν ναι, τότε ενημερώνουμε τον πίνακα B αφού πλέον γνωρίζουμε ότι η εναέρια σφαίρα χτύπησε την πολυκατοικία που εξετάζουμε. Συνεχίζουμε με τον ίδιο τρόπο έως ότου η στοίβα να αδειάσει ή μέχρι να βρούμε κάποια σφαίρα που βρίσκεται σε μεγαλύτερο ύψος από την πολυκατοικία που εξετάζουμε. Στη συνέχεια προσθέτουμε στην αρχή της στοίβας την σφαίρα που φεύγει από την πολυκατοικία που εξετάζουμε. Η στοίβα υποχρεωτικά θα παραμείνει ταξινομημένη καθώς το πρώτο της στοιχείο πριν την εισαγωγή είναι μεγαλύτερο από το στοιχείο που πρόκειται να εισαχθεί.

Ο αλγόριθμος είναι γραμμικός διότι η συνάρτηση POP τρέχει μία φορά για κάθε σφαίρα το πλήθος των οποίων είναι N. Επιπλέον, γίνεται ένα POP για

[1] Ο τελεστής $\bigcup$ εφαρμοσμένος σε μονοσύνολο επιστρέφει το μοναδικό στοιχείο του μονοσυνόλου.

Σχήμα 1: Ο αλγόριθμος 5 για $j = 4$ πριν την εσωτερική επανάληψη



κάθε επανάληψη της εσωτερικής while πέρα από την πρώτη επανάληψη, συνεπώς το πλήθος των επαναλήψεων είναι καθολικά γραμμικά φραγμένο.

Algorithm 5 Άσκηση 5

```
1:  procedure BUILDINGS(A, N)
2:      B ← [ ]
3:      for i ← N downto 1 do
4:          B[i] ← 0
5:      bullets ← EMPTYSTACK()
6:      for i ← N downto 1 do
7:          while ¬EMPTY(bullets) do
8:              bullet ← TOP(bullets)
9:              if A[bullet] < A[i] then
10:                 B[bullet] ← i
11:                 POP(bullets)
12:             else
13:                 break
14:         PUSH(bullets, i)
15:     return B
```