Πίνακες
Ασκήσεις

Δημήτριος Μάγος

17 Μαρτίου 2023

Γραμμή	#ΣΥΒ	Περιγραφή
4 :	5	Μία πρόσθεση, μία αφαίρεση και τρεις λογικές πράξεις
5 :	5	Κλήση συνάρτησης Swap, μία αφαίρεση $(j-1)$, τρεις εκχωρήσεις μέσα στη συνάρτηση Swap
6 :	2	Μείωση τιμής δείκτη j και εκχώρηση

Πίνακας 1: Αριθμός ΣΥΒ ανά επανάληψη εσωτερικού βρόχου Αλγόριθμου 1

Άσκηση 1. Να υπολογίσετε αναλυτικά την πολυπλοκότητα χειρότερης και καλύτερης περίπτωσης της ενθετικής ταξινόμησης (Αλγόριθμος 1). Να διερευνήσετε για ποια στιγμιότυπα προκύπτει η κάθε περίπτωση.

Αλγόριθμος 1 Ενθετική Ταξινόμηση

Απαιτείται: πίνακας A πίνακας A με στοιχεία στις θέσεις από lo έως hi, όπου $lo \leq hi$.
Επιστρέφεται: Ταξινομημένος πίνακας A

```
1: function Insert_Sort(int A[], int lo, int hi)
2:     for i ← lo + 1; i ≤ hi; i++ do
3:         j ← i;
4:         while j ≥ lo + 1 and A[j] < A[j − 1] do
5:             Swap(A[j], A[j − 1]);
6:             j − −;
7:         end while
8:     end for
9: end function
```

Λύση. Ο αριθμός των στοιχείων του πίνακα δίνεται από τη σχέση

$$n = hi - lo + 1. \tag{1}$$

Ένας αναλυτικός υπολογισμός του αριθμού των ΣΥΒ που εκτελεί η Ενθετική Ταξινόμηση ακολουθεί. Αναλυτικά σε κάθε επανάληψη του εσωτερικού βρόχου - Γραμμές 4-7 - εκτελούνται τα ΣΥΒ που παρουσιάζονται στον Πίνακα 1 Επομένως ο συνολικός αριθμός των ΣΥΒ που εκτελούνται σε κάθε επανάληψη του εσωτερικού βρόχου είναι 12. Επιπλέον θα εκτελεστούν πέντε ΣΥΒ προκειμένου η συνθήκη στη Γραμμή 4 να πάρει την τιμή `False` και ένα ΣΥΒ για την αρχικοποίηση του δείκτη j (Γραμμή 3).
Στη χειρότερη περίπτωση θα εκτελεστεί μία επανάληψη του εσωτερικού βρόχου για κάθε τιμή του δείκτη j ενώ στην καλύτερη περίπτωση δεν θα εκτελεστεί καμία. Επομένως ο αριθμός των ΣΥΒ που εκτελούνται στις Γραμμές 3 έως 7, έστω $A(i)$, βρίσκεται ανάμεσα στα όρια

$$6 \leq A(i) \leq 12(i-2+1)+6 = 12(i-1)+6 = 12i-6. \tag{2}$$

Σε κάθε επανάληψη του εξωτερικού βρόχου (Γραμμή 2) εκτελούνται τρία ΣΥΒ: ένα για την εκτίμηση της τιμής της συνθήκης $i \leq n$ και δύο στη εντολή $i++$. Ο αριθμός των επαναλήψεων είναι ίσος με $n-1$. Επίσης εκτελούνται δύο επιπλέον ΣΥΒ: ένα για την αρχικοποίηση του δείκτη i στην τιμή 2 και ένα για την αποτίμηση της $i \leq n$ σε

`False` προκειμένου να τερματίσει ο αλγόριθμος. Συνολικά ο αριθμός $T(n)$ των ΣΥΒ που εκτελεί η ενθετική ταξινόμηση ικανοποιεί τη σχέση

$$3(n-1)+2+5(n-1) \leq T(n) \leq 3(n-1)+2+\sum_{i=2}^{n}(12i-6) \tag{3}$$

$$\Rightarrow 8(n-1)+2 \leq T(n) \leq \frac{1}{2}(12n^2+5n-12). \tag{4}$$

Η τιμή της $T(n)$ είναι ίση με το άνω φράγμα που προκύπτει από την (4) αν τα στοιχεία του πίνακα A είναι ταξινομημένα σε φθίνουσα σειρά. Επομένως η ενθετική ταξινόμηση έχει χρόνος εκτέλεσης χειρότερης περίπτωσης $\Theta(n^2)$. Στον αντίποδα όταν τα στοιχεία του πίνακα A είναι ήδη ταξινομημένα σε αύξουσα σειρά έχουμε την καλύτερη περίπτωση: τότε η τιμή της $T(n)$ είναι ίση με το κάτω φράγμα της (4). Άμεσα προκύπτει ότι ο χρόνος εκτέλεσης καλύτερης περίπτωσης του Αλγόριθμου 1 είναι τάξης $\Theta(n)$. ■

Άσκηση 2. Να εκπονήσετε αλγόριθμο ο οποίος να δέχεται σαν είσοδο έναν ταξινομημένο πίνακα ακεραίων A και έναν ακέραιο a και να εισάγει στον πίνακα το a σε τέτοια θέση ώστε ο πίνακας που θα προκύπτει να είναι επίσης ταξινομημένος.

Λύση. Θεωρούμε ότι ο ταξινομημένος πίνακας A περιέχει στοιχεία από τις θέσεις 1 ως n. Θα χρησιμοποιήσουμε τη συνάρτηση Exists_in_Group (Σημειώσεις: Αλγόριθμος 15) η οποία θα επιστρέψει τη θέση που βρίσκεται το μικρότερο στοιχείο που είναι μεγαλύτερο-ίσο με το a. Θα βάλουμε το a στη θέση αυτή αφού πρώτα μετακινήσουμε όλα τα στοιχεία από τη θέση αυτή και μετά μία θέση δεξιότερα. Ο Αλγόριθμος 2 κωδικοποιεί την περιγραφόμενη διαδικασία. ■

Αλγόριθμος 2 Εισαγωγή ακέραιου a σε έναν ταξινομημένο πίνακα A

Απαιτείται: Πίνακας A με στοιχεία στις θέσεις από 1 ώς n, ακέραιος a.
Επιστρέφεται: Ταξινομημένος πίνακας A μετά την εισαγωγή του a

```
1: function Sum_Equals_a(int a, int A[], int n)
2:     k ← Search_in_Group(a, A, 1, n);
3:     for i ← n; i ≥ k; i − − do
4:         A[i + 1] ← A[i];
5:     end for
6:     A[k] ← a;
7: end function
```

Άσκηση 3. Δίνεται μια ακολουθία ακεραίων αριθμών οι οποίοι είναι αποθηκευμένοι σε έναν πίνακα A στις θέσεις από 1 ως n. Να εκπονηθεί αλγόριθμος ο οποίος να υπολογίζει το πλήθος των τριάδων των αριθμών που περιέχονται στον πίνακα και το άθροισμα τους είναι ίσο με μηδέν.

Λύση. Μία αφελής προσέγγιση είναι να πάρουμε το άθροισμα κάθε δυνατής τριάδας αριθμών και να το συγκρίνουμε με το μηδέν. Επειδή ο αριθμός των διακριτών τριάδων δίνεται από τον τύπο

$$\binom{n}{3} = \frac{n(n-1)(n-2)}{6},$$

μία τέτοια προσέγγιση οδηγεί σε αλγόριθμο πολυπλοκότητας $O(n^3)$.

Ταξινομώντας την ακολουθία και χρησιμοποιώντας την τεχνική των δύο δεικτών μπορούμε να επιτύχουμε χαμηλότερη πολυπλοκότητα. Συγκεκριμένα ο Αλγόριθμος 3 υπολογίζει το ζητούμενο πλήθος επιτυγχάνοντας πολυπλοκότητα $O(n^2)$. ■

Αλγόριθμος 3 Υπολογισμός πλήθους τριάδων στοιχείων με άθροισμα μηδέν

Απαιτείται: ακέραιος n, πίνακας A με στοιχεία στις θέσεις από 1 ώς n
Επιστρέφεται: Πλήθος τριάδων στοιχείων με άθοοισμα 0

```
1:  function Triads(int A[], int n)
2:      Ταξινόμηση A;
3:      count ← 0;
4:      for k ← 1; k ≤ n; k++ do
5:          i ← k + 1;
6:          j ← n;
7:          while i < j do
8:              diff ← A[k] + A[i] + A[j];
9:              if diff = 0 then
10:                 count++;
11:                 i++; j−−;
12:             else if diff < 0 then
13:                 i++;
14:             else
15:                 j−−;
16:             end if
17:         end while
18:     end for
19:     return count;
20: end function
```

Άσκηση 4. (Τεχνική δύο δεικτών) Δίνεται μια ακολουθία θετικών ακεραίων αριθμών οι οποίοι είναι αποθηκευμένοι σε έναν πίνακα A στις θέσεις από lo ως hi. Ζητείται αλγόριθμος γραμμικού χρόνου που να υπολογίζει το μεγαλύτερο δυνατό αριθμό a, για τον οποίο να υπάρχουν ακέραιοι p, q με την ιδιότητα

$$\sum_{i=lo}^{lo+p-1} A[i] = a = \sum_{j=hi-q+1}^{hi} A[j]$$

και επιπλέον $p + q \leq hi - lo + 1$.
Προσέξτε ότι το πρόβλημα έχει πάντα λύση (για $a = 0$ και $p = q = 0$).

Λύση. Θα χρησιμοποιήσουμε την τεχνική των δύο δεικτών· ο δείκτης i εκκινεί από θέση lo και βαίνει αυξανόμενος ενώ ο δείκτης j από θέση hi και βαίνει μειούμενος. Αν το άθροισμα από τα αριστερά είναι ίσο με το άθροισμα από τα δεξιά αυτή είναι μία πιθανή τιμή για το a - στην περίπτωση αυτή μεταβάλλουμε τον έναν από τους δύο δείκτες. Αν το άθροισμα από αριστερά είναι μεγαλύτερο από το άθροισμα από τα δεξιά τότε μεταβάλλουμε τον δείκτη j ενώ στην αντίθετη περίπτωση μεταβάλλουμε τον δείκτη i. Σε κάθε μία από τις δύο τελευταίες περιπτώσεις αυξάνουμε το αντίστοιχο άθροισμα. Ο Αλγόριθμος 4 κωδικοποιεί τη διαδικασία. ■

Άσκηση 5 (Κυλιόμενο παράθυρο)**.** Δίνεται θετικός φυσικός αριθμός m και μια ακολουθία θετικών ακεραίων αριθμών οι οποίοι είναι αποθηκευμένοι σε έναν πίνακα A

Αλγόριθμος 4 Υπολογισμός μεγαλύτερου αριστερού-δεξιού αθροίσματος

Απαιτείται: Πίνακας A με στοιχεία στις θέσεις από lo ώς hi
Επιστρέφεται: Τιμή μεγαλύτερου αριστερού - δεξιού αθροίσματος a

```
1:  function Left_Right_Sum(int A[], int lo, int hi,)
2:      i ← lo;
3:      j ← hi;
4:      a ← 0;
5:      left_sum ← 0;
6:      right_sum ← 0;
7:      while i < j do
8:          left_sum ← left_sum + A[i];
9:          right_sum ← right_sum + A[j];
10:         A[i] ← 0;
11:         A[j] ← 0;
12:         if left_sum > right_sum then
13:             j − −;
14:         else if left_sum < right_sum then
15:             i + +;
16:         else
17:             a ← left_sum;
18:             i + +;
19:         end if
20:     end while
21:     return a;
22: end function
```

στις θέσεις από 1 ως n. . Ζητείται αλγόριθμος πολυπλοκότητας $O(\max\{n, m\})$ που να υπολογίζει το μέγιστο πολλαπλάσιο του m που μπορεί να προκύψει ως άθροισμα (οσωνδήποτε) διαδοχικών όρων της ακολουθίας.

Λύση. Θα χρησιμοποιήσουμε την τεχνική του κυλιόμενου παραθύρου (δες αντίστοιχη ενότητα στις σημειώσεις) προκειμένου να έχουμε στη διάθεση μας το άθροισμα συνεχόμενων στοιχείων του πίνακα εκκινώντας από τη θέση 1. Στη συνέχεια, για κάθε άθροισμα υπολογίζεται το υπόλοιπο της διαίρεσης με το m· η τιμή αυτή βρίσκεται ανάμεσα στο 0 και στο $m - 1$. Αν το υπόλοιπο είναι ίσο με μηδέν τότε το άθροισμα είναι πολλαπλάσιο του m και επομένως συγκρίνεται με το μεγαλύτερο μέχρι εκείνη τη στιγμή άθροισμα που αποτελεί πολλαπλάσιο του m. Αν το υπόλοιπο, έστω $ypol$, είναι διάφορο του μηδενός τότε πολλαπλάσιο του m είναι η διαφορά

$$sum[i] - sum[j] = A[j + 1] + \cdots + A[i] \tag{5}$$

όπου $sum[i]$ είναι το άθροισμα (από τη θέση 1) στην παρούσα θέση i (παρόν άθροισμα) και $sum[j]$ είναι το άθροισμα (από τη θέση 1) μέχρι τη θέση j όπου η θέση j είναι η πρώτη θέση στην οποία εμφανίστηκε το ίδιο υπόλοιπο (δηλαδή υπόλοιπο ίσο με $sum[i] \bmod m$). Άρα στην περίπτωση αυτή η διαφορά (5) συγκρίνεται με την μέχρι εκείνη τη στιγμή μεγαλύτερο άθροισμα. Ο Αλγόριθμος 5 κωδικοποιεί τη διαδικασία αυτή. ■

Άσκηση 6. Να εκπονήσετε αλγόριθμο ο οποίος με τη χρήση μίας στοίβας Q να τυπώνει τη δυαδική αναπαράσταση ενός θετικού ακεραίου n. Να υπολογίσετε την πολυπλοκότητα του αλγόριθμου.

Λύση. Ο αλγόριθμος αποτελεί μία επαναληπτική διαδικασία όπου σε κάθε επανάληψη θα υπολογίζεται το πηλίκο και το υπόλοιπο του n με το 2, θα προστίθεται το υπόλοιπο στη στοίβα, θα εκχωρείται στη μεταβλητή n το πηλίκο της διαίρεσης. Οι επαναλήψεις θα ολοκληρωθούν όταν το πηλίκο της διαίρεσης θα γίνει ίσο με 0. Στη συνέχεια θα εξαχθούν τα περιεχόμενα της στοίβας τυπώνοντας κάθε αριθμό με τη σειρά που θα εξαχθεί. Ο Αλγόριθμος 6 κωδικοποιεί τη διαδικασία. Παρατηρούμε ότι σε κάθε επανάληψη του βρόχου (Γραμμή 4) εκτελείται σταθερός αριθμός από ΣΥΒ. Επομένως αρκεί να υπολογίσουμε τον αριθμό των επαναλήψεων προκειμένου να βρούμε την πολυπλοκότητα του αλγόριθμου. Η συνθήκη του βρόχου αποτιμάται σε `False` μετά από k επαναλήψεις για τις οποίες ισχύει

$$\lfloor \frac{n}{2^k} \rfloor = 0 \Rightarrow \frac{n}{2^k} < 1 \Rightarrow k > \lg n$$

και επομένως το k είναι ο μικρότερος ακέραιος που είναι αυστηρά μεγαλύτερος από $\lg n$. Συνεπώς, $k = \lfloor \lg n \rfloor + 1$. ■

Άσκηση 7. Να εκπονήσετε αλγόριθμο ο οποίος να δέχεται σαν είσοδο μία στοίβα και να παράγει μία έτερη στοίβα στην κορυφή της οποίας να βρίσκεται το τελευταίο στοιχείο της πρώτης στοίβας και κάθε άλλο στοιχείο της πρώτης στοίβας να βρίσκεται μία θέση παρακάτω στη δεύτερη στοίβα. Για παράδειγμα, αν η στοίβα Q είναι η αρχική στοίβα τότε η Q' είναι η στοίβα που πρέπει να παράγει ο αλγόριθμος σας, όπου

$$Q = \begin{vmatrix} 4 \\ 17 \\ 9 \\ 23 \end{vmatrix}, \; Q' = \begin{vmatrix} 23 \\ 4 \\ 17 \\ 9 \end{vmatrix}.$$

Αλγόριθμος 5 Υπολογισμός μεγαλύτερου αθροίσματος που είναι πολλαπλάσιο του m

Απαιτείται: Ακέραιοι n, m, πίνακας A με στοιχεία στις θέσεις από 1 ώς n
Επιστρέφεται: Τιμή μεγαλύτερου αριστερού - δεξιού αθροίσματος a

```
1: function Max_Mul_Sum(int A[], int n, int m)
2:     max_sum ← 0;
3:     sum[1] ← A[1];
4:     for i ← 2; i ≤ n; i++ do
5:         sum[i] ← sum[i − 1] + A[i];
6:     end for
7:     for i ← 1; i ≤ m − 1; i++ do
8:         first_pos_mod[i] ← −1;
9:     end for
10:    for i ← 1; i ≤ n; i++ do
11:        cur_mod ← sum[i] mod m;
12:        if cur_mod = 0 then
13:            if max_sum < sum[i] then
14:                max_sum ← sum[i];
15:            end if
16:        else
17:            if first_pos_mod[cur_mod] = −1 then
18:                first_pos_mod[cur_mod] ← i;
19:            else
20:                j ← first_pos_mod[cur_mod];
21:                if max_sum < sum[i] − sum[j] then
22:                    max_sum ← sum[i] − sum[j];
23:                end if
24:            end if
25:        end if
26:    end for
27:    return max_sum;
28: end function
```

Αλγόριθμος 6 Δυαδική αναπαράσταση ενός θετικού ακέραιου n

Απαιτείται: ακέραιος $n > 0$
Επιστρέφεται: δυαδική αναπαράσταση του n

```
1: function Bin_Rep(int n)
2:     Δημιουργία στοίβας Q;
3:     size_Q ← 0;
4:     while n > 0 do
5:         Push(Q, size_Q, n mod 2);
6:         n ← ⌊n/2⌋;
7:     end while
8:     while size_Q > 0 do
9:         print Pop(Q, size_Q);
10:    end while
11:    return max_sum;
12: end function
```

Λύση. Έστω Q η στοίβα στην οποία βρίσκονται αποθηκευμένοι οι αριθμοί. Το ζητούμενο θα προκύψει σε δύο φάσεις. Στην πρώτη φάση θα παράγουμε μία ενδιάμεση στοίβα $\hat{Q}$ η οποία θα περιέχει τα στοιχεία της Q σε αντίστροφη σειρά εκκινώντας από το δεύτερο στοιχείο. Στη δεύτερη φάση θα παραχθεί η στοίβα Q' όπου θα περιέχει σε αντίστροφη σειρά τα στοιχεία της $\hat{Q}$ και στο τέλος θα προστεθεί στην Q' το πρώτο στοιχείο της Q. ■

Αλγόριθμος 7 Κυκλική μετακίνηση στοιχείων της στοίβας Q κατά μία θέση

Απαιτείται: στοίβα Q, ακέραιος $size\_Q$ που περιέχει το πλήθος των στοιχείων της Q
Επιστρέφεται: στοίβα Q' η οποία περιέχει τα στοιχεία της Q κυκλικά μετατοπισμένα κατά μία θέση

1: **function** Stack_Rotate(`int` $Q[]$, `int` $size\_Q$)
2: Δημιουργία στοίβας $\hat{Q}$;
3: $first\_element \leftarrow textprocPop(Q, size\_Q)$;
4: Δημιουργία στοίβας $\hat{Q}$;
5: $size\_\hat{Q} \leftarrow 0$;
6: **while** $size\_Q > 0$ **do**
7: Push($\hat{Q}, size\_\hat{Q}$, Pop($Q, size\_Q$));
8: **end while**
9: Δημιουργία στοίβας Q';
10: $size\_Q' \leftarrow 0$;
11: **while** $size\_\hat{Q} > 0$ **do**
12: Push($Q', size\_Q'$, Pop($\hat{Q}, size\_\hat{Q}$));
13: **end while**
14: Push($Q', size\_Q', first\_element$);
15: **end function**