Αριστοτελειο Πανεπιστημιο Θεσσαλονικης
Πολυτεχνικη Σχολη
Τμημα Ηλεκτρολογων Μηχανικων και Μηχανικων Υπολογιστων

Πρώτη Εργασία – Διαίρει & Βασίλευε

TH2802 – Ανάλυση και Σχεδιασμός Αλγορίθμων

Ομάδα 31:

Καπετάνιος Αντώνιος [AEM 10417] (kapetaat@ece.auth.gr)
Κολιαμήτρα, Ελευθερία [AEM 10422] (eleftheak@ece.auth.gr)
Μπαξεβάνης Γεώργιος [AEM 10301] (geompakar@ece.auth.gr)

20 Απριλίου 2023

Περιεχόμενα

Συμβολισμοί

Στην παρούσα αναφορά έχουν υιοθετηθεί οι παρακάτω συμβολισμοί.

Συμβολισμός ψευδογλώσσας	**Ερμηνεία**
a $\leftarrow$ b	καταχώρηση του b στο a
$++$ a	προσαύξηση του a κατά 1
`uint` a	μη αρνητικός ακέραιος αριθμός
`array` a	μία μονοδιάστατη, zero-indexed, διατεταγμένη λίστα στοιχείων, δηλαδή ένα διάνυσμα στοιχείων
arr[a : b]	διαμέριση του πίνακα `arr` στην οποία ανήκουν τα στοιχεία $\{\texttt{arr[a]}, \dots, \texttt{arr[b} - 1]\}$ με τη σειρά που εμφανίζονται στον `arr`
code //comment	ό,τι βρίσκεται δεξιά των // είναι σχόλιο (in-line comment)

Πίνακας 1: Συμβολισμοί σε περιβάλλον ψευδογλώσσας.

Συμβολισμός συναρτήσεων	**Ερμηνεία**
$\lg a$	$\log_2 a$
$\ln a$	$\log_e a$
$\log a$	$\log_{10} a$
$\lceil x \rceil$	συνάρτηση ceiling: $\min\{n \mid n \geqslant x,\ \text{integer } n\}$ [2]
$\lfloor x \rfloor$	συνάρτηση floor: $\max\{n \mid n \leqslant x,\ \text{integer } n\}$ [2]

Πίνακας 2: Συμβολισμοί συνήθων συναρτήσεων.

Πρόβλημα 1

Πρόβλημα 1 *Εν όψει εκλογών, έχουμε δημοσκοπικά δεδομένα για κάθε χωριό και πόλη της περιφέρειας που μας ενδιαφέρει. Για κάθε μία από αυτές τις τοποθεσίες γνωρίζουμε τι δήλωσε ο κάθε πολίτης, που πήρε μέρος στη δημοσκόπηση, πως προτίθεται να ψηφίσει. Θέλουμε να μάθουμε εάν υπάρχει υποψήφιος (και ποιος είναι αυτός) ο οποίος έχει συγκεντρώσει περισσότερες από τις μισές ψήφους σε ένα συγκεκριμένο μέρος.*

Το **Προβλήμα 1** αναφέρεται στο μέγεθος των *μισών ψήφων*, το οποίο μπορεί να ερμηνευθεί είτε ως προς το πλήθος των πολιτών οι οποίοι συμμετείχαν στη δημοσκόπηση, είτε ως προς τον συνολικό πληθυσμό της τοποθεσίας, ανεξαρτήτως του αριθμού των συμμετεχόντων στη δημοσκόπηση. Για την πρώτη ερμηνεία, ως είσοδο χρειαζόμαστε μόνον τον πίνακα των υποψηφίων και τα αποτελέσματα της δημοσκόπησης, ενώ για την δεύτερη ερμηνεία χρειαζόμαστε ως είσοδο μία επιπλέον σταθερά η οποία θα μας πληροφορεί για τον συνολικό πληθυσμό της τοποθεσίας. Στις επόμενες ενότητες υιοθετούμε την πρώτη ερμηνεία. Ωστόσο, ας σημειωθεί πως η προσαρμογή των προτεινόμενων αλγορίθμων στην δεύτερη περίπτωση είναι εύκολη και δεν επηρεάζει τον χρόνο εκτέλεσής τους.

Επίλυση σε $\mathcal{O}(n^2)$

Στην ενότητα αυτή προτείνεται ένας αλγόριθμος επίλυσης του **Προβλήματος 1** ο οποίος θα αποδείξουμε πως έχει χρόνο εκτέλεσης $\mathcal{O}(n^2)$.

Στην πραγματικότητα καλούμαστε να υπολογίσουμε τη συχνότητα εμφάνισης κάθε στοιχείου ενός μονοδιάστατου πίνακα, στον πίνακα αυτόν. Στη συνέχεια, πρέπει να βρούμε – εάν υπάρχουν – τα στοιχεία των οποίων η συχνότητα εμφάνισης πληροί ένα συγκεκριμένο κριτήριο. Στην περίπτωση του **Προβλήματος 1** ο πίνακας αυτός είναι τα αποτελέσματα της δημοσκόπησης και η συχνότητα εμφάνισης κάθε υποψηφίου ισοδυναμεί με τις ψήφους που έχει καταφέρει να συγκεντρώσει στη συγκεκριμένη τοποθεσία. Τέλος, πρέπει να βρούμε εάν υπάρχει και ποιος είναι αυτός ο υποψήφιος ο οποίος έχει συγκεντρώσει περισσότερες από τις μισές ψήφους.

Algorithm 1 An algorithm of time complexity $\mathcal{O}(n^2)$ that checks whether or not any of the candidates has gathered more than half of the votes. The following procedure takes as input an array C containing the names of the candidates and an array P containing the vote – according to the gallop – of every citizen.

```
1: procedure MAIN(array C, array P)
2:     array POP_CAND ← ∅    //empty array
3:     for all candidate ∈ C do
4:         popularity ← 0
5:         for all citizen ∈ P do
6:             if (P[citizen] == candidate) then
7:                 ++popularity
8:             end if
9:         end for
10:        if (popularity > ⌈P.length()/2⌉) then
11:            POP_CAND.append(candidate)
12:            return POP_CAND
13:        end if
14:    end for
15:    return POP_CAND
16: end procedure
```

Για την παραπάνω διαδικασία προτείνουμε τον ακόλουθο αλγόριθμο. Για κάθε έναν υποψήφιο εκτελούμε τα παρακάτω βήματα. Πρώτον αρχικοποιούμε το πλήθος των ψήφων του εκάστοτε υποψηφίου σε 0 (μηδέν). Έπειτα, για τον εκάστοτε υποψήφιο, διαβάζουμε τον πίνακα των αποτελεσμάτων της δημοσκόπησης και κάθε φορά που συναντάμε το όνομά του προσαυξάνουμε το πλήθος των ψήφων του κατά 1. Στη συνέχεια, ελέγχουμε εάν έχει συγκεντρώσει περισσότερες από τις μισές ψήφους. Εάν ναι, προσθέτουμε το ονοματεπώνυμό του σε έναν – έως τώρα κενό – μονοδιάστατο πίνακα τον οποίο και επιστρέφουμε και τερματίζουμε τη διαδικασία. Ειδάλλως, συνεχίζουμε στον επόμενο υποψήφιο. Στην περίπτωση κατά την οποία δεν πληροί κανένας υποψήφιος το παραπάνω κριτήριο επιστρέφουμε έναν κενό πίνακα. Όλα τα παραπάνω δίνονται σε μορφή ψευδογλώσσας στο **Algorithm 1**.

Θα αναλύσουμε τον χρόνο εκτέλεσης του παραπάνω αλγορίθμου με τη βοήθεια της υλοποίησης σε ψευδογλώσσα. Έστω πως υπάρχουν M υποψήφιοι και N πολίτες συμμετείχαν στη δημοσκόπηση. Η γραμμή 2 έχει

σταθερό χρόνο εκτέλεσης $\mathcal{O}(1)$. Το σώμα της **for** (γραμμές 4–12) εκτελείται το πολύ M φορές. Η γραμμή 4 τρέχει σε σταθερό χρόνο $\mathcal{O}(1)$. Ο έλεγχος της συνθήκης της γραμμής 6 πραγματοποιείται $M \cdot N$, το πολύ, φορές και εκτελείται πάντα σε σταθερό χρόνο $\mathcal{O}(1)$. Ομοίως, η γραμμή 7 είναι και αυτή σταθερής χρονικής πολυπλοκότητας και εκτελείται το πολύ $M \cdot N$ φορές. Παρόμοια, η συνθήκη της γραμμής 10 εκτελείται M, το πολύ, φορές σε σταθερό χρόνο $\mathcal{O}(1)$ και οι γραμμές 11 και 12 εκτελούνται το πολύ μία φορά. Τέλος, η γραμμή 15 εκτελείται μια, το πολύ, φορά σε σταθερό χρόνο $\mathcal{O}(1)$. Επομένως, για τον χρόνο εκτέλεσης του **Algorithm 1**, T_1, έχουμε

$$T_1(N, M) = \mathcal{O}(1) + M \cdot \{\mathcal{O}(1) + N \cdot [\mathcal{O}(1) + \mathcal{O}(1)] + \mathcal{O}(1) + \mathcal{O}(1) + \mathcal{O}(1)\} + \mathcal{O}(1)$$

δηλαδή

$$T_1(N, M) = \mathcal{O}(M \cdot N). \tag{1}$$

Χωρίς βλάβη της γενικότητας θέτουμε $n = \max\{M, N\}$. Τότε, η έκφραση (1) του χρόνου εκτέλεσης του Algorithm 1 γίνεται

$$T_1(n) = \mathcal{O}(n^2) \tag{2}$$

■

Επίλυση σε $\mathcal{O}(n \lg n)$ χρήσει *Διαίρει & Βασίλευε*

Στην παρούσα ενότητα επιλύουμε το **Πρόβλημα 1** προτείνοντας έναν αλγόριθμο χρονικής πολυπλοκότητας $\mathcal{O}(n \lg n)$ ο οποίος έχει σχεδιαστεί ακολουθώντας την πρακτική *Διαίρει & Βασίλευε*. Παρακάτω, αναλύουμε τον αλγόριθμο αυτόν, προτείνουμε μία υλοποίησή του σε ψευδογλώσσα και αποδεικνύουμε πως όντως είναι χρονικής πολυπλοκότητας $\mathcal{O}(n \lg n)$.

Ο αλγόριθμος που προτείνουμε ακολουθεί το σύνηθες μοτίβο των αλγορίθμων της λογικής *Διαίρει & Βασίλευε*. Ως είσοδο δέχεται έναν πίνακα με τα ονόματα όλων των υποψηφίων C (με πληθάριθμο M) και έναν δεύτερο πίνακα P (με πληθάριθμο N), κάθε στοιχείο του οποίου αντιπροσωπεύει την ψήφο ενός πολιτη (η ψήφος καταγράφεται με το ονοματεπώνυμο του υποψηφίου). Ως έξοδο, επιστρέφουμε έναν πίνακα ενός στοιχείου ο οποίος περιέχει το όνομα – αν υπάρχει – του υποψηφίου που έχει συγκεντρώσει περισσότερες από τις μισές ψήφους. Ειδάλλως, επιστρέφεται ένας κενός πίνακας.

Αρχικά, δημιουργούμε μία mutable δομή D στην οποία αποθηκεύουμε δυάδες κλειδιών *(keys)* με τις αντίστοιχες τιμές τους *(values)*, παρόμοια με ένα hashtable ή με ένα dictionary της γλώσσας Python [5]. Μία τέτοια δομή επιτρέπει την αναζήτηση ενός κλειδιού (και την πρόσβαση στην τιμή που συνοδεύει το κλειδί) σε σταθερό χρόνο $\mathcal{O}(1)$ [1] [2]. Σκοπός μας είναι να έχουμε πρόσβαση στο πλήθος των ψήφων ενός υποψηφίου σε σταθερό χρόνο $\mathcal{O}(1)$. Συνεπώς, τα κλειδιά στην δομή αυτή θα είναι τα ονόματα των υποψηφίων και η τιμή θα αντιπροσωπεύει το πλήθος των ψήφων που έχει συγκεντρώσει ο εκάστοτε υποψήφιος. Επομένως, για όλα τα κλειδιά, η αρχική τιμή θα είναι μηδέν.

Εν συνεχεία, δημιουργούμε έναν κενό πίνακα POP_CAND στον οποίο, εάν κάποιος υποψήφιος έχει συγκεντρώσει περισσότερες από τις μισές ψήφους θα προσθέσουμε το όνομά του. Στο τέλος της διαδικασίας θα επιστρέψουμε τον πίνακα αυτόν είτε είναι κενός, είτε περιέχει ένα (μόνο) ονοματεπώνυμο.

Έπειτα, ξεκινάμε την διαδικασία προσμέτρησης των ψήφων. Ελέγχουμε εάν το μήκος του πίνακα των ψήφων είναι γνησίως μεγαλύτερο του μηδενός. Εάν ο έλεγχος ήταν αληθής, πραγματοποιούμε έναν δεύτερο έλεγχο· εάν το μήκος του πίνακα, δηλαδή το πλήθος των στοιχείων του, είναι ακριβώς ίσο με το 1 ή εάν είναι μεγαλύτερο του 1. Τότε, έχουμε τις εξής περιπτώσεις:

i. **το πλήθος των στοιχείων είναι** $N > 1$. Υποθέτουμε πως το τρέχον στιγμιότυπο της διαδικασίας προσμέτρησης των ψήφων είναι το F. Τότε, δημιουργούμε έναν *νέο* πίνακα $\mathrm{P}_{\texttt{left}}$ ο οποίος αποτελεί διαμέριση του αρχικού πίνακα από το index 0 έως και $\lceil N/2 \rceil - 1$, δηλαδή θα είναι $\mathrm{P}_{\texttt{left}} = \mathrm{P}\,[0 : \lceil N/2 \rceil]$ και επαναλαμβάνουμε αναδρομικά τη διαδικασία προσμέτρησης ψήφων έχοντας ως είσοδο τον *νέο* υποπίνακα $\mathrm{P}_{\texttt{left}}$. Μόλις επιστρέψουμε στο στιγμιότυπο F της διαδικασίας, δημιουργούμε και πάλι έναν *νέο* πίνακα $\mathrm{P}_{\texttt{right}}$ ο οποίος αποτελεί διαμέριση του αρχικού πίνακα από το index $\lceil N/2 \rceil$ έως και $N - 1$, δηλαδή θα είναι $\mathrm{P}_{\texttt{right}} = \mathrm{P}\,[\lceil N/2 \rceil : N]$ και επαναλαμβάνουμε αναδρομικά τη διαδικασία προσμέτρησης ψήφων έχοντας ως είσοδο τον *νέο* υποπίνακα $\mathrm{P}_{\texttt{right}}$. Επιστρέφοντας στο στιγμιότυπο F της διαδικασίας δεν μένει παρά να επιστρέψουμε στην όποια διαδικασία δημιούργησε το στιγμιότυπο F.

[1] *From a practical standpoint, the most important hash technique invented in the late 1970s is probably the method that Witold Lipski called* ***linear hashing****. […]*

In the theoretical realm, more complicated methods have been devised by which it is possible to guarantee $\mathcal{O}(1)$ *maximum time per access, with* $\mathcal{O}(1)$ *average amortized time per insertion and deletion, regardless of the keys being examined; moreover, the total storage used at any time is bounded by a constant times the number of items currently present, plus another additive constant.* [4]

[2] Η χρήση μίας τέτοιας δομής αν και εξυπηρετεί την μείωση του χρόνου εκτέλεσης, επιβαρύνει την χωρική πολυπλοκότητα του αλγορίθμου. Δηλαδή, απαιτείται δέσμευση περισσότερης μνήμης.

ii. **το πλήθος των στοιχείων είναι** $N = 1$. Υποθέτουμε πως το τρέχον στιγμιότυπο της διαδικασίας προσμέτρησης των ψήφων είναι το F. Τότε, αυξάνουμε την τιμή *(value)* που αντιστοιχεί στο κλειδί *(key)* P[0] της δομής D, κατά 1. Έπειτα, επιστρέφουμε στη διαδικασία η οποία δημιούργησε το F.

Μόλις τελειώσει η διαδικασία προσμέτρησης ψήφων, στη δομή D θα είναι αποθηκευμένο το πλήθος των ψήφων που συγκέντρωσε κάθε υποψήφιος. Στη συνέχεια, για κάθε έναν υποψήφιο ελέγχουμε – μέσω της δομής D – εάν οι ψήφοι που έχει συγκεντρώσει είναι περισσότερες από τις μισές. Εάν ναι, προσθέτουμε το ονοματεπώνυμό του στον πίνακα POP_CAND, επιστρέφουμε τον πίνακα και τερματίζουμε τη διαδικασία. Διαφορετικά, συνεχίζουμε στον επόμενο υποψήφιο. Εάν κανείς δεν έχει συγκεντρώσει παραπάνω από τις μισές ψήφους, επιστρέφουμε τον κενό πίνακα POP_CAND και τερματίζουμε τη διαδικασία.

Συνοψίζοντας, διαιρέσαμε αναδρομικά το αρχικό πρόβλημα μεγέθους N σε δύο υποπροβλήματα ίσου μεγέθους (± 1) έως ότου να εκφυλιστεί σε N προβλήματα μεγέθους 1. Όλα τα παραπάνω, σε μορφή ψευδοκώδικα, δίνονται στο **Algorithm 2**, όπου ξεκινάμε από την διαδικασία MAIN στη γραμμή 18 και η διαδικασία FREQUENCY αντιστοιχεί στην διαδικασία προσμέτρησης των ψήφων.

Algorithm 2 An algorithm of time complexity $\mathcal{O}(n \lg n)$ that checks whether or not any of the candidates has gathered more than half of the votes. The procedure MAIN takes as input an array C containing the names of the candidates and an array P containing the vote – according to the gallop – of every citizen. The procedure FREQUENCY takes as input a dictionary D and an array P containing votes.

```
1: procedure FREQUENCY(dictionary D, array P)
2:     if (P.length() > 0) then
3:         len ← P.length()
4:         if (len > 1) then
5:             mid ← ⌈(len−1)/2⌉
6:             left ← P[0 : mid]
7:             FREQUENCY(D, left)
8:             right ← P[mid : len]
9:             FREQUENCY(D, right)
10:        else if (len == 1) then
11:            ++D[P[0]]  //access the tuple ⟨P[0], value⟩ of D
                          and increment the value field by 1.
12:        end if
13:    else
14:        return ERROR
15:    end if
16:    return
17: end procedure

18: procedure MAIN(array C, array P)
19:    dictionary D : ⟨string key, uint value⟩ ← ∅
20:    for all candidate ∈ C do
21:        D.append ⟨candidate, 0⟩
22:    end for
23:    array POP_CAND ← ∅
24:    FREQUENCY(D, P)
25:    half_v ← ⌈P.length()/2⌉
26:    for all candidate ∈ C do
27:        if (D[candidate] > half_v) then
28:            POP_CAND.append(candidate)
29:            return POP_CAND
30:        end if
31:    end for
32:    return POP_CAND
33: end procedure
```

Σε ό,τι αφορά τη χρονική πολυπλοκότητα του **Algorithm 2** θα έχουμε: η χρονική πολυπλοκότητα κάθε μίας από τις γραμμές 19, 23 και 25 είναι σταθερού χρόνου, δηλαδή $\mathcal{O}(1)$. Η γραμμή 20 και το σώμα της **for** εκτελούνται M φορές – κάθε φορά για ένα διαφορετικό στοιχείο του C – και η γραμμή 20 εκτελείται μία επιπλέον φορά, όπου πλέον δεν υπάρχουν στοιχεία του C τα οποία δεν έχουν ελεγχθεί και έτσι ο

program counter προχωρά στην γραμμή 23. Επομένως, η χρονική πολυπλοκότητα των γραμμών 20–22 είναι $\mathcal{O}(M+1)$, δηλαδή $\mathcal{O}(M)$. Για την γραμμή 24 υποθέτουμε χρόνο εκτέλεσης $T_F(N,M)$. Για την **for** της γραμμής 26 και το σώμα της, με την ίδια ανάλυση που έγινε για την **for** της γραμμής 20, καταλήγουμε σε $T(M) = M \cdot [\mathcal{O}(1) + \mathcal{O}(1) + \mathcal{O}(1)] = \mathcal{O}(M)$. Για την γραμμή 32 είναι $\mathcal{O}(1)$. Συνεπώς, ο χρόνος εκτέλεσης του **Algorithm 2**, T_2, υπολογίζεται ως $T_2(N,M) = \mathcal{O}(1)+\mathcal{O}(M)+\mathcal{O}(1)+T_F(N,M)+\mathcal{O}(1)+\mathcal{O}(M)+\mathcal{O}(1)$. Δηλαδή, θα είναι

$$T_2(N,M) = T_F(N,M) + \mathcal{O}(M) \tag{3}$$

Επόμενο βήμα είναι ο υπολογισμός του χρόνου εκτέλεσης της διαδικασίας FREQUENCY (γραμμές 1–17), $T_F(N,M)$. Το βασικό ζήτημα είναι ο προσδιορισμός της χρονικής πολυπλοκότητας των αναδρομών. Δύο τρόποι προσέγγισης, δεδομένου πως ο αλγόριθμος ακολουθεί τη σχεδιαστική αρχή *Διαίρει & Βασίλευε*, είναι το **Master Theorem** και το **δένδρο αναδρομών** (recursion tree). Αρχικά παραθέτουμε το Master Theorem [1][7].

Έστω ένα πρόβλημα μεγέθους n το οποίο προσεγγίζεται επιλύοντας αναδρομικά a υποπροβλήματα μεγέθους n/b και συνδυάζοντας τις επιμέρους λύσεις σε $\mathcal{O}(n^d)$ χρόνο, για κάποια $a, b, d > 0$. Ο χρόνος εκτέλεσης είναι

$$T(n) = aT\left(\left\lceil \frac{n}{b} \right\rceil\right) + \mathcal{O}(n^d). \tag{4}$$

Θεώρημα 1 (Master Theorem) *Εάν είναι $T(n) = aT\left(\left\lceil \frac{n}{b} \right\rceil\right) + \mathcal{O}(n^d)$ για κάποιες σταθερές $a > 0$, $b > 1$ και $d \geqslant 0$, τότε*

$$T(n) = \begin{cases} \mathcal{O}(n^d) & \text{if } d > \log_b a \\ \mathcal{O}(n^d \lg n) & \text{if } d = \log_b a \\ \mathcal{O}(n^{\log_b a}) & \text{if } d < \log_b a. \end{cases} \tag{5}$$

Στο **Algorithm 2**, το πρόβλημα μεγέθους N το διαμελίζουμε σε δύο υποπροβλήματα μεγέθους $N/2$. Δηλαδή, με όρους του Master Theorem, $a = 2$ και $b = 2$. Οι γραμμές 2, 3, 4, 5, 6, 8 αφορούν την διάσπαση του προβλήματος στα επιμέρους υποπροβλήματα και κάθε εκτέλεσή τους γίνεται σε σταθερό χρόνο $\mathcal{O}(1)$. Οι γραμμές 10 και 11 αφορούν τη διαδικασία που λαμβάνει χώρα στο βαθύτερο επίπεδο της αναδρομής και κάθε εκτέλεσή τους πραγματοποιείται σε επίσης σταθερό χρόνο $\mathcal{O}(1)$. Ο συνδυασμός των λύσεων των υποπροβλημάτων είναι στην πραγματικότητα η εντολή **return** της γραμμής 16 κάθε εκτέλεση της οποίας κοστίζει $\mathcal{O}(1)$ χρόνο καθώς η μόνη της δουλειά είναι να επιστρέψει στην διαδικασία από την οποία κλήθηκε το τρέχον στιγμιότυπο της FREQUENCY. Δηλαδή, ο συνδυασμός όλων των λύσεων πραγματοποιείται σε χρόνο $N \cdot \mathcal{O}(1) = \mathcal{O}(N)$. Συνεπώς, είναι $d = 1 = \log_2 2$, δηλαδή ο χρόνος εκτέλεσης της διαδικασίας FREQUENCY [3] είναι

$$T_F(N) = \mathcal{O}(N \lg N). \tag{6}$$

Συνδυάζοντας τις εξισώσεις (3) και (6) προκύπτει

$$T_2(N,M) = \mathcal{O}(N \lg N) + \mathcal{O}(M).$$

Χωρίς βλάβη της γενικότητας θέτουμε $n = \max\{N, M\}$. Τότε, η προηγούμενη έκφραση του χρόνου εκτέλεσης του **Algorithm 2** γίνεται $T_2(n) = \mathcal{O}(n \lg n) + \mathcal{O}(n)$, δηλαδή

$$T_2(n) = \mathcal{O}(n \lg n). \tag{7}$$

■

Θα αποδείξουμε την χρονική πολυπλοκότητα της υπορουτίνας FREQUENCY, $T_F(N)$, και χρήσει του δένδρου αναδρομών. Αρχικά παραθέτουμε κάποιους χρήσιμους ορισμούς και θεωρήματα σχετικά με τα *δένδρα με ρίζα*.

Ορισμός 1 (Επίπεδο ή βάθος) *Το **επίπεδο** ή **βάθος** μιας κορυφής v είναι η απόστασή της από την ρίζα. Η ρίζα του δένδρου έχει βάθος* 0. [3]

Ορισμός 2 (Υψος) *Το **ύψος** ενός δένδρου με ρίζα είναι το μήκος του μεγαλύτερου μονοπατιού από την ρίζα προς κάθε άλλη κορυφή ή διαφορετικά το μέγιστο των επιπέδων του δένδρου.* [6]

[3] Στην εξίσωση (3) κάναμε την γενική υπόθεση πως ο χρόνος T_F είναι συνάρτηση των N και M. Τελικά, μετά την μελέτη της υπορουτίνας FREQUENCY, καταλήγουμε πως ο χρόνος εκτέλεσης T_F είναι συνάρτηση μόνον του N.

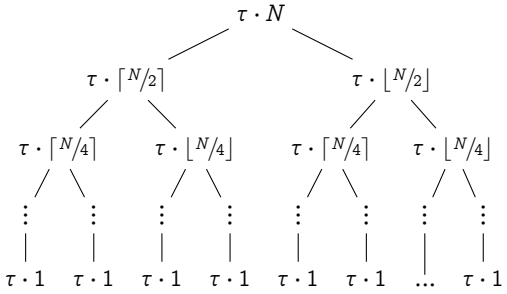
Ορισμός 3 (*m*-αδικό δένδρο) *Ένα δένδρο με ρίζα καλείται* ***m*-αδικό δένδρο***, εάν κάθε εσωτερική κορυφή δεν έχει περισσότερα από m παιδιά.* [6]

Ορισμός 4 (Γεμάτο *m*-αδικό δένδρο) *Ένα δένδρο με ρίζα καλείται* ***γεμάτο m-αδικό δένδρο***, *εάν κάθε εσωτερική κορυφή έχει ακριβώς m παιδιά* [6]

Ορισμός 5 (Ισοζυγισμένο δένδρο) *Ένα δυαδικό δένδρο είναι* ***ισοζυγισμένο***, *εάν για κάθε ακμή, το πλήθος των κορυφών του δεξιού και του αριστερού υποδένδρου του διαφέρουν το πολύ κατά ένα.* [3]

Θεώρημα 2 *Για ένα m-αδικό δένδρο με ύψος h και l φύλλα ισχύει $h \geqslant \lceil \log_m l \rceil$. Εάν το δένδρο είναι γεμάτο m-αδικό και ισοζυγισμένο, τότε είναι $h = \lceil \log_m l \rceil$.* [6]

Όπως αναφέρθηκε παραπάνω, κάθε μία από τις γραμμές 2, 3, 4, 5, 6, 8, 10, 11 και 16 εκτελείται σε $\mathcal{O}(1)$ χρόνο. Επομένως, η διαίρεση του προβλήματος στα επιμέρους προβλήματα και ο συνδυασμός των επιμέρους λύσεων γίνονται σε σταθερό χρόνο. Σε σταθερό χρόνο εκτελείται και η καταμέτρηση της ψήφου (όταν, δηλαδή, το μέγεθος του υποπροβλήματος είναι 1). Υποθέτουμε πως ο σταθερός χρόνος που απαιτείται για τις παραπάνω διαδικασίες είναι τ. Τότε, προκύπτει ένα δένδρο αναδρομών όπως του **Σχήματος 1**.



Σχήμα 1: Δένδρο αναδρομών της διαδικασίας `FREQUENCY`.

Στην γενική περίπτωση, το εν λόγω δένδρο αναδρομών δεν θα είναι πάντα ισοζυγισμένο αλλά θα είναι πάντα γεμάτο (λόγω του τρόπου κατασκευής του). Ωστόσο, εφ' όσον μελετάμε τη χρονική πολυπλοκότητα στην χειρότερη περίπτωση, θεωρούμε πως για δεδομένο μέγεθος αρχικού προβλήματος, θα είναι ισοζυγισμένο. Έτσι, λοιπόν, το δένδρο έχει $\lceil \lg N \rceil + 1$ επίπεδα (θεώρημα 2) και N φύλλα. Κάθε επίπεδο L συνεισφέρει στον συνολικό χρόνο εκτέλεσης κατά $2^L \cdot [\tau \cdot (N/2^L)] = \tau \cdot N$. Άρα, ο συνολικός χρόνος εκτέλεσης της διαδικασίας `FREQUENCY` είναι

$$\begin{aligned} T_F(N) &= \mathcal{O}\left(\tau \cdot N \cdot (\lceil \lg N \rceil + 1)\right) \\ &= \mathcal{O}\left(\tau \cdot N \cdot \lceil \lg N \rceil + \tau \cdot N\right) \end{aligned}$$

δηλαδή

$$T_F(N) = \mathcal{O}\left(N \lg N\right). \tag{8}$$

Συνδυάζοντας τις εξισώσεις (3) και (8) έχουμε

$$T_2(N, \mathrm{M}) = \mathcal{O}\left(N \lg N\right) + \mathcal{O}\left(M\right). \tag{9}$$

Τέλος, χωρίς βλάβη της γενικότητας, θέτουμε και πάλι $n = \max\{N, M\}$ και η (9) γράφεται

$$T_2(n) = \mathcal{O}\left(n \lg n\right) \tag{10}$$

■

Επίλυση σε $\mathcal{O}(n)$

Για την επίλυση του **Προβλήματος 1** σε χρόνο $\mathcal{O}(n)$ θα στηριχτούμε στον αλγόριθμο της προηγούμενης ενότητας. Συγκεκριμένα, θα χρησιμοποιήσουμε και πάλι μια δομή dictionary μέσω της οποίας θα επιτυγχάνουμε πρόσβαση στις ψήφους κάθε υποψηφίου σε, κατά μέσο όρο, χρόνο $\mathcal{O}(1)$ [4]. Ο αλγόριθμος της παρούσης ενότητας διαφοροποιείται από αυτόν της προηγούμενης στην μέθοδο της καταμέτρησης των ψήφων. Προηγουμένως, η καταμέτρηση έγινε αναδρομικά, ενώ τώρα θα πραγματοποιηθεί με σειριακή προσπέλαση του πίνακα των αποτελεσμάτων της δημοσκόπησης. Έχοντας, λοιπόν, αυτό το σκαρίφημα προχωράμε στην αναλυτική περιγραφή του αλγορίθμου.

Ως είσοδο έχουμε έναν πίνακα `C`, M στοιχείων, στον οποίο είναι αποθηκευμένα τα ονοματεπώνυμα όλων των υποψηφίων και έναν πίνακα `P`, N στοιχείων, ο οποίος περιέχει τις ψήφους των πολιτών που συμμετείχαν στη δημοσκόπηση. Ως έξοδος επιστρέφεται είτε ένας κενός πίνακας, είτε ένας πίνακας με το ονοματεπώνυμο του υποψηφίου ο οποίος έχει συγκεντρώσει περισσότερες από τις μισές ψήφους.

Πρώτο βήμα είναι η δημιουργία μίας κενής mutable δομής dictionary, D, στην οποία θα αποθηκεύονται συμβολοσειρές ως κλειδιά *(keys)* και μη αρνητικοί ακέραιοι ως τιμές *(values)*. Τον ρόλο των κλειδιών, στην προκειμένη, τον έχουν τα ονοματεπώνυμα των υποψηφίων και τα πεδία των τιμών αντιπροσωπεύουν τις ψήφους που έχει συγκεντρώσει ο κάθε υποψήφιος. Επομένως, καταχωρούμε στη δομή D κάθε υποψήφιο του πίνακα υποψηφίων C ως κλειδί και αρχικοποιούμε την τιμή που του αντιστοιχεί σε μηδέν (0). Έπειτα, δημιουργούμε έναν κενό πίνακα POP_CAND τον οποίο θα επιστρέψουμε στο τέλος του αλγορίθμου.

Επόμενο βήμα είναι η σειριακή προσπέλαση του πίνακα των αποτελεσμάτων της δημοσκόπησης, P. Κάθε ονοματεπώνυμο που είναι αποθηκευμένο στον πίνακα P, το αναζητούμε ως κλειδί στο dictionary D και προσαυξάνουμε το πεδίο της τιμής *(value)* του κατά ένα. Με το πέρας της διαδικασίας αυτής, σε κάθε κλειδί–υποψήφιο αντιστοιχεί μία μη αρνητική τιμή–πλήθος ψήφων.

Algorithm 3 An algorithm of time complexity $\mathcal{O}(n)$ that checks whether or not any of the candidates has gathered more than half of the votes. The procedure `MAIN` takes as input an array `C` containing the names of the candidates and an array `P` containing the vote – according to the gallop – of every citizen.

```
1: procedure MAIN(array C, array P)
2:     dictionary D : ⟨string key, uint value⟩ ← ∅
3:     for all candidate ∈ C do
4:         D.append ⟨candidate, 0⟩
5:     end for
6:     array POP_CAND ← ∅
7:     for all citizen ∈ P do
8:         ++D[P[citizen]]
9:     end for
10:    half_v ← ⌈P.length() / 2⌉
11:    for all candidate ∈ C do
12:        if (D[candidate] > half_v) then
13:            POP_CAND.append(candidate)
14:            return POP_CAND
15:        end if
16:    end for
17:    return POP_CAND
18: end procedure
```

Εν συνεχεία, προσπελάζουμε το dictionary D μέσω των κλειδιών του. Για κάθε κλειδί ελέγχουμε εάν η τιμή που του αντιστοιχεί είναι μεγαλύτερη του $\lceil N/2 \rceil$. Εάν ναι, το όνομα του υποψηφίου–κλειδί προστίθεται στον πίνακα POP_CAND ο οποίος επιστρέφεται και τερματίζεται ο αλγόριθμος. Εάν όχι, προχωράμε στο επόμενο κλειδί. Τέλος, εάν δεν βρεθεί κάποιος υποψήφιος ο οποίος να έχει συγκεντρώσει περισσότερες από τις μισές ψήφους, δηλαδή εάν η τιμή που του αντιστοιχεί στο D δεν είναι μεγαλύτερη του $\lceil N/2 \rceil$, τότε επιστρέφεται ο κενός πίνακας POP_CAND και τερματίζεται η διαδικασία. Ο αλγόριθμος δίνεται σε μορφή ψευδογλώσσας στο **Algorithm 3**.

Επόμενο βήμα είναι να αναλύσουμε την χρονική πολυπλοκότητα του **Algorithm 3** και να αποδείξουμε πως είναι όντως $\mathcal{O}(n)$.

Η γραμμή 2 τρέχει μία φορά και σε σταθερό χρόνο $\mathcal{O}(1)$ καθότι δημιουργείται κενό dictionary. Το σώμα της **for** της γραμμής 3 θα εκτελεστεί ακριβώς M φορές, κάθε φορά σε σταθερό χρόνο $\mathcal{O}(1)$ [4]. Συνεπώς, η πολυπλοκότητά του είναι $\mathcal{O}(M)$. Η γραμμή 6 εκτελείται μία φορά και σε σταθερό χρόνο $\mathcal{O}(1)$. Το σώμα της **for** της γραμμής 7 θα εκτελεστεί ακριβώς M φορές, κάθε φορά σε σταθερό χρόνο $\mathcal{O}(1)$. Επομένως, ο συνολικός χρόνος εκτέλεσής του θα είναι $\mathcal{O}(M)$. Η γραμμή 10 εκτελείται μία φορά και σε σταθερό χρόνο $\mathcal{O}(1)$. Η γραμμή 12 εκτελείται M, το πολύ, φορές και κάθε φορά σε σταθερό χρόνο $\mathcal{O}(1)$. Σε σταθερό χρόνο τρέχουν και οι γραμμές 13 και 14 οι οποίες δύναται να εκτελεστούν μία, το πολύ, φορά. Τέλος, η γραμμή 17 θα εκτελεστεί μία, το πολύ, φορά και σε σταθερό χρόνο $\mathcal{O}(1)$. Βάσει των παραπάνω, ο συνολικός χρόνος εκτέλεσης του **Algorithm 3** θα είναι

$$\begin{aligned} T_3(N, M) &= \mathcal{O}(1) + M \cdot \mathcal{O}(1) + \mathcal{O}(1) + N \cdot \mathcal{O}(1) + \mathcal{O}(1) + M \cdot [\mathcal{O}(1) + \mathcal{O}(1) + \mathcal{O}(1)] + \mathcal{O}(1) \\ &= \mathcal{O}(M) + \mathcal{O}(N). \end{aligned}$$

Χωρίς βλάβη της γενικότητας θέτουμε $n = \max\{M, N\}$. Τότε, ο συνολικός χρόνος εκτέλεσης του **Algorithm 3** είναι γραμμικός και γράφεται

$$T_3(n) = \mathcal{O}(n) \tag{11}$$

Σε ό,τι αφορά την διατήρηση της γραμμικότητας, απαραίτητη προϋπόθεση είναι να μην ισχύει καμία από τις ακόλουθες σχέσεις

$$N \to M^2 \tag{12}$$

$$M \to N^2. \tag{13}$$

Διότι τότε, η χρονική πολυπλοκότητα θα ήταν $\mathcal{O}(n^2)$. ■

Πρόβλημα 2

Πρόβλημα 2 *Έστω πίνακας T με στοιχεία n θετικούς ακεραίους με εύρος $[0, \dots, k]$ (k ακέραιος). Δίνεται ο αλγόριθμος*

Algorithm 4

```
1: for i = 0, ..., k do
2:     H[i] = 0
3: end for
4: for j = 1, ..., n do
5:     H[T[j]] = H[T[j]] + 1
6: end for
7: for i = 1, ..., k do
8:     H[i] = H[i] + H[i − 1]
9: end for
10: for j = n, ..., 1 do
11:     S[H[T[j]]] = T[j]
12:     H[T[j]] = H[T[j]] − 1
13: end for
```

Ο αλγόριθμος του **Προβλήματος 2** είναι ο αλγόριθμος ταξινόμησης *counting sort* (όπως συναντάται στην αγγλική βιβλιογραφία). [4][7]

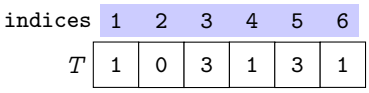
Περιγραφή πίνακα S

Αρχικά, στο **Πρόβλημα 2** δίνεται ένας πίνακας n στοιχείων, T, με εύρος από 0 έως k, όπου k θετικός ακέραιος, τον οποίο θέλουμε να ταξινομήσουμε. Χρησιμοποιώντας έναν βοηθητικό πίνακα H, ο οποίος είναι zero-indexed, μεγέθους $k + 1$ και έχοντας αρχικοποιήσει όλα του τα στοιχεία σε μηδέν (γραμμές 1 έως και 3), με την **for** των γραμμών 4 έως και 6 σαρώνουμε τον πίνακα T και σε κάθε επανάληψη του βρόχου χρησιμοποιούμε την τιμή $T[j]$ ως δείκτη για τον πίνακα H και πάμε στην θέση $H[T[j]]$, όπου αυξάνουμε το περιεχόμενό της κατά ένα, έτσι ώστε με το πέρας της **for** των γραμμών 4 έως και 6, ο H να είναι πίνακας συχνοτήτων εμφάνισης του κάθε ακέραιου (index του H) στον T.

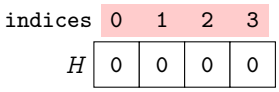
Έπειτα, έχοντας συμπληρώσει τον πίνακα H για κάθε ακέραιο (index του) με το πλήθος εμφανίσεων αυτού στον T, περνάμε στο **for** των γραμμών 7 έως και 9, όπου ενημερώνουμε τον H με τον ακόλουθο τρόπο. Αρχικά, αφήνουμε το περιεχόμενο στην θέση $H[0]$ αναλλοίωτο και για το $H[1]$ μέχρι και το $H[k]$ ενημερώνουμε το $H[i]$ με την τιμή $H[i] + H[i - 1]$, δηλαδή την τιμή του αθροίσματος του περιεχομένου της θέσης του H που υποδεικνύεται από τον μετρητή του βρόχου επανάληψης συν το περιεχόμενο της προηγούμενης θέσης του H. Οπότε, τελικά θα έχουμε έναν πίνακα H ο οποίος σε κάθε θέση του, x, θα περιέχει την θέση–index του ακεραίου x στον ταξινομημένο πίνακα.

Τώρα, έχοντας την νέα μορφή του πίνακα H στα χέρια μας θα χρησιμοποιήσουμε έναν ακόμη βοηθητικό πίνακα S μεγέθους n (και zero-indexed) στον οποίο **θα αποθηκευτεί το ταξινομημένο αποτέλεσμα**. Ξεκινάμε τον μετρητή του **for**, των γραμμών 10 έως και 13, από το τέλος του πίνακα T, δηλαδή από το n και τον μειώνουμε μέχρι να φτάσει και στην θέση 1. Καθώς σαρώνουμε τον T με τον μετρητή j, χρησιμοποιούμε ως δείκτη το περιεχόμενο του $T[j]$ για να προσπελάσουμε το αντίστοιχο στοιχείο στον H, δηλαδή $H[T[j]]$, και αφού πλέον ο H περιέχει τις θέσεις τον ακεραίων στον πίνακα S, θέτουμε $S[H[T[j]]] = T[j]$, καταχωρώντας έτσι στον πίνακα S, τα στοιχεία του πίνακα T, στις σωστές–ταξινομημένες θέσεις τους. Έπειτα της τοποθέτησης του στοιχείου $T[j]$ στην σωστή του θέση στον ταξινομημένο πίνακα S, μειώνουμε το περιεχόμενο του πίνακα H στην θέση $T[j]$ κατά ένα. Αυτή η κίνηση είναι ιδιαίτερα χρήσιμη στην περίπτωση ύπαρξης διπλοτύπων στον πίνακα T διότι μόλις συναντήσουμε έναν ακέραιο για δεύτερη (ή οποιαδήποτε) φορά, θα τον τοποθετήσουμε στην αμέσως προηγούμενη θέση από το στοιχείο με την ίδια τιμή.

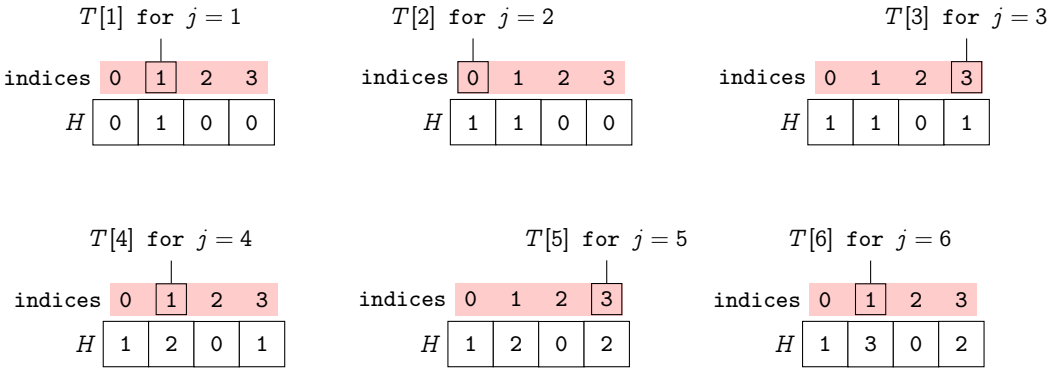
Παρακάτω, παραθέτουμε ένα **παράδειγμα εκτέλεσης** του αλγορίθμου με $T = [1, 0, 3, 1, 3, 1]$. Είναι, δηλαδή, $n = 6$ και $k = 3$.



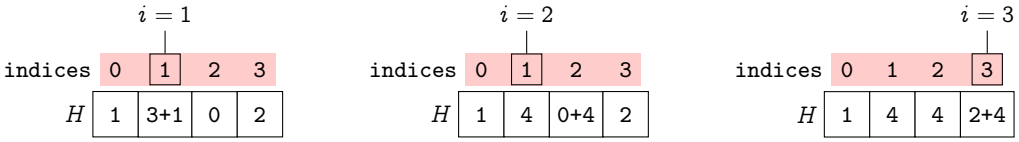
Βήμα 1. Δημιουργία και αρχικοποίηση του πίνακα H σε μηδενικά — γραμμές 1 έως και 3:



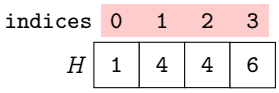
Βήμα 2. Σαρώνουμε τον πίνακα T και για κάθε ακέραιο j που συναντάμε αυξάνουμε κατά ένα 1 το περιεχόμενο της θέσης του H με index $T[j]$. Τελικά, σε κάθε θέση j του πίνακα H θα υπάρχει αποθηκευμένη η συχνότητα εμφάνισης του j στον πίνακα T. Δηλαδή, για κάθε ακέραιο j κάνουμε $H[T[j]] = H[T[j]]+1$, $j \in [1, n]$.



Βήμα 3. Στον πίνακα H του προηγούμενου βήματος, ξεκινώντας από την δεύτερη θέση, δηλαδή το $H[1]$, προσθέτουμε στο ήδη υπάρχον περιεχόμενο της εκάστοτε θέσης του πίνακα H, το περιεχόμενο της προηγούμενης θέσης (αυτός είναι και ο λόγος για τον οποίο αφήνουμε αμετάβλητο το $H[0]$) — γραμμές 7 έως και 9:



Τελικά, θα είναι

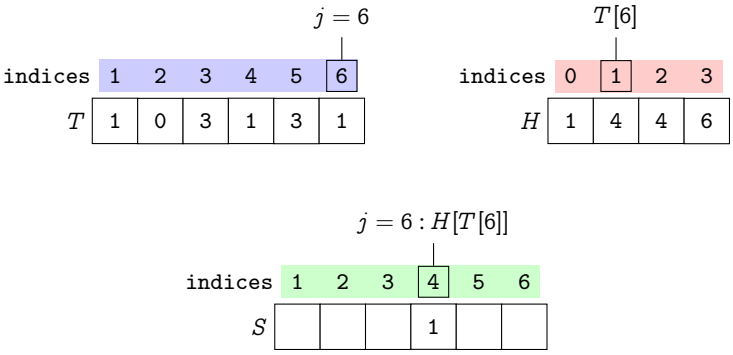


Η νέα μορφή του πίνακα H μας δίνει την πληροφορία *σε ποια θέση του ταξινομημένου πίνακα θα τοποθετήσουμε για τελευταία φορά*[4] *κάθε ακέραιο (index του πίνακα H).*

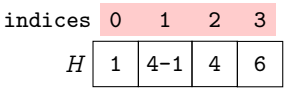
[4] Με την έκφραση «*τελευταία φορά*» εννοούμε πως στην περίπτωση όπου κάποιος ακέραιος, x, εμφανίζεται στον T περισσότερες από μία φορές, τότε το στοιχείο με τιμή x το οποίο έχει μεγαλύτερο index στον T από όλα τα υπόλοιπα στοιχεία του T με τιμή x, θα εμφανιστεί στον S στην θέση–index που του αντιστοιχεί σύμφωνα με τον H μετά το πέρας του **Βήματος 3**.

Βήμα 4. Σε έναν νέο πίνακα S, μεγέθους n, αποθηκεύουμε το ταξινομημένο αποτέλεσμα. Ο μετρητής του βρόχου επανάληψης των γραμμών 10 έως και 13 ξεκινάει από το n και μειώνεται έως και την τιμή 1. Χρησιμοποιούμε ως δείκτη για την πρόσβαση στον πίνακα H την τιμή που είναι αποθηκευμένη στο $T[j]$, δηλαδή διαβάζουμε την τιμή $H[T[j]]$ η οποία μας δίνει την θέση της τελευταίας εμφάνισης του ακεραίου $T[j]$ στον ταξινομημένο πίνακα S (σε κάθε iteration). Επομένως, θα είναι $S[H[T[j]]] = T[j]$.

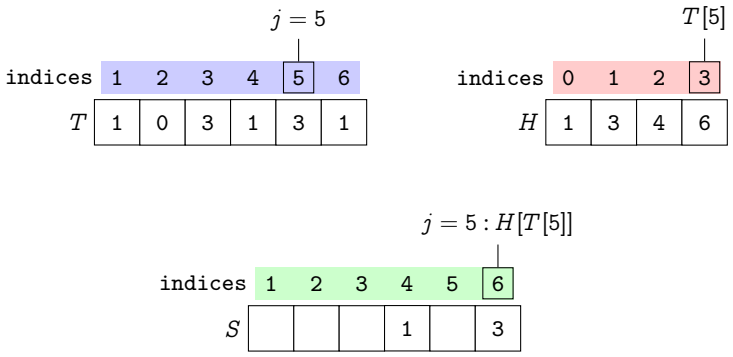
- $j = 6$:



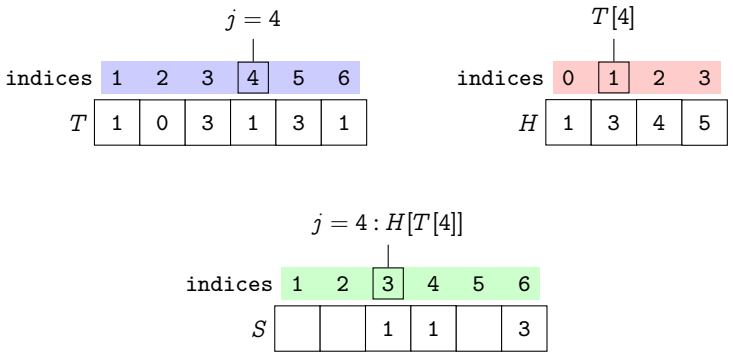
Αφού τοποθετήσουμε τον πρώτο ακέραιο στον ταξινομημένο πίνακα S πρέπει να μειώσουμε το περιεχόμενο στην αντίστοιχη καταχώρηση του H κατά 1, διότι την επόμενη φορά που θα συναντήσουμε τον συγκεκριμένο ακέραιο στον πίνακα T θα πρέπει να τοποθετηθεί στην αμέσως προηγούμενη θέση στον S. Δηλαδή, η νέα μορφή του H είναι



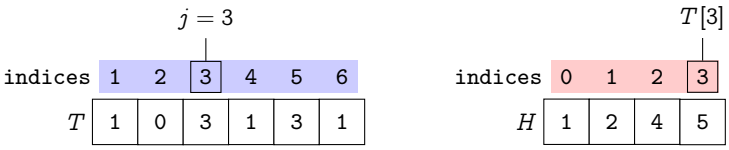
- $j = 5$:

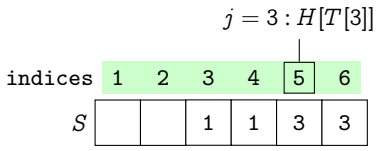


- $j = 4$:

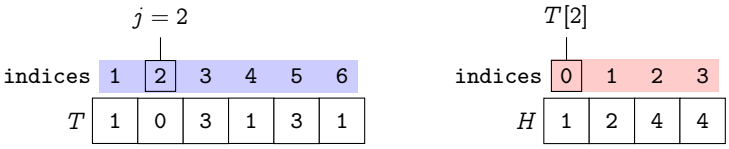


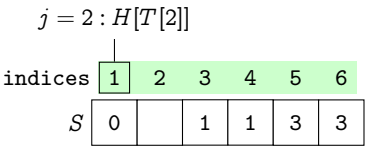
- $j = 3$:



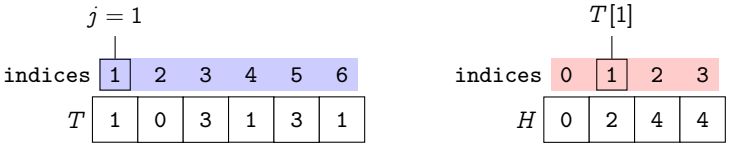


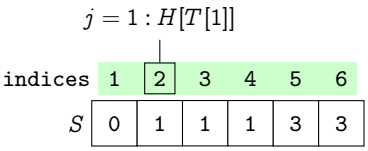
- $j = 2$:



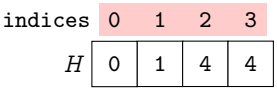


- $j = 1$:

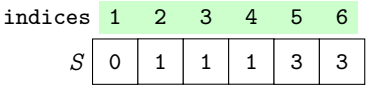




Τελικά, ο πίνακας H είναι



και ο ταξινομημένος πίνακας S είναι



Μέσω του παραπάνω παραδείγματος επαληθεύεται ο ισχυρισμός πως **ο πίνακας S είναι η ταξινομημένη μορφή του πίνακα T** και επιπλέον αναδεικνύεται μία σημαντική ιδιότητα του αλγορίθμου *counting sort*. Στοιχεία του πίνακα T τα οποία έχουν την ίδια τιμή, εμφανίζονται στον πίνακα S με την ίδια σειρά που εμφανίζονται και στον T [7]. Αλγόριθμοι ταξινόμησης με αυτήν την ιδιότητα αποκαλούνται *stable* [4]. Αυτο επιτυγχάνεται ξεκινώντας τον μετρητή της `for` των γραμμών 10-13, από την τιμή n.

Ανάλυση χρόνου εκτέλεσης

Στην παρούσα ενότητα θα αναλύσουμε τον χρόνο εκτέλεσης του **Algorithm 4** (counting sort). Ωστόσο, πριν αυτού, αξίζει να γίνουν κάποιες παρατηρήσεις σχετικά με την φύση του αλγορίθμου. Ο **Algorithm 4** ανήκει σε μία οικογένεια αλγορίθμων ταξινόμησης οι οποίοι δεν χρησιμοποιούν συγκρίσεις μεταξύ των στοιχείων όπως παραδείγματος χάριν ο quick-sort ή ο bubble-sort. Για τους αλγόριθμους ταξινόμησης των οποίων η αρχή λειτουργίας στηρίζεται σε συγκρίσεις μεταξύ των στοιχείων αποδεικνύεται πως η χρονική τους πολυπλοκότητα έχει κάτω φράγμα και συγκεκριμένα είναι $\Omega(n \lg n)$ [7]. Ωστόσο, εγκαταλείποντας το μοντέλο των συγκρίσεων, παύει να ισχύει και το κάτω φράγμα $\Omega(n \lg n)$ [7].

Αρχικά, ο πίνακας με τα δεδομένα προς ταξινόμηση, T, έχει n στοιχεία. Κάθε μία από τις γραμμές 2, 5, 8, 11 και 12 του **Algorithm 4** απαιτεί σταθερό χρόνο εκτέλεσης $\mathcal{O}(1)$. Επομένως, έχουμε

i. Το σώμα του `for` των γραμμών $1-3$ τρέχει $k+1$ φορές. Επομένως, ο χρόνος εκτέλεσής του είναι $(k+1)\cdot\mathcal{O}(1) = \mathcal{O}(k+1) = \mathcal{O}(k)$.

ii. Το σώμα του `for` των γραμμών $4-5$ εκτελείται n φορές. Επομένως, ο χρόνος εκτέλεσής του είναι $n\cdot\mathcal{O}(1) = \mathcal{O}(n)$.

iii. Το σώμα του `for` των γραμμών $7-9$ εκτελείται k φορές. Επομένως, ο χρόνος εκτέλεσής του είναι $k\cdot\mathcal{O}(1) = \mathcal{O}(k)$.

iv. Το σώμα του `for` των γραμμών $10-13$ εκτελείται n φορές. Επομένως, ο χρόνος εκτέλεσής του είναι $n\cdot(\mathcal{O}(1)+\mathcal{O}(1)) = \mathcal{O}(n)$.

Προσθέτοντας τους παραπάνω χρόνους προκύπτει πως τελικά ο χρόνος εκτέλεσης του **Algorithm 4** είναι

$$T(n,k) = \mathcal{O}(k) + \mathcal{O}(n) + \mathcal{O}(k) + \mathcal{O}(n)$$

ή

$$T(n,k) = \mathcal{O}(n+k). \tag{14}$$

Δηλαδή, ο αλγόριθμος είναι γραμμικής χρονικής πολυπλοκότητας. Προκειμένου να επιτύχουμε την καλύτερη δυνατή λειτουργία του δεν θα πρέπει το εύρος τιμών, k, να είναι πολύ μεγαλύτερο από το n αλλά, να έχει ως άνω όριο το $\mathcal{O}(n)$, δηλαδή $k = \mathcal{O}(n)$. Αυτή είναι κιόλας η συνήθης περίπτωση εφαρμογής του αλγορίθμου, διότι τότε ο χρόνος εκτέλεσής του γίνεται $T(n) = \mathcal{O}(2\cdot n) = \mathcal{O}(n)$ [7]. Επιπλέον προϋπόθεση για την διατήρηση της γραμμικότητας του χρόνου εκτέλεσης του **Algorithm 4** είναι το k να μην προσεγγίζει το n^2, διότι τότε, ο χρόνος εκτέλεσης τους αλγορίθμου θα είναι $\mathcal{O}(n+n^2)$, δηλαδή $\mathcal{O}(n^2)$ καθιστώντας τον μη αποδοτικό.

Βιβλιογραφία

[1] Dasgupta, S. and Papadimitriou, C.H. and Vazirani, U.V. *Algorithms*. McGraw-Hill Higher Education, 2006. ISBN: 9780077388492.

[2] Graham, R.L. and Knuth, D.E. and Patashnik, O. *Concrete Mathematics: A Foundation for Computer Science*. A foundation for computer science. Addison-Wesley, 1994. ISBN: 9780201558029.

[3] Gross, J.L. and Yellen, J. and Anderson, M. *Graph Theory and Its Applications*. Textbooks in mathematics. CRC Press, Taylor & Francis Group, 2019. ISBN: 9781482249484.

[4] D.E. Knuth. *The Art of Computer Programming: Volume 3: Sorting and Searching*. Pearson Education, 1998. ISBN: 9780321635785.

[5] *Pyhton 3 Documentation: The Python Standard Library: Built-in Types: Mapping Types — dict*. https://docs.python.org/3/library/stdtypes.html. Python 3.11.2.

[6] Rosen, K.H. *Discrete Mathematics and Its Applications*. McGraw-Hill, 2019. ISBN: 9781260091991.

[7] Thomas H. Cormen and Charles E. Leiserson and Ronald L. Rivest and Clifford Stein. *Introduction to Algorithms*. Third Edition. The MIT Press, 2009. ISBN: 9780262033848.