Αλγόριθμοι - Εργασία 1

Δεληγιαννάκης Χαράλαμπος
4383

Απρίλιος 2024

Εισαγωγή

1 Πρώτη άσκηση

Δεδομένου ότι:
$f_1 = 2^{2n} + 3^n \quad f_2 = n^{100} + 2^{\frac{n}{2}} \quad f_3 = n2^{\frac{logn}{4}} + 2^{\frac{2}{3}logn} \quad f_4 = 2^{nlogn} + (logn)^{\sqrt{n}}$

$f_5 = 2^{\frac{loglogn}{2}} + loglogn \quad f_6 = 9^{\frac{n}{2}} + 16^{\frac{n}{8}}\sqrt{n} \quad f_7 = 2^{\frac{2}{3}logn} + (logn)^{100} \quad f_8 = n^{\frac{7}{8}} + n2^{\frac{logn}{2}}$

Η τελική διάταξη για μεγάλα n είναι η εξής:
$f_5 < f_7 < f_3 < f_8 < f_2 < f_6 < f_1 < f_4$

Για να αποδείξουμε αυτήν την διάταξη, θα αποδείξουμε κάθε γειτονική ανίσωση
($f_5 < f_7$, $f_7 < f_3$, ..., $f_1 < f_4$ για μεγάλα n).

Ωστόσο, πρωτού γίνει αυτό, θα βρούμε τον μεγαλύτερο όρο από κάθε ζευγάρι για μεγάλα n, ώστε να χρησιμοποιήσουμε μόνο αυτόν, αγνοώντας τον άλλον, στις συγκρίσεις μεταξύ των συναρτήσεων.

Σύμφωνα με γνωστό θεώρημα, αν $\lim_{n\to\infty} \frac{f(n)}{g(n)} = 0$ τότε ισχύει ότι $f(n) = O(g(n))$. Άρα, εφόσον το κάθε όριο ισούται με μηδέν, σημαίνει ότι ο παρανομαστής έχει μεγαλύτερη πολυπλοκότητα για μεγάλα n από τον αριθμητή, οπότε θα κρατήσουμε αυτόν:

$$f_1 : \lim_{n\to\infty} \frac{3^n}{\mathbf{2^{2n}}} = 0 \quad f_2 : \lim_{n\to\infty} \frac{n^{100}}{\mathbf{2^{\frac{n}{2}}}} = 0 \quad f_3 : \lim_{n\to\infty} \frac{2^{\frac{2}{3}logn}}{\mathbf{n2^{\frac{logn}{4}}}} = 0 \quad f_4 : \lim_{n\to\infty} \frac{(logn)^{\sqrt{n}}}{\mathbf{2^{nlogn}}} = 0$$

$$f_5 : \lim_{n\to\infty} \frac{loglogn}{\mathbf{2^{\frac{loglogn}{2}}}} = 0 \quad f_6 : \lim_{n\to\infty} \frac{16^{\frac{n}{8}}\sqrt{n}}{\mathbf{9^{\frac{n}{2}}}} = 0 \quad f_7 : \lim_{n\to\infty} \frac{(logn)^{100}}{\mathbf{2^{\frac{2}{3}logn}}} = 0 \quad f_8 : \lim_{n\to\infty} \frac{n^{\frac{7}{8}}}{\mathbf{n2^{\frac{logn}{2}}}} = 0$$

Επομένως, η τελική διάταξη δικαιολογείται ώς εξής:

$$\lim_{n\to\infty} \frac{2^{\frac{loglogn}{2}}}{2^{\frac{2}{3}logn}} = 0 \to f_5 = O(f_7)$$

$$\lim_{n\to\infty} \frac{2^{\frac{2}{3}logn}}{n2^{\frac{logn}{4}}} = 0 \to f_7 = O(f_3)$$

$$\lim_{n\to\infty} \frac{n2^{\frac{logn}{4}}}{n2^{\frac{logn}{2}}} = 0 \to f_3 = O(f_8)$$

$$\lim_{n\to\infty} \frac{n2^{\frac{logn}{2}}}{2^{\frac{n}{2}}} = 0 \to f_8 = O(f_2)$$

$$\lim_{n\to\infty} \frac{2^{\frac{n}{2}}}{9^{\frac{n}{2}}} = 0 \to f_2 = O(f_6)$$

$$\lim_{n\to\infty} \frac{9^{\frac{n}{2}}}{2^{2n}} = 0 \to f_6 = O(f_1)$$

$$\lim_{n\to\infty} \frac{2^{2n}}{2^{nlogn}} = 0 \to f_1 = O(f_4)$$

2 Δεύτερη άσκηση

Για τη λύση του συγκεκριμένου προβλήματος, θα δημιουργήσουμε μια παραλλαγή του γνωστού αλγορίθμου BFS (Breadth First Search - Αναζήτηση Κατά Βάθος) σε γράφο. Αυτός αλγόριθμος είναι μια πρώτη προσέγγιση για τη λύση του προβλήματος. Γενικά, ο BFS χρησιμοποιείται για την εύρεση του συντομότερου μονοπατιού σε έναν γράφο από έναν αρχικό κόμβο σε έναν τελικό. Η χρονική πολυπλοκότητα του είναι $O(V + E)$ όπου V είναι το πλήθος των κόμβων και E το πλήθος των ακμών. Επειδή είναι γνωστός αλγόριθμος, γνωρίζουμε ότι είναι ορθός, οπότε θα το χρησιμοποιήσουμε ως δεδομένο. Παρ'όλα αυτά, επειδή υπάρχει τροποποίηση του γράφου (αντιστροφή κατεύθυνσης όλων των ακμών) σε κάθε περιττή χρονική μονάδα ($t = 1, 3, ...$), θα χρειαστεί να τροποποιήσουμε τον BFS, ώστε να δουλεψει για το συγκεκριμένο πρόβλημα. Ο αλγόριθμος για αυτό είναι το εξής:

1. Ξεκινάμε τρέχοντας BFS στον αρχικό κόμβο s.
2. Σε κάθε επανάληψη, αν η χρονική μονάδα είναι περιττή προχωράμε στον επόμενο κόμβο με βάση τον γράφο με τις κανονικές κατευθύνησεις, αλλιώς παραμένουμε ακίνητοι (στον ίδιο κόμβο).
3. Συνεχίζουμε μέχρι να φτάσουμε στον τελικό κόμβο t.
4. Επιστρέφουμε το μικρότερο μονοπάτι από τον s στον t.

Ας λύσουμε ένα πρόβλημα με τον πιο γρήγορο χρονικά τρόπο σε έναν τυχαίο γράφο.

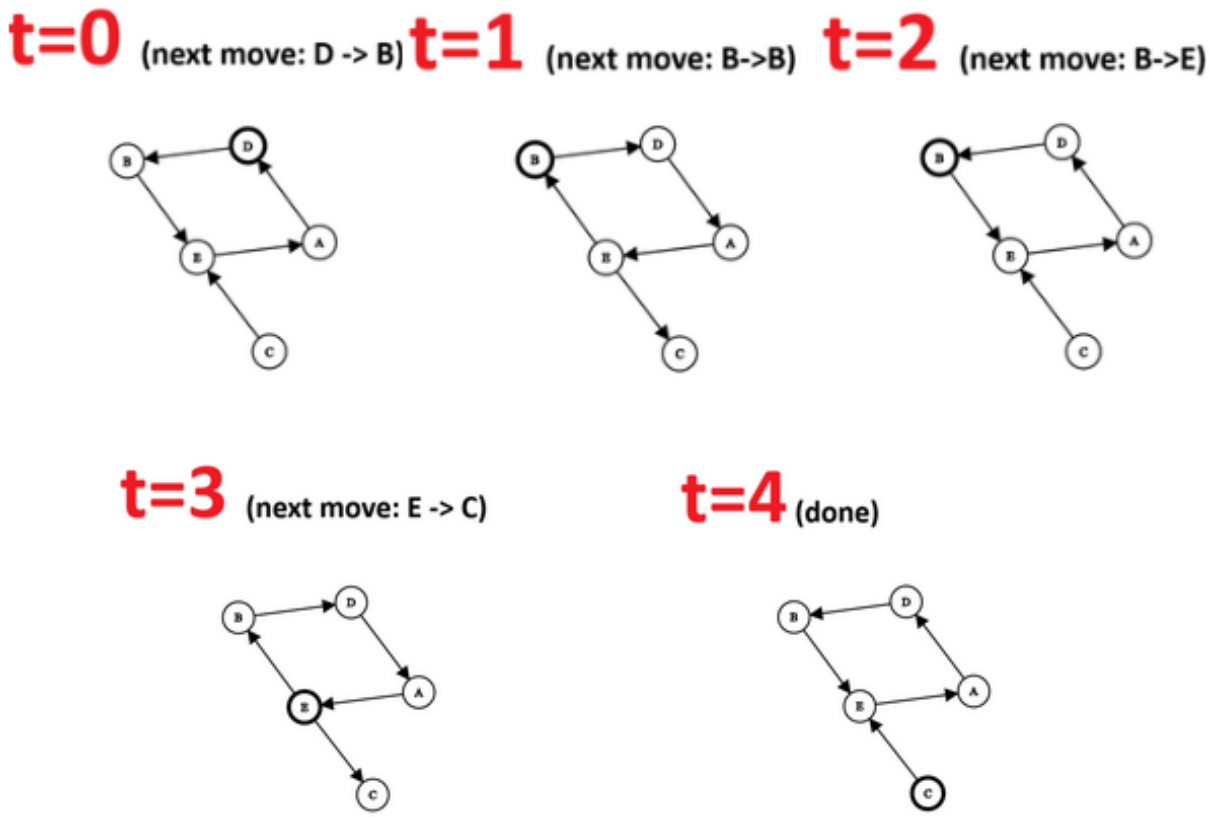

Figure 1: Γρηγορότερη λύση του προβλήματος

Στον συγκεκριμένο γράφο, ο παραπάνω αλγόριθμος δεν θα λύσει το πρόβλημα, καθώς για $t = 3$ θα μείνει στον κόμβο E, και απο εκεί και πέρα για κάθε άρτιο χρόνο δεν θα βρίσκει τρόπο να πάει στον τελικό κόμβο C και για κάθε περιττό χρόνο θα μένει στον ίδιο χρόνο.

Για να αντιμετωπιστεί αυτή η περίπτωση, χρειάζεται μια επιπλέον τροποποίηση που θα αντιμετωπίσει και αυτές τις περιπτώσεις. Ο αλγόριθμος θα τρέχει με τον εξής τρόπο:

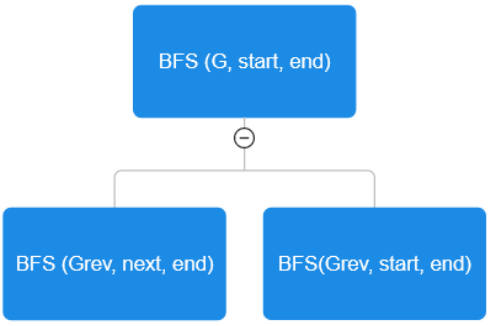

Figure 2: Δέντρο εκτέλεσης που θα δημιουργήσει ο αλγόριθμος

Δηλαδή, ο αλγόριθμος θα ξεκινήσει τρέχοντας BFS στον κανονικό γράφο (G) από τον αρχικό κόμβο προς τον τελικό. Έπειτα, ο κόμβος του δέντρου θα δημιουργήσει δύο παιδία. Το πρώτο παιδί θα τρέξει τον BFS από τον επόμενο κόμβο που επισκέπτεται το BFS του γονιού αλλά στον γράφο με τις αντιστραμμένες ακμές. Το δεύτερο παιδί θα τρέξει BFS από τον ίδιο κόμβο του γονέα, επίσης στον γράφο με τις αντιστραμμένες ακμές. Ουσιαστικά, το δεύτερο παιδί εκτελεί την περίπτωση που μένουμε στον ίδιο κόμβο για μια χρονική περίοδο. Πρωτού συνεχίσουμε την ανάλυση, ας δούμε ένα άλλο παράδειγμα στο οποίο θα εκτελεστεί αυτός ο αλγόριθμος. Έστω ο γράφος:

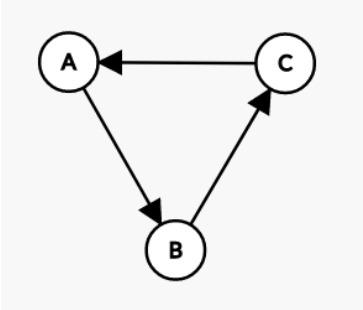

Figure 3: Γράφος παραδείγματος

Το δέντρο εκτέλεσης θα είναι το εξής:

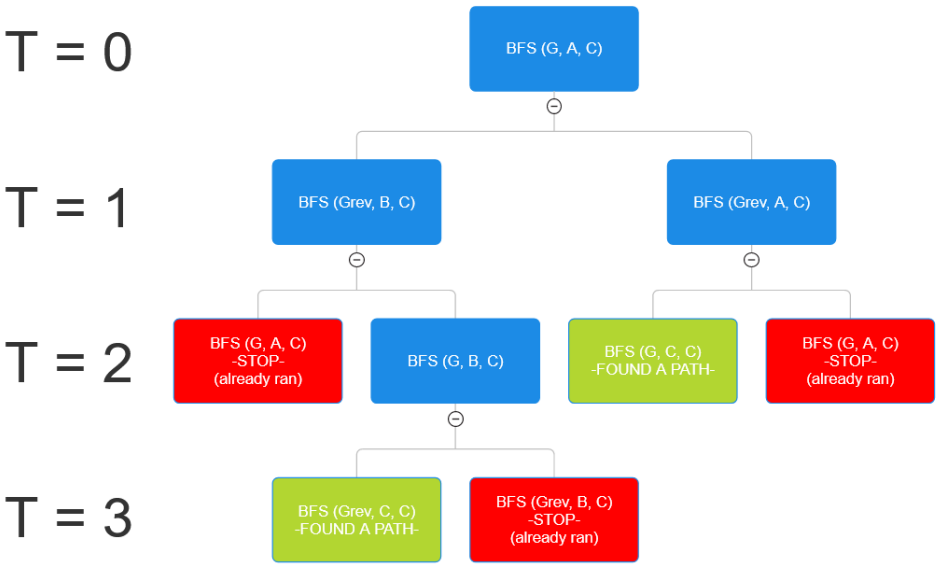

Figure 4: Δέντρο εκτέλεσης παραπάνω αλγορίθμου

Όπως βλέπουμε, ο αλγόρθιμος θα τρέξει με τον παραπάνω τρόπο. Για κάθε κόμβο υπάρχουν τρεις περιπτώσεις:

1. Να παράγει κανονικά τα παιδία σύμφωνα με το πρότυπο.

2. Να έχει τρέξει ήδη οπότε να μην παράγει κάτι.

3. Να καταλήξει στον τελικό κόμβο οπότε να μην παράγει κάτι.

Εφόσον τερματίσει, θα βρει το κοντινότερο μονοπάτι ξεκινώντας από τη ρίζα και καταλήγοντας σε κόμβο που είναι σημαδεμένος με $FOUND$. Αυτό θα αποτελέσει τη λύση του προβλήματος (A > A > C).

Υποσημείωση: Όταν λέμε να τρέξει BFS από έναν κόμβο, εννούμε να κάνει το επόμενο βήμα που θα έκανε ο BFS σε επόμενο κόμβο, και όχι να τρέξει ολόκληρος ο BFS. Δηλαδή, O(1). Σχετικά με την ορθότητα, για να λύσουμε αυτό το πρόβλημα χωρίς τη χρήση αλγορίθμου, θα χρειαστεί για κάθε κόμβο σε κάποια χρονική στιγμή, είτε να μείνουμε ακίνητοι, είτε να επισκεφτούμε επόμενο κόμβο. Μετά από κάποιο βήμα, θα αντιστρέφουμε τον γράφο και συνεχίζοντας με αυτόν τον τρόπο κάποτε θα καταλήξουμε τη λύση του. Αυτό ακριβώς κάνει και ο αλγόριθμος καθώς σύμφωνα με το πρότυπο δέντρο, εξετάζονται και οι δυο περιπτώσεις για κάθε κόμβο. Επίσης, εγγυάται ότι θα βρεθεί η γρηγορότερη λύση καθώς στο τέλος διαλέγεται το πιο σύντομο μονοπάτι από τα συνολικά μονοπάτια που από τον αρχικό κόμβο καταλήγουν στον τελικό.

Σχετικά με την πολυπλοκότητα, ο τροποποιημένος BFS αλγόριθμος θα έχει το πολύ $2(V+E)$ βήματα, όπως ο κανονικός BFS αλλά επί δύο καθώς θα τρέξει και για τον G και για τον $Grev$. Κάθε φορά που γίνεται κάποιο βήμα, θα αποθηκεύονται οι παράμετροι σε έναν hash map και όταν είναι να γίνει επόμενο βήμα θα χρεισατεί να ελέγξουμε αν έχει εκτελεστεί ήδη, δηλαδή αν υπάρχουν οι παράμετροι ήδη στον hash map. Αυτό είναι $O(1)$ καθώς αυτή η λειτουργία είναι γνωστό ότι γίνεται σε σταθερό χρόνο. Επίσης, όταν τερματίσει θα χρειαστεί να βρει το κοντινότερο μονοπάτι της λύσης, δηλαδή του κόμβου BFS(G, end, end) ή BFS(Grev, end, end). Αυτό, σύμφωνα με γνωστό αλγόριθμο, είναι ο $O(n)$ με n το πλήθος τον κόμβων, που σε αυτήν την περίπτωση είναι το πολύ $2V$, άρα $O(V)$. Η αναζήτηση αν υπάρχει κάποιος παρόμοιος κόμβος θα γίνεται για κάθε κόμβο, ενώ η αναζήτηση του κοντινότερου μονοπατιού θα γίνει στο τέλος. Επομένως, η τελική χρονική πολυπλοκότητα είναι $O(2(V+E)+V)$ δηλαδή $O(V+E)$.

3 Τρίτη άσκηση

Το συγκεκριμένο πρόβλημα έχει μία αποδοτική λύση πολυπλοκότητας $O(nlogn)$ χρησιμοποιώντας τη μέθοδο "διαίρει και βασίλευε". Η γενική περιγραφή του αλγορίθμου, ώστε να βρεθεί το μέγιστο άθορισμα υποπίνακα σε έναν πίνακα είναι η εξής:

1. Εύρεση μέγιστου αθροίσματος υποπίνακα στο αριστερό μισό του αρχικού πίνακα. T(n/2)

2. Εύρεση μέγιστου αθροίσματος υποπίνακα στο δεξί μισό του αρχικού πίνακα. T(n/2)

3. Συγχώνευση πινάκων και έυρεση μέγιστου αθροίσματος υποπίνακα που περιέχει το μεσαίο στοιχείο του συγχωνεμένου πίνακα. O(n)

Ο παρπάνω αλγόριθμος βρίσκει τα 3 υπογραμμισμένα αποτελέσματα και η λύση που επιστρέφει είναι το μεγαλύτερο από αυτά.

Τα πρώτα δύο βήματα γίνονται με αναδρομική κλήση, που είναι βασικό χαρακτηριστικό της μεθόδου "διαίρει και βασίλευε". Παρ'όλα αυτά, τα αποτελέσματα που θα προκύψουν από αυτά τα δύο δεν αρκούν από μόνα τους, καθώς θα χρειαστεί να βρούμε και το μέγιστο άθροισμα υποπίνακα που περιέχει το μεσαίο στοιχείο του συγχωνεμένου πίνακα.

Για παράδειγμα, έστω ο πίνακας [-3, 1, 5, 7, -10]. Το μέγιστο άθροισμα υποπίνακα στο αριστερό μισό του είναι το 6, και στο δεξί μισό του είναι το 7. Επομένως, αν δεν βρίσκαμε το άθροισμα που περιγράφεται στο Βήμα 3, ο αλγόριθμος θα επέστρεφε το μεγαλύτερο από τα δυό αθροίσματα, δηλαδή το 7. Παρ'όλα αυτά, όπως είναι αντιληπτό, το μέγιστο άθροισμα υποπίνακα είναι 13, δηλαδή ο υποπίνακας [1, 5, 7]. Το σφάλμα αυτό θα προέκυπτε επειδή στη συγχώνευση τον πινάκων δεν εξετάστηκε και η οριακή περίπτωση, δηλαδή το μέγιστο άθροισμα υποπίνακα να προέρχεται και από τα δυό μισά.

Ο τρόπος που μπορούμε να αντιμετωπίσουμε αυτό το πρόβλημα είναι απλός. Όταν ο πίνακας συγχωνευτεί, θα χρειαστεί να βρούμε δύο αθροίσματα. Το πρώτο άθροισμα είναι το μέγιστο άθροισμα του αριστερού μέρους που περιέχει το μεσαίο στοχείο, ξεκινώντας από αυτό και πηγαίνοντας προς τα αριστερά μέχρι το πρώτο στοιχείο. Σε κάθε επανάληψη, προσθέτουμε το νέο στοιχείο που βρήκαμε πηγαίνοντας αριστερά και συγκρίνουμε το νέο άθροισμα με το μέγιστο άθορισμα. Το δεύτερο άθροισμα είναι το μέγιστο άθροισμα του δεξιού μέρους που περιέχει το μεσαίο στοχείο, ξεκινώντας από αυτό και πηγαίνοντας προς τα δεξιά μέχρι το τελευταίο στοιχείο. Σε κάθε επανάληψη, προσθέτουμε το νέο στοιχείο που βρήκαμε πηγαίνοντας δεξιά και συγκρίνουμε το νέο άθροισμα με το μέγιστο άθροισμα. Έπειτα, ως τελικό μέγιστο άθροισμα του υποπίνακα που περιέχει το μεσαίο στοιχείο θα επιστρέφεται το μεγαλύτερο μεταξύ των δύο αυτών αθροισμάτων και του αθροίσματος των αθροισμάτων μείον το μεσαίο στοιχείο (καθώς περιέχεται και στα δύο αθροίσματα οπότε πρέπει να αφαιρεθεί μια φορά). Η περιγραφή αυτού το αλγορίθμου μετά τη συγχώνευση είναι η εξής:

1. Έυρεση μέγιστου αθροίσματος στο αριστερό μέρος του συγχωνεμένου πίνακα και μέσα σε αυτό να περιέχεται το μεσαίο στοιχείο.

2. Έυρεση μέγιστου αθροίσματος στο δεξί μέρος του συγχωνεμένου πίνακα και μέσα σε αυτό να περιέχεται το μεσαίο στοιχείο.

3. Επιστροφή του μεγαλύτερου, μεταξύ του αθροίσματος στο βήμα 1, του αθροίσματος στο βήμα 2 και του αθροίσματος των δύο αυτών αθροισμάτων μείον το μεσαίο στοιχείο.

Για παράδειγμα στον πίνακα [-3, 1, 5, 7, -10], ο συγκεκριμένος αλγόριθμος στο βήμα 1 θα έβρισκε 6 (5+1), στο βήμα 2 θα έβρισκε 12 (5+7) και θα επέστρεφε το μεγαλύτερο μεταξύ του 6, 12 και 6 + 12 - 5 = 13, δηλαδή το 13.

Σχετικά με την ορθότητα του αλγορίθμου, θα αποδείξουμε επαγωγικά ότι ο αλγόριθμος λειτουργεί σωστά και επιστρέφει το επιθυμητό αποτέλεσμα. Αρχικά, για πίνακα με $n = 1$ θέσεις, το αποτέλεσμα θα ήταν το μοναδικό στοιχείο του πίνακα, πράγμα που θα επέστρεφε και ο αλγόριθμος αν ακολουθούσαμε τα βήματα. Στη συνέχεια, ας υποθέσουμε οτι αλγόριθμος δουλέυει για πίνακα k θέσεων. Θα πρέπει να αποδείξουμε ότι θα δουλεύει και για πίνακα $k + 1$ θέσεων. Εφόσον ο πίνακας με $k + 1$ θέσεις θα σπάσει, τότε θα έχουμε υποπίνακες με περίπου $k/2$ στοιχεία, στους οποίους ο αλγόριθμος δουλέυει από την υπόθεση. Δηλαδή, επειδή ο αλγόριθμος λειτουργεί σωστά για κάθε μικρότερη ακολουθία, συμπεριλαμβανομένων και των περιπτώσεων με άρτιο και περιττό μήκος, μπορούμε να συμπεράνουμε ότι θα λειτουργεί σωστά και για ακολουθίες μεγαλύτερου μήκους. Άρα, με επαγωγή αποδείξαμε ότι ο αλγόριθμος είναι ορθός.

Σχετικά με την πολυπλοκότητα του αλγορίθμου, ο τύπος της αναδρομής είναι ο εξής: $T(n) = 2T(n/2) + n$. Δηλαδή, σε κάθε επανάληψη, σπάμε τον πίνακα στα 2 και ασχολούμαστε με το κάθε μέρος ξεχωριστά, επομένως προκύπτει το $2T(n/2)$. Επίσης, σε κάθε επανάληψη, θα χρεαστεί η εύρεση του μέγιστου αθροίσματος υποπίνακα που περιέχει το μεσαίο στοιχείο του συγχωνεμένου πίνακα. Ο αλγόριθμος για να γίνει αυτό περιγράφτηκε παραπάνω και έχει πολυπλοκότητα $\Theta(n)$ καθώς διατρέχουμε όλα τα στοιχεία του πίνακα (μία φορά από τη μέση προς τα αριστερά και μία φορά από τη μέση προς τα δεξιά). Για να βρούμε την πολυπλοκότητα, θα εφαρμόσουμε Master Method στον παραπάνω αναδρομικό τύπο. Ισχύει ότι $a = b^k$, δηλαδή $2 = 2^1$, επομένως η πολυπλοκότητα είναι $\Theta(nlogn)$.

Υποσημείωση: Σε περίπτωση που θέλαμε να επιστρέφουμε και τους μήνες, δηλαδή τις θέσεις του πίνακα που υπάρχει το μέγιστο άθροισμα, θα δημιουργούσαμε δύο pointers που θα είχαν την αρχή και το τέλος του και θα ενημερωνόντουσαν ανάλογα.

4 Τέταρτη άσκηση

4.1 Πρώτο ζητούμενο

Ο συγκεκριμένος αλγόριθμος συγχωνεύει δυό συστοιχίες και το αποτέλεσμά της με την επόμενη, ώσπου να έχουν τελειώσει οι συστοιχίες. Ο τρόπος συγχώνευσης θα γίνει με τον γνωστό αλγόριθμο συγχώνευσης δύο ταξινομημένων πινάκων σε γραμμικό χρόνο. Αυτός ο

αλγόιρθμος βασίζεται την εναλλαγή στοιχείων από τους δύο πίνακες μέχρις ότου ο ένας από τους δύο πίνακες να εξαντληθεί. Σε κάθε επανάληψη, συγκρίνονται τα αντίστοιχα στοιχεία από τους δύο πίνακες και τοποθετείται το κατάλληλο στον τελικό πίνακα, ώσπου ένας πίνακας να μείνει άδειος. Τότε, τοποθετούμε τα υπόλοιπα στοιχεία του άλλου πίνακα στον τελικό και παράγεται το επιθυμητό αποτέλεσμα. Η ορθότητα του αλγορίθμου πηγάζει από το γεγονός ότι κάθε φορά που τοποθετεί ένα στοιχείο στον τελικό πίνακα, είναι αναγκασμένο να είναι το μικρότερο διαθέσιμο στοιχείο ανάμεσα στα στοιχεία που εξακολουθούν να περιμένουν για εισαγωγή. Αυτό εξασφαλίζει ότι ο τελικός πίνακας θα είναι επίσης ταξινομημένος. Τώρα, όσον αφορά στην πολυπλοκότητα, αυτός ο αλγόριθμος λειτουργεί σε γραμμικό χρόνο. Αυτό σημαίνει ότι η πολυπλοκότητά του είναι $O(n)$, όπου n είναι το συνολικό μέγεθος των δύο αρχικών πινάκων. Αυτό είναι δυνατό επειδή ο αλγόριθμος εξετάζει και τα δύο στοιχεία των πινάκων μόνο μία φορά κατά τη διάρκεια της εκτέλεσής του.

Σχετικά με τον αλγόριθμο της άσκησης, εφόσον ο τρόπος συγχώνευσης είναι ορθός και η σειρά με την οποία συγχωνεύει τους πίνακες είναι επίσης ορθή, αποδεικνύεται ότι είναι ορθός. Επίσης, η πολυπλοκότητά του είναι $O(kn)$, καθώς χρησιμοποιείται ο αλγόριθμος συγχώνευσης με γραμμική πολυπλοκότητα $k-1$ φορές, δηλαδή περίπου k φορές.

4.2 Δεύτερο ζητούμενο

Το συγκεκριμένο πρόβλημα θα μπορούσε να λυθεί γρηγορότερα με τη χρήση της μεθόδου "διαίρει και βασίλευε". Ο αλγόριθμος για αυτό αποτελείται από τα εξής βήματα:

1. Χωρίζουμε τις ταξινομημένες συστοιχίες στη μέση (αναδρομικά το ίδιο για κάθε μισό)

2. Φτάνοντας στο σημείο που έχουμε απομονώσει κάθε συστοιχία ξεχωριστά, συγχωνεύουμε τις δύο ταξινομημένες συστοιχίες κάθε ομάδας για να δημιουργήσουμε μία νέα ταξινομημένη συστοιχία, μέχρι να καταλήξουμε στην τελική ταξινομημένη συγχωνεμένη συστοιχία που θα περιέχει kn στοιχεία.

Προκειμένου να γίνει κατανοητός ο παραπάνω αλγόριθμος, παρουσιάζεται γραφικά το δέντρο που θα δημιουργήσει ο αλγόριθμος σε ένα παράδειγμα. Έστω υπάρχουν οι αρχικοί πίνακες [7, 15, 20] [4, 8, 10] [1, 24, 30] [12, 13, 14], δηλαδή $k = 4$ και $n = 3$. Ο αλγόριθμος θα σπάει τις 4 ταξινομημένες συστοιχίες στη μέση μέχρι να καταλήξει σε κάθε συστοιχία ξεχωριστά. Έτσι, καταλήγει στα φύλλα του δέντρου τα οποία περιέχουν τις αρχικές συστοιχίες. Έπειτα, ανεβαίνοντας από κάτω προς τα πάνω συγχωνεύει ανά δύο τις συστοιχίες μέχρι να φτάσει στην τελική ταξινομημένη συστοιχία.
Στο σχήμα, με κόκκινο ακολουθείται η πορεία του πρώτου βήματος, με πράσινο του δεύτερου, καθώς και σε πράσινο πλαίσιο αναγράφονται τα αποτελέσματα του βήματος 2 σε κάθε επανάληψη.

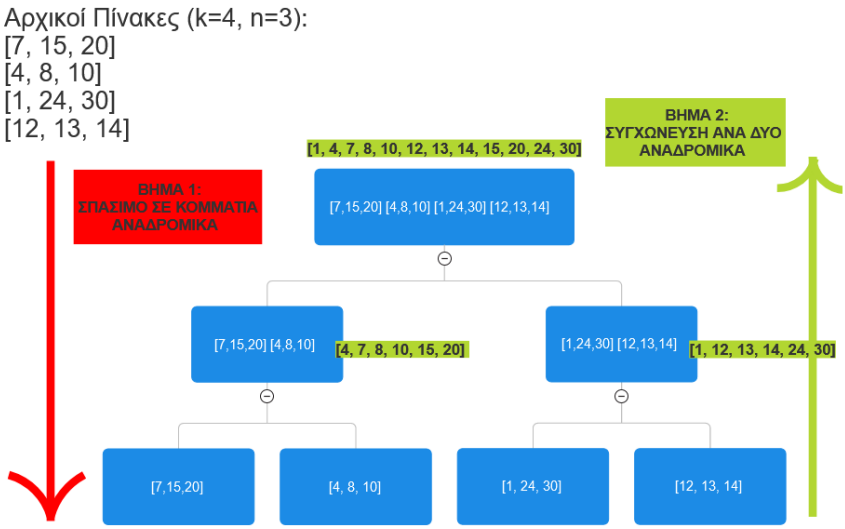

Figure 5: Δέντρο αναδρομής που θα δημιουργήσει ο αλγόριθμος

Ο τρόπος που γίνεται η συγχώνευση δύο ταξινομημένων συστοιχιών είναι με τη χρήση του ήδη γνωστού αλγορίθμου συγχώνευσης ταξινομημένων πινάκων σε γραμμικό χρόνο, που χρησιμοποιεί έναν δείκτη για κάθε συστοιχία, συγκρίνει τα επιμέρους στοιχεία και αυξάνει τον αντίστοιχο δείκτη. Επειδή ο αλγόριθμος είναι γνωστός, γνωρίζουμε ότι είναι ορθός και έχει γραμμική πολυπλοκότητα.

Σχετικά με την ορθότητα του αλγορίθμου της άσκησης, θα αποδείξουμε επαγωγικά ότι ο αλγόριθμος λειτουργεί σωστά και επιστρέφει το επιθυμητό αποτέλεσμα. Αρχικά για $k = 1$ ταξινομημένες συστοιχίες, δεν χρειάζεται κάποια εκτέλεση γιατί η λύση είναι ήδη έτοιμη, ενώ για $k = 2$ συστοιχίες ο αλγόριθμος απλά θα συγχωνεύσει τις δύο συστοιχίες. Στη συνέχεια, ας υποθέσουμε ότι ο αλγόριθμος δουλέυει για λ ταξινομημένες συστοιχίες. Θα πρέπει να αποδείξουμε ότι δουλεύει και για $\lambda+1$ ταξινομημένες συστοιχίες. Εφόσον οι $\lambda+1$ ταξινομημένες συστοιχίες σπάσουν στα δύο, τότε θα έχουμε περίπου $\lambda/2$ ταξινομημένες συστοιχίες για τις οποίες ο αλγόριθμος θα δουλέψει από την επαγωγική υπόθεση (καθώς αφού δουλεύει για λ συστοιχίες θα δουλεύει και για $\lambda/2$). Δηλαδή, επειδή ο αλγόριθμος λειτουργεί σωστά για κάθε μικρότερη ακολουθία, συμπεριλαμβανομένων και των περιπτώσεων με άρτιο και περιττό μήκος, μπορούμε να συμπεράνουμε ότι θα λειτουργεί σωστά και για ακολουθίες μεγαλύτερου μήκους. Άρα, με επαγωγή αποδείξαμε ότι ο αλγόριθμος είναι ορθός.

Σχετικά με την πολυπλοκότητα του αλγορίθμου. Σε κάθε επίπεδο του δέντρου αναδρομής,

κάθε στοιχείο του πίνακα επεξεργάζεται μόνο μία φορά. Έτσι, συνολικά, έχουμε $O(n)$ επεξεργασία (συγχώνευση με γραμμική πολυπλοκότητα) σε κάθε επίπεδο του δέντρου. Εφόσον το ύψος του δέντρου είναι $logk$, ο συνολικός αριθμός των επιπέδων του είναι επίσης $logk$. Συνεπώς, η συνολική πολυπλοκότητα είναι $O(nlogk)$.

5 Πέμπτη άσκηση

5.1 Πρώτο ζητούμενο

Θα χρησιμοποιήσουμε τους εξής πίνακες:

1. **colors**: Αποθηκεύει την ποσότητα των μπαλών για κάθε χρώμα. Αποτελεί την μοναδική είσοδο του αλγορίθμου.

2. **boxes**: Αποθηκεύει κάθε κουτί με το περιεχόμενό του. Μπορούμε να έχουμε κλάση Box και ο πίνακας να περιλαμβάνει αντικείμενα της κλάσης. Αποτελεί την μοναδική έξοδο του αλγορίθμου.

Έστω N ο αριθμός των χρωμάτων. Επομένως, σύμφωνα με την περιγραφή του προβλήματος, ο αριθμός των κουτιών είναι επίσης N, ενώ ο συνολικός αριθμός των μπαλών είναι $20 * N$.

Ο αλγόριθμος περιγράφεται ως εξής:

1. Αρχικοποίησε του πίνακα **boxes** με 0 μπάλες σε κάθε κουτί.

2. Προσπέλασε τον πίνακα **boxes**.

3. Για το τρέχον κουτί, προσπέλασε τον πίνακα **colors** μέχρι να βρεις χρώμα i που να έχει το πολύ 20 μπάλες ($colors[i] \leq 20$).

4. Βάζουμε τις μπάλες στο τρέχον καλάθι, και τις μηδενίζουμε στον πίνακα **colors**.

   (a) Αν υπάρχει και άλλος χώρος στο καλάθι, ξαναπροσπέλασε τον πίνακα **colors** μέχρι να βρεις κάποιο χρώμα που μπορεί να γεμίσει το τρέχον κουτί. Αφαίρεσε από τον πίνακα **colors** όσες μπάλες χρησιμοποίησες από αυτό το χρώμα.

   (b) Αν δεν υπάρχει άλλος χώρος στο καλάθι, μην κάνεις τίποτα.

5. Πήγαινε στο επόμενο κουτί αν υπάρχει. Αν δεν υπάρχει τερμάτισε (η περίπτωση που υπάρχει μόνο ένα κουτί στο πρόβλημα).

   (a) Αν δεν είναι το τελευταίο, πήγαινε σε αυτό και ξεκίνα από το Βήμα 3.

   (b) Αν είναι το τελευταίο, βάλε σε αυτό τις υπόλοιπες μπάλες.

6. Επίστρεψε τον πίνακα **boxes**.

Για παράδειγμα, ας δούμε τη σειρά με την οποία θα μπουν οι μπάλες σε κάθε κουτί σε ένα παράδειγμα τις εκφώνησης.

Έστω ότι έχουμε 4 κουτιά και 80 μπάλες: 6 κίτρινες, 13 κόκκινες, 28 μπλε και 33 πράσινες:

Στο Κουτί 1 θα μπουν οι 6 κίτρινες και μετά 14 μπλε καθώς το γεμίζουν ακριβώς. Άρα 13 κόκκινες, 14 μπλε και 33 πράσινες ακόμα.

Στο Κουτί 2 θα μπουν οι 13 κόκκινες και μετά 7 μπλε καθώς το γεμίζουν ακριβώς. Άρα 7 μπλε και 33 πράσινες ακόμα.

Στο Κουτί 3 θα μπουν οι 7 μπλε και μετά 13 πράσινες καθώς το γεμίζουν ακριβώς. Άρα 20 πράσινες ακόμα.

Στο Κουτί 4 θα μπουν οι υπόλοιπες μπάλες, δηλαδή οι 20 πράσινες.

5.2 Δεύτερο ζητούμενο

Για να αποδείξουμε την ορθότητα του αλγορίθμου, θα αποδείξουμε ότι ισχύουν οι τρεις παρακάτω προτάσεις:

1. Κάθε κουτί περιέχει μπάλες με το πολύ δύο διαφορετικά χρώματα.
2. Οι μπάλες τοποθετούνται σε κάποιο κουτί χωρίς να υπερβαίνεται το όριο των 20 μπαλών.
3. Κάθε μπάλα τοποθετείται σε κάποιο κουτί.

Κάθε κουτί περιέχει μπάλες με το πολύ δύο διαφορετικά χρώματα

Αυτό εξασφαλίζεται από τα βήματα 2, 3, 4 καθώς είτε κάποιο κουτί θα έχει μόνο μπάλες ενός χρώματος , είτε ακριβώς δυό χρωμάτων.

Οι μπάλες τοποθετούνται σε κάποιο κουτί χωρίς να υπερβαίνεται το όριο των 20 μπαλών

Αυτό προφανώς εξασφαλίζεται από τα βήματα 3, 4 καθώς βάζουν μπάλες σε κουτί είτε μέχρι να γεμίσει τελείως.

Κάθε μπάλα τοποθετείται σε κάποιο κουτί

Σύμφωνα με τα παραπάνω, έχουμε αποδείξει ότι ο αλγόριθμος θα τρέχει επιθυμητά μέχρι και το προτελευταίο κουτί. Θα χρειαστεί να εξετάσουμε ξεχωριστά το τελευταίο κουτί και να αποδείξουμε ότι μπορούν να μπουν σε αυτό όλες οι μπάλες που απομένουν χωρίς να παραβιάζεται κάποιος κανόνας (είτε σχετικά με τον αριθμό των μπαλών είτε σχετικά με τα χρώματά τους). Όταν φτάσουμε στο τελευταίο κουτί, θα έχουμε τις εξής περιπτώσεις:

1. Αν στο πρόβλημα υπάρχει $N = 1$ κουτί, τότε απλά θα μπουν οι μπάλες του μοναδικού χρώματος μέσα, οι οποίες σύμφωνα με την εκφώνηση θα είναι ακριβώς 20.

2. Αν στο πρόβλημα υπάρχουν $N > 1$ κουτιά, τότε όταν φτάσουμε στο τελευταίο κουτί, σύμφωνα με τα παραπάνω, όλα τα προηγούμενα κουτιά θα περιέχουν ακριβώς 20 μπάλες με κάθε κουτί να έχει το λιγότερο 1 χρώμα και το πολύ 2 διαφορετικά χρώματα. Άρα, αποδείξαμε ότι όταν φτάσουμε στο τελευταίο κουτί, θα έχουμε το πολύ 2 χρώματα να μοιράσουμε, πράγμα που ικανοποιεί τον κανόνα σχετικά με τη διαφορετικότητα των χρωμάτων σε κάθε κουτί. Επίσης, το άθροισμα των μπαλών που απομένουν γνωρίζουμε ότι θα είναι ακριβώς 20 εφόσον οι συνολικές μπάλες είναι $20 * N$ και στα προηγούμενα $N - 1$ κουτιά έχουν τοποθετηθεί ακριβώς 20 μπάλες.

Με αυτόν τον τρόπο, αποδεικνύεται ότι ο αλγόριθμος είναι ορθός.

5.3 Τρίτο ζητούμενο

Ας υπολογίσουμε την πολυπλοκότητα του κάθε βήματος και στο τέλος τη συνολική πολυπλοκότητα:

1. Αρχικοποίηση - $O(n)$
2. Προσπέλαση - $O(n)$
3. Προσπέλαση εσωτερική στο Βήμα 2 - $O(n)$
4. Προσθήκη μπαλών / υπολογισμοί - $O(1)$
   (a) Προσπέλαση εσωτερική στο Βήμα 2 - $O(n)$
   (b) -
5. Επόμενο κουτί αν υπάρχει - $O(1)$
   (a) Έλεγχος - $O(1)$
   (b) Προσθήκη μπαλών / υπολογισμοί - $O(1)$
6. Επιστροφή - $O(1)$

Αρχικά, υπάρχει αρχικοποίηση. Στη συνέχεια υπάρχει μια επανάληψη από το 1 μέχρι και το N, και μέσα σε αυτήν την επανάληψη υπάρχουν άλλες δύο επαναλήψεις από το 1 μέχρι και το N. Οι υπόλοιπες λειτουργίες γίνονται σε σταθερό χρόνο. Άρα η συνολική πολυπλοκότητα είναι $O(n + n * 2n)$, δηλαδή $O(n^2)$, όπου n ο αριθμός των χρωμάτων/κουτιών.