Δεύτερη προγραμματιστική εργασία - Μάθημα Αλγορίθμων

Νικόλαος Σμυρνιούδης (3170148)

17 Απριλίου 2019

1 Άσκηση 2.3

Το πρόβλημα χωρίζεται σε υποπροβλήματα $OPT(i)$ τα οποία αναπαριστούν τα ελάχιστα βήματα του βατράχου μέχρι να φτάσει στο φύλλο i. Με βάση την optimal substructutre ιδιότητα αυτών των προβλημάτων θα ισχύει για το πρόβλημα $OPT(i)$:

$$OPT(i) = \begin{cases} 0, & \text{αν } i \leq 0 \\ OPT(i) = min\{OPT(j) + 1 | j < i & \\ \text{ και το i είναι προσβάσιμο απο το φύλλο j}\}, & \text{αλλιώς} \end{cases}$$

1.1 Η ιδιότητα Optimal Substructure

Για τα φύλλα $(1, ..., n)$ εστω το βέλτιστο μονοπάτι $1 \rightarrow n$ και έστω ο προηγούμενος κόμβος απο το n στο βέλτιστο μονοπάτι y. Τότε το βέλτιστο μονοπάτι $1 \rightarrow n$ περιέχει και το βέλτιστο μονοπάτι $1 \xrightarrow{p_1} y$.

Εστω πως υπάρχει καποιο καλύτερο μονοπάτι $1 \xrightarrow{p_2} y$ που απαιτεί x_2 βηματα και έστω πως το p_1 απαιτέι x_1 βήματα για τα οποία ισχύει $x_1 > x_2$. Τότε το μονοπάτι $1 \xrightarrow{p1} y \rightarrow n$ θα αποτελείται απο $x_1 + 1$ βήματα που είναι πιο πολλά από τα βήματα του μονοπατιού $1 \xrightarrow{p2} y \rightarrow n$ με $x_2 + 1$ βήματα κάτι που είναι άτοπο επειδή εξ'υποθέσεως το $1 \xrightarrow{p1} y \rightarrow n = 1 \rightarrow n$ είναι το βέλτιστο μονοπάτι.

1.2 Πολυπλοκότητα

Το πρόγραμμα εκτός απο το $OPT(i)$ του κάθε προβλήματος κρατάει για το φύλλο i και το $prev(i)$ το οποίο αναπαριστά τον προηγούμενο κόμβο στο βέλτιστο μονοπάτι απο την αρχή μεχρι το φύλλο i.

Ο αλγόριθμος γεμιζει τον πίνακα OPT με ένα βρόχο και σε καθε κελί του πίνακα εκτελει το min σε ενα εμφωλιασμένο βρόχο. Αυτοί οι υπολογισμοί συνολικά χρειάζονται $\mathcal{O}(n^2)$ χρόνο. Στον τελευταίο βρόχο του προγράμματος με την βοήθεια μιας στοίβας υπολογίζεται η λύση με έναν βρόχο στον πίνακα OPT.

Συνολικά δηλαδή η πολυπλοκότητα του αλγορίθμου είναι

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

2 Άσκηση 2.4

Το πρόβλημα χωρίζεται σε υποπροβλήματα $OPT(i, j)$ τα οποία αναπαριστούν την βέλτιστη λύση με τα αντικείμενα $1, ..., i$ (στην διαταξη που δίνονται) και ζητούμενες θερμίδες j. Η αναδρομική λύση με βάση την optimal substructure ιδιότητα είναι:

$$OPT(i,j) = \begin{cases} 0, \text{ αν } i \leq 0 \text{ η } j \leq 0 \\ max\{OPT(i-1,j), OPT(i-1, j-cals(i)+cals(i)\}, & \text{αλλιώς} \end{cases}$$

Ως μια λύση θεωρείται μία δυάδα (x, y) οπου x είναι οι θερμίδες και y τα λιπαρά. Οι λύσεις στο max συγκρίνονται ως εξής:

Έστω δύο λύσεις για το πρόβλημα. Η $a = (x_1, y_1)$ και η $b = (x_2, y_2)$. Η λύση a είναι καλύτερη απο την b (α > b) αν ($x_1 > x_2$ η (αν $x_1 = x_2$ τότε $y_1 < y_2$)).

2.1 Η ιδιότητα Optimal Substructure

Η βέλτιστη λύση για το πρόβλημα με τα φαγητά $(1, ..., n)$ και ζητούμενες θερμίδες W (η οποία είναι οι συχνότητες $(f_1, ..., f_n)$) περιέχει την βέλτιστη λύση για τα φαγητά $(1, .., n-1)$ με ζητούμενες θερμίδες $W - f_n * cals(n)$.

Έστω πως υπάρχει καποιά καλύτερη λύση $(f'_1, ..., f'_{n-1})$ για το υποπρόβλημα και παράγει μια λύση με (x_2, y_2). και έστω πως η $(f_1, ..., f_{n-1})$ παράγει μια λύση (x_1, y_1) για το ίδιο υποπρόβλημα. Τότε για τις δύο αυτές λύσεις θα ισχύει:

$$(x_1, y_1) < (x_2, y_2) \tag{1}$$
$$(x_1 + cals(n), y_1 + fats(n)) < (x_2 + cals(n), y_2 + fats(n)) \tag{2}$$
$$(\text{αρχική λύση}) < (\text{υποθετική λύση} + f_1 * \text{ αντικείμενο } 1)\} \tag{3}$$

Το οποίο είναι άτοπο γιατι εξ'υποθέσεως η αρχική λύση είναι βέλτιστη.

2.2 Πολυπλοκότητα

Ο Αλγόριθμος λειτουργεί γεμίζοντας τον πίνακα OPT μεγέθους $n \times W$ οπου n to πλήθος διαφορετικών αντικειμένων και W οι ζητούμενες θερμίδες, ετσι ώστε όταν υπολογίζει την βελιστη λύση του υποπροβλήματος $OPT(i, j)$ οι δύο συνιστώσες του σύμφωνα με την αναδρομική εξίσωση να έχουν ήδη υπολογιστεί. Ο υπολογισμός της max συνάρτησης γίνεται ομώς σε $\mathcal{O}(1)$ χρόνο αφού πρεπει να συγκριθούν μόνο δύο λύσεις. Άρα ο υπολογισμός των βέλτιστων θερμιδών και λιπαρών γίνεται σε $\mathcal{O}(n*W)$ χρόνο (ψευδοπολυωνυμικός).

Ο υπολογισμός της λύσης γίνεται με μια στοίβα. Στην αρχή το πρώτο υποψήφιο αντικείμενο είναι το τελευταίο δηλαδή το n (στην διάταξη που δίνονται) με ζητούμενες θερμίδες τις wantedCalories. Αν απο τις δύο συνιστώσες υπερισχύει η λύση που δεν περιέχει το αντικείμενο n (δηλαδή η $OPT(n-1,j)$ τότε δεν προστίθεται στην στοίβα το αντικείμενο n. Αν απο την άλλη υπερισχύσει η λύση που περιέχει το αντικείμενο n (δηλαδή η $OPT(n-1,j-cals(n))$) τότε προστίθεται στην στοίβα το αντικείμενο n. Ετσι ο υπολογισμός συνεχίζει αναδρομικά στο υποπρόβλημα που έδωσε την καλύτερη λύση στο αρχικό πρόβλημα. Τελικά κάθε αντικείμενο θα εξεταστεί το πολύ μια φορά και συνεπώς ο υπολογισμός της λύσης θα είναι $\mathcal{O}(n)$.

Συνολικά ο χρόνος θα είναι:

$$\mathcal{O}(n*W)+\mathcal{O}(n)=\mathcal{O}(n*W)$$

3 Άσκηση 2.5

Algorithm 1 ελαχιστασημεία(X)

Require: Σύνολο είναι σύνολο διαστημάτων $[s_i, f_i]$

ταξινόμηση X ως πρός τα f_i
$Y = \emptyset$
while X δεν είναι κενο **do**
  I = min(X);
  τοποθέτησε ενα σημείο x στο $I.f_i$
  $Y = Y \cup \{x\}$
  αφαίρεσε απο το όλα τα σημεία που επικαλύπτονται με το x
end while
return Y

3.1 Απόδειξη ορθότητας

Η απόδειξη γίνεται επαγωγικά ως προς τον αριθμό διαστημάτων. Για $n=1$ το αποτέλεσμα ισχύει με τετριμμένο τρόπο.

Έστω πως ο άπληστος είναι βέλτιστος για διαστήματα σε πλήθος μικρότερο ή ίσο με k

Για πλήθος διαστημάτων $k+1$ Ο αλγόριθμος με είσοδο διαστήματα $[s_1, f_1], ..., [s_{k+1}, f_{k+1}]$ (αύξουσα σειρά με βάση τα f_i) επιλέγει ως πρωτο σημείο το f αφαιρεί τα διαστηματα που επικαλύπτονται με αυτό το σημείο και συνεχίζει αναδρομικα.

Ο βέλτιστος θα πρέπει να έχει επιλέξει ενα σημείο που να καλύπτει το πρώτο διάστημα $s_1 <= x <= f_1$. Ωστόσο επειδη το f_1 είναι μικρότερο απο όλα τα άλλα f_i ο απληστος αλγόριθμος αποκλείεται να καλύπτει λιγότερα διαστήματα (η διαφορετικά) απο τον βέλτιστο γιατι αυτό θα σήμαινε πως τουλάχιστον ενα διαστημα εκτός του πρώτου τελειώνει πριν το f_1 που είναι άτοπο. Για τον άπληστο αλγόριθμο θα μένουν αλλα $y_1 <= k$ διαστήματα να καλύψει ενώ για τον βέλτιστο $y_2 <= k$ και ισχύει πως $y_1 <= y_2$ και τα διαστήματα του βέλτιστου ειναι υποσύνολο των διαστημάτων του απλήστου. Με βάση την επαγωγική υπόθεση όμως ο άπληστος για αυτα τα y_1 διαστήματα βρίσκει τον βέλτιστο αριθμό σημείων m και επειδή τα διαστήματα του απλήστου είναι υποσύνολο των διαστημάτων που έχει να καλύψει ακόμα ο βέλτιστος θα ισχύει $m <= n$ οπου n το πλήθος σημείων που επιλέγει για τα υπόλοιπα διαστήματα ο βέλτιστος. Ετσι ομως θα ισχύει:

$$m \leq n \tag{4}$$

$$m + 1 \leq n + 1 \tag{5}$$

$$\text{πληθος σημείων απλήστου} \leq \text{πλήθος σημείων βελτίστου} \tag{6}$$

Και επειδή ισχύει το αντίστροφο απο την ιδιότητα του βέλτιστου αλγορίθμου θα ισχύει πληθος σημείων απλήστου = πλήθος σημείων βελτίστου. Τέλος με την ολοκλήρωση της επαγωγής θα ισχύει πως ο άπληστος αλγόριθμος βρίσκει την βέλτιστη λύση για κάθε πλήθος διαστημάτων.

3.2 Πολυπλοκότητα

Η ταξινόμηση μπορεί να γίνει σε $\mathcal{O}(nlogn)$ χρόνο. Υστερα ο βρόχος while εκτελείται το πολύ n φορές. Μέσα στον βρόχο μπορούν να εκτελεστούν όλες οι εντολές σε $\mathcal{O}(1)$ χρόνο (με σωστή υλοποίηση) εκτός απο την αφαίρεση των επικαλυπτόμενων διαστημάτων η οποία εκτελείται σε $\mathcal{O}(n)$ χρόνο (μια σάρωση του X) και την έυρεση του min. Συνολικά λοιπόν ο αλγόριθμος τρέχει σε χρόνο πολυπλοκότητας $\mathcal{O}(n^2)$ ως πρός το πλήθος διαστημάτων εισόδου n.

4 Άσκηση 2.6

4.1 Απόδειξη ορθότητας

Η απόδειξη γίνεται με επαγωγή ως προς το πλήθος σημείων n στο Y.

Αν n = 1. Ο αλγόριθμος παράγει τετριμμένα το βέλτιστο αποτέλεσμα.

Έστω πως ισχύει για $n <= k$. Έστω ένα πρόβλημα με $k + 1$ σημεία $x_1, ..., x_{k+1}$. Για να καλύψει το σημείο x_1 ο αλγόριθμος επίλέγει το διάστημα y_1 που περιέχει το x_1 και έχει το μεγαλύτερο δυνατό χρόνο λήξης f_1. Στην λύση του βέλτιστου αλγορίθμου το διάστημα y_2 που επιλέγεται θα καλύπτει το x_1 και θα έχει χρόνο λήξης f_2. Θα ισχύει όμως $f_1 >= f_2$ με βάση την άπληστη επιλογή. Αυτο σημαίνει τα σημεία που μένει να καλύψει

Algorithm 2 ελαχισταδιαστήματα(X, Y)

Require: Σύνολο Y είναι σύνολο διαστημάτων $[s_i, f_i]$, Συνολο X είναι σύνολο σημείων x_i

1: ταξινόμηση X ως πρός τα x_i
2: $S = \emptyset$
3: **while** X δεν είναι κενο **do**
4: $x = \min(X)$;
5: εστω y το διάστημα που καλύπτει το x και εχει το μεγαλύτερο δυνατό χρόνο λήξης f (αν δεν υπάρχει επέστρεψε το κένο σύνολο και λήξε)
6: αφαίρεσε απο το X όλα τα σημεία που επικαλύπτονται με το y
7: $S = S \cup \{y\}$
8: **end while**
9: **return** S

ο βέλτιστος αλγόριθμος είναι υποσύνολο των σημείων που μένει να καλύψει ο άπληστος γιατι το διάστημα του απλήστου καλύπτει τουλάχιστον όσα σημεία καλύπτει το διάστημα του βελτίστου. Αν μένουν για τον απληστο u_1 σημεία, για τον βέλτιστο θα μενουν u_2 σημεία (εκ των οποίων και του απλήστου). Ο απληστος θα χρειαστεί εναν αριθμό m διαστημάτων ενώ ο βέλτιστος n αλλα απο την επαγωγική υπόθεση ο άπληστος για τα u_1 αυτά διαστήματα θα βρεί τον βέλτιστο αριθμο διαστημάτων και επειδη αυτα τα σημεία θα περιέχονται και στα απομείναντα σημεία του βελτίστου θα ισχύει:

$$m \leq n \tag{7}$$

$$m + 1 \leq n + 1 \tag{8}$$

$$\text{πληθος διαστημάτων απλήστου} \leq \text{πλήθος διαστημάτων βελτίστου} \tag{9}$$

Και επειδή ισχύει το αντίστροφο απο την ιδιότητα του βέλτιστου αλγορίθμου θα ισχύει πληθος σημείων απλήστου = πλήθος σημείων βελτίστου. Τέλος με την ολοκλήρωση της επαγωγής θα ισχύει πως ο άπληστος αλγόριθμος βρίσκει την βέλτιστη λύση για κάθε πλήθος σημείων.

4.2 Πολυπλοκότητα

Έστω n το πλήθος σημείων και m το πλήθος διαστημάτων. Η ταξινόμηση γίνεται σε χρόνο $\mathcal{O}(nlogn)$. Ο βρόχος επαναλαμβάνεται το πολύ n φορές. Μέσα στον βρόχο η γραμμή 4 εκτελείται σε χρόνο $\mathcal{O}(n)$ (απαιτείται σάρωση). Η έυρεση του y εκτελείται με μια σάρωση του πίνακα διαστημάτων σε $\mathcal{O}(m)$ χρόνο και τέλος η γραμμή 7 με την βοήθεια μιας στοίβας μπορεί να εκτελεσθεί σε χρόνο $\mathcal{O}(1)$.

Συνολικά δηλαδή η πολυπλοκότητα του προγράμματος είναι:

$$\mathcal{O}(nlogn) + \mathcal{O}(n) * (\mathcal{O}(n) + \mathcal{O}(m) + \mathcal{O}(1)) = \tag{10}$$

$$= \mathcal{O}(nlogn) + \mathcal{O}(n^2) + \mathcal{O}(n * m) \tag{11}$$

$$= \mathcal{O}(n^2) + \mathcal{O}(n * m) \tag{12}$$