Εθνικό Μετσόβιο Πολυτεχνείο
Σχολή ΗΜ&ΜΥ
Αλγόριθμοι και Πολυπλοκότητα
7^{ο} εξάμηνο, Ροή Λ
Ακαδημαϊκή περίοδος: 2011-2012

2^{η} Σειρά Γραπτών Ασκήσεων

Γερακάρης Βασίλης
<vgerak@gmail.com>
Α.Μ.: 03108092

10 Ιανουαρίου 2012

1 Επιτροπή Αντιπροσώπων (KT 4.15)

Ταξινομούμε τον πίνακα σε αύξουσα σειρά με βάση τα f_i. Βρίσκουμε το διάστημα (έστω m) με το μέγιστο f_m που η αρχή του s_m βρίσκεται πρίν το τέλος του 1ου, f_1 και το επιλέγουμε ως αντιπρόσωπο. Θα αποδείξουμε ότι η επιλογή που κάνουμε (σε κάθε βήμα) είναι και η βέλτιστη δυνατή.

Έστω k, ένας διαφορετικός αντιπρόσωπος, που είναι η βέλτιστη επιλογή. Δεν είναι δυνατόν να έχει δείκτη $k > m$, αφού έτσι δε θα επικάλυπτε το 1ο διάστημα (εξ'ορισμού του m). Αφού λοιπόν $k \leq m$ θα ισχύει και $f_k \leq f_m$. Άρα ο k επικαλύπτει **το πολύ** όσα διαστήματα επικαλύπτει η επιλογή μας, επομένως η επιμέρους λύση μας είναι η βέλτιστη. Επαγωγικά προκύπτει η ορθότητα του αλγορίθμου μας.

Algorithm 1 Άσκηση 1

Sort A on ascending order using f_i as key
procedure RepresentantivesSelect(A, N)
    if $N = 0$ **then**
        return $RepresentativesList$
    else
        $i \leftarrow 1$
        $max \leftarrow 0$
        $posmax \leftarrow 0$
        while $i \leq N$ **do**
            if $f_i > max$ **and** $s_i < f_1$ **then**
                $max \leftarrow f_i$
                $posmax \leftarrow i$
        $RepresentativesList.append(posmax)$
        $j \leftarrow posmax$
        while max **>** s_j **do**
            $A.remove(j)$
            $N \leftarrow N - 1$
            $j \leftarrow j + 1$
        for $j \leftarrow posmax$ **to** 1 **step** -1 **do**
            $A.remove(j)$
            $N \leftarrow N - 1$
        return RepresentativesSelect (A, N)

2 Βιαστικός Μοτοσυκλετιστής

Ταξινομούμε τα διαστήματα σε αύξουσα σειρά με βάση τα όρια ταχυτήτων v_i και δημιουργούμε τον πίνακα χρόνων t, όπου $t_i = l_i/v_i$. Έστω T ο χρόνος που θα ξεπεράσουμε το όριο ταχύτητας και u τα χιλιόμετρα κατά τα οποία θα το ξεπεράσουμε. Έαν $T \leq t_1$, τότε κάνουμε T λεπτά με $v_1 + u$ km/h, και τα υπόλοιπα με το όριο ταχύτητας, αλλιώς κάνουμε t_1 λεπτά υπερβαίνοντας το όριο και επαναλαμβάνουμε τη διαδικασία για $T - t_1$ λεπτά για το επόμενο διάστημα.

Θα αποδείξουμε ότι η επιλογή που κάνουμε σε κάθε βήμα, είναι η βέλτιστη δυνατή. Ο συνολικός χρόνος που χρειάζεται για να φτάσει στον προορισμό του είναι:

$T_{total} = \sum_{i=1}^{n} t_i = \sum_{i=1}^{n} \frac{l_i}{u_i}$.

Η μέγιστη (ποσοστιαία) ελάττωση των χρονικών διαστημάτων προκύπτει αν προσθέσουμε την ταχύτητα u στο μικρότερο παρονομαστή v_i. Καθώς λοιπόν όλες οι τιμές είναι δεδομένες και σταθερές (T, v_i, l_i), οποιοδήποτε άλλο διάστημα αν επιλέγαμε, θα έδινε ποσοστιαίο κέρδος χρόνου **το πολύ** όσο αυτό που υπολογίσαμε, επομένως η επιμέρους λύση μας είναι η βέλτιστη.

Algorithm 2 Άσκηση 2

Sort A on ascending order using v_i as key
for $i \leftarrow 1$ **to** N **do**
    $t_i \leftarrow l_i/v_i$
procedure TimeSelect(A, N)
    $i \leftarrow 1$
    while $T \geq t_i$ **do**
        $Selection.append(i, t_i)$
        $T \leftarrow T - t_i$
        $i \leftarrow i + 1$
    $Selection.append(i, T)$
    return $Selection$

Η πολυπλοκότητα του αλγορίθμου είναι $\Theta(n \log n)$, όσο χρειάζεται η ταξινόμηση. Στον ταξινομημένο πίνακα το αποτέλεσμα προκύπτει σε γραμμικό χρόνο. Στην περίπτωση που η υπέρβαση στο όριο γινόταν κατά παράγοντα α > 1, η επιλογή του διαστήματος δε θα επηρρέαζε το αποτέλεσμα, αφού το ποσοστιαίο κέρδος θα ήταν το ίδιο για όλα τα διαστήματα.

3 Βότσαλα στη Σκακιέρα (DVP 6.5)

3.1 Σύγκριση με άπληστο αλγόριθμο

Ο άπληστος αλγόριθμος μας εγγυάται ότι θα πάρει τουλάχιστον το 25% της βέλτιστης λύσης. Το χειρότερο πιθανό σενάριο για την άπληστη επιλογή είναι ένα grid της παρακάτω μορφής, με N ένα μεγάλο θετικό αριθμό:

0	N-1	0
N-1	N	N-1
0	N-1	0
0	0	0

Στην περίπτωση αυτή ο λόγος της άπληστης λύσης προς τη βέλτιστη είναι $Q = \frac{N}{4N-4}$. Το όριο αυτής της ποσότητας όταν $N \to \infty$ είναι 1/4 ή 25%.

3.2 Βέλτιστη λύση με χρήση δυναμικού προγραμματισμού

Έστω A,B,C,D οι 4 σειρές του προβλήματος. Μπορούμε να δημιουργήσουμε 8 διαφορετικούς έγκυρους συνδυασμούς επιλογής νομισμάτων, καθένας από τους οποίους είναι συμβατός με ορισμένους, από την επιλογή της προηγούμενης στήλης.

i	Selection	Profit(k)	Valid Prev. Status	Valid(k-1)
0	$\emptyset$	0	ALL	0,1,2,3,4,5,6,7
1	A	Val[A]	$\emptyset$,B,C,D,(B,D)	0,2,3,4,6
2	B	Val[B]	$\emptyset$,A,C,D,(A,C),(A,D)	0,1,3,4,5,7
3	C	Val[C]	$\emptyset$,A,B,D,(B,D),(A,D)	0,1,2,4,6,7
4	D	Val[D]	$\emptyset$,A,B,C,(A,C)	0,1,2,3,4
5	(A,C)	Val[A] + Val[C]	$\emptyset$,B,D,(B,D)	0,2,4,6
6	(B,D)	Val[B] + Val[D]	$\emptyset$,A,C,(A,C)	0,1,3,5
7	(A,D)	Val[A] + Val[D]	$\emptyset$,B,C	0,2,3

Θα λύσουμε το πρόβλημα με χρήση δυναμικού προγραμματισμού. Θεωρούμε τον πίνακα $P[0..N][0..7]$, όπου κάθε γραμμή αντιστοιχεί στον αντίστοιχο συνδυασμό που καταγράφεται στον παραπάνω πίνακα. Δεδομένης της σκακιέρας Val (4 x N) με τις αντίστοιχες τιμές, προκύπτει ο παρακάτω αλγόριθμος:

Algorithm 3 Άσκηση 3

```
procedure OptimalPebbling(Val, N)
    for i ← 0 to 7 do
        P[0][i] ← 0
    for i ← 1 to N do
        for j ← 0 to 7 do
            P[i][j] = Profit[i] + max{∀k ∈ Valid(j − 1), P[i − 1][k]}
    OPT = 0
    for j ← 0 to 7 do
        if OPT < P[N][i] then
            OPT ← P[N][i]
    return OPT
```

Υπάρχουν 41 διαφορετικοί συνδυασμοί σε κάθε βήμα, οπότε η χρονική πολυπλοκότητα του αλγορίθμου είναι: $\Theta(41 * N) = \Theta(N)$

4 Χωρισμός Κειμένου σε Γραμμές

Θεωρούμε τη συνάρτηση κόστους γραμμής που περιέχει τις λέξεις από i εώς και j ως

$$v(i, j) = s_k^2 = (C + 1 - \sum_{p=1}^{j} (l_p + 1))^2$$

Στην περίπτωση που η παραπάνω τιμή είναι αρνητική (δηλαδή η λέξη δε χωράει στη γραμμή), η v επιστρέφει ∞.

Ορίζουμε την αναδρομική σχέση: OPT(j) = $\begin{cases} v(1, j) & \text{if } c(1, j) < \infty \\ \min\limits_{1 \leq k < j} (f(k) + v(k + 1, j)) & \text{if } c(1, j) = \infty \end{cases}$

Και με χρήση δυναμικού προγραμματισμού προκύπτει η λύση σε χρονο O(j^2).

Αξίζει να σημειωθεί ότι αν κάναμε χρήση των ευρημάτων των Galil και Park (όπως δημοσιεύτηκαν στο paper A linear-time algorithm for concave one-dimensional dynamic programming), εκμεταλλευόμενοι τη φυσιολογία του προβήματος, θα μπορούσαμε να κατεβάσουμε την πολυπλοκότητα σε γραμμικό χρόνο.

5 Αντίγραφα Αρχείου (KT 6.12)

Έστω OPT(j) το ελάχιστο κόστος μιας διευθέτησης που καλύπτει τους servers 1 εώς j, θεωρώντας ότ τοποθετούμε ένα αντίγραφο στον server j. Ψάχνουμε τις πιθανές θέσεις (i) με i < j που μπορούμε να τοποθετήσουμε ένα αντίγραφο καταλήγωντας σε βέλτιστη λύση. Έστω ότι βρίσκεται στη θέση i = k. Το συνολικό κόστος που προκύπτει για το OPT(j) είναι ίσο με το άθροισμα:

* Του κόστους c_j

* Του OPT(k), αφού ώς το k θεωρούμε ότι επιλέξαμε βέλτιστα

* Του κόστους πρόσβασης για κάθε server από $i \to j$, το οποίο ισούται με

$$\sum_{i=k}^{j} b_k(j-k)$$

Από τις j (σε πλήθος) αυτές λύσεις, επιλέγουμε τη βέλτιστη, αυτή με το ελάχιστο κόστος. Επομένως η αναδρομική μας σχέση είναι η εξής:

$$OPT(j) = c_j + \min_{0 \le i \le j}(OPT(i) + \sum_{i=k}^{j} b_k(j-k))$$

Θεωρώντας ότι OPT(0) = 0 μπορούμε να παράγουμε τις τιμές του OPT αυξάνοντας το j. Η βέλτιστη λύση έχει τιμή OPT(n) και για να βρούμε τη διευθέτηση που την προκάλεσε κάνουμε ένα backtracking, διατηρώντας τα i από τα OPT(i) στοιχεία που επιλέγονταν.

Για τον υπολογισμό του κάθε j χρησιμοποιούμε O(j) χρόνο, άρα η τελική πολυπλοκότητα είναι $O(n^2)$.