Αλγόριθμοι & Πολυπλοκότητα

Σημειώσεις

Δευτέρα, 5 Δεκεμβρίου 2005

Greedy

Θα επιλύσουμε με άπληστο αλγόριθμο τον ακόλουθο περιορισμό του Discrete Knapsack (πρόβλημα σακιδίου ακεραίων τιμών).

Δίνονται: η χωρητικότητα W του σακιδίου και n αντικείμενα $x_1, x_2, \ldots, x_n$ με αντίστοιχες αξίες $p_1, \ldots, p_n$ και αντίστοιχα βάρη $w_1, \ldots, w_n$ τέτοια ώστε η σειρά των αντικειμένων ταξινομημένα κατά αύξον βάρος να είναι ίδια με τη σειρά των αντικειμένων ταξινομημένα κατά φθίνουσα αξία.

Ζητείται ένα υποσύνολο των αντικειμένων που το συνολικό τους βάρος να μην υπερβαίνει την τιμή W και η συνολική τους αξία να είναι όσο το δυνατόν μεγαλύτερη.

Χωρίς βλάβη της γενικότητας, έστω ότι το πρώτο αντικείμενο έχει τη μεγαλύτερη αξία, το δεύτερο την αμέσως επόμενη, κόκ. Ισχύει δηλαδή:

$$p_1 > p_2 > \ldots > p_n \tag{1}$$

Επειδή η σειρά των αντικειμένων αν τα ταξινομήσουμε κατά αύξον βάρος είναι ίδια, ισχύει επιπλέον:

$$w_1 < w_2 < \ldots < w_n \tag{2}$$

Κατά συνέπεια ο λόγος $\frac{p_i}{w_i}$ μειώνεται καθώς προχωρούμε από το πρώτο αντικείμενο στα επόμενα:

$$\frac{p_1}{w_1} > \frac{p_2}{w_2} > \ldots > \frac{p_n}{w_n} \tag{3}$$

Το άπληστο κριτήριο για την εύρεση λύσης στο πρόβλημα υποδεικνύει ότι πρέπει κάθε φορά να παίρνουμε το αντικείμενο που έχει το μέγιστο λόγο αξίας προς βάρος και ταυτόχρονα χωράει στο σακίδιο. Άρα ένας άπληστος αλγόριθμος θα σαρώνει τα αντικείμενα ξεκινώντας από το αντικείμενο 1 (αφού αυτό έχει το μεγαλύτερο λόγο αξίας προς βάρος) και θα παίρνει όσα αντικείμενα μπορεί μέχρι να μην χωράνε άλλα στο σακίδιο. Μόλις συναντήσει το πρώτο αντικείμενο που δεν χωράει στο σακίδιο μπορεί να σταματήσει, αφού όλα τα επόμενα αντικείμενα θα έχουν μεγαλύτερο βάρος (λόγω της εξ. 2).

Ο αλγόριθμος είναι ο εξής: (επιστρέφει τη συνολική αξία των αντικείμενων που επέλεξε)

```
1: i ← 1
2: capacity ← W
3: p ← 0
4: while capacity ≥ w_i and i ≤ n do
5:     πάρε το αντικείμενο x_i στη λύση
6:     capacity ← capacity − w_i
7:     p ← p + p_i
8:     i ← i + 1
9: end while
10: return p
```

Η πολυπλοκότητα χρόνου του παραπάνω αλγορίθμου είναι $O(n)$, αφού στη χειρότερη περίπτωση διατρέχει μία φορά όλα τα αντικείμενα[1].

Θα αποδείξουμε ότι η λύση που επιστρέφει ο αλγόριθμος είναι βέλτιστη. Κατ'αρχήν, παρατηρούμε ότι ο αλγόριθμος ξεκινάει από το πρώτο αντικείμενο και επιλέγει αντικείμενα μέχρι να μην χωράνε άλλα, άρα το σύνολο των αντικειμένων που επιλέγει είναι ένα σύνολο της μορφής:

$$G = \{x_1, x_2, \ldots, x_\rho\} \tag{4}$$

Έστω τώρα μία βέλτιστη λύση X, και ας υποθέσουμε ότι τα αντικείμενα του X ταξινομημένα κατά αύξον index είναι τα:

$$X = \{x_{\sigma_1}, x_{\sigma_2}, \ldots, x_{\sigma_k}\} \tag{5}$$

Έστω ότι τα αντικείμενα που ανήκουν στη βέλτιστη λύση είναι διαφορετικά από τα αντικείμενα που επιλέγει ο greedy αλγόριθμος, και ας υποθέσουμε ότι το αντικείμενο με δείκτη j είναι το πρώτο που ανήκει στη λύση G αλλά δεν ανήκει στη βέλτιστη λύση X. Αυτό σημαίνει ότι $\sigma_1 = 1$, $\sigma_2 = 2$, ..., $\sigma_{j-1} = j - 1$ και $\sigma_j > j$.

Επειδή $\sigma_j > j$, από τις σχέσεις 1 και 2 έπεται ότι το x_{σ_j} έχει μικρότερη αξία και μεγαλύτερο βάρος από το x_j. Άρα αν από τη βέλτιστη λύση X αφαιρέσουμε το x_{σ_j} και προσθέσουμε το x_j παίρνουμε μία λύση που εξακολουθεί να είναι εφικτή (αφού το αντικείμενο που προσθέσαμε έχει μικρότερο βάρος από αυτό που αφαιρέσαμε), ενώ ταυτόχρονα έχει και μεγαλύτερη συνολική αξία (αφού το αντικείμενο που προσθέσαμε έχει μεγαλύτερη αξία από αυτό που αφαιρέσαμε). Καταλήγουμε σε άτοπο, αφού είχαμε υποθέσει ότι η λύση X είναι βέλτιστη.

[1] Αν θεωρήσουμε ότι τα αντικείμενα δεν είναι αρχικά ταξινομημένα κατά φθίνοντα λόγο αξίας προς βάρος, τότε πρέπει πριν εκτελέσουμε τον παραπάνω αλγόριθμο να δαπανήσουμε χρόνο $O(n \log n)$ για να τα ταξινομήσουμε. Άρα η συνολική χρονική πολυπλοκότητα θα είναι $O(n \log n) + O(n) = O(n \log n)$.

Κατά συνέπεια τα αντικείμενα που ανήκουν στη βέλτιστη λύση είναι τα ίδια με αυτά που ανήκουν στη λύση του greedy αλγόριθμου. Άρα ο greedy αλγόριθμος δίνει βέλτιστη λύση.

Dynamic programming

Πρόβλημα εκτύπωσης

Θα θεωρήσουμε το πρόβλημα της "σωστής" εκτύπωσης μίας παραγράφου στον εκτυπωτή. Το κείμενο είναι μία ακολουθία n λέξεων $w_1, \ldots, w_n$ με αντίστοιχα μήκη $l_1, l_2, \ldots, l_n$ και θέλουμε να την εκτυπώσουμε στον εκτυπωτή. Κάθε γραμμή χωράει το πολύ W χαρακτήρες.

Αν μία γραμμή περιέχει τις λέξεις $w_i, \ldots, w_j$, το πλήθος των κενών που θα παραμείνουν στο τέλος της γραμμής είναι:

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

Θέλουμε η εκτύπωση να ελαχιστοποιεί το άθροισμα των κύβων των παραπάνω ποσοτήτων για όλες τις γραμμές, εκτός από την τελευταία.

Λύση με δυναμικό προγραμματισμό

Θα θεωρήσουμε στα επόμενα ότι δεν υπάρχει λέξη στην είσοδο που να μην χωράει σε μία γραμμή. Δηλαδή για κάθε $i : 1 \leq i \leq n$ ισχύει $l_i \leq W$.

Ας δούμε αρχικά γιατί ο άπληστος αλγόριθμος αποτυγχάνει να δώσει τη βέλτιστη λύση στο πρόβλημα της εκτύπωσης. Το άπληστο κριτήριο για αυτό το πρόβλημα υποδεικνύει μία λύση στην οποία γεμίζουμε κάθε φορά μία γραμμή μέχρις ότου να μην χωράει την επόμενη λέξη. Τότε αλλάζουμε γραμμή και συνεχίζουμε ομοίως. Έστω ότι δίνεται ως είσοδος η εξής ακολουθία λέξεων:

$$W = (\mathrm{a}, \mathrm{ab}, \mathrm{a}, \mathrm{abc}, \mathrm{abcd})$$

η οποία θέλουμε να εκτυπωθεί σε γραμμές με μέγιστο μήκος $W = 6$. Η λύση του άπληστου αλγόριθμου θα είναι η ακόλουθη:

```
|a ab a|
|abc   |
|abcd  |
```

και θα έχει κόστος $0^3 + 3^3 + 0 = 27$. Όμως, κατεβάζοντας το τελευταίο a της πρώτης γραμμής στη δεύτερη, παίρνουμε μία λύση με κόστος $2^3 + 1^3 + 0 = 9$:

```
|a ab  |
|a abc |
|abcd  |
```

Δοκιμάζουμε λοιπόν να λύσουμε το πρόβλημα με δυναμικό προγραμματισμό. Ορίζουμε αρχικά τα ακόλουθα μεγέθη:

- $\text{extras}\,[i,j]$, για $1 \leq i \leq j \leq n$: το πλήθος των κενών που θα μείνουν στο τέλος μίας γραμμής που περιέχει τις λέξεις $w_i, \ldots, w_j$. Ορίζουμε:

$$\text{extras}\,[i,j] = W - (j-i) - \sum_{k=i}^{j} l_k \tag{6}$$

  Παρατηρούμε ότι η ποσότητα αυτή μπορεί να είναι αρνητική, αν επιλέξουμε να τοποθετήσουμε στην ίδια γραμμή λέξεις οι οποίες δεν χωράνε σε μία γραμμή.

- $\text{linecost}\,[i,j]$, για $1 \leq i \leq j \leq n$: το κόστος που συνεισφέρει μία γραμμή που περιέχει τις λέξεις $w_i, \ldots, w_j$, στο άθροισμα που θέλουμε να ελαχιστοποιήσουμε. Ορίζουμε:

$$\text{linecost}\,[i,j] = \begin{cases} \infty & , \quad \text{αν extras}\,[i,j] < 0 \\ 0 & , \quad \text{αν } j = n \text{ και extras}\,[i,j] \geq 0 \\ (\text{extras}\,[i,j])^3 & , \quad \text{αλλιώς} \end{cases} \tag{7}$$

  Παρατηρούμε ότι αν οι λέξεις $w_i, \ldots, w_j$ δεν χωράνε στην ίδια γραμμή τότε το αντίστοιχο κόστος είναι άπειρο. Άρα δεν υπάρχει περίπτωση μία τέτοια διευθέτηση των λέξεων να αποτελεί μέρος βέλτιστης λύσης. Επίσης, το κόστος της γραμμής που περιέχει την τελευταία λέξη είναι 0, με την προϋπόθεση ότι οι λέξεις που έχουν τοποθετηθεί στην τελευταία γραμμή χωράνε στην ίδια γραμμή. Έτσι αγνοούμε το κόστος της τελευταίας γραμμής.

Η λύση που ψάχνουμε θα πρέπει να ελαχιστοποιεί το άθροισμα των linecost για όλες τις γραμμές του κειμένου. Τα υποπροβλήματα που θα μας οδηγήσουν στη βέλτιστη λύση είναι της μορφής:

"*Βρες μία βέλτιστη διευθέτηση των λέξεων* $w_1, \ldots, w_j$", για $j = 1, \ldots, n$

Πραγματικά, έστω μία βέλτιστη διευθέτηση των λέξεων $w_1, \ldots, w_j$, και ας υποθέσουμε ότι ξέρουμε πως σε αυτή τη διευθέτηση η τελευταία γραμμή περιέχει τις λέξεις $w_i, \ldots, w_j$. Τότε οι προηγούμενες γραμμές θα περιέχουν τις λέξεις $w_1, \ldots, w_{i-1}$, και μάλιστα διευθετημένες κατά βέλτιστο τρόπο[2].

Αν λοιπόν είναι $\text{cost}[j]$ το κόστος της βέλτιστης λύσης για τις λέξεις $w_1, \ldots, w_j$ στην οποία ξέρουμε ότι η τελευταία γραμμή περιέχει τις λέξεις

[2] Αν δεν ήταν βέλτιστα διευθετημένες, τότε θα μπορούσαμε να αλλάξουμε αυτές τις γραμμές ώστε να συνεισφέρουν λιγότερο στη συνάρτηση κόστους, παίρνοντας τελικά καλύτερη λύση για τις λέξεις $w_1, \ldots, w_j$.

$w_i, \ldots, w_j$, ισχύει[3]:

$$\text{cost}\,[j] = \text{cost}\,[i-1] + \text{linecost}\,[i,j]$$

Φυσικά, ποτέ δεν είναι γνωστό ποια θα είναι η πρώτη λέξη της τελευταίας γραμμής. Άρα πρέπει να δοκιμάσουμε όλες τις πιθανές επιλογές για το i: η πρώτη λέξη της τελευταίας γραμμής μπορεί να είναι οποιαδήποτε από τις $w_1, \ldots, w_j$. Τελικά το $\text{cost}\,[j]$ για $j = 0, 1, \ldots, n$ μπορεί να οριστεί αναδρομικά ως εξής:

$$\text{cost}\,[j] = \begin{cases} 0 & , \quad \text{αν } j = 0 \\ \min_{1 \le i \le j} \left(\text{cost}\,[i-1] + \text{linecost}\,[i,j]\right) & , \quad \text{αν } j > 0 \end{cases} \tag{8}$$

1: **for** $i \leftarrow 1$ **to** n **do** ▷ Υπολογισμός των $\text{extras}\,[i,j]$
2: $\quad \text{extras}\,[i,i] \leftarrow W - l_i$
3: $\quad$ **for** $j \leftarrow i+1$ **to** n **do**
4: $\quad\quad \text{extras}\,[i,j] \leftarrow \text{extras}\,[i,j-1] - l_j - 1$
5: $\quad$ **end for**
6: **end for**
7: **for** $i \leftarrow 1$ **to** n **do** ▷ Υπολογισμός των $\text{linecost}\,[i,j]$
8: $\quad$ **for** $j \leftarrow i$ **to** n **do**
9: $\quad\quad$ **if** $\text{extras}\,[i,j] < 0$ **then**
10: $\quad\quad\quad \text{linecost}\,[i,j] \leftarrow \infty$
11: $\quad\quad$ **else if** $j = n$ **and** $\text{extras}\,[i,j] \geq 0$ **then**
12: $\quad\quad\quad \text{linecost}\,[i,j] \leftarrow 0$
13: $\quad\quad$ **else**
14: $\quad\quad\quad \text{linecost}\,[i,j] \leftarrow (\text{extras}\,[i,j])^3$
15: $\quad\quad$ **end if**
16: $\quad$ **end for**
17: **end for**
18: $\text{cost}\,[0] \leftarrow 0$ ▷ Υπολογισμός των $\text{cost}\,[j]$,$p\,[j]$
19: **for** $j \leftarrow 1$ **to** n **do**
20: $\quad \text{cost}\,[j] \leftarrow \infty$
21: $\quad$ **for** $i \leftarrow 1$ **to** j **do**
22: $\quad\quad$ **if** $\text{cost}\,[i-1] + \text{linecost}\,[i,j] < \text{cost}\,[j]$ **then**
23: $\quad\quad\quad \text{cost}\,[j] \leftarrow \text{cost}\,[i-1] + \text{linecost}\,[i,j]$
24: $\quad\quad\quad p\,[j] \leftarrow i$
25: $\quad\quad$ **end if**
26: $\quad$ **end for**
27: **end for**

---

[3]Για να υπολογίσουμε με αυτόν τον αναδρομικό τύπο την τιμή $\text{cost}\,[1]$ μας χρειάζεται η τιμή $\text{cost}\,[0]$, την οποία μπορούμε να θέσουμε ίση με 0.

Ο αλγόριθμος που λύνει το πρόβλημα με βάση την παραπάνω αναδρομική σχέση, αρχικά υπολογίζει τις τιμές $\text{extras}[i,j]$, για $1 \leq i \leq j \leq n$. Στη συνέχεια υπολογίζει τις τιμές $\text{linecost}[i,j]$ για $1 \leq i \leq j \leq n$. Τέλος, υπολογίζει τα κόστη $\text{cost}[j]$ και έναν πίνακα p του οποίου η τιμή $p[j]$ είναι ο δείκτης της λέξης που βρίσκεται στην αρχή της γραμμής στην οποία βρίσκεται η λέξη w_j. Έτσι, μετά την ολοκλήρωση του υπολογισμού του $\text{cost}[n]$, ξέρουμε ότι η τελευταία γραμμή περιέχει τις λέξεις $w_{p[n]}, \ldots, w_n$, η προτελευταία γραμμή περιέχει τις λέξεις $w_{p[p[n]-1]}, \ldots, w_{p[n]-1}$, κόκ.

Είναι ξεκάθαρο ότι τόσο η χρονική όσο και η χωρική πολυπλοκότητα του αλγορίθμου είναι $\Theta\left(n^2\right)$. Αν παρατηρήσουμε ότι κάθε γραμμή χωράει το πολύ $\left\lceil \frac{W}{2} \right\rceil$ λέξεις[4], τότε μπορούμε να βελτιώσουμε τον αλγόριθμο ώστε να έχει χρονική και χωρική πολυπλοκότητα $\Theta(nW)$. Μία γραμμή που περιέχει τις λέξεις $w_i, \ldots, w_j$ περιέχει ακριβώς $j - i + 1$ λέξεις. Με βάση την παραπάνω παρατήρηση, ξέρουμε ότι αν $j - i + 1 > \left\lceil \frac{W}{2} \right\rceil$ τότε $\text{linecost}[i,j] = \infty$. Άρα ο αλγόριθμος χρειάζεται να υπολογίζει και να αποθηκεύει μόνο τις τιμές των $\text{extras}[i,j]$ και $\text{linecost}[i,j]$ για $j - i + 1 \leq \left\lceil \frac{W}{2} \right\rceil$. Επίσης, το εσωτερικό **for** loop στον υπολογισμό των $\text{cost}[j]$,$p[j]$ χρειάζεται να διατρέχει μόνο τις τιμές από $\max\left\{1, j - \left\lceil \frac{W}{2} \right\rceil + 1\right\}$ ως j.

- Πώς μπορούμε να βελτιώσουμε τη χωρική πολυπλοκότητα ακόμη περισσότερο, σε τάξη μεγέθους $\Theta(n)$;

[4] Αυτό συμβαίνει επειδή κάθε λέξη καταλαμβάνει τουλάχιστον ένα χαρακτήρα, και υπάρχει πάντα ένα κενό μεταξύ δύο διαδοχικών λέξεων στην ίδια γραμμή.