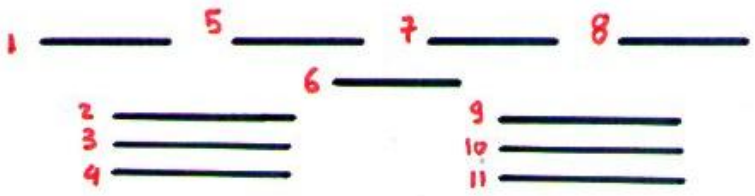
Εθνικό Μετσόβιο Πολυτεχνείο
Σχολή Ηλεκτρολόγων Μηχανικών & Μηχανικών Υπολογιστών
Αλγόριθμοι & Πολυπλοκότητα – 7ο Εξάμηνο 2018-2019
Αβραμίδης Κλεάνθης – ΑΜ: 03115117

2η Γραπτή Άσκηση – Απαντήσεις

➢ *Θέμα 1*

a) Για τα δοθέντα κριτήρια έχουμε:

1. *Λιγότερες Επικαλύψεις*: Δεν οδηγεί σε βέλτιστη λύση. Το δείχνουμε με αντιπαράδειγμα:

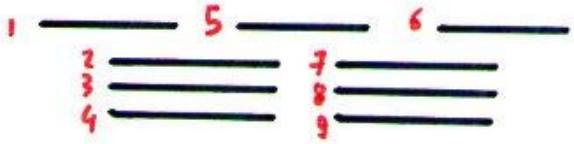


Στην παραπάνω διάταξη, η μόνη βέλτιστη λύση είναι η {1,5,7,8}. Σύμφωνα με τον προτεινόμενο αλγόριθμο, αρχικά θα επιλεχθεί το μάθημα 6 καθώς έχει τις λιγότερες επικαλύψεις. Ταυτόχρονα θα αφαιρεθούν τα μαθήματα 5,7 που το επικαλύπτουν, τα οποία όμως είναι και στοιχεία της βέλτιστης λύσης. Συνεπώς το κριτήριο δεν είναι ορθό.

2. *Μεγαλύτερη Διάρκεια*: Δεν οδηγεί σε βέλτιστη λύση. Το δείχνουμε με αντιπαράδειγμα:

Στην παραπάνω διάταξη, η μόνη βέλτιστη λύση είναι η {1,3} αλλά ο προτεινόμενος αλγόριθμος αφαιρεί το μάθημα 3, που έχει τη μεγαλύτερη διάρκεια. Συνεπώς το κριτήριο δεν είναι ορθό.

3. *Περισσότερες Επικαλύψεις*: Δεν οδηγεί σε βέλτιστη λύση. Το δείχνουμε με αντιπαράδειγμα:



Στην παραπάνω διάταξη, η μόνη βέλτιστη λύση είναι η {1,5,6}. Σύμφωνα με τον προτεινόμενο αλγόριθμο, αρχικά θα αφαιρεθεί το μάθημα 5 καθώς έχει τις περισσότερες επικαλύψεις. Το 5 ωστόσο είναι στοιχείο της βέλτιστης λύσης, συνεπώς ούτε και αυτό το κριτήριο είναι ορθό.

Για την περίπτωση κατά την οποία τα μαθήματα δίνουν διδακτικές μονάδες και στοχεύουμε στη μεγιστοποίησή τους, θα προσαρμόσουμε την στρατηγική του ελάχιστου χρόνου ολοκλήρωσης ως εξής:

1) Ταξινομούμε τα n μαθήματα κατά αύξοντα χρόνο ολοκλήρωσης.
2) Αρχικοποιούμε πίνακα $A[n]$ στον οποίο το $A[i]$, όπου $i = \{1,2,\dots,n\}$ θα δηλώνει το μέγιστο αριθμό διδακτικών μονάδων που προκύπτει από το σύνολο των πρώτων i μαθημάτων, καθώς και μία λίστα.
3) Υλοποιούμε την εξής αναδρομική συνάρτηση:

$$A[i] = \begin{cases} points[i] & if\ i = 1 \\ \max\{\, A[i-1], A[j] + points[i] \,\} & if\ i \neq 1 \end{cases}$$

όπου j το αμέσως προηγούμενο μάθημα που δεν επικαλύπτεται με το i. Σε περίπτωση που $\max\{\, A[i-1], A[j] + points[i] \,\} = A[j] + points[i]$ αποθηκεύουμε στη λίστα το συγκεκριμένο μάθημα και αφαιρούμε όσα το επικαλύπτουν.

4) Τελικά, το ζητούμενο σύνολο μαθημάτων περιέχεται στη λίστα και δίνει $A[n]$ διδακτικές μονάδες.

Θα αποδείξουμε την ορθότητα της συγκεκριμένης προσέγγισης με επαγωγή.

- Επαγωγική Βάση: Προφανώς $A[1] = points[1]$.
- Επαγωγική Υπόθεση: Υποθέτουμε ότι η τιμή $A[i-1]$ είναι βέλτιστη, αντικατοπτρίζει δηλαδή το μέγιστο αριθμό διδακτικών μονάδων για το σύνολο των πρώτων $i-1$ μαθημάτων.
- Επαγωγικό βήμα: Προσθέτουμε μάθημα i για το οποίο υπάρχουν 2 πιθανά σενάρια. Είτε θα περιέχεται στη βέλτιστη επιλογή μαθημάτων είτε όχι.
  - ✓ Αν δεν περιέχεται, για το σύνολο των πρώτων i μαθημάτων ο μέγιστος αριθμός διδακτικών μονάδων ταυτίζεται με αυτόν του συνόλου των πρώτων $i-1$ μαθημάτων: $A[i] = A[i-1]$.
  - ✓ Αν περιέχεται, τότε η $A[i]$ θα προκύπτει από τον αριθμό των διδακτικών μονάδων του μαθήματος i συν την βέλτιστη λύση για τα πρώτα j μαθήματα, όπου j το αμέσως προηγούμενο μάθημα που δεν επικαλύπτεται με το i. Δηλαδή $A[i] = A[j] + points[i]$.

  Βέλτιστη λύση δίνει προφανώς το σενάριο που οδηγεί σε μεγαλύτερο αριθμό διδακτικών μονάδων. Σε περίπτωση που ισχύει το δεύτερο σενάριο, το μάθημα εντάσσεται και στη λίστα.

Ο παραπάνω αλγόριθμος χρησιμοποιεί την τεχνική του Δυναμικού Προγραμματισμού για να βελτιώσει την απόδοσή του. Συγκεκριμένα υλοποιεί τον πίνακα A και επιλύει μία φορά το υποπρόβλημα $A[i]$ όπως ορίστηκε παραπάνω, αποθηκεύοντας την τιμή του. Στη συνέχεια, εφόσον η τιμή $A[i]$ απαιτηθεί για την επίλυση ενός μεγαλύτερου υποπροβλήματος, θα είναι άμεσα διαθέσιμη.

Η πολυπλοκότητα που προκύπτει είναι $nlogn$ για την ταξινόμηση των μαθημάτων, n για την γραμμική τους διάσχιση και για κάθε ένα από τα n μαθήματα απαιτείται (ίσως) επιπλέον $logn$ χρόνος για να βρεθεί με δυαδική αναζήτηση το μάθημα j. Συνολικά $\boldsymbol{O(nlogn)}$.

b) Η greedy ιδέα για την επίλυση του προβλήματος είναι να επισκεπτόμαστε τα μαθήματα στο τέλος τους, ούτως ώστε να μεγιστοποιήσουμε τον αριθμό των μαθημάτων που θα μπορούσαν να τα επικαλύπτουν. Για να οδηγηθούμε ωστόσο στη βέλτιστη εφαρμογή του κανόνα απαιτείται κατάλληλη ταξινόμηση για τα μαθήματα. Ο αλγόριθμος είναι ο εξής:

1) Ταξινομούμε τα n μαθήματα κατά αύξοντα χρόνο ολοκλήρωσης.
2) Ξεκινάμε από το πρώτο στη σειρά μάθημα, το οποίο επισκεπτόμαστε στο τέλος του. Κάνουμε την ανακοίνωση σε αυτό, καθώς φυσικά και σε όσα μαθήματα το επικαλύπτουν εκείνη τη στιγμή.
3) Διαγράφουμε από τη σειρά τα μαθήματα στα οποία έγινε η ανακοίνωση και αποθηκεύουμε το f_i.
4) Επαναλαμβάνουμε τα βήματα 2 και 3 μέχρι να τελειώσουν τα διαθέσιμα μαθήματα.
5) Οι ζητούμενες χρονικές στιγμές είναι τα f που έχουμε αποθηκεύσει (βήμα 2).

Η ορθότητα της συγκεκριμένης διαδικασίας έγκειται στο ότι κάθε επίσκεψή μας μεγιστοποιεί τον αριθμό των μαθημάτων τα οποία διαγράφουμε και κατ' επέκταση στο ότι ο μέγιστος αριθμός επικαλυπτόμενων μαθημάτων για ένα μάθημα συμβαίνει στο τέλος του. Αποδεικνύουμε αυτή τη θέση:

Έστω ότι δοθέν μάθημα i επικαλύπτει μέγιστο αριθμό άλλων μαθημάτων στο σημείο $f_i - x$ (ανοικτό). Θα αποδείξουμε ότι $x = 0$. Θεωρούμε μάθημα $[s = f_i - x,\ f > f_i)$ το οποίο επικαλύπτει το δοθέν μάθημα στο διάστημα $[f_i - x,\ f_i)$ και δεν έχει συνυπολογιστεί. Για να ισχύει η αρχική μας υπόθεση αρκεί να μηδενίσουμε το διάστημα επικάλυψης. Συνεπώς $x = 0$ και το ζητούμενο σημείο είναι το τέλος του δοθέντος μαθήματος.

Τέλος, σημειώνουμε πως επισκεπτόμαστε πάντα το τέλος του πρώτου διαθέσιμου μαθήματος στη σειρά, καθώς αν επιλέξουμε κάποιο επόμενο δε θα μπορέσουμε να ενημερώσουμε το πρώτο, μιας και θα έχει λήξει ($f_1 < f_2$ λόγω ταξινόμησης). Τα παραπάνω στοιχεία μας επιτρέπουν να ολοκληρώσουμε τις ανακοινώσεις σε όλα τα μαθήματα, με τις ελάχιστες δυνατές επισκέψεις. Η πολυπλοκότητα του αλγόριθμου είναι $nlogn$ για τις ταξινομήσεις και n για τη γραμμική διάσχιση των μαθημάτων. Συνολικά έχουμε $\boldsymbol{O(nlogn)}$.

➢ *Θέμα 2*

Η greedy ιδέα για την επίλυση του προβλήματος είναι πως το ακριβότερο κουτί που μπορεί να σπάσει θα βρίσκεται υποχρεωτικά στη βέλτιστη λύση για δεδομένο σετ. Εφαρμόζουμε τη λογική με τον εξής αλγόριθμο:

1) Ταξινομούμε τα κουτιά ως προς την αξία τους u και τα σφυριά ως προς τη δύναμή τους f.
2) Σπάμε το ακριβότερο διαθέσιμο κουτί με το λιγότερο δυνατό σφυρί που μπορεί να το κάνει, αν υπάρχει.
3) Διαγράφουμε το κουτί και επαναλαμβάνουμε το βήμα 2 μέχρι να εξετάσουμε όλα τα κουτιά του σετ.

Θα αποδείξουμε ότι η ιδέα της επιλογής πάντα του ακριβότερου διαθέσιμου κουτιού (που δύναται να σπάσει) μας οδηγεί σε βέλτιστη λύση: Υποθέτουμε μια βέλτιστη λύση. Αν η λύση περιέχει το ακριβότερο κουτί του σετ η πρόταση επιβεβαιώνεται. Αν δεν το περιέχει, θα υπάρχει σύμφωνα με την πρόταση σφυρί που το σπάει. Το σφυρί αυτό χρησιμοποιήθηκε προφανώς στη βέλτιστη λύση. Ανταλλάσσουμε λοιπόν το κουτί που έσπασε με το ακριβότερο. Το κέρδος της βέλτιστης λύσης δεν χειροτερεύει, επομένως καταλήξαμε και πάλι σε βέλτιστη λύση.

Από τα σφυριά που σπάνε το ακριβότερο κουτί, επιλέγουμε προφανώς αυτό με τη μικρότερη δύναμη, ούτως ώστε να προφυλάξουμε τα δυνατά σφυριά για τη συνέχεια. Η πολυπλοκότητα του αλγόριθμου είναι $2 * nlogn$ για τις ταξινομήσεις, n για τη γραμμική διάσχιση των κουτιών και $logn$ για την εύρεση του κατάλληλου σφυριού με εξαντλητική δυαδική αναζήτηση. Συνολικά $\boldsymbol{O(nlogn)}$.

➢ *Θέμα 3*

Θέλουμε να επιλέξουμε ένα αντικείμενο από κάθε χώρα ούτως ώστε να μεγιστοποιήσουμε την συνολική συναισθηματική αξία με δεδομένο budget. Ορίζουμε την συνάρτηση $f[i, c]$ όπου i η χώρα, c το budget και f η μέγιστη δυνατή συναισθηματική αξία, προκειμένου να εκφράσουμε το γενικό αυτό πρόβλημα συναρτήσει ανάλογων υποπροβλημάτων. Έστω λοιπόν μια βέλτιστη συλλογή αντικειμένων $f[i, c]$. Αν αφαιρέσουμε το αντικείμενο που πήραμε από την i χώρα, τότε η $f[i-1, c-c_{ij}]$ θα είναι επίσης βέλτιστη. Η αρχή της βελτιστότητας έγκειται στο ότι το υποπρόβλημα $f[i-1, c-c_{ij}]$ είναι ανεξάρτητο (όπως έχει οριστεί) των επιλογών μας στις επόμενες χώρες, άρα μπορεί να υπολογιστεί ξεχωριστά και να χρησιμοποιηθεί στη συνέχεια. Από κει και πέρα, εφόσον για τη χώρα i εξετάσουμε κάθε αναμνηστικό σε συνδυασμό με το υποπρόβλημα που αυτό συνεπάγεται για τις προηγούμενες χώρες, η βέλτιστη επιλογή θα είναι προφανώς η επιλογή της εναλλακτικής που μεγιστοποιεί την συναισθηματική αξία.

Μπορούμε λοιπόν να γράψουμε την αναδρομική σχέση ως εξής: $f[i, c] = \max_{\mathrm{j}} \left\{ f\left[i-1, c-c_{ij}\right] + p_{ij} \right\}$ όπου $j = \{1,2, \dots, k_i\}$ το εκάστοτε αναμνηστικό της χώρας i. Θα εφαρμόσουμε την τεχνική του Δυναμικού Προγραμματισμού προκειμένου να λύσουμε bottom-up το πρόβλημα, αποθηκεύοντας κάθε φορά τις τιμές που θα χρειαστούμε σε μεγαλύτερα υποπροβλήματα. Έτσι, ξεκινάμε από την πρώτη χώρα και βρίσκουμε την τιμή της f για κάθε δυνατό κόστος από 0 μέχρι C (σε χρόνο $k_1 C$). Κάνουμε το ίδιο για κάθε επόμενη χώρα, αξιοποιώντας την αναδρομική σχέση και το ότι η προηγούμενη τιμή της f που ζητείται είναι άμεσα διαθέσιμη. Η ζητούμενη ακολουθία αγορών δίνεται μέσω της αναδρομής από την τιμή $f[n, C]$ σε χρόνο n στη χειρότερη περίπτωση. Για να συμπληρωθεί ο πίνακας θα απαιτηθεί χρόνος $k_i C$ ανά χώρα i και συνολικά $\boldsymbol{O(sum(k) * C)}$.

➢ *Θέμα 4*

Θέλουμε να βρούμε την συντομότερη διαδρομή ανάμεσα στα κουτιά, ούτως ώστε να φάμε $Q +$ σοκολατάκια. Το πρόβλημα έχει συνθήκες που περιορίζουν το ποια κουτιά μπορούμε να επισκεφθούμε σε κάθε μετακίνηση: Κάθε κουτί από το οποίο τρώμε πρέπει να έχει περισσότερα σοκολατάκια από το προηγούμενο καθώς και άλλου είδους (θεωρούμε πως κάθε κουτί έχει διαφορετικό αριθμό από σοκολατάκια). Επίσης, τρώμε όλα τα σοκολατάκια από ένα κουτί, επομένως μπορούμε να το επισκεφθούμε μόνο μία φορά. Αντιστοιχίζουμε λοιπόν σε κάθε κουτί i ένα σύνολο S_i το οποίο θα περιέχει όλα τα κουτιά τα οποία είναι επισκέψιμα από αυτό.

Στη συνέχεια εξετάζουμε αναδρομικά τα βήματα που απαιτούνται για κάθε επισκέψιμο κουτί ούτως ώστε να φαγωθούν όσα σοκολατάκια απομένουν μέχρι τον στόχο. Για να πάμε δηλαδή κατά βέλτιστο τρόπο από ένα κουτί στο στόχο αρκεί να ελέγξουμε τη βέλτιστη λύση για κάθε επισκέψιμο κουτί. Η αρχή βελτιστότητας του προβλήματος έγκειται στο ότι, εφόσον τρώμε πάντα περισσότερο, η διαδρομή που θα ακολουθήσουμε όντας σε κουτί i είναι ανεξάρτητη από το πώς φτάσαμε σε αυτό, μιας και θα περιλαμβάνει κουτιά με περισσότερα σοκολατάκια από όλα τα πιθανά κουτιά που μας οδήγησαν στο i. Συνεπώς μπορούμε να υπολογίσουμε ανεξάρτητα το υποπρόβλημα κάθε επισκέψιμου κουτιού. Επιλέγοντας μετά την συντομότερη διαδρομή από όλες τις εναλλακτικές, διασφαλίζουμε ότι η λύση μας για το κουτί i είναι βέλτιστη.

Για να εκφράσουμε αυτή τη σχέση ορίζουμε συνάρτηση $steps[i, q]$ όπου i το προς εξέταση κουτί, q τα σοκολατάκια που πρέπει να φαγωθούν και $steps$ ο ελάχιστος χρόνος για να το πετύχουμε. Μπορούμε τώρα να εκφράσουμε την αναδρομή:

$$steps[i,q] = \begin{cases} 0 & if\ q \le 0 \\ \infty & if\ S_i = \emptyset \\ \min\limits_{j \in S_i} \{steps[j, q - q_i] + |i - j|\} \end{cases}$$

Θα εφαρμόσουμε την τεχνική του Δυναμικού Προγραμματισμού προκειμένου να λύσουμε bottom-up το πρόβλημα, αποθηκεύοντας κάθε φορά τις τιμές που θα χρειαστούμε σε μεγαλύτερα υποπροβλήματα. Έτσι, ξεκινάμε από την απλούστερη κατηγορία υποπροβλημάτων (1 σοκολατάκι) και υπολογίζουμε τα βήματα που απαιτούνται για κάθε κουτί, προσθέτοντας σε κάθε επανάληψη ένα σοκολατάκι μέχρι Q σε αριθμό και αξιοποιούμε για τον υπολογισμό των βημάτων την παραπάνω αναδρομική συνάρτηση. Η διαδρομή μας ξεκινάει από το κουτί p, το οποίο ωστόσο δεν τρώμε απαραίτητα. Θα πρέπει να ελέγξουμε όλα τα κουτιά, με την απόστασή τους από το p, προκειμένου να βρούμε τη βέλτιστη εκκίνηση. Αυτό γίνεται μέσω της σχέσης $\min\{steps(i, Q) + |p - i|\}$. Η λύση είναι η διαδρομή που προκύπτει από την αναδρομή για αυτή την εκκίνηση.

Θεωρώντας ότι το σύνολο S είναι σταθερό για κάθε κουτί και ελέγχονται όλα τα κουτιά για τον σχηματισμό του, η διαδικασία αυτή ολοκληρώνεται σε χρόνο n^2 για τον υπολογισμό n συνόλων S σε γραμμικό χρόνο το καθένα και Qn^2 για τον υπολογισμό όλων των τιμών της $steps$. Αυτό διότι, με τη βοήθεια της αναδρομής, κάθε τιμή της θα απαιτεί γραμμικό χρόνο. Η συνολική πολυπλοκότητα του αλγορίθμου είναι $O(n^2 + Qn^2 + n) = \boldsymbol{O(Qn^2)}$. Ο αλγόριθμος σε μορφή ψευδοκώδικα:

```
min_steps (box, q, S)
{
        min = ∞
        for j in S {
            steps[box, q] = |i − j| + min_steps(j, q − q_box, S_j)
            if steps[box, q] < min then min = steps[box, q]
        }
        return min
}

for i from 1 until n
    for j from 1 until n
        if (i.q < j.q & i.t ≠ j.t) then add(j, S_i)

for j from 1 until Q
    for i from 1 until n
        min_steps(i, j, S_i)

for i from 1 until n
    return min{steps(i, Q) + |p − i|}
```

- ***Θέμα 5***

Η βασική άπληστη ιδέα για τον καθορισμό του είδους κάθε κεραίας, ώστε να καταλήξουμε σε ελάχιστη κατανάλωση ισχύος, είναι το να επιλέγουμε συνεχώς τη φθηνότερη διαθέσιμη εναλλακτική, δηλαδή δέκτη. Συγκεκριμένα:

1) Ξεκινάμε από τα αριστερά, θεωρώντας την πρώτη κεραία υποχρεωτικά πομπό.
2) Θεωρούμε την επόμενη κεραία δέκτη, καθώς είναι γενικώς φθηνότερη επιλογή από τον πομπό.
3) Προσθέτουμε την κεραία σε min heap με κριτήριο τη διαφορά $T - R$.
4) Αν ο συνολικός αριθμός των δεκτών γίνει μεγαλύτερος των πομπών (συμβαίνει στα 3,5,7,… στοιχεία) τότε μετατρέπουμε σε πομπό τον δέκτη που βρίσκεται στην κορυφή του heap, καθώς έχοντας τη μικρότερη διαφορά $T - R$ θα δώσει το ελάχιστο επιπλέον κόστος, διατηρώντας βέλτιστη λύση.
5) Επαναλαμβάνουμε από το βήμα 2 μέχρι να καθοριστεί το είδος κάθε κεραίας.
6) Στο τέλος καταλήγουμε σε διάταξη με την ελάχιστη δυνατή κατανάλωση ισχύος.

Η ορθότητα της παραπάνω διαδικασίας έγκειται στο ότι ξεκινάμε από μια προφανώς βέλτιστη λύση και συνεχώς κάνουμε τις βέλτιστες επεκτάσεις. Επιλέγουμε στην αρχή πάντα δέκτη, που είναι η φθηνότερη επιλογή και αν απαιτηθεί πομπός για να είναι έγκυρη η λύση, κάνουμε την μετατροπή που θα μας κοστίσει επιπλέον το ελάχιστο. Η μετατροπή αυτή αναζητείται σε όλο το εύρος της μέχρι στιγμής διάταξης, επομένως διασφαλίζουμε πως μέχρι και η τελική διάταξη των n κεραιών θα είναι βέλτιστη. Η πολυπλοκότητα του αλγορίθμου είναι n για τη γραμμική διάσχιση των κεραιών και για κάθε κεραία $logn$ για την τοποθέτησή της στο heap καθώς και σταθερός χρόνος για την εύρεση του δέκτη που θα μετατραπεί, αν χρειάζεται. Συνολικά έχουμε $\boldsymbol{O(nlogn)}$.

- ✓ *<u>Παρατηρήσεις επί των απαντήσεων:</u>*