DP Optimisation using Convex Hull Trick Technique

Δημήτρης Χρήστου
29/11/2018

Πρόβλημα 1: Covered Walkway Problem

Εκφώνηση: https://www.e-olymp.com/en/problems/3972

Λύση:

Θεωρούμε ως είσοδο τα σημεία x_i για i=0,1,...,N-1 και την σταθερά c. Αρχικά θα επιλύσουμε το πρόβλημα με Δυναμικό Προγραμματισμό. Είναι εύκολο να κάνουμε τις εξής παρατηρήσεις:

1. Κάθε στέγη θα ξεκινάει από κάποιο σημείο x_i της εισόδου και θα τελειώνει σε κάποιο σημείο x_j (πιθανώς και το ίδιο). Σε κάθε άλλη περίπτωση, θα καλύπτουμε χώρο που δεν είναι απαραίτητο να καλυφθεί και έτσι η λύση δεν θα είναι βέλτιστη.

2. Η πρώτη στέγη σε μία βέλτιστη λύση θα ξεκινάει από το σημείο x_0 ενώ η τελευταία στέγη θα τελειώνει στο σημείο x_{N-1}.

Επομένως, μπορούμε να αρχίσουμε από το σημείο x_{N-1} και να εξετάσουμε ποιο από τα σημεία x_i όταν χρησιμοποιηθεί ως αρχή της τελευταίας στέγης θα δώσει το ελάχιστο κόστος. Είναι:

$$cost(i) = \begin{cases} 0, & i = -1 \\ \min\limits_{0 \le j \le i} \{cost(j-1) + (x_i - x_j)^2 + c\}, & \text{αλλιώς} \end{cases}$$

Η παραπάνω αναδρομική σχέση εκφράζει το ελάχιστο κόστος που χρειάζεται για να σκεπάσουμε τα σημεία x_0 έως x_i, και επιλέγει το κατάλληλο σημείο x_j στο οποίο ξεκινάει η τελευταία στέγη μέσω του ελαχίστου. Η λύση του προβλήματός μας είναι η τιμή **cost(N)**. Η πολυπλοκότητα της παραπάνω λύσης, αν υλοποιηθεί όπως ένα κλασσικό πρόγραμμα δυναμικού προγραμματισμού (είτε bottom-up είτε top-down προσέγγιση), μπορεί να υπολογιστεί ως:

$$complexity = \#subproblems \cdot subproblem\_complexity$$

Συμπληρώνουμε έναν πίνακα N θέσεων και για κάθε στοιχείο πρέπει να υπολογίσουμε μία ελάχιστη τιμή που απαιτεί γραμμικό χρόνο εν γένει. Επομένως η πολυπλοκότητα της λύσης είναι $O(N^2)$.

Μπορούμε να τα καταφέρουμε καλύτερα; Ναι, αν χρησιμοποιήσουμε μία τεχνική βελτιστοποίησης DP. Συγκεκριμένα, θα προσπαθήσουμε να μειώσουμε το χρόνο που απαιτείται για τον υπολογισμό του ελαχίστου από γραμμικό σε σταθερό, πετυχαίνοντας μία γραμμική πολυπλοκότητα O(N). Μετασχηματίζουμε την αναδρομική σχέση που γράψαμε με τον ακόλουθο τρόπο:

$$cost(i) = \min_{0 \le j \le i} \{cost(j-1) + (x_i - x_j)^2 + c\} = \min_{0 \le j \le i} \{cost(j-1) + x_i^2 - 2x_i x_j + x_j^2 + c\} \Rightarrow$$

$$\boxed{cost(i) = x_i^2 + c + \min_{0 \le j \le i} \{cost(j-1) + x_j^2 - 2x_i x_j\}}$$

Από την παραπάνω μορφή προκύπτει ότι το $x_i^2 + c$ δεν παίζει κανένα ρόλο στον υπολογισμό του ελαχίστου, αφού απλά προστίθεται ανεξάρτητα από την επιλογή του j για την αποτίμηση του υποπροβλήματος. Για την συνέχεια του προβλήματος, υϊοθετούμε τον συμβολισμό:

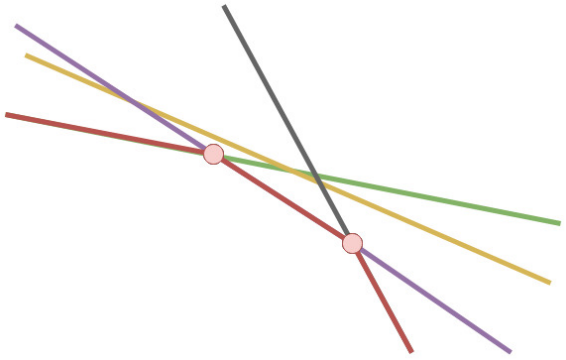
$$b[j] = cost(j-1) + x_j^2$$
$$a[j] = -2x_j$$

Επομένως, η αναδρομική μας σχέση ανάγεται στην μορφή:

$$cost(i) = x_i^2 + c + \min_{0 \le j \le i} \{a[j]x_i + b[j]\}$$

Παρατηρούμε ότι οι ποσότητες μέσα στο ελάχιστο θυμίζουν εξίσωση ευθείας με κλίση a και σημείο τομής με y'y το b, αποτιμημένες για $x = x_i$. Έτσι, προκύπτει η ακόλουθη διατύπωση του προβλήματος: **Δεδομένου ενός συνόλου ευθειών με παραμέτρους (a_j, b_j), βρες ποια από αυτές δίνει τη μικρότερη τιμή για $x = x_i$**. Τέτοιου είδους προβλήματα επιλύονται σε γραμμικό χρόνο με μία τεχνική που καλείται **Convex Hull Trick**. Απαραίτητη προϋπόθεση για να εφαρμοστεί η τεχνική είναι οι κλίσεις a_j των ευθειών να αποτελούν φθίνουσα ακολουθία ($a_j \ge a_{j+1}$) και οι τιμές στις οποίες υπολογίζουμε το ελάχιστο να αποτελούν αύξουσα ακολουθία ($x_i \le x_{i+1}$).

Στο δικό μας πρόβλημα, επειδή τα x_i δίνονται ταξινομημένα σε αύξουσα σειρά, πληρούνται και οι δύο προϋποθέσεις και μπορούμε να συνεχίσουμε. Ουσιαστικά, θέλουμε έναν τρόπο να γνωρίζουμε ποια ευθεία ελαχιστοποιεί ένα τυχαίο x∈R δεδομένου ενός συνόλου ευθειών με παραμέτρους (a_j, b_j), για j=0,1,..i ώστε να απαντήσουμε στο ελάχιστο που εμφανίζει το cost(i). Ισοδύναμα, θέλουμε να γνωρίζουμε για κάθε x, ποια ευθεία έχει την μικρότερη τιμή. Συνεπώς, η πληροφορία που χρειάζεται να κρατάμε είναι το **κυρτό περίβλημα** (convex hull) του συνόλου των ευθειών.

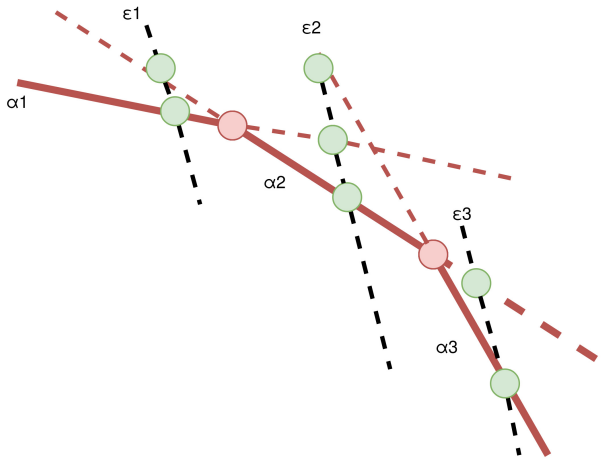


Σχήμα 1: (Κάτω) Κυρτό Περίβλημα Ευθειών

Στην πράξη λοιπόν, πρέπει να κρατάμε μία δομή όπως αυτή του σχήματος 1, που να δηλώνει με ποια σειρά εμφανίζονται οι ευθείες στο κυρτό περίβλημα. Για παράδειγμα, στο σχήμα 1 πρέπει να κρατάμε την πληροφορία {πράσινο, μωβ, γκρι}. Επειδή κάθε φορά που υπολογίζουμε μία τιμή πρέπει να προσθέτουμε και την αντίστοιχη ευθεία (a_i, b_i) στο κυρτό περίβλημα, η δομή αυτή θα είναι δυναμική. Επίσης εκτός από την λειτουργία της πρόσθεσης ευθείας στο περίβλημα, θέλουμε να εξετάζουμε και ποια ευθεία ελαχιστοποιεί κάποιο συγκεκριμένο x (querying) για να αποτιμούμε γρήγορα τα ελάχιστα. Όπως θα δούμε, υπό τις προϋποθέσεις που αναφέραμε, και οι δύο λειτουργίες γίνονται σε O(1) (τυπικά amortized O(1)).

Ας ξεκινήσουμε την ανάλυση της λειτουργίας προσθήκης μίας ευθείας στο κυρτό περίβλημα. Εφόσον $0 > a[j] \ge a[j+1] \forall j$, ξέρουμε με βεβαιότητα ότι η νέα ευθεία με κλίση a[j] θα έχει πιο αρνητική κλίση από όλες όσες συνιστούν το κυρτό περίβλημα (ευθείες a[0],a[1],...,a[j-1]), και κατά συνέπεια μετά από κάποιο x αυτή θα δίνει τις μικρότερες τιμές. Πρέπει όμως να βρούμε σε ποιο σημείο του κυρτού περιβλήματος πρέπει να ενταχθεί (πιθανόν να διώξει τελείως κάποιες ευθείες). Ο τρόπος με τον οποίο αυτό γίνεται είναι απλός: ξεκινώντας από την τελευταία ευθεία του κυρτού περιβλήματος, ελέγχουμε αν το σημείο τομής της νέας ευθείας με αυτήν είναι πιο κάτω (στον άξονα y) από το σημείο τομής της προτελευταίας ευθείας με την νέα. Αν αυτό ισχύει, τότε η νέα ευθεία

εισάγεται στο τέλος του περιβλήματος. Διαφορετικά, πετάμε την τελευταία ευθεία από το κυρτό περίβλημα και επαναλαμβάνουμε.



Σχήμα 2: Εισαγωγή Ευθείας σε Κυρτό Περίβλημα

Ας θεωρήσουμε το διπλανό παράδειγμα. Αν θέλουμε να εισάγουμε την ε3, τότε βλέπουμε ότι το σημείο τομής της με την α3 είναι πιο κάτω από το σημείο τομής της με την α2. Άρα απλά μπαίνει στο τέλος του περιβλήματος. Αν θέλουμε όμως να εισάγουμε την ε2, το σημείο τομής της με την α3 είναι πιο πάνω από το σημείο τομής της με την α2. Άρα η α3 διαγράφεται. Έπειτα, το σημείο τομής της ε2 με την α2 είναι πιο κάτω από το σημείο τομής της ε2 με την α1, άρα τοποθετούμε την ε2 μετά την α2. Αντίστοιχα, η ε1 διαγράφει τις α2 και α3 και τοποθετείτε μετά την α1. Τονίζεται ότι αν δεν γνωρίζαμε ότι οι κλίσεις αποτελούν φθίνουσα ακολουθία, ο παραπάνω αλγόριθμος εισαγωγής δεν δουλεύει.

Επομένως, ποιά είναι η πολυπλοκότητα εισαγωγής μίας ευθείας στο κυρτό περίβλημα; Θα μπορούσαμε να πούμε ότι είναι O(n), γιατί μία ευθεία μπορεί να διασχίσει όλο το κυρτό περίβλημα μέχρι να βρει την θέση της. Στην πραγματικότητα όμως αυτό δεν αληθεύει. Κάθε ευθεία, μπορεί να ελεγχθεί "αρνητικά" μία ακριβώς φορά, αφού στη συνέχεια την διαγράφουμε. Στο κυρτό περίβλημα εισάγονται συνολικά n-ευθείες, άρα **όλες οι εισαγωγές συνολικά κοστίζουν O(n)**. Επομένως, κάθε μεμονωμένη εισαγωγή κοστίζει O(1) (η τεχνική αυτή καλείται amortization, `https://en.wikipedia.org/wiki/Amortized_analysis`). Διαισθητικά, αυτό σημαίνει ότι αν συμβεί μία πολύ κακή εισαγωγή όπου χρειάζεται να ψάξω όλο το κυρτό περίβλημα, τότε στο τέλος θα διαγράψω πάρα πολλές ευθείες, το κυρτό περίβλημα θα μικρύνει και έτσι όλες οι υπόλοιπες ευθείες θα εισαχθούν πολύ εύκολα. Επίσης είναι απαραίτητο να ξεκινάω από το τέλος και όχι από την αρχή του κυρτού περιβλήματος για να πετύχω O(1). Διαφορετικά, μπορεί κάθε ευθεία να μπαίνει στο τέλος και να απαιτεί πλήρη διάσχυση του κυρτού περιβλήματος χωρίς να διαγράφει τίποτα.

Με παρόμοια λογική αναλύουμε και την λειτουργία των ερωτήσεων (queries). Αυτήν την φορά ξεκινάμε από την αρχή και ψάχνουμε σε ποιά περιοχή του convex hull ανήκει το στοιχείο x. Επειδή το επόμενο στοιχείο γνωρίζουμε ότι θα είναι μεγαλύτερο, όσες περιοχές απορριφθούν από το πρώτο σίγουρα θα απορριφθούν και από το δεύτερο, και από το τρίτο, κοκ, αφού κινούμαστε προς τα δεξιά. Επομένως, καθώς ψάχνουμε από την αρχή του κυρτού, μπορούμε να πετάμε ότι δεν μας κάνει, πετυχαίνοντας και πάλι κόστος ερώτησης amortized O(1).

Με αυτόν τον τρόπο, μπορούμε να υπολογίζουμε τα ελάχιστα σε χρόνο O(1) και να ανανεώνουμε το περίβλημα επίσης σε χρόνο O(1), καθιστώντας την λύση του προβλήματος γραμμική. Τα βήματα εκτέλεσης του αλγορίθμου είναι:

1. Αρχικοποίηση cost[-1] = 0 και CH={} (κυρτό περίβλημα)
2. Για κάθε i από 0 έως (N-1), εισάγουμε την ευθεία (a_i, b_i) στο κυρτό περίβλημα (απαιτείται έως και η τιμή cost[i-1]). Έπειτα, κάνουμε query πάνω στο CH για την τιμή x_i. Οι δύο αυτές λειτουργίες γίνονται όπως περιγράψαμε. Αφού προσδιοριστεί το j που ελαχιστοποιεί την ποσότητα, προσθέτουμε στο αποτέλεσμα $x_i^2 + c$ και υπολογίζουμε το cost[i].
3. Αφού ολοκληρώσουμε για όλα τα i, εμφανίζουμε το cost(N-1).

Πρόβλημα 2: Leaves

Εκφώνηση: `https://www.spoj.com/problems/NKLEAVES/`

Λύση:

Θεωρούμε ως είσοδο στο πρόβλημα τα βάρη των φύλλων w_i για i=1,...,n και το πλήθος k των σωρών που πρέπει να τοποθετήσουμε. Όπως και πριν, ξεκινάμε διατυπώνοντας μία αναδρομική σχέση δυναμικού προγραμματισμού, που εκφράζει **το ελάχιστο κόστος που απαιτείται με την βέλτιστη τοποθέτηση k-σωρών, εξετάζοντας τα φύλλα 1 έως n**. Είναι:

$$cost(i,k) = \begin{cases} 0, & i = k \\ \min\limits_{k \le j \le i} \{cost(j-1,k-1) + \sum\limits_{j+1 \le \lambda \le i} w_\lambda(\lambda - j)\}, & \text{i>k} \end{cases}$$

Αν πρέπει να τοποθετήσω k σωρούς σε k θέσεις προφανώς η βέλτιστη (και μοναδική) λύση είναι να τοποθετήσω μία σωρό σε κάθε θέση που δίνει μηδενικό κόστος μεταφοράς. Αν έχω περισσότερες από k-θέσεις, τότε εξετάζω όλες τις θέσεις που είναι μεγαλύτερες ή ίσες του k (ώστε μετά να έχω τουλάχιστον τόσα φύλλα όσες και σωρούς) και υπολογίζω το κόστος για k-1 σωρούς έως και την θέση j-1 συν το κόστος μεταφοράς από την θέση i στη θεση j όπου και τοποθετώ τη σωρό. Προφανώς και επιλέγω το άθροισμα που ελαχιστοποιεί το κόστος. **Η λύση του προβλήματος είναι η τιμή cost(n,k)**.

Έχουμε n·k υποπροβλήματα (θέσεις στον πίνακα cost) και για κάθε υποπρόβλημα απαιτείται μία ελαχιστοποίση πάνω σε τμήμα της προηγούμενης στήλης (για k σωρούς αναζητώ λύσεις στην στήλη k-1). Επιπλέον, σε κάθε προς εξέταση τιμή για το ελάχιστο, πρέπει να υπολογίσουμε και ένα άθροισμα n-το πολύ στοιχείων. Επομένως για κάθε στοιχείο απαιτείται τετραγωνικός χρόνος και μία απλή υλοποίηση της αναδρομικής θα οδηγήσει σε λύση πολυπλοκότητας $O(kn^3)$.

Γενικά, το πρώτο πρόβλημα που πρέπει να αντιμετωπίσουμε είναι ο χρόνος που απαιτείται για τα αθροίσματα μέσα στα ελάχιστα. Μία καλή ιδέα είναι να έχουμε υπολογίσει εκ των προτέρων τρέχοντα αθροίσματα πάνω στην είσοδο (running sums) σε γραμμικό χρόνο, και να εκφράσουμε την αναδρομική σχέση συναρτήσει αυτών των αθροισμάτων. Συγκεκριμένα, έστω:

$$S(i) = \sum_{1 \le k \le i} w_k$$

$$SW(i) = \sum_{1 \le k \le i} w_k \cdot k$$

Επειδή ισχύουν οι αναδρομικές σχέσεις:

$$S(i) = w_i + S(i-1)$$

$$SW(i) = i \cdot w_i + SW(i-1)$$

οι παραπάνω ποσότητες υπολογίζονται σε γραμμικό χρόνο. Έτσι, μπορούμε να γράψουμε την αναδρομική μας σχέση ως:

$$cost(i,k) = \begin{cases} 0, & i = k \\ \min\limits_{k \le j \le i} \{cost(j-1,k-1) + \sum\limits_{j+1 \le \lambda \le i} w_\lambda \lambda - j \cdot \sum\limits_{j+1 \le \lambda \le i} w_\lambda\}, & \text{i>k} \end{cases}$$

ή ισοδύναμα:

$$cost(i,k) = \begin{cases} 0, & i = k \\ \min\limits_{k \le j \le i} \{cost(j-1,k-1) + SW(i) - SW(j) - jS(i) + jS(j)\}, & \text{i>k} \end{cases}$$

Η υλοποίηση της αναδρομικής σχέσης με χρήση running sums εξασφαλίζει σταθερό χρόνο για τον υπολογισμό των αθροισμάτων, άρα η συνολική πολυπλοκότητα του προβλήματος βελτιώνεται σε $O(n^2)$.

Η επόμενη βελτίωση που μπορούμε να κάνουμε αφορά τον υπολογισμό των ελαχίστων. Όπως και στο προηγούμενο πρόβλημα, ξεκινάμε από την αναδρομική σχέση και προσπαθούμε να την μετασχηματίσουμε σε μία μορφή που να εφαρμόζεται το Convex Hull Trick. Είναι:

$$cost(i,k) = \min_{k \le j \le i} \{cost(j-1,k-1) + SW(i) - SW(j) - jS(i) + jS(j)\} \Rightarrow$$
$$cost(i,k) = SW(i) + \min_{k \le j \le i} \{cost(j-1,k-1) - SW(j) + jS(j) - jS(i)\}$$

Για την συνέχεια του προβλήματος, ακολουθούμε τον συμβολισμό:

$$b[j,k] = cost(j-1,k-1) + jS(j) - SW(j)$$
$$a[j] = -j$$

Επομένως, η αναδρομική μας σχέση ανάγεται στην μορφή:

$$cost(i,k) = SW(i) + \min_{k \le j \le i} \{a[j]S(i) + b[j,k]\}$$

Παρατηρούμε ότι η ακολουθία a[j] είναι φθίνουσα καθώς αυξάνεται το j και ότι τα S(i) συνιστούν αύξουσα ακολουθία. Επίσης, τα b[j,k] υπολογίζονται σε σταθερό χρόνο δεδομένου ότι έχουμε υπολογίσει όλα τα στοιχεία της στήλης (k-1). Για την πρώτη στήλη, αφού αν έχουμε μόνο μία σωρο πρέπει υποχρεωτικά να την τοποθετήσουμε στη θέση 1, η λύση του προβλήματος είναι τετριμμένη.

Συνεπώς, δεδομένης της προηγούμενης στήλης, κάθε ελάχιστο που χρειάζεται να υπολογίζω σε μία γραμμή k μπορεί να βρεθεί σε amortized O(1) χρησιμοποιώντας το Convex Hull Trick ακριβώς με τον ίδιο τρόπο που το χρησιμοποιήσαμε στο προηγούμενο πρόβλημα. Επισημαίνεται ότι κάθε φορά που αλλάζουμε στήλη (δηλαδή τελειώνουμε με τους υπολογισμούς για κάποιο k) και προχωράμε στην επόμενη, αρχικοποιούμε ένα νέο κυρτό περίβλημα και αρχίζουμε πάλι από την αρχή, όπως περιγράψαμε στο πρόβλημα 1. Επομένως, η συνολική πολυπλοκότητα της λύσης είναι O(kn).

Γενίκευση Μεθόδου

Συμπερασματικά, ο τρόπος με τον οποίο αντιμετωπίζουμε τέτοια προβλήματα είναι η αναγωγή της αναδρομικής σχέσης σε ένα πρόβλημα ελαχιστοποίησης ή μεγιστοποίησης κάποιας τιμής πάνω σε ένα σύνολο ευθειών. Συγκεκριμένα, επιδιώκουμε τον μετασχηματισμό σε αναδρομικές σχέσεις της μορφής:

$$dp[i] = f[i] + \min_{0 \le j < i} \{a[j] \cdot x[i] + b[j, dp(j-1)]\}$$
$$a[j] \ge a[j+1], \quad x[i] \le x[i+1]$$

ή αντίστοιχα (αν και δεν το συναντήσαμε σε κάποιο πρόβλημα, η λογική είναι παρόμοια):

$$dp[i] = f[i] + \max_{0 \le j < i} \{a[j] \cdot x[i] + b[j, dp(j-1)]\}$$
$$a[j] \le a[j+1], \quad x[i] \le x[i+1]$$

Στη δεύτερη περίπτωση, για να έχουμε εισαγωγή ευθείας σε amortized O(1) χρειάζεται οι κλίσεις των ευθειών να δίνονται σε αύξουσα σειρά ενώ στα προβλήματα ελαχιστοποίησης είναι απαραίτητο οι κλίσεις των ευθειών να δίνονται σε φθίνουσα σειρά.

Η μέθοδος αυτή επιτρέπει την βελτίωση της πολυπλοκότητας από τετραγωνική σε γραμμική, αφού υπολογίζουμε τα μέγιστα/ελάχιστα σε σταθερό χρόνο. Επίσης ρίχνει πολυπλοκότητες $O(kn^2)$ σε $O(kn)$.

Τέλος, αξίζει να σημειωθεί ότι αν τα "κλειδιά" x[i] πάνω στα οποία ελαχιστοποιώ/μεγιστοποιώ τις ευθείες δεν δίνονται σε αύξουσα σειρά, τότε το querying δεν μπορεί να γίνει σε amortized O(1) (εφόσον η θέση του x[i] δεν δίνει πληροφορίες για την θέση του x[i+1] και επομένως δεν μπορούμε να διαγράψουμε ευθείες από το convex hull), αλλά και πάλι μπορούμε να εκμεταλλευτούμε ότι το convex hull διατηρείται ταξινομημένο και να αναζητήσουμε την κατάλληλη ευθεία με binary search. Με αυτόν τον τρόπο ρίχνουμε πολυπλοκότητες O(n^2) σε O(nlogn).

Φυσικά, δεν μπορούν όλα τα προβλήματα να επιλυθούν με αυτόν τον τρόπο, ενώ παράλληλα υπάρχουν διαφορετικές (και πιο προχωρημένες) τεχνικές βελτιστοποίησης προβλημάτων DP, όπως **Divide and Conquer Optimisation**, **Knuth Optimisation** και **Lagrange Optimisation**.