Αλγόριθμοι και Πολυπλοκότητα
2η Σειρά Γραπτών και Προγραμματιστικών Ασκήσεων

CoReLab

ΣΗΜΜΥ - Ε.Μ.Π.

Δεκέμβριος 2018

Outline

Άσκηση 1 (α.1): Δρομολόγηση Μαθημάτων

Το Πρόβλημα Δρομολόγησης Μαθημάτων

Είσοδος: Χρονικά διαστήματα διδασκαλίας μαθημάτων $[s_i, f_i)$, $i = 1, \ldots, n$.

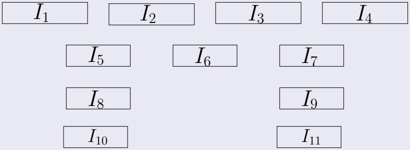
Ζητείται: Μεγίστου πλήθους σύνολο μαθημάτων χωρίς επικαλύψεις.

Άσκηση 1 (α.1): Άπληστα Κριτήρια Δρομολόγησης Μαθημάτων

1. Λιγότερες Επικαλύψεις

Λάθος!

Αντιπαράδειγμα:



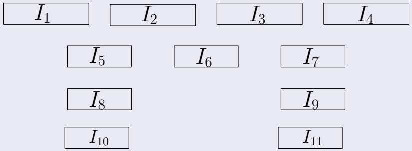
- Λύση Κριτηρίου, S_1: Επιλέγει τα I_6, I_1, I_4. $\Longrightarrow |S_1| = 3$.
- Βέλτιστη Λύση, OPT: $\{I_1, I_2, I_3, I_4\} \Longrightarrow |OPT| = 4$.

Άσκηση 1 (α.1): Άπληστα Κριτήρια Δρομολόγησης Μαθημάτων

1. Λιγότερες Επικαλύψεις

Λάθος!

Αντιπαράδειγμα:



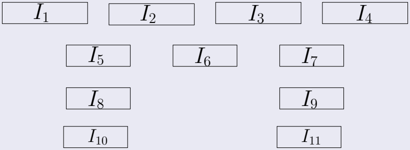
- Λύση Κριτηρίου, S_1: Επιλέγει τα I_6, I_1, I_4. $\Longrightarrow |S_1| = 3$.
- Βέλτιστη Λύση, OPT: $\{I_1, I_2, I_3, I_4\} \Longrightarrow |OPT| = 4$.

Άσκηση 1 (α.1): Άπληστα Κριτήρια Δρομολόγησης Μαθημάτων

1. Λιγότερες Επικαλύψεις

Λάθος!

Αντιπαράδειγμα:



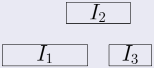
- Λύση Κριτηρίου, S_1: Επιλέγει τα I_6, I_1, I_4. $\Longrightarrow |S_1| = 3$.
- Βέλτιστη Λύση, OPT: $\{I_1, I_2, I_3, I_4\} \Longrightarrow |OPT| = 4$.

Άσκηση 1 (α.1): Άπληστα Κριτήρια Δρομολόγησης Μαθημάτων

2. Μεγαλύτερη Διάρκεια

Λάθος!

Αντιπαράδειγμα:

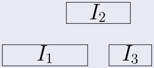


- Λύση Κριτηρίου, S_2: Διαγράφει πρώτα το I_1 και μετά το I_2 $\implies |S_2| = |\{I_3\}| = 1$.
- Βέλτιστη Λύση, OPT: $\{I_1, I_3\} \implies |OPT| = 2$.

2. Μεγαλύτερη Διάρκεια

Λάθος!

Αντιπαράδειγμα:



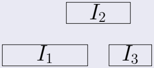
- Λύση Κριτηρίου, S_2: Διαγράφει πρώτα το I_1 και μετά το I_2 $\Longrightarrow |S_2| = |\{I_3\}| = 1$.
- Βέλτιστη Λύση, OPT: $\{I_1, I_3\} \Longrightarrow |OPT| = 2$.

Άσκηση 1 (α.1): Άπληστα Κριτήρια Δρομολόγησης Μαθημάτων

2. Μεγαλύτερη Διάρκεια

Λάθος!

Αντιπαράδειγμα:



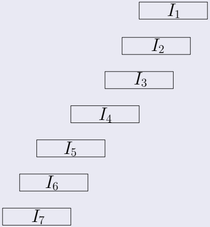
- Λύση Κριτηρίου, S_2: Διαγράφει πρώτα το I_1 και μετά το I_2 $\Longrightarrow |S_2| = |\{I_3\}| = 1$.
- Βέλτιστη Λύση, OPT: $\{I_1, I_3\} \Longrightarrow |OPT| = 2$.

Άσκηση 1 (α.1): Άπληστα Κριτήρια Δρομολόγησης Μαθημάτων

3. Περισσότερες Επικαλύψεις

Λάθος!

Αντιπαράδειγμα:



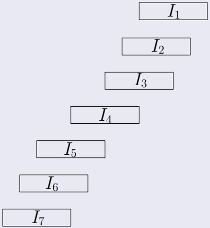
- Λύση Κριτηρίου, S_3: Διαγράφει πρώτα το I_4, έπειτα τα I_3, I_5, I_2, I_6. $\implies |S_3| = |\{I_1, I_7\}| = 2$.
- Βέλτιστη Λύση, OPT: $\{I_1, I_4, I_7\} \implies |OPT| = 3$.

Άσκηση 1 (α.1): Άπληστα Κριτήρια Δρομολόγησης Μαθημάτων

3. Περισσότερες Επικαλύψεις

Λάθος!

Αντιπαράδειγμα:



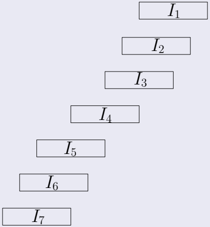
- Λύση Κριτηρίου, S_3: Διαγράφει πρώτα το I_4, έπειτα τα I_3, I_5, I_2, I_6. $\Longrightarrow |S_3| = |\{I_1, I_7\}| = 2$.
- Βέλτιστη Λύση, OPT: $\{I_1, I_4, I_7\} \Longrightarrow |OPT| = 3$.

Άσκηση 1 (α.1): Άπληστα Κριτήρια Δρομολόγησης Μαθημάτων

3. Περισσότερες Επικαλύψεις

Λάθος!

Αντιπαράδειγμα:



- Λύση Κριτηρίου, S_3: Διαγράφει πρώτα το I_4, έπειτα τα I_3, I_5, I_2, I_6. $\Longrightarrow |S_3| = |\{I_1, I_7\}| = 2$.
- Βέλτιστη Λύση, OPT: $\{I_1, I_4, I_7\} \Longrightarrow |OPT| = 3$.

Άσκηση 1 (α.2): Δρομολόγηση Μαθημάτων με Βάρη

Είσοδος: Χρονικά διαστήματα μαθημάτων $[s_i, f_i)$, $i = 1, \ldots, n$ με αντίστοιχες διδακτικές μονάδες w_i.

Ζητείται: Σύνολο μαθημάτων χωρίς επικαλύψεις με μέγιστο συνολικό άθροισμα διδακτικών μονάδων.

Άσκηση 1 (α.2): Δρομολόγηση Μαθημάτων με Βάρη

Λύση:

- Ταξινομούμε τα χρονικά διαστήματα σε αύξουσα σειρά με βάση το χρόνο ολοκλήρωσής τους, έστω για ευκολία $f_1, \ldots, f_n$ αυτή η σειρά.
- Συμβολίζουμε με $C[i]$ το μέγιστο σύνολο διδακτικών μονάδων μίας βέλτιστης λύσης για τα μαθήματα $\{1, \ldots, i\}$.

Προφανώς, η ζητούμενη λύση είναι η $C[n]$ και η σχέση αρχικοποιείται στο $C[1] = w_1$.

Άσκηση 1 (α.2): Δρομολόγηση Μαθημάτων με Βάρη

Λύση:
Για την κατασκευή της αναδρομικής σχέσης, χρειαζόμαστε έναν πίνακα μήκους n, κάθε στοιχείο prev$[i]$ του οποίου θα περιέχει το δείκτη του μαθήματος που τελειώνει αργότερα και δεν έχει επικάλυψη με το i.
Η αναδρομική σχέση είναι:

$$C[i] = \max\{w_i + C[\text{prev}[i]], C[i-1]\}$$

- Ταξινόμηση διαστημάτων και κατασκευή πίνακα prev: $O(n \log n)$.
- Εύρεση ενός εκ των n στοιχείων του πίνακα της C : $O(1)$.

Άρα συνολικά:

Πολυπλοκότητα

$$O(n \log n)$$

Σημείωση: Η ανακατασκευή της λύσης γίνεται από τον πίνακα, σε $O(n)$.

Άσκηση 1 (β): Ανακοινώσεις

Είσοδος: Χρονικά διαστήματα μαθημάτων $[s_i, f_i)$, $i = 1, \ldots, n$.

Ζητείται: Σύνολο στιγμών που αγγίζουν όλα τα μαθήματα και έχουν ελάχιστο πλήθος.

Άσκηση 1 (β): Ανακοινώσεις

Λύση: Θεωρούμε ότι μία ανακοίνωση τη χρονική στιγμή f_i ενημερώνει και τους φοιτητές του μαθήματος i.

- Ταξινομούμε τα χρονικά διαστήματα σε αύξουσα σειρά με βάση το χρόνο ολοκλήρωσής τους, έστω για ευκολία $f_1, \ldots, f_n$ αυτή η σειρά.
- Βάζουμε στη λύση μας το σημείο f_i του πρώτου ακάλυπτου διαστήματος, σημειώνουμε ποιά διαστήματα έχουμε καλύψει και συνεχίζουμε με το επόμενο ακάλυπτο διάστημα. (βλ. "Ασκήσεις σε άπληστους αλγόριθμους και δυναμικό προγραμματισμό")

Πολυπλοκότητα

$$O(n \log n)$$

Άσκηση 1 (β): Ανακοινώσεις

Απόδειξη ορθότητας:

- Έστω P το σύνολο στιγμών που επέλεξε ο αλγόριθμός μας, O το σύνολο της βέλτιστης λύσης και έστω ότι στο πρώτο σημείο στο οποίο διαφέρουν ο αλγόριθμός μας επιλέγει $p = f_k - 1$ (για να καλύψει το διάστημα $[s_k, f_k)$) ενώ η βέλτιστη λύση έχει στην ίδια θέση τη στιγμή o.
- Αν $o > p$, τότε η βέλτιστη λύση δεν αγγίζει το διάστημα k (αφού μέχρι τώρα καλύπτουν τα ίδια διαστήματα) και οι φοιτητές του μαθήματος k μένουν ανενημέρωτοι στη βέλτιστη λύση! Άτοπο.
- Αν $o < p$: Μέχρι τώρα και οι δύο λύσεις έχουν καλύψει όλα τα διαστήματα που τελειώνουν πριν τη στιγμή f_k. Άρα αντικαθιστώντας το o με $p = f_k - 1$ η βέλτιστη λύση συνεχίζει να καλύπτει το $[s_k, f_k)$ και αποκλείεται να χάνει κάποιο άλλο διάστημα γιατί κάθε άλλο ακάλυπτο διάστημα τελειώνει μετά το f_k άρα αν το άγγιζε με το o θα το αγγίζει και με το p.

Outline

Άσκηση 2: Τηλεπαιχνίδι

Είσοδος: Ο παίκτης έχει n σφυριά μιας χρήσης, το καθένα με με δύναμη $f_j > 0$ και n κουτιά, που το καθένα περιέχει $v_i \geq 0$ ευρώ και αντέχει δύναμη $p_i \geq 0$.

Ζητείται: Ένα ταίριασμα M των σφυριών με τα κουτιά ώστε να μεγιστοποιείται η συνολική αξία των σπασμένων κουτιών που κερδίζει ο παίκτης.

$$\sum_{i \in W} v_i, \qquad \text{όπου } W = \{i \mid f_{M(i)} \geq p_i\}$$

Άσκηση 2: Τηλεπαιχνίδι

Άπληστο κριτήριο # 1

Προσπάθησε να σπάσεις κουτιά με όσο δυνατόν μεγαλύτερη αξία.

Άπληστο κριτήριο # 2

Προσπάθησε να μην χρησιμοποιείς σφυριά με μεγάλο f_i εκεί που δεν χρειάζεται.

Λύση

- Ταξινόμησε τα κουτιά σε φθίνουσα σειρά v_i
- Στο κουτί i, αντιστοίχισε το σφυρί με το ελάχιστο δυνατό βάρος ώστε $f_i \geq p_i$.
- Αν για κανένα από τα σφυριά που μένουν, δεν ισχύει $f_i \geq p_i$, αντιστοίχισε το κουτί με το ελάχιστο f_i

Άσκηση 2: Τηλεπαιχνίδι

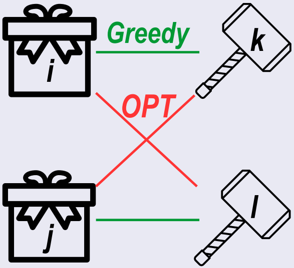
Ορθότητα

Έστω κάποια βέλτιστη λύση του προβλήματος OPT. Εξετάζω τα κουτιά σε φθίνουσα σειρά v_i, και έστω ότι οι λύσεις διαφέρουν για πρώτη φορά στο κουτί i που η Greedy αντιστοιχεί στο σφυρί j. Διακρίνω περιπτώσεις:

- Η Greedy λύση χάνει το κουτί i και η OPT το κερδίζει. Άτοπο!
- Η OPT χάνει το κουτί και η Greedy το κερδίζει. Μπορώ να αντιστοιχίσω το κουτί i στο σφυρί j, στην OPT λύση, χωρίς να χάσω αξία (αφού η OPT χρησιμοποιεί το σφυρί j για να σπάσει - η να μη σπάσει - κουτί μικρότερης αξίας).

Ορθότητα (Συν.)

- Η OPT σπάει το i με σφυρί μικρότερου βάρους. Άτοπο!
- Η OPT σπάει το i με σφυρί μεγαλύτερου βάρους.



Μπορώ να κάνω την αλλαγή $i \to k$, $j \to l$ στην OPT λύση και να μη χαλάσω την αξία της λύσης ($v_i > v_j$).

Με το ίδιο επιχείρημα, μπορώ να κάνω τη βέλτιστη ίδια με την greedy χωρίς να χαλάσω την αξία της. Άρα η greedy είναι βέλτιση. □

Άσκηση 2: Τηλεπαιχνίδι

Χρονική Πολυπλοκότητα

Βήματα αλγορίθμου:

- Ταξινόμηση κουτιών: $O(n \log n)$
- Για κάθε κουτί i, αναζήτηση του ελάχιστου f_i ώστε $f_i \geq p_i$. **Δεν αρκεί ταξινόμηση ως προς** f_i, καθώς τα σφυριά που χρησιμοποιούνται 'χαλάνε' τη δυαδική αναζήτηση. Με χρήση δέντρου δυαδικής αναζήτησης: $O(\log n)$

Συνολικά

$$O(n \log n)$$

Outline

Άσκηση 3: Αναμνηστικά

Είσοδος: n πόλεις, Προυπολογισμός C, Πίνακας p_{ij} συναισθηματικής αξίας, Πίνακας c_{ij} χρηματικής αξίας.

Έξοδος: Επιλογή αντικειμένων $(a_1, \ldots, a_n)$ ένα για κάθε πόλη, ώστε $\sum_{j=1}^{n} c_{ja_j} \leq C$ και να μεγιστοποιείται η συνολική συναισθηματική αξία.

Άσκηση 3: Αναμνηστικά

Διαίσθηση

- Έστω $(a_1^*, \ldots, a_n^*)$ η βέλτιστη λύση του προβλήματος.
- Τότε, η $(a_1^*, \ldots, a_{n-1}^*)$ είναι βέλτιστη λύση του προβλήματος με $C' = C - c_{na_n}$ για τις $n-1$ πόλεις.
- Αρχή βελτιστότητας $\Longrightarrow$ δυναμικός προγραμματισμός!

Άσκηση 3: Αναμνηστικά

Λύση

- Έστω A ένας $n \times C$ πίνακας με $A[i,j]$ η μέγιστη δυνατή συναισθηματική αξία για τις πόλεις 1 έως i με προυπολογισμό j.
- Ακραίες περιπτώσεις:
  - $A[i,j] = -1$ αν τα χρήματα j δεν επαρκούν για να αγοραστεί αντικείμενο από την πόλη i
  - $A[1,j]$ υπολογίζεται εύκολα.
- Αναδρομική σχέση : $A[i,j] = \max_{1 \leq u \leq k_i, c_{iu} \leq j}(A[i-1, j-c_{iu}] + p_{iu})$

Λύση

- Έστω A ένας $n \times C$ πίνακας με $A[i,j]$ η μέγιστη δυνατή συναισθηματική αξία για τις πόλεις 1 έως i με προυπολογισμό j.
- Ακραίες περιπτώσεις:
  - $A[i,j] = -1$ αν τα χρήματα j δεν επαρκούν για να αγοραστεί αντικείμενο από την πόλη i
  - $A[1,j]$ υπολογίζεται εύκολα.
- Αναδρομική σχέση : $A[i,j] = \max_{1 \leq u \leq k_i, c_{iu} \leq j}(A[i-1, j-c_{iu}] + p_{iu})$

Λύση

- Έστω A ένας $n \times C$ πίνακας με $A[i,j]$ η μέγιστη δυνατή συναισθηματική αξία για τις πόλεις 1 έως i με προυπολογισμό j.
- Ακραίες περιπτώσεις:
  - $A[i,j] = -1$ αν τα χρήματα j δεν επαρκούν για να αγοραστεί αντικείμενο από την πόλη i
  - $A[1,j]$ υπολογίζεται εύκολα.
- Αναδρομική σχέση : $A[i,j] = \max_{1 \leq u \leq k_i, c_{iu} \leq j}(A[i-1, j-c_{iu}] + p_{iu})$

Άσκηση 3: Αναμνηστικά

Αλγόριθμος

- Γεμίζουμε τον πίνακα ξεκινώντας από $i = 1$ μέχρι n χρησιμοποιώντας την αναδρομή.
- Η βέλτιστη επιλογή αντικειμένων προκύπτει από τις επιλογές του u που δίνει το max.
- Χρονική πολυπλοκότητα $O(nCK)$, όπου $K = \max_i k_i$.
- Δεν υπάρχει πολυωνυμικός αλγόριθμος, εκτός αν $P = NP$ (αναγωγή από Discrete Knapsack).

Άσκηση 3: Αναμνηστικά

Αλγόριθμος

- Γεμίζουμε τον πίνακα ξεκινώντας από $i = 1$ μέχρι n χρησιμοποιώντας την αναδρομή.
- Η βέλτιστη επιλογή αντικειμένων προκύπτει από τις επιλογές του u που δίνει το max.
- Χρονική πολυπλοκότητα $O(nCK)$, όπου $K = \max_i k_i$.
- Δεν υπάρχει πολυωνυμικός αλγόριθμος, εκτός αν $P = NP$ (αναγωγή από Discrete Knapsack).

Outline

Άσκηση 4: Σοκολατάκια (γνησίως αύξουσα κατανάλωση)

Είσοδος: Ακολουθία n ζευγών (q_i, t_i), όπου q_i το πλήθος από σοκολατάκια που περιέχει το κουτί i και t_i ο τύπος τους, αρχική θέση (κουτί) p και ελάχιστο πλήθος Q σοκολατακίων που πρέπει να καταναλωθούν. Τα κουτιά είναι διατεταγμένα σε γραμμή και το κόστος μετακίνησης από το i στο j είναι $|i - j|$.

Ζητείται: Σε κάθε βήμα μπορούμε να φάμε μόνο περισσότερα σοκολατάκια και διαφορετικού τύπου από το προηγούμενο και ξεκινάμε από το κουτί p. Ποιος είναι ο ελάχιστος χρόνος (και αλληλουχία κουτιών) για να φάμε τουλάχιστον Q σοκολατάκια.

Άσκηση 4: Σοκολατάκια (γνησίως αύξουσα κατανάλωση)

Λύση: Δυναμικός Προγραμματισμός.
Αφού τελειώνουμε με κατανάλωση (δεν συμφέρει να μετακινηθούμε μόλις φάμε Q σοκολατάκια), μπορούμε να ασχοληθούμε με τα υπο-προβλήματα $c[i,j]$ όπου όπου $c[i,j]$ ο ελάχιστος χρόνος που χρειαζόμαστε για να φάμε i σοκολατάκια, τρόγωντας στο τέλος από το κουτί j.

Αναδρομική Σχέση

$$c[min\{q_j + q, Q\}][j] = \min_{\substack{\{i \mid i \in \{1,..,n\}, \\ q_i < q_j, \\ t_i \neq t_j\}}} \{c[q][i] + |i - j|\}$$

Αρχή βελτιστότητας: Αν η βέλτιστη ακολουθία κουτιών που τρώμε, για να φάμε i σοκολατάκια καταλήγοντας στο κουτί j είναι k_a, k_b, k_o, k_i, η βέλτιστη ακολουθία για να φάμε $i - q_j$ σοκολατάκια καταλήγοντας στο κουτί k_o είναι k_a, k_b, k_o.

Άσκηση 4: Σοκολατάκια (γνησίως αύξουσα κατανάλωση)

Λύση: Δυναμικός Προγραμματισμός.
Αφού τελειώνουμε με κατανάλωση (δεν συμφέρει να μετακινηθούμε μόλις φάμε Q σοκολατάκια), μπορούμε να ασχοληθούμε με τα υπο-προβλήματα $c[i,j]$ όπου όπου $c[i,j]$ ο ελάχιστος χρόνος που χρειαζόμαστε για να φάμε i σοκολατάκια, τρόγωντας στο τέλος από το κουτί j.

Αναδρομική Σχέση

$$c[min\{q_j + q, Q\}][j] = \min_{\substack{\{i \mid i \in \{1,..,n\}, \\ q_i < q_j, \\ t_i \neq t_j\}}} \{c[q][i] + |i - j|\}$$

Αρχή βελτιστότητας: Αν η βέλτιστη ακολουθία κουτιών που τρώμε, για να φάμε i σοκολατάκια καταλήγοντας στο κουτί j είναι k_a, k_b, k_o, k_i, η βέλτιστη ακολουθία για να φάμε $i - q_j$ σοκολατάκια καταλήγοντας στο κουτί k_o είναι k_a, k_b, k_o.

Λύση:

$$c[min\{q_j + q, Q\}][j] = \min_{\substack{\{i \mid i \in \{1,..,n\}, \\ q_i < q_j, \\ t_i \neq t_j\}}} \{c[q][i] + |i - j|\}$$

- Τρέχουμε για q από 1 μέχρι Q. Σε κάθε βήμα, για κάθε κουτί j υπολογίζουμε το πόσο χρειαζόμαστε για να φάμε $q + q_j$ σοκολατάκια φτάνοντας στο j.
- Αφού $q_i > 0, \forall i$, αν αρχικοποιήσουμε σωστά τις αρχικές καταστάσεις, υπολογίζουμε τα υποπροβλήματα με τη σωστή σειρά.
- Αρχικοποίηση: $c[i][j] = \infty$ και $\forall j \in n, c[\min\{q_j, Q\}][j] = |p - j|$
- Επιστρέφουμε το $\min_{j \in \{1,...,n\}} c[Q][j]$.

Λύση:

$$c[min\{q_j + q, Q\}][j] = \min_{\substack{\{i | i \in \{1,..,n\},\\ q_i < q_j,\\ t_i \neq t_j\}}} \{c[q][i] + |i - j|\}$$

- Τρέχουμε για q από 1 μέχρι Q. Σε κάθε βήμα, για κάθε κουτί j υπολογίζουμε το πόσο χρειαζόμαστε για να φάμε $q + q_j$ σοκολατάκια φτάνοντας στο j.
- Αφού $q_i > 0, \forall i$, αν αρχικοποιήσουμε σωστά τις αρχικές καταστάσεις, υπολογίζουμε τα υποπροβλήματα με τη σωστή σειρά.
- Αρχικοποίηση: $c[i][j] = \infty$ και $\forall j \in n, c[\min\{q_j, Q\}][j] = |p - j|$
- Επιστρέφουμε το $\min_{j \in \{1,...,n\}} c[Q][j]$.

Άσκηση 4: Σοκολατάκια (γνησίως αύξουσα κατανάλωση)

Λύση:

$$c[min\{q_j + q, Q\}][j] = \min_{\substack{\{i | i \in \{1,..,n\},\\ q_i < q_j,\\ t_i \neq t_j\}}} \{c[q][i] + |i - j|\}$$

- Τρέχουμε για q από 1 μέχρι Q. Σε κάθε βήμα, για κάθε κουτί j υπολογίζουμε το πόσο χρειαζόμαστε για να φάμε $q + q_j$ σοκολατάκια φτάνοντας στο j.
- Αφού $q_i > 0, \forall i$, αν αρχικοποιήσουμε σωστά τις αρχικές καταστάσεις, υπολογίζουμε τα υποπροβλήματα με τη σωστή σειρά.
- Αρχικοποίηση: $c[i][j] = \infty$ και $\forall j \in n, c[\min\{q_j, Q\}][j] = |p - j|$
- Επιστρέφουμε το $\min_{j \in \{1,...,n\}} c[Q][j]$.

Άσκηση 4: Σοκολατάκια (γνησίως αύξουσα κατανάλωση)

Λύση:

$$c[min\{q_j + q, Q\}][j] = \min_{\substack{\{i|i\in\{1,..,n\},\\ q_i<q_j,\\ t_i\neq t_j\}}} \{c[q][i] + |i-j|\}$$

- Τρέχουμε για q από 1 μέχρι Q. Σε κάθε βήμα, για κάθε κουτί j υπολογίζουμε το πόσο χρειαζόμαστε για να φάμε $q + q_j$ σοκολατάκια φτάνοντας στο j.
- Αφού $q_i > 0, \forall i$, αν αρχικοποιήσουμε σωστά τις αρχικές καταστάσεις, υπολογίζουμε τα υποπροβλήματα με τη σωστή σειρά.
- Αρχικοποίηση: $c[i][j] = \infty$ και $\forall j \in n, c[\min\{q_j, Q\}][j] = |p - j|$
- Επιστρέφουμε το $\min_{j\in\{1,...,n\}} c[Q][j]$.

Άσκηση 4: Σοκολατάκια (γνησίως αύξουσα κατανάλωση)

Λύση:

$$c[min\{q_j + q, Q\}][j] = \min_{\substack{\{i | i \in \{1,..,n\},\\ q_i < q_j,\\ t_i \neq t_j\}}} \{c[q][i] + |i - j|\}$$

- Τρέχουμε για q από 1 μέχρι Q. Σε κάθε βήμα, για κάθε κουτί j υπολογίζουμε το πόσο χρειαζόμαστε για να φάμε $q + q_j$ σοκολατάκια φτάνοντας στο j.
- Αφού $q_i > 0, \forall i$, αν αρχικοποιήσουμε σωστά τις αρχικές καταστάσεις, υπολογίζουμε τα υποπροβλήματα με τη σωστή σειρά.
- Αρχικοποίηση: $c[i][j] = \infty$ και $\forall j \in n, c[\min\{q_j, Q\}][j] = |p - j|$
- Επιστρέφουμε το $\min_{j \in \{1,...,n\}} c[Q][j]$.

Άσκηση 4: Σοκολατάκια (γνησίως αύξουσα κατανάλωση)

Πολυπλοκότητα

$O(Qn^2)$ για την επίλυση των $Q * n$ υποπροβλημάτων του $c[i][j]$ (n συγκρίσεις στο καθένα) $+$ $O(n)$ για την εύρεση του ελαχίστου $c[Q][j] =$

$$O(Qn^2)$$

.

Ορθότητα: Αφού σε κάθε επανάληψη, για κάθε κουτί j μελετάμε ως πιθανά 'προηγούμενα' μόνο κουτιά i με q_i αυστηρά μικρότερο του q_j, οποιαδήποτε αλληλουχία κουτιών για κάθε υποπρόβλημα του $c[i][j]$ είναι αδύνατον να περιέχει κύκλο (να χρησιμοποιεί το ίδιο κουτί πάνω από μια φορά).

Άσκηση 4: Σοκολατάκια (γνησίως αύξουσα κατανάλωση)

Πολυπλοκότητα

$O(Qn^2)$ για την επίλυση των $Q * n$ υποπροβλημάτων του $c[i][j]$ (n συγκρίσεις στο καθένα) $+$ $O(n)$ για την εύρεση του ελαχίστου $c[Q][j]$ $=$

$$O(Qn^2)$$

.

Ορθότητα: Αφού σε κάθε επανάληψη, για κάθε κουτί j μελετάμε ως πιθανά 'προηγούμενα' μόνο κουτιά i με q_i αυστηρά μικρότερο του q_j, οποιαδήποτε αλληλουχία κουτιών για κάθε υποπρόβλημα του $c[i][j]$ είναι αδύνατον να περιέχει κύκλο (να χρησιμοποιεί το ίδιο κουτί πάνω από μια φορά).

Άσκηση 4: Σοκολατάκια (γενική μορφή)

Ορισμός: Ίδιο πρόβλημα με πριν, μόνο που τώρα επιτρέπεται να τρώμε περισσότερα η ίσα σε πλήθος σοκολατάκια σε κάθε κατανάλωση από την προηγούμενη.

Άσκηση 4: Σοκολατάκια (γενική μορφή)

Πρόβλημα: Αφού ο αλγόριθμος δεν 'θυμάται' σε κάθε βήμα από ποια κουτιά έχει φάει, σε περίπτωση που επιτρέψουμε την κατανάλωση ίσου αριθμού από σοκολατάκια από βήμα σε βήμα, ο αλγόριθμος μπορεί να έχει κάνει κύκλους μεταξύ κουτιών με ίδιο αριθμό από σοκολατάκια. Δηλαδή, η λογική της ορθότητας της προηγούμενης μορφής του προβλήματος δεν ισχύει.

Λύση: Θα ορίσουμε έναν πίνακα 'μνήμης' $path[q][j][i]$ με την σύμβαση $path[q][j][i] = 0$ αν η ακολουθία κουτιών που τρώμε για να καταναλώσουμε στον ελάχιστο δυνατό χρόνο q σοκολατάκια τρώγοντας στο τέλος από το κουτί j δεν περιλαμβάνει το κουτί i, και $path[q][j][i] = 1$ αλλιώς.

Άσκηση 4: Σοκολατάκια (γενική μορφή)

Πρόβλημα: Αφού ο αλγόριθμος δεν 'θυμάται' σε κάθε βήμα από ποια κουτιά έχει φάει, σε περίπτωση που επιτρέψουμε την κατανάλωση ίσου αριθμού από σοκολατάκια από βήμα σε βήμα, ο αλγόριθμος μπορεί να έχει κάνει κύκλους μεταξύ κουτιών με ίδιο αριθμό από σοκολατάκια. Δηλαδή, η λογική της ορθότητας της προηγούμενης μορφής του προβλήματος δεν ισχύει.

Λύση: Θα ορίσουμε έναν πίνακα 'μνήμης' $path[q][j][i]$ με την σύμβαση $path[q][j][i] = 0$ αν η ακολουθία κουτιών που τρώμε για να καταναλώσουμε στον ελάχιστο δυνατό χρόνο q σοκολατάκια τρώγοντας στο τέλος από το κουτί j δεν περιλαμβάνει το κουτί i, και $path[q][j][i] = 1$ αλλιώς.

Άσκηση 4: Σοκολατάκια (γενική μορφή)

Λύση: Έτσι, η παραπάνω αναδρομική σχέση μετατρέπεται ως εξής:

$$c[min\{q_j + q, Q\}][j] = \min_{\substack{\{i | i \in \{1,..,n\}, \\ q_i < q_j \vee (q_i = q_j \wedge path[q][i][j] = 0), \\ t_i \neq t_j\}}} \{c[q][i] + |i - j|\}$$

- Σε κάθε βήμα ανανεώνουμε κατάλληλα τον πίνακα *path*.
- Αυτό σημαίνει ότι αν για προηγούμενο του κουτιού j χρησιμοποιήσουμε το i στη βέλτιστη κατανάλωση $q + q_j$ σοκολατιών (δηλ. το i ελαχιστοποιεί την σχέση) πρέπει να θέσουμε $path[q + q_j][j][i'] = 1$ για $i' = i$ και για κάθε i' που ανήκει στο μονοπάτι αυτής της λύσης μέχρι το κουτί i.

Άσκηση 4: Σοκολατάκια (γενική μορφή)

<u>Λύση</u>: Έτσι, η παραπάνω αναδρομική σχέση μετατρέπεται ως εξής:

$$c[min\{q_j + q, Q\}][j] = \min_{\substack{\{i | i \in \{1,..,n\}, \\ q_i < q_j \vee (q_i = q_j \wedge path[q][i][j]=0), \\ t_i \neq t_j\}}} \{c[q][i] + |i - j|\}$$

- Σε κάθε βήμα ανανεώνουμε κατάλληλα τον πίνακα *path*.
- Αυτό σημαίνει ότι αν για προηγούμενο του κουτιού j χρησιμοποιήσουμε το i στη βέλτιστη κατανάλωση $q + q_j$ σοκολατιών (δηλ. το i ελαχιστοποιεί την σχέση) πρέπει να θέσουμε $path[q + q_j][j][i'] = 1$ για $i' = i$ και για κάθε i' που ανήκει στο μονοπάτι αυτής της λύσης μέχρι το κουτί i.

Άσκηση 4: Σοκολατάκια (γενική μορφή)

Λύση: Έτσι, η παραπάνω αναδρομική σχέση μετατρέπεται ως εξής:

$$c[min\{q_j + q, Q\}][j] = \min_{\substack{\{i | i \in \{1,..,n\},\\ q_i < q_j \vee (q_i = q_j \wedge path[q][i][j] = 0),\\ t_i \neq t_j\}}} \{c[q][i] + |i - j|\}$$

- Σε κάθε βήμα ανανεώνουμε κατάλληλα τον πίνακα *path*.
- Αυτό σημαίνει ότι αν για προηγούμενο του κουτιού j χρησιμοποιήσουμε το i στη βέλτιστη κατανάλωση $q + q_j$ σοκολατιών (δηλ. το i ελαχιστοποιεί την σχέση) πρέπει να θέσουμε $path[q + q_j][j][i'] = 1$ για $i' = i$ και για κάθε i' που ανήκει στο μονοπάτι αυτής της λύσης μέχρι το κουτί i.

Άσκηση 4: Σοκολατάκια (γενική μορφή)

Πολυπλοκότητα

Σε κάθε αναδρομή έχω επιπλέον n πράξεις για τις ενημερώσεις του πίνακα *path*. Οπότε, πάλι έχουμε $O(Q2n^2) = O(Qn^2)$.

Outline

Άσκηση 5: Πομποί και Δέκτες

Είσοδος: Ακολουθία n (άρτιος) ζευγών (T_i, R_i), όπου T_i η κατανάλωση ισχύος της κεραίας i όταν λειτουργεί ως πομπός και $R_i \leq T_i$ όταν λειτουργεί ως δέκτης.

Ζητείται: Ελάχιστη συνολική κατανάλωση ισχύος, όπου για κάθε i που λειτουργεί ως πομπός υπάρχει ένα και μοναδικό $j > i$ που λειτουργεί ως δέκτης.

Λύση: Δυναμικός Προγραμματισμός

Αναδρομική Σχέση

$$C[i,j] = \min\{C[i-1,j+1] + R_i, C[i-1,j-1] + T_i\}$$

όπου $C[i,j]$ το κόστος της βέλτιστης λύσης μέχρι και την i-οστή κεραία, έχοντας $j \leq i$ πομπούς που δεν έχουν αντιστοιχηθεί σε δέκτες προς το παρόν.

Άσκηση 5: Πομποί και Δέκτες

Λύση:

$$C[i,j] = \min\{C[i-1,j+1] + R_i, C[i-1,j-1] + T_i\}$$

Αν έχουμε τη βέλτιστη λύση για $i-1$ κεραίες τότε:

- Είτε υπάρχουν $j+1$ πομποί που δεν έχουν αντιστοιχηθεί σε δέκτες και η i-οστή κεραία γίνεται δέκτης οπότε μένουν j πομποί που δεν έχουν αντιστοιχηθεί σε δέκτες και προστίθεται R_i στη συνολική κατανάλωση ισχύος.
- Είτε υπάρχουν $j-1$ πομποί που δεν έχουν αντιστοιχηθεί σε δέκτες και η i-οστή κεραία γίνεται πομπός οπότε μένουν j πομποί που δεν έχουν αντιστοιχηθεί σε δέκτες και προστίθεται T_i στη συνολική κατανάλωση ισχύος.

Λύση:
Η ζητούμενη λύση είναι η $C[n, 0]$.

Πολυπλοκότητα

Υπολογισμός ενός εκ των $\Theta(n^2/2)$ στοιχείων του C σε $O(1) \Longrightarrow$

$$O(n^2)$$

Υπάρχει πιο αποδοτικός αλγόριθμος;

Άσκηση 5: Πομποί και Δέκτες

Λύση:

Η ζητούμενη λύση είναι η $C[n, 0]$.

Πολυπλοκότητα

Υπολογισμός ενός εκ των $\Theta(n^2/2)$ στοιχείων του C σε $O(1) \Longrightarrow$

$$O(n^2)$$

Υπάρχει πιο αποδοτικός αλγόριθμος;

Άσκηση 5: Πομποί και Δέκτες

Καλύτερη Λύση: Για κάποια ανάθεση πομπών-δεκτών έστω $S[i]$ το πλήθος πομπών μείον το πλήθος δεκτών στο διάστημα $[1, i]$. Μία ανάθεση είναι έγκυρη αν και μόνο αν για κάθε $i \in \{1, ..., n\}$ ισχύει $S[i] \geq 0$ και $S[n] = 0$.

Θεωρούμε ότι έχουμε μια βέλτιστη έγκυρη ανάθεση στο διάστημα $[1, i]$. Αν ο i είναι περιττός, για να την επεκτείνουμε βάζουμε ένα δέκτη στο $i + 1$ ($S[i + 1] = 0$). Αν είναι άρτιος, βάζουμε ένα δέκτη στο $i + 1$, αλλά έχουμε $S[i + 1] = -1$, άρα κάποιος δέκτης στο $[1, i + 1]$ πρέπει να μετατραπεί σε πομπό: Μετατρέπουμε αυτόν που ελαχιστοποιεί την κατανάλωση, δηλαδή αυτόν με το ελάχιστο $T_j - R_j$. Τώρα έχουμε $S[i + 1] = 0$.

Για να βρίσκουμε το δέκτη με το ελάχιστο $T_j - R_j$ θα χρειαστούμε ένα min-heap στο οποίο οι λειτουργίες εισαγωγής και εξαγωγής ελαχίστου στοιχείου γίνονται το πολύ σε $O(\log n)$.

Άσκηση 5: Πομποί και Δέκτες

Αλγόριθμος:

- Αρχικοποίησε ένα κενό min-heap Q
- Για $i = 1 \ldots n$
  - Θεώρησε τον i ως δέκτη, πρόσθεσε R_i στο συνολικό κόστος και βάλε το ζεύγος $(T_i - R_i, i)$ στο Q
  - Αν $S[i] < 0$, αφαίρεσε από το Q το δέκτη με το ελάχιστο $T_j - R_j$, μετάτρεψέ τον σε πομπό και αύξησε το συνολικό κόστος κατά $T_j - R_j$.

Χρονική Πολυπλοκότητα

$$O(n \log n)$$

Μένει να δείξουμε το επαγωγικό βήμα, δηλαδή ότι η λύση στο διάστημα $[1, i+1]$ είναι βέλτιστη, δεδομένου ότι η λύση στο $[1, i]$ είναι βέλτιστη.

- Αν i περιττός ($S[i] = 1$), τότε προφανώς η θέση $i+1$ πρέπει να έχει δέκτη για να είναι έγκυρη η λύση και γιατί είναι το φθηνότερο. Άρα η λύση μας είναι βέλτιστη.
- Αν i άρτιος ($S[i] = 0$) τότε αν προσθέσουμε δέκτη στη θέση $i+1$ για να είναι έγκυρη η λύση πρέπει να μετατρέψουμε ένα δέκτη σε πομπό. Προφανώς αφού η λύση είναι μέχρι στιγμής ελαχίστου κόστους, προσθέτοντας το μικρότερο $T_j - R_j$ λόγω της αλλαγής (και R_{i+1} αντί T_{i+1}) έχουμε λύση μικρότερου δυνατού κόστους άρα ακόμα βέλτιστη.

Παράδειγμα:

i	1	2	3	4	5	6	7	8
$S(i)$								
T_i	8	9	5	6	10	30	12	2
R_i	6	3	1	4	3	7	5	1

Κατανάλωση: 0

Άσκηση 5: Πομποί και Δέκτες

Παράδειγμα:

i	1	2	3	4	5	6	7	8
$S(i)$	-1							
T_i	8	9	5	6	10	30	12	2
R_i	6	3	1	4	3	7	5	1

Κατανάλωση: $0 + 6 = 6$

Παράδειγμα:

i	1	2	3	4	5	6	7	8
$S(i)$	1							
T_i	8	9	5	6	10	30	12	2
R_i	6	3	1	4	3	7	5	1

Κατανάλωση: $6 + (8\text{-}6) = 8$

Άσκηση 5: Πομποί και Δέκτες

Παράδειγμα:

i	1	2	3	4	5	6	7	8
$S(i)$	1	0						
T_i	8	9	5	6	10	30	12	2
R_i	6	3	1	4	3	7	5	1

Κατανάλωση: $8 + 3 = 11$

Παράδειγμα:

i	1	2	3	4	5	6	7	8
$S(i)$	1	0	-1					
T_i	8	9	5	6	10	30	12	2
R_i	6	3	1	4	3	7	5	1

Κατανάλωση: $11 + 1 = 12$

Παράδειγμα:

i	1	2	3	4	5	6	7	8
$S(i)$	1	0	1					
T_i	8	9	5	6	10	30	12	2
R_i	6	3	1	4	3	7	5	1

Κατανάλωση: $12 + (5\text{-}1) = 16$

Παράδειγμα:

i	1	2	3	4	5	6	7	8
$S(i)$	1	0	1	0				
T_i	8	9	5	6	10	30	12	2
R_i	6	3	1	4	3	7	5	1

Κατανάλωση: $16 + 4 = 20$

Άσκηση 5: Πομποί και Δέκτες

Παράδειγμα:

i	1	2	3	4	5	6	7	8
$S(i)$	1	0	1	0	-1			
T_i	8	9	5	6	10	30	12	2
R_i	6	3	1	4	3	7	5	1

Κατανάλωση: $20 + 3 = 23$

Παράδειγμα:

i	1	2	3	4	5	6	7	8
$S(i)$	1	0	1	2	1			
T_i	8	9	5	6	10	30	12	2
R_i	6	3	1	4	3	7	5	1

Κατανάλωση: 23 + (6-4)=25

Άσκηση 5: Πομποί και Δέκτες

Παράδειγμα:

i	1	2	3	4	5	6	7	8
$S(i)$	1	0	1	2	1	0		
T_i	**8**	9	**5**	**6**	10	30	12	2
R_i	6	**3**	1	4	**3**	**7**	5	1

Κατανάλωση: 25+7=32

Παράδειγμα:

i	1	2	3	4	5	6	7	8
$S(i)$	1	0	1	2	1	0	-1	
T_i	8	9	5	6	10	30	12	2
R_i	6	3	1	4	3	7	5	1

Κατανάλωση: 32+5=37

Παράδειγμα:

i	1	2	3	4	5	6	7	8
$S(i)$	1	2	3	4	3	2	1	
T_i	8	9	5	6	10	30	12	2
R_i	6	3	1	4	3	7	5	1

Κατανάλωση: 37+(9-3)=43

Άσκηση 5: Πομποί και Δέκτες

<u>Παράδειγμα</u>:

i	1	2	3	4	5	6	7	8
$S(i)$	1	2	3	4	3	2	1	0
T_i	8	9	5	6	10	30	12	2
R_i	6	3	1	4	3	7	5	1

Κατανάλωση: 43+1=44

Outline

1η Προγραμματιστική: Κύριος Creosot

Είσοδος: Ακολουθία συντελεστών γευστικής απόλαυσης εδεσμάτων $x_1, \ldots, x_N$ και τρεις ακέραιοι a, b, c, οι μυστικές γευστικές παράμετροι.

Έξοδος: Μέγιστος συνολικός συντελεστής γευστικής απόλαυσης από τον χωρισμό των N εδεσμάτων σε πιάτα. Κάθε πιάτο πρέπει να περιέχει διαδοχικά εδέσματα $(i, i+1, \ldots, i+k)$ και έχει συντελεστή απόλαυσης $ax^2 + bx + c$ όπου $x = x_i + x_{i+1} + \ldots + x_{i+k}$.

1η Προγραμματιστική: Κύριος Creosot

Λύση: Δυναμικός προγραμματισμός.

- Συμβολίζουμε με $C[i]$ το μέγιστο δυνατό συνολικό συντελεστή χρησιμοποιώντας μόνο τα εδέσματα $1, 2, ..., i$.
- Αν έχουμε υπολογίσει όλα τα $C[j]$, για $0 < j < i$ ελέγχουμε κάθε δυνατό σπάσιμο $(0, ..., j-1), (j, ..., i)$ και διαλέγουμε τη λύση που μεγιστοποιεί τον συντελεστή.

Αναδρομική σχέση

$$C[i] = \max_{0<j<i} \{C[j-1] + a(\sum_{k=j}^{i} x_k)^2 + b(\sum_{k=j}^{i} x_k) + c\}$$

Λύση: Δυναμικός προγραμματισμός.

- Συμβολίζουμε με $C[i]$ το μέγιστο δυνατό συνολικό συντελεστή χρησιμοποιώντας μόνο τα εδέσματα $1, 2, ..., i$.
- Αν έχουμε υπολογίσει όλα τα $C[j]$, για $0 < j < i$ ελέγχουμε κάθε δυνατό σπάσιμο $(0, ..., j-1), (j, ..., i)$ και διαλέγουμε τη λύση που μεγιστοποιεί τον συντελεστή.

Αναδρομική σχέση

$$C[i] = \max_{0<j<i}\{C[j-1] + a(\sum_{k=j}^{i} x_k)^2 + b(\sum_{k=j}^{i} x_k) + c\}$$

Λύση: Δυναμικός προγραμματισμός.

- Συμβολίζουμε με $C[i]$ το μέγιστο δυνατό συνολικό συντελεστή χρησιμοποιώντας μόνο τα εδέσματα $1, 2, ..., i$.
- Αν έχουμε υπολογίσει όλα τα $C[j]$, για $0 < j < i$ ελέγχουμε κάθε δυνατό σπάσιμο $(0, ..., j-1), (j, ..., i)$ και διαλέγουμε τη λύση που μεγιστοποιεί τον συντελεστή.

Αναδρομική σχέση

$$C[i] = \max_{0<j<i} \{C[j-1] + a(\sum_{k=j}^{i} x_k)^2 + b(\sum_{k=j}^{i} x_k) + c\}$$

Λύση: Με χρήση πίνακα αθροισμάτων S με $S[i] = \sum_{k=1}^{i} x_i$, η σχέση γίνεται:

$$C[i] = \max_{0<j<i} \{C[j-1] + a(S[i] - S[j-1])^2 + b(S[i] - S[j-1]) + c\}$$

- Επιστρέφουμε τον όρο $C[N]$.

Πολυπλοκότητα

$$\Theta(N^2)$$

1η Προγραμματιστική: Κύριος Creosot

Λύση: Με χρήση πίνακα αθροισμάτων S με $S[i] = \sum_{k=1}^{i} x_i$, η σχέση γίνεται:

$$C[i] = \max_{0<j<i} \{C[j-1] + a(S[i] - S[j-1])^2 + b(S[i] - S[j-1]) + c\}$$

- Επιστρέφουμε τον όρο $C[N]$.

Πολυπλοκότητα

$$\Theta(N^2)$$

Λύση: Με χρήση πίνακα αθροισμάτων S με $S[i] = \sum_{k=1}^{i} x_i$, η σχέση γίνεται:

$$C[i] = \max_{0<j<i} \{C[j-1] + a(S[i] - S[j-1])^2 + b(S[i] - S[j-1]) + c\}$$

- Επιστρέφουμε τον όρο $C[N]$.

Πολυπλοκότητα

$$\Theta(N^2)$$

Βελτίωση: Μπορούμε να ρίξουμε την πολυπλοκότητα σε γραμμική, η $\Theta(NlogN)$, χρησιμοποιώντας το Convex Hull Optimization.
Μετά από πράξεις:

$$C[i] = aS^2[i] + bS[i] + c + \max_{0<j<i}\{(C[j] + aS^2[j] - bS[j]) - 2aS[j]S[i]\}$$

- Οι όροι έξω από το max εξαρτώνται μόνο από το i και δεν επηρεάζουν τη μεγιστοποίηση της συνάρτησης.
- Θέτουμε: $a = -2aS[j]$, $b = C[j] + aS^2[j] - bS[j]$, $y = C[i]$ και $x = S[i]$. Έτσι έχουμε συνάρτηση μορφής $y = ax + b$, όπου τα a, b εξαρτώνται από το j.
- Άρα θέλουμε την ευθεία που μεγιστοποιεί το y στη θέση $x = S[i]$

1η Προγραμματιστική: Κύριος Creosot (bonus)

Βελτίωση: Μπορούμε να ρίξουμε την πολυπλοκότητα σε γραμμική, η $\Theta(N log N)$, χρησιμοποιώντας το Convex Hull Optimization.
Μετά από πράξεις:

$$C[i] = aS^2[i] + bS[i] + c + \max_{0<j<i}\{(C[j] + aS^2[j] - bS[j]) - 2aS[j]S[i]\}$$

- Οι όροι έξω από το max εξαρτώνται μόνο από το i και δεν επηρεάζουν τη μεγιστοποίηση της συνάρτησης.
- Θέτουμε: $a = -2aS[j]$, $b = C[j] + aS^2[j] - bS[j]$, $y = C[i]$ και $x = S[i]$. Έτσι έχουμε συνάρτηση μορφής $y = ax + b$, όπου τα a, b εξαρτώνται από το j.
- Άρα θέλουμε την ευθεία που μεγιστοποιεί το y στη θέση $x = S[i]$

Βελτίωση: Μπορούμε να ρίξουμε την πολυπλοκότητα σε γραμμική, η $\Theta(NlogN)$, χρησιμοποιώντας το Convex Hull Optimization.
Μετά από πράξεις:

$$C[i] = aS^2[i] + bS[i] + c + \max_{0<j<i}\{(C[j] + aS^2[j] - bS[j]) - 2aS[j]S[i]\}$$

- Οι όροι έξω από το max εξαρτώνται μόνο από το i και δεν επηρεάζουν τη μεγιστοποίηση της συνάρτησης.
- Θέτουμε: $a = -2aS[j]$, $b = C[j] + aS^2[j] - bS[j]$, $y = C[i]$ και $x = S[i]$. Έτσι έχουμε συνάρτηση μορφής $y = ax + b$, όπου τα a, b εξαρτώνται από το j.
- Άρα θέλουμε την ευθεία που μεγιστοποιεί το y στη θέση $x = S[i]$

1η Προγραμματιστική: Κύριος Creosot (bonus)

Βελτίωση: Μπορούμε να ρίξουμε την πολυπλοκότητα σε γραμμική, η $\Theta(NlogN)$, χρησιμοποιώντας το Convex Hull Optimization.
Μετά από πράξεις:

$$C[i] = aS^2[i] + bS[i] + c + \max_{0<j<i}\{(C[j] + aS^2[j] - bS[j]) - 2aS[j]S[i]\}$$

- Οι όροι έξω από το max εξαρτώνται μόνο από το i και δεν επηρεάζουν τη μεγιστοποίηση της συνάρτησης.
- Θέτουμε: $a = -2aS[j]$, $b = C[j] + aS^2[j] - bS[j]$, $y = C[i]$ και $x = S[i]$. Έτσι έχουμε συνάρτηση μορφής $y = ax + b$, όπου τα a, b εξαρτώνται από το j.
- Άρα θέλουμε την ευθεία που μεγιστοποιεί το y στη θέση $x = S[i]$

Outline

2η Προγραμματιστική: Social Credits

<u>Είσοδος</u>: Ακολουθία N φυσικών $s_1, \ldots, s_N$, τα social credits των ατόμων που μπορούμε να συναντήσουμε, με τη σειρά που μπορούν να γίνουν οι συναντήσεις.

<u>Έξοδος</u>: Ο μέγιστος αριθμός ατόμων που μπορούμε να συναντήσουμε, έτσι ώστε κάθε άτομο που συναντάμε, με μια το πολύ εξαίρεση, να έχει υψηλότερο social credit από το προηγούμενο.

Λύση:

- Κάθε εφικτή λύση 'σπάει' σε δυο αύξουσες υποακολουθίες.
- Άρα ψάχνουμε ένα σημείο $k \in 1, 2, ..N$ που μεγιστοποιεί το άθροισμα της μέγιστης αύξουσας υποακολουθίας που ξεκινάει από θέση $s_l \geq 1$ και τελειώνει στην θέση $f_l \leq k$ και της μέγιστης φθίνουσας υποακλουθίας που ξεκινά από την θέση $f_r \leq N$ και τελειώνει σε θέση $s_r > k$.
- Τρέχουμε τον γνωστό αλγόριθμο δυναμικού προγραμματισμού για longest increasing subsequences (LIS) δυο φορές. Μια για να βρούμε όλα τα LIS από τη θέση 1 μέχρι τις θέσεις j, $\forall j | 1 \leq j \leq N$ και μια για όλα τα longest decreasing subsequences (LDS) από τη θέση N μέχρι τις θέσεις l, $N \geq l \geq 1$.

Λύση:

- Κάθε εφικτή λύση 'σπάει' σε δυο αύξουσες υποακολουθίες.
- Άρα ψάχνουμε ένα σημείο $k \in 1, 2, ..N$ που μεγιστοποιεί το άθροισμα της μέγιστης αύξουσας υποακολουθίας που ξεκινάει από θέση $s_l \geq 1$ και τελειώνει στην θέση $f_l \leq k$ και της μέγιστης φθίνουσας υποακλουθίας που ξεκινά από την θέση $f_r \leq N$ και τελειώνει σε θέση $s_r > k$.
- Τρέχουμε τον γνωστό αλγόριθμο δυναμικού προγραμματισμού για longest increasing subsequences (LIS) δυο φορές. Μια για να βρούμε όλα τα LIS από τη θέση 1 μέχρι τις θέσεις j, $\forall j | 1 \leq j \leq N$ και μια για όλα τα longest decreasing subsequences (LDS) από τη θέση N μέχρι τις θέσεις l, $N \geq l \geq 1$.

Λύση:

- Κάθε εφικτή λύση 'σπάει' σε δυο αύξουσες υποακολουθίες.
- Άρα ψάχνουμε ένα σημείο $k \in 1, 2, ..N$ που μεγιστοποιεί το άθροισμα της μέγιστης αύξουσας υποακολουθίας που ξεκινάει από θέση $s_l \geq 1$ και τελειώνει στην θέση $f_l \leq k$ και της μέγιστης φθίνουσας υποακλουθίας που ξεκινά από την θέση $f_r \leq N$ και τελειώνει σε θέση $s_r > k$.
- Τρέχουμε τον γνωστό αλγόριθμο δυναμικού προγραμματισμού για longest increasing subsequences (LIS) δυο φορές. Μια για να βρούμε όλα τα LIS από τη θέση 1 μέχρι τις θέσεις j, $\forall j | 1 \leq j \leq N$ και μια για όλα τα longest decreasing subsequences (LDS) από τη θέση N μέχρι τις θέσεις l, $N \geq l \geq 1$.

2η Προγραμματιστική: Social Credits

Longest increasing subsequences: Υπολογισμός με γνωστό αλγόριθμο σε $O(NlogN)$:
Έστω πίνακας $low[]$ όπου $low[l] = i$ ο δείκτης για τον οποίο το s_i να είναι το μικρότερο στοιχείο στο οποίο τελειώνει κάποια αύξουσα ακολουθία μήκους l και L_{max}, το μέγιστο μήκος υποακολουθίας που έχει βρεθεί (αρχικοποιούμε με 1).

- Για κάθε i από 1 μέχρι N:
  - Ψάχνουμε το $\max\limits_{l \in \{0,\ldots,L_{max}\} | s_{low[l]} < s_i} \{l\}$, με δυαδική αναζήτηση στα $\{s_{low[1]}, s_{low[2]}, \ldots\}$.
  - Το μήκος της μεγαλύτερης υποακολουθίας που τελειώνει στο s_i, $LIS[i]$ θα είναι $l + 1$ και θέτουμε $low(l + 1) = i$.
  - Προσαυξάνουμε κατάλληλα το L_{max} (και φυσικά θυμόμαστε για κάθε $LIS[i]$ το προηγούμενο στοιχείο στην ακολουθία).

Longest increasing subsequences: Υπολογισμός με γνωστό αλγόριθμο σε $O(NlogN)$:
Έστω πίνακας $low[]$ όπου $low[l] = i$ ο δείκτης για τον οποίο το s_i να είναι το μικρότερο στοιχείο στο οποίο τελειώνει κάποια αύξουσα ακολουθία μήκους l και L_{max}, το μέγιστο μήκος υποακολουθίας που έχει βρεθεί (αρχικοποιούμε με 1).

- Για κάθε i από 1 μέχρι N:
  - Ψάχνουμε το $\max_{l \in \{0,\dots,L_{max}\} | s_{low[l]} < s_i} \{l\}$, με δυαδική αναζήτηση στα $\{s_{low[1]}, s_{low[2]}, \dots\}$.
  - Το μήκος της μεγαλύτερης υποακολουθίας που τελειώνει στο s_i, $LIS[i]$ θα είναι $l + 1$ και θέτουμε $low(l + 1) = i$.
  - Προσαυξάνουμε κατάλληλα το L_{max} (και φυσικά θυμόμαστε για κάθε $LIS[i]$ το προηγούμενο στοιχείο στην ακολουθία).

Longest increasing subsequences: Υπολογισμός με γνωστό αλγόριθμο σε $O(NlogN)$:
Έστω πίνακας $low[]$ όπου $low[l] = i$ ο δείκτης για τον οποίο το s_i να είναι το μικρότερο στοιχείο στο οποίο τελειώνει κάποια αύξουσα ακολουθία μήκους l και L_{max}, το μέγιστο μήκος υποακολουθίας που έχει βρεθεί (αρχικοποιούμε με 1).

- Για κάθε i από 1 μέχρι N:
  - Ψάχνουμε το $\max_{l \in \{0,\ldots,L_{max}\} | s_{low[l]} < s_i} \{l\}$, με δυαδική αναζήτηση στα $\{s_{low[1]}, s_{low[2]}, \ldots\}$.
  - Το μήκος της μεγαλύτερης υποακολουθίας που τελειώνει στο s_i, $LIS[i]$ θα είναι $l+1$ και θέτουμε $low(l+1) = i$.
  - Προσαυξάνουμε κατάλληλα το L_{max} (και φυσικά θυμόμαστε για κάθε $LIS[i]$ το προηγούμενο στοιχείο στην ακολουθία).

Longest increasing subsequences: Υπολογισμός με γνωστό αλγόριθμο σε $O(NlogN)$:
Έστω πίνακας $low[]$ όπου $low[l] = i$ ο δείκτης για τον οποίο το s_i να είναι το μικρότερο στοιχείο στο οποίο τελειώνει κάποια αύξουσα ακολουθία μήκους l και L_{max}, το μέγιστο μήκος υποακολουθίας που έχει βρεθεί (αρχικοποιούμε με 1).

- Για κάθε i από 1 μέχρι N:
  - Ψάχνουμε το $\max_{l \in \{0,\ldots,L_{max}\} | s_{low[l]} < s_i} \{l\}$, με δυαδική αναζήτηση στα $\{s_{low[1]}, s_{low[2]}, \ldots\}$.
  - Το μήκος της μεγαλύτερης υποακολουθίας που τελειώνει στο s_i, $LIS[i]$ θα είναι $l + 1$ και θέτουμε $low(l + 1) = i$.
  - Προσαυξάνουμε κατάλληλα το L_{max} (και φυσικά θυμόμαστε για κάθε $LIS[i]$ το προηγούμενο στοιχείο στην ακολουθία).

Longest increasing subsequences: Υπολογισμός με γνωστό αλγόριθμο σε $O(NlogN)$:
Έστω πίνακας $low[]$ όπου $low[l] = i$ ο δείκτης για τον οποίο το s_i να είναι το μικρότερο στοιχείο στο οποίο τελειώνει κάποια αύξουσα ακολουθία μήκους l και L_{max}, το μέγιστο μήκος υποακολουθίας που έχει βρεθεί (αρχικοποιούμε με 1).

- Για κάθε i από 1 μέχρι N:
  - Ψάχνουμε το $\max_{l \in \{0,\ldots,L_{max}\} | s_{low[l]} < s_i} \{l\}$, με δυαδική αναζήτηση στα $\{s_{low[1]}, s_{low[2]}, \ldots\}$.
  - Το μήκος της μεγαλύτερης υποακολουθίας που τελειώνει στο s_i, $LIS[i]$ θα είναι $l + 1$ και θέτουμε $low(l + 1) = i$.
  - Προσαυξάνουμε κατάλληλα το L_{max} (και φυσικά θυμόμαστε για κάθε $LIS[i]$ το προηγούμενο στοιχείο στην ακολουθία).

Πολυπλοκότητα:

- Αντίστοιχα βρίσκουμε το διάνυσμα $LDT[j]$ (longest decreasing subsequences).
- Επιστρέφουμε το $max_{k \in \{1,2,\dots,N\}}\{LIT[k] + LDT[k]\}$.

Πολυπλοκότητα

$$O(NlogN) + O(NlogN) + O(N) = O(NlogN)$$

Πολυπλοκότητα:

- Αντίστοιχα βρίσκουμε το διάνυσμα $LDT[j]$ (longest decreasing subsequences).
- Επιστρέφουμε το $max_{k \in \{1,2,...,N\}}\{LIT[k] + LDT[k]\}$.

Πολυπλοκότητα

$$O(NlogN) + O(NlogN) + O(N) = O(NlogN)$$

Πολυπλοκότητα:

- Αντίστοιχα βρίσκουμε το διάνυσμα $LDT[j]$ (longest decreasing subsequences).
- Επιστρέφουμε το $max_{k \in \{1,2,\ldots,N\}}\{LIT[k] + LDT[k]\}$.

Πολυπλοκότητα

$$O(NlogN) + O(NlogN) + O(N) = O(NlogN)$$

Πολυπλοκότητα:

- Αντίστοιχα βρίσκουμε το διάνυσμα $LDT[j]$ (longest decreasing subsequences).
- Επιστρέφουμε το $max_{k \in \{1,2,\ldots,N\}}\{LIT[k] + LDT[k]\}$.

Πολυπλοκότητα

$$O(NlogN) + O(NlogN) + O(N) = O(NlogN)$$