Φροντιστηριακό Μάθημα Εργαστηριακής Εξέτασης progintro

Ακ. Έτος 2017-2018

Μέρος Α1
Πρακτικές Συμβουλές Πάνω στις Πολυπλοκότητες των υποβολών

Πολυπλοκότητες Υποβολών

Ο grader έχει τη δυνατότητα να εκτελεί περίπου **10^6 πράξεις το δευτερόλεπτο.** Αυτό σημαίνει ότι "*συνήθως*" θα υπάρχουν 3 πιθανές λύσεις για τα προβλήματα

1. Κακή (Naive) = Δοκίμασε τα πάντα και δώσε μια απάντηση (π.χ. δοκίμασε όλες της μεταθέσεις ενός πίνακα με n στοιχεία. Δίνει πολυπλοκότητα O(n!).)
2. Μέτρια = Προϋπολόγισε κάποια σημεία του προβλήματος (π.χ. αθροίσματα κτλ.) και τρέξε κάτι καλύτερο από το naive. Πολυπλοκότητα Õ(n^d) - χειρότερη της βέλτιστης λύσης. Συνήθως πολύ εύκολη υλοποίηση.
3. Βέλτιστη = Περνάει όλα τα testcases του grader (φανερά και κρυφά). Είναι κομψή και μικρή, ωστόσο απαιτεί **ΣΚΕΨΗ**! Είναι συνήθως O(n) ή O(n log n) ανάλογα με τους περιορισμούς του προβλήματος.

Χρήσιμο άρθρο για μελέτη: https://discrete.gr/complexity/?el

Παραδείγματα

Έχουμε ένα πρόβλημα με όριο N <= 10^6 (π.χ. το sumint που θα λύσουμε σήμερα). Ένας αλγόριθμος με χρόνο εκτέλεσης O(N) θα μας έδινε όλες τις μονάδες, ενώ ένας αλγόριθμος O(N^2) μέρος αυτών. Ένας αλγόριθμος με χρόνο O(N^3) δεν θα μας έδινε και πολλές μονάδες.

Μπορεί πολλές φορές τα όρια να μας “αποκαλύψουν” τι αλγόριθμο χρειαζόμαστε για να λύσουμε το πρόβλημα.

Μέρος Α2
Μερικές χρήσιμες Τεχνικές

Long Long Integers

Σε πολλά προβλήματα το μέγεθος του INT δεν αρκεί για να χωρέσει το αποτέλεσμα. (π.χ. σε 64-bit αρχιτεκτονική ο μέγιστος signed int είναι ο 2^15 - 1). Για αυτό **πρέπει να χρησιμοποιούμε (unsigned) long (long) int!** Αλλιώς κινδυνεύουμε να χάσουμε testcases λόγω υπερχείλισης! Συμβαίνει συχνά!

Hint (για να μη γράφουμε πολλές φορές):

typedef long long llint; // type definition

llint *A;

Prefix Sums

Τα Prefix Sums μας επιτρέπουν σε χρόνο O(1) να έχουμε το άθροισμα των στοιχείων από A[i] ως A[j]. Προϋπολογίζονται εύκολα καθώς διαβάζουμε το array.

Παράδειγμα:

```
int N, x; cin >> N; llint S[N+1]; S[0] = 0;

for (int i = 1; i <= N; i++) {

    cin >> x; S[i] = S[i-1] + x;

}

cout << “Sum from A[i] to A[j] is” << S[j] - S[i-1] << endl;
```

STL (standard library της C++)

Η C++ (στην οποία συνίσταται να γράφετε την ώρα της εξέτασης) είναι μια γλώσσα με μια θαυμάσια βιβλιοθήκη με πάρα πολλά πράγματα. Συνίσταται να έχετε (ενδεικτικά) γνώση κάποιων βιβλιοθηκών της STL (όπως π.χ. η **algorithm** έχει διαθέσιμους αλγορίθμους για sorting, binary search και άλλα καλούδια).

Reference: STL Library

Μέρος Β
Λύνοντας Προβλήματα

sumint

Δίνονται μια ακολουθία $a_1,...,a_N$ αποτελούμενη από Ν θετικούς ακέραιους και ένας θετικός ακέραιος Κ. Ζητείται να υπολογίσουμε το **πλήθος** των διαστημάτων $a_i,...,a_j$ της ακολουθίας, με 1<=i<=j<=N, για τα οποία **το άθροισμα** των όρων **δεν ξεπερνά το Κ**, δηλ. έχουμε ότι Σ {p=i}^j a_p <= Κ.

Περιορισμοί:
2 <= N <= 2.000.000
1 <= a[i] <= 1.000.000
1 <= K <= 50.000.000

Πιθανές Λύσεις

Naive:

Για όλα τα i και όλα τα j:

Για κάθε διάστημα από a[i] ως a[j] υπολόγισε το άθροισμα a[i] + a[i+1] + … + a[j]

Αν είναι <= K αύξησε το μετρητή

Πολυπλοκότητα: O(n^3) (ΚΑΚΗ)

Βελτίωση Naive

Υπολόγισε τα Prefix Sums όλου του πίνακα

Για όλα τα i και όλα τα j:

Αν είναι S[j] - S[i-1] <= K αύξησε το μετρητή

Βελτίωση: Δεν υπολογίζουμε συνέχεια τα αθροίσματα

Πολυπλοκότητα: O(n^2) - ΚΑΛΥΤΕΡΗ

Γραμμική Λύση (Βέλτιστη)

1. Κρατάμε τα Prefix Sums σε ένα array S
2. Κρατάμε δύο δείκτες l, r στην αρχή του array και κινούμαστε προς τα δεξιά
3. Όσο το άθροισμα από l ως r το οποίο ισούται με S[r] - S[l-1] είναι <= K, αύξησε τον δείκτη r
4. Το πλήθος των διαστημάτων που περικλείονται είναι r - l. Είναι εύκολο να αποδείξουμε ότι **δεν διπλομετράμε! (Απόδειξη)**

Πολυπλοκότητα: Ο l διατρέχει όλο τον S και ο r είναι πάντα στο διάστημα l <= r <= N. Επομένως η χρονική πολυπλοκότητα είναι T(N) = O(N)

Γιατί ένας αλγόριθμος που θα έβρισκε τα ξένα μεταξύ τους διαδοχικά διαστήματα με άθροισμα <= K θα έδινε λάθος λύση;

elev2

Ένας ανελκυστήρας χωράει το πολύ δύο άτομα μέγιστου βάρους Β κιλών (και οι δύο μαζί). Στο ισόγειο, περιμένουν Ν άτομα να χρησιμοποιήσουν τον ανελκυστήρα για να ανέβουν στον τελευταίο όροφο. Ευτυχώς, γνωρίζουμε τα βάρη W[i] όλων τους. Γράψτε ένα πρόγραμμα που να διαβάζει αυτά τα δεδομένα και να **βρίσκει το ελάχιστο πλήθος διαδρομών** που πρέπει να κάνει ο ανελκυστήρας, για να μεταφερθούν όλα τα άτομα.

Περιορισμοί:
1 <= N <= 1.000.000
1 <= B <= 1.000.000
1 <= W[i] <= B

Λύση με διάταξη των ατόμων - O (N logN)

1. Ταξινόμησε τα W[i] σε χρόνο O(N log N) (π.χ. με quicksort)
2. Βάλε έναν δείκτη l να δείχνει το πρώτο στοιχείο και έναν r να δείχνει το τελευταίο. Αn W[l] + W[r] <= B αύξησε τον l και το πλήθος των μεταφορών.

```
int l = 0, count = 0; // W is sorted

for (int r = N - 1; l <= r; r--) {

        if (l < r && W[l] + W[r] <= B) l++;

        count++;

}
```

Λύση με στοιχεία σε πίνακα - O(B + N)

Λογική ίδια με την προηγούμενη λύση. Η διαφορά είναι ότι **κρατάμε τα βάρη των ατόμων σε ενα νέο int[] πίνακα μεγέθους B+1**. Η υπόλοιπη διαδικασία είναι **ακριβώς η ίδια** με τη μόνη διαφορά ότι πλέον θα αναφερόμαστε μόνο σε έγκυρα (μη μηδενικά) στοιχεία του πίνακα (και ότι τα for τα κάνουμε while). Οι δείκτες l και r πλέον δείχνουν στο ελάχιστο και το μέγιστο στοιχείο αντίστοιχα. Πολυπλοκότητα: O(B) + O(N) = O(B + N) = O(max{N, B}). [FYI: Counting Sort]

```
int W[B+1] = {0}, x;
int l = INT_MAX, r = INT_MIN;
for (int i = 0; i < N; i++) {
        cin >> x;
        W[x]++;
        l = min(x, l);
        r = max(x, r);
}
```

```
while (l <= r) {
        if (l + r <= B) {
                W[l]--;
                while (W[l] == 0) l++;
        }
        count++; W[r]--;
        while (W[r] == 0) r--;
}
```

icecream

Κατά μήκος μιας ευθείας οδού κατοικούν N παιδιά. Το κάθε παιδί i κατοικεί σε μια διαφορετική ακέραια συντεταγμένη a[i] επί της οδού. Πλησιάζει το καλοκαίρι, και ο παγωτατζής της γειτονιάς σκέφτεται σε ποιο σημείο της οδού θα εγκαταστήσει το κιόσκι του. Ο παγωτατζής γνωρίζει ότι κάθε παιδί είναι διατεθειμένο να **διανύσει απόσταση μικρότερη ή ίση του K** για να αγοράσει παγωτό. Έτσι θέλει να εγκαταστήσει το κιόσκι του στο σημείο όπου θα μπορεί να εξυπηρετήσει όσο το δυνατόν περισσότερα παιδιά. Να γράψετε ένα πρόγραμμα που υπολογίζει το **μέγιστο πλήθος παιδιών** που μπορεί να εξυπηρετήσει ο παγωτατζής.

Περιορισμοί:
1 <= N <= 1.000.000
1 <= K <= 5.000.000
0 <= a[i] <= 40.000.000

Βέλτιστη Λύση

1. Θεωρούμε δείκτες l, r που δείχνουν αντίστοιχα στο αριστερότερο a[l] και το δεξιότερο παιδί a[r] που μπορεί να εξυπηρετήσει ο παγωτατζής. Για να ανήκουν και τα δύο στο “πεδίο” του παγωτατζή πρέπει η απόστασή τους να είναι το πολύ η “διάμετρος”: a[r] - a[l] <= 2*K. Διατρέχοντας τον πίνακα a διακρίνουμε τις εξής περιπτώσεις:
   a. Αν αυτό ισχύει μπορούμε να προσπαθήσουμε να μεγαλώσουμε το διάστημα προς τα δεξιά αυξάνοντας τον r. Έτσι συμπεριλαμβάνουμε το r-οστό παιδί και πάμε στο επόμενο. (αύξηση counter)
   b. Αν δεν ισχύει μικραίνουμε το διάστημά μας προχωρώντας τον l δείκτη και πλέον το l-οστό παιδί δεν εξυπηρετείται. (μείωση counter)
2. Στο τέλος της επανάληψης ελέγχουμε αν η λύση μας βελτιώθηκε σε σχέση με την προηγούμενη. Πολυπλοκότητα: O(N)

Βαρεθήκατε να λύνετε προβλήματα (!);

Μπορείτε να δοκιμάσετε τις ικανότητες σας και σε άλλες παρόμοιες πλατφόρμες με προβλήματα! Ενδεικτικά μερικές

- Codeforces
- Codechef
- Hackerrank
- CS Academy
- SPOJ
- Project Euler

και πολλές άλλες (google them)!

Ευχαριστούμε πολύ!
Q&A

kameranis@gmail.com
panagiotis.kostopanagiotis@gmail.com
papachristoumarios@gmail.com
Source Code: https://goo.gl/cY3SSt