Prime Factorization
(Ανάλυση σε γινόμενο πρώτων παραγόντων)

Πρώτοι αριθμοί ονομάζονται οι θετικοί ακέραιοι των οποίων οι μόνοι θετικοί διαιρέτες είναι το 1 και ο εαυτός τους. Το 1 εξ ορισμού δεν είναι πρώτος.

Θεμελιώδες Θεώρημα της Αριθμητικής:

Κάθε φυσικός αριθμός μεγαλύτερος του 1 μπορεί να αναλυθεί κατά μοναδικό τρόπο σε γινόμενο πρώτων παραγόντων, αν δεν λάβουμε υπόψη μας την σειρά των παραγόντων του.

Δηλαδή για κάθε $n \in \mathbb{Z}^+, n > 1$ ισχύει πως:

$$n = {p_1}^{a_1} \cdot {p_2}^{a_2} \cdot \ldots \cdot {p_k}^{a_k}$$

Όπου $p_1, p_2, \ldots, p_k$ διαφορετικοί πρώτοι αριθμοί ταξινομημένοι σε αύξουσα σειρά και $a_1, a_2, \ldots, a_k$ θετικοί ακέραιοι.

Παραδείγματα:

$$12 = 2^2 \cdot 3$$

$$42 = 2 \cdot 3 \cdot 7$$

$$23 = 23\ (είναι\ πρώτος)$$

$$40158 = 2 \cdot 3^2 \cdot 23 \cdot 97$$

Χρήσεις:

Οι χρήσεις της ανάλυσης σε γινόμενο πρώτων παραγόντων ενός αριθμού είναι πολλές και μπορούν να βρεθούν σε πολλά προβλήματα και σε διάφορες μορφές

- Πλήθος των διαιρετών ενός αριθμού
- Άθροισμα των διαιρετών ενός αριθμού
- Εύρεση ελάχιστου κοινού πολλαπλασίου και μέγιστου κοινού διαιρέτη
- Έλεγχος διαιρετότητας μεγάλων αριθμών

Στην συνέχεια θα εξετάσουμε κάποιες από αυτές τις χρήσεις

Αλγόριθμος οικοδόμησης της ανάλυσης αριθμού

Μια πρώτη σκέψη είναι ο διαδοχικός έλεγχος των πρώτων αριθμών. Αν ένας πρώτος διαιρεί τον αρχικό μας αριθμό τότε ανήκει στην ανάλυση του αριθμού. Περισσότερες λεπτομέρειες στην πιο κάτω υλοποίηση. Ολική πολυπλοκότητα του αλγορίθμου είναι

$O(\frac{N}{\log N})$. Ο λόγος είναι επειδή στην χειρότερη περίπτωση πρέπει να περάσουμε από όλους τους πρώτους αριθμούς, των οποίων το πλήθος είναι της τάξης του $\frac{N}{\log N}$. Ο καλύτερος τρόπος για να βρούμε τους πρώτους αριθμούς είναι το κόσκινο του Ερατοσθένη, του οποίου την πολυπλοκότητα δεν προσθέσαμε σε αυτό τον αλγόριθμο.

```
typedef pair < int, int > ii; /// shortcut για την χρήση ενός ζεύγους
vector < int > primes;    /// πρώτοι αριθμοί από το κόσκινο

vector < ii > factorization(int x) {      /// result[i].first: pi (πρώτος)
    vector < ii > result;                 /// result[i].second: ai (εκθέτης)

    for (int i=0; i<primes.size(); i++) {
        if (primes[i] > x) break;  /// ξεπεράσαμε το x, δεν θα βρούμε άλλους πρώτους

        if (x%primes[i] == 0) { /// υπάρχει στην ανάλυση
            result.push_back(ii(primes[i], 0));  /// τον προσθέτουμε
            while (x%primes[i] == 0) {    /// και βρίσκουμε τον εκθέτη
                result.back().second ++;
                x /= primes[i];
            }
        }
    }

    return result;
}
```

Μπορούμε ωστόσο και καλύτερα. Το κύριο πρόβλημα αυτής της μεθόδου είναι το ότι εξετάζει πολλούς πρώτους οι οποίοι δεν διαιρούν τον αριθμό μας. Ένας τρόπος για να αποφύγουμε τους τόσους «άστοχους» ελέγχους, είναι να τροποποιήσουμε ελαφρώς το κόσκινο του Ερατοσθένη.

Δηλαδή, αντί να αποθηκεύουμε για ένα αριθμό μόνο το ότι είναι πρώτος ή σύνθετος, μπορούμε να αποθηκεύουμε και τον μεγαλύτερο πρώτο που τον διαιρεί (ή τον εαυτό του). Έτσι, μπορούμε να βρούμε αναδρομικά τους πρώτους παράγοντες του αριθμού απευθείας.

Καθώς ο κάθε πρώτος είναι μεγαλύτερος ή ίσος του 2, η συνολική χρονική πολυπλοκότητα του αλγορίθμου μας είναι $O(N \log \log N)$ αρχικά για το κόσκινο και $O(\log N)$ για κάθε αριθμό που θέλουμε να αναλύσουμε σε γινόμενο πρώτων παραγόντων. Το N είναι ο μέγιστος αριθμός που θα εξετάσουμε.

Ακολουθεί ο κώδικας για το τροποποιημένο κόσκινο:

```
bitset < MAXN > bs;
int max_prime[MAXN];

void sieveEratosthenes() {
    bs.set();
    bs[0] = 0;
    bs[1] = 0;

    for (int i=2; i<MAXN; i++) if (bs[i]) {  /// αν το i είναι πρώτος
        max_prime[i] = i;   /// ο μέγιστος πρώτος είναι ο εαυτός του
        for (int j=2*i; j<MAXN; j += i) {/// για όλα τα πολλαπλάσια του i
            bs[j] = 0;    /// δεν είναι πρώτος
            max_prime[j] = i;    /// ο μέγιστος (μέχρι τώρα) πρώτος
        }                        /// είναι το i
    }
}
```

Ο κώδικας για την ανάλυση ενός αριθμού:

```
typedef pair < int, int > ii;
int max_prime[MAXN];

vector < ii > factorization (int x) {
    vector < ii > result;  /// η σημασία είναι όπως πριν

    while (x > 1) {  /// όταν το x γίνει ίσο με 1
                     /// δεν υπάρχουν άλλοι παράγοντες
        result.push_back(ii(max_prime[x], 0));
        int current_prime = max_prime[x];/// ο επόμενος
        while (x%current_prime == 0) {  /// πρώτος
            result.back().second ++; /// για να βρούμε τον
            x /= current_prime;     /// εκθέτη στην ανάλυση
        }
    }

    return result;
}
```

Χρήσεις της ανάλυσης σε γινόμενο πρώτων παραγόντων:

1. Πλήθος διαιρετών ενός αριθμού

Αν $n = {p_1}^{a_1} \cdot {p_2}^{a_2} \cdot \ldots \cdot {p_k}^{a_k}$, τότε $d = (a_1 + 1)(a_2 + 1) \cdot \ldots \cdot (a_k + 1)$, όπου d το πλήθος των διαιρετών ενός αριθμού

2. Άθροισμα διαιρετών ενός αριθμού

Χρησιμοποιώντας το πιο πάνω, έστω s το άθροισμα των διαιρετών του n. Τότε:

$$s = (1 + p_1 + {p_1}^2 + \cdots + {p_1}^{a_1}) \cdot \ldots \cdot (1 + p_k + {p_k}^2 + \cdots + {p_k}^{a_k})$$

$$s = \prod_{i=1}^{k} (1 + p_i + {p_i}^2 + \cdots + {p_i}^{a_i})$$

$$s = \prod_{i=1}^{k} \frac{{p_i}^{a_i+1} - 1}{p_i - 1}$$

3. Έστω ότι έχουμε δύο αριθμούς x_1 και x_2 για τους οποίους ισχύει πως:

$$x_1 = {p_1}^{a_1} \cdot {p_2}^{a_2} \cdot \ldots \cdot {p_k}^{a_k}$$
$$x_2 = {p_1}^{b_1} \cdot {p_2}^{b_2} \cdot \ldots \cdot {p_k}^{b_k}$$

Τότε το Ελάχιστο Κοινό τους Πολλαπλάσιο (Ε.Κ.Π.) είναι ίσο με:

$${p_1}^{\max(a_1,b_1)} \cdot {p_2}^{\max(a_2,b_2)} \cdot \ldots \cdot {p_k}^{\max(a_k,b_k)}$$

Και ο Μέγιστος Κοινός Διαιρέτης (Μ.Κ.Δ.):

$${p_1}^{\min(a_1,b_1)} \cdot {p_2}^{\min(a_2,b_2)} \cdot \ldots \cdot {p_k}^{\min(a_k,b_k)}$$

4. Διαιρετότητα μεγάλων αριθμών (παράδειγμα):

Μας δίνεται ένας αριθμός $N = x!$, όπου $x! = 1 * 2 * 3 * \ldots * x$ και ένας αριθμός M, όπου $x, M \leq 10^6$. Πρέπει να ελέγξουμε αν $M \mid N$, αν δηλαδή ο Μ διαιρεί τον Ν.

Με τα όρια που μας έχουν δοθεί, ο Ν είναι αστρονομικά μεγάλος αριθμός (εκατομμύρια ψηφία). Οπότε δεν μπορούμε να ελέγξουμε άμεσα αν ισχύει η πάνω διαιρετότητα.

Μια καλή ιδέα, είναι να αναλύσουμε και τους δύο αριθμούς (Ν, Μ) σε γινόμενο πρώτων αριθμών. Ο Μ γίνεται απευθείας, ενώ με τον Ν θα προσθέτονται διαδοχικά εκθέτες στην ήδη προϋπάρχουσα ανάλυση, για τους αριθμούς $1..10^6$.

Έτσι, για να διαιρεί ο Μ τον Ν, πρέπει κάθε παράγοντα που έχει ο Μ, να τον περιλαμβάνει και ο Ν και μάλιστα ο εκθέτης του Μ να είναι μικρότερος ή ίσος του εκθέτη του Ν.