



Problem Tutorial: “Non-trivial Energy of Crystals”

Subtask 1

Note that in this subtask $n \leq 10$, which means we can enumerate all possible distributions of crystals among the chambers. Each crystal has 2 options for which chamber it can end up in; the choices are made independently, so there are a total of 2^n distributions. We will use the standard technique—mask enumeration. Each bit of the mask indicates which chamber a particular crystal goes into: if the i -th position of the mask is 0, then the i -th crystal goes to the first chamber; otherwise, it goes to the second.

Next, for each distribution, we will check that no pair of crystals within the same chamber sums to a prime number. This can be done by enumerating all pairs of elements in each chamber and checking their sum s for primality (by dividing by all numbers from 2 to $\lfloor \sqrt{s} \rfloor$). Pseudocode:

```
1 for mask = 0 .. (1<<n)-1
2   split crystals into two chambers by mask
3   ok = true
4   for each chamber
5     for all pairs (i, j) inside it
6       if isPrime(a[i] + a[j]) then ok = false
7   if ok then output this distribution
8 output -1 -1
```

The final asymptotic complexity is:

$$\mathcal{O}(2^n \cdot n^2 \cdot \sqrt{A}),$$

where A is the maximum possible sum of two crystal energies.

Subtasks 2 and 3

In these subtasks, $n \leq 200$, so the naive enumeration of distributions is no longer suitable; therefore, we will translate the problem into the language of graphs. We will represent the crystals as vertices of a graph, and we will draw an edge between vertices i and j if and only if $a_i + a_j$ is a prime number.

Then, the only thing left to do is to color the resulting graph with 2 colors such that the ends of each edge are of different colors. If such a coloring exists, then all vertices of one color can be placed in one chamber (the sum of any two energies within one part is not prime). The coloring can be found using a series of depth-first or breadth-first searches.

The final asymptotic complexity is:

$$\mathcal{O}(n^2 \cdot \sqrt{A}),$$

since all pairs (i, j) are considered, and for each pair, we check for primality $s = a_i + a_j$ as in subtask 1, while the series of searches runs in $\mathcal{O}(n^2)$.

In subtasks 1 and 2, for checking the primality of sums s , it was sufficiently efficient to enumerate all numbers from 2 to s .

Subtasks 4 and 5

Here $n \leq 3000$ and $A \leq 6 \cdot 10^6$, so we need to speed up the primality check. We can build a Sieve of Eratosthenes and precompute for each number from 1 to A whether it is prime. Then, the primality check of a sum can be performed in $\mathcal{O}(1)$, and we only need to enumerate all pairs.

The final asymptotic complexity is:

$$\mathcal{O}(A \log \log A + n^2).$$



This solution is definitely suitable for subtask 4. If we reduce memory consumption and implement the sieve more efficiently, it may also work for subtask 5.

Subtask 6 and complete solution

For the complete solution, it is sufficient to note that we can place all crystals with even energy in one chamber and those with odd energy in another. Indeed, the sum of any two even numbers is even and at least 4, hence not prime. The sum of any two odd numbers is also even and at least 6, so it is also not prime. Therefore, within each chamber, there will be no pairs of crystals that violate the condition.

The final asymptotic complexity is:

$$\mathcal{O}(n).$$

Solutions that cause overflow of 32-bit data types pass the tests of group 6.

Problem Tutorial: “Selling Apartments”

Subtasks

- $k = 1$;
- $k = t$;
- All a_i are equal;
- All b_i are equal.

When $k = 1$, we can sell apartments without restrictions, so we sell them starting from the most expensive ones.

When $k = t$, at most one apartment can be sold from each district. From each district, we choose the most expensive apartment, and only among these do we greedily choose the answer.

When all b_i are equal, all apartments belong to the same district, so we sell them starting from the most expensive one, placing the sales on days $1, k + 1, 2k + 1, \dots, xk + 1$.

When all a_i are equal, all apartments have the same price, which means we need to sell the maximum possible number of apartments.

Complete solution

- In each district, no more than $\left\lceil \frac{T}{k} \right\rceil$ apartments can be sold;
- Moreover, exactly $\left\lfloor \frac{T}{k} \right\rfloor + 1$ apartments can be sold only in $T \% k$ districts, while in the others — at most $\left\lfloor \frac{T}{k} \right\rfloor$;
- We consider apartments in descending order of price and greedily include an apartment in the answer if we can (if in the corresponding district it is still possible to sell one more apartment).
- It is claimed that the answer cannot be larger.
- Indeed, suppose the same limit $\left\lfloor \frac{T}{k} \right\rfloor$ is imposed on every district. Then we get a problem of merging several sorted arrays, which is solved greedily (we keep only the first $\left\lfloor \frac{T}{k} \right\rfloor$ apartments in each



district). Now suppose that in $T \% k$ districts we may additionally take the $\left\lfloor \frac{T}{k} \right\rfloor + 1$ -st apartment.

Then let us create one more array in which we store the list of all $\left\lfloor \frac{T}{k} \right\rfloor + 1$ -st apartments, and limit its size by $T \% k$. It is clear that the optimal set of apartments is obtained by the greedy algorithm. It remains to prove that a valid arrangement exists.

- It is claimed that for the obtained set of apartments, there exists an arrangement that does not violate the rule.
- We construct the arrangement as follows. Order the districts by the number of apartments in descending order. If the number of selected apartments is less than the number of days, add virtual apartments with zero price, each from its own individual district. Now we place on the leftmost free day an apartment from the district with the largest number of apartments, and for the next $k - 1$ days we do not consider this district. Then on every day from the 1st to the $(T - k + 1)$ -st, at least k different districts will be available for choice, while from at most $k - 1$ of them we cannot take an apartment right now.
- An alternative method — first distribute the apartments into blocks of k consecutive days so that no block contains two apartments from the same district. Then process the blocks from left to right and place apartments as follows: all apartments from districts that appeared in the previous block are placed in the same positions, and the rest are placed arbitrarily. Note that there may be one smaller block — for the algorithm to work correctly, the arrangement must start with it.

Problem Tutorial: “Arseniy’s Problem”

Subtask 1

We will iterate through all n -digit numbers in increasing order. We will store in cnt_S the number of considered n -digit numbers with a digit sum exactly equal to S . If after considering the next n -digit number x , we have such a cnt_S that $cnt_S = k$, then the current x is the answer to the problem. If the iteration ends and such a cnt_S does not appear, then there is no answer.

The final asymptotic complexity:

$$\mathcal{O}(10^n \cdot n).$$

Subtasks 2–4

We will use the idea of dynamic programming. Let $dp_{l,t}$ be the number of strings of length l made up of digits with a digit sum t , where $0 \leq t \leq 9l$. Then:

$$dp_{l,t} = \begin{cases} 1, & \text{if } l = 1; \\ \sum_{d=\max(0, t-9(l-1))}^{\min(9, t)} dp_{l-1, t-d}, & \text{if } l > 1. \end{cases}$$

Note that if $dp_{l,t} \geq k$, it is not necessary to know the exact value of $dp_{l,t}$. To construct the answer, it is sufficient to know that $dp_{l,t} \geq k$.

Let $m(S)$ be the number of n -digit numbers with a digit sum S , where $0 \leq S \leq 9n$. Then:

$$m(S) = \begin{cases} 1, & \text{if } n = 1; \\ \sum_{D=\max(1, S-9(n-1))}^{\min(9, S)} dp_{n-1, S-D}, & \text{if } n > 1. \end{cases}$$



We will iterate through possible digit sums S . If $m(S) \geq k$, we will find x_S —the k -th smallest n -digit number with a digit sum S . Then the answer to the problem will be the minimum of the found x_S .

We will construct x_S digit by digit from left to right. Let the first pos digits be already chosen, and their sum be p . We try to place the digit d at position pos . After it, there should be $rem = n - pos - 1$ digits with a digit sum $need = S - (p + d)$. Let cnt be the number of ways to complete these rem digits. Then $cnt = dp_{rem, need}$ if $0 \leq need \leq 9 \cdot rem$. Otherwise, $cnt = 0$.

We will iterate through d in increasing order and maintain the current value of k . For the current d , we look at cnt . If $cnt < k$, then all numbers with this digit come before the sought number. We skip them entirely, replacing k with $k - cnt$, and try the next d . If $cnt \geq k$, then the sought k -th number starts with the chosen pos digits and the digit d . We fix d and move to the next position, increasing p by d .

Thus, the digit at each position is uniquely determined, and after n steps, we obtain the number x_S , which by construction is the k -th smallest n -digit number with a digit sum S . Pseudocode:

```

1 for pos = 0 .. n-1
2   for d = (1 if pos == 0 and n > 1 else 0) .. 9
3     rem = n - pos - 1
4     need = S - (p + d)
5     if 0 <= need <= 9 * rem
6       cnt = dp[rem][need]
7     else
8       cnt = 0
9
10    if cnt < k
11      k -= cnt
12    else
13      xS[pos] = d
14      p += d
15      break

```

Thus, dp is computed in $\mathcal{O}(n \cdot 9n) = \mathcal{O}(n^2)$, and iterating through all S and constructing the k -th number for each suitable sum occurs in $\mathcal{O}(9n \cdot n \cdot 10) = \mathcal{O}(n^2)$.

The final asymptotic complexity:

$$\mathcal{O}(n^2).$$

Solutions where x_S and the answer to the problem are stored in a 64-bit data type pass the tests of group 3. Solutions where k , x_S , and the answer to the problem are stored in a 32-bit data type pass the tests of group 2.

Subtasks 5–7 and complete solution

Consider n -digit numbers of the form $10^{n-1} + y$, where y is an m -digit number, with $1 < m < n$.

There are a total of $9 \cdot 10^{m-1}$ such numbers. The digit sum of a number of the form $10^{n-1} + y$ can take values from 2 to $9m + 1$, meaning there are $9m$ distinct values for this sum. Therefore, there will be such a digit sum that for numbers of this form it occurs at least

$$\left\lceil \frac{9 \cdot 10^{m-1}}{9m} \right\rceil = \left\lceil \frac{10^{m-1}}{m} \right\rceil$$

times. Thus, if the condition

$$10^{m-1} \geq m \cdot k,$$

is satisfied, then the answer to the problem takes the form $10^{n-1} + y$. We just need to find y .

Since $k \leq 10^{18}$, for a complete solution it is sufficient to consider m up to $\min(21, n)$, take the minimum m for which the answer exists (if there is none, then there is no answer), and then y can be found using



the solution to subtask 4. We will output $10^{n-1} + y$ if $m < n$, otherwise just y . If $m < n$, then the decimal representation of such a number starts with one, followed by $n - m - 1$ zeros, and then the m digits of the number y .

We also need to remember to consider n -digit numbers of the form $10^{n-1} + y$, where y is a digit, meaning $m = 1$. A number of this form is the answer only when $k = 1$. In this case, $y = 0$, meaning the answer is 10^{n-1} .

Since m is a constant, y is found in $\mathcal{O}(m^2) = \mathcal{O}(1)$, and the answer is output in $\mathcal{O}(n)$.

The final asymptotic complexity:

$$\mathcal{O}(n).$$

Solutions where y is stored in a 64-bit data type pass the tests of group 7. Solutions where k and y are stored in a 32-bit data type pass the tests of group 6. Solutions where brute force is used to find y instead of dynamic programming pass the tests of group 5.

Problem Tutorial: “Elephant Filimon and the Digit Three”

Short statement

You are given a number n . In one operation, you may add to it a number of the form 3, 33, 333, 3333, ..., that is, a number consisting only of digit 3. A number is called beautiful if all its digits do not exceed 3, i.e. they belong to the set $\{0, 1, 2, 3\}$. You need to find the minimum number of operations after which the number becomes beautiful.

Subtask discussion

Subtask 1

For small constraints, we can simply simulate column addition and use dynamic programming over digits.

Let $dp[i][add][carry]$ denote whether it is possible to process the lowest i digits if:

- add is the number of operations that still affect the current digit,
- $carry$ is the carry from the previous digit.

If the current digit is s_i , then after all additions we get:

$$s_i + 3 \cdot add + carry.$$

We need the last digit to be at most 3:

$$(s_i + 3 \cdot add + carry) \bmod 10 \leq 3.$$

The new carry is:

$$\left\lfloor \frac{s_i + 3 \cdot add + carry}{10} \right\rfloor.$$

After that, we can choose how many operations continue further.

The complexity of this solution is about

$$\mathcal{O}(\text{len}^4(n)).$$



Subtasks 2–4

We can optimize the DP by storing not a boolean value, but the minimum number of operations:

$$dp[i][carry] = \text{minimum number of operations.}$$

This reduces the number of dimensions and gives a solution of complexity $O(\text{len}^3(n))$.

Subtask 5 (answer ≤ 2)

If it is known that the answer does not exceed 2, we can simply check:

- whether it is possible in 0 operations,
- in 1 operation,
- in 2 operations.

This can be done by trying all numbers of the form 3, 33, ...

Subtasks 6–8

We can fix the number of operations a and check whether it is possible to make the number beautiful.

This already leads to the main idea of the full solution.

Full solution

Step 1. Multiply everything by 3

If a number is beautiful (all its digits are ≤ 3), then after multiplying it by 3, all its digits will belong to the set $\{0, 3, 6, 9\}$.

One operation of adding a number of the form $33 \dots 3$ becomes, after multiplication by 3, adding a number of the form $99 \dots 9 = 10^k - 1$.

So the problem is equivalent to the following:

Starting from the number $3n$, obtain a number consisting only of digits $\{0, 3, 6, 9\}$ using the minimum number of operations, where in one operation you may add a number of the form $10^k - 1$.

Step 2. Decompose the operation

If we perform a operations, then in total we add

$$(10^{k_1} - 1) + \dots + (10^{k_a} - 1) = (10^{k_1} + \dots + 10^{k_a}) - a.$$

Therefore, after a operations we get the number $3n - a + \sum 10^{k_i}$.

So the problem reduces to the following:

Consider the number $m = 3n - a$. Is it possible, using a additions of powers of ten, to obtain a number consisting only of digits $\{0, 3, 6, 9\}$?



Step 3. What adding 10^k does

Adding 10^k increases one digit by 1, and carries turn some 9's into 0's. This is acceptable, because both 0 and 9 are allowed.

Therefore, we may think of one operation as increasing some digit by 1.

Step 4. How much each digit must be increased

Suppose a digit equals d . Then the minimum number of increments needed to make it one of $\{0, 3, 6, 9\}$ is:

$$\text{need}(d) = (3 - d \bmod 3) \bmod 3.$$

Define:

$$\text{dist}(x) = \sum \text{need}(\text{digits of } x).$$

Lemma

The number of operations a is sufficient if and only if

$$\text{dist}(3n - a) \leq a.$$

Proof

Each addition of a power of ten increases one digit by 1, so to fix all digits we need at least dist operations.

If $\text{dist} \leq a$, we can first fix all digits, and apply the remaining operations to higher digits — carries will not spoil the allowed digits.

Therefore, the condition is necessary and sufficient.

Algorithm

1. Compute the number $3n$.
2. Compute $\text{dist}(3n)$.
3. For $a = 0, 1, 2, \dots$:
 - if $\text{dist}(3n - a) \leq a$, output a ;
 - otherwise decrease the number by 1 and update dist .

Problem Tutorial: “Shustrik and Boxes”

Note that, in order to compute the sum of products of quadruples, it is sufficient to know how to merge the information of two connected components when they are united.

Indeed, let A_1 be the sum of the numbers in the first component, A_2 be the sum of products of all pairs of numbers in the first component, A_3 be the sum of products of all triples of numbers, and A_4 be the sum of products of all quadruples of numbers. Similarly, let B_1, B_2, B_3, B_4 be the corresponding sums for the second component. Then, for the component C obtained by merging components A and B , these sums can be computed as follows:

$$C_1 = A_1 + B_1$$



$$C_2 = A_2 + B_2 + A_1 \cdot B_1$$

$$C_3 = A_3 + B_3 + A_1 \cdot B_2 + A_2 \cdot B_1$$

$$C_4 = A_4 + B_4 + A_1 \cdot B_3 + A_2 \cdot B_2 + A_3 \cdot B_1$$

We will iterate over the boxes and, for each set of edges, compute the required value.

To speed up the solution, it is enough to maintain a rollback *DSU* (disjoint set union) and use the idea of pouring (edges) from the largest sub-box. To do this, it is sufficient to run a *DFS* over the nested boxes, always visiting the smallest sub-boxes first and then rolling the *DSU* state back to what it was before processing that sub-box. After that, compute the answer in the largest sub-box and keep all edges nested inside it. Then add all edges from the remaining sub-boxes to the *DSU*. This yields an $n \log^2 n$ solution.

There also exists a solution in $n \log n \cdot \alpha(n)$, where $\alpha()$ is the asymptotic complexity of *DSU* with two heuristics applied (that is, without using rollbacks at all). How to implement such a solution is left as an exercise for the reader.