CODE FESTIVAL Qual C 解説

sugim48, wo01

2017 年 10 月 22 日

For International Readers: English editorial starts at page 10.

A: Can you get AC?

読み込んだ文字列 S に対して、S_i が 'A'、S_{i+1} が 'C' となる i が存在するかどうか判定すれば良いです (ここで、S_i は S の i 文字目を表します)。これは、ループを用いてすべての i を試すことで判定できます。

使いたい言語に高度な文字列処理ライブラリがある場合、それを使うのも良いでしょう。

以下に C++ によるコード例を示します。

```
#include<cstdio>
#include<cstring>

using namespace std;

char ch[6];

int main(){
    scanf("%s", ch);
    int N = strlen(ch);
    for(int i = 0; i + 1 < N; ++i){
        if(ch[i] == 'A' && ch[i + 1] == 'C'){
            printf("Yes\n");
            return 0;
        }
    }
    printf("No\n");
    return 0;
}
```

B: Similar Arrays

数列 b が数列 A に似ているとき、すべての i について b_i の値は $A_i - 1, A_i, A_i + 1$ のいずれかです。したがって、数列 A に似ている数列 b は 3^N 個あります。このうち、積 $b_1 b_2 \ldots b_N$ が偶数となるような b の個数を求める必要があります。

積 $b_1 b_2 \ldots b_N$ が偶数となる条件は、ある i が存在して b_i が偶数となることです。

「ある i が存在して」という形の条件を満たす数列の個数は求めにくいので、この条件を満たさない数列の個数を求めることを考えてみましょう。すなわち、すべての i について b_i が奇数となるような数列 b の個数を求めます。

このような数列 b は、各 i について、

- A_i が偶数のとき、$b_i = A_i - 1$ または $b_i = A_i + 1$
- A_i が奇数のとき、$b_i = A_i$

を満たします。

よって、A_i が偶数である i の個数を e とすると、このような数列 b は 2^e 個あります。

以上から、求めるべき値は $3^N - 2^e$ となります。e の値は $O(N)$ 時間で求まるので、全体の計算量は $O(N)$ となります。

この問題の場合は N の値が小さいので、A と似ている 3^N 個の数列それぞれについて条件をみたすかどうか調べる $O(3^N N)$ 時間のアルゴリズムで解くことも可能です。

以下に C++ によるコード例を示します。

```
#include<cstdio>

using namespace std;

int N;
int A[10];

int main(){
    scanf("%d", &N);
    int all = 1, bad = 1;
    for(int i = 0; i < N; ++i){
        scanf("%d", A + i);
        all *= 3;
        if(A[i] % 2 == 0) bad *= 2;
    }
    printf("%d\n", all - bad);
    return 0;
}
```

C : Inserting 'x'

s の先頭文字と末尾文字に注目します．s の先頭文字と末尾文字が同じ場合，s から先頭文字と末尾文字を取り除いても答えは変わりません．逆に，s の先頭文字と末尾文字が異なる場合，s の先頭または末尾へ `x` を追加することで，s の先頭文字と末尾文字を一致させる必要があります．ただし，s の先頭文字も末尾文字も `x` でない場合，s の先頭文字と末尾文字を一致させるのは不可能です．

以上の考察より，次の方法によって答えが求まります．

s が空でない間，次の処理を繰り返す:

- s の先頭文字と末尾文字が同じ場合 $\rightarrow$ s から先頭文字と末尾文字を取り除く
- s の先頭文字と末尾文字が異なる場合
  - s の先頭文字が `x` である場合 $\rightarrow$ s の末尾へ `x` を追加する
  - s の末尾文字が `x` である場合 $\rightarrow$ s の先頭へ `x` を追加する
  - s の先頭文字も末尾文字も `x` でない場合 $\rightarrow$ 答えは `-1` である

最終的に，`x` を追加した回数が答えである．

いくつかの言語では，長さ N の文字列に対して文字を追加 / 削除するのに $\mathrm{O}(N)$ 時間が掛かります．よって，以上の方法をそのまま実装すると，計算時間が $\mathrm{O}(|s|^2)$ となり，TLE してしまいます．そこで，s の先頭位置および末尾位置を指す変数 l および r を用意し，s を変更する代わりに l, r を増減させるようにします．例えば，「s の末尾へ `x` を追加する」→「s から先頭文字と末尾文字を取り除く」という一連の処理は，「l を 1 だけ増やす」という処理と等価です．すると，計算時間は $\mathrm{O}(|s|)$ となり，TLE しません．

D : Yet Another Palindrome Partitioning

この解説では，文字列は英小文字のみからなるものとします．文字列 s について，次の 2 つの条件は等価です．

- s の文字を並べ替えて回文が得られる．
- 英小文字 $c = \mathtt{a}, \mathtt{b}, \ldots, \mathtt{z}$ のうち，「s の中に c が奇数回現れる」ような c は高々 1 種類である．

ここで，文字列 s のハッシュ値 $h(s)$ を，次のように定義します．

- s の中に $\mathtt{a}$ が奇数回現れる $\Leftrightarrow$ $h(s)$ の下から第 1 ビットは 1 である
- s の中に $\mathtt{b}$ が奇数回現れる $\Leftrightarrow$ $h(s)$ の下から第 2 ビットは 1 である
- $\vdots$
- s の中に $\mathtt{z}$ が奇数回現れる $\Leftrightarrow$ $h(s)$ の下から第 26 ビットは 1 である

すると，文字列 s について，次の 2 つの条件は等価です．

- s の文字を並べ替えて回文が得られる．
- $h(s)$ は 2 のべき乗または 0 である．

さて，長さ N の入力文字列 s に対して，次で定義される数列 $(a_0, a_1, \ldots, a_N)$ を計算しておきます．

$$a_i := h(s[0, i))$$

a_0, a_1, ... の順に計算していくことで，この数列は $\mathrm{O}(N)$ 時間で計算できます．すると，各 $0 \leq l \leq r \leq N$ について，$h(s[l, r)) = a_l \oplus a_r$ が成り立ちます．ただし，$\oplus$ は bitwise XOR の記号です．よって，次の 2 つの条件は等価です．

- $s[l, r)$ の文字を並べ替えて回文が得られる．
- $a_l \oplus a_r$ は 2 のべき乗または 0 である．

以上の考察をもとに，答えを計算する方法を考えます．各 $0 \leq i \leq N$ について，$opt_i :=$ ($s[0, i)$ の分割数の最小値) と定義します．すると，opt_0, opt_1, ... の順に次のように計算できます．

- $opt_0 = 0$.
- 各 $1 \leq i \leq N$ について，$S_i := \{j \mid 0 \leq j < i$ かつ $a_j \oplus a_i$ は 2 のべき乗または 0 である $\}$ とすると，$opt_i = \min_{j \in S_i}\{opt_j\} + 1$.

最終的に，opt_N が答えです．しかし，この方法では，各 opt_i を計算するのに $\mathrm{O}(N)$ 時間が掛かるので，全体で $\mathrm{O}(N^2)$ 時間が掛かって TLE してしまいます．

以上の方法を高速化することを考えます．次のように配列 dp を定義します．

- 今，各 $0 \leq j < i$ について opt_j が計算済みであるとする．各 $0 \leq x < 2^{26}$ について，$T_x := \{j \mid 0 \leq j < i$ かつ $a_j = x\}$ とすると，$dp_x := \min_{j \in T_x}\{opt_j\}$ と定義する．ただし，T_x が空の場合，$dp_x := \infty$ と定義する．

すなわち，配列 dp とは，「今までに計算した数列 opt について，ハッシュ値ごとに最小値を記録したもの」です．新たに各 opt_i を計算したとき，dp の更新は $\mathrm{O}(1)$ 時間でできます．また，各 opt_i の計算は，次のように $\mathrm{O}(1)$ 時間でできます．

- $X := \{0, 2^0, 2^1, \ldots, 2^{25}\}$ とすると，$opt_i = \min_{x \in X}\{dp_{a_i \oplus x}\} + 1$.

この方法は全体で $\mathrm{O}(N)$ 時間であり，十分高速です．なお，dp は長さ 2^{26} の 32 ビット整数型の配列ですが，これは 256 MB なので MLE しません．

E: Cubes

まず、実際の問題を少し変更したものである、以下の問題を考えてみましょう。

> xyz- 空間に 1 辺の長さが 1 の立方体のブロックがしきつめられている。すなわち、($i < 0$ なども含む) すべての整数 i, j, k について、点 (i, j, k), $(i+1, j, k)$, ..., $(i+1, j+1, k+1)$ を頂点とするブロックが存在する。もとの問題と同じようにこれをブロック (i, j, k) と呼ぶ。
> いま、点 $(0, 0, 0)$ と点 (A, B, C) を結ぶ線分にそって針金を通した。ブロック (x, y, z) であって、ある針金が通ったブロック (x', y', z') が存在して両者の距離が D 以下となるものの個数を (適切な mod で) 求めよ。

この問題は以下のようにして解くことができます。

まず、条件を満たすブロックの集合 S を別の方法で表現することから始めましょう。

1 辺の長さが $2D+1$ の立方体 $\mathcal{C}$ を考え、この中心のブロックが (x, y, z) であることを、立方体 $\mathcal{C}$ が位置 (x, y, z) にあるということにします。

また、針金が通ったブロックを座標が小さい順に $(0, 0, 0) = (x_1, y_1, z_1)$, (x_2, y_2, z_2), ..., $(x_{A+B+C-2}, y_{A+B+C-2}, z_{A+B+C-2}) = (A-1, B-1, C-1)$ とします (針金が通ったブロックが $A+B+C-2$ 個あることの証明は省略します)。

このとき、集合 S は、立方体 $\mathcal{C}$ が位置 (x_1, y_1, z_1), (x_2, y_2, z_2), ..., $(x_{A+B+C-2}, y_{A+B+C-2}, z_{A+B+C-2})$ の順に移動するときに $\mathcal{C}$ が通った範囲に存在するブロックの集合と一致します。

このことから、集合 S を以下のように集合 S_1, S_2, ..., $S_{A+B+C-2}$ に分割することができます。

- S_1 を、立方体 $\mathcal{C}$ が (x_1, y_1, z_1) にあるときに $\mathcal{C}$ に含まれていたブロックの集合とします。
- S_i $(2 \leq i \leq A+B+C-2)$ を、立方体 $\mathcal{C}$ が位置 (x_i, y_i, z_i) に移動したときに $\mathcal{C}$ に含まれていたブロックであって、どの S_j $(j < i)$ にも含まれていないブロックの集合とします。

このとき、明らかに $|S_1| = (2D+1)^3$, $|S_i| = (2D+1)^2$ $(2 \leq i \leq A+B+C-2)$ が成り立ちます。

よって、$|S| = \sum_i |S_i| = (2D+1)^3 + (A+B+C-3)(2D+1)^2$ となります。

ではもとの問題に戻りましょう。

ブロックの集合 T を $0 \leq x < A$, $0 \leq y < B$, $0 \leq z < C$ を満たすブロック (x, y, z) の集合とします。このとき、求めるべきものは集合 $S \cap T = \bigcup_i (S_i \cap T)$ の大きさです。集合 $S \cap T$ を S' と、$S_i \cap T$ を S'_i と書くことにします。

前の問題と異なり、$|S'_i|$ はいろいろな値を取りうるので、$|S'|$ を簡単な式で書くことはできません。しかし、$|S'_i|$ と $|S'_{i+1}|$ が異なる値となる i $(2 \leq i < A+B+C-2)$ は $O(D)$ 個しかないことがわかります。

実際、このような i は以下の条件のうちの少なくとも 1 つを満たし、そのような i は $O(D)$ 個しかありません。

- $x_i < D$ かつ $x_i \neq x_{i+1}$
- $y_i < D$ かつ $y_i \neq y_{i+1}$
- $z_i < D$ かつ $z_i \neq z_{i+1}$

- $x_i \geq A-1-D$ かつ $x_i \neq x_{i+1}$
- $y_i \geq B-1-D$ かつ $y_i \neq y_{i+1}$
- $z_i \geq C-1-D$ かつ $z_i \neq z_{i+1}$

このような i を $i_1, i_2, \ldots, i_n$ とし、各 i_j について $(x_{i_j}, y_{i_j}, z_{i_j})$ を求めておきます (これは $O(D)$ または $O(D \log D)$ 時間で行うことができます)。すると、各 j について、$|S'_{i_j}| + \cdots + |S'_{i_{j+1}-1}|$ は $O(1)$ 時間で求められます。

以上から、求めるべき値 $|S'| = \sum_i |S'_i|$ は $O(D \log D)$ 時間で計算できます。

F : Three Gluttons

$n = \frac{N}{3}$ とします．

まず，順列 a, b, c すべてが定まっている状況を仮定し，考察を始めましょう．A が食べる寿司を順に $(a_{i_1}, \ldots, a_{i_n})$ $(1 \leq i_1 < \cdots < i_n \leq N)$ とします．同様に，B が食べる寿司を順に $(b_{j_1}, \ldots, b_{j_n})$ $(1 \leq j_1 < \cdots < j_n \leq N)$ とし，C が食べる寿司を順に $(c_{k_1}, \ldots, c_{k_n})$ $(1 \leq k_1 < \cdots < k_n \leq N)$ とします．このとき，A, B, C がそれぞれ指定の順に寿司を食べ切るための必要十分条件は，以下のようになります．

- $(a_{i_1}, \ldots, a_{i_n}, b_{j_1}, \ldots, b_{j_n}, c_{k_1}, \ldots, c_{k_n})$ はすべて相異なる．
- 各 $t \in \{1, \ldots, n\}$ について，$(a_1, a_2, \ldots, a_{i_t}, b_1, b_2, \ldots, b_{j_t}, c_1, c_2, \ldots, c_{k_t})$ 中に a_{i_t}, b_{j_t}, c_{k_t} はそれぞれちょうど 1 回ずつ現れる．

元の問題に戻り，順列 a, b のみが定まっている状況で，次の問題を考えましょう．

> A が食べる寿司を順に $(a_{i_1}, \ldots, a_{i_n})$ $(1 \leq i_1 < \cdots < i_n \leq N)$ とする．同様に，同様に，B が食べる寿司を順に $(b_{j_1}, \ldots, b_{j_n})$ $(1 \leq j_1 < \cdots < j_n \leq N)$ とする．さらに，C が食べる寿司を順に $(x_1, \ldots, x_n)$ とする．このとき，A, B, C がそれぞれ指定の順に寿司を食べ切るような順列 c は何通りか？

順列 c が 1 通り以上であるための必要十分条件 $(\star)$ は，以下のようになります．

- $(a_{i_1}, \ldots, a_{i_n}, b_{j_1}, \ldots, b_{j_n}, x_1, \ldots, x_n)$ はすべて相異なる．
- 各 $t \in \{1, \ldots, n\}$ について，$(a_1, a_2, \ldots, a_{i_t}, b_1, b_2, \ldots, b_{j_t})$ 中に a_{i_t}, b_{j_t} はそれぞれちょうど 1 回ずつ現れる．
- 各 $t \in \{1, \ldots, n\}$ について，$(a_1, a_2, \ldots, a_{i_t}, b_1, b_2, \ldots, b_{j_t})$ 中に x_t は現れない．

条件 $(\star)$ が成り立たない場合，順列 c は 0 通りです．逆に，条件 $(\star)$ が成り立つ場合，順列 c の通り数は N のみに依存する値 $g(N)$ であることが分かります．$g(N)$ は DP などを用いて簡単に求まります．順列 c の通り数が $(i_1, \ldots, i_n)$ および $(j_1, \ldots, j_n)$ および $(x_1, \ldots, x_n)$ に依存しないという点がポイントです．

以上より，次の問題の答えに $g(N)$ を掛けたものが，元の問題の答えとなります．

> $(i_1, \ldots, i_n)$ $(1 \leq i_1 < \cdots < i_n \leq N)$ と，$(j_1, \ldots, j_n)$ $(1 \leq j_1 < \cdots < j_n \leq N)$ と，$(x_1, \ldots, x_n)$ の組であって，条件 $(\star)$ を満たすものは何通りか？

$(i_1, \ldots, i_n)$ $(1 \leq i_1 < \cdots < i_n \leq N)$ および $(j_1, \ldots, j_n)$ $(1 \leq j_1 < \cdots < j_n \leq N)$ を固定します．これらが条件 $(\star)$ の 2 番目の条件を満たしているとき，条件 $(\star)$ の 1 番目および 3 番目の条件を満たすような $(x_1, \ldots, x_n)$ を数え上げることを考えます．$t = n, \ldots, 1$ の順に x_t の値を決めていくことにします．各 $t \in \{1, \ldots, n\}$ について，x_t に割り当てることが**できない**のは，以下の値です．

- 各 $t' \in \{t+1, \ldots, n\}$ について，$a_{i_{t'}}$ および $b_{j_{t'}}$ および $x_{t'}$．
- $\{a_1, a_2, \ldots, a_{i_t}\} \cup \{b_1, b_2, \ldots, b_{j_t}\}$ の元．

前者と後者は互いに素なので，x_t に割り当てることが**できる**値は $f(i_t, j_t, t)$ 通りです．ただし，$f(i, j, t) := N - 3(n-t) - |\{a_1, a_2, \ldots, a_i\} \cup \{b_1, b_2, \ldots, b_j\}|$ と定義します．よって，$(x_1, \ldots, x_n)$ は $\prod_{t \in \{1, \ldots, n\}} f(i_t, j_t, t)$

通りです．以上より，$(i_1, \ldots, i_n)$ $(1 \leq i_1 < \cdots < i_n \leq N)$ と，$(j_1, \ldots, j_n)$ $(1 \leq j_1 < \cdots < j_n \leq N)$ の組であって，条件 $(\star)$ の 2 番目の条件を満たすもの全通りについて，$\prod_{t \in \{1,\ldots,n\}} f(i_t, j_t, t)$ の総和を取ったものが求まればよいです．これは，累積和を用いた DP によって，$\mathrm{O}(N^3)$ 時間で計算できます．

CODE FESTIVAL 2017 Qual C Editorial

Problems by sugim48 and wo01

Oct 21, 2017

A: Can you get AC?

We are asked to determine if there exists i such that $S_i =$ 'A' and $S_{i+1} =$ 'C' for the given string S (S_i denotes the i-th character in S). This can be done by enumerating all i with a loop.

You can also use the string library in your language.

C++ code follows:

```cpp
#include <cstdio>
#include <cstring>

using namespace std;

char ch[6];

int main(){
    scanf("%s", ch);
    int N = strlen(ch);
    for(int i = 0; i + 1 < N; ++i){
        if(ch[i] == 'A' && ch[i + 1] == 'C'){
            printf("Yes\n");
            return 0;
        }
    }
    printf("No\n");
    return 0;
}
```

B: Similar Arrays

If an integer sequence b is similar to the sequence A, for each i, b_i is $A_i - 1$, A_i or $A_i + 1$. Thus, there are 3^N sequences b that are similar to A. Among them, we need to count the ones such that the product $b_1 b_2 \dots b_N$ is even.

The product $b_1 b_2 \dots b_N$ is even if and only if there exists i such that b_i is even.

Since it is not very easy to count the sequences that satisfies the condition in the form "there exists i such that...", instead we will count the ones that does NOT satisfy the condition. That is, we will count the sequences b such that for each i, b_i is odd.

Such a sequence b satisfies the following for each i:

- If A_i is even, $b_i = A_i - 1$ or $A_i + 1$.
- If A_i is odd, $b_i = A_i$.

Thus, there are 2^e such sequences b, where e is the number of i such that A_i is even.

Therefore, the answer is $3^N - 2^e$. Since e can be found in $O(N)$ time, the total time complexity is $O(N)$.

There is another solution. Since N is small, we can enumerate all 3^N sequences that are similar to A, and check if each of them satisfies the condition, resulting in an $O(3^N N)$-time solution.

C++ code follows:

```cpp
#include <cstdio>

using namespace std;

int N;
int A[10];

int main(){
    scanf("%d", &N);
    int all = 1, bad = 1;
    for(int i = 0; i < N; ++i){
        scanf("%d", A + i);
        all *= 3;
        if(A[i] % 2 == 0) bad *= 2;
    }
    printf("%d\n", all - bad);
    return 0;
}
```

C : Inserting 'x'

Let us look at the first and last characters in s. When these two characters are the same, the answer does not change if they are removed from s. On the other hand, if they are different, we need to add `x` to the beginning or end of s so that the first and last characters in s are the same. However, this is impossible if neither the first nor the last character in s is `x`.

From these observations, the answer can be found as follows:

While s is not empty repeat:

- If the first and last characters in s are the same: Remove the first and last characters from s.
- If the first and last characters in s are different:
  - If the first character in s is `x`: Append `x` to the end of s.
  - If the last character in s is `x`: Append `x` to the beginning of s.
  - If neither the first nor the last character in s is `x`: The answer is `-1`.

The answer is the number of times `x` is added.

In some languages, it takes $O(N)$ time to append or remove a character to a string of length N, so naive implementation of the above takes $O(|s|^2)$ time, which will result in TLE. In order to handle this, we will use two pointers l and r that point to the current first and last characters in s, and instead of directly changing s, we will increment l or decrement r. For example, the following sequence of two operations, "appending `x` to the end of s" and then "removing the first and last characters from s", is equivalent to "incrementing l by one". This improves the time complexity to $O(|s|)$, which avoids TLE.

D : Yet Another Palindrome Partitioning

In this editorial, all strings consists of lowercase English letters. For a string s, the following two conditions are equivalent:

- It is possible to permute the characters in s and obtain a palindrome.
- Among the lowercase English letters $c = \mathtt{a}, \mathtt{b}, \ldots, \mathtt{z}$, there is at most one letter c such that c occurs in s an odd number of times.

Here, we define the hash $h(s)$ of s as follows:

- $\mathtt{a}$ occurs in s an odd number of times $\Leftrightarrow$ The least significant bit of $h(s)$ is 1
- $\mathtt{b}$ occurs in s an odd number of times $\Leftrightarrow$ The second least significant bit of $h(s)$ is 1
- $\vdots$
- $\mathtt{z}$ occurs in s an odd number of times $\Leftrightarrow$ The 26-th least significant bit of $h(s)$ is 1

Then, the following two conditions are equivalent for a string s:

- It is possible to permute the characters in s and obtain a palindrome.
- $h(s)$ is either 0 or a power of 2.

We will also construct a sequence $(a_0, a_1, \ldots, a_N)$ defined as follows for the given string s of length N:

$$a_i := h(s[0, i))$$

This sequence can be found in $\mathrm{O}(N)$ time by finding a_0, a_1, ... in this order. Then, for each $0 \leq l \leq r \leq N$, $h(s[l, r)) = a_l \oplus a_r$ holds, where $\oplus$ stands for a bitwise XOR. Thus, the following two conditions are equivalent:

- It is possible to permute the characters in $s[l, r)$ and obtain a palindrome.
- $a_l \oplus a_r$ is 0 or a power of 2.

Based on these observations, we will consider how to calculate the answer. For each $0 \leq i \leq N$, let opt_i be (The fewest number of strings $s[0, i)$ needs to be partitioned into). Then, we can calculate opt_0, opt_1, ..., in order, as follows:

- $opt_0 = 0$.
- For each $1 \leq i \leq N$, let $S_i := \{j \mid 0 \leq j < i \text{ and } a_j \oplus a_i \text{ is 0 or a power of 2}\}$, and then $opt_i = \min_{j \in S_i}\{opt_j\} + 1$.

The answer can be obtained as opt_N. However, directly calculating each opt_i as above takes $\mathrm{O}(N)$ time, which results in the total time complexity of $\mathrm{O}(N^2)$ and getting TLE.

Let us improve this by defining an array dp as follows:

- Assume that for each $0 \leq j < i$, we have found opt_j. For each $0 \leq x < 2^{26}$, let us define dp_x as $\min_{j \in T_x}\{opt_j\}$, where $T_x := \{j \mid 0 \leq j < i \text{ and } a_j = x\}$. Here, $dp_x := \infty$ if T_x is empty.

In order words, the array dp is a memoization of the minimum values in opt calculated so far for each value of the hash. When a new opt_i is calculated, dp can be updated in O(1) time. Also, each opt_i can be found in O(1) time, as follows:

- $opt_i = \min_{x \in X}\{dp_{a_i \oplus x}\} + 1$, where $X := \{0, 2^0, 2^1, \ldots, 2^{25}\}$

The total time complexity of this method is O(N), which is fast enough. Also, dp does not cause MLE, since it is an array of 2^{26} 32-bit integers, which is 256 MB.

E: Cubes

We will first consider a modified version of the problem:

> The xyz-space is filled with a cubic blocks of side 1. That is, for each triple of (possibly negative) integer i, j, k, there exists a block that has vertices (i, j, k), $(i+1, j, k)$, $\ldots$, $(i+1, j+1, k+1)$. We will call this block (i, j, k), similarly to the original problem.
> We have passed a wire with a negligible thickness through a segment connecting the points $(0, 0, 0)$ and (A, B, C). How many blocks (x, y, z) satisfy the following condition: "there exists a block (x', y', z') such that the wire passes inside the block (x', y', z') and the distance between the blocks (x,y,z) and (x',y',z') is at most D", modulo some prime.

This can be solved as follows.

First, we will express the set of the blocks that satisfies the condition, D, in a different way.

Let us consider a cube $\mathcal{C}$ of side $2D + 1$. We will say that cube $\mathcal{C}$ is at the position (x, y, z) when the central block of C is (x, y, z).

Also, let the blocks penetrated by the wire be $(0, 0, 0) = (x_1, y_1, z_1)$, (x_2, y_2, z_2), $\ldots$, $(x_{A+B+C-2}, y_{A+B+C-2}, z_{A+B+C-2}) = (A-1, B-1, C-1)$, in increasing order of coordinates. (We will omit the proof of the fact that there are $A + B + C - 2$ blocks penetrated by the wire.)

Then, the set S coincides with the set of the blocks in the region passed by $\mathcal{C}$ when $\mathcal{C}$ travels from (x_1, y_1, z_1) to (x_2, y_2, z_2), $\ldots$, $(x_{A+B+C-2}, y_{A+B+C-2}, z_{A+B+C-2})$.

From this fact, we will divide the set S into sets S_1, S_2, $\ldots$, $S_{A+B+C-2}$ as follows:

- Let S_1 be the set of the blocks contained in $\mathcal{C}$ when $\mathcal{C}$ is at the position (x_1, y_1, z_1).
- Let S_i $(2 \leq i \leq A + B + C - 2)$ be the set of the blocks contained in $\mathcal{C}$ when $\mathcal{C}$ travels to the position (x_i, y_i, z_i), and not contained in any S_j $(j < i)$.

Then, $|S_1| = (2D + 1)^3$, $|S_i| = (2D + 1)^2$ $(2 \leq i \leq A + B + C - 2)$ clearly holds.

Thus, $|S| = \sum_i |S_i| = (2D + 1)^3 + (A + B + C - 3)(2D + 1)^2$.

Now, we will return to the original problem.

Let the set T be the set of blocks (x, y, z) such that $0 \leq x < A$, $0 \leq y < B$, $0 \leq z < C$. Then, what we seek is the size of the set $S \cap T = \bigcup_i (S_i \cap T)$. We will denote $S \cap T$ by S', and $S_i \cap T$ by S'_i.

This time, $|S'_i|$ can take various values, so $|S'|$ cannot be expressed by a simple formula. However, it can be seen that there are only $\mathrm{O}(D)$ values of i $(2 \leq i < A + B + C - 2)$ such that $|S'_i|$ and $|S'_{i+1}|$ are different.

In fact, such i satisfies at least one of the following conditions, and this is why there are only $\mathrm{O}(D)$ of them:

- $x_i < D$ and $x_i \neq x_{i+1}$
- $y_i < D$ and $y_i \neq y_{i+1}$
- $z_i < D$ and $z_i \neq z_{i+1}$

- $x_i \geq A - 1 - D$ and $x_i \neq x_{i+1}$
- $y_i \geq B - 1 - D$ and $y_i \neq y_{i+1}$
- $z_i \geq C - 1 - D$ and $z_i \neq z_{i+1}$

Let these i be $i_1, i_2, \ldots, i_n$, and for each i_j, we will find $(x_{i_j}, y_{i_j}, z_{i_j})$ beforehand (this can be done in $\mathrm{O}(D)$ or $\mathrm{O}(D \log D)$ time.) Then, for each j, $|S'_{i_j}| + \cdots + |S'_{i_{j+1}-1}|$ can be found in $\mathrm{O}(1)$ time.

Therefore, the value we seek, $|S'| = \sum_i |S'_i|$, can be computed in $\mathrm{O}(D \log D)$ time.

F : Three Gluttons

Let $n = \frac{N}{3}$.

First, we will start by assuming that the permutations a, b and c are all fixed. Let the sushi eaten by A be $(a_{i_1}, \ldots, a_{i_n})$ $(1 \leq i_1 < \cdots < i_n \leq N)$, in order. Similarly, let the sushi eaten by B be $(b_{j_1}, \ldots, b_{j_n})$ $(1 \leq j_1 < \cdots < j_n \leq N)$, and the sushi eaten by C be $(c_{k_1}, \ldots, c_{k_n})$ $(1 \leq k_1 < \cdots < k_n \leq N)$. Then, a necessary and sufficient condition for A, B and C to be able to eat all sushi in the specified order is as follows:

- $(a_{i_1}, \ldots, a_{i_n}, b_{j_1}, \ldots, b_{j_n}, c_{k_1}, \ldots, c_{k_n})$ are all different.
- For each $t \in \{1, \ldots, n\}$, each of a_{i_t}, b_{j_t} and c_{k_t} occurs exactly once in $(a_1, a_2, \ldots, a_{i_t}, b_1, b_2, \ldots, b_{j_t}, c_1, c_2, \ldots, c_{k_t})$.

Let us return to the original problem, and consider the following problem where only the permutations a and b are fixed:

> Let the sushi eaten by A be $(a_{i_1}, \ldots, a_{i_n})$ $(1 \leq i_1 < \cdots < i_n \leq N)$, in order. Similarly, let the sushi eaten by B be $(b_{j_1}, \ldots, b_{j_n})$ $(1 \leq j_1 < \cdots < j_n \leq N)$, and the sushi eaten by C be $(x_1, \ldots, x_n)$. Then, how many permutations c there are such that A, B and C can eat all sushi in specified orders?

A necessary and sufficient condition $(\star)$ for there to be at least one c is as follows:

- $(a_{i_1}, \ldots, a_{i_n}, b_{j_1}, \ldots, b_{j_n}, x_1, \ldots, x_n)$ are all different.
- For each $t \in \{1, \ldots, n\}$, each of a_{i_t} and b_{j_t} occurs exactly once in $(a_1, a_2, \ldots, a_{i_t}, b_1, b_2, \ldots, b_{j_t})$.
- For each $t \in \{1, \ldots, n\}$, x_t does not occur in $(a_1, a_2, \ldots, a_{i_t}, b_1, b_2, \ldots, b_{j_t})$.

If the condition $(\star)$ does not hold, there are no permutations c. If the condition $(\star)$ holds, it can be seen that the number of permutations c only depends on N, and can be expressed as $g(N)$. $g(N)$ can be easily found using DP or some other method. The essence is that the number of c does not depend on $(i_1, \ldots, i_n)$, $(j_1, \ldots, j_n)$ or $(x_1, \ldots, x_n)$.

Thus, the answer to the original problem is the answer of the following problem, multiplied by $g(N)$:

> How many triples $(i_1, \ldots, i_n)$ $(1 \leq i_1 < \cdots < i_n \leq N)$, $(j_1, \ldots, j_n)$ $(1 \leq j_1 < \cdots < j_n \leq N)$, $(x_1, \ldots, x_n)$ satisfy the condition $(\star)$?

We will fix $(i_1, \ldots, i_n)$ $(1 \leq i_1 < \cdots < i_n \leq N)$ and $(j_1, \ldots, j_n)$ $(1 \leq j_1 < \cdots < j_n \leq N)$. Let us count the triples $(x_1, \ldots, x_n)$ that satisfy the first and third conditions in $(\star)$ when the above two satisfy the second condition in $(\star)$. We will decide the values of x_t in the order of $t = n, \ldots, 1$. For each $t \in \{1, \ldots, n\}$, the following values CANNOT be assigned to x_t:

- $a_{i_{t'}}$, $b_{j_{t'}}$ and $x_{t'}$ for $t' \in \{t+1, \ldots, n\}$.
- The elements in $\{a_1, a_2, \ldots, a_{i_t}\} \cup \{b_1, b_2, \ldots, b_{j_t}\}$.

The above two are mutually exclusive, so the number of values that CAN be assigned to x_t is $f(i_t, j_t, t)$, where $f(i, j, t) := N - 3(n - t) - |\{a_1, a_2, \ldots, a_i\} \cup \{b_1, b_2, \ldots, b_j\}|$. Thus, the number of $(x_1, \ldots, x_n)$ is $\prod_{t \in \{1,\ldots,n\}} f(i_t, j_t, t)$. Therefore, we want to find the sum of $\prod_{t \in \{1,\ldots,n\}} f(i_t, j_t, t)$ over all pairs $(i_1, \ldots, i_n)$ $(1 \leq i_1 < \cdots < i_n \leq N)$ and $(j_1, \ldots, j_n)$ $(1 \leq j_1 < \cdots < j_n \leq N)$ that satisfy the second condition in $(\star)$. This can be computed in $\mathrm{O}(N^3)$ time by DP based on accumulated sums.