CODE FESTIVAL 2016 Qual A Editorial

English editorial starts on page 6.

A : CODEFESTIVAL 2016

いくつかの方法を C++ のコードと併せて紹介します.

s から部分文字列を 2 つ取り出し，空白文字で連結して出力する

```
string s; cin >> s;
string s1 = s.substr(0, 4);
string s2 = s.substr(4, 8);
cout << s1 << ' ' << s2 << endl;
```

s に空白文字を挿入し，出力する

```
string s; cin >> s;
s.insert(4, " ");
cout << s << endl;
```

s を一文字ずつ出力し，4 文字目の直後に空白文字を出力する

```
string s; cin >> s;
for (int i = 0; i < 12; i++) {
        cout << s[i];
        if (i == 3) cout << ' ';
}
cout << endl;
```

B : 仲良しうさぎ

まず思いつく方法は，$1 \leq i < j \leq N$ のペアを全探索し，$a_i = j$ かつ $a_j = i$ を満たすペアを数えるという方法です．しかし，この方法の時間計算量は $\mathrm{O}(N^2)$ であり，$N = 10^5$ では実行時間制

限に間に合いません．そのため，より速い方法を考えなければなりません．

実は，$1 \leq i < j \leq N$ のペアを全探索する必要はありません．うさぎ i と仲良しなペアになりうるうさぎは，うさぎ a_i だけです．そのため，$1 \leq i \leq N$ を全探索し，$a_{a_i} = i$ を満たす i を数えれば十分です．ただし，この方法では，それぞれの仲良しなペアを 2 回ずつ数えてしまっているので，最後に答えを 2 で割らなければなりません．この方法の時間計算量は $\mathrm{O}(N)$ であり，$N = 10^5$ でも実行時間制限に間に合います．

C：次のアルファベット

辞書順最小のものを求める典型的なテクニックとして，「先頭の要素から順に見ていき，今見ている要素をできるだけ小さくする，ということを貪欲に行う」という方法があります．この問題もこの方法で解けます．

今見ている文字を c として，c をできるだけ小さくすることを考えます．c を `a` へ変えるために，最少で t 回の操作が必要であるとします．現時点の残り操作回数 K と比べたとき，$K \geq t$ ならば，c は t 回の操作によって `a` へ変えるべきです．逆に $K < t$ ならば，c は操作を行わずそのままにしておくべきです．このような処理を，先頭の文字から末尾の文字まで順に行います．

最後に，余った操作回数 K を消費しなければなりません．これは，末尾の文字に K 回の操作を行えばよいです．ただし，愚直に K 回の操作を行うプログラムでは，$K = 10^9$ のときに実行時間制限に間に合いません．そのため，K を 26 で割った余りで置き換えた後，愚直に K 回の操作を行う，などの工夫が必要です．

なお，C++ では次のテクニックが有用です．

- `c - 'a'` で，文字 c が何番目のアルファベットか (0-indexed) を求められる．
- `(char)('a' + k)` で，k 番目のアルファベット (0-indexed) を求められる．

D：マス目と整数

次の条件をすべて満たす整数行列 $(b_{i,j})$ $(1 \leq i \leq R,\ 1 \leq j \leq C)$ が存在するか判定する問題です．

- 条件 1：各 $1 \leq k \leq N$ について，$b_{r_k,c_k} = a_k$ である．
- 条件 2：$b_{i,j}$ は 0 以上である．
- 条件 3：各 $1 \leq i \leq R-1,\ 1 \leq j \leq C-1$ について，$b_{i,j} + b_{i+1,j+1} = b_{i,j+1} + b_{i+1,j}$ である．

まず，条件 3 を考察します．条件 3 の式を変形すると，$b_{i,j+1} - b_{i,j} = b_{i+1,j+1} - b_{i+1,j}$ となります．この式は，「行 i と行 $i+1$ の間で，列 j と列 $j+1$ の相対的な値の関係が等しい」という意味です．この式が各 $1 \leq j \leq C-1$ について成り立つと，「行 i と行 $i+1$ の間で，列 $1, 2, \ldots,$

C の相対的な値の関係が等しい」という意味になります．さらに，この式が各 $1 \leq i \leq R-1$ について成り立つと，「すべての行の間で，列 $1, 2, \ldots, C$ の相対的な値の関係が等しい」という意味になります．以上の考察より，条件 3 は次の条件 3' と同値です．

- 条件 3'：ある整数列 $(x_1, x_2, \ldots, x_R)$ と $(y_1, y_2, \ldots, y_C)$ が存在して，$b_{i,j} = x_i + y_j$ である．

とりあえず，条件 1 と条件 3' を満たす $(b_{i,j})$ が存在するか判定する問題を考えてみましょう．この問題は次のように言い換えられます．

> 整数列 $(x_1, x_2, \ldots, x_R)$ と $(y_1, y_2, \ldots, y_C)$ であって，各 $1 \leq k \leq N$ について $x_{r_k} + y_{c_k} = a_k$ を満たすものは存在するか？

この問題を解くために，次のグラフを考えます．まず，上側に R 個の頂点を並べ，下側に C 個の頂点を並べます．次に，各 $1 \leq k \leq N$ について，上側の r_k 番目の頂点と下側の c_k 番目の頂点の間に重み a_k の辺を張ります．このグラフの各頂点に整数の重みを設定することで，各辺について両端点の重みの和が辺の重みに一致するようにできるか判定すればよいです．各連結成分について，ある頂点の重みを適当な値（例えば 0）に決めると，隣り合う頂点の重みが順次決まっていきます．連結成分の頂点の重みがすべて決まった後，各辺について矛盾がないかチェックすればよいです．これは，深さ優先探索 (DFS) などを用いると，時間計算量 $\mathrm{O}(R + C + N)$ でできます．

次に，条件 2 を考察します．実は，条件 2 は次の条件 2' と同値です．

- 条件 2'：整数列 $(x_1, x_2, \ldots, x_R)$ と $(y_1, y_2, \ldots, y_C)$ であって，すべての要素が 0 以上のものが存在する．

条件 2' $\rightarrow$ 条件 2 は明らかです．条件 2 $\rightarrow$ 条件 2' を示します．$(b_{i,j})$ がすべて 0 以上であるとして，ある整数列 (x_i) と (y_j) によって $b_{i,j} = x_i + y_j$ と表されるとします．このとき，負の要素が含まれるのは (x_i) か (y_j) の高々一方だけです．(x_i) に負の要素が含まれるとします．「(x_i) の各要素を $+1$ し，(y_j) の各要素を -1 する」という操作を行うと，$b_{i,j}$ を変えずに (x_i) と $(y_i j)$ を変えることができます．(x_i) に負の要素が含まれなくなるまで，この操作を繰り返したとします．すると，(x_i) の最小値は 0 なので，(y_j) の最小値は非負であることが分かります．以上より，条件 2 $\rightarrow$ 条件 2' が示せました．

条件 1 と条件 3' を満たす $(b_{i,j})$ が存在するとき，さらに条件 2' を満たす $(b_{i,j})$ が存在するか判定する問題を考えましょう．先述のグラフのある連結成分 C について，上側の頂点の重みの最小値を x_C とし，下側の頂点の重みの最小値を y_C とします．このとき，$x_C + y_C \geq 0$ ならば，先述の操作を繰り返すことで，連結成分の頂点の重みをすべて 0 以上にできます．逆に $x_C + y_C < 0$ ならば，どうしても連結成分の頂点に負の重みが含まれてしまいます．まとめると，すべての連結成分について $x_C + y_C \geq 0$ ならば答えは `Yes` となり，そうでなければ答えは `No` となります．

E : LRU パズル

まず，$N = 1$ の場合を観察してみます．配列 $(1, 2, ..., M)$ に操作列 a を適用して得られる配列を $f_M(a)$ と書きます．例えば，$M = 6$，$a = (6, 3, 1, 3, 6, 3)$ の場合，$f_M(a) = (3, 6, 1, 2, 4, 5)$ となります．$f_M(a)$ は次のように求められます．まず，空の配列 b を用意します．次に，a を末尾の要素から順に見ていき，今見ている要素 a_i が b に含まれていなければ，a_i を b の末尾へ追加します．最後に，1 から M までの整数のうち，b に含まれていないものを昇順に並べ，b の末尾へ追加します．最終的な b が $f_M(a)$ となります．

問題は次のように言い換えられます．

> 操作列 a を，N 個の部分列 $a_1, a_2, \ldots, a_N$ へ分離し，$f_M(a_1), f_M(a_2), \ldots, f_M(a_N)$ がすべて等しくなるようにできるか？

$f_M(a_1), f_M(a_2), \ldots, f_M(a_N)$ がすべて等しい場合，明らかに $f_M(a_1)$ の第 1 要素は $f_M(a)$ の第 1 要素と一致しなければなりません．同様に，$f_M(a_1)$ の第 2 要素は $f_M(a)$ の第 2 要素と一致しなければなりません．これを繰り返すことで，$f_M(a_1), f_M(a_2), \ldots, f_M(a_N)$ は $f_M(a)$ と一致しなければならないことが示せます．以上より，問題は次のように言い換えられます．

> 操作列 a を，N 個の部分列 $a_1, a_2, \ldots, a_N$ へ分離し，$f_M(a_1), f_M(a_2), \ldots, f_M(a_N)$ がすべて $f_M(a)$ と一致するようにできるか？

例えば，$N = 2, M = 6, a = (6, 3, 1, 3, 6, 3)$ の場合を考えます．この場合，$f_M(a) = (3, 6, 1, 2, 4, 5)$ なので，この配列を目標に a を部分列へ分離します．例えば，$a = (6, 3, 1, 3, 6, 3)$ と分離し，$a_1 = (6, 3, 3)$，$a_2 = (1, 6, 3)$ とすると，$f_M(a_1) = f_M(a_2) = f_M(a)$ となります．

とりあえず，言い換え後の問題を $N = 2$ の場合に解くことを考えます．まず，$f_M(a)$ を求めておきます．操作列 a を部分列 a_1 と a_2 へ分離した後，先述の $N = 1$ の場合と同様に b_1 と b_2 をそれぞれ求め，ともに $f_M(a)$ と一致させるのが目標です．このとき，a を a_1 と a_2 へ分離していきながら，逐次的に b_1 と b_2 を更新していくことにします．b_1 と b_2 は常に $f_M(a)$ の prefix であるという条件を満たしながら，b_1 と b_2 をできるだけ伸ばすことを考えると，次のような貪欲法が最適だということが分かります．まず，空の配列 b_1 と b_2 を用意します．a を末尾の要素から順に見ていき，今見ている要素 a_i が

- b_1 にも b_2 にも含まれない場合：a_i を b_1 の末尾へ追加する．
- b_1 に含まれ，b_2 に含まれない場合：a_i を b_2 の末尾へ追加しても prefix の条件が満たされるならば，追加する．
- b_1 にも b_2 にも含まれる場合：何もしない．

という操作を繰り返します．その後，先述の $N = 1$ の場合と同様に，b_1 と b_2 をそれぞれ長さ M

まで拡張し，$b_1 = b_2 = f_M(a)$ をチェックすればよいです．

一般の $N \geq 2$ の場合も，$N = 2$ の場合と同様に解くことができます．まず，$f_M(a)$ を求めておき，空の配列 $b_1, b_2, \ldots, b_N$ を用意します．操作列 a を末尾の要素から順に見ていき，今見ている要素 a_i を含まない配列 b_j のうち，b_j の末尾へ a_i を追加しても prefix の条件が満たされるものをひとつ選び，b_j の末尾へ a_i を追加する，という操作を繰り返します．その後，先述の $N = 1$ の場合と同様に，$b_1, b_2, \ldots, b_N$ をそれぞれ長さ M まで拡張し，$b_1 = b_2 = \cdots = b_N = f_M(a)$ をチェックすればよいです．この方法は正しいのですが，時間計算量を抑えるために実装を工夫しなければなりません．

例えば，次のように実装を工夫すればよいです．配列 $freq_0, freq_1, \ldots, freq_M$ を次のように定義します．

$$freq_k = (\text{配列 } b_1, b_2, \ldots, b_N \text{ のうち，長さがちょうど } k \text{ であるものの個数})$$

最初，$b_1, b_2, \ldots, b_N$ はすべて空なので，$freq_0 = N$，$freq_1 = freq_2 = \cdots = freq_M = 0$ です．操作列 a を末尾の要素から順に見ていき，今見ている要素を a_i とします．a_i が $f_M(a)$ のうち k 番目 (0-indexed) の位置にあるとします．このとき，a_i を末尾へ追加するべき配列 b_j は，長さがちょうど k のものです．よって，$freq_k$ を参照し，$freq_k \geq 1$ ならば，$freq_k$ を -1 して $freq_{k+1}$ を $+1$ します．逆に $freq_k = 0$ ならば，何もしません．この操作を繰り返すと，最終的な $b_1, b_2, \ldots, b_N$ の長さの分布が求まります．特に，最も短い b_j の長さが求まります．これまでの説明では，$b_1, b_2, \ldots, b_N$ をそれぞれ長さ M まで拡張し，$b_1 = b_2 = \cdots = b_N = f_M(a)$ をチェックしていましたが，実は最も短い b_j だけをチェックすれば十分であることが分かります．この実装の時間計算量は $\mathrm{O}(N + M + Q)$ となり，実行時間制限に間に合います．

CODE FESTIVAL 2016 Qual A Editorial (English)

A : CODEFESTIVAL 2016

We will introduce several possible approaches along with sample C++ codes:

Cut out two substrings from s, then concatenate them with a space

```
string s; cin >> s;
string s1 = s.substr(0, 4);
string s2 = s.substr(4, 8);
cout << s1 << ' ' << s2 << endl;
```

Directly insert a space into s using standard libraries

```
string s; cin >> s;
s.insert(4, " ");
cout << s << endl;
```

Print one letter of s at a time, and print a space just after the fourth letter

```
string s; cin >> s;
for (int i = 0; i < 12; i++) {
        cout << s[i];
        if (i == 3) cout << ' ';
}
cout << endl;
```

B : Friendly Rabbits

The first approach that comes to mind would be to enumerate all possible pairs of $1 \leq i < j \leq N$, then count the ones that satisfy $a_i = j$ and $a_j = i$. However, the time complexity of

this approach is $O(N^2)$, which will exceed the time limit when $N = 10^5$. A faster approach is required.

Actually, it is not necessary to enumerate all possible pairs of $1 \leq i < j \leq N$. The only rabbit that can form a friendly pair with rabbit i, is rabbit a_i. Therefore, it is enough to just enumerate all $1 \leq i \leq N$, then count the ones that satisfy $a_{a_i} = i$. Note that in this approach each friendly pair is counted twice, thus we have to divide the count by 2 to obtain the final answer. The time complexity of this approach is $O(N)$, which will be within the time limit even when $N = 10^5$.

C : Next Alphabet

When the problem asks for the lexicographically smallest sequence, the following greedy strategy typically works: focus on one element at a time in order from the beginning of the sequence, and each time make the current element as small as possible. This problem is no exception.

Suppose that we are focusing on a letter c in the string. Consider making this letter as small as possible. Assume that in order to change c into `a` the operation has to be performed at least t times. Compare t with the remaining number of operations, K. If $K \geq t$, then c should be changed into `a` by performing the operation t times. Otherwise ($K < t$), c should be left as it is. Repeat this procedure from the beginning of s to the end.

After the procedure is executed on all letters in s, the remaining number of operations, K, must be consumed. We can simply perform the operation K times on the last letter in the string. Note that programs that naively performs the operation K times will time out when $K = 10^9$. A simple solution would be to replace K with K modulo 26 before naively performing the operation.

When implementing the solution in C++, the following techniques could be useful:

- A letter c is the (`c - 'a'`)-th letter of the alphabet (0-indexed).
- The k-th letter of the alphabet (0-indexed) is `(char)('a' + k)`.

D : Grid and Integers

The problem asks to determine whether there exists an integer matrix $(b_{i,j})$ ($1 \leq i \leq R$, $1 \leq j \leq C$) that satisfies:

- Condition 1 : for each $1 \leq k \leq N$, $b_{r_k,c_k} = a_k$.
- Condition 2 : $b_{i,j}$ is non-negative.

- Condition 3 : for each $1 \leq i \leq R-1$, $1 \leq j \leq C-1$, $b_{i,j} + b_{i+1,j+1} = b_{i,j+1} + b_{i+1,j}$.

First, let us examine Condition 3. The formula can be transformed into: $b_{i,j+1} - b_{i,j} = b_{i+1,j+1} - b_{i+1,j}$. This can be interpreted as follows: in row i and row $i+1$, the relationships (differences) between the elements at column j and column $j+1$ are the same. This holds on all $1 \leq j \leq C-1$, thus, in row i and row $i+1$, the relationships among the elements at all columns are the same. Furthurmore, this holds on all $1 \leq i \leq R-1$, thus, in all rows, the relationships among the elements at all columns are the same. Therefore, Condition 3 is equivalent to the following:

- Condition 3' : There exists two integer sequences $(x_1, x_2, \ldots, x_R)$ and $(y_1, y_2, \ldots, y_C)$ such that $b_{i,j} = x_i + y_j$.

For now, let us determine the existence of $(b_{i,j})$ that satisfies Condition 1 and 3'. This problem can be rephrased as follows:

> Does there exist two integer sequences $(x_1, x_2, \ldots, x_R)$ and $(y_1, y_2, \ldots, y_C)$, such that for each $1 \leq k \leq N$, $x_{r_k} + y_{c_k} = a_k$?

Consider the following graph. The graph has $R+C$ vertices, R on the upper side and C on the lower side. For each $1 \leq k \leq N$, the r_k-th vertex on the upper side and the c_k-th vertex on the lower side is connected by an edge with weight a_k. The objective is to assign an integer weight to each vertex, so that for each edge the sum of the weight of the two incident vertices is equal to the weight of the edge. For each connected component, choose one vertice and assign an arbitrary weight (for example, 0), and the weights of the remaining vertices can be determined accordingly. After all vertices in the component are assigned a weight, check if there is no inconsistency on any edge. This can be done with the time complexity of $\mathrm{O}(R+C+N)$ using methods such as Depth First Search.

We will now examine Condition 2. Actually, it is equivalent to the following:

- Condition 2' : There exists two integer sequences $(x_1, x_2, \ldots, x_R)$ and $(y_1, y_2, \ldots, y_C)$ such that all elements are non-negative.

It is obvious that Condition 2' $\rightarrow$ Condition 2. We will show that Condition 2 $\rightarrow$ Condition 2'. Suppose that all $(b_{i,j})$ are non-negative, and they are represented as $b_{i,j} = x_i + y_j$ by some integer sequence (x_i) and (y_j). Then, at most one of (x_i) and (y_j) contains negative elements. Assume that (x_i) contains negative elements. We can modify (x_i) and (y_j) without affecting $b_{i,j}$ by performing the following operation: add 1 to all (x_i), and subtract 1 from all (y_j). Suppose that we repeatedly performed this operation until (x_i) does not contain a negative element. Then, the minimum element in (x_i) is 0, thus the minimum element in (y_j)

is non-negative. Therefore, it is shown that Condition 2 → Condition 2'.

Let us consider this problem: when there exists $(b_{i,j})$ that satisfies Condition 1 and 3', determine the existence of $(b_{i,j})$ that satisfies Condition 2' in addition. Recall the aforementioned graph. For a connected component C in this graph, let x_C be the minimum weight of the vertices on the upper side, and y_C be the minimum weight of the vertices on the lower side. Then, if $x_C + y_C \geq 0$, we can eliminate the negative weights of the vertices of the component by repeatedly performing an operation similar to the one used in the proof of Condition 2' → Condition 2. Otherwise ($x_C + y_C < 0$), we cannot completely eliminate the negative weights of the vertices of the component. Therefore, the final answer will be `Yes` if $x_C + y_C \geq 0$ for any connected component C. Otherwise, the answer will be `No`.

E : LRU Puzzle

First, observe the case where $N = 1$. Let $f_M(a)$ denote the array obtained by applying a sequence of operations a to the array $(1, 2, ..., M)$. For example, when $M = 6$ and $a = (6, 3, 1, 3, 6, 3)$, $f_M(a) = (3, 6, 1, 2, 4, 5)$. $f_M(a)$ can be found as follows. First, let b be an empty array. Then, we will examine each element of a in reverse order from the end of a and perform the following procedure. If the current element a_i is not contained in b, then append a_i to the end of b. After this procedure is executed on all elements of a, the integers from 1 to M that is not yet contained in b, are appended to the end of b in ascending order. The obtained array b is equal to $f_M(a)$.

The problem is rephrased as follows:

> Is it possible to separate the sequence of operations a into N subsequences a_1, a_2, ..., a_N, such that $f_M(a_1)$, $f_M(a_2)$, ..., $f_M(a_N)$ are all equal?

If $f_M(a_1)$, $f_M(a_2)$, ..., $f_M(a_N)$ are all equal, then the first element of $f_M(a_1)$ must be equal to the first element of $f_M(a)$. Similarly, the second element of $f_M(a_1)$ must be equal to the second element of $f_M(a)$. Repeating this argument, it can be shown that all of $f_M(a_1)$, $f_M(a_2)$, ..., $f_M(a_N)$ must be equal to $f_M(a)$, thus we can rephrase the problem again as follows:

> Is it possible to separate the sequence of operations a into N subsequences a_1, a_2, ..., a_N, such that $f_M(a_1)$, $f_M(a_2)$, ..., $f_M(a_N)$ are all equal to $f_M(a)$?

Let us consider the case where $N = 2$, $M = 6$ and $a = (6, 3, 1, 3, 6, 3)$. Here, the objective is to satisfy $f_M(a_1) = f_M(a_2) = f_M(a) = (3, 6, 1, 2, 4, 5)$. This can be achieved by, for example, separating $a = (6, 3, 1, 3, 6, 3)$ into $a_1 = (6, 3, 3)$ and $a_2 = (1, 6, 3)$.

For now, let us focus on the case where $N = 2$. First, find $f_M(a)$ using the method explained

at the beginning. Our objective is to separate a into a_1 and a_2 so that $f_M(a_1) = f_M(a_2) = f_M(a)$. Let b_1 and b_2 be the arrays that are used to evaluate $f_M(a_1)$ and $f_M(a_2)$ using the method, respectively. We will sequentially update b_1 and b_2 while separating a into a_1 and a_2. Considering that we want to extend b_1 and b_2 as much as possible under the condition that b_1 and b_2 must always be prefixes of $f_M(a)$, we can adopt the following greedy strategy. Examine each element of a in reverse order from the end of a, and execute the following procedure:

- When the current element a_i is contained in neither b_1 nor b_2, append a_i to the end of b_1.
- When a_i is contained in b_1, but not in b_2, if appending a_i to the end of b_2 does not violate the prefix condition, append a_i to the end of b_2.
- When a_i is contained in both b_1 and b_2, do nothing.

After this procedure is executed on all element of a, extend each of b_1 and b_2 to length M as in the method of finding $f_M(a)$, and check if $b_1 = b_2 = f_M(a)$.

The general case where $N \geq 2$ can be solved similarly. First, find $f_M(a)$ and let b_1, b_2, ..., b_N be empty arrays. Examine each element of a in reverse order from the end of a, and execute the following procedure: choose an array b_j which does not contain the current element a_i, such that appending a_i to the end of b_j does not violate the prefix condition, and append a_i to the end of b_j. After this procedure is executed on all element of a, extend each of b_1, b_2, ..., b_N to length M as in the method of finding $f_M(a)$, and check if $b_1 = b_2 = \cdots = b_N = f_M(a)$. The remaining problem is to implement this solution within the time limit.

One possible implementation is as follows. Let the arrays $freq_0$, $freq_1$, ..., $freq_M$ be defined as follows:

$$freq_k = (\text{the number of the arrays of length exactly } k \text{ among } b_1, b_2, \ldots, b_N)$$

At first, $freq_0 = N$, $freq_1 = freq_2 = \cdots = freq_M = 0$, since all of b_1, b_2, ..., b_N are empty. Suppose that we are executing the procedure on a_i. The arrays b_j that we should append a_i to, are the ones with the length exactly k. Thus, if $freq_k \geq 1$, subtract 1 from $freq_k$ and add 1 to $freq_{k+1}$. Otherwise, do nothing. By repeating this operation, we can find the final destribution of the lengths of b_1, b_2, ..., b_N, especially the length of the smallest b_j. So far, we extended each of b_1, b_2, ..., b_N to length M and checked if $b_1 = b_2 = \cdots = b_N = f_M(a)$, but actually it is enough to check for only the smallest b_j. The time complexity of this implementation is $O(N + M + Q)$, which will be within the time limit.