Permutazione genetica (`permutazione`)

Limite di tempo: 1.0 secondi
Limite di memoria: 256 MiB

Da quando ha cominciato a lavorare come ricercatore, Giorgio si diverte a fare esperimenti su cavie animali. In particolare si è deciso ad ottenere due topi superintelligenti, proprio come nel suo cartone preferito: per raggiungere questo obiettivo però, Giorgio deve incrementare esponenzialmente il potenziale genetico dei due topi da laboratorio a sua disposizione. Il nostro protagonista ritiene infatti che una semplice mutazione genetica non sia sufficiente a raggiungere l'obiettivo, e per questo motivo decide di produrre una *permutazione genetica*.

Per gli scopi di Giorgio, definiamo una *permutazione genetica* come una stringa lunga 26 caratteri che vanno dalla lettera `A` alla `Z`. In particolare, affinché la mutazione abbia effetto, è necessario che la sua *funzione espotenziale* valga esattamente K.

Per calcolare la funzione espotenziale di una permutazione genetica dobbiamo osservare le variazioni tra le 25 coppie di lettere consecutive (le prime due lettere, le seconde due lettere, e così via). Ad esempio, consideriamo la permutazione genetica banale:

`A, B, C, D, E, F, G, H, I, J, K, L, M, N, O, P, Q, R, S, T, U, V, W, X, Y, Z`

Le variazioni tra le coppie di lettere consecutive sono tutte di una sola lettera (infatti la distanza tra `A` e `B`, tra `B` e `C` e così via, è sempre pari a 1). Prendiamo l'insieme di tutte le variazioni, cancelliamo i duplicati, ed il numero di valori rimanenti sarà pari alla funzione espotenziale della permutazione!

Lo stesso vale per quest'altra permutazione, dato che la distanza tra `Z` e `Y` è ancora pari ad 1:

`Z, Y, X, W, V, U, T, S, R, Q, P, O, N, M, L, K, J, I, H, G, F, E, D, C, B, A`

Un esempio più significativo permette di capire meglio:

`P, L, T, A, V, H, F, W, D, Z, J, O, S, G, U, K, X, R, I, N, C, Y, E, Q, M, B`

In questo caso le distanze tra le lettere consecutive sono:

$$4, 8, 19, 21, 14, 2, 17, 19, 22, 16, 5, 4, 12, 14, 10, 13, 6, 9, 5, 11, 22, 20, 12, 4, 11$$

Eliminando i duplicati otteniamo:

$$2, 4, 5, 6, 8, 9, 10, 11, 12, 13, 14, 16, 17, 19, 20, 21, 22$$

Essendoci 17 valori diversi, la funzione espotenziale vale 17.

Scrivi un programma che, ricevuto K, produca una qualsiasi permutazione genetica che abbia funzione espotenziale pari a K.

Implementazione

Dovrai sottoporre esattamente un file con estensione `.c`, `.cpp` o `.pas`.

> ☞ Tra gli allegati a questo task troverai un template (`permutazione.c`, `permutazione.cpp`, `permutazione.pas`) con un esempio di implementazione da completare.

Se sceglierai di utilizzare il template, dovrai implementare la seguente funzione:

C/C++	`void permuta(int K, char* P[]);`
Pascal	`procedure permuta(K: longint; var P: array of char);`

In cui:

- L'intero K è il valore della funzione espotenziale richiesto.
- L'array di caratteri `P` andrà riempito dalla funzione con 26 caratteri (indicizzati da 0 a 25) che formano una permutazione genetica corretta.

Dati di input

Il file `input.txt` è composto da un'unica riga contenente l'unico intero K.

Dati di output

Il file `output.txt` è composto da un'unica riga contenente una stringa di 26 caratteri, la risposta a questo problema.

Assunzioni

- $1 \leq K \leq 25$.

Assegnazione del punteggio

Il tuo programma verrà testato su diversi test case raggruppati in subtask. Per ottenere il punteggio relativo ad un subtask, è necessario risolvere correttamente tutti i test relativi ad esso.

- **Subtask 1** [**10 punti**]: Casi d'esempio.
- **Subtask 2** [**20 punti**]: $K \leq 3$.
- **Subtask 3** [**30 punti**]: $K \geq 23$.
- **Subtask 4** [**40 punti**]: Nessuna limitazione specifica.

Esempi di input/output

input.txt	output.txt
1	ABCDEFGHIJKLMNOPQRSTUVWXYZ
1	ZYXWVUTSRQPONMLKJIHGFEDCBA
17	PLTAVHFWDZJOSGUKXRINCYEQMB