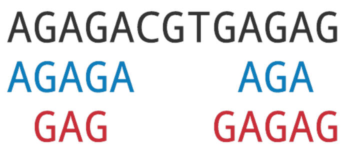
1B Find the Most Frequent Words in a String

Frequent Words Problem
Find the most frequent words k-mers in a string.

Input: A DNA string *Text* and an integer *k*.
Output: All most frequent *k*-mers in *Text* (in any order).



Formatting

Input: A DNA string *Text* followed by an integer *k*.
Output: All most frequent *k*-mers in *Text* (in any order).

Constraints

- The length of *Text* will be between 1 and 10^4.
- The integer *k* will be between 1 and 10^2.
- *Text* will be a DNA string.

Test Cases

Case 1

Description: The sample dataset is not actually run on your code.

Input:

```
ACGTTGCATGTCGCATGATGCATGAGAGCT
4
```

Output:

```
CATG GCAT
```

Case 2

Description: This dataset just checks if you're counting the first *k*-mer in *Text* (`TGG` in this example). If you do not count the first *k*-mer (`TGG`), you will get the following "most frequent" *k*-mers in addition to `TGG`: `ACT`, `CAC`, `CCA`, `CTT`, `GGT`.

Input:

```
TGGTAGCGACGTTGGTCCCGCCGCTTGAGAATCTGGATGAACATAAGCTCCCACTTGGCTTATTCAGAG...
...AACTGGTCAACACTTGTCTCTCCCAGCCAGGTCTGACCACCGGGCAACTTTTAGAGCACTATCGTG...
...GTACAAATAATGCTGCCAC
3
```

Output:

```
TGG
```

Case 3

Description: This dataset just checks if you're counting the last *k*-mer in *Text* (`TTTT` in this example). If you do not count the last *k*-mer (`TTTT`), you will get the following "most frequent" *k*-mers in addition to `TTTT`: `AACG`, `AATA`, `ACAA`, `CAAC`, `CTGG`, `CTGG`, `CTTT`, `TTGC`, `TTTG`.

Input:

```
CAGTGGCAGATGACATTTTGCTGGTCGACTGGTTACAACAACGCCTGGGGCTTTTGAGCAACGAGACTTT...
...TCAATGTTGCACCGTTTGCTGCATGATATTGAAAACAATATCACCAAATAAATAACGCCTTAGTAAG...
...TAGCTTTT
4
```

Output:

```
TTTT
```

Case 4

Description: This dataset checks if your code correctly handles cases where there are overlapping occurrences of *Pattern* throughout *Text*. For example, `AACAACAA` contains two occurrences of `AACAA` (**`AACAA`**`CAA` and `AAC`**`AACAA`**), so if your code counts `AACAACAA` as one occurrence of `AACAA`, your code will fail on this test case.

Input:

```
ATACAATTACAGTCTGGAACCGGATGAACTGGCCGCAGGTTAACAACAGAGTTGCCAGGCACTGCCGCTG...
...ACCAGCAACAACAACAATGACTTTGACGCGAAGGGGATGGCATGAGCGAACTGATCGTCAGCCGTCA...
...GCAACGAGTATTGTTGCTGACCCTTAACAATCCCGCCGCACGTAATGCGCTAACTAATGCCCTGCTG
5
```

Output:

```
AACAA
```

Case 5

Description: This test dataset checks if your code correctly handles ties (i.e. your code actually outputs ALL "most frequent" *k*-mers, and not just a single "most frequent" *k*-mer). For example, in the string `ATATA`, there are two "most frequent" *k*-mers: `AT` and `TA`. `AT` occurs twice (**`ATAT`**`A`), and `TA` occurs twice (`A`**`TATA`**), so both of these should be output (separated by a space character).

Input:

```
CCAGCGGGGGTTGATGCTCTGGGGGTCACAAGATTGCATTTTTATGGGGTTGCAAAAATGTTTTTTACGG...
...CAGATTCATTTAAAATGCCCACTGGCTGGAGACATAGCCCGGATGCGCGTCTTTTACAACGTATTGC...
...GGGGTAAAATCGTAGATGTTTTAAAATAGGCGTAAC
5
```

Output:

```
AAAAT GGGGT TTTTA
```

Case 6

Description: A larger dataset of the same size as that provided by the randomized autograder. Check input/output folders for this dataset.