2D Implement GreedyMotifSearch

Greedy Motif Search Problem
Implement GreedyMotifSearch.

Input: A collection of strings *Dna*, and integers *k* and *t*.
Output: A collection of strings resulting from running GreedyMotifSearch(*Dna*, *k*, *t*).

```
tACCTtaa
ATGTctgt
cgGCGTta
tcagAGGT
ctaACGAg
```

Formatting

Input: Space-separated integers *k* and *t*, followed by a newline-separated collection of strings *Dna*.
Output: A space-separated list of strings resulting from running GreedyMotifSearch(*Dna*, *k*, *t*) (If at any step you find more than one *Profile*-most probable *k*-mer in a given string, use the one occurring first).

Constraints

- The integer *k* will be between 1 and 10^2.
- The integer *t* will be between 1 and 10^2.
- The number of strings in *Dna* will be between 1 and 10^2.
- The length of each string in *Dna* will be between 1 and 10^2.
- Each string in *Dna* will be a DNA string.

Test Cases

Case 1

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

Input:
```
3 5
GGCGTTCAGGCA AAGAATCAGTCA CAAGGAGTTCGC CACGTCAATCAC CAATAATATTCG
```

Output:
```
CAG CAG CAA CAA CAA
```

Case 2

Description: This dataset checks that your code always picks the first-occurring Profile-most Probable *k*-mer in a given sequence of *Dna*. In the first sequence (`GCCCAA`), `GCC` and `CCA` are both Profile-most Probable *k*-mers. However, you must return `GCC` since it occurs earlier than `CCA`. Thus, if the first sequence of your output is `CCA`, this test case fails your code.

Input:
```
3 4
GCCCAA GGCCTG AACCTA TTCCTT
```

Output:
```
GCC GCC AAC TTC
```

Case 3

Description: This dataset checks if your code has an off-by-one error at the beginning of each sequence of *Dna*. Notice that the first four motifs of the solution occur at the beginning of their respective sequences in *Dna*, so if your code did not check the first *k*-mer in each sequence of *Dna*, it would not find these sequences.

Input:

```
5 8
GAGGCGCACATCATTATCGATAACGATTCGCCGCATTGCC
TCATCGAATCCGATAACTGACACCTGCTCTGGCACCGCTC
TCGGCGGTATAGCCAGAAAGCGTAGTGCCAATAATTTCCT
GAGTCGTGGTGAAGTGTGGGTTATGGGGAAAGGCAGACTG
GACGGCAACTACGGTTACAACGCAGCAACCGAAGAATATT
TCTGTTGTTGCTAACACCGTTAAAGGCGGCGACGGCAACT
AAGCGGCCAACGTAGGCGCGGCTTGGCATCTCGGTGTGTG
AATTGAAAGGCGCATCTTACTCTTTTCGCTTTCAAAAAAA
```

Output:

```
GAGGC TCATC TCGGC GAGTC GCAGC GCGGC GCGGC GCATC
```

Case 4

Description: This dataset checks if your code has an off-by-one error at the end of each sequence of *Dna*. Notice that the first two motifs of the solution occur at the end of their respective sequences in *Dna*, so if your code did not check the end *k*-mer in each sequence of *Dna*, it would not find these sequences.

Input:

```
6 5
GCAGGTTAATACCGCGGATCAGCTGAGAAACCGGAATGTGCGT
CCTGCATGCCCGGTTTGAGGAACATCAGCGAAGAACTGTGCGT
GCGCCAGTAACCCGTGCCAGTCAGGTTAATGGCAGTAACATTT
AACCCGTGCCAGTCAGGTTAATGGCAGTAACATTTATGCCTTC
ATGCCTTCCGCGCCAATTGTTCGTATCGTCGCCACTTCGAGTG
```

Output:

```
GTGCGT GTGCGT GCGCCA GTGCCA GCGCCA
```

Case 5

Description: This test dataset checks if your code is correctly breaking ties when calling Profile-most Probable *k*-mer. Specifically, it makes sure that, when you call Profile-most Probable *k*-mer, in the event of a tie, you choose the first-occurring *k*-mer.

Input:

```
5 8
GACCTACGGTTACAACGCAGCAACCGAAGAATATTGGCAA
TCATTATCGATAACGATTCGCCGGAGGCCATTGCCGCACA
GGAGTCTGGTGAAGTGTGGGTTATGGGGCAGACTGGGAAA
GAATCCGATAACTGACACCTGCTCTGGCACCGCTCTCATC
AAGCGCGTAGGCGCGGCTTGGCATCTCGGTGTGTGGCCAA
AATTGAAAGGCGCATCTTACTCTTTTCGCTTAAAATCAAA
GGTATAGCCAGAAAGCGTAGTTAATTTCGGCTCCTGCCAA
TCTGTTGTTGCTAACACCGTTAAAGGCGGCGACGGCAACT
```

Output:

```
GCAGC TCATT GGAGT TCATC GCATC GCATC GGTAT GCAAC
```

Case 6

Description: A larger dataset of the same size as that provided by the randomized autograder.