The contest is in progress. It ends about 24 hours from now.

Lenovo Problem

Lenovo is working hard to help scientists in biology specially in Bioinformatics to build computers that can handle big biological computations and handling huge set of information. In this problem you will need to help solve one of those tasks.

In Bioinformatics, the term "Sequence Alignment" is used to describe the task of arranging two or more biological sequences (DNA, RNA or Protein) in such a way that regions of similarity between the sequences can be easily identified and thus conclusions about the evolutionary relationships between them may be inferred. In order to make the conserved regions more apparent, usually each sequence is represented in a distinct row within a matrix, whereas gaps are also inserted between the sequences' residues so that identical or similar characters are aligned in successive columns [Source: Wikipedia: "Sequence alignment"]. An example of one possible alignment for the sequences Seq1 = ACGCATTCG, Seq2 = ACGAGTGG, Seq3 = CGATTAG is presented at Table 1:

Table 1. One possible alignment for the Sequences: Seq1 = ACGCATTCG, Seq2 = ACGAGTGG, Seq3 = CGATTAG (Gaps are denoted by a dash "-")

Seq 1:	ACGCATTCG
Seq 2:	ACG-AGTGG
Seq 3:	-CG-ATTCG

This alignment indicates that the first residue (A) of Seq 1 is alignment to the first residue (A) of Seq 2 and to nothing (gap denoted by a dash) to Seq 3. In a similar way, from the second column of the alignment it is made clear that the second residue of Seq 1 (C) is aligned to the second residue (C) of Seq 2 and to the first residue (C) of Seq 3.

Obviously, this is only one possible alignment for the aforementioned sequences and many more arrangements are possible. Could you help me create a program that can efficiently generate all possible unique (gapped) alignments for a given set of sequences?

Task

Your task is to develop a program that can efficiently generate all the possible unique alignments for a given set of sequences .

To better understand the task of the present problem, please consider the following two examples:

Example 1:

if the input sequences were: Seq 1 = AC and Seq 2 = GG, then the following 13 unique alignments are possible:

`Alignment :1`
`AC`
`GG`
	`Alignment :4`
	`A-C`
	`GG-`
		`Alignment :7`
		`-AC`
		`G-G`
			`Alignment :10`
			`A--C`
			`-GG-`
				`Alignment :13`
				`--AC`
				`GG--`

`Alignment :2`
`AC-`
`G-G`
	`Alignment :5`
	`A-C`
	`-GG`
		`Alignment :8`
		`AC--`
		`--GG`
			`Alignment :11`
			`-AC-`
			`G--G`	

`Alignment :3`
`AC-`
`-GG`
	`Alignment :6`
	`-AC`
	`GG-`
		`Alignment :9`
		`A-C-`
		`-G-G`
			`Alignment :12`
			`-A-C`
			`G-G-`	

Example 2: if the input sequences were: Seq 1 = ACT and Seq 2 = GC, then the following 25 unique alignments are possible:

Alignment :1	Alignment :6	Alignment :11	Alignment :16	Alignment :21
ACT	ACT-	A-CT	ACT--	A--CT
GC-	--GC	-GC-	---GC	-GC--
Alignment :2	Alignment :7	Alignment :12	Alignment :17	Alignment :22
ACT	AC-T	A-CT	AC-T-	-ACT-
G-C	G-C-	-G-C	--G-C	G---C
Alignment :3	Alignment :8	Alignment :13	Alignment :18	Alignment :23
ACT	AC-T	-ACT	AC--T	-AC-T
-GC	-GC-	GC--	--GC-	G--C-
Alignment :4	Alignment :9	Alignment :14	Alignment :19	Alignment :24
ACT-	AC-T	-ACT	A-CT-	-A-CT
G--C	--GC	G-C-	-G--C	G-C--
Alignment :5	Alignment :10	Alignment :15	Alignment :20	Alignment :25
ACT-	A-CT	-ACT	A-C-T	--ACT
-G-C	GC--	G--C	-G-C-	GC---

Since the number of possible alignments grows exponentially by the number of the sequences to be aligned and the number of residues forming each sequence, you can safely assume that only “toy examples” (i.e. only a few sequences comprised of a limited number of residues) will be considered.

As it may be inferred by the examples above: All the sequences in each alignment have the same length (number of columns) and whenever it is required gaps are inserted between the sequences’ residues If no gaps need to be inserted then they are skipped (like for example in the alignment 1 of Example 1). In other words, columns containing only gaps are considered irrelevant to the alignment task and should not be printed. In every alignment the order of the sequences is kept (e.g. in the examples above, always Seq 1 is in the first line of the output followed by Seq 2) Also, the order of the residues forming each sequence is always respected (i.e. if the sequence is ACT then always the A appears first then the C and finally the T) Regarding the order of the alignments in the final result set, the following rules have to be respected: The alignments should be sorted in increasing order according to the length (number of columns) of the alignment If two or more sequences have the same length, then these alignments should be sorted in alphabetical order according to the sequences in the alignment: Initially, the first sequence of the alignments will be used to decide on the order of the sequences If there still exist two or more alignments that have exactly the same length and also are identical in terms of the first sequence, then the alphabetical order of the second sequence of the alignments will be used to decide the ranking of these alignments. If there are still two or more alignments that have exactly the same length and also are identical in terms of the first and second sequence, the algorithm will continue likewise using the third, fourth etc sequence of these alignments. Please also refer to the provided examples where the alignments are listed in the proper order according to the above sorting rules. For instance, in Example 1, Alignment 1 is positioned first in the list of results since its length (2) is smaller than the length of any other alignment in the result set. Also, in compliance with the ordering rules, Alignment 2 of Example 1 is ordered before Alignment 4 since the first sequence of Alignment 2 (AC-) precedes alphabetically the corresponding first sequence of Alignment 4 (A-C). Moreover, the ordering between Alignment 2 and Alignment 3 (which both have length equal to 3 and sequence AC- as the first sequence) is decided by the alphabetical order of the second sequence in these alignments, where the second sequence of the Alignment 2 (G-G) precedes alphabetically the corresponding second sequence of Alignment 3 (-GG). In order to avoid issues from printing huge result sets, instead of outputting all the possible alignments for a given set of sequences, your program should output only the total number of possible alignments as well as the alignments at some predefined positions that would be given as input to the program.

Input Your program will receive the following input and in the following order from the Standard Input Stream: N: This number will always be a positive integer value smaller than 6 representing the number of sequences to be aligned Then N lines will follow representing each one of the sequences to be aligned. Each sequence will be comprised of M residues belonging to the alphabet A={A, C, G, T} A positive integer value K will follow representing the number of alignments that would have to be outputted to the screen Then K lines will follow each one containing an integer value representing a position in the ordered list of alignments that should be printed on the standard output stream (Note: You should consider these positions to be 1-based meaning that if for example the position equals to 4, then the 4th alignment in the ordered list should be printed).

Output Your program should output to the Standard Output Stream the total number of possible alignments for the given set of sequences as well as the alignments (as contained in the ordered result set) at the positions specified from the given input. In particular, the output should be formatted as follows: The first line of the output should always be: Possible Alignments: e.g. Possible Alignments: 25 Note: There is a space between the colon and the number Then K sets of lines should follow each one printing the alignment at the specified position of the ordered list of results. The format for the alignments should be as follows: If no alignment exists at the given position then the following line should be printed: There is no alignment at position: e.g. There is no alignment at position: 1000 Note: There is a space between the colon and the position If there exists an alignment at the given position, then at first the following line should be printed: Alignment at Position: e.g. Alignment at Position: 15 Note: There is a space between the colon and the position And then the alignment at this position of the ordered list should printed with the sequences ordered as they were given in the input. If for example the input sequences were Seq 1 = ACT and Seq 2 = GC (as in Example 2), and the given position was 15, the program should print: -ACT G--C Note: Each sequence ends with a newline character and does not contain any spaces before or after the sequence

Sample Input 1

```
2
AC
GG
6
-9
1
13
5
100
8
```

Sample Output 1

```
Possible Alignments: 13
There is no alignment at position: -9
Alignment at Position: 1
AC
GG
Alignment at Position: 13
--AC
GG--
Alignment at Position: 5
A-C
-GG
There is no alignment at position: 100
Alignment at Position: 8
AC--
--GG
```

Sample Input 2

```
2
ACT
GC
5
1
9
22
14
32
```

Sample Output 2

```
Possible Alignments: 25
Alignment at Position: 1
ACT
GC-
Alignment at Position: 9
AC-T
--GC
Alignment at Position: 22
-ACT-
G---C
Alignment at Position: 14
-ACT
```

```
G-C-
There is no alignment at position: 32
```

Problem author: IEEE

Suggest Edits

Emacs Normal Vim

Select Language: C

save code

```c
#include <stdio.h>
#include <string.h>
#include <math.h>
#include <stdlib.h>

int main() {

    /* Enter your code here. Read input from STDIN. Print output to STDOUT */
    return 0;
}

```

Line: 1 Col: 1 Count: 190

☐ **Use a custom test case**

Upload Code as File

Compile & Test Submit Code