Lab session 5: Imperative Programming

This is the fifth lab of a series of weekly programming labs. You are required to solve these programming exercises and submit your solutions to the automatic assessment system *Themis*. The system is 24/7 online. Note that the weekly deadlines are very strict (they are given for the CET time zone)! If you are a few seconds late, Themis will not accept your submission (which means you failed the exercise). Therefore, you are strongly advised to start working on the weekly labs as soon as they appear. Do not postpone making the labs! You can find Themis via the url `https://themis.housing.rug.nl`

Note: The theme of this lab is recursion. Each problem must be solved using recursion! The teaching assistants will (after the deadline) manually check the submissions that are accepted by Themis. If a passed program does not make use of recursion, then the scored points are subtracted by the teaching assistants. You are not allowed to use the math.h library.

Problem 1: Sum of Divisors

The number `12` has `5` proper divisors (divisors that are not itself): `[1 2 3 4 6]`. There are two ways to obtain `12` by summing a subset of its proper divisors:

1. `2 + 4 + 6 = 12`
2. `1 + 2 + 3 + 6 = 12.`

Note that each divisor can only be used once. As such a sequence such as `1 + 1 + 4 + 6 = 12` is invalid.

Write a program that accepts on its input an integer n (where $3 \leq n \leq 1600$), and outputs in how many different ways n can be expressed by summing a subset of its proper divisors. Computing possible subsequences becomes more computationally expensive as the number of divisors of number `n` increases. You may assume that the input would not yield more than 24 divisors which corresponds to $n = 360$.

Example 1:
input:
```
12
```
output:
```
2
```

Example 2:
input:
```
24
```
output:
```
5
```

Example 3:
input:
```
120
```
output:
```
278
```

Problem 2: Removing Values

Consider the sequence `[1 9 2 3 5 6 7]`. The longest strictly increasing subsequence is `[2 3 5 6 7]`. However if we are allowed to remove the number `9`, the longest increasing subsequence would have been `[1 2 3 5 6 7]`.

The input of this problem consists of a line containing two positive integers k and n (where $n \leq 10000$ and $k < n$), followed by a line containing the sequence of n integers. The output is the length of the longest increasing subsequence possible when removing at most k values in the original sequence. Since many deletions yield many possible subsequences, you may assume that the number of deletions is relatively small for large n.

Example 1:

input:
```
1 7
1 9 2 3 5 6 7
```
output:
```
6
```

Example 2:

input:
```
2 12
1 2 2 3 3 4 4 5 6 7 8 9
```
output:
```
8
```

Example 3:

input:
```
5 10
7 3 4 8 5 4 12 9 8 10
```
output:
```
5
```

Problem 3: Constrained Bitstrings

A binary string consists of sequence of `0`s and `1`s. In this exercise you are asked to generate all bitstrings under the following constraints:

1. no more than two `0`s or `1`s are allowed directly next to each other,
2. some positions in the string are prefilled by `0`s and `1`s which cannot be altered,
3. the other positions are wildcards indicated with an `'?'` in which you can pick a `0` or `1` as long as the first constraint is met.

An example of such a constrained bitstring that satisfied the first condition is `[1100101]` as there are no more than 2 `0`s or `1`s directly next to each other.

Now consider the bitstring `[0??010]`. There are three different options to fill the `'?'` slots namely: `[001010]`, `[010010]` and `[011010]`.

Write a program that accepts on the input two lines. The first line is a single integer n denoting the length of the constrained bitstring. The second line provides the constrained bitstring. On the output you should print all the possible combinations under the provided constraints in lexicographical ordering (i.e. in increasing binary order).

Example 1:

input:
```
6
0??010
```
output:
```
001010
010010
011010
```

Example 2:

input:
```
8
0?1?01?0
```
output:
```
00100100
00100110
00110100
00110110
01100100
01100110
```

Example 3:

input:
```
5
??101
```
output:
```
00101
01101
10101
```

Problem 4: Talking Knights

In a game of chess, a knight can make the following moves: move two squares vertically and one square horizontally, or move two squares horizontally and one square vertically. All possible moves are illustrated in the left subfigure below (red cell is the starting position, green cells can be reached in `1` move).

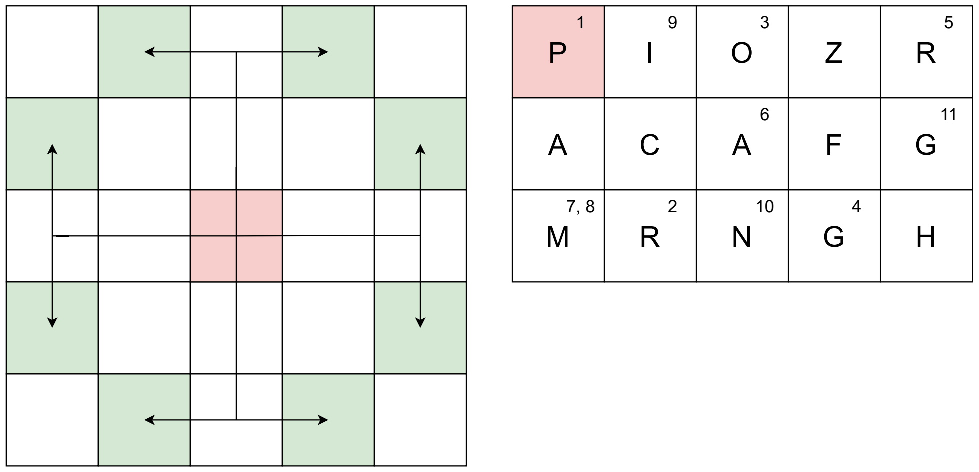

Figure 1: Left: moveset of a knight, Right: an example of a happy talking knight.

In this version knights move over an `n x m` grid in which each cell represents a character. The goal of this exercise is to see whether a word can be formed by subsequently performing legal moves over the `n x m` grid. In the right subfigure above, the red cell is the starting position represented by the coordinate `(0, 0)` while the bottom-right position is represented by the coordinate `(4, 2)`. The numbers in the top-right corner of each cell represent the order in which the cells are reached. Note that you are allowed to use a character *more than once* when arrived at a cell. If you follow the specified order, the word `programming` will be displayed.

Since there is only one knight on a single board, there is no easy way to communicate with other knights. The only way to communicate with another knight is by forming words on the `n x m` grid. If the knight is unable to form a specified word it becomes unhappy because it is not able to communicate it with others.

Write a program that first reads one line holding two integers `n` and `m` which specifies the size of the grid. Second, read another line holding two integers representing the starting position of the knight. Then read the `n x m` grid. Finally read a word to be found in the `n x m` grid. On the output you should print `Happy` if the knight is able to form the specified word. Otherwise print `Unhappy`.

Example 1:

input:
```
5 3
0 0
PIOZR
ACAFG
MRNGH
11
PROGRAMMING
```
output:
```
Happy
```

Example 2:

input:
```
5 3
4 1
PIOZR
ACAFG
MATGH
4
GOAT
```
output:
```
Happy
```

Example 3:

input:
```
5 5
1 1
ZABLO
DAEFG
HIJIK
CLMNO
PQTRS
6
ACTION
```
output:
```
Unhappy
```