Palindrome

PROBLEM

A palindrome is a symmetrical string, that is, a string read identically from left to right as well as from right to left. You are to write a program which, given a string, determines the minimal number of characters to be inserted into the string in order to obtain a palindrome.

As an example, by inserting 2 characters, the string "Ab3bd" can be transformed into a palindrome ("dAb3bAd" or "Adb3bdA"). However, inserting fewer than 2 characters does not produce a palindrome.

INPUT

The input file name is `PALIN.IN`. The first line contains one integer: the length of the input string N, $3 \le N \le 5000$. The second line contains one string with length N. The string is formed from uppercase letters from 'A' to 'Z', lowercase letters from 'a' to 'z' and digits from '0' to '9'. Uppercase and lowercase letters are to be considered distinct.

OUTPUT

The output file name is `PALIN.OUT`. The first line contains one integer, which is the desired minimal number.

EXAMPLE INPUT AND OUTPUT

`PALIN.IN`

```
5
Ab3bd
```

`PALIN.OUT`

```
2
```

Car Parking

PROBLEM

A parking center by the Great Wall has a long row of parking places. One end of the row is considered left and the other end is considered right. The row is full of cars. Each car has a type and several cars may be of the same type. The types are identified by integer values. A number of workers decide to order the cars parked in the row in ascending order from left to right by the car types using the following method. In what is called a round, each of the workers can simultaneously drive one car out of its place and then park it in a place from where a car has been moved out in the same round. It may be that some workers are not moving a car in a round. For efficiency, a small number of rounds is preferable.

Suppose that N is the number of cars and W is the number of workers. You are to write a program which, given the types of the parked cars and the number of workers, finds such a way to sort the cars that the number of rounds needed is at most $\lceil N/(W-1) \rceil$, that is $N/(W\text{-}1)$ rounded up. The minimal number of rounds is never greater than $\lceil N/(W-1) \rceil$.

Consider the following example. There are 10 parked cars of types 1,2,3, and 4 with 4 workers. The initial placement of the cars from left to right identified by their types is
2 3 3 4 4 2 1 1 3 1.
The minimal number of rounds is three, and the rounds can be implemented so that the placement after each round is as follows:
2 1 1 4 4 2 3 3 3 1 – after round 1,
2 1 1 2 4 3 3 3 4 1 – after round 2, and
1 1 1 2 2 3 3 3 4 4 – after round 3.

INPUT

The input file name is `CAR.IN`. The first line in the input file contains three integers. The first integer is the number of cars N, $2 \le N \le 20000$. The second integer is the number of types M, $2 \le M \le 50$. The car types are identified by the integers from 1 to M. There is at least one car of each type. The third integer is the number of workers W, $2 \le W \le M$. The second line contains N integers, where the ith integer is the type of the ith car in the row, starting from the left end of the row.

OUTPUT

The output file name is `CAR.OUT`. The first line of the output file contains one integer R, which is the number of rounds in the solution. The next R lines describe the rounds ordered from 1 to R. In each line, the first integer is the number of cars C, which are moved in that round. After that follow $2C$ integers, identifying car positions. The car positions are identified by the integers from 1 to N starting at the left end. The

first two are a pair describing how one of the cars is moved: the first integer is the position from the left end before the round and the second is the position from the left after the round. The next two integers are a pair describing how another car is moved, and so on. There may be several different solutions for these R lines, and your program only needs to output one of them.

EXAMPLE INPUT AND OUTPUT

CAR.IN

```
10 4 4
2 3 3 4 4 2 1 1 3 1
```

CAR.OUT

```
3
4 2 7 3 8 7 2 8 3
3 4 9 9 6 6 4
3 1 5 5 10 10 1
```

PARTIAL CREDIT

Suppose that your program's output for an evaluation run is R and $\lceil N/(W-1) \rceil$ is Q. If in your program's output the R rounds are not described correctly or they do not produce the desired order for the cars, then your score is 0. Otherwise, your score will be calculated from the maximum score as follows.

- $R \le Q$ 100% Score
- $R = Q+1$ 50% Score
- $R = Q+2$ 20% Score
- $R >= Q+3$ 0% Score

Median Strength

PROBLEM

A new space experiment involves N objects, labeled from 1 to N. It is known that N is odd. Each object has a distinct but unknown strength expressed by a natural number. For each strength Y, it holds that $1 \le Y \le N$. The object with median strength is the object X such that there are equally many objects having smaller strength than X as there are objects having greater strength than X. You are to write a program that determines the object with median strength. Unfortunately, the only way to compare the strengths is by a device that, given three distinct objects, determines the object with median strength among the three objects.

LIBRARY

You are given a library named `device` with three operations:

- `GetN`, to be called once at the beginning without arguments; it returns the value of N.
- `Med3`, called with three distinct object labels as arguments; it returns the label of the object with median (middle) strength.
- `Answer`, to be called once at the end, with one object label as argument; it reports the label of object X with median strength and properly ends the execution of your program.

The library `device` produces two text files: `MEDIAN.OUT` and `MEDIAN.LOG`. The first line of file `MEDIAN.OUT` contains one integer: the label of the object passed to the library in the call to `Answer`. The second line will contain one integer: the number of calls to `Med3` that have been performed by your program. The dialogue between your program and the library is recorded in the file `MEDIAN.LOG`.

Instruction for Pascal programmers: Include the import statement
```
uses device;
```
in the source code.

Instructions for C/C++ programmers: Use the instruction
```
#include "device.h"
```
in the source code, create a project `MEDIAN.PRJ` and add the files `MEDIAN.C` (`MEDIAN.CPP`) and `DEVICE.OBJ` into this project.

EXPERIMENTATION

You can experiment with the library by creating a text file `DEVICE.IN`. The file must contain two lines. The first line must contain one integer: the number of objects N. The second line must contain the integers from 1 to N in some order: the ith integer is the strength of the object with label i.

EXAMPLE

`DEVICE.IN`

```
5
2 5 4 3 1
```

The file `DEVICE.IN` above describes an input with 5 objects and strengths as below:

Label	1	2	3	4	5
Strength	2	5	4	3	1

Here is a correct sequence of 5 library calls:

1. `GetN` (in Pascal) or `GetN()` (in C/C++) returns 5.
2. `Med3(1,2,3)` returns 3.
3. `Med3(3,4,1)` returns 4.
4. `Med3(4,2,5)` returns 4.
5. `Answer(4)`

CONSTRAINTS

- For the number of objects N we have $5 \le N \le 1499$ and N is odd.
- For the object labels i, we have $1 \le i \le N$.
- For the object strengths Y, we have $1 \le Y \le N$ and all strengths are distinct.
- Pascal library file name: `device.tpu`
- Pascal function and procedure declarations:

```
function GetN: integer;
function Med3(x,y,z:integer):integer;
procedure Answer(m:integer);
```

- C/C++ library file names: `device.h`, `device.obj` (use large memory model)
- C/C++ function headers:

```
int GetN(void);
int Med3(int x, int y, int z);
void Answer(int m);
```

- No more than 7777 calls of function `Med3` are allowed per run.
- Your program must not read or write any files.