# Task Overview Sheet

| TASK | | MAINTAIN | CODE | REVERSE |
|---|---|---|---|---|
| **Time limit per test case** | | 1 second CPU | 2 seconds CPU | N/A |
| **Memory limit** | | 64 MB | 64 MB | N/A |
| **Compiler options** | **C** | `-pipe -O2 -lm` | | N/A |
| | **C++** | `-pipe -include /usr/include/stdlib.h -O2 -lm` | | N/A |
| | **Pascal** | `-So -O1 -XS` | | N/A |
| **Number of tests** | | 20 | 20 | 16 |
| **Maximum points per test** | | 5 | 5 | 7 |
| **Maximum total points** | | 100 | 100 | 100 |
| **Program header comment when using C** | | `/*`<br>`TASK: maintain`<br>`LANG: C`<br>`*/` | `/*`<br>`TASK: code`<br>`LANG: C`<br>`*/` | N/A |
| **Program header comment when using C++** | | `/*`<br>`TASK: maintain`<br>`LANG: C++`<br>`*/` | `/*`<br>`TASK: code`<br>`LANG: C++`<br>`*/` | N/A |
| **Program header comment when using Pascal** | | `{`<br>`TASK: maintain`<br>`LANG: PASCAL`<br>`}` | `{`<br>`TASK: code`<br>`LANG: PASCAL`<br>`}` | N/A |
| **Solution acceptance rule** | | Example input is solved correctly. | Example input is solved correctly. | The file is in the format specified. |

# Trail Maintenance (interactive task)

## TASK

Farmer John's cows wish to travel freely among the N ($1 \le N \le 200$) fields (numbered 1…N) on the farm, even though the fields are separated by forest. The cows wish to maintain trails between pairs of fields so that they can travel from any field to any other field using the maintained trails. Cows may travel along a maintained trail in either direction.

The cows do not build trails. Instead, they maintain wild animal trails that they have discovered. On any week, they can choose to maintain any or all of the wild animal trails they know about.

Always curious, the cows discover one new wild animal trail at the beginning of each week. They must then decide the set of trails to maintain for that week so that they can travel from any field to any other field. Cows can only use trails which they are currently maintaining.

The cows always want to minimize the total length of trail they must maintain. The cows can choose to maintain any subset of the wild animal trails they know about, regardless of which trails were maintained the previous week.

Wild animal trails (even when maintained) are never straight. Two trails that connect the same two fields might have different lengths. While two trails might cross, cows are so focused, they refuse to switch trails except when they are in a field.

At the beginning of each week, the cows will describe the wild animal trail they discovered. Your program must then output the minimum total length of trail the cows must maintain that week so that they can travel from any field to any other field, if there exists such a set of trails.

**Input:** *standard input*

- The first line of input contains two space-separated integers, N and W. W is the number of weeks the program will cover ($1 \le W \le 6000$).
- For each week, read a single line containing the wild animal trail that was discovered. This line contains three space-separated integers: the endpoints (field numbers) and the integer length of that trail (1…10000). No wild animal trail has the same field as both of its endpoints.

**Output:** *standard output*

Immediately after your program learns about the newly discovered wild animal trail, it should output a single line with the minimum total length of trail the cows must maintain so that they can travel from any field to any other field. If no set of trails allows the cows to travel from any field to any other field, output "`-1`".

Your program must exit after outputting the answer for the last week.

*Example exchange:*

| *Input* | *Output* | *Explanation* |
|---|---|---|
| `4 6` | | |
| `1 2 10` | | |
| | `-1` | No trail connects 4 to the rest of the fields. |
| `1 3 8` | | |
| | `-1` | No trail connects 4 to the rest of the fields. |
| `3 2 3` | | |
| | `-1` | No trail connects 4 to the rest of the fields. |
| `1 4 3` | | |
| | `14` | Maintain 1 4 3, 1 3 8, and 3 2 3. |
| `1 3 6` | | |
| | `12` | Maintain 1 4 3, 1 3 6, and 3 2 3. |
| `2 1 2` | | |
| | `8` | Maintain 1 4 3, 2 1 2, and 3 2 3. |
| | *program exit* | |

## CONSTRAINTS

| | |
|---|---|
| Running time | 1 second of CPU |
| Memory | 64 MB |

## SCORING

You will receive full points on each test case for which your program produces the correct output. No partial credit will be given on any test case.

# Comparing Code

## TASK

Racine Business Networks (RBN) has taken the Heuristic Algorithm Languages (HAL) company to court, claiming that HAL has taken source code from RBN UNIX™ and contributed it to the open-source operating system HALnix.

RBN and HAL both use a programming language with one statement per line, each of the form: `STOREA = STOREB + STOREC` where `STOREA`, `STOREB`, and `STOREC` are variable names. In particular, the first variable name starts in the first column and is followed by a space, an equals sign, a space, a second variable name, a space, the addition symbol, a space, and a third variable name. The same variable name might appear more than one time on a single line. Variable names consist of 1…8 uppercase ASCII letters ('A'…'Z').

RBN claims that HAL copied a consecutive sequence of lines directly from RBN's source code, making only minor modifications:

- RBN claims that HAL changed some of the variable names in order to disguise their crime. That is, HAL took a series of lines from RBN's program and, for each variable in it, changed all occurrences of that variable to a new variable name, although the new variable name might be the same as the original. Of course, no two variables were changed to the same new variable name.
- RBN also claims HAL might have changed the order of the right-hand side of some lines: `STOREA = STOREB + STOREC` might have been changed to `STOREA = STOREC + STOREB`.
- RBN claims that HAL did not change the order of the lines of RBN's source code.

Given source code for programs from RBN and HAL, find the longest consecutive sequence of lines from HAL's program that could have come from a consecutive sequence of lines from RBN's program using the modifications above. Note that the sequences of lines from the two programs do not have to start at the same line number in both files.

**Input:** `code.in`

- The first line of input contains two space-separated integers, R and H ($1 \le R \le 1000$; $1 \le H \le 1000$). R is the number of lines of source code in RBN's program; H is the number of lines of source code in HAL's program.
- The next R lines contain RBN's program.
- The next H lines contain HAL's program.

*Example input:*

```
4 3
RA = RB + RC
RC = D + RE
RF = RF + RJ
RE = RF + RF
HD = HE + HF
HM = HN + D
HN = HA + HB
```

**Output:** `code.out`

The output file should contain a single line with a single integer that is the length of the longest consecutive sequence of lines that HAL might have copied from RBN and transformed.

*Example output:*

```
2
```

Lines 1-2 of RBN's program are the same as lines 2-3 of HAL's program, if the following variable name substitutions are performed on RBN's program: RA → HM, RB → D, RC → HN, D → HA, RE → HB. There is no matching with three or more lines.

**CONSTRAINTS**

| | |
|---|---|
| Running time | 2 seconds of CPU |
| Memory | 64 MB |

**SCORING**

You will receive full points on each test case for which your program produces a correct output file. No partial credit will be given on any test case.

# Reverse (output-only task)

## TASK

Consider a Two-Operation Machine (TOM for short) with nine registers, numbered 1…9. Each register stores a non-negative integer in the range 0…1000. The machine has two operations:

| | |
|---|---|
| `S i j` | Store one plus the value of register i into register j. Note that i may equal j. |
| `P i` | Print the value of register i. |

A TOM program includes a set of initial values for the registers and a sequence of operations. Given an integer N ($0 \le N \le 255$), generate a TOM program that prints the decreasing sequence of integers N, N-1, N-2, …, 0. The maximum number of consecutive S-operations should be as small as possible.

Example of a TOM program and its execution for N=2:

| Operation | New Register Values | | | | | | | | | Printed Value |
|---|---|---|---|---|---|---|---|---|---|---|
| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | |
| Initial values | 0 | 2 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | |
| `P 2` | 0 | 2 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 2 |
| `S 1 3` | 0 | 2 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | |
| `P 3` | 0 | 2 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |
| `P 1` | 0 | 2 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

Input cases are numbered 1 through 16 and are available via the contest server.

## Input:

- The first line of the input file contains K, an integer specifying the input case number.
- The second line of input contains N.

*Example input:*

```
1
2
```

## Output:

The first line of output should be the string "`FILE reverse K`", where K is the case number.

The second line of output should contain nine space-separated values representing the desired initial values of the registers, in order (register 1, then register 2, etc.).

The rest of the output file should contain the ordered list of operations to be performed, one per line. Thus, the third line contains the first operation to perform, and so on. The last line of the file should be the one that prints 0. Each line should be a valid operation. The instructions should be formatted as in the example output.

*Example output #1 (partial points):*

```
FILE reverse 1
0 2 0 0 0 0 0 0 0
P 2
S 1 3
P 3
P 1
```

*Example output #2 (full points):*

```
FILE reverse 1
0 2 1 0 0 0 0 0 0
P 2
P 3
P 1
```

**SCORING**

Scoring of each test case will be based on correctness and optimality of the TOM program given.

*Correctness:* 20%

A TOM program is correct if it performs no more than 131 consecutive S-operations and the sequence of values printed by it is correct (contains exactly N+1 integers, starting at N and ending at 0). If any S-operation causes a register to overflow, the TOM program is considered incorrect.

*Optimality:* 80%

Optimality of a correct TOM program is measured by the maximum number of consecutive S-operations in the program, which should be as small as possible. Scoring will be based on the difference between your TOM program and the best known TOM program.