Torneio Inter-Universitário de Programação

TIUP'2007 – 3ª etapa

http://tiup.di.fc.ul.pt/

Faculdade de Ciências da Universidade de Lisboa

16 de Maio de 2007
17h00m – 20h00m

Contents

Informação geral

1. A duração da prova é de 3 horas para os 5 problemas.
2. Os programas devem ler do standard-input e escrever no standard-output.
3. Os inputs usados para testar os programas submetidos obedecem às seguintes normas:

   - Não há espaços no fim de cada linha e todas as linhas terminam com mudança de linha.

   - Não se usa espaçamento múltiplo, a menos que o enunciado do problema respectivo o especifique explicitamente.

1. Os outputs devem obedecer às mesmas normas do ponto anterior.
2. O servidor Mooshak da prova (http://tiup.di.fc.ul.pt/) está configurado para usar as seguintes versões de compiladores: {gcc,g++} 4.1.1.; java 1.5.0_11 (Sun).

Comissão Científica da 3ª Etapa do TIUP2007

- Alexandre Pinto <apinto@di.fc.ul.pt>
- António Ferreira <asfe@di.fc.ul.pt>
- Carlos Teixeira <cjct@di.fc.ul.pt>
- Francisco Couto <fcouto@di.fc.ul.pt>
- Francisco Martins <fmartins@di.fc.ul.pt>
- Isabel Nunes <in@di.fc.ul.pt>
- Mário Calha <mjc@di.fc.ul.pt>
- Teresa Chambel <tc@di.fc.ul.pt>

Problem A

Text on 9 Keys

T9, which stands for ***T****ext on* ***9*** *keys*, is a predictive text technology for SMS composition in mobile phones. **T9**'s objective is to make it easier to type text messages on small mobile devices. The technology allows words to be entered by a single keypress for each letter.

With this feature ON, the user can write, for example, the word "concurso" by pressing the keys 26628776 one at a time, in sequence, instead of having to press the 2 key three times (to obtain the letter 'c'), the 6 key 3 times ('o'), the 6 key two times ('n') and so on, until, finally (with his finger already aching), the 6 key three times (to obtain the last 'o').
While in **T9** mode, at any time during the writing of the word "concurso" (for example, after pressing 266), the user may ask to see all words in the mobile phone internal dictionary beginning with letters that can be obtained with the sequence already written. For example, after pressing 266, one would obtain the words "amor", "ano", "boné", "bobadela", "bom", "bombeiro", "comando", "combinação", "comer", "comprar" "concurso", "conselho", etc.

Problem
Your friend wants to implement a simplified **T9** option in his machine, and you are going to help him. Your task is to find, among a given dictionary, the words that match a given initial sequence of digits between 2 and 9 knowing that the letters that can be obtained from each key are as shown in the following table (notice that your friend is not interested on having any numbers, lowercase letters, nor any characters with accents or diacritical marks as, for example, á, õ, ê, ç, etc):

Digit	Letters obtained
2	A, B, C
3	D, E, F
4	G, H, I
5	J, K, L
6	M, N, O
7	P, Q, R, S
8	T, U, V
9	W, X, Y, Z

Input
The first line of the input contains a sequence *s* of *n* digits (*n*<=20), standing for the keys that are pressed to write a (part of a) word. The following lines define the dictionary: each line contains one word (of 20 characters at most). The dictionary contains a maximum of 10 000 words.

Output
The first line of the output contains the sequence *s*. The next lines contain the words in the dictionary, by ascending lexicographic order, whose first *n* characters correspond to the input sequence *s* (according to the **T9** table above).

Sample Input 1

```
737
ANANAS
CARAMELO
BANANA
MACA
PAPAIA
REQUEIJAO
PERA
MANGA
PERABACATE
ROMA
```

Sample Output 1

```
737
PERA
PERABACATE
REQUEIJAO
```

Sample Input 2

```
233767
AZINHEIRA
CARVALHO
CEDRO
CEREJEIRA
CIPRESTE
EUCALIPTO
FAIA
LARANJEIRA
MACIEIRA
NOGUEIRA
OLIVEIRA
PEREIRA
PLATANO
```

Sample Output 2

```
233767
```

Problem B

Odious Numbers

An **odious** number is a nonnegative number that has an odd number of 1s in its binary representation. The first few odious numbers are therefore 1, 2, 4, 7, 8, 11, 13, 14, 16, 19… Numbers that are not odious are said to be **evil** numbers.

Problem

Decide which numbers, in a given closed interval, are odious numbers, and find all pairs of consecutive numbers (n,n+1), where n is both an odious and an odd number and n+1 is both an evil and an even number: (odious_odd,evil_even).

Input

The input has one line containing two positive integers between 1 and 10000 each, separated by a space; the first integer – L – is the low limit of the closed interval; the second one – H – is the high limit.

Output

The output is composed of the pairs of consecutive numbers oe(odious_odd,evil_even) and the set of odious numbers in the form o{o1,o2,o3…} that lie in the interval [L,H]. There must be a line for each pair oe(odious_odd,evil_even) and a final line with the set of odious numbers. In the absence of resulting pairs, no line is written for the pairs. In the absence of resulting odious numbers, the last line contains the empty set o{}.

Sample Input 1

```
1 20
```

Sample Output 1

```
oe(11,12)
oe(19,20)
o{1,2,4,7,8,11,13,14,16,19}
```

Sample Input 2

```
3 3
```

Sample Output 2

```
o{}
```

Problem C

Similarity between DNA Sequences

DNA sequences (genes) can be represented as sequences of four letters ACGT, corresponding to the four submolecules forming DNA. When biologists find a new sequence, they typically want to know what other sequences it is most similar to. One way of computing how similar two sequences are is to find the longest DNA sequence which is a subsequence of both sequences.

Problem
You are to find the longest DNA sequence which is both a subsequence of two given sequences. A subsequence of some sequence is a new sequence which is formed from the original sequence by deleting some of the elements without disturbing the relative positions of the remaining elements. In some cases, the result will be the empty string. When there are more than one longest subsequence, you may break ties by selecting the one more close to the left on the first DNA sequence.

Input
The input has two lines, each containing a DNA sequence.

Output
The first two lines of the output contain the original ADN sequences given in the input. The third line contains the statement "Longest common sequence:" and, finally, the last line contains the longest common sequence between the two given DNA sequences. If there are no common subsequences, this fourth line is not written.

Sample Input 1
```
ACGGTGTCGTG
CGTTCGGCTA
```

Sample Output 1
```
ACGGTGTCGTG
CGTTCGGCTA
Longest common sequence:
CGTTCGT
```

Sample Input 2
```
ATGC
CGTA
```

Sample Output 2
```
ATGC
CGTA
Longest common sequence:
A
```

Problem D

Battle of Programs

Core War is a programming game, created by D. G. Jones and A. K. Dewdney (1984), in which two programs—the *warriors*—compete for the control of a *MARS* (*Memory Array Redcode Simulator*) computer. The warriors are written in a special assembly language named **Redcode**. The goal of the game is to cause the opposing program to terminate, leaving your program in possession of the machine.

Problem
Construct a *MARS* virtual machine to run a battle between two warriors.

The *core* or *arena* (the memory of the MARS computer) is a circular random access memory of instructions, empty except for the competing programs. The basic unit of memory is one instruction, instead of one byte as is usual.
Each Redcode *instruction* contains five parts:

- the *OpCode* itself;
- the type of the source value (*TA*);
- the source field (*A field*);
- the type of the target value (*TB*);
- the target field (*B field*).

The assembly mnemonics and their informal semantics is explained below:

Mnemonic	A field	B field	#code	Meaning
DAT		value	0	Data variables. If executed, kills the running program. Fields TA, TB, and A are 0.
MOV	source	target	1	Copies data from *source* to *target*.
ADD	source	target	2	Adds *source* to *target*. Stores the result in *target*.
JMP	target		4	Continues the execution in *target* address. Fields TB and B are 0.
CMP	source	target	8	Compares *source* and *target* and skips the next instruction if they are equal.

Programs have no way of knowing where they reside in memory or where the memory ends, since there are no absolute addresses. Addresses are always computed relatively to the current address of execution. So address **0** means the current instruction, not the first instruction in the memory; the next instruction has address **1**. All values are computed modulo the size of the memory (remember that the memory is circular).

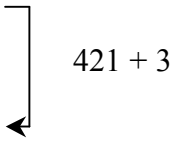
Value types (TA and TB) define the meaning of the values in fields (A and B) and may be of three forms:
- 0 means an immediate value, *i.e.* the value of the field is interpreted as an instruction or a number (it can not be used as an address);
- 1 means a direct address, *i.e.* the value of the field is treated as an address relatively to the address of the current instruction;
- 2 means an indirect address, *i.e.* the value of the field represents the address where there is a value to be treated as a direct address.

As an example, instruction 1 0 4 1 3 is interpreted as

1	OpCode: MOV (see table above)
0	TA: Field A is an immediate value
4	Field A: is number (DAT instruction) 4
1	TB: Field B is a direct address
3	Field B: address of the instruction three positions after the current instruction

If the address of the instruction is 421

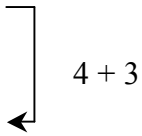
Memory Address	Instructions
…	…
421	1 0 4 1 3
422	…
423	…
424	0 0 0 0 4
…	



The MARS virtual machine executes one instruction at a time, and then proceeds to the next one in the memory, unless the instruction explicitly tells it to jump to another address. On start up, MARS loads both warriors into memory and starts executing the first instruction of the first program loaded. Then it runs the first instruction of the second program and continues executing instructions alternately for both programs, one instruction at a time. If MARS tries to execute a DAT instruction of one program, the computation terminates and the other program wins the battle.

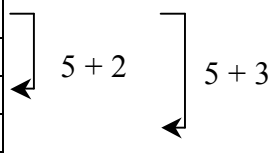
As an example consider the following program for warrior 1, loaded at absolute address 4 of a memory of size 20 (the program for warrior 2 is loaded, for example, at absolute address 13). In this example we consider absolute addresses between 0 and 19.

Memory Address	Instructions
0	0 0 0 0 0
…	…
4	2 0 4 1 3
5	1 0 1 2 2
6	4 1 18 0 0
7	0 0 0 0 19
8	



The first instruction adds (OpCode is 2) immediate number 4 (TA is 0) to the value in direct address 3 (TB is 1). This means that number 4 should be added to the value in address 7 (the address of the add instruction 4 plus 3). The value in address 7 becomes 0 0 0 0 3 (mind the modular addition (19 + 4) mod 20).

Memory Address	Instructions
…	…
4	2 0 4 1 3
5	1 0 1 2 2
6	4 1 18 0 0
7	0 0 0 0 3
8	



After executing the first instruction of warrior 2, the program proceeds by executing the second instruction of warrior 1 (in adress 5): move (OpCode is 1) immediate value 1

(TA is 0) to indirect address 2 (TB is 2). To resolve an indirect address, first access memory address 7 (the address of the *mov* instruction 5 plus 2) and then use this number (3) as a direct address. Therefore, immediate value 1 will be written in memory address 8 (the address of the *mov* instruction 5 plus 3 — the value in memory address 7). After executing the second instruction of warrior 2, the next instruction of warrior 1 (in address 6) is executed: it tells MARS to proceed execution (jump) in instruction 4 (the address of the jump instruction (6 + 18) mod 20).

This program will write DAT instructions every four addresses of memory. Why? It hopes to hit its opponent and forces him to execute a DAT instruction, thus loosing the battle.

Input

The input file contains the information about the size of the arena, how many steps should be run in case no program dominates MARS, and the worriers themselves.

- Line 1 indicates the size of the memory;
- Line 2 specifies the maximum number of instructions to run for each program;
- Line 3 contains the absolute address where program 1 must be loaded.
- Line 4 and the following include the list of instructions for program 1. The list is terminated with a line containing 9 0 0 0 0.
- The line after the list of instructions of program 1 contains the absolute address where program 2 must be loaded.
- The following lines consist of the list of the instructions for program 2 (terminated also with 9 0 0 0 0).

Each instruction is in a line and is composed of 5 numbers (as described above) and an optional string containing a comment to facilitates the reading of the program and has no meaning for the virtual machine. For the sake of readability (in comments) we prefix immediate values with symbol # and indirect addresses with symbol @. The line indicating the end of a program list (9 0 0 0 0) has no meaning in MARS (it is not an instruction) and must not be loaded into memory.

Output

The simulator should output:

- In line 1 the number of steps executed;
- In line 2 the winner of the battle
  - 0 – tie game;
  - 1 – player one won;
  - 2 – player two won;

In line 3 and the following the contents of the MARS memory, one instruction per line.

Sample Input

```
20
7
4
2 0 4 1 3        ADD #4, 3
1 0 1 2 2        MOV #1, @2
4 1 18 0 0      JMP -2
0 0 0 0 19      DAT -1
```

```
9 0 0 0 0        END
13
1 1 0 1 1        MOV 0, 1
9 0 0 0 0        END
```

Sample Output

```
7
0
1 1 0 1 1
0 0 0 0 0
0 0 0 0 0
0 0 0 0 0
2 0 4 1 3
1 0 1 2 2
4 1 18 0 0
0 0 0 0 11
0 0 0 0 1
0 0 0 0 0
0 0 0 0 0
0 0 0 0 0
0 0 0 0 1
1 1 0 1 1
1 1 0 1 1
1 1 0 1 1
1 1 0 1 1
1 1 0 1 1
1 1 0 1 1
1 1 0 1 1
```

Problem E

POP

The workers of the POP (Prevention of Plagiarism) department in TOFU (Tellers of Fairytales Unlimited) kingdom are tired of spending hours and hours and hours of their time, reading and comparing the stories created by their eldest in search of any traces of "inspiration borrowing", more commonly known as plagiarism.

Problem

You are going to help POP workers by building a program that finds any repeated piece of text in any two stories.

Input

The first line contains the first story. The second line contains the second story. These stories only contain alphanumeric characters (no characters with accents or diacritical marks as, for example, á, õ, ê, ç, etc, are allowed), commas, points, and spaces. There can be only one space between words.

Output

The output contains all pieces of consecutive text (with no leading nor trailing spaces) with more than 5 characters, that appear in the two stories. These pieces appear one per line, by descending order of length. If there are more than one piece with the same length, the order will be the same as the one in which they appear in the first story.

Sample Input 1

```
Once upon a time in the TOFU kingdom, a dwarf had an
incredible idea. He thought he would be capable of
painting the whole landscape red.
Once upon a time in the TOFU land, a goblin had an
inconceivable idea. He thought he could paint the whole
landscape blue.
```

Sample Output 1

```
Once upon a time in the TOFU
ble idea. He thought he
the whole landscape
had an inc
```

Sample Input 2

```
roses are red, violets are blue, sugar is sweet, and so
are you.
we will buy very pretty things, a-walking through the
faubourgs, violets are blue, roses are red, i love my
loves.
```

Sample Output 2

```
violets are blue,
roses are red,
```