Introduction et quelques problèmes simples

Objectifs de ce premier TP : présentation du cours, résolution de quelques problèmes simples.
Notions : problème $3n + 1$, déplacement de blocs, nombres chanceux, tri de crêpes

1 Introduction

Je vais me contenter de reprendre les mots de Nicolas Schabanel : « L'écriture d'un programme est souvent l'objet d'un compromis entre son temps d'exécution (efficacité algorithmique de la solution codée) et son temps de développement (réflexion, implantation et débuggage). Deux attitudes caricaturales sont : a) rechercher la solution la plus efficace théoriquement (c.-à-d. quand n temps vers l'infini) même si elle est très compliquée à implémenter ou b) implémenter la première idée venue. L'objectif de ce cours est d'essayer de trouver un équilibre entre ces deux façons d'agir, idéalement de développer rapidement et de façon sûre une solution efficace sans avoir à débugguer.

Le choix de la solution à retenir repose typiquement sur le temps disponible d'une part et les caractéristiques particulières des instances à résoudre d'autre part (taille, décomposition éventuelle,...). Le choix de la structure de données est également un paramètre crucial d'efficacité.

Dans ce cours, nous étudierons différents problèmes (algorithmiques purs) que nous cherchons à résoudre effectivement en écrivant un programme (dans le langage de votre choix - C/C++/Java...) qui sera testé/validé sur des jeux de données. L'objectif de ce cours est à la fois d'obtenir une compréhension profonde des algorithmes classiques et de leurs performances réelles, et d'apprendre à résoudre efficacement et effectivement un problème algorithmique. Le cours se déroulera en salle machine et se décomposera en essentiellement deux types de sessions : Algorithmique avancée et implantation des algorithmes : algorithmes classiques : enveloppe convexe, couplage parfait, flot, systèmes dans Z/2Z, composantes fortement connexes, branch-and-bound, algorithme de Gauss,... structures de données classiques : différentes tables de hachage, tables dynamiques, tas, union-find,... Résolution effective de problèmes : choisir la structure de données, écrire un pseudo-code lisible, implémenter, éviter d'avoir à débugguer en temps limité ou non Ce cours a également pour but de préparer trois équipes d'étudiants (environ) au concours d'algorithmique et programmation « ACM International Collegiate Programming Contest », et dans le cadre de cette préparation, de constituer une base de données d'algorithmes écrits de façon lisible et efficace qui accompagnera nos candidats (un listing de 50 pages de code est autorisé lors des épreuves). »

2 Mise en route

Tout d'abord, arrangez-vous entre vous pour former des équipes de 2 ou 3 personnes. Les équipes de 4 ne seront pas acceptées. Normalement, une bonne répartition consiste en une personne qui s'occupe du code et une ou deux autres qui s'occupent de l'algorithmique.
Inscrivez votre équipe sur le site des annales : `http://icpcres.ecs.baylor.edu/onlinejudge/`. Il y a suffisamment de matière sur ce site pour vous entraîner si vous êtes motivés.

3 Les langages utilisés : C, C++, Java

Ces 3 langages sont acceptés par le juge en ligne. Cependant, le C++ est conseillé : on peut s'en servir comme du C pour les débutants, et il permet d'utiliser facilement des structures de données évoluées. Faire du code orienté objet n'est absolument pas indispensable, surtout s'il vous ralentit.
Voici quelques liens pour débuter en C++ :

- `http://www.4p8.com/eric.brasseur/cppcen.html` : une introduction au C++ , axée sur les apports du C++ par rapport au C. On y voit que finalement, l'aspect objet du C++ est loin d'être la seule nouveauté.
- `http://www.fredosaurus.com/notes-cpp/index.html` : référence plus complète du langage
- `http://www.cplusplus.com/reference/stl/` (*Standard Template Library*) : une librairie standard qui va bien vous dépanner avec ses nombreuses structures de données.
- `http://cppreference.com/` : une autre référence sur le C++

Pour le Java, une seule adresse : `http://java.sun.com/`. Vous pourrez y trouver des tutoriaux, la doc de référence, etc.. Cependant, si vous ne connaissez pas le Java, je vous conseille vivement le C++.

4 Quelques problèmes simples

4.1 The $3n + 1$ problem

Le problème est disponible en ligne ici : `http://acm.uva.es/p/v1/100.html`.

Background

Problems in Computer Science are often classified as belonging to a certain class of problems (e.g., NP, Unsolvable, Recursive). In this problem you will be analyzing a property of an algorithm whose classification is not known for all possible inputs.

The Problem

Consider the following algorithm :

1. input n
2. print n
3. if $n = 1$ then STOP
4. if n is odd then $n \leftarrow 3 \times n + 1$
5. else $n \leftarrow n/2$
6. GOTO 2

Given the input 22, the following sequence of numbers will be printed 22 11 34 17 52 26 13 40 20 10 5 16 8 4 2 1
It is conjectured that the algorithm above will terminate (when a 1 is printed) for any integral input value. Despite the simplicity of the algorithm, it is unknown whether this conjecture is true. It has been verified, however, for all integers n such that $0 < n < 1,000,000$ (and, in fact, for many more numbers than this.)
Given an input n, it is possible to determine the number of numbers printed (including the 1). For a given n this is called the cycle-length of n. In the example above, the cycle length of 22 is 16.
For any two numbers i and j you are to determine the maximum cycle length over all numbers between i and j.

The Input

The input will consist of a series of pairs of integers i and j, one pair of integers per line. All integers will be less than $1,000,000$ and greater than 0.
You should process all pairs of integers and for each pair determine the maximum cycle length over all integers between and including i and j.
You can assume that no operation overflows a 32-bit integer.

The Output

For each pair of input integers i and j you should output i, j, and the maximum cycle length for integers between and including i and j. These three numbers should be separated by at least one space with all three numbers on one line and with one line of output for each line of input. The integers i and j must appear in the output in the same order in which they appeared in the input and should be followed by the maximum cycle length (on the same line).

Sample Input

```
  1 10
100 200
201 210
900 1000
```

Sample Output

```
  1 10 20
100 200 125
201 210 89
900 1000 174
```

4.2 The Blocks Problem

Le problème est disponible en ligne ici : `http://online-judge.uva.es/p/v1/101.html`.

Background

Many areas of Computer Science use simple, abstract domains for both analytical and empirical studies. For example, an early AI study of planning and robotics (STRIPS) used a block world in which a robot arm performed tasks involving the manipulation of blocks. In this problem you will model a simple block world under certain rules and constraints. Rather than determine how to achieve a specified state, you will "program" a robotic arm to respond to a limited set of commands.

The Problem

The problem is to parse a series of commands that instruct a robot arm in how to manipulate blocks that lie on a flat table. Initially there are n blocks on the table (numbered from 0 to $n - 1$) with block b_i adjacent to block b_{i+1} for all $0 \leq i < n - 1$ as shown in the diagram below :
Figure : Initial Blocks World
The valid commands for the robot arm that manipulates blocks are :

move a onto b where a and b are block numbers, puts block a onto block b after returning any blocks that are stacked on top of blocks a and b to their initial positions.

move a over b where a and b are block numbers, puts block a onto the top of the stack containing block b, after returning any blocks that are stacked on top of block a to their initial positions.

pile a onto b where a and b are block numbers, moves the pile of blocks consisting of block a, and any blocks that are stacked above block a, onto block b. All blocks on top of block b are moved to their initial positions prior to the pile taking place. The blocks stacked above block a retain their order when moved.

pile a over b where a and b are block numbers, puts the pile of blocks consisting of block a, and any blocks that are stacked above block a, onto the top of the stack containing block b. The blocks

stacked above block a retain their original order when moved.

quit terminates manipulations in the block world.

Any command in which $a = b$ or in which a and b are in the same stack of blocks is an illegal command. All illegal commands should be ignored and should have no affect on the configuration of blocks.

The Input

The input begins with an integer n on a line by itself representing the number of blocks in the block world. You may assume that $0 < n < 25$. The number of blocks is followed by a sequence of block commands, one command per line. Your program should process all commands until the quit command is encountered.
You may assume that all commands will be of the form specified above. There will be no syntactically incorrect commands.

The Output

The output should consist of the final state of the blocks world. Each original block position numbered i (where n is the number of blocks) should appear followed immediately by a colon. If there is at least a block on it, the colon must be followed by one space, followed by a list of blocks that appear stacked in that position with each block number separated from other block numbers by a space. Don't put any trailing spaces on a line.
There should be one line of output for each block position (i.e., n lines of output where n is the integer on the first line of input).

Sample Input

```
10
move 9 onto 1
move 8 over 1
move 7 over 1
move 6 over 1
pile 8 over 6
pile 8 over 5
move 2 over 1
move 4 over 9
quit
```

Sample Output

```
0 : 0
1 : 1 9 2 4
2 :
3 : 3
4 :
5 : 5 8 7 6
6 :
7 :
8 :
9 :
```

4.3 Stacks of Flapjacks

Le problème est disponible en ligne ici : `http://acm.uva.es/p/v1/120.html`.

Background

Stacks and Queues are often considered the bread and butter of data structures and find use in architecture, parsing, operating systems, and discrete event simulation. Stacks are also important in the theory of formal languages.
This problem involves both butter and sustenance in the form of pancakes rather than bread in addition to a finicky server who flips pancakes according to a unique, but complete set of rules.

The Problem

Given a stack of pancakes, you are to write a program that indicates how the stack can be sorted so that the largest pancake is on the bottom and the smallest pancake is on the top. The size of a pancake is given by the pancake's diameter. All pancakes in a stack have different diameters.
Sorting a stack is done by a sequence of pancake "flips". A flip consists of inserting a spatula between two pancakes in a stack and flipping (reversing) the pancakes on the spatula (reversing the sub-stack). A flip is specified by giving the position of the pancake on the bottom of the sub-stack to be flipped (relative to the whole stack). The pancake on the bottom of the whole stack has position 1 and the pancake on the top of a stack of n pancakes has position n.
A stack is specified by giving the diameter of each pancake in the stack in the order in which the pancakes appear.
For example, consider the three stacks of pancakes below (in which pancake 8 is the top-most pancake of the left stack) :

8	7	2
4	6	5
6	4	8
7	8	4
5	5	6
2	2	7

The stack on the left can be transformed to the stack in the middle via flip(3). The middle stack can be transformed into the right stack via the command flip(1).

The Input

The input consists of a sequence of stacks of pancakes. Each stack will consist of between 1 and 30 pancakes and each pancake will have an integer diameter between 1 and 100. The input is terminated by end-of-file. Each stack is given as a single line of input with the top pancake on a stack appearing first on a line, the bottom pancake appearing last, and all pancakes separated by a space.

The Output

For each stack of pancakes, the output should echo the original stack on one line, followed by some sequence of flips that results in the stack of pancakes being sorted so that the largest diameter pancake is on the bottom and the smallest on top. For each stack the sequence of flips should be terminated by a 0 (indicating no more flips necessary). Once a stack is sorted, no more flips should be made.

Sample Input

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

Sample Output

```
  1 2 3 4 5
0
5 4 3 2 1
1 0
5 1 2 3 4
1 2 0
```

4.4 Lucky numbers

Le problème est disponible en ligne ici : `http://acm.uva.es/p/v109/10909.html`.

Background

Lucky numbers are defined by a variation of the well-known sieve of Eratosthenes. Beginning with the natural numbers strike out all even ones, leaving the odd numbers 1, 3, 5, 7, 9, 11, 13, ... The second number is 3, next strike out every third number, leaving 1, 3, 7, 9, 13, ... The third number is 7, next strike out every seventh number and continue this process infinite number of times. The numbers surviving are called lucky numbers. The first few lucky numbers are :

$$1, 3, 7, 9, 13, 15, 21, 25, 31, 33, \ldots$$

In this problem your task is to test whether a number can be written as the sum of two lucky numbers.

The Input

The input file contains at most 100000 lines of input. Each line contains a single integer n ($0 < n \leq 2000000$). Input is terminated by end of file.

The Output

For each line of input produce one line of output. This line should be of one of the following types depending on whether n is expressible as the sum of two lucky numbers.

n is not the sum of two luckies!
n is the sum of L1 and L2.

For the second case, always make sure that (L2-L1) is nonnegative and minimized.

Sample Input

```
  11
12
```

Sample Output

```
  11 is not the sum of two luckies!
12 is the sum of 3 and 9.
```

Introduction et quelques problèmes simples

Un corrigé