Due date: 11:00pm Sunday October 12, 2008

Background

Sorting, the act of putting items in a certain order, is an interesting and significant problem in the field of computing. Most sorting problems can be solved on *O(nlogn)* time, but others fall into the category of NP-complete problems. To solve these efficiently for large input sizes we ca often apply dynamic programming techniques or utilize heuristic-based approaches.

For this assignment you will be solving the problem of sorting Lego pieces. You will solve it for small input sizes so that you may use brute-force sorting techniques.

Problem Description

As many of you know, CST uses Lego in some second term courses. At the start of the term the Lego is neatly sorted such that each type of lego piece has its own container. At the end of term the Lego is typically all mixed up in the containers. In this problem we have **n** types of Lego pieces, and **n** storage containers.

You will be given an input file which indicates the number of pieces of each type of Lego that are currently in each container. To sort the Lego you must move pieces such that each container holds only one type of Lego piece.

The challenge for this sorting problem is to minimize the number of pieces of Lego that you have to move.

You may assume that the containers are large enough to hold all the pieces of any specific type, and that there will never be more than $\mathbf{m = 2^{31}}$ Lego pieces. For purposes of this assignment you may also assume that $\mathbf{1 \le n \le 10}$.

Input

Each input file contains **t** test cases, where $\mathbf{1 \le t \le 100}$. Each test case begins with a line containing a single integer **n** indicating the number of containers of Lego to be sorted. The second line of each test case contains **n** unique alphabetic characters. Each character represents a specific type of Lego piece that you need to sort. The next **n** lines each contain **n** integers indicating the number of pieces of each type that are in each container.

For example, the test case ...

```
2
a b
3 5
7 2
```

... indicates that there are two types of Lego (a and b) to sort. The first container currently has 3 pieces of a and 5 pieces of b. The second container has 7 pieces of a and 2 pieces of b.

Integers and characters on a line will be separated by one or more spaces.

An end of file character indicates that there are no more test cases to process.

Output

Output is to the console.

For each test case there will be one line of output indicating which types of pieces go in which containers so as to minimize the total number of Lego pieces moved. You must also output the minimum number of pieces moved.

The output should consist of a string of the **n** characters that represent the types of Lego pieces.

The first character of the string represents the type of piece associated with the first container, the second character of the string represents the type of piece associated with the second container, and the $\mathbf{n}^{th}$ character represents the type of piece associated with the $\mathbf{n}^{th}$ container.

The integer indicating the minimum number of pieces moved must follow the string.

If more than one assignment of pieces to containers yields the minimum number of movements then the alphabetically first string representing a minimal configuration should be printed.

Sample Input

```
2
a b
3 5
7 2
3
x y z
4 6 4
6 4 6
4 6 4
4
C A T S
6 1 3 2
4 9 5 1
7 1 3 6
5 3 2 8
```

Sample Output

```
ba 5                                    // note: this is the correct output - v2
xzy 28
TACS 39
```

Other Requirements and Guidelines

1. ***You may work in pairs***, or by yourself. If you work with someone else, each person is expected to contribute equally to the assignment. Reports of partner-exploitation should be directed to the course instructor. The names and ID's of both partners must be included as per the submission guidelines below.
2. For this assignment you ***must use Java***, and your program must compile using the Java 6 release.
3. Your program must read the ***input from a text file***. The name of the text file will be passed as the only ***command line argument*** (as per the following submission requirements).
4. If your solution requires you to generate permutations, you must implement the ***Johnson-Trotter algorithm***.
5. You are free to use any of the data structures or algorithms in the ***Java Collections Framework***, or you can develop your own structures and algorithms if you prefer.
6. Your program will be evaluated against the ***instructors test files***, which will not be provided in advance. It is up to you to understand the requirements and make sure your solution meets them.
7. Programming style, design, technique, and inline documentation are up to you. This is not a course in software engineering or software design, however, you are expected to write code that can be easily understood by the instructor, and adheres to ***sound principles of program design*** as taught to you in previous CST courses. Also, since this is an algorithm course, we will be looking at the overall efficiency and sophistication of the algorithms and structures that you use.
8. Each source code file in your submission must include a ***header block*** listing the author(s) name(s), date, and description of the file contents.
9. These requirements are subject to change. Any clarifications or changes will be announced in WebCT, and documented in a new version of this file. It is your responsibility to ***monitor WebCT*** so that you are aware of any assignment updates or changes.

Submission Details

To receive full marks, your assignment must be submitted to WebCT before ***11:00pm Sunday October 12, 2008.*** WebCT will be configured such that assignments submitted after this time will be marked as ***late***. Late assignments will be graded as follows:

- for each day (24hrs or portion thereof) the assignment is late, 15 marks will be deducted, up to a maximum of 75. Programs that are more than 5 days late will not be graded.

There will be no exceptions to this policy, regardless of your reason. It is up to you to manage your time and get the assignment submitted ***before*** the deadline.

Your submitted assignment will consist of a single zip file. When your submitted zip file is unzipped, it should create a directory with your real name(s) (initial & last name). For example, if I was working with Albert on the assignment, unzipping will create a directory called: rneilson_awei (or something similar).

Within this directory I expect to find a directory containing all the Java source files required to build your program. Your submission must not include java SDK files, and must meet the following criteria:

a. Your "main" function must be in a class called Main, in a package called "lego", and in a subdirectory called "lego" under the directory created when your submitted file is unzipped. In the above example, this means main should be in the file rneilson_awei\lego\Main.java
b. Note that packages correspond to subdirectories. This means you need to put the line: package lego; in Main.java. You will probably need to do the same with the other source files.

How do I test your assignment?

1. I don't use NetBeans. I just compile your program using javac. Your program must compile fine when I use the following commands to compile it: `javac lego\Main.java` (assuming I am in directory rneilson_awei).
2. Next I will execute your program, passing the name of one of my input files, and redirecting the output to a file. For example: `java lego/Main testdata.txt > out.txt`

Grading and Evaluation

Your program will be graded according to the marks distribution shown below. The instructor's test data is designed to make sure you understood and followed the requirements. Don't assume anything!

Within each category, marks will be assigned for choice of data structures and algorithms, as well as correctness and efficiency of the solution(s).

Programs that do not compile will receive a grade of zero.

Programs that do not read the input file (ie: crash on input) will receive a grade of zero.

Programs that do not accept the file name as a command line argument will receive a maximum grade of 75/100.

Lots of marks will be deducted for programs that are poorly designed (ie: lack modularization and/or are confusing and difficult to understand.

Mark Distribution

Ability to follow instructions 10%

- Did you follow instructions for naming of directories, building, etc?

Program Design 20%

The program design portion of your grade will be based on (but not limited to) things like

- Choice of algorithms and data structures.
- Modularity, readability, and overall organization of the program.
- Overall efficiency of the solution and the algorithms used in the program.

Output Formatting 10%

- The output is in exactly the format specified in the assignment, with no extraneous output.

Correctness of Solution 60%

The correctness portion of your grade will be based on (***but not limited to***) things like

- Program runs without error / exception.
- Output is correct.