PROBLEM A
BULK MAILING

An organization that wishes to make a large mailing can save postage by following U.S. Postal Service rules for a bulk mailing. Letters in zip code order are bundled into packets of 10-15 letters each. Bundles may consist of letters in which all 5 digits of zip code are the same (5-digit bundles), or they may consist of letters in which only the first 3 digits of zip code are the same (3-digit bundles). If there are fewer than 10 letters to make up a bundle of either type, those letters are mailed first class.

You are to write a program to read a data set of 5-digit zip codes, one per line, until end of file. The file name is **bulk.dat**. Your program should count the number of 5-digit bundles, 3-digit bundles, and first class letters. You should include as many letters as possible in 5-digit bundles first, then as many as possible in 3-digit bundles, with as few bundles of 10 to 15 letters as possible. For example, if there are 31 letters with the same zip code, they must be combined into exactly three 5-digit bundles.

Not all zip codes in the data set will be valid. A valid zip code consists of exactly 5 digits (0-9), all of which cannot be 0. Non-numeric characters are not allowed. At the end of your output, print the invalid zip codes found. (Duplicates need only be printed once.)

Print a report that lists 5-digit zip code bundles first, with the number of letters and number of bundles for each zip code. Next list all 3-digit zip code bundles with the same two counts, followed
by all zip codes that are not bundled and to be sent first class. At the end print totals of letters and bundles, followed by the number of invalid zip codes and a list of these. Single space the report, and print blank lines following the heading, before the total line, and between the three groups of zip codes. For 3-digit bundles, print the zip codes in the form dddxx, where ddd represents the three significant digits and xx represents the last two digits to be omitted. Your output should be similar to that shown in the sample.

For example (input continues on 2nd page):

SAMPLE INPUT:

```
95864
95864
95864
95867
95920
9j876
95616
95616
95747
95814
95818
95818
8976
95818
95818
95819
95819
00000
95819
95819
95819
```

PRODUCES THIS OUTPUT:

```
ZIP          LETTERS     BUNDLES

95819          11           1
95864          10           1

958xx          25           2

95616           2           0
95747           1           0
95920           1           0

TOTALS         50           4

INVALID ZIP CODES
9j876
8976
00000
123456
```

```
95819
95819
95825
95825
95825
95825
95825
95826
95826
95826
95826
95826
95826
95827
8976
95833
95833
95833
95833
95819
95819
95819
95819
95833
95833
95833
95864
95864
95864
123456
95864
95864
95864
95864
```

PROBLEM B
IMMEDIATE DECODABILITY

An encoding of a set of symbols is said to be *immediately decodable* if no code for one symbol is the prefix of a code for another symbol. We will assume for this problem that all codes are in binary, that no two codes within a set of codes are the same, that each code has at least one bit and no more than ten bits, and that each set has at least two codes and no more than eight.

Examples: Assume an alphabet that has symbols {A, B, C, D}

The following code is immediately decodable:
A:01 B:10 C:0010 D:0000

but this one is not:
A:01 B:10 C:010 D:0000 (Note that A is a prefix of C)

Write a program that accepts as input a series of groups of records from a file named **codes.dat**. Each record in a group contains a collection of zeroes and ones representing a binary code for a different symbol. Each group is followed by a single separator record containing a single 9; the separator records are not part of the group. Each group is independent of other groups; the codes in one group are not related to codes in any other group (that is, each group is to be processed independently). For each group, your program should determine whether the codes in that group are immediately decodable, and should print a single output line giving the group number and stating whether the group is, or is not, immediately decodable.

Sample Input: (this input describes the examples above):

```
01
10
0010
0000
9
01
10
010
0000
9
```

Sample output:

```
Set 1 is immediately decodable
Set 2 is not immediately decodable
```

PROBLEM C
FILE MAPPING

It is often helpful for computer users to see a visual representation of the file structure on their computers. The "explorer" in Microsoft Windows is an example of such a system. Before the days of graphical user interfaces, however, such visual representations were not possible. The best that could be done was to show a static "map" of directories and files, using indentation as a guide to directory contents. For example:

```
ROOT
|     DIR1
|     File1
|     File2
|     File3
|     DIR2
|     DIR3
|     File1
File1
File2
```

This shows that the root directory contains two files and three subdirectories. The first subdirectory contains 3 files, the second is empty and the third contains one file.

Write a program that reads a series of data sets representing a computer file structure. The input file name is **filemap.dat**. A data set ends with a line containing a single *, and the end of valid data is denoted by a line containing a single #. The data set contains a series of file and directory names. (The root directory is assumed to be the starting point.) The end of a directory is denoted by a ']'. Directory names begin with a lower case 'd' and file names begin with a lower case 'f'. File names may or may not have an extension (such as fmyfile.dat or fmyfile). File and directory names may not contain spaces.

For example:

```
file1
file2
dir3
dir2
file1
file2
]
]
file4
dir1
]
file3
*
file2
file1
*
#
```

```
DATA SET 1:
ROOT
|     dir3
|     dir2
|     file1
|     file2
|     dir1
file1
file2
file3
file4

DATA SET 2:
ROOT
file1
file2
```

Note that the contents of any directory should list any subdirectories first, followed by files, if any. All files should be in alphabetical order within each directory. Note that each data set output is marked by the label "DATA SET x:", where x denotes the number of the set, beginning at 1. Note

also the blank line between the output data sets. Each level of indentation should show a "|" followed by 5 spaces.

PROBLEM D
The Gourmet Club

The gourmet club of ACM City has 16 members. They contracted the proprietor of the local French restaurant Chateau Java to arrange dinner parties for 5 consecutive evenings. They asked to be seated around 4 tables, 4 persons per table. They also stipulated that during the 5 evenings, every member of the club will share a table exactly once with each member of the club. Mr. I.B. Emm, the restaurateur, assigned his Maitre D' the task of scheduling the seating for the 5 evenings. On the first evening, the Maitre D' seated the members as they arrived and recorded their seating. Each subsequent evening, he carefully planned the seating to match the requirement that no member will be dining twice with some other member. Unfortunately, the Maitre D' disappeared on the morning of the fourth evening. Mr. Emm was left only with his notes which included the recorded seating arrangements during the previous 3 evenings. As he was trying to schedule the seating for the remaining evenings, it dawned on him that this task may not be that easy. He is asking for your help to try and see whether the remaining two evenings can be scheduled. The following is a sample of the Maitre D's seating arrangements during the first 3 evenings:

```
ABCD  EFGH  IJKL  MNOP
AEIM  BFJN  CGKO  DHLP
AFKP  BGLM  CHIN  DEJO
```

The members of the gourmet club were identified by the letters A,B,C,...,P.
Each line represents one evening of seating with each set of four letters a single table. Thus on the first evening A dines with B, C and D etc. Write a program that will read from the input file the seating arrangement of the first three evenings and will either complete the schedule or determine that the Maitre D' screwed up.

Input will be a text file. Each data set will be 3 lines. Each line will consist of four blocks, each 4 letters long. All letters will be in upper case. Blocks will be separated by "white space". Data sets will be separated by blank lines. For a successful schedule, echo the input and add two lines showing the successful schedule. If it is not possible to complete the schedule, do not echo the input, but print "It is not possible to complete this schedule." Separate output for each data set with a blank line.

Sample Input:

```
ABCD  EFGH  IJKL  MNOP
AEIM  BFJN  CGKO  DHLP
AFKP  BGLM  CHIN  DEJO
```

Sample Output (for given input):

```
It is not possible to complete this schedule.
```

The input data file is named **gourmet.dat.**

PROBLEM E
Chutes and Ladders

A popular board game for children is called "Chutes and Ladders". The board has squares which are numbered from 1 to 100, and players have counters which start on the theoretical square 0. The players take turns at throwing a die with the numbers 1 to 6 on it, and each moves his or her counter forward the number of squares corresponding to the number on the die (the square they reach is found by adding the die number to the square number their counter is on). The first person to reach square 100 is the winner.

The interest is caused by the fact that pairs of squares are connected together by "ladders" (which connect a lower-numbered square to a higher-numbered square) and "chutes" (which run from high to low). If a counter lands on the start of a chute or ladder (i.e., this is the square reached after throwing the die), then the counter is moved to the corresponding square at the end of the chute or ladder. Note that landing on the end square of a ladder or a chute has no effect, only the start square counts. Furthermore, there are some squares such that if a player's counter lands on them, then the player must either miss the next turn, or immediately throw the die again for another turn, depending on what is written on the board. A miss-a-turn or extra-turn square is never the start or end of a ladder or chute. If a player is on square 95 or higher, then a die throw which takes them past 100 must be ignored - thus a player on square 99 must ignore all throws which are not 1.

Input will be from a file called **chutes.dat**, and will start with a set of less than 1000 die throws which you must use for all games, starting each new game with the first player "throwing" the first number in the set, the next player "throwing" the second number, and so on. This set of die throws will simply be a list of random numbers between 1 and 6, separated by single spaces, with not more than 80 characters on each line. It will be terminated by the number 0. After this set of die throws, there will be one or more game sets. Each game set is in three parts. The first part is a line containing a single number giving the number of players in the game. This will be more than 1 and less than 6. Then the board is described, in two parts. The first part lists the ladders and the chutes on the board, each ladder or chute being defined on a single line. Each is given by two numbers, from 1 to 99,separated by one or more spaces. The first number gives the start square, and the second number gives the end square; so it is a ladder if the first number is less than the second number, and a chute if the order is the other way. The chute/ladder definitions are terminated by a line containing two 0's. The second part of the board description gives the lose-a-turn/extra-turn squares, if there are any. These are single numbers, one per line, defining the squares. If the number is negative, its positive counterpart is a lose-a-turn square; if positive, it represents an extra-turn square. (For example, -16 means that square 16 on the board is a lose-a-turn square, while a 25 means that players landing on square 25 must immediately roll again.) The end of this set of descriptions, and of the game description, is given by a single 0. The end of all the game descriptions is given by a game with the number of players equal to 0.

Output must be one line for each game in the input, giving the number of the player who wins the game. Every game will determine a winner in fewer throws than those given at the start of the data.

See example I/O on next page.

EXAMPLE

Input

```
3 6 3 2 5 1 3 4 2 3 1 2 0
2
6 95
99 1
0 0
-3
98
0
2
3 99
6 90
0 0
0
0
```

Output

```
2
2
```

PROBLEM F
STAMPS

Have you done any Philately lately?

You have been hired by the Ruritanian Postal Service (RPS) to design their new postage software. The software allocates stamps to customers based on customer needs and the denominations that are currently in stock.

Ruritania is filled with people who correspond with stamp collectors. As a service to these people, the RPS asks that all stamp allocations have the maximum number of different types of stamps in it. In fact, the RPS has been known to issue several stamps of the same denomination in order to please customers (these count as different types, even though they are the same denomination). The maximum number of different types of stamps issued at any time is twenty-five.

To save money, the RPS would like to issue as few duplicate stamps as possible (given the constraint that they want to issue as many different types). Further, the RPS won't sell more than four stamps at a time.

Input:

The input for your program will be pairs of positive integer sequences, consisting of two lines, alternating until end-of-file. The first sequence are the available values of stamps, while the second sequence is a series of customer requests. For example:

```
1 2 3 0     ; three different stamp types
7 4 0       ; two customers
1 1 0       ; a new set of stamps (two of the same type)
6 2 3 0     ; three customers
```

Note: the comments in this example are *not* part of the data file; data files contain only integers. The data file name is **stamps.dat**.

Output:

For each customer, you should print the "best" combination that is exactly equal to the customer's needs, with a maximum of four stamps. If no such combination exists, print "none".
The "best" combination is defined as the maximum number of different stamp types. In case of a tie, the combination with the fewest total stamps is best. If still tied, the set with the highest single-value stamp is best. If there is still a tie, print "tie".

For the sample input file, the output should be:

```
7 (3): 1 1 2 3
4 (2): 1 3
6 ---- none
2 (2): 1 1
3 (2): tie
```

That is, you should print the customer request, the number of types sold and the actual stamps. In case of no legal allocation, the line should look like it does in the example, with four hyphens after a space. In the case of a tie, still print the number of types but do not print the allocation (again, as in the example).