REGULATIONS

Due date: 22 April 2013, 23:59, Monday (Not subject to postpone)

Submission: Electronically. You will be submitting your program source code written in a file which you will name as the2.c through the COW web system. Resubmission is allowed (till the last moment of the due date), The last will replace the previous.

Team: There is no teaming up. The take home exam has to be done/turned in individually.

Cheating: This is an exam: all parts involved (source(s) and receiver(s)) get zero+both parts will be subject to disciplinary action.

PROBLEM

We are all familiar with back-of-the-book indexes that tell us at which pages a word we are looking for appears. Indeed this very basic indexing mechanism is also implemented for various tasks in computer science, even for the Web search. Roughly, a Web index is a mapping such that each word is associated with a list of “web page” that includes this particular word. Further information, such as the positions of the word at a web page, is also stored. An example of a very basic index is illustrated below:

<table>
<thead>
<tr>
<th>Words</th>
<th>List of pages</th>
</tr>
</thead>
<tbody>
<tr>
<td>yellow</td>
<td>[1, 3, 8, 17, 45], [5, 2, 3, 9], [10, 1, 2]</td>
</tr>
<tr>
<td>smart</td>
<td>[5, 1, 9], [9, 1, 2]</td>
</tr>
<tr>
<td>phone</td>
<td>[3, 1, 3], [5, 2, 4, 10] [9, 2, 25, 29]</td>
</tr>
</tbody>
</table>

In the above example, the entry [1, 3, 8, 17, 45] means that term “yellow” appears in web page 1 (we assume that each web page has an integer identifier) exactly 3 times, at the positions 8, 17 and 45 (i.e., assuming a page is a sequence of words starting from the position 1) in the page. Of course, in practice, there is no […] but the list is simply composed as a sequence of integers like 1, 3, 8, 17, 45, 5, 2, 3, 9, etc.

When you search a query like “smart phone”, first you find the terms in the index and then their lists are intersected, so that pages that include all query terms are determined. For the above example, the intersection would give the pages 5 and 9. Furthermore, the word positions must be checked since the words “smart” and “phone” must follow each other. In page 5, the word “smart” appears at position 9, and “phone” appears at position 10; so page 5 is in the final search result. For page 9, “smart” appears at position 2, but “phone” appears at positions 25 and 29; so page 9 is not what we look for. The final query result will be page 5.

In this THE, you are required to implement a similar indexing and searching mechanism. The corresponding input, for the example above, read to be from the standard input is:
3
yellow 12 1 3 8 17 45 5 2 3 9 10 1 2
smart 6 5 1 9 9 1 2
phone 11 3 1 3 5 2 4 10 9 2 25 29
smart phone

This is five lines of input. The input terminates with an EOF. All input fields are separated by at least one blank character.

SPECIFICATIONS

• Input layout is as follows:

First line:

is a simple integer showing the number \( n \) of words in the index (in the example we have exactly 3 words: yellow, smart and phone).

Each of the following \( n \) lines:

Information about a single word (that is a member) of the index. The line starts with the word, followed by the number \( m \) of integers that follow in this line. These \( m \) integers encode page information that word appears in.

\[\text{pageno}_1 \text{ count_of_positions} \text{ position}_1 \text{ position}_2 \ldots \text{ position}_{\text{count_of_positions}}\]

which is immediately followed (in the same line) by

\[\text{pageno}_2 \text{ count_of_positions} \text{ position}_1 \text{ position}_2 \ldots \text{ position}_{\text{count_of_positions}}\]

and so on. Note that

\[m = \sum_{\text{all pages containing the word}} (2 + \text{count_of_positions})\]

The positions in a page are given in the increasing order.

As far as the example is concerned:

In the second line, the first entry is the word (yellow), second is length of the list (12) for this word, third entry is the number of the first page (page number:1) (yellow) appears in, the fourth entry is the number of appearances of (yellow) in this page (3 times) and the following three entries are the positions: 8 17 45 at which (yellow) appears.

The list continues with two other pages. Namely (page number:5), which has (2) appearances of (yellow) at positions 3 and 9, respectively, and then (page number:10), which has (1) appearance at position 2.

In the third line, we have the word (smart) and its list.

In the fourth line, we have the word (phone) and its list.

Last line:

is the search query for your system. It can have any number of words.

• Output layout:

Your output will be the list of the numbers of the pages that include all of the words in the query and the words must strictly follow each other. That is, if the query is made up of 3 words in a row:
word₁ word₂ word₃

_pageᵢ_ will be in the result if _word₁_ appears at position (p), _word₂_ appears at position (p+1)
and _word₃_ appears at position (p + 2) in _pageᵢ_.

If any one of the query words does not appear in the index, the intersection will be empty
set, and your program should print -1, which denotes that no results can be found. If you
find a non-empty result, just print to the standard output a list of page ids (integer values)
separated by a single blank character.

- All integer values are of type int
- Words are strings containing lower/upper case letters of the ASCII table (and no blanks).
- The words can be listed in any order, in other words, the input lines are not alphabetically
  sorted.
- You will read this index from the standard input and you must store it using necessary
dynamic arrays. You cannot assume an array for some “maximum number of terms” or
a list for “maximum number of documents”! For the above example you should create
your dynamic arrays to store only 3 words (and for each word you should only allocate the
necessary space, you cannot assume a “maximum word length” and make an unnecessary
allocation of that “maximum word length”). For the word “yellow” in the above example
you should store only 12 integers, but no more!

- **Hint:** In order to store the parts of the index given above, you may need to use more than
  one dynamic arrays which have the structure: “a pointer array with elements pointing to
arrays which are of varying lengths”.

- While reading input lines, you can assume each line (also the last one) is at most 1000
  characters.

- While searching a query term in the index, your method should be case-insensitive (for
  instance, if query has “SmaRt”, it must match to the word in the index if it is written as
  either “smart”, or “SMART”, or “Smart”, or “SMart”, etc.).

- Words are unique. In other words no two input lines that introduce index information start
  with the same word.

- Any page information of any word contains at least one position.

- Any word in the index has at least one page information.

- There is no enforced order on the output list.

**EVALUATION**

- The use of struct or union is **forbidden**. Not complying will result in a zero grade.

- There will be no erroneous test input.

- All submissions will be compiled and run under strictly equal conditions on multiple data
  sets.