**Coding assignment**

Write a program that reads a file of text fragments and attempts to reconstruct the original document out of the fragments. The fragments were created by duplicating the original document many times over and chopping each copy into pieces. The fragments overlap one another and your program will search for overlaps and align the fragments to reassemble them into their original order.

 This task is based on a technique used for [genome shotgun sequencing](http://en.wikipedia.org/wiki/Shotgun_sequencing) in the [Human Genome Project](http://www.ornl.gov/sci/techresources/Human_Genome/home.shtml). Current sequencing technology can only sequence chunks of 500 or so base pairs, yet the human genome is billions of pairs long. Thus, the DNA to be sequenced is cloned several times and the clones are randomly cut into sequenceable-sized chunks. The reassembly step aligns those chunks to reconstruct the original strand. Nova had a fascinating show on [Cracking the Code of Life](http://www.pbs.org/wgbh/nova/genome/). Their online animation of a [genome sequencer](http://www.pbs.org/wgbh/nova/genome/sequencer.html) explains the process of gene sequencing. The program you will write simulates a simplified version of the assembly step.

**Greedy match and merge**

The program operates in a series of rounds. In each round, it searches the collection to find the pair of fragments with the maximal overlap match and merges those two fragments. This match/merge operation decreases the total number of fragments by one. The program repeats the match/merge operation until only one fragment remains.

Matching a pair of fragments means finding a position to align the two for the maximal overlapping match. In each round, you find the pair of fragments in the collection with longest such overlap and merge them.

Consider a collection with fragments s1-s4 shown below (extra spaces were inserted between letters for clarity):

s1: a l l i s w e l l  
s2: e l l t h a t e n  
s3: h a t e n d  
s4: t e n d s w e l l

On the first round, the longest overlap found is a 6-character overlap between s2 and s3 when aligned as below:

e l l t h a t e n  
 h a t e n d

The fragments s2 and s3 would be removed and replaced with their merged result s5:

s5: e l l t h a t e n d

The new, merged fragment is a candidate for future rounds; the two fragments it was composed from are no longer considered. On the next round, the longest overlap is 5 characters between s5 and s4 aligned as below:

e l l t h a t e n d  
 t e n d s w e l l

The fragments s5 and s4 would be removed and replaced with their merged result s6:

s6: e l l t h a t e n d s w e l l

The last round merges s1 and s6 in their maximal overlap alignment of 3 characters:

a l l i s w e l l  
 e l l t h a t e n d s w e l l

The one remaining fragment is the final result:

a l l i s w e l l t h a t e n d s w e l l

A match is also possible when one fragment is completely contained within another. Consider:

s1: s w e l l t h a  
s2: e l l

The entire fragment s2 is contained within s1. When these two fragments are merged, the result is simply s1.

Your program can break ties arbitrarily. If several pairs have the same maximal length overlap, choose whichever pair you like to merge. If there are two equally maximal alignments for a pair (e.g. abxy and xyab can merge to either abxyab or xyabxy), you can choose either. If two fragments have no overlap, their merge is simply the concatenation of the two strings, which can be done in either order. Because of differences in how ties are broken, some inputs can have more than one possible correct reconstruction. However, most fragment collections have such extensive redundancy that there is exactly one correct reassembled result.

 Note that all characters in the overlap must match exactly in sequence. Unlike DNA strands which have "noisy" data in which gaps and mutations must be accommodated for, the sequences in our simplified process are always "clean" and must match perfectly to be merged.

**A little computer science theory**

Optimal reassembly is an example of the *shortest common superstring* problem, a classic problem in theoretical computer science. Given a set of strings S, find the shortest string that contains all the strings in S as substrings. The solution for shortest superstring is *NP-hard*, which means it is believed that no polynomial (i.e. "efficient") solution exists. Our approach is a *greedy strategy* because it finds a local maximum, the longest overlap, and assumes making this locally optimal choice will eventually lead to the globally optimal result. It might, but it might not. Our approach will find a common superstring, but it is not guaranteed to be the shortest. We are not asking you to guarantee an optimal result -- that is a more difficult problem and its solution would run much more slowly.