Explanation

[Read](http://rosalind.info/problems/grph/explanation/) [Edit](http://rosalind.info/problems/grph/explanation/edit/) [View history](http://rosalind.info/problems/grph/explanation/history/) [Get source](http://rosalind.info/problems/grph/explanation/?raw=1)

[Overlap Graphs](http://rosalind.info/problems/grph/)

Last modified on June 17, 2015, 8:04 p.m. by [Michael Beaver](http://rosalind.info/users/mbeaver502/).

**Algorithm idea**

For this exercise you should check every suffix from each sequence of length 3 and check for matching prefixes in every other sequence. The simplest way would be to cycle through every possible sequence twice

**\*\*For\*\* each sequence do \*\*seq[end-k:end]\*\* # Slice the suffix**

**\*\*For\*\* each other sequence do \*\*seq[0:k]\*\* # Slice the prefix**

**\*\*If\*\* sequences are the same then skip**

**\*\*If\*\* \*\*suffix == prefix\*\* then \*\*print name1+’ ‘+name2\*\***

The main challenge is to keep track of the sequences and names, so the correct data structure should be chosen to maintain the name associated to the sequence being tested. This can be easily achieved by using lists and tuples. In graph theory, a digraph edge is represented by the pair of nodes connected: (v,w)(v,w), where vv is the start and ww is the end. It is intuitive, then, that the tuple data structure is used to represent this relationship.

**Complexity analyses and correctness**

The algorithm requires comparing n prefixes for each suffix, thus n2n2 comparisons have to be made, where nn is the number of input sequences. Though, remember that part of the requirement for an overlap graph is that a node is not compared to itself (a directed loop). Thus, we should not check a DNA string's suffix against the prefix of the same DNA string. This would violate the definition of the overlap graph. While this cuts down on some comparisons, the algorithmic complexity is still near O(n2)O(n2).

**Any advanced information**

Since we are examining these graphs essentially as tables/matrices, O(n2)O(n2) is almost a necessity. Solutions with better algorithmic complexity may exist, however.