Code Jam 2016 - Round 1B

Analysis: Technobabble

Technobabble: Analysis

Small dataset

Each topic on the list could be "faked" or "un-faked." One natural brute force solution is to enumerate all possible orderings of the topics, and pick the one that has the most faked topics.

It's easy to check whether a topic could have been faked: simply check whether the first word of the topic appears as a first word earlier in the list, and the second word of the topic appears as a second word earlier in the list. Note that marking a topic as faked does not change whether or not topics later in the list can be marked as faked, so using this strategy, it is optimal to always count a topic as faked if possible.

However, there are **N**! possible orderings, which is too large to enumerate even for the small dataset where **N** ≤ 16 (16! is on the order of 21 trillion). We need a better approach. Rather than trying to maximize the number of faked topics, let's think about the reverse problem: trying to minimize the number of un-faked topics.

The key observation is that any possible set of un-faked topics must contain every first word at least once and every second word at least once — otherwise, there would be a faked topic that contained a word that was not available for the faker to use. Conversely, any set containing every first word at least once and every second word at least once could be a possible set of un-faked topics — simply put all the topics from the un- faked set at the top of the list. So, the question we need to answer is this: *What is the smallest set of topics that contains every first word at least once and every second word at least once?*

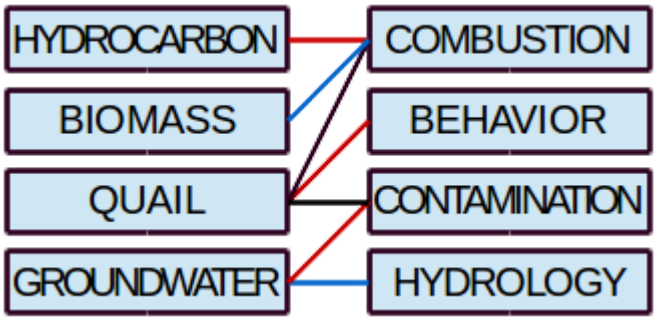
A brute-force approach, which works for the Small dataset, is to enumerate all subsets of topics and pick the subset with the fewest topics that covers every first word at least once and every second word at least once. Since there are $2^{\mathbf{N}}$ subsets, this solution runs in exponential time, which is fine for the Small dataset (2^{16} = 65,536).

Large dataset

For the large dataset, the exponential time solution will not work. It turns out that there is a polynomial time solution to the problem. We will illustrate the solution using graph theory.

Let each word be a vertex in a bipartite graph in which each topic is an edge connecting two vertices. The sample input below corresponds to the following graph (we'll explain the colors of the edges in a moment).

```
HYDROCARBON COMBUSTION
BIOMASS COMBUSTION
QUAIL COMBUSTION
QUAIL BEHAVIOR
QUAIL CONTAMINATION
GROUNDWATER CONTAMINATION
GROUNDWATER HYDROLOGY
```



The problem we're trying to solve on this graph is the *minimum edge cover*; that is, finding the smallest set of edges such that each vertex is connected to at least one edge. This corresponds to the smallest set of topics containing every first word at least once and every second word at least once.

The minimum edge cover problem is related to finding a *maximum matching* of a graph (the largest set of edges without any common vertices): the two will always have the same number of connected components. If it's not immediately obvious why this is the case, convince yourself by drawing some graphs on paper and trying to come up with a counter-example. We can additionally observe that every vertex left out of a maximum matching must be connected to a vertex in the maximum matching; otherwise, we could have added that pair to the maximum matching. With these facts, we can use a two-step algorithm to compute a minimum edge cover on our bipartite graph:

1. Find a maximum cardinality bipartite matching of the graph, which can be done in polynomial time using an approach such as the Ford-Fulkerson algorithm or the Hopcroft–Karp algorithm. One such matching is shown above in red.
2. Iterate over the remaining edges, and greedily add edges that connect to an unused vertex. The edges added by this step are shown above in blue.

In fact, all we really need to know is the *size* of a minimum edge cover: the number of edges in a maximum matching plus the number of vertices not included in a maximum matching. The solution to the problem is then simply the total number of edges (topics) minus the size of a minimum edge cover.