Fixed Parameter algorithm implementation

- The fixed parameter implementation of the the 'k' size vertex cover problem is in **python**.
- The program uses three external libraries for computations:
  1. **sys** for standard input reading from and writing into a file
  2. **itertools** for generating 'k' sized subsets from the vertices set
  3. **time** for measuring the time to run the main function (time for generating the graph, vertices and edges lists from the input is excluded)

Functions:

1. checkNoEdges

Input: a graph (a dictionary of lists)

Computation: A loop iterates over the graph and checks if the length of each list of adjacent vertices of each vertex in the graph is zero. If lengths of all the vertex lists is zero, it returns True (Graph has no edges). Otherwise, it returns False.

Output: True / False

2. numberOfEdges

Input: a graph (a dictionary of lists)

Computation: A loop iterates over the graph and counts the length of each list of adjacent vertices of each vertex in the graph. If then sums up all the list lengths and returns half of that sum value (each edge is counted twice).

Output: An integer (number of edges in the graph)

3. removeVertex

Input: a graph (a dictionary of lists) and a list (list of vertices to be removed from the graph)

Computation: A loop iterates over the vertex list and searches for the vertex in the graph and adjacency lists of each vertex in the graph. If it is found, it is deleted from the graph. Similarly, the edges that are covered by the vertex are also deleted in the edge list. The resultant graph is returned

Output: a graph (a dictionary of lists)

4. checkIfVertexCover

Input: a list of vertices (lst) and the edge list (edges)

Computation: A loop iterates over the vertices list and removes all the edges in the edge list that the particular vertex covers. When the loop ends, if the edge list is empty, it returns True (the vertices cover the entire edge set). Otherwise, it returns False.

Output: True / False

5. reduceGraph

Input: a graph (a dictionary of lists) and an integer k (VC parameter)

Computation: This function applies the rules of kernelization and removes all appropriate vertices to produce the kernel instance of the original graph. Then it passes the kernel of the graph and the resultant value of k to the function vertexCover.

6. vertexCover

Input: a list of vertices (lst) and the edge list (edges)

Computation: This function starts the timer and uses the itertools library and generates a list (allPossibleKCombinations) of all possible subsets of k-size from the vertices set of the kernel of the graph. Then it passes each set of allPossibleKCombinations to the checkIfVertexCover function and checks if each of the k-size vertices set is a vertex cover of the original graph. As soon as it encounters that a particular instance of the k-sized vertices is a vertex cover, it breaks out of the loop and prints 'True' along with the solution (set of vertices). It also displays the time (in seconds) for which which the main function got executed.

Output: True / False

7. Main function

The program accepts the input file name as a command line input. It then reads through the file line by line and generates the graph, edges and vertices lists.

It then calls the reduceGraph function on the input graph and paramter k that returns True if the graph has a vertex cover of size K.

Execution:

To execute the program:

python fpt.py 'input.txt' > 'output.txt'