Data Structures	Assignment 4
CCO 2103-01, Spring 2023	Due at 5:00pm, Friday, June 2

Hand in your solutions electronically using LearnUs. As was announced in class, you are not allowed to use any data structure libraries, including the linked lists, stacks, queues, trees, binary search trees, priority queues, and graphs provided by JDK. When in doubt, contact the course staff.

Submit your source code(s), zipped as `yourStudentID.zip`. For example, if your student ID is `2022000000`, then you must zip all your source code(s) into `2022000000.zip` and submit this file. Each class should have its own `.java` file, of which the filename is the same as the class name. Do *not* include your student ID as part of the class names.

This assignment consists of one programming task, which will count as 6/10 of an assignment when the final grade is calculated.

(1) (60 points) Given a connected simple undirected graph $G = (V, E)$, we say its edge $f \in E$ is *redundant* if the graph remains connected even after deleting f. Note that deleting an edge is different from deleting its endpoints: deletion of an edge does not change the graph's vertex set.

Write a Java method that, given a connected simple undirected graph, determines whether each edge of the graph is redundant.

Your code must have a class named `As41` that has a ***static*** method called `getRed()`. Its declaration is as follows:

```
public static boolean[] getRed(int n, int m, int[] x, int[] y);
```

- The parameter `n` is the number of vertices. The vertices are numbered from 0 to $\texttt{n} - 1$.
- The parameter `m` is the number of edges. The edges are numbered from 0 to $\texttt{m} - 1$.
- The parameters `x` and `y` specify the edges. For each `i` ($0 \leq \texttt{i} \leq \texttt{m} - 1$), `x[i]` and `y[i]` are (the numeric IDs of) the two endpoints of Edge `i`.
- Your program must allocate and return a `boolean` array of size `m`. For each `i` ($0 \leq \texttt{i} \leq \texttt{m} - 1$), your program must set the `i`-th entry of the returned array to `true` if Edge `i` is redundant and `false` otherwise.

You can assume that $1 \leq \texttt{n}, \texttt{m} \leq 200,000$.

You are free to add your own methods, fields, and classes as you see fit, but you need to implement all of those.

To ease your testing, we will provide a skeleton code. The skeleton code reads the input from `input.txt` in the current working directory, and outputs the answer reported by your code, as is, to `output.txt` in the current working directory.

The first line of `input.txt` contains `n` and `m`, in that order, separated by a space. Each of the following `m` lines of the file contains (the numeric IDs of) the two endpoints of each edge, also separated by a space. The input therefore consists of $m + 1$ lines in total.

The output file consists of `m` lines. For $0 \leq i \leq m - 1$, the $(i + 1)$-st line is `redundant` if your program reported that Edge `i` is redundant; `not redundant` otherwise. The output file will contain a single line containing `error` if your program does not return a boolean array of size at least `m`.

The entry point of the skeleton code is `As41Driver.main()`. Your method must return within 2 seconds on the TA's computer. To help your testing, the skeleton code will output the total amount of time elapsed to the standard output. This time, of course, does not include the file I/O time. (The skeleton code provided does not terminate your code even if it spends more than 2 seconds.)

Submit all your source code(s), including `As41.java` and any other source files you created. Note that `As41Driver.java` is the skeleton code and you must not submit this file. In other words, you must ***not*** modify this skeleton code. When we grade your code, we will use a different driver to test your solution.

Example 1

`input.txt`

```
3 2
0 1
1 2
```

`output.txt`

```
not redundant
not redundant
```

Example 2

`input.txt`

```
3 3
0 1
1 2
2 0
```

`output.txt`

```
redundant
redundant
redundant
```