CS 237	Discrete Mathematics I

Awareness

Graphs *Exploration*

Goal

The purpose of this exploration is for you to discover how to deal computationally and cognitively with graphs, and become aware of certain graph properties and metaproperties.

Requirements

Now that you know what complete (K_n) graphs are and what subgraphs are, it's time to apply that knowledge. As I state in my manifesto:

> *Allow that there is more than one way to say the same thing. Don't be confused by alternative phraseology, but learn the words and the wording used in this field of study.*

The alternative phraseology for a complete graph is a word you're all familiar with, except in high school you might have pronounced it as *click* instead of the correct way that rhymes with the second syllable of *antique*. The word, of course, is *clique*.[1]

Your task is to write some C++ code that *verifies* this graph property. To save you the trouble of looking up the definition, a clique is a subgraph of a given graph in which every two vertices are connected by an edge. An **anti**-clique is a subgraph in which every two vertices are **not** connected by an edge. (Note that this is the same as saying that **no** two vertices in this subgraph are connected. Or in other words, they form an *independent set* of vertices—vertices that are all independent of each other.) Searching through a specified graph, your code will check the alleged "clique-ness" or "anti-clique-ness" of a given list of vertices.

Graphs will be read in from a file (specified on the command line) containing up to one million (1,000,000) edges! Each line of the input file represents one edge, and consists of two nonnegative integers that represent the vertices of the edge. All graphs will be connected, and the vertex numbers will be contiguous from 0 to n, where n is some number less than or equal to 60,000.[2]

Code in the provided `checkcliqueStub.cpp` file handles reading an input file into a Graph object, but does not handle error conditions[3] on the grounds that they are absent.

Test input files are in the directory `/home/cs237/files/`. The files are named `graph1.in`, `graph2.in`, `graph3.in`, `graph4.in`, `graph5.in`, and `graph6.in` — representing successively bigger and more complex graphs.

Your specific task is to flesh out the function named `checkCliqueOrAntiClique` per the instructions (see the `TODO:`) in the `checkcliqueStub.cpp` file. Your output must be in the following format (not the right numbers, just an example):

[1] Rosen postpones mentioning this term until the end of the chapter, in the Supplementary Exercises, numbers 11-13 on page 678.
[2] For example, a file will not have vertices 1, 2, 3, 4, 5 and 10. The vertices may not be in order in the file, however.
[3] Self-loops or redundant (directed) edges. For example: **3 3** is a self loop. **3 5** and **5 3** are the same edge.

```
The graph specified in /home/cs237/files/graph1.in
does contain a clique of size 6 with vertices
3 5 9 30 100 129

The graph specified in /home/cs237/files/graph2.in
does contain an anti-clique of size 10 with vertices
2 4 6 8 10 12 14 16 18 20

The graph specified in /home/cs237/files/graph3.in
DOES NOT contain a clique of size 6 with vertices
3 5 9 30 100 129

The graph specified in /home/cs237/files/graph4.in
DOES NOT contain an anti-clique of size 10 with vertices
2 4 6 8 10 12 14 16 18 20
```

If conditions are right in the Linux Lab, you can build, test, and be reminded of how to submit your code and write-up via the command:

```
make it just so
```

Objectives

There are several objectives for this exploration. In your write-up, your focus will be on *making a good connection* between something you learned and each of the eight objectives below (the first three of which are from the course objectives found in the syllabus):

1. Master the basic terminology and operations of logic, sets, functions and graphs.
2. Demonstrate logical reasoning through solving problems.
3. Interpret the meaning of mathematical statements in the context of real-world applications.
4. Discern between effective and ineffective approaches to problem solving.
5. Solve problems using limited or constrained resources.
6. Synthesize new problem solving concepts by putting old concepts together in novel ways.
7. Recognize the relative importance of different elements of a computer science problem.
8. Know how discrete mathematics applies to all parts of computer science.

This is not limited to what you learned while exploring graphs with this exploration alone, but includes the whole semester's learning activities.

Grading Criteria

The rubric below is meant to guide you in the production of a solution of exceptional quality.

	Exceptional 100%	**Good 90%**	**Acceptable 70%**	**Developing 50%**	**Missing 0%**
Application — using what you've learned 30%	**Good**, plus you made **great** connections for all eight objectives.	**Acceptable**, plus write-up has **good** headings for the introduction, the 8 objectives' sections, and the conclusion.	**Developing**, plus fixed any typos, and report is in the required format, per the usual templates. In addition, you made connections for some of the 8 objectives.	Acquired new knowledge and figured out how it applies. However, write-up has grammar or spelling mistakes, or other mechanical infelicities.	No application of anything learned. No write-up (aka report).
Correctness/ Completeness 40%	**Good**, plus code passes the four tests for `graph5.in` and `graph6.in`.	**Acceptable**, plus code passes the four tests for `graph3.in` and `graph4.in`.	Code passes the four tests for `graph1.in` and `graph2.in`.	Code compiles and runs without crashing (e.g., no segmentation faults), and has fleshed-out stubs (see TODO comment in stub code provided).	Code does not even compile.
Elegance 30%	Code is correct, efficient and cohesive, and has minimal repetitious code.	Code is correct, efficient and cohesive.	Code is correct and efficient.	Code is correct, as first and foremost, an elegant solution is a correct solution.	No elegance whatsoever. The code is not even correct.