Title:	Methods and Tools for Software Engineering
Course ID:	ECE 650 Section 001
LEARN:	`https://learn.uwaterloo.ca`
Piazza:	`https://piazza.com/uwaterloo.ca/fall2024/ece650/home`
Lectures:	TTh 5:30 – 6:50 PM EST
Instructor:	Dr. Albert Wasef, `awasef@uwaterloo.ca`
TA:	Josh Sun, `q84sun@uwaterloo.ca`
TA:	Leo Zhang, `l536zhang@uwaterloo.ca`

Office hours by appointment. Begin all email subjects with `[ECE650]`. Use **Piazza** instead of email whenever possible!

Assignment 4 - Due Monday, November 11

The skeleton for this assignment is available at the master branch of `https://git.uwaterloo.ca/ece650-f24/skeleton` in directory `a4`. Follow the instructions in Assignment 0 to correctly fetch and merge the files from the skeleton!

For this assignment, you are to augment your code from Assignment 2 to solve the minimal Vertex Cover problem for the input graph. Your approach is based on a polynomial time reduction to CNF-SAT, and use of a SAT solver. The following are the steps you should take for this assignment.

SAT Solver

We will be using MiniSat SAT solver available at `https://git.uwaterloo.ca/ece650-common/minisat.git`

MiniSat provides a `CMake` build system. You can compile it using the usual sequence:

```
cd PROJECT && mkdir build && cd build && cmake ../ && make
```

The build process creates an executable `minisat` and a library `libminisat.a`. You will need to link against the library in your assignment.

Play around with it. Sample files are available in the `https://git.uwaterloo.ca/ece650-common/minisat-example` repository on GitLab.

Incorporate SAT

Create a polynomial reduction of the decision version of VERTEX COVER to CNF-SAT. We have discussed the reduction in class. It is also available under the name `a4_encoding.pdf` on LEARN. You are allowed to use your own reduction provided it is sound and polynomial-time. Implement the reduction and use `minisat` as a library to solve the minimum VERTEX COVER problem for the graphs that are input to your program (as in Assignment 2).

As soon as you get an input graph via the 'V' and 'E' specification you should compute a minimum-sized Vertex Cover, and immediately output it. The output should just be a sequence of vertices in increasing order separated by one space each. You can use `qsort(3)` or `std::sort` for sorting.

Assuming that your executable is called `ece650-a4`, the following is a sample run of your program:

```
$ ./ece650-a4
V 5
E {<1,5>,<5,2>,<1,4>,<4,5>,<4,3>,<2,4>}
4 5
```

The lines starting with `V` and `E` are the inputs to your program, and the last line is the output. Note that the minimum-sized vertex cover is not necessarily unique. You need to output just one of them.

Marking

We will try different graph inputs and check what vertex cover you output. We will only test your program with syntactically and semantically correct inputs.

- Marking script for compile/make etc. fails: automatic 0
- Your program runs, awaits input and does not crash on input: + 20
- Passes Test Case 1: + 25
- Passes Test Case 2: + 25
- Passes Test Case 3: + 25
- Programming style: + 5

CMake

As discussed below under "Submission Instructions", you should use a `CMakeLists.txt` file to build your project. We will build your project using the following sequence:

```
cd a4 && mkdir build && cd build && cmake ../
```

If your code is not compiled from scratch (i.e., from the C++ sources), you get an automatic 0.

Submission Instructions

You should place all your files in your GitLab repository in directory `a4`. The directory should contain:

- All your C++ source-code files.
- A `CMakeLists.txt`, that builds your C++ executable `ece650-a4`.
- A file `user.yml` that includes your name, WatIAM, and student number. Note that *WatIAM* is the user name for your Quest account, e.g. `awasef`, and a *student number* is an 8-digit number, e.g. `20397238`.

See `README.md` for any additional information.

You should assume that MiniSat will be placed in directory `a4/minisat`. Your submission should include **only** your own code (including code provided by us in the skeleton). If your code is not compiled from scratch (i.e., from the C++ sources), you get an automatic 0.

A Polynomial-Time Reduction from VERTEX-COVER to CNF-SAT

A *vertex cover* of a graph $G = (V, E)$ is a subset of vertices $C \subseteq V$ such that each edge in E is incident to at least one vertex in C.

VERTEX-COVER is the following problem:

- Input: An undirected graph $G = (V, E)$, and an integer $k \in [0, |V|]$.
- Output: True, if G has a vertex cover of size k, false otherwise.

CNF-SAT is the following problem:

- Input: a propositional logic formula, F, in Conjunctive Normal Form (CNF).
  That is, $F = c_1 \wedge c_2 \wedge \ldots \wedge c_m$, for some positive integer m. Each such c_i is called a "clause". A clause $c_i = l_{i,1} \vee \ldots \vee l_{i,p}$, for some positive integer p. Each such $l_{i,j}$ is called a "literal." A literal $l_{i,j}$ is either an atom, or the negation of an atom.
- Output: True, if F is satisfiable, false otherwise.

We present a polynomial-time reduction from VERTEX-COVER to CNF-SAT. A polynomial-time reduction is an algorithm that runs in time polynomial in its input. In our case, it takes as input G, k and produces a formula F with the property that G has a vertex cover of size k if and only if F is satisfiable.

The use of such a reduction is that given an instance of VERTEX-COVER that we want to solve, (G, k), we use the reduction to transform it to F, and provide F as input to a SAT solver. The true/false answer from the SAT solver is the answer to the instance of VERTEX-COVER. Assuming the SAT solver works efficiently (for some characterization of "efficient"), we now have an efficient way of solving VERTEX-COVER. Furthermore, the satisfying assignment from the SAT solver can be used to re-construct the solution to VERTEX-COVER.

The reduction

Given a pair (G, k), where $G = (V, E)$, denote $|V| = n$. Assume that the vertices are named $1, \ldots, n$. Construct F as follows.

- The reduction uses $n \times k$ atomic propositions, denoted $x_{i,j}$, where $i \in [1, n]$ and $j \in [1, k]$. A vertex cover of size k is a list of k vertices. An atomic proposition $x_{i,j}$ is true if and only if the vertex i of V is the jth vertex in that list.
- The reduction consists of the following clauses
  - At least one vertex is the ith vertex in the vertex cover:

    $$\forall i \in [1, k], aclause(x_{1,i} \vee x_{2,i} \vee \cdots \vee x_{n,i})$$

  - No one vertex can appear twice in a vertex cover.

    $$\forall m \in [1, n], \forall p, q \in [1, k] with p < q, aclause(\neg x_{m,p} \vee \neg x_{m,q})$$

    In other words, it is not the case that vertex m appears both in positions p and q of the vertex cover.
  - No more than one vertex appears in the mth position of the vertex cover.

    $$\forall m \in [1, k], \forall p, q \in [1, n] with p < q, aclause(\neg x_{p,m} \vee \neg x_{q,m})$$

  - Every edge is incident to at least one vertex in the vertex cover.

    $$\forall \langle i, j \rangle \in E, aclause(x_{i,1} \vee x_{i,2} \vee \cdots \vee x_{i,k} \vee x_{j,1} \vee x_{j,2} \vee \cdots \vee x_{j,k})$$

The number of clauses in the reduction is $k + nk2 + kn2 + |E|$.