# CS/SE 2XB3 — Lab 9

Due Sunday, April 11th, 11:59pm

---

Submit the lab on Avenue. Note, Avenue must receive your lab by the due date. **Do not leave your submission to the last minute! Late submissions, even by seconds, will not be graded.** You have been warned.

This lab is worth (59/7)% of your final grade in the course. Read this document carefully and completely. **Whenever I ask you to discuss something, I implicitly mean to include that discussion in your lab report. Furthermore, when I ask you to run an experiment or evaluate the performance of a function/method include scatter plots where appropriate. Do not import external libraries, especially those containing data structures you are meant to implement!** Include all timing experiments in your `code.py` file.

## Purpose

The goals of this lab are as follows:

1. Explore the Vertex Cover problem and understand its challenges
2. Devise and implement two approximations to the Vertex Cover problem
3. Empirically analyse approximation and make conclusions regarding how good an approximation it is

## Submission

**Note: The due date of this lab is later than usual.** With the term coming to an end we have a bit more flexibility and I do not mind giving you some extra time to work on the remainder of the labs. The labs will still be posted each weekend.

You will submit your lab via Avenue. You will submit your lab as three separate files:

- `code.py`
- `vertex_cover.py`
- report.pdf (or docx, etc.)

Keep the sources code of all empirical experiments you run in `code.py`. Only one member of your lab group will submit to Avenue – see the section below for more details on that. Your report .pdf will contain all information regarding your group members, i.e. name, students number, McMaster email, and enrolled lab section. This will be given on the title page of the report.

For the remainder of this document, read carefully to gauge what other material is required in the report. Your report should be professional and free from egregious grammar, spelling, and formatting errors. Moreover, all graphs/figures in your report should be professional and clear. You may lose grades if this is not the case. Organize the report in a such a way to make things easy for the TA to grade. You are not doing yourself any favors by making your report difficult to mark!

## Testing

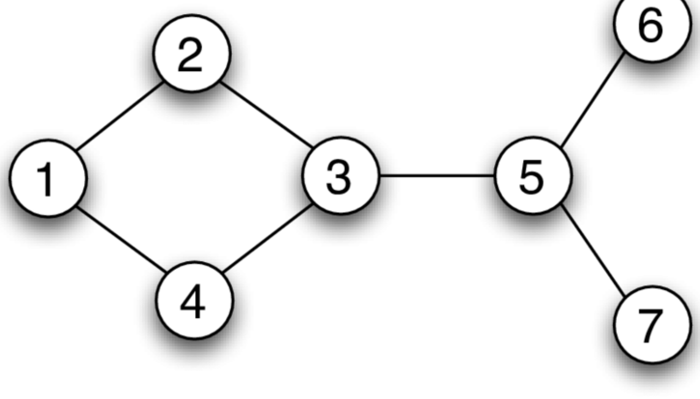
As you will see in lecture (or find out on your own). Testing the correctness and/or the quality of your Vertex Cover approximations is challenging. This is because determining a minimum Vertex Cover of a graph even with a moderate number of nodes (say 20) is taxing. When initially testing your algorithms, stick to smaller graphs so you can compare it to a known minimum Vertex Cover. Then move on to testing it on larger graphs. Here, you will not necessarily know when the minimum Vertex Cover is, but you can ensure your algorithm is returning a valid answer.

## Vertex Cover

We will be reviewing the vertex cover problem in lecture this week. However, it is relatively straight forward to understand. Given an undirected unweighted graph $G = (V, E)$, a subset of $V$, call it $C$, is a vertex cover of $G$ if and only if:

$$\forall (v_1, v_2) \in E : v_1 \in C \text{ or } v_2 \in C.$$

Informally, all edges in the graph must be *covered* by $C$. An edges is covered by $C$ is its source or destination node (or both) is in $C$. As an example, in the graph below



the following would all be valid vertex covers:

- $\{1, 2, 3, 4, 5, 6, 7\}$
- $\{2, 3, 4, 6, 7\}$
- $\{1, 3, 5\}$
- $\{2, 4, 5\}$

Note, the final two vertex cover above are minimum vertex covers. Furthermore, the following are **not** vertex covers

- $\{\}$
- $\{1, 3, 6\}$
- $\{1, 2, 4, 6, 7\}$

In `lab10.py` you will find a function `vertex_cover()` which takes in a graph and returns a minimum vertex cover of that graph as a list of nodes. As mentioned previously, this function is impractical for even moderately sized graphs.

## Approximation Implementations 50%

It is your job to develop and analyse two different approximations to the vertex cover problem. By an approximation I mean an algorithm which takes in a graph and returns a vertex cover as a list of nodes, but that vertex cover may not be a minimum vertex cover.
As an example, an approximation may behave as follows:

1. Initial $C$ to $\{\}$
2. Add a random node which is in $V$ but not in $C$ to $C$
3. Check if $C$ is a vertex cover, if it is return it, else go to step 2.

As you can see, the above will always return a valid vertex cover, but the vertex cover it returns may be far from optimal. Your approximations must:

1. Be somewhat intelligent, straight up returning $V$ will not count
2. Run in polynomial time

Implement your approximations as `vc_approx1(G)` and `vc_approx2(G)` in your `vertex_cover.py` file. Furthermore, **in your report give a plain English description of how your approximations work.** You may also be interested to learn that if you find an algorithm to find a minimum vertex cover in polynomial time there is a $1,000,000 prize for you (no joke).

## Approximation Experiments 50%

Often times with approximation algorithms it can be formally shown that they will always be within a certain factor of the optimal. For example, if in general I have an optimal vertex cover, $C^*$, for a some graph $G$, and my approximation returns a vertex cover $C$ for the same graph, perhaps I can guarantee that $|C| \leq 2|C^*|$. If this were the case, I would call my algorithm a "2-Approximation".

I am purposefully leaving this somewhat open ended, but I want you to run some experiments to determine how good or bad your approximations actually are. You could do this by thinking about what sort of graphs could really screw up your algorithms. Or you could run some tests on some random graphs and compare them to an optimal solution. You of course do not need to prove anything formally, but your conclusions should be backed up by empirical data. Include your experiments, discussion, and conclusions in your report.