# Assignment 9: Solve an NP-Hard Problem

## Learning Outcomes

* Solve an NP-Hard problem.

## Motivation

This isn't our first homework assignment involving an NP-Hard problem. 0-1 Knapsack and Sudoku are also NP-Hard. Just what does NP-Hard mean? It means that our best known deterministic algorithms aren't in polynomial time.

What does it mean for an algorithm to be "deterministic" or "non-deterministic"?

* A deterministic algorithm works the same way every time regardless of the data thrown at it. They can be reliably measured. The simple linear search or Merge Sort are both deterministic algorithms. There's no way to contrive a dataset to make Merge Sort bad. (I love Merge Sort.)
* A non-deterministic algorithm isn't reliable in terms of its execution time. Quicksort isn't reliable because a dataset can be contrived to make the execution time extremely slow (but Quicksort is still a polynomial time algorithm). Our Sudoku solver isn't reliable because it depends on random numbers. Our Dynamic Programming 0-1 knapsack solution depends on a small knapsack weight or the algorithm can be extremely inefficient. Radix Sort depends on the largest integer having a small number of digits.

Finally, what's the difference between NP-Hard and NP-Complete?

* NP-Complete is a special class of NP-Hard problems where no one has proved beyond the shadow of a doubt that there might be a deterministic polynomial time algorithm to solve these problems.
* Every problem known to be NP-Complete can be translated into any other NP-Complete problem in polynomial time.
* We can verify solutions to these problems in polynomial time. For example, you can verify a Sudoku solution in linear time.
* We do know of non-deterministic polynomial-time algorithms. "NP" stands for "Non-deterministic polynomial-time".

0-1 Knapsack, Sudoku, SAT, and Max Clique are all NP-Complete problems. When you hear that a problem is "NP-Complete", it means that it's roughly the difficulty as every other NP-Complete problem.

## Maximum Clique

In graph theory, a graph is made up of vertices and edges. In a symmetric or undirected graph, if a graph connects two vertices in one direction, it is automatically assumed that they are connected in the opposite direction also. These two vertices are called **adjacent**.

If two vertices are adjacent, then they form a **clique**. More vertices can join the clique as long as they all are adjacent to every other member in the clique. If a vertex is adjacent to some of the members (but not all) in a clique, then that vertex is not in the clique.

* A clique of 1 has 0 edges.
* A clique of 2 has 1 edge.
* A clique of 3 has 3 edges.
* A clique of 4 has 6 edges.
* A clique of 5 has 10 edges.
* A clique of has edges.

A Maximum Clique is the largest subset of vertices that form a clique.

## Instructions

Solve the Maximum Clique problem by listing the vertices in the Maximum Clique. You can model your solution based on any approach seen this year. I personally studied my brute force 0-1 Knapsack solution before writing my solution for this problem. My solution is 78 lines (not including the matrix code) and uses a brute force approach. Wish to use a non-deterministic polynomial-time algorithm? Go for it.

## Inputs

* The input will represent a graph in edge list form. The first integer is the number of vertices. The second integer is the number of edges. The next lines will contain the edge information.
* Each edge will be represented by a starting vertex, an ending vertex, and a weight value. (In this case, the weight value is always going to be 1. I did this to easily reuse my matrix code from earlier in the semester.)
* The supplied graph to my matrix library is not assumed to be symmetric, which means I give each edge two you twice (once in a -> b form and again in b -> a form).
* The graphs are also not going to be reflexive. The diagonal should be all zeros. Your algorithm shouldn't care about reflexive links.

## Outputs

You should display the list of vertices which make up the Maximum Clique. It is possible for some of these examples to have multiple solutions. If you have the same number of vertices as I have in my results, it means you are doing fine.

## Helpful tools

I created my own randomly-generated graph creation script Python script to properly test my algorithm. I'll supply a few test cases and the code for that script.

## Examples.

Example 1. Three Connected Points

Here are three connected points.

3  
6  
0 1 1  
1 0 1  
0 2 1  
2 0 1  
1 2 1  
2 1 1

Results.

[ [ 0 1 1 ]  
 [ 1 0 1 ]  
 [ 1 1 0 ] ] 3x3  
  
  
Vertices in Max Clique:  
0 1 2

Example 2. Three Points,Not Fully Connected.

3  
4  
0 1 1  
1 0 1  
0 2 1  
2 0 1

Results.

[ [ 0 1 1 ]  
 [ 1 0 0 ]  
 [ 1 0 0 ] ] 3x3  
  
  
Vertices in Max Clique:  
0 2

Example 3. Three Points, No Edges

3  
0

That's the entire file. Here are the results. A single point is a clique of 1.

[ [ 0 0 0 ]  
 [ 0 0 0 ]  
 [ 0 0 0 ] ] 3x3  
  
  
Vertices in Max Clique:  
2

Example 4. The House

This has 5 points and if you draw it on paper, it kinda looks like a house.

5  
16  
0 1 1  
1 0 1  
0 4 1  
4 0 1  
1 2 1  
2 1 1  
1 4 1  
4 1 1  
1 3 1  
3 1 1  
4 2 1  
2 4 1  
2 3 1  
3 2 1  
4 3 1  
3 4 1

Four of the points in this graph are fully connected.

[ [ 0 1 0 0 1 ]  
 [ 1 0 1 1 1 ]  
 [ 0 1 0 1 1 ]  
 [ 0 1 1 0 1 ]  
 [ 1 1 1 1 0 ] ] 5x5  
  
  
Vertices in Max Clique:  
1 2 3 4

Example 5. A randomly generated graph.

I created this graph with my script. It has 10 points and 30 edges (each edge is listed twice).

10  
60  
2 3 1  
3 2 1  
2 9 1  
9 2 1  
0 4 1  
4 0 1  
2 4 1  
4 2 1  
4 7 1  
7 4 1  
3 9 1  
9 3 1  
5 3 1  
3 5 1  
1 8 1  
8 1 1  
5 2 1  
2 5 1  
7 5 1  
5 7 1  
9 4 1  
4 9 1  
3 8 1  
8 3 1  
7 0 1  
0 7 1  
5 8 1  
8 5 1  
7 3 1  
3 7 1  
6 4 1  
4 6 1  
9 7 1  
7 9 1  
6 3 1  
3 6 1  
2 1 1  
1 2 1  
0 8 1  
8 0 1  
1 3 1  
3 1 1  
4 8 1  
8 4 1  
5 4 1  
4 5 1  
5 1 1  
1 5 1  
7 8 1  
8 7 1  
9 6 1  
6 9 1  
1 7 1  
7 1 1  
8 6 1  
6 8 1  
2 7 1  
7 2 1  
6 2 1  
2 6 1

Results.

[ [ 0 0 0 0 1 0 0 1 1 0 ]  
 [ 0 0 1 1 0 1 0 1 1 0 ]  
 [ 0 1 0 1 1 1 1 1 0 1 ]  
 [ 0 1 1 0 0 1 1 1 1 1 ]  
 [ 1 0 1 0 0 1 1 1 1 1 ]  
 [ 0 1 1 1 1 0 0 1 1 0 ]  
 [ 0 0 1 1 1 0 0 0 1 1 ]  
 [ 1 1 1 1 1 1 0 0 1 1 ]  
 [ 1 1 0 1 1 1 1 1 0 0 ]  
 [ 0 0 1 1 1 0 1 1 0 0 ] ] 10x10  
  
  
Vertices in Max Clique:  
1 3 5 7 8

## Turn it in

Upload a zip file containing the files that use used or wrote yourself to the drop box on D2L:

* Make sure your name, CSCI 4270, and the Programming Assignment Number appear in comments in all of your files.
* Note: NO CREDIT will be given to programming assignments that do not compile.
* Make sure you have compiled and tested your program before handing it in.