CSC 520 Spring 2024
HW #2 — Complexity
25 points

Submit a .zip archive containing 3 files, or each file separately:

- A plain text .txt file that affirms the honor code. For individuals: `On my honor as an SFSU student, I, <name>, have neither given or received inappropriate help with this homework assignment.` For groups: `On our honor as SFSU students, we, <names>, have neither given or received inappropriate help with this homework assignment. All group members participated in this work, and all concur with the submission.`
- A .py Python source code file with your problem 1 solution.
- A .py Python source code file with your problem 2 solution.

1. (12.5 points). An instance of OneColorDup is a white-space delimited list of edges, separated by a semi-colon (';') from a white-space delimited list of node colorings. An edge is formatted as two node names separated by a comma (e.g., `"a,b"`), and a node coloring as a node name and a color separated by a colon (e.g., `"b:blue"`). For example, `'a,b b,c c,d d,a ; a:red b:blue c:yellow d:blue'` is an instance of OneColorDup.

A OneColorDup instance is a positive instance iff it encodes a graph with a directed cycle that goes through a node of every color in the colorings; no two adjacent nodes in that cycle have the same color; exactly one color occurs twice; and no color occurs more than two times. For example:

If I = `'a,b b,c c,d d,a ; a:red b:blue c:yellow d:blue'` then `'a,b,c,d'` verifies I as a positive instance of OneColorDup.

If I = `'a,b b,c c,a d,a ; a:red b:blue c:green d:yellow'` then `'a,b,c'` cannot verify I as a positive instance of OneColorDup because none of a, b, or c are yellow nodes.

If I = `''a,b  b,c  c,d  d,e e,f f,a; a:red b:blue c:red d:blue e:red f:yellow'` then `'a,b,c,d,e,f'` cannot verify I as a positive instance of OneColorDup because there are 3 red nodes in the proposed cycle.

Complete VfyOneColorDup-template.py, included with the test materials, so that it verifies OneColorDup in polynomial time. The only required import is graph.py from the *WCBC* library, also included with the test materials. An important indication that your changes are successful is that none of the test cases included with the module are flagged with **, meaning that the actual result differed from the expected one.

The test harness output from VfyOneColorDup-template.py. The explanations are hand coded in the test harness:

```
** test #1 VfyOneColorDup("a,b  b,c  c,d  d,a; a:red b:blue c:yellow
d:blue","maybe","a,b,c,d"): expected "unsure", received "correct"
test #1 Explanation: solution too long

** test #2 VfyOneColorDup("a,b  b,c  c,d  d,a; a:red b:blue c:yellow
d:blue","no","a,b,c,d"): expected "unsure", received "correct"
test #2 Explanation: can't verify negative instance

VERBOSE: VerifyOneColorDup()  Cycles must have at least 2 nodes.
test #3 VfyOneColorDup("a,a; a:red","yes","a"): expected "unsure", received "unsure"
test #3 Explanation: One node does not a cycle make

** test #4 VfyOneColorDup("a,b  b,c  c,d  d,a ; a:red b:blue c:yellow
d:blue","yes","e,a,b,c,d"): expected "unsure", received "correct"
test #4 Explanation: e not in graph

** test #5 VfyOneColorDup("a,b  b,a  c,d  d,a ; a:red b:blue c:yellow
d:blue","yes","a,b,a,d"): expected "unsure", received "correct"
test #5 Explanation: "a" occurs twice
```

```
** test #6 VfyOneColorDup("a,b  b,d  c,d  d,a; a:red b:blue c:yellow
d:blue","yes","a,b,c,d"): expected "unsure", received "correct"
test #6 Explanation: No b-c edge

** test #7 VfyOneColorDup("a,b  b,c  c,d  d,a; a:red b:blue c:blue
d:yellow","yes","a,b,c,d"): expected "unsure", received "correct"
test #7 Explanation: consecutive blues

** test #8 VfyOneColorDup("a,b  b,c  c,a  d,a; a:red b:blue c:green
d:yellow","yes","a,b,c"): expected "unsure", received "correct"
test #8 Explanation: no yellow in cycle

** test #9 VfyOneColorDup("a,b  b,c  c,d  d,e e,f f,a; a:red b:blue c:red d:blue e:red
f:yellow","yes","a,b,c,d,e,f"): expected "unsure", received "correct"
test #9 Explanation: 3 reds

** test #10 VfyOneColorDup("a,b  b,c  c,d  d,a; a:red b:blue c:white
d:yellow","yes","a,b,c,d"): expected "unsure", received "correct"
test #10 Explanation: no color duplicated

test #11 VfyOneColorDup("a,b  b,c  c,d  d,a; a:red b:blue c:yellow
d:blue","yes","a,b,c,d"): expected "correct", received "correct"
test #11 Explanation: a-b-c traverses every color; no consecutive nodes in cycle same
color; blue occurs twice
```

In contrast, this is the test harness output from a working version of OneDupNode, which generates verbose output to explain "unsure" returns.

```
test #1 VfyOneColorDup("a,b  b,c  c,d  d,a; a:red b:blue c:yellow
d:blue","maybe","a,b,c,d"): expected "unsure", received "unsure"
test #1 Explanation: solution too long

VERBOSE: VerifyOneColorDup()  solution != "yes"
test #2 VfyOneColorDup("a,b  b,c  c,d  d,a; a:red b:blue c:yellow d:blue","no","a,b,c,d"):
expected "unsure", received "unsure"
test #2 Explanation: can't verify negative instance

VERBOSE: VerifyOneColorDup()  Cycles must have at least 2 nodes.
test #3 VfyOneColorDup("a,a; a:red","yes","a"): expected "unsure", received "unsure"
test #3 Explanation: One node does not a cycle make

VERBOSE: VerifyOneColorDup() "e" in hint but not graph
DEV: {'a': 'red', 'b': 'blue', 'c': 'yellow', 'd': 'blue'}
test #4 VfyOneColorDup("a,b  b,c  c,d  d,a ; a:red b:blue c:yellow
d:blue","yes","e,a,b,c,d"): expected "unsure", received "unsure"
test #4 Explanation: e not in graph

VERBOSE: VerifyOneColorDup() "a" occurs twice in cycle
test #5 VfyOneColorDup("a,b  b,a  c,d  d,a ; a:red b:blue c:yellow
d:blue","yes","a,b,a,d"): expected "unsure", received "unsure"
test #5 Explanation: "a" occurs twice

VERBOSE: VerifyOneColorDup()  No edge connecting "b" to "c"
DEV: "['c,d', 'a,b', 'b,d', 'd,a']"
test #6 VfyOneColorDup("a,b  b,d  c,d  d,a; a:red b:blue c:yellow
d:blue","yes","a,b,c,d"): expected "unsure", received "unsure"
test #6 Explanation: No b-c edge
```

```
VERBOSE: VerifyOneColorDup() Adjacent nodes with color "blue"
test #7 VfyOneColorDup("a,b  b,c  c,d  d,a; a:red b:blue c:blue
d:yellow","yes","a,b,c,d"): expected "unsure", received "unsure"
test #7 Explanation: consecutive blues

VERBOSE: VerifyOneColorDup() No "yellow" colored nodes in cycle.
test #8 VfyOneColorDup("a,b  b,c  c,a  d,a; a:red b:blue c:green d:yellow","yes","a,b,c"):
expected "unsure", received "unsure"
test #8 Explanation: no yellow in cycle

VERBOSE: VerifyOneColorDup() "red" occurs 3 times in cycle
test #9 VfyOneColorDup("a,b  b,c  c,d  d,e e,f f,a; a:red b:blue c:red d:blue e:red
f:yellow","yes","a,b,c,d,e,f"): expected "unsure", received "unsure"
test #9 Explanation: 3 reds

VERBOSE: VerifyOneColorDup() No color duplicated
test #10 VfyOneColorDup("a,b  b,c  c,d  d,a; a:red b:blue c:white
d:yellow","yes","a,b,c,d"): expected "unsure", received "unsure"
test #10 Explanation: no color duplicated

test #11 VfyOneColorDup("a,b  b,c  c,d  d,a; a:red b:blue c:yellow
d:blue","yes","a,b,c,d"): expected "correct", received "correct"
test #11 Explanation: a-b-c traverses every color; no consecutive nodes in cycle same
color; blue occurs twice
```

2. (12.5 points) A **node cover** is a subset of the nodes in an unweighted, undirected graph, specified as a white space delimited sequence of edges, such that **every node in the encoded graph has at least one end point in the node cover** subset. For example, the graph `'a,b a,c b,c'` has a node cover of size 2: {a,b}.

An instance of HasNodeCover is a graph encoding as specified above, separated by a "`;`" from the **maximum acceptable node cover size**. For example, `'a,b a,c b,c;2'` is an instance of HasNodeCover.

An instance of HasNodeCover is a positive instance iff it encodes a graph with a node cover no larger than the specified maximum. For example, `'a,b a,c b,c;2'` is a positive instance of HasNodeCover, but `'a,b a,c b,c;1'` is a negative instance since there is no single node that covers the graph.

An **independent set** is a subset of the nodes in an unweighted, undirected graph, specified as a white space delimited sequence of edges, such that there are **no edges between any two nodes in the independent set**. For example, the graph "`a,a a,c a,d a,e b,c b,d b,e c,d c,e d,e`" has an independent set of size 2, {`a`,`b`} since there is no link connecting nodes `a` and `b`. Note that loop back nodes are not considered when specifying independent sets.

An instance of HalfIndependentSet is also an unweighted, undirected graph specified as a series of edges. For example, "`a,d a,e a,f b,d b,e b,f c,c c,f e,f`".

A HalfIndependentSet instance is a positive instance iff at least half the nodes in the graph form an independent set. For example, "`a,d a,e a,f b,d b,e b,f c,c c,f e,f`" is a positive instance because {`a`, `b`, `c`} forms an independent set of size 3, out of a total of 6 graph nodes. But "`a,a a,c a,d a,e b,b b,c b,d b,e c,d c,e d,e`" is a negative instance, because it has no independent set larger than 2, which is one less than half the number of the nodes in the graph.

Complete the template file ConvertNodeCoverToHalfIndependentSet-template.py to prove HalfIndependentSet is NP-hard by polyreduction from HasNodeCover. Add a comment at the end arguing that your code is a valid proof; you may wish to consider Karp's 1972 paper, in which he showed that solving NodeCover was as hard as solving SAT (see below for copies of presentation slides on the Node Cover, Clique, and Independent Set problems) . You are welcome to call ConvertCliqueToHalfIndependentSet (in test materials), and ncToClique (defined in template), but are not required to do so. It is possible to make the needed code changes in a handful of lines. The test harness works in the usual way.

Output from ConvertNodeCoverToHalfIndependentSet-template.py before any changes, with unexpected results flagged by **. Note the verbose output from VfyHalfIndependentSet.py, included with the test materials. If you do not want to see verbose output from any module, just set verbose to false at the top of the file.

```
VERBOSE: vfyNCViaVfyHalfIndependentSet(): "z" is not a valid int encoding
test #1 vfyNCViaVfyHalfIndependentSet("a,b a,c a,d a,e b,f;z", "yes", "a f"): expected
"unsure", received "unsure"
test #1 Explanation: "z" is not an encoding of a decimal integer

VERBOSE: ConvertNodeCoverToHalfIndependentSet() node covering size "0" < 1
DEV: vfyNCViaVfyHalfIndependentSet(): hint for VfyHalfIndependentSet: "b e c d "
VERBOSE: VfyHalfIndependentSet() ""b e c d " not independent set, edge "e,c" exists"
test #2 vfyNCViaVfyHalfIndependentSet("a,b a,c a,d a,e b,f;0", "yes", "a f"): expected
"unsure", received "unsure"
test #2 Explanation: |node cover subset| must be greater than 0

DEV: vfyNCViaVfyHalfIndependentSet(): hint for VfyHalfIndependentSet: "b e c d "
test #3 vfyNCViaVfyHalfIndependentSet("a,b a,c a,d a,e b,f;2", "yes", "a f"): expected
"correct", received "correct"
test #3 Explanation: {a f} is node cover

DEV: vfyNCViaVfyHalfIndependentSet(): hint for VfyHalfIndependentSet: "e b c d g "
** test #4 vfyNCViaVfyHalfIndependentSet("a,b a,c a,d a,e b,f b,g;2", "yes", "a f"):
expected "unsure", received "correct"
test #4 Explanation: {a f} is not a node cover

DEV: vfyNCViaVfyHalfIndependentSet(): hint for VfyHalfIndependentSet: "e f "
VERBOSE: VfyHalfIndependentSet() 2 nodes in hint, 3 needed
** test #5 vfyNCViaVfyHalfIndependentSet("a,b b,c c,d d,e d,f;4", "yes", "a b c d"):
expected "correct", received "unsure"
test #5 Explanation: {a b c d} is node cover

DEV: vfyNCViaVfyHalfIndependentSet(): hint for VfyHalfIndependentSet: "e f g "
VERBOSE: VfyHalfIndependentSet() 3 nodes in hint, 4 needed
test #6 vfyNCViaVfyHalfIndependentSet("a,b a,c a,d a,e b,f b,g;4", "yes", "a b c d"):
expected "unsure", received "unsure"
test #6 Explanation: {a b c d } is not a node cover

DEV: vfyNCViaVfyHalfIndependentSet(): hint for VfyHalfIndependentSet: "e f "
VERBOSE: VfyHalfIndependentSet() 2 nodes in hint, 3 needed
test #7 vfyNCViaVfyHalfIndependentSet("a,b b,c c,d d,e d,f;3", "yes", "a b c d"): expected
"unsure", received "unsure"
test #7 Explanation: |{a b c d}| > 3

DEV: vfyNCViaVfyHalfIndependentSet(): hint for VfyHalfIndependentSet: "b c "
test #8 vfyNCViaVfyHalfIndependentSet("a,b a,c;1", "yes", "a"): expected "correct",
received "correct"
test #8 Explanation: {a} is a node cover

DEV: vfyNCViaVfyHalfIndependentSet(): hint for VfyHalfIndependentSet: "a c "
** test #9 vfyNCViaVfyHalfIndependentSet("a,b a,c;1", "yes", "b"): expected "unsure",
received "correct"
test #9 Explanation: {b} is not a node cover
```

CLIQUE
Input: graph G, positive integer k
Property: G has a set of k mutually adjacent nodes.

Karp's definition of Clique
Each node in a clique is connected to all the other nodes in the clique.

NODE COVER
Input: graph G', positive integer l
Property: There is a set $R \subseteq N'$ such that $|R| \leqslant l$ and every arc is incident with some node in R. [N' *is the set of nodes in the graph* G'.]

Karp's definition of Node Cover

In other words, the *node cover* of a graph is a set of vertices that includes at least one endpoint of every edge of the graph.

Karp's reduction from Clique to Node Cover

$$\text{CLIQUE} \propto \text{NODE COVER}$$

G' is the complement of G.
$l = |N| - k$

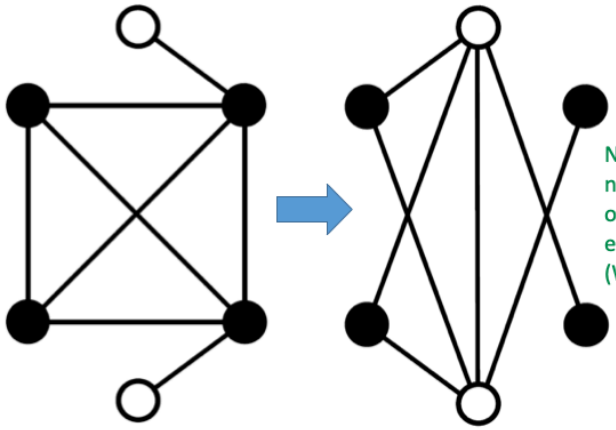
G' is a *complement* of G if G' has an edge everywhere G doesn't, and vice versa.

|R|, the number of nodes to cover G' = the number of nodes - the size of the clique.

Example of Clique $\leq_P$ Node Cover

Exercise: prove this maps positive to positive and negative to negative

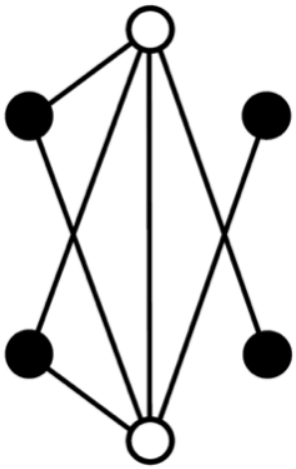
lique ≝ a set of nodes uch that every node in ıe set is connected to l the others. (Black ɔdes in G.)



Node cover ≝ a set of nodes including at least one end point of every edge in the graph. (White nodes in G'.)

Clique instance G with $k = 4$ Node Cover instance G' with $l = 6 - 4 = 2$

Node Cover ≤P Independent Set



IndependentSetD(g,n) ≝ ∃S((S is a subset of the nodes in graph g) and (|S| ≥ n) and (no node in S is adjacent to any other node in S)). In other words, there are no edges connecting any two nodes in S.

For example, if g_1 = `'a,e a,b c,e d,e'`, then $'g_1;'3'$ is a positive instance of IndependentSet because there are no edges connecting nodes b, c, and d. But g_1,`'4'` is a negative instance. And if g_2 = `'a,e a,b c,d c,e d,e'`, then g_2,`'3'` is also a negative instance of IndependentSet, since the addition of the edge c,d means that b, c, and d are not completely disconnected from each other.

The diagram, taken from the previous slide, shows a node cover subset (white disks). Every edge in the diagram must have a white disk as one or both end points, so there can be no edge connecting two blacks disks.
∴ the black disks form an independent set.