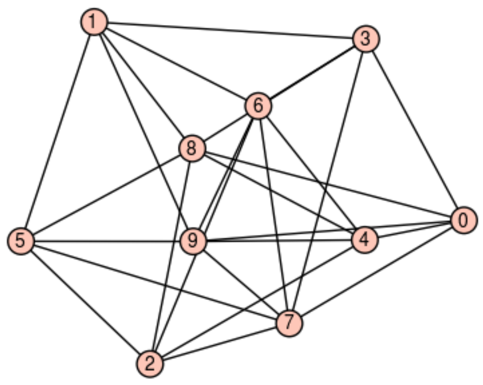
```
# In all that follows, alpha(G) will represent the independence number of the graph G,
# eta(G) will denote the Haemers' minimum rank of G, and vcc(G) will denote the
# vertex clique cover number of G.

# There are 50 graphs of order 10 where the method of Proposition 3.9 is used.
# Recall, the basic method of that Proposition is to assume eta(G) = alpha(G)
# look at a matrix fitting G, and arrive at a contradiction.

# Of the 50 graphs, 13 have alpha(G) = 4 and vcc(G) = 5,
# and 37 have alpha(G) = 3 and vcc(G) = 4.

# Since the method of Proposition 3.9 takes up so much space, we try to reduce
# the amount of work by considering spanning subgraphs.  That is, recall from
# Remark 2.1 that for H a spanning subgraph of G, eta(G) <= eta(H).
# So, for example, if we can show one graph with alpha(G) = 3 and vcc(G) = 4
# has eta(G) = 4, then any spanning subgraph, H, of G will satisfy eta(H) >= 4.


# This is the group of 13 graphs with alpha(G) = 4 and vcc(G) = 5
Group_45 = ['I?qa``ed_', 'I?qa``edg', 'I?qa`ped_', 'I?qa`peYO', 'I?qa`pedg', 'I?qa`peYW', 'I?qa`pheW',
'I?qaeM[Yg', 'I?qdRI\Mo', 'I?q`qhiXO', 'I?q`qhieg', 'I?q`qhiXW', 'ICQ`e`dn?']

print "Group of matrices with alpha(G) = 4, vcc(G) = 5"

max_size = max(Graph(graph6).size() for graph6 in Group_45)
print "Max size:", max_size

for graph6 in Group_45:
    g = Graph(graph6)
    print g.size()

print "Graphs with max size are:"

for graph6 in Group_45:
    if Graph(graph6).size() == max_size:
        print graph6

print ""

# This is the group of 13 graphs with independence number 3 and vcc 4
Group_34 = ['IEhethk\o', 'IEhethkzo', 'IEhetjkfo', 'IEhetjktW', 'IEhethmmO', 'IEhethm}_', 'IEhethm}O',
'IEhethm\o', 'IEhethm]g', 'IEhethm~_', 'IEhethmzo', 'IEhethm]w', 'IEhetixtW', 'IEhethnlo', 'IEherhmmO',
'IEherhmnO', 'IEherZsVo', 'IEherYulW', 'IEhevU{Zo', 'IEher^snG', 'IEhbtj{ro', 'IEhbve{tW', 'IEhbuu{tW',
'IEhbtn{ro', 'IEhuVquZ_', 'IEhuTzqZ_', 'IEhuTzifo', 'IEhuTxy|_', 'IEhuTxyto', 'IEhuTxymW', 'IEhuTxm}_',
'IEhuTxmv_', 'IEhuTxm~_', 'IEhuTxmzo', 'IEhuTzmfo', 'IEhutzifo', 'IEhutziZg']

print "Group of matrices with alpha(G) = 3, vcc(G) = 4"

max_size = max(Graph(graph6).size() for graph6 in Group_34)
print "Max size:", max_size

for graph6 in Group_34:
    g = Graph(graph6)
    print g.size()

print "Graphs with max size are:"

for graph6 in Group_34:
    if Graph(graph6).size() == max_size:
        print graph6
```

```
Group of matrices with alpha(G) = 4, vcc(G) = 5
Max size: 20
16
17
17
17
18
18
18
19
20
18
19
19
18
Graphs with max size are:
```

```
I?qdRI\Mo

Group of matrices with alpha(G) = 3, vcc(G) = 4
Max size: 27
24
25
25
25
24
25
25
25
25
26
26
26
25
26
24
25
25
25
26
26
26
26
26
27
25
25
26
26
26
26
26
26
27
27
27
27
27
Graphs with max size are:
IEhbtn{ro
IEhuTxm~_
IEhuTxmzo
IEhuTzmfo
IEhutzifo
IEhutziZg
```

```
# We start by figuring out eta(G) for the group of 37 graphs with alpha(G) = 3 and vcc(G) = 4
# We do so by finding eta(G) for the 6 graphs with size 27 first
# and finding which of the others are spanning subgraphs.
# There will be a few graphs that are not a spanning subgraph of at least one of these.
# We will deal with those at the end.

# Again, with every graph, recall that our method is to assume eta(G) = alpha(G) = 3,
# and find a contradiction, which will imply eta(G) = 4.

# First graph of size 27

h=Graph('IEhbtn{ro')
h.show()
print maximum_independent_sets(h)
```



```
[[0, 1, 2], [0, 5, 6], [3, 4, 5], [3, 9, 2], [4, 1, 7]]
```

```
# We start with a general matrix fitting G.
# Here, 2 represents an entry that must be nonzero, 1 represents a free entry,
# and 0 represents that the entry must be 0.

# Note, since Python starts arrays at 0, we think of the first row or column as
# row or column 0, and the last as 9 (since there are 10 total rows and columns)

original_mat = h.am() + 2
mat = copy(original_mat)
print mat
```

```
[2 0 0 1 1 0 0 1 1 1]
[0 2 0 1 0 1 1 0 1 1]
[0 0 2 0 1 1 1 1 1 0]
[1 1 0 2 0 0 1 1 1 0]
[1 0 1 0 2 0 1 0 1 1]
[0 1 1 0 0 2 0 1 1 1]
[0 1 1 1 1 0 2 1 0 1]
[1 0 1 1 0 1 1 2 0 1]
[1 1 1 1 1 1 0 0 2 0]
[1 1 0 0 1 1 1 1 0 2]
```

```
# Column 3 (which stands for the fourth column) is a combination of columns 0 and 1.
# Looking in row 4, either mat[4, 0] = 0 or column 3 is a multiple of column 1.
# So, assume mat[4, 0] = 0
mat[4, 0] = 0
print mat
```

```
[2 0 0 1 1 0 0 1 1 1]
[0 2 0 1 0 1 1 0 1 1]
[0 0 2 0 1 1 1 1 1 0]
[1 1 0 2 0 0 1 1 1 0]
[0 0 1 0 2 0 1 0 1 1]
[0 1 1 0 0 2 0 1 1 1]
[0 1 1 1 1 0 2 1 0 1]
[1 0 1 1 0 1 1 2 0 1]
[1 1 1 1 1 1 0 0 2 0]
[1 1 0 0 1 1 1 1 0 2]
```

```
# Now, we see that row 4 is a multiple of row 2.
mat = compare_rows(mat, 2, 4)
print mat
```

```
[2 0 0 1 1 0 0 1 1 1]
[0 2 0 1 0 1 1 0 1 1]
[0 0 2 0 2 0 1 0 1 0]
[1 1 0 2 0 0 1 1 1 0]
[0 0 2 0 2 0 1 0 1 0]
[0 1 1 0 0 2 0 1 1 1]
[0 1 1 1 1 0 2 1 0 1]
[1 0 1 1 0 1 1 2 0 1]
[1 1 1 1 1 1 0 0 2 0]
[1 1 0 0 1 1 1 1 0 2]
```

```
# Column 5 must be a multiple of Column 1
mat = compare_columns(mat, 1, 5)
print mat
```

```
[2 0 0 1 1 0 0 1 1 1]
[0 2 0 1 0 2 1 0 1 1]
```

```
 [0 0 2 0 2 0 1 0 1 0]
 [1 0 0 2 0 0 1 1 1 0]
 [0 0 2 0 2 0 1 0 1 0]
 [0 2 1 0 0 2 0 1 1 1]
 [0 0 1 1 1 0 2 1 0 1]
 [1 0 1 1 0 0 1 2 0 1]
 [1 1 1 1 1 1 0 0 2 0]
 [1 1 0 0 1 1 1 1 0 2]
```

```
# Column 7 multiple of Column 0 and Row 3 multiple of Row 0
mat = compare_columns(mat, 0, 7)
mat = compare_rows(mat, 0, 3)
print mat
```

```
[2 0 0 2 0 0 0 2 1 0]
[0 2 0 1 0 2 1 0 1 1]
[0 0 2 0 2 0 1 0 1 0]
[2 0 0 2 0 0 0 2 1 0]
[0 0 2 0 2 0 1 0 1 0]
[0 2 1 0 0 2 0 0 1 1]
[0 0 1 1 1 0 2 0 0 1]
[2 0 1 1 0 0 1 2 0 1]
[0 1 1 1 1 1 0 0 2 0]
[1 1 0 0 1 1 1 1 0 2]
```

```
# Row 6 is multiple of Row 2
mat = compare_rows(mat, 2, 6)
print mat
```

```
[2 0 0 2 0 0 0 2 1 0]
[0 2 0 1 0 2 1 0 1 1]
[0 0 2 0 2 0 2 0 0 0]
[2 0 0 2 0 0 0 2 1 0]
[0 0 2 0 2 0 1 0 1 0]
[0 2 1 0 0 2 0 0 1 1]
[0 0 2 0 2 0 2 0 0 0]
[2 0 1 1 0 0 1 2 0 1]
[0 1 1 1 1 1 0 0 2 0]
[1 1 0 0 1 1 1 1 0 2]
```

```
# Column 9 is a multiple of Column 1
mat = compare_columns(mat, 1, 9)
print mat
```

```
[2 0 0 2 0 0 0 2 1 0]
[0 2 0 1 0 2 1 0 1 2]
[0 0 2 0 2 0 2 0 0 0]
[2 0 0 2 0 0 0 2 1 0]
[0 0 2 0 2 0 1 0 1 0]
[0 2 1 0 0 2 0 0 1 2]
[0 0 2 0 2 0 2 0 0 0]
[2 0 1 1 0 0 1 2 0 0]
[0 0 1 1 1 1 0 0 2 0]
[1 2 0 0 1 1 1 1 0 2]
```

```
# Row 8 is a multiple of Row 2 (which is also a multiple of row 4)
mat = compare_rows(mat, 2, 8)
mat = compare_rows(mat, 2, 4)
print mat
```

```
[2 0 0 2 0 0 0 2 1 0]
[0 2 0 1 0 2 1 0 1 2]
[0 0 2 0 2 0 2 0 0 0]
[2 0 0 2 0 0 0 2 1 0]
[0 0 2 0 2 0 2 0 0 0]
[0 2 1 0 0 2 0 0 1 2]
[0 0 2 0 2 0 2 0 0 0]
[2 0 1 1 0 0 1 2 0 0]
[0 0 2 0 2 0 0 0 2 0]
[1 2 0 0 1 1 1 1 0 2]
```

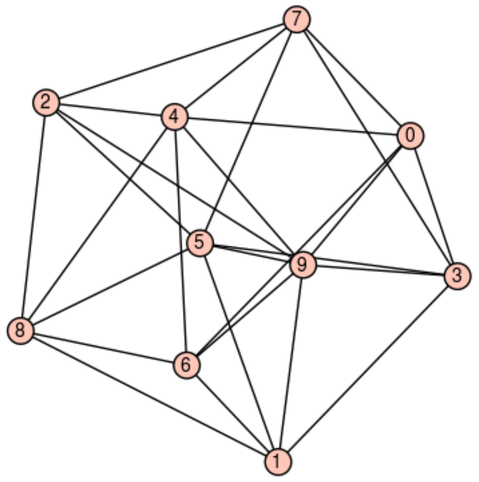
```
# Contradiction, therefore, mat[4, 0] can not be 0 and column 3 must be a multiple of column 1.
# The same argument on the rows shows mat[0, 4] = 2 and row 3 is a multiple or row 1
print original_mat
print ""

mat = copy(original_mat)
mat[4, 0] = 2
mat[0, 4] = 2
mat = compare_rows(mat, 1, 3)
mat = compare_columns(mat, 1, 3)
print mat
```

```
[2 0 0 1 1 0 0 1 1 1]
[0 2 0 1 0 1 1 0 1 1]
[0 0 2 0 1 1 1 1 1 0]
[1 1 0 2 0 0 1 1 1 0]
[1 0 1 0 2 0 1 0 1 1]
[0 1 1 0 0 2 0 1 1 1]
[0 1 1 1 1 0 2 1 0 1]
[1 0 1 1 0 1 1 2 0 1]
[1 1 1 1 1 1 0 0 2 0]
[1 1 0 0 1 1 1 1 0 2]

[2 0 0 0 2 0 0 1 1 1]
[0 2 0 2 0 0 1 0 1 0]
[0 0 2 0 1 1 1 1 1 0]
[0 2 0 2 0 0 1 0 1 0]
[2 0 1 0 2 0 1 0 1 1]
[0 0 1 0 0 2 0 1 1 1]
[0 1 1 1 1 0 2 1 0 1]
[1 0 1 0 0 1 1 2 0 1]
[1 1 1 1 1 1 0 0 2 0]
[1 0 0 0 1 1 1 1 0 2]
```

```
# What has been accomplished is deleting 4 edges (0, 3), (1, 5), (3, 7), (1, 9)
# to create a new graph H.  What this says is, if eta(G) is 3, the matrix fitting G
# and having rank 3 must also fit H.  Thus, eta(H) <= 3.  But, eta(H) = 4, as we can see below.
# Therefore, eta(G) >= 4, or eta(G) = 4.
h.delete_edges([(0, 3), (1, 5), (3, 7), (1, 9)])

print "The independence number of h is:", len(h.independent_set())
print "Trying to increase the lower bound by Theorems 3.2 and 3.6 gives:", increase_lower_bound(h)
```

```
The independence number of h is: 3
Trying to increase the lower bound by Theorems 3.2 and 3.6 gives: 4
```

```
# Second graph of size 27:

h=Graph('IEhuTxm~_')
h.show()
print maximum_independent_sets(h)
```



```
[[0, 1, 2], [3, 2, 6], [7, 8, 9]]
```

```
original_mat = h.am() + 2
mat = copy(original_mat)
print mat
```

```
    [2 0 0 1 1 0 1 1 0 1]
    [0 2 0 1 0 1 1 0 1 1]
    [0 0 2 0 1 1 0 1 1 1]
    [1 1 0 2 0 1 0 1 0 1]
    [1 0 1 0 2 0 1 1 1 1]
    [0 1 1 1 0 2 0 1 1 1]
    [1 1 0 0 1 0 2 0 1 1]
    [1 0 1 1 1 1 0 2 0 0]
    [0 1 1 0 1 1 1 0 2 0]
    [1 1 1 1 1 1 1 0 0 2]
```

```
# Here, we see that column 3 is a linear combination of columns 0 and 1.
# Based on row 4, we see that either mat[4,0] = 0
# or column 3 is a multiple of column 1.  Let's assume mat[4, 0] = 0.

mat[4, 0] = 0
print mat
```

```
    [2 0 0 1 1 0 1 1 0 1]
    [0 2 0 1 0 1 1 0 1 1]
    [0 0 2 0 1 1 0 1 1 1]
    [1 1 0 2 0 1 0 1 0 1]
    [0 0 1 0 2 0 1 1 1 1]
    [0 1 1 1 0 2 0 1 1 1]
    [1 1 0 0 1 0 2 0 1 1]
    [1 0 1 1 1 1 0 2 0 0]
    [0 1 1 0 1 1 1 0 2 0]
    [1 1 1 1 1 1 1 0 0 2]
```

```
# Now, row 4 must be a multiple of row 2
mat = compare_rows(mat, 2, 4)
print mat
```

```
    [2 0 0 1 1 0 1 1 0 1]
    [0 2 0 1 0 1 1 0 1 1]
    [0 0 2 0 2 0 0 1 1 1]
    [1 1 0 2 0 1 0 1 0 1]
    [0 0 2 0 2 0 0 1 1 1]
    [0 1 1 1 0 2 0 1 1 1]
    [1 1 0 0 1 0 2 0 1 1]
    [1 0 1 1 1 1 0 2 0 0]
    [0 1 1 0 1 1 1 0 2 0]
    [1 1 1 1 1 1 1 0 0 2]
```

```
# Column 5 is a multiple of column 1
mat = compare_columns(mat, 1, 5)
print mat
```

```
    [2 0 0 1 1 0 1 1 0 1]
    [0 2 0 1 0 2 1 0 1 1]
    [0 0 2 0 2 0 0 1 1 1]
    [1 1 0 2 0 1 0 1 0 1]
    [0 0 2 0 2 0 0 1 1 1]
    [0 2 1 1 0 2 0 1 1 1]
    [1 0 0 0 1 0 2 0 1 1]
    [1 0 1 1 1 0 0 2 0 0]
    [0 1 1 0 1 1 1 0 2 0]
    [1 1 1 1 1 1 1 0 0 2]
```

```
# Row 6 is a multiple of Row 0
mat = compare_rows(mat, 0, 6)
print mat
```

```
    [2 0 0 0 1 0 2 0 0 1]
    [0 2 0 1 0 2 1 0 1 1]
    [0 0 2 0 2 0 0 1 1 1]
    [1 1 0 2 0 1 0 1 0 1]
    [0 0 2 0 2 0 0 1 1 1]
    [0 2 1 1 0 2 0 1 1 1]
    [2 0 0 0 1 0 2 0 0 1]
    [1 0 1 1 1 0 0 2 0 0]
    [0 1 1 0 1 1 1 0 2 0]
    [1 1 1 1 1 1 1 0 0 2]
```

```
# Col 3 is a multiple of col 1 and col 7 is a multiple of column 2
mat = compare_columns(mat, 1, 3)
mat = compare_columns(mat, 2, 7)
print mat
```

```
[2 0 0 0 1 0 2 0 0 1]
[0 2 0 2 0 2 1 0 1 1]
[0 0 2 0 2 0 0 2 1 1]
[1 2 0 2 0 1 0 0 0 1]
[0 0 2 0 2 0 0 2 1 1]
[0 2 1 2 0 2 0 1 1 1]
[2 0 0 0 1 0 2 0 0 1]
[1 0 2 0 1 0 0 2 0 0]
[0 0 0 0 1 1 1 0 2 0]
[1 1 0 1 1 1 1 0 0 2]
```

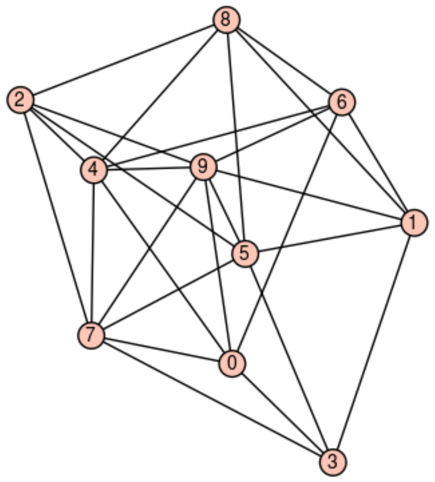
```
# Contradiction now because row 8 starts 0, 0, 0 so it can't be a linear comb of rows 0, 1, 2
# Therefore, mat[4, 0] is NOT 0.  And, column 1 IS multiple of column 3.  The same argument works on the
rows.

print original_mat
print ""

mat = copy(original_mat)
mat[4, 0] = 2
mat[0, 4] = 2
mat = compare_rows(mat, 1, 3)
mat = compare_columns(mat, 1, 3)
```

```
[2 0 0 1 1 0 1 1 0 1]
[0 2 0 1 0 1 1 0 1 1]
[0 0 2 0 1 1 0 1 1 1]
[1 1 0 2 0 1 0 1 0 1]
[1 0 1 0 2 0 1 1 1 1]
[0 1 1 1 0 2 0 1 1 1]
[1 1 0 0 1 0 2 0 1 1]
[1 0 1 1 1 1 0 2 0 0]
[0 1 1 0 1 1 1 0 2 0]
[1 1 1 1 1 1 1 0 0 2]

[2 0 0 0 2 0 1 1 0 1]
[0 2 0 2 0 1 0 0 0 1]
[0 0 2 0 1 1 0 1 1 1]
[0 2 0 2 0 1 0 0 0 1]
[2 0 1 0 2 0 1 1 1 1]
[0 1 1 1 0 2 0 1 1 1]
[1 0 0 0 1 0 2 0 1 1]
[1 0 1 0 1 1 0 2 0 0]
[0 0 1 0 1 1 1 0 2 0]
[1 1 1 1 1 1 1 0 0 2]
```

```
# The result is essentially deleting edges (0, 3), (6, 1), (7, 3), (8, 1)
h.delete_edges([(0, 3), (6, 1), (7, 3), (8, 1)])

print "The independence number of h is:", len(h.independent_set())
print "Trying to increase the lower bound by Theorems 3.2 and 3.6 gives:", increase_lower_bound(h)
```

```
The independence number of h is: 3
Trying to increase the lower bound by Theorems 3.2 and 3.6 gives: 4
```

```
# Third graph of size 27

h=Graph('IEhuTxmzo')
h.show()
print maximum_independent_sets(h)

original_mat = h.am() + 2
mat = copy(original_mat)
print mat
```



```
[[0, 1, 2], [3, 2, 6], [3, 8, 9]]
[2 0 0 1 1 0 1 1 0 1]
[0 2 0 1 0 1 1 0 1 1]
[0 0 2 0 1 1 0 1 1 1]
[1 1 0 2 0 1 0 1 0 0]
[1 0 1 0 2 0 1 1 1 1]
[0 1 1 1 0 2 0 1 1 1]
[1 1 0 0 1 0 2 0 1 1]
[1 0 1 1 1 1 0 2 0 1]
[0 1 1 0 1 1 1 0 2 0]
[1 1 1 0 1 1 1 1 0 2]
```

```
# Column 3 is a lin comb of columns 0 and 1.  Based on row 4, either mat[4, 0] = 0 or column 3 is a
# multiple of column 1.  Let us assume mat[4, 0] = 0.
mat[4, 0] = 0
print mat
```

```
[2 0 0 1 1 0 1 1 0 1]
[0 2 0 1 0 1 1 0 1 1]
[0 0 2 0 1 1 0 1 1 1]
[1 1 0 2 0 1 0 1 0 0]
[0 0 1 0 2 0 1 1 1 1]
[0 1 1 1 0 2 0 1 1 1]
[1 1 0 0 1 0 2 0 1 1]
[1 0 1 1 1 1 0 2 0 1]
[0 1 1 0 1 1 1 0 2 0]
[1 1 1 0 1 1 1 1 0 2]
```

```
# Row 4 is a multiple of row 2
mat = compare_rows(mat, 2, 4)
print mat
```

```
[2 0 0 1 1 0 1 1 0 1]
[0 2 0 1 0 1 1 0 1 1]
[0 0 2 0 2 0 0 1 1 1]
[1 1 0 2 0 1 0 1 0 0]
[0 0 2 0 2 0 0 1 1 1]
[0 1 1 1 0 2 0 1 1 1]
[1 1 0 0 1 0 2 0 1 1]
[1 0 1 1 1 1 0 2 0 1]
[0 1 1 0 1 1 1 0 2 0]
[1 1 1 0 1 1 1 1 0 2]
```

```
# Column 5 is a multiple of column 1
mat = compare_columns(mat, 1, 5)
print mat
```

```
[2 0 0 1 1 0 1 1 0 1]
[0 2 0 1 0 2 1 0 1 1]
[0 0 2 0 2 0 0 1 1 1]
[1 1 0 2 0 1 0 1 0 0]
[0 0 2 0 2 0 0 1 1 1]
[0 2 1 1 0 2 0 1 1 1]
[1 0 0 0 1 0 2 0 1 1]
[1 0 1 1 1 0 0 2 0 1]
[0 1 1 0 1 1 1 0 2 0]
[1 1 1 0 1 1 1 1 0 2]
```

```
# Row 6 is a multiple of Row 0
mat = compare_rows(mat, 0, 6)
print mat
```

```
[2 0 0 0 1 0 2 0 0 1]
[0 2 0 1 0 2 1 0 1 1]
[0 0 2 0 2 0 0 1 1 1]
[1 1 0 2 0 1 0 1 0 0]
[0 0 2 0 2 0 0 1 1 1]
[0 2 1 1 0 2 0 1 1 1]
[2 0 0 0 1 0 2 0 0 1]
[1 0 1 1 1 0 0 2 0 1]
[0 1 1 0 1 1 1 0 2 0]
[1 1 1 0 1 1 1 1 0 2]
```

```
# Column 3 is a multiple of column 1 and column 7 is a multiple of column 2
mat = compare_columns(mat, 1, 3)
mat = compare_columns(mat, 2, 7)
print mat
```

```
[2 0 0 0 1 0 2 0 0 1]
[0 2 0 2 0 2 1 0 1 1]
[0 0 2 0 2 0 0 2 1 1]
[1 2 0 2 0 1 0 0 0 0]
[0 0 2 0 2 0 0 2 1 1]
[0 2 1 2 0 2 0 1 1 1]
[2 0 0 0 1 0 2 0 0 1]
[1 0 2 0 1 0 0 2 0 1]
[0 0 0 0 1 1 1 0 2 0]
[1 0 1 0 1 1 1 1 0 2]
```

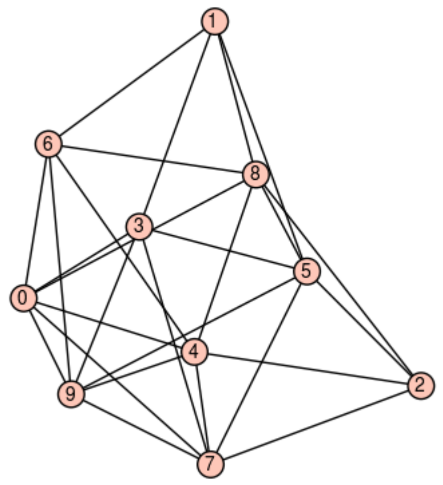
```
# Contradiction in row 8, so mat[4, 0] is not 0 and column 3 is a multiple of column 1.
# Similarly, with rows.

print original_mat
print ""

mat = copy(original_mat)
mat[4, 0] = 2
mat[0, 4] = 2
mat = compare_rows(mat, 1, 3)
mat = compare_columns(mat, 1, 3)
print mat
```

```
[2 0 0 1 1 0 1 1 0 1]
[0 2 0 1 0 1 1 0 1 1]
[0 0 2 0 1 1 0 1 1 1]
[1 1 0 2 0 1 0 1 0 0]
[1 0 1 0 2 0 1 1 1 1]
[0 1 1 1 0 2 0 1 1 1]
[1 1 0 0 1 0 2 0 1 1]
[1 0 1 1 1 1 0 2 0 1]
[0 1 1 0 1 1 1 0 2 0]
[1 1 1 0 1 1 1 1 0 2]

[2 0 0 0 2 0 1 1 0 1]
[0 2 0 2 0 1 0 0 0 0]
[0 0 2 0 1 1 0 1 1 1]
[0 2 0 2 0 1 0 0 0 0]
[2 0 1 0 2 0 1 1 1 1]
[0 1 1 1 0 2 0 1 1 1]
[1 0 0 0 1 0 2 0 1 1]
[1 0 1 0 1 1 0 2 0 1]
[0 0 1 0 1 1 1 0 2 0]
[1 0 1 0 1 1 1 1 0 2]
```

```
# This is like deleting edges: (0, 3), (1, 6), (1, 8), (1, 9), (3, 7)
h.delete_edges([(0, 3), (6, 1), (7, 3), (8, 1), (1, 9)])
print "The independence number of h is:", len(h.independent_set())
print "Trying to increase the lower bound by Theorems 3.2 and 3.6 gives:", increase_lower_bound(h)
```

```
The independence number of h is: 3
Trying to increase the lower bound by Theorems 3.2 and 3.6 gives: 4
```

```
# Fourth graph of size 27

h=Graph('IEhuTzmfo')
h.show()
print maximum_independent_sets(h)

original_mat = h.am() + 2
mat = copy(original_mat)
print mat
```



```
[[1, 2, 0], [1, 2, 9], [3, 2, 6]]
[2 0 0 1 1 0 1 1 1 1]
[0 2 0 1 0 1 1 0 1 0]
[0 0 2 0 1 1 0 1 1 0]
[1 1 0 2 0 1 0 1 0 1]
[1 0 1 0 2 0 1 1 1 1]
[0 1 1 1 0 2 0 1 1 1]
[1 1 0 0 1 0 2 0 1 1]
[1 0 1 1 1 1 0 2 0 1]
[1 1 1 0 1 1 1 0 2 0]
[1 0 0 1 1 1 1 1 0 2]
```

```
# Column 3 is a lin comb of 0 and 1.  Looking in row 4, either mat[4, 0] = 0 or
# column 3 is a multiple of column 1.
# Assume mat[4, 0] = 0.
mat[4, 0] = 0
print mat
```

```
[2 0 0 1 1 0 1 1 1 1]
[0 2 0 1 0 1 1 0 1 0]
[0 0 2 0 1 1 0 1 1 0]
[1 1 0 2 0 1 0 1 0 1]
[0 0 1 0 2 0 1 1 1 1]
[0 1 1 1 0 2 0 1 1 1]
[1 1 0 0 1 0 2 0 1 1]
[1 0 1 1 1 1 0 2 0 1]
[1 1 1 0 1 1 1 0 2 0]
[1 0 0 1 1 1 1 1 0 2]
```

```
# Row 4 is a multiple of row 2
mat = compare_rows(mat, 2, 4)
print mat
```

```
[2 0 0 1 1 0 1 1 1 1]
[0 2 0 1 0 1 1 0 1 0]
[0 0 2 0 2 0 0 1 1 0]
[1 1 0 2 0 1 0 1 0 1]
[0 0 2 0 2 0 0 1 1 0]
[0 1 1 1 0 2 0 1 1 1]
[1 1 0 0 1 0 2 0 1 1]
[1 0 1 1 1 1 0 2 0 1]
[1 1 1 0 1 1 1 0 2 0]
[1 0 0 1 1 1 1 1 0 2]
```

```
# Column 5 is a multiple of column 1 and column 9 is a multiple of column 0.
mat = compare_columns(mat, 1, 5)
mat = compare_columns(mat, 0, 9)
print mat
```

```
    [2 0 0 1 1 0 1 1 1 2]
    [0 2 0 1 0 2 1 0 1 0]
    [0 0 2 0 2 0 0 1 1 0]
    [1 1 0 2 0 1 0 1 0 1]
    [0 0 2 0 2 0 0 1 1 0]
    [0 2 1 1 0 2 0 1 1 0]
    [1 0 0 0 1 0 2 0 1 1]
    [1 0 1 1 1 0 0 2 0 1]
    [0 1 1 0 1 1 1 0 2 0]
    [2 0 0 1 1 0 1 1 0 2]
```

```
# Row 6 is a multiple of row 0 and row 9 is also
mat = compare_rows(mat, 0, 6)
mat = compare_rows(mat, 0, 9)
mat = compare_rows(mat, 0, 6)
print mat
```

```
    [2 0 0 0 1 0 2 0 0 2]
    [0 2 0 1 0 2 1 0 1 0]
    [0 0 2 0 2 0 0 1 1 0]
    [1 1 0 2 0 1 0 1 0 1]
    [0 0 2 0 2 0 0 1 1 0]
    [0 2 1 1 0 2 0 1 1 0]
    [2 0 0 0 1 0 2 0 0 2]
    [1 0 1 1 1 0 0 2 0 1]
    [0 1 1 0 1 1 1 0 2 0]
    [2 0 0 0 1 0 2 0 0 2]
```

```
# Column 3 is a multiple of column 1, column 7 is a multiple of column 2
mat = compare_columns(mat, 1, 3)
mat = compare_columns(mat, 2, 7)
print mat
```

```
    [2 0 0 0 1 0 2 0 0 2]
    [0 2 0 2 0 2 1 0 1 0]
    [0 0 2 0 2 0 0 2 1 0]
    [1 2 0 2 0 1 0 0 0 1]
    [0 0 2 0 2 0 0 2 1 0]
    [0 2 1 2 0 2 0 1 1 0]
    [2 0 0 0 1 0 2 0 0 2]
    [1 0 2 0 1 0 0 2 0 1]
    [0 0 0 0 1 1 1 0 2 0]
    [2 0 0 0 1 0 2 0 0 2]
```

```
# Contradiction in row 8, so mat[4, 0] is not 0 and column 3 is a multiple of column 1.
# Similarly with rows.

print original_mat
print ""

mat = copy(original_mat)
mat[4, 0] = 2
mat[0, 4] = 2
mat = compare_rows(mat, 1, 3)
mat = compare_columns(mat, 1, 3)
print mat
```

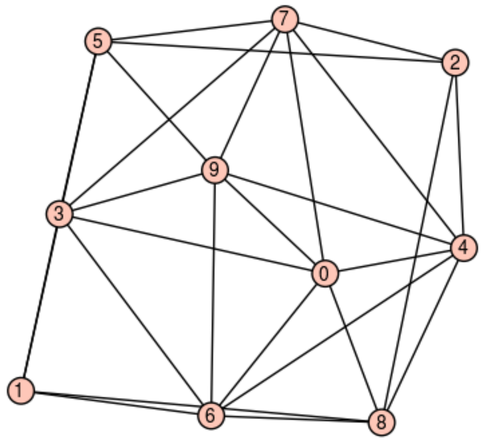
```
    [2 0 0 1 1 0 1 1 1 1]
    [0 2 0 1 0 1 1 0 1 0]
    [0 0 2 0 1 1 0 1 1 0]
    [1 1 0 2 0 1 0 1 0 1]
    [1 0 1 0 2 0 1 1 1 1]
    [0 1 1 1 0 2 0 1 1 1]
    [1 1 0 0 1 0 2 0 1 1]
    [1 0 1 1 1 1 0 2 0 1]
    [1 1 1 0 1 1 1 0 2 0]
    [1 0 0 1 1 1 1 1 0 2]

    [2 0 0 0 2 0 1 1 1 1]
    [0 2 0 2 0 1 0 0 0 0]
    [0 0 2 0 1 1 0 1 1 0]
    [0 2 0 2 0 1 0 0 0 0]
    [2 0 1 0 2 0 1 1 1 1]
    [0 1 1 1 0 2 0 1 1 1]
```

```
[1 0 0 0 1 0 2 0 1 1]
[1 0 1 0 1 1 0 2 0 1]
[1 0 1 0 1 1 1 0 2 0]
[1 0 0 0 1 1 1 1 0 2]
```

```
# This is same as deleting edges (0, 3), (1, 6), (1, 8), (3, 7), (3, 9)
h.delete_edges([(0, 3), (6, 1), (7, 3), (8, 1), (3, 9)])
print "The independence number of h is:", len(h.independent_set())
print "Trying to increase the lower bound by Theorems 3.2 and 3.6 gives:", increase_lower_bound(h)
```

```
The independence number of h is: 3
Trying to increase the lower bound by Theorems 3.2 and 3.6 gives: 4
```

```
# Fifth graph of size 27

h=Graph('IEhutzifo')
h.show()
print maximum_independent_sets(h)

original_mat = h.am() + 2
mat = copy(original_mat)
print mat
```



```
[[1, 2, 0], [1, 2, 9]]
[2 0 0 1 1 0 1 1 1 1]
[0 2 0 1 0 1 1 0 1 0]
[0 0 2 0 1 1 0 1 1 0]
[1 1 0 2 0 1 1 1 0 1]
[1 0 1 0 2 0 1 1 1 1]
[0 1 1 1 0 2 0 1 0 1]
[1 1 0 1 1 0 2 0 1 1]
[1 0 1 1 1 1 0 2 0 1]
[1 1 1 0 1 0 1 0 2 0]
[1 0 0 1 1 1 1 1 0 2]
```

```
# We can see for sure that row 9 is a multiple of row 0 and similarly with columns
mat = compare_rows(mat, 0, 9)
mat = compare_columns(mat, 0, 9)
print mat
```

```
[2 0 0 1 1 0 1 1 0 2]
[0 2 0 1 0 1 1 0 1 0]
[0 0 2 0 1 1 0 1 1 0]
[1 1 0 2 0 1 1 1 0 1]
[1 0 1 0 2 0 1 1 1 1]
[0 1 1 1 0 2 0 1 0 0]
[1 1 0 1 1 0 2 0 1 1]
[1 0 1 1 1 1 0 2 0 1]
[0 1 1 0 1 0 1 0 2 0]
[2 0 0 1 1 0 1 1 0 2]
```

```
# Column 3 is a lin comb of columns 0 and 1.  Thus, mat[4, 0] = 0 or column 3 is a multiple of column 1.
# Let's assume mat[4, 0] = 0.  Since column 9 is multiple of column 0, we also have mat[4, 9] = 0.
mat[4, 0] = 0
mat[4, 9] = 0
print mat
```

```
[2 0 0 1 1 0 1 1 0 2]
[0 2 0 1 0 1 1 0 1 0]
[0 0 2 0 1 1 0 1 1 0]
```

```
[1 1 0 2 0 1 1 1 0 1]
[0 0 1 0 2 0 1 1 1 0]
[0 1 1 1 0 2 0 1 0 0]
[1 1 0 1 1 0 2 0 1 1]
[1 0 1 1 1 1 0 2 0 1]
[0 1 1 0 1 0 1 0 2 0]
[2 0 0 1 1 0 1 1 0 2]
```

```
#Row 4 is a multiple of row 2
mat = compare_rows(mat, 2, 4)
print mat
```

```
[2 0 0 1 1 0 1 1 0 2]
[0 2 0 1 0 1 1 0 1 0]
[0 0 2 0 2 0 0 1 1 0]
[1 1 0 2 0 1 1 1 0 1]
[0 0 2 0 2 0 0 1 1 0]
[0 1 1 1 0 2 0 1 0 0]
[1 1 0 1 1 0 2 0 1 1]
[1 0 1 1 1 1 0 2 0 1]
[0 1 1 0 1 0 1 0 2 0]
[2 0 0 1 1 0 1 1 0 2]
```

```
# Column 5 is a multiple of column 1
mat = compare_columns(mat, 1, 5)
print mat
```

```
[2 0 0 1 1 0 1 1 0 2]
[0 2 0 1 0 2 1 0 1 0]
[0 0 2 0 2 0 0 1 1 0]
[1 1 0 2 0 1 1 1 0 1]
[0 0 2 0 2 0 0 1 1 0]
[0 2 1 1 0 2 0 1 0 0]
[1 0 0 1 1 0 2 0 1 1]
[1 0 1 1 1 0 0 2 0 1]
[0 0 1 0 1 0 1 0 2 0]
[2 0 0 1 1 0 1 1 0 2]
```

```
# Row 6 is a multiple of row 0, Row 8 is a multiple of row 2 (and both are multiples of row 4 also)
mat = compare_rows(mat, 0, 6)
mat = compare_rows(mat, 2, 8)
mat = compare_rows(mat, 2, 4)
mat = compare_rows(mat, 2, 8)
print mat
```

```
[2 0 0 1 1 0 2 0 0 2]
[0 2 0 1 0 2 1 0 1 0]
[0 0 2 0 2 0 0 0 2 0]
[1 1 0 2 0 1 1 1 0 1]
[0 0 2 0 2 0 0 0 2 0]
[0 2 1 1 0 2 0 1 0 0]
[2 0 0 1 1 0 2 0 0 2]
[1 0 1 1 1 0 0 2 0 1]
[0 0 2 0 2 0 0 0 2 0]
[2 0 0 1 1 0 1 1 0 2]
```

```
# Contradiction in column 7, so mat[4, 0] is not 0 and column 3 is multiple of column 1
print original_mat
print ""

mat = copy(original_mat)
mat[4, 0] = 2
mat[0, 4] = 2
mat = compare_rows(mat, 1, 3)
mat = compare_columns(mat, 1, 3)
print mat
```

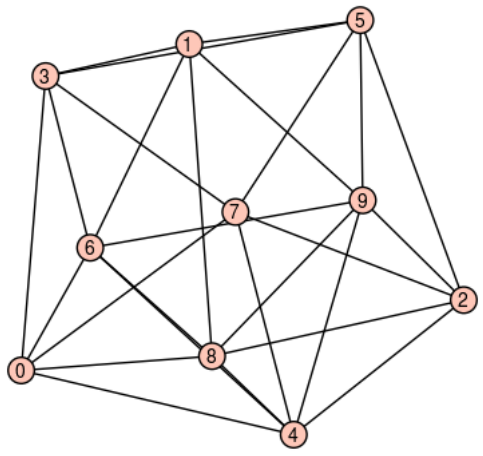
```
[2 0 0 1 1 0 1 1 1 1]
[0 2 0 1 0 1 1 0 1 0]
[0 0 2 0 1 1 0 1 1 0]
[1 1 0 2 0 1 1 1 0 1]
[1 0 1 0 2 0 1 1 1 1]
[0 1 1 1 0 2 0 1 0 1]
[1 1 0 1 1 0 2 0 1 1]
[1 0 1 1 1 1 0 2 0 1]
[1 1 1 0 1 0 1 0 2 0]
[1 0 0 1 1 1 1 1 0 2]

[2 0 0 0 2 0 1 1 1 1]
```

```
    [0 2 0 2 0 1 1 0 0 0]
    [0 0 2 0 1 1 0 1 1 0]
    [0 2 0 2 0 1 1 0 0 0]
    [2 0 1 0 2 0 1 1 1 1]
    [0 1 1 1 0 2 0 1 0 1]
    [1 1 0 1 1 0 2 0 1 1]
    [1 0 1 0 1 1 0 2 0 1]
    [1 0 1 0 1 0 1 0 2 0]
    [1 0 0 0 1 1 1 1 0 2]
```

```
# This is like deleting edges (0, 3), (7, 3), (9, 3), (1, 8)
h.delete_edges([(0, 3), (7, 3), (8, 1), (3, 9)])
print "The independence number of h is:", len(h.independent_set())
print "Trying to increase the lower bound by Theorems 3.2 and 3.6 gives:", increase_lower_bound(h)
```

```
    The independence number of h is: 3
    Trying to increase the lower bound by Theorems 3.2 and 3.6 gives: 4
```

```
# Sixth graph of size 27

h=Graph('IEhutziZg')
h.show()
print maximum_independent_sets(h)

original_mat = h.am() + 2
mat = copy(original_mat)
print mat
```



```
    [[0, 1, 2]]
    [2 0 0 1 1 0 1 1 1 0]
    [0 2 0 1 0 1 1 0 1 1]
    [0 0 2 0 1 1 0 1 1 1]
    [1 1 0 2 0 1 1 1 0 0]
    [1 0 1 0 2 0 1 1 1 1]
    [0 1 1 1 0 2 0 1 0 1]
    [1 1 0 1 1 0 2 0 1 1]
    [1 0 1 1 1 1 0 2 0 0]
    [1 1 1 0 1 0 1 0 2 1]
    [0 1 1 0 1 1 1 0 1 2]
```

```
# Column 3 is a lin comb of columns 0, 1.  Thus, mat[4, 0] = 0 or column 3 is a multiple of column 1.
# Assume mat[4, 0] = 0.
mat[4, 0] = 0
print mat
```

```
    [2 0 0 1 1 0 1 1 1 0]
    [0 2 0 1 0 1 1 0 1 1]
    [0 0 2 0 1 1 0 1 1 1]
    [1 1 0 2 0 1 1 1 0 0]
    [0 0 1 0 2 0 1 1 1 1]
    [0 1 1 1 0 2 0 1 0 1]
    [1 1 0 1 1 0 2 0 1 1]
    [1 0 1 1 1 1 0 2 0 0]
    [1 1 1 0 1 0 1 0 2 1]
    [0 1 1 0 1 1 1 0 1 2]
```

```
# Row 4 is a multiple of row 2
mat = compare_rows(mat, 2, 4)
print mat
```

```
[2 0 0 1 1 0 1 1 1 0]
[0 2 0 1 0 1 1 0 1 1]
[0 0 2 0 2 0 0 1 1 1]
[1 1 0 2 0 1 1 1 0 0]
[0 0 2 0 2 0 0 1 1 1]
[0 1 1 1 0 2 0 1 0 1]
[1 1 0 1 1 0 2 0 1 1]
[1 0 1 1 1 1 0 2 0 0]
[1 1 1 0 1 0 1 0 2 1]
[0 1 1 0 1 1 1 0 1 2]
```

```
# Col 5 is a multiple of col 1
mat = compare_columns(mat, 1, 5)
print mat
```

```
[2 0 0 1 1 0 1 1 1 0]
[0 2 0 1 0 2 1 0 1 1]
[0 0 2 0 2 0 0 1 1 1]
[1 1 0 2 0 1 1 1 0 0]
[0 0 2 0 2 0 0 1 1 1]
[0 2 1 1 0 2 0 1 0 1]
[1 0 0 1 1 0 2 0 1 1]
[1 0 1 1 1 0 0 2 0 0]
[1 0 1 0 1 0 1 0 2 1]
[0 1 1 0 1 1 1 0 1 2]
```

```
# Row 6 is a multiple of Row 0
mat = compare_rows(mat, 0, 6)
print mat
```

```
[2 0 0 1 1 0 2 0 1 0]
[0 2 0 1 0 2 1 0 1 1]
[0 0 2 0 2 0 0 1 1 1]
[1 1 0 2 0 1 1 1 0 0]
[0 0 2 0 2 0 0 1 1 1]
[0 2 1 1 0 2 0 1 0 1]
[2 0 0 1 1 0 2 0 1 0]
[1 0 1 1 1 0 0 2 0 0]
[1 0 1 0 1 0 1 0 2 1]
[0 1 1 0 1 1 1 0 1 2]
```

```
# Col 7 is a multiple of col 2
mat = compare_columns(mat, 2, 7)
print mat
```

```
[2 0 0 1 1 0 2 0 1 0]
[0 2 0 1 0 2 1 0 1 1]
[0 0 2 0 2 0 0 2 1 1]
[1 1 0 2 0 1 1 0 0 0]
[0 0 2 0 2 0 0 2 1 1]
[0 2 1 1 0 2 0 1 0 1]
[2 0 0 1 1 0 2 0 1 0]
[1 0 2 1 1 0 0 2 0 0]
[1 0 0 0 1 0 1 0 2 1]
[0 1 0 0 1 1 1 0 1 2]
```

```
# Row 8 is a multiple of row 0, row 9 is a multiple of Row 1
mat = compare_rows(mat, 0, 8)
mat = compare_rows(mat, 1, 9)
print mat
```

```
[2 0 0 0 1 0 2 0 2 0]
[0 2 0 0 0 2 1 0 1 2]
[0 0 2 0 2 0 0 2 1 1]
[1 1 0 2 0 1 1 0 0 0]
[0 0 2 0 2 0 0 2 1 1]
[0 2 1 1 0 2 0 1 0 1]
[2 0 0 1 1 0 2 0 1 0]
[1 0 2 1 1 0 0 2 0 0]
[2 0 0 0 1 0 2 0 2 0]
[0 2 0 0 0 2 1 0 1 2]
```

```
# Contradiction in col 3 so mat[4, 0] is not 0, col 3 is mult of col 1
print original_mat
print ""

mat = copy(original_mat)
mat[4, 0] = 2
mat[0, 4] = 2
mat = compare_rows(mat, 1, 3)
mat = compare_columns(mat, 1, 3)
print mat
```

```
[2 0 0 1 1 0 1 1 1 0]
[0 2 0 1 0 1 1 0 1 1]
[0 0 2 0 1 1 0 1 1 1]
[1 1 0 2 0 1 1 1 0 0]
[1 0 1 0 2 0 1 1 1 1]
[0 1 1 1 0 2 0 1 0 1]
[1 1 0 1 1 0 2 0 1 1]
[1 0 1 1 1 1 0 2 0 0]
[1 1 1 0 1 0 1 0 2 1]
[0 1 1 0 1 1 1 0 1 2]

[2 0 0 0 2 0 1 1 1 0]
[0 2 0 2 0 1 1 0 0 0]
[0 0 2 0 1 1 0 1 1 1]
[0 2 0 2 0 1 1 0 0 0]
[2 0 1 0 2 0 1 1 1 1]
[0 1 1 1 0 2 0 1 0 1]
[1 1 0 1 1 0 2 0 1 1]
[1 0 1 0 1 1 0 2 0 0]
[1 0 1 0 1 0 1 0 2 1]
[0 0 1 0 1 1 1 0 1 2]
```

```
# This is like deleting edges (0, 3), (7, 3), (1, 8), (1, 9)
h.delete_edges([(0, 3), (7, 3), (1, 8), (1, 9)])
print "The independence number of h is:", len(h.independent_set())
print "Trying to increase the lower bound by Theorems 3.2 and 3.6 gives:", increase_lower_bound(h)
```

```
The independence number of h is: 3
Trying to increase the lower bound by Theorems 3.2 and 3.6 gives: 4
```

```
# Of the graphs with alpha(G) = 3, vcc(G) = 4, there are 6 graphs of size 27, which is the max.
# We found all 6 of these had eta(G) = 4.
# Therefore, any spanning subgraph of one of the graphs in this group has lower bound 4
# (and already had upper bound 4), so has eta(G) = 4.
# So, we look for all such spanning subgraphs of these 6 graphs.

Group_34 = ['IEhethk\o', 'IEhethkzo', 'IEhetjkfo', 'IEhetjktW', 'IEhethmmO', 'IEhethm}_', 'IEhethm}O', 'IEhe
'IEhethm]g', 'IEhethm~_', 'IEhethmzo', 'IEhethm]w', 'IEhetixtW', 'IEhethnlo', 'IEherhmmO', 'IEherhmnO', 'IEh
'IEherYulW', 'IEhevU{Zo', 'IEher^snG', 'IEhbtj{ro', 'IEhbve{tW', 'IEhbuu{tW', 'IEhbtn{ro', 'IEhuVquZ_', 'IEh
'IEhuTzifo', 'IEhuTxy|_', 'IEhuTxyto', 'IEhuTxymW', 'IEhuTxm}_', 'IEhuTxmv_', 'IEhuTxm~_', 'IEhuTxmzo', 'IEh
'IEhutzifo', 'IEhutziZg']

print "Number of graphs in group:", len(Group_34)

graphs_with_max_size = ['IEhbtn{ro', 'IEhuTxm~_', 'IEhuTxmzo', 'IEhuTzmfo', 'IEhutzifo', 'IEhutziZg']

all_subgraphs = []

for big_graph6 in graphs_with_max_size:
    big_graph = Graph(big_graph6)
    for small_graph6 in Group_34:
        small_graph = Graph(small_graph6)
        subgraph = big_graph.subgraph_search(small_graph)
        if subgraph is not None:
            if all_subgraphs.count(small_graph6) == 0:
                all_subgraphs.append(small_graph6)

print "Each of the following graphs is a spanning subgraph of one of the 6 graphs with size 27:"
print all_subgraphs
print "The number of such graphs is:", len(all_subgraphs)
print ""

# The question is, which graphs are not in here.
graphs_still_need = list(set(Group_34).difference(set(all_subgraphs)))
print "The following graphs are not spanning subgraphs:", graphs_still_need
```

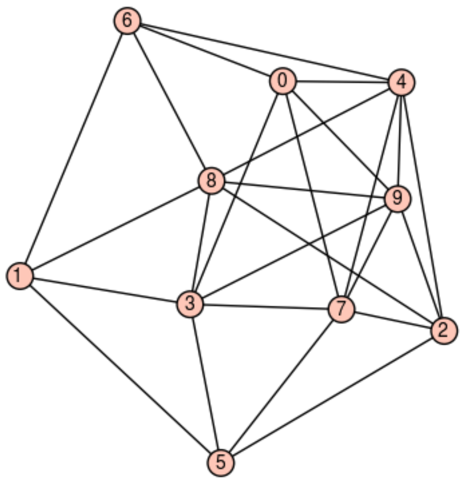
```
Number of graphs in group: 37
```

```
Each of the following graphs is a spanning subgraph of one of the 6
graphs with size 27:
['IEhbtj{ro', 'IEhbtn{ro', 'IEhethk\\o', 'IEhethkzo', 'IEhethmmO',
'IEhethm}O', 'IEhethm\\o', 'IEhethmzo', 'IEherhmmO', 'IEherhmnO',
'IEhuTzqZ_', 'IEhuTxm}_', 'IEhuTxmv_', 'IEhuTxm~_', 'IEhethm}_',
'IEhethm]g', 'IEhethm~_', 'IEhethm]w', 'IEhbuu{tW', 'IEhuTxyto',
'IEhuTxmzo', 'IEhetjkfo', 'IEhethnlo', 'IEherYulW', 'IEhbve{tW',
'IEhuTzifo', 'IEhuTzmfo', 'IEherZsVo', 'IEhevU{Zo', 'IEhutzifo',
'IEher^snG', 'IEhutziZg']
The number of such graphs is: 32

The following graphs are not spanning subgraphs: ['IEhetixtW',
'IEhuTxy|_', 'IEhuTxymW', 'IEhetjktW', 'IEhuVquZ_']
```

```
for graph_index in graphs_still_need:
    g = Graph(graph_index)
    print g.size()

# So, graphs 1, 2 are the size 26 graphs.  Are the other 3 subgraphs?
g = Graph(graphs_still_need[1])
for graph_index in graphs_still_need:
    if g.subgraph_search(Graph(graph_index)) is not None:
        print graph_index
```

```
25
26
26
25
25
IEhuTxy|_
IEhuVquZ_
```

```
# The above reveals that IEhetjktW is a spanning subgraph of IEhuTxymW and
# that IEhuVquZ_ is a spanning subgraph of IEhuTxy|_
# Therefore, if we show eta(G) = 4 for the 3 graphs IEhuTxymW, IEhuTxy|_, IEhetixtW
# we will show eta(G) = 4 for all 5 graphs in this last group, which will finish
# off the entire group with alpha(G) = 3, vcc(G) = 4.


# First extra graph from Group_34

h=Graph('IEhuTxymW')
h.show()
print maximum_independent_sets(h)

original_mat = h.am() + 2
mat = copy(original_mat)
print mat
```



```
[[1, 0, 2], [3, 2, 6], [5, 9, 6], [8, 0, 5]]
[2 0 0 1 1 0 1 1 0 1]
[0 2 0 1 0 1 1 0 1 0]
[0 0 2 0 1 1 0 1 1 1]
[1 1 0 2 0 1 0 1 1 1]
[1 0 1 0 2 0 1 1 1 1]
[0 1 1 1 0 2 0 1 0 0]
[1 1 0 0 1 0 2 0 1 0]
[1 0 1 1 1 1 0 2 0 1]
```

```
    [0 1 1 1 1 0 1 0 2 1]
    [1 0 1 1 1 0 0 1 1 2]
```

```
# Col 3 is lin comb of 0 and 1.  So, mat[4, 0] = 0 or col 3 is multiple of col 1.
# Assume mat[4, 0] = 0.
mat[4,0] = 0
print mat
```

```
    [2 0 0 1 1 0 1 1 0 1]
    [0 2 0 1 0 1 1 0 1 0]
    [0 0 2 0 1 1 0 1 1 1]
    [1 1 0 2 0 1 0 1 1 1]
    [0 0 1 0 2 0 1 1 1 1]
    [0 1 1 1 0 2 0 1 0 0]
    [1 1 0 0 1 0 2 0 1 0]
    [1 0 1 1 1 1 0 2 0 1]
    [0 1 1 1 1 0 1 0 2 1]
    [1 0 1 1 1 0 0 1 1 2]
```

```
# Row 4 is multiple of row 2
mat = compare_rows(mat, 2, 4)
print mat
```

```
    [2 0 0 1 1 0 1 1 0 1]
    [0 2 0 1 0 1 1 0 1 0]
    [0 0 2 0 2 0 0 1 1 1]
    [1 1 0 2 0 1 0 1 1 1]
    [0 0 2 0 2 0 0 1 1 1]
    [0 1 1 1 0 2 0 1 0 0]
    [1 1 0 0 1 0 2 0 1 0]
    [1 0 1 1 1 1 0 2 0 1]
    [0 1 1 1 1 0 1 0 2 1]
    [1 0 1 1 1 0 0 1 1 2]
```

```
# Col 5 is a multiple of col 1.
mat = compare_columns(mat, 1, 5)
print mat
```

```
    [2 0 0 1 1 0 1 1 0 1]
    [0 2 0 1 0 2 1 0 1 0]
    [0 0 2 0 2 0 0 1 1 1]
    [1 1 0 2 0 1 0 1 1 1]
    [0 0 2 0 2 0 0 1 1 1]
    [0 2 1 1 0 2 0 1 0 0]
    [1 0 0 0 1 0 2 0 1 0]
    [1 0 1 1 1 0 0 2 0 1]
    [0 0 1 1 1 0 1 0 2 1]
    [1 0 1 1 1 0 0 1 1 2]
```

```
# Row 6 mult of row 0, row 8 mult of row 2 (and row 4 already mult of row 2)
mat = compare_rows(mat, 0, 6)
mat = compare_rows(mat, 2, 8)
mat = compare_rows(mat, 2, 4)
mat = compare_rows(mat, 2, 8)
print mat
```

```
    [2 0 0 0 1 0 2 0 0 0]
    [0 2 0 1 0 2 1 0 1 0]
    [0 0 2 0 2 0 0 0 2 1]
    [1 1 0 2 0 1 0 1 1 1]
    [0 0 2 0 2 0 0 0 2 1]
    [0 2 1 1 0 2 0 1 0 0]
    [2 0 0 0 1 0 2 0 0 0]
    [1 0 1 1 1 0 0 2 0 1]
    [0 0 2 0 2 0 0 0 2 1]
    [1 0 1 1 1 0 0 1 1 2]
```

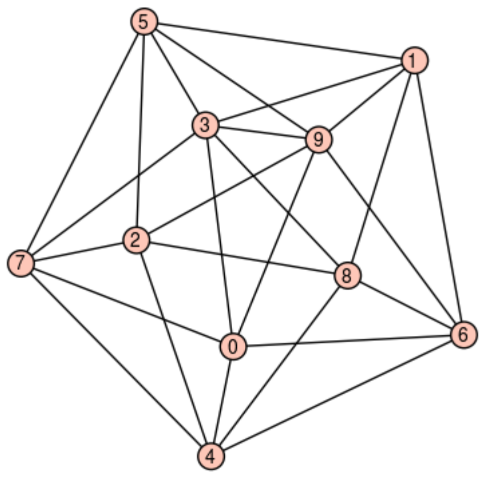
```
# Contradiction, mat[4, 0] not 0 and column 3 multiple of column 1 is true
print original_mat
print ""

mat = copy(original_mat)
mat[4, 0] = 2
mat[0, 4] = 2
mat = compare_rows(mat, 1, 3)
mat = compare_columns(mat, 1, 3)
print mat
```

```
[2 0 0 1 1 0 1 1 0 1]
[0 2 0 1 0 1 1 0 1 0]
[0 0 2 0 1 1 0 1 1 1]
[1 1 0 2 0 1 0 1 1 1]
[1 0 1 0 2 0 1 1 1 1]
[0 1 1 1 0 2 0 1 0 0]
[1 1 0 0 1 0 2 0 1 0]
[1 0 1 1 1 1 0 2 0 1]
[0 1 1 1 1 0 1 0 2 1]
[1 0 1 1 1 0 0 1 1 2]

[2 0 0 0 2 0 1 1 0 1]
[0 2 0 2 0 1 0 0 1 0]
[0 0 2 0 1 1 0 1 1 1]
[0 2 0 2 0 1 0 0 1 0]
[2 0 1 0 2 0 1 1 1 1]
[0 1 1 1 0 2 0 1 0 0]
[1 0 0 0 1 0 2 0 1 0]
[1 0 1 0 1 1 0 2 0 1]
[0 1 1 1 1 0 1 0 2 1]
[1 0 1 0 1 0 0 1 1 2]
```

```
# Like deleting edges (0, 3), (1, 6), (1, 8), (3, 7), (3, 9)
h.delete_edges([(0, 3), (1, 6), (1, 8), (3, 7), (3, 9)])
print "The independence number of h is:", len(h.independent_set())
print "Trying to increase the lower bound by Theorems 3.2 and 3.6 gives:", increase_lower_bound(h)
```

```
The independence number of h is: 3
Trying to increase the lower bound by Theorems 3.2 and 3.6 gives: 4
```

```
# Second extra graph from Group_34

h=Graph('IEhuTxy|_')
h.show()
print maximum_independent_sets(h)

original_mat = h.am() + 2
mat = copy(original_mat)
print mat
```



```
[[0, 1, 2], [0, 8, 5], [3, 2, 6], [7, 8, 9]]
[2 0 0 1 1 0 1 1 0 1]
[0 2 0 1 0 1 1 0 1 1]
[0 0 2 0 1 1 0 1 1 1]
[1 1 0 2 0 1 0 1 1 1]
[1 0 1 0 2 0 1 1 1 0]
[0 1 1 1 0 2 0 1 0 1]
```

```
    [1 1 0 0 1 0 2 0 1 1]
    [1 0 1 1 1 1 0 2 0 0]
    [0 1 1 1 1 0 1 0 2 0]
    [1 1 1 1 0 1 1 0 0 2]
```

```
# Col 3 is lin comb of cols 0 and 1.  Thus, mat[4, 0] = 0 or col 3 is multiple of col 1.
# Assume mat[4, 0] = 0
mat[4, 0] = 0
print mat
```

```
    [2 0 0 1 1 0 1 1 0 1]
    [0 2 0 1 0 1 1 0 1 1]
    [0 0 2 0 1 1 0 1 1 1]
    [1 1 0 2 0 1 0 1 1 1]
    [0 0 1 0 2 0 1 1 1 0]
    [0 1 1 1 0 2 0 1 0 1]
    [1 1 0 0 1 0 2 0 1 1]
    [1 0 1 1 1 1 0 2 0 0]
    [0 1 1 1 1 0 1 0 2 0]
    [1 1 1 1 0 1 1 0 0 2]
```

```
# Row 4 is mult of row 2
mat = compare_rows(mat, 2, 4)
print mat
```

```
    [2 0 0 1 1 0 1 1 0 1]
    [0 2 0 1 0 1 1 0 1 1]
    [0 0 2 0 2 0 0 1 1 0]
    [1 1 0 2 0 1 0 1 1 1]
    [0 0 2 0 2 0 0 1 1 0]
    [0 1 1 1 0 2 0 1 0 1]
    [1 1 0 0 1 0 2 0 1 1]
    [1 0 1 1 1 1 0 2 0 0]
    [0 1 1 1 1 0 1 0 2 0]
    [1 1 1 1 0 1 1 0 0 2]
```

```
# Col 5 is mult of col 1
mat = compare_columns(mat, 1, 5)
print mat
```

```
    [2 0 0 1 1 0 1 1 0 1]
    [0 2 0 1 0 2 1 0 1 1]
    [0 0 2 0 2 0 0 1 1 0]
    [1 1 0 2 0 1 0 1 1 1]
    [0 0 2 0 2 0 0 1 1 0]
    [0 2 1 1 0 2 0 1 0 1]
    [1 0 0 0 1 0 2 0 1 1]
    [1 0 1 1 1 0 0 2 0 0]
    [0 0 1 1 1 0 1 0 2 0]
    [1 1 1 1 0 1 1 0 0 2]
```

```
# Row 6 is mult of row 0, row 8 is mult of row 2 (also mult of row 4)
mat = compare_rows(mat, 0, 6)
mat = compare_rows(mat, 2, 8)
mat = compare_rows(mat, 2, 4)
mat = compare_rows(mat, 2, 8)
print mat
```

```
    [2 0 0 0 1 0 2 0 0 1]
    [0 2 0 1 0 2 1 0 1 1]
    [0 0 2 0 2 0 0 0 2 0]
    [1 1 0 2 0 1 0 1 1 1]
    [0 0 2 0 2 0 0 0 2 0]
    [0 2 1 1 0 2 0 1 0 1]
    [2 0 0 0 1 0 2 0 0 1]
    [1 0 1 1 1 0 0 2 0 0]
    [0 0 2 0 2 0 0 0 2 0]
    [1 1 1 1 0 1 1 0 0 2]
```

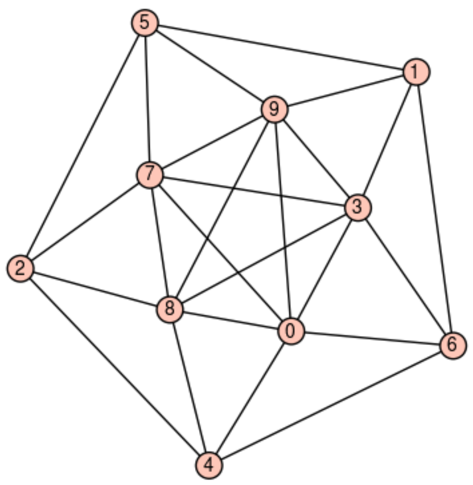
```
# Contradiction, so mat[4, 0] not 0 and col 3 is mult of col 1
print original_mat
print ""

mat = copy(original_mat)
mat[4, 0] = 2
mat[0, 4] = 2
mat = compare_rows(mat, 1, 3)
mat = compare_columns(mat, 1, 3)
print mat
```

```
[2 0 0 1 1 0 1 1 0 1]
[0 2 0 1 0 1 1 0 1 1]
[0 0 2 0 1 1 0 1 1 1]
[1 1 0 2 0 1 0 1 1 1]
[1 0 1 0 2 0 1 1 1 0]
[0 1 1 1 0 2 0 1 0 1]
[1 1 0 0 1 0 2 0 1 1]
[1 0 1 1 1 1 0 2 0 0]
[0 1 1 1 1 0 1 0 2 0]
[1 1 1 1 0 1 1 0 0 2]

[2 0 0 0 2 0 1 1 0 1]
[0 2 0 2 0 1 0 0 1 1]
[0 0 2 0 1 1 0 1 1 1]
[0 2 0 2 0 1 0 0 1 1]
[2 0 1 0 2 0 1 1 1 0]
[0 1 1 1 0 2 0 1 0 1]
[1 0 0 0 1 0 2 0 1 1]
[1 0 1 0 1 1 0 2 0 0]
[0 1 1 1 1 0 1 0 2 0]
[1 1 1 1 0 1 1 0 0 2]
```

```
# Same as deleting edges (0, 3), (1, 6), (3, 7)
h.delete_edges([(0, 3), (1, 6), (3, 7)])
print "The independence number of h is:", len(h.independent_set())
print "Trying to increase the lower bound by Theorems 3.2 and 3.6 gives:", increase_lower_bound(h)
```

```
The independence number of h is: 3
Trying to increase the lower bound by Theorems 3.2 and 3.6 gives: 4
```

```
# Third extra graph from Group_34

h=Graph('IEhetixtW')
h.show()
print maximum_independent_sets(h)

original_mat = h.am() + 2
mat = copy(original_mat)
print mat
```



```
[[1, 0, 2], [1, 4, 7], [3, 4, 5], [5, 8, 6], [6, 9, 2]]
[2 0 0 1 1 0 1 1 1 1]
[0 2 0 1 0 1 1 0 0 1]
[0 0 2 0 1 1 0 1 1 0]
[1 1 0 2 0 0 1 1 1 1]
[1 0 1 0 2 0 1 0 1 0]
[0 1 1 0 0 2 0 1 0 1]
```

```
[1 1 0 1 1 0 2 0 0 0]
[1 0 1 1 0 1 0 2 1 1]
[1 0 1 1 1 0 0 1 2 1]
[1 1 0 1 0 1 0 1 1 2]
```

```
# col 3 is lin comb of cols 0, 1.  Therefore, mat[4, 0] = 0 or col 3 is mult of col 1.
# Assume mat[4, 0] = 0.
mat[4,0] = 0
print mat
```

```
[2 0 0 1 1 0 1 1 1 1]
[0 2 0 1 0 1 1 0 0 1]
[0 0 2 0 1 1 0 1 1 0]
[1 1 0 2 0 0 1 1 1 1]
[0 0 1 0 2 0 1 0 1 0]
[0 1 1 0 0 2 0 1 0 1]
[1 1 0 1 1 0 2 0 0 0]
[1 0 1 1 0 1 0 2 1 1]
[1 0 1 1 1 0 0 1 2 1]
[1 1 0 1 0 1 0 1 1 2]
```

```
# Row 4 is mult of row 2
mat = compare_rows(mat, 2, 4)
print mat
```

```
[2 0 0 1 1 0 1 1 1 1]
[0 2 0 1 0 1 1 0 0 1]
[0 0 2 0 2 0 0 0 1 0]
[1 1 0 2 0 0 1 1 1 1]
[0 0 2 0 2 0 0 0 1 0]
[0 1 1 0 0 2 0 1 0 1]
[1 1 0 1 1 0 2 0 0 0]
[1 0 1 1 0 1 0 2 1 1]
[1 0 1 1 1 0 0 1 2 1]
[1 1 0 1 0 1 0 1 1 2]
```

```
# Col 5 is mult of col 1, col 7 is mult of col 0
mat = compare_columns(mat, 1, 5)
mat = compare_columns(mat, 0, 7)
print mat
```

```
[2 0 0 1 1 0 1 2 1 1]
[0 2 0 1 0 2 1 0 0 1]
[0 0 2 0 2 0 0 0 1 0]
[1 0 0 2 0 0 1 1 1 1]
[0 0 2 0 2 0 0 0 1 0]
[0 2 1 0 0 2 0 0 0 1]
[0 0 0 1 1 0 2 0 0 0]
[2 0 1 1 0 0 0 2 1 1]
[1 0 1 1 1 0 0 1 2 1]
[1 1 0 1 0 1 0 1 1 2]
```

```
# Contradiction, so mat[4, 0] is not 0 and col 3 is mult of col 1
print original_mat
print ""

mat = copy(original_mat)
mat[4, 0] = 2
mat[0, 4] = 2
mat = compare_rows(mat, 1, 3)
mat = compare_columns(mat, 1, 3)
print mat
```

```
[2 0 0 1 1 0 1 1 1 1]
[0 2 0 1 0 1 1 0 0 1]
[0 0 2 0 1 1 0 1 1 0]
[1 1 0 2 0 0 1 1 1 1]
[1 0 1 0 2 0 1 0 1 0]
[0 1 1 0 0 2 0 1 0 1]
[1 1 0 1 1 0 2 0 0 0]
[1 0 1 1 0 1 0 2 1 1]
[1 0 1 1 1 0 0 1 2 1]
[1 1 0 1 0 1 0 1 1 2]

[2 0 0 0 2 0 1 1 1 1]
[0 2 0 2 0 0 1 0 0 1]
[0 0 2 0 1 1 0 1 1 0]
[0 2 0 2 0 0 1 0 0 1]
[2 0 1 0 2 0 1 0 1 0]
```

```
[0 0 1 0 0 2 0 1 0 1]
[1 1 0 1 1 0 2 0 0 0]
[1 0 1 0 0 1 0 2 1 1]
[1 0 1 0 1 0 0 1 2 1]
[1 1 0 1 0 1 0 1 1 2]
```

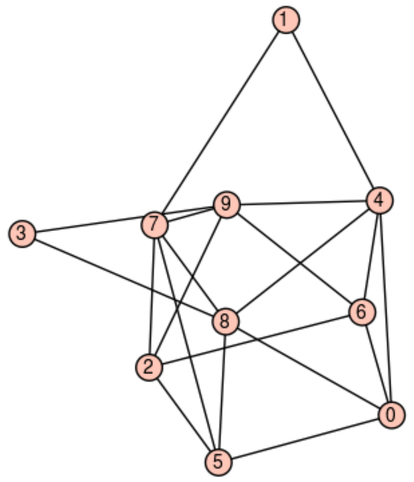
```
# Like deleting edges (0, 3), (1, 5), (3, 7), (3, 8)
h.delete_edges([(0, 3), (1, 5), (3, 7), (3, 8)])
print "The independence number of h is:", len(h.independent_set())
print "Trying to increase the lower bound by Theorems 3.2 and 3.6 gives:", increase_lower_bound(h)
```

```
The independence number of h is: 3
Trying to increase the lower bound by Theorems 3.2 and 3.6 gives: 4
```

```
# We now turn to the group of 13 graphs with alpha(G) = 4, vcc(G) = 5.
# We start with the graphs with the most edges, one with 20, and two with 19.
# The code below, shows that all 13 of these graphs is a spanning subgraphs of at
# least one of the 3 graphs_with_max_size.
# So, if we can show eta(G) = 5 for those 3, we are finished with all graphs of order 10.

Group_45 = ['I?qa``ed_', 'I?qa``edg', 'I?qa`ped_', 'I?qa`peYO', 'I?qa`pedg', 'I?qa`peYW', 'I?qa`pheW', 'I?qa
qdRI\Mo', 'I?q`qhiXO', 'I?q`qhieg', 'I?q`qhiXW', 'ICQ`e`dn?']

# The first graph in this list is size 20, the next two are size 19.
graphs_with_max_size = ['I?qdRI\Mo', 'I?q`qhiXW', 'I?q`qhieg']

# Subgraphs
all_subgraphs = []

for big_graph6 in graphs_with_max_size:
    big_graph = Graph(big_graph6)
    for small_graph6 in Group_45:
        small_graph = Graph(small_graph6)
        subgraph = big_graph.subgraph_search(small_graph)
        if subgraph is not None:
            if all_subgraphs.count(small_graph6) == 0:
                all_subgraphs.append(small_graph6)


print "Each of the following graphs is a spanning subgraph of one of the 3 graphs mentioned above, with size
print all_subgraphs
print "The number of such graphs is:", len(all_subgraphs)
print ""

# The question is, which graphs are not in here.
graphs_still_need = list(set(Group_45).difference(set(all_subgraphs)))
print "The following graphs are not spanning subgraphs:", graphs_still_need
```

```
Each of the following graphs is a spanning subgraph of one of the 3
graphs mentioned above, with size 19 or 20:
['I?qa``ed_', 'I?qa`peYO', 'I?qaeM[Yg', 'I?qdRI\\Mo', 'I?q`qhiXO',
'ICQ`e`dn?', 'I?qa``edg', 'I?qa`peYW', 'I?q`qhiXW', 'I?qa`ped_',
'I?qa`pedg', 'I?qa`pheW', 'I?q`qhieg']
The number of such graphs is: 13

The following graphs are not spanning subgraphs: []
```

```
# First graph from Group_45

h=Graph('I?qdRI\Mo')
h.show()
print maximum_independent_sets(h)

original_mat = h.am() + 2
mat = copy(original_mat)
print mat
```



```
[[1, 3, 0, 2], [1, 3, 5, 6]]
[2 0 0 0 1 1 1 0 1 0]
[0 2 0 0 1 0 0 1 0 0]
[0 0 2 0 0 1 1 1 0 1]
[0 0 0 2 0 0 0 0 1 1]
[1 1 0 0 2 0 1 0 1 1]
[1 0 1 0 0 2 0 1 1 0]
[1 0 1 0 1 0 2 0 0 1]
[0 1 1 0 0 1 0 2 1 1]
[1 0 0 1 1 1 0 1 2 0]
[0 0 1 1 1 0 1 1 0 2]
```

```
# col 4 is lin comb of cols 0 and 1.  Looking in row 5 shows mat[5, 0] = 0 or col 4 is mult of col 1.
# Assume mat[5, 0] = 0.
mat[5, 0] = 0
print mat
```

```
[2 0 0 0 1 1 1 0 1 0]
[0 2 0 0 1 0 0 1 0 0]
[0 0 2 0 0 1 1 1 0 1]
[0 0 0 2 0 0 0 0 1 1]
[1 1 0 0 2 0 1 0 1 1]
[0 0 1 0 0 2 0 1 1 0]
[1 0 1 0 1 0 2 0 0 1]
[0 1 1 0 0 1 0 2 1 1]
[1 0 0 1 1 1 0 1 2 0]
[0 0 1 1 1 0 1 1 0 2]
```

```
# Row 5 is mult of Row 2
mat = compare_rows(mat, 2, 5)
print mat
```

```
[2 0 0 0 1 1 1 0 1 0]
[0 2 0 0 1 0 0 1 0 0]
[0 0 2 0 0 2 0 1 0 0]
[0 0 0 2 0 0 0 0 1 1]
[1 1 0 0 2 0 1 0 1 1]
[0 0 2 0 0 2 0 1 0 0]
[1 0 1 0 1 0 2 0 0 1]
[0 1 1 0 0 1 0 2 1 1]
[1 0 0 1 1 1 0 1 2 0]
[0 0 1 1 1 0 1 1 0 2]
```

```
# Col 6 is mult of col 0, col 9 is mult of col 3
mat = compare_columns(mat, 0, 6)
mat = compare_columns(mat, 3, 9)
print mat
```

```
[2 0 0 0 1 1 2 0 1 0]
[0 2 0 0 1 0 0 1 0 0]
[0 0 2 0 0 2 0 1 0 0]
[0 0 0 2 0 0 0 0 1 2]
[1 1 0 0 2 0 1 0 1 0]
[0 0 2 0 0 2 0 1 0 0]
[2 0 1 0 1 0 2 0 0 0]
[0 1 1 0 0 1 0 2 1 0]
[0 0 0 0 1 1 0 1 2 0]
[0 0 1 2 1 0 0 1 0 2]
```

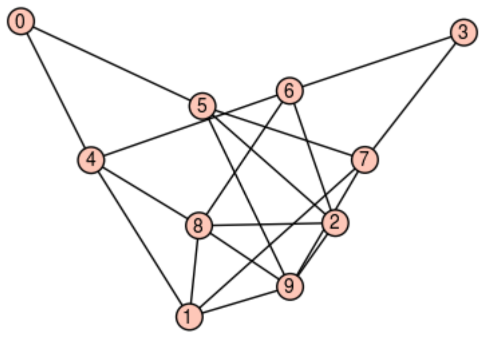
```
# Contradiction in row 8 so mat[5, 0] is not 0, and col 4 is a multiple of col 1.
# Same can be said with rows.
print original_mat
print ""

mat = copy(original_mat)
mat[5, 0] = 2
mat[0, 5] = 2
mat = compare_rows(mat, 1, 4)
mat = compare_columns(mat, 1, 4)
print mat
```

```
    [2 0 0 0 1 1 1 0 1 0]
    [0 2 0 0 1 0 0 1 0 0]
    [0 0 2 0 0 1 1 1 0 1]
    [0 0 0 2 0 0 0 0 1 1]
    [1 1 0 0 2 0 1 0 1 1]
    [1 0 1 0 0 2 0 1 1 0]
    [1 0 1 0 1 0 2 0 0 1]
    [0 1 1 0 0 1 0 2 1 1]
    [1 0 0 1 1 1 0 1 2 0]
    [0 0 1 1 1 0 1 1 0 2]

    [2 0 0 0 0 2 1 0 1 0]
    [0 2 0 0 2 0 0 0 0 0]
    [0 0 2 0 0 1 1 1 0 1]
    [0 0 0 2 0 0 0 0 1 1]
    [0 2 0 0 2 0 0 0 0 0]
    [2 0 1 0 0 2 0 1 1 0]
    [1 0 1 0 0 0 2 0 0 1]
    [0 0 1 0 0 1 0 2 1 1]
    [1 0 0 1 0 1 0 1 2 0]
    [0 0 1 1 0 0 1 1 0 2]
```

```
# Like deleting edges (0, 4), (1, 7), (4, 6), (4, 8), (4, 9)
h.delete_edges([(0, 4), (1, 7), (4, 6), (4, 8), (4, 9)])
print "The independence number of h is:", len(h.independent_set())
print "Trying to increase the lower bound by Theorems 3.2 and 3.6 gives:", increase_lower_bound(h)
```

```
   The independence number of h is: 4
   Trying to increase the lower bound by Theorems 3.2 and 3.6 gives: 5
```

```
# Second graph from Group_45

h=Graph('I?q`qhiXW')
h.show()
print maximum_independent_sets(h)

original_mat = h.am() + 2
mat = copy(original_mat)
print mat
```



```
    [[0, 3, 1, 2]]
    [2 0 0 0 1 1 0 0 0 0]
    [0 2 0 0 1 0 0 1 1 1]
    [0 0 2 0 0 1 1 0 1 1]
    [0 0 0 2 0 0 1 1 0 0]
    [1 1 0 0 2 0 1 0 1 0]
    [1 0 1 0 0 2 0 1 0 1]
    [0 0 1 1 1 0 2 0 1 0]
    [0 1 0 1 0 1 0 2 0 1]
    [0 1 1 0 1 0 1 0 2 1]
    [0 1 1 0 0 1 0 1 1 2]
```

```
# col 4 is lin comb of cols 0, 1.  mat[5, 0] = 0 or col 4 is mult of col 1.
# Assume mat[5,0] = 0
mat[5, 0] = 0
print mat
```

```
[2 0 0 0 1 1 0 0 0 0]
[0 2 0 0 1 0 0 1 1 1]
[0 0 2 0 0 1 1 0 1 1]
[0 0 0 2 0 0 1 1 0 0]
[1 1 0 0 2 0 1 0 1 0]
[0 0 1 0 0 2 0 1 0 1]
[0 0 1 1 1 0 2 0 1 0]
[0 1 0 1 0 1 0 2 0 1]
[0 1 1 0 1 0 1 0 2 1]
[0 1 1 0 0 1 0 1 1 2]
```

```
# Row 5 is mult of row 2
mat = compare_rows(mat, 2, 5)
print mat
```

```
[2 0 0 0 1 1 0 0 0 0]
[0 2 0 0 1 0 0 1 1 1]
[0 0 2 0 0 2 0 0 0 1]
[0 0 0 2 0 0 1 1 0 0]
[1 1 0 0 2 0 1 0 1 0]
[0 0 2 0 0 2 0 0 0 1]
[0 0 1 1 1 0 2 0 1 0]
[0 1 0 1 0 1 0 2 0 1]
[0 1 1 0 1 0 1 0 2 1]
[0 1 1 0 0 1 0 1 1 2]
```

```
# Col 6 is mult of col 3, col 8 is mult of col 1
mat = compare_columns(mat, 3, 6)
mat = compare_columns(mat, 1, 8)
print mat
```

```
[2 0 0 0 1 1 0 0 0 0]
[0 2 0 0 1 0 0 1 2 1]
[0 0 2 0 0 2 0 0 0 1]
[0 0 0 2 0 0 2 1 0 0]
[1 1 0 0 2 0 0 0 1 0]
[0 0 2 0 0 2 0 0 0 1]
[0 0 1 2 1 0 2 0 0 0]
[0 0 0 0 0 1 0 2 0 1]
[0 2 1 0 1 0 0 0 2 1]
[0 1 1 0 0 1 0 1 1 2]
```

```
# Contradiction in row 7 so mat[5, 0] is not 0, and col 4 is a multiple of col 1.
# Same can be said with rows.
print original_mat
print ""

mat = copy(original_mat)
mat[5, 0] = 2
mat[0, 5] = 2
mat = compare_rows(mat, 1, 4)
mat = compare_columns(mat, 1, 4)
print mat
```

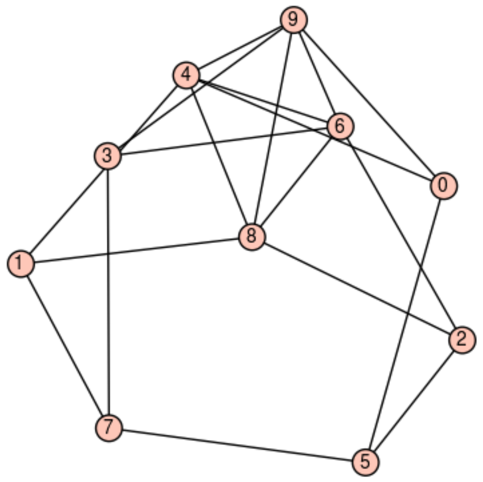
```
[2 0 0 0 1 1 0 0 0 0]
[0 2 0 0 1 0 0 1 1 1]
[0 0 2 0 0 1 1 0 1 1]
[0 0 0 2 0 0 1 1 0 0]
[1 1 0 0 2 0 1 0 1 0]
[1 0 1 0 0 2 0 1 0 1]
[0 0 1 1 1 0 2 0 1 0]
[0 1 0 1 0 1 0 2 0 1]
[0 1 1 0 1 0 1 0 2 1]
[0 1 1 0 0 1 0 1 1 2]

[2 0 0 0 0 2 0 0 0 0]
[0 2 0 0 2 0 0 0 1 0]
[0 0 2 0 0 1 1 0 1 1]
[0 0 0 2 0 0 1 1 0 0]
[0 2 0 0 2 0 0 0 1 0]
[2 0 1 0 0 2 0 1 0 1]
[0 0 1 1 0 0 2 0 1 0]
[0 0 0 1 0 1 0 2 0 1]
[0 1 1 0 1 0 1 0 2 1]
```

```
[0 0 1 0 0 1 0 1 1 2]
```

```
# Like deleting edges (0, 4), (4, 6), (1, 7), (1, 9)
h.delete_edges([(0, 4), (4, 6), (1, 7), (1, 9)])
print "The independence number of h is:", len(h.independent_set())
print "Trying to increase the lower bound by Theorems 3.2 and 3.6 gives:", increase_lower_bound(h)
```

```
The independence number of h is: 4
Trying to increase the lower bound by Theorems 3.2 and 3.6 gives: 5
```

```
# Third graph from Group_45

h=Graph('I?q`qhieg')
h.show()
print maximum_independent_sets(h)

original_mat = h.am() + 2
mat = copy(original_mat)
print mat
```



```
[[0, 1, 2, 3]]
[2 0 0 0 1 1 0 0 0 1]
[0 2 0 0 1 0 0 1 1 0]
[0 0 2 0 0 1 1 0 1 0]
[0 0 0 2 0 0 1 1 0 1]
[1 1 0 0 2 0 1 0 1 1]
[1 0 1 0 0 2 0 1 0 0]
[0 0 1 1 1 0 2 0 1 1]
[0 1 0 1 0 1 0 2 0 0]
[0 1 1 0 1 0 1 0 2 1]
[1 0 0 1 1 0 1 0 1 2]
```

```
# col 4 is lin comb of cols 0, 1.  Thus, mat[5, 0] = 0 or col 4 is mult of col 1.
# Assume mat[5, 0] = 0.
mat[5, 0] = 0
print mat
```

```
[2 0 0 0 1 1 0 0 0 1]
[0 2 0 0 1 0 0 1 1 0]
[0 0 2 0 0 1 1 0 1 0]
[0 0 0 2 0 0 1 1 0 1]
[1 1 0 0 2 0 1 0 1 1]
[0 0 1 0 0 2 0 1 0 0]
[0 0 1 1 1 0 2 0 1 1]
[0 1 0 1 0 1 0 2 0 0]
[0 1 1 0 1 0 1 0 2 1]
```

```
    [1 0 0 1 1 0 1 0 1 2]
```

```
# Row 5 is mult of row 2
mat = compare_rows(mat, 2, 5)
print mat
```

```
    [2 0 0 0 1 1 0 0 0 1]
    [0 2 0 0 1 0 0 1 1 0]
    [0 0 2 0 0 2 0 0 0 0]
    [0 0 0 2 0 0 1 1 0 1]
    [1 1 0 0 2 0 1 0 1 1]
    [0 0 2 0 0 2 0 0 0 0]
    [0 0 1 1 1 0 2 0 1 1]
    [0 1 0 1 0 1 0 2 0 0]
    [0 1 1 0 1 0 1 0 2 1]
    [1 0 0 1 1 0 1 0 1 2]
```

```
# Col 6 is mult of col 3, col 8 is mult of col 1
mat = compare_columns(mat, 3, 6)
mat = compare_columns(mat, 1, 8)
print mat
```

```
    [2 0 0 0 1 1 0 0 0 1]
    [0 2 0 0 1 0 0 1 2 0]
    [0 0 2 0 0 2 0 0 0 0]
    [0 0 0 2 0 0 2 1 0 1]
    [1 1 0 0 2 0 0 0 1 1]
    [0 0 2 0 0 2 0 0 0 0]
    [0 0 1 2 1 0 2 0 0 1]
    [0 0 0 0 0 1 0 2 0 0]
    [0 2 1 0 1 0 0 0 2 1]
    [1 0 0 1 1 0 1 0 0 2]
```

```
# Contradiction in row 7 so mat[5, 0] is not 0, and col 4 is a multiple of col 1.
# Same can be said with rows.
print original_mat
print ""

mat = copy(original_mat)
mat[5, 0] = 2
mat[0, 5] = 2
mat = compare_rows(mat, 1, 4)
mat = compare_columns(mat, 1, 4)
print mat
```

```
    [2 0 0 0 1 1 0 0 0 1]
    [0 2 0 0 1 0 0 1 1 0]
    [0 0 2 0 0 1 1 0 1 0]
    [0 0 0 2 0 0 1 1 0 1]
    [1 1 0 0 2 0 1 0 1 1]
    [1 0 1 0 0 2 0 1 0 0]
    [0 0 1 1 1 0 2 0 1 1]
    [0 1 0 1 0 1 0 2 0 0]
    [0 1 1 0 1 0 1 0 2 1]
    [1 0 0 1 1 0 1 0 1 2]

    [2 0 0 0 0 2 0 0 0 1]
    [0 2 0 0 2 0 0 0 1 0]
    [0 0 2 0 0 1 1 0 1 0]
    [0 0 0 2 0 0 1 1 0 1]
    [0 2 0 0 2 0 0 0 1 0]
    [2 0 1 0 0 2 0 1 0 0]
    [0 0 1 1 0 0 2 0 1 1]
    [0 0 0 1 0 1 0 2 0 0]
    [0 1 1 0 1 0 1 0 2 1]
    [1 0 0 1 0 0 1 0 1 2]
```

```
# Like deleting edges (0, 4), (4, 6), (1, 7), (4, 9)
h.delete_edges([(0, 4), (4, 6), (1, 7), (4, 9)])
print "The independence number of h is:", len(h.independent_set())
print "Trying to increase the lower bound by Theorems 3.2 and 3.6 gives:", increase_lower_bound(h)
```

evaluate

```
    The independence number of h is: 4
    Trying to increase the lower bound by Theorems 3.2 and 3.6 gives: 5
```