In [1]:
```
###################################
# This Sage worksheet is written by Chassidy Bozeman and Carolyn Reinh
# using code written by numerous other people.
# The focus of this worksheet is token addition and removal (TAR) and
# reconfiguration graphs for power domination in support of the paper
# Power domination reconfiguration
# by
# Beth Bjorkman, Chassidy Bozeman, Daniela Ferrero, Mary Flagg,
# Cheryl Grood, Leslie Hogben, Bonnie Jacob, Carolyn Reinhart
#
# To run computations, you must first enter the function F- cells at t
###################################
```

In [ ]:
```
# Functions
```

In [1]:
```
# F-1 load main minimum rank/maximum nullity and zero forcing function
# zero forcing, PSD forcing, and minimum rank code
# developed by S. Butler, L. DeLoss, J. Grout, H.T. Hall, J. LaGrange,
# updated and maintained by Jephian C.-H. Lin
load("minrank_aux-master/load_all.py")
load_all(local=True)
```

```
Compiling ./mr_JG-master/Zq_c.pyx...
xrange test failed: define xrange = range
Loading Zq_c.pyx...
ld: warning: duplicate -rpath '/private/var/tmp/sage-10.3-current/loca
l/lib' ignored
ld: warning: duplicate -rpath '/private/var/tmp/sage-10.3-current/loca
l/lib' ignored
ld: warning: duplicate -rpath '/private/var/tmp/sage-10.3-current/loca
l/lib' ignored
ld: warning: duplicate -rpath '/private/var/tmp/sage-10.3-current/loca
l/lib' ignored
ld: warning: -ld_classic is deprecated and will be removed in a future
release
ld: warning: passed two min versions (11.0, 11.0) for platform macOS. U
sing 11.0.
ld: warning: passed two min versions (11.0, 11.0) for platform macOS. U
sing 11.0.
Compiling ./mr_JG-master/zero_forcing_64.pyx...
Loading Zq.py...
Loading zero_forcing_64.pyx...
```

```
ld: warning: duplicate -rpath '/private/var/tmp/sage-10.3-current/local/lib' ignored
ld: warning: duplicate -rpath '/private/var/tmp/sage-10.3-current/local/lib' ignored
ld: warning: duplicate -rpath '/private/var/tmp/sage-10.3-current/local/lib' ignored
ld: warning: duplicate -rpath '/private/var/tmp/sage-10.3-current/local/lib' ignored
ld: warning: -ld_classic is deprecated and will be removed in a future release
ld: warning: passed two min versions (11.0, 11.0) for platform macOS. Using 11.0.
ld: warning: passed two min versions (11.0, 11.0) for platform macOS. Using 11.0.
Compiling ./mr_JG-master/zero_forcing_wavefront.pyx...
Loading zero_forcing_wavefront.pyx...
ld: warning: duplicate -rpath '/private/var/tmp/sage-10.3-current/local/lib' ignored
ld: warning: duplicate -rpath '/private/var/tmp/sage-10.3-current/local/lib' ignored
ld: warning: duplicate -rpath '/private/var/tmp/sage-10.3-current/local/lib' ignored
ld: warning: duplicate -rpath '/private/var/tmp/sage-10.3-current/local/lib' ignored
ld: warning: -ld_classic is deprecated and will be removed in a future release
ld: warning: passed two min versions (11.0, 11.0) for platform macOS. Using 11.0.
ld: warning: passed two min versions (11.0, 11.0) for platform macOS. Using 11.0.
Loading minrank.py...
Loading inertia.py...
Loading general_Lib.sage...
---sshow, multi_sshow, tuple_generator, minimal_graphs, empty_array, all_one_matrix, elementary_matrix, eigens_multi, sort_dictionary, has_minor, etc.
Loading oc_diag_analysis.sage...
---gZ_leq, find_gZ, find_EZ, diagonal_analysis, etc.
Loading xi_dict.py...
---SAPreduced_matrix, has_SAP, find_ZFloor, Zsap, etc.
Loading mu_dict.py...
---get_mu_from_dict, find_mu, etc.
Loading SXP.sage...
This code contains extra copy of Z_game, Zell_game, Zplus_game, for the completeness of Zsap_game program.
Loading matrix_forcing.py...
```

In [4]:
```
# F-Power: Power domination TAR Functions
# enter this cell in order to run computations in this section

def get_minimal_subsets(sets):
    sets = sorted(map(set, sets), key=len)
```

```
    minimal_subsets = []
    for s in sets:
        if not any(minimal_subset.issubset(s) for minimal_subset in mi
            minimal_subsets.append(s)
    return minimal_subsets

# Test for power dominating set adapted from code by Brian Wissman
# input: a graph G and a subset of its vertices V
# output: true/false depending if V is a power dominating set of G
def isPowerDominatingSet(G,V):
    N=[]
    for i in V:
        N+=G.neighbors(i)
    NVert=sorted(set(N+list(V)))
    A=zerosgame(G,NVert)
    if len(A)==G.order():
        return True
    else:
        return False

# input: a graph G
# output: the power domination number of G
def PowerDom(G):
    V = G.vertices()
    for i in range(1,len(V)+1):
        S = subsets(V,i)
        for s in S:
            if isPowerDominatingSet(G,s):
                p=len(s)
                return p
# Power dominating sets

# All power dominating sets of G of size k
# input: a graph G and a positive integer k
# output: a list of all power dominating sets of G of size k
def PDsets(G,k):
    ord=G.order()
    V = G.vertices()
    AllSubs = set(Subsets(V,k))
    PD=[]
    for s in AllSubs:
        if isPowerDominatingSet(G,s):
            PD.append(s)
    return PD

# All power dominating sets of G of size <= k
# input: a graph G and a positive integer k
# output: a list of all power dominating sets of G of size <= k
def PD_up_to_size_k(G,k):
    S=[]
    for i in [0..k]:
        PD_size_i=PDsets(G,i)
```

```
        S=S+PD_size_i
    return S

# Minimal power dominating sets of G
# input: a graph G
# output: a list of all minimal power dominating sets of G \
def PD_min_sets(G):
    ord = G.order()
    PD = PD_up_to_size_k(G,ord)
    mPD = get_minimal_subsets(PD)
    return mPD

# Power domination number and upper power domination number of G
# input: a graph G
# output: the upper minimal zero forcing number
def PSDbar(G):
    min_list = PD_min_sets(G)
    PDbar = max(len(x) for x in min_list)
    PD = min(len(x) for x in min_list)
    return PD, PDbar

# Power domination TAR reconfiguration graph
# adapted from code by Chassidy Bozeman
# input: A graph G and a positive integer k
# output: The power domination TAR reconfiguration graph on power domi

def TAR_reconfig(G,k):
    S=PD_up_to_size_k(G,k)     # creates a list of all zf sets up to s
    H=Graph()    # creates empty graph
    H.add_vertices(S)     # add zf sets up to size k to the vertices o
    # The following part determines if the size of two skew forcing se
    # by one and if one is a subset of the other. If both are true, an
    for i in range(len(S)):
        for j in range(i+1, len(S)):
            if len(S[i])-len(S[j]) in {-1,1}:
                    if set(S[i]).issubset(set(S[j])):
                        H.add_edge(S[i],S[j])
                    else:
                        if set(S[j]).issubset(set(S[i])):
                            H.add_edge(S[i],S[j])
    return H

# Token Jumping = Token Exchange
def TE_reconfig(G,k):
    S=PDsets(G,k)
    #creates a list of all dominating sets of size k.
    H=Graph()
    #creates empty graph
    H.add_vertices(S)
    # add power dom sets of size k to the vertices of H
    # The part below is for token exchange. If two power dominating se
    # same size and they differ by one element, then an edge is added
```

```
    for i in range(len(S)):
        for j in range(i+1, len(S)):
            if len(S[i])==len(S[j]):
                if len(set(S[i]).difference(set(S[j])))==1:
                    H.add_edge(S[i],S[j])
    return H
```

In [ ]:
```
# Uniqueness of Some Token Addition and Removal Reconfiguration and Co
```

In [5]:
```
def TAR_reconfig_comparison(G):
    examples = set([])
    n=G.order()
    GpG=PowerDom(G)
    H=TAR_reconfig(G,n)
    for g in graphs(n):
        Gpg=PowerDom(g)
        if GpG==Gpg:
            K=TAR_reconfig(g,n)
            if H.is_isomorphic(K):
                if not g.is_isomorphic(G):
                    print('Found one')
                    examples.add(g.graph6_string())
    return examples
```

In [6]:
```
G=graphs.CompleteBipartiteGraph(3,3)
TAR_reconfig_comparison(G)
```

Out[6]:
```
set()
```

In [7]:
```
G=graphs.CompleteBipartiteGraph(3,4)
TAR_reconfig_comparison(G)
```

Out[7]:
```
set()
```

In [31]:
```
G=graphs.CompleteBipartiteGraph(3,5)
TAR_reconfig_comparison(G)
```

Out[31]:
```
set()
```

In [32]:
```
G=graphs.CompleteBipartiteGraph(4,4)
TAR_reconfig_comparison(G)
```

Out[32]:
```
set()
```

In [10]:
```
G=graphs.CompleteBipartiteGraph(4,5)
TAR_reconfig_comparison(G)
```

Out[10]:
```
set()
```

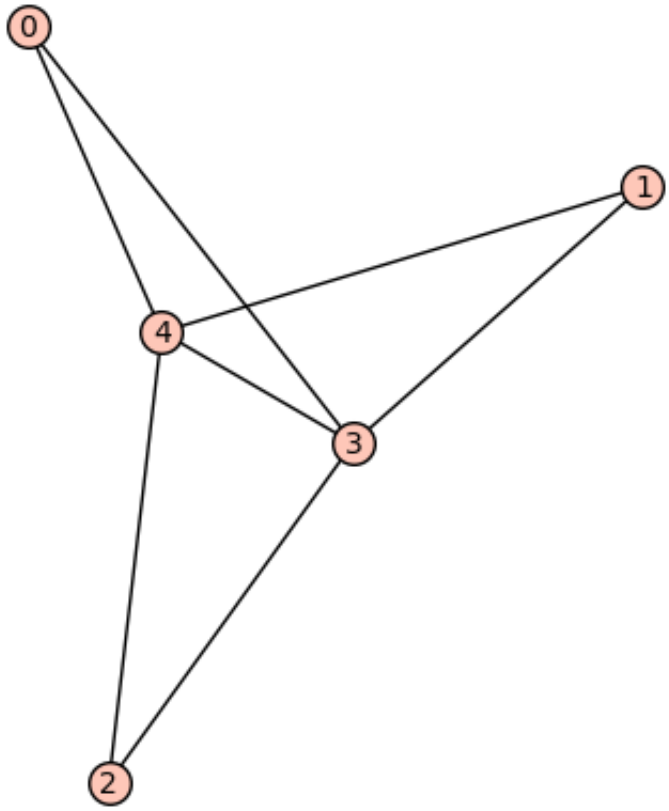
In [ ]:
```
# As a comparision, this program finds the one graph G=K_2,3+e that is
```

In [8]:
```
G=graphs.CompleteBipartiteGraph(2,3)
TAR_reconfig_comparison(G)
```

```
Found one
```

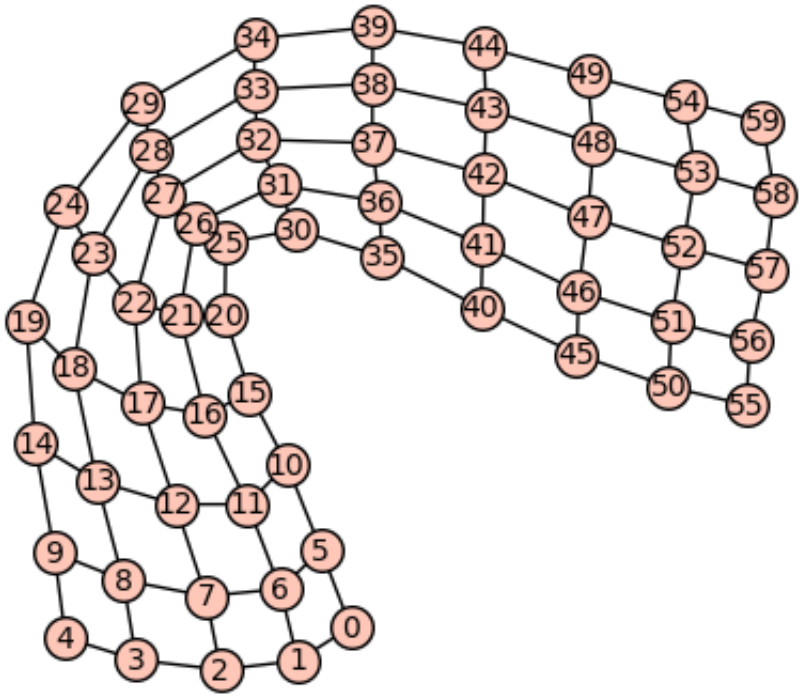
Out[8]: {'DF{'}

In [9]:
```
g=Graph("DF{")
show(g)
```

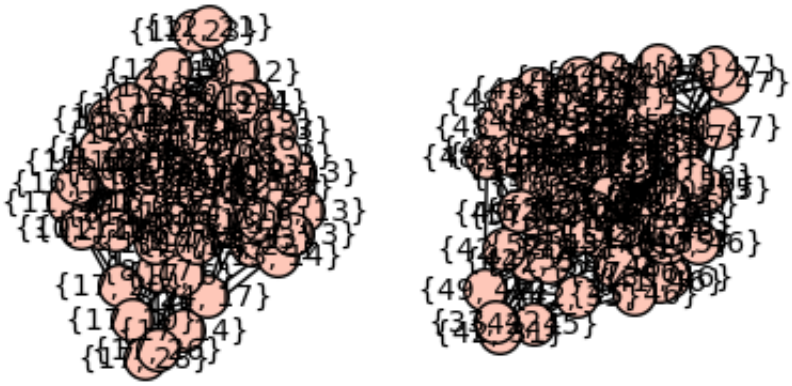


In [ ]:
```
# Token Jumping Reconfiguration for Grid Graphs
# Proposition 5.12
```

In [22]:
```
# 5 x 12 grid
p12=graphs.PathGraph(12)
p=graphs.PathGraph(5)
g=p12.cartesian_product(p)
g.relabel()
show(g)
```



In [19]:
```
H=TE_reconfig(g,2)
show(H)
```



In [20]:
```
H.vertices()
```

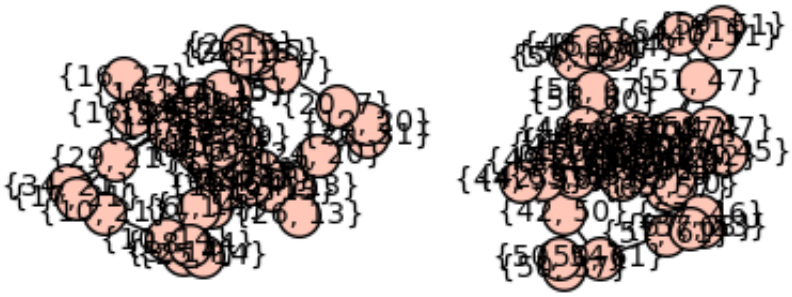
Out[20]:
```
[{9, 13},
 {52, 45},
 {4, 5},
 {16, 7},
 {9, 5},
 {8, 5},
 {42, 35},
 {8, 11},
 {0, 11},
 {19, 13},
 {4, 13},
 {58, 55},
 {50, 59},
 {36, 47},
 {18, 13},
```

```
{5, 7},
{52, 55},
{17, 26},
{9, 6},
{45, 46},
{58, 43},
{17, 6},
{24, 9},
{6, 15},
{43, 52},
{53, 46},
{48, 53},
{1, 12},
{17, 20},
{54, 55},
{48, 58},
{56, 46},
{12, 15},
{1, 9},
{9, 19},
{17, 28},
{42, 31},
{38, 47},
{9, 18},
{19, 4},
{8, 19},
{50, 47},
{13, 22},
{48, 43},
{1, 7},
{48, 39},
{8, 13},
{56, 35},
{50, 35},
{0, 15},
{40, 47},
{58, 52},
{48, 49},
{18, 7},
{40, 50},
{56, 58},
{0, 9},
{1, 11},
{3, 5},
{56, 47},
{37, 46},
{10, 7},
{56, 53},
{56, 59},
{41, 52},
{18, 3},
{13, 6},
```

```
{56, 41},
{40, 51},
{11, 15},
{2, 11},
{9, 12},
{3, 7},
{20, 5},
{41, 46},
{3, 13},
{44, 53},
{44, 54},
{54, 47},
{54, 39},
{40, 46},
{14, 7},
{16, 1},
{24, 13},
{42, 45},
{17, 10},
{46, 55},
{24, 3},
{57, 46},
{50, 58},
{56, 54},
{48, 57},
{50, 54},
{40, 55},
{48, 51},
{24, 17},
{16, 11},
{50, 46},
{49, 42},
{59, 44},
{8, 7},
{4, 7},
{6, 7},
{19, 12},
{52, 53},
{42, 53},
{41, 50},
{3, 6},
{1, 4},
{43, 54},
{11, 6},
{0, 7},
{51, 52},
{1, 15},
{3, 12},
{50, 52},
{19, 3},
{48, 54},
{50, 53},
```

```
 {44, 47},
 {48, 59},
 {8, 17},
 {40, 56},
 {58, 39},
 {58, 47},
 {11, 5},
 {12, 23},
 {17, 14},
 {33, 42},
 {58, 44},
 {51, 46},
 {49, 52},
 {58, 51},
 {42, 39},
 {56, 52},
 {12, 5},
 {2, 7},
 {9, 7},
 {16, 5},
 {11, 22},
 {35, 46},
 {8, 1},
 {1, 3},
 {57, 52},
 {12, 21},
 {5, 15},
 {48, 37},
 {10, 11},
 {0, 3},
 {52, 54},
 {59, 52},
 {42, 51},
 {2, 13},
 {1, 20},
 {13, 14},
 {48, 44},
 {51, 54},
 {11, 20}]
```

In [21]:
```
# 6 x 12 grid
p12=graphs.PathGraph(12)
p=graphs.PathGraph(6)
g=p12.cartesian_product(p)
g.relabel()
H=TE_reconfig(g,2)
show(H)
H.is_connected()
```



Out[21]: False

In [26]:
```
# 7 x 12 grid
p12=graphs.PathGraph(12)
p=graphs.PathGraph(7)
g=p12.cartesian_product(p)
g.relabel()
H=TE_reconfig(g,2)
H.is_connected()
```

Out[26]: False

In [27]:
```
# 8 x 12 grid
p12=graphs.PathGraph(12)
p=graphs.PathGraph(8)
g=p12.cartesian_product(p)
g.relabel()
H=TE_reconfig(g,2)
H.is_connected()
```

Out[27]: False

In [ ]:
```

```