Classification of Sterk Elliptic Subdiagrams

October 24, 2023

[1]:
```
from IPython.display import Math
import numpy as np
import pandas as pd
from IPython.display import HTML
from collections import Counter
from sage.modules.free_module_integer import IntegerLattice

H = IntegralLattice("H")
E8 = IntegralLattice("E8").twist(-1)
E82 = E8.twist(2)
H2 = H.twist(2)
```

[2]:
```
def Coxeter_Diagram(M):
    nverts = M.ncols()
    # print(str(nverts) + " vertices")
    G = Graph()
    vertex_labels = dict();
# plot_coxeter_diagram(G)

    vertex_colors = {
        '#F8F9FE': [], # white
        '#BFC9CA': [], # black
    }

    for i in range(nverts):
        for j in range(nverts):
            mij = M[i, j]
            if i == j:
                if mij == -2:
                    vertex_colors["#F8F9FE"].append(i) # white
                    continue
                if mij == -4:
                    vertex_colors["#BFC9CA"].append(i) # black
                    continue
                continue
            if mij > 0:
                G.add_edge(i, j, str(mij) )
                continue
```

```
    assert len( vertex_colors["#F8F9FE"]) + len( vertex_colors["#BFC9CA"]) ==␣
 ↪nverts
    G.vertex_colors = vertex_colors
    return G

def plot_coxeter_diagram(G, v_labels, pos={}):
    n = len( G.vertices() )
    vlabs = {v: k for v, k in enumerate(v_labels)}
    if pos == {}:
        display(G.plot(
            edge_labels=True,
            vertex_labels = vlabs,
            vertex_size=200,
            vertex_colors = G.vertex_colors
        ))
    else:
        display(G.plot(
            edge_labels=True,
            vertex_labels = vlabs,
            vertex_size=200,
            vertex_colors = G.vertex_colors,
            pos = pos
        ))

def root_intersection_matrix(vectors, labels, bil_form):
    n = len(vectors)
    M = zero_matrix(ZZ, n)
    nums = Set(range(n))
    for i in range(n):
        for j in range(n):
            M[i, j] = bil_form( vectors[i], vectors[j] )

    # print("Diagonal entries/square norms: ")
    # display(M.diagonal())

    # Labels!


    # df = pd.DataFrame(M, columns=labels, index=labels)
    # display(HTML(df.to_html()))

    # Must be symmetric
    assert M.is_symmetric()

    # Must have -2 or -4 on the diagonal
    s = Set( M.diagonal() )
    assert s in Subsets( Set( [-2, -4] ) )
```

```
    # Diagonals should be square norms of vectors
    for i in range(n):
        assert M[i, i] == bil_form(vectors[i], vectors[i])



    return M

def is_elliptic_matrix(M):
    return (-1 * M).is_positive_definite()

def is_parabolic_matrix(M):
    return (-1 * M).is_positive_semidefinite()

def roots_from_subgraph(H):
    return [V[index] for index in H.vertices()]
```

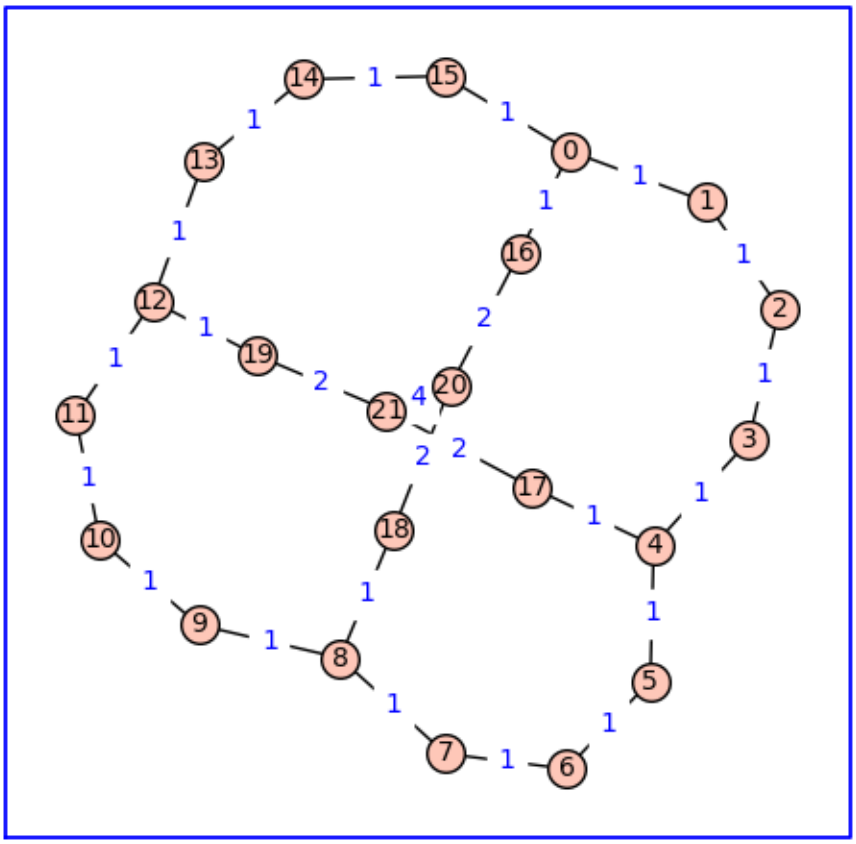
[3]:
```
L_20_2_0 = H.direct_sum(H2).direct_sum(E8).direct_sum(E8)

dot = lambda x,y : x * L_20_2_0.gram_matrix() * y
nm = lambda x: dot(x, x)

Gram_L_20_2_0 = L_20_2_0.gram_matrix()
Gram_L_20_2_0.subdivide([2, 4, 12], [2, 4, 12])
L_20_2_0_dual_changeofbasis = Gram_L_20_2_0.inverse()
L_20_2_0_dual_changeofbasis.subdivide([2, 4, 12], [2, 4, 12])

e,f ,ep,fp, a1,a2,a3,a4,a5,a6,a7,a8, a1t,a2t,a3t,a4t,a5t,a6t,a7t,a8t = L_20_2_0.
 ↪basis()
eb,fb, epb,fpb, w1,w2,w3,w4,w5,w6,w7,w8, w1t,w2t,w3t,w4t,w5t,w6t,w7t,w8t =␣
 ↪L_20_2_0_dual_changeofbasis.columns()

# display(Math("$(18, 2, 0)="), Gram_L_20_2_0)
# display(Math("$(18, 2, 0)^{\\vee} =$"), L_20_2_0_dual_changeofbasis)

# The primes are the image of the diagonal embedding from E_8(2)
a1p = a1 + a1t
a2p = a2 + a2t
a3p = a3 + a3t
a4p = a4 + a4t
a5p = a5 + a5t
a6p = a6 + a6t
a7p = a7 + a7t
a8p = a8 + a8t

w1p = w1 + w1t
```

```
w2p = w2 + w2t
w3p = w3 + w3t
w4p = w4 + w4t
w5p = w5 + w5t
w6p = w6 + w6t
w7p = w7 + w7t
w8p = w8 + w8t

MS      = [e,f,   ep,fp,   a1,a2,a3,a4,a5,a6,a7,a8,␣
 ↪a1t,a2t,a3t,a4t,a5t,a6t,a7t,a8t]
MS_dual = [eb,fb, epb,fpb, w1,w2,w3,w4,w5,w6,w7,w8,␣
 ↪w1t,w2t,w3t,w4t,w5t,w6t,w7t,w8t]

VS      = [ep,fp,   a1,a2,a3,a4,a5,a6,a7,a8, a1t,a2t,a3t,a4t,a5t,a6t,a7t,a8t]
VS_dual = [epb,fpb, w1,w2,w3,w4,w5,w6,w7,w8, w1t,w2t,w3t,w4t,w5t,w6t,w7t,w8t]
```

[4]:
```
# Root vectors for (18, 2, 0), roots taken from above, v_i are according to␣
 ↪numerical labeling above

v1 = a8t
v2 = ep + fp + w1 + w8t
v3 = a1
v4 = a3
v5 = a4
v6 = a5
v7 = a6
v8 = a7
v9 = a8
v10 = ep + fp + w8 + w1t
v11 = a1t
v12 = a3t
v13 = a4t
v14 = a5t
v15 = a6t
v16 = a7t

v17 = ep + w8t
v18 = a2
v19 = ep + w8
v20 = a2t

v21 = fp - ep
v22 = 5 ep + 3 fp + 2 w2 + 2 w2t

V = [v1, v2, v3, v4, v5, v6, v7, v8, v9, v10, v11, v12, v13, v14, v15, v16,␣
 ↪v17, v18, v19, v20, v21, v22]
labels = [f"$v_{ {r + 1} }$" for r in range( len(V) )]
```

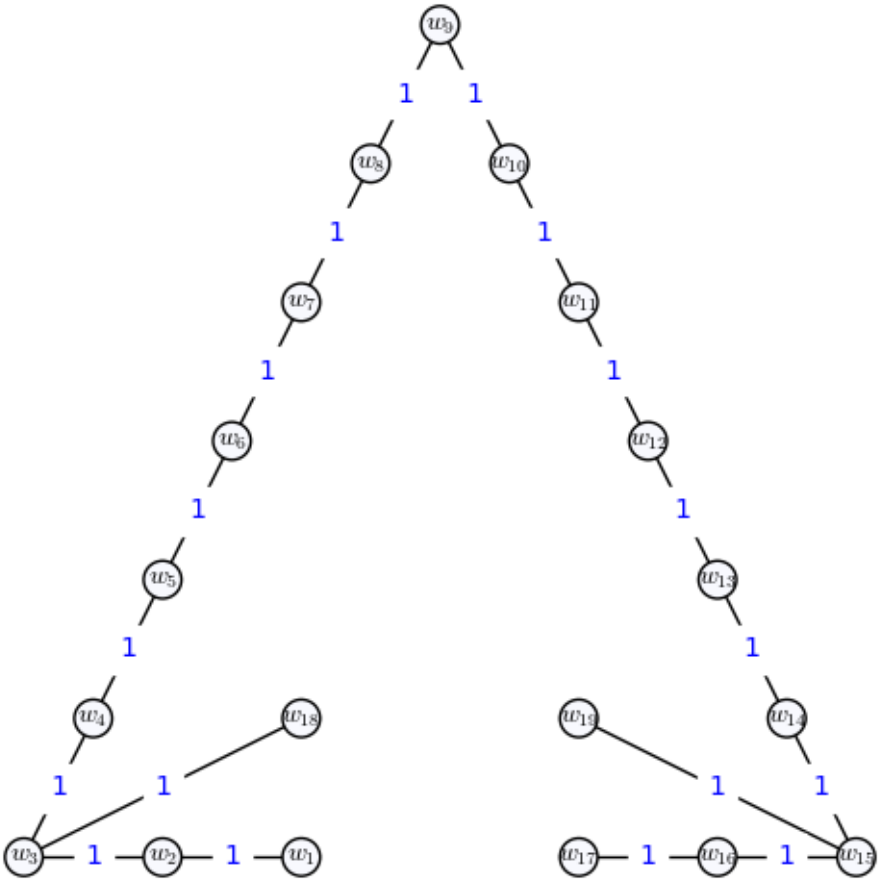
```
MV = root_intersection_matrix(V, labels = labels, bil_form=dot)

# df = pd.DataFrame(MV, columns=labels, index=labels)
# HTML(df.to_html())
G = Coxeter_Diagram(MV)
G.plot(edge_labels=True, graph_border=True)
```

[4]:

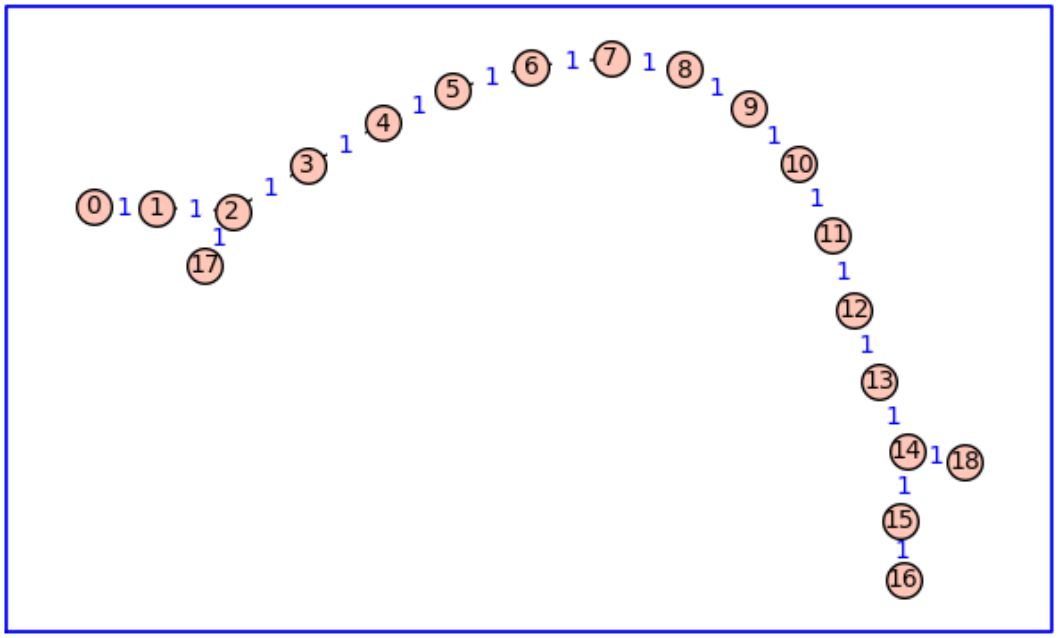


[5]:
```
# Root vectors for (18, 0, 0), roots taken from above, w_i are according to␣
 ↪numerical labeling above

w1 = a1
w2 = a3
w3 = a4
w4 = a5
w5 = a6
w6 = a7
w7 = a8
w8 = w8 + e
```

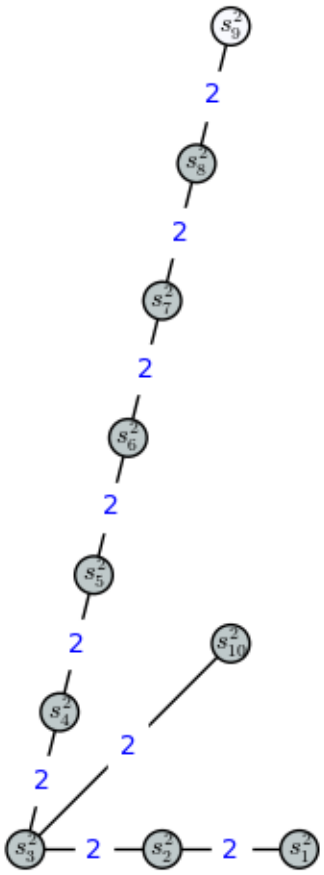
```
w9 = f- e
w10 = w8t + e
w11 = a8t
w12 = a7t
w13 = a6t
w14 = a5t
w15 = a4t
w16 = a3t
w17 = a1t
w18 = a2
w19 = a2t

W = [w1, w2, w3, w4, w5, w6, w7, w8, w9, w10, w11, w12, w13, w14, w15, w16,␣
 ↪w17, w18, w19]
MW = root_intersection_matrix(W, labels = [f"$w_{ {r + 1} }$" for r in range(␣
 ↪len(W) )], bil_form=dot)


G = Coxeter_Diagram(MW)
plot_coxeter_diagram(
    G,
    v_labels = [f"$w_{ {i + 1} }$" for i in range( 19 )],
    pos = {
        0: [-4, 0],
        1: [-8, 0],
        2: [-12, 0],
        3: [-10, 4],
        4: [-8, 8],
        5: [-6, 12],
        6: [-4, 16],
        7: [-2, 20],
        8: [0, 24],
        9: [2, 20],
        10: [4, 16],
        11: [6, 12],
        12: [8, 8],
        13: [10, 4],
        14: [12, 0],
        15: [8, 0],
        16: [4, 0],
        17: [-4, 4],
        18: [4, 4]
    }
)
G.plot(edge_labels=True, graph_border=True)
```


w_9
w_8
w_{10}
w_7
w_{11}
w_6
w_{12}
w_5
w_{13}
w_4
w_{18}
w_{19}
w_{14}
w_3
w_2
w_1
w_{17}
w_{16}
w_{15}

[5]:


0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18

[6]:
```
# Sterk 2

s2_1 = w1 + w17
s2_2 = w2 + w16
s2_3 = w3 + w15
s2_4 = w4 + w14
s2_5 = w5 + w13
s2_6 = w6 + w12
s2_7 = w7 + w11
s2_8 = w8 + w10
s2_9 = w9
s2_10 = w18 + w19

S2 = [s2_1, s2_2, s2_3, s2_4, s2_5, s2_6, s2_7, s2_8, s2_9, s2_10]
MS2 = root_intersection_matrix(S2, labels = [f"$s^2_{ {r + 1} }$" for r in␣
 ↪range( len(S2) )], bil_form=dot )

G_Sterk_2 = Coxeter_Diagram(MS2)
plot_coxeter_diagram(
    G_Sterk_2,
    v_labels = [f"$s^2_{ {i + 1} }$" for i in range( 22 )],
    pos = {
        0: [0, 0],
        1: [-4, 0],
        2: [-8, 0],
        3: [-7, 4],
        4: [-6, 8],
        5: [-5, 12],
        6: [-4, 16],
        7: [-3, 20],
        8: [-2, 24],
        9: [-2, 6]
    }
)
```



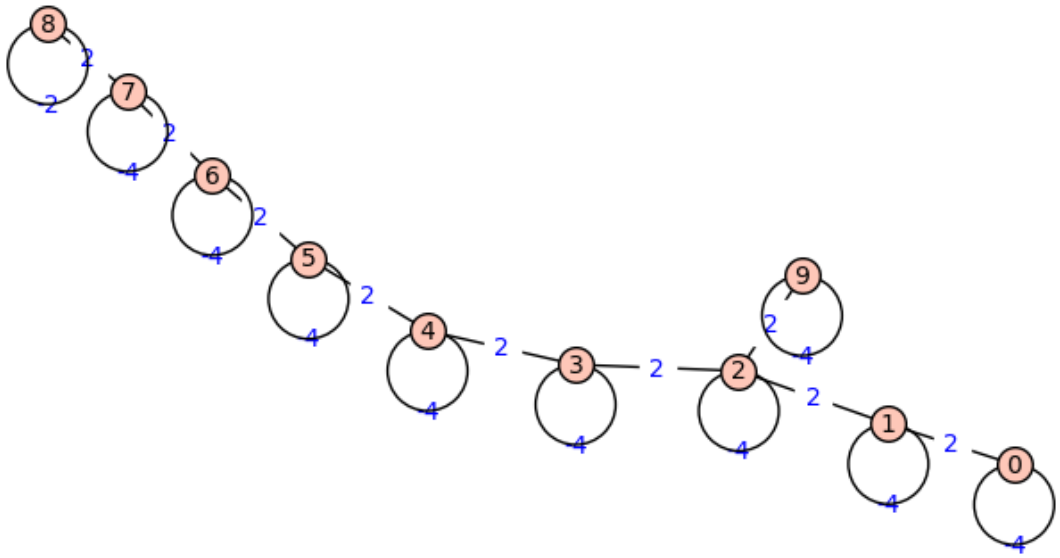
[7]:
```
positions = {
    0: [0, 0],
    1: [-4, 0],
    2: [-8, 0],
    3: [-7, 4],
    4: [-6, 8],
    5: [-5, 12],
    6: [-4, 16],
    7: [-3, 20],
    8: [-2, 24],
    9: [-2, 6]
}
```

[8]:
```
def matrix_to_graph(M):
    nverts = M.ncols()
    G = Graph(loops=True)
    for i in range(nverts):
        for j in range(nverts):
            mij = M[i, j]
```

```
            if mij == 0:
                continue
            G.add_edge(i, j, str(mij) )
    return G

def graph_to_matrix(G):
    verts = G.vertices()
    n = len(verts)
    M = zero_matrix(ZZ, n)
    for e in G.edges():
        M[ e[0], e[1] ] = e[2]
        M[ e[1], e[0] ] = e[2]
    return M

def sterk_2_subgraph_to_matrix(H):
    H_roots = [ S2[index] for index in H.vertices() ]
    return root_intersection_matrix(H_roots, labels = H.vertices(),␣
 ↪bil_form=dot)
```

[9]:
```
G_Sterk_2_loops = matrix_to_graph(MS2)
G_Sterk_2_loops.plot(edge_labels=True)
```

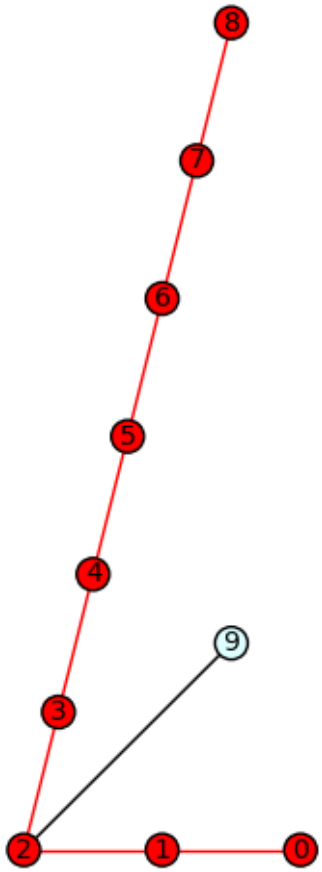
[9]:



[10]:
```
Sterk_B9 = [0,1,2,3,4,5,6,7,8]

H_B9 = G_Sterk_2_loops.subgraph(Sterk_B9)

H_B9_roots = [S2[index] for index in Sterk_B9]
```

```
M_Sterk_B9 = root_intersection_matrix(H_B9_roots, labels = H_B9.vertices(),␣
 ↪bil_form=dot)

show(H_B9.vertices())

assert is_elliptic_matrix(M_Sterk_B9)

red_edges = [ e for e in G_Sterk_2_loops.edges() if e in H_B9.edges() ]
red_vertices = [ v for v in G_Sterk_2_loops.vertices() if v in H_B9.vertices() ]

display(G_Sterk_2_loops.plot(
    vertex_size=150,
    edge_colors={'red': red_edges},
    vertex_color='lightcyan',
    vertex_colors={'red': red_vertices},
    pos=positions
))
show(M_Sterk_B9)
show(M_Sterk_B9.dimensions())
```

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



$$\begin{pmatrix} -4 & 2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 2 & -4 & 2 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 2 & -4 & 2 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 2 & -4 & 2 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 2 & -4 & 2 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 2 & -4 & 2 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 2 & -4 & 2 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 2 & -4 & 2 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 2 & -2 \end{pmatrix}$$

$(9, 9)$

[11]:
```
# Type A

def graph_A_n(n):
    return matrix_to_graph( matrix_A_n(n) )

def matrix_A_n(n):
    return IntegralLattice(f"A{n}").twist(-1).gram_matrix()

def graph_A_n_2(n):
    return matrix_to_graph( matrix_A_n_2(n) )

def matrix_A_n_2(n):
    return IntegralLattice(f"A{n}").twist(-2).gram_matrix()

# Type B

def graph_B_n_2(n):
    m=n-1
    G = Graph(loops=True)
    for i in range(m):
        G.add_edge(i, i, -4)
        G.add_edge(i, i+1, 2)
    G.add_edge(m, m, -2)
    return G

def matrix_B_n_2(n):
    return graph_to_matrix( graph_B_n_2(n) )

# Type C

def graph_C_n_2(n):
    m=n-1
    Gp = Graph(loops=True)
    for i in range(m):
```

```
        Gp.add_edge(i, i, -2)
        Gp.add_edge(i, i+1, 2)
    Gp.add_edge(m, m, -4)
    return Gp

def matrix_C_n_2(n):
    return graph_to_matrix( graph_C_n_2(n) )

# Type D

def graph_D_n(n):
    return matrix_to_graph(matrix_D_n(n))

def matrix_D_n(n):
    return IntegralLattice(f"D{n}").twist(-1).gram_matrix()

def graph_D_n_2(n):
    return matrix_to_graph(matrix_D_n_2(n))

def matrix_D_n_2(n):
    return IntegralLattice(f"D{n}").twist(-2).gram_matrix()

# Type E6

def graph_E_6():
    return matrix_to_graph(matrix_E_6())

def matrix_E_6():
    return IntegralLattice("E6").twist(-1).gram_matrix()

def graph_E_6_2():
    return matrix_to_graph(matrix_E_6_2())

def matrix_E_6_2():
    return IntegralLattice("E6").twist(-2).gram_matrix()

# Type E7

def graph_E_7():
    return matrix_to_graph(matrix_E_7())

def matrix_E_7():
    return IntegralLattice("E7").twist(-1).gram_matrix()

def graph_E_7_2():
    return matrix_to_graph(matrix_E_7_2())
```

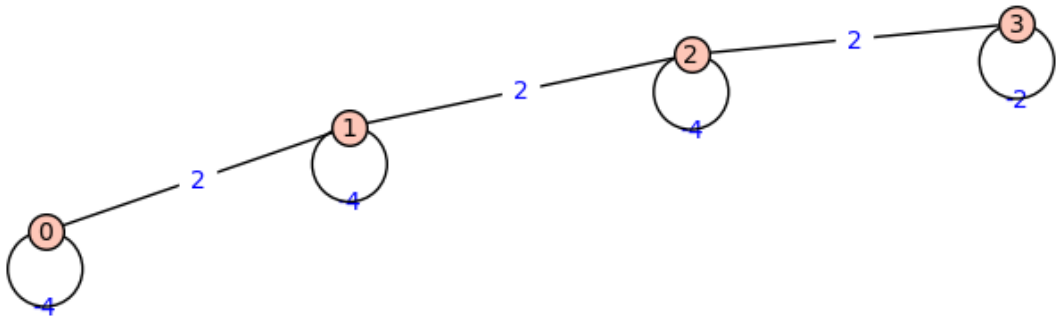
```
def matrix_E_7_2():
    return IntegralLattice("E7").twist(-2).gram_matrix()

# Type E8

def graph_E_8():
    return matrix_to_graph(matrix_E_8())

def matrix_E_8():
    return IntegralLattice("E8").twist(-1).gram_matrix()

def graph_E_8_2():
    return matrix_to_graph(matrix_E_8_2())

def matrix_E_8_2():
    return IntegralLattice("E8").twist(-2).gram_matrix()

def graph_G_2():
    G = Graph(loops=True)
    G.add_edge(0, 1, 2)
    G.add_edge(0, 0, -2)
    G.add_edge(1, 1, -4)
    return G

def matrix_G_2():
    return graph_to_matrix(graph_G_2())

def get_all_rank_n_types(n):
    if n == 1:
        return [
            (f"A_{n}(2)" ,matrix_A_n_2(1))
        ]
    if n == 2:
        return [
            (f"A_{n}(2)", matrix_A_n_2(2)),
            (f"G_{2}", matrix_G_2())
        ]
    else:
        Ms = [
            (f"A_{n}(2)", matrix_A_n_2(n)),
            (f"B_{n}(2)", matrix_B_n_2(n)),
            (f"C_{n}(2)", matrix_C_n_2(n)),
            (f"D_{n}(2)", matrix_D_n_2(n))
        ]
        if n == 6:
            Ms.append(
                (f"E_6(2)", matrix_E_6_2() )
```

```
            )
        elif n == 7:
            Ms.append(
                (f"E_7(2)", matrix_E_7_2() )
            )
        elif n == 8:
            Ms.append(
                (f"E_8(2)", matrix_E_8_2() )
            )
        return Ms

type_A_graphs = [ graph_A_n_2(n) for n in range(11) ]
type_B_graphs = [ graph_B_n_2(n) for n in range(11) if n > 2]
type_C_graphs = [ graph_C_n_2(n) for n in range(11) if n > 2]
type_D_graphs = [ graph_D_n_2(n) for n in range(11) if n > 3]
type_E_graphs = [ graph_E_6_2(), graph_E_7_2(), graph_E_8_2() ]

type_A_matrices = [ matrix_A_n_2(n) for n in range(11) ]
type_B_matrices = [ matrix_B_n_2(n) for n in range(11) if n > 2 ]
type_C_matrices = [ matrix_C_n_2(n) for n in range(11) if n > 2]
type_D_matrices = [ matrix_D_n_2(n) for n in range(11) if n > 3]
type_E_matrices = [ matrix_E_6_2(), matrix_E_7_2(), matrix_E_8_2() ]

# show(type_A_matrices)
# show(type_E_matrices)
# show( get_all_rank_n_types(6) )
```

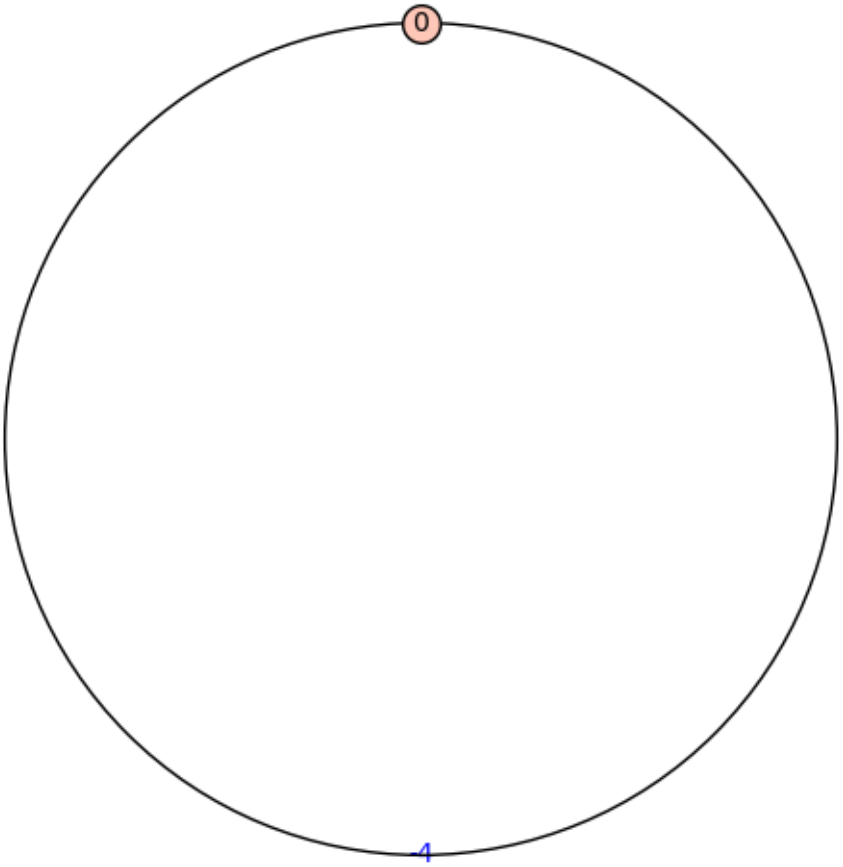
[12]:
```
assert matrix_B_n_2(9) == M_Sterk_B9
assert H_B9.is_isomorphic(graph_B_n_2(9))
```

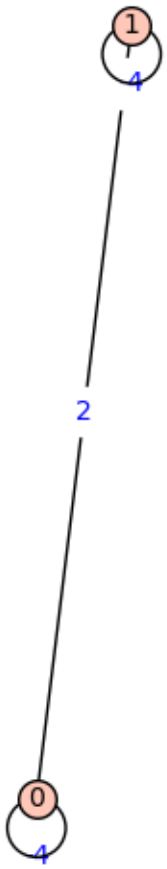
[13]:
```
graph_B_n_2(4).plot(edge_labels=True)
```

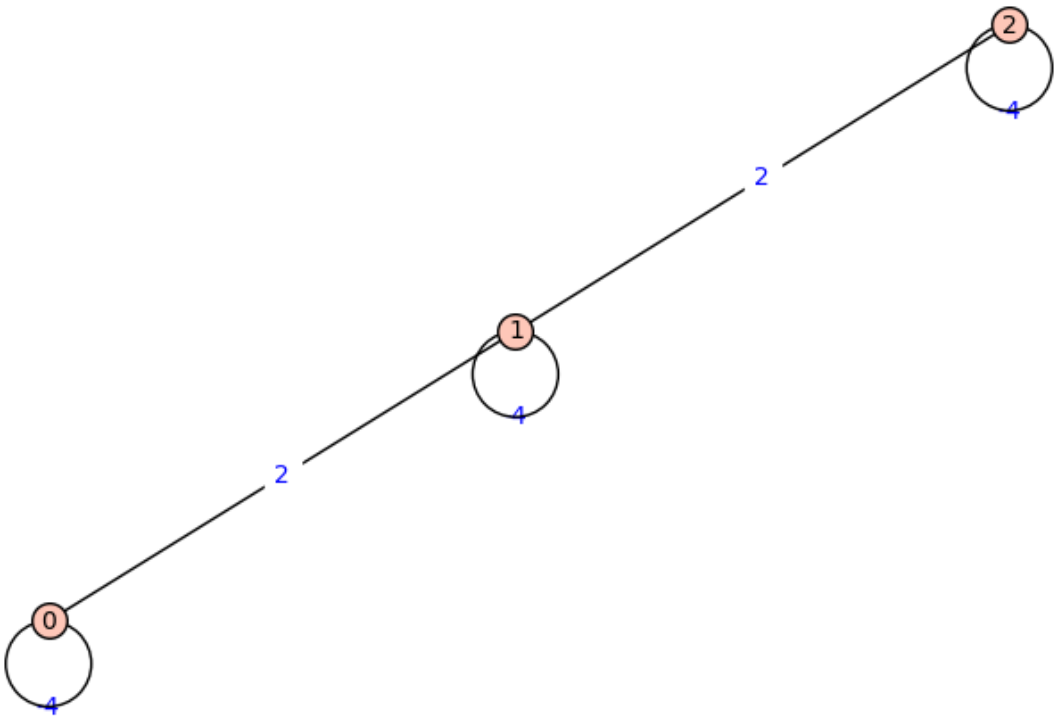
[13]:

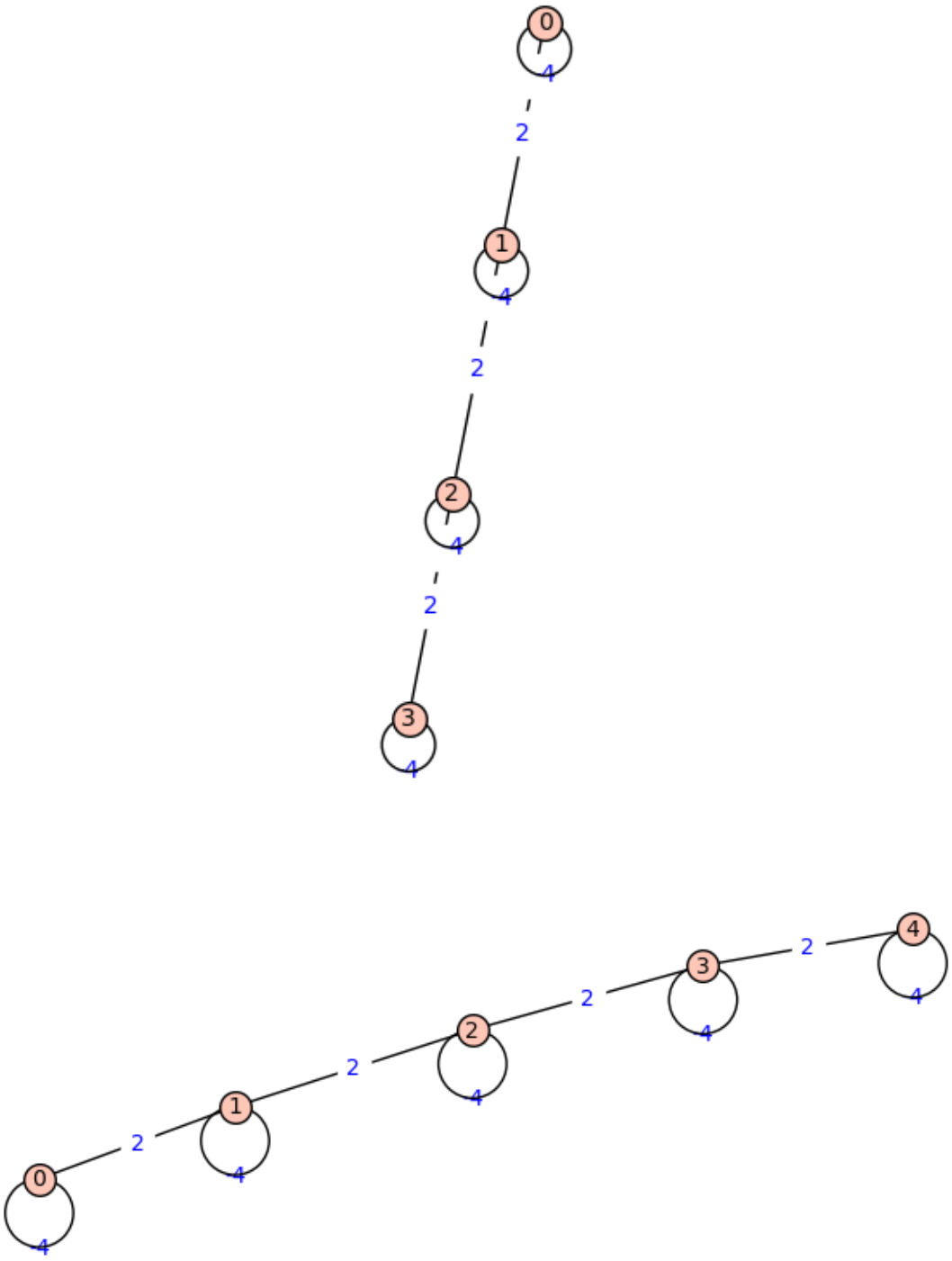


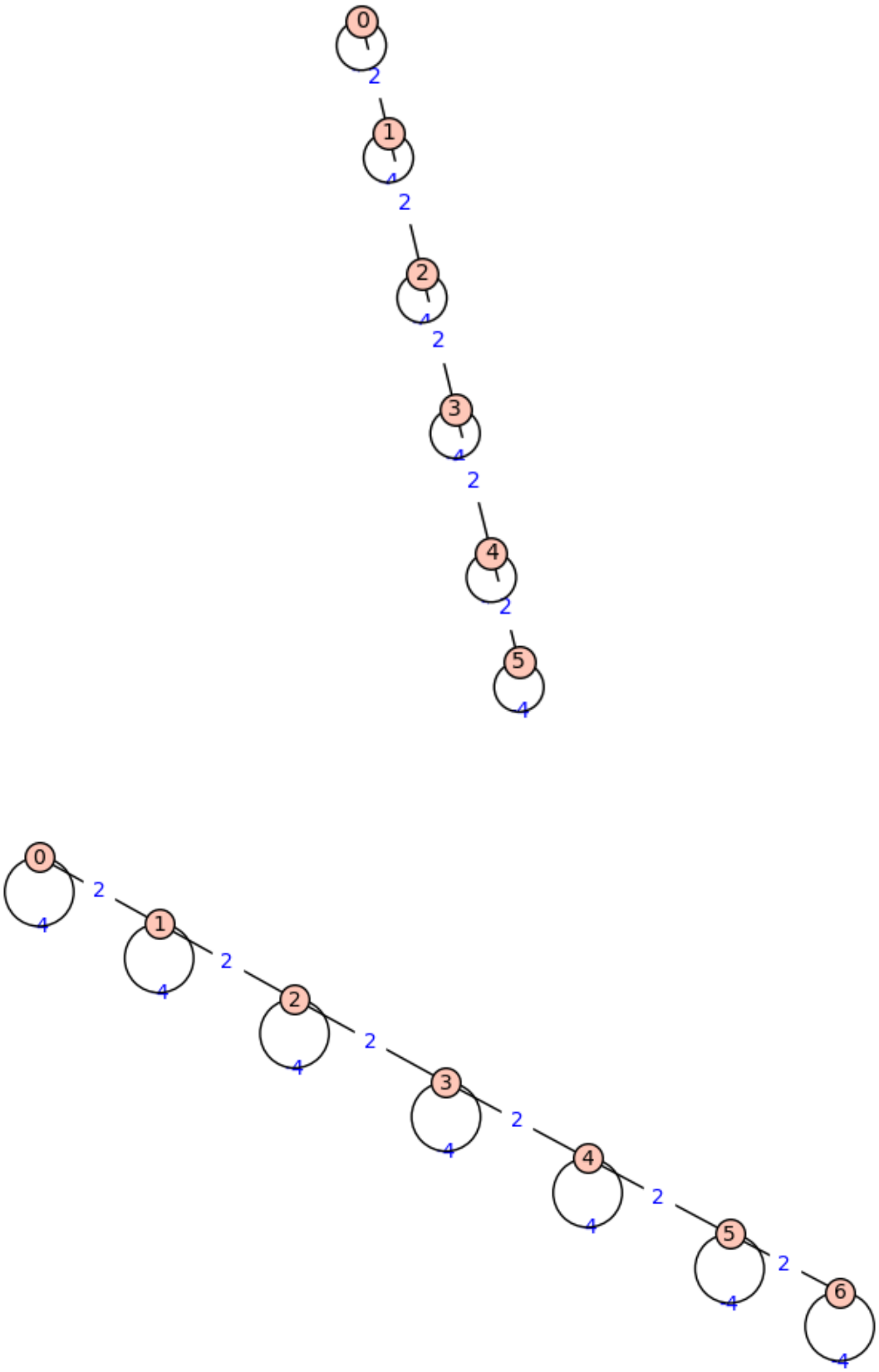
[14]:
```
# Type A_n
for g in type_A_graphs:
    show( g.plot(edge_labels=True) )
```

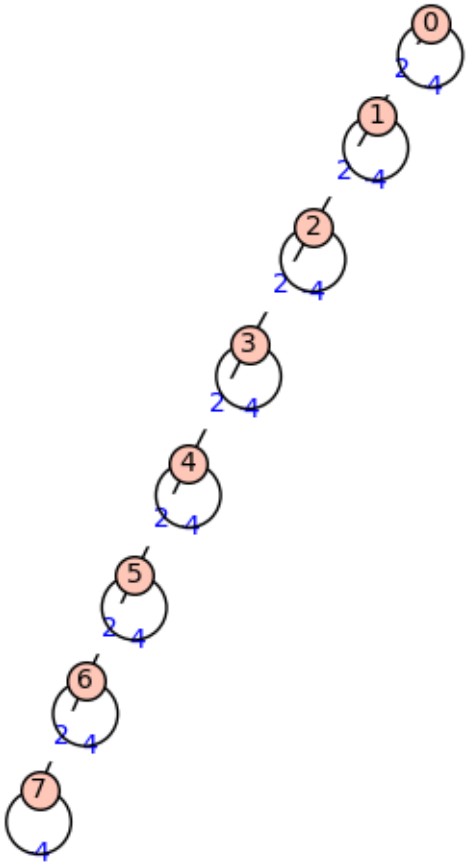

0
-4

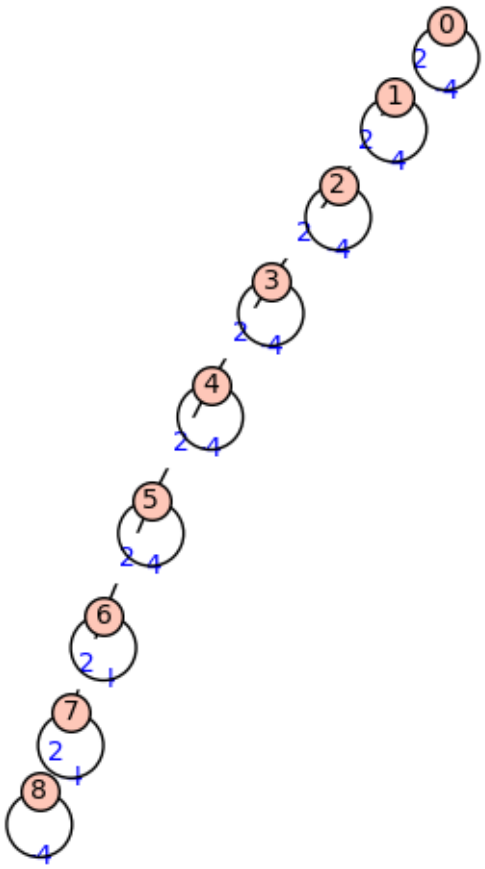

1
4
2
0
4

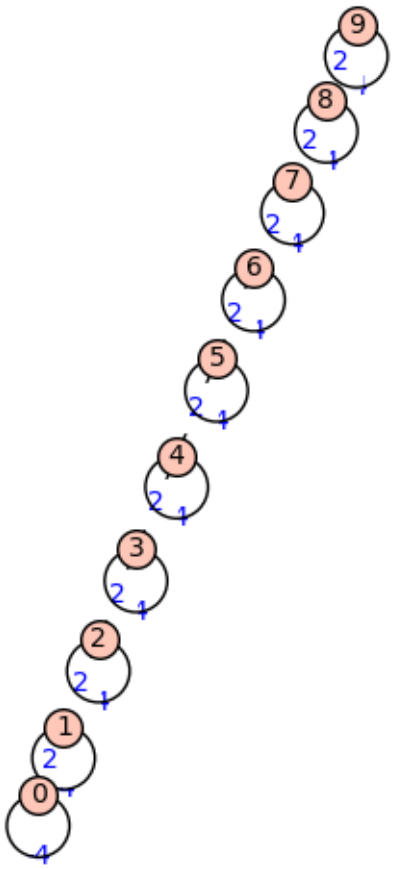

2
4
2
1
4
2
0
4


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

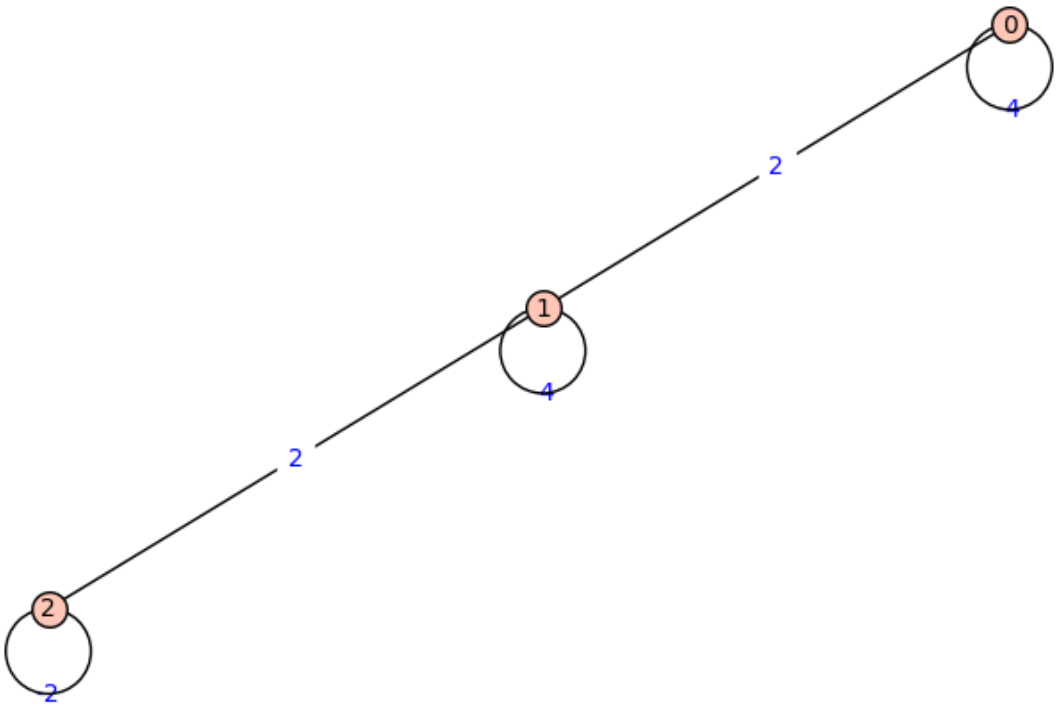

0
2
1
4
2
2
4
2
3
4
2
4
2
5
4
0
2
1
4
2
2
4
2
3
4
2
4
2
5
4
2
6
4

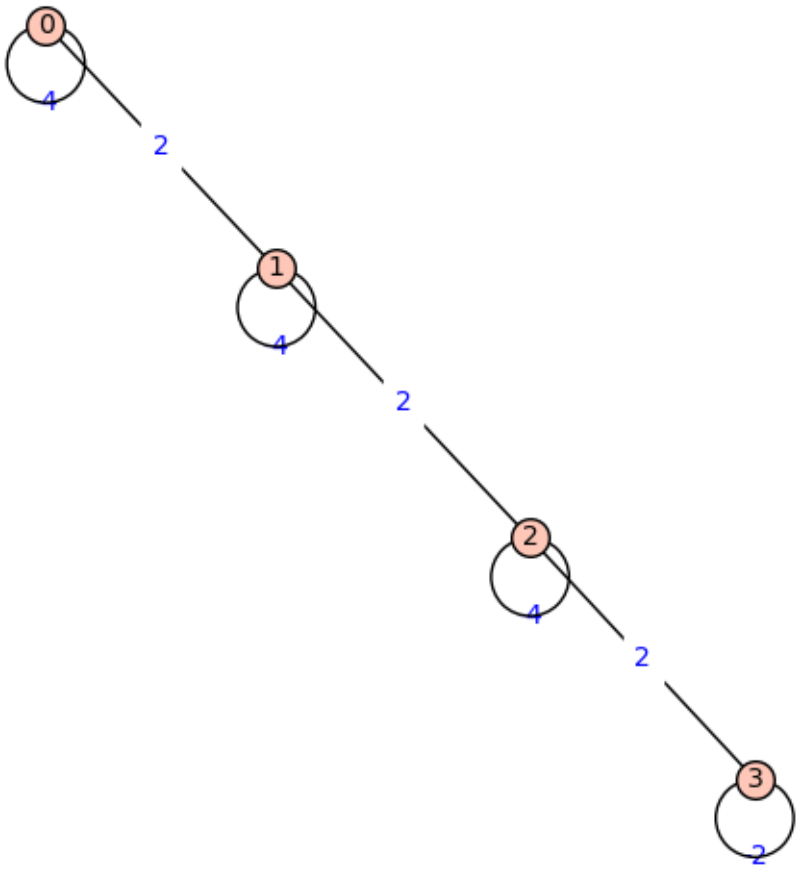

0
2
4
1
2
4
2
2
4
3
2
4
4
2
4
5
2
4
6
2
4
7
4

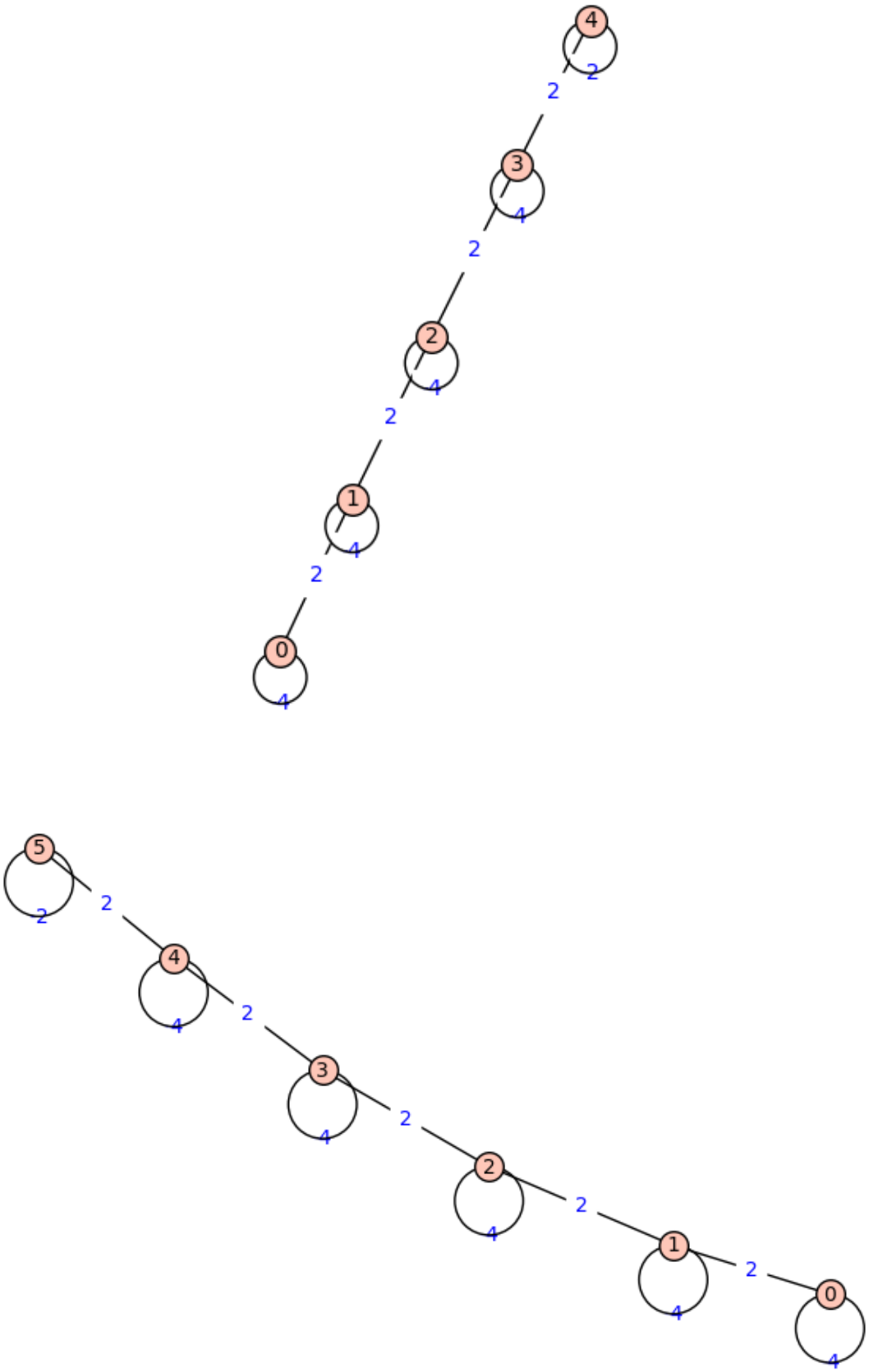

0
2
4
1
2
4
2
2
4
3
2
4
4
2
4
5
2
4
6
2
1
7
2
1
8
4

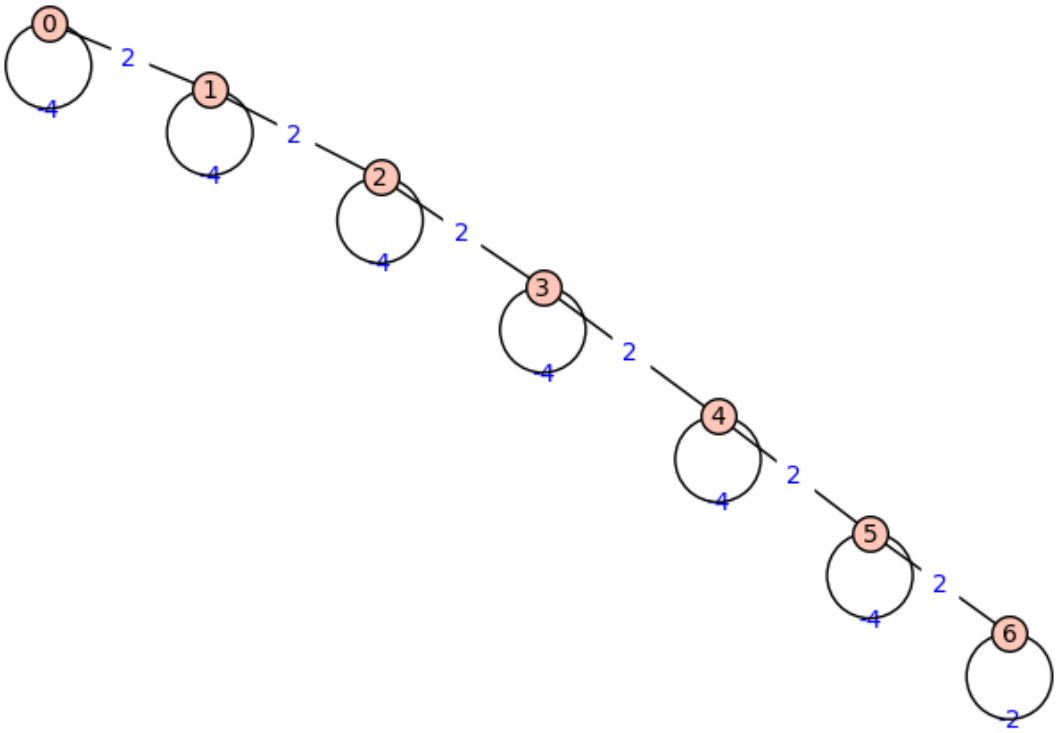


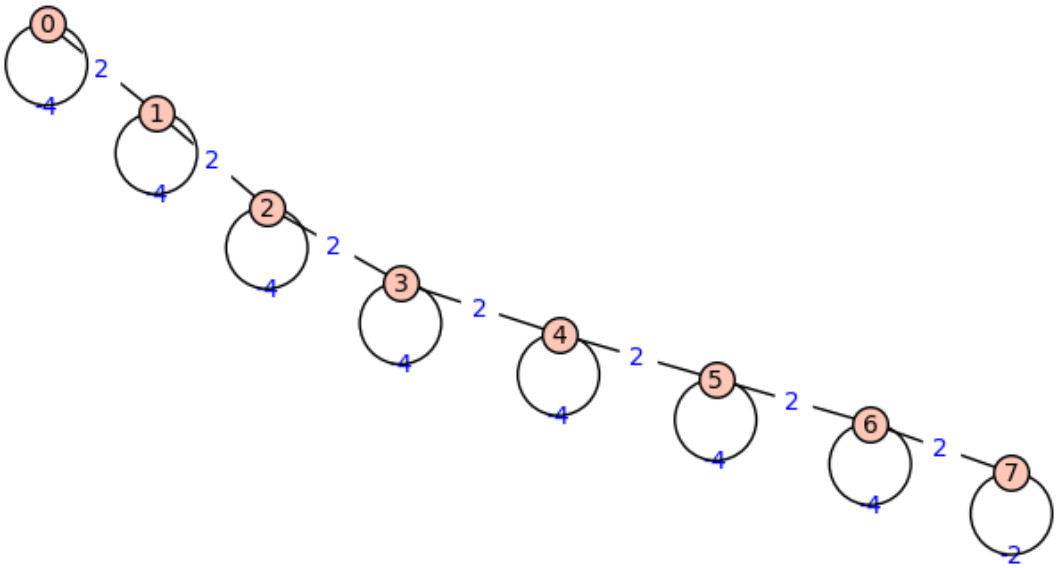
[15]:
```
# Type A_n(2)
for g in type_B_graphs:
    show( g.plot(edge_labels=True) )
```

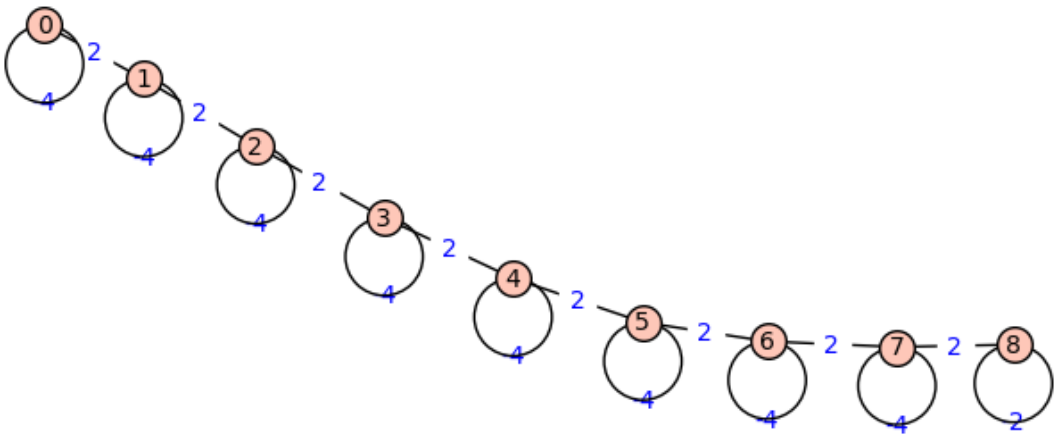

0
4
2
1
4
2
2
2

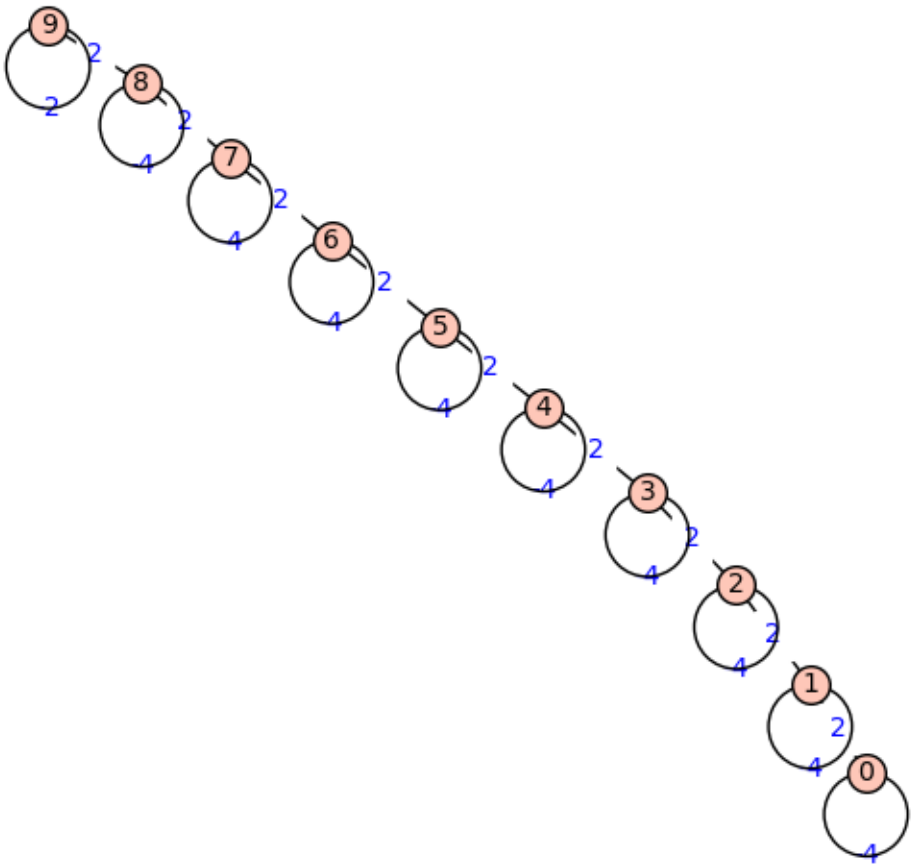

0
4
2
1
4
2
2
4
2
3
2


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

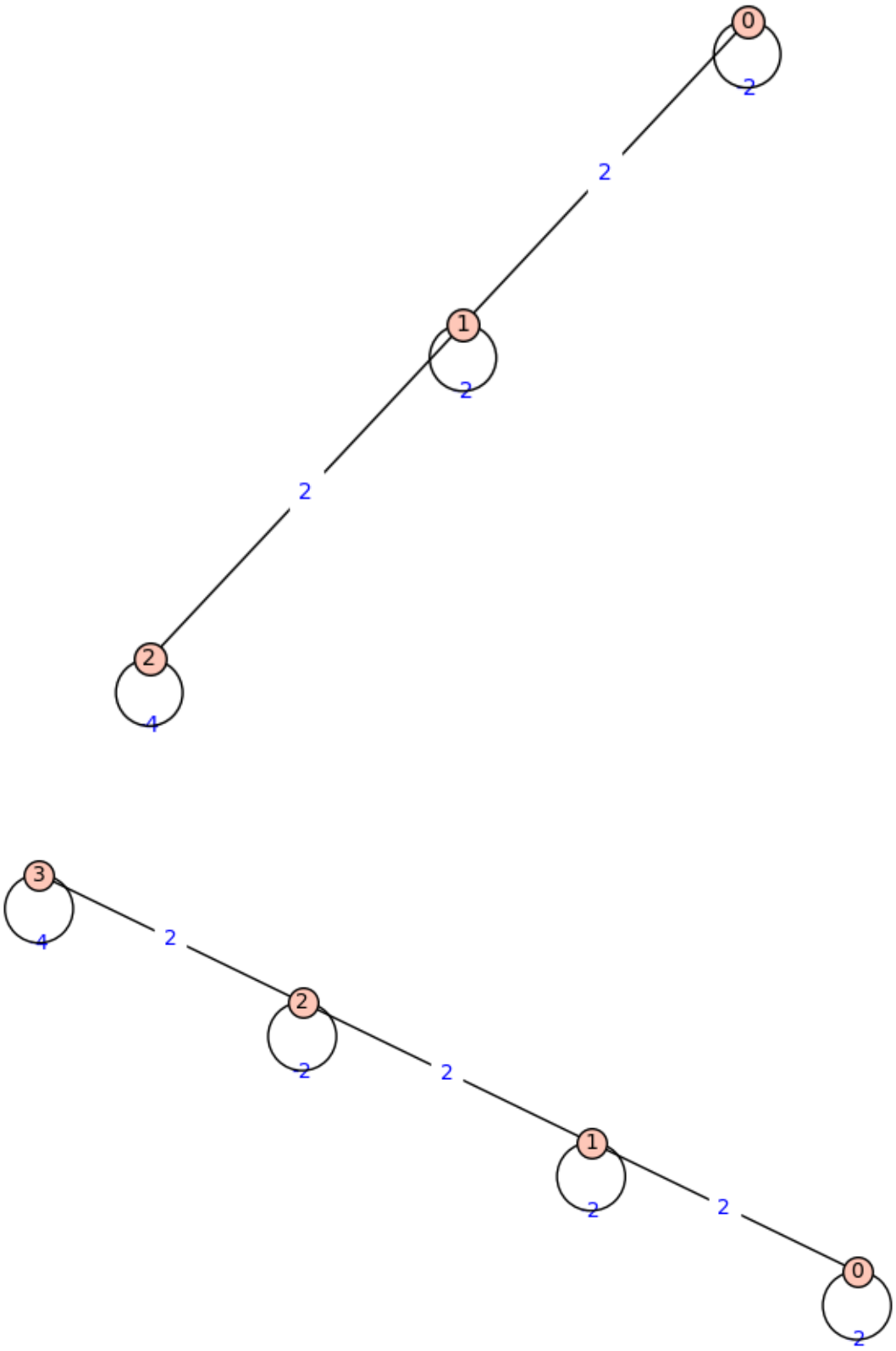

0
1
2
3
4
5
6
2
2
2
2
2
2
4
4
4
4
4
4
2

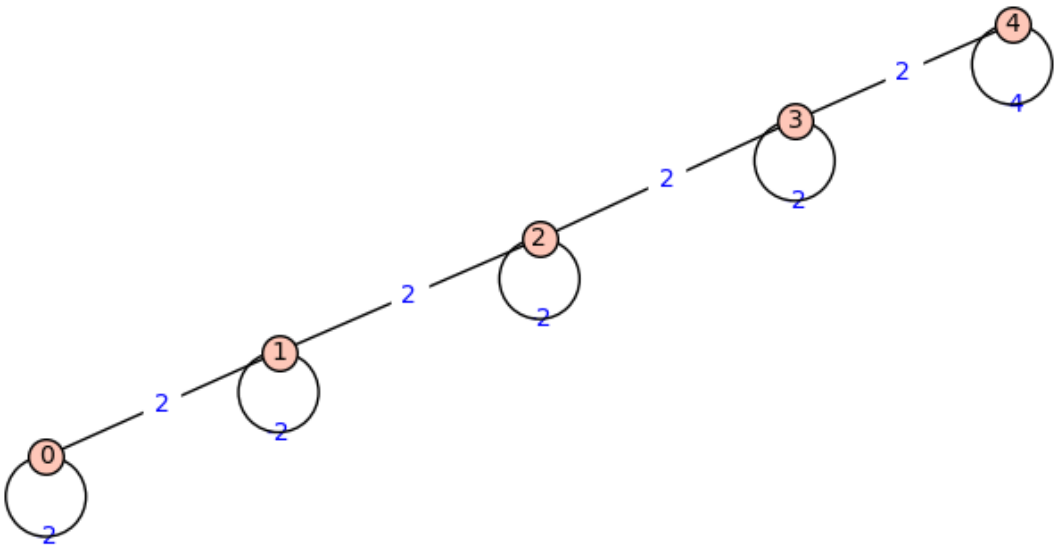

0
1
2
3
4
5
6
7
2
2
2
2
2
2
2
4
4
4
4
4
4
2

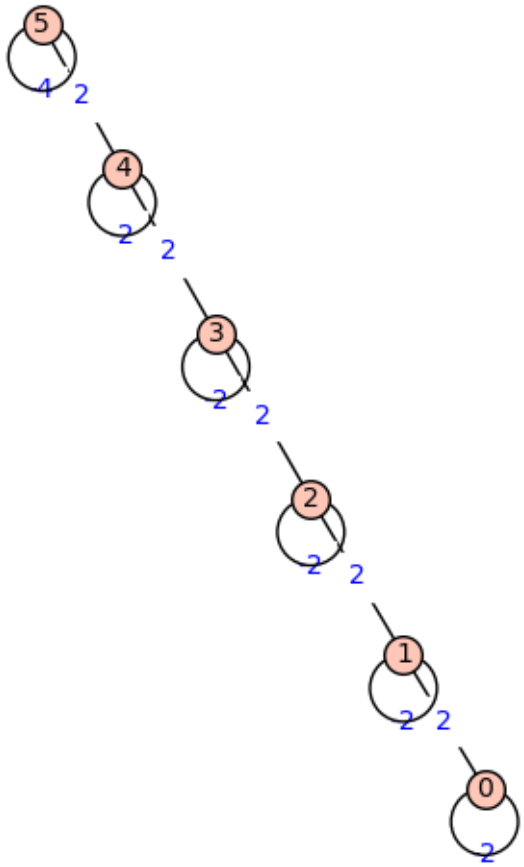


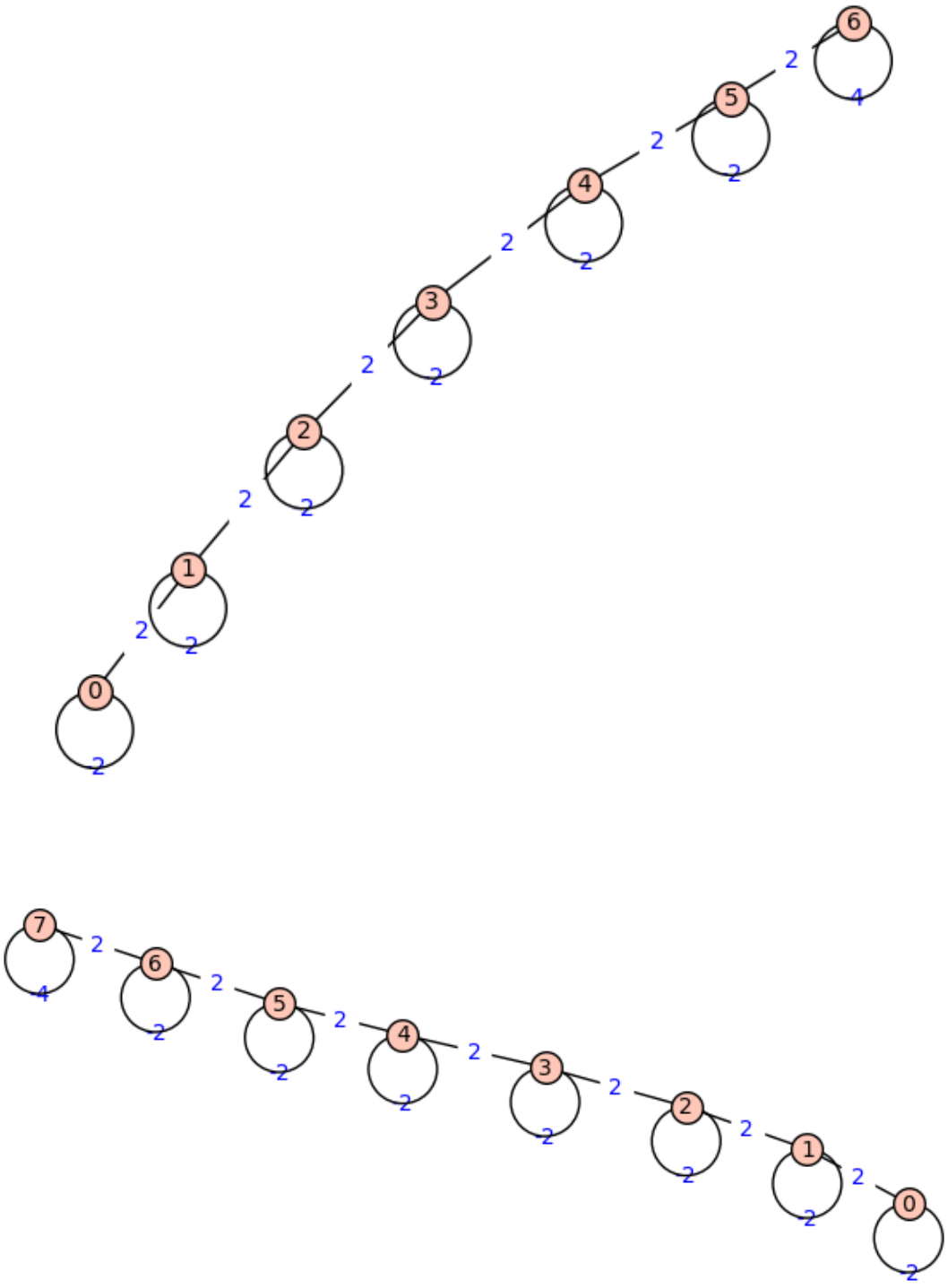


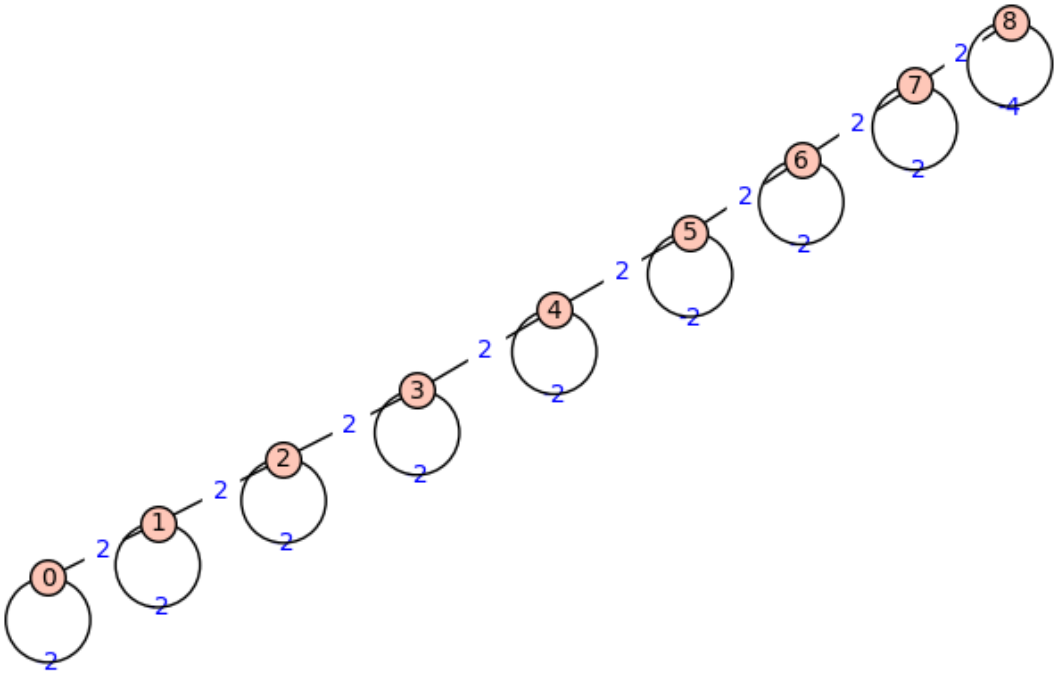
[16]:
```
# Type B_n(2)
for g in type_C_graphs:
    show( g.plot(edge_labels=True) )
```

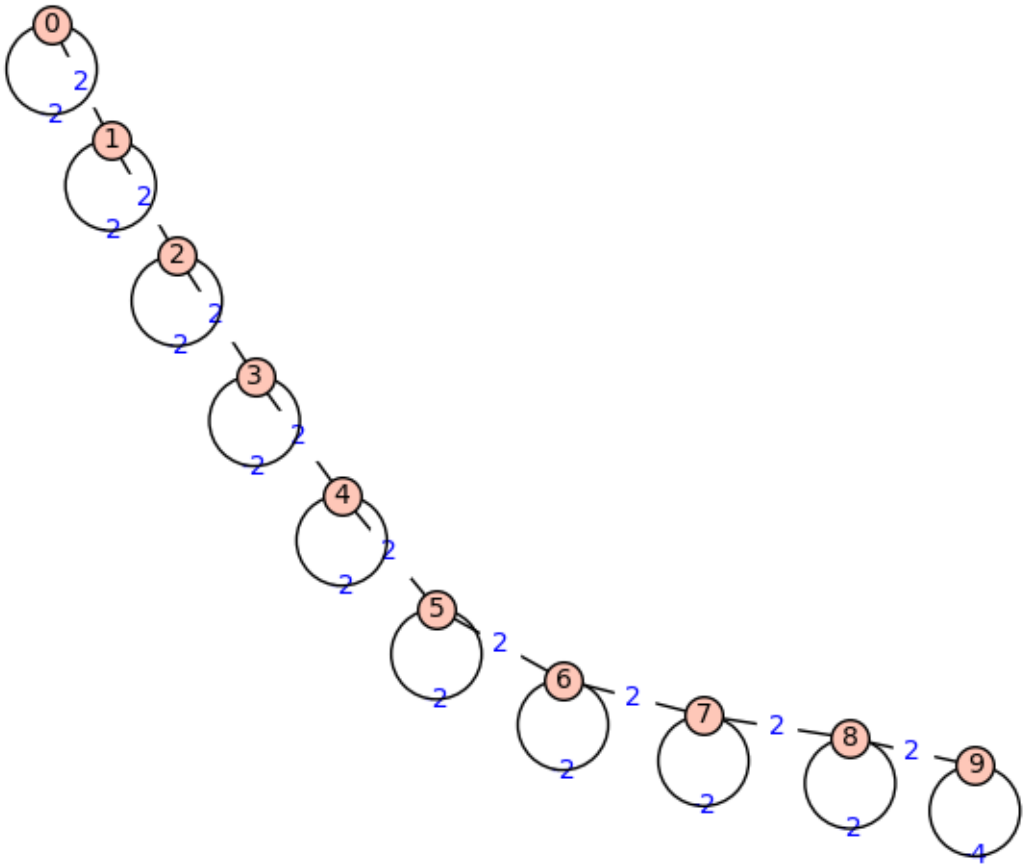

0
2
2
1
2
2
2
4
3
4
2
2
2
2
1
2
2
0
2


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

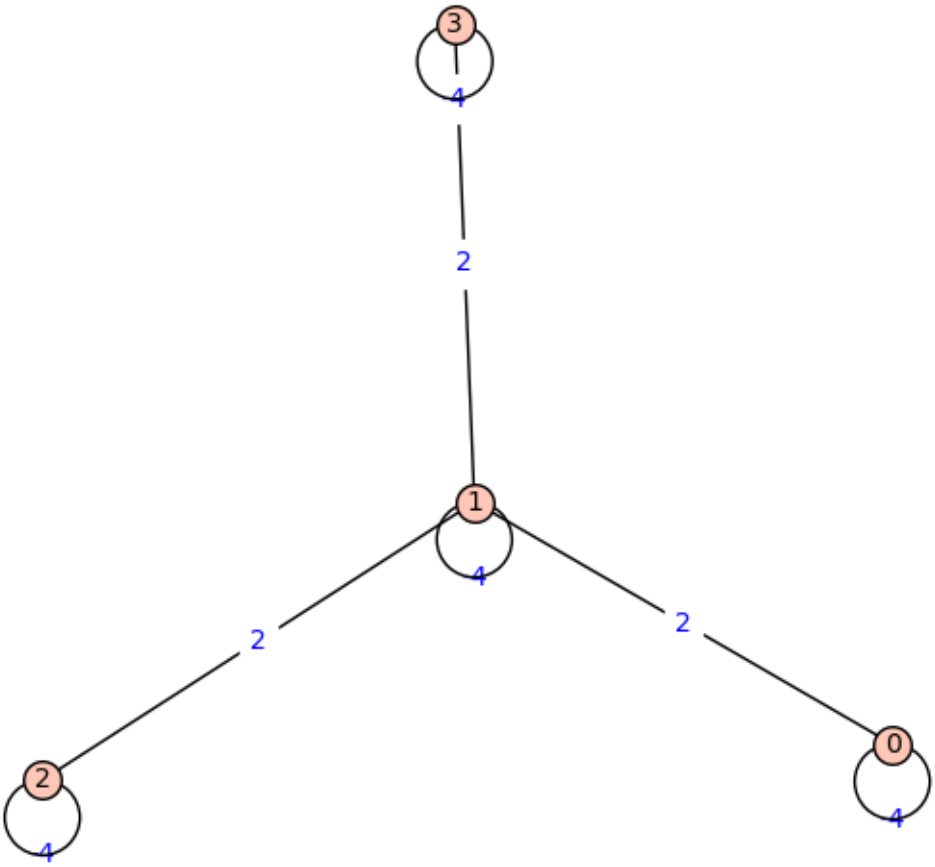

0
1
2
3
4
5
2
2
2
2
2
2
2
2
2
2
4

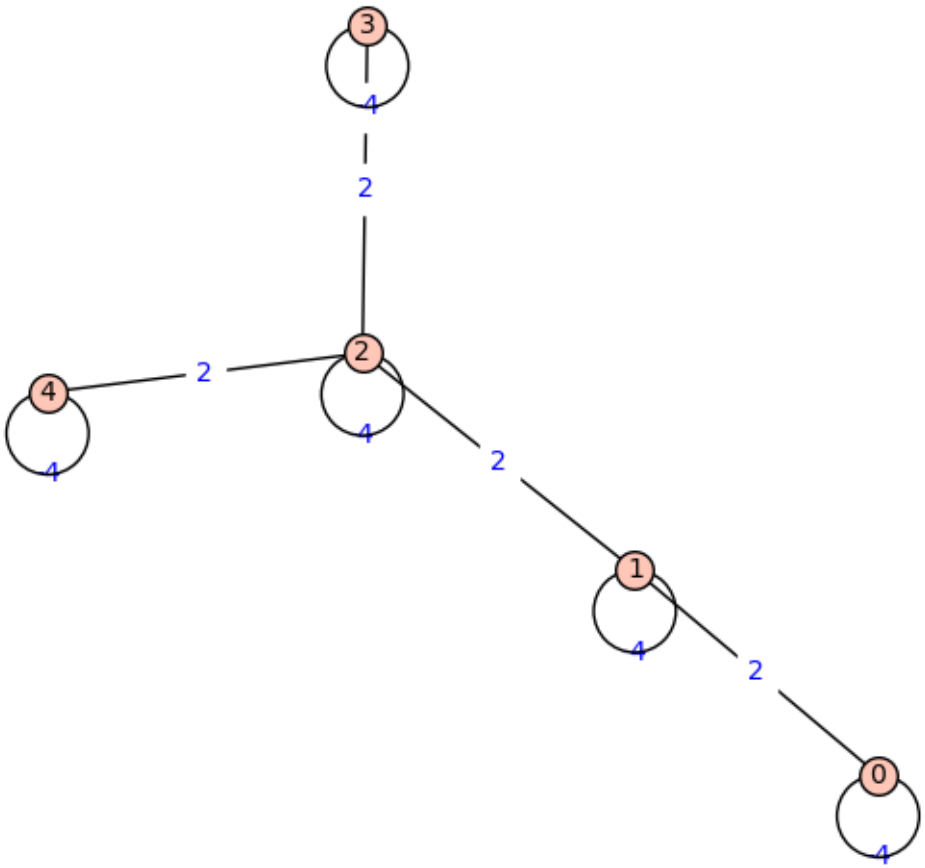

6
2
5
2
4
2
3
2
2
2
1
2
0
2
7
2
6
2
5
2
4
2
3
2
2
2
1
2
0

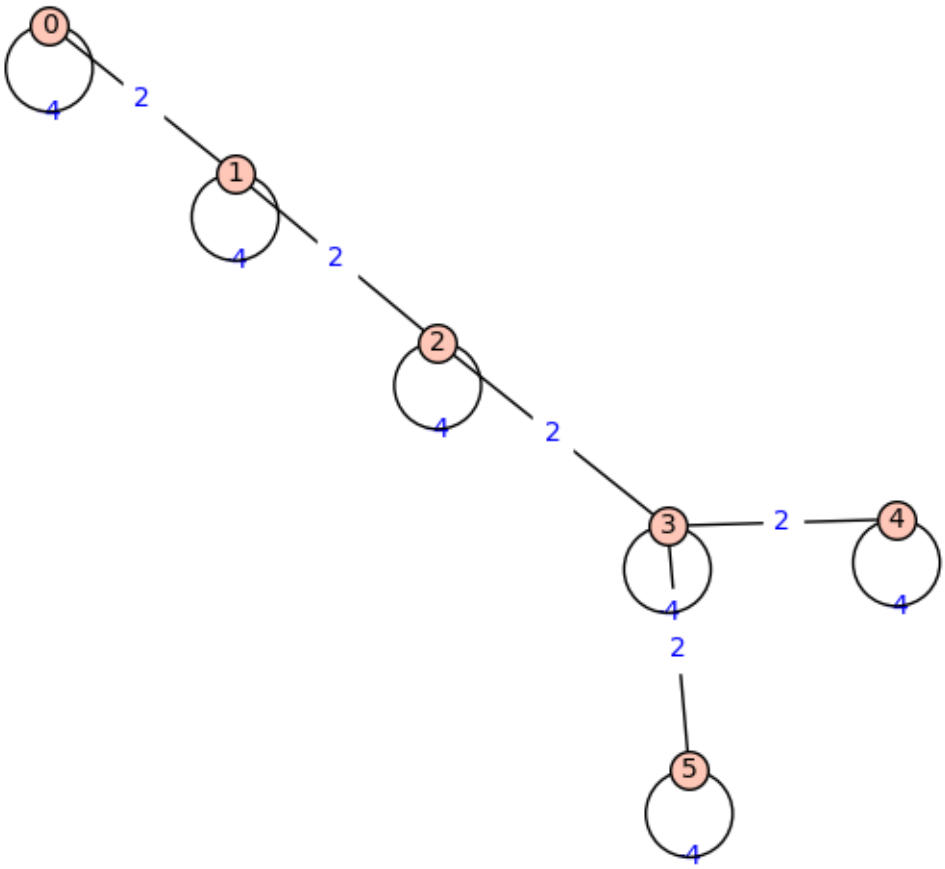

0
1
2
3
4
5
6
7
8
2

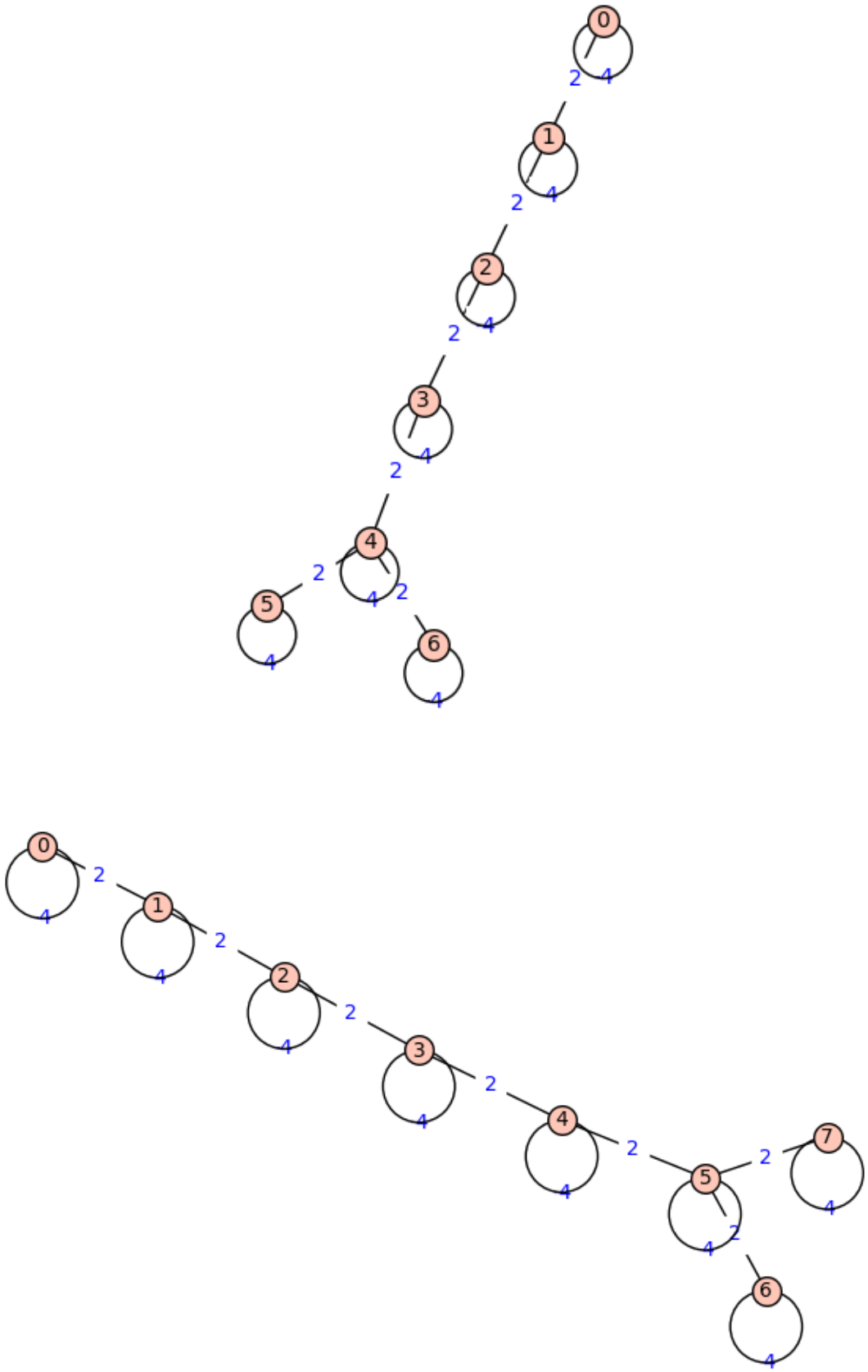

0
1
2
3
4
5
6
7
8
9
2

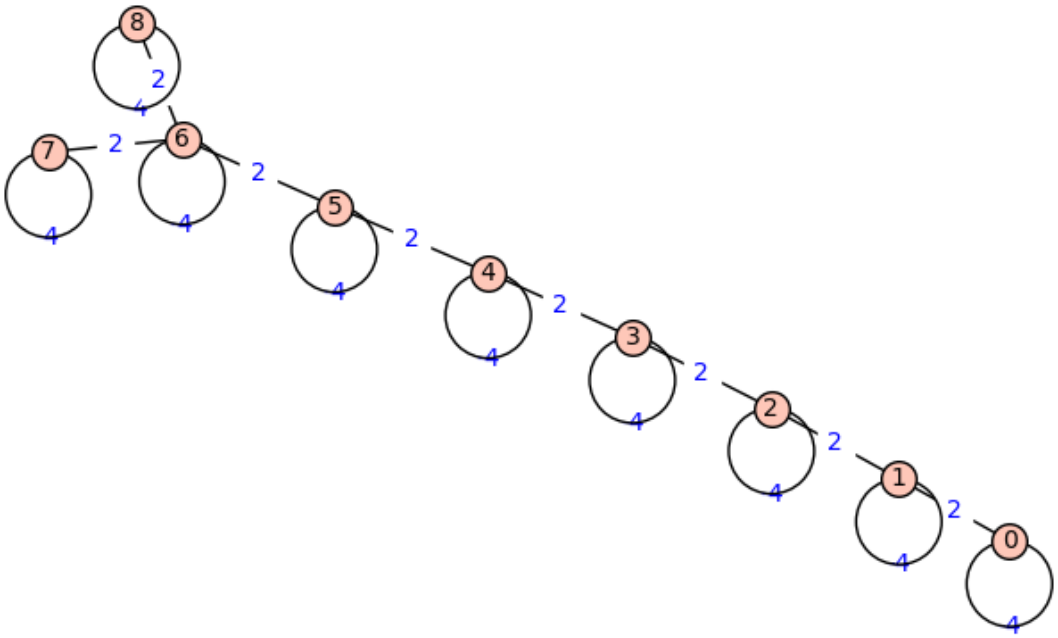
[17]:
```
for g in type_D_graphs:
    show( g.plot(edge_labels=True) )
```

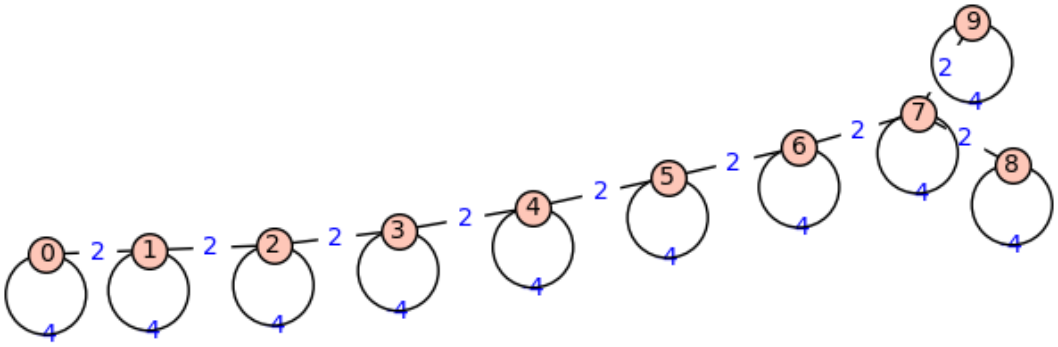



3
4
2
2
4
4
2
1
4
2
0
4

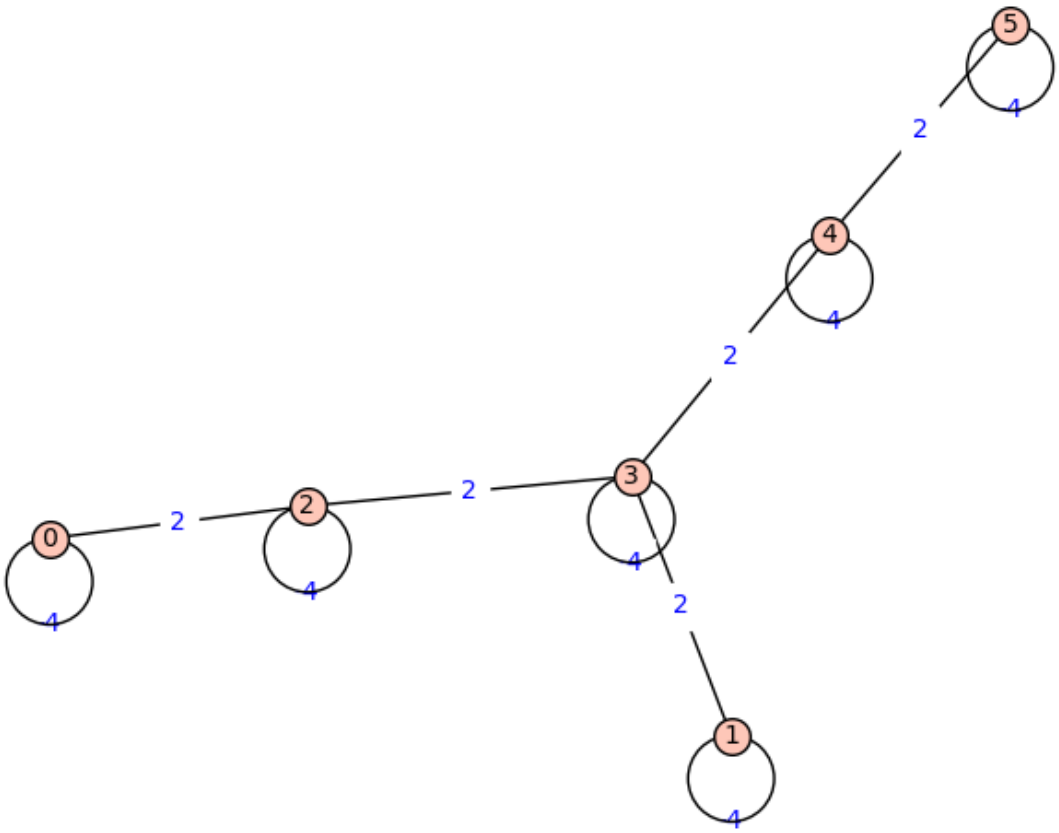

0
1
2
3
4
5
2
2
2
2
2
4
4
4
4
4
4

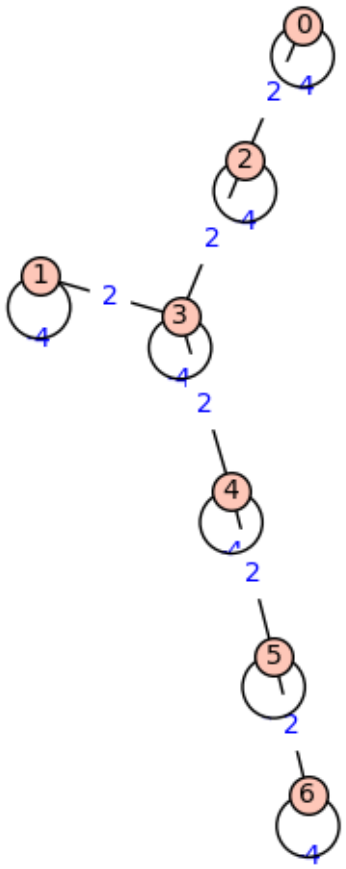

0
1
2
3
4
5
6
7
2
4

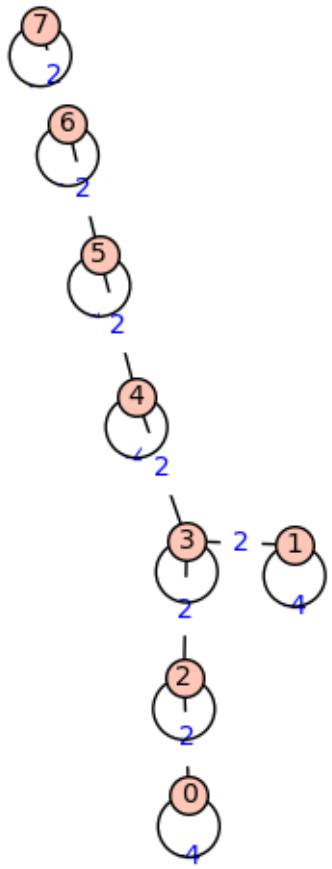




[18]:
```
for g in type_E_graphs:
    show( g.plot(edge_labels=True) )
```


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


0
2
4
2
2
4
2
1
2
3
4
2
4
2
5
2
6
4



[19]:
```
test_num = 20

for n in [i+2 for i in range(test_num)]:
    M = matrix_A_n(n)
    G = graph_A_n(n)
    assert graph_to_matrix( matrix_to_graph(M) ) == M
    assert matrix_to_graph( graph_to_matrix(G) ) == G

for n in [i+2 for i in range(test_num)]:
    M = matrix_A_n_2(n)
    G = graph_A_n_2(n)
    assert graph_to_matrix( matrix_to_graph(M) ) == M
    assert matrix_to_graph( graph_to_matrix(G) ) == G

for n in [i+2 for i in range(test_num)]:
    M = matrix_B_n_2(n)
    G = graph_B_n_2(n)
    assert graph_to_matrix( matrix_to_graph(M) ) == M
    assert matrix_to_graph( graph_to_matrix(G) ) == G
```

```
for n in [i+2 for i in range(test_num)]:
    M = matrix_C_n_2(n)
    G = graph_C_n_2(n)
    assert graph_to_matrix( matrix_to_graph(M) ) == M
    assert matrix_to_graph( graph_to_matrix(G) ) == G

for n in [i+2 for i in range(test_num)]:
    M = matrix_D_n(n)
    G = graph_D_n(n)
    assert graph_to_matrix( matrix_to_graph(M) ) == M
    assert matrix_to_graph( graph_to_matrix(G) ) == G

for n in [i+2 for i in range(test_num)]:
    M = matrix_D_n_2(n)
    G = graph_D_n_2(n)
    assert graph_to_matrix( matrix_to_graph(M) ) == M
    assert matrix_to_graph( graph_to_matrix(G) ) == G


assert graph_to_matrix( matrix_to_graph( matrix_E_6() ) ) == matrix_E_6()
assert graph_to_matrix( matrix_to_graph( matrix_E_7() ) ) == matrix_E_7()
assert graph_to_matrix( matrix_to_graph( matrix_E_8() ) ) == matrix_E_8()

assert graph_to_matrix( matrix_to_graph( matrix_E_6_2() ) ) == matrix_E_6_2()
assert graph_to_matrix( matrix_to_graph( matrix_E_7_2() ) ) == matrix_E_7_2()
assert graph_to_matrix( matrix_to_graph( matrix_E_8_2() ) ) == matrix_E_8_2()

assert matrix_to_graph( graph_to_matrix( graph_E_6() ) ) == graph_E_6()
assert matrix_to_graph( graph_to_matrix( graph_E_7() ) ) == graph_E_7()
assert matrix_to_graph( graph_to_matrix( graph_E_8() ) ) == graph_E_8()

assert matrix_to_graph( graph_to_matrix( graph_E_6_2() ) ) == graph_E_6_2()
assert matrix_to_graph( graph_to_matrix( graph_E_7_2() ) ) == graph_E_7_2()
assert matrix_to_graph( graph_to_matrix( graph_E_8_2() ) ) == graph_E_8_2()

show("Tests passed")
```

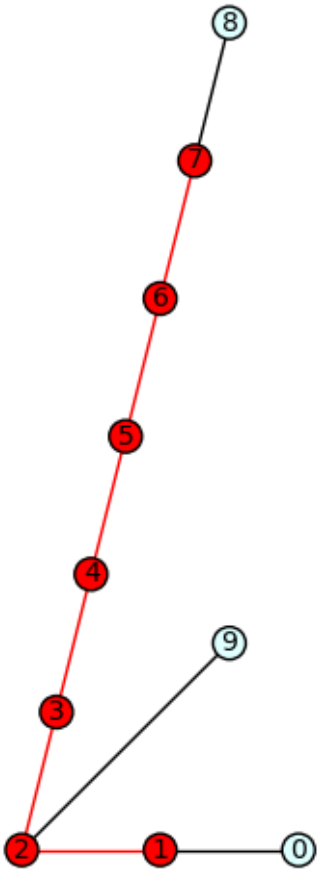
```
Tests passed
```

[20]:
```
def is_elliptic_subgraph(H):
    H_roots = [S2[index] for index in H.vertices()]
    M_H = root_intersection_matrix(H_roots, labels = H.vertices(), bil_form=dot)
    return is_elliptic_matrix(M_H)

def is_parabolic_subgraph(H):
    pass
```

```
def is_maximal_elliptic_subgraph(H):
    pass

def is_maximal_parabolic_subgraph(H):
    pass

def plot_subgraph(H):
    red_edges = [ e for e in G_Sterk_2_loops.edges() if e in H.edges() ]
    red_vertices = [ e for e in G_Sterk_2_loops.vertices() if e in H.vertices()
 ↪]
    display(G_Sterk_2_loops.plot(
        vertex_size=150,
        edge_colors={'red': red_edges},
        vertex_color='lightcyan',
        vertex_colors={'red': red_vertices},
        pos={
            0: [0, 0],
            1: [-4, 0],
            2: [-8, 0],
            3: [-7, 4],
            4: [-6, 8],
            5: [-5, 12],
            6: [-4, 16],
            7: [-3, 20],
            8: [-2, 24],
            9: [-2, 6]
        }
    ))
```

[21]:
```
G_Sterk_2_subgraph_vertices = subsets( set(list( G_Sterk_2_loops.vertices()) ) )

small_subgraph_sample = [G_Sterk_2_loops.subgraph(l) for l in
 ↪G_Sterk_2_subgraph_vertices]

subgraphs = list(reversed(sorted(
    [H for H in small_subgraph_sample if H.is_connected() and
 ↪is_elliptic_subgraph(H) ],
    key=len
)))

show(len(subgraphs))
plot_subgraph( subgraphs[10] )
```

65



[22]:
```
from collections import Counter

ranks = [len(H.vertices()) for H in subgraphs]
# show(ranks)
d = Counter( ranks )
n = len( G_Sterk_2_loops.vertices() )
for i in reversed(range(n+1)):
    show(f"Number of rank {i} elliptic subdiagrams: {d[i]}")
show("---------------------------------")
show(f"Total number of elliptic subdiagrams: {len(ranks) }")
```

```
Number of rank 10 elliptic subdiagrams: 0
```

```
Number of rank 9 elliptic subdiagrams: 1
```

```
Number of rank 8 elliptic subdiagrams: 5
```

```
Number of rank 7 elliptic subdiagrams: 6
```

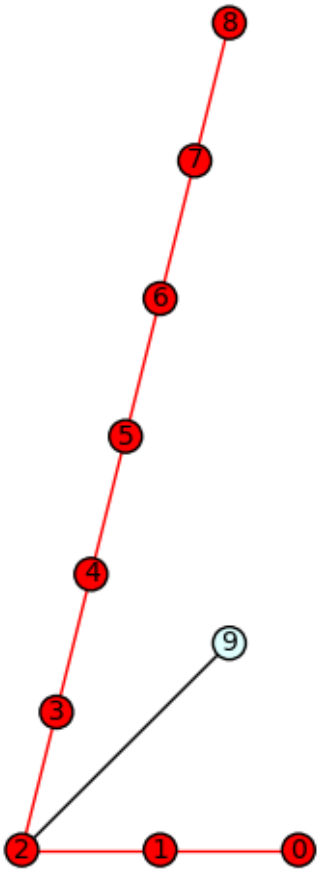
```
Number of rank 6 elliptic subdiagrams: 7
```

```
Number of rank 5 elliptic subdiagrams: 8
```

```
Number of rank 4 elliptic subdiagrams: 9

Number of rank 3 elliptic subdiagrams: 9

Number of rank 2 elliptic subdiagrams: 9

Number of rank 1 elliptic subdiagrams: 10

Number of rank 0 elliptic subdiagrams: 1

----------------------------------

Total number of elliptic subdiagrams: 65
```

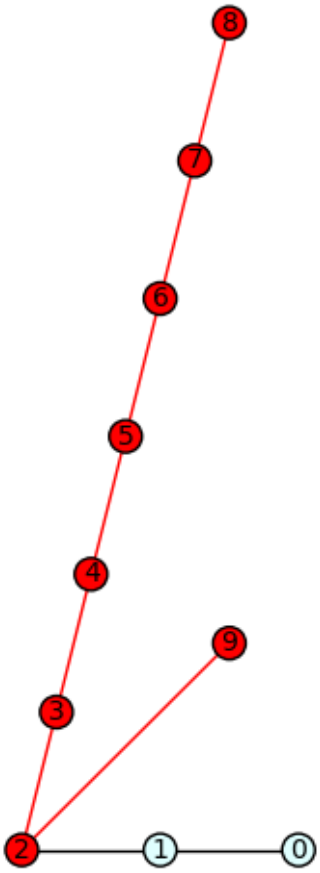
[23]:
```python
for i in reversed(range(n+1)):
# for i in [8, 9]:
    rank_i_subgraphs = [H for H in subgraphs if len(H.vertices() ) == i]
    show(f"Rank {i}: ")
    ade_types = get_all_rank_n_types(i)
    for H in rank_i_subgraphs:
        plot_subgraph(H)
        M_H = sterk_2_subgraph_to_matrix(H)
        this_type = [ x[0] for x in ade_types if x[1].is_similar(M_H) ]
        show(this_type)
    show("--------------------------------------")
```

```
Rank 10:

--------------------------------------

Rank 9:
```

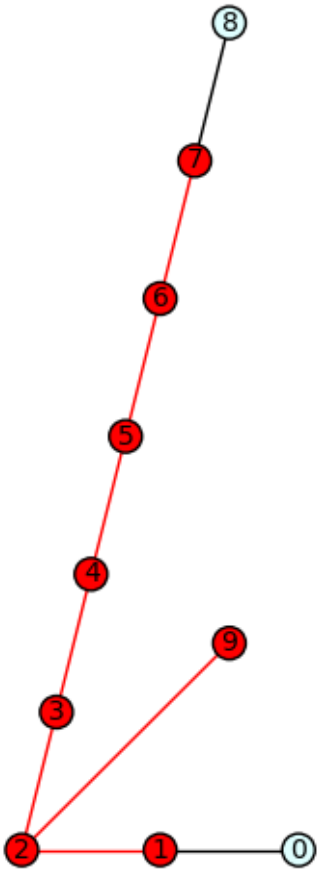


[B_9(2)]

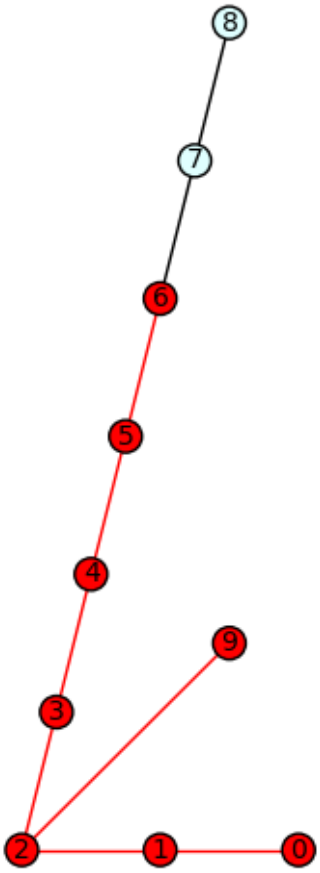
Rank 8:



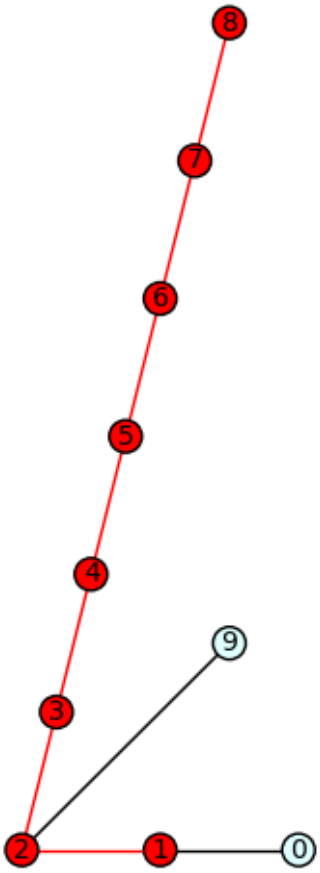
[B_8(2)]



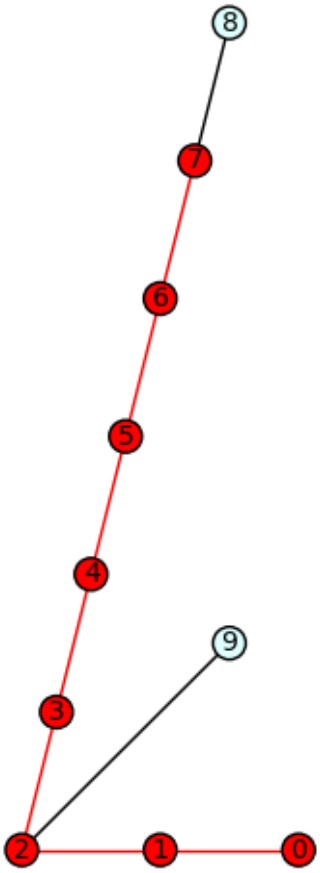
[D_8(2)]



[E_8(2)]

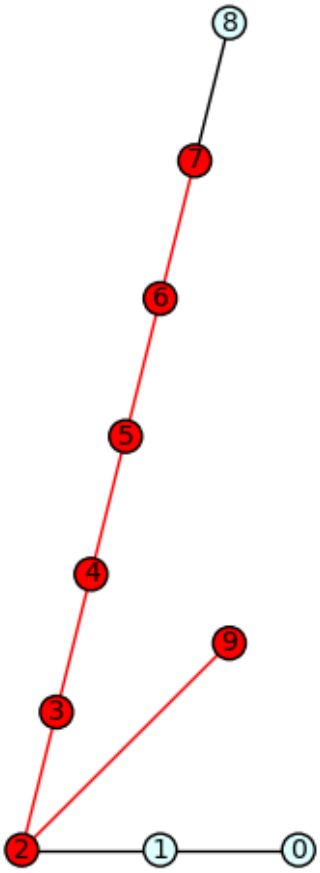


[B_8(2)]

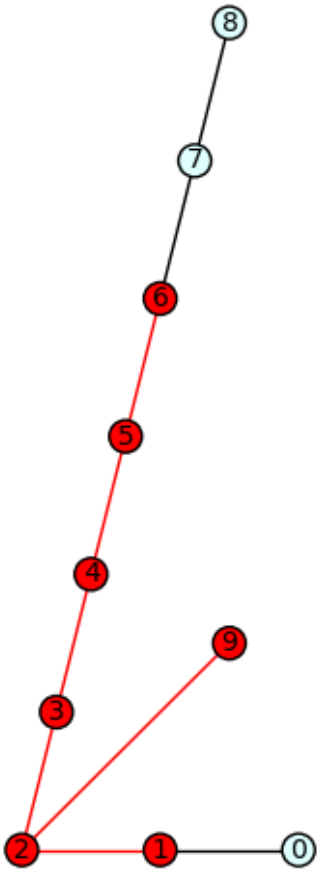


[A_8(2)]

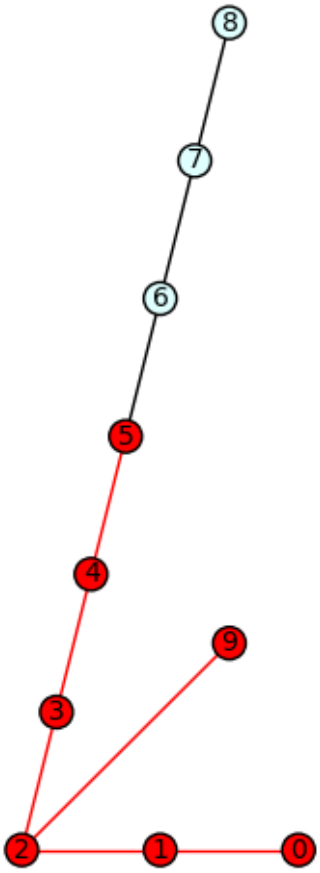
Rank 7:



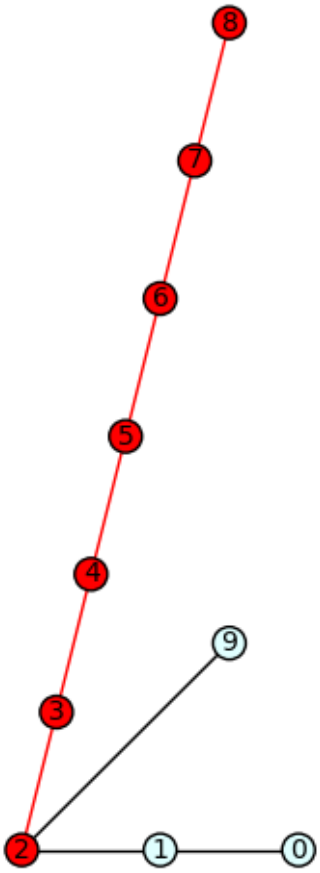
[A_7(2)]



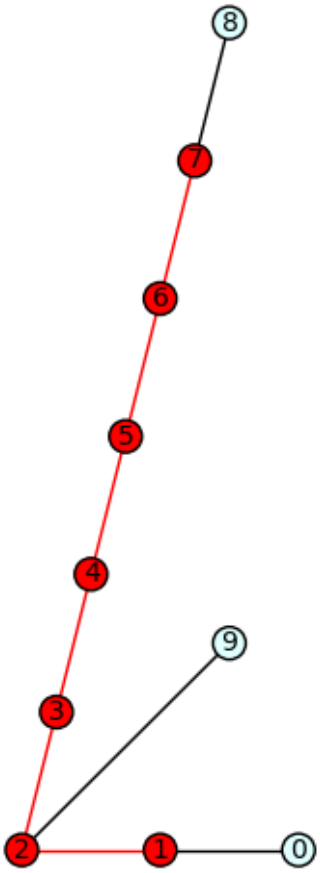
[D_7(2)]



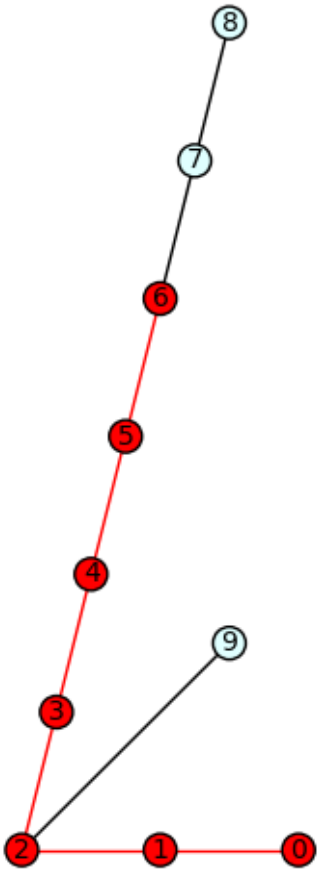
[E_7(2)]



[B_7(2)]

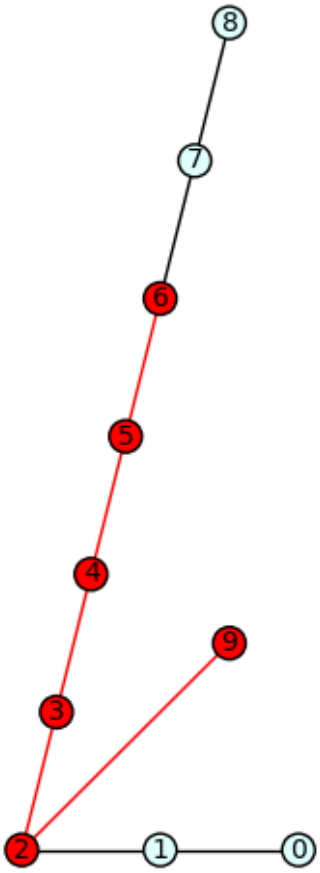


[A_7(2)]

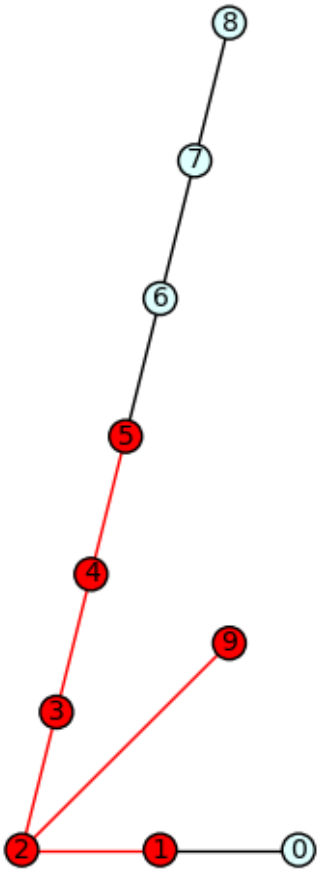


[A_7(2)]

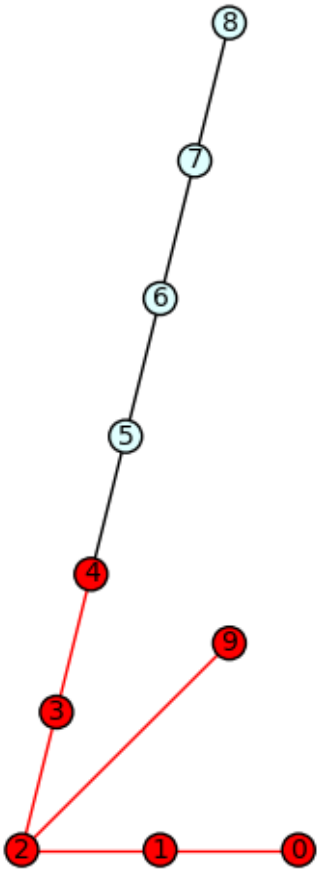
Rank 6:



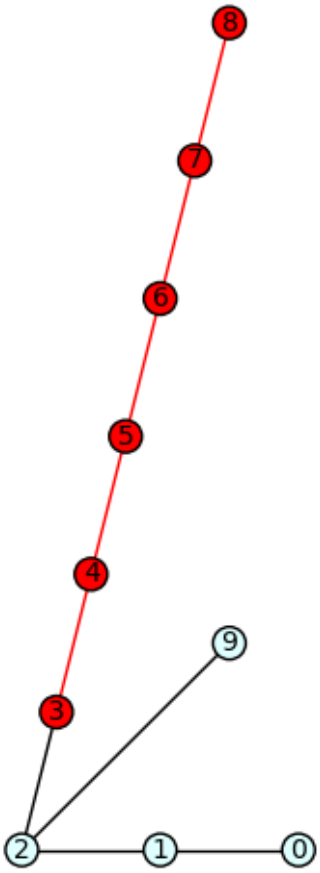
[A_6(2)]



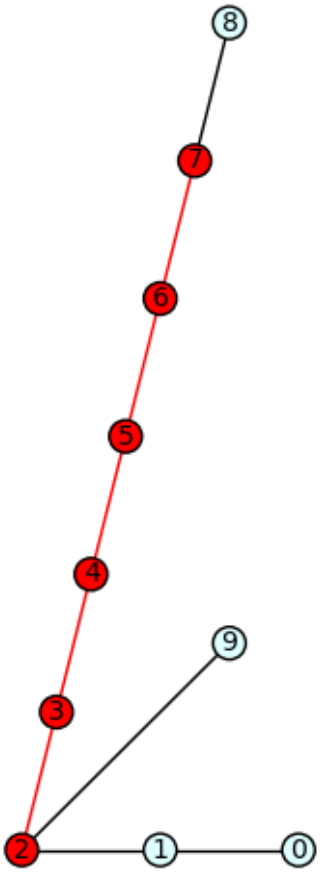
[D_6(2)]



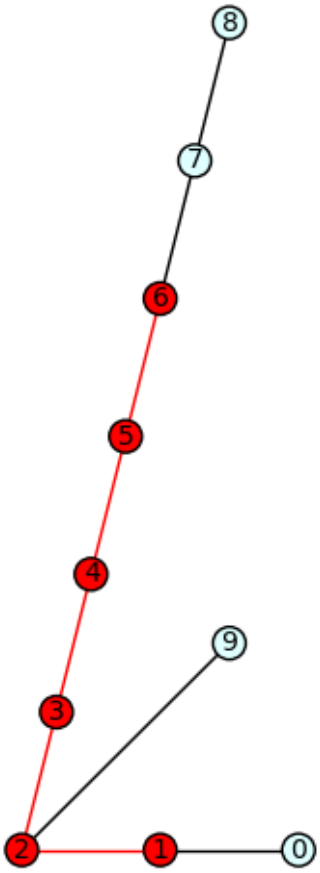
[E_6(2)]



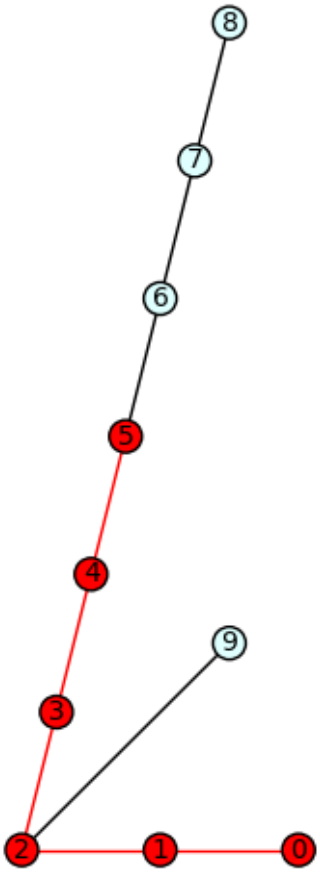
[B_6(2)]



[A_6(2)]

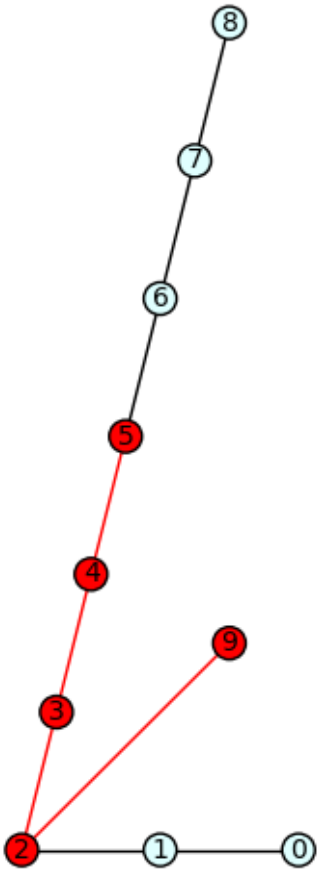


[A_6(2)]

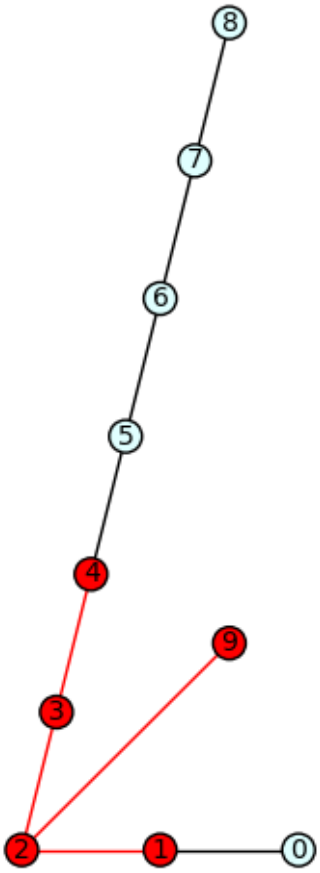


[A_6(2)]

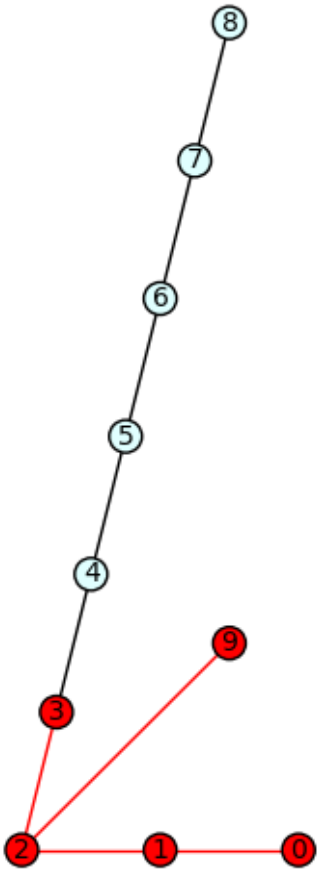
Rank 5:



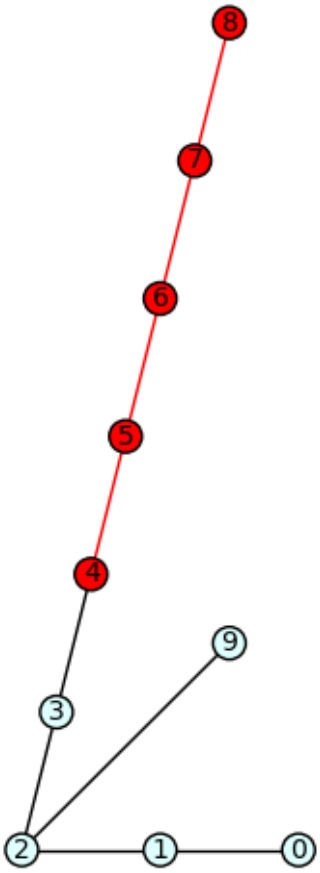
[A_5(2)]



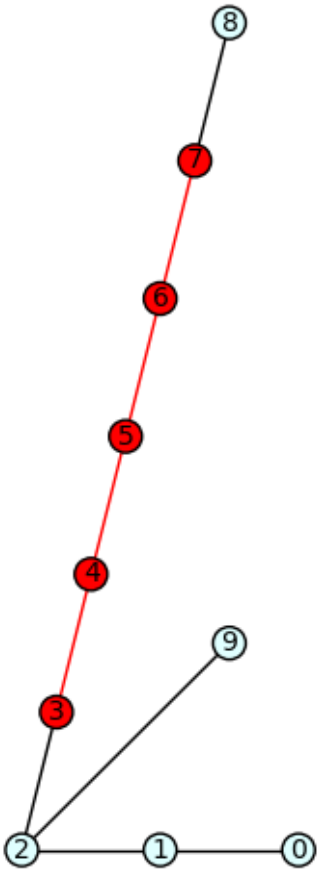
[D_5(2)]



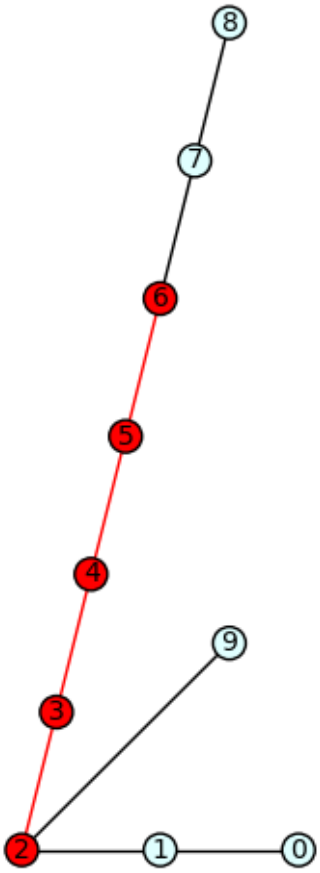
[D_5(2)]



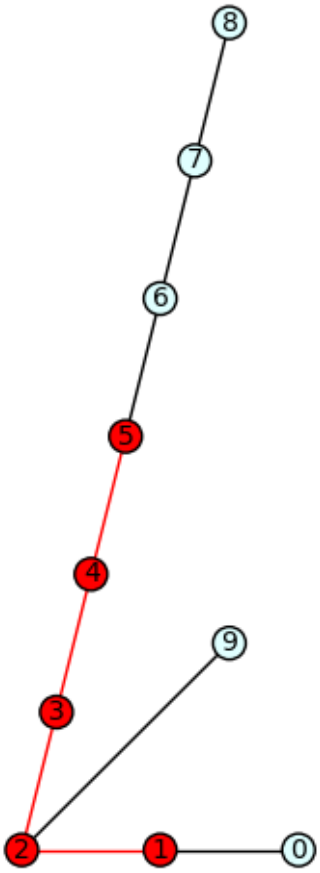
[B_5(2)]



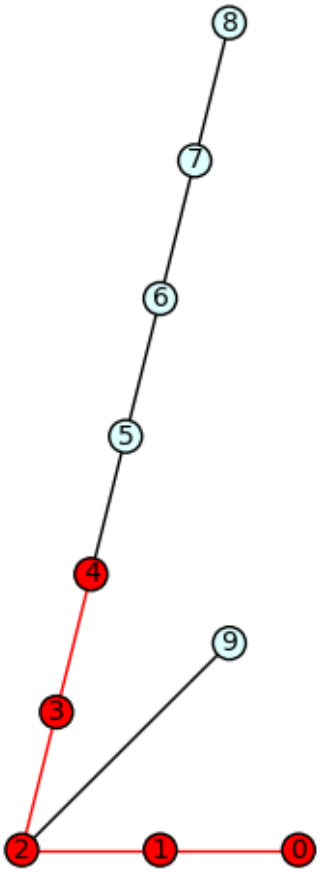
[A_5(2)]



[A_5(2)]

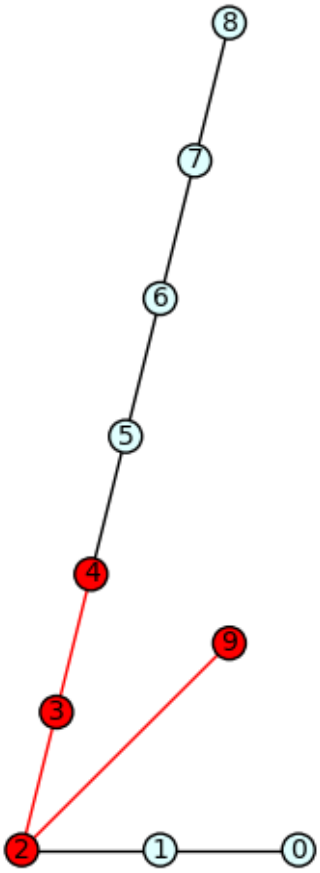


[A_5(2)]

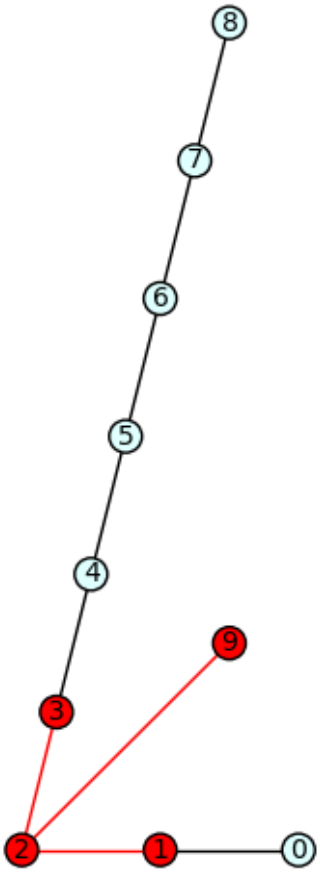


[A_5(2)]

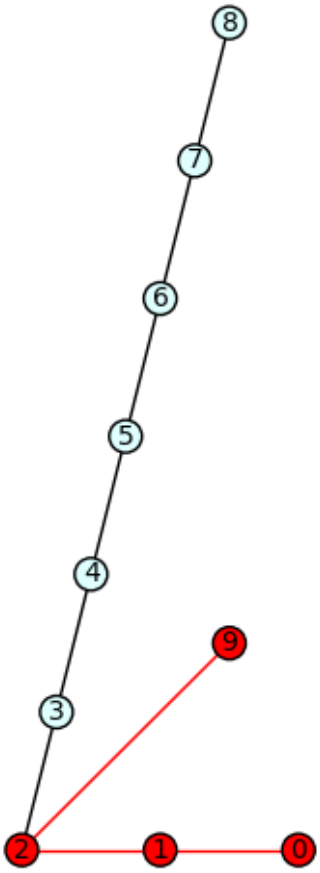
Rank 4:



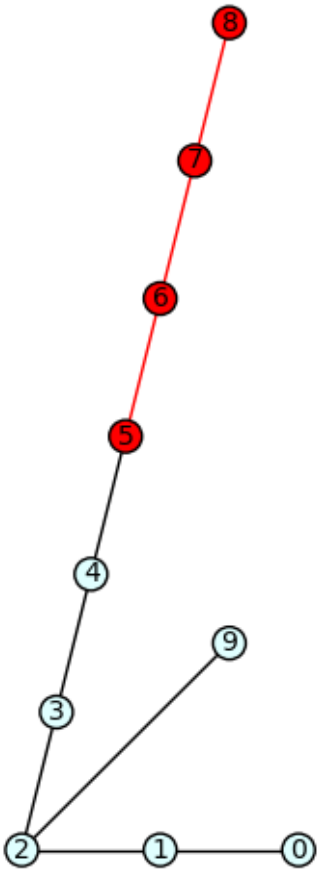
[A_4(2)]



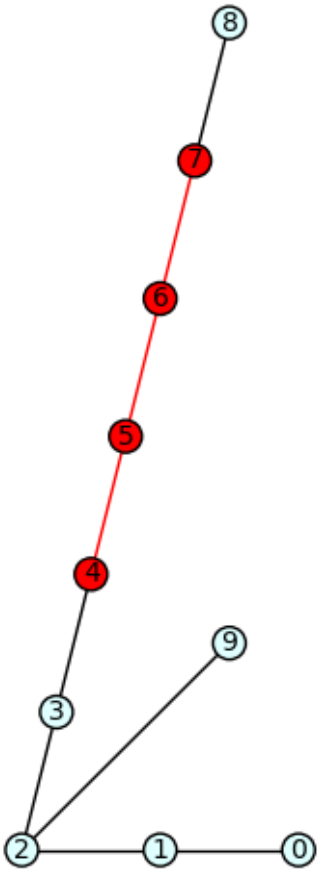
[D_4(2)]



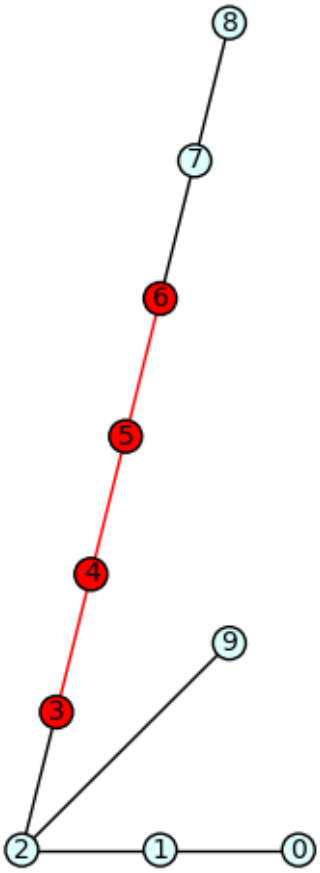
[A_4(2)]



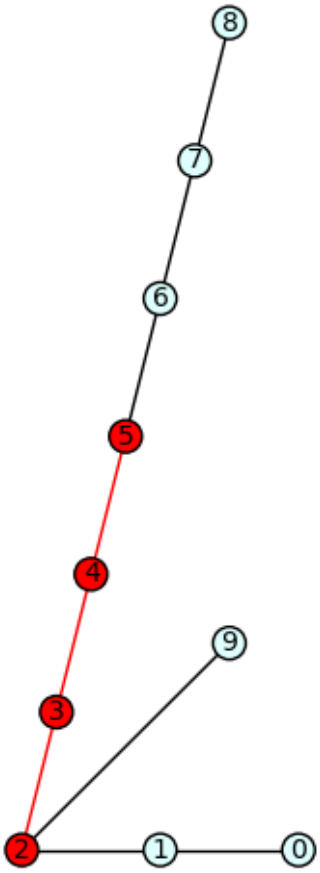
[B_4(2)]



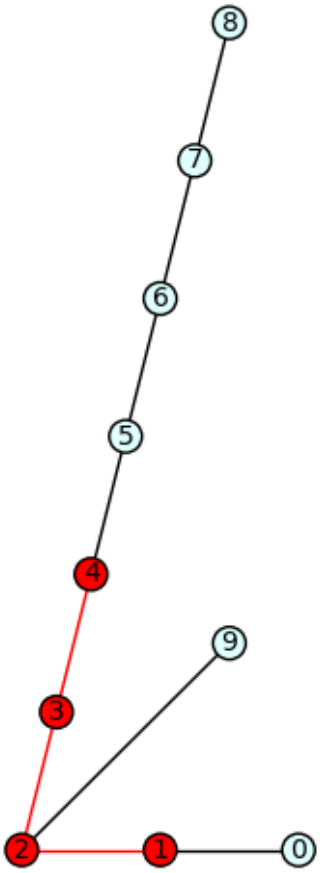
[A_4(2)]



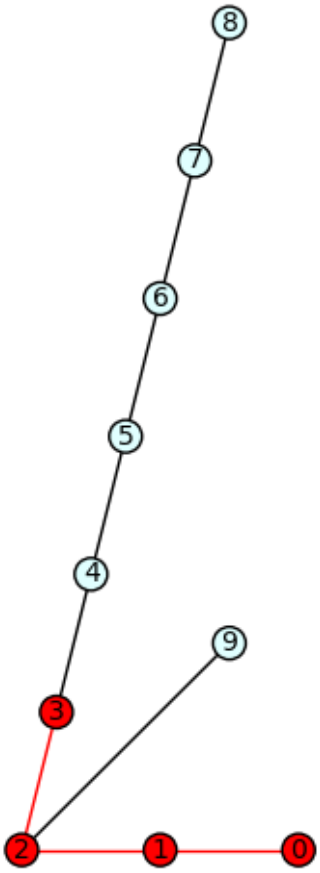
[A_4(2)]



[A_4(2)]

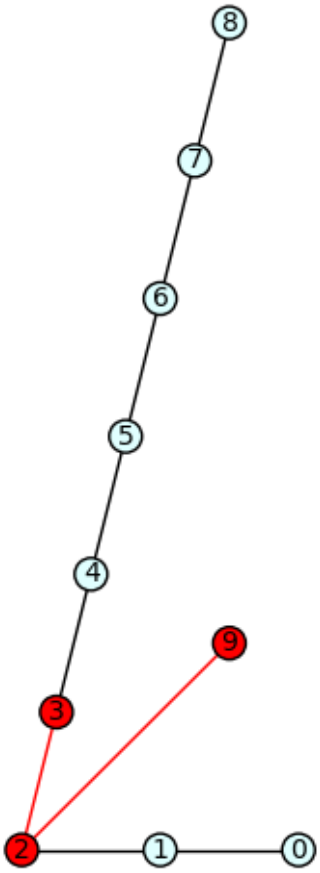


[A_4(2)]

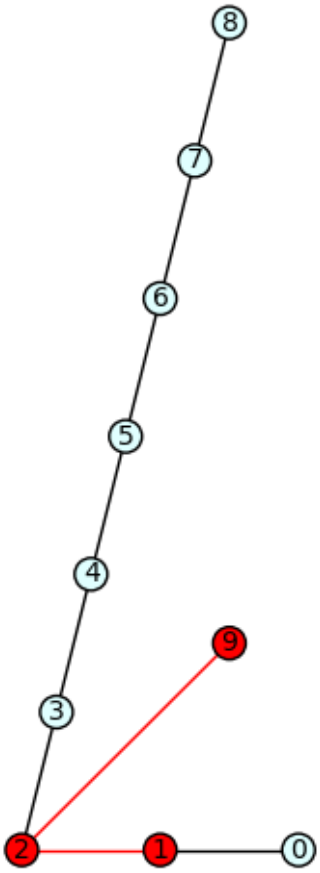


[A_4(2)]

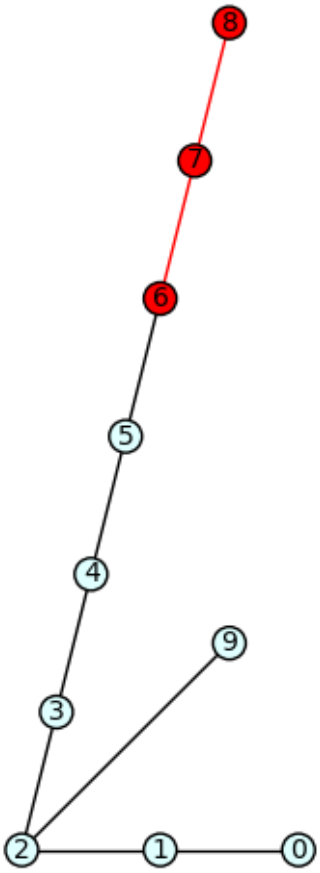
Rank 3:



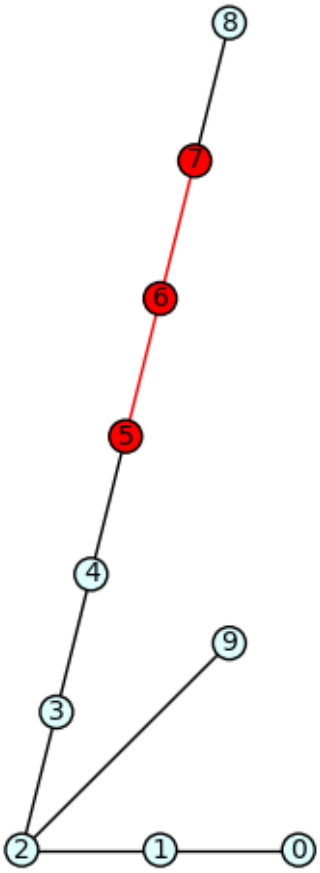
[A_3(2),D_3(2)]



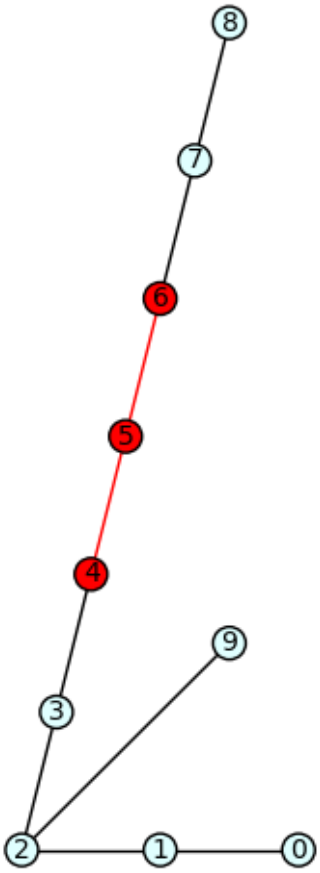
[A_3(2),D_3(2)]



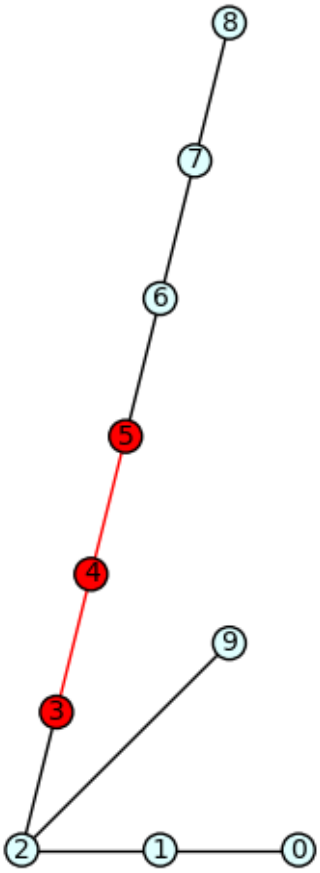
[B_3(2)]



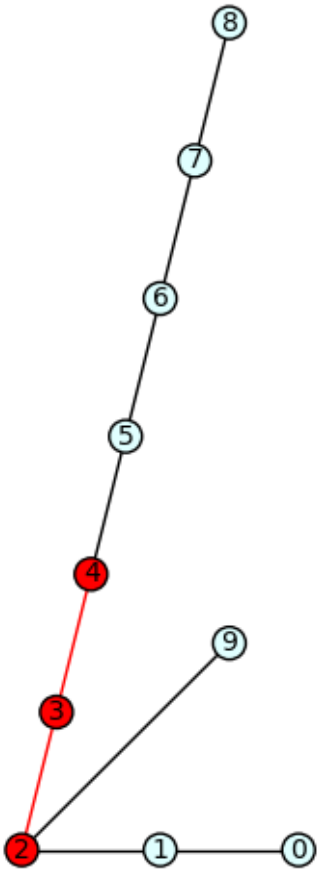
[A_3(2),D_3(2)]



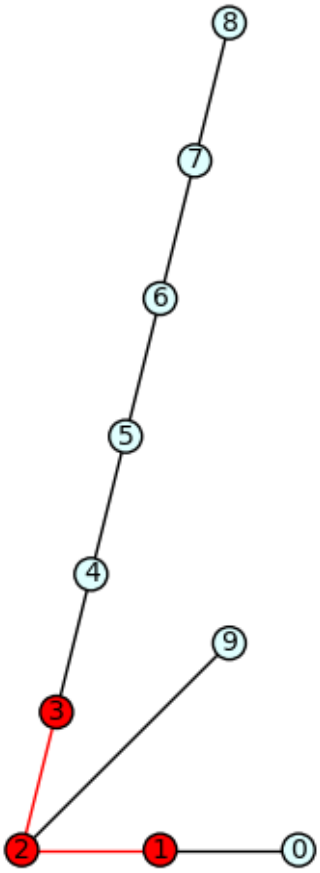
[A_3(2),D_3(2)]



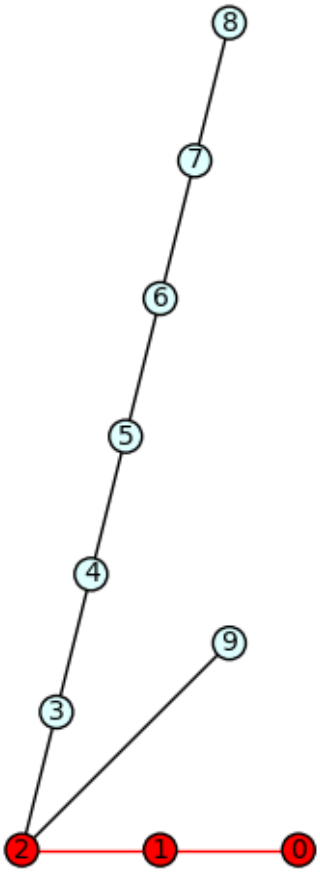
[A_3(2),D_3(2)]



[A_3(2),D_3(2)]

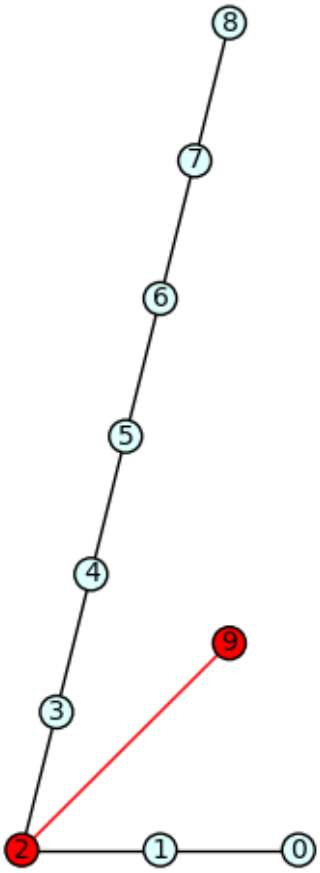


[A_3(2),D_3(2)]

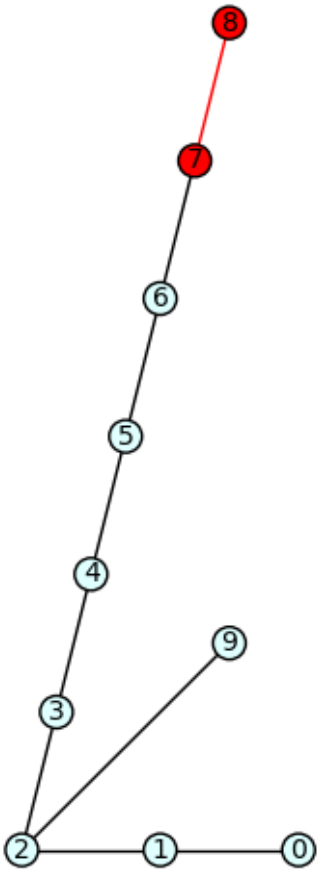


[A_3(2),D_3(2)]

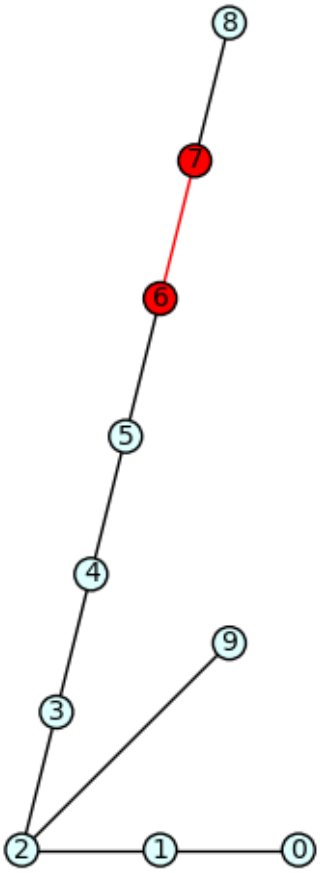
Rank 2:



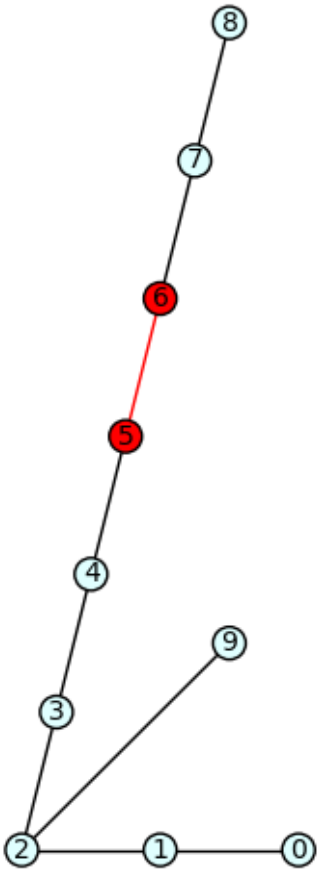
[A_2(2)]



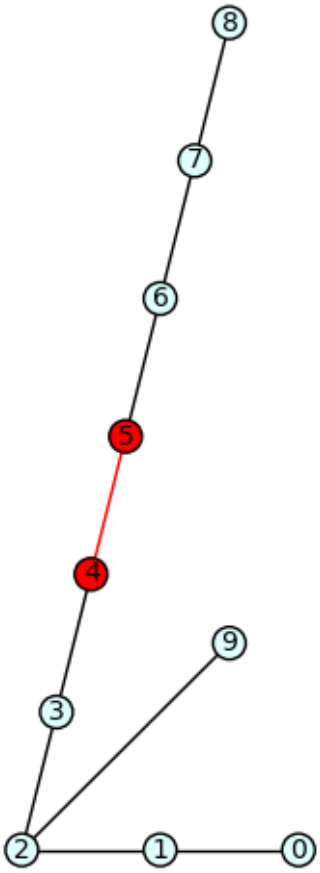
[G_2]



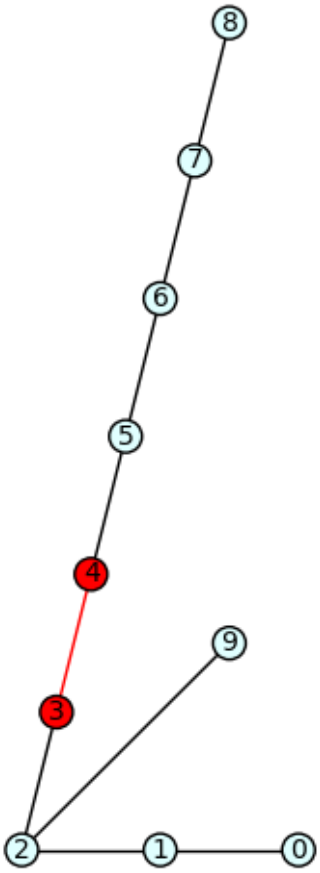
[A_2(2)]



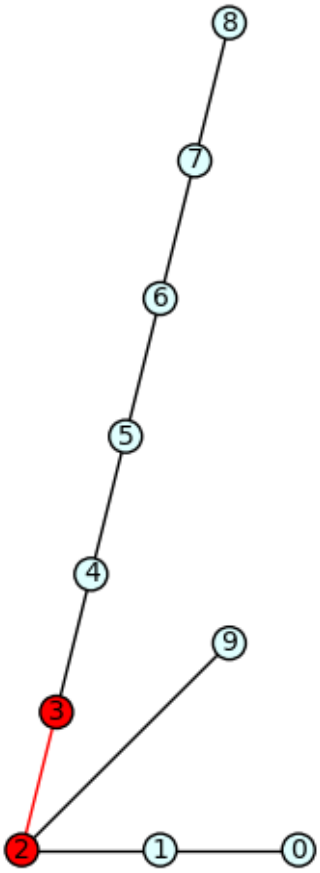
[A_2(2)]



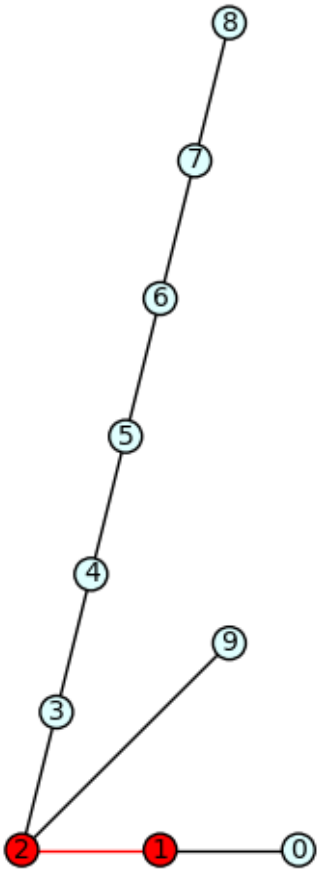
[A_2(2)]



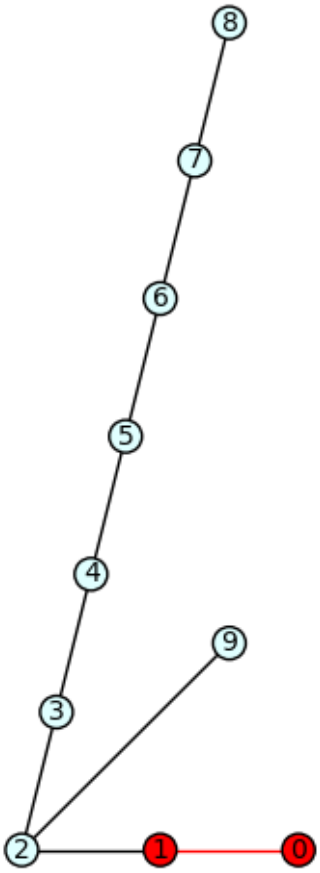
[A_2(2)]



[A_2(2)]

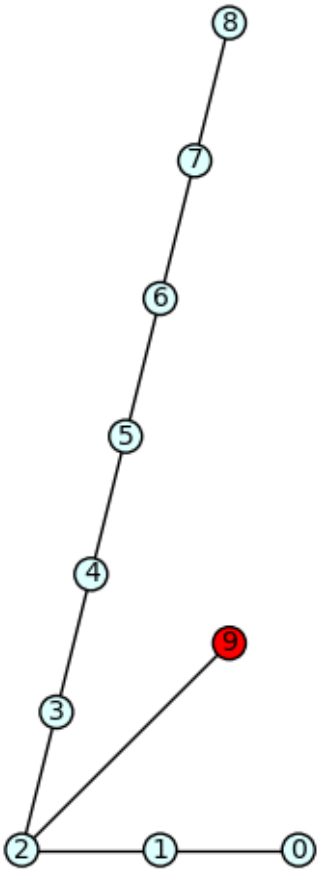


[A_2(2)]

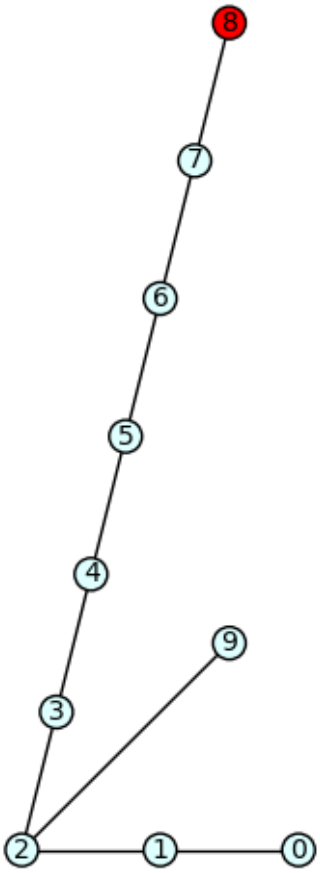


[A_2(2)]

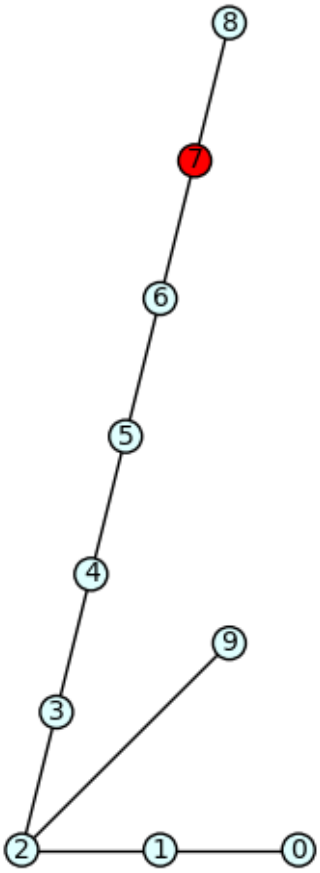
Rank 1:



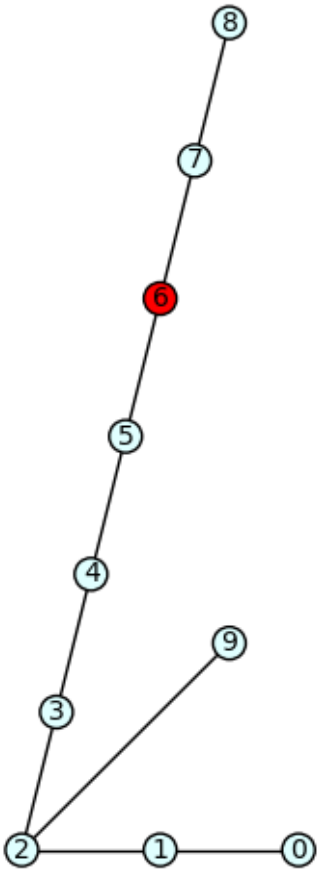
[A_1(2)]


8
7
6
5
4
9
3
2
1
0

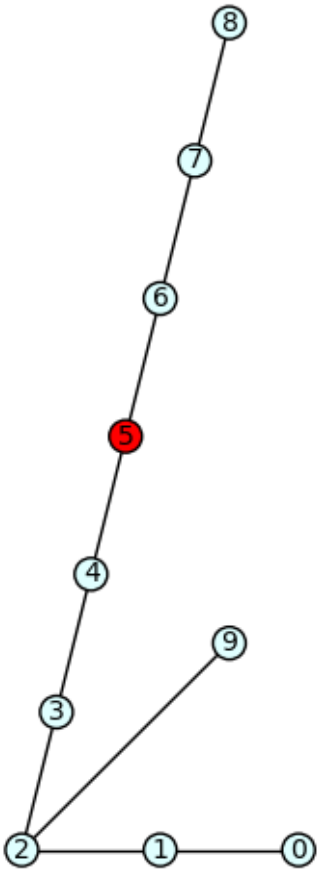
[]



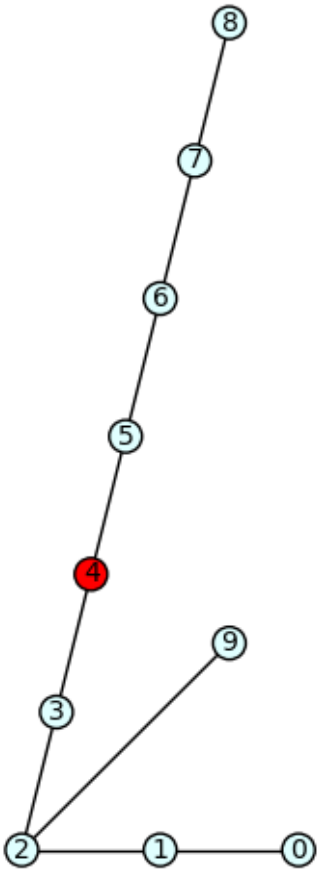
[A_1(2)]



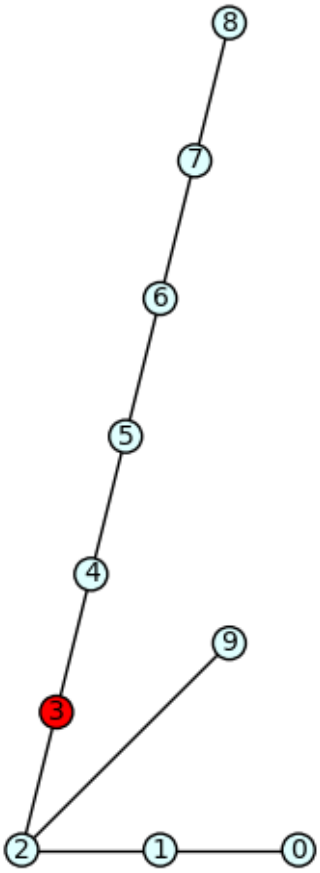
[A_1(2)]



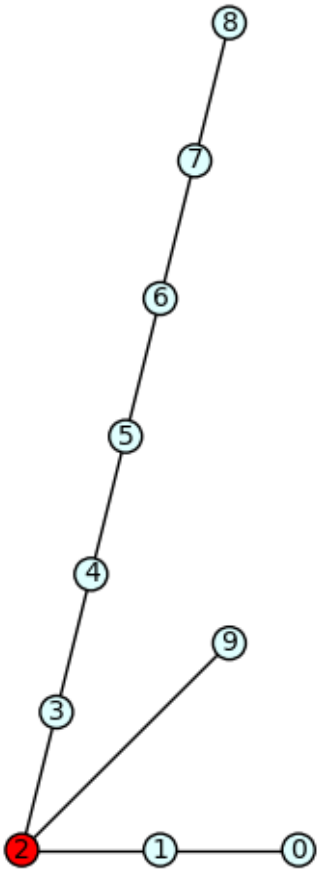
[A_1(2)]



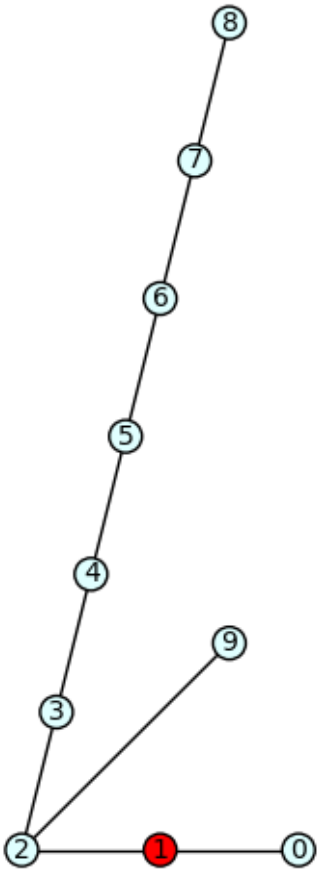
[A_1(2)]



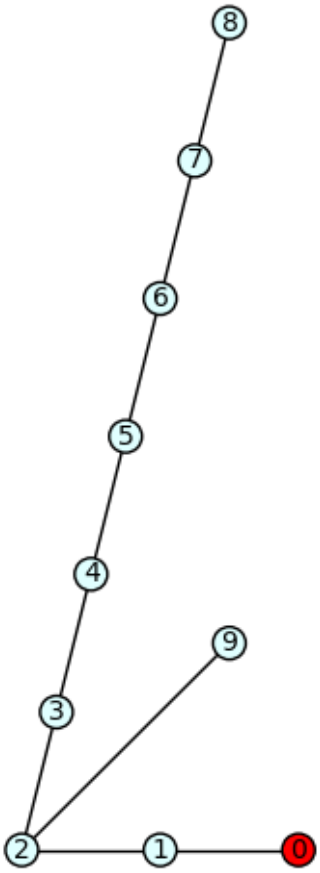
[A_1(2)]



[A_1(2)]



[A_1(2)]



[A_1(2)]

Rank 0:

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[23], line 5
      3 rank_i_subgraphs = [H for H in subgraphs if len(H.vertices() ) == i]
      4 show(f"Rank {i}: ")
----> 5 ade_types = get_all_rank_n_types(i)
      6 for H in rank_i_subgraphs:
      7     plot_subgraph(H)

Cell In[11], line 124, in get_all_rank_n_types(n)
    115     return [
    116         (f"A_{n}(2)", matrix_A_n_2(Integer(2))),
    117         (f"G_{Integer(2)}", matrix_G_2())
    118     ]
    119 else:
```

```
    120     Ms = [
    121         (f"A_{n}(2)", matrix_A_n_2(n)),
    122         (f"B_{n}(2)", matrix_B_n_2(n)),
    123         (f"C_{n}(2)", matrix_C_n_2(n)),
--> 124         (f"D_{n}(2)", matrix_D_n_2(n))
    125     ]
    126     if n == Integer(6):
    127         Ms.append(
    128             (f"E_6(2)", matrix_E_6_2() )
    129         )

Cell In[11], line 55, in matrix_D_n_2(n)
     54 def matrix_D_n_2(n):
---> 55     return IntegralLattice(f"D{n}").twist(-Integer(2)).gram_matrix()

File /usr/lib/python3.11/site-packages/sage/modules/
 ↪free_quadratic_module_integer_symmetric.py:242, in IntegralLattice(data, basi
    240 else:
    241     from sage.combinat.root_system.cartan_matrix import CartanMatrix
--> 242     inner_product_matrix = CartanMatrix(data)
    243 if basis is None:
    244     basis = matrix.identity(ZZ, inner_product_matrix.ncols())

File /usr/lib/python3.11/site-packages/sage/misc/classcall_metaclass.pyx:320, i␣
 ↪sage.misc.classcall_metaclass.ClasscallMetaclass.__call__ (build/cythonized/
 ↪sage/misc/classcall_metaclass.c:1903)()
    318 """
    319 if cls.classcall is not None:
--> 320     return cls.classcall(cls, *args, **kwds)
    321 else:
    322     # Fast version of type.__call__(cls, *args, **kwds)

File /usr/lib/python3.11/site-packages/sage/combinat/root_system/cartan_matrix.
 ↪py:307, in CartanMatrix.__classcall_private__(cls, data, index_set,␣
 ↪cartan_type, cartan_type_check, borcherds)
    305         data[(reverse[j], reverse[i])] = -1
    306 else:
--> 307     M = matrix(data)
    308     if borcherds:
    309         if not is_borcherds_cartan_matrix(M):

File /usr/lib/python3.11/site-packages/sage/matrix/constructor.pyx:643, in sage
 ↪matrix.constructor.matrix (build/cythonized/sage/matrix/constructor.c:2811)()
    641 """
    642 immutable = kwds.pop('immutable', False)
--> 643 M = MatrixArgs(*args, **kwds).matrix()
    644 if immutable:
    645     M.set_immutable()
```

```
File /usr/lib/python3.11/site-packages/sage/matrix/args.pyx:656, in sage.matrix
 ↪args.MatrixArgs.matrix (build/cythonized/sage/matrix/args.c:8165)()
    654     True
    655 """
--> 656 self.finalize()
    657
    658 cdef Matrix M

File /usr/lib/python3.11/site-packages/sage/matrix/args.pyx:881, in sage.matrix
 ↪args.MatrixArgs.finalize (build/cythonized/sage/matrix/args.c:9868)()
    879     self.typ = self.get_type()
    880     if self.typ == MA_ENTRIES_UNKNOWN:
--> 881         raise TypeError(f"unable to convert {self.entries!r} to a␣
 ↪matrix")
    882
    883 # Can we assume a square matrix?

TypeError: unable to convert 'D0' to a matrix
```