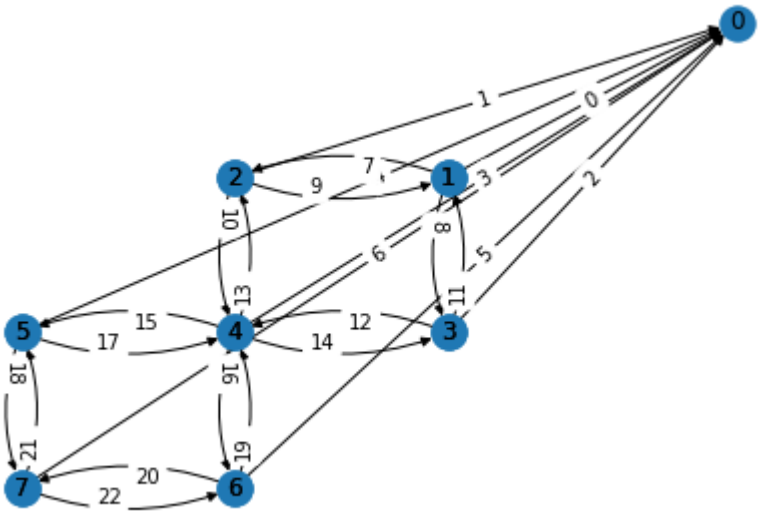
In [1]:
```
%run Function_set_dic.ipynb
import networkx as nx
import numpy as np
from collections import defaultdict
import random
from numpy import random
import matplotlib.pyplot as plt
```

In [2]:
```
network,pos = Make_Question(3,3,Density = 0.85, option = "1-norm", distance

print(network)
```

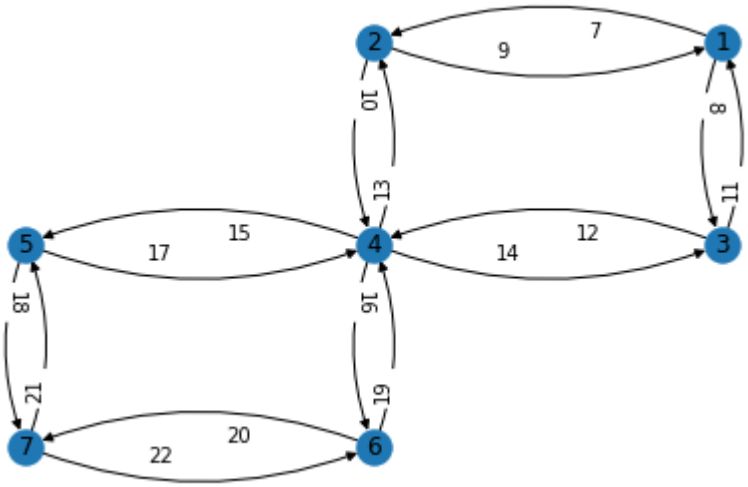
```
[[0. 0. 0. 0. 0. 0. 0. 0.]
 [1. 0. 1. 1. 0. 0. 0. 0.]
 [1. 1. 0. 0. 1. 0. 0. 0.]
 [1. 1. 0. 0. 1. 0. 0. 0.]
 [1. 0. 1. 1. 0. 1. 1. 0.]
 [1. 0. 0. 0. 1. 0. 0. 1.]
 [1. 0. 0. 0. 1. 0. 0. 1.]
 [1. 0. 0. 0. 0. 1. 1. 0.]]
```

In [3]:
```
G, compelete_label, graph = complete_Graph(network,pos)
print(compelete_label)
```

```
{(1, 0): '0', (2, 0): '1', (3, 0): '2', (4, 0): '3', (5, 0): '4', (6, 0): '5',
(7, 0): '6', (1, 2): '7', (1, 3): '8', (2, 4): '10', (3, 4): '12', (4, 5): '1
5', (4, 6): '16', (5, 7): '18', (6, 7): '20', (2, 1): '9', (3, 1): '11', (4,
2): '13', (4, 3): '14', (5, 4): '17', (6, 4): '19', (7, 5): '21', (7, 6): '22'}
```



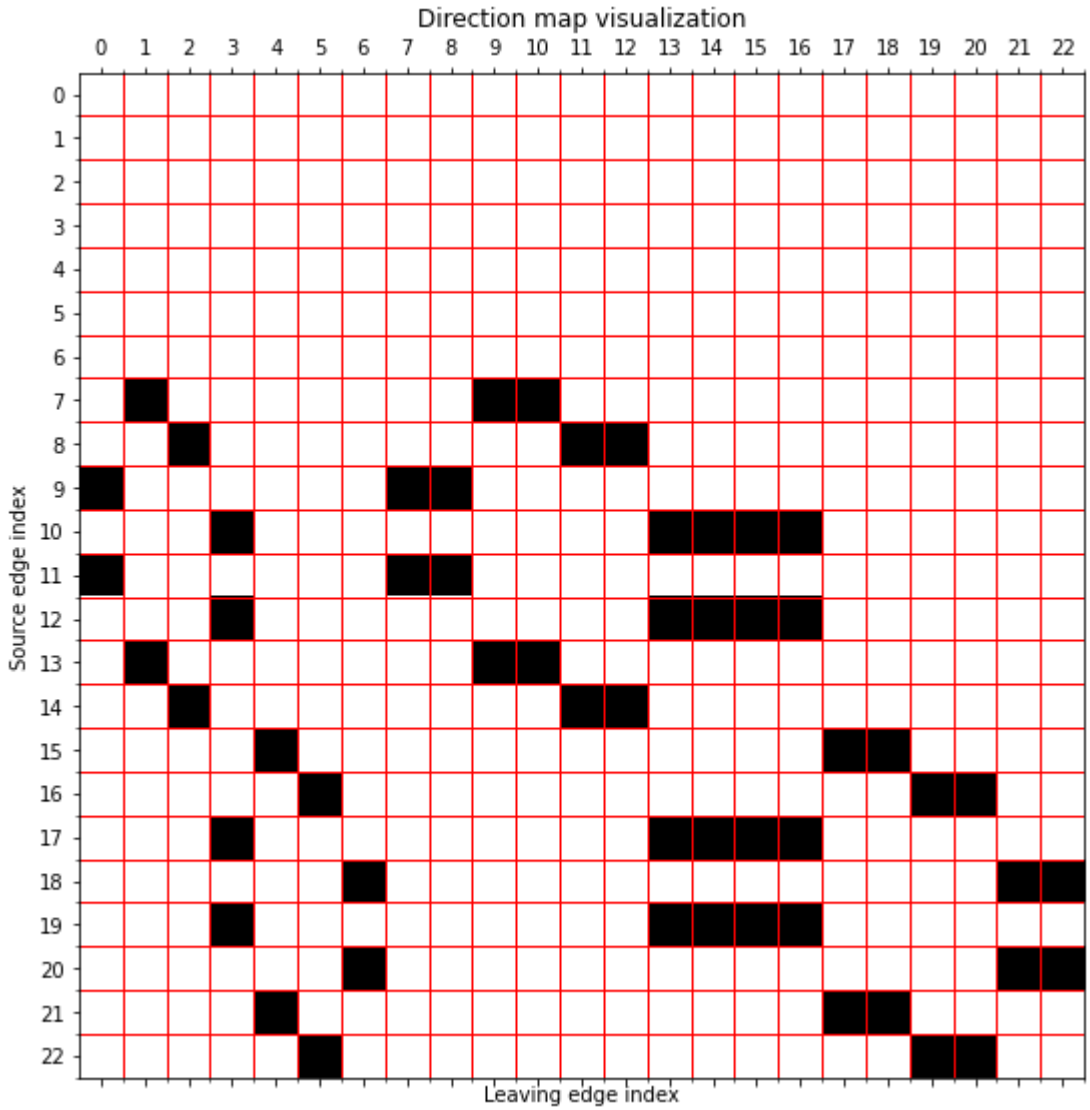
In [4]:
```
1 G_prime, sub_edge_label, sub_graph = sub_Graph (network,pos)
```

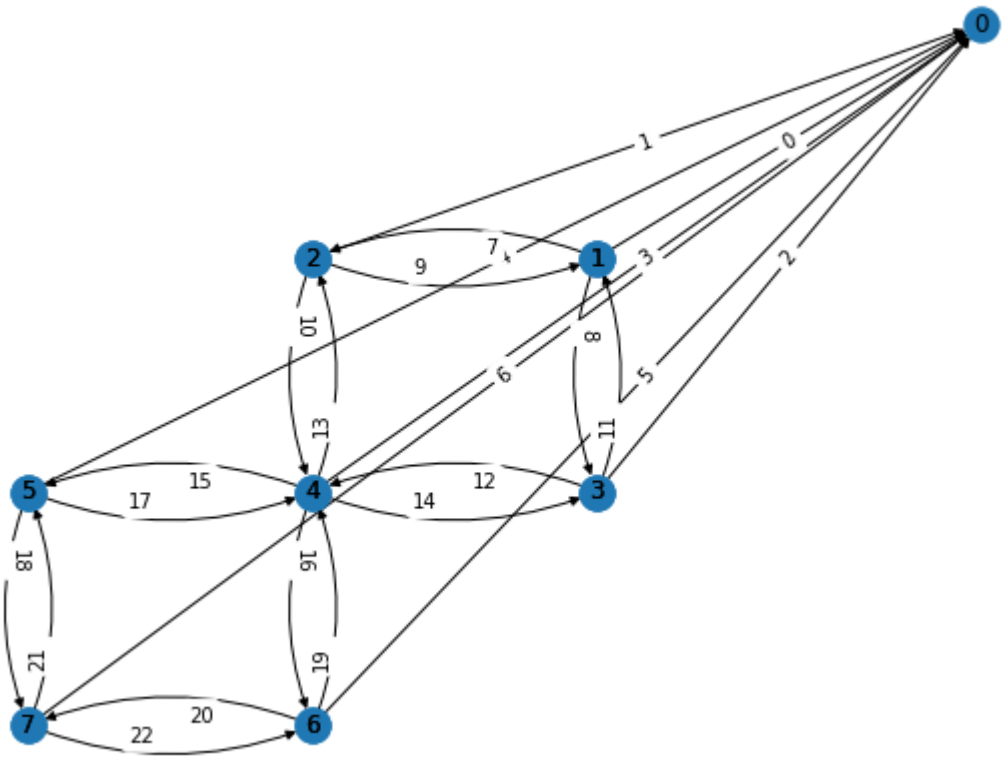


In [5]:
```
1 %run Function_set_dic.ipynb
2 Final_map, lowest_edge_number = edge_map(G, compelete_label, sub_edge_label)
3
```

In [89]:

```
#matfig = plt.figure(figsize=(Final_map.shape[0],Final_map.shape[1]))
matfig = plt.figure(figsize=(9,9))
plt.matshow(Final_map,cmap=plt.cm.binary,fignum=matfig.number)
ax = plt.gca()
#plt.matshow(Final_map,cmap=plt.cm.binary)
plt.xlabel('Leaving edge index')
plt.ylabel('Source edge index')
plt.title ("Direction map visualization")

# Major ticks
ax.set_xticks(np.arange(0, Final_map.shape[0], step=1))
ax.set_yticks(np.arange(0, Final_map.shape[0], step=1))

# Labels for major ticks
ax.set_xticklabels(np.arange(0, Final_map.shape[0], step=1))
ax.set_yticklabels(np.arange(0, Final_map.shape[0], step=1))

# Minor ticks
ax.set_xticks(np.arange(0.5, Final_map.shape[0]+0.5, step=1), minor=True)
ax.set_yticks(np.arange(0.5, Final_map.shape[0]+0.5, step=1), minor=True)

# Gridlines based on minor ticks
ax.grid(which='minor', color='r', linestyle='-', linewidth=1)

plt.show()


plt.figure(2,figsize=(8,6))
_, _, _ = complete_Graph(network,pos)
```


Direction map visualization
Source edge index
Leaving edge index
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22



In [90]:
```
# How many panel ---> Max electricity level wire could reach
Q = len(pos)-1

# How many edges
edge_number = len(compelete_label)

# make dictionary
edge_dictionary,Total = make_dictionary(edge_number, Q)

# basic_cost shape => (1,number_of_edge)
# such as : basic_cost = np.vstack(np.ones(number_of_edge))
basic_cost = np.ones(edge_number)

# flow cost (p)
flow_cost = np.ones(edge_number)

# initialize a QUBO matrix needed
QUBO_matrix_initial = np.zeros((edge_number*Q,edge_number*Q))
```

In [ ]:
```

```

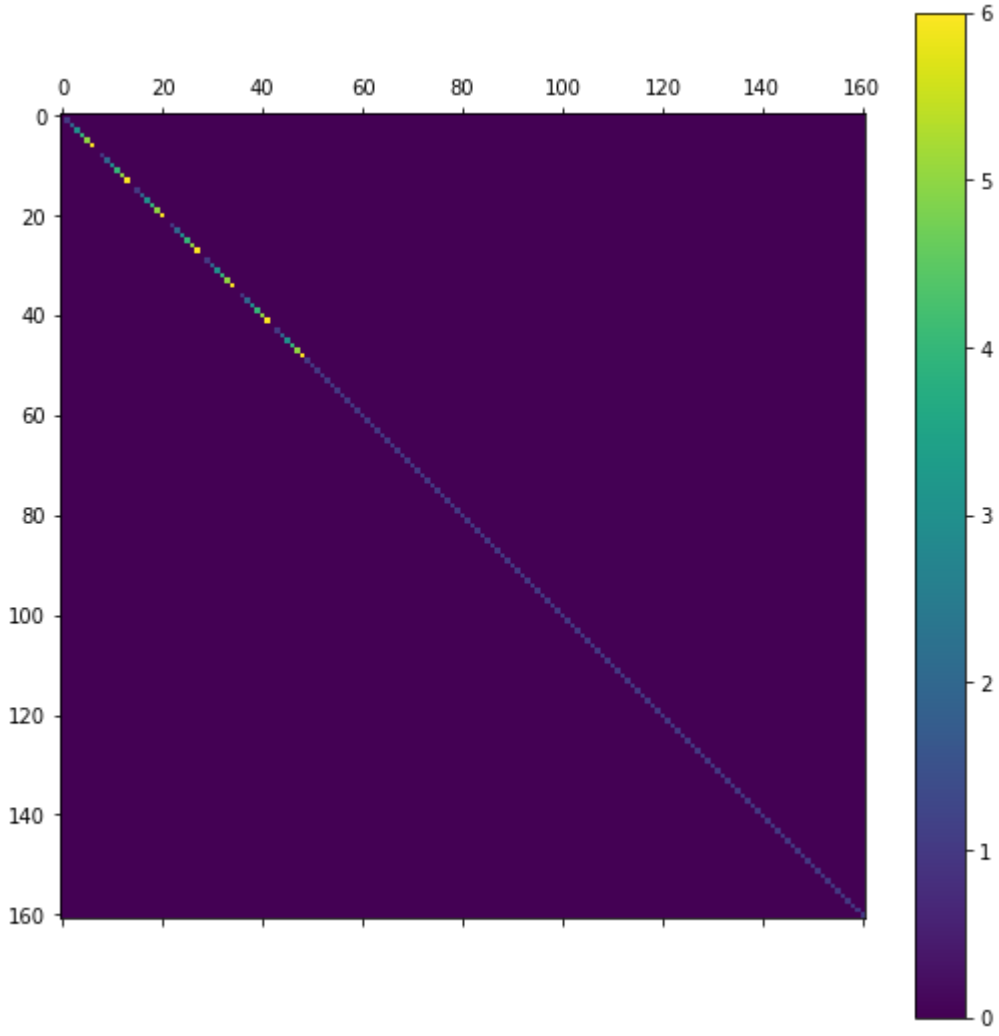
In [91]:
```
edge_dictionary["x_%d_%d"%(0,0)]
```

Out[91]: 0

In [92]:
```
%run Function_set_dic.ipynb

QUBO_Obj = Objective(Q, edge_number, edge_dictionary, QUBO_matrix_initial, b
```

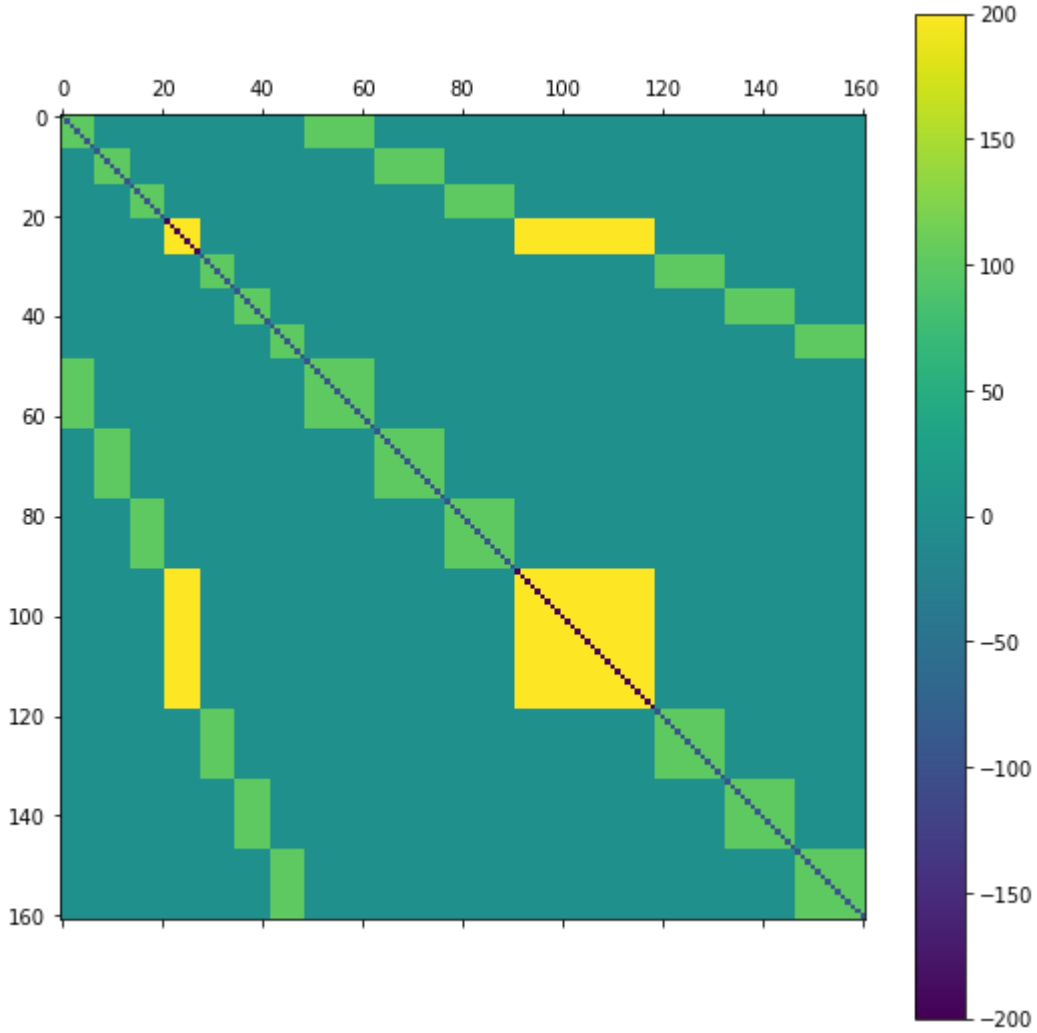
In [93]:
```
#a = np.array([1,3,5])
#b = np.array([2,4,6,8,10,12])
#x,y = np.meshgrid(a,b)
#print(x,y)
#np.concatenate((x.reshape(-1,1), y.reshape(-1,1)), axis=1)
penal = 50
```

In [94]:
```
matfig = plt.figure(figsize=(9,9))
plt.matshow(QUBO_Obj,fignum=matfig.number)
plt.colorbar()
plt.show()
```



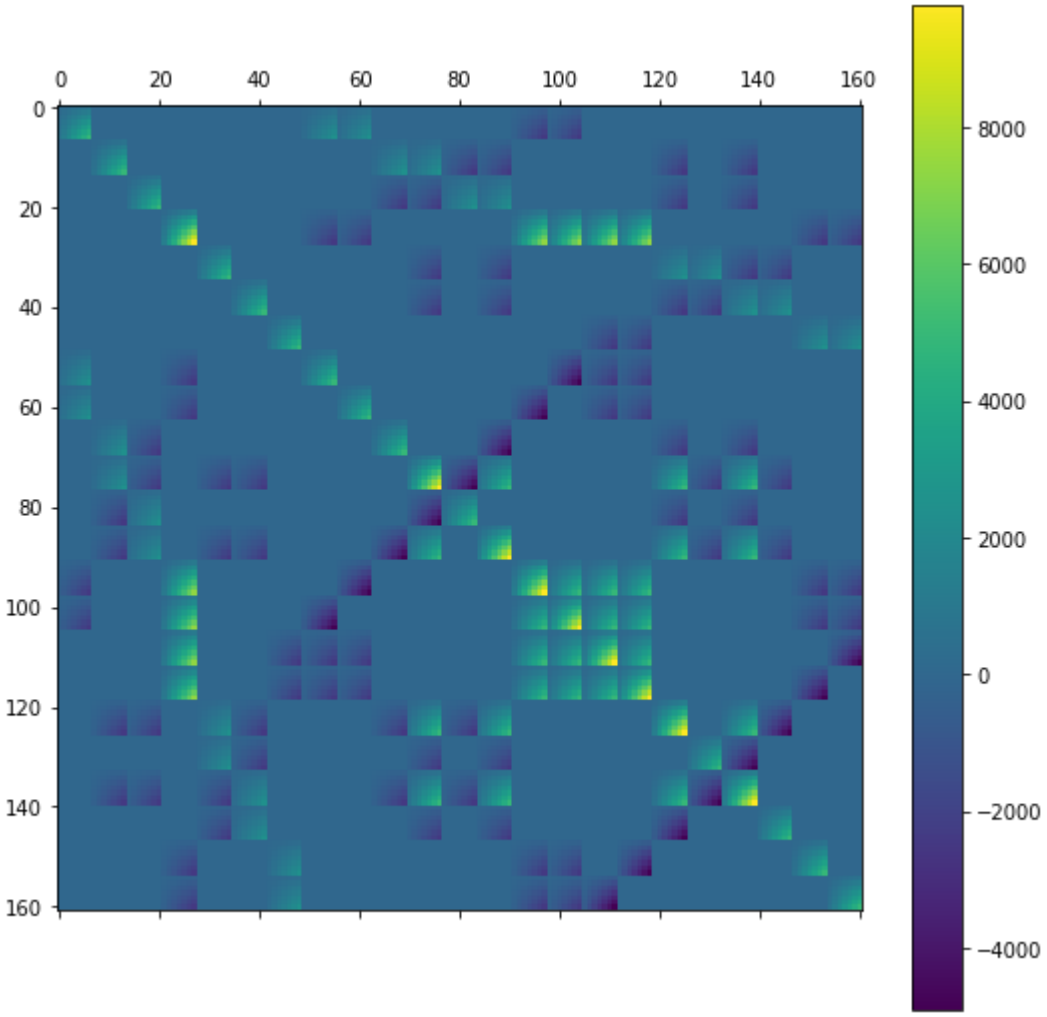
In [95]:
```
%run Function_set_dic.ipynb
QUBO_Constraint_1 = Constraint_1(Final_map, Q, edge_number, edge_dictionary,
```

In [96]:
```
matfig = plt.figure(figsize=(9,9))
plt.matshow(QUBO_Constraint_1,fignum=matfig.number)
plt.colorbar()
plt.show()
```



In [97]:
```
%run Function_set_dic.ipynb
QUBO_Constraint_2 = Constraint_2(Final_map, Q, edge_number, edge_dictionary,
```

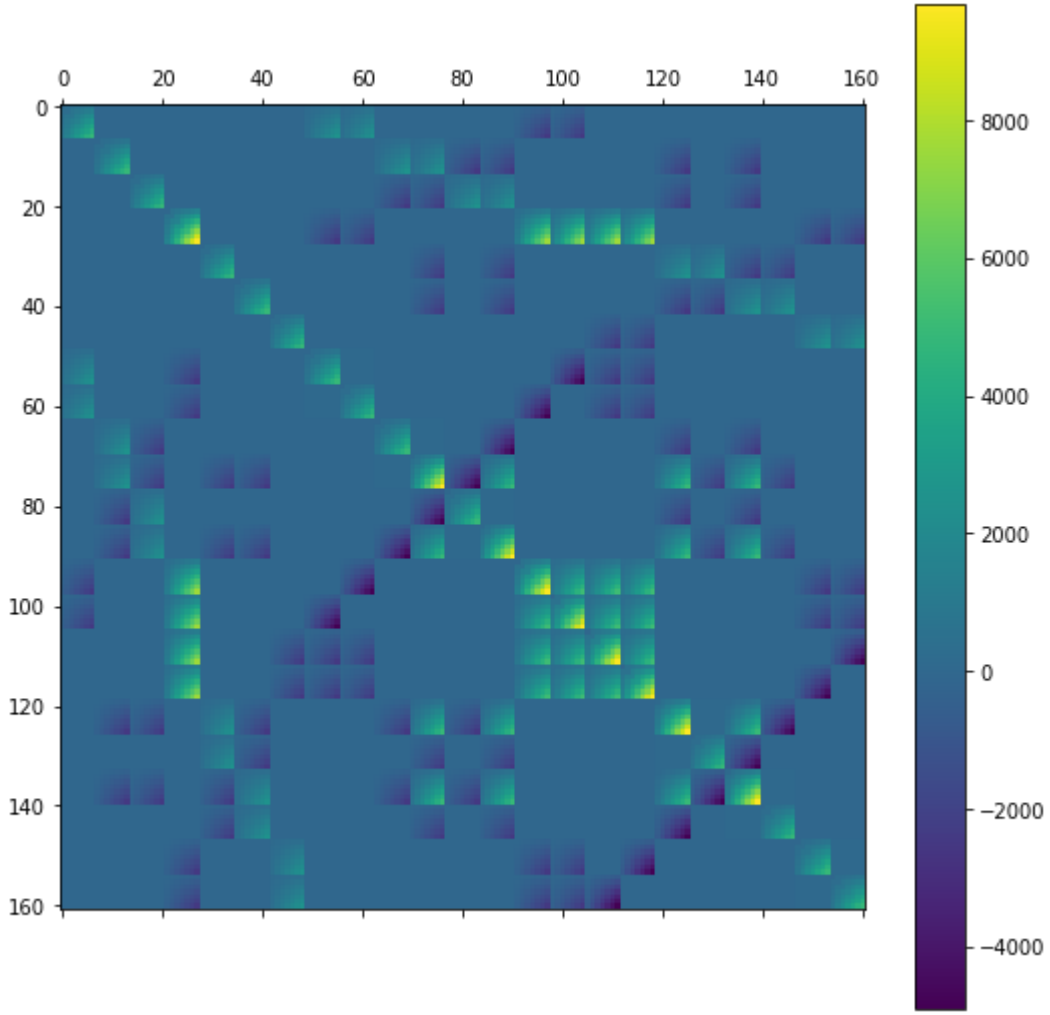
In [98]:
```
matfig = plt.figure(figsize=(9,9))
plt.matshow(QUBO_Constraint_2,fignum=matfig.number)
plt.colorbar()
plt.show()
```



In [99]:
```
QUBO_matrix = QUBO_Obj + QUBO_Constraint_1 + QUBO_Constraint_2

print(np.shape(QUBO_matrix ))
```

```
(161, 161)
```

In [100]:
```
matfig = plt.figure(figsize=(9,9))
plt.matshow(QUBO_matrix,fignum=matfig.number)
plt.colorbar()
plt.show()
```

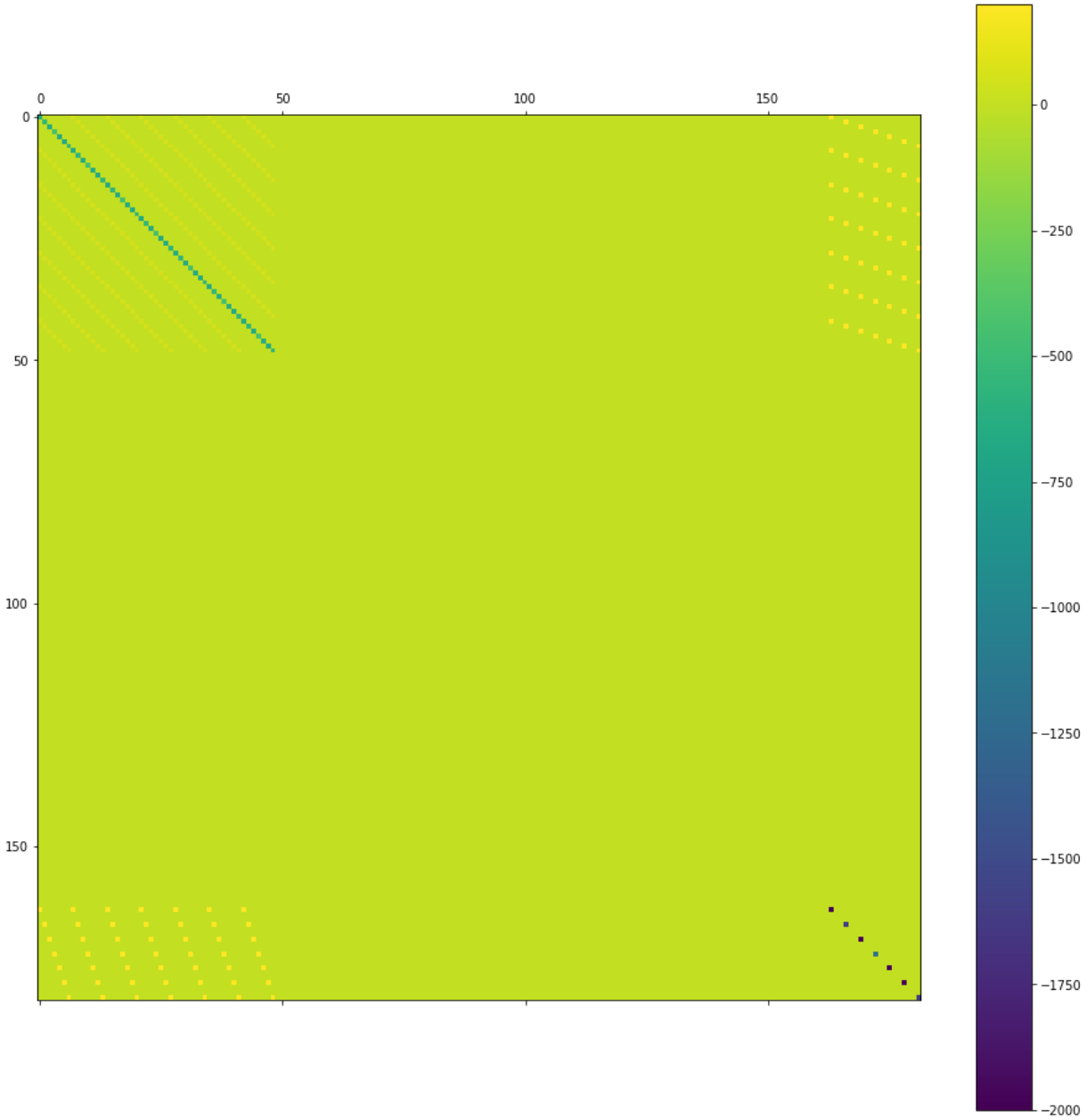


In [101]:
```
%run Function_set_dic.ipynb
m_t_u,m_t_l = capacity_subtree_limit_generator(pos,Q)

print(m_t_u)
m_t_l[:]=1
print(m_t_l)


QUBO_matrix_copy = QUBO_matrix
```

```
[7. 6. 7. 5. 7. 7. 6.]
[1. 1. 1. 1. 1. 1. 1.]
```

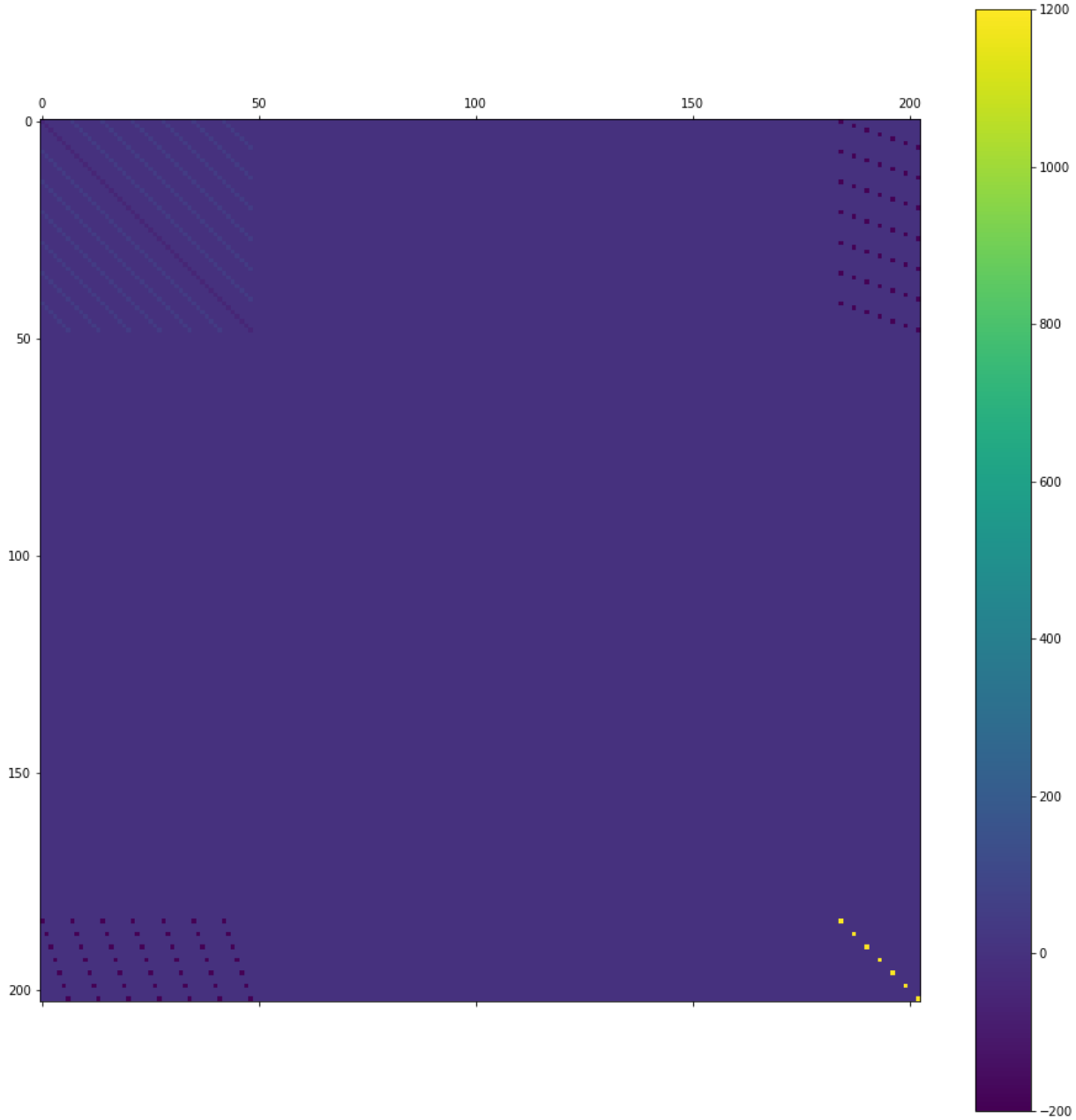
In [102]:
```
QUBO_matrix, QUBO_C3_p1, edge_dictionary_C31 = Constraint_3_part_1(Final_map
```

In [103]:
```
matfig = plt.figure(figsize=(16,16))
plt.matshow(QUBO_C3_p1 ,fignum=matfig.number)
plt.colorbar()
plt.show()
```

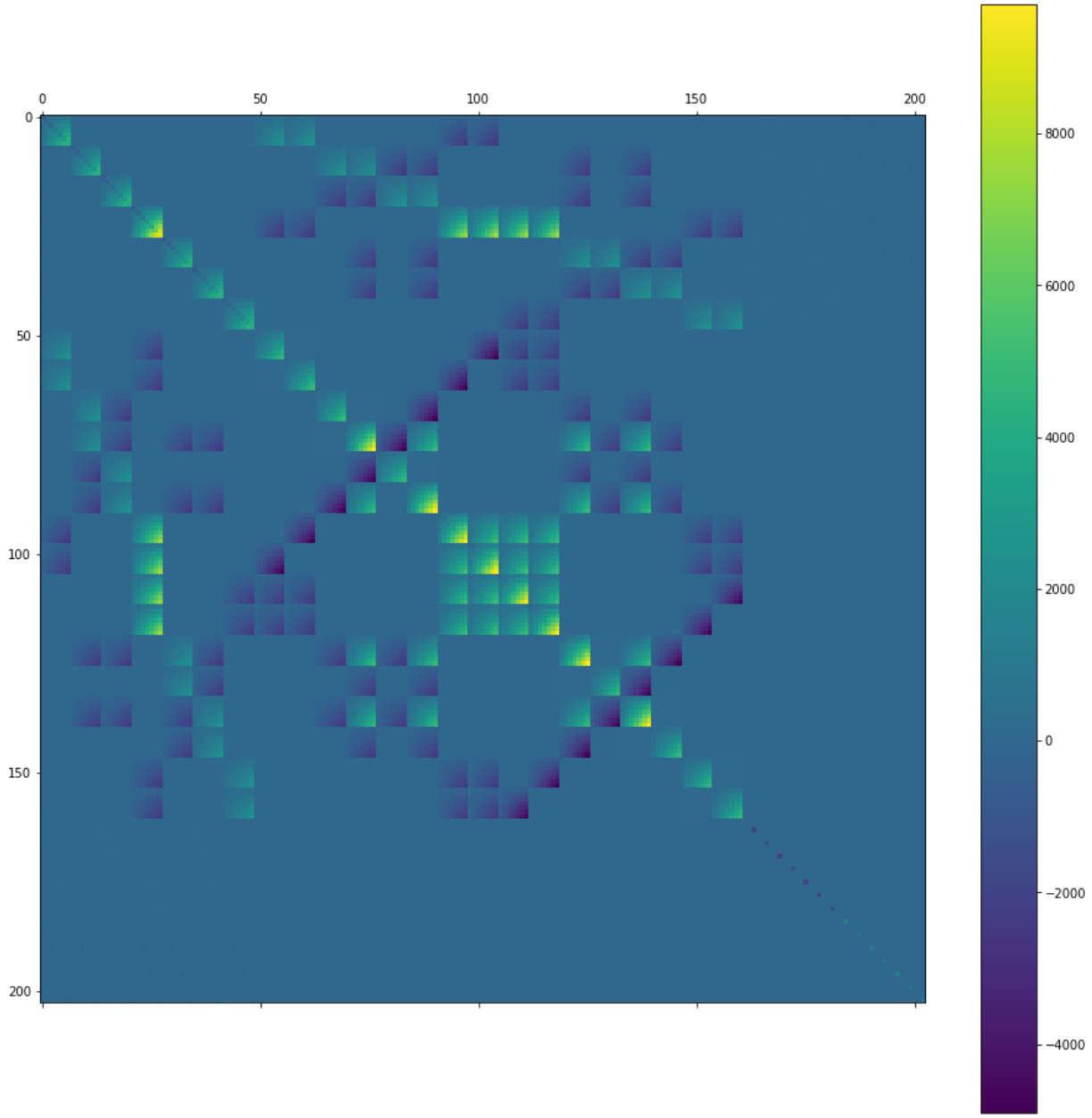


In [104]:
```
QUBO_matrix, QUBO_C3_p2, edge_dictionary_C32 = Constraint_3_part_2(Final_map
```

In [105]:
```
matfig = plt.figure(figsize=(16,16))
plt.matshow(QUBO_C3_p2,fignum=matfig.number)
plt.colorbar()
plt.show()
```



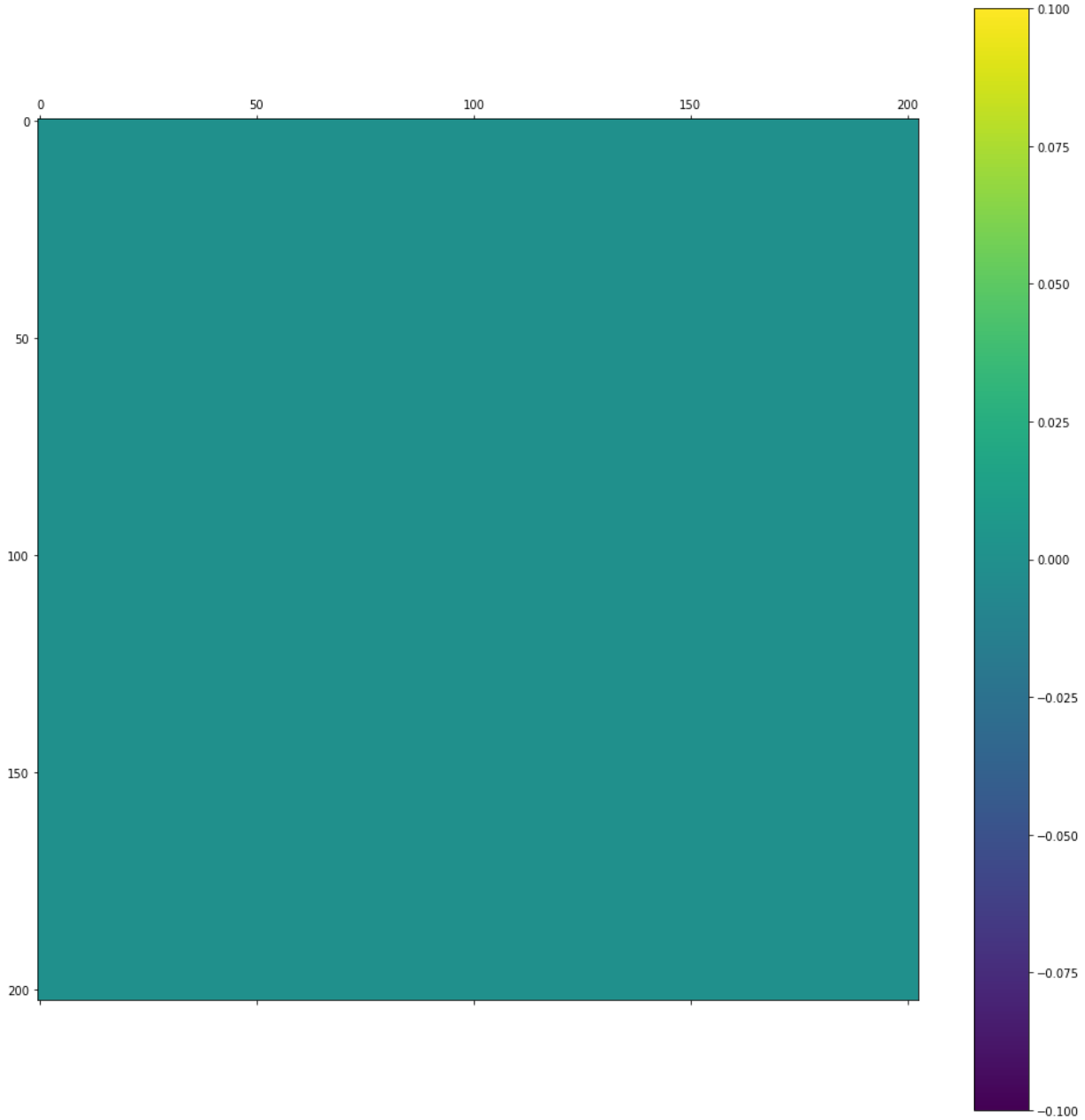
In [106]:
```
matfig = plt.figure(figsize=(16,16))
plt.matshow(QUBO_matrix,fignum=matfig.number)
plt.colorbar()
plt.show()
```



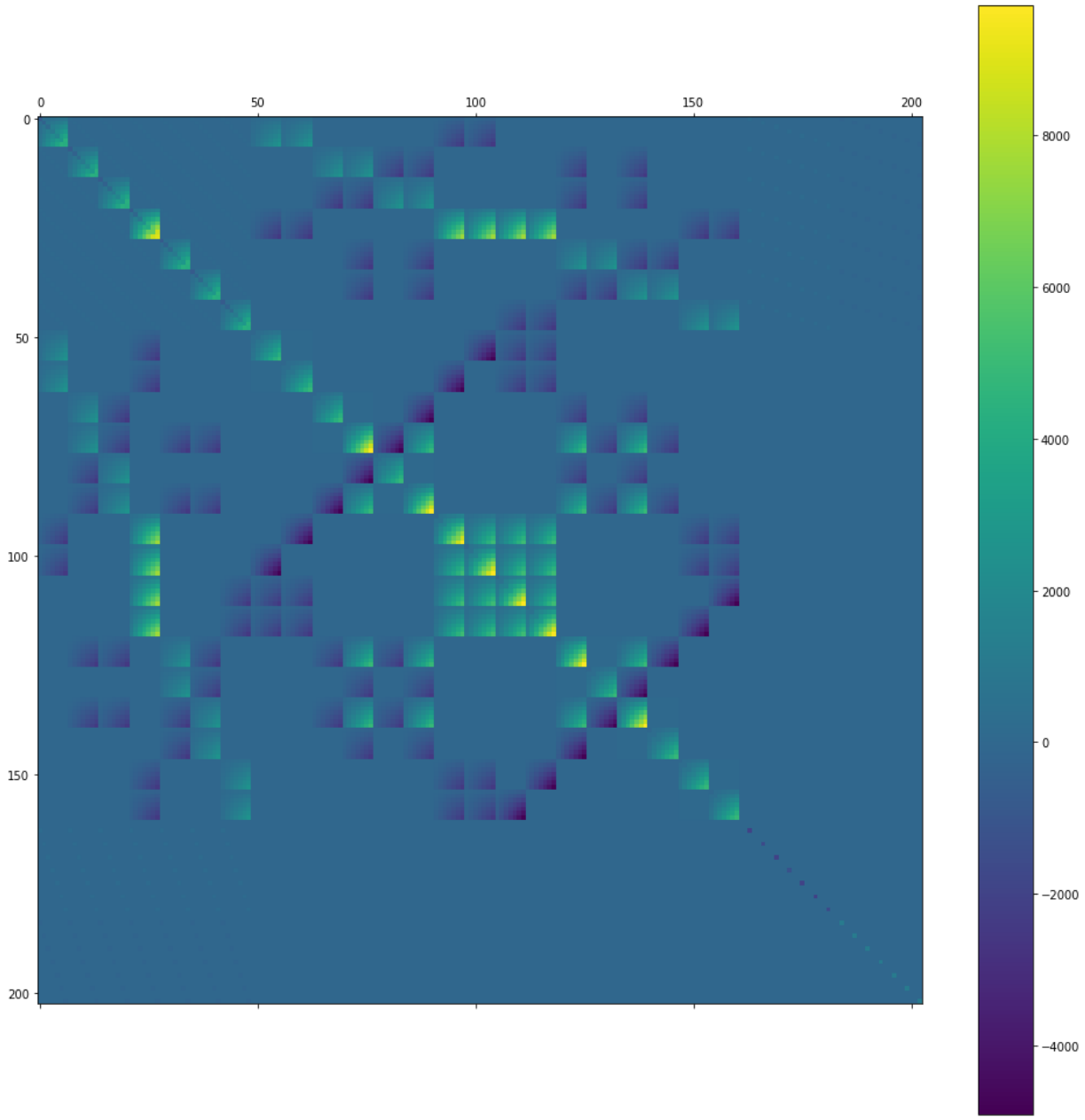
In [107]:
```
# Bus edge Ban map( Q ** 2)  可以通过就是 1 不可以就是 0
Ban_map_Bus = np.ones( Q**2 )



Final_QUBO_matrix, QUBO_Constraint_4 = Constraint_4(QUBO_matrix, Q, Ban_map_

print(np.shape(Final_QUBO_matrix))
```

```
(203, 203)
```

In [108]:
```
matfig = plt.figure(figsize=(17,17))
plt.matshow(QUBO_Constraint_4,fignum=matfig.number)
plt.colorbar()
plt.show()
```



In [109]:
```
matfig = plt.figure(figsize=(17,17))
plt.matshow(Final_QUBO_matrix,fignum=matfig.number)
plt.colorbar()
plt.show()
```



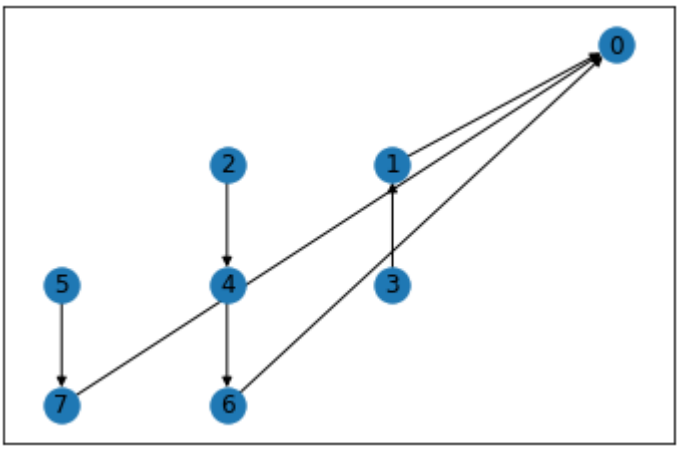
In [110]:
```
from collections import defaultdict

from dwave.system.samplers import DWaveSampler
from dwave.system.composites import EmbeddingComposite
import networkx as nx
import numpy as np
import dwave.inspector
import dimod
from dwave.system import LeapHybridSampler
import matplotlib
from matplotlib import pyplot as plt
```

In [111]:
```
QUBO = Final_QUBO_matrix
```

In [112]:

```
width,height = np.shape(QUBO)

QUBO_dictionary = defaultdict(int)
for i in range(width):
    for j in range(height):
        QUBO_dictionary[(i,j)] = QUBO[i,j]


# Select a solver
sampler = LeapHybridSampler()

sampleset = sampler.sample_qubo(QUBO)


OP = sampleset.to_pandas_dataframe()
import pandas as pd

OP = OP.sort_values("energy")
OP_final = OP.to_numpy()

rank = 0

OP_opt = OP_final[rank,:].flatten()
sample = OP_opt.astype(int)

sampleset.info['qpu_access_time']
sampleset.info

location = np.where(OP_opt[:-2])[0]
print(location)
Q = len(pos)-1
number_of_edge = len(compelete_label)

final_index = location[location <= Q*number_of_edge ]
print(final_index)


final_G = nx.DiGraph()
final_G = nx.from_numpy_array(np.zeros_like(network),create_using=nx.DiGraph
for value in final_index:
    print(str(value//Q))
    final_G .add_edges_from([edge for edge, label in compelete_label.items()

nx.draw_networkx(final_G, pos)
```

```
[  0  36  37  42  74  81 112 130 163 164 166 169 170 172 173 174 175 176
 178 180 181 188 189 191 194 195 198 201]
[  0  36  37  42  74  81 112 130]
0
5
5
6
10
11
16
18
```



In [113]:
```
1 OP
```

Out[113]:

	0	1	2	3	4	5	6	7	8	9	...	195	196	197	198	199	200	201	202	energy	num_occu
0	1	0	0	0	0	0	0	0	0	0	...	1	0	0	1	0	0	1	0	-13693.0	

1 rows × 205 columns

In [ ]:
```
1
```

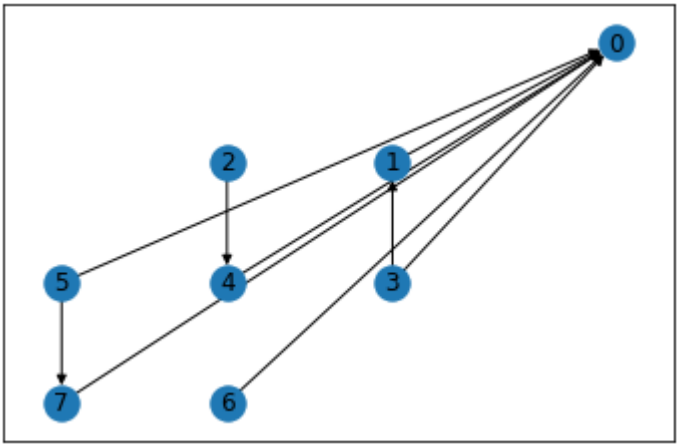
In [ ]:
```
1
```

In [114]:
```
1 print(Q)
2 print(final_index//Q)
3 print(final_index%Q)
```

```
7
[ 0  5  5  6 10 11 16 18]
[0 1 2 0 4 4 0 4]
```

In [87]:
```
1
2
3
4     #G.add_edges_from([(1, 2), (1, 3), (2, 3)])
```

In [88]:
```
1 nx.draw_networkx(final_G, pos)
2
```



In [57]:
```
1 [edge for edge, label in compelete_label.items() if label == str(0)]
```

Out[57]:
```
[(1, 0)]
```

In [ ]:
```
1
```

In [ ]:
```
1
```