In [90]:
```
class HopfieldNetwork:
    def __init__(self,num_neurons):
        self.num_neurons=num_neurons
        self.weights=np.zeros((num_neurons,num_neurons))

    def sigmoid(self,x):
        return (1-np.exp(-x))/(1+np.exp(-x))

    def train(self,vectors):
        for vector in vectors:
            vector=np.reshape(vector,(self.num_neurons,1))
            self.weights+=np.dot(vector,vector.T)
        print("Weights : ",self.weights)
        print()

    def recall(self,input_vector):
        output=np.dot(self.weights,input_vector)
        output=self.sigmoid(output)
        output=np.sign(output)
        return output
```

In [91]:
```
network=HopfieldNetwork(8)

vectors=np.array([[1,1,1,-1,-1,-1,1,-1],
                  [-1,1,-1,1,-1,1,-1,1],
                  [-1,-1,1,1,-1,-1,1,1],
                  [-1,1,-1,-1,1,1,-1,1]
                 ])

network.train(vectors)

vectors=np.array([[1,1,1,-1,-1,0,1,-1],
                  [-1,1,-0,1,-1,1,-1,1],
                  [-1,0,1,1,-1,-1,1,1],
                  [-1,1,-1,-1,0,1,-1,1]
                 ])

for vector in vectors:
    output=network.recall(vector)
    print("input : ",vector)
    print("output : ",output)
```

```
Weights :  [[ 4.  0.  2. -2.  0. -2.  2. -4.]
 [ 0.  4. -2. -2.  0.  2. -2.  0.]
 [ 2. -2.  4.  0. -2. -4.  4. -2.]
 [-2. -2.  0.  4. -2.  0.  0.  2.]
 [ 0.  0. -2. -2.  4.  2. -2.  0.]
 [-2.  2. -4.  0.  2.  4. -4.  2.]
 [ 2. -2.  4.  0. -2. -4.  4. -2.]
 [-4.  0. -2.  2.  0.  2. -2.  4.]]

input :  [ 1  1  1 -1 -1  0  1 -1]
output :  [ 1.  1.  1. -1. -1. -1.  1. -1.]
input :  [-1  1  0  1 -1  1 -1  1]
output :  [-1.  1. -1.  1. -1.  1. -1.  1.]
input :  [-1  0  1  1 -1 -1  1  1]
output :  [-1. -1.  1.  1. -1. -1.  1.  1.]
input :  [-1  1 -1 -1  0  1 -1  1]
output :  [-1.  1. -1. -1.  1.  1. -1.  1.]
```

In [ ]:

In [ ]: