```python
import numpy as np
from more_itertools import unique_everseen


def img_from_sp(sp_img, sp_label):
    """
    restore an image from superpixels
    :param sp_img: superpixel image
    :param sp_label: superpixel segmentaion label maps
    :return: rimg: restored image
    """
    assert len(sp_img)== sp_label.max()+1, "inconsistent superpixel number"
    rimg = np.zeros(sp_label.shape+(3,))
    for i in range(len(sp_img)):
        rimg[sp_label==i, :] = sp_img[i]
    return rimg.astype('uint8')


def lbmap_from_sp(sp_lbmap, sp_label):
    """
    restore a label map from superpixels
    :param sp_lbmap: superpixel-level label
    :param sp_label: superpixel segmentation label
    :return: r_lbmap: restored label map
    """
    assert len(sp_lbmap)== sp_label.max()+1, "inconsistent superpixel number"
    r_lbmap = np.zeros(sp_label.shape)
    for i in range(len(sp_lbmap)):
        r_lbmap[sp_label==i] = sp_lbmap[i]
    return r_lbmap


def adjacent_matrix(sp_label):
    """
    adjacent matrix of superpixels
    :param sp_label:
    :return: edges: edges of the undirected graph
    """
    # should be improve
    sp_label_l = np.zeros(sp_label.shape, dtype='int')
    sp_label_u = np.zeros(sp_label.shape, dtype='int')
    sp_label_u[:-1, :] = sp_label[1:, :]
    sp_label_l[:, :-1] = sp_label[:, 1:]

    dl = sp_label_l - sp_label
    dl[:, -1] = 0

    du = sp_label_u - sp_label
    du[-1, :] = 0

    node_out = np.concatenate((sp_label[dl != 0], sp_label[du != 0]))
    node_in = np.concatenate((sp_label_l[dl != 0], sp_label_u[du != 0]))
    edges = np.stack((node_out, node_in), 1)
    edges = np.concatenate((edges, edges[:, ::-1]), 0)
    edges = np.unique(edges, axis=0)
    return edges


def make_graph(sp_label):
```

```
    """
    add boundary connections and far connections to the graph
    :param sp_label: superpixel segmentation
    :return: edges: edges of the new undirected graph
    """
    edges = adjacent_matrix(sp_label)
    sp_num = sp_label.max()

    # add boundary connections
    top_bd = np.unique(sp_label[0, :])
    left_bd = np.unique(sp_label[:, 0])
    bottom_bd = np.unique(sp_label[-1, :])
    right_bd = np.unique(sp_label[:, -1])

    boundary = np.concatenate((top_bd, left_bd, bottom_bd, right_bd))
    boundary = np.unique(boundary)

    nb = len(boundary)
    node_out = boundary.repeat(nb)
    node_in = node_out.reshape((nb, nb)).T.ravel()

    bd_edges = np.stack((node_out, node_in), axis=1)
    bd_edges = bd_edges[bd_edges[:, 0] != bd_edges[:, 1]]

    edges = np.concatenate((edges, bd_edges), 0)
    edges = np.unique(edges, axis=0)

    # add far connections
    far_in = []
    far_out = []
    for i in range(sp_num):
        temp = edges[edges[:, 0] == i, 1]
        _far_in = []
        for t in temp:
            _far_in.append(edges[edges[:, 0] == t, 1])
        _far_in = np.concatenate(_far_in)
        _far_out = np.ones(_far_in.shape, dtype='int') * i
        far_in.append(_far_in)
        far_out.append(_far_out)

    far_in = np.concatenate(far_in)
    far_out = np.concatenate(far_out)
    far_edges = np.stack((far_out, far_in), 1)
    far_edges = np.unique(far_edges, axis=0)
    far_edges = far_edges[far_edges[:, 0] != far_edges[:, 1]]

    edges = np.concatenate((edges, far_edges), 0)
    # edges = [set(v) for v in edges]
    # edges = list(unique_everseen(edges))
    # edges = [list(v) for v in edges]
    edges = np.array(edges)
    return edges
```