```
from copy import deepcopy
from numpy import zeros, ix_, array, roll, argmax
def strings_overlap(s1, s2):
    if (s1 in s2): return len(s1)
    if (s2 in s1): return len(s2)
    return max([i for i in range(min(len(s1), len(s2)) + 1) if s1[len(s1)-i:] == s2[:i]])
def strings_merge(s1, s2):
    if (s1 in s2): return s2
    if (s2 in s1): return s1
    overlap = strings_overlap(s1, s2)
    return s1 + s2[overlap:]
def greedy_min_max_contain_string(strings):
    s = {i for i in strings}
    pairs = [(i, j, strings_overlap(i, j)) for i in s for j in s if not i is j]
    while len(pairs) > 1:
        max_pair = max(pairs, key=lambda x: x[2])
        tmp = strings_merge(max_pair[0], max_pair[1])
        s.remove(max_pair[0])
        s.remove(max_pair[1])
        s.add(tmp)
        pairs = [(i, j, strings_overlap(i, j)) for i in s for j in s if not i is j]
    return [i for i in s][0]
# print (greedy_min_max_contain_string(['hello', 'world', 'lol']))

def min_max_contain_string(strings):
#strings` must be set
    s = strings
    pairs = [(i, j, strings_overlap(i, j)) for i in s for j in s if not i is j]
    # print ('--------')
    # print (s)
    # print (pairs)
    results = list()
    if len(s) == 1:
        # print [i for i in s][0]
        return [i for i in s][0]
    for p in pairs:
        new_s = deepcopy(s)
        tmp = strings_merge(p[0], p[1])
        new_s.remove(p[0])
        new_s.remove(p[1])
        new_s.add(tmp)
        results.append(min_max_contain_string(new_s))
    return min(results, key=lambda x:len(x))

s_strings = {'hello', 'world', 'lol'}

def get_max_ind(ar):
    pos = argmax(ar)
    # print(ar)
    return (int(pos / ar.shape[1]), int(pos % ar.shape[1]))

class superstring4:
    def __init__(self, strings):
        self.n = len(strings)
        self.strings = strings
        # print (self.n)
        self.ov = zeros((self.n, self.n))
        for i in range(self.n):
            for j in range(self.n):
                if i != j:
                    self.ov[i][j] = strings_overlap(strings[i], strings[j])
    def greedy_assignment(self):
        rows = list(range(self.n))
        cols = list(range(self.n))
        asg = zeros(self.n)
        while len(rows) and len(cols):
            mi = get_max_ind(self.ov[ix_(array(rows), array(cols))])
            mi = (rows[mi[0]], cols[mi[1]])
            asg[mi[0]] = mi[1]
            # print(mi)
            if mi[0] in rows:
                rows.remove(mi[0])
            if mi[1] in cols:
                cols.remove(mi[1])
        return asg
    def get_cycle(self, asg, start, used):
        res = list()
        j = int(start)
        while True:
            used[j] = 1
            res.append(j)
            j = int(asg[j]) #avoid warning
            if j == start:
                break;
        return res
    def greedy_cover(self):
        used = zeros(self.n)
        asg = self.greedy_assignment()
        res = list()
        for i in range(self.n):
            if not used[i]:
                res.append(self.get_cycle(asg, i, used))
        return res
        # print(self.ov[ix_([rows, cols])])

    #works only if second not contains in first(in the middle of the string)
    def pref(self, s1, s2):
        assert(not s2 in s1)
        ol = max([i for i in range(min(len(s1), len(s2)) + 1) if s1[len(s1)-i:] == s2[:i]
])
        return s1[:len(s1) - ol]
    def merge(self, strings):
        assert(len(strings) > 0)
        res = ""
        last = strings[0]
        for s in strings[1:]:
            if (s in last):
                last = s
                continue
            res = res + self.pref(last, s)
            # print ('add pref of:', last, s, '=', self.pref(last, s))
            last = s
        res = res + strings[-1]
        return res
    def rotate_max_cycle(self, cycle):
        c = array(cycle)
        res = list()
        for i in range(len(cycle)):
            res.append(self.merge(roll(c, i)))
        # print (res)
        return min(res, key=lambda x: len(x))
    def solve(self):
        cover = self.greedy_cover()
        res = ""
        for c in cover:
            s = [self.strings[int(i)] for i in c]
            res = res + self.rotate_max_cycle(s)
        return res
```