--- Step D: Final fallback mapping for remaining unmapped atoms

```
remaining = [atom for atom in section if final[atom[0]] == ""]
count = len(remaining)
# ensure only C or O remain
if count in (5,4,3,2,1):
    if any(atom[1].upper() not in ('C','O') for atom in remaining):
        raise ValueError("non-benzene 6-ring section has non C/O
atom")
if count == 6:    # Fall back to the original oxygen-based mapping.
    array_O = [atom[0] for atom in section if atom[1].upper() == "O"
and final[atom[0]] == ""]
    if len(array_O) == 1:
        target_idx = array_O[0]
        target = next(a for a in section if a[0] == target_idx)
        a_str = generate_random_string()
        bead = "SN4a" + a_str
        final[target[0]] = bead
        for tup in target[4]:
            neighbor_idx = section[tup[0]][0]
            final[neighbor_idx] = bead
    elif len(array_O) == 2:
        for idx in array_O:
            a_str = generate_random_string()
            bead = "SN3a" + a_str
            final[idx] = bead
            atom_obj = next(a for a in section if a[0] == idx)
            for tup in atom_obj[4]:
                neighbor_idx = section[tup[0]][0]
                final[neighbor_idx] = bead
    array1 = [atom[0] for atom in section if final[atom[0]] == ""]
    if len(array1) == 3:
        a_str = generate_random_string()
        bead = "SC3" + a_str
        for idx in array1:
            final[idx] = bead
    elif len(array1) == 6:
        group1 = array1[:3]
        group2 = array1[3:]
        a_str1 = generate_random_string()
        bead1 = "SC3" + a_str1
        for idx in group1:
            final[idx] = bead1
        a_str2 = generate_random_string()
        bead2 = "SC3" + a_str2
        for idx in group2:
            final[idx] = bead2
    else:
        raise ValueError("Fallback mapping for non-benzene 6-ring with
count {} not implemented (TODO).".format(count))
elif count == 5:
    # find the C-atom with exactly one unmapped C neighbor
    cand = None
    for atom in remaining:
        nbrs = [section[t[0]] for t in atom[4]
                if final[ section[t[0]][0] ] == ""]
        if len(nbrs) == 1 and nbrs[0][1].upper() == 'C':
            cand = (atom, nbrs[0]); break
    if not cand:
        raise ValueError("Non-benzene 6-ring: no singleton C neighbor
for count=5")
    atom, nbr = cand
    old_bead = final[nbr[0]]
    # find all indices in final with old_bead
    targets = [i for i,v in enumerate(final) if v == old_bead]
    new_bead = old_bead
    if new_bead.startswith('T'):
        new_bead = 'S' + new_bead[1:]
    for i in targets:
        final[i] = new_bead
    final[atom[0]] = new_bead
    # now fall through to count=4 logic
    count = 4
    remaining = [atom for atom in section if final[atom[0]] == ""]
```

```
if count == 4:
    # for each atom with only one unmapped neighbor, build type string
    for atom in remaining:
        nbrs = [section[t[0]] for t in atom[4] if
final[ section[t[0]][0] ] == ""]
        if len(nbrs) == 1:
            types = atom[1].upper() + nbrs[0][1].upper()
            # find T-keys with val[0]==3 or 5 matching types or
reversed
            for key,val in martini_dict.items():
                if key.startswith('T') and val[0] in (3,5) and (val[2]
== types or val[2] == types[::-1]):
                    bead = key + generate_random_string()
                    final[atom[0]] = bead
                    final[nbrs[0][0]] = bead
                    break
elif count == 3:
    # center atom has two unmapped neighbors
    center = next(atom for atom in remaining
                  if len([t for t in atom[4] if
final[ section[t[0]][0] ] == ""]) == 2)
    nbrs = [section[t[0]] for t in center[4] if
final[ section[t[0]][0] ] == ""]
    s = nbrs[0][1].upper() + center[1].upper() + nbrs[1][1].upper()
    for key,val in martini_dict.items():
        if key.startswith('S') and val[0] in (3,5) and (val[2] == s or
val[2] == s[::-1]):
            bead = key + generate_random_string()
            final[center[0]] = bead
            for n in nbrs: final[n[0]] = bead
            break
    # if still unmapped, do the “merge” logic
    if final[center[0]] == "":
        merged = False

        # 1) look for a C-neighbor with a T-bead neighbor
        for nbr in nbrs:
            if nbr[1].upper() == 'C':
                # gather all neighbor indices (inner + outer)
                inner_idxs = [section[t[0]][0] for t in nbr[4]]
                outer_idxs = [full_mapping[t[0]][t[1]][0] for t in
nbr[3]]
                all_idxs = inner_idxs + outer_idxs

                t_beads = [final[i] for i in all_idxs if
final[i].startswith('T')]
                if t_beads:
                    old = t_beads[0]
                    new = 'S' + old[1:]
                    # reassign everywhere
                    for j, v in enumerate(final):
                        if v == old:
                            final[j] = new
                    final[nbr[0]] = new
                    merged = True
                    break

        # 2) if still not merged, fall back to any S-bead neighbor
        if not merged:
            for nbr in nbrs:
                inner_idxs = [section[t[0]][0] for t in nbr[4]]
                outer_idxs = [full_mapping[t[0]][t[1]][0] for t in
nbr[3]]
                all_idxs = inner_idxs + outer_idxs

                s_beads = [final[i] for i in all_idxs if
final[i].startswith('S')]
                if s_beads:
                    bead = s_beads[0]
                    final[nbr[0]] = bead
                    merged = True
                    break

        # 3) if we still failed, error out
        if not merged:
            raise ValueError(
                "Merging is not possible; mapping failed for count=3"
            )
        count = 2
        remaining = [atom for atom in section if final[atom[0]] == ""]
if count == 2:
    a1,a2 = remaining
    types = a1[1].upper() + a2[1].upper()
    for key,val in martini_dict.items():
        if key.startswith('T') and val[0] in (3,5) and (val[2] ==
types or val[2] == types[::-1]):
            bead = key + generate_random_string()
            final[a1[0]] = bead
            final[a2[0]] = bead
            break
elif count == 1:
    lone = remaining[0]
    if lone[1].upper() == 'O':
        raise ValueError("non benzene 6-ring has a lone Oxygen left")
    # else C
    # try find T-bead neighbor
    # check its mapped neighbors
    nbrs = [section[t[0]] for t in lone[4]]
    t_beads = [final[n[0]] for n in nbrs if
final[n[0]].startswith('T')]
    if t_beads:
        old = t_beads[0]
        new = old
        if not new.startswith('S'): new = 'S' + new[1:]
        for i,v in enumerate(final):
            if v == old: final[i] = new
        final[lone[0]] = new
    else:
        s_beads = [final[n[0]] for n in nbrs if
final[n[0]].startswith('S')]
        if s_beads:
            final[lone[0]] = s_beads[0]
        else:
            raise ValueError("non benzene 6-ring: cannot assign lone C
bead")
return final
```

I have found the culprit to every problem that I am having

The problem is that this section is assigning overlapping beads:

for example, I have count = 4 going into this step, and the formation is O-C-C-C, O-C is being mapped but the C that is already mapped with O is then being replaced and mapped with the next C and so on. We need to fix this issue for all count:

the issue is that we only check if the neighbor is already mapped and not itself, for every atom in remaining, we need to also check if that atom itself is already mapped, if it is we need to move on to the next

next is the fixing of lone oxygen atoms and assigning them with appropriate bead

I have printed out what the oxygen atom is usually next to and I have found out the following candidates: SC3, SN4a, SN6, TN4a, SC4, SN2a, SN1, SN4, SN6d, TN1 in order of amount of appearance, now it is to work on each case separately to see which case I can work with

so looking at this, I noticed that SN4a and SN4 for example is too similar so I must have a better way of finding out which bead it is

so first, I will strip the last 6 characters of the string of the final of each neighbor and then strip all of the '+' in the string and what is left is what we are working with

case 1: if all finals of all neighbors are empty then throw an error saying that lone atom is next to unmapped atoms

case 2: if one of the finals is 'SC3' after being stripped:

//so SC3 is composed of CCC in a straight line, so we need to figure out 2 things

1. whether this oxygen connection to the neighbor has 1 or 2 bonds
2. whether this oxygen is connecting to the middle atom of the SC3 bead (since SC3 is 3 beads)

we can figure out 1 by checking the connection between the oxygen atom and the neighbor that has final that is SC3

we can figure out 2 by checking all of the indices in final that is assigned the same as the neighbor atom (unstripped version) then we find out which section these atoms are from and find out the atom in the center (the atom that are connected to the other 2 atoms (inner or outer does not matter), as long as there is an atom that is connected to the other 2 atoms in the SC3, that atom is the central atom) then figure out if that atom is also connecting to the oxygen atom

so in total we have 4 cases:

1. 1 bond and connected to the center, in this case we will reassign all 4 atoms (the oxygen and the 3 in SC3) as SN6 + random string
2. 2 bond and connected to the center, reassign all as SN6a +random string
3. 1 bond and not center, reassign all as N6 + random string
4. 2 bond and not center, reassign all as N6a + random string

case 3: “SC4”

this case we need to figure out if

 the lone oxygen has 1 or 2 connection to the neighbor

so in total we have 2 cases:

1. 1 bond: reassign all four as N6 + random string
2. 2 bond: reassign all four as N6a + random string

case 4: “TN4a”

this case we need to figure out if

the lone oxygen has 1 connection and it is connecting to a ‘C’ Carbon that has the final which is TN4a. if this is the case:

reassign all as SN5a + random string (should be 2 atoms that is merged to this oxygen atom)

otherwise skip this case

case 5: “SN6”

similar to TN4a:

```
if bond order 1 and connect to C: reassign all as N5 +random

if bond order 2 and connect to C: reassign all as N5a + random

case 6: “SN4a”

if bond order 1 and connect to C: reassign all as N5a + random

if bond order 2 and connect to C reassign all as N4a + random


solution:

we have to build a cache that call all of the section of the neighbor
to do the actual checking

# ------------------------------------------------------------------
# Build cache: global index -> (section index, local index)
global_to_sec_loc: Dict[int, (int,int)] = {}
for sec_idx, sec in enumerate(full_mapping):
    for loc_idx, atom_info in enumerate(sec):
        gi = atom_info[0]
        global_to_sec_loc[gi] = (sec_idx, loc_idx)
# ------------------------------------------------------------------
```

Next fix:

```
           if len(f_tups) == 2:
                fa1 = full_mapping[f_tups[0][0]][f_tups[0][1]]
                fa2 = full_mapping[f_tups[1][0]][f_tups[1][1]]
                if atom[1].upper() == 'C' and fa1[1].upper() == 'C'
and fa2[1].upper() == 'C':
                    rstr = generate_random_string()
                    bead = "SC3" + rstr
                    final[atom[0]] = bead
                    final[fa1[0]] = bead
                    final[fa2[0]] = bead
                    continue
                else:
                    raise ValueError("2 size 1 neighbors but are not
all C")
```

this is not very comprehensive

we need to change all of this

so if len(f_tups) == 2, there should be more cases, actually we need to look for each outter atom:

1. what is the atom?
2. what is the outter atom

So now we have these cases:

1. the atom is C
    a. one of the outer atom is C:
        i. whatever the other atom is use the pick bead key with bond order 1 and kind ‘S’ to get the key and assign all 3 atoms accordingly
    b. else, we have these cases:
        i. O and O: SN5a
        ii. N and N: SN1

    otherwise throw 2 size 1 neighbor incompatible
2. the atom is O:
    a. we have only 1 case for the foreign atoms:
        i. C and C: SN3a
        ii. otherwise throw 2 size 1 neighbor incompatible

There are lots of other bug fixing that is not shown here

in total the algorithm is able to predict the martini beads for 3000+ different TPCN molecules and 41/128 Cancer molecules