Task 5: Mapping of DOX

problem: currently non-benzene 6-ring section has very little rules since we did not need any more rules and the mapping from the building block table required very little ruyles for the non-benzene 6-ring

time to add more rules:

we have not accounted for outer connection whatsoever, so that is what we will do here.

General rules: these are singular connections (mostly). The path we will take has these steps:

1. do the ring-ring border nodes first. How does this work:

   We will trace through all nodes in the section:

   we check to see if any node that has not been mapped (final = “”) that has a foreign connection to a group that its section != 0 (2nd index of atom)-> we have a ring-ring connection

we then check the inner neighbors of this node:

- here there are 2 possible cases:
- case 1: no inner neighbors of this node has a foreign neighbor of the same section as this node
    - this case we will throw an error saying: ring-ring connections through a tube is not yet supported with a todo to fix this part later:
- case 2: we found 1 inner neighbor that has a foreign neighbor of the same section as this node:
    - in this case we can break it down into 3 cases:
    - Case 1: one of these node have 2 foreign neighbors where the other foreign neighbor (not the together one) has section == 0 (non ring) and the section has size 1
        - In this case, we assign the two atoms and this foreign atom depending on the atom type of the foreign atoms, we can copy the benzene section for this:

```
bond_order = atom[3][0][2]
if foreign_atom[1].upper() == 'O':
  if bond_order == 2:
    bead = "SN6a" + rstr
  elif bond_order == 1:
```

- bead = "SN6" + rstr
- else:
- bead = ""
- elif foreign_atom[1].upper() == 'N':
- bead = "SN6d" + rstr
- elif foreign_atom[1].upper() == 'S':
- bead = "SC6" + rstr
- elif foreign_atom[1].upper() == 'CL':
- bead = "SX3" + rstr
- elif foreign_atom[1].upper() == 'I':
- bead = "X1" + rstr
- elif foreign_atom[1].upper() == 'C':
- bead = "SC4" + rstr
- elif foreign_atom[1].upper() == 'BR':
- bead = "SX2" + rstr
- else:
- bead = ""
- then assign all 3 atoms to this bead

- Case 2: two of these nodes have 2 foreign neighbors that where the other foreign neighbor (not the together one) has section == 0 (non ring) and the section has size 1
  - we throw an error saying the intermediate section is too complex for current pattern matching
  - add a todo to add rules to fix this section
- Case 3: Else:we just assign the two atoms we have as TC5
- Note that in Case 3 for this case, this is just saying that these 2 atoms we are working with has no foreign section that is a lone atom.

2. now we will do any double bonds sections

   We will trace through all nodes in the section:

   we check to see if any node that has not been mapped (final = “”) that has an inner connection to another node of bond amount 2 -> we have a double bonds connection

we then check the outer neighbors of this node:

- in this case we can break it down into 3 cases:

- Case 1: one of these node have a foreign neighbors with section == 0 (non ring) and the section has size 1
    - In this case, we assign the two atoms and this foreign atom depending on the atom type of the foreign atoms, we can copy the benzene section for this:

```
bond_order = atom[3][0][2]
if foreign_atom[1].upper() == 'O':
  if bond_order == 2:
    bead = "SN6a" + rstr
  elif bond_order == 1:
    bead = "SN6" + rstr
  else:
    bead = ""
elif foreign_atom[1].upper() == 'N':
  bead = "SN6d" + rstr
elif foreign_atom[1].upper() == 'S':
  bead = "SC6" + rstr
elif foreign_atom[1].upper() == 'CL':
  bead = "SX3" + rstr
elif foreign_atom[1].upper() == 'I':
  bead = "X1" + rstr
elif foreign_atom[1].upper() == 'C':
  bead = "SC4" + rstr
elif foreign_atom[1].upper() == 'BR':
  bead = "SX2" + rstr
else:
  bead = ""
```

    - then assign all 3 atoms to this bead
- Case 2: both of these node have a foreign neighbor with section == 0 (non ring) and the section has size 1:
    - in this case, for each ring node (original node), we will map it with its foreign node with the following rules: (copy from benzene section)

```
bond_order = atom[3][0][2]
if foreign_atom[1].upper() == 'O':
  if bond_order == 2:
    bead = "TN6a" + rstr
  elif bond_order == 1:
```

- bead = "TN6" + rstr
- else:
- bead = ""
- elif foreign_atom[1].upper() == 'C':
- bead = "TC4" + rstr
- elif foreign_atom[1].upper() == 'BR':
- bead = "TX2" + rstr
- elif foreign_atom[1].upper() == 'N':
- bead = "TN6d" + rstr
- elif foreign_atom[1].upper() == 'S':
- bead = "TC6" + rstr
- elif foreign_atom[1].upper() == 'CL':
- bead = "TX3" + rstr
- elif foreign_atom[1].upper() == 'I':
- bead = "TX1" + rstr
- final[atom[0]] = bead
- final[foreign_atom[0]] = bead
- So we should have 2 beads each representing 2 atoms

- Case 3: Else:we just assign the two atoms we have as TC5
- Note that in Case 3 for this case, this is just saying that these 2 atoms we are working with has no foreign section that is a lone atom.

3. Now we will work with any other atoms with a foreign neighbor with size 1 and is non-ring that has not been mapped :

   We will trace through all nodes in the section:

   we check to see if any node that has not been mapped (final = “”) that has a foreign connection to a group that its section == 0 (2nd index of atom) and the foreign section size = 1-> we have a inner-outer connection that still needs to be mapped

   a. now that that is satisfied, we need to check this node’s inner connections (index 4 of atom) . do array1 = []. For each inner connection we will check:
      1. to see if the inner neighbor node has final[index 0]= “” if it is not then we continues to the next inner neighbor node: if it is we checks
      2. we will check if both of the inner neighbor nodes’ inner neighbor nodes has final[index 0] = “” (this is quite convoluted and 2 steps ahead) if this is true then we continues. If it is not then we checks

3. we will check if this inner neighbor node has an foreign node of section == 0 and section size = 1 if it is we continues, if it is not then put this inner neighbor atom to an array1

b. Now that is done, if this array 1 have size 0, we will go back to the original node and map it with the foreign node as a pair following benzene rules:

```
bond_order = atom[3][0][2]
if foreign_atom[1].upper() == 'O':
    if bond_order == 2:
        bead = "TN6a" + rstr
    elif bond_order == 1:
        bead = "TN6" + rstr
    else:
        bead = ""
elif foreign_atom[1].upper() == 'C':
    bead = "TC4" + rstr
elif foreign_atom[1].upper() == 'BR':
    bead = "TX2" + rstr
elif foreign_atom[1].upper() == 'N':
    bead = "TN6d" + rstr
elif foreign_atom[1].upper() == 'S':
    bead = "TC6" + rstr
elif foreign_atom[1].upper() == 'CL':
    bead = "TX3" + rstr
elif foreign_atom[1].upper() == 'I':
    bead = "TX1" + rstr
final[atom[0]] = bead
final[foreign_atom[0]] = bead
```

c. if this array1 have size 1, we map the original node with the node in this array and the foreign nodes as 1 bead following the benzene rules for 3 atom beads:

```
bond_order = atom[3][0][2]
if foreign_atom[1].upper() == 'O':
    if bond_order == 2:
        bead = "SN6a" + rstr
    elif bond_order == 1:
        bead = "SN6" + rstr
    else:
        bead = ""
elif foreign_atom[1].upper() == 'N':
```

i. bead = "SN6d" + rstr
j. elif foreign_atom[1].upper() == 'S':
k. bead = "SC6" + rstr
l. elif foreign_atom[1].upper() == 'CL':
m. bead = "SX3" + rstr
n. elif foreign_atom[1].upper() == 'I':
o. bead = "X1" + rstr
p. elif foreign_atom[1].upper() == 'C':
q. bead = "SC4" + rstr
r. elif foreign_atom[1].upper() == 'BR':
s. bead = "SX2" + rstr
t. else:
u. bead = ""
v. then assign all 3 atoms to this bead

d. if this array 2 have size 2, we will throw an error saying this non-ring section is too complex to be mapped with a todo to fix this later

4. now that every single foreign section and double bonds had been accounted for, we can trace through this section 1 last times, we first counts the numbers of atoms that had not been mapped. and if the count is 6, we implement what we had before

def map_nonbenzene_6_ring_section(section: List[List[Any]], final: List[str], martini_dict: Dict[str, List[Any]]) -> List[str]:

""" Map a non-benzene ring section that contains 6 atoms.

```
Pseudocode:
  - Create an array (array_O) by looping through the section:
      For every atom that has:
         * element (index 1) == "O" (case-insensitive)
         * final[atom[0]] == ""
      add the atom's global index (atom[0]) to array_O.

  - If array_O has length 1:
      * Generate a random string (a).
      * Let target be the atom corresponding to array_O[0].
      * Set final[target[0]] = "SN4a" + a.
      * For the first two inner neighbors of target (if they exist), assign
```

```
        final[section[tup[0]][0]] = "SN4a" + a.

  - If array_O has length 2:
      * For each atom index in array_O:
          - Generate a random string.
          - Set final[atom_index] = "SN3a" + a.
          - For the first two inner neighbors of that atom (if they
exist),
            assign final[section[tup[0]][0]] = "SN3a" + a.

  - Build array1:
      * Loop through the section and add the global index of every
atom that is still unmapped (final[atom[0]] == "").

  - If array1 has length 3:
      * Generate a random string.
      * For every atom in array1, assign final[atom[0]] = "SC3" + a,
        and also assign the same bead name to its first two inner
neighbors (if they exist).

  - If array1 has length 6:
      * Break array1 into two groups of 3 (assumed connected).
      * For each group, generate a random string and assign the same
bead (e.g. "SC3" + a)
        to all atoms in that group.

  - Finally, check that every atom in the section is mapped; if not,
raise an error.
"""
# Step 1: Build array_O for atoms with element "O" that are unmapped.
array_O = []
for atom in section:
    if atom[1].upper() == "O" and final[atom[0]] == "":
        array_O.append(atom[0])

# Step 2: If array_O has length 1, assign bead "SN4a" + random string.
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 the first two inner neighbors:
    for tup in target[4]:
        neighbor_idx = section[tup[0]][0]
        final[neighbor_idx] = bead

# Step 3: If array_O has length 2, assign bead "SN3a" + random string
for each.
elif len(array_O) == 2:
    for idx in array_O:
        a_str = generate_random_string()
        bead = "SN3a" + a_str
        final[idx] = bead
        atom = next(a for a in section if a[0] == idx)
        for tup in atom[4]:
            neighbor_idx = section[tup[0]][0]
            final[neighbor_idx] = bead

# Step 4: Build array1 of any remaining unmapped atoms.
array1 = [atom[0] for atom in section if final[atom[0]] == ""]

# Step 5: Depending on the size of array1, assign beads.
if len(array1) == 3:
    a_str = generate_random_string()
    bead = "SC3" + a_str
    for idx in array1:
        final[idx] = bead

elif len(array1) == 6:
    # Break array1 into two groups of 3.
    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

# Step 6: Final check that every atom is mapped.
for atom in section:
    if final[atom[0]] == "":
        raise ValueError("Non-benzene 6-ring section not fully
mappable!")
return final
```

If the count is 5:

if the count is 4:

if the count is 3:

if the count is 2:

if the count is 1:

for all of these cases throws an error saying this is soon to be implemented for non-benzene 6 ring with a todo to add more rules to this part later.