1. Fix

```
coord_lookup = {
  c['index']: (c['x'] / 10.0, c['y'] / 10.0, c['z'] / 10.0)
  for c in atom_coords
}
```

This means that the code is now correctly identify the distance as nm

2. The ITP files need the following fix:
a. resid: make it all res
b. atom: make it all C1, C2, ...
3. The Gro files need the following fix:
a. have the first column be 1res for all

4. More fix the the ITP file: Add to the bead connection 2 more column:
a. b0 value: distance between 2 beads based on the distance formula
b. k value: leave it as 20000 for all column

Code fixing:

Fix an issue in 6-ring part where C=N is assigned as TC5

add another section between section C and D that does the following:

for every unmapped inner atom that is not C or O, check to see if it has a non-ring connection that is of length 1 and the atom is C, if it is, use the pick key where the element is the inner element and bond order is 1 and kind is T then add it to both of the atoms
This is to account for some of the inner atoms that is not C or O

Fix an issue with 5-bead mapping where it would throw an error every time.

Add more bracket handling for case of [2H]

```
# Bracket handling: “[ ... ]”
if ch == '[':
  # Find the matching ']'
```

```
    j = smiles.find(']', i + 1)
    if j == -1:
        raise ValueError(f"Unmatched '[' at position {i}")

    # Extract the content inside brackets
    inside = smiles[i + 1 : j]
    # Strip any non-letter character, keep only A–Z or a–z
    letters_only = "".join(c for c in inside if c.isalpha())
    # Emit each letter as its own token
    for letter in letters_only:
        tokens.append(letter)
    # Advance i to character after ']'
    i = j + 1
    continue
```

Fix to the dictionary:

notice how I have a lot of duplicate here, it overlaps one another I can list all of the key-value pair first, so that all of the key and value pairs are saved even if the keys are the same, then loop through that array one by one to add the key-value pairs to the dictionary, but whenever a key overlap, you add a + and check if it still overlap until all keys are unique and the dictionary will be unique

THIS SECTION IS LISTING NEW BEADS:

```
    ("P6?",  [11, 2, "S(C)(=O)(=O)(C)"  , 0, ""]),
    ("C2?",  [11, 2, "Ge(C)(C)(C)(C)"   , 0, ""]),
    ("C2?",  [11, 2, "Pb(C)(C)(C)(C)"   , 0, ""]),
    ("C2?",  [11, 2, "Si(C)(C)(C)(C)"   , 0, ""]),
```

("C2?", [11, 2, "Sn(C)(C)(C)(C)" , 0, ""]),

("P2?", [11, 1, "C(O)(=O)(CO)", 0, ""]),

("SC6?", [11, 1, "S(=O)(=O)(C)", 0, ""]),

("P2?", [11, 1, "C(O)(=O)(CN)", 0, ""]),

("SX4e?", [11, 1, "C(Cl)(F)(F)", 1, ""]),

("X4e?", [11, 1, "C(OC)(F)(F)", 1, ""]),

("SX4e?", [11, 1, "C(O)(F)(F)", 1, ""]),

("X4e?", [11, 1, "C(=C)(F)(F)", 1, ""]),

("X4e?", [11, 1, "C(Cl)(CF)(F)", 1, ""]),

("X4e?", [11, 1, "C(=O)(CF)(N)", 1, ""]),

("X2?", [11, 1, "C(F)(Cl)(Cl)", 0, ""]),

("X2?", [11, 1, "C(CCl)(Cl)(Cl)", 0, ""]),

("X2?", [11, 1, "C(=CCl)(Cl)(Cl)",0, ""]),

("SX2?", [11, 1, "C(C)(Cl)(Cl)", 0, ""]),

("SX2?", [11, 1, "C(C)(Cl)(CCl)", 0, ""]),

("SX2?", [11, 1, "C(=C)(Cl)(Cl)", 0, ""]),

("X1?", [11, 1, "C(Br)(Cl)(Br)", 0, ""]),

("X1?", [11, 1, "C(Br)(Br)(Br)", 0, ""]),

("X1?", [11, 1, "C(=CBr)(Br)(Br)",0, ""]),

("X2?", [11, 1, "C(Cl)(Cl)(Br)", 0, ""]),

("SP2?", [11, 1, "C(C)(=O)(=O)", 0, ""]),

("N5a?", [11, 1, "C(C)(C=O)(=C)", 0, ""]),

("N6?", [11, 1, "C(=C)(C)(CO)", 0, ""]),

("P4?", [11, 1, "C(O)(C)(CO)", 0, ""]),

("SN4?", [11, 1, "N(O)(C)(C)", 0, ""]),

("SP4?", [11, 1, "C(O)(=O)(=O)", 0, ""]),

("P6?", [11, 1, "C(CN)(=O)(N)", 0, ""]),

("P6?", [11, 1, "C(OC)(=O)(N)", 0, ""]),

("P6?", [11, 1, "C(NC)(=O)(N)", 0, ""]),

("SP6?", [11, 1, "C(N)(=O)(N)", 0, ""]),

("P6?", [11, 1, "C(NO)(=O)(N)", 0, ""]),

("P6?", [11, 1, "C(NO)(=O)(C)", 0, ""]),

("P5?", [11, 1, "C(NN)(=O)(C)", 0, ""]),

("SP6?", [11, 1, "C(N)(=S)(N)", 0, ""]),

("P6?", [11, 1, "C(N)(=O)(NN)", 0, ""]),

("SP5?", [11, 1, "C(N)(=N)(N)", 0, ""]),

("X3?", [11, 1, "C(C)(C)(CCl)", 0, ""]),

("X3?", [11, 1, "C(O)(C)(CCl)", 0, ""]),

("SX3?", [11, 1, "C(C)(=C)(Cl)", 0, ""]),

("X3?", [11, 1, "C(C)(CC)(Cl)", 0, ""]),

("X2?", [11, 1, "C(C)(C)(CBr)", 0, ""]),

("SC3?", [11, 1, "Pb(C)(C)(C)", 0, ""]),

("TP4?", [11, 0, "OO", 0, ""]),

("TN3a?", [11, 0, "NO", 0, ""]),

("TN3a?", [11, 0, "N=O", 0, ""]),

("TN1a?", [11, 0, "NN", 0, ""]),

("SN1a?", [11, 0, "CNN", 0, ""]),

("SN2?", [11, 0, "NC=S", 0, ""]),

("SN2?", [11, 0, "N=C=S", 0, ""]),

("SN2?", [11, 0, "NC=N", 0, ""]),

("SP2?", [11, 0, "O=C=O", 0, ""]),

("SP1?", [11, 0, "C=C=O", 0, ""]),

("SP6?", [11, 0, "O=S=O",   0, ""]),

("TC6?", [11, 0, "C=S",   0, ""]),

("SC6?", [11, 0, "CS=O",   0, ""]),

("SC6?", [11, 0, "S=C=S",   0, ""]),

("SN2?", [11, 0, "S=PC",   0, ""]),

("SP4?", [11, 0, "OCO",   0, ""]),

("SN2a?", [11, 0, "C=CO",   0, ""]),

("SX2?", [11, 0, "C=CBr",   2, ""]),

("SX2?", [11, 0, "BrCBr",   0, ""]),

("SX1?", [11, 0, "ICl",   0, ""]),

("SX3h?", [11, 0, "ClCBr",   0, ""]),

("SX3h?", [11, 0, "BrCCl",   0, ""]),

("SX3h?", [11, 0, "ClCF",   0, ""]),

("SX3h?", [11, 0, "ClHg",   0, ""]),

("SX3h?", [11, 0, "HgCl",   0, ""]),

("SX3h?", [11, 0, "ClN",   0, ""]),

("SX3h?", [11, 0, "NCl",   0, ""]),

("SX3h?", [11, 0, "FCCl",   0, ""]),

("SX3?", [11, 0, "C=CCl",   0, ""]),

("SX3?", [11, 0, "ClC=C",   0, ""]),

("SC3?", [11, 0, "CPbC",   0, ""]),

("SC3?", [11, 0, "CSiC",   0, ""]),

("X3h", [7, 0, "ClCCCl",   0, ""]),

("SX3h",[7, 0, "ClCCl",   0, ""]),

("TX3h",[7, 0, "CCl",   0, ""]),

("TX3h",[7, 0, "ClC",   0, ""]),

```
("SC2", [7, 0, "CCC",        1, ""]),
("SX4e",[7, 0, "FCF",        1, ""]),
("SC6", [7, 0, "CCS",        2, ""]),
("SX3", [7, 0, "CCCl",       2, ""]),
("SX3", [7, 0, "ClCC",       2, ""]),
("SX4e",[7, 0, "CCF",        2, ""]),
("X1",  [7, 0, "CCI",        2, ""]),
("SX2", [7, 0, "CCBr",       2, ""]),
("SX2", [7, 0, "BrCC",       2, ""]),
("TC6", [7, 0, "CS",         2, ""]),
("TX1", [7, 0, "CI",         2, ""]),
("TX2", [7, 0, "CBr",        2, ""]),
("TX2", [7, 0, "BrC",        2, ""]),
("TX4e",[7, 0, "CF",         2, ""]),
("X1",  [7, 2, "C(Br)(Br)(Br)(Br)", 0, "."]),
("X2",  [7, 2, "C(Cl)(Cl)(Cl)(Cl)", 0, "."]),
("X3",  [7, 2, "C(F)(F)(F)(F)"    , 0, "."]),
("C2",  [7, 2, "C(C)(C)(C)(C)"    , 0, ""]),
("X3",  [7, 2, "C(C)(C)(C)(Cl)"   , 0, ""]),
("X2",  [7, 2, "C(C)(C)(C)(Br)"   , 0, ""]),
("X1",  [7, 2, "C(C)(C)(C)(I)"    , 0, ""]),
```

This current part is very limited in what it can do so let's fix it:

```
# CASE 3: 4 edge atoms.
elif num_edges == 4:
   # Find the center atom that has 4 inner connections.
```

```
    center_local = None
    for i, atom in enumerate(section):
        if len(atom[4]) == 4:
            center_local = i
            break
    if center_local is None:
        raise ValueError("Non-ring section with 4 edges does not have
an atom with 4 inner connections.")

    # Build array4: for each inner connection of the center atom,
trace to the final edge atom.
    #  start by adding the central atom's global index.
    array4 = [section[center_local][0]]
    for (nbr, _) in section[center_local][4]:
        final_edge_local = get_final_edge(section, center_local, nbr)
        # Only add if the final atom is an edge and its type is 'F'
        if final_edge_local is not None and
section[final_edge_local][5] and section[final_edge_local][1] == 'F':
            array4.append(section[final_edge_local][0])

    # Ensure exactly 4 branch edge atoms were found.
    if len(array4) != 4:
        raise ValueError("Non-ring section cannot be mapped: expected
4 branch edge atoms, found {}.".format(len(array4)))

    # Assign the bead "SX4e" with a random tag to all atoms in array4.
    rstr = generate_random_string()
    for idx in array4:
        final[idx] = "SX4e" + rstr

    # Build array5: the list of atoms still unmapped.
    array5 = [atom[0] for atom in section if final[atom[0]] == ""]

    # If exactly 2 atoms remain and their types are C and O, assign
bead "TP1d" with a random tag.
    if len(array5) == 2:
        types = [next(a for a in section if a[0] == idx)[1].upper()
for idx in array5]
        if sorted(types) == ['C', 'O']:
```

```
            rstr2 = generate_random_string()
            for idx in array5:
                final[idx] = "TP1d" + rstr2
            return final
        else:
            raise ValueError("Non-ring section cannot be mapped:
remaining atoms are not C and O (found: {}).".format(types))
    else:
        raise ValueError("Non-ring section cannot be mapped:
unexpected number of unmapped atoms in 4-edge mapping (found {} in
array5).".format(len(array5)))
```

currently it only accounts for 1 specific case that is C(CO)(F)(F)(F), which is not what we want, so let's break it down into 2 parts:

part 1: if the amount of atoms is 6: we will do the following:

find the center atom (the only atom that has 4 inner connections)

there should now be 4 path, and 1 will have 2 atoms and the rest will have 1 atoms

now we should break this section into 2 parts,

the first part is the path with the 2 atoms,

we will change both of the edge of the two atoms to true

remove the connection between the atom that is connected to the center and the center atom

now make those two atoms its separate section and update mapping

run this function again on this section

the rest are the center atom connected to the other 3 singular path

make this part its own section and disconnect the center connection with the other path

now update mapping and run the function again on this section

part 2: if the amount of atoms is 5: we will do the following:

find the center atom (the only atom that has 4 inner connections)

there should now be 4 path, and all 4 path will have only 1 atoms

we will do the same as if there are 3 edges, by put the center node and the each branches into a bracket and then look into the dictionary to find the bead then throw an error stating the center and the branches if it cannot find it

```
# CASE 2: 3 edge atoms.
elif num_edges == 3:
    # Find the atom with 3 inner connections.
   center_local = None
    for i, atom in enumerate(section):
        if len(atom[4]) == 3:
            center_local = i
            break
    if center_local is None:
        raise ValueError("Non-ring section with 3 edges does not have
an atom with 3 inner connections.")
    # Trace each branch from the center atom.
    branch_paths = []
    for (nbr, _) in section[center_local][4]:
        branch_str = trace_branch(section, center_local, nbr)
        branch_paths.append(branch_str)

    # 1) Try section 7 matching
    import re
    candidate_keys = []
    for key, val in martini_dict.items():
        if val[0] == 7 and val[1] == 1 and "(" in val[2]:
            prefix, remainder = val[2].split("(", 1)
            if prefix.strip() != section[center_local][1]:
                continue
            branch_segments = re.findall(r'\(([^)]+)\)', val[2])
            if len(branch_segments) != len(branch_paths):
                continue
            # order-independent match
            unmatched = branch_segments.copy()
            matched_all = True
            for bp in branch_paths:
                for bs in list(unmatched):
```

```
                    if bp == bs:
                        unmatched.remove(bs)
                        break
                else:
                    matched_all = False
                    break
            if matched_all:
                candidate_keys.append(key)
    candidate_keys = list(set(candidate_keys))

    # 2) Fallback to section 11 if needed
    if not candidate_keys:
        for key, val in martini_dict.items():
            if val[0] == 11 and val[1] == 1 and "(" in val[2]:
                prefix, remainder = val[2].split("(", 1)
                if prefix.strip() != section[center_local][1]:
                    continue
                branch_segments = re.findall(r'\(([^)]+)\)', val[2])
                if len(branch_segments) != len(branch_paths):
                    continue
                unmatched = branch_segments.copy()
                matched_all = True
                for bp in branch_paths:
                    for bs in list(unmatched):
                        if bp == bs:
                            unmatched.remove(bs)
                            break
                    else:
                        matched_all = False
                        break
                if matched_all:
                    candidate_keys.append(key)
        if candidate_keys:
            warnings.warn(
                f"Warning: falling back to section 11 for 3-edge
mapping (center: {section[center_local][1]} and branches:
{branch_paths})",
                UserWarning
            )
```

```
        candidate_keys = list(set(candidate_keys))

    # 3) Commit or error
    if len(candidate_keys) == 1:
        rstr = generate_random_string()
        for atom in section:
            final[atom[0]] = candidate_keys[0] + rstr
        return final
    elif not candidate_keys:
        raise ValueError(
            f"Non-ring section cannot be mapped: no candidate bead
found for 3-edge mapping (center: {section[center_local][1]} and
branches: {branch_paths})."
        )
    else:
        raise ValueError(
            f"Non-ring section cannot be mapped: ambiguous candidate
keys for 3-edge mapping (center: {section[center_local][1]} and
branches: {branch_paths})."
        )
```

Here is the example for 3 edge bead mapping,

however in this case for 4 edge beads, to make the distinction, we will check in the section where val[1] = 2 instead, the rest is the same

then the error throw is for 4-edge mapping (center: ... identical)

After everything, we should have the correct bead mapping of every 4 edge 1-bead mappable beads, sometimes the mapping will have 1 more sections due to split

more fixing:

For the part that has 6 total atoms:

fix this part only. we only change the edge status for the atom that is connected to the center atom on the branch path = 2 side

so only that one specific atom we will change the edge status

Next is the connection:

we will only remove the connection between the center atom and that one atom that we just change the edge status

the other way around, remove the connection to the center atom of that new edge atom

we need to change mapping so that it will correctly display the two section as 2 different sections

we can run the rest accordingly

NOW THAT IT WORKS, TIME TO ADD MANY MORE BEADs, these are 1 center and 4 branches.